

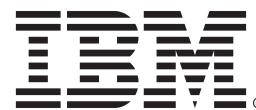
Is Function-as-a-Service (FaaS) a Good Fit for Latency-critical Services

**Haoran Qiu¹, Saurabh Jha¹, Subho Banerjee¹, Archit Patke¹, Chen Wang², Hubertus Franke²
Zbigniew Kalbarczyk¹, Ravishakar Iyer¹**

¹ University of Illinois, Urbana-Champaign

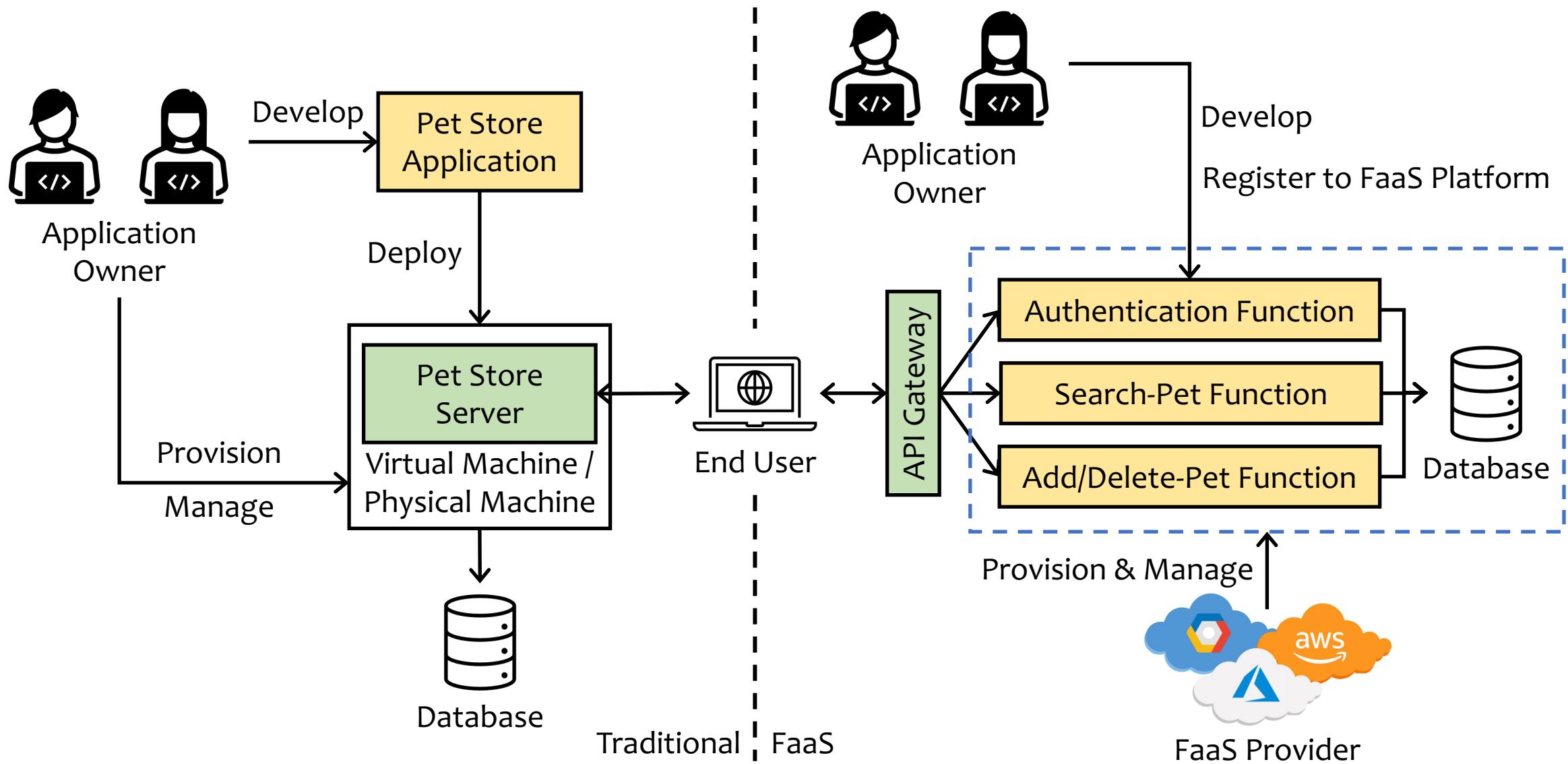


² IBM Research

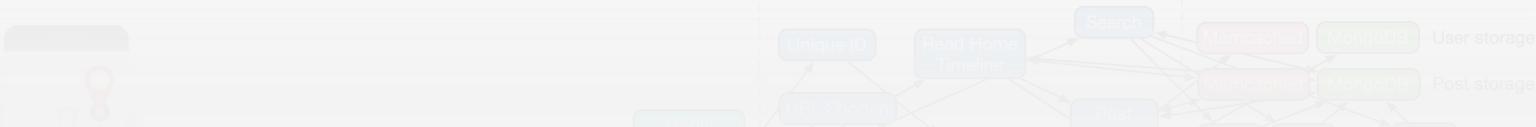


Paper link: <https://www.serverlesscomputing.org/wosc7/papers/p1>
Seventh International Workshop on Serverless Computing (WoSC7) 2021

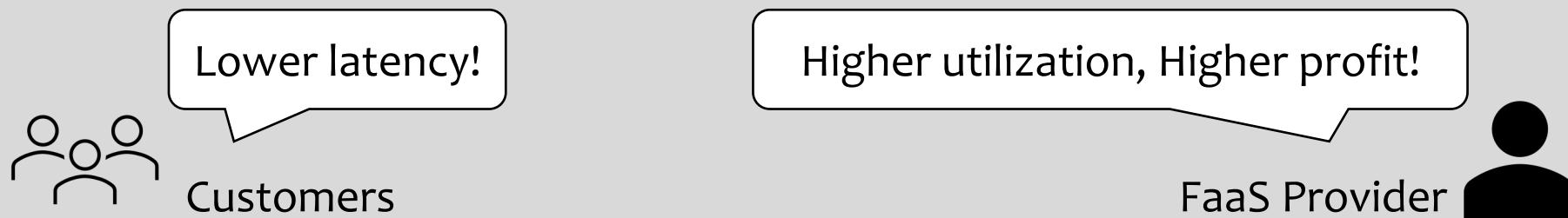
Traditional vs. FaaS – An Example



Latency-critical Services



- 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?



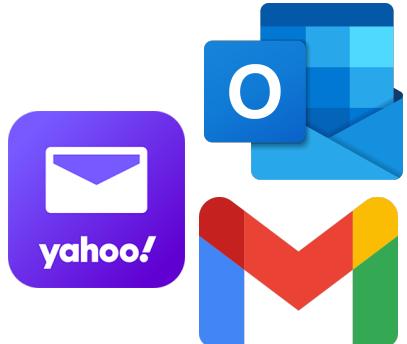
Web Mail Service

Machine Translation

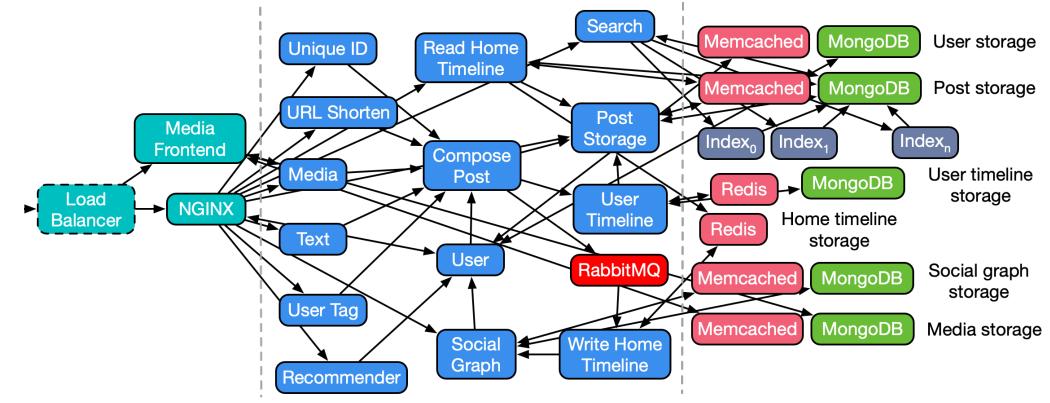
Latency-critical Services



Online Navigation



Web Mail Service



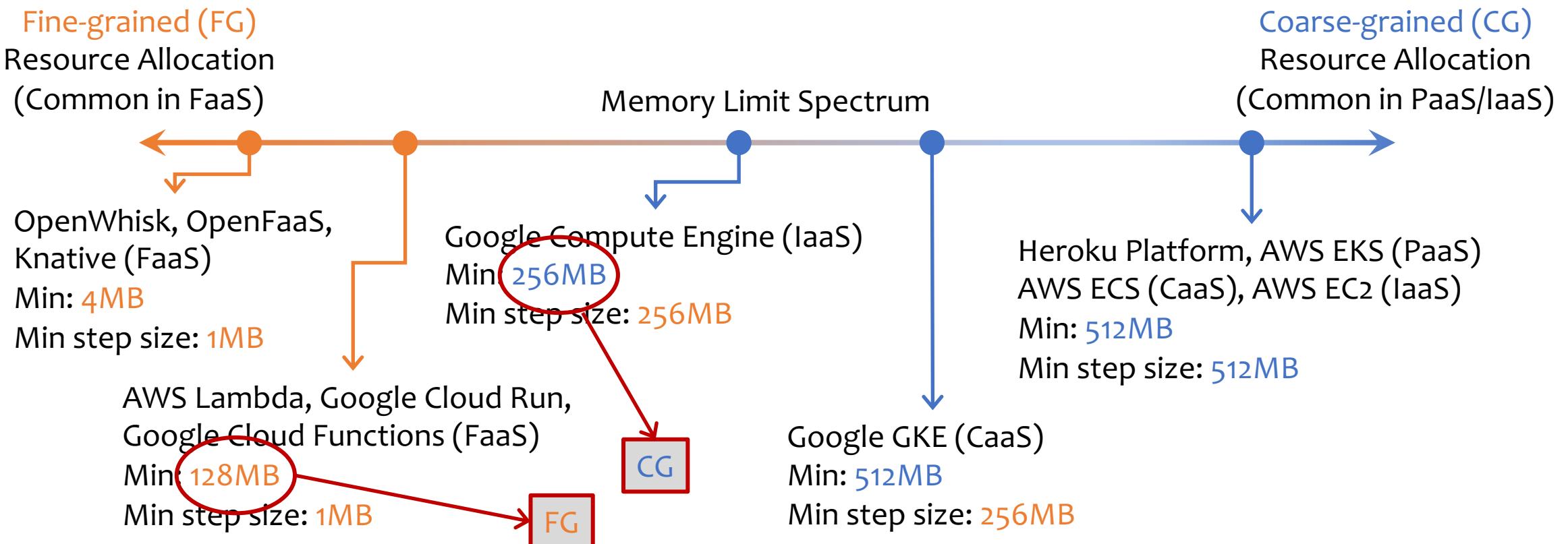
Social Network



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%
- 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?

