COURSE OUTLINE

PART I — CONCEPTS

1. Objectives
   a. Course Objectives
   b. Course Structure
   c. NOS/VE Objectives
2. NOS/VE Structures
   a. Packaging
   b. Table Segments
   c. Components
   d. Memory Layout
3. Job Flow
   a. Initiation
   b. Command Processing
   c. Termination
4. File Flow
   a. Open
   b. Transfer
   c. Close

PART II — COMMUNICATION

5. Resources
   a. Documentation
   b. System Initialization
   c. Load Map
6. Internal Communication
   a. Call/Return
   b. Interrupts
   c. Monitor
   d. Traps
7. External Communication
   a. Dual State
   b. Logs and Statistics
   c. Message Generator
   d. Keypoint

PART III — JOB/PROGRAM MANAGEMENT

8. Job Control
   a. Queued File Management
   b. Job Initiation
   c. Job Termination
9. Program Execution
   a. Task Management
   b. Loader
   c. Condition Handling
10. SCL Interpreter
    a. Control
    b. Command Processors

COURSE STRUCTURE

PART I — CONCEPTS
1. Objectives
2. NOS/VE Structure
3. Job Flow
4. File Flow

PART II — COMMUNICATION
5. Resources
6. Internal Communication
7. External Communication

PART III — JOB/PROGRAM MANAGEMENT
8. Job Control
9. Program Execution
10. SCL Interpreter

PART IV — FILES
11. Permanent File
12. Logical I/O
13. Physical I/O

NOS/VE OBJECTIVES

OBJECTIVES	STRATEGIES
RAM	HARDWARE (migds
CONFIGURABILITY	SASD structured analysis structured design
EXPANDABILITY	CYBIL
USABILITY	STANDARDS
CONSISTENCY	COMMAND INTERFACE
EFFICIENCY	PROGRAM INTERFACE
SECURITY	DUAL STATE
MIGRATION EASE	CP OPERATING SYSTEM (pp-s kwijt raken)
	CODE ISOLATION
	SYSTEM USING ITSELF

300 regels/maand
5% fouten

↳ IO queues
rgn subcatalogus
pf system

Multics in PL1

Book: SIS

PHASED RELEASES

R1 Basic Operating System
Disk and Tape Drivers
FORTRAN and COBOL
Dual-State
Conversion Aids

Online documentation
FMH

R2 Stand-Alone System
Unit Record Drivers
Interactive Facility
Products and Utilities — APL, LISP, Basic, UNIX

R3 Competitive System
Networks
Applications — nastran, unistruct etc
Etc.

HARDWARE CONSIDERATIONS

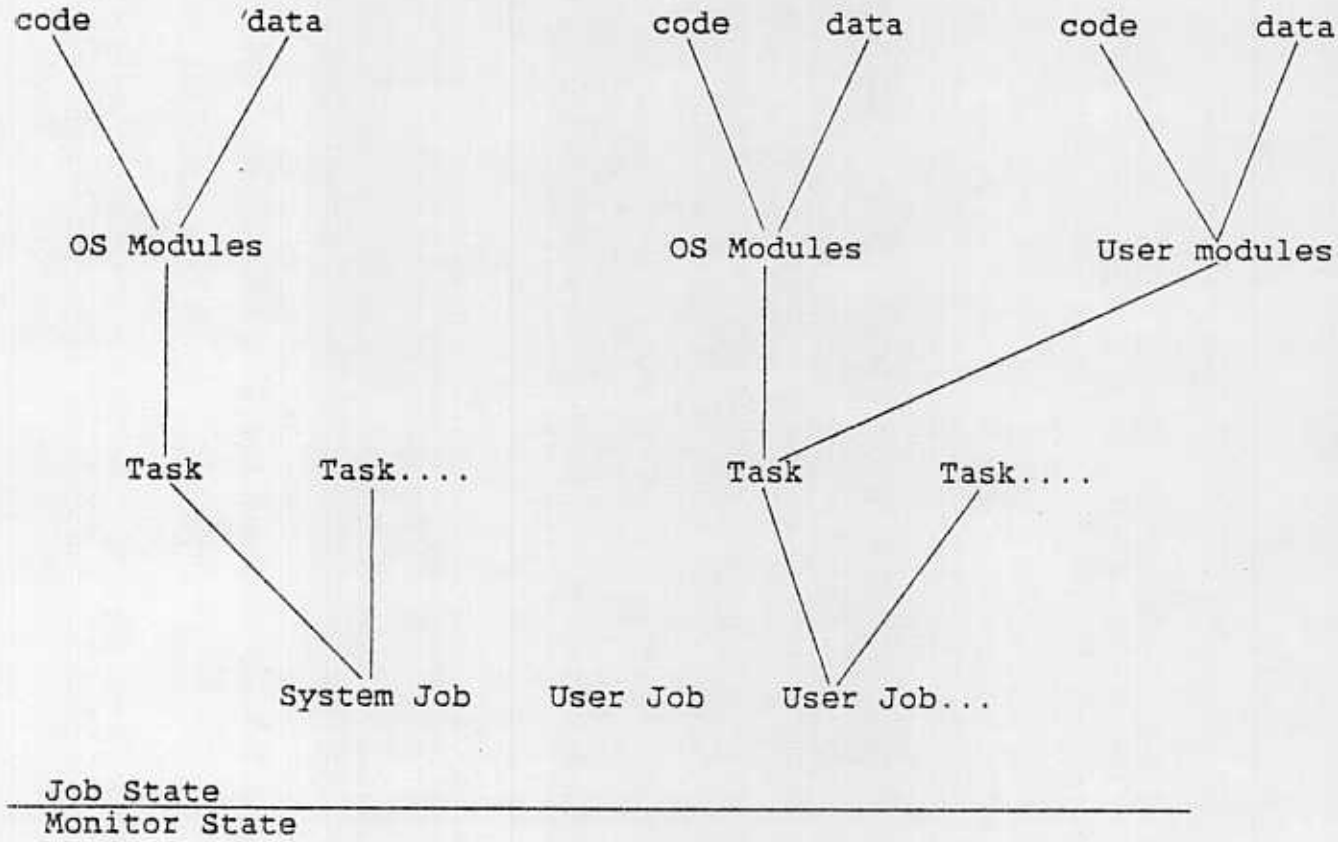
monitor instr privilege.

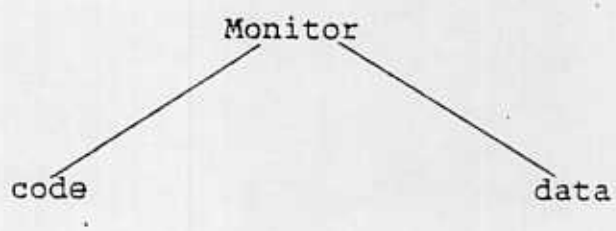
geen exchange interrupt

- JOB STATE vs MONITOR STATE
  - Instruction Privilege
  - Interrupts
- VIRTUAL ADDRESS SPACE 40gb * 2 miljard bits
  - Large
  - Segmented
  - Protected
- COMMUNICATION
  - Call/Return
  - Exchange
  - Traps

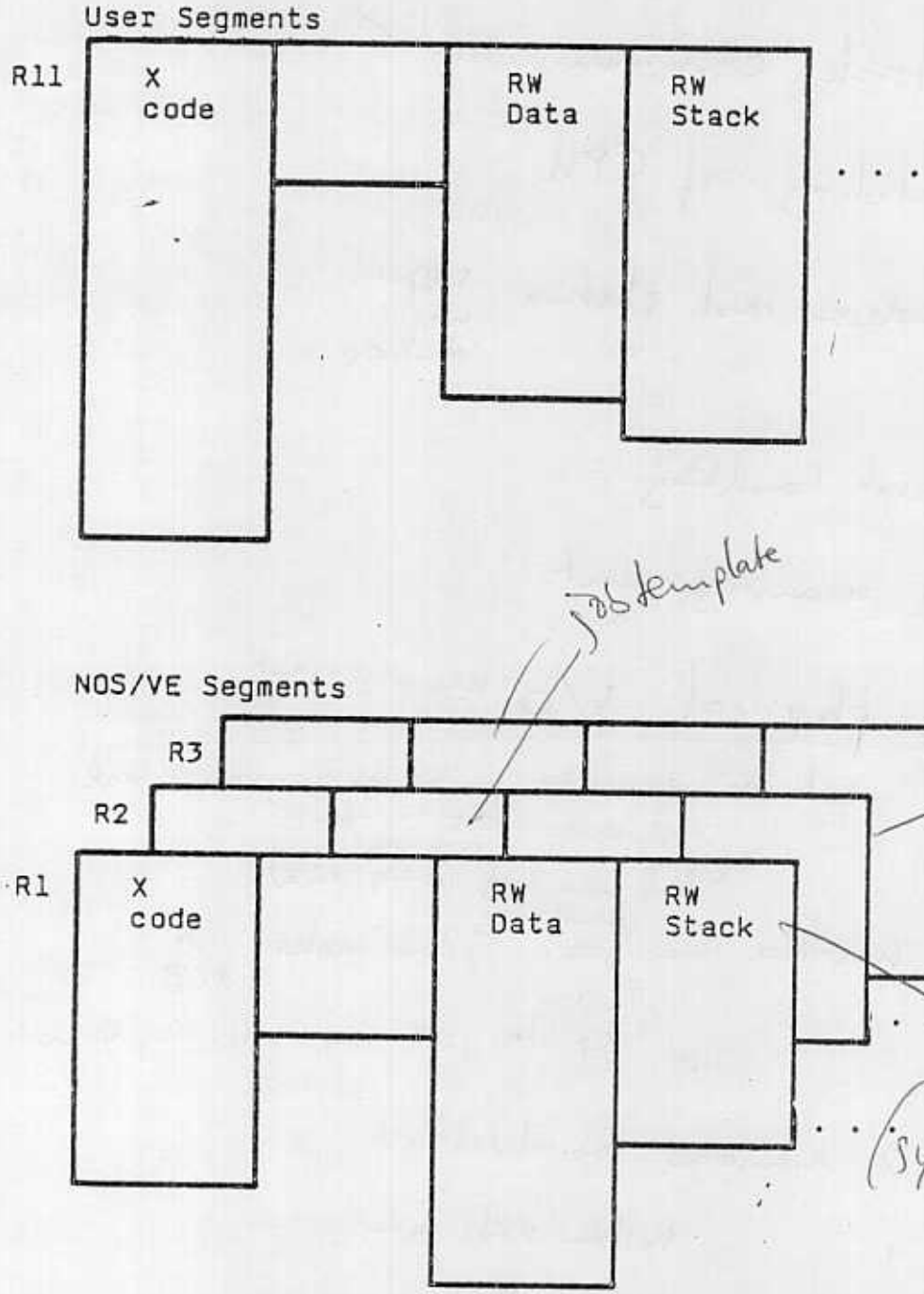
CP4 mtr = monolitisch programma
non pageable (wired pages)
RN = 1 (executie)
niet groot?

OS HIERARCHY

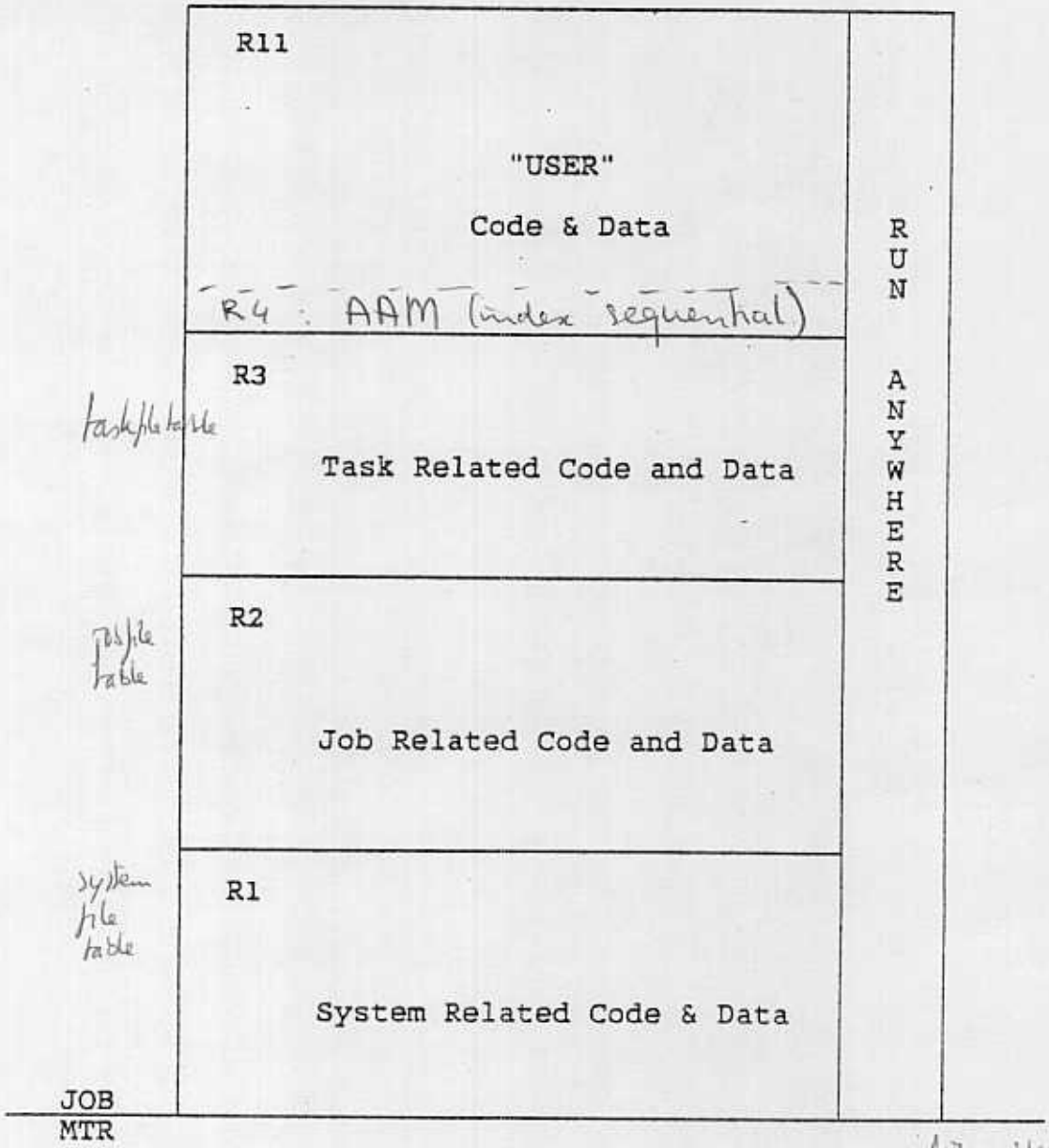




USER ADDRESS SPACE



PACKAGING



taskfile table

jobfile table

system file table

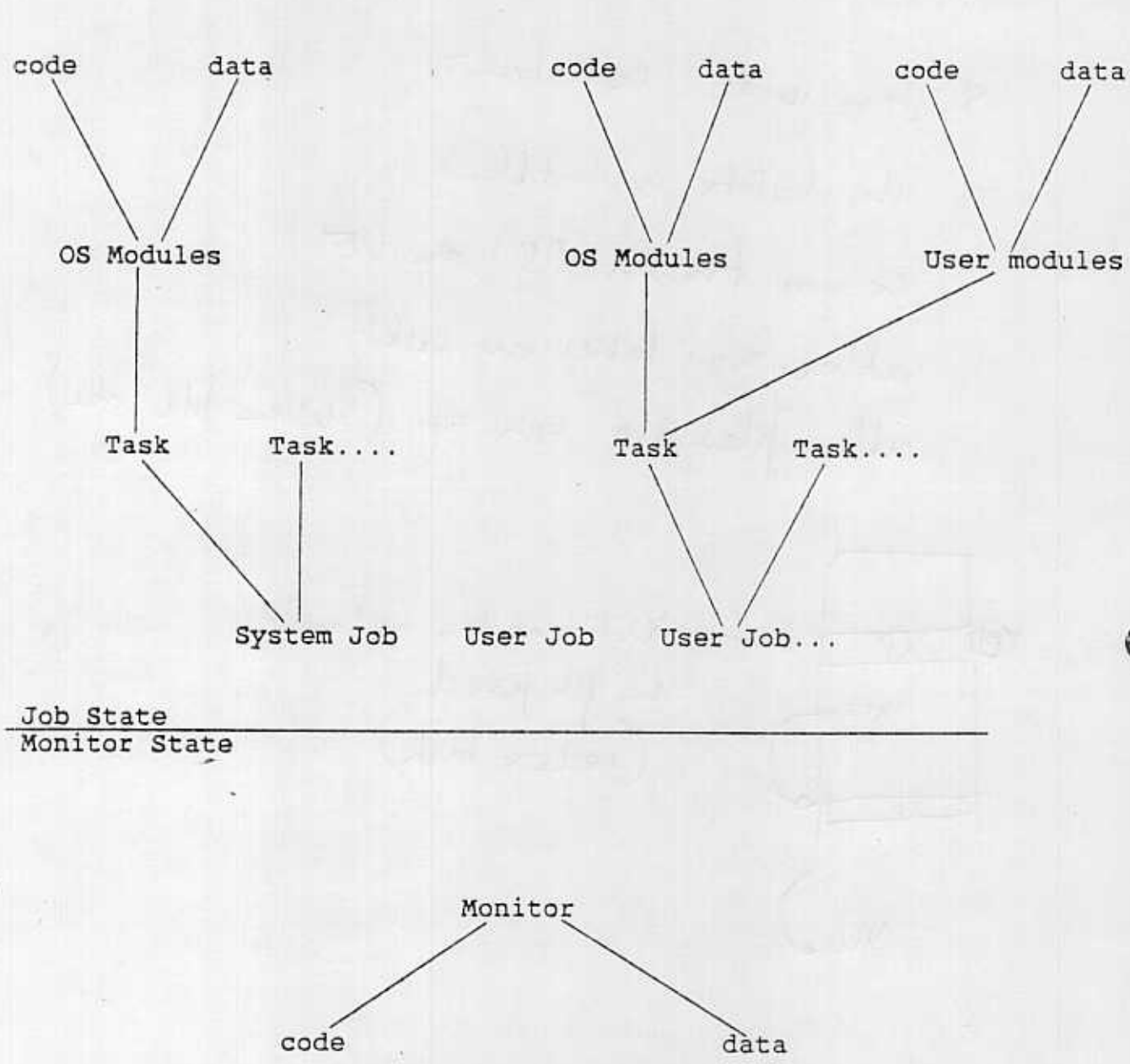
operating system tasks (op signals)
op reschilleder CPU PP
maintenance routines

NOVE = interrupt driven
NOS = table "
parameters via X registers

(DC → geen verandering van OS-core)
hooks bestaan voor CyBil procs

Control Data Private

binary releases


SHARED DATA
code
data
OS Modules
Task
Task....
System Job
code
data
code
data
OS Modules
User modules
Task
Task....
User Job
User Job...
Job State
Monitor State
Monitor
code
data

Ring 14 en 15
= executie zonder
toegang tot programmatuur

TABLE RESIDENCE

schrijf Ring!

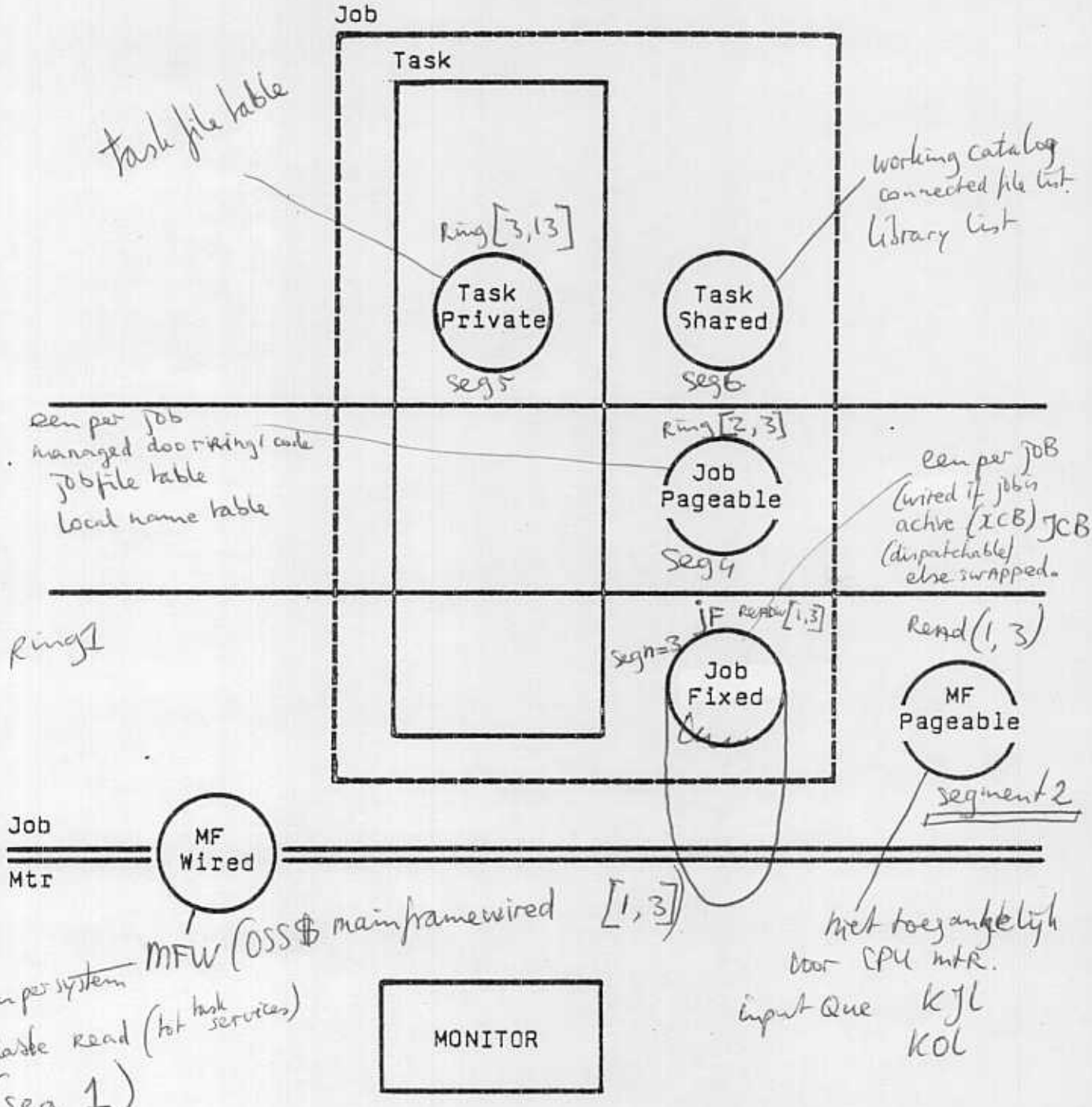
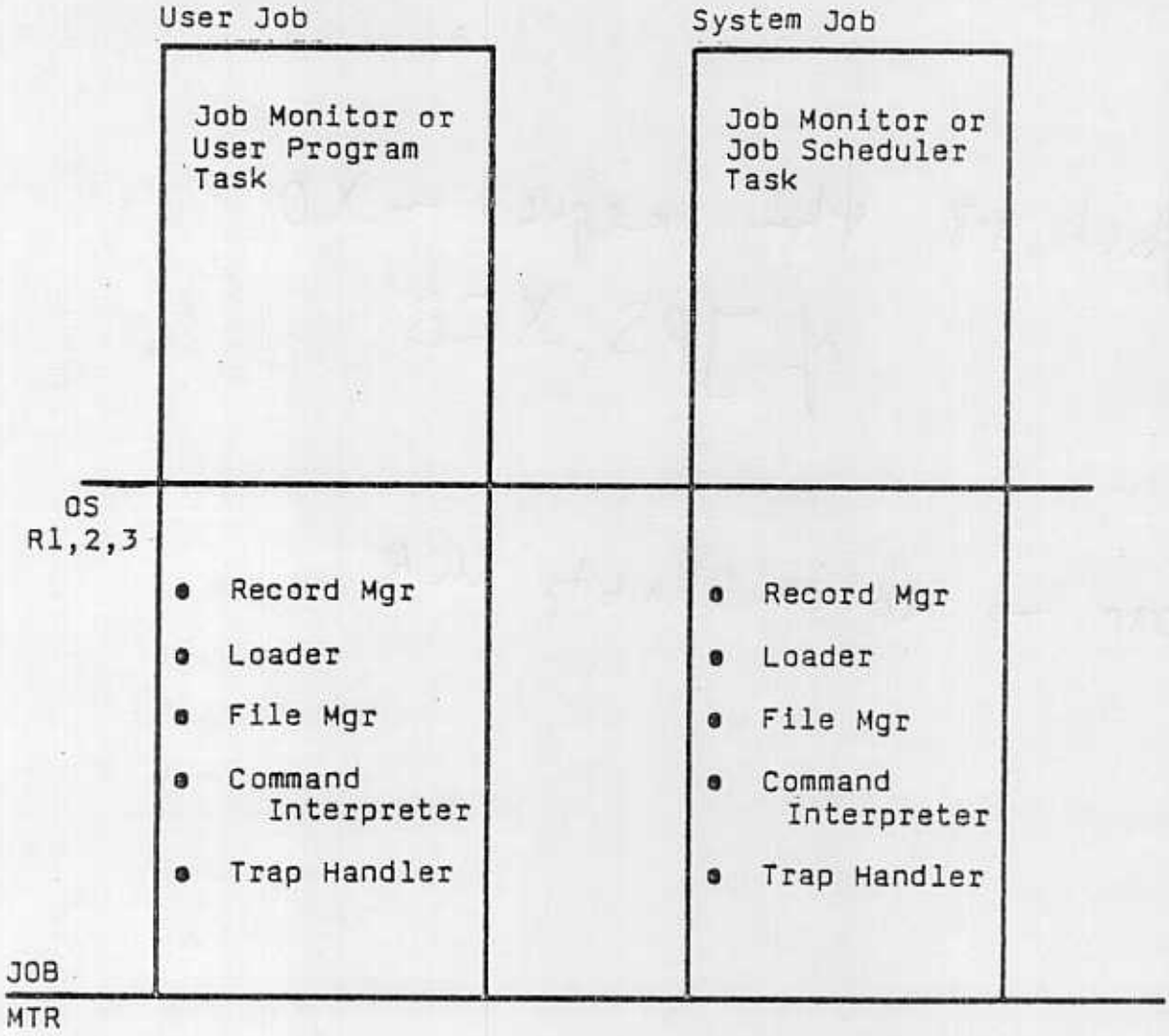


TABLE SEGMENT ATTRIBUTES

Segment	Name	Rings	
1	OSV$MAINFRAME-WIRED	(1,3)	Always in real memory. One per system. Monitor read and write.
2	OSV$MAINFRAME-PAGEABLE	(1,3)	Pageable. One per system.
3	OSV$JOB-FIXED	(1,3)	Wired when the job is active; swapped when the job is swapped. One per job. Monitor read and write.
4	OSV$JOB-PAGEABLE	(2,3)	One per job.
5	OSV$TASK-PRIVATE	(3,3)	One per task.
6	OSV$TASK-SHARED	(3,13)	One per job. Pageable. Shared with other tasks of the same job.
7	OSV$TASK-PRIVATE-R11	(11,11)	One per task. Pageable. Not shared.

CDC behoudt zich recht voor
Ringen te wijzigen
taskes te moven naar andere Ring.

FUNCTIONS



MONITOR

- Task Dispatcher
- Physical I/O
- Page Manager

COMMUNICATIONS

TRANSFER OF CONTROL

	VOLUNTARY	INVOLUNTARY
SAME STATE	CALL/RETURN	TRAP INTERRUPT - CONDITION BIT SET
CHANGE TO JOB STATE	EXCHANGE INSTRUCTION	NO
CHANGE TO MONITOR STATE	EXCHANGE INSTRUCTION	EXCHANGE INTERRUPT - MONITOR COND. BIT SET

asynchron

bit 58 in MCR on – call
off – interrupt.

signals of system flags.
task opstarten met
Free flag set → trap interrupt
→ flag handler.

DATA

	VOLUNTARY	INVOLUNTARY
SAME STATE	o PARAMETERS o TABLES	o CALL STACK
CHANGE TO JOB STATE	o SIGNAL, FLAG o SHARED SEGMENT	NO
CHANGE TO MONITOR STATE	o EXCHANGE PACKAGE o SHARED SEGMENT	o EXCHANGE PACKAGE

EXCHANGE/INTERRUPT

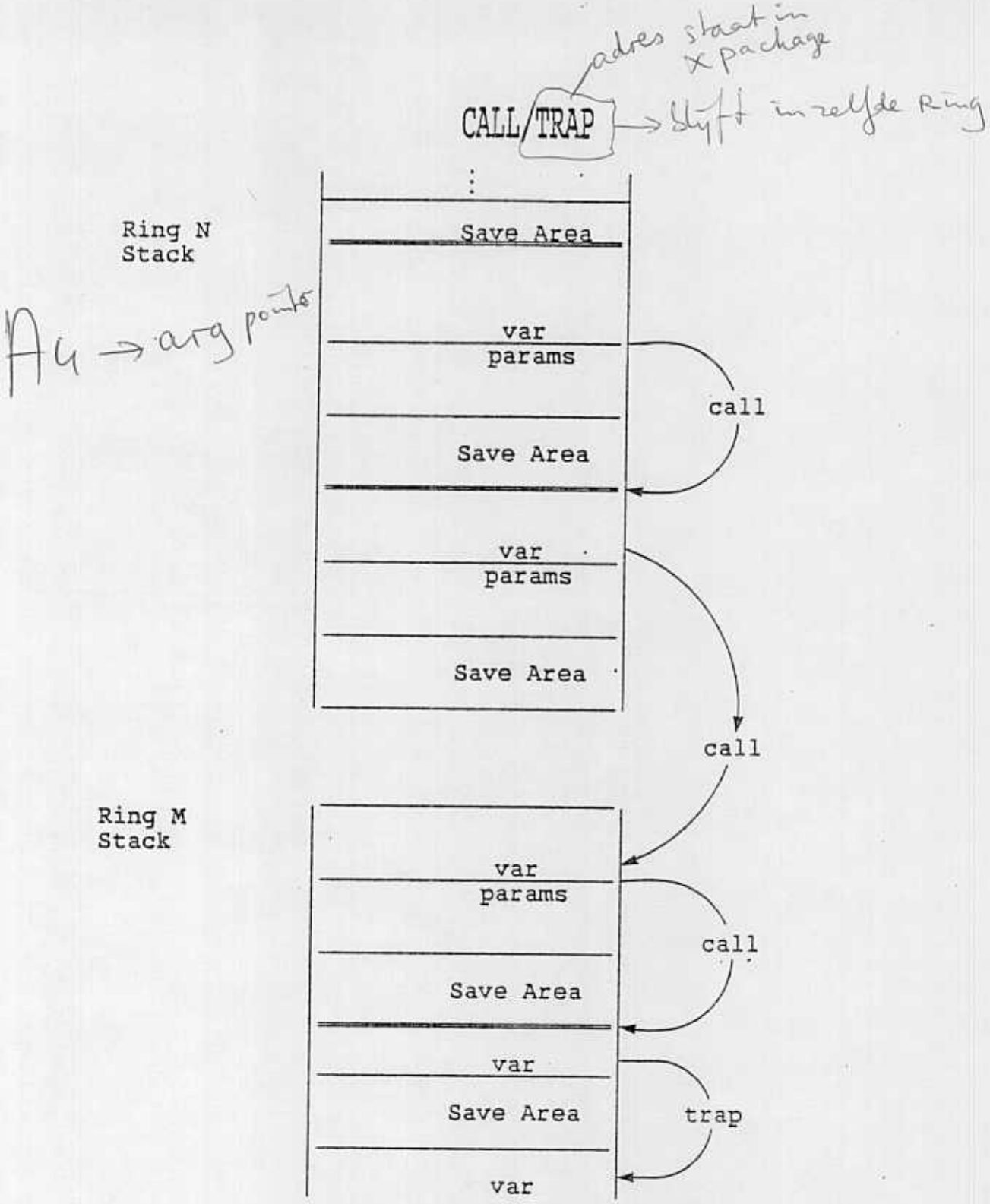
* EXCHANGE JUMP
The exchange jump instruction is used to change state.

Job state programs will exchange to monitor at the PVA in the monitor state XP P register. The system call bit in the MCR is set and the request will be in XO.

Monitor will find the XP of the appropriate task in the XCB entry for that task and exchange to the XP address. A system signal, a system flag or a MCR condition might indicate a special reason for the entry. In that case, monitor will set the free flag in the job state XP and execute the exchange jump. A trap will occur immediately in job state.

* EXCHANGE INTERRUPT
Exchange interrupts occur in job state when a selected monitor condition occurs. Monitor runs at the PVA in the monitor state XP.

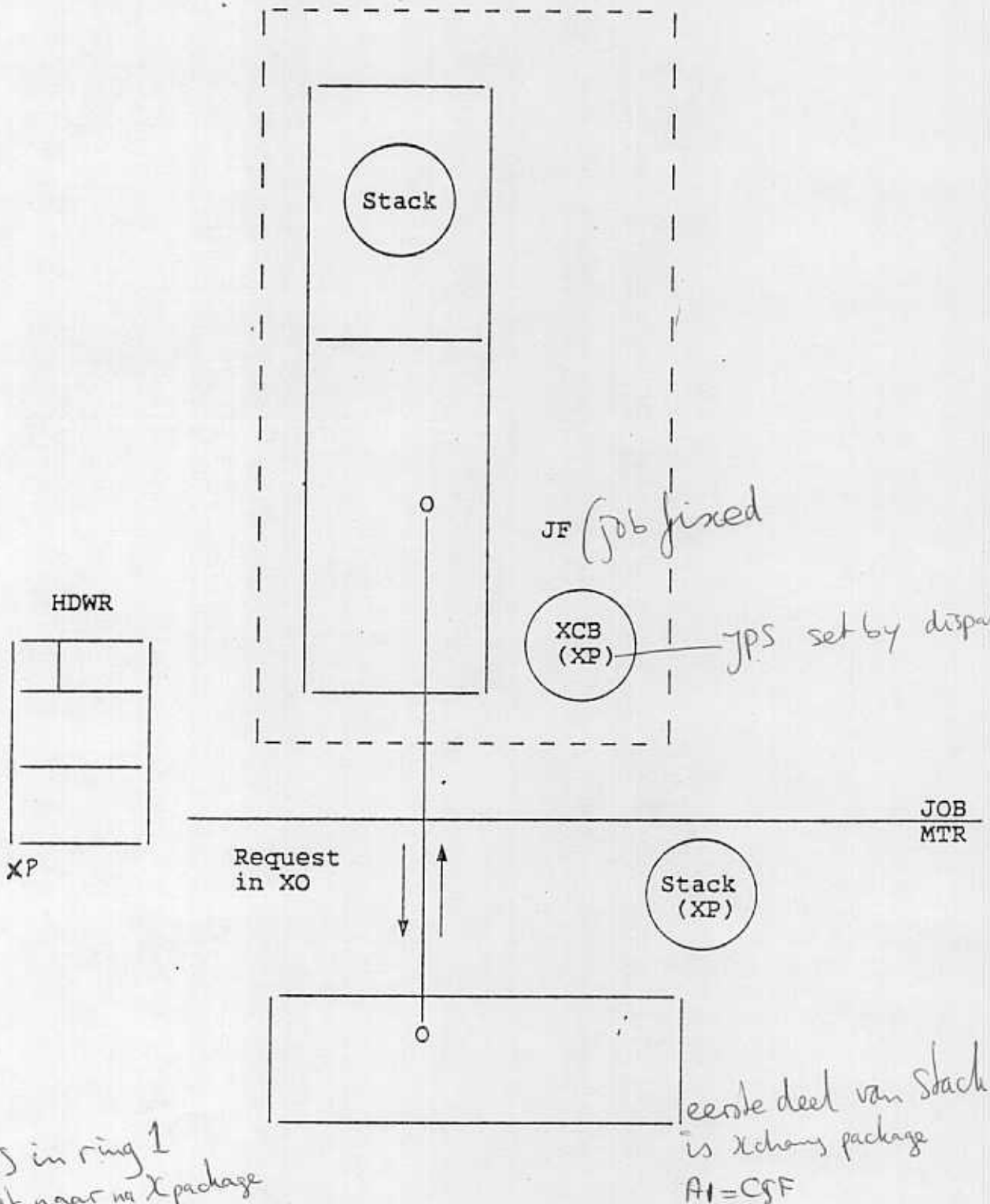
* TRAP INTERRUPT
If traps are enabled, a trap interrupt will occur when a selected user condition occurs in the job state or a selected monitor condition occurs in monitor state. In either case there is no exchange. A stack frame is built and the trap handler is executed.


CALL/TRAP
Ring N
Stack
Save Area
var
params
call
Save Area
var
params
Save Area
call
Ring M
Stack
var
params
call
Save Area
var
Save Area
trap
var

M < N

EXCHANGE

Request in X0



TOS in ring 1
- wijst naar nu X package

eerste deel van Stack
is xchang package
A1 = CSF

DATA SEGMENTS (Read, Write Permission)

Ring numbers R1 and R2 are in the Segment Description Table (SDT).
Read Bracket: 1..R2.
Write Bracket: 1..R1.

Suppose R is the ring of execution, then a data segment may be read if R $\leq$ R2 and written if R $\leq$ R1.

CODE SEGMENTS (Read, Execute Permission)

Ring numbers R1 and R2 are in the Segment Description Table (SDT). R3 is in the binding segment entry. The linker constructs the binding segment and will insert R3 only if the procedure is gated.

Read Bracket:	1..R2
Execute Bracket:	R1..R2
Call Bracket:	R2+1..R3

Suppose that R is the ring of execution, then a call to a procedure in the range R1..R2 is valid and the procedure will run in Ring R; a call to a procedure in the range R2+1..R3 is valid and the procedure will run in ring R2.

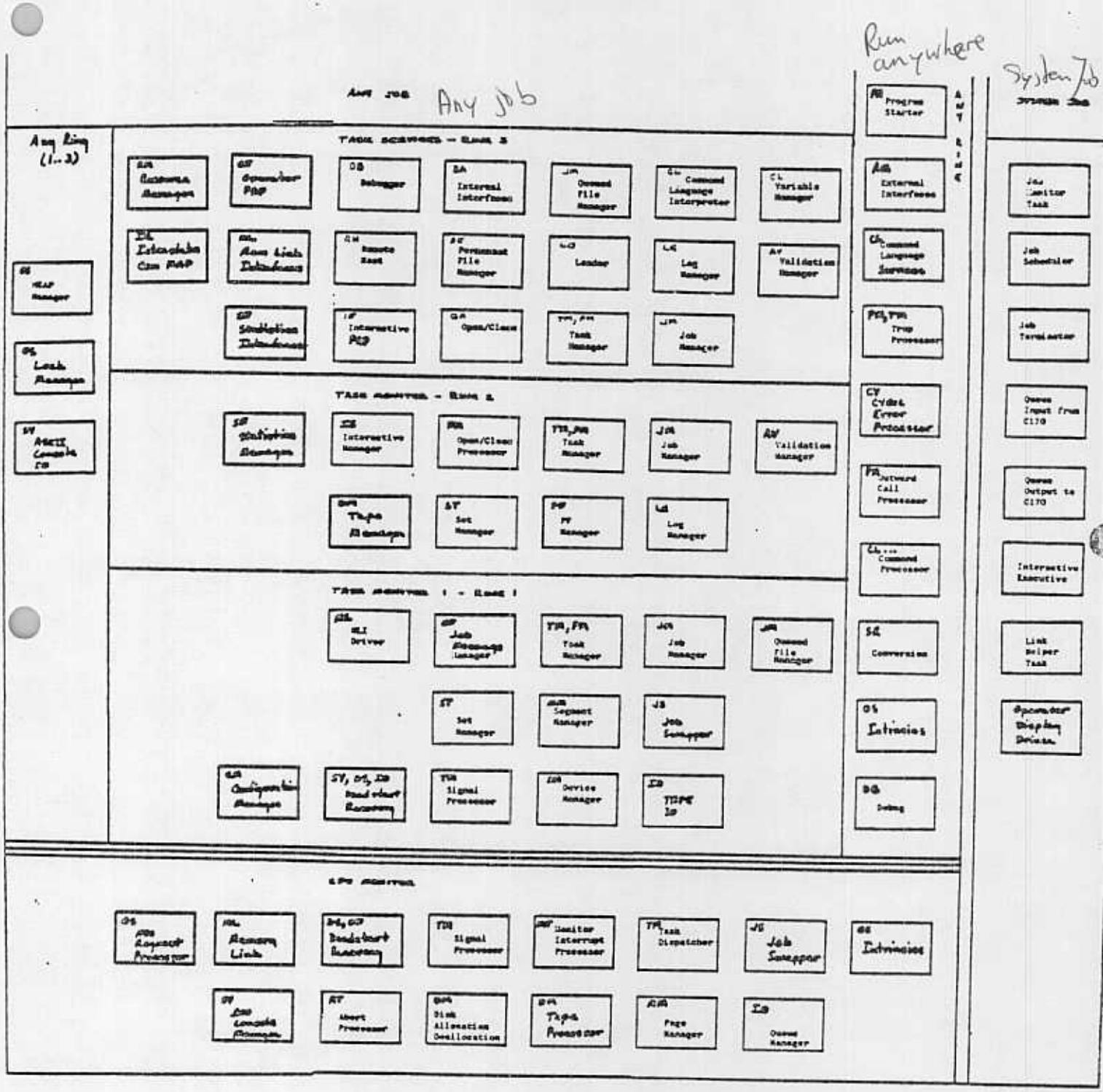
DATA

ACCESS FROM / RINGS	1	2	3	B	D
1,3	R W	R	R		
2,3	R W	R W	R		
3,D	R W	R W	R W	R	R
B,B	R W	R W	R W	R W	

CODE

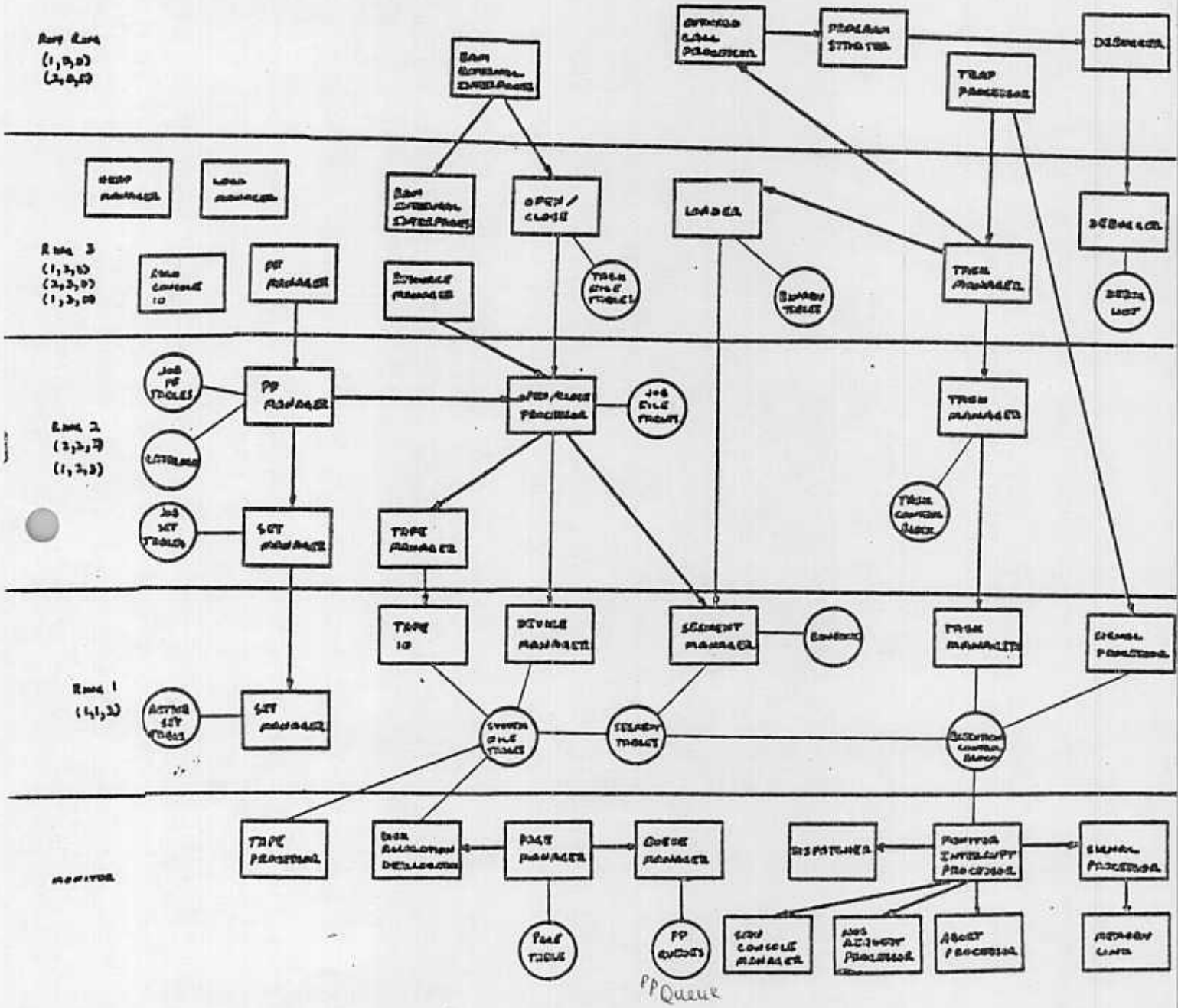
CALLED FROM / RINGS	1	2	3	4	B	D	F
1,1,3	1	1	1				
2,2,3		2	2				
2,3,D		2	3	3	3	3	
1,D,D	1	2	3	4	B	D	
2,D,D		2	3	4	B	D	
B,B,B					B		

NOS/VE MODULES



O = tables

NOS/VE MODULES AND TABLES - 1

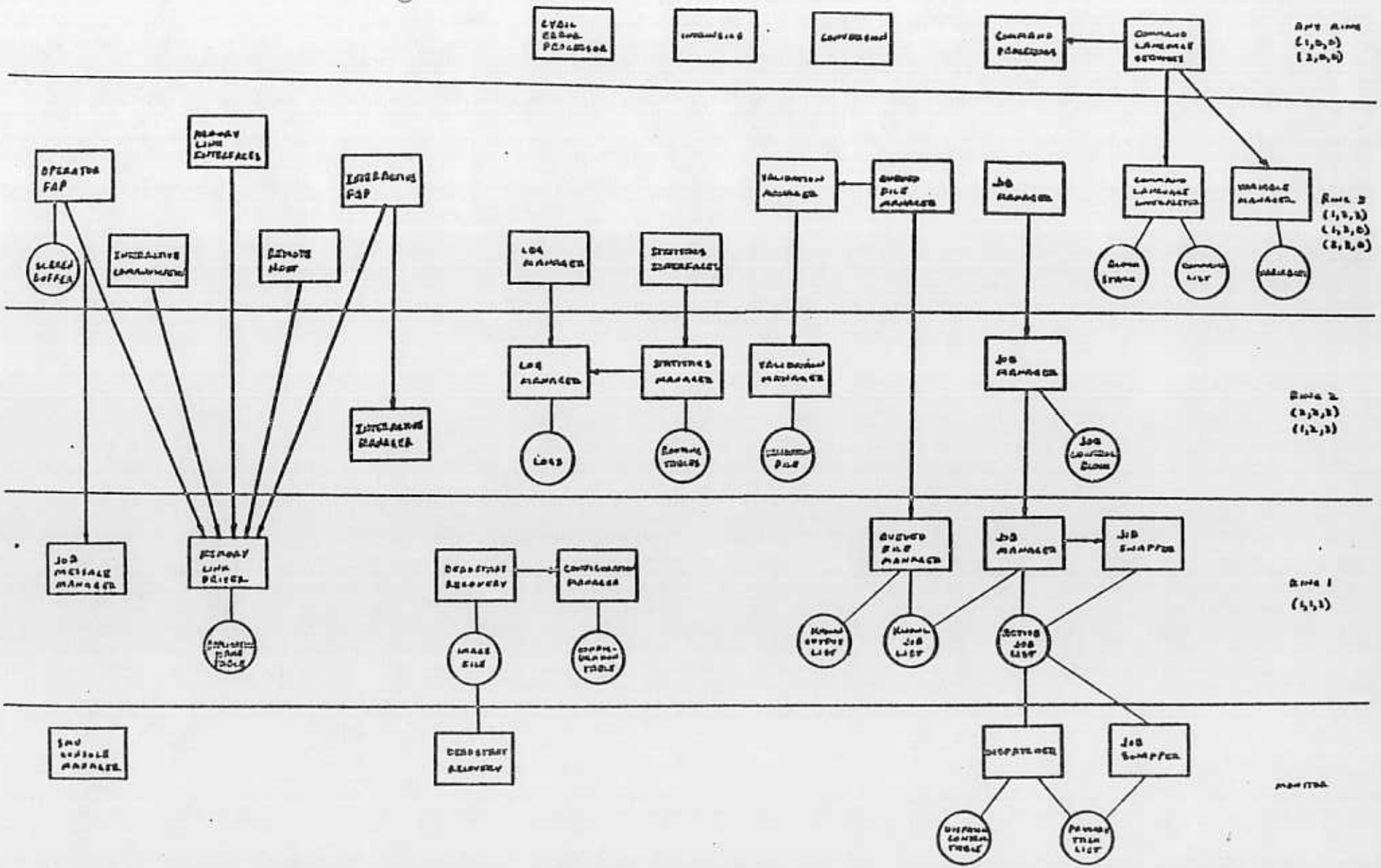


Code in Ring 3 wordt door Ring 1 by initialization gedruckt

(System CORE to klein magelijk)

NOS/VE MODULES AND TABLES - 2

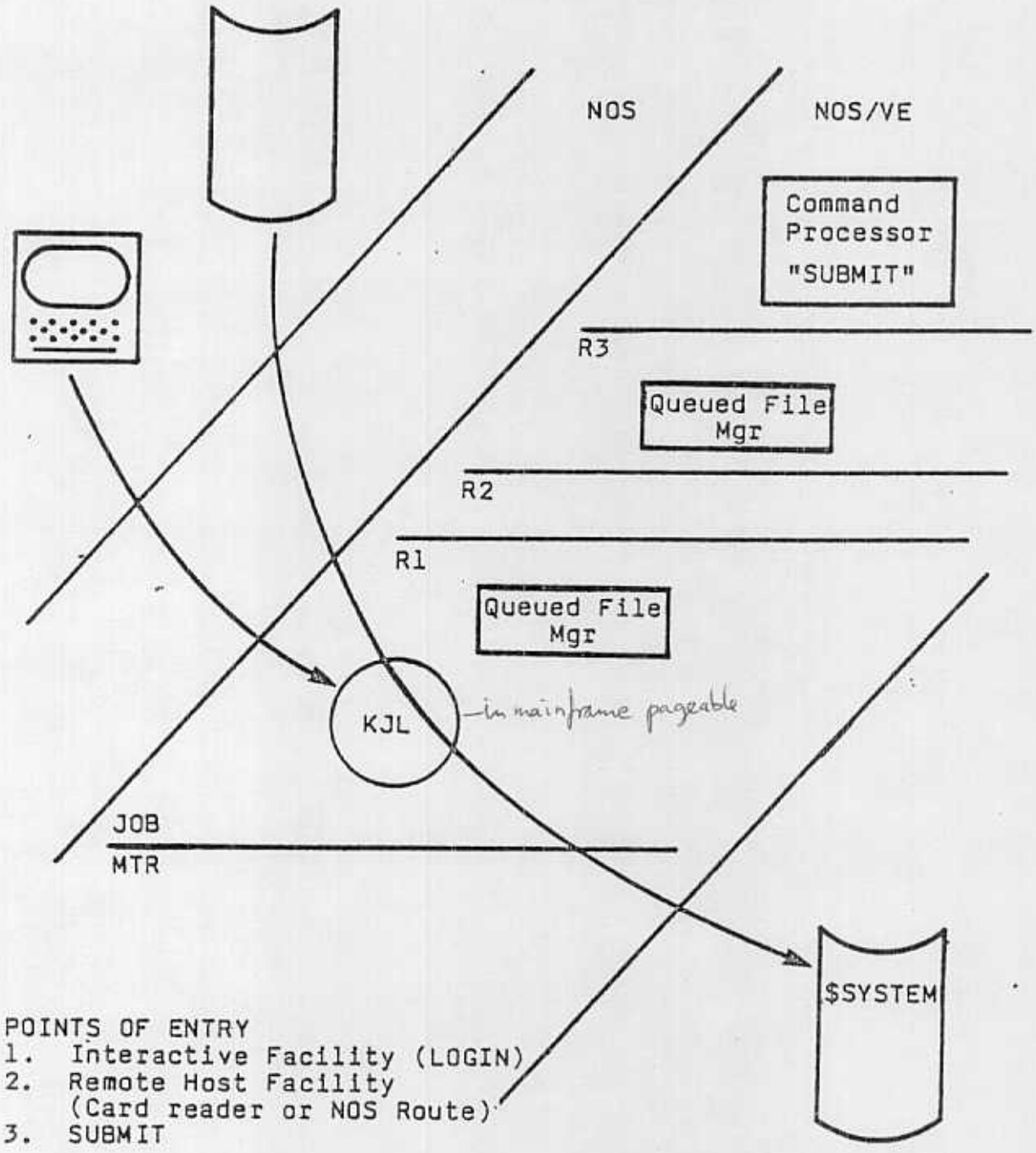
Program interface



SUBMIT JOB

```
   .
   .
   .
/collect_text   ABC
login JHW81
execute DEF                    ------PMP$EXECUTIVE
Logout
**
/submit_job  ABC               ------JMP$ROUTE
   .
   .
   .
```

1
JOB ENTRY



POINTS OF ENTRY
1. Interactive Facility (LOGIN)
2. Remote Host Facility
   (Card reader or NOS Route)
3. SUBMIT

JOB QUEUEING

* Queued file manager is part of task services. It processes the jmp$route request.

* Queued files are validated and registered in the $SYSTEM catalog and queued through the known job list (KJL).

* The KJL entry for a job is linked into a thread which represents one of the following states.

"Deferred"	waiting for a time interval to elapse
"Queued"	waiting to be initiated
"Initiated"	active, inactive or swapped out but available for execution
"Terminated"	completed but output files queued for disposition

Mainframe pageable. [1,3]

JOB SCHEDULER

periodic

signal

* Job scheduler executes as a task in the system job.

* Job scheduler determines:

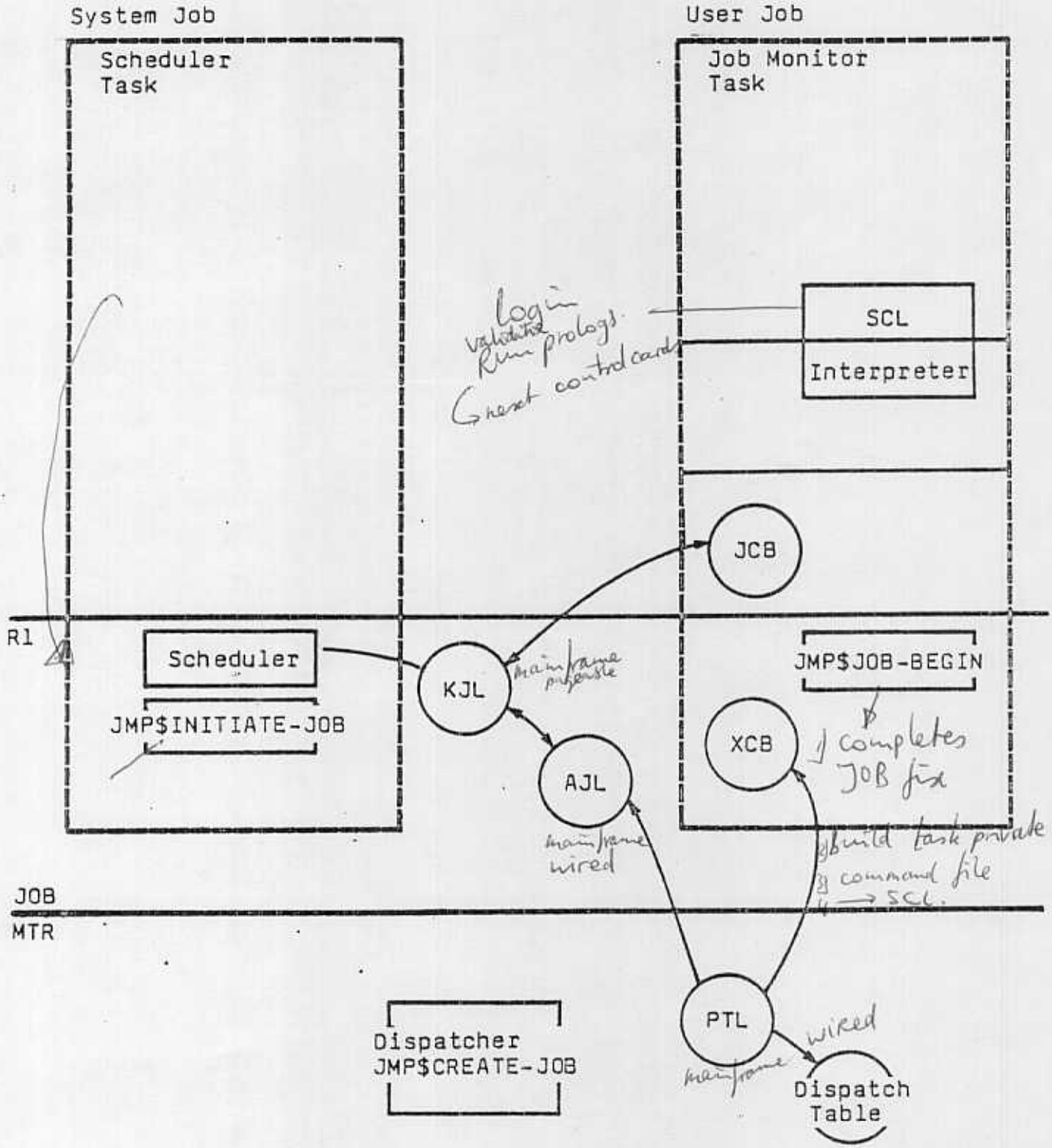
  -Order in which jobs in the input queue should be initiated
  -When a job should be swapped into or out of memory

* Some examples of scheduling criteria are:

  -Current priority within job class
  -Job resource requirements
  -Job class and status
  -Current system resource availability

* Job scheduler monitors the available mix of queued and initiated jobs and prioritizes them based on current system usage.

2
INITIATE JOB



JOB INITIATION

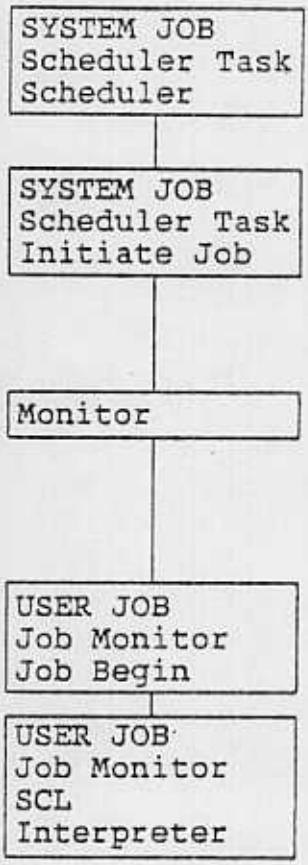
* When a job is selected, it is given an entry in the Active Job List (AJL)

* Initiate_job with the help of monitor initialized the OSS$job_fixed segment for the new job. The Job Control Block (JCB) is built. The Execution Control Block (XCB) is initialized with the XP for the first task to run (Job Monitor)

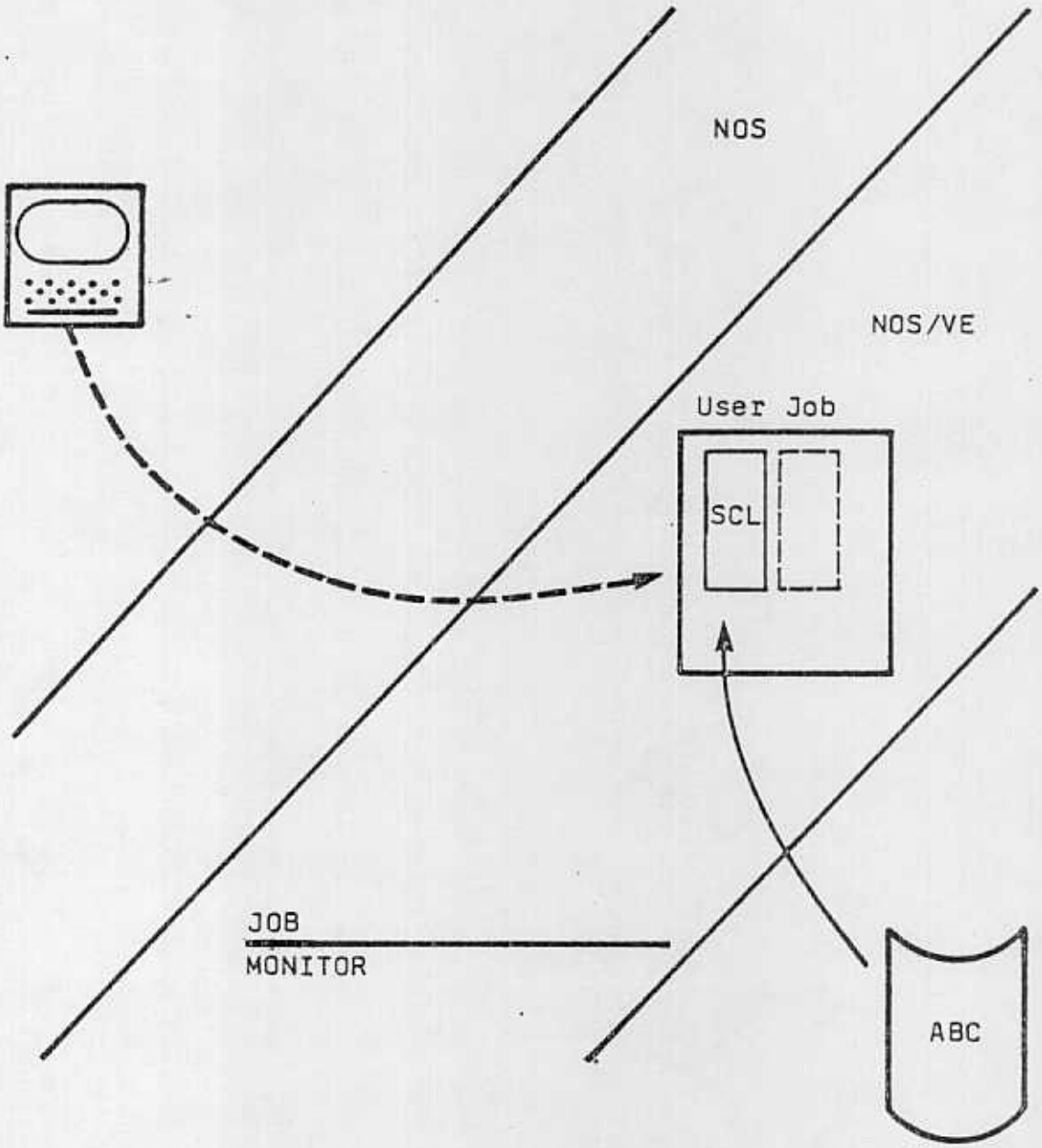
* Monitor creates a Primary Task List (PTL) entry and logs the job monitor task into the dispatch table. The new job waits its turn. Eventually the dispatcher gives the new job its first time slice.

* Job begin initializes ~~OSS$job_pageable and~~ OSS#task-shared segments. The command file, output file, and job log are also initialized.

* The SCL interpreter interprets the first command (LOGIN). The user is validated and the prolog is executed.



3
COMMAND PROCESSING



* SCL reads the command from the $COMMAND file.

* SCL searches for the command in the command list, by default -

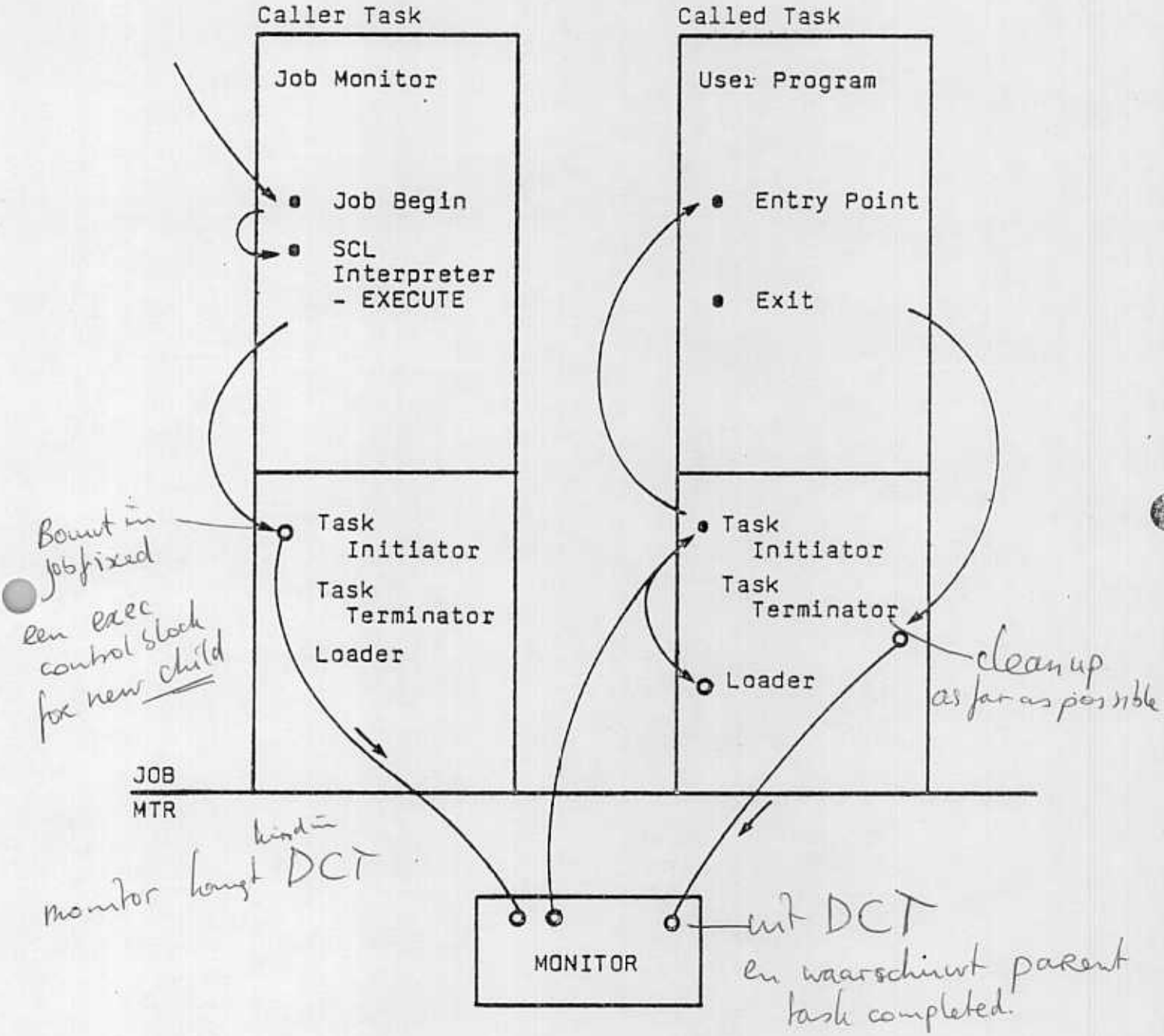
  $LOCAL
  $SYSTEM

* If SCL finds the command it runs the CYBIL procedure which must have been provided to process it. This procedure can run as part of the current task or as a new task.

* If the command is a file name call, it might be a program or an SCL procedure file.

* In all cases, the SCL Interpreter passes the command parameter list to a processor. The processor can now use other SCL interfaces to crack the command.

PROGRAM EXECUTION EXAMPLE



PROGRAM EXECUTION FLOW

* Program or command requests program execution
* Calls task initiator

* Builds tables for the new task

* Exchanges to system monitor to request task initiation

* Links new task into CPU dispatch list

* CPU is dispatched to new task

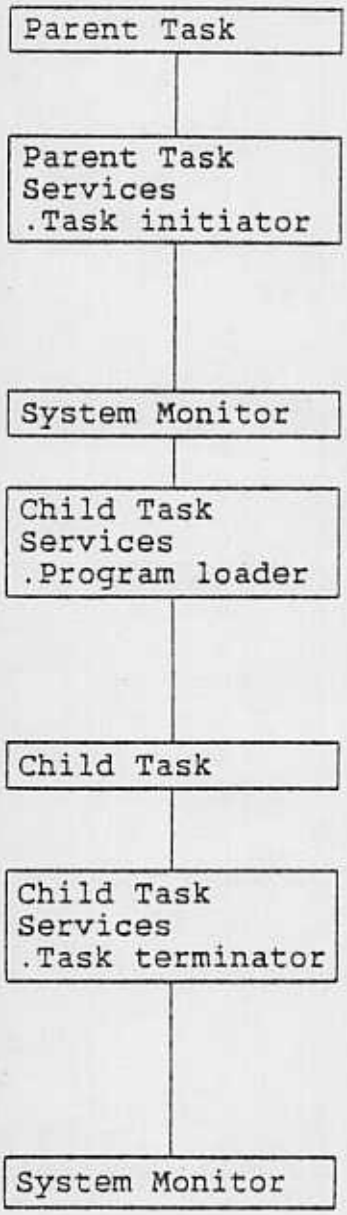
* Loader loads object module
* Loader passes control to initial entry point

* New task executes asynchronously to caller task

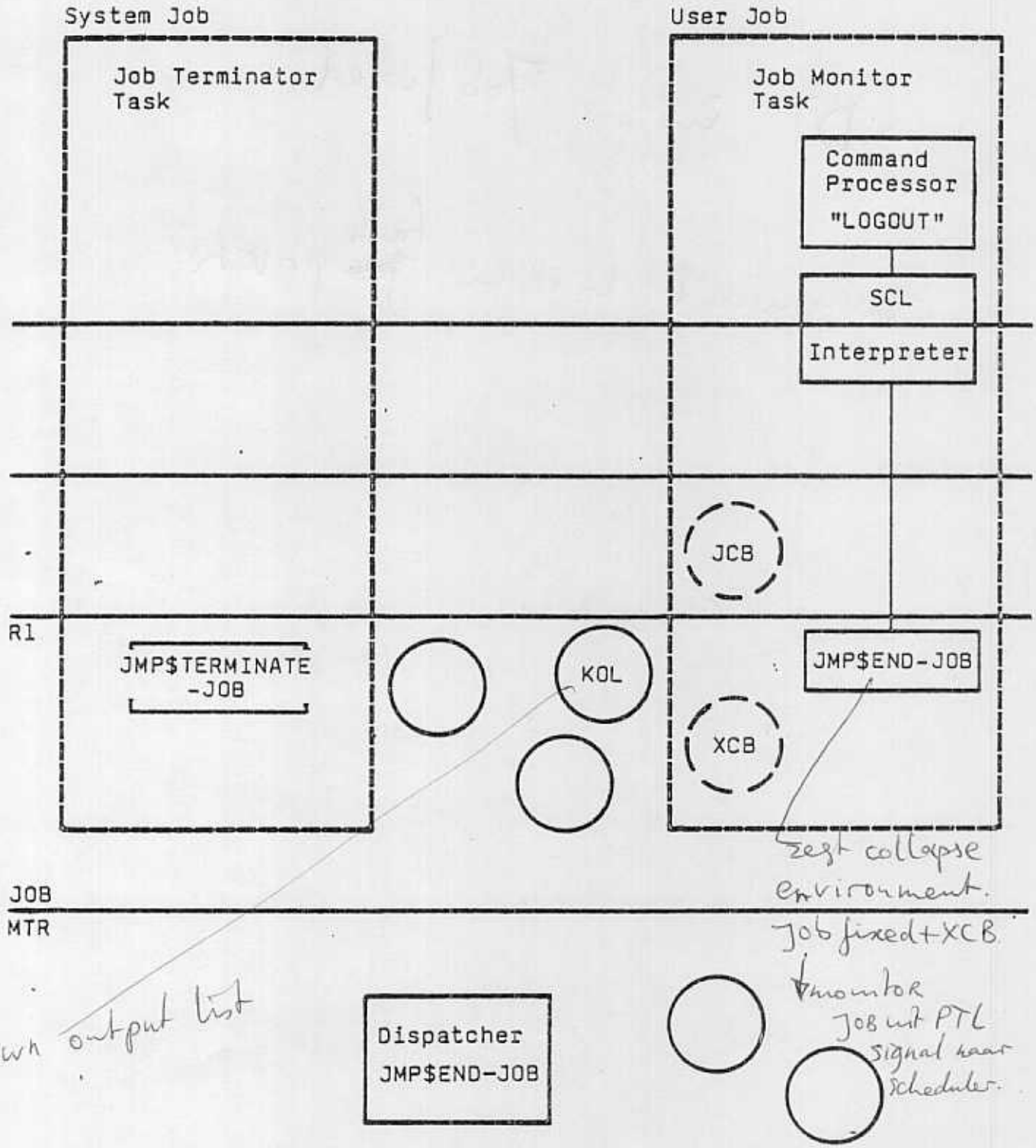
* New task calls exit interface

* Cleans up task
* Exchanges to system monitor to request task termination

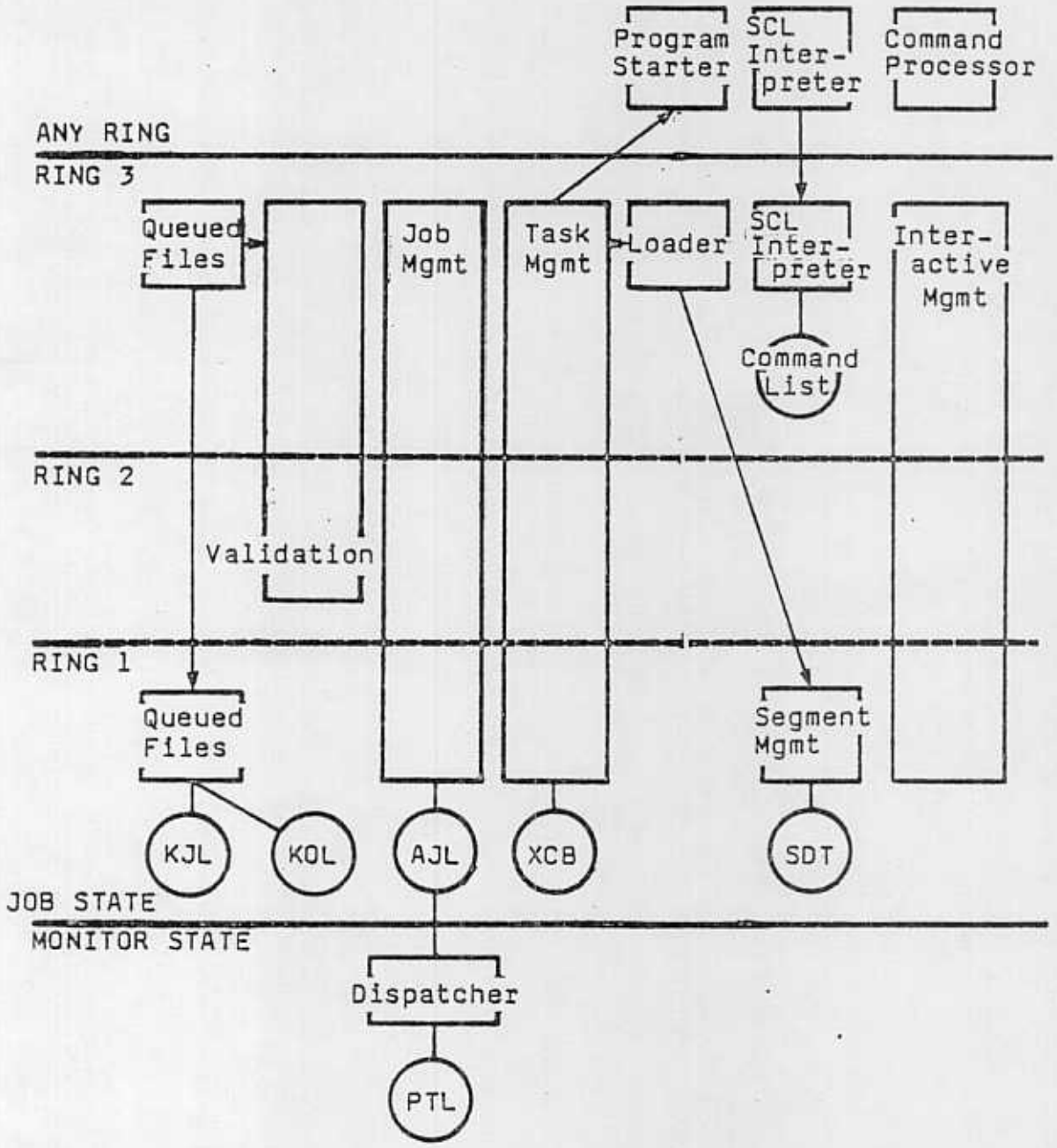
* Remove task entries from dispatch list
* Informs caller that callee has terminated



4
JOB TERMINATION



JOB FLOW
PACKAGING



JOB FLOW TABLES

COMMAND LIST	This list will be searched by SCL interpreter. If the command if found, a command processor will be called.
KJL-jmt$known_job_list_entry (JMDKJL)	All jobs in the system have an entry on this table.
KOL-jmt$known_output_list_entry (JMDKOL)	All output files waiting for routing have entries on this table.
AJL-jmt$active_job_list (JMDAJL)	All jobs that have been initiated and are not swapped have entries on this list.
XCB-ost$execution_control_block (OSDXCB)	Every task in a job has an XCB. This table contains the tasks exchange package.
SDT-mmt$segment_descriptor_table (MMDSDT)	One SDT per task. Every segment in every task has an entry in an SDT. The SDT is used by hardware to relate VM address to real memory addresses.
TCT-tmt$dispatch_control_table (TMDDCT)	The dispatcher organizes tasks in this table by priority and chooses the appropriate candidate for execution.
PTL-tmt$primary_task_list (TMDPTL)	This monitor table contains global information about every task in the system.
JCB-jmt$job_control_block (JMDJCB)	There is one JCB per job. The JCB contains limits, statistics, names, etc., for the job.

