NUTANIX™

Lessons Learned Building a Production Memory-Overcommit Solution

Florian Schmidt, Ivan Teterevkov

KVM FORUM, SEPTEMBER 2021

Building a Practical MemOC Solution

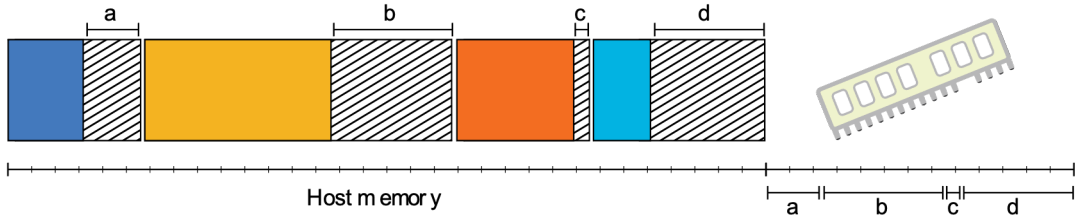
- Create a self-adapting memory overcommit solution for KVM/QEMU
- Easy, right?
- Realization: we can't write our memory management from scratch
- Instead, leverage existing technology
- Outcome: solution that uses
  - Linux MM
  - cgroups
  - virtio-balloon
  - procfs
  - … plus a central control tool that ties them all together

- This talk is about design choices and challenges on the way

Basic Reclamation Techniques

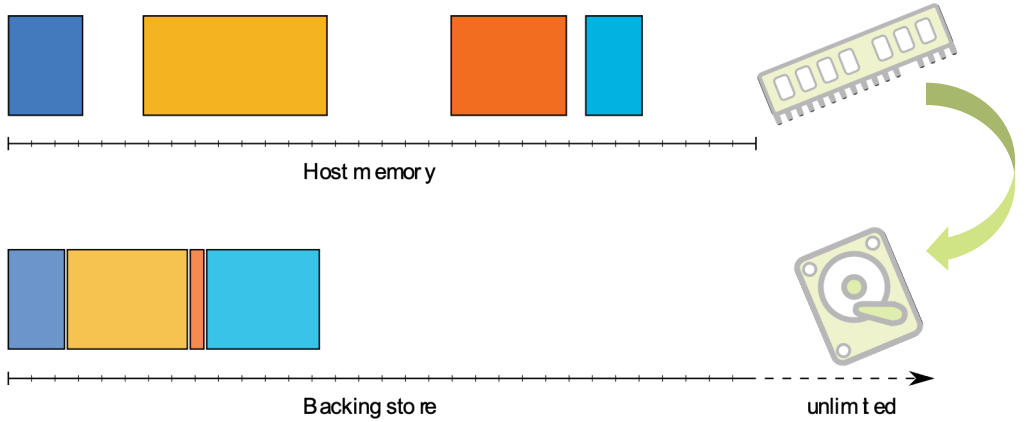
- Two main practical solutions: ballooning and hypervisor swap

