S60

Fast Path Data Entry Database Performance Considerations

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

IMS Technical Conference

Miami Beach, FL October 22-25, 2001

RETURN TO INDEX

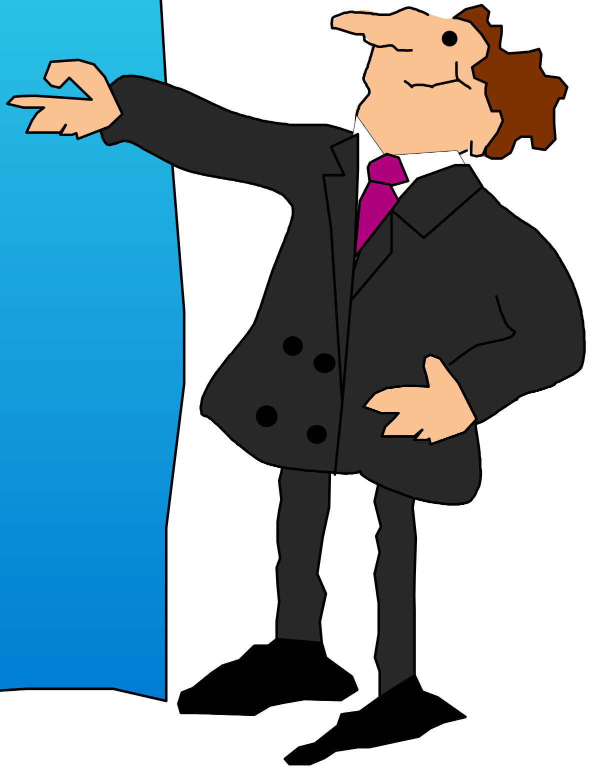
Agenda

Audience

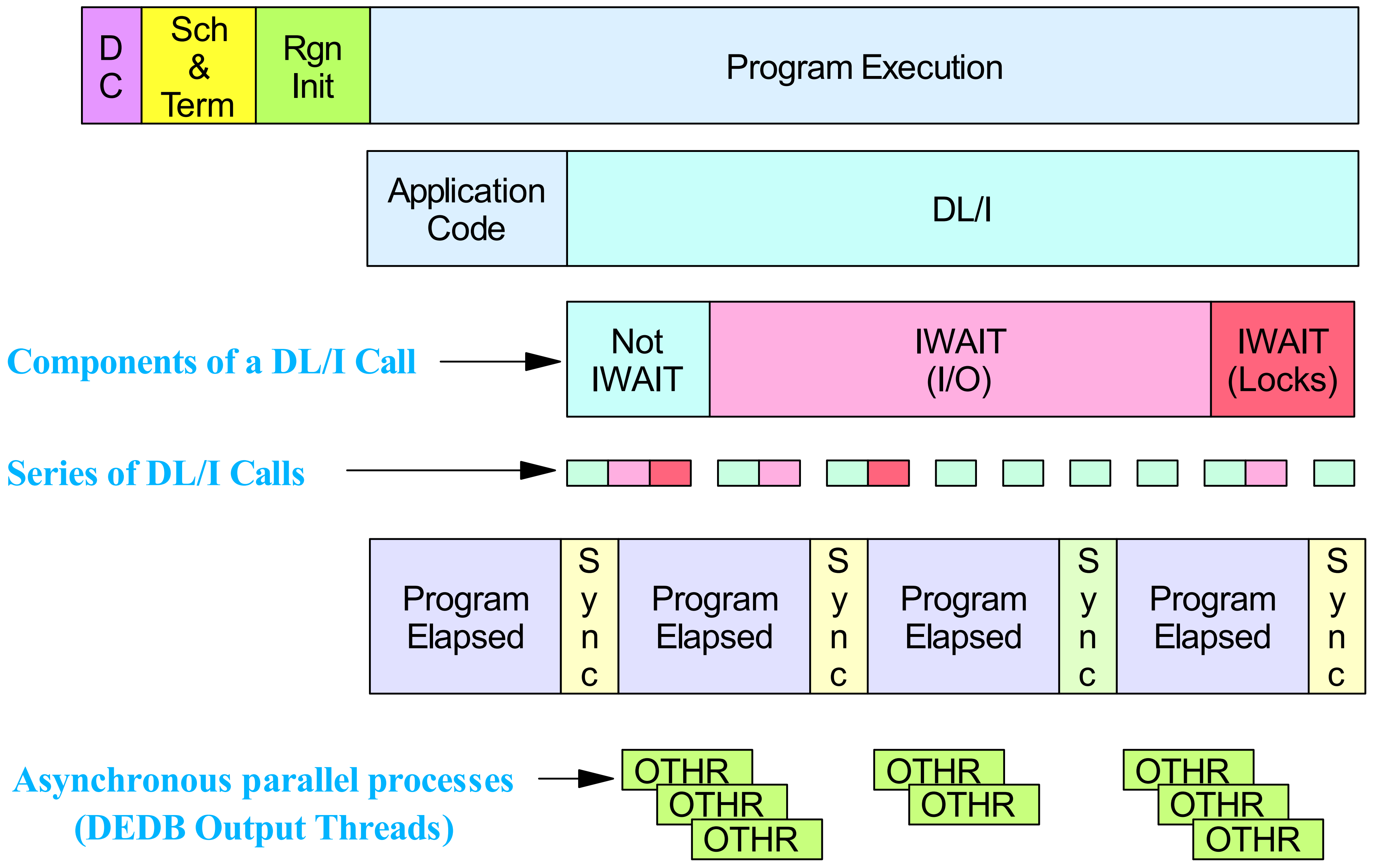
- ★ Users familiar with
  DEDB Fundamentals

Topics

- ★ DEDB structure
- ★ DEDB processing
- ★ Performance issues
- ★ Monitoring and tuning



Response Time Components

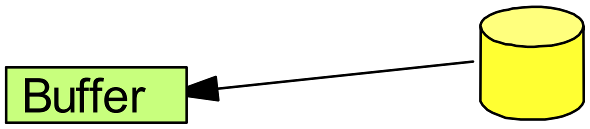


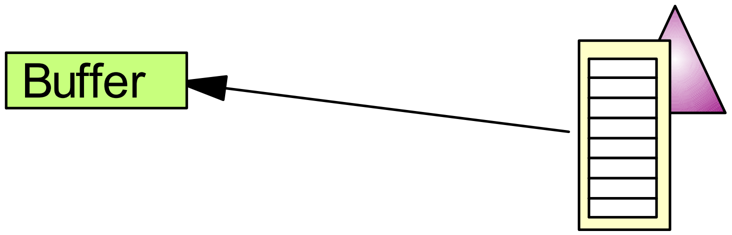
DEDB Performance Components

During DEDB call processing

- Not IWAIT
  - Call pathlength
  - Buffer management
    - Wait for buffer

- IWAIT
  - Lock contention
    - CI lock
    - UOW lock
  - Read I/O from DASD
    - To retrieve segment
    - To find space to insert segment
  - VSO GET from data space or CF
    - To retrieve segment
    - To find space to insert segment

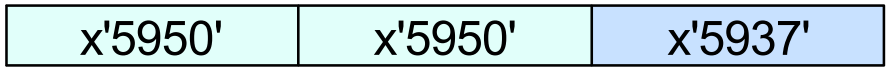




DEDB Performance Components ...

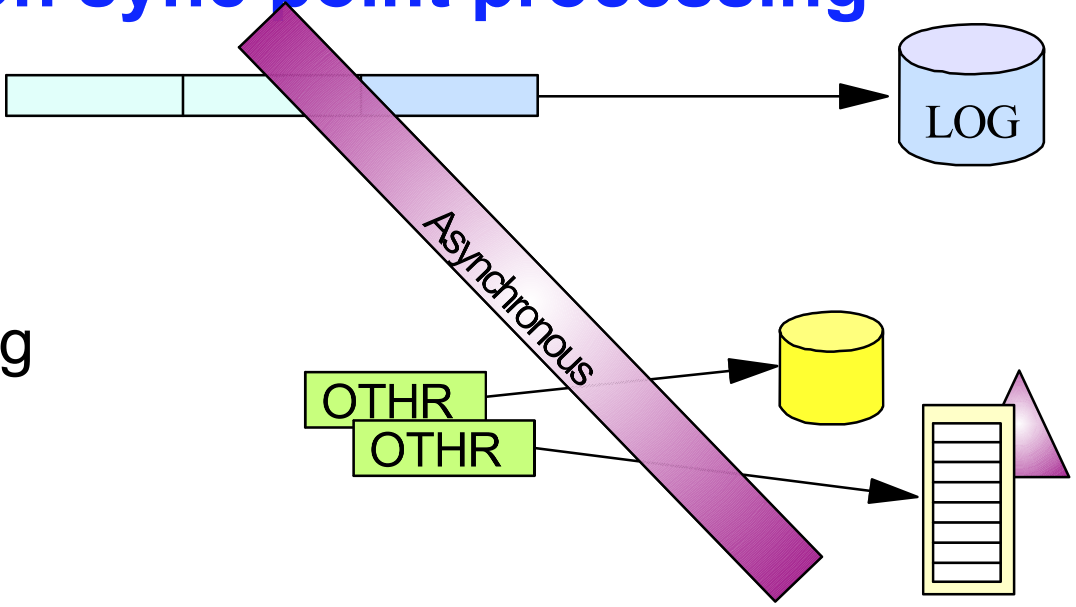
During dependent region sync point processing

