Analytical Approaches for Dynamic Scheduling in Cloud Environments

Seyyed Ahmad Javadi

PACE

Stony Brook University

Iran University of Science and Technology

January 1, 2020

Cloud Computing

Cloud providers

Operate cloud infrastructures

Tenants

Rent Virtual Machines (VMs)

Cloud Computing

Cloud providers

Operate cloud infrastructures

Tenants

Rent Virtual Machines (VMs)

Benefits:

Economical virtual machines

Elasticity

…

Challenges:

Performance issues

Security concerns

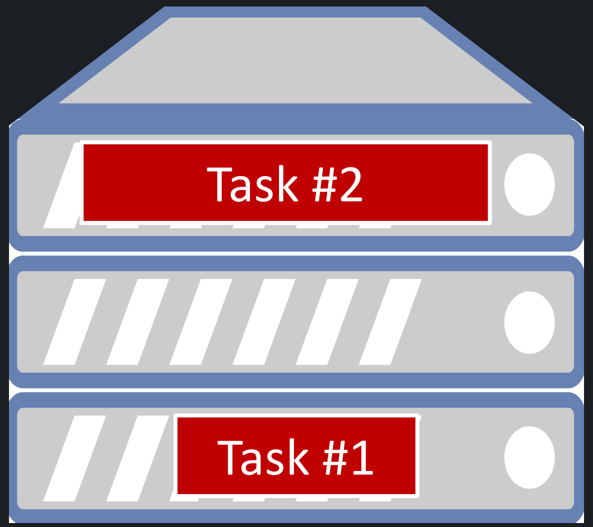
…

Scheduling (Computing)

- Different ways of describing general scheduling problem
- Our purpose: Distributed process scheduling

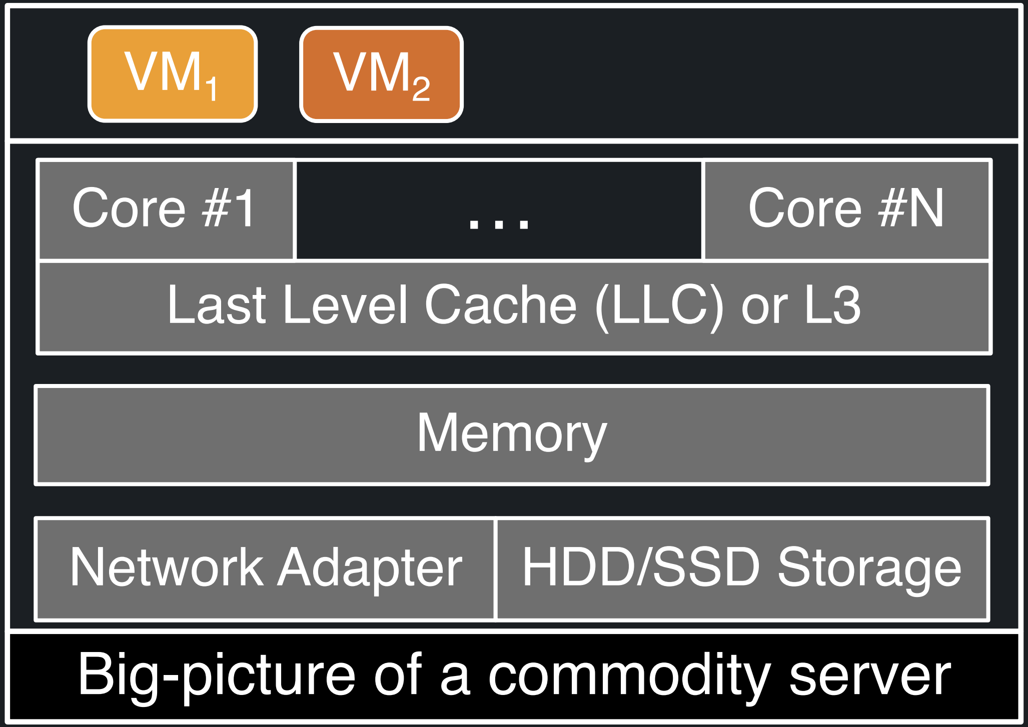
Scheduling (Computing)

- Different ways of describing general scheduling problem
- Our purpose: Distributed process scheduling



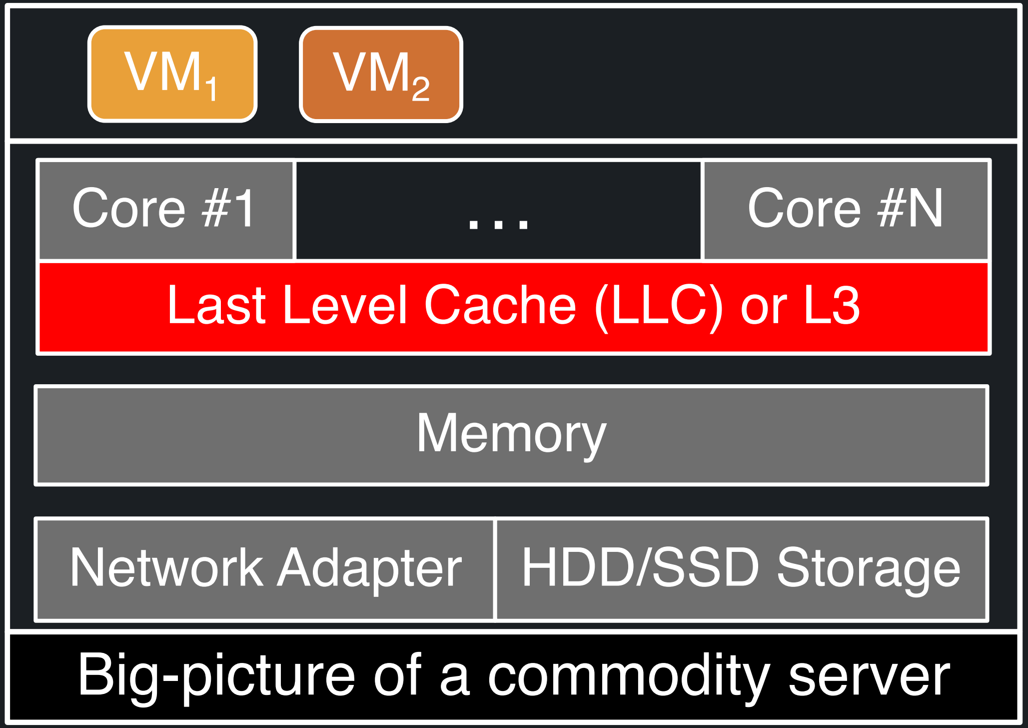
Performance Interference

- *Multi-tenancy* is a main design principle of *cloud computing*
  - The immediate challenge is resource contention



Performance Interference

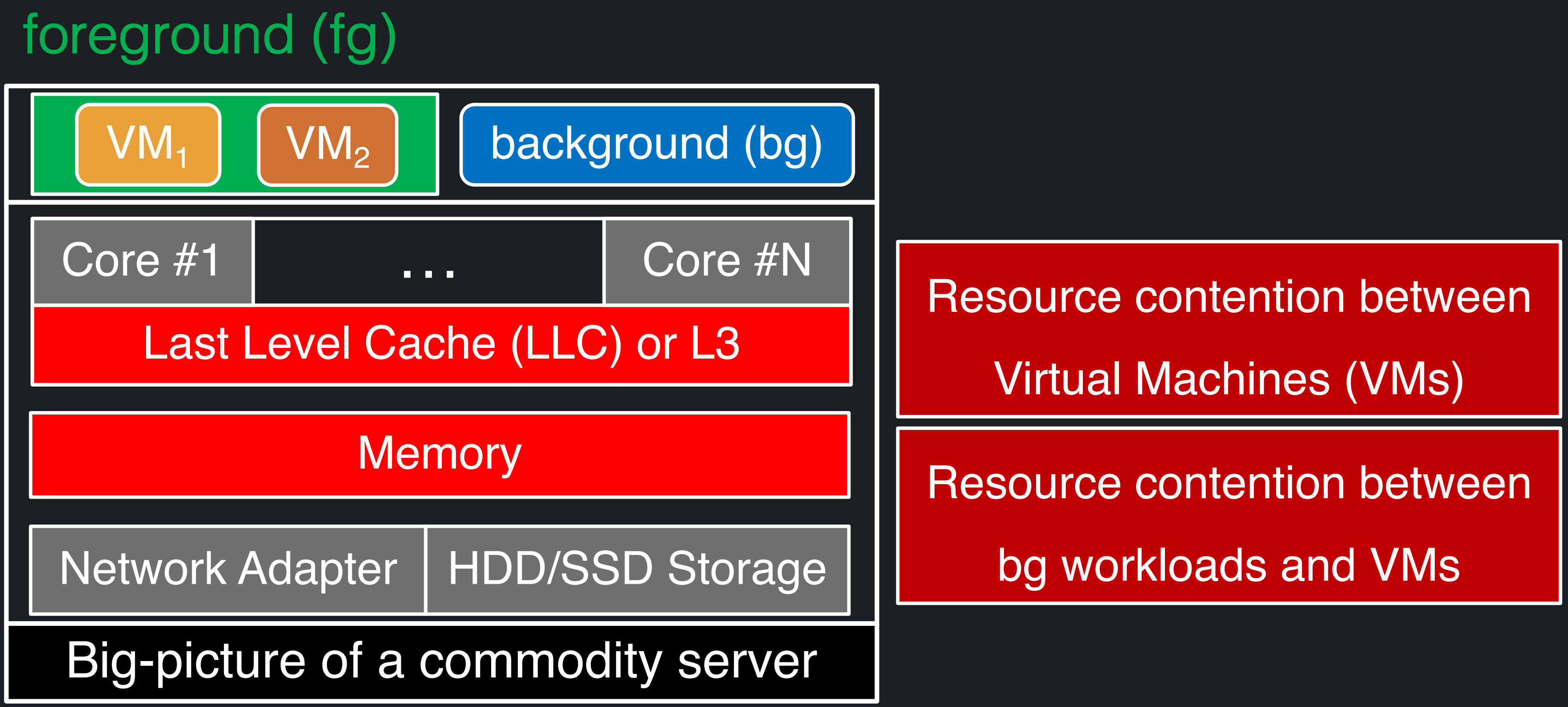
- *Multi-tenancy* is a main design principle of *cloud computing*
  - The immediate challenge is resource contention



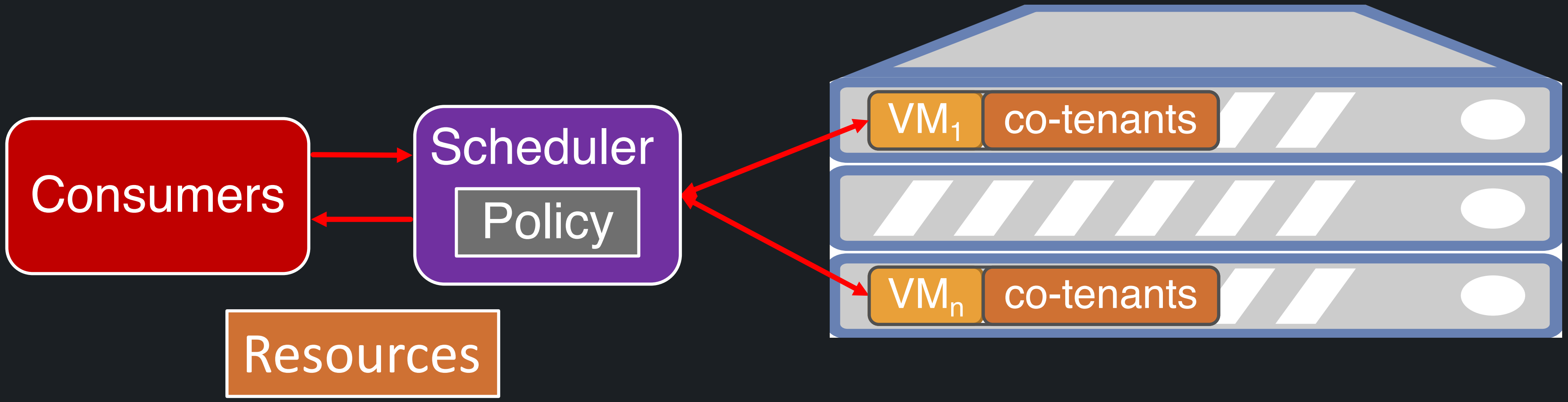
Resource contention between Virtual Machines (VMs)

Performance Interference

- *Multi-tenancy* is a main design principle of *cloud computing*
  - The immediate challenge is resource contention

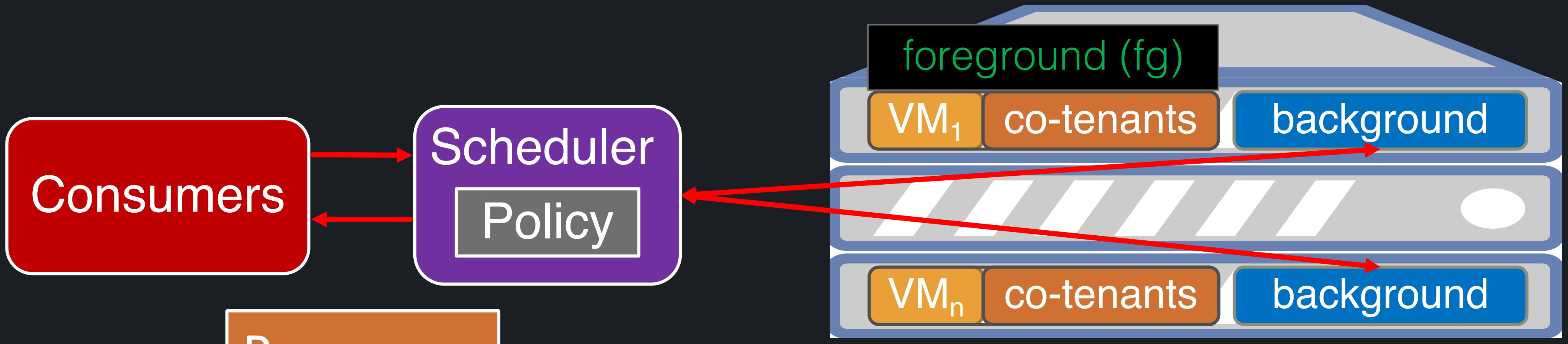


Scheduling Challenges in Cloud



Cluster of Virtual Machines (VMs) → **SC_1**: Variable resource capacity

Scheduling Challenges in Cloud



Resources

Cluster of Virtual Machines (VMs) → **SC$_1$**: Variable resource capacity

Idle resources of physical hosts → **SC$_2$**: Foreground workload is black-box

Using *analytical approaches* to perform *dynamic scheduling* is critical to address the outlined challenges.

