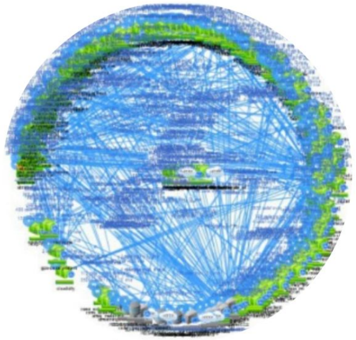
Autothrottle:

A Practical Bi-Level Approach to Resource Management for SLO-Targeted Microservices

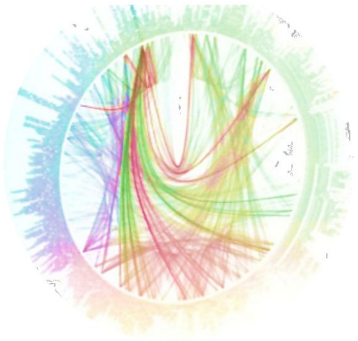
Zibo Wang[1,2], Pinghe Li[3], Chieh-Jan Mike Liang[1], Feng Wu[2], Francis Y. Yan[1]

[1] *Microsoft Research*
[2] *University of Science and Technology of China*
[3] *ETH Zurich*

Cloud applications are shifting toward microservices

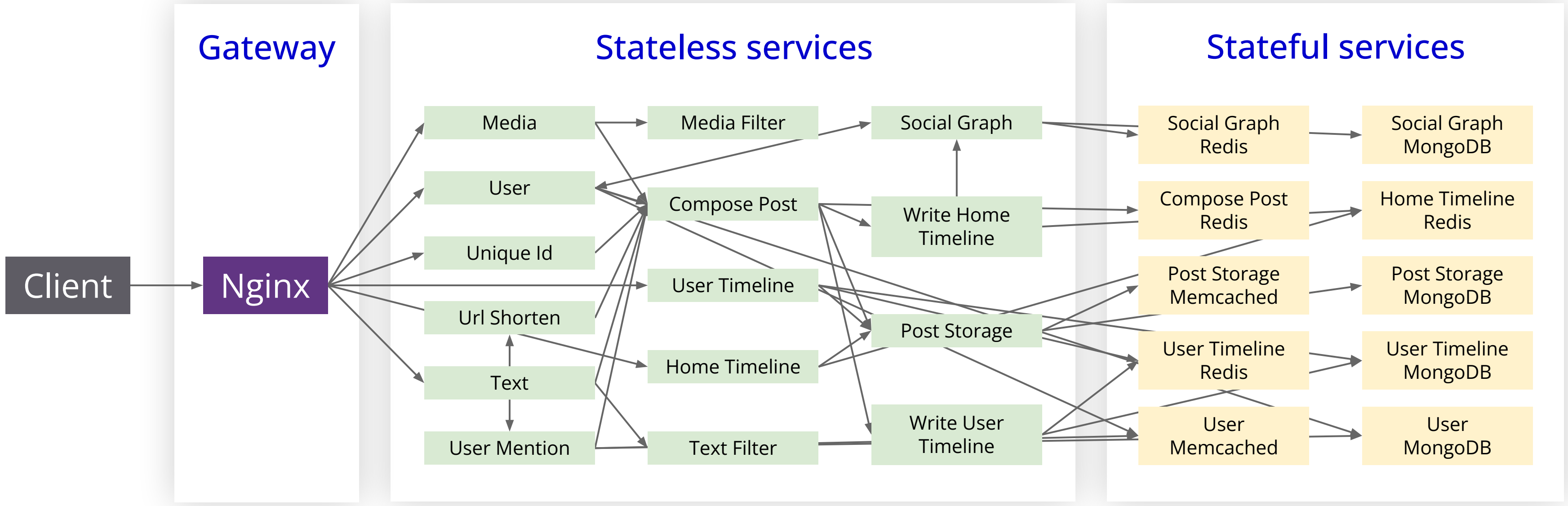


Netflix

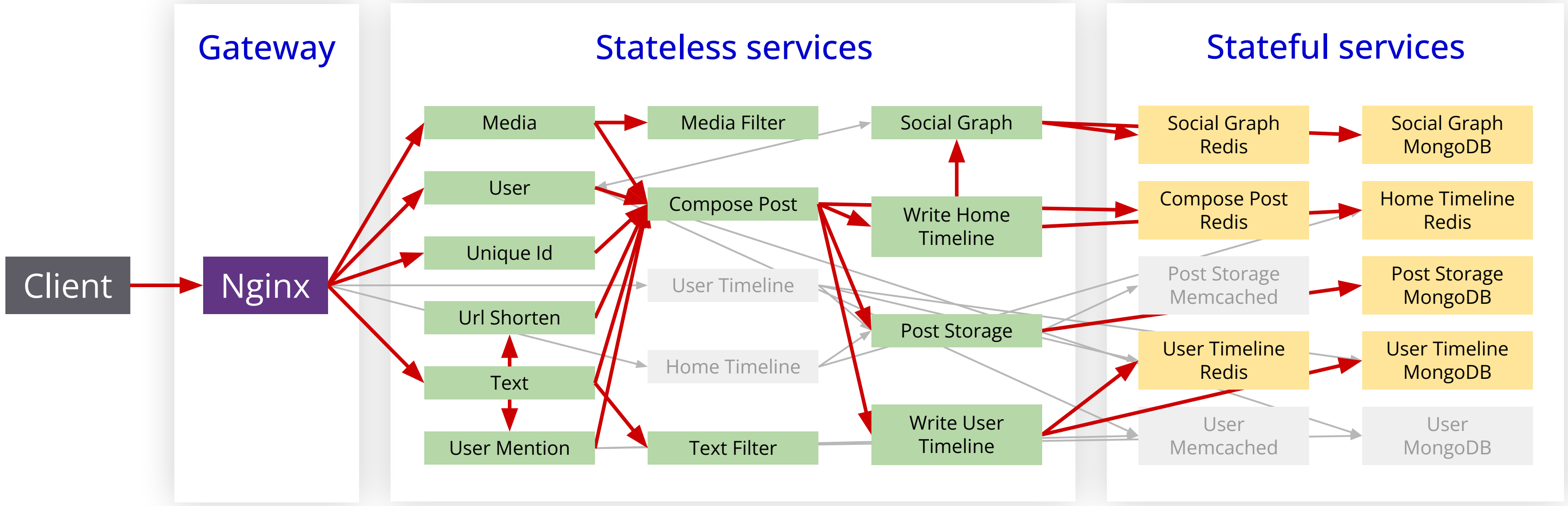


Twitter

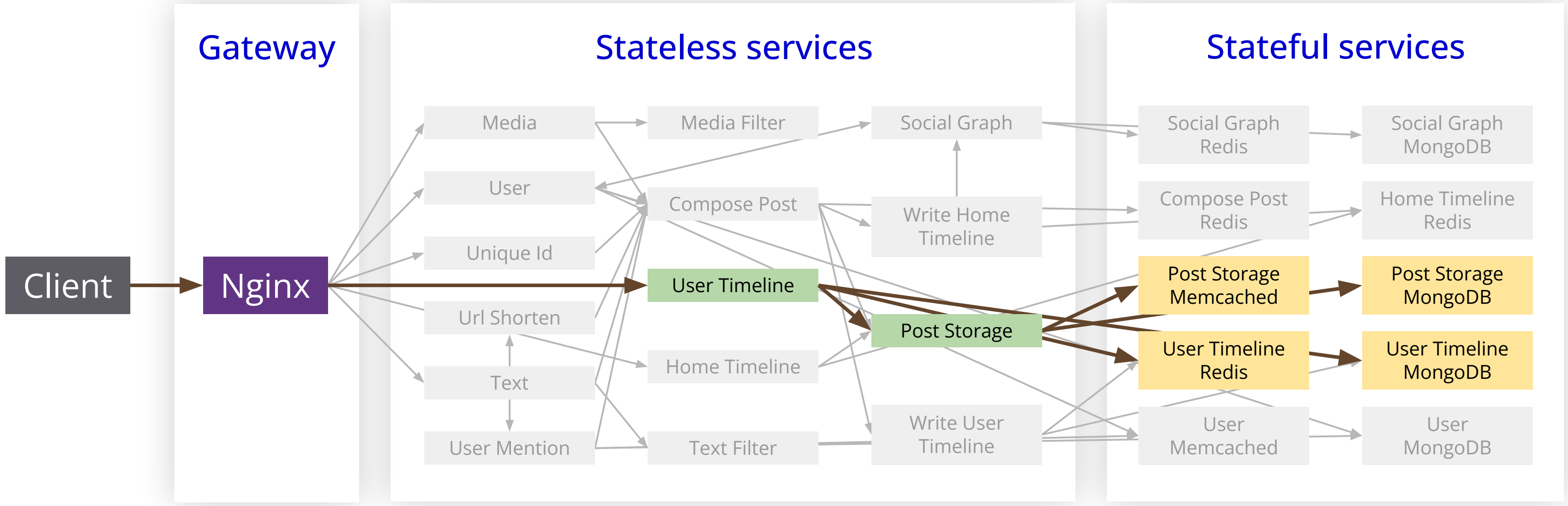
What microservice applications look like



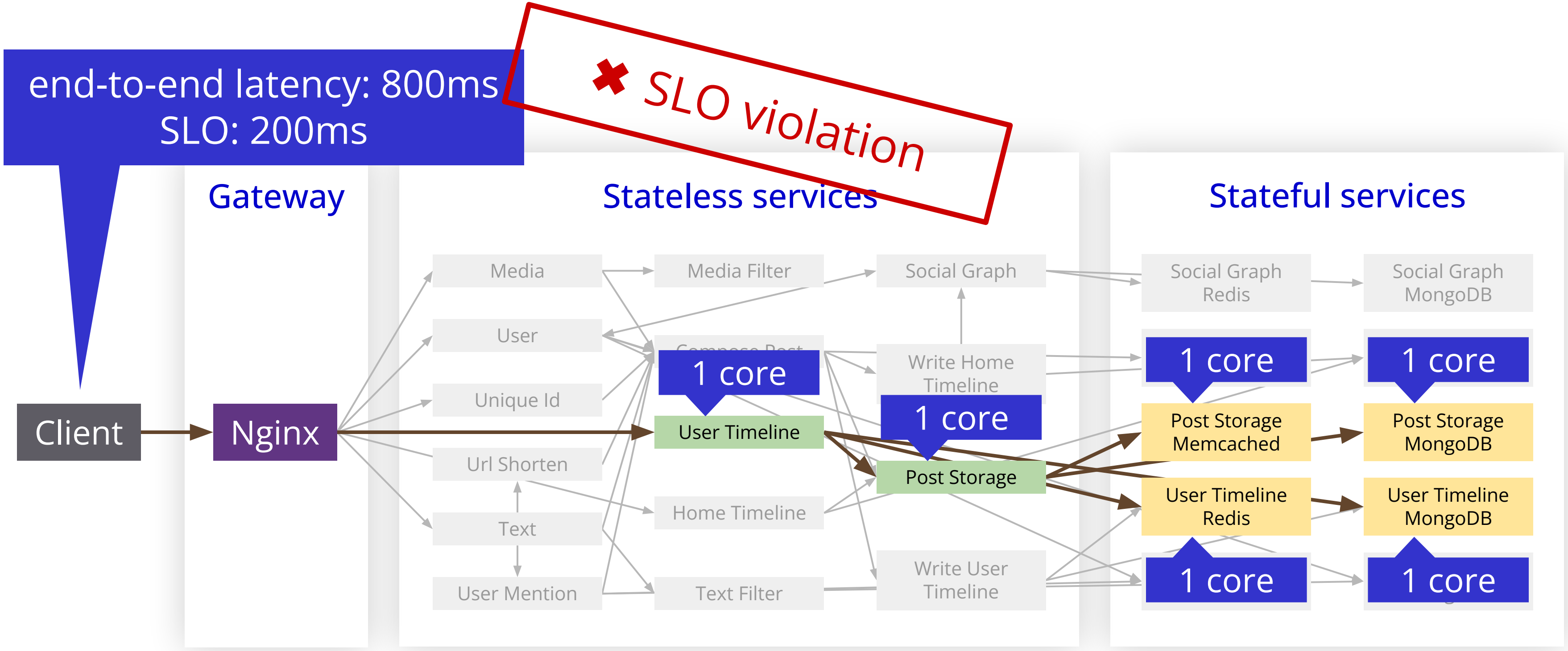
A client request traverses many services



