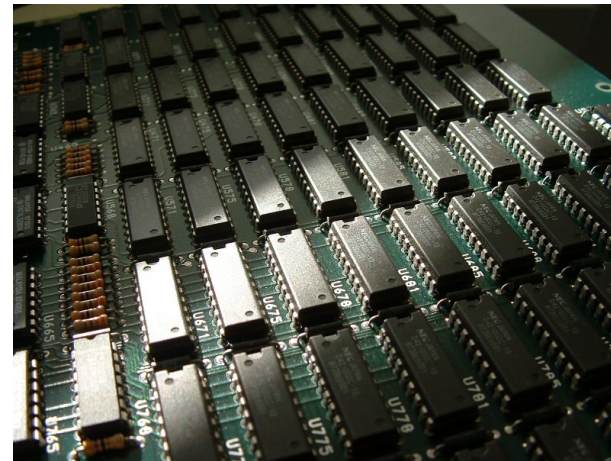


Advanced Storage Management for Model 204

Alex Kodat
Sirius Software Inc.



Computer Storage

- All processing on a computer done by the CPU (or CPUs)
- All data processed by CPU must (at least temporarily) be stored in *main storage*
 - Sometimes called *central storage*
 - Sometimes called *RAM* (for Random Access Memory)
 - Or *core* (by old timers)
 - This storage is what we're talking about in this presentation



A History of Mainframe (360/370/XA/ESA/390/z) Storage (cont..)

- Original 360s used 24-bit addresses so maximum storage usable on a machine was 16 megabytes
- Subsequent models supported virtual addressing
 - More on virtual addressing in a bit
 - Virtual and real address
- XA architecture switched to using 31-bit addresses
 - So maximum addressable memory was 2 gigabytes
 - More than anyone could possibly ever need
 - Right? Right?



A History of Mainframe (360/370/XA/ESA/390/z) Storage (..cont)

- ESA architecture made it possible to go beyond 2 gigabytes using “expanded storage”
 - Expanded storage not addressable the way normal main storage was accessible
 - So generally used as very high-speed swap area
- Z architecture switched to 64-bit addresses
 - So 2^{64} or 16 gigabyte gigabytes or 16 megabyte terabytes 16,384 petabytes or 16 exabytes of storage addressable
 - More than anyone could possibly ever need
 - Right? Right?