Outline

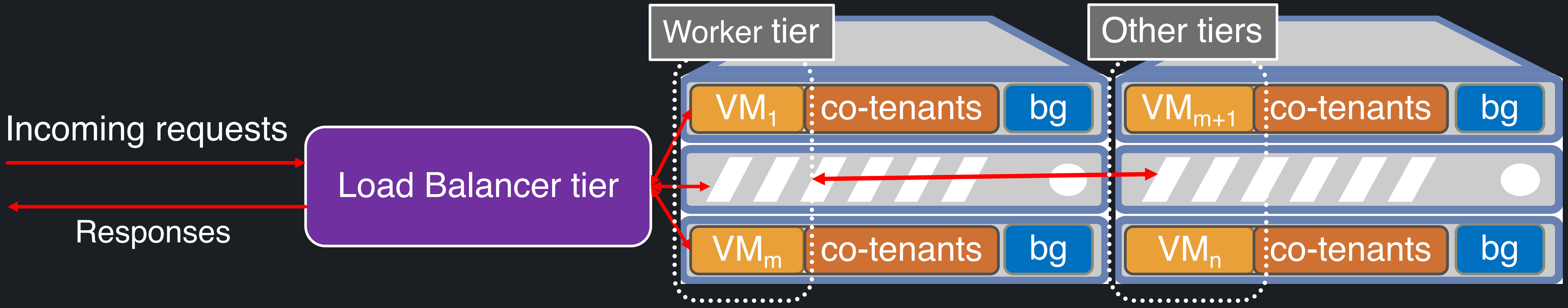
- DIAL: Dynamic interference-aware load balancing
- Scavenger: Resource-adaptive batch scheduling
- Future directions and conclusions

Outline

- **DIAL: Dynamic interference-aware load balancing**
  - IEEE Transactions on Cloud Computing (early access)
- Scavenger: Resource-adaptive batch scheduling
- Future directions and conclusions

Problem: Dealing with Interference

- Generic cloud application containing ***Load Balancer*** *and* ***Worker*** *tiers*

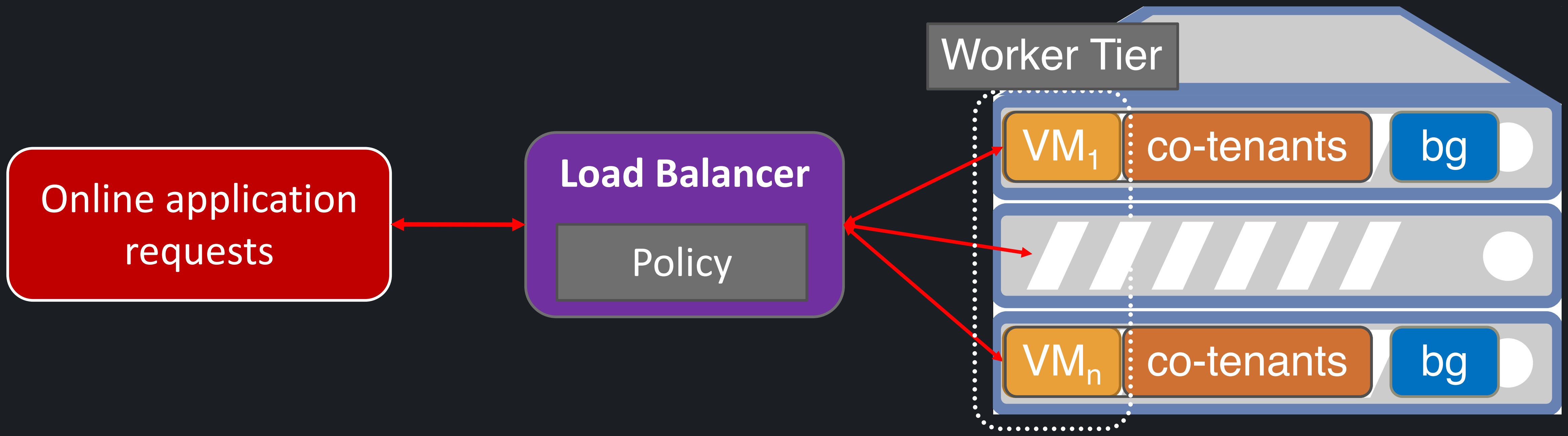


Load-balanced web applications

Online data stores (Pinot)

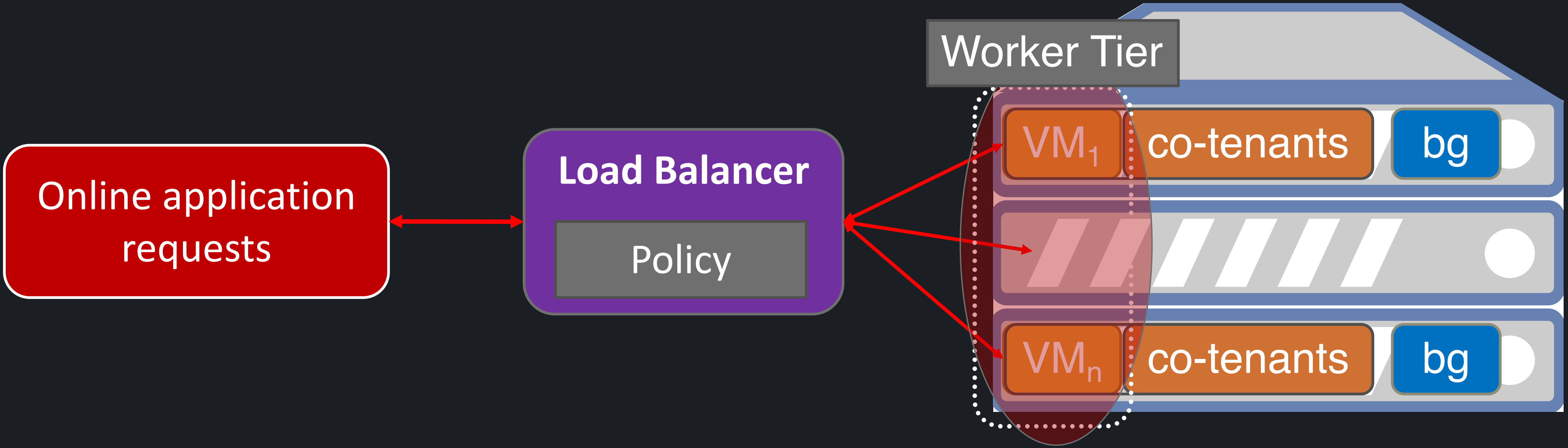
Problem statement: How can load-balanced applications *mitigate the impact of interference*?