Ballooning



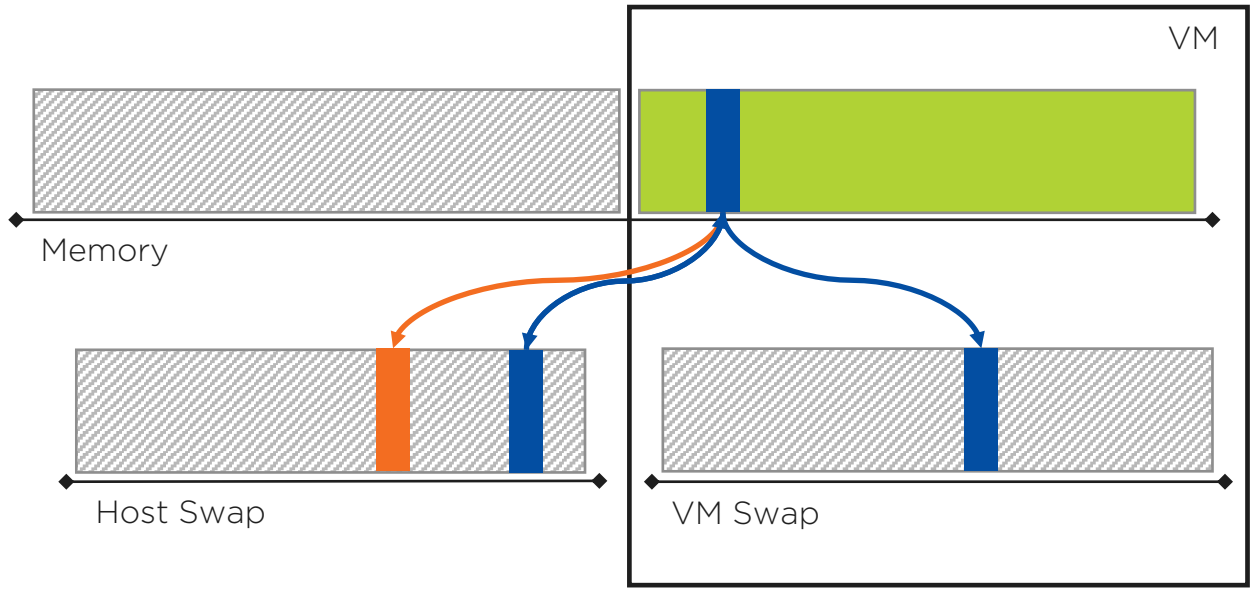
- virtio driver inside guest hands memory back
- + Guest can choose memory to give up
- + Might not cause swapping
- – Requires guest cooperation
- – State lost on guest reboot

Hypervisor swap



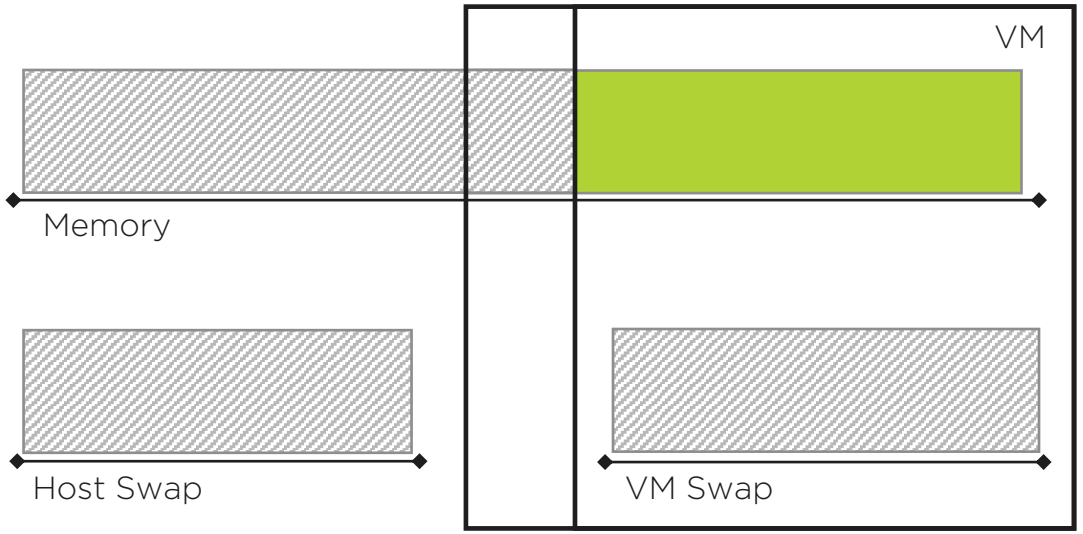
- Treat VM like application: swap out, control via cgroup
- + No guest cooperation required
- – Performance (I/O)
- – “Double swapping”

