

Unit 5

IMS Locking, IMS Logger, Syncpoint Processing & IMS System Services

After completing this unit, you should be able to:

- List two different lock managers that are available for use by IMS
- Recognize the different data sets used by the IMS Logger and some of the key characteristics of these data sets
- Describe how to specify Logger-related options and what Log Archive options are available
- Understand how the contents of log records can be used for system, database, and application recovery
- Describe the processing that occurs as part of application syncpoint processing and facilities available within IMS that assist with application restart processing

Program Isolation (1 of 2) ^{LOCK Mgmt}

- **Program Isolation** (PI) facility is IMS's original Lock Manager
 - key function of any lock manager: preventing other units of work from accessing updated data prior to the changes being committed
 - PI allows multiple units of work (UOW) the ability to concurrently access the same database (but not the same segment occurrence!) for update while preserving integrity
- **Dynamic Backout facility:**
 - Works with the IMS Lock Managers to automatically remove the effects of any program that abnormally ends (ABEND) before committing its updates
 - Eliminates partial updates - uncommitted updates are backed-out prior to lock release
 - One of the more common reasons for Dynamic Backout being invoked is to manage ABENDs triggered due to *lock failures*



KIESSLICH CONSULTING

KC110 unit 5 page 2

Notes:

Program Isolation is the a very commonly used IMS lock manager. The purpose of any lock manager is to prevent data from being accessed by a UOW while another is in the process of making updates. From a high-level view, PI works very similarly to IMS's other lock manager, IRLM. We will discuss IRLM later.

- PI is implemented through control blocks in the IMS DLISAS
 - + Each instance of these blocks represents either a resource or requestor of a resource

Program Isolation (2 of 2)

LOCK Mgmt

- **Deadlocks** are a variety of lock failure that is possible with any lock manager

– PI handles deadlocks through Identification and avoidance:



- PI analyzes for the possibility of a deadlock as part of granting lock requests
 - If PI determines that granting a lock would produce a deadlock, it selects *loser* UOW for *pseudo-ABEND* and *dynamic back-out*
 - *Loser* UOW is re-scheduled for later (with IMS TM) if possible
 - The ability to automatically reschedule the UOW is one of the factors PI uses to select the *loser* – that is, the UOW easier to reschedule if it is more likely to be ABENDED if involved in a deadlock
- A lengthy wait for a lock is not necessarily a deadlock
 - PI will wait indefinitely for locks as long as they are not deadlocks

RC110 Unit 5 page 5

Notes:

One of the possible results of locking is that a deadlock situation could develop. PI identifies impending deadlocks, and works with IMS to identify which UOW should be backed out to avoid the deadlock. For IMS TM transactions, the input message associated with the failed (*loser*) UOW is requeued and reprocessed later.

One significant difference between PI and IRLM is that there is no *time-out* value associated with PI; when using PI, a wait for a lock could last seconds, minutes, or even hours.

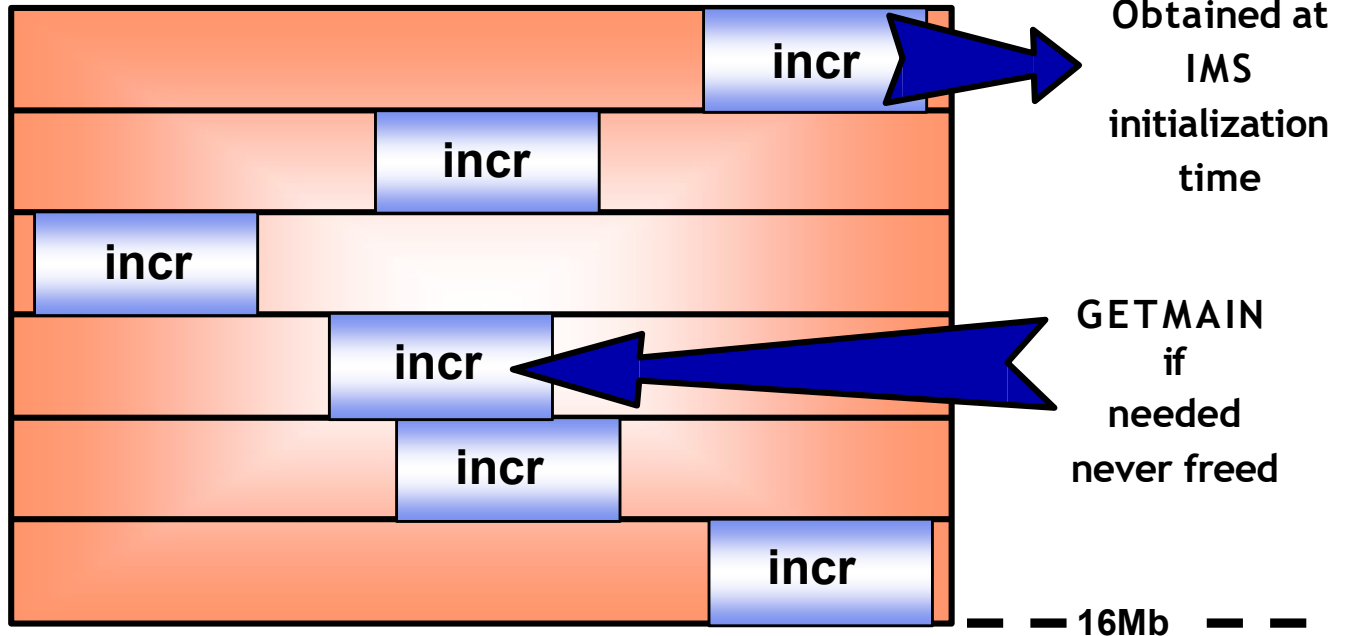
Program Isolation ENQ/DEQ Pool

LOCK Mgmt

Specified in IMS startup procedure

PIMAX = (max. bytes)

PIINCR = (increment)



KISSLICH CONSULTING

KC110 unit 5 page 4

ENQ/DEQ pool resides in DLISAS EPA storage along with database buffers and other DL/I storage pools. Storage acquired dynamically in specific increments to specific maximum:

Available for reuse at SYNCPOINT/CHECKPOINT.

If no more space available the PSB abends with U775.

Isolate problem applications using LOCKMAX= (PSB or DLI/DBB/BMP/MPP procedure) which forces a PSB pseudo abend U3301.

Estimated Size = (peak number of calls per unit of work/recovery) * (average number of enqs/deqs per call [use 3]) * (locksize [48 bytes]) * (max # of active PSTs)

In estimating size of the PI ENQ/DEQ pool, you should focus on BMPs as they are likely to hold many more locks than transactions. Even if IRLM is used, specify a max space of at least 40K for HD space management purposes. IRLM lock storage in IRLM EPA.

