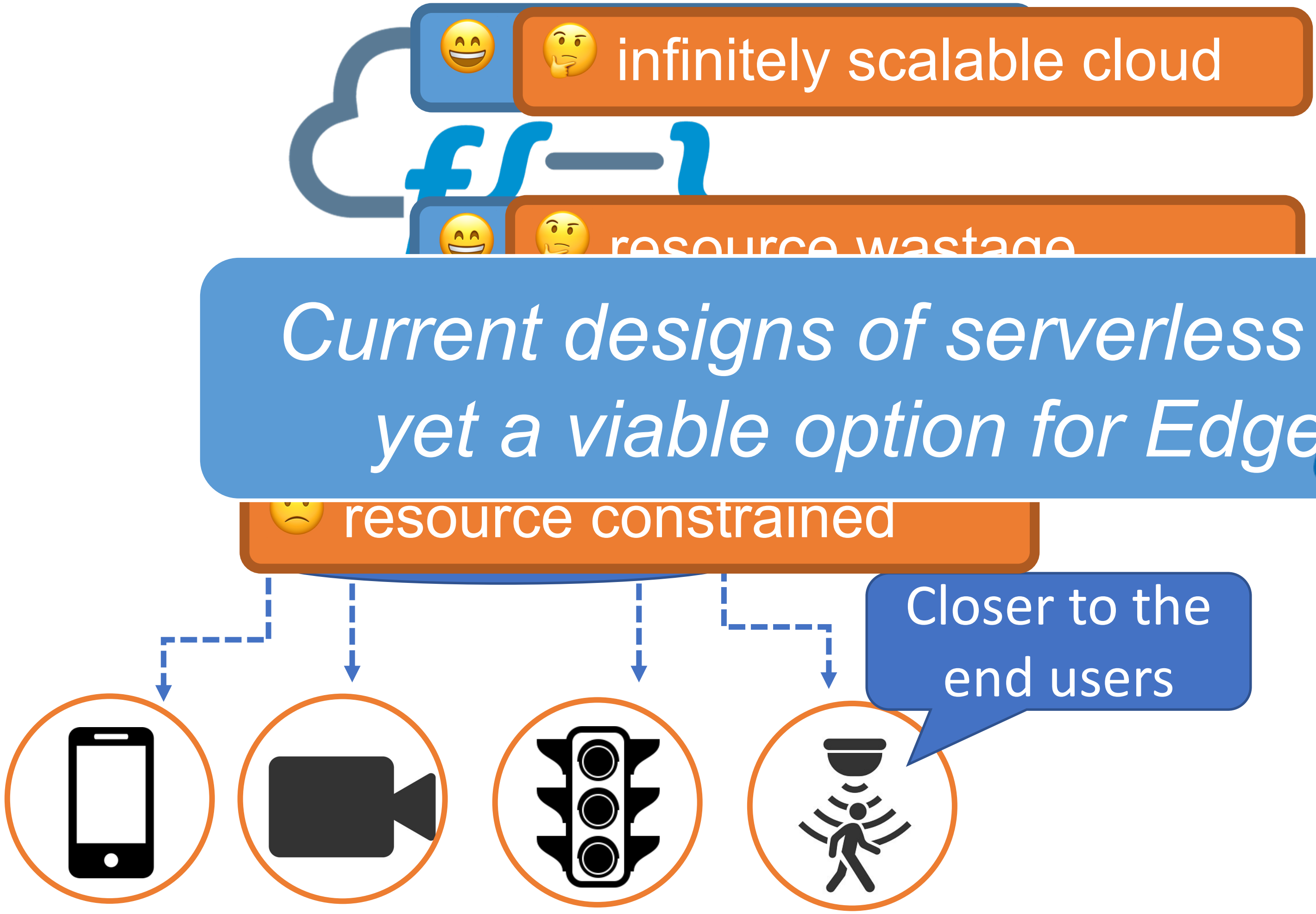
Mu: An Efficient, Fair and Responsive Serverless Framework for Resource-Constrained Edge Clouds

Viyom Mittal[*], **Shixiong Qi**[*], Ratnadeep Bhattacharya[+], Xiaosu Lyu[+], Junfeng Li[§], Sameer G Kulkarni[†], Dan Li[§], Jinho Hwang[★], K. K. Ramakrishnan[*], Timothy Wood[+]

[*]University of California, Riverside, [+]George Washington University, [§]Tsinghua University, [†]Indian Institute of Technology, Gandhinagar, [★]Facebook Inc.