- Logical logging
  - LOGL latch



After dependent region sync point processing

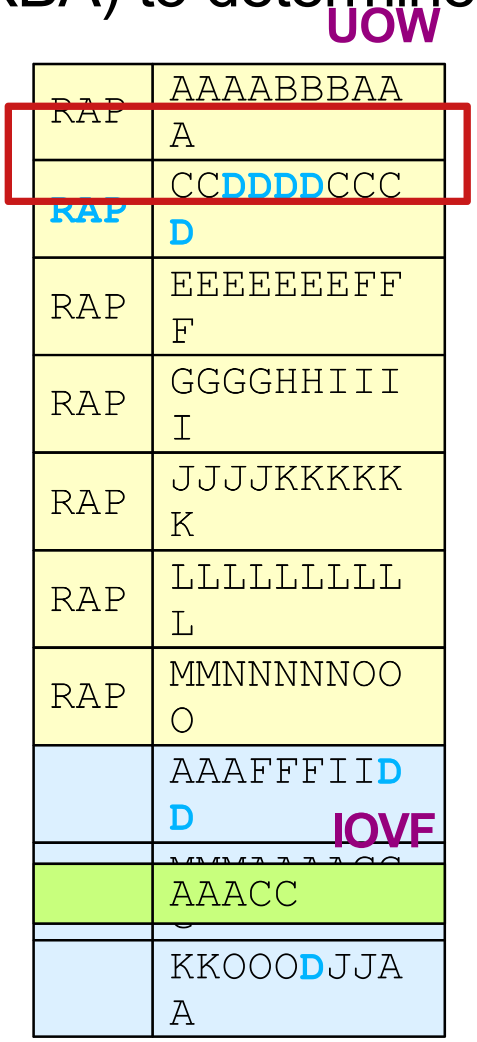
- Physical logging
  - OLDS or WADS
- Output thread processing
  - DASD writes
  - VSO PUTs
- Locks and buffers held
  - Until output thread completes



DEDB Call

Retrieval call

- Call randomizer (Area/RAP) or follow pointer (RBA) to determine CI
- Look for CI in buffer pool
- If not in buffer pool
  - Get CI lock (may have to wait)
  - Get buffer (may have to wait)
  - Read CI from DASD or VSO (wait)
- Look for segment in CI
- If not in CI
  - Follow pointers until found (or GE)
    - May have to repeat "get lock, get buffer, read CI" step



DEDB Call ...

Retrieval call ...

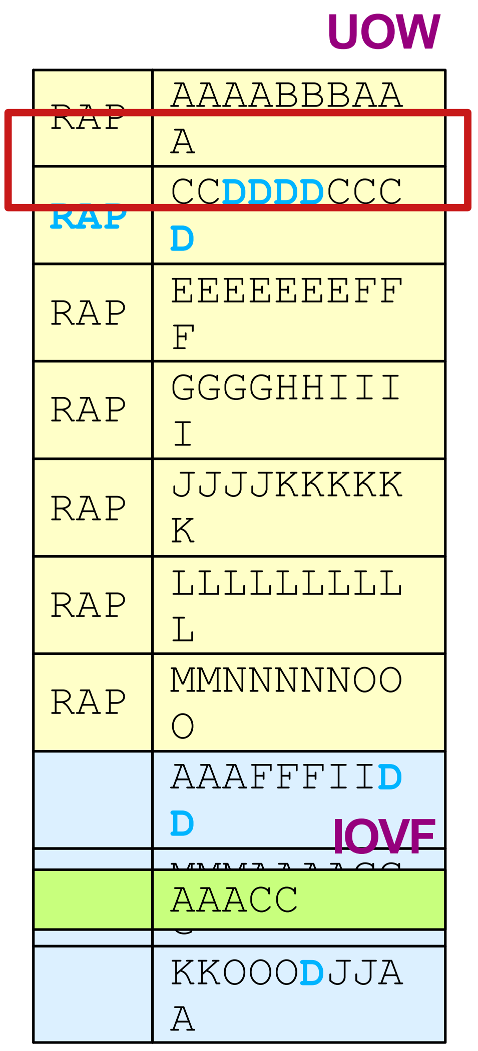
- When segment found
  - Expand if segment compressed
  - Pass segment to program I/O area

Delete call

- Similar to full function
  - Update data in buffers
  - May require additional CIs to retrieve and delete children

```
GHU   ROOTD  (1 I/O)
DLET         (2 I/Os)
```

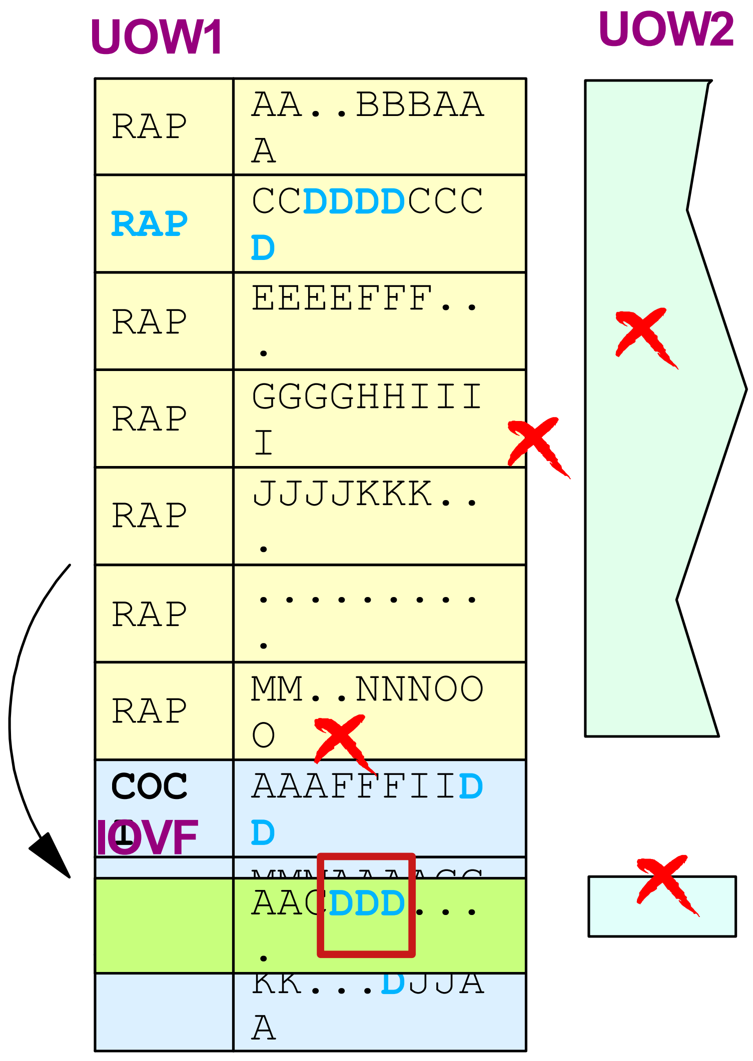
    - Requires a total of 3 I/Os at call time
  - Additional I/Os required when DLET frees up DOVF or IOVF CI completely



DEDB Call ...

Insert call

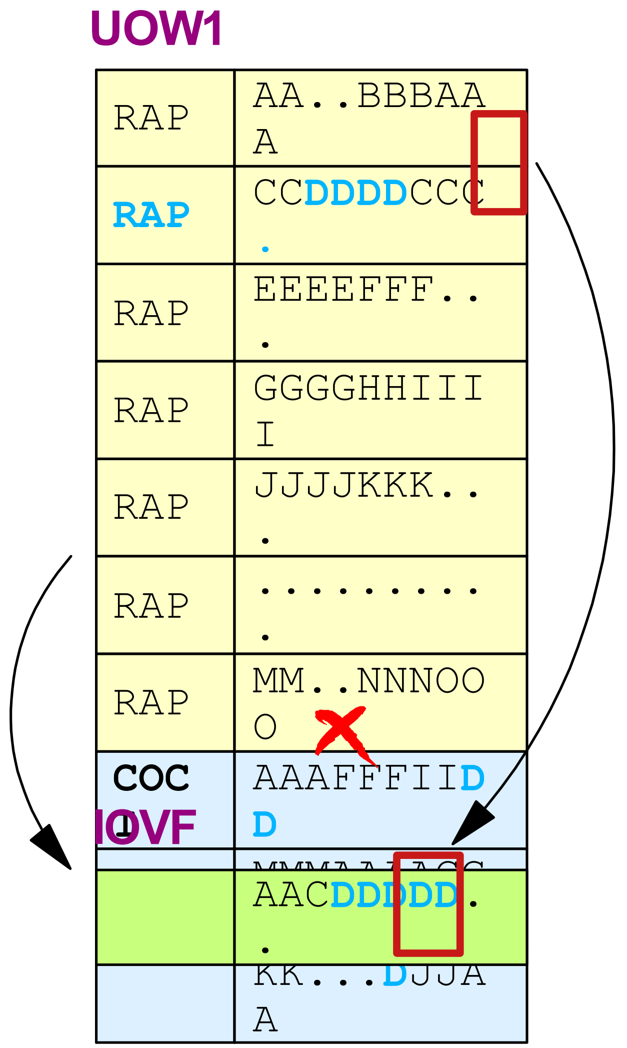
- Determine "most desirable CI"
  - For root segment
    - RAP CI
  - For dependent segment
    - Root CI
- Try to place segment in MDCI
- If no room, find space (probably involves *locking and I/O*)
  - In DOVF (always look here first)
  - In IOVF (if no room in DOVF)
  - Never in another RAP CI or another UOW
  - Never share an IOVF CI with data from another UOW



DEDB Call ...