DIAL: High-level Idea



- ➢ Cannot observe host resources
  - *Cannot quantify interference*

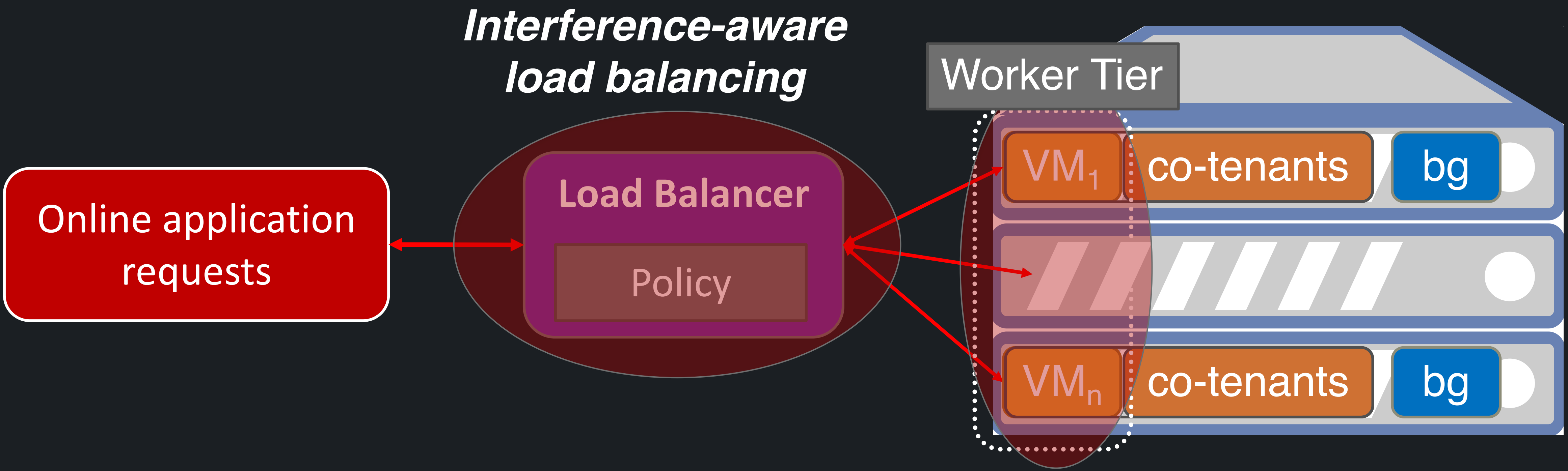
DIAL: High-level Idea



- Cannot observe host resources
  - *Cannot quantify interference*

→ ***Infer the interference***

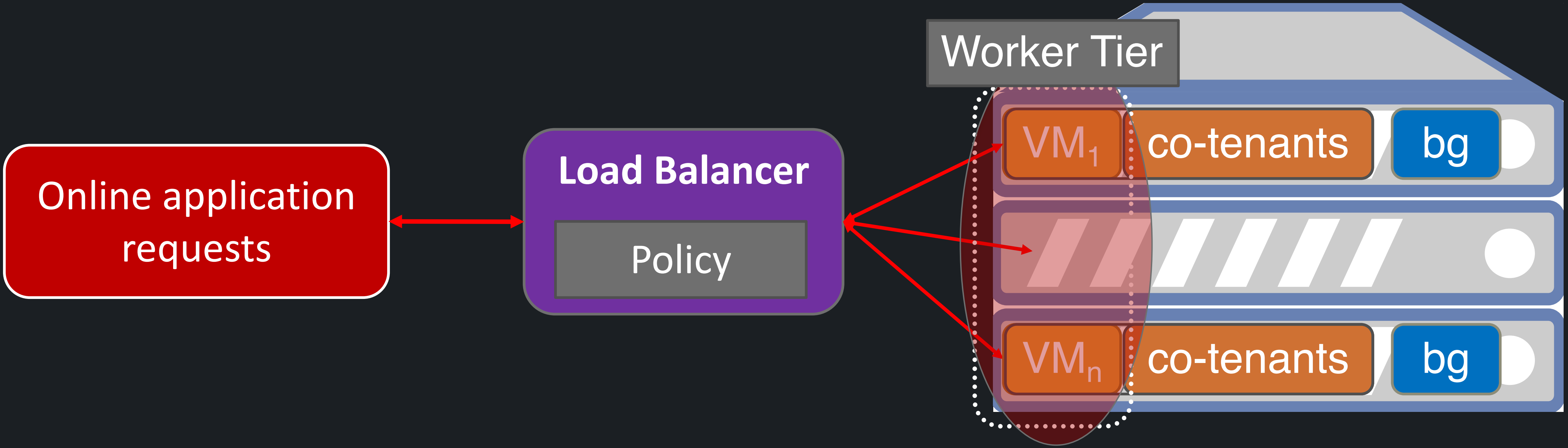
DIAL: High-level Idea



- Cannot observe host resources
  - *Cannot quantify interference*

Infer the interference

DIAL: High-level Idea

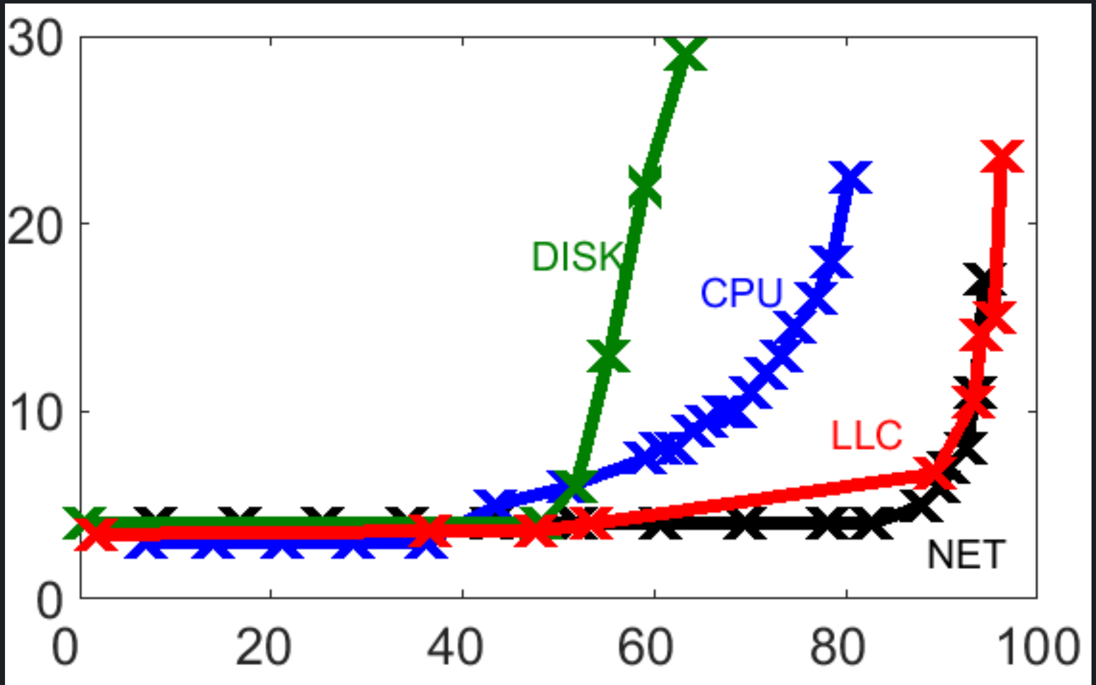


- Cannot observe host resources
  - *Cannot quantify interference*

→ ***Infer the interference***

Analyzing Interference

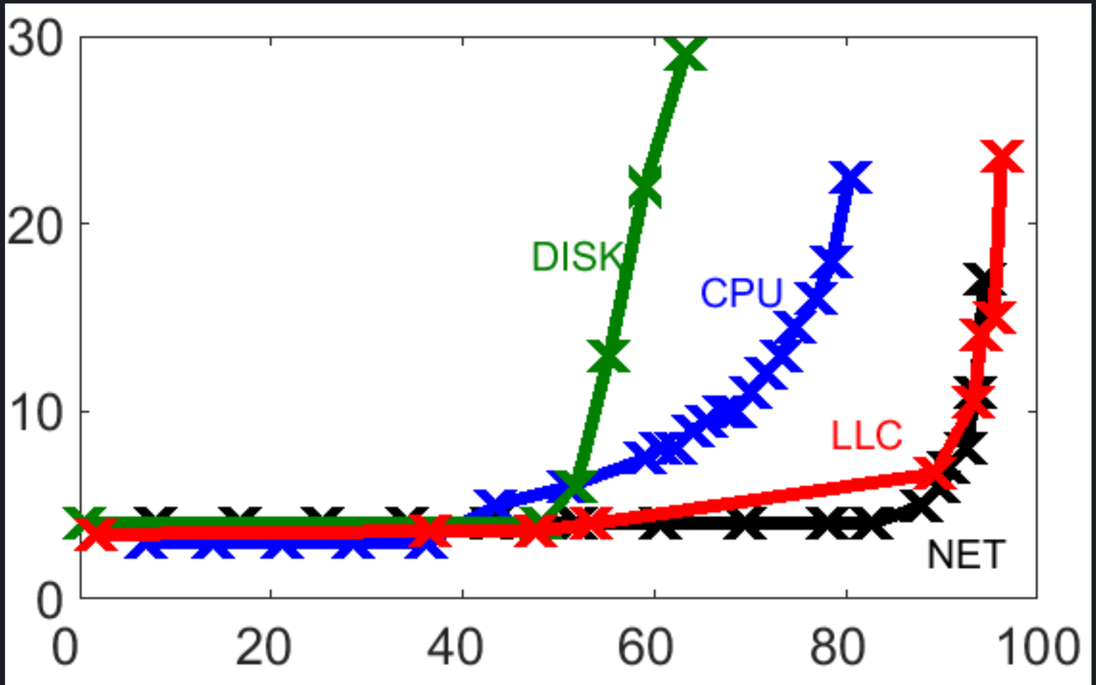
90%ile Response time (ms)



Total usage (fg + co-tenants), in % →

Goal: *Can we infer co-tenants' usage from RT and fg load?*

Analyzing Interference



90%ile Response time (ms)

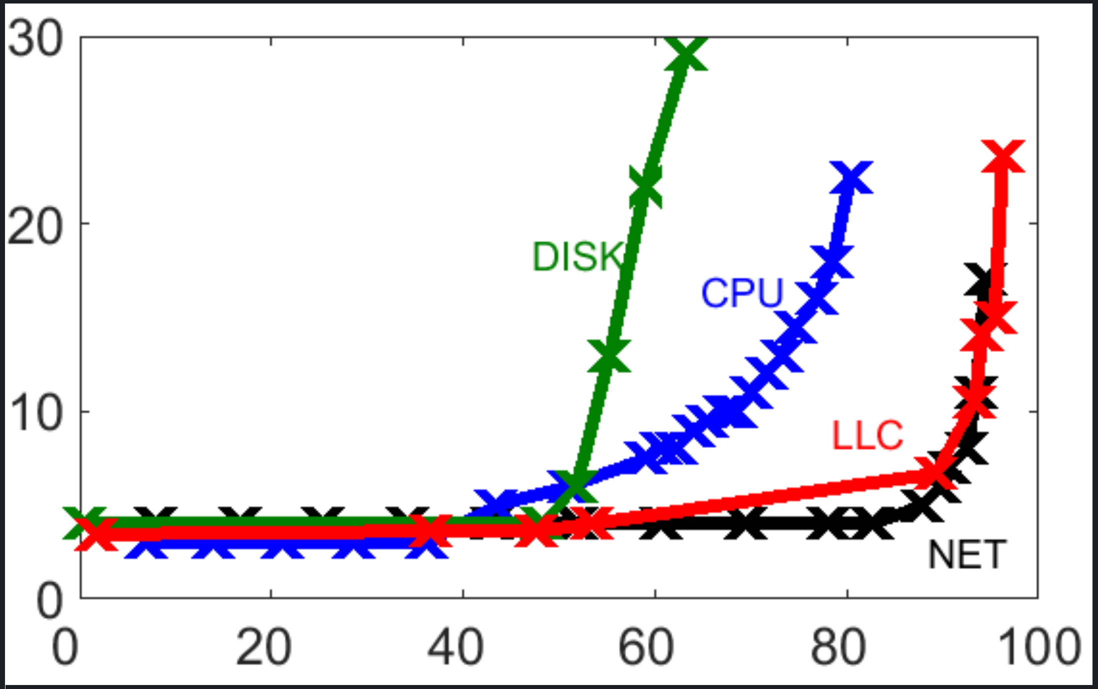
Total usage (fg + co-tenants), in % →

Observation:

Non-linear curves

Goal: *Can we infer co-tenants' usage from RT and fg load?*

Analyzing Interference



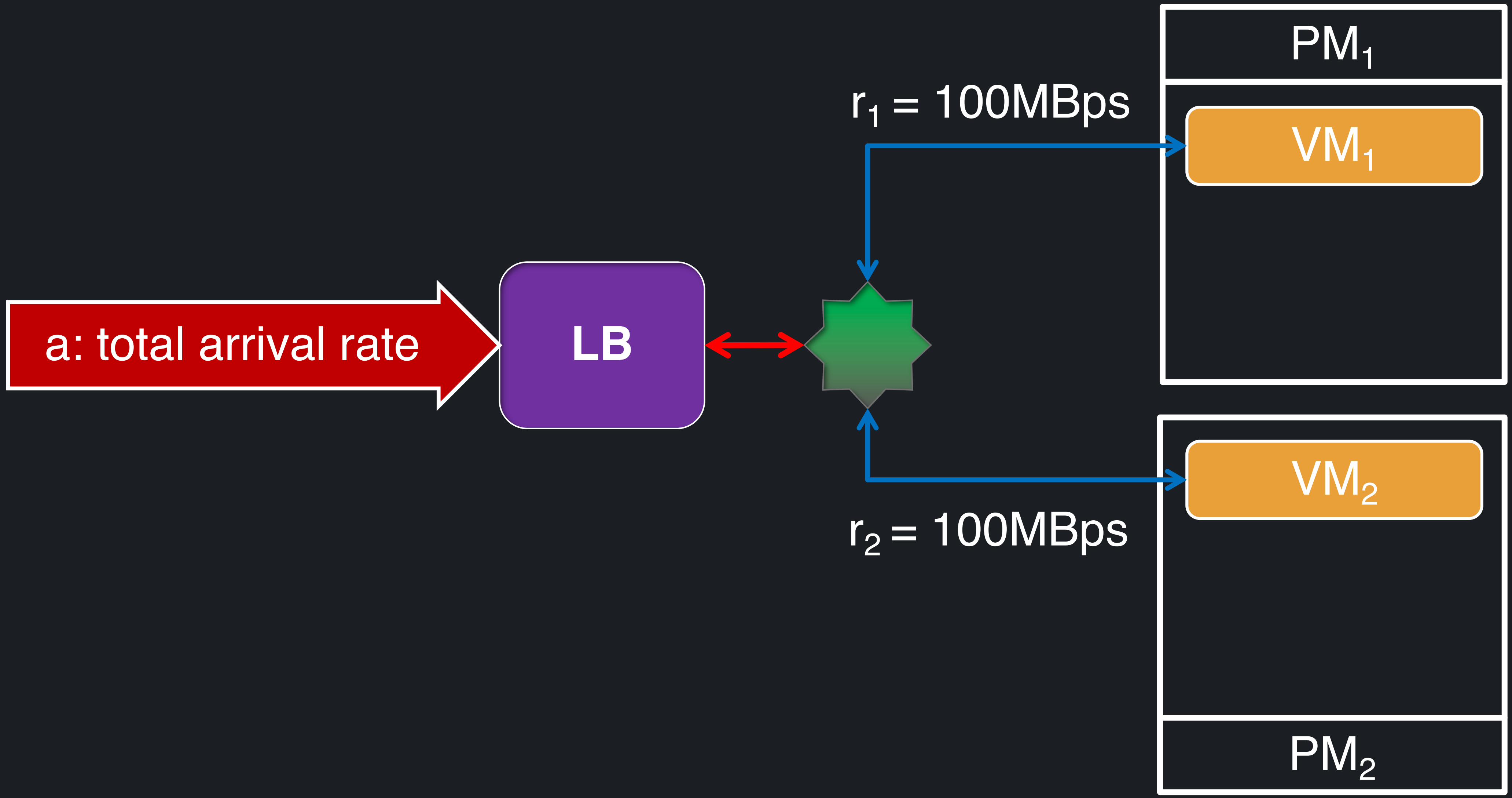
Observation:

Non-linear curves

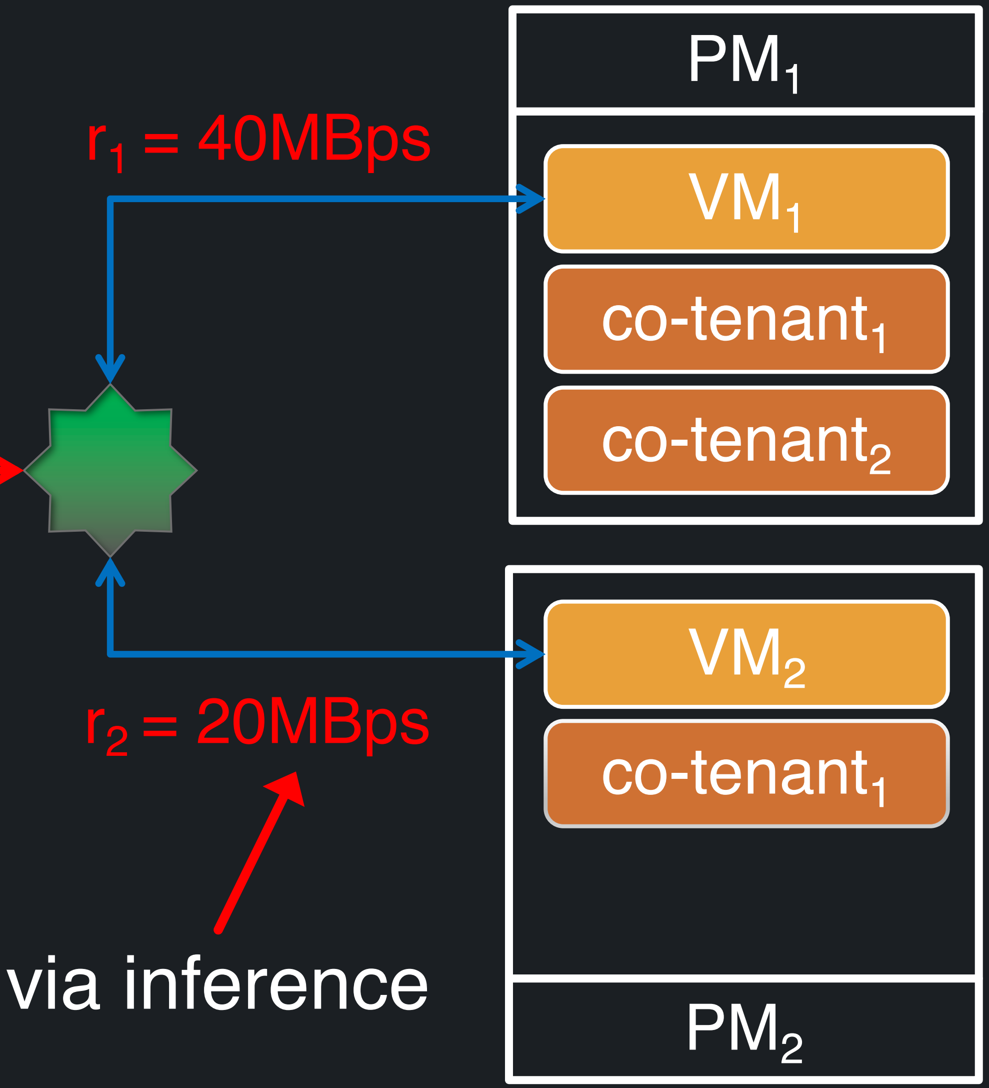
Queueing + Regress

We look at ***slope of curve*** and use that, along with ***queuing theory***, to detect ***how much resources are being taken away***.