November 1st, 2021

Serverless Cloud



Current designs of serverless p
yet a viable option for Edge

Challenges to Using Existing Approaches in Edge Clouds

Challenge 1: Imprecise Resource Provision

Existing autoscaling design depends on user inputs

- 🙁 Users are unaware of runti e feat s of functions
- 🙁 Inappropriate para ter c s

Missed SLO

Single metric-based au y)

- 🙁 Insufficient to achieve optimal autoscaling

Slow resource provision in case of traffic bursts

- 🙁 Long response time, SLO violations

Challenges to Using Existing Approa… es in Edge Clouds

Challenge 2: Unfair Functio… ement

Existing placem… efficiency

- 🙁 With no con… ns
- 🙁 Unfair re…

🤔 **If two functions that … ve the same SLO are going to be placed…**

Function-1: More resource provisioned and Better SLO

Function-2: Less resource provisioned and Worse SLO

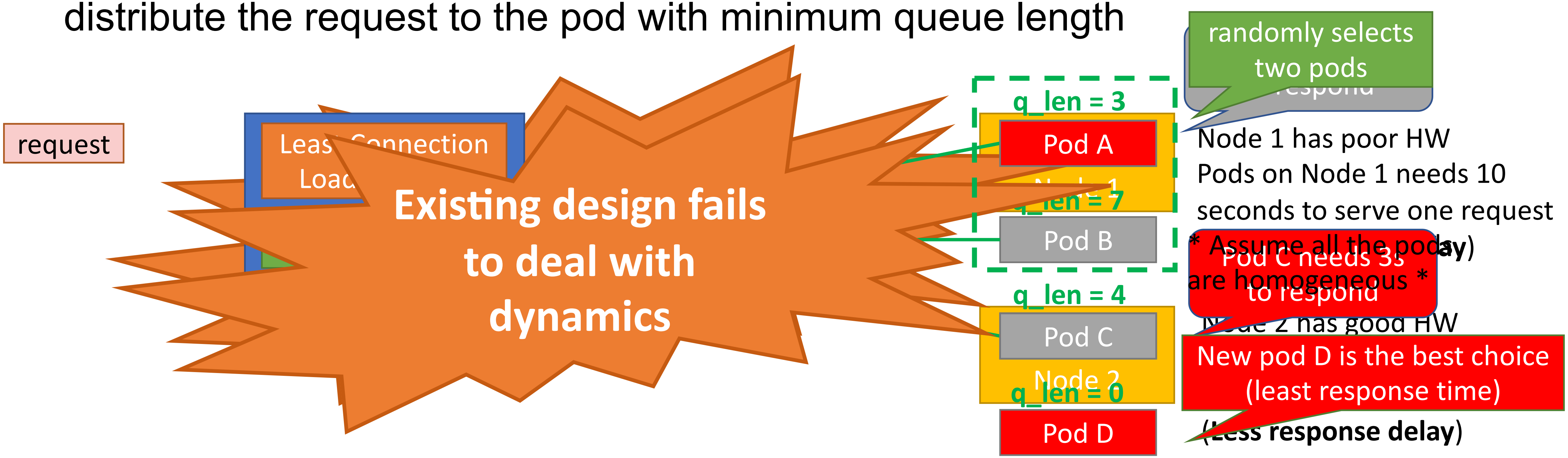
UC RIVERSIDE

Challenges to Using Existing Approaches in Edge Clouds

Challenge 3: Unawareness of Resource Heterogeneity and System Dynamics

🙁 **Resource Heterogeneity and System Dynamics can lead to poor load balancing decision**

Least connection LB: Track the queue length at backend pods and distribute the request to the pod with minimum queue length



