z/OS Basics: Virtual Storage Management (VSM) Overview

Elpida Tzortzatos – email: elpida@us.ibm.com

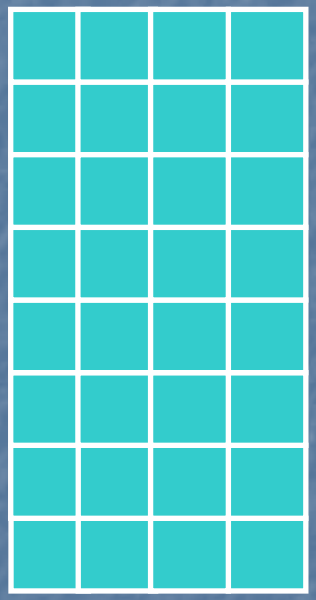
Agenda

- z/OS Memory/Storage Types
- z/OS Memory Managers
- 31-Bit VSM
  - Basics
  - Recent Enhancements
- 64-Bit VSM/RSM
  - Basics
  - Recent Enhancements
- Appendices
  - CSA Common Storage Tracker

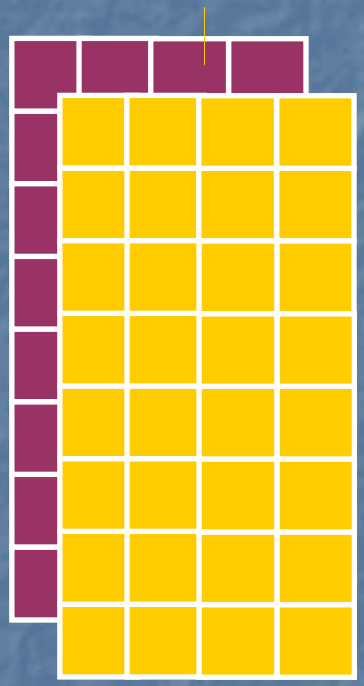
z/OS Memory/Storage Types

z/OS Memory Types

- Memory management views



Real storage ***frames*** (physical CPU memory)



Virtual ***pages*** in multiple address spaces (created through DAT)

Auxiliary ***slots*** on (DASD) volumes (paging datasets)

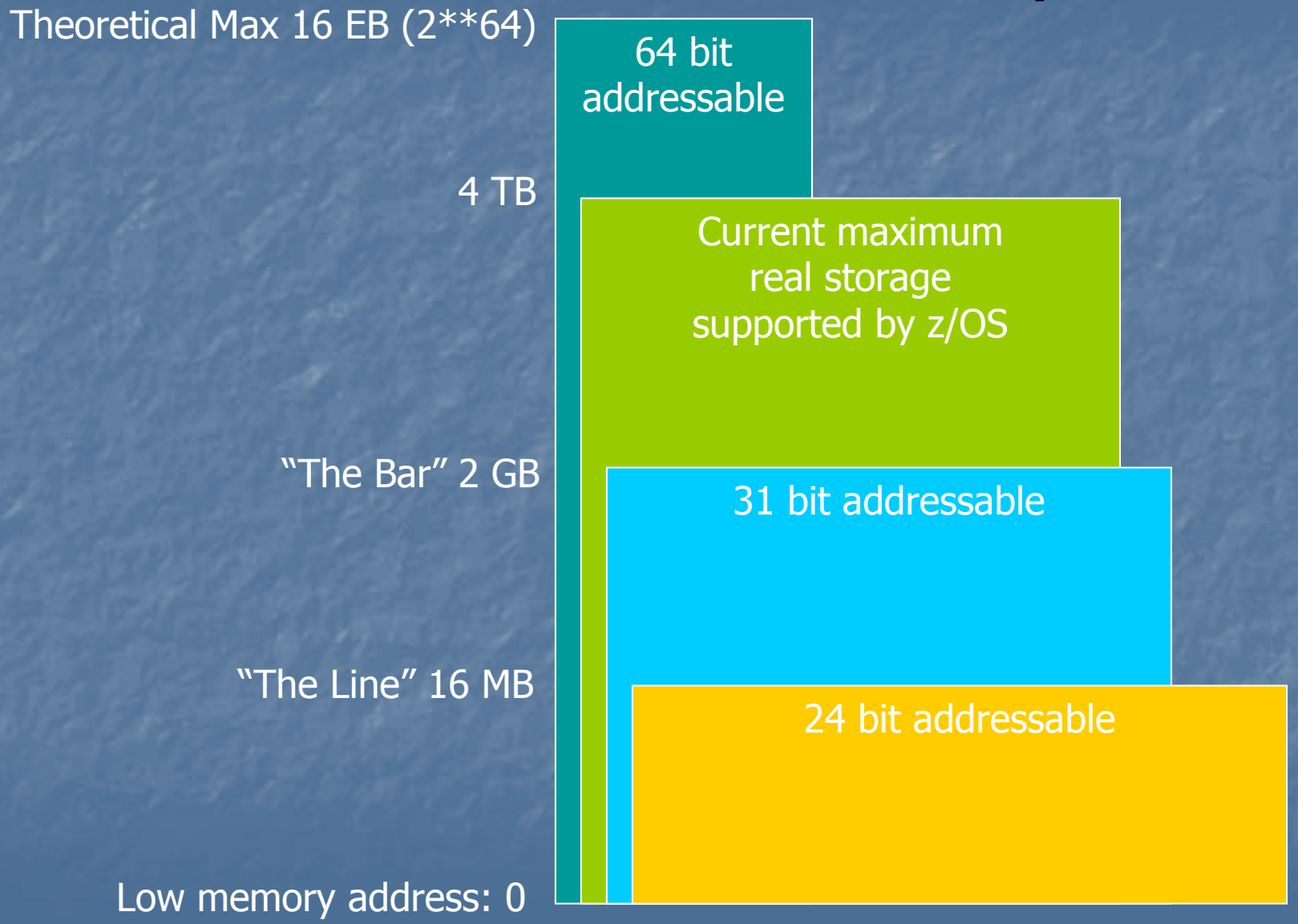
z/OS Memory Controls

- Real
  - Mixture of z/OS internal thresholds and system-wide parameters
- Virtual
  - Customer-specified policies via JCL, System Management Facility (SMF) parameters and exits
  - Generally either prevent work exceeding the limits from starting or causes work to terminate if it exceeds a limit during execution
  - Process-level granularity
- Aux
  - z/OS internal thresholds coupled with paging dataset configuration: number of datasets, size of each

Storage Types (all with respect to z/OS's point of view)

- Virtual memory/storage
  - DAT-on addressing
  - Explicitly allocated either by the system or applications
  - 24, 31, or 64 bit residency mode (rmode)
  - 31-bit virtual is allocated/freed in multiple of 8 byte chunks
  - 64-bit virtual is allocated in 1MB multiples on a 1M boundary
  - Virtual storage attributes are specified when virtual storage is allocated
    - Fixed (pinned), DREF, pageable, 1MB pages
    - When physical resources (real memory/paging space) are assigned to virtual storage depends on the virtual storage attributes
- Real memory/storage
  - DAT-off addressing, Real Space ALET, I/O, some privileged op-codes specify real addresses
  - 4 KB page frames (often: just "frames"), 1MB page frames (often just "large pages")
  - 24, 31, or 64 bit residency mode (rmode)
- Each memory allocation request to z/OS specifies allowable virtual and real rmodes
  - Based on addressing mode of application and whether or not the "real view" will be used
  - Example: the ordinary 31-bit application requires 31-bit virtual pages but the frame backing each page may have a 64-bit real address
- Expanded storage
  - No longer used by z/OS
  - Mid-1980s to early 2000s, electronic storage with characteristics between real and auxiliary in terms of response time and monetary cost
  - Functioned mostly as a fast synchronous paging device
- Auxiliary (storage, on Disk/DASD)
  - Used for paging and swapping
  - 4 KB slots in one or more paging datasets
  - Invisible to applications