Replace call

- If segment length does not change
  - Replace in place
- If segment length changes (larger or smaller)
  - *Internal delete and insert*
    - Try to place in MDCI
  - Segment may move
    - Could be more I/O "get lock, get buffer, read CI"



Where to Look

Fast Path Basic Tools - DEDB Pointer Checker

- "Segment Placement Analysis"
  - Try to size AREA to minimize number of segments in DOVF & IOVF

```
SEGMENT PLACEMENT ANALYSIS
                                       ---IN RAA BASE---      ----IN DOVF----    ----IN IOVF----
  SEGNAME    SCD  LVL   TOT #OCCS      NO. OCCS     P/C      NO. OCCS     P/C    NO. OCCS     P/C

  TSSROOT      1    1          83            44    53.0            17    20.5          22    26.5
  TSSDIR1      3    2         317           189    59.6            76    24.0          52    16.4
  TSSD11       4    3           6             0     0.0             6   100.0           0     0.0
  TSSD111      5    4          21             0     0.0            21   100.0           0     0.0
  TSSD12       6    3           0
  TSSDIR2      7    2         676           225    33.3           261    38.6         190    28.1
  TSSDIR3      8    2         173            93    53.8            39    22.5          41    23
```

- "Free Space Analysis"
  - Reports free space in BASE, DOVF, and IOVF
  - For best random performance, minimize RAP CIs with no free space

DEDB Locking

DEDB "record" locking is at CI level

- Entire CI locked (may have to wait)

If HSSP active in AREA

- UOW is locked
  - Both HSSP and non-HSSP programs get UOW lock
- Discussed later

Fast path manages its own locks

- Calls Program Isolation or IRLM only for deadlock detection
- If deadlock detected
  - Same as full function, except BMP gets FD status instead of U777

Lock not released until buffer is released

- Some exceptions

Where to Look

Fast Path Log Analysis Report (DBFULTA0)

- "Detail Exception Report"
  - Reports by individual transaction occurrence

```
TRANCODE     SYNC POINT    ......   CONTENTIONS
 OR PSB         TIME       ......   CI  UW  OB  BW

DEDBTRN1     15:04:18.15             2   0   0   0

DEDBTRN2     15:04:18.18             0   1   0   0

HSSPBMP      15:04:18:20             0   0   0   0
```

  - Transactions wait for UOW lock only when in contention with HSSP BMP (or Online Reorganization Utility)

Where to Look ..

Fast Path Log Analysis Report

- "Overall Summary of Resource Usage and Contentions ..."
  - Summarizes by Transaction Code or PSB

```
TRANCODE  NO. OF   ......  CONTENTIONS   TRAN
 OR PSB   TRANS            TOT TOT CI/   RATE
                           UOW OBA SEC   /SEC

DEDBTRN1    1492             0   0   1      3
DEDBTRN2   22986            17   0   0     45
DEDBTRN3   18520             0   0   2     37
```

  - DEDBTRN3 CI contentions could be a problem

Program Isolation Trace Report

- PI used by fast path only when wait condition occurs

Buffer Management

Buffer allocation for read I/O (or VSO get)

- If all NBA buffers in use
  - Steal all unmodified buffers for reuse
  - Release locks on stolen buffers
- If cannot steal
  - Get OBA latch to use OBA buffers
  - May have to wait for latch (reported as OB CONTENTION)
    - No problem if don't have to wait
- Buffer allocated from available buffers
  - /DIS POOL FPDB

```
FPDB POOL:
AVAIL=1846 WRITING=92 PGMUSE=419 UNFIXED=2643
```

    - If none AVAIL, program waits (reported as BW CONTENTION)
- Program cannot exceed NBA + OBA
  - U1033 abend (or FR status for BMP)

Buffer Management ...

During program execution (before commit)

- Program uses NBA (+ OBA) buffers
  - OBA latch acquired, if necessary
- Updates held in buffers, changes not logged (yet)
  - Not written to DASD
- If program abends, backout not needed
  - Updates in buffers are discarded

DBFX	DBFX	DBFX	DBFX	DBFX	DBFX	DBFX	DBFX	DBFX
DBFX	DBFX	DBFX						
							CSDB	CSDB
OBA	OBA	OBA	OBA	OBA	OBA			
NBA1		NBA3						
NBA1	NBA2	NBA3						
NBA1	NBA2	NBA3	NBA4					
NBA1	NBA2	NBA3	NBA4					
NBA1	NBA2	NBA3	NBA4					

Program has updated 6 CIs (buffers) in 5 NBA and 1 OBA buffers

Where to Look

Fast Path Log Analysis Report

- "Detail Exception Report"

```
TRANCODE     SYNC POINT   ......          ADS             CONTENTIONS
 OR PSB         TIME      ......        RD UPD            CI UW OB BW

DEDBTRN1     15:04:18.15               215 102             0  0  1  1
BUFFER - NBA= 100 OVFN=  17 STEAL=   2 WAIT=  1
```

  - Program used all 100 NBA buffers and invoked buffer stealing twice
  - Program used 17 OBA buffers
  - Program had to wait for ...
    - OBA latch - not too bad if only occasionally
    - Available buffer - should never happen - increase BFIX
  - "Overall Summary of Resource Usage ..." gives averages/totals by Trancode or PSB

Sync Point Processing

Phase 1

- Logically log all updates
  - x'5950'
- Logically log x'5937' sync record
  - Equivalent to full function x'37'
- May want more OLDS buffers to handle sudden spikes during sync point processing
  - Especially if high update BMPs
  - Monitor logger statistics in x'4507' log record

Physical logging

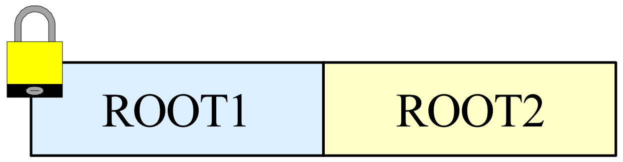
- BMPs
  - Checkwrite x'5937'
- MPPs and IFPs
  - Wait for buffer to be written when buffer full or ...

Sync Point Processing ...

Phase 2

- Release all unneeded buffers and locks
  - Buffers/CIs containing unmodified data
- Transfer lock ownership of updated CIs to Output Threads
  - Once transferred, even this program cannot access CI

```
GU ROOT1
  Lock CI
SYNC
  Transfer lock to output thread
GU ROOT2
  Wait for lock (CI contention)
```



  - Happens most frequently in sequential BMPs
    - Can be avoided by using HSSP

Output Thread Processing

Invoked by physical logger

- When x'5937' written to OLDS or WADS

Output Threads

- One per area
- Execute under SRB in control region
- Updated CIs chain written to Area Data Sets
  - ***Asynchronous*** to dependent region processing
- Seldom a problem unless not enough threads (OTHR=nnn)
  - Very cheap - be generous - set OTHR=255
- When complete
  - Buffers freed and locks released

