Practical Efficient Microservice Autoscaling with QoS Assurance (PEMA)

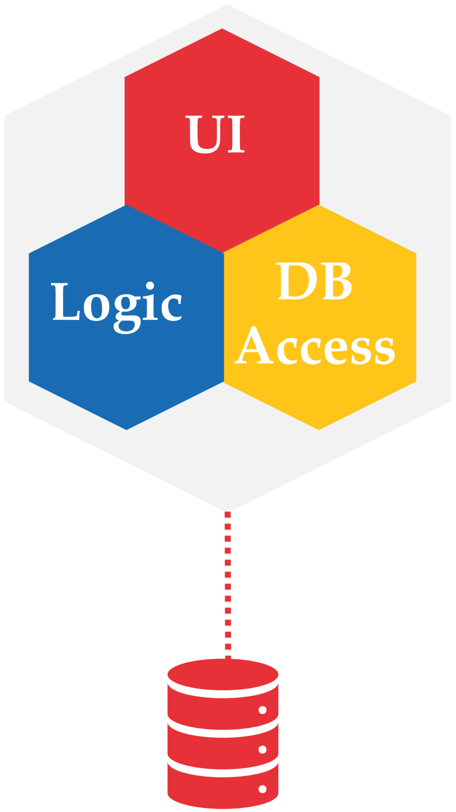
Md Rajib Hossen[1], **Mohammad A Islam[1]**, Kishwar Ahmed[2]
[1]The University of Texas at Arlington, [2]University of South Carolina, Beaufort

THE UNIVERSITY OF TEXAS AT ARLINGTON

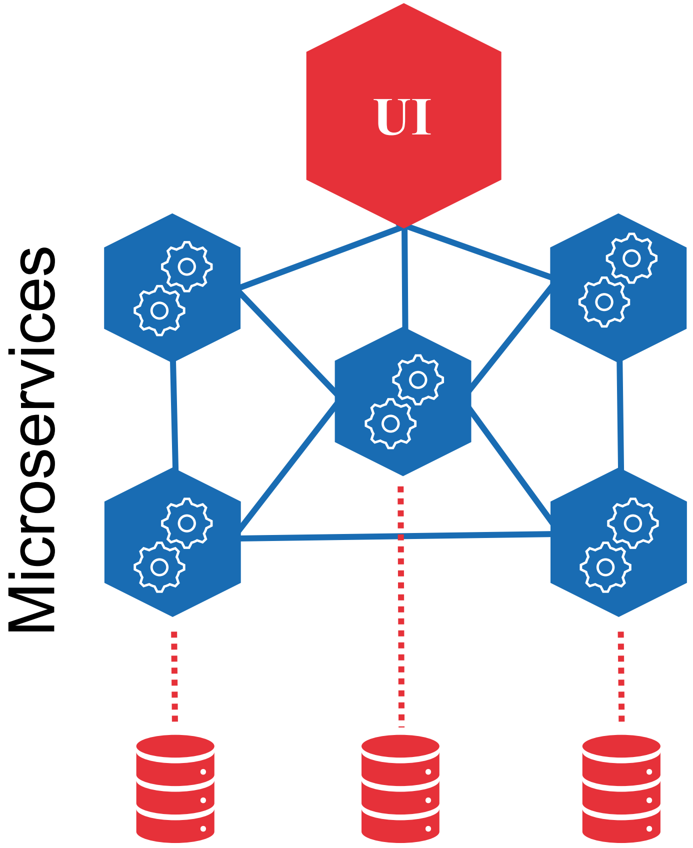
UNIVERSITY OF SOUTH CAROLINA BEAUFORT

Acknowledgement: This work is supported in parts by the NSF under grant number CNS-2104925.

Monolithic vs. Microservice



Monolithic
Architecture

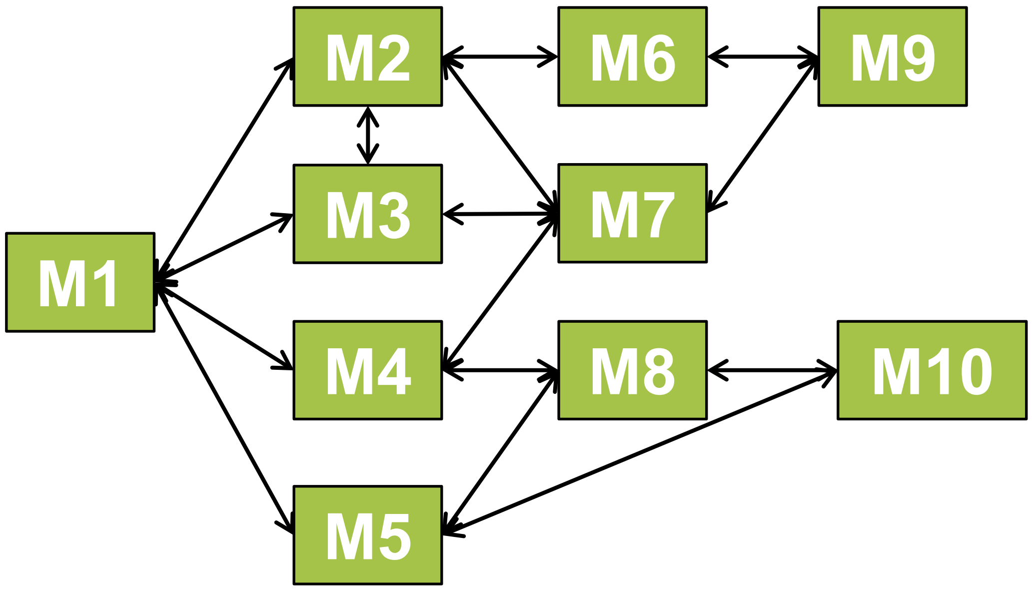


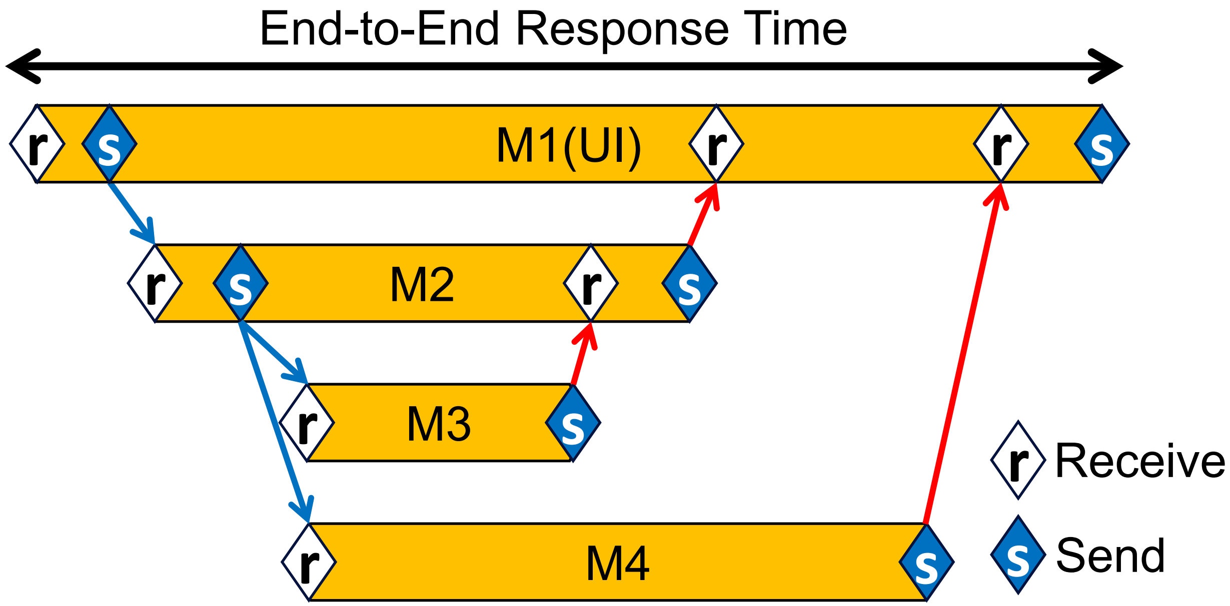
Microservices
Architecture

Advantages of Microservices
- ➢ Easier DevOps management
- ➢ Lightweight
- ➢ Agile resource management
- ➢ Better scaling
- ➢ Fault-tolerance
- ➢ Platform agnostic compatibility

Challenges in Microservice Management

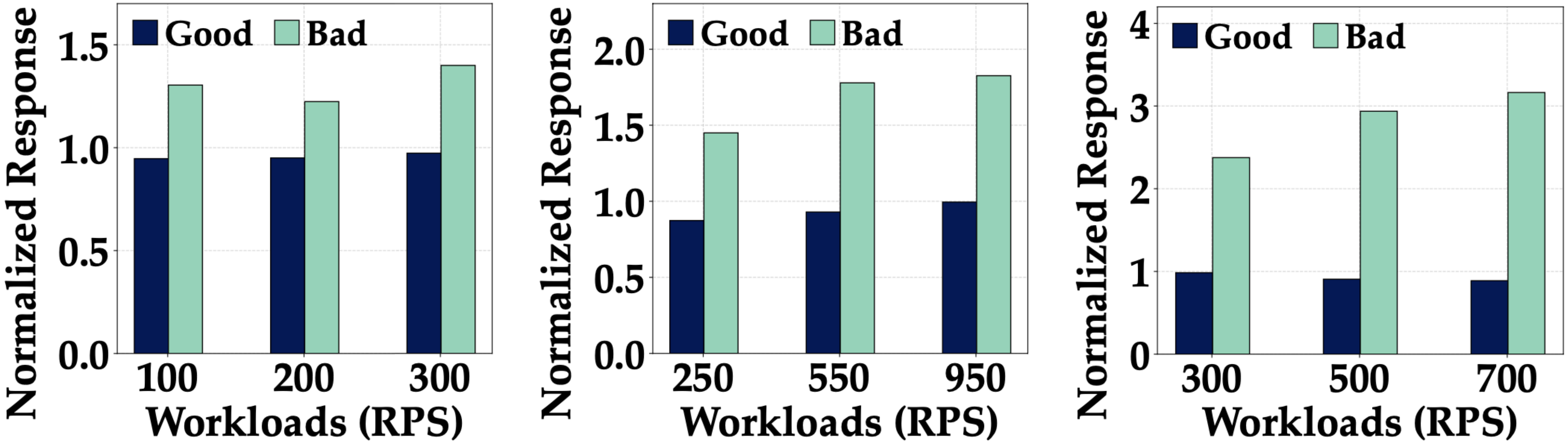
- Large configuration space.
- Complex communication and inter-dependency
- End-to-end response time depends on multiple services





Resource Distribution is Critical

- End-to-end response times for the same total resource varies significantly depending on the resource distribution among micoservices

