



IBM Linux Technology Center

Collaborative Memory Management in Hosted Linux Systems

*Martin Schwidefsky, Hubertus Franke,
Ray Mansell, Damian Osisek,
Himanshu Raj, Jonghyuk Choi
July 22nd, 2006*

Motivation and Problem Statement

- **System Consolidation**, i.e. the “hosting” of multiple workloads, to
 - ▶ Increase resource utilization, reduce Hardware cost
 - ▶ System management reduction
 - ▶ Licensing cost reduction
 - **Virtual Servers**: Sun Containers, Vserver, OpenVZ, Jails, chroot, ...
 - ▶ Virtualize within a single OS using OS isolation features
 - Pro: efficiency, uses OS resource sharing
 - Con: Limits fault isolation and creates OS+library version dependency
 - **Virtual Machines**: z/VM, VMWare, Xen, ...
 - ▶ Host multiple independent Guest OS
 - Pro: total isolation of guest OS, host OS + library version independent
 - Con: limited resource sharing, additional overhead
- ➔ This talk focuses on **Virtual Machines** and in particular on memory as shared a resource

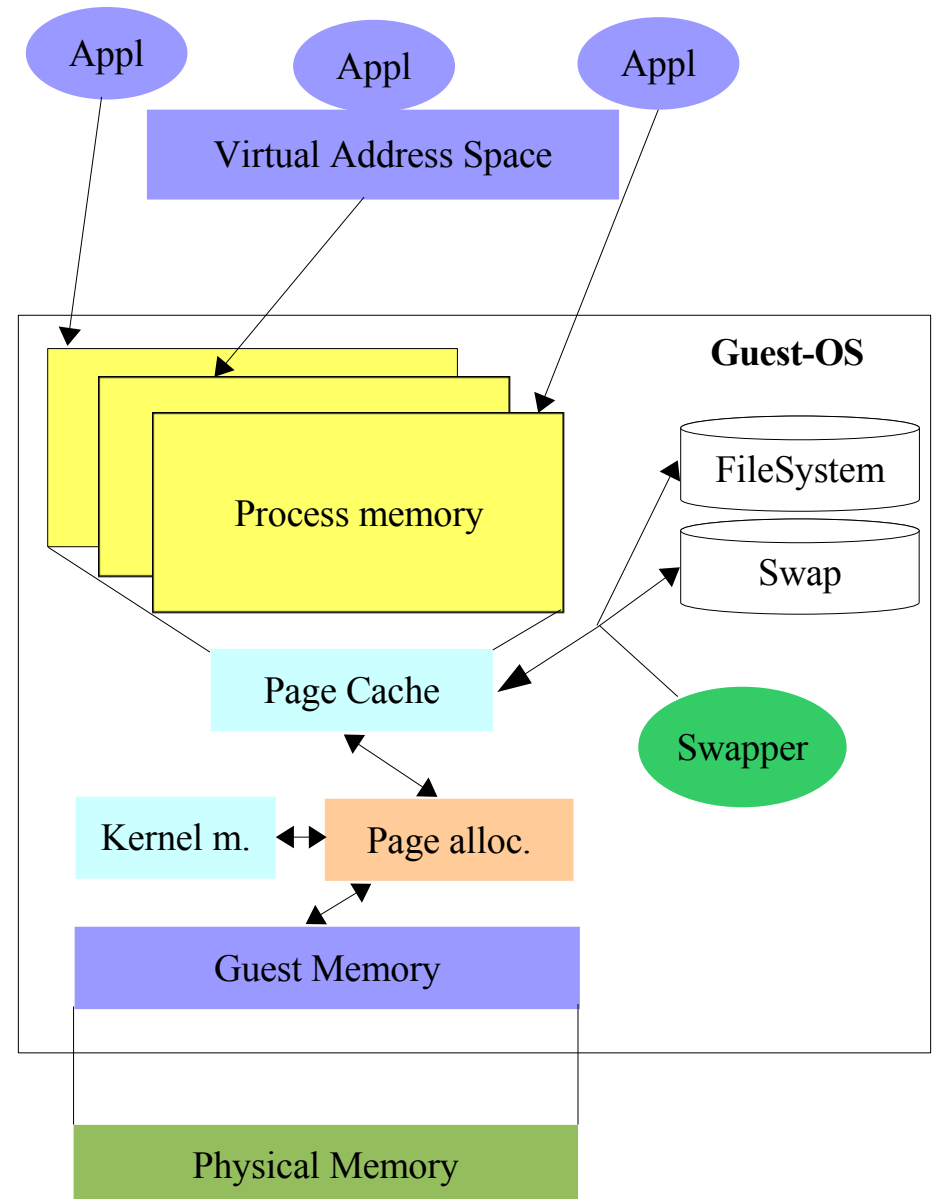
Motivation and Problem Statement - cont

- CPU and I/O are “renewable” resources with limited / fixed overhead for switching between users
 - ▶ Fine grained sharing is possible
- Memory maintains state and sharing it is difficult
 - ▶ The memory content must be preserved, I/O is expensive
 - ▶ Too frequent switching leads to trashing
- Target: large number of virtual machines ($n \times 100$ on System z)
- High level of memory overcommitment
- Workload by definition must be bursty in nature
- Memory requirements of guest's are time variant

➔ What is the right method to share memory among many guests?