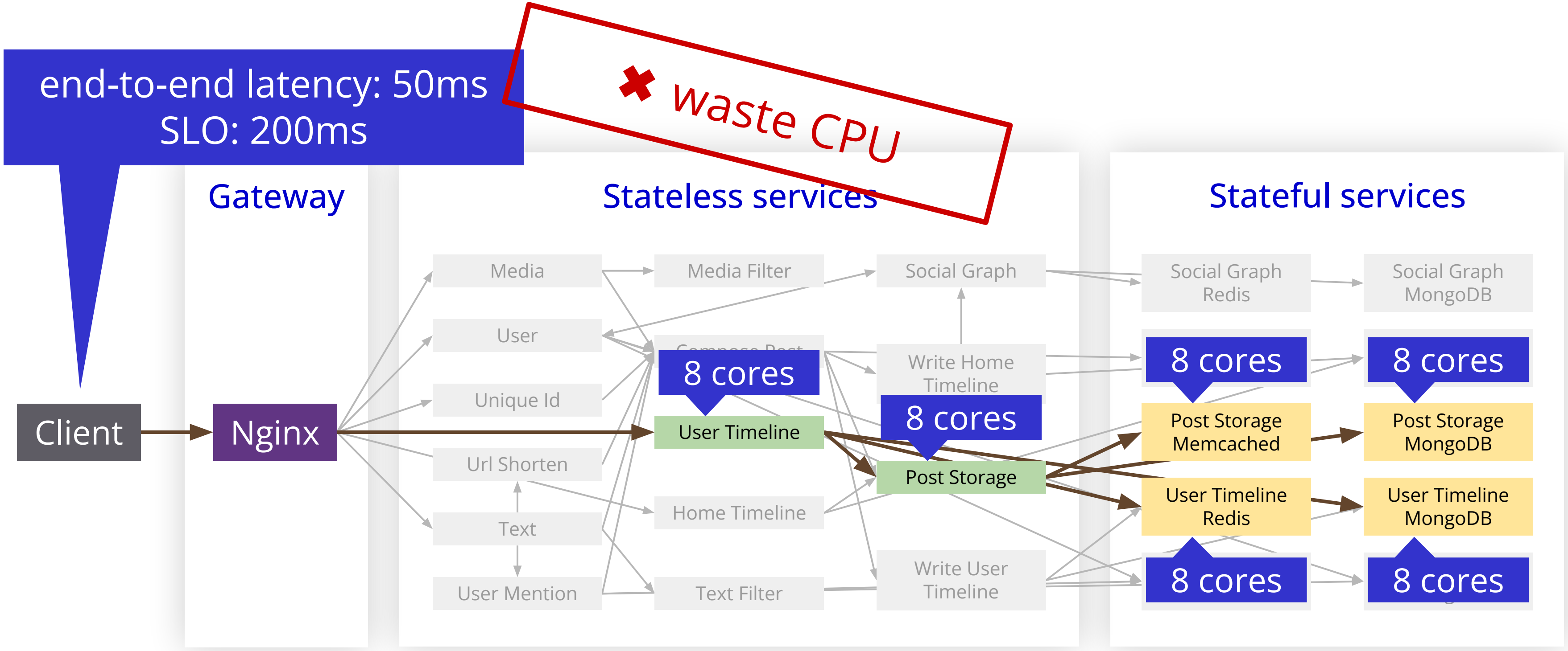
Different requests have different trajectories



Inadequate CPU allocations => high application latency

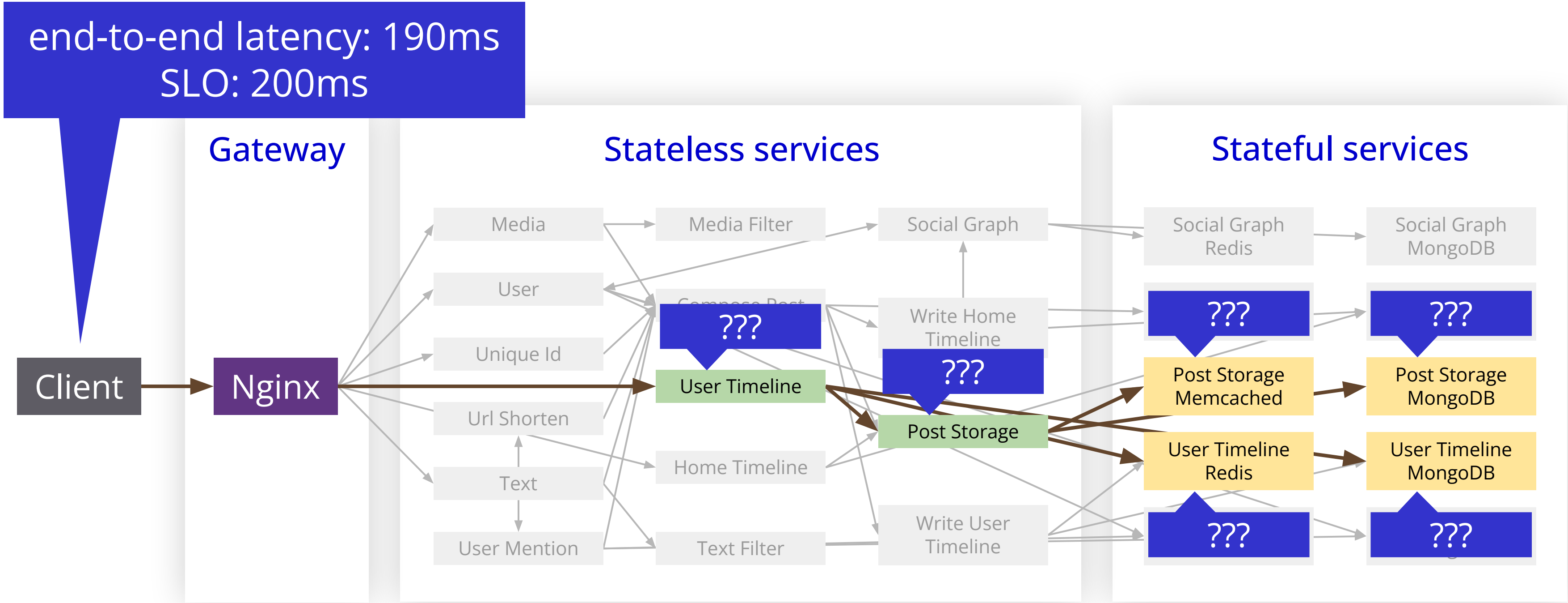


Excessive CPU allocations => waste of resources



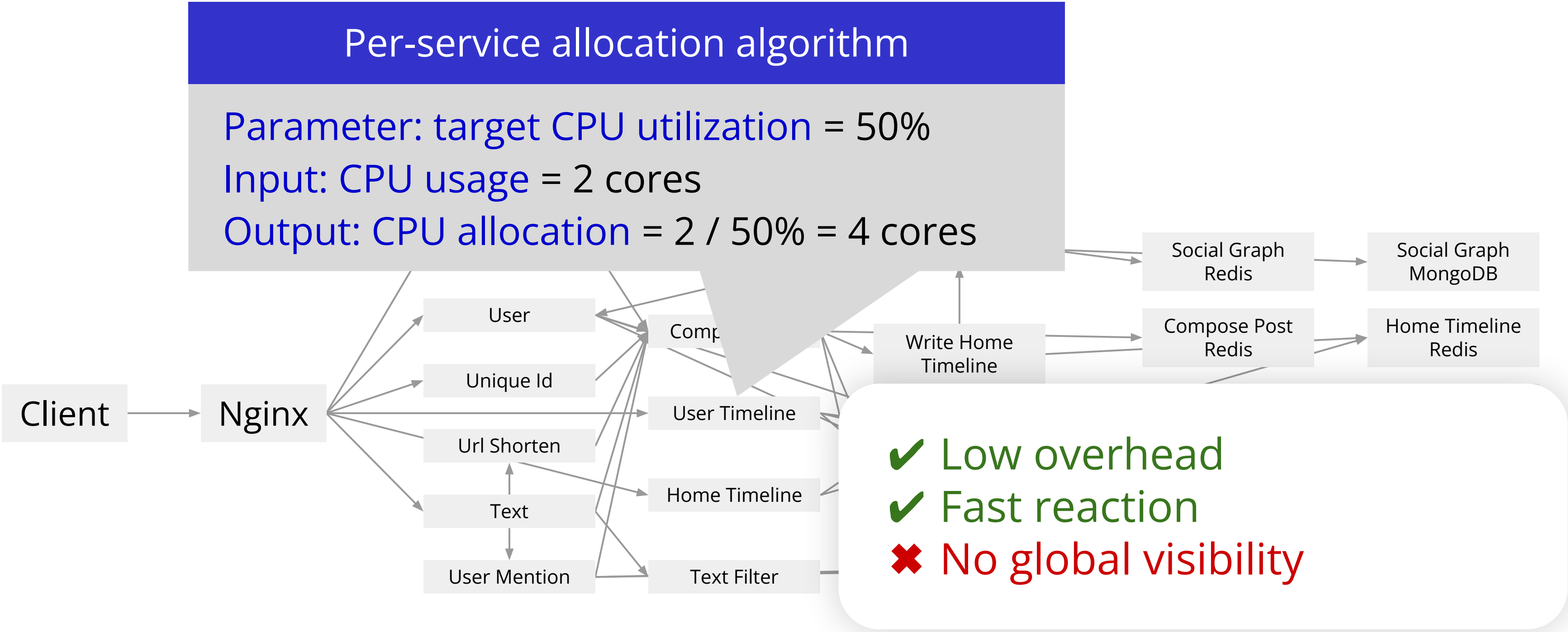
Minimizing CPU allocation while meeting SLO

- Search space grows exponentially with number of services
- Mapping from CPU allocations to latency is unclear



