Variability in Operating Systems

Brian Kocoloski

Assistant Professor in CSE Dept.

October 8, 2018

Cloud Computing

aws

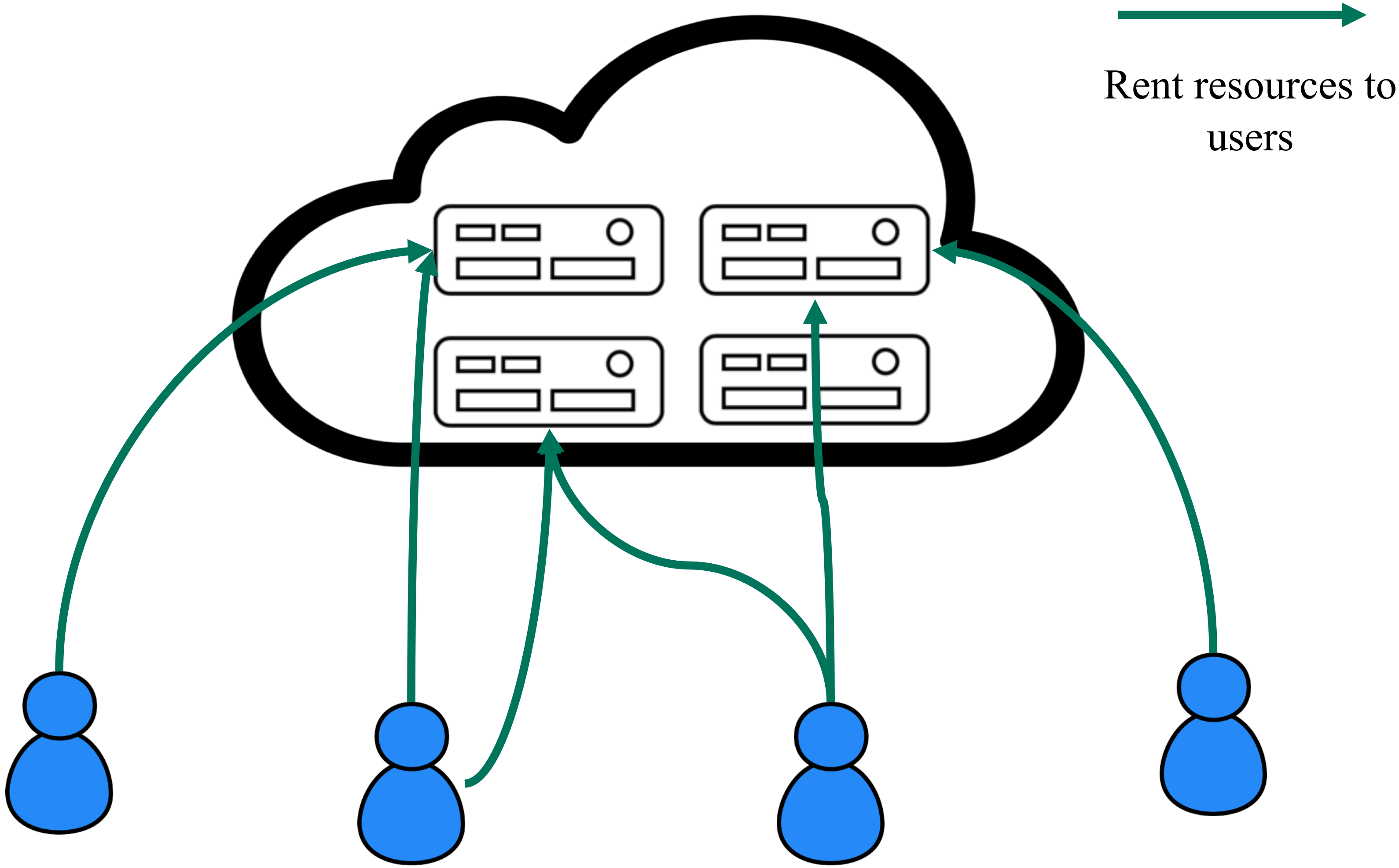
Google Cloud Platform

Azure

Current estimate is that 94% of all computation will be performed "in the cloud" by 2021

https://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.html

CLOUD COMPUTING



Why is Cloud Computing Attractive

- From the user's perspective
  - Don't need to purchase own machines
  - Don't need to maintain infrastructure
    - Power/cooling/maintenance
    - Lower IT costs
  - Dynamically scale resources based on need
    - e.g., webserver with "bursty" traffic can dynamically scale up its virtual server capacity

- From the cloud provider's perspective
  - Can consolidate multiple users on same underlying infrastructure
  - Resource sharing increases revenue

SUPERCOMPUTING

Aurora supercomputer expected in 2021

- First "exascale" machine in the United States
  - Likely more than 50K server nodes
  - Likely more than 1M cores

- Capable of one billion billion floating point calculations per second

The Need for Predictability

- Some applications struggle to make use of the vast resources of clouds and supercomputing systems

- Latency sensitive cloud applications
  - Paper from Google: Dean and Barroso. “The tail at scale”, CACM 56(2), 2013

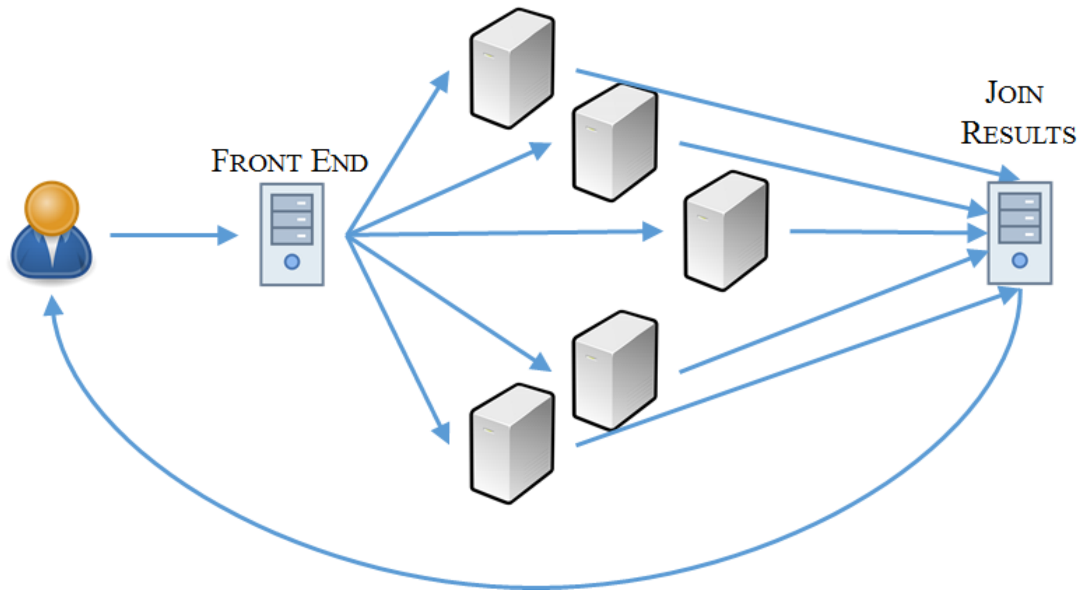
- Bulk synchronous applications
  - Common with HPC, machine learning, graph analytics

- Real-time computing workloads

PROBLEMS IN THE CLOUD

<u>*The tail at Scale*</u>

[Dean and Barroso, CACM 56(2), 74-80, 2015]



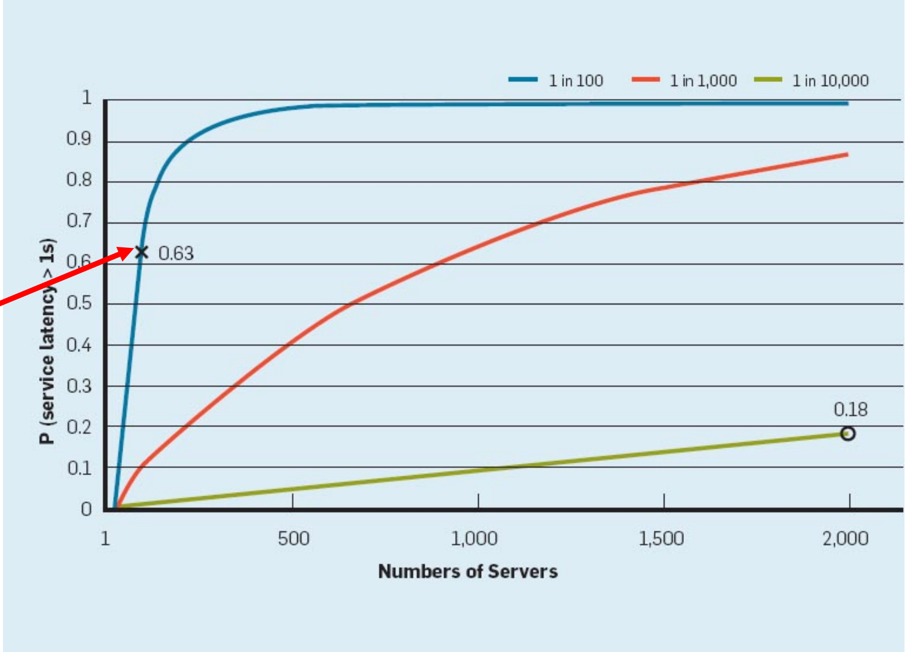
Each  incurs some latency with some probability

