

z/VM 6.3: Changes in Memory Management

Session 15745



John Franciscovich
francisj@us.ibm.com



Agenda

- Objectives and strategies of the z/VM Large Memory enhancement

- Key features of the z/VM Large Memory enhancement
 - Algorithmic concepts: new, changed, or obsolete
 - Basic flows and data structures
 - Tuning options

- Planning for z/VM Large Memory
 - Paging DASD calculations
 - Reminders about best practices with respect to paging I/O

- Workloads

- Summary

Objectives and Strategies

- Objectives:
 - Support **1 TB** of central memory in a partition
 - Support large guests in such a context
 - Retain ability to overcommit memory

- Strategies:
 - Repair or replace memory management algorithms that
 - do not scale well
 - are grossly unfair

- Specifically:
 - **Page reorder** is a real problem area. Get rid of it.
 - **Demand scan** has scaling problems and frame ordering problems. Repair them.
 - Introduce a new **global aging list** concept to add accuracy to frame reclaim decisions.
 - Improve **fairness** of frame steal when memory is constrained.
 - Improve effectiveness of keeping virtual machine memory specified by **SET RESERVED** resident in memory.
 - Extend **SET RESERVED** to **DCSS**es such as MONDCSS.

New Algorithms and Behaviors

New Approach: Highlights

- Objective: keep the **available lists** populated just right

- New **demand scan** design tries to improve occupancy fairness in the face of storage constraint

- In-use frames are tracked by a new hierarchical data structure:
 - Valid, often-touched frames are at the top
 - Demand scan pushes frames downward as they seem to increase in reclaim appeal
 - Best reclaim candidates are at the bottom

- DASD use for paging is changed to be more friendly to reclaim and to storage subsystems
 - Pages valid on DASD are not rewritten
 - Pages are rewritten to the same slots
 - Channel program can do fully discontinuous reads or writes
 - z/VM can pre-write pages to DASD

New Approach: Management of The Available Lists

Old way

Each **list** had a low threshold and a high threshold

After every free storage request call, demand scan was kicked off if a **list** fell below its low threshold

The <2G lists were repopulated by demand scan

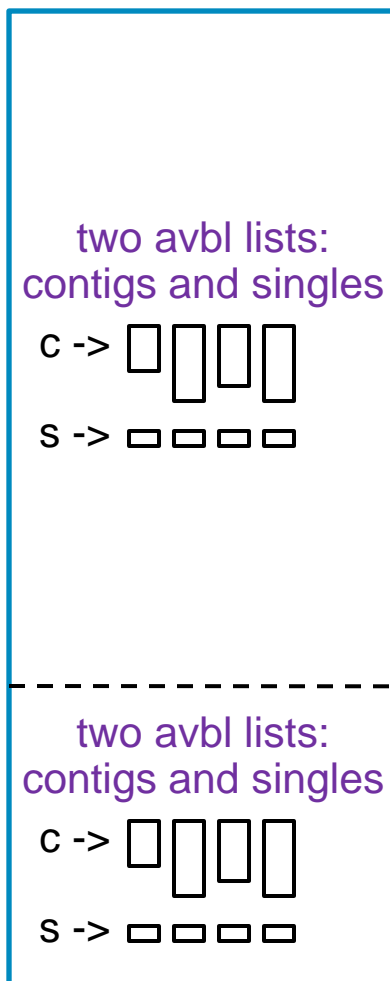
<2G Use Policy:

Pre-6.2: used <2G first

In 6.2: used <2G proportionally

In 6.3: uses <2G last

2 GB



New way

Each **kind of free storage request call** has a low and a high threshold:

- TYPE=ANY contigs
- TYPE=ANY singles
- TYPE=BELOW contigs
- TYPE=BELOW singles

Contiguous lists are protected from being completely raided by singles requests

After every request, the low threshold for **every type of request** is evaluated

If a TYPE=ANY low threshold is breached, demand scan is kicked off

If the <2G lists are empty, a frame table scan is kicked off

The Old Demand Scan Visit Policy

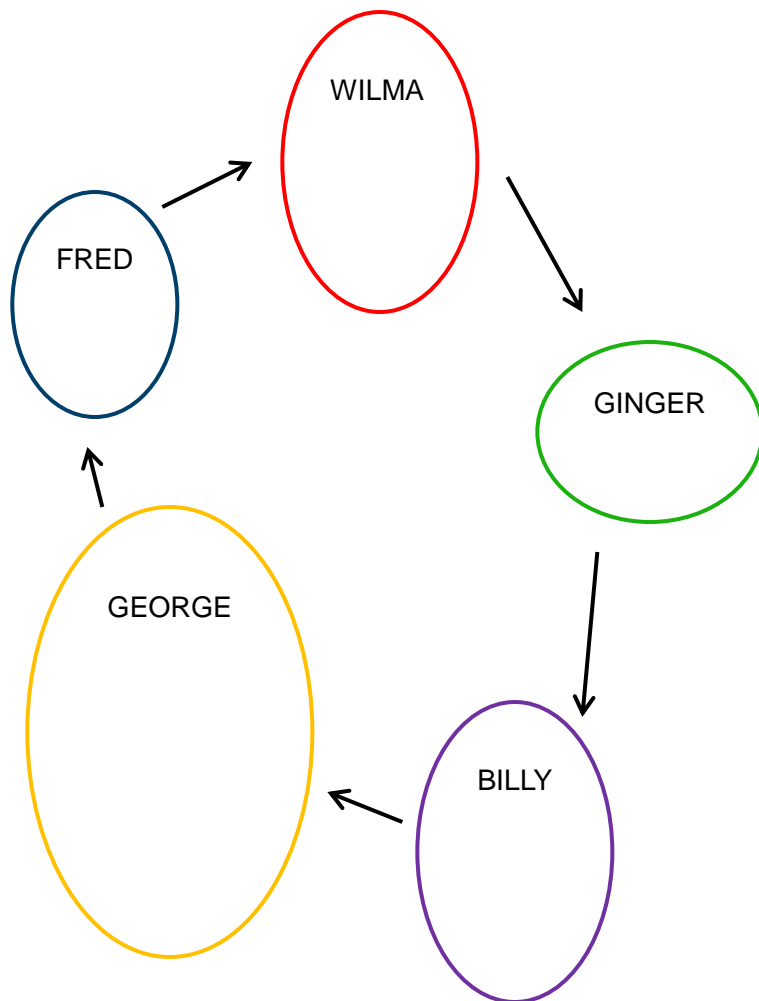
- Three-pass model:
 - **Pass 1:** tried to be friendly to dispatched users
 - Unreferenced shared-address-space pages
 - Long-term-dormant guests
 - Eligible-list guests
 - Dispatch-list guests' unreferenced pages down to Working Set Size
 - **Pass 2:** a little more aggressive... like pass 1 except:
 - Avoided shared address spaces
 - Would take from dispatch-list guests down to their SET RESERVED
 - **Pass 3:** emergency scan
 - Anything we can find