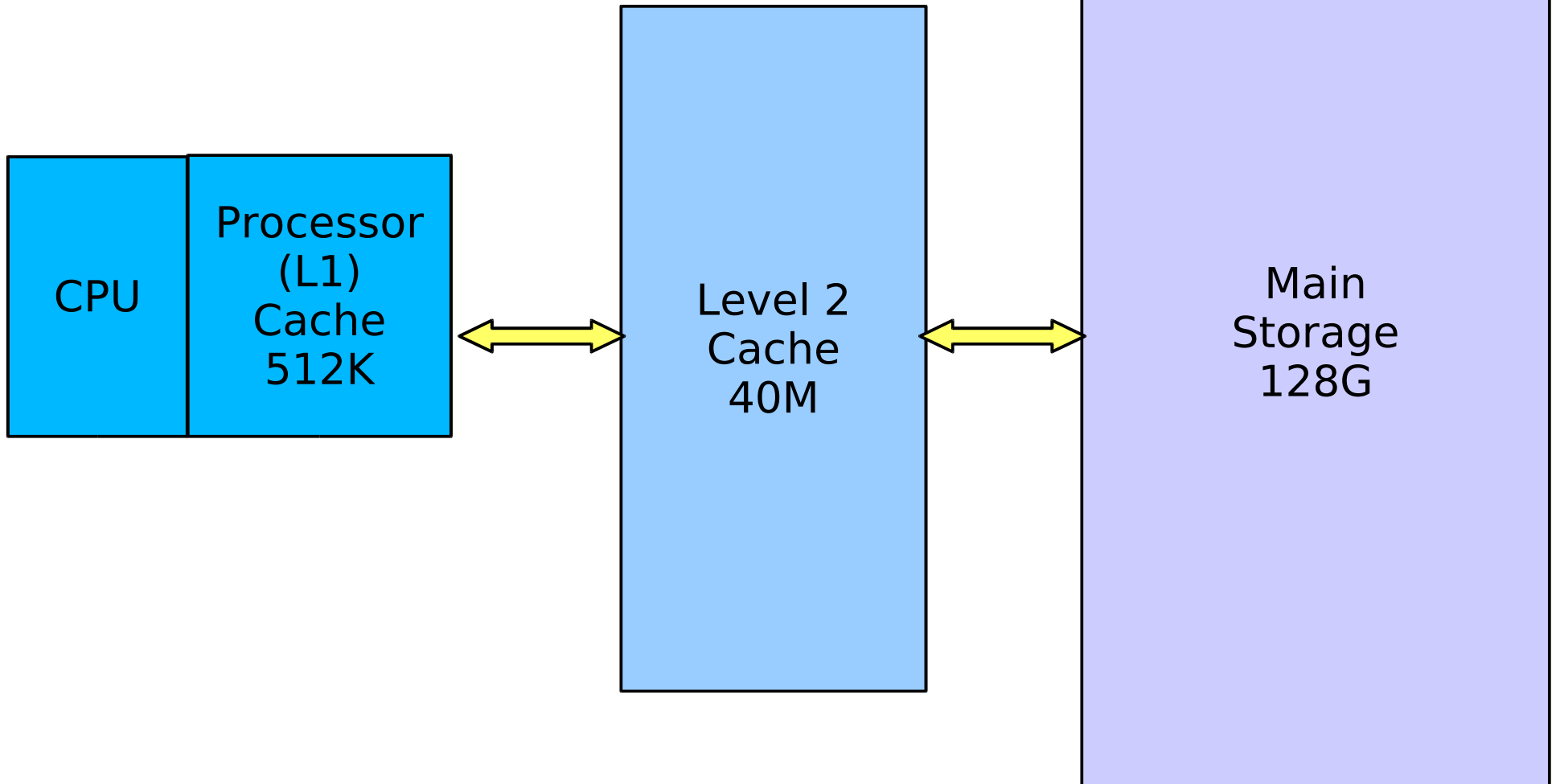


Moore's "Law" – Storage Version

- The amount of main storage available on machines doubles every two years
- Current limit is about 512G (2^{39} bytes)
 - Wow!
- So 64-bit architecture will hit a wall in 50 years
 - Here's betting Moore's Law will hit physical walls before then
- Storage access speeds not increasing as fast as storage quantity
 - So more and more levels of memory cache required to avoid slowing down the processor



Internal Processor Cache (sizes approximate and boxes not to scale)



Working Set Sizes

- The amount of data that is **repeatedly** referenced in over a certain number of machine instructions
 - Or unit time, but more convenient to think in terms of machine instructions
- There's not much to be done for one-of references
 - These will go at the slowest possible rate (physical DASD speeds)
- If the working set size is larger than the amount of storage available to it, an application will run slower than the processor is capable of running it
 - That is, storage becomes a *bottleneck*



Real Storage vs. Virtual Storage

- Real storage is the actual storage hardware and is accessed by storage addresses
 - Formerly 24-bit addresses, then 31-bit, and now 64-bit to allow use of 16M, 2G and now 2^{64} bytes of real storage
- Applications, however, do not refer to real storage addresses
 - If they did, the application would have to be aware if a page was moved, or application pages could not be moved which would severely restrict the operating system's ability to manage main storage
 - So applications refer to virtual addresses which are translated into real addresses via *dynamic address translation* (DAT).



Dynamic Address Translation

- Reference to a virtual address translated by the hardware through operating system managed hierarchical tables to real addresses
 - Highly optimized since almost all mainframe code runs with DAT on
- It is possible that a virtual storage reference does not map to a real storage address
 - 0C4, for example
 - More common (hopefully) is the operating system has chosen to save the page on disk to make room for other pages
 - Such a reference generates a page fault exception
 - Operating system must then decide whether it's an error (0C4) or whether it should bring in the page and retry the instruction



Virtual Storage in a z/Architecture World (cont..)