The total latency is a function of the slowest 

PROBLEMS IN THE CLOUD

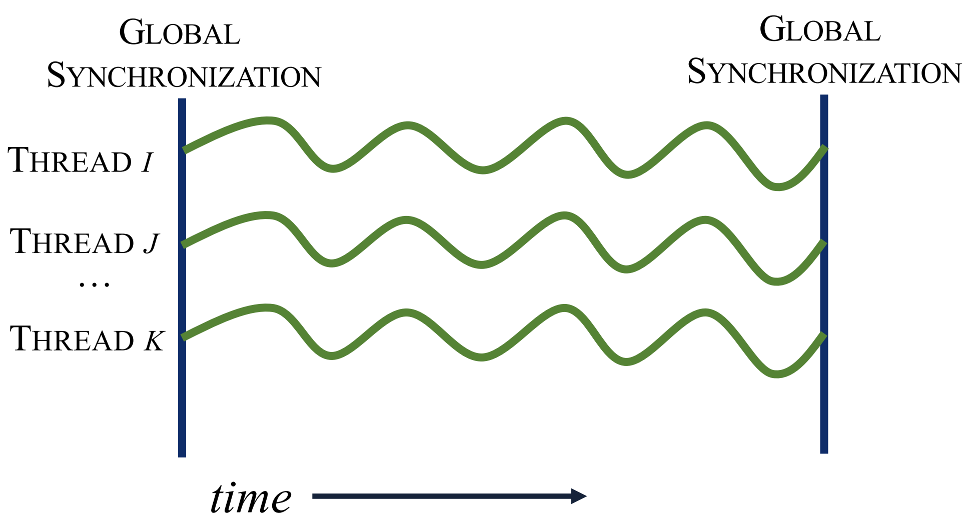
- When user requests require many individual components, the probability of an overall service slowdown increases
  - **Longest latency dictates overall service performance**

- ➢ Assume 100  needed to handle a user request
- ➢ P(one server slow) = 1%
- ➢ P(overall slowdown)
  = 1 – (.99^100) ~ 63%
- ➢ **63% of all services are slowed by the 1/100 outliers**



SPATIAL VARIABILITY

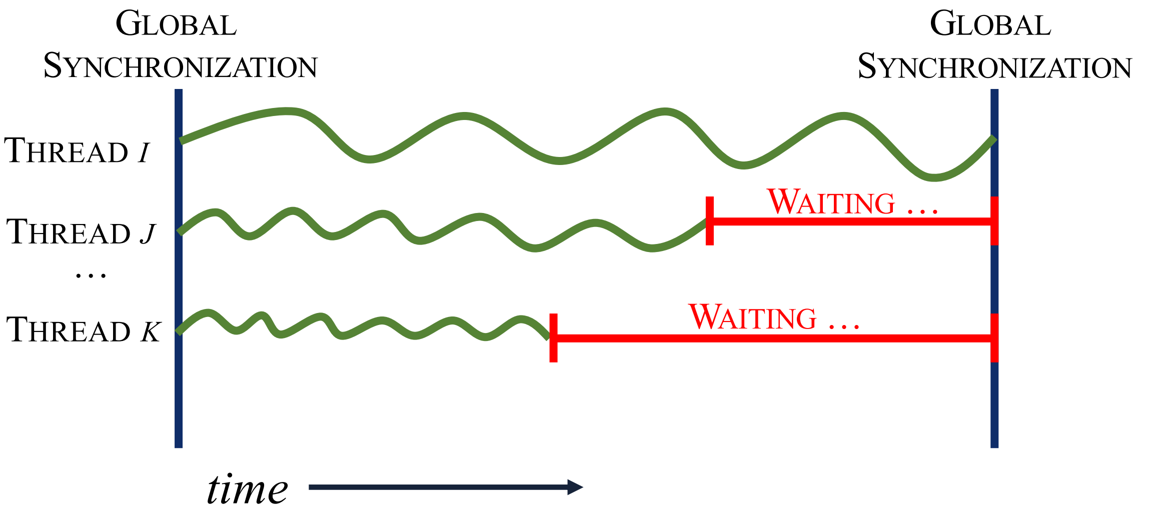
Without variability, all threads make equal progress in equal time



SPATIAL VARIABILITY

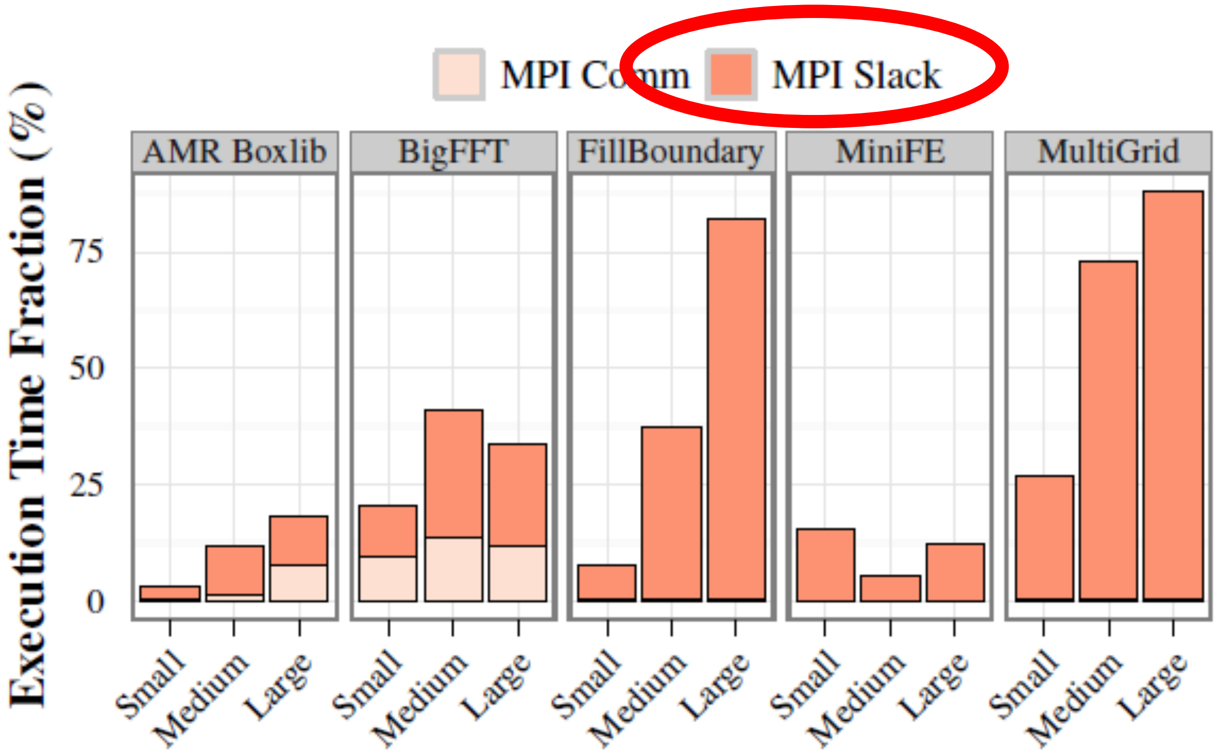
With variability, *some threads progress slower than others*

Variability slows *global synchronization*
(extends runtime, wastes power, wastes energy)



PROBLEMS IN BSP APPLICATIONS

- Variability is a major challenge for tightly synchronized applications



http://portal.nersc.gov/project/CAL/designforward.htm

- Over 75% of execution time spent blocked on global synchronization
- Up to 90% of cpu dynamic power consumption wasted

Dealing with Variability

Takeaways

- Outliers are important
  - *"Techniques that concentrate on these slow outliers can yield dramatic reductions in overall service performance" (Dean and Barosso)*
- Removing variability at small scale translates to significant gains at large scale
  - 5% improvement in small scale performance is significant
- Improving the worst case is more important than improving the average case
  - Metrics: at small scale, standard deviation is at least as important as mean

