PerfIso: Performance Isolation for Commercial Latency-Sensitive Services

Călin Iorgulescu
EPFL

Reza Azimi
Brown University

Youngjin Kwon
University of Texas

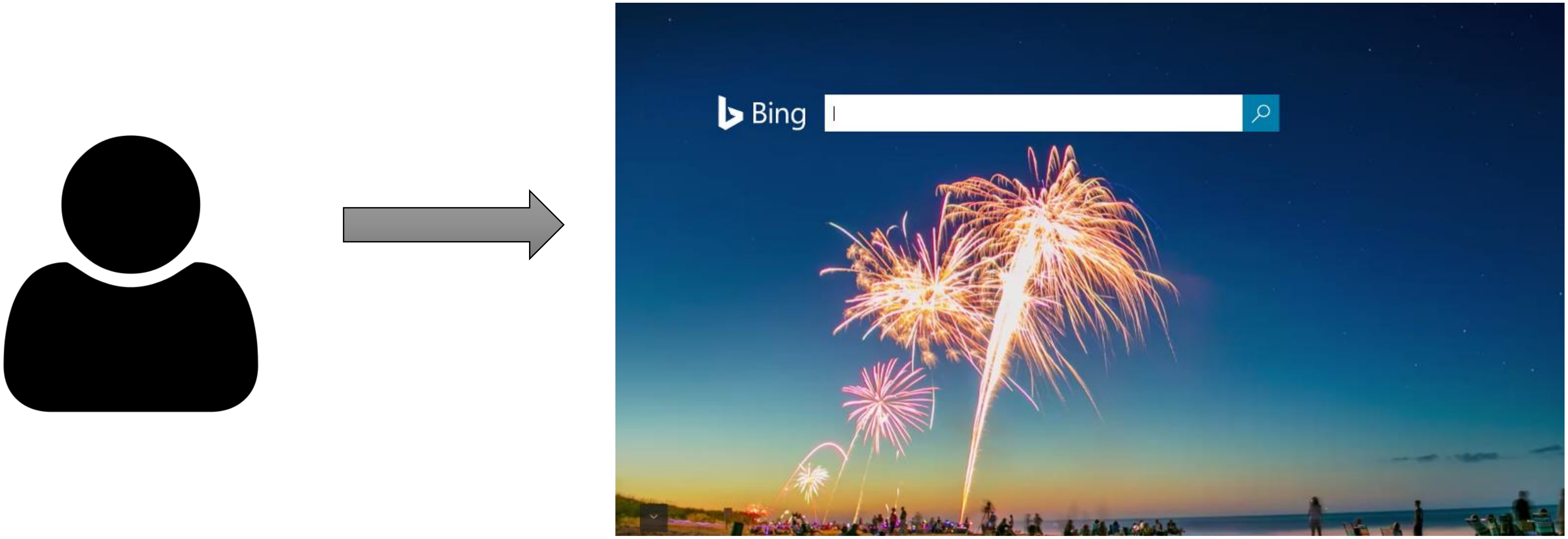
Sameh Elnikety, Manoj Syamala, Vivek Narasayya
Microsoft Research

Herodotus Herodotou
Cyprus University of Technology

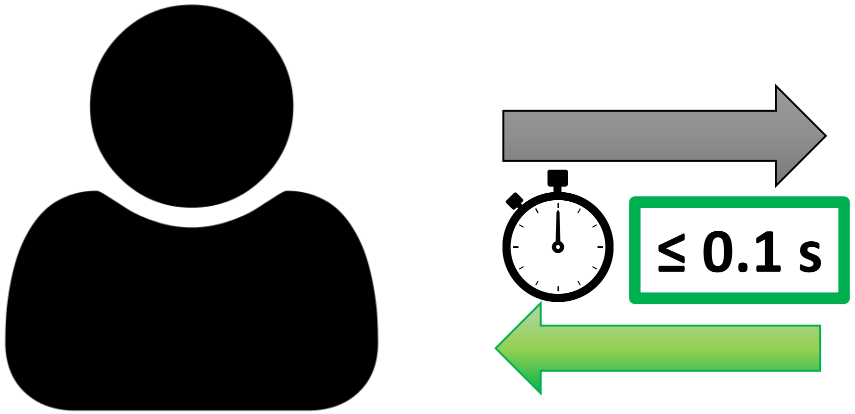
Paulo Tomita, Alex Chen, Jack Zhang, Junhua Wang
Microsoft Bing

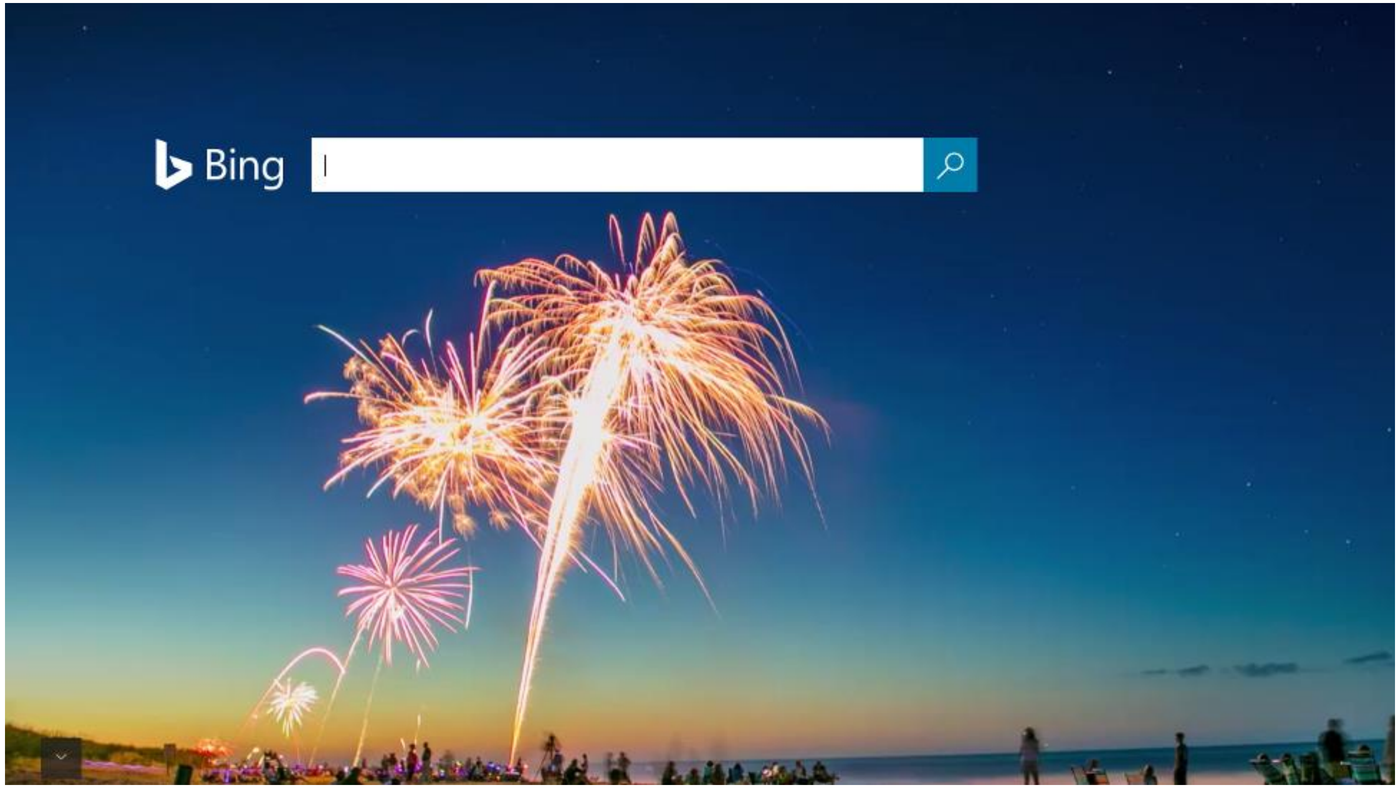
Interactive services must feel instantaneous

Interactive services must feel <u>instantaneous</u>

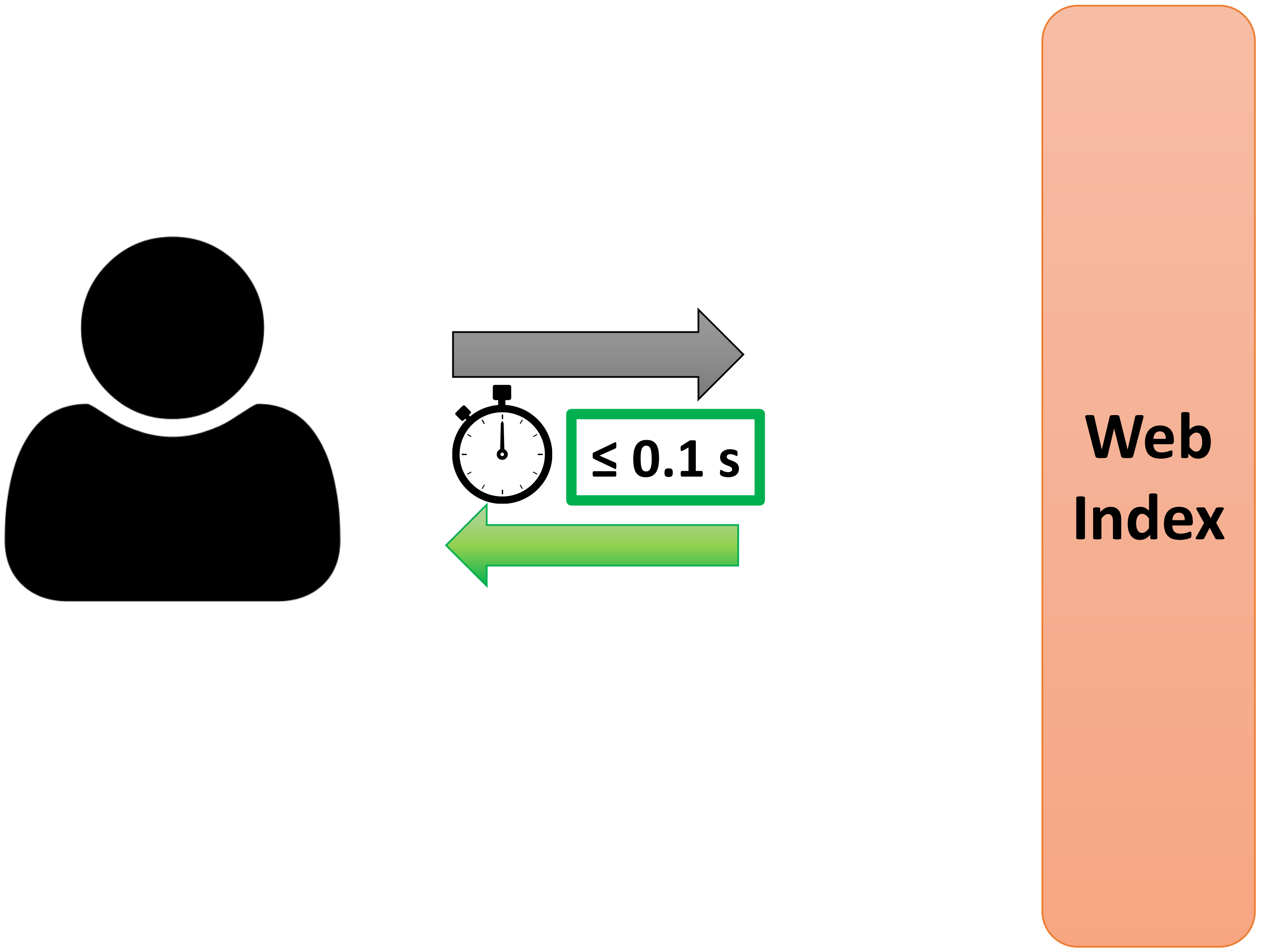


Interactive services must feel <u>instantaneous</u>

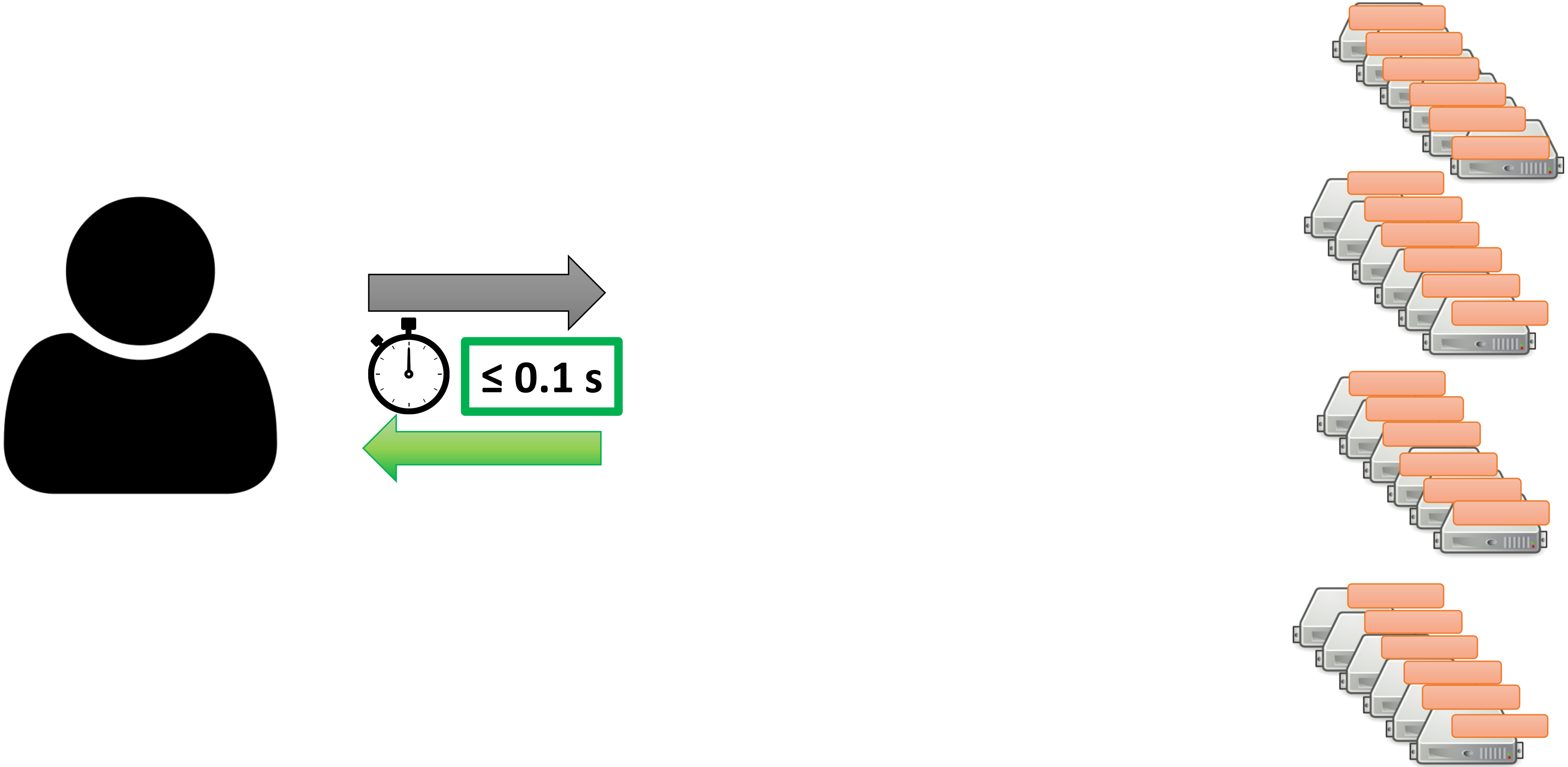




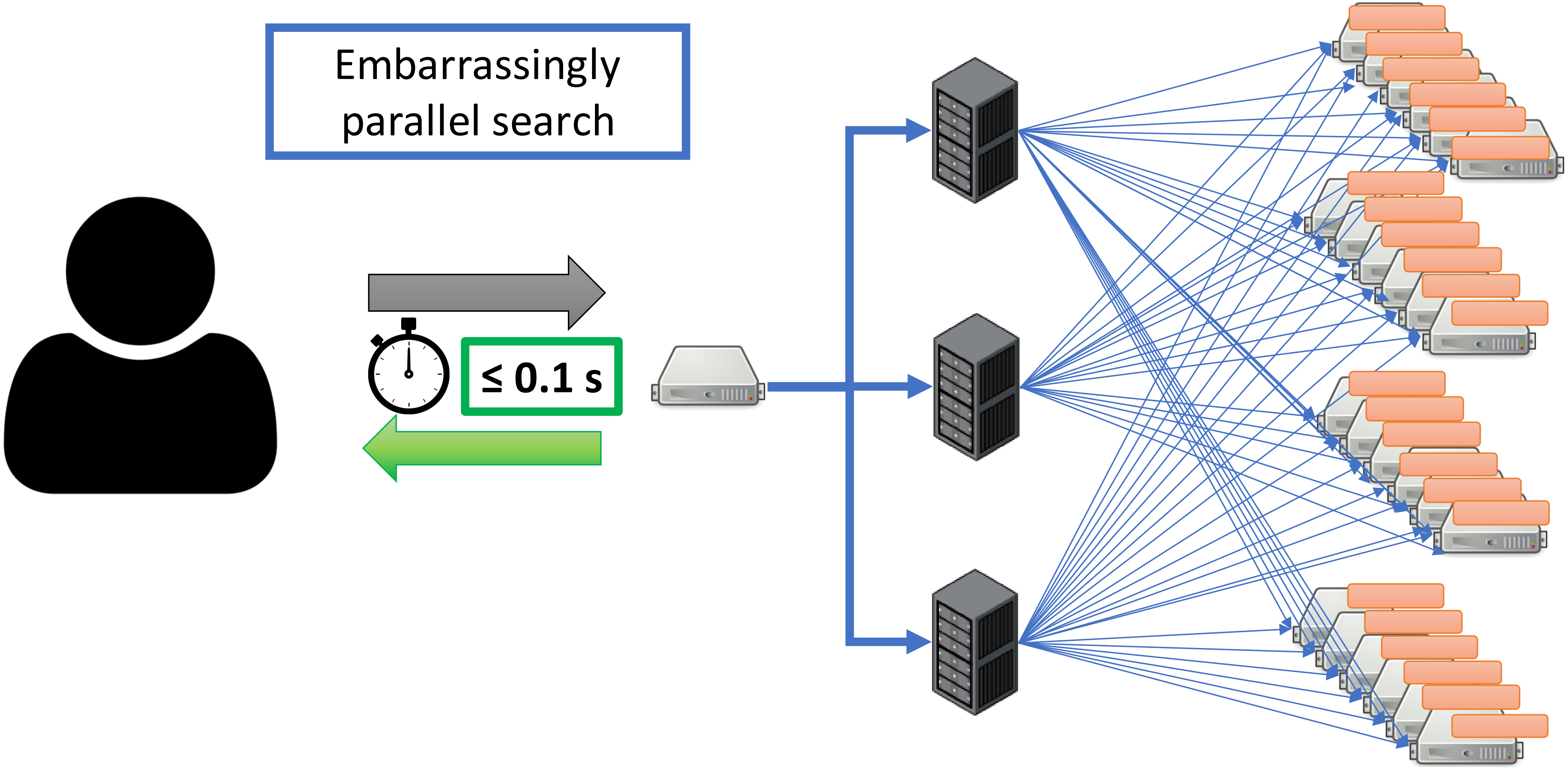
A single query involves hundreds of machines!