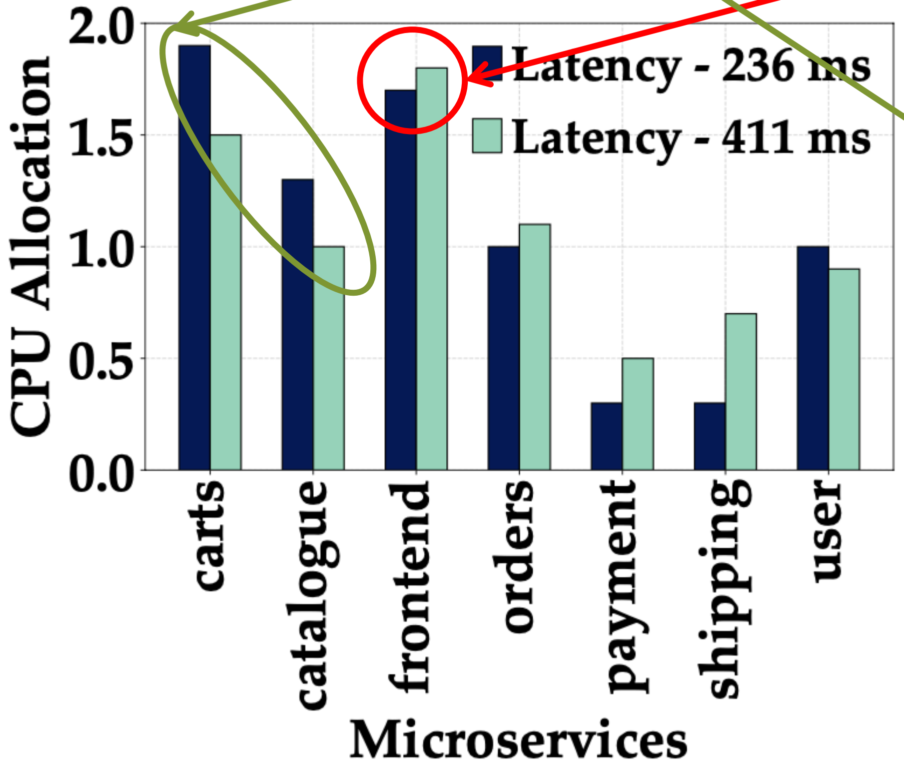


(a) TrainTicket (b) SockShop (c) HotelReservation

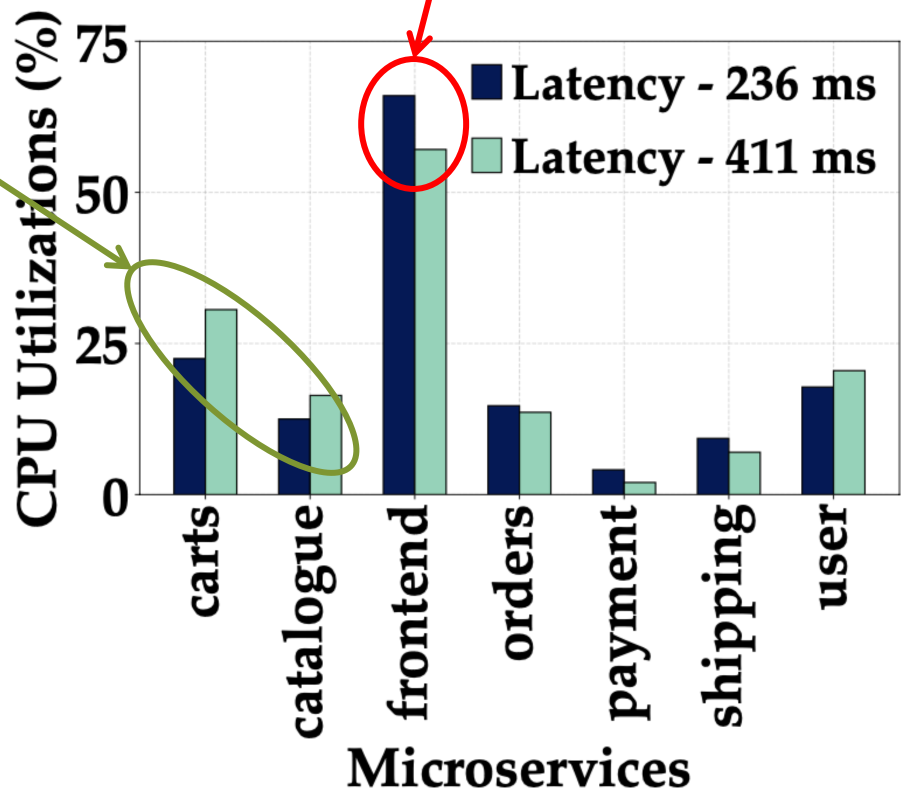
"Good" Resource Distribution is Hard to Identify

Impact of resource change on utilization varies

Resource Increased for highly utilized service



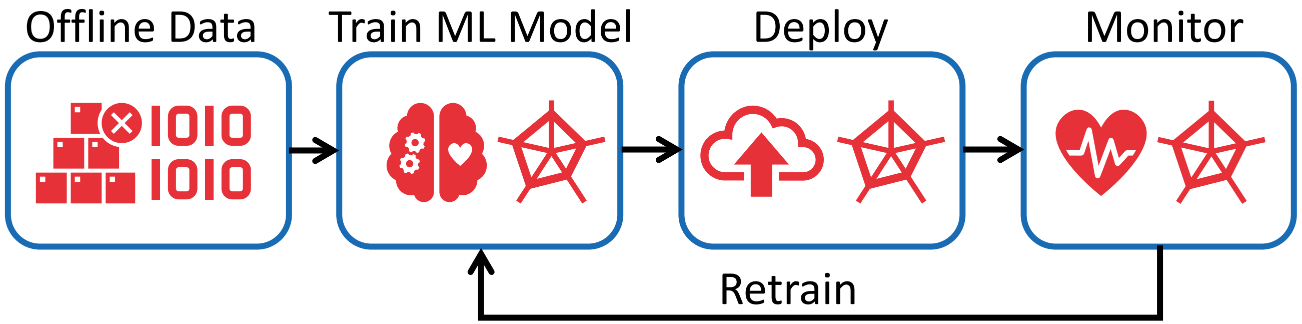
(a) CPU allocation



(b) Utilization

Limitations of Existing Approaches

- Existing cloud managers cannot capture microservice dynamics
- Machine Learning-Based approaches[1,2,3]
  - High-resolution data for offline training
  - Intentional SLO violation
  - Workload change Requires retraining for system changes



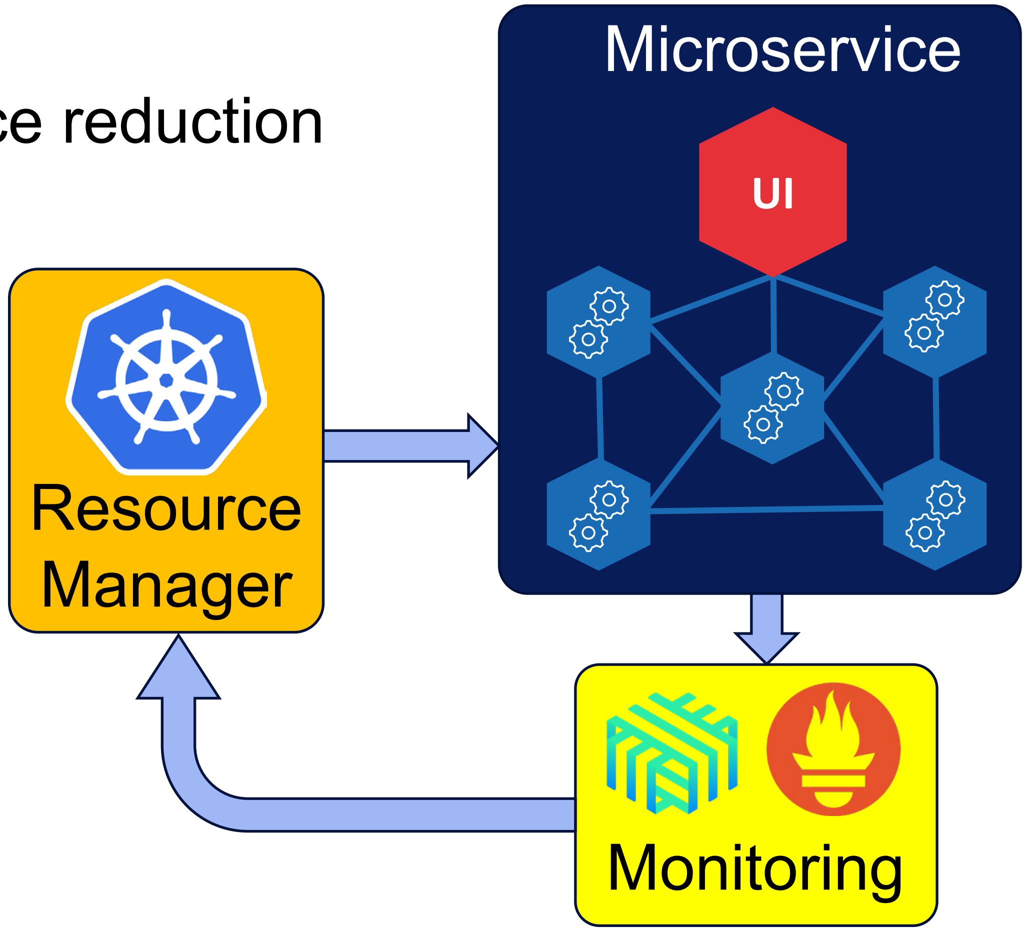
1. Qiu, Haoran, et al. "FIRM: An Intelligent Fine-grained Resource Management Framework for SLO-Oriented Microservices." *OSDI,* 2020.
2. Zhang, Yanqi, et al. "Sinan: ML-based and QoS-aware resource management for cloud microservices." *ASPLOS,* 2021.
3. Hou, Xiaofeng, et al. "AlphaR: learning-powered resource management for irregular, dynamic microservice graph." *IPDPS*, 2021.