DEDB Virtual Storage Option

Application program access

- From global buffers

DLI read/write access

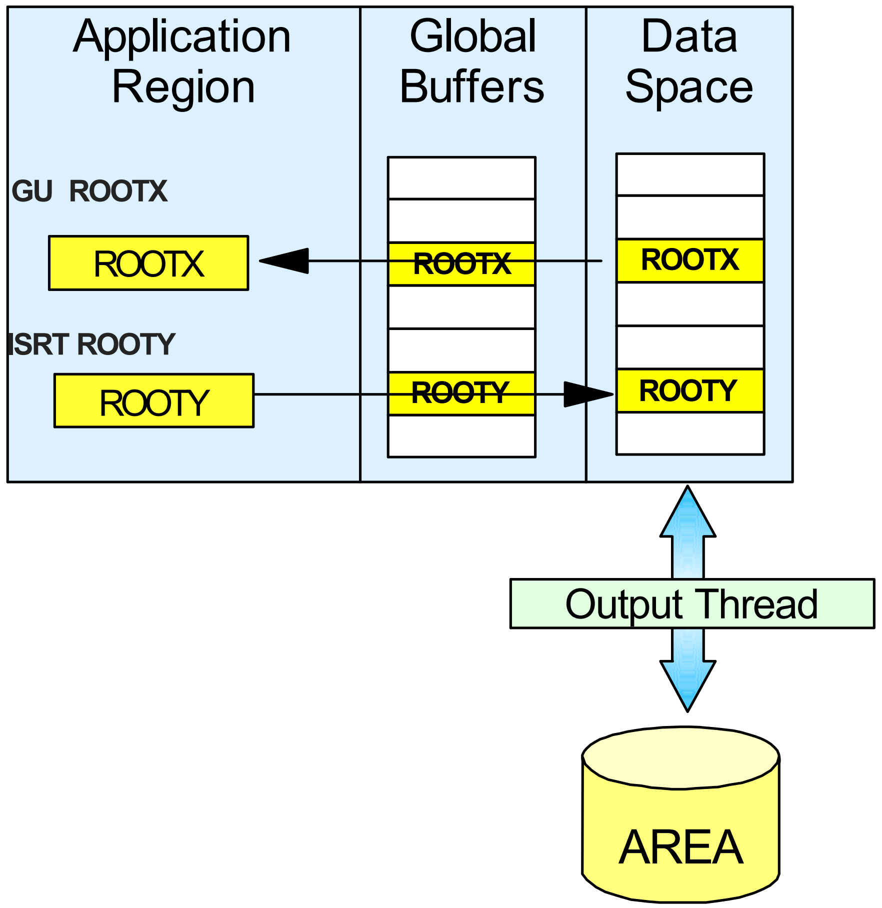
- From Data Space

Cast-out processing

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

Performance

- *Read access at memory speeds*



Shared VSO

Application program access

- From private buffers

DLI read/write access

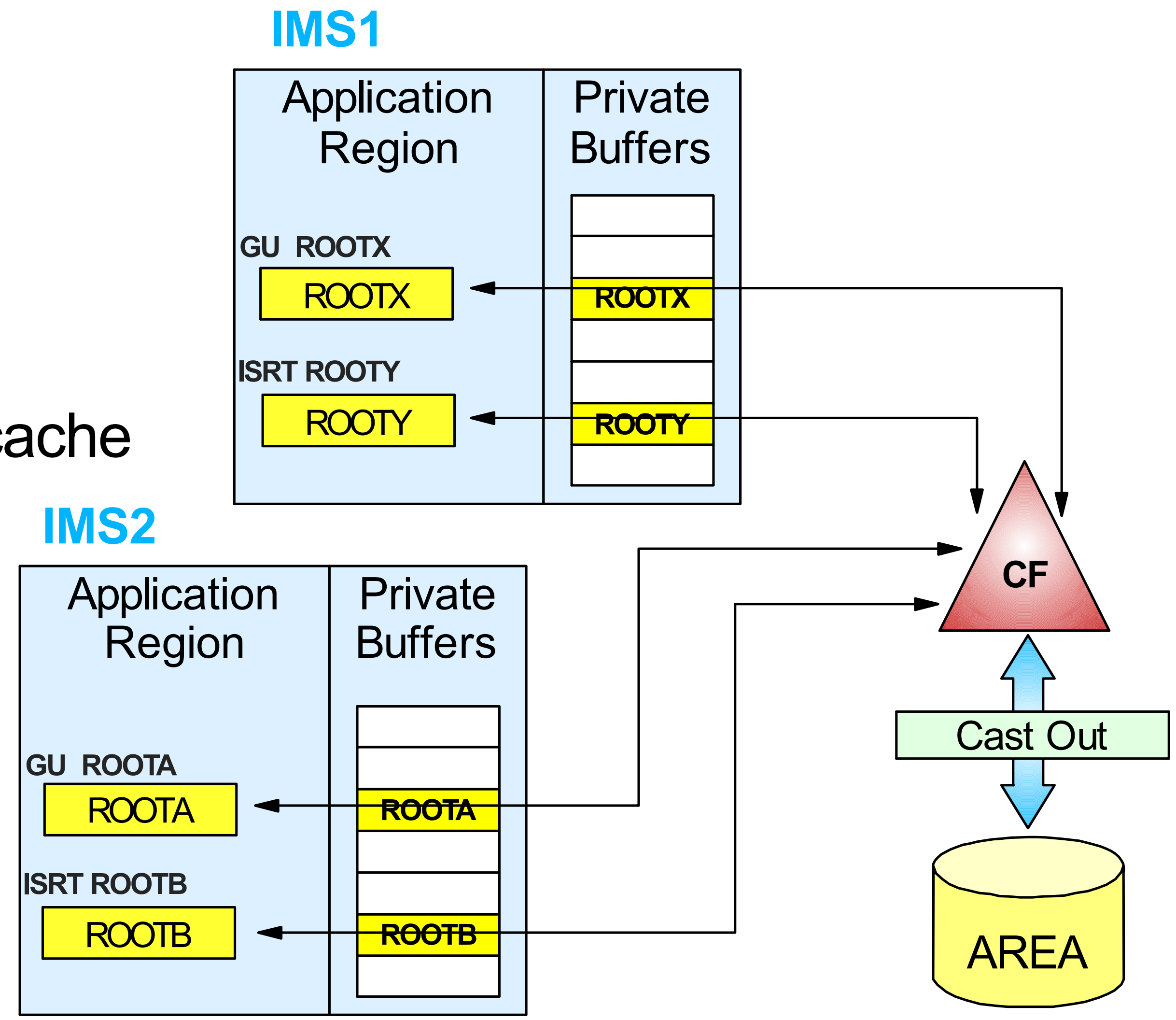
- From Cache Structure

Cast-out processing

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

Performance

- *Read access at CF access speeds*



Where to Look

Fast Path Log Analysis Report

- "Detail Exception Report"
  - VGET replaces ADS RD

```
TRANCODE     SYNC POINT    ......           ADS              CONTENTIONS
 OR PSB         TIME       ......         RD  UPD            CI  UW  OB  BW

DEDBTRN1     15:04:18.15                  15    2             0   0   0   0
VSO      - VGET    21 VPUT     1 DGET     0
```

- "Summary of VSO Activity"
  - VSO GETS and PUTS would be DASD Reads and Writes

```
 SHR(0/1)     VSO        VSO        DASD       DASD       I/O
   AREA       GETS       PUTS       GETS       PUTS       SCHED
 ________   ________   ________   ________   ________   ________

 IAGAI401     41855      23067          0        300          6
 BQSPB4A2      7931       7097          0        396          6
```

Sequential Processing

Many BMPs process data sequentially

- GN processing
- GU processing with driver file
  - Driver file must be sorted inArea/RAP sequence

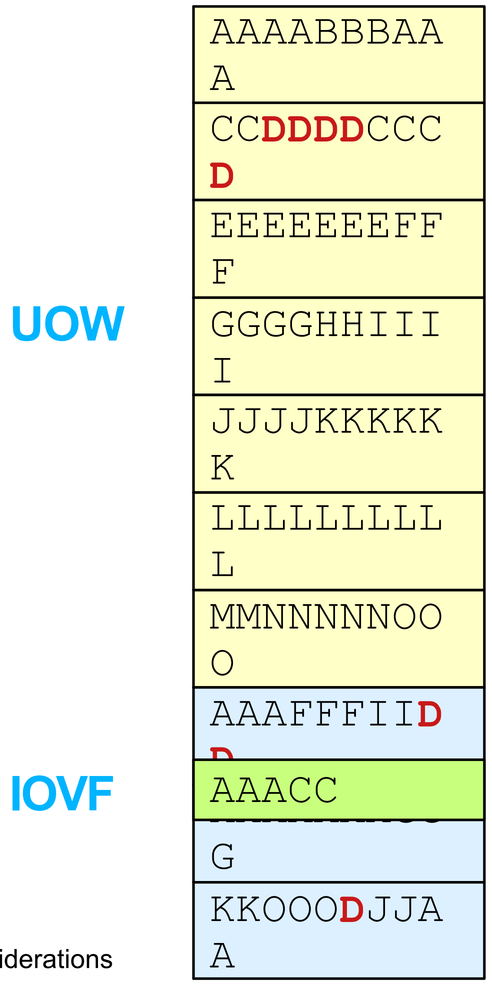
Two techniques to optimize sequential processing

- Database (area) space allocation
- High speed sequential processing (HSSP)

Space Management for Seq'l BMPs ...

Sequential processing

- To optimize (minimum I/Os), reduce space allocation
  - UOW space management keeps data together
  - May hurt random processing



Although data overflows, all data can be retrieved sequentially with 11 I/Os.

Random retrieval of record D would require 3 I/Os.

Sequential retrieval of record D would require 0 I/Os (CIs already in buffers).

High Speed Sequential Processing (HSSP)