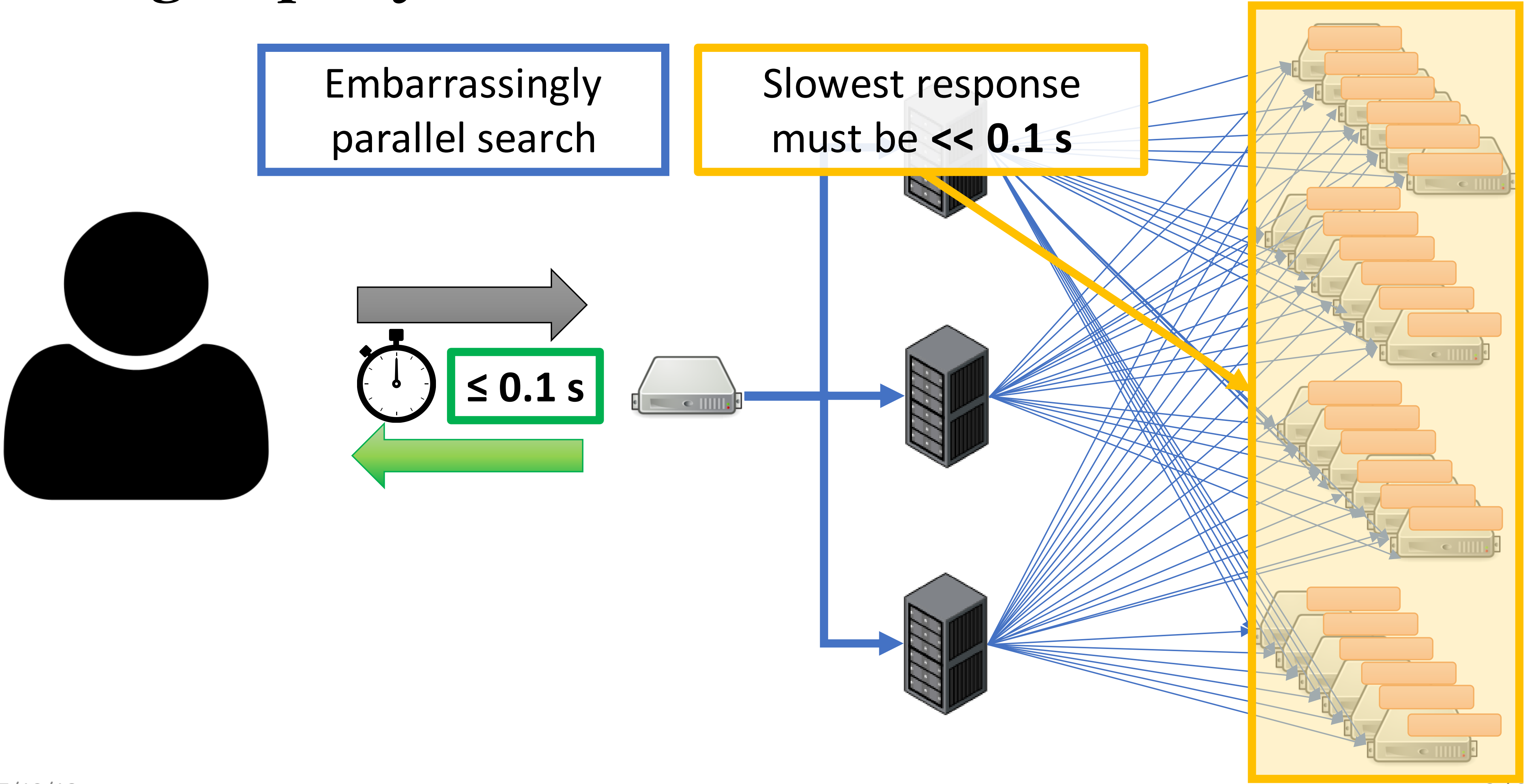
A single query involves hundreds of machines!



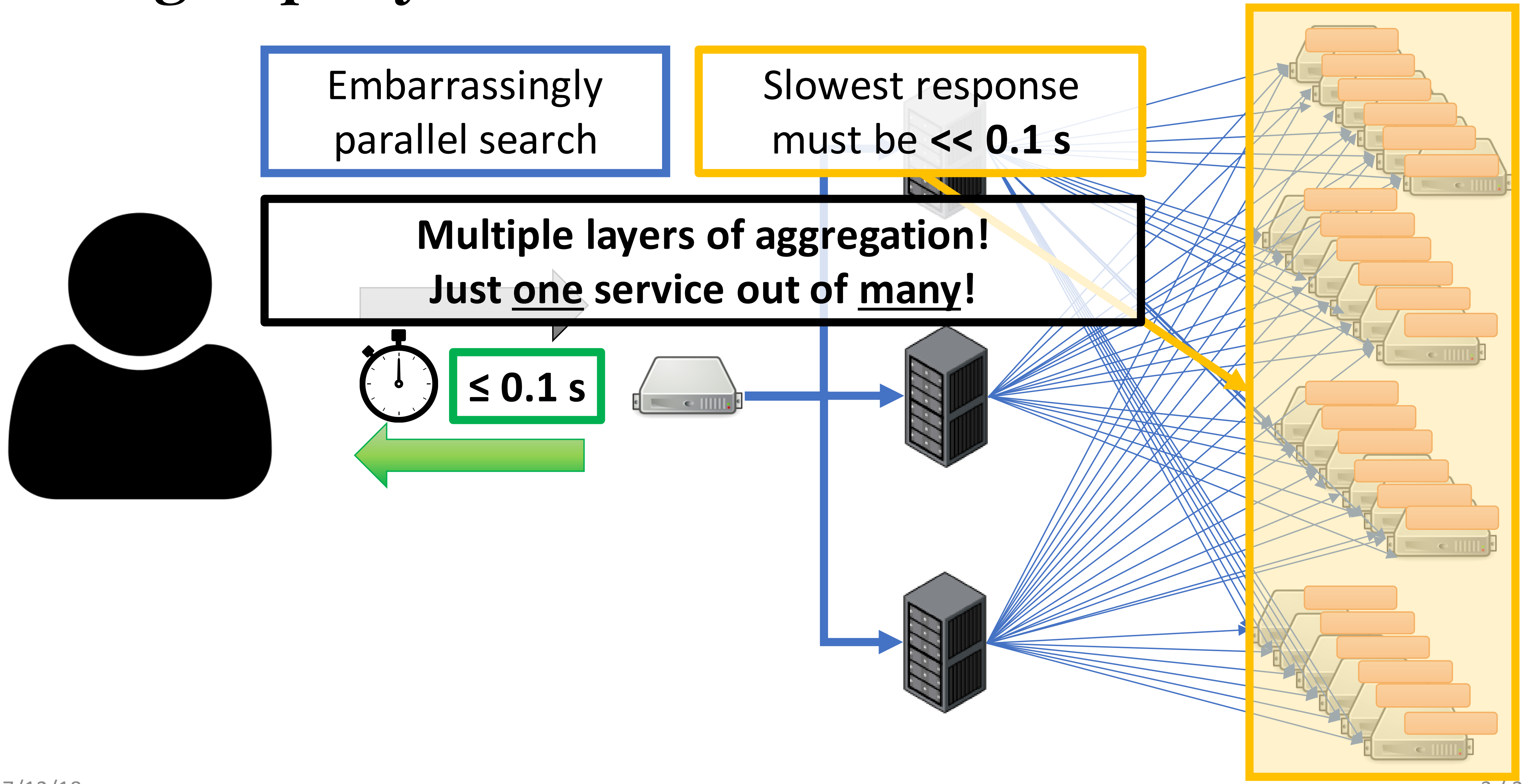
A single query involves hundreds of machines!



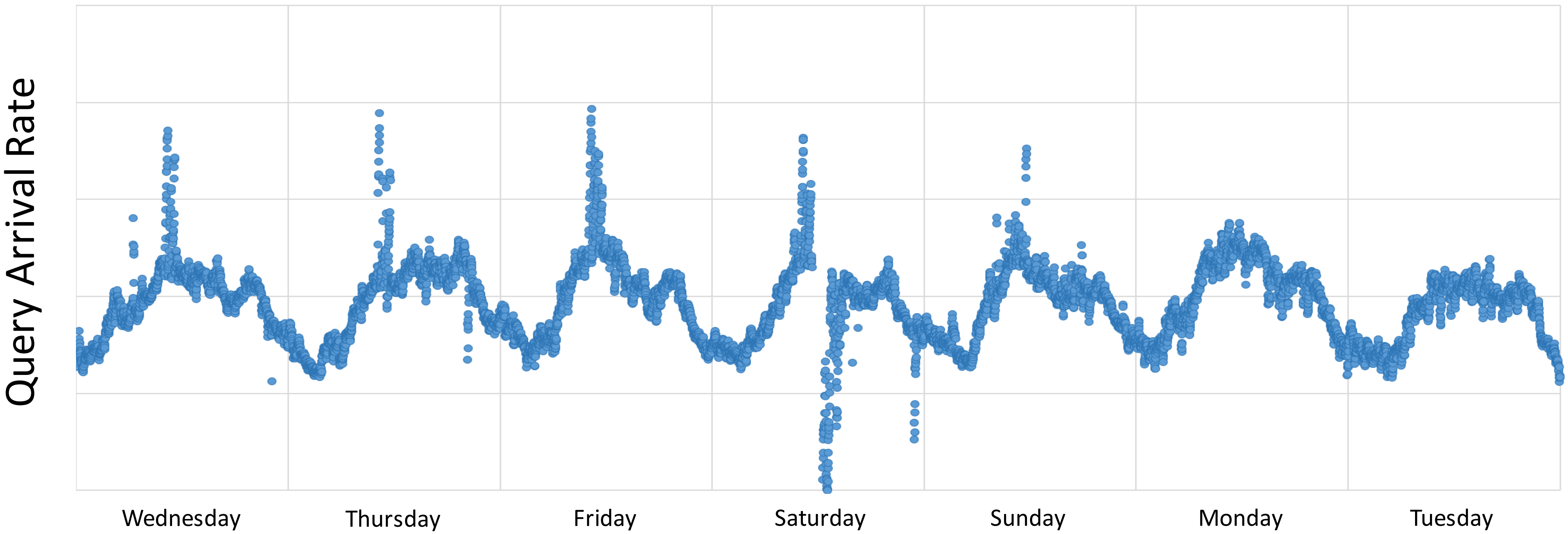
A single query involves hundreds of machines!



A single query involves hundreds of machines!

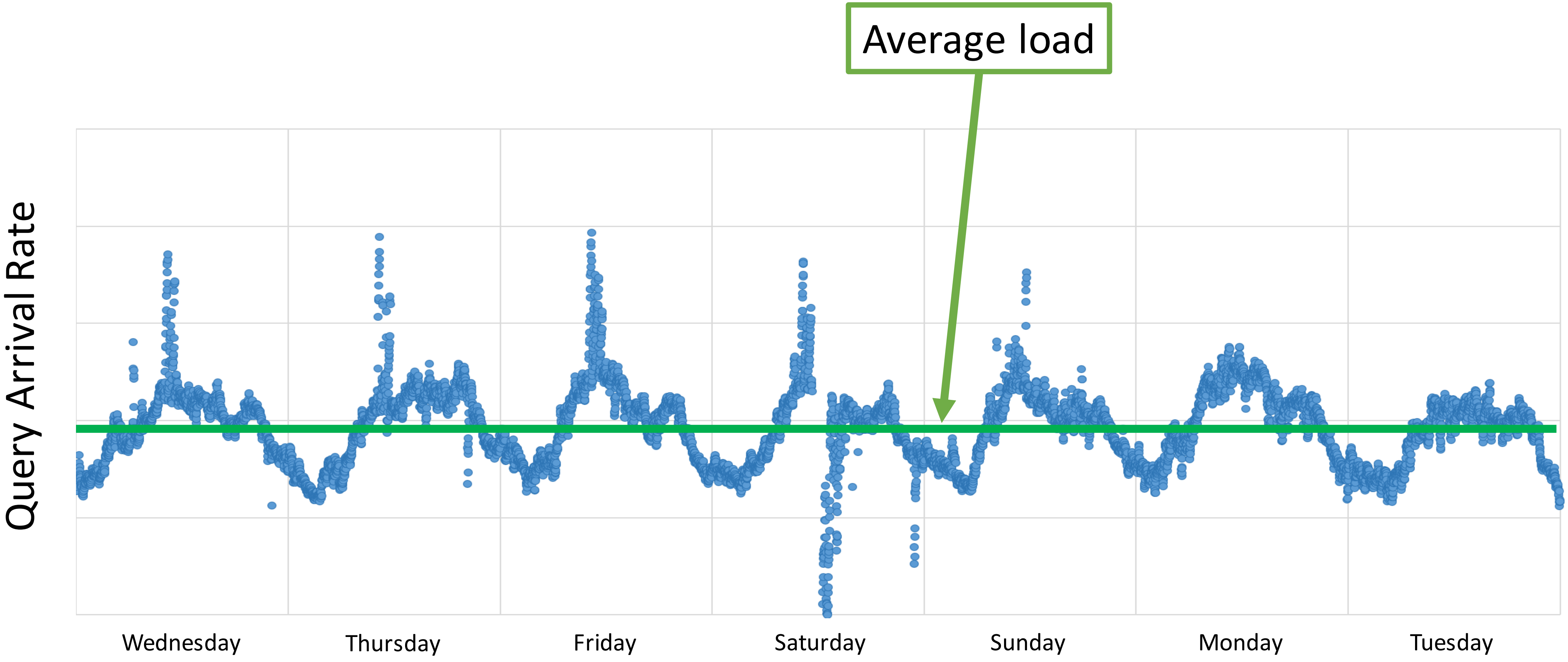


Machines are provisioned for peak load



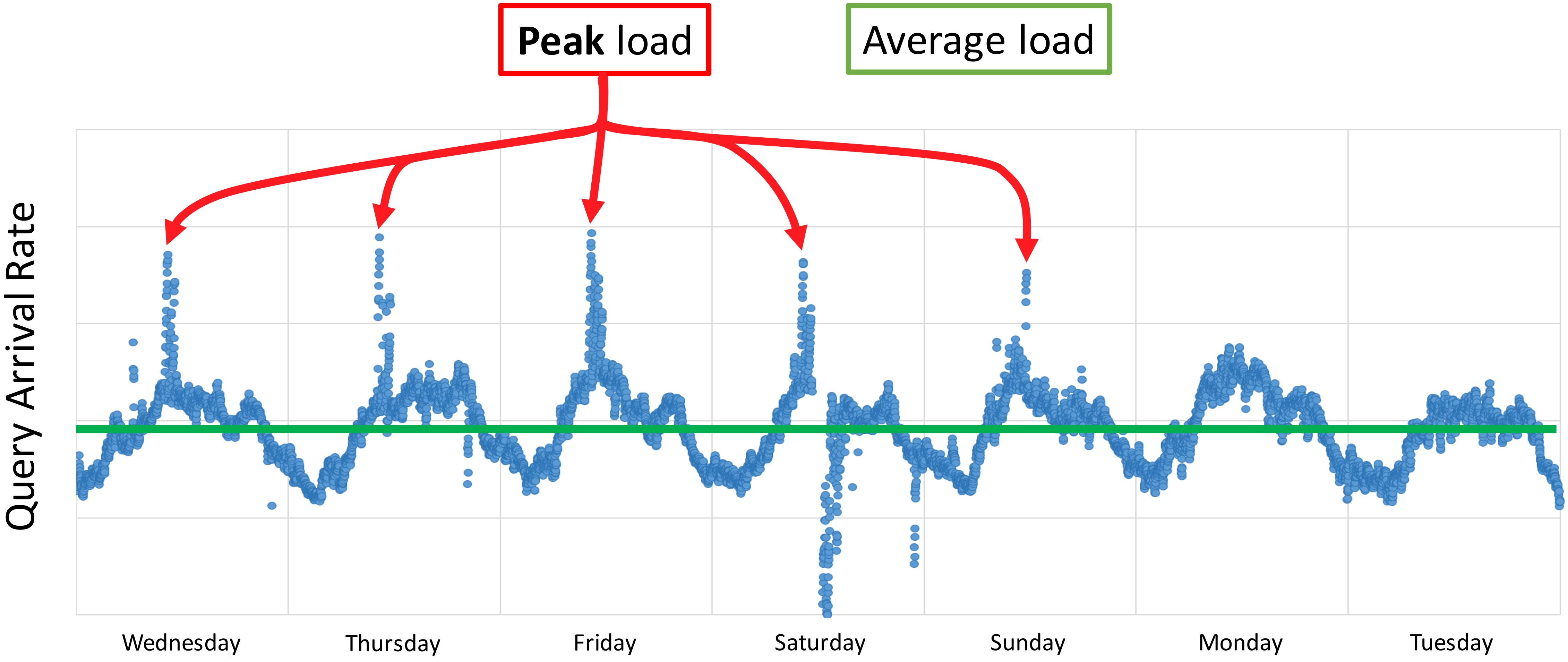
Request-rate variation for a *Microsoft Bing* sub-cluster over 1 week in 2017