Virtual/Real Map



z/OS Memory Managers

z/OS Memory Managers: VSM

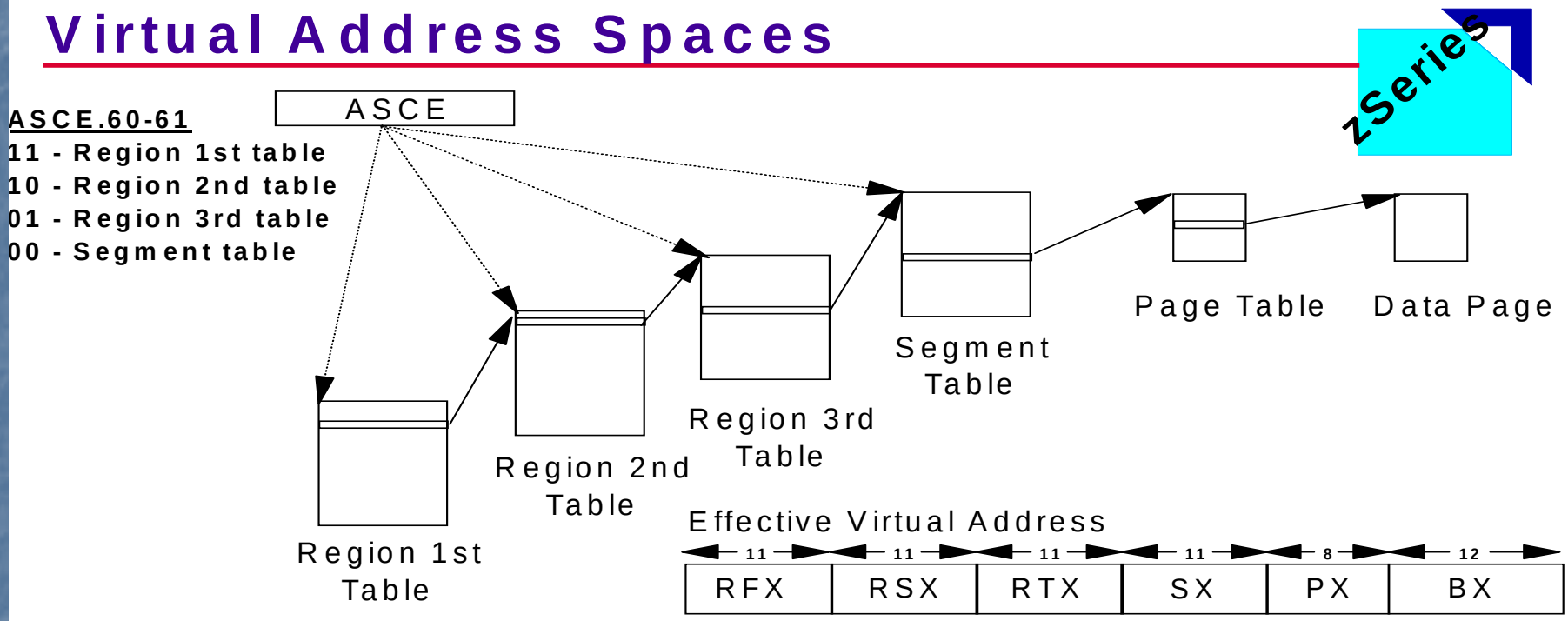
- Virtual Storage Manager
- Address Space-centric view of the system and processes
- Objectives
  1. Control the allocation/deallocation of 31/64-bit virtual storage addresses
  2. Obey policies imposed on it by customer specified limits
     1. Each installation can use virtual storage parameters (in Sys1.Parmlib,JCL, SMF) to specify how certain virtual storage areas are to be allocated to programs
  3. Efficiency – minimum overhead per request
- Associate a storage protection key with each virtual storage block requested
- Maintain storage use information by generating SMF records.

z/OS Memory Managers: RSM

- Real Storage Manager
- Frame-centric view of the system and processes
- Objectives
  1. Keep the system up
  2. Obey policies imposed on it by the customer, SRM
  3. Efficiency – minimum overhead per request
  4. Manage 64-bit storage requests
- Resource pools
  - Expanded frames (historical only)
  - Real storage frames
    - Individual frames
    - Frame pairs: 2 contiguous in real on particular boundary
    - Quad frames: 4 contiguous in real on particular boundary
- Major policy knobs that control its behavior
  - Optional per-process minimum/maximum number of frames
    - Processes below minimum stolen from last, above maximum first
  - System-wide thresholds controlling paging
    - Begin stealing when less than X frames unallocated, steal until Y frames available

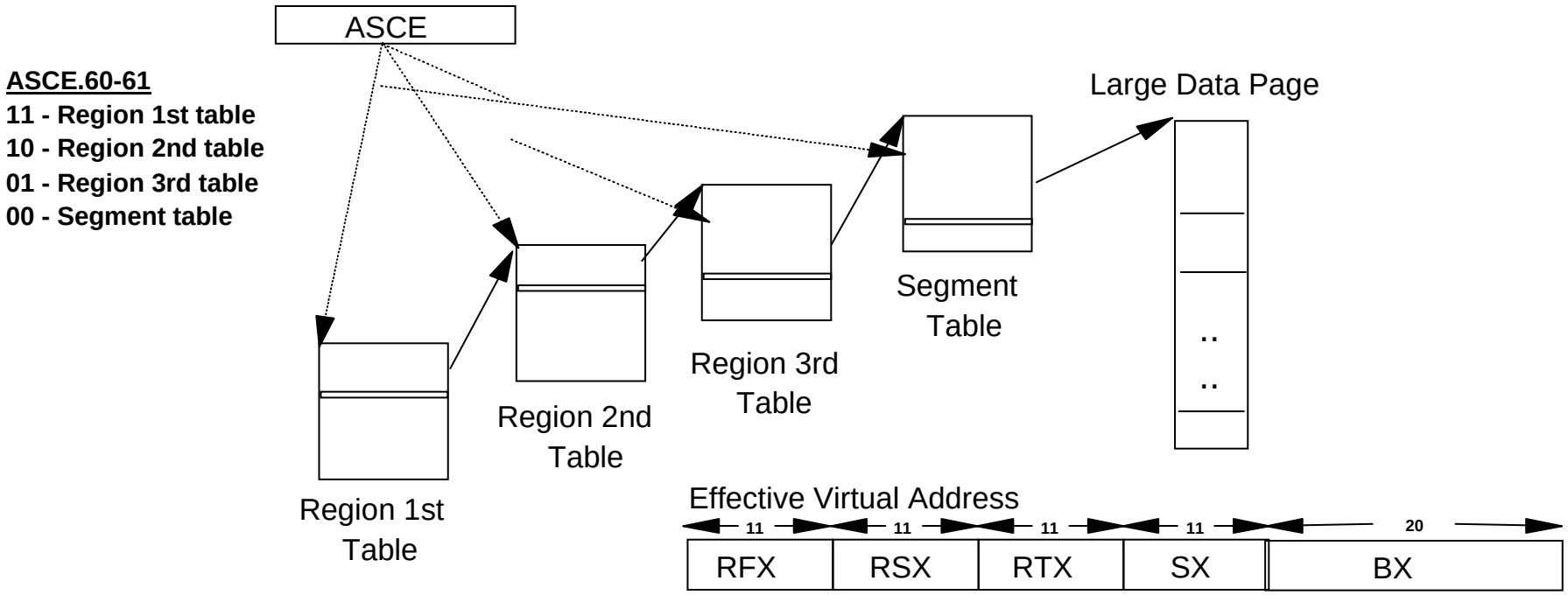
Dynamic Address Translation...

Virtual Address Spaces



- Translation can start at R1T, R2T, R3T or SGT
- The starting point of the translation is designated in the Address Space Control Element (ASCE.60-61)

Large (1MB) Page DAT



- Translation can start at R1T, R2T, R3T or SGT
- The starting point of the translation is designated in the Address Space Control Element (ASCE.60-61)

z/OS Memory Managers: ASM

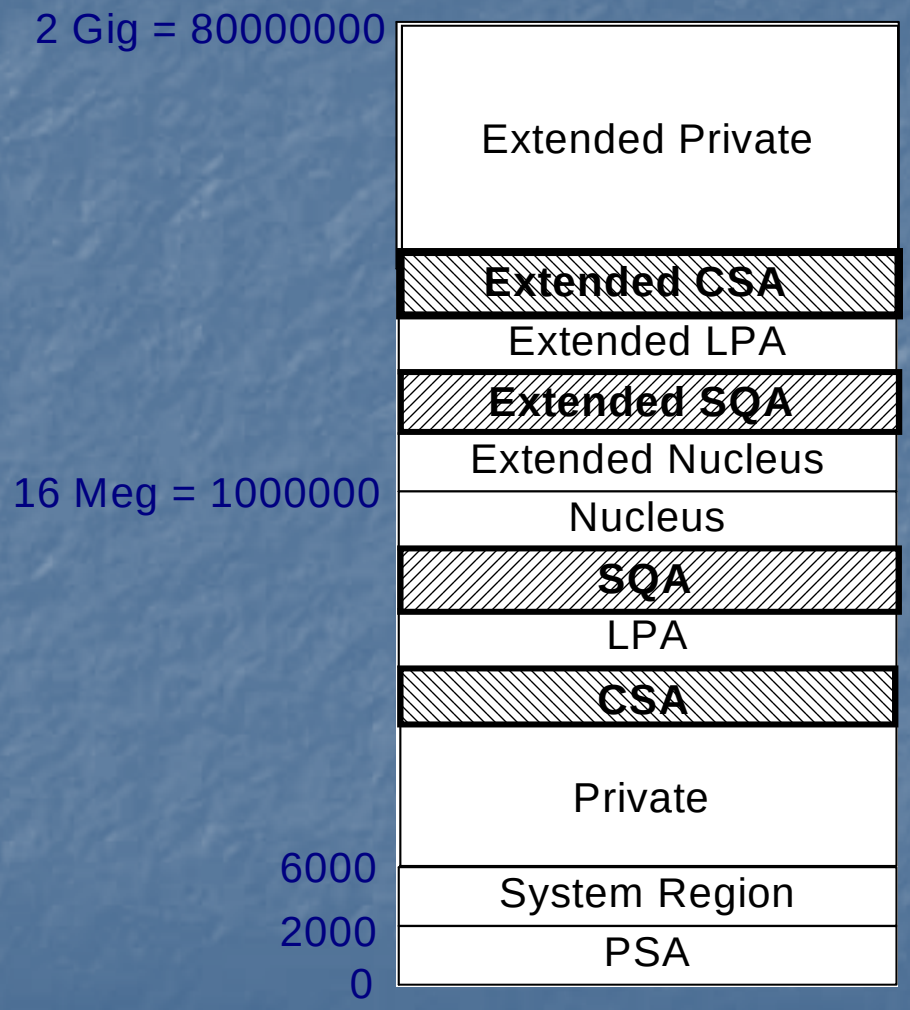
- Auxiliary Storage Manager
- Paging-centric view of the system; virtually no process awareness
- Objectives
  1. Efficiency – minimum overhead per request
     - Highly optimized
       - Slot allocation efficiency falls dramatically once utilization of a dataset rises above 30%
       - ASM tries to equalize the number of slots allocated across the datasets, not their % allocated
     - Relies on other managers to only give it sensible requests
     - An auxiliary storage slot is allocated to a page when the page is paged out
     - An aux slot is freed when the page is freed or reused when the page is paged out again
     - Periodically RSM calls ASM to free slots for pages that are changed in real storage
- Resource pools
  - 1-n Paging datasets formatted into 4 KB slots
    - Called paging datasets, but used for both paged out and physically swapped frames
- Major policy knobs that control its behavior
  - Number of paging datasets and size of each
    - Datasets may be different sizes

