Containers and Virtual Machines at Scale: A Comparative Study

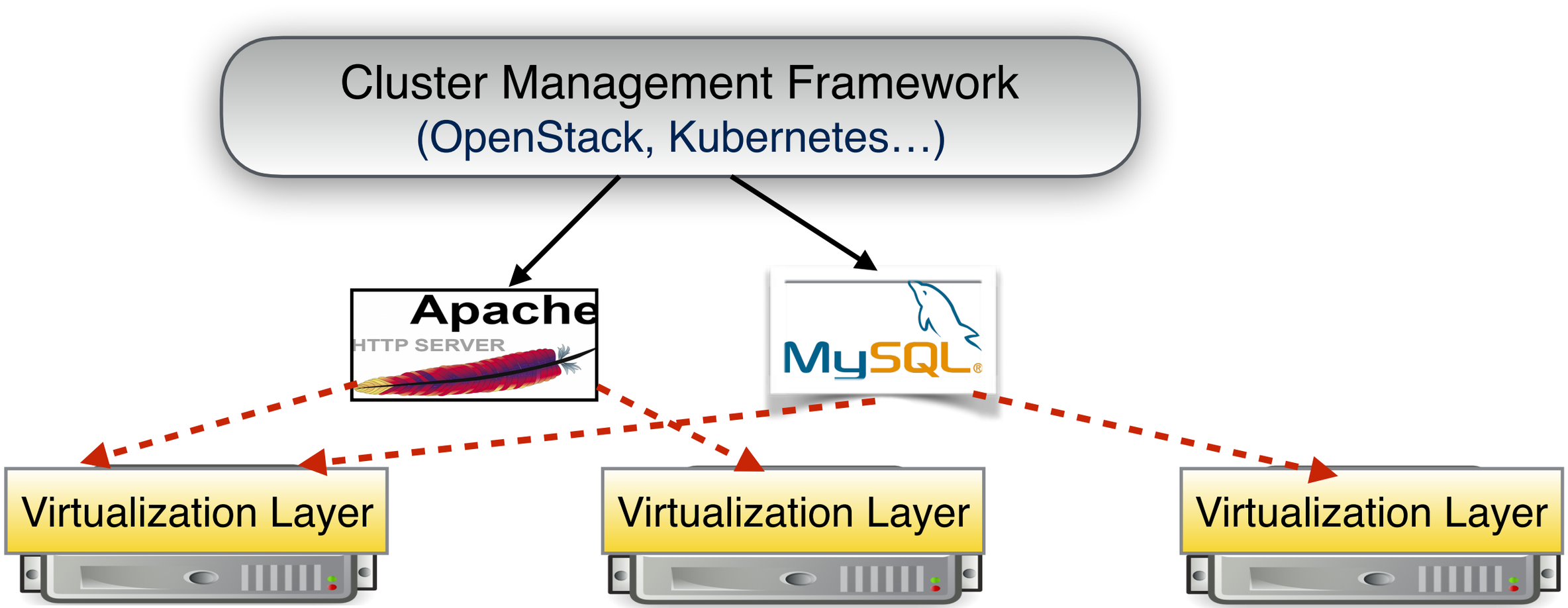
Prateek Sharma
Lucas Chaufournier
Prashant Shenoy
Y.C. Tay

UMASS AMHERST

NUS National University of Singapore

Virtualization in Data Centers and Clouds

- Data centers and clouds host multiple applications
- Virtualization and cluster management frameworks used for managing and multiplexing resources
- Virtualization options: Hardware-level and Operating System-level



System Virtualization Today

	Hardware	**Operating System**
	Virtual Machines	Containers
		
Example	VMWare, Xen, …	Docker, LXC, …
Management	OpenStack,…	Kubernetes,…
Performance	Noticeable overhead	“Lightweight virtualization”

Problem Statement

- Hardware virtualization predominant (especially in public clouds)
- OS virtualization (Containers) is being rapidly adopted

VM 

vs. 

Container

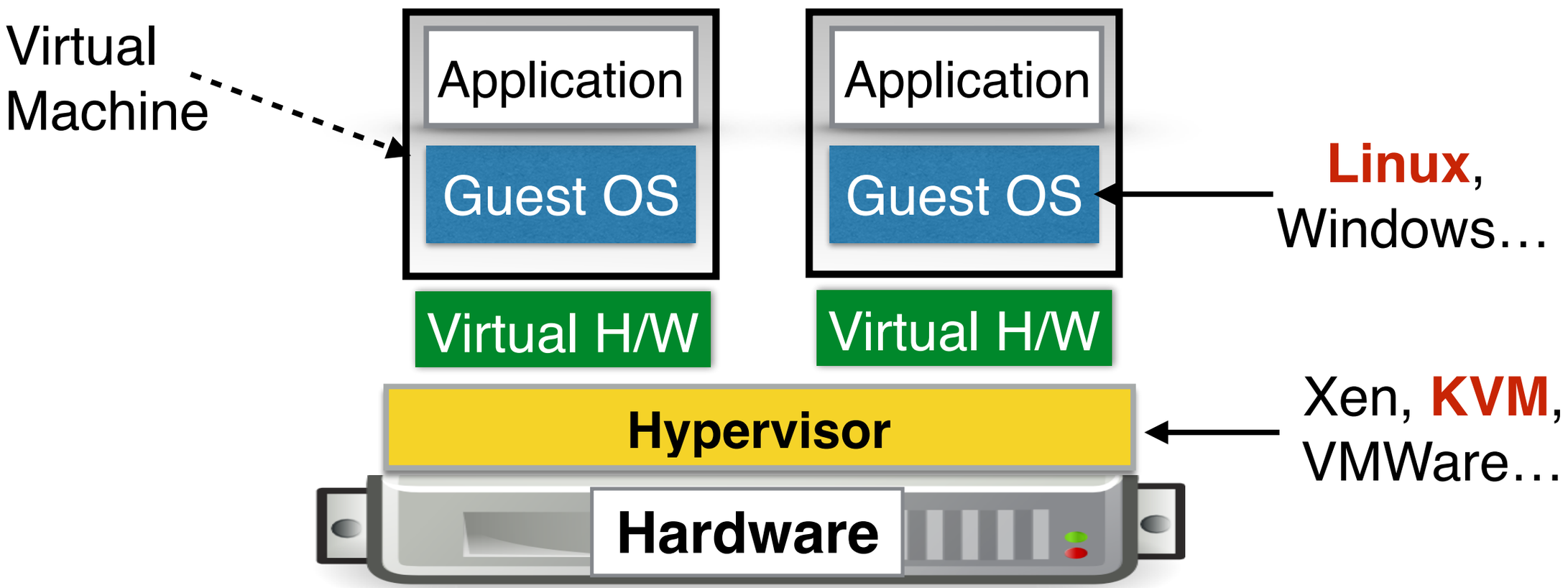
- How do they compare in multi-tenant data centers and cloud environments?
  1. What are the tradeoffs in application performance?
  2. How do they affect resource management and deployment?
  3. What is their influence on software development?