Challenges to Using Existing Approaches in Edge Clouds

Challenge 4: Approximate metrics collection

Serverless platform relies on pod metrics to guide resource management

Existing design relies on approximate metrics collection to address scaling

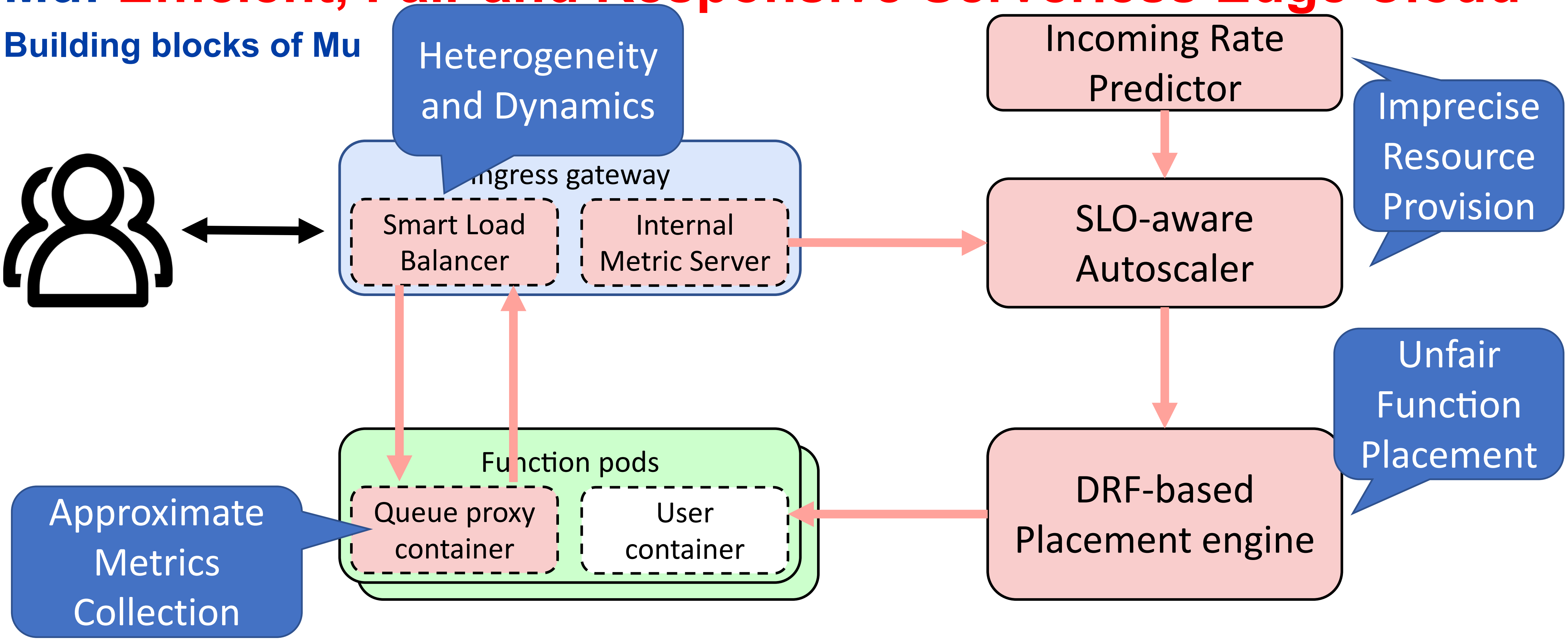
- 🙁 an inaccurate view of system status
- 🙁 negative impact on resource provision, load balancing...

Need a precise, lightweight and scalable metric collection mechanism

UC RIVERSIDE

Mu: Efficient, Fair and Responsive Serverless Edge Cloud

Building blocks of Mu



SLO-aware Autoscaler

Idea:

😄 From user's perspective, providing SLO is more meaningful rather than inte[…] RPS)

✓ Single metric-based autoscaling (RPS, Concurrency)

✓ Existing autoscaling design depends on user inputs

How:

1️⃣ […]sers only provide the target SLO of their function

2️⃣ Provision resources by factoring in both the incoming request rate and the queue length

3️⃣ Ensure SLOs by factoring in the average request execution time

✓ Avoid over-allocation of resources to ensure performance with just the right amount of resources.

Incoming Rate Predictor

Our Goal:

🤔 Provision adequate resource in case of traffic bursts

Idea:

😄 Mu uses a simple online linear regression model to predict the incoming rate based on previous observations

😄 Mu uses multi-armed bandit to improve accuracy

Features:

1. lightweight and fast
2. Accurate prediction - dynamically select the model with minimum error

Placement Engine

Consider fairness and efficiency together

2-stage heuristic algorithm

- ***Resource fairness*** between serverless functions
  - Function selection based on Dominant Resource Fairness (DRF)
- ***Resource efficiency*** between nodes
  - Node selection based on scoring
    - Alignment [1], WorstFit [2], and BestFit [2].
    - reduce the resource fragmentation, minimize unfairness