- z/architecture hardware supported 64-bit real and virtual addresses from day one
 - However, programs could run in 64-bit mode, 31-bit mode or even 24-bit mode
 - Obviously, programs running in 31-bit mode could not access >2G storage
 - Storage above 16M (24-bit) is called *above-the-line* storage
 - Storage above 2G (31-bit) is called *above-the-bar* storage
- Original releases of z/OS (and z/VM) took advantage of real above-the-bar storage
 - By mapping below-the-bar virtual addresses to above-the-bar real addresses
 - So applications unaware that they are using above-the-bar storage



Virtual Storage in a z/Architecture World (..cont)

- With early z/OS, it was possible to have multiple 2G jobs (address spaces) all in real storage at once
- But any individual job could only use up to 2G of storage
 - So there was no 64-bit support for virtual addresses
 - A job could use more than 2G by using dataspace (a separate address space owned by a job)
 - But a pain to access and manage dataspace
- z/OS 1.3 introduced support for virtual 64-bit addresses and this support has been gradually enhanced since then



Storage Usage in Model 204

- The disk buffer pool
- Servers
- Everything else
- When data 204 wants is not in storage 204 tries to find something else to do
 - That is, run another user
 - Doesn't help if no other user to run
- Page faults, however, are not reflected to 204 and are handled by the operating system **synchronously**.
 - So are generally very bad
 - Except maybe single-user runs



Paging and Model 204

- Used to be considered “death” to an Online
 - Page faults were slow (uncached disk I/O) and synchronous from the Online level
 - The whole Online stopped while a page fault was being serviced
 - Each page fault per second reduced Online speed by 3.3%
 - 30 page faults a second meant an Online was getting almost nothing done
- Maybe less bad now
 - Page faults often handled by moving data to/from other storage
 - Perhaps a way of getting around the 31-bit (2G) virtual storage limit



The Disk Buffer Pool

- Contains database pages
 - At most sites, the largest volume of data
 - If not, have you considered Dbase?
 - So generally impossible to keep completely in storage
 - Even pages that are in storage need to be written when updated
 - Copy of page on DASD must be kept more or less up to date
- Contains work pages
 - Work files
 - CCATEMP files
 - The bulk of work pages at most sites



So How Big Should the Buffer Pool Be?

- The bigger the better
 - Definition of a Model 204 Expert – Someone who learns about your system, studies mountains of performance reports, and then tells you to increase your number of buffers
- But how big can it go?
 - Theoretical limit is $2G/(6184+\text{overhead})$ buffers or about 330,000 buffers
 - Since 204 uses 31-bit addresses
 - But this leaves no space for anything else
 - So more realistic limit is say 300,000 buffers
 - How high have folks gone?



The Disk Buffer Pool – Size Matters

- Optimal buffer pool size = Available storage up to 2G
 - Bytes required for servers
 - Bytes required for other stuff
- Nowadays “other stuff” is noise. Call it 100M and don't worry about it
- So buffer pool sizing is largely a trade-off between server space and buffer pool space
- However, if you don't use IOS branch entry (XMEMOPT X'02' bit), per-buffer control blocks must be below the 16M line so limited to many fewer buffers
 - At most 55,000 but probably less



IOS Branch Entry (XMEMOPT X'02' bit)

- Allows per-buffer control blocks to be moved above-the-line (to address > 16M)
 - So allows many more buffers
- Has less CPU overhead than EXCP (the MVS standard)
 - The biggest overhead in IOS branch entry is page-fixing pages about to be read/written
 - Since I/O subsystem can't afford to have pages move underneath it
 - Page fixing surprisingly expensive and requires all kinds of MVS locks
 - So a cheap/easy performance gain by page-fixing the disk buffers and control blocks ★
 - PAGEFIX X'60' bits
 - Disk buffers shouldn't be paging anyway



Using IOS Branch Entry

- Install as an SVC
 - Beware – sometimes release dependencies between 204 and SVC
 - XMEMSVC parameter must be set
 - Generally a PIA
- Link M204XSVC with ONLINE (and BATCH204) load modules
 - Requires module to be APF authorized
 - Does anyone really care?
 - Makes life **so** much easier
 - The recommended approach