This Talk

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%
- 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?

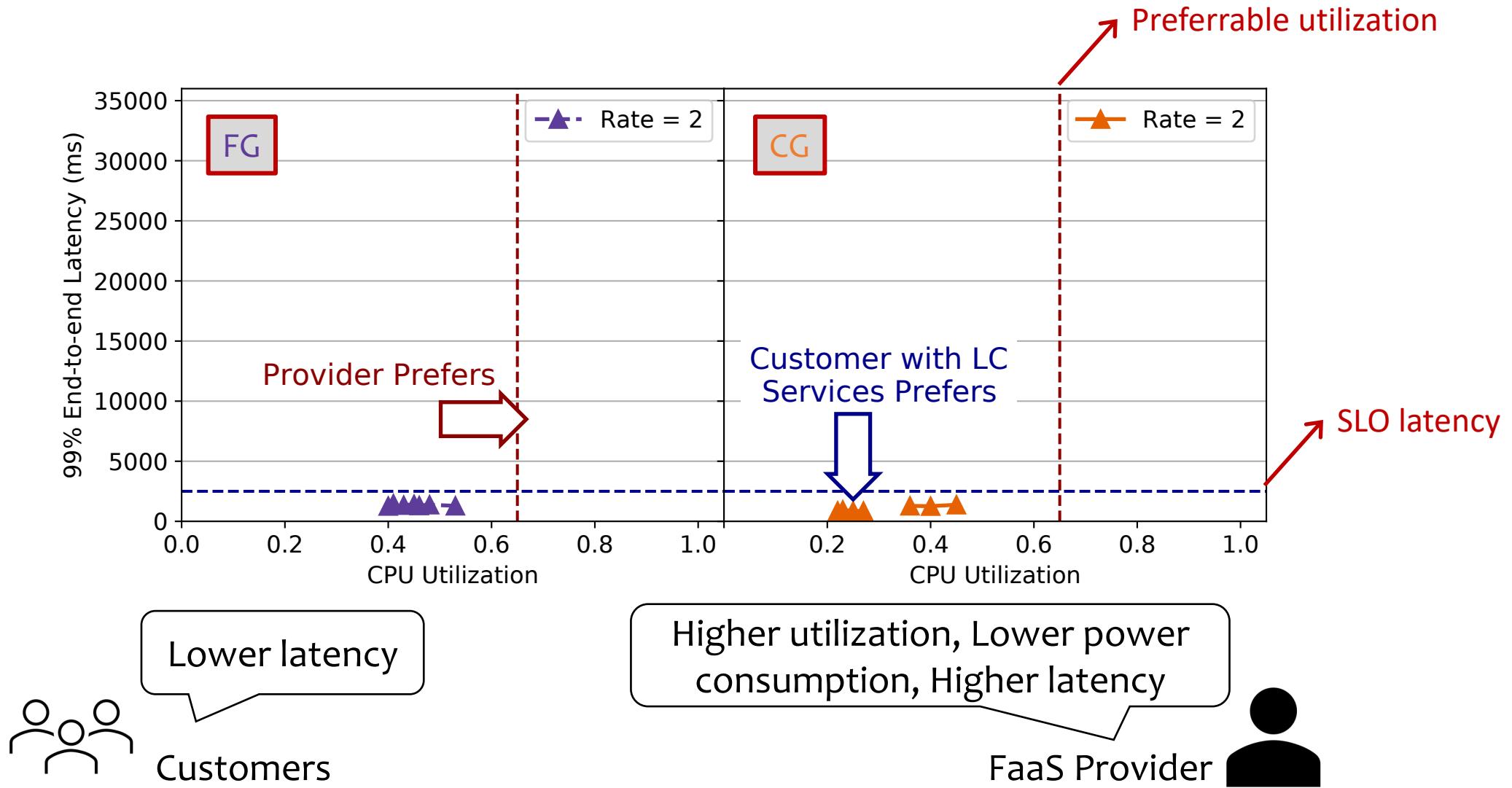
This Talk

Goal and Key Findings

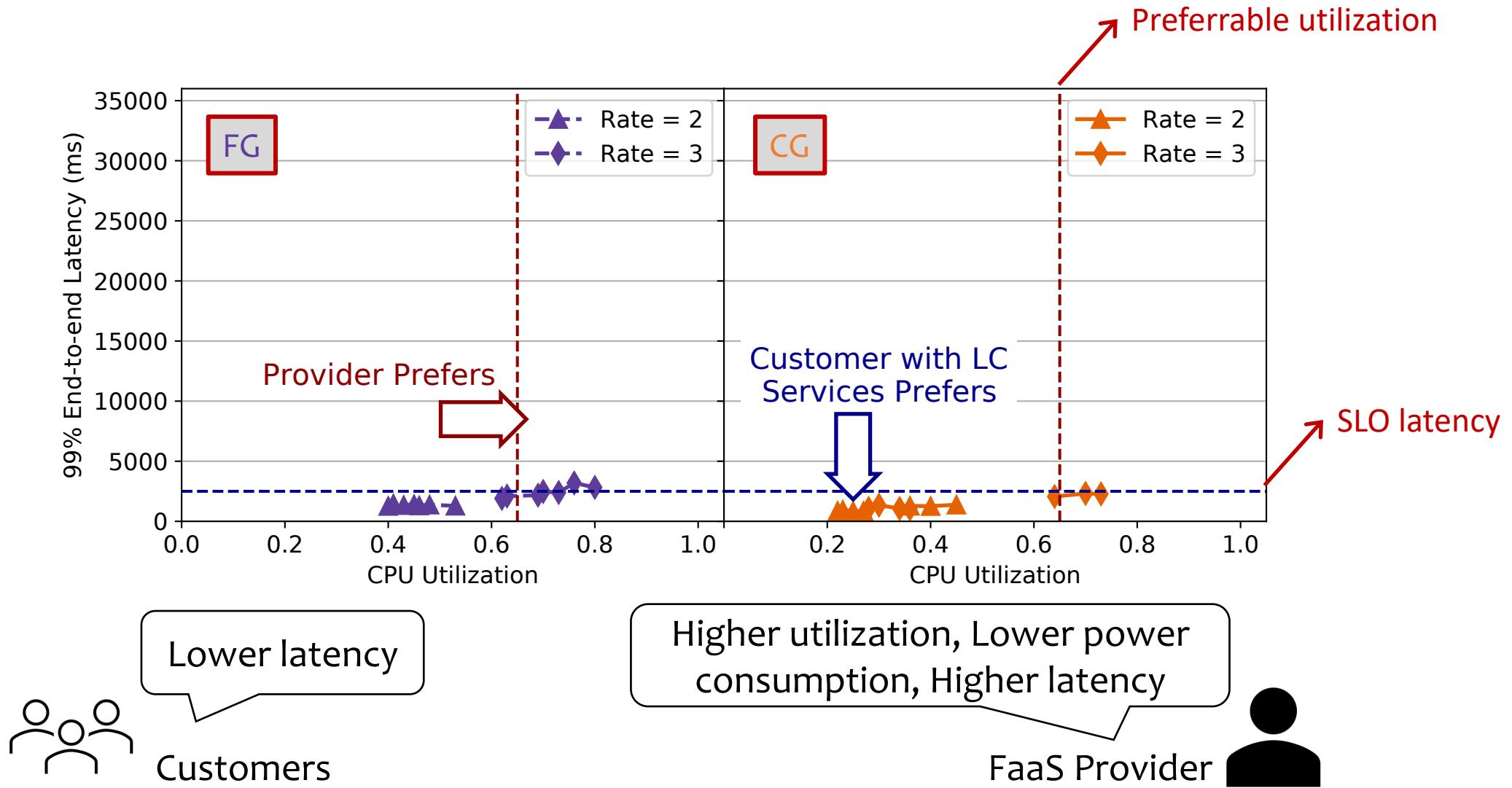
- What is the trade-off among power consumption, CPU utilization, and end-to-end latency in the decision-making of choosing a workload consolidation policy?
 - Increasing resource granularity (e.g., increasing a container's allocated memory limit from 128 MB to 256 MB):
 - Reduces tail latency by up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