Optimal Weight Derivation



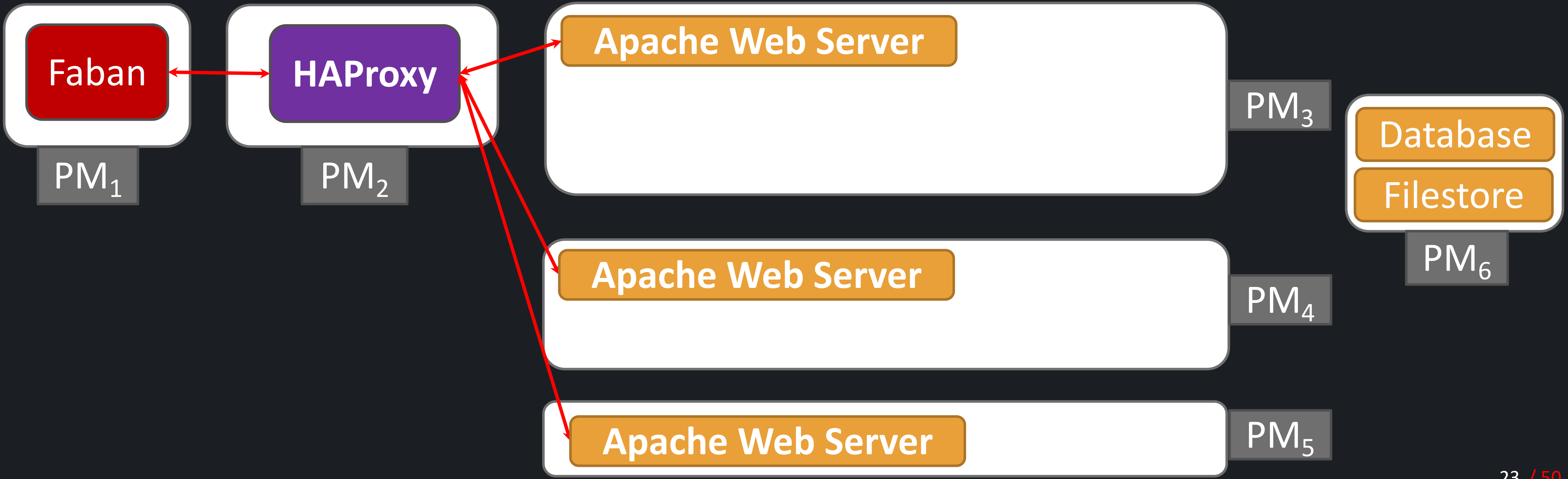
Optimal Weight Derivation



$$p_i^* = \left(r_i \sum_{j=1}^{n} \sqrt{r_j} - \sqrt{r_i} \sum_{j=1}^{n} r_j + a\sqrt{r_i} \right) / \left(a \sum_{j=1}^{n} \sqrt{r_j} \right)$$

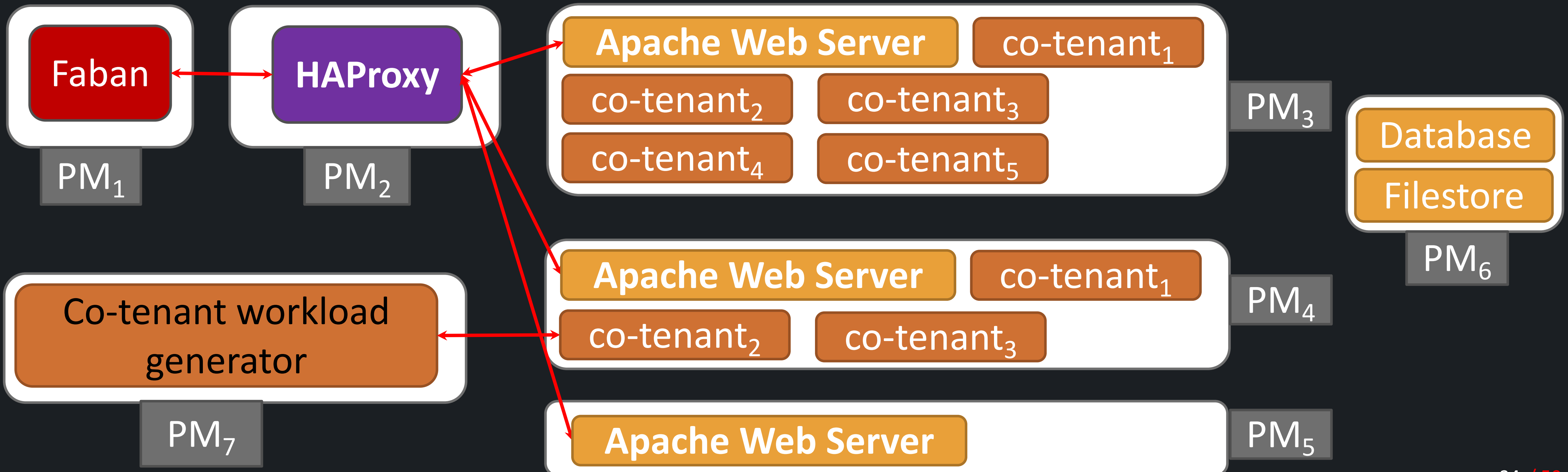
Experimental Setup

- Physical Machine
  - Ubuntu 14.04; OpenStack
  - 12 cores, 48GB DRAM, 1 Gb/s network
- Virtual Machine
  - 4 vCPUs, 4GB of memory



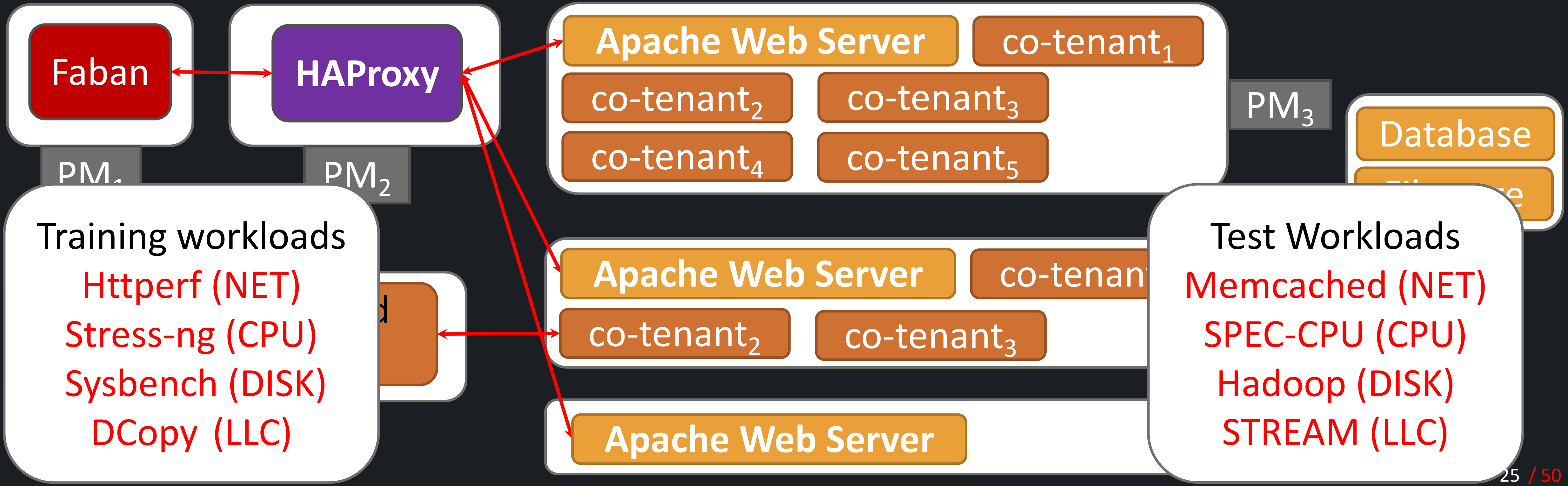
Experimental Setup

- Physical Machine
  - Ubuntu 14.04; OpenStack
  - 12 cores, 48GB DRAM, 1 Gb/s network
- Virtual Machine
  - 4 vCPUs, 4GB of memory



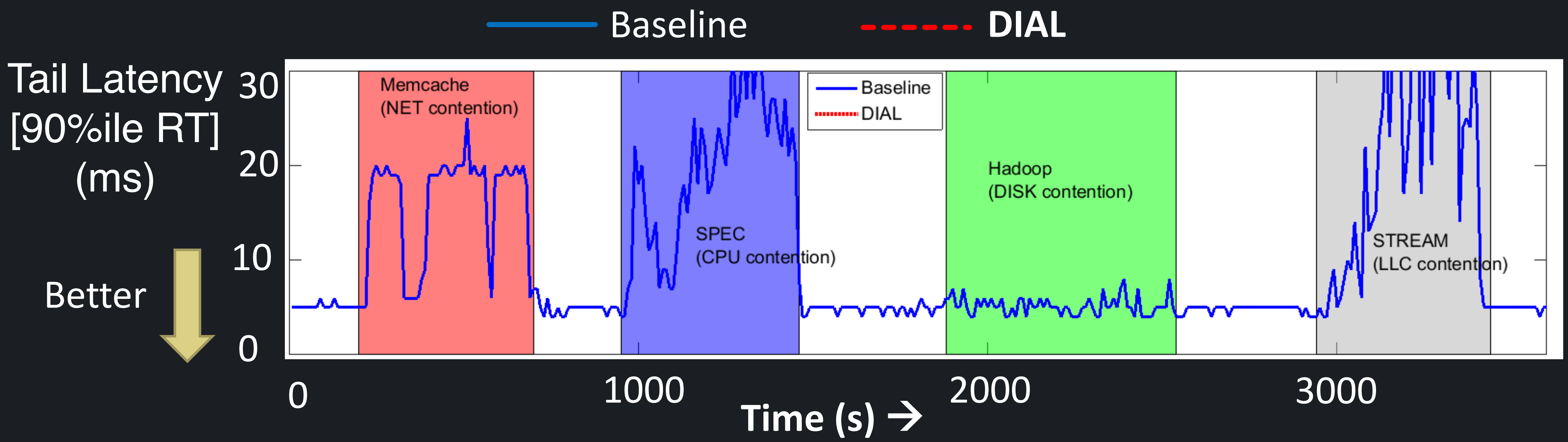
Experimental Setup

- Physical Machine
  - Ubuntu 14.04; OpenStack
  - 12 cores, 48GB DRAM, 1 Gb/s network
- Virtual Machine
  - 4 vCPUs, 4GB of memory



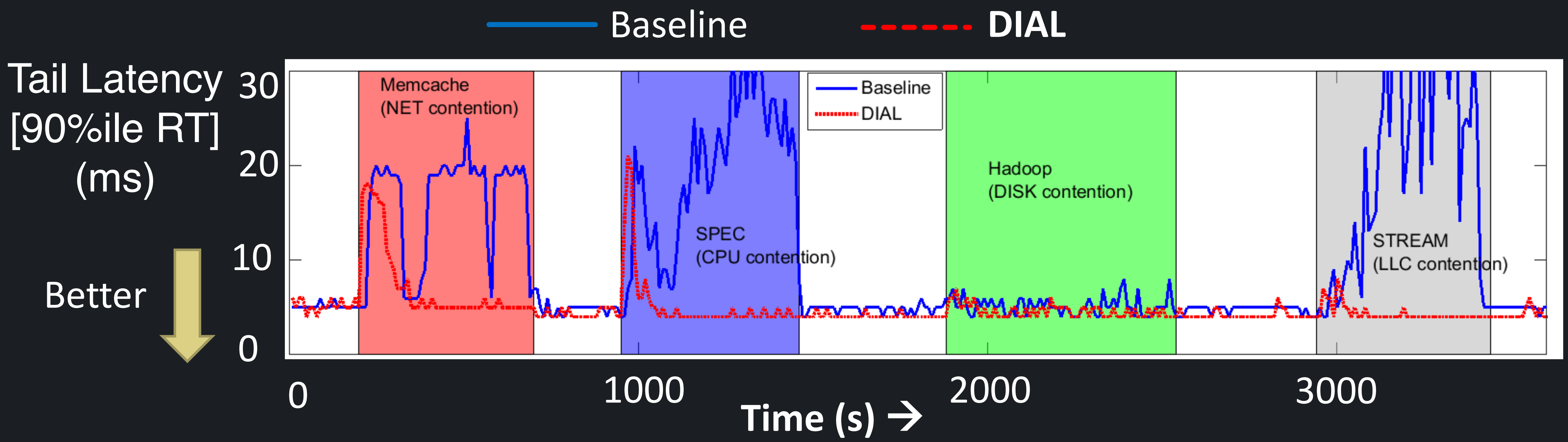
DIAL: OpenStack + CloudSuite

- Baseline: Round-robin algorithm and DIAL is disabled
- DIAL: Using optimal weights in weighted round robin algorithm