31-Bit VSM

Virtual memory/storage

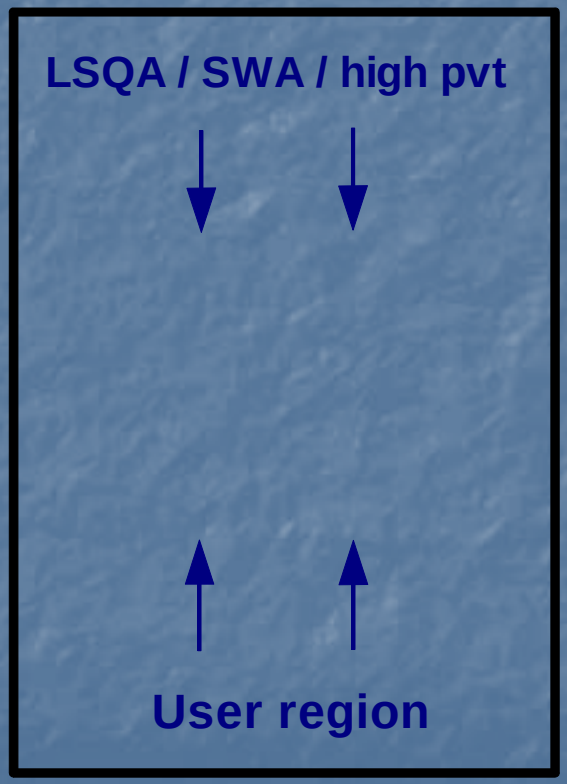
- DAT-on addressing
- Explicitly allocated either by the system or applications
- 24, 31, or 64 bit residency mode (rmode)
- 31-bit virtual is allocated/freed in multiple of 8 byte chunks
- 64-bit virtual is allocated in 1MB multiples on a 1M boundary
- Virtual storage attributes are specified when virtual storage is allocated
  - Fixed (pinned), DREF, pageable, 1MB pages
  - When physical resources (real memory/paging space) are assigned to virtual storage depends on the virtual storage attributes

31-Bit Address Space Memory Map



- **Global storage includes Nucleus, Extended Nucleus, SQA, ESQA, LPA, ELPA, CSA, and ECSA**
- **Global storage managed by VSM includes SQA, ESQA, CSA, and ECSA**
- **Upper boundary of ECSA storage and lower boundary of CSA storage are always on a Megabyte boundary**
- **Sizes of SQA, ESQA, CSA, and ECSA storage are specified at IPL time via the IEASYSxx parmlib member**

Local Storage Area



Authorized storage (LSQA, SWA, and high private) is assigned from the high end of the local storage box. In other words, this storage grows from the top of the box down.

Unauthorized storage (user region) is assigned from the low end of the local storage box. It grows from the bottom of the box up.

z/OS Memory Managers: VSM

VSM Storage Management Rules

- MVS manages storage through the use of subpools designed to accommodate a variety of storage needs
- Storage is allocated or assigned to a subpool in one page (4K) multiples
- Storage belonging to different **subpools** cannot occupy the same page
- Storage with different storage **keys** cannot occupy the same page
- Storage belonging to different **TCBs** cannot occupy the same page

z/OS Memory Managers: VSM

VSM Storage Management Rules

- When there is not enough storage above the line to fulfill an above the line storage request, VSM will attempt to honor the request from below the line instead
- LSQA / SWA / high private pages may not intermix with user region pages
- Unless otherwise directed on the GETMAIN request, VSM will give out storage at the high end of the page first

z/OS Memory Managers: VSM

Private Subpool Attributes

- **Subpool numbers** 0 - 255
- **Storage protection** Keys 0 - 15

- **User Region** subpools
  - 0 - 132, 250 - 252
  - TCB-related
  - Keyed storage
  - **Un**authorized
  - General purpose subpools

- **High Private** subpools
  - 229, 230, 249
  - TCB-related
  - Keyed storage
  - Authorized
  - Special authorized application storage needs

- **LSQA**
  - 255 (mainly)
  - Fixed, key0 storage
  - Address space-related, not TCB-related

*** See MVS Diagnosis: Reference , Chapter 8, for additional subpool information.**

Virtual Storage Allocation

- GETMAIN, STORAGE OBTAIN, or CPOOL macro required for virtual storage allocation.
- GETMAIN will get storage from subpool 0
- STORAGE OBTAIN obtains storage in the primary address space (by default) or in the address space defined through the ALET parameter
  NOTE: Compared to GETMAIN, STORAGE OBTAIN provides an easier-to-use interface and has fewer restrictions. For programs running in AR mode or cross-memory mode use the STORAGE OBTAIN macro to obtain storage.
- Cellpool (carve storage as wanted after doing an initial GETMAIN - specify size of storage needed at each time)

GETMAIN and STORAGE OBTAIN Using LOC Option

- LOC parameter on GETMAIN macro and/or STORAGE OBTAIN indicates location of requested virtual storage and real storage when the page is fixed.
- Allows storage to be acquired anywhere in the 2 gigabyte address range.
- Caller can be in 24, 31, or 64 bit AMODE
- All values and addresses are treated as 31-bit values and addresses.
- Specifying LOC(xx,64) indicates that central storage (real) can be located anywhere in 64-bit storage.
- The LOC parameter is only valid for the following GETMAIN options
  - RU - Register, Unconditional (get storage, if not available, ABEND)
  - RC - Register, Conditional (get storage, return code, won't ABEND)
  - VRU - Variable, Unconditional (between 200 and 500 bytes long, will try to give 500 at worst, give 200, if not 200 ABEND)
  - VRC - Variable, Conditional (if not minimum, get return code)

Returning Virtual Storage

- STORAGE RELEASE releases storage in the primary address space (by default) or in the address space defined through the ALET parameter
- FREEMAIN options
  - Register, Unconditional
  - Register, Conditional
- To free storage above the 16 MB line the RU/RC options of the FREEMAIN macro must be used.

Obtaining Storage via CPOOL

- The CPOOL macro provides faster storage management than GETMAIN macro
- The following services are available:
  - Create a Cell Pool (Build)
    - ?CPOOL (BUILD) . . . ;
  - Obtain a Cell from an existing pool
    - ?CPOOL (GET) COND . . . ;
  - Extend a Cell pool, if necessary
    - ?CPOOL (GET) UNCOND . . . ;
  - Return a Cell to a Cell pool
    - ?CPOOL (FREE) . . . ;
  - Free an entire Cell Pool
    - ?CPOOL (DELETE) . . . ;

VSM GETMAIN Changes in z/OS 1.10

- **Problem:**
  - ***A large number of datasets are opened in the DB2 address space. This causes VSAM to do a lot of GETMAINs/STORAGE OBTAINs** in subpool 252.***
    - **This creates a long chain of DQEs for subpool 252 in the DB2 address space. Long DQE chains cause performance problems**

*** Subsequent references to GETMAIN also apply to STORAGE OBTAIN**

VSM GETMAIN Changes in z/OS 1.10

- **Solution**
  - ***Allocate virtual storage described by DQEs for low private from the bottom of the page up, instead of from the top of the page down***
  - ***This change in VSM allocation processing allows for a new allocation request to be merged into an existing DQE since the address range is contiguous; only the DQE size has to change***