Outline

- Motivation and problem statement
- **Virtualization background**
- Performance comparison
- Managing and deploying applications
- Hybrid virtualization approaches
- Related work
- Conclusion

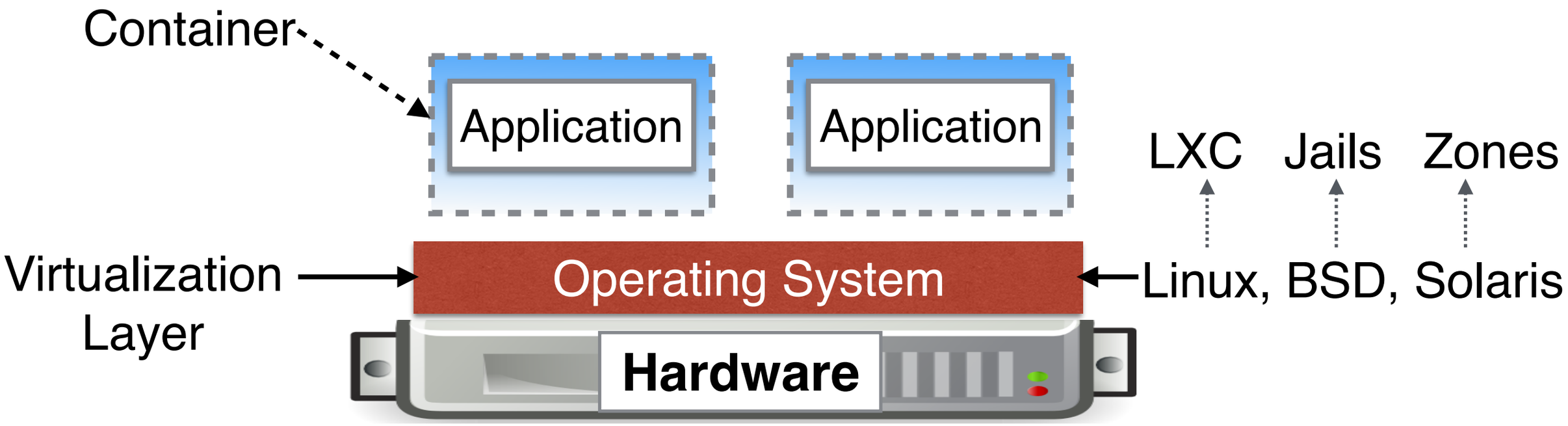
Hardware Virtualization

- *Hypervisor* exposes virtual hardware (CPU, memory, I/O)
- Applications run on top of a guest Operating System
- Multiple layers between application and hardware may reduce performance



Operating System Virtualization

- Lightweight virtualization of OS resources and interfaces
- Applications run in Containers and share the underlying OS
- Cgroups in Linux for isolating resources (cpu, mem, I/O,…)
- Namespaces to isolate process hierarchies, sockets, filesystem,…

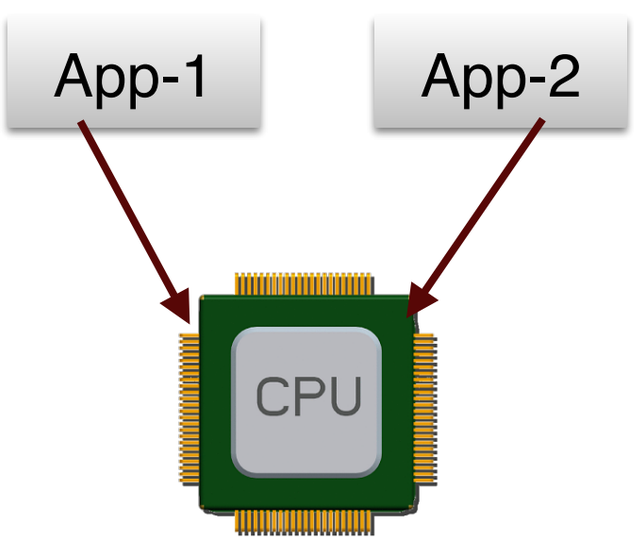


Outline

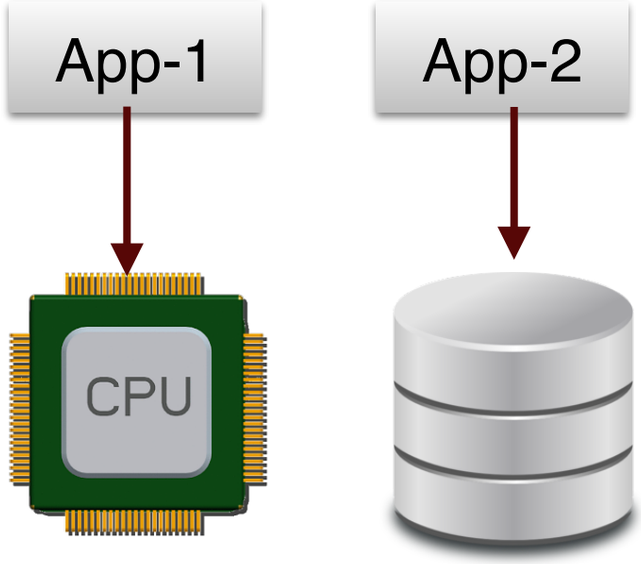
- Motivation and problem statement
- Virtualization background
- **Performance comparison**
- Managing and deploying applications
- Hybrid virtualization approaches
- Related work
- Conclusion

Performance Evaluation Setup

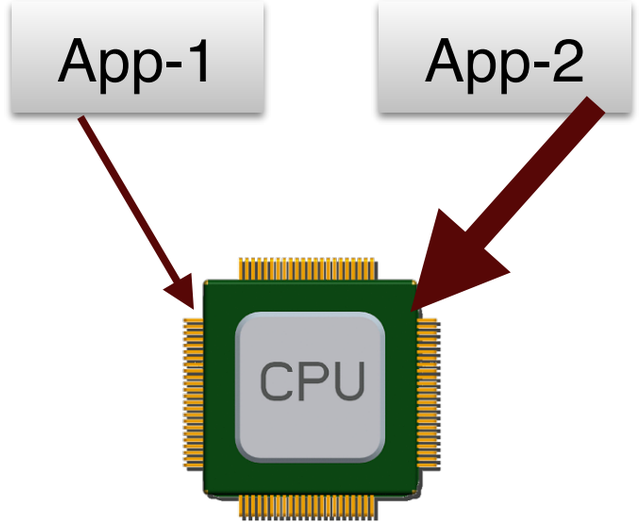
- KVM for hardware virtualization vs. Linux Containers (LXC)
- Containers and VMs given same physical resources using cgroups
- CPU, memory, I/O intensive benchmarks
- In multi-tenant environments, applications can face interference
- Performance depends on type of co-located application:



Competing

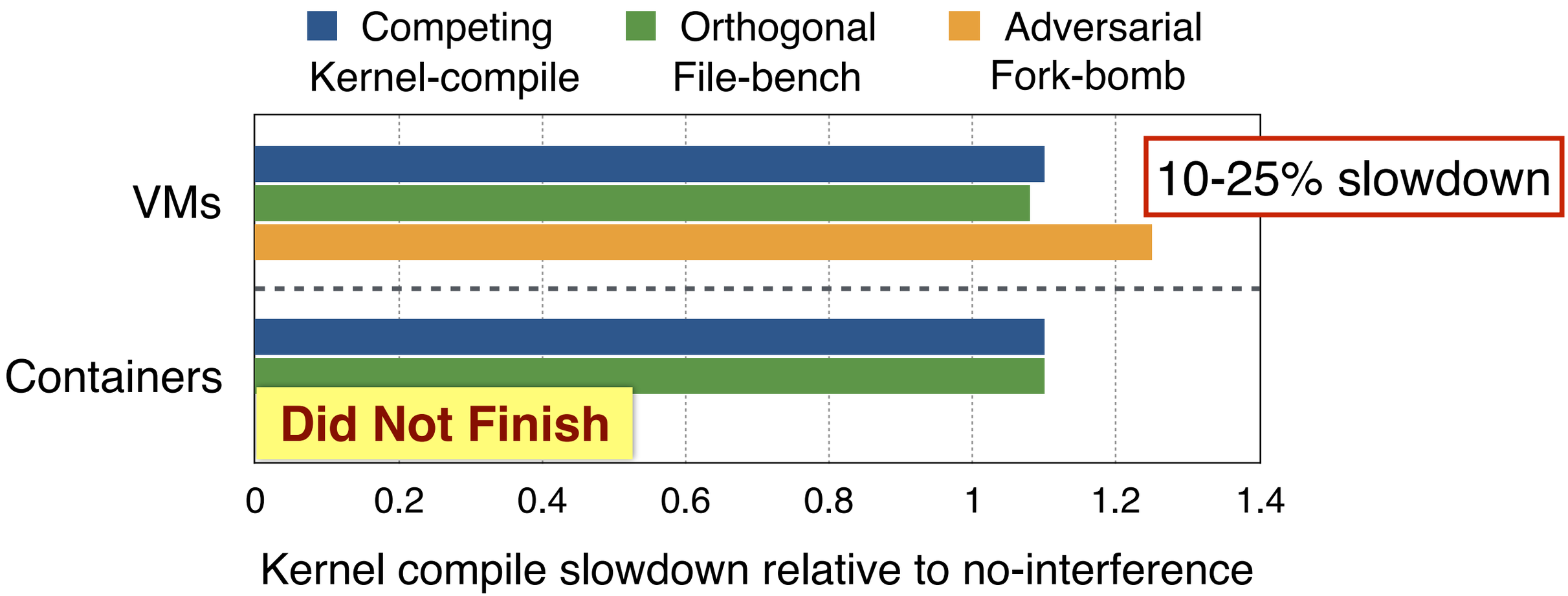


Orthogonal



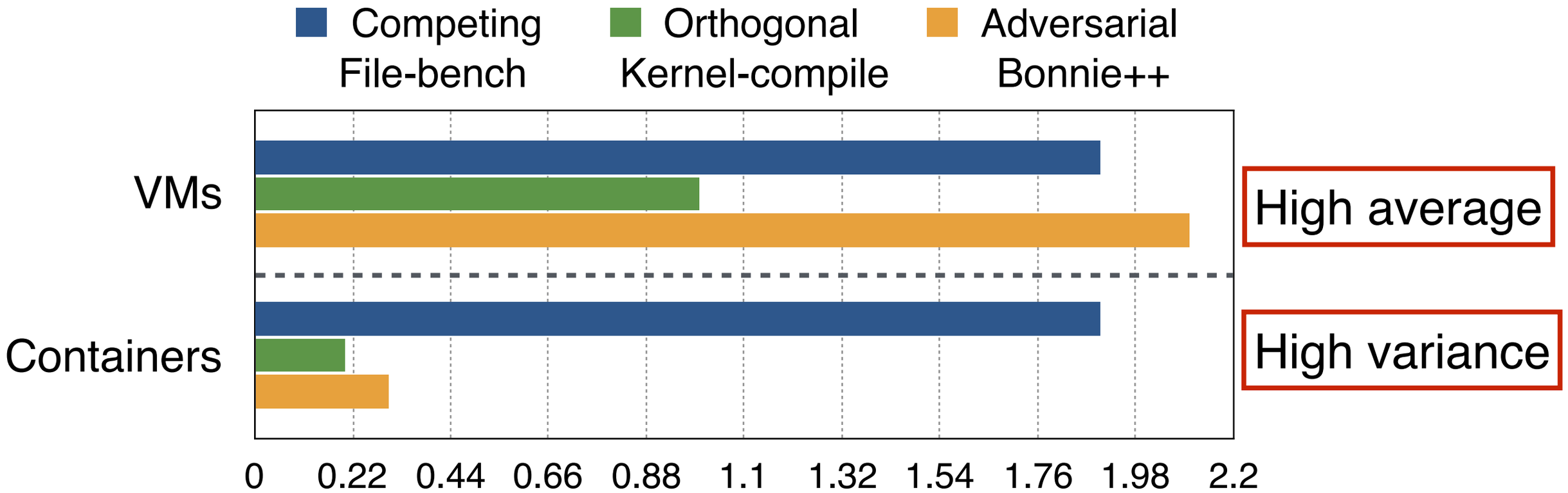
Adversarial
(Intentional starvation)

CPU Isolation

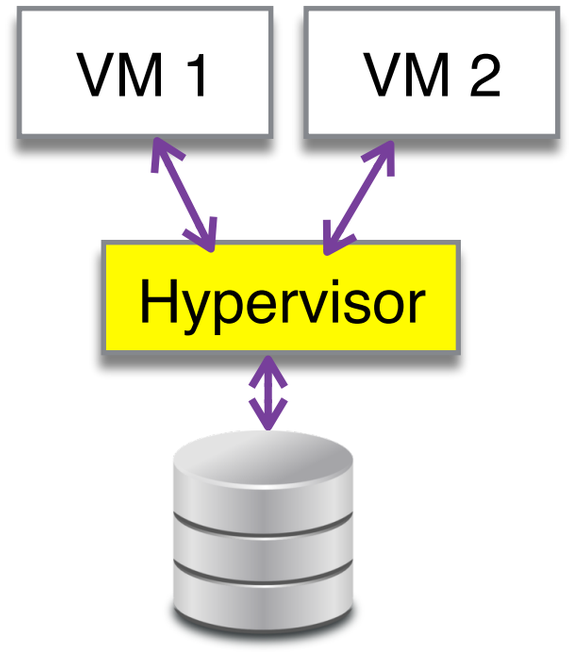


- Container and VM CPUs pinned
- Containers can face starvation due to shared OS CPU scheduler
- **Shared OS or hypervisor components can reduce isolation**
- **Map applications to servers carefully to minimize interference**

I/O Isolation



- Equal allocation of I/O bandwidths using cgroups
- Disk I/O handled by hypervisor (virtIO in KVM)
- **Both VMs and containers have shared I/O paths, causing interference**

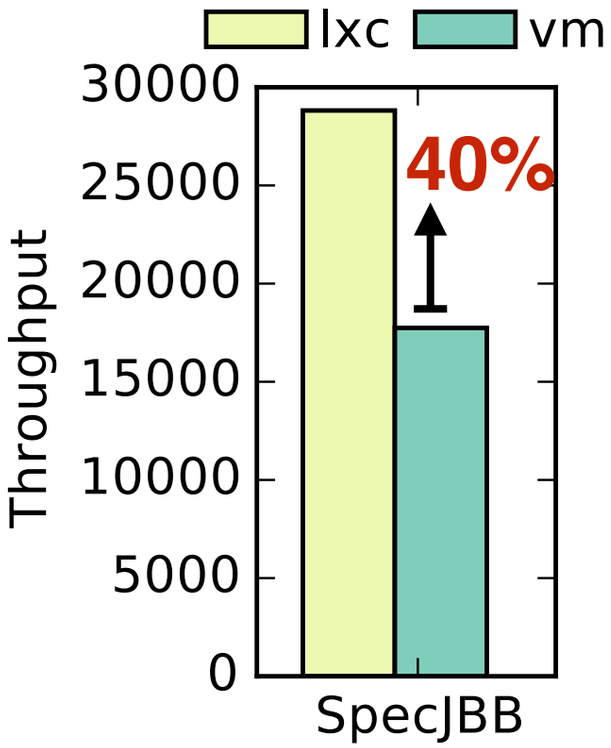


Outline

- Motivation and problem statement
- Virtualization background
- Performance comparison
- **Resource management and deployment**
- Hybrid virtualization approaches
- Related work
- Conclusion