IRLM

LOCK Mgmt

- IRLM was introduced to enable IMS *Data Sharing*
 - This lock manager was later adopted by DB2
- Data Sharing use is not a requirement for the use of IRLM instead of PI:
 - Lock granularity is different between PI and IRLM
 - PI locks at the Segment and Database record level and IRLM locks at the Block Level for updates
 - In most cases, PI uses less CPU than IRLM
- We will discuss IRLM further in the Data Sharing topic
- There is a very good doc about [IMS Locking](#)



KIESSLICH CONSULTING

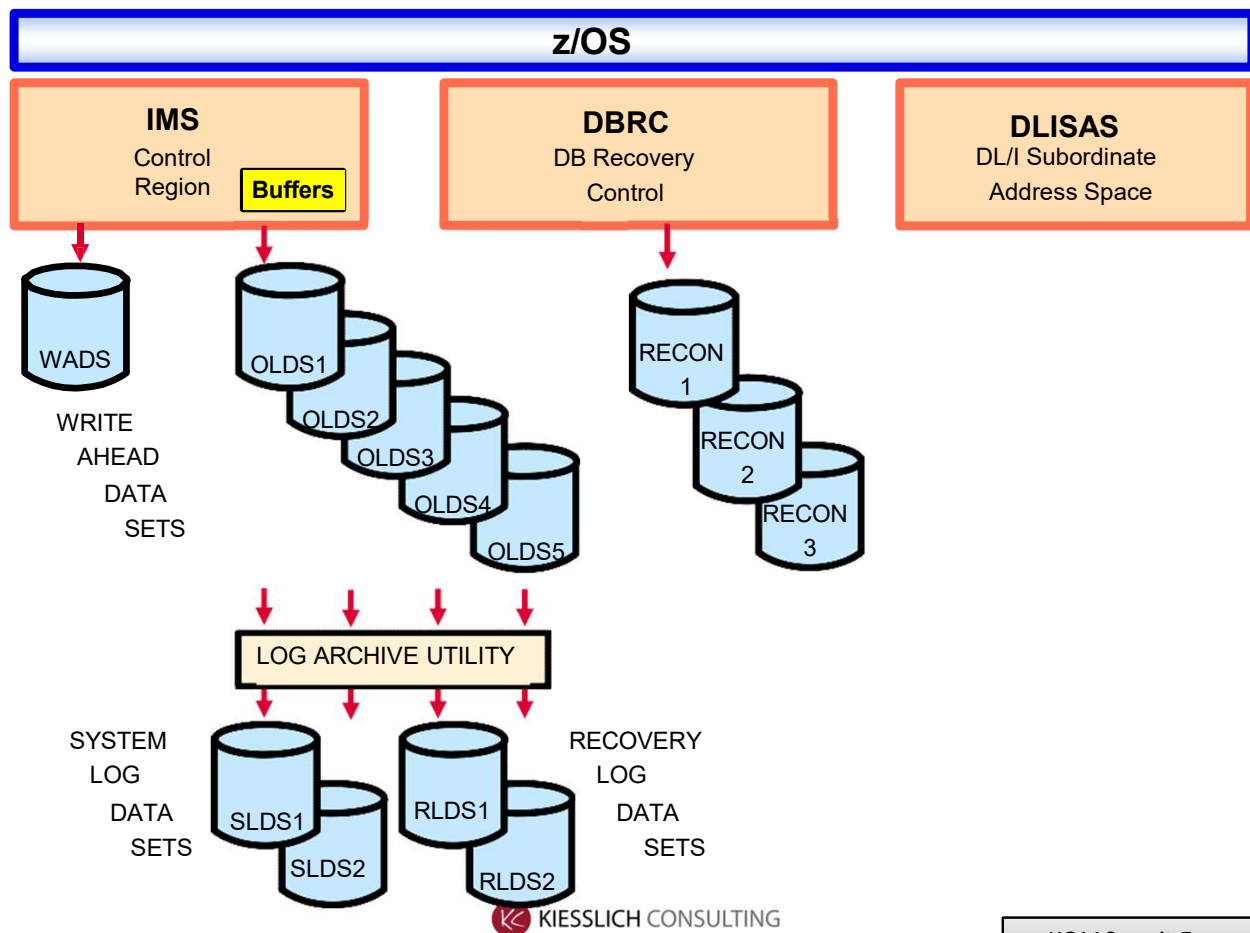
KC110 unit 5 page 5

IMS Ressource Lock Manager – first called that way

Notes:

IRLM must be used if you need to perform IMS Data Sharing. Data Sharing is the function that permits multiple IMS systems to share IMS databases with integrity – even when the IMS systems are running on different processors.

Common Logging Facility overview LOG Mgmt



KC110 unit 5 page 6

„Write ahead“ – explained next foils ; DBRC – will be taught in separate class
SLDS / RLDS (optional) - a topic for DB recovery, CA a.s.o.

Notes: An active IMS System writes to two types of log data sets from ***common*** Log Buffers in Control Region storage (or in ECSA Storage allocated by the control Region):

1. **WADS** (Write Ahead DataSets) that are used to externalize log records that IMS considers critical. The contents of these datasets is constantly being overwritten as their content is rewritten to the OLDS. Because WADS data is copied to OLDS datasets, there is no additional need to *archive* the contents of the WADS.
2. **OLDS** (Online Log DataSets) are IMS's primary active log and are used in pairs. As these logs are filled, they are used in a *round-robin* order.

An IMS system will periodically submit an Archive Job. As noted above, OLDS pairs eventually fill up. In order to maintain system and database recoverability, their contents must be preserved before they can be reused. The IMS Archive Utility is a batch job generated by DBRC that can create the following data set types from the filled OLDS; they are:

1. **System Log DataSet (SLDS):** Might contain data from one or more OLDS; used primarily for database recovery or Emergency Restart. At least one SLDS must always be created by the Archive Utility/Job. A second SLDS dataset can be optionally created by an execution of the Archive utility, depending on how the JCL for the Archive utility is tailored.
2. **Recovery Log DataSet (RLDS):** Contains only log records needed for database recovery.

The creation of the RLDSs is optional, depending on how the JCL for the Archive utility is tailored, one or two RLDS datasets can be created at the same time that the one or two SLDS datasets are created.

Common Logging Facility (1 of 3)

LOG Mgmt

- Recording of IMS/DB and IMS/TM system activity:
 - *Before* and *After* images of changed DL/I segments
 - System status via periodic checkpoints
 - Input and Output messages are also logged
- Provides integrity and recoverability of DB/TM systems:
 - Used for Dynamic Backout of failing transactions
 - Used for system *Warm* and *Emergency* restarts
 - Used for database recovery
 - Used for performance and accounting statistics

Common Logging Facility (2 of 3)

LOG Mgmt

- IMS internally has two log processes:
 - Logical Logger moves data to log buffer
 - *WRITE* - requests
 - For example, database *before* and *after* records
 - Physical Logger writes/reads data between log buffer and disk
 - A *CHECK WRITE* is a synchronous process to verify certain records are on Disk
 - If not, a request to initiate I/O is issued and requesting ITASK will wait for completion
 - Check Write requests can result in partial buffers being written to the WADS
 - Check Write example: Purge changed DB buffers for buffer steal
 - A *WAIT WRITE* is also a synchronous request to externalize log records
 - Similar to *Check Write* except Wait Write does not initiate I/O request
 - Requesting ITASK will be notified when its log records have been written by :
 - > *Piggybacking* on Check Writes (or the log buffer filling) of other ITASKS
 - > A *timer pop* that periodically writes log records during periods of low activity
 - Wait Write example: Commit related records at application syncpoint
 - *READ* requests read data from disk to buffer
 - Used mainly for dynamic backout processing



KISSLICH CONSULTING

KC110 unit 5 page 8

Notes:

These are the types of operations that can be requested of the “IMS Logger”

These requests are not explicitly requested by applications that use IMS; these requests are invoked by various components of IMS as required.

Logical Logger code will get called from EVERYWHERE !