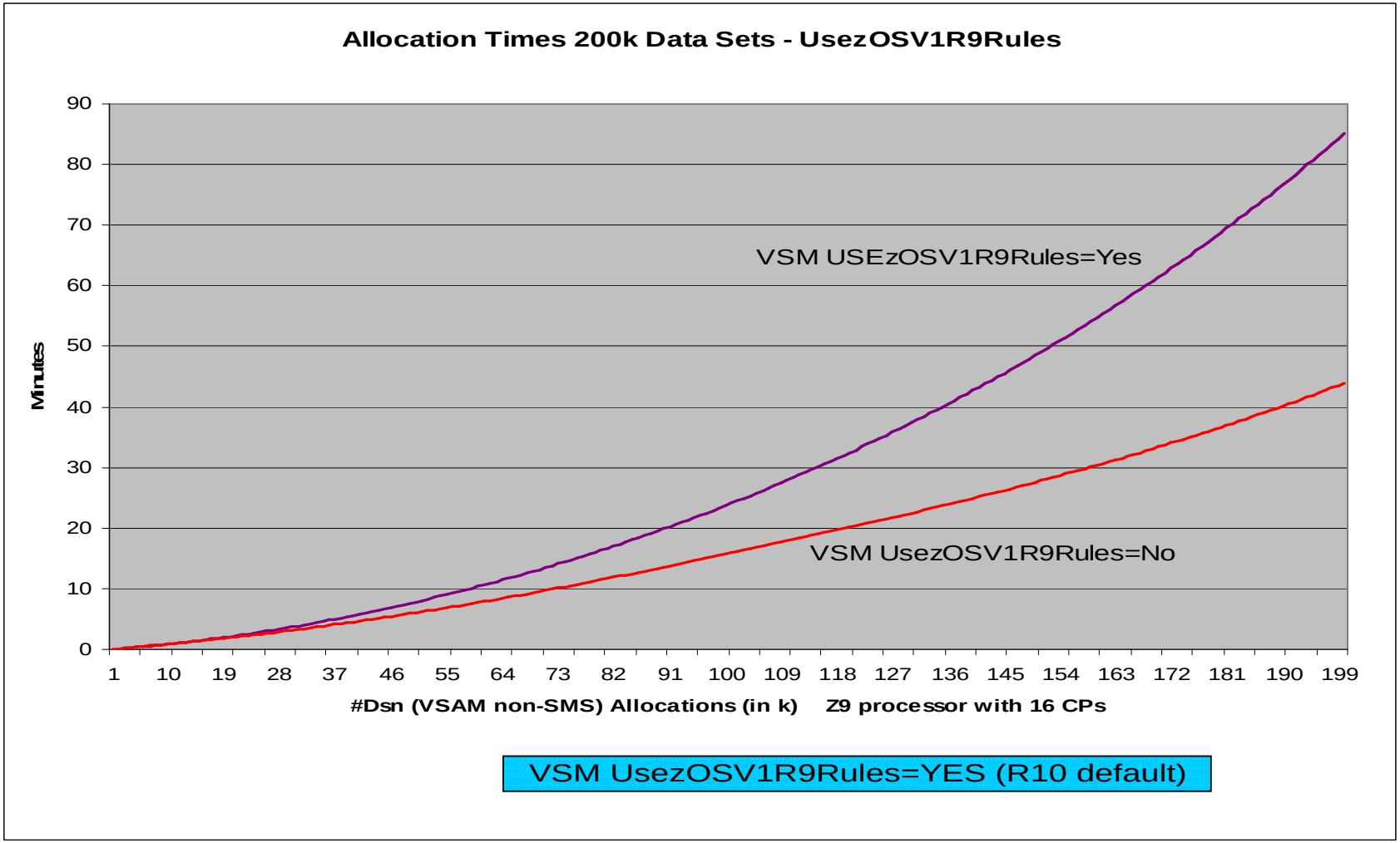
VSM GETMAIN Changes in z/OS 1.10

- **Implementation Details (via DIAGxx)**

With APAR OA27291 / PTF UA45583 installed:

- ***To enable this new behavior, code the following in the active DIAGxx member:***
  - VSM UsezOSV1R9Rules(No)
- ***To revert back to the old algorithm of allocating virtual storage in low private subpools, use "set diag=xx" where DIAGxx specifies the following:***
  - VSM UsezOSV1R9Rules(Yes) – this is the z/OS 1.10 system default
- ***Allocations are performed according to the current setting for UsezOSV1R9Rules; prior allocations are not affected by changing the DIAGxx value***

VSM Changes in z/OS 1.10



VSM GETMAIN Changes in z/OS 1.10

- **How can this change affect you?**
  - ***Properly coded programs can benefit from these changes. Some, such as DB2, may get a significant performance benefit.***
  - ***This change can have a negative affect on some programs which have made unwarranted assumptions about internal VSM behavior.***

    1) **These changes may give the perception in some cases that storage is not being cleared to zero as it previously was. However, storage is cleared by the system no differently in z/OS 1.10 than it was previously.**
    2) **These changes mean that a program cannot assume that a GETMAINed area ends on the last byte of the page; while this was never guaranteed, it was a common VSM behavior prior to z/OS 1.10.**

VSM GETMAIN Changes in z/OS 1.10

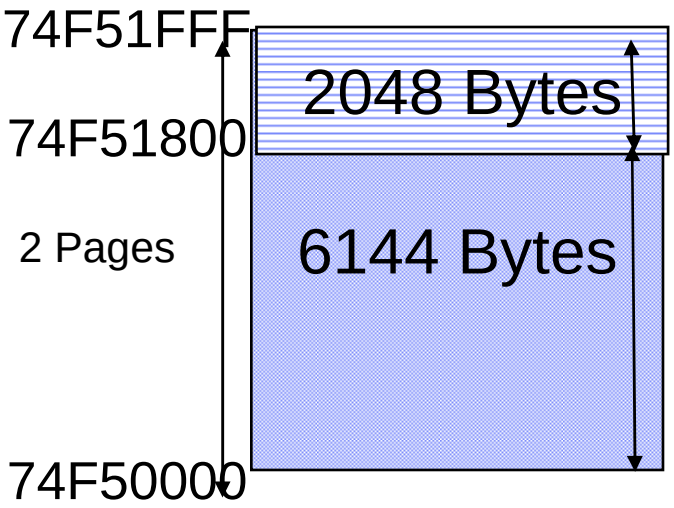
- **Cautionary Example #1**
  - ***Do not assume storage is cleared to zeroes unless the GETMAIN either***
    - **Is for 8192 bytes or more from a pageable, private storage subpool.**
    - **Is for 4096 bytes or more from a pageable, private storage subpool, with BNDRY=PAGE specified**
    - **Specifies CHECKZERO=YES and return code is 0.**
  - ***Storage requests that previously returned an address on a freshly GETMAINed page (and was therefore cleared to zeroes) may now return an address on an existing page that contains residual (garbage) data.***

VSM GETMAIN Changes in z/OS 1.10

- **Cautionary Example #2**
  - ***Do not make start/end boundary alignment assumptions based on the size of virtual storage being allocated***
    - **Use STARTBDY to specify the boundary the obtained storage must start on**
    - **Use CONTBDY to specify the boundary the obtained storage must be contained within**
  - ***Storage requests that previously returned an address that ended on the last byte of a page may now return an address that does not end on a page boundary.***

Example 2: VSM Low Private Storage Processing pre-z/OS 1.10

Getmain for 8192 bytes from subpool 252 then freemain 2048 bytes at address 74F51800

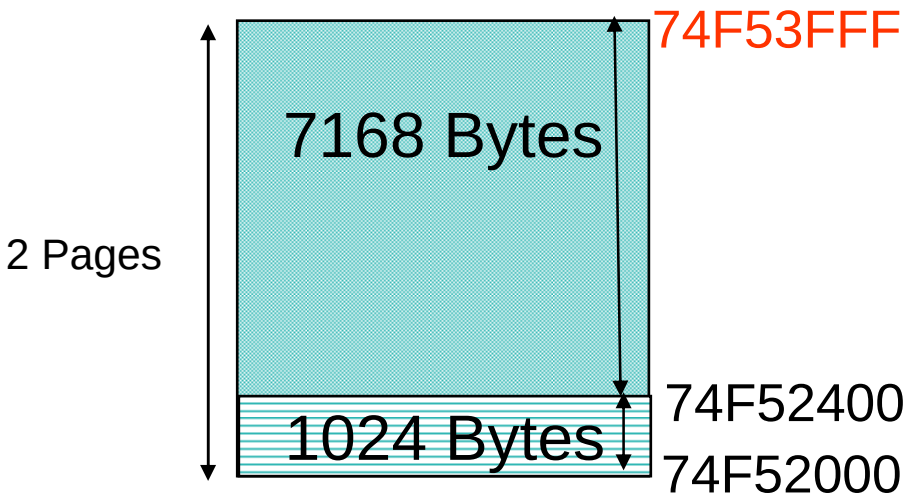


Returned Getmain Address = 74F50000

DQE A: Addr 74F50000 size '2000'X (2 pages)

FQE: Addr 74F51800 size '800'X

Getmain for 7168 bytes from subpool 252 returned virtual ends on last byte of the page



