A37

Fast Path Data Entry Database Advanced Topics

Bill Stillwell, Dallas Systems Center
stillwel@us.ibm.com

Miami Beach, FL October 22-25, 2001

RETURN TO INDEX

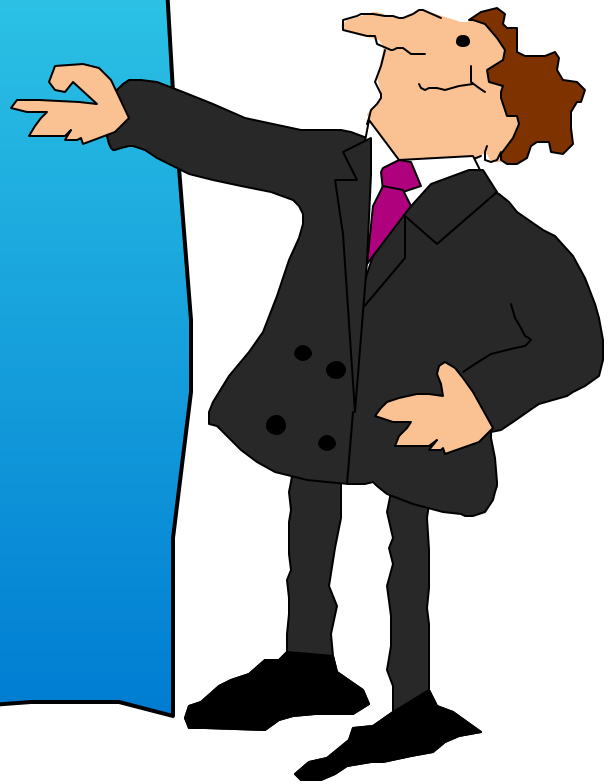
Agenda

Audience

- ★ Users familiar with DEDB Fundamentals

Topics

- ★ High Speed Sequential Processing
- ★ Data Sharing
- ★ Sequential Dependent Processing
- ★ Virtual Storage Option



High Speed Sequential Processing

Benefits

- High performance sequential or skip sequential processing
- Sequential BMP
- Online Reorganization
  - Uses same techniques

Implementation

- Allocates private buffer sets
- Performs ***chained read*** of complete UOW into buffer set
- Optionally performs ***anticipatory read*** of next UOW into another buffer set
- Optionally creates ***asynchronous image copy (ASIC)***

HSSP Characteristics

Non-message driven BMPs only

Processing option (H)

- Set in PCB statement of PSB

```
PCB1  PCB  TYPE=DB,DBDNAME=ACCTDB,PROCOPT=Hx
```

- Can be disabled at execution time in BMP JCL

```
//DFSCTL    DD    *

 SETO       DB=ACCTDB,PCB=PCB1,NOPROCH
```

HSSP Characteristics ...

Requests UOW locking

- Lock full UOW instead of each CI
  - Less overhead

Uses area private buffers

- Allocated to BMP in ECSA
  - Page fixed when allocated
- Contain CIs from UOW being processed
- Used in addition to NBA/OBA buffers
  - IOVF and SDEP still use NBA/OBA

HSSP Characteristics ...

Performance improvements due to

- Chained reads
  - Reads full UOW (BASE + DOVF)
- Anticipatory reads
  - Reads next UOW while current UOW being processed
- IOVF and SDEPs read as required

Asynchronous image copy (ASIC)

- Optional
- Creates fuzzy image copy as application proceeds through Area
  - Saves having to read again to take IC

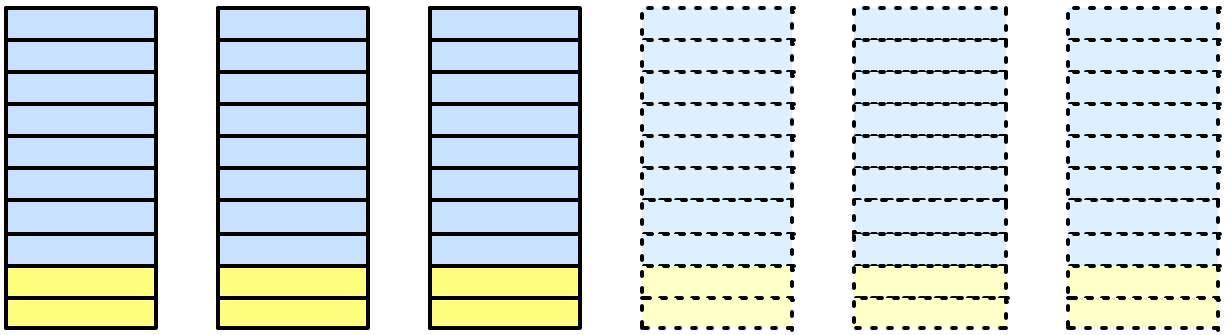
HSSP Buffer Allocation

Private buffers are allocated in sets

- Buffer set contains enough buffers to hold one UOW
  - **UOW=(10,2),SIZE=4096** requires buffer sets with 10 4K buffers
- Buffer sets are allocated in ECSA but are dedicated to BMP

Initial allocation is 3 buffer sets

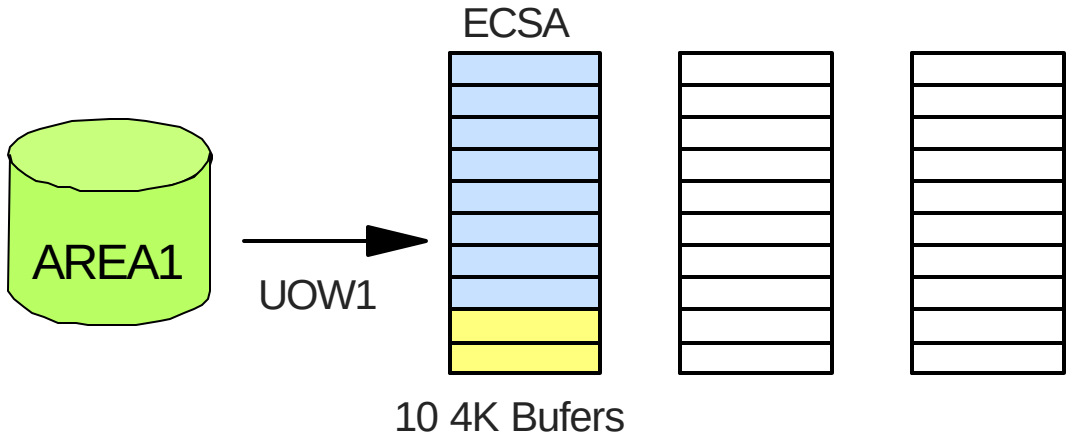
- Allocated buffers are page fixed
- Dynamically increased to 6 buffer sets if needed



HSSP Read Processing

First request for CI

- Get **EXCLUSIVE** UOW lock on first UOW
  - Get exclusive lock even if PROCOPT=G
- Chain read all CIs in UOW into one buffer set
  - Read all CIs in UOW with one I/O



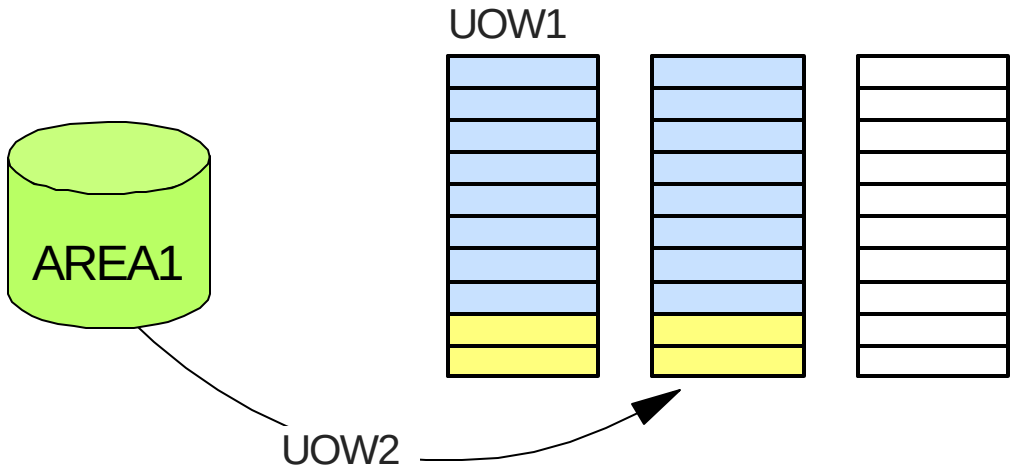

```
DBD     DBDNAME=ACCTDB,ACCESS=DEDB,....

AREA    DD1=AREA1,SIZE=4096,UOW=(10,2),ROOT=(1000,100)
```

HSSP Read Ahead

While program is processing first UOW

- Get conditional lock on next UOW
- If conditional lock fails, do not proceed with Read Ahead
  - We'll read it later



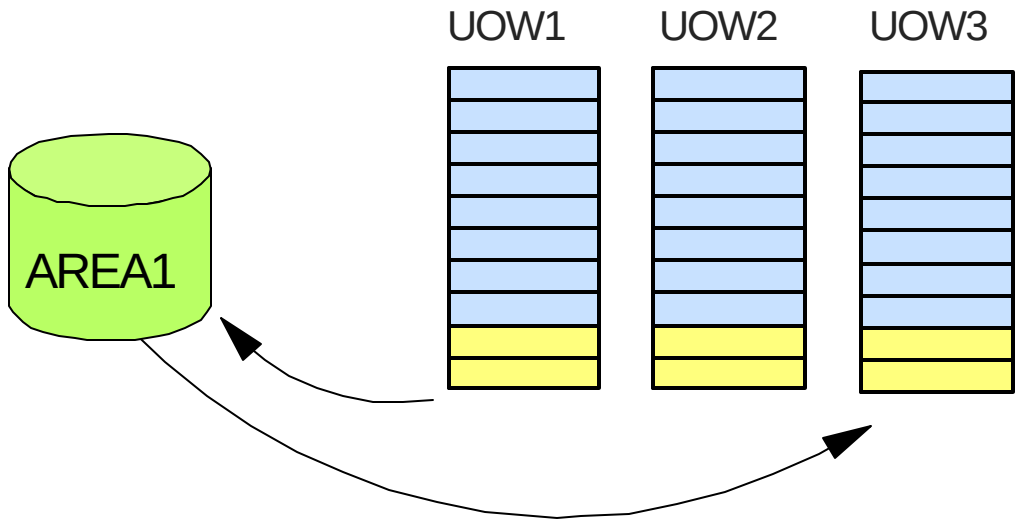
Get Next Processing

Program must proceed forward in area

- Backward reference causes **ROLB** and **FY** status code
  - To go backward use another PCB

During normal (non-sync point) processing

- One buffer set used for current UOW
- One buffer set used for read ahead
- One buffer set used for previous UOW output thread processing



Get Next Processing ...

