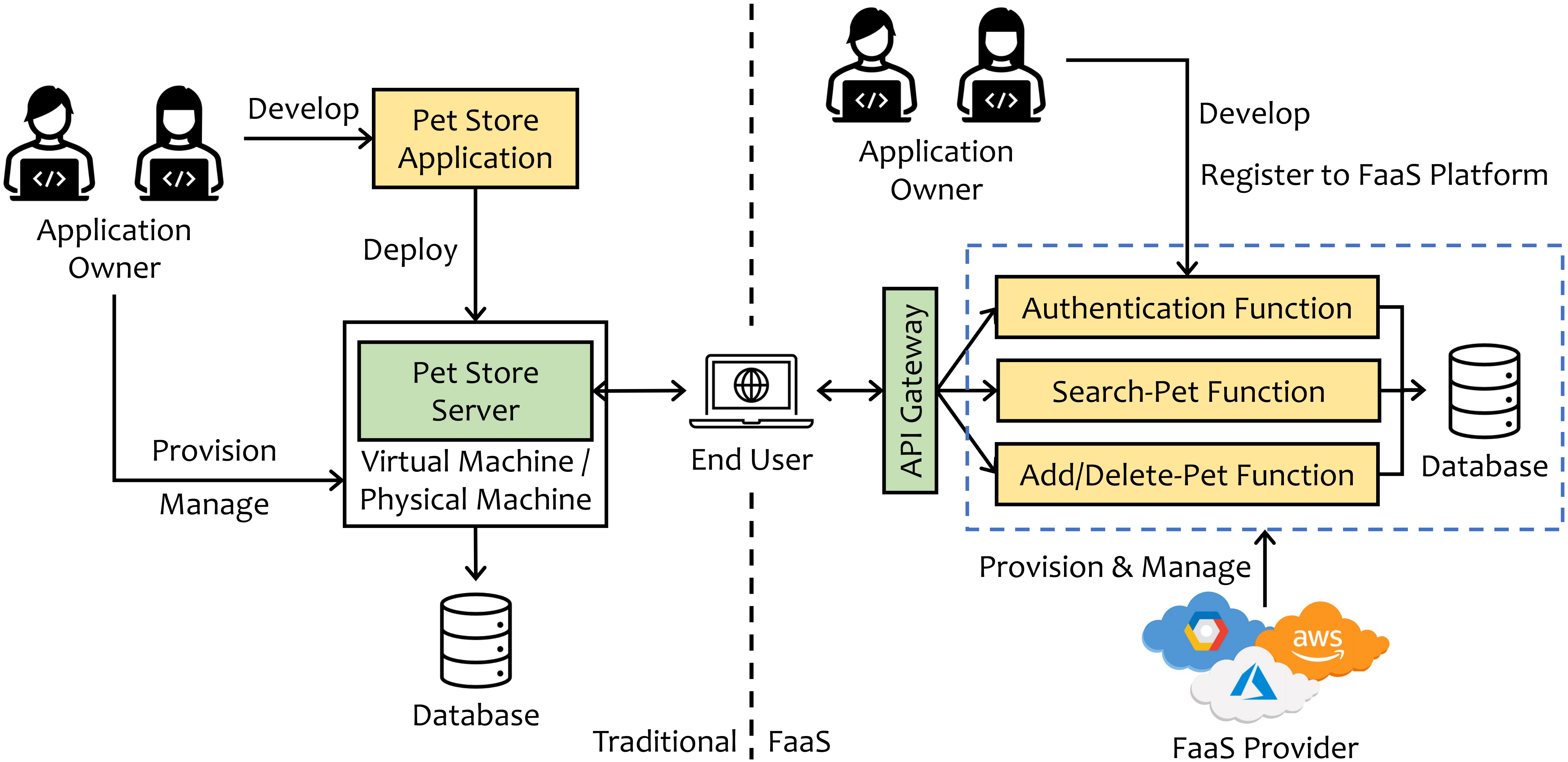
Is Function-as-a-Service (FaaS) a Good Fit for Latency-critical Services

Haoran Qiu[1], Saurabh Jha[1], Subho Banerjee[1], Archit Patke[1], Chen Wang[2], Hubertus Franke[2]
Zbigniew Kalbarczyk[1], Ravishakar Iyer[1]

[1] University of Illinois, Urbana-Champaign [2] IBM Research

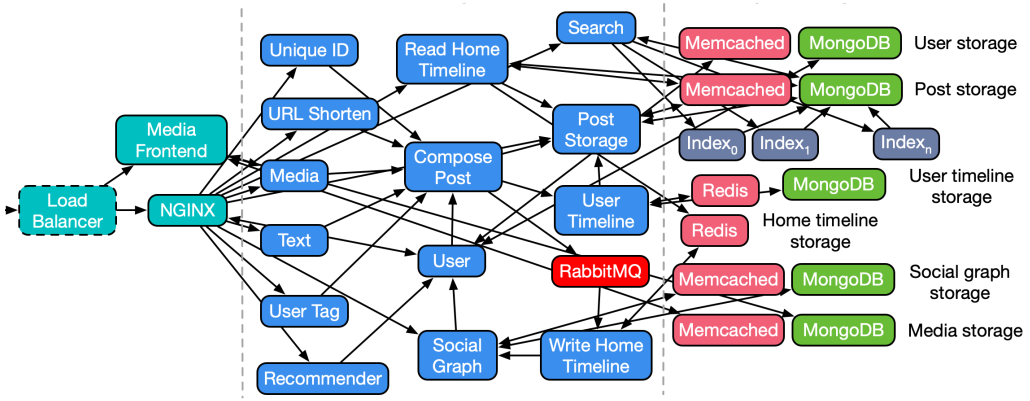
Paper link: https://www.serverlesscomputing.org/wosc7/papers/p1
Seventh International Workshop on Serverless Computing (WoSC7) 2021

Traditional vs. FaaS – An Example



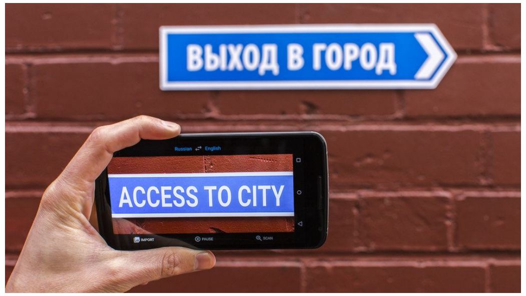
Latency-critical Services

Online Navigation



Social Network

Web Mail Service



Machine Translation

Latency-critical Services

- Latency-critical services are typically user-facing and operate with strict service-level objectives (SLOs) on the end-to-end latency, especially the tail latency (e.g., 99th percentile of the requests returned to users < 100ms).
- Question: Is FaaS a good fit for latency-critical services?

Lower latency!

Customers

Higher utilization, Higher profit!

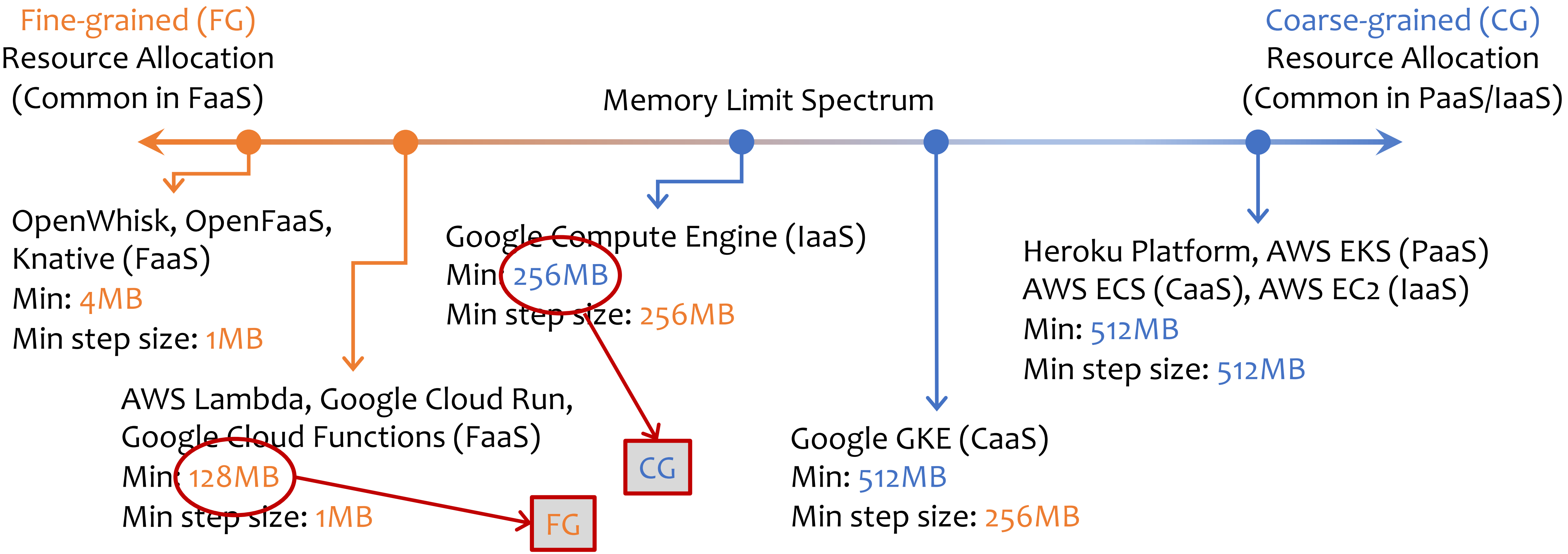
FaaS Provider

Web Mail Service

Machine Translation

Resource Granularity in Workload Consolidation Policies

- We tune the memory limit of each container as FaaS platform allocates other type of resources proportionally to memory limits
- Resource granularities are discrete points on a spectrum



Goal and Key Findings

- What is the trade-off among power consumption, CPU utilization, and end-to-end latency in the decision-making of choosing a workload consolidation policy?
  - Increasing resource granularity (e.g., increasing a container's allocated memory limit from 128 MB to 256 MB):
    - Reduces tail latency by up to 2x,
    - Consumes up to 1.75× more power,
    - Reduces CPU utilization by up to 59%

This Talk

- How is the performance variation affected by fine-grained workload consolidation?
- How do different workload consolidation policies affect the breakdown percentages of different phases in the end-to-end latency?

Goal and Key Findings

- What is the trade-off among power consumption, CPU utilization, and end-to-end latency in the decision-making of choosing a workload consolidation policy?
  - Increasing resource granularity (e.g., increasing a container's allocated memory limit from 128 MB to 256 MB):
    - Reduces tail latency by up to 2x,
    - Consumes up to 1.75× more power,
    - Reduces CPU utilization by up to 59%

This Talk

- How is the performance variation affected by fine-grained workload consolidation?
  - Shared resource contention leads to tail-latency increase of up to 32.6x, 28.9x, and 4.4x for CPU, memory, and LLC sensitive workloads
    - With state-of-the-art resource partitioning, tail-latency increase becomes 8.3x, 21.5x, and 2.3x
- How do different workload consolidation policies affect the breakdown percentages of different phases in the end-to-end latency?

