COS 318: Operating Systems

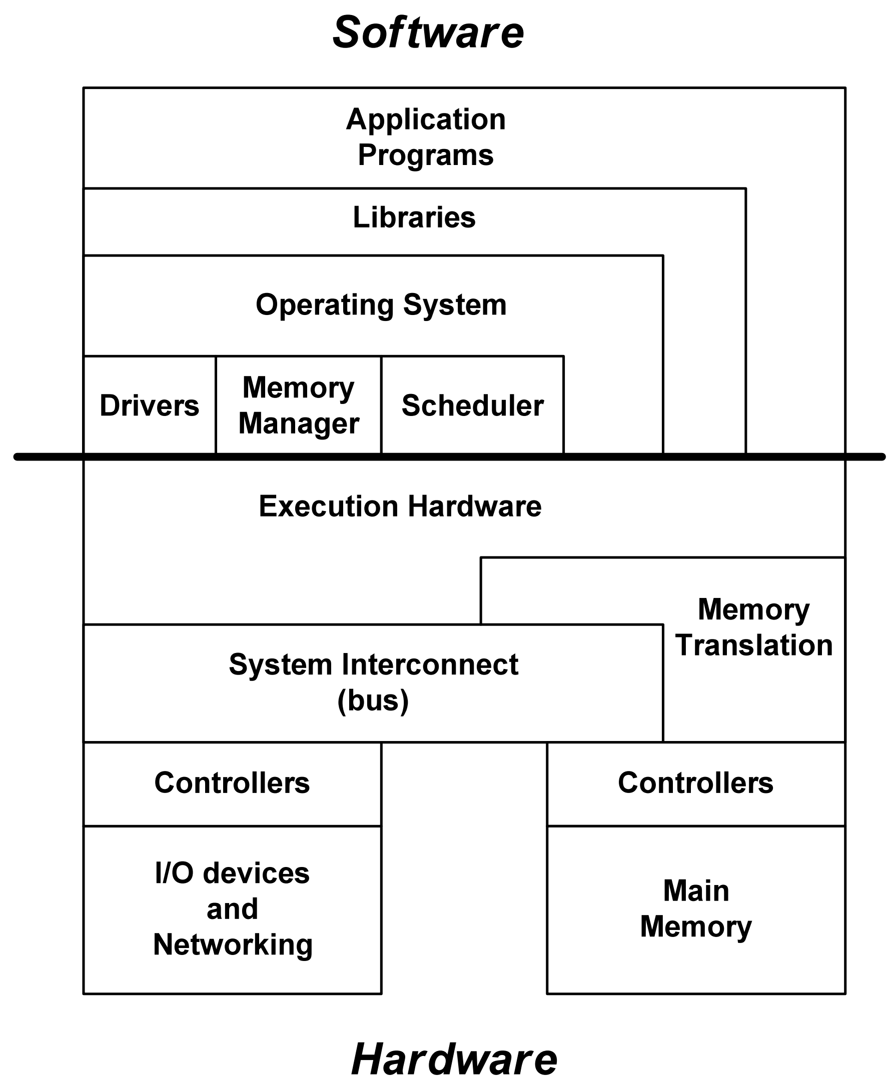
Virtual Machine Monitors

Prof. Margaret Martonosi
Computer Science Department
Princeton University

Abstraction

- Computer systems are built on levels of abstraction

- **Higher level of abstraction hide details at lower levels**

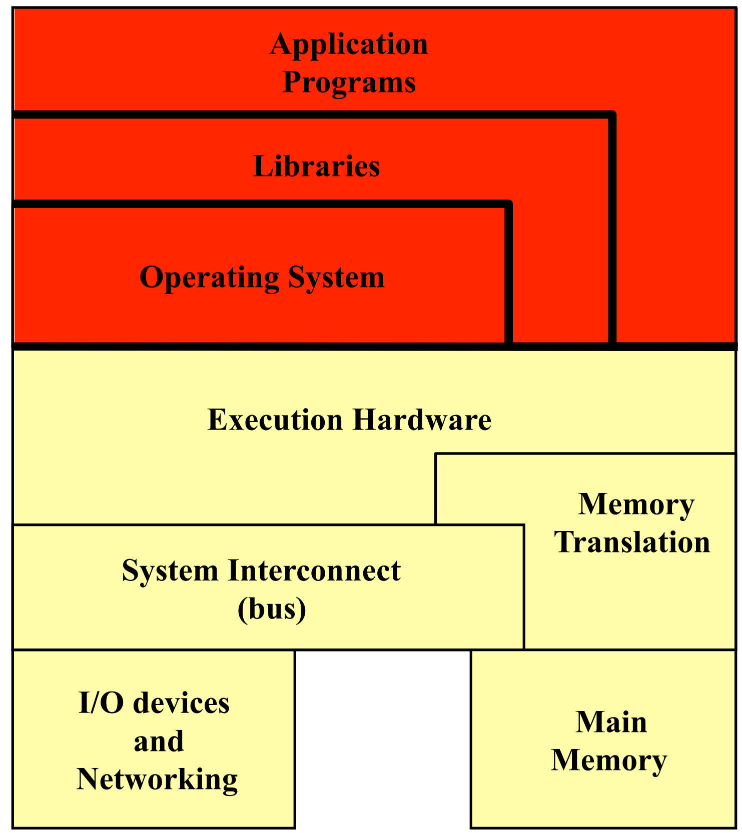


The "Machine"

- Different perspectives on what the *Machine* is:
- OS developer

Instruction Set Architecture

- ISA
- Major division between hardware and software

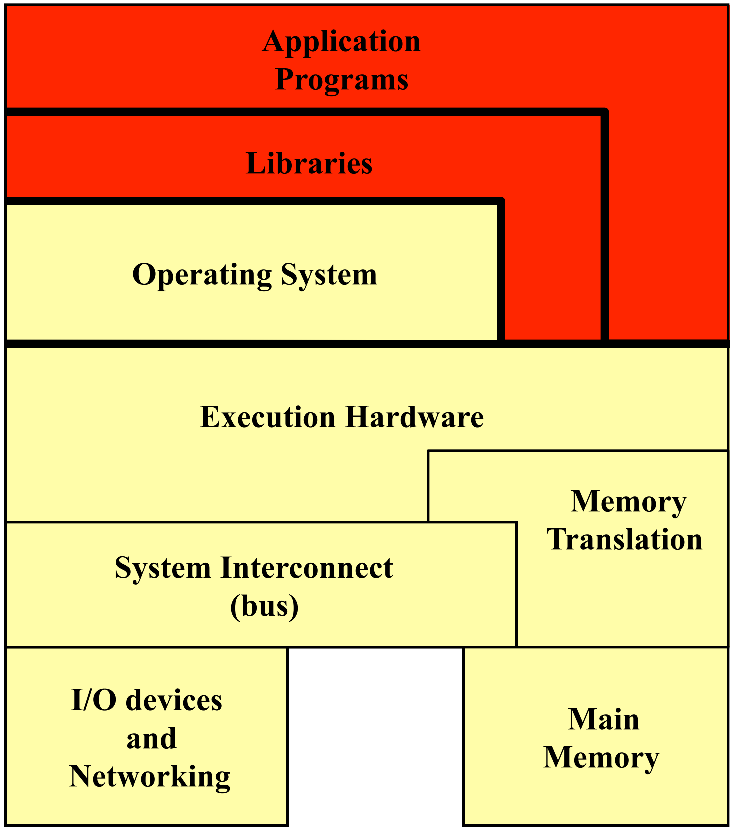


The "Machine"

- Different perspectives on what the *Machine* is:
- Compiler developer

Application Binary Interface

- ABI
- User ISA + OS calls

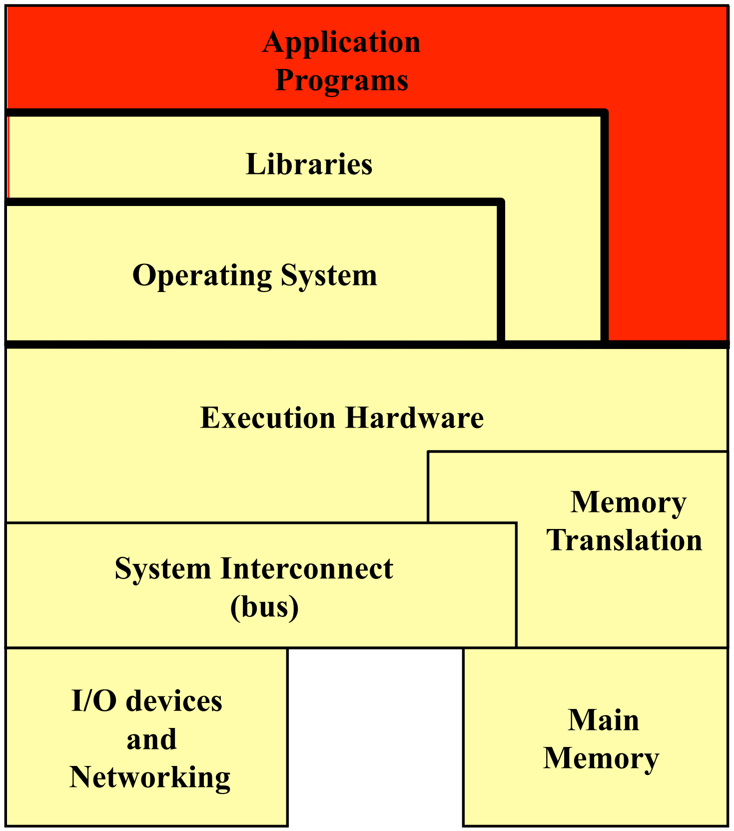


The "Machine"