Resource Overcommitment

- Cluster managers *overcommit* resources to increase efficiency
  - Allocate more resources than are actually available
- Hypervisors support overcommitment mechanisms for VMs
- OS overcommits container resources using soft-limits
  - Free (but allocated) resources can be used by other applications



- Memory overcommitted by 2x for SpecJBB
  - Excessive page faults in guest OS, because VM memory limits are hard
- **Performance depends on overcommitment capabilities of virtualization layer**

Migration

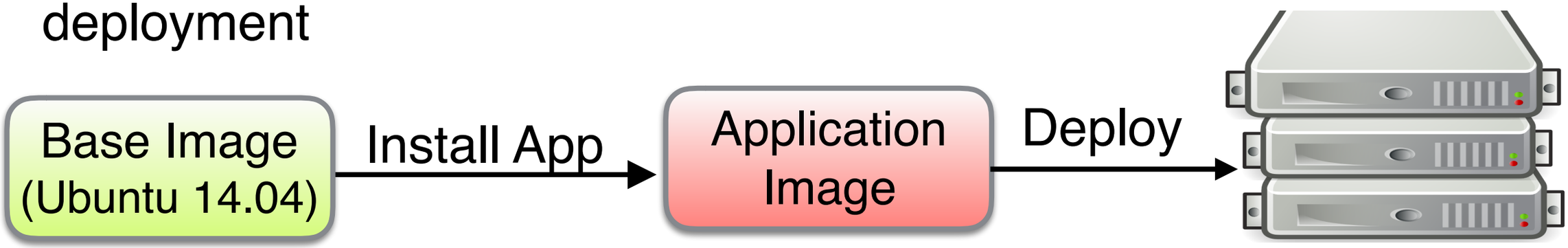
- Live-migration used for load balancing and consolidation
- VM migration implemented in all hypervisors
- Container migration is not production-ready
  - Involves careful preservation of OS state (process table, network sockets, open file descriptors,…)
  - Containers can instead be checkpointed and restarted

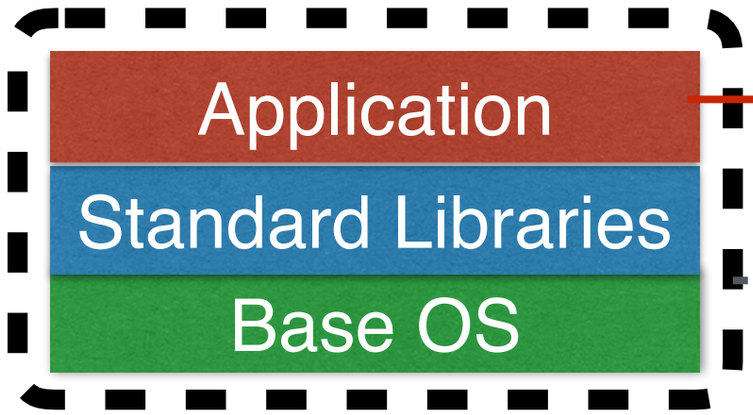
Memory size (GB)	Container	VM
Kernel Compile	0.42	4
YCSB	4	4
SpecJBB	1.7	4
File bench	2.2	4

- Container memory state is smaller
- VMs must also save guest OS and cache pages

Getting Applications Into Production

- Getting applications from development to production involves creating disk images
- Fast image creation enables rapid testing and continuous deployment





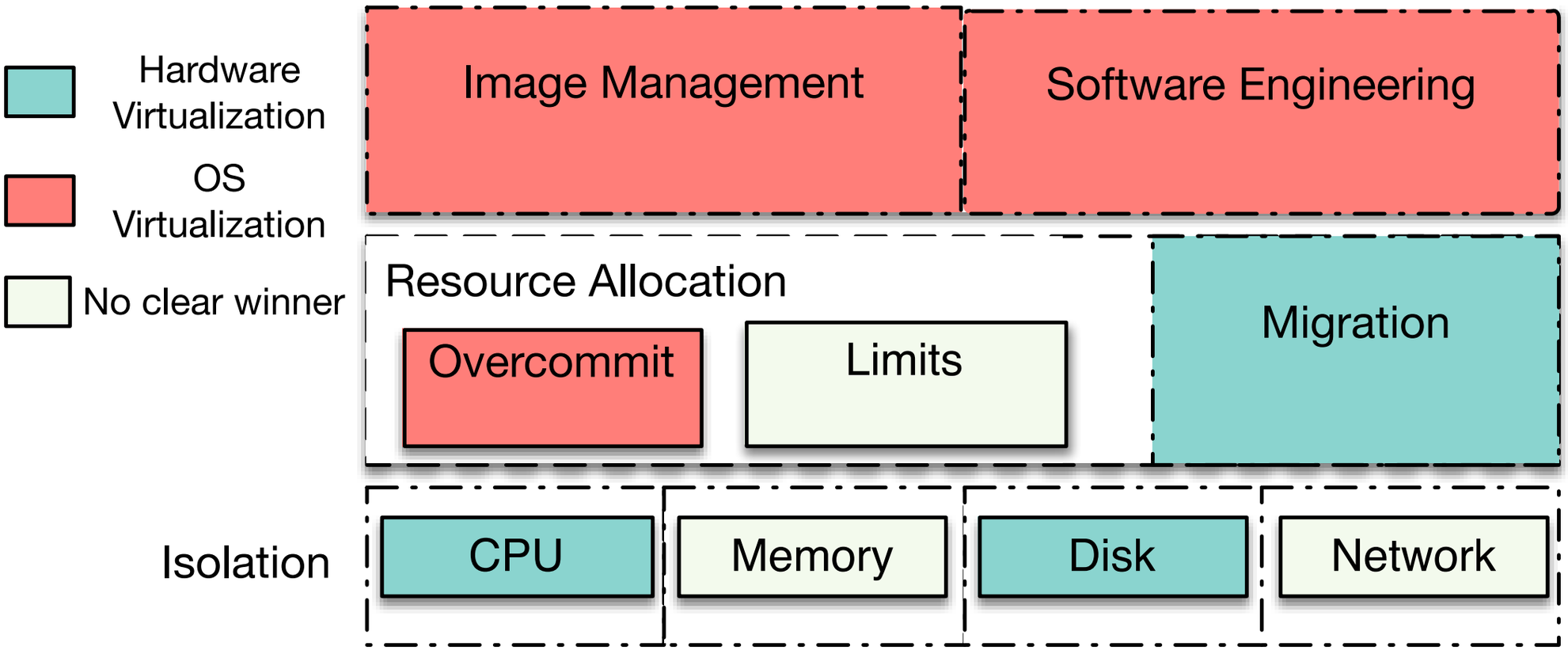
Container images have only application layer