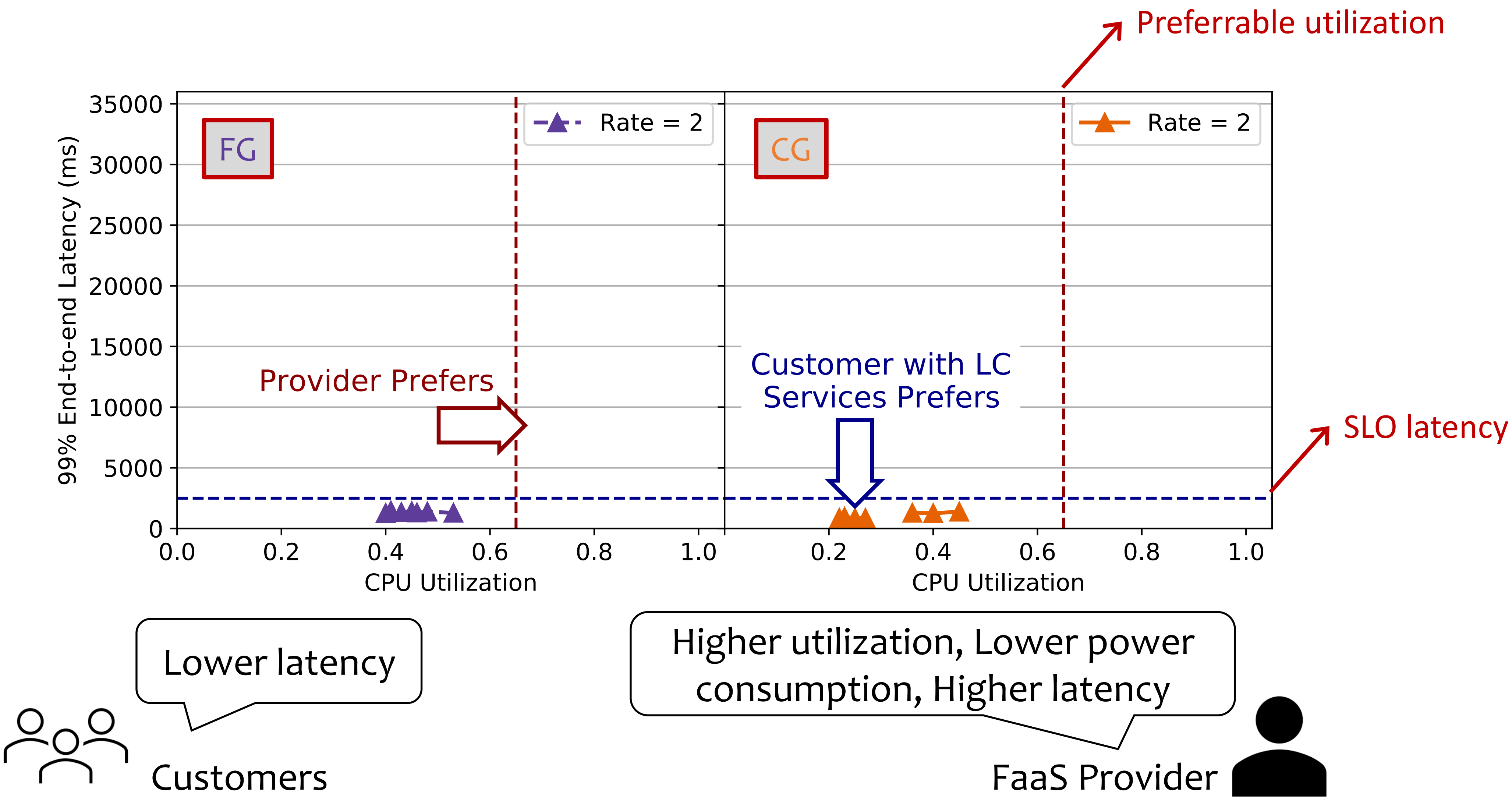
Goal and Key Findings

- What is the trade-off among power consumption, CPU utilization, and end-to-end latency in the decision-making of choosing a workload consolidation policy?
  - Increasing resource granularity (e.g., increasing a container's allocated memory limit from 128 MB to 256 MB):
    - Reduces tail latency by up to 2x,
    - Consumes up to 1.75× more power,
    - Reduces CPU utilization by up to 59%

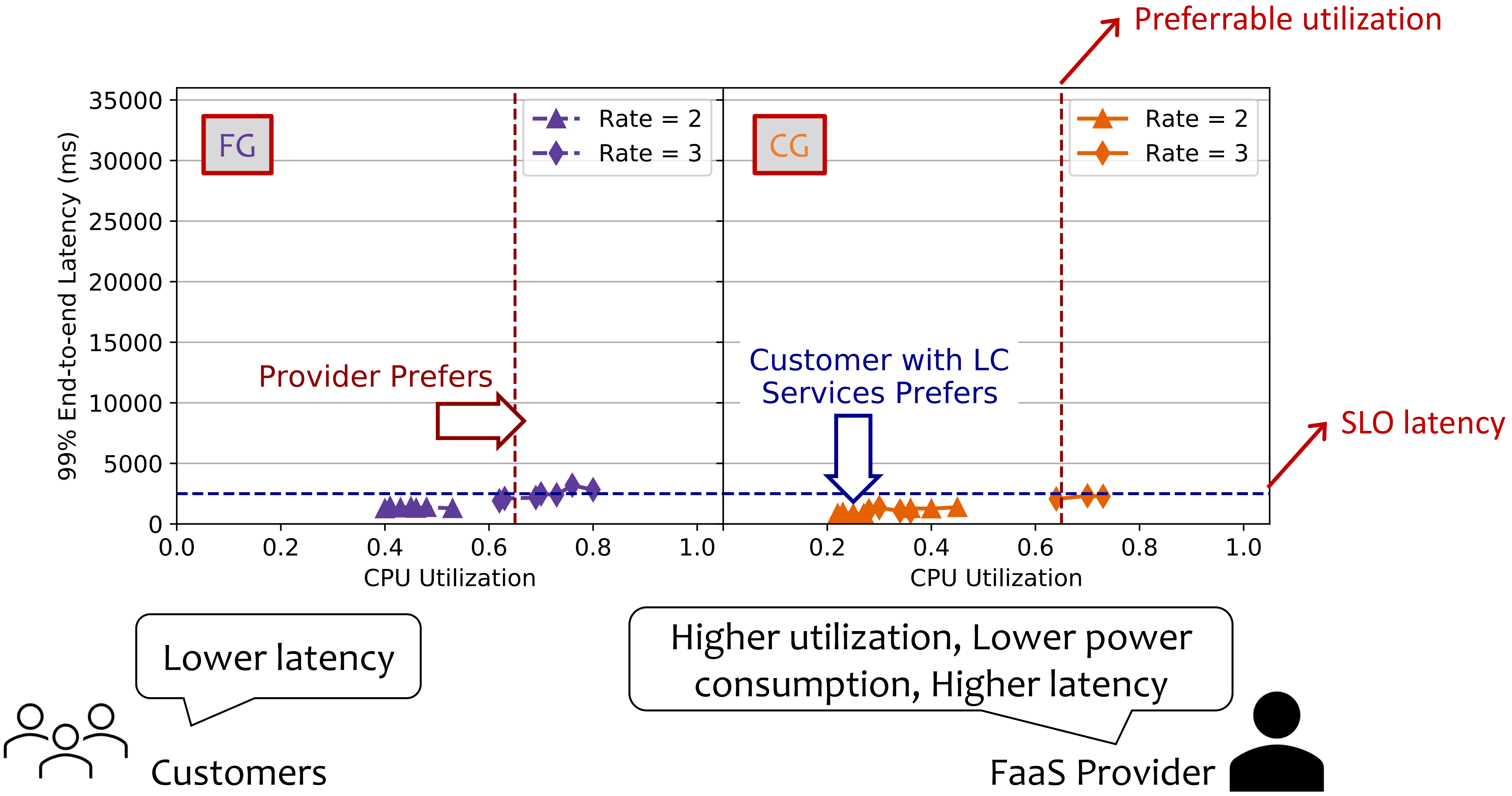
This Talk

- How is the performance variation affected by fine-grained workload consolidation?
- How do different workload consolidation policies affect the breakdown percentages of different phases in the end-to-end latency?
  - Increasing the horizontal concurrency (i.e., number of containers) from 2 to 12 on a single server via decreasing resource granularity:
    - Reduces tail wait time by 49.5x but increases tail init time by 1.3x and increases tail execution time by 15.6x
    - End-to-end latency breakdown varies with concurrency and workloads