DIAL: OpenStack + CloudSuite

- Baseline: Round-robin algorithm and DIAL is disabled
- DIAL: Using optimal weights in weighted round robin algorithm

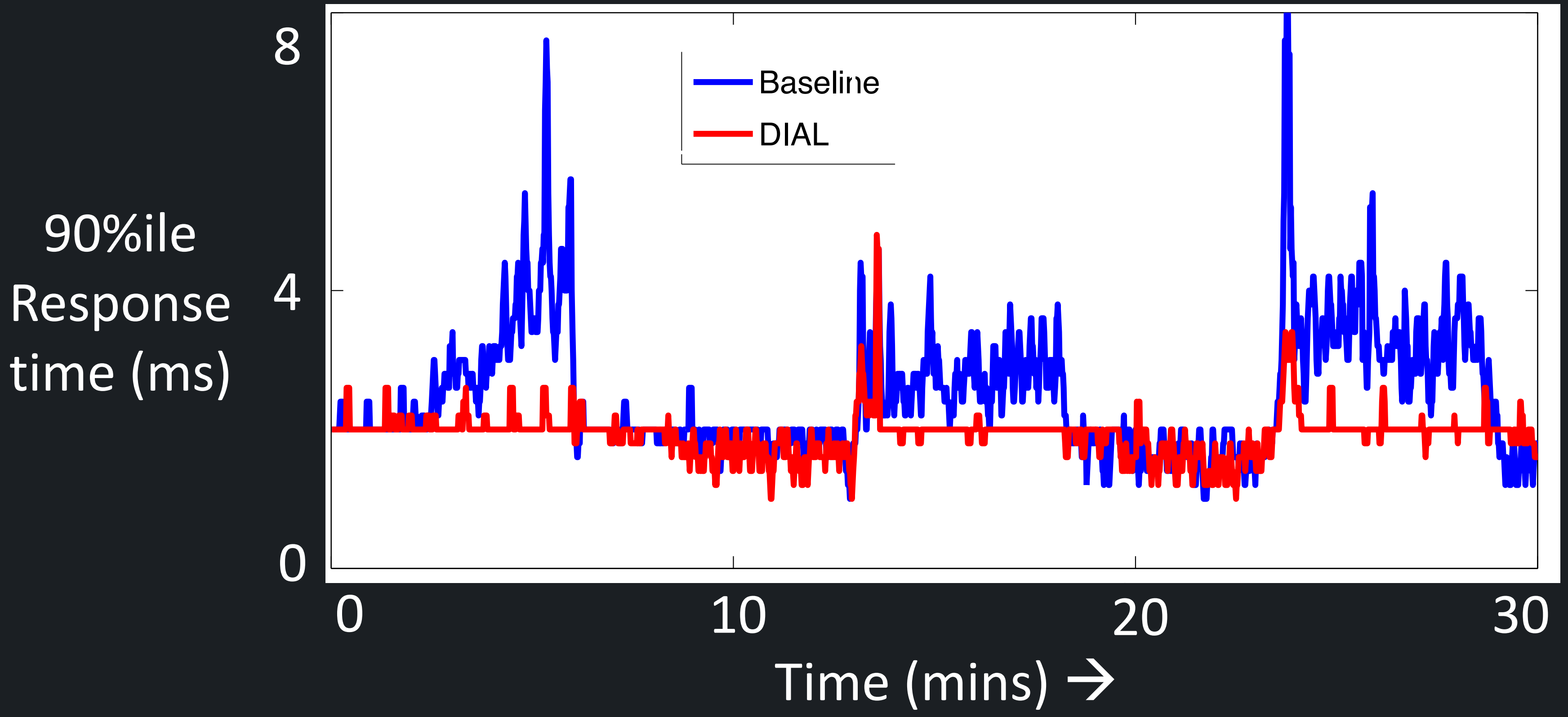


average reduction

↓ 39% ↓ 56% ↓ 16% ↓ 59%

DIAL: AWS + CloudSuite

- 10 Apache VMs
- LLC contention via AWS dedicated hosts

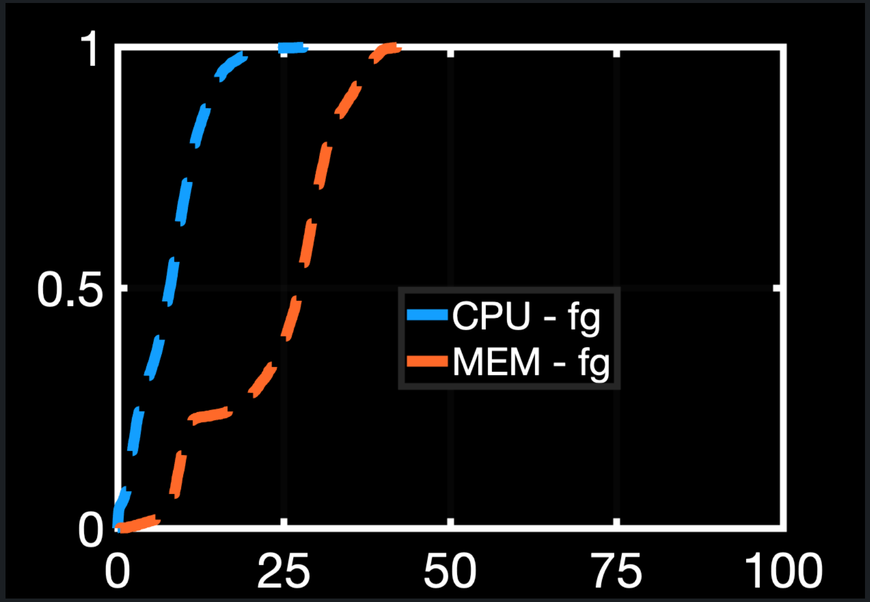


↓ 22%

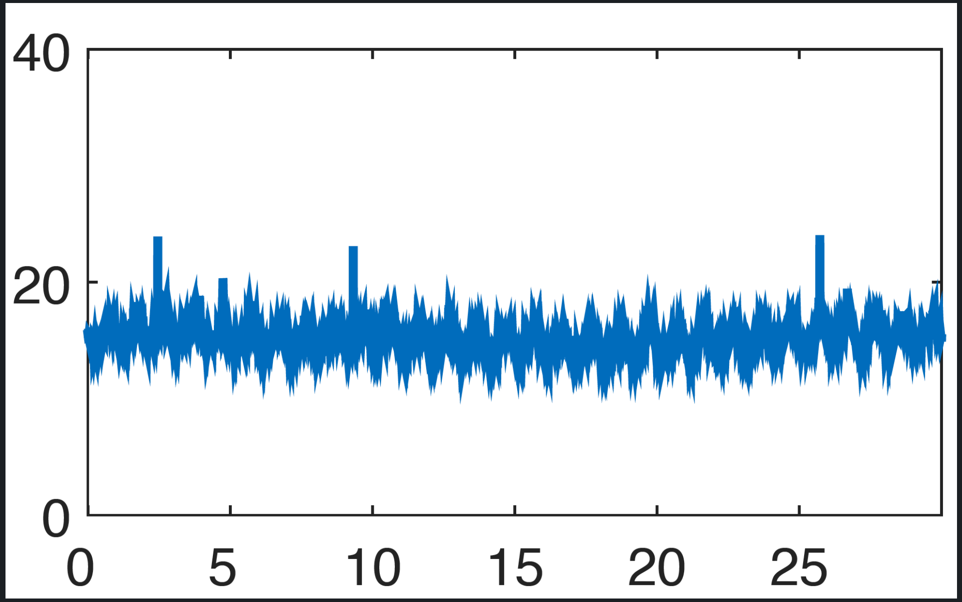
Outline

- DIAL: Dynamic interference-aware load balancing
- **Scavenger: Resource-adaptive batch scheduling**
  - 10th ACM Symposium on Cloud Computing 2019
- Future directions and conclusions

Low Resource Utilization in Cloud Environments



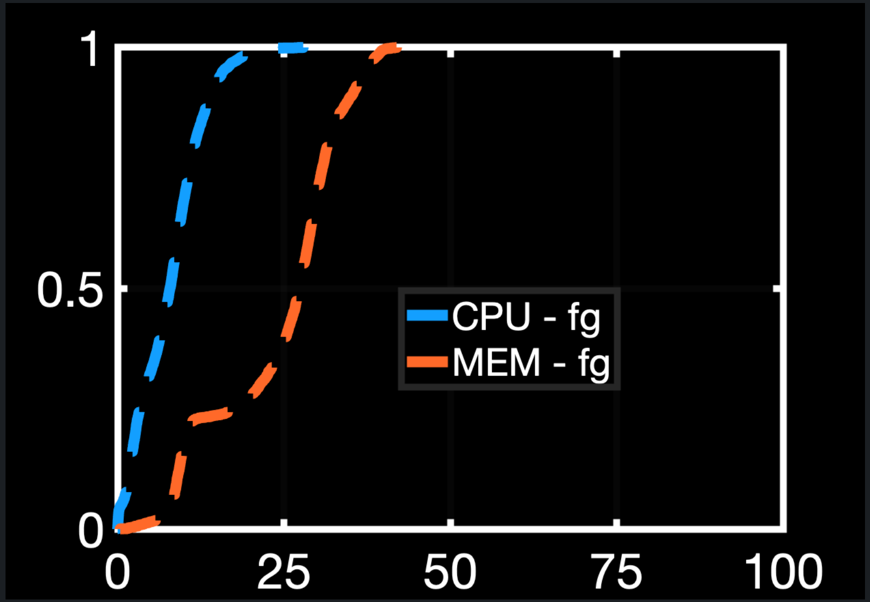
CDF of average CPU and memory usage, Alibaba cluster trace (2018).



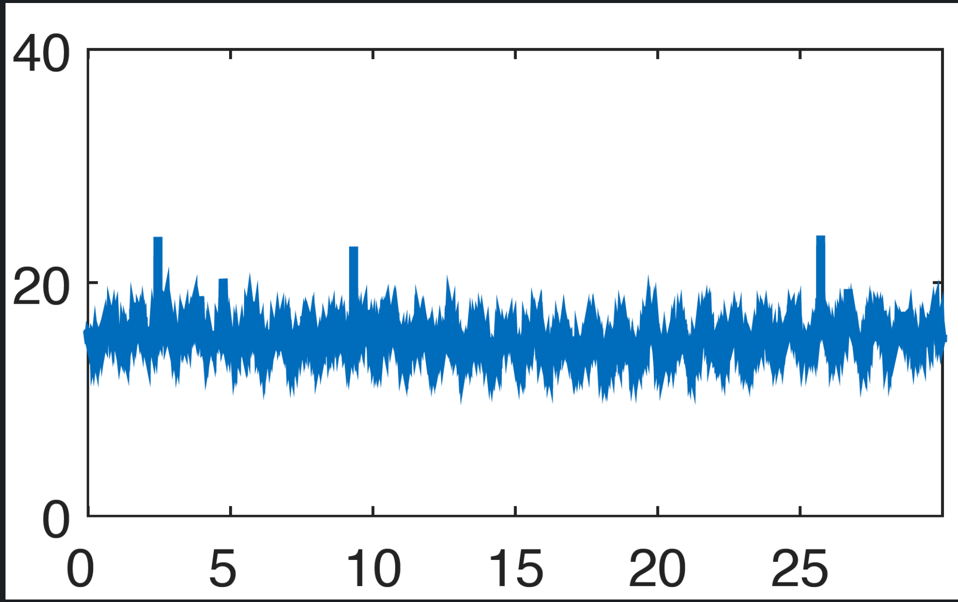
VM-level CPU usage for the Azure trace (2017).

fg = foreground/online workload

Low Resource Utilization in Cloud Environments



CDF of average CPU and memory usage,

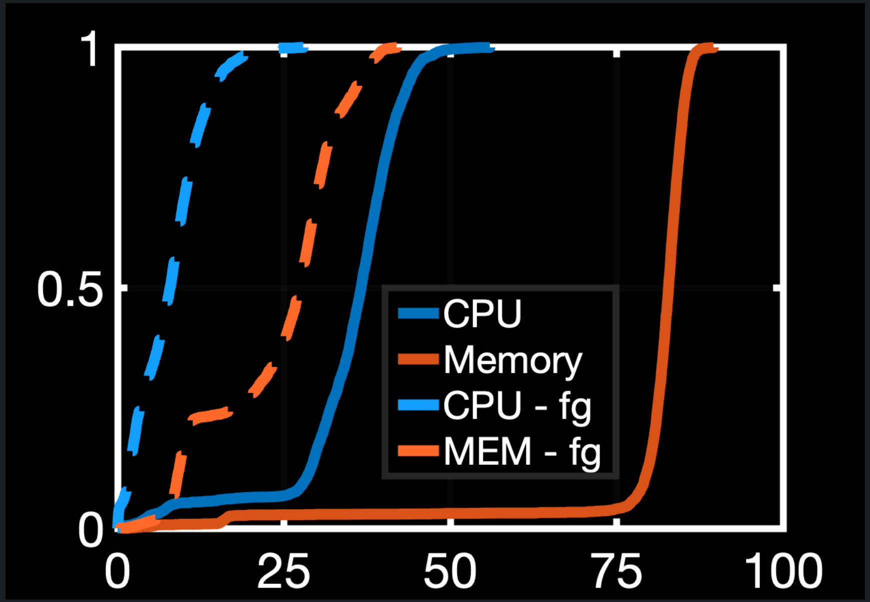


VM-level CPU usage for the Azure

Great opportunity to use cloud idle resources

Opportunity: Running Background Batch Workload

Cumulative probability, F(x)