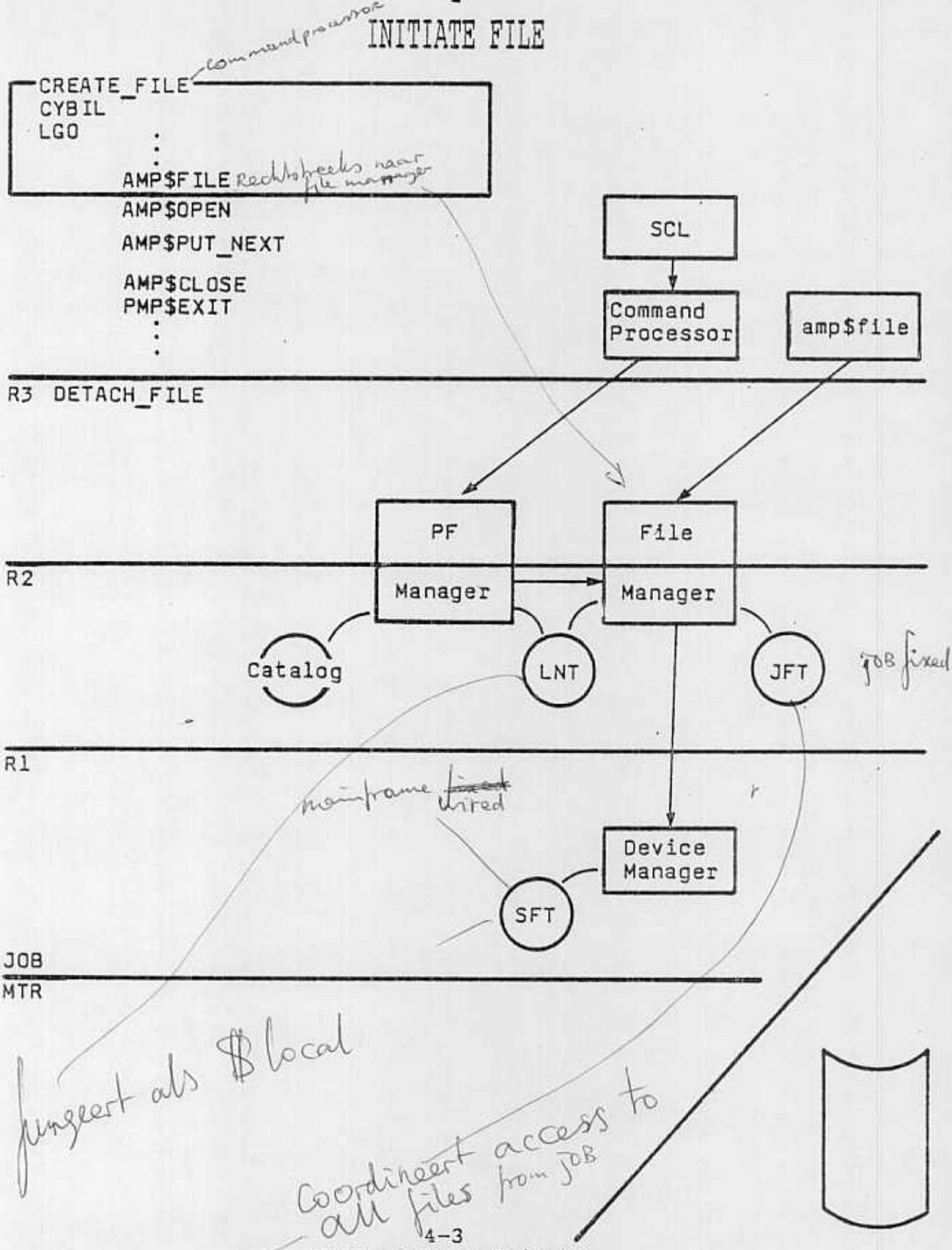
( ) Common Deck Name

CREATE FILE

```
.
.
.
CREATE_FILE EX
CYBIL ...
LGO
                .
                .
                .
      AMP$FILE ('EX', attributes...
      AMP$OPEN ('EX', amc$record,
        attributes, fid...

      AMP$PUT-NEXT (fid,...

      AMP$CLOSE (fid,...
      PMP$EXIT (status)

DETACH_FILE EX
```

1

INITIATE FILE



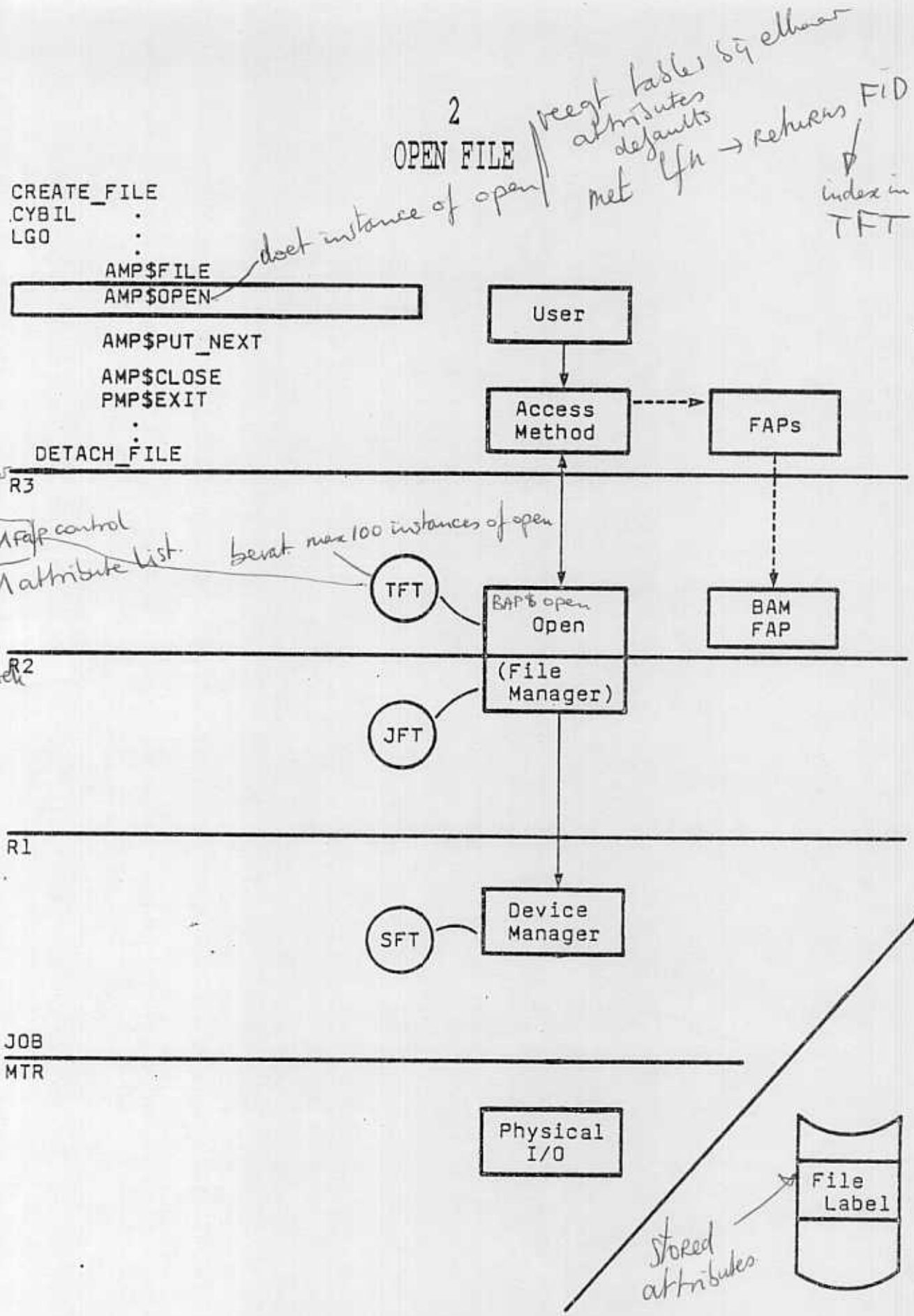
FILE INITIATION

* If the first mention of the file is on a command, then the Job File Table (JFT), the Local Name Table (LNT), and the System File Table (SFT) are built. The commands are:

  REQUEST - TAPE
  REQUEST - TERMINAL
  PRINT
  FILE
  Any PF Command

* For amp$file and some other requests, an auxiliary request table is built. The file tables are built when the file is opened for the first time.

2
OPEN FILE


CREATE_FILE
CYBIL .
LGO .
.
AMP$FILE
AMP$OPEN
AMP$PUT_NEXT
AMP$CLOSE
PMP$EXIT
.
.
DETACH_FILE
R3
R2
R1
JOB
MTR
User
Access Method
FAPs
TFT
BAP$ Open
Open
(File Manager)
BAM FAP
JFT
SFT
Device Manager
Physical I/O
File Label
Stored attributes

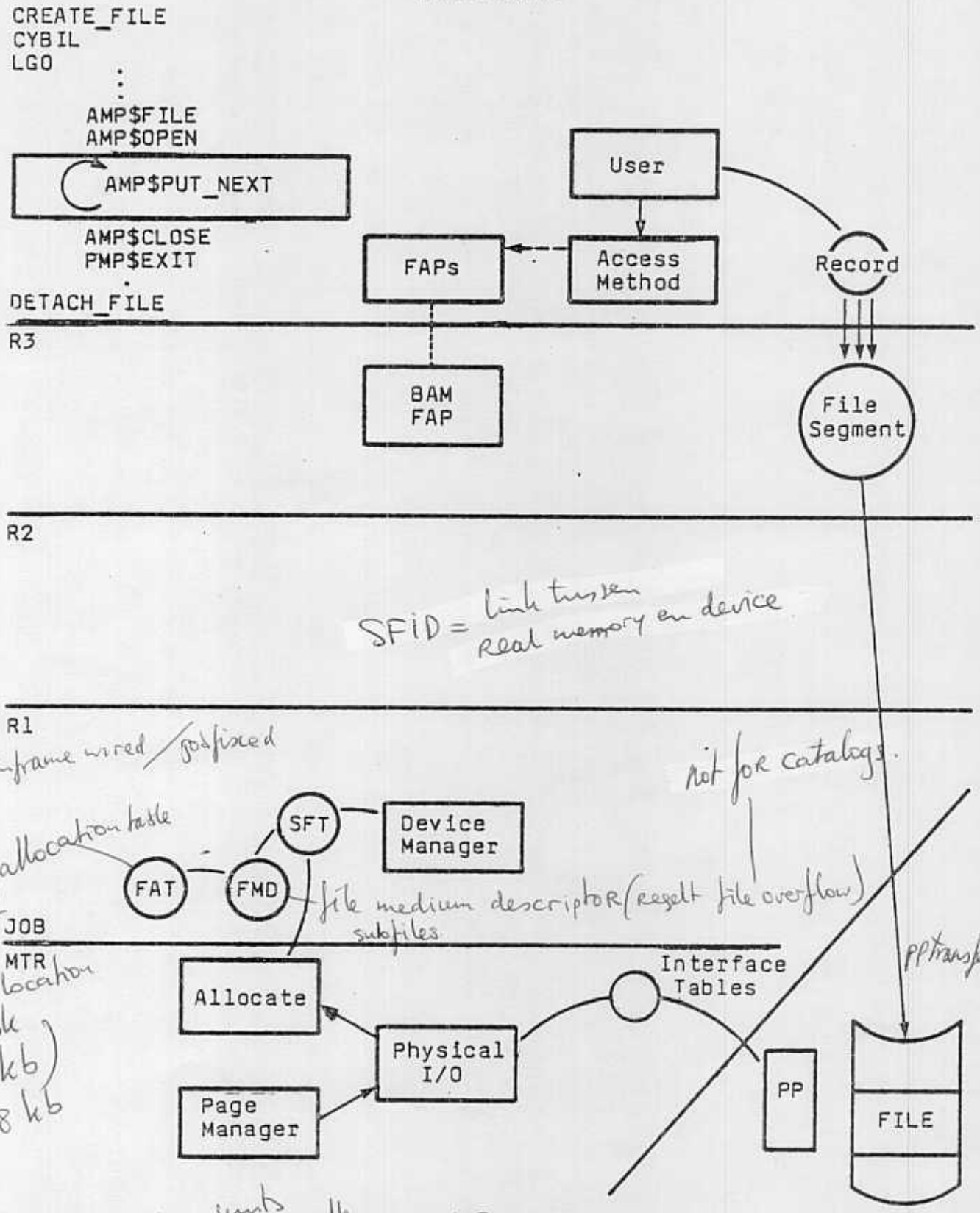
* When the file is opened, the information from commands, program interface requests and AMP$OPEN will be used. The precedence is:

    1. AMP$OPEN
    2. Commands
    3. Requests

* Open entails various processing depending on the file residence and direction of transfer. For example:

DISK	File attributes are read or written
TAPE	Labels are checked or created (R2)
TERMINAL	Attributes are sent to the interactive facility

WRITE FILE



ACCESS LEVELS

* PUT-NEXT
The access method gets records from the user's buffer and puts them in the file segment that is opened for that purpose (i.e., the system does segment level access). The paging mechanism will take care of real memory and device manager will make sure that space is allocated on the disk. When the filled pages are needed by the system, page manager will instruct the physical I/O component to transfer them.

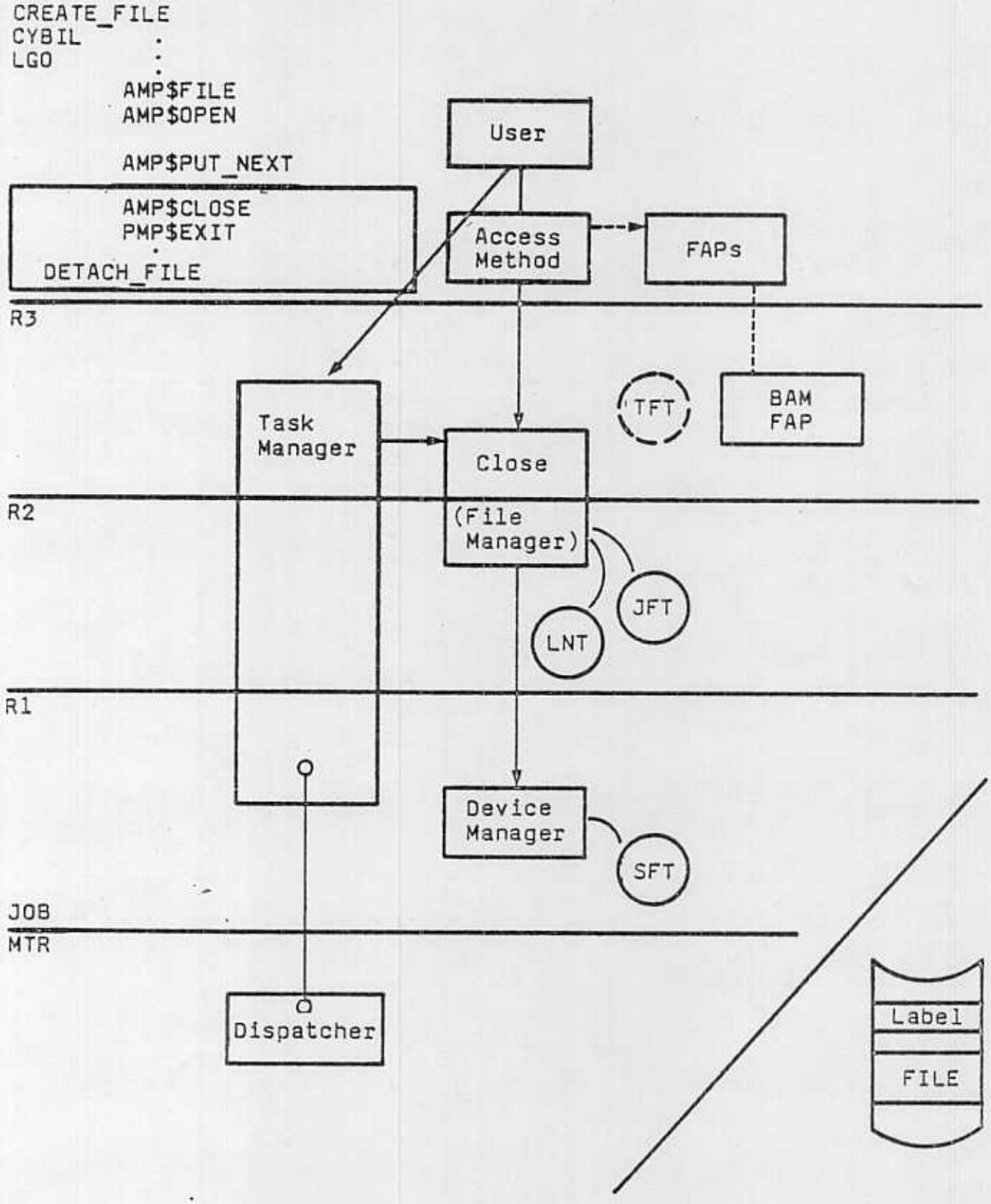
* GET-NEXT
Again the access method opens a file segment. Page faults will occur when the needed data is not in real memory. But the access methods is not aware of that; it simply copies the records from the file segment to the user's record buffer.

* Segment Level Access
If the user opens the file for segment level access, the file segment is directly addressable by the user.

* Page Manager
After Systems initiation, all disk I/O is initiated by page manager which is a part of system monitor. Page manager runs as the result of a page fault (page seek without file condition).

On a read, page manager will find a free page, initiate physical I/O and suspend the task. When I/O is complete, the task is reactivated. On a write, page manager will find a free page and return to the faulting task. The hardware will mark the page modified when data is written to it. Page manager will write the modified page to backing store (disk) eventually.

4
CLOSE FILE


CREATE_FILE
CYBIL
LGO
AMP$FILE
AMP$OPEN
AMP$PUT_NEXT
AMP$CLOSE
PMP$EXIT
DETACH_FILE
User
Access Method
FAPs
R3
Task Manager
TFT
BAM FAP
Close
(File Manager)
R2
JFT
LNT
R1
Device Manager
SFT
JOB
MTR
Dispatcher
Label
FILE

CLOSE/RETURN

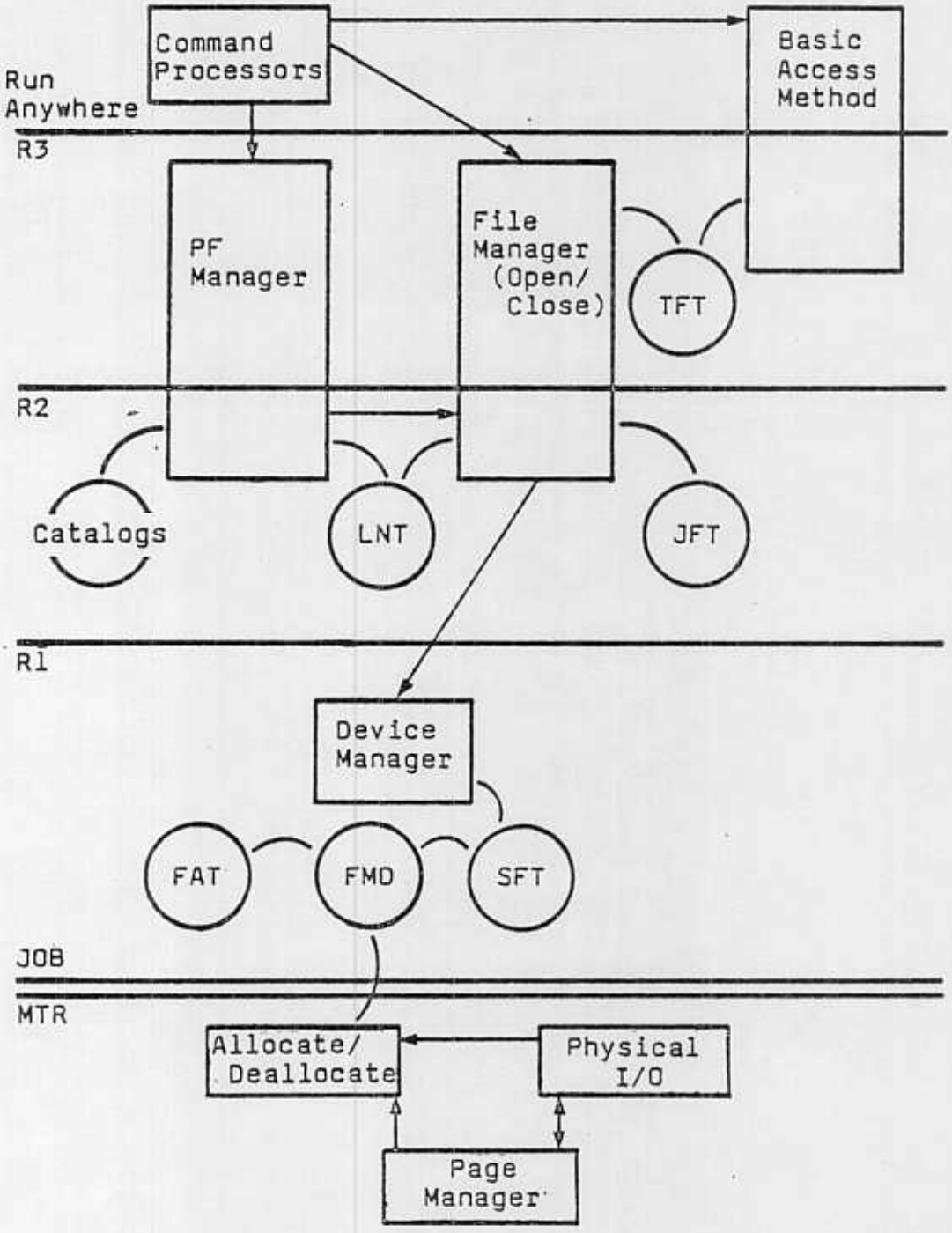
* AMP$CLOSE is a request to close this instance of open. The Task File Table (TFT) will be dismantled if there are no other opens. At task termination (PMP$EXIT, for example) all files in the task will be closed once for each instance of open. Job and System File Tables will remain.

* RETURN will cause all references to the file to be deleted. Examples of file disposition are:

DISK	Temporary files will no longer be accessible. Permanent files will be known through the user's catalog only.
TAPE	Trailer labels will be processed (R2). The volume will be returned.
TERMINAL	Disconnected and returned.

* At job termination all files are closed and returned.

FILE FLOW
PACKAGING

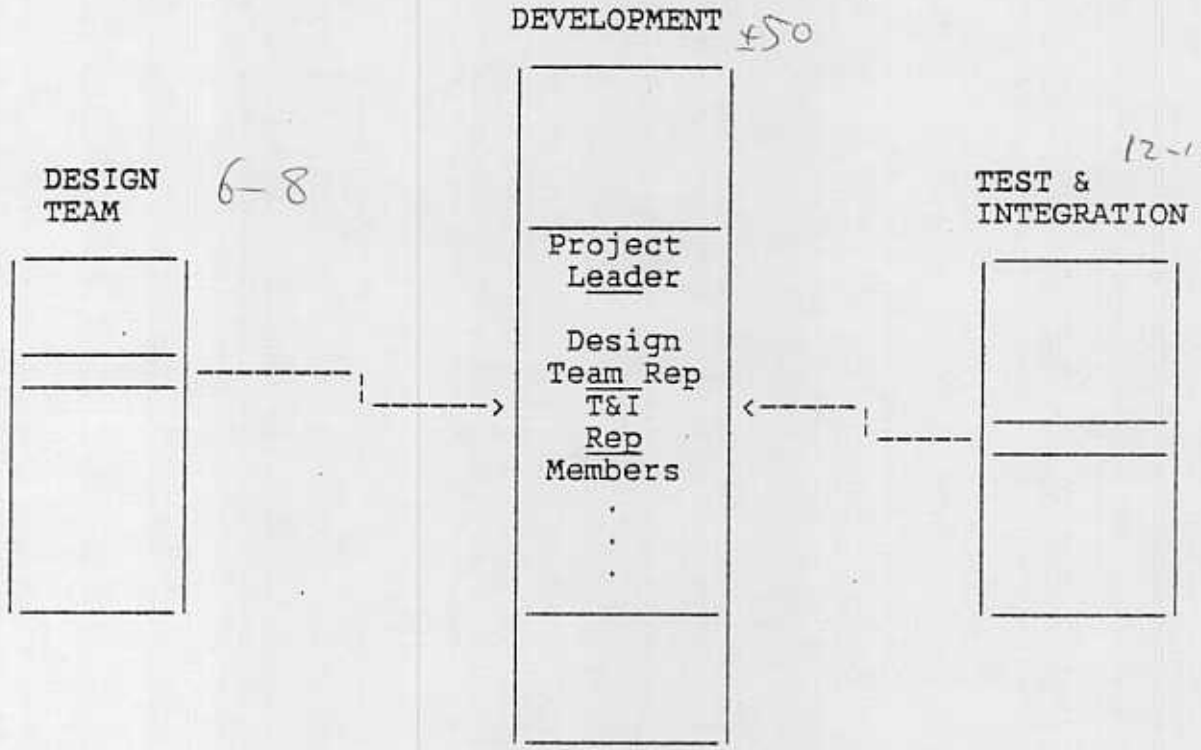


FILE FLOW TABLES

TFT-bat$task_file_table (BADTFT)	All files opened by a task are controlled by this table. Entries contain pointers to record and block descriptors, file attributes, user request tables, and file access procedures.
LNT-fmt$local_name_table (FMDLNT)	This table controls the files known to a job by name. It keeps track of the request and attribute info which is global to the job.
JFT-fmt$job_file_table (FMDJFT)	This table has information about all the files known to the job including unnamed segments like stack and binding.
Catalogs	Each user has a master permanent file catalog.
SFT-dmt$file_descriptor_table (DMDSFD)	These tables have entries for all files in the system at a given time. Entries point to tables which describe the file on the device.
FMD-dmt$ms_file_medium_descriptor (DMDFMD)	This table lists the volumes on which a file has been allocated. The portion of the file on a particular volume is called a subfile.
FAT-dmt$ms_file_allocation_table (DMDFAT)	There is one FAT per subfile. It describes the physical location of the file on the device.

( ) Common Deck Name

NOS/VE PROJECT ORGANIZATION



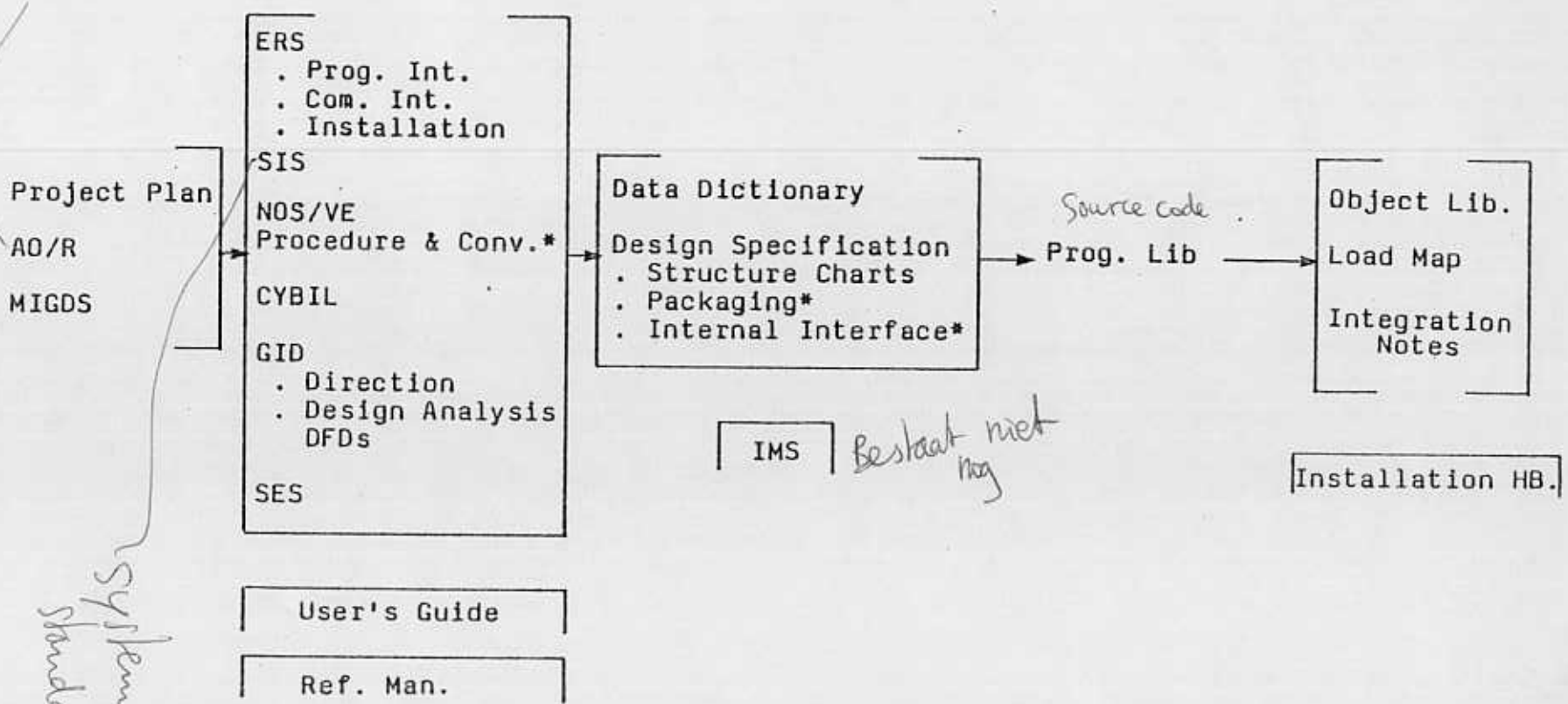
DESIGNERS

- Job Mgmt
- Program Mgmt
- I/O
- Dual State
- Deadstart

DEVELOPMENT GROUPS

- PFs
- Physical I/O
- Logical I/O
- Dual State Communication
- Logs
- Program Control
- Program Execution
- Job Mgmt
- Command Language
- Monitor
- Maintenance
- Deadstart

MATERIALS



*Class Handout

NOS/VE PROCEDURES & CONVENTIONS

1. Introduction — development cycle
2. Design Team
3. Document Review Process
4. Product Identifiers
5. Design Documentation
6. Procedure Interface Conventions
7. NOS/VE Program Library Conventions
8. CYBIL Coding Conventions — message handling
9. Keypoint Usage
10. Code Submittat Process
11. NOS/VE Document Maintenance
12. Data Dictionary Conventions
13. Yourdon Methodology
14. Code Review Process

PREFIX NOMENCLATURE

SYNTAX:

XXC$ . . . = Constant

XXT$ . . . = Type

XXE$ . . . = Error Code

XXP$ . . . = Procedure

XXM$ . . . = Module

XXV$ . . . = Variable

XXK$ . . . = Keypoint

XXS$ . . . = Segment

XXF$ . . . = File

ID CODE (XX):

AM = Access Methods

CL = Command Language

IC = Interstate Communication

IF = Interactive Facility

JM = Job Management

OF = Operator Facility

OS = Operating System

PF = Permanent Files

PM = Program Management

RM = Resource Management

SF = Statistics Facility

MM = Memory Management

FM = File Management

MT = Monitor

LO = Loader

CY = CYBIL

IO = Input/output

OC = Object Code

PU = PF Utilities

SR = Conversion Services

DB = Debug

BA = Basic Access

RH = Remote Host

ML = Memory Link

II = Interactive Interface

DP = Display

SY = System

ST = Sets

TM = Task Management

DM = Device Management

LG = Logs

AV = Accounting/Validation

DS = Deadstart

CM = Configuration Management

JS = Job Swapper

DECK NAMING CONVENTION

pptzzzz

pp = two character identifier

t = deck type

zzzz = mnemonic name

DECK TYPES

M = CYBIL

P = PP Assembler

A = CP Assembler

X = XREF declarations*

D = Type and Constant declarations*

H = Documentation Header*

I = In-line procedure*

E = Example

* = common deck

INTERNAL INTERFACE

- Chapter Descriptions
- Procedure Descriptions

  -Request Description
  -Parameter Description
  -XREF Declarations
  -Common Deck Calls

- Common Deck Expansions
- Topics

  CP MONITOR
  Job Management
  Resource Management
  Segment/Memory Mgmt
  Memory Mgmt
  Queued Files
  Program Mgmt
  Preemptive Communication
  File Mgmt
  PF Mgmt
  SCL
  Interstate Com.
  Memory Link
  Log Mgmt
  System Access
  Accounting
  Operator Facility

- Intrinisics

inline procedures/functions
direct machine instructions

INTRINSICS

```
1 #CALLER_ID (ID)
2 #CALL_MONITOR (REQBLK)
3 #COMPARE (S1,S2): RESULT
4 #COMPARE_COLLATED (S1, S2, TABLE): RESULT
5 #COMPARE_SWAP (LOCK, INITIAL, NEW, ACTUAL, RESULT
6 #DISABLE_TRAPS (OLD_TE)
7 #ENABLE_TRAPS (OLD_TE)
8 #FREE_RUNNING_CLOCK (PORT): INTEGER
9 #HASH_SVA (SVA, INDEX, COUNT, FOUND)
10 #INTERRUPT_PROCESSOR (PORT_SELECTOR)
11 #KEYPOINT (CLASS, EXPRESSION, CODE)
12 #MOVE (SOURCE, DESTINATION, LENGTH)
13 #OFFSET (PVA): INTEGER
14 #PREVIOUS_SAVE_AREA: POINTER
15 #PROGRAM_ERROR
16 #PTR: (DISP, BASE_POINTER):  CELL
17 #PURGE_BUFFER (OPTION, ADDRESS)
18 #READ_REGISTER (REGID): REGISTER_VALUE
19 #REAL_MEMORY_ADDRESS (PVA,RMA)
20 #REL (POINTER, BASE_POINTER): INTEGER
21 #RING (PVA): 0 .. 15
22 # RESTORE_TRAPS (OLD_TE)
23 #SCAN (SELECT, STRING, INDEX, FOUND)
24 #SEGMENT (PVA): )..4995
25 #STORE_BIT (BIT_VALUE, BIT_VARIABLE)
26 #TEST_ALTER_CONDITION_REG (SELOPT, BITNUM, BRANCH_EXIT)
27 #TEST_SET_BIT (BIT_VARIABLE, PREVIOUS_VALUE)
28 #TRANSLATE (TABLE, SOURCE, DESTINATION)
29 #WRITE_REGISTER (REGID, REGISTER_VALUE)
```

Note: see Internal Interface

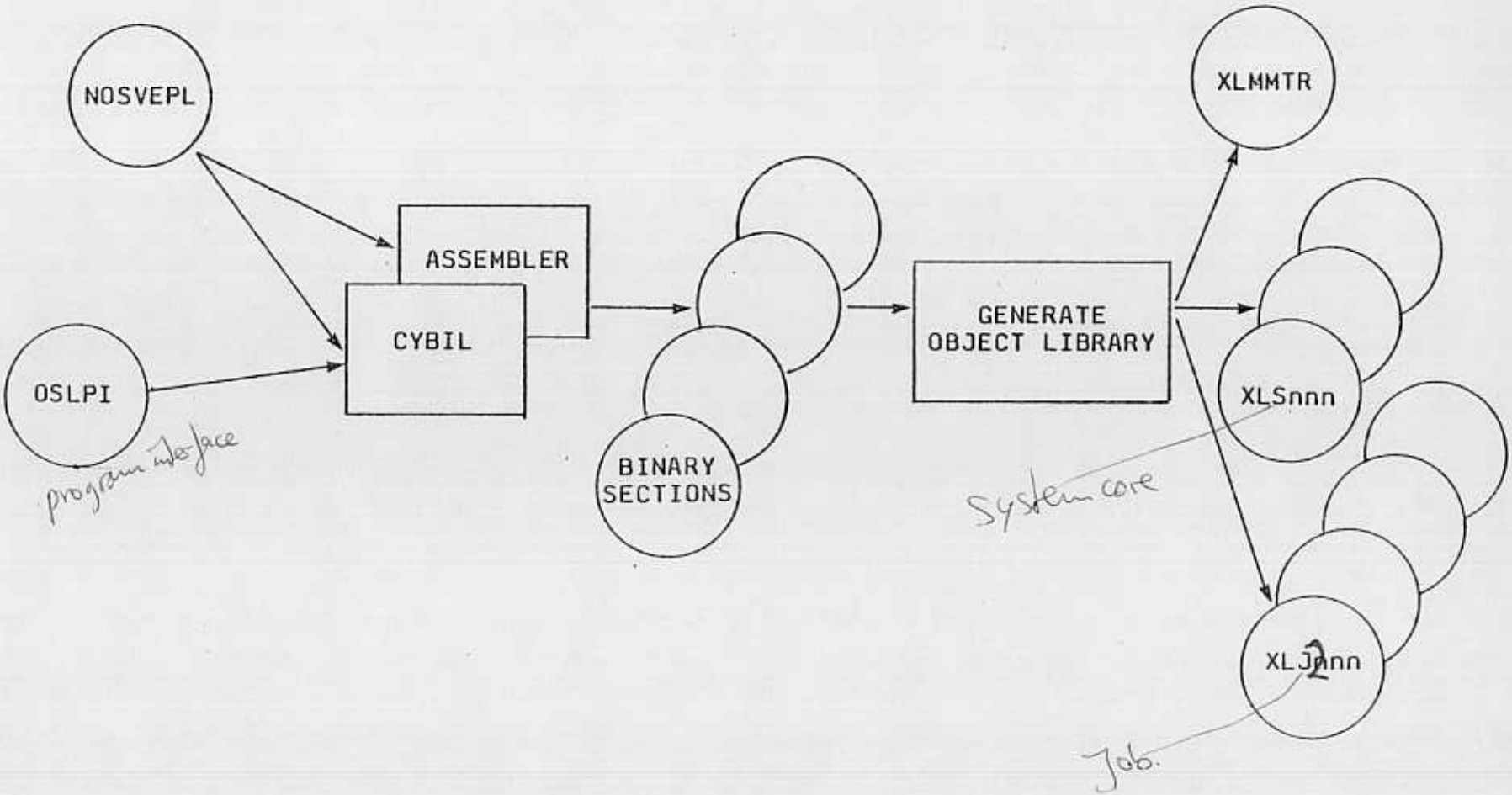
SYSTEM INITIALIZATION PROCESS

1. Library Generation

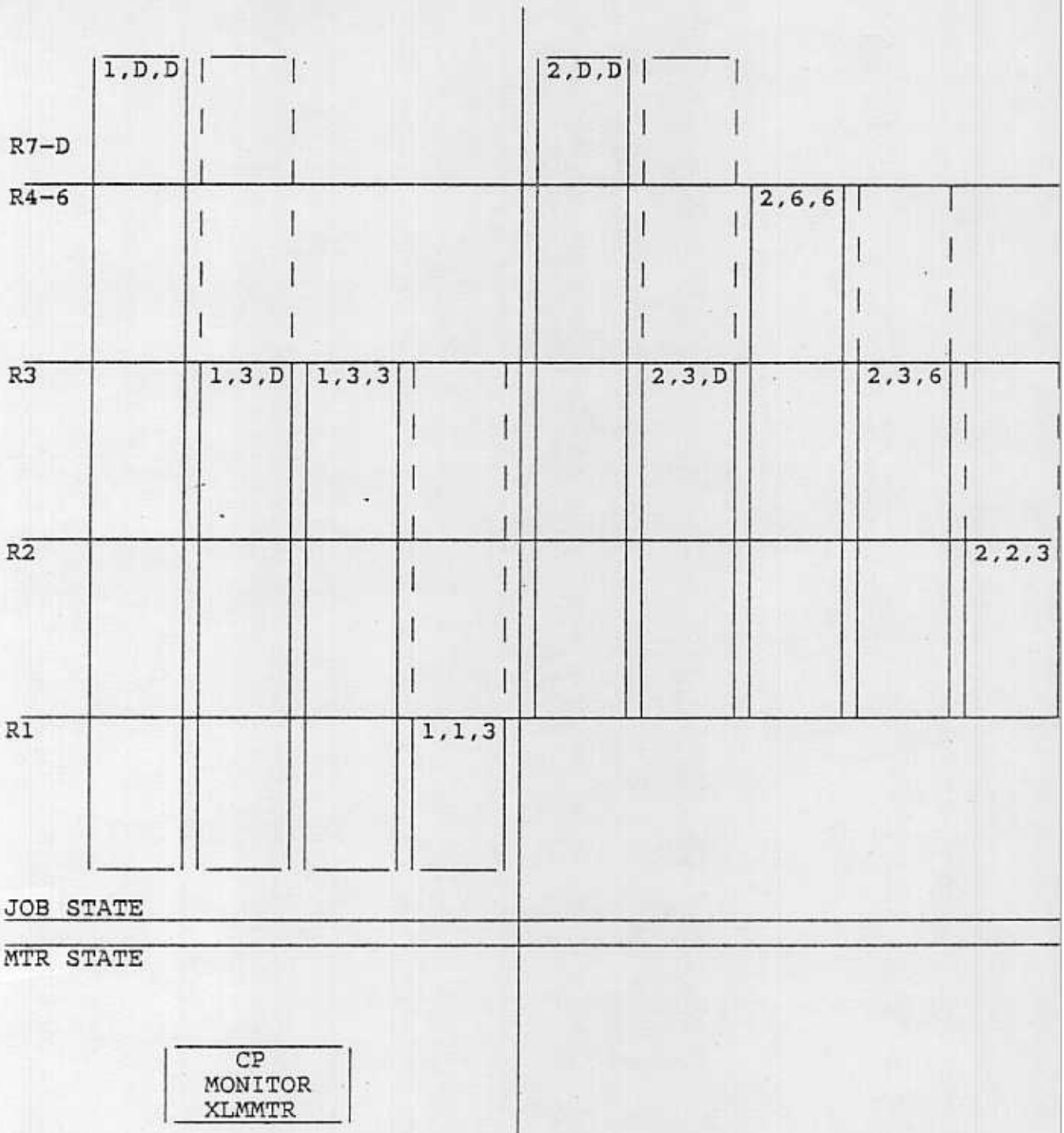
2. System Generation

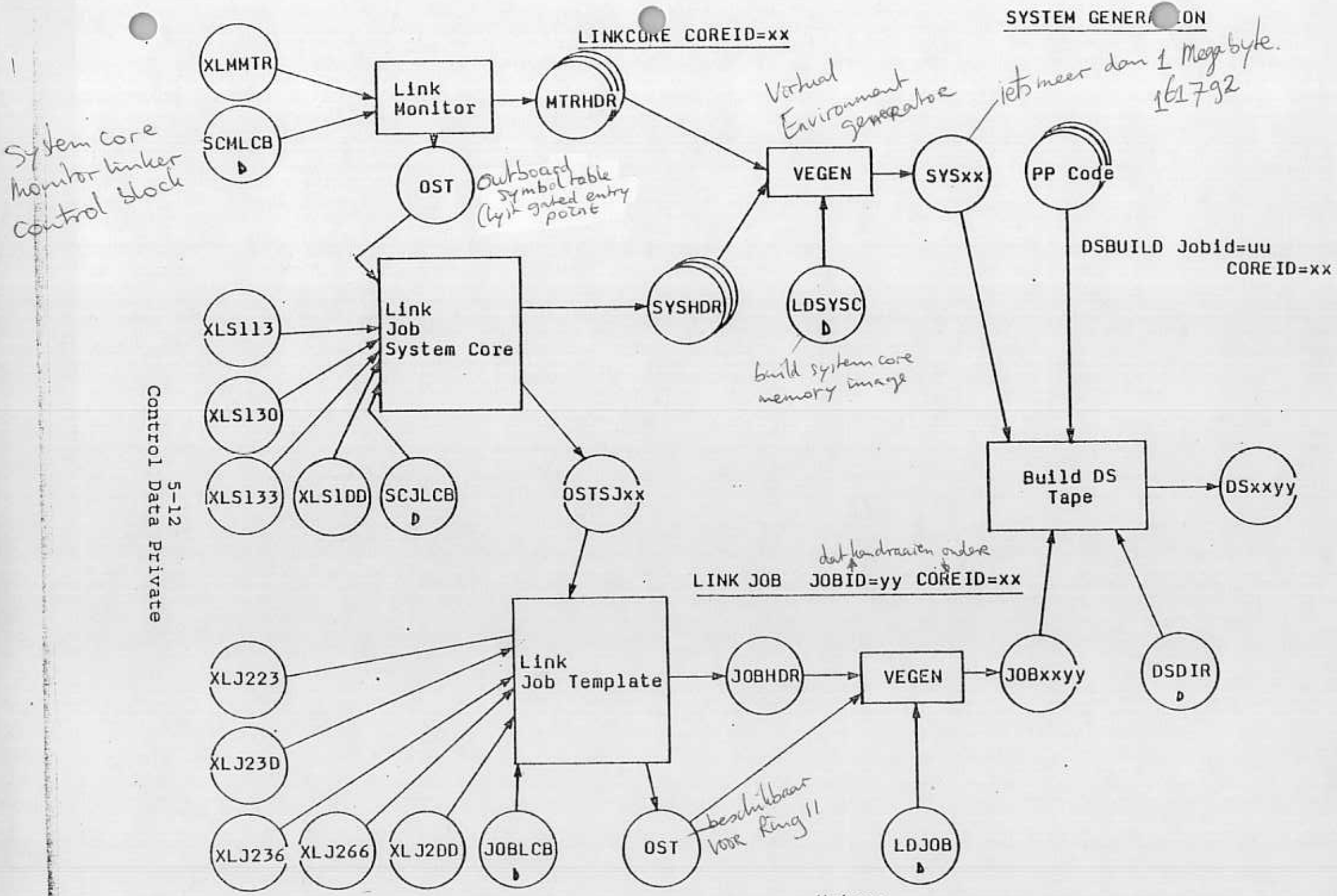
3. System Initialization

GENERATE LIBRARY



CODE SEGMENTS



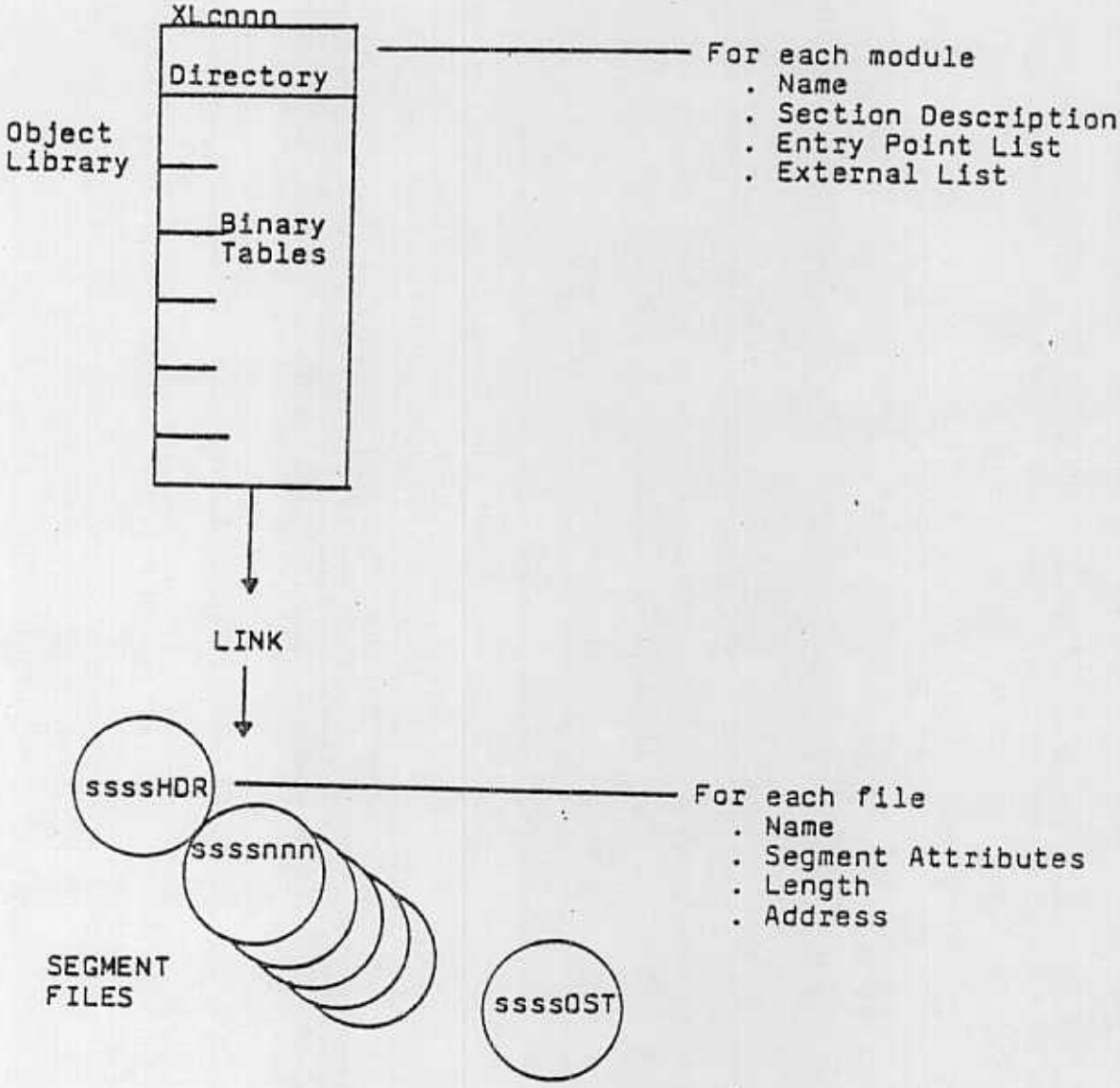

SYSTEM GENERATION
LINKCORE COREID=xx
XLMMTR
SCMLCB
D
Link Monitor
MTRHDR
System core monitor linker control block
OST
outboard symbol table (by it gated entry point
Virtual Environment generator
VEGEN
SYSxx
iets meer dan 1 Mega byte. 161792
PP Code
DSBUILD Jobid=uu COREID=xx
XLS113
XLS130
XLS133
XLS1DD
SCJLCB
D
Link Job System Core
SYSHDR
LDSYSC
D
build system core memory image
OSTSJxx
Build DS Tape
DSxxyy
LINK JOB JOBID=yy COREID=xx
dat handraien onder
XLJ223
XLJ23D
XLJ236
XLJ266
XLJ2DD
JOBLCB
D
Link Job Template
JOBHDR
OST
beschikbaar voor Ring 11
VEGEN
LDJOB
D
JOBxxyy
DSDIR
D
Notes:
(D) - directive file.

FILE DESCRIPTIONS

NAME	AREA	TYPE	COMMENTS
NOSVEPL		PL	Contains all code & data source for NOS/VE except program interface.
OSLPI		PL	Program Interface
XLMMTR	Monitor	Object Lib.	Library of monitor modules
XLSnnn	System Core	Object Lib.	Library to run in rings (n,n,n) Task monitor.
XLJnnn	Job Template	Object Lib.	Library to run in rings (n,n,n) Task services.
SCMLCB	Monitor	Directives	System Core/Monitor State Linker Control Block
SCJLCB	System Core	Directives	System Core/Job State Linker Control Block
JOBLCB	Job Template	Directives	Linker Control Block
OST			Outboard Symbol Table. List of gated entry points.
OSTSJxx	System Core	OST	System Core/Job State OST. System with id=xx.
MTRHDR	Monitor	Segment Files	This HDR describes a collection of "seed" files with names MTR101, MTR102, etc.
SYSHDR	System Core	Segment Files	This HDR file describes a collection of "seed" files with names SYS101, SYS102, etc.
JOBHDR	Job Template	Segment Files	This HDR file describes a collection of "seed" files with names JOB101, JOB102, etc.

FILE DESCRIPTIONS
(Continued)

NAME	AREA	TYPE	COMMENTS
LDSYSC	System Core	Directives	Load Directives for use by the Virtual Environment Generator (VEGEN) to build the system core memory image.
LDJOB	Job Template	Directives	Load Directives for use by VEGEN to build a job template memory image.
SYSxx	System Core	Memory Image	This core is sufficient to initialize a system with id=xx in 1M bytes.
JOBxxyy	Job Template	Memory Image	This template will run under the system with id=xx. The id of the template is yy.
PP Code			Set of peripheral drivers to run in the PPs.
DSDIR		Directives	Control the building of the DS Tape.
DSxxyy		DS Tape	


XLcnnn
Directory
Object
Library
Binary
Tables
For each module
. Name
. Section Description
. Entry Point List
. External List
LINK
ssssHDR
ssssnnn
SEGMENT
FILES
ssssOST
For each file
. Name
. Segment Attributes
. Length
. Address

LOAD MODULE

```
MODULE  = CLP$VARIABLE_ACCESS_MANAGER        LANGUAGE = CYBIL
FILE    = XLJ2DD      1982/09/18     12:11:52
-------------------------------------------------------------------------
WORKING STORAGE - RE_2DD
READ                                    398    2 01D 00053988    (2,D,D)
BINDING - RB_XXX
READ BINDING                            210    2 01B 00018840    (2,D,D)
WORKING STORAGE - OSS$JOB_PAGED_LITERAL
READ                                     AA    2 00B 0001DE60    (2,D,D)
WORKING STORAGE - CLS$PDT
READ                                     4C    2 00B 0001DF10    (2,D,D)
WORKING STORAGE - CLS$PDT_PARAMETERS
READ                                    360    2 00B 0001DF60    (2,D,D)
WORKING STORAGE - CLS$PDT_NAMES_AND_DEFAULTS
READ                                    24C    2 00B 0001E2C0    (2,D,D)
CODE - RE_2DD                                    - CLP$DECLARE_VARIABLE_COMM
READ EXECUTE                            920    2 01D 00053D20    (2,D,D)
CODE - RE_2DD                                    - CLP$REMOVE_VARIABLE_COMMA
READ EXECUTE                            200    2 01D 00054640    (2,D,D)
CODE - RE_2DD                                    - CLP$READ_VARIABLE
READ EXECUTE                            230    2 01D 00054840    (2,D,D)
CODE - RE_2DD                                    - CLP$WRITE_VARIABLE
READ EXECUTE                            118    2 01D 00054A70    (2,D,D)
CODE - RE_2DD                                    - CLP$PROCESS_ASSIGNMENT
READ EXECUTE                            398    2 01D 00054888    (2,D,D)
CODE - RE_2DD                                    - CLP$DEREFERENCE_VARIABLE
READ EXECUTE                            208    2 01D 00054F20    (2,D,D)
CODE - RE_2DD                                    - CLP$DEREFERENCE_NAME
READ EXECUTE                             90    2 01D 00055128    (2,D,D)
CODE - RE_2DD                                    - CLP$COMPLETE_VARIABLE_SCA
READ EXECUTE                            548    2 01D 00055188    (2,D,D)

ENTRY POINT DEFINITIONS                                ADDRESS
  CLP$DECLARE_VARIABLE_COMMAND                    2 01D 00053D20
  CLP$REMOVE_VARIABLE_COMMAND                     2 01D 00054640
  CLP$READ_VARIABLE                       GATED   2 01D 00054840
  CLP$WRITE_VARIABLE                      GATED   2 01D 00054A70
  CLP$PROCESS_ASSIGNMENT                          2 01D 00054888
  CLP$DEREFERENCE_VARIABLE                        2 01D 00054F20
  CLP$DEREFERENCE_NAME                            2 01D 00055128
  CLP$COMPLETE_VARIABLE_SCAN                      2 01D 00055188
EXTERNAL ENTRY POINTS REFERENCED
  CLP$EXPRESSION_SCANNER                  CLP$TEST_RANGE
  CLP$REFERENCE_VARIABLE                  CLP$CONVERT_INTEGER_TO_STRING
  CLP$BALANCE_PARENTHESES                 CLP$CONVERT_TO_OST$STATUS
  CLP$INTERNAL_CREATE_VARIABLE            CLP$INITIALIZE_VARIABLE
  CLP$ASSIGN_VARIABLE_VALUE               CLP$CHECK_NAME_FOR_BOOLEAN
  CLP$CONVERT_STRING_TO_NAME              CLP$CREATE_VARIABLE
  CLP$SCAN_EXPRESSION                     CLP$GET_SET_COUNT
  CLP$GET_VALUE                           CLP$DELETE_VARIABLE
  CLP$REWRITE_VARIABLE                    CLP$SCAN_PARAMETER_LIST
```

SEGMENT DESCRIPTION
PART 1

```
SES/C180 LINKER OUTPUT
- PRIMARY ENTRY POINT = PMP$TASK_BEGIN
FILE NAME/                                        LOAD/              RING
  SECTION NAMES
-------------------------------------------------------------------------
PJBX101
READ WRITE                                  1D00  * 004 00000000   (2,3,3)
  OSS$JOB_PAGEABLE

PJBX102
READ WRITE                                  3AE4  * 005 00000000   (3,D,D)
  OSS$TASK_PRIVATE

PJBX103
READ WRITE                                  1078  * 006 00000000   (3,D,D)
  OSS$TASK_SHARED

PJBX104
READ WRITE                                   318  * 007 00000000   (B,B,B)
  OSS$TASK_PRIVATE_RING_11

PJBX105
READ EXECUTE-LOCAL PRIV                     58E88  * 01A 00000000   (2,2,3)
  RE_223

PJBX106
BINDING                                     1FD02  * 01B 00000000   (2,D,D)
  BINDING

PJBX107
READ EXECUTE-LOCAL PRIV                     BAD50  * 01C 00000000   (2,3,D)
  RE_23X

PJBX108
READ EXECUTE-LOCAL PRIV                     A89F6  * 01D 00000000   (2,D,D)
  RE_2DD

PJBX109
READ EXECUTE                                17750  * 01E 00000000   (B,B,B)
  RE_BBB

PJBX11D
READ                                        63E88  * 00B 00000000   (2,D,D)
  OSS$MESSAGE_TEMPLATE_STRINGS
```

SYSTEM INITIALIZATION

1. Deadstart (x.NVEffff)
   System device established
   Memory partitioned
   Page table built
   System Core copied to Real Memory
   NOS exchanges to Monitor (Dispatcher)

2. System Job
   Configuration established
   Job Template established
   System Tasks started
   - Job scheduler & terminator
   - Operator Communication
   - Memory Link
   - Interactive Executive & Remote Host

3. User Job

COMMUNICATION

MECHANISMS

Call
Return
Exchange Jump
Exchange Interrupt
Trap Interrupt

PROCESSORS

Monitor Interrupt Processor (MIP)
    Request Processors

Trap Handler
    Signal Handler
    System Flag Handler
    Monitor Fault Handlers

STACK DATA MAPPING

1. SFSA-stack frame save area

    - o Typically words 0-4. length in word 2
    - o See diagram and CYBIL definition

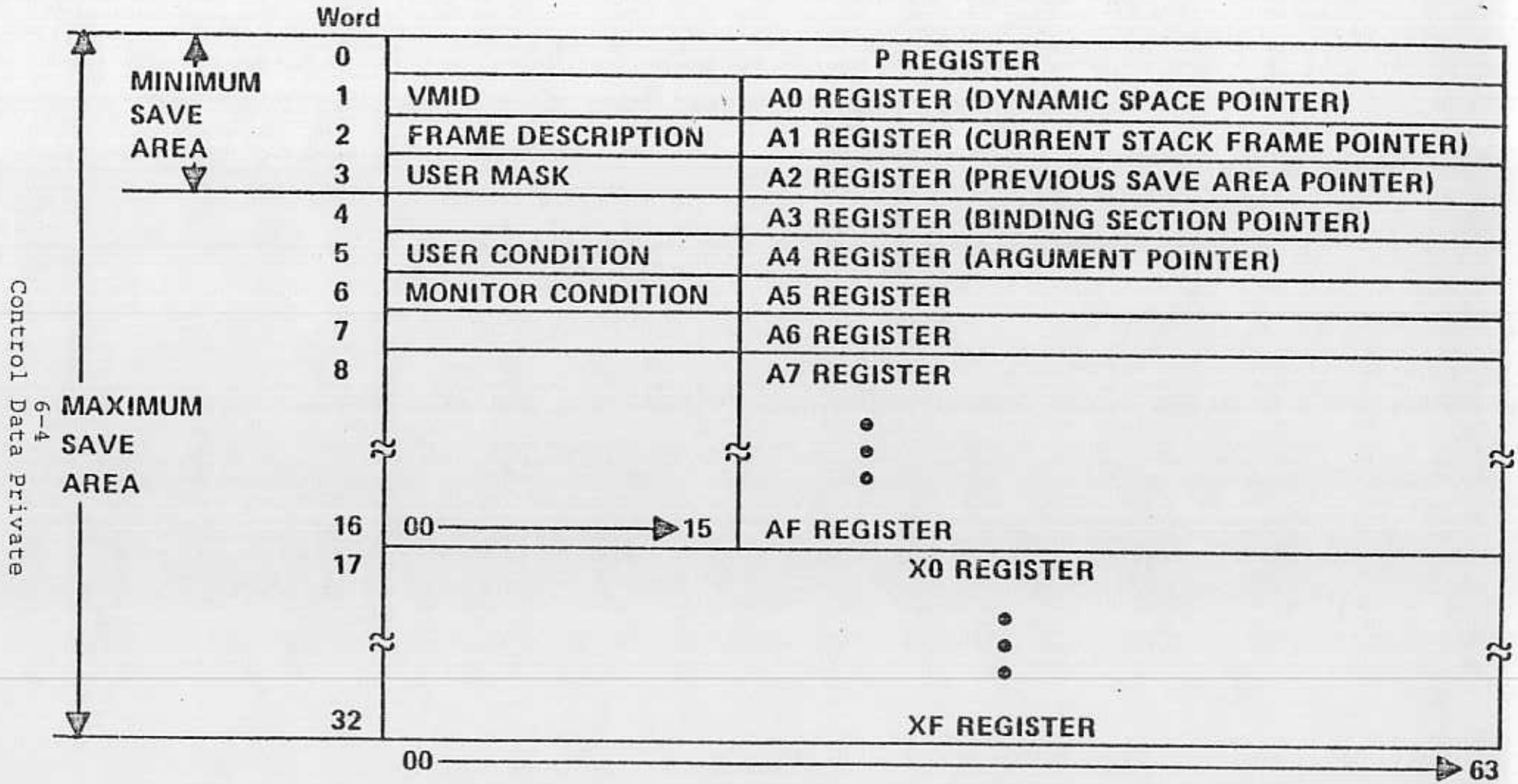
2. Automatic Variables

    - o First two words not used
    - o Each variable starts a new word
    - o Array and record components are byte aligned unless packed
    - o Packed components are bit aligned except characters, integers, and pointers

3. Parameters

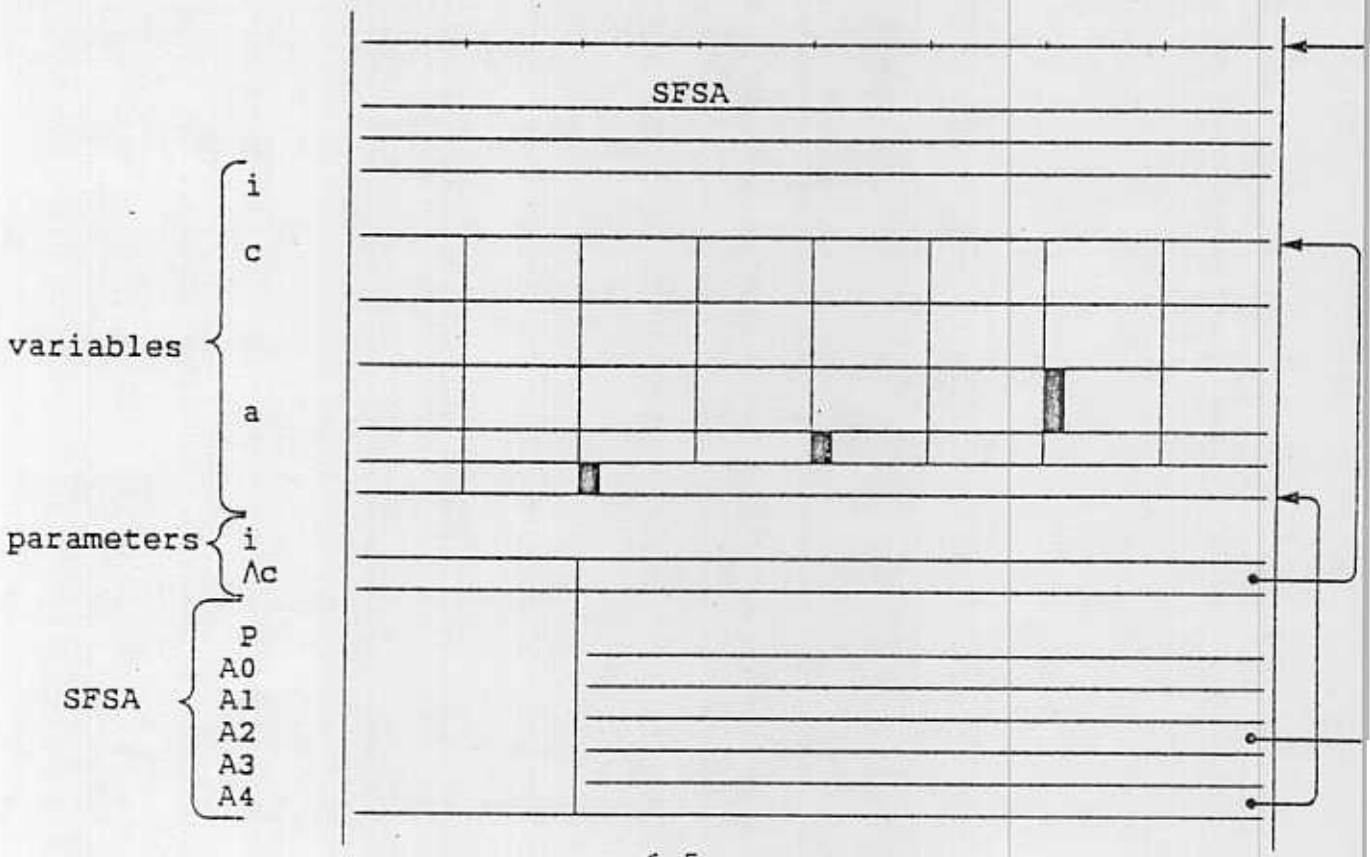
    - o Each parameter starts a new word
    - o VAR parameters are passed as pointers
    - o The pointer to the parameters is in A4

STACK FRAME SAVE AREA



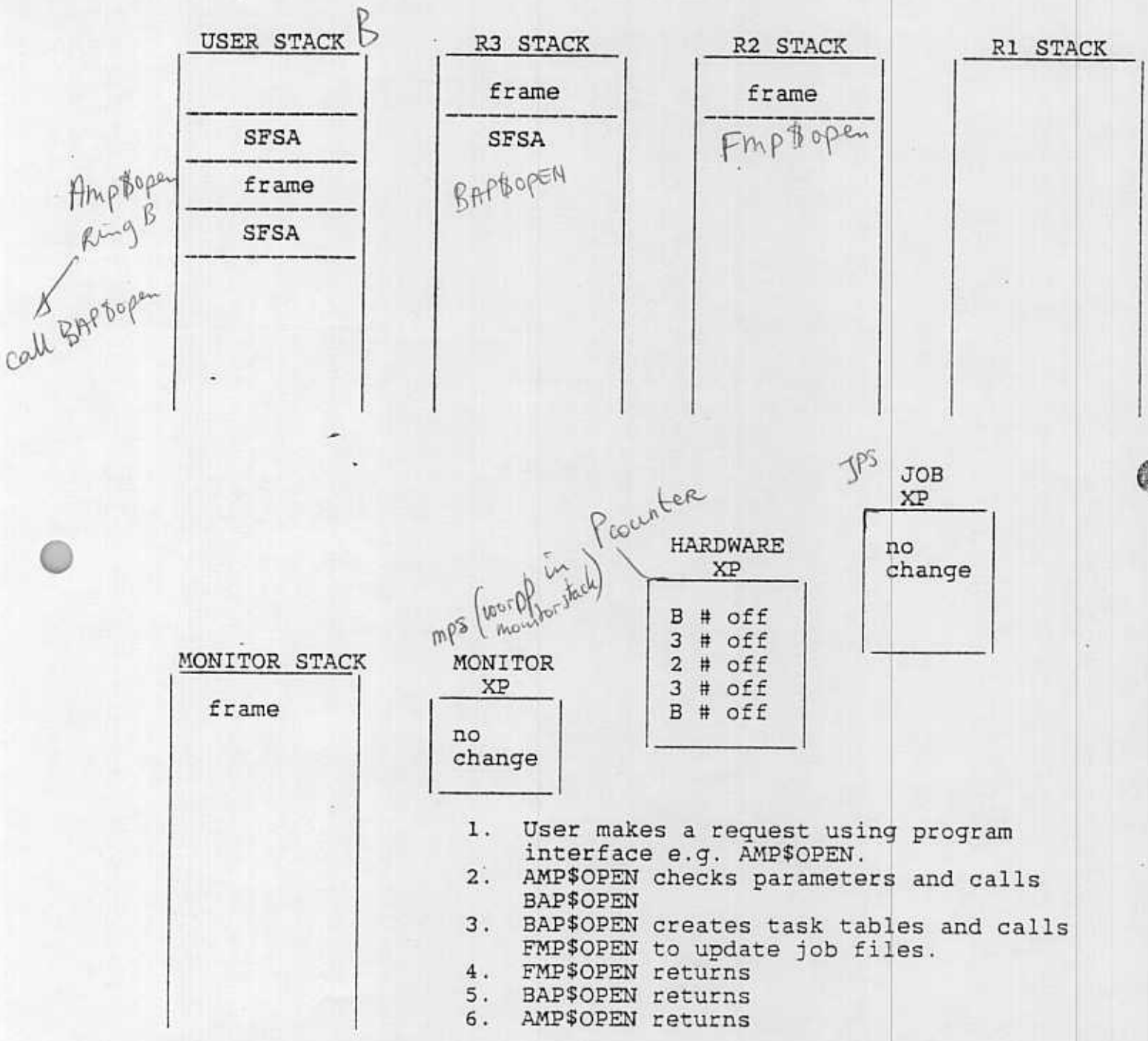
DATA MAPPING
EXAMPLE

```
MODULE x;
   PROCEDURE ABC,
     VAR
       i: 1..10,
       c: string(10),
       a: array  1..3  of string(6),
                  .
                  .
                  .
     XYZ(i,c);
                  .
                  .
                  .

   PROCEND ABC;

   PROCEDURE XYZ (i:1..10; VAR s:string(10));
                  .
                  .
                  .
```