VMs contain entire OS, and have larger images

Time (s)	VM (Vagrant)	Docker
MySQL	236	129
NodeJS	304	49

- Docker: 2-6x faster

Big Picture

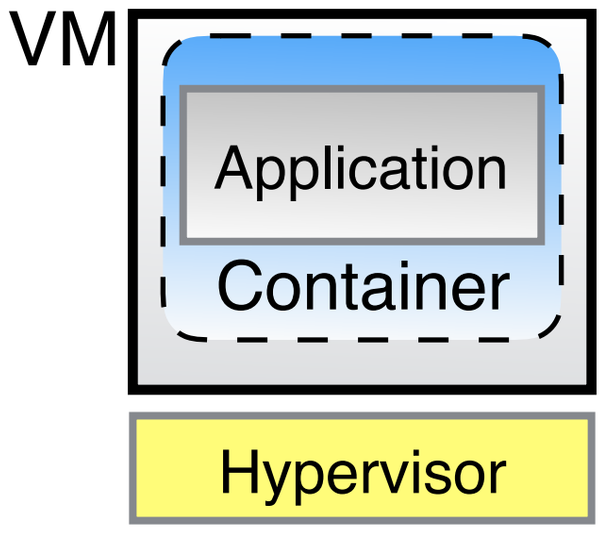


- Performance depends on workload and neighbors
  - Interference because of shared resources
- Different application management and deployment capabilities
- Neither is best for *all* use-cases and scenarios
- **Can we get the best of both worlds?**

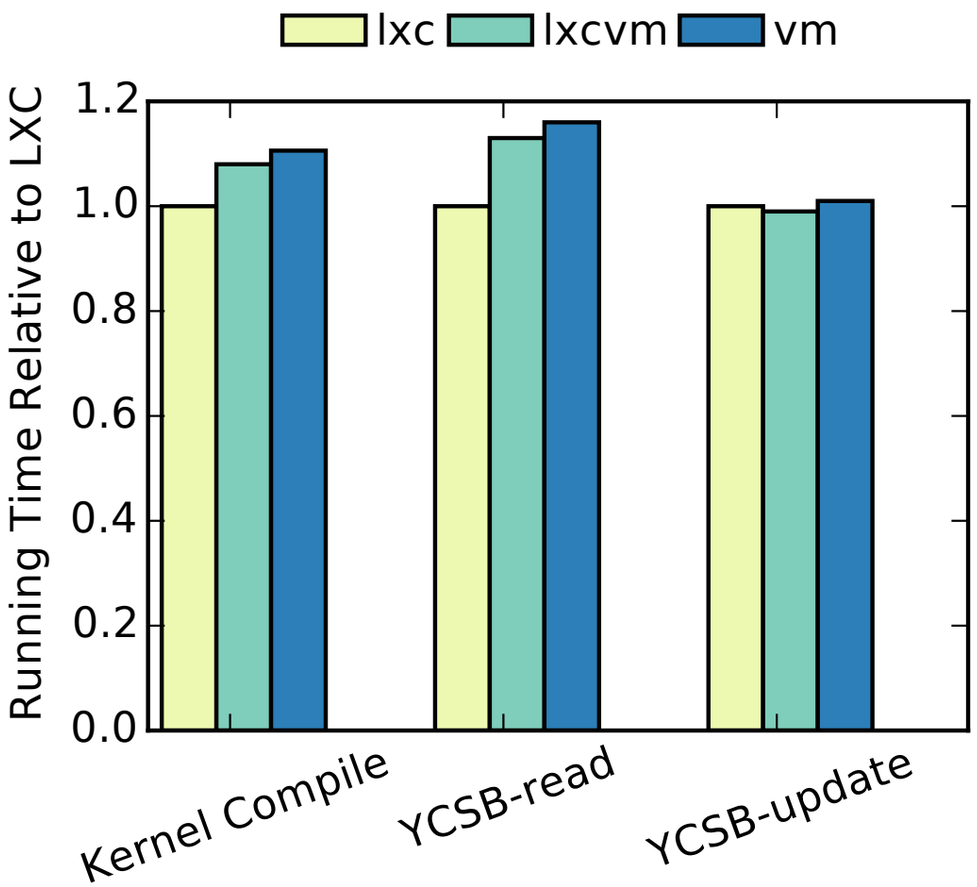
Outline

- Motivation and problem statement
- Virtualization background
- Performance comparison
- Managing and deploying applications
- **Hybrid virtualization approaches**
- Related work
- Conclusion

Containers Inside VMs



- Run containers inside VMs
- Isolation and security of VMs
- Image-management, provisioning of containers



- Performance comparable to VMs

Promising architecture which combines benefits of today's hardware and OS virtualization

Lightweight VMs

- Heavily optimized and stripped-down hardware VMs
- Make VMs more like containers
- **Isolation properties of VMs + low overhead of containers**
  - Intel ClearLinux, VMware Project Bonneville
- Directly access host file system. No virtual disk abstraction.

	Boot-time (s)
Container	0.3
ClearLinux	**0.8**
KVM	~10

- Fast boot-times enable VMs to be used in container pipelines
  - Quick deployment tests, rapid scaling,…

Related Work

- Felter et.al. — *Single* container vs. VM performance study
- Agarwal et.al. — Comparison of memory footprint
- Containers in HPC (Ruiz et.al) , data processing (Xavier et. al)
- Alternate virtualization architectures — Unikernels, NoHype (Keller et.al)

Conclusion

- Containers and VM represent two choices in system virtualization
- Different tradeoffs in performance isolation, deployment
- “Right platform” depends on workload, environment, and use-case
- Nested containers and lightweight VMs are two promising hybrid strategies
- **Fast changing landscape: change in performance and capabilities will require continuing evaluation**