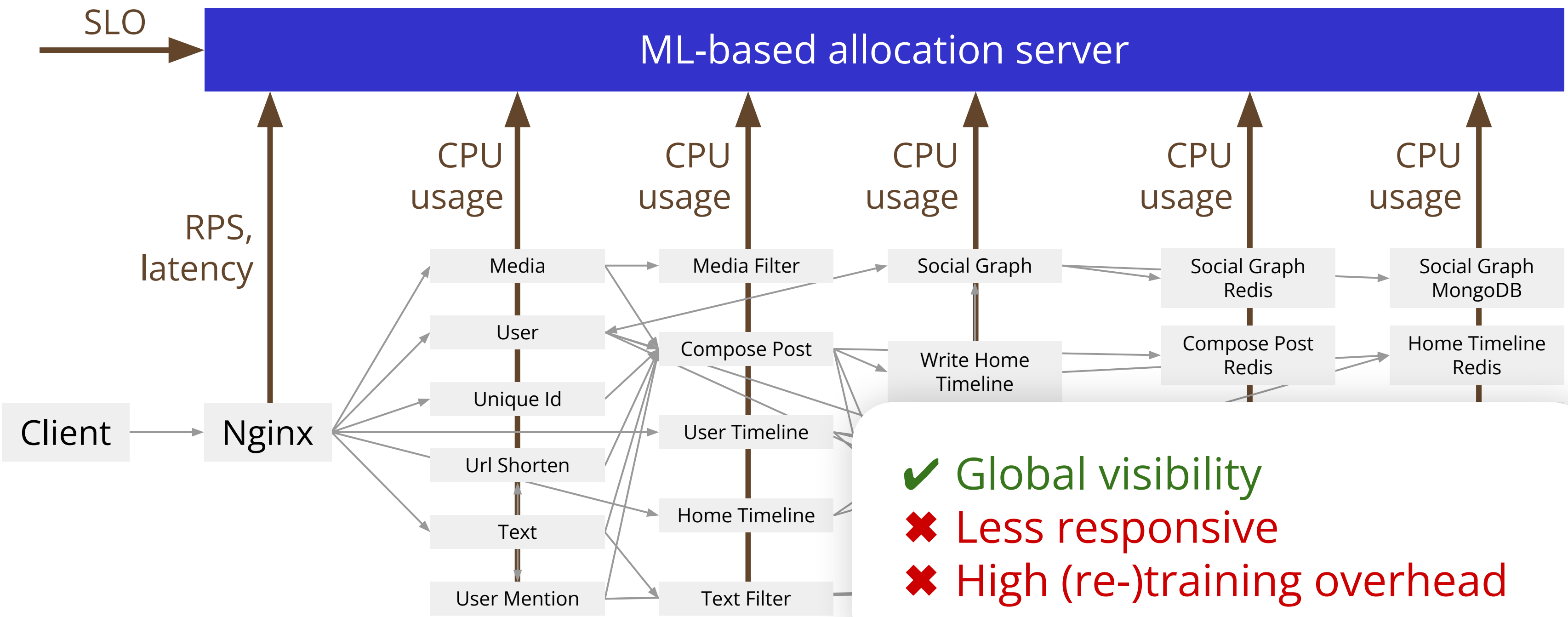
Existing approach: service-level allocation

- Example: Kubernetes' default heuristics



Existing approach: application-level allocation

- Example: Sinan (ASPLOS '21)

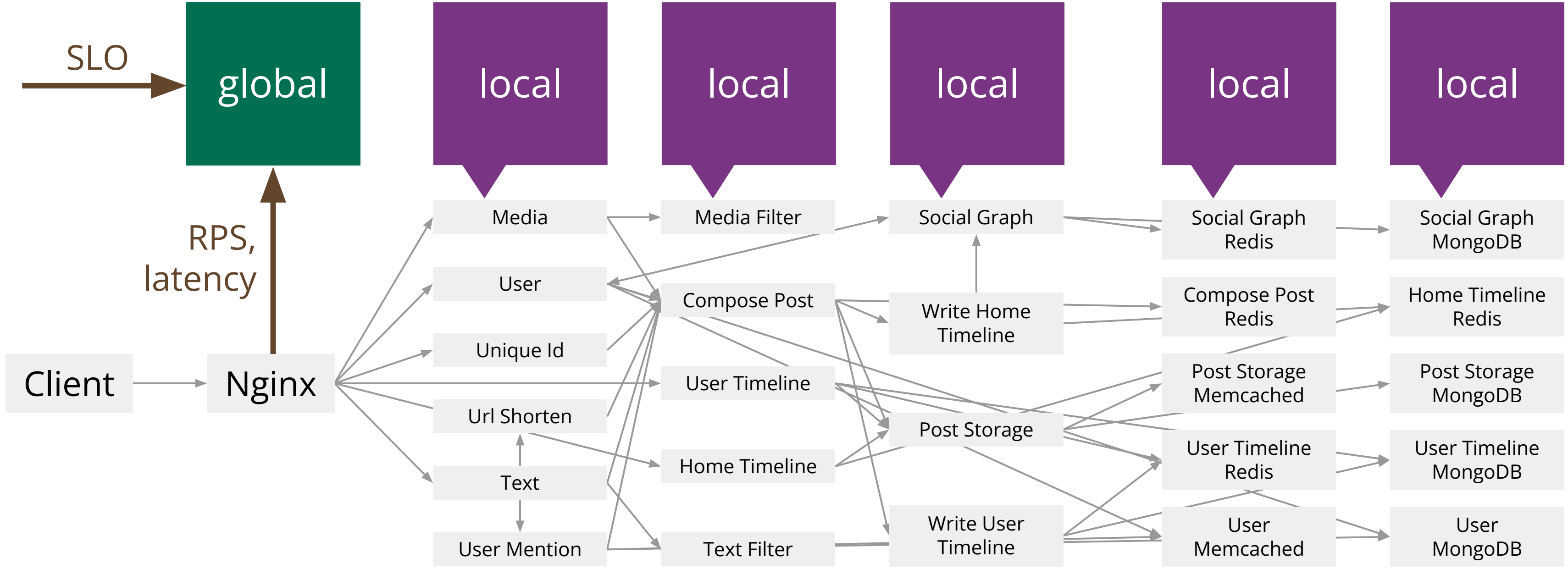


✔ Global visibility
✖ Less responsive
✖ High (re-)training overhead

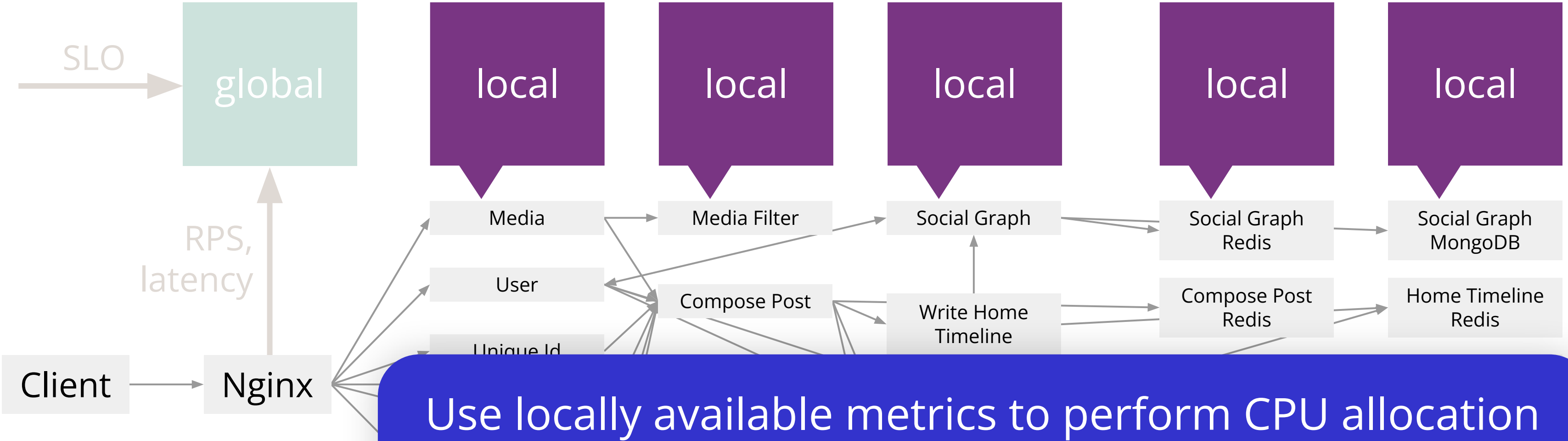
How to obtain the best of both worlds?

	Service-level	Application-level	???
Low overhead Fast reaction	✓	✗	✓
Global visibility	✗	✓	✓

Our bi-level approach to resource management



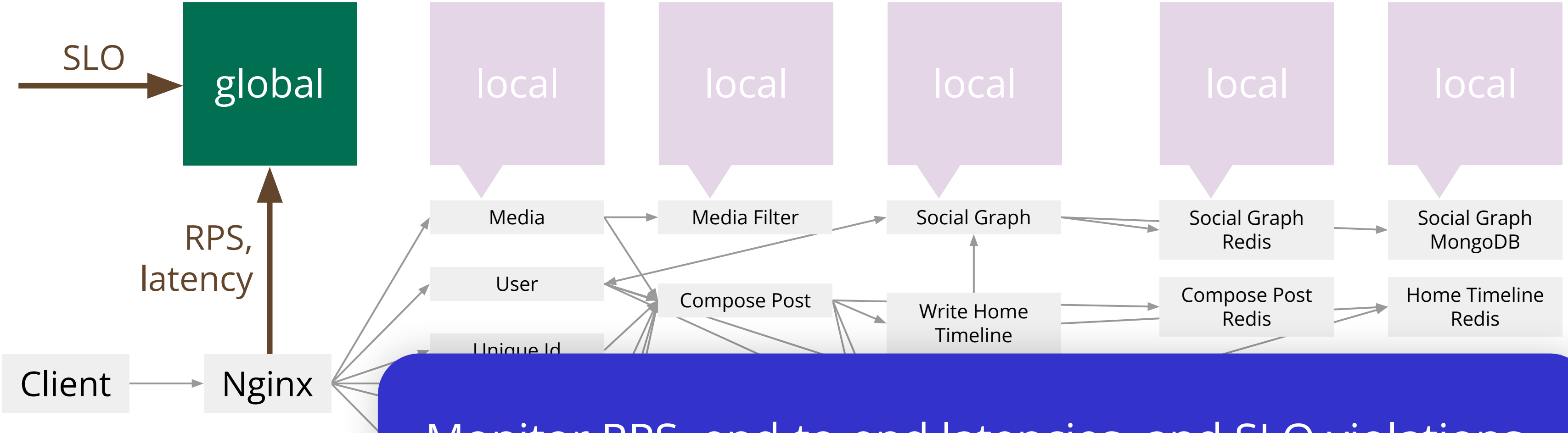
Our bi-level approach to resource management



Use locally available metrics to perform CPU allocation

✔ Low overhead
✔ Fast reaction

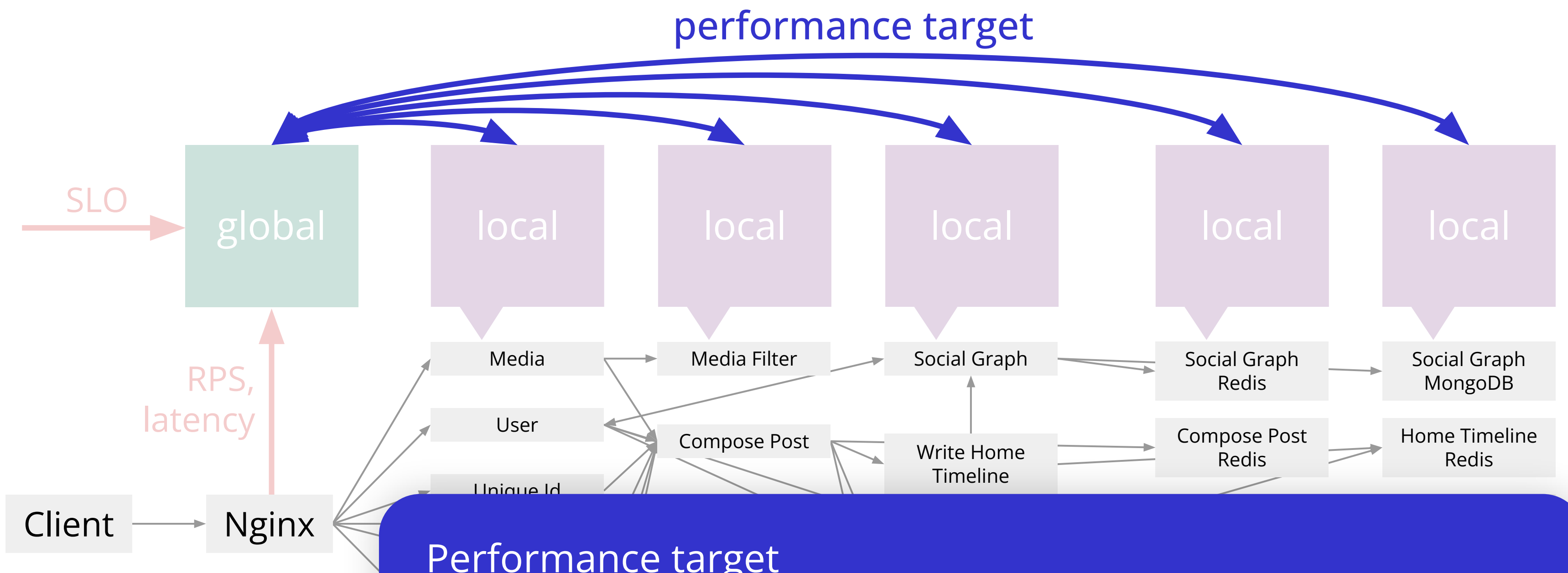
Our bi-level approach to resource management