Machines are provisioned for peak load



Request-rate variation for a *Microsoft Bing* sub-cluster over 1 week in 2017

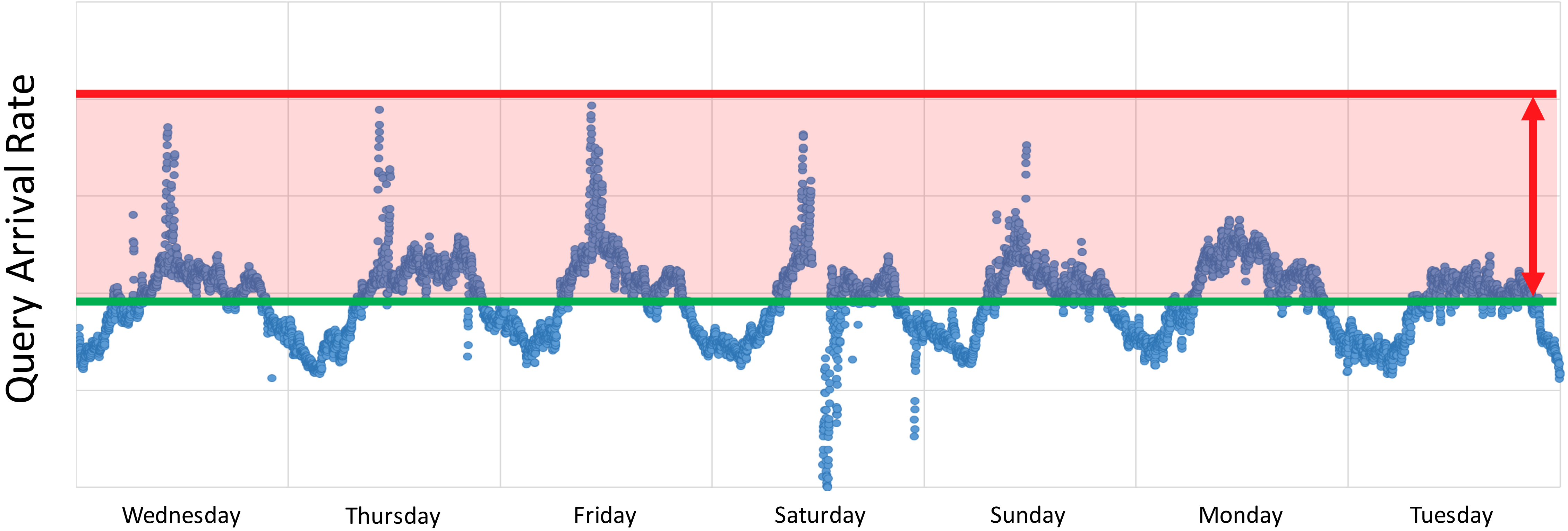
Machines are provisioned for peak load



Request-rate variation for a *Microsoft Bing* sub-cluster over 1 week in 2017

Machines are provisioned for peak load

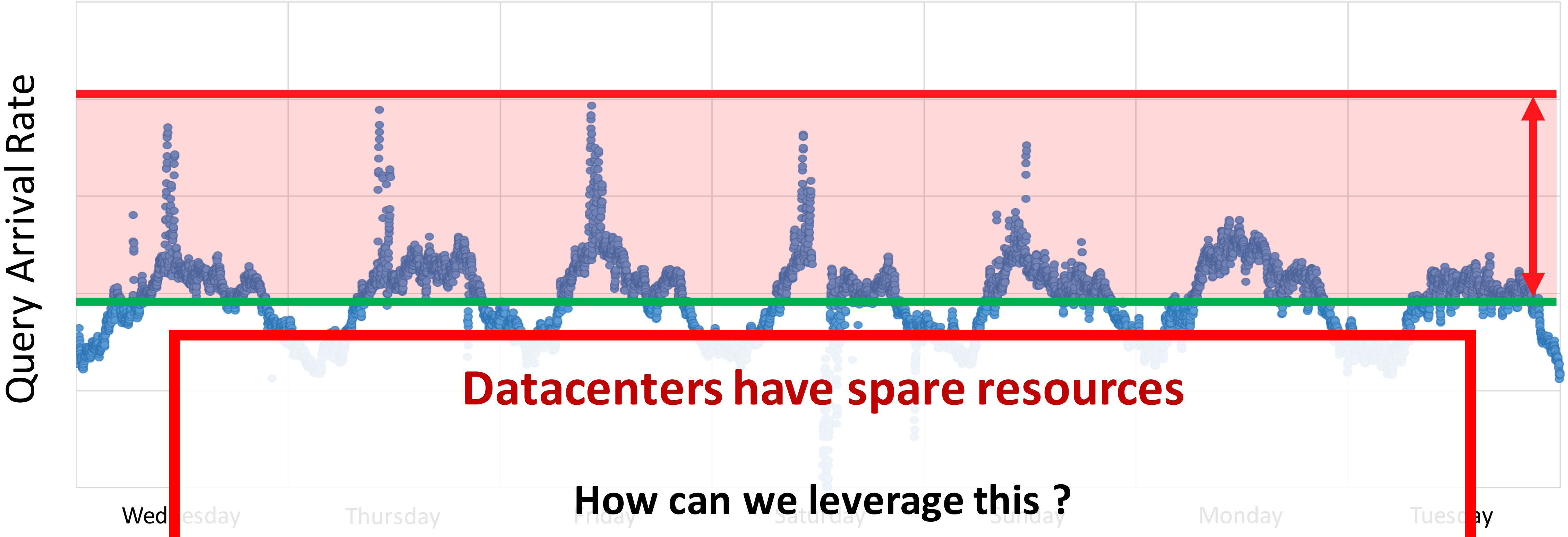
Peak load >> Average load



Request-rate variation for a *Microsoft Bing* sub-cluster over 1 week in 2017

Machines are provisioned for peak load

Peak load >> Average load



Datacenters have spare resources

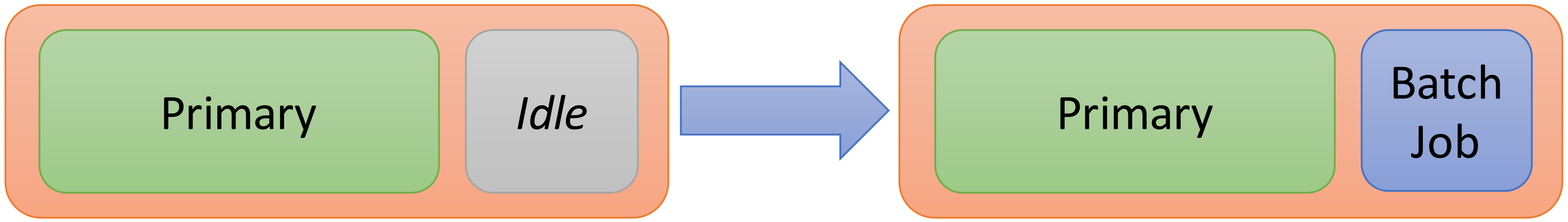
How can we leverage this ?

Request-rate variation for a *Microsoft Bing* sub-cluster over 1 week in 2017

Solution: colocate batch jobs with online services

- Get spare resources to do useful work
- **Primary** tenant – <u>guaranteed</u> performance
  - e.g., Bing IndexServe
- **Secondary** tenant – <u>best-effort</u> performance
  - e.g., Apache Spark

Without colocation *With colocation + **PerfIso***



PerfIso: performance isolation for online services

***PerfIso*: performance isolation for online services**

Provides **performance isolation** of Primary

- Maintains P99 of response-times (10s of ms) under colocation

PerfIso: performance isolation for online services

Provides **performance isolation** of Primary

- Maintains P99 of response-times (10s of ms) under colocation

Increases **system efficiency**

- 45% of the CPU is used to do useful batch work

PerfIso: performance isolation for online services

Provides performance isolation of Primary

- Maintains P99 of response-times (10s of ms) under colocation

Increases system efficiency

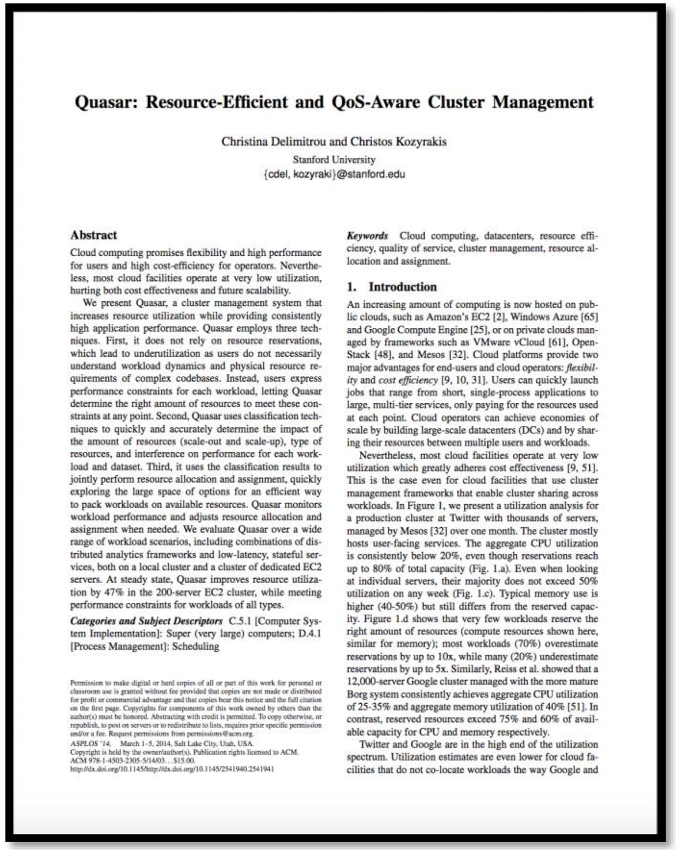
- 45% of the CPU is used to do useful batch work

Deployed on over 90,000 servers

- Many different interactive services and hardware setups

Many papers published on performance isolation

Quasar [ASPLOS '14]

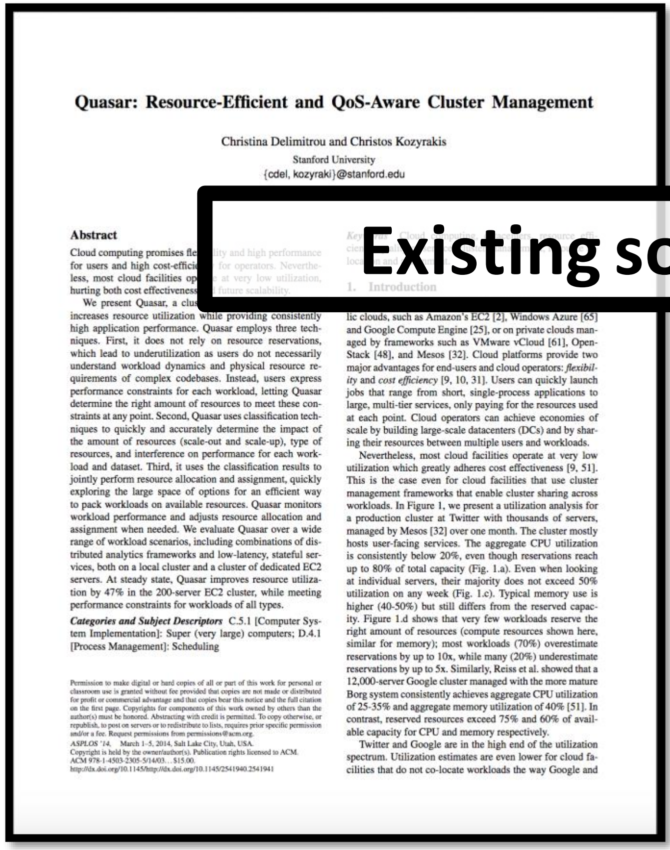


Heracles [ISCA '15]

Elfen [USENIX ATC '16]

Many papers published on performance isolation

Quasar [ASPLOS '14] **Heracles** [ISCA '15] **Elfen** [USENIX ATC '16]


Quasar: Resource-Efficient and QoS-Aware Cluster Management

Christina Delimitrou and Christos Kozyrakis
Stanford University
{cdel, kozyraki}@stanford.edu


Heracles: Improving Resource Efficiency at Scale

David Lo[†], Liqun Cheng[‡], Rama Govindaraju[‡], Parthasarathy Ranganathan[‡] and Christos Kozyrakis[†]
Stanford University[†] Google, Inc.[‡]


Elfen Scheduling: Fine-Grain Principled Borrowing from Latency-Critical Workloads using Simultaneous Multithreading

Xi Yang[†] Stephen M. Blackburn[†] Kathryn S. McKinley[‡]
[†]Australian National University [‡]Microsoft Research

Existing solutions do not fit our requirements

PerfIso: Requirements

PerfIso: Requirements

1. **“Black-box”: Fewest assumptions about tenants** (wider applicability)

PerfIso: Requirements

1. **“Black-box”: Fewest assumptions about tenants** (wider applicability)

2. **“Standalone”: Primary acts like it runs alone** (negligible interference)

PerfIso: Requirements

1. **"Black-box": Fewest assumptions about tenants** (wider applicability)

2. **"Standalone": Primary acts like it runs alone** (negligible interference)

3. **"Integrability": Minimize software-stack changes** (easy deployment)

Why is Performance Isolation hard?

Interactive services – highly sensitive to interference!

Leaf-servers keep 99th percentile low

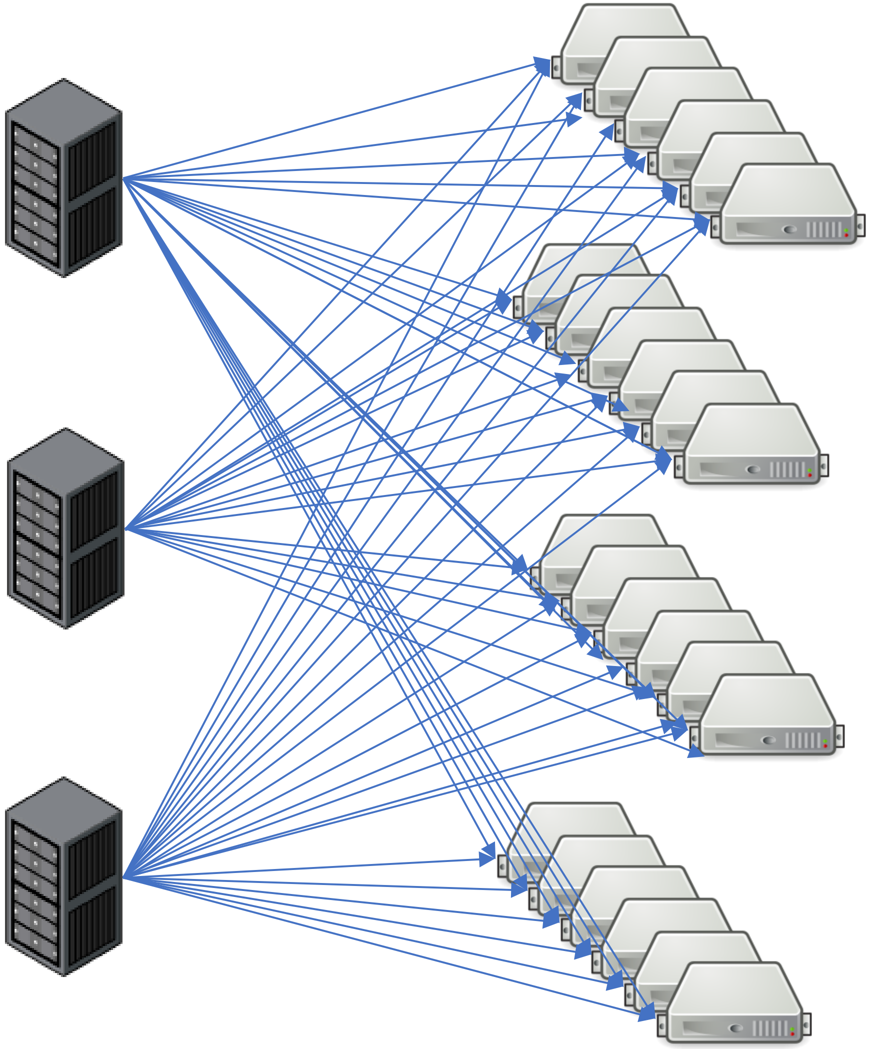
- Over 10 years of optimization work!
  - e.g., compression, adaptive parallelism, etc.

How often does the 99th percentile occur?

- For 10,000 queries / s → 100 times / s

What happens in a 100-node fanout?

- <u>Every query runs at the 99th percentile</u>!



The Primary demands many resources quickly

- **Bing IndexServe**: multi-threaded web-index server

➢ Up to 15 threads wake up in $5\mu s$[1]

[1]Constant query rate 4,000 Q/s, 500k queries experiment

The Primary demands many resources quickly

- **Bing IndexServe**: multi-threaded web-index server

➢ Up to 15 threads wake up in $5\mu s$[1]

- **Burstiness due to query-processing optimizations!**
  - some queries will spawn many workers

[1]Constant query rate 4,000 Q/s, 500k queries experiment

The Primary demands many resources quickly

- **Bing IndexServe**: multi-threaded web-index server

➢ <u>Up to 15 threads wake up in $5\mu s$</u>[1]

- **Burstiness due to query-processing optimizations!**
  - some queries will spawn many workers

- **Workload arrives in bursts – exacerbates problem**

[1]Constant query rate 4,000 Q/s, 500k queries experiment

The Primary must behave as if it were *standalone*

The Primary must behave as if it were *standalone*

- Primary's resource demands must be <u>fulfilled instantly</u>.

The Primary must behave as if it were *standalone*

- Primary's resource demands must be fulfilled instantly.
- Any delays → performance penalties incurred

The Primary must behave as if it were *standalone*

- Primary's resource demands must be <u>fulfilled instantly</u>.
- Any delays → <u>performance penalties</u> incurred
- Any resource can become a <u>performance bottleneck</u>.

The Primary must behave as if it were *standalone*

- Primary's resource demands must be <u>fulfilled instantly</u>.

- Any delays → <u>performance penalties</u> incurred

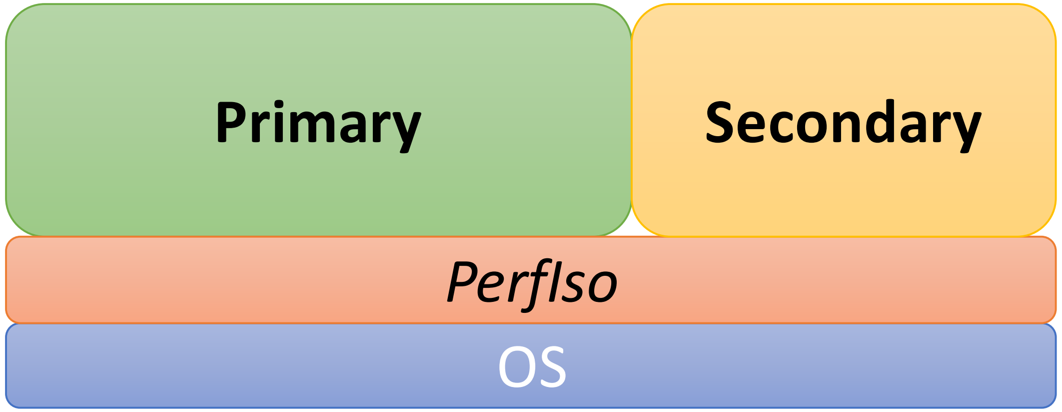
- Any resource can become a <u>performance bottleneck</u>.

If a query is delayed, it is already too late!

