PriorityMeister: Tail Latency QoS for Shared Networked Storage

Timothy Zhu, Alexey Tumanov, Michael A. Kozuch,
Mor Harchol-Balter, Gregory R. Ganger
SoCC 2014

Presenter: Yosub Shin

Motivation

- Internet companies want better tail latencies
- 99.9th or even 99.99th percentiles matter
- e.g. Displaying Facebook newsfeed: Requires~1000 RPC calls. If 999 calls return in 50ms and one call takes 3s, the end-to-end response time = 3s.

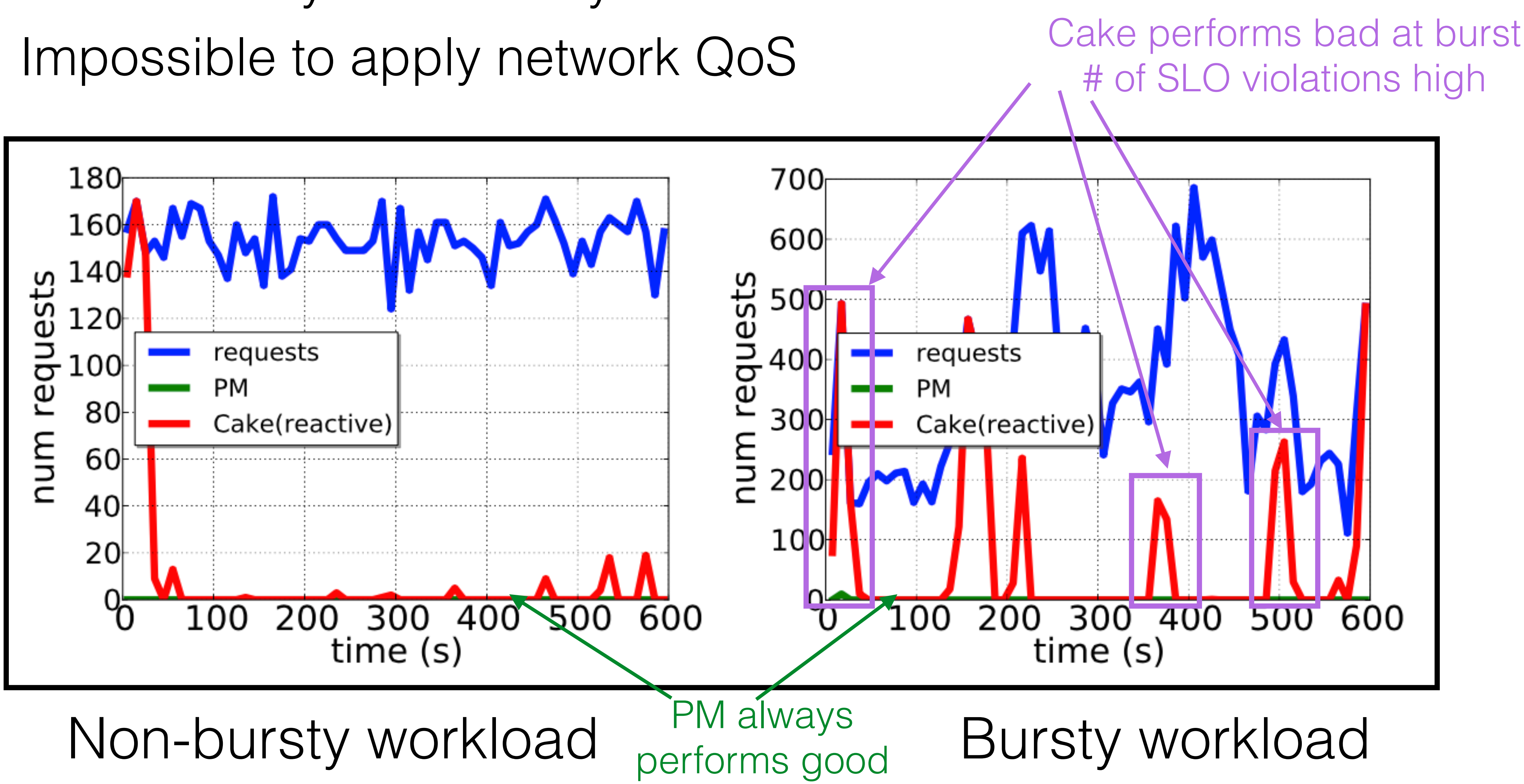
Challenges

- Bursty workloads cause queuing
- End-to-end latency is affected by multiple stages
  - Outgoing network packets
  - Storage requests
  - Incoming network packets

Prior Works

Cake

- Cake: Reactive feedback-control built for tail latency
- Bad for bursty workloads
- Handles only one latency-sensitive workload
- Impossible to apply network QoS



Enter PriorityMeister

- Proactive QoS(Quality of Service) system
- Achieves end-to-end latency SLO(Service Level Objective)

  I want Workload A to respond in 30ms,
  Workload B to respond in 50ms.
- Multi-tenant, multi-resource
- How?
  - **Priority**
  - **Rate Limiting**

Contributions of This Paper

- Algorithm that automatically determines priority and rate limits for each workload at each stage
- Built a real QoS system consisting of network and storage which outperforms existing approaches
- Robust to mis-estimation of storage or network performance and workload mis-behavior