- Different perspectives on what the *Machine* is:
- Application programmer

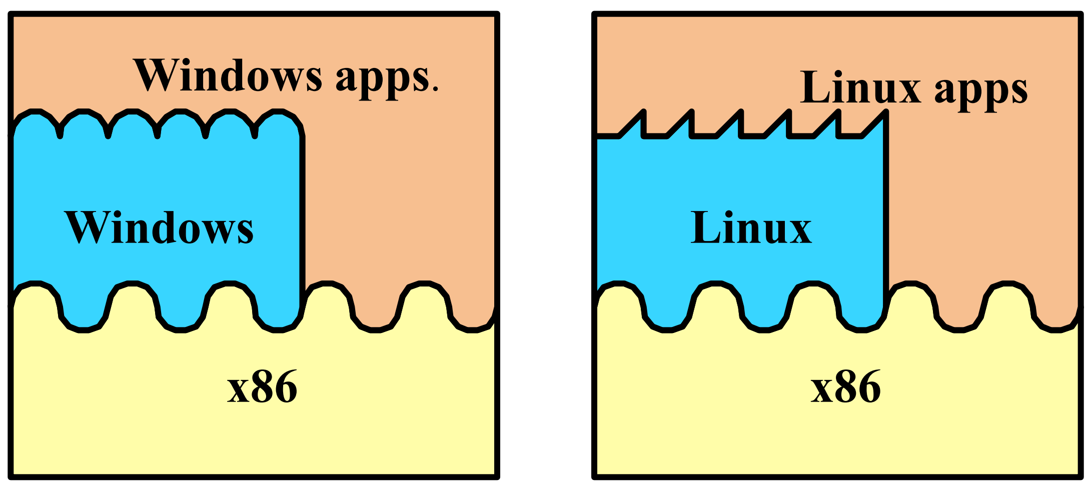
Application Program Interface

- API
- User ISA + library calls



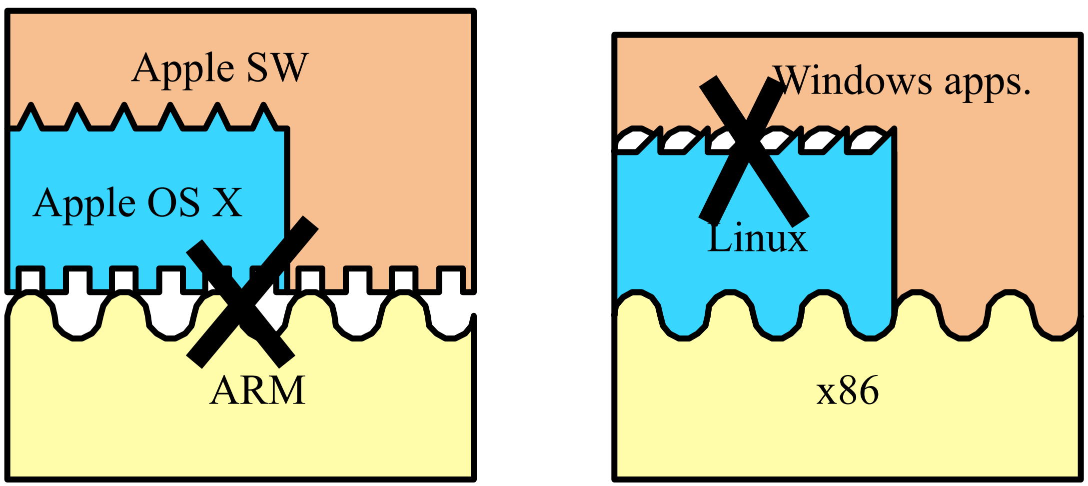
Advantages of Abstraction & Standard Interfaces

- Major design tasks are decoupled
  - In space and time
- Different hardware and software development schedules
- Software can run on any machine supporting a compatible interface



But, where are we now? …

- Software compiled for one ISA will not run on hardware with a different ISA
  - ARM vs x86?
- Even if ISAs are the same, OSes may differ
  - Windows 8 vs. Linux?
- Binary may not be optimized for the specific hardware platform it runs on
  - Intel Pentium 4 binaries on an AMD Athlon?



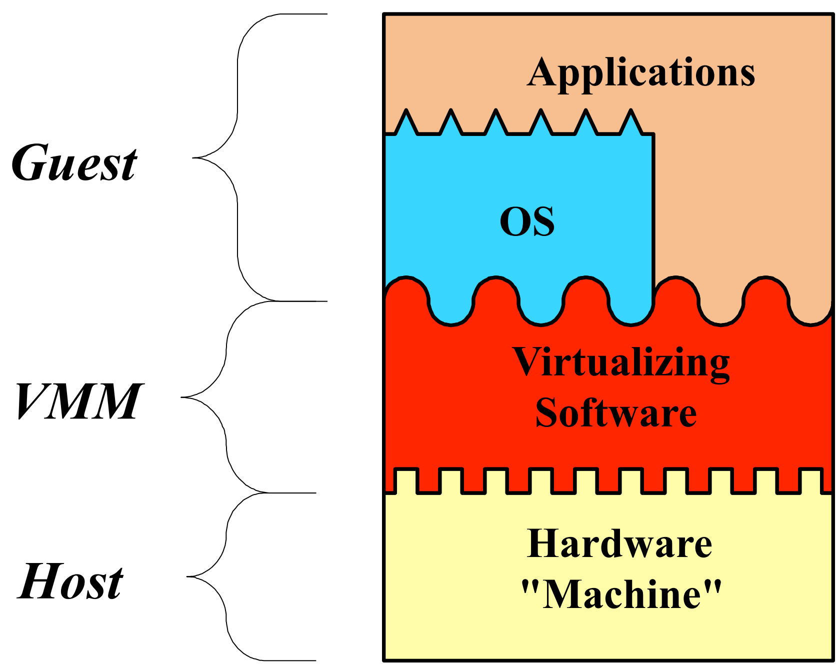
Hardware Resources

- Conventional system software manages hardware resources directly
  - An OS manages the physical memory of a specific size
  - I/O devices are managed as physical entities
- Difficult to share resources except through OS
  - All users of hardware must use the same OS
  - All users are vulnerable to attack from other users sharing the resource (via security holes in OS)

Virtual Machines

add *Virtualizing Software* to a *Host* platform
and support *Guest* process or system on a *Virtual Machine* (VM)

Example: System Virtual Machine

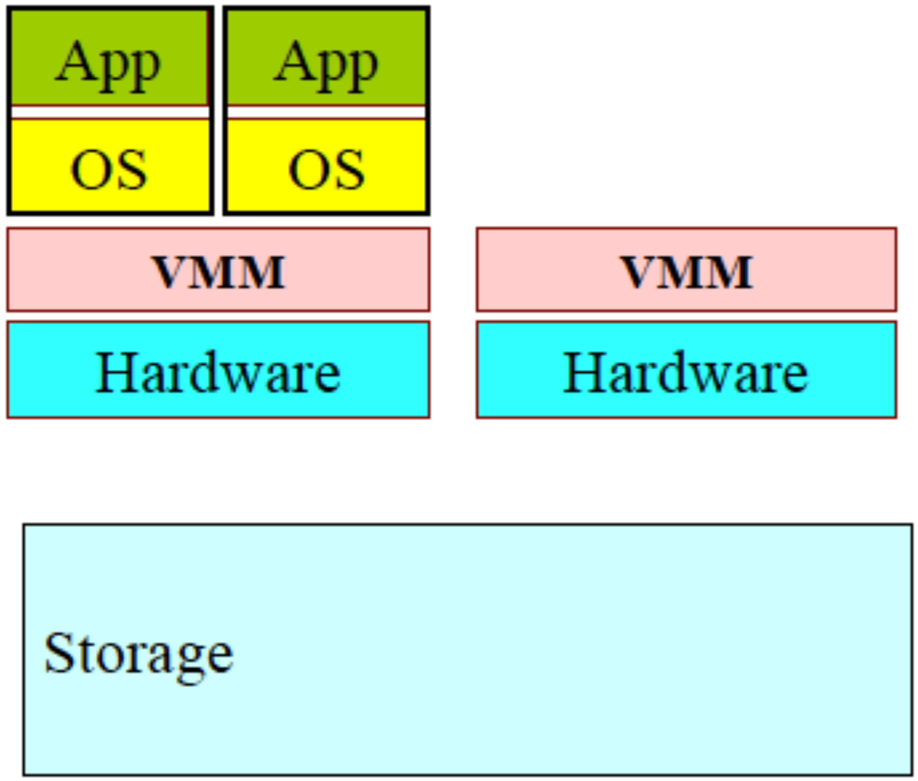


Goal: Guest OS & Apps unaware of Virtualization underneath them.

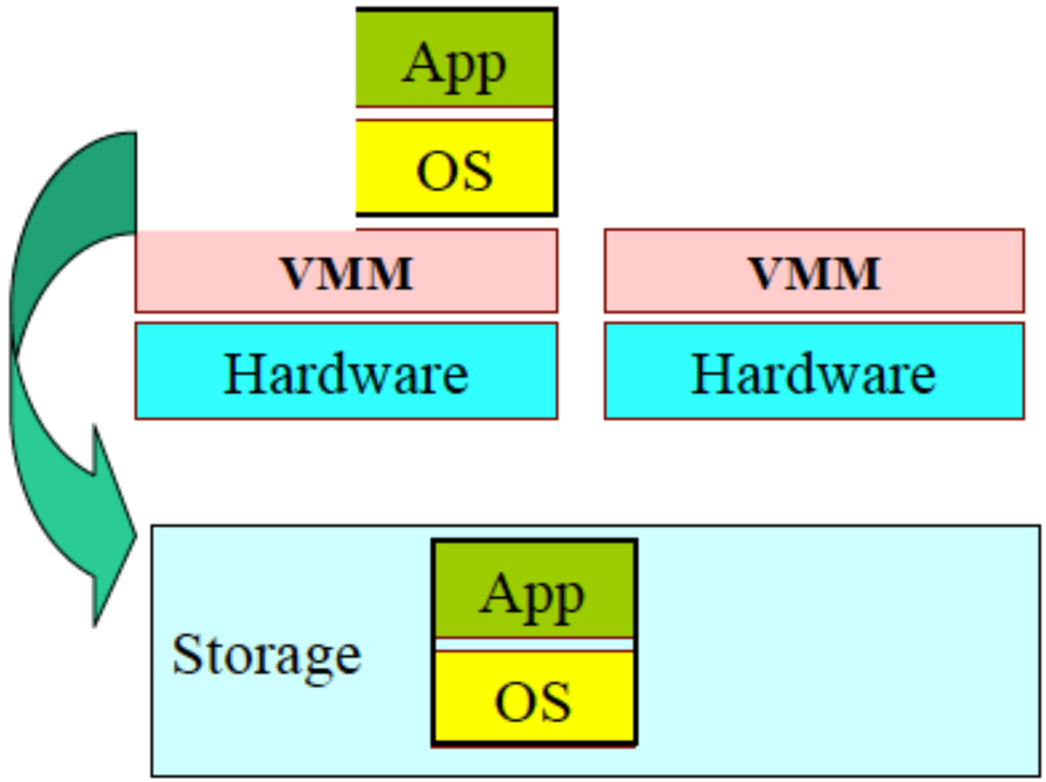
Virtual Machines: Introduction

- Have been around since 1960's on mainframes
  - used for multitasking
  - Good example – VM/370
- Have resurfaced on commodity platforms
  - Server Consolidation
  - Web Hosting centers
  - High-Performance Compute Clusters
  - Managed desktop / thin-client
  - Software development / kernel hacking