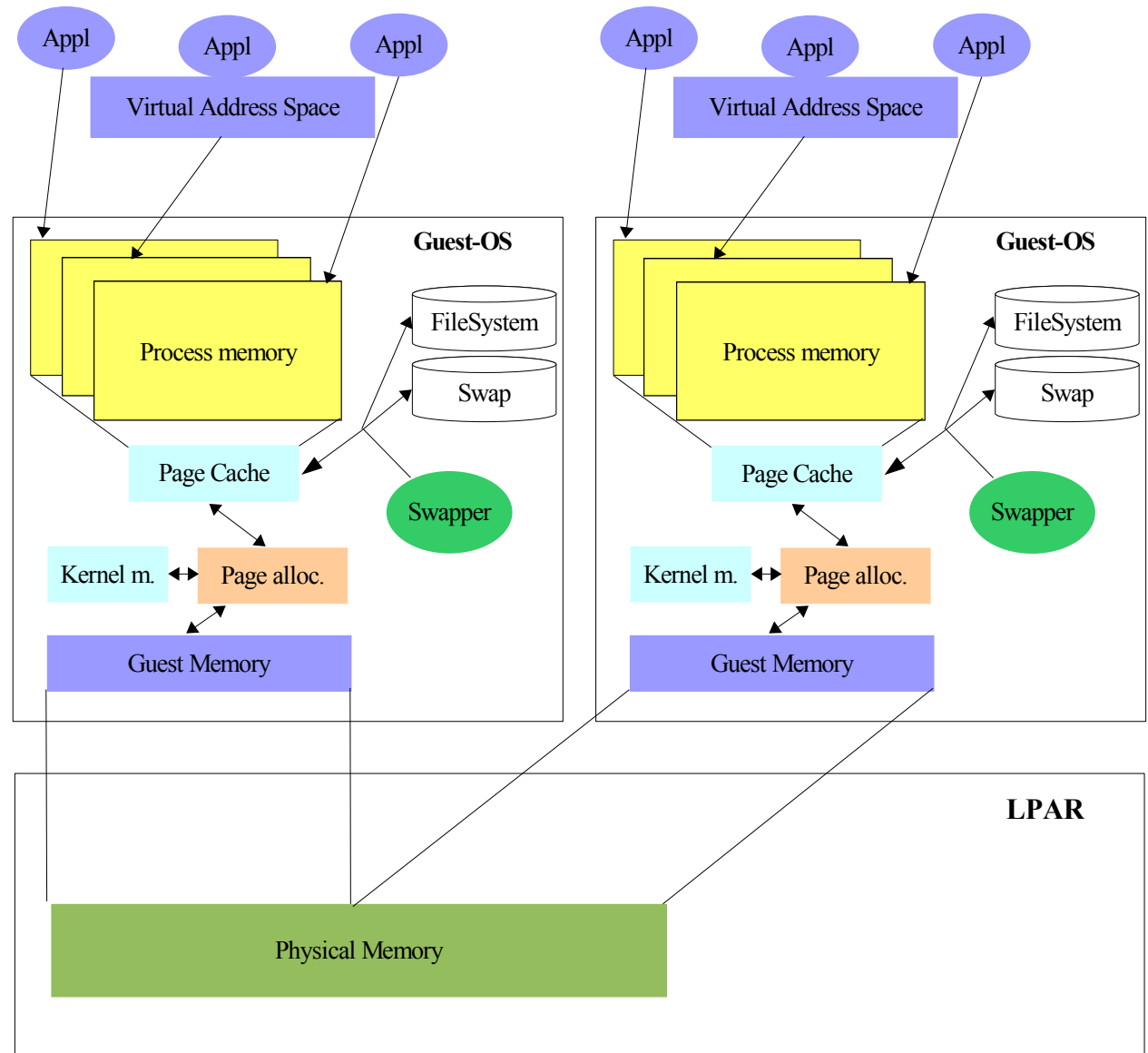
OS Virtual Memory Management (VMM)

- Multiple virtual address spaces for the different user applications
- VMM must create the illusion of a single level storage
 - ▶ Hardware provides protection and VA-PA address translation
 - ▶ Management granularity is based on pages
- Memory pressure origins from application mix
- OS determines which application pages should be retained in memory
- Page caching tries to make maximum use of the available memory



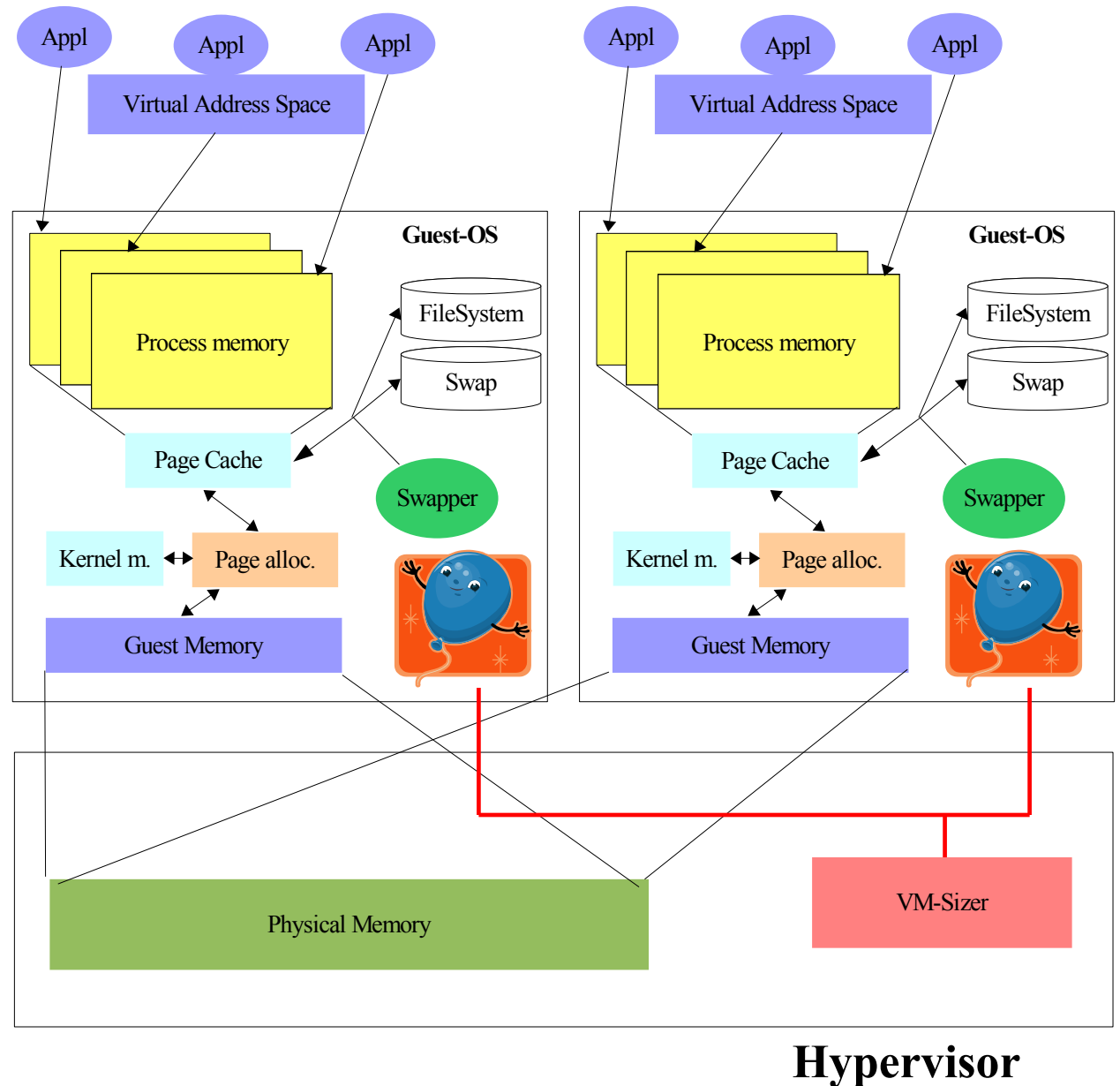
Memory Management in logical partitioned systems

- Each logical system gets a fixed amount of physical memory
- Para-virtualization or host/shadow page tables guarantee isolation
- No memory over-commitment
- easy ... and boring