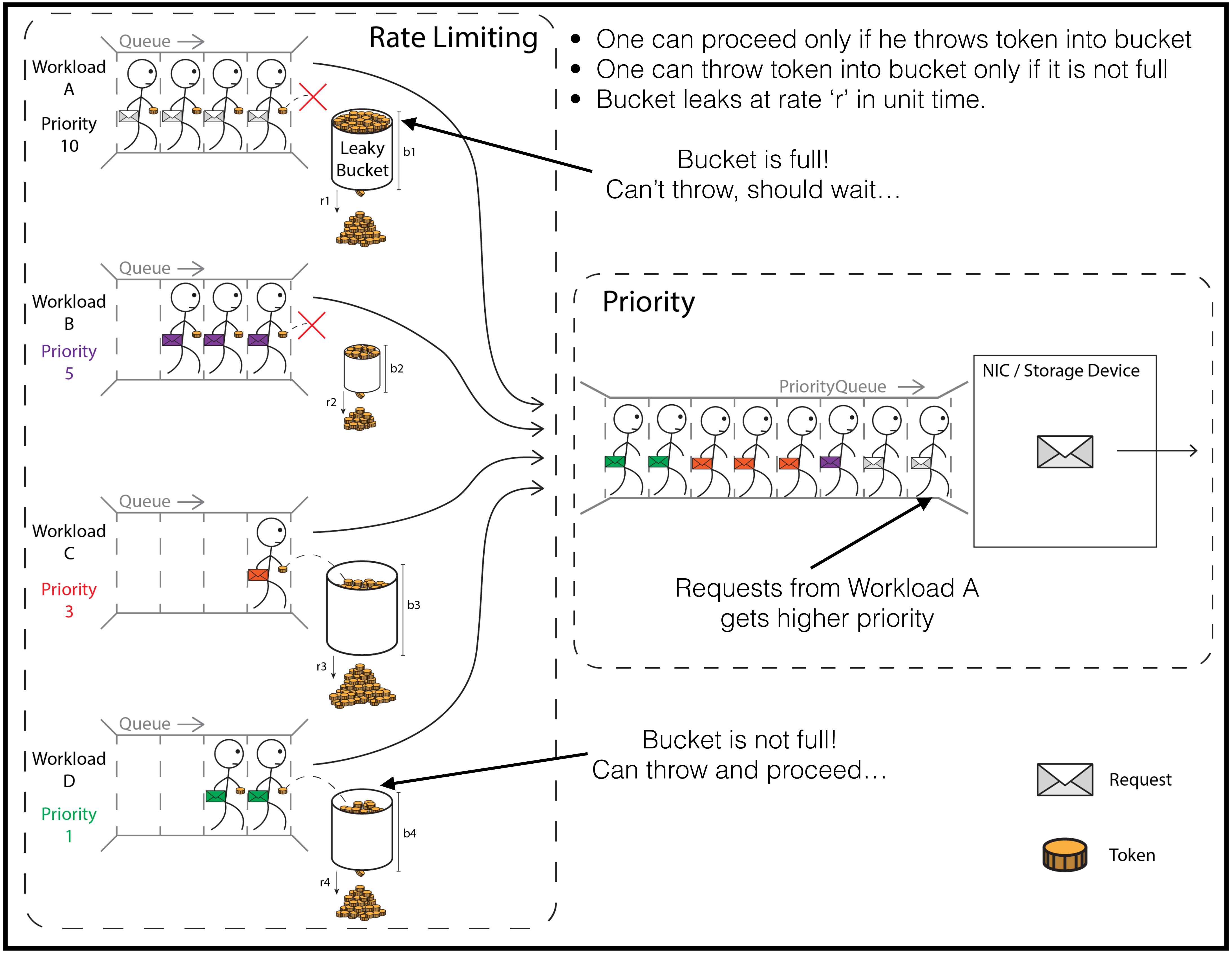
Intuition

Priority & Rate Limiting

e.g. I want Workload A to respond in 30ms,
Workload B to respond in 50ms

- **Priority**: In order to meet tight latency requirements
    - NOT to be confused with importance to the user
- **Rate Limiting**: To prevent starvation of lower priority workloads

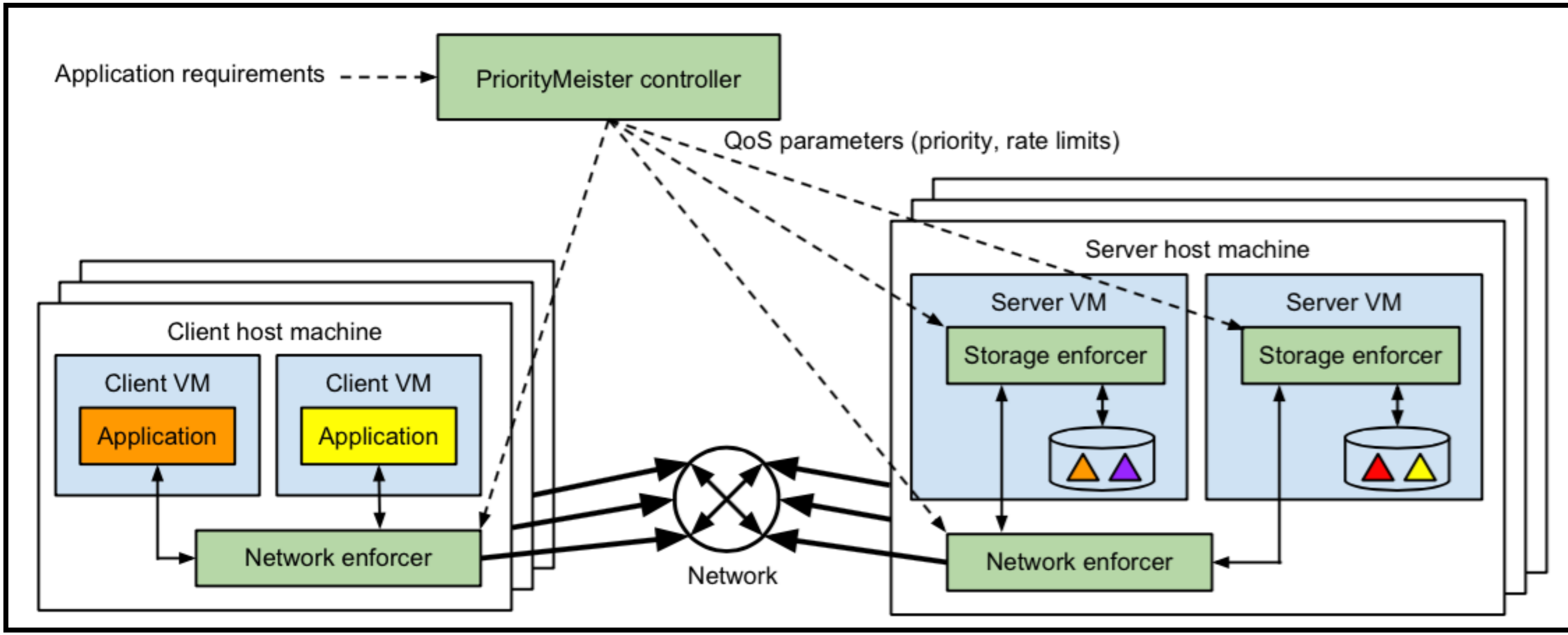
Intuition



Rate Limiting
Queue
Workload A
Priority 10
Leaky Bucket
b1
r1
Workload B
Priority 5
b2
r2
Workload C
Priority 3
b3
r3
Workload D
Priority 1
b4
r4
• One can proceed only if he throws token into bucket
• One can throw token into bucket only if it is not full
• Bucket leaks at rate 'r' in unit time.
Bucket is full!
Can't throw, should wait…
Priority
PriorityQueue
NIC / Storage Device
Requests from Workload A
gets higher priority
Bucket is not full!
Can throw and proceed…
Request
Token