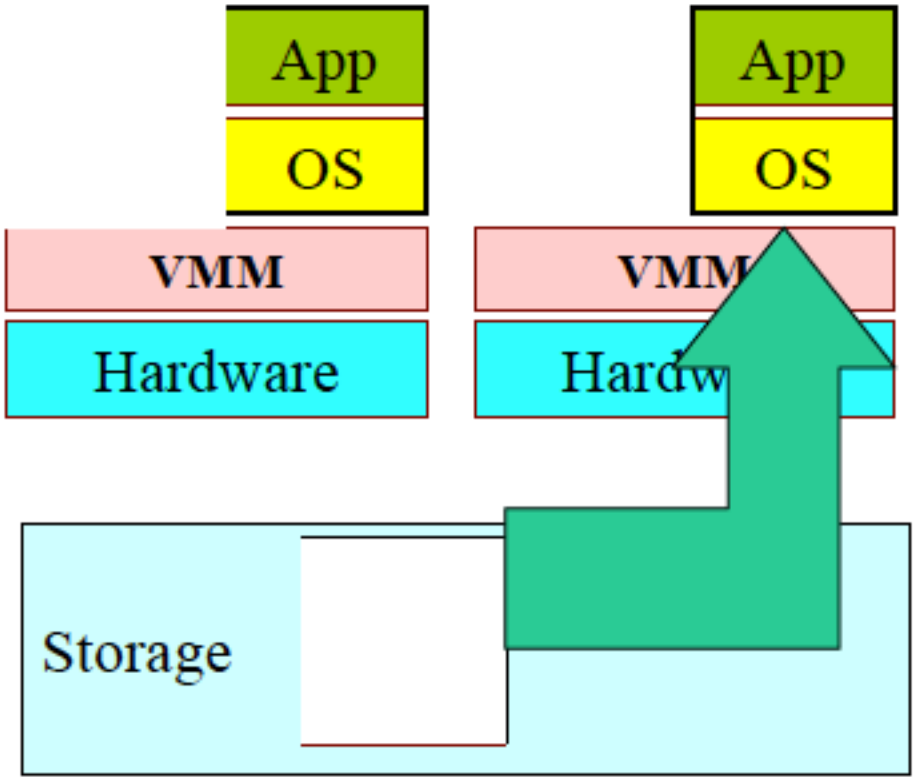
VMM Functions: Multiplex VMs



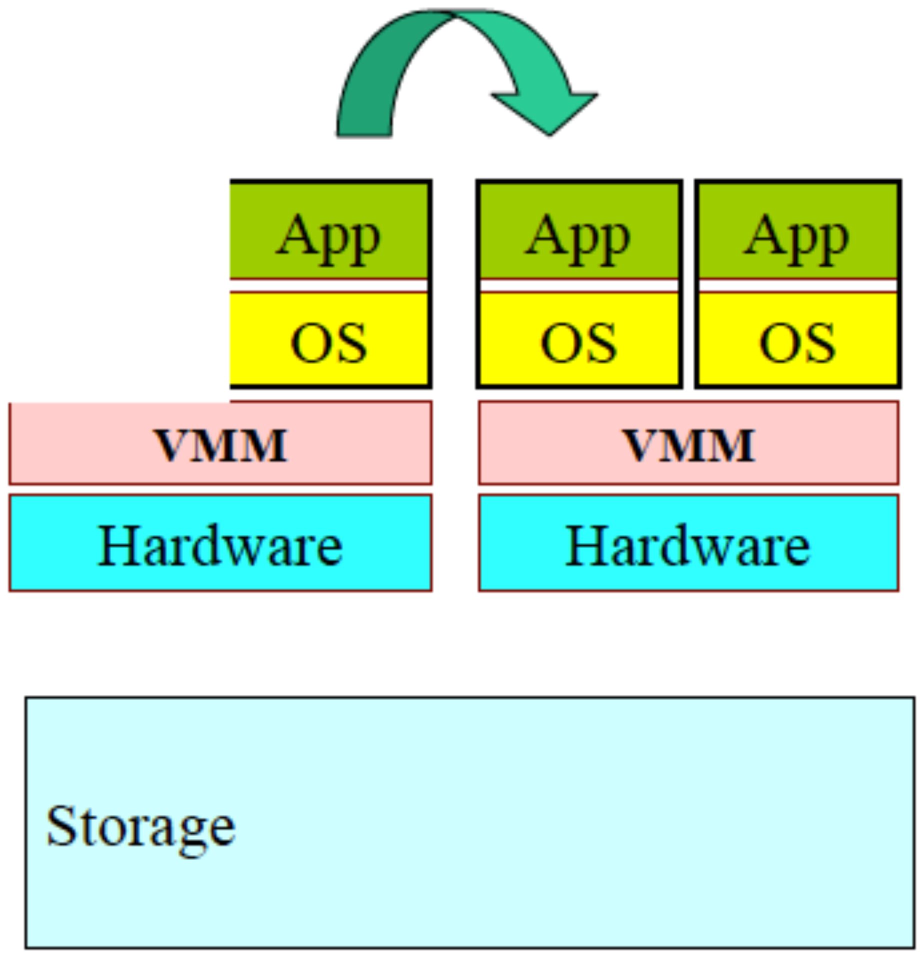
VMM Functions: Suspend a VM



VMM Functions: Resume (Provision)



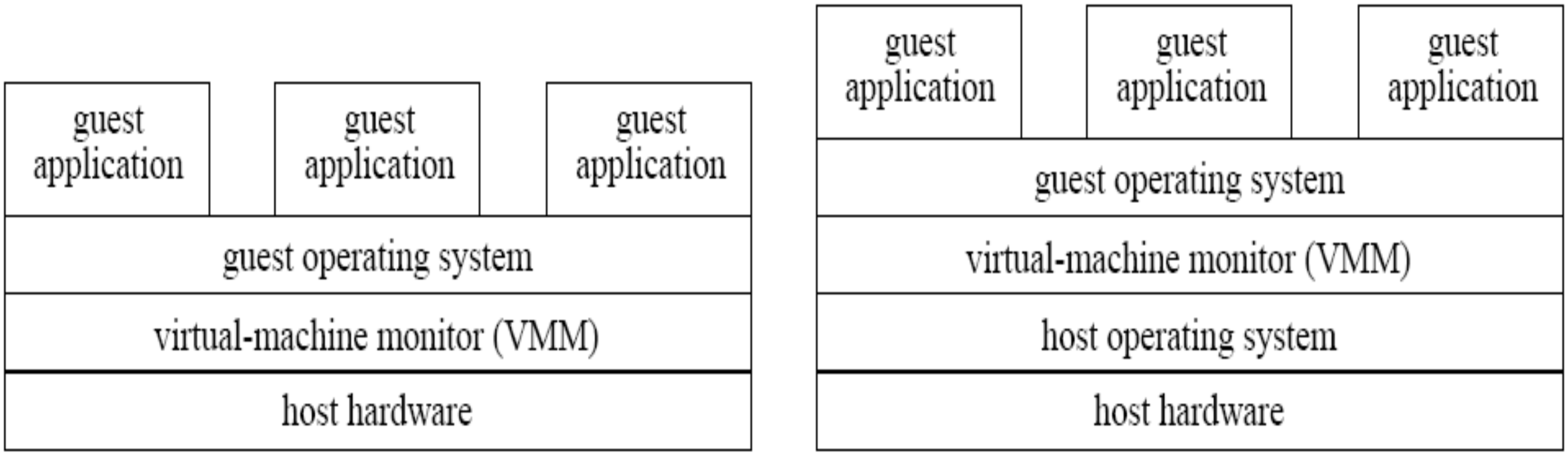
VMM Functions: Migrate



Goals

- Manageability
  - Ease maintenance, administration, provisioning, etc.
- Performance
  - Overhead of virtualization should be small
- Power Savings
  - Server Consolidation
- Isolation
  - Activity of one VM should not impact other active VMs
  - Data of one VM is inaccessible by another
- Scalability
  - Minimize cost per VM

VMM Types

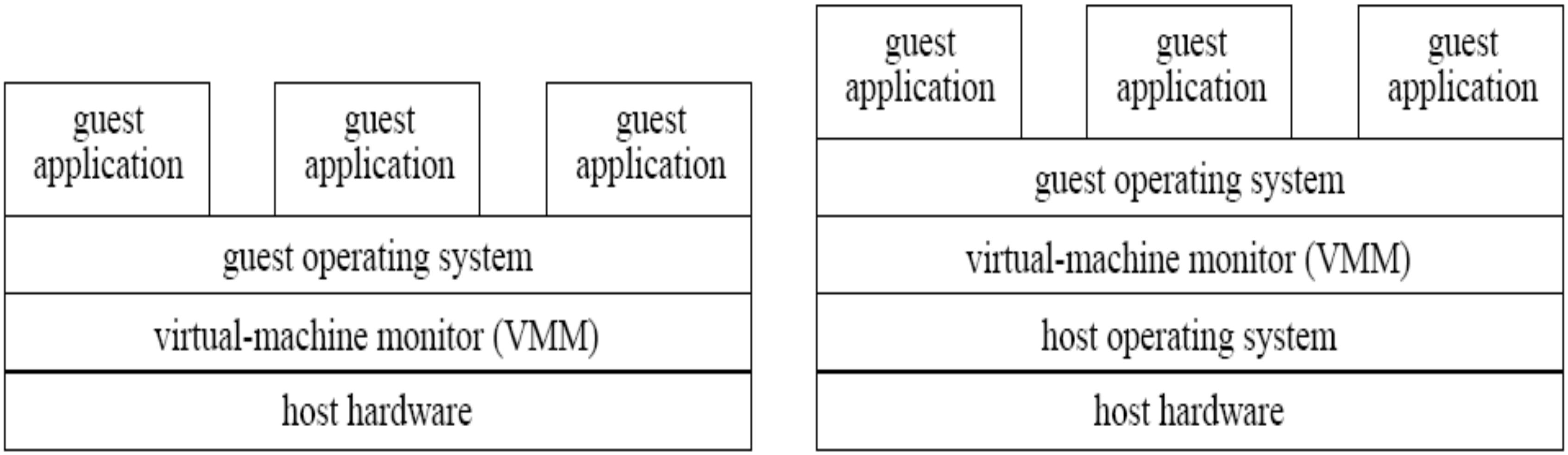


Type I VMM

Type II VMM

For VM approaches you have used, which type are they?

VMM Challenges



Type I VMM

Type II VMM

What seems difficult about building VM approaches?

Virtual Machine Monitor (VMM)

- Resides as a layer below the (guest) operating system
- Presents a hardware interface to a (guest) OS
- Multiplexes resources between several virtual machines (VMs)
- Performance Isolates VMs from each other

When/Why/How would all this be useful?

Virtualization Styles

- Fully virtual*izing* VMM
  - Virtual machine looks exactly like some physical machine.
  - (But maybe not the one you're running on right now.)
  - Run OS or other software unchanged (from the machine the VM mimics)
- Para- virtual*izing* VMM
  - Some architecture features are hard to virtualize, so exact copy is too difficult (or slow).
  - Instead, punt on a few features.
  - VMM provides idealized view of hardware and then fixes under the covers.
  - Since the VMM doesn't match any real hardware, an OS running on it MUST be changed, not legacy.

