The Role of Migrations in Data Center Management

ANDREA SEGALINI

VIRTUALIZED COMPUTING INFRASTRUCTURE - M2 IFI/UBINET

8/2/2021

Why this presentation

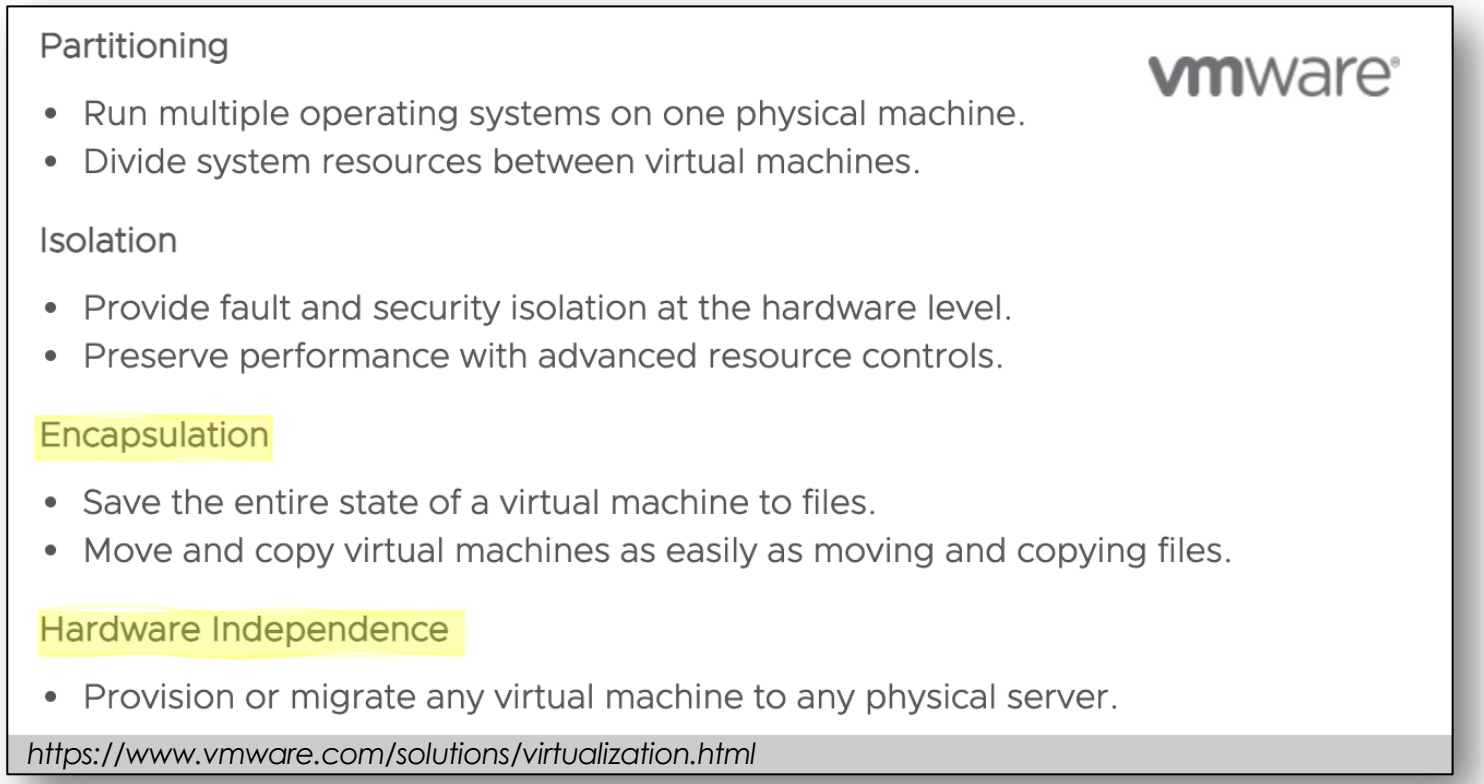
- PhD candidate with Signet team (I3S)

- In our group we research virtualized infrastructures
  - Networking (data center networks, overlays, SDN, etc.)
  - Computing (Hypervisors, containers, resource management, etc.)

- Specifically, I study:
  - Resource efficiency for data centers (DCs)
  - Host maintenance in DCs
- Both these activity leverage migration

Virtualization

- Virtualization is the corner stone of cloud computing
  - All its flavors: full virtualization (VMs), OS level (container), network (VNFs), etc.
- Let's focus on VMs as a reference, the perks are:



Partitioning

- Run multiple operating systems on one physical machine.
- Divide system resources between virtual machines.

Isolation

- Provide fault and security isolation at the hardware level.
- Preserve performance with advanced resource controls.

Encapsulation

- Save the entire state of a virtual machine to files.
- Move and copy virtual machines as easily as moving and copying files.

Hardware Independence

- Provision or migrate any virtual machine to any physical server.

https://www.vmware.com/solutions/virtualization.html

- We can migrate VMs!

Migrations in DC Management

- What are the reasons to migrate within a DC?

(1) Resource management:

- Load balancing
  - Run out of resources (CPU, RAM, BW, etc.) on a host (Hotspot mitigation)
- Consolidation
  - Too many unused resources on a host (reduce hosts powered-on)

(2) Fault-recovery and Maintenance:

- Incoming disaster (e.g., Earthquake warnings in Japan, floodings)
- DC power infrastructure maintenance
- Host HW Failure
  - Bad DIMMs, bad disks, etc.
- HW upgrades
  - Install new RAM, CPUs, deploy new net fabric
- Host SW upgrades
  - Platform SW upgrades
  - Hypervisor upgrades

Flavors of Migration

- Migrations is “moving” a VM from host A to B
  - *How* is what defines the type of migration

- Traditional VM migrations
  - VM termination/restart
  - VM cold migration
  - VM live migration
    - Pre-copy
    - Post-copy

VM Termination/Restart

- Most simple migration

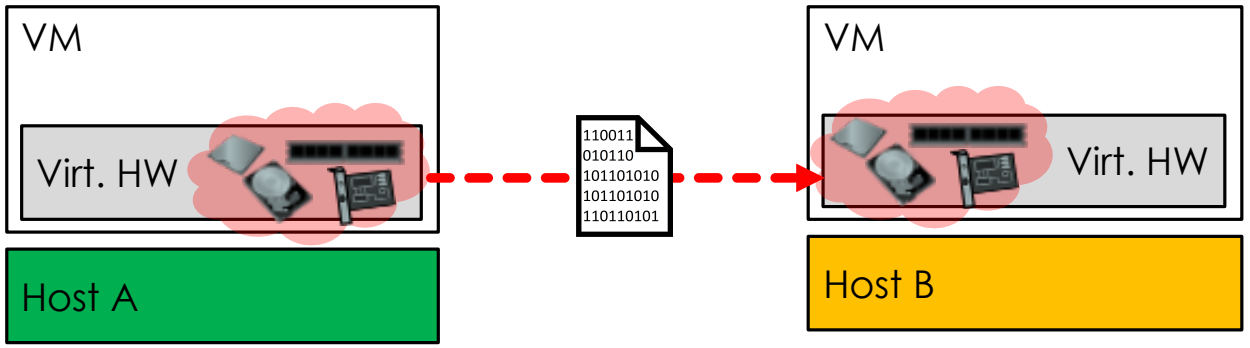
1. VM shut-down
2. Moves relevant VM config files:
   - booting disk (e.g., qcow, vdi)
   - VM configuration (vCPUs, vNICs, etc.)
   - etc…
3. VM started on new host

- Fast migration
  - Boot time of the VM
- No state preserved
  - Partial results lost
  - Stateful services need to warm-up again (e.g., Cache servers)

VM Cold Migration

- Known by many names
  - Checkpoint/restore, stop&copy, Suspend-to-disk/Resume-from-disk, etc.

1. VM paused
2. Serialize virtual hardware state (vCPU registers, VM RAM content, etc.) into a stream of bytes
3. Serialized state transferred to destination
   - Typically using the network
4. VM resumes from saved state on another host

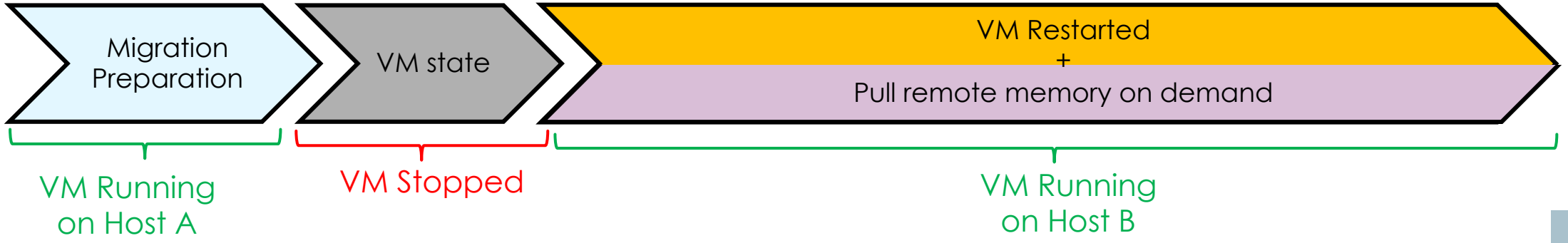


- State entirely preserved
- VM stopped for long time (transfer state)
  - Especially serializing RAM
  - Great Downtime

VM Live Migration (I)

- Of all the parts that make up a VM state, RAM is the largest
  - Many Gigabytes (1TB VMs exists!) against few MBs for vCPUs, virtual network cards (vNICs), etc.
- RAM transferred while the VM runs:
  - Pre-copy: iterative transfer of RAM

  - Post-copy: On demand memory transfer



VM Live Migration (II)

- Live migrations because VMs are paused for a brief time, imperceptible from the outside

- Pre-copy
  - State preserved
  - Little downtime (50-200ms[1])
  - Same memory region transferred several times
  - For very active VM may not converge

- Post-copy
  - State preserved
  - Little downtime (50-200ms[1])
  - Memory transferred once
  - Access to remote memory take x100[2]
  - Failure on any of the host = VM failure (more fragile)

[1] *Ruprecht, Adam, et al. "VM live migration at scale", 2018*

[2] *https://gist.github.com/hellerbarde/2843375*

Flavors of Migration

- Migrations is "moving" a VM from host A to B
  - *How* is what defines the type of migration

- Traditional VM migrations
  - VM termination/restart
  - VM cold migration
  - VM live migration
    - Pre-copy
    - Post-copy

- New flavors
  - VM-to-container migration
  - Zero-copy local-host migration

Migrations in DC Management

- What are the reasons to migrate within a DC?

(1) Resource management:

- Load balancing
  - Run out of resources (CPU, RAM, BW, etc.) on a host (Hotspot mitigation)
- Consolidation
  - Too many unused resources on a host

We'll see how:

- VM-to-container migration to improve resource utilization (Massive consolidation)
- Zero-copy local-host migration to improve efficiency of host SW upgrades

(2) Fault-recovery and Maintenance:

- Incoming disaster (e.g., Earthquake warnings in Japan, floodings)
- DC power infrastructure maintenance
- Host HW Failure
  - Bad DIMMs, bad disks, etc.
- HW upgrades
  - Install new RAM, CPUs, deploy new net fabric
- Host SW upgrades
  - Platform SW upgrades
  - Hypervisor upgrades

Migrations in DC Management

- What are the reasons to migrate within a DC?