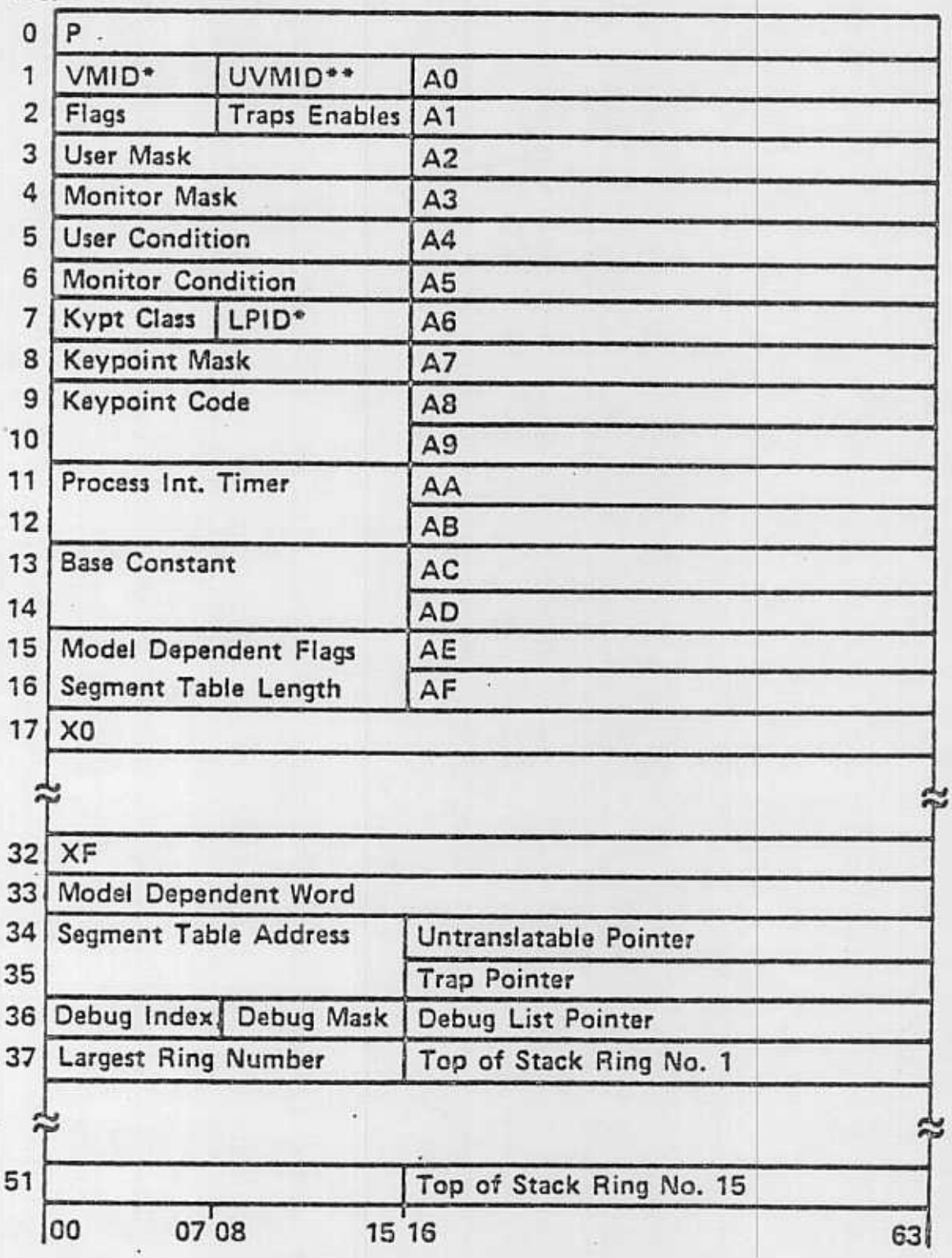
TRANSFER OF CONTROL

AMP$OPEN



1. User makes a request using program interface e.g. AMP$OPEN.
2. AMP$OPEN checks parameters and calls BAP$OPEN
3. BAP$OPEN creates task tables and calls FMP$OPEN to update job files.
4. FMP$OPEN returns
5. BAP$OPEN returns
6. AMP$OPEN returns

EXCHANGE PACKAGE



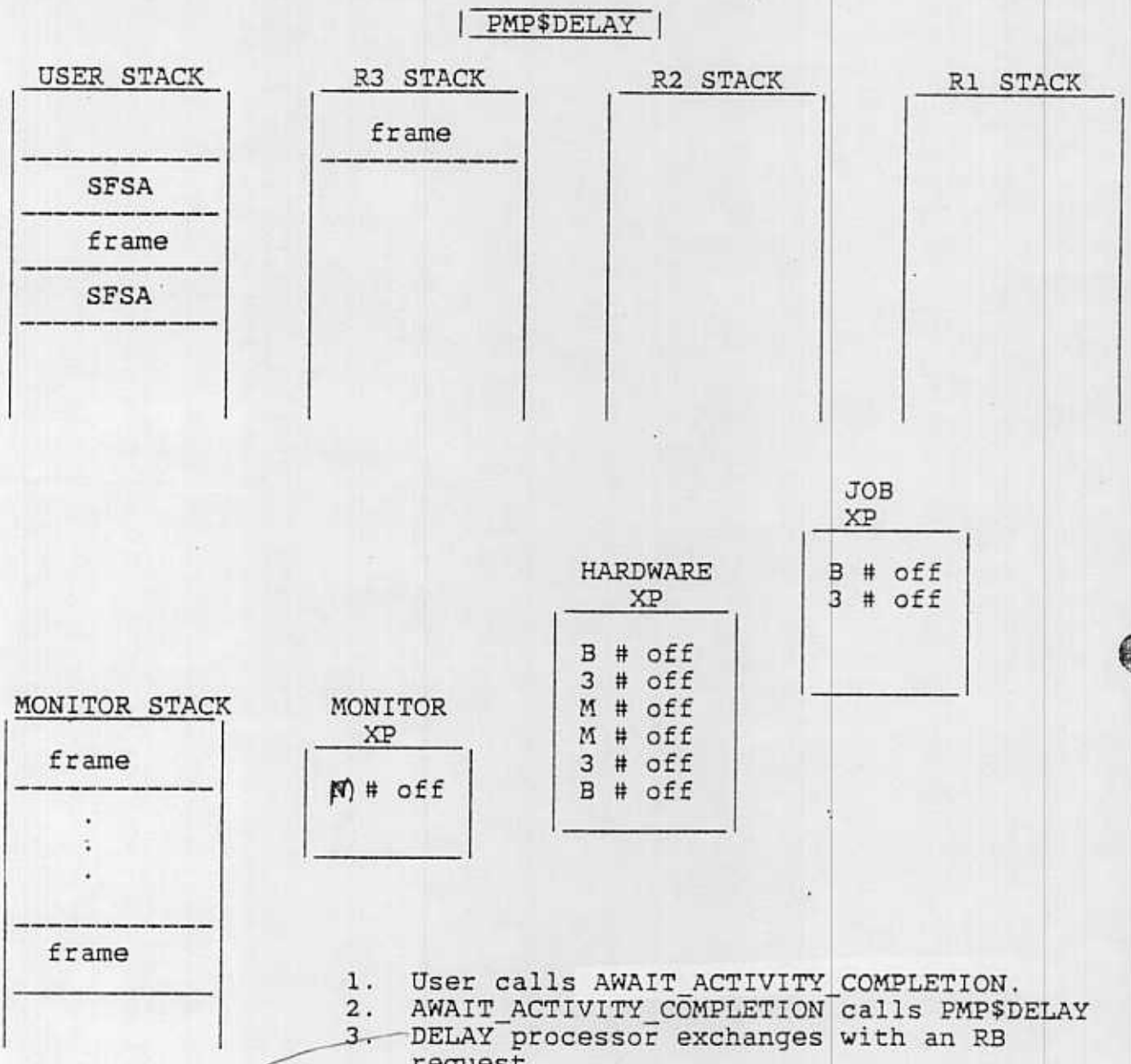
* Virtual Machine Identifier

** Untranslatable Virtual Machine Identifier

*** Last Processor Identification

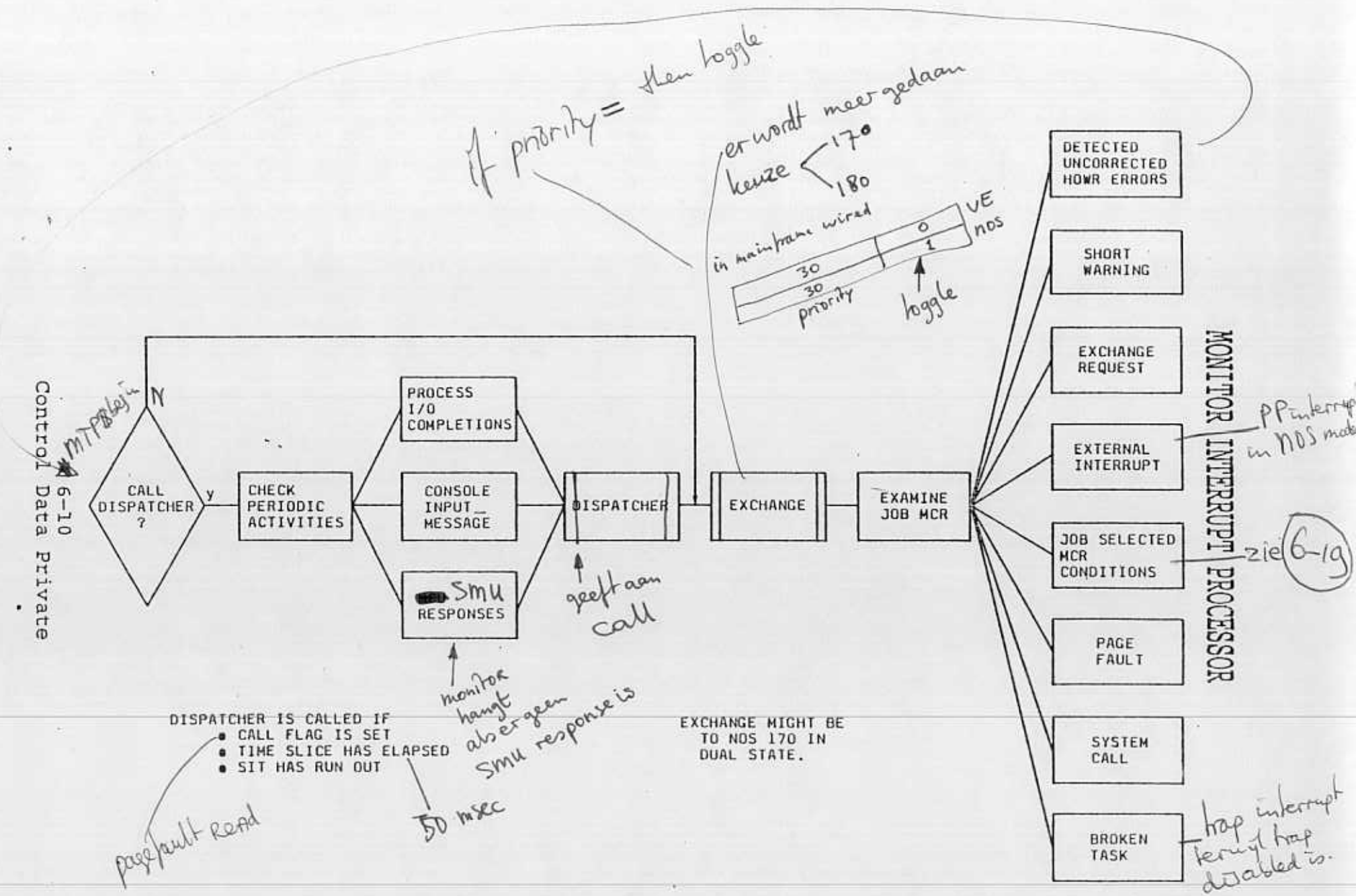
CONTROL DATA PRIVATE

TRANSFER OF CONTROL

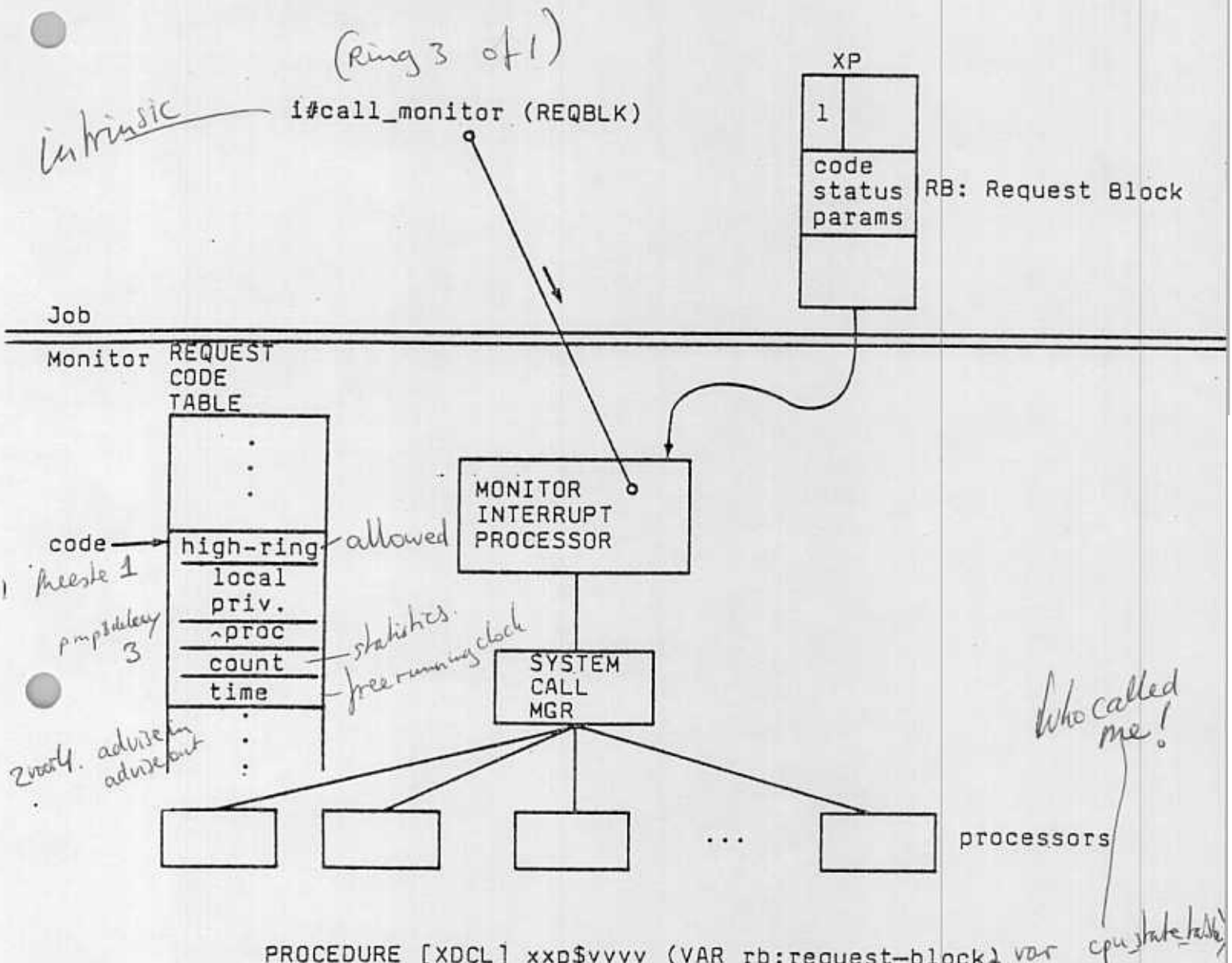


1. User calls AWAIT_ACTIVITY_COMPLETION.
2. AWAIT_ACTIVITY_COMPLETION calls PMP$DELAY
3. DELAY processor exchanges with an RB request.
4. Monitor Interrupt processor delays the task.
5. Monitor returns to the task.
6. Delay processor returns.
7. AWAIT_ACTIVITY_COMPLETION returns.

Time	Hardware XP	R11 Stack	R3 Stack	R2 Stack	R1 Stack	Monitor Stack		Task XP Area	Monitor XP Event Area	Event: Program Interface Request
T_0	B# n	B.frame$_1$	empty	empty	empty	M.frame$_1$		Program Starter	MIP	
T_1	3# n	SFSA	3.frame$_1$							P.I. request eg AMP$ RETURN
T_2	1# n		SFSA		1.frame					internal interface I.I. request to update system file tables (R1)
T_3	3# n		3.frame$_1$							R1 returns
T_4	B# n	B.frame$_1$								R3 returns
T_5										Page Fault
T_6	M# n					M.frame$_1$		Interrupt user B# n		Page fault (on READ)
T_7	"					SFSA M.frame$_2$				Call interrupt handler
T_8	"					M.frame$_1$				Return From Page Manager
T_9	"					SFSA M.frame$_3$				Call Dispatcher
T_{10}	"					M.frame$_1$				Return from Dispatcher
T_{11}	?									Exchange to new task (changes JPS)
T_{12}										⋮
T_{13}	B# n					M.frame$_1$				Exchange to this task


if priority = then toggle
er wordt meer gedaan
keuze 170
180
in maintane wired
VE
NOS
0
1
30
30
priority
toggle
MTPB bij
N
CALL DISPATCHER ?
y
CHECK PERIODIC ACTIVITIES
PROCESS I/O COMPLETIONS
CONSOLE INPUT_ MESSAGE
SMU RESPONSES
DISPATCHER
geeft aan call
EXCHANGE
EXAMINE JOB MCR
DETECTED UNCORRECTED HOWR ERRORS
SHORT WARNING
EXCHANGE REQUEST
EXTERNAL INTERRUPT
JOB SELECTED MCR CONDITIONS
PAGE FAULT
SYSTEM CALL
BROKEN TASK
MONITOR INTERRUPT PROCESSOR
PP interrupt in NOS mode
zie 6-19
monitor hangt als er geen SMU response is
DISPATCHER IS CALLED IF
• CALL FLAG IS SET
• TIME SLICE HAS ELAPSED
• SIT HAS RUN OUT
50 msec
page fault read
EXCHANGE MIGHT BE TO NOS 170 IN DUAL STATE.
trap interrupt termyl trap disabled is.

SYSTEM CALL PROCESSING



MCR → system call bit set in XP (left in X_0)

JPS

X_2 | XF ↑ parameter

X_1 = Status Reply

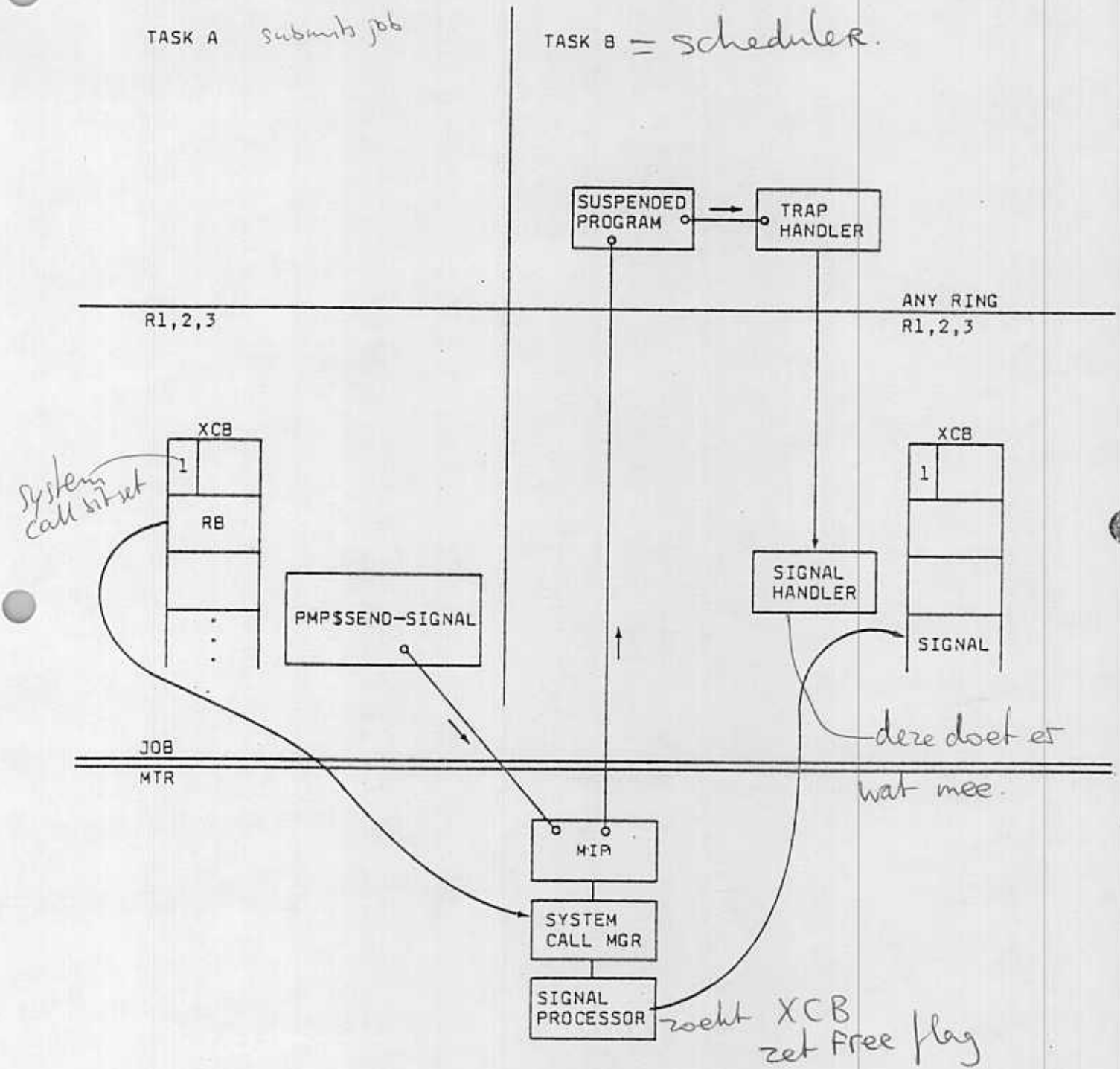
REQUEST BLOCK

```
MEMORY MANAGER REQUEST DEFINITIONS
  1 MMT$RB_ADVISE
  2 MMT$RB_ASSIGN_FLAWED_MEMORY
  3 MMT$RB_ASSIGN_REAL_PAGE
  4 MMT$RB_FLAW_PAGE
  5 MMT$RB_FREE_FLUSH
  6 MMT$RB_UNFLAW_PAGE
TASK MANAGER REQUEST DEFINITIONS
  1 TMT$RB_INITIATE_JOB
  2 TMT$RB_INITIATE_TASK
  3 TMT$RB_CYCLE
  4 TMT$RB_DELAY
  5 TMT$RB_EXIT_JOB
  6 TMT$RB_EXIT_TASK
  7 TMT$RB_SEND_SIGNAL
  8 TMT$RB_WAIT_SIGNAL
  9 TMT$RB_CHANGE_SEGMENT_TABLE
```

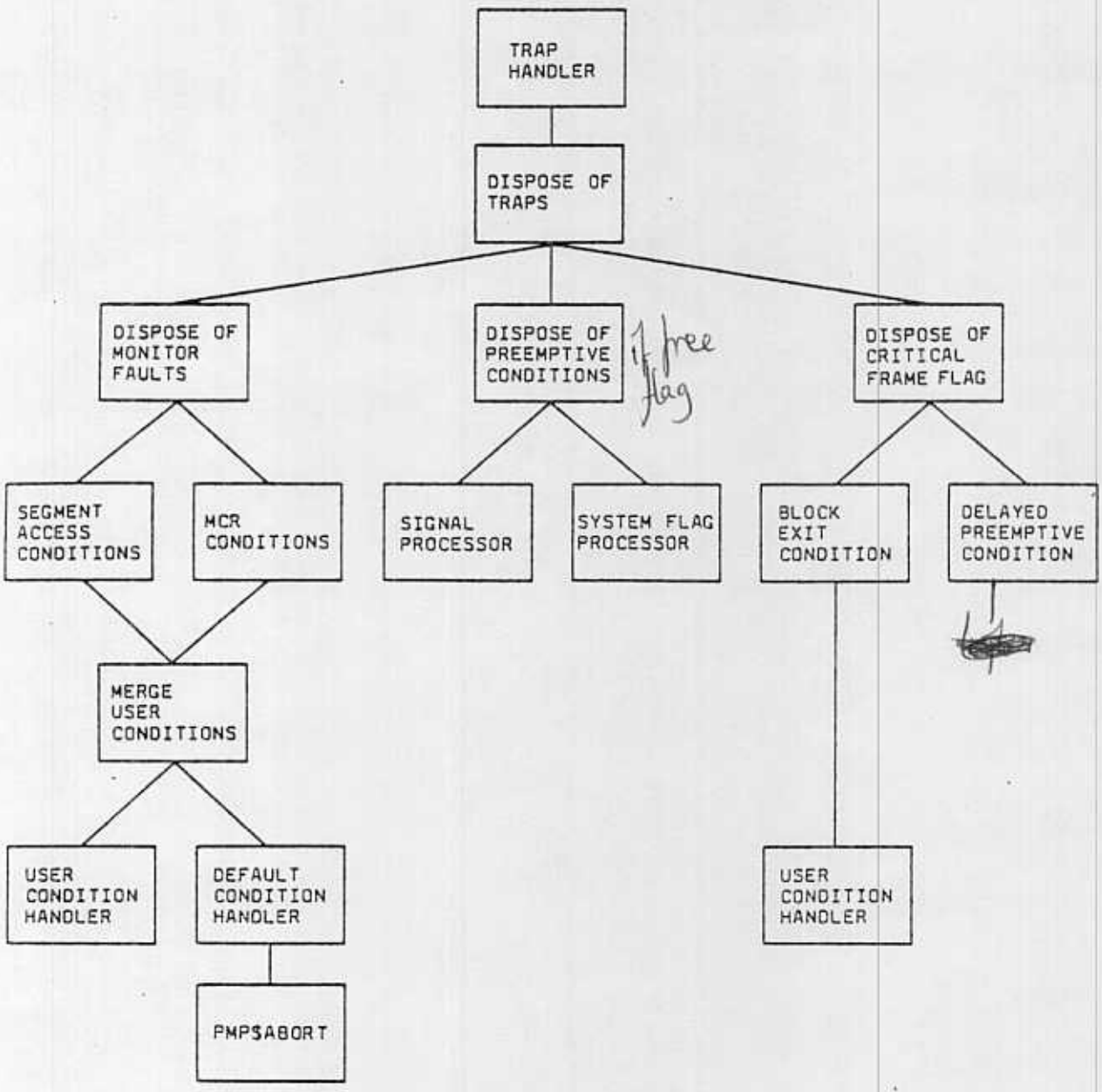
Note: see Internal Interface.

Time	Hardware XP	R11 Stack	R3 Stack	R2 Stack	R1 Stack	Monitor Stack		Task XP Area	Monitor XP Event	Event **Access Violation**
T_0	B# n_1	B.frame$_1$	empty	empty	empty	M.frame$_1$		Program starter	MIP	
T_1	M# n_1							B# n_1		Access Violation
T_2	"									Set free flag in UCR
T_3	B# n_1								MIP	Exchange
T_4	B# n_2	SFSA B.frame$_2$								Trap handler—XCB.UCR
T_5	3# n_1	SFSA	3.frame$_1$							on condition not set Call default Condition Handler
T_6	3# n_2		SFSA 3.frame$_2$							Call PMP$ABORT
T_7	3# n_2	X								Pop R11 stack — see if post-processing needed
T_8	M# n_1							3# n_2		Request Task Termination
T_9	M# n_2		X	X	X	SFSA M.frame$_2$		X		Call System Call processor
T_{10}						M.frame$_1$				Return after terminating task
T_{11}										Call dispatcher to pick new task
T_{12}										
T_{13}										

SIGNAL HANDLING



TRAP HANDLER (assembler)



TRAP HANDLING

- Trap Handler runs at the ring of the interrupted program.
- Dispose-of-traps runs in ring 1.
- Dispose-of-traps checks the reason for entry in this order:
  1. Monitor Faults
     Segment Access Conditions, MCR conditions and User conditions are merged together. If a user condition handler exists it will be found in the stack. The default condition handler will abort the task in all cases.
  2. Preemptive Conditions
     All signals and flags will be processed if the free flag is set. If the ring of execution is lower than the recognition ring, the critical frame flag will be set in the first stack frame above the recognition ring. In all other cases the signal or flag handler will be called.
  3. Critical Frame Flag
     If the critical frame flag indicates a delayed signal or flag it will be resolved by dispose-of-preemptive-conditions. If the user has established a block exit handler, the handler will be found in the stack frame and called.
- A Daley diagram of these modules is included in the chapter on program management.

SIGNAL PROCEDURES

not externalized

SEND SIGNAL

```
PMP$SEND_SIGNAL(recipient,signal,status)
```

SIGNAL HANDLER

```
ppP$HANDLE_SIGNAL_xxx(originator,signal)
```

DEFINE HANDLER (for test only)

```
PMP$DEFINE_SIGNAL_HANDLER(id,handler,recog_ring,status)
```

SIGNALS

Memory Link	MLP$ handle_signal interprets 'sub_signals' and calls a handler.
Interactive	IFP$hande_signal passes info between the interactive exec., job monitor, and user tasks.
Callend	PMP$child_terminator_handler.
Scheduler	JMP$_handle_gfm_ia_signal processes the signal from QF manager that interactive job has been routed.

Memory link
interactive
suspend child
scheduler

SYSTEM FLAGS

SET FLAG

```
PMP$SET_SYSTEM_FLAG(flag_id,recipient,status)
```

FLAG HANDLER

```
ppP$HANDLE_FLAG_xxx(flag_id)
```

DEFINE HANDLER (for test only)

```
PMP$DEFINE_SYSTEM_FLAG_HANDLER(id,handler,recog_ring,st)
```

FLAGS

Statistics	AVP$monitor_statistics_handler
Terminate	PMP$terminate_flag_handler
Drop	JMP$handle_drop_job_flag
Linked Signals	TMP$dispose_mainframe_signals This flag indicates that a signal occurred while the task was swapped.

MONITOR FAULTS

FAULT HANDLER

ppP$HANDLE_FAULT_xxx(fault,save_area)

DEFINE HANDLER (for test only)

PMP$DEFINE_MONITOR_FAULT(id,handler,status)

FAULTS

Instruction Specification Error
Address Specification Error
Access Violation
Environment Specification Error
Outward Call/Inward Return

SEGMENT ACCESS CONDITIONS

Read beyond EOI ~~xxxx~~ mass storage limit
Write beyond msl
Segment access error
Key lock violation
Ring violation
I/O read error

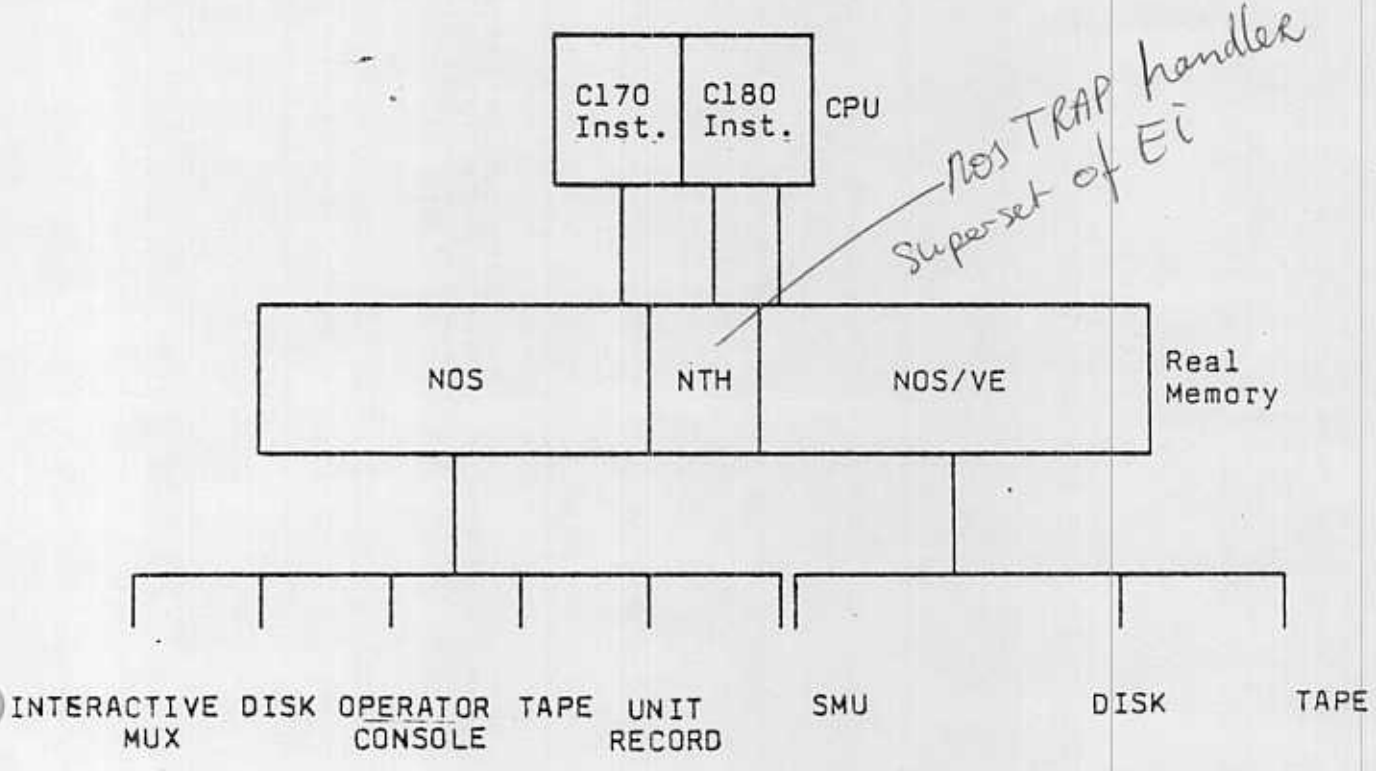
MONITOR CONDITION REGISTER

BIT NUMBER AND DEFINITION			TRAPS ENABLED		TRAPS DISABLED		
			TRAP ENABLE F/F SET AND TRAP ENABLE DELAY F/F CLEAR AND MASK BIT SET		TRAP ENABLE F/F CLEAR OR TRAP ENABLE DELAY F/F SET AND MASK BIT SET		MASK BIT CLEAR
			JOB MODE	MONITOR MODE	JOB MODE	MONITOR MODE	JOB OR MONITOR MODE
0	Processor Detected Malfunction	Mon	EXCH	TRAP	EXCH	HALT	HALT
1	Memory Detected Malfunction	Mon	EXCH	TRAP	EXCH	HALT	HALT
2	Power Warning	Sys	EXCH	TRAP	EXCH	STACK	STACK
3	· Instruction Specification Error	Mon	EXCH	TRAP	EXCH	HALT	HALT
4	· Address Specification Error	Mon	EXCH	TRAP	EXCH	HALT	HALT
5	Exchange Request	Sys	EXCH	TRAP	EXCH	STACK	STACK
6	· Access Violation	Mon	EXCH	TRAP	EXCH	HALT	HALT
7	· Environment Specification Error	Mon	EXCH	TRAP	EXCH	HALT	HALT
8	External Interrupt	Sys	EXCH	TRAP	EXCH	STACK	STACK
9	Page Table Search Without Find	Mon	EXCH	TRAP	EXCH	HALT	HALT
10	System Call	Status - This bit is a flag only and does not cause any hardware action.					
11	System Interval Timer	Sys	EXCH	TRAP	EXCH	STACK	STACK
12	Invalid Segment	Mon	EXCH	TRAP	EXCH	HALT	HALT
13	· Outward Call/Inward Return	Mon	EXCH	TRAP	EXCH	HALT	HALT
14	Soft Error Log	Sys	EXCH	TRAP	EXCH	STACK	STACK
15	Trap Exception	Status - This bit is a flag only and does not cause any hardware action.					

USER CONDITION REGISTER

BIT NUMBER AND DEFINITION			TRAPS ENABLED		TRAPS DISABLED		
			TRAP ENABLE F/F SET AND TRAP ENABLE DELAY F/F CLEAR AND MASK BIT SET		TRAP ENABLE F/F CLEAR OR TRAP ENABLE DELAY F/F SET AND MASK BIT SET		MASK BIT CLEAR
			JOB MODE	MONITOR MODE	JOB MODE	MONITOR MODE	JOB OR MONITOR MODE
0	Privileged Instruction Fault	Mon	TRAP	TRAP	EXCH	HALT	These mask bits are permanently set.
1	Unimplemented Instruction	Mon	TRAP	TRAP	EXCH	HALT	
2	Free Flag	User	TRAP	TRAP	STACK	STACK	
3	Process Interval Timer	User	TRAP	TRAP	STACK	STACK	
4	Inter-ring Pop	Mon	TRAP	TRAP	EXCH	HALT	
5	Critical Frame Flag	Mon	TRAP	TRAP	EXCH	HALT	
6	Keypoint	User	TRAP	TRAP	STACK	STACK	
7	Divide Fault	User	TRAP	TRAP	STACK	STACK	STACK
8	Debug	User	TRAP	TRAP	Debug bit will not set when traps disabled.		
9	Arithmetic Overflow	User	TRAP	TRAP	STACK	STACK	STACK
10	Exponent Overflow	User	TRAP	TRAP	STACK	STACK	STACK
11	Exponent Underflow	User	TRAP	TRAP	STACK	STACK	STACK
12	F. P. Loss of Significance	User	TRAP	TRAP	STACK	STACK	STACK
13	F. P. Indefinite	User	TRAP	TRAP	STACK	STACK	STACK
14	Arithmetic Loss of Significance	User	TRAP	TRAP	STACK	STACK	STACK
15	Invalid BDP Data	User	TRAP	TRAP	STACK	STACK	STACK

DUAL STATE CONFIGURATION



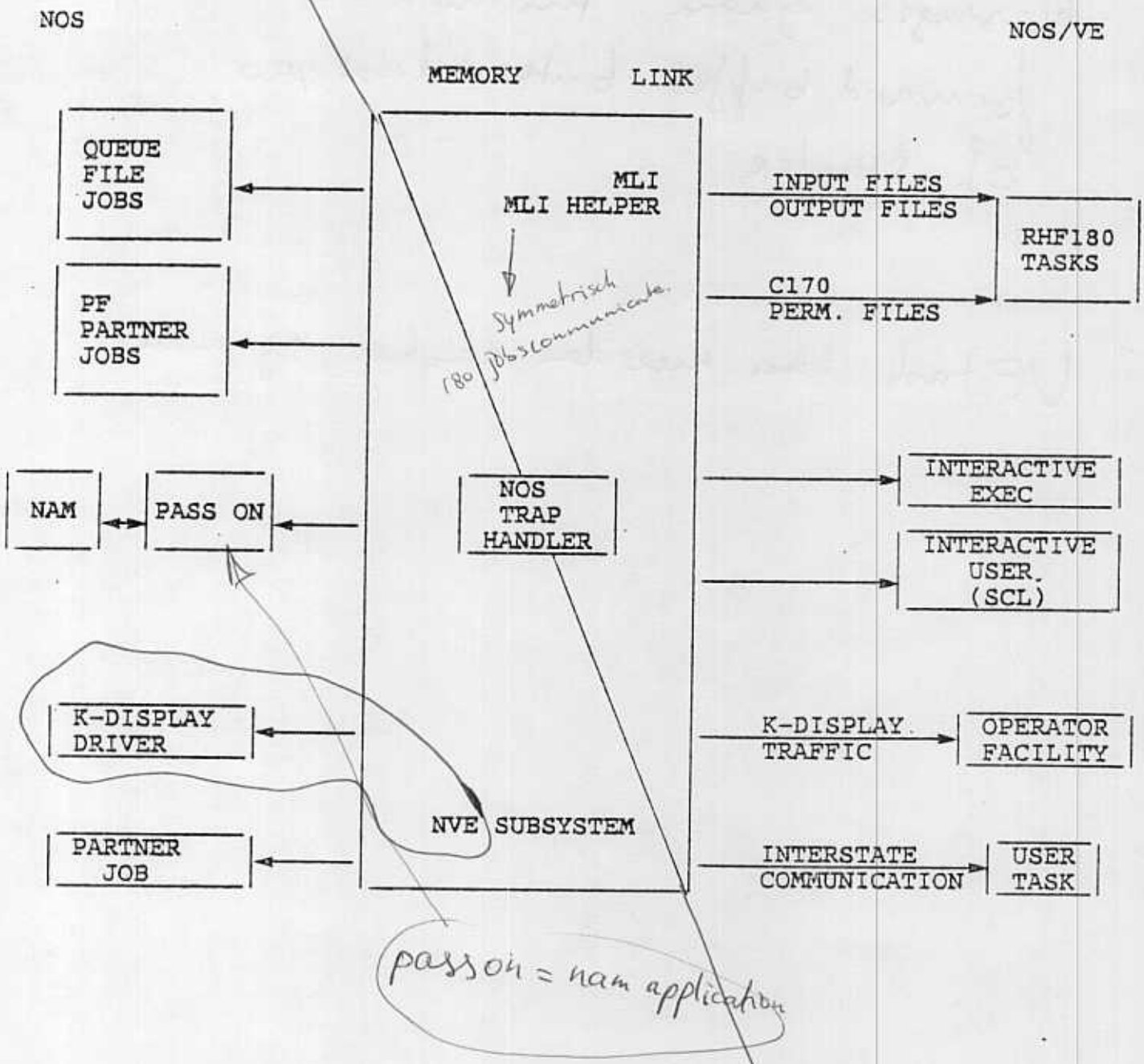
017 instr. van NOS → NTH

VIRTUAL ENVIRONMENT PARTITIONING

- The system resources are partitioned between CYBER 170 and CYBER 180 logical machines
- CPU is partitioned using the VMID field in the exchange package. Determines how the CPU will
  - Fetch and interpret instructions
  - Interpret the register file
  - Interpret interrupts
- CPU access to central memory
  - CYBER 170 addresses map into real memory addresses 0-N
  - CYBER 180 addresses map into (N+1) - (memory size-1)
- PPU access to central memory
  - PPUs are assigned to either 170 system or NOS/VE (dedicated)
  - IOU bounds register limits write access to CM
- Channels are software partitioned to access only CYBER 170 or CYBER 180 peripheral devices (except maintenance channel).

opbrengen/down.

LINK FACILITY



deze calls worden door een FAP gedaan

MEMORY LINK
INTERNAL INTERFACES

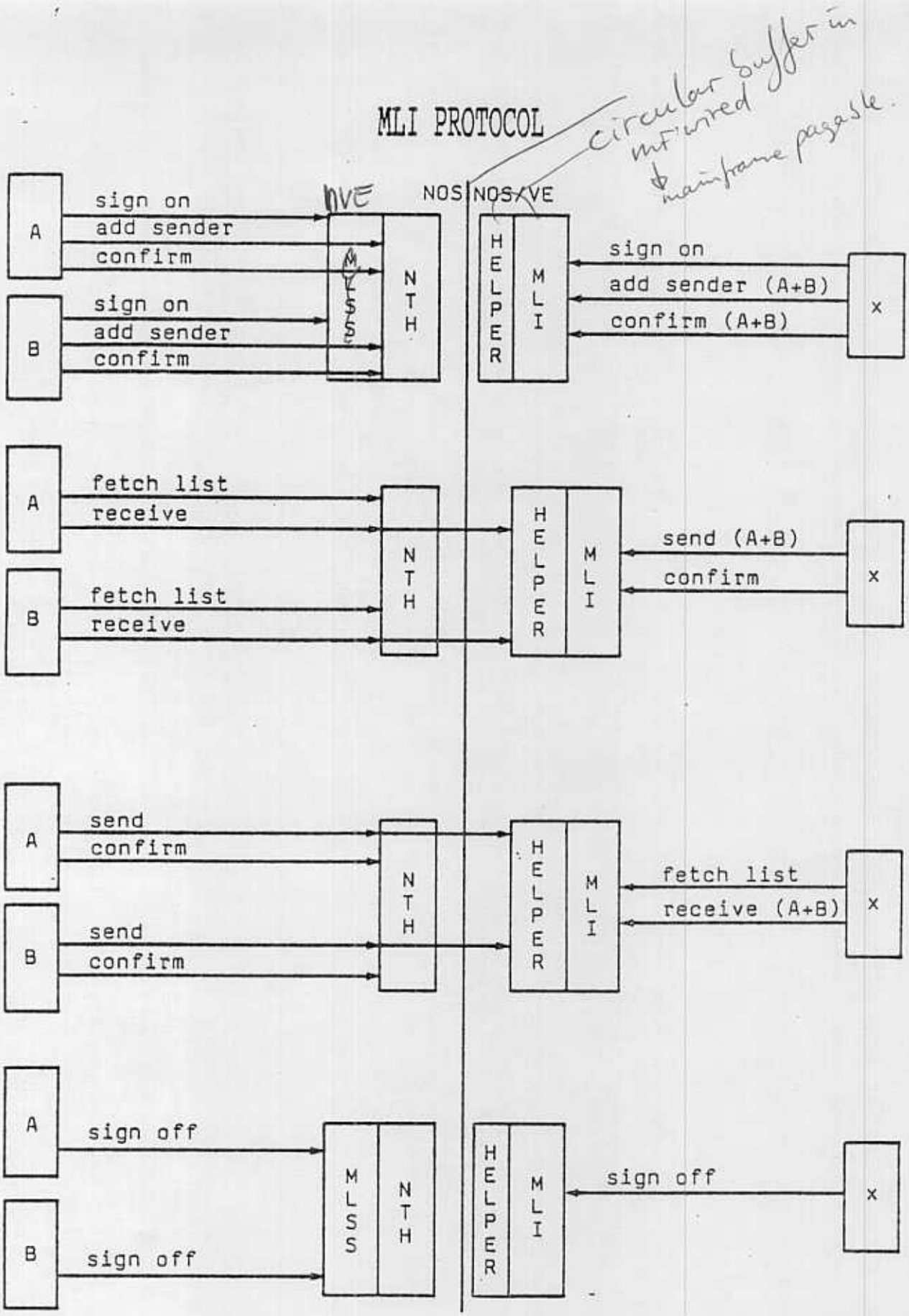
```
MLP$SIGN_ON
        (name,max_msgs,unique_name,status)

MLP$SIGN_OFF
        (name,status)

MLP$ADD_SENDER
        (name,sender_name,status)

MLP$DELETE_SENDER
        (name,sender_name,status)

MLP$CONFIRM_SEND
        (name,destination_name,status)

MLP$SEND_MESSAGE
        (name,info,signal,message_area,message_length,destination
        name,status)

MLP$FETCH_RECEIVE_LIST
        (name,sender_name,list,count,status)

MLP$RECEIVE_MESSAGE
        (name,info,signal,message_area,msg_length,msg_area
        length,receive_index,sender_name,status)
```

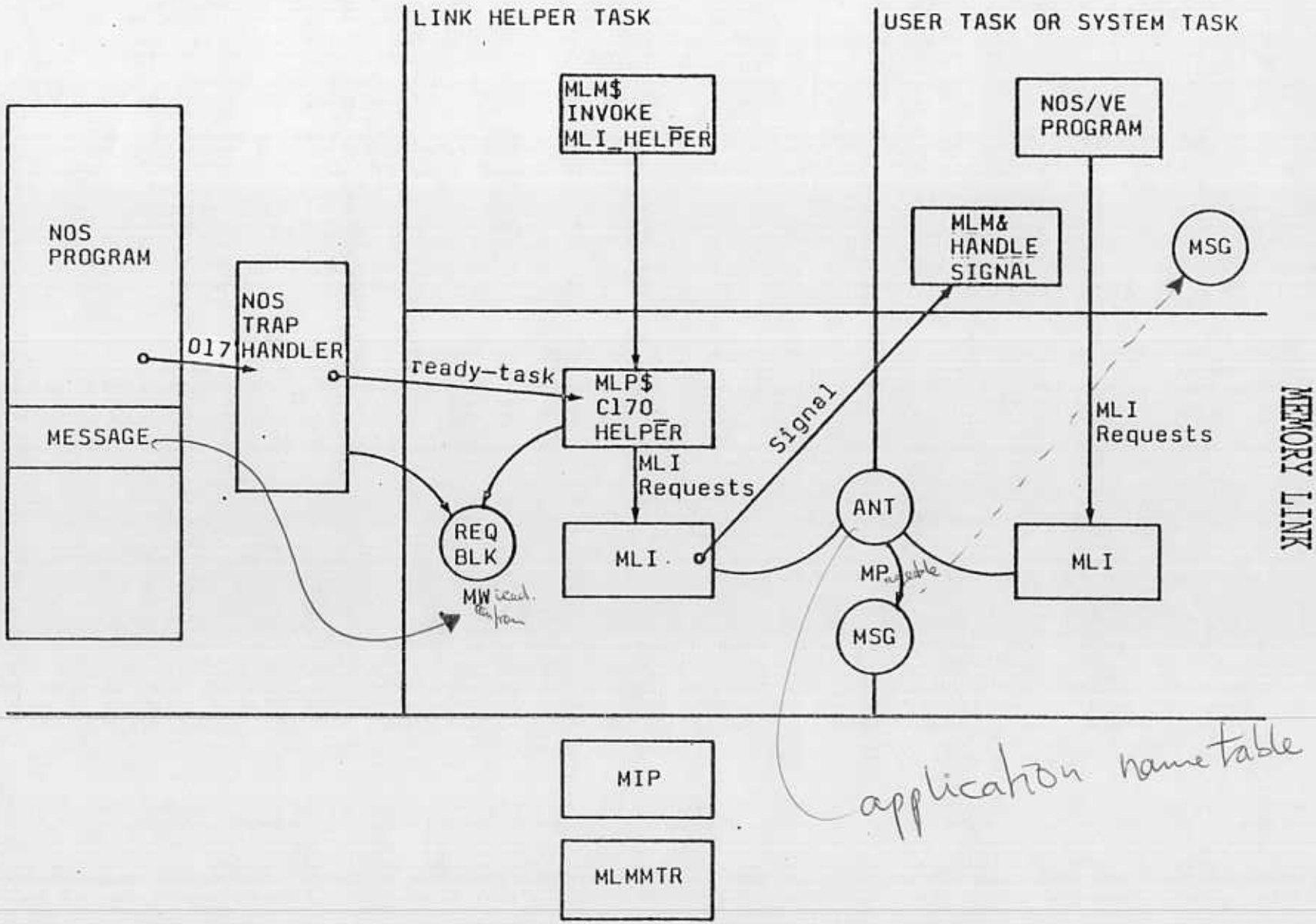
voeg toe aan receive list

zijn er boodschappen

haal op.

control information


MLI PROTOCOL
circular buffer in mf'ware & mainframe pagable.
NVE
NOS
NOS/VE
A
B
sign on
add sender
confirm
MLSS
NTH
HELPER
MLI
sign on
add sender (A+B)
confirm (A+B)
X
fetch list
receive
send (A+B)
send
fetch list
receive (A+B)
sign off


LINK HELPER TASK
USER TASK OR SYSTEM TASK
MLM$ INVOKE MLI_HELPER
NOS/VE PROGRAM
NOS PROGRAM
NOS TRAP HANDLER
MLM& HANDLE SIGNAL
MSG
017
ready-task
MLP$ C170 HELPER
MLI Requests
Signal
MLI Requests
MESSAGE
REQ BLK
ANT
MLI
MLI
MP available
MSG
MIP
MLMMTR
application name table
MEMORY LINK

MEMORY LINK SOFTWARE

- The NOS program uses CYBIL procedures or COMPASS macros or FORTRAN subroutines to communicate with the NOS/VE job. These translate to 017 instructions which are trapped by the NOS trap handler (NTH).

- When an 017 instruction is executed, the NOS trap handler runs in NOS/VE instruction mode. It moves the message into a circular buffer. The entries are called request blocks. Trap handler issues a monitor request to ready the link_helper task.

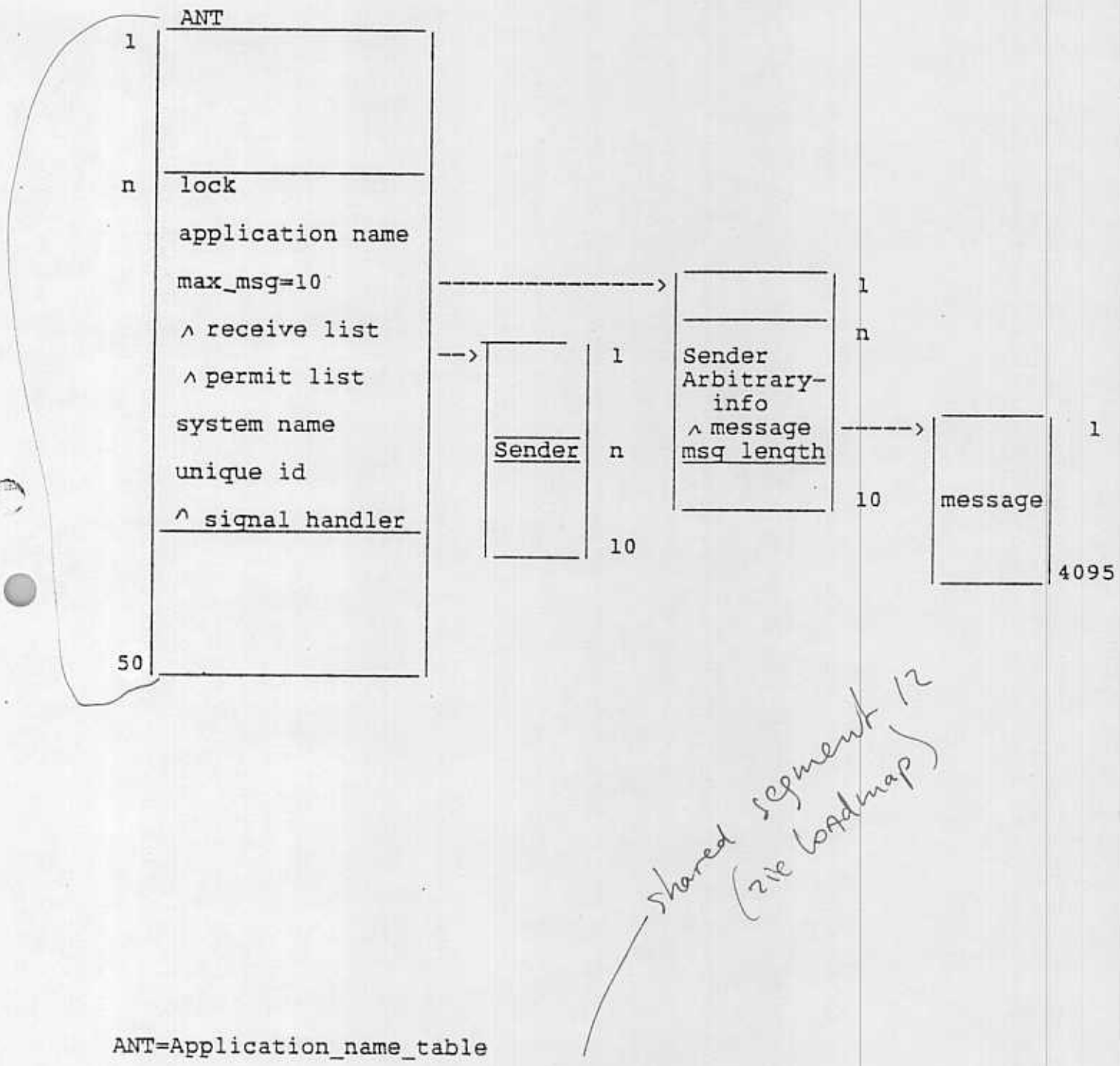
- MLI helper transfers the message from the circular buffer to the mainframe pageable segment using MLI transfer requests. Some NOS/VE applications are signaled when there is a message received for that application.

- MLI manages the queue and services the requests issued by MLI helper, Interactive Facility, and so on.

- The NOS/VE program transfers the message into its buffer using MLI requests. Applications which use the facility are:

  RHF180
  Interactive Exec.
  Interactive Facility
  Operator Facilty
  Interstate Communication (users)

MEMORY LINK TABLES

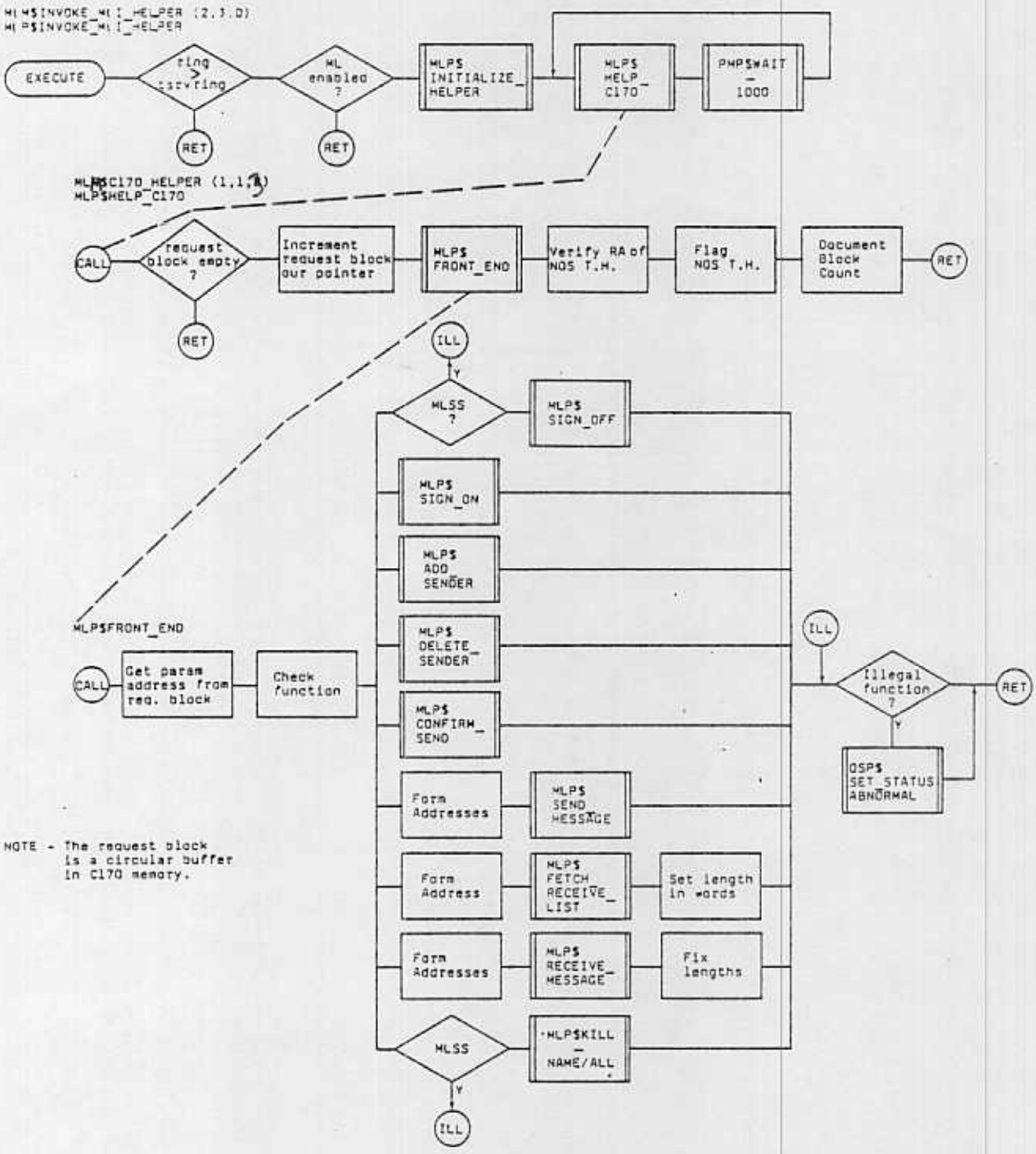


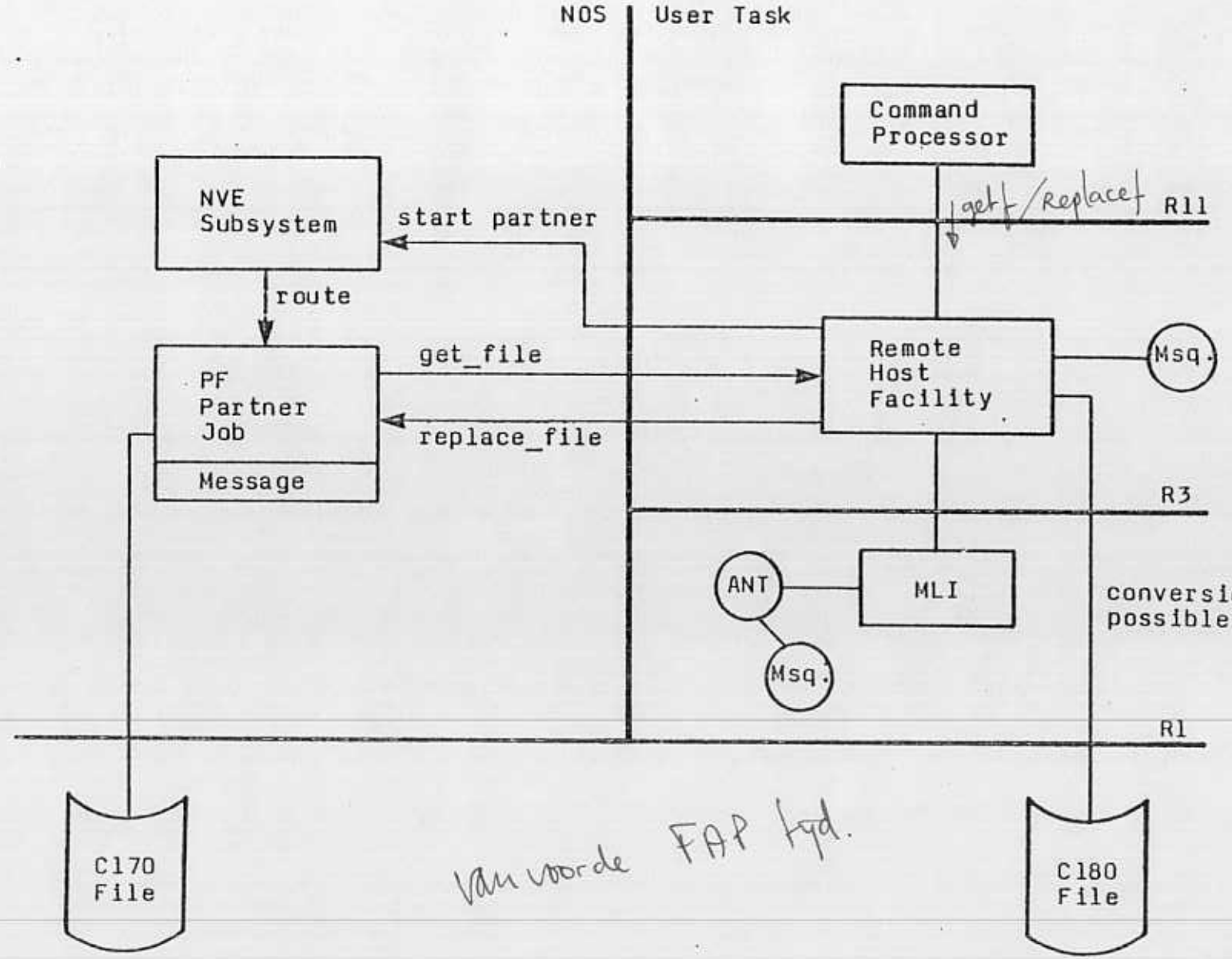
shared segment 12 (via loadmap)

ANT=Application_name_table

All tables are in mainframe pageable segment.

ML HELPER


MLMS$INVOKE_MLI_HELPER (2,3,0)
MLPS$INVOKE_MLI_HELPER
EXECUTE
ring > currring
RET
ML enabled ?
RET
MLPS$ INITIALIZE_ HELPER
MLPS$ HELP_ C170
PMPS$WAIT _ 1000
MLMS$C170_HELPER (1,1,0)
MLPS$HELP_C170
CALL
request block empty ?
RET
Increment request block our pointer
MLPS$ FRONT_END
Verify RA of NOS T.H.
Flag NOS T.H.
Document Block Count
RET
ILL
Y
MLSS ?
MLPS$ SIGN_OFF
MLPS$ SIGN_ON
MLPS$ ADD_ SENDER
MLPS$ DELETE_ SENDER
MLPS$ CONFIRM_ SEND
Form Addresses
MLPS$ SEND_ MESSAGE
Form Address
MLPS$ FETCH RECEIVE_ LIST
Set length in words
Form Addresses
MLPS$ RECEIVE_ MESSAGE
Fix lengths
MLSS
MLPS$KILL _ NAME/ALL
Y
ILL
MLPS$FRONT_END
CALL
Get param address from req. block
Check function
ILL
Illegal function ?
Y
RET
OSP$ SET_STATUS ABNORMAL
NOTE - The request block is a circular buffer in C170 memory.



RHF-PERM FILES

REMOTE HOST FACILITY
(Permanent Files)

ENVIRONMENT:

Task in any job.

ACTIVATED:

By command interfaces get_file and replace_file.

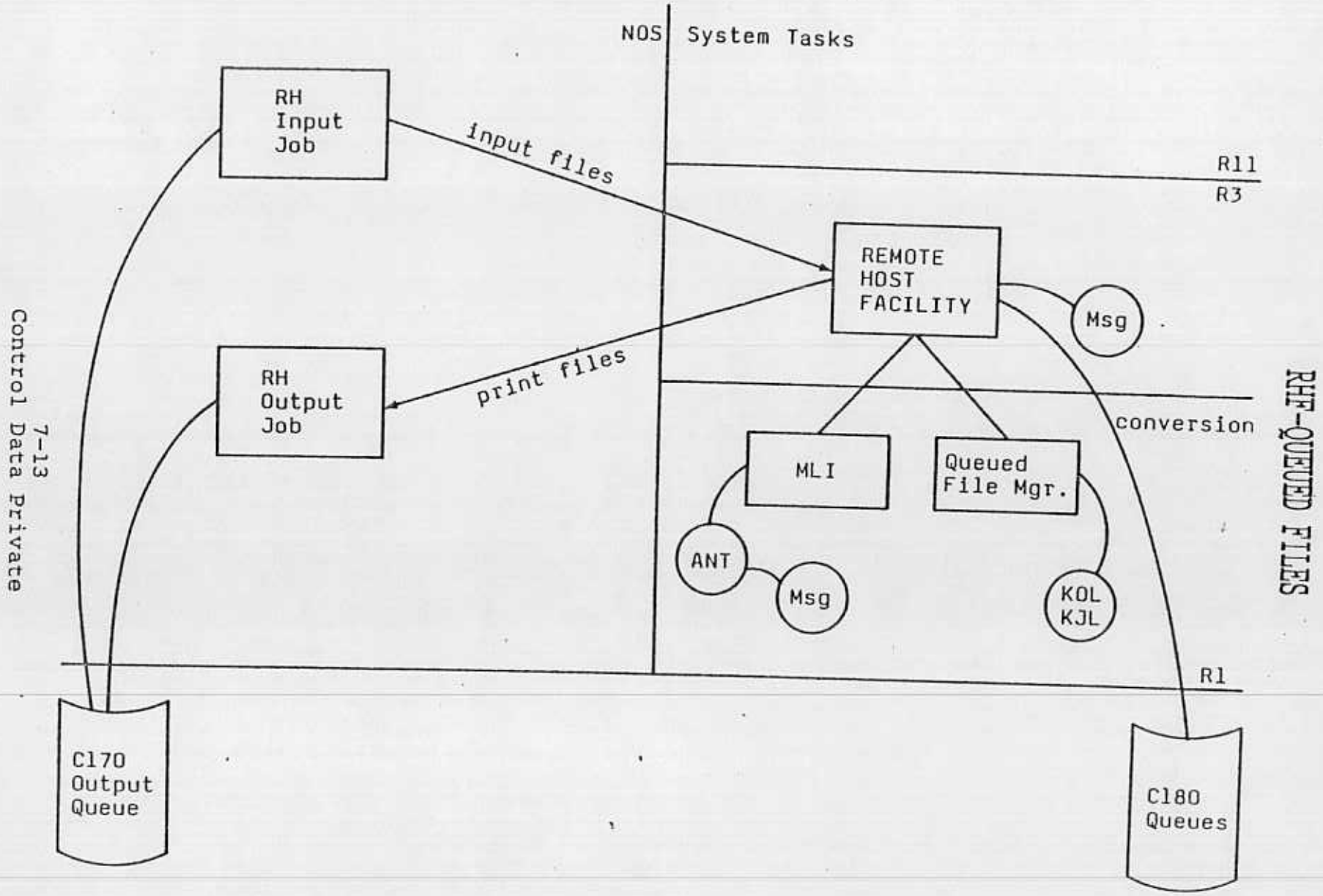
PARTNER:

The NVE subsystem routes a NOS job which is brought to a control point. This job signs on to the memory link and begins communication with the remote host facility running on behalf of a NOS/VE task.

PROCESSING:

NOS/VE initiates the transfer. The records are transferred one at a time. Usually the file undergoes some conversion. The conversion is always performed on the NOS/VE side.

Ztype Record <==> Vtype record



REMOTE HOST FACILITY
(Queued Files)

ENVIRONMENT:

RHIN and RHOUT are tasks in the system job.

ACTIVATED:

By the deadstart procedure.

PARTNER:

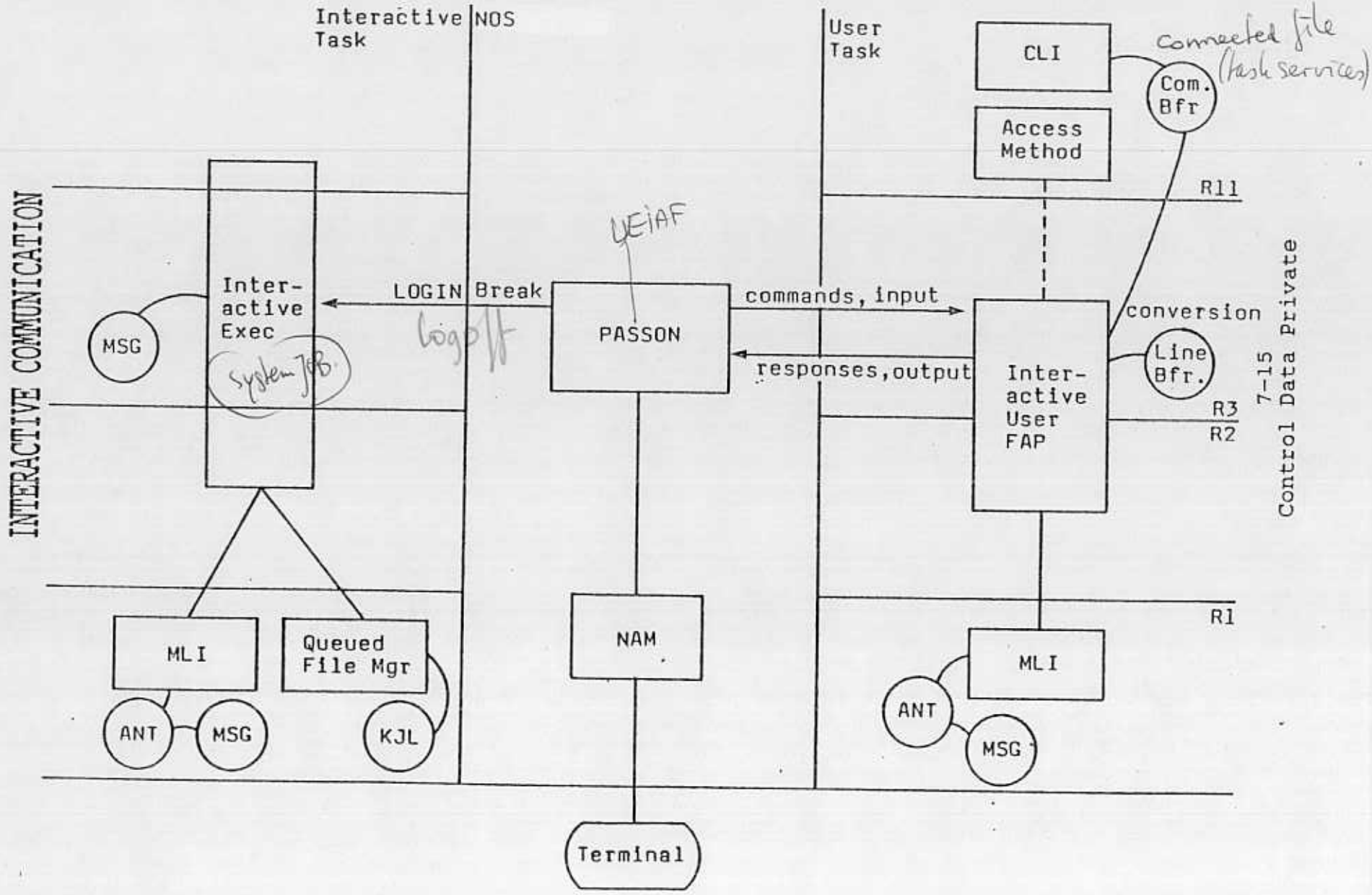
NOS system jobs which are always at a control point.