This Talk

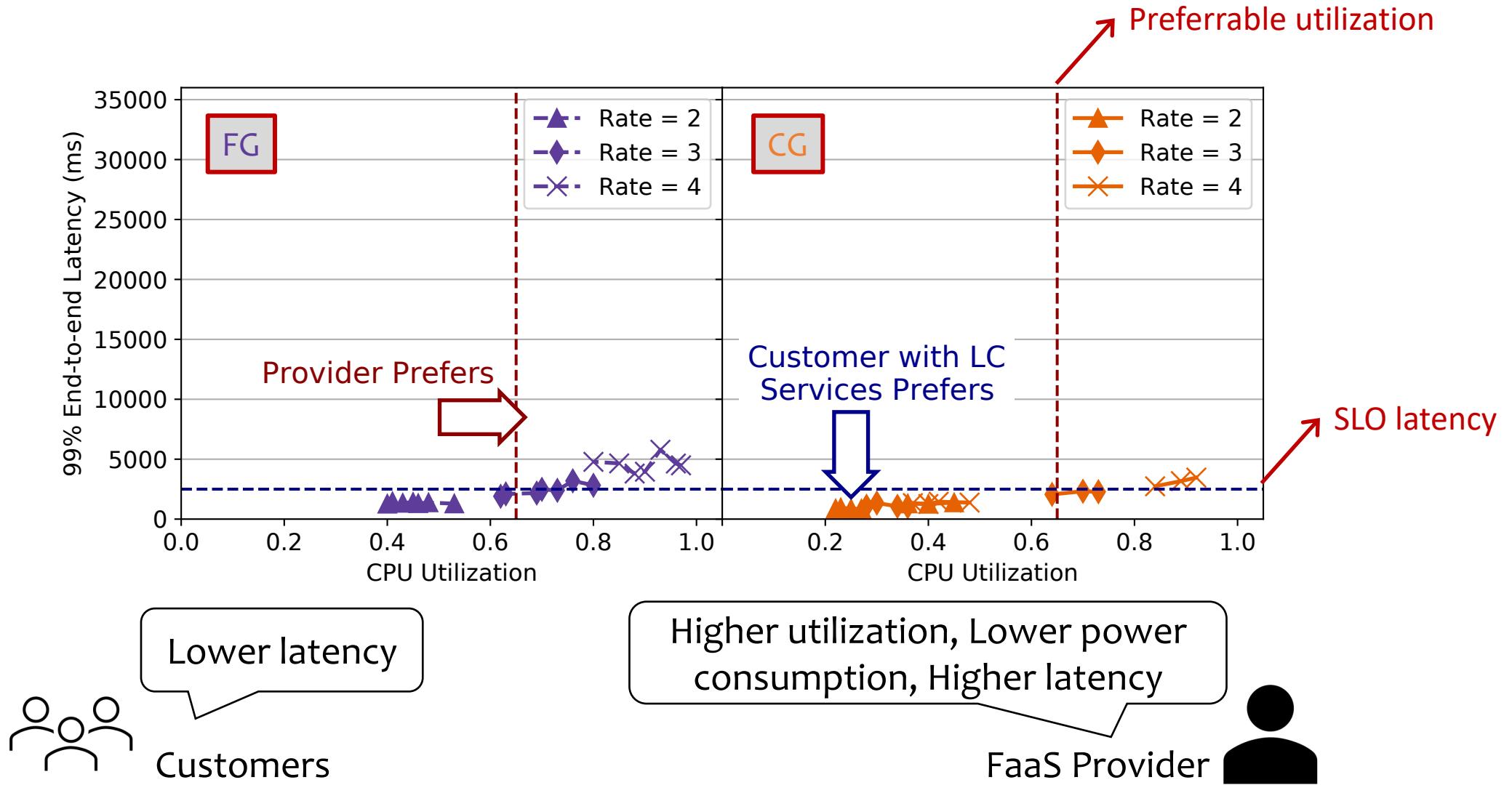
Latency-Utilization-Power Trade-off



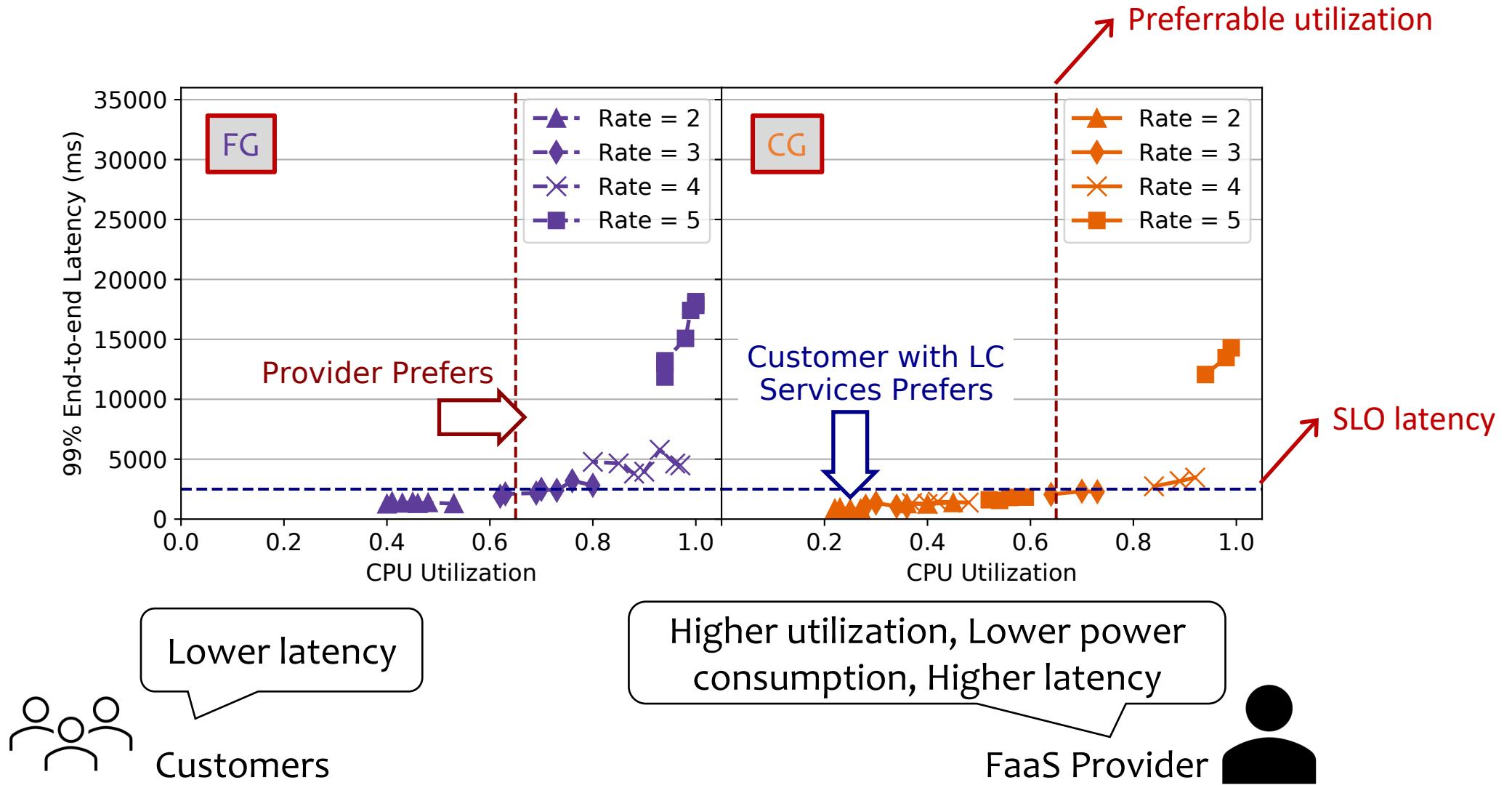
Latency-Utilization-Power Trade-off



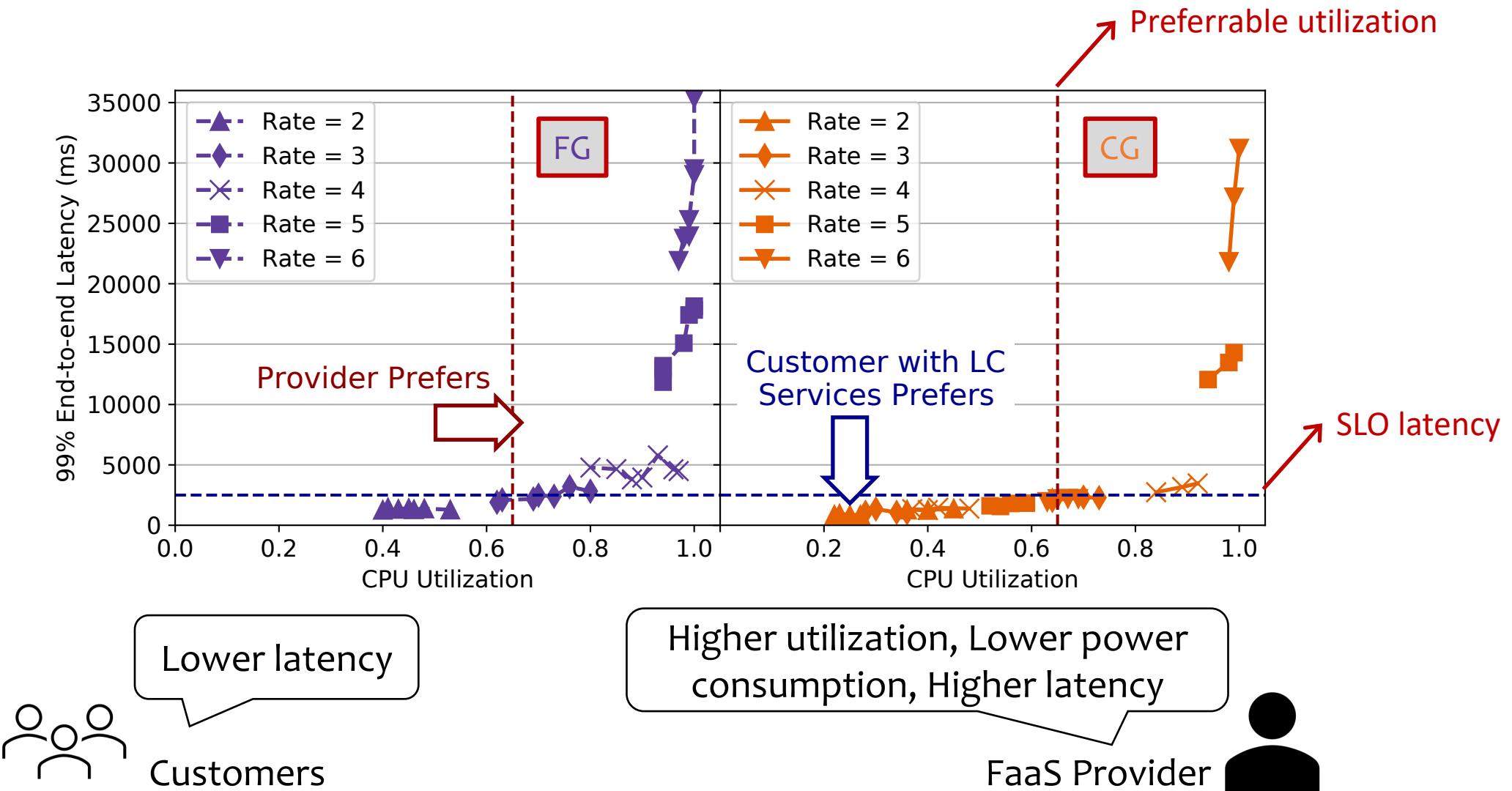
Latency-Utilization-Power Trade-off



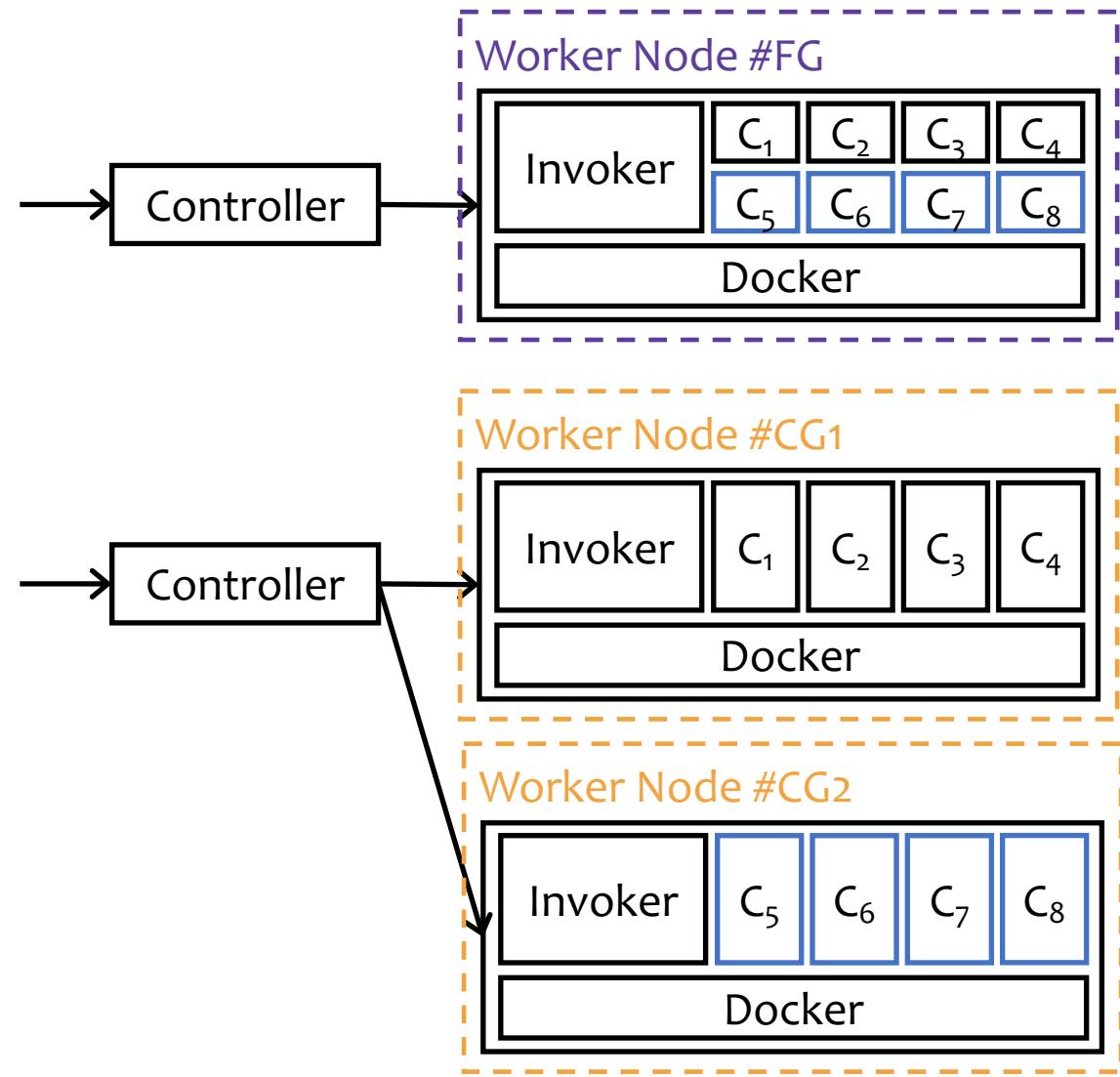
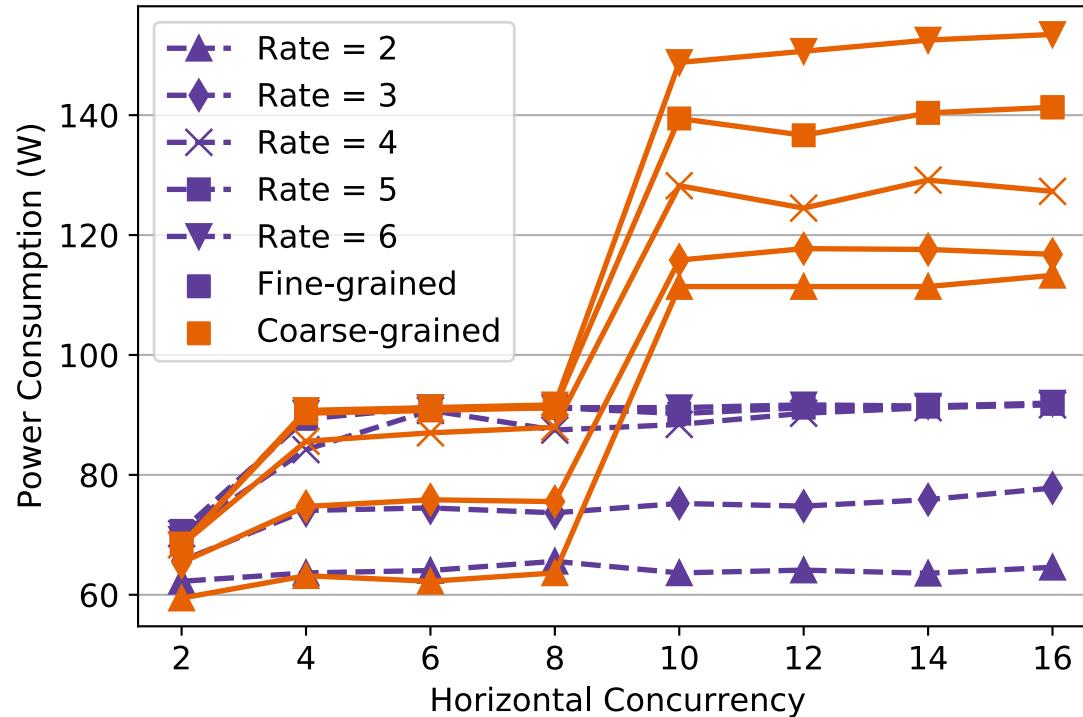