(1) Resource management:

- Load balancing
  - Run out of resources (CPU, RAM, BW, etc.) on a host (Hotspot mitigation)
- Consolidation
  - Too many unused resources on a host (reduce hosts powered-on)

(2) Fault-recovery and Maintenance:

- Incoming disaster (e.g., Earthquake warnings in Japan, floodings)
- DC power infrastructure maintenance
- Host HW Failure
  - Bad DIMMs, bad disks, etc.
- HW upgrades
  - Install new RAM, CPUs, deploy new net fabric
- Host SW upgrades
  - Platform SW upgrades
  - Hypervisor upgrades

VM-to-container Migration for Massive Consolidation

Server Consolidation 101- DC Costs

- Resources (CPUs, RAM, disks, bandwidth, etc.) are used to power users' workloads (VMs)
- Equipment (hosts, racks, net switches, etc.) provide those resources
  - Equipment costs money (CapEx)
  - Equipment consumes energy (OpEx)
    - DCs release 43M tons/year of CO_2 (2010)[1]

Total Cost of Ownership (TCO) for a 10k sq. ft. data center[2]:

	Unit	Estimated Cost	Annualized Cost
Data Center			
Electrical and Cooling	Cost/Watt	$ 15	$4.0 M
10,000 Square Foot Building	Cost/Square Foot	$ 300	$0.20 M
Facility Depreciation	Years	15	--
PUE	(Unitless)	1.5	--
Server Cost	per Server	$ 5000	$5.0 M
Server Refresh Lifecycle	Years	5	--
Electricity Cost	per Kwh	$ 0.10	--
Server Average Power	Watts	300	$3.1 M
TOTAL	--	--	$12.3M

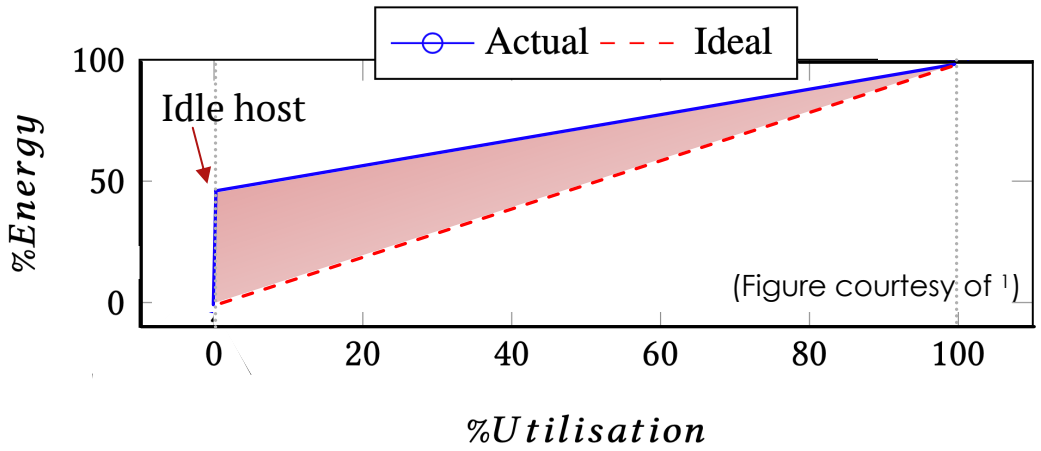
- ~25% of OpEx is electricity
- Reduce power consumption is key to reduce costs

[1] *Helali, Leila, and Mohamed Nazih Omri. "A survey of data center consolidation in cloud computing systems.", 2021*

[2] *Gough, Corey and Steiner, Ian and Saunders, Winston, Energy Efficient Servers: Blueprints for Data Center Optimization, 2015*

Server Consolidation 101- Maximize Utilization

- Let's focus on hosts, hosts are not energy proportional



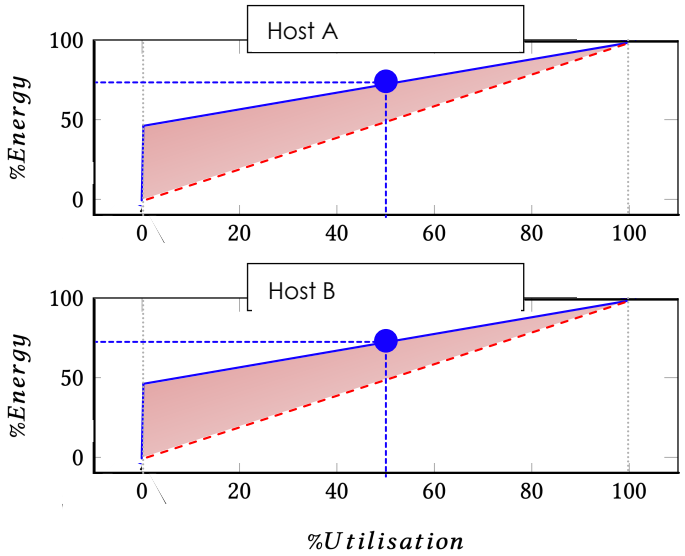
- Host do not consume proportionally to their load

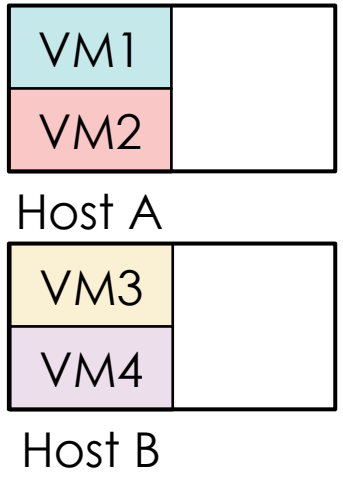
Better shut down a machine than keeping it underloaded

- That's why **server consolidation**:
  1. Pack VMs on same hosts to maximize resource utilization
  2. Power-off empty hosts

[1]*Nitu, Vlad, et al. "Welcome to zombieland: Practical and energy-efficient memory disaggregation in a datacenter."*

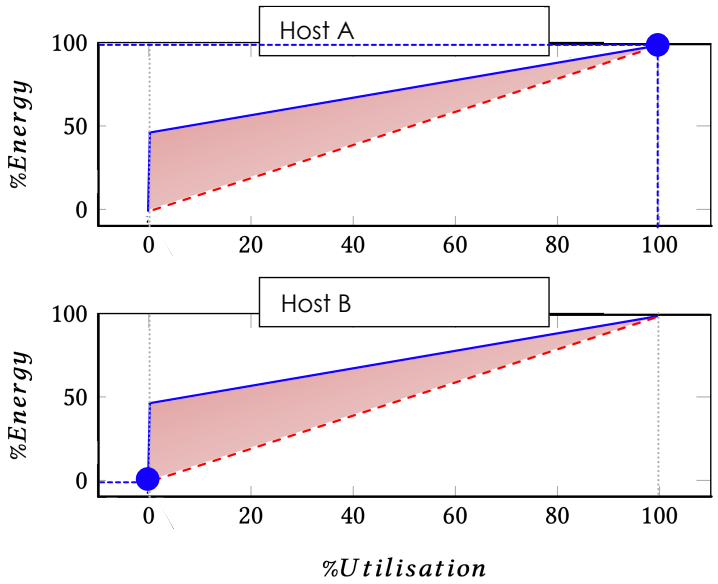
No consolidation

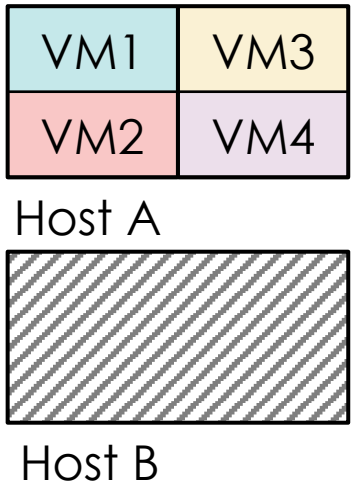




- Both hosts half loaded
  - Total consumption: 150%

Consolidation

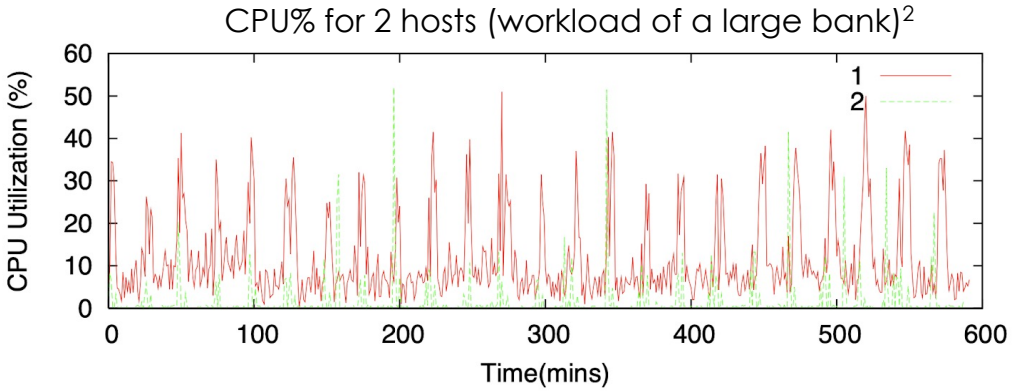




- Host fully loaded & Host shut down
  - Total consumption: 100%

Server Consolidation 101- Oversubscription (I)

- Reality is not as simple…



- DC utilization reported between 5-25% up to 40-60%

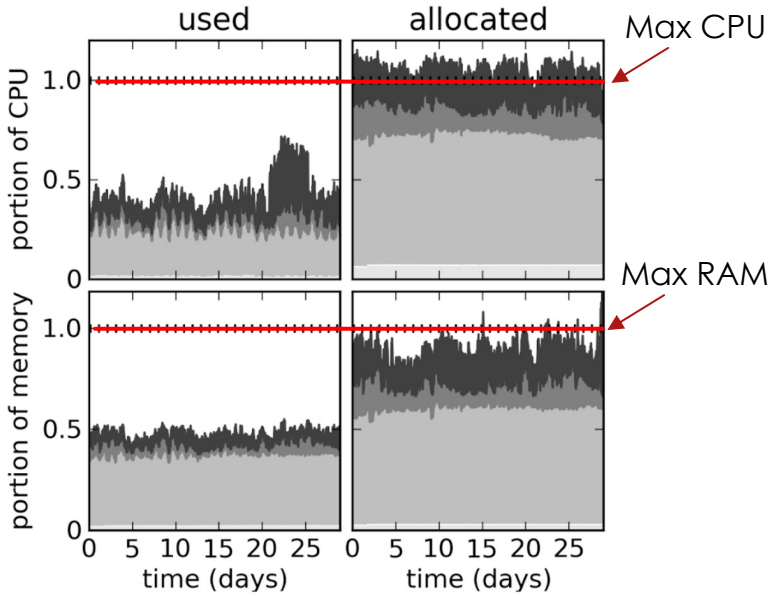
- Services show burstiness associated with patterns (night, day, week-ends, holidays)
  - VM resources allocated based on peak usage (worst case)
  - VMs do not peak at the same time… resource wasted

- Oversubscription: allocate more than the capacity
  - Bet against VMs' usage peaking at the same time

[1]*Verma, Akshat, Juhi Bagrodia, and Vimmi Jaiswal. "Virtual machine consolidation in the wild.", 2014*

Server Consolidation 101- Oversubscription (II)

Cluster-wide Resource Allocated vs. used in Google[1]:



- Expressed as a ratio virtual resource vs. physical resource
  - E.g., Nutanix typical is 1:1.35 (for each real CPU there is a 1.35 vCPUs)
- Problem: Hotspots!
  - Usage above max resource
  - Degradation of VM performances

- CPU is oversubscribed more aggressively
  - Minor performance issues (CPU easy to multiplex)
- RAM is another story…
  - Nutanix forbids RAM oversubscription
  - When RAM exhausted, data must be moved to swap (disk) x100 – x10 000 slower[2]
  - Worst case: thrashing

[1] *Reiss, Charles, et al. "Towards understanding heterogeneous clouds at scale: Google trace analysis.", 2012*

[2] *https://gist.github.com/hellerbarde/2843375*

Dynamic Consolidation

- Where migration comes into play

- Migration used to resolve Hotspots
  - VM moved away to alleviate load on a host
  - E.g., Google leverages termination/restart (preemption) & live migration (GCE)

- Also, migration to **re-**consolidate VMs
  - Re-compact VM and used on a group of hosts
    - Increase global resource utilization
  - Oversubscribe new resource
    - Send VM to a fully loaded host
    - Adapt requested allocation vs. actual resource usage

Idle VMs in IaaS Data Centers

- In 2017 Koomey reported 30% idle VMs for 5 DCs[1]
  - Idle VMs show little or no sign of activity
  - Common for long running service (mail, web server, etc.) or test VMs

- Idle VMs lock resource allocated, particularly RAM
  - RAM usually not oversubscribed
  - Waste of RAM, a crucial resource in DC
    - RAM is often the real limit to consolidation

