

Unit 3 – Transaction Scheduling / Execution & IMS DB Processing

What this unit is about

After IMS has completed the processes related to classifying and queuing input messages, instances of IMS transactions will remain in the message queues waiting to be scheduled. In this unit, we will examine what processing the IMS Scheduler must perform as part of selecting, and preparing, a message for processing by a Message Processing region.

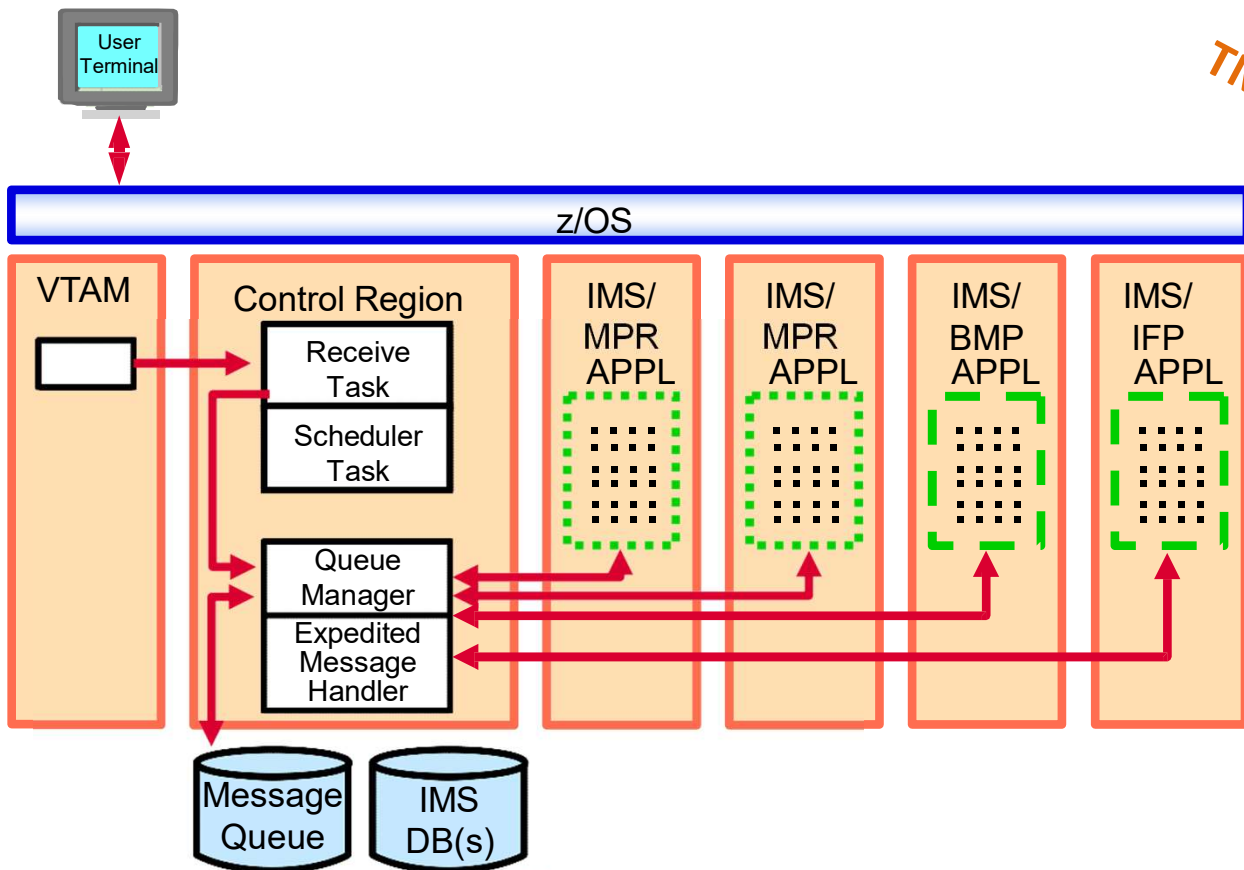
What you should be able to do

After completing this unit, you should be able to:

- Describe the Transaction attributes of Class and Priority and how the IMS Scheduler uses these attributes to select transactions for processing
- Understand other Transaction attributes, and programming techniques, that can be used to improve the effectiveness of IMS Scheduling
- List the major Scheduler-related storage pools and the process of loading the required scheduler-related control blocks into these pools
- Understand the options available and how to specify IMS database-related buffers

IMS/TM Tran/MSG Execution flow

TM



KIESSLICH CONSULTING

KC110 unit 3 page 2

Notes:

An MPR (Message Processing Region) is the execution/working environment for application program.

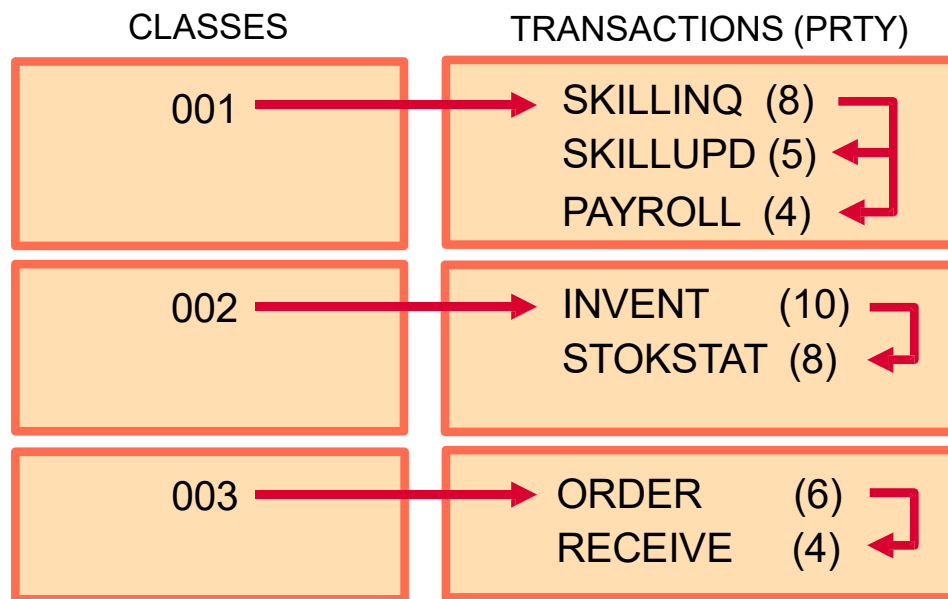
There are (usually) multiple Message Processing regions active at any given time.

Although we are showing an IFP region and the Expedited Message Handler, we will not be discussing these.

Message Classes and Priorities

- TRANSACTIONS are assigned two (2) key attributes:
 1. Class, and
 2. Priority:

TM



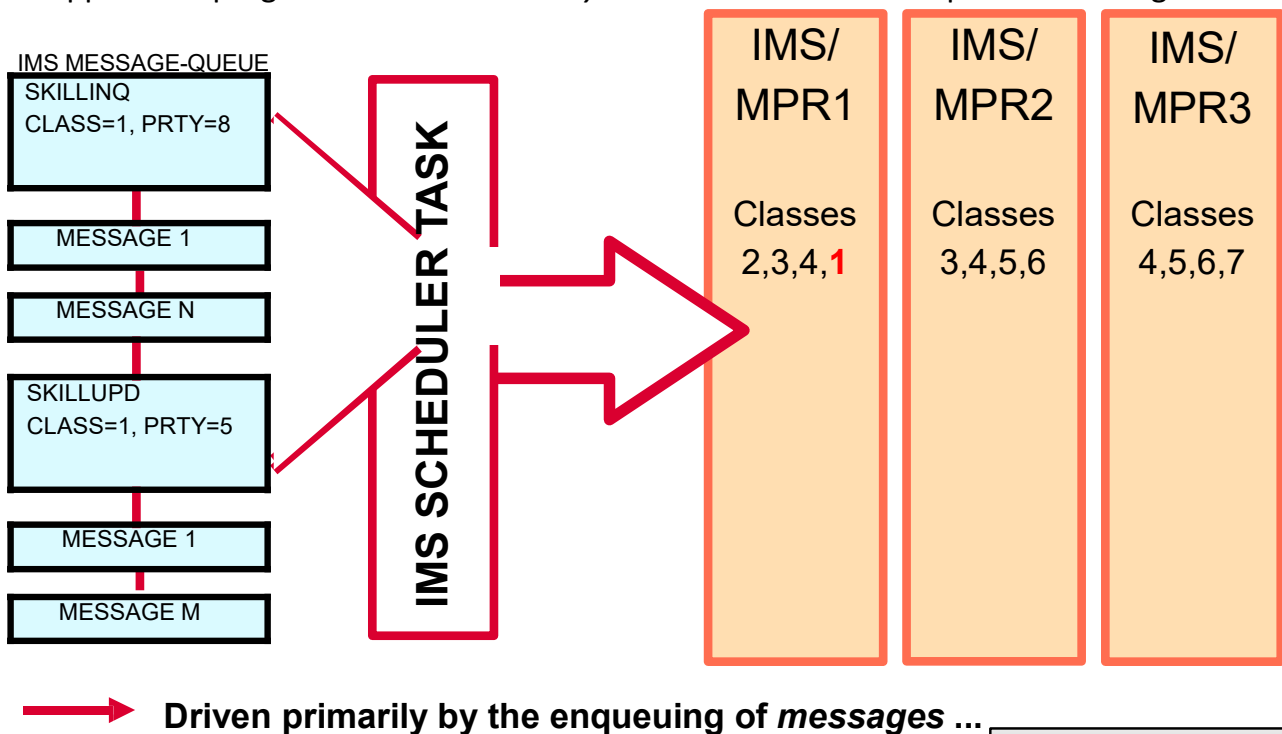
→ Transactions are *ordered by* priority within class...

- Class and priority can be changed via /ASSIGN command

IMS Message Scheduling

TM

- Scheduling is the process in which IMS matches up messages to be processed with available Message Processing Regions (MPRs)
- Application programs are *automatically* scheduled into MPRs to process messages



KC110 unit 3 page 4

Notes:

The importance of a class number (1-999) is solely dependent on the current configuration of executing MPRs.

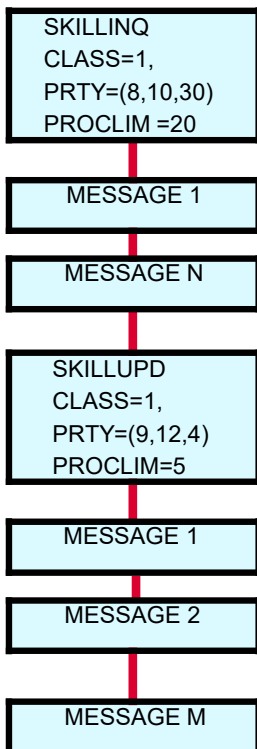
- In this example, *Class 1* seems unimportant since it can only be executed in a single MPR (MPR1)
- Also, in this example, *Class 4* seems to be the most important since it can be run in any of the three MPRs
- APPL PGMs are not “scheduled” , they are loaded (SMB points to PGM) , so the region gets control (after prime) and the very first GU IOPCB is already done 😊 - and then region controller BAKR to program controller ! With parm according defined / assigned PGMNAME (load / fetch)

Message Queue *Limit Priority*

TM

- Additional factors affecting scheduling

IMS MESSAGE-QUEUE



	TIME 1	TIME 2	TIME 3
Normal Prio			
LIMIT PRIORITY	8	10	8
LIMIT COUNT	30	30	30
CURRENT QUEUE #	8	31	11
PROCESSING LIMIT	20	20	20
NORMAL PRIORITY			
LIMIT PRIORITY	9	9	9
LIMIT COUNT	12	12	12
CURRENT QUEUE #	4	4	4
CURRENT QUEUE #	2	1	3
PROCESSING LIMIT	5	5	5

Notes:

The purpose of Limit Priority is to encourage the Scheduler to select transactions that are queuing instead of others that have a higher *Normal Priority*.

In this example, at Time 2, the Priority of SKILLINQ is set to 10 instead of the normal 8 in response to queuing.

The Scheduling decision (what transaction to process) for an MPR is made when it is available (not processing):

- After scheduling has completed and processing begins, the MPR will continue to process the selected transaction even if more important transactions are queued.
- The PROCLIM parameter permits the IMS Scheduler the opportunity to periodically re-validate that MPRs are processing the *right* work

SCHEDULING , PRTY and PROCLIM

TM

Please review by reading the IMS docs !!

- Here [\(MPP Scheduling\)](#)
- Here [\(PRTY\)](#)
- And here : [\(PROCLIM\)](#) or [HERE](#)
- Related reading: For information on scheduling options, see : [Choosing IMS options for performance](#)
- ... also here: Dave Viguers "[Scheduling](#)" summary

Program scheduling:

If the response-time breakdown data indicates large and variable input queue times, check the Region Occupancy figures.

If all message regions have high occupancy, then another message region might be required. Alternatively, it might be possible to reduce occupancy by reducing program load or program execution times. If some or all of the IMS message regions are not busy, analysis of IMS PA Transaction Transit reports by transaction and class probably shows that one transaction or class is more critically hit than others. In this case, you should review the designation of classes and the allocation of classes to regions. PROCLIM and PARLIM should be reviewed also.

Related reading: For information on scheduling options, see ...

<https://www.ibm.com/docs/en/ims/15.5.0?topic=tuning-choosing-ims-options-performance>

Programs executing as wait-for-input never show 100% occupancy even when they are in the region 100% of the time. Zero occupancy might be cause to review operator procedures, with instructions to manage the number of message regions based upon display of the queues.

The IMS PA Transaction Transit reports Graphic Summary is useful to analyze input queuing time by time of input across the entire measurement period. This can be used to discover if high input queue times result from a transient peak in transaction volumes or from a more sustained phenomenon.

DFSILTA0 can be used for the same purpose, although its output is numeric rather than graphic.

Dave's Scheduling is at the same folder here.

PARLIM:

In local env. It's the value compared against TRAN queue depth (see also MAXRGN parm at TRAN stmt),

In shared env. it's different.

The problem: IMS doesn't know the actual message queue count due to messages being queued out on the coupling facility / Shared Queue.

See new field SMBGUCNT which represents the successful consecutive message GU count - now compared with PARLIM.

(Instead of local IMS behavior : comparing the number of queued messages with the PARLIM threshold value)

PARLIM=0 is triggering always another region start and schedule (limited maybe by MAXRGN only) , but could cause a lot false schedules.