Architecture Overview

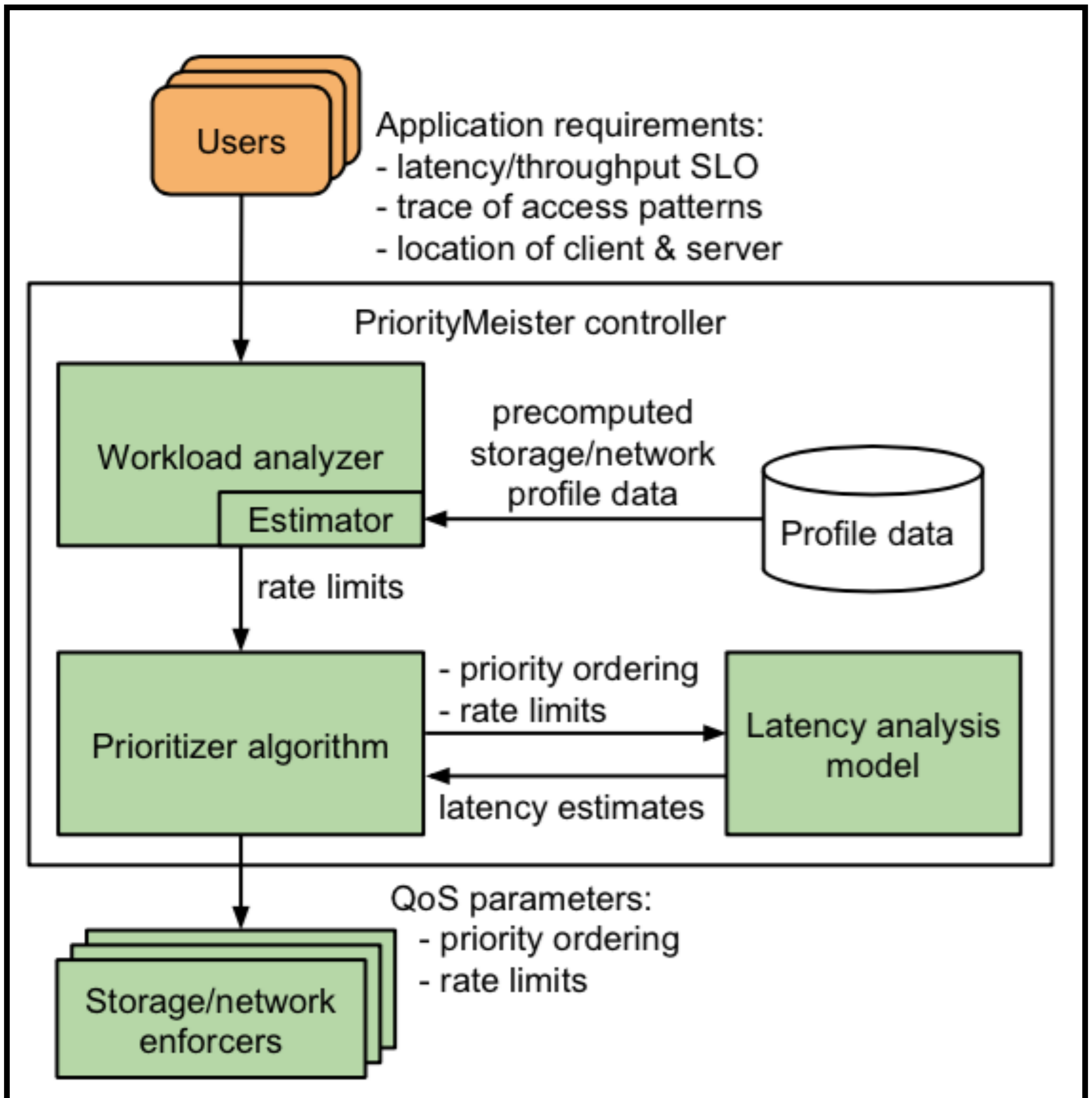
Components



- Queue at each component of machine(Network, Storage)
- Each stage has independent priorities and rate limits