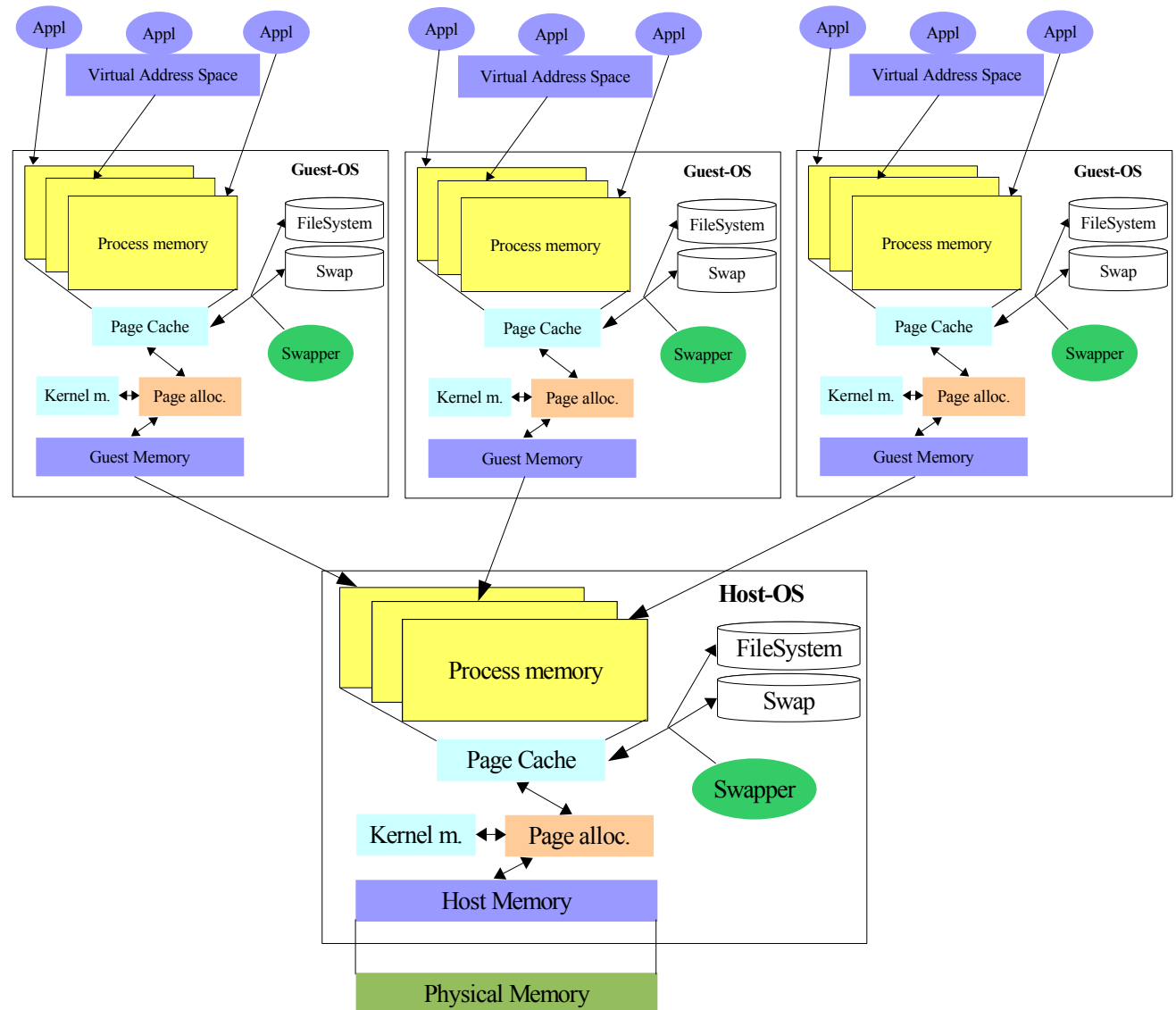
Dynamic Memory Partitioning

- Resource Manager in the host tracks memory utilization, computes target memory sizes (Working Set Size Estimation)
 - ▶ Swap rate
 - ▶ Fault Rate
 - ▶ Sampling
- Ballooner
 - ▶ Inflates / deflates to adjust memory use
- Memory pressure is forced back into the guest OS



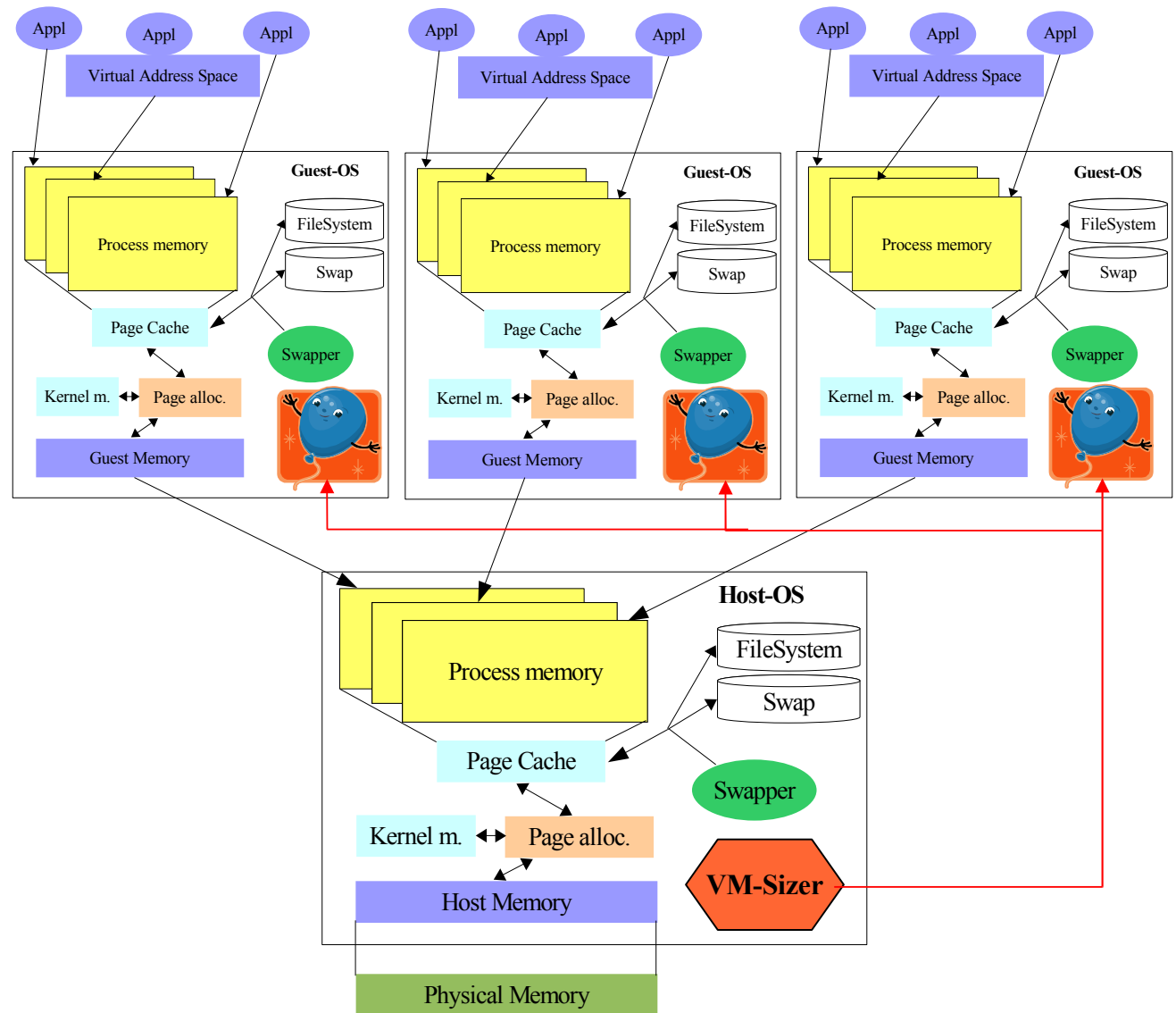
Memory management in a paging hypervisor