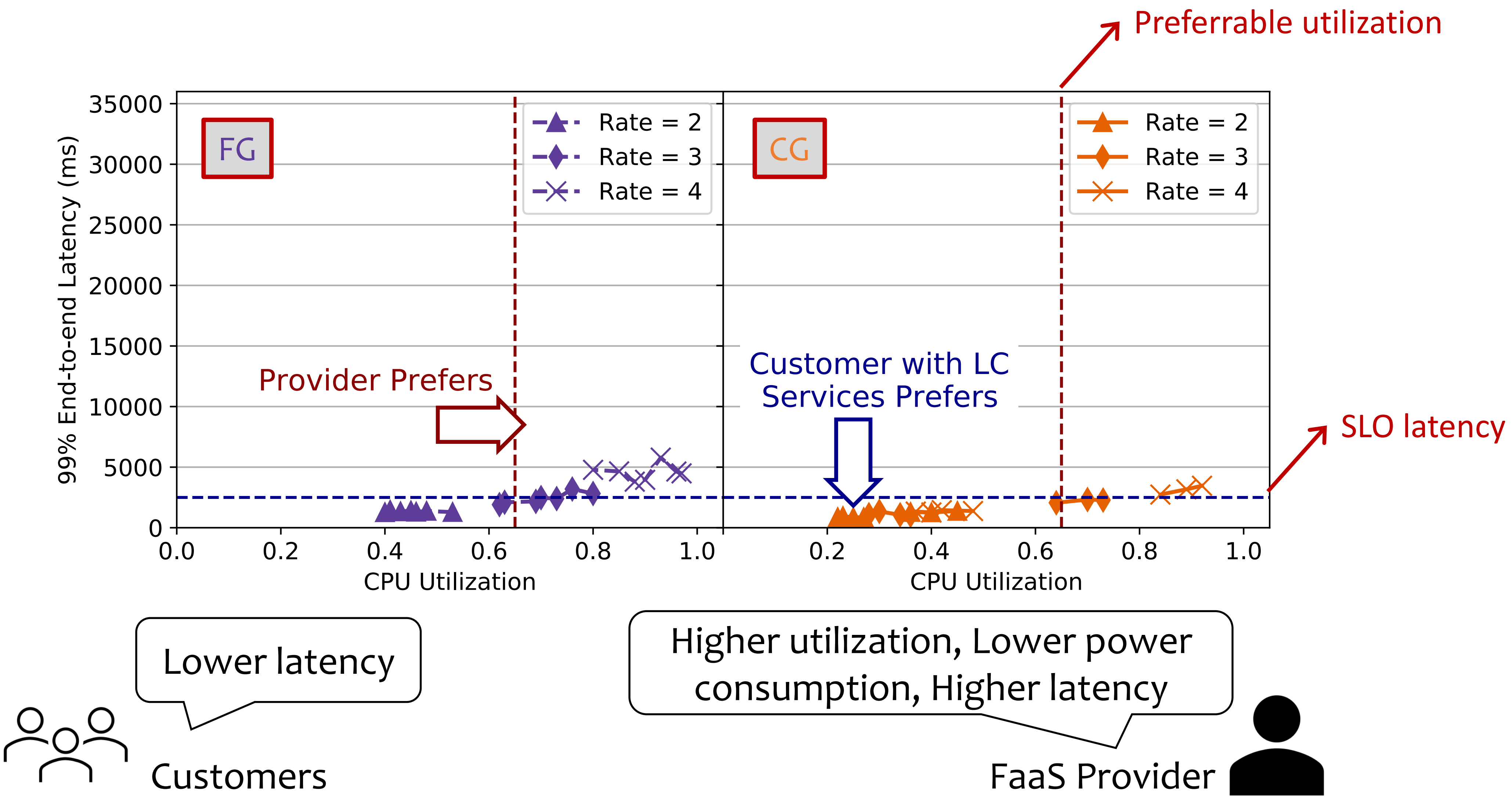
Latency-Utilization-Power Trade-off



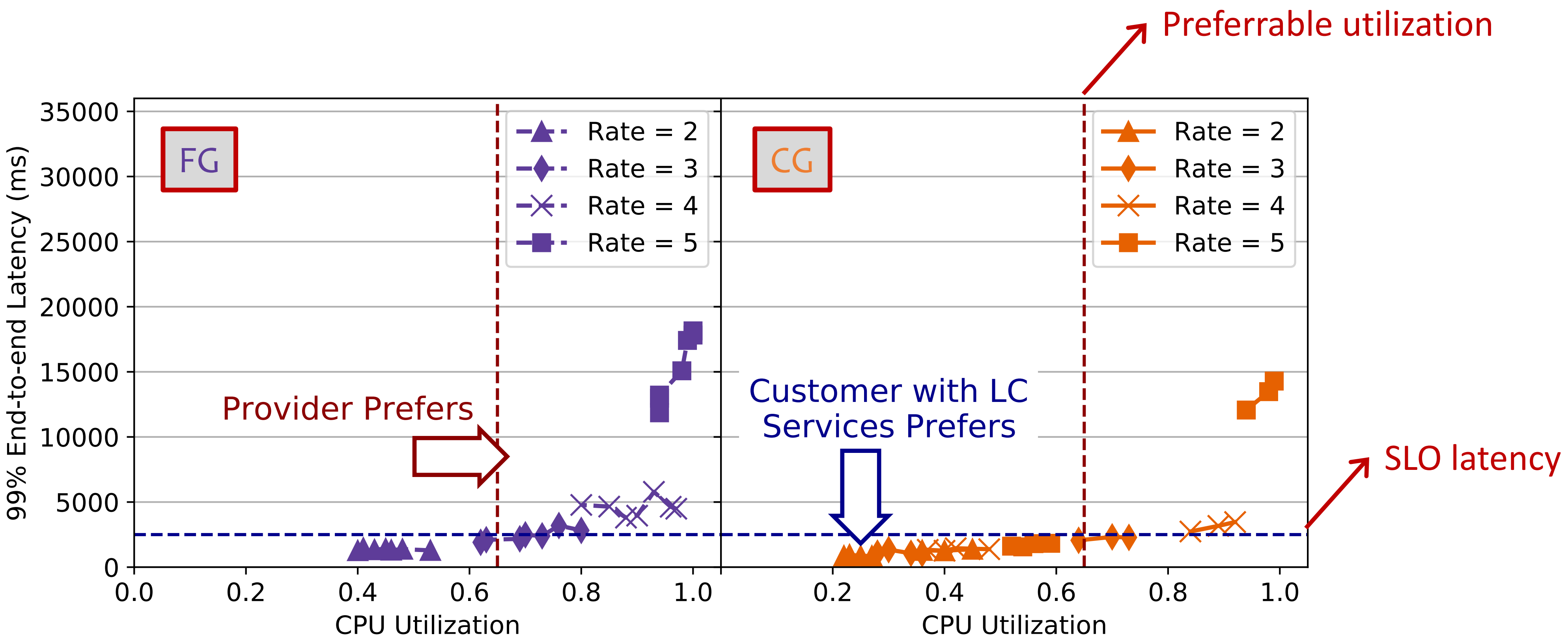
Latency-Utilization-Power Trade-off

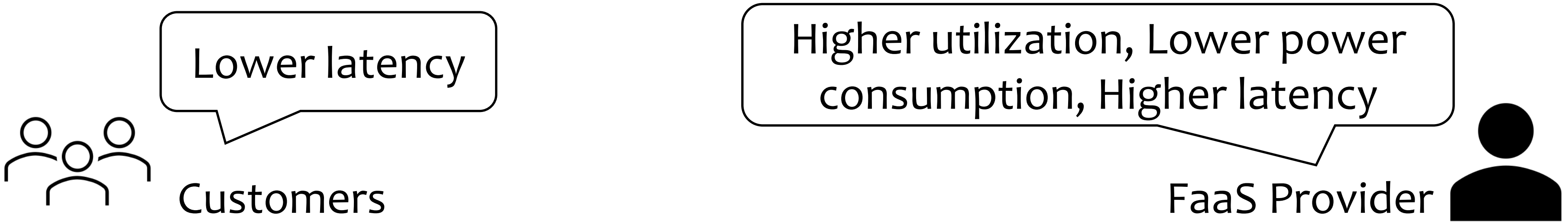


Latency-Utilization-Power Trade-off

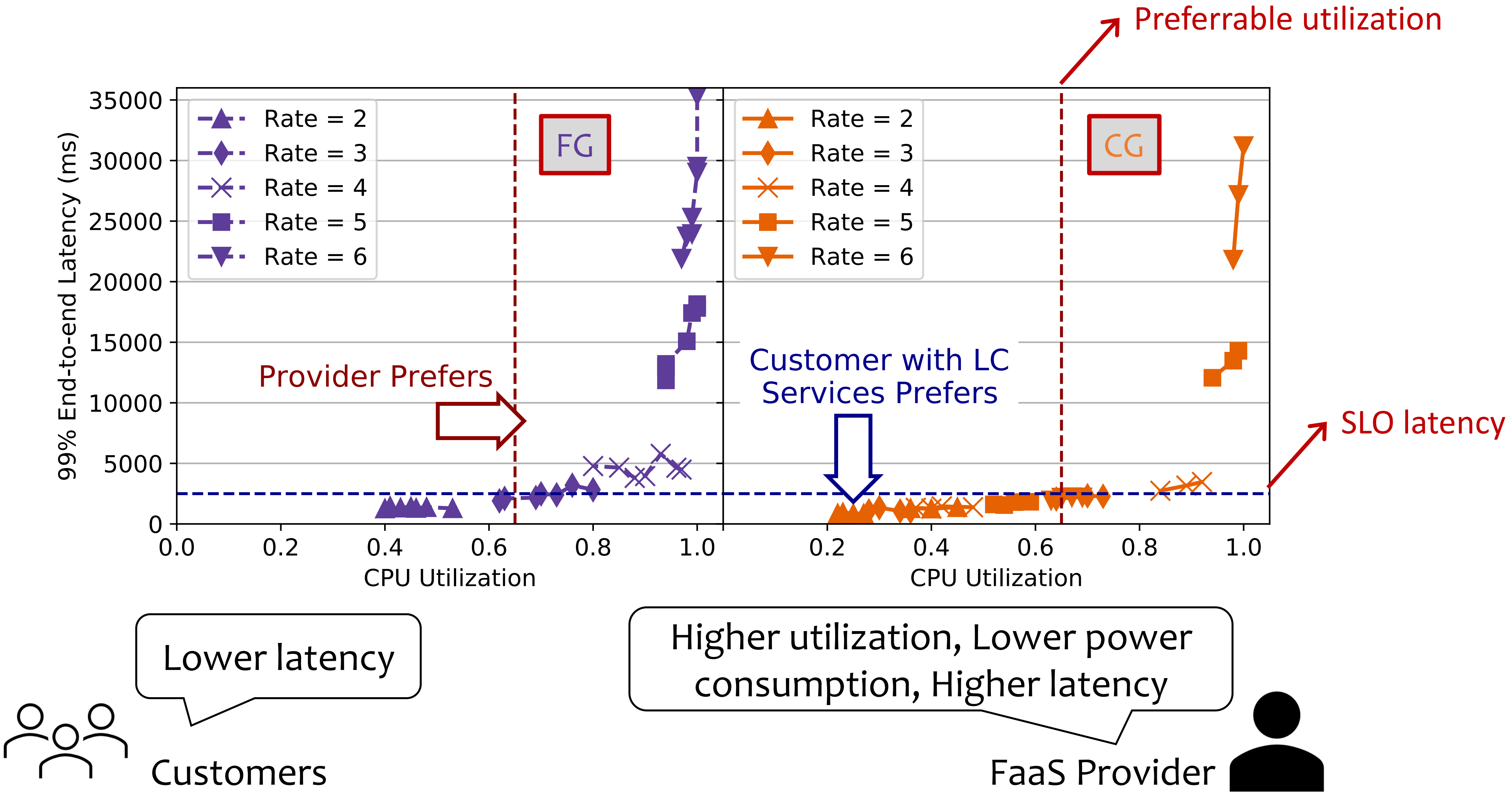


Latency-Utilization-Power Trade-off

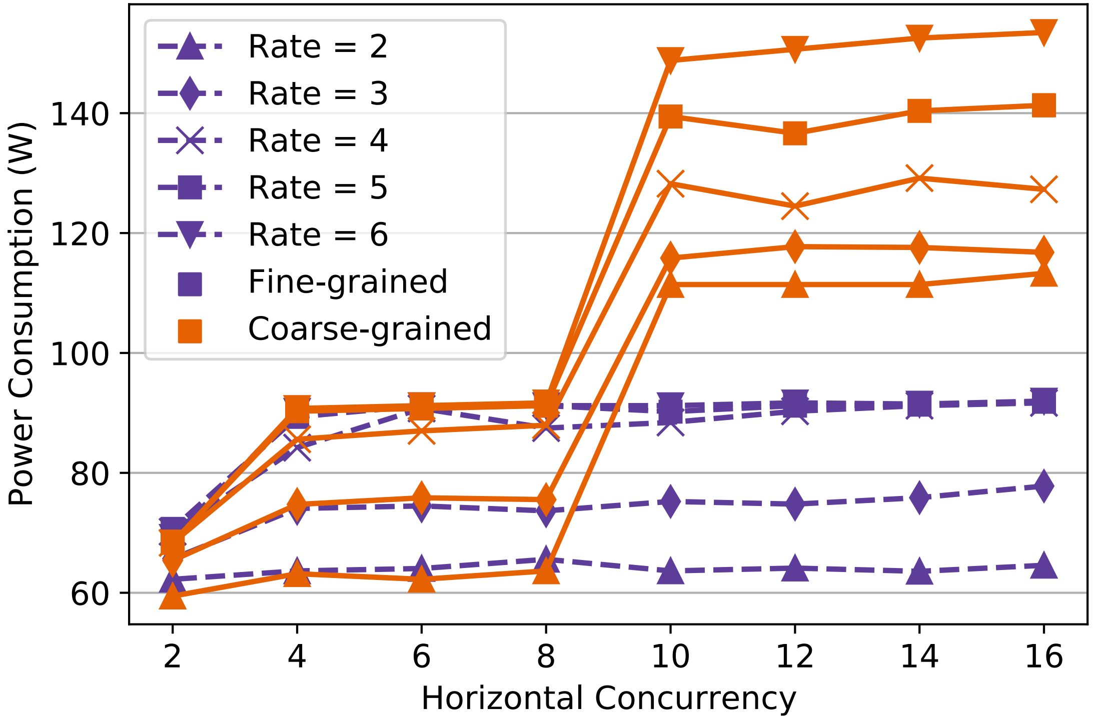


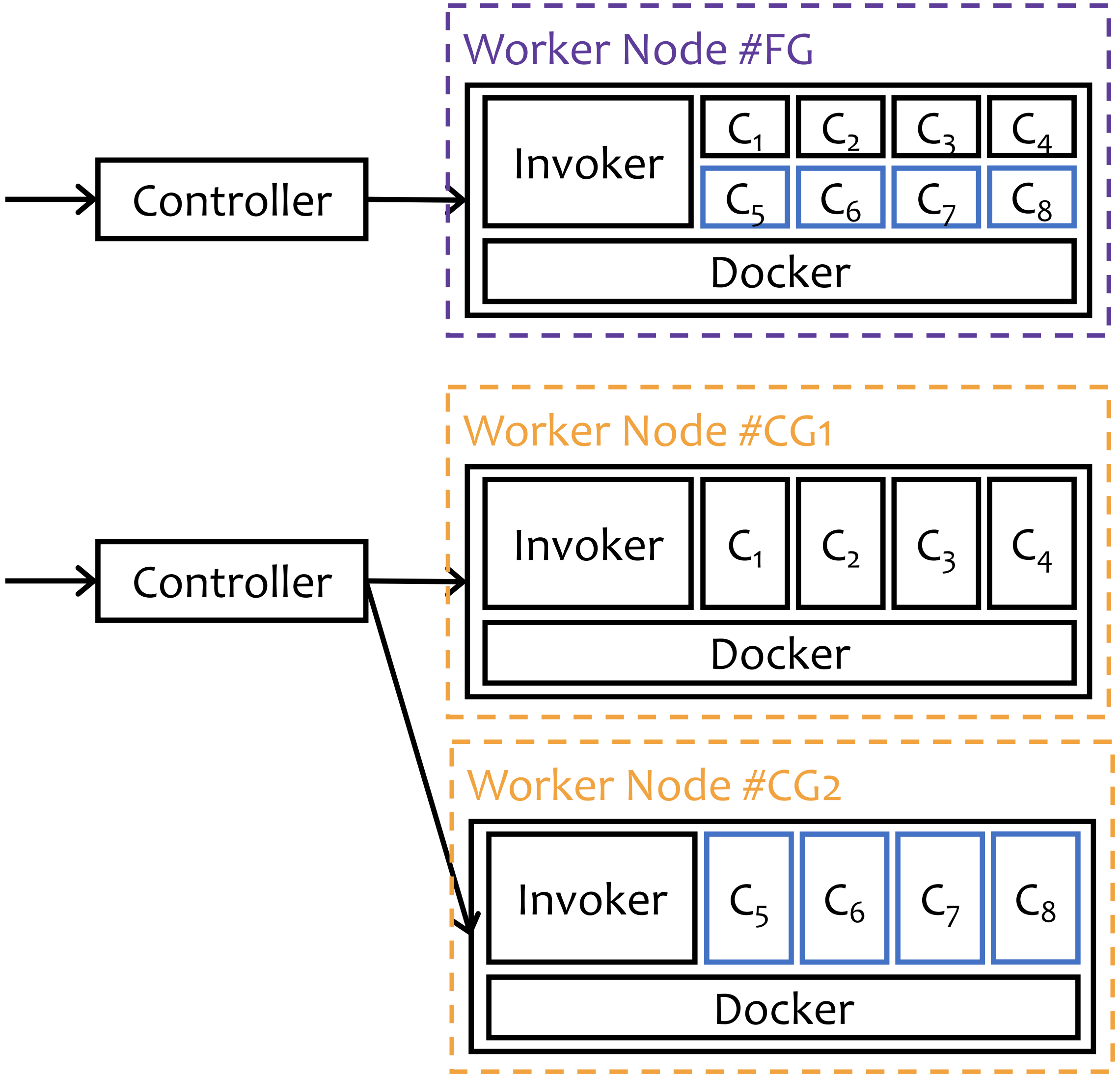


Latency-Utilization-Power Trade-off



Latency-Utilization-Power Trade-off





Latency-Utilization-Power Trade-off

[Implication] An FG policy leads to lower operation costs (up to 1.75× less) and better server utilization efficiency (up to 59% higher), while a CG policy offers the customers lower end-to end latency (up to 2× less).

The conflicting goals of the two parties raise questions,

- On the pricing model: how to balance the needs of both parties?
- On the provider-customer interface: how should resource and performance needs be conveyed?