Servers

- Hold applications and short-term application data
- A running Model 204 user must be inside a server
 - Server for a running user is in main storage
- An Online can hold many different size servers
- Size of required server depends on application complexity
 - If an application exceeds largest server size in an Online, either the largest server size needs to be increased or the application must be changed (probably split into two or more independent procedures)



Server Swapping

- If number of users equals number of servers ($NUSERS=NSERVS$) then no server swapping is ever performed
- If there are more users than servers, users must be moved into servers before being run
 - Moving user into the server is a server swap-in
 - Before swapping a user in, 204 must swap the current user out – a server swap-out
 - So server swaps always happen in pairs – a swap-out and a swap-in



No Server Swapping: $NUSERS = NSERVS$

- Most efficient solution
 - No extra data movement for server data
 - All users always ready to run
- But might require a lot of storage
 - At large sites, simply not an option
 - For example, if $NUSERS=4000$ and the servers are all 600,000 bytes big, the servers would require about 2.4G which will not fit in the addressable 2G area
 - At smaller sites, either wastes a lot of space on idle servers or depends on operating system paging to only use active servers
 - Better to throw the extra space at the buffer pool



Server Swapping to Disk (cont..)

- Most storage efficient
 - Main storage only used for active servers
 - Little penalty for overallocating the number of users
 - Extra disk space, but who cares?
- But relatively slow
 - A swap-in/swap-out of 600,000 byte servers will take at least 17 milliseconds on a traditional FICON channel
 - Non FICON? Fuggedaboutit
 - This is a **long** time on modern processors and can exacerbate other bottlenecks
 - Newer FICON (FICON Express 4) reduces time to a more acceptable 4 milliseconds



Server Swapping to Disk (..cont..)

- A server must fit on a single cylinder
 - So maximum server size is maximum amount of data that will fit on a (3390) cylinder = 849,960 bytes
 - People are starting to bump against this limit
 - Horrors! Can't they write tight applications?
 - How many “modern” platforms allow a user to run in an 849,960 byte footprint? Most recommendations for Java suggest 32M per user for starters, and work your way up from there
 - Having programmers splitting applications up to run in a 849K footprint doesn't seem like a particularly good use of anyone's time
 - Avoiding inter-procedure transfers is a good thing
- It is possible to overcome this limitation
 - Perhaps with “striping” to allow huge servers



Server Swapping to Disk (..cont)

- As with disk I/O, IOS branch entry recommended
 - Doesn't save much below-the-line (16M) storage, but make server I/Os much more efficient
 - Page fix servers and server control blocks for extra efficiency (PAGEFIX X'020004' bits)
 - ➔ The heck with it? Page fix everything? PAEGFIX=X'FFFFFFFF'
- With high speed FICON, heavily cached (or solid state) DASD, striping, server swapping to disk could again be the preferred approach... with some work...
 - Multi-cylinder swapping
 - Anticipatory server swapping?
 - Avoiding swap of large unused parts of servers
 - Let us/CCA know if this is of interest



Server Swapping to Storage

- Seems silly, doesn't it?
 - Moving data from one part of storage to another. What's the point?
- Originally, it allowed using expanded storage as a swap area
 - Expanded storage couldn't be used for much else
 - The operating system could never use more than 2G of **real** storage, but machines were being shipped with 8G or more
- But there is no expanded storage on newer machines
 - Operating system can now use more than 2G of storage
 - However, swapping is to a dataspace which allows 204 to get around the 2G virtual address limitation
- If you're not struggling with 2G limit, `NUSERS=NSERVS` might make more sense



Enabling Server Swapping to Storage

- Simply eliminate CCASERVR DD card from JCL
 - 204 will detect missing CCASERVR and try to allocate a dataspace
 - Since more than 2G worth of data might be swapped to dataspace, 204 will get multiple dataspaces as needed
- DSPOPT system parameter controls type of swapping
 - X'01' bit – Use page-oriented swapping algorithms
 - X'02' bit – Use hiperspace. Now meaningless