- Guest-OS looks like a process to the host
- Guest real memory = host virtual memory
- To make things interesting
 - ▶ Host does memory overcommitment
 - ▶ Host does on demand paging just like any OS
- Examples:
 - ▶ z/VM
 - ▶ VMware workstation



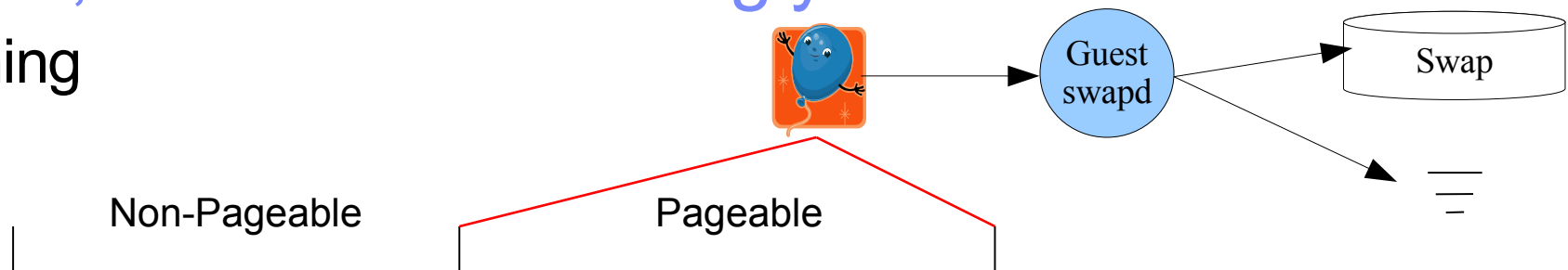
Ballooning with a paging hypervisor

- Since the Host-OS has no idea of how pages are used in the guest, it must swap each page
- High I/O rate
- Deploying balloons makes sense in order to reduce host I/O, typically the ballooner will first evict read-cached data



The good, the Bad and the Ugly ...

Ballooning



■ Advantages

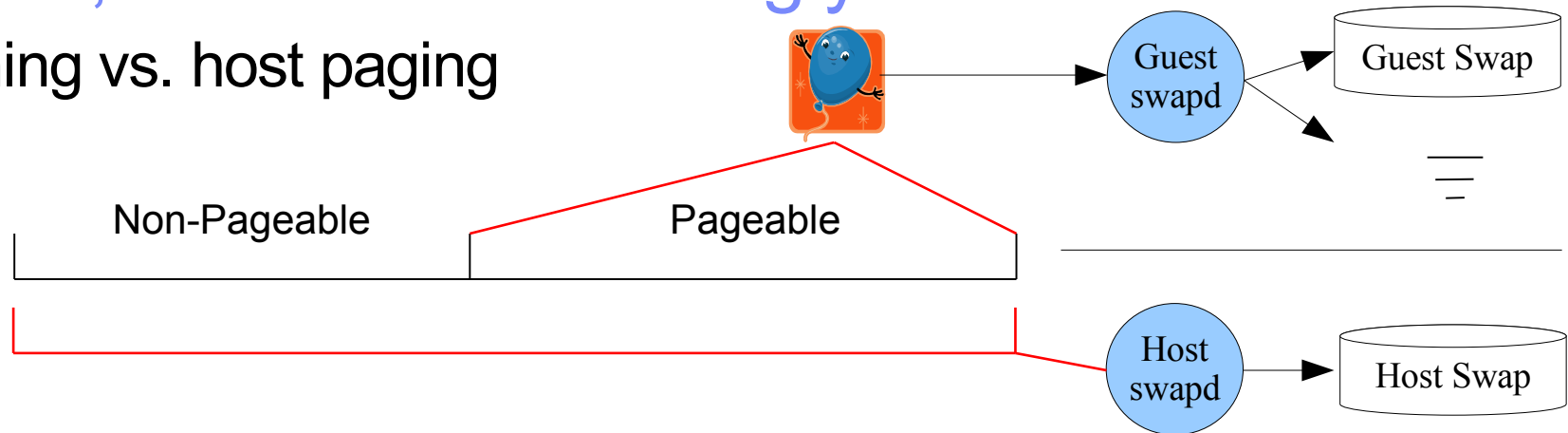
- ▶ Simple to implement on the guest OS
- ▶ Simple interface, any message channel will do
- ▶ Can get rid of pages without I/O

■ Disadvantages:

- ▶ Hard to implement on the host OS (working set size estimation)
- ▶ It is possible to crash guest OS if the balloon gets too big
- ▶ Overhead to move memory, two ballooners need to be invoked
- ▶ Non-scalable, it gets hard and harder to find freeable pages with an increasing number of guests

The Good, the Bad and the Ugly ... cont

Ballooning vs. host paging



■ Advantages

- ▶ Entire guest memory can be swapped in the host
- ▶ Host has global view

■ Disadvantages:

- ▶ Dual swap problem
- ▶ Must always preserve guest pages (host swapping)

■ We want to combine the best of both approaches in the host

- ▶ Allow host to swap guest non-page-able memory
- ▶ Allow host to throw away guest's page-able memory without invoking the guest

Guest page hinting

- Basic principle:
 - ▶ Pass usage information from pageable guest to host
 - ▶ Allow the host to “steal” pages based on the usage information
 - ▶ Deliver “discard faults” if the guest accesses pages removed by the host
 - ▶ The guest recreates content of discarded page