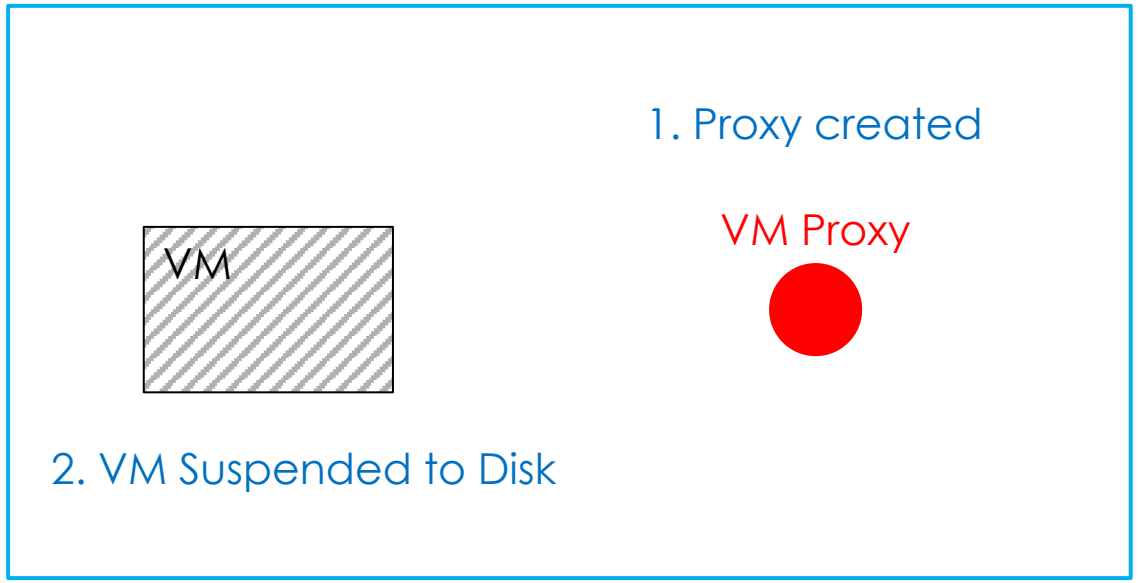
[1] *https://www.anthesisgroup.com/wp-content/uploads/2019/11/Comatose-Servers-Redux-2017.pdf, 2017*

Our Solution

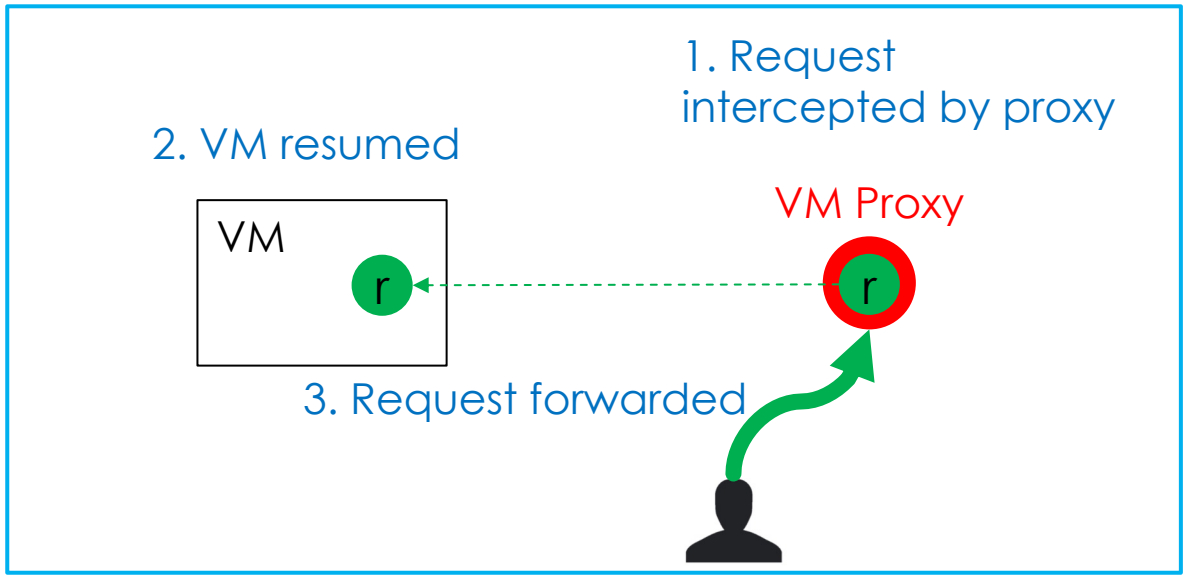
- How to reclaim memory from idle VMs:
  - Pause VM?
    - Does not release RAM, and this is the major problem…
  - VM suspend-to-disk?
    - VM state + VM RAM saved in a file to be resumed later
    - RAM is released
    - **New requests lost!** (VM not running)

- High level strategy :
  1. Suspend-to-disk an idle VM
     - Release the resource
  2. Replace the network presence suspended VM with a lightweight proxy
     - New user requests intercepted & VM restoration triggered

- When a VM is detected idle:



- When a new request arrives



- How do we implement the proxy?
- Trivial solution:
  1. Lightweight entity (e.g., a process) listening on IP addr. of VM
  2. Network packets re-routed to the proxy
  3. If packets on a TCP/UDP port, restore VM
- Problem: not general!
  - Application level keep-alive unsupported (proxy is dumb!)
  - False positives: any packets trigger the restoration

VM-to-Container Migration (I)

- How do we create a *smart* proxy?

- Inside the VM there are Processes listening on TCP/UDP ports i
  - We call them *Gateway Processes*:

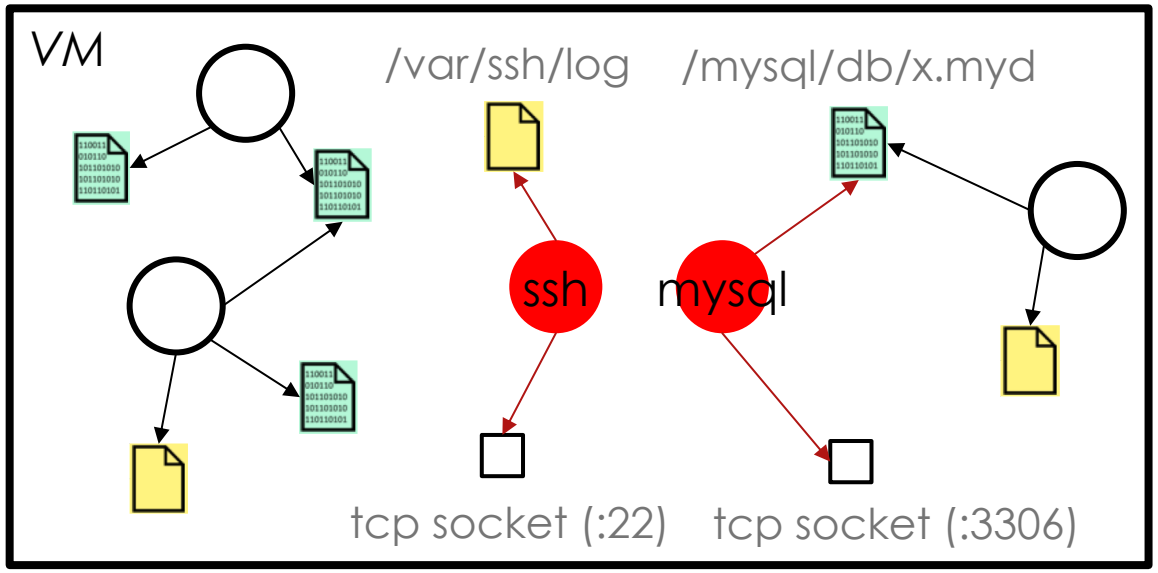
```
[alf@server ~]$ netstat -atu
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:ssh             0.0.0.0:*               LISTEN
tcp        0      0 server:ssh              192.168.1.48:59476      ESTABLISHED
tcp6       0      0 [::]:mysql              [::]:*                  LISTEN
tcp6       0      0 [::]:ssh                [::]:*                  LISTEN
udp        0      0 0.0.0.0:slingshot       0.0.0.0:*
udp        0      0 0.0.0.0:7091            0.0.0.0:*
udp        0      0 0.0.0.0:bootpc          0.0.0.0:*
udp6       0      0 [::]:25087              [::]:*
udp6       0      0 server:dhcpv6-client    [::]:*
udp6       0      0 [::]:59809              [::]:*
[alf@server ~]$
```

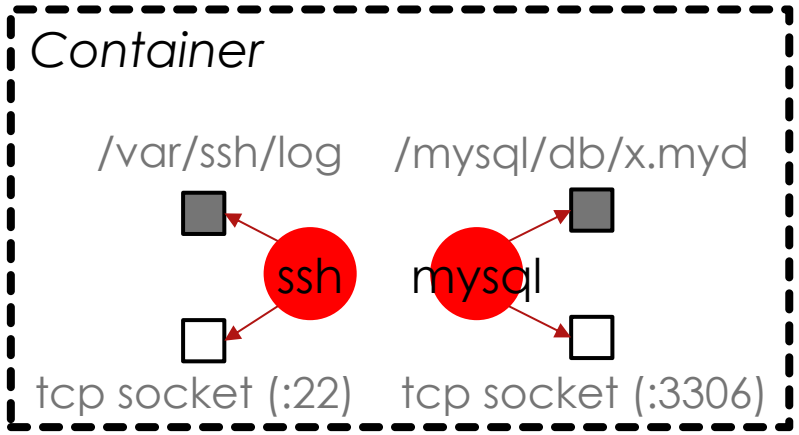
e.g., DB server with mysql + ssh server

- The gateway processes are the only entrance (gateway) from the outside to the VM

VM-to-Container Migration (II)

1. Identify the gateway processes
2. Identify opened file descriptors (files, network sockets, IPC objects, etc.)
3. Create a **container** (OS level virt.) + create environment for gateway processes
   - Network sockets preserve their state
   - Files and other IPC objects (to communicate w/ other processes) recreated as "phony files"
4. Transplants the gateway processes from VM to container





VM-to-Container Migration (III)

- Gateway Process is, well, a process
  - How do you transplant a process?

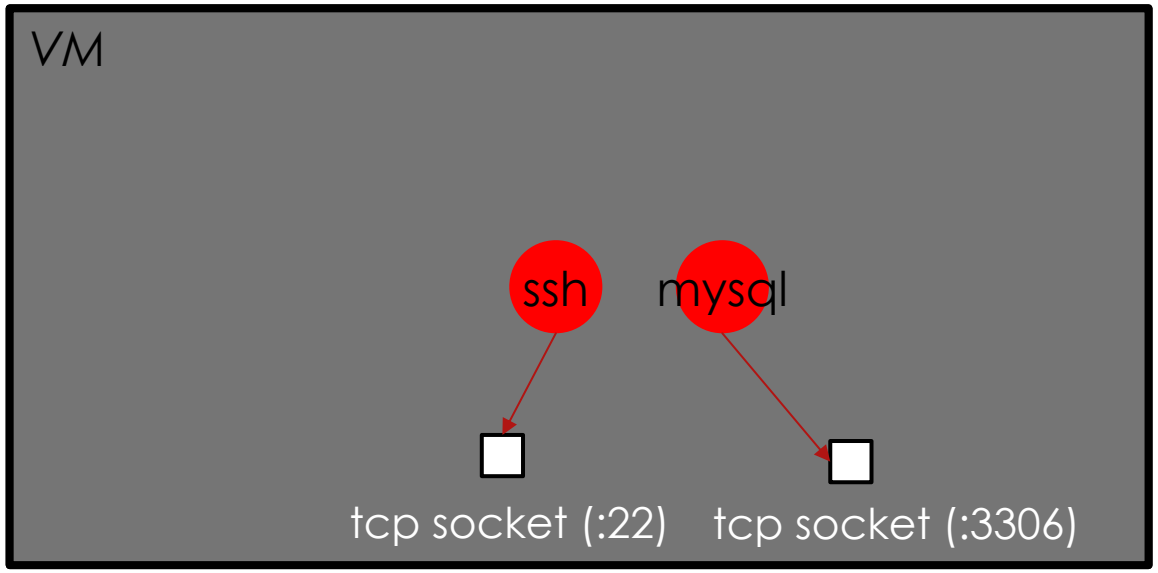
- **CRIU** (**C**heckpoint/**R**estore **I**n **U**ser-space) [1]
  - Checkpoint/restore framework utility Linux processes
  - Process state (instruction pointer, PID, memory, file descriptor state, etc.) saved to file
  - Process can resume from saved state
  - Works with all recent Linux kernels