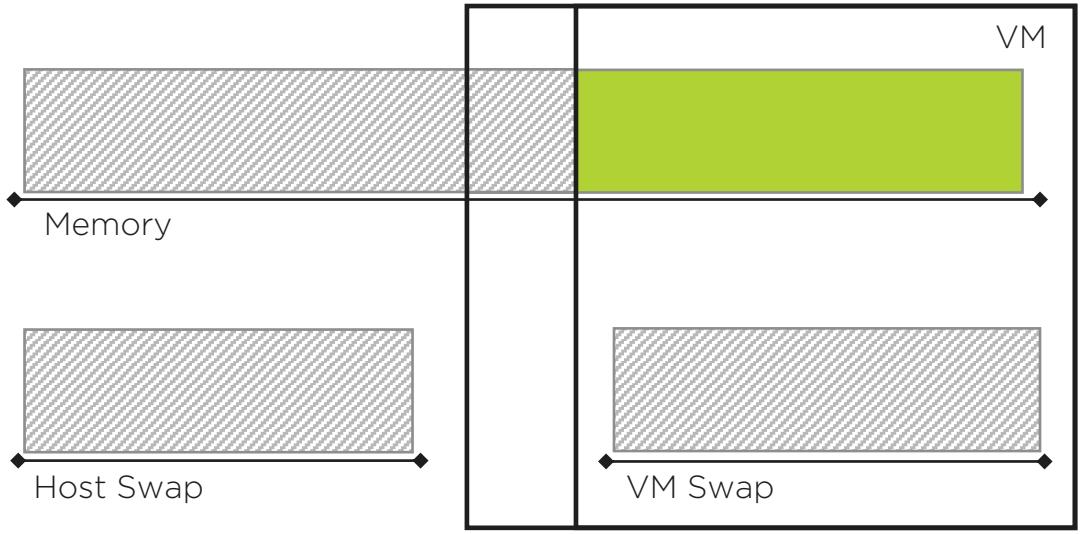
Double Swapping



- Host finds idle memory to swap out
- Shortly after, guest also finds that memory to swap out
- Result: swap in -> out cycle
  - Or even out -> in -> out
- Potential memory thrashing, lots of I/O -> bad performance
- Well-functioning memory management can make this worse: likely find the same pages