X = Average usage

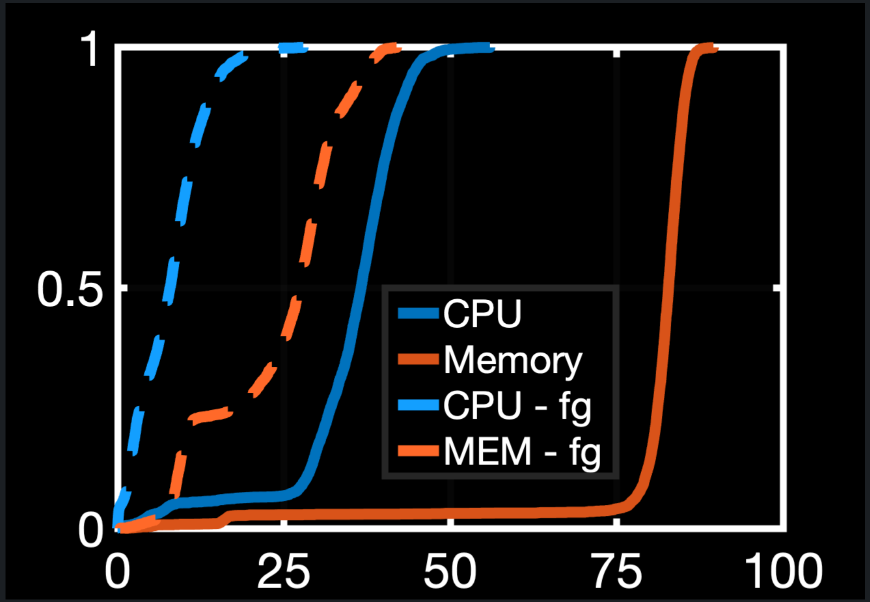
CDF of average CPU and memory usage, Alibaba cluster trace (2018).

bg = background/batch workload

- Key challenge: Resource contention
  - May violate SLOs of ***foreground dynamic workload***
  - Foreground workload is a ***black-box***, SLOs not known

Opportunity: Running Background Batch Workload

Cumulative probability, F(x)



X = Average usage

- Key challenge: Resource contention
  - May violate SLOs of ***foreground dynamic workload***
  - Foreground workload is a ***black-box***, SLOs not known

Problem statement: How to schedule background batch jobs to improve utilization without hurting black-box foreground performance?

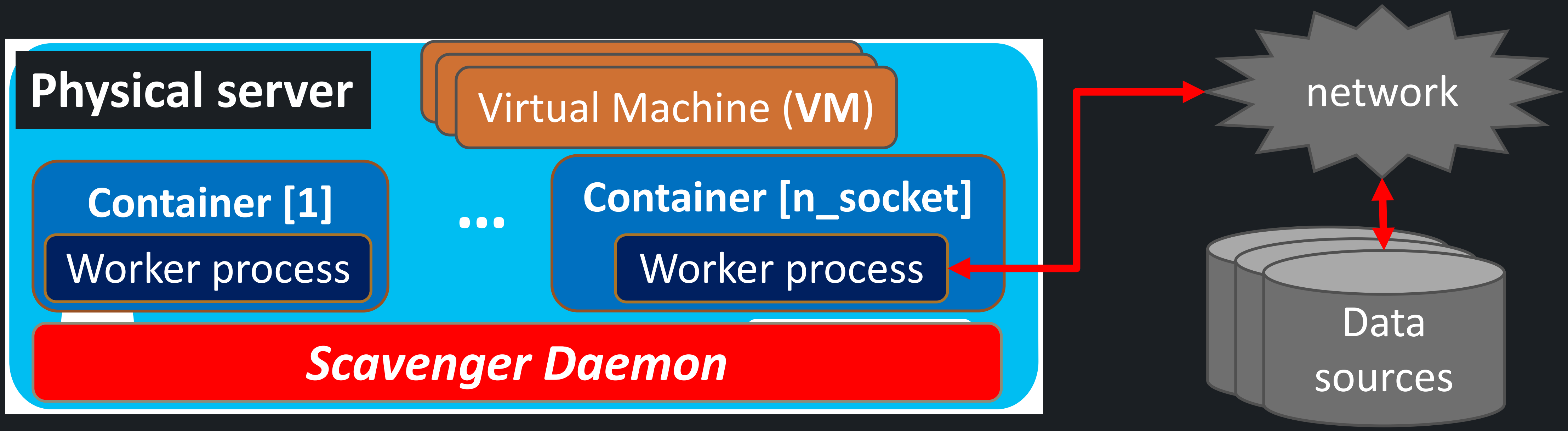
Prior approaches

- ***Treat foregroud as white-box (assume SLO is known)***
  - Bistro (ATC'15, Facebook)
  - Heracles (ISCA'15, Google)
  - History-based harvesting (OSDI'16, Microsoft)
  - PARTIES (ASPLOS '19, SAIL group-Cornell Uni.)

- ***Typically focus only on one resource (need some critical profiling)***
  - dCat (EuroSys'18, IBM)
  - PerfIso (ATC'18, Microsoft)
    - o Reprofiles often if workload changes

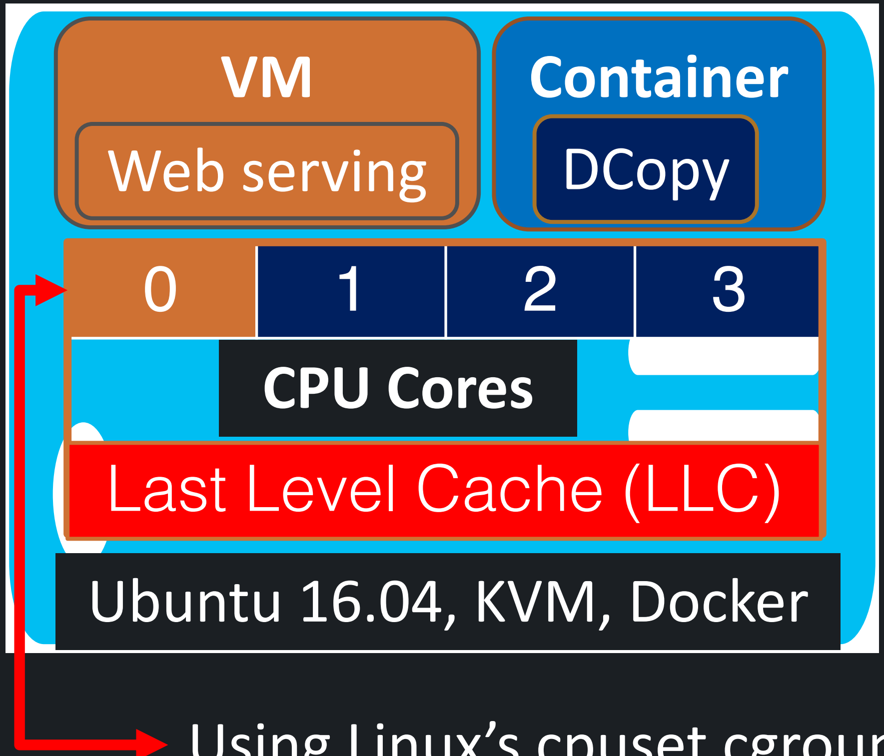
Our approach: Scavenger

- Considers foreground workloads as a ***black-box***
- Takes ***multiple resources*** (processor, memory, nw) into account
- Is a dynamic and tunable solution
- Uses container as the ***agile execution environment*** for batch jobs



Scavenger Daemon

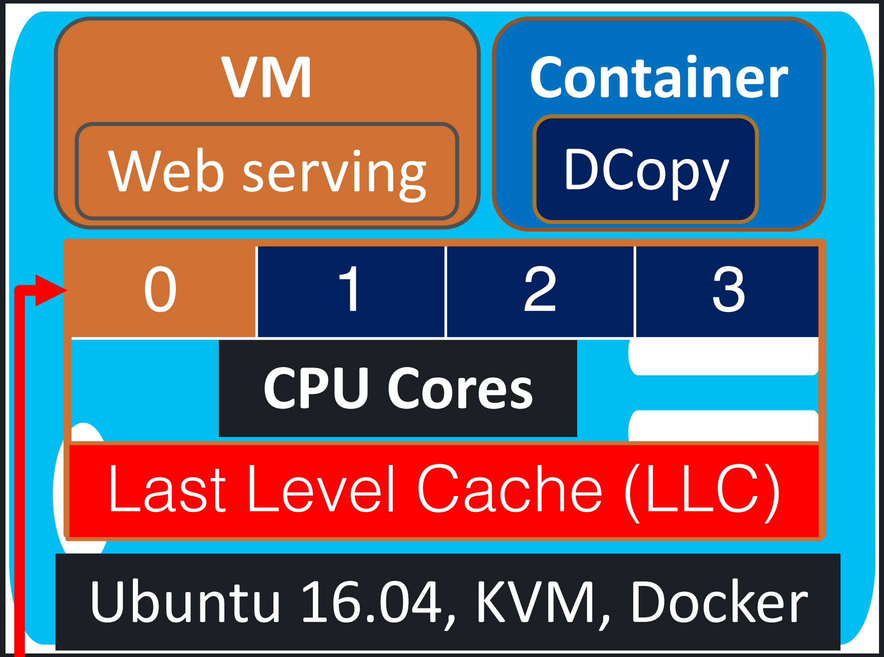
- Background resource regulation is the main design decision
  - Dealing with resource contention is challenging



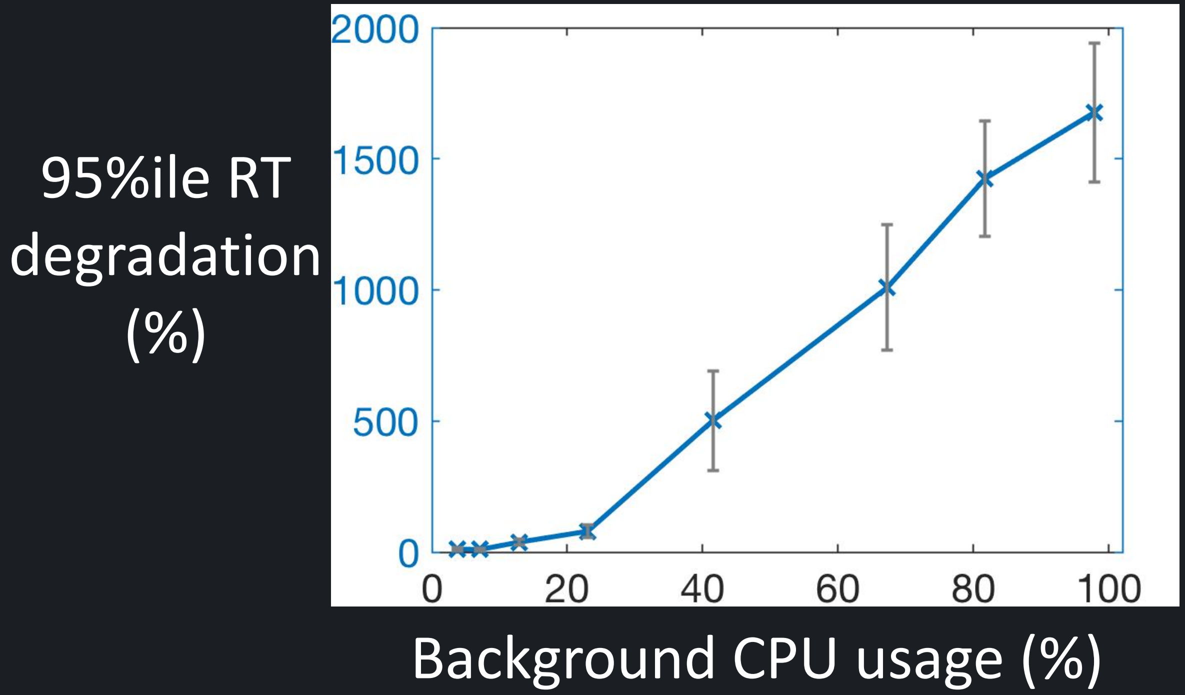
Using Linux's cpuset cgroups

Scavenger Daemon

- Background resource regulation is the main design decision
  - Dealing with resource contention is challenging