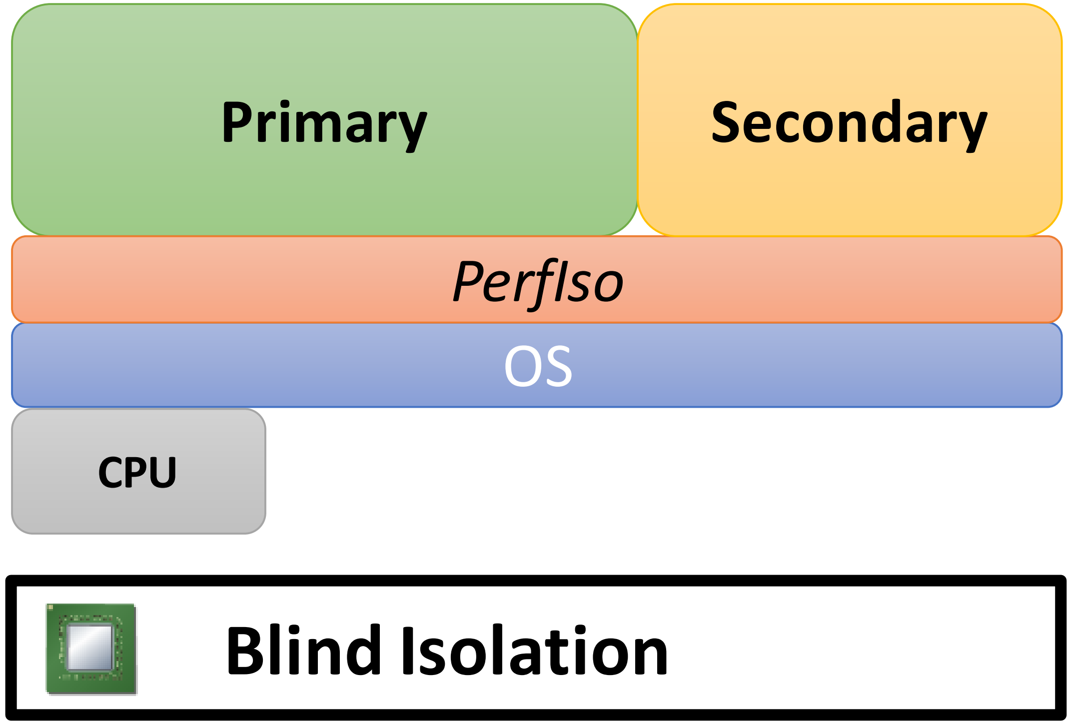
PerfIso

PerfIso: Implemented as a <u>user-mode</u> service

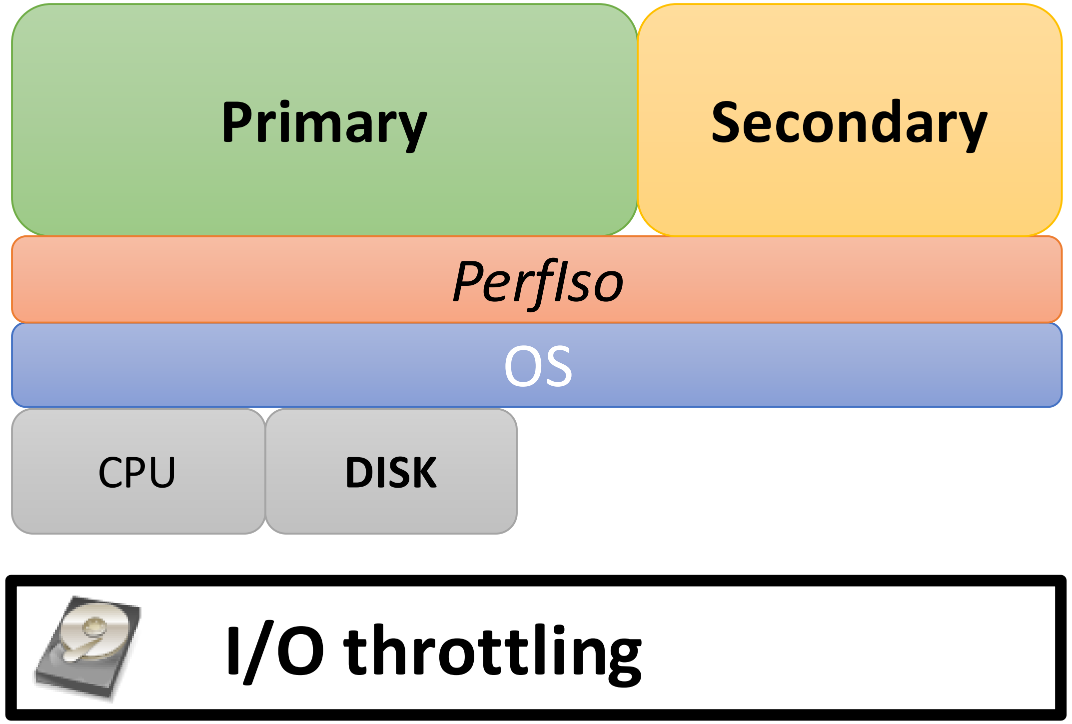
- Only keeps track of **Secondary**'s PID



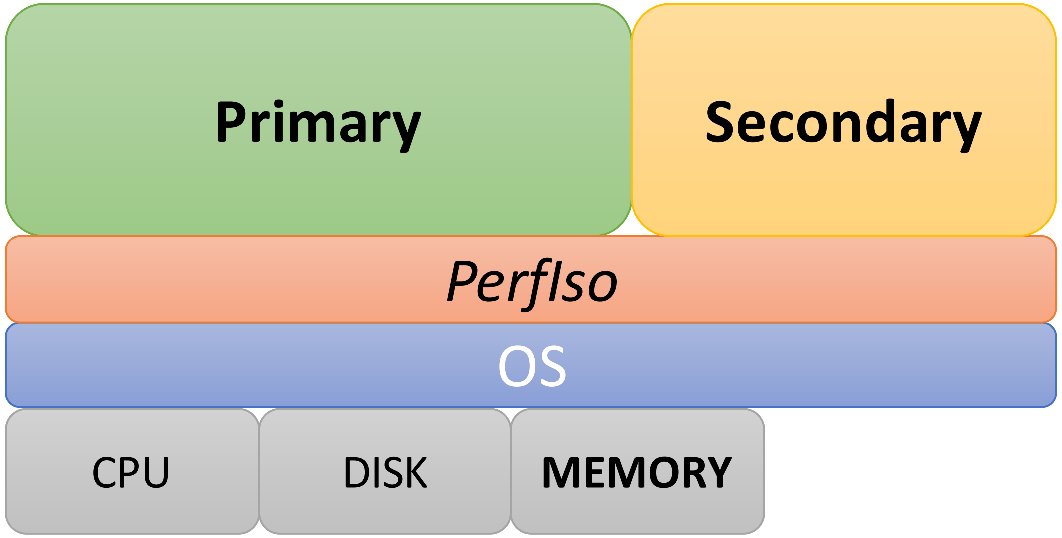
PerfIso: Managed resources



PerfIso: Managed resources

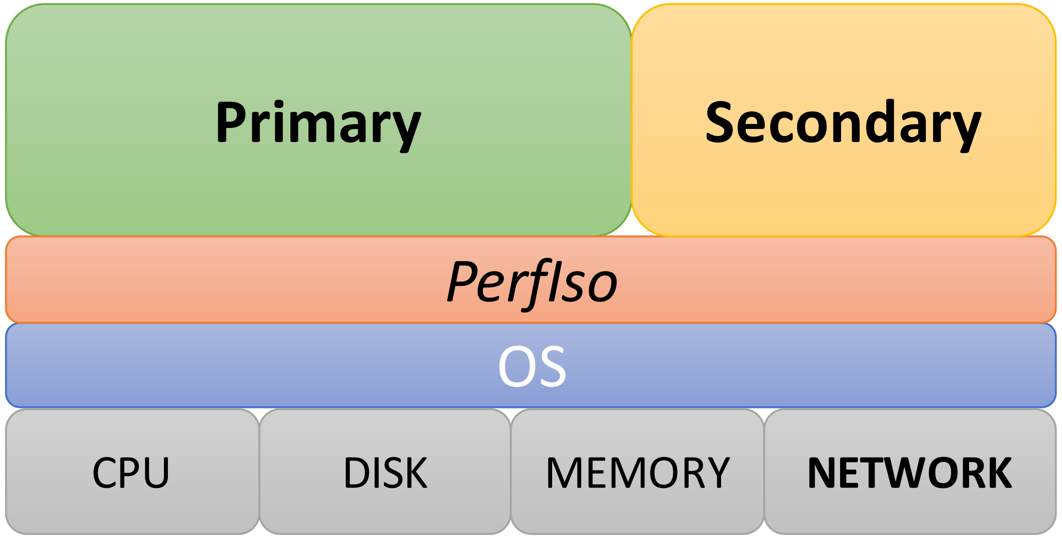


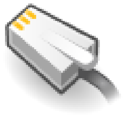
PerfIso: Managed resources



Restrict footprint

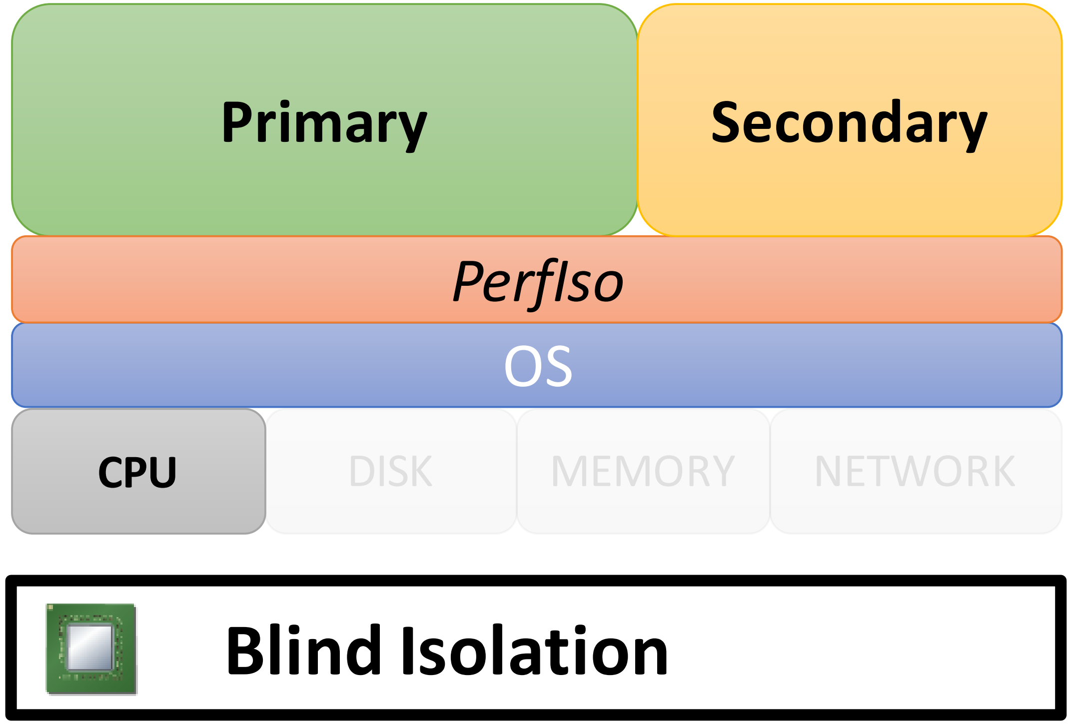
PerfIso: Managed resources





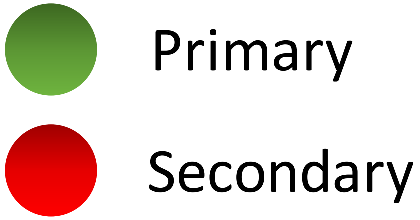
Throttle egress packets

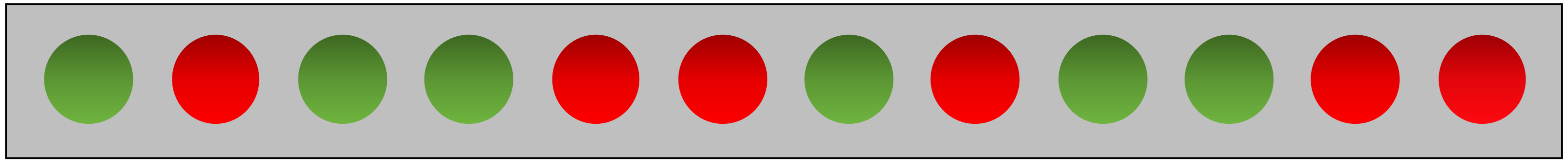
PerfIso: CPU is the most important resource



CPU sharing without *PerfIso*

- **Primary** and **Secondary** compete for cores.
- **Secondary** is aggressive: no idle cores exist.

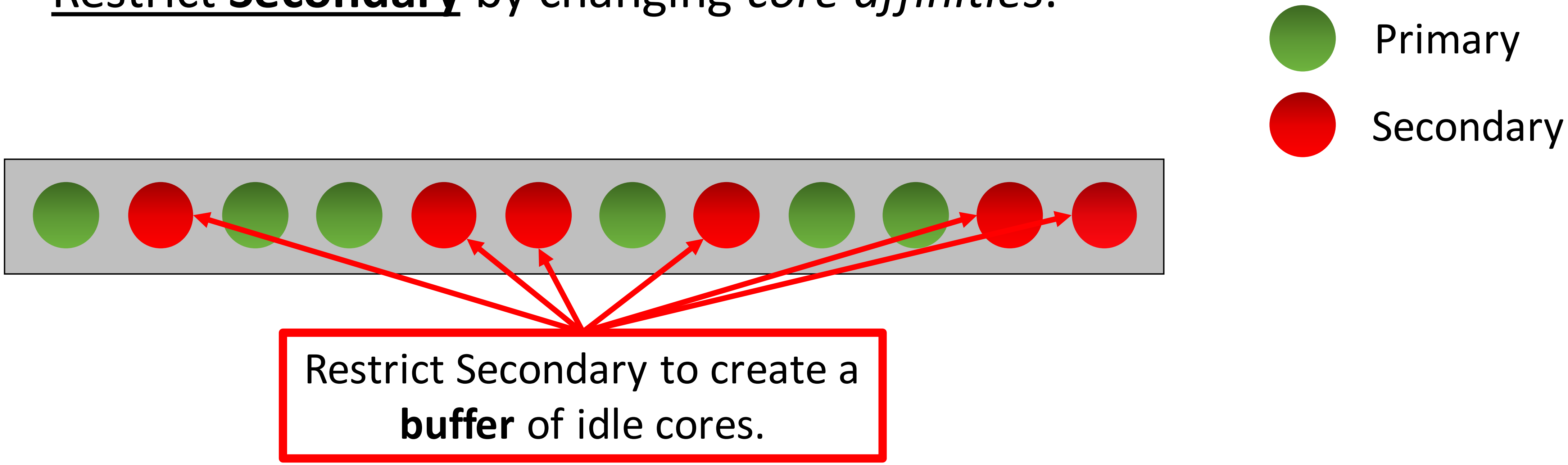




Machine with **12 cores**

CPU Blind Isolation: Keep a "buffer" of idle cores

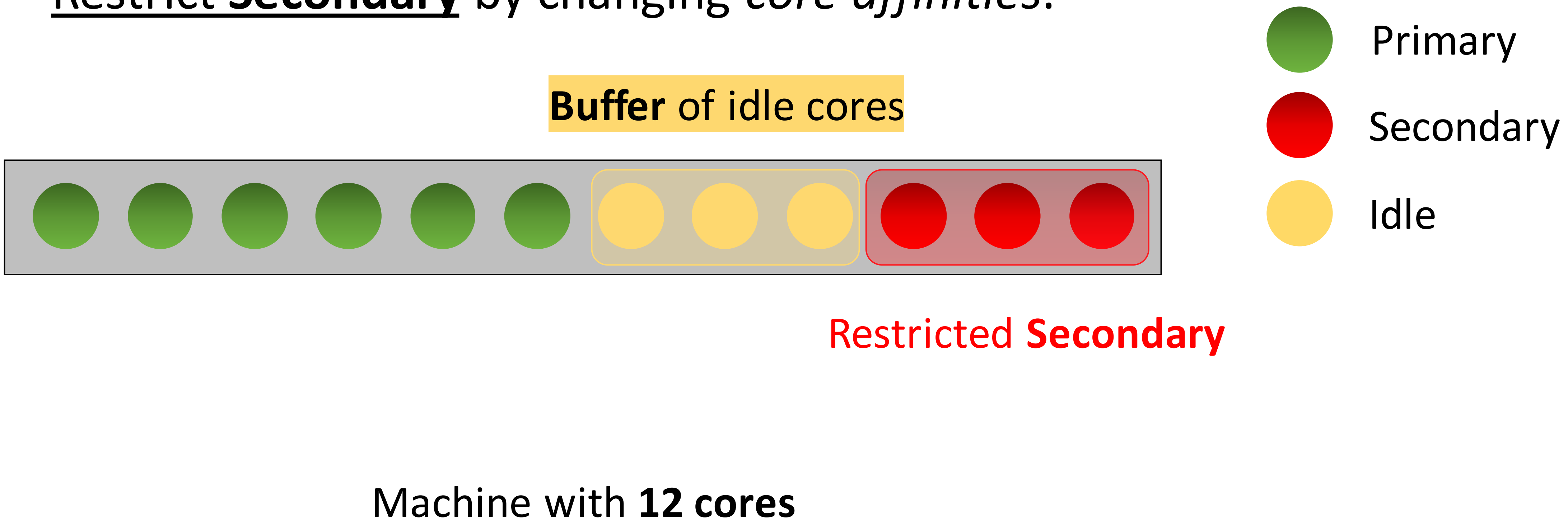
- *PerfIso* only knows the **Secondary**.
- Restrict **Secondary** by changing *core affinities*.



Machine with **12 cores**

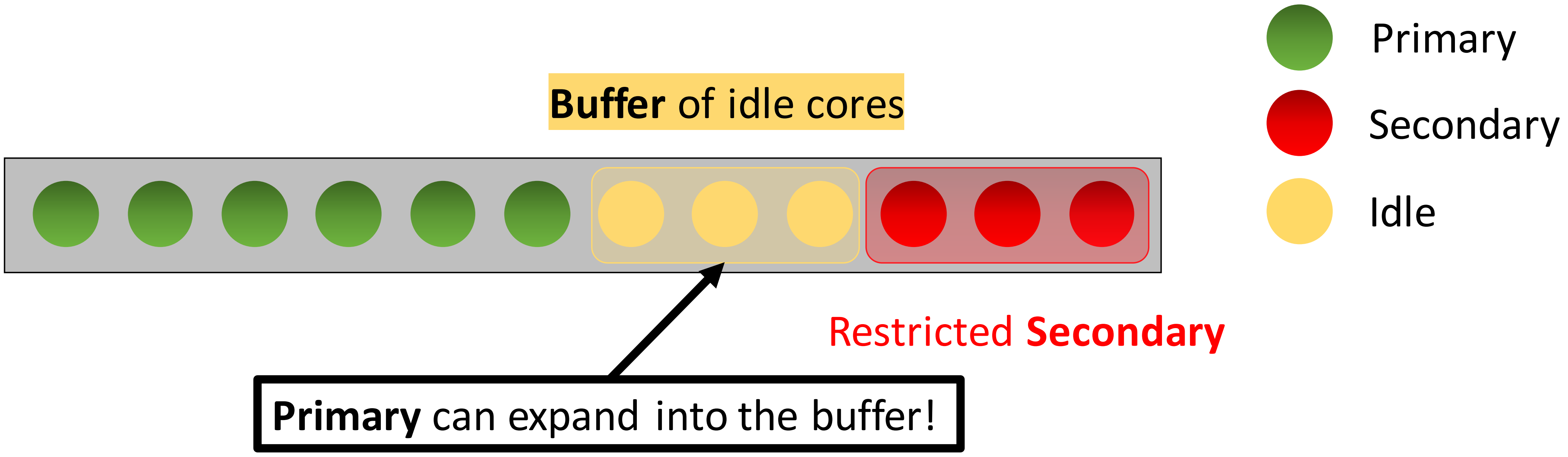
CPU Blind Isolation: Keep a "buffer" of idle cores

- *PerfIso* only knows the **Secondary**.
- <u>Restrict **Secondary**</u> by changing *core affinities*.