The Old Demand Scan Problems

- We found a number of problems over time, to various degrees, such as:
 - Pass 1 tended to be too soft.
 - Scheduler lists tended not to portray “active” in a way usable by storage management.
 - Stole a lot from the first few guests visited.
 - **SET RESERVED** was not being observed.

- It used the System z page reference bit R to track page changes
 - Required lots of RRBE instructions to keep track of recent reference habits
 - RRBE can be an expensive instruction
 - (Large resident frame list) + (long RRBE instruction) = problems in Reorder

New Approach: The New Demand Scan Visit Policy



- **Used to:**
 - Visit according to scheduler lists
 - Take heavily at each visited guest
 - Start over at list tops every pass
 - Take from private VDISKS nearly last
 - A “take” was truly a **reclaim** of a frame
- **Now:**
 - Cyclically visits the logged-on guests
 - Keeps a visit cursor so it can resume
 - Takes a little and then moves to next
 - Takes from private VDISKS much earlier
 - A “take” is now just a push of in-use frames down toward **eventual reclaim**
- **Effects**
 - Better equalizing in the face of storage constraint
 - Better equalizing on the notion of “hot” vs. “cold” pages

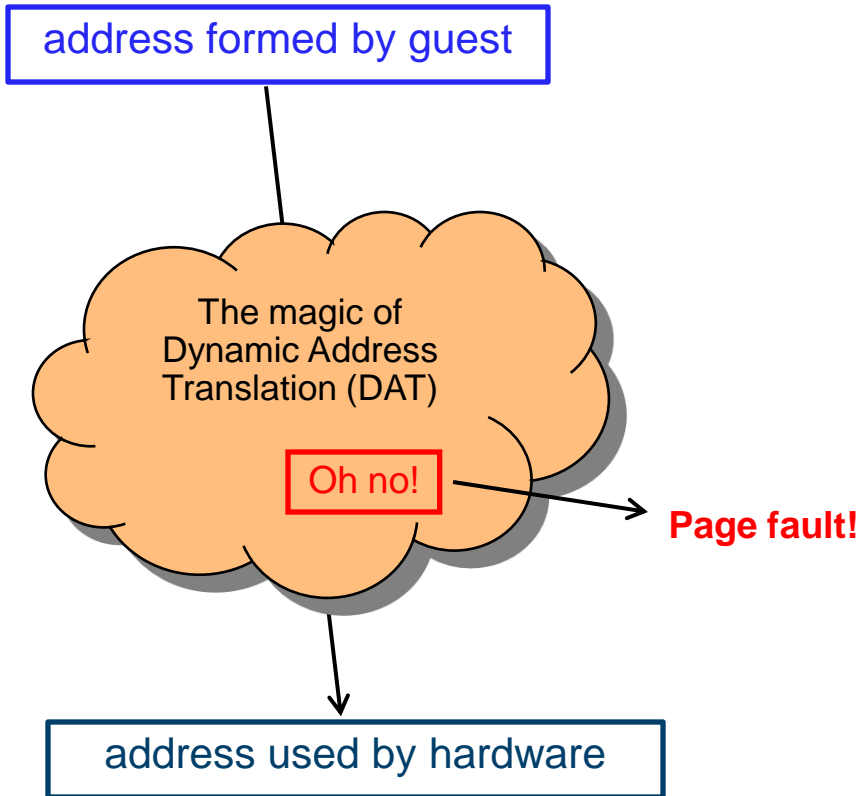
New Approach: Other New Things About Demand Scan

- Gives up control periodically
 - Lets other things happen
 - Avoids long-running “blackouts”

- Tries harder to be “fair” in the face of constraint.

- Aspects of “fairness”:
 - Treat identical guests identically
 - Use a guest’s size and estimation of its page touch rate to decide how much to take
 - Take from large guests who touch their pages less often before taking from small guests who touch their pages a lot
 - Don’t take from a guest’s working set if another guest is not stripped to its working set
 - During startup (when page touch rate data is available) take an amount of pages proportional to each guest’s size

New Approach: Trial Invalidation



- Page table entry (PTE) contains an “invalid” bit
- What if we:
 - Keep the PTE intact but set the “invalid” bit
 - Leave the frame contents intact
 - Wait for the guest to touch the page
- A touch will cause a page fault, but...
- On a fault, there is nothing really to do except:
 - Clear the “invalid” bit
 - Move the frame to the front of the frame list to show that it was recently referenced
- We call this **trial invalidation**.