Monitor RPS, end-to-end latencies, and SLO violations

✔ Global visibility

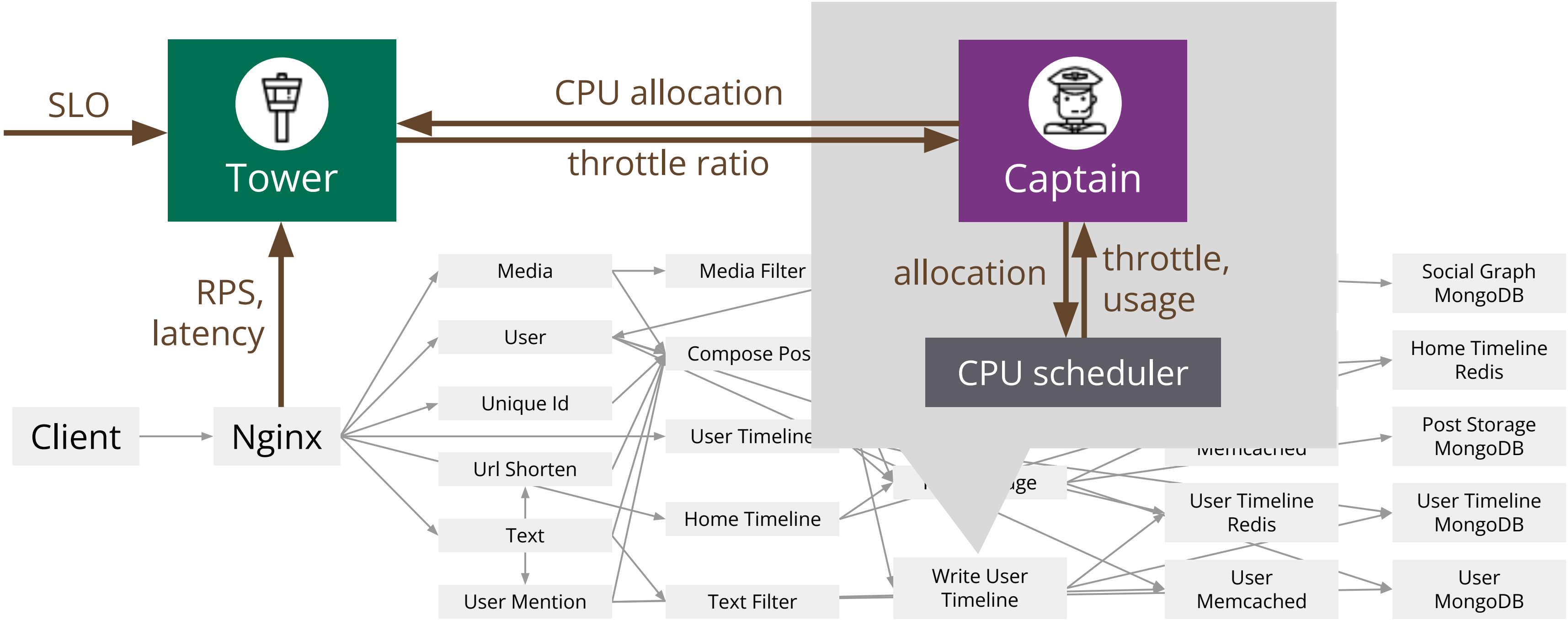
Our bi-level approach to resource management



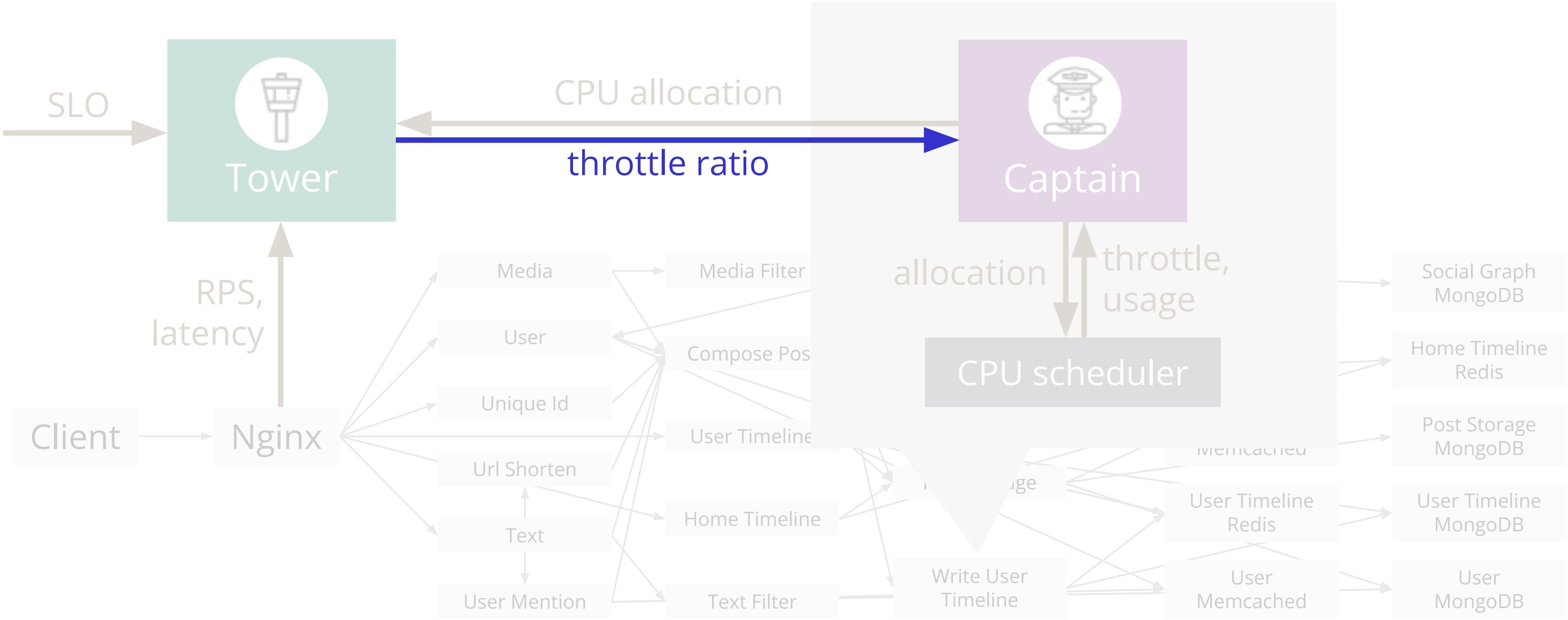
Performance target

- Periodically determined by the global controller
- Enabling local controllers to remain autonomous