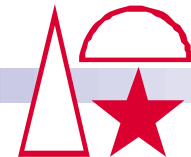
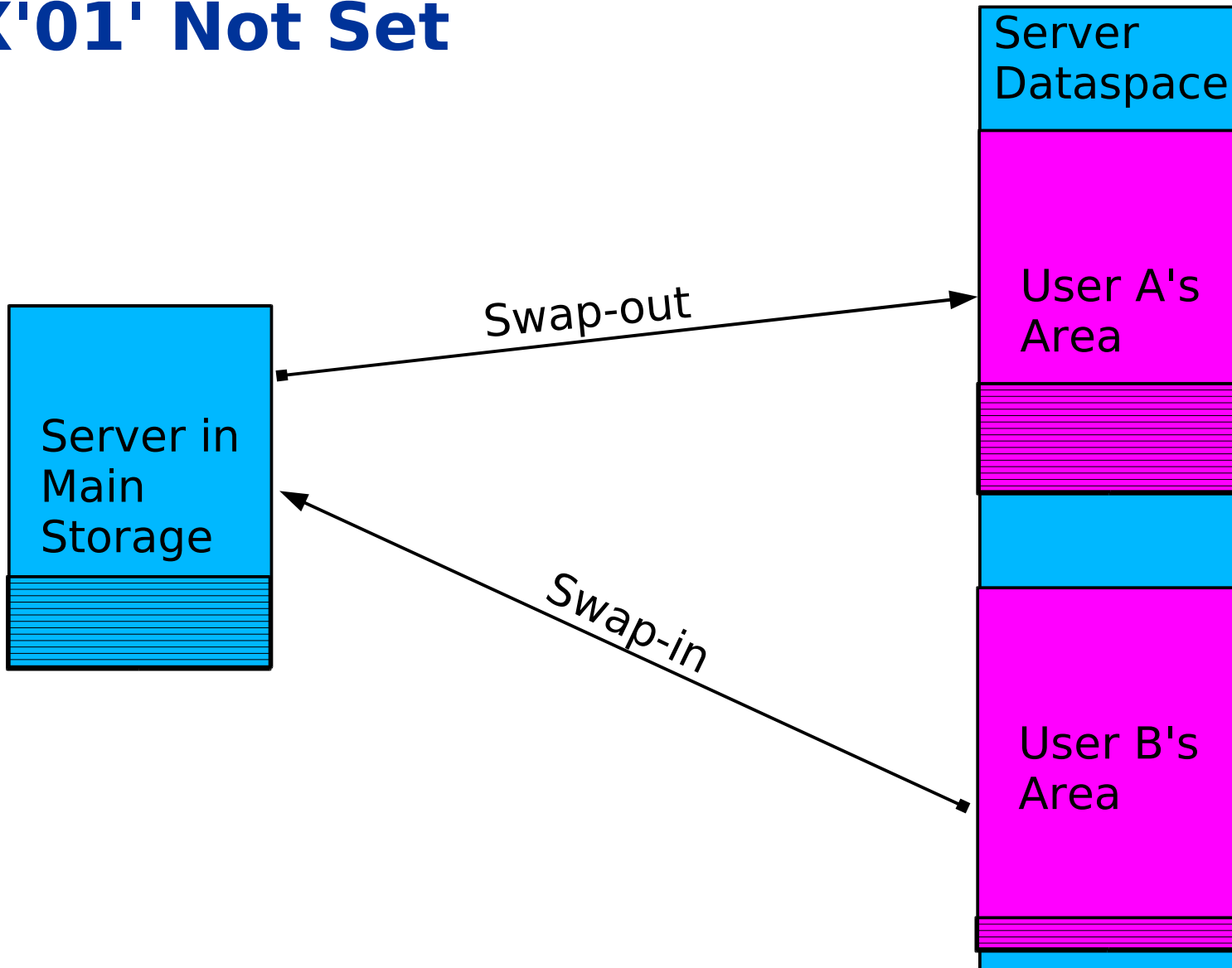


Server Swapping to Storage with DSPOPT X'01' Not Set

- Area in dataspace maps directly to a particular user's server
- When user swaps out, byte oriented move (MVCL instruction) moves server data contiguously to user's server area
- When user swaps in, byte-oriented move (MVCL instruction) moves server data back into server from contiguous area in dataspace



Server Swapping to Storage with DSPOPT X'01' Not Set



Server Swapping with DSPOPT X'01' Off

- Algorithm is very simple
- But data movement is byte-oriented so **slow**
- Data is also duplicated
 - When user swapped in, copy of user's data is also still in the dataspace