New Approach: Two-Section Frame-Owned Lists

Frame list types:

A user frame list

The private VDISKS
(new!)

The shared pages

Active

frame
frame
frame
...
frame
frame
frame

Active: frames that are in use

- Roughly in order by when they became valid.

No longer is a frame list ever
searched, sorted, or reordered.

IBR

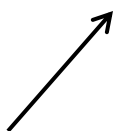
frame
frame
...
frame

IBR: invalid but resident

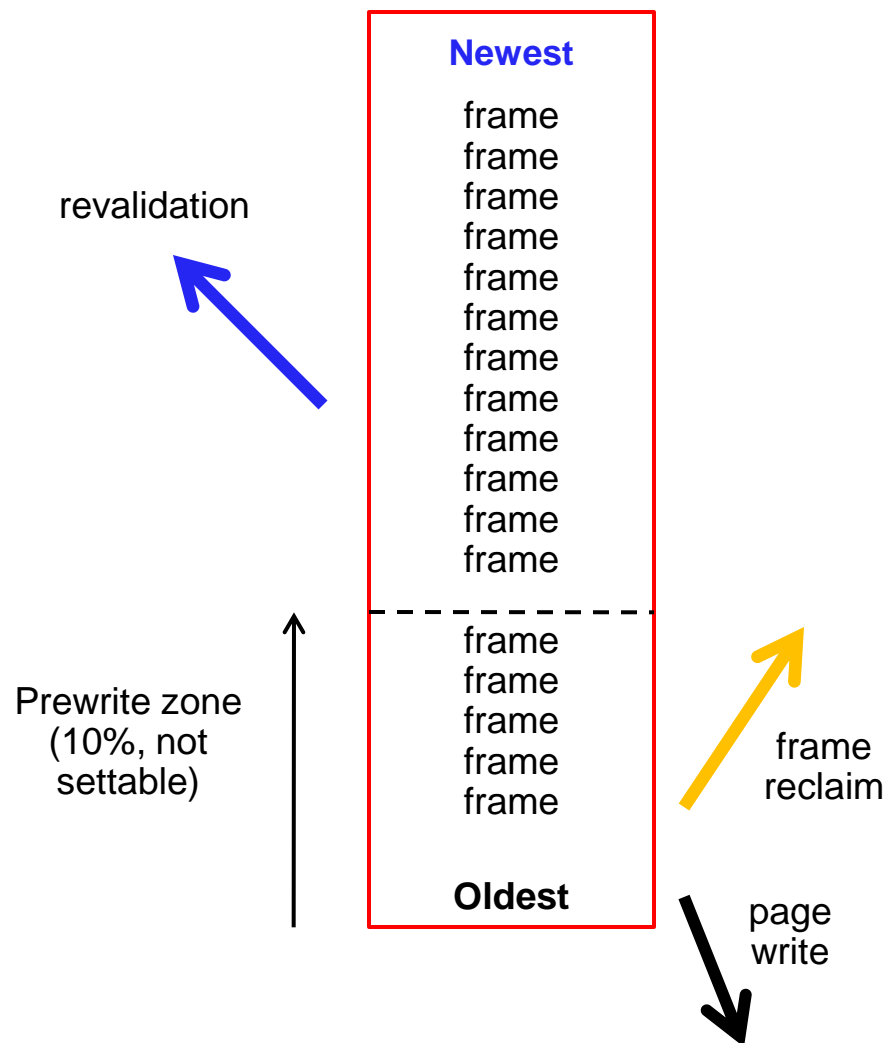
- Marked invalid in page table entry
- If ever referenced, fault resolution moves to top of active
- This gives us a way to detect lack of reference

- "We try to keep [it] rather small."
- Influenced by revalidation rate.

Demand scan
decides
where the
line is.