- Main targets are identification of
 - ▶ unused pages
 - ▶ non-dirty pages with a backing (file and swap cache)

Anticipated Benefits

- Host memory management efficiency
 - ▶ More intelligent selection of page frames to be reclaimed (unused pages)
 - ▶ Reduced reclaim overhead: avoid page writes where possible
- Guest memory management efficiency
 - ▶ Option to avoid double-clearing of pages on reuse
 - ▶ Option to favor host-resident pages on allocation requests
- **Reduce guest memory footprint without guest invocation**
- **Replace 2 host I/O operations with 1 or no guest I/O operation**
- **Reduce latency in the host to get to memory under constraints**

Page states

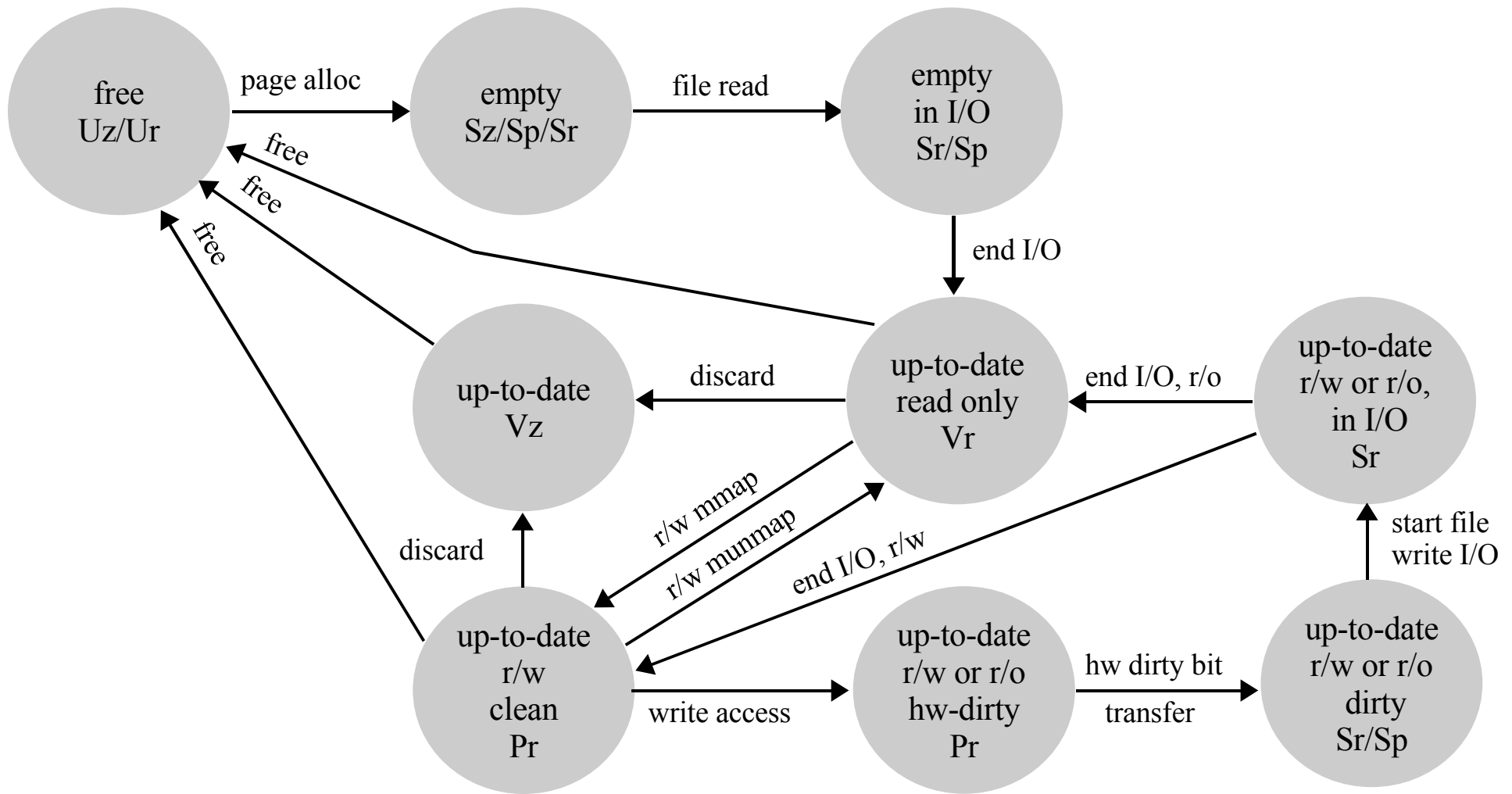
- 4 guest page states
 - ▶ **Stable (S)**: page has essential content the guest can't recreate
 - ▶ **Unused (U)**: no useful content and any access to the page will cause an addressing exception
 - ▶ **Volatile (V)**: page has useful content. The host can discard the page anytime. The guest gets a discard fault on access for discarded pages
 - ▶ **Potentially Volatile (P)**: same as Volatile (V) but host needs to check the dirty bit

- 3 host page states
 - ▶ **Resident (r)**: page is present in host memory
 - ▶ **Preserved (p)**: page is not present in host memory but the content is preserved somewhere by the host
 - ▶ **Zero (z)**: page is not present in host memory, the content is zero.