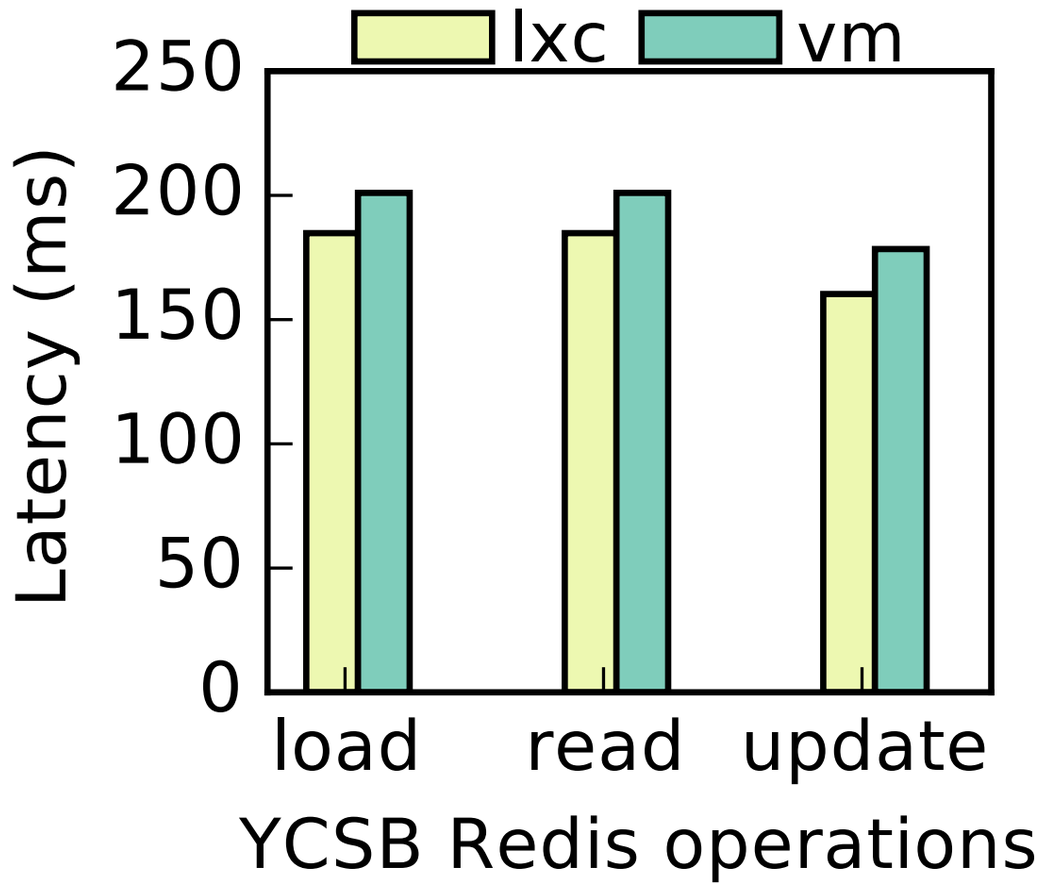
Thank You

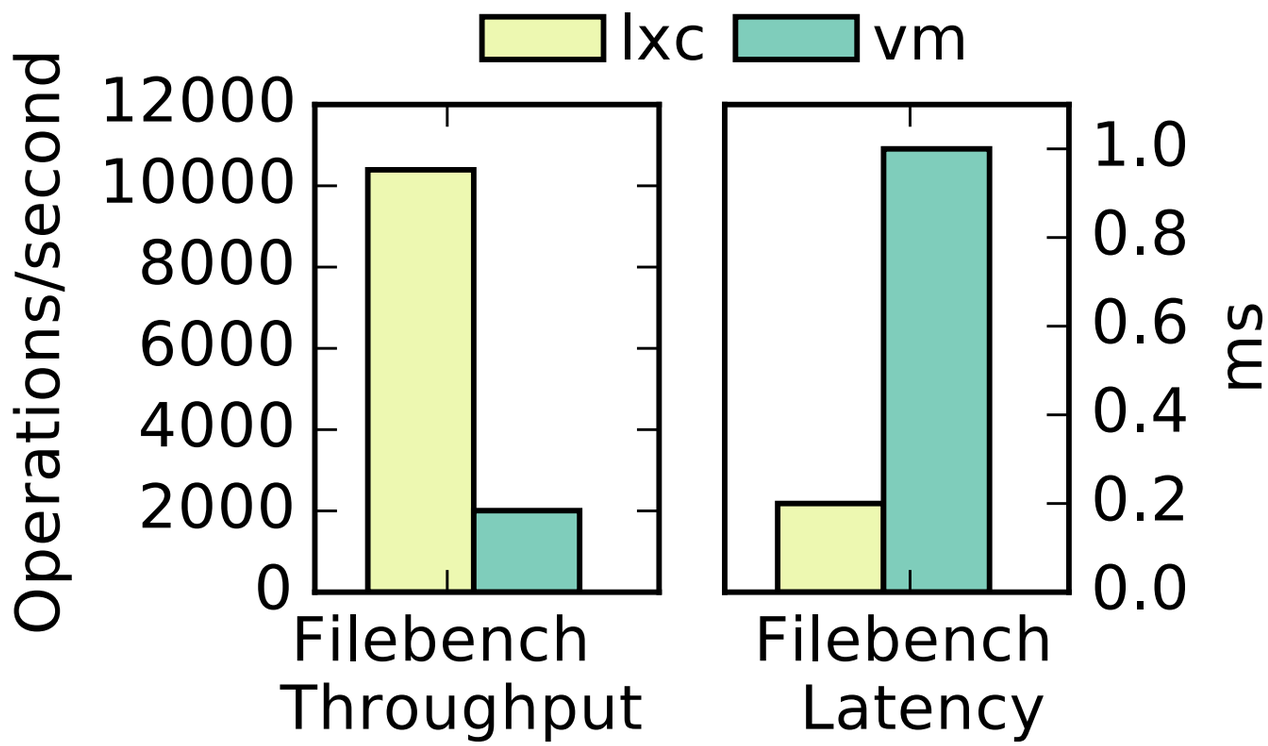
Questions?