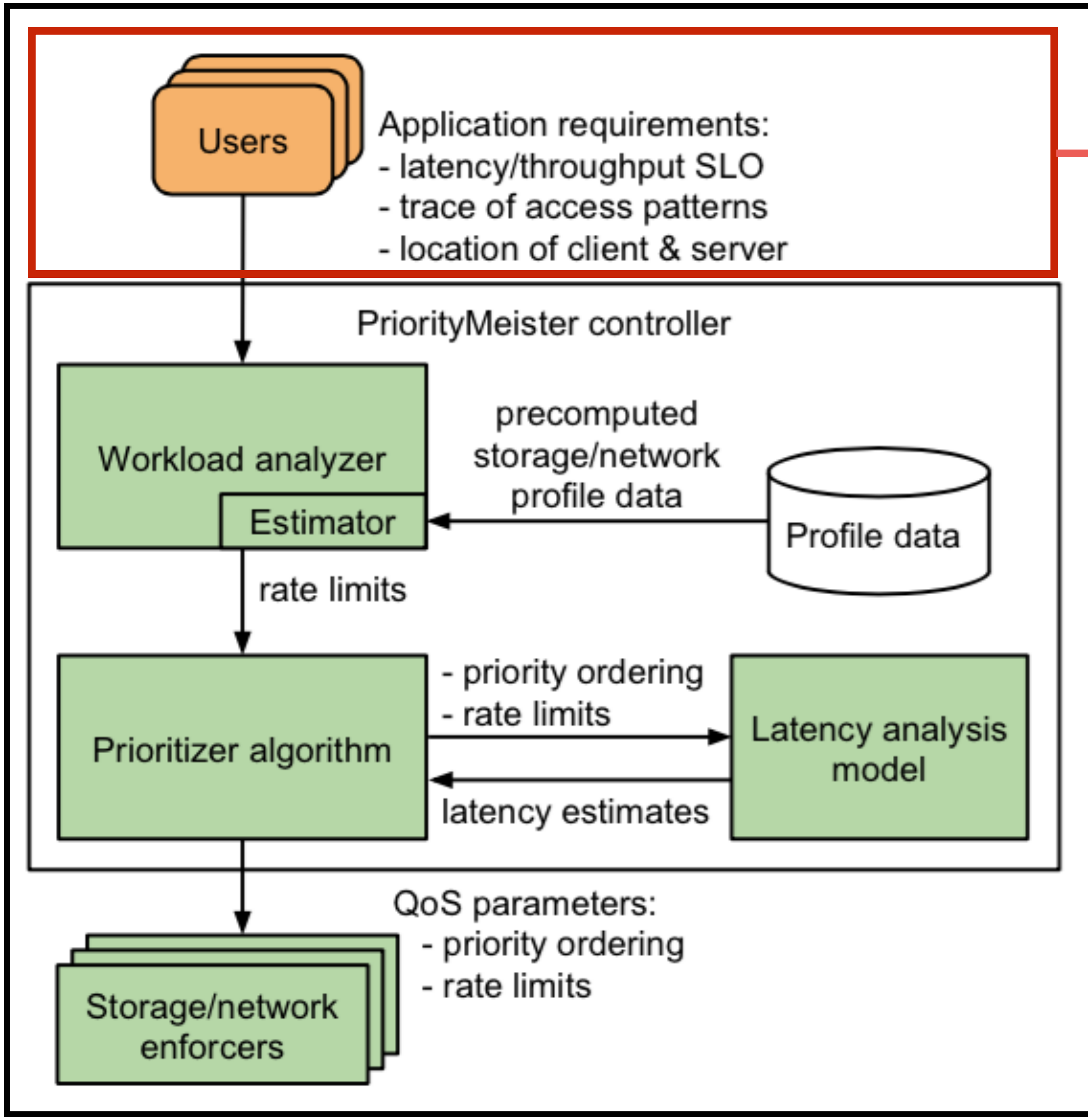
Architecture Overview

Workflow



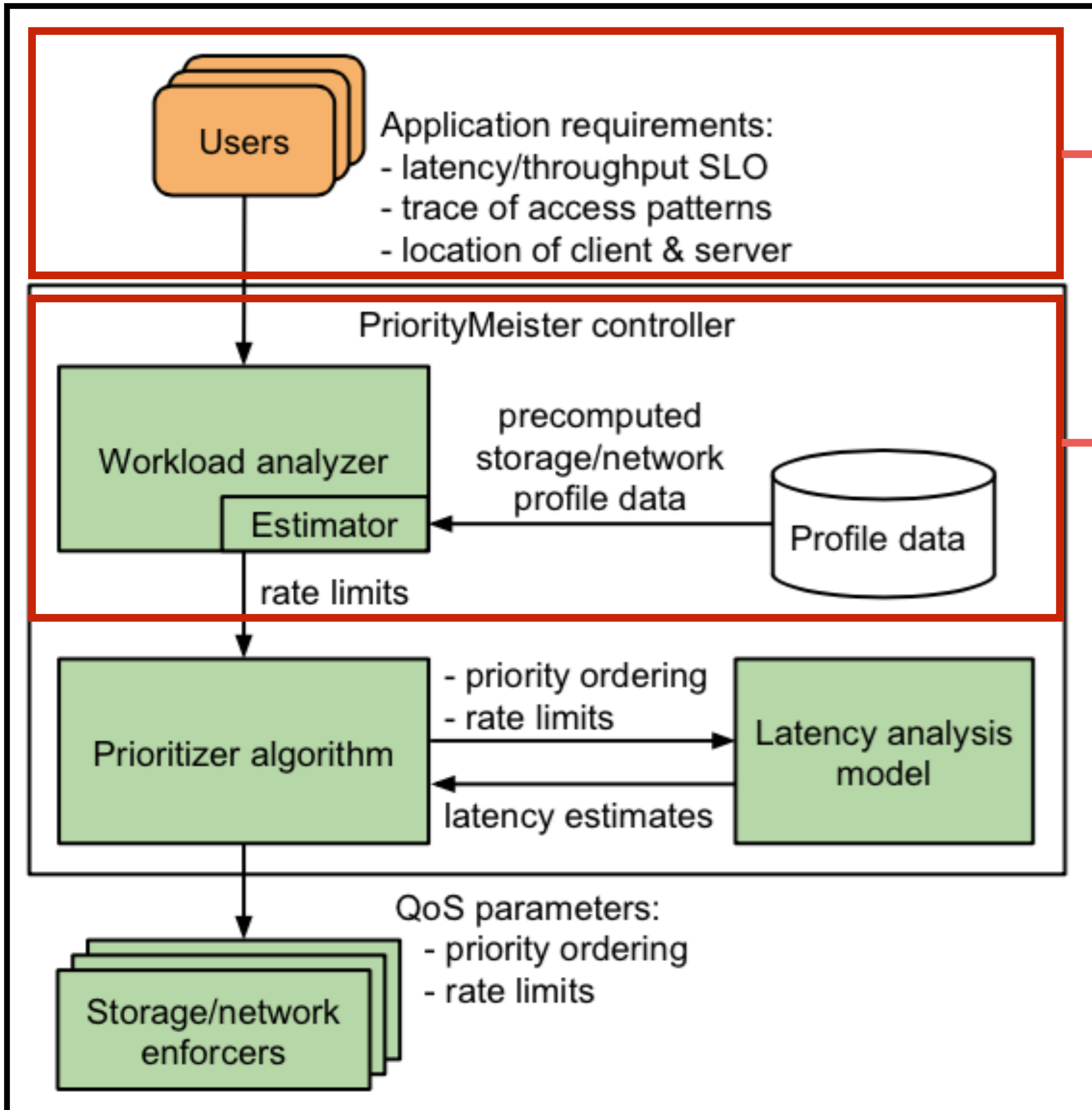
Architecture Overview

Workflow



User initiates system with SLO requirements

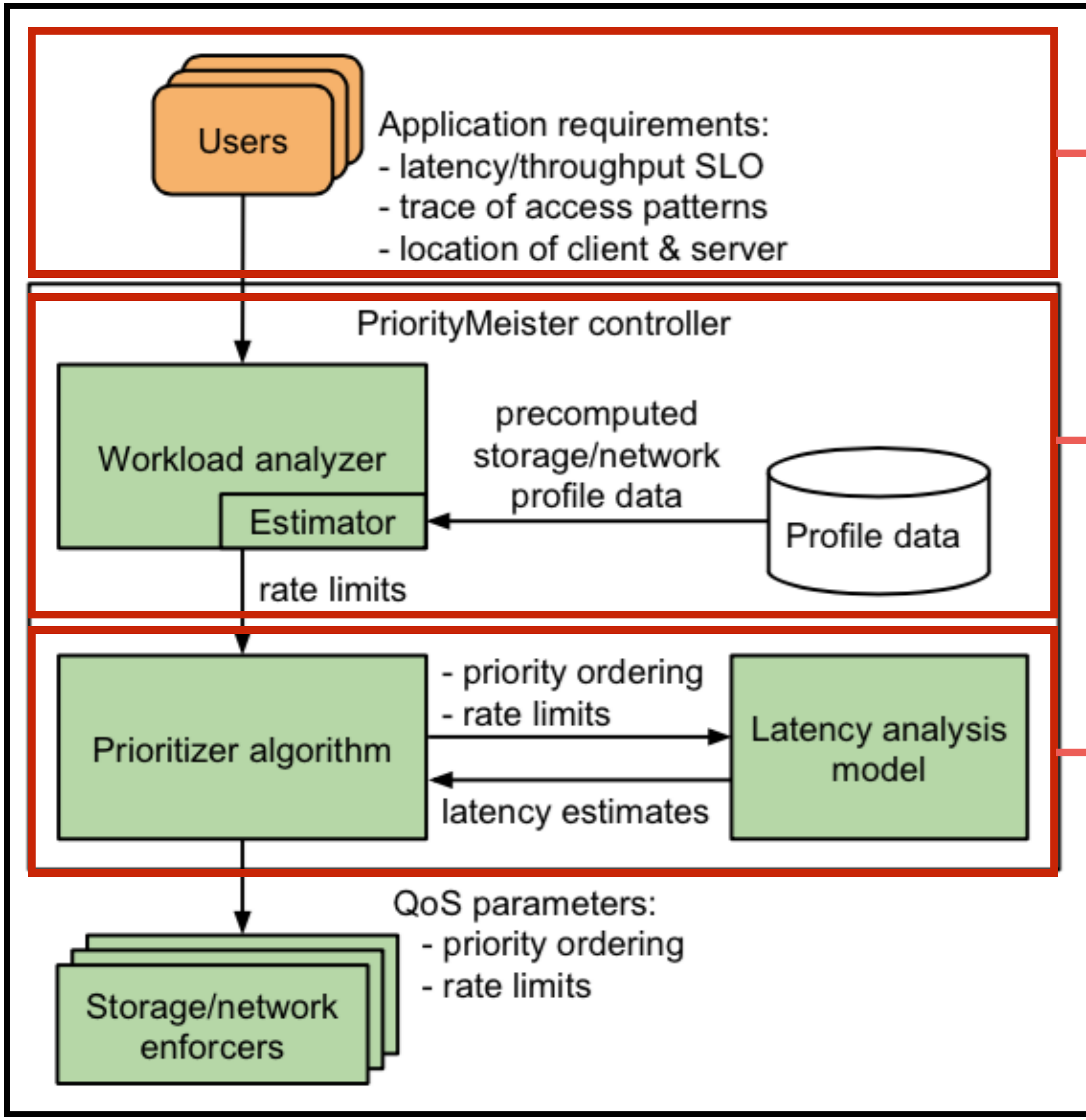
Workflow



User initiates system with SLO requirements