Latency-Utilization-Power Trade-off



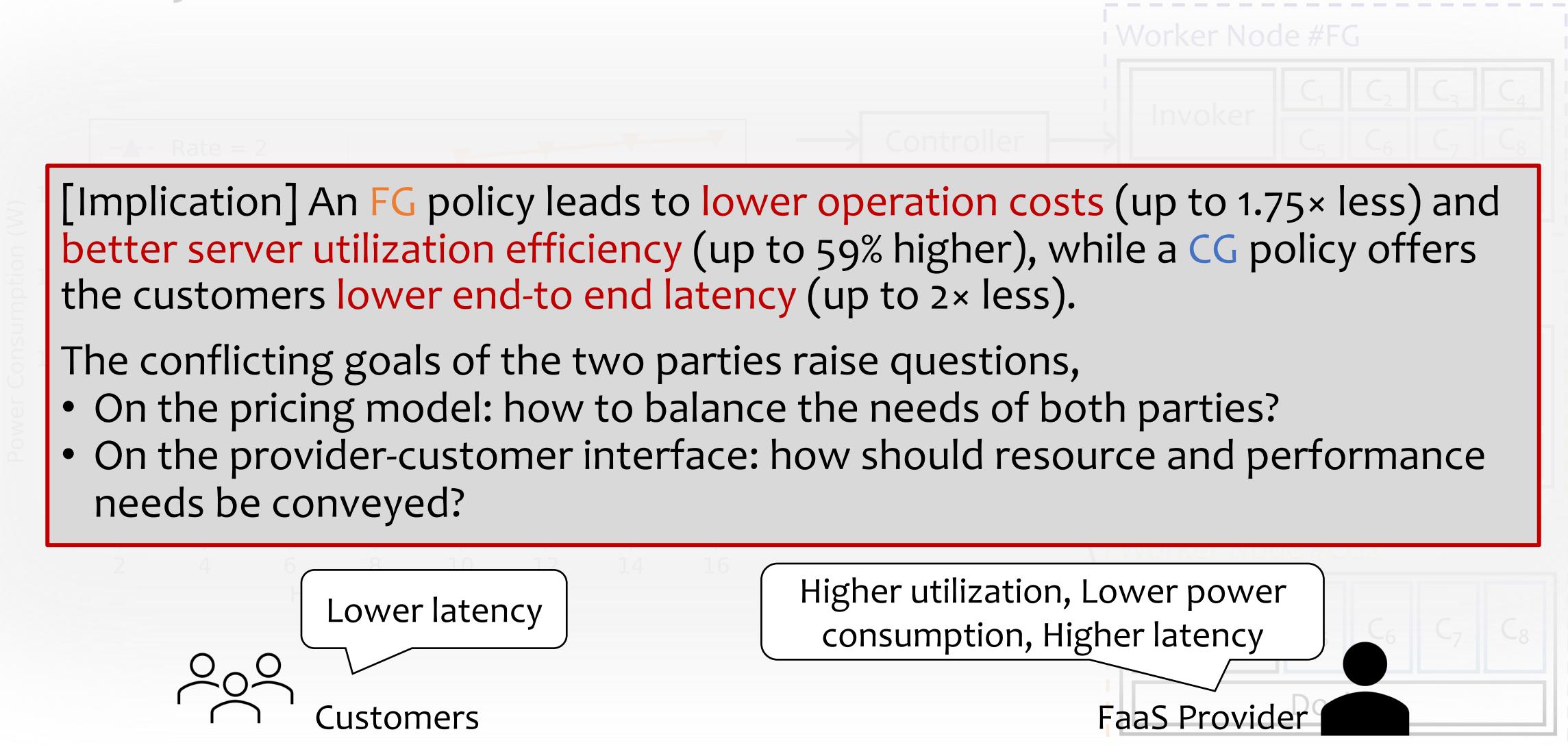
Latency-Utilization-Power Trade-off



Latency-Utilization-Power Trade-off



Latency-Utilization-Power Trade-off



Thank you!

Check out our paper for more details:

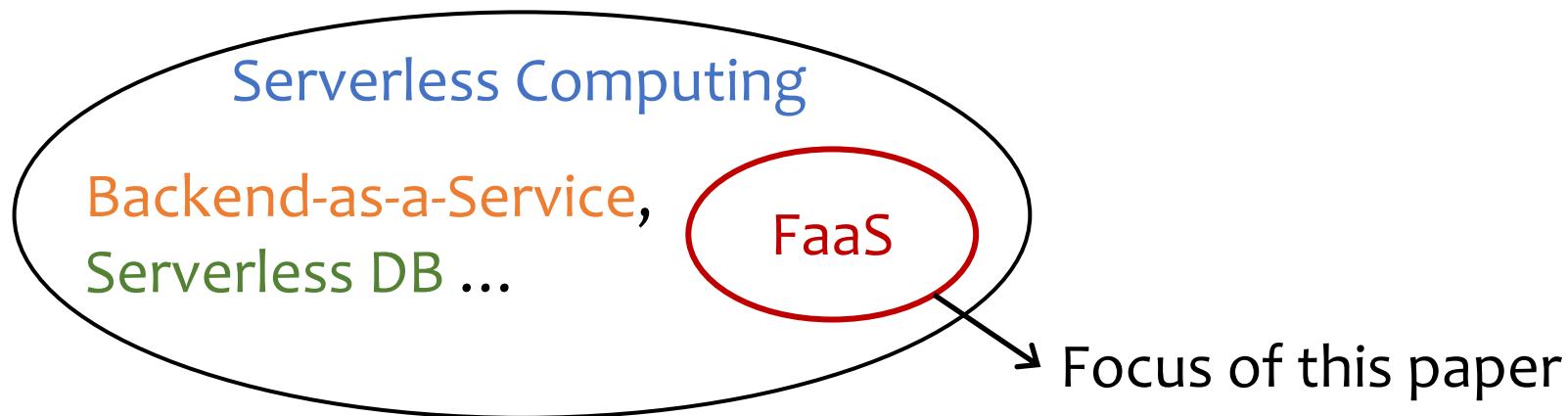
<https://www.serverlesscomputing.org/wosc7/papers/p1>