Implicit checkpointing : next GU , end (GoBack)

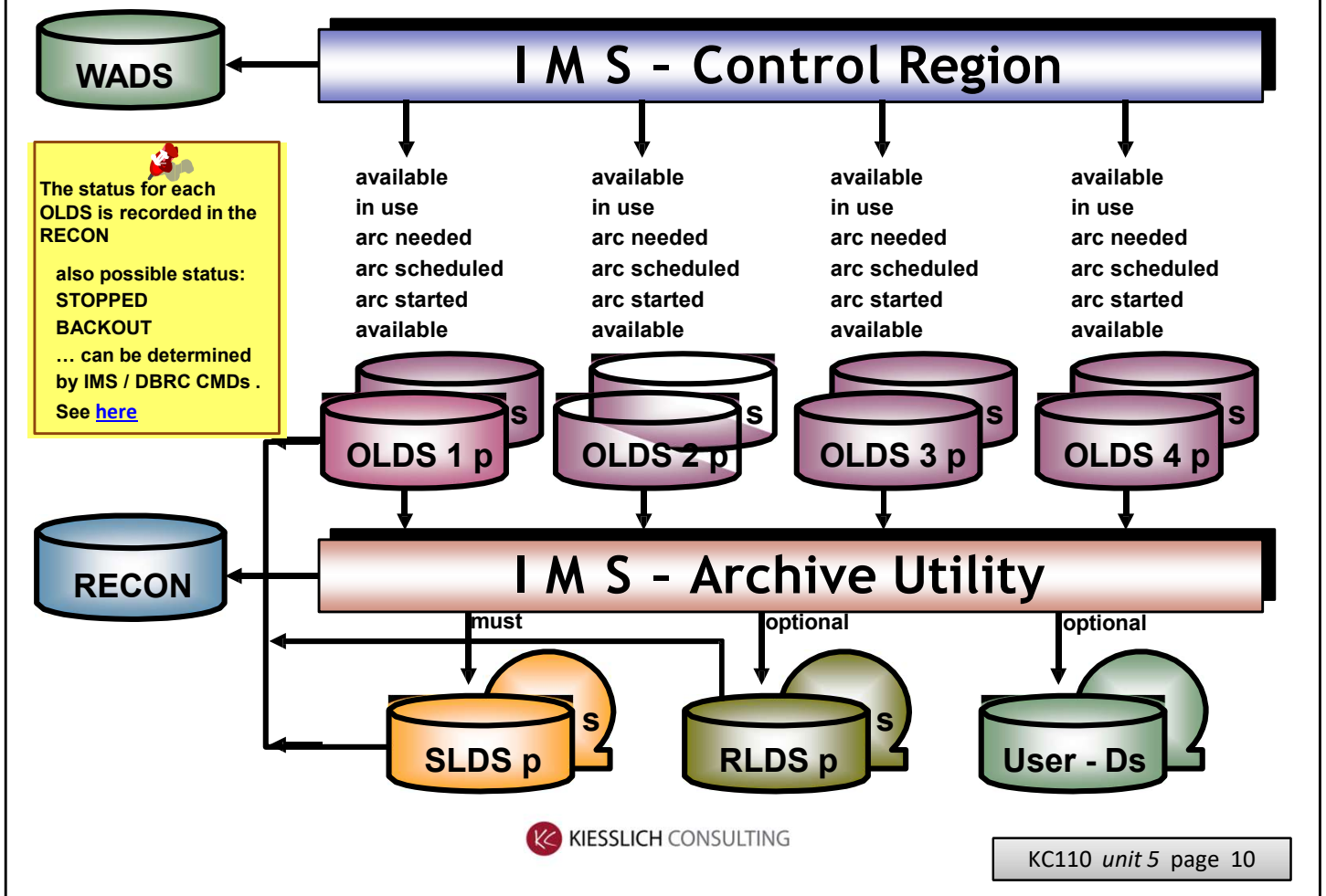
- EXPRESS PCB : writing to queue / updating immediately
- It's similar to DB2 option IMMEDWRI
 - setting via DSNZPARM (the „parm block“ of DB2 – similar to DFSPBxxx)

Common Logging Facility (3 of 3)

LOG Mgmt

- Log Write-Ahead (LWA) - WADS:
 - To guarantee database integrity, log records are physically written before database CI's/Blocks are physically written
 - Only selected *events* qualify for this technique
 - The DCLWA= parm (on TRANSACT macro) also controls when log records associated with messages must be written for the transaction
 - Implemented through the use of Check Writes and Wait Writes

Principle of IMS DASD logging LOG Mgmt



Dual logging :

DD via DFSMDA : OLP00 / OLS00 – prealloc. to different DASD (control units) – well, old fashioned ☺ !

Status tracked in RECON !!!

- RECON access for any change , for SLDS /RLDS LOGALL / SUBSYS , ... !!!
- And in parallel to DB activity ! (PRA ?)
- Status : backout / WRITE ERROR ... see link:

<https://www.ibm.com/docs/en/ims/15.5.0?topic=commands-display-olds-command>

Or

<https://www.ibm.com/docs/en/ims/15.1.0?topic=requests-olds-query-request-typeolds>

Option SLDSREAD ON/OFF

Explanation see „Response ET“



and



LOG Mgmt

- **OLDS** and **WADS** are written from common IMS buffers
 - Contains log records needed for:
 - IMS System Restart
 - Database Recovery
 - Application Dynamic Backout
- Online Log Data Set (**OLDS**):
 - Are only written when the IMS log buffer is filled
 - Consists of multiple data sets - usually in pairs
 - Allows IMS to continue logging when an *OLDS switch* occurs because of the following events:
 - I/O error – especially OLDS is full
 - Operator commands /DBR or /DBD
 - Written in a *wrap-around* manner
 - Contents archived by Archive Job to the System Log Data Set (SLDS)
- Write Ahead Data Set (**WADS**):
 - Written in response to a Check Write or *timer pop*
 - Contains committed log records not yet written to OLDS
 - Only short (2048 or 4096 bytes of data), full blocks are written
 - Can be used to *close* the OLDS in an emergency restart

Definition of IMS log datasets (1)

LOG Mgmt

- **OLDS**: Online Log Data Set
 - Contains all log records required for IMS restart, dynamic backout and recovery
 - BLKSIZE min 6K and max 30K
 - 26K was previous recommended blocksize because it was half-track blocking
 - > Currently, if OLDS Blocksize is divisible by 4K, Log Buffers will be stored above 2 GB bar and WADS blocksize will be 4K instead of 2K
 - For 3390 BLKSIZE=24K is **recommended**
 - Dual OLDS **recommended**
- **WADS**: Write Ahead Data Set
 - Might contain committed log records not yet written to OLDS
 - Guarantees log records available for restart/recovery after system failure
 - Can be formatted at startup
 - Dual WADS **usually recommended**
 - Spare WADS **always recommended**
 - BLKSIZE=2080 (2048 + 32 Prefix) or 4128
 - Min TRKs = $[(\text{OLDS-BLKSIZE} / 4096) + 1] * 2$
 - WADS size of 4-5 cylinders is usually recommended
 - Max TRKs = $((\text{OLDS-BLKSIZE} / 2048) + 1) * \text{OLDS-BUFNO}$

KC110 unit 5 page 12

WADS write concept is new ! See following foil 15
OLDS also ready for striping onto the large vols

Definition of IMS log datasets (2)

LOG Mgmt

- SLDS: System Log Data Set
 - Contains archived log records
 - Dual SLDS recommended
- RLDS: Recovery Log Data Set
 - Contains log records needed to recover databases (DB change and I/O-error records, and records that record the start and end of UOWs)
 - Dual RLDS recommended
- Note that all of these log data set types provide the option of IMS software-implemented *mirroring*
 - If you **do** select to have a dual copy of a data set, make sure that it is not allocated on the same VOL=SER as the primary copy
 - The major purpose of a second copy is to reduce or eliminate the impact of a hardware failure

Specifying IMS logging options

LOG Mgmt

Some **OLDS/WADS attributes** extracted from DFSVSMnn member of PROCLIB

```
OLDSDEF OLDS = (00,01,.....,99),  
MODE = DUAL / SINGLE,  
BUFNO = 3 → 9999,  
DEGRADE=YES / NO  
  
WADSDEF = (0,1,....,9)  
ARCHDEF ALL MAXOLDS(1)
```

- If no DD statements in IMS JCL, then DFSMDA members required for:
 - DFSOLPnn, DFSOLSnn, DFSWADSnn
 - **The use of DFSMDA (Dynamic allocation) for OLDS and WADS data sets is strongly recommended**
- WADS duplexing via WADS = S/D parameter in IMS startup procedure **ONLY – not** in DFSVSMnn
- **Minimum of 3** OLDS subparameter required
- BUFNO, MODE extracted from DFSVSMnn (OLDSDEF)
- **Minimum of 1** WADS subparameter
- OLDS and WADS data sets must be preallocated and cataloged
 - Be careful of multiple data set extents - especially with SMS, only first extent will be used

KC110 unit 5 page 14

“ Be careful of multiple data set extents - especially with SMS, only first extent is used “

Well , pre alloc with “Contig” / continuous avoids that splitting of your requested primary space

Note:

Most of the characteristics associated with IMS Logs, and IMS log buffers are specified in the DFSVSMnn member of IMS.PROCLIB; this is the same member that is also used for database buffer specifications.