Lower latency

Customers

Higher utilization, Lower power consumption, Higher latency

FaaS Provider

Thank you!

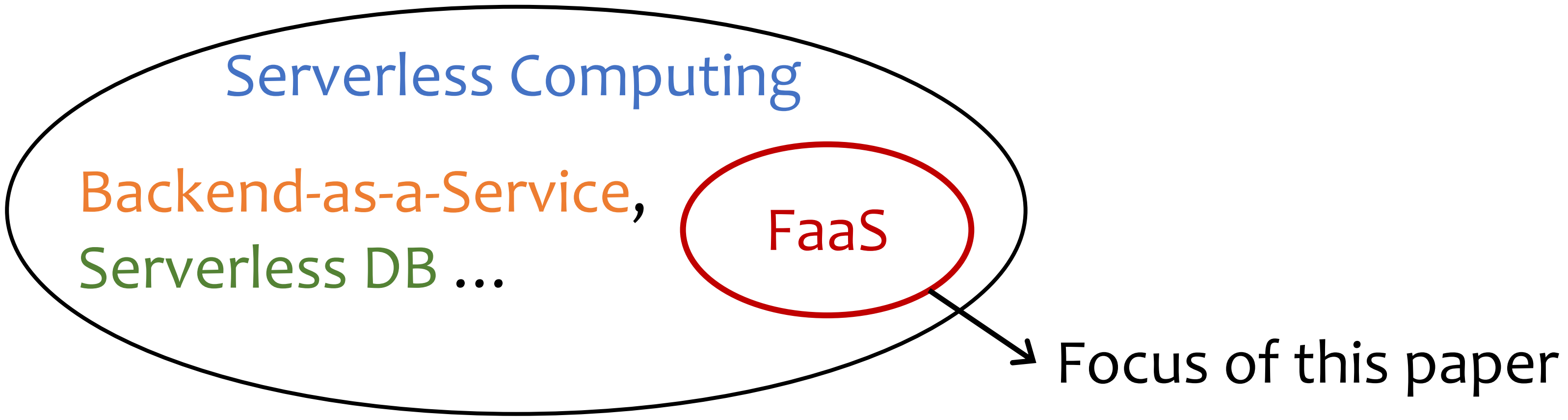
Check out our paper for more details:
https://www.serverlesscomputing.org/wosc7/papers/p1

Code available at: https://github.com/James-QiuHaoran/serverless-wosc7

Backup Slides

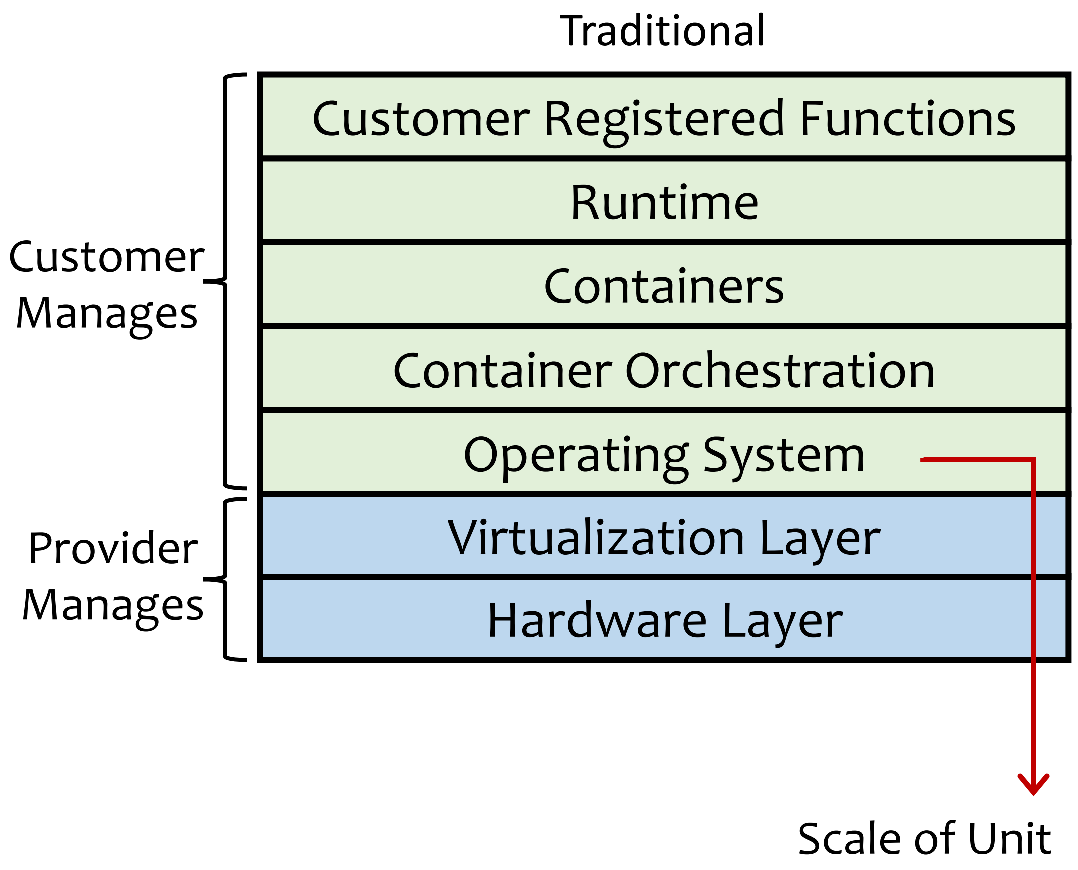
Background: Serverless Function-as-a-Service (FaaS)

- Serverless computing
  - Cloud provider allocates and scales compute resources
  - Customers are charged for the compute resources used
- Function-as-a-Service (FaaS)
  - Customers writes code that only tackles application logic; uploads it to FaaS platform
  - No need to configure/manage the provisioning and maintenance of the resources
  - E.g., Google Cloud Functions, AWS Lambda, IBM Cloud Functions, Azure Functions



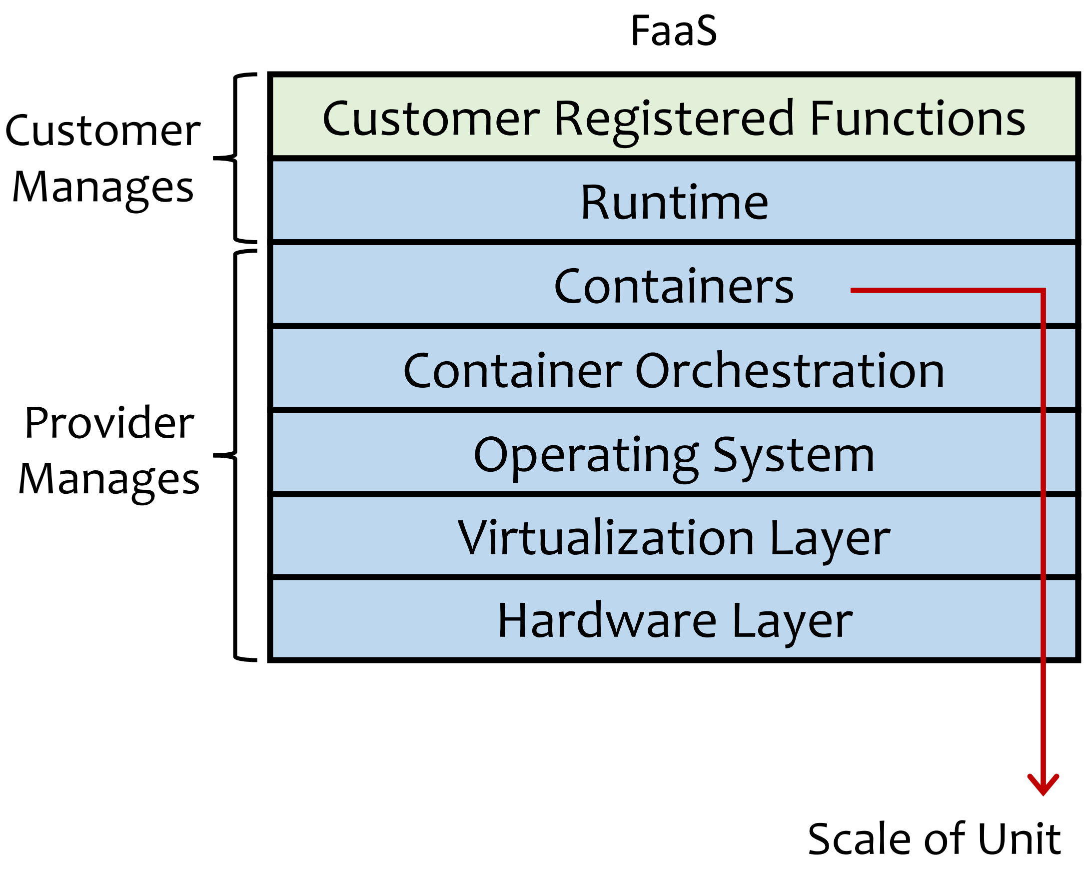
System Stack Management – Traditional vs. FaaS