Hybrid Overcommit

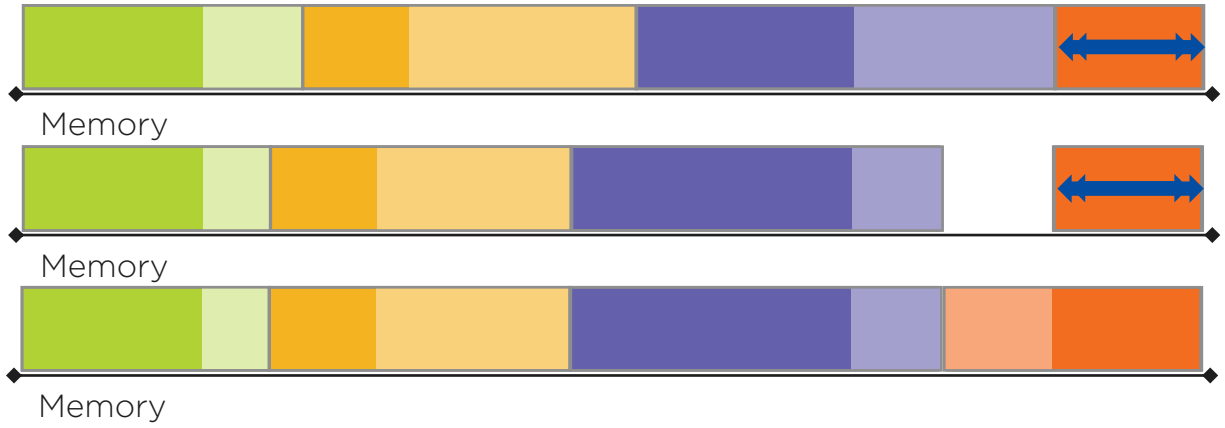




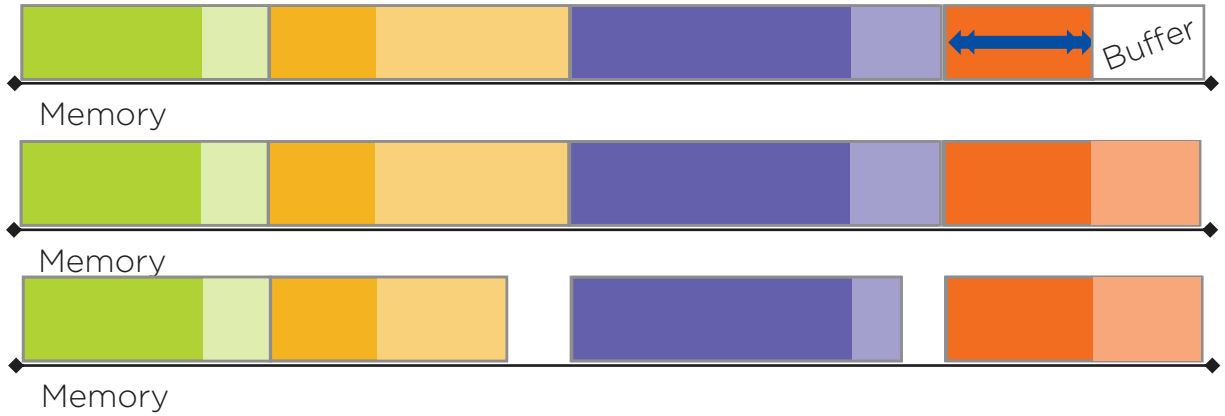
- Problem: Ballooning might not be available/reliable, hypervisor swap has performance issues
- Combine to get the best of both worlds
- Guiding principle: use ballooning where possible, fall back to hypervisor swap
  - Shrink VMs: balloon out memory before reducing cgroup limit
  - Grow VMs: increase cgroup limit before ballooning in memory
- Give up on balloon if it doesn't progress 

Overcommit Memory Buffer

- Problem: want to react quickly to growth demands
  - To grow a VM, need to shrink others
  - Shrinking a VM can be slow (balloon API, swap I/O)



- Solution: Keep some memory "buffer" unused by any VM
  - Grow quickly, then reclaim buffer afterwards to prepare for next event



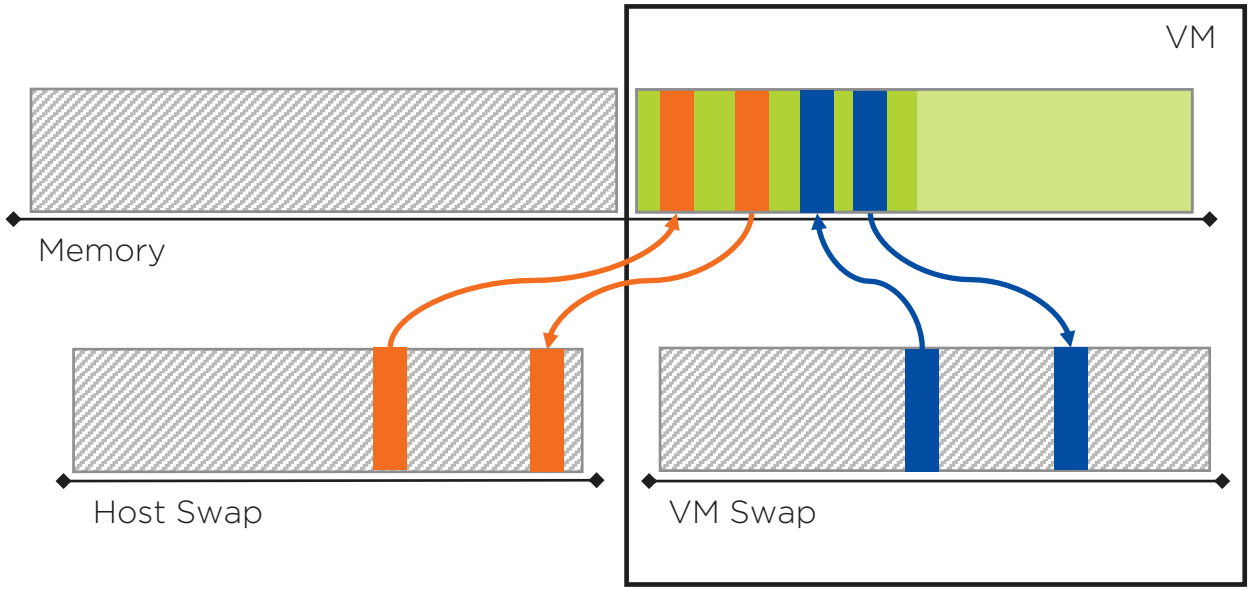
- Tradeoff: reaction speed vs efficiency

Memory Stats

- Problem: How do we know which VMs need memory, and which ones can give up memory?