http://cs.umass.edu/~prateeks

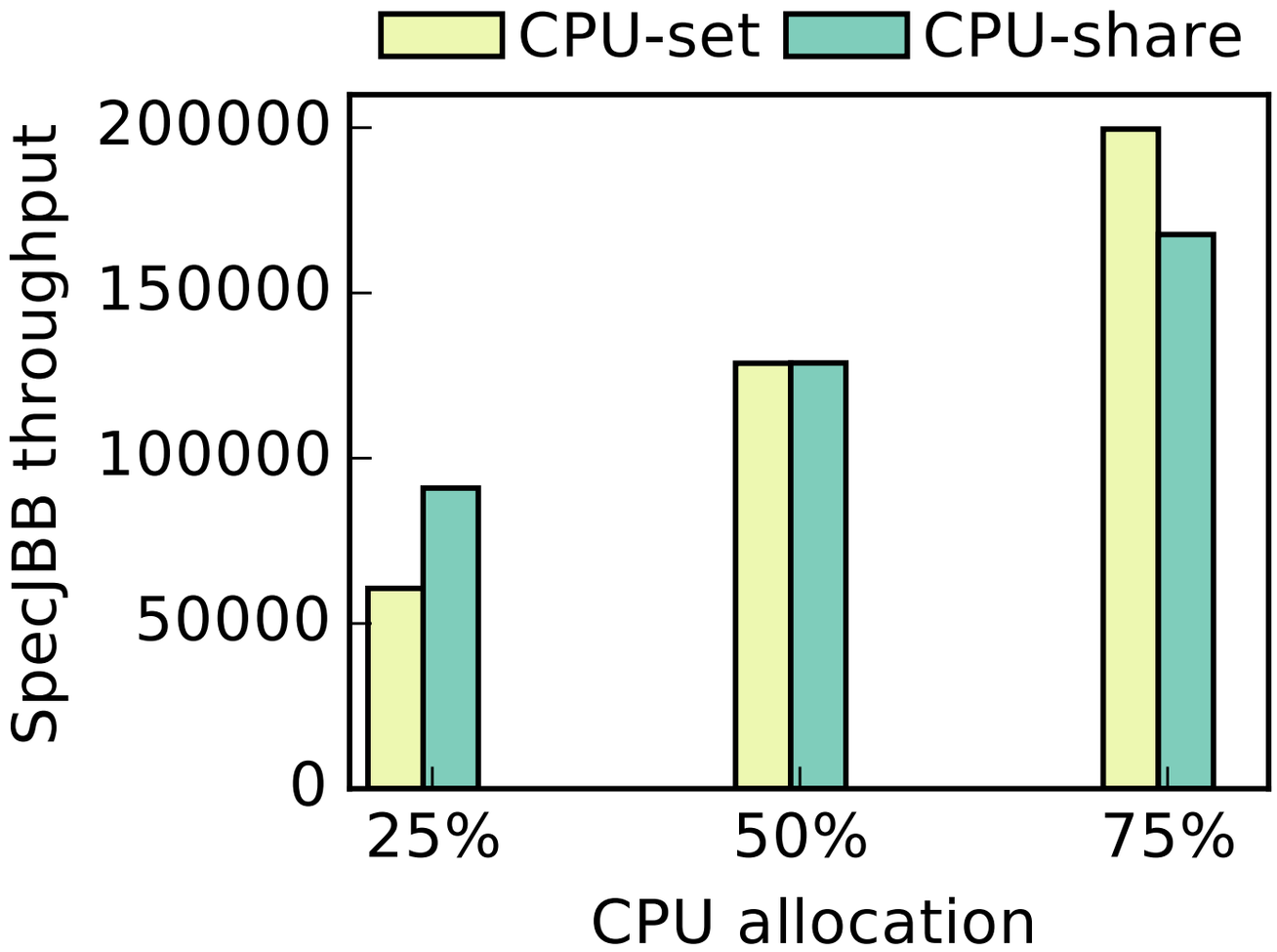
Backup Slides

Baseline

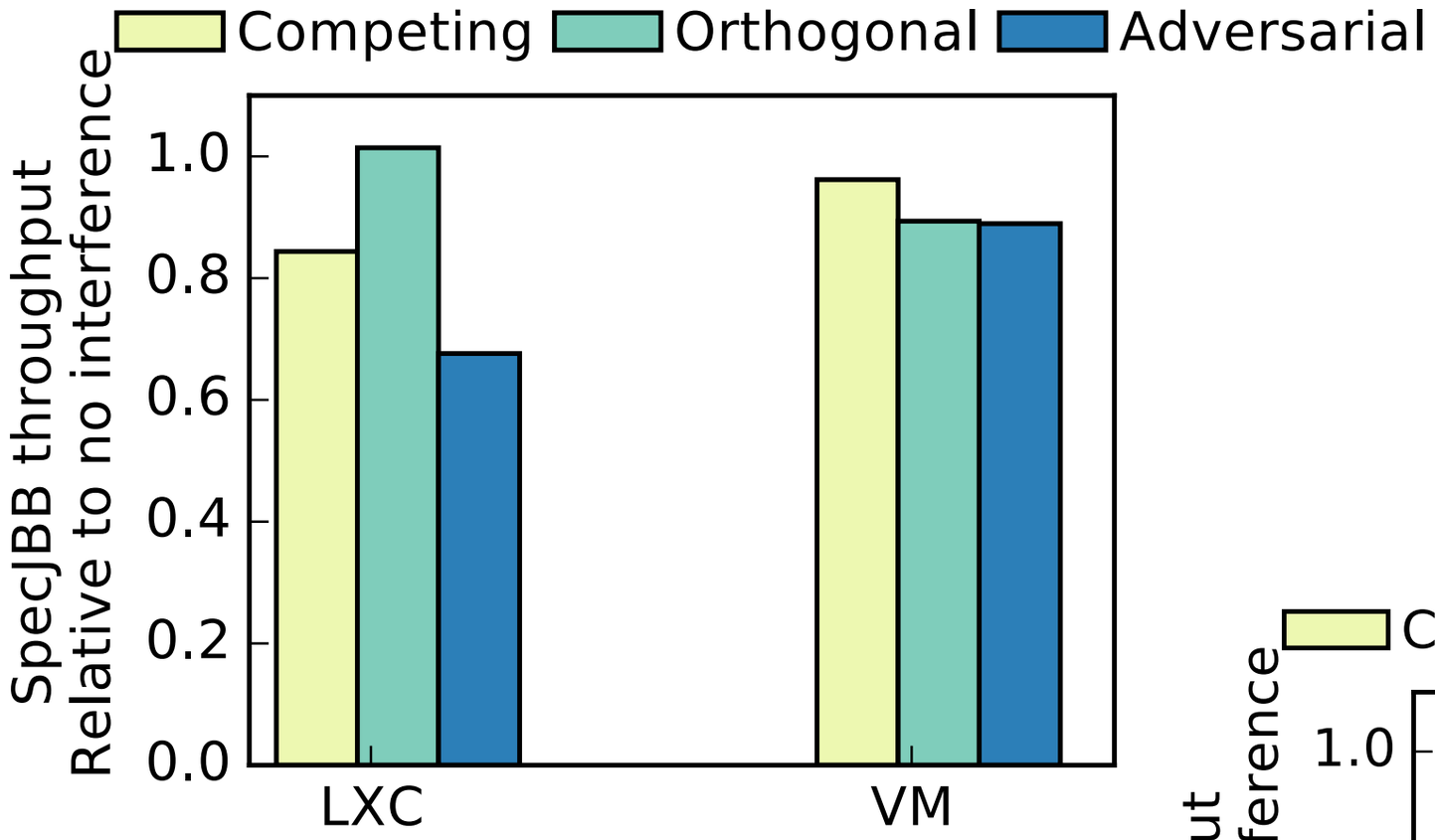

lxc
vm
250
200
150
100
50
0
Latency (ms)
load
read
update
YCSB Redis operations

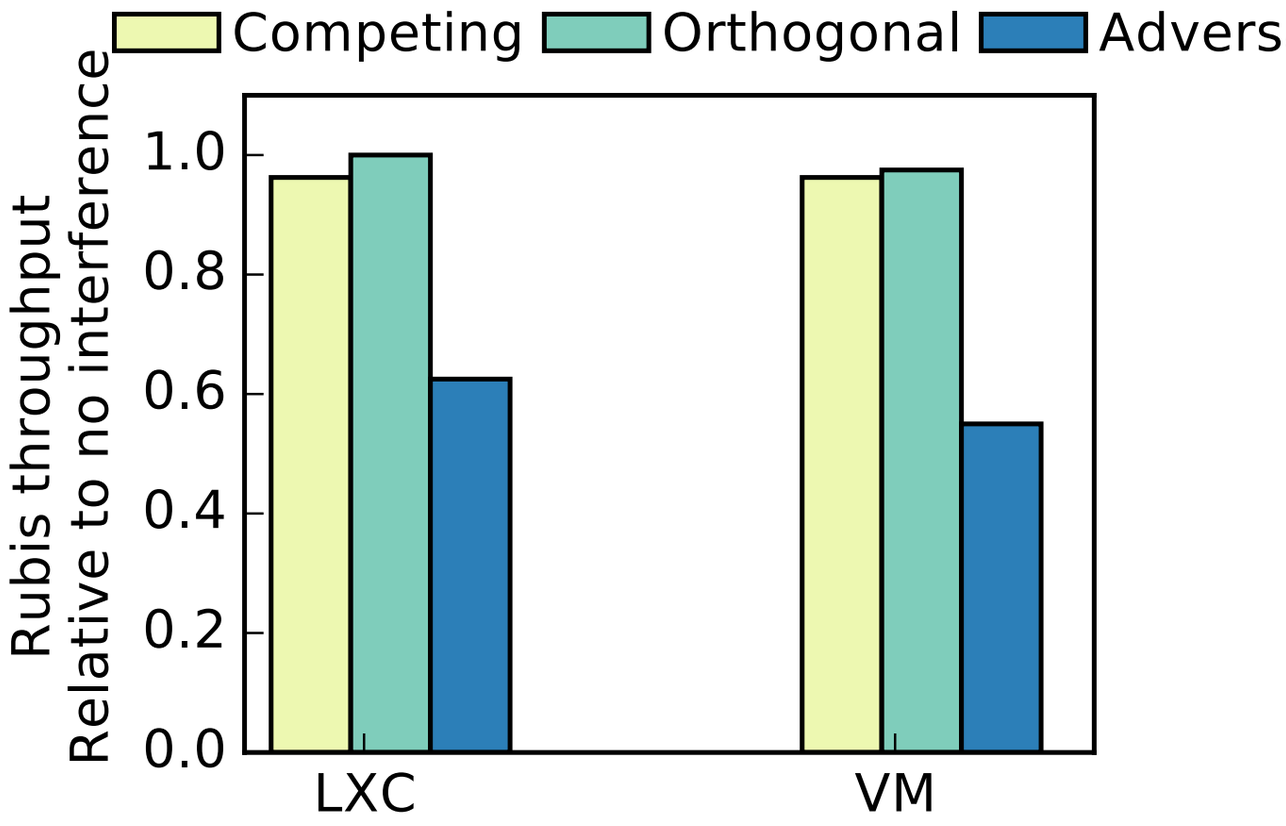

lxc
vm
12000
10000
8000
6000
4000
2000
0
Operations/second
Filebench
Throughput
1.0
0.8
0.6
0.4
0.2
0.0
ms
Filebench
Latency