CPU Blind Isolation: Keep a "buffer" of idle cores

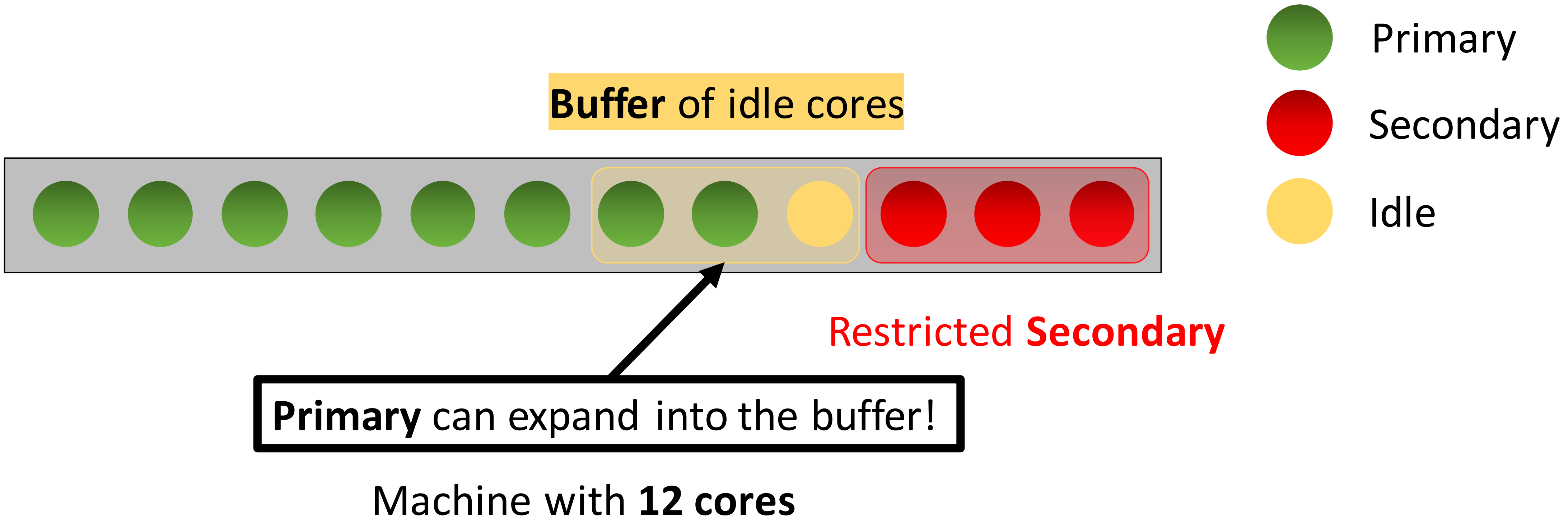
- **Primary** is unrestricted. **Secondary** is restricted.



Machine with **12 cores**

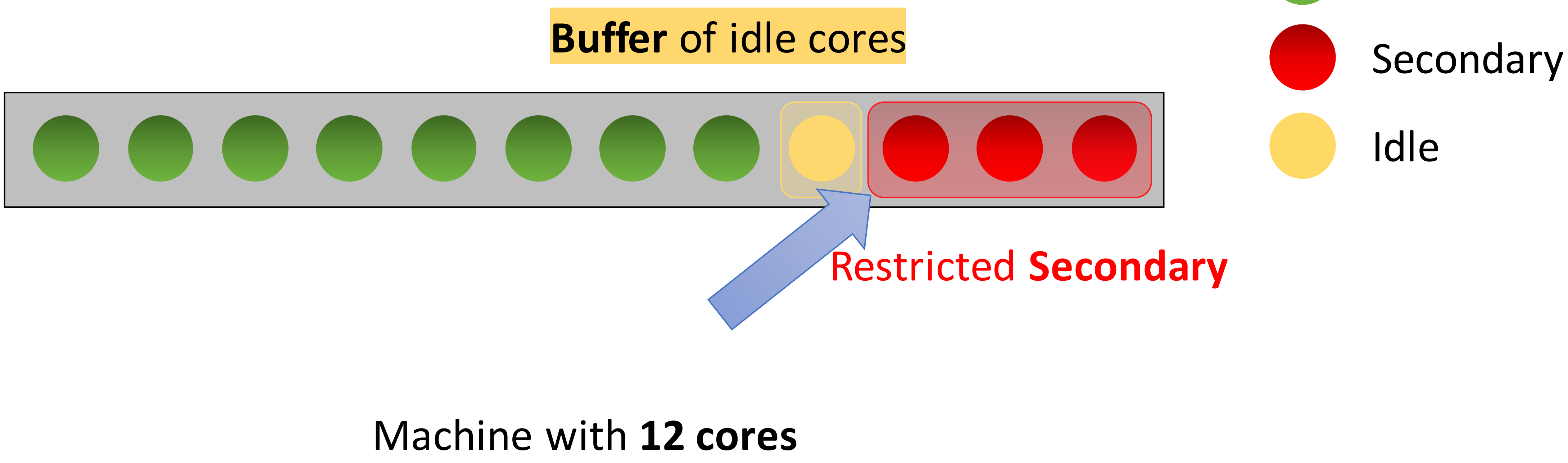
CPU Blind Isolation: Keep a "buffer" of idle cores

- **Primary** is <u>unrestricted</u>. **Secondary** is <u>restricted</u>.



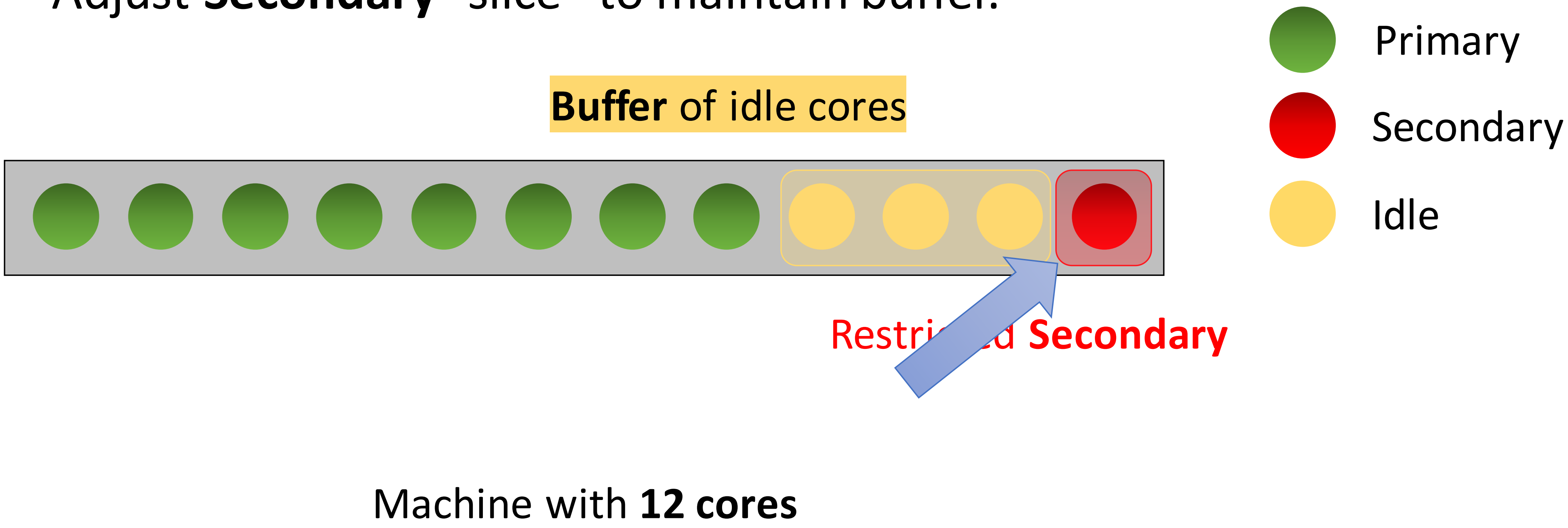
CPU Blind Isolation: React to bursts from Primary

- Continuously read idle core status.
- Adjust **Secondary** ”slice” to maintain buffer.



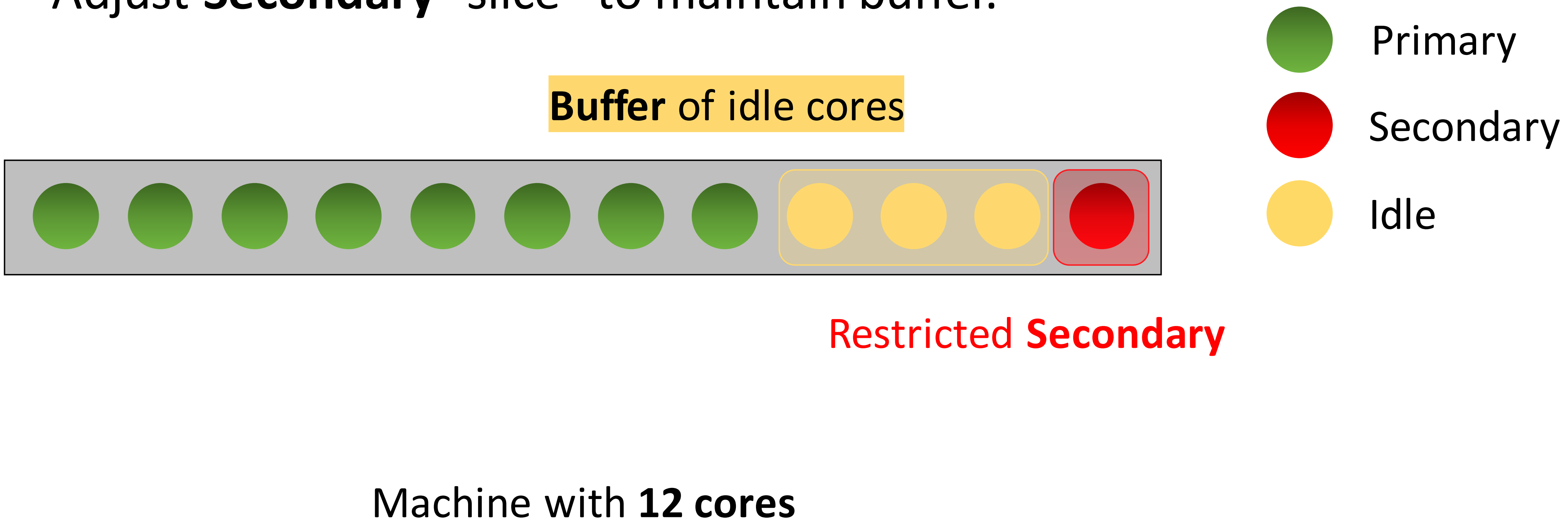
CPU Blind Isolation: React to bursts from Primary

- Continuously read idle core status.
- Adjust **Secondary** ”slice” to maintain buffer.



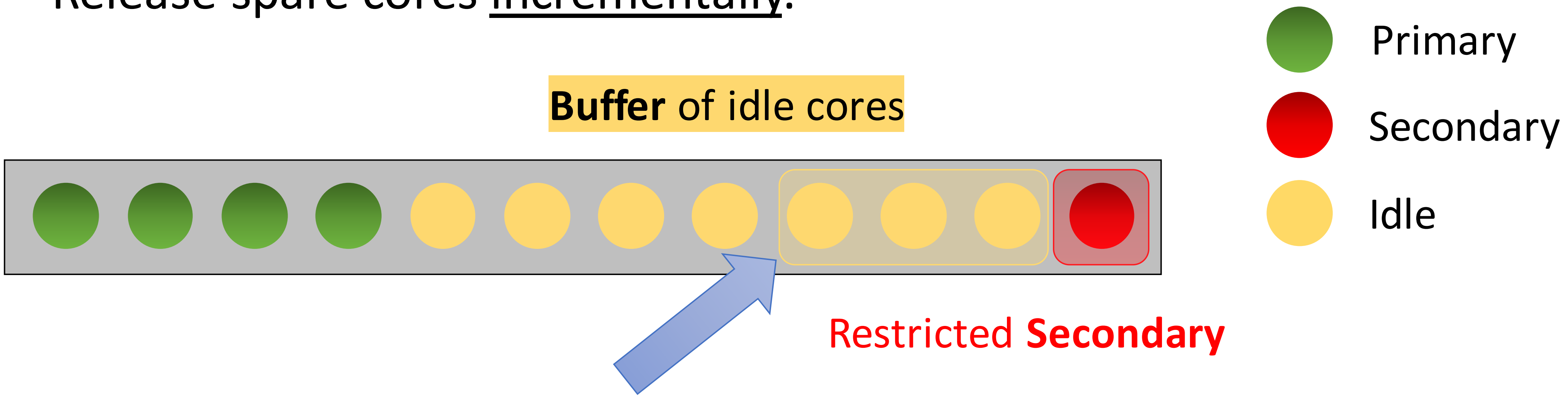
CPU Blind Isolation: React to bursts from Primary

- <u>Continuously</u> read idle core status.
- Adjust **Secondary** ”slice” to maintain buffer.



CPU Blind Isolation: Secondary gets spare cores

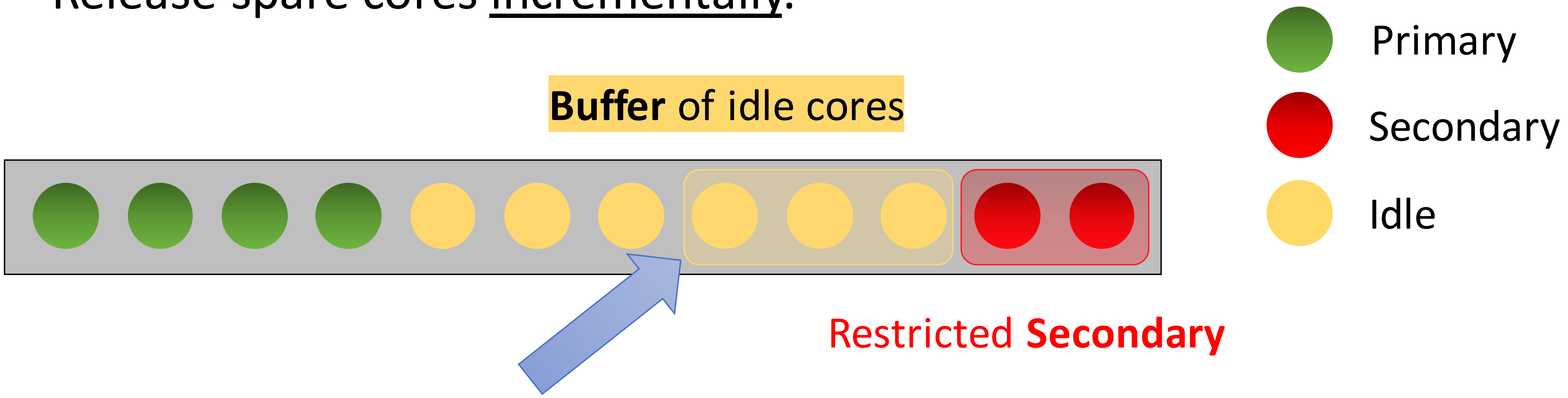
- Allow **Secondary** to use spare idle cores.
- Release spare cores incrementally.



Machine with **12 cores**

CPU Blind Isolation: Secondary gets spare cores

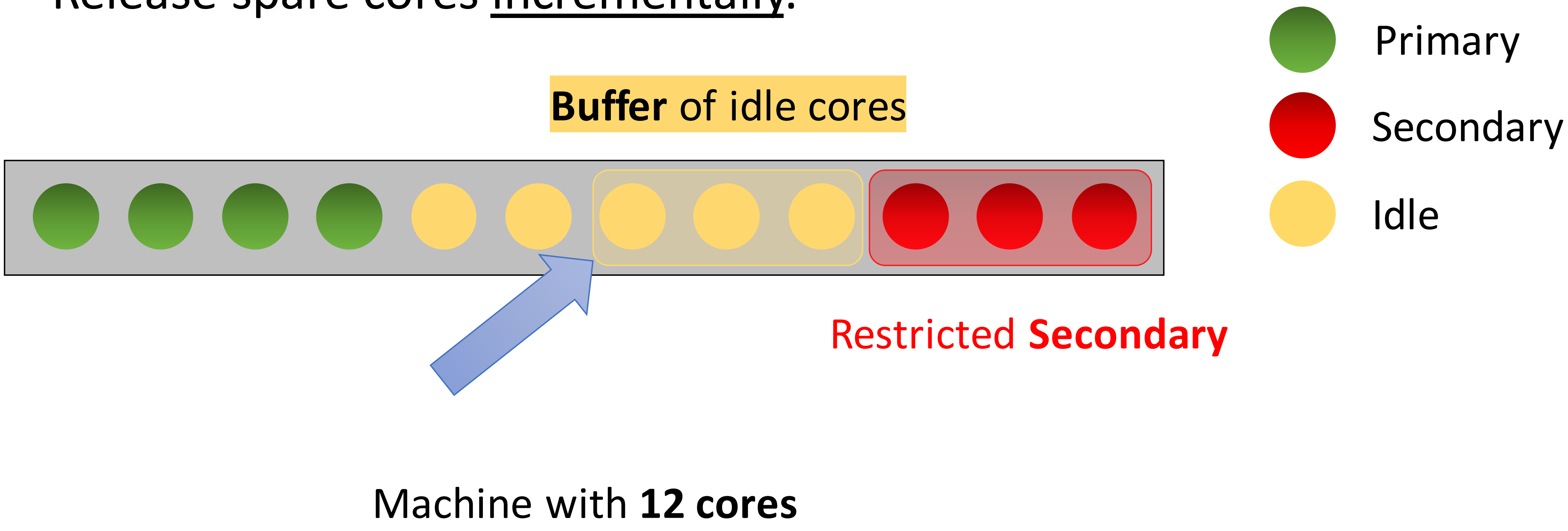
- Allow **Secondary** to use spare idle cores.
- Release spare cores <u>incrementally</u>.