PROCESSING:

NOS/VE asks NOS if there is an input job for NOS/VE. NOS scans the output queue. If there is such a job, it is sent a record at at time. The z-records are converted to v-records by NOS/VE. The file is given to the queued file manager to be entered in the KJL.

PROCESSING (RHOUT):

NOS/VE tells NOS that there is a file in the NOS/VE output queue (KOL). NOS/VE converts the v-records to the z-records and sends them to NOS a record at a time.


INTERACTIVE COMMUNICATION
Interactive Task
NOS
User Task
CLI
Com. Bfr
connected file (task services)
Access Method
R11
Inter-active Exec
MSG
LOGIN
Break
logoff
UEIAF
PASSON
commands, input
responses, output
conversion
Line Bfr.
Inter-active User FAP
R3
R2
7-15 Control Data Private
MLI
Queued File Mgr
ANT
MSG
KJL
NAM
R1
MLI
ANT
MSG
Terminal

INTERACTIVE COMMUNICATION

ENVIRONMENT:

The interactive executive runs as a task in the system job. The interactive FAP is part of task services for every task.

ACTIVATED:

The interactive executive is activated by the deadstart procedure. The interactive FAP is associated with interactive files by command language interpreter.

PARTNER:

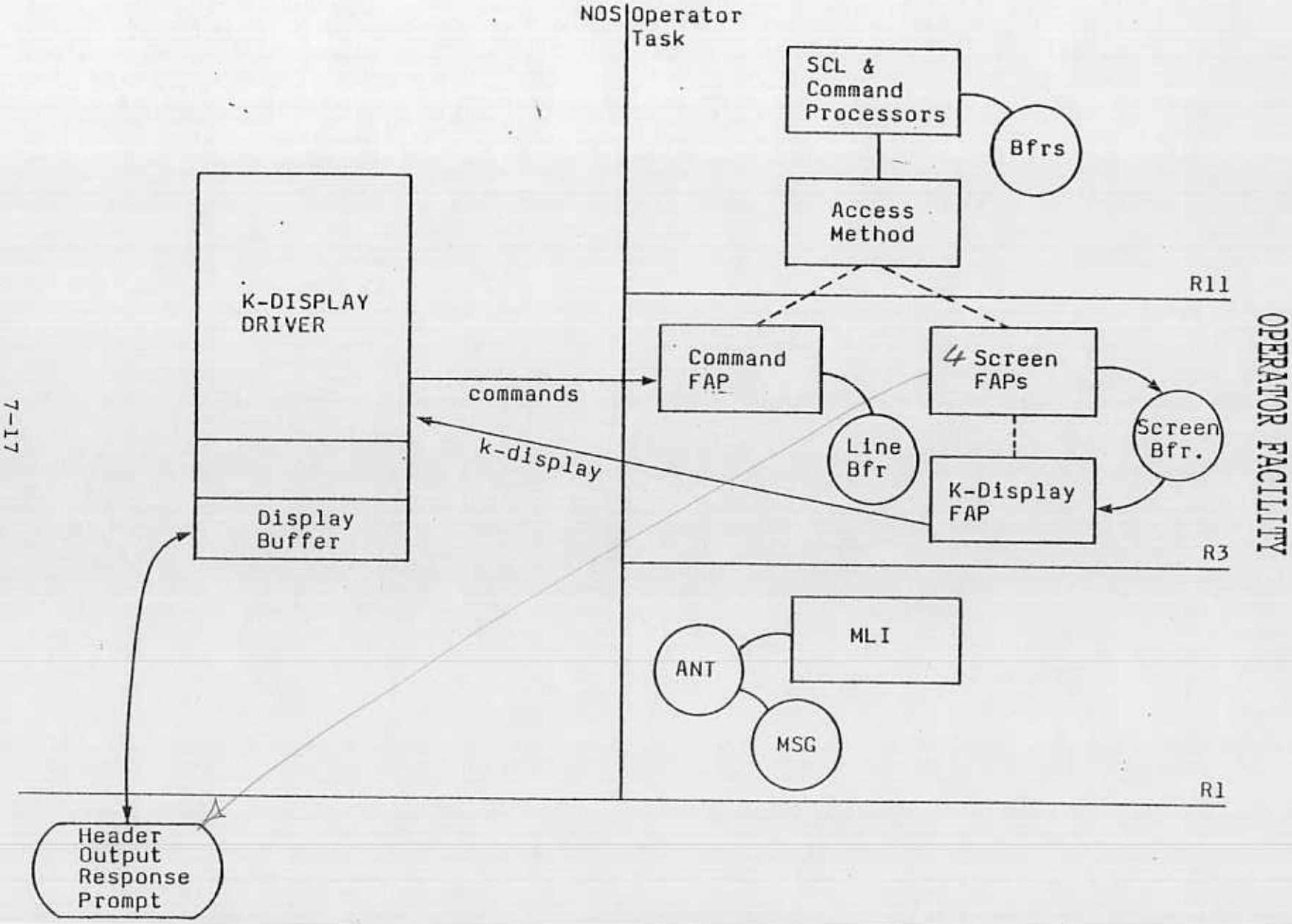
PASSON is an application of NAM. PASSON handles all information for the VEIAF application. LOGIN and interactive break are passed to the memory link application called interactive executive. Other commands are passed to the user task.

PROCESSING (INTERACTIVE EXECUTIVE):

When the interactive executive is executed by the job monitor of the system job, it is possible for users of NOS/VE to login. When they do, the executive calls queued file manager to make entries in the input queue (KJL).

PROCESSING (INTERACTIVE FAP):

Eventually the job is scheduled and the job monitor runs. Job monitor executes the command language interpreter (CLI). When CLI opens an interactive file, the interactive FAP is linked and entered. The FAP signs on to the memory link and establishes communication with PASSON. Now each time CLI wants to communicate with the user, it writes or reads the communication file and the FAP handles the memory link transfers.


NOS Operator Task
SCL & Command Processors
Bfrs
Access Method
R11
K-DISPLAY DRIVER
Command FAP
4 Screen FAPs
commands
k-display
Line Bfr
Screen Bfr.
K-Display FAP
Display Buffer
R3
MLI
ANT
MSG
R1
Header
Output
Response
Prompt
OPERATOR FACILITY

ENVIRONMENT:

A task in the system job.

ACTIVATED:

By the deadstart procedure.

PARTNER:

NVE is a NOS subsystem. It must occupy a control point from before NOS/VE deadstart untii NOS/VE termination.

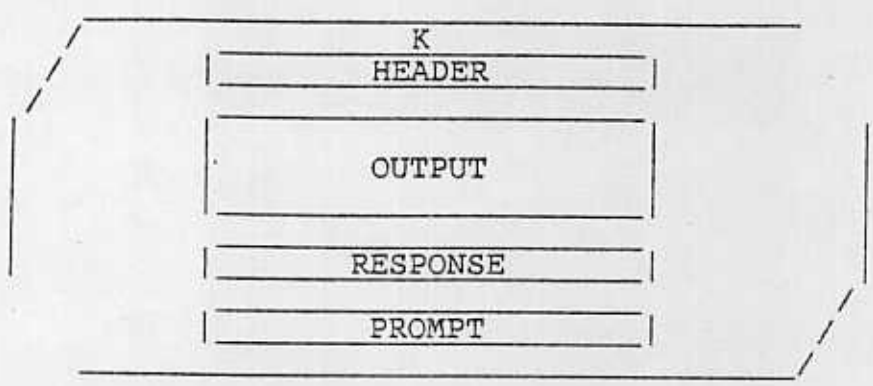
NVE includes the K display driver. In this role it is similar to PASSON. It transfers the NOS operator's console K display to and from the operator facility via the memory link.

PROCESSING:

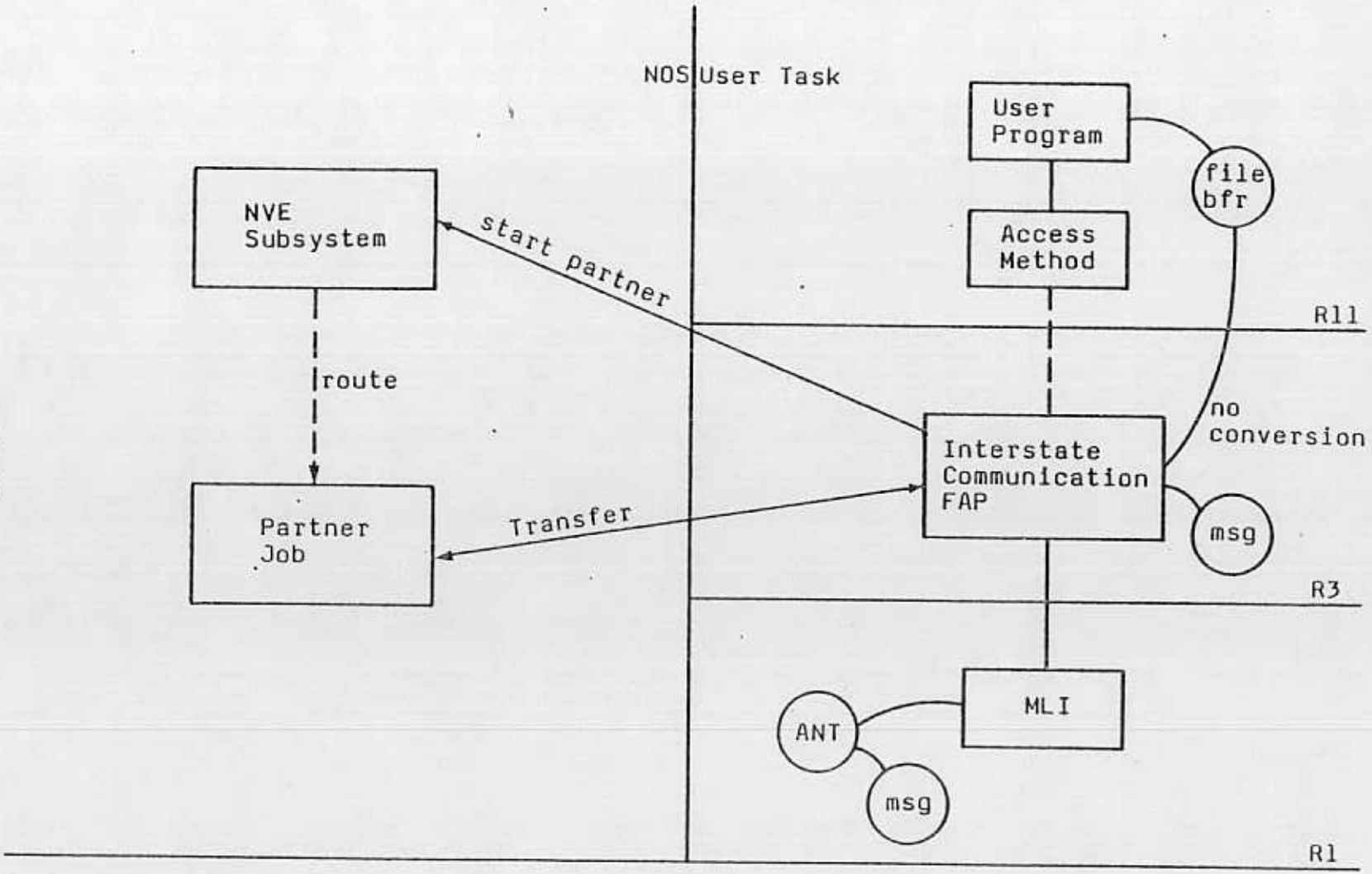
The operator console is treated like a terminal. For input, the operator facility uses a FAP which is nearly the same as the interactive FAP. CLI reads and writes the communication file and processes the commands in the same manner as it would for any interactive user. The only difference is that this task of the system job has privilege which extends to command repertoire.

For output, the operator facility uses a collection of FAPs. the K display is divided as shown below. When a procedure writes to a part of the screen, it writes to the lfn which has the FAP for that part of the screen. This FAP adds the information to a screen buffer.

Periodically, the screen buffer is written. Another FAP makes the memory link requests to transfer it to the partner.



iedere task kan dit doen



INTERSTATE COMMUNICATION

ENVIRONMENT:

Any task in any job.

ACTIVATED:

When OPEN finds icp$fap_control as the FAP attribute of the link file.

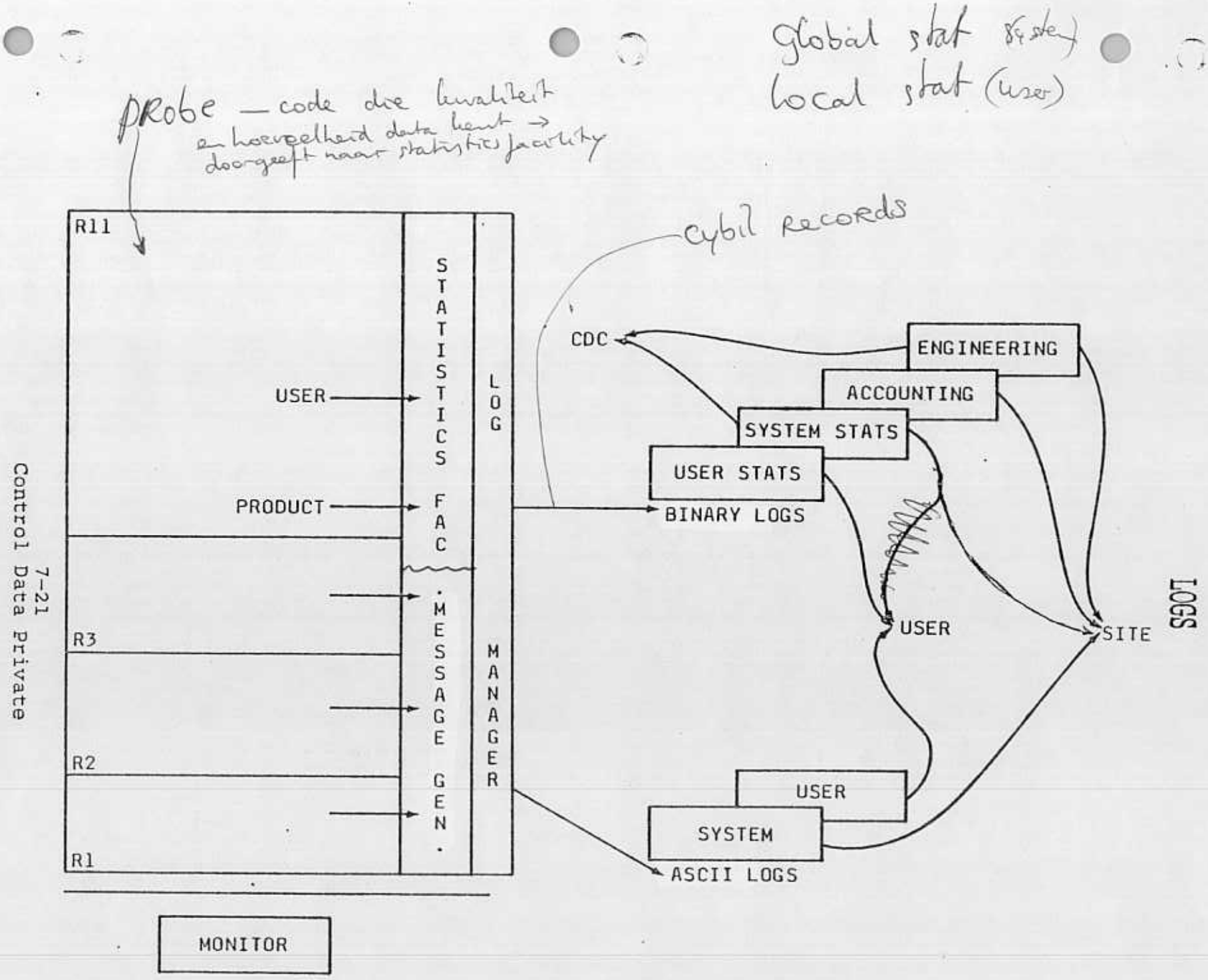
PARTNER:

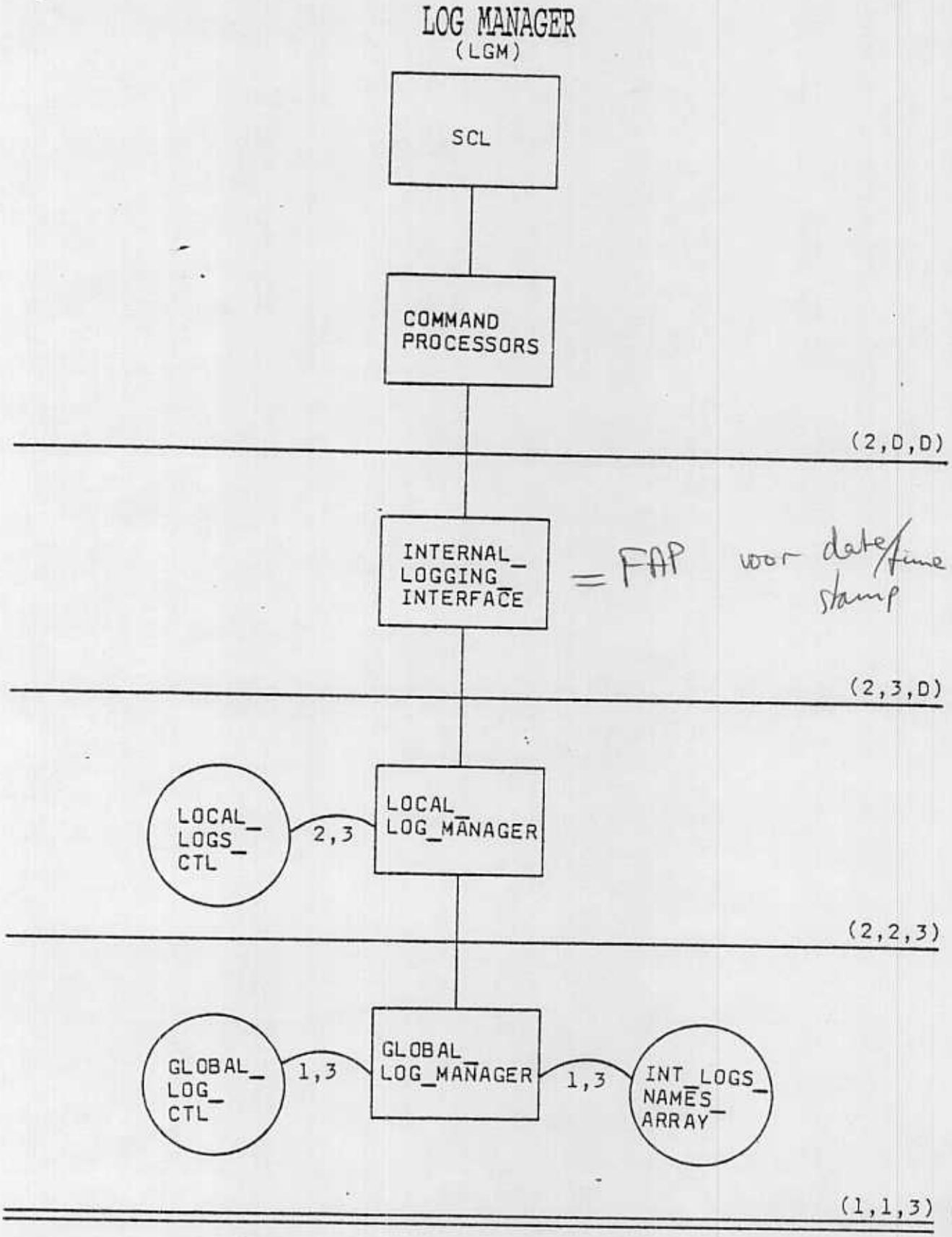
The partner is written by the user in FORTRAN, CYBIL or COMPASS. Requests are available which synchronize the communication with the NOS/VE task.

PROCESSING:

The NOS/VE program does normal I/O (open, get, put) on the file. On OPEN the string variable named in the user_info field of the link file attributes is transferred by the FAP to the NVE application. NVE routes the string (presumably a command stream). The partner starts.

On get (or put), the FAP transfers from (or to) the partner. The FAP enforces a serial protocol; that is, each 'message' must be received before another can be sent.


global stat (& site)
local stat (user)
probe — code die kwaliteit en hoeveelheid data haalt → doorgeeft naar statistics facility
cybil records
R11
R3
R2
R1
USER
PRODUCT
STATISTICS FAC.
MESSAGE GEN.
LOG MANAGER
MONITOR
CDC
ENGINEERING
ACCOUNTING
SYSTEM STATS
USER STATS
BINARY LOGS
USER
SITE
USER
SYSTEM
ASCII LOGS
LOGS


LOG MANAGER
(LGM)
SCL
COMMAND PROCESSORS
(2,D,D)
INTERNAL_ LOGGING_ INTERFACE
= FAP voor date/time stamp
(2,3,D)
LOCAL_ LOGS_ CTL
2,3
LOCAL_ LOG_MANAGER
(2,2,3)
GLOBAL_ LOG_ CTL
1,3
GLOBAL_ LOG_MANAGER
1,3
INT_LOGS_ NAMES_ ARRAY
(1,1,3)

Any functional area Log manager

LGP$ADD_ENTRY_TO_BINARY_LOG
(log,entry_address,log_address,cycle,status)

LGP$APPEND_JOB_LOG_TO_OUTPUT
(status)

LGP$BUILD_DISPLAY_OF_ASCII_LOG not in MAP.
(log,scroll_size,status)

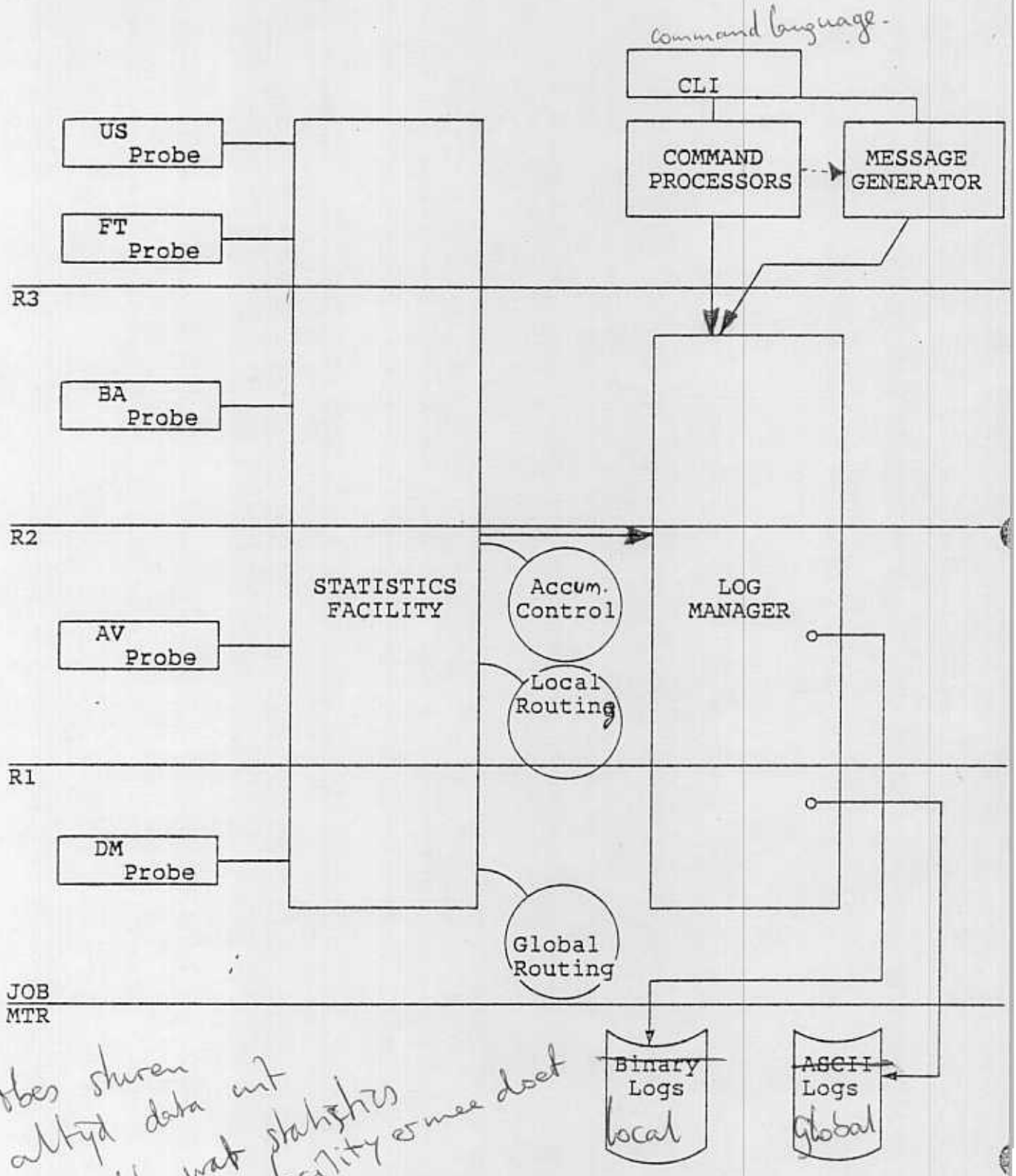
LGP$INTERCEPT_LOG_IO_REQUEST

(fid,call_block,layer_no,status)
This is a FAP.

LGP$SETUP_ACCESS_TO_LOCAL_LOGS(status)

LGP$SETUP_ACCESS_TO_GLOBAL_LOGS(logs,status) not in MAP

STATISTICS VS LOG MANAGER



probes sturen
altijd data uit
ongeacht wat statistics
facility ermee doet

PROBE

A PROBE IS THAT PORTION OF SOFTWARE RESPONSIBLE FOR COLLECTING AND EMITTING A STATISTIC TO THE STATISTICS FACILITY.

- PROBES ARE EMBEDDED IN KEY AREAS OF THE SYSTEM, BUT ARE NOT SUBJECT TO GUIDELINES LIKE KEYPOINTS.
- THE PRECISE LOCATION OF PROBES AND THE INFORMATION REPORTED WILL BE DETERMINED BY THE REQUIREMENTS OF THE COMPONENTS IN WHICH THEY LIE.
- THE FREQUENCY AT WHICH A PROBE EMITS A STATISTIC TO THE STATISTICS FACILITY IS DETERMINED BY THE SUPERORDINATE COMPONENT.
- A PROBE DOES NOT ASCRIBE ANY INHERENT QUALITIES TO A STATISTIC.
- THERE SHOULD BE A ONE-TO-ONE CORRESPONDENCE BETWEEN A PROBE AND STATISTIC.

NOS/VE STATISTIC

A NOS/VE STATISTIC HAS THREE COMPONENTS:

- STATISTIC CODE : AN ORDINAL THAT UNIQUELY IDENTIFIES THE STATISTIC.
- DESCRIPTIVE DATA: A STRING INDICATING THE OCCURRENCE OF A SYSTEM OR JOB EVENT.
- COUNTERS : A SEQUENCE OF COUNTERS CONTAINING REPORTED VALUES OF SYSTEM OR JOB VARIABLES.

PRODUCT STATISTICS COLLECTED BY NOS/VE

In general, the O/S is responsible for collecting job step statistics that can be determined external to the product, that is statistics that the O/S is capable of determining.

For each product identified in SIS section 4.1 that is directly invoked by the user, e.g., via command or as a program initiated task, NOS/VE will record resources used per invocation. Resources accounted for include:

- Total CP-time
- Maximum virtual memory used
- Maximum real memory used
- Average working set size — alle Real mempage frames die bot iets beboren (task, job, system)
- CP-time per memory size used
- Number of I/O requests
- Number of data read/written to files

Additional data to be collected for each invocation of a product include:

- Origin of job step - batch command, terminal command, procedure file, executing job.
- Type of termination - normal, product error, time limit, invalid memory request, operator drop, and so on. A recovered condition does not cause product termination.
- Abnormal conditions recovered from.
- Average interactive response time for interactive products - the average elapsed time between input data available and output data issued to terminal.
- The fact that the product was invoked (added to count of the number of separate invocations).
- Number of modules loaded (input units for the loader)
- Source languages of modules loaded (added to language usage count).

STATISTICS MANAGER TABLES

GLOBAL BINARY LOG FORMAT

BINARY DATE AND TIME
STATISTIC CODE
STATISTIC IDENTIFIER
JOB SEQUENCE NUMBER
GLOBAL TASK ID
CONDENSING FREQUENCY
NUMBER OF COUNTERS
DESCRIPTIVE DATA SIZE
COUNTER_1
COUNTER_2
.
.
.
COUNTER_N
DESCRIPTIVE DATA
.
.
.
.

ACCUMULATION CONTROL

STATISTIC CODE
ACCUMULATION CONTROL TYPE
ACCUMULATION ADDRESS
FREQUENCY ADDRESS
THRESHOLD
FORWARD LINK
BACKWARD LINK

GLOBAL AND LOCAL ROUTING CONTROL TABLE

STATISTIC CODE
IDENTIFIER
ROUTING CONTROL TYPE
ENABLED
CONDENSING_ADDRESS
THRESHOLD
INTERVAL SIZE
INTERVAL_END_TIME
LOG_CYCLE
FORWARD_LINK
BACKWARD LINK

FEATURES

CONDENSING

The first counter of a statistic can be condensed, that is, the information will be collected in the counter until either time runs out or a certain number occur. When the condensing threshold is reached, a new entry is logged and collecting starts again. This might be used to count page faults or total monitor time.

ACCUMULATING

Accumulation also involves collecting occurrences of an event. When the threshold (limit) is reached, some action is taken. Typically the job monitor will be signaled and will take further action. Currently this is being used for CP time and SRUs.

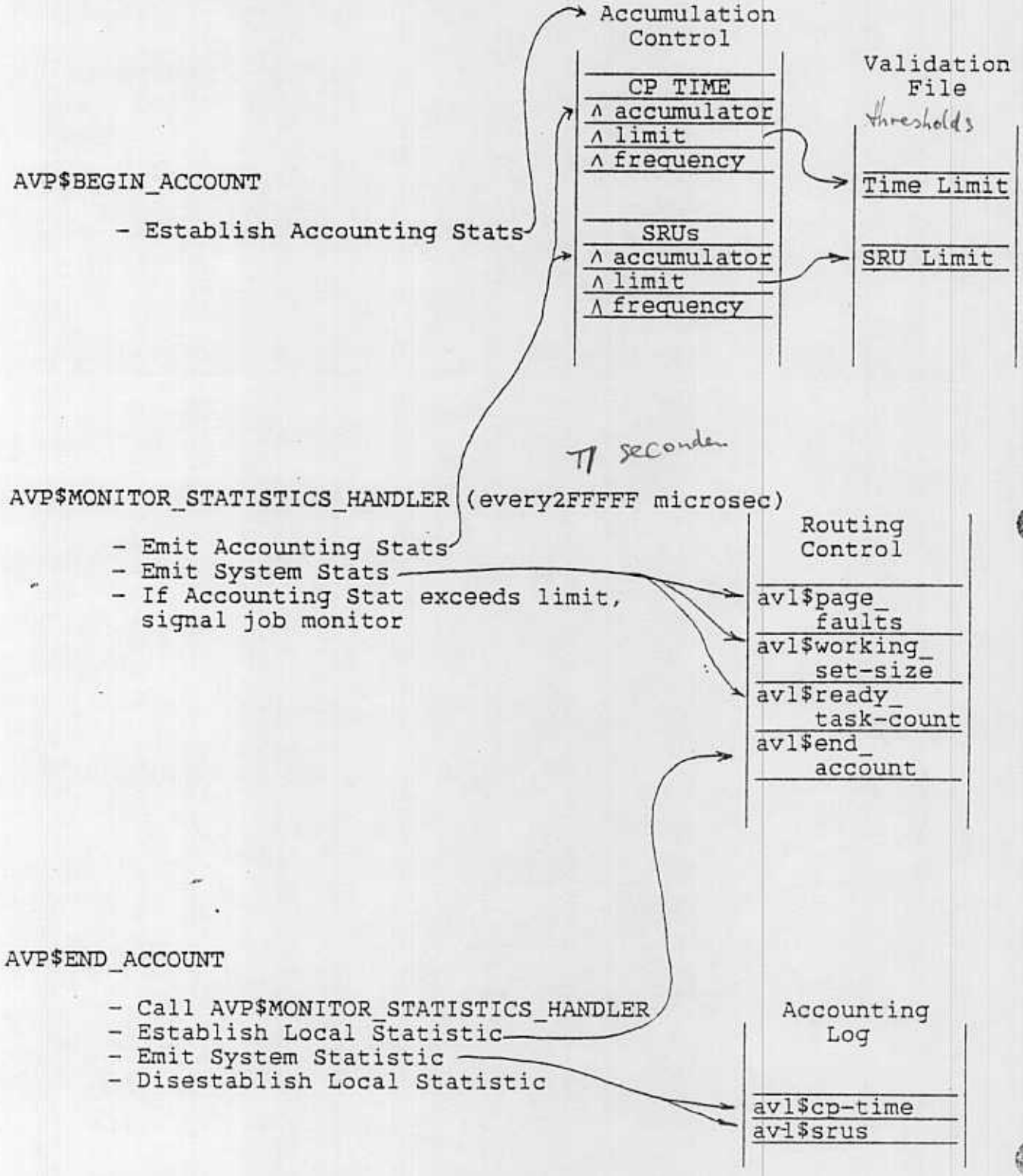
BREAKOUT

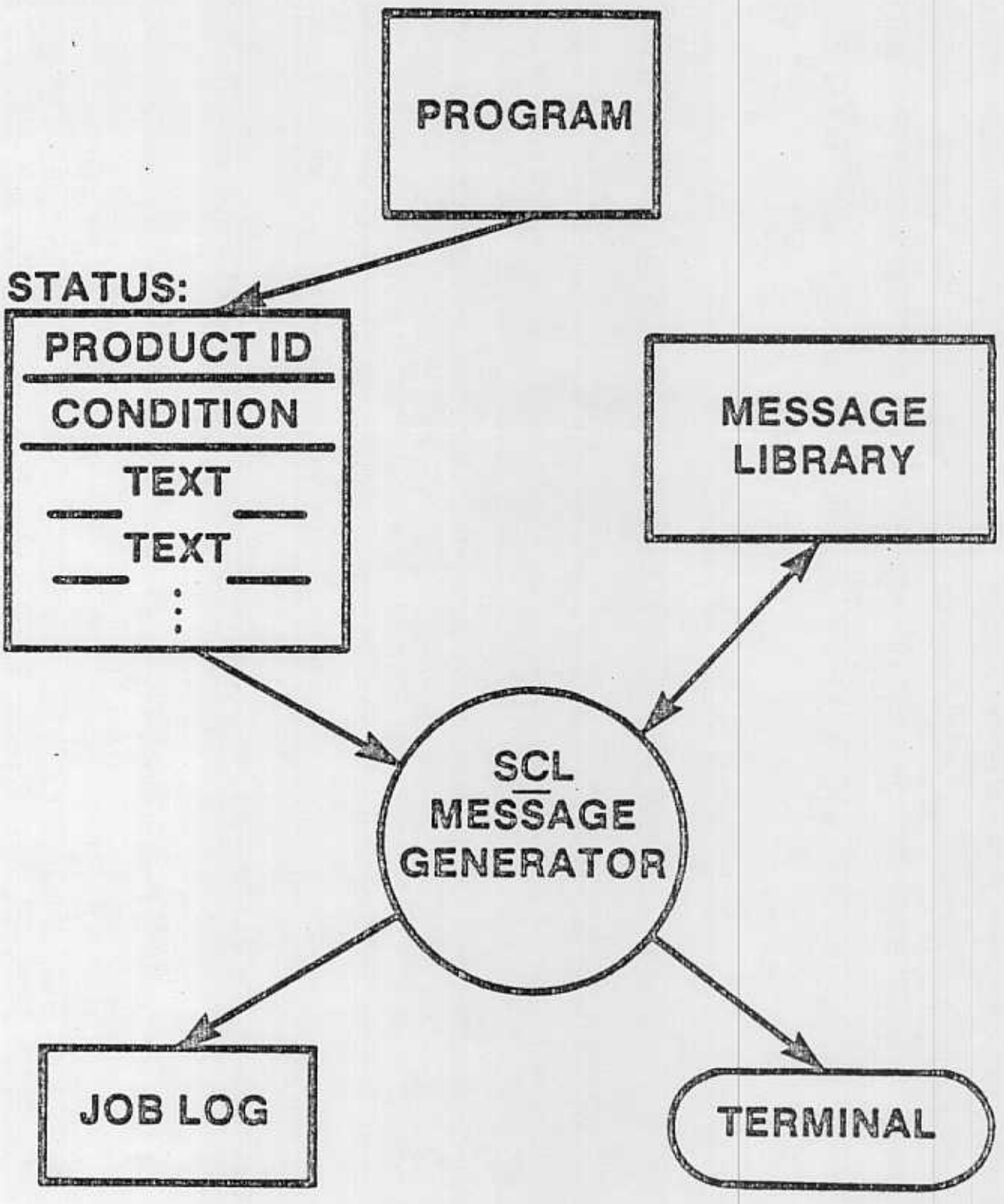
Sometimes it is necessary to keep seek local and global statistics of the same thing. An example might be job time. It would be necessary to have total job time as well as the time for individual jobs to compute standard deviation. If breakout is established, the statistics manager will enter in both the local and the global logs.

naar meerdere logs

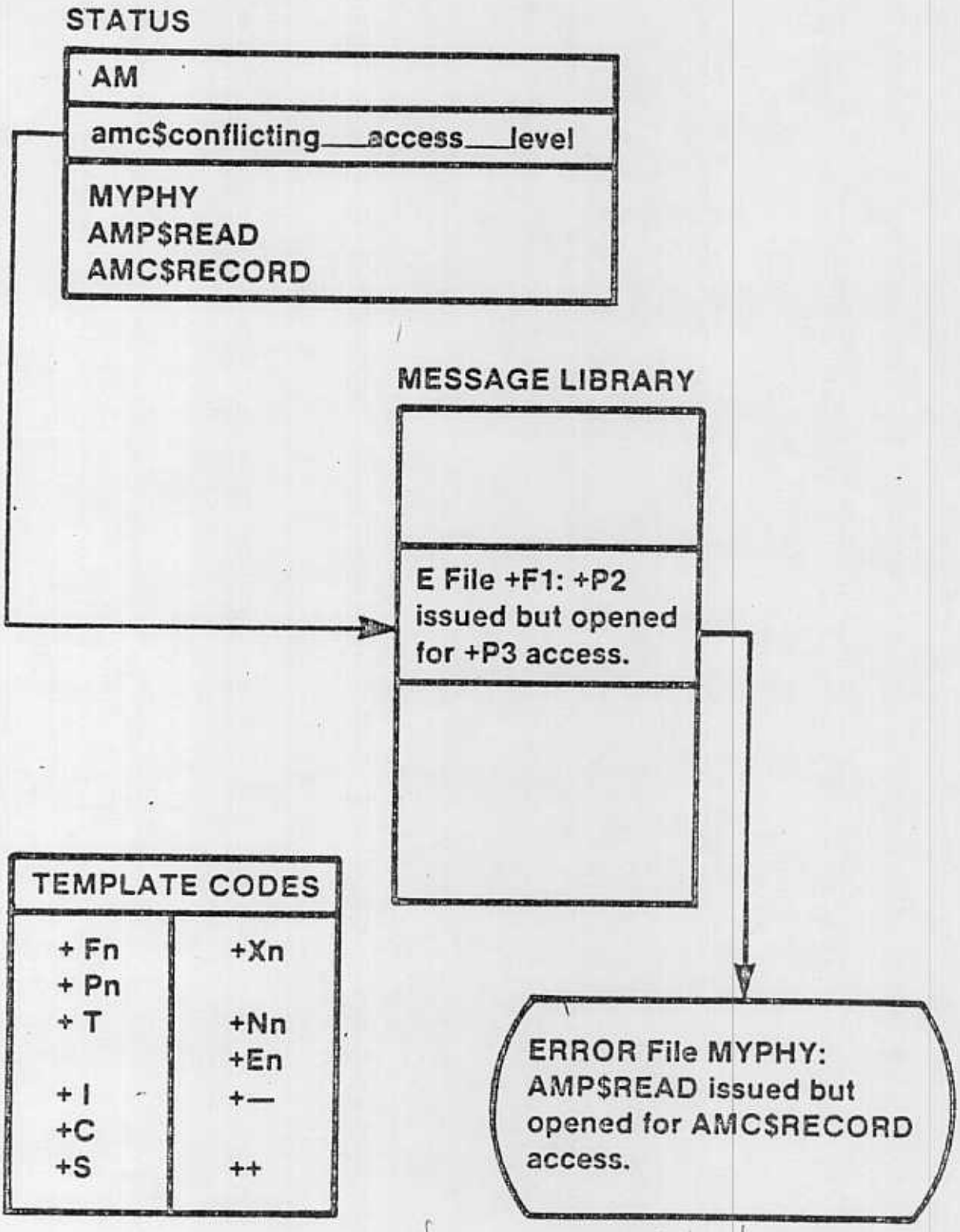
```
PROCEDURE [XREF] sfp$establish_system_statistic (identifier:
    sft$statistic_identifier;

    statistic_code: sft$statistic_code;

    log_name: pmt$global_binary_logs;

    breakout: boolean;

    condensing_control: sft$condensing_control;

    VAR status: ost$status);


PROCEDURE [XREF] sfp$enable_system_statistic (statistic_group:
    sft$statistic_group;

    VAR status: ost$status;


PROCEDURE [XREF] sfp$disable_system_statistic (statistic_group;
    sft$statistic_group;

    VAR status: ost$status);


PROCEDURE [XREF] Sfp$disestablish_system_stat (identifier:
    sft$statistic_identifier;

    statistic_code: sft$statistic_code;

    log_name: pmt$global_binary_logs;

    breakout: boolean;

    VAR status: ost$status);


PROCEDURE [XREF] sfp$emit_system_statistic (identifier:
    sft$statistic_identifier;

    statistic_code: sft$statistic_code:

    descriptive_data: sft$descriptive_data;

    counter: sft$counters;

    VAR status: ost$status);
```

ACCOUNTING


Accumulation Control
CP TIME
∧ accumulator
∧ limit
∧ frequency
SRUs
∧ accumulator
∧ limit
∧ frequency
Validation File
thresholds
Time Limit
SRU Limit
AVP$BEGIN_ACCOUNT
- Establish Accounting Stats
AVP$MONITOR_STATISTICS_HANDLER (every2FFFFF microsec)
- Emit Accounting Stats
- Emit System Stats
- If Accounting Stat exceeds limit, signal job monitor
Routing Control
avl$page_faults
avl$working_set-size
avl$ready_task-count
avl$end_account
AVP$END_ACCOUNT
- Call AVP$MONITOR_STATISTICS_HANDLER
- Establish Local Statistic
- Emit System Statistic
- Disestablish Local Statistic
Accounting Log
avl$cp-time
avl$srus


PROGRAM
STATUS:
PRODUCT ID
CONDITION
TEXT
TEXT
MESSAGE LIBRARY
SCL MESSAGE GENERATOR
JOB LOG
TERMINAL

MESSAGE GENERATION

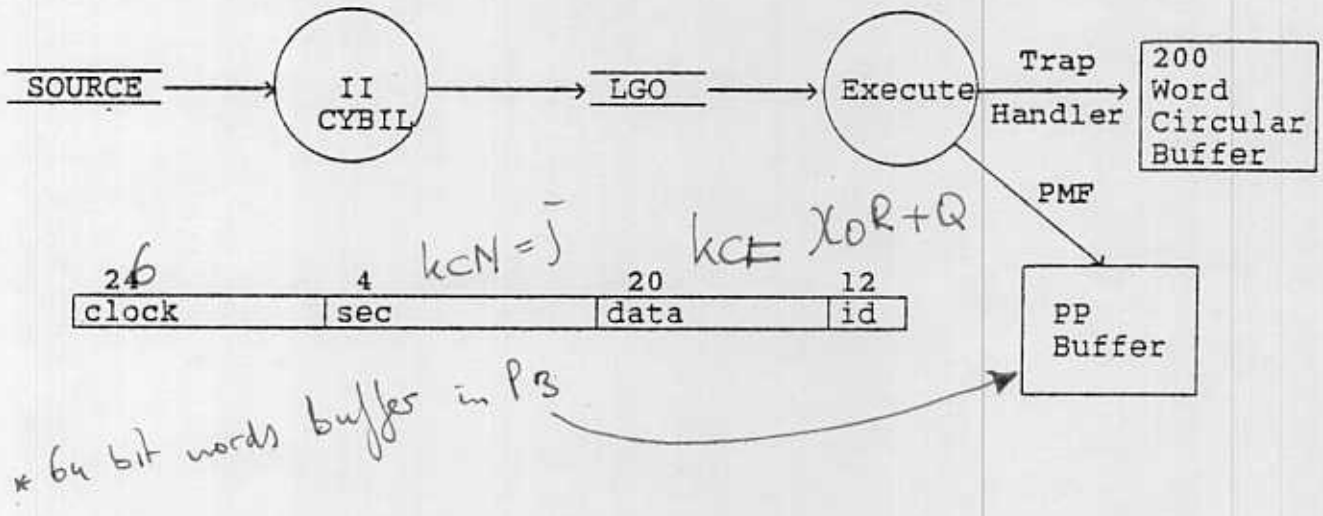


system interface of program interface

KEYPOINT INSTRUCTIONS

- Special instructions in C180 state which may be collected through PMF hardware or cause a software trap (if selected).
- Puts four basic items into 64-bit FIFO buffer entry:
  - Overflow indicator
  - Time stamp
  - Keypoint class
  - Data value
- Obviously requires cooperation from software to work. Conventions are described in SIS/180 and in NOS/VE Procedures and Conventions. Additional data available in Integration Procedures Notebook.
- Control and implementation through common decks and intrinsics.

  #KEYPOINT(class,data,id)



KEYPOINT CLASS

- 16 classes to allow hierarchical priority of keypoint collection.

- Class usage defined in SIS/180.

Class	Usage
0	O/S data
1	O/S unusual
2	O/S procedure entry
3	O/S procedure exit
4	O/S debug
5	Reserved for O/S
6	Product set data
7	Product set unusual
8	Product set procedure entry
9	Product set procedure exit
10	Product set debug
11-14	Reserved for end users
15	PMF hardware start/stop control

gated procedures

major internal procedures

XDCL door
member functions
generic.

PMF SOFTWARE

- Data collection
  - Operates under NOS/170 with both CP and PP code.
  - Various options to handle counter and event control capabilities.

- Data reduction
  - Keypoint processor

    Common to all keypoint data reduction processing. Accepts keypoint data from three sources:

    . 180 Simulator Keypoint File

    : NOS/VE Software Keypoint File

    . PMF hardware through PMF data collection

  - Data reduction

    There will be multiple data reduction programs as the facility is implemented. Those now underway are:

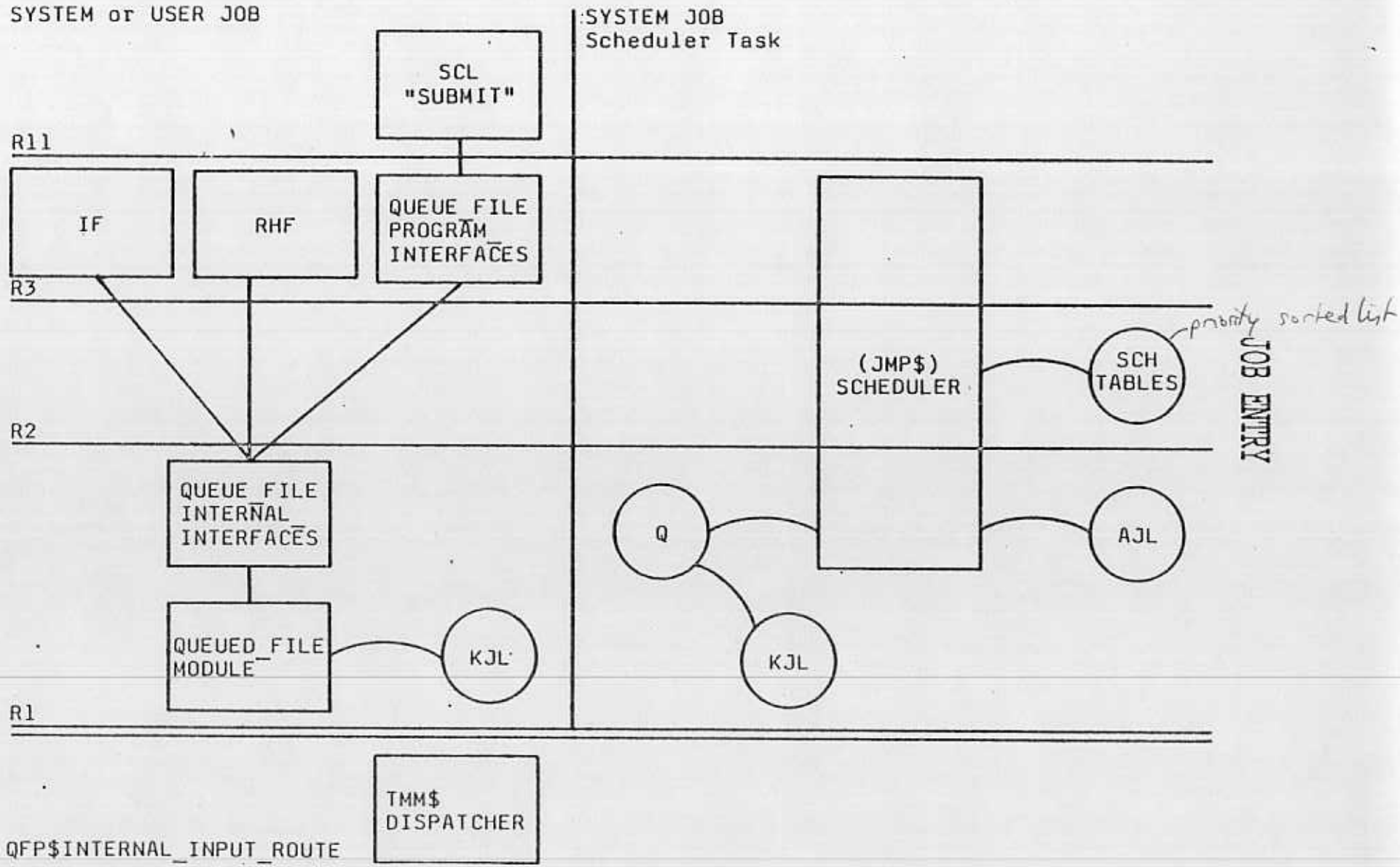
    . "Current" Data Reduction (on the fly)

      provides system level reporting on keypoints and associated data (mainly resource usage information).

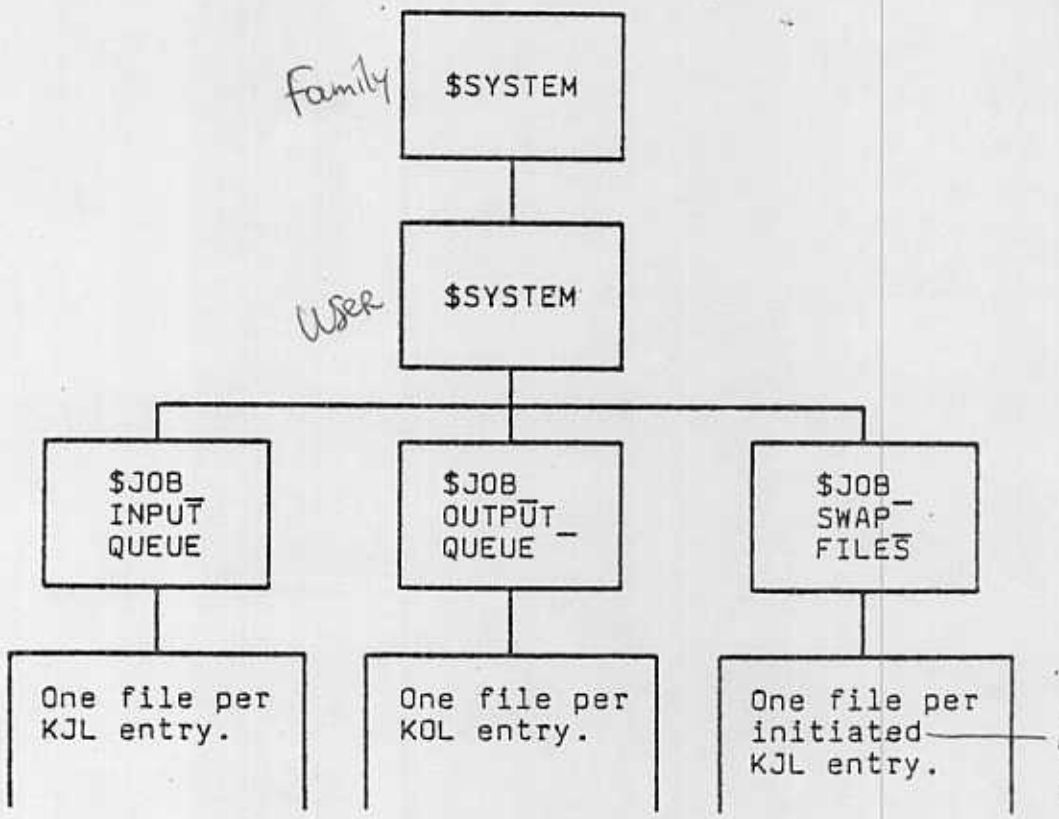
    . Entry/Exit Analysis Reports

      Provides data on major software modules called by task and associated resource usage.

  In the future, there will be a database implementation allowing long term tracking and correlation of data from various collection runs and sources.


SYSTEM or USER JOB
SYSTEM JOB
Scheduler Task
SCL
"SUBMIT"
R11
IF
RHF
QUEUE_FILE
PROGRAM_
INTERFACES
R3
(JMP$)
SCHEDULER
priority sorted list
SCH
TABLES
JOB ENTRY
R2
QUEUE_FILE
INTERNAL_
INTERFACES
Q
AJL
QUEUED_FILE
MODULE
KJL
KJL
R1
TMM$
DISPATCHER
QFP$INTERNAL_INPUT_ROUTE

QUEUED FILES



FILE NAMES: 1. user_job_name
2. system_job_name
AAAA$,AAAB$,...

RECOVERY: The $SYSTEM catalog is recovered like any PF catalog. Information in the system file labels (SFL) of the files is sufficient to reconstruct the KJL and the KOL.

Queue file = PF met attributes.

SWAP file = file die ontstaat (leeg) als job file wordt gecreëerd
alle pages in Real Memory van job
alle tasks van

Basic objective = class scheduler → white paper

SCHEDULING OVERVIEW

SCHEDULING CLASSES

Currently, jobs can be divided into three classes: system, batch, and interactive. Scheduler's class attribute table is used to delineate the classes.

Low, high, and initial priorities are defined as are memory values. The exclude class flag will inhibit the initiation of jobs from this class. The self terminating capability will allow queued jobs of a class to be initiated even though the maximum active jobs for that class have been exceeded. The job will be up long enough to bring itself down. Currently interactive class jobs have this capability. — drop byv.

SWAP CONTROL

a) The maximum number of swapped jobs in a class.

b) The maximum overall number of swapped jobs.

Swapping is initiated as a result of three conditions:

a) If the scheduler determines that the system is thrashing, a candidate will be chosen and swapout will be performed. The two rules given above will be overridden.

b) The scheduler will periodically examine the input and active (swapped) job queues. If a job in input has a higher priority than one executing, a swap request will be issued for the active job. This swap request obeys the two parameters governing the swap function.

Vrij memory gebruiken — als het kan

c) If memory contention is high and a terminal break occurs, that job will be swapped.

page aging interval adjustment.

→ alles weg ook uit mainframe wired

PRIORITY ADJUSTMENT

Job priority adjustment is limited to aging queued jobs, aging swapped jobs, and adjusting the priorities of executing jobs.

The aging function will increment job priority based on values local to the class. There are two aging increments for each class: one for input list and the other for swap list. The aging function will be performed on a periodic basis.

Executing jobs will have their priorities adjusted according to several factors. If the job has just been swapped in, it will be given a priority boost to prevent it from being swapped out immediately. If the job's ready task count falls to zero, it will lose priority points. (This may or may not initiate swapping.) If the job's time or memory limits are exceeded, it will be switched to another internal class. Currently there are secondary interactive and batch classes.

When a swapped job receives a signal, the scheduler will increase that job's priority which will result in the job being activated sooner.

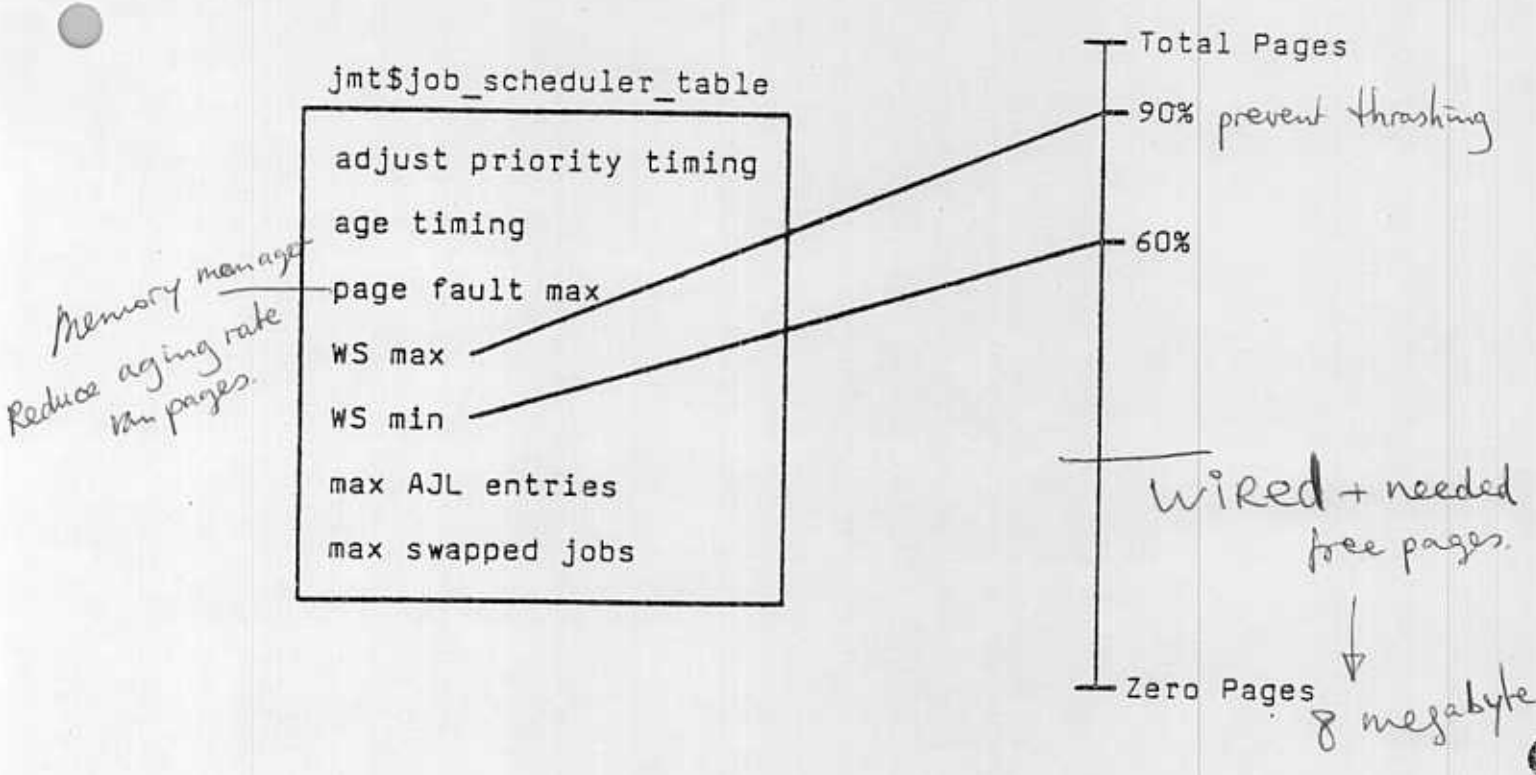
WORKING SET

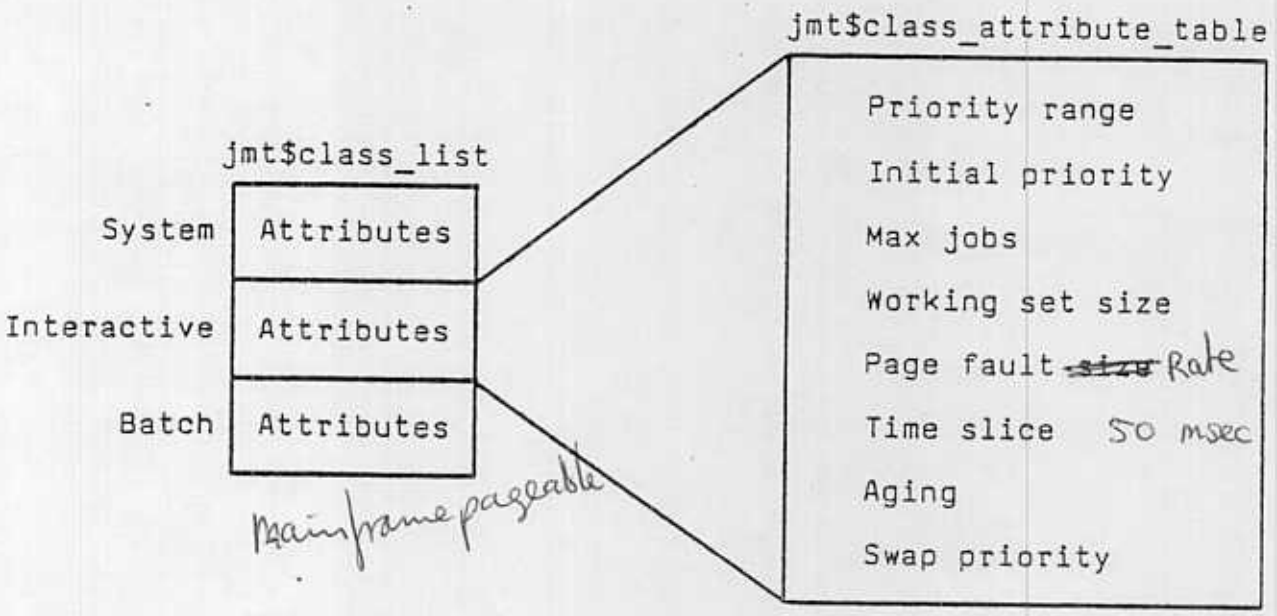
The task working set is the collection of pages that are assigned to a particular task at a particular time. Pages are aged by memory manager and the least recently used pages are removed from the task WS. When a new page is needed, the OS finds a free page and adds it to the task WS. Thus the task WS vacillates. For best performance the WS should be fairly constant. This depends on certain programming techniques and memory manager's aging algorithm. See Ch. 13 for more detail.

The job working set is the collection of all the working sets of all the tasks in the job plus the pages shared by the job (e.g., job fixed). When a job is swapped, the job working set is swapped.

page aging per ~~page~~ job.

SCHEDULING TABLES





SCHEDULING PROCESS

1. Check for Thrashing

   - Add Working Set (ws) from all AJL entries.
   - If the sum is in the thrashing range, swap jobs til the sum is out of that range. Start with the job with largest ws.
   - Stop.

2. Check page fault rate (R2)

   - If page fault rate > page fault max in jmt$job_scheduler_table, increase memory manager's aging interval.

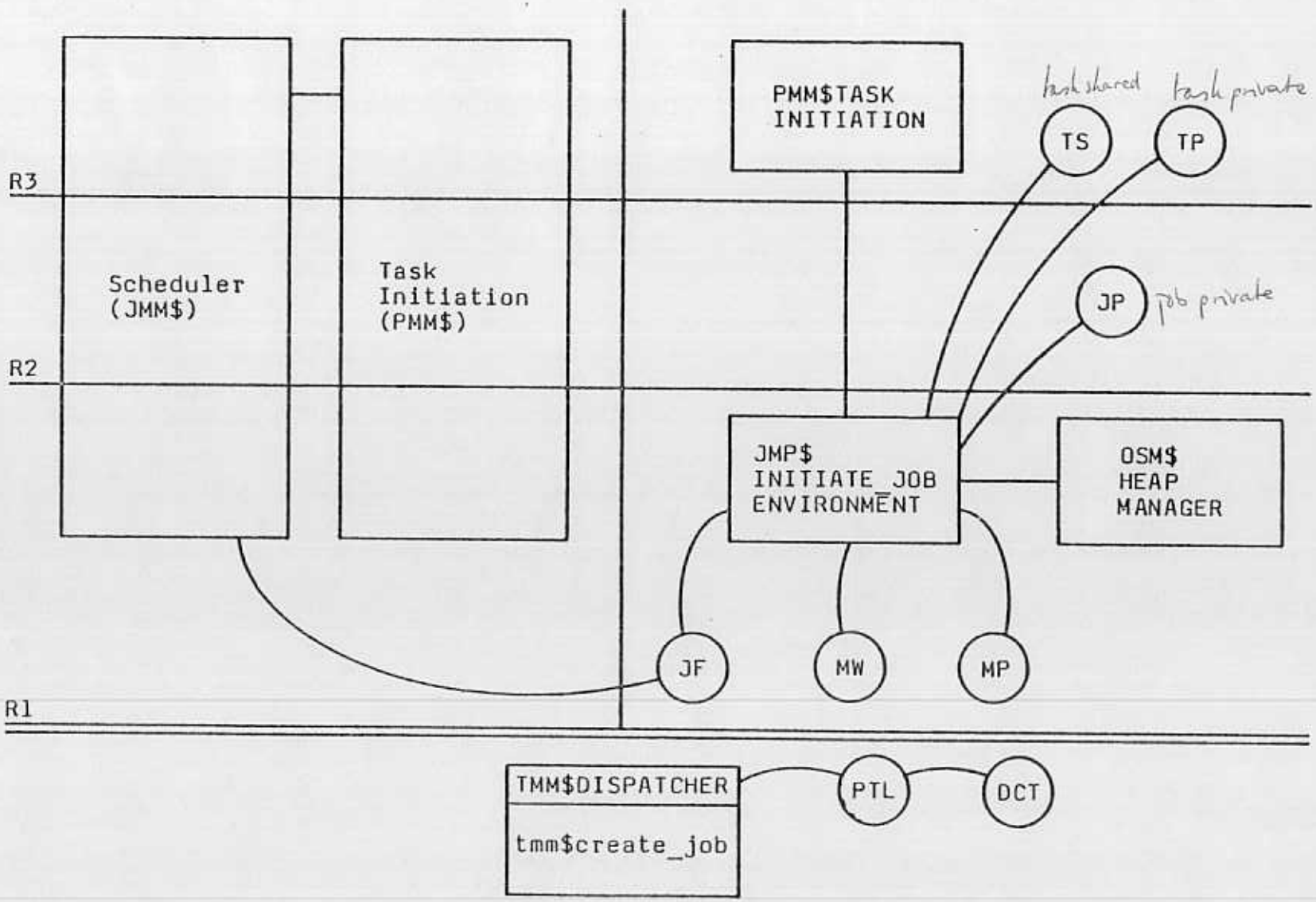
3. Fill Free Memory

   - Built temporary queues for each state (active, queued, swapped) for each class (batch, interactive, system, etc.).
   - Calculate the number of free pages between the current value and ws_max.
   - Select the algorithm (proc). The only R1 algorithm gets the highest priority queued job from each class and compares it with the highest priority swapped job. If the queued job wins it is initiated, otherwise swap. Continue until ws_min $\leq$ ws $<$ ws_max.
   - Stop.

in JOB fixed een SDT van mainframe wrld

↓

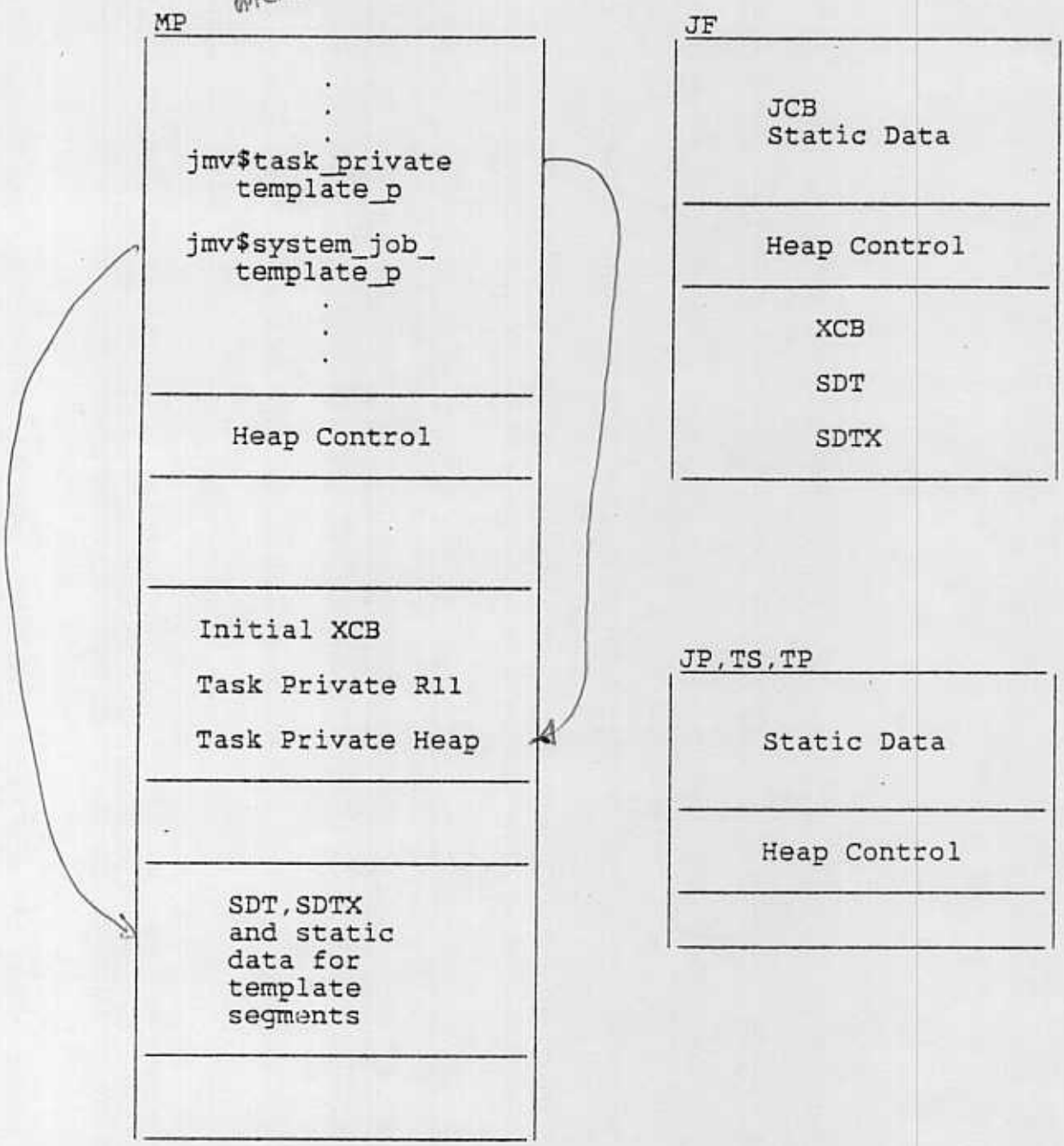
entry in PTL (task erby) DCT (dispatchable)



3 outboard symbol tables

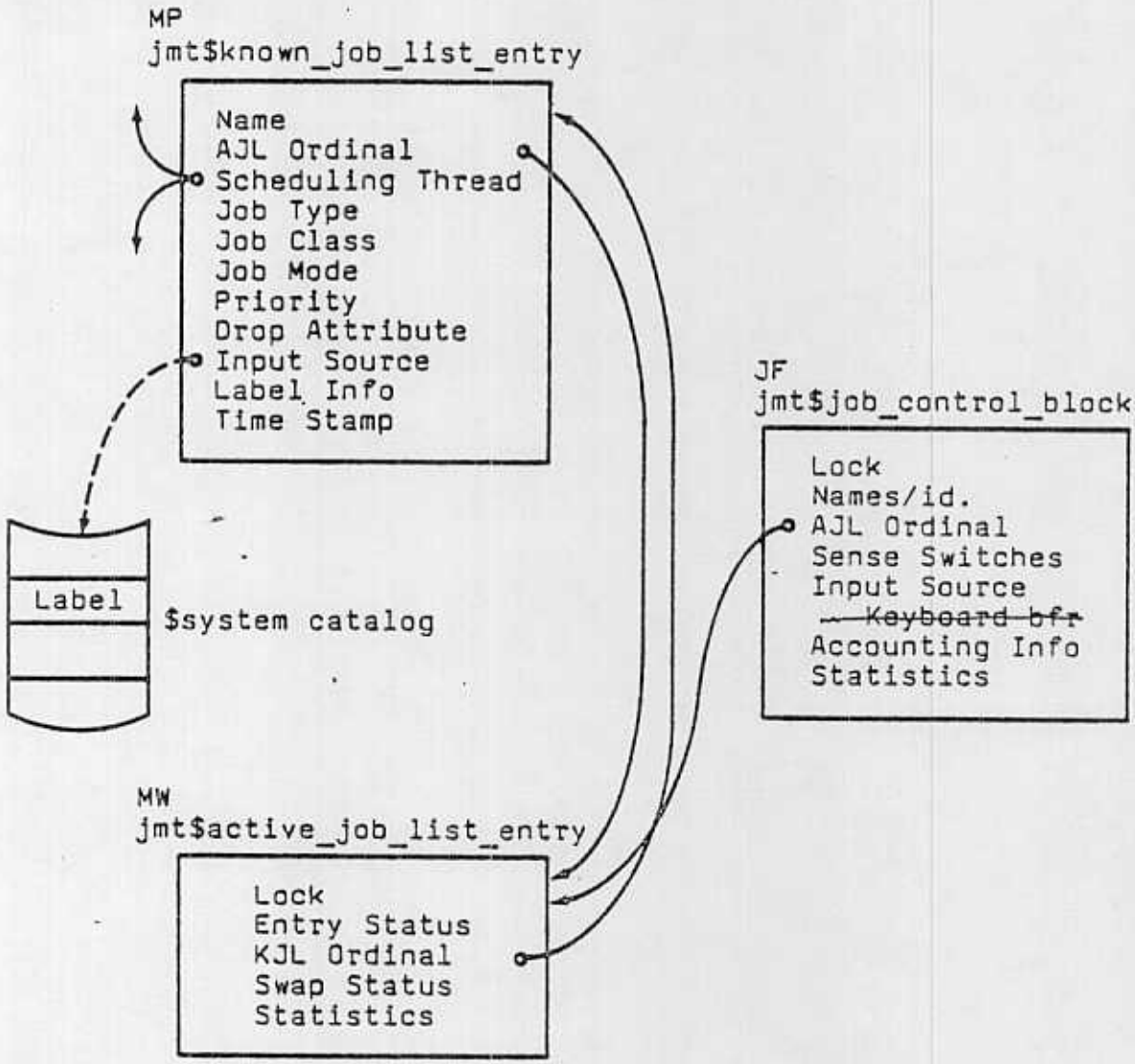
JOB TEMPLATES = renaming symbols

inherited segment

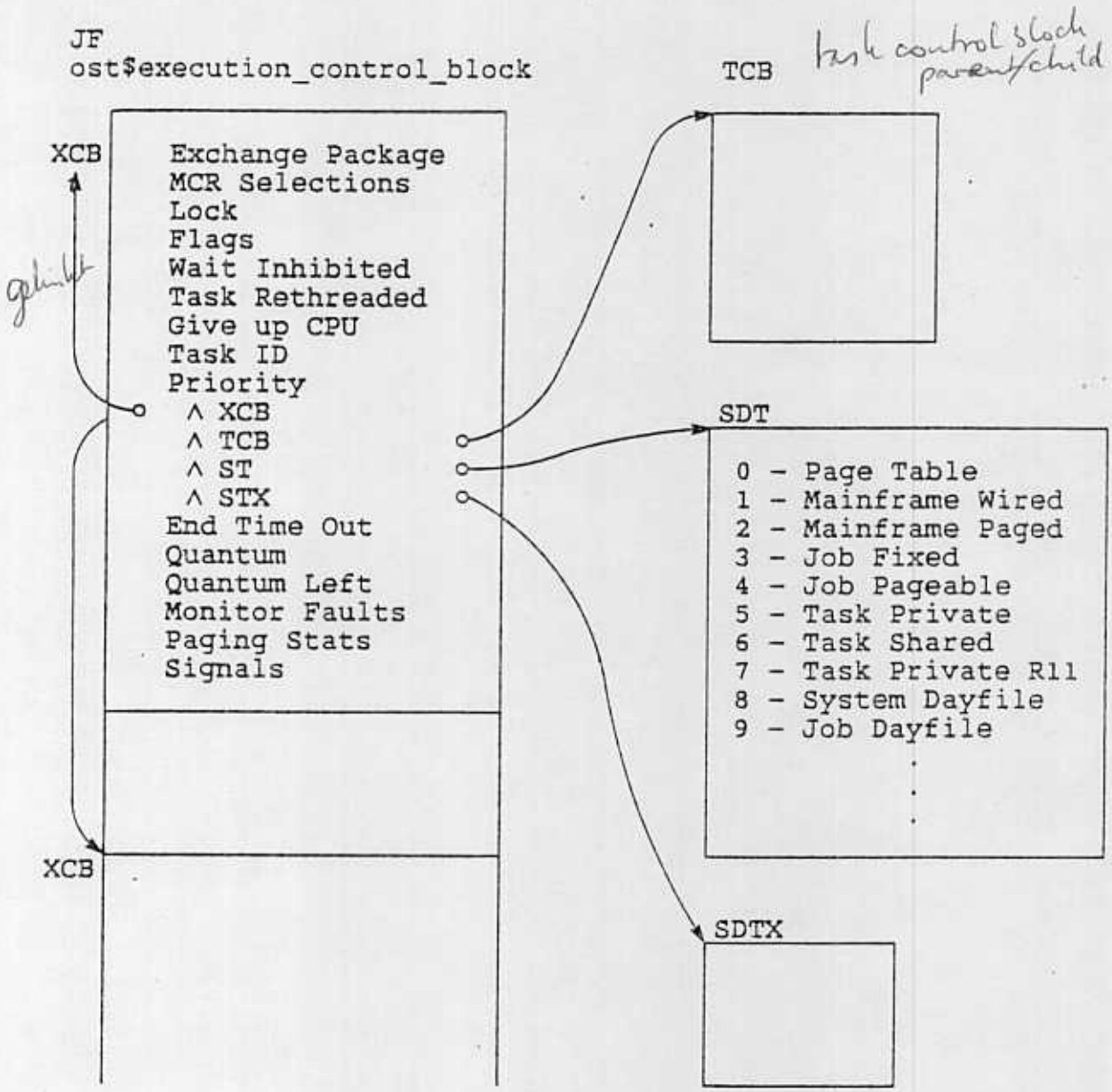


Scheduler creates all segments.
Scheduler initializes Job Fixed.
Initialize_Job_Environment initializes other segments.