See the NOTE box here:

<https://www.ibm.com/docs/en/ims/15.5.0?topic=environments-transact-macro>

More scheduling options

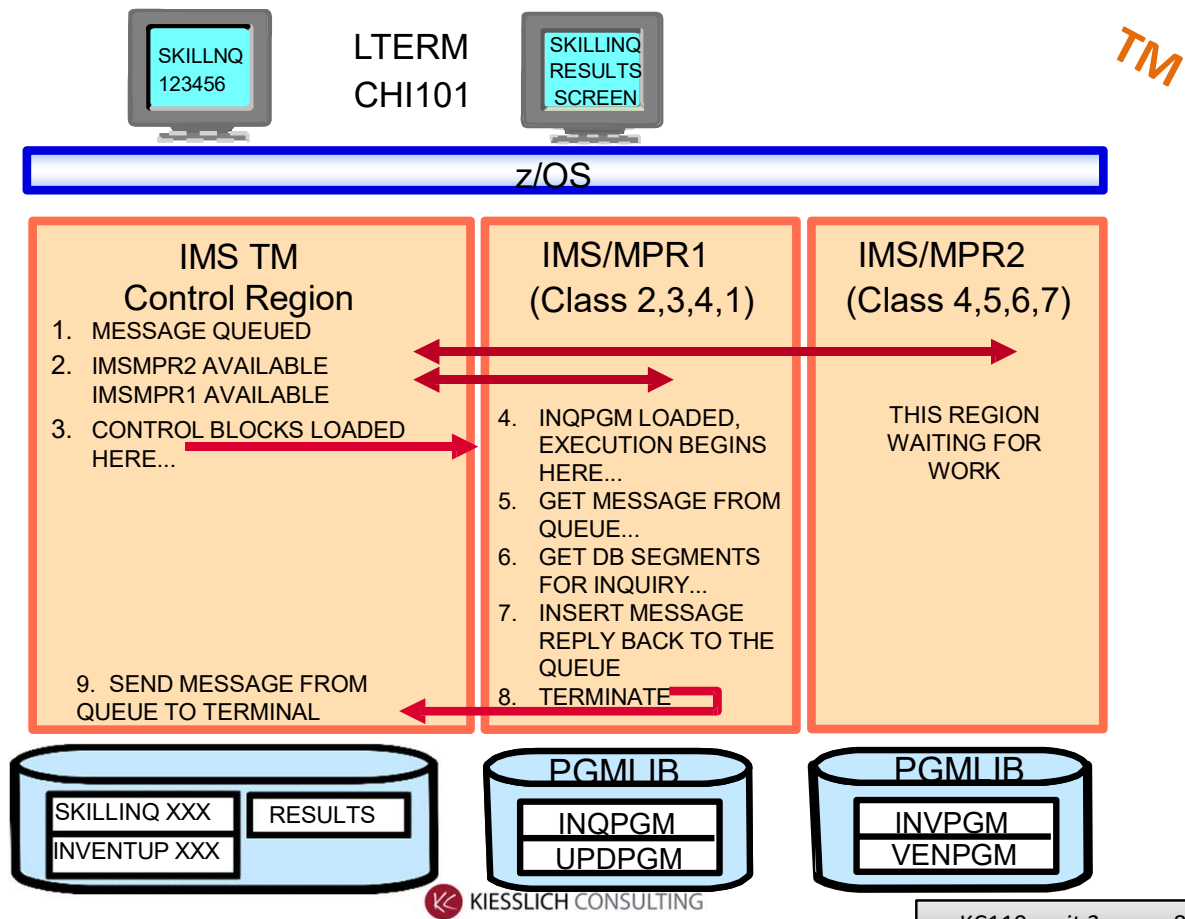
TM

- *Wait-For-Input (WFI)* transactions:
 - Allows a program to remain scheduled to wait for the next transaction, subject to PROCLIM=
 - The idea here is to allow an already scheduled MPR to wait for message from an important transaction instead of permitting the MPR to be scheduled for another transaction
- Parallel scheduling:
 - Allows the same program to be scheduled in parallel address-spaces at the same time
 - Subject to queue-count of each transaction code Upper limit maximum
 - This is not the default and may not always be desirable
 - A consequence of parallel scheduling is that transactions can not always process in FIFO sequence
- After scheduling completes, the application program load modules must be loaded before execution begins:
 - This might take a relatively long time
 - IMS *Preload* options results in loading the application programs into certain regions as the regions initialize instead of after scheduling

Notes:

There are many more IMS transaction attributes than are shown here. Almost all can be dynamically modified without requiring an IMS system restart.

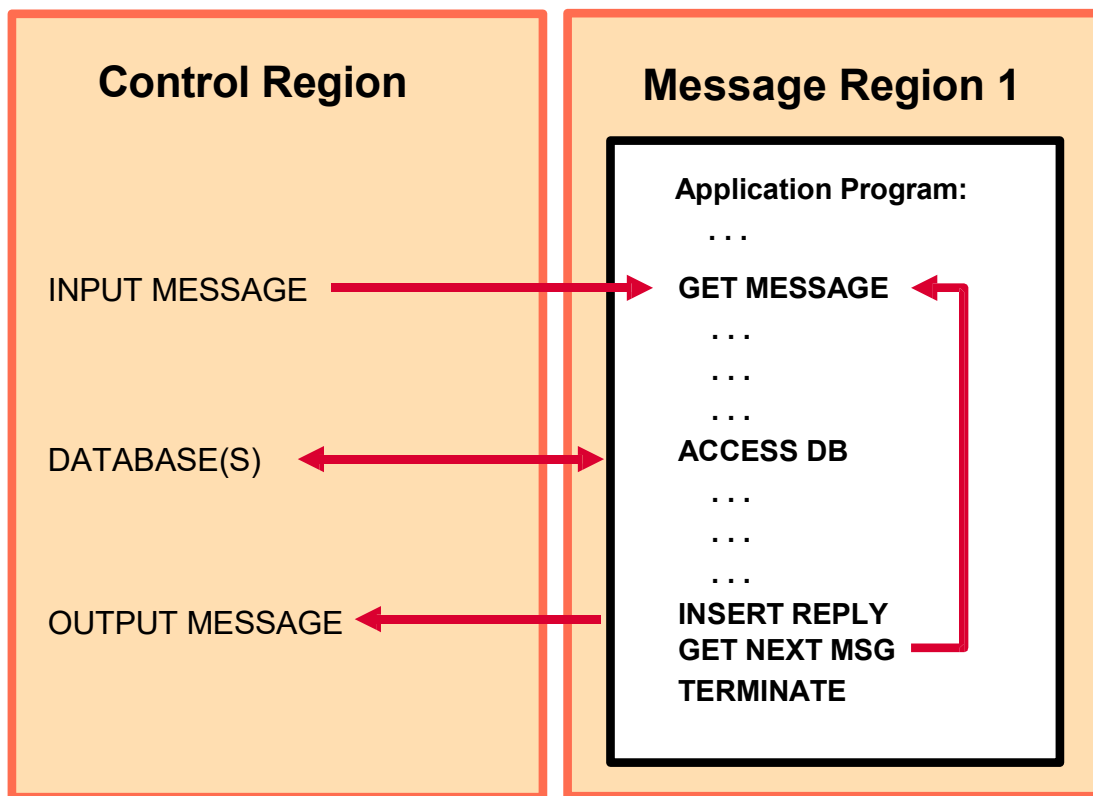
Message Processing Programs



Scheduled Transaction flow

TM

```
// EXEC PGM=DFSRRRC00, PARM='MSG,002003004001,...'
```



KISSLICH CONSULTING

KC110 unit 3 page 9

Notes:

The EXEC statement at the top of this page shows the first few parameters associated with the JCL of a Message Processing Region.

- Note the '002003004001' portion of the PARM field. This indicates that *THIS* MPR will primarily try to process class 002 transactions; if none available, process class 3 transactions; and so on.
- „The sequence of specifying the classes determines relative class priority within the message region. In the example, all class 1 messages are selected for scheduling before any class 2 messages are considered. Class numbers cannot be greater than the maximum number of classes that are specified during system definition.”

COBOL coding example

TM

READ THE INCOMING MESSAGE:

CALL 'CBLTDLI' USING GU, IOPCB, IO-AREA

CHECK THE STATUS CODE:

IF STATUS-CODE = 'QC'

END THE PROGRAM - OUT OF MESSAGES

HANDLE THE REQUEST:

BLAH, BLAH, BLAH.

SEND THE REPLY

CALL 'CBLTDLI' USING ISRT, IOPCB, IO-AREA.

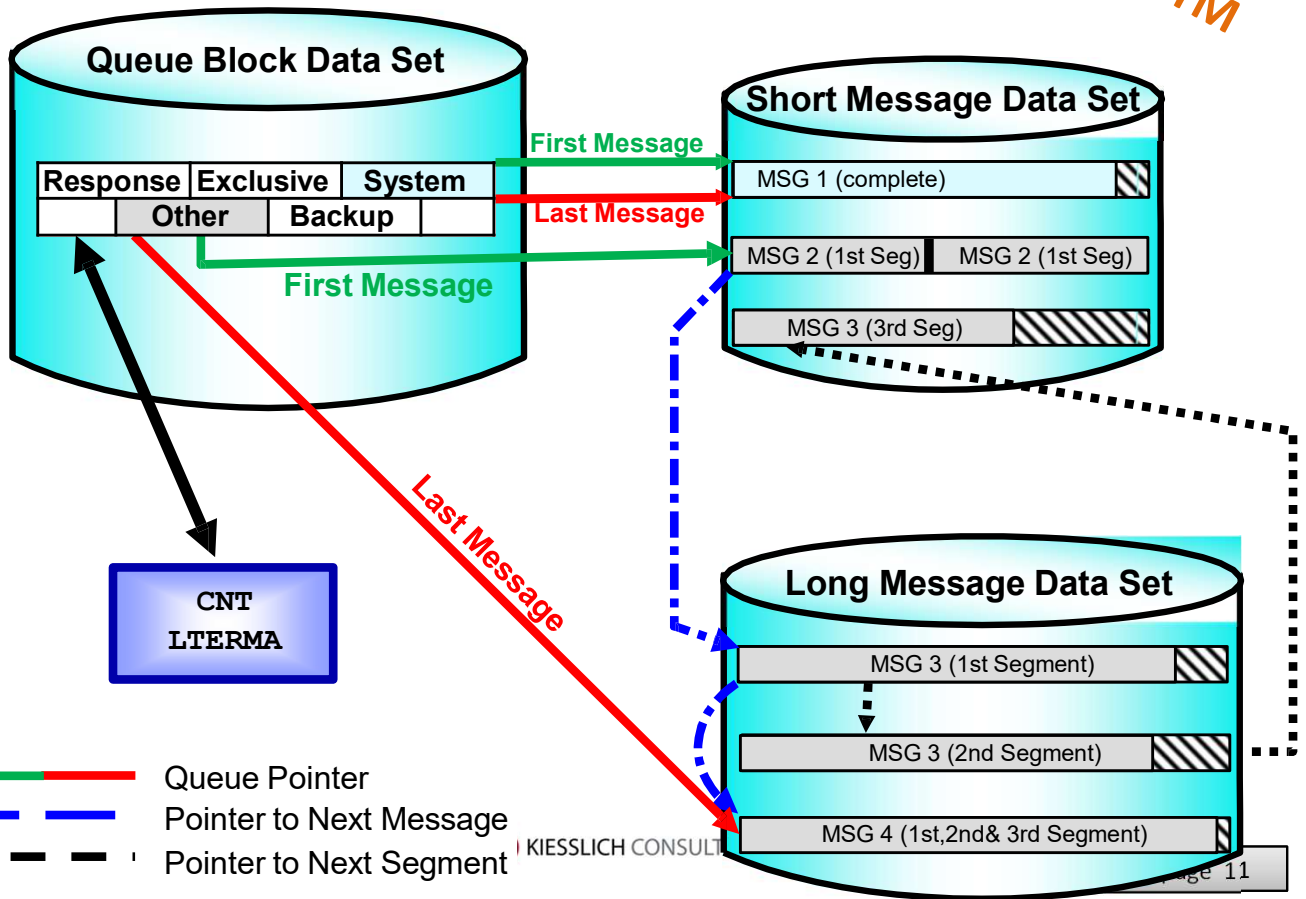
LOOP TO THE TOP TO GET THE NEXT MESSAGE.

Notes:

Note that the IMS TM calls are very similar to IMS DB calls and that unit of data transfer between IMS and the program is a segment.

Output messages queued to final destination

TM



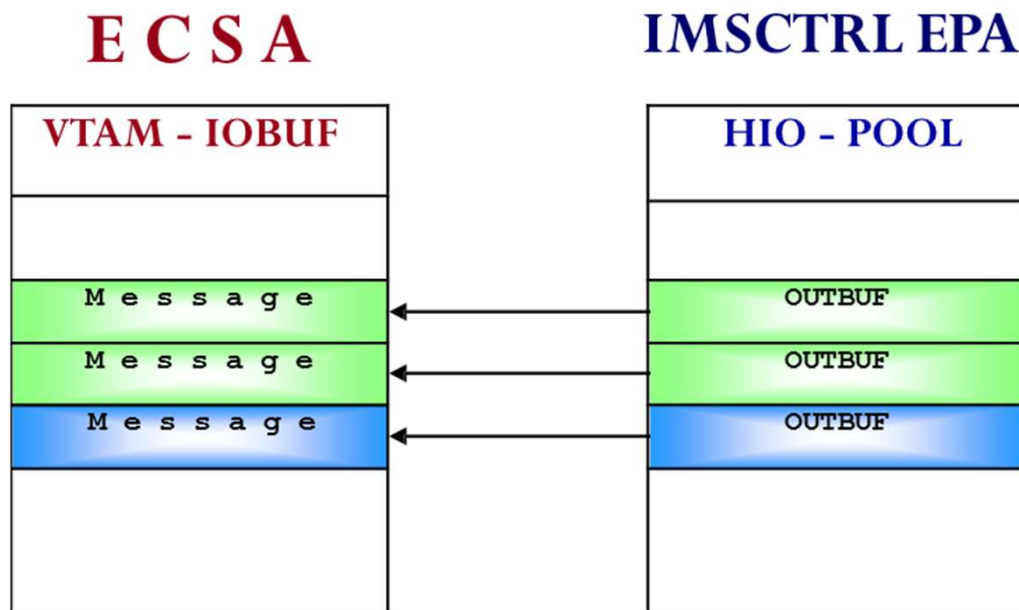
LOCAL Queue : Queue DATASETs !!!

Notes:

Here is a depiction of how multiple messages, potentially each composed of multiple message segments, are chained from the appropriate LTERM/QBLKS destination following commit.

SharedQueues : totally different !! See SharedQueues Setup Class !!!