Overview of My Research

1. Hobbes: a new operating system designed to enable predictable performance via performance isolation

2. Analysis of low-level OS variability present in software technologies used in the cloud

Overview of My Research

1. Hobbes: a new operating system designed to enable predictable performance via performance isolation

2. Analysis of low-level OS variability present in software technologies used in the cloud

LIGHTWEIGHT KERNELS

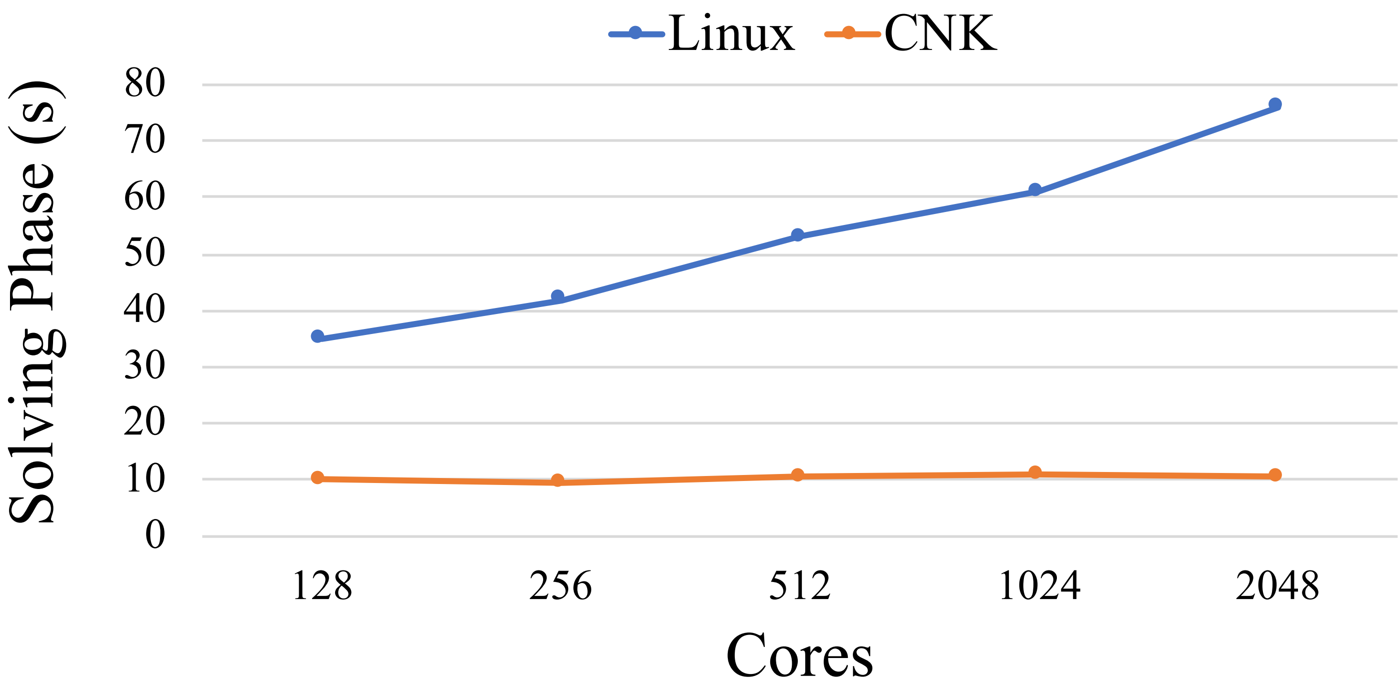
- Operating systems designed specifically for supercomputers
  - Give application direct control of hardware
  - Simplified algorithms for scheduling + memory mgmt
  - Primary goal: consistent, predictable performance

- Long history of scalability on supercomputers

Sandia National Laboratories

Kitten, Sandia's most recent lightweight kernel

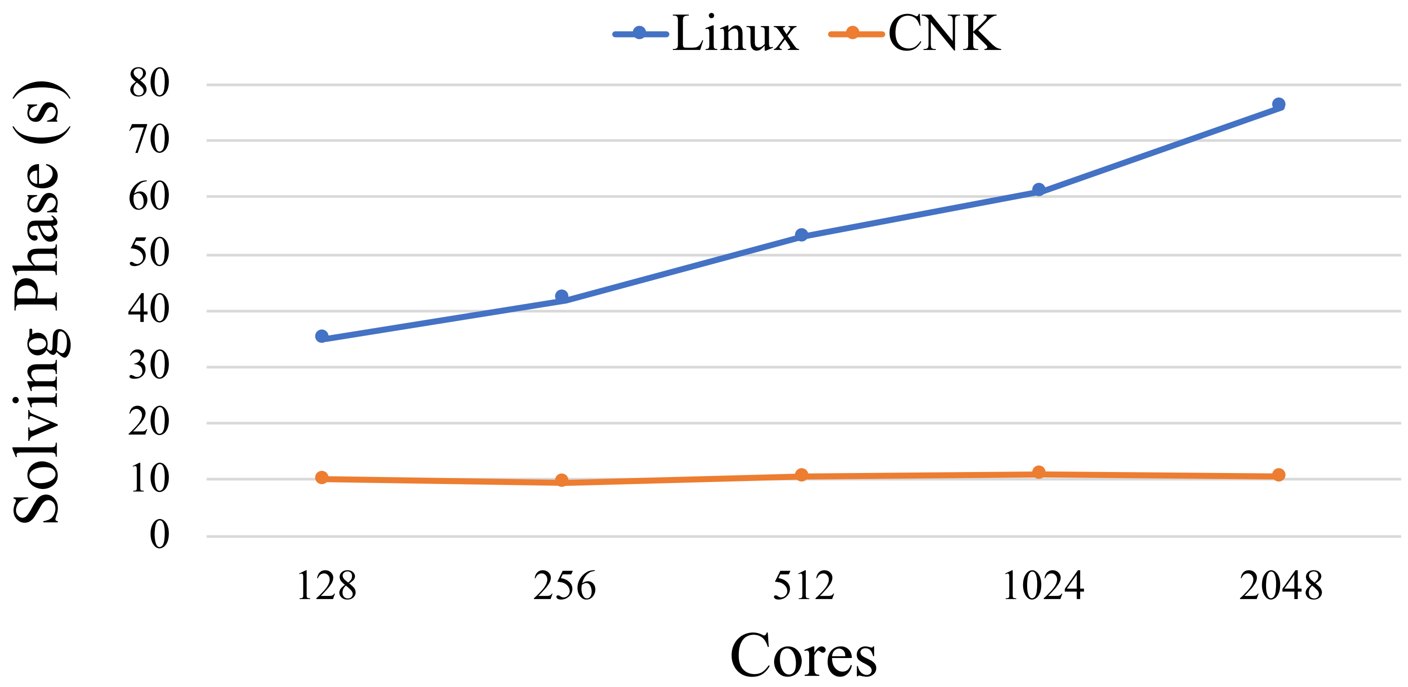
OS Comparison on IBM Blue Gene/P



Adaptive MultiGrid on IBM BG/P

Morari et. al, *IPDPS 2012*

OS Comparison on IBM Blue Gene/P



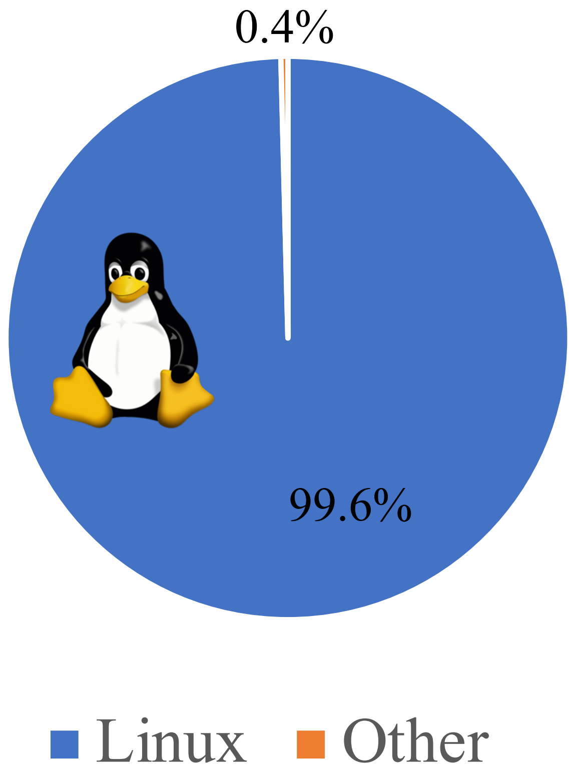
Adaptive MultiGrid on IBM BG/P
Morari et. al, *IPDPS 2012*

So lightweight kernels are used on all large scale computers, right?

LINUX IS NECESSARY

- Performance is not the only consideration
- Technical reasons
  - Huge suite of device drivers, network stacks, file systems, etc.
- Non-technical reasons
  - Familiar development environment
  - Ease of programmability
  - Lots of system calls

Operating System Share of Top500 (Nov. 2016)



https://www.top500.org/

The Hobbes Exascale OS/R

- Started as Department of Energy exascale OS and runtime project
  - http://xstack.sandia.gov/hobbes/

- Vision: we need to support application composition (e.g., simulation + analysis + visualization)

- My work: dynamic runtime reconfiguration of the operating system

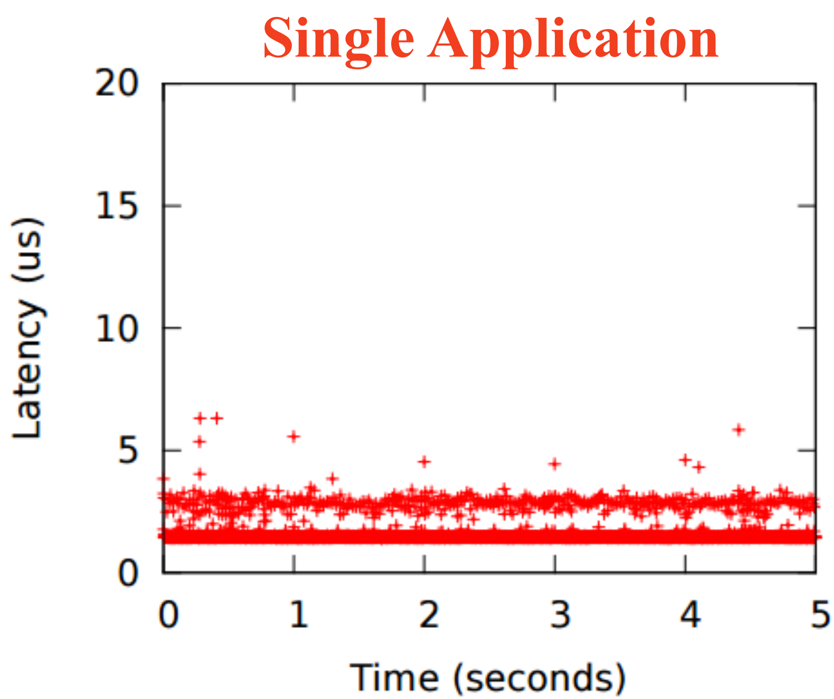
Performance Isolation

Handling complex workload mixes across different users is necessary in clouds and HPC systems

Common in cloud systems (multi-tenancy)