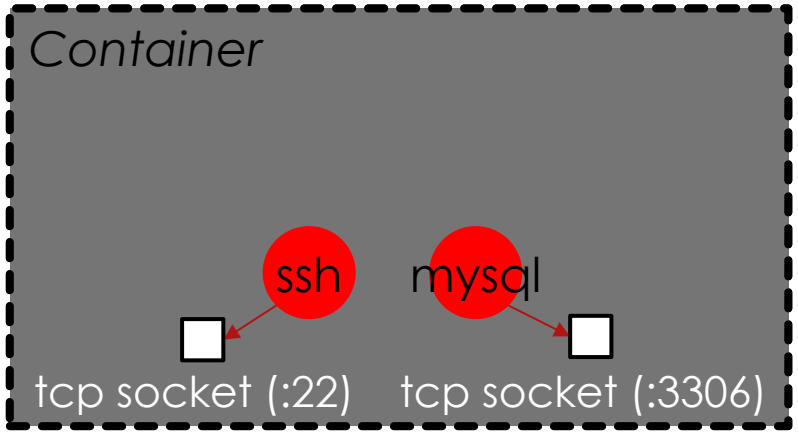
[1] *https://criu.org/*

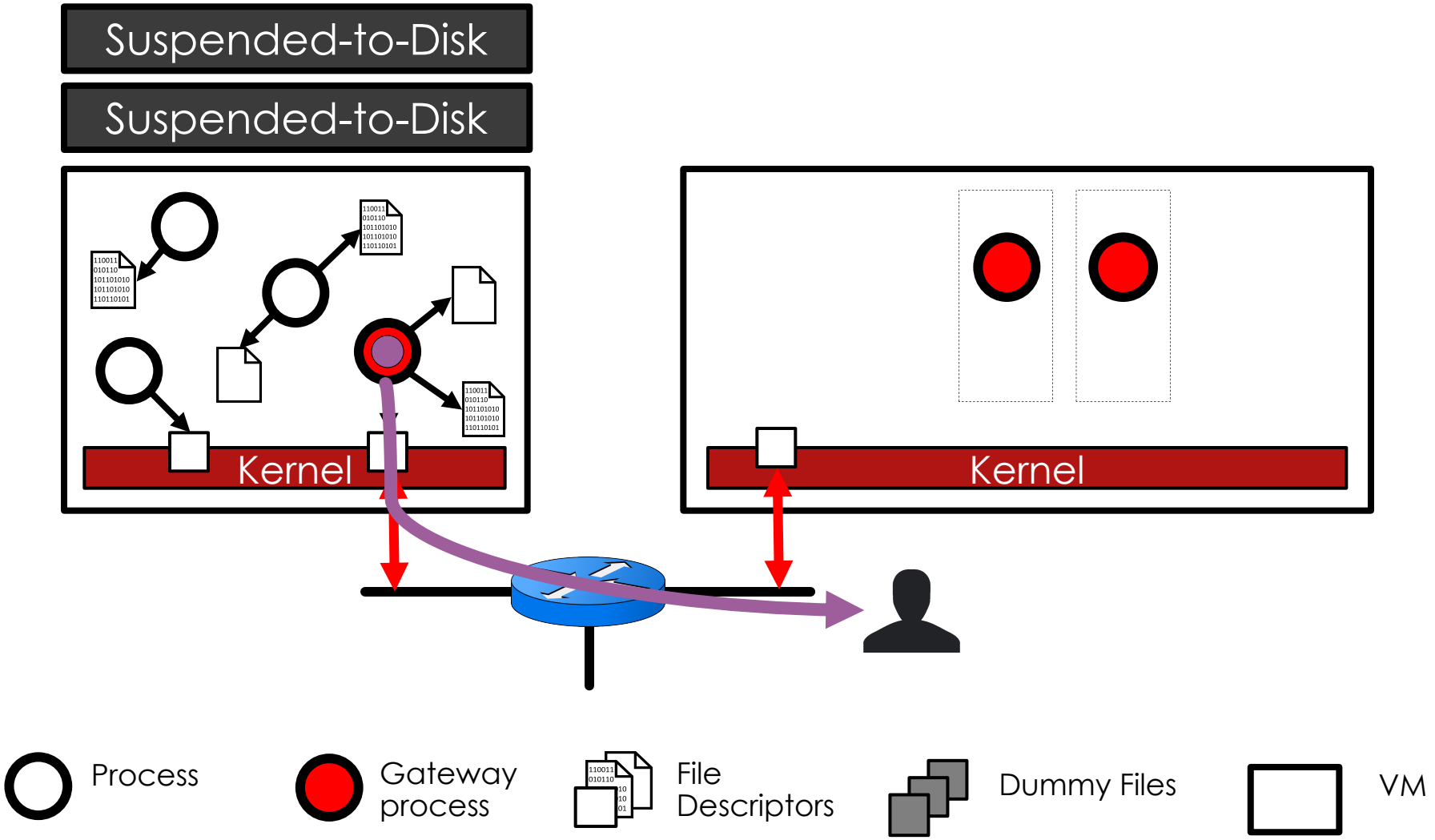
VM-to-Container Migration (IV)

- In the container, gateway process can reply to trivial requests (e.g., pings, keep-alive requests)
- In the container, gateway process cannot reply non-trivial requests (need file descriptors in VMs!)

- Fine, as long as VM is idle
  - VM and the container are equivalent
  - **We migrated a VM to a container**

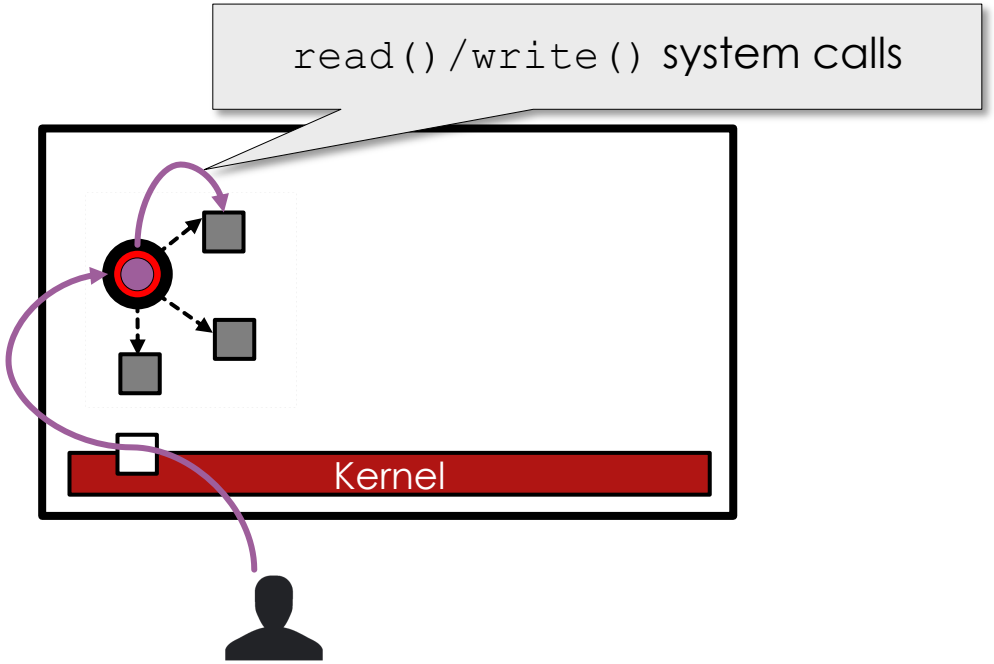




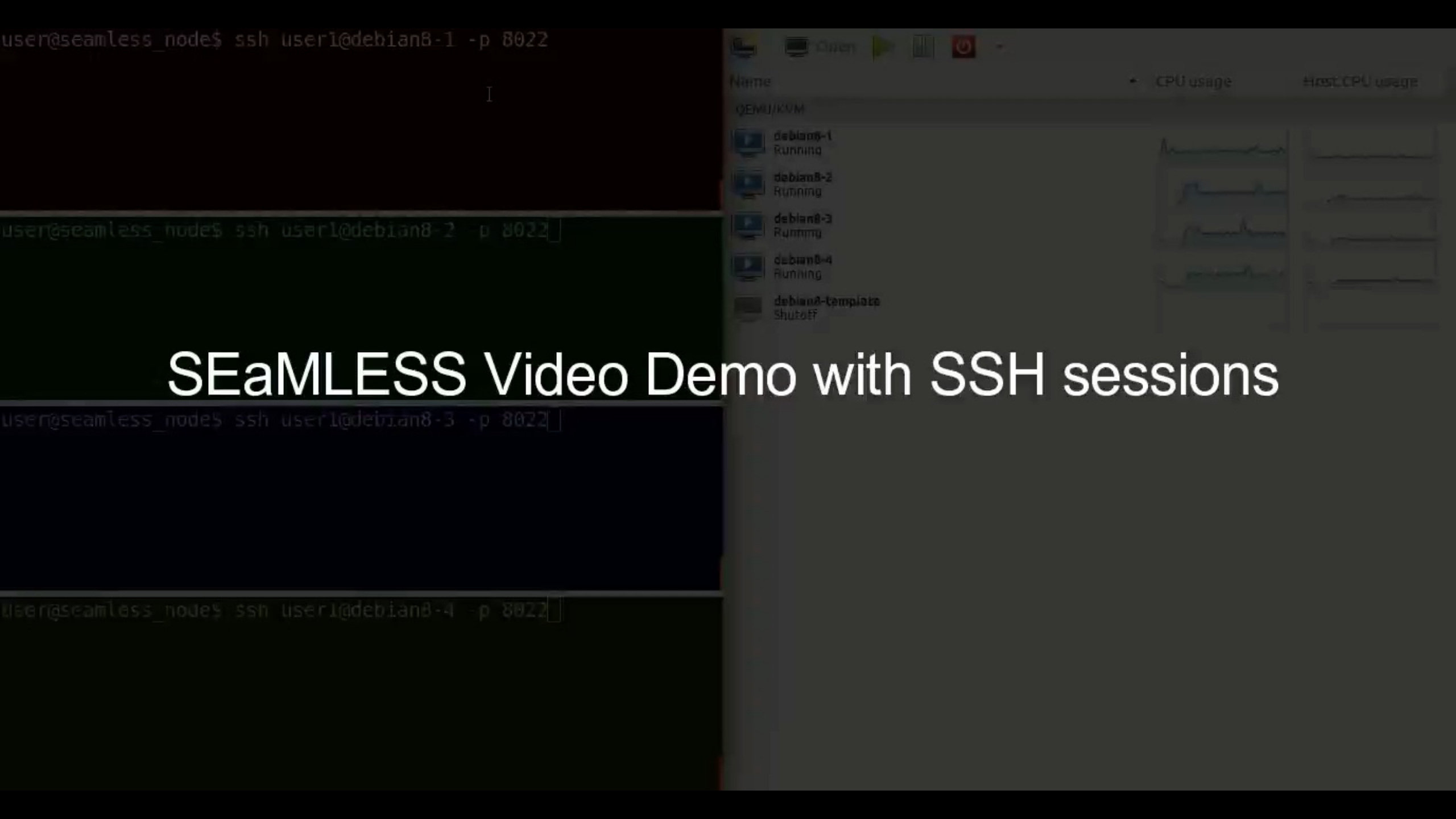

Suspended-to-Disk
Suspended-to-Disk
Kernel
Kernel
Process
Gateway process
File Descriptors
Dummy Files
VM

User-activity Detection

- What triggers the inverse migration? AKA How we detect users' requests?
- We monitor the gateway process (GW) interacts with the environment
  - If GW accesses anything not is available only inside the VMs is user activity
    - E.g., Access to "phony files"



- In Linux, processes interact with the environment through system calls
- `ptrace` is a Linux interface to intercept system calls
- If `read()/write()/open()` detected the process is stopped and reverse migration triggered

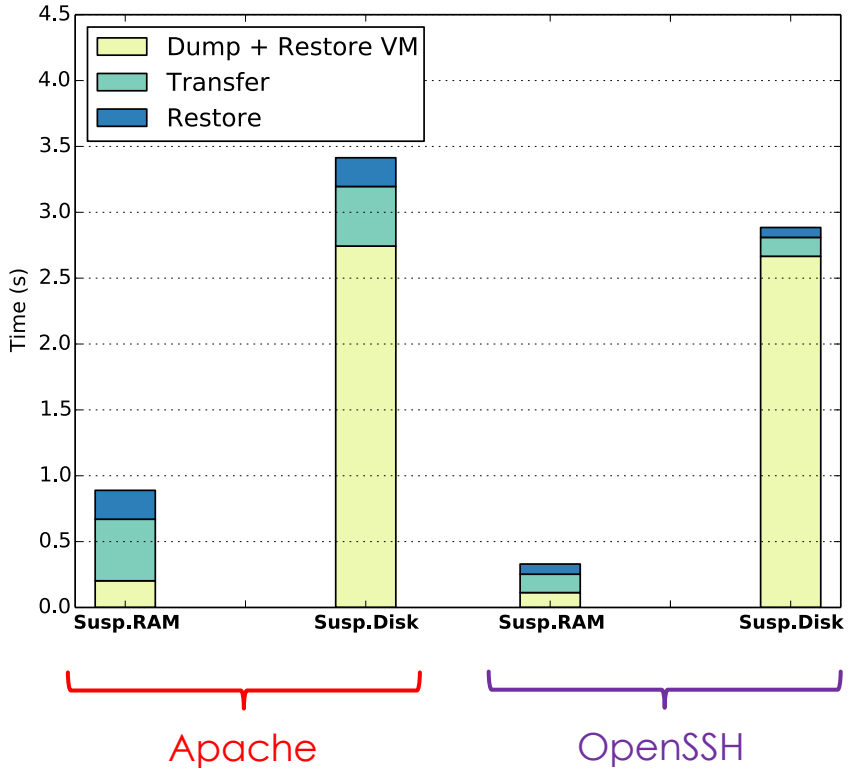

user@seamless_node$ ssh user1@debian8-1 -p 8022
user@seamless_node$ ssh user1@debian8-2 -p 8022
user@seamless_node$ ssh user1@debian8-3 -p 8022
user@seamless_node$ ssh user1@debian8-4 -p 8022
Open
Name
CPU usage
Host CPU usage
QEMU/KVM
debian8-1
Running
debian8-2
Running
debian8-3
Running
debian8-4
Running
debian8-template
Shutoff
SEaMLESS Video Demo with SSH sessions

Some Result – Migration Overhead (I)

- Upon user activity detected, how much does it take to restore the original VM?