Practical **E**fficient **M**icroservice **A**utoscaling (PEMA)

- Online and not data intensive
- No intentional QoS violation
- Adaptive to changes

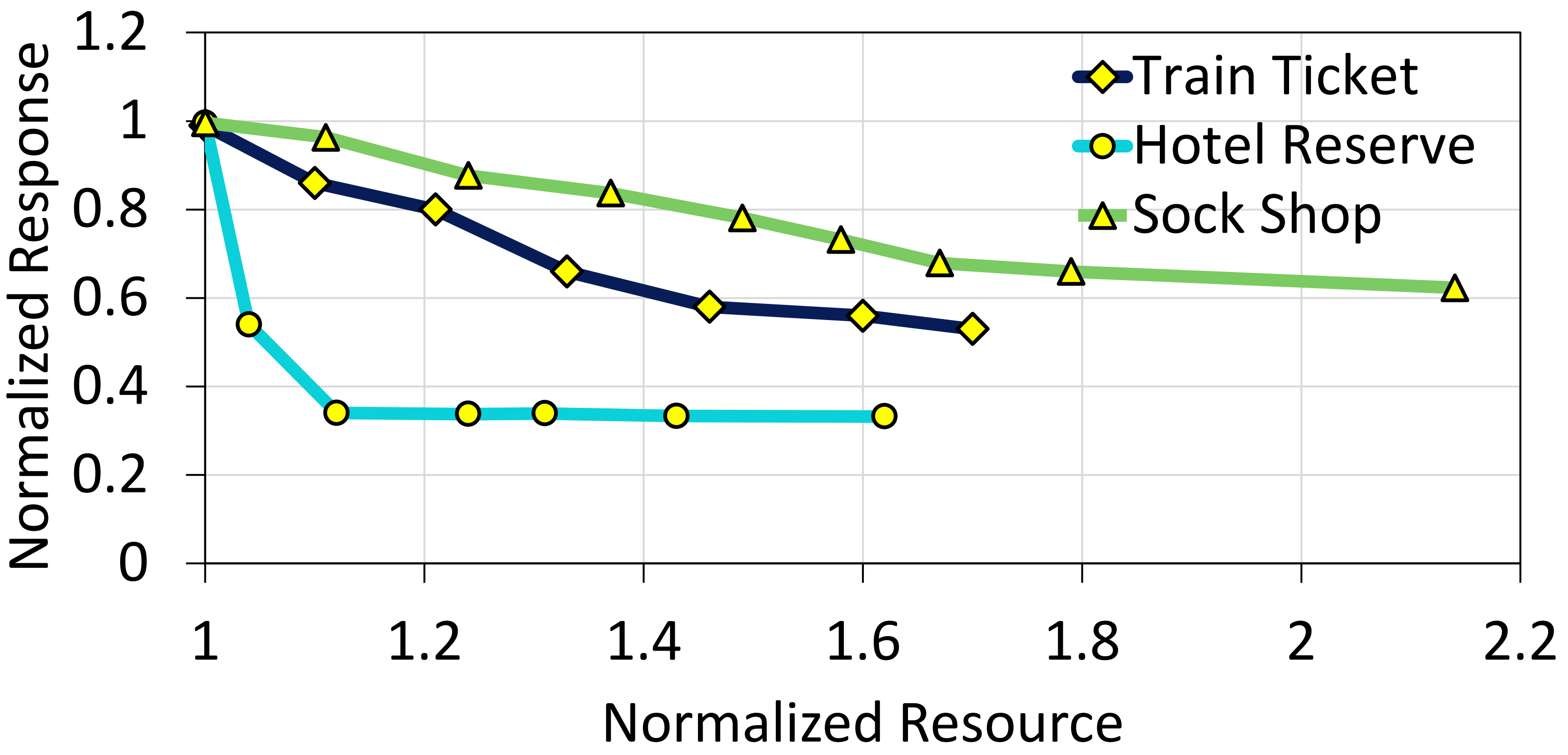
Feedback-Based Navigation

- Start with “enough” resource to satisfy SLO
- Opportunistic Resource Reduction
  - If response time < SLO → resource reduction
- Ensures no QoS violation
- Online and not data intensive



Feedback-Based Navigation

- Gradually reduce resource to push the response time close to SLO
- Monotonic resource reduction



Resource Reduction

- How many microservices to reduce the resources from?

$$n^t = \mathrm{N} \cdot \frac{R - r^{t-1}}{\alpha R}$$

Number of Microservices for to cut resource

Distance from SLO

Total Number of Microservice

Scaling

Resource Reduction

- How much to reduce?

$$\Delta^t = \beta \cdot \frac{R - r^{t-1}}{\alpha R}$$

Resource Reduction

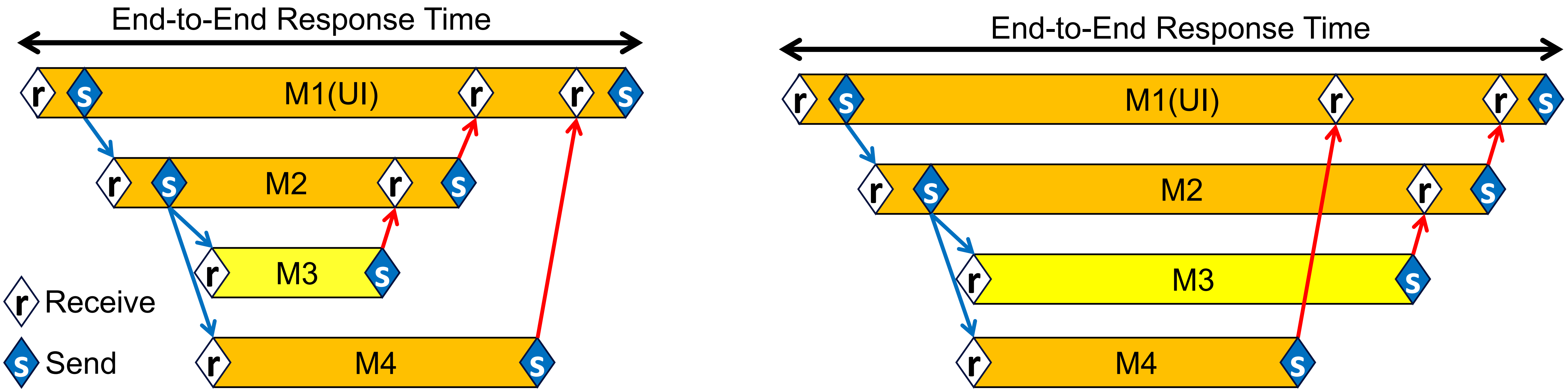
Distance from SLO

Maximum Resource Reduction in one step

Scaling

Problem with Feedback-Based Navigation

- Does not (yet) consider for resource efficiency
- Response time ≅ SLO does not mean no resource reduction opportunities left
- One bottleneck service can push the response close to SLO
- We need microservice-wise augmentation



Bottleneck Identification

- Which metrics reveal a bottleneck service?
- To find out, we create bottlenecks and track the microservice-wise metrics
- We test classification accuracy for different combinations of metrics

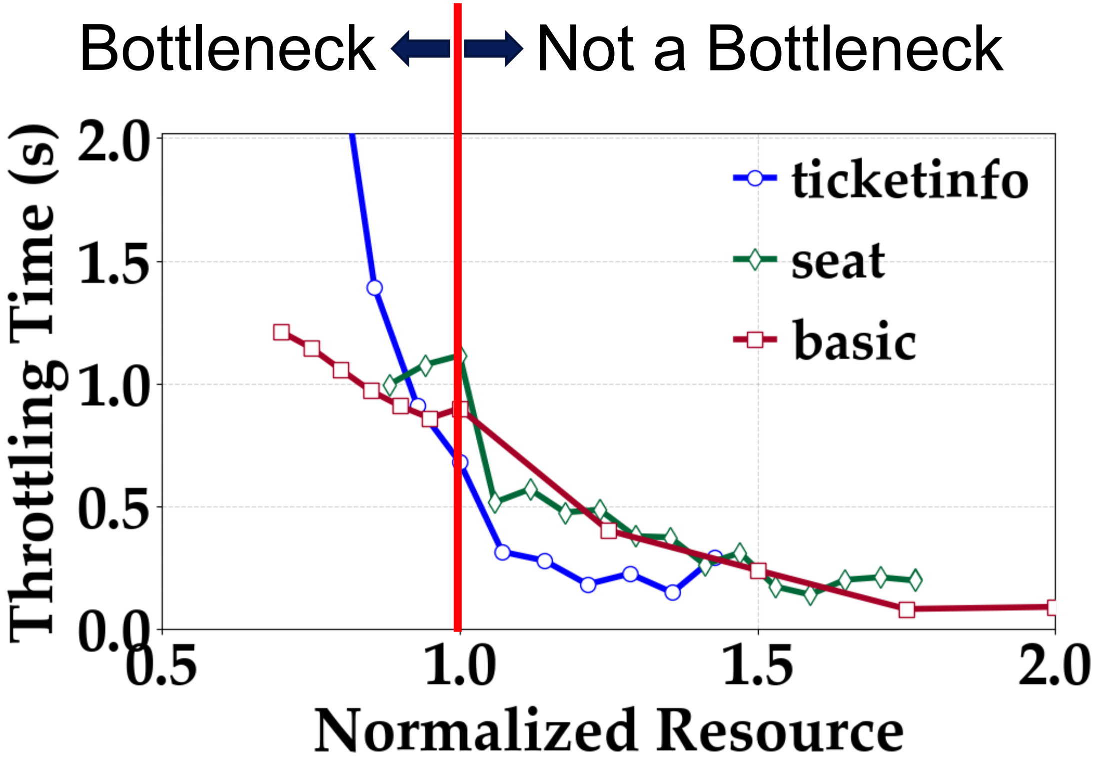
Collected metrics for each microservice

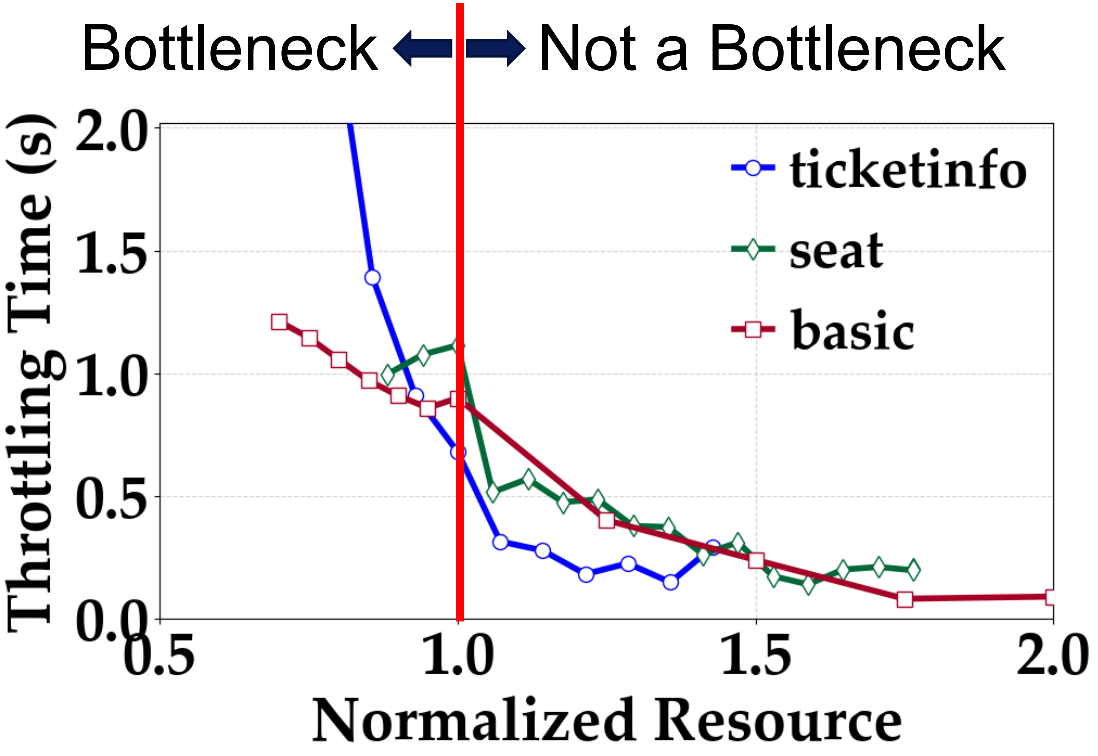
- `CPU Utilizations`
- `CPU Throttles`
- `Memory utilization`
- `service count`
- `service total`
- `self min`
- `self max`
- `self total`
- `self avg`