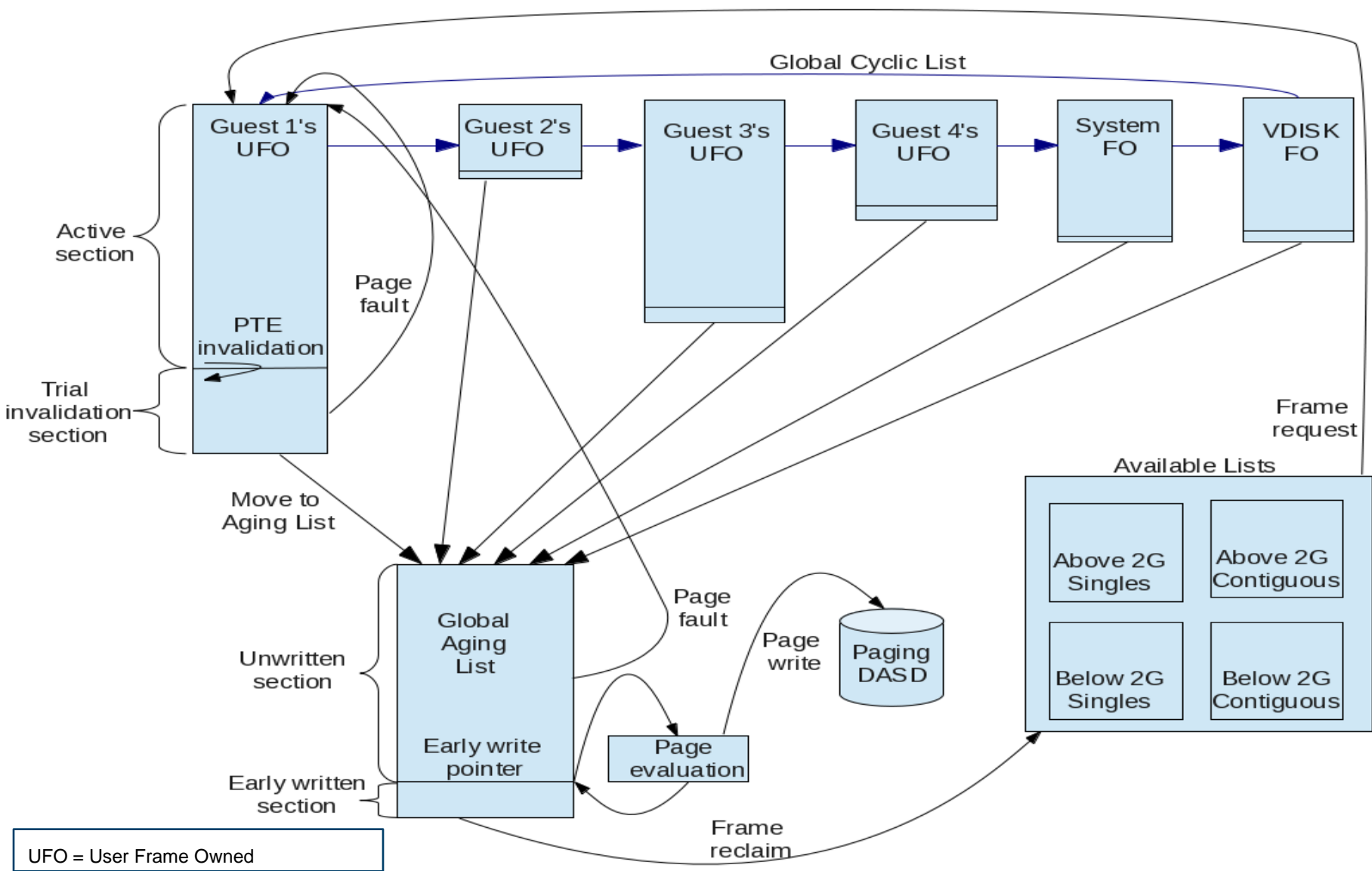


New Approach: Global Aging List

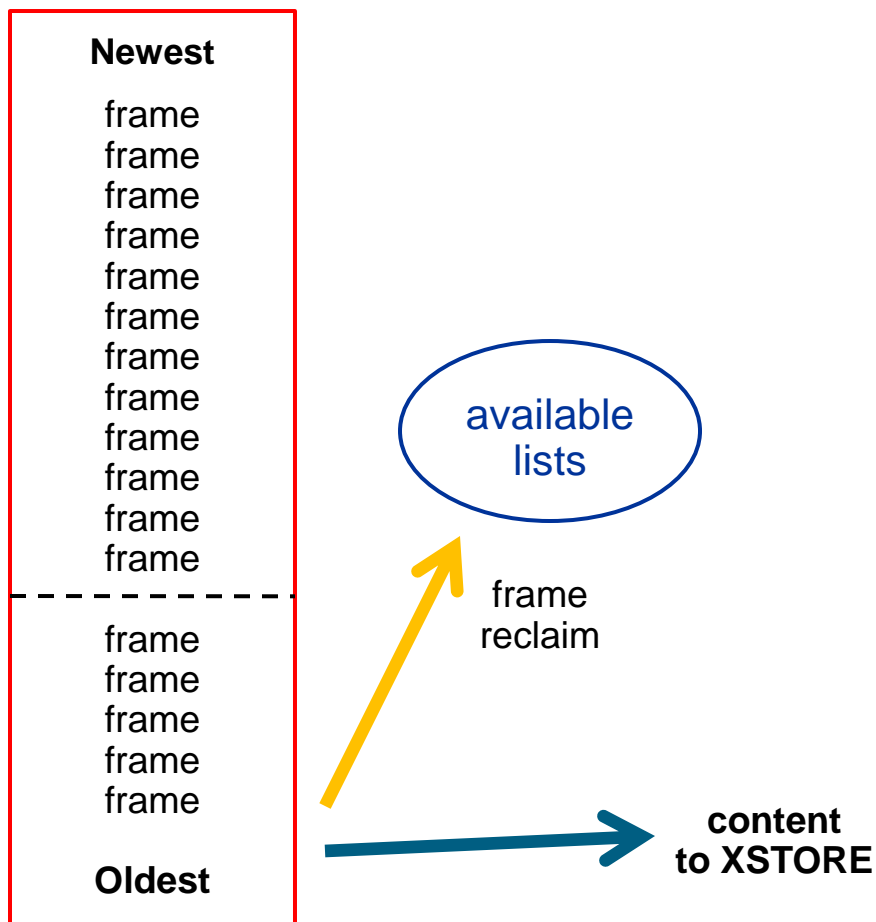


- Size of global aging list can be specified...
... but is best left to the system to manage
- All of the pages here are **IBR**
- Demand scan fills it from the top
- Revalidated pages return to their owned lists
- We pre-write changed pages up from the bottom of the list.
- The global aging list accomplishes the age-filtering process that XSTORE used to accomplish.
- We no longer suggest XSTORE for paging, but we will use it if it's there.