- In traditional cloud computing paradigms, customers configure and pay for the cloud resources that they requested
  - E.g., the number of cores and amount of memory for a virtual machine
- Customers tend to overprovision compute resources to meet application end-to-end performance goals
- Operating system (VM) is the scale of unit



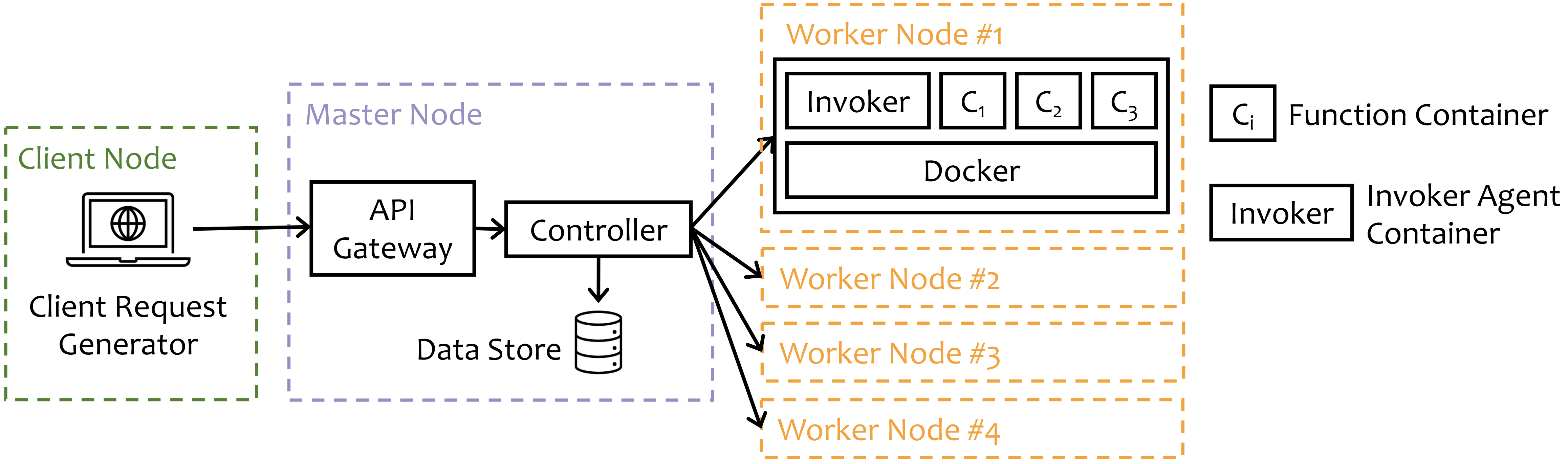
System Stack Management – Traditional vs. FaaS

- FaaS frees application developers from infrastructure management
  - E.g., resource provisioning, scaling
- Customers are charged by the compute resource usage during the execution time (no expense for idle times)
- FaaS provider creates containers for a function, scales the number of containers, and co-locates multiple containers on the same server (i.e., workload consolidation)
  - At the cost of higher end-to-end function request latencies (up to 2x from our evaluation results)

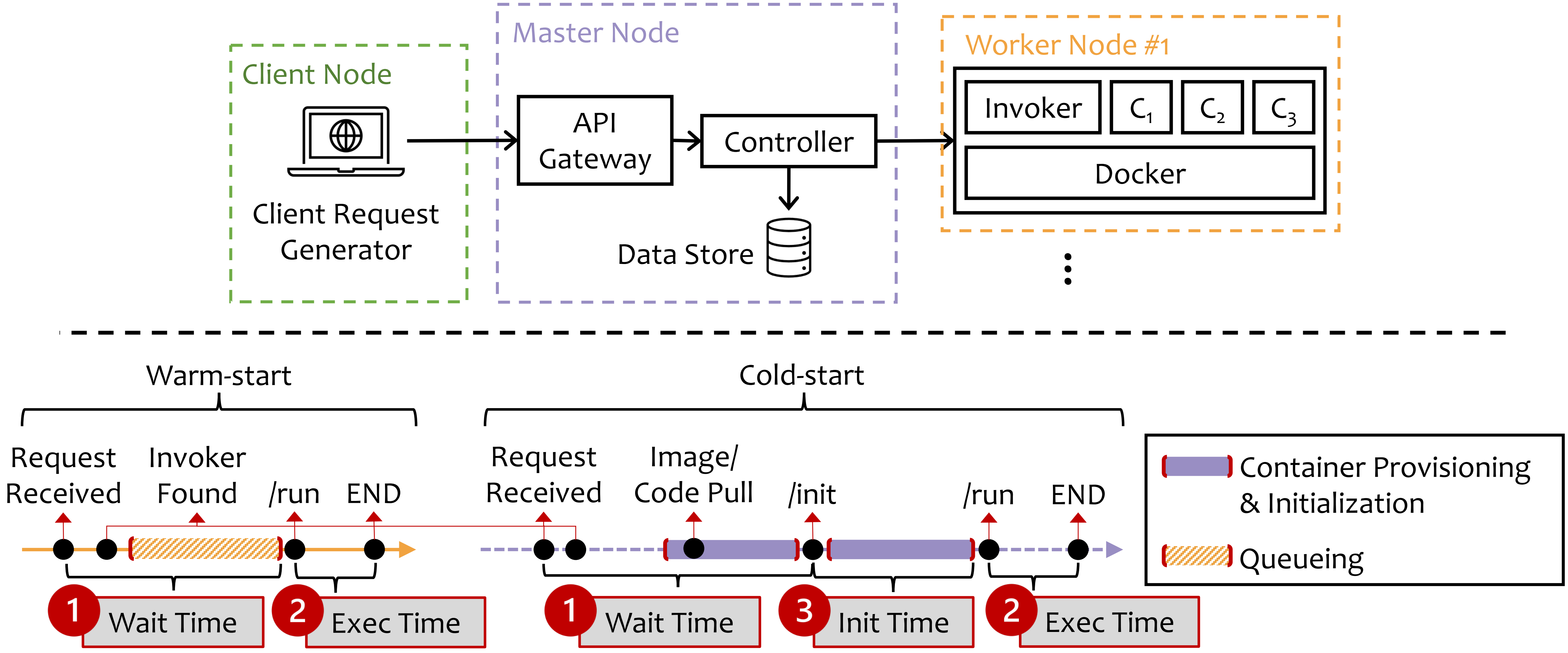


Experimental Setup Overview

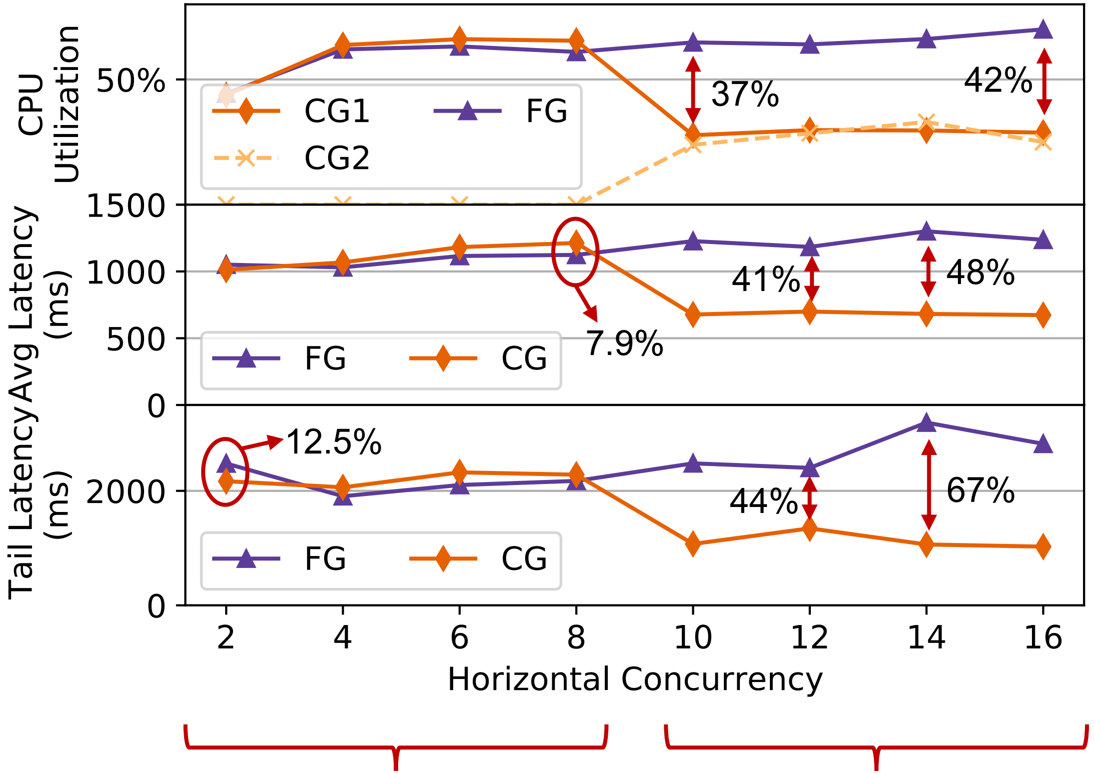
- Measurements from the execution of 2 widely used FaaS benchmark suites
  - ServerlessBench, FaaS-Profiler
- Benchmarks running on an open-sourced FaaS platform -- OpenWhisk
- Deployed on IBM Cloud with 1 master node and 4 worker nodes