Application Name	Bottleneck Services	Accuracy (%)
Train Ticket	Seat	94.18
	Seat, Ticketinfo	96.2
	Basic	98.5
	Basic, Seat	99.1
Sock Shop	Carts	100
	Carts, Orders	98.3
Hotel Reservation	Front-end	97.8
	Front-end, Search	95.6

Bottleneck Identification

- `CPU throttling` rapidly increases after bottleneck
- `CPU utilization` increases as we move closer to bottleneck
- Bottleneck thresholds vary across services





Microservice-Wise Augmentation

- When deciding which microservice to reduce resource from
  - Microservices over CPU throttling threshold are filtered out
  - Microservices close to their CPU utilization threshold are chosen with a low probability

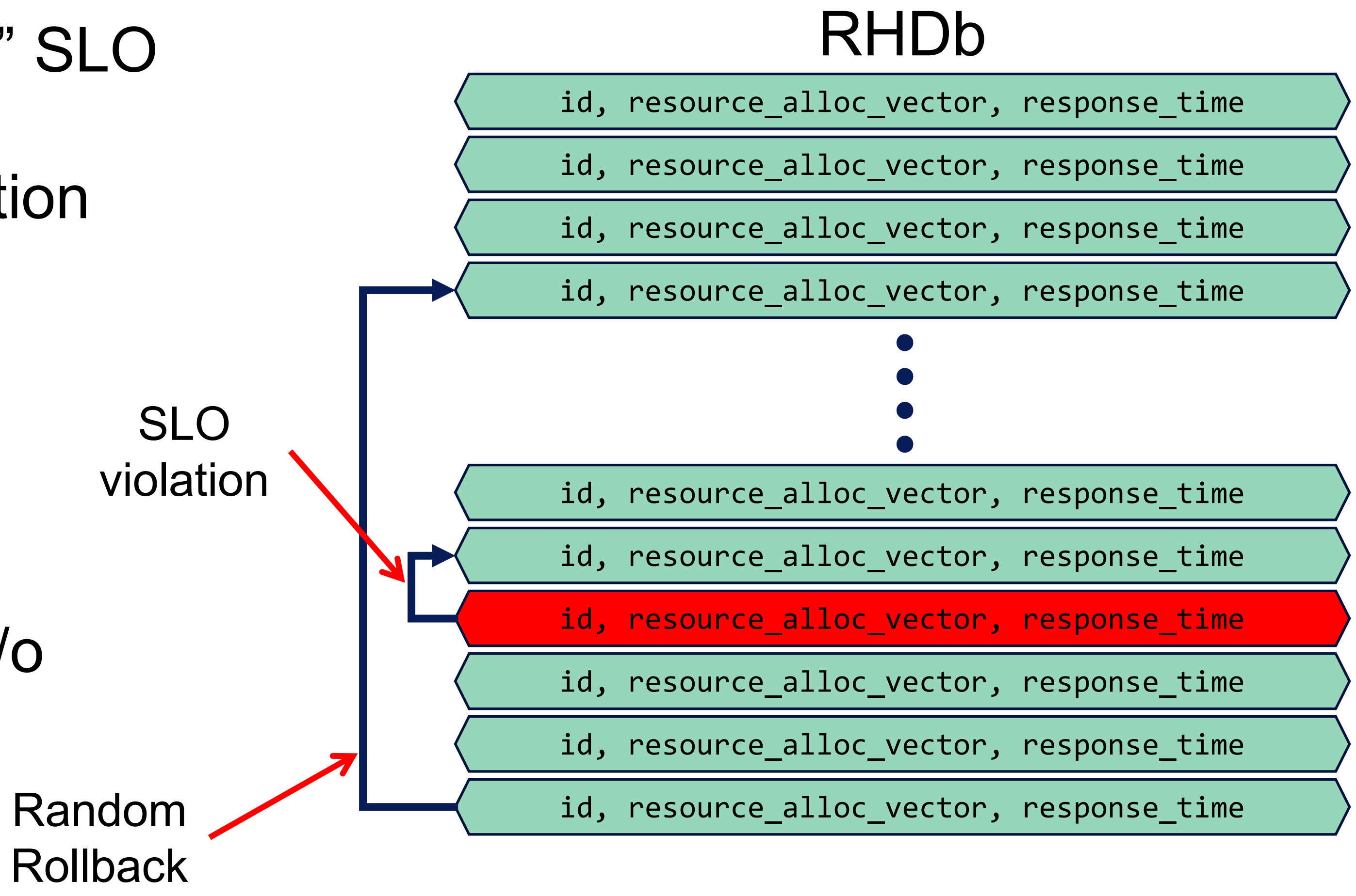
$$p_i^t = 1 - \frac{u_i^{*t-1} - min_{i \in \mathrm{I}^t}(u_i^{*t-1})}{1 - min_{i \in \mathrm{I}^t}(u_i^{*t-1})}$$

- Runtime update of the bottleneck thresholds

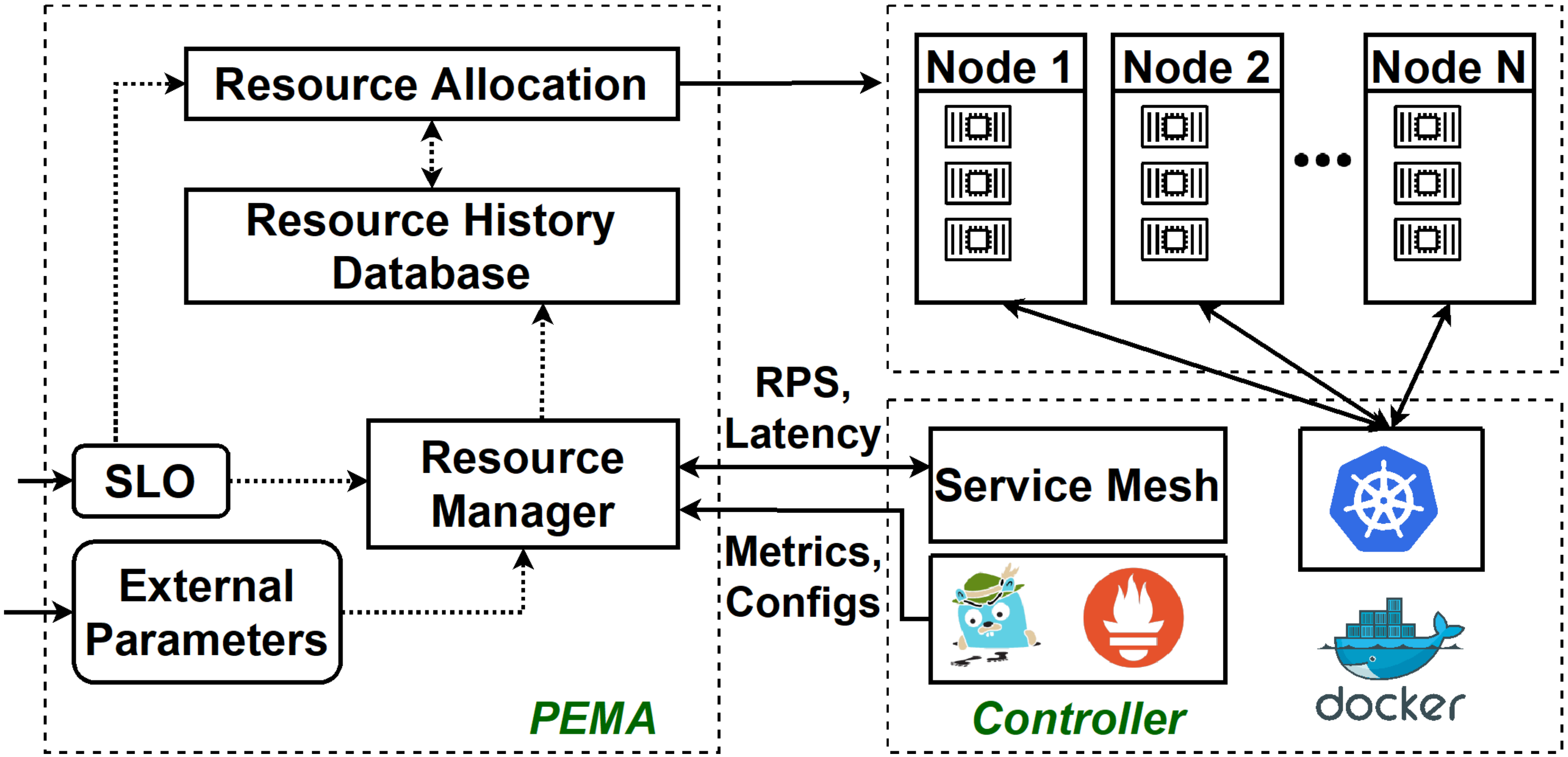
$$U_i^{th} = \max(U_i^{th}, u_i^{t-1})$$
$$H_i^{th} = \max(H_i^{th}, H_i^{t-1})$$

Iterative Resource Allocation

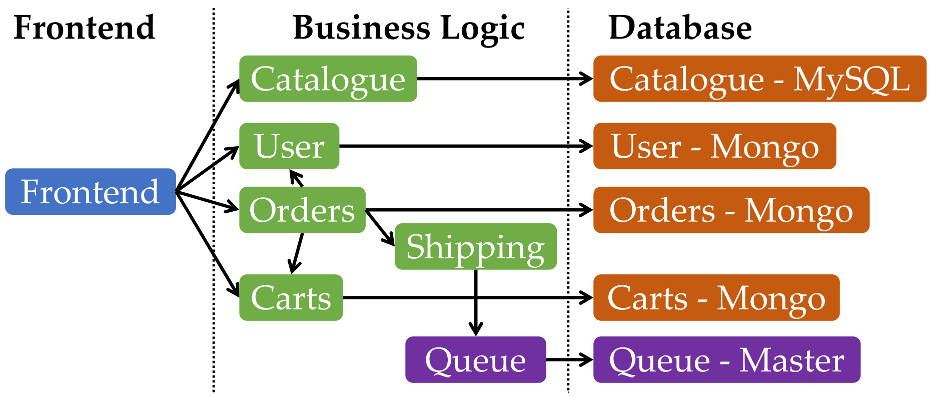
- PEMA can cause “unintentional” SLO violations
  - We rollback to a prior allocation
  - We keep all past resource allocations in a “Resource Allocation History Database (RHDb)”
- Escape suboptimum
  - Randomly roll-back (even w/o SLO violation) to a past configuration