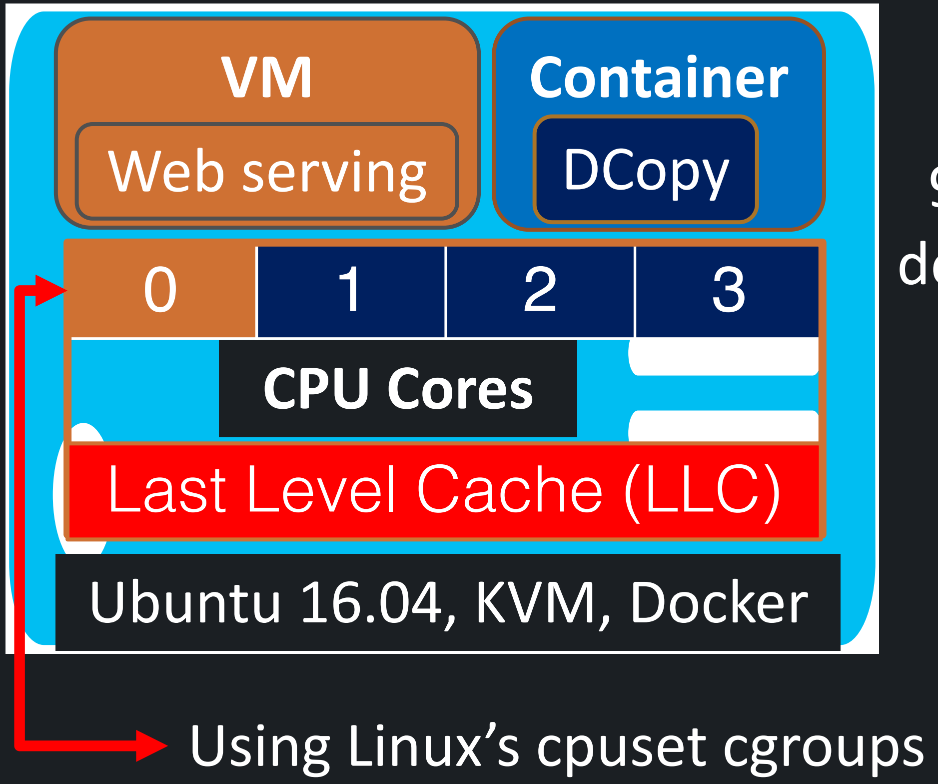


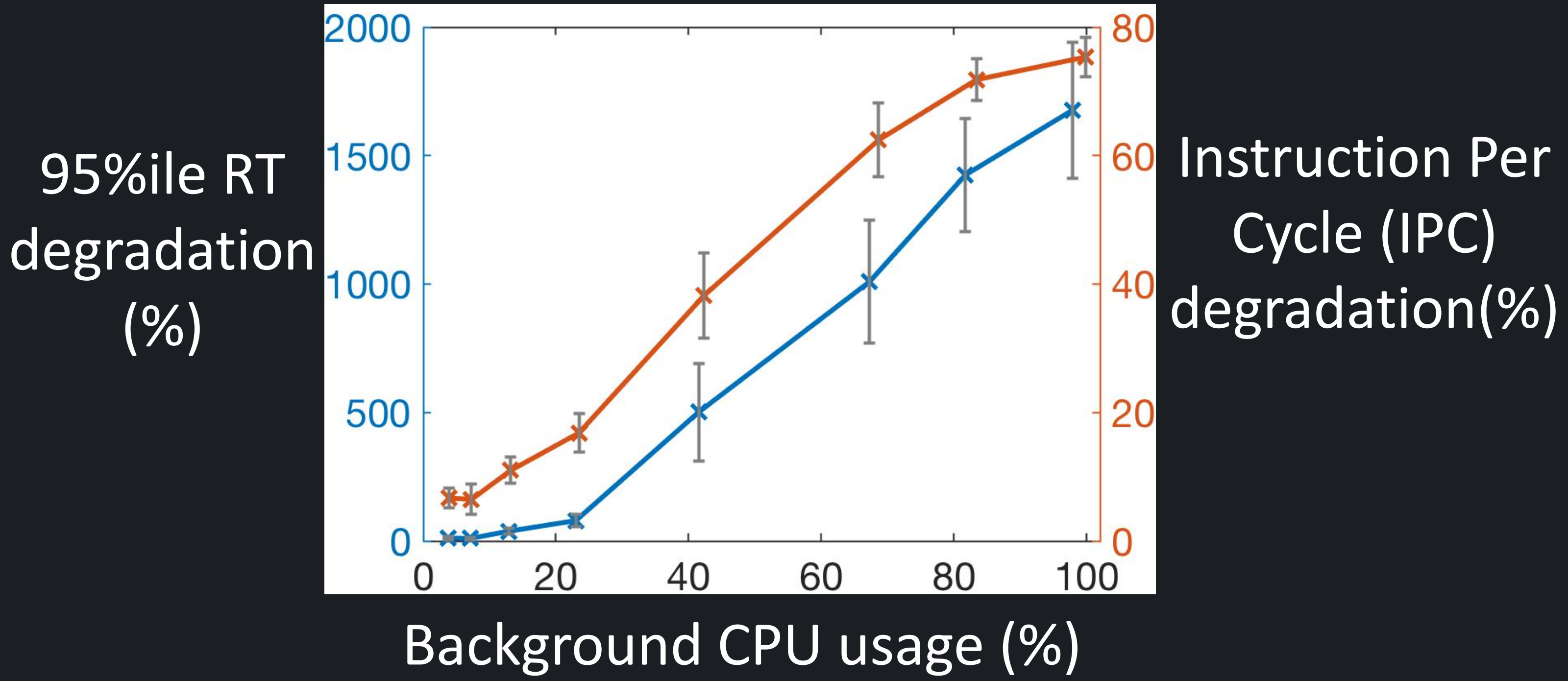
Using Linux's cpuset cgroups



Scavenger Daemon

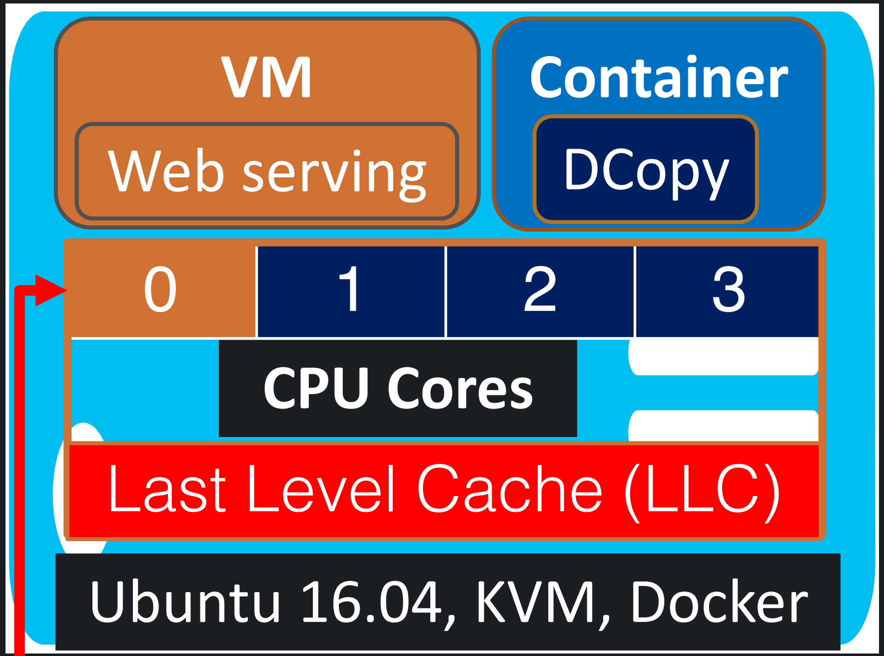
- Background resource regulation is the main design decision
  - Dealing with resource contention is challenging





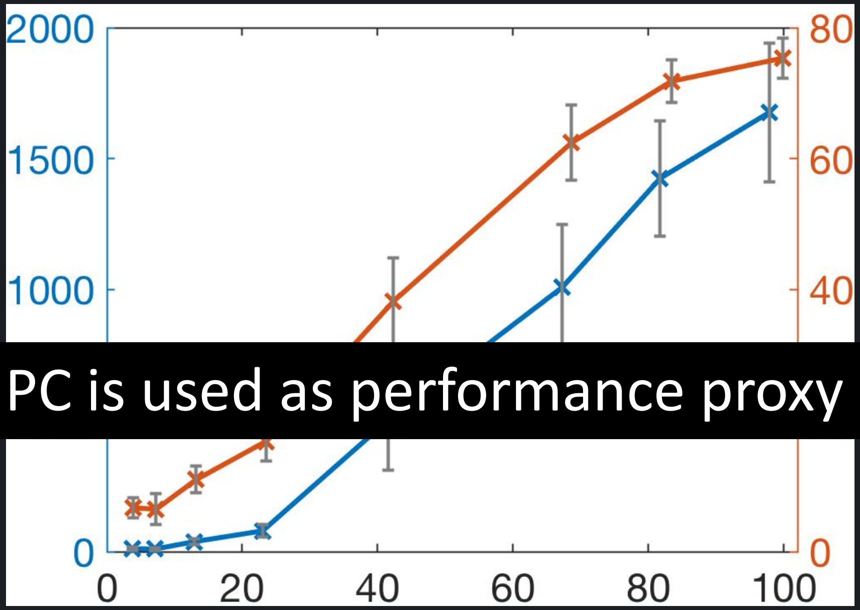
Scavenger Daemon

- Background resource regulation is the main design decision
  - Dealing with resource contention is challenging



Using Linux's cpuset cgroups

95%ile RT degradation (%)



Instruction Per Cycle (IPC) degradation(%)

IPC is used as performance proxy

Background CPU usage (%)

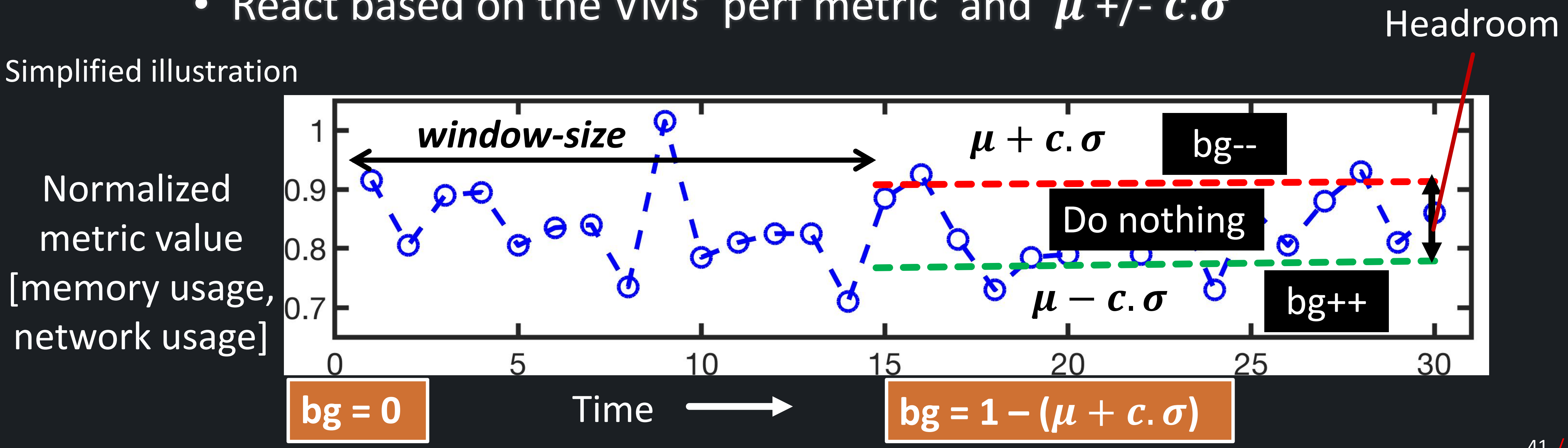
Resource Regulation Algorithm

- Scavenger determines availability of resources for bg jobs
  - Background CPU load (cgroups)
    - CPU quota (maximum CPU cycles given to a process under the CFS)
  - Memory capacity (libvit)
  - Network bandwidth (TC)

Resource Regulation Algorithm

- Our generic online algorithm
  - Monitor VMs' perf metric (e.g., memory usage) for window-size
  - Calculate mean, μ, and standard deviation, σ
  - React based on the VMs' perf metric and μ +/- $c.\sigma$

Simplified illustration



Evaluation Methodology

- Scavenger prototype implementation
  - Largely written in C++ and shell script (~750 lines of code)

Foreground	Training	CloudSuite	Widely used benchmark suite
	Testing	TailBench	Designed for latency-critical applications
Background (SparkBench)	KMeans		A popular clustering algorithm
	SparkPi		Computes Pi with very high precision