When program crosses UOW boundary

- If call requires a CI in another UOW
  - Past path returns **GC** status code to program
  - Database position remembered
- Program must issue **CHKP** call before another call to that PCB
  - Else, internal **ROLB** and **FR** status code
  - OK to issue calls to other PCBs
- When CHKP completes
  - Repeat (GN) call
  - Database position not lost at CHKP

HSSP Sync Point

At sync point time

- Current buffer set passed to output thread processing

When sync point processing completes

- Buffer set used for read ahead becomes current UOW
- New available buffer set used for Read Ahead

When output thread completes

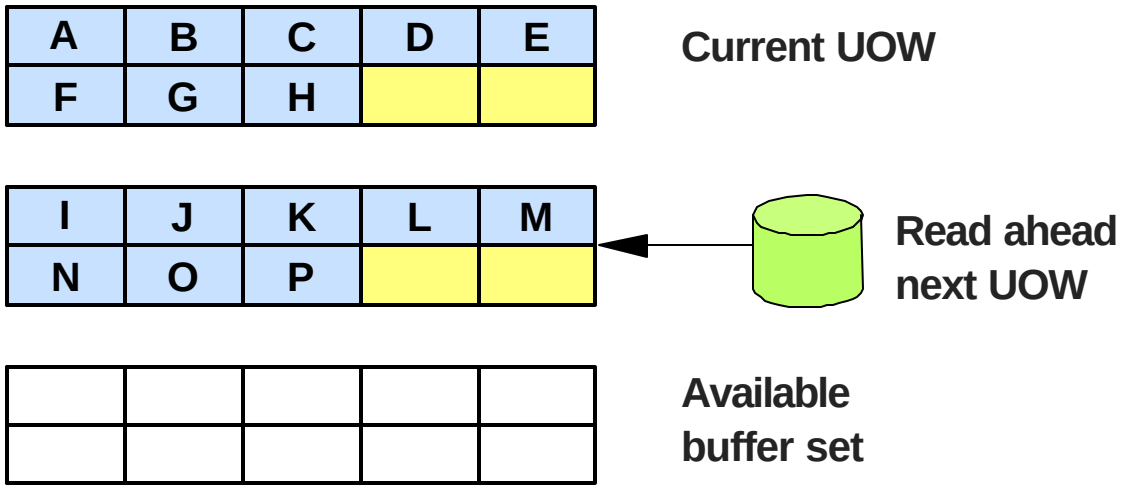
- UOW lock released
- Buffer set freed

HSSP Example

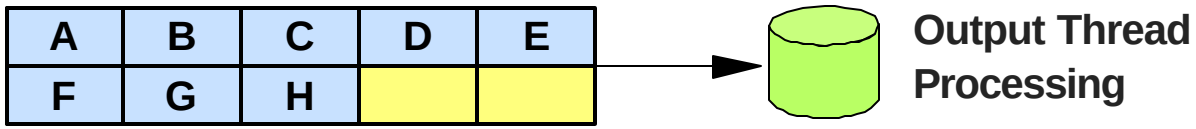
GN LOOP
- A (in buffers)
- B (in buffers)
- C (in buffers)
- D E F G (in buffers)
- H (in buffers)
- I (GC status)

CHKP
- SYNC POINT PROCESSING
- Don't wait for any I/O

GN LOOP
- I (in buffers)



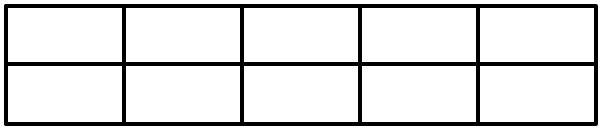
Asynchronous Process
- When logging complete



HSSP Example ...

GN LOOP
- I (in buffers)
- J (in buffers)
- K (in buffers)
- L M N O (in buffers)
- P (in buffers)
- Q (GC status)

CHKP
- SYNC POINT PROCESSING

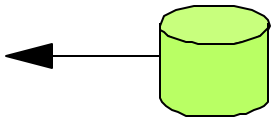


Buffer set freed when output thread processing completes.

I	J	K	L	M
N	O	P		

UOW in buffers from previous read ahead.

Q	R	S	T	U
V	W	X		



Read ahead next UOW

Process continues until program terminates.

HSSP Summary

Improves performance of sequential BMPs

- Chained reads and read ahead
- *Possible* to process entire database with *no synchronous I/O*

Image copy option

- Standard - tape or disk

High Speed Online Reorganization

Online utility

- Reorganizes Area without taking Area offline
- Concurrent update processing of Area allowed

Uses same technique as HSSP

- Lock UOW
- Read UOW
- Reorganize UOW
- Write UOW
- Release lock

DEDB Data Sharing Considerations

Benefits

- Same as for any database
  - Access by multiple IMSs
- No added restrictions for DEDBs

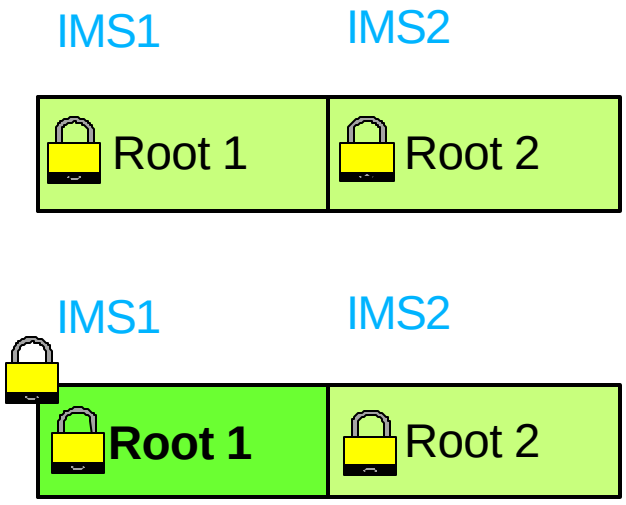
Implementation differences

- Special requirements for Shared SDEPs
  - Need to provide chronological sequence
- No cache structure unless VSO
  - VSO areas stored in *Store-in Cache* structure

Full Function vs DEDB

Full Function databases

- Require cache structure in coupling facility
- Why?
  - Because FF locking is at record level
    - Multiple IMSs may have same block in buffers but want to update different records
  - Updates acquire block lock
    - Only one IMS can update at a time
- Cache structure used for
  - Registering interest in CI or block
  - Buffer invalidation
  - OSAM data caching (optional)
- Cache structure accessed
  - Before every read (register interest)
  - After every write (buffer invalidation)

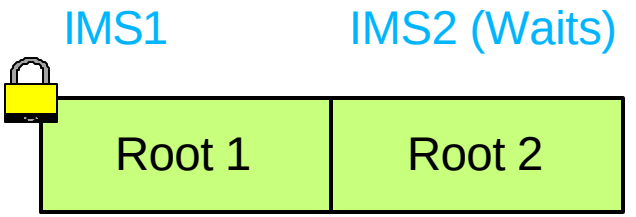


When IMS1 updates block, IMS2's buffer is invalidated.

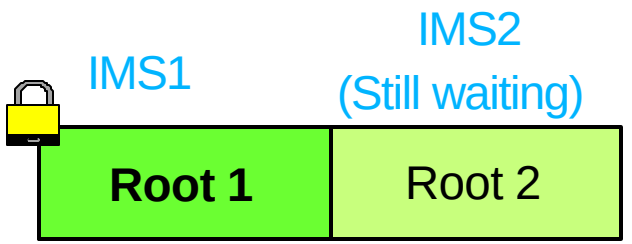
Full Function vs DEDB ...

DEDBs

- No cache structure needed
  - Except for shared VSO
- Why?
  - Fast path locks at CI level
  - Can't have same CI in multiple IMS buffers
- Shared VSO
  - Uses Store-in Cache structure
    - Data read from and written to cache
    - DASD updated once per system checkpoint
- Shared SDEPs
  - Uses timestamps to maintain chronological sequence

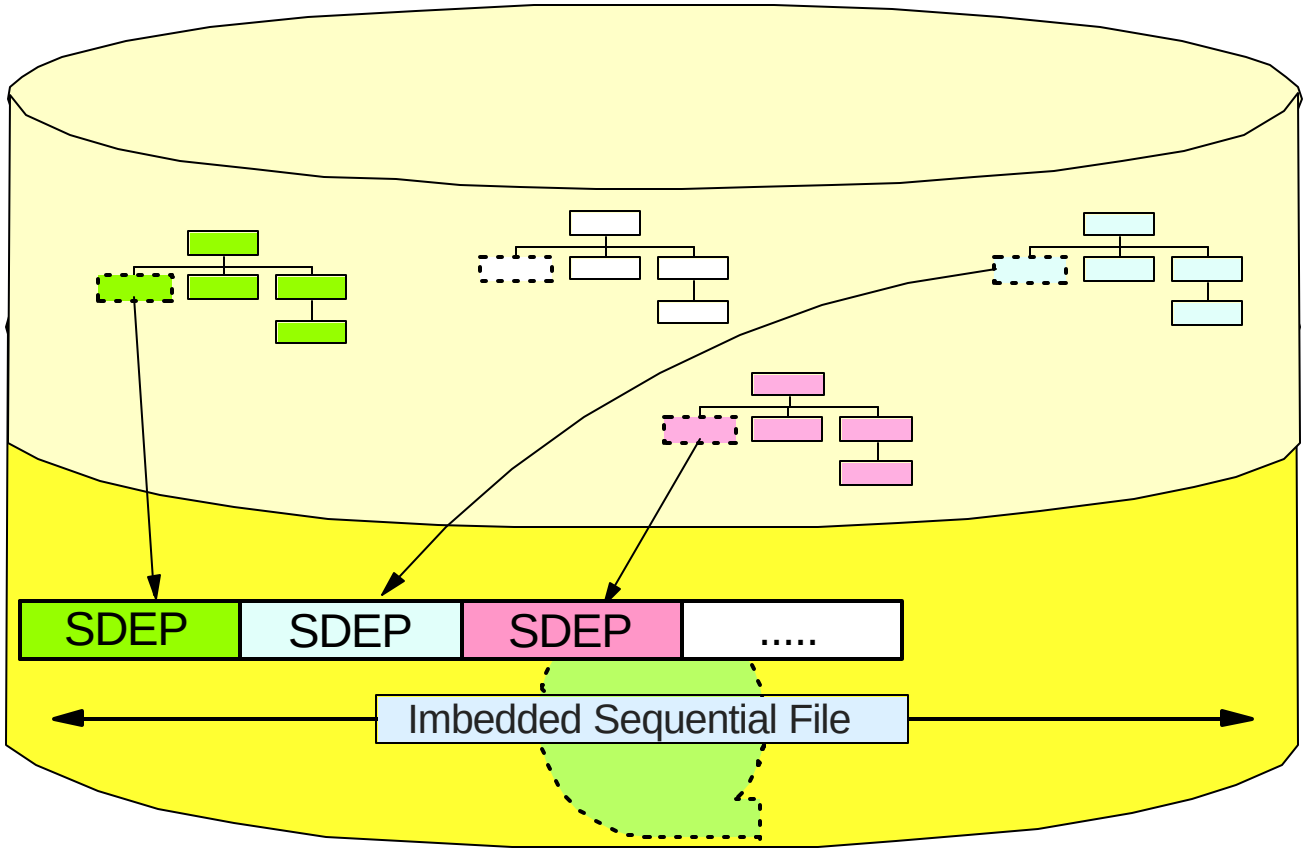


Buffer cannot be in IMS2's buffer pool.



When IMS1 updates block, no need to invalidate IMS2's buffer.

Sequential Dependents - A Reminder

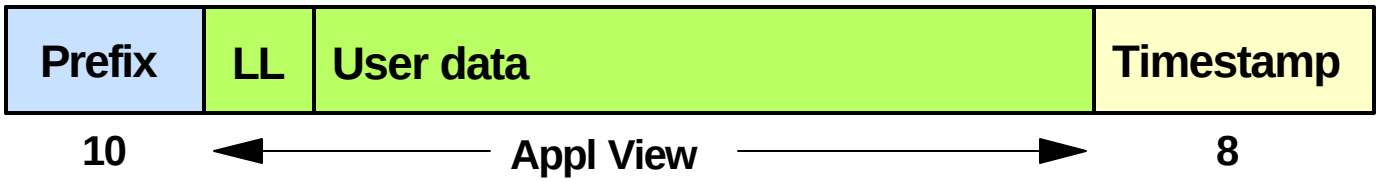


SDEPs are stored in chronological sequence at end of Area Data Set, regardless of what root they were inserted under.