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

Backup Slides

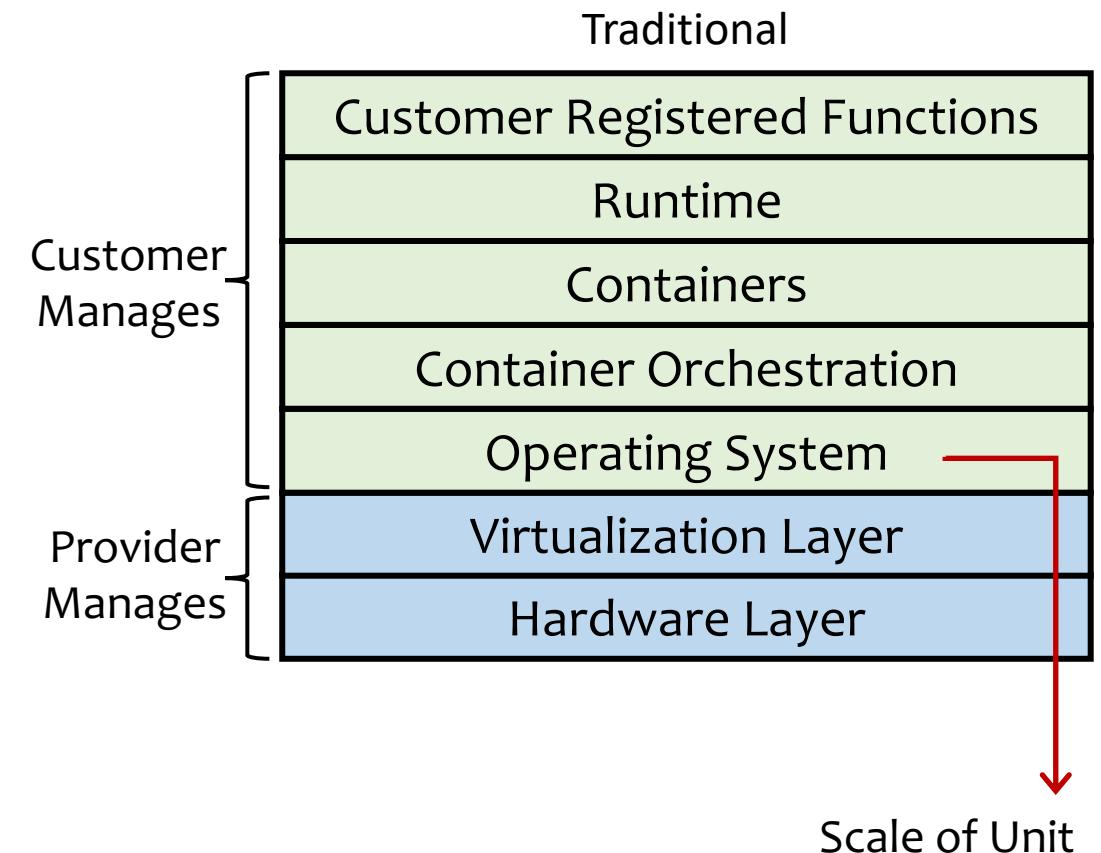
Background: Serverless Function-as-a-Service (FaaS)

- 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



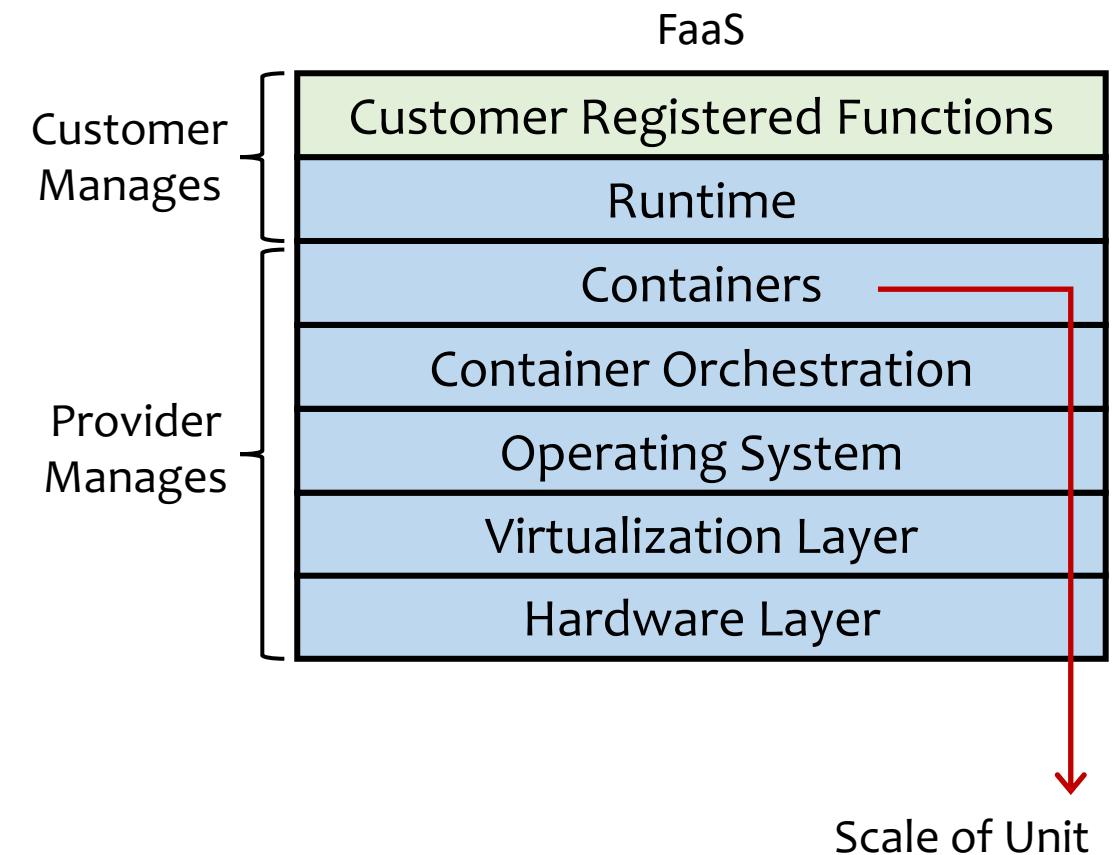
System Stack Management – Traditional vs. FaaS

- 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



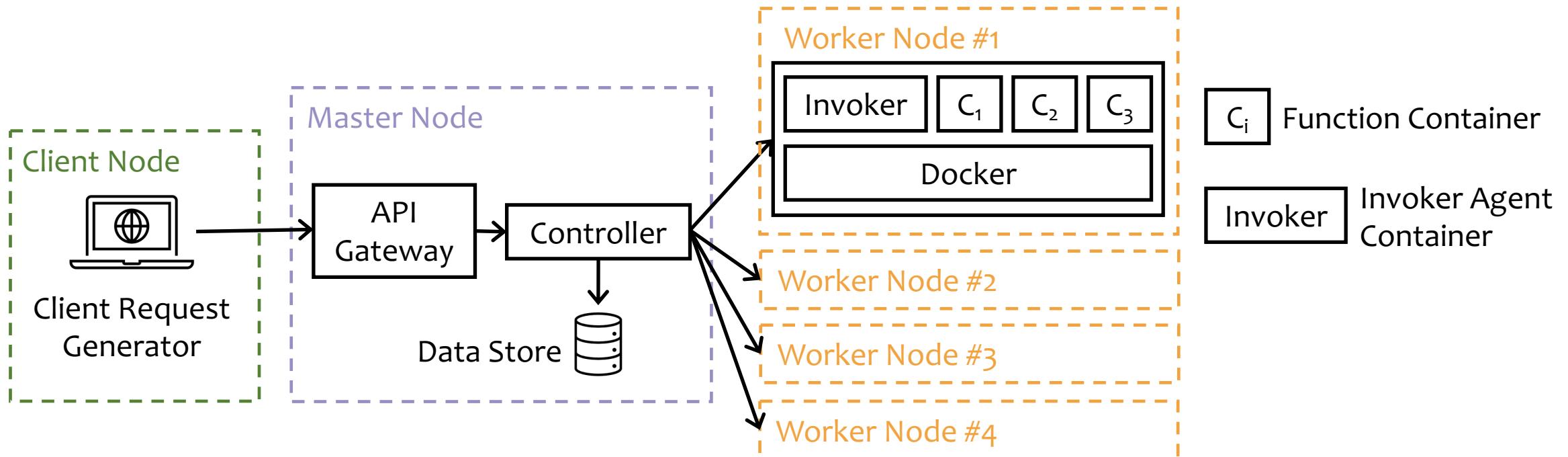
System Stack Management – Traditional vs. FaaS

- 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)