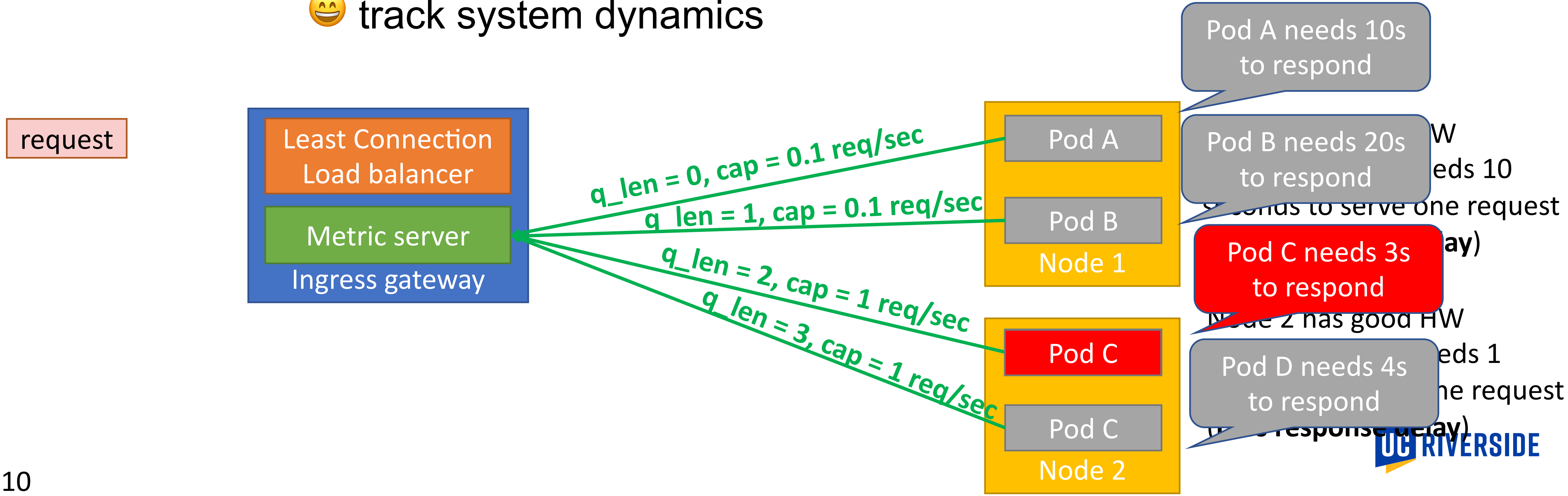
*Call two stages iteratively until there are no resources left or all functions are placed

1. R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella. 2014. Multi-resource packing for cluster schedulers. ACM SIGCOMM Computer Communication Review 44, 4 (2014), 455–466.
2. C. A. Psomasand, J. Schwartz. Beyond beyond dominant resource fairness: Indivisible resource allocation in clusters. Tech Report Berkeley, Tech. Rep. (2013).

Load Balancer

Our Goal: 🤔 to be aware of resource heterogeneity and system dynamics

- 😄 use extra metrics to estimate response time of each pod
  - 😄 differentiate “fast” and “slow” pods in the system
- 😄 use “piggybacked” metrics of each pods instead of “two random choices”
  - 😄 track system dynamics



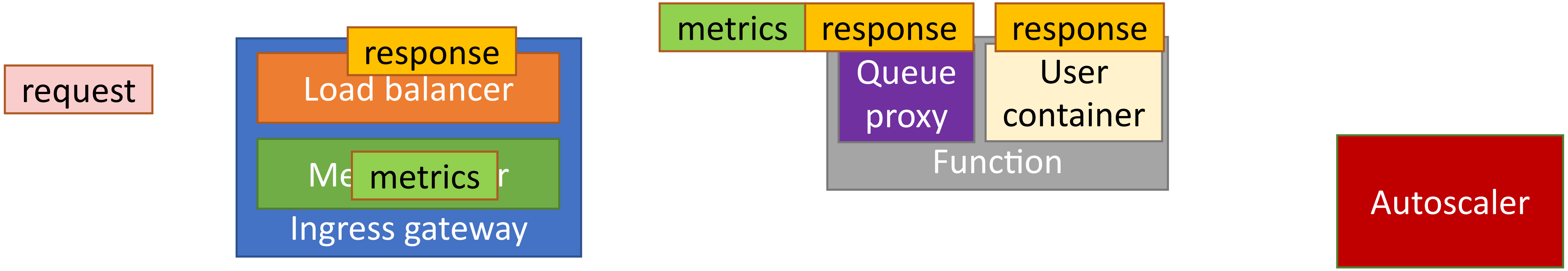
Metric collection

Our goal:

- Precise metric collection that can reflect latest system state and achieve good scalability

Precise | Dynamics | Heterogeneity | Scalable

- The re ponse is instantaneo providing t metrics based for the latest s tem state
- Piggybacking also help us accommodate heterogeneity and system dynamics
- Piggybacking eliminates the explicit need of metric collection



UC RIVERSIDE

Summary of overall evaluation

Experiment setup

- Three different system configurations

Mu

Default Knative with RPS autoscaling (RPS)

Default Knative with Concurrency autoscaling (CC)

  - Scaling target is fair among different scaling policies
  - System is slightly overloaded

- Evaluated metrics
  - Latency and Fairness
  - Pod allocation (efficiency)
  - SLO performance

Azure workloads

Parameter/Specification		Values
Invocation Range	W-1	41-230 rps
	W-2	69-182 rps
Average invocations	W-1	154 rps
	W-2	146 rps
Container Concurrency		4
Grace Flag (Mu only)		16
Execution time		500ms
Maximum pod capacity		48
CPU and Mem. per pod		7 cores, 30GB
Target	RPS	8
	CC	40
SLO		5 seconds