Memory Management Algorithm Visualization



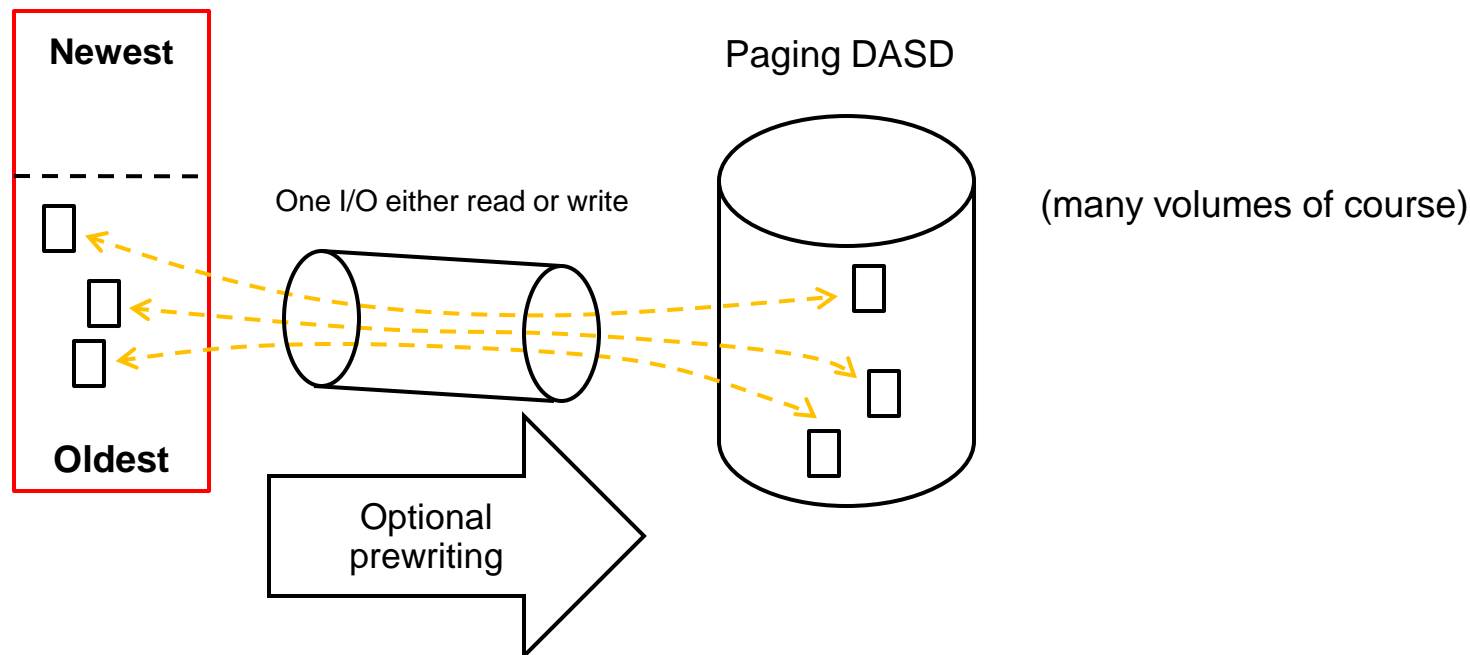
New Approach: What About XSTORE?



- We will use XSTORE if it is there.
- XSTORE is now the *second* line of defense.
- *When frame is reclaimed*, if XSTORE is present, we put a copy of the page there.
 - Even if the frame has already been prewritten
- On fault, if content is still in XSTORE, it comes back from there.
- **If you decide to keep XSTORE, do NOT put MDC in XSTORE unless heavy CMS workload.**

New Approach: How We Now Use Paging DASD

Global aging list



Highlights of new DASD techniques:

- A page almost always goes back to its same DASD slot.
 - Exceptions: clogged or DRAINEd volume
- A page not changed since last read from DASD is almost never rewritten.
 - Exceptions: DRAINEd volume
- The paging channel program can handle discontinuity on both ends, whether read or write.

New Approach: Large Real Implies Large Virtual, So...

- z/VM holds its DAT management structures in CP-owned pageable address spaces
- These *Page Table Resource Manager* address spaces are named PTRM0000, PTRM0001, ...
- You will see them in the z/VM Performance Toolkit FCX134 DSPACESH report
- The number and size of these address spaces control how much logged-on guest real (aka virtual memory) the system can support

- In z/VM 6.2:
 - There were **16** of them: ..., PTRM000F
 - They were created as needed
 - With 16 of these, can address **8 TB** of virtual

- In z/VM 6.3:
 - There are now **128** of them: ..., PTRM007F
 - All are created at system initialization
 - With 128 of these, can address **64 TB** of virtual

New Behavior: CP SET RESERVED command

- We now do much better at honoring the setting
 - Revisit your uses to see whether you were trying to compensate

- Pages can be now be reserved for **NSS** and **DCSS** as well as virtual machines
 - Set *after* **CP SAVESYS** or **SAVESEG** of NSS or DCSS
 - Segment *does not need to be loaded* in order to **SET RESERVE** for it
 - A new instance of an NSS or DCSS *does not* inherit a pending-purge instance's RESERVED setting
 - Recommended for **MONDCSS**

- You can set a system-wide maximum (**SYSMAX**) on the number of reserved pages

- RESERVED settings *do not* survive IPL
 - Consider CP command in the CP directory (not for NSS or DCSS though)

Removed Behavior: Reorder

- z/VM no longer does Reorder processing
 - No longer a trade-off with larger virtual machines

- Commands remain for compatibility but have no impact
 - **CP SET REORDER** command gives RC=6005, “not supported”.
 - **CP QUERY REORDER** command says it’s OFF.