Communications overview: Output message (1 of 2)

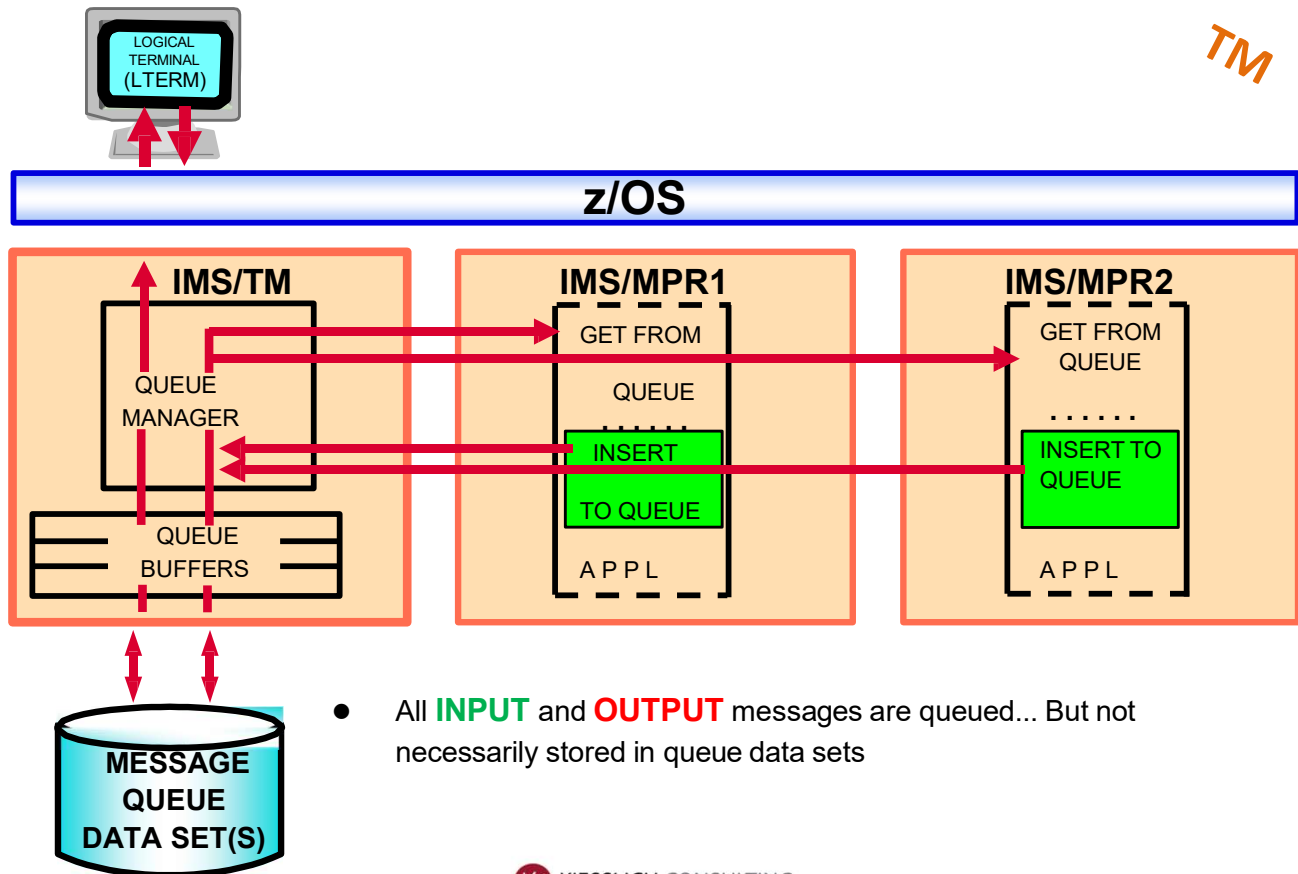


Communications overview: Output message (2 of 2)

- HIOP (High I/O Pool) resides in IMS CTRL EPA:
 - Fixed Pool storage management so additional storage can be allocated upon demand.
 - Alternate primary and secondary storage quantities can be defined in DFSSPMxx member.
 - HIOP contains primarily for buffers used for VTAM output message
 - Also used for RECANY pool, VTAM MSC Link Buffers, and MFS workarea
 - Defined in:
 - OUTBUF=size parameter of TYPE/TERMINAL macro
 - The size **must** be equal or lower than *PRIMARY UNIT Size* defined in the Request Unit Size of MODETAB
 - See topic z/OS-VTAM INTERFACE
- CIOP (Communication I/O Pool) resides in IMS CTRL PVA
 - Fixed Pool storage management so additional storage can be allocated upon demand, but is generally <100K in size
 - CIOP contains MFSTEST Buffers (/TEST MFS) which, if requested, will never be freed; also contains IMS System restart work areas

Message Queue: Output messages (1 of 2)

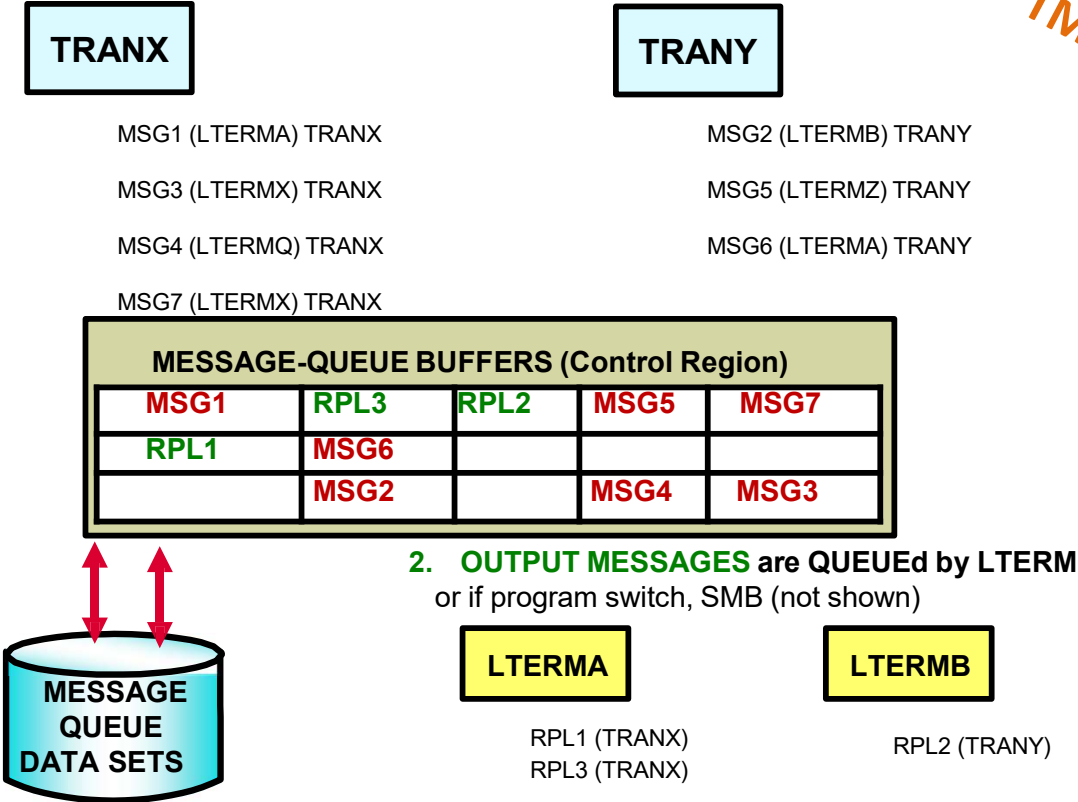
TM



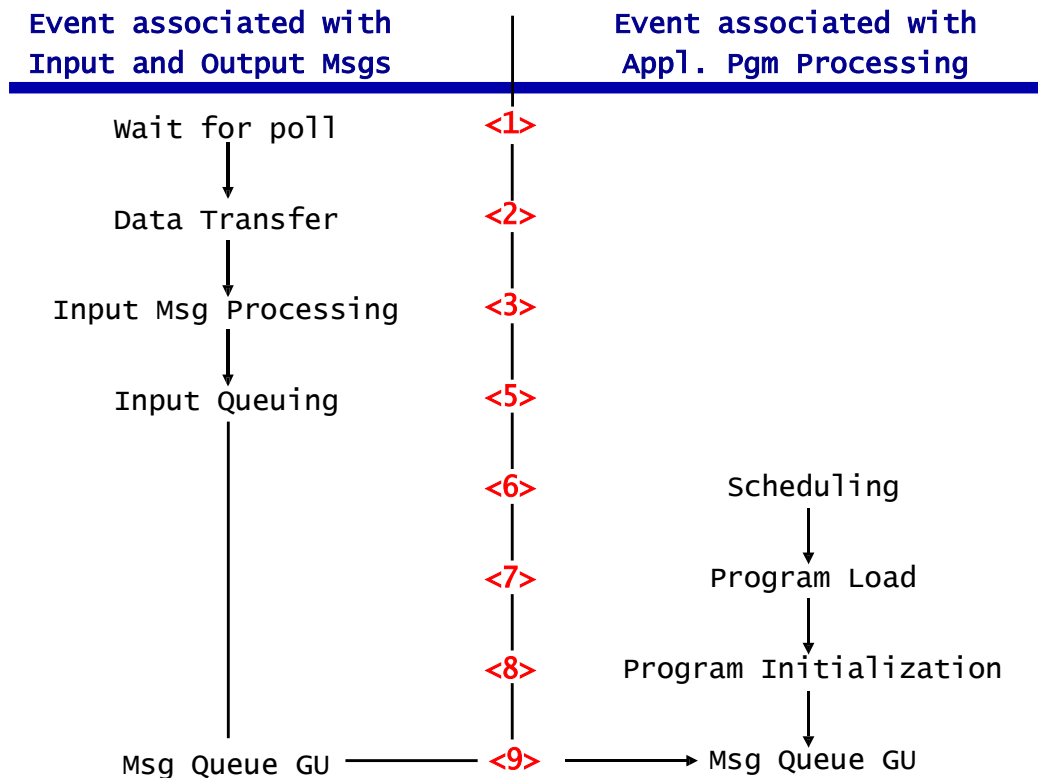
Message queue: Output messages (2 of 2)

1. **INPUT TRANSACTION** Messages are QUEUED by SMB

TM



Transaction Flow summary (1 of 2)



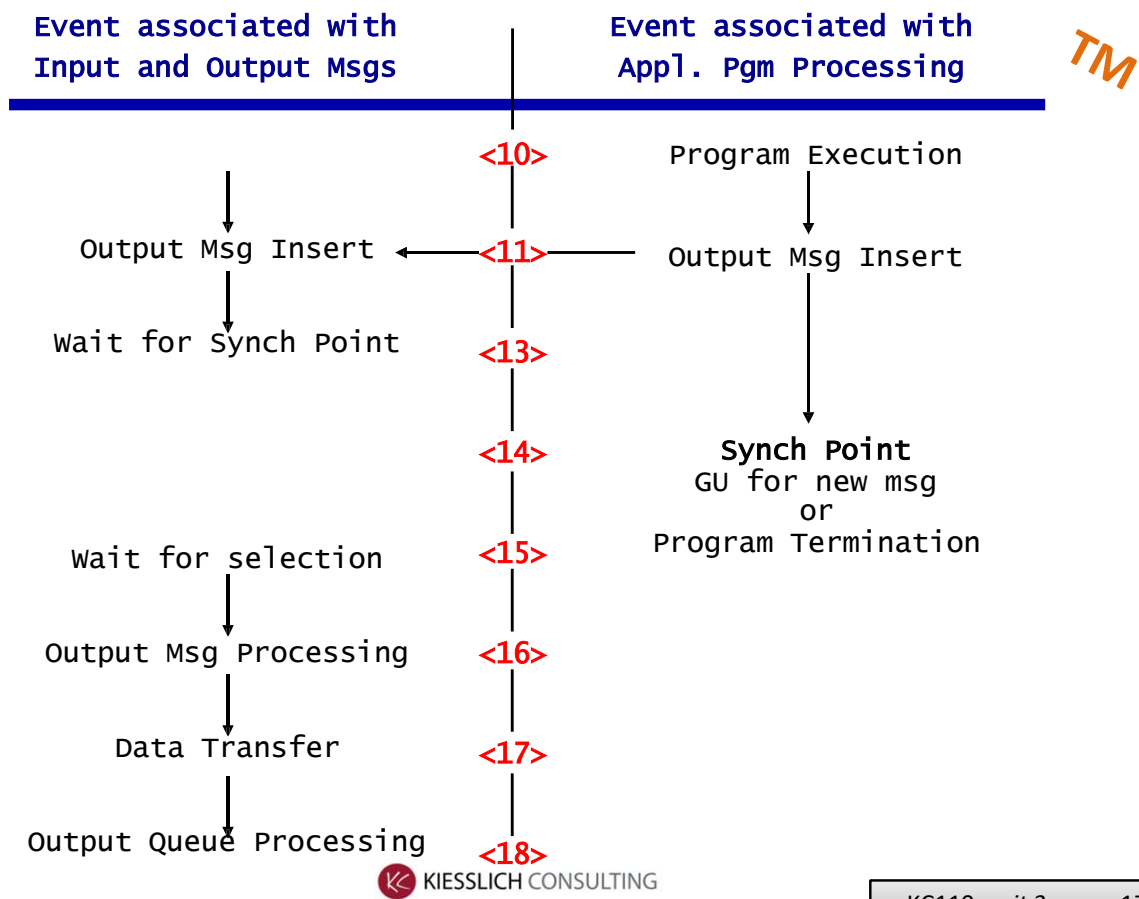
TM

Notes:

On this page and the next, we review the steps in the *life* of an IMS transaction. The important point to note here is that processing of transactions occurs autonomously in MPRs after they have been scheduled; the Control Region is responsible for receiving and delivering the input and output.

Missing <4> : in case of SharedQ is CQS put / get

Transaction Flow summary (2 of 2)



Missing <12> : in case of SharedQ is CQS put / get

IMS/TM TX : COBOL coding example

TM

```
      READ THE INCOMING MESSAGE:
CALL 'CBLTDLI' USING GU, IOPCB, IO-
AREA
      CHECK THE STATUS CODE:

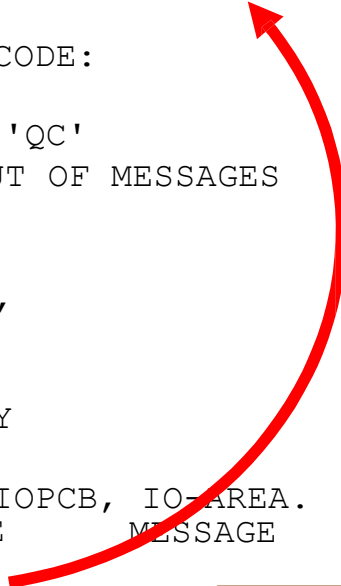
      IF STATUS-CODE = 'QC'
      END THE PROGRAM - OUT OF MESSAGES

      HANDLE THE
      REQUEST: BLAH,
      BLAH, BLAH.

      SEND THE REPLY

CALL 'CBLTDLI' USING ISRT, IOPCB, IO-AREA.
LOOP TO THE TOP TO GET THE MESSAGE
NEXT

      (ISSUE GU Call) .
      OR
      GOBACK (terminate)
```



Either a new GU Call to the IOPCB
or
Normal Program End
Trigger IMS Syncpoint Processing

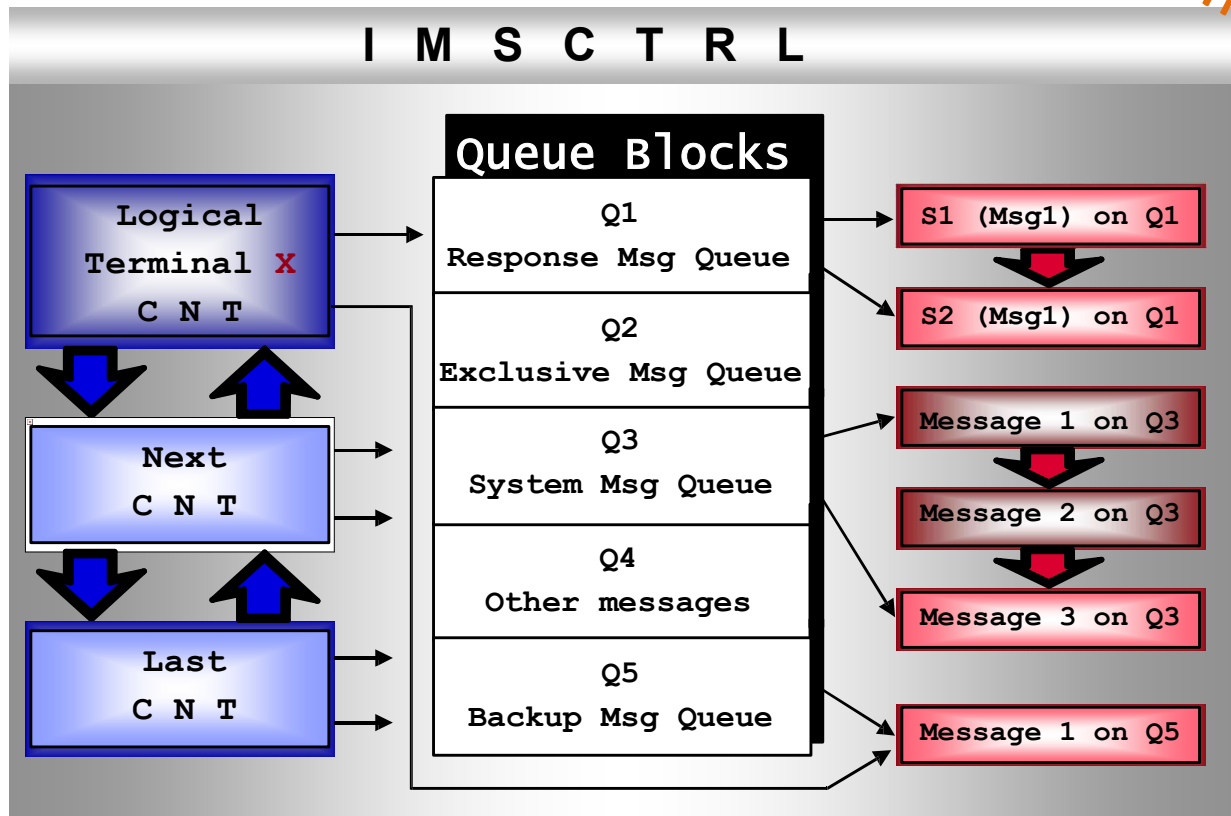
 KIESSLICH CONSULTING

KC110 unit 3 page 18

Notes:

The GU call to the IOPCB is one of the ways that an application can “**explicitly**” commit a UOW. In most cases, program end serves as an implicit commit point; several examples in which this is not true are when using FastPath and Java being two examples where an explicit commit is expected.