UC RIVERSIDE

Latency and Fairness

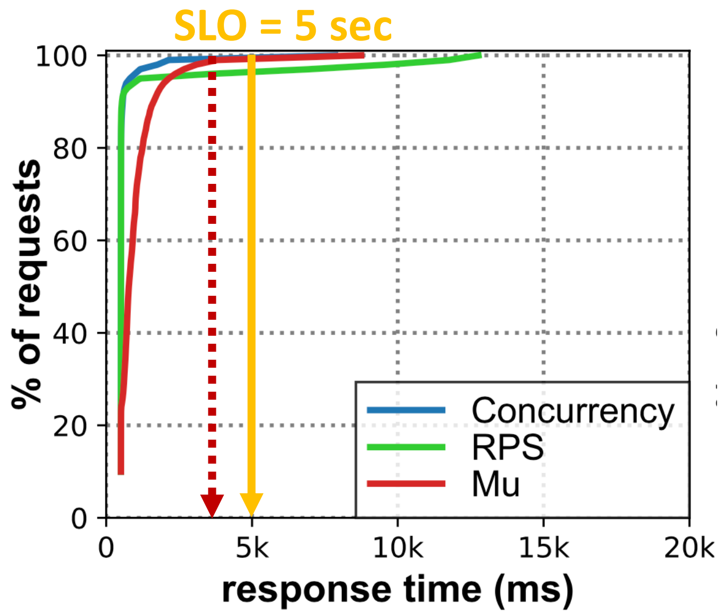
Response time CDF for Mu and standard Knative approaches (CC and RPS)

Mu has good control over response time

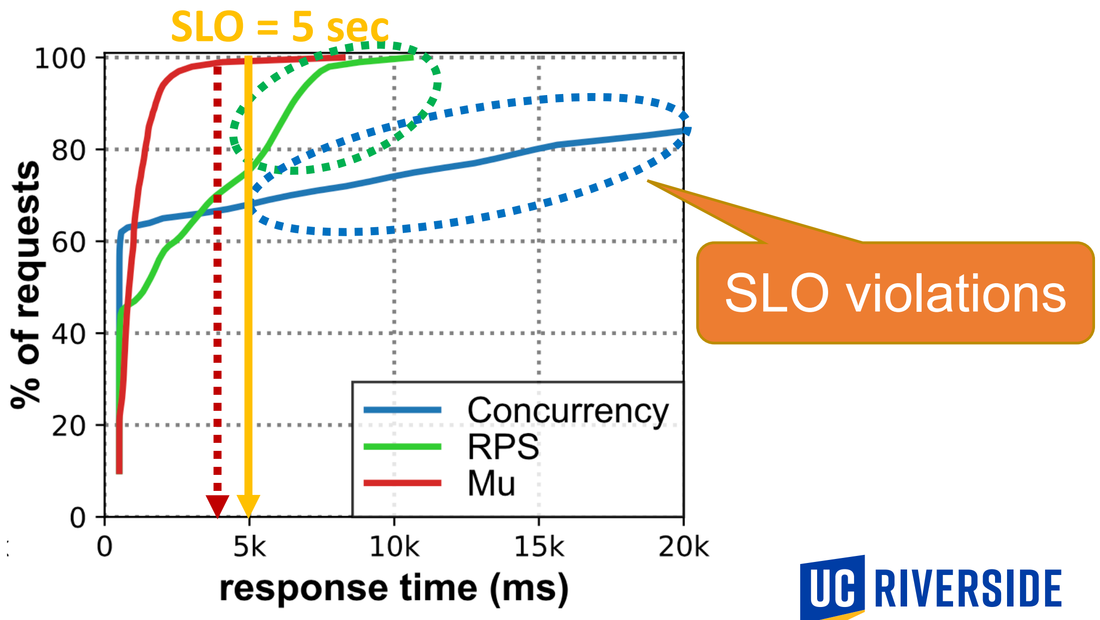
- Mu limits the tail latency with SLO of 5 seconds **for both workloads**
- Standard Knative approaches result in much larger response time tail

Mu achieves better fairness

- Mu treats **Workload-1** and **Workload-2** equally
- Standard Knative approaches unfairly treat **Workload-2**