Returned Getmain Address = 74F52400

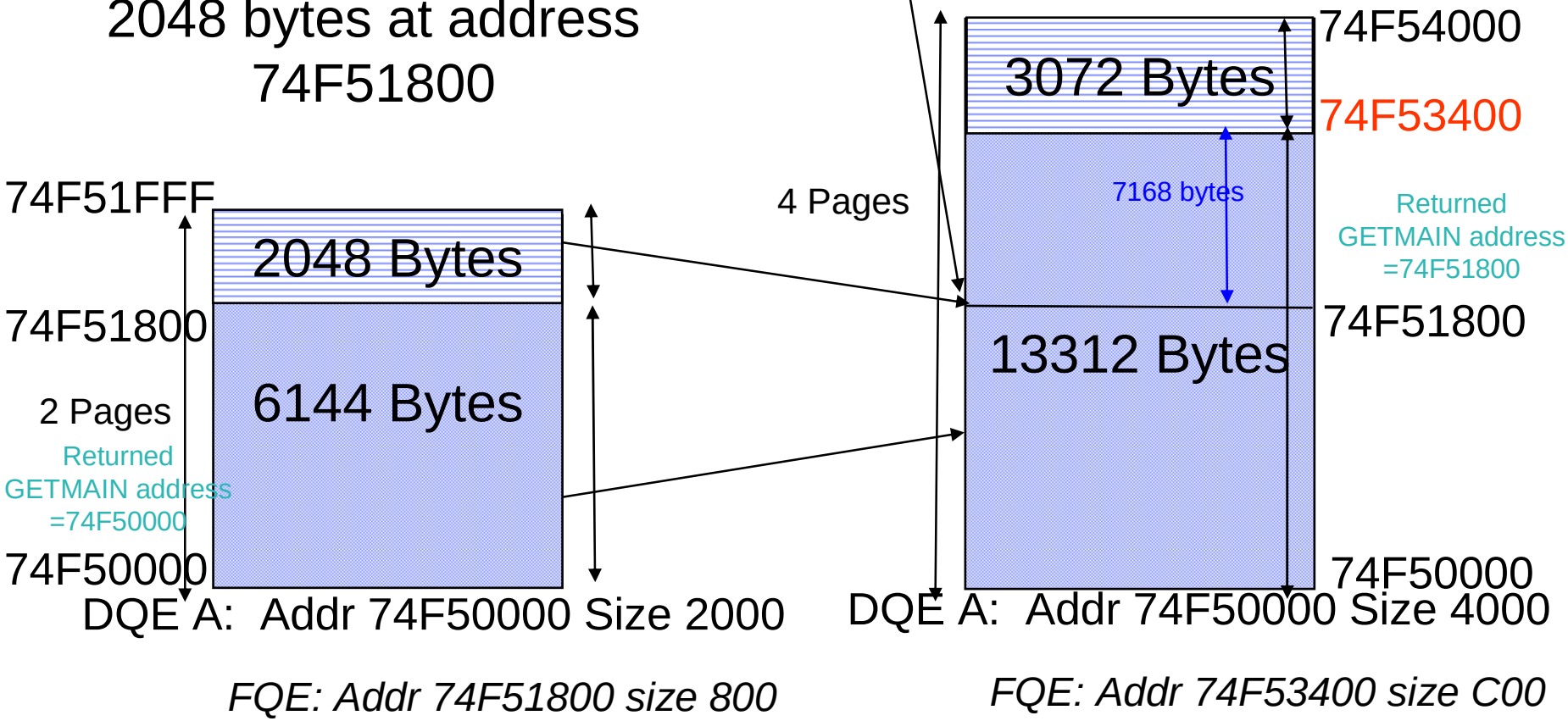
DQE B: Addr 74F52000 Size '2000'X

FQE: Addr 74F52000 size '400'X

Example 2: VSM Low Private Storage Processing z/OS 1.10

Getmain for 8192 bytes from subpool 252 then freemain 2048 bytes at address 74F51800

Getmain for 7168 bytes from subpool 252: returned virtual does *not* end on a page boundary



VSM GETMAIN Changes in z/OS 1.10

SUMMARY of pre-z/OS V1R9 allocation behavior:

- ***Storage is more likely to be obtained from a fresh page (which makes it more likely to be cleared to binary zeroes)***
- ***Storage is allocated from the top (high address) of the page to bottom (lower address)***
- ***Unless a GETMAIN request can be satisfied entirely from an existing FQE, a new DQE must be obtained for each GETMAIN request***

VSM Diagxx Changes in z/OS 1.10

SUMMARY of z/OS V1R10 allocation behavior:

- **Using VSM UsezOSV1R9Rules(YES)**
  - ***Same as pre-z/OS 1.10***
- **Using VSM USEzOSV1R9Rules(NO)**
  - ***Storage requests are more likely to be carved from areas that were previously obtained with the GETMAIN requests (which means they may contain residual data).***
  - ***Storage is allocated from the bottom (lower address) of the page to top (higher address).***
  - ***Storage requests may now be satisfied partly from an FQE allowing new allocation request to be merged into an existing DQE***

VSM CPOOL Changes in z/OS 1.9

- **Problem Statement:**
  - Heavy usage of the CPOOL system service leads to CPU “Hot Cache Lines” for the CPOOL headers, degrading system performance
- **Solution:**
  - Provide a multiple header option for CPOOL to eliminate the “Hot Cache Line” problem

VSM CPOOL Changes in z/OS 1.9

- Using **CPOOL Service Enhancements** customers are expected to…
  - See improved performance in workloads involving heavy usage of z/OS UNIX and/or GRS services
- Value:
  - Customers are expected to get greater throughput on their systems leading to more transactions per second, etc…

VSM CPOOL Changes in z/OS 1.9

- **CPOOL Service Enhancements** will allow a system component or authorized application to:
  - Use **MULTIHDR=YES** on **CPOOL BUILD** to create a multiple header Cell Pool
  - Use **MULTIHDR=YES** on **CPOOL GET/FREE** to get and free elements from the Cell Pool
  - Use **CPOOL LIST, CPOOL DELETE, VSMLOC** and **IPCS RUNCPOOL** functions against the multiple header Cell Pool

VSM CPOOL Changes in z/OS 1.9

CPOOL BUILD

- New option **MULTIHDR=YES** causes creation of a cell pool with 256 byte headers for the maximum number of CPUs allowed on the system
  - Available to authorized callers only (System Key, APF Authorized or Supervisor State)

- New keywords available with **MULTIHDR=YES** only:
  - **MAXCELLS=nnnn** indicates maximum number of cells to be allowed in cell pool before expansion will be stopped on conditional **GET** requests.
  - **CELLSPERCPU=nnnn** indicates the number of cells to be allocated per CPU extent

VSM CPOOL Changes in z/OS 1.9

CPOOL GET

- New option **MULTIHDR=YES** causes the obtain of a cell from the header associated with the running CPU

- **COND=YES** callers will get a zero cell address returned if the maxcells limit has been reached for the cell pool and no cells are currently available for the running CPU

VSM CPOOL Changes in z/OS 1.9

CPOOL FREE

- New option **MULTIHDR=YES** causes the release of a cell to the header associated with the CPU it was obtained from

VSM CPOOL Changes in z/OS 1.9

CPOOL DELETE

- Causes the deletion of the multiple header cell pool including the freeing of all cells and header storage

CPOOL LIST

- Returns the list of extents allocated in the multiple header cell pool

DIAGxx

VSM ALLOWUSERKEYCSA(NO|YES)

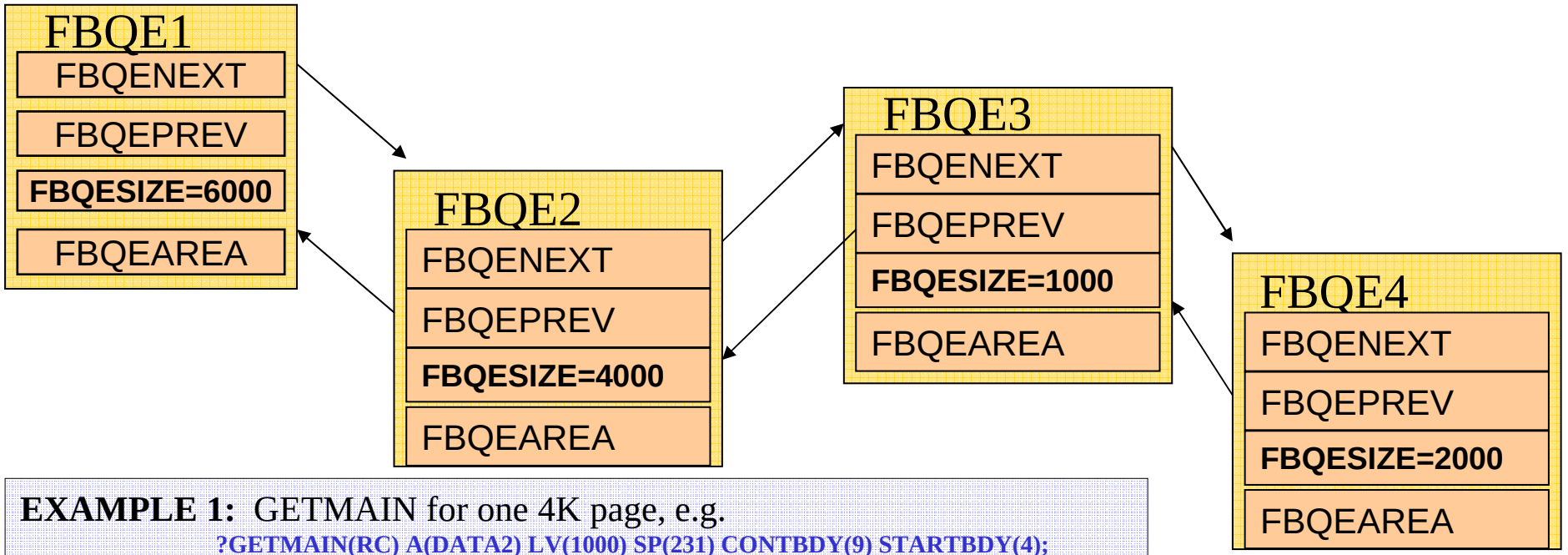
- ***NO prevents user key CSA from being allocated by failing any attempt to obtain user key from a CSA subpool (through GETMAIN or STORAGE OBTAIN) with a B04-5C, B0A-5C, or B78-5C abend. The default is NO. IBM recommends that you should not specify ALLOWUSERKEYCSA(YES). User key CSA creates a security risk because any unauthorized program can modify it.***