Becoming more common in supercomputers as well

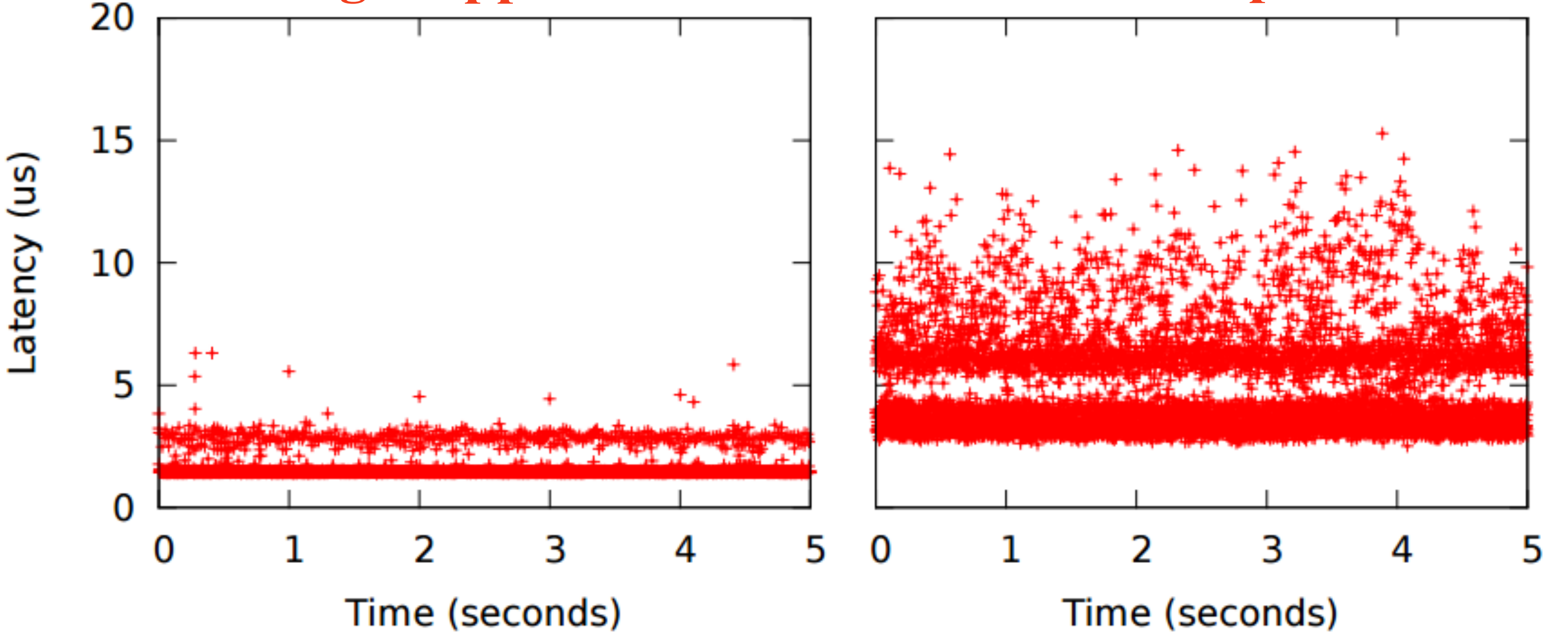
Kernel Interference (Linux)



Each point represents the latency of an OS interruption

Kernel Interference (Linux)

Single Application **With Competition**



Why Does This Happen?

- Linux is a commodity OS that generally does not care about extreme scale features
  - Cares about running anywhere and everywhere
  - No understanding of how this impacts massive scale applications

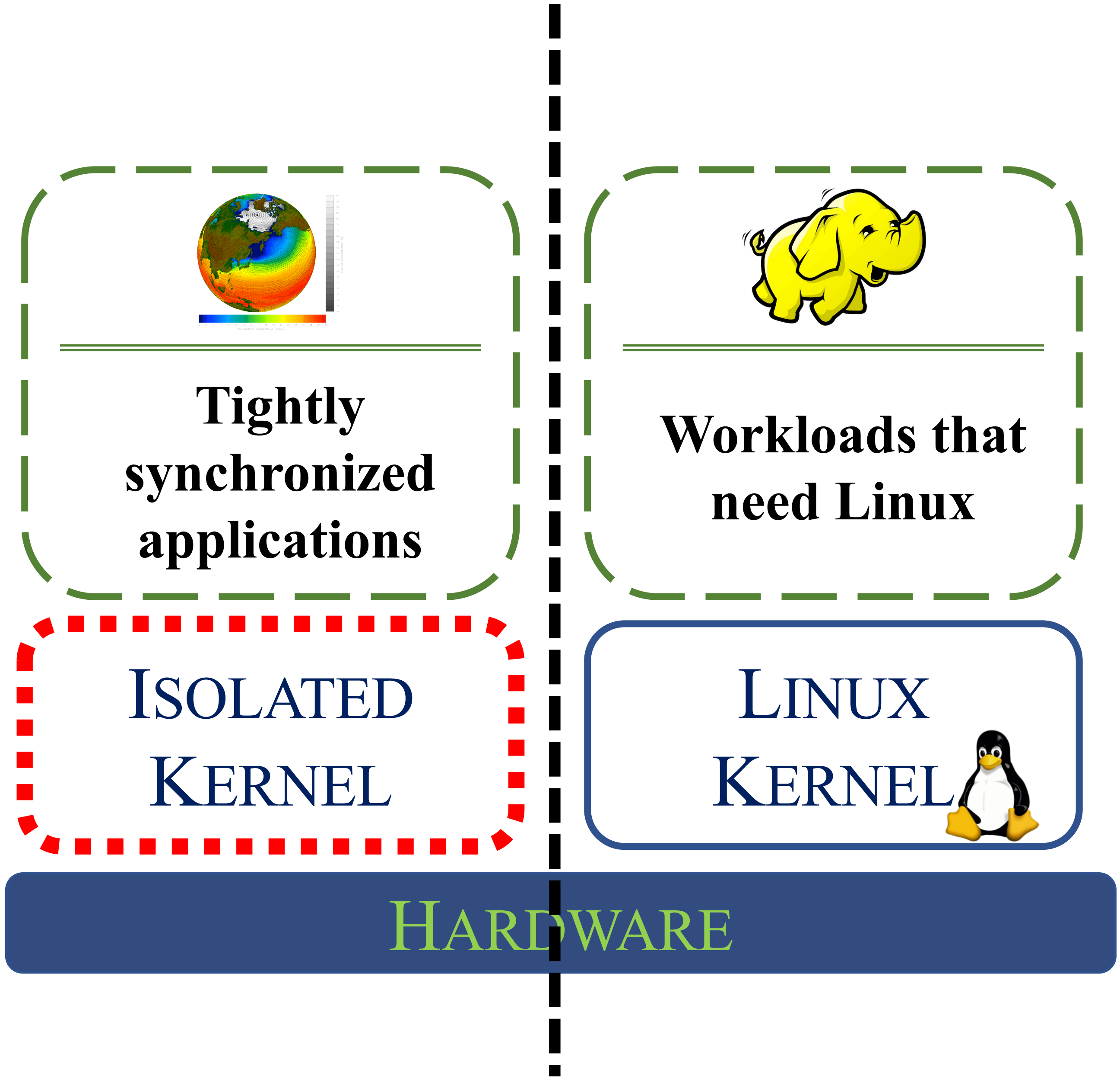
- Our novel insight: *OS resources* generate variability
  - B. Kocoloski, J. Ouyang, and J. Lange, **"A Case for Dual Stack Virtualization: Consolidating HPC and Commodity Applications in the Cloud,"** *SOCC '12*
  - B. Kocoloski and J. Lange, **"HPMMAP: Lightweight Memory Management for Commodity Operating Systems,"** *IPDPS '14*
  - B. Kocoloski and J. Lange, "**Lightweight Memory Management for High Performance Applications in Consolidated Environments,"** *TPDS '16*

- Page table locks, page caches, scheduling queues all examples of contended OS resources

Target

Performance isolation

between applications at the OS level


Tightly synchronized applications
Workloads that need Linux
ISOLATED KERNEL
LINUX KERNEL
HARDWARE

KITTEN LIGHTWEIGHT KERNEL

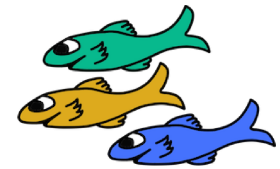
- Lightweight kernel (LWK) from Sandia National Laboratories designed to execute massively parallel HPC applications

- Major design goal: provide more repeatable performance than general purpose OS (like Linux) for tightly synchronized workloads