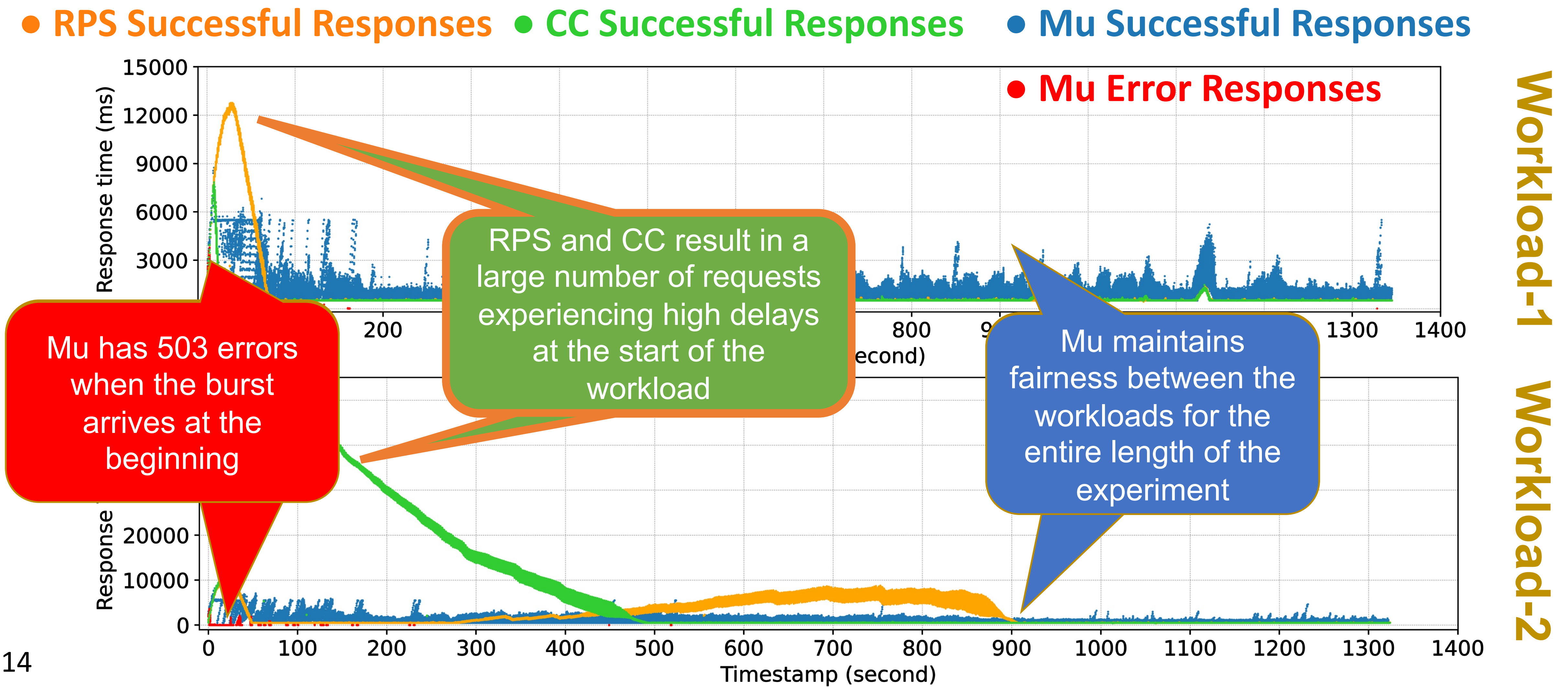
Workload-1



Workload-2

Latency and Fairness

Time series of Response Time for Mu and standard Knative approaches

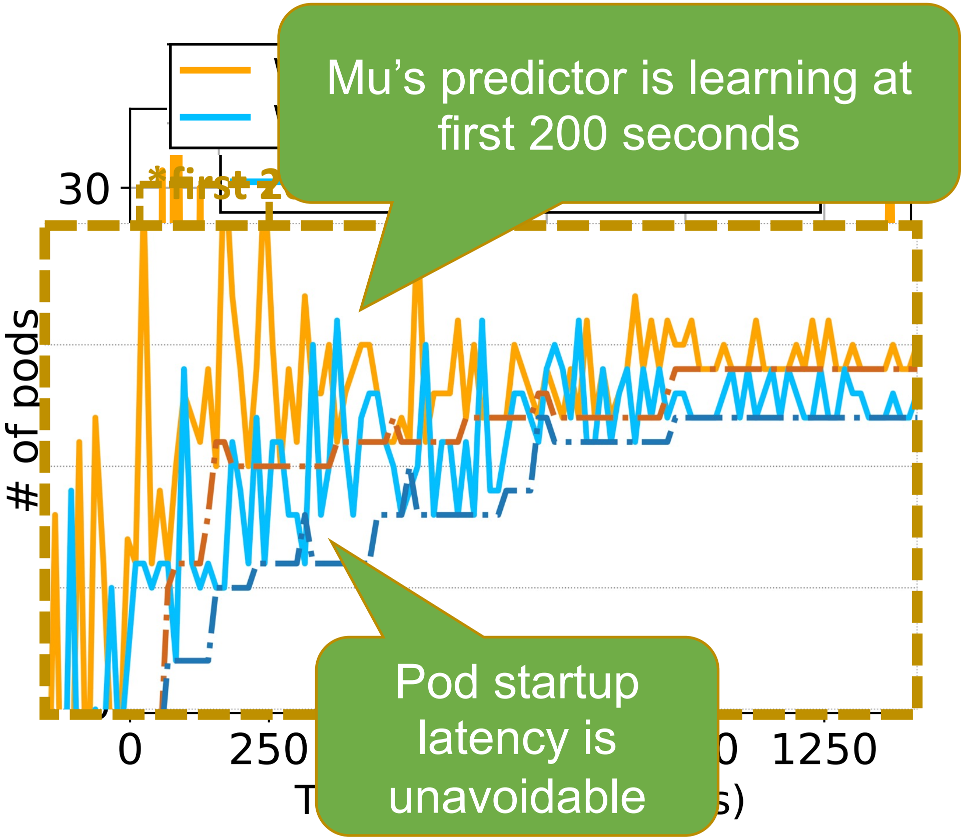