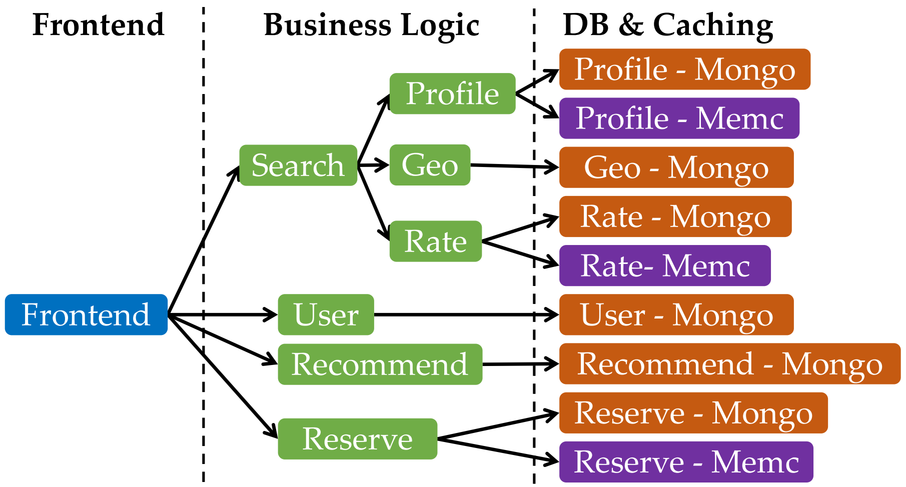
Practical **E**fficient **M**icroservice **A**utoscaling (PEMA)