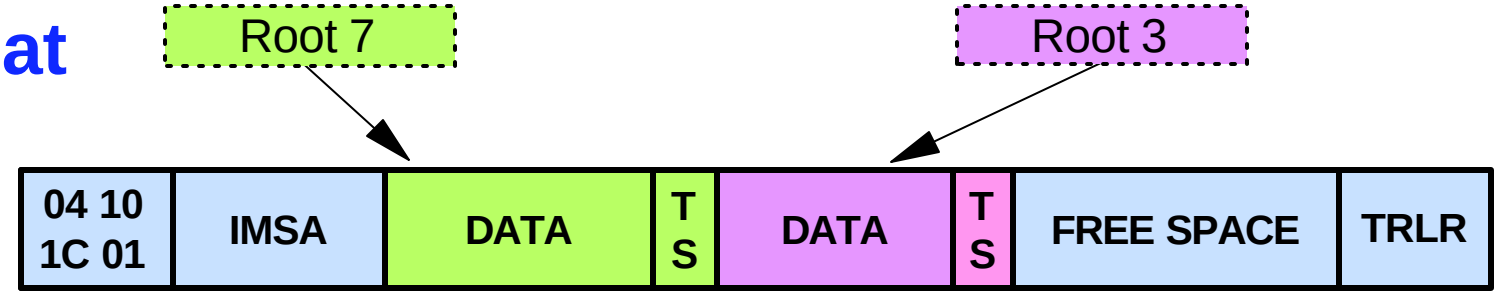
SDEP Formats

Segment format changed in V6 to support BLDS

- ***Timestamp (8-byte STCK value) added to end of segment***
  - Timestamp is ***application sync point time***
  - All segments inserted during same sync interval have same T/S
  - Timestamps are transparent to applications (added/removed by IMS)



CI format

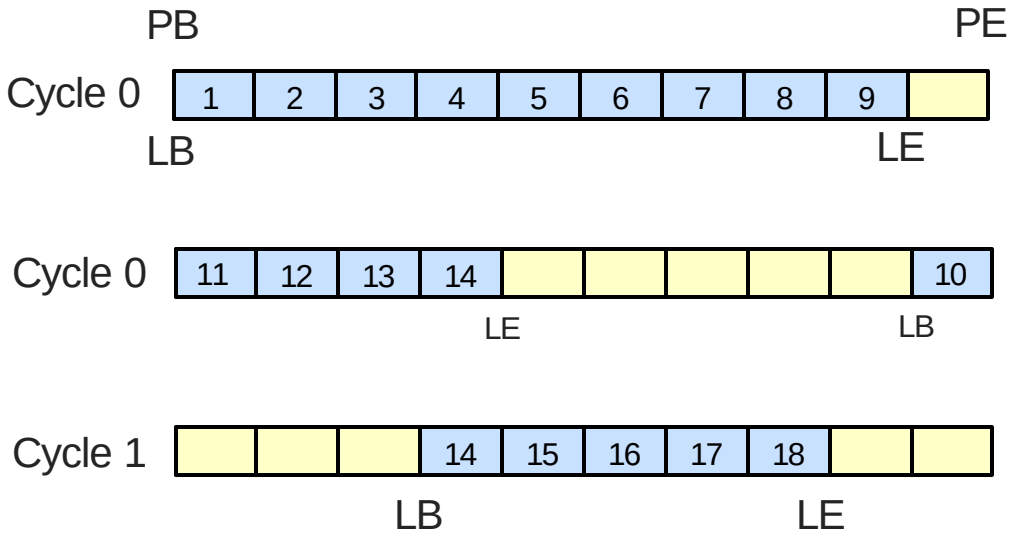


Prefix (4) — **IMSID (8)** — **Segment Data and Free Space with 8-byte timestamps (CISIZE-23)** — **Trailer (11)**

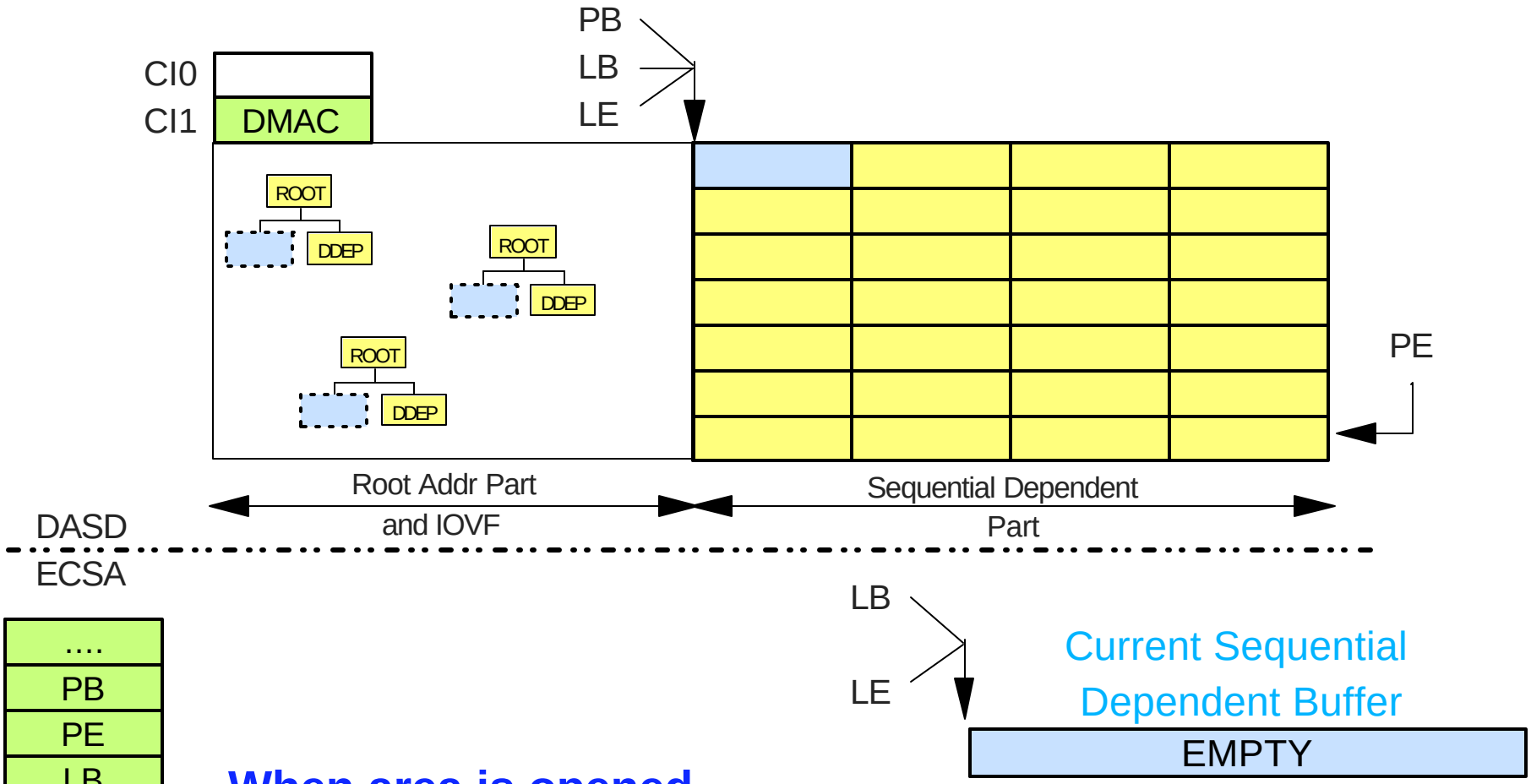
Sequential Dependent Management

SDEP fields in DMAC

- Physical beginning (PB)
  - First SDEP CI in Area
- Physical end (PE)
  - End of data set
- Logical beginning (LB)
  - Oldest active SDEP in Area
- Logical end (LE)
  - Where next SDEP is to be inserted
- Cycle count (CYCLE)
  - Number of times SDEP part reused



Sequential Dependent Management



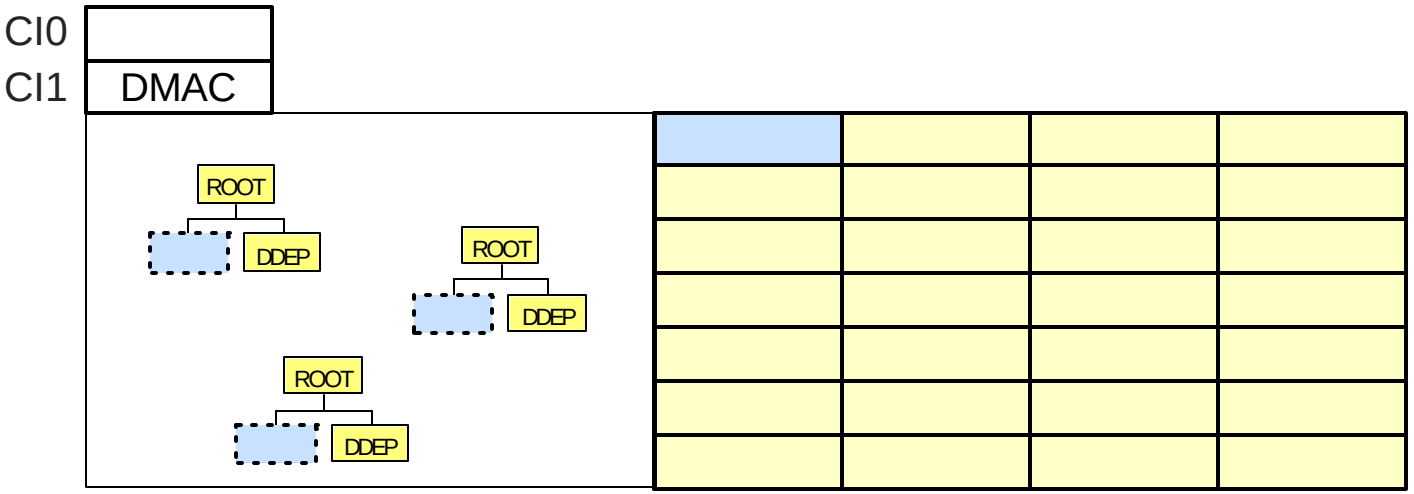
When area is opened

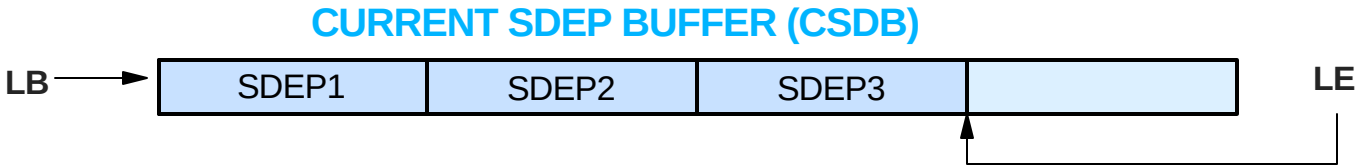
- ✓ DMAC read into buffer and then copied to ECSA

When first SDEP inserted

- ✓ SDEP buffers allocated in global buffer pool
- ✓ One of these is called the CSDB

Sequential Dependent Management ...