Generate rate limits

Workflow



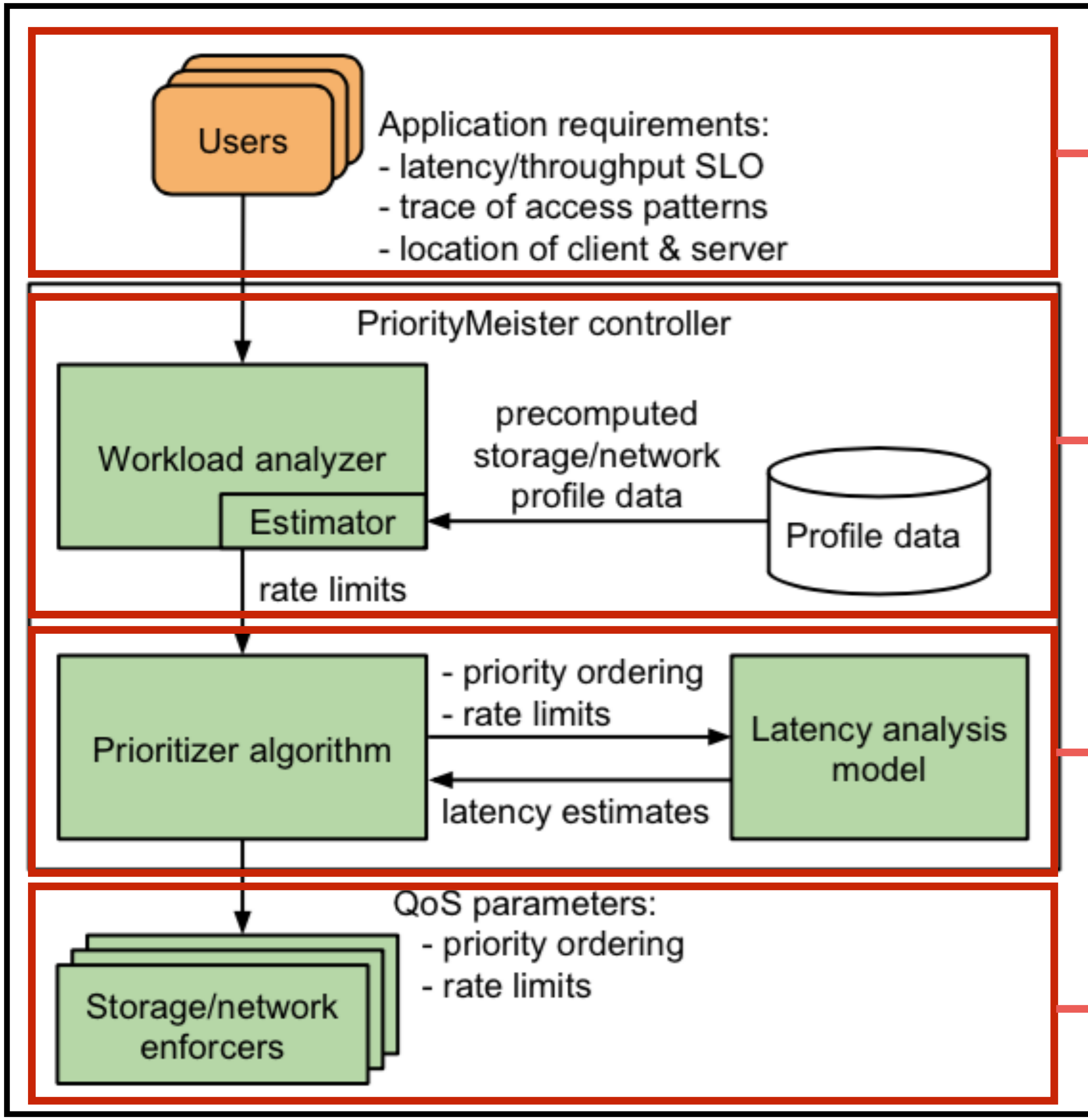
User initiates system with SLO requirements

Generate rate limits

Determine priority orderings

Architecture Overview

Workflow



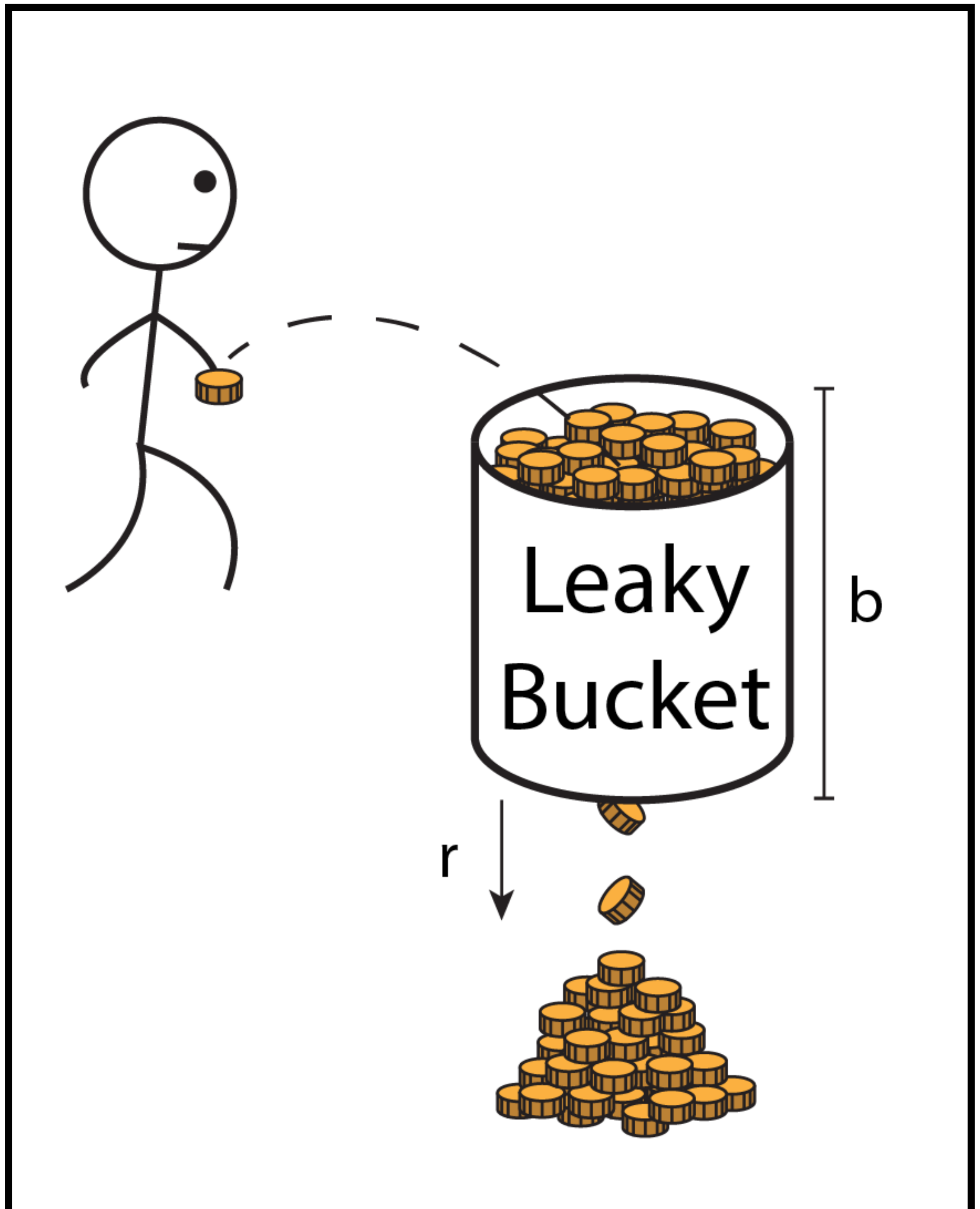
User initiates system with SLO requirements