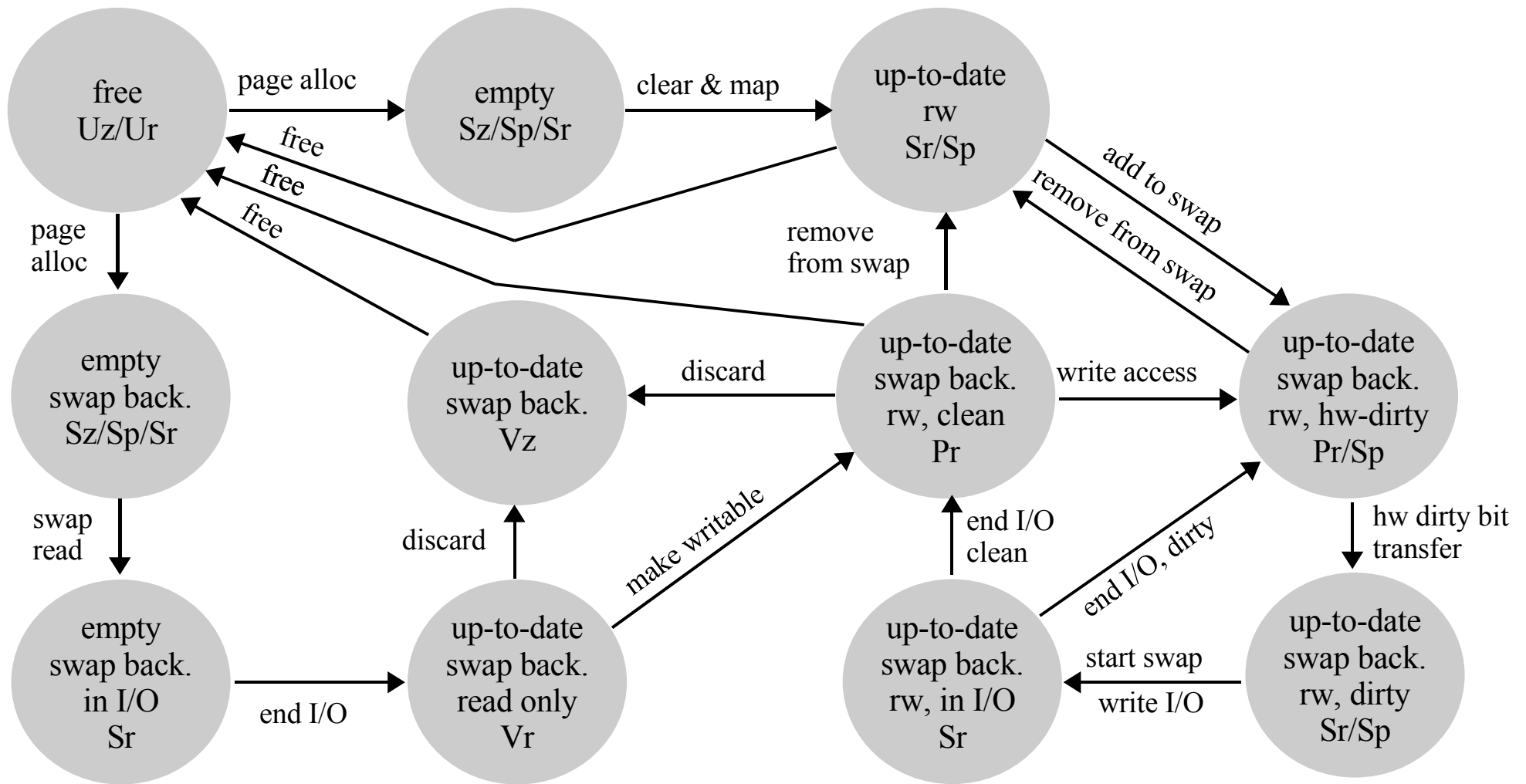
Finite state machine



Life of a file page



Life of an anonymous page

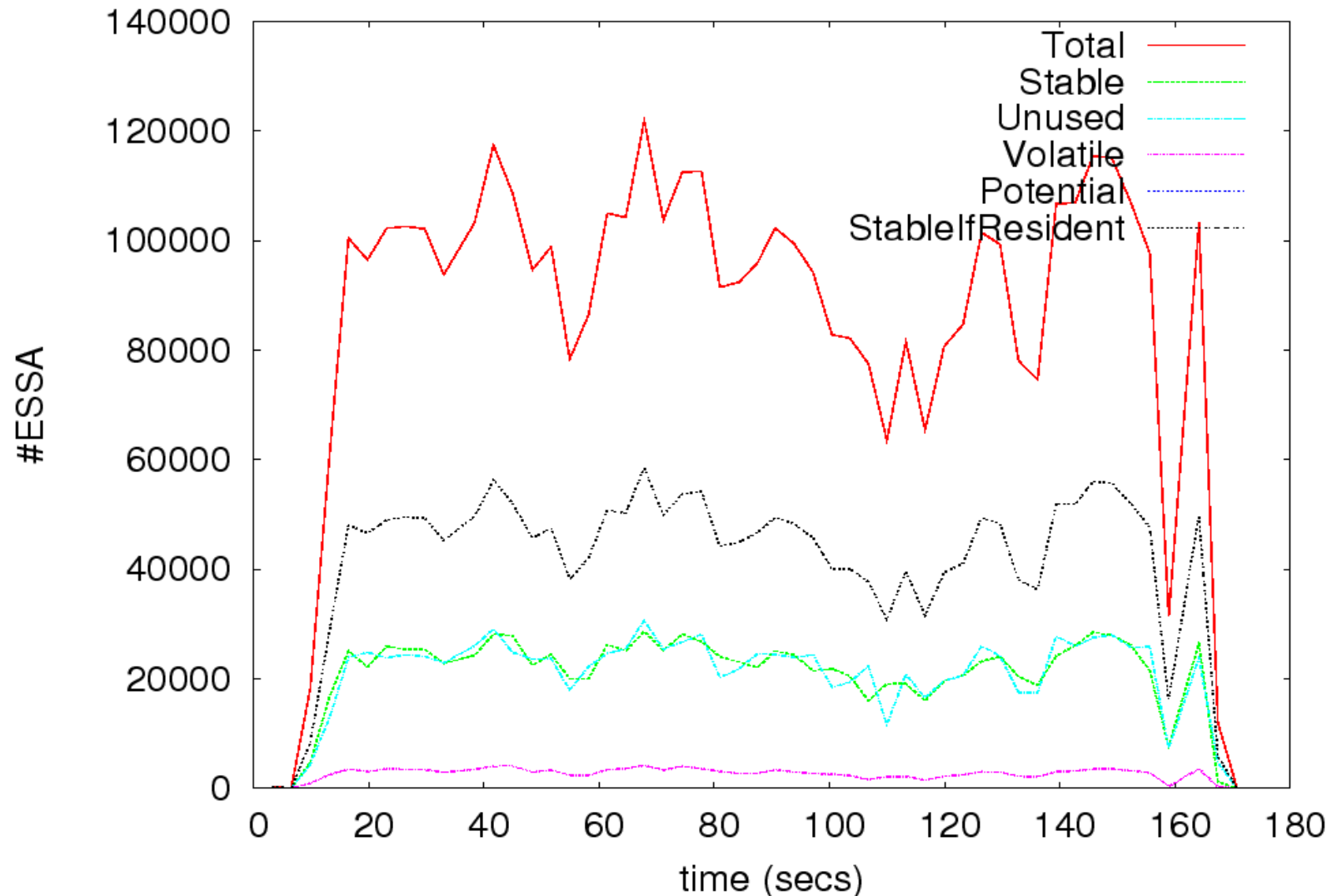


Communicating page state changes

- Guest state changes need to be fast
 - ▶ a lot of state changes occur when Linux is working
 - ▶ the state change may not cause a break into the hypervisor
- “Extract and Set Storage Attribute” instruction
 - ▶ ESSA r1,r2,m3
 - ▶ r1 (output): receives old page state
 - 2- bit guest state (Stable, Unused, Potentially Volatile, Volatile)
 - 2-bit host state (resident, preserved, logically zero)
 - ▶ r2 (input): contains guest absolute address of target page
 - ▶ m3 (immediate operand): specifies operation to be performed
“get state”, “set stable”, “set unused”, “set volatile”, “set pvolatile”,
“set stable make resident” , ”set stable if resident”
- ESSA is millicoded