Machine with **12 cores**

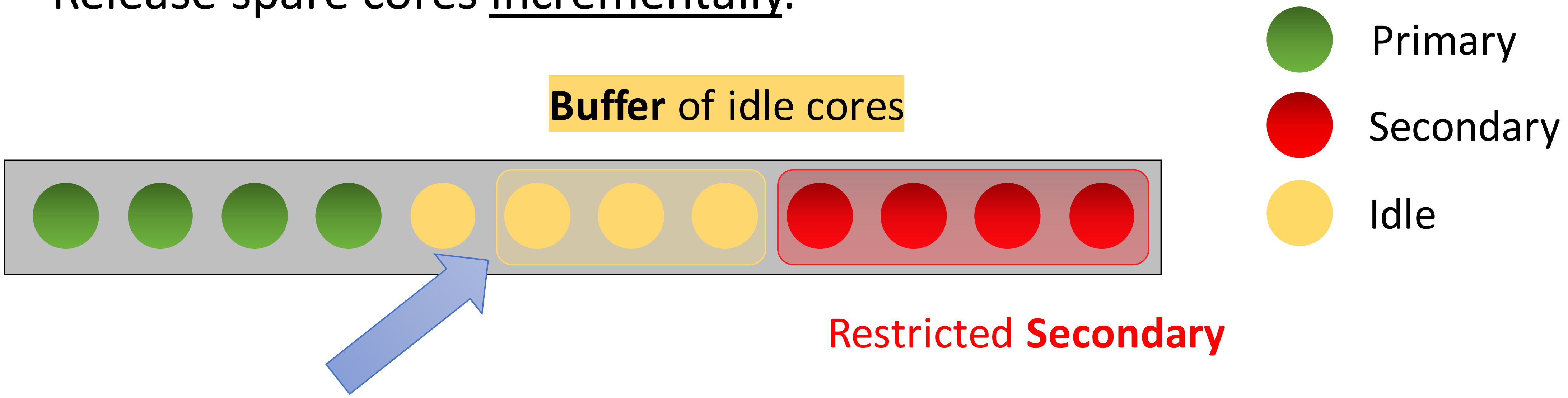
CPU Blind Isolation: Secondary gets spare cores

- Allow **Secondary** to use spare idle cores.
- Release spare cores <u>incrementally</u>.



CPU Blind Isolation: Secondary gets spare cores

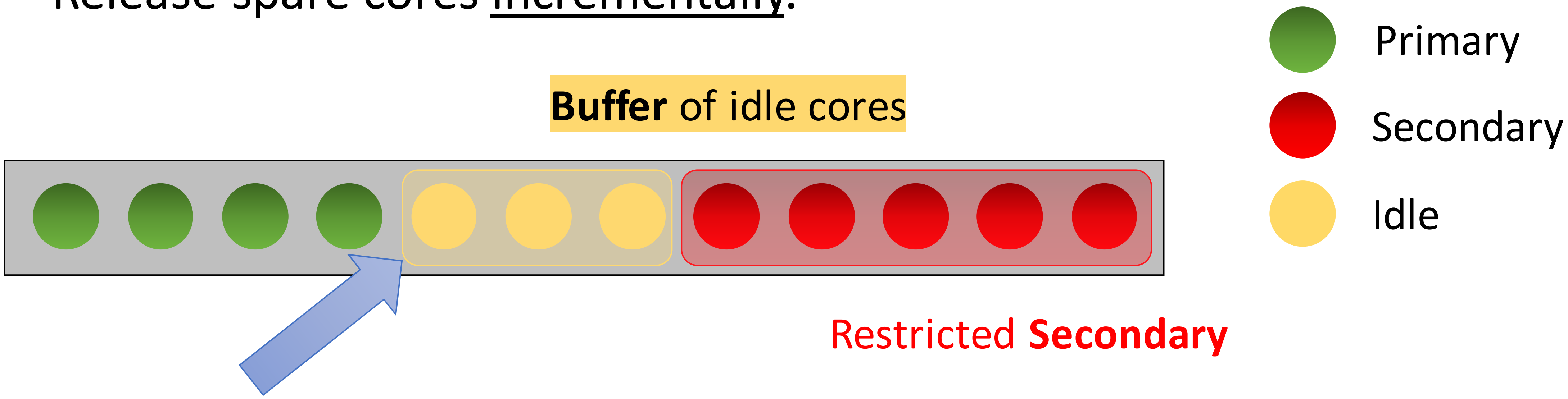
- Allow **Secondary** to use spare idle cores.
- Release spare cores <u>incrementally</u>.



Machine with **12 cores**

CPU Blind Isolation: Secondary gets spare cores

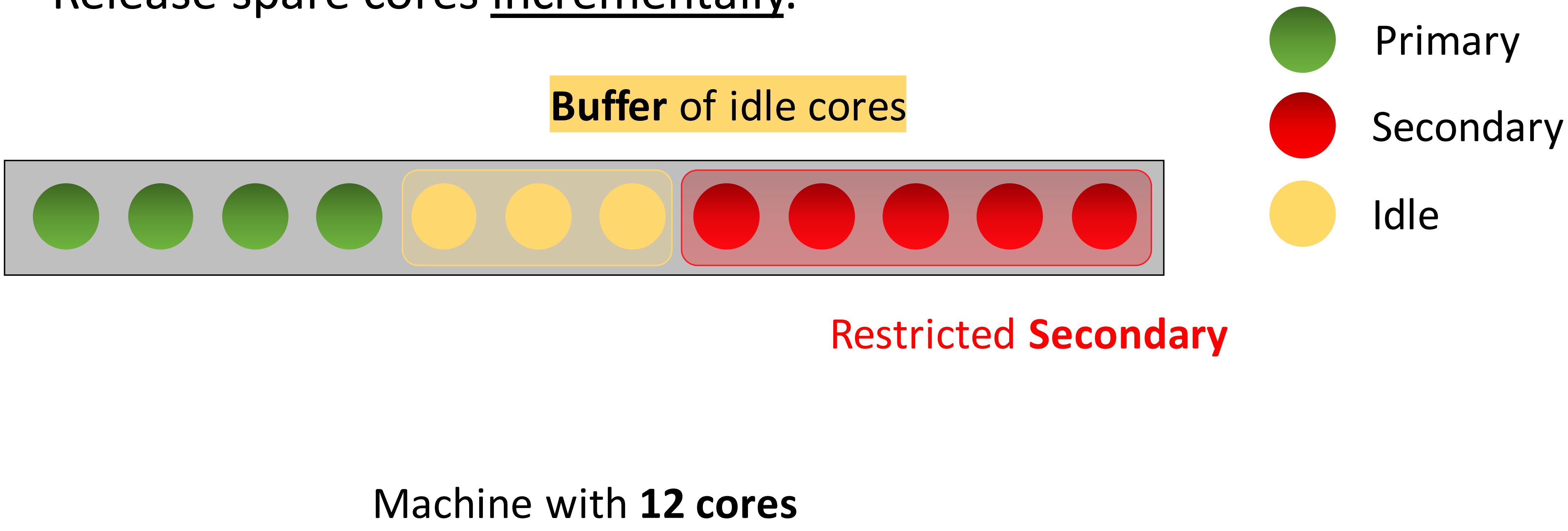
- Allow **Secondary** to use spare idle cores.
- Release spare cores <u>incrementally</u>.



Machine with **12 cores**

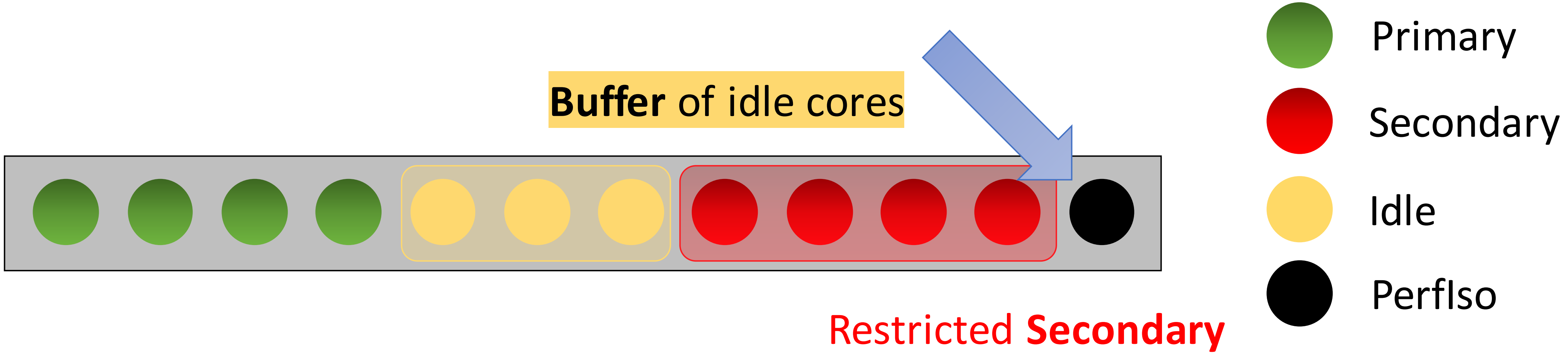
CPU Blind Isolation: Secondary gets spare cores

- Allow **Secondary** to use spare idle cores.
- Release spare cores <u>incrementally</u>.



CPU Blind Isolation: We dedicate 1 core to *PerfIso*

- ***PerfIso*** does continuous polling → we affinitize it to 1 core.



Machine with **12 cores**

Evaluation

Experiment testbed

Hardware

- Intel Xeon E5 – 24 cores (48 w/ HT)
- 128GB RAM

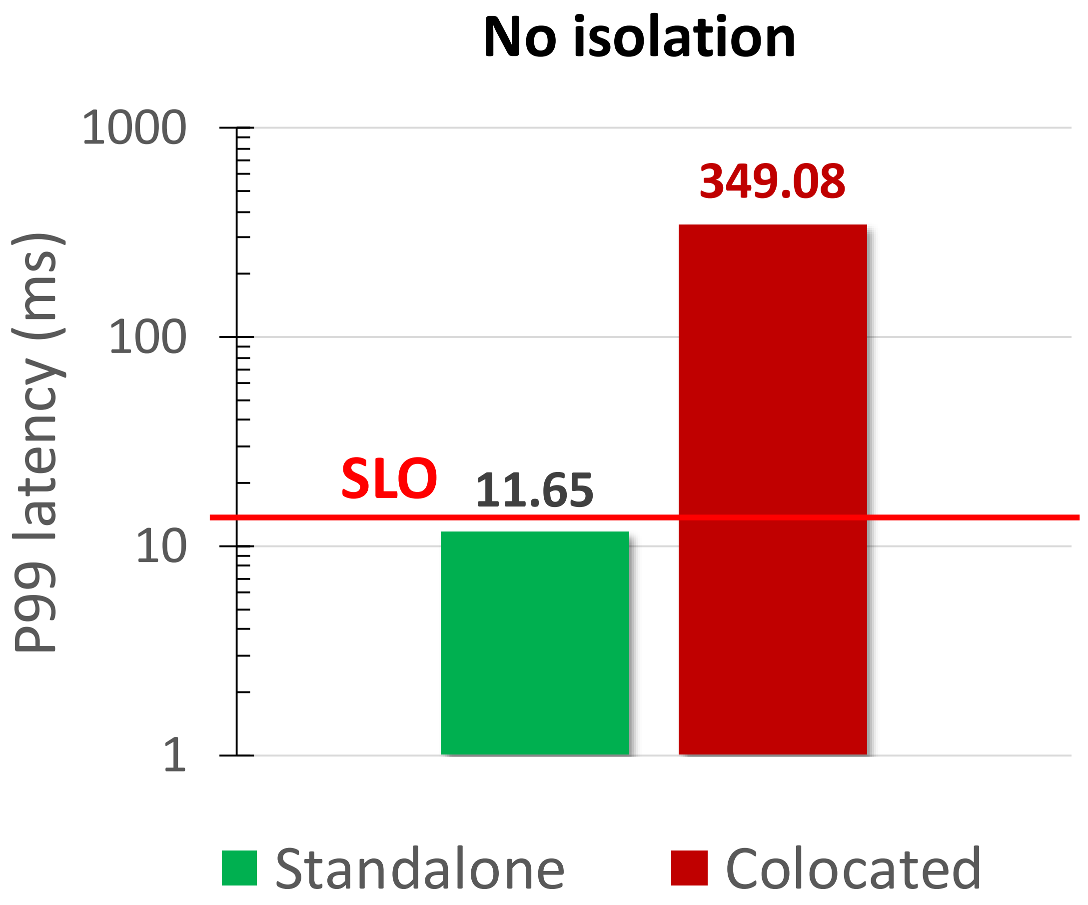
Primary: *Bing IndexServe*

- 569 GB index-slice
- Open-loop client
- 500,000 queries @ 2,000 Q / s

Secondary: CPU micro-benchmark

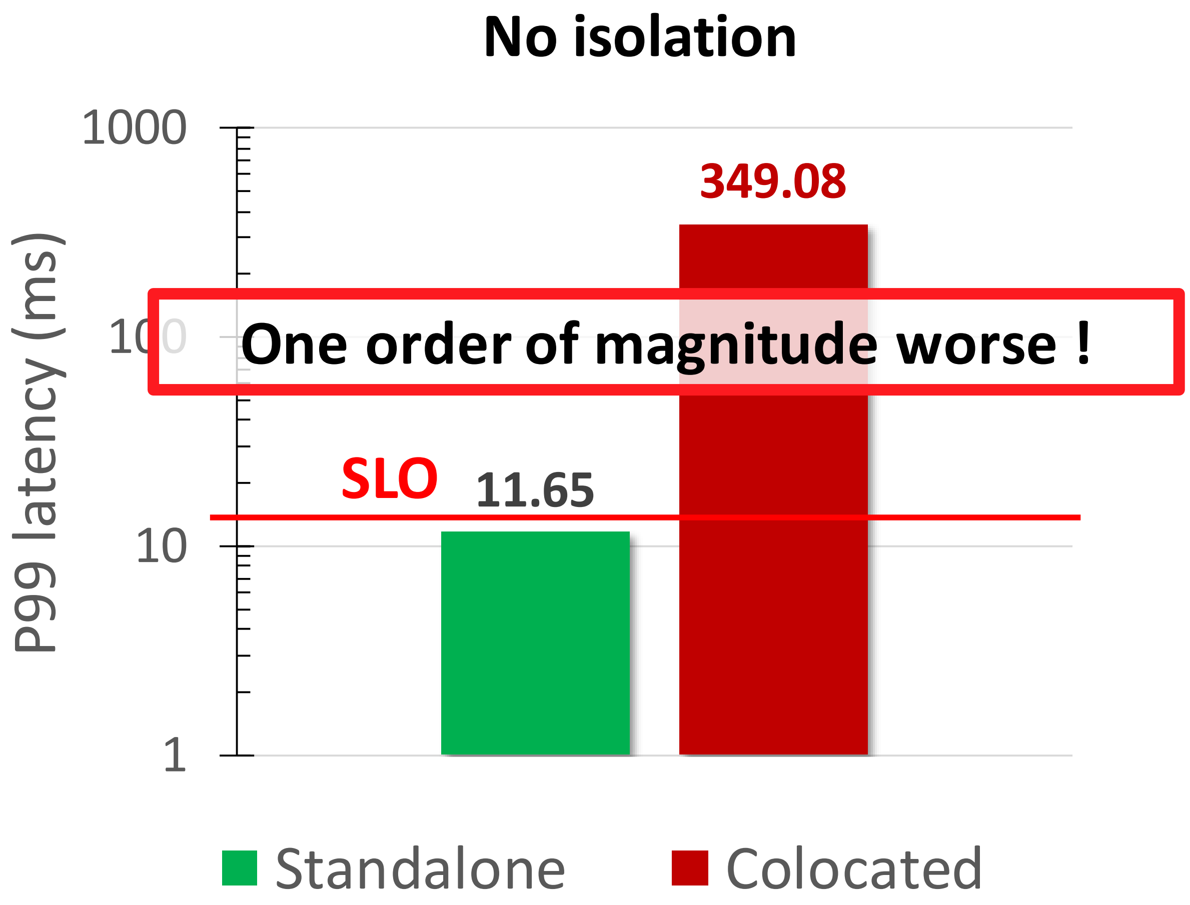
Single server: *PerfIso* protects tail-latency

Secondary: CPU-intensive micro-benchmark



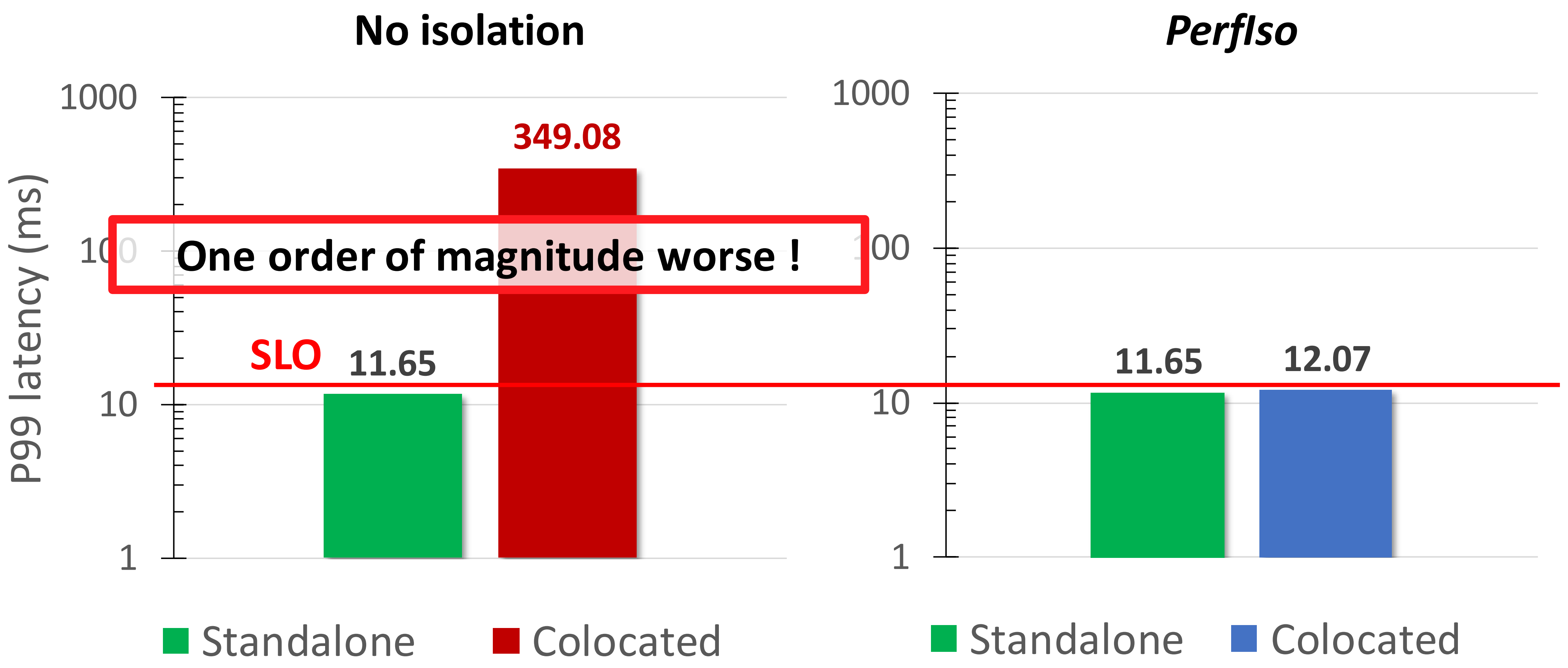
Single server: *PerfIso* protects tail-latency