MSGQ Pool: Output message queuing (1 of 2)



Notes:

Messages destined for a given logical terminal are queued in one of a set of 5 chains for that terminal's QBLKS. The queuing of output messages to a QBLKS destination occurs as part of application commit processing.

On which particular output subqueue a message is queued, depends on the type of the message.

The QBLK of each logical terminal is pointed to by the LTERM's CNT control block. CNT = Communication Name Table

MSGQ Pool: Output message queuing (2 of 2)

Q1 RESPONSE QUEUE

This queue is used for response type messages, when a terminal is in response mode or conversational mode. This queue can contain only a single message.

TM

Q2 EXCLUSIVE MODE QUEUE

This queue is used for replies to a transaction from a terminal in exclusive mode (/EXC command).

Q3 SYSTEM MESSAGE QUEUE

This queue is used for system messages, which includes /BRO messages, output from the /DIS command and all system generated messages DFS.... .

Q4 OTHER MESSAGES

This queue is used for application program output, message switches and alternate PCB output.

Q5 BACKUP MESSAGE QUEUE

This queue is used to resend messages from terminals with the resend feature. It is also used for conversational replies, which are kept for the purposes of /HOLD and /RELEASE and for resending on restart. In these cases IMS has to keep the last conversational response, even though they were successfully received.

Q0 INCORE QUEUE

Note that the documentation also refers to an incore queue called Q0. All messages in this queue are sent immediately and do not use QPOOL or queue data sets.

Local Queues ☺ !

Message Processing Calls

TM

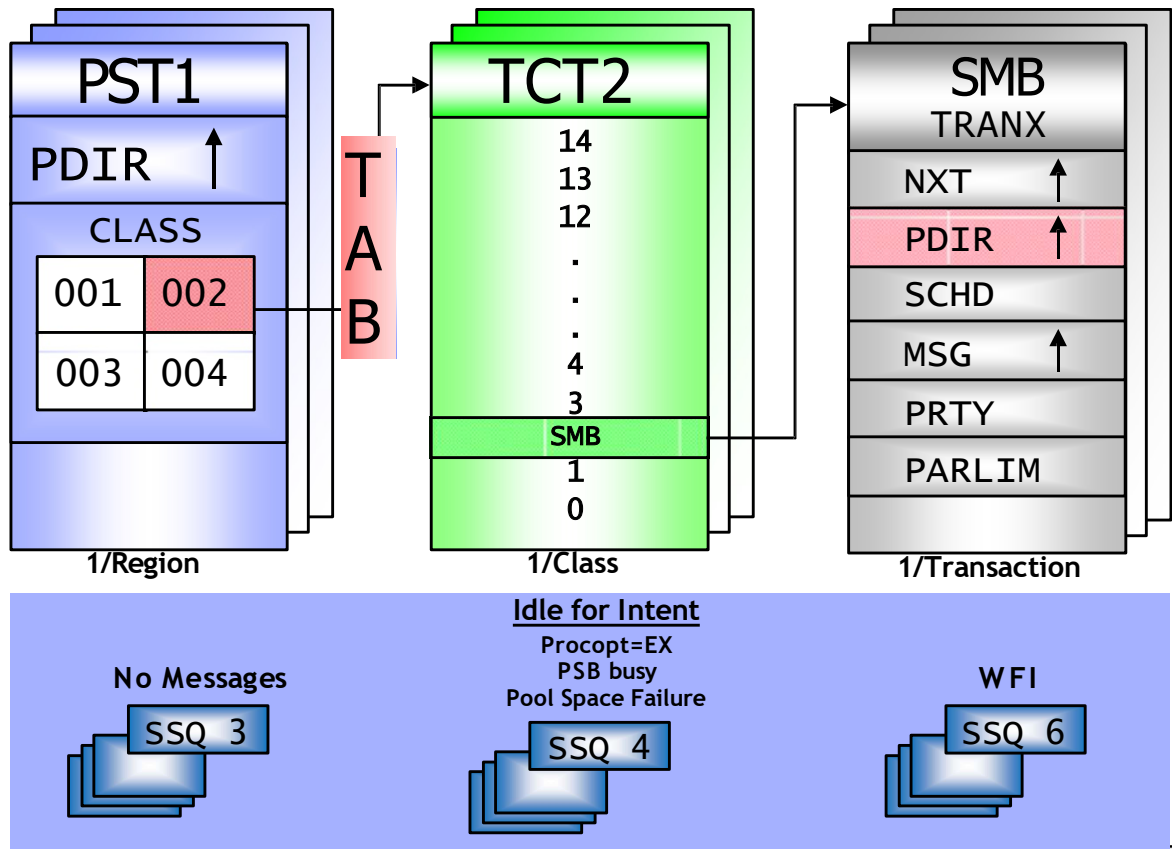
FUNCTION	PCB	CODE
RETRIEVING MESSAGES: GET UNIQUE GET NEXT	IO-PCB IO-PCB	GUbb GNbb
SENDING MESSAGES to ORIGINATING TERMINAL: INSERT	IO-PCB	ISRT
SENDING MESSAGES to ALTERNATE TERMINALS: - CHANGE - INSERT - PURGE - By EXPRESS PCB	ALT-PCB ALT-PCB ATL-PCB	CHNG ISRT PURG
CHECKPOINTING of the BATCH APPLICATION: CHECKPOINT SYNC-POINT	IO-PCB IO-PCB	CHKP SYNC

Acrord32.exe /A "page=124"

C:\Users\T450p\Desktop\WundW_final\Material\dfsapgl0_notes.pdf

Scheduling a Transaction: The *Internal* Perspective

TM



TAB – Transaction Anchor Block

IMS Transaction Schedule: Algorithm (1 of 2)

TM

- Scheduler gets control:
 - At MPR initialization
 - When a new message is enqueued on an SMB
 - At application program termination
 - If a CICS thread is released
- Schedule transaction to process in dependent regions on their request
- Select Transactions to schedule by priority within class
 - By class:
 - Assign a class to each TX code (1-999)
 - Assign 1-4 classes to each MPR
 - By priority within class:
 - Assign a priority to each TX code (0-14)
 - Optional automatic priority change (Limit Priority) when there is a large queue build-up
- Allocate and reserve resources for dependent regions/threads

IMS Transaction Schedule: Algorithm (2 of 2)

- Maintain priority control when resource not available (via SSQs) TM
- Once all resources are available *Message Priming* takes place which means:
 - Copy TX (SPA) MESSAGE into PSBW Pool in preparation for 'GU I/O-PCB'
 - Therefore, first application call **should** be 'GU I/O-PCB'
- As Scheduling completes, the MPR is activated to load then execute the application program
- Free resources when application program terminates:
 - Free scheduler-related resources after Commit or ABEND processing has finished
 - Database changes are either finalized or rolled-back and locks are released
 - The completion of application processing will cause this entire process to repeat and allow another transaction to process in the dependent region

TM



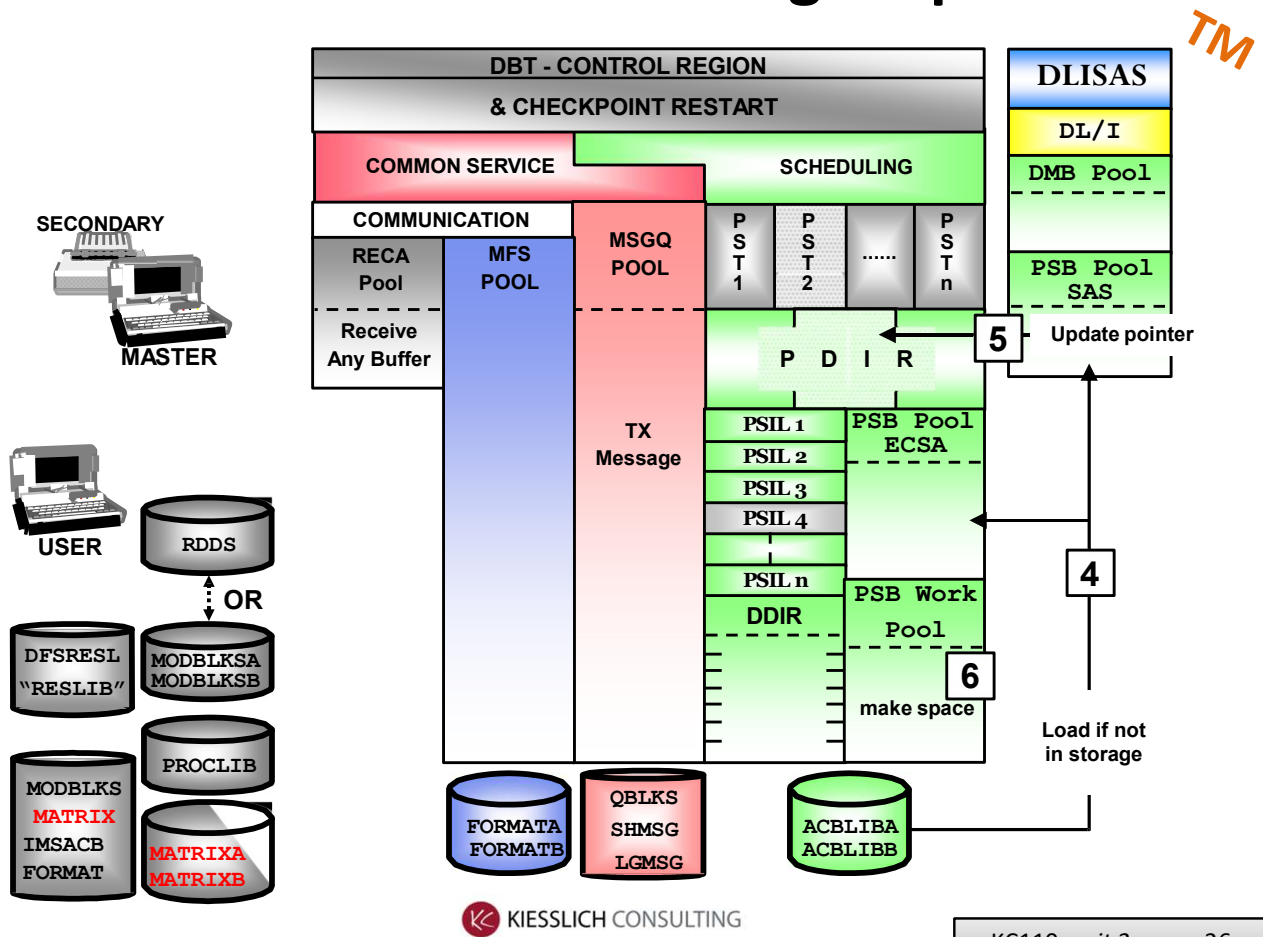
- If not:

BLR (Block Loader) gets control **[1]**.

DMBs (Data Management Block) already in storage (information in DDIR)
- If not:

BLR (Block Loader) gets control **[2]** and **[3]**.

IMS TRAN Scheduling Steps...



PSB (Program Specification Block) already in storage (information in PDIR).

If not:

- BLR (Block Loader) gets control [4] and [5].

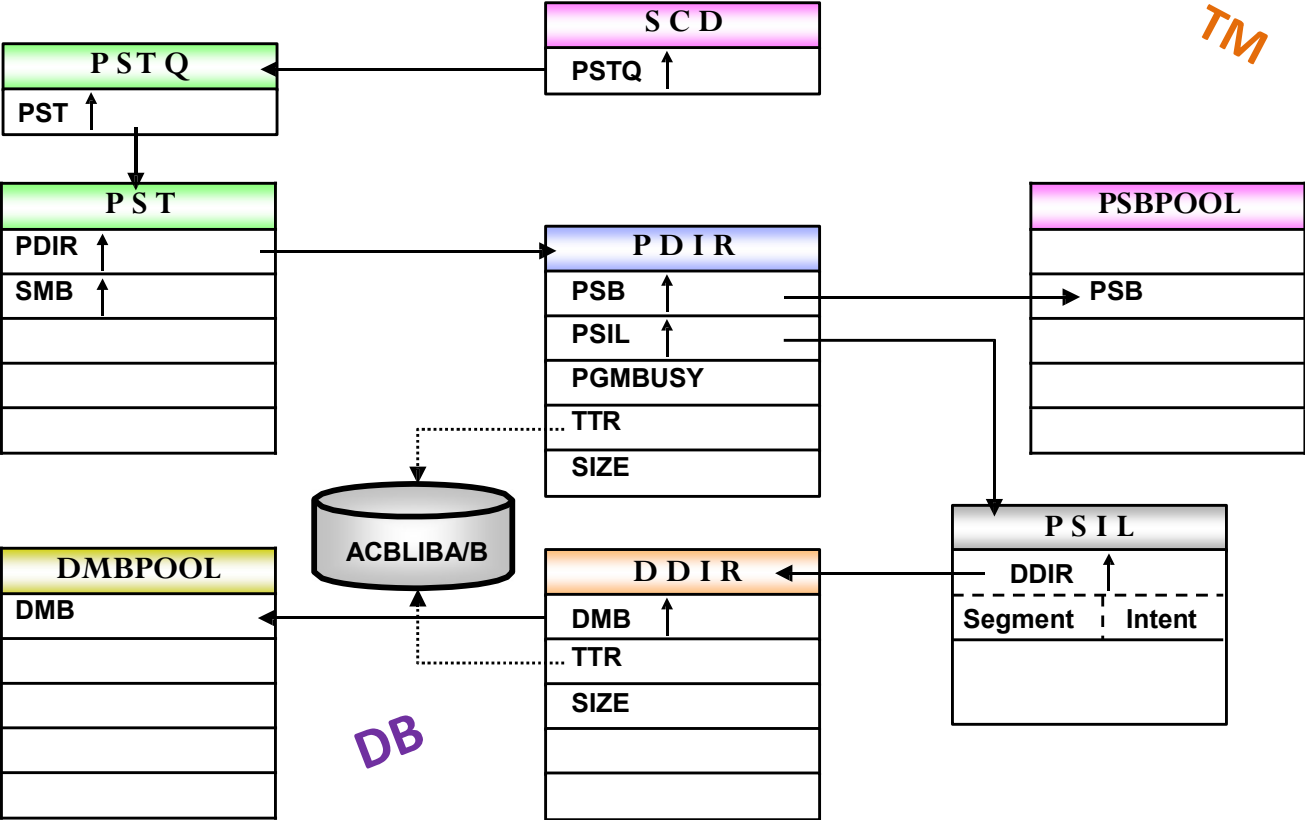
To get space in the PSB Work pool:

- BLR (Block Loader) gets control [6].
- If not: POOL SPACE FAILURE.