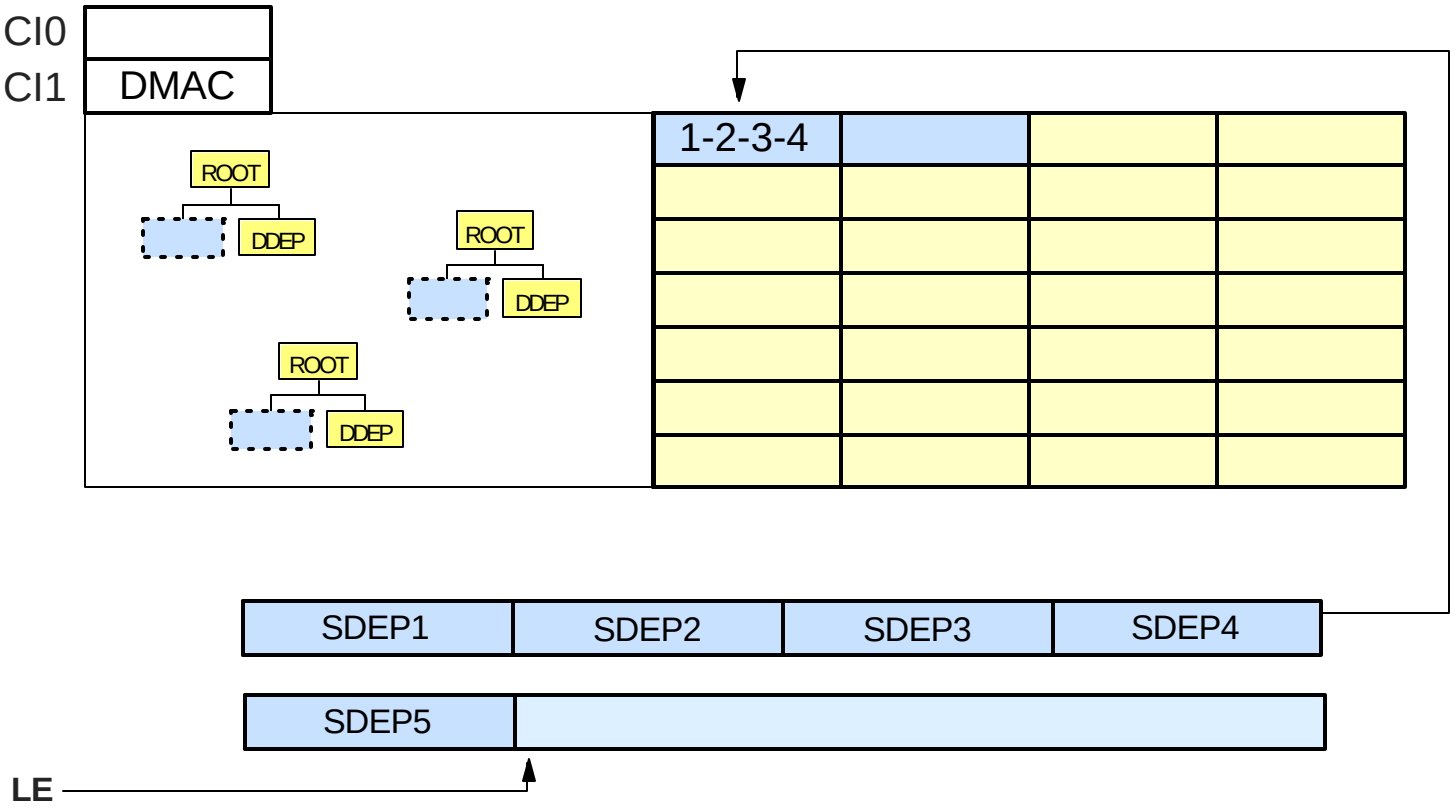
When SDEP inserts are ***committed***

- ✓ Move SDEP to the next available space in the CSDB
- ✓ Increment LE pointer

Do not write CSDB to DASD until full

- ✓ Many inserts per I/O

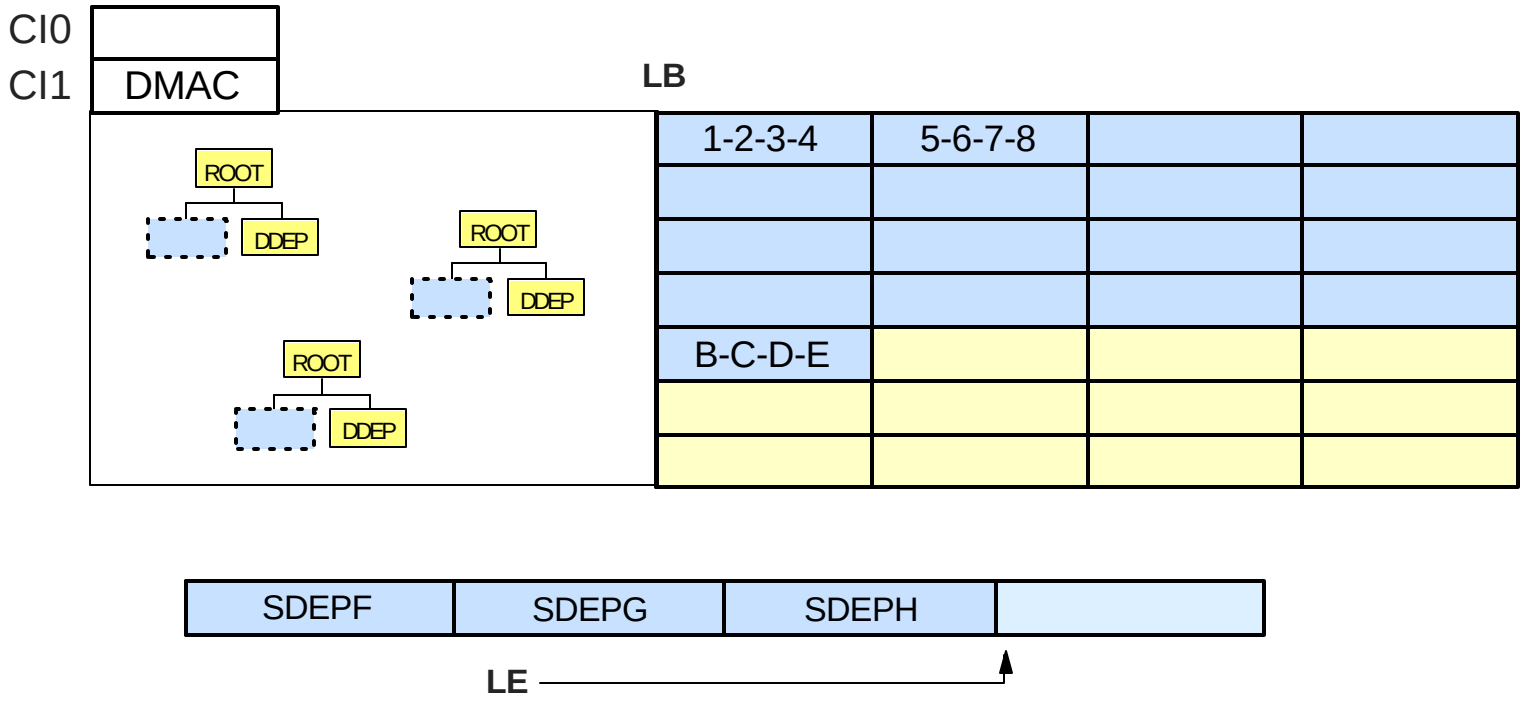
Sequential Dependent Management ...



When CSDB is full (i.e. no room for SDEP5)

- ✓ Allocate a new CSDB
- ✓ Write full CSDB to DASD

Sequential Dependent Management ...

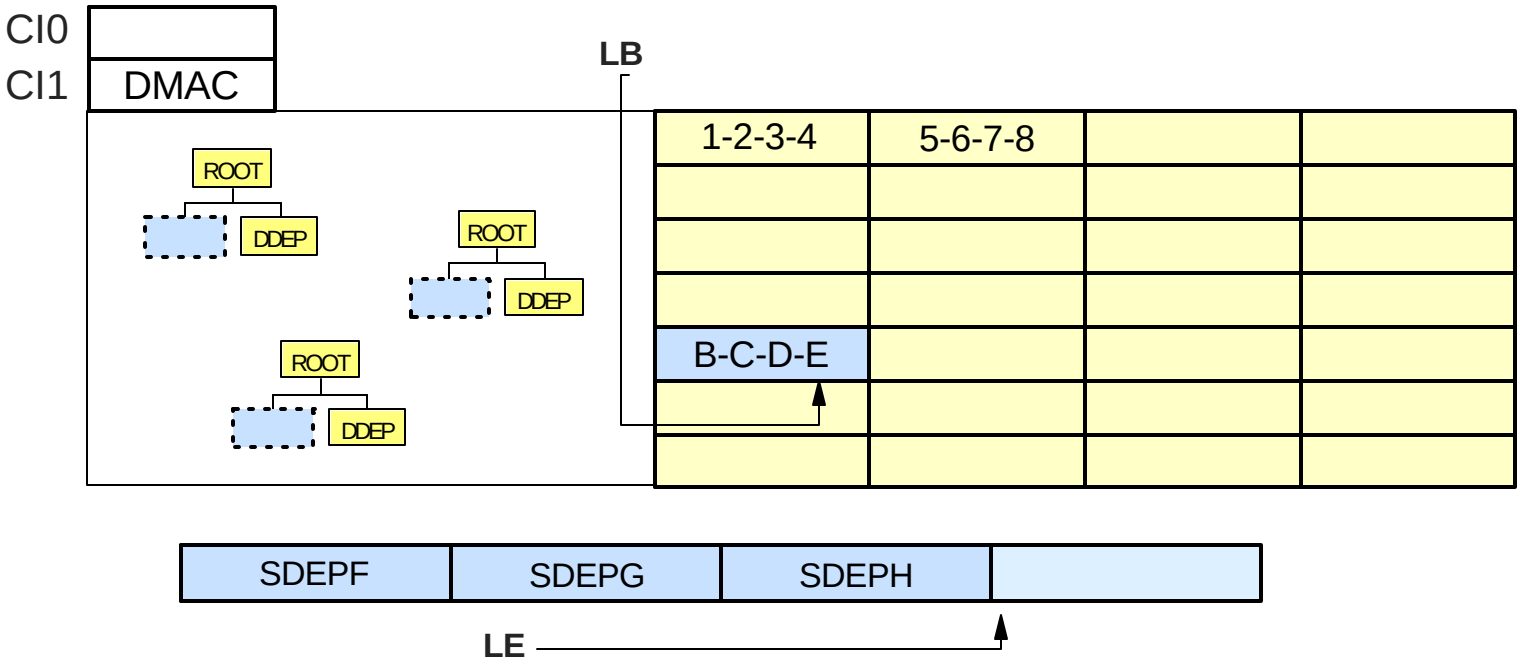


SDEP Scan Utility can read any sequence of segments between LB and LE

Examples

- ✓ From SDEP1 to SDEPH
- ✓ From SDEP7 to SDEPD

Sequential Dependent Management ...

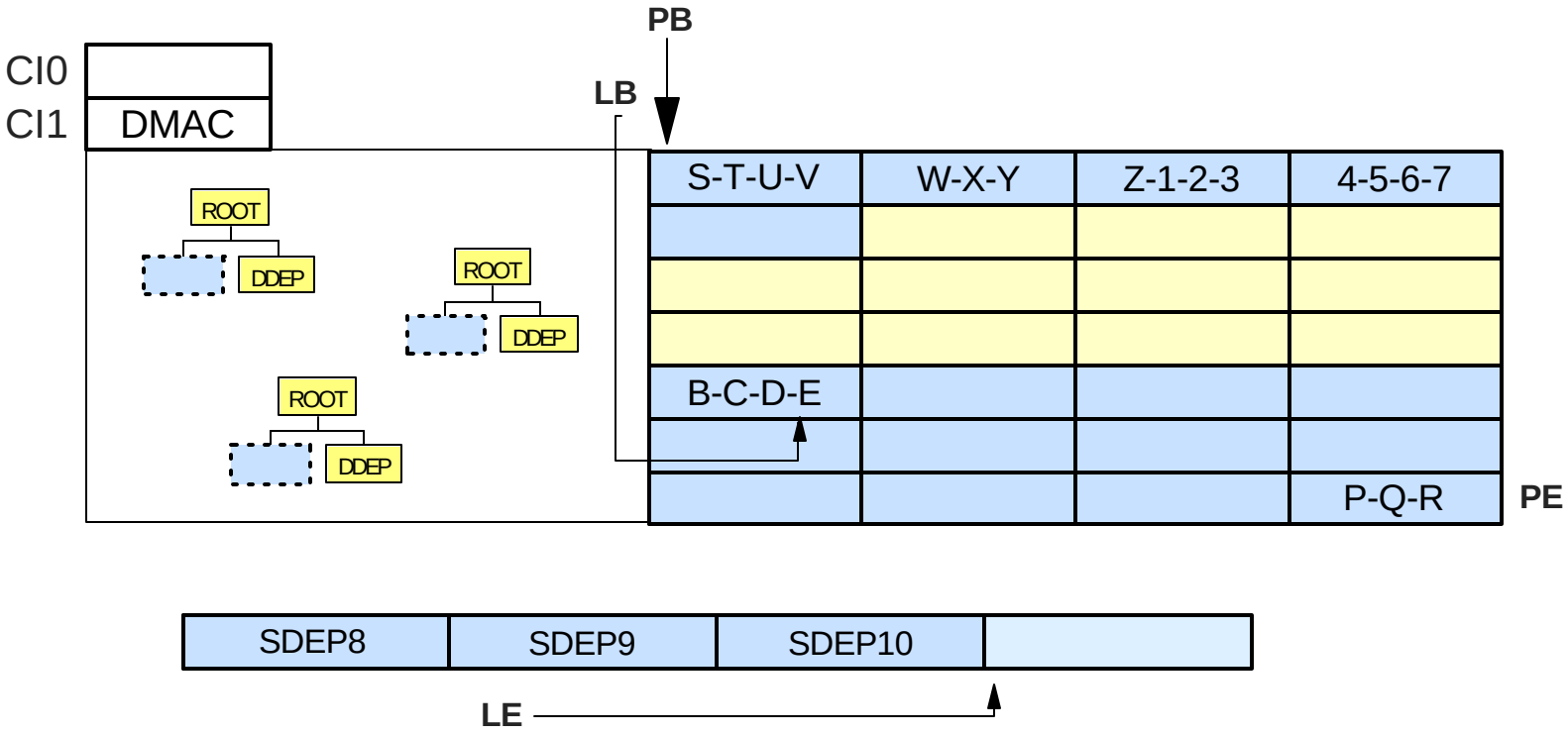


SDEP Delete Utility can delete from LB forward by modifying Logical Beginning pointer

- ✓ Example: From beginning (SDEP1) to SDEPD

Note that SDEP1 thru SDEPD still *physically exist* in AREA, but are *not accessible*

Sequential Dependent Management ...