Requested in PCB

```
PCB    TYPE=DB,DBDNAME=ACCTDB,PROCOPT=HA
```

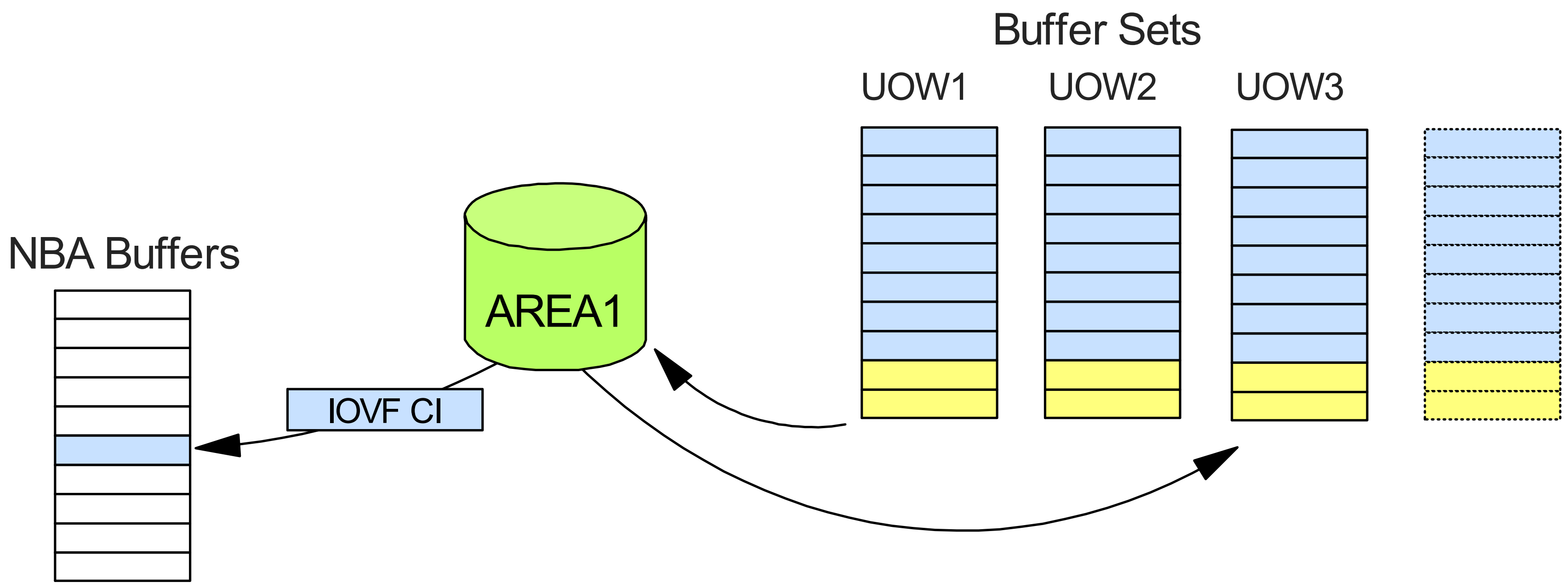
Restrictions

- ► Only one HSSP or Online Utility active in Area at a time
  - – Usually not an issue
- ► Program must move forward in database
  - – ROLB and FY status for backward reference
- ► Program must issue CHKP after getting GC status
  - – GC means crossing UOW boundary
  - – OK to issue calls to other PCBs

HSSP - Get Next Processing

During program execution

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



HSSP - The Up Side

Extremely fast

- ► Anticipatory reads can eliminate all synchronous I/O
  - – Following is actual example using third party product

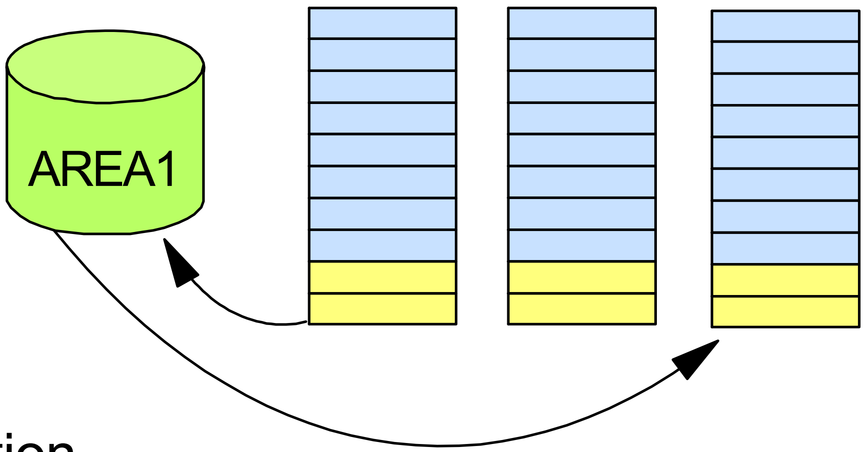
```
         CALL  LEV          STAT                     ELAPSED TIME
PCB NAME FUNC  NO.  SEGMENT CODE   CALLS   IWAITS   MEAN    MAXIMUM

ACCTDB   GN   (01) ACCTRT          26246        0     91      14425
         GNP  (02) ACCTDEP1        26246        0    147      16064
         REPL (02) ACCTDEP1         6864        0     66       3155
         GNP  (01) ACCTRT     GE  145886        0     64      59710
         ...
         ...
         DL/I PCB SUBTOTAL        278491        0     84
```

Asynchronous image copy

- ► Eliminates another scan of database

HSSP - The Down Side

Resource requirements



- Page fixes at least 3 buffer sets
  - Each buffer set holds one UOW
- Can dominate device
  - Chained reads/writes of entire UOW
  - May delay random request by transaction

Locking

- May have 3 or more UOWs locked at once
  - Current, previous, next
  - Sometimes reads farther ahead
  - Locks are EXCLUSIVE even if PROCOPT=HG
- Additional locking for online transactions
  - Each lock request preceeded by SHARE lock request for UOW
    - Can't get CI lock if HSSP holds EXCLUSIVE lock on UOW

Sequential Dependent Processing

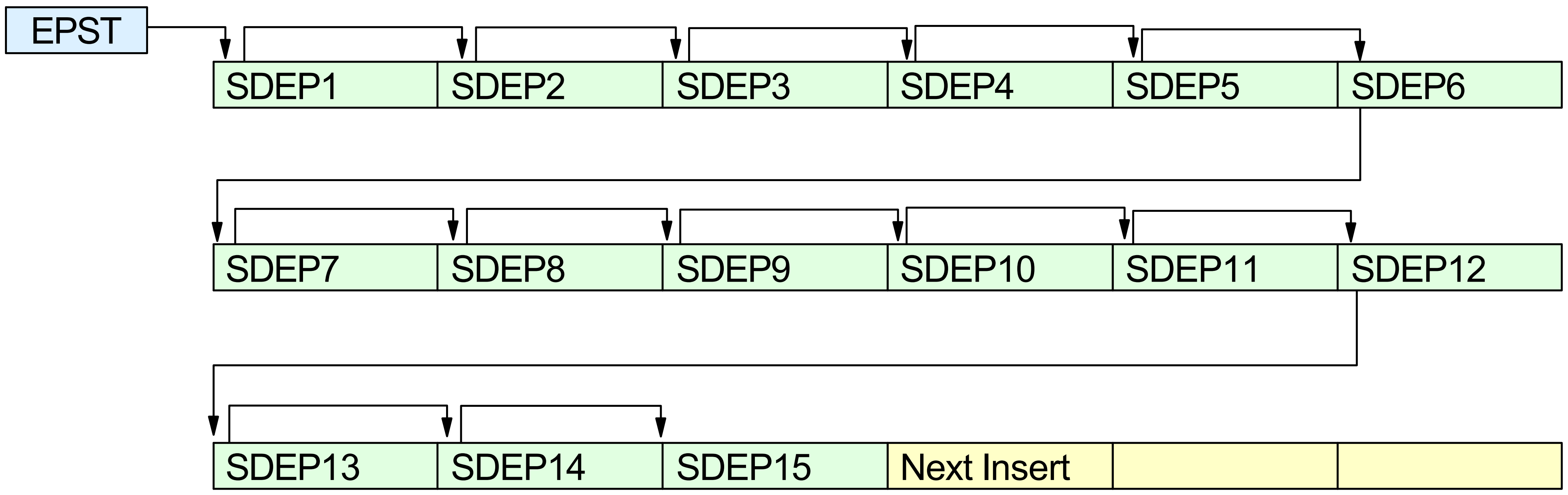
Definitions

- NBA and OBA buffers
  - SDEP inserts held in these buffers until sync time
  - Part of dependent region buffer allocation
- CSDB - Current Sequential Dependent Buffer
  - One for each area with SDEPs
  - Used at sync time to insert committed SDEPs
  - When CSDB is full, get another from PACI set
- PACIs - Preallocated CIs
  - CIs preallocated for each area with SDEPs
  - Used at sync time when CSDB is filled
  - Do not occupy buffers until actually used as CSDB
  - *Requires I/O* to preallocate CIs
    - Get Area Lock - Read 2nd CI (DMAC)
    - Allocate next PACI set (update DMAC) - Logically log allocation
    - CHKW log - Write 2nd CI - Release Area Lock

Sequential Dependent Processing ...

At call time

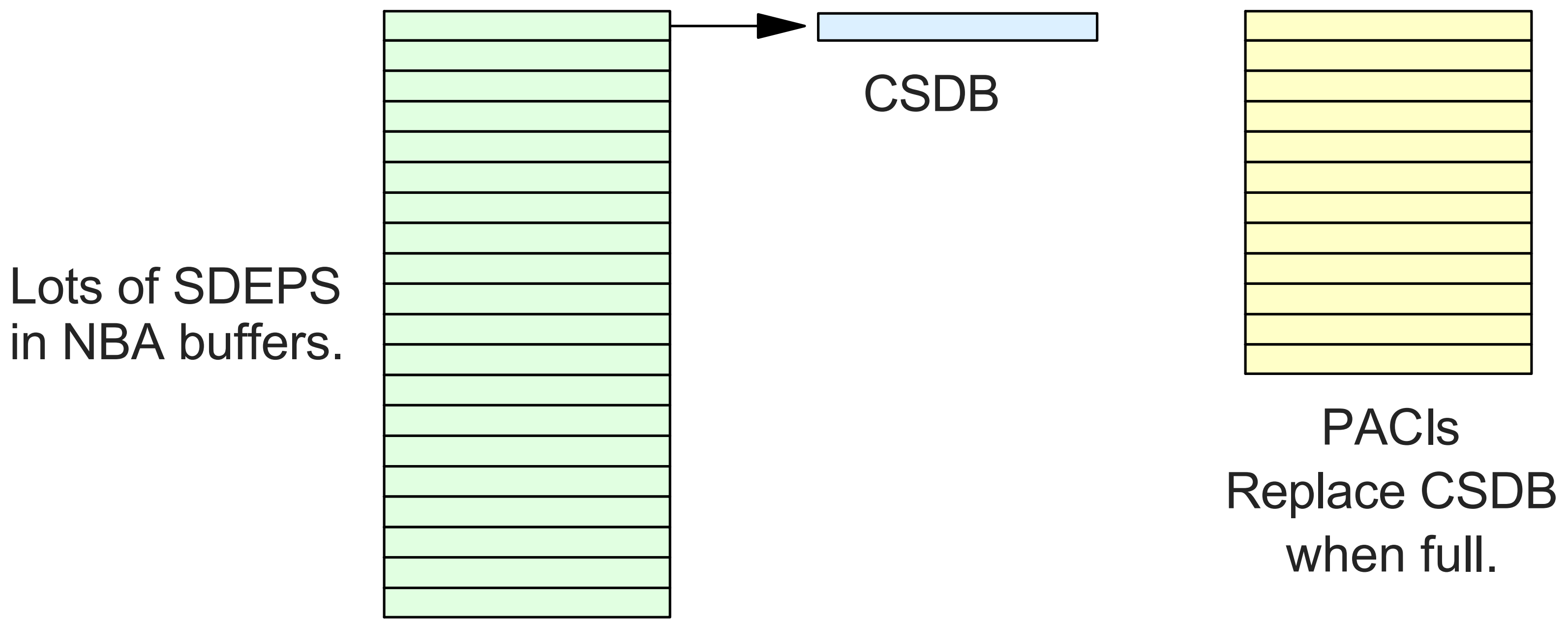
- SDEP inserts placed into program's NBA buffers
  - Inserted segments chained off EPST control block
- FP keeps track of how many CIs will be needed at sync time
  - Allocate additional PACIs if necessary
    - *Asynchronous* process at call time