Program code is loaded into MPR. Databases allocated if available and currently unallocated. Databases authorized if registered with DBRC. UOR (Unit Of Recovery) started if update intent.

IMS/TM Control Block Relationship

TM



Functions of Block Loader (or Block Mover)

Get space in PSB Pool for block required

[1] Read and load PSIL (if not in pool)


- Check for database conflicts
- Get space in DMB pool for blocks required

[2] Read and load DMB (if not in pool) [4]

Read and load PSB (if not in pool) [6] Get space in PSB Work pool

- The Block Loader (runs under a dependent TCB) can acquire resources for different PSTs in parallel but only for different pools.
- Scheduling **waits** when resource not available and might eventually **fail** if space can not be allocated

TM



IMS does **n o t** require all databases to be available to allow successful scheduling (see topic "Enhanced Scheduling")

IMS Scheduling Related Pools and CBs (1 of 4)

TM

- PST
 - Partition Specification Table resides in ECSA:
 - PST is used to control each thread and region
 - Size 10K,..,15K (if IRLM and FP)
 - MAXREGN= or PST= parameter defines the minimum of PSTs
 - MAXPST= parameter defines the maximum of PSTs
- PSIL
 - PSB Segment Intent List resides in ECSA:
 - Created by ACBGEN
 - Used to check for DB conflicts (for example, PROCOPT=EX)
 - Size = PSBSIZE - (SASSIZE + CSASIZE) from ACBGEN **

**

See foil 38

IMS Scheduling Related Pools and CBs (2 of 4)

TM

- PSB:
 - PSB pool location in storage depends on LSO= value:
 - If LSO not= S, complete PSB is in ECSA
 - If LSO=S (there is a DLISAS address space), PSB is *split*
 - PSBP (JCL Parameter CSAPSB=) is in ECSA
 - DPSB (JCL Parameter DLIPSB=) is DLISAS EP
 - Space is required during application program execution and marked inactive as part of termination processing
 - If not enough free space available, the PSB cannot be scheduled
(Pool Space Failure)
 - IMS will delete inactive PSBs in pool until sufficient contiguous space is created
 - Space requirements listed in ACBGEN
- PDIR
 - PSB Directory resides in ECSA:
 - Contains information of PSB (that is, location of PSB)
 - 56 bytes per APPLCTN macro
 - Built at IMS Initialization
 - Might be updated at schedule time

IMS Scheduling Related Pools and CBs (3 of 4)

TM

- DMB:
 - DMB (Database) Pool resides in EP of DLISAS or Control Region
 - Execution time version of DBD
 - DMB= parameter defined in execution JCL
 - If not enough free space available, the PSB cannot be scheduled
(Pool Space Failure)
 - IMS will delete inactive DMBs in pool and close the associated database data sets until sufficient contiguous space is created
 - This can cause a disastrous impact to system performance
 - Space requirements listed in ACBGEN
- DDIR:
 - DMB Directory resides in ECSA
 - Contains information of DBD (that is, location of DBD)
 - 76 bytes per DATABASE macro
 - Built at IMS Initialization
 - Might be updated at schedule time

IMS Scheduling Related Pools and CBs (4 of 4)

TM

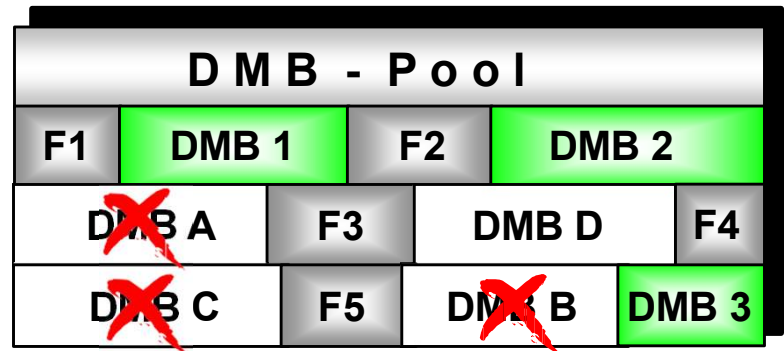
- PSBW:
 - PSB Work pool resides in ECSA
 - Inter address space work area to process DL/I requests
 - I/O area (DB segment data)
 - SSAs
 - PCB key feedback area
 - Space is required during application program execution and freed as part of termination processing. If **insufficient**, contiguous space available, the PSB cannot be scheduled (**Pool Space Failure**)
 - Space listed in ACBGEN
 - PSBW= parameter defined in BUFPOOLS macro or execution JCL
- DBWP:
 - DBWP resides in CSA
 - Work area for DL/I:
 - Call analyzer
 - Delete processing
 - Retrieve module
 - Segment compression routines
 - A DBWP space failure *pseudo abends* the transaction
 - Not really a Scheduler-related pool
 - DBWP= parameter defined in execution JCL

DMB Pool Space Handling: Non-resident pool

F3 = remembered
free space

1,...,3 = DMB busy

A,...,D = DMB inactive
A is the oldest

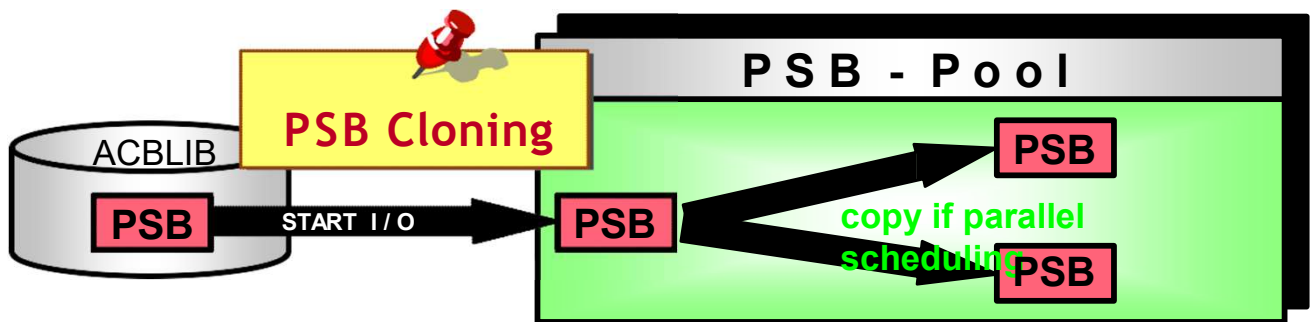


If DMB pool space needed, IMS must
close
the database (all data sets) before
its DMB can be stolen

Notes:

- Database Management Block resides in Ext Private DLI space ;
- DMB sizes (pool recommendation) is listed in ACBGEN.
- Space is managed on a variation of a LRU (Least-Recently-Used) algorithm: Oldest DMB is analyzed and its space stolen only if the steal will produce sufficient space; If not, next oldest will be examined; this will continue until a large enough unused DMB can be identified and freed; If this still does not work, IMS frees all unused from oldest to newest; if this still does not produce enough space (maybe all/most DMBs are busy), a Pool Space failure occurs.
- No pool *compression* but adjacent free space is consolidated and largest free space is remembered. Each block requires contiguous space so fragmentation is possible; stealing a DMB can have a serious performance impact since this requires that the associated DB gets closed!
- With Online Change a new/changed DMB will be loaded into the non-resident pool (even if RESIDENT is specified) until next IMS start (makes it resident again).

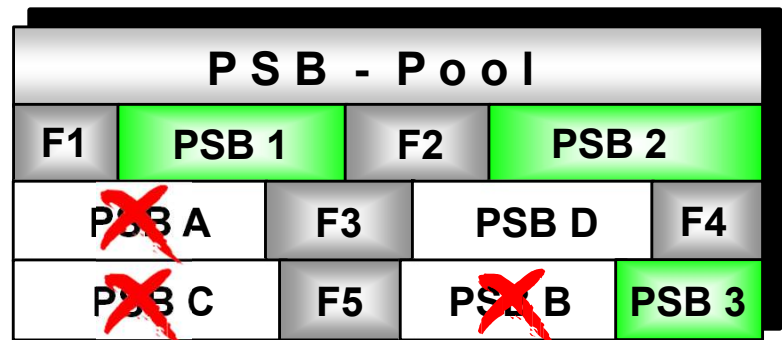
PSB Pool Space Handling: Non-resident pool



F3 = Remembered free space

1,...,3 = PSB busy

A,...,D = PSB inactive
A is the oldest



New PSB

KISSLICH CONSULTING

KC110 unit 3 page 34

Notes:

Program Specification Block resides in ECSA / EDLI.

PSB sizes are listed in ACBGEN.

PSB is required only during application program execution

If parallel scheduling, each dependent region requires its own copy of the PSB:

Additional I/O to ACBLIB might not be needed as a PSB can be cloned by copying from a busy copy elsewhere in storage.

Space is managed in the same way as was described for DMBs, but a steal will not cause as severe an impact as no data sets are closed as part of a PSB steal.

With Online Change, a new/changed PSB (as DMB) will be loaded into the non-resident PSB (as DMB) pool -even if RESIDENT is specified - until next IMS start.

PSB/DMB Pool: RESIDENT Pool (1 of 2)

- Each resident pool resides in EDLI or ECSA and is a different pool that is separate from the PSB/DMB non-resident pool
 - Size will be evaluated during IMS initialization
 - Based on size of resident PSB/DMBs; NOT on SysProg specification
 - Resident PSBs/DMBs are never deleted
 - The IMS startup parameter RES=Y creates resident pools and loads:
 - PSIL
 - PSBs defined as RESIDENT (defined in IMS Gen APPLCTN macro with the *RESIDENT* parameter or through a DRD PROGRAM definition with the *RESIDENT(Y)* attribute)
 - DMBs defined as RESIDENT (defined in IMS Gen DATABASE macro with the *RESIDENT* parameter or through a DRD DATABASE definition with the *RESIDENT(Y)* attribute)
- If parallel scheduling and the PSB is busy, it is copied from the Resident PSB pool into the NON-resident PSB pool

Notes:

The size for Resident pools is not directly specified in IMS parameter. As part of startup, IMS allocates sufficient space to store control blocks with this attribute

PSB/DMB Pool: RESIDENT Pool (2 of 2)

- With online change, a new/changed PSB will be loaded into the non-resident pool
- PSB pool until next IMS start
 - Even if RESIDENT is specified:
 - *Old* version remains in resident pool but is not used
 - This should be considered if frequent use of Online Change and RESIDENT

„Considered“ only if private storage use grows and grows ... potentially paging then (all what's not fixed)

PSB Cast Out Algorithm Redesign (V12+)

- PSB cast out (making room for a new PSB being scheduled by “casting out” an old not-in-use PSB) can be inefficient:
 - Current algorithm is fine if a single PSB can be found which, if freed, will make the requisite space in the pool for the new PSB, but...
 - If more than one PSB needs to be freed, then algorithm starts “randomly” throwing out PSBs from the pool, hoping that a big enough hole will be made for the new PSB.
 - Several latches (internal fencing) prevent scheduling of any other work during this cast out processing (since PSB pool is necessary in scheduling)
 - PSBs are chained in physical order within the pool, so:
 - For each PSB checked, also check its neighboring PSBs, neighbor’s neighbors, etc. Free a group known to make the space vs. random “Swiss cheese” freeing.
 - Also better / clever latching now (only once)

There are several inefficiencies in the way that PSB cast out processing works today (module DFSDLMP0).

There is lots of latching and unlatching around the unchaining of various database blocks that is unnecessary, because the latch header is often the same for these blocks. Rather than pay the overhead of latching and unlatching for each block unchained, the code could be changed to get the latch once, unchain all blocks that need to be under that latch, and then release the latch.

The cast out algorithm itself has a mode into which it can fall that has caused performance problems at some customers.

- “Pass 1” of the cast out algorithm scans the PSB pool looking for the oldest not-currently-in-use PSB in the pool that is large enough such that casting it out of the pool will make enough room to contain the new PSB trying to come into the pool. If such a PSB can be found, then the algorithm is done, and the space is found reasonably fast.
- “Pass 2” of the cast out algorithm is the problematic part. If there is no one single PSB that can make the space, the code picks the least recently used PSB, removes it from the pool, and then retries the get for space. If the search fails again, it picks the second least recently used PSB and removes it, and retries. It keeps doing this until a big enough hole is made in the pool for the new PSB to fit. All of this is done while holding several scheduling-related latches, and thus, scheduling is delayed for other PSTs while the current PST works on finding space in the pool. For a large pool, this is sort of like trying to make a door in a wall by shooting it at random spots with a BB gun. It

might make a hole eventually, but it will take a while.

It turns out that the PSBs are chained together in physical order within the pool. So it is possible to locate the PSB on the right and the left of a candidate PSB. If those PSBs are also not in use, then their space could be added to the current candidate's space to make a more intelligent choice about what PSBs to free, rather than waiting for the needed space to be formed by random "Swiss cheese" freeing.

Note: This changed cast out algorithm is incorporated in IMS V13.

PSB/DMB/PSBW sizes from ACBGEN output (1 of 2)

DFSUACB0 MESSAGES AND CONTROL STATEMENTS

BUILD PSB=ALL

DFS0940I DBD ADFASIGN HAS BEEN ADDED IN LIBRARY. DMB SIZE = 000640 BYTES
DFS0940I DBD ADFAAUDT HAS BEEN ADDED IN LIBRARY. DMB SIZE = 000904 BYTES
DFS0940I DBD ADFAMSGS HAS BEEN ADDED IN LIBRARY. DMB SIZE = 000696 BYTES
DFS0940I PSB ADFABCTL HAS BEEN ADDED IN LIBRARY. PSB SIZE = 008544 BYTES
DFS0941I PSB ADFABCTL IF USING DL/I SEPARATE ADDRESS SPACE, CSA SIZE = 000608,
SAS SIZE = 007888,
DFS0589I PROCESSING COMPLETED FOR PSB---ADFABCTL. PCB = 000960, PSB = 008544,
WORKAREA = 002848,
TOTAL SIZE = 011392
DFS0593I PSB--ADFABCTL WORKAREA BREAKOUT. NDX = 000256, XIO = 000008,
IOA = 001728, SEG = 000008, SSA = 00840
DFS0940I PSB ADFABCTP HAS BEEN ADDED IN LIBRARY. PSB SIZE = 008816 BYTES
DFS0941I PSB ADFABCTP IF USING DL/I SEPARATE ADDRESS SPACE, CSA SIZE = 008160.
DFS0589I PROCESSING COMPLETED FOR PSB---ADFABCTP. PCB = 000960, PSB = 008816,
WORKAREA = 002848,
TOTAL SIZE = 011664
DFS0593I PSB--ADFABCTP WORKAREA BREAKOUT. NDX = 000256, XIO = 000008,
IOA = 001728, SEG = 000008,

PSB
Pool

DMB
Pool

Going into PSBP
going into DPSB

Needed from
PSBW

PSB/DMB/PSBW sizes from ACBGEN output (2 of 2)

DFS0589I PROCESSING COMPLETED FOR PSB----Z . PCB = 000608, PSB = 001136,
WORKAREA = 001744,
TOTAL SIZE = 002880
DFS0593I PSB--Z WORKAREA BREAKOUT. NDX = 000056, XIO = 000688, IOA = 000420,
SEG = 00008,SSA = 00560

DFS0591I MAX PCB SIZE = 001544,MAX PSB SIZE = 009104, MAX WORKAREA SIZE = 012240
MAX TOTAL SIZE = 021200
DFS0942I IF USING DL/I SAS,MAX CSA = 000704 MAX SAS = 008448 AVERAGE CSA = 000348
AVERAGE SAS = 002922.