If you are an application programmer, how could you figure out whether your code is running FV, PV, or non-Virtualized?

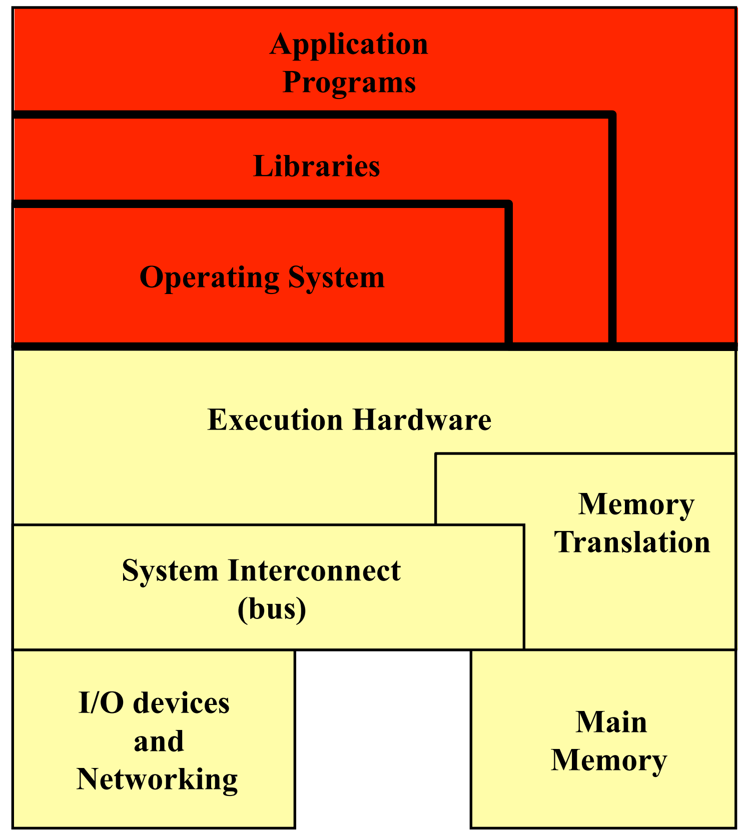
VMM Implementation

Should efficiently virtualize the hardware

- Provide illusion of multiple machines
- Retain control of the physical machine

Subsystems

- Processor Virtualization
- I/O virtualization
- Memory Virtualization



Processor Virtualization

Popek and Goldberg (1974)

- Sensitive instructions: only executed in kernel mode
- Privileged instructions: trap when run in user mode
- CPU architecture is virtualizable only if sensitive instructions are subset of privileged instructions

- When guest OS runs a sensitive instruction, must trap to VMM so it maintains control

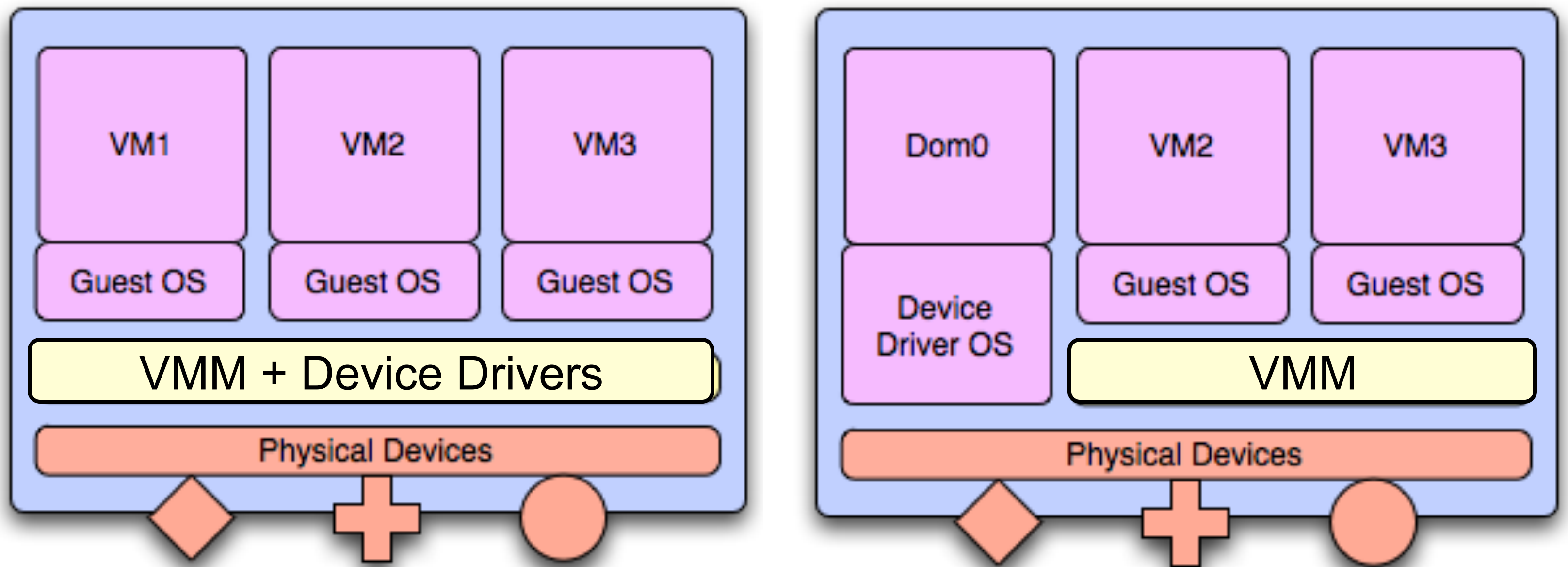
x86 Processor Virtualization

- x86 architecture is not fully virtual*izable*
  - Certain privileged instructions behave differently when run in unprivileged mode
    - POPF instruction that is used to set and clear the interrupt-disable flag. If run in user mode, it has no effect: it's a NO-OP.
  - Certain unprivileged instructions can access privileged state
- Techniques to address inability to virtualize x86
  - Replace non-virtualizable instructions with easily virtualized ones statically (Paravirtualization)
  - Perform Binary Translation (Full Virtualization)

I/O Virtualization

- Issue: lots of I/O devices
- Problem: Writing device drivers for all I/O device in the VMM layer is not a feasible option
- Insight: Device driver already written for popular Operating Systems
- Solution: Present *virtual* I/O devices to *guest* VMs and channel I/O requests to a trusted *host* VM running popular OS

I/O Virtualization



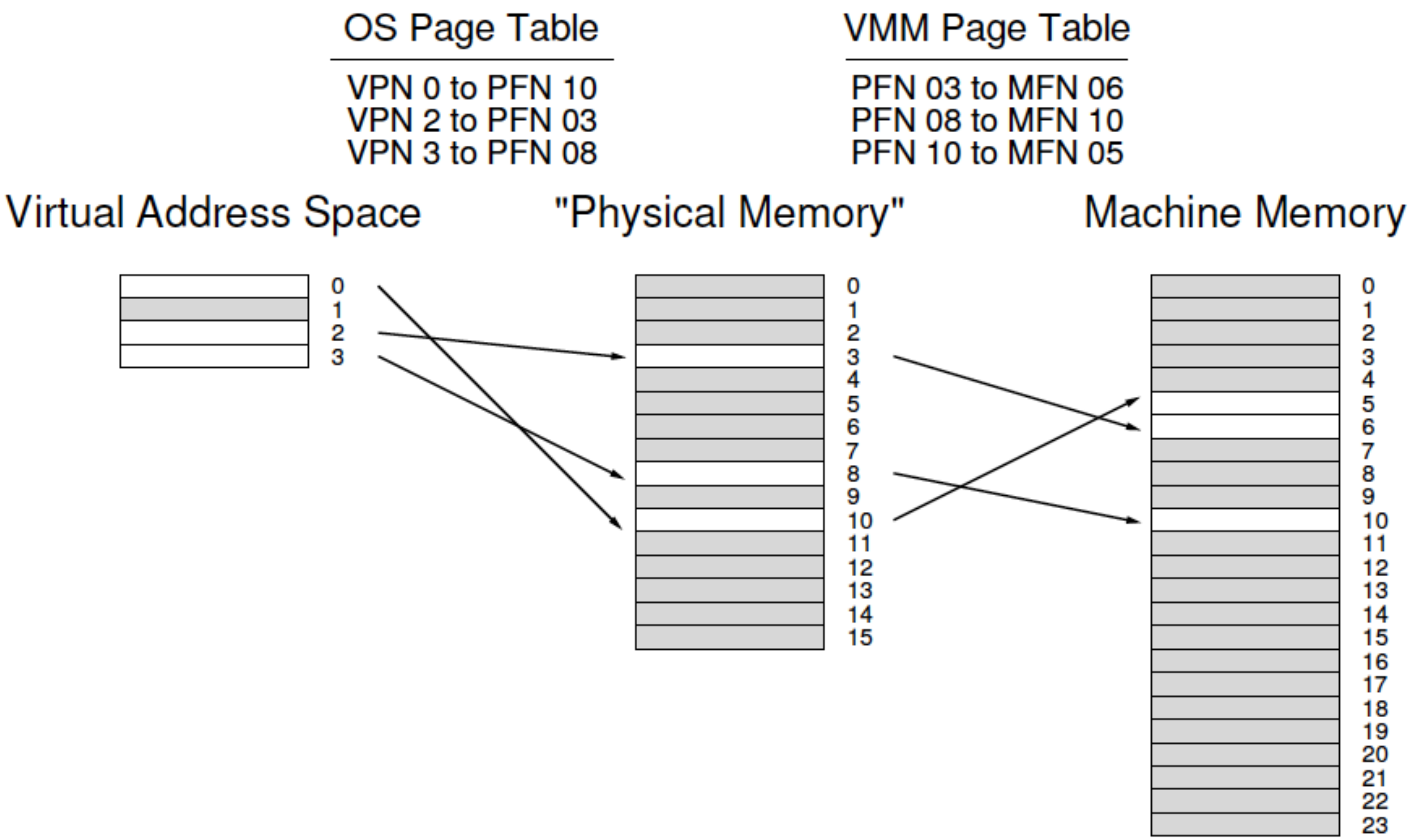
Higher performance, but PITA to write all the drivers

Lower performance, but reuses drivers guest OS already has.