Sequential Dependent Processing ...

At sync point time

- SDEPs moved from NBA buffers to CSDB
- When CSDB full
  - Schedule output thread and use next PACI
- Allocate additional PACIs if necessary
  - *Synchronous* process at sync time



PACI Allocation Process

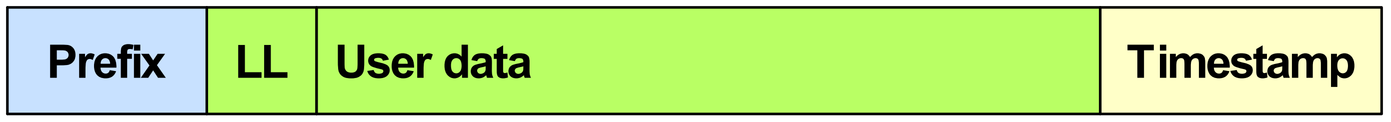
When SDEP CIs are required for insert activity

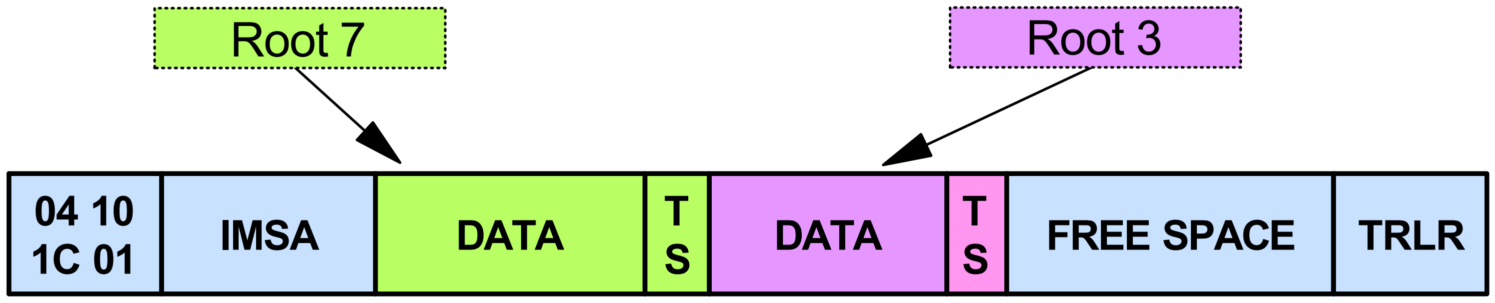
- Get Area Lock (exclusive) to serialize allocation
- Read 2nd CI to find next CI to allocate (DMACNXTS)
- Allocate one or more CIs for local IMS (PACIs)
  - Initial allocation is 3 CIs
  - Subsequent allocations based on usage rate
- Update allocation cursor (DMACNXTS)
- Create allocation x'5957' log record and write to DASD
  - Identifies allocated CIs and updated DMACNXTS
- Write 2nd CI back to DASD with updated allocation cursor (DMACNXTS)
- Release Area Lock

SDEP Utilities

Timestamps

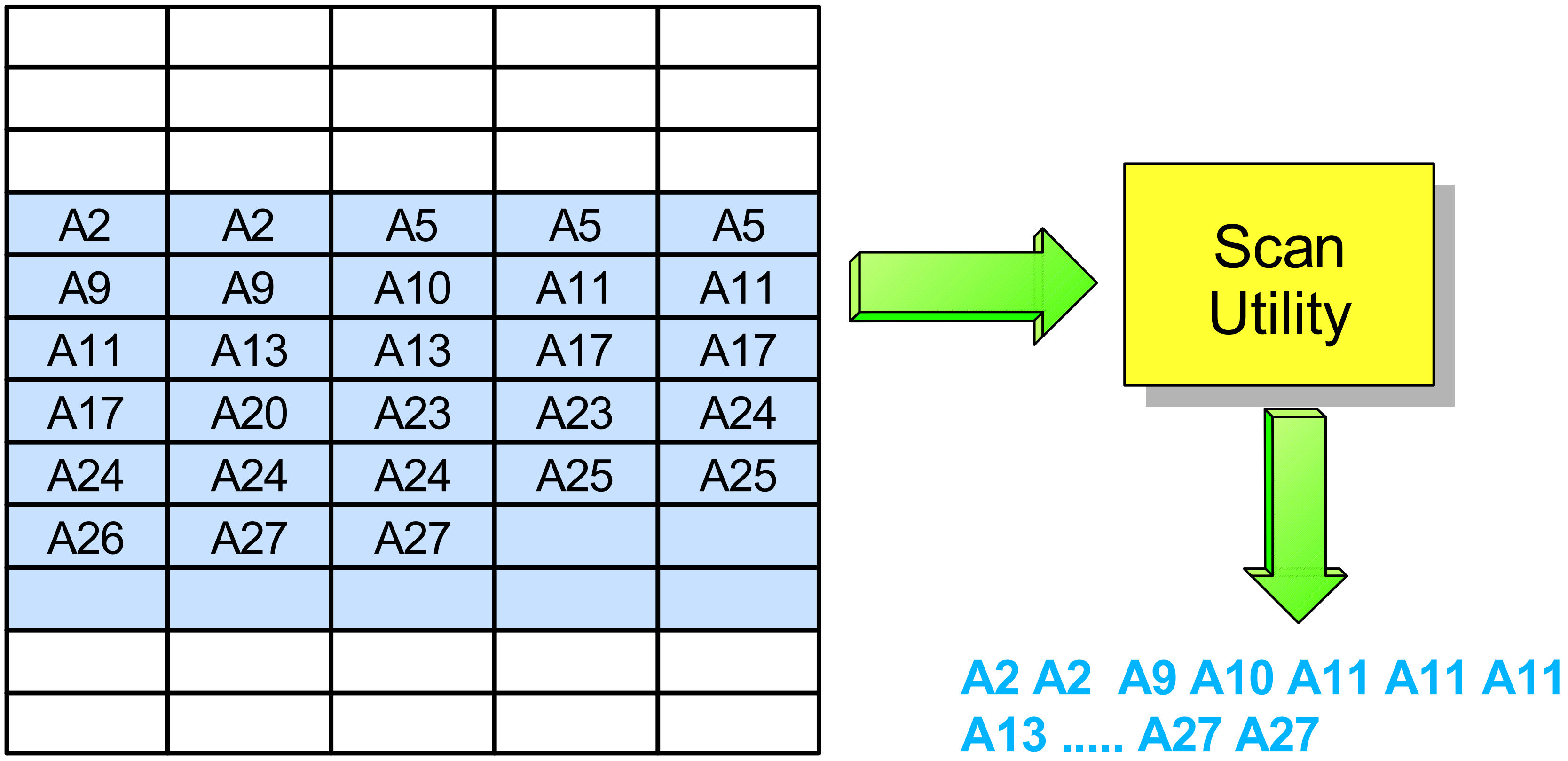
- All SDEPs (V6+) have timestamps
- All SDEPs inserted during same sync interval have same T/S





- Scan and Delete ranges are based on these timestamps
  - Can override with V5COMP control card