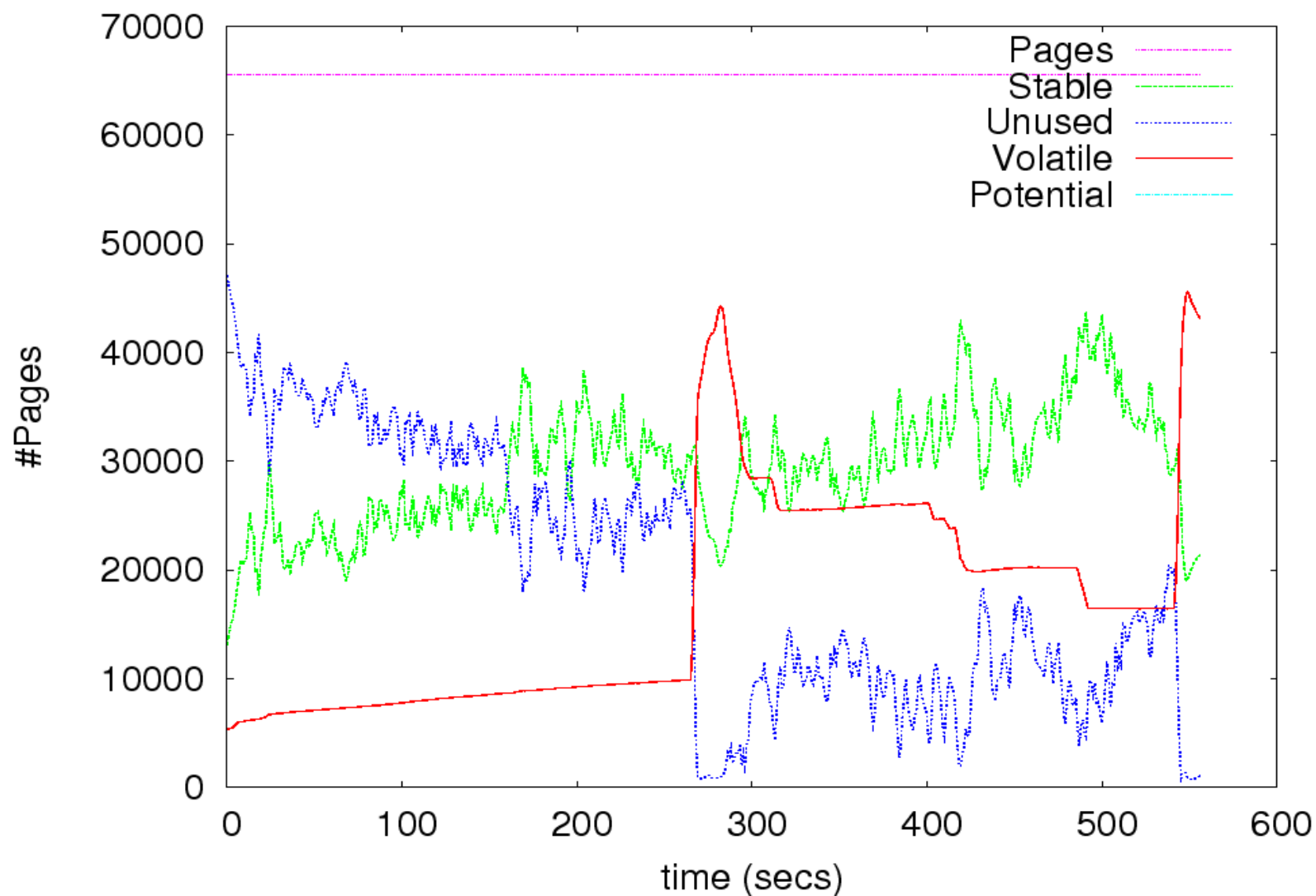
State transition overhead

- Benchmark: kernel compile on 4 way without host paging
- ~80-120K ESSA / sec, ~0.25% overhead