VSM BESTFITCSA(NO|YES)

- **Indicates how GETMAIN or STORAGE OBTAIN process requests for (E)CSA storage.**
- **NO indicates to use a "first fit" algorithm in certain situations such as when the STARTBDY and CONTBDY options are used. This is the default, and matches the behavior on all current releases. However in some environments this can lead to (E)CSA fragmentation.**
- **YES indicates to always use a "best fit" algorithm. IBM recommends that you specify YES for this option to minimize (E)CSA fragmentation and to prevent user and system outages due to requests for (E)CSA storage that cannot be satisfied.**

DIAGxx: VSM BESTFITCSA

CSA GETMAINs specifying STARTBDY or CONTBDY keywords are satisfied with a "best fit" algorithm by specifying VSM BESTFITCSA(YES) in the active DIAGxx member of SYS1.PARMLIB



EXAMPLE 1: GETMAIN for one 4K page, e.g.

?GETMAIN(RC) A(DATA2) LV(1000) SP(231) CONTBDY(9) STARTBDY(4);

- VSM BESTFITCSA(NO) (default behavior): satisfied from FBQE1
- VSM BESTFITCSA(YES): satisfied from FBQE3 (an exact fit)

EXAMPLE 2: GETMAIN for three 4K pages, e.g.

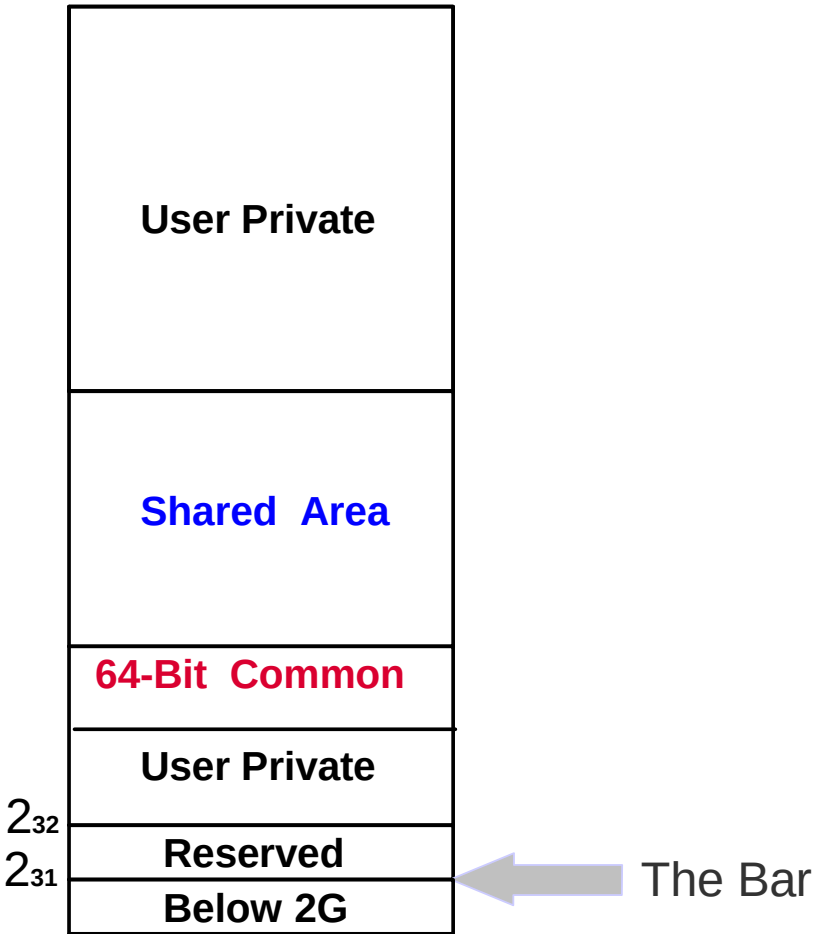
?GETMAIN(RC) A(DATA2) LV(3000) SP(231) CONTBDY(9) STARTBDY(4);

- VSM BESTFITCSA(NO) (default behavior): satisfied from FBQE1
- VSM BESTFITCSA(YES): satisfied from FBQE2

64-Bit VSM/RSM

64-Bit Address Space Memory Map

– *z/OS virtual memory above 2GB is organized as memory objects which programs create.*
  - a memory object is a contiguous range of virtual addresses
  - they are allocated as a number of application pages which are 1MB on a 1MB boundary



64-Bit Memory Object Operations (IARV64)

- Managing Memory Objects
  - Getstor - create a Private Memory Object (only for private memory objects)
  - Changeguard - increase or decrease the amount of usable memory in a memory object (only for private memory objects)
  - Getshared – create a Shared Memory Object (only for shared memory objects)
  - Sharememobj - allows an address space to access Shared Memory Objects (only for shared memory objects)
  - Changeaccess - manages the type of access an address space has to the Shared Virtual Storage (only for shared memory objects)
  - Getcommon- create a Common Memory Object (only for common memory objects)
  - Detach - delete Memory Objects (applies to all memory objects: private, shared, common)

IARST64/IARCP64 in z/OS 1.10

- **Problem:** Virtual Storage Constraint

  - as we strive for Virtual Storage Constraint Relief (VSCR), many components will be asked to move their blocks above the bar

  - every user of 64 bit storage will have to write their own storage manager to hand out smaller pieces of the storage to their application

- **Solution:** Provide a set of services that will allow components to obtain storage as needed without having to write their own storage manager

  - IARST64 – allows the caller to request private or common storage in sizes from 1 byte to 64K.

  - IARCP64 – allows the caller to create a private or common storage cell pool with cells in sizes from 1 byte to almost half a meg

- **Benefit:** These services will help free up storage below the bar

IARST64/IARCP64 in z/OS 1.10

- Using 64 Bit Storage Services, the customer can:
  1. request private or common storage in sizes from 1 byte to 64K
  2. create a private or common storage cell pool with cells in sizes from 1 byte to almost half a meg

- Value:
  1. alleviate virtual storage constraint below the bar
  2. components obtain storage as needed without having to write their own storage manager

IARST64/IARCP64 in z/OS 1.10

- IARST64 (assembler and PLX macro)
  - equivalent of GETMAIN/FREEMAIN or STORAGE OBTAIN/RELEASE for 64 bit storage
  - GET/FREE 1 to 64K bytes of private or common
  - Full doc in macro prolog

- IARCP64 (assembler and PLX macro)
  - equivalent of CPOOL for 64 bit storage
  - BUIILD/GET/FREE/DELETE a pool supporting cells 1 byte to ½ meg of private or common
  - Full doc in macro prolog