- Be aware of reorder settings when using Live Guest Relocation between z/VM 6.2 and z/VM 6.3

Changed Behavior: Eligible List

- One of the factors to the creation of an **eligible list** is the concept of “loading users”
 - Governed by **SET SRM LDUBUF**
 - A virtual machine is characterized as a “loading user” if its count of page faults in a dispatch slice exceeds a threshold
 - **SET SRM LDUBUF** attempts to keep the system from over-committing paging devices to the point of thrashing

- Changes in z/VM 6.3 paging algorithms can affect the number of virtual machines that are marked as “loading” users and therefore cause **eligible lists** to be formed where they had not formed prior to z/VM 6.3
 - Definition of page fault slightly different
 - Rate at which system can page fault has increased

- Recommend monitoring for eligible lists and adjusting the following as appropriate
 - **SET QUICKDSP**
 - **SET SRM LDUBUF**

- IBM is investigating improvements to avoid the unnecessary eligible list formation.

New or Changed Commands

Commands: Knobs You Can Twist

Concept	Knob	Comments
Size of the global aging list	Command: CP SET AGELIST ...	Sets the size of the global aging list, in terms of: - A fixed amount (e.g., GB) - A percent of DPA (preferred)
Whether early writes are allowed	Config file: STORAGE AGELIST ... Lookup: CP QUERY AGELIST	The default is 2% of DPA. Seems OK. Sets whether early writes are allowed. (If storage-rich, say NO.)
Amount of storage reserved for a user or for a DCSS	Command: CP SET RESERVED ... Config file: STORAGE RESERVED ... Lookup: CP QUERY RESERVED ...	You can set RESERVED for: - A user - An NSS or DCSS You can also set a SYSMAX on total RESERVED storage. Config file can only set SYSMAX.

Commands: Other Interesting “Queries”

Query or Lookup	Comments
CP INDICATE LOAD	The STEAL-nnn% field no longer appears in the output.
CP INDICATE NSS	Includes a new “instantiated” count. Number of pages that exist. Sum of locus counts might add to more than “instantiated”.
CP INDICATE USER	Includes a new “instantiated” count. Sum of locus counts might add to more than “instantiated”.
CP INDICATE SPACES	Includes a new “instantiated” count.

Required Planning

Planning for Large Memory

- **Normal best practices for migrating from an earlier release still apply.**

- **Change your paging XSTORE into central**
 - XSTORE provided an aging function. It let us catch ***Least Recently Used*** "mistakes".
 - The new ***IBR*** concept and ***global aging list*** provide the same function but do so more efficiently in central storage.

- **Plan enough DASD paging space**
 - The system now pre-writes pages to DASD.
 - See space calculation on a later slide

- **Plan a robust paging DASD configuration**
 - Use plenty of paging volumes
 - Make the volumes all the same size
 - Put only paging space on the volumes you use for paging
 - Spread the paging volumes through your logical control units
 - Avoid logical control units that you know are hot on application I/O
 - Use plenty of chpids
 - Do not use ESCON chpids
 - Do not mix ECKD paging and SCSI paging
 - Leave reserved slots in the CP-owned list

Planning for Large Memory

- Look at your **CP SET RESERVED** settings to make sure they're right.
 - Revisit scenarios where you looked at this capability and it wasn't effective

- Add **CP SET RESERVED** settings for DCSSes or NSSes if you'd like
 - MONDCSS is a good one to consider

- If you increase central, make sure you also increase dump space
 - See "*Allocating Space for CP Hard Abend Dumps*" in CP Planning and Administration

Planning DASD Paging Space

- Calculate sum of:
 - Logged-on virtual machines' primary address spaces, plus...
 - Any data spaces they create, plus...
 - Any VDISKS they use, plus...
 - Total number of shared NSS or DCSS pages, ... and then ...
 - Multiply this sum by 1.01 to allow for PGMBKs and friends

- Add to that sum:
 - Total number of CP directory pages (reported by DIRECTXA), plus...
 - Min (10% of central, 4 GB) to allow for system-owned virtual pages

- Then multiply by some safety factor (1.25?) to allow for growth or uncertainty

- Remember that your system will take a PGT004 if you run out of paging space

- Consider using something that alerts on page space, such as Operations Manager for z/VM

Planning to Learn About Your System's Performance

- While you are still on the earlier release, collect measurement data:
 - Know what your key success metrics are and what their success thresholds are
 - Transaction rates – *only you* know where these are on your workloads
 - MONWRITE files – some tips:
 - When: Daily peaks? Month-end processing? Quarter-end processing?
 - Collection tips: <http://www.vm.ibm.com/devpages/bkw/monwrite.html>

- Then go ahead and try z/VM 6.3

- When you start running on z/VM 6.3, collect the very same measurement data

- Compare z/VM 6.3 back to z/VM 6.2 to see what the effect is on your workload

Planning to Keep Your System Maintained

- Current RSU is 6303 (April 8, 2014)

- Keep listening:
 - www.vm.ibm.com
 - The IBMVM mailing list

- See also the PSP bucket for z/VM 6.3

Comments on Workloads

z/VM Large Memory: Amenable Workloads

- **Best benefit:** workloads highly affected by reorder or old demand scan
 - Large guests affected by reorder delays
 - Long demand scans looking for <2G frames

- **Less benefit:** workloads that were doing fine before
 - Storage-rich workloads
 - Running fine paging to only XSTORE
 - No problems with long demand scans
 - Small guests not affected by reorder

- Let's look at some examples

The “Sweet Spot” Workload

Our synthetic workload called *Sweet Spot* imitates behaviors we have seen in customer-supplied MONWRITE data.

	zVM 6.2	z/VM 6.3	Delta	Pct. Delta
Cstore	256	384	128	
Xstore	128	0	-128	
External Throughput (ETR)	0.0746	0.0968	0.0222	29.8%
Internal Throughput (ITR)	77.77	105.60	27.83	35.8%
System Util/Proc	31.4	4.7	-26.7	-85.0%
T/V Ratio	1.51	1.08	-0.43	-28.5

By getting rid of both reorders and spin lock contention, we achieved huge drops in %CPU and T/V.