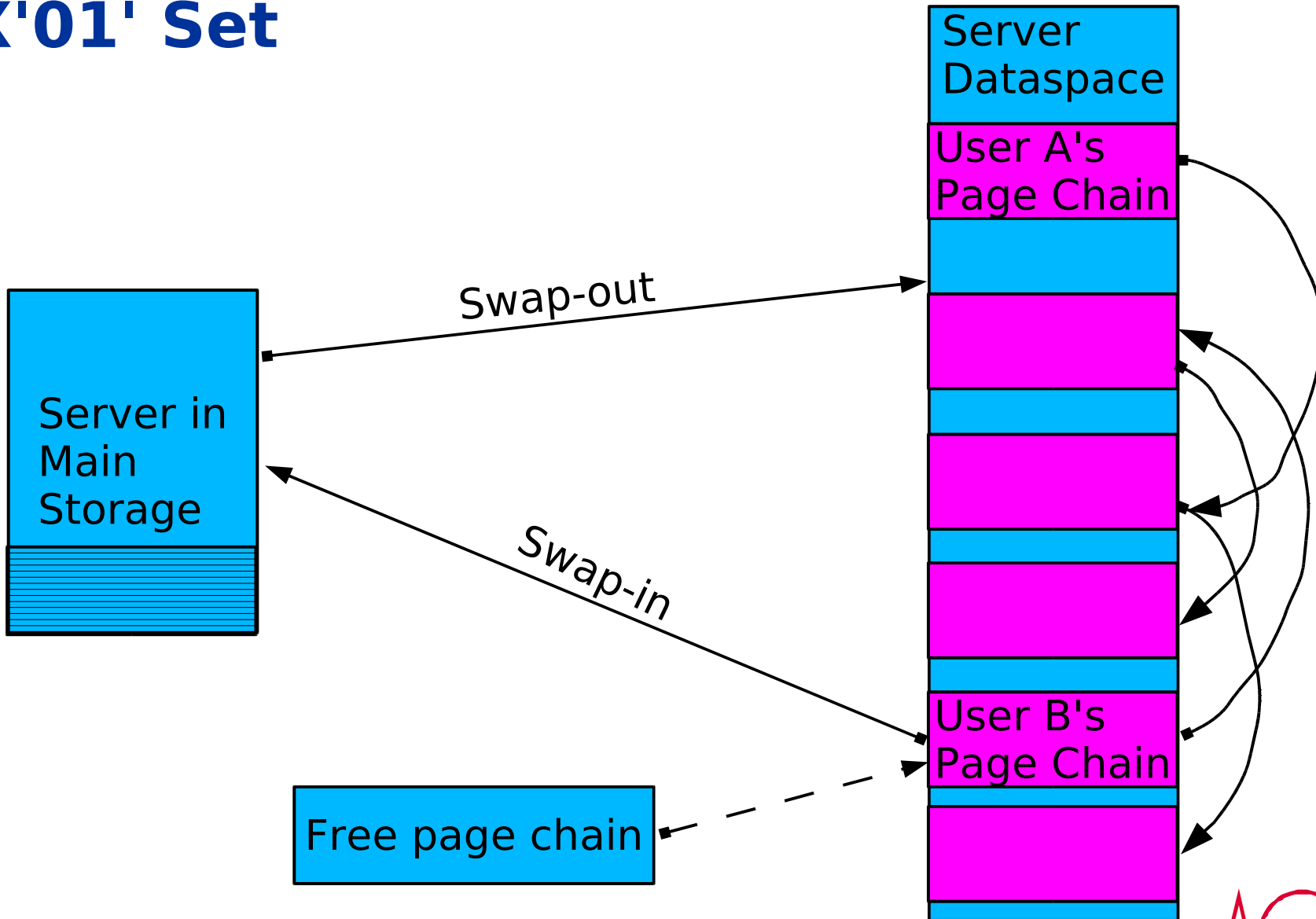


Server Swapping with DSPOPT X'01' Set

- A pool of dataspace pages is managed on an MRU basis
- When a user swaps-out the required dataspace pages are pulled off the MRU queue
 - The users pages are then moved to the dataspace using a page-oriented move (MVPG)
- When user swaps-in the user's pages are moved into the server using page-oriented move (MVPG)
 - As pages are moved into server, the dataspace pages are freed up by placing them back on the MRU queue
- Caveat emptor – If SERVSZ set very tight, it might need to be increased for DSPOPT X'01'