When physical end of Area Data Set is reached

- ✓ SDEPs wrap to physical beginning
  - CYCLE count incremented by one
- ✓ If space at beginning has not been deleted
  - Program gets area full condition (FS status code)
  - MTO gets DFS2765W message

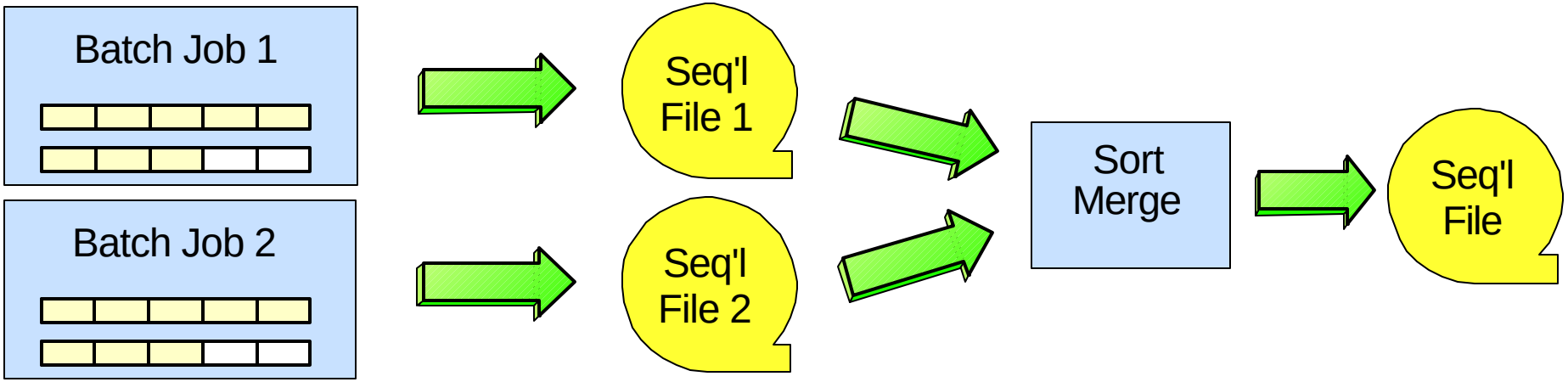
Sharing Sequential Dependents

Think of the SDEP part of a DEDB as a ***sequential file embedded within the Area Data Set***

- A batch job would write records to the file until the buffer is full
  - QSAM would write the full buffer to the output file (tape or DASD)
- The batch job just continues filling the next buffer
  - Doesn't wait for the previous buffer to be written

Imagine two batch jobs writing to same seq'l file

- Can't do it - need to merge them



Data Sharing Considerations

SDEP CIs allocated to sharing IMSs on CI basis

- Allocated CI filled by just one IMS
- Individual CIs are not shared for insert purposes
- Individual CIs are shared for read or utility purposes

Sequential Dependent Segments have timestamps

- Added to the end of the segment by Fast Path
- All segments inserted in same sync interval have same timestamp
- Removed before returning to application Get call and on Scan output
- Scan and Delete utilities *start/stop on timestamp boundary*

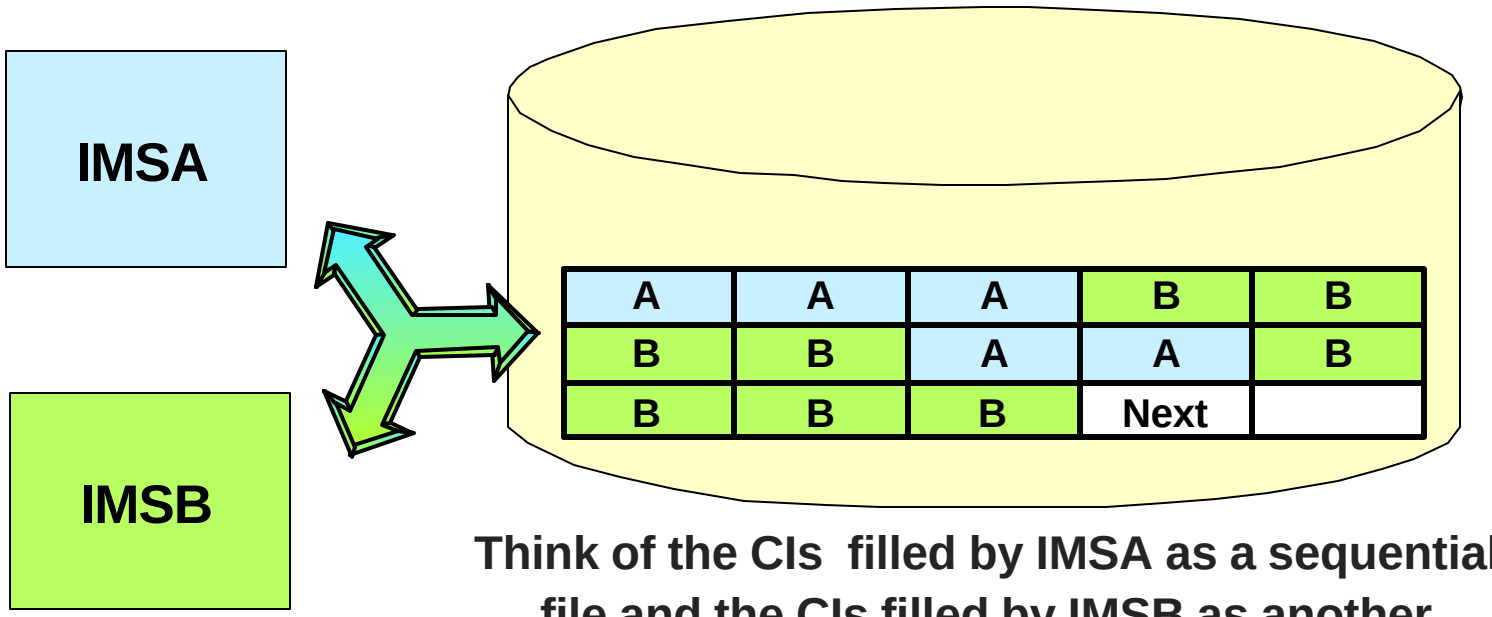
Timestamps used by Scan to sequence SDEPs

- Utility sorts on timestamp

SDEP CI Allocation

SDEP space allocated to sharing IMSs on CI basis

- Allocated CI used by just one IMS for insert
  - Individual CIs are not shared for insert purposes
- Any IMS can read any CI
  - Individual CIs are shared for read or utility purposes

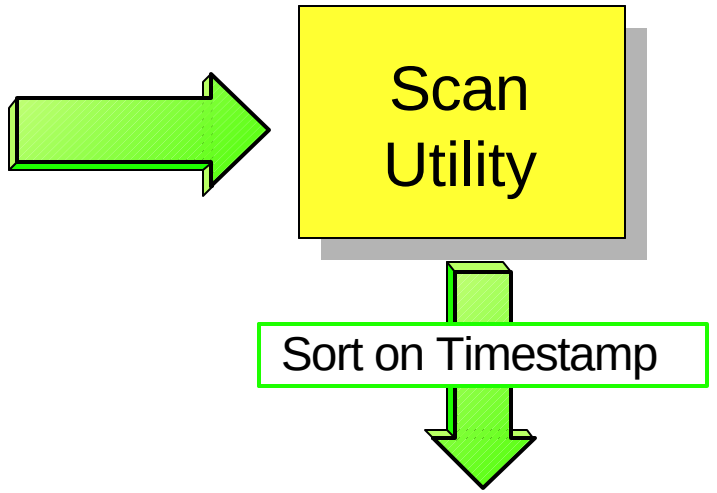


Think of the CIs filled by IMSA as a sequential file and the CIs filled by IMSB as another sequential file.

Use of Timestamps

A2	A2	A5	A5	A5
A9	A9	A10	A11	A11
A11	A13	A13	A17	A17
B4	B4	B4	B6	B6
B6	B8	B16	B22	
A17	A20	A23	A23	A24
A24	A24	A24	A25	A25
A26	A27	A27		

Free (unallocated) CIs
CIs allocated to IMSA
CIs allocated to IMSB



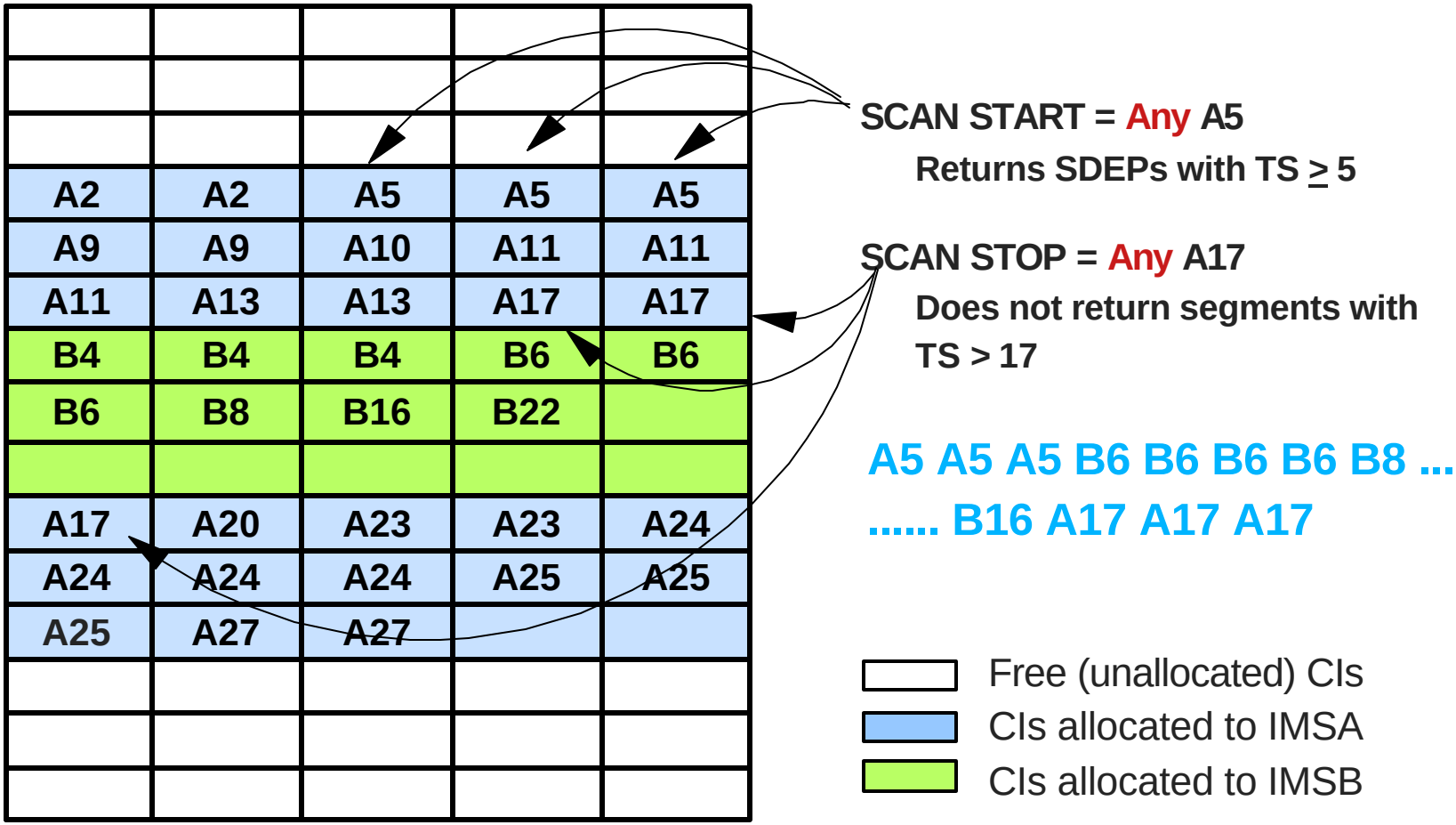
A2 A2 B4 B4 B4 A5 A5 A5 A5
B6 B6 B6 B8 A9 A10 A11 A11
A11 A13 ..... A27

Output (SCANCOPY) of Scan Utility is *sorted by timestamp*, producing a single, merged sequential file.

SDEP Utility Processing

Scan and Delete utilities function on *timestamp boundary*

- Scan retrieves based on value of timestamp



SDEP Summary

Unique segment type for DEDBs only

Audit trail example

- Insert SDEP under updated root
- Retrieve at night and create audit trail, history, ...
- Why?
  - Retrieve data only for updated records (e.g. accounts)

Mass insert

- BMP reads input sequential file
- Inserts records as SDEPs under one or two dummy roots
- Scan later, process, and delete
- Why?
  - Gets data into recoverable format quickly for later processing

DEDB Virtual Storage Option

Benefits

- High performance access to frequently accessed data

Implementation

- Area data read from DASD once and stored in
  - MVS data space when not shared
  - Coupling Facility structure when shared

Good replacement for MSDBs

- Similar performance to MSDBs
- No MSDB restrictions
- Standard backup and recovery procedures

VSO Approach

Load DEDB area into MVS data space

- User option
  - Specified by area in DBRC
- Satisfy user requests from data space
  - No call-related I/O

Provide application program compatibility with MSDBs

- Fixed length segments
- FLD calls

Defining VSO Areas

VSO-related keywords on INIT.DBDS and CHANGE.DBDS commands

VSO	Defines the area as a VSO area
NOVSO	Defines the area as a non-VSO area
PREOPEN	Area opened after the first checkpoint following control region initialization
NOPREO	Area opened during the first call to the area
PRELOAD	For VSO area, causes the entire area (other than the SDEP part) to be loaded into a data space when it is opened
NOPREL	For VSO areas, defines the area as load on demand. When a CI is read from DASD as a result of call processing, it is copied into a data space.