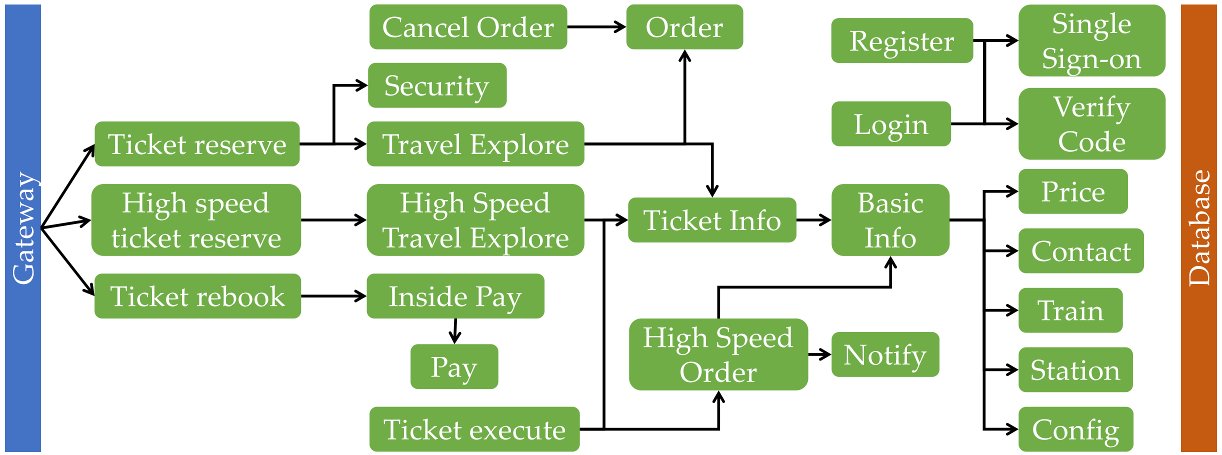
Microservice Implementations



Sock Shop



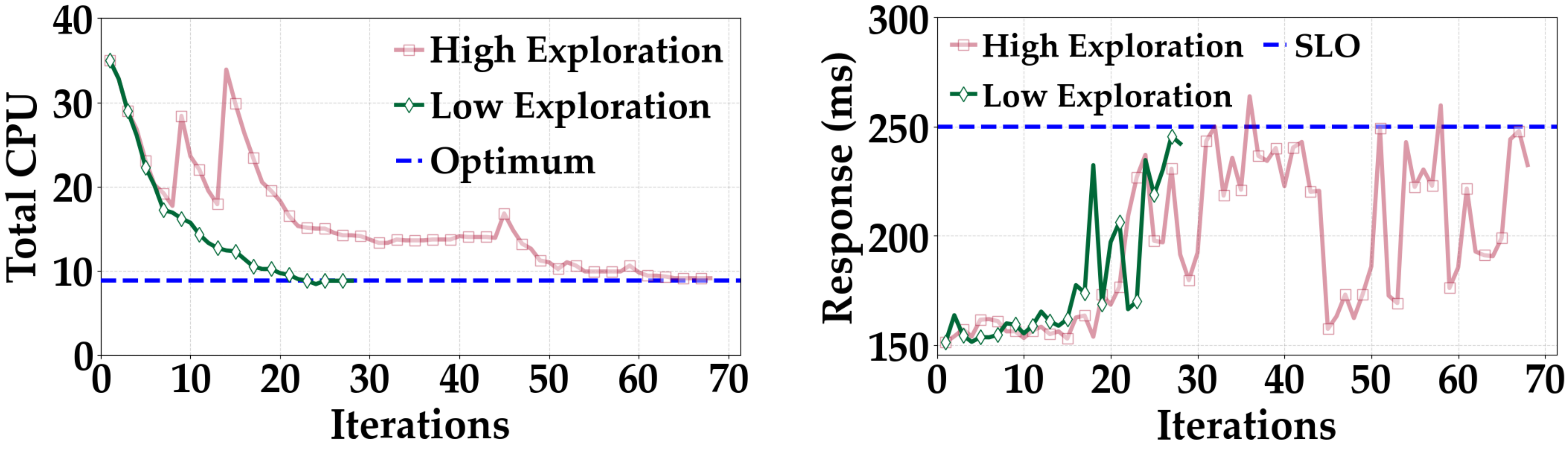
Hotel Reservation



Train Ticket

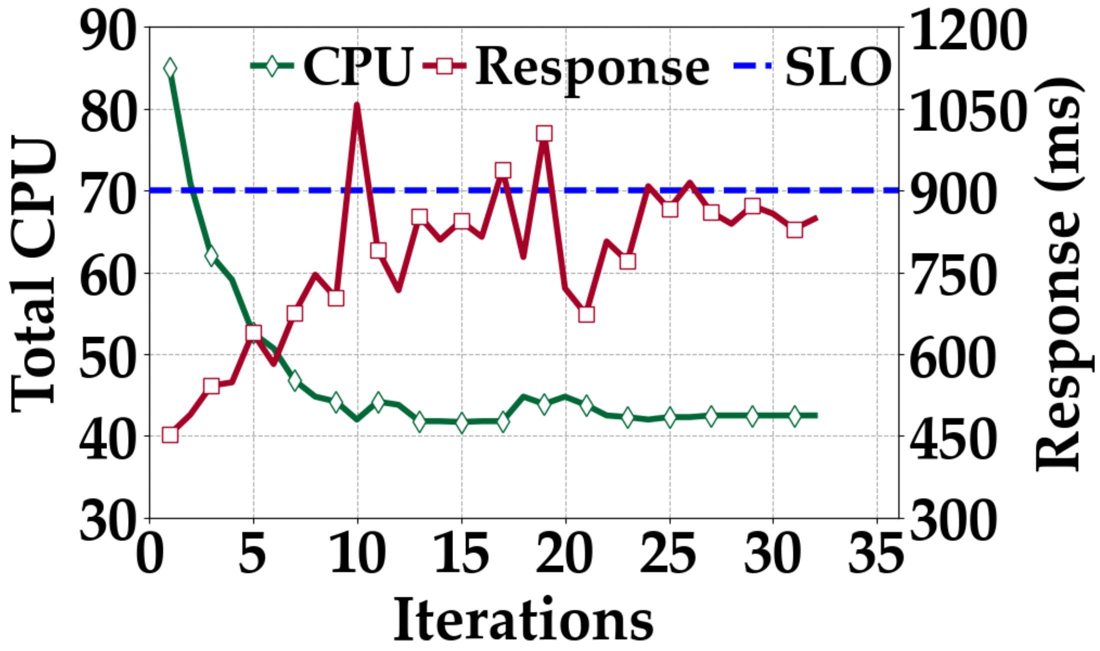
Convergence

Execution of PEMA in Sock Shop

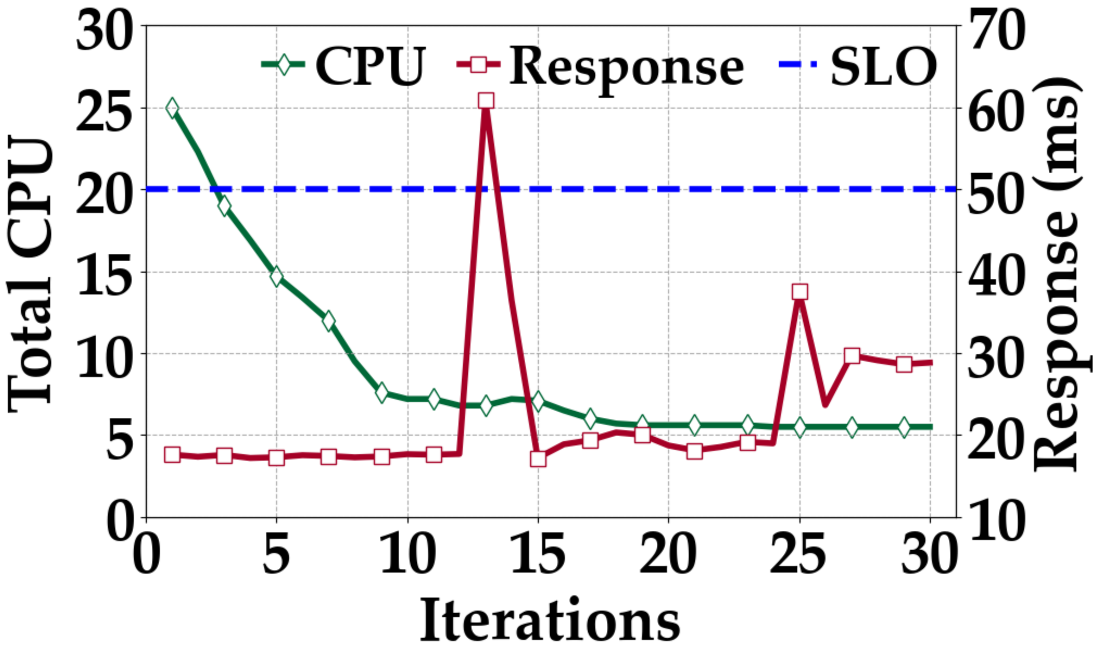


Convergence

Execution of PEMA in Train Ticket

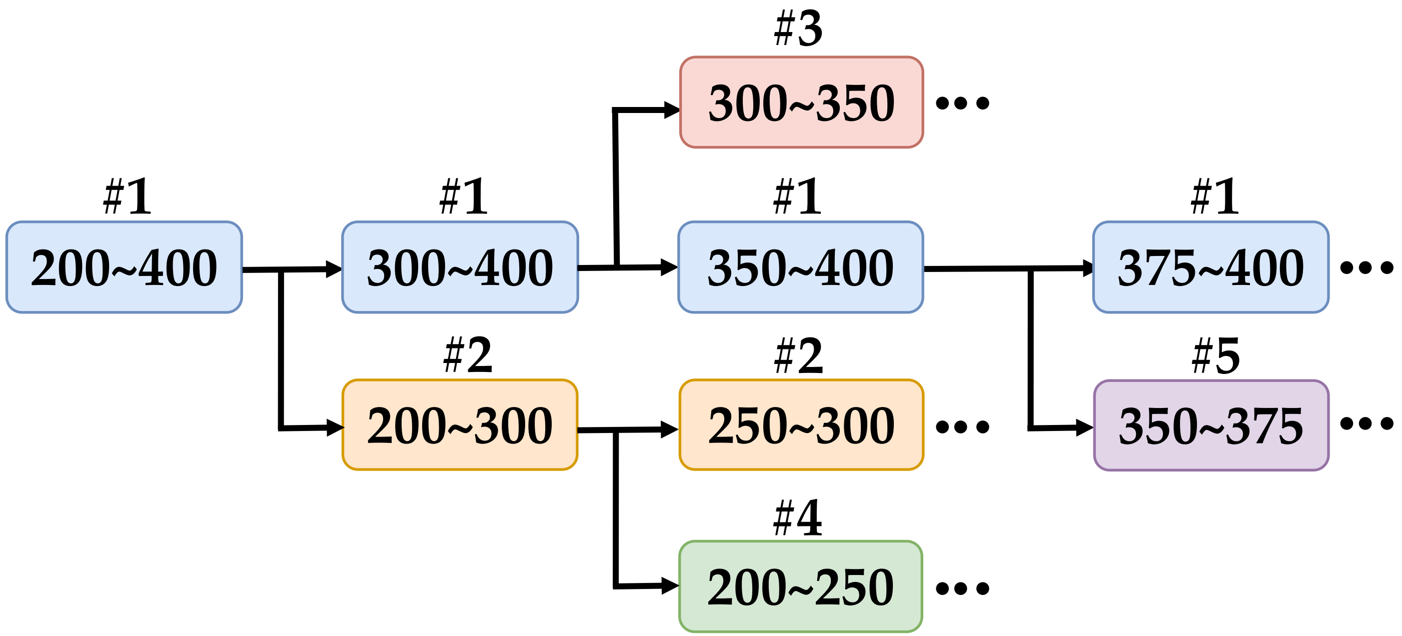


Execution of PEMA in Hotel Reservation



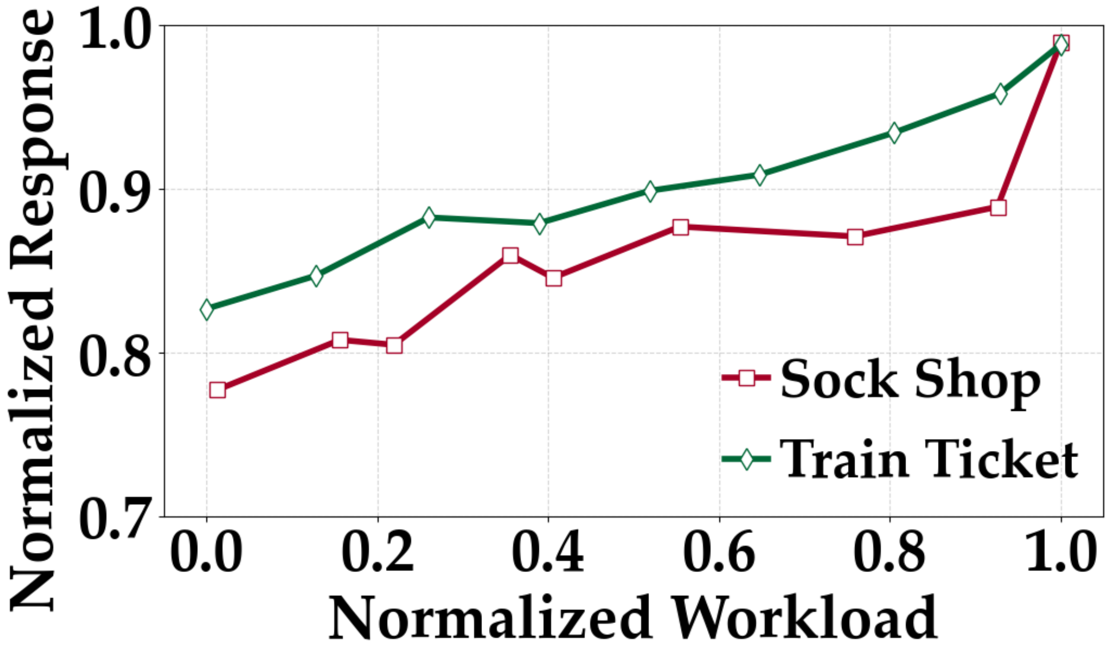
Adapting to Workload Variation

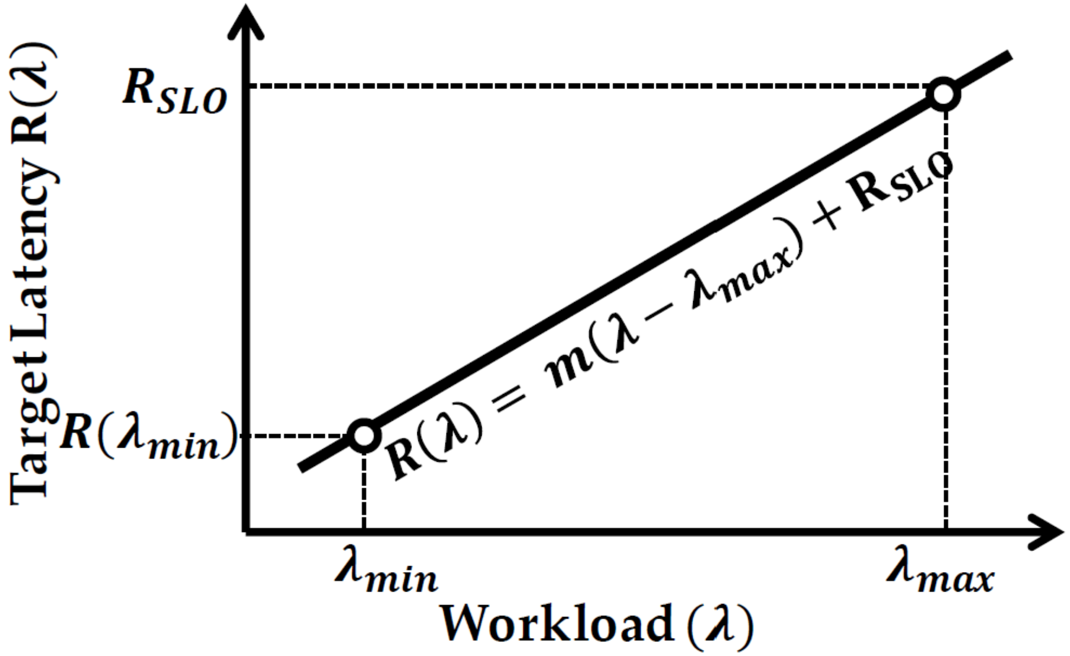
- We divide workloads into ranges (e.g.,200~400→200~225, 225~250,…)
- Each workload range has its own resource manager and RHDb
- We bootstrap the resource allocation