Non-Data Sharing Example

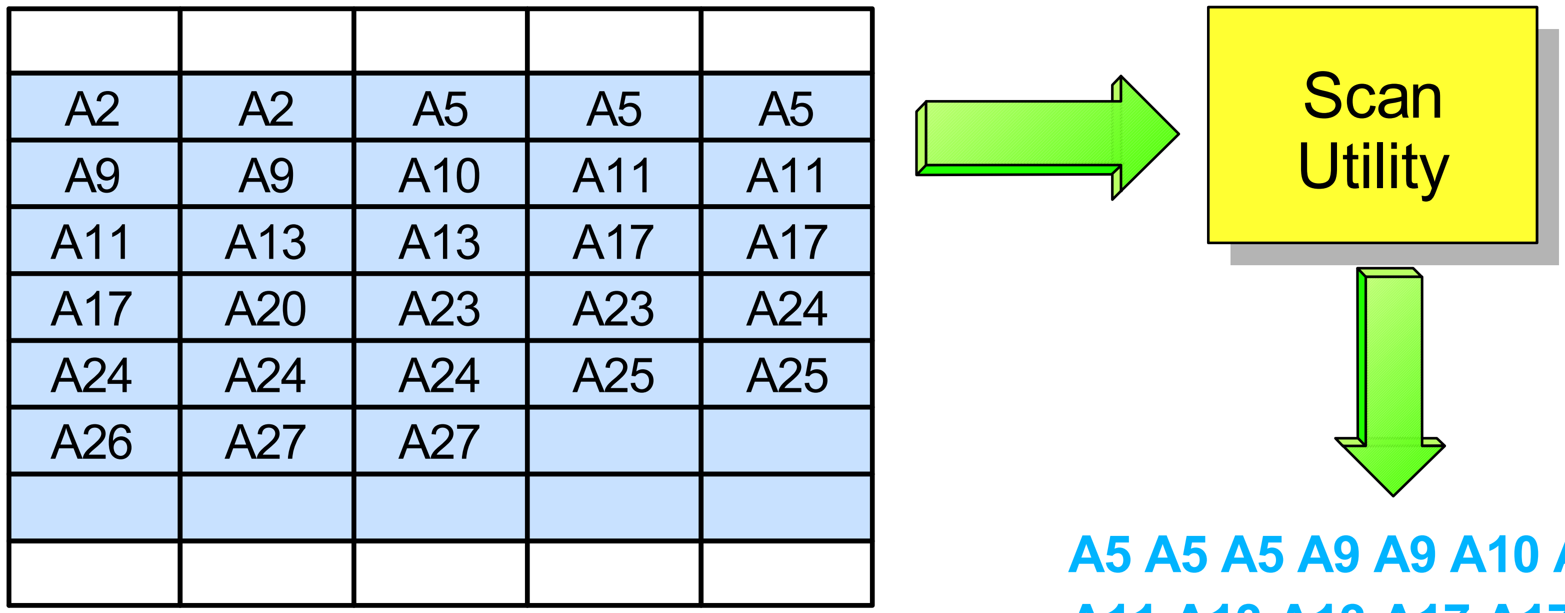


A2	A2	A5	A5	A5
A9	A9	A10	A11	A11
A11	A13	A13	A17	A17
A17	A20	A23	A23	A24
A24	A24	A24	A25	A25
A26	A27	A27		

A2 A2 A9 A10 A11 A11 A11 A13 ..... A27 A27

- ★ **Full Scan**
  - ✓ **Scan and return all SDEPs**

Non-Data Sharing Example ...

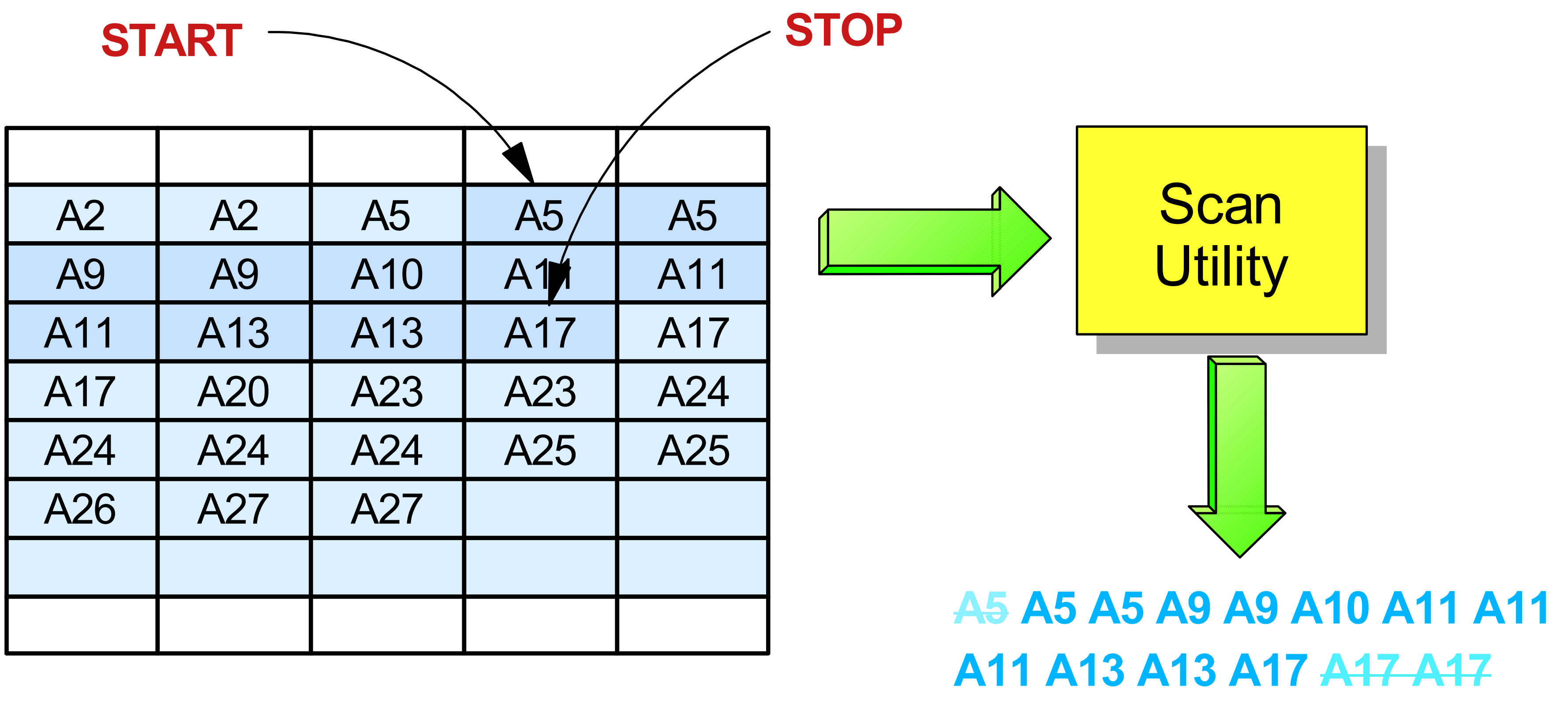


A2	A2	A5	A5	A5
A9	A9	A10	A11	A11
A11	A13	A13	A17	A17
A17	A20	A23	A23	A24
A24	A24	A24	A25	A25
A26	A27	A27		

A5 A5 A5 A9 A9 A10 A11 A11 A11 A13 A13 A17 A17 A17

- ★ **Partial Scan**
  - ✓ **Scan and return all SDEPs between Time 5 and Time 17**
- ★ **Utility reads all SDEP CIs and discards those not between Time 5 and Time 17**
  - ✓ **Necessary in data sharing environment**

Non-Data Sharing Example (V5COMP)