Memory Virtualization

- Traditional way is to have the VMM maintain a shadow of the VM's page table
- The shadow page table controls which pages of machine memory are assigned to a given VM
- When guest OS updates its page table, VMM updates the shadow

Another layer of indirection…



Case Study: VMware ESX Server

- Type I VMM - Runs on bare hardware
- Full-virtualized – Legacy OS can run unmodified on top of ESX server
- Fully controls hardware resources and provides good performance

ESX Server – CPU Virtualization

- Most user code executes in Direct Execution mode; near native performance

- Uses *runtime* Binary Translation for x86 virtualization
  - Privileged mode code is run under control of a Binary Translator, which emulates problematic instructions
  - Fast compared to other binary translators as source and destination instruction sets are nearly identical

ESX Server – Memory Virtualization

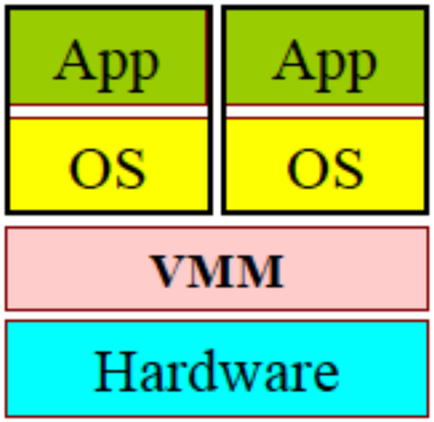
- Maintains shadow page tables with virtual to machine address mappings.
- Shadow page tables are used by the physical processor
- Guest OS page table: maps virtual addresses to “physical” addresses (note quotes)
- ESX maintains the pmap data structure per VM: maps “physical” to machine address mappings
- Shadow page table holds the combined effects of these two map steps
- ESX can easily remap a machine page when needed

ESX Server – I/O Virtualization

- Has highly optimized storage subsystem for networking and storage devices
  - Directly integrated into the VMM
  - Uses device drivers from the Linux kernel to talk directly to the device
- Low performance devices are channeled to special "host" VM, which runs a full Linux OS

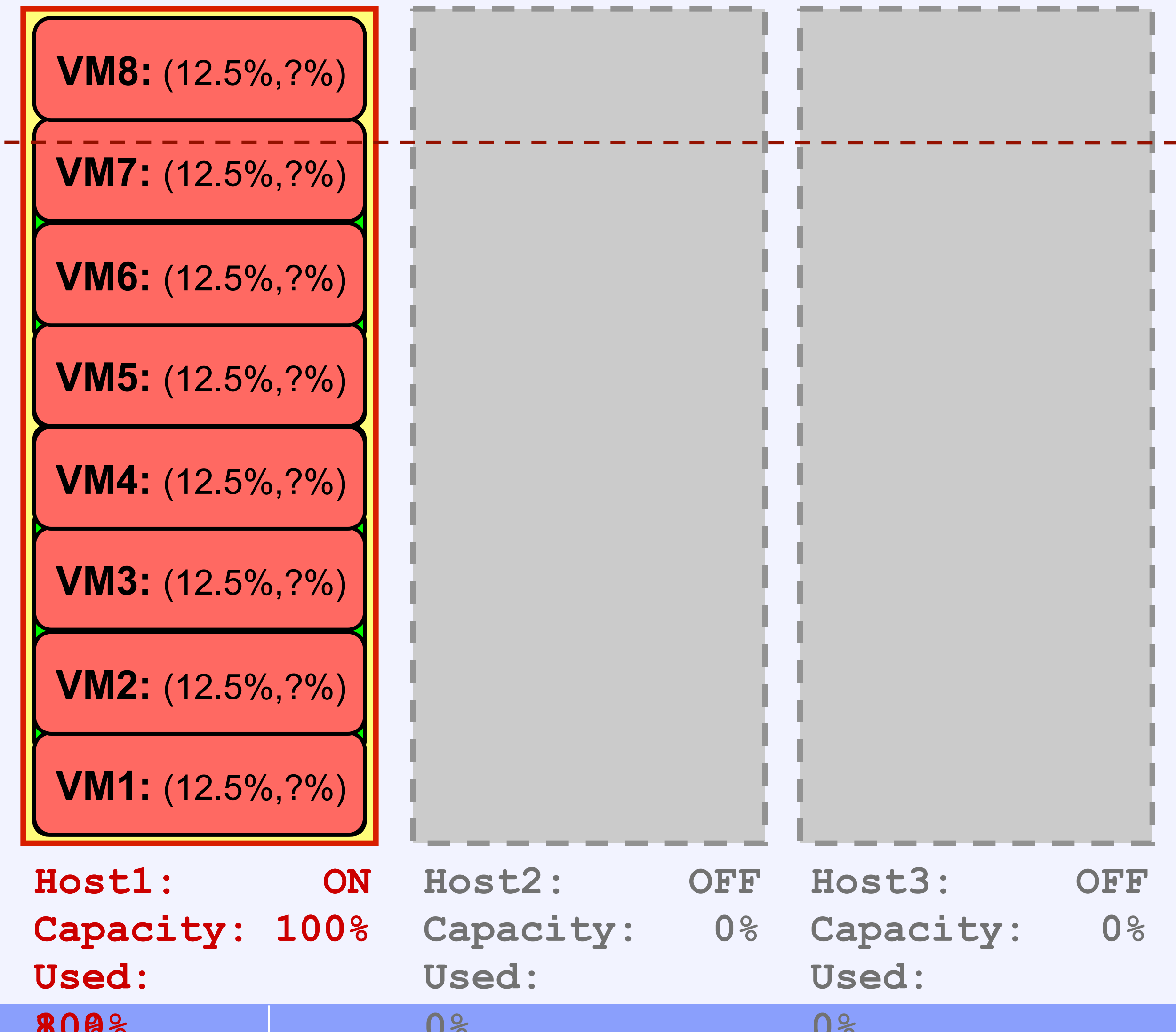
Virtualization: Remaining challenges?

- One big one is "demand estimation"
- Normally the OS manages resources, but in this case, the VMM is supposed to mediate between multiple entities each running different apps/OSs.
- How to know what they really need?
- One example:



Canturk Isci, James Hanson, Ian Whalley, Malgorzata Steinder and Jeff Kephart , Runtime Demand Estimation for Effective Dynamic Resource Management. In IEEE/ IFIP Network Operations and Management Symposium (NOMS). Osaka, Japan, Apr. 2010.

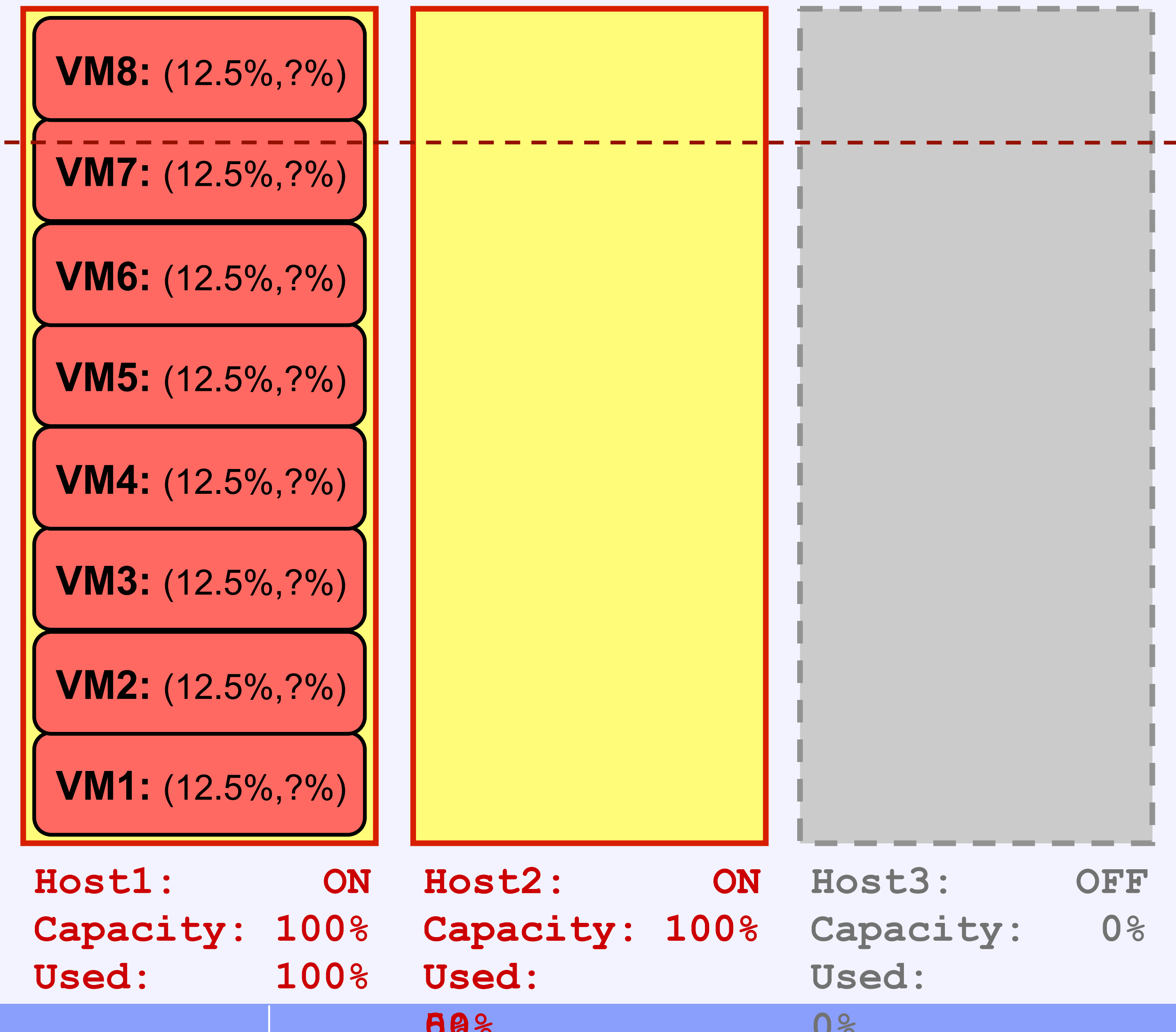
A Motivating Example:
Dynamic Resource Management w/o Demand Knowledge



0. Initially all VMs: 10% demand
1. Demand rises to 25%

No contention
Resource contention
VM: (<usage>,<demand>)