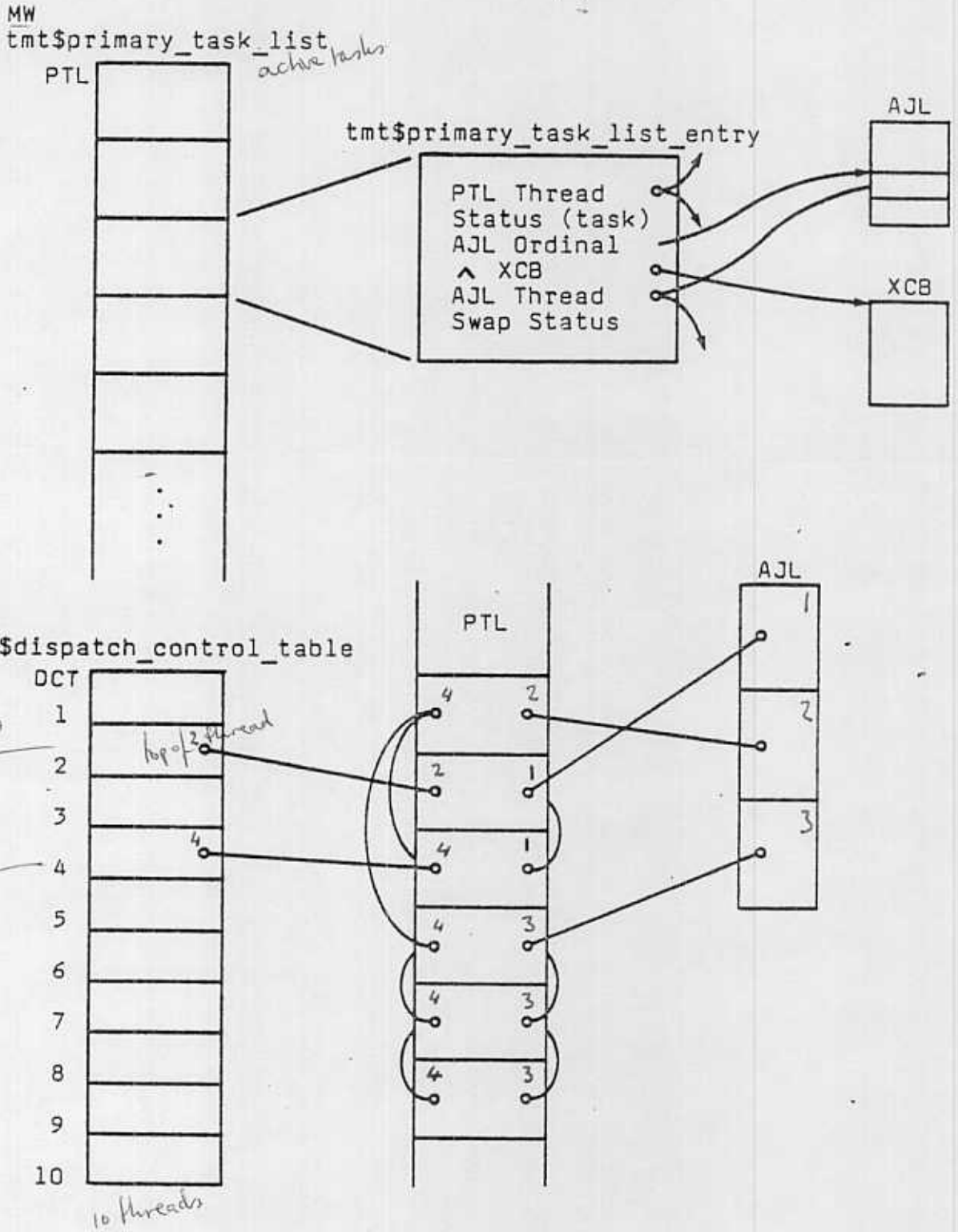
JOB CONTROL TABLES



EXECUTION CONTROL BLOCK



TASK DISPATCHING TABLES



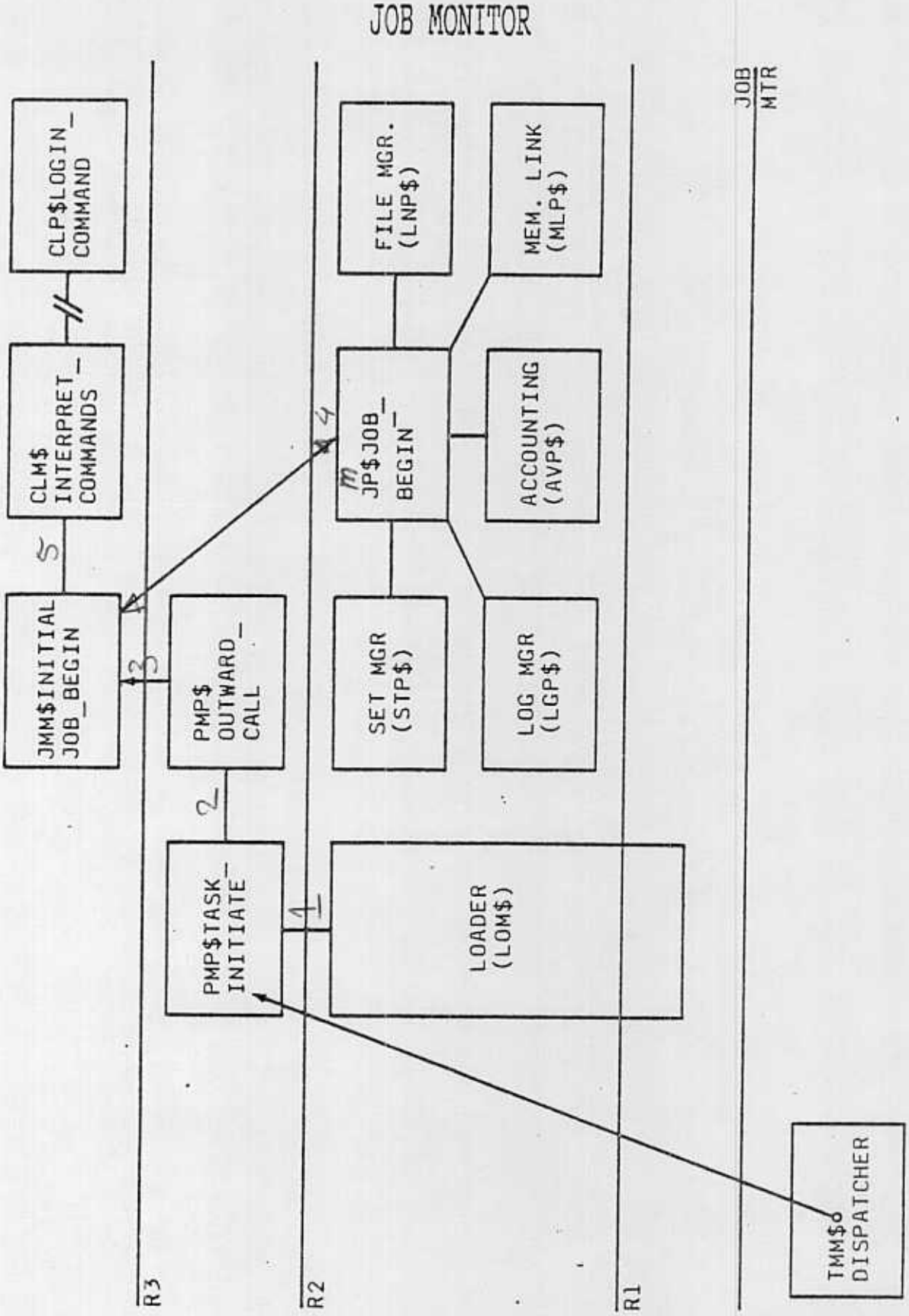
TASK DISPATCHING

* Currently (R1) all tasks are on DCT thread 4 unless they have a system table locked. Tasks with a system table locked are put on thread 2, and the rethreaded field in the XCB is marked true.

* All tasks on the highest priority thread get 50 m-sec time slice in a round robin fashion as long as there are active tasks on that thread.

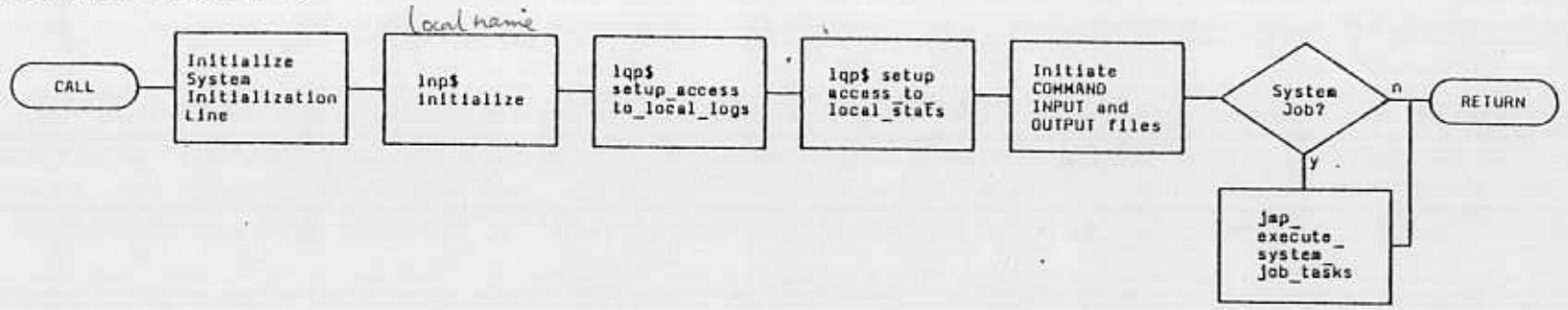
* In future releases, all 10 threads will be used. Different threads will have different time slices. These algorithms have not been defined yet.

* NOS - NOS/VE scheduling is done in NOS and MIP. If the current NOS job has higher priority, NOS runs; if the current NOS/VE task has higher priority, NOS/VE runs. If the priorities are equal (NOS job default = NOS/VE task = 30) then the CPU is toggled between states. Currently 50 ms are awarded to each side but that can be changed to favor one side or the other. NOS trap handler does the timing. Idle is in NOS.

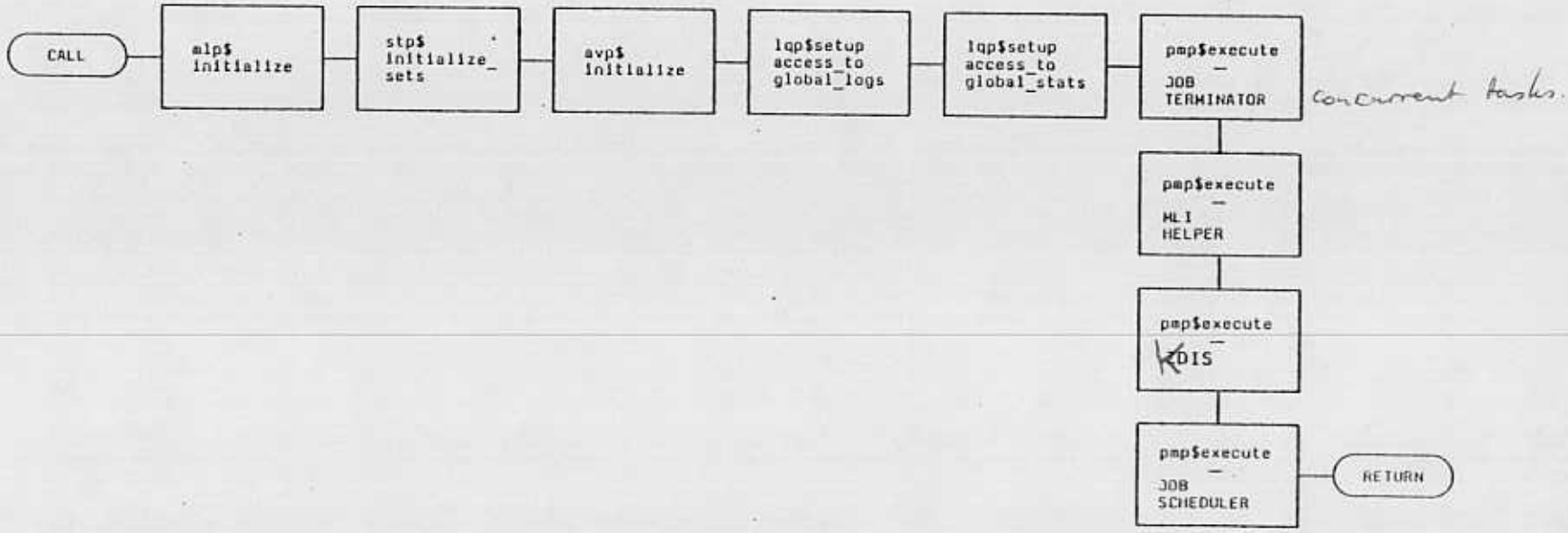

JOB MONITOR
CLP$LOGIN_ COMMAND
CLM$ INTERPRET_ COMMANDS
JMM$INITIAL JOB_BEGIN
PMP$ OUTWARD_ CALL
PMP$TASK INITIATE
LOADER (LOM$)
SET MGR (STP$)
LOG MGR (LGP$)
JP$JOB_ BEGIN
FILE MGR. (LNP$)
ACCOUNTING (AVP$)
MEM. LINK (MLP$)
JOB MTR
TMM$0 DISPATCHER
R3
R2
R1
1
2
3
4
5

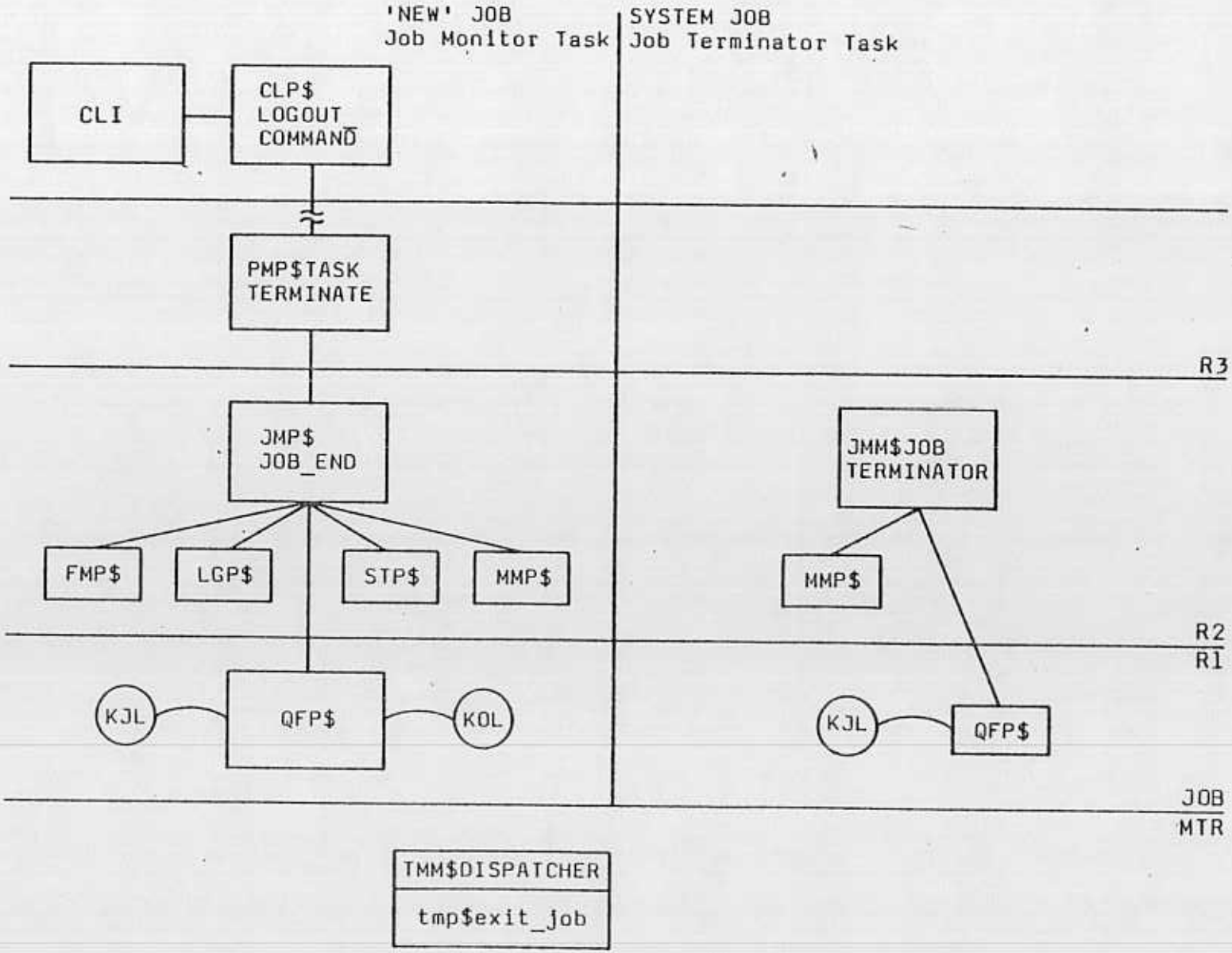
23 D

jmm$JOB_MONITOR (2, 2, 3)

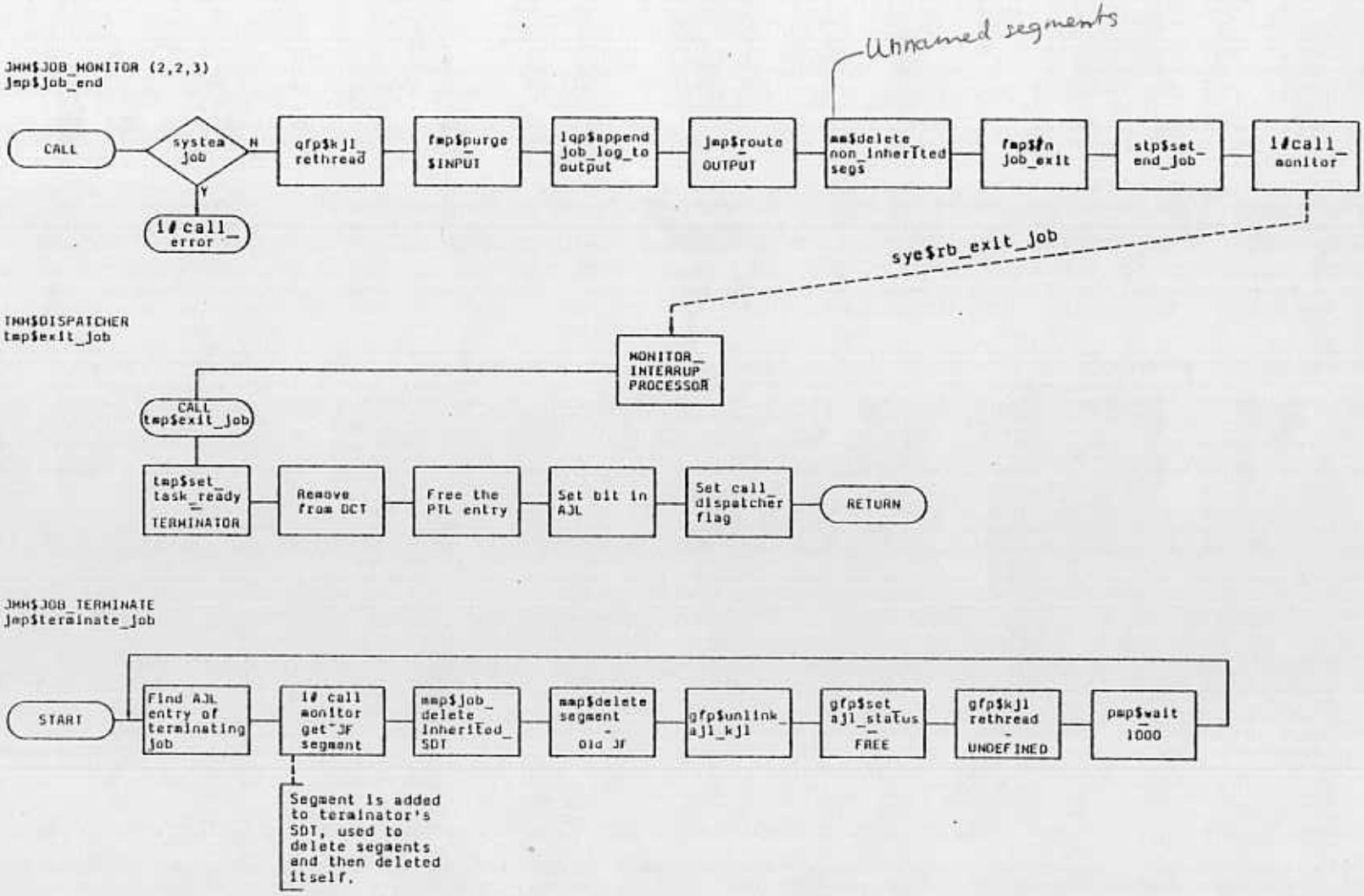


jmp_execute_system_job_task

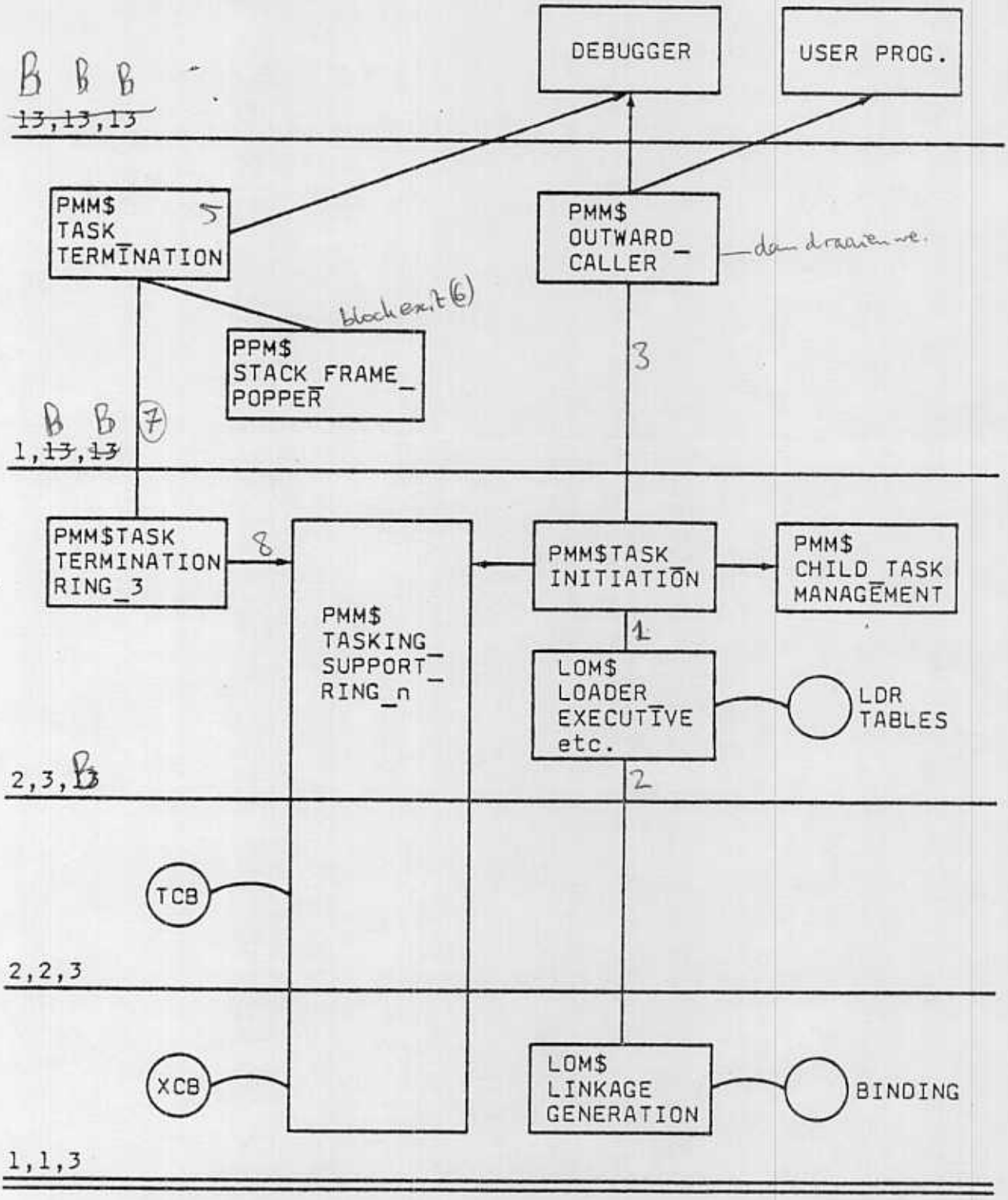




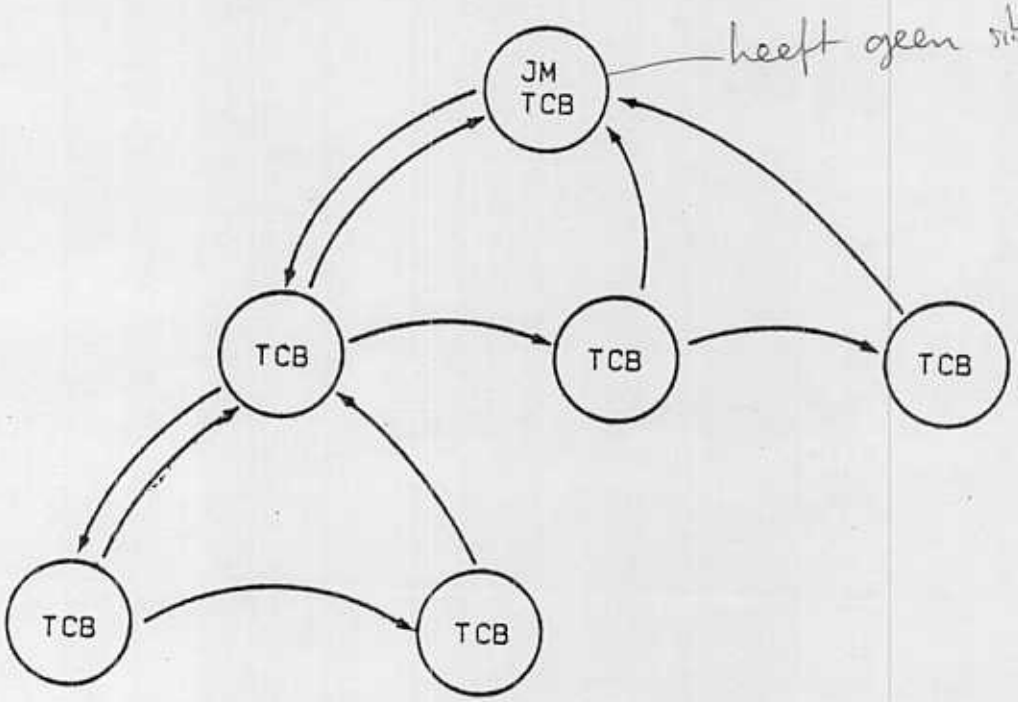
JOB TERMINATION


JMH$JOB_MONITOR (2,2,3)
jmp$job_end
Unnamed segments
CALL
system job
N
Y
l#call_error
qfp$kjl_rethread
fmp$purge_$INPUT
lqp$append job_log_to output
jmp$route_OUTPUT
mm$delete non_inherited segs
fmp$fn job_exit
stp$set_end_job
l#call_monitor
sys$rb_exit_job
JMH$DISPATCHER
tmp$exit_job
MONITOR_INTERRUP PROCESSOR
CALL tmp$exit_job
tmp$set_task_ready_TERMINATOR
Remove from DCT
Free the PTL entry
Set bit in AJL
Set call_dispatcher flag
RETURN
JMH$JOB_TERMINATE
jmp$terminate_job
START
Find AJL entry of terminating job
l# call monitor get JF segment
mmp$job_delete_inherited_SDT
mmp$delete segment_Old JF
gfp$unlink_ajl_kjl
gfp$set_ajl_status_FREE
gfp$kjl rethread_UNDEFINED
pmp$wait_1000
Segment is added to terminator's SDT, used to delete segments and then deleted itself.

PROGRAM CONTROL AND LOADER



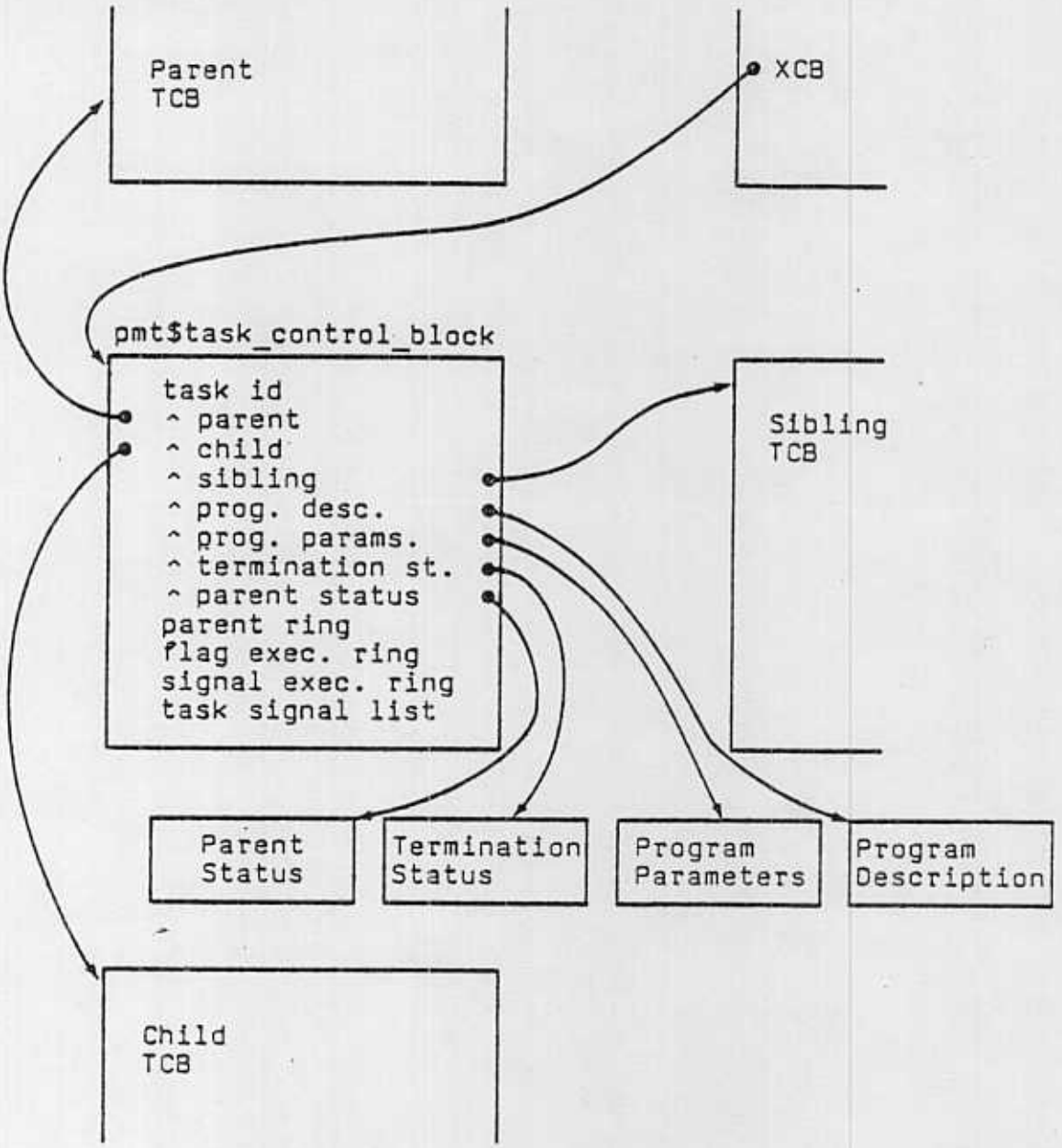
PARENT/CHILD/SIBLING — is broer of zuster



draad alleen naar siblings aflopen.

kind. doet abort → rethread.

TASK CONTROL BLOCK



PARENT/CHILD REQUESTS

PMP$VERIFY_CURRENT_CHILD (tid,current)

PMP$SIGNAL_ALL_CHILD_TASKS (signal,status)

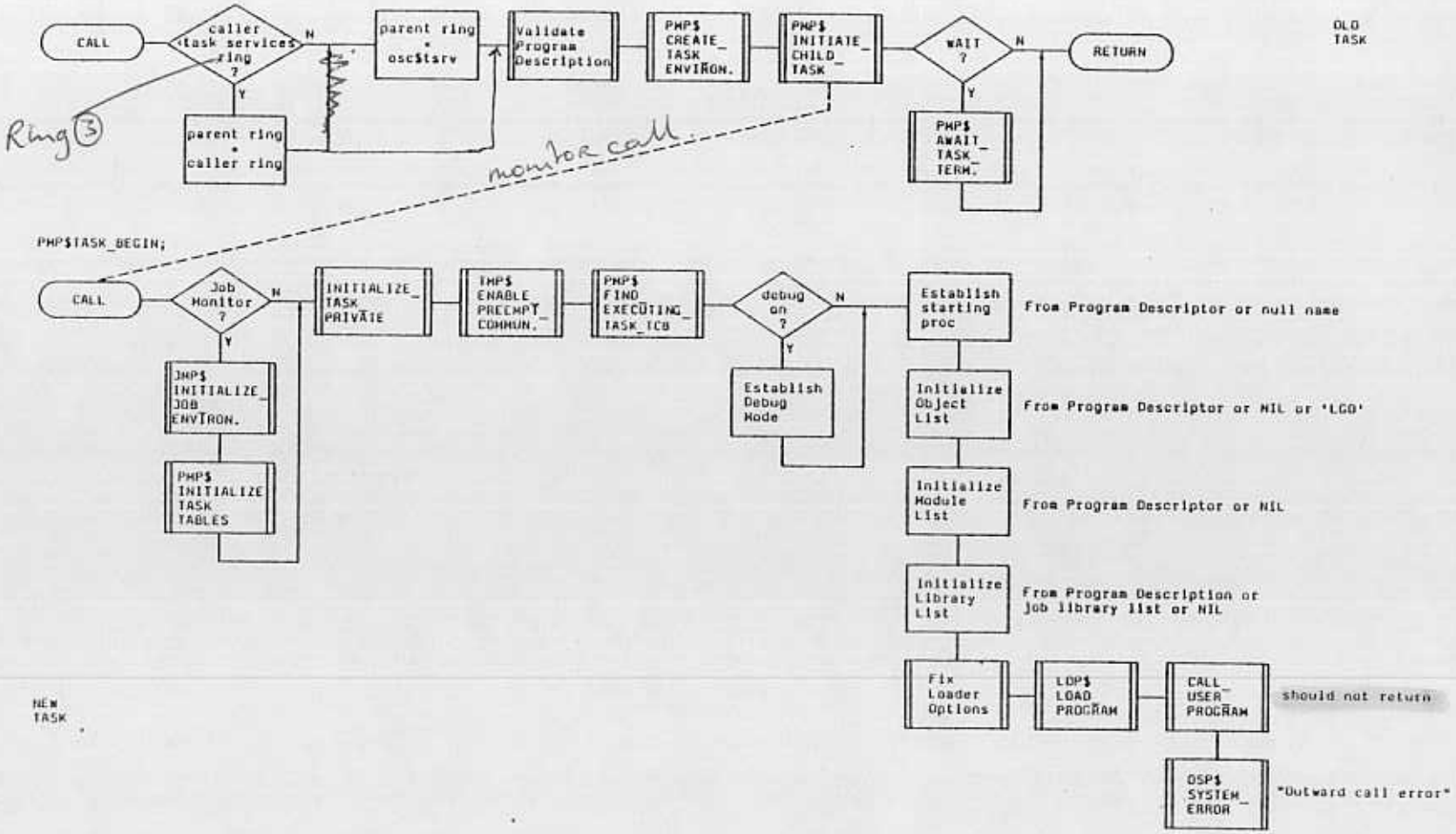
PMP$FLAG_ALL_CHILD_TASKS (flag,status)

PMP$REVOKE_PROGRAM_TERMINATION

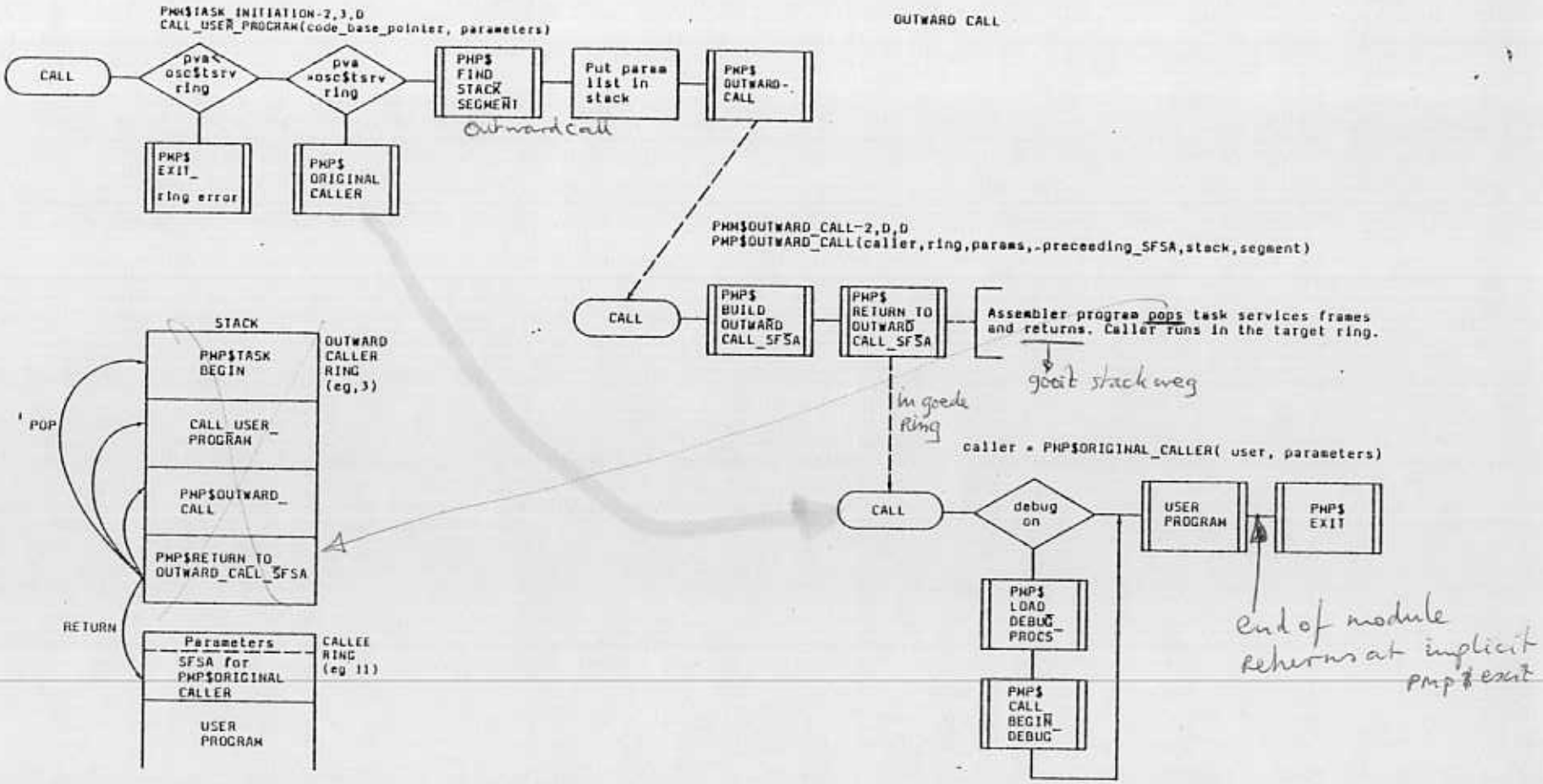
↳ opnieuw opstarten van termination (max 1000 keer

TASK INITIATION

PHM$TASK_INITIATION 2,3,0
PHP$EXECUTE(prog_desc,prog_parameters,wait,task_id,task_status,status);



TASK INITIATION (Continued)



TASK TERMINATION LEVELS

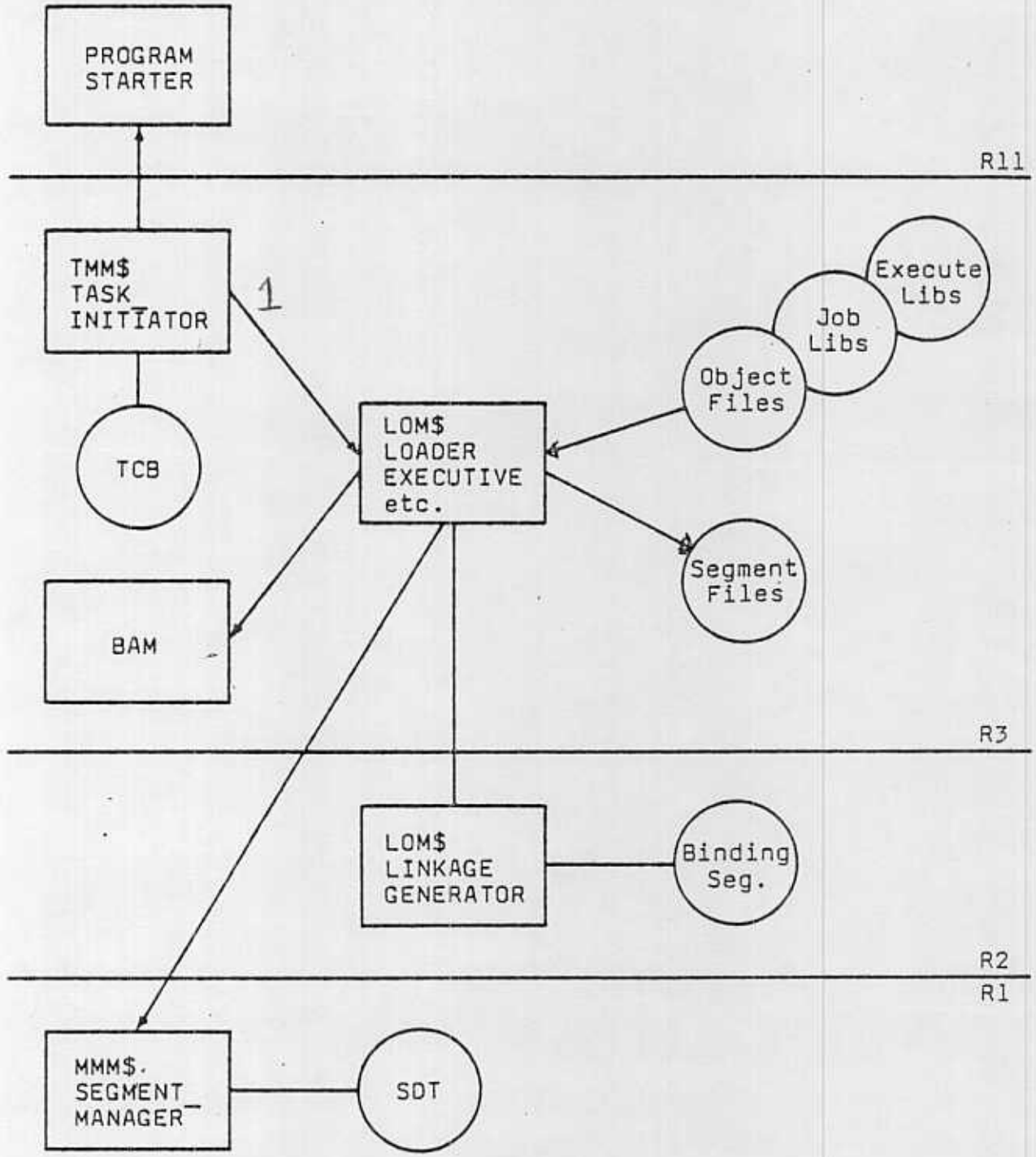
1. Unwinding
   - o Revoke program termination (Debugger)
   - o Pop stack frames--block exit processing
   - o Close files at each 'active ring' to ring 3
   - o Child Task Cleanup
     - Abnormal--kill all child tasks
     - Normal--await child termination
   - o Clean up task environment
   - o SCL Clean up

2. Unwinding Impossible
   - o Stack, for example, is bad
   - o Child task cleanup
   - o Clean up task environment — close door user FAP moet nog gedaan worden

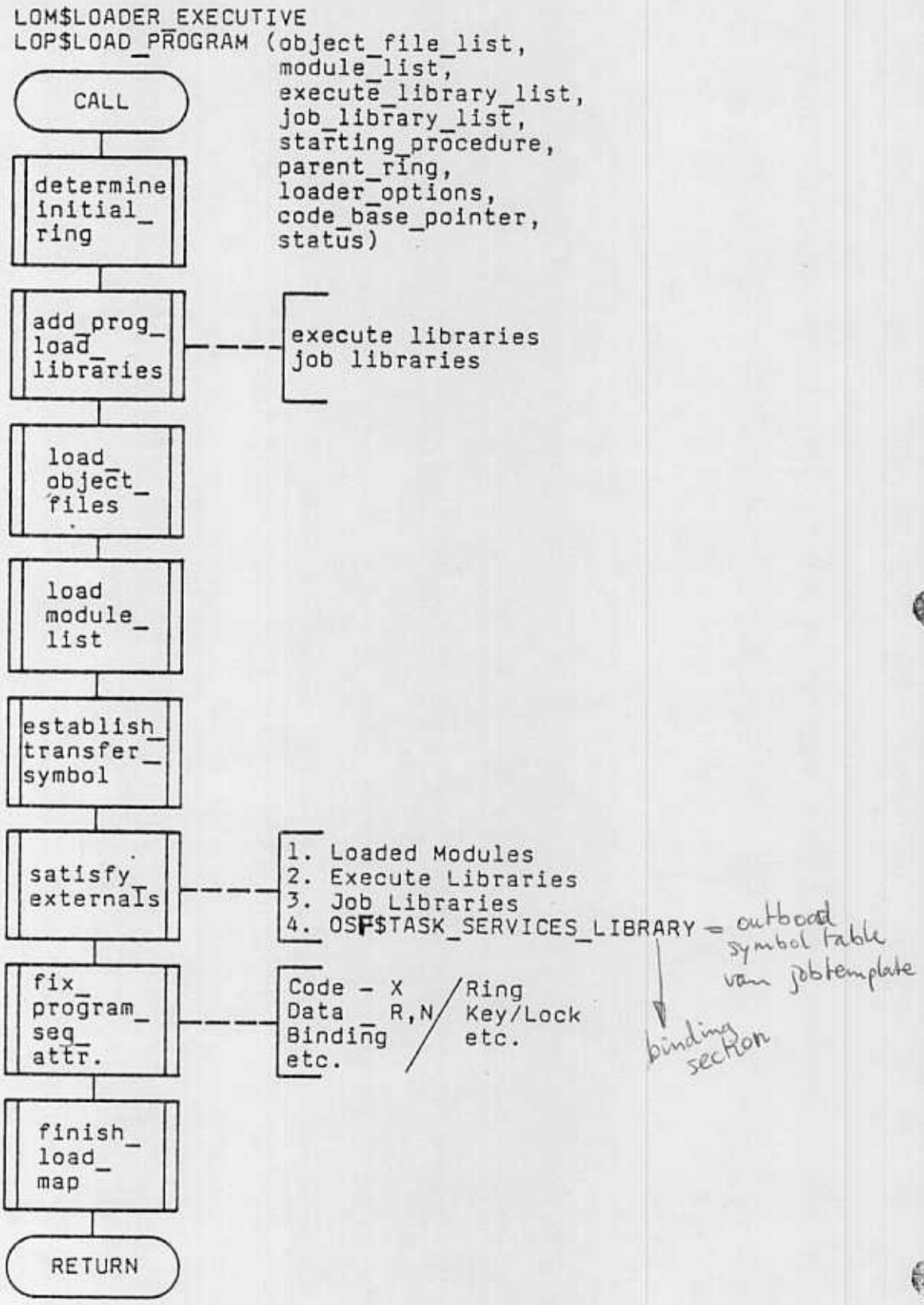
3. Broken Task
   - o Monitor detects user fault with traps disabled
   - o Fix trap handler tables to see broken task flag

4. Monitor Kill.

LOADER CONTEXT



LOADER EXECUTIVE



OBJECT MODULE INTERNAL FORMAT

* Each object module is a set of records on the object file

* The object record descriptor contains
  - o Item type
  - o Record length

* Item types

  IDR: Identification of module and attributes

  LIB: Libraries from which to satisfy external references

  SDC: Length and attributes of each section, code, working storage, binding, and all common blocks

  TEX: Text to be placed in each section

  RPL: Text to be repetitively placed in each section

  BIT: Inserts bit-level data into a section

  EPT: Defines an address in a section as an entry point

  RIF: Identifies addresses that must be relocated by the library generator when binding modules together

  ADR: Allows PVAs to be built at load time (when ring, segment number, and offset are known)

  XRL: List of external references to be satisfied

  BTI: Binding template describes the contents of a location in the binding section

  TRA: Terminates the object module and gives the primary entry point

USER
COMMAND
STREAM
(VALIDATED FOR
RING 11)

•
•
•

FTN,I=MAIN,B=LGO
FTN,I=SUB,B=LGO
LGO

•
•
•

LOCAL FILE LGO R1=11, R2=11, R3=11

IDR	• NAME • TIME & DATE CREATED • ETC.
LIB	• FTNLIB
SDC	CODE SECTION
SDC	BINDING SECTION
SDC	WORKING STORAGE SECTION
SDC	COMMON BLOCKS
TEX, RPL BIT, REL ADR, XRL EPT, BIN RECORDS FOR CODE, BINDING AND WORKING STORAGE SECTIONS	
TRA	• STARTING ADDRESS • END OF MODULE
IDR	
LIB	• FTNLIB
SDC	• CODE
SDC	• BINDING
SDC	• WORKING STORAGE
SDC	• COMMON BLOCKS
TEX, RPL BIT, REL ADR, XRL EPT, BIN RECORDS FOR CODE BINDING AND WORKING STORAGE SECTIONS	
TRA	

OBJECT MODULE FOR MAIN

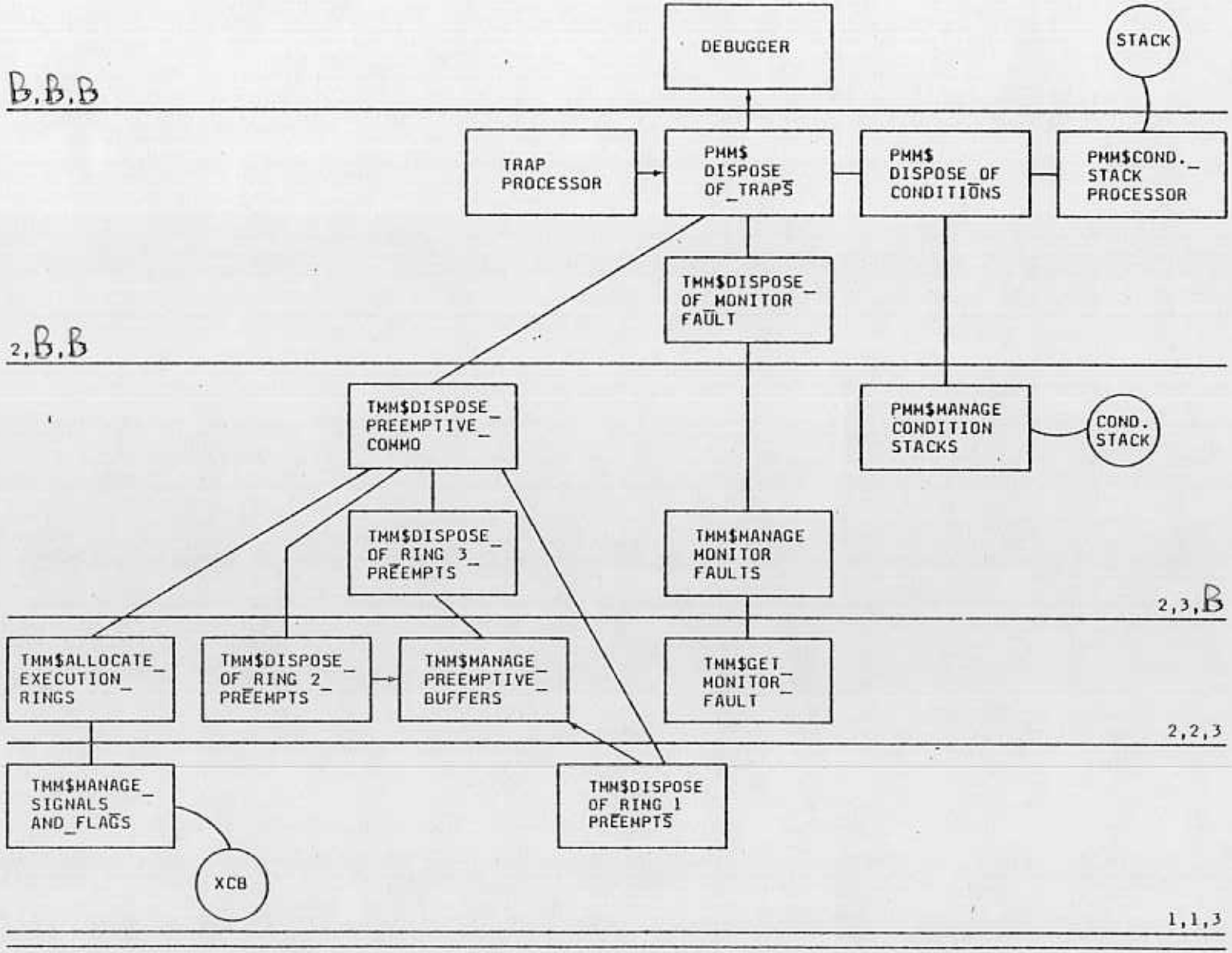
OBJECT MODULE FOR SUB

OBJECT LIST (1)

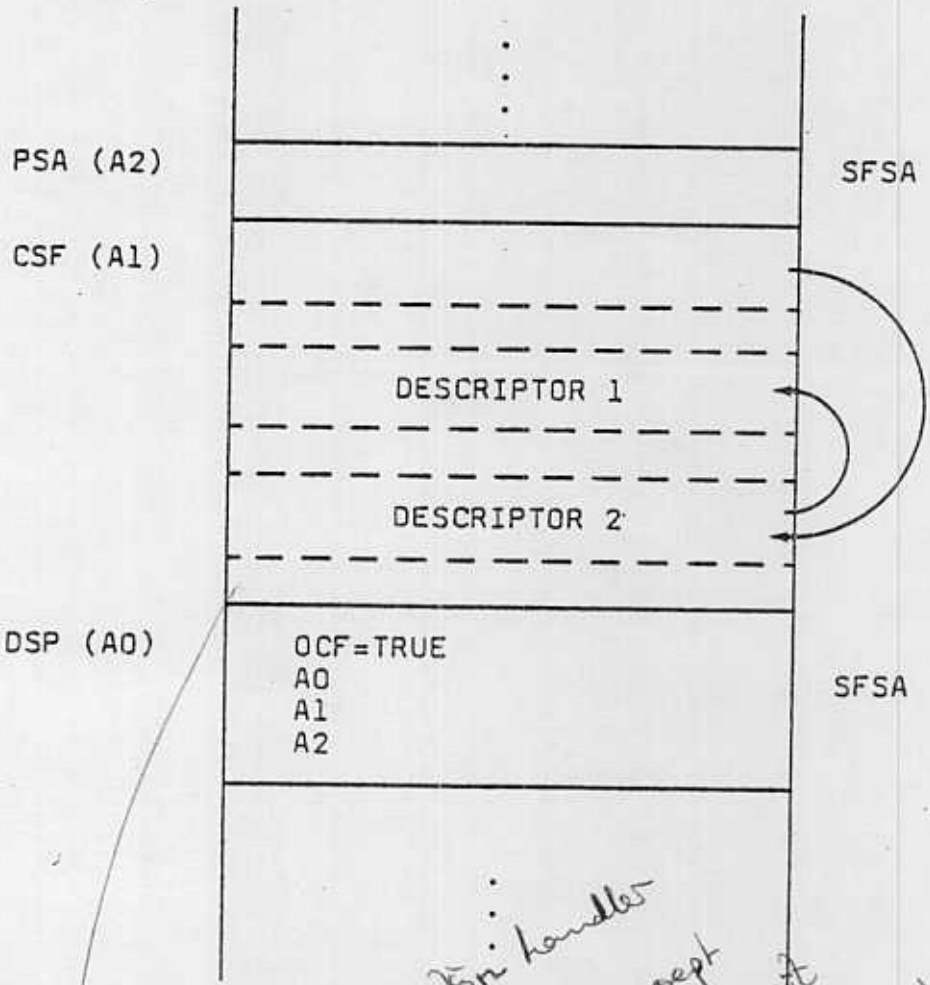
```
 1  MODULE SQACI;
 2
 3    CONST
 4      MIN = 1,
 5      MAX = 30;
 6
 7    TYPE
 8      T_ARRAY = ARRAY [MIN .. MAX] OF INTEGER;
 9
10    PROCEDURE SQUARER (B: T_ARRAY;
11      VAR SQ: T_ARRAY);
12
13      VAR
14        J: MIN .. MAX;
15
16      FOR J := MIN TO MAX DO
17        SQ [J] := B [J] * B [J];
18      FOREND;
19    PROCEND SQUARER;
20
21    PROGRAM MAIN;
22
23      VAR
24        BASE: T_ARRAY,
25        SQUARE: T_ARRAY,
26        I: MIN .. MAX;
27
28      FOR I := MIN TO MAX DO
29        BASE [I] := I;
30      FOREND;
31      SQUARER (BASE, SQUARE);
32    PROCEND MAIN;
33  MODEND SQACI;
```

```
/ses.cybil sqaci b*bbb ci
*    COMPILING  SQACI
*    END CYBIL    SQACI -> LISTING, BBB
/ses.objlist bbb
1IDR  RN= 1  SQACI                                    V1.2 MVS 10:02:14 10/13/81
      GREATEST SECTION ORD= 2  GEN ID=CYBIL
      GEN NAME VERS=C180 CYBIL 1.0                    LEVEL 81188
      COMMENTARY=

 LIB  RN= 2  CYBILIB

 SDC  RN= 3  KIND=CODE     ATTRIBUTES=RX ORDINAL= 0 LENGTH=00000118
      OFFSET= 0 ALIGNMENT= 8

 SDC  RN= 4  KIND=BINDING  ATTRIBUTES=RB ORDINAL= 1 LENGTH=00000020
      OFFSET= 0 ALIGNMENT= 8

 SDC  RN= 5  KIND=WORKING  ATTRIBUTES=R ORDINAL= 2 LENGTH=0000001F
      OFFSET= 0 ALIGNMENT= 8

 EPT  RN= 6  SECTION= 0  OFFSET=000000B8  ATTRIBUTES=
        NAME=MAIN                                     LANGUAGE=CYBIL
      DECLARATION MATCHING:  REQUIRED-TRUE VALUE-0A9A10EC520777

 BTI  RN= 7  BINDING OFFSET=0000000A  CURRENT MODULE
        SECTION= 2  OFFSET=00000000  KIND=POINTER

 ADR  RN= 8  VALUE SECTION= 2  DEST. SECTION= 1
      KIND=POINTER   VALUE OFFSET=00000000  DEST. OFFSET=0000000A

 BTI  RN= 9  BINDING OFFSET=00000010  EXTERNAL REF
        NAME=CYP$ERROR                                ADDRESS=EXT PROC

 EXT  RN= 10  NAME=CYP$ERROR                                LANGUAGE=CYBIL
      DECLARATION MATCHING:  REQUIRED-FALSE VALUE-0000000000000000
      SECTION ORDINAL= 1 OFFSET=00000010  KIND=EXT PROC  OFFSET OPERAND=00000000
```

```
RIF  RN= 11  SECTION= 0 OFFSET=000000A6 R-SECTION= 1 CONTAINER=Q FIELD  ADDRESS=BYTE-

RIF  RN= 12  SECTION= 0 OFFSET=000000B2 R-SECTION= 1 CONTAINER=Q FIELD  ADDRESS=WORD-

TEX  RN= 13  LENGTH=001F SECTION= 2 OFFSET=00000000
     53514143  49202020  20202020  20202020  20202020  20202020
     20202020  202020

TEX  RN= 14  LENGTH=00B4 SECTION= 0 OFFSET=00000000
     8E100020  84450000  09068E00  00F08516  00127656  09F00000
     09F00000  3D128D03  001E8D0E  00103D14  96420036  96230034
     83120003  D015001F  84450008  A9560003  8D5E0011  96450025
     96530023  D015001F  84160012  A9570003  9645001B  96530019
     D015001F  84170012  A9580003  96450011  9653000F  1187D765
     70001188  D7778000  26571186  DF576000  9C32FFCD  04323D2D
     94000003  3D3D0905  8E000018  835D0000  835E0001  8436000A
     85560012  B30000FF  B5350002

TEX  RN= 15  LENGTH=005A SECTION= 0 OFFSET=000000B8
     8E100208  3D128D03  001E8D0E  001C3D14  9642FFE6  9623FFE4
     8312003E  D01501F7  A9560003  8D5E001D  9645FFD7  9653FFD5
     D01501F7  DF156008  9C32FFE9  8E1501F8  8E160010  85560000
     8E160100  85560008  8D000020  B035FFDF  0435

TRA  RN= 16  NAME=MAIN
```


DEBUGGER
STACK
B,B,B
TRAP PROCESSOR
PHM$ DISPOSE_ OF_TRAPS
PHM$ DISPOSE_OF CONDITIONS
PHM$COND._ STACK PROCESSOR
TMM$DISPOSE_ OF_MONITOR FAULT
2,B,B
TMM$DISPOSE_ PREEMPTIVE_ COMMO
PHM$MANAGE CONDITION STACKS
COND. STACK
TMM$DISPOSE_ OF_RING_3_ PREEMPTS
TMM$MANAGE MONITOR FAULTS
2,3,B
TMM$ALLOCATE_ EXECUTION_ RINGS
TMM$DISPOSE_ OF_RING_2_ PREEMPTS
TMM$MANAGE_ PREEMPTIVE_ BUFFERS
TMM$GET_ MONITOR_ FAULT
2,2,3
TMM$MANAGE_ SIGNALS_ AND_FLAGS
TMM$DISPOSE OF_RING_1 PREEMPTS
XCB
1,1,3

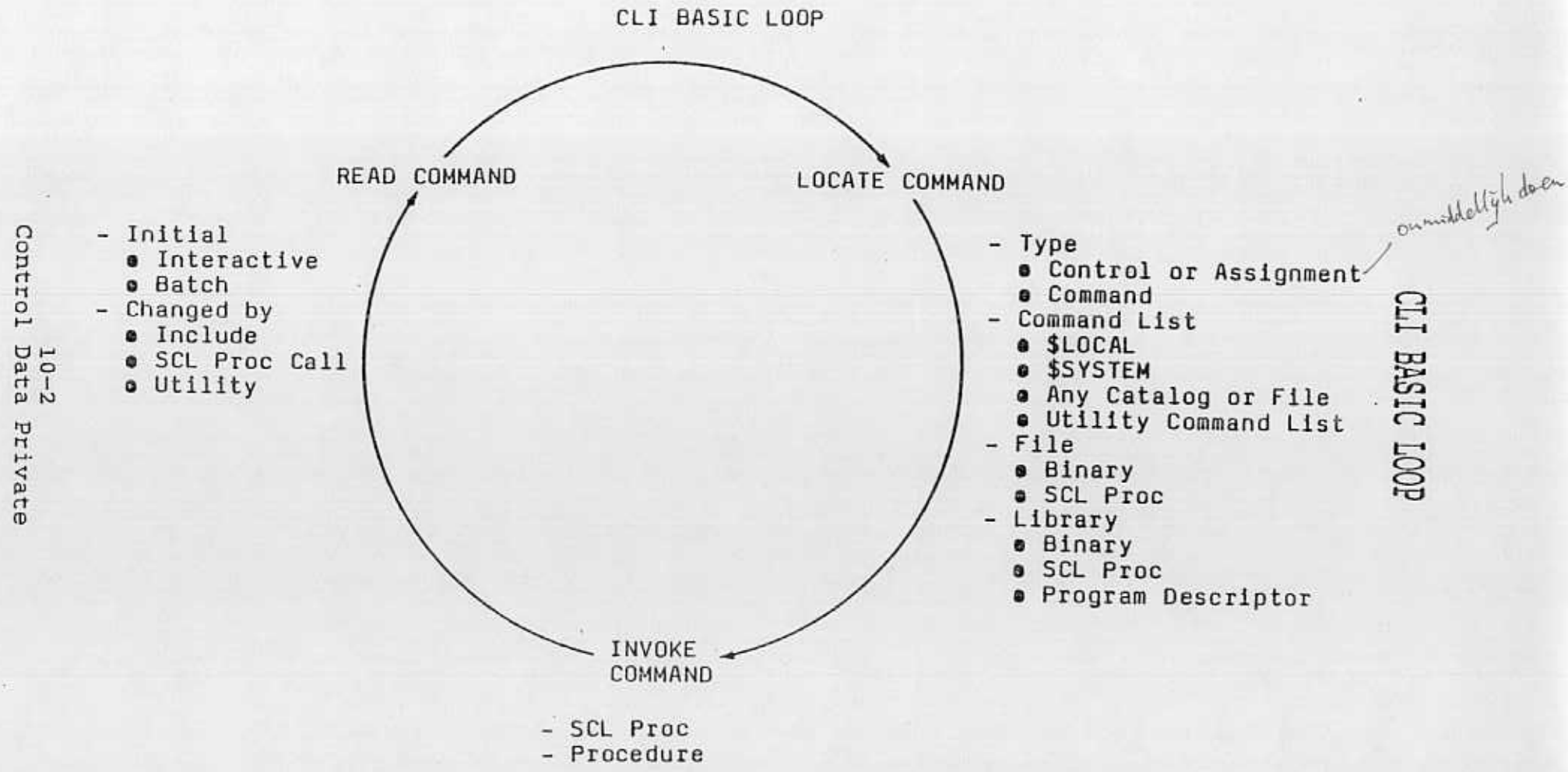
CONDITION HANDLING



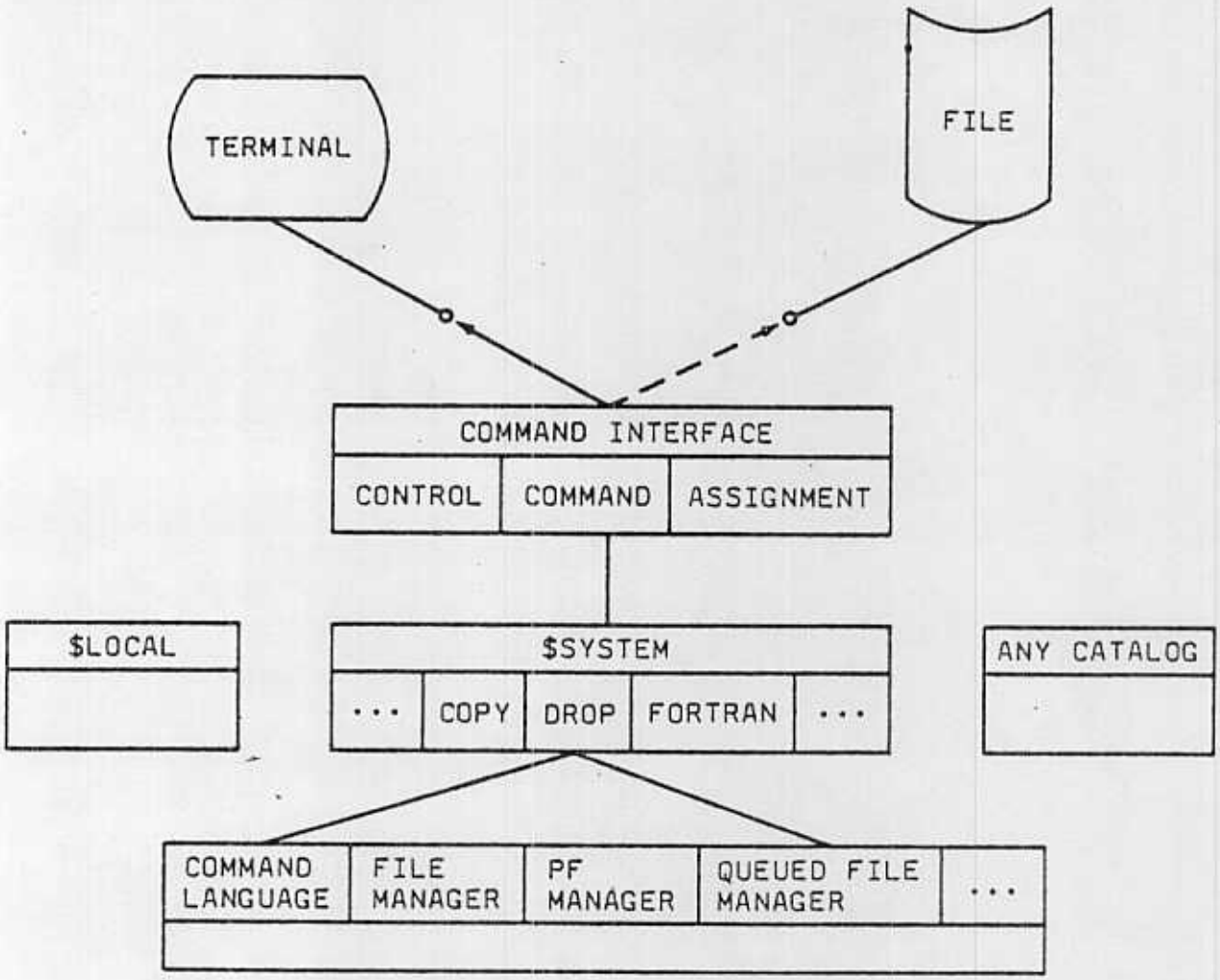
block exit → condition handler
het block waar establish aanroept
kan niet triggeren op block exit
als je de stack popt in buitenblock dan is de OCF weg.