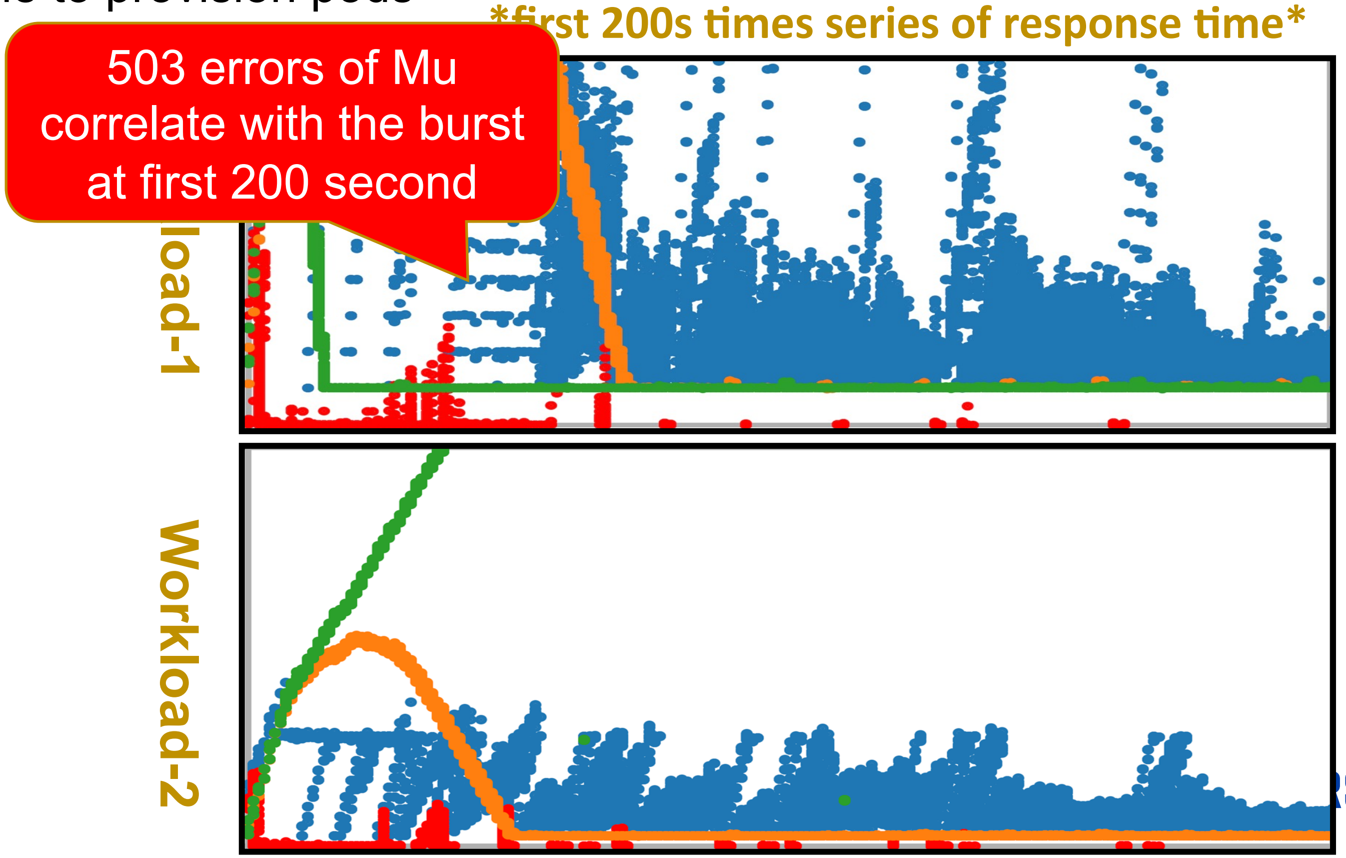
SLO Performance

Mu: Correlation between 503 errors and occurrence of bursts

Most of the 503 errors occur when the burst arrives at the beginning (see first 200 seconds)