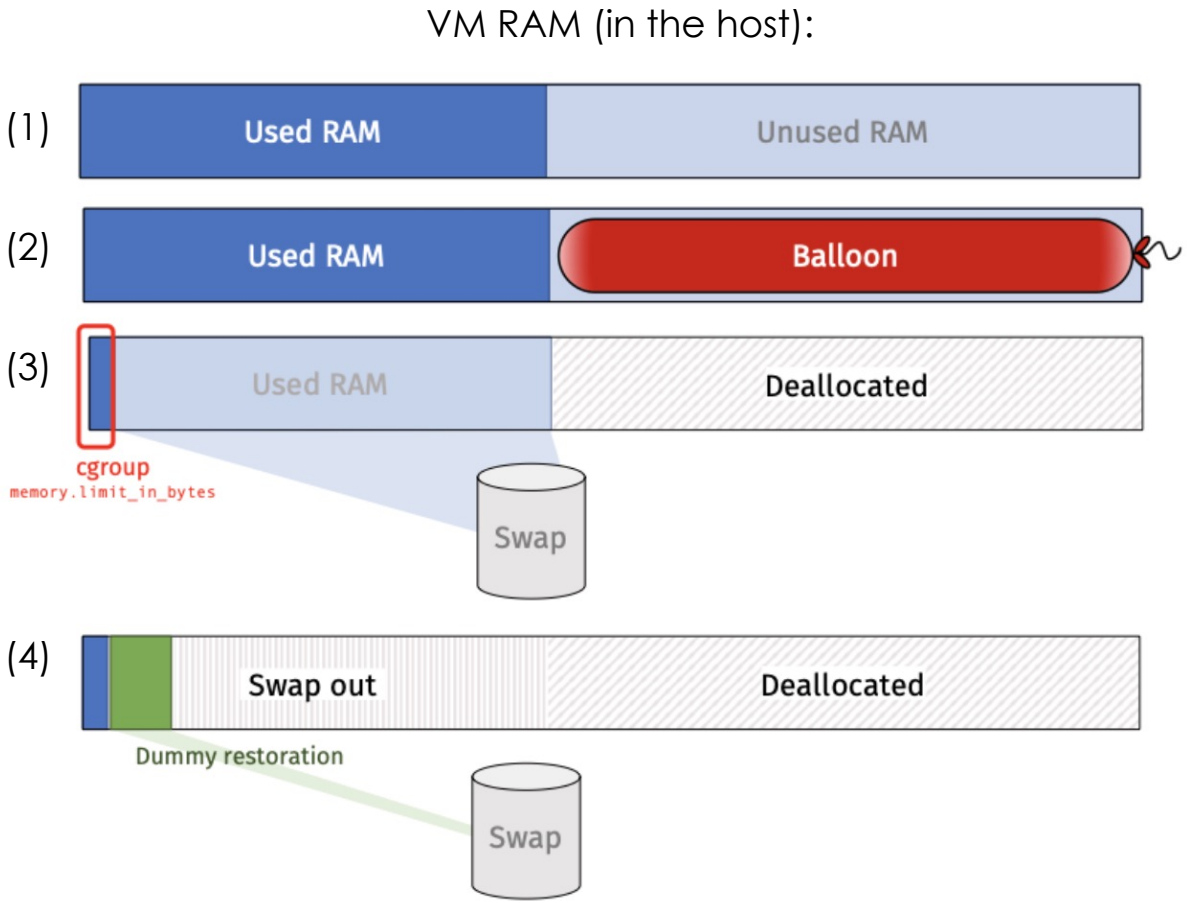
- Measure for **Apache** (web server) and **OpenSSH**
  - **Dump + Restore VM**: time to checkpoint the GW process & restore the VM
  - **Transfer**: send GW state to the VM (over network)
  - **Restore**: restore GW inside the VM

- Suspend-to-disk restoration is slow
  - Entire VM RAM loaded from disk before resuming
  - Bad *response time*



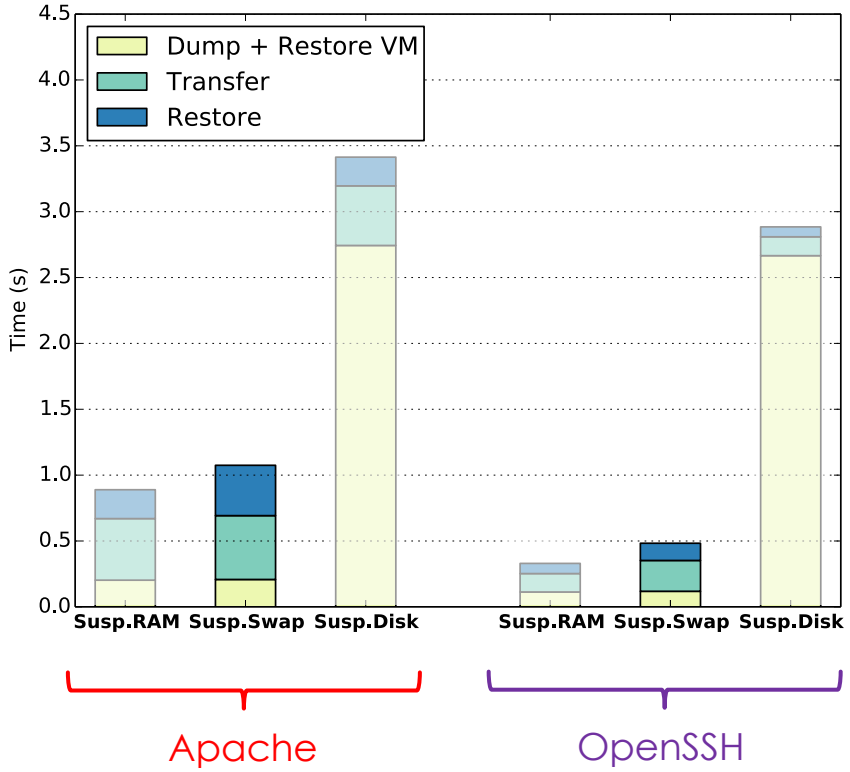
Suspend-to-Swap

- Exploit the properties of swap space
  - Lazy read: only accessed memory is swapped back into RAM
- Force the idle VM to swap-out entirely
  - Only a *hot* part of memory remains in RAM



Some Result – Migration Overhead (II)

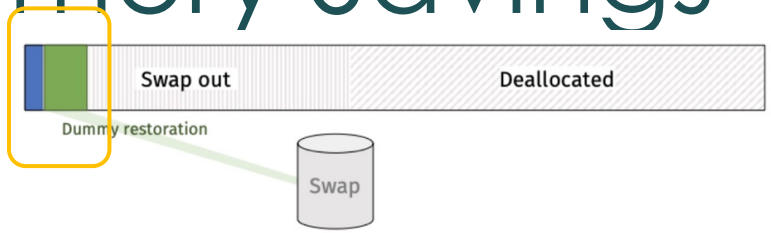
- Upon user activity detected, how much does it take to restore the original VM?
- Measure for **Apache** (web server) and **OpenSSH**
  - Dump + Restore VM: time to checkpoint the GW process & restore the VM
  - Transfer: send GW state to the VM (over network)
  - Restore: restore GW inside the VM
- Suspend-to-Swap combines the best of both worlds
  - Response time ~= Susp.RAM ☺
- What about memory?



32

Some Result – Memory Savings

Suspend-to-swap leaves some VM memory in host RAM



- Suspend-to-Swap applied to Amazon Web Services (AWS) VM sizes:

AWS Instances	Size (GB)	Reduced VM (GB)	Memory Saving (GB)
t1.micro	0.61	0.501	0.109
c1.medium	1.70	0.474	1.226
m1.small	1.70	0.474	1.226
c3.large	3.75	0.496	3.254
m1.medium	3.75	0.496	3.254
m3.medium	3.75	0.496	3.254
c1.xlarge	7.00	0.515	6.485
m1.large	7.50	0.511	6.989
m1.xlarge	15.00	0.527	14.473
m2.xlarge	17.10	0.560	16.540
m2.2xlarge	34.20	0.598	33.602

- VM reduced to ~500MB, no matter the initial size

VM-to-Container Migration - Conclusion

- Novel migration to convert idle VMs to lightweight containers
- Release resource from idle VMs
  - Suspend-to-Swap deallocates the majority of RAM
- Fully transparent & application agnostic
  - Smart proxy (container) intercepts new requests and signals the VM restart
- Fast response time
  - between 0.5 and 2 seconds

SEaMLESS *retrofits* resource utilization optimization when VMs stay idle for a long time providing a fast and transparent experience

Check out more on our article:
Segalini A., Lopez Pacheco D., Jacquemart Q., Rifai M., Urvoy-Keller G., Dione M., "Towards massive consolidation in data centers with seamless", CCGRID 2018
https://hal.archives-ouvertes.fr/hal-01877886/document

Zero-copy Local-host Migration for Host SW Upgrades

Migrations in DC Management

- What are the reasons to migrate within a DC?

(1) Resource management:

- Load balancing
  - Run out of resources (CPU, RAM, BW, etc.) on a host (Hotspot mitigation)
- Consolidation
  - Too many unused resources on a host (reduce hosts powered-on)

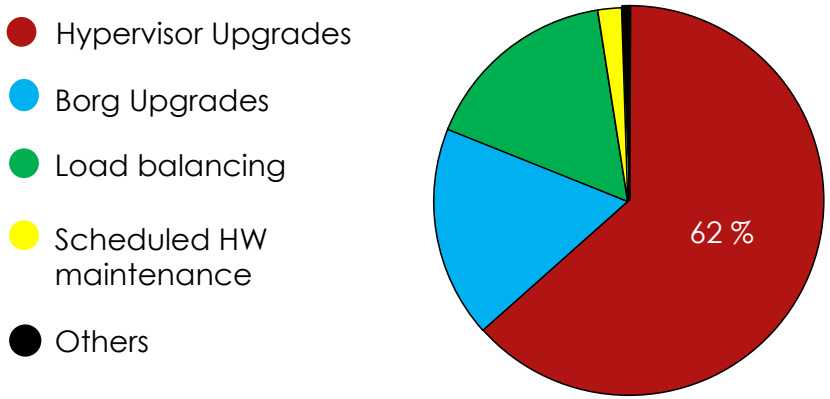
(2) Fault-recovery and Maintenance:

- Incoming disaster (e.g., Earthquake warnings in Japan, floodings)
- DC power infrastructure maintenance
- Host HW Failure
  - Bad DIMMs, bad disks, etc.
- HW upgrades
  - Install new RAM, CPUs, deploy new net fabric
- Host SW upgrades
  - Platform SW upgrades
  - Hypervisor upgrades

Migrations for Fault Recovery & Maintenance

- Many reasons to migrate
  - Incoming disaster, HW upgrades, host SW upgrades

Causes for migration in Google Cloud Engine (GCE) (January-March 2017)[1]



- In Google:
  - < 6% HW maintenance + others
  - ~12.5% load balancing
  - ~12.5% Borg upgrades (Google proprietary cluster manager)
  - ~62% are hypervisor upgrade