Implementing bi-level approach with **Autothrottle**

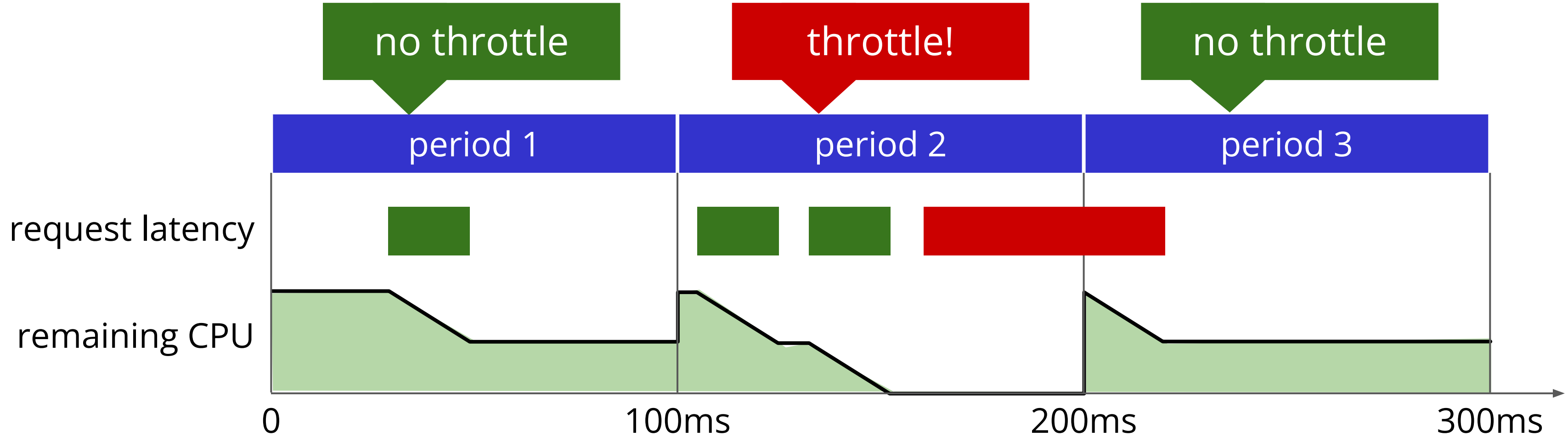


Interface: throttle ratio



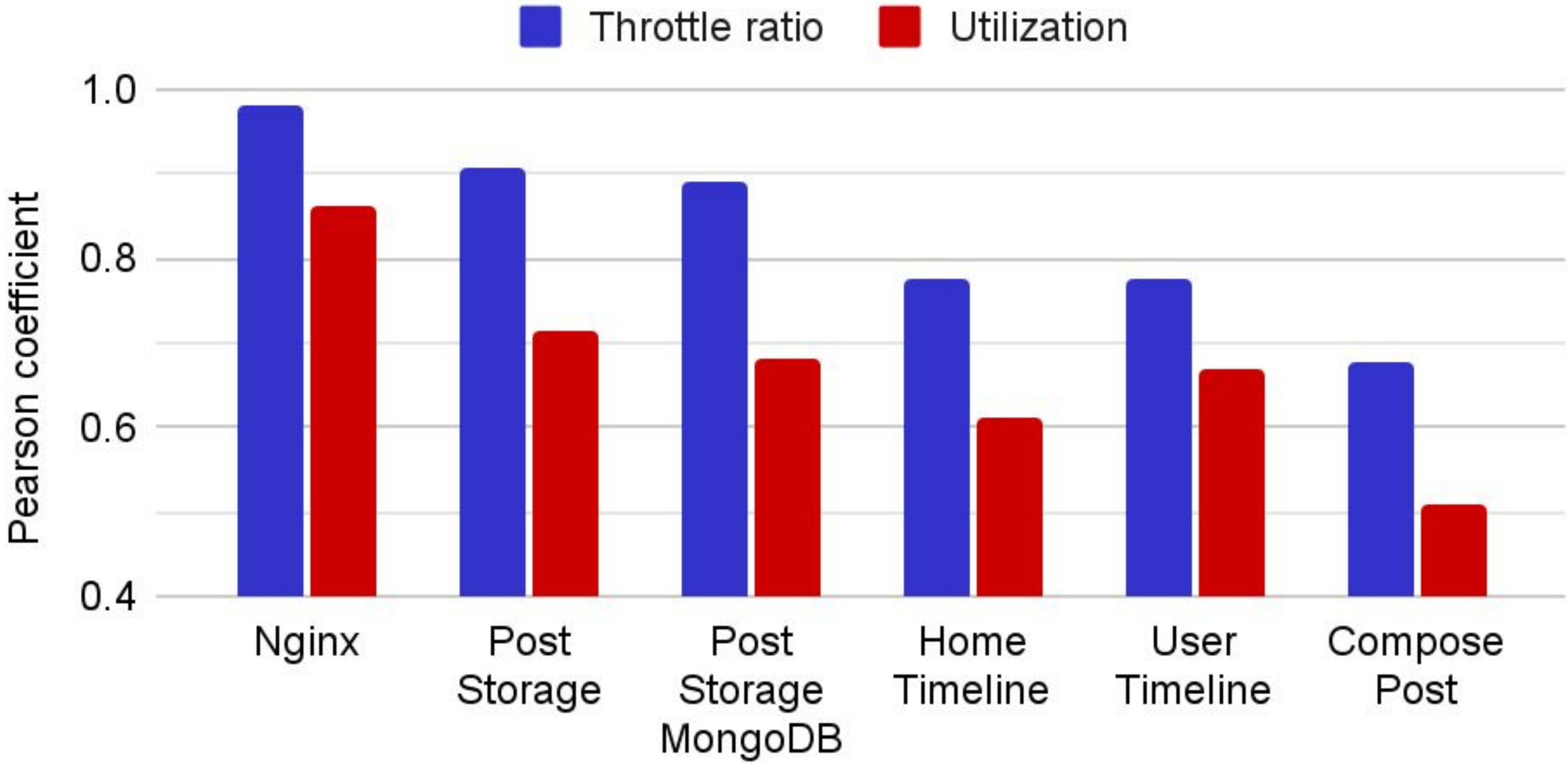
Interface: throttle ratio

- Example: Linux CFS (Completely Fair Scheduler)

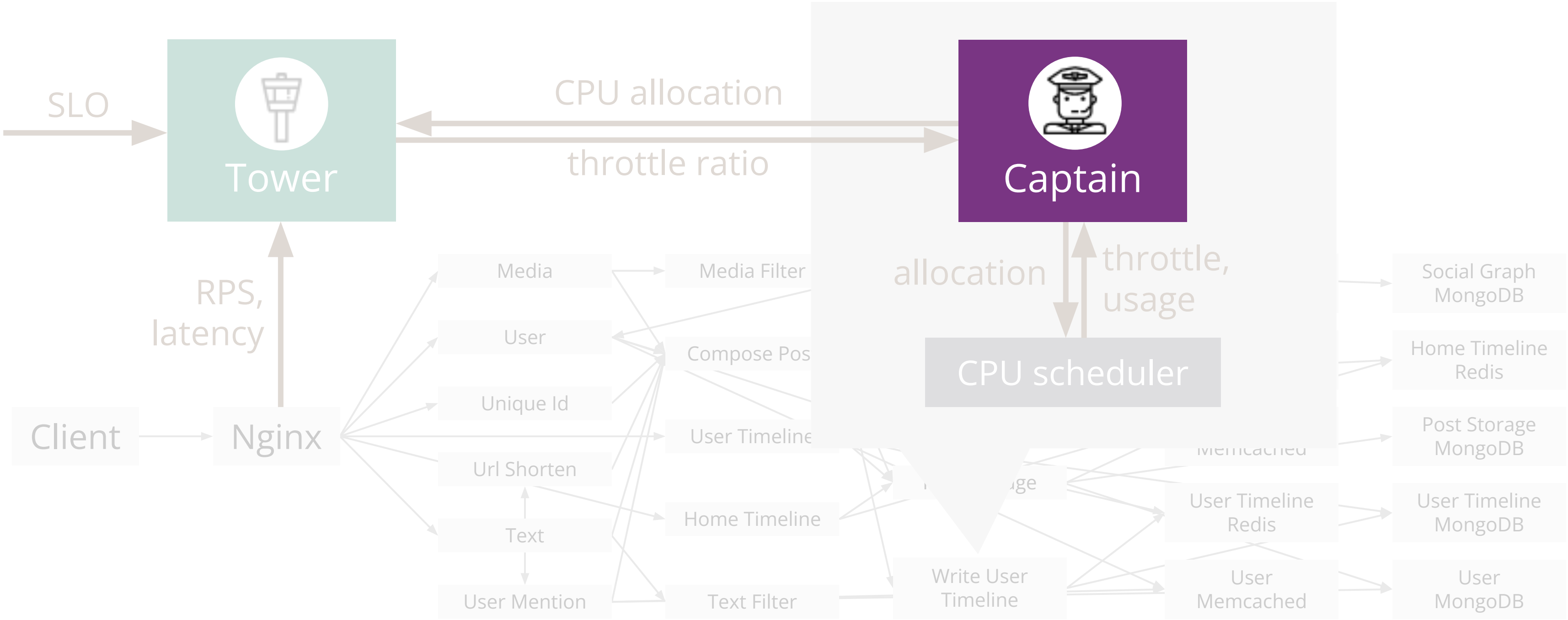


$$\text{throttle ratio} = \frac{\text{1 period}}{\text{3 periods}}$$

Throttle ratio has a higher correlation with latency

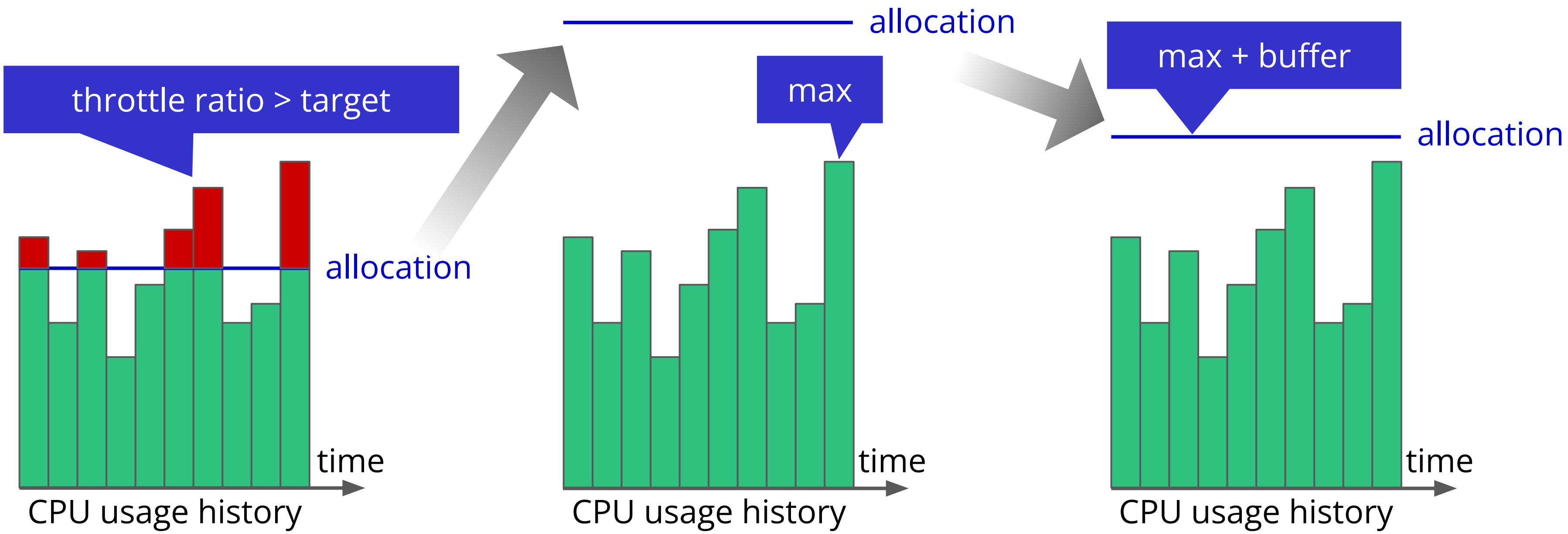


Service-level: fast and lightweight Captains

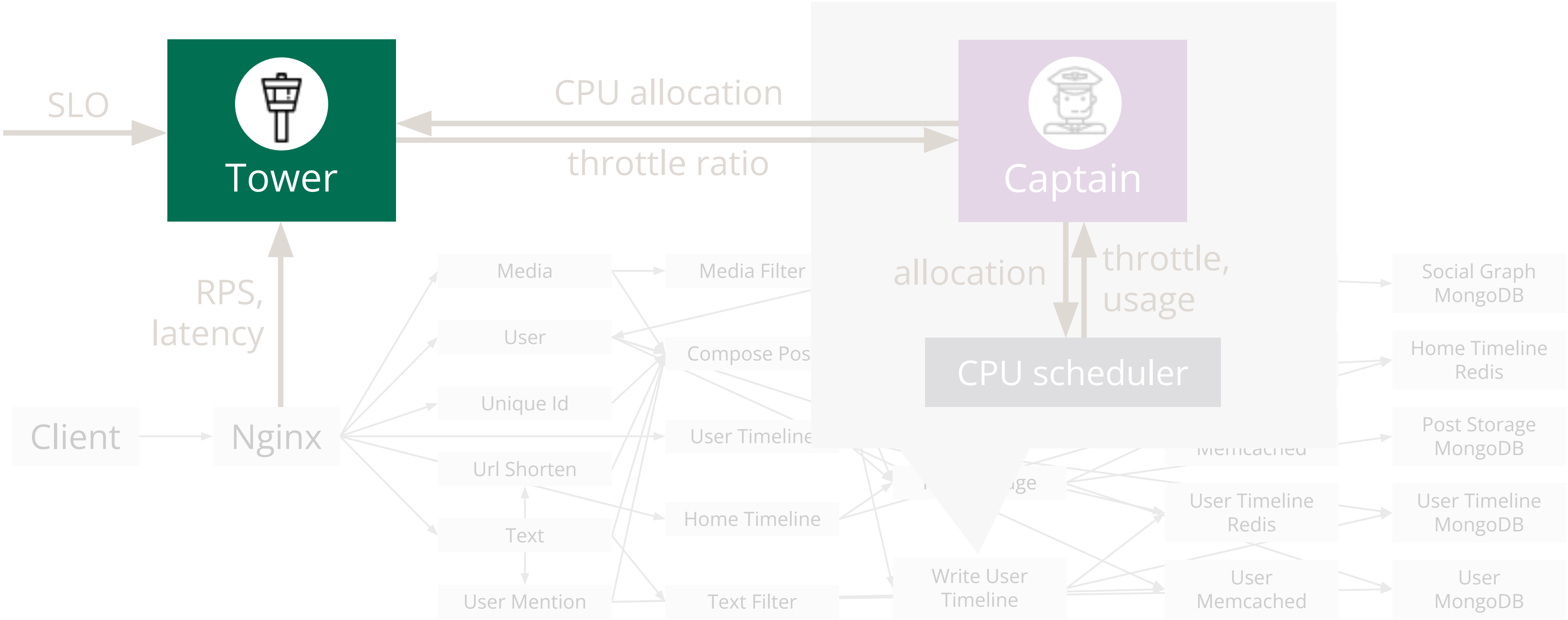


Service-level: fast and lightweight Captains

- Closed-loop control based on throttle ratio target
- Collect data every 100ms, adjust allocation every 1s