Generate rate limits

Determine priority orderings

Send priorities and rate limits to enforcers

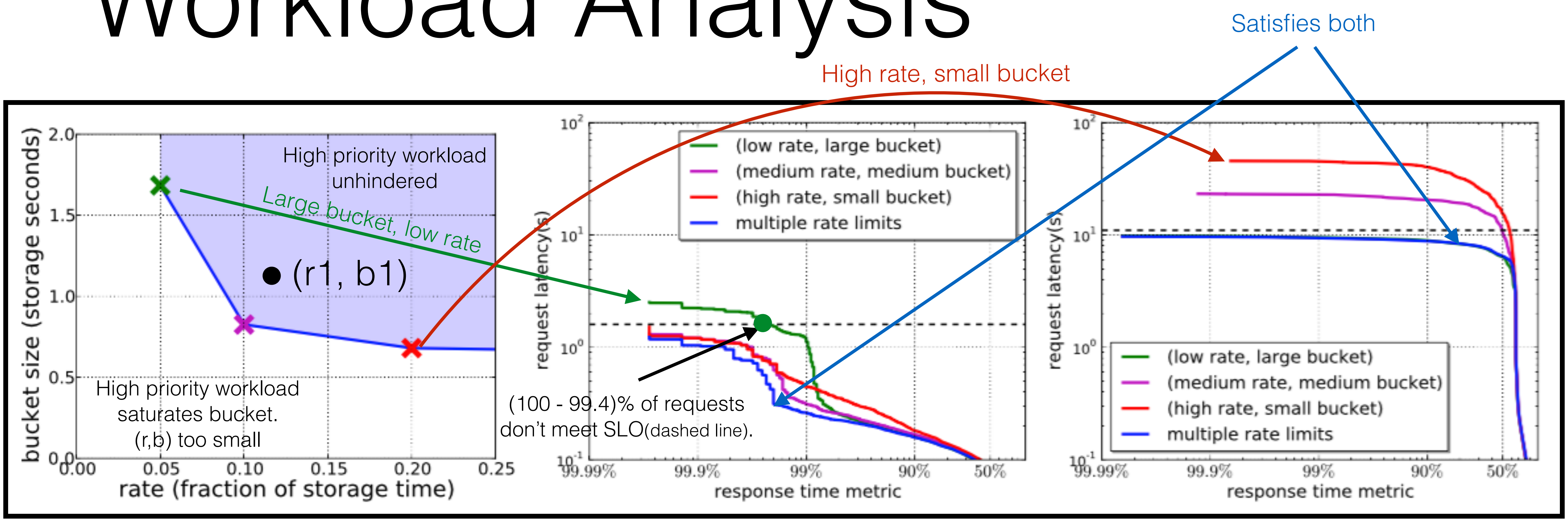
Leaky Token Bucket Model



- Token(s) == size of request
  Storage: Amount of storage time required
  Network: Number of transmitted bytes
- (r, b) pair determines bucket's behavior
- r: leaking rate
- b: bucket size(in #tokens)
- Throwing a new token into bucket only allowed when bucket not full

How to Limit Rates?

Workload Analysis

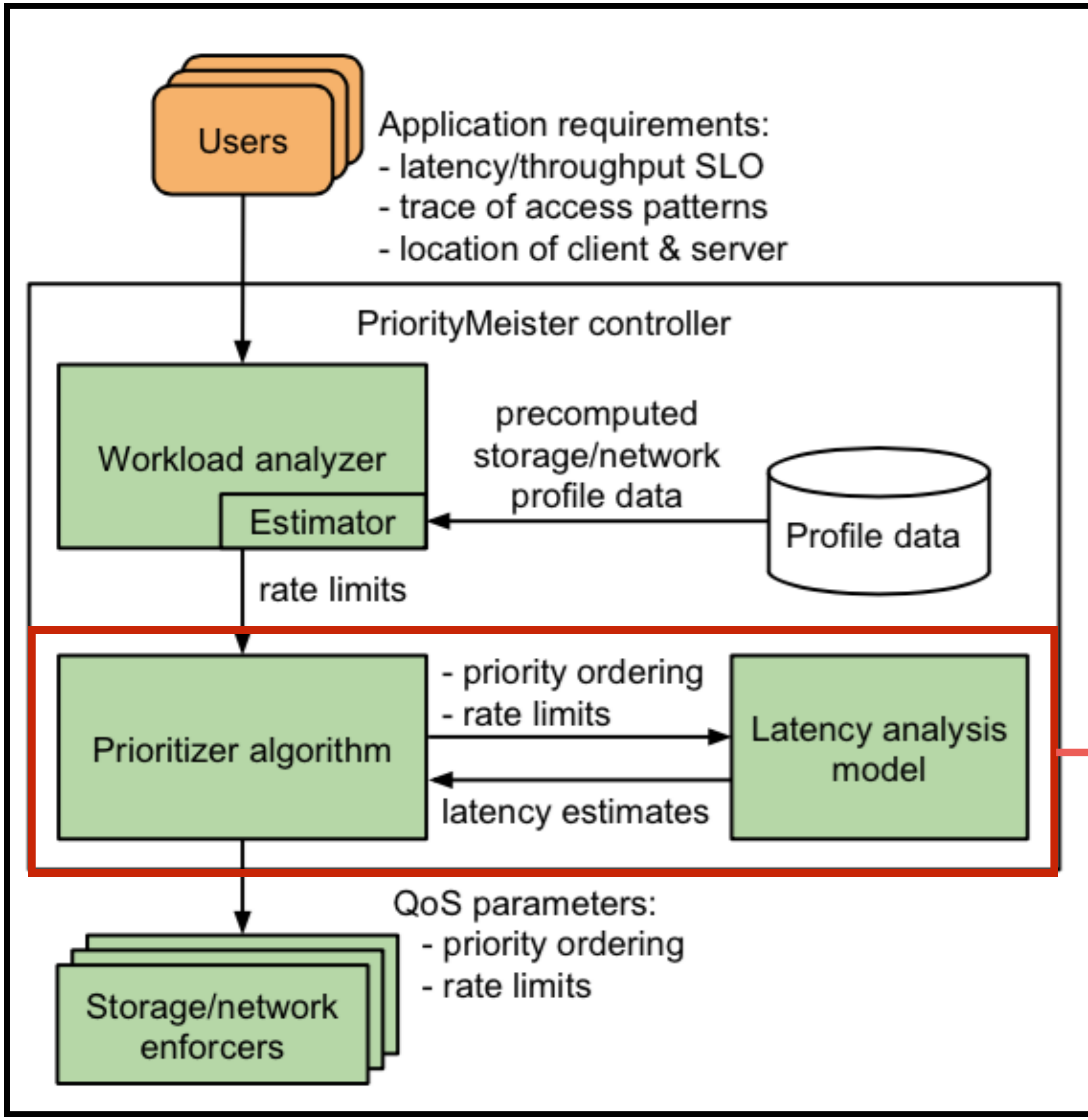


rate limit pairs of high priority workload W_A | latency of medium priority workload W_B | latency of low priority workload W_C