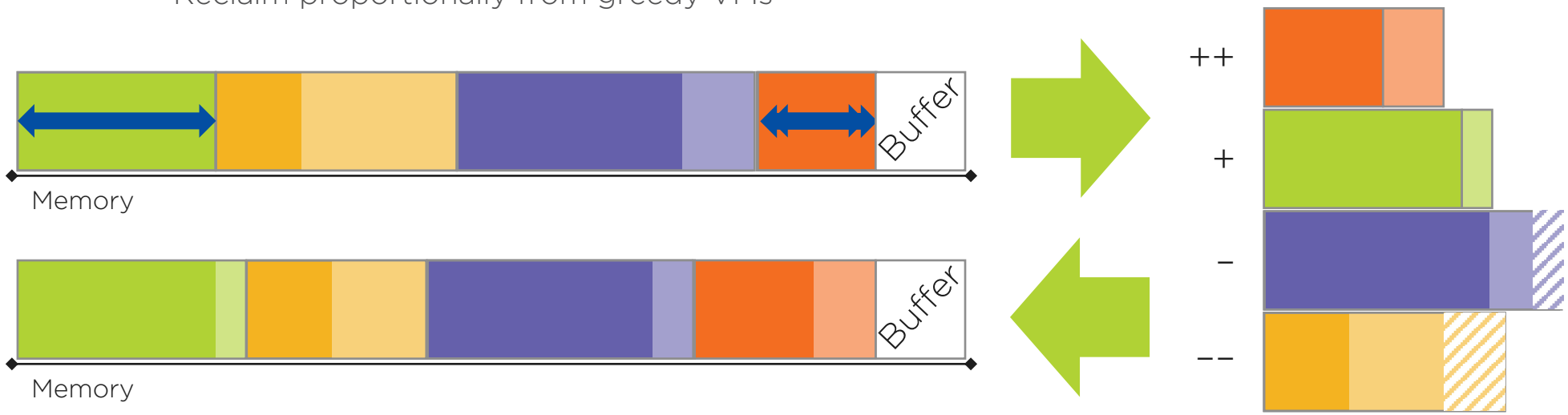
- VM-level: via balloon driver
  - Guest swap in / swap out
  - Reclaimable memory: “usable”

- Hypervisor-level:
  - Host swapin: majflts of QEMU
  - Host swapout: no direct way
  - Reclaimable memory: WSS estimation (more later)