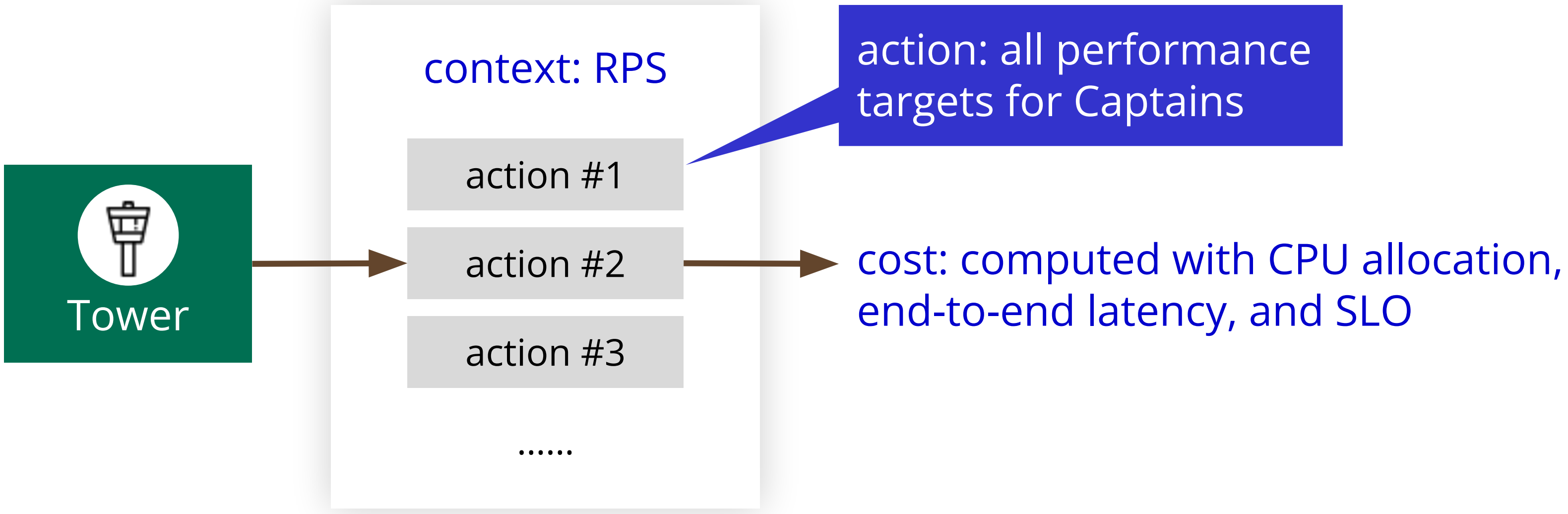


Application-level: online learning Tower



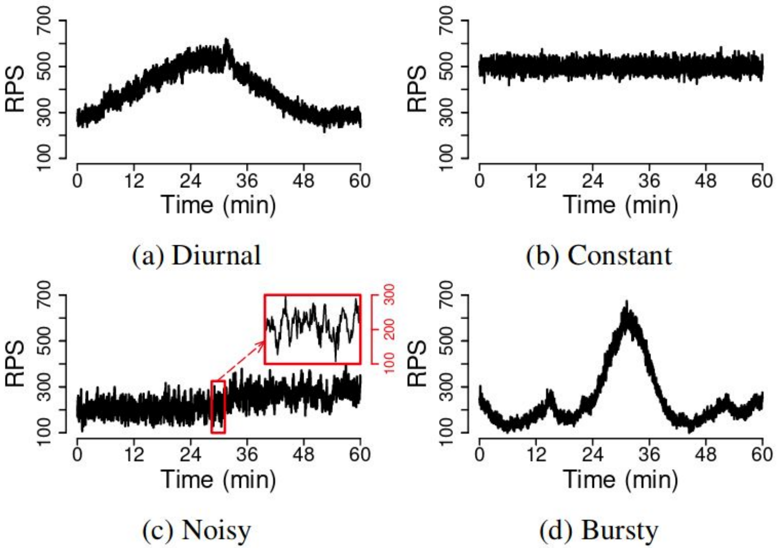
Application-level: online learning Tower

- Determine the best throttle targets for Captains to achieve
- Lightweight online learning: contextual bandit algorithm
  - One step per minute, each step runs in ~100ms

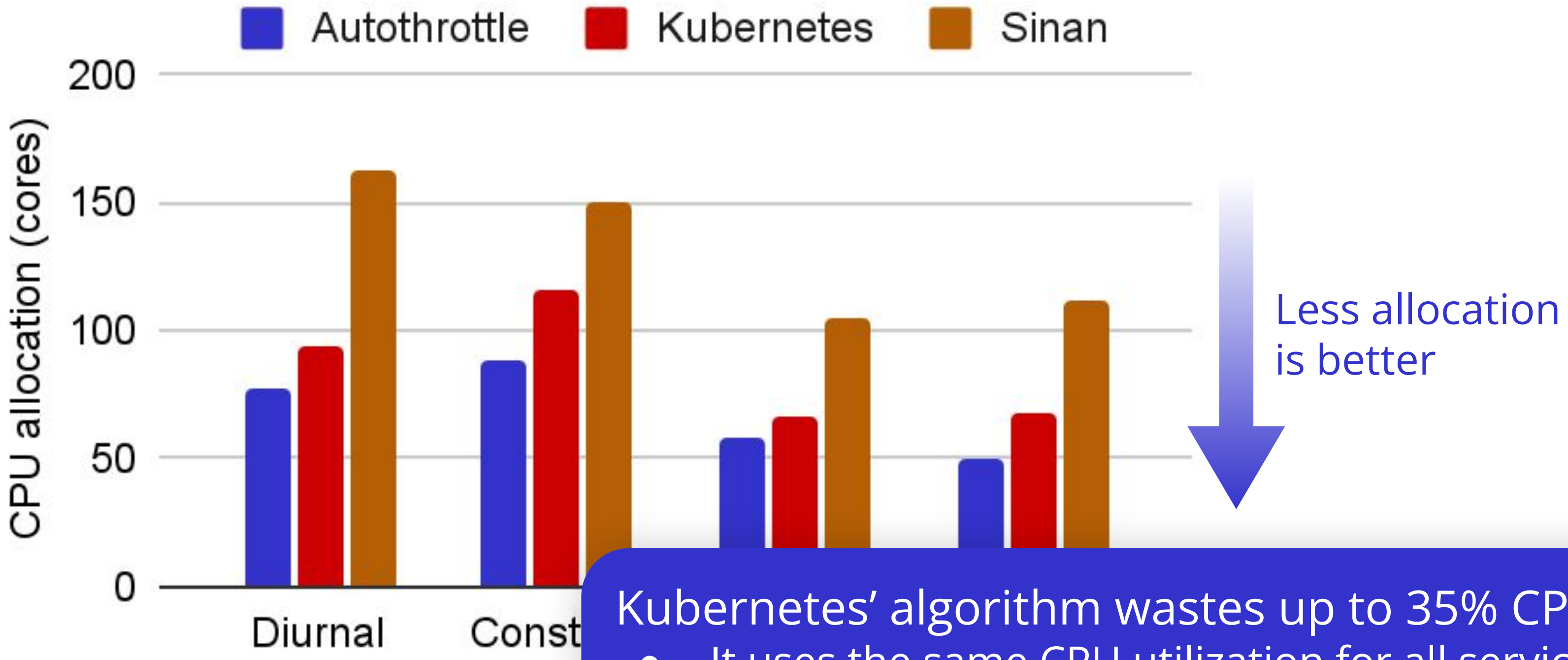


Evaluation methodology

- Testbed: 5 Azure VMs, 160 CPU cores in total
- 4 workload traces
  - with patterns commonly observed in production environments
  - e.g. Puffer’s streaming requests, Google’s cluster usage, and Twitter tweets
- 3 benchmark applications
  - Train-Ticket
  - Hotel-Reservation from DeathStarBench
  - **Social-Network** used in Sinan



Evaluation results



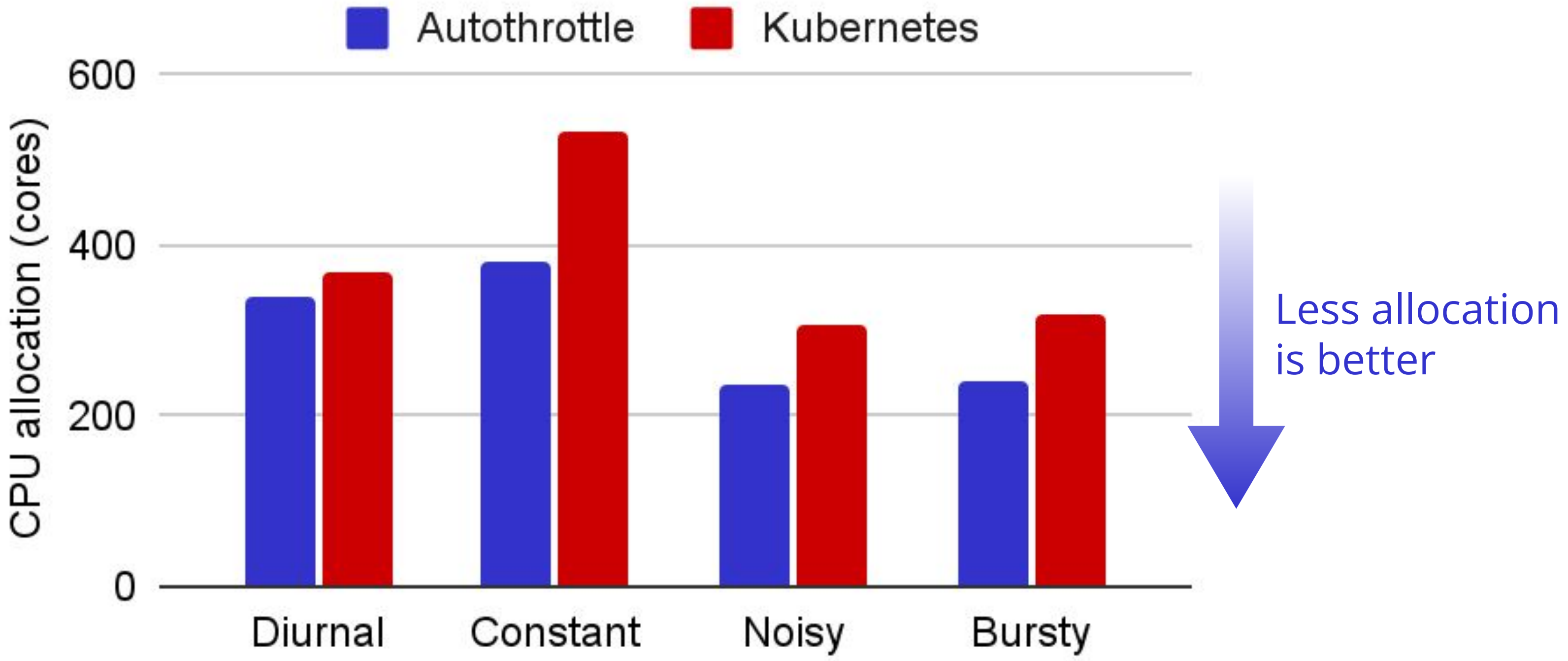
Kubernetes’ algorithm wastes up to 35% CPU