Server Swapping to Storage with DSPOPT X'01' Set



Server Swapping With DSPOPT X'01' On

- Algorithm more complex
- Data movement is **fast**
 - In fact, asynchronous on most modern hardware
- No duplication of data
 - Dataspace pages for swapped in users reused for swapped out users
- Algorithm more clever about not swapping “holes” in VTBL and STBL so less data swapped
- Algorithm doesn't swap out VTBL, STBL, NTBL and QTBL for non-logged in users so less data swapped
- Recommendation: Use DSPOPT X'01' if server swapping to a dataspace



Server Swapping and MP/204

- If swapping large servers to/from storage, server swaps can be CPU intensive
 - Even using MVPG instructions
- So best not to tie up maintask doing server swaps
 - Set SCHEDOPT X'08' bit to allow subtasks to do server swaps



QTBL and NTBL

- Two server tables that are (almost always) static at evaluation time
- Can be quite large percentage of a user's server
 - Especially QTBL
 - Typical range is 30% to 80%
- Many users often run the same handful of procedures over and over again
 - > 90% of the runs are the same 20 procs is typical in most environments
- So it might save storage if users shared QTBL and NTBL



Shared/Resident QTBL (and NTBL)

- Only used for pre-compiled APSY procedures (duh)
- Each shared QTBL/NTBL is stored in a separate area of main storage
 - Up to RESSIZE bytes
 - If 204 wants to make a request resident, but doing so would exceed RESSIZE, the request is not made resident
- Once a request is made resident, it remains resident until the subsystem is stopped
 - Some exceptions that we won't get into here



Benefits of Resident QTBL/NTBL

- When a user using resident QTBL gets swapped out, the QTBL and NTBL parts don't need to get swapped
 - So saves CPU and makes swaps faster
 - If server swapping to storage and multiple users using the request at one time, storage saved because only one copy of QTBL/NTBL in storage
- When doing an APSY load of a resident request, QTBL and NTBL don't need to be loaded
 - So saves CPU and makes APSY loads faster
- When a request made resident, CCATEMP pages for QTBL and NTBL for saved request freed
 - So saves CCATEMP space
 - And storage, possibly (more on this later)



How 204 Decides to Make a Request Resident

- During APSY load it compares the count of times a user has server swapped while running the procedure with RESTHRSH (system parameter)
 - If greater than or equal it tries to make the request resident
- During APSY load it compares the count of times a user has done an APSY load for the procedure with RESLTHR
 - If greater than or equal it tries to make the request resident



RESTHRSH or RESLTHR?

- Doing more server swaps or APSY loads?
 - See system *APSYLD* and *SERVRD* stats
- Is CPU or main storage more of an issue?
 - RESTHRSH does more for main storage if server swapping to storage and DSPOPT X'01' set
- If you have plenty of spare main storage and not bumping against the 2G limit, set RESSIZE big and set both RESLTHR and RESTHRSH



RESLTHR/RESTHRSH Approach Advantages

- Simple
 - Two simple parameters to set



RESLTHR/RESTHRSH Approach

Disadvantages

- Tricky to set
 - If set too low, batch stuff that runs at start of day might be made resident and prevent other things from becoming resident
 - If set too high, might take a long time for requests to become resident



Resident QTBL/NTBL Miscellany

- MONITOR SUBSYS (PROCCT) and SirMon 5.2 provide useful “how are you doing” info
- Unfortunately, no proc by proc breakdown of what's resident and what's not
- SIRAPSY command provides “manual” control over which procedures should be resident
 - Based on SirAud info?
 - But not generally available
 - And no one's actually bothered with it
- Bad interaction with non-pagefixed servers
- Any interest in more details/controls?