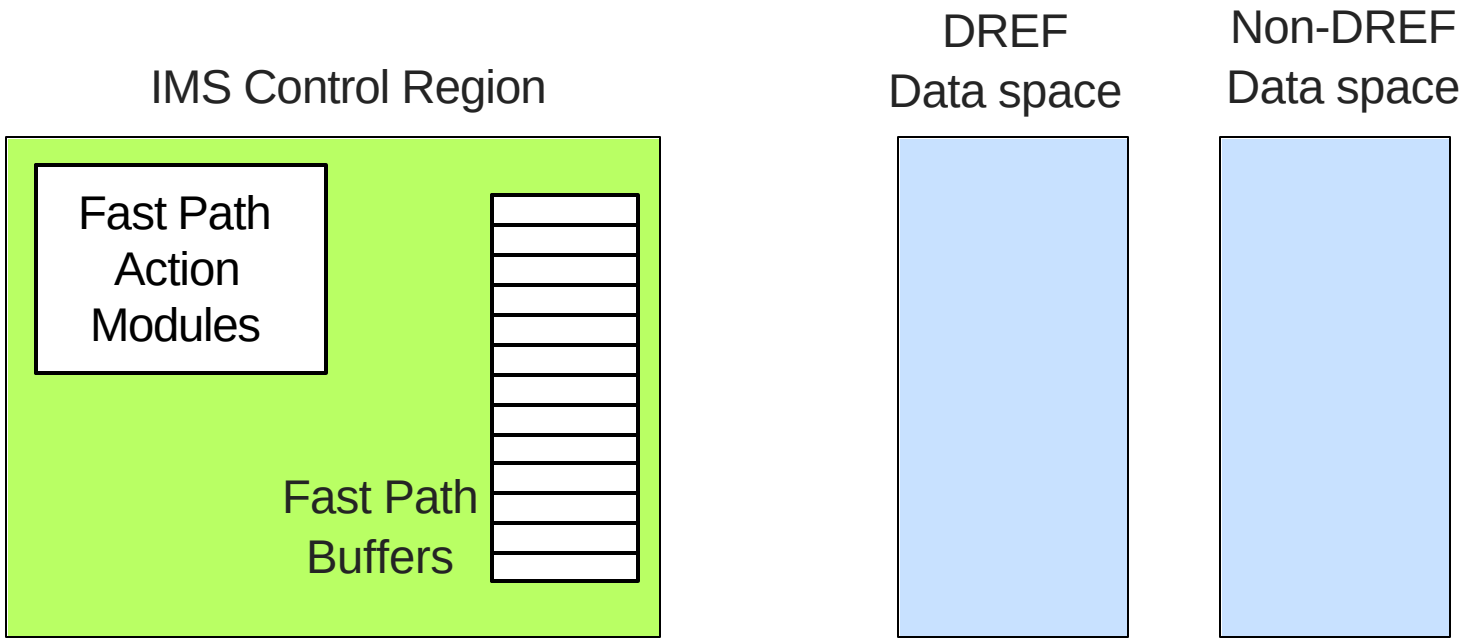
Defining VSO Areas ...

DBRC registration

INIT.DB DBD(dbdname) SHARELVL(0|1|2|3)

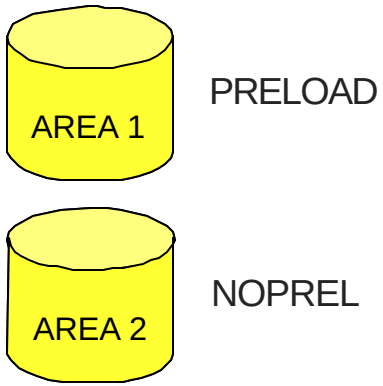
INIT.DBDS DBD(dbdname) AREA(areaname) -
GENMAX(n) -
VSO | NONVSO -
PREOPEN | NOPREO -
PRELOAD | NOPREL

Data Space Acquisition



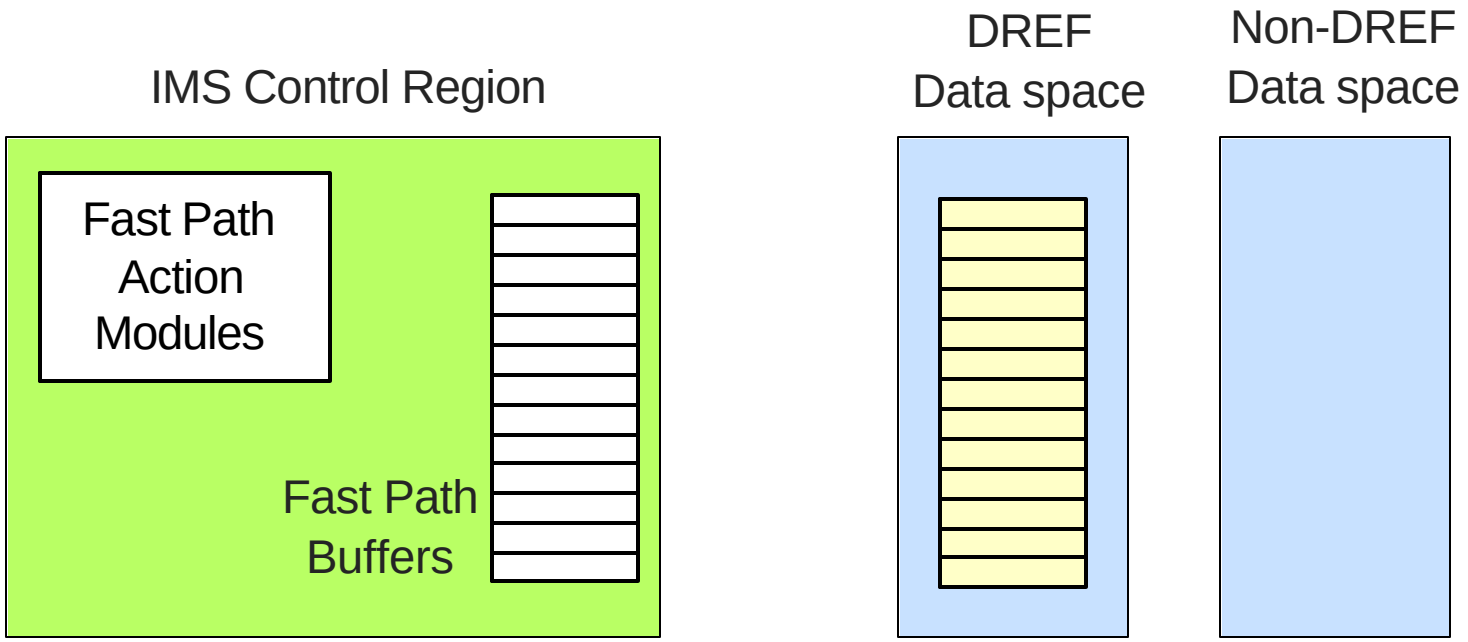
Two maximum size (2 GB) data spaces are obtained at control region initialization

- ✓ One with DREF (disabled reference) for preloaded areas
- ✓ One without DREF option for non-preloaded areas



Additional data spaces acquired, as needed

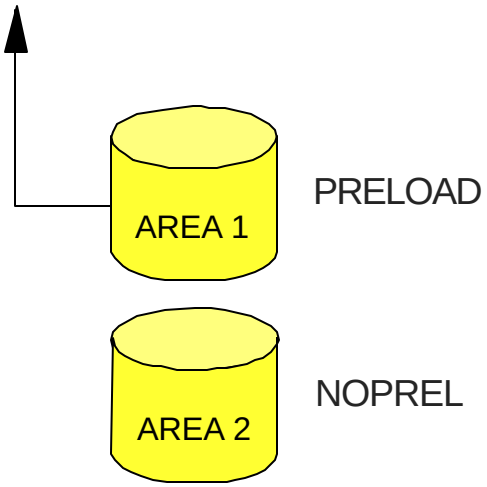
Data Space Access



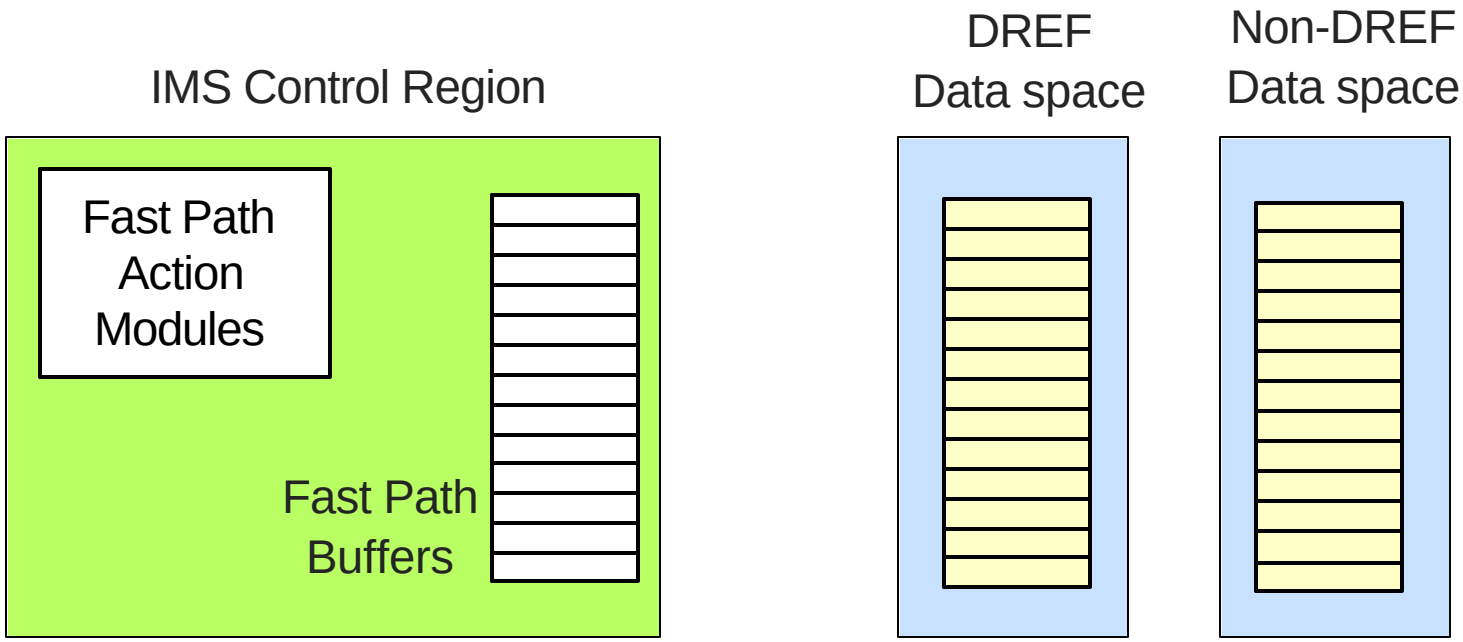
Linear mapping of CIs to virtual storage address in data spaces

PRELOAD areas

- ✓ Loaded into DREF data space when control region initialization completes
- ✓ Data read directly into data space, one UOW at a time

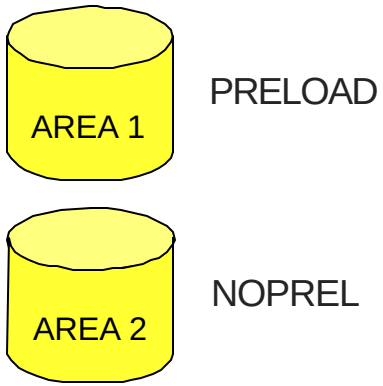


Data Space Access ...