- OLDS DASD Logging
- SLDS / RLDS ... TAPE / DASD Logging

WADS Writes and Definition

LOG Mgmt

- The concept of WADS track groups is not used any longer since IMS V12
 - WADS should be sized to provide enough space for any OLDS buffers not yet written at any time plus one track
 - WADS writes are changed from previous IMS versions
 - IMS 12 writes to WADS from previously written log record in the buffer to the last log record in the buffer
 - WADS written in wrap-around fashion
- Performance tipp : WADS could be kept in cache in storage subsystem

Maximum WADS size before IMS V12 with 200 24K buffers:

$(\text{OLDS block size} / \text{WADS segment size} + 1) \times (\text{no. of OLDS Buffers})$

$((24\text{K}/4\text{K}) + 1) \times 200 = \underline{1400 \text{ tracks}}$

Maximum WADS size with IMS 12 with 200 24K buffers:

3390 Model 9 allows 56664 bytes per track

$56664 / 24\text{K} = 2 \text{ blocks per track}$

$200 \text{ buffers} / 2 + 1 = \underline{101 \text{ tracks}}$

KISSLICH CONSULTING

KC110 unit 5 page 15

IMS 12 changes the way that WADS writes are done. The concept of track groups is not used with IMS 12. This changes the calculation for the space required for the WADS and changes the data written by log ahead requests.

In IMS 12 the WADS should be sized to provide enough space for the data in the OLDS buffers which have not yet been written to disk at any time plus one track. This may dramatically reduce the space requirement for the WADS. In previous versions the WADS was sized by using the WADS track group concept. A track group was the OLDS block size/WADS segment size plus 1. A WADS segment size was 2K for OLDS buffers below the bar in real storage and 4K for OLDS buffers above the bar in real storage. The maximum WADS size was $(\text{OLDS block size} / \text{WADS segment size} + 1) \times (\text{number of OLDS buffers})$ tracks. For example, an installation with 200 24K OLDS buffers above the bar could use a WADS with 1400 tracks. With IMS 12, a system with 200 24K buffers would require no more than enough tracks to hold $200 \times 24\text{K}$ plus one track. That would be approximately 101 tracks.

In previous versions of IMS the WADS was written in segments from the OLDS buffers. Successive writes were to different tracks. The scheme is much simpler in IMS 12. Each WADS write is to the next block in the data set. The data written to the WADS includes the data that was not previously written up to the last record in the buffer.

In order to provide the best response times for WADS writes the WADS data should be kept in the cache of the storage subsystem. Since the WADS is written in a wrap-around fashion, this means that the entire WADS should be in the cache for optimum

performance.

Latest news are :

WADS as linear VSAM

Then ZHYPERWRITE= parameter in the new LOGGER section of the DFSDFxxx
PROCLIB member can be used .

DBRC functions (1 of 2)

RECOV Mgmt

- Two types of DBRC used
- *Log-Control*, is mandatory and its primary function is to track IMS log data set usage:
 - Maintains information about OLDS status, for example, INUSE ARCHIVE-NEEDED AVAILABLE, and so on
 - Maintains information on data set names, creation times, and contents of the various RLDS and SLDS data sets
- Optionally, on a case-by-case basis, databases can be registered to DBRC
 - Once registered, DBRC provides the same functions for databases that were called *Share-Control* prior to IMS 6.1
 - These functions include:
 - Assistance in database integrity and recovery
 - Preventing access to databases in a way that could impact recoverability
 - An example is when a job is prevented from updating a database not yet Image Copied

KC110 unit 5 page 16

The use of Database Recovery Control (DBRC) is mandatory in order to allow IMS at initialization to determine which logs were in use when it was last active. - "LOG CONTROL" (in coordination with RDS)

DBRC functions (2 of 2)

RECOV Mgmt

- The functions provided for Registered Databases (continued)...:
 - Records the occurrence of key events that impact DB usability such as:
 - Database Image Copy (ICs), Change Accumulation (CA), Reorganization (Reorg) and Recovery (DBR) Job executions
 - Automatic JCL generation of needed job-streams
 - Database Sharing, with integrity, between IMS systems
 - Intra-processor (same z/OS) or inter-processor (different z/OS) block-level sharing with integrity
 - Changes (in addition to DBRC changes) are required when implementing Data Sharing

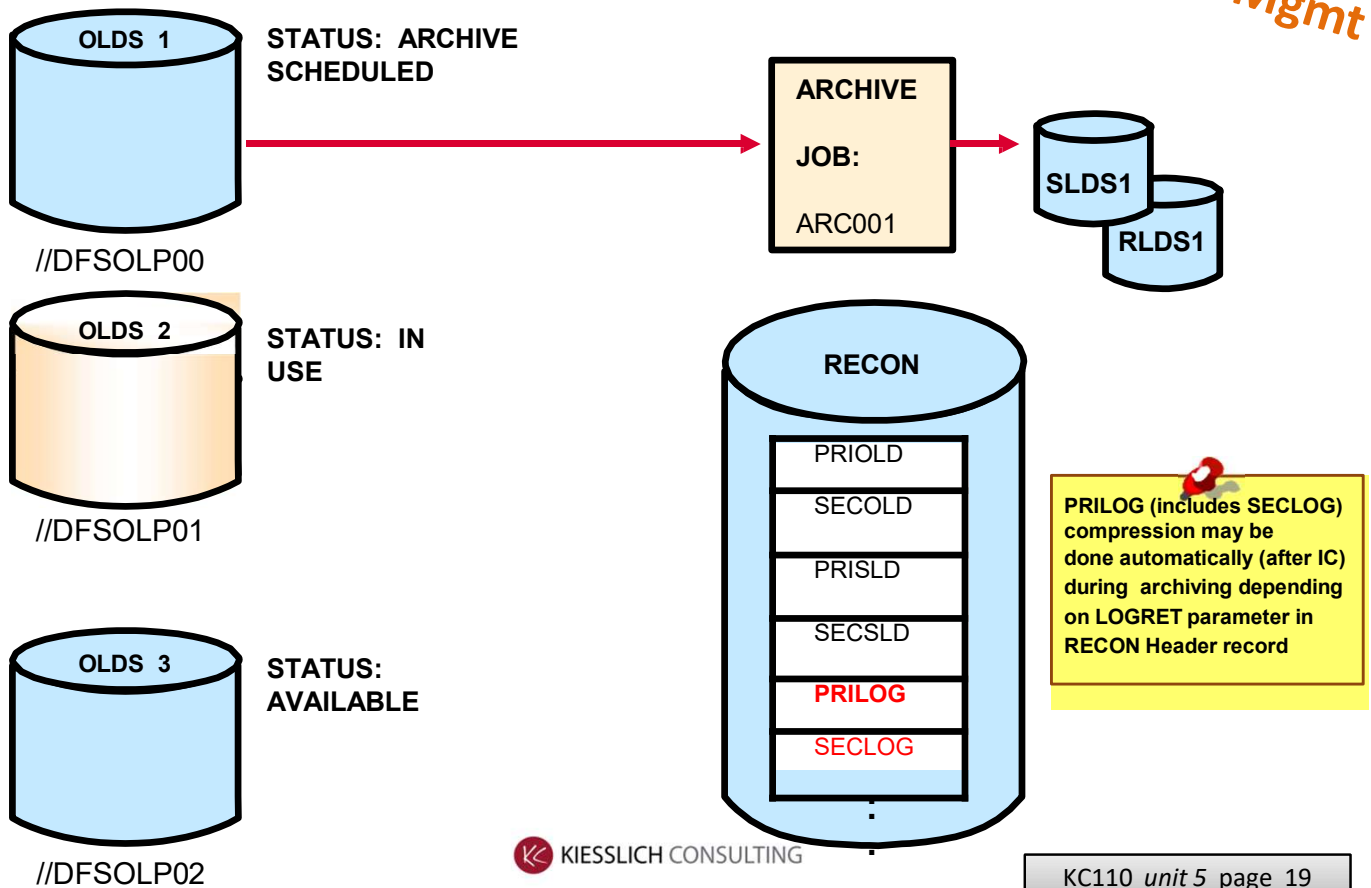
RECON maintained Information (1)