- Simplified, lightweight resource management

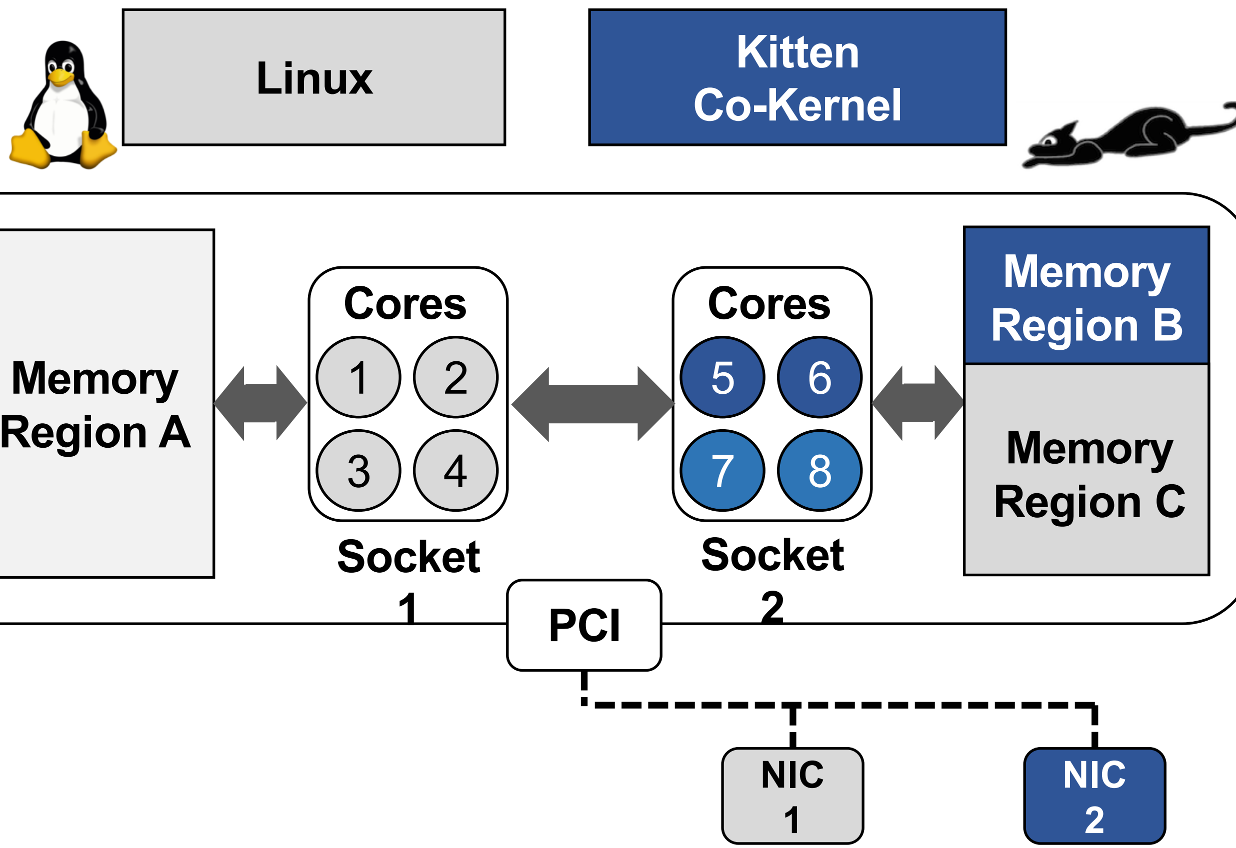
https://software.sandia.gov/trac/kitten

PISCES CO-KERNELS

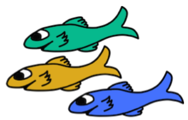


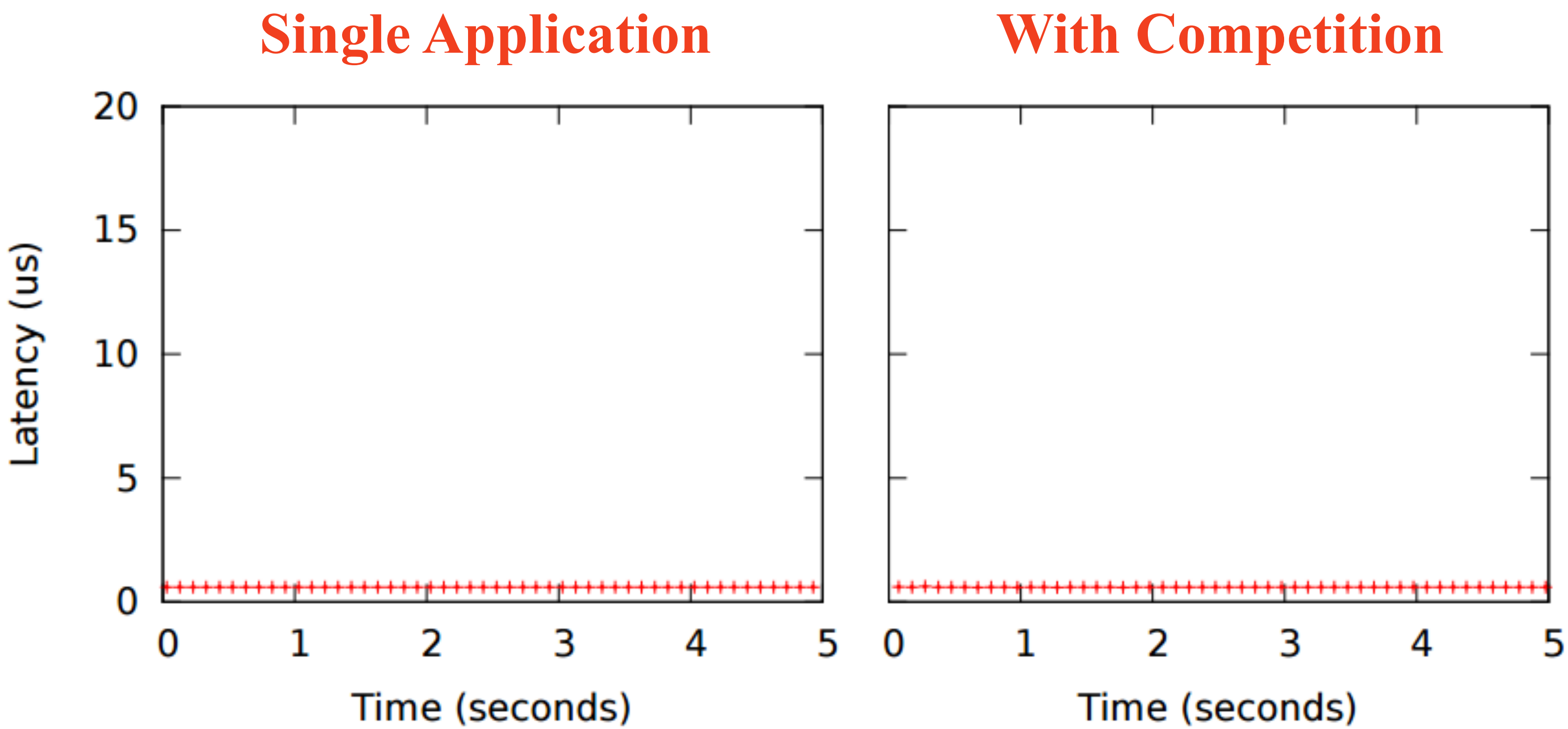
- We designed a co-kernel framework to boot multiple lightweight operating systems "next to Linux"
  - J. Ouyang, B. Kocoloski, J. Lange, K. Pedretti "**Achieving Performance Isolation with Lightweight Co-Kernels**," *HPDC '15*
  - B. Kocoloski et al., **"System-Level Support for Composition of Applications,"** *ROSS '15*
- *Complete isolation between separate OS kernels*
- Each OS runs its own scheduler, memory manager, network stacks, device drivers, etc.
- Hardware partitioned at *runtime* using Linux resource offlining utilities

Approach: Partition + Isolate

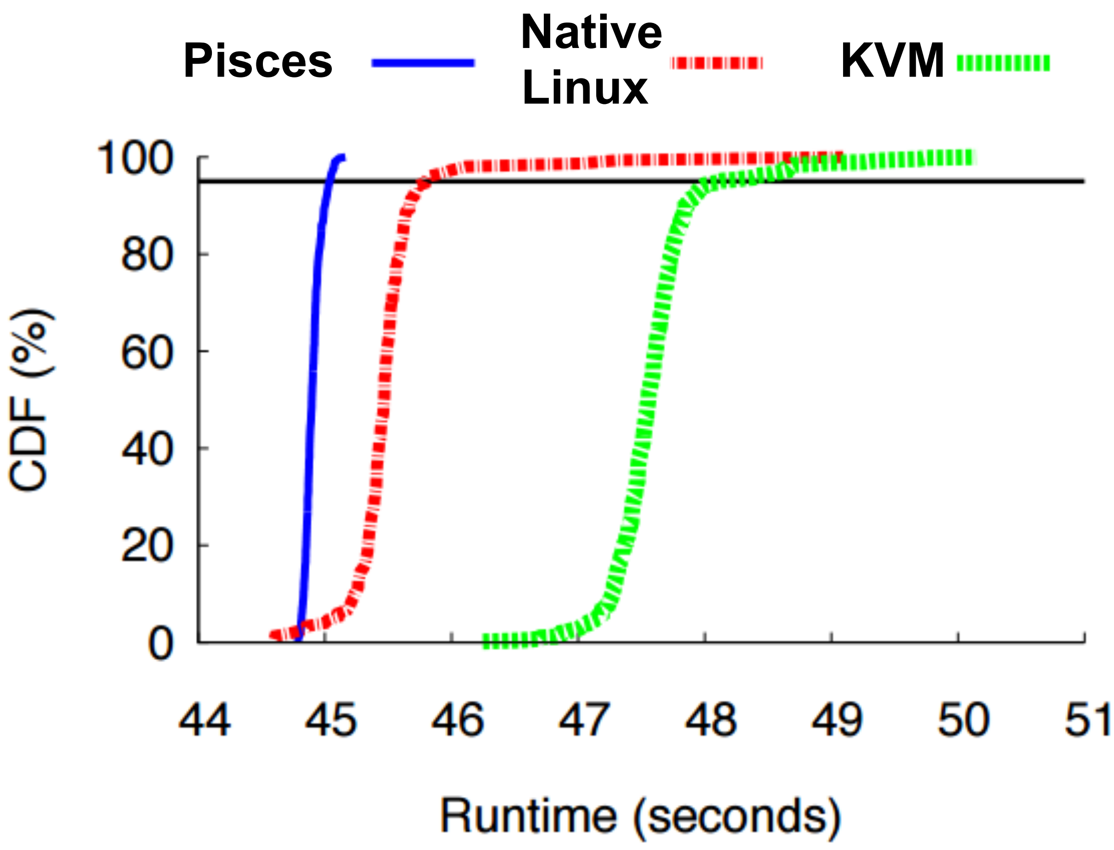


Kernel Interference (Pisces + Kitten)





ELIMINATION OF OUTLIERS (HPCCG)



- A few percentage points on average is nice …
- But *removal of outliers is critical* to achieve scalability

Overview of My Research

1. Hobbes: a new operating system designed to enable predictable performance via performance isolation

2. Analysis of low-level OS variability present in software technologies used in the cloud

What is going on in the kernel?