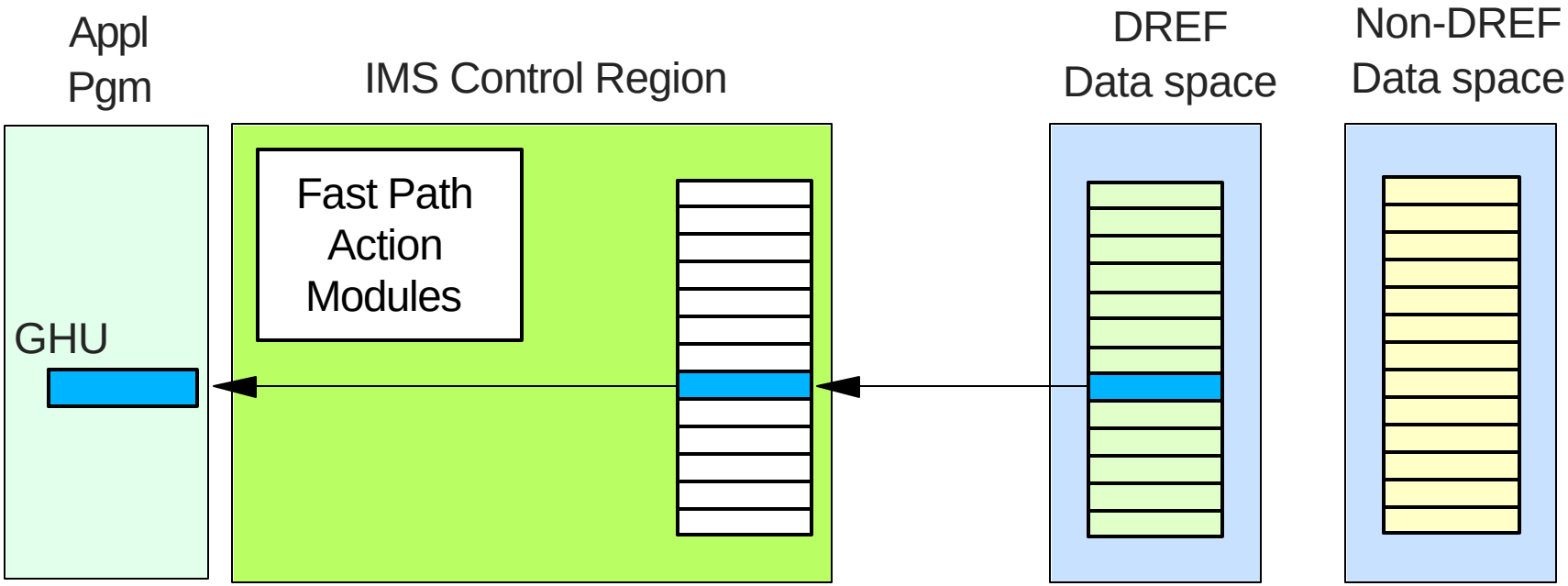


NOPREL areas

- ✓ Space allocated in data space but no data preloaded
- ✓ CI loaded into non-DREF data space when first referenced
- ✓ CI remains in data space until Area closed or /VUNLOAD command

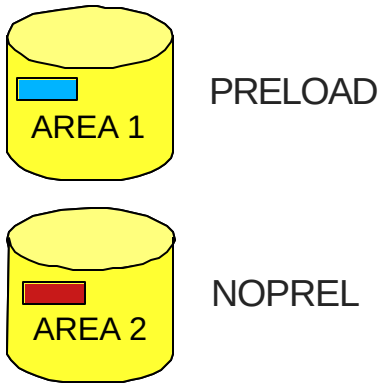


Input Processing (CI in Data Space)

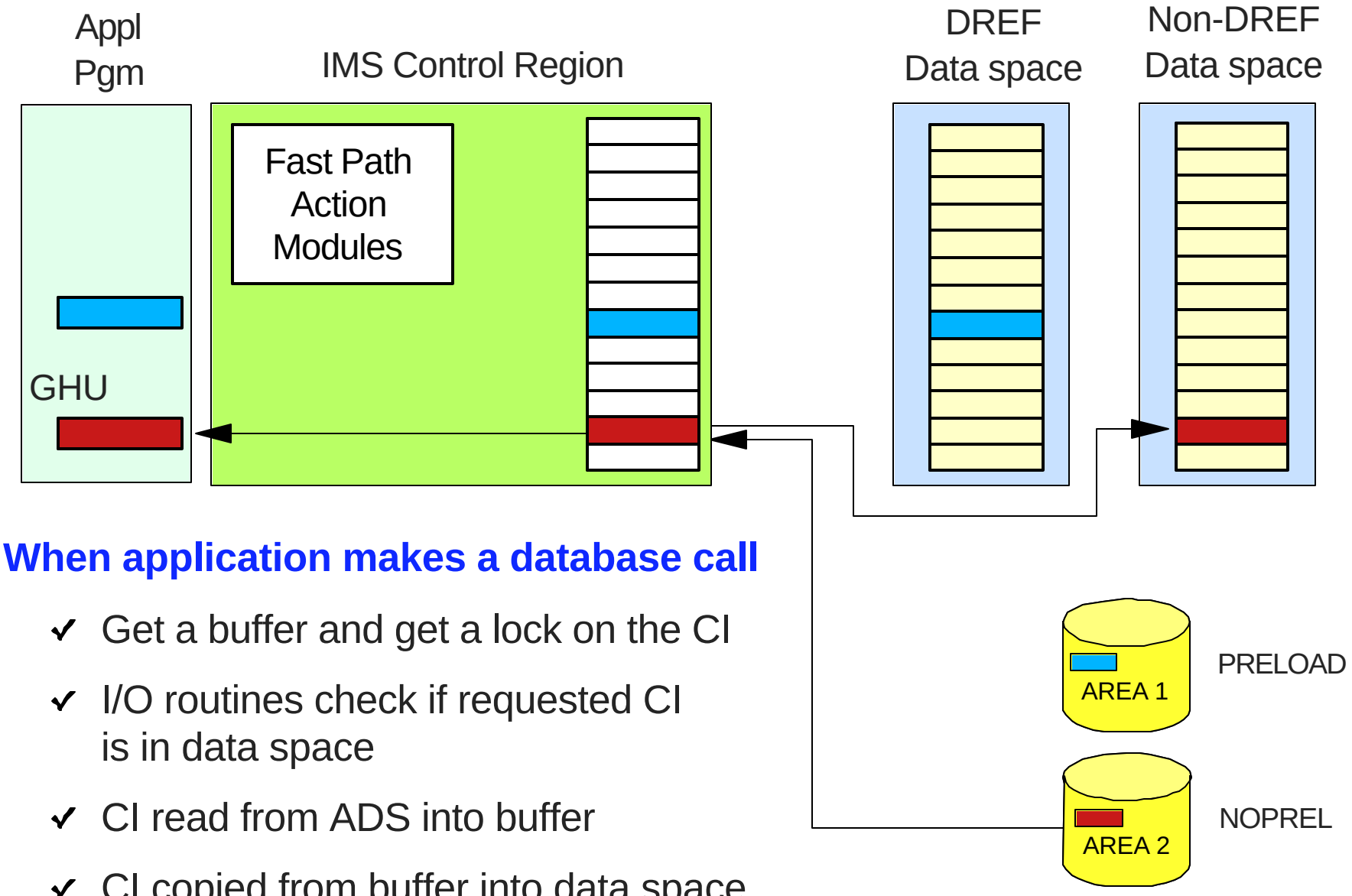


When application makes database call

- ✓ Get a buffer and get a lock on the CI
- ✓ I/O routines check if requested CI is in data space
- ✓ CI copied from data space to Fast Path buffer
- ✓ Fast Path action modules access data in Fast Path buffer



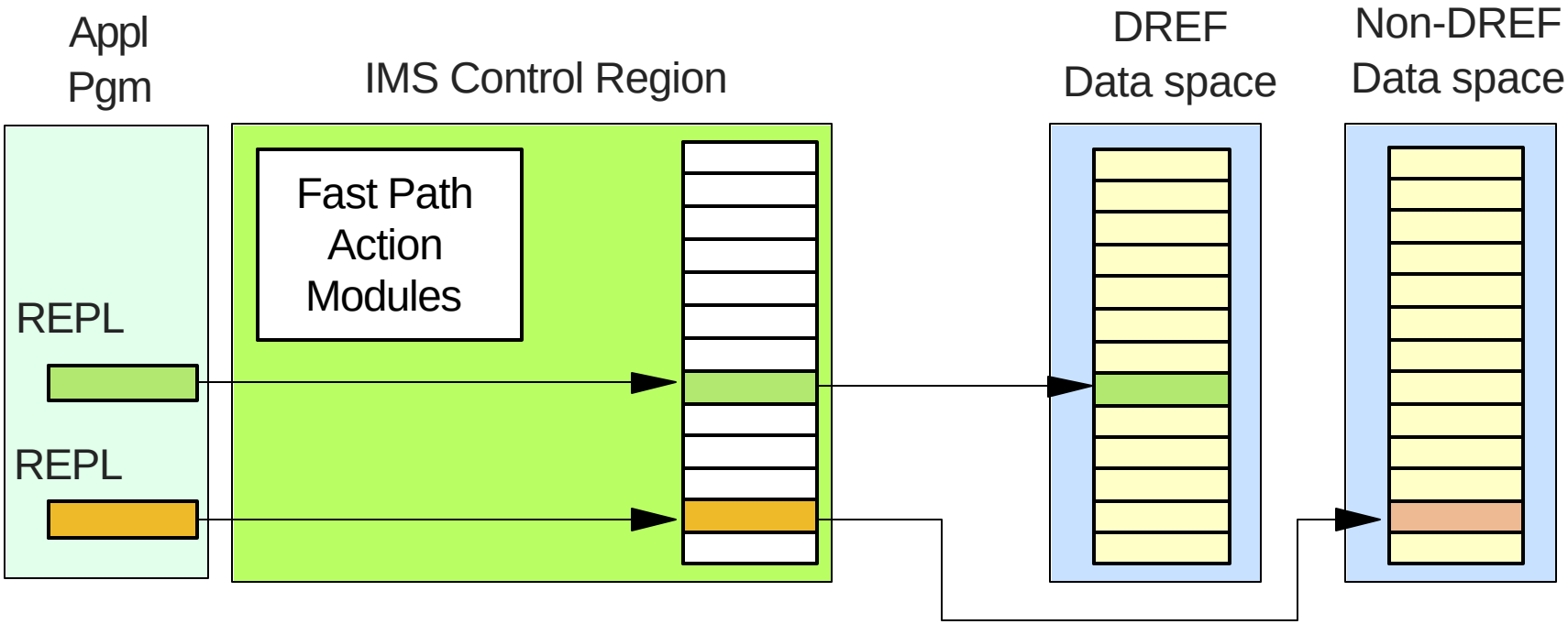
Input Processing (CI Not in Data Space)



When application makes a database call

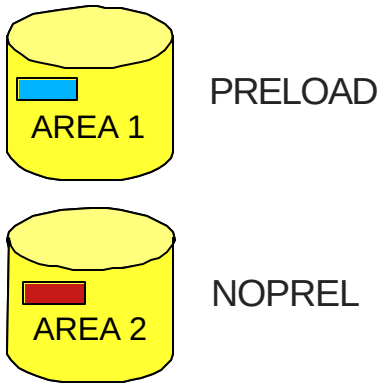
- ✓ Get a buffer and get a lock on the CI
- ✓ I/O routines check if requested CI is in data space
- ✓ CI read from ADS into buffer
- ✓ CI copied from buffer into data space
- ✓ Action modules access data in buffer

Update Processing (Sync Point Phase 2)