- It uses the same CPU utilization for all services

Sinan wastes even more CPU

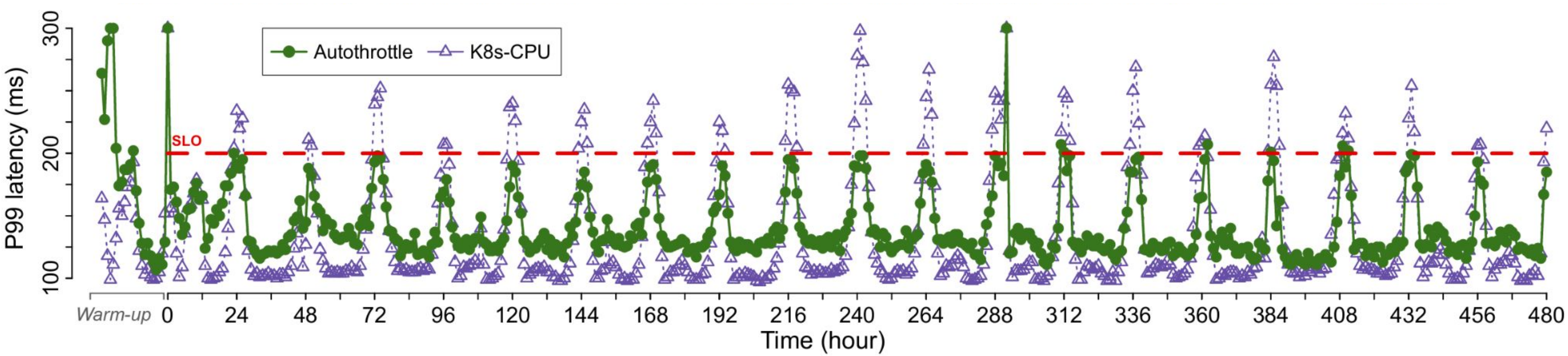
- Its search space is too large to explore

Large-scale evaluation on a 512-core cluster



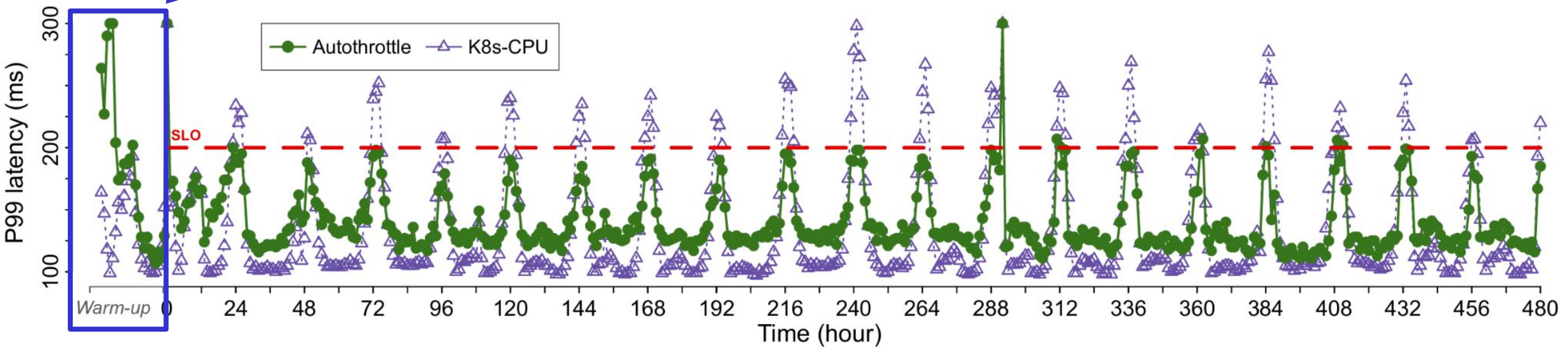
Kubernetes' algorithm wastes up to 39% CPU