The “Sweet Spot” Workload

- Closer look at how the fairness and workloads may result in different results.
- Sweet Spot workload has four groups of virtual machines. Some benefit more than others.

	z/VM 6.2	z/VM 6.3	Delta	Pct. Delta
System External Throughput	0.0746	0.0968	0.0222	29.8%
User Group 1 ETR	0.0065	0.0128	0.0063	96.9%
User Group 2 ETR	0.0138	0.0236	0.0098	71.0%
User Group 3 ETR	0.0268	0.0264	-0.0004	-1.5%
User Group 4 ETR	0.0275	0.0341	0.0066	24.0%

Workload: The Apache Paging Workload

Our Linux-based workload called *Apache Paging* is built to page heavily to DASD almost no matter how much central or XSTORE we give it.

	z/VM 6.2	z/VM 6.3
Cstore (GB)	256	384
Xstore (GB)	128	0
External Throughput (ETR)	1.000	1.024
Internal Throughput (ITR)	1.000	1.017
Xstore paging / second	82489	0
DASD paging / second	33574	31376

This is an example of a workload where the limit comes from something large memory will not fix.

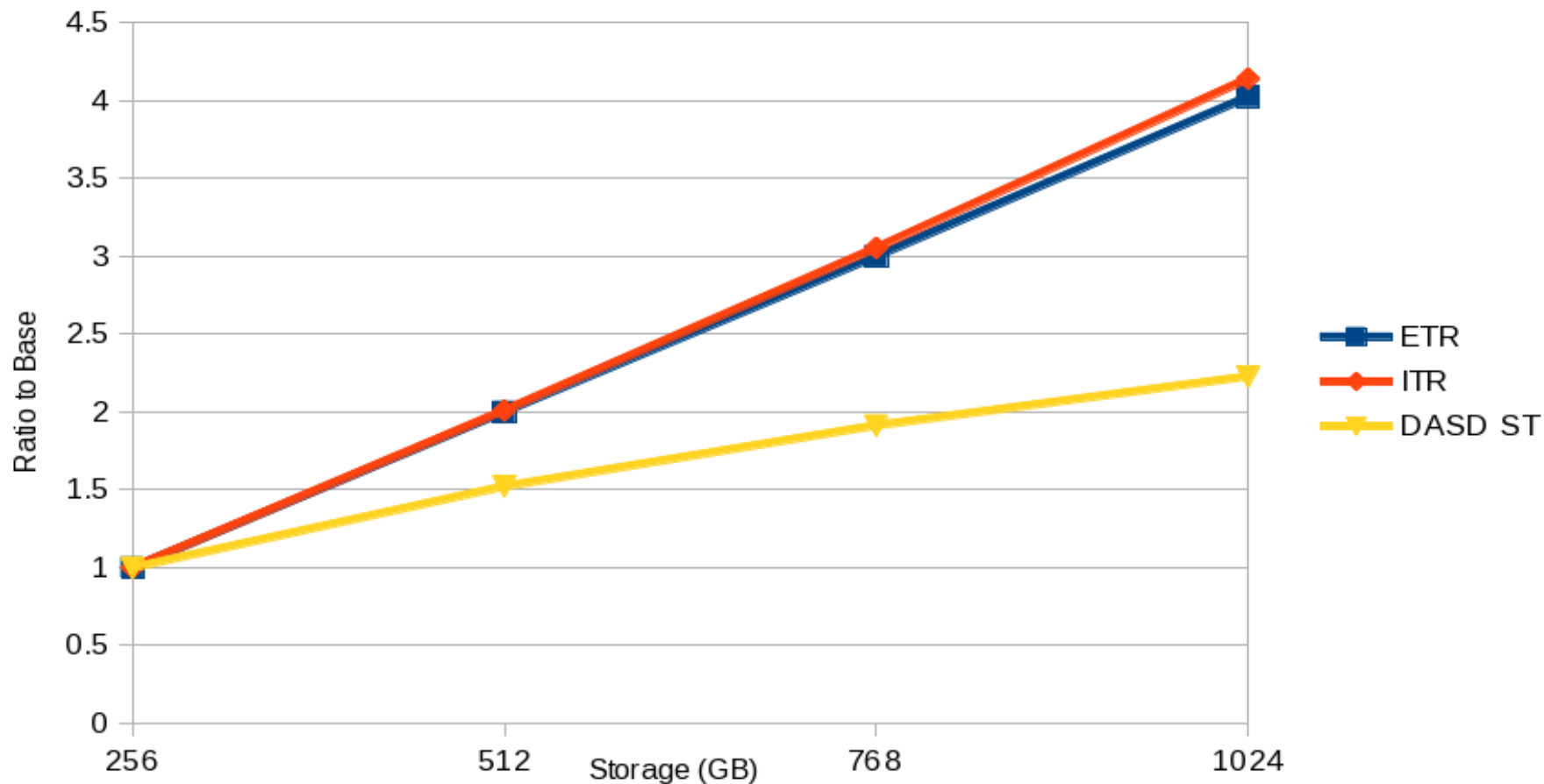
Large Memory Scaling Measurements

1. **VIRSTOR** – Test case system started with CMS boot strap with controls over memory reference patterns and processor usage.
 - Create workload similar to resource usage from customer Monwrite data

2. **Linux Apache Static Web serving**
 - Measure and test levels of servers at peak usage for 256 GB in an overcommitted environment