DFS0590I END OF ACBLIB MAINTENANCE. HIGHEST CONDITION CODE WAS 00000008

POOL Sizing

Notes:

For messages:

- DFS0940I
- DFS0941I
- DFS0589I
- DFS0593I

See *IMS Messages and Codes Reference* manual.

Pool Space estimation for IMS System (1 of 2)

$$\text{Size} = 12k + 4k * \text{MAXPST}$$
$$\text{Size} = \text{max PSBW size} * \text{MAXPST}$$

Non - resident and non-parallel scheduling:

Size = Sum of PSB(CSA) sizes + Online Change Size

= Sum of PSB(DLI) sizes + Online Change Size =

Sum of DMB sizes + Online Change

Non - resident and parallel scheduling:

$$\text{Size} = \text{Sum of PSB(CSA) sizes} + \text{max PSB size (CSA)} * (\text{MAXPST} - 1) * 2.5 + \text{Online Change}$$
$$\text{Size} = \text{Sum of PSB(DLI) sizes} + \text{max PSB size (DLI)} * (\text{MAXPST} - 1) * 2.5 + \text{Online Change}$$

Size = Sum of DMB sizes + Online Change



This is a starting point for a new IMS system for which you have no historical pool usage information:

- | | |
|-----------------|--|
| - Online change | Size of resources activated via Online Change or DRD. |
| - * 2.5 | Factor to avoid/minimize fragmentation for PSBs, since larger are more prone to fragmentation than DMBs. |
| - MAXPST | Sum of MAXTHRDs plus max number of regions (MPPs, BMPs). |

All sizes are reported (or can be calculated) from ACBGEN output.

Only online PSBs/DMBs should be considered and of course not those PSBs and DMBs (DBDs) used only for Batch processing.

Pool Space estimation for IMS System (2 of 2)

Resident and non-parallel scheduling:

PSBP(CSA)	Size = Online Change
PSBP(DLI)	Size = Online Change
DMBP	Size = Online Change

Resident and parallel scheduling:

PSBP(CSA)	Size = max PSB(CSA) size * (MAXPST - 1) * 2.5 + Online Change
PSBP(DLI)	Size = max PSB(DLI) size * (MAXPST - 1) * 2.5 + Online Change
DMBP	Size = Online Change

Consider [ACBIN64](#) (DFSDFxxx Mbr) ... more a few foils later !

<https://www.ibm.com/docs/en/ims/15.5.0?topic=pools-creating-sizing-64-bit-storage-pool>

DB Pools Backed by 64-bit Real Storage

- DB storage pools moved to 64-bit *real* storage (still in 31-bit virtual). When page fixed, these pools will now use 64-bit real storage:
 - DBWP: DB work pool
 - DLDP: DMB pool
 - DLMP: PSB CSA pool
 - DPSB: DLI PSB pool
 - PSBW: PSB work pool
- Target customers:
 - Customers with large pools (typically, PSB), who
 - Want to page fix them to avoid delays referencing old PSBs that have been paged out, but
 - Who cannot page fix them due to 31-bit real storage constraints.

Prior to IMS V12, the pools listed here were all obtained in 31-bit *virtual* storage, backed by 31-bit *real* storage when page fixed. In IMS V12, they will continue to be in 31-bit virtual, but will be allowed to be backed by 64-bit real. The target customers for this are those who have large pools, and who want to page fix them for performance, but who cannot because doing so causes a 31-bit real storage shortage.

DB Pools Backed by 64-bit Real Storage...

- The DB pools had been allocated LOC=(31,31)
 - 1st 31 = allocate in 31-bit virtual storage
 - 2nd 31 = allocate in 31-bit real storage *when page fixed*
- When not page fixed, storage can be anywhere in real storage.
- When page fixed, storage is forced to the real range specified by the second value of the LOC= parameter
 - For large fixed pools, this can cause a real storage shortage, because when fixed, LOC=(31,31) constrains real storage to 31-bit.
- In IMS V12, the DB pools are allocated with LOC=(31,64)
 - Virtual storage remains 31-bit, but...
 - Real storage can be 64-bit, even when fixed – no longer constrained!

DLI pools had been allocated with LOC=(31,31). The first parameter indicates where the virtual storage should be located... 31-bit in this case. The second parameter indicates where the real storage backing the virtual storage should be located *when fixed*. Also 31-bit in this case.

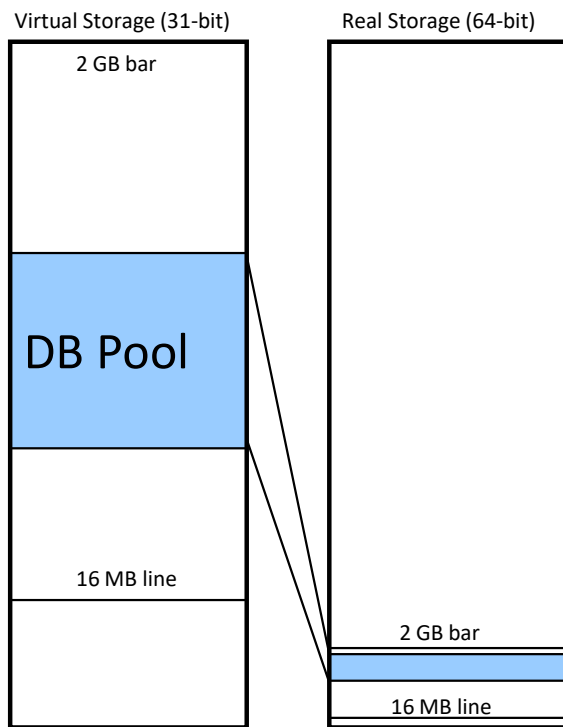
When storage is allocated and is not page fixed, it can reside anywhere in real storage. This is because non-fixed storage pages can be stolen and paged out, moved, etc. at any time. No one can (or should) be concerned with a real address of a non-fixed page of storage, because as soon as you know the address, it can change.

However, when pages are page fixed, then the second LOC= parameter comes into play as a constraint. Pages that are allocated LOC=(31,31) are forced to be backed by 31-bit real storage. 31-bit real is, today, a scarce resource. If a customer has large DB pools and wants to page fix them, they can run into 31-bit real storage problems, because there is not enough to hold the DB pools plus other users of 31-bit real storage.

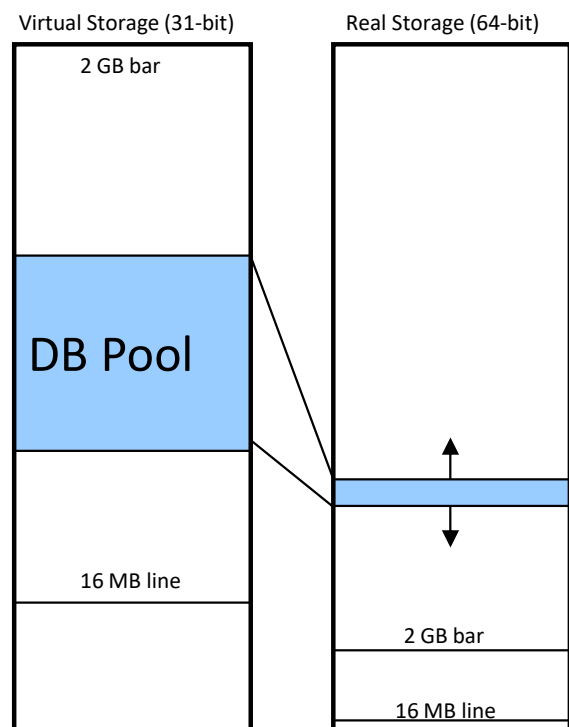
In IMS V12, DB pools are now allocated with LOC=(31,64). Virtual storage is still located in 31-bit, but now the pages themselves can be in 64-bit real, even when they are page fixed.

DB Pools Backed by 64-bit Real Storage...

IMS V11 and earlier (page fixed pools)



IMS V12 (page fixed pools)



This chart shows this pictorially. The left half of the chart shows the pre-V12 situation. The DB pools live in 31-bit virtual storage, which means they must be allocated with a virtual address less than X'80000000'. If these pools are page fixed, their real storage also must be in storage < X'80000000'. In the 64-bit real space, 31-bit storage is quite a small part of the total storage available, and it is easy to run out.

The right half of the chart shows the IMS V12 situation. Here, the pools are still in 31-bit virtual. However, when they are page fixed, they can now live in 64-bit real storage, and have a much wider range of allocation possibilities.

Also note: Although the pictures on this chart show the pool's real storage as one contiguous piece, be aware that this is not true. The real storage frames can be scattered throughout the real storage – there is no requirement for them to be contiguous.

64-bit ACB storage pool (1 of 2)

- Prior to IMS 11
 - At control region initialization :
 - Resident DMBs and PSBs are loaded into 31-bit extended storage of DLISAS or CTL (w/o DLISAS)
 - DEDBs are loaded into ECSA
 - During execution
 - non-resident DMBs and PSBs are loaded on demand into 31-bit non-resident pools (DMB and PSB pools)
- V11
 - An **optional** 64-bit storage pool to cache ACB members can be created in IMS 11 for non-resident PSBs and DMBs
 - The goal is to improve storage utilization and performance
 - Reduces I/Os to the ACBLIB and
 - Improves ACBLIB performance for shops with large ACBLIBs

The 64-bit ACB storage pool enhancement provides a separate pool for non-resident ACBs as an option to improve storage utilization and performance for those customers who have heavy I/O activity to the ACBLIB or have large ACBLIBs with many members.

64-bit ACB storage pool (2 of 2)

- With IMS 11 and 64-bit ACB storage pool
 - At control region initialization : Same as pre-IMS 11
 - During execution :
 - non-resident DMBs and PSBs are loaded into 64-bit ACB storage pool after being loaded on demand into 31-bit non-resident pools
 - Resident DMBs and PSBs will not go into 64-bit ACB storage pool
 - DEDBs will not go into 64-bit ACB storage pool
 - Supported in all online configurations (IMS/TM, DBCTL, DCCTL, SAS and non-SAS, XRF, FDBR)
 - DCCTL only has PSBs
 - No batch support
 - DOPT PSBs not supported

At execution time, as non-resident PSBs and DMBs are loaded from ACBLIB into the 31-bit non-resident pools, these non-resident PSBs and DMBs are also loaded into the 64-bit ACB storage pool, so they will be more easily accessible later.

All online configurations of IMS have support for a 64-bit ACB storage pool.
There is no support for this capability in batch (well, why it should there ??)
DOPT PSBs are not supported.

Specifying the 64-bit ACB storage pool

- Specification of the 64-bit ACB storage pool is in the new DATABASE section of the DFSDFxxx PROCLIB member
- Parameter is ACBIN64=nnn where nnn is the number of gigabytes for the 64-bit ACB storage pool (1–999)
- 64-bit ACB pool needs to be large enough to contain both non-resident PSBs and non-resident DMBs
 - Minimum would be sum of sizes of 31-bit non-resident PSB and DMB pools
 - Maximum would be a total size of all non resident ACB members
 - Recommendation is to start with 1 or 2 gigabytes

```
<SECTION=DATABASE>
```

```
ACBIN64=1
```



Remember
to specify
ACBIN64= in
DFSDFxxx
member for
FDBR if using

The 64-bit ACB storage pool is defined in a new section of the DFSDFxxx PROCLIB member called DATABASE. The parameter that must be specified is ACBIN64=nnn where nnn is the number of gigabytes of storage for this new pool. If the ACBIN64 parameter is not present, the 64-bit ACB storage pool will not be created and used.

ACBIN64 considerations (1 of 3)

- Scheduling considerations with non-resident ACB resources
 - At first scheduling of a program , a PSB and any related DMBs are loaded into the 31-bit non-resident pools and are also loaded into the 64-bit ACB storage pool.
 - At subsequent schedulings of this program , ACB members not found in the 31-bit non-resident pools are copied from the 64bit ACB storage pool (above the bar) back to the 31-bit non-resident pools (which avoids I/O to ACBLIB).
 - If the 64-bit ACB storage pool is full , the LRU algorithm will be used to remove old members to make room for new members

ACBIN64 considerations (2 of 3)

- Online change considerations for affected ACB members
 - will be removed from the 31-bit non-resident pools
 - will be Deleted from the 64-bit ACB storage pool
- Type-2 DELETE command considerations :
DELETE DB and DELETE PGM will
 - Remove ACB members from the 31-bit non-resident pools
 - Delete ACB members from the 64-bit ACB storage pool
- (Minimal) Impact on managing DMBs
 - DMBs today are either defined as resident or are always in the non-resident DMB pool
 - Most likely new 64-bit ACB storage pool will have minimal impact

ACBIN64 considerations (3 of 3)

- Impact of managing PSBs will depend on scheduling patterns (WFIs have no scheduling)
 - PSBs defined as resident today :
 - If large number of PSBs to be scheduled, investigate reducing / eliminating resident PSBs, increasing the size of the non-resident PSB pool, and using the 64-bit ACB storage pool
 - No noticeable performance impact of retrieving the PSB from 64-bit ACB pool versus from the resident PSB pool
 - PSBs using the non-resident PSB pool today
 - If non-resident PSB pool is sized larger to reduce/eliminate ACBLIB I/Os, investigate using a smaller non-resident PSB pool with the 64-bit ACB storage pool
 - 64-bit ACB pool removes potential I/O for PSBs
- **Benefits to summarize**
 - Improved technique for better management of non-resident ACBs
 - Goal is to improve storage utilization and performance for ACBs
 - Reduces I/Os to the ACBLIB
 - Improves ACBLIB performance for customers with large ACBLIBs
 - Improves ACB usability for customers where ACBLIB access impacts performance and growth

The actual impact will be based on the scheduling patterns of each IMS system !

For PSBs that are defined as resident today, if many of these are scheduled, it may be more efficient to make some or all of them non-resident, increase the size of the non-resident PSB pool, and use the 64-bit ACB storage pool to access them when needed. You can use the saving from reducing/eliminating resident PSBs to increase the size of the non-resident PSB pool. For PSBs that are non-resident today, if the non-resident PSB pool has been sized larger to reduce/eliminate ACBLIB I/Os, then using the 64-bit ACB storage pool would make it possible to reduce the size of the non-resident PSB pool. This would be helpful for customers with large numbers of PSBs that cannot always be found in the non-resident PSB pool today.

Querying the ACBIN64 storage pool

A new QUERY POOL TYPE(ACBIN64) can be used to monitor the usage of the 64-bit pool

```
QUERY POOL TYPE(ACBIN64) SHOW(STATISTICS)
```

PoolName	Type	CC	Size	Mbrs	Used	Free	Overflow
ACBIN64	Cache64	0	2048	10000	25	75	0

```
QUERY POOL TYPE(ACBIN64) SHOW(ALL)
```

PoolName	Type	CC	Size	Mbrs	Used	Free	Overflow	Gets	Hit	Miss
ACBIN64	Cache64	0	2048	3700	25	75	5	10000	90	10

Isrt	Del	Lmbr	Ltype	Lsize	Smbr	Stype	Ssize
300	20	PAYROLL	PSB	2000	DEBIT	INT	100

Here is an example of the new formats for QUERY POOL TYPE(ACBIN64) command.