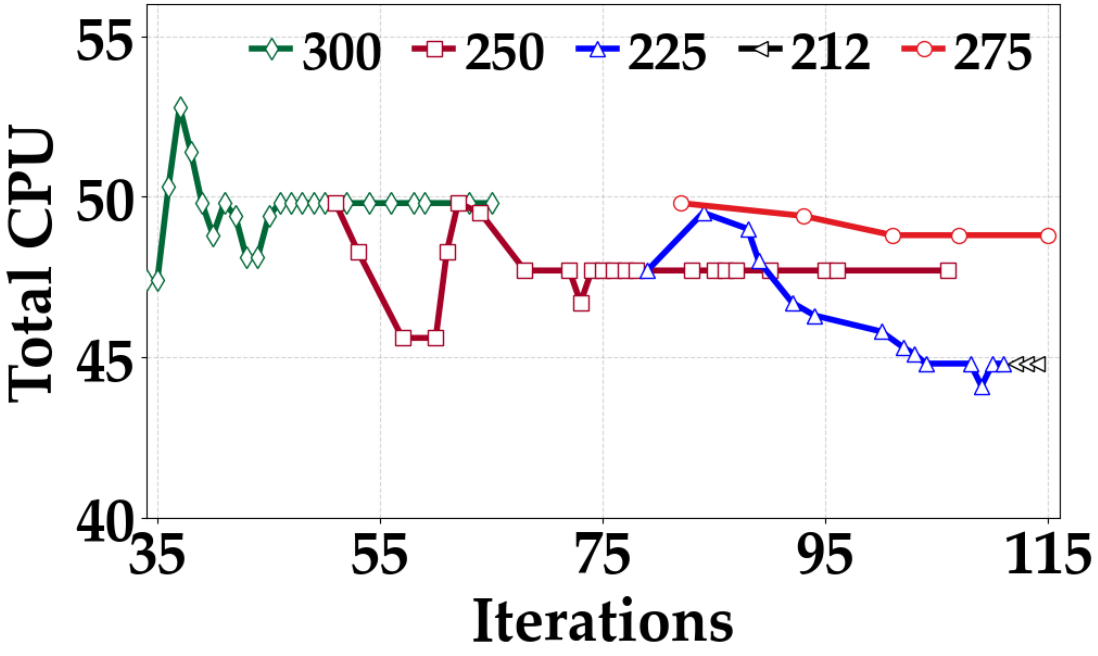
Dynamic SLO Target

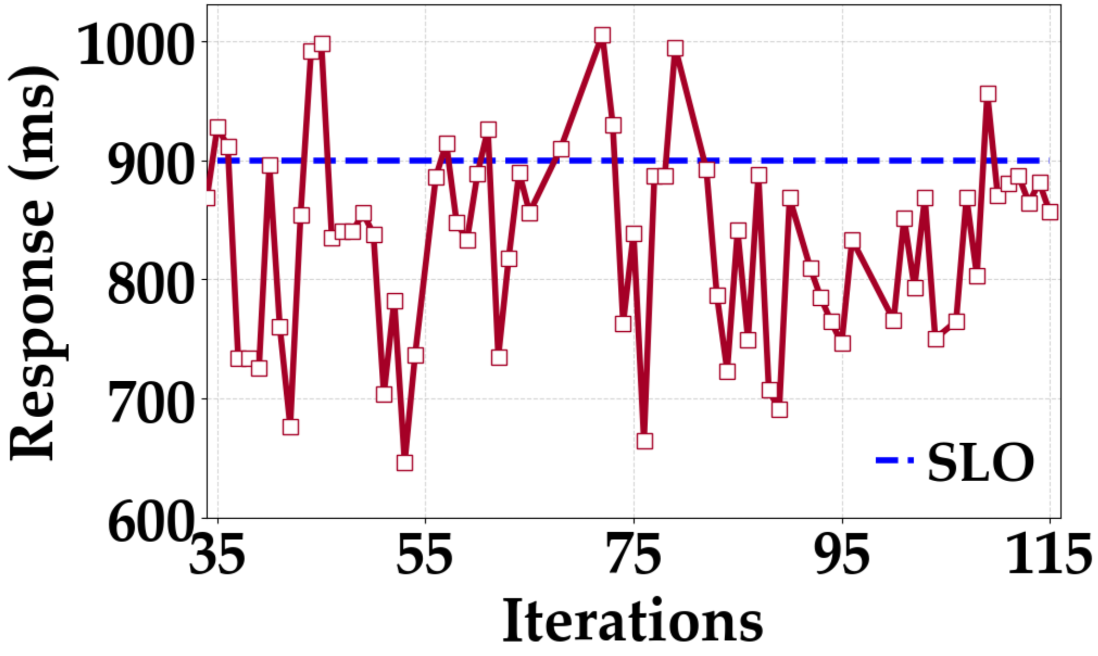
- When the workload range is large, lower workloads within a range will trigger resource reduction
- Solution: Dynamic SLO target