- Motivation: let's try to understand more specifically what is going on in the kernel that generates variability

- This is a problem outside of just BSP
  - Hard real-time applications (e.g., control system in nuclear power plant)
  - Cyber-physical systems, esp. with real-time components (e.g., real-time vision processing for autonomous vehicles)
  - Latency-sensitive cloud applications (tail at scale)

HIGH LEVEL PROBLEM: WORST CASE != AVERAGE CASE

- Dependence on worst-case performance is what unifies these workloads
- Problem: almost all computational platforms rely on the Linux kernel, which is (generally) not designed with worst-case performance characteristics in mind
- Competition: workloads compete for each other for resources; the focus here is on understanding how a shared OS kernel could be subject to competition

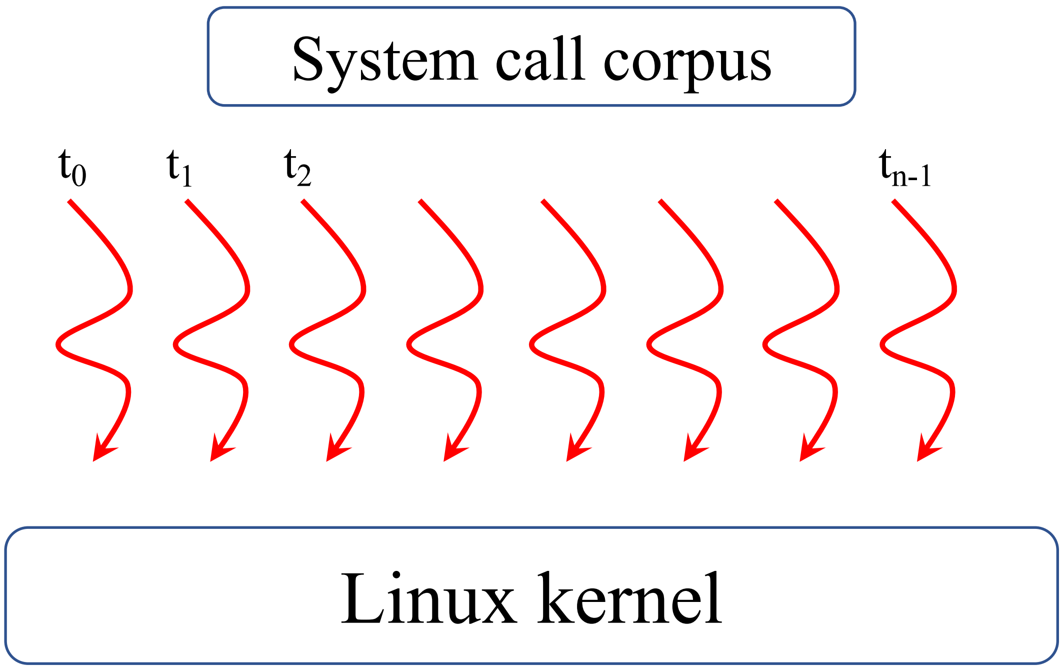
METHODOLOGY

Each thread does nothing but issue system calls to the kernel

- Higher levels of parallelism stress the ability of the kernel to isolate workloads from each other

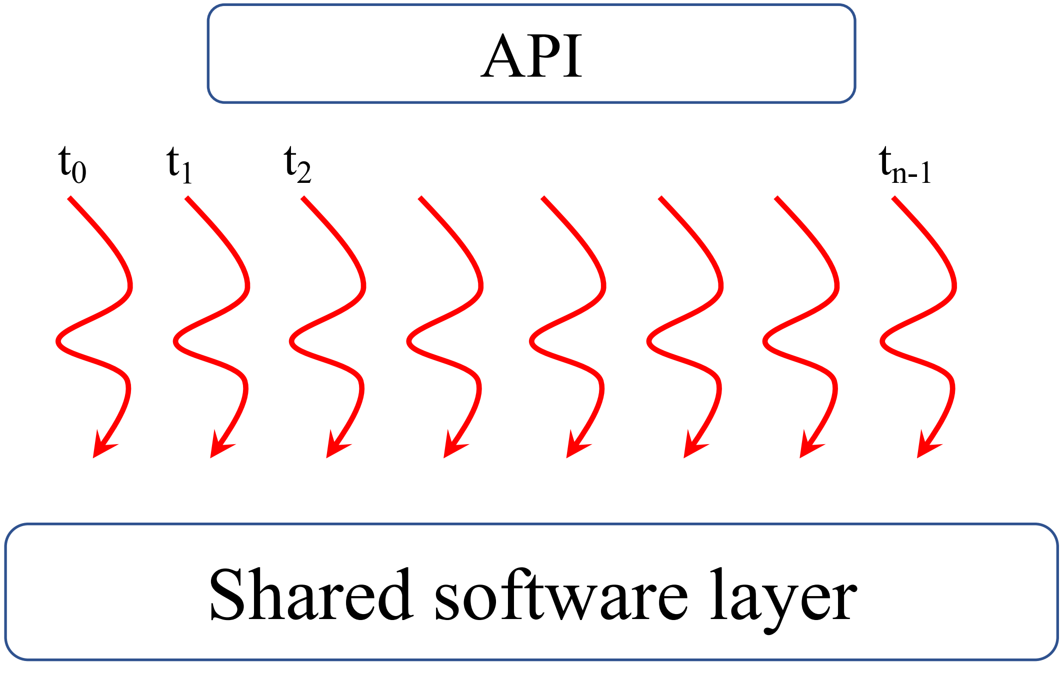
Workload is not hardware intensive – it relies almost exclusively on software efficiency

- Locks on data structures
- Software caches (e.g. page cache, SLAB allocator)

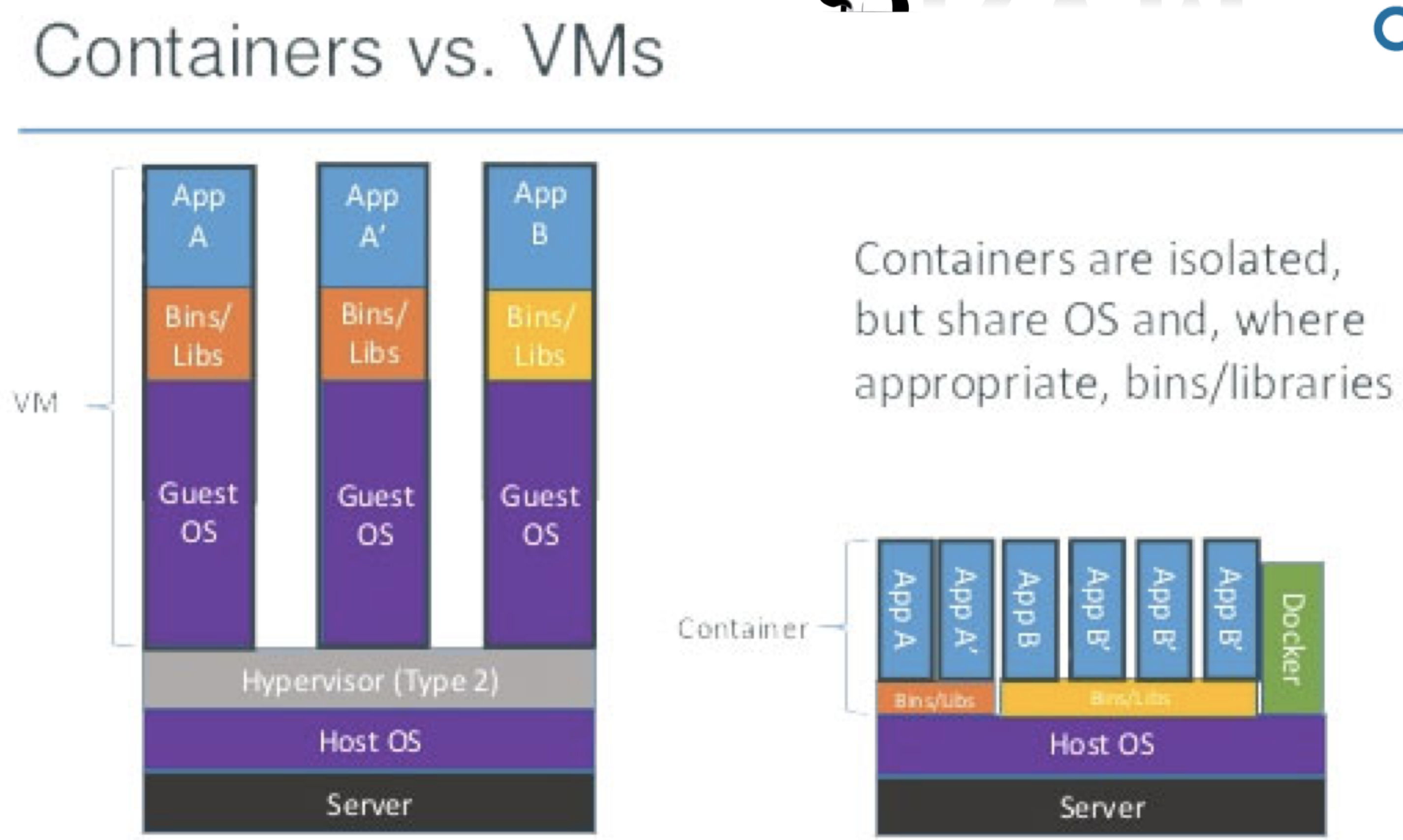


DEPLOYING SOFTWARE IN THE CLOUD

- Beyond understanding kernel variability, we can extend this framework to study variability that arises from concurrent contention to ***any*** shared software layer