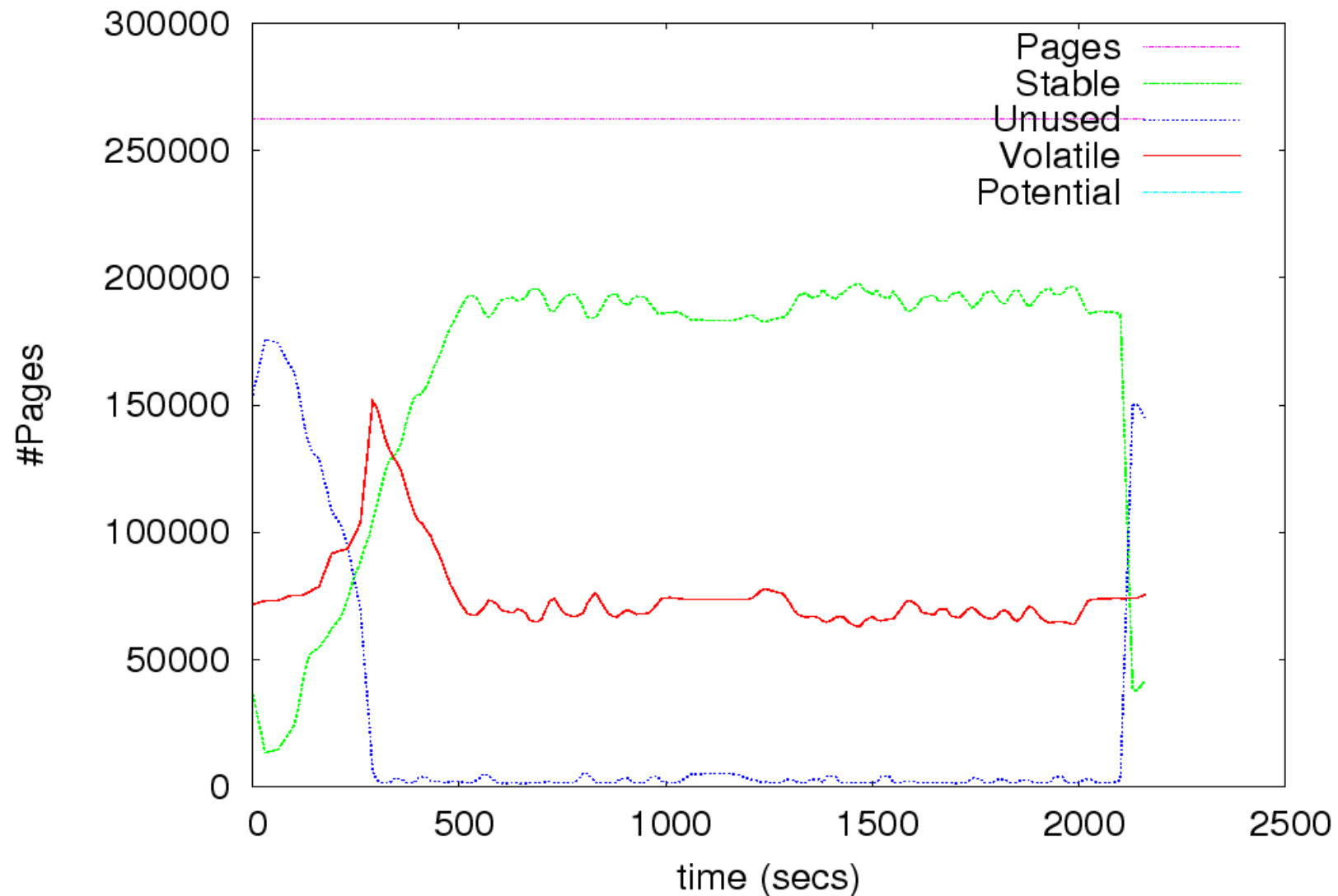
State distribution example - cont

- 2 way 256M guest during a kernel compile



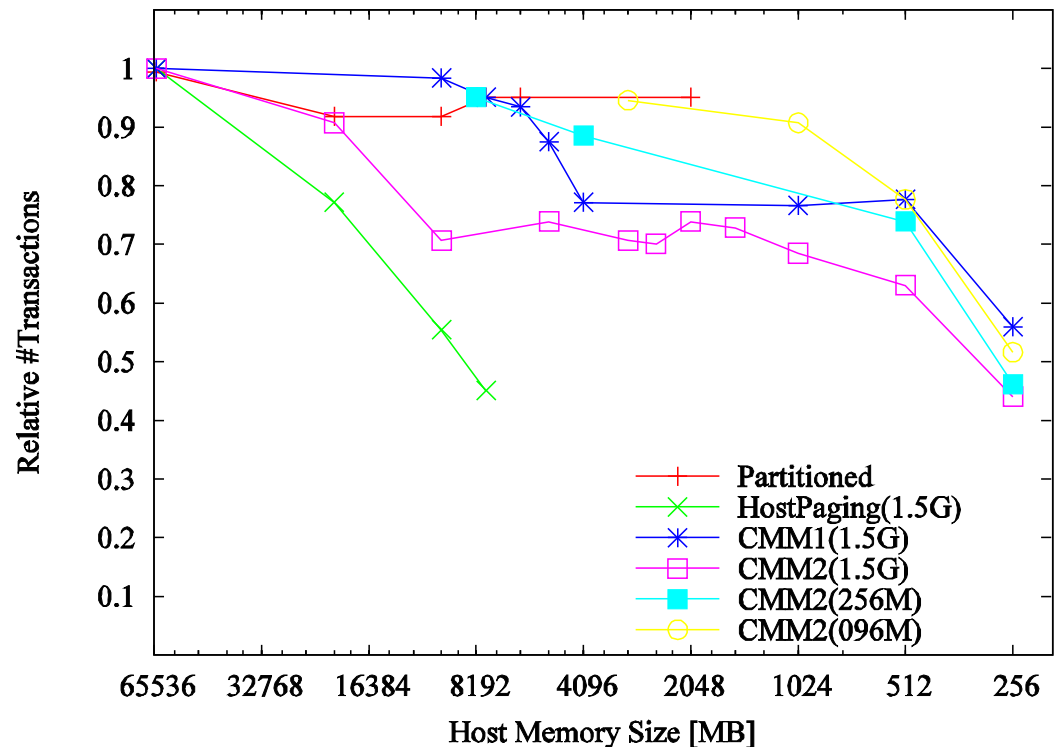
State distribution example

- 1-way 1GB guest running SpecWeb2005



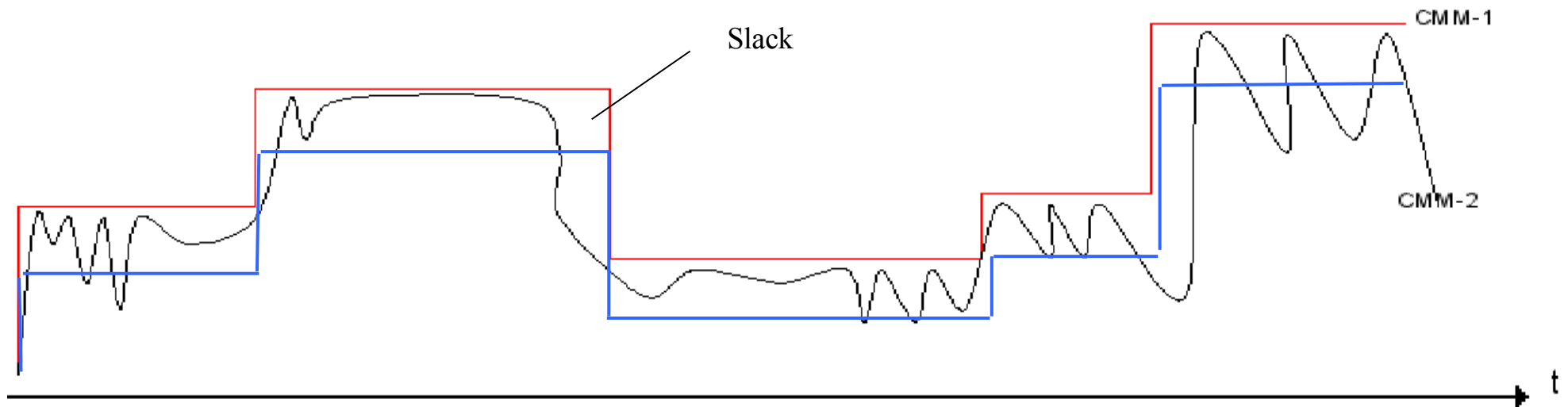
Scalability Analysis (AWM)

- N clients connect to N servers randomly, N^2 connections
- The servers are serving 1MB files from a 10GB backend
- z/VM host with large expanded storage
- host paging poor (due to high host I/O)
- ballooning outperforms guest page hinting for large guest memory sizes (1.5GB)
- guest page hinting outperforms ballooning for smaller guest memory sizes (256MB/96MB)
- Partitioning curve indicates that the working set size is small



Final solution: combine ballooner and page hinting

- Utilize ballooner for sizing an image for a stable workload with regard to memory requirement
 - ▶ Uses working set size estimation techniques to determine how big to size a guest in the **long term**
 - ▶ Slow and delayed reaction
 - ▶ Leave some slack to allow page hinting to operate
- Page hinting better for **short term** adjustments
 - ▶ Assist host's anonymous paging with status information



Questions?