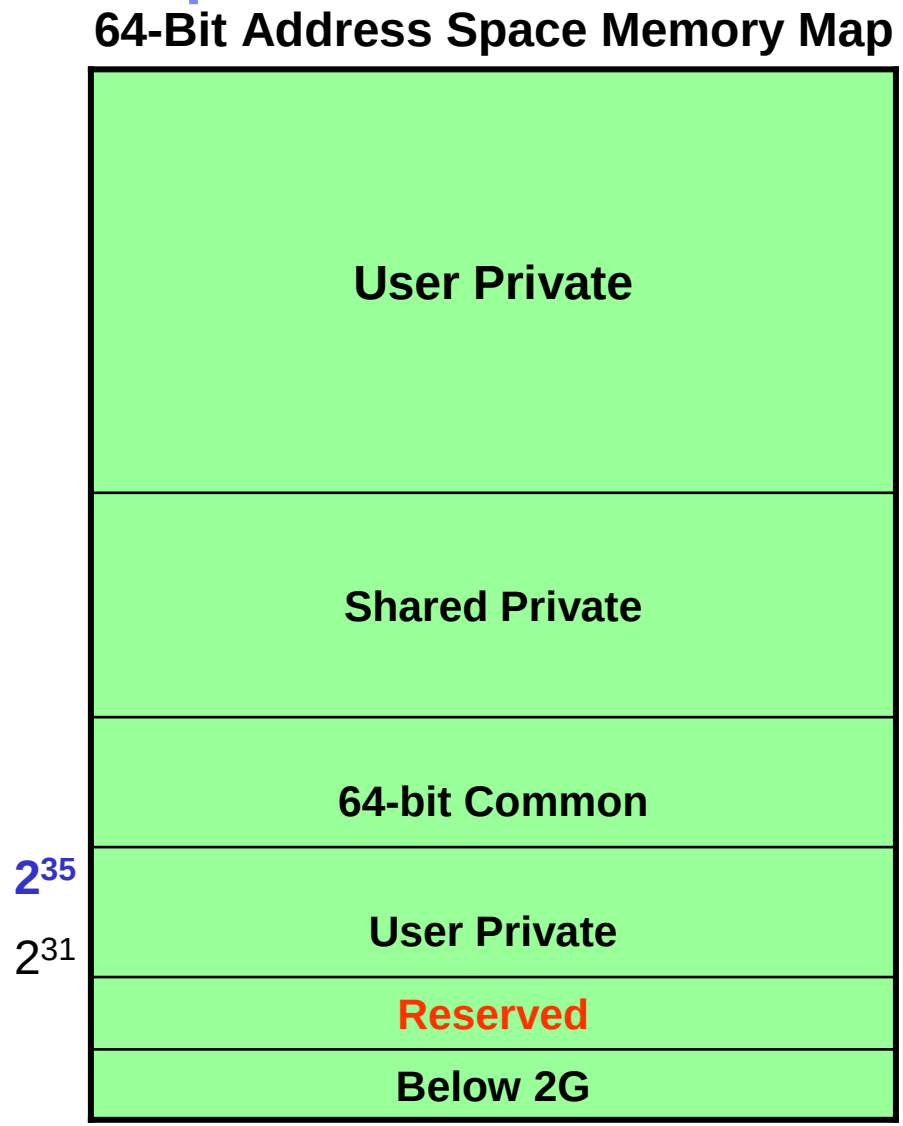
JAVA Pointer Compression - Overview

- Problem Statement / Need Addressed:
  - JAVA can improve their performance by using 64-bit storage
  - Using 32 bit pointer compression, JAVA can access addresses in the 2G-32G range without incurring the performance cost of using 64-bit addresses
  - Need a way to explicitly request storage in the 2G-32G range
- Solution:
  - RSM will allow JAVA to explicitly request 64-bit storage in the 2G-32G area (via an internal interface)
- Benefit:
  - Improved performance with JAVA by allowing the use of 64-bit storage

JAVA Pointer Compression - Component Externals

As a result of this support, the boundary for the low user private area above 2G is now 2^{35} instead of 2^{32}.

Memory objects allocated by JAVA will start at x'80000000'.
Non-JAVA requests will start at x'8_00000000'



JAVA Pointer Compression - Installation

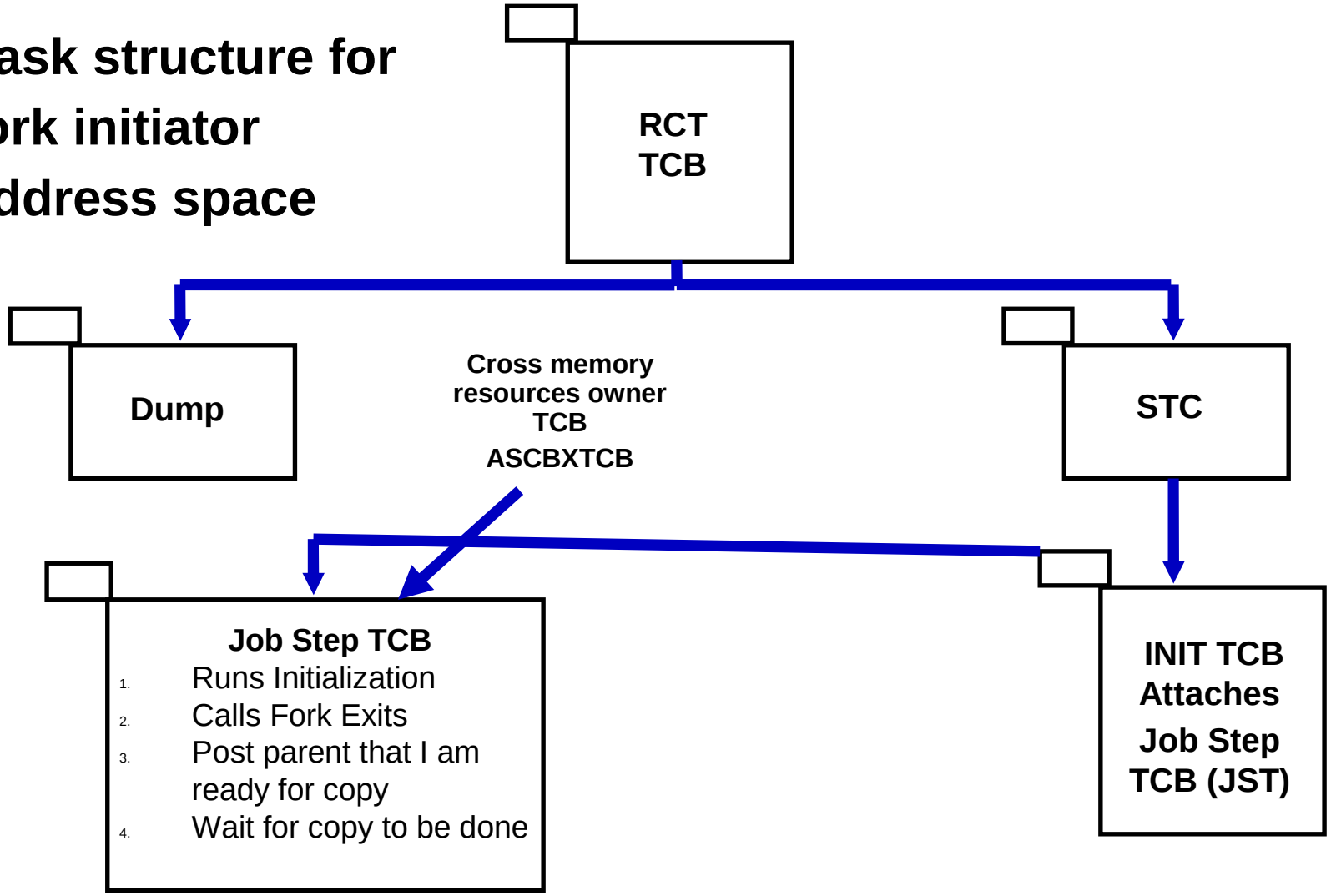
- This enhancement is available in the z/OS v1r11 (HBB7760) release.
- In addition, it is available via APAR OA26294:
  - PTFs:
    - UA44790 for HBB7730 (z/OS v1r8)
    - UA44791 for HBB7740 (z/OS v1r9)
    - UA44792 for HBB7750 (z/OS v1r10)

z/OS 1.12 Changes to IARV64 GETSTOR

- Problem Statement / Need Addressed:
  - ***Today's fork 64-bit copy processing fails when high virtual storage is allocated in the child space at the time when the copy of the parent 64-bit virtual is copied into the child space***
    - **RSM fails the copy with Return Code 8 Reason code '6E000300'x – Child address space contains memory objects that were previously allocated**
    - **Forks fails with Return code 70 Reason code '0B1505C1'x - An invocation of IARV64FC service failed. Action: Retry the operation at a later time**