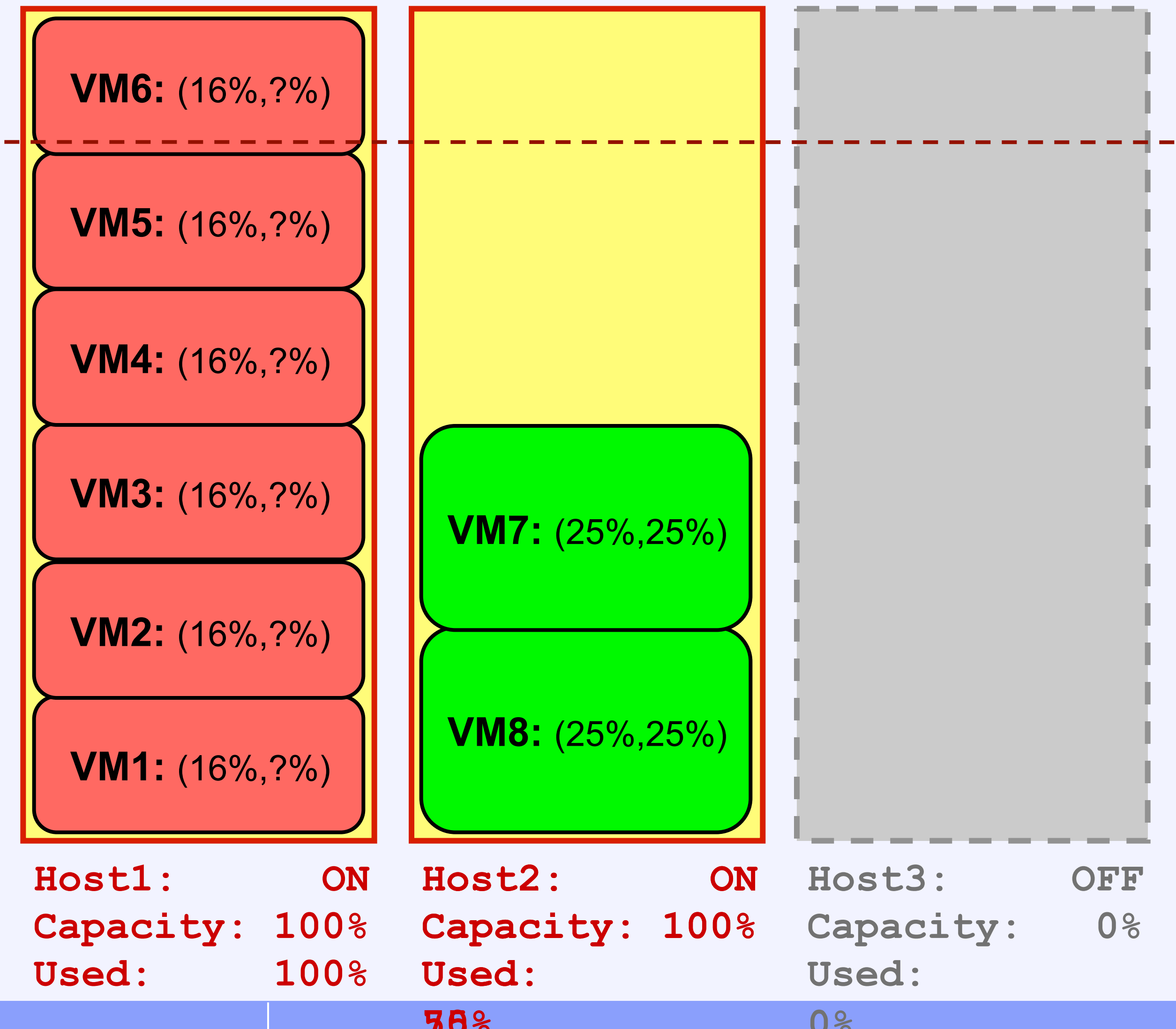
A Motivating Example:
Dynamic Resource Management w/o Demand Knowledge



0. Initially all VMs: 10% demand
1. Demand rises to 25%
   Host2 powered ON
   Two VMs moved out

No contention
Resource contention
VM: (<usage>,<demand>)

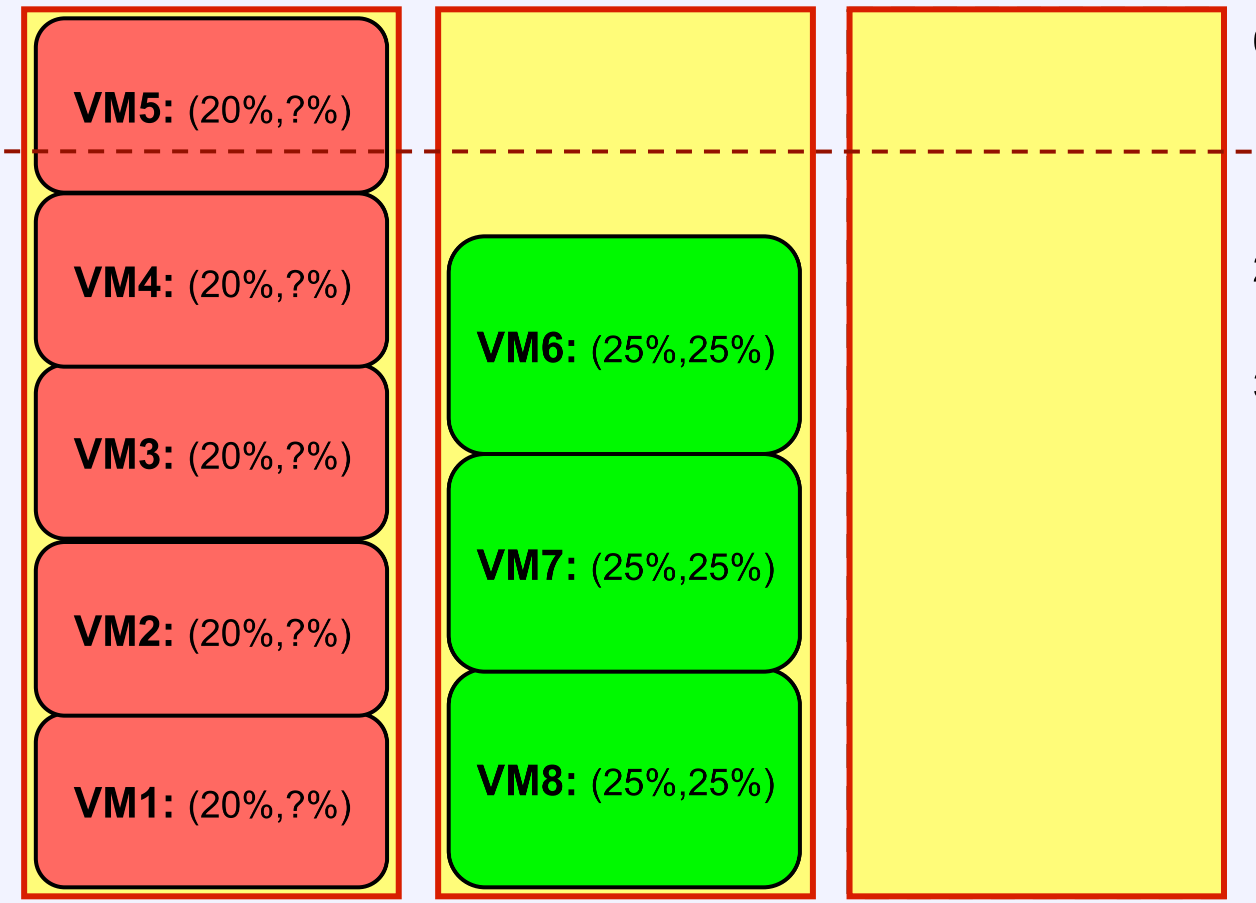
A Motivating Example:
Dynamic Resource Management w/o Demand Knowledge



0. Initially all VMs: 10% demand
1. Demand rises to 25%
   Host2 powered ON
   Two VMs moved out
2. VMs on Host1 inflate more
   One more VM moved out

No contention
Resource contention
VM: (<usage>,<demand>)

A Motivating Example:
Dynamic Resource Management w/o Demand Knowledge



Host1: ON
Capacity: 100%
Used: 100%

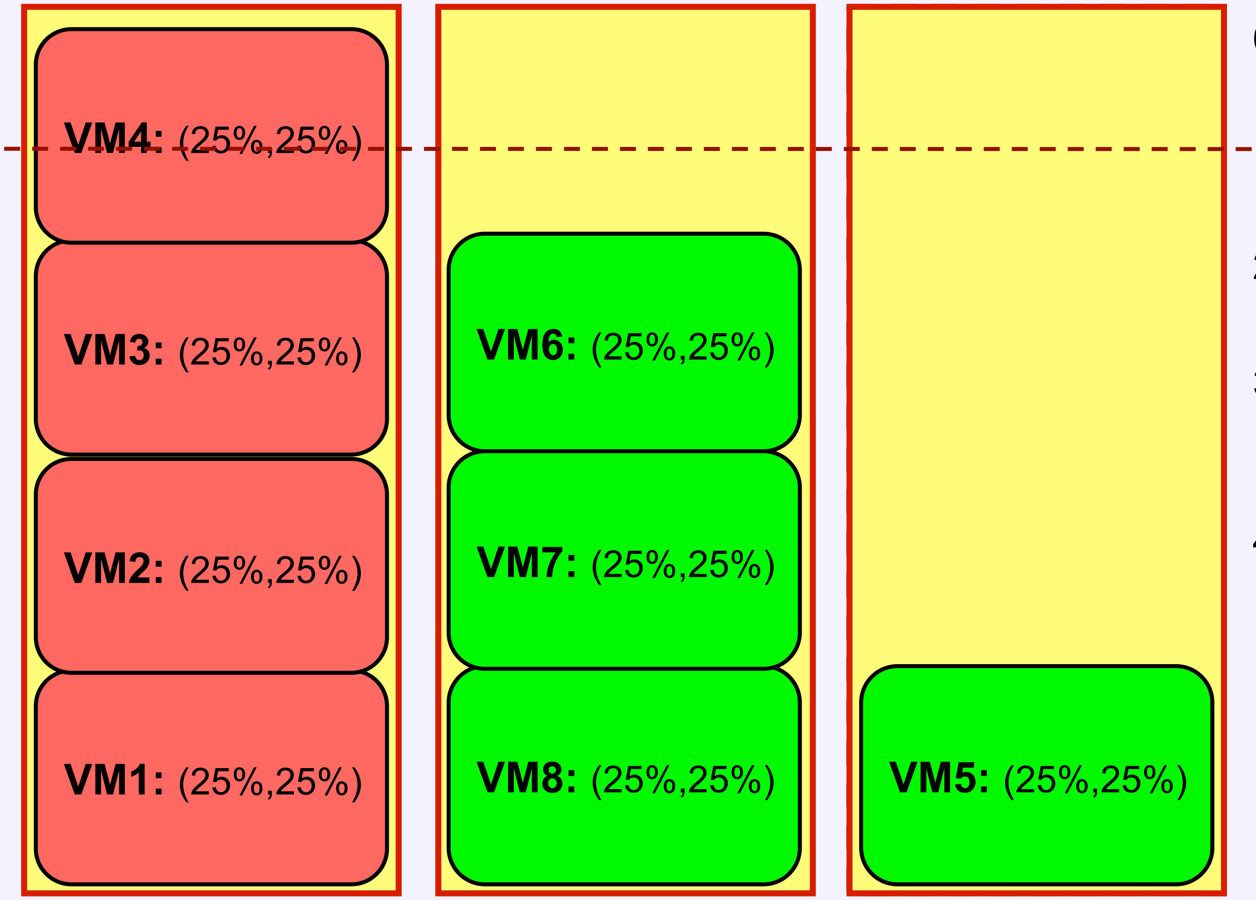
Host2: ON
Capacity: 100%
Used: 75%

Host3: OFF ON
Capacity: 100%
Used: 0 25%

0. Initially all VMs: 10% demand
1. Demand rises to 25%
   Host2 powered ON
   Two VMs moved out
2. VMs on Host1 inflate more
   One more VM moved out
3. VMs on Host1 inflate more
   Host3 powered ON
   One more VM moved out