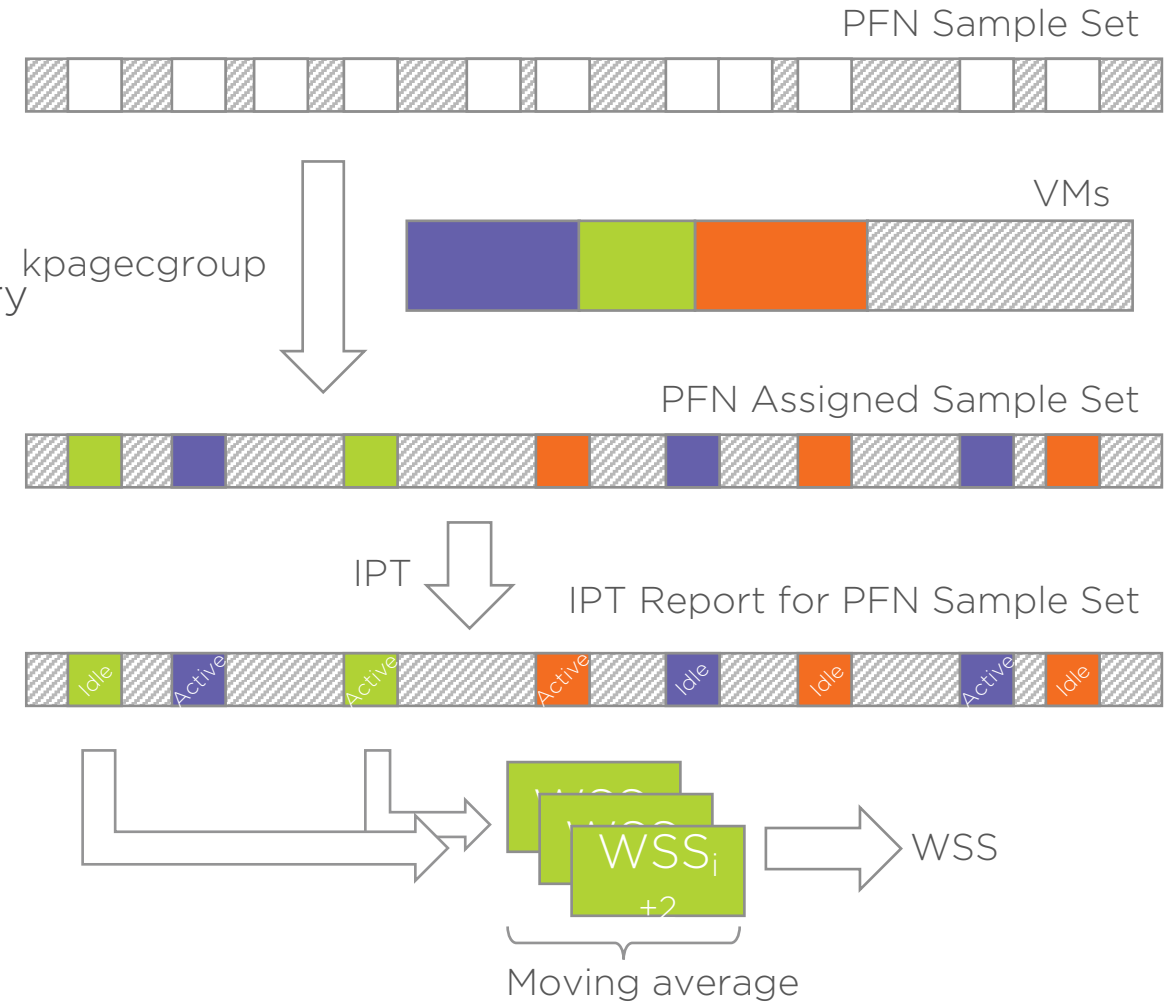
A Simple Memory Overcommit Algorithm

- How to derive grow/shrink decisions from stats?
  - A VM can be “needy”: (significant) swapping → needs more memory
  - … or “greedy”: has unused memory
- Problem: We *can’t know* how much memory a needy VM needs
- Algorithm:
  - Order by “neediness”
  - Hand out based on list position and growth potential
  - Reclaim proportionally from greedy VMs



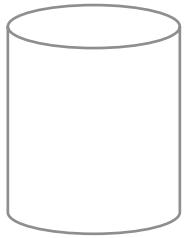
Working Set Size (WSS)

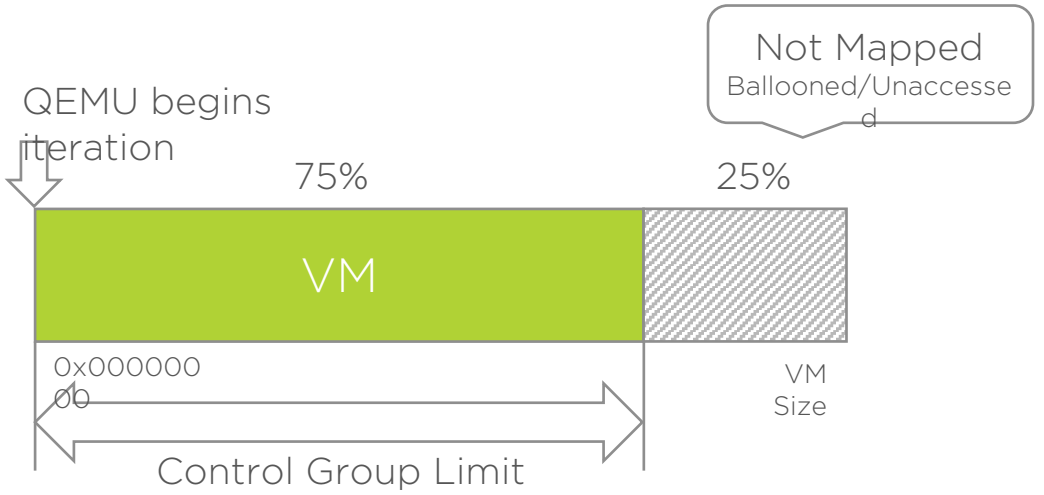
- Reliable and accurate metric
  - Trusted – does not rely on guest
  - Host computes per-VM metric
- Higher estimate means that VM needs memory
- Based on Linux Idle Page Tracker (IPT)
  - /sys/kernel/mm/page_idle
- How to select Page Frame Numbers (PFN)?
  - Sample guest memory – /proc/pid/pagemap
  - Sample host memory – /proc/kpagecgroup
- How to address "noise"?
  - Post-processing: moving average