RECOV Mgmt

- Recovery Control Data Sets (RECONs)
- Automatic recording of information about:
 - Online Log Data Set (OLDS)
 - System Log Data Set (SLDS)
 - Recovery Log Data Set (RLDS)
- Generation of utility JCL via DBRC command for:
 - Log Archive
 - Log Recovery

RECON maintained Information (2)

RECOV Mgmt



The minimum information recorded about logs includes:

1. Subsystem ID
2. DDNAME
3. Data Set name
4. Start timestamp
5. Stop timestamp
6. Status
7. Archive timestamp
8. Archive JOB Name

Therefore the PRILOG compression will be triggered by some CMDs / activity

DBRC Database Registration

RECOV Mgmt

- Advantages of Registering Databases:
 - Records recovery-related information in RECON
 - Example: Provides information on which RLDS has log data for each database data set (DBDS) registered with DBRC
 - Provides databases with enhanced integrity by identifying valid recovery points for databases
 - DBRC GENJCL feature Generates JCL for recovery-related utilities:
 - Used to select correct data sets to be used by Recovery and Image-Copy
 - Consolidates changes of log records by creating the correct input for *Change Accumulation* Utility Jobs
 - Reduces or Eliminates errors
 - Verifies that JCL used with IMS Utilities has valid input specified
- Registration of databases is performed through DBRC commands
 - DBRC is covered in detail in the 32 hrs DBRC class CM20xx



KIESSLICH CONSULTING

KC110 unit 5 page 20

DBRC was originally designed to record recovery related information for IMS databases and to automate the recovery of the databases when required.

DBRC still permits these functions and is specified on a database-by-database basis through the process of *Registering* databases.

This registration process involves running a DBRC utility with control statements that specify information about the database (for example, DBD Name, Dataset Name, retention period for recovery information, and so on) being registered.

„recovery related“ :

recover to a point back in time (IC + maybe LOGs or Cas)

forward recov to time of abend (IC + Cas / LOGs)

CA: cumulated “after image“ info

GENJCL.USER for your own JCL ☺

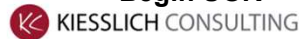
Checkpoint ID Table – RDS (1 of 2) Restart (RECOV) Mgmt

Assume System Failure at 12:45 – restart must examine logs starting from 9:00 in support of PST 2 backout

Checkpoint ID Table			
PST	prev Sys-CHKPT	Appl - Commitpoint	VOL=SER
1	9:00	9:05 end	=
2	→ 9:00	9:10 begin	
3	→ 10:00	10:05 begin	
4	11:00	11:05 end	
5	→ 12:00	12:05 begin	
.			
.			

Most recent System (simple)
Checkpoint prior to PST UOW start

where: end = End UOR (Synchpoint)
begin = Begin UOR



KC110 unit 5 page 21

Restart Data Set (RDS):

Contains the Checkpoint ID Table to determine restart checkpoint

Keeps track of all PSTs (up to 999) plus additional entries for other terminating (/CHE FREEZE) and non-terminating (/CHE SNAPQ) checkpoints

Can be formatted at startup

Restart Algorithm:

Locate and read the most recent system checkpoint prior to the oldest application program Commit Point for all active PSTs at the time of failure

Read log forward and restore system to status at the time of failure for all

Uncommitted DB change records collected in virtual storage

OLDS in use at the time of failure closed and new OLDS opened

Uncommitted DB changes backed out, Uncommitted Output-Messages backed out, TX MSGs (if specified) queued for scheduling

Checkpoint ID Table – RDS (2 of 2)

Restart
(RECOV)
Mgmt

```
IPCS OUTPUT STREAM -----
CHECKPOINT ID TABLE

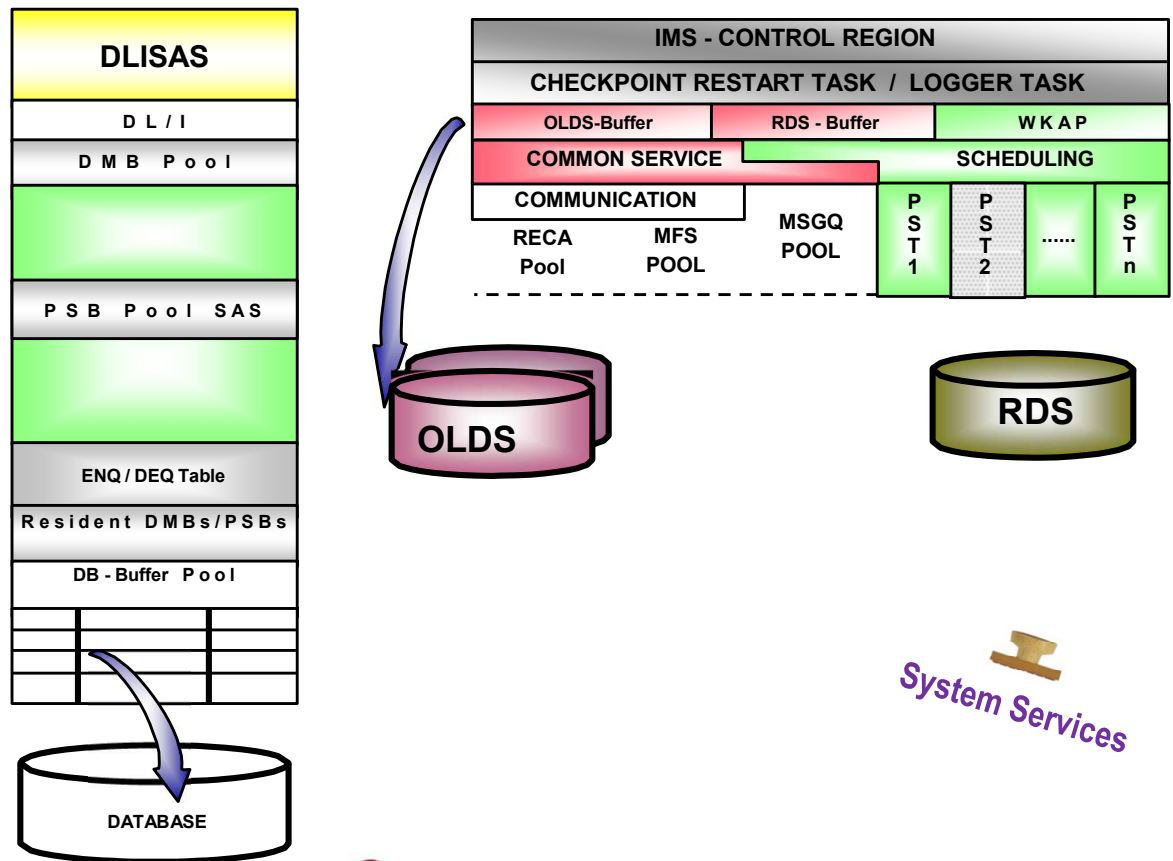
18F4A000 00000000 00001000 1A010040 00000000 *.....*
18F4A010 00000000 00000000 00000000 00000001 *.....*
18F4A020 00000000 00000000 00000000 C2C3D7E3 *.....BCPT*
18F4A030 00000000 00000000 CABC11DE 5E31B0CB *.....;...*
18F4A040 D9E5C5E3 02040200 05660000 00000000 *RVET.....*
18F4A050 C2C3D7E3 18F4A000 00000001 00000366 *BCPT.4.....*
18F4A060 E2C9C4E7 18F49000 00000002 00000000 *SIDX.4.....*

Command ==>                                SCROLL ==> CSR
F1=HELP   F2=SPLIT  F3=END    F4=RETURN  F5=RFIND  F6=MORE   F7=UP
F8=DOWN   F9=SWAP   F10=LEFT  F11=RIGHT  F12=CURSOR

. . . . .
Menu Utilities Compilers Help

BROWSE      IMS12A.RDS                      Line 00000000 Col 001 080
***** Top of Data *****
.....BCPT.....-ú;.çôRVET.....Ã.....
.....ö.....&.....SIDX.....-ú;.¼»...°... IMSA
.....ü.....Q.....LCRE.....-ú;.:.....ç
Command ==>                                Scroll ==> PAGE
```