	DESCRIPTION	SCOPE	INFO RETURNED
USER	Selector name ^handler	Current Ring	Condition Condition Descriptor passed on PMP$CAUSE_CONDITION
INTERACTIVE	Selector id:0..255 ^handler	All Rings	Condition
SYSTEM	Selector Set of MCR, UCR Loop Prevention ^handler	Current Ring	Condition Save Area of frame that caused the condition
BLOCK	Selector Set of reason CFF ^handler	Frame	Condition Save Area of frame attempting Return, Pop or non-local exit
JOB RESOURCE	Selector id:0..255 ^handler	All Rings	Condition
SEGMENT ACCESS	Selector id:0..255 Segment Number Loop Prevention ^handler	Current Ring	Condition Save Area of frame that caused the condition
PROGRAM INTERNAL TIMER	Selector ^handler	All Rings	Status
COMBINATION	Selector Set of category ^handler	-	-

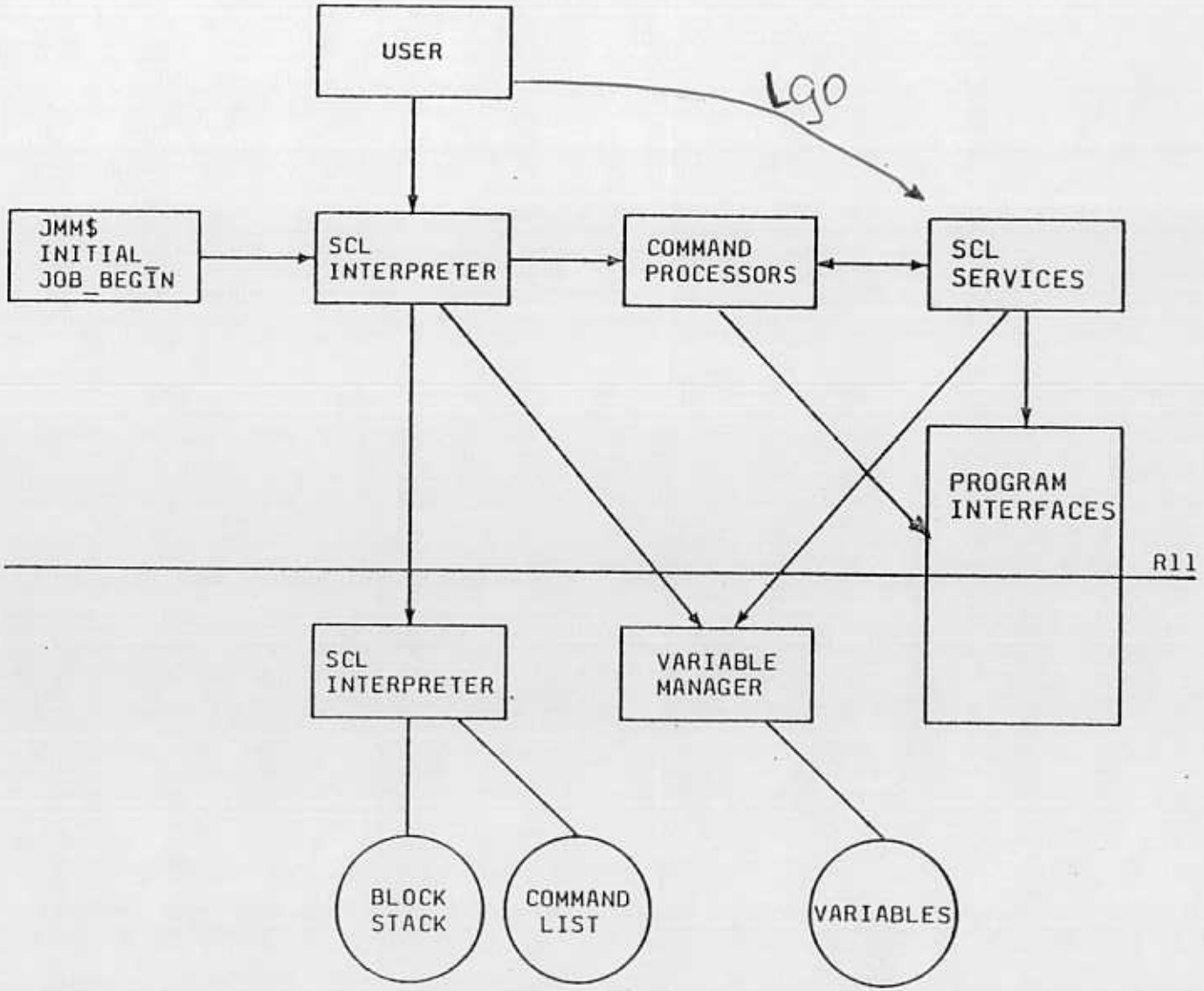
NOTE: See Program Interface

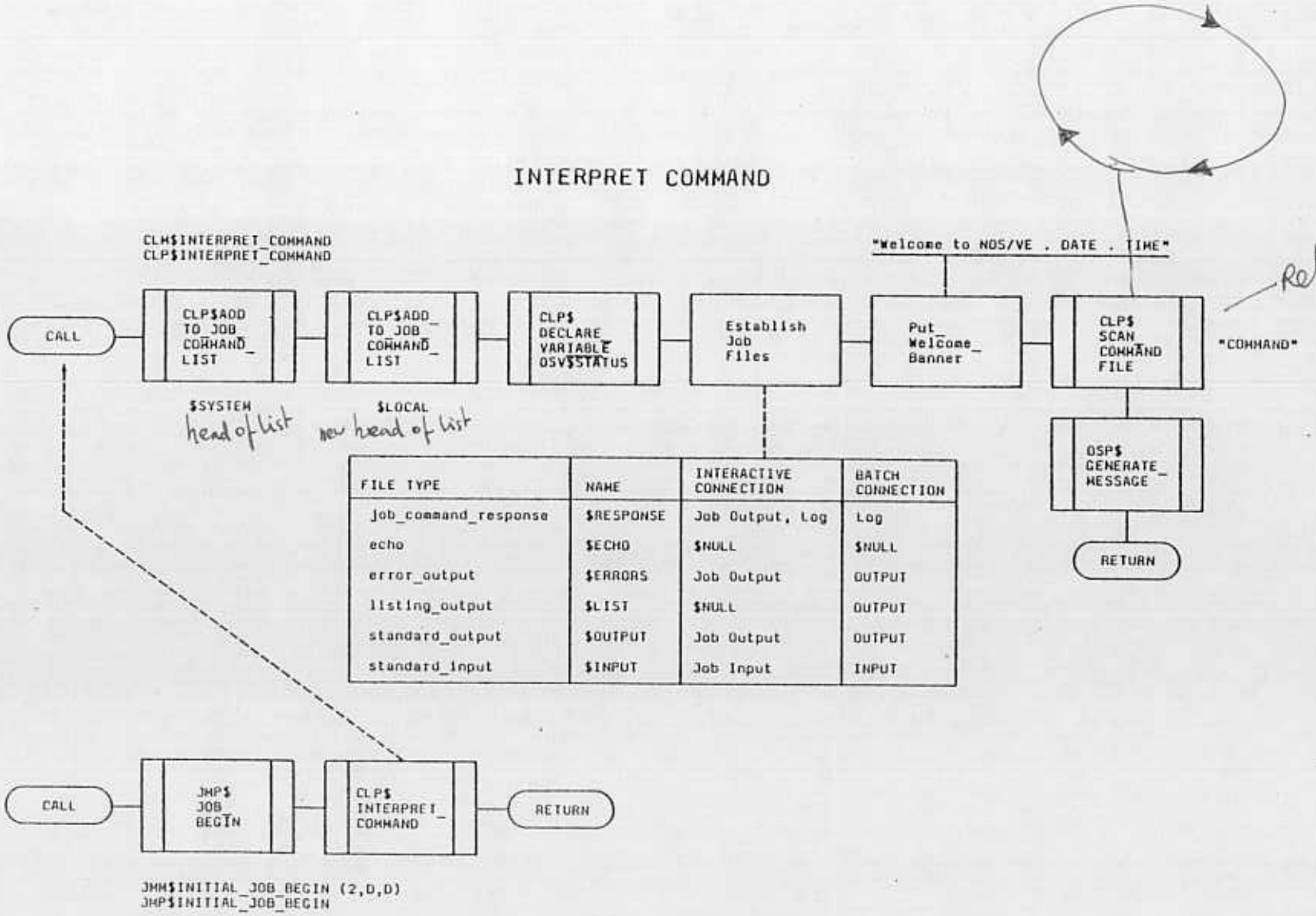

CLI BASIC LOOP
READ COMMAND
LOCATE COMMAND
INVOKE COMMAND
- Initial
• Interactive
• Batch
- Changed by
• Include
• SCL Proc Call
• Utility
- Type
• Control or Assignment
• Command
- Command List
• $LOCAL
• $SYSTEM
• Any Catalog or File
• Utility Command List
- File
• Binary
• SCL Proc
- Library
• Binary
• SCL Proc
• Program Descriptor
- SCL Proc
- Procedure

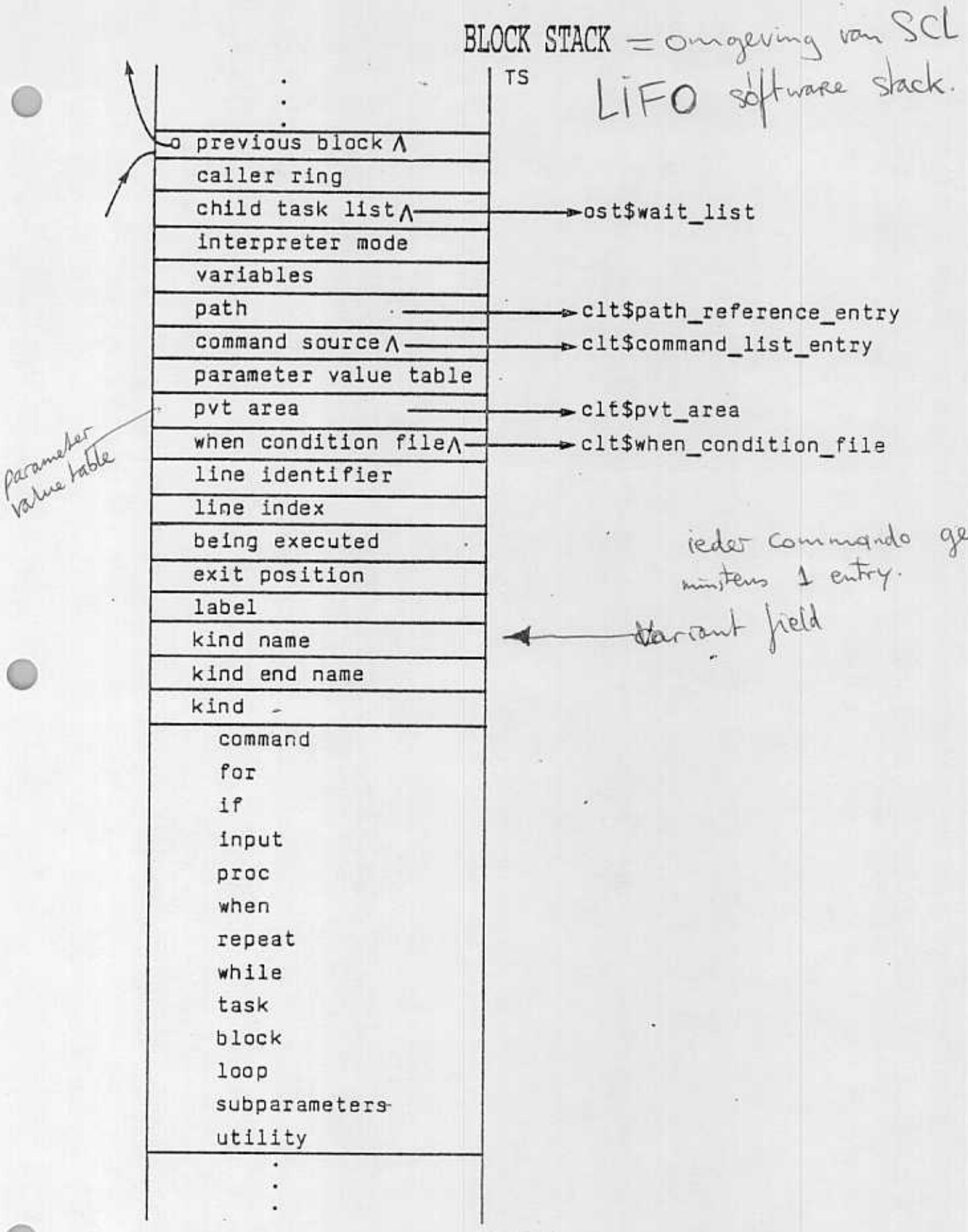
COMMAND PROCESSING

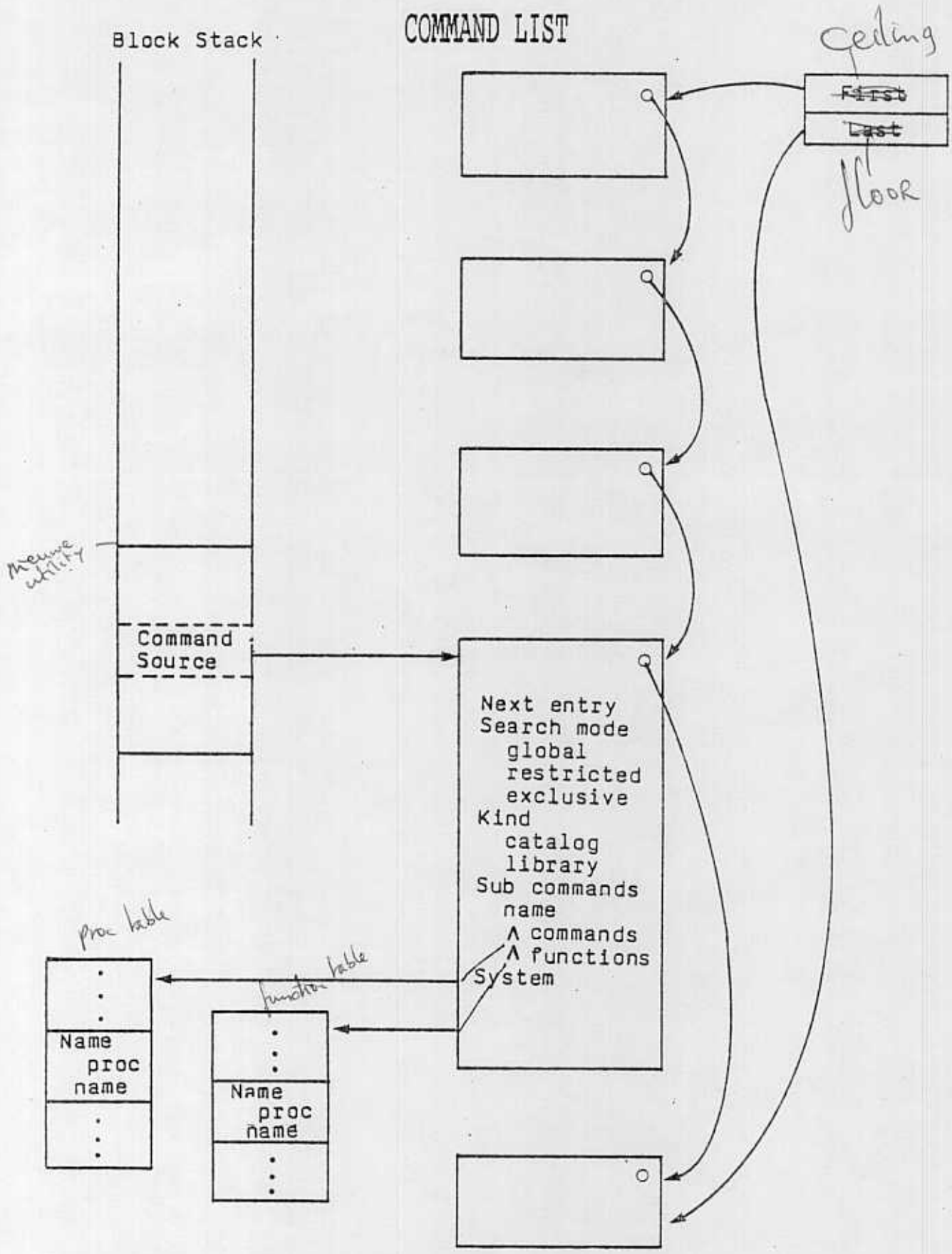


SCL CONTEXT


USER
Lgo
JMM$
INITIAL_
JOB_BEGIN
SCL
INTERPRETER
COMMAND
PROCESSORS
SCL
SERVICES
PROGRAM
INTERFACES
R11
SCL
INTERPRETER
VARIABLE
MANAGER
BLOCK
STACK
COMMAND
LIST
VARIABLES
R3


INTERPRET COMMAND
CLH$INTERPRET_COMMAND
CLP$INTERPRET_COMMAND
"Welcome to NOS/VE . DATE . TIME"
Return (cannot
CALL
CLP$ADD TO_JOB COMMAND_ LIST
CLP$ADD_ TO_JOB COMMAND_ LIST
CLP$ DECLARE VARIABLE OSV$STATUS
Establish Job Files
Put_ Welcome_ Banner
CLP$ SCAN_ COMMAND FILE
"COMMAND"
$SYSTEM
head of list
$LOCAL
new head of list
OSP$ GENERATE_ MESSAGE
RETURN
FILE TYPE
NAME
INTERACTIVE CONNECTION
BATCH CONNECTION
job_command_response
$RESPONSE
Job Output, Log
Log
echo
$ECHO
$NULL
$NULL
error_output
$ERRORS
Job Output
OUTPUT
listing_output
$LIST
$NULL
OUTPUT
standard_output
$OUTPUT
Job Output
OUTPUT
standard_input
$INPUT
Job Input
INPUT
CALL
JMP$ JOB BEGIN
CLP$ INTERPRET_ COMMAND
RETURN
JMM$INITIAL_JOB_BEGIN (2,D,D)
JMP$INITIAL_JOB_BEGIN
INTERPRET COMMAND

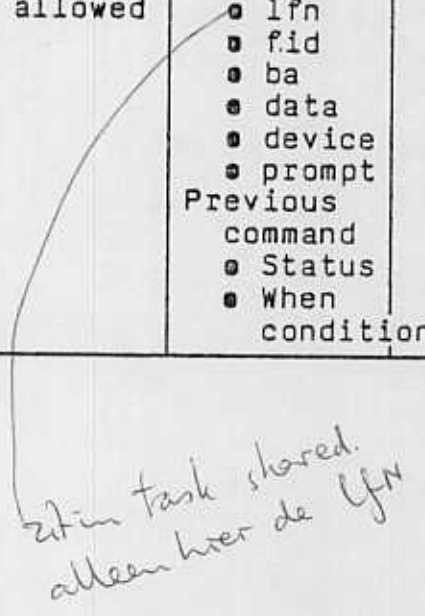

BLOCK STACK = omgeving van SCL
LIFO software stack.
TS
previous block
caller ring
child task list
ost$wait_list
interpreter mode
variables
path
clt$path_reference_entry
command source
clt$command_list_entry
parameter value table
pvt area
clt$pvt_area
when condition file
clt$when_condition_file
parameter value table
line identifier
line index
being executed
exit position
label
kind name
variant field
kind end name
kind
command
for
if
input
proc
when
repeat
while
task
block
loop
subparameters
utility
ieder commando genereert minstens 1 entry.


COMMAND LIST
Block Stack
ceiling
First
Last
floor
Command Source
Next entry
Search mode
global
restricted
exclusive
Kind
catalog
library
Sub commands
name
^ commands
^ functions
System
Proc table
Function table
Name
proc
name
Name
proc
name

BLOCK STACK ENTRY

clt$block_stack_entry

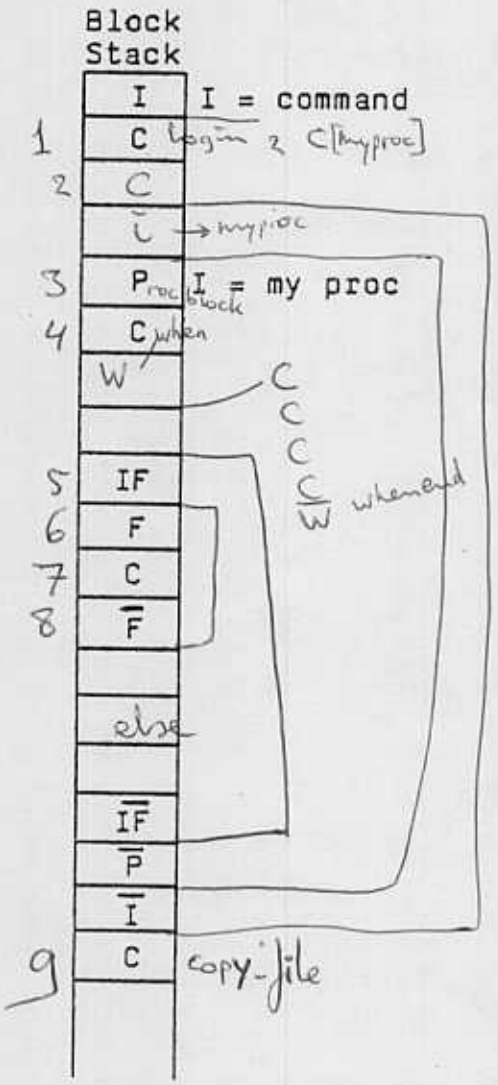
fixed part						
Command	For	IF	Input Proc When	Repeat While	Task	Block Loop Sub Utility
Kind • regular • program • include file • include line • execute	Variable value limit Increment	Condition met Else allowed	Inherited block Input • lfn • fid • ba • data • device • prompt Previous command • Status • When condition	Control exp.	Tid Kind Link Parent Current block Display log index	-



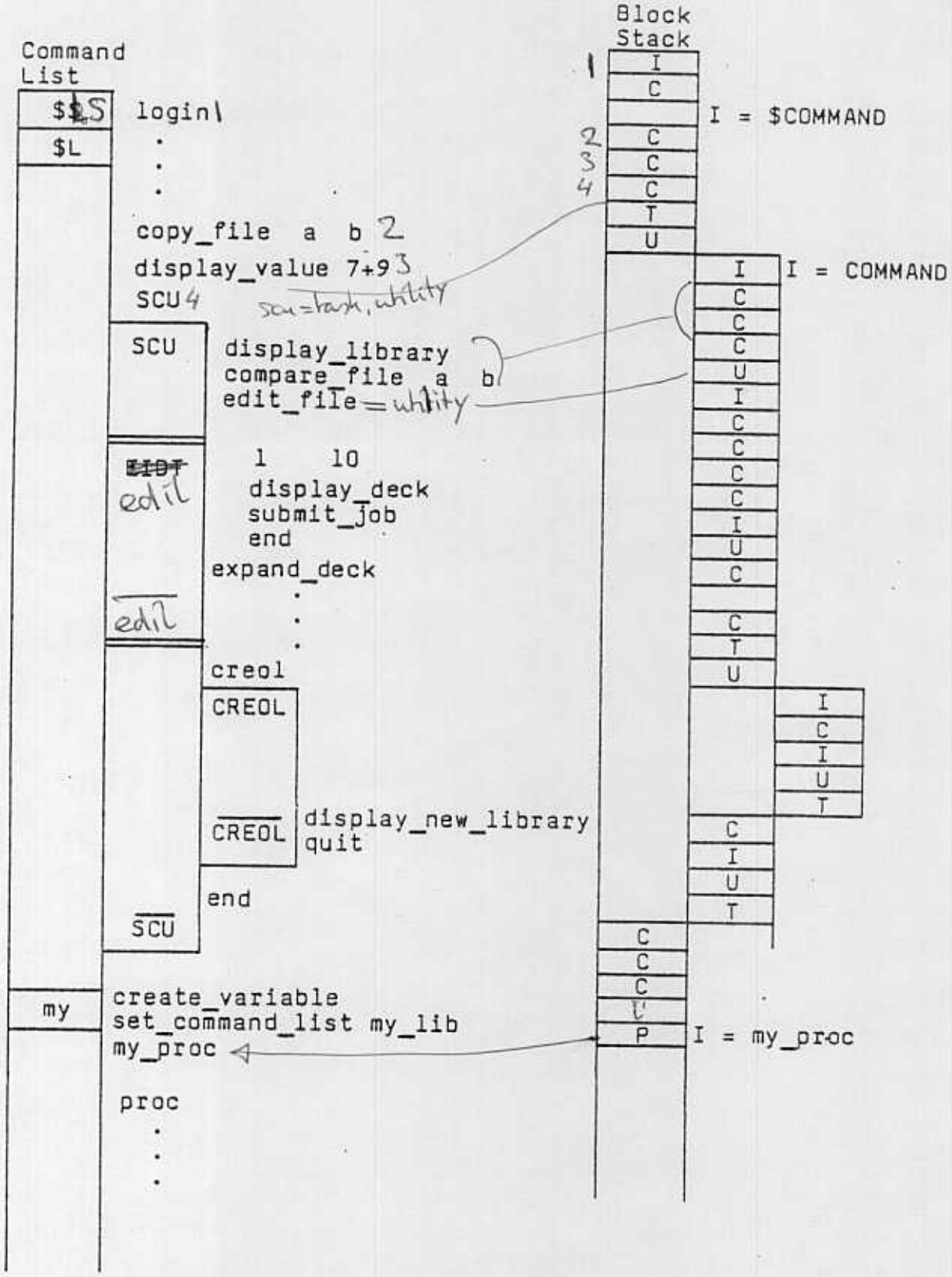
BLOCK STACK EXAMPLE

```
1 login ——— JOB monitor set up
2 my_proc x

  3 PROC name (p1, status)
  4 WHEN condition
      file_name
    WHENEND
  5 IF $value (p1) = specified THEN
    6 FOR i = 1 to $set_count (p1) DO
      7 display_file
    8 FOREND
    ELSE
      display_file all

    IFEND
    PROCEND


9 copy_file  a  b
             .
             .
             .
```

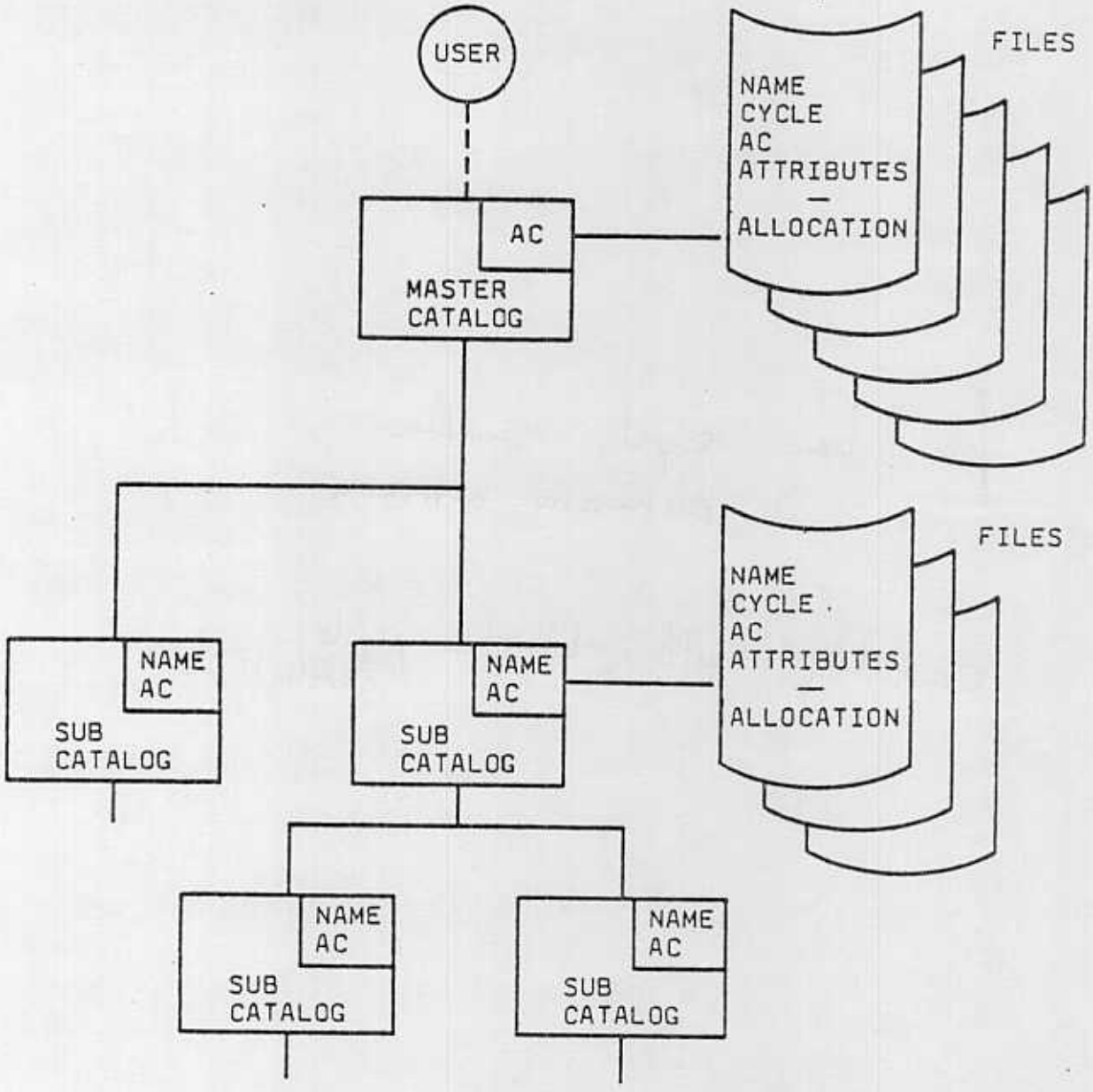
C = command



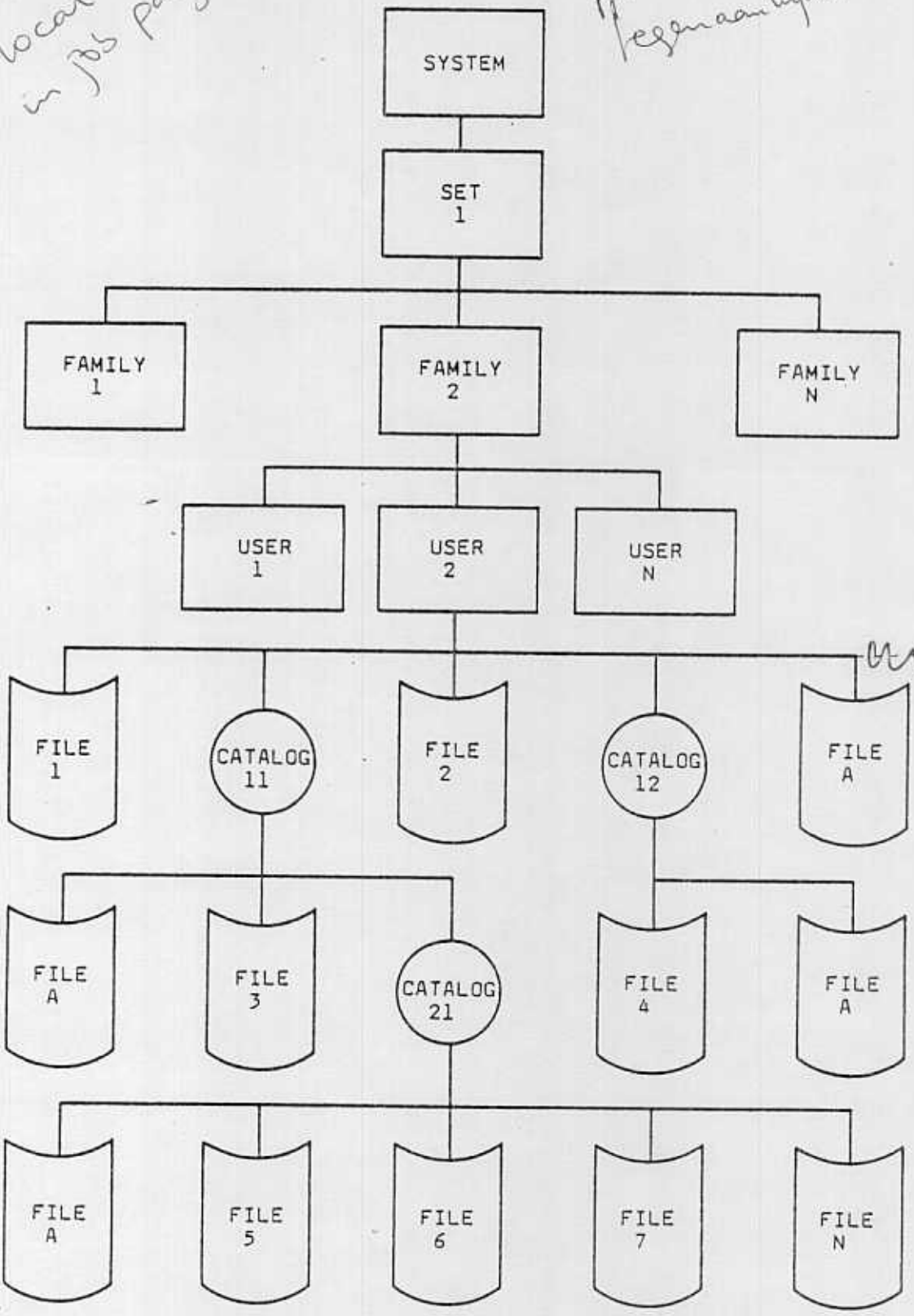
COMMAND LIST/BLOCK STACK EXAMPLE



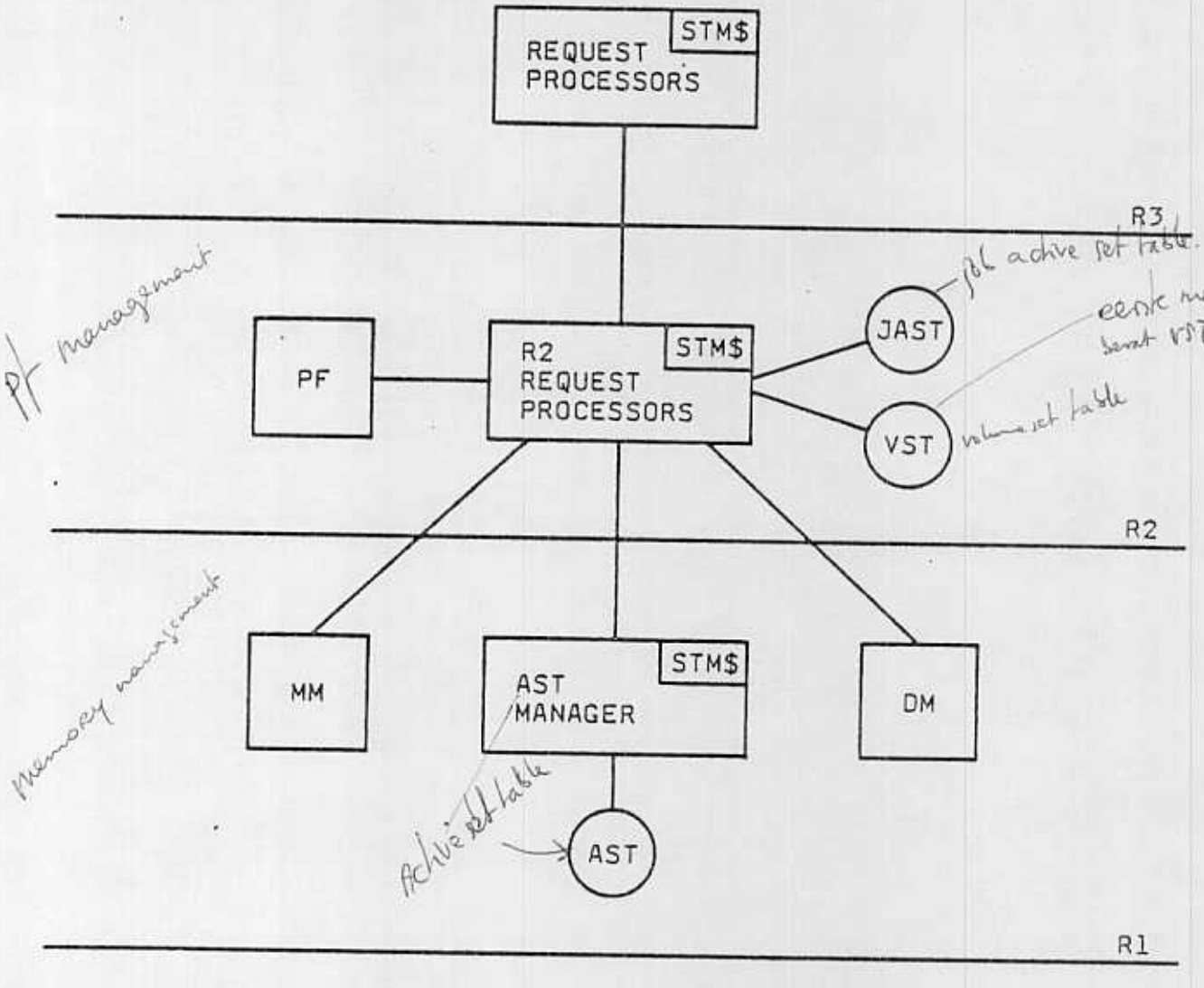
CATALOG/PERM FILE (zoals gebruikt het rest)



CATALOG TREE STRUCTURE



SET MANAGER



File allocation is niet voor set manager.
wel verschaft en kijk van kandidaat volumes voor Device management.

SET MANAGER TABLES

AST — main/prime pageable

stt$active_set_table
STDAST

Name
Master VSN
  Member VSNs
Set Owner
Number of Jobs Using Set
Root object list locator

JAST — job pageable (subtable)

stt$job_active_set_table
STDJAST

Name
Set Owner
Root object list locator

VST

stt$vol_set_table
STDVST

VSN
Name
Member
   Master VSN
Master
   Set Owner
   Root object list locator
     Member VSNs
   VST heap
   Segment size fixer

SET INTERNAL INTERFACES

FROM OUTSIDE SET MGR.

STP$CREATE_SET

STP$ADD_MEMBER_VOL_TO_SET

STP$PURGE_SET

STP$REMOVE_MEMBER_VOL_FROM_SET

STP$ASSOCIATE_CATALOG

FROM WITHIN SET MGR.

STP$CREATE_VOL_SET_TABLE

STP$GET_ROOT_OBJECT_LOCATOR

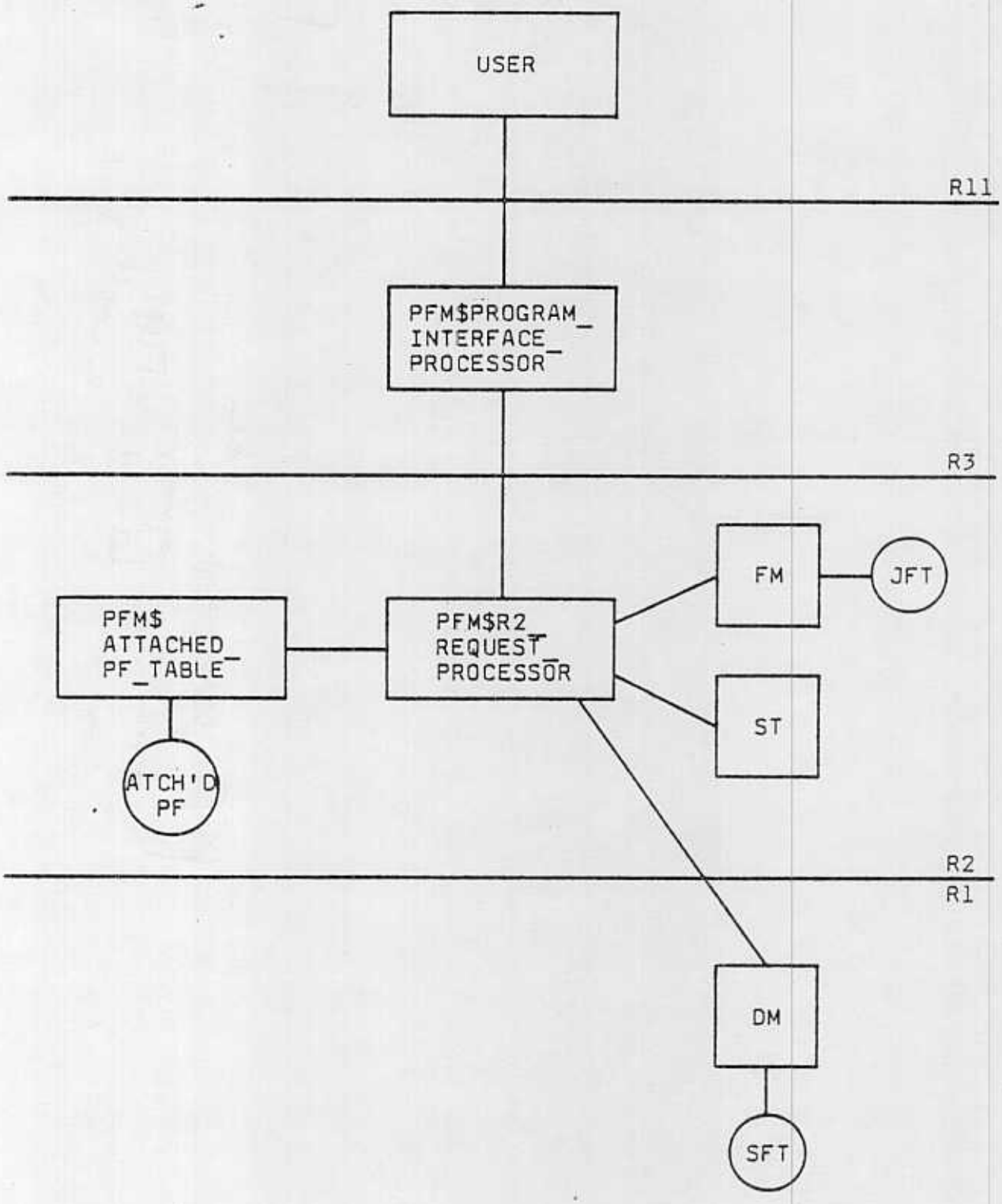
STP$GET_SET_OWNER

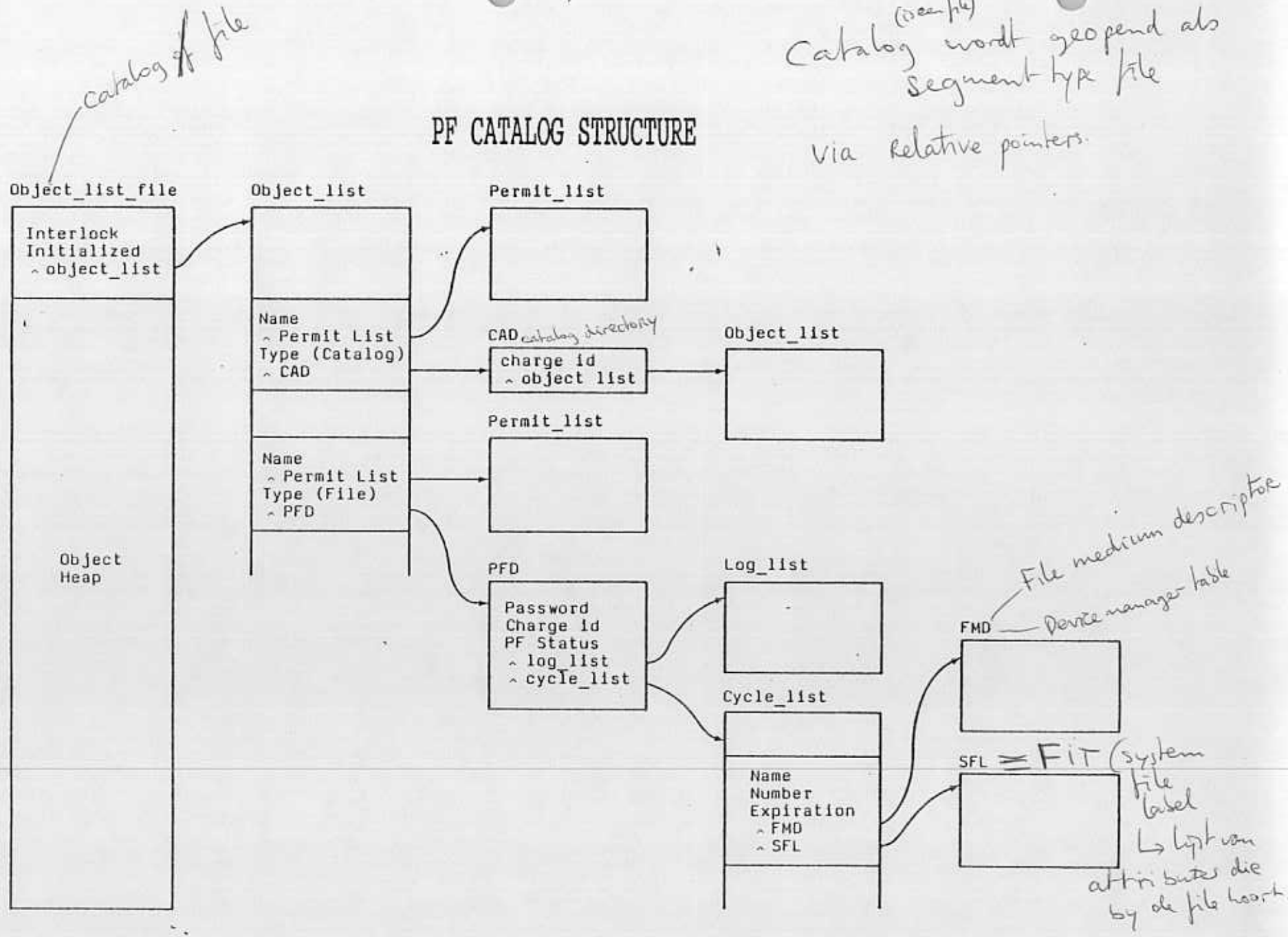
STP$CHECK_CATALOG_ASSOCIATION

STP$CHANGE_ACCESS_TO_SET

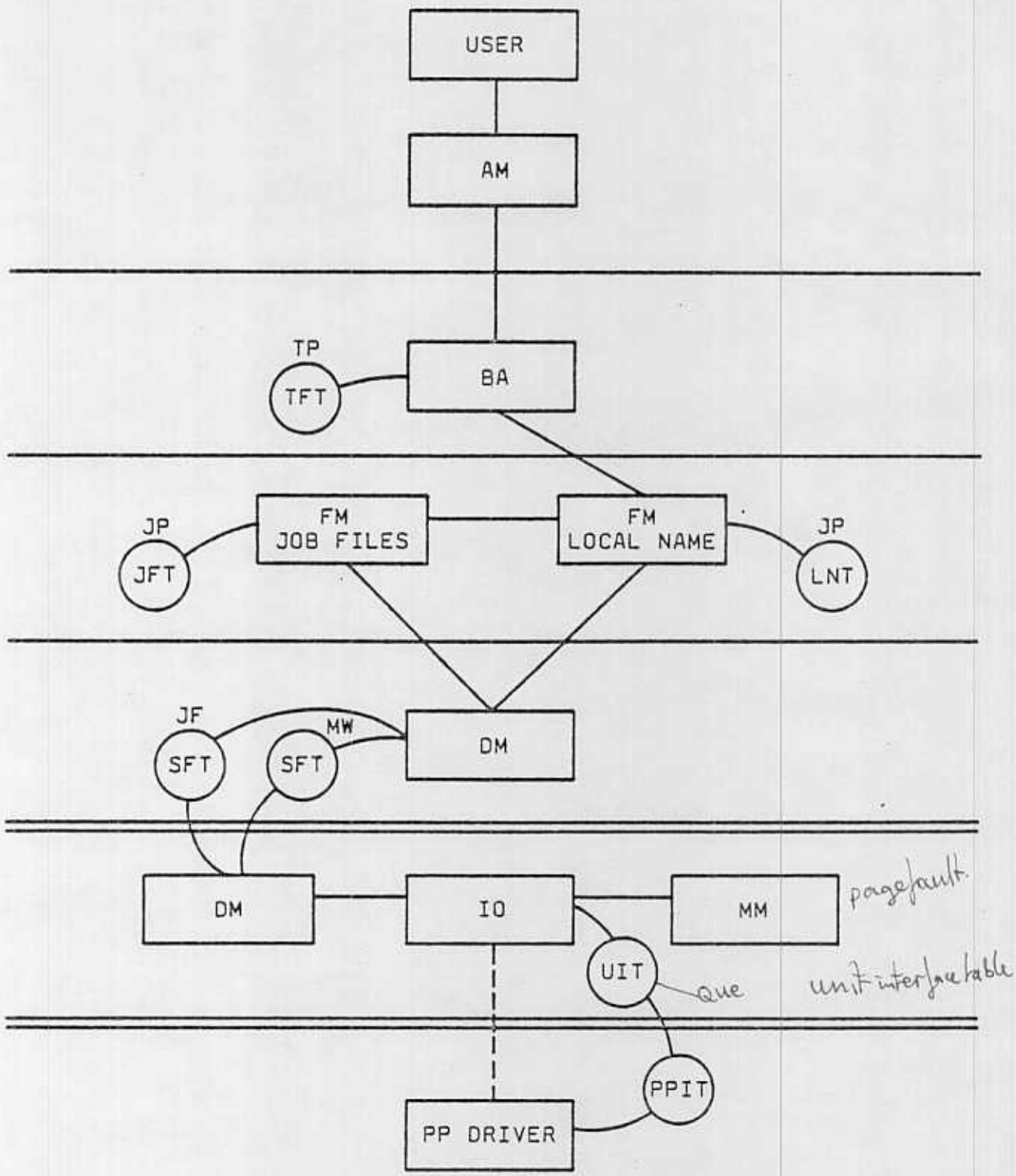
STP$SET_END_JOB

PF MANAGER

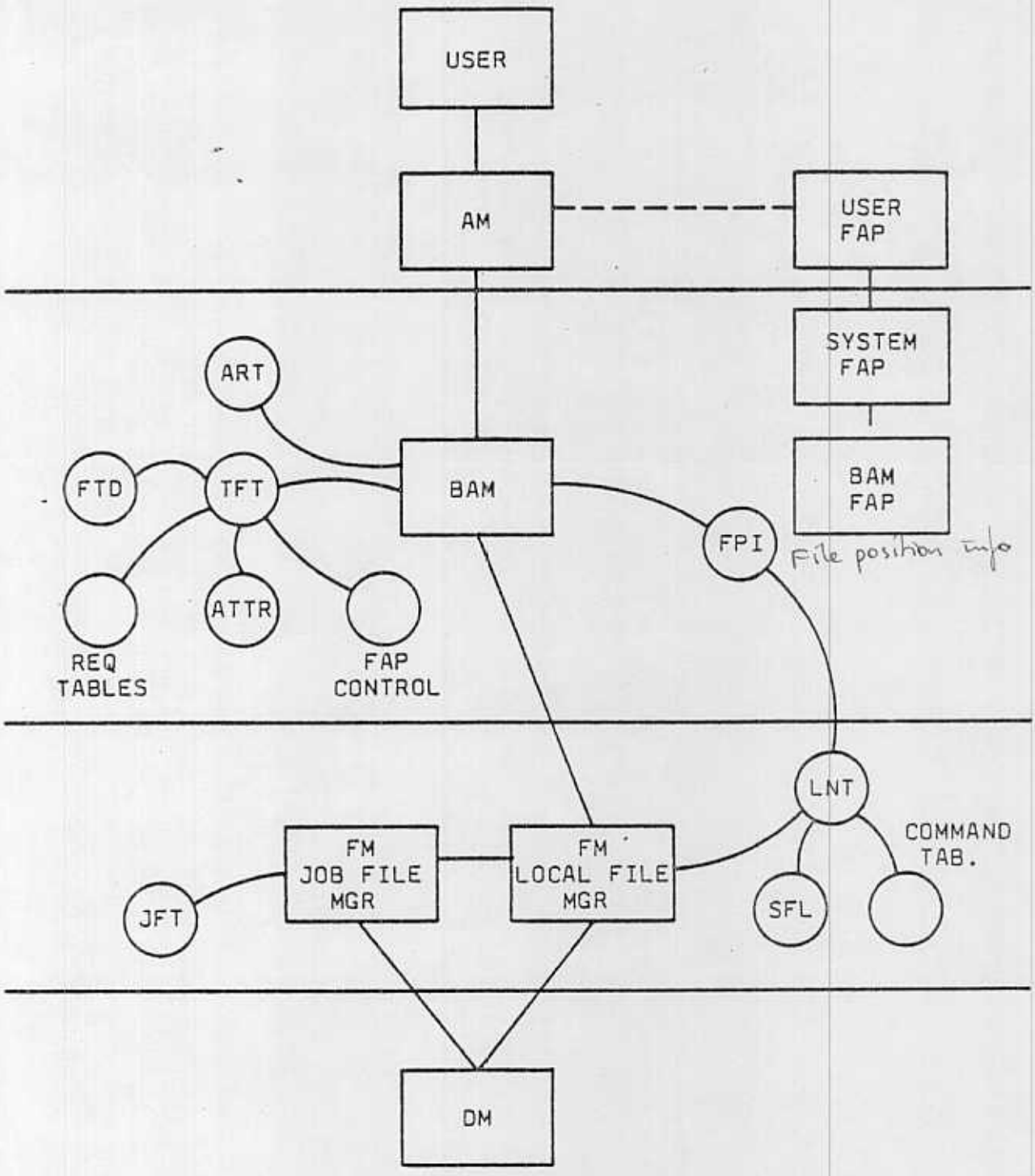

USER
R11
PFM$PROGRAM_
INTERFACE_
PROCESSOR
R3
FM
JFT
PFM$
ATTACHED_
PF_TABLE
PFM$R2_
REQUEST_
PROCESSOR
ST
ATCH'D
PF
R2
R1
DM
SFT


Catalog of file
PF CATALOG STRUCTURE
Catalog (open file) wordt geopend als Segment type file
via Relative pointers.
Object_list_file
Interlock
Initialized
^ object_list
Object
Heap
Object_list
Name
^ Permit List
Type (Catalog)
^ CAD
Name
^ Permit List
Type (File)
^ PFD
Permit_list
CAD catalog directory
charge id
^ object list
Object_list
Permit_list
PFD
Password
Charge id
PF Status
^ log_list
^ cycle_list
Log_list
Cycle_list
Name
Number
Expiration
^ FMD
^ SFL
FMD
File medium descriptor
Device manager table
SFL ≡ FIT (system file label
↳ bijt van attributes die by de file hoort.

FILE MANAGEMENT COMPONENTS



BASIC ACCESS METHOD



FILE ACCESS PROCEDURES

1. USER

   Interstate Communication
   User Defined

2. CONNECTED FILE

3. SYSTEM

   Advanced Access Method
   ~~Connected File~~
   Operator Facility
   Interactive Facility
   Interstate Communication
   Logging

4. BASIC ACCESS METHOD = laatste FAP

ATTRIBUTES

* Permanent attributes are established on the first open of a new file.

* Permanent attributes are never changed (R1). release 1

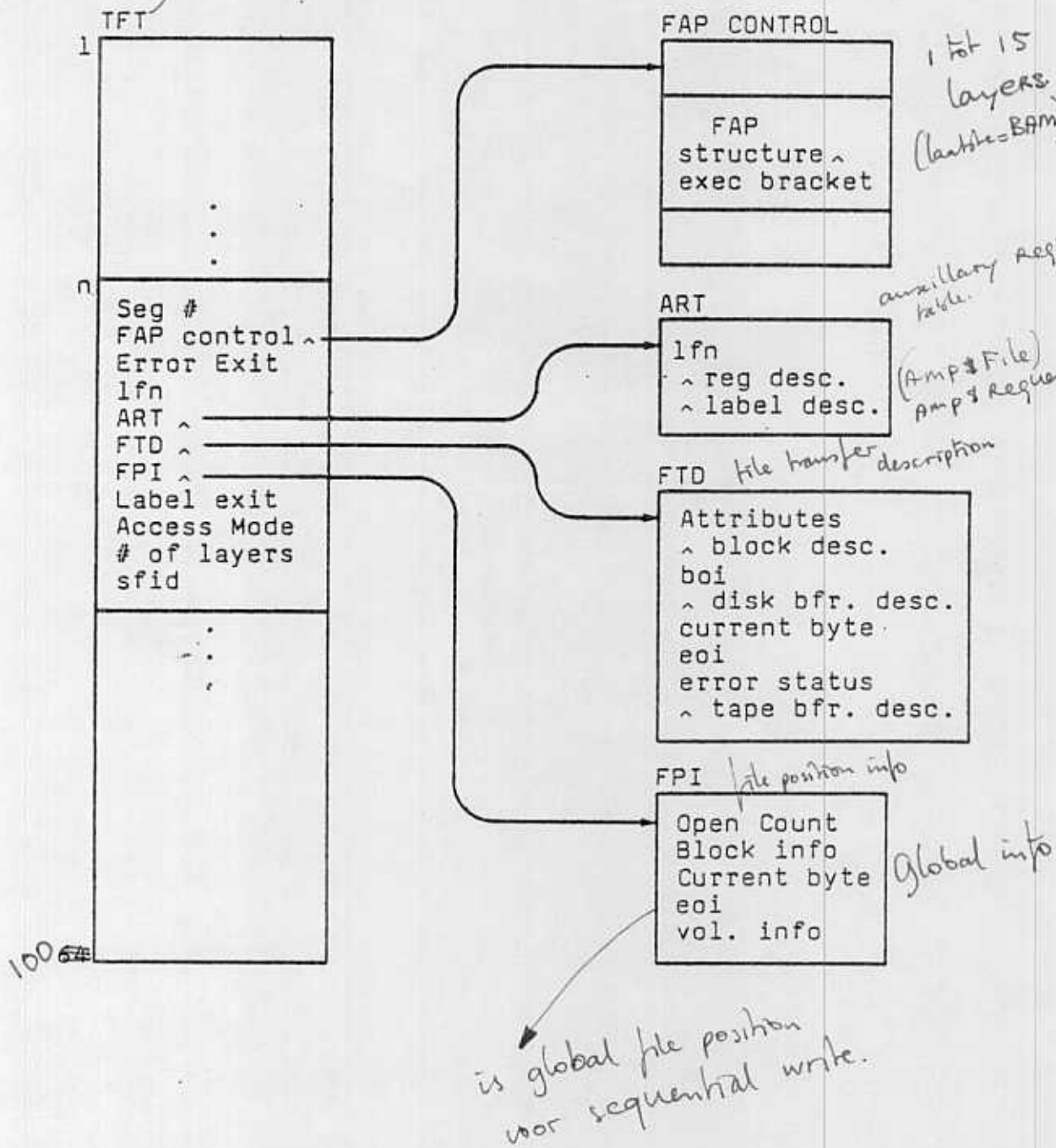
* Source of permanent attributes:

  FAB Request
  Open Request
  Commands
  Other program interface requests
  Defaults

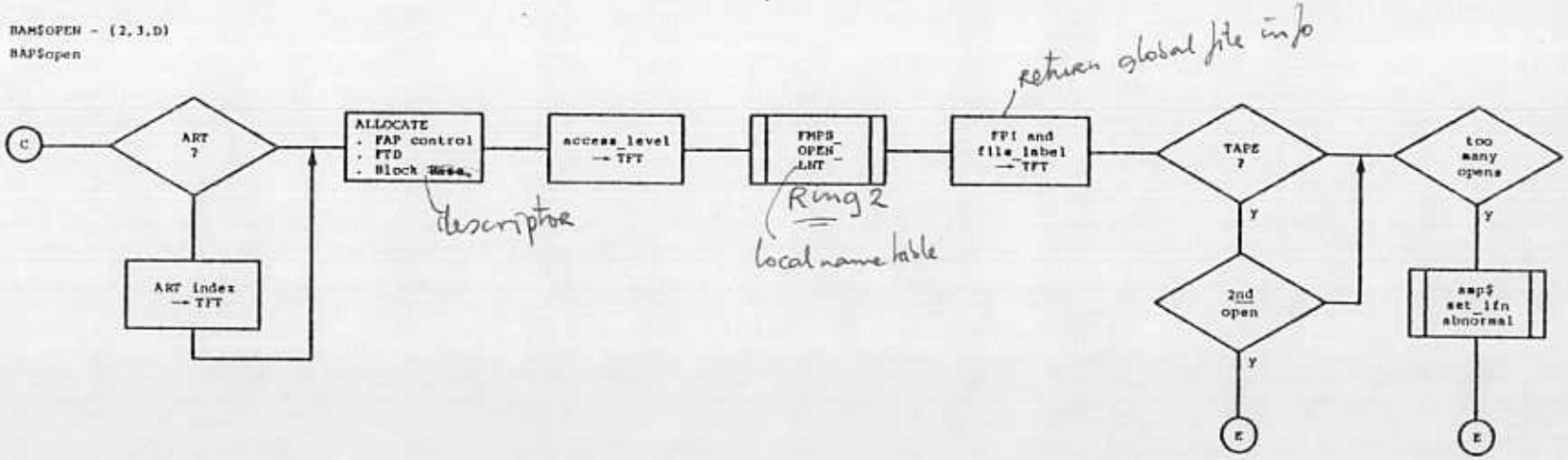
* Source of temporary attributes:

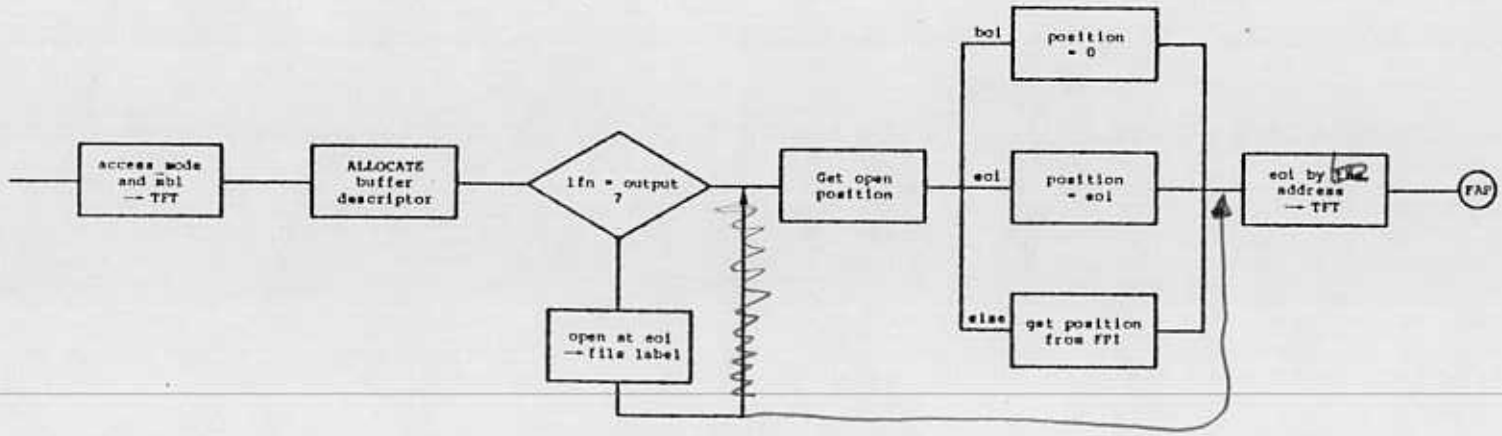
  Store request
  Open
  Commands
  Other program interface requests
  SFL
  Defaults

TASK FILE TABLES

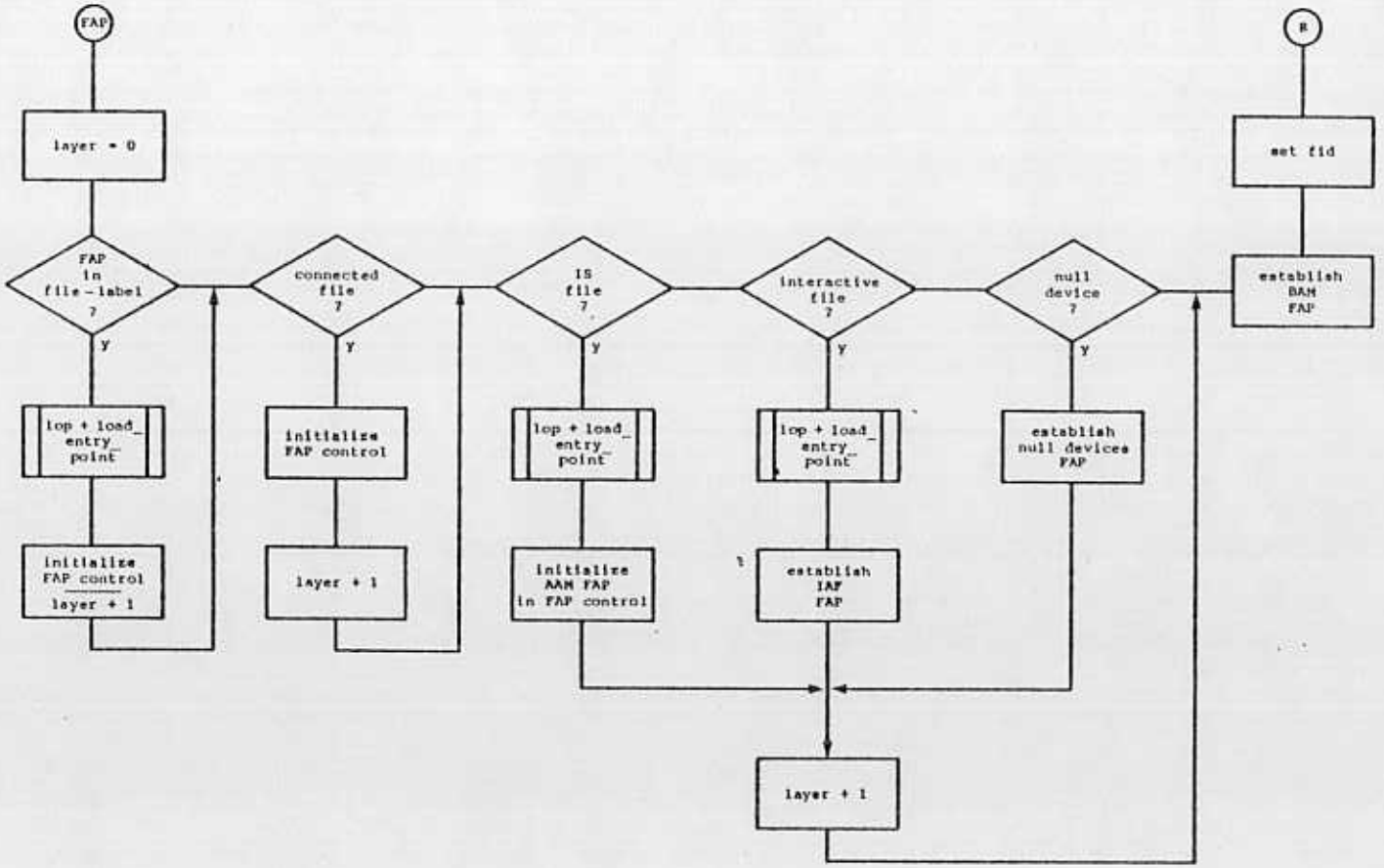


BAM$OPEN - (2,3,D)
BAP$open





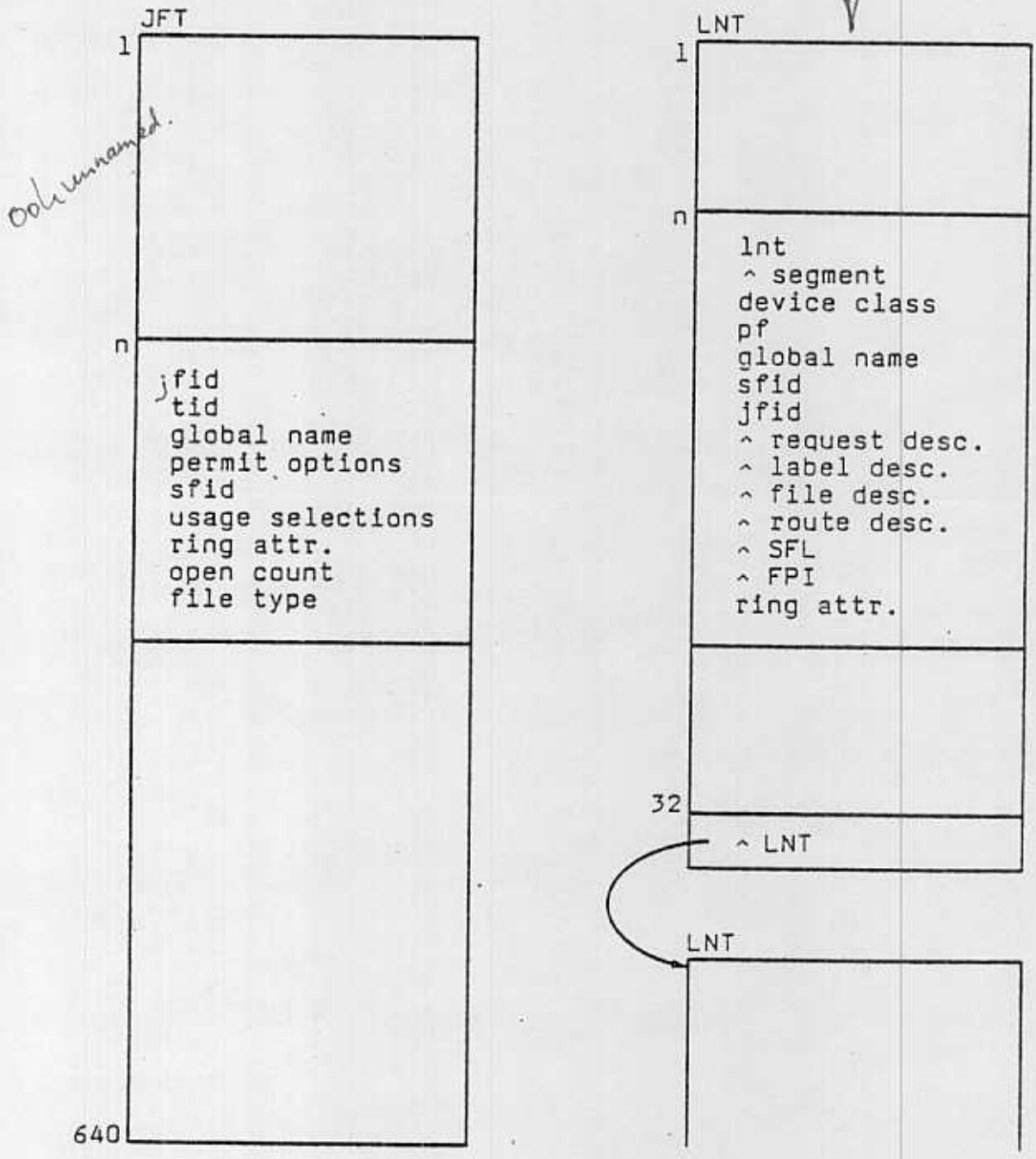
BAP$OPEN continued



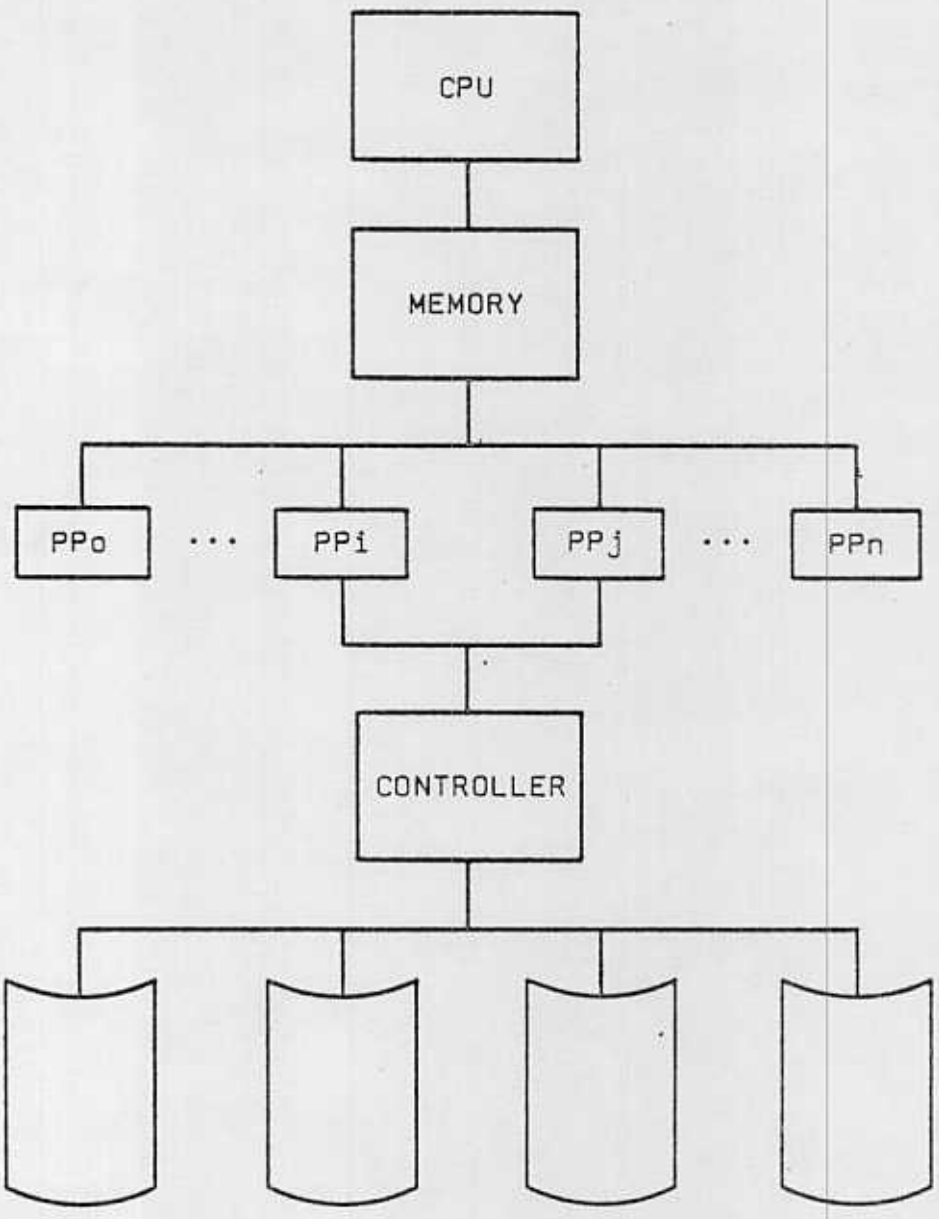
indexed by job file id.