A 21-day comparison



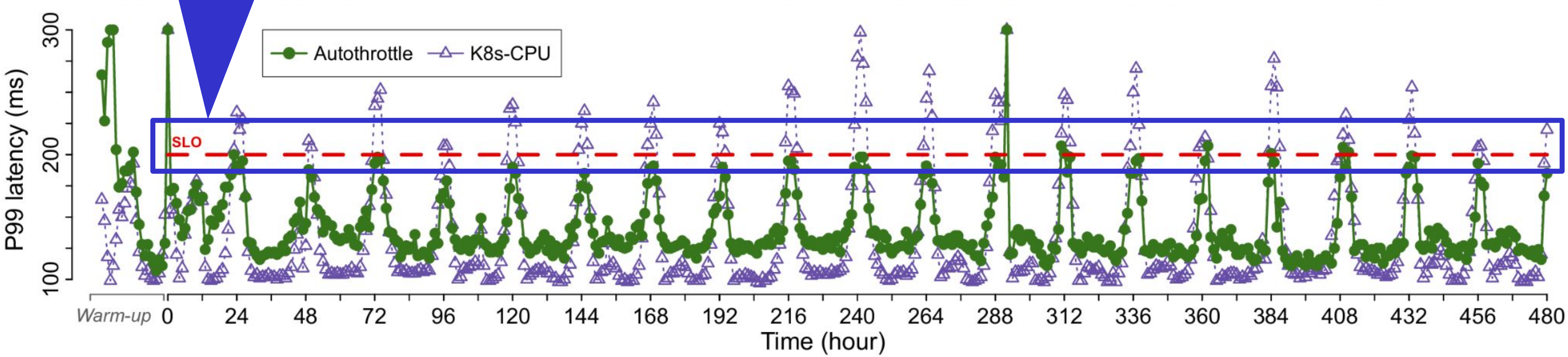
A 21-day comparison

Autothrottle: automatic exploration
K8s-CPU: manual parameter tuning

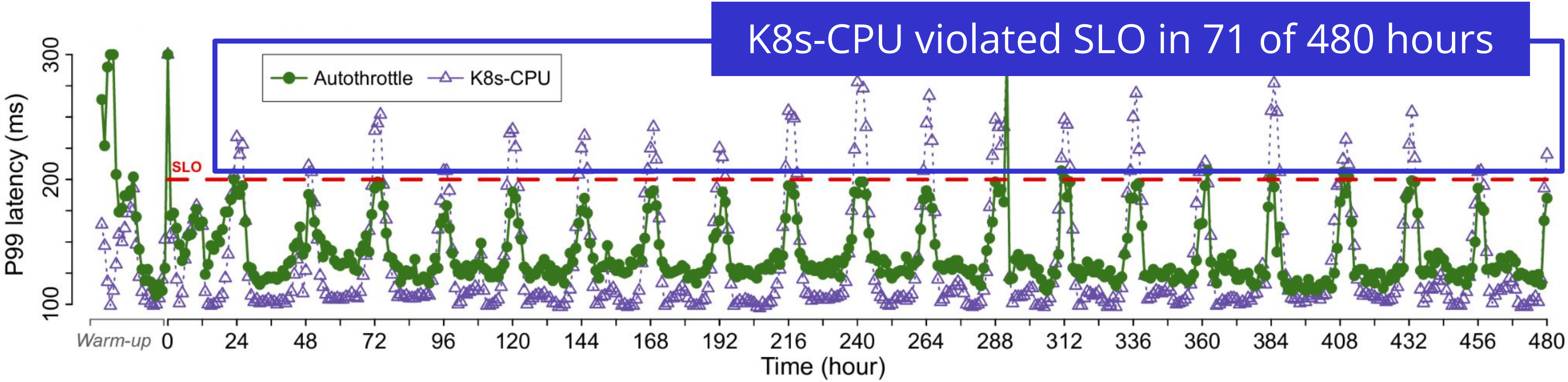


A 21-day comparison

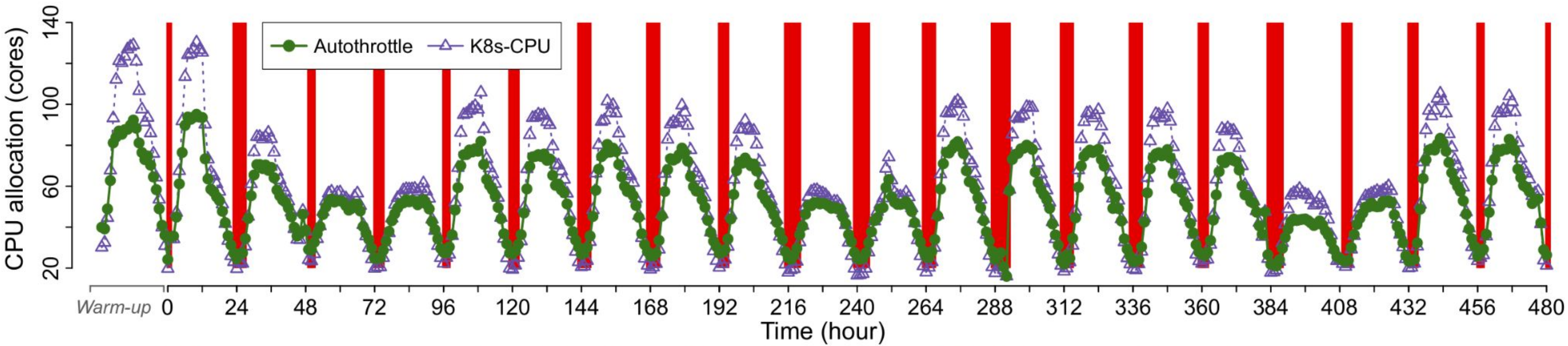
SLO = 200ms

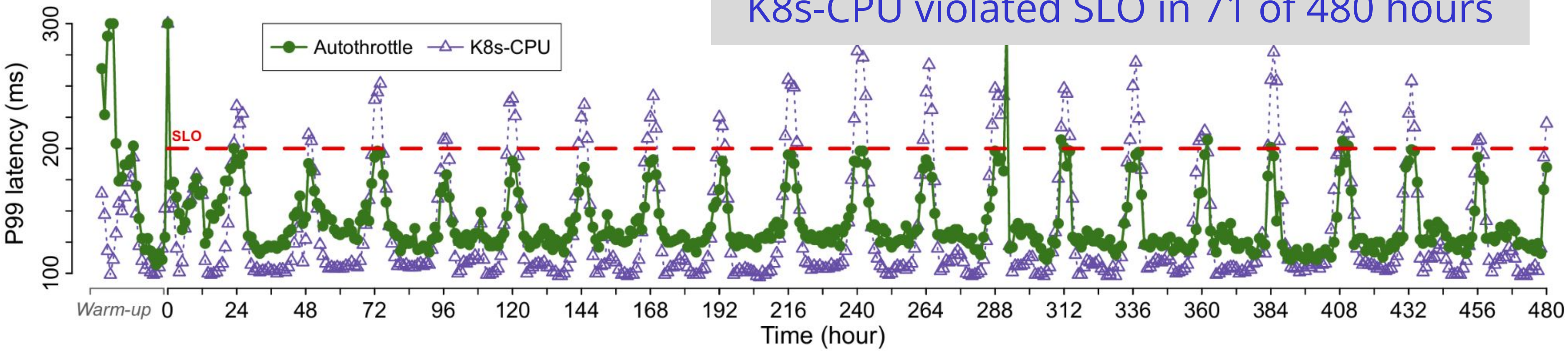


A 21-day comparison



A 21-day comparison

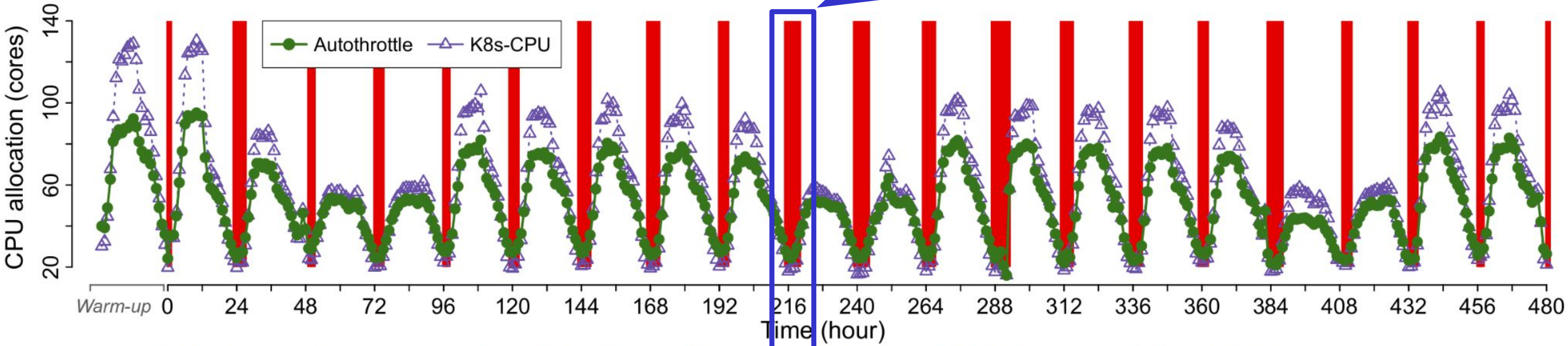




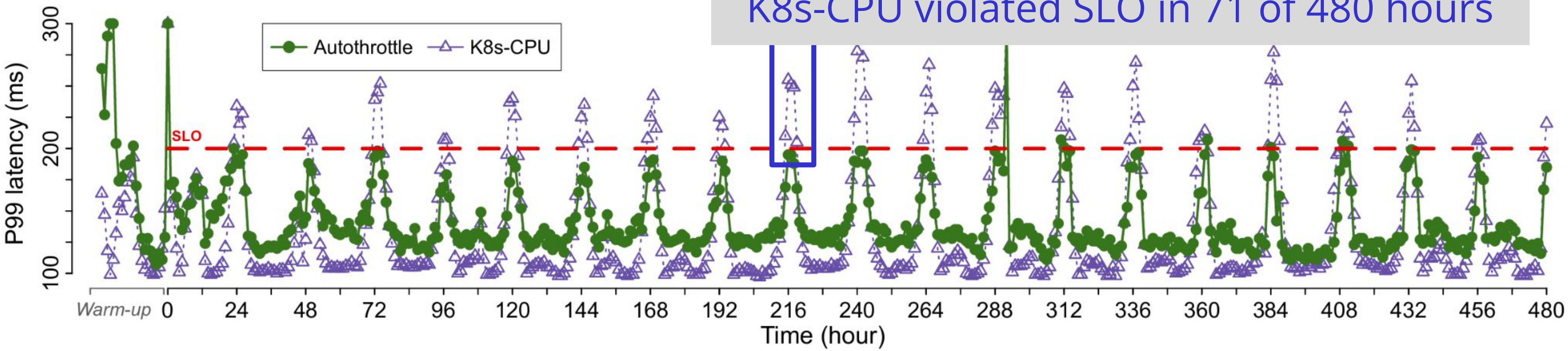
K8s-CPU violated SLO in 71 of 480 hours

A 21-day comparison

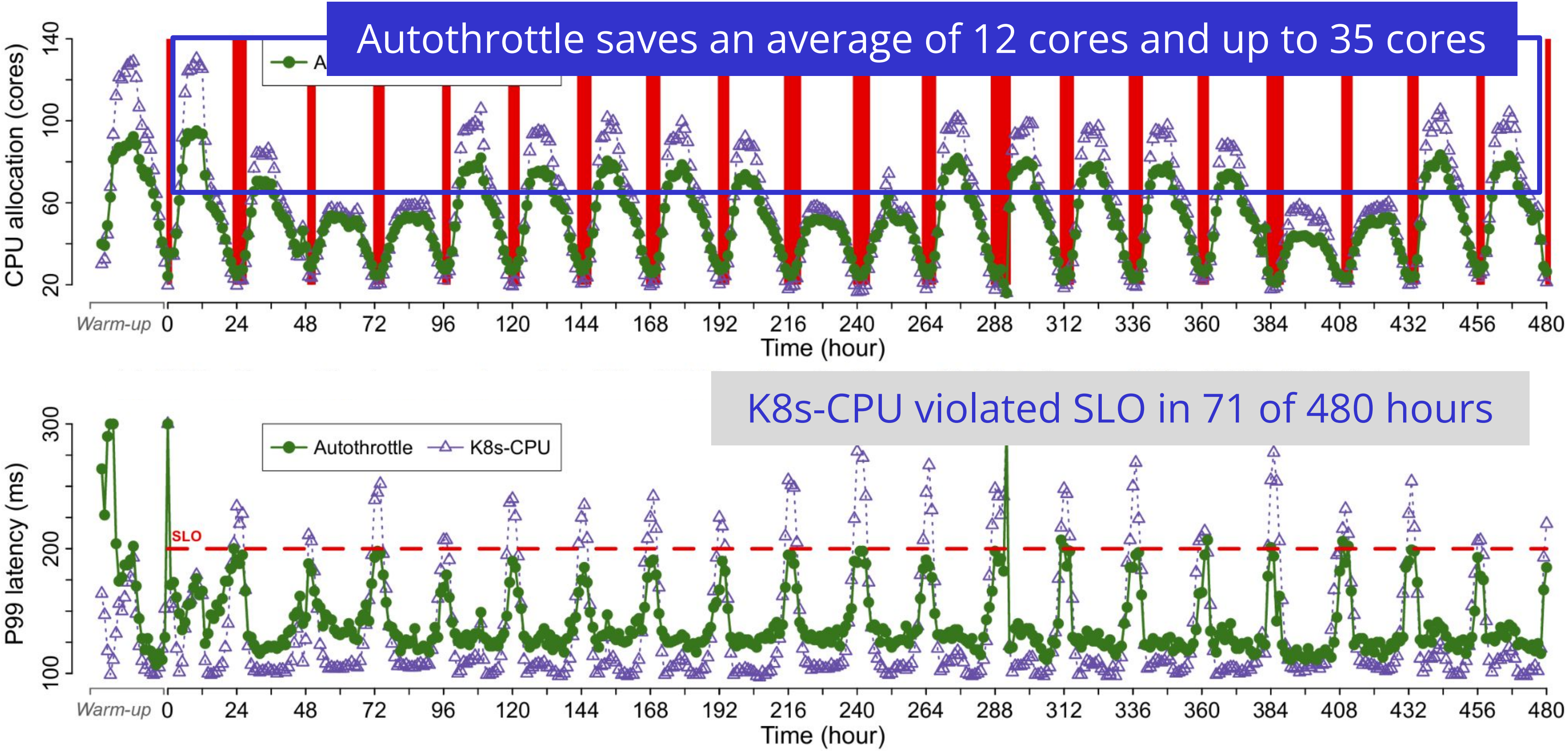
Red boxes mark K8s-CPU's SLO violations



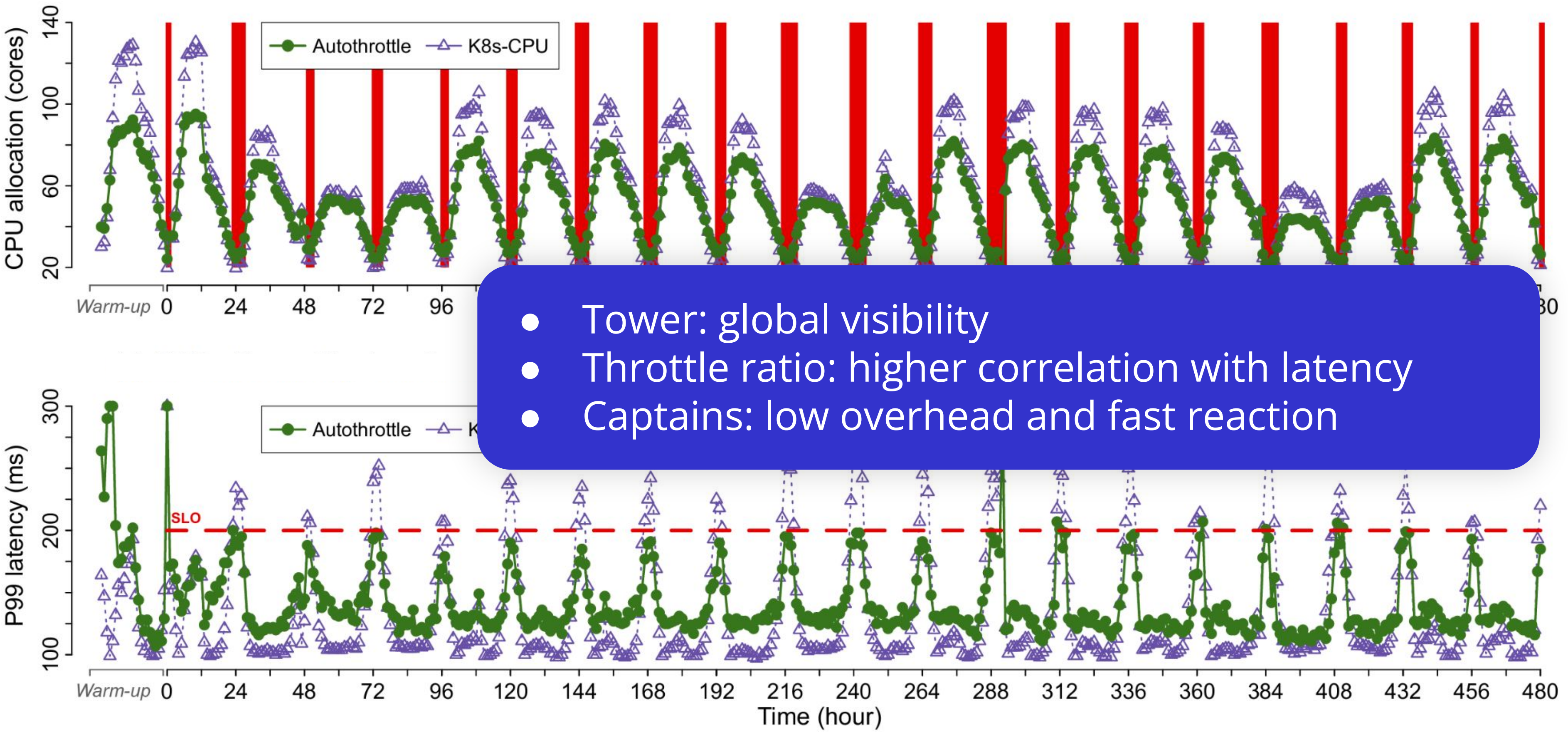
K8s-CPU violated SLO in 71 of 480 hours



A 21-day comparison



A 21-day comparison



- Tower: global visibility
- Throttle ratio: higher correlation with latency
- Captains: low overhead and fast reaction

Conclusion

- Autothrottle: a bi-level learning-assisted resource management framework for SLO-targeted microservices.
- Results show a CPU saving up to 26% while satisfying SLO
- Open-sourced at https://github.com/microsoft/autothrottle