z/OS 1.12 Changes to IARV64 GETSTOR

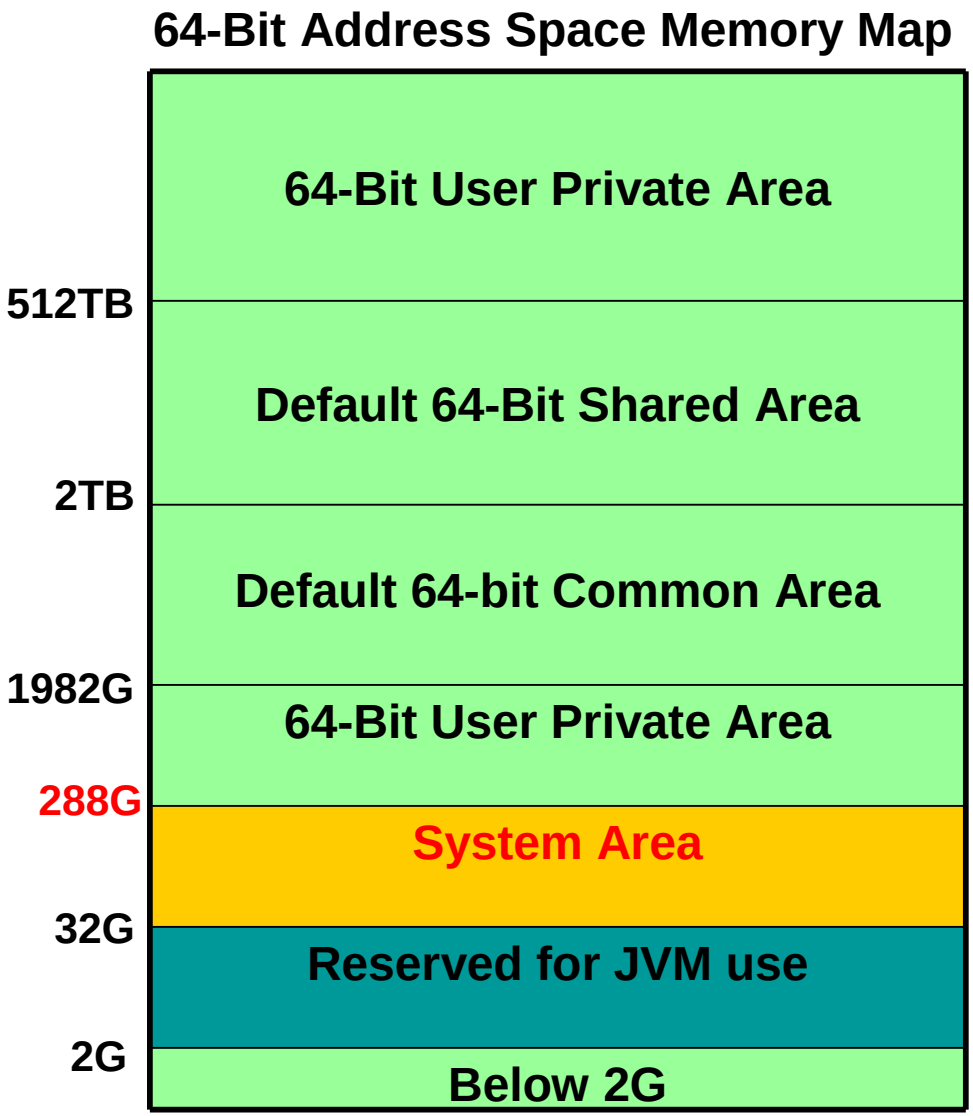
Task structure for fork initiator address space



Overview

As a result of this support, a new system area will be created in the address space 64-bit map

Memory objects allocated in the System Area will start at x'8_00000000'



z/OS 1.12 Changes to IARV64 GETSTOR

- **Solution:**
  - ***A new keyword will be added to the* IARV64 REQUEST=GETSTOR, LOCALSYSAREA=NO|YES *to indicate that the memory object should be allocated from the System Area of the 64-bit address space map*** *(Note: Only authorized users may use this new keyword)*
  - ***When LOCALSYSAREA=YES is specified, MEMLIMIT is ignored***
  - ***During the 64-bit copy phase for fork processing memory objects that are allocated in the system area will not be copied to the child space***
  - ***Fork will fail only if the child has 64-bit memory objects allocated in the User Private Area***
  - ***Checkpoint restart will fail only if RACF builds generic profile tables allocated in the User Private Area***
- **Benefit:**
  - ***Allows fork processing to continue to use 64-bit virtual under any circumstance***

Appendix 1

Monitoring Common Storage Usage via CSA Tracker and IPCS

Common Storage Tracker (or CSA Tracker)

- **Tracks owners of currently GETMAINed SQA/ESQA and CSA/ECSA storage**
  - Address and length of GETMAINed storage
  - ASID, jobname, and PSW address of owner
  - Time and date of GETMAIN
- **Activated via DIAGxx parmlib member**
  - Can be activated/deactivated at any time
  - DIAGxx: VSM TRACK CSA(ON) SQA(ON)
  - SET DIAG=XX

CSA Tracker Terminology

- S – SQA
- C – CSA
- AC - Active storage; the address space that performed the storage request is still active
- OG - Owner Gone storage; the address space that performed the storage request has terminated
- SYSTEM-OWNED storage - storage obtained by the system (e.g. dispatcher, IOS, timer) and not related to a specific address space
- NO DETAIL storage - storage for which CSA tracker has no information; perhaps it was obtained early in IPL before tracker was activated
- CAUB - an internal VSM control block

Formatting CSA Tracker Data

- **VERBX VSMDATA 'OWNCOMM DETAIL'**
  - Formats a detailed report of global storage usage
  - Identifies how much SQA, ESQA, CSA, and ECSA storage is owned by....
    - The system
    - Non-system functions
    - Owner gone functions (owning address space has terminated)
  - For each acquired area of global storage, gives...
    - Storage address and length
    - ASID, jobname, address of owner
    - Time, date of GETMAIN / STORAGE request

VERBX VSMDATA “OWNCOMM DETAIL’

```
                                                          Date       Time
ASID Job Name Id       St T Address  Length   Ret Addr MM/DD/YYYY HH:MM:SS CAUB     GQE
---- -------- -------- -- - -------- -------- -------- ---------- -------- -------- --------
0028 IBMUSER  TSU00016 Ac C 00B44400 00000088 23FCC9D6 09/12/2005 16:45:45 0241FEB0 01DDD6A0
 Data ------> 23FAF2D8 23DB5D80 E3606000 00000088  *..2Q..).T-....h*

0028 IBMUSER  TSU00016 Ac S 00FC5018 00000030 00CA5024 09/12/2005 16:45:45 0241FEB0 01E35AF0
 Data ------> 00000000 00000000 00F97B80 00000028  *.........9#.....*

0028 IBMUSER  TSU00016 Ac S 00FC5048 00000030 00CA5024 09/12/2005 16:45:45 0241FEB0 01E35148
 Data ------> 00000000 00000000 00F97B80 00000028  *.........9#.....*

0028 IBMUSER  TSU00016 Ac S 022E7A28 00000018 039E5364 09/12/2005 16:45:57 0241FEB0 01E43C88
 Data ------> E2E8E2F1 40404040 00000000 00000000  *SYS1    ........*

0028 IBMUSER  TSU00016 Ac S 02313068 00000080 23F3D918 09/12/2005 16:45:58 0241FEB0 01E35C40
 Data ------> D1E2C1C2 00000000 00000080 01000001  *JSAB............*

0028 IBMUSER  TSU00016 Ac S 02546000 00000060 00D3D1AE 09/12/2005 16:45:58 0241FEB0 01E35C70
 Data ------> E2E3D8C5 F500005C 00000000 00000000  *STQE5..*........*

0028 IBMUSER  TSU00016 Ac S 025C1578 00000048 2466F036 09/12/2005 16:45:57 0241FEB0 01E35BC8
 Data ------> D3D4C1C2 00000000 7FF4EF60 7FF4EF60  *LMAB....."4.-"4.-*
```

Formatting CSA Tracker Data

- **VERBX VSMDATA 'OWNCOMM DETAIL'**
  - Can be sorted in various ways with SORTBY keyword:
    VERBX VSMDATA 'OWNCOMM DETAIL SORTBY(xxxxxxxx)'
    - By address (ADDRESS)
    - By time (TIME)
    - By length (LENGTH)
    - By address within ASID (ASIDADDR)
    - By length within ASID (ASIDLEN)
  - Can use IPCS SORT or ISPF data reduction techniques to massage CSA tracker data even further

Appendix - 2

Types of VSM Control Blocks

Each of these VSM control blocks contains an ADDR and a SIZE field
- **ADDR - address of the storage area mapped by the control block**
- **SIZE - size of the storage area mapped by the control block**

- **FBQE -** each FBQE maps a page or block of contiguous **unallocated pages** (fully free)
- **AQAT -** maps a page or block of contiguous pages of **allocated SQA/LSQA**
  - **DFE -** maps **free storage within the allocation** represented by its associated AQAT
- **DQE -** maps a page or block of contiguous pages of **allocated CSA / private** (except LSQA) storage
  - **FQE -** maps **free storage within the allocation** represented by its associated DQE

Appendix 3

- Documentation
  - SA22-7591 - MVS Initialization and Tuning Guide
  - SA22-7592 - MVS Initialization and Tuning Reference
  - SA22-7627 – MVS System Commands
  - SA22-7994 - IBM Health Checker for z/OS V1R9.0 User's Guide
  - SA22-7605 - MVS Assembler Services Guide
  - SA22-7606 – MVS Programming Assembler Services Reference
  - GA22-7589 – MVS Diagnosis Tools and Service Aids
  - GA22-7588 - MVS Diagnosis: Reference
  - GA22-7588 - MVS Interactive Problem Control System (IPCS) Commands
  - SA22-7597 – MVS JCL Reference
  - SA22-7593 – MVS Installation Exits
  - Helpful info items www.ibm.com/support/us then search on the following:
    - OW35742 - Features introduced by VSMDATA SUMMARY report and how to take advantage of them
    - T1000077 - How to diagnose a Global Storage Shortage

Questions ?