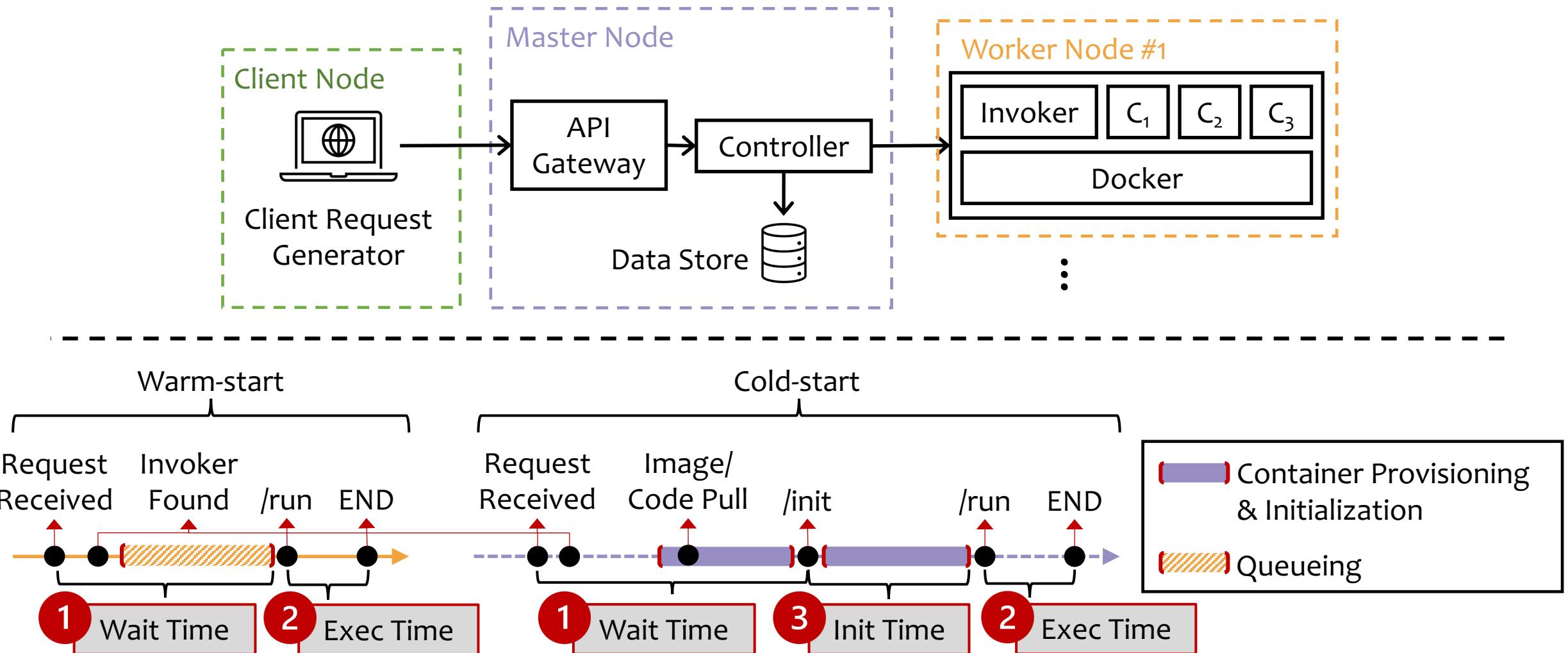


Experimental Setup Overview

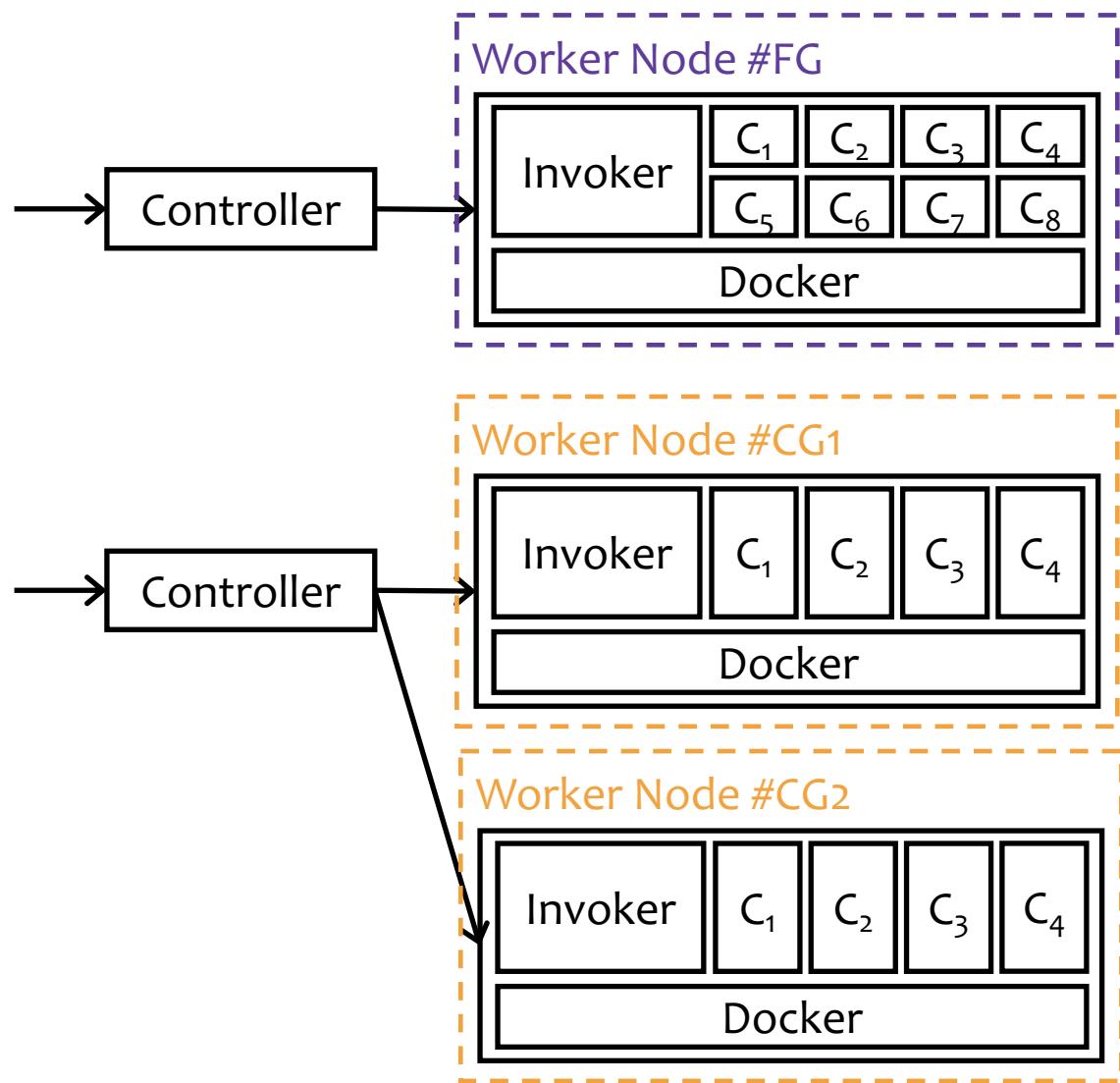
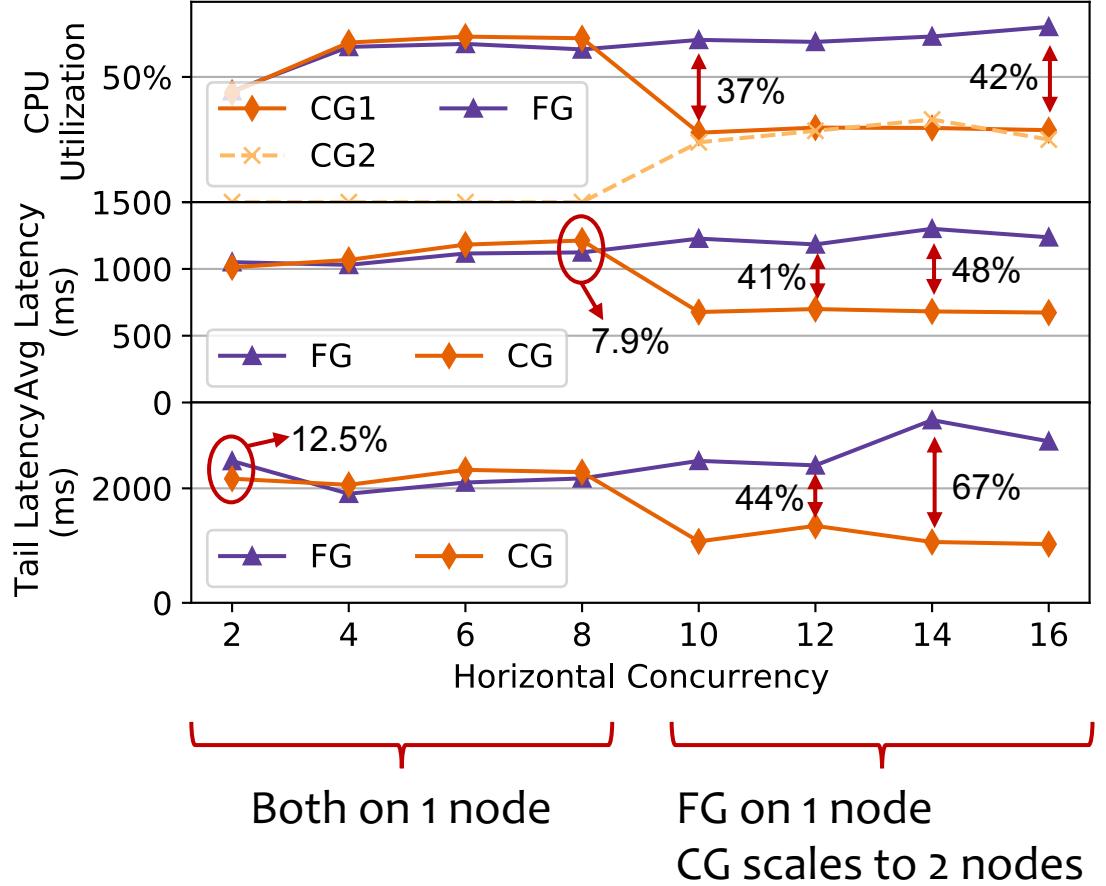
- Measurements from the execution of 2 widely used FaaS benchmark suites
 - ServerlessBench, FaaS-Profiler
- Benchmarks running on an open-sourced FaaS platform -- OpenWhisk
- Deployed on IBM Cloud with 1 master node and 4 worker nodes