No contention
Resource contention
VM: (<usage>,<demand>)

A Motivating Example: Dynamic Resource Management w/o Demand Knowledge



Host1: ON
Capacity: 100%
Used: 105%

Host2: ON
Capacity: 100%
Used: 75%

Host3: ON
Capacity: 100%
Used: 25%

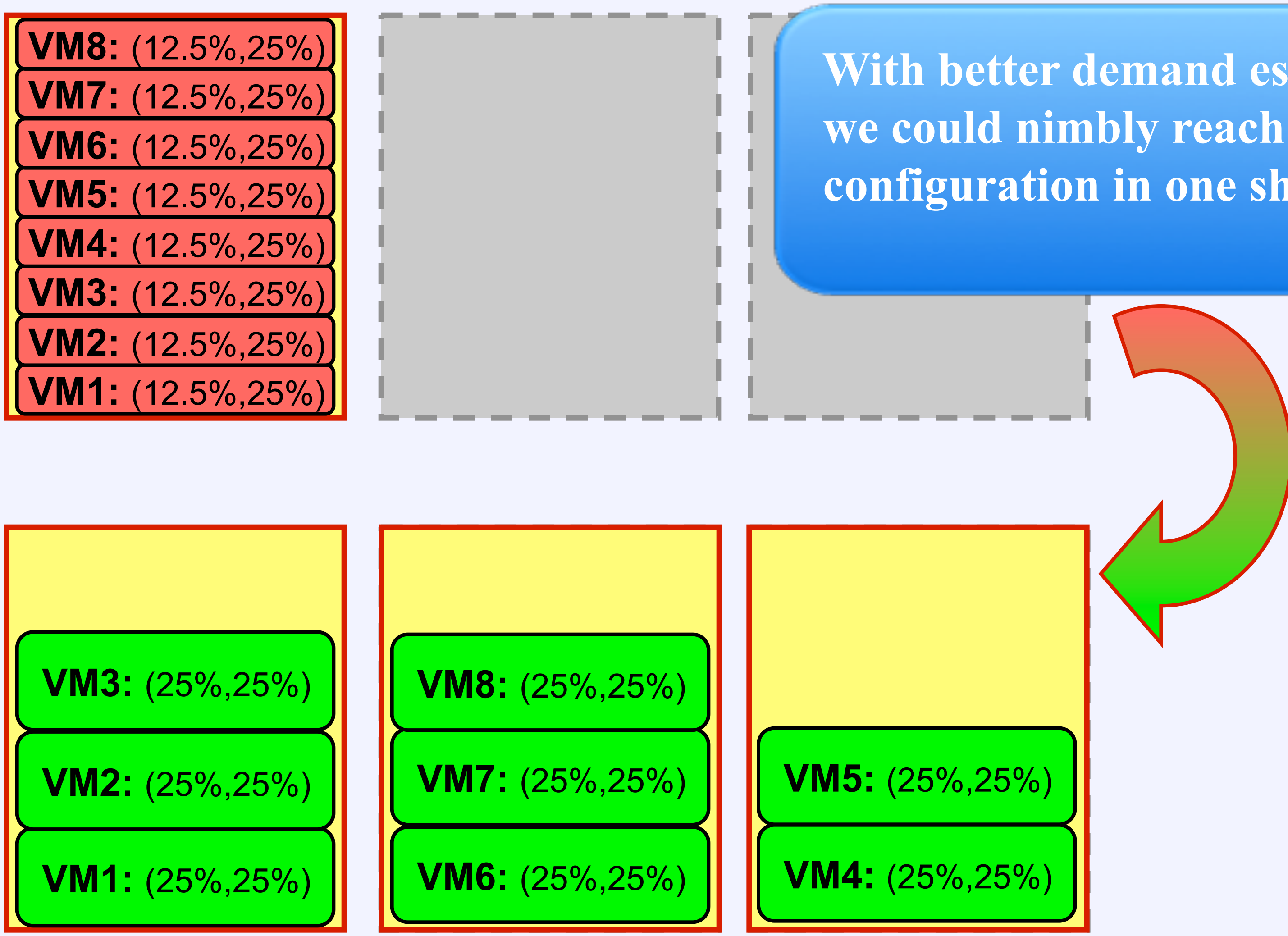
0. Initially all VMs: 10% demand
1. Demand rises to 25%
   Host2 powered ON
   Two VMs moved out
2. VMs on Host1 inflate more
   One more VM moved out
3. VMs on Host1 inflate more
   Host3 powered ON
   One more VM moved out
4. VMs on Host1 inflate more
   One more VM moved out

Not very nimble!

No contention
Resource contention
VM: (<usage>,<demand>)

If We Only Knew **Actual** Demand

With better demand estimation: we could nimbly reach the right configuration in one shot!



Our Solution: CPU Accounting & Demand Estimation

- Hypothetical Example: (t~ hypervisor scheduling quantum)

1 Host: 2PCPUs 200% Capacity
10 VMs: 1VCPU, 80% CPU: [4t Run + 1t Sleep]

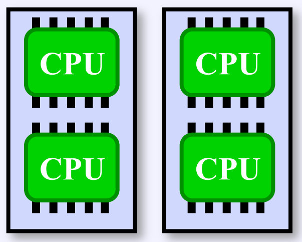
U: Used | R: Ready | W: Wait

t:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
VM1:	U	R	R	R	R	U	R	R	R	R	U	R	R	R	R	U	**W**	R	R	R	U	R	R	R
VM2:	U	R	R	R	R	U	R	R	R	R	U	R	R	R	R	U	**W**	R	R	R	U	R	R	R
VM3:	R	U	R	R	R	R	U	R	R	R	R	U	R	R	R	R	U	**W**	R	R	R	U	R	R
VM4:	R	U	R	R	R	R	U	R	R	R	R	U	R	R	R	R	U	**W**	R	R	R	U	R	R
VM5:	R	R	U	R	R	R	R	U	R	R	R	R	U	R	R	R	R	U	**W**	R	R	R	U	R
VM6:	R	R	U	R	R	R	R	U	R	R	R	R	U	R	R	R	R	U	**W**	R	R	R	U	R
VM7:	R	R	R	U	R	R	R	R	U	R	R	R	R	U	R	R	R	R	U	**W**	R	R	R	U
VM8:	R	R	R	U	R	R	R	R	U	R	R	R	R	U	R	R	R	R	U	**W**	R	R	R	U
VM9:	R	R	R	R	U	R	R	R	R	U	R	R	R	R	U	R	R	R	R	U	**W**	R	R	R
VM10:	R	R	R	R	U	R	R	R	R	U	R	R	R	R	U	R	R	R	R	U	**W**	R	R	R

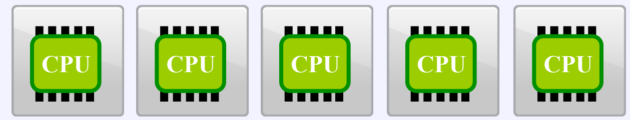
$$CPU\_Demand \approx \frac{Used(+System)}{Used + Wait(+System)} = \frac{Used(+System)}{Stats\_Period - Ready}$$

Evaluation Case Study

- Cluster:

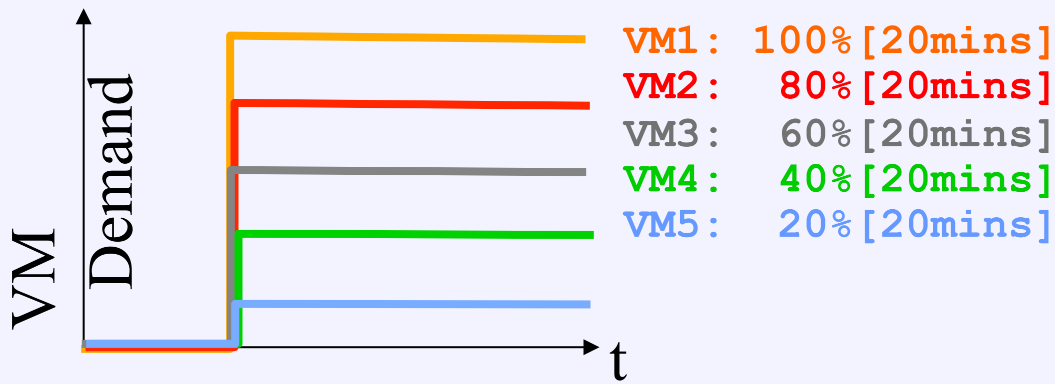


2 Hosts
2 CPU
200% each



5VMs, 1CPU, 0-100% each

Load Configuration:

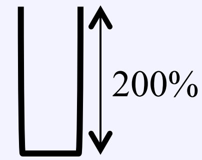


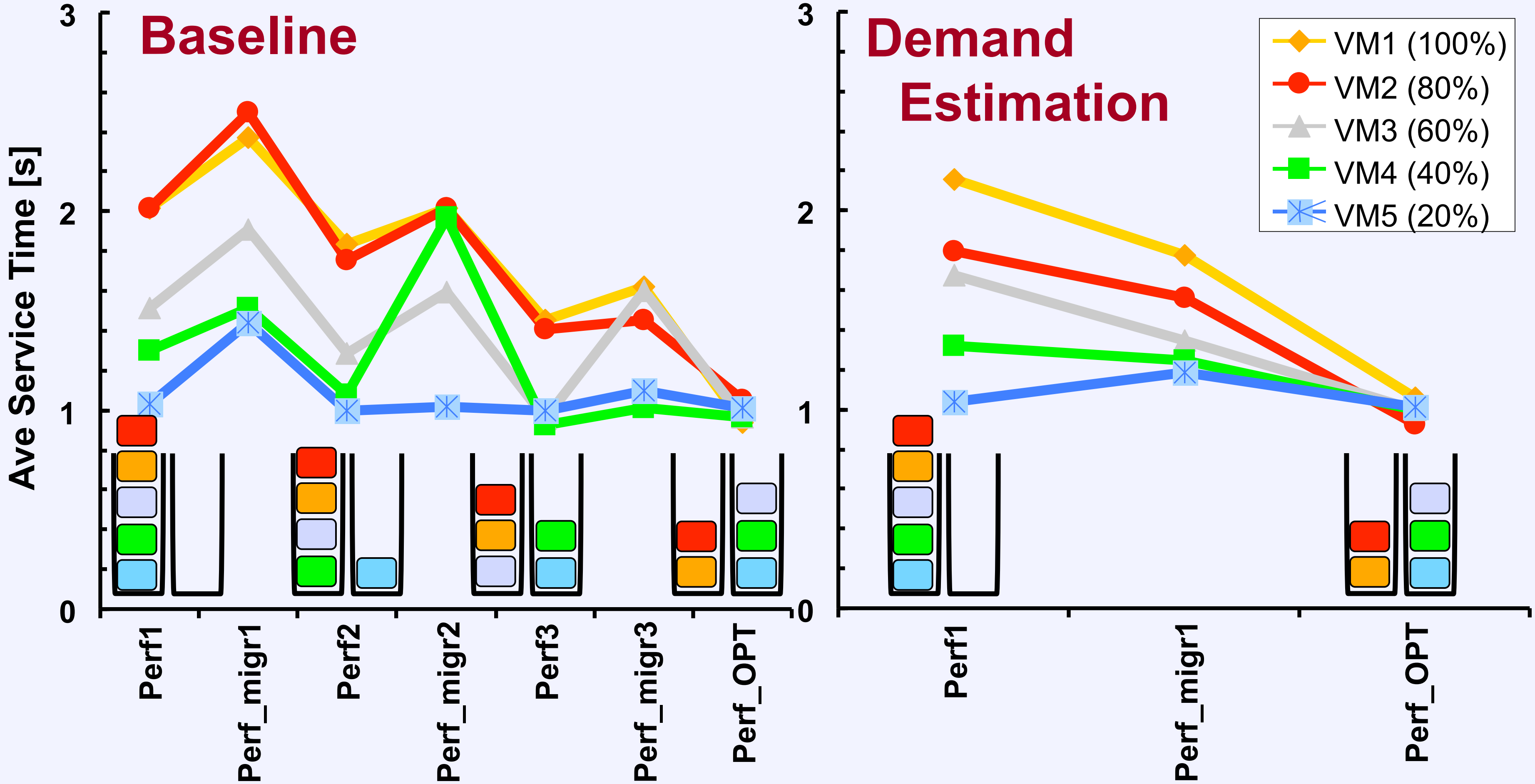
Experiment similar to motivational example

- Pack all VMs on Host1
- Increase load
- Evaluate with and without demand estimation

Dynamic Placement Steps

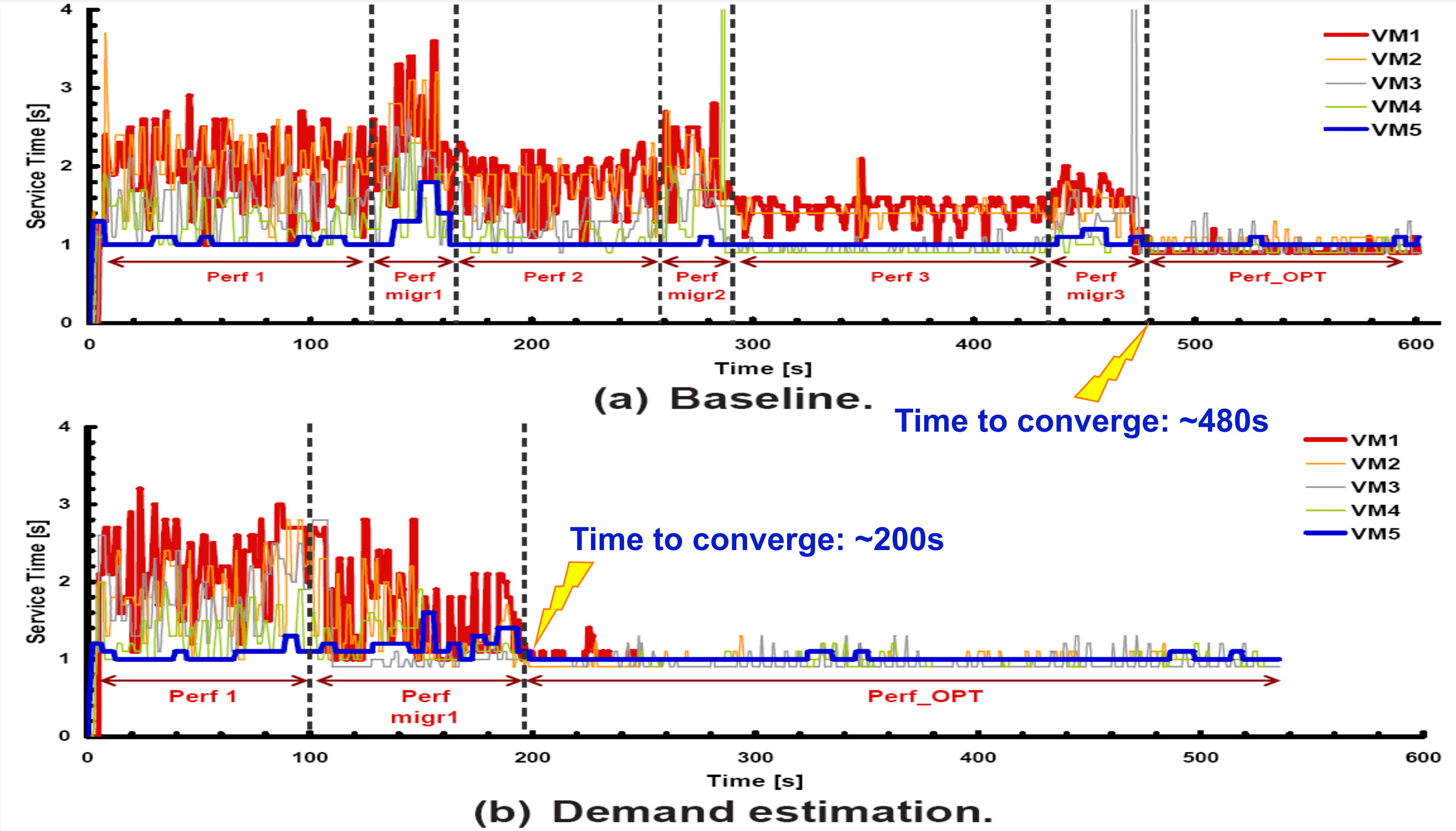
Each host: 200% capacity





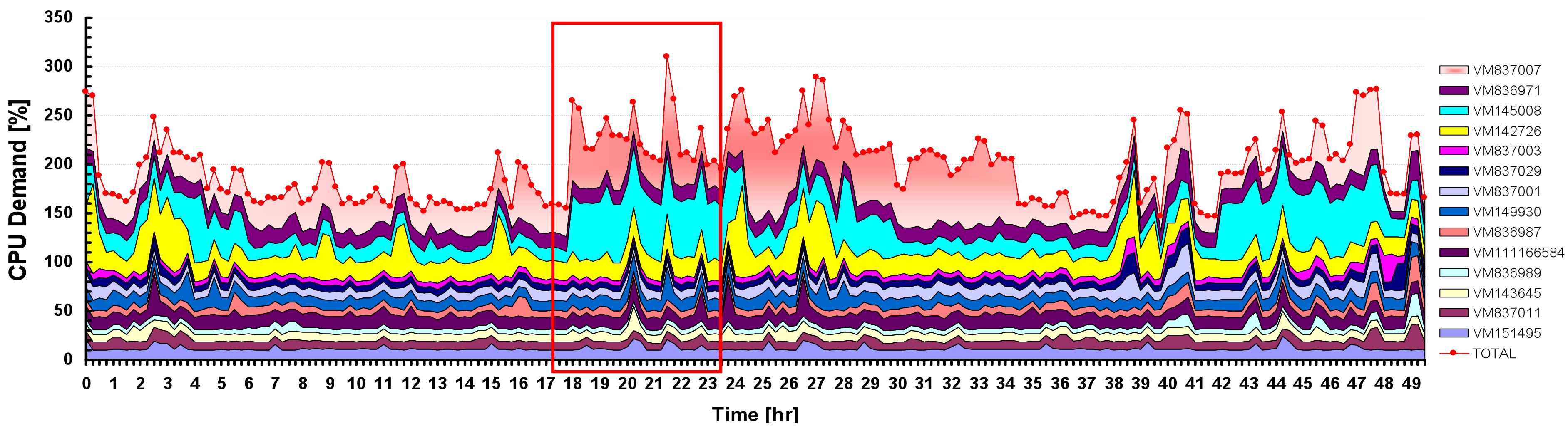
One-shot resolution instead of the iterative process!

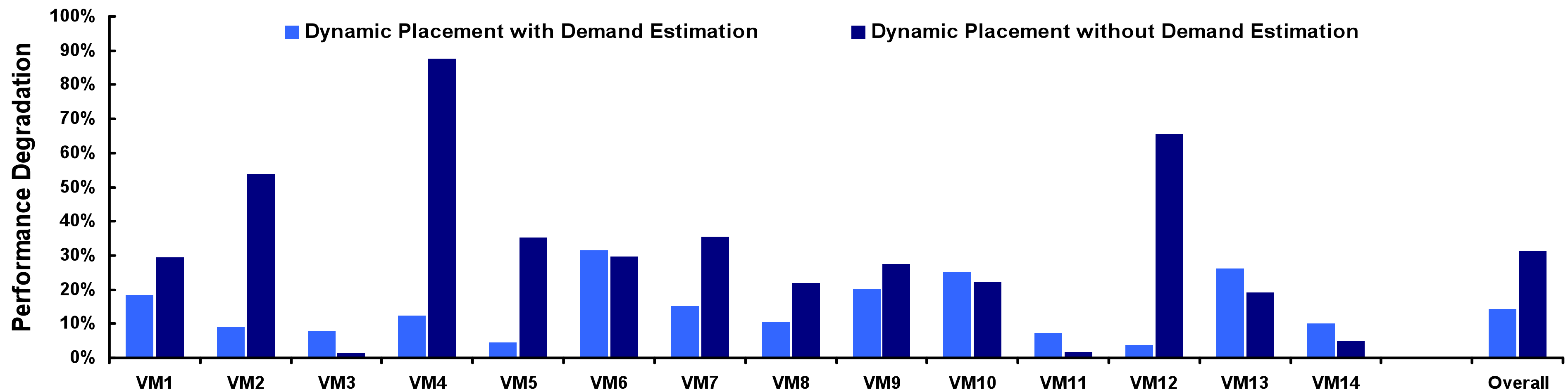
Performance: Time to Converge



Reduce time required to converge to the optimal allocation by 2X!

Evaluation with Data Center Loads





Reduce aggregate performance degradation by 2X across all VMs!

Summary

- Virtualization is here:
  - Compatibility and abstraction
  - Server Consolidation
  - Migration and maintenance
- Interesting design problems:
  - Hardware Support
  - Software design approaches
- Tons of Policy and measurement issues:
  - Managing performance, power, fairness, …

COS 318: Operating Systems

Virtual Machine Monitors

Prof. Margaret Martonosi
Computer Science Department
Princeton University

Acknowledgments: Canturk Isci, Ravi Nair, Mendel Rosenblum, James E. Smith.