FILE MANAGER TABLES

indexed by file name.



MASS STORAGE DEVICES



844-4x
885-1x
~~885-4x~~

DEFINITIONS

MAU--The minimum addressable unit is the quantum of data transfer between a driver and a mass storage device. It is a constant 2048 bytes in length. Standard software is released with page size ≥ 2048 bytes (MAU). Special systems could have page size < 2048 bytes but page size could never be changed without file conversion.

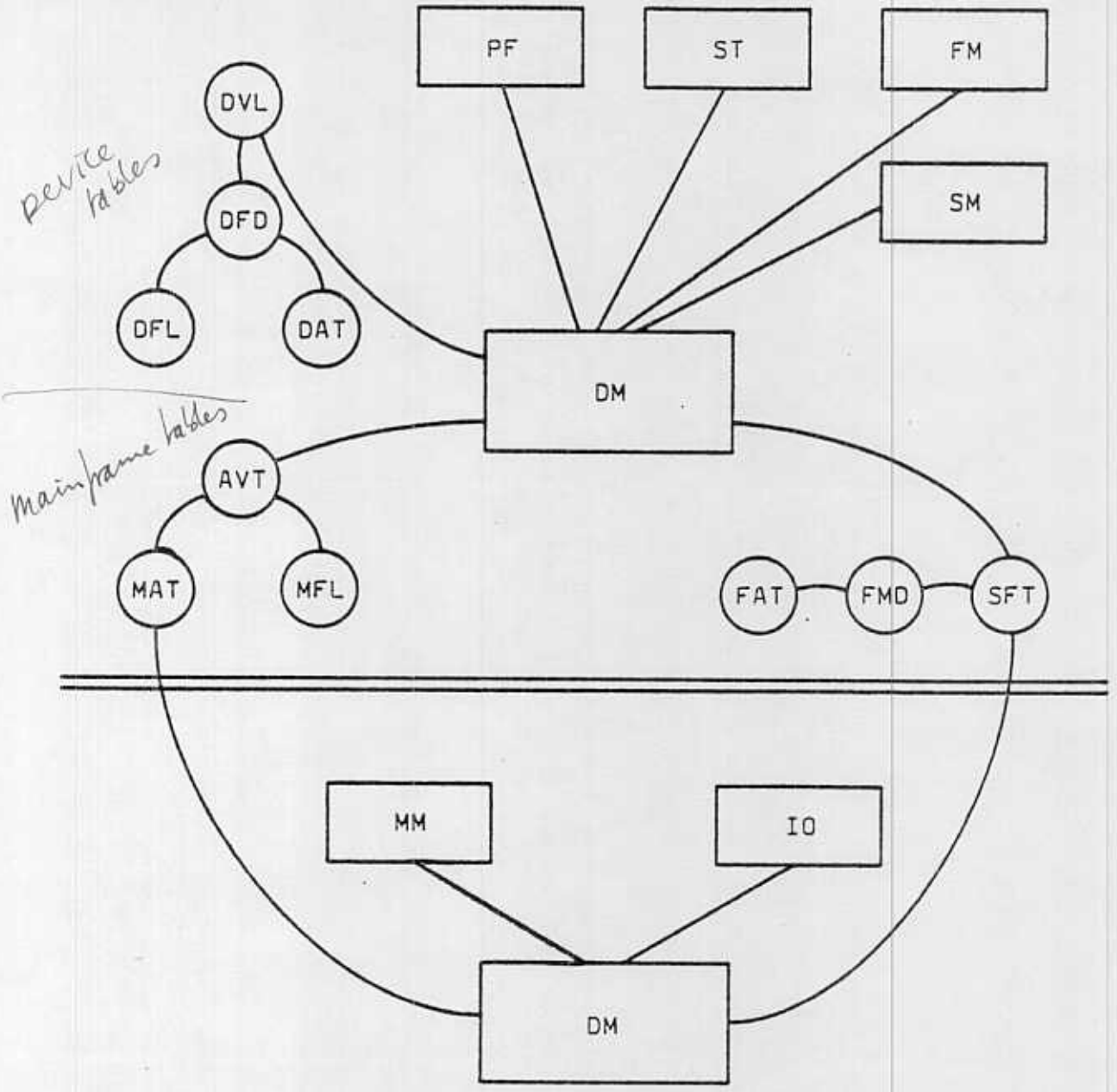
DAU--The device allocation unit is the quantum of device allocation. It is a device dependent, integral multiple of MAU.

nu = DAU = 2 MAU

ALLOCATION UNIT—A power of 2 multiples of contiguous DAUs on a device. An allocation unit does not span cylinders on a device. A physical I/O request does not span allocation units. Expressed as A1, A2, A4, A8, A16, A32, A64, A128, A256. Current default = Au (16K) = 2 pages

	844-4x	885-1x	885-4x
Capacity			
Cylinders/Spindle	823	843	843
Tracks/Cylinder	19	40	10
MAU/DAU (bytes)	(4096) 2	(4096) 2	(4096) 2
Total (*10^6 bytes)	151.6	552.5	552.5
Performance			
Seconds/Revolution	.0167	.0167	.0167
Transfer rate (bytes/sec)	.589x10^6	.981x10^6	3.924x10^6
Allocation			
DAU/A1 (Bytes)	(4096) 1	(4096) 1	(4096) 1
DAU/A2 (Bytes)	(8192) 2	(8192) 2	(8192) 2
.	.	.	.
.	.	.	.
.	.	.	.
DAU/A32	32	32	32
DAU/A64	(180224) 44	64	64
DAU/A128	44	128	128
DAU/A256	44	(655360)160	(655360)160

DEVICE MANAGER CONTEXT



DM TABLES

SFT	SYSTEM FILE TABLE	1 entry/file
FMD	FILE MEDIUM DESCRIPTOR	1 entry/subfile
FAT	FILE ALLOCATION TABLE	1 entry/allocation
DVL	DEVICE LABLE	1/volume
DFD	DEVICE FILE DIRECTORY	1 entry/device file
DFL	DEVICE FILE LIST	1 entry/subfile
DAT	DEVICE ALLOCATION TABLE	1 entry/AU
AVT	ACTIVE VOLUME TABLE	1 entry/volume
MFL	MAINFRAME FILE LIST	1 entry/new file
MAT	MAINFRAME ALLOCATION TABLE	1 entry/available AU

DEVICE MANAGER USERS

- FILE MANAGER
  - Locally Named File Mgr.
    - File Allocation
    - Set Mass Storage Limit
  - Job File Mgr. (temporary files only)
    - Create File
    - Assign File to Device
    - Destroy File

- MEMORY MANAGER
  - Store ASID in SFT for Sharing
  - Provide transfer unit offset and length

- PHYSICAL IO
  - Device Address for IO transfers
  - Check initial write of new allocation
  - Flaws

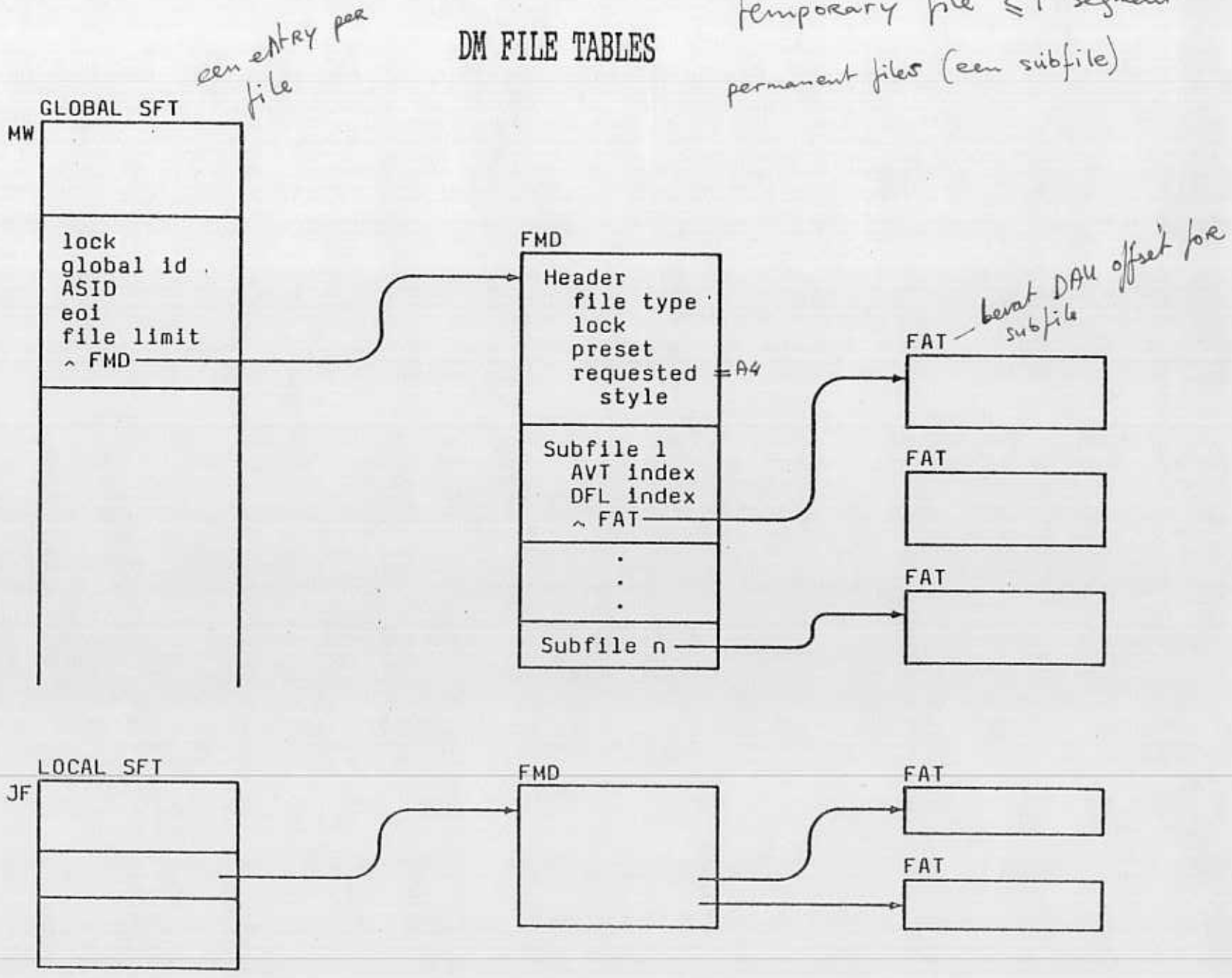
- MANAGE SETS
  - Add volume to Set
  - Remove volume from Set

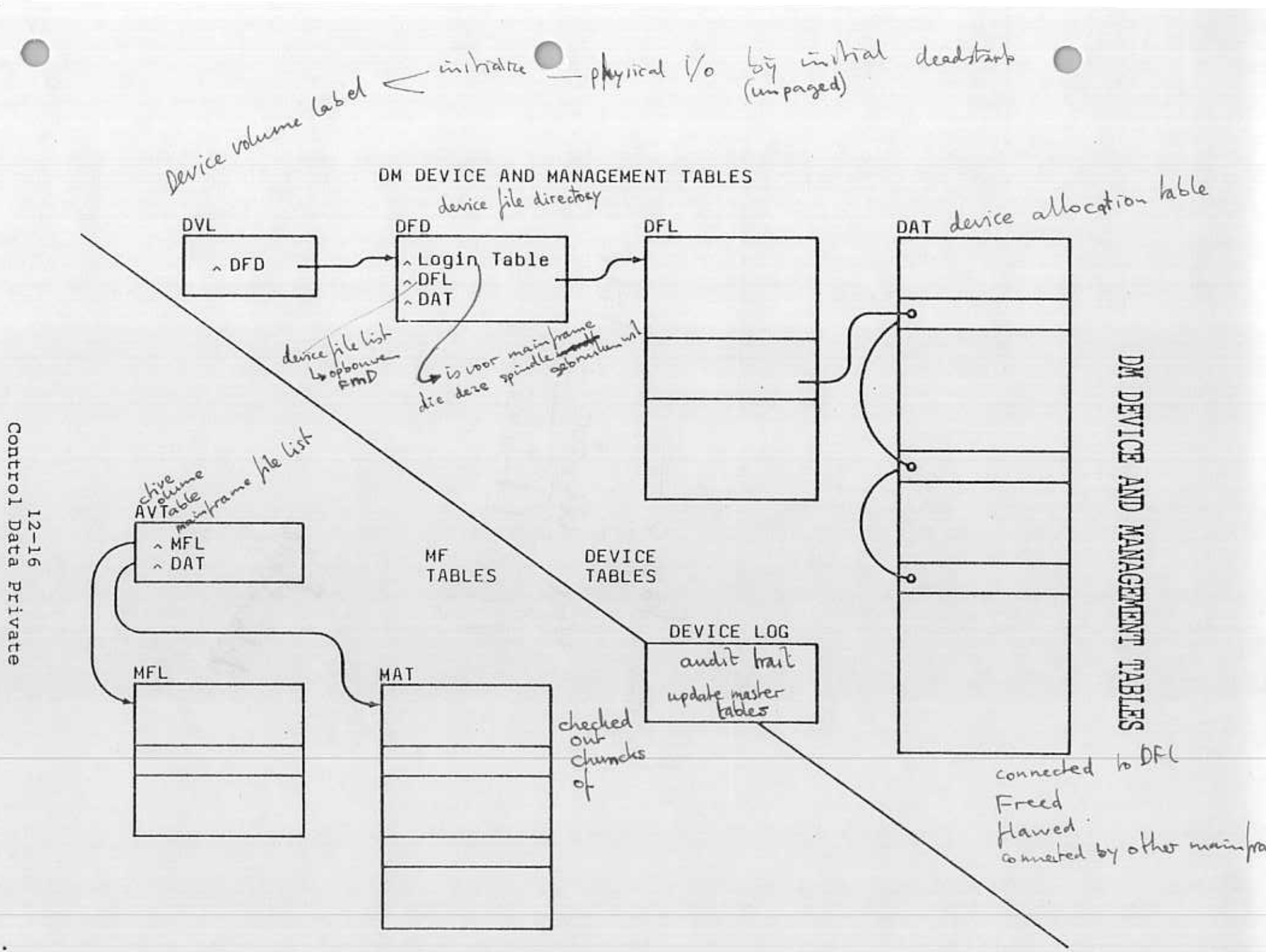
- MANAGE PFs
  - Get FMD for storage in PF Catalog
  - Manage FMD on attach/detach
  - Destroy PF
  - Lock and Unlock Catalog File

DM FILE TABLES

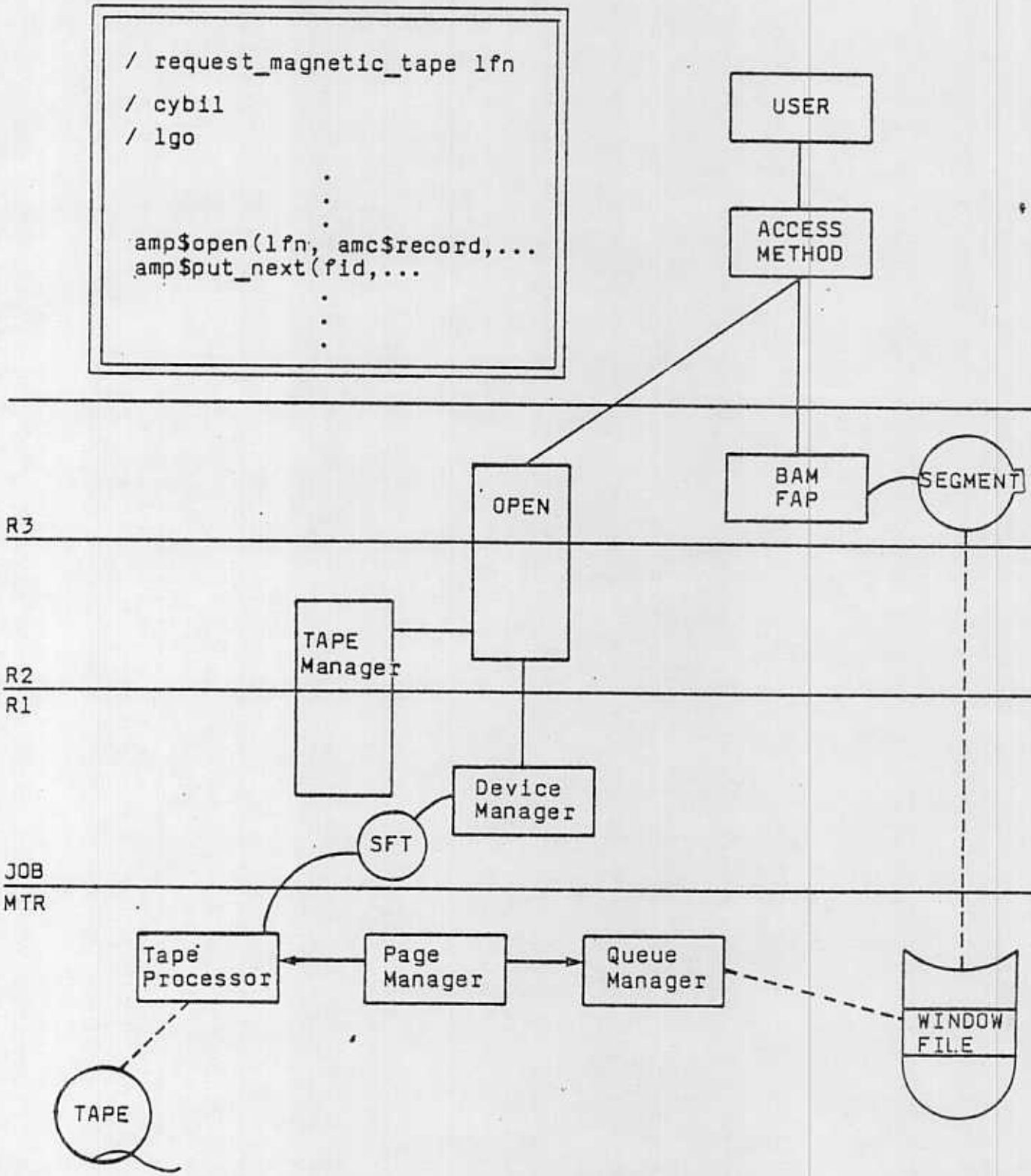
temporary file ≤ 1 segment

permanent files (een subfile)




Device volume label
initialize
physical I/O by initial deadstart (unpaged)
DM DEVICE AND MANAGEMENT TABLES
device file directory
DVL
^ DFD
DFD
^ Login Table
^ DFL
^ DAT
DFL
DAT device allocation table
device file list
is voor mainframe die deze spindle
AVTable
active volume
mainframe file list
^ MFL
^ DAT
MF TABLES
DEVICE TABLES
MFL
MAT
checked out chunks of
DEVICE LOG
audit trail
update master tables
connected to DFL
Freed
Flawed
connected by other mainfra
DM DEVICE AND MANAGEMENT TABLES

MAGNETIC TAPE

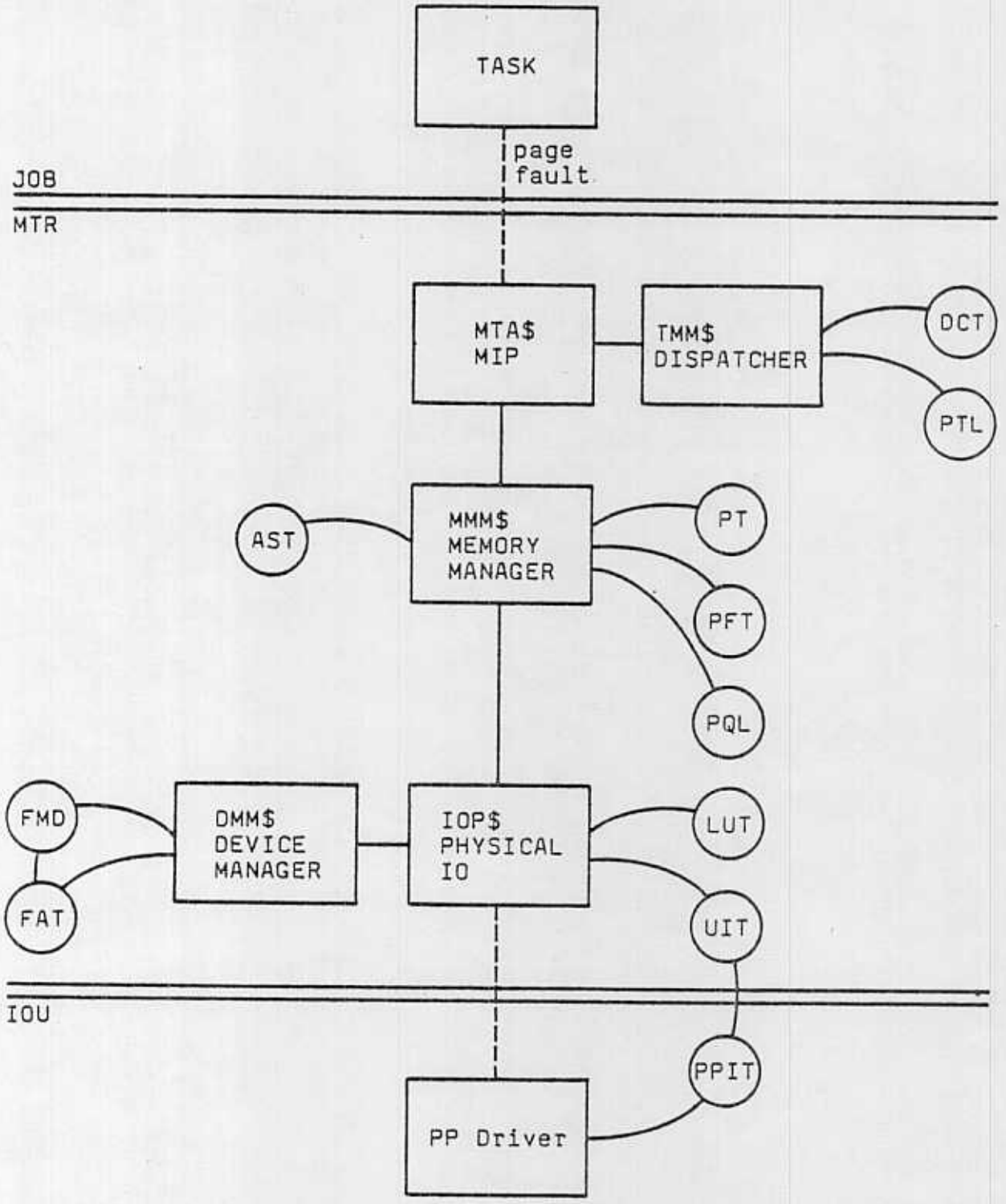


The user opens the file in the usual way and makes get_next and put_next requests in the usual way. The system provides device independence. The only restrictions are that the file can be opened only once and that it must be a sequential file opened for record level access. Labels are not supported on Release 1.

BAM opens the file for segment level access. On the first access, a page fault occurs. The system assigns real memory pages for 10 blocks and allocates a window file on disk to hold 10 blocks. Ten blocks are transferred by the PP driver to real memory. When the transfer is complete, the program continues.

Often the "window" stays in real memory until the program exhausts it. If the pages must be paged out, they are paged to the window file on the disk. Page faults occur when the page resides on disk. The window file never changes size. Device manager advances the segment offset so that the file looks like a normal sequential file.

MEMORY MANAGER CONTEXT



TABLES

PTL	Primary Task List-TM	1 entry/task
DCT	Dispatch Control Table-TM	1/mainframe
PT	Page Table-SY Hardware	1 entry/active page
PFT	Page Frame Table Software	1 entry/page
PQL	Page Queue List PFT tops of threads	1/mainframe
AST	Active Segment Table AST index → ASID	1/active segment
FMD	File Medium Descriptor	1/file
FAT	File Allocation Table	1/subfile
LUT	Logical Unit Table	1/drive
UIT	Unit ~~Information~~ replace Table IO request queue	1/drive
PPIT	PP Interface Table	1/drive

MODULES

MONITOR INTERRUPT HANDLER

- o Receive Page Fault
- o Call Memory Manager to process fault
- o Call Physical IO Mgr to process completion

DISPATCHER

- o Adjust wait status
- o Pick next task to execute

MEMORY MANAGER

- o Process Page Fault
- o Manage Working Set (aging)
- o Lock/Unlock pages — PP pages master selected sys.

PHYSICAL IO

- o Link requests
- o Alert PP
- o Process IO completion status

DEVICE MANAGER

- o Provide physical addresses
- o Allocate space

PP DRIVER

- o Function and status the device
- o Read/Write the device
- o Read/Write Real Memory

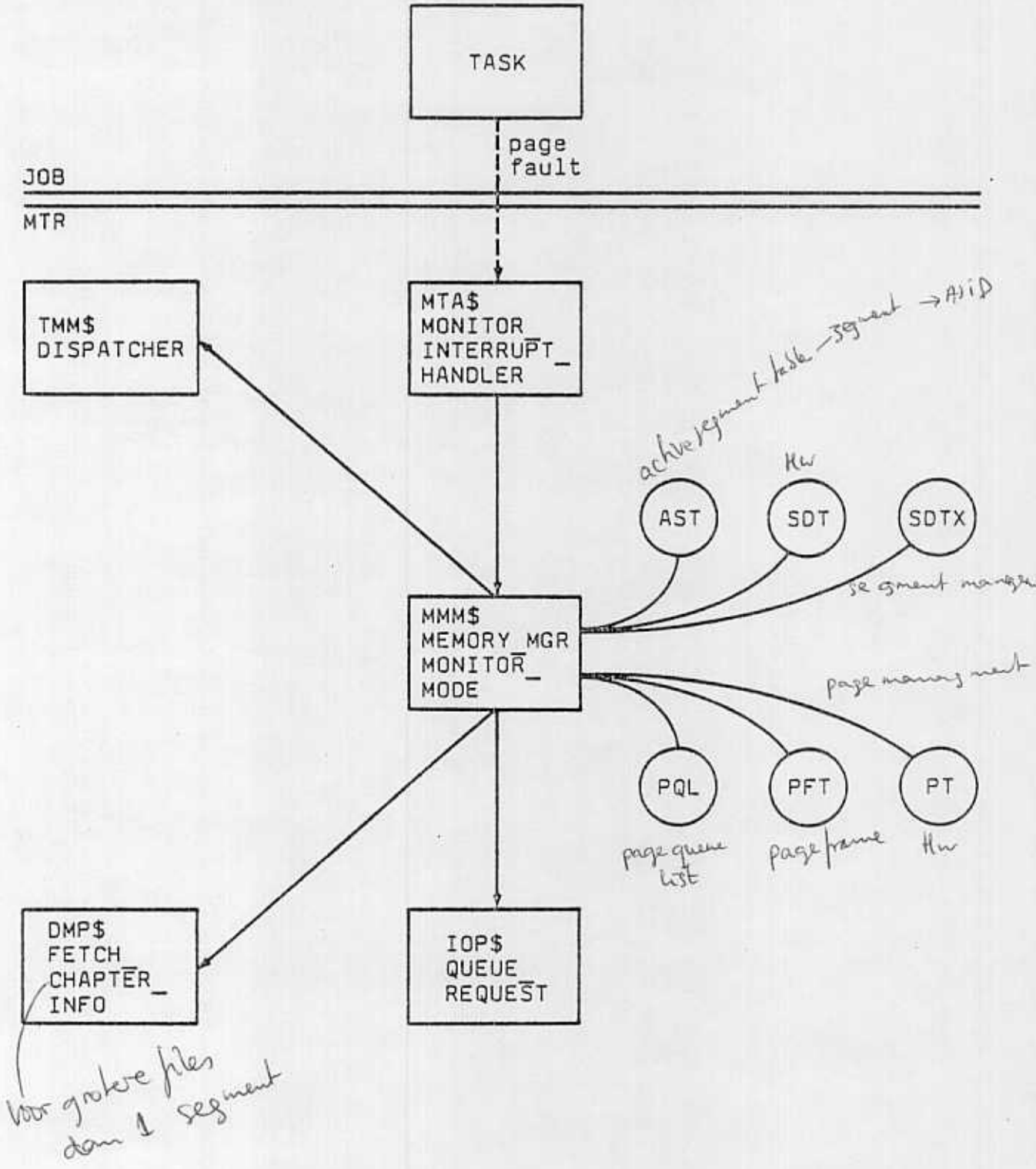
PHYSICAL IO

1. PROCESS PAGE FAULT

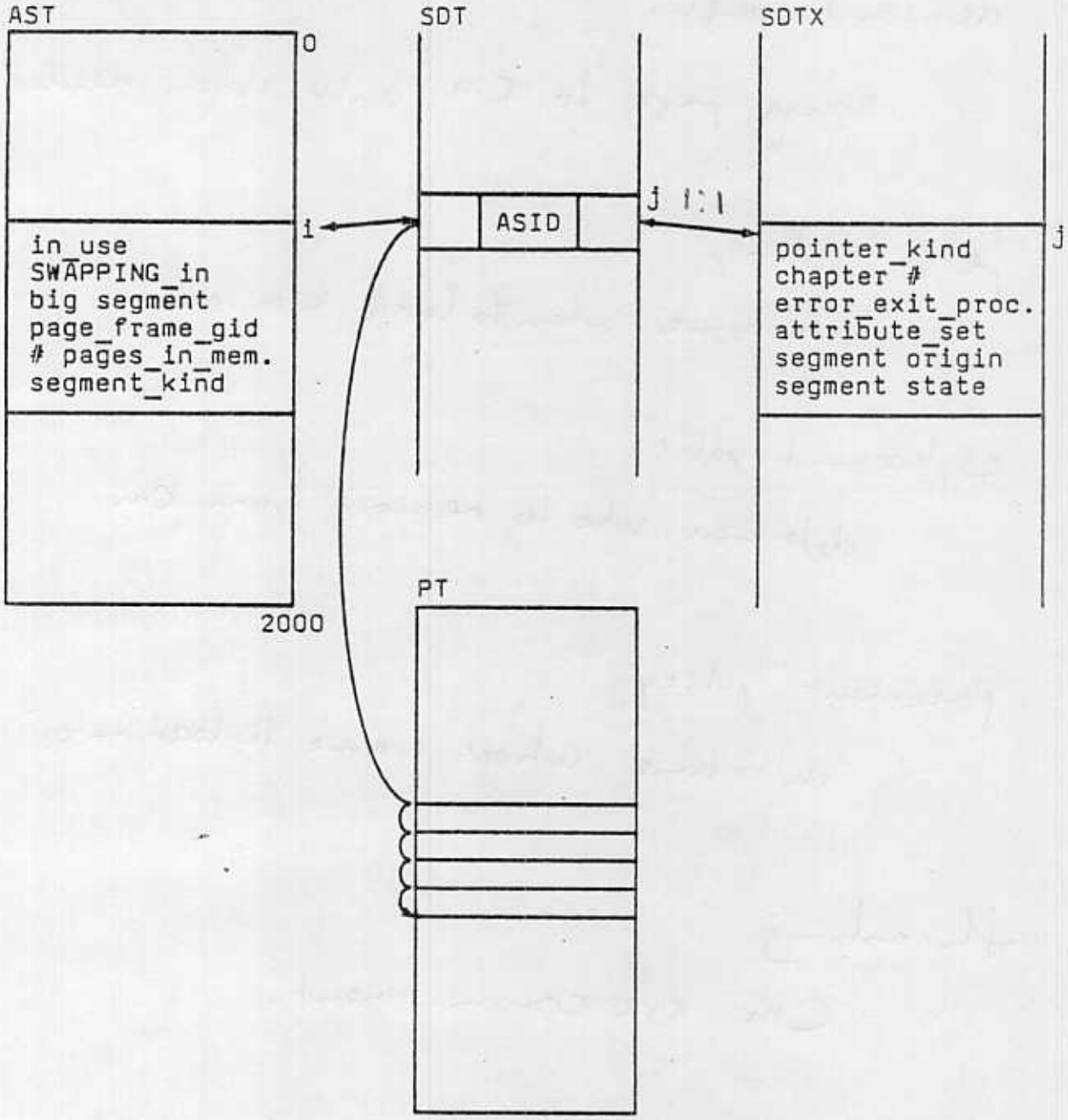
2. INITIATE PHYSICAL IO

3. PROCESS IO COMPLETION

PROCESS PAGE FAULT



SEGMENT TABLES

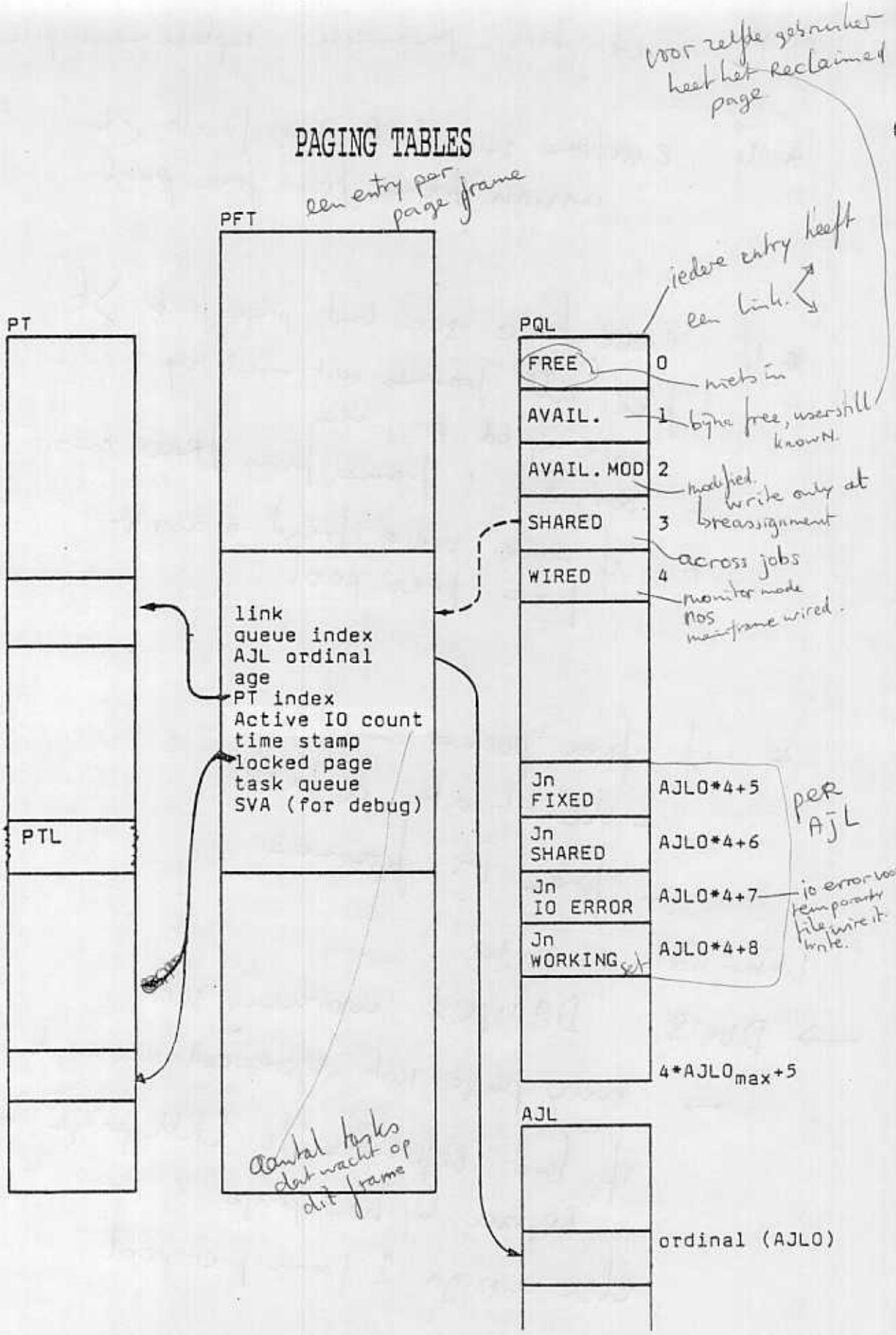


Software hashes the AST index to assign the ASID.

Hardware hashes the ASID and page offset to find the page table index. A sequential search of the next 32 entries might follow.

page table full
↳ 32 entries full.

PAGING TABLES

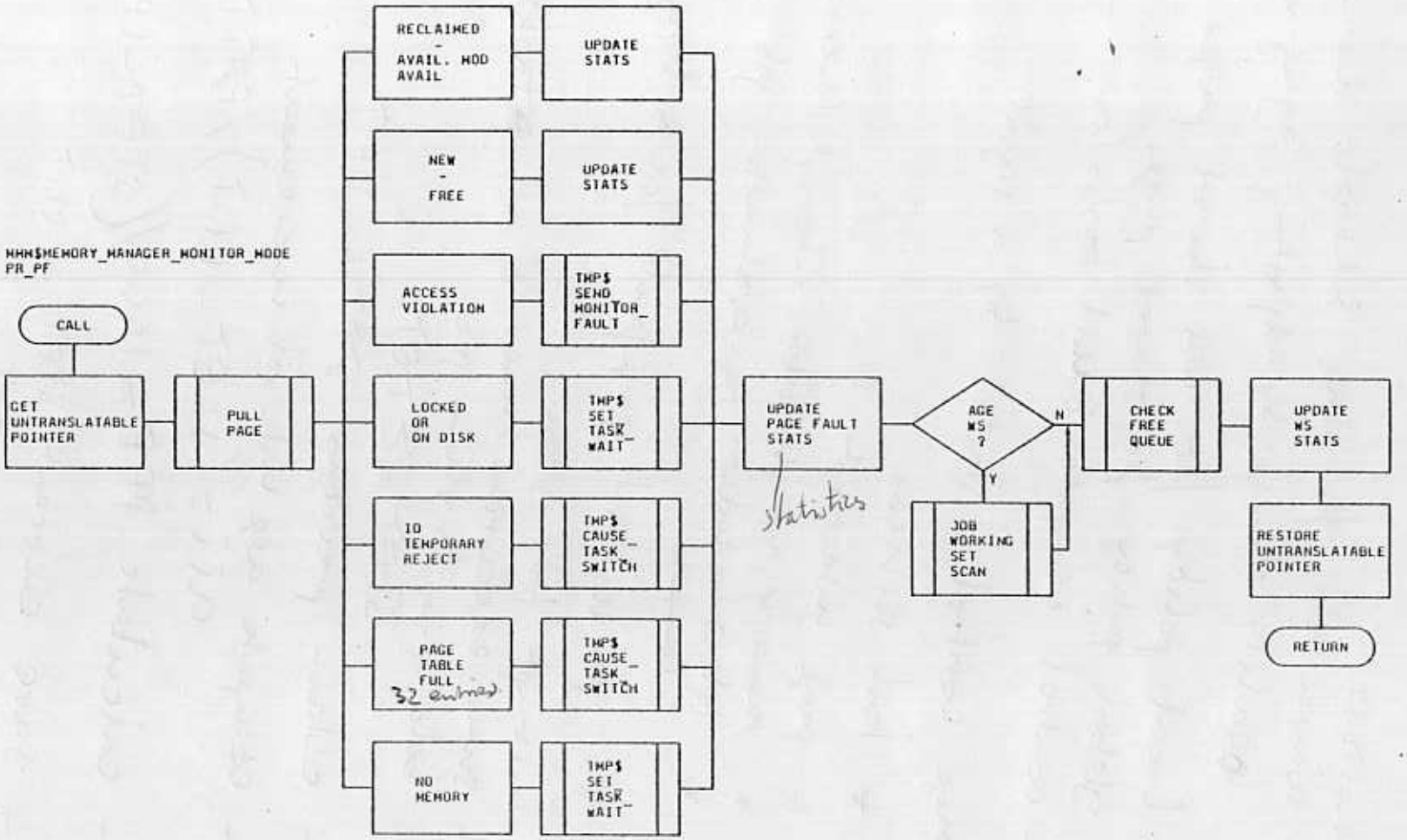


voor reeds gebruikt heet het Reclaimed page

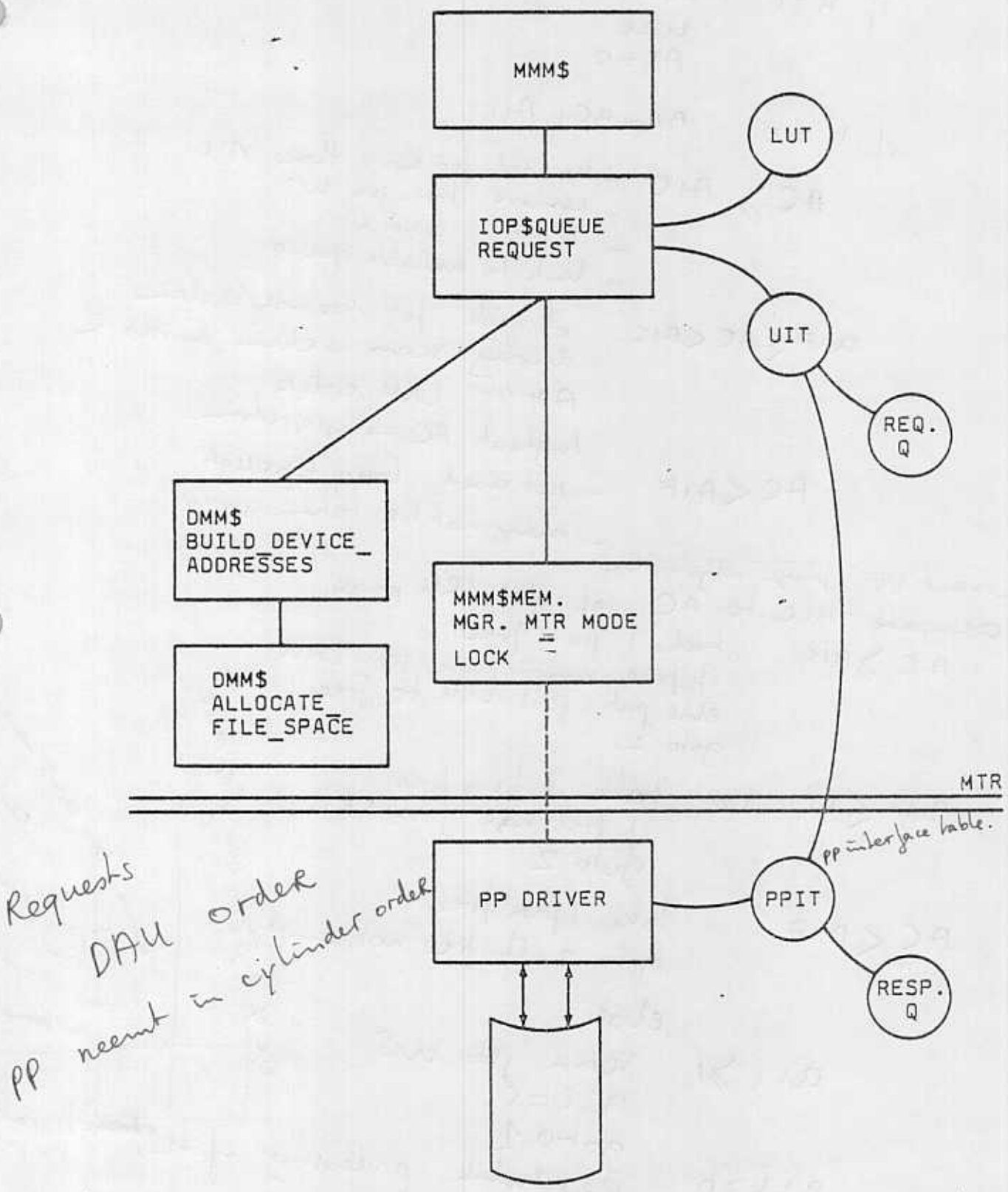
1] hardware houdt het bij
methoevaak

2] jekent de toekomst niet.
↳ PFF = page fault frequency

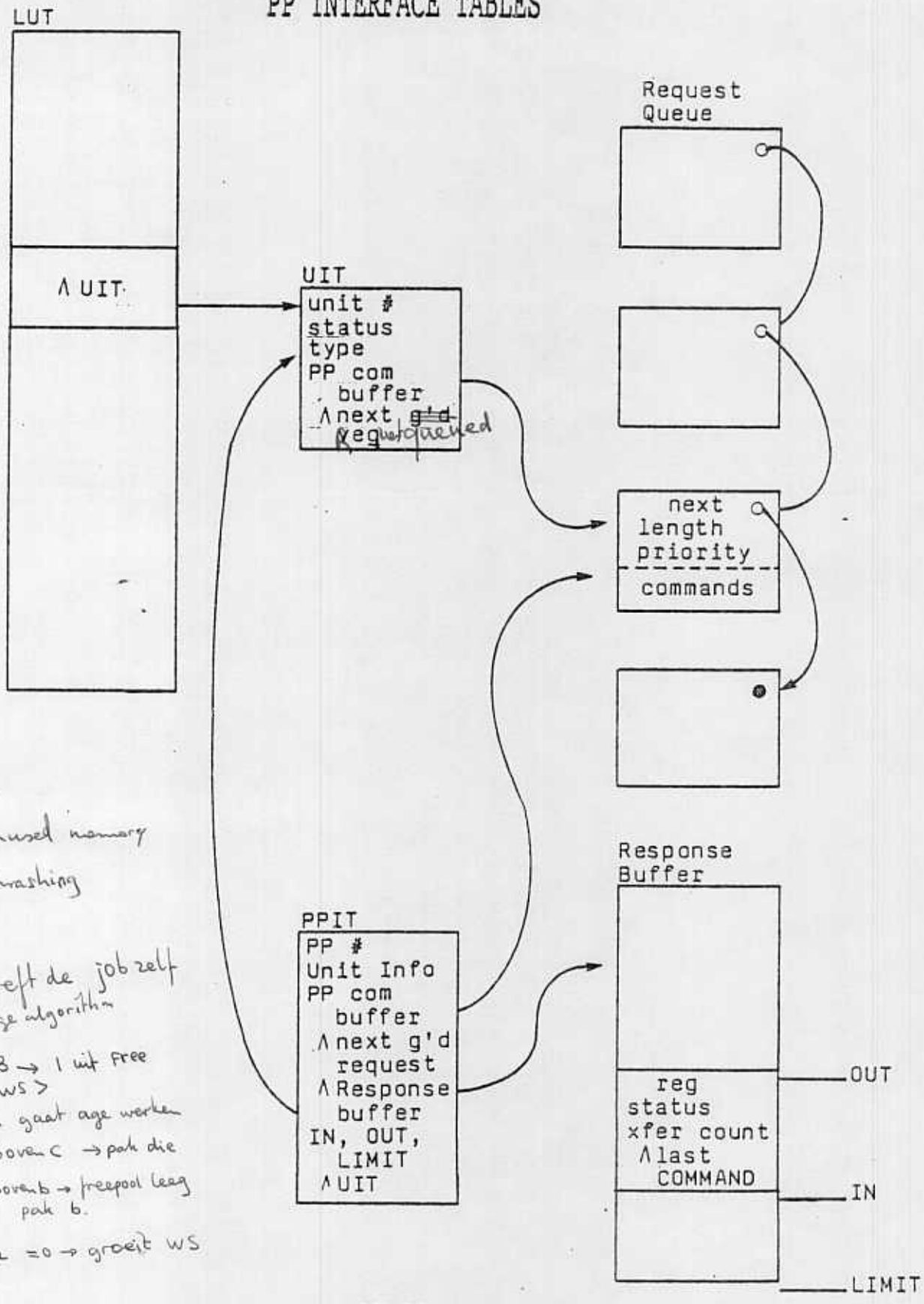
PROCESS PAGE FAULT



INITIATE PHYSICAL IO



PP INTERFACE TABLES



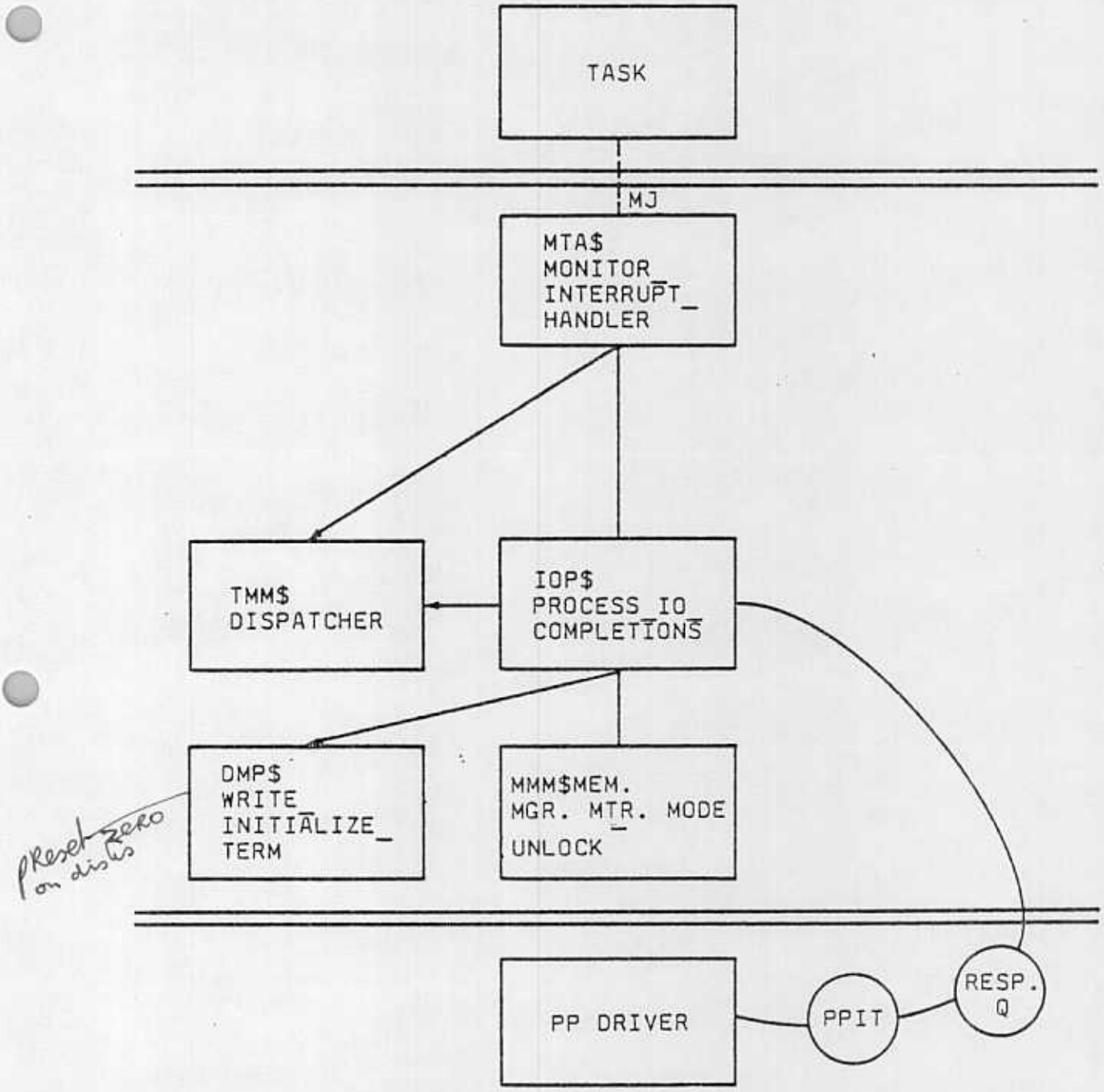
Ceiling te hoog unused memory
Floor te laag → thrashing

← algorithme betreft de job zelf
local page algorithm

alle pages in A-B → 1 uit free ws >
dan gaat age werken
→ boven c → pak die
→ boven b → freepool leeg pak b.

→ als age = 0 → groeit WS

COMPLETE IO REQUEST



NO ERRORS--Ready Task
PF ERROR--Notify PF Manager
READ ERROR--Abort Task
WRITE ERROR--Leave page in_memory

APPENDIX A

PACKAGING

NOS/VE DESIGN SPECIFICATION

PART III

SYSTEM PACKAGING

TABLE OF CONTENTS

1.0 SYSTEM STRUCTURE

A basic objective is to provide a well defined system structure which will result in a highly reliable system and one that can grow over time in an orderly and cost effective manner.

In order to meet this objective, a set of hardware and software conventions are imposed on both user and system code. This allows the normal protection, debugging, loading, code maintenance, accounting, and error handling methods of the user and the system to be the same. This also facilitates movement of services between user and system.

1.1 GENERAL STRUCTURE ELEMENTS

Jobs, tasks and modules represent the basic structure elements for all services provided by NOS/VE. They have the general relationship shown in figure 1. Each element has a set of unique execution attributes, interface conventions and resource requirements. System and application programmers make services available to users with combinations of these elements.

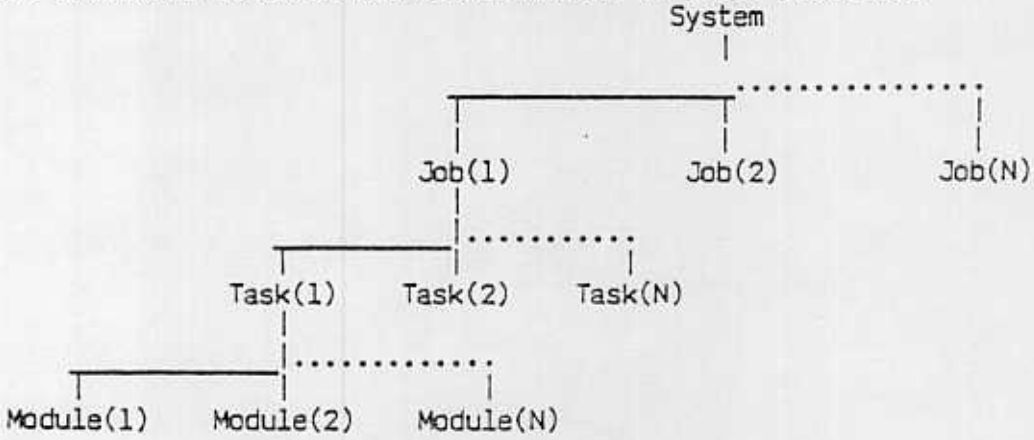


Figure 1 - Structure Elements

Each level contains a system element which monitors the progress of other elements within that level. The job level contains a system job which schedules, initiates, and terminates (normal or abnormal) user and system jobs. Within each job resides a system task which initiates and terminates tasks of the job. Within each task resides a collection of system modules which assist in the initiation and termination of the task.

1.1.1 JOB ELEMENT

The general facility for presenting work to the system is a job. Jobs run on behalf of a specific user whose identification is the basis of the system access control mechanisms. In addition to batch or interactive jobs that are submitted by end users, the operating system and various subsystems not initiated by end users also run as jobs. Since all jobs are protected and compete for resources via the same mechanism, it is anticipated that the addition of new subsystem jobs will be quite straightforward.

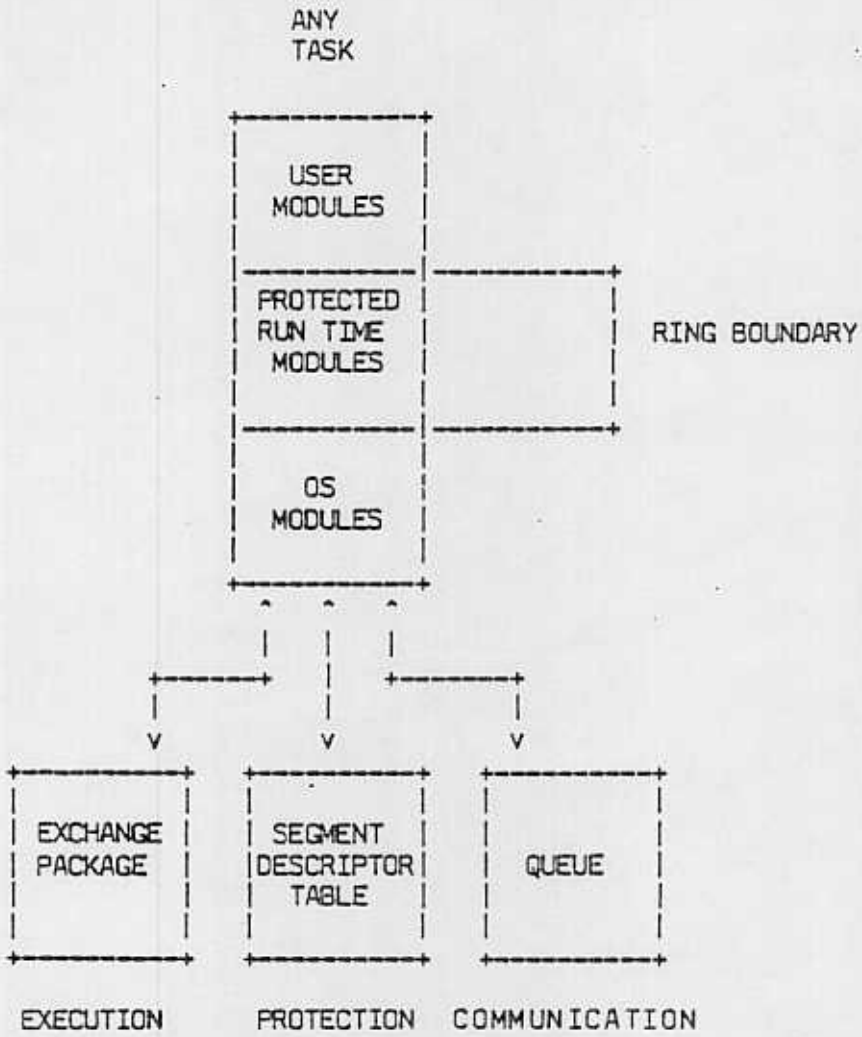
Every job consists of multiple tasks. An important characteristic of a job is that all tasks executing within the job share a common set of operating system services that are determined at the time of job initiation. These service modules, called task services, are the mechanism through which operating system functions are made available. They are constructed from a job template that is selected based on job type. This allows different jobs to have different services.

1.1.2 TASK ELEMENT

A task is the execution of a program. A program is a set of modules organized to perform some specific function (e.g. compile COBOL statements, copy a file). Tasks are protected from one another, can be dynamically created and destroyed, can communicate with other tasks and can execute asynchronous with other tasks. Tasks are the only asynchronous execution unit supported by NOS/VE.

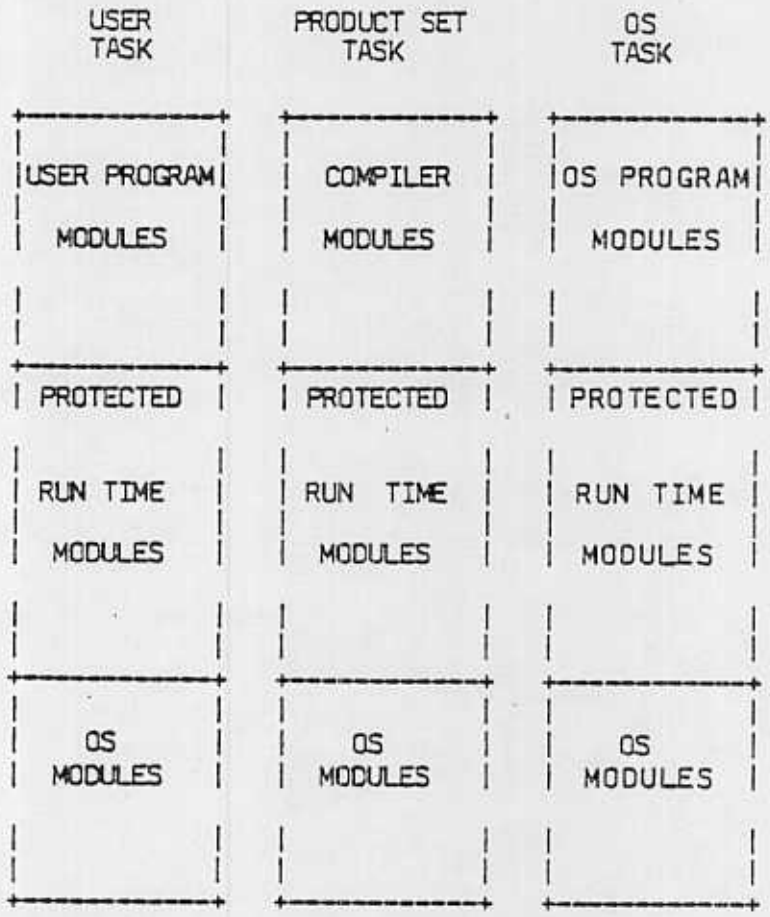
Tasks then are the environment for providing functions that are natural to place outside of the requesting environment. Tasks are requested via an operating system request. They have their own (clock) accounting, scheduling, and execution characteristics. Tasks can come and go independently and represent a mechanism which is used to control memory usage (e.g., each pass of a compiler as a separate task). Protection is enforced by different segment descriptor tables for the caller and callee.

The figure below illustrates a task environment.



Every task looks similar to NOS/VE in that it has an exchange package which defines execution status, a segment descriptor table which defines protection, a queue which defines a communication path and a collection of modules which define the program. The collection of modules can include "user" modules, application or run time service modules and operating system modules. The address space of each task is subdivided by a ring protection hierarchy. An attribute of a module is its ring of execution. Each task will include modules which are protected from each other by executing in different ring brackets.

All tasks, regardless of the type of function they perform, have the same appearance as illustrated below.



1.1.3 MODULE ELEMENT

Modules are the environment for the set of services that are natural to place within the environment of the caller. These services are provided as procedures and are interfaced via the standard procedure call. They have the same (clock) accounting, scheduling, and execution characteristics as the caller. Examples include file access methods, loading, table handling and Fortran object time. The available services can be added dynamically by explicit requests of the loader. Protection enforced by the ring hardware may exist between the caller and callee.

1.2 NOS/VE STRUCTURE

NOS/VE utilizes the task and module structure elements to package the operating system services. Some of its tasks execute as part of the "user" jobs and some execute as part of NOS/VE system jobs. NOS/VE also collects together a set of modules that perform the lowest level operating system functions into a special environment called the CPU Monitor. The operating system services are provided within three basic environments:

CPU Monitor (one per system)
NOS/VE Modules (modules within each task)
Operating System Tasks (executing within "user" jobs, and executing within "system" jobs)

Every request a user makes of the system is translated into communication with one or more of these environments. Whenever operating system extensions are being implemented, the conventions and interfaces of these environments must be understood and used.

1.2.1 CPU MONITOR ENVIRONMENT

CPU Monitor is that portion of the operating system that is most directly related to the hardware environment. It provides:

- Basic intertask communication (signals)
- CPU Dispatching
- Basic CPU Scheduling
- Changing Task Status
- Interrupt Handling
- Page Management
- Basic Physical I/O Management

CPU Monitor is interrupt driven, nonpageable, and represents the most thoroughly debugged, least frequently changed code within the operating system.

1.2.1.1 CPU Monitor Request Handling

CPU monitor requests are only made by Task Services and Task Monitor functions. These requests are made using the hardware exchange instruction. Parameters are passed in the hardware registers.

1.2.2 NOS/VE MODULES ENVIRONMENT

NOS/VE modules are the set of operating system modules that execute within the environment of a task. These modules perform the operating system functions that are most directly related to the requestor's environment. To provide for maximum protection and RAM these modules are divided into Task Services modules and Task Monitor modules.

1.2.2.1 Task Services Modules

Task services modules provide the user interface to NOS/VE capabilities for:

- File Management
- Access Methods
- Program Management
- Job Management
- Resource Allocation

Task services is a collection of protected procedures. These procedures are directly callable by user code via the call instruction. The call causes a change in privilege for the called procedure, allowing these operating system services to execute with more or different privileges than the calling procedure. This type of structure allows protected operating system services to execute within the user environment. Task services provide a central interface for all requests and responses made and received by a task. If the requested service is not supported directly by task services, the request is passed on to CPU Monitor or to an operating system task. Task services occupies rings 3 to 6 within each address space. Only ring 3 is used for release 1 of NOS/VE.

1.2.2.1.1 TASK SERVICES REQUEST HANDLING

There are multiple task service entry points gated to requestors. Every call to a task service must supply a status variable of type ost$status. The parameter rules will conform to those of CYBIL.

1.2.2.2 Task Monitor Modules

Task monitor modules perform the more privileged functions of NOS/VE and execute at rings 1 and 2. These modules are a collection of procedures that interface to NOS/VE basic system tables (e.g. segment table, system file tables, catalogs, execution control tables) and to the CPU Monitor. The ring 2

procedures manage job global tables (i.e. accessible in all tasks of a job). The ring 1 procedures manage system wide tables (i.e. accessible in all tasks of all jobs) and are more privileged and critical to the integrity of the system. Task Monitor procedures are not directly callable by "users"; only NOS/VE Task Services procedures can directly interface to Task Monitor procedures.

1.2.3 OPERATING SYSTEM TASKS

Operating system tasks are those portions of the operating system that are relatively independent of the requestor's environment. They may execute asynchronous to the requestor and provide major portions of:

- Job Management
- Job Scheduling
- Operator Communications
- Device Drivers
- Hardware Maintenance

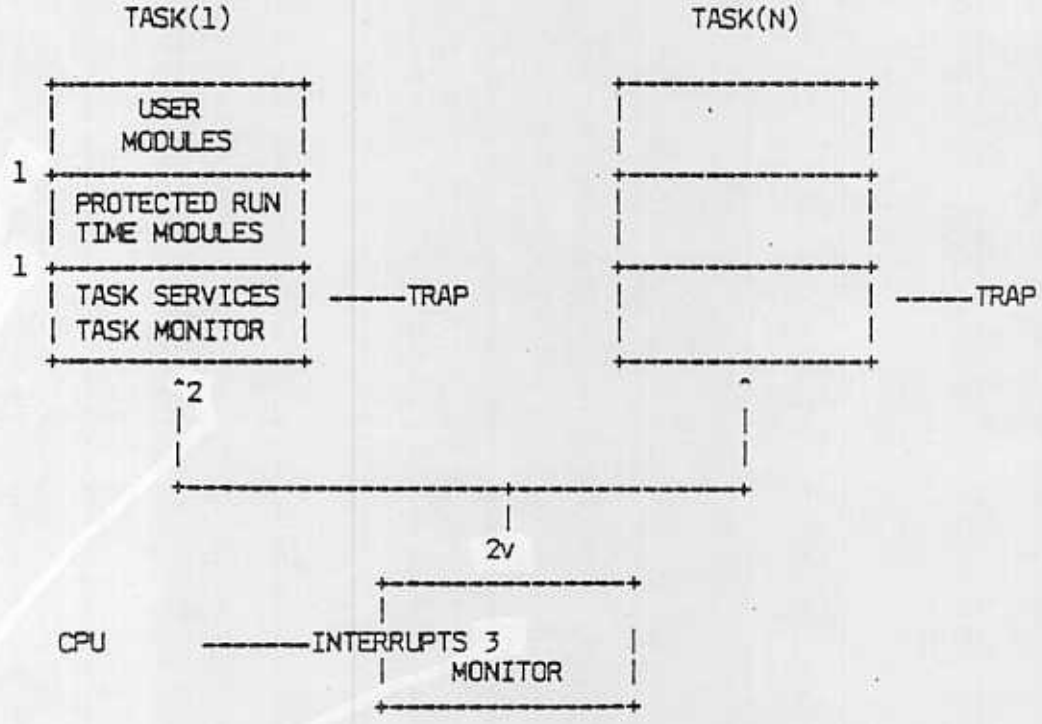
Execution of a system task is triggered by a signal passed into its communication queue. Tasks may execute in different processors. The device drivers, for example, are system tasks which execute on the IOU.

1.2.4 OPERATING SYSTEM COMMUNICATION

The operating system functions communicate using a basic signal handling service. The signals have a fixed format, a maximum size and are used by the operating system primarily for communication between address spaces. CPU Monitor is responsible for placing signals into the proper signal queue and for notifying the proper Task Monitor that a signal exists. Task Monitor is responsible for taking signals out of the communication queue and passing it to a Task Services signal handler. Routing, based on signal type, to a signal processor within Task Services will be effected by the Signal Handler.

1.2.5 OPERATING SYSTEM ENVIRONMENT SUMMARY

The following figure summarizes the basic environments and interfaces of NOS/VE.



1 - INTERFACED VIA THE CALL INSTRUCTION, CYBIL PARAMETERS FOR COMMUNICATION, RINGS FOR PROTECTION

2 - INTERFACED VIA THE SYSTEM CALL, SIGNALS FOR COMMUNICATION, SEGMENT TABLES FOR PROTECTION

3 - INTERRUPTS ARE PROCESSED BY CPU MONITOR OR ARE TRANSLATED INTO SIGNALS

1.2.6 SEGMENT USAGE

1.2.6.1 Ring Assignment for a User Task

AREA	DATA PORTION	CODE PORTION	WHEN CREATED
USER APPLICATION PROGRAM	WORKING STORAGE, STACK, USER DATA	APPLICATION PROGRAM	AT LOAD TIME ACCORDING TO LIBRARY LIST IN PROGRAM DESCRIPTOR
PROTECTED RUN TIME MODULES	WORKING STORAGE, STACK	DATA BASE MANAGER	
TASK SERVICES/ TASK MONITOR MODULES	WORKING STORAGE, STACK, TABLES FOR JOB, TABLES FOR SYSTEM	RECORD MANAGER LOADER, PROGRAM COMM., TRAP HANDLING	A JOB TYPE TEMPLATE SUPPLIED BY SYSTEM GENERATION WHICH IS USED BY JOB INITIATION

This diagram illustrates:

1. Examples of code which exist at each ring bracket
2. Examples of private data at each ring bracket
3. When the data and code segments are created

Entry points to task services are created by system generation within the loader symbol table and are dynamically linked to external references from user and protected run time procedures by the loader.

1.2.6.2 Segment Assignments for User Modules

The following example demonstrates how the loader allocates and initializes segments based on information contained in compiler generated object text.

- Object Text Topology

RECORD TYPE	SAMPLE CONTENTS
(identificaton record)	name, date, generator name
(section definition)	code, binding, working storage, protection
(interpretive text)	text, replication, bit, entry, external
(transfer...end of text)	

- Generated Object Text

CODE SECTION (R,X)	STATIC SECTION (R,W)
. Non selfmodifying instructions	. Modifiable data

BINDING SECTION (B)	LITERAL SECTION (R)
. Base address of other sections . All procedure descriptions	. Constant data

DYNAMIC WORKING STORAGE SECTIONS (R,W)

. Common blocks
. Data allocated at run time

- Mapping sections to segments (assume 2 modules) providing an executable entity.

LGO file Module 1
Module 2
EOF

Segments

Segment N (R,X)	Segment N+1 (B)	Segment N+2 (R,W)
Code Section M(1)	Binding Section M(1)	Static Data M(1)
Code Section M(2)	Binding Section M(2)	Static Data M(2)
		Any Named Common

Segment N+3 (R)	Segment N+4 (R,W,E)	Segment N+5 (R,W,E)
Literals M(1)	Universal Heap	Run Time Stack
Literals M(2)	(Grow)	(Grow)

The binding segment contains pointers to static, literals, code and other binding sections. The advantages of using segments include:

- Independent growth
- Integrity by separation
- Supports code sharing
- Non rewrite of code and constants (paging or swapping)

R - Read
E - Extensible
B - Binding
W - Write
X - Execute

2.0 SYSTEM TABLES AND INTERFACES

2.1 GENERAL GUIDELINES

The operating system is dependent on the use of tables to provide interfaces between different system modules and between the system and the user, and to describe the basic objects supported by the system and how these objects are related. When a table is defined within the system, consideration must be given to the following six general characteristics.

- Protection - Should the information be protected by hardware from inadvertent write operations? Must the information be protected from malicious write/read operations?

- Scope - Should the information be local to a user or should it be made global and shareable by other users? In general, information should be globally defined only when required. Keeping information local to a user has two advantages: 1) this information is private and no other user can interfere with it, and 2) if most of the tables required by a Job are collected locally, it is easier for the system to keep track of a user (swapping, restart, paging critical tables, etc.).

- Residence - Should the information be pageable or locked down? Whenever possible, information should be pageable. It should be locked down only when an obvious efficiency case exists. Three points can be made: 1) System Monitor cannot tolerate access interrupts, so any information referenced by System Monitor must be in real memory at the time of reference, 2) I/O channels use absolute addresses and require that real memory exists when in operation, and 3) there are degrees of paging, that is, some information must be present if a task is to use the CPU and can only be explicitly removed.

- Life Cycle - When will the table come into existence and when will it disappear? The data to describe a job is divided into environments which will go away, when the job terminates, when a task terminates, when the system crashes, and environments which will live forever unless explicitly removed.

- Crash Resistance - When the system crashes, how will the tables be reconstructed? What impact will there be on recovery if the tables cannot be reconstructed? Will the corrupting of the tables cause a system crash? What protection will be provided to detect corruption?

- Structure - The general structure of each of the NOS/VE Tables Area is the same and allocation of entries within a particular table is the same.

The contents (entries) of NOS/VE are position independent, that is,

a) the order and number of static entries in tables areas can vary from build to build;

b) the order and number of static entries in tables areas (task and job private) can vary among job types; and

c) the order and number of dynamic entries in the tables areas can vary among instances of execution.