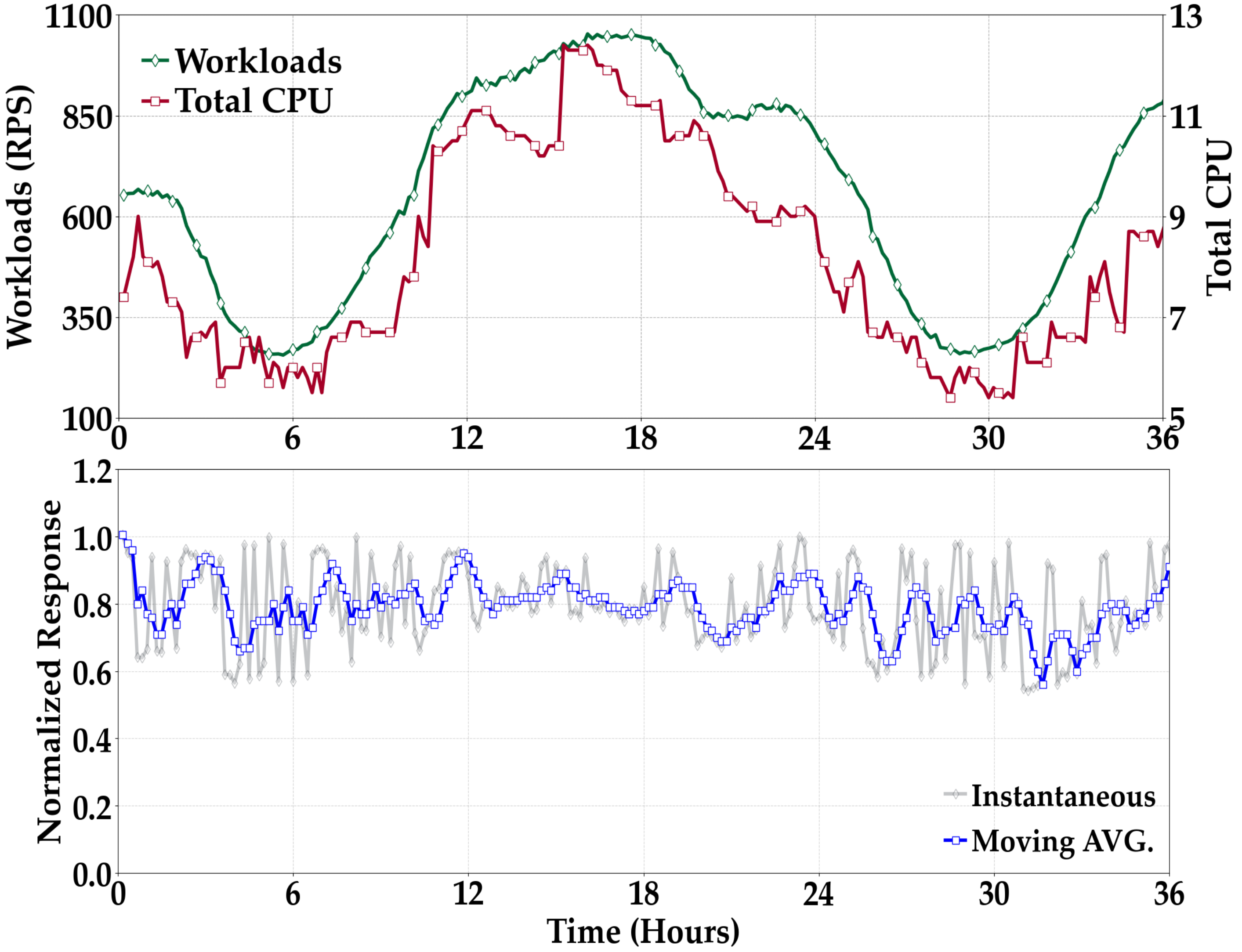


Dynamic SLO in Execution

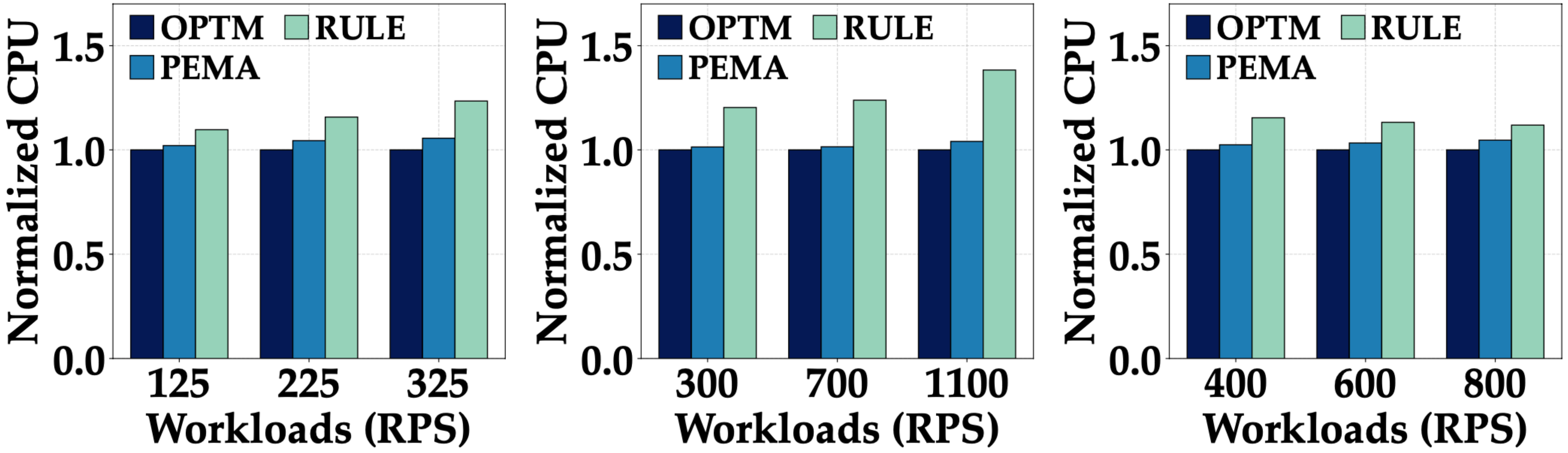




Extended Execution

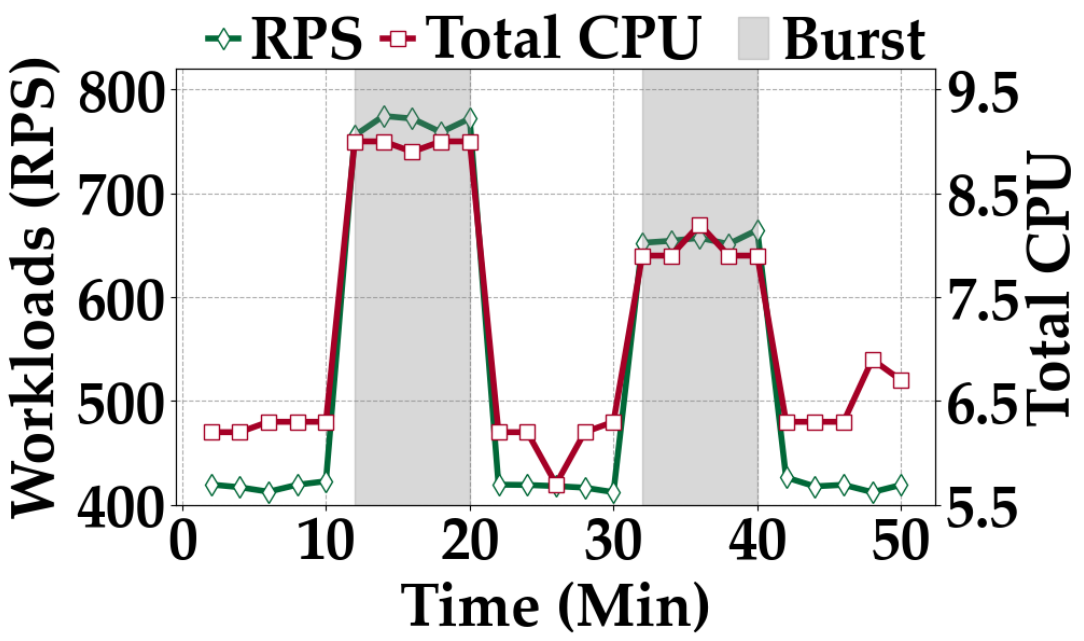


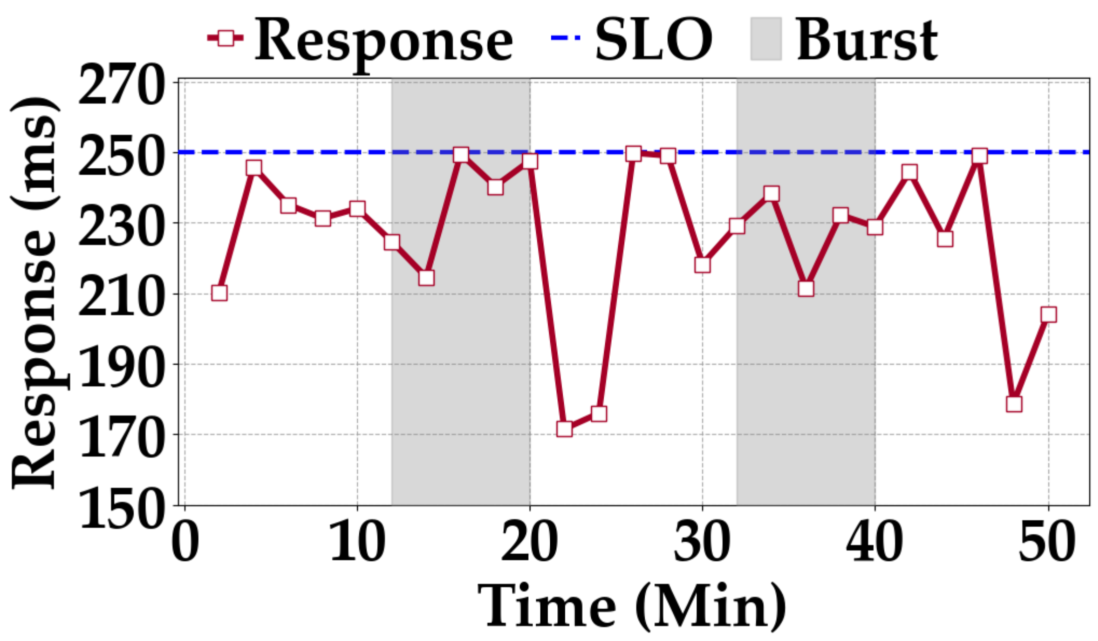
Resource Efficiency



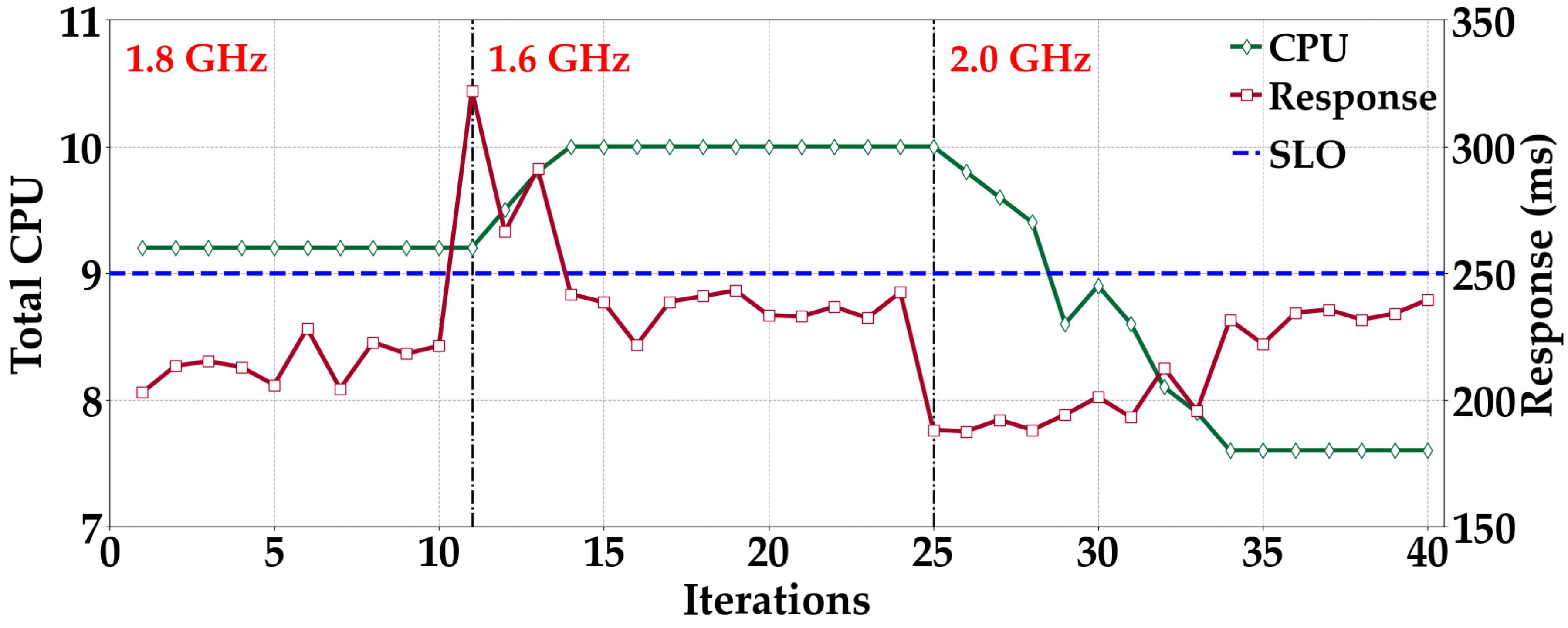
(a) TrainTicket (b) SockShop (c) HotelReservation

Bursty Workloads

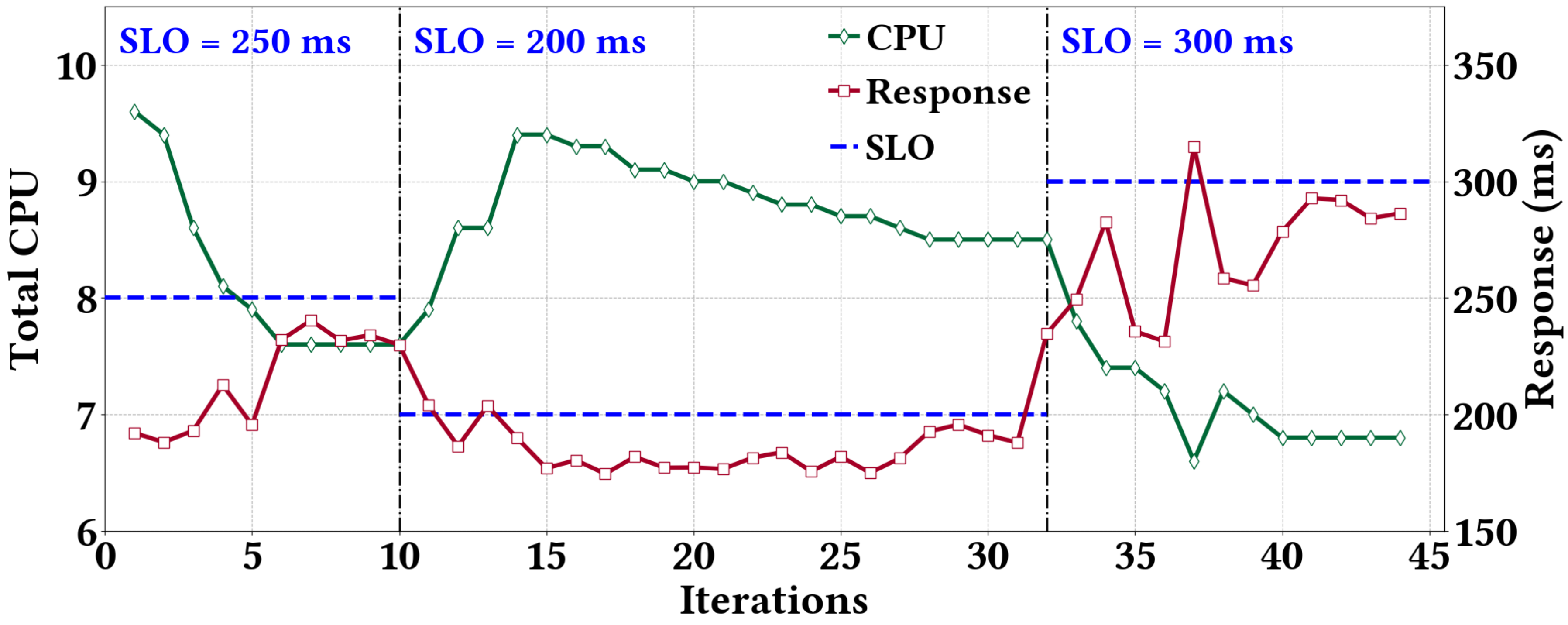




Adaptive to System Changes



Adaptive to SLO Variations



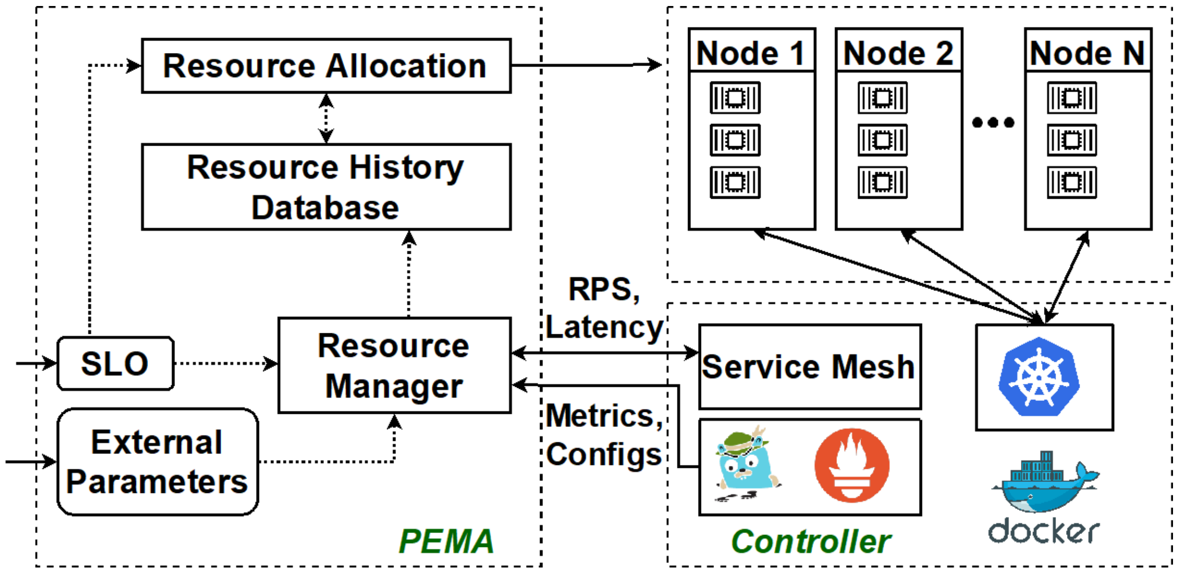
Limitations

- Design limitations
  - Lightweight design → cannot capture ML-like details
  - Randomized exploration cannot guarantee optimality

- Implementation limitations
  - Suffers through unintentional SLO violation until next update
  - Degree of SLO violation is not considered during rollback
  - Does not utilize the past resource allocation history
  - Manges only CPU allocations
  - Does not explicitly address vertical and horizontal scaling

Key Take Away

- PEMA – Practical Efficient Microservice Autoscaling
  - Online and not data intensive
  - No intentional QoS violation
  - Adaptive to changes



Thank You!

Questions?