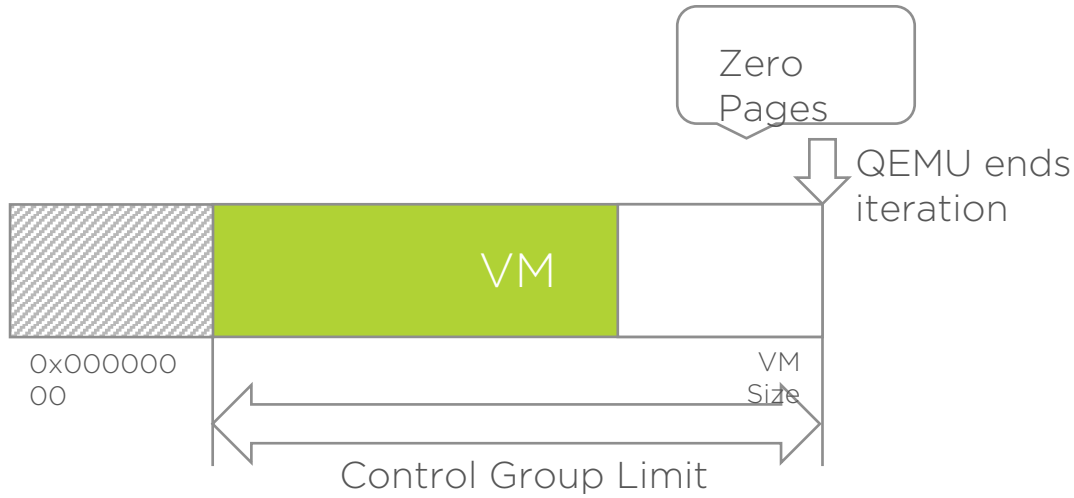


Live Migration

- Applicable to shared memory
- Problems
  - Unnecessary allocation of zero pages
  - Unnecessary host swap I/O
  - Mangled working set of VM
- A solution
  - Use madvise() with MADV_COLD
- Algorithm
  - Check if PM_PRESENT is unset in pagemap
  - Check if page is zero
    - Causes allocation if unmapped
  - Call madvise() with MADV_COLD