Secondary: CPU-intensive micro-benchmark



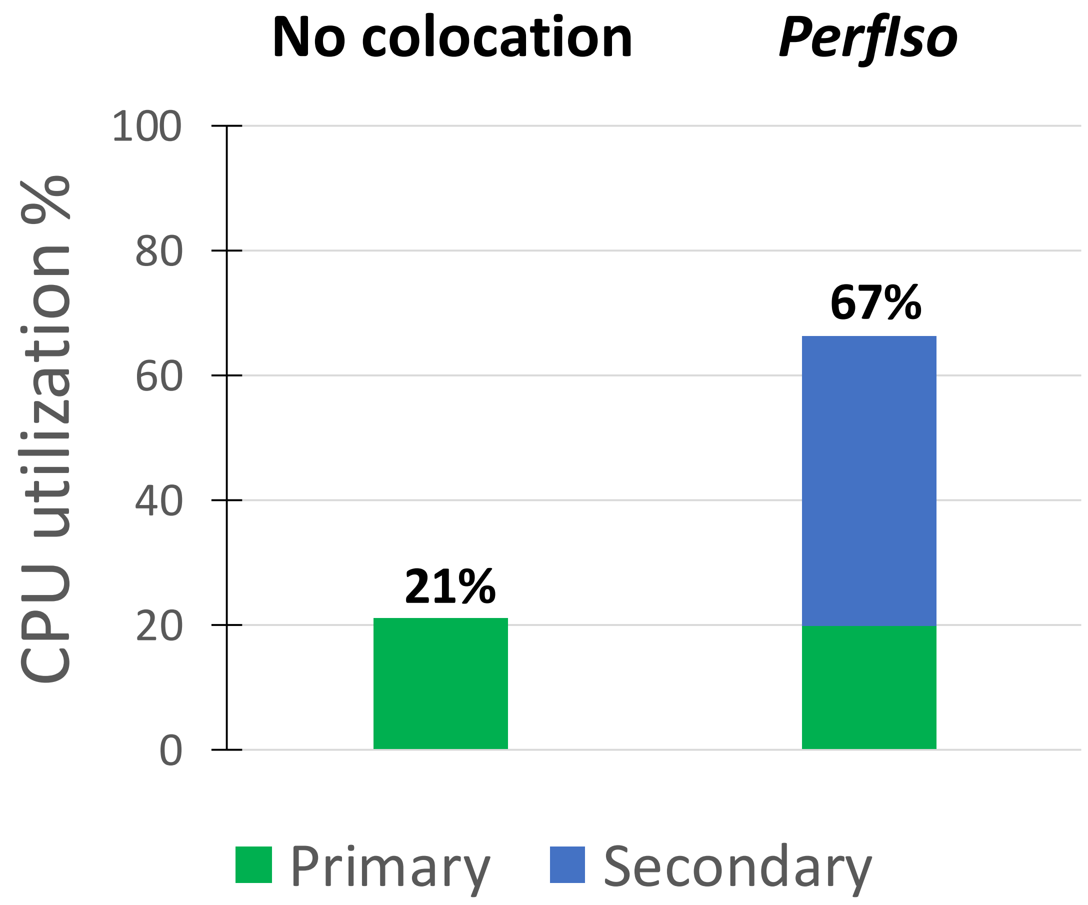
Single server: *PerfIso* protects tail-latency

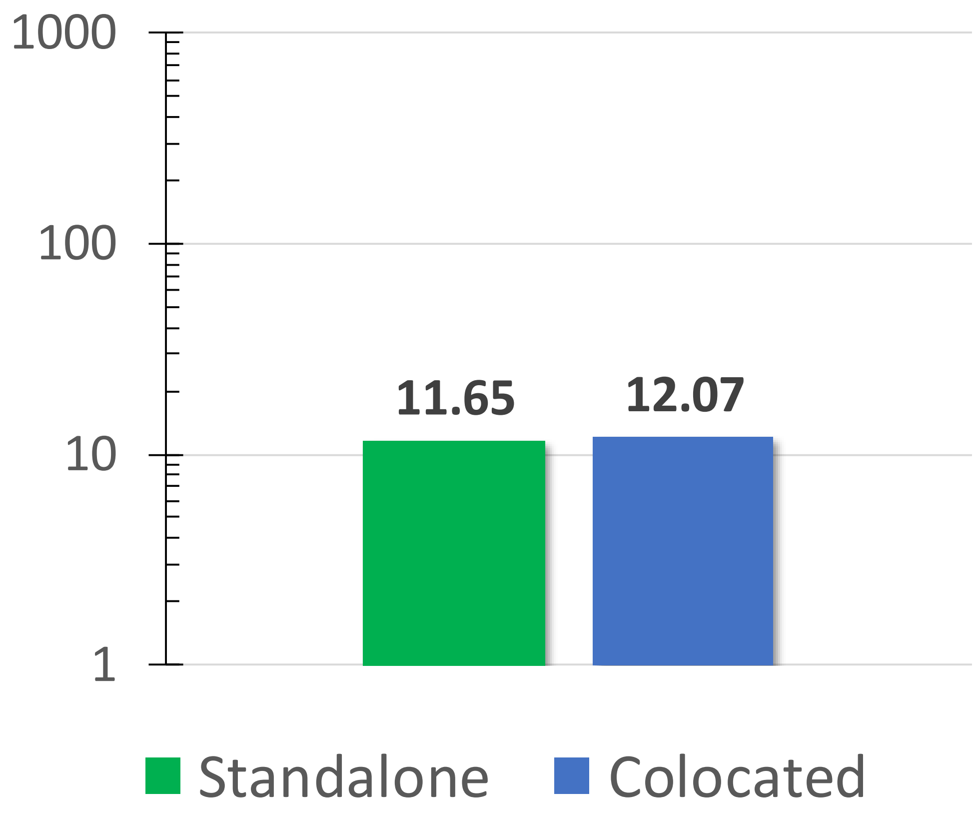
Secondary: CPU-intensive micro-benchmark



Single server: CPU utilization 3x higher!

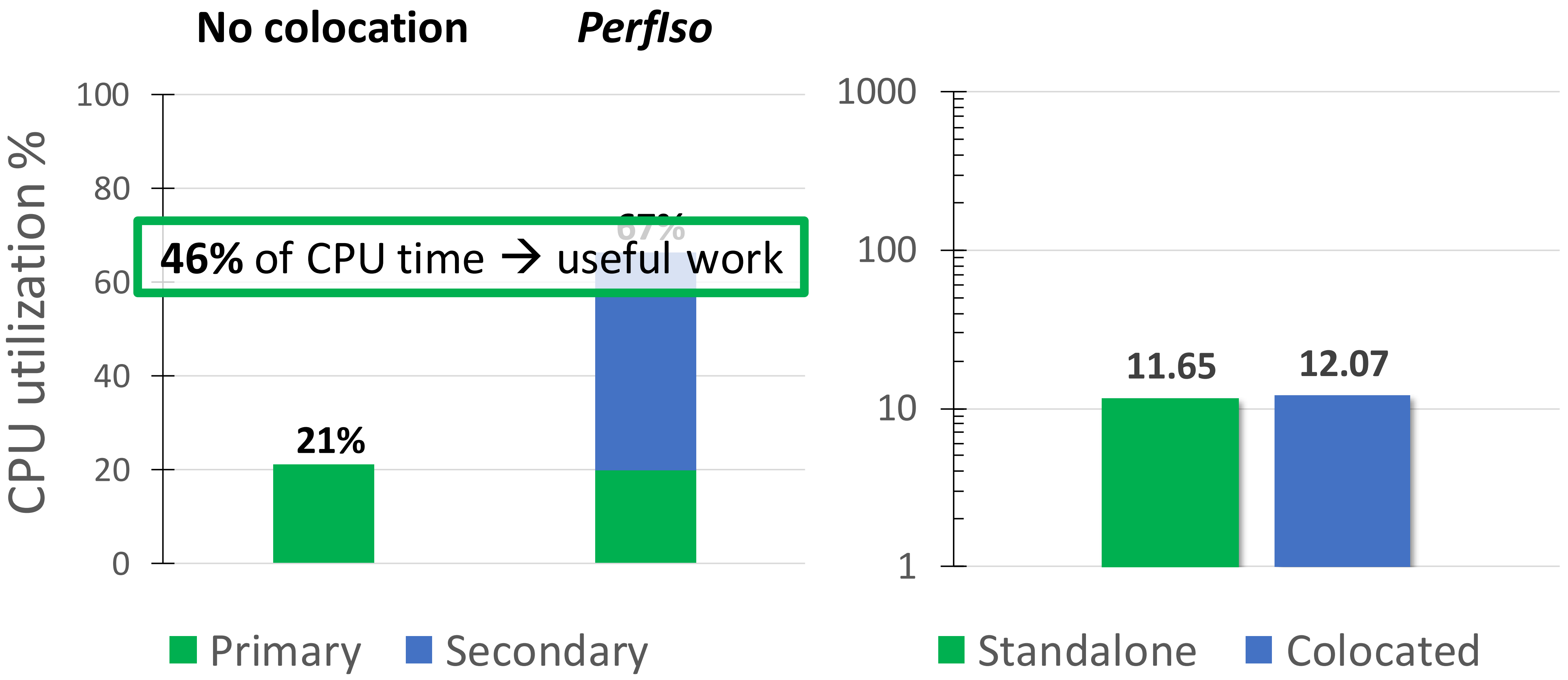
Secondary: CPU-intensive micro-benchmark





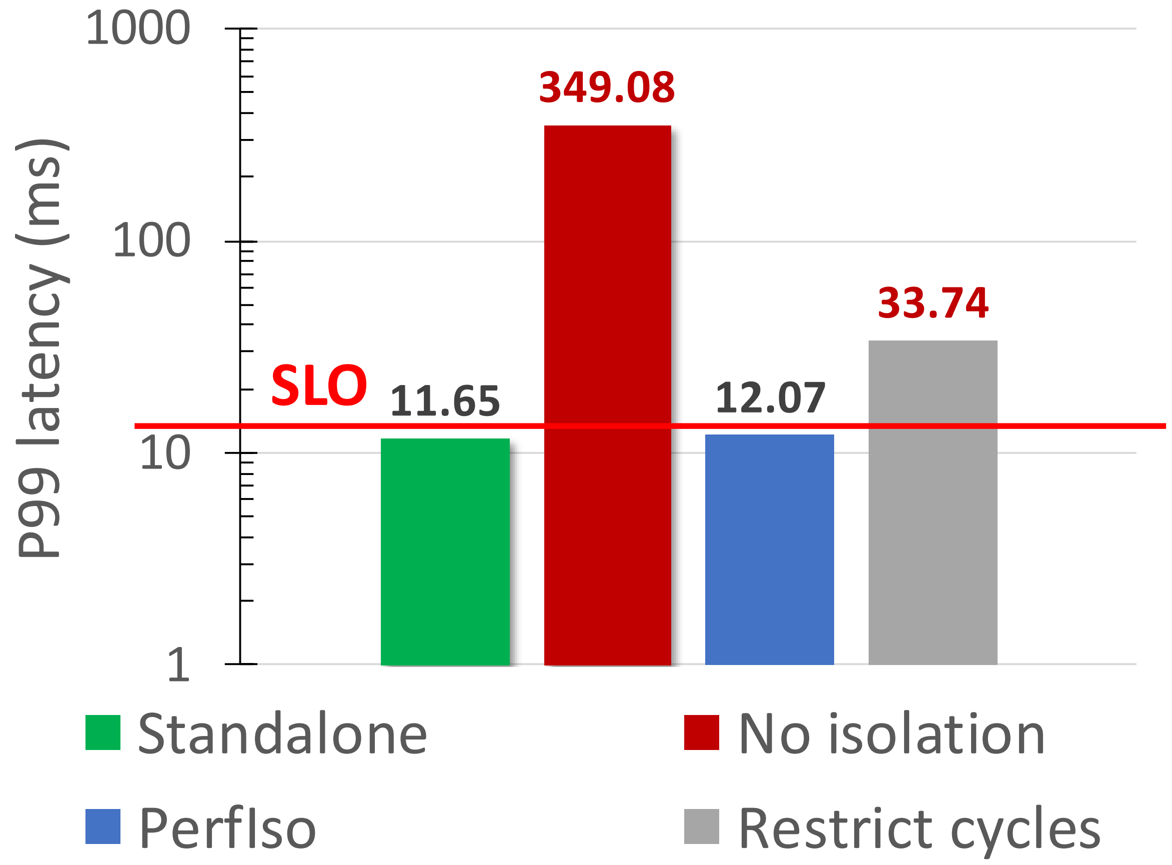
Single server: CPU utilization 3x higher!