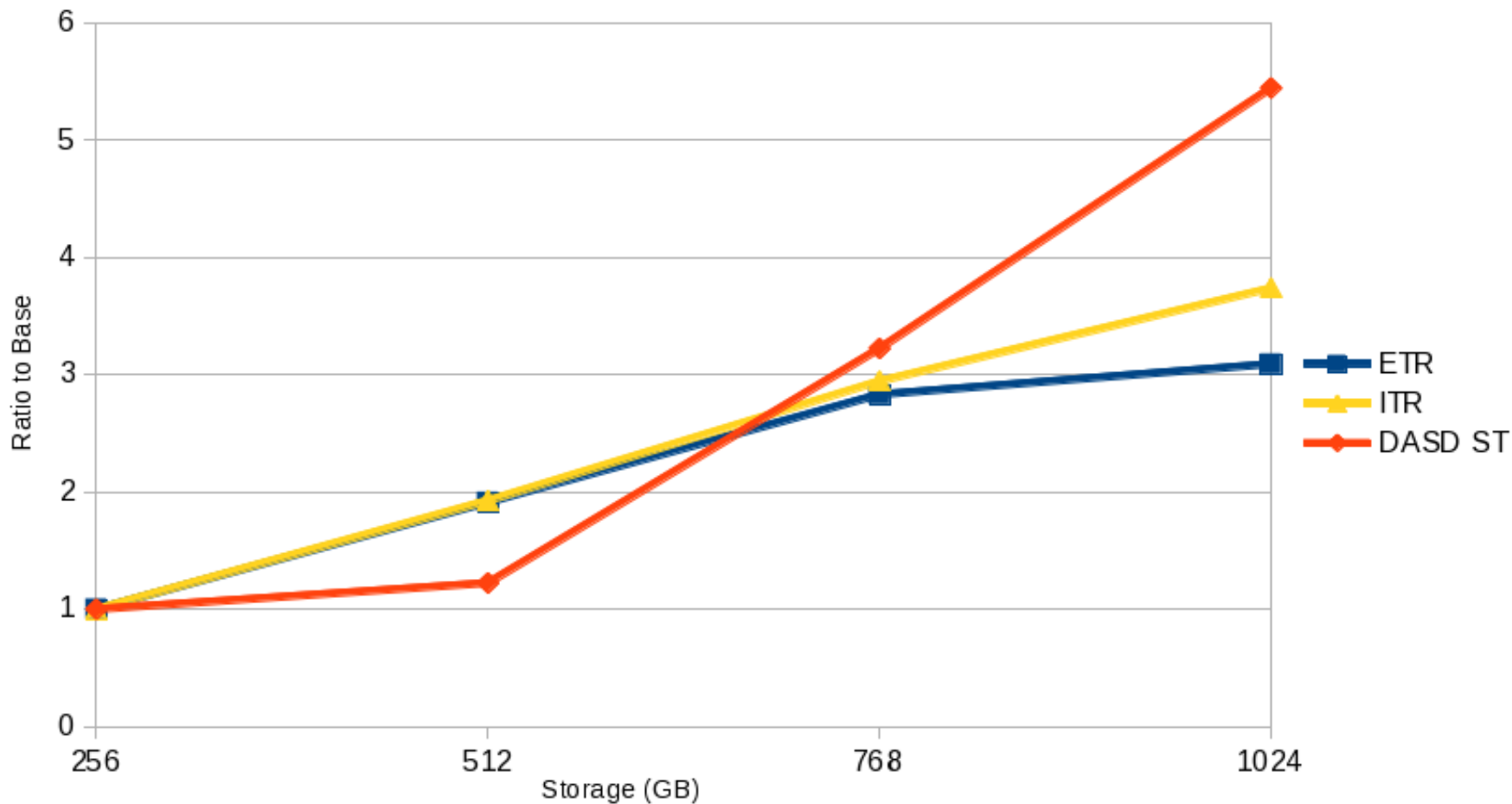
 - Scale up from there to 1 TB
 - All resources scaled up, though note that while additional DASD space was provided, it was on the same storage server.

VIRSTOR Workload in Overcommitted Environment



ETR = External Throughput; ITR = Internal Throughput; DASD ST = DASD Service Time

Apache Workload in Overcommitted Environment



ETR = External Throughput; ITR = Internal Throughput; DASD ST = DASD Service Time

CP Monitor and Performance Toolkit

Session 15741: Understanding z/VM 6.3 Through New Performance Toolkit Reports

Wednesday, 4:15 - (Bill Bitner)

Summary

z/VM Large Memory: Summary

- **Objective was to get rid of algorithmic constraints that stopped growth**

- **Things we got rid of:**
 - Reorder
 - Using the scheduler lists to visit users
 - Taking a large amount when we visit a user
 - Excessively favoring VDISKs as regards memory residency
 - Problems in evaluating depletion of available lists
 - Excessive or unnecessary rewriting of DASD
 - Dependency on long-running System z instructions

- **Things we added:**
 - Visiting all users round-robin
 - Taking only a little when we visit
 - Visiting VDISKs sooner
 - Detecting available list depletion a little more smartly
 - Scatter-to-scatter paging channel program
 - Using trial invalidation

- **Effect: workloads constrained on z/VM 6.2 should go better on z/VM 6.3**

References

- z/VM CP Planning and Administration
- z/VM CP Commands and Utilities
- z/VM Performance Report: www.vm.ibm.com/perf/

Thanks!

John Franciscovich
IBM
z/VM Design and Development
Endicott, NY

francisj@us.ibm.com

Session 15745

www.SHARE.org/Pittsburgh-Eval