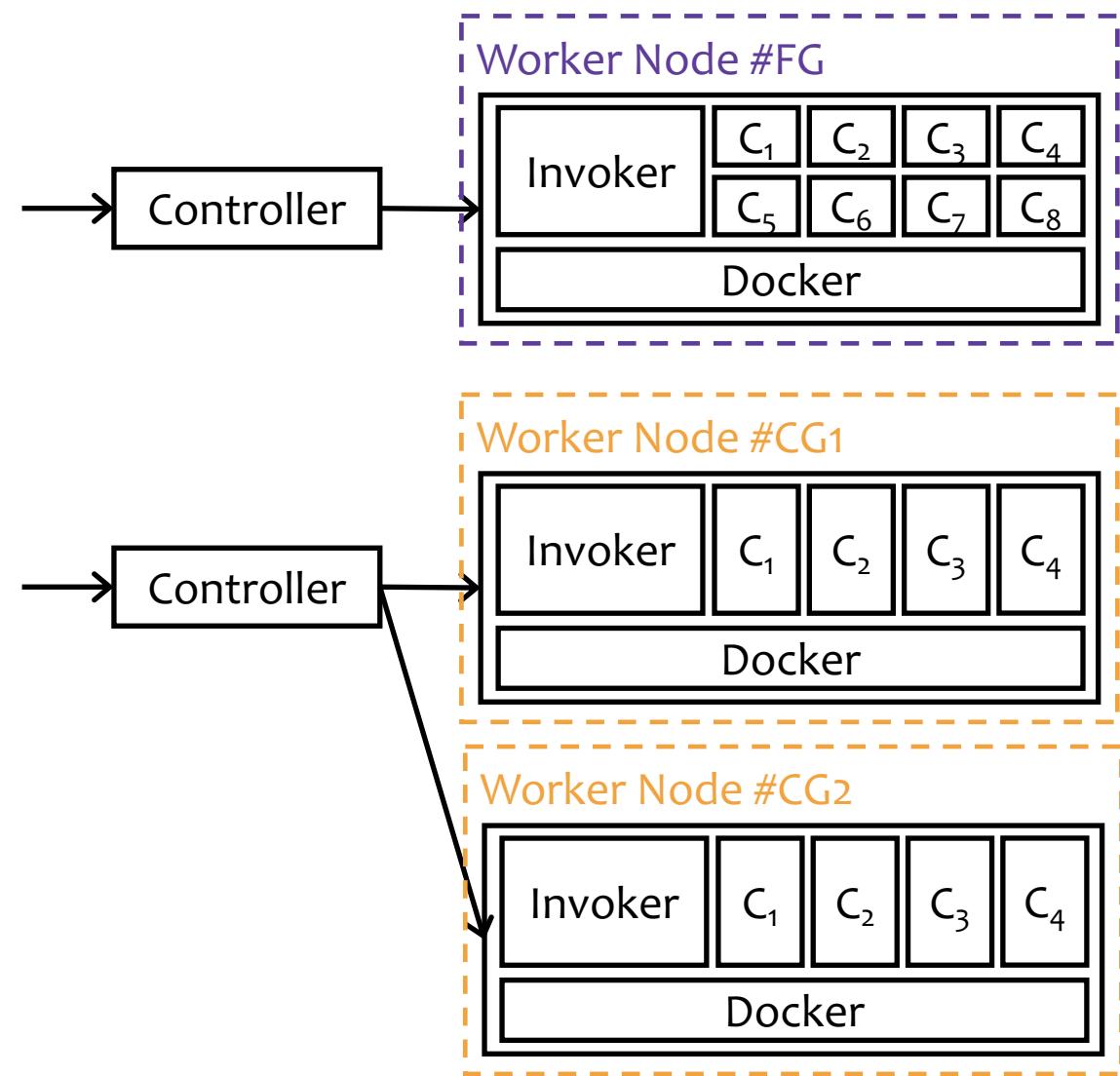
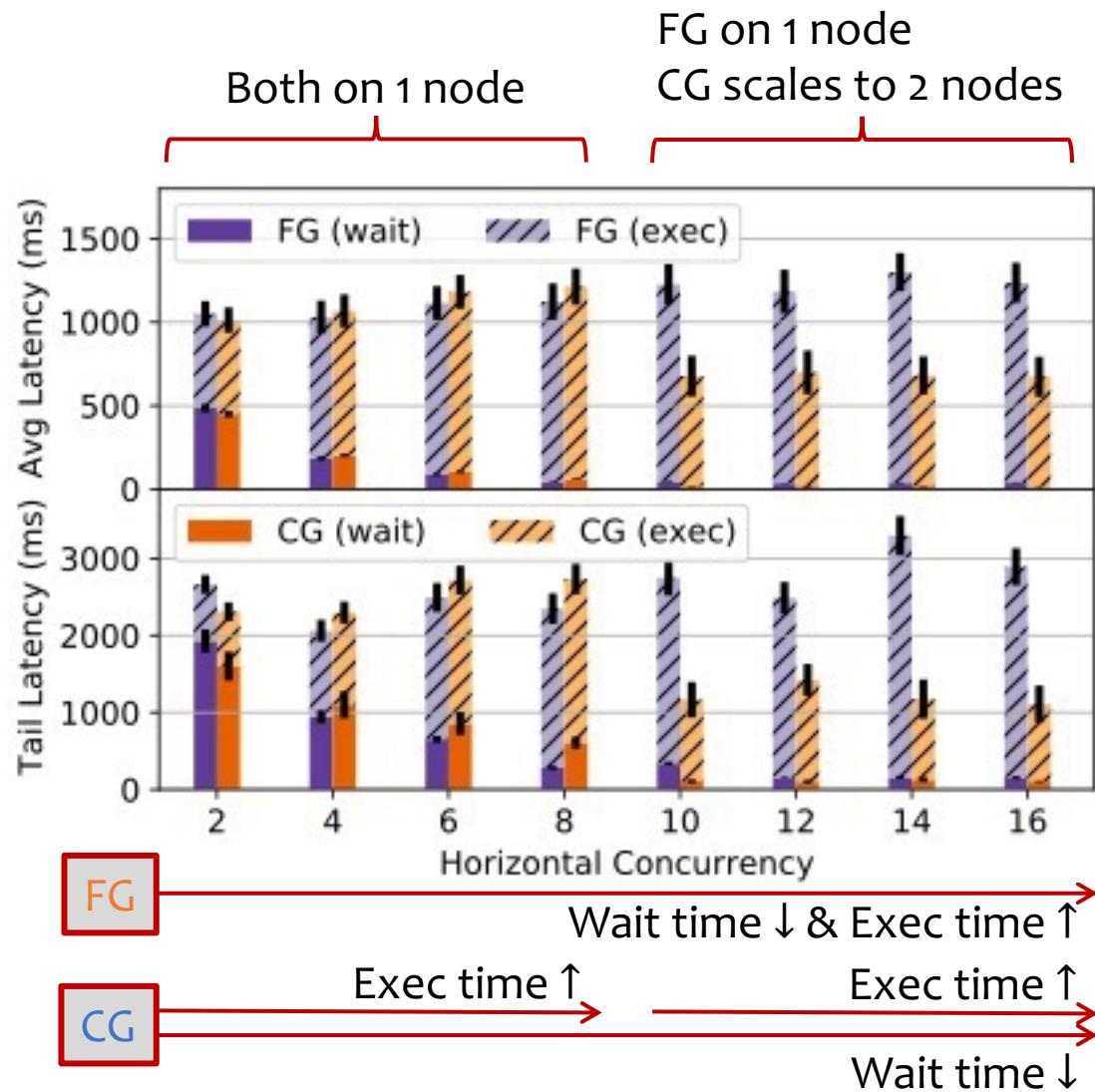
Concept Overview



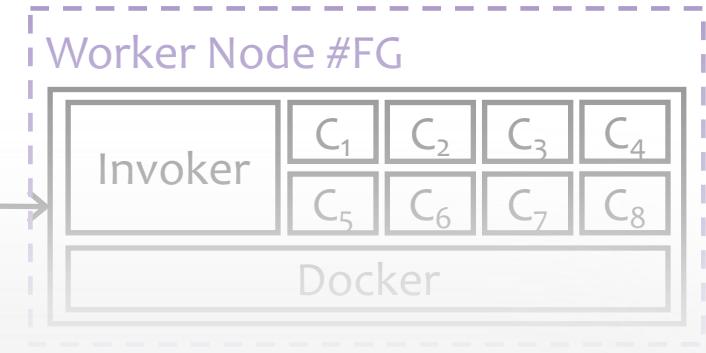
Latency Variation



Latency Variation

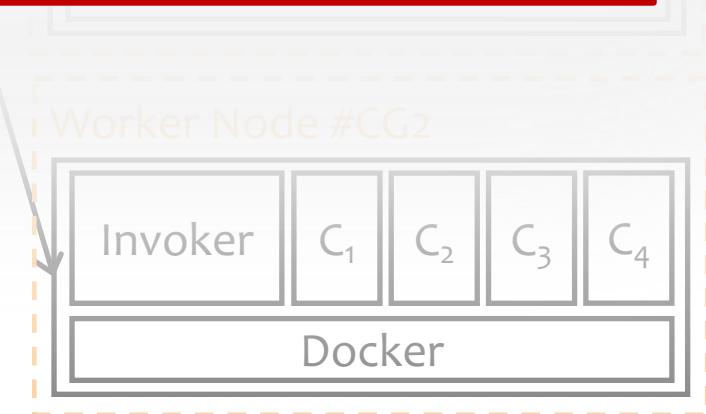
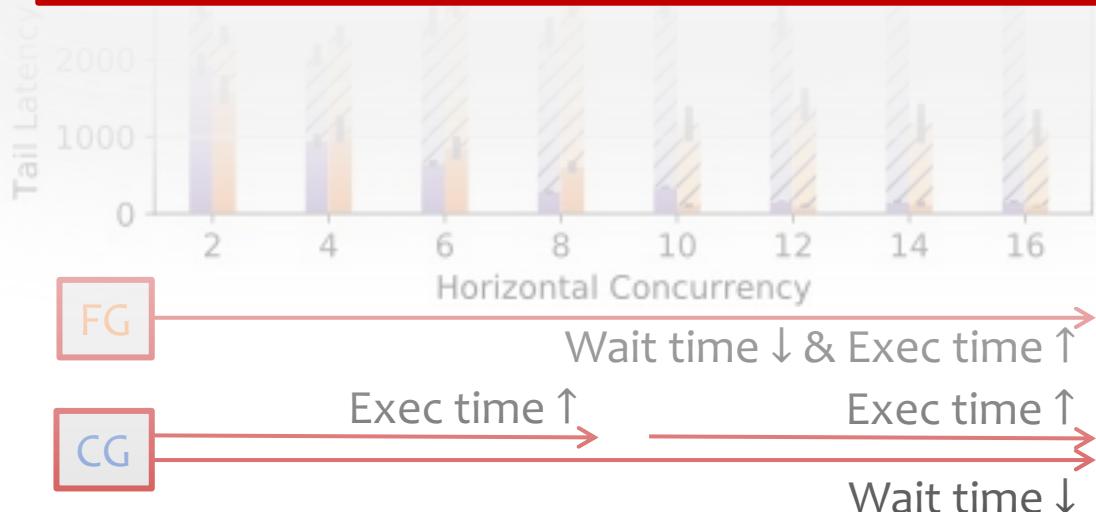