Concept Overview

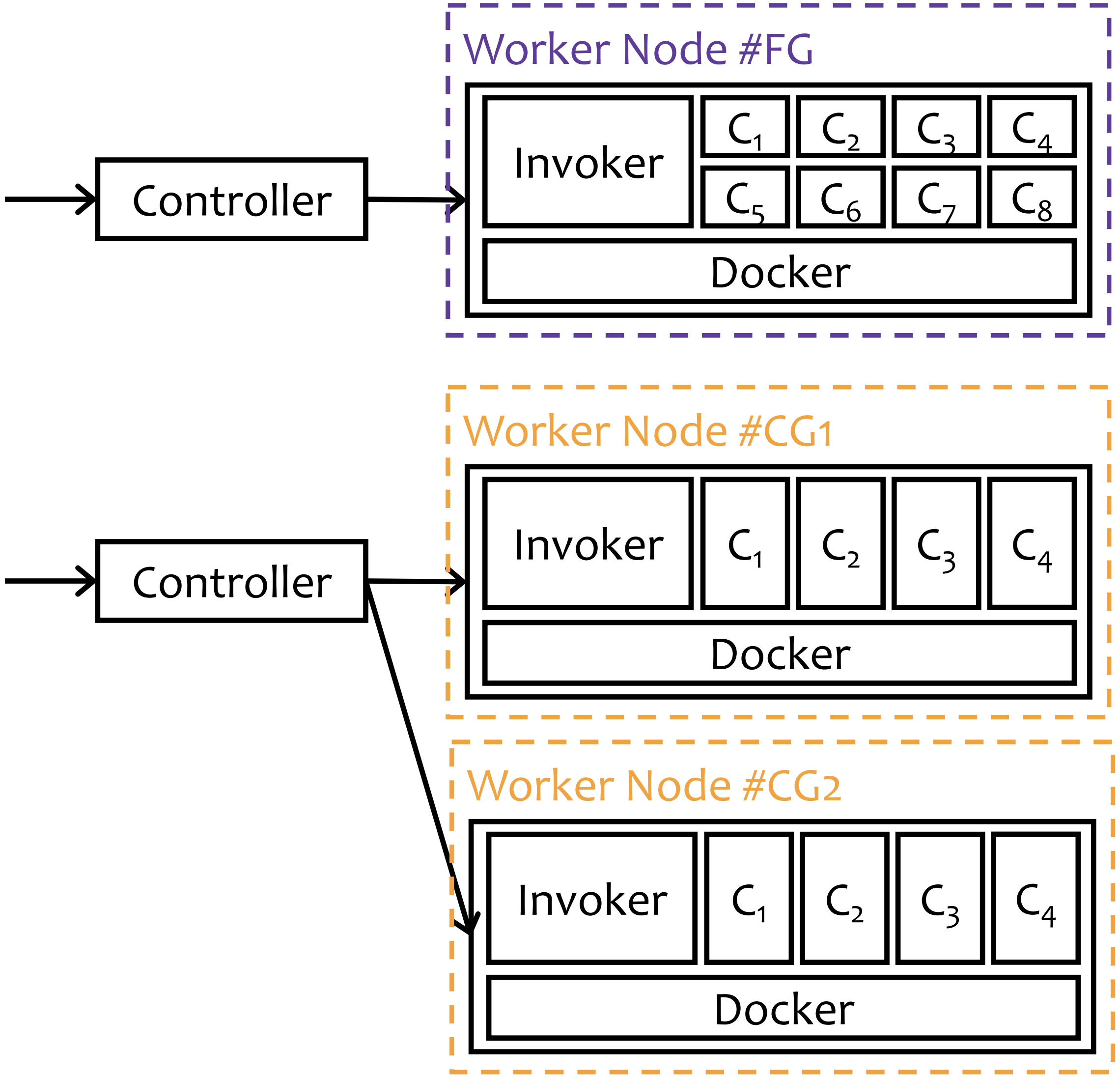


Latency Variation

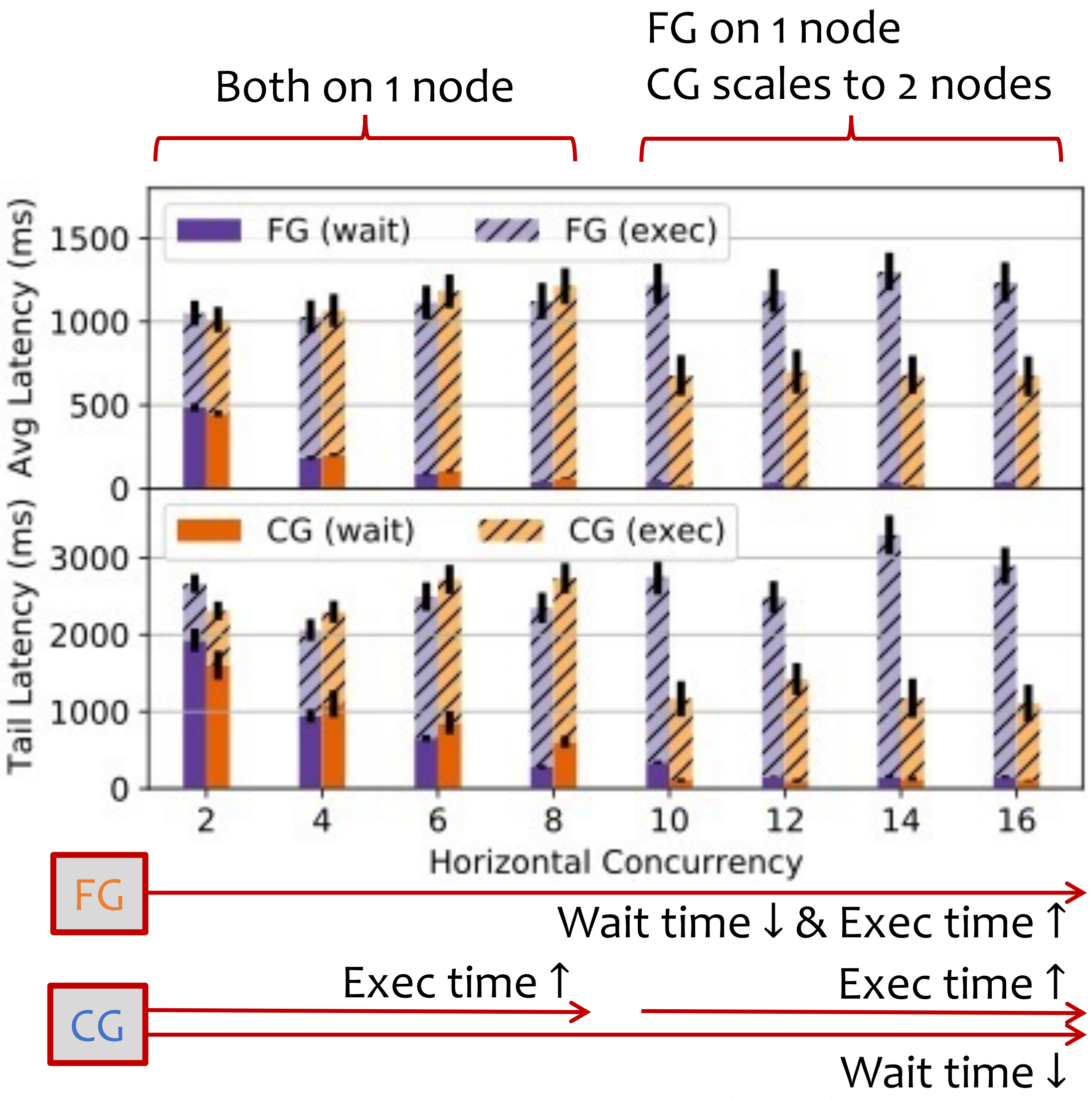


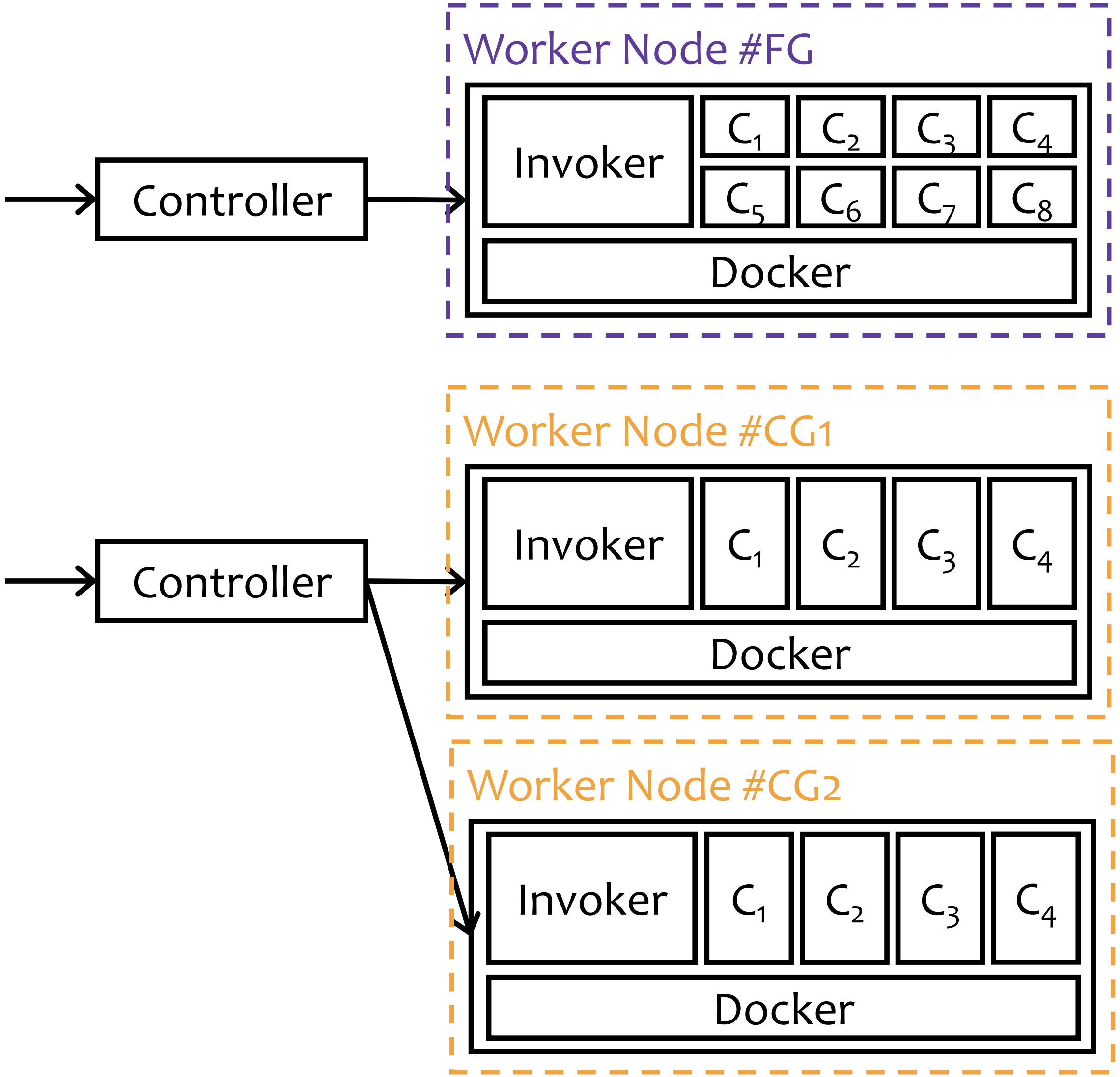
Both on 1 node

FG on 1 node
CG scales to 2 nodes

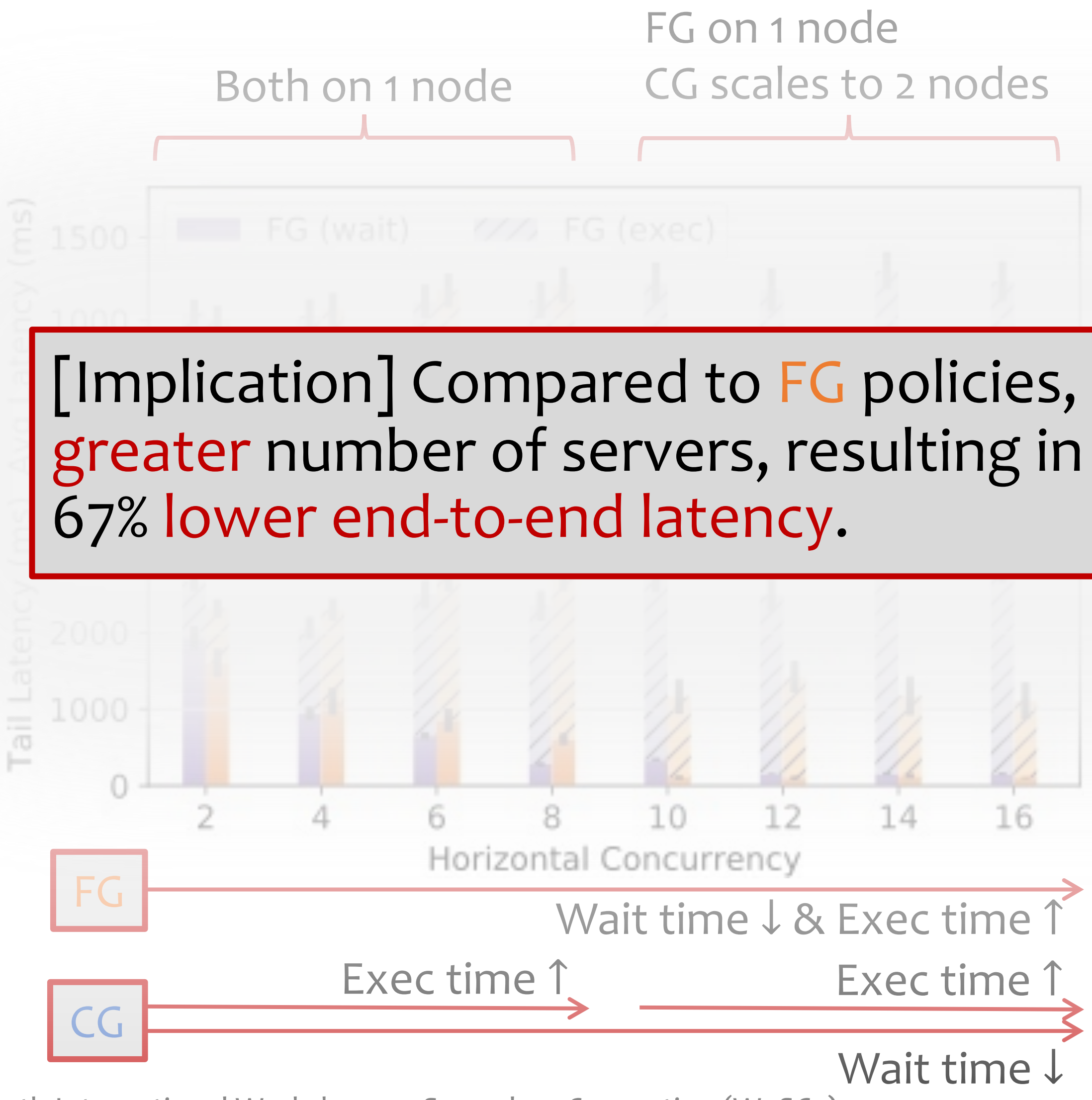


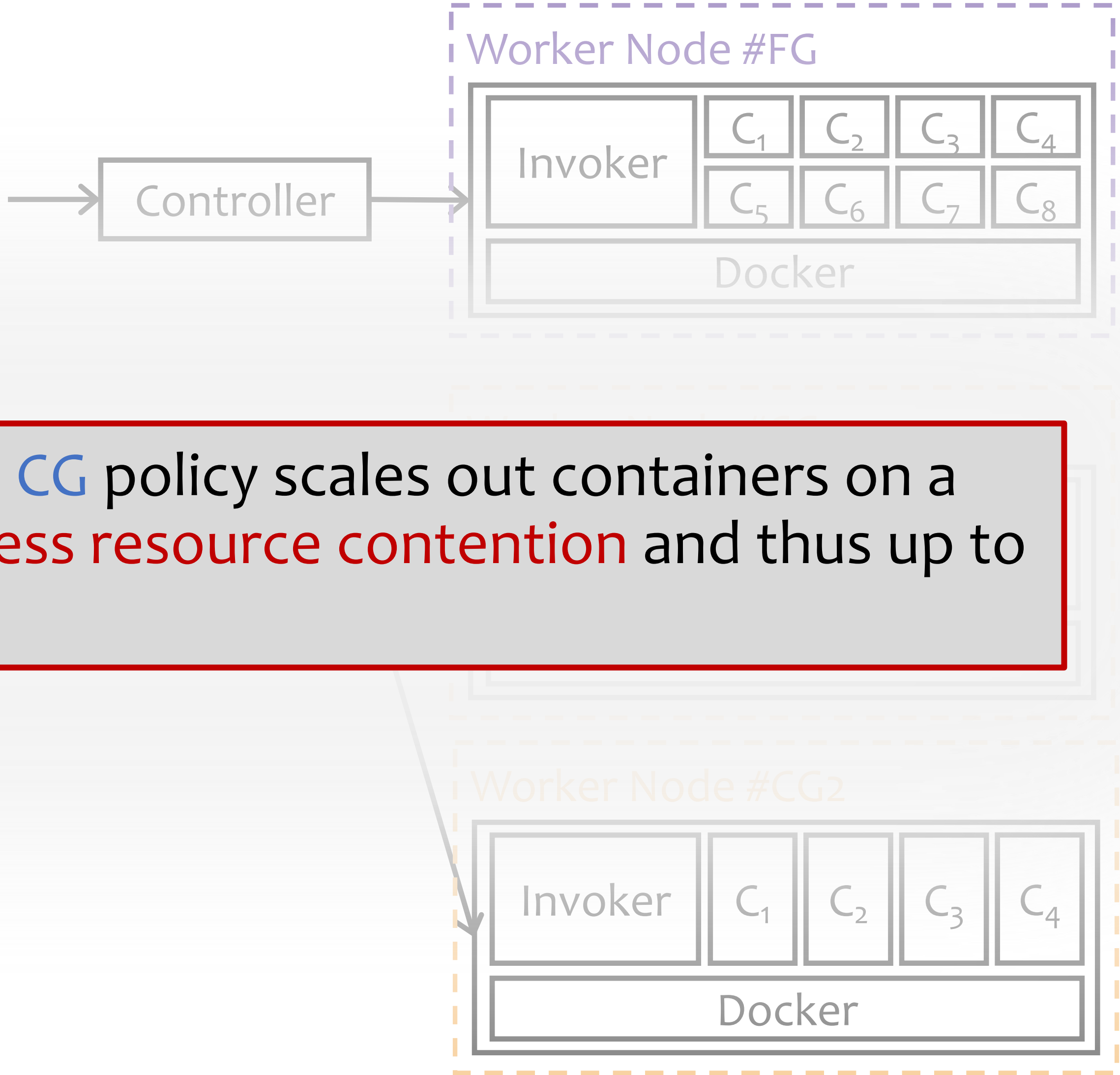