APSY Loads

- Data for procedure copied from CCATEMP to server
- Much faster than a compile
- But...
 - Compilation pages take up lots of space in buffer pool
 - Lots of DKPRs to read pages in
 - Relatively slow byte-oriented move (MVCL instruction)
 - Because 204 page size does not match hardware page size
- So...



APSYPAGE

- System parameter set at system start-up
- Indicates number of (hardware) pages to reserve for saved compilations
 - In a dataspace, so doesn't count against 2G limit
 - Not in buffer pool
- When doing an APSY load, 204 looks to see if request saved in APSYPAGE pool
 - If not, load done from CCATEMP
 - And then tries to save request to APSYPAGE pool
 - If yes, load done from APSYPAGE pool



Advantages of Loading from APSYPAGE Pool

- Pages linked together in storage so no DKPRs
- Pages not in buffer pool so more pages available for other uses
 - With no 2G penalty since pages in a separate dataspace
- Pages in APSYPAGE pool use hardware page size so can use very fast MVPG instruction



Issues With APSYPAGE

- Most result from APSYPAGE being written for small amounts of expanded storage
- If cached hiperspace used, operating system can steal APSYPAGE pages
 - Very friendly but...
 - Don't set DSPOPT X'40' bit ★
- Because requests saved in APSYPAGE pool stealable (LRU), they are also saved in CCATEMP
 - No matter how big APSYPAGE is
 - Currently, no way around this
- Caveat emptor – If SERVSZ set very tight, it might need to be increased for APSYPAGE>0



CCATEMP in Storage

- In dataspace, like servers
 - So, again, no 2G penalty
- Indicated by setting TEMPPAGE system parameter
- Because CCATEMP page size different from hardware page size, no page-oriented (MVPG) move option available
 - But still **much** faster than I/O, even to solid-state DASD
- Pages must be moved into buffer pool before being used



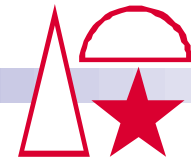
To Use CCATEMP in Storage or Not?

- If only enough space for either APSYPAGE or TEMPPAGE, APSYPAGE generally preferable ★
 - Bulk of most people's CCATEMP pages are saved APSY requests
 - APSYPAGE removes saved request pages from buffer pool
 - Sometimes reducing I/O by more than total CCATEMP I/O!
 - For APSY loads, APSYPAGE saves CPU, TEMPPAGE does not
- If you have storage to burn, sure, put CCATEMP in storage
 - But this is quite wasteful
 - Many requests in both CCATEMP and APSYPAGE dataspace
 - And for a while in the buffer pool, too



Resident Requests and APSYPAGE

- If request goes resident, APSYPAGE and TEMPPAGE space associated with request is freed
- So no storage cost in making request resident
- However, there **is** a 2G limit cost
 - Resident requests use below-the-bar storage
 - APSYPAGE/TEMPPAGE pages use dataspace storage
- So if you're not struggling with the 2G limit and you have spare real storage on your system, set RESSIZE high and RESTHRSH and RESLTHR low
 - And use APSYPAGE for VTBL and STBL
 - No harm done if RESSIZE overdone



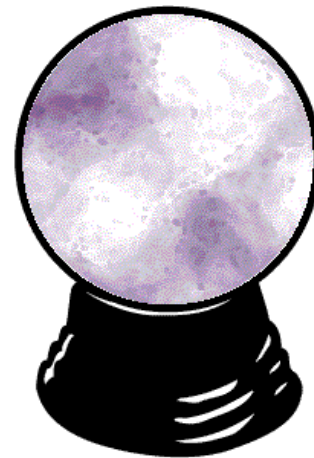
Stuff I've Ignored

- Below-the-line (16M) issues
 - Split (above-the-line) 204 load module solves many of these issues
- Object space allocation
 - CCATEMP vs. server size
- Mid-size data structures
 - User KOMMs
 - Record locking table
 - User/file mode table



The Future

- Not long before everyone can throw 2G at 204
 - And high-end customers will be able to throw 8G+
- 204 will continue adding capabilities to take advantage of ever increasing amounts of storage
 - 64-bit buffer pool
 - Servers above-the-bar?
 - Monster servers?
 - In storage files?



Any questions or comments?



Sirius Software, Inc.