Latency Variation

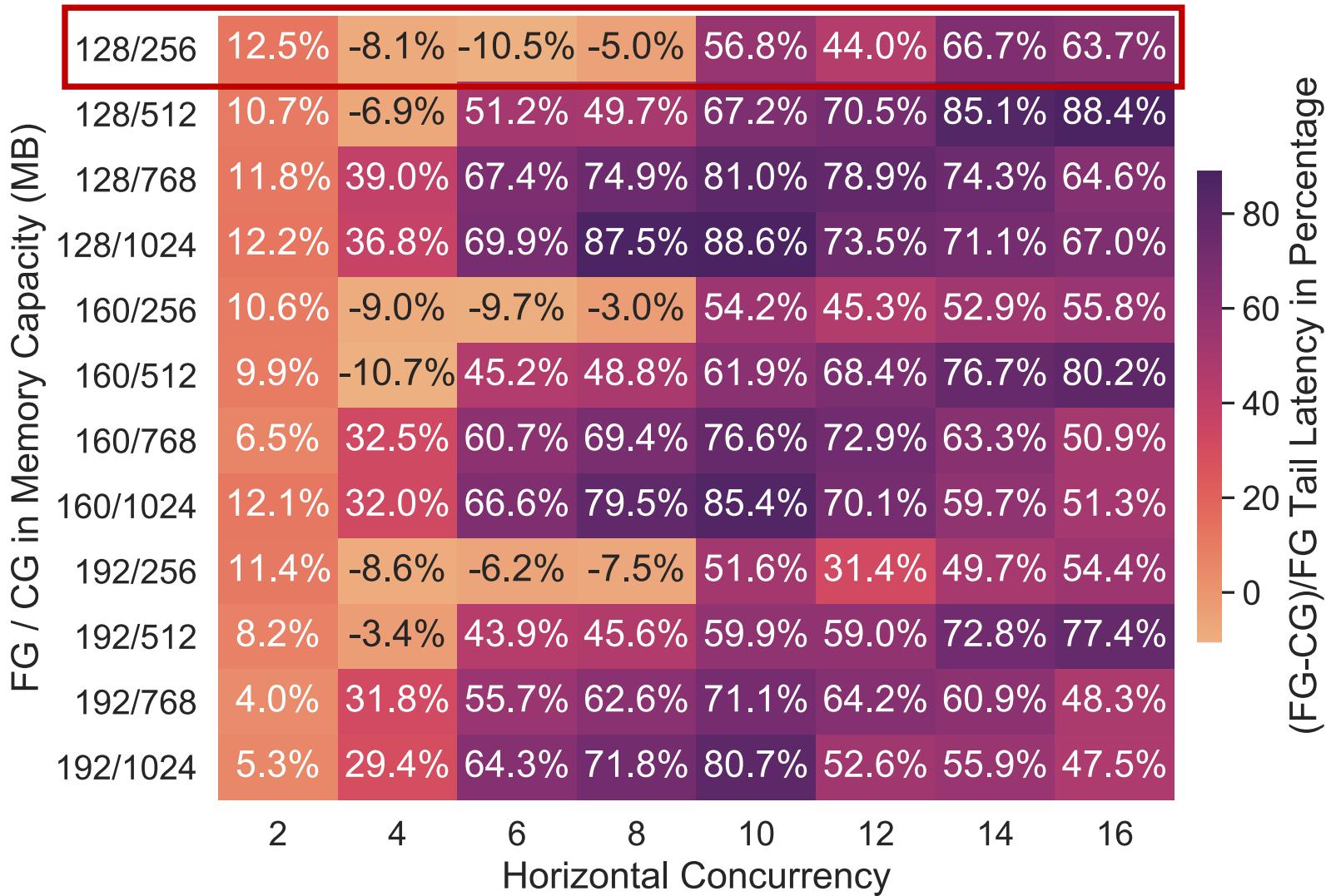


Invoker → Docker → Container

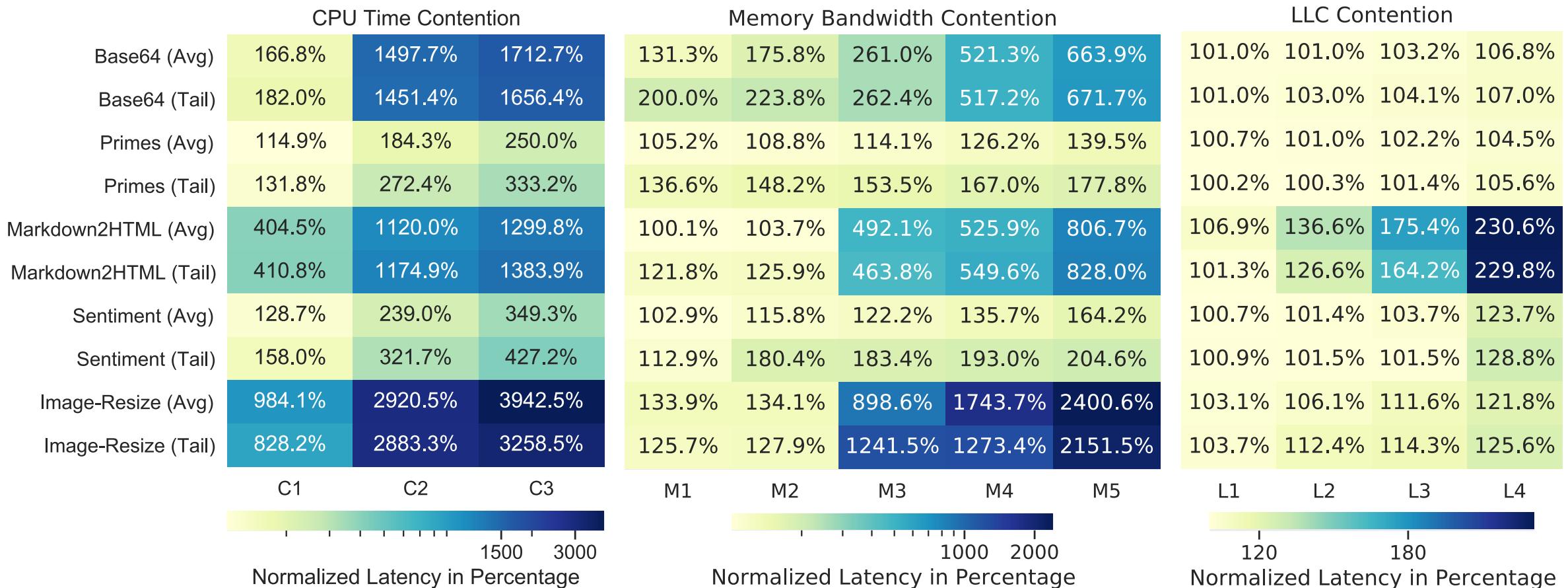
[Implication] Compared to FG policies, a CG policy scales out containers on a greater number of servers, resulting in less resource contention and thus up to 67% lower end-to-end latency.



Latency-Utilization-Power Trade-off



Performance Interference

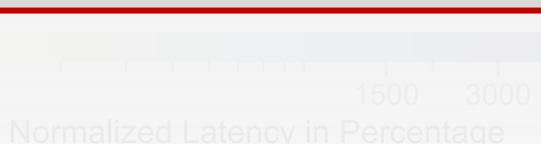


Performance Interference

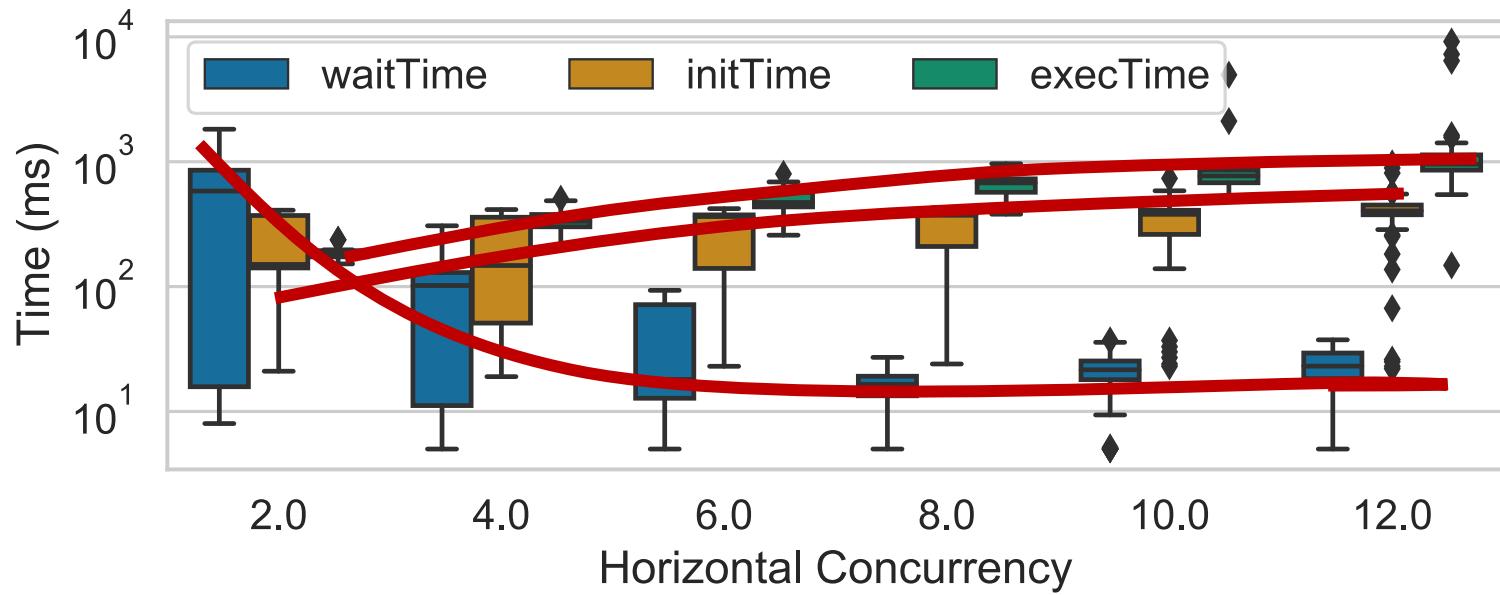
	CPU Time Contention				Memory Bandwidth Contention				LLC Contention				
Base64 (Avg)	166.8%	1497.7%	1712.7%		131.3%	175.8%	261.0%	521.3%	663.9%	101.0%	101.0%	103.2%	106.8%
Base64 (Tail)	182.0%	1451.4%	1656.4%		200.0%	223.8%	262.4%	517.2%	671.7%	101.0%	103.0%	104.1%	107.0%
Padam (Avg)	114.0%	194.2%	250.0%		105.3%	169.8%	114.1%	136.2%	120.5%	100.7%	101.0%	102.2%	104.5%
Padam (Tail)	114.0%	194.2%	250.0%		105.3%	169.8%	114.1%	136.2%	120.5%	100.7%	101.0%	102.2%	104.5%

[Implication]

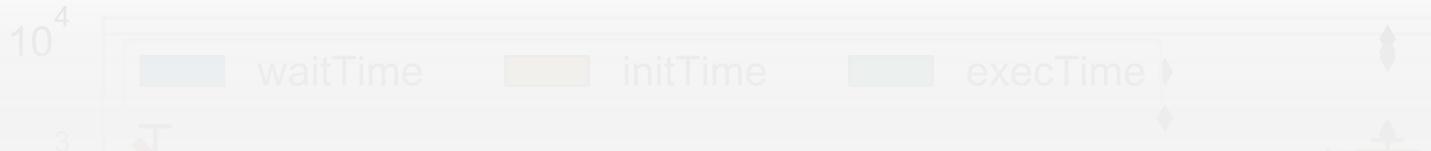
- Performance isolation should be carefully assessed to prevent SLO violations due to resource sharing.
- However, when thousands of function containers are consolidated on a single server, state-of-the-art resource partitioning fails to mitigate the performance interference, still with up to **8.3×, 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