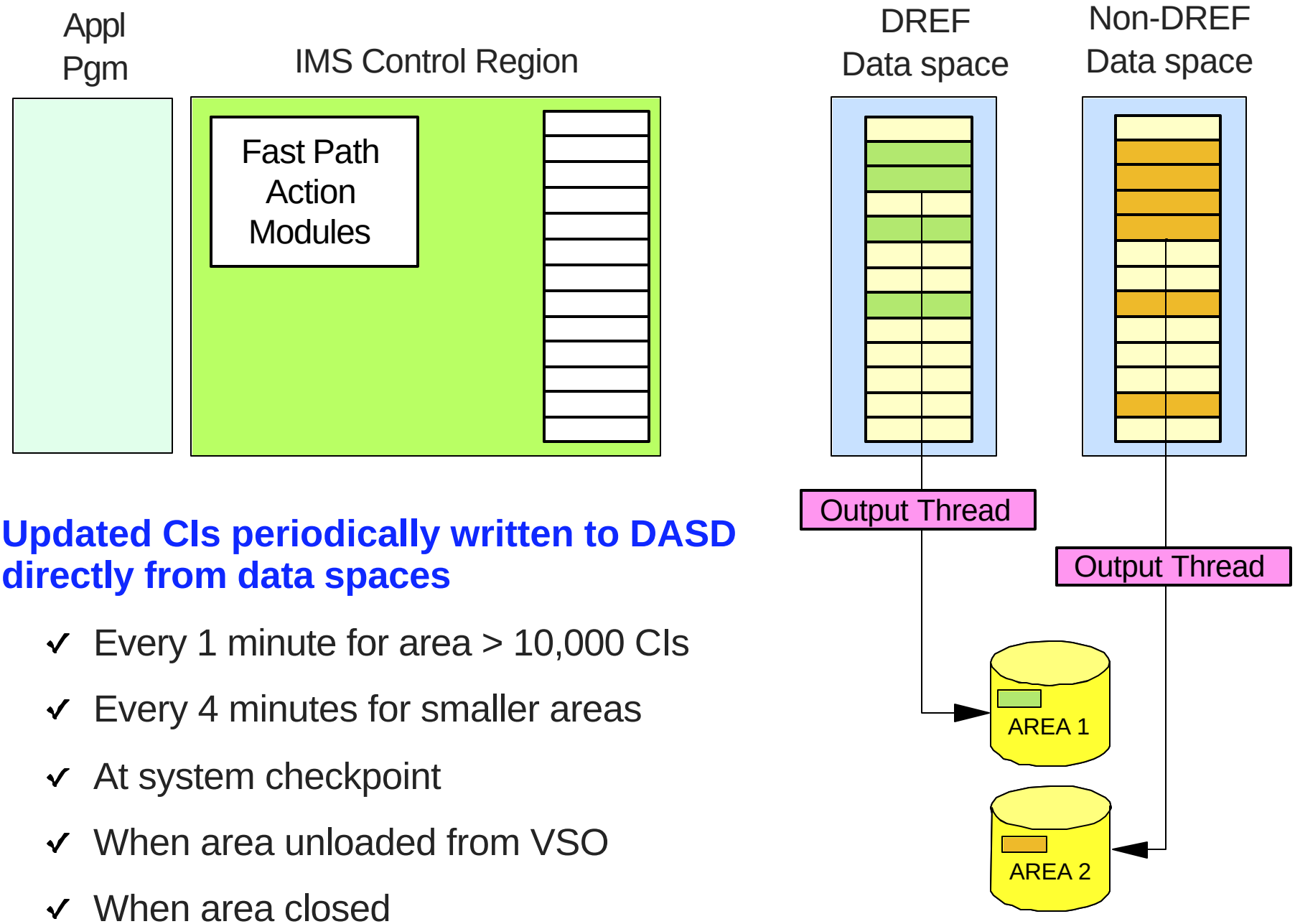


When application commits updates

- ✓ Updated CIs copied to data space
  - No need to wait for physical logging
- ✓ Bitmap of updated CIs maintained in data space
  - For externalizing later
- ✓ Locks are released and buffer returned to available queue
  - Much sooner than with NOVSO



Output Processing



Updated CIs periodically written to DASD directly from data spaces

- ✓ Every 1 minute for area > 10,000 CIs
- ✓ Every 4 minutes for smaller areas
- ✓ At system checkpoint
- ✓ When area unloaded from VSO
- ✓ When area closed

VSO Restrictions

VSO areas must be registered with DBRC

The maximum allowable size for a VSO area is 2 GB - 200K

- 200K used to track contents of data space

SHARELVL 1 supported but be careful

- A given VSO area CI is read only once
- RO subsystem will never see updates to a given CI made by an updating subsystem after RO subsystem has performed the one read

SHARELVL 2 | 3

- Discussed later

Impact on Fast Path Utilities

DEDB High Speed Reorganization Utility

- May be used to reorganize areas with the VSO attribute
- Input and output requests are satisfied via VSO data spaces

DEDB ADS Create Utility

- May be used to create additional copies of a VSO area
- Input requests are satisfied via VSO data spaces
- VSO facilities not used for output functions

DEDB Multiple Area Data set Compare Utility

- Cannot be run for an area that is in a data space

Fast Path Log Analysis Utility

- Provides performance statistics for VSO areas

VSO Commands

/DIS FPVIRTUAL

- Display fast path data space usage by VSO areas

```
DATASPACE   MAXSIZE(4K)  AREANAME    AREASIZE (4K)  OPTION
000         524263                                  DREF
                         DB21AR2     167            PREO, PREL

                         DB21AR4     42             PREO, PREL
001         524263
                         DB21AR1     84             PREO
                         DB21AR3     84
```

VSO Commands ...

/VUNLOAD

- Remove VSO area from data space

DFS0448I VUN COMMAND COMPLETED. AREA = DB21AR5 RC=0

/START AREA DB21AR5

- Reload VSO area into data space

Non-shared VSO Summary

Provides I/O performance characteristics similar to MSDBs while allowing

- Hierarchical structures
- Insert/delete capability
- DBRC support
- Standard recovery procedures
- No ETO implications
- No APPC or OTMA restrictions

Provides application program compatibility with MSDBs

- Fixed length segments
- FLD call support

DEDB Virtual Storage Option

Application program access

- From global buffers

DLI read/write access

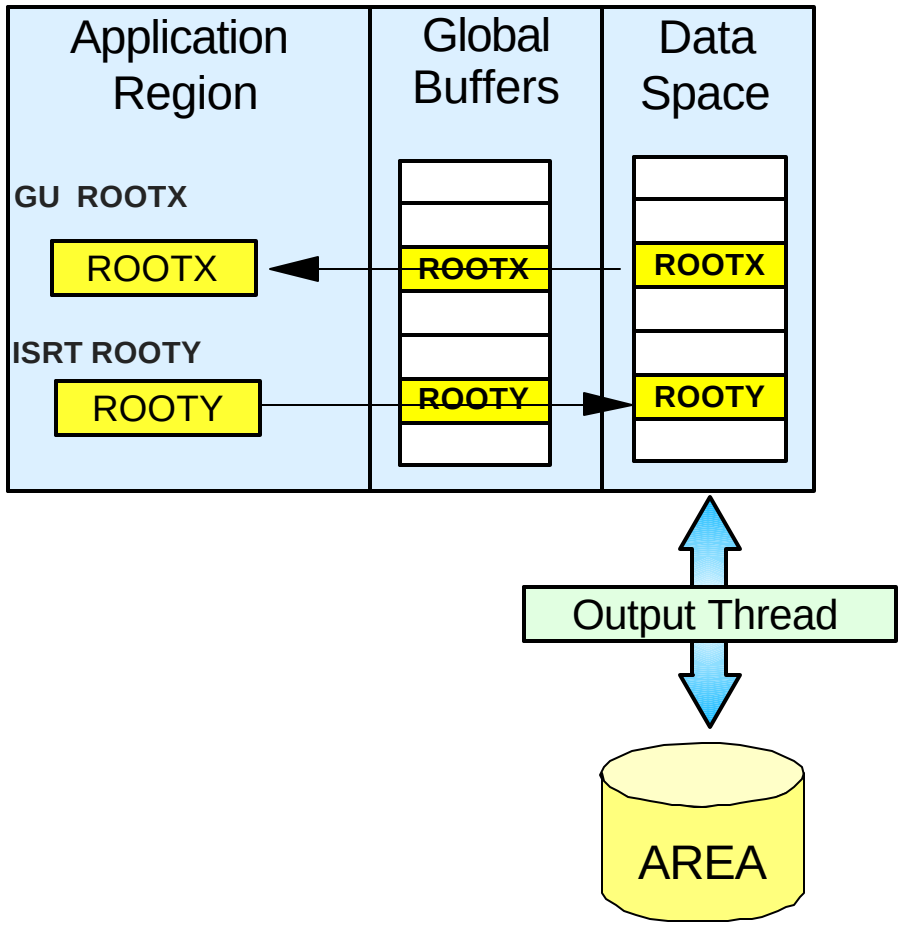
- From Data Space

Cast-out processing

- Writing updated CIs in data space to DASD
- Occurs at system checkpoint
  - Asynchronous output thread

Performance

- Read access at memory speeds



BLDS for DEDBs with VSO Areas

Coupling Facility cache structures replace MVS data spaces

- ***Store-in Cache*** structures hold Area CIs in CF
  - Used to store data and for buffer invalidation
  - One structure for each shared VSO area
    - Dual structures optional
- CIs loaded into cache structure
  - At IMS initialization (PRELOAD option)
  - When first referenced

Private buffer pools used within each IMS

- Shared VSO access uses private buffer pools
  - Fast path global pool not used
  - Can be dedicated to single area
- Private pools provide ***look-aside buffering***

Shared VSO

Application program access

- From private buffers

DLI read/write access

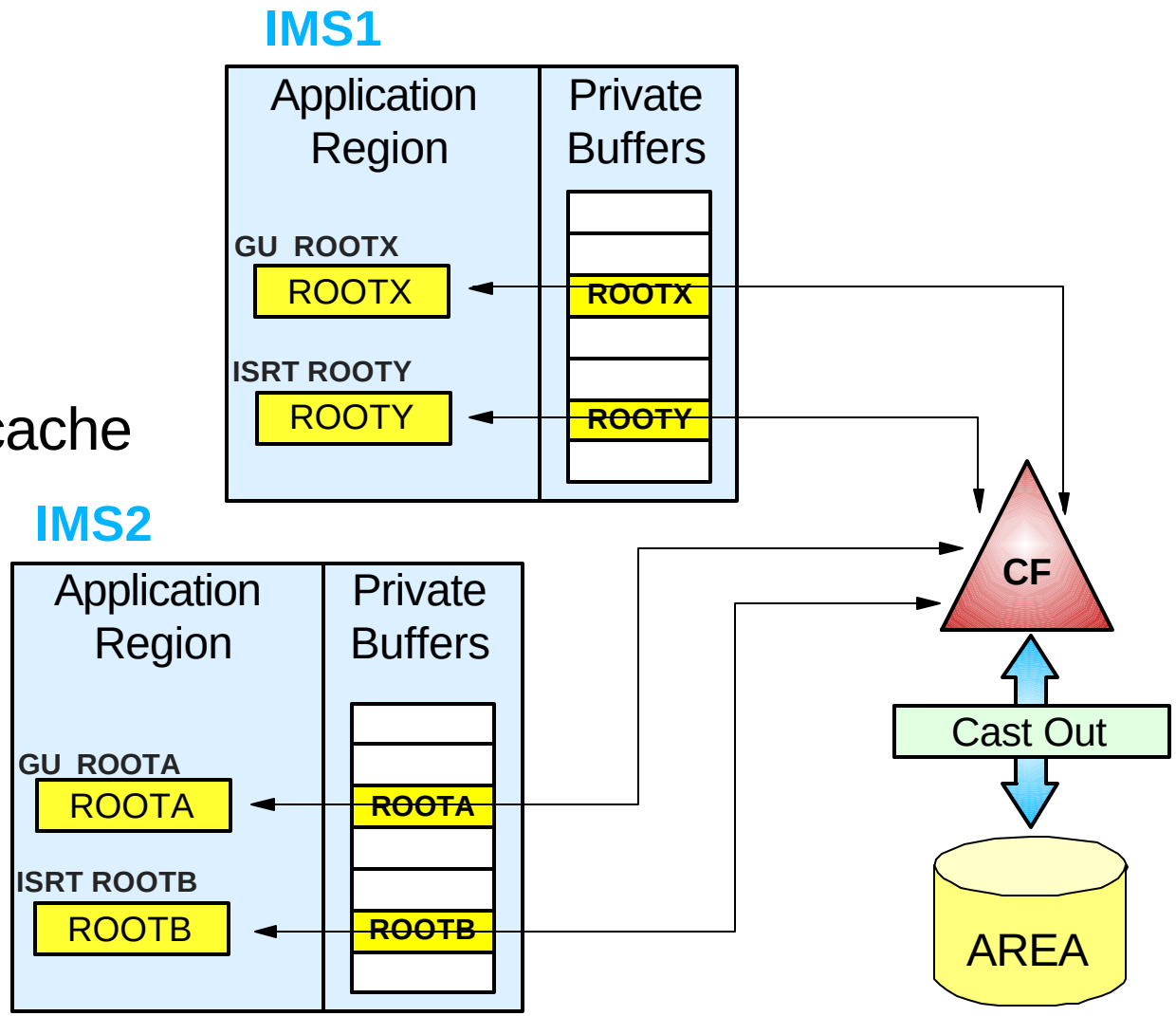
- From Cache Structure

Cast-out processing

- Writing updated CIs in cache structure to DASD
- Occurs at system checkpoint

Performance

- Read access at CF access speeds



VSO Summary

Good alternative too MSDBs

- Good performance without restrictions

Good option for any

- Small heavily used database
  - Preload entire area
- Large database with "hotspots"
  - Don't preload, load only active data

Performance still good even when sharing

- Access at CF speeds
- Private buffers provide look-aside buffering

RETURN TO INDEX