Statistics for ACBIN64 storage pool

PoolNm	Pool name (ACBIN64)
Type	CACHE64
Size	Pool size in megabytes
Mbrs	Total number of buffers stored in the pool, whether in use or not
Used	Number of buffers currently in use (number of ACBLIB members in pool)
Free	The percentage of the pool that has not been reserved for buffers or control data
Overflow	Total number of overflow buffers in use
Gets	Number of FIND calls, whether successful or not
Hit	Number of successful FIND calls
Miss	Number of unsuccessful FIND calls
Isrt	Number of buffers added to the pool
Del	Number of buffers deleted from the pool, including castouts
Lmbr	Name of largest member in the 64-bit pool
Ltype	The resource type of the largest member in the 64-bit storage pool
Lsize	Size in kilobytes (K) of the largest member
Smbr	Name of smallest member in the 64-bit pool
Stype	The resource type of the smallest member in the 64-bit storage pool
Ssize	Size in kilobytes (K) of the smallest member.

The same type of information that is provided by a type-2 QUERY POOL TYPE(ACBIN64) command will be logged in a new type '4515' checkpoint log record.

Monitoring the ACBIN64 storage pool

- New log record – type X'4515'
 - Contains statistics from new QUERY POOL TYPE(ACBIN64) command
- New monitor record types
 - Type 74 - Issued when a get request for a PSB in the 64-bit pool is started
 - Type 75 - Issued when a get request for a PSB in the 64-bit pool ends
 - Type 76 - Issued when a get request for a DMB in the 64-bit pool is started
 - Type 77 - Issued when a get request for a DMB in the 64-bit pool ends
- Supported by IMS Monitor in the Region IWAIT Scheduling + Termination report
- Supported by IMS PA tool

The IMS Monitor will record four new record types for usage of the 64-bit ACB storage pool: type 74 indicates that a get request for a PSB in the 64-bit pool has started, type 75 indicates that a get request for a PSB in the 64-bit pool has ended, type 76 indicates that a get request for a DMB in the 64-bit pool has started, and type 77 indicates that a get request for a DMB in the 64-bit pool has ended.

These new monitor record types are supported by the IMS Monitor and the IMS PA tool.

ACBIN64 storage pool – IMS Monitor REGION IWAIT report

IMS MONITOR *** REGION IWAIT *** TRACE START 2008 123, 08:01:32 TRACE STOP 2008 123, 08:11:48 PAGE 0018

```

.....IWAIT TIME.....
**REGION      2  OCCURRENCES      TOTAL      MEAN      MAXIMUM      FUNCTION  MODULE
-----
SCHEDULING + TERMINATION
                2      32975611      16487805      23293621      NO MESSAGES  MSC
...SUB-TOTAL...
                2      32975611      16487805
                1          5807          5807          5807      PSB=DDLTRN24  BLR-64BIT
                1          1985          1985          1985      INT=DDLTRN24  BLR-64BIT
                3          5115          1705          1965      PSB=BMPFPE07  BLR
                1          1154          1154          1154      INT=BMPFPE07  BLR
                3          3040          1013          1199      PSB=BMPFPE05  BLR
                1          1028          1028          1028      INT=BMPFPE05  BLR
                1          1739          1739          1739      PSB=BMPFPE02  BLR-64BIT
                1          1628          1628          1628      INT=BMPFPE02  BLR-64BIT
                1           3100          3100          3100      PSB=BMP255    BLR-64BIT
..TOTAL...
                15      33056434      972248
DL/I CALLS
-----

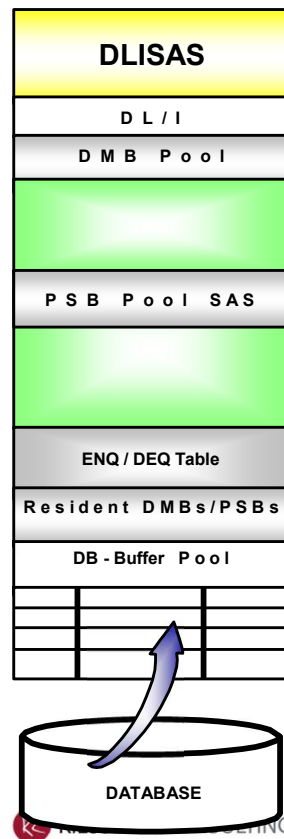
```

This is an example of the IMS Monitor REGION IWAIT report that shows activity in the 64-bit ACB storage pool.

IMS DB processing (Full Function)

DB

IMS DB Basics now ?



KC110 unit 3 page 55

Notes:

After scheduling completes, most interaction between the application and IMS databases involve the DLISAS Address Space

Databases opened when first accessed

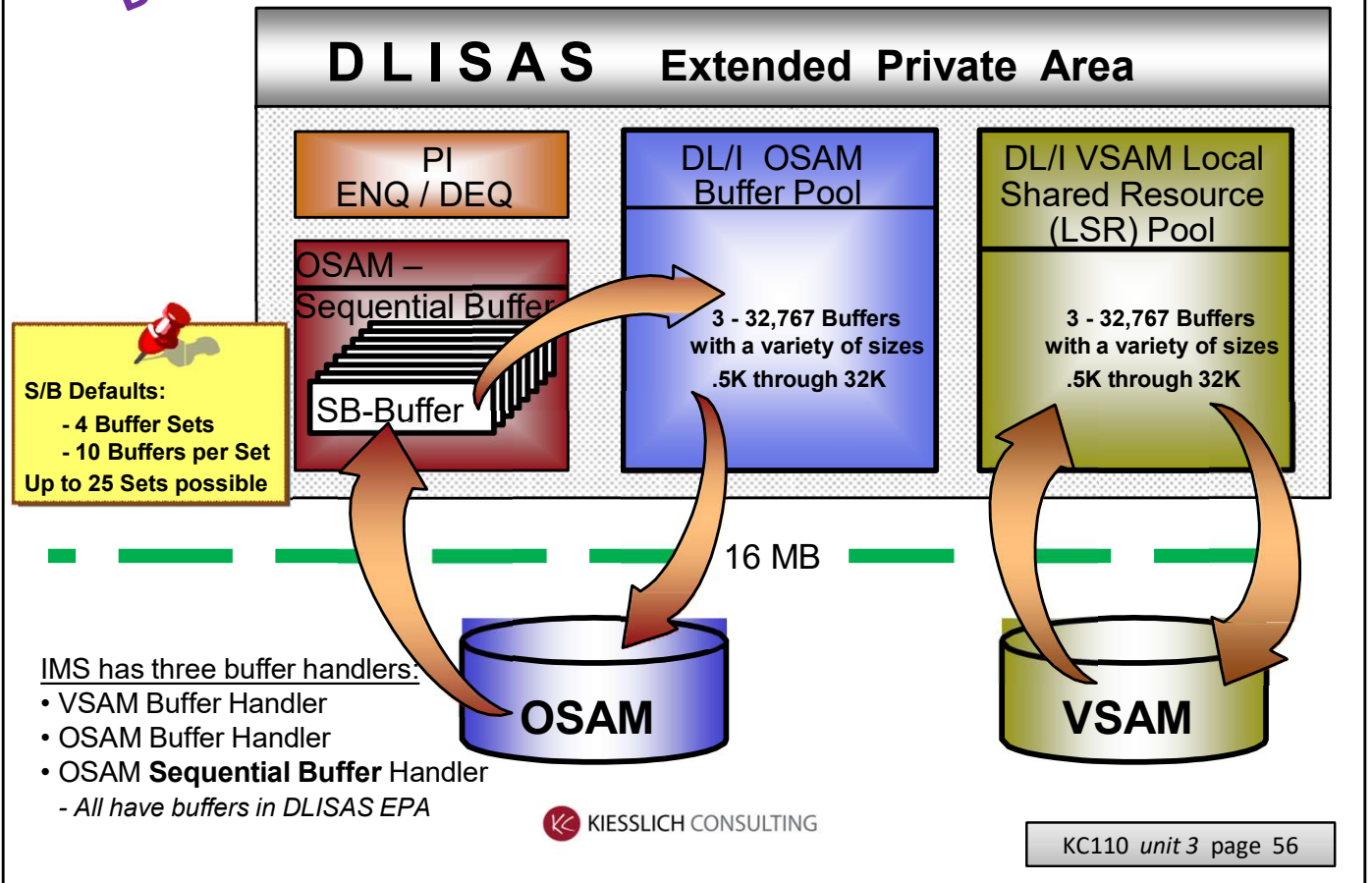
Read I/O (if segment not in DB buffers)

Data updated in buffer after locks have been granted

- Updates logged
- Write I/O usually deferred until commit point Updates locked until commit point.

IMS Database Buffers

DB



Notes:

In addition to storage for the Pools we have discussed earlier, the DLISAS also is used for IMS Database Buffers and Lock Tokens ("PI ENQ/DEQ Pool") if Program Isolation is used as the lock manager.

OSAM SB ? Anticipating ... "in advance buffering" assuming more sequential processing

Only OSAM 😊 possible – "we" own it – IMS core code !

OSAM Buffer Pool Definition

CONTROL STATEMENTS (DFSVSM**) specifications:

IOBF=(bufsize,# buffers,fix1,**fix2**,id)
DBD=dbdname(data set number,id)



Sample:

```
IOBF= (512 , 12 , Y , Y)
IOBF= (2048 , 5 , Y)
IOBF= (4080 , 6 , Y , Y , PROD)
IOBF= (4096 , 6 , N , Y)
IOBF= (12288 , 4 , Y , Y , CUST)
DBD=DBD3 (1 , PROD)
DBD=DBD4 (2 , CUST)
```

IOBF Defines each OSAM subpool

- bufsize and #buffers is used to specify size and number of buffers in this subpool
- no subpool specified: there is only one global ... number 001 / or the first as global one ... 001 , for all those which are not assigned to a specific other pool id
- The following parameters (values are Y / N) request long-term-page-fix for:
 - ✓ fix1 – for Buffers and Buffers Prefixes (Header) - You might like to specify "Y"
 - ✓ fix2 – for Buffer Prefixes and Subpool Header (control blocks) - You always should specify "Y"
 - ✓ id (optional) is 1 - 4 alphanumeric characters to name the subpool

DBD is used to assign the database data set to that certain OSAM buffer subpool with a matching ID of OSAM pool

- DATA SET NUMBER is determined by the order of the DATASET macros in the specified DBD.

Please note changes here when in HALDB !

Structure of VSAM Shared Resource Pool

DB

VSAM Buffer Handler Pool

Buffer Pool Prefix
Sub - Pool 1 Prefix
Sub - Pool 2 Prefix
Sub - Pool 3 Prefix
Sub - Pool 4 Prefix

VSAM Shared Resource Pool

512	512	512	512	512	512	512	512
1024		1024		1024		1024	
1024		1024		1024		1024	
2048				2048			
2048				2048			
4096							
4096							
4096							
4096							

Buffer Pool Prefixes are used by the buffer handler to identify the content of individual buffers.

VSAM Subpool Definition is specified in Proclib member DFSVSAMP (BATCH) or DFSVSMxx (Online).

IMS sample VSAM buffer pool for a control region:

VSABF=512,8 VSABF=1024,8 VSABF=2048,4 VSABF=4096,4

OLDSDEF OLDS=(00,01,02),BUFNO=005,MODE=DUAL, WADSDEF WADS=(0,1)

BGWRT=YES,INSERT=SEQ,VSAMPLS=LOCL

DB

VSAM Buffer Pool Definition example

OPTIONS VSAMPL=LOCL (default) no HIPERSPACE is considered		
POOLID = IDG		General Pool:
VSRBF =		For all DBs not assigned to a
..... =		specific POOLID and
		must be the first subpool
POOLID = ID1		
VSRBF = 1024,8		
VSRBF = 2048,16		
VSRBF = 4096,30		
POOLID = ID2, FIXINDEX=YES		
VSRBF = 2048,24,I		
VSRBF = 4096,10		
POOLID = ID3, FIXDATA=YES		
VSRBF = 1024,12		
DBD = DBD1I,(1,ID1)		
DBD = DBD1D,(1,ID1)		
DBD = DBD3I,(1,ID2)		
DBD = DBD4I,(1,ID2)		
DBD = DBD4D,(1,ID3)		

WHERE: POOLID : Pool Id used to assign data set to pool
 nn,nn,type : TYPE: I = Index Subpool, D = Data Subpool
 FIXDATA =, FIXINDEX= : Page fix buffers
 FIXBLOCK = : Page fix control blocks
 DBD = : Assign data set to matching Pool ID

KC110 unit 3 page 59

Definition in:

IMS.PROCLIB (DFSVSMMxx)

or in:

//DFSVSAMP DD

Virtual Storage Constraint Relief (VSCR) was provided long ago for IMS buffers.

- Buffers and VSAM control blocks are allocated in DLISAS EPA so a large number of buffers might be used.

Multiple VSAM LSR Pools (multiple *POOLID* statements) permit:

- Multiple subpools with **same** buffer size
- DB might be assigned to a specific LSR pool
- Subpool might be designated as a KSDS INDEX ONLY subpool
- Subpool could/should be large enough to enable most of the KSDS INDEX CIs for important databases to remain in storage
- Number of buffers 3, ..., 32 767

DB

VSAM Buffer Pool Definition example

OPTIONS VSAMPL=LOCL (default) no HIPERSPACE is considered		
POOLID	= IDG	General Pool:
VSRBF	=.....	For all DBs not assigned to a
.....	=.....	specific POOLID and
		must be the first subpool
POOLID	= ID1	
VSRBF	= 1024,8	
VSRBF	= 2048,16	
VSRBF	= 4096,30	
POOLID	= ID2, FIXINDEX=YES	
VSRBF	= 2048,24,I	
VSRBF	= 4096,10	
POOLID	= ID3, FIXDATA=YES	
VSRBF	= 1024,12	
DBD	= DBD1I,(1,ID1)	
DBD	= DBD1D,(1,ID1)	
DBD	= DBD3I,(1,ID2)	
DBD	= DBD4I,(1,ID2)	
DBD	= DBD4D,(1,ID3)	