- When the predictor has not yet learned the characteristics of the workload
- The system takes a certain amount of time to provision pods





SLO Performance Summary

Mu achieves better SLO than standard Knative approaches

- 96.8% requests served within SLO compared to RPS (86.2%) and CC (84.2%)

Mu uses SLO-aware admission control and returns 503 errors

- This avoids the build up of a large queue with the arrival of a ***burst*** of requests
- RPS and CC choose to buffer the burst of requests -> SLO violation

503 errors result in limited negative impact and bring more SLO benefit compared to a large queue

- Better SLO performance
- Low tail latency

503 errors returned by Mu impacts relatively small number **(<5%)** of requests,

CC and RPS build up a large queue resulting in very long latencies

25-30% of total requests failed to meet SLO

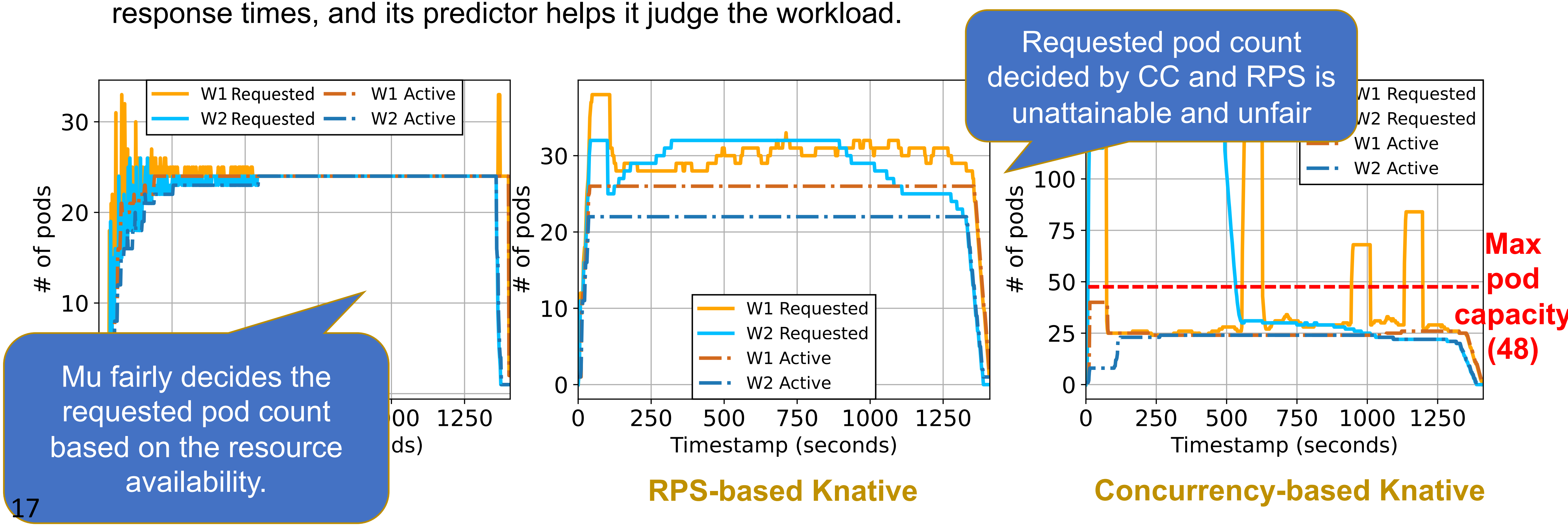
		[illegible] (ms)	503 error / total requests	Requests served within SLO
[illegible]	Workload-1	3805	[illegible]	[illegible]6.5%
	[illegible]	[illegible]073	[illegible]	[illegible]7.0%
RPS	[illegible]	[illegible]757	[illegible]	[illegible]6.4%
	Workload-2	8808	0 / 209905	75.5%
Concurrency	Workload-1	2141	0 / 220126	99.6%
	Workload-2	49526	0 / 209905	68.0%

Pod Allocation

Time series of Pod counts for Mu and standard Knative approaches

Mu uses less pods than standard Knative approaches

- On average RPS and Concurrency use 18% and 17% more pods than Mu
- Mu tends to request fewer pods since its goal is to meet SLOs, not necessarily minimize response times, and its predictor helps it judge the workload.



RPS-based Knative

Concurrency-based Knative

Conclusion

Mu achieves better

- SLO-aware Autoscaler / Incoming rate predictor
- DRF-based placement engine
- Smart load balancer
- Piggybacked metrics

01 latency performance

02 resource fairness

03 resource efficiency

04 SLO performance

compared to Knative

An Efficient, Fair and Responsive Serverless Framework for Resource-Constrained Edge Clouds

UC RIVERSIDE