Syncpoint processing (1 of 2)



Notes:

At UOW syncpoint, all logging is completed and all database updates are completed before response messages are sent, before scheduler related and locking related resources can be freed

Resources (scheduler / Lock mgr) - PSB in Work pool, DMBs in WP , Locks held, ...

Syncpoint processing (2 of 2)

- Syncpoint processing common to IMS/TM, CICS and BMP:
 - Changed data committed:
 - Full function changes written to database
 - DEDB changes become eligible to be written (asynchronously)
 - Locks released - changed data available for other users
 - Pool space is retained until program **termination** (except for CICS, see below)
- Commit point processing specific to IMS/TM:
 - MSDBs updates and SPA changes made permanent
 - Output messages enqueued to final destination
- Commit point processing specific to CICS:
 - Pool space becomes eligible for other use
 - PSB and DMBs marked inactive and PSBW space freed
 - Done even if the next PSB scheduled is the one we are terminating here
 - CICS thread released at PSB termination
 - CICS thread - PST assignment will **not** be released
 - Dependent on workload, additional threads (above MINTHRD) and their associated PSTs are freed


System Services

Here are some of the key events that occur as part of an application committing of a UOW.

Implicitly next GU IOPCB

Explicitly end, CHKPT calls, other (ROLB , ...) calls

Multi UOR in one UOW

...

IMS Storage Management POOLS (1 of 2)

- IMS has different storage manager modules, they are:
 - The old Pool Storage Manager - DFSISMNO
 - Also called CBT (Control Block Table) Storage manager
 - The new Pool Storage Manager - DFSPOOL
 - Also called the Dynamic Storage Manager
- The old Pool Storage Manager manages pools that are fixed in size at IMS Initialization:
 - Space reassignment and reclamation are not sophisticated
 - IMS can run out of space for pools managed by this method
 - Unfortunately, all Scheduler-related pools (PSB, DMB, PSBW, others as DLMP, DPSB, DBWP, EPCB, MAIN and so on) use this old Manager


System Services

PSBW, DLMP, DPSB, DBWP, EPCB, MAIN a.o. will still be managed with the old pool manager DFSISMNO.

Some IMS storage pools are managed by a so called pool manager, DFSPOOL. Those pools are CIOP, HIOP, SPAP, CESS, EMHB, FPWP, LUMC und LUMP. Pools as QBUF, QBFL, MFBP und EPCB will be allocated from the new Pool Manager but not controlled.

IMS Storage Management POOLS (2 of 2)

- The New Pool Storage Manager is better than the old one, at responding to varying demands for storage:
 - A user-specified amount of space is allocated at IMS initialization
 - Fixed length buffers / space chunks are assigned in a *best fit* manner for variable length requests
 - Pools managed by this Storage Manager can dynamically grow in size, if necessary
 - The CIOP and the HIOP are two pools managed this way (and SPAP, CESS, EMHB, FPWP, LUMC und LUMP)
- Third (newest) type of storage management is done by DYNAMIC CONTROL BLOCK STRUCTURE (CBS)
 - Allocating blocks on page boundaries (4k) – minimizes paging
 - Can release unused **IPAGES** : If no blocks are allocated out of the IPAGE , it is freed (timing interval scans)
 - CBS has header and entry portion (see DFSCBTS and CBTE)


System Services



KIESSLICH CONSULTING

KC110 unit 5 page 26

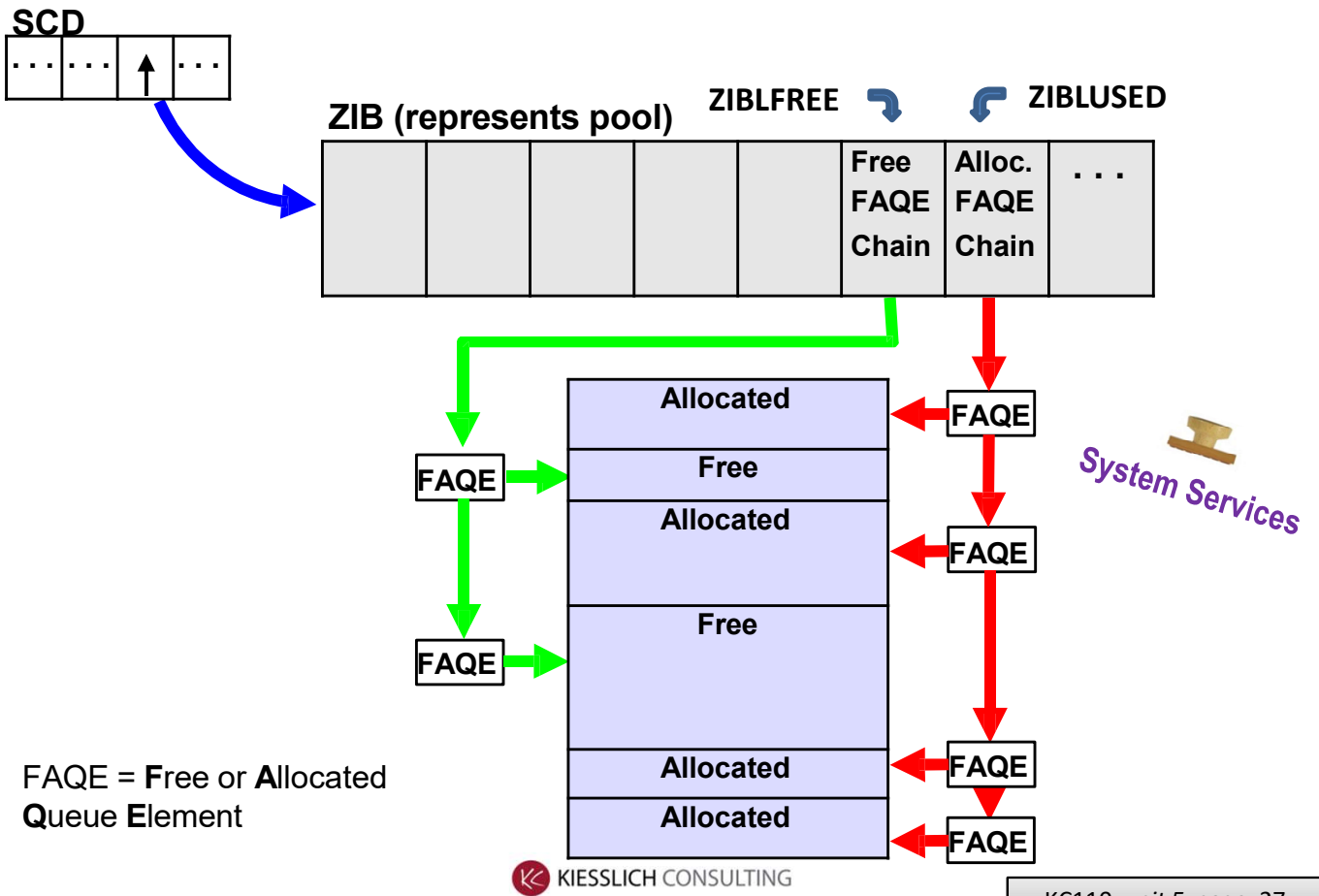
The new pool manager DFSPPOOL is a dynamic storage manager using fixed length buffers for variable length requests. The best fitting fixed buffer is taken for a request. The total size of the pool will vary when IMS is running.