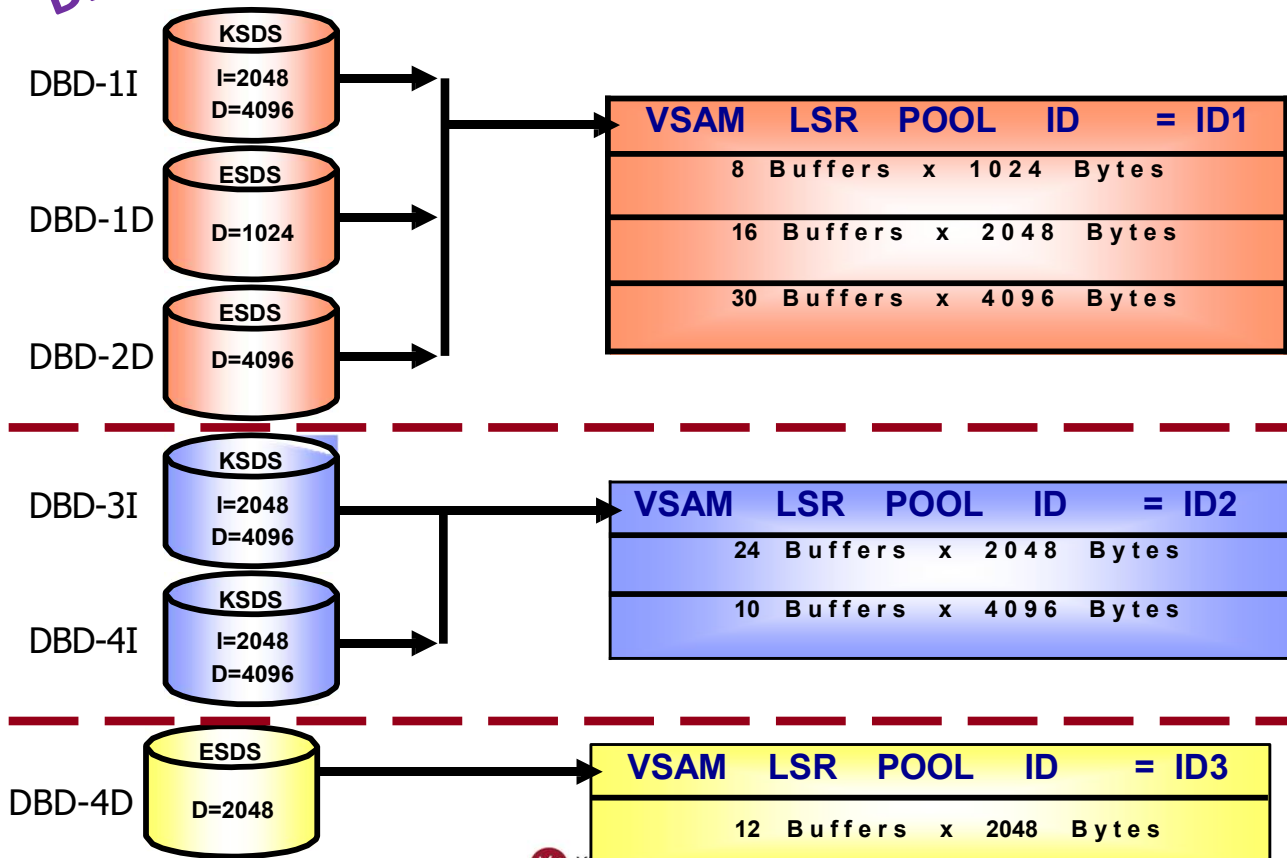
WHERE: POOLID : Pool Id used to assign data set to pool
nn,nn,type : TYPE: I = Index Subpool, D = Data Subpool
FIXDATA =, FIXINDEX= : Page fix buffers
FIXBLOCK = : Page fix control blocks
DBD = : Assign data set to matching Pool ID

KC110 unit 3 page 60

For HALDB is different !

Using Multiple VSAM Buffer Pools

DB



KC110 unit 3 page 61

Notes:

In this example, we have multiple database data sets that have a common CI Size:

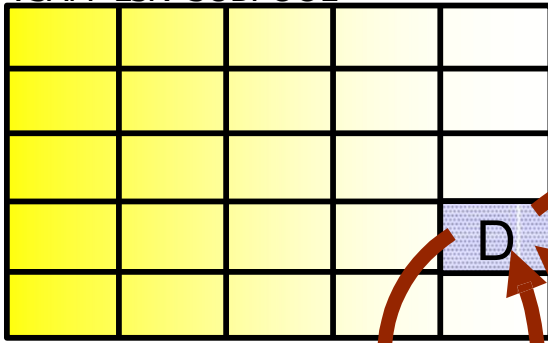
- Note that the KSDS Index components for DBD-1I and DBD-3I and ESDS Data component for DBD-4D have a common CI size of 2048.
- By being able to isolate these three VSAM components into separate buffer pools, we prevent buffers stealing between these databases.

Allocation of a data set to a subpool occurs at **OPEN** time only.

DB

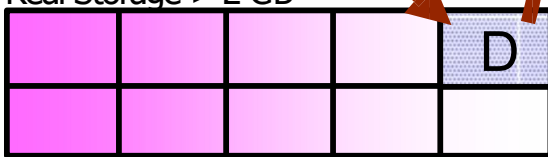
VSAM HiperSpace

VSAM LSR SUBPOOL

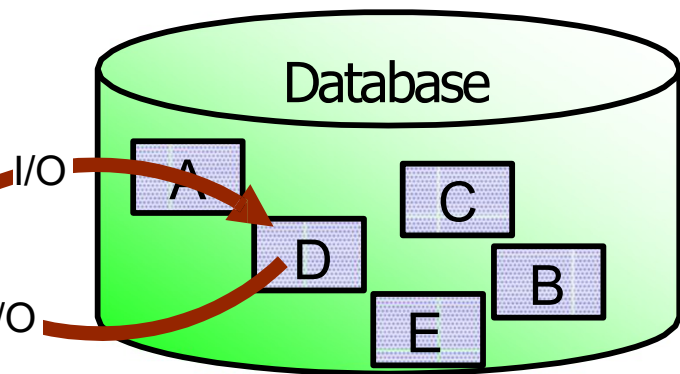


Addressability: Byte

Real Storage > 2 GB



Addressability : Page



HiperSpace Buffering was appropriate when IBM Processors had two types of Storage: *Central* and *Expanded*

- Current z-Series Processors only have Central storage
 - Use of HiperSpace, while still supported, involves moving data from one area of memory to another
- The movement of IMS HiperSpace Buffers from one portion of memory to another is wasted overhead
- Convert your existing IMS VSAM HiperSpace to *real* IMS VSAM Buffers

Notes:

HiperSpace was implemented to improve Buffer Hits and reduce Read I/O operations when the amount of LSR buffers was limited by hardware constraints.

Although no longer recommended, if HiperSpace is used the following points apply: LSR subpools using HiperSpace should be multiples of 4 K.

Each subpool has one set of BUFCs (buffer control blocks) and two sets of buffers:

- Each BUFC points to a buffer, either in the address space or in HiperSpace
- Both sets of buffers are managed on a least-recently-used basis

When a buffer is stolen from the address space buffer pool, it is moved to the HiperSpace buffer pool. If the buffer was altered, it is written to DASD prior to being moved.

When a buffer is stolen from the HiperSpace pool, its contents will be discarded.

When a VSAM request requires a CI, VSAM will look for the CI in buffer to avoid I/O.

If requested CI found in HiperSpace, VSAM will move HiperSpace buffer contents to an address space buffer. This will usually force an address space buffer to HiperSpace.

I/O always performed with address space buffers.

Application programs (IMS) cannot reference buffers in HiperSpace.

HiperSpace is no longer recommended as a performance option.

HiperSpace Activation

Example:

Options VSAMPLS=LOCL

VSRBF = 2048,12

VSRBF = 4096,10,D,**HS30**

VSRBF = 8192,4,I,**HS20,HSO**

VSRBF = 12288,6,,**HS20,HSR**

Notes:

Hiperspace parameters added to VSRBF=

HSnn number of Hiperspace buffers needed:

- n = 0 to 16,777,215

HSOPTIONAL Hiperspace buffers optional (default):

- Initialization will *proceed* even if not enough HiperSpace is available to meet requested amount

HSREQUIRED Hiperspace buffers required:

- Initialization will fail if not enough HiperSpace is available.

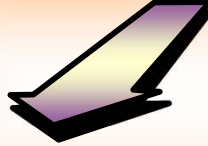
Buffers for smaller buffer sizes (that is, <4K) should be in 4K increments since 4K is the management size.

VSAM Number of Strings

DFSVSMSx

x

POOLID = ID1,STRINGMN = # of VSAM Strings



Max # of concurrent VSAM requests
for FF DBs using POOLID = ID1

OPTIONS VSAMPLS = LOCL,STRINGMX = # of VSAM Strings



Max # of concurrent VSAM requests
for **all** pools

Notes:

The Interface between IMS and VSAM requires a String in order to perform I/O operations.

In general, the parameter used to specify the number of strings should not be specified. STRINGNM and STRINGMX parameters can be used to allow independent control of VSAM number of strings allocated for DB I/O.

STRINGMX allows PSTs and VSAM strings to be tuned independently (consider MAXPST=).

BMP processing: Non-message-driven (1 of 2)

DB

- BMP started by JCL or z/OS console

```
// EXEC IMSBATCH, MBR=pgmname, PSB=psbname, . . .
```

- Security verification
- PSB and DMBs loaded (if currently not in pools)
 - Databases authorized if registered with DBRC
- Region (PST) assigned or created (up to MAXPST=)
- UOR started (if UPDATE intent)
- Multiple (hopefully! *) UORs
- PSB remains scheduled until PGM termination

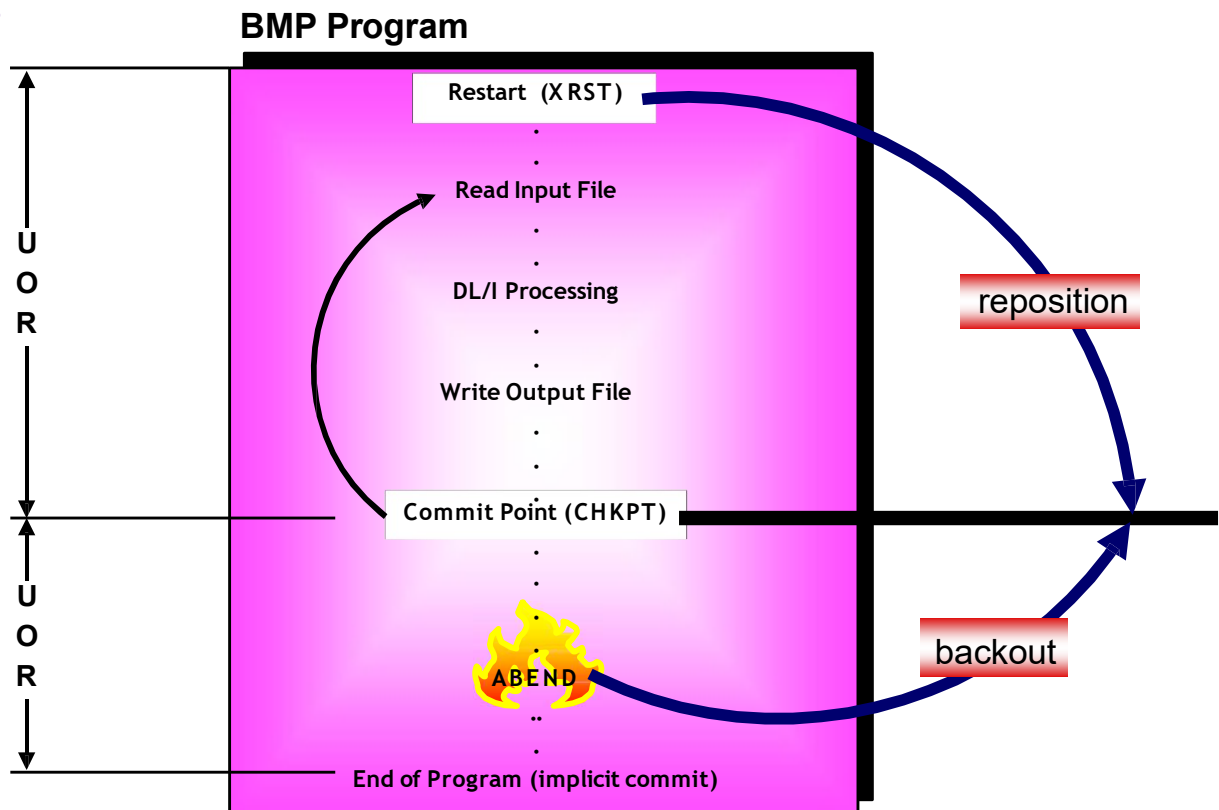
Notes:

Most of the BMP processing performed by customers is through the so called non-message-driven BMPs. These jobs use the services of the Control Region, the DLISAS and DBRC, but **do not access IMS Message queues**.

* Hopefully – because overhead of region start / stop and work inbetween (checkpoints !!!)

BMP processing: Non-message-driven (2 of 2)

DB

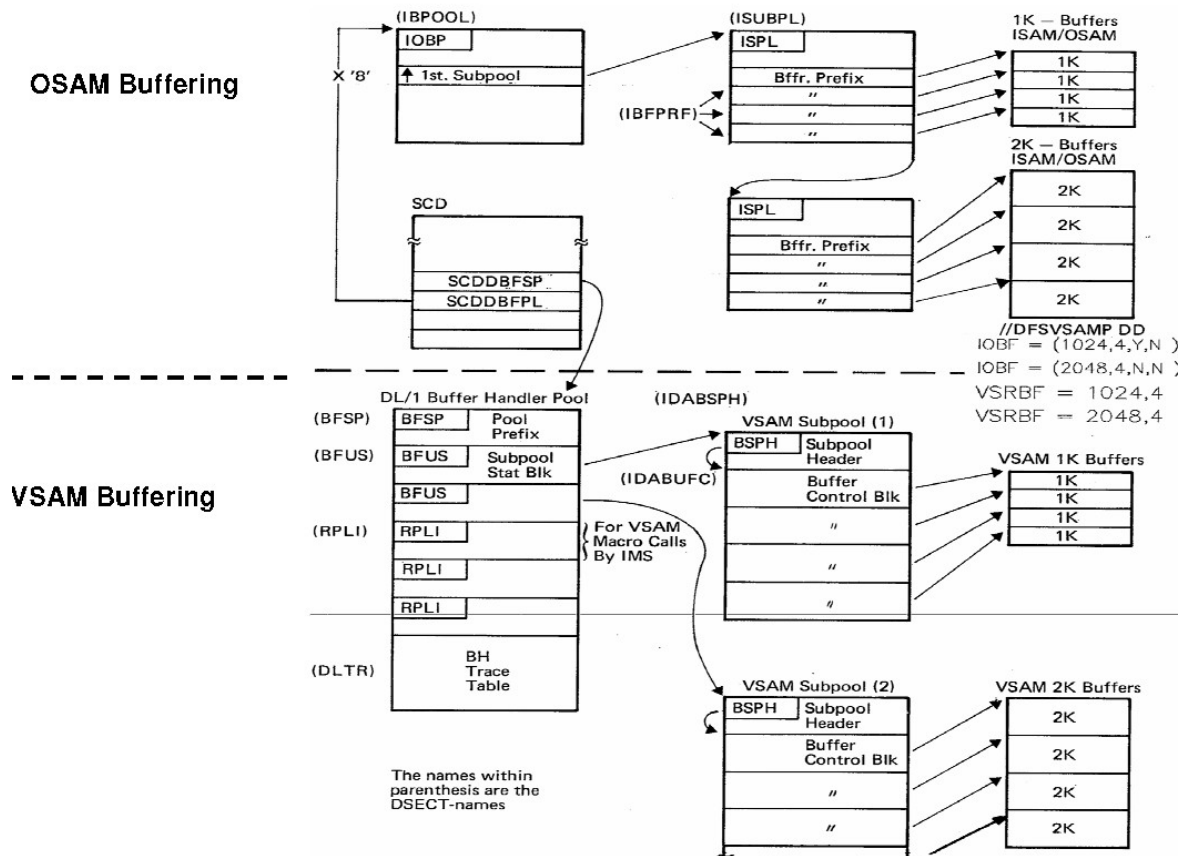


Notes:

An option available to any type of BMP (non-message-driven shown here) is the use of the CHKPT and XRST calls. When used together, these calls permit application to periodically commit work that can be restarted from the point of failure without the need to redo all earlier processing.

CHKPT file - GSAM

Database buffers – Control block overview



Deep dive digging here: IMS Debugging class !