The allocation of entries in NOS/VE tables should require minimal interaction among development projects; is controlled at the source level; via CYBIL; and is managed by execution and the system generator.

The general structure, allocation technique or order, value assignment tactics of NOS/VE tables should not impose undue constraints on the structure of entries contained in tables areas.

The allocation of entries and the assignment of values to entries in NOS/VE tables should be postponed as long as is feasible - priority order:

a) execution time
   1) first use time
   2) task initiation time
   3) job initiation time
   4) system initiation time
b) system generation time
c) source (compile) time.

2.2 TABLES AREAS

TABLES AREA	R1, R1
TASK SHARED	3, 13
TASK PRIVATE	3, 13
JOB PRIVATE PAGEABLE	2, 13
JOB PRIVATE FIXED	1, 3
MAINFRAME PAGEABLE	1, 3
MAINFRAME WIRED	1, 3

2.3 TABLES AREA GUIDELINES

2.3.1 JOB PRIVATE FIXED

The Job Private Fixed tables area is the container for tables shared among monitor and all tasks of a job. Job Private Fixed tables reside in non-pageable memory because of monitor access. Therefore, care should be exercised to minimize the amount of space allocated to entries which are not accessed by monitor.

2.3.1.1 Job Private Fixed Static Section

The Job Private Fixed static section is the container for statically allocated tables entries. Static entries are allocated at compile time, via CYBIL static variable declarations, which specify the Job Private Fixed tables area. Statically allocated table entries are those which are somewhat constant in nature for the duration of the job. Such entries may also be "root" pointers to dynamically allocated entries in the Job Private tables area.

The allocator of a static entry is responsible for the initial value assignment to that entry.

2.3.1.2 Job Private Fixed Dynamic Section

The Job Private Fixed dynamic section is the container for dynamically allocated (CYBIL allocate or next statements) tables entries. Dynamic entries vary in number and size - their lifetime is often less than the life of the job. Dynamic entries whose lifetime is less than that of the job must be freed (CYBIL free statement) when their lifetime expires - the responsibility for freeing lies with the ultimate allocator.

2.3.2 JOB PRIVATE PAGEABLE

The Job Private Pageable tables area is the container for tables shared among all tasks of a job. Table entries residing in this tables area are not accessible by monitor.

2.3.2.1 Job Private Pageable Static Section

The Job Private Pageable static section is the container for statically allocated table entries. Static entries are allocated at compile time, via CYBIL static variable declarations, which specify the Job Private Pageable tables area.

Statically allocated table entries are those which are somewhat constant in nature for the duration of the job. Such entries may also be "root" pointers to dynamically allocated entries in the Job Private tables area.

The allocator of a static entry is responsible for the initial value assignment to that entry.

2.3.2.2 Job Private Pageable Dynamic Section

The Job Private Pageable dynamic section is the container for dynamically allocated (CYBIL allocate or next statements) table entries. Dynamic entries vary in number and size - their lifetime is often less than the life of the job. Dynamic entries whose lifetime is less than that of the job must be freed (CYBIL free statement) when their lifetime expires - the responsibility for freeing lies with the ultimate allocator.

2.3.3 TASK PRIVATE

The Task Private tables area is the container for tables shared among procedures in task services and task monitor of a task. Task Private is pageable. Table entries residing in this tables area are not accessible by other tasks or monitor.

2.3.3.1 Task Private Static Section

The Task Private static section is the container for statically allocated tables entries. Static entries are allocated at compile time, via CYBIL static variable declarations, which specify the Task Private tables area.

Statically allocated table entries are those which are somewhat constant in nature for the duration of the task. Such entries may also be "root" pointers to dynamically allocated entries in the Task Private tables area.

The allocator of a static entry is responsible for the initial value assignment to that entry.

2.3.3.2 Task Private Dynamic Section

The Task Private dynamic section is the container for dynamically allocated (CYBIL allocate or next statements) table entries. Dynamic entries vary in number and size - their lifetime is often less than the life of the task. Dynamic entries whose lifetime is less than that of the task must be freed (CYBIL free statement) when their lifetime expires - the responsibility for freeing lies with the ultimate allocator.

2.3.4 MAINFRAME PAGEABLE

The Mainframe Pageable tables area is the container for tables shared among all jobs in the system. This tables area is writable by R1 task monitor and readable up to task services. The mainframe pageable tables area is not accessible to monitor.

2.3.4.1 Mainframe Pageable Static Section

The Mainframe Pageable static section is the container for statically allocated table entries. Static entries are allocated at compile time, via CYBIL static variable declarations, which specify the Mainframe Pageable tables area.

Statically allocated table entries for those which are somewhat constant in nature for the duration of the system. Such entries may also be "root" pointers to dynamically allocated entries in the System Private tables area.

The allocator of a static entry is responsible for the initial value assignment to that entry.

2.3.4.2 Mainframe Pageable Dynamic Section

The Mainframe Pageable dynamic section is the container for dynamically allocated (CYBIL allocate or next statements) table entries. Dynamic entries vary in number and size - their lifetime is often less than the life of the system. Dynamic entries whose lifetime is less than that of the system must be freed (CYBIL free statement) when their lifetime expires - the responsibility for freeing lies with the ultimate allocator.

2.3.5 MAINFRAME WIRED

The Mainframe Wired tables area is the container for tables shared among monitor and all jobs in the system. The Mainframe Wired tables reside in wired memory due to monitor access. Therefore, care should be exercised to minimize the amount of space allocated to entries which are not accessed by monitor.

2.3.5.1 Mainframe Wired Static Section

Only monitor software can allocate static table entries in the Mainframe Wired static section.

The Mainframe Wired static section is the container for statically allocated table entries. Static entries are allocated at compile time, via CYBIL static variable declarations, which specify the Mainframe Wired tables area.

Statically allocated table entries are those which are somewhat constant in nature for the duration of the system. Such entries may also be "root" pointers to dynamically allocated entries in the System Private tables area. The allocator of a static entry is responsible for the initial value assignment to that entry.

2.3.5.2 Mainframe Wired Dynamic Section

The Mainframe Wired dynamic section is the container for dynamically allocated (CYBIL allocate or next statements) table entries. Dynamic entries vary in number and size - their lifetime is often less than the life of the system. Dynamic entries whose lifetime is less than that of the system must be freed (CYBIL free statement) when their lifetime expires - the responsibility for freeing lies with the ultimate allocator.

APPENDIX B

EXERCISES

DAY 1 ASSIGNMENTS

1. Read the SYSTEM PACKAGING tract in Appendix A of the handout.
2. Get copies of the following:
   a. NOS/VE Procedures & Conventions (1/person)
   b. Loadmap in the DEV1 catalog (1/4 people)
   c. Hints and Integration Procedures Notebook (Optional)
   d. Sorted list of NOS/VE Modules (1/person)
   e. Internal Interface (1/person)
3. Answer the following questions
   a. What is a task? What does each task have uniquely? (tables, modules, etc) What does each task share with other tasks of the same job?
   b. What is a job template? How does it relate to the system core?
   c. How do jobs interface to one another; tasks; modules?
   d. Note the privelege of each of the following components; name a function of each:
      1. CPU Monitor
      2. R1 (Task Monitor)
      3. R2 (Task Monitor)
      4. R3 (Task Services)
      5. Operating system task
   e. The Execution Control (XCB) contains the exchange package for a task. Its residence is in the Job Fixed (JF) segment. SCL Interpreter's block stack is changed whenever a command is processed. It is resident in the Task Shared (TS) segment.

      Discuss the choice of residence for these tables based on the following criterion:
      Protection
      Scope
      Residence
      Life Cycle

      When is each table initiated?

DOCUMENT RETRIEVAL

1.	NOS/VE Procedures & Conventions	SES,MAD.LISTPC	1
2.	LOADMAP	SES,DEV1.NVEMAP PMPXX TWO PRINT SES,DEV1.NVEMAP PMPXXYY TWO PRINT	2
3.	Helpful Hints	SES,DEV1.LISTHINTS	
4.	Sorted List of NOS/VE Modules	SES,DEV1.NVEREP PRINT	2
5.	Internal Interface	SES,MAD.LISTCII or SES,MAD.LISTNII	1
6.	Integration Procedures Notebook	ATTACH,IPNDOC/UN=DEV1 SES.PRINT,IPNDOC	2
7.	Selected Modules	SES,DEV1.LISTNVE.. (list of deck names).. PRINT	2
8.	Tables	SES,DEV1.COMLIST.. id=(list of 2-char ids).. apl=NOSVEPL un=INT2.. PRINT	
9.	Selected Tables - 'tab' is a CYBIL module with *call's.	SES.GENCOMP sf=tab.. b=NOSVEPL un=DEV1 NONEST	
10.	Common Deck Cross Reference	SES,SCL.XREFCD	
11.	General Internal Design	ATTACH,GIDP1R6/UN=MAD SES.FORMAT GIDP1R6 TXTFORM	

NOTES:

1. Documented in NOS/VE Procedures and Conventions
2. Documented in Integration Procedures Notebook

DAY 2 ASSIGNMENTS

1. Use a LOADMAP to complete the table on the next page.

2. Read Ch. 2 (CPU Monitor) in Internal Interface.
   Ch. 8 (Program Mgmt) in Internal Interface.
   Ch. 30 (Intrinsics) in Internal Interface.
   Ch. 8 (Program Library Conventions) in NOS/VE Procedures & Conventions.
   Ch. 1 (NOS/VE System Overview) in Integration Procedures Notebook.
   Ch. 2 (Overview of Integration Process,(Sections 1-4) in Integration Procedures Notebook.)

3. Fill in the events on the provided time line
   a. User program is running (R11)
   b. User establishes a condition handler for arithmetic overflow (R3)
   c. pmp$establish_condition_handler returns.
   d. User generates the overflow condition; trap handles runs.
   e. Trap handler calls user condition handler.
   f. User condition handles returns.
   g. Trap handler returns.

4. Fill in the events on the provided time line
   a. SCL Interpreter reads 'ATTACH_FILE...' from $COMMAND (R11) and calls the command processor (clp$attach_command) (R11)
   b. clp$attach_command calls pfp$attach (R11)
   c. pfp$attach calls pfp$r2_attach (R2)
   d. pfp$r2 attach calls pmp$delay because file is not available (R2)
   e. pmp$delay exchanges (to monitor)

      ⋮

   f. Monitor exchanges after delay
   g. pmp$delay returns
   h. pfp$r2_attach calls fmp$attach_job_file since file is available; LNT and JFT entries will be built.
   i. fmp$attach_job_file returns
   j. pfp$r2_attach returns
   k. pfp$attach returns
   l. clp$attach_command returns
   m. SCL Interpreter reads from $COMMAND.

Module Name	Procedure / Variable Name	PVA	Rings
Tmm $ dispatcher 228 page8 page 85	TMP $ delay page8 XDCL	6CBB 1006 0000 ~~70F0~~	(1,1,3) Monitor mode
pmm$ Runanywhere_ prg_services	PMP $ Cycle 1,14,3648 page 36	3608 D0 14 0000 ~~1CF4~~	(1,D,D) System Core JOB mode
MONITOR INTERRUPT HANDLER 93C	MTP $ Begin XDCL TRap_handler XDCL	1 6 0 1 6 4B0	(1,1,3) System core monitor mode.
Sym$ JOB_fixed_template Jstack2 page 41 800	> JOB STK2 XDCL	3830 B01D 0000 ~~3E40~~	(1,1,1)
BAM $ open page16 50F	BAP$OPEN Gated	2 01C 5C0	(2,3,D)
page 761	BAP$ OPEN gated = 5C0	3580 3 B016 0006 ~~0F30~~	(2,3,D)
Sym $ JOB_fixed_template page 41	OSV $ JOB_fixed_heap	1850 1 003 0000 ~~1E6P~~	(1,3,3)
Jmm$ initiate_job environment page 46	XDCL OSV$ JOB_PAGEABLE_HEAP	2 4 147A page 48	(2,3 3)
CLM_interpret commands page 199 (440) COMMAND INTERPRETER	CLM $ include (~~440~~) CLP$ scan_command_file scan_command_line CLP$ interpret_commands XDCL page 199	2 1D 19CE0	(2,D,D)

B-4

Time	HARDWARE XP	R11 Stack	R3 Stack	R2 Stack	R1 Stack	Monitor stack	Task XP Area	Monitor XP Area	Event
T_0	B # N1	B frame1	—	—	—	M fR1	program starter	MiP	program running
T_1		params							prepare parameters on stack.
T_2	B # n2 call	B frame2 SFSA							call pmp$ establish_condition_handler 2- 1D-' 5EB70 (page 233 2DD
T_3	put hooks for trap handler		add pointer to stack						
T_4	B # n1 return	pop SFSA							
T_5	arithoverflow								
T_6	trap interrupt B # n2	SFA B frame2							trap handler
T_7	trap handler B # n3								user TRAP handler
T_8	build environment for cond-handler	SFSA'							return
T_9	Return to condition handler								
T_{10}	Return # from condition handler	pop sfsa				on condition flag. (in			stack frame van caller)
T_{11}	Return	SFSA							
T_{12}									
T_{13}									

Time	HARDWARE XP	R11 Stack	R3 Stack	R2 Stack	R1 stack	Monitor stack	Task XP Area	Monitor XP Area	Event
T_0	B ≠ n1	B fr1	—	—	—	m. fr1	program starter	mip	Scl runs RingB jobmonitor
T_1	B ≠ n2	SFSA1 / B fr2							CLP$ ATTACH_command 2DD
T_2	B ≠ n3	SFSA2 / B fr3							PFP $ ATTACH 2DD → PFP$ R3_attach
T_3	3 n / 2 n1	SFSA3	3 fr1 / SFSA	2 fR1					PFP$ R2_ATTACH 223
T_4	2 n2			SFSA1 / 2 Fr2					PmP$ delay 1DD
T_5	m n						2 n2		exch → mtR
T_6	2 n2							mip	exch from monitor
T_7	2 n1			2 fr1					PFP$ R2_attach back again
T_8	2 n2			SFSA1 / 2 fr2					FmP$ attach_jobfile 223 CNT ring1 & JFT
T_9	2 n1								PFP$ R2_ATTACH
T_{10}	B n3	SFSA2 / B fr3							PFP$ ATTACH
T_{11}	B n2	SFSA1 / B fr2							CLP$ ATTACH_command
T_{12}	B n1	B fr1							Scl Running command.
T_{13}									

DAY 3 ASSIGNMENTS

1. Read

   Ch. 14 (Memory Link Interface) in Internal Interface.

   Ch. 10 (Key point Usage) in NOS/VE Procedures & Conventions.

   Ch. 11 (Message Handling) in NOS/VE Procedures & Conventions.

2. Describe SCL Interpreter's block stack entries for the following command sequence:

```
LOGIN
GET_FILE CSOURCE A6
CYBIL CSOURCE
LGO
REPLACE_FILE report A6
```

APPENDIX C

DUMP

DUMP QUESTIONS

1. Orientation
   a. Where did the processor stop?
      (NOS, NOSVE monitor state, NOSVE job state)
      P= 1006.0001.1538 → ring1 MPS herberyt beginsituatie
      NOSVE monitor state.

   b. Was there a hardware error?
      SS.P3 = 28 SS.104 = 10

   c. Three exchange packages are dumped. What state is each? What program is 'represented' by each? What real address are they at?
      page2 MPS at 252000
      page4 JPS at 6B3260
      page1 XJ in Registers.

   d. Why was monitor entered? (Look at MCR)
      MCR 0020 Job monitor

   e. How many active stack frames are there in monitor's stack? in the current job stack? 11

T1	100F	0000	0308
T2	2010	0000	03C0
T3	3011	0000	08F0
TB	B03B	0000	1F68

   f. Mark the end of the stack in the dump of the stack segments. Use the TOS registers from the current job exchange package.
      B not present.

CNV$ controller_data_pointer

2. Call stack of current job.
   a. If your load map matched the dump (which it doesn't) which procedure exchanged to monitor? Trace the previous stack frame (PSA) address back 3 frames.

   b. What module was the first to run in ring B? the last?

   c. What segment is the binding segment for rings 1, 2, 3, B?

   d. Find AMM$RETURN in program interface and the loadmap. Use the dump of stack frame 7 to determine what file is being returned.

1 003 100

1 001 DgBO
1 003 0000 = 3011 0000 0020 0000

3. Job Fixed for current job
   a. Find jmv$jmtr_xcb and jmv$jcb in the loadmap. What is the value of these variables?

   b. What is the system generated name for the job?

   AAKT

   c. What does the p-address field in the exchange package in the XCB contain?

   301D AB276

   d. The segment description table (SDT) address starts at 2ED (in the SCB). How many entries appear to be in the SDT? (They are 1 word long and end at AE8.

   17*4 = 68 = 104B =

   1000100

```
 14:12:49     restart file_name=CRASH sim180
 14:12:49      ** I SM 6052: CYBER 180 SIMULATOR V6.8 LEV 133 (NIGOS REV 5)
** **
*     NOTICE:    Simulator status change
*                      enter
*              'SES,STATUS.SIM180'
** **
 14:15:03     QR DL=P3_PROCESSOR_registers
PROCESSOR MAINT REGISTERS

                                                                  LABEL = P3_PROCESSOR_REGISTERS
        BC=0000 CC00                          CCE=0000                          CFF=   0
       CMA=0000 0C00 0000 0005                CMB=0000 0000 0000 0301         CMCE=0000 0000 0000 0000
       CME=0000                               DEC=0000 0000 0FC0 21F8           DI=00
       DLP=0000 000Q 0000                      DM=00                          EID=0030 0002
       JPS=006B 3260                           KC=0000 0000                   KCN=   0
        KM=0000                               MCE=0000                        MCR=0000
       MDF=0000                               MDW=0000 0000 0000 0000          MM=FFFC
       MPS=0025 2000                          OCF=   0                         OI=0000 0000 0000 0007
         P=0000 4006 0001 1538                PID=00                         PIT=FA58 512D
       PFS=0000 0000 0000 0000               PFS2=0000 0000 0000 0000        PFS3=0000 0000 0000 0000
      PFS4=0000 0C00 0000 0000               PFS5=0000 0000 0000 000C        PFS6=0000 0000 0000 0000
      PFS7=0000 0C00 0000 0000               PFS8=0000 0000 0000 0000        PFS9=0000 0000 0000 0000
      PFSA=0000 0C00 0000 0000               PFSB=0000 0000 0000 0000        PFSC=0000 0000 0000 0000
      PFSD=0000 C000 0000 0000               PFSE=0000 0000 0000 0000        PFSF=0000 0000 0000 0000
      PMF1=0000 00D0 FF10 00D0               PMF2=00E0 FFFF 04FF 0000             010D 050C 0703 9014
           E928 846E 2C9A F3E5                    0120 E287 0000 0064             0000 550D 0060 7183
           008D 0A82 3308 A88D                    0000 0000 0000 0000             0000 0000
        PSM=70                                 PTA=0020 0000                   PTL=07
        PTM=0000 C000 0000 0000               PTM1=0000 0000 0000 0000        PTM2=0000 0000 0000 0000
       PTM3=0000 0000 0000 0000                RCE=0000                       RDMP=0000 0000 0000 0000
        SIT=FA58 D3AE                           SS=2828 2828 2828 2828         STA=0025 21A0
        STL=0078                                TE=   2                         IP=1007 0000 0360
        UCR=0000                                UM=FF7F                        UTP=0000 0000 0000
       VMCL=C000

 14:15:05     DR PMR=ALL FN=SESLOG
 14:15:06     QR DL=I3_IOU_registers
IOU MAINT REGISTERS

                                                                  LABEL = I3_IOU_REGISTERS
       DEC=0000 CC00 0000 0000               EID=0000 0000 0220 FFFF         FSM=0000 0000 0000 0000
       FS1=0000 0C00 0000 0000               FS2=0000 0000 0000 0000          OI=0000 0FFF AFFF 0F1F
       OSB=130F 1F1F 0000 0100                SR=0000 0000 0000 BF00          SS=1010 1010 1010 1010
        TM=0000 0000 0000 0000

 14:15:07     DR IMR=ALL FN=SESLOG
 14:15:09     QR DL=M3_MEMORY_registers
MEMORY MAINT REGISTERS

                                                                  LABEL = M3_MEMORY_REGISTERS
         BR=4000 0000 0FF6 0000               CEL0=0000 0000 0000 0000        CEL1=0000 0000 0000 0000
      CEL2=0000 C000 0000 0000                CEL3=0000 0000 0000 000C         DEC=0000 0000 0200 0000
       EID=0000 0000 0130 0002                  OI=0010 0000 0000 0000          SS=0000 0000 0000 0000
     UEL1C=0000 CC0C 0000 0000              UEL11=0000 0000 0000 0000       UEL12=0000 000C 0000 0000
     UEL13=0000 0C00 0000 0000              UEL20=0000 0000 0000 0000       UEL21=0000 0000 0000 0000
     UEL22=0000 0C00 0000 0000              UEL23=0000 0000 0000 000C
```

```
 14:15:13    OR RF=ALL FN=SESLOG
 14:15:15    OR DL=ACTIVE_EXCHANGE_PACKAGE
LIVE EXCHANGE REGISTERS

        A0=1008 C00C 0800
        A3=1007 C000 0078
        A6=1001 0003 5CA0
        A9=1C08 0C0C 0796
        AC=1C01 C001 7458
        AF=10C8 CC0C 0D37
        D1=00
        EA=   0
       KEF=   0
       LRN=0001
       MDW=0000 0C00 0000 0000
         P=0000 1C06 0001 14FE
       STA=0025 21A0
      TOS1=1008 CC0C 0830
      TOS4=0000 C000 0000
      TOS7=0000 CC00 0000
      TOSA=0000 0C00 0000
      TOSD=0000 0C00 0000
        TP=1007 CC00 0360
       UTP=0000 0000 0000
        X0=0000 1006 0002 03A3
        X3=0000 0000 0000 0621
        X6=0000 0C00 0007 C000
        X9=0000 0000 7FFF FFFF
        XC=0000 0C00 0000 0000
        XF=0000 CC00 0000 0004

        A1=1008 0000 0830
        A4=1008 0000 0798
        A7=1008 0000 0798
        AA=1006 0000 8380
        AD=1000 0000 0000
        BC=0000 0000
       DLP=0000 0000 0000
        KC=0000 0000
        KM=0000
       MCR=0000
        MM=FFFC
       PIT=FA58 79B2
       STL=0078
      TOS2=0000 0000 0000
      TOS5=0000 0000 0000
      TOS8=0000 0000 0000
      TOSB=0000 0000 0000
      TOSE=0000 0000 0000
       UCR=0000
     UVMID=   0
        X1=0000 0000 0000 0000
        X4=0000 0000 0000 8060
        X7=0000 0000 0001 76DE
        XA=0000 0000 0001 A9A0
        XD=0000 0000 0000 0003

                                    LABEL = ACTIVE_EXCHANGE_PACKAGE
        A2=1008 0000 07C8
        A5=10C8 0000 0758
        A8=1001 0C00 CCD0
        AB=1001 0002 192E
        AE=1008 0000 09A0
       CFF=   0
        DM=00
       KCH=   0
      LPID=00
       MDF=0000
       OCF=   0
       PND=   0
        TE=   2
      TOS3=0000 0000 0000
      TOS6=0000 0000 0000
      TOS9=0000 0000 0000
      TOSC=0000 0000 0000
      TOSF=0000 0000 0000
        UM=FF7F
      VMID=   0
        X2=0000 0000 0000 056A
        X5=0000 0000 0000 7FFF
        X8=0000 1001 000A A920
        XB=0000 0000 0000 0000
        XE=0000 0000 0000 14E5

 14:15:16    DR LEX=ALL FN=SESLOG
 14:15:18    OR DL=EXCHANGE_PACKAGE_at_MPS
DISPLAY EXCHANGE PACKAGE

    P=C000 1006 0000 02EA   MCR=0000                UCR=0000
   MM=FFFC                   UM=FF7F              UTP=0000 0000 0000
UVMID=   C                  TP=1007 0000 0360      TE=   2
  DLP=0000 0000 0000        DM=00                  DI=00
  STL=0078                 LRN=   1                KC=0000 0000
  KCH=   0                 PIT=FFFF 0B6D          CFF=   0
  KEF=   C                 PND=   0                BC=0000 0000

                                    LABEL = EXCHANGE_PACKAGE_AT_MPS
 MDW=0000 0000 0000 0000
VMID=   0
LPID=00
 STA=0025 21A0
  KM=0000
 OCF=   0
 MDF=0000

 14:15:19    DEP EXC=MON FN=SESLOG
A REGISTERS

                                    LABEL = EXCHANGE_PACKAGE_AT_MPS
     A0 = 1008 0C00 0588                A1 = 1008 0000 0568          A2 = FFFF 8000 0000
     A3 = 1007 CC00 0000                A4 = 1003 0000 6230          A5 = 1001 0000 0000
     A6 = 1021 0C00 D260                A7 = 1001 0000 2890          A8 = 1001 0000 0358
     A9 = 1001 0000 CC48                AA = 1001 0002 11C0          AB = 1001 0002 192E
     AC = 1001 0005 3988                AD = 1000 0000 0000          AE = 1007 0000 0210
     AF = 1021 C000 D2E8
X REGISTERS

                                    LABEL = EXCHANGE_PACKAGE_AT_MPS
     X0 = 0000 0000 0000 0000           X1 = 0000 0098 3CF5 D9AA  <  X2 = 0000 0000 0003 68EB     h
     X3 = 0000 0C00 0000 0020           X4 = 0000 0000 0000 0000     X5 = 0000 0000 FFFF F1CD
     X6 = 0000 0000 0000 0736  6        X7 = 0000 0000 0001 0398     X8 = 0000 0000 0000 0010
     X9 = 0000 0CC0 0000 0FFF           XA = 0000 0000 0001 A9A0     XB = 0000 0000 0000 0000
     XC = 0000 0C00 C000 0C00           XD = 0000 0000 0000 0003     XE = 0000 0000 0000 1698
     XF = 0000 0CC0 0000 003C  <
TOS REGISTERS

                                    LABEL = EXCHANGE_PACKAGE_AT_MPS
     T1 = 1008 0000 0830
```

```
  14:15:20     DR A=ALL X=ALL T=1 EXC=MON FN=SESLOG
  14:15:22     QR DL=SDT_at_MPS
1ABSOLUTE  002521A0..C025248F                                                    LABEL = SDT_AT_MPS  3

 002521A0  CA1106CC C0000000   CA1302C0 00000000   00000000 00000000   BA1FFFFF 00000000
 002521C0  FE1F8CCC C0C00000   BE1180C1 0000C000   88110474 00000000   8C11C440 00000000                t       a
 002521E0  8A110040 00000000   00000000 0000C000   00000000 00000C00   0000C000 00000000   a
 00252200  00000000 00000000   000C00C0 0000C000   00000000 C0000C00   0000C000 00000000
           000001 Duplicate Lines
 00252240  CA13042F C0000000   CA1307ED 00000000   CA130669 00000000   CA13035D 00000000   /               I       ]
 00252260  CA13033C 00000000   CA1300C9 0000C000   CA13046A 00000000   CA13C10E 00000000   <               J
 00252280  CA138310 00000000   CA1384EC 0000C000   CA130445 00000000   CA138394 000C0000                   E
 002522A0  CA138602 00000000   CA138568 0000C000   CA1306BE 00000000   CA13803A 00000000   h               :
 002522C0  CA130148 00000000   CA138020 0000C000   CA1305E2 C0000000   CA130169 00000000   K               I
 002522E0  CA138692 00000000   CA130457 00000000   CA1302DE 00000000   CA1301F4 C0000000                   W
 C0252300  0A130728 00000000   0A138034 00000000   00000000 00000000   0000C000 00000000   +               4
 00252320  00000000 C0000000   00000000 0000C000   00000000 00000000   00000000 00000000
           000012 Duplicate Lines
  14:15:23     DMI S=STA RC=100 EXC=MON AH=RMA FN=SESLOG
  14:15:24     QR DL=MONITORS_CALL_STACK

* STACK FRAME 001                                                                LABEL = MONITORS_CALL_STACK
* P=1 006 000002EA    MTMSMONITOR_INTERRUPT_HANDLER  +000002EA
* STACK FRAME 001

* END OF STACK FRAME ENCOUNTERED

  14:15:25     TB RC=30 EXC=MON DF=DBGFILE FN=SESLOG
  14:15:27     QR DL=MONITORS_CALL_STACK

* STACK FRAME 001                                                                LABEL = MONITORS_CALL_STACK
* C0C0C0C0     1021C000  DZE8C800      1
* 00CC0008     00001001  00000358            X
* 00C00010     0000C0C0  00000000
* 00000018     000CC000  00000000

* SAVE AREA

* P=1 006 000002EA   VMID=0  MTMSMONITOR_INTERRUPT_HANDLER    +000002EA
* UM=FF7F    UCR=0000    MCR=0000

* A0=1 008 00000568      A1=1 008 C00CC568
* A2=F FFF 80C00000      A3=1 007 000C0C00
* A4=1 003 00006230      A5=1 001 00000000
* A6=1 021 0C00D260      A7=1 001 C00C2890
* A8=1 001 C0000358      A9=1 001 0000CC48
* AA=1 001 000211C0      AB=1 001 C002192E
* AC=1 C01 00053988      AD=1 000 C00CC000
* AE=1 007 00000210      AF=1 021 000CD2E8

* X0=000C0C00  00000000      X1=0000005B  3CF5D9AA
* X2=0C000000  000368E8      X3=00000CC0  00C00020
* X4=0C0C0000  0CC00C00      X5=C0C00C00  FFFFF1CD
* X6=0C0CCC00  00000736      X7=00000000  00010398
* X8=00000C00  00C00010      X9=00000000  00C00FFF
* XA=000CC000  00C1A9A0      XB=00C00C00  00C00000
* XC=000C0000  00C00000      XD=00000C00  00000003
* XE=00000000  00C01698      XF=00000000  00C00C3C

* END OF STACK FRAME ENCOUNTERED
```

```
 14:15:35    DM 1C0000000(16) EXC=MON RC=2000 AM=PVA FN=SESLOG
 14:15:40    QR DL=EXCHANGE_PACKAGE_at_JPS
DISPLAY EXCHANGE PACKAGE
                                                                  LABEL = EXCHANGE_PACKAGE_AT_JPS  4
    P=C00C 1014 0000 1FF6   MCR=0020                UCR=0000                MDW=0000 0000 0000 0000
   MM=FFFC                   UM=FF77                UTP=1FFF 7FFF FFFF      VMID=    0
UVMID=    0                  TP=100E 0000 0CF8       TE=    2             LPID=00
  DLP=FFFF 8000 0000         DM=00                   DI=00                 STA=0068 3620
  STL=0064                  LRM=   F                 KC=0000 0000           KM=0000
  KCN=    0                 PIT=766C 2577           CFF=    0             OCF=    0
  KEF=    0                 PND=    0                BC=0000 0260         MDF=0000

 14:15:41    DEP EXC=JOB FN=SESLOG
A REGISTERS
                                                                  LABEL = EXCHANGE_PACKAGE_AT_JPS
     A0 = 100F 0C00 0308                A1 = 100F 0000 0308                A2 = 100F 0000 0280
     A3 = F000 0000 0000                A4 = 100F 0000 0268                A5 = 3011 0000 0108
     A6 = 100F 0000 0260                A7 = 100F 0000 02A0                A8 = FFFF 8000 0000
     A9 = 100F 0000 03E7                AA = 200C 0000 12D0                AB = FFFF 8000 0000
     AC = 100F 0000 02A0                AD = 3041 0007 8200                AE = FFFF 8000 0000
     AF = 2042 0000 2240
X REGISTERS

     X0 = 2101 100D 0002 5D02 ł     ]   X1 = 0083 0107 0100 02A0           X2 = 0000 0000 0000 0001
     X3 = 0000 FFFF 8000 0000           X4 = 0000 0000 0000 0002           X5 = 0000 0000 0000 000E
     X6 = 0000 0000 0000 AFEC           X7 = 0000 0000 7FFF FFFF           X8 = 0000 0000 0000 0050      P
     X9 = 0000 0C00 FFFF FFFA           XA = C840 6902 E8C7 6A43  ø1   JC  XB = 0000 0000 0000 2420      $
     XC = 0000 0000 0000 0000           XD = 0000 009B 3CF5 DD00     <     XE = 0000 0000 0000 09CA
     XF = 0000 0000 0000 1001
TOS REGISTERS
                                                                  LABEL = EXCHANGE_PACKAGE_AT_JPS
     T1 = 100F 0000 0308                T2 = 2010 0000 05C0                T3 = 3011 0000 08F0
     T4 = 0000 0000 0000                T5 = 0000 0000 0000                T6 = 0000 0000 0000
     T7 = 0000 0C00 0000                T8 = 0000 0000 0000                T9 = 0000 0000 0000
     TA = 0000 0000 0000                TB = B03B 0000 1F68                TC = 0000 0000 0000
     TD = 0000 0000 0000                TE = 0000 0000 0000                TF = 0000 0000 0000
```

```
  14:15:42    DR A=ALL X=ALL T=ALL EXC=JOB FN=SESLOG
  14:15:44    QR DL=SDT_at_JPS
1ABSOLUTE  00683620..0068393F

                                                                                                                    LABEL = SDT_AT_JPS

 00683620  C8110600 00000000   CA130200 0000C000   CA13814A 00000000   CA138568 00000000                                J       h
 00683640  CA23012D 00040060   8A3D0031 00040060   CA3D0656 00040060   8A3B0045 00040060  # -   *  = 1   *  = V   *    E   *
 0068366C  8A23029C 00000000   8A230161 0000C000   00000000 00000000   88206072 00040060  #          # #          _ r   *
 00683680  881F0380 00000000   A81103CF 0000C000   8C1D0280 00000000   8A11C130 00000000                             0
 00683640  8A220292 00000000   8A33040F 00000000   8A110300 00000000   A813C680 00000000  =          3
 006836C0  A81E0580 00000000   8C1D05C0 00000000   88DD0100 00000000   0000C000 0C000C00
 006836E0  00000000 00000000   00000000 00000000   A8228628 00040060   8C2DC420 00040060                 = (   *  -      *
 00683700  A823858C 00040060   A82D02C8 00040060   98880120 00040060   0000C000 00000000  #    *  -    *              *
 00683720  00000000 00000000   00000000 00000000   00000000 00000000   00000000 00000000
 00683740  CA1380C0 00000000   8A330131 0000C000   983D000A 00000C00   8A330331 C0000000            3 1      =          3 1
 00683760  98BB0346 C0000000   984D8288 0000C000   984D8688 00000000   8A238490 00000000    F       M          M          #
 00683780  984D8188 00000000   8A230710 00000000   8A230090 00000000   8A230490 00000000  M         #          8          #
 006837A0  987D8588 00000000   984D8388 0000C000   984D0446 00000000   98448078 00000000  )         M          M F        D x
 006837C0  8A330731 00000000   8C2F07B9 00000000   8ADD0071 00000000   8A3FC481 00000000  3 1       /            q        1
 006837E0  983D07FC 00000000   88BB0281 0000C000   8A8B8170 00000C00   8A88C381 00000000  =                      p
 00683800  8A330181 00000000   8A3F0581 0000C000   983300C6 00000000   8A330456 00000000  3         ?          3          3 V
 00683820  8A330716 C0000000   0A22825A 0000C000   8A220271 00000000   0A11C260 00000000  3         = Z        = q          *
 00683840  00000000 00000000   00000000 0000C000   00000000 00000000   0000C000 00000000
           000007 Duplicate Lines
  14:15:45    DMI S=STA RC=100 AM=RMA EXC=JOB FN=SESLOG
  14:15:47    QR DL=CALL_STACK_for_the_CURRENT_Job

                                                                                  LABEL = CALL_STACK_FOR_THE_CURRENT_JOB

* STACK FRAME 001
* P=1 014 00001FF6
* STACK FRAME 001

* STACK FRAME 002
* P=1 00D 00025E42   DMM$DETACH_FILE                  +0000011A
* STACK FRAME 002

* STACK FRAME 003
* P=2 01A 000214A0   PFM$FILE_SYSTEM_INTERFACES       +00000030
* STACK FRAME 003

* STACK FRAME 004
* P=2 01A 0002ABC6   PFM$R2_REQUEST_PROCESSOR         +0000022E
* STACK FRAME 004

* STACK FRAME 005
* P=2 01A 00014E5A   FMM$LOCAL_NAME_TABLE_MANAGER     +00000412
* STACK FRAME 005
```

21458 PFM$research_permanent_file

```
* STACK FRAME 006
* P=3 01C C0029D90   BAM$RETURN                  +00000110
* STACK FRAME 006

* STACK FRAME 007
* P=3 01D 00003D70   ANM$RETURN                  +000000C8
* STACK FRAME 007

* STACK FRAME 008
* P=3 01C 0000AFFC   AVM$VALIDATION_KERNEL       +0000005C
* STACK FRAME 008

* STACK FRAME 009
* P=3 01C 0000B354   AVM$VALIDATION_KERNEL       +00000334
* STACK FRAME 009

* STACK FRAME 010
* P=B 026 C0060678
* STACK FRAME 010

* STACK FRAME 011
* P=B 026 0006E19A
* STACK FRAME 011

* STACK FRAME 012
* P=B 01D 0005A29C   CLM$PROCESS_COMMANDS        +000000FC
* STACK FRAME 012

* STACK FRAME 013
* P=B 01D CC058CFC   CLM$PROCESS_COMMANDS        +000001E4
* STACK FRAME 013

* STACK FRAME 014
* P=B 01D 00057F7E   CLM$PROCESS_COMMANDS        +0000032E
* STACK FRAME 014

* STACK FRAME 015
* P=B 01D 000575B8   CLM$PROCESS_COMMANDS        +00000558
* STACK FRAME 015

* STACK FRAME 016
* P=B 01D 0002DA4C   CLM$INCLUDE                 +0000007C
* STACK FRAME 016

* STACK FRAME 017
* P=B 01D 0002D078   CLM$INCLUDE                 +000002C0
* STACK FRAME 017

* STACK FRAME 018
* P=B 01D 0002D47A   CLM$INCLUDE                 +00000042
* STACK FRAME 018

* STACK FRAME 019
* P=B 01D 0002CD5C   CLM$INCLUDE                 +00000374
* STACK FRAME 019

* STACK FRAME 020
* P=B 026 0006E0D4
* STACK FRAME 020
```

```
* STACK FRAME 021
* P=B 01D C0074738   PMM$OUTWARD_CALL                    +000000E0
* STACK FRAME 021

* END OF STACK FRAME ENCOUNTERED

  14:15:48     TB RC=50 EXC=JOB DF=DBGFILE FN=SESLOG
  14:18:37     OR DL=CALL_STACK_for_the_CURRENT_Job

* STACK FRAME 001
* SAVE AREA

* P=1 014 00001FF6   VMID=0
* UM=FF77     UCR=C000     MCR=0020

* A0=1 00F 00000308       A1=1 00F 000C0308
* A2=1 00F 0C000280       A3=F 000 00000000
* A4=1 00F 00000268       A5=3 011 00000108
* A6=1 00F 00000268       A7=1 00F 000002A0
* A8=F FFF 80000000       A9=1 00F 000003E7
* AA=2 00C 00001200       AB=F FFF 8000C000
* AC=1 00F 0C0002A0       AD=3 041 C0078200
* AE=F FFF 80000000       AF=2 042 00002240

* X0=21011C00  00025002       X1=00830107  01C002A0
* X2=00000000  00000001       X3=0000FFFF  80C00000
* X4=C0000000  0CCC0C02       X5=00000000  0000000E
* X6=000C0000  0CC0AFEC       X7=00000C00  7FFFFFFF
* X8=00000C00  00C00C50       X9=00000C00  FFFFFFFA
* XA=C84C6902  E8C76A43       XB=00000C00  00002420
* XC=00000000  00000000       XD=00000C98  3CF5D000
* XE=00000000  000009CA       XF=00000000  00001001

* STACK FRAME 002
* SAVE AREA

* P=1 0CD 00025E42   VMID=0  DMM$DETACH_FILE                +0000011A
* UM=FF77

* A0=1 0CF 00000280       A1=1 00F 000C0000
* A2=2 010 00000578       A3=1 00E 000C2888
* A4=2 010 00000558       A5=3 011 C00C0108
* A6=1 00F 00000268

* X2=00000000  00CC0001       X3=C000FFFF  80000000
* X4=00000000  00000002

* STACK FRAME 003
* SAVE AREA

* P=2 01A 000214A0   VMID=0  PFM$FILE_SYSTEM_INTERFACES      +00000030
* UM=FF77

* A0=2 010 0CC00578       A1=2 010 000CC508
* A2=2 010 000004C0       A3=2 018 C00C1D08
* A4=2 010 00000488       A5=2 010 00000558
* A6=3 011 00000108

* X0=00000000  00040060
```

1 014 IED8= CMV$ controller_data_pointer

TOS

Binding section.

SCO

Binding

In SDT

```
* STACK FRAME 004
* SAVE AREA

* P=2 01A C002ABC6   VMID=0  PFM$R2_REQUEST_PROCESSOR          +0000022E
* UM=FF77

* A0=2 010 C0C004C0      A1=2 010 00000250
* A2=2 010 000001F8      A3=2 01B 00002FC0
* A4=2 010 000001A8      A5=2 01C C0000488
* A6=3 011 00000108

* X0=00000000  00040060

* STACK FRAME 005
* SAVE AREA

* P=2 01A 00014E5A   VMID=0  FMM$LOCAL_NAME_TABLE_MANAGER      +00000412
* UM=FF77

* A0=2 010 000001F8      A1=2 010 000C0000
* A2=3 011 00000878      A3=2 01B 000C0BC8
* A4=3 011 C0C00830      A5=3 011 00000108
* A6=2 010 000001A8      A7=3 011 C000071F

* X2=00000000  00CC0001      X3=00000000  00000000

* STACK FRAME 006
* SAVE AREA

* P=3 01C 00029D90   VMID=0  BAM$RETURN                        +00000110
* UM=FF77

* A0=3 011 0C000878      A1=3 011 000007C0
* A2=3 C11 00C00770      A3=3 01B 000C8328
* A4=3 011 00000730      A5=3 011 00000108
* A6=3 011 0000071F      A7=3 011 00000830
* A8=3 011 00000800      A9=3 005 0000814C
* AA=F FFF 80000000

* X2=00000000  00000001      X3=00000000  00000000
* X4=0C00FFFF  80000C00

* STACK FRAME 007
* SAVE AREA

* P=3 01D 00003D70   VMID=0  AMM$RETURN                        +000000C8
* UM=FF77

* A0=3 011 00000770      A1=3 011 000006F8
* A2=3 011 000006C0      A3=3 01B 00013420
* A4=3 011 C0000690      A5=3 011 C0000108
* A6=3 011 CCC0071F      A7=3 011 00000730

* X2=00000000  00000001

* STACK FRAME 008
* SAVE AREA

* P=3 01C C0CCAFFC   VMID=0  AVM$VALIDATION_KERNEL             +0000005C
```

```
* UM-FF77

* A0-3 011 00000EC0      A1-3 011 00000670
* A2-3 011 000005C8      A3-3 018 00007358
* A4-3 011 00000508

* X0-00000000  00020040

* STACK FRAME 009
* SAVE AREA

* P-3 01C 00008354   VMID-0  AVMSVALIDATION_KERNEL          +00000334
* UM-FF77

* A0-3 011 000005C8      A1-3 011 000C0000
* A2-B 03B 00001F10      A3-3 018 C0007358
* A4-B 03B 0C001CE0      A5-B 03B C000C668
* A6-3 011 00000508      A7-3 01C C00C9A28

* X2-00000000  00000000      X3-00000000  000F0010
* X4-00000000  00C002A9      X5-C0000C00  00000000
* X6-00000000  000C0002      X7-C0000C00  00C000C0
* X8-00000000  00C00103      X9-00000000  0000000E
* XA-00000000  00000029      XB-00000C00  00000101
* XC-00000C0C  0000000F      XD-00000C00  00000003

* STACK FRAME 010
* SAVE AREA

* P-B 026 C006D678   VMID-0                                  1768
* UM-FF77

* A0-B 03B 00001F10      A1-B 03B 00001CA0
* A2-B 03B 0C001C58      A3-B 035 C00C0010
* A4-B 03B 00001B10      A5-B 03B C000C668
* A6-B 03A 00000C00      A7-B 026 0008E398
* A8-B 03B 00001CE0

* X0-00000000  00050080

* STACK FRAME 011
* SAVE AREA

* P-B 026 0006E19A   VMID-0                                  *
* UM-FF77

* A0-B 03B 00001C58      A1-B 03B 000C1980
* A2-B 03B 00001938      A3-B 035 C00C0010
* A4-B 03B 000017E8      A5-B 03B 000C0668
* A6-B 03B 00001810

* X2-C00C0000  00000001

* STACK FRAME 012
* SAVE AREA

* P-B 01D 0005A29C   VMID-0  CLI$PROCESS_COMMANDS            +000000FC
* UM-FF77

* A0-B 03B 00001938      A1-B 03B 000C1790
```

```
* A2=B 03B 0C0C1718      A3=B 01B 00018BA0
* A4=B 03B 0C00158C      A5=B 03B C00C0668
* A6=B 03B 000017E8

* X0=00000000  00020060

* STACK FRAME 013
* SAVE AREA

* P=B 01D 00058CFC   VMID=0  CLM$PROCESS_COMMANDS          +000001E4
* UM=FF77

* A0=B 03B 00001718      A1=B 03B 000C14D8
* A2=B 03B 00001480      A3=B 01B 00018BA0
* A4=B 03B 00001308      A5=B 03B 00000668
* A6=B 03B 00001307      A7=B 01D 00056A80
* A8=B 03B 00001580

* X2=00000000  00000001      X3=00000000  00000000
* X4=00000000  000C0002      X5=00000000  00C006E4
* X6=00000000  00C0003F

* STACK FRAME 014
* SAVE AREA

* P=B 01D 00057F7E   VMID=0  CLM$PROCESS_COMMANDS          +0000032E
* UM=FF77

* A0=B 03B 00001480      A1=B 03B 000C11F8
* A2=B 03B 000011A8      A3=B 01B 00018BA0
* A4=B 03B 00001100      A5=B 03B 00000668
* A6=B 03B 00001308      A7=B 01D 00056A80
* A8=F FFF 80000CC0

* X2=0000FFFF  800C0000

* STACK FRAME 015
* SAVE AREA

* P=B 01D 000575B8   VMID=0  CLM$PROCESS_COMMANDS          +00000558
* UM=FF77

* A0=B 03B 000011A8      A1=B 03B 00000F30
* A2=B 03B 00000EA8      A3=B 01B 00018BA0
* A4=B 03B 00000E78      A5=B 03B 00000668
* A6=B 03B 00001100      A7=B 01D 00056A80

* X0=C0000000  00040070

* STACK FRAME 016
* SAVE AREA

* P=B 01D C002DA4C   VMID=0  CLM$INCLUDE                   +0000007C
* UM=FF77

* A0=B 03B 00000EA8      A1=B 03B 000C0E08
* A2=B 03B 00C00D68      A3=B 01B C0016760
* A4=B 03B 00000AF0      A5=B 03B 000C0E78
* A6=B 03B 00C00668      A7=B 01B 00016A38
* A8=F FFF 80000CCC      A9=B 006 C0017E20
```

```
* AA=B 005 0000C20C      AB=B 01B 000C6FC0
* AC=B 011 00000A10      AD=B 006 000029A0
* AE=B 03B 00000AF0

* X0=00000000  0C0300E0

* STACK FRAME 017
* SAVE AREA

* P=B 01D 0002DD78   VMID=0  CLMSINCLUDE                    +000002C0
* UM=FF77

* A0=B 03B 00000D68      A1=B 03B C00C0AF0
* A2=B 03B 00000AB0      A3=B 01B C0016760
* A4=B 03B 00000A80      A5=B 03B 000C0668
* A6=B 03B 00CC0CD0      A7=B 03B 000C10F0
* A8=F FFF 80000CC0      A9=B 006 00017E20
* AA=B 005 00C0C2CC      AB=B 01B 00006FC0
* AC=B 011 00000A10      AD=B 006 000C29A0
* AE=B 03B 00000AF0

* X2=00000000  00CC0001      X3=00000000  00C00000
* X4=00000000  00CC00FF      X5=00000000  00C00032

* STACK FRAME 018
* SAVE AREA

* P=B 01D 0002D47A   VMID=0  CLMSINCLUDE                    +00000042
* UM=FF77

* A0=B 03B 00C00AB0      A1=B C3B 00000A50
* A2=B 03B 00000A00      A3=B 01B 00016760
* A4=B 03B 00000948      A5=B 03B 00000668

* X0=00000C00  00020050

* STACK FRAME 019
* SAVE AREA

* P=B 01D C002CD5C   VMID=0  CLMSINCLUDE                    +00000374
* UM=FF77

* A0=B 03B 0C000A00      A1=B 03B 000C05C8
* A2=B 03B 00000568      A3=B 01B C0016760
* A4=B 03B 00000408      A5=B 03B 00000148
* A6=B 01D 0002C7C8      A7=B 03B 00000948

* X2=00000000  00000000

* STACK FRAME 020
* SAVE AREA

* P=B 026 0006E0D4   VMID=0                                 *
* UM=FF77

* A0=B 03B 00000568      A1=B 03B 000003E0
* A2=B 03B 00000398      A3=B 035 00000010
* A4=B 03B 00000370      A5=B 03B C00C0148
* A6=B 03B 00000408      A7=B 026 C008E586
* A8=B 026 000BE5E0      A9=B 026 000BE398
```

```
* X0=00000000  00040090

* STACK FRAME 021
* SAVE AREA

* P=B 01D 00074738   VMID=0  PMM$OUTWARD_CALL                    +000000E0
* UM=FF77

* A0=B 038 00000398       A1=B 038 00000128
* A2=F FFF 800000C0       A3=B 018 0001AA48
* A4=B 038 C00000C0       A5=B 038 00000370
* A6=B 006 00000350

* X0=00000000  00020060

* END OF STACK FRAME ENCOUNTERED

  14:18:38    DSF RC=50 EXC=JOB DF=DBGFILE S=SAVE FN=SESLOG
  14:21:38    QR DL=RING_1_CALL_STACK_CURRENT_Job
1SEGMENT=  000F   BYTE OFFSET  00000000..000270FF

                                                                  LABEL = RING_1_CALL_STACK_CURRENT_JOB

000C0C00  00001000 0000C0CC   000C100F 00000000   0154100F 00000000   0000C000 00000001
00000020  00002041 C0000000   006C10CF 00000022   30110000 01080000   100F1003 0000D828
000C0040  30111003 00000801   000010CF 000002A0   00C0201C 003A0042   00000000 00000041
00000060  100F0000 00420004   100F10C1 0001C8E8   20100000 09200C40   0100100D 0004886C
000C0080  000010C1 0001DED8   00701001 0009E000   01772010 00000F00   FFFC100E 00005038
000000A0  00002010 00000ED0   000C10CD 000474D8   0000100F 00000058   00002010 00000920
00000CC0  00000000 C0040070   0000100F 00000048   00002010 00000920   00000000 00040070
000CC0E0  FF771003 C000D260   FFFC100F 000000E2   00002010 00000010   00001003 0000D260
000C0100  000010CF 000000FA   000000C0 0000C001   00000000 00000000   0000C000 0000000A
000C0120  00001001 C0073228   010CFFFF 8000C000   00000000 01290124   100FC000 00180000
00000140  100F0000 0122000E   301100CC C108C023   00C0100D 00001008   0000100F 00000150
00000160  007010CF 00000000   FF772010 00000B10   FFFC100E 00000050   00002010 000C0A50
00000180  00003011 00000108   000C100F 00000128   0000100D 00000E28   00001001 0001C8E8
000C01AC  000010CF 0000008A   00000000 0009E000   100F0000 01422108   100F1001 00073228
000C01CC  00001CCD C001F701   10010007 32280000   100F0000 00180124   100FC000 018A0000
000C01E0  100F00C0 01C72108   000C10CD 000205A0   0000100F 000001E8   0040100F 000001A0
000C0200  FF7710CF 00000150   FFFC10CE 00002270   0000100F 00000128   C000FFFF 80000000
00000220  00000000 00400104   100F00C0 00780001   20100000 0C38C8E8   2010C000 0C380000
00000240  00000000 00400104   100F00C0 01280000   0000100D 00019C0E   0000100F 00000250
000C0260  0050100F C0000000   FF7710CF 000002A0   00000000 0000000E   100FC000 00820D50
000C0280  100F0CC0 00470C38   00000CCC 00030050   0000100D 00019718   0000C000 00000001
000C02A0  21C11CCD C0025002   008301C7 0100C2A0   0000100D 00025E42   0000100F 00000280
000002C0  02641CCF C0000000   FF772010 0000C578   FFFC100E 00002888   00002010 00000558
000C02E0  00003011 C0000108   000010CF 00000268   00000000 00000001   0000FFFF 80000000
00000300  00000000 00000002   00000000 0002C007   0000FFFF 80000000   0002C2F7 03761B1A
000C0320  20425800 C0000000   000C10CF 0000030F   0000FFFF 80000000   00001003 00830107
00000340  100F0000 0082FB05   100F0000 0047FB05   0000100D 0001F676   0000100F 00000350
00000360  0263100F 000002F0   FF7710CF 000002A0   FFFC100E 00002108   0000100F 00000268
000C038C  000010CF 00000047   000C10CF 0000C082   00000000 00000000   0000FFFF 80000000
```

```
 14:21:39    DM 0F00000000(16) EXC=JCB RC=2C000 AM=PVA FN=SESLOG
 14:21:43    QR DL=RING_2_CALL_STACK_CURRENT_Job
1SEGMENT=  0010   BYTE OFFSET  000000C0..000270FF
```

13

LABEL = RING_2_CALL_STACK_CURRENT_JOB

000C0000	00000C00 00000000	00000000 0000C000	00003011 00000C20	20102010 000001D8				
C0000020	1F243931 37363639	39503353 30303032	44313938 32303932	33543132 31333330	891 7669 9P3S 0002	D198 2092 3T12 1330		
00000040	00000000 0000301C	00000000 0000C003	44313938 32303932	33543132 31333330		D198 2092 3T12 1330		
000C0060	00002004 00013020	201000C0 0062001F	30110000 07202045	3011C000 07202020				
000C0080	24393137 36363939	50335330 30303244	31393832 30393233	54313231 33333020	$917 6699 P3S0 002D	1982 0923 T121 330		
000000A0	0000201A C0015FC8	0C002010 0000C0A0	00402010 00000000	FF773011 00000E48				
000000C0	FFFC2018 00000BC8	00003011 00000E08	00000000 00020040	00003011 00000BB7				
000C00E0	C00C3011 C0000BBF	00002010 00000068	00000000 00000001	01002010 00000000				
CC0CC1C0	00000000 00000035	00000000 0000C002	20102010 00000248	0100C000 00000002				
00000120	00000000 0000301C	000C0000 00000003	0000301C 01FD0000	2004C000 00D801FD				
00000140	30110000 07200000	30110000 07203020	20100000 014A001F	0000201A 00008DE0				
C0000160	00002010 00000158	24393137 36363939	50335330 30303244	31393832 30393233				
00000180	54313231 33333098	000C2004 00013020	00000000 00000001	0000C000 000C0001	T121 330			
000C01A0	0000301C 00000008	00000000 00000015	30110000 01080002	3011C000 071FD260				
000001C0	30110CC0 0108FB05	0C0C0000 0000FB05	00002004 00000008	24393137 36363939			$917 6699	
0C0C01E0	50335330 30303244	31393832 30393233	54313231 33333000	0000201A 00014E5A	P3S0 002D 1982 0923	T121 330		HZ
C00C0200	00002010 000001F8	02732010 00000000	FF773011 00000878	FFFC2018 000C08C8				
00000220	00003011 00000830	00003011 0000C108	00002010 000001A8	00003011 0000071F				
00000240	00000000 00000001	000000C0 0000C000	50335330 30303244	2010C000 04C00C1F	P3S0 002D			
000C0260	01313231 33333068	01002010 0000C3A0	01002010 00000268	02732010 000000F8	121 330h			
00000280	FF772C10 000000A0	FFFC2018 00000BC8	00002010 00000068	00003011 C0000720				
000C02A0	00002C10 00000228	000C20C4 000000F0	00C00000 00000001	0000C000 00000000				
000002C0	00000000 0C000000	FFFF2020 20202020	20200A00 C0020201	0000C000 00000002				
000C02E0	00000000 0000FB05	000020C3 0000D260	00000000 0000FB05	0000C000 0000FB05				
00000300	000020C4 C00000F0	00000000 0000C000	30110000 08400000	24393137 36363939			$917 6699	
00000320	50335330 3C303244	31393832 30393233	54313231 33333044	31393832 30393233	P3S0 002D 1982 0923	T121 330D 1982 0923		
CCCCC340	54313231 33333044	0000201A 00010020	00002010 00000300	20100000 0378001F	T121 330D			
0C000360	20100C00 034E0001	0001001F 4E414D45	20040001 52400000	002CC000 00010000				
00000380	0008201A 00F02020	4E5E4553 45542020	20202020 20202020	20202020 20202020				
0C0CC3AC	20202C20 20202039	2C040001 5270C000	002C0000 00010000	00082010 C0000250				
000003C0	000202F7 D3761B1A	20425818 00000820	00000000 0000001E	C000C000 00000000				
000C03E0	00000000 C0830107	00000000 00000001	4E564553 45542020	20202020 20202020				
C00C0400	20202C20 20202020	20202020 20202001	01010100 C2000000	03000202 F7D373F0				
00000420	2171C458 01290124	0C2100E8 2042C000	00002042 00000040	20420000 00A00000				
00000440	0C8C00C0 00010000	00A30120 42000002	89002003 0000D260	00002042 00000289				
C0000460	C0002C42 CC002240	2C420000 22400000	01C20000 00010000	005A2003 0000D260				
0C0C048C	52091700 14280387	20422042 00830107	00002042 00002240	20422042 00000000				
000004A0	3C110CC0 01080000	20420C00 2256C001	20100000 03F00438	3011C000 01080000				
000C04C0	0000201A 0002ABC6	00002010 000004C0	00602010 00000250	FF772010 000C01F8				
000C04E0	FFFC2018 00002FC0	00002010 000001A8	00002010 00000488	00003011 C0000108				
000C0500	0000000C 0CC40060	00002010 0000C1A8	00003011 00000108	00002010 000004A0				
0C0C0520	01C02C10 00000528	5209170C 0D230276	5209170D 142A02AA	5209170D 142A02AA				
00000540	04FE0CC0 C0010001	00010001 0001C002	01000000 00000020	20420000 00830107				
CC0C0560	20100CC0 C5270000	201C0000 052F3233	30110000 01081B1A	0000201A 000214A0				
00000580	00002010 0C000578	006C2010 00000508	FF772010 000004C0	FFFC2018 00001DD8				
000C05A0	0CC02010 00000488	00002010 00000558	00003011 00000108	0000C000 00040060				
000005C0	20100000 04360001	20100CC0 041FC000	00C00000 C0000000	24393137 36363939			$917 6699	
000005E0	5033533C 30303244	01393832 30393233	54313231 33333005	00002010 00065C08	P3S0 002D 982 0923	T121 330		
0C000600	00002010 000005F8	02842010 000003E0	FF772010 00000398	FFFC2018 000198F0				
000C0620	00002010 00000358	20100000 05200015	20100000 05200000	0000201A 0002F920				
0C0C0640	00002010 00000638	0C4C2010 00000508	FF772010 000004C0	FFFC2018 00002FC0				

```
  14:21:44    DM 1000000000(16) EXC=JOB RC=2C000 AM=PVA FN=SESLOG
  14:21:47    QR 'DL=RING_3_CALL_STACK_CURRENT_Job
1SEGMENT=  0011   BYTE OFFSET  00000000..000270FF

                                                                  LABEL = RING_3_CALL_STACK_CURRENT_JOB

00000000  30110000 00200000  000000C0 00000000  0000B03B 000020B8  01063011 00000550  0                    j       0    P
00000020  00003011 00000570  30063011 00000590  01863011 00C00580  4556414C 20202020    0      p 0 0          0       EVAL
C00C0040  20202020 20202020  20202020 20202020  20202020 20202054  45535449 4E472020                          T ESTI NG
000C0060  20202C20 20202020  20202020 20202020  20202020 20205252  4556414C 20202020                        RR EVAL
000C0080  202C2C20 20202020  20202020 20202020  20202020 2020208A  54455354 494E4720                          TEST IHG
0C0CC0A0  20202C2C 20202020  2C2C2020 20202020  20202020 202020C0  4E564553 45542020                          NVES ET
000C00C0  20202C20 20202020  20202020 20202020  20202020 20202003  20202020 000F0010
C00C00E0  30113041 C0079FE0  00000000 00000026  00000000 000000FF  3041C000 00007FFF  0 0A                 &         0A
000C0100  FFFF0000 00000017  0100100F 0000C0A0  0000C000 C000201C  0060C000 01290124                                 ) $
000C0120  30110000 01080000  100F0000 001AD828  30110000 0108D060  0000C000 00000064  0                    ( 0       *          d
0C0C0140  00000000 00000042  007C10C3 00000260  25012010 00000004  0042100E 00002088          B  p       * 2       B
0C0C016C  000C10CF 0000005A  000C201C 00000108  0000100F 00000060  00000000 0000000A          Z                  *
000C0180  00000000 C0030070  0C000000 0000FB05  02000010 0E000052  98FFFF80 00000040          P                  R       B
000CC1A0  00000000 2100E801  290124C0 00000000  00000000 32280000  00001000 000480C4    $     ) $                2(
0000D1C0  0000100F 000000C8  026D10CF 00000000  FF772010 00000D58  FFFC100E 00005038          '     m       M     X       PB
000001E0  00002010 00000D10  0C0C2010 00000C86  00001000 00047AD8  0000C000 00000001                          t
00000200  00C000C0 00000000  0C000000 0000C00A  30063011 00000278  3006C001 140C0000                    0 0     x 0
C00C0220  01450000 02580032  300600C1 15600000  03123011 00000200  3011C000 021801C0  E    X Z 0     *     0       0
00000240  300600C1 56840146  000C30C6 00001080  44465336 33362020  20202020 20202020  0    V  F   0       DFS6 36
000C0260  20202020 2C202020  202C2020 20202041  36333658 20202020  20202020 20202020                  A 636X
000C0280  20202020 20202020  20202020 20204143  434F554E 54202020  20202020 20202020                AC COUN T
000002A0  20202020 20202020  20202020 2050524F  4A454354 20202020  20202020 20202020              PRO JECT
000002C0  20202020 20202020  20202020 4E564553  45542020 20202020  20202020 20202020            NVES ET
000002E0  20202020 20202020  20202000 00202020  20202020 20202020  20202020 20202020
000C0300  20202020 20202020  2C2C2020 20202020  20202020 20202020  20202020 20202020
          CCC00E Duplicate Lines
000C03E0  20202C20 20202020  20202020 20000020  20202020 20202020  20202020 20202020
000C0400  20202C20 20202020  20202020 20202020  20202020 20202020  20202020 20202020
          000006 Duplicate Lines
000C04E0  20202020 20202020  20202020 2020200B  0B000100 00000000  0000C000 00020050                              P
0C0CC500  00008026 C0014590  3011C0C0 0550967F  30110000 000F0010  30110000 01080250     &   E  0     P  0         0       P
000C0520  00C000C0 000002A9  30110000 0540C029  30110000 00E20101  B038C000 0660000F          0     a ) 0           j     h
C00C0540  301BCCCC 7638FFFF  80000000 0000C003  24393137 36363939  50335330 30303244  0    v8                 $917 6699 P3S0 002D
CC0C0560  31393832 30393233  54313231 33333005  20202020 20202020  20202020 20202020  1982 0923 T121 330
000C0580  20202C20 20202020  2C2C2020 20202080  44465336 33362020  20202020 20202020                        DFS6 36
000CC5A0  20202020 20202020  20202020 20202001  B03B0000 1DC80000  0206C000 00010000                          j
000005C0  01030000 000C000D  0000301C 00008354  00003011 000005C8  02703011 0C000000                0       T   0        )0
0C0C05E0  FF77B03B C0001F10  FFFC301B 00007358  00008038 00001CE0  00008038 00000668  M j           0     sX    j          j     h
000C0600  00003011 C0000508  000C301C 00009A28  00000000 00000000  0000C000 000F0010    0           Q     (
00000620  0000CC00 CC0002A9  000000C0 0000C000  00000000 00000002  00000000 000000C0                                 )
0C0C0640  0C0000C0 CC0001C3  000000C0 0000C00E  00000000 00000029  0000C000 00000101                               0       P
0C000660  00000CC0 0000000F  0C000000 0000C003  00000000 C00C000D  3011C000 06980050                               0       P
00000680  00000000 00000001  B0383011 000006A0  30110000 06A00010  30110000 01080770                    )0         0
C00C06A0  24393137 36363939  50335330 30303244  31393832 30393233  54313231 33333000  $917 6699 P3S0 002D 1982 0923 T121 330
000006C0  0000301C CC00AFFC  00003011 000006C0  00403011 00000670  FF773011 000005C8    0          0        30      p  u0
```

AMM$Return

```
 0C0C06E0  FFFC3018 00007358   0C003011 00000508   00000C00 C0020040   00003011 0C000690    0     aX   0                a   0
 00000700  00000000 00020060   30113011 00000250   00003011 00000750   0000C000 00000001          0 0     P   0      P
 00000720  00000000 0000301C   000C301C 06CF2020   30110000 07500001   0000C000 00000003          0   0         0      P
 000C0740  30110000 071F2020   30110000 01062020   24393137 36363939   50335330 30303244  0          0             $917 6699 P350 002D
 C00C076C  31393832 30393233   54313231 33333044   00003010 00003D70   00003011 00000770  1982 0923 T121 330D    0    =p   0     P
 0C000780  02723011 000006F8   FF773011 000006C0   FFFC3018 00013420   00003011 00000690  r0         H0          0    4    0
 00000740  00003011 C0000108   000C3011 0000071F   00003011 00000730   0000C000 00000001   0          0           0    0
 000C07C0  00003011 000006F8   000C0000 00090070   00703011 00000670   FF773011 00000858   0                    p  p0     p  H0     X
 C00C07E0  1F243931 37363639   39503353 30303032   44313938 32303932   33543132 31333330  $91 7669 9P35 0002 D198 2092 3T12 1330
 00000800  24393137 36363939   50335330 30303244   31393832 30393233   54313231 33333000  $917 6699 P350 002D 1982 0923 T121 330
 C0000820  20203011 00000800   00000000 00000064   30110000 0800001F   0000C000 00000003   0                   d 0
 00000840  30110000 071F3070   30110000 01080015   00003010 00006110   24393137 36363939  0       0p 0           0    a  $917 6699
1SEGMENT=  0011    BYTE OFFSET  00000000..000270FF                        SIM180 V6.8 LEV 132      09/23/82    14:21:49      PAGE     2
                                                                                                LABEL = RING_3_CALL_STACK_CURRENT_JOB
 000000860 50335330 30303244   31393832 30393233   54313231 33333058   0000301C 00029D90  P350 002D 1982 0923 T121 330X   0
 C00C0880  00003011 00000878   02A43011 000007C0   FF773011 00000770   FFFC3018 00008328   0     H   0          H0     p   0     (
 000C08A0  00003011 C0000730   00003011 00000108   00003011 0000071F   00003011 C0000830   0     0   0           0          0     0
 000C08C0  00003011 00000800   00003005 0000814C   766CFFFF 80000000   0000C000 00000001   0           0     L vi
 000008E0  00000000 00000000   0000FFFF 8000C000   01700000 00000000   0000C000 00000000                     P
 00000900  000000C0 00000000   00C00000 00000000   00000000 00000000   0000301C 00027EFA                                 0   -
 000C0920  00003011 00000918   00703011 000008A0   FF773011 00000850   FFFC3018 00007FB8   0          p0          H0     P   0
 000C0940  00003011 00000828   0000301C 00027B70   00003011 000008C8   00003011 00000720   0     (    0     (p    0          0
 000C0960  000000C0 C0040070   000000C0 00000003   30110000 00900000   3011C000 06A00601         p                0          0
```

$ Unique

TOJ 3= 3 11 8BF

BAM$Return.

1021

16

LABEL = JOB_FIXED_FOR_CURRENT_JOB

AAKT SBIG D jobname

X E VAL
TE STIN G

A CCOU NT
PR OJEC T

XCB

00000000	00000000 00000000	41414B54 24424947	44202020 20202020	20202020 20202020	
000C0020	20202020 20202020	20202020 00250545	56414C20 20202020	20202020 20202020	
000C0040	20202C20 20202020	2C2C2020 20205445	5354494E 47202020	20202020 20202020	
000C0C60	20202C20 2C202020	20202020 20001506	000DFFFF FFFF0000	1003C000 25280000	
000C0080	00000C41 43434F55	4E542020 20202020	20202020 20202020	20202020 2C202020	
000000A0	20205052 4F4A4543	542C2020 20202020	20202020 20202020	20202020 20202020	
0C0C00C0	20FFFF80 C0000000	00000000 0000C000	00000000 00000000	0000C000 00000000	
000C00E0	0000000C C0000000	00000000 0000C000	00000000 00000000	0000C000 00000000	
C000C100	0000301D 000A8276	000C3011 00000400	00023011 0000C4D0	FF773011 0C000470	
00000120	FFFCFCCC C0000000	0C0C3011 00000440	00203011 00C00440	00003005 C00025F8	
000C0140	0000301C C0083AD8	00003011 0000C328	0000FFFF 80000000	7FAD8006 000004F8	
C0000160	99728C05 0000D0A0	00003011 00000328	01008006 00009D7F	00003011 00000480	
CC0CC18C	00633011 00000470	1B202020 202020FF	FFFFFFFF FF202020	0000C000 CC0C0001	
000C01A0	00000C00 0000000D	00000CC0 0000C004	00000000 000003E8	0000C000 00000002	
0C0001C0	00C00C00 C0000002	0C000000 00000002	00000000 0000000E	0000C000 00000000	
000001E0	00000000 00000004	00000000 0000C019	0000009A 5B5A9DDC	0000C000 000007DF	
00000200	03E8009A C0001001	0C0C00C0 0000C000	006C1FFF 7FFFFFFF	68DD100E 000C0CF8	
000C0220	0000FFFF 80000000	000F10CF 00000000	00002010 00000000	00003011 00000400	
000C0240	00000000 00000000	00000000 00000000	00000000 00000000	0000C000 00000000	
00000260	0CCC0C00 00000000	000C00C0 0000C000	00000000 00000000	00008028 00002A00	
C000C280	00000CCC C0000000	000C00C0 00000000	00000000 00000000	0000C000 00000000	
	000001 Duplicate Lines				
000002C0	00000000 C0000000	00000000 00000000	00000001 00000015	06000000 0400FFFF	
000002E0	80000000 20040000	1E400000 00011003	00000800 00000320	0000CC00 00000008	
00000300	10030000 0BF00000	0E100000 0000C000	0024FFFF FFFFFFFF	0000C000 005376AC	
C000C320	00000CC0 00108A04	00000000 00000000	00000000 00000000	0000A9AF 00000000	
00000340	00000000 C0000087	0C0000C0 00000000	00000000 00000000	0000C000 00000000	
00000360	00000000 00000000	00000000 00000000	00000000 00000000	00000000 00000000	
	000002 Duplicate Lines				
000003C0	00000000 C0000000	00000000 0000C000	00220604 00000003	30110000 02001506	
000003E0	30110CCC 02580410	30110000 02582020	30180000 00000000	0000C000 00000000	
00000400	00000000 00000000	00000000 00000000	00000000 00000000	0000C000 00000000	
	000003 Duplicate Lines				
000C04B0	00000000 00000000	0C000024 4A4F424D	4E545220 20202020	20202020 20202020	
000004A0	20202020 20202020	20200000 00000000	00000000 00000000	0000C000 00000000	
C00C04C0	00000000 00000000	00000000 00000000	00000000 C0000000	0000000b 00000000	
	000031 Duplicate Lines				
000008C0	00000000 00000000	00000000 0000C000	C8110600 00000000	CA13C200 000b0000	
000008E0	CA13814A 00000000	CA138568 0000C000	CA23012D 00040060	8A3DC626 00040060	
000C09D0	CA3D0656 00040060	8ABB0126 00040060	8A230290 00000000	8A230161 00080000	
CC0C0920	00000000 C0000000	88208072 00040060	881F0380 00000000	A81103CF 00000000	
000C0940	8C1D0280 C0000000	8A1105C6 0000C000	8A220205 00000000	8A33C3C6 00000000	
000C0960	8A110300 00000000	A0130680 00000000	A81E0580 00000000	8C1DC500 0C000000	
00000980	88DD0100 00000000	00000000 0000C000	00000000 0000CC00	0000C000 00000000	
0C0C09A0	A8228628 00040060	8C2D0420 0004C060	A823858C 00040060	A82DC2C8 00040060	
000C09C0	98BB0120 00040060	0C000000 0000C000	00000000 00000000	0000C000 00000000	
000C09E0	00000000 00000000	0C0C00C0 00000000	CA138000 C0C00C00	8A33C526 00000000	
00000A00	8A330326 00000000	8C2F02A6 00000000	BABB03CE 00000000	8A3300A6 C0000000	
000C0A20	8A3F04A6 00000000	8A23849C 00000000	88330216 00000000	8A230710 00000000	
C00C0A40	8A230090 00000000	8A230490 0000C000	8A330316 00000000	8A330716 00000000	
0C0C0A60	8A330096 00000000	8A330716 00000000	8A330496 00000000	8A330316 00000000	
000C0A80	8A330196 00000000	8A330316 0000C000	88330596 00000000	08330216 00000000	
00000AA0	8A330C56 C0000000	8A330316 0000C000	8A330456 00000000	8A33C716 00000000	
000C0AC0	83300C6 00000000	8ADD0256 0000C000	98BB05AA 00000000	0A11C260 00000000	
00000AE0	0A22008E C0000000	0A11C260 0000C000	00000000 00000000	0000C000 C0000000	

$ JOBM NTR