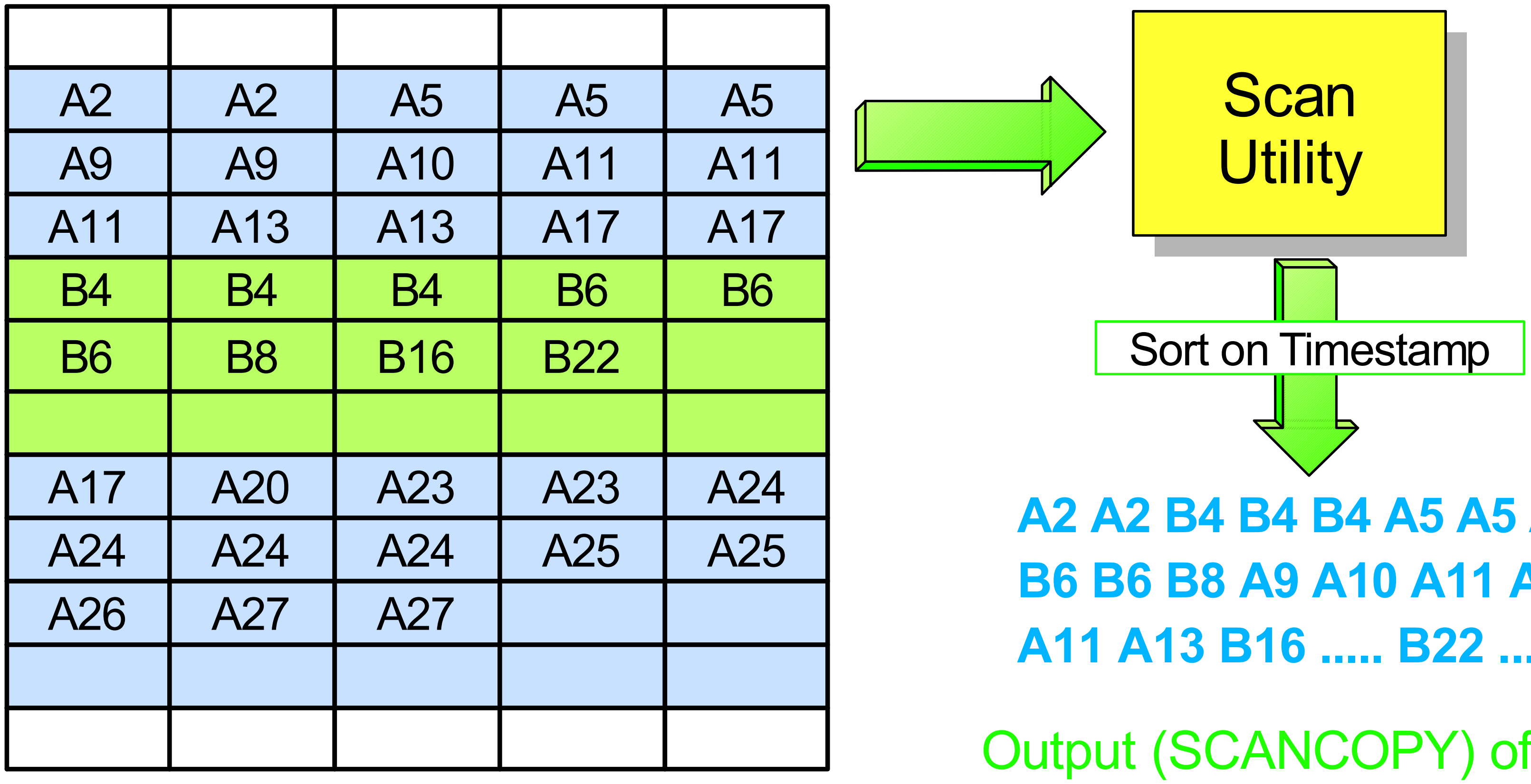


A2	A2	A5	A5	A5
A9	A9	A10	A11	A11
A11	A13	A13	A17	A17
A17	A20	A23	A23	A24
A24	A24	A24	A25	A25
A26	A27	A27		

~~A5~~ A5 A5 A9 A9 A10 A11 A11 A11 A13 A13 A17 ~~A17 A17~~

- ★ **Partial Scan with V5COMP control card**
  - ✓ **Must identify *exact segment* where to start and stop**
- ★ **Utility reads only CIs between START and STOP**
  - ✓ **Returns all segments between START and STOP**

Data Sharing Example



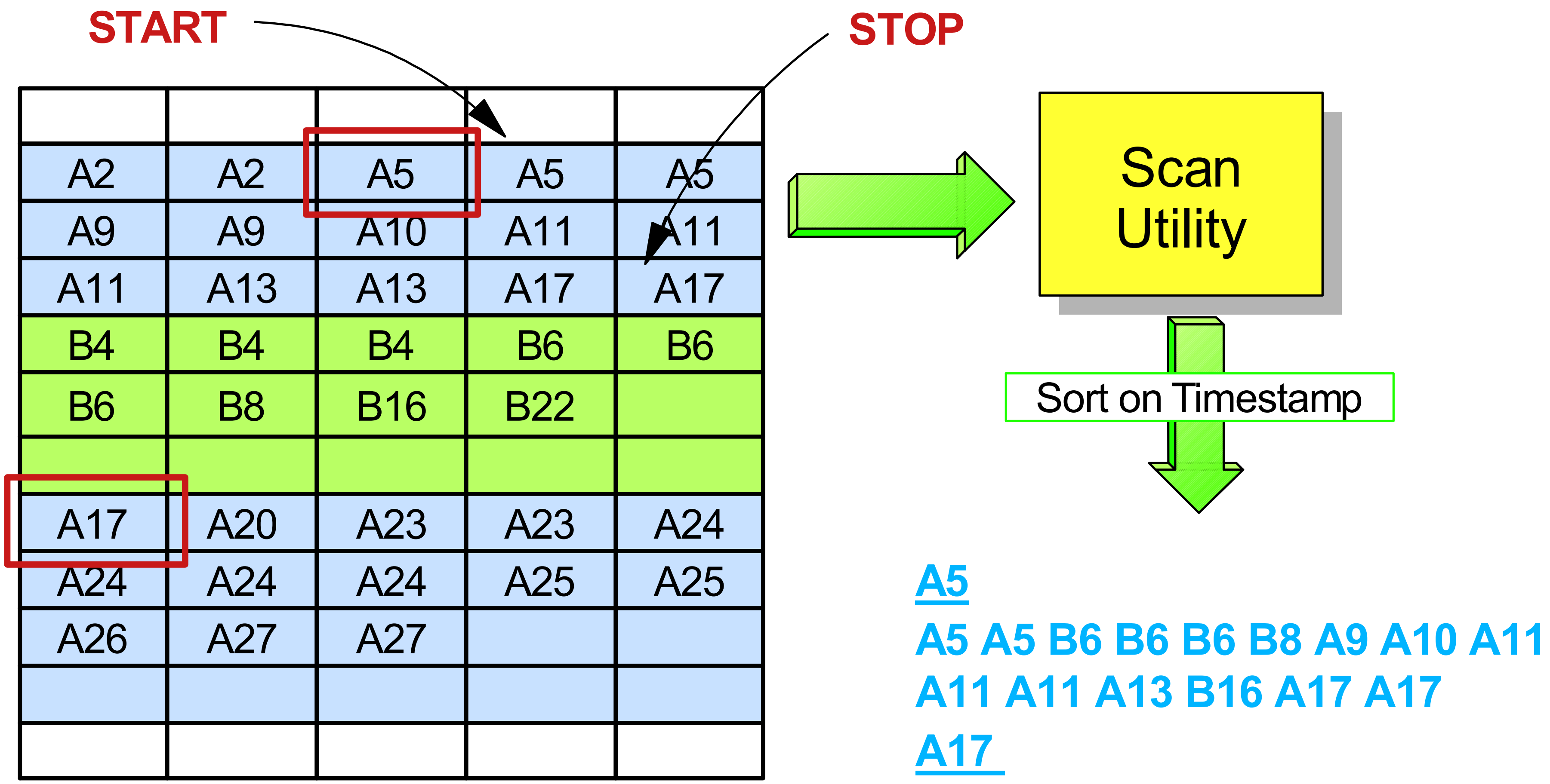
A2	A2	A5	A5	A5
A9	A9	A10	A11	A11
A11	A13	A13	A17	A17
B4	B4	B4	B6	B6
B6	B8	B16	B22	
A17	A20	A23	A23	A24
A24	A24	A24	A25	A25
A26	A27	A27		

A2 A2 B4 B4 B4 A5 A5 A5 B6 B6 B6 B8 A9 A10 A11 A11 A11 A13 B16 ..... B22 .... A27

Output (SCANCOPY) of Scan Utility is ***sorted by timestamp***, producing a single, merged sequential file. Can override by specifying **NOSORT** control card.

- ★ **Full Scan**
  - ✓ **Scan and return all SDEPs**

Data Sharing Example ...



- ★ **Partial Scan**
  - ✓ **Time 5 thru Time 17**
- ★ **Read all SDEP CIs**
  - ✓ **Discard SDEP if T/S not between Time 5 and Time 17**

SDEP Performance Considerations

Don't insert too many SDEPs per sync interval

- Extends SDEP chains while inserting (longer path lengths)
- Uses up PACIs too quickly
  - Can't keep up with asynchronous preallocation
  - Synchronous preallocation required at sync point time

If doing mass inserts

- Use multiple areas to help PACI allocation process

Use V5COMP when not data sharing

- Makes utilities function exactly as in V5 and earlier
- Only reads CIs between START and STOP control cards
- Be very careful if using V5COMP with data sharing
  - May not get expected results

Review

DEDB performance is a function of

- Pathlength
  - Usually not an issue and little you can do about it
- Lock contention
  - Same considerations as full function except lock is at CI level
  - Watch out for contention with output threads
- Read I/O
  - Writes are asynchronous
- Logging
  - Log records can be generated in large spurts at sync point time
  - Better performance also means generating log records faster
  - May need more OLDS buffers to handle spikes
- Buffer management
  - Don't run out of buffers (U1033 or FR status)
  - No look-aside buffering

Review ...

DEDB performance can be improved using

- Space management
  - Can optimize for random or sequential processing
    - UOW concept keeps related data together
  - Use Areas for parallel processing by BMPs
- HSSP
  - UOW locking
  - Chained reads
  - Anticipatory reads
- VSO
  - Eliminates all synchronous I/O and greatly reduces asynchronous I/O
  - Can be used as alternative to look-aside buffering

Review ...

SDEP processing is an animal of its own

- Avoid mass inserts in one sync interval (if possible)
- Use many areas for mass inserts
- Be judicious about Scan and Delete START, STOP, V5COMP

Fast Path Tools

Fast Path Basic Tools (5655-E30)

- ► DEDB Unload/Reload
- ► DEDB Pointer Checker
- ► DEDB Tuning Aid

Fast Path Online Tools (5655-E31)

- ► DEDB Online Pointer Checker
- ► DEDB Online Extract

Fast Path Log Analysis Utility (DBFULTA0)

- ► Detail Listing of Exception Transactions
- ► Overall Summary of Reource Usage and Contentions
- ► Several others

Fast Path Tools ...

IMS Performance Analyzer

- Log based reports
  - Uses IMS logs
    - Fast path puts lots of statistics in its log records
  - Produces reports similar to those from DBFULTA0 and IMSPARS
- Monitor based reports
  - Uses IMS Monitor data set
    - IMS V7 added fast path records to the IMS Monitor
    - Requires IMS PA to create reports using these records
  - Produces reports similar to those from IMS Monitor print program and IMSASAP

RETURN TO INDEX