- Assume highest priority for W_A, calculate (r, b) pairs big enough for the $\text{trace}_{\text{given}}$ to run under $\text{SLO}_{\text{given}}$
- Want to decide smallest (r, b) pair s.t. lower priority workloads are allowed to run under their SLOs
- Larger bucket size(b) leads to higher tail latency in medium priority workload W_B
- Larger rate(r) leads to higher tail latency in low priority workload W_C
- **Key Idea:** Use multiple (r, b) pairs on the blue line and allow throwing tokens into bucket only when tokens can be added to all (r, b) buckets.

How to determine priorities?

Workflow



Determine
priority orderings

How to determine priorities?

Prioritizer Algorithm

- Input: workload SLOs, rate limits
- Output: **priorities for each stage at each workload** s.t. each workload's estimated worst-case latency is less than SLO
  via Latency Analysis Model
- |(# workloads)|!^|(# stages)| possibilities: too large ☹
- Polynomial search possible(w/ greedy algorithm)! ☺

1. Assign lowest priority workload first!(If workload can still satisfy SLO)
2. For unassigned workload w/ lowest violation:
   (estimated latency) - (SLO)
   - For stage w/ lowest latency, assign lowest priority
     (Intuition: Take best performing workload/stage, assign lowest priority s.t. worst-case latency is improved)

How to estimate worst-case latencies given priorities?

Latency Analysis Model

- Input: priorities assignment, rate limits
- Output: **worst-case latencies for each stage at each workload**
- $\alpha(t)$: max. # bytes that arrive in any period of time t
- $\beta(t)$: min. # bytes serviced in any period of time t
- Worst-case latency: max. horizontal dist. b/w α and β
- $\alpha_w(t) = \min_i(r_i * t + b_i)$ (fastest rate at which rate limiter allows requests through)
- $\beta_w(t)$: Calculated with Linear Programming
  (time, flow, rate limit, and work conservation constraints)

Experiments

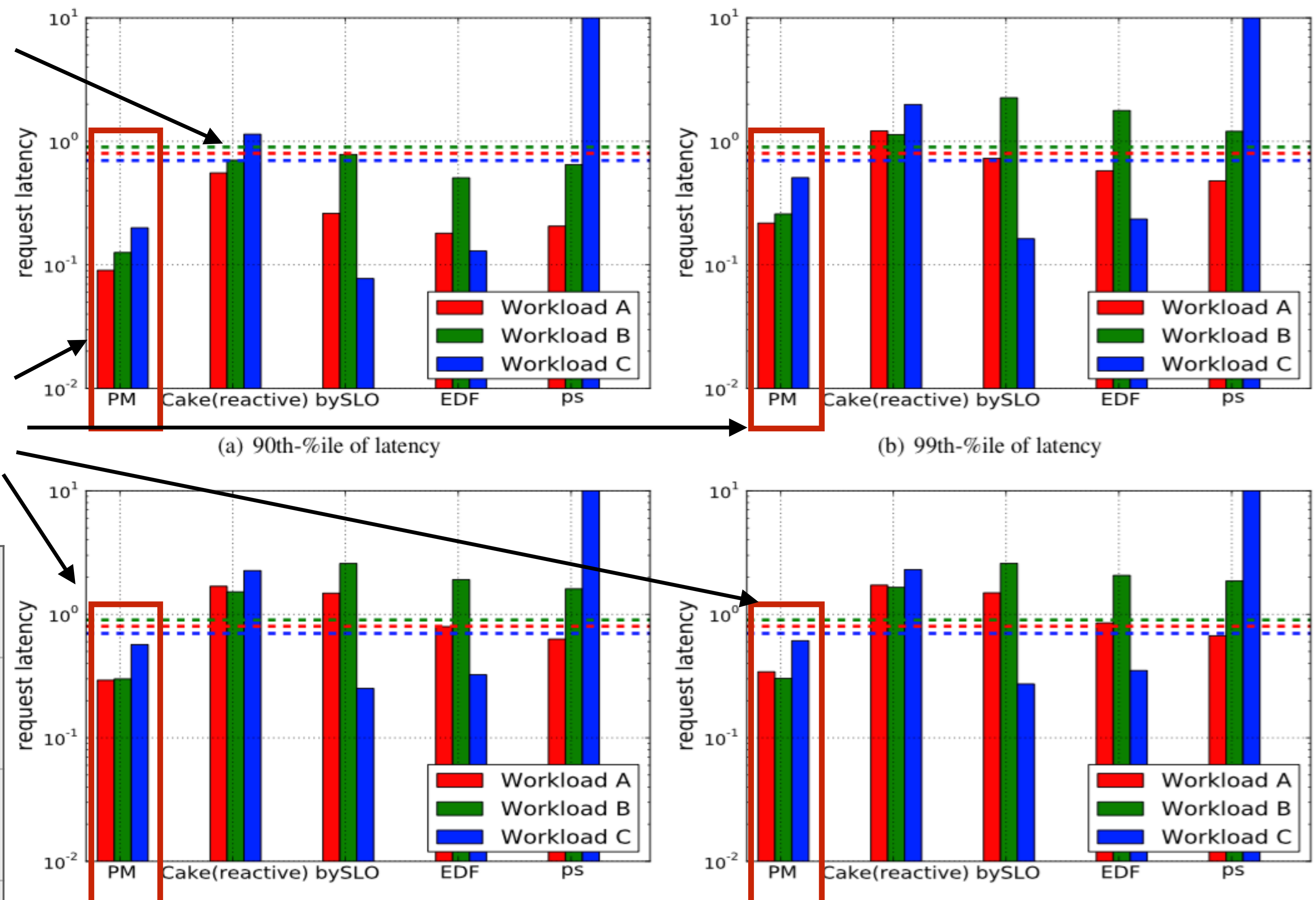
- Tail latency performance
- Latency under bursty workloads
- Mis-behaving workloads
- Network bottlenecked workloads
- Latency under estimator inaccuracy
- Latency under varying SLO permutations

Experiments

Tail Latency Performance(1/2)