CPU with different Cgroups setting



More Interference



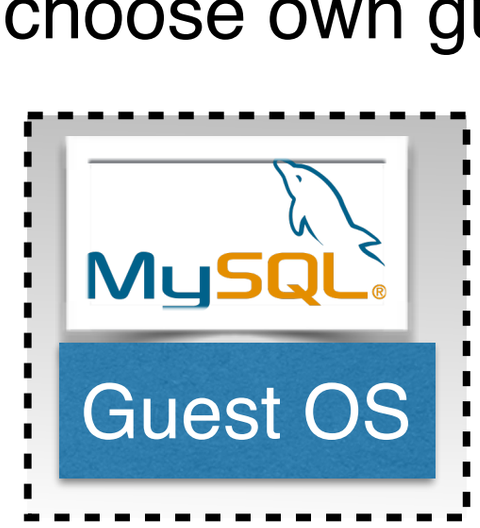


Today's Virtualization

Hardware Virtualization

- Applications run inside *Virtual Machines*
- Vmware, Xen, KVM,…
- Can choose own guest OS

VM



Operating System Virtualization

- Applications run inside *Containers*
- Docker, Jails, LXC,…
- Applications share the OS
- Lightweight virtualization

Container

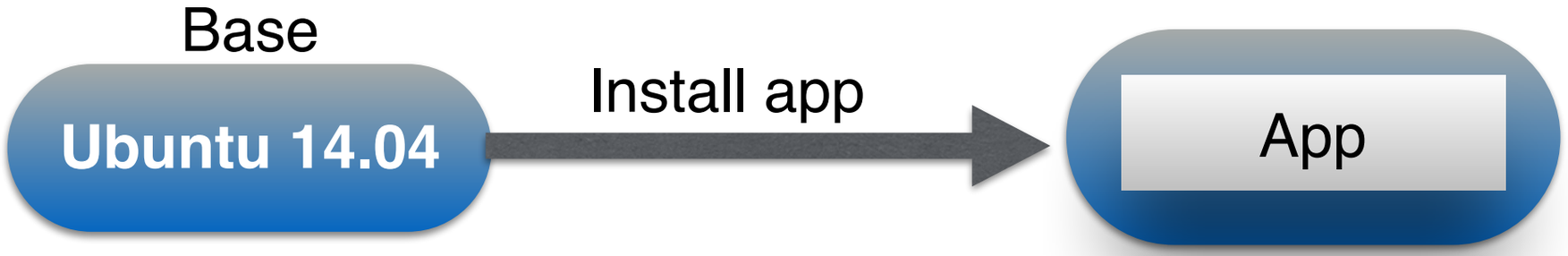
- Cluster-wide management tools help users deploy applications:
  - Placement of applications
  - Horizontal scaling

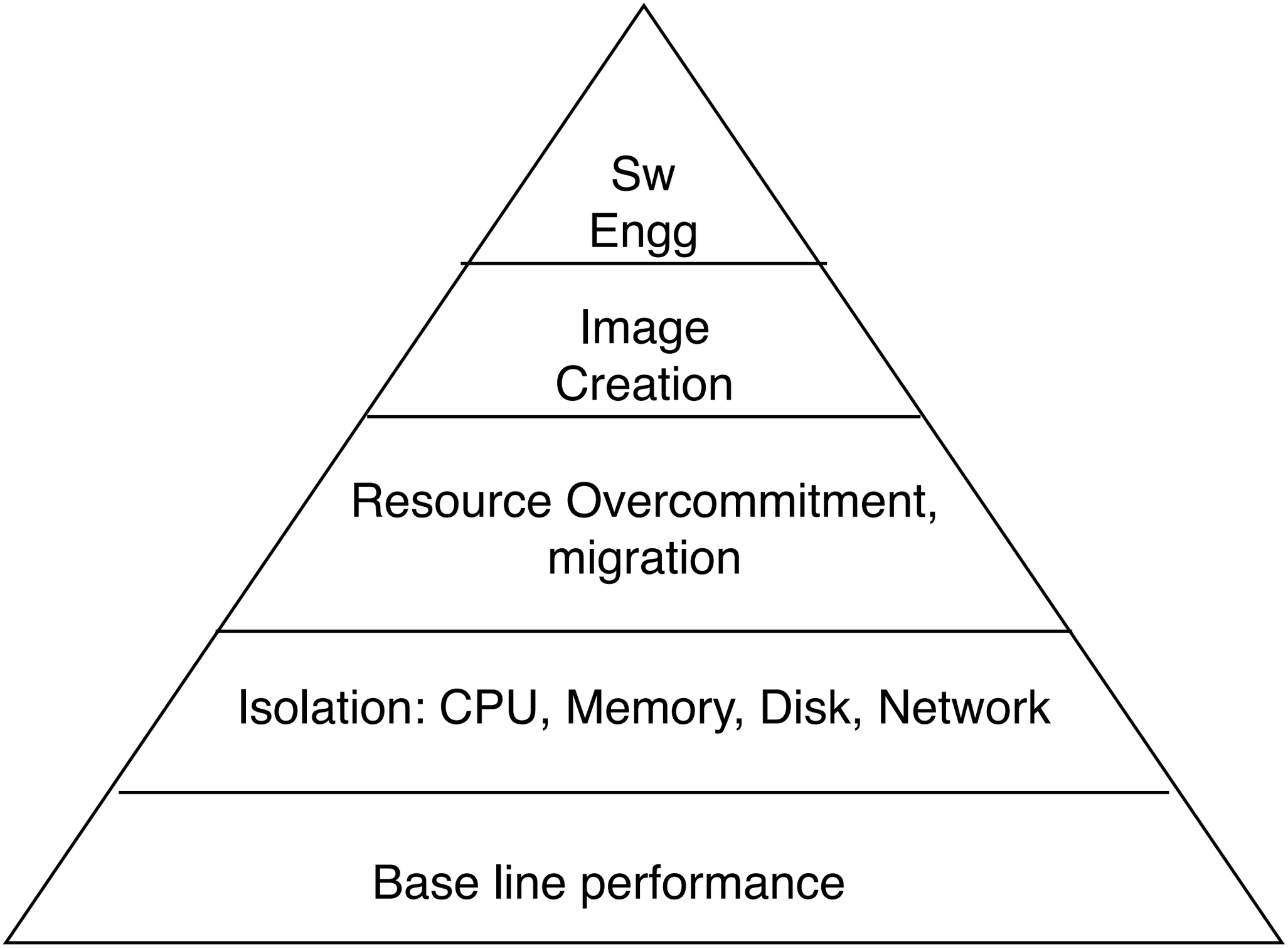
Application Deployment

Image size	VM	LXC	Docker
MySQL	1.68 GB	0.4 GB	**112 KB**
NodeJS	2.05 GB	0.6 GB	**72 KB**

Docker: 2-6x smaller

- VMs contain entire OS, and have larger images
- Docker stores only differences (application layer)




Sw
Engg
Image
Creation
Resource Overcommitment,
migration
Isolation: CPU, Memory, Disk, Network
Base line performance