Instead of FAQE's, the new storage manager makes it with bitmaps. The Diagnosis Guide and Reference has a nice description about the new and old storage manager.

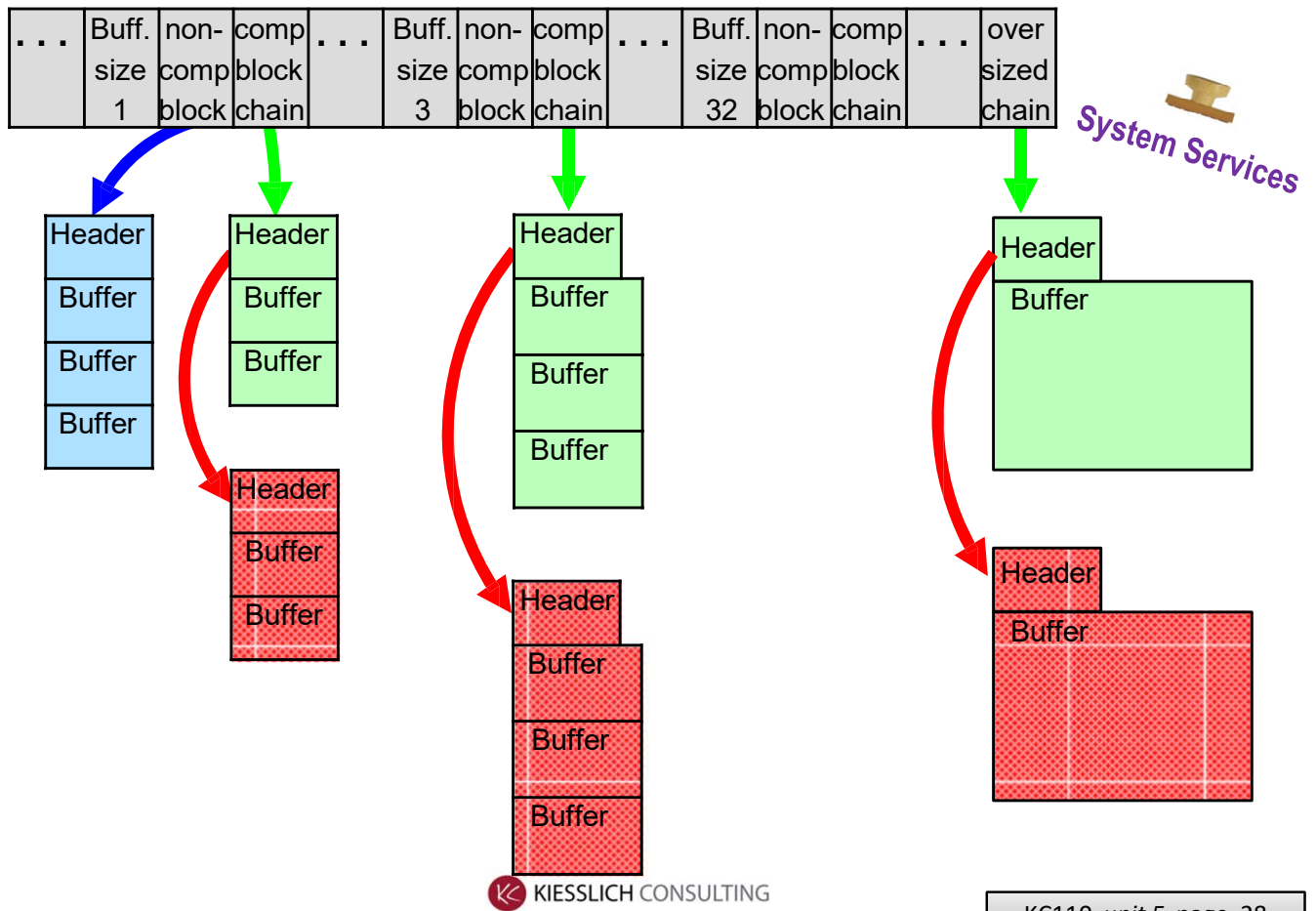
Most control blocks (except the very old and first ones) are in that way built, the DYNAMIC CONTROL BLOCK STRUCTURE (CBS) . It also attempts to keep the CSA storage usage to a minimum, releases blocks when not in use and minimizes paging by allocating blocks on page boundaries.

The CBS is built on an IPAGE basis. That is, a page is allocated and the control blocks are carved out of the page. The reason in using the term 1IPAGE1 is that the page size may not necessarily be 4k. but a size determined by the user of the block. When a request for a particular block is issued and the block is not available, another IPAGE is allocated and the block is returned to the caller. During time intervals, the IPAGES are scanned. If no blocks are allocated out of the IPAGE , it is freed.