Latency Variation





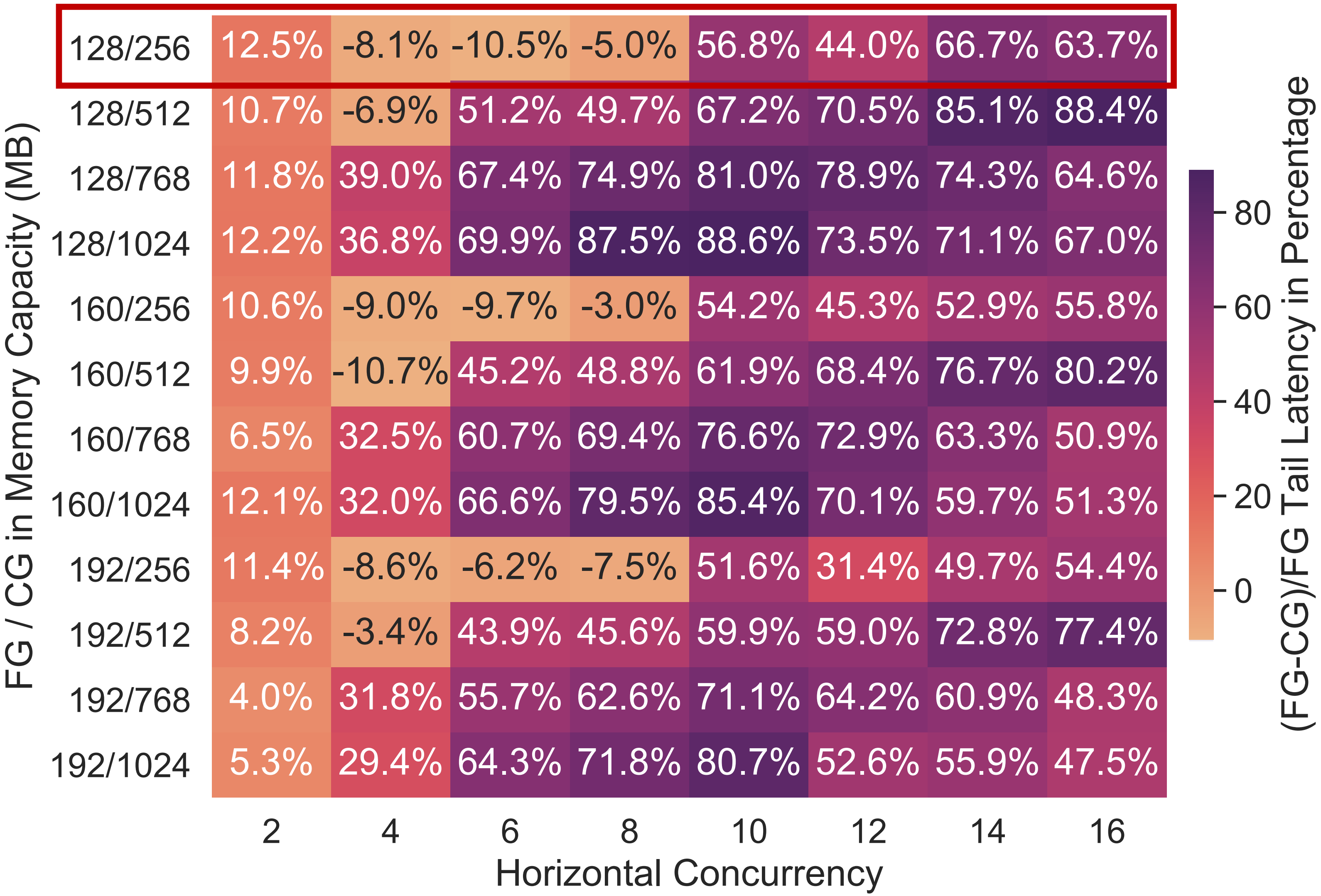
Latency Variation





[Implication] Compared to FG policies, a CG policy scales out containers on a greater number of servers, resulting in less resource contention and thus up to 67% lower end-to-end latency.