[1]*Ruprecht, Adam, et al. "VM live migration at scale", 2018*

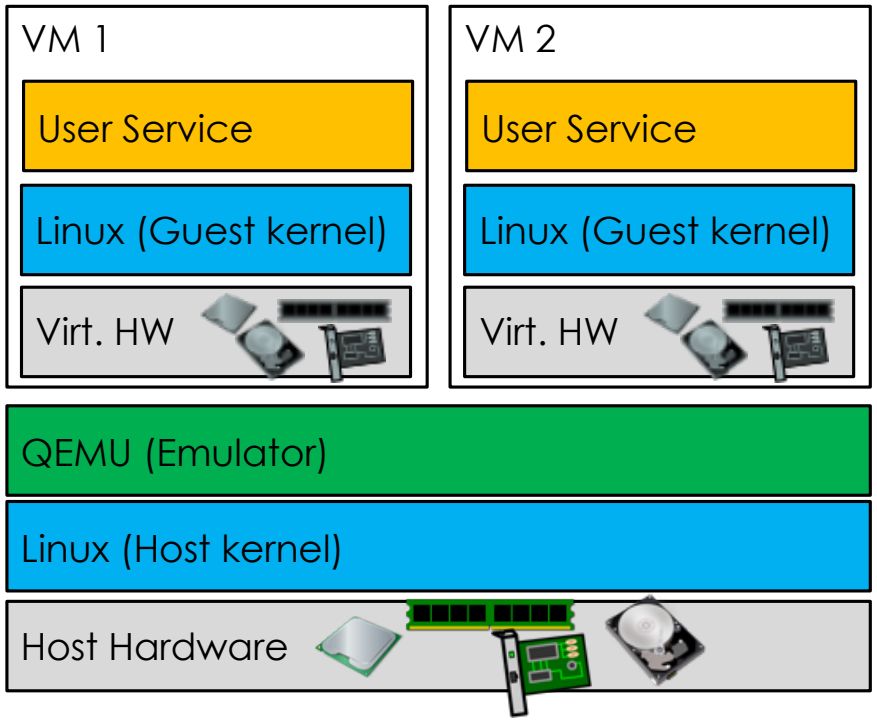
Hypervisor Upgrades (I)

- What upgrading a hypervisor means?
- Depends on the type (type-1, typ-2), however, there are always *at least 2 components*:
  - Host Kernel
    - Manages the host hardware (CPU scheduling, I/O access, etc.)
    - E.g., Linux for KVM-hypervisor, Xen (Kernel)
  - Virtual Hardware Emulator
    - Emulates VMs' virtual hardware on host kernel
    - E.g., QEMU, VirtualBox for KVM-hypervisors
  - N.B., sometimes emulation is done in the kernel (Xen)

- *Upgrading a hypervisor = replace host kernel + replace emulator*

Hypervisor Upgrades (II)

- Let's consider the case for a KVM-hypervisor where host kernel and emulator are well defined



- Upgrading a KVM-hypervisor:
  - Replace QEMU (emulator)
  - Replace Linux (host kernel)
- It requires a host reboot!
  - Disruptive operation, VMs are lost!

Related Work (I)

- How to avoid the disruption from the host reboot?

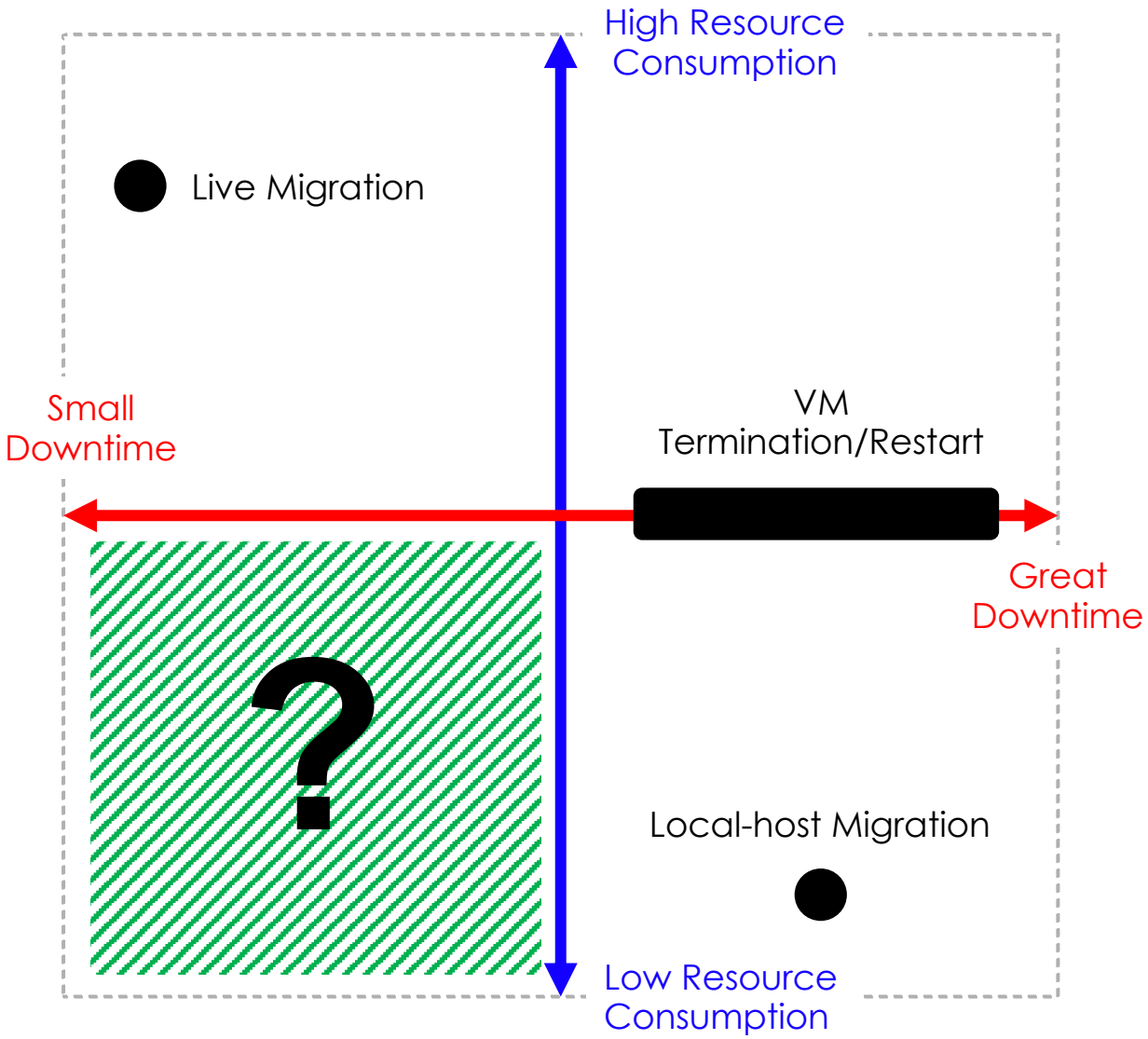
1. Live kernel upgrade
   - *E.g., Orthus (KVM-hypervisor), kpatch, ksplice, kGraft (Linux), etc.*
   - Not general, only address small fixes

2. Live migration + Host reboot
   - *E.g., Google Cloud Engine, Microsoft Azure*
   - First live migrate VMs away, replace kernel+emulator, then reboot

3. VM termination/restart + Host reboot
   - *E.g., Amazon AWS, Google Preemptible instances*
   - Terminate VMs, restart VMs away, replace SW + reboot

4. Local-host migration + Host reboot
   - Suspend-to-disk VMs, replace SW + reboot, then restore VMs

Related Work (II)

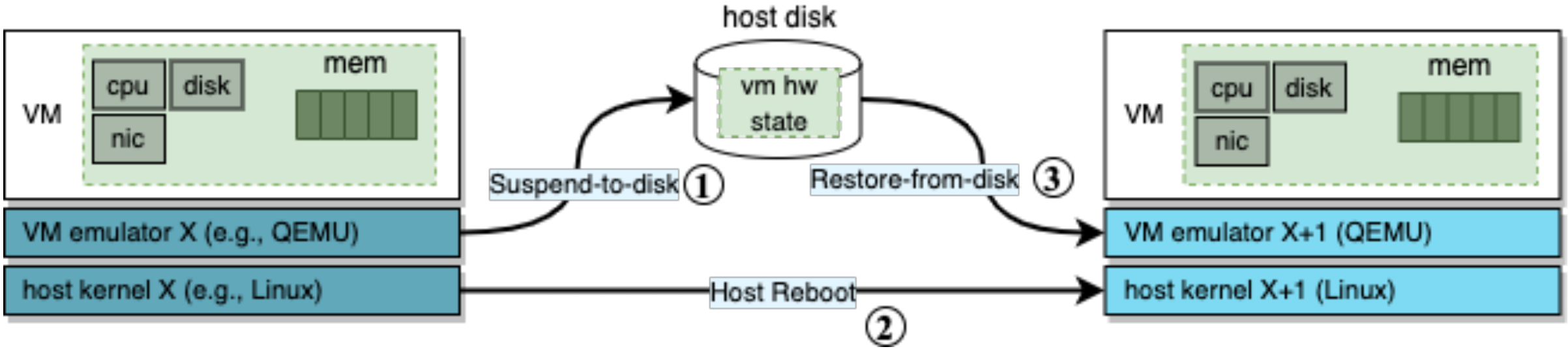


- Downtime = time VMs unavailable
- Resource cons. = additional CPU, RAM, BW in the cluster

- Live Migration
  - Small downtime (50-200ms)
  - Consume resource (BW + CPU-RAM)
- VM Termination/restart
  - High downtime (notably for stateful service)
  - Non-transparent
- Local-host migration
  - High downtime:
    Host reboot (1-2 mins) + VM resume-from-disk

Can we find a compromise?

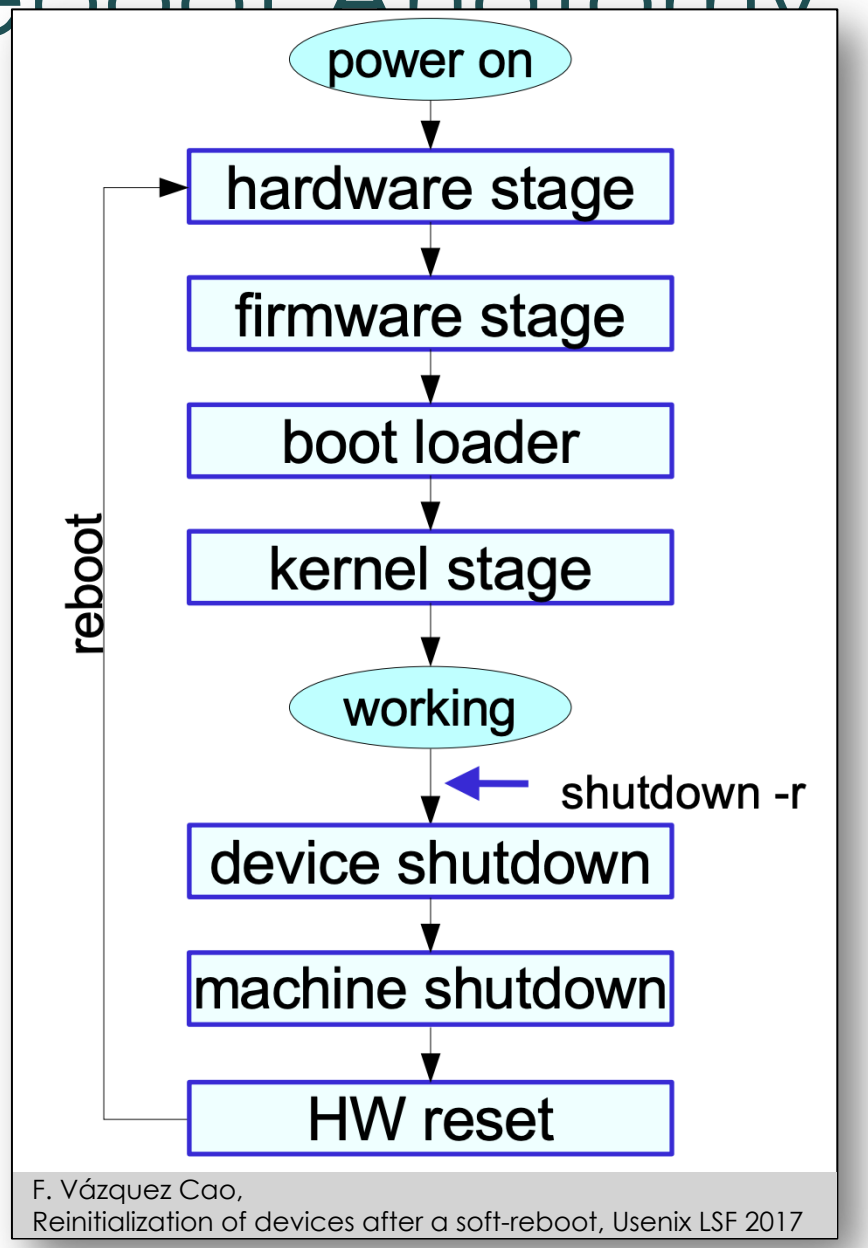
Local-host Migration



- The problem with this solution:
  - VM state, including RAM, must be serialized/deserialized to/from disk
    - w/ SSD on a full 256GB host = 17 minutes
  - Also *host reboot* is inefficient
    - On enterprise machines = 1:30-2:20 minutes