Containers and VMs



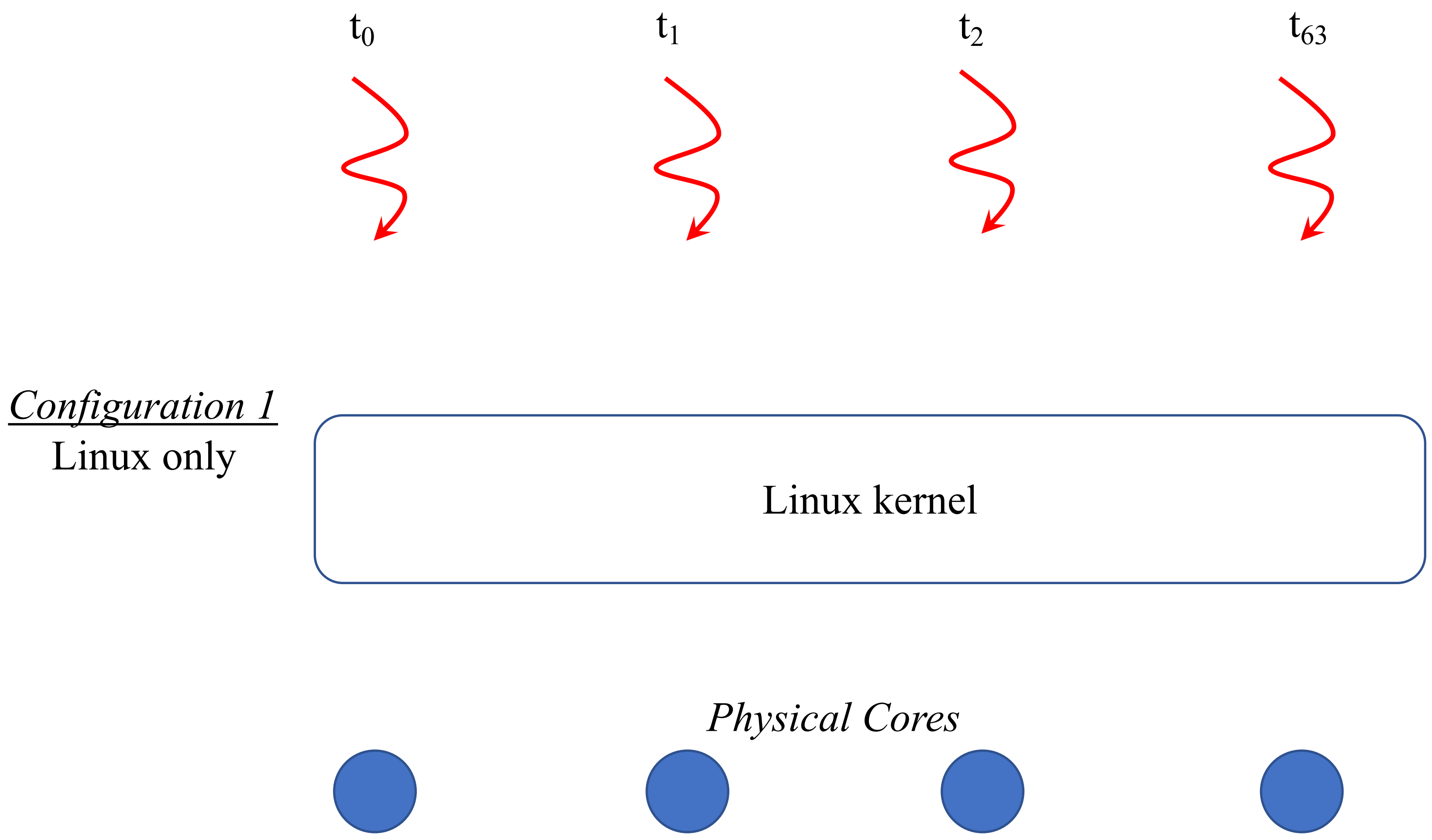
EXPERIMENTAL SETUP

- 64-core machine

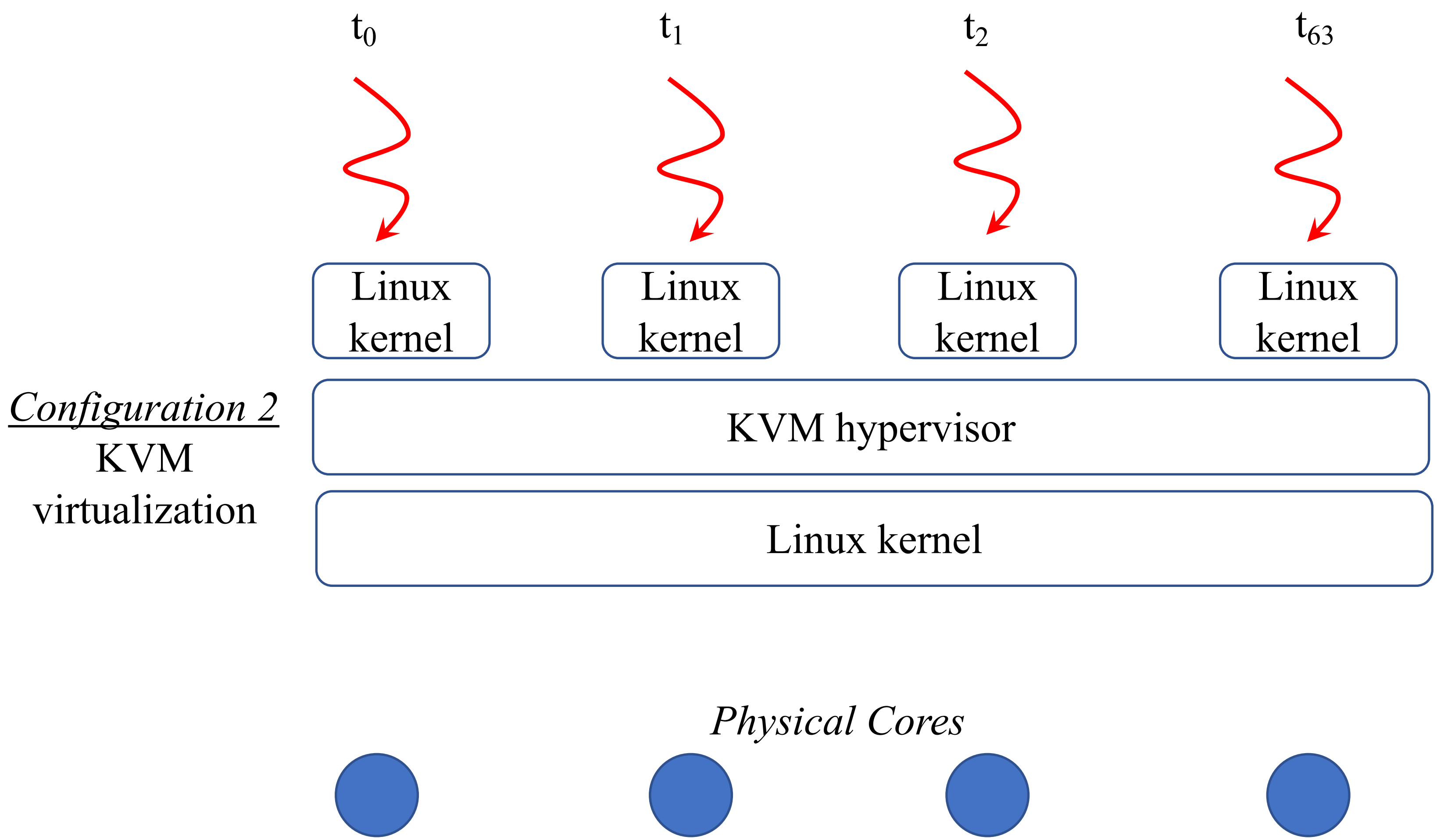
- Each core executes a set of 3,000 + system calls concurrently with every other core

- Three configurations:
  - 64 native Linux processes
  - 64 1-core virtual machines
  - 64 1-core containers