Secondary: CPU-intensive micro-benchmark



Restricting CPU cycles does not work

Secondary: CPU-intensive micro-benchmark

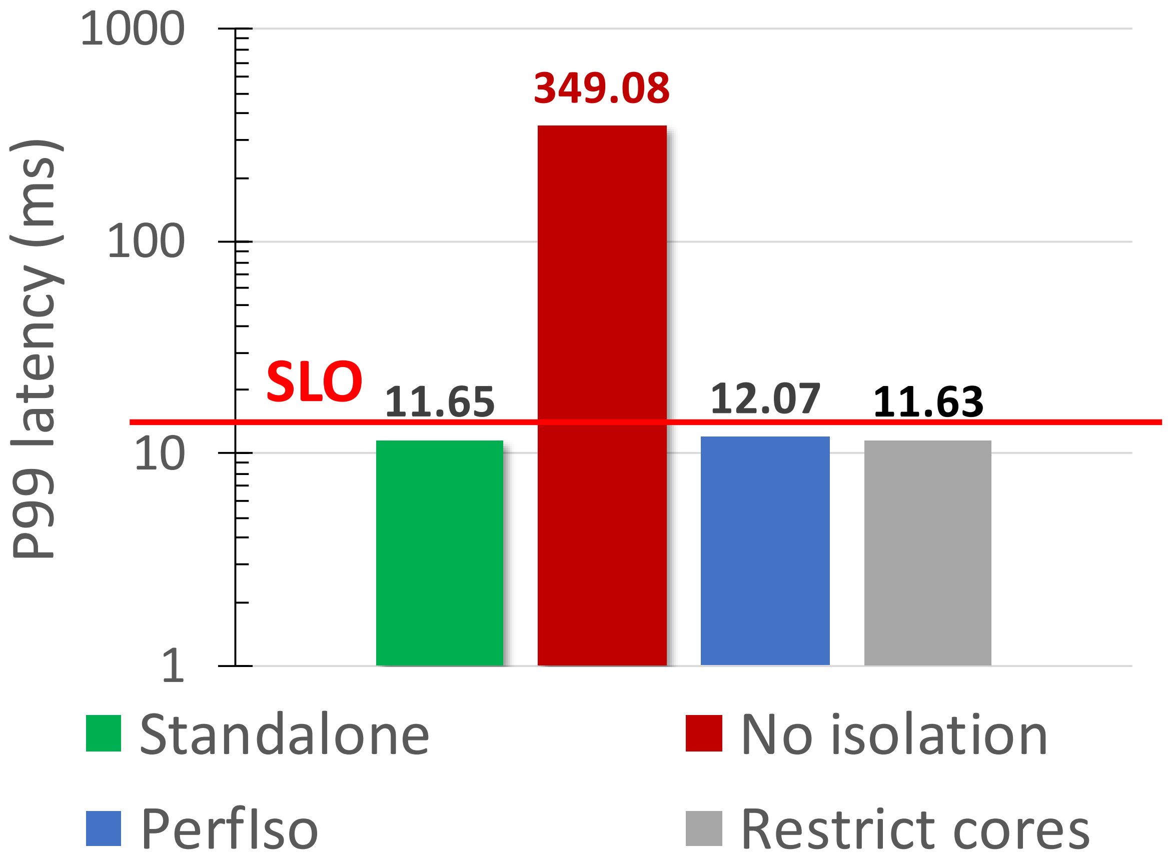


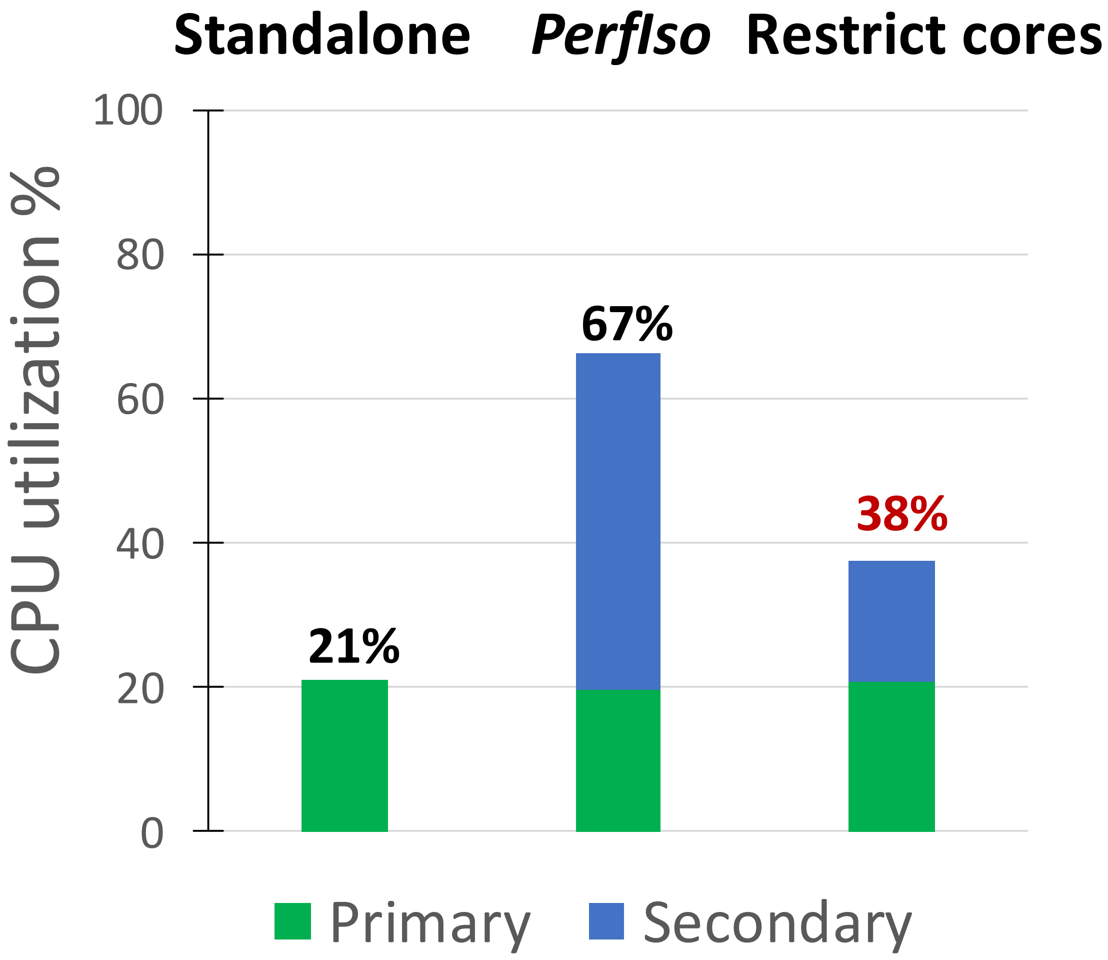
Secondary → 5% of CPU cycles

P99 latency – **3x higher than SLO!**

Restricting CPU cores does not work

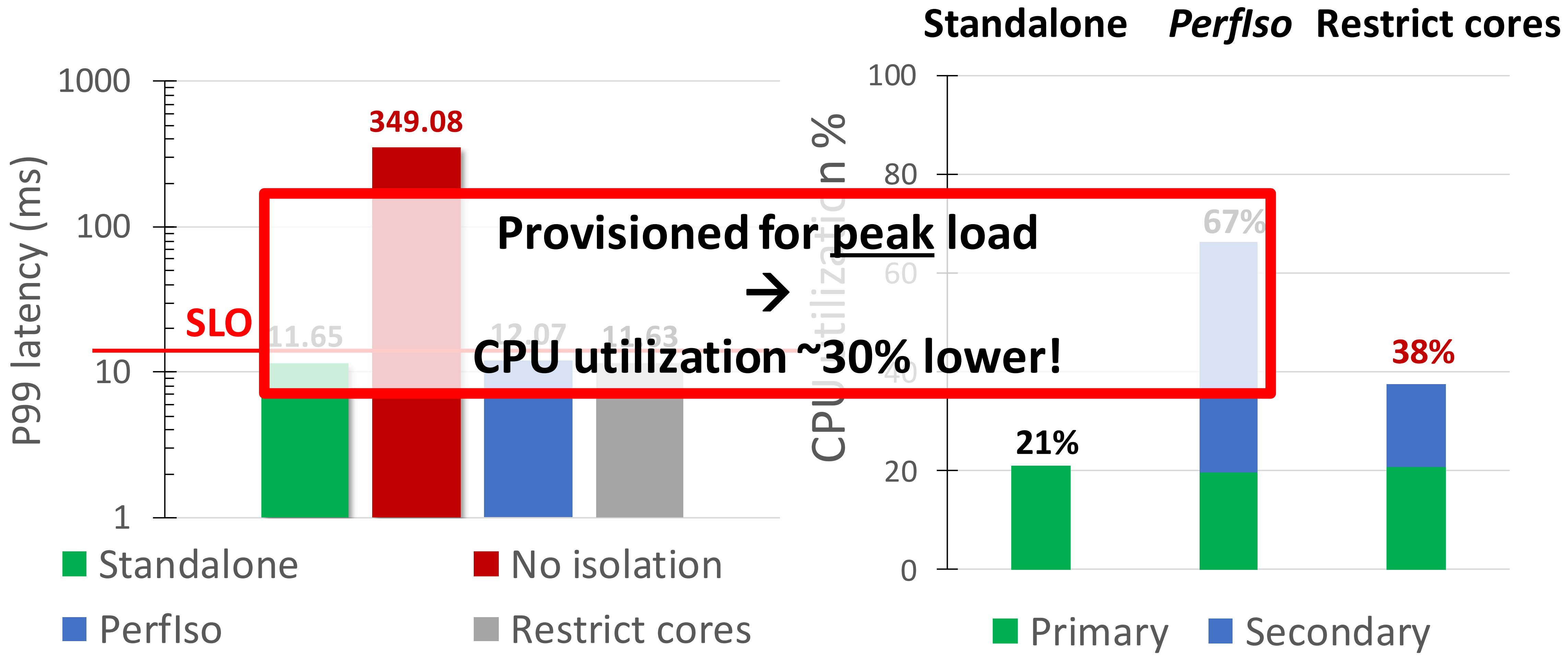
Secondary: CPU-intensive micro-benchmark





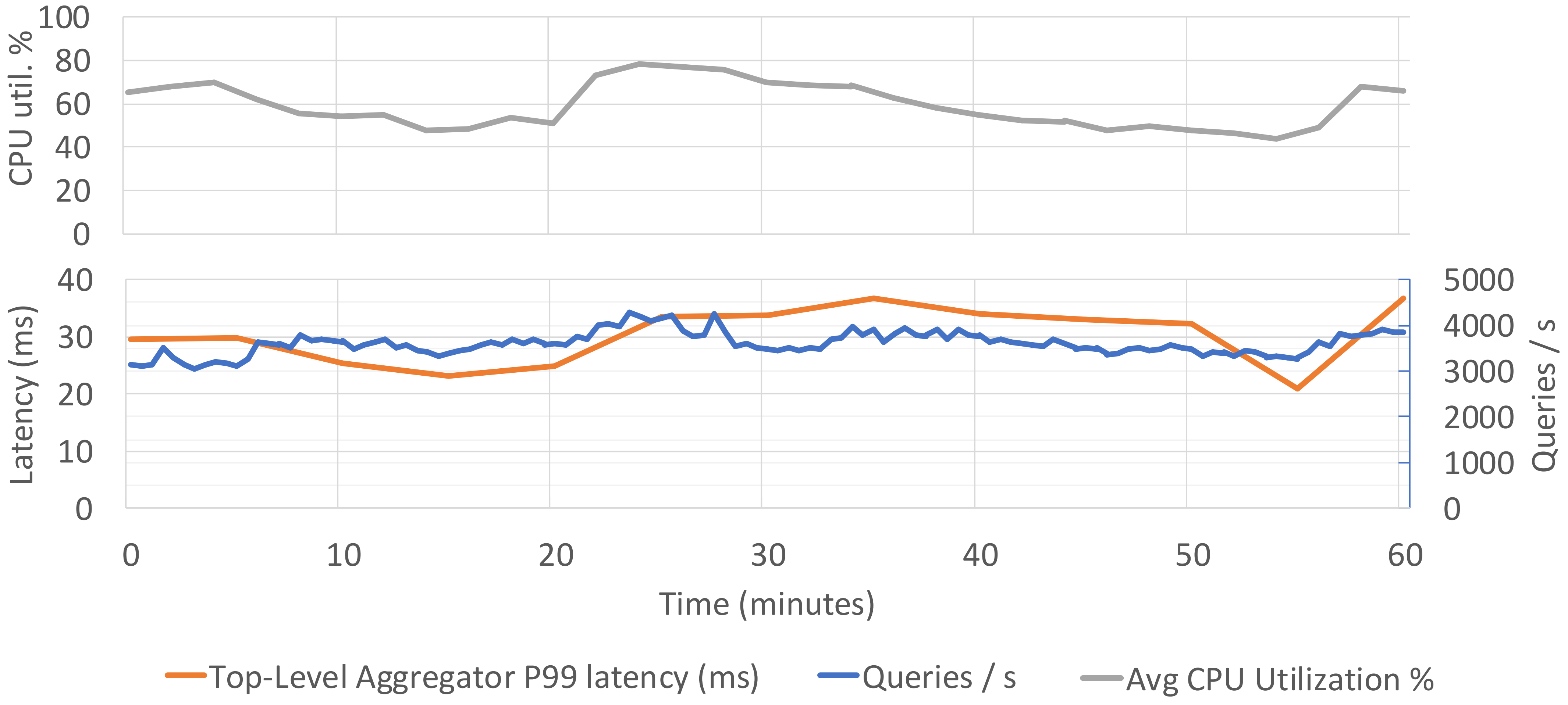
Restricting CPU cores does not work

Secondary: CPU-intensive micro-benchmark



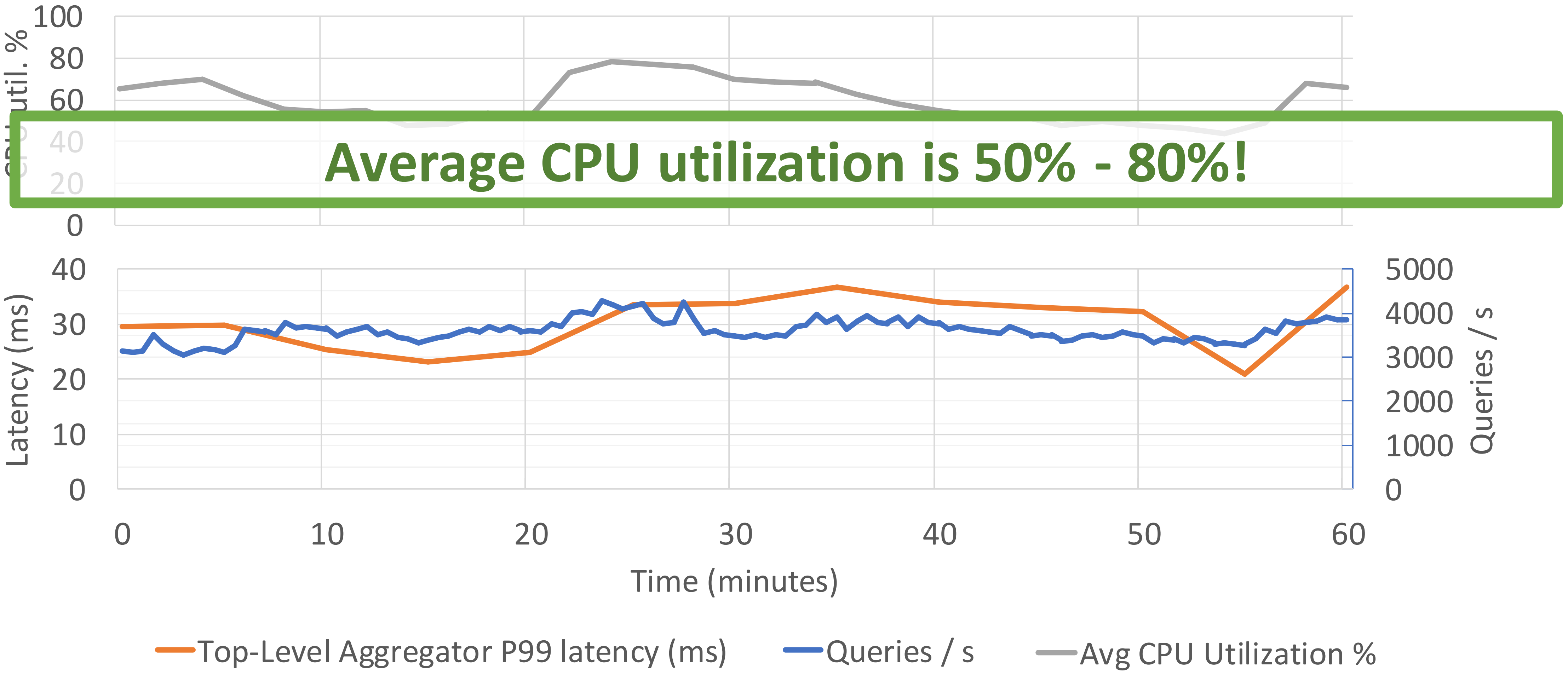
1-hour run of 650 machine cluster

Secondary: Machine-Learning computation



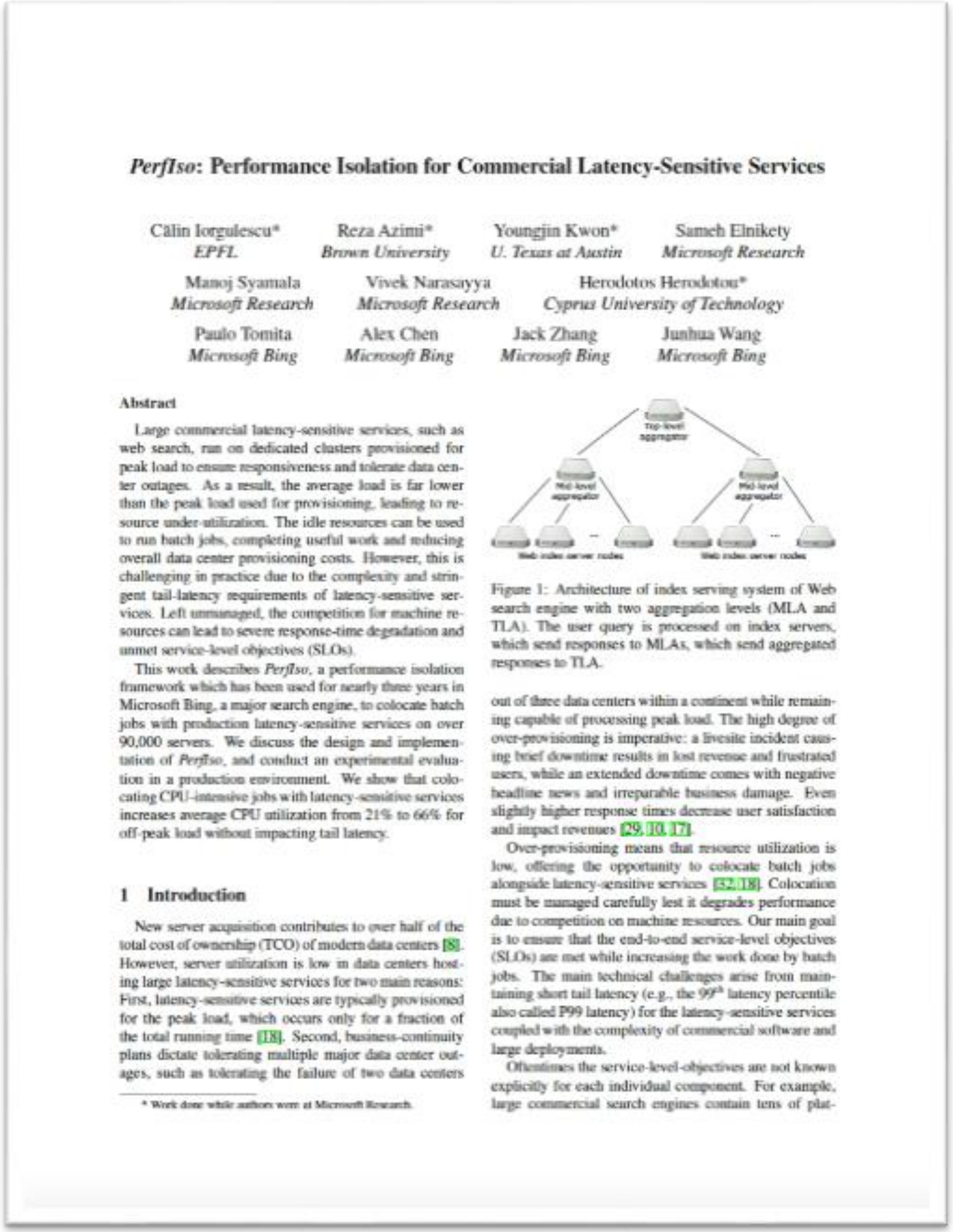
1-hour run of 650 machine cluster

Secondary: Machine-Learning computation



Interesting details in the paper

- Effectiveness of static CPU isolation methods
  - Restricting CPU cycles
  - Restricting CPU cores
- Comparison of state-of-the-art techniques
- Managing disk, memory, and network



PerfIso: Performance Isolation for Commercial Latency-Sensitive Services

Călin Iorgulescu* (EPFL), Reza Azimi* (Brown University), Youngjin Kwon* (U. Texas at Austin), Sameh Elnikety (Microsoft Research), Manoj Syamala (Microsoft Research), Vivek Narasayya (Microsoft Research), Herodotos Herodotou* (Cyprus University of Technology), Paulo Tomita (Microsoft Bing), Alex Chen (Microsoft Bing), Jack Zhang (Microsoft Bing), Junhua Wang (Microsoft Bing)

Abstract

Large commercial latency-sensitive services, such as web search, run on dedicated clusters provisioned for peak load to ensure responsiveness and tolerate data center outages. As a result, the average load is far lower than the peak load used for provisioning, leading to resource under-utilization. The idle resources can be used to run batch jobs, completing useful work and reducing overall data center provisioning costs. However, this is challenging in practice due to the complexity and stringent tail-latency requirements of latency-sensitive services. Left unmanaged, the competition for machine resources can lead to severe response-time degradation and unmet service-level objectives (SLOs).

This work describes *PerfIso*, a performance isolation framework which has been used for nearly three years in Microsoft Bing, a major search engine, to colocate batch jobs with production latency-sensitive services on over 90,000 servers. We discuss the design and implementation of *PerfIso*, and conduct an experimental evaluation in a production environment. We show that colocating CPU-intensive jobs with latency-sensitive services increases average CPU utilization from 21% to 66% for off-peak load without impacting tail latency.

1 Introduction

New server acquisition contributes to over half of the total cost of ownership (TCO) of modern data centers [8]. However, server utilization is low in data centers hosting large latency-sensitive services for two main reasons: First, latency-sensitive services are typically provisioned for the peak load, which occurs only for a fraction of the total running time [18]. Second, business-continuity plans dictate tolerating multiple major data center outages, such as tolerating the failure of two data centers out of three data centers within a continent while remaining capable of processing peak load. The high degree of over-provisioning is imperative: a livesite incident causing brief downtime results in lost revenue and frustrated users, while an extended downtime comes with negative headline news and irreparable business damage. Even slightly higher response times decrease user satisfaction and impact revenues [29, 10, 17].

Over-provisioning means that resource utilization is low, offering the opportunity to colocate batch jobs alongside latency-sensitive services [32, 18]. Colocation must be managed carefully lest it degrades performance due to competition on machine resources. Our main goal is to ensure that the end-to-end service-level objectives (SLOs) are met while increasing the work done by batch jobs. The main technical challenges arise from maintaining short tail latency (e.g., the 99^{th} latency percentile also called P99 latency) for the latency-sensitive services coupled with the complexity of commercial software and large deployments.

Oftentimes the service-level-objectives are not known explicitly for each individual component. For example, large commercial search engines contain tens of plat-

Figure 1: Architecture of index serving system of Web search engine with two aggregation levels (MLA and TLA). The user query is processed on index servers, which send responses to MLAs, which send aggregated responses to TLA.

* Work done while authors were at Microsoft Research.

PerfIso: colocate batch jobs with online services

PerfIso: colocate batch jobs with online services

- **Black-box**: do not tailor to one specific service

PerfIso: colocate batch jobs with online services

- **Black-box**: do not tailor to one specific service
- **Robustness**: favor user-mode over kernel implementation

PerfIso: colocate batch jobs with online services

- **Black-box**: do not tailor to one specific service
- **Robustness**: favor user-mode over kernel implementation
- **Headroom**: some core-slack makes Primary behave like standalone

PerfIso: colocate batch jobs with online services

- **Black-box**: do not tailor to one specific service
- **Robustness**: favor user-mode over kernel implementation
- **Headroom**: some core-slack makes Primary behave like standalone

- <u>CPU Blind Isolation → colocation without impacting service performance</u>