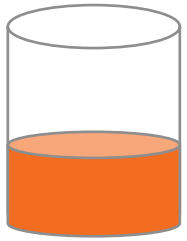


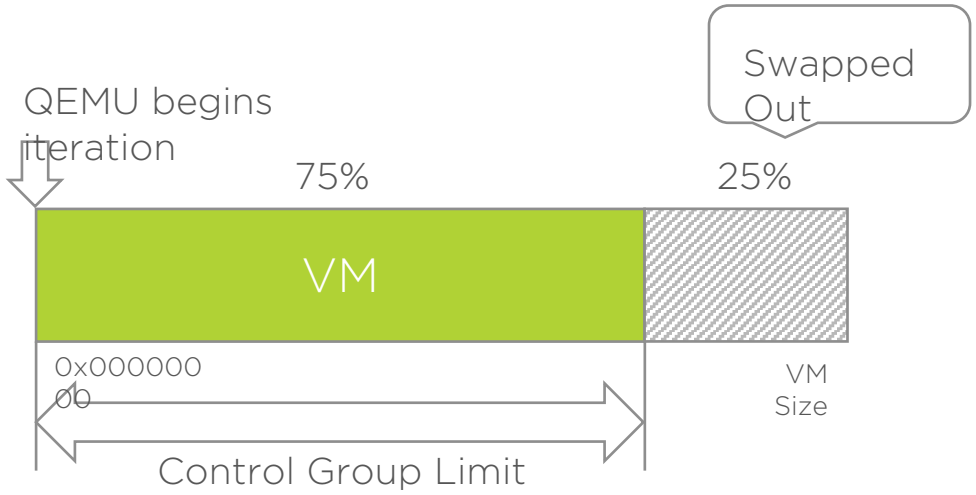


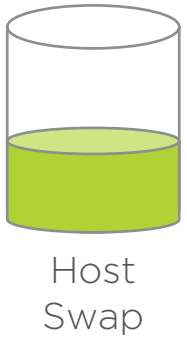


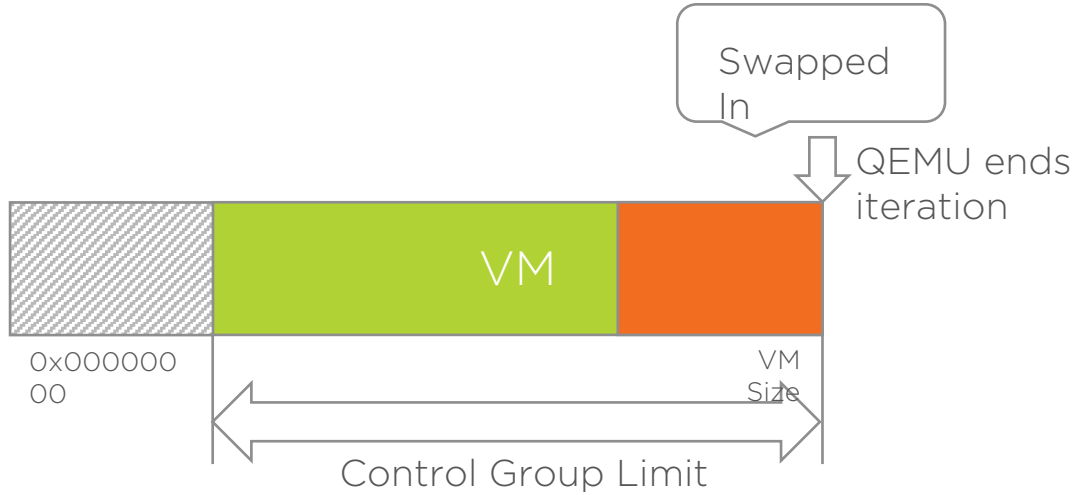
Live Migration (continued)

- Applicable to shared memory
- Problems
  - QEMU reads all guest memory
  - Unnecessary host swap-out
  - Mangled working set of VM
- A solution
  - Use madvise() with MADV_PAGEOUT
- Algorithm
  - Identify swapped-out pages
  - Transfer to destination
    - Causes swap-in
  - Call madvise() with MADV_PAGEOUT









Identify Swapped-out Pages

- How to identify paged-out shared pages?
  - Currently pagemap interface does not support PM_SWAP and other flags for shared memory
    - (Pagemap works well with private memory)
- Possible solutions
  - Improve pagemap implementation to query XArray swap-cache
  - Alternatively, use lseek() with SEEK_DATA and SEEK_HOLE along with mincore()

Future Work

- Avoid transferring swapped-out pages during live migration
- Use shared swap space
  - Accessible to both source and destination hosts, e.g. NAS, SAN
  - Transfer metadata only

Lessons Learned

1. Use swapping and ballooning for memory overcommit
2. Combine stats and deduce need and greed
3. Give memory in proportion to ranked needs
4. Use sampling with idle page tracker
5. Shared memory != private memory
6. Use madvise in QEMU with shared memory
7. Improve pagemap to identify swapped shared memory
8. Linux ecosystem is solid foundation for memory overcommit
9. … just needs something to tie pieces together