Old Pool Storage Manager

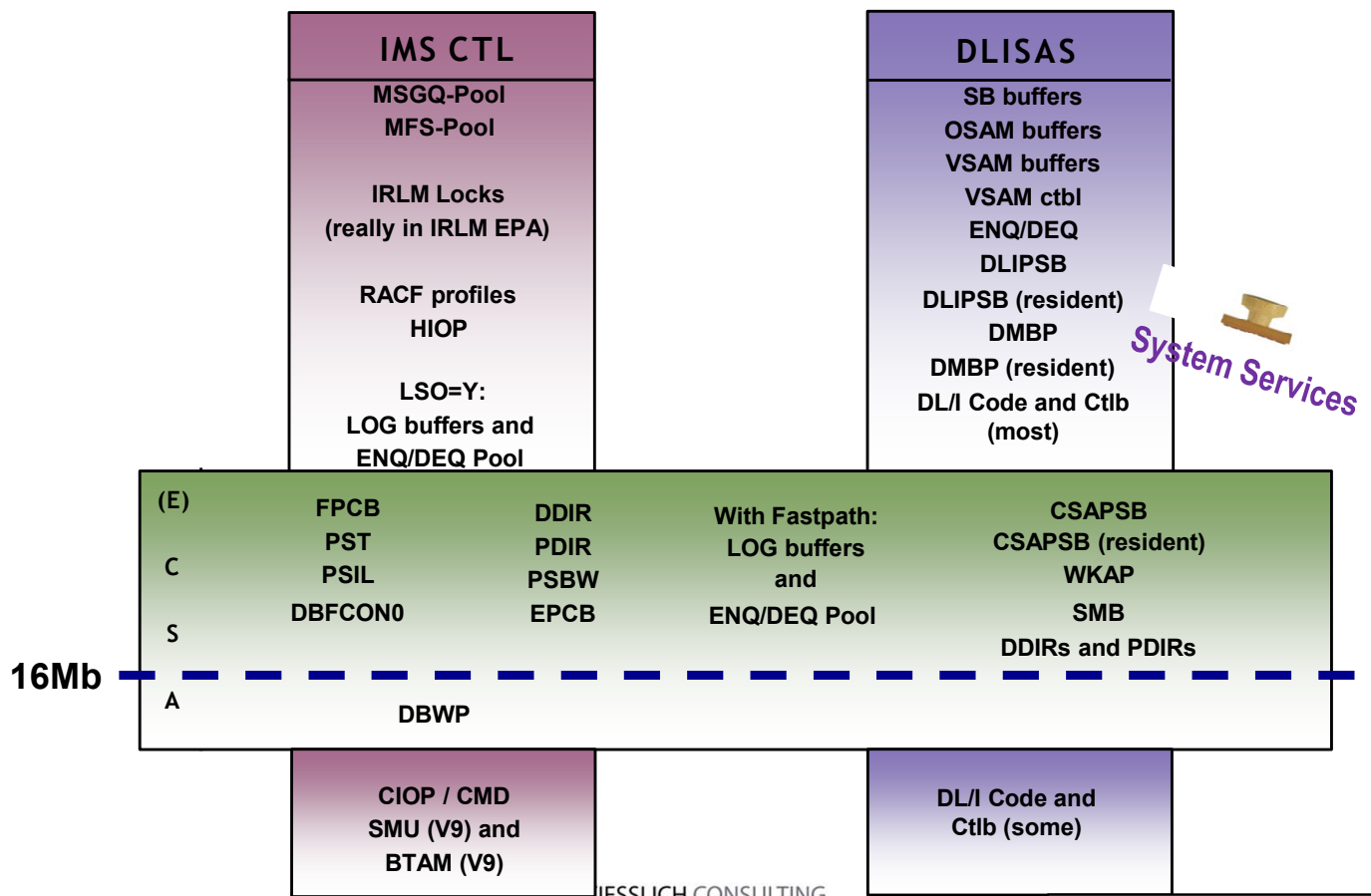


New Pool Storage Manager



POOLHDRs , pointed from hashtable which is pointed from the SCD (SCDSMHT).
Some pools, DB and LOG bfrs are now "above the bar" exploring the 64bit addressable areas

z/OS: IMS CSA and Priv. Storage use



RIESSLICH CONSULTING

KC110 unit 5 page 29

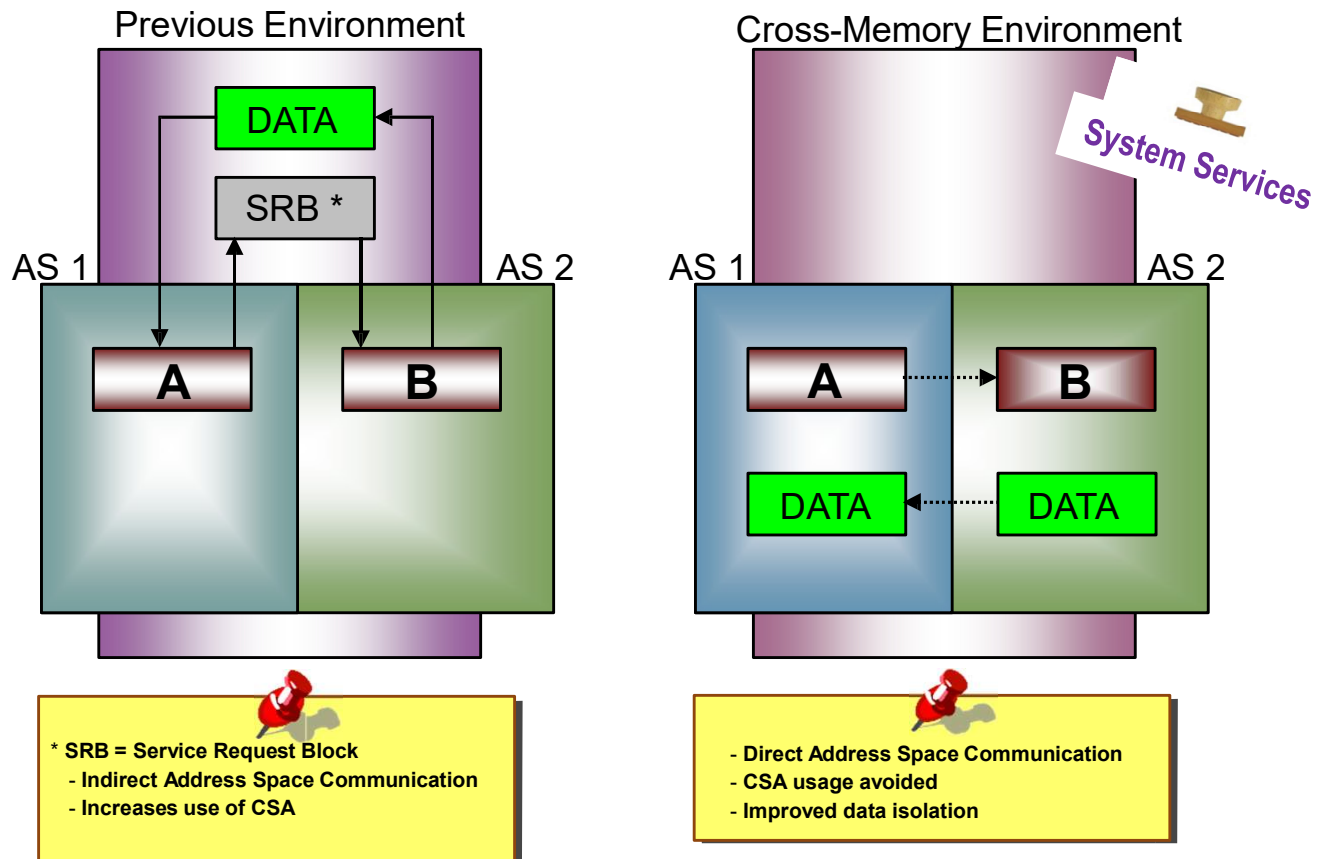
Note recent changes : pools above the bar (64bit)

VSAM ctbl blks NOT in DLISAS , they reside in z/OS (VSAM) , only BFSP are in DLISAS

Notes:

This is a summary of where various storage areas that we have discussed are stored. A slight misrepresentation is that IRLM locks are stored in Extended Private storage of the IRLM address space, not extended private of Control region storage as is depicted here for simplicity.

z/OS Cross-Memory Services with IMS



KISSLICH CONSULTING

KC110 unit 5 page 30

Without cross-memory services, when a program in one address space (Program A) has to call a program in a second address space (Program B),

Program A first obtains supervisor state and then schedules a system routine, which is subsequently given control. The system routine can then permit Program B to begin. With the cross-memory facility, Program A invokes a cross-memory service linkage mechanism directly and begins execution.

As another example, let's say we want to move data from address space 2 (AS 2) to address space 1 (AS 1). Without cross-memory services, B first obtains supervisor state, then writes data into Common Service Area (CSA) and schedules a supervisor program. When the supervisor program executes, it writes data from the CSA to AS 1. However, using the cross-memory service facility, Program B simply moves the data between the user address spaces.

Neither Program A nor Program B has to be in supervisor state. Keeping the data and code in each address space's private area improves isolation and reduces the demand on system virtual storage (shared CSA). This isolation of data enhances the system's integrity.