- *We can drastically optimize (1) and (3) thanks to a smart approach on (2)*

Reboot Anatomy

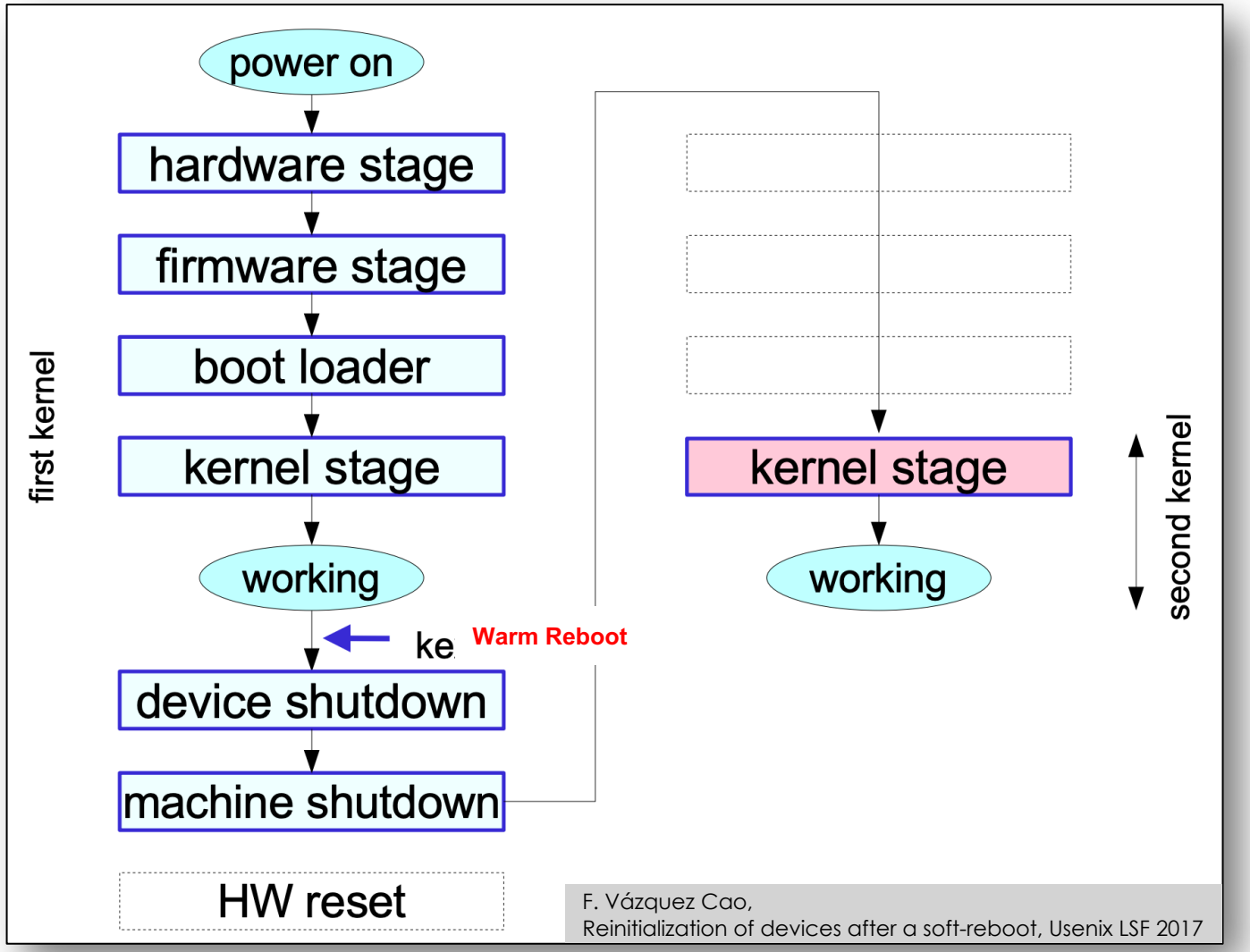


F. Vázquez Cao,
Reinitialization of devices after a soft-reboot, Usenix LSF 2017

- A classical host reboot involves:
  - Hardware reset: CPU, devices and DRAM is cleared
  - Firmware initializes hardware + loads bootloader
  - Bootloader load kernel image in RAM + `jmp` to kernel entry point
  - Kernel initializes (CPU sched, mem management, etc.) until user-space ready
- The objective is upgrade the hypervisor: *replace kernel and emulator*
  - Only kernel stage is needed
  - Also skipping hardware reset = DRAM not reset

Warm Reboots

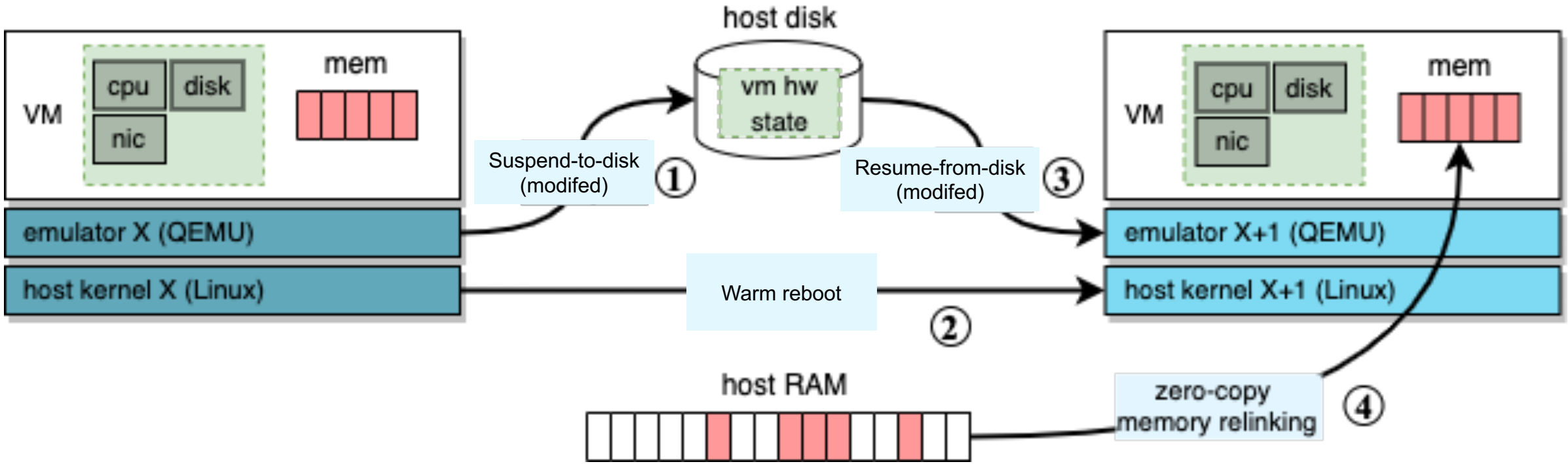
- Warm reboot is the solution!



- Linux implementation named `kexec`
- Current kernel acts as a bootloader
  - Places in RAM new kernel image
  - `jmp` to new kernel's entry point
- Two fundamental properties:
  1. 6x faster than traditional reboot
  2. DRAM not reset

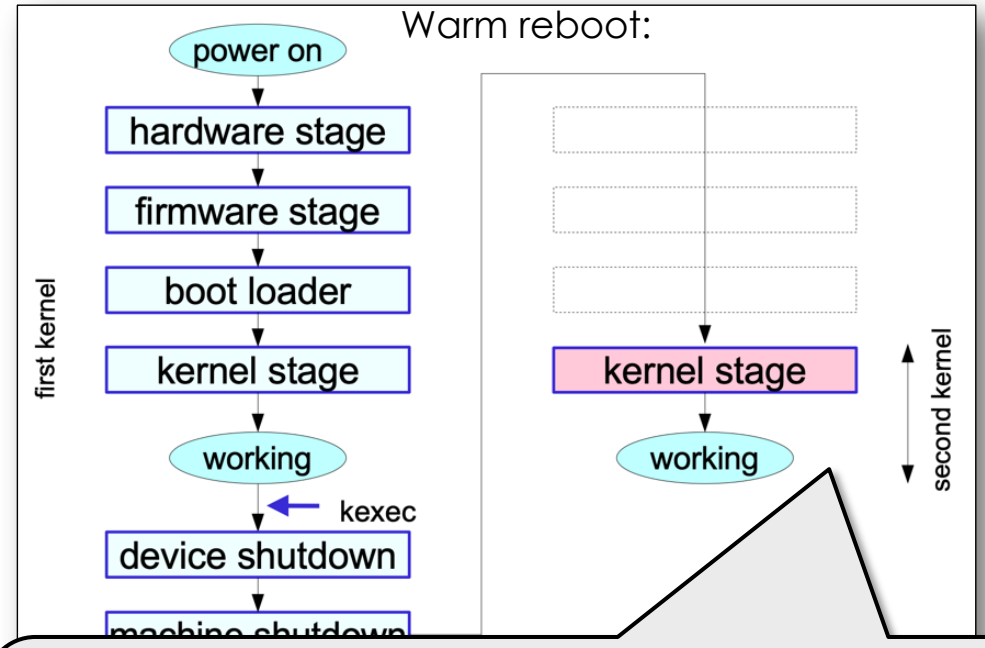
Our solution: Zero-copy Local-host Migration

- Hypervisor kernel + emulator replaced w/ warm reboot
- VM state backed-up to disk
  - VM RAM stays in-place, no need to serialize/deserialize (*zero-copy*)
  - Suspend/restore of VM fast and in constant time w/ respect to RAM

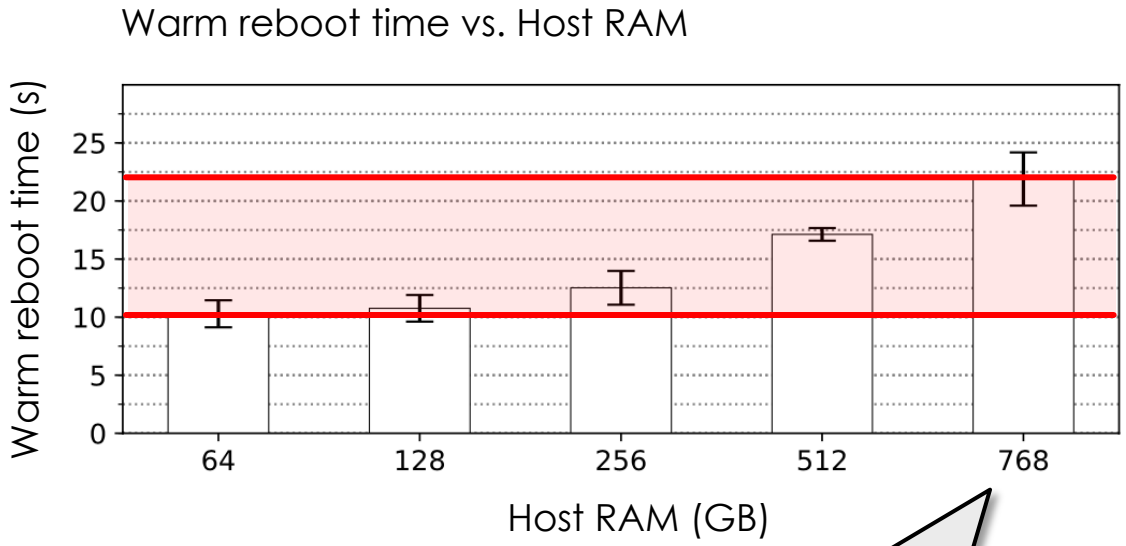


Lazy Memory Initialization (I)

- Two problems remain



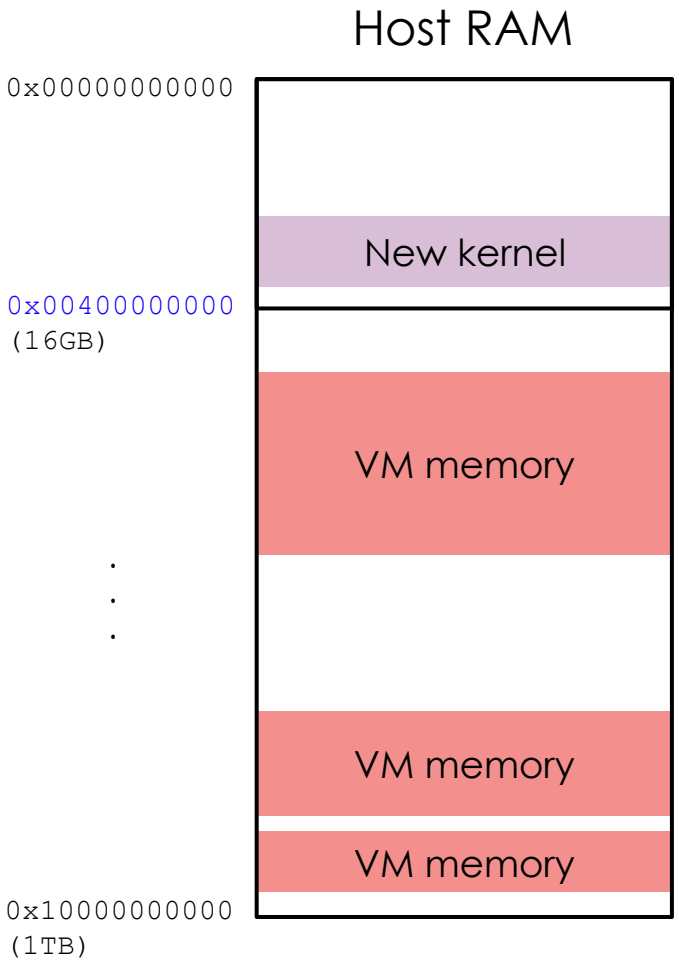
- What prevents the new kernel from corrupting VM RAM?
  - Do we have to modify the kernel memory allocation?