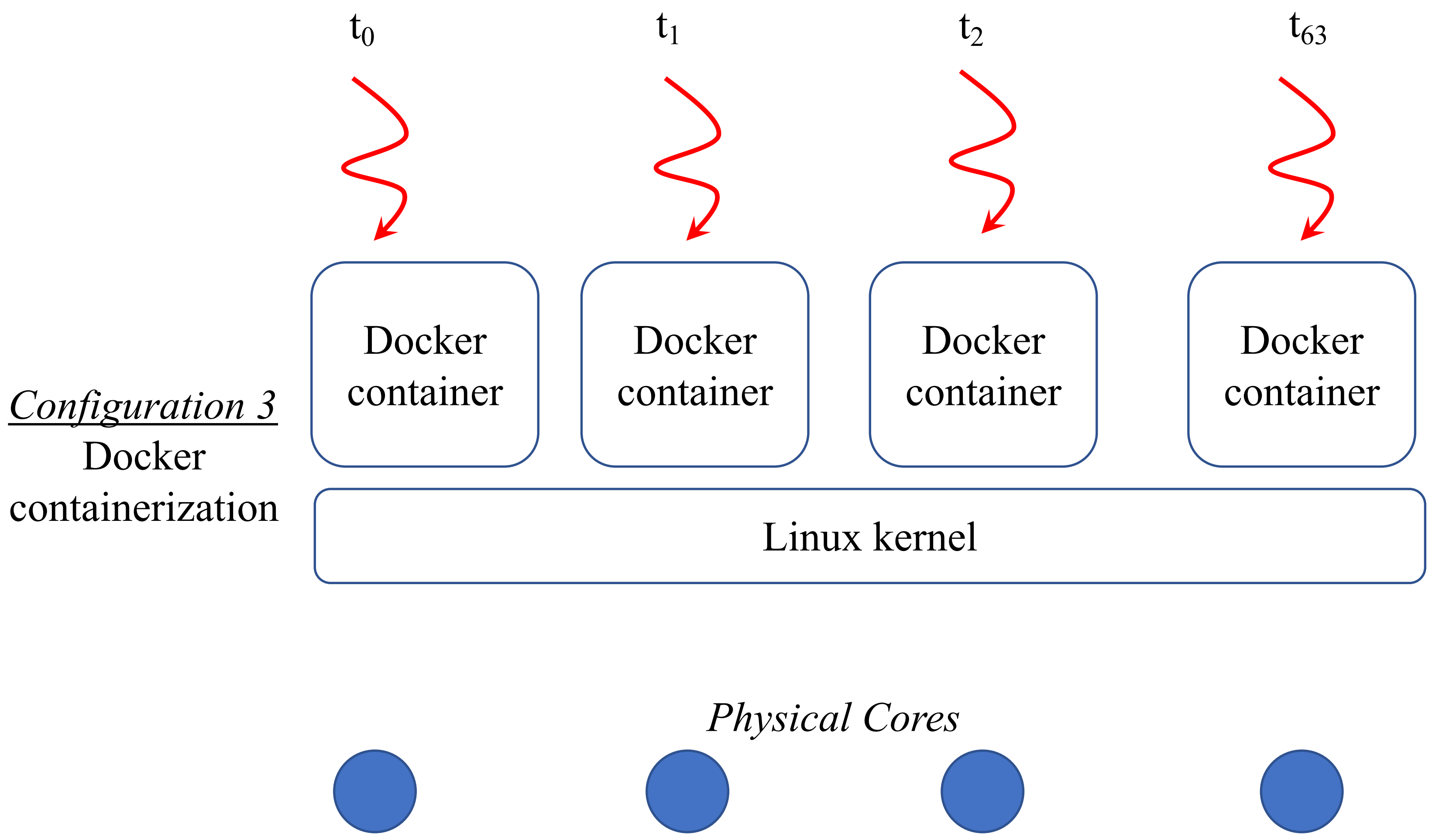
SETUP


t_0
t_1
t_2
t_{63}
Configuration 1
Linux only
Linux kernel
Physical Cores

Setup


t_0
t_1
t_2
t_{63}
Linux kernel
Linux kernel
Linux kernel
Linux kernel
Configuration 2
KVM virtualization
KVM hypervisor
Linux kernel
Physical Cores

SETUP



t_0
t_1
t_2
t_{63}
Docker container
Docker container
Docker container
Docker container
Configuration 3
Docker containerization
Linux kernel
Physical Cores

System Call Performance

	% of system calls with **median** below					
	1μs	10μs	100μs	1ms	10ms	>10ms
Linux	11.12	73.76	96.67	98.81	99.14	0.86
KVM	8.34	57.23	93.11	99.43	99.84	0.16
Docker	7.35	65.87	97.04	98.45	99.67	0.33

Table 1. Breakdown of median system call performance in Linux, KVM, and Docker

	% of system calls with **99th percentile** below					
	1μs	10μs	100μs	1ms	10ms	>10ms
Linux	0.01	31.71	93.28	97.78	99.89	0.11
KVM	0.02	28.37	75.22	99.16	99.81	0.19
Docker	0	19.43	93.63	97.8	99.1	0.9

Table 2. Breakdown of 99th percentile system call performance in Linux, KVM, and Docker

Lack of VM Boundary Causes up to 100x worse 99th %ile Performance

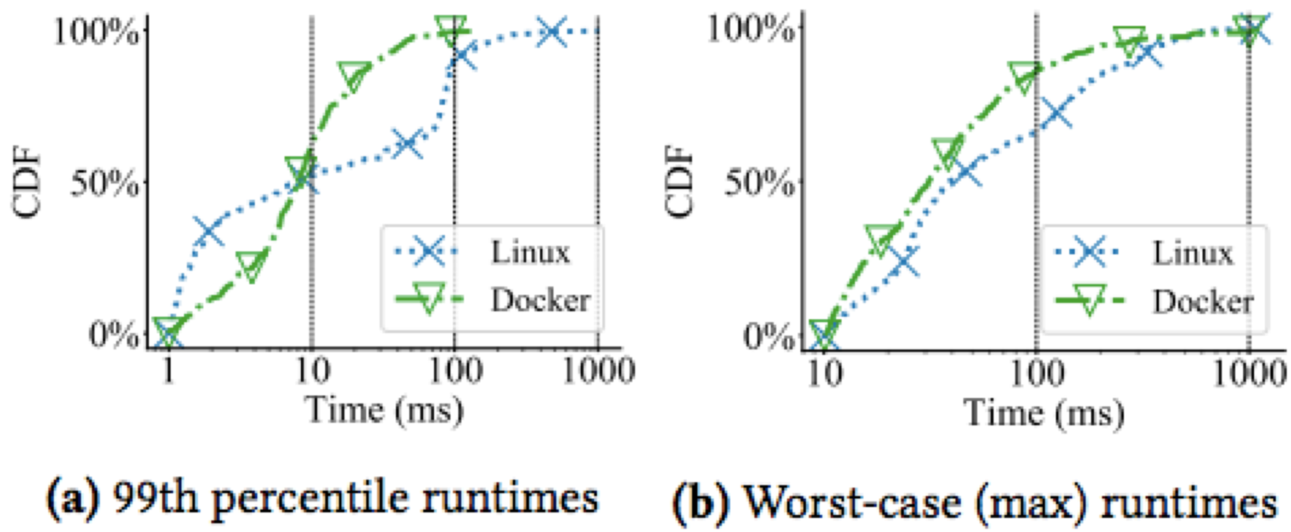


Figure 2. System call outlier distribution in Linux and Docker. All system calls either have 99th percentiles in KVM less than 1ms (a), or worst-case runtimes in KVM less than 10 ms (b)

VMs Much More Effective at Limiting Worst-case Behavior

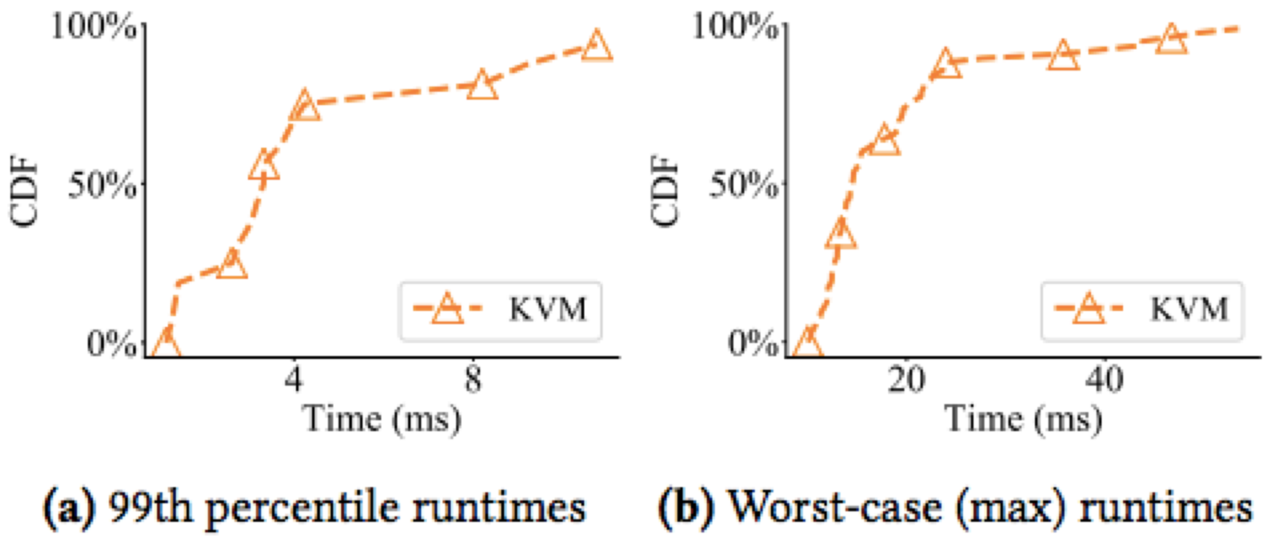


Figure 3. System call outlier distribution in KVM. All system calls either have 99th percentiles in Linux less than 1ms (a), or worst-case runtimes in Linux less than 10ms (b)

SUMMARY

- Worst-case performance is important for many applications

- Linux is not built to provide good worst-case performance, particularly due to contention that spills across workloads

- Techniques such as virtualization help, but other approaches may be better

WORKING IN MY LAB

- Things you will need (in order from most to least important)
    1. Ability to articulate interest in an area that I have some expertise
        - e.g., cloud, supercomputing, real-time, reliability, support for machine learning applications
    2. Firm understanding of low level programming languages (e.g., C)
    3. Solid background in statistics

- Skills you will develop
    - Understanding of low-level hardware/software performance
    - Systems building and evaluation
    - Ability to design and carry out experimental research