Sensitivity analysis → Experimental evaluation

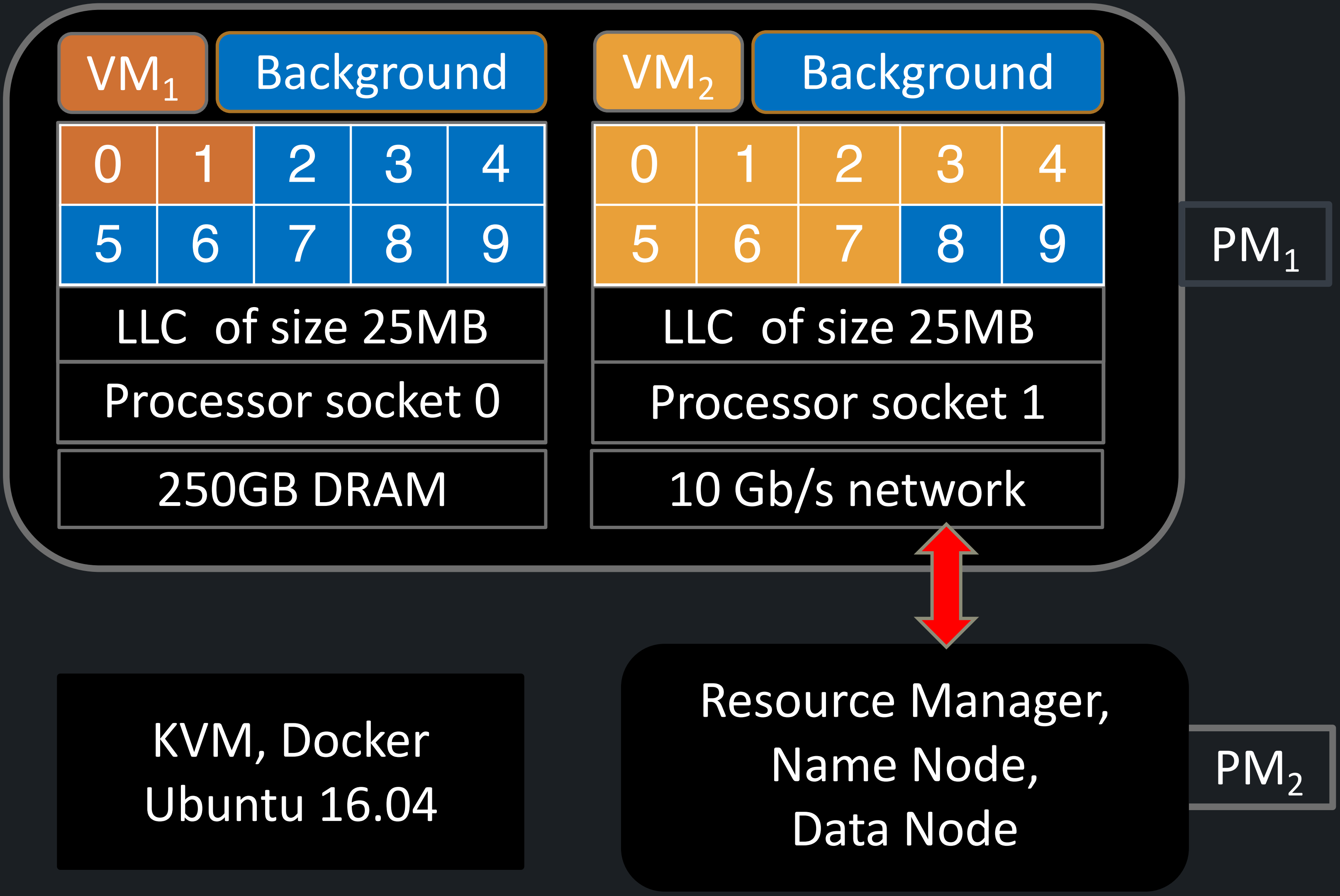
TailBench

The load generators employed in TailBench are open-loop.

Workload	Domain	Tail latency scale
Xapian	Online search	Milliseconds
Moses	Real-time translation	Milliseconds
Silo	In-memory database (OLTP)	Microseconds
Specjbb	Java middleware	Microseconds
Masstree	Key-value store	Microseconds
Shore	On-disk database (OLTP)	Milliseconds
Sphinx	Speech recognition	Seconds
Img-dnn	Image recognition	Milliseconds

http://people.csail.mit.edu/sanchez/papers/2016.tailbench.iiswc.pdf

Cloud Testbed

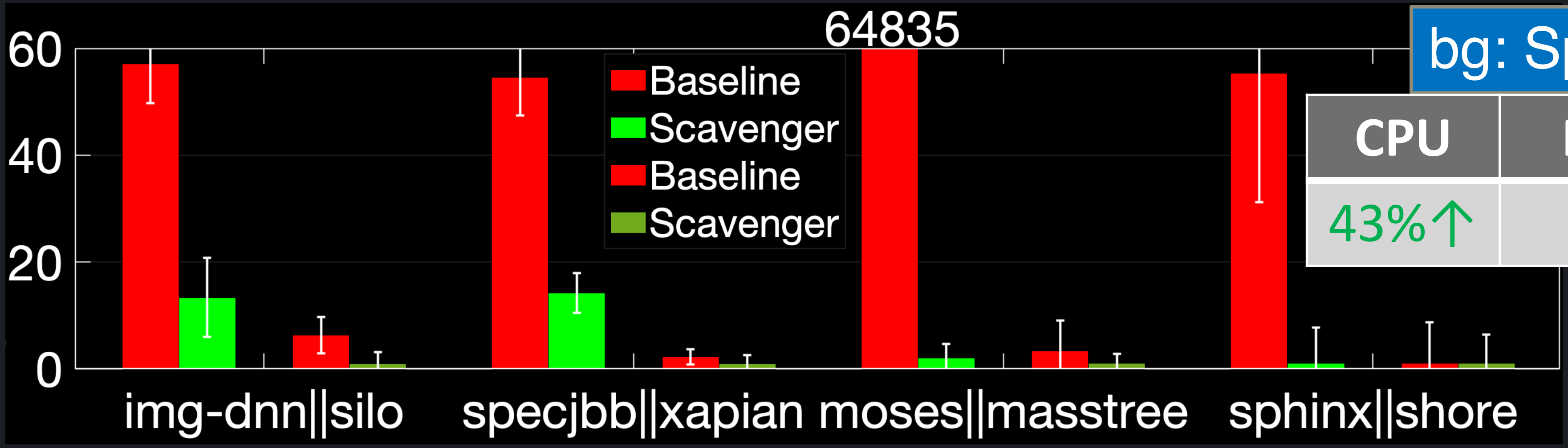


Evaluation with Spark jobs as background

VM$_1$ Workload || VM$_2$ Workload

95%ile latency degradation (%)

Better

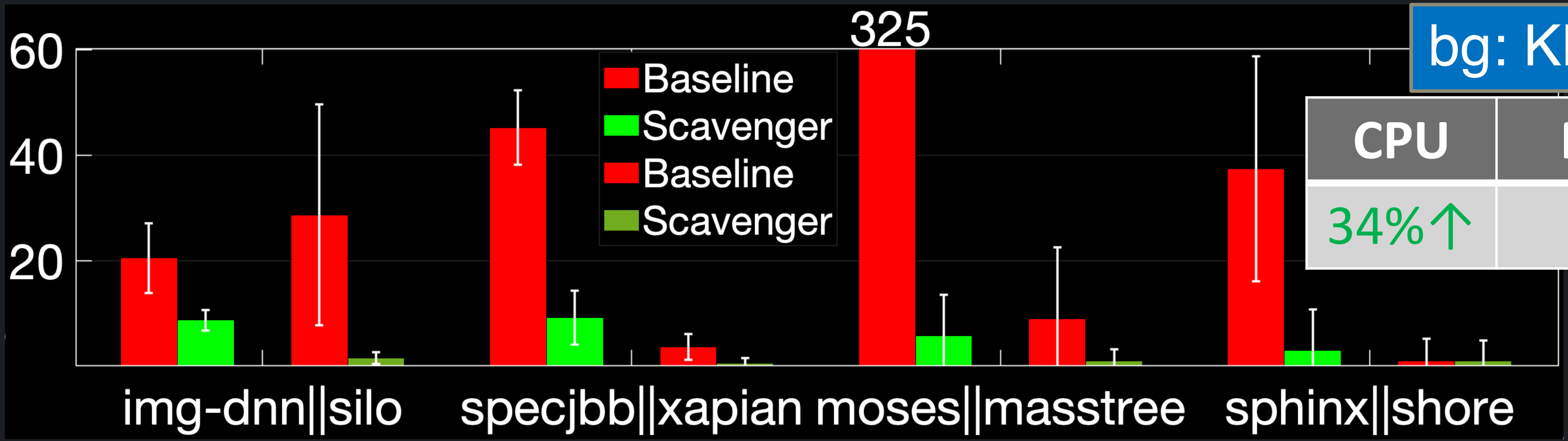


bg: SparkPi

CPU	Memory
43%↑	201%↑

95%ile latency degradation (%)

Better



bg: KMeans

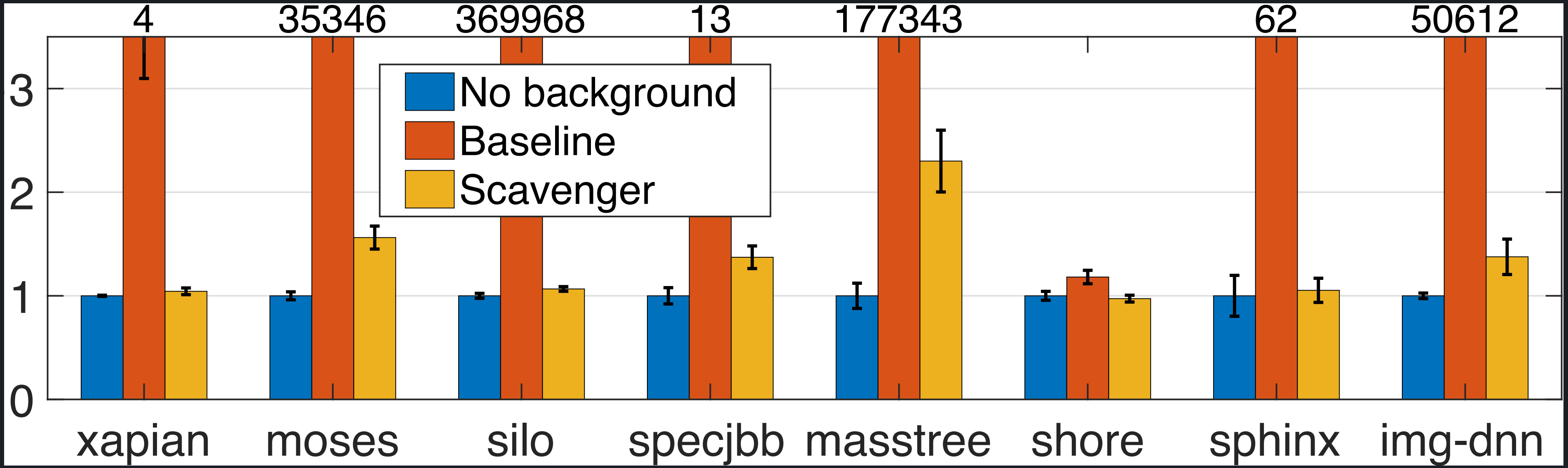
CPU	Memory
34%↑	321%↑

Limit Study With DCopy as the Background

Cloud testbed: 4-vCPU foreground VM, 6-core background DCopy container.

Normalized 95%ile latency

Better



3-5% CPU ↑

Scavenger can *successfully and aggressively* regulate bg workload to mitigate its impact on fg performance.

Outline

- ➢ DIAL: Dynamic interference-aware load balancing
- ➢ Scavenger: Resource-adaptive batch scheduling
- ➢ **Future directions and conclusions**

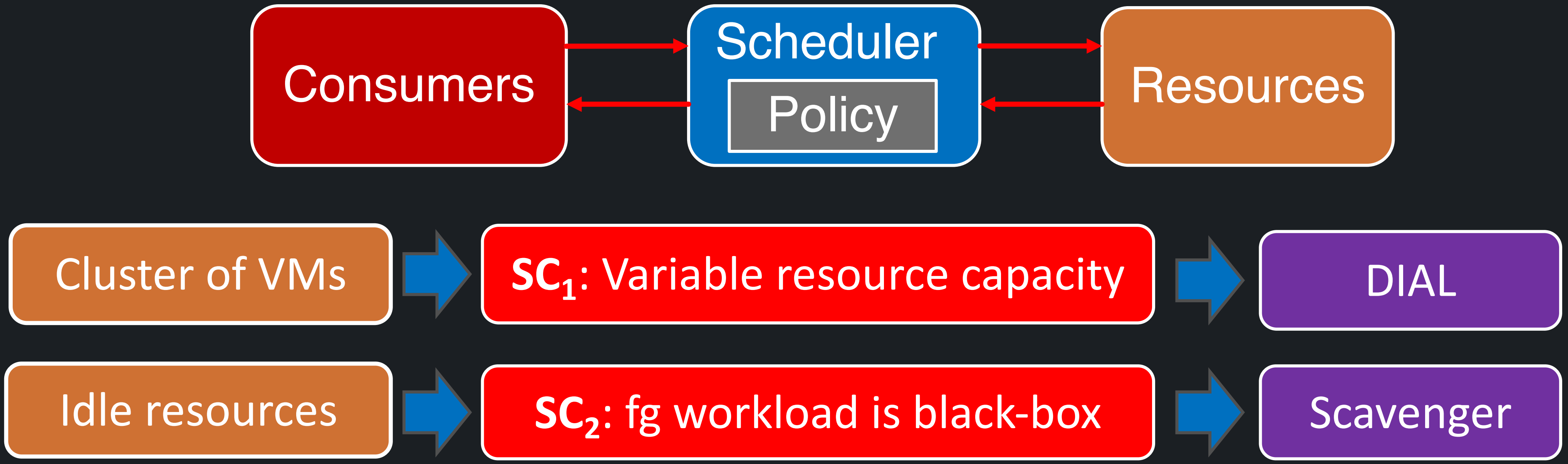
Future Direction

- Using Machine Learning (ML) techniques for
  - Predicting the resource demand of customers workloads
  - Tuning the solution parameters dynamically $(\mu + c(t).\sigma)$
  - ML deployment challenge
    - Easy and simple deployment in production systems

- Extending Scavenger for CAT-equipped servers
  - LLC allocation based on the Scavenger's regulation algorithm
  - Background workload will be allowed to use idle cores in full capacity

Conclusions

- Scheduling is a key component of applications
  - It faces new challenges in cloud environments
- Analytical approaches can address these challenges



Analytical Approaches for Dynamic Scheduling in Cloud Environments

Q&A

Seyyed Ahmad Javadi (sjavadi@cs.stonybrook.edu)

PACE Lab at Stony Brook University

25th International Computer Conference (CSICC 2020)