Latency-Utilization-Power Trade-off



Performance Interference

CPU Time Contention

	C1	C2	C3
Base64 (Avg)	166.8%	1497.7%	1712.7%
Base64 (Tail)	182.0%	1451.4%	1656.4%
Primes (Avg)	114.9%	184.3%	250.0%
Primes (Tail)	131.8%	272.4%	333.2%
Markdown2HTML (Avg)	404.5%	1120.0%	1299.8%
Markdown2HTML (Tail)	410.8%	1174.9%	1383.9%
Sentiment (Avg)	128.7%	239.0%	349.3%
Sentiment (Tail)	158.0%	321.7%	427.2%
Image-Resize (Avg)	984.1%	2920.5%	3942.5%
Image-Resize (Tail)	828.2%	2883.3%	3258.5%

1500 3000

Normalized Latency in Percentage

Memory Bandwidth Contention

	M1	M2	M3	M4	M5
Base64 (Avg)	131.3%	175.8%	261.0%	521.3%	663.9%
Base64 (Tail)	200.0%	223.8%	262.4%	517.2%	671.7%
Primes (Avg)	105.2%	108.8%	114.1%	126.2%	139.5%
Primes (Tail)	136.6%	148.2%	153.5%	167.0%	177.8%
Markdown2HTML (Avg)	100.1%	103.7%	492.1%	525.9%	806.7%
Markdown2HTML (Tail)	121.8%	125.9%	463.8%	549.6%	828.0%
Sentiment (Avg)	102.9%	115.8%	122.2%	135.7%	164.2%
Sentiment (Tail)	112.9%	180.4%	183.4%	193.0%	204.6%
Image-Resize (Avg)	133.9%	134.1%	898.6%	1743.7%	2400.6%
Image-Resize (Tail)	125.7%	127.9%	1241.5%	1273.4%	2151.5%

1000 2000

Normalized Latency in Percentage

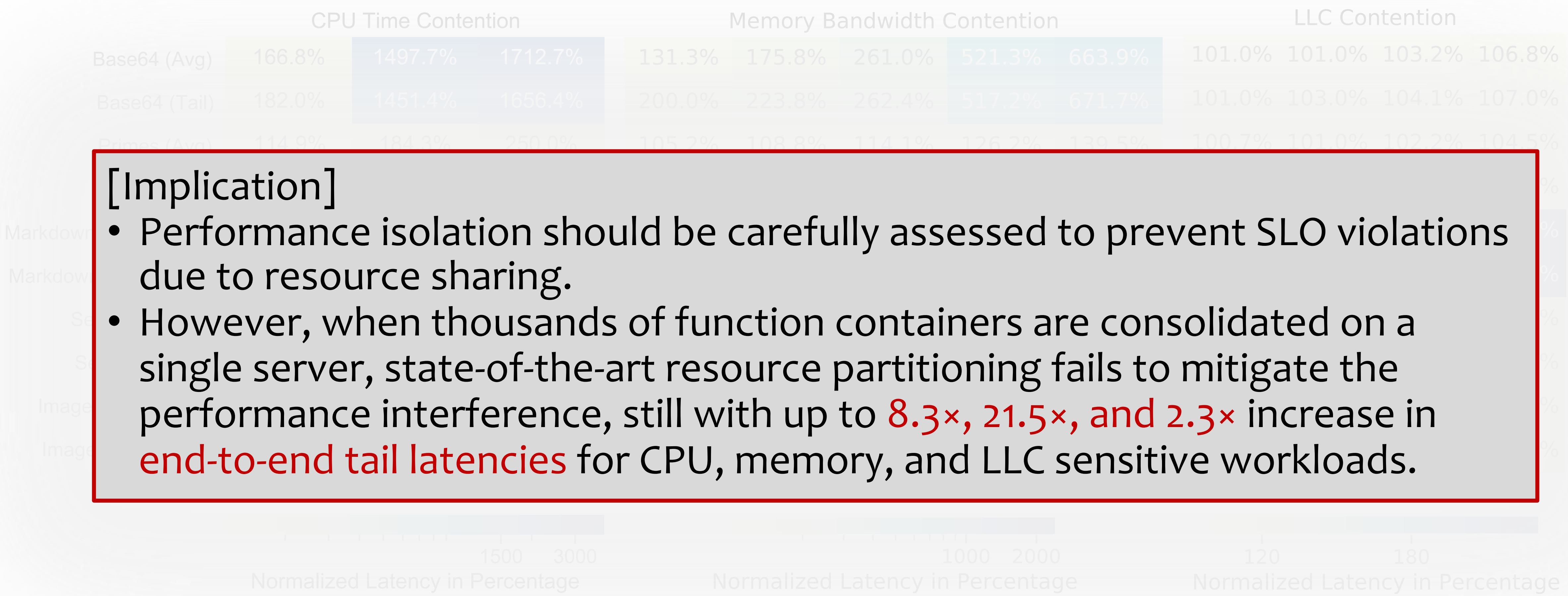
LLC Contention

	L1	L2	L3	L4
Base64 (Avg)	101.0%	101.0%	103.2%	106.8%
Base64 (Tail)	101.0%	103.0%	104.1%	107.0%
Primes (Avg)	100.7%	101.0%	102.2%	104.5%
Primes (Tail)	100.2%	100.3%	101.4%	105.6%
Markdown2HTML (Avg)	106.9%	136.6%	175.4%	230.6%
Markdown2HTML (Tail)	101.3%	126.6%	164.2%	229.8%
Sentiment (Avg)	100.7%	101.4%	103.7%	123.7%
Sentiment (Tail)	100.9%	101.5%	101.5%	128.8%
Image-Resize (Avg)	103.1%	106.1%	111.6%	121.8%
Image-Resize (Tail)	103.7%	112.4%	114.3%	125.6%

120 180

Normalized Latency in Percentage

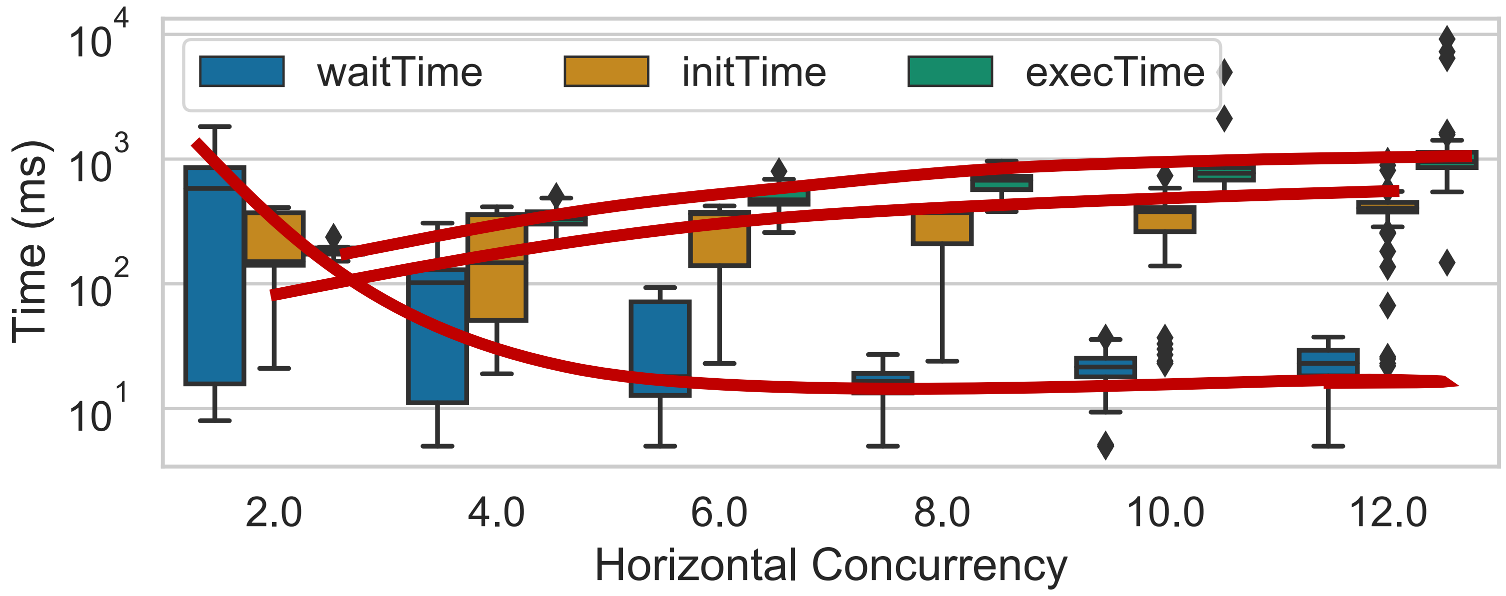
Performance Interference



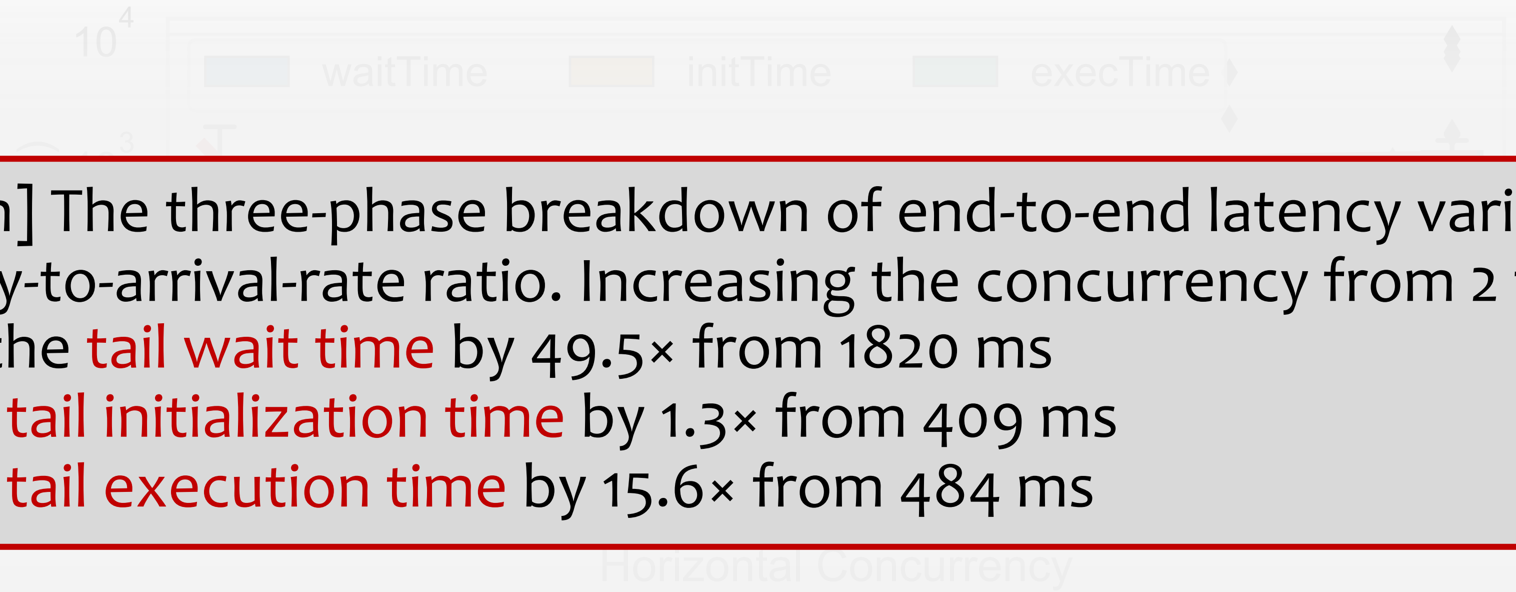
[Implication]

- Performance isolation should be carefully assessed to prevent SLO violations due to resource sharing.
- However, when thousands of function containers are consolidated on a single server, state-of-the-art resource partitioning fails to mitigate the performance interference, still with up to 8.3×, 21.5×, and 2.3× increase in end-to-end tail latencies for CPU, memory, and LLC sensitive workloads.

End-to-end Latency Breakdown



End-to-end Latency Breakdown



[Implication] The three-phase breakdown of end-to-end latency varies with the concurrency-to-arrival-rate ratio. Increasing the concurrency from 2 to 12:

- Reduces the tail wait time by 49.5× from 1820 ms
- Increases tail initialization time by 1.3× from 409 ms
- Increases tail execution time by 15.6× from 484 ms