- More RAM = longer warm reboot
  - 12 secs more for 768GB host!
- Kernel memory initialization is slow and RAM dependent!

Lazy Memory Initialization (II)

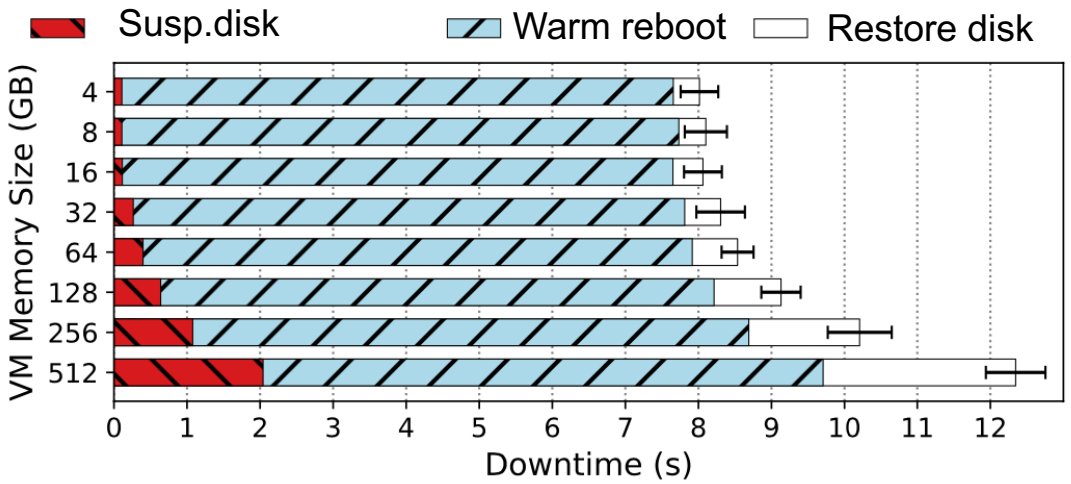
- Solution: lazy memory initialization

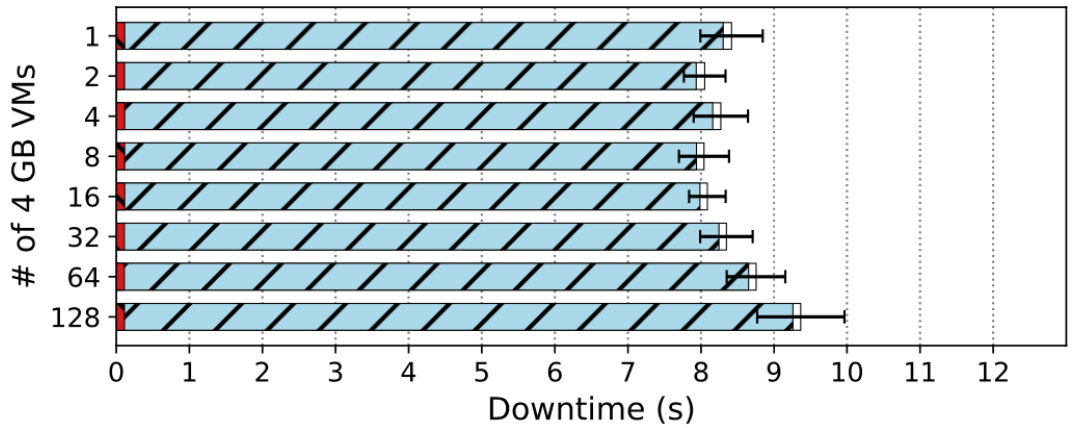


- Eventually whole RAM is initialized
  - VM accesses + background thread
- VM memory gets preserved in host RAM

Some Results – Host Upgrade Time

- Zero-copy local-host migration duration (breakdown):



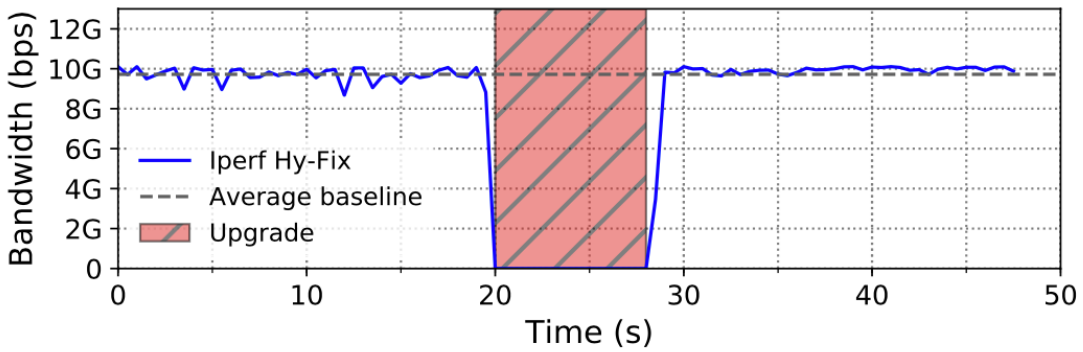


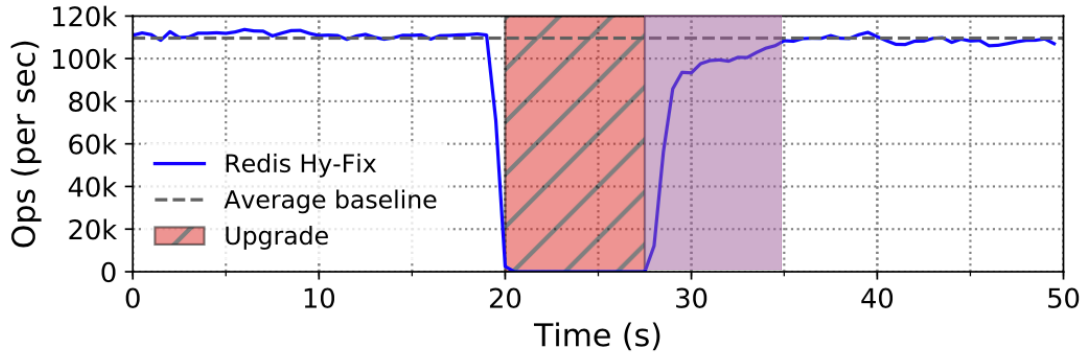
- Traditional local-host migration takes 17 minutes for a 256GB hypervisor

- Zero-copy local-host migration takes 9.5-12.5 seconds for a 512GB
  - Hardly grows with host size

Some Results – Upgrade in Action

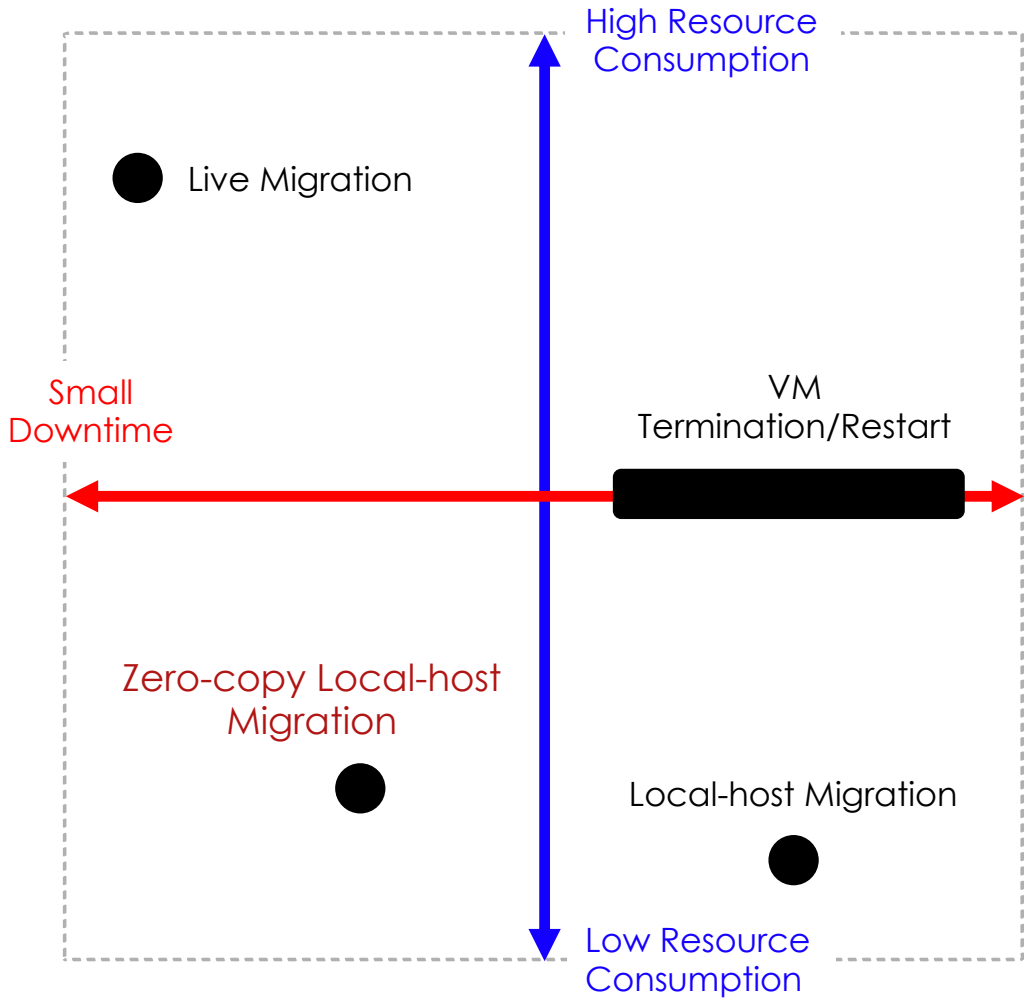
- Hosted app. Performance during migration:
  - iperf (network benchmark)
  - Redis (in-memory database)





- Lazy memory initializing affects memory latency, thus Redis throughput
  - Limited impact (18% less)
  - Disappears in a matter of seconds

Zero-copy Local-host Migration - Conclusions



- With zero-copy local-host migration:
  - Host SW upgrade completes in (for big TB machines)
  - No external resource needed
- DC administrator can upgrade any number of hosts at once
  - Impossible w/ live migration (takes tens of days[1])
- Downtime 100x worst than live migration
  - Critical VMs will still use live migration
  - Reduced the global number of live migrations

The Role of Migrations in Data Center Management

FINAL REMARKS

Final Conclusions

- Migrations are fundamental in DCs
  - Resource management
  - Fault-recovery & maintenance
- New flavors of migrations can target precise scenarios
  - Idle VMs: *VM-to-containers*
  - Host SW maintenance: *Zero-copy local-host migration*
- New technologies are emerging in DCs
  - NonVolatile Memory (NVM)
  - Ever faster networks (100Gb/s NICs, SmartNICs, DPDK, etc.)
  - Etc.
- How migrations will use these technologies?

Thanks for your attention!

QUESTIONS?