SLO Lines

Only PriorityMeister always satisfies SLO



(a) 90th-%ile of latency

(b) 99th-%ile of latency

(c) 99.9th-%ile of latency

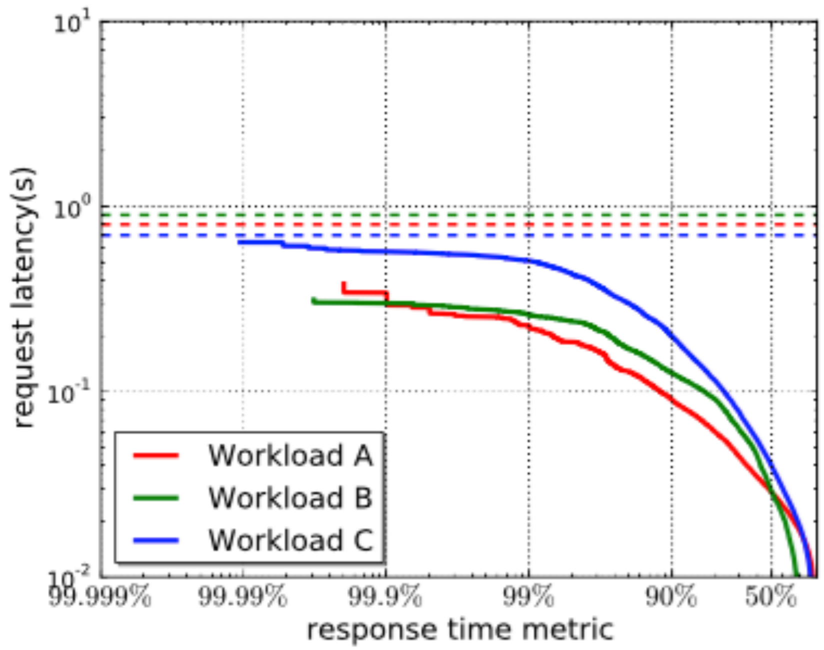
(d) 99.99th-%ile of latency

PM	PriorityMeister
Cake	Reactive Control
bySLO	Priority for lower SLO
EDF	Earliest Deadline First
PS	Proportional Sharing

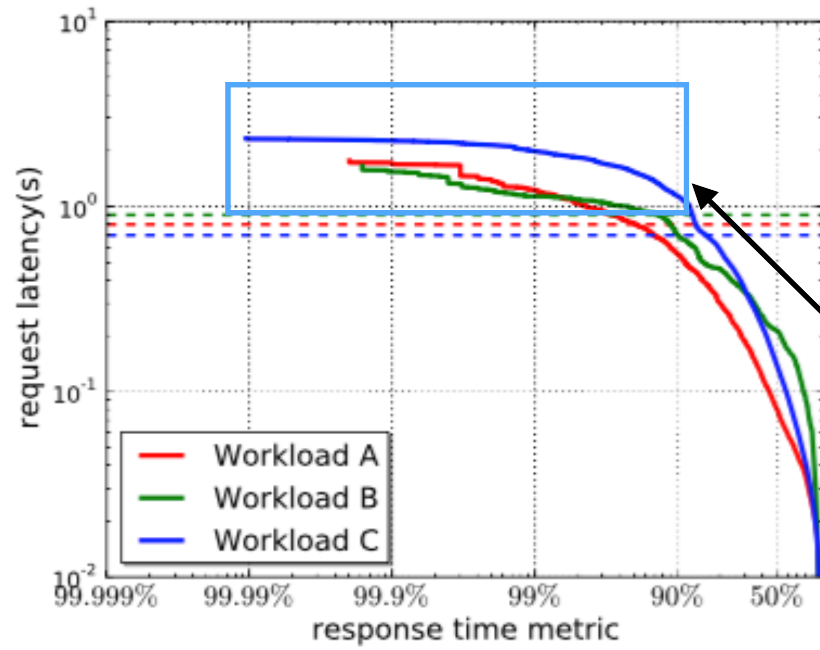
Workload	Workload A	Workload B	Workload C
Trace	Display Ad	MSN Storage	LiveMaps

Experiments

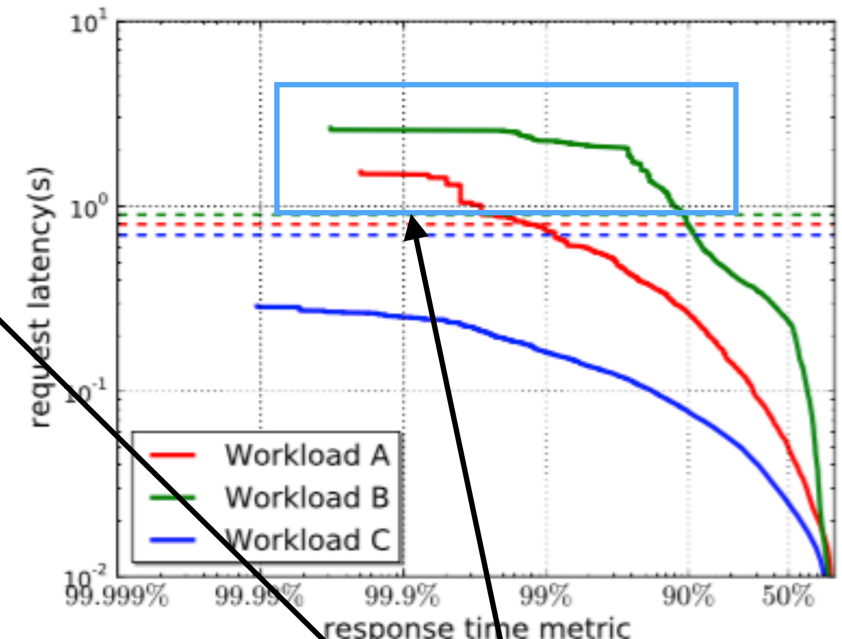
Tail Latency Performance(2/2)



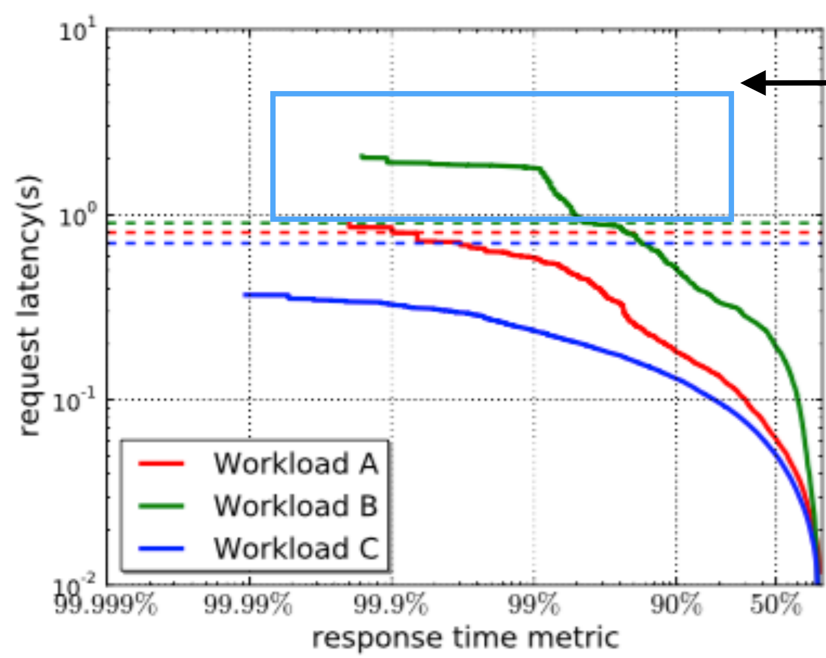
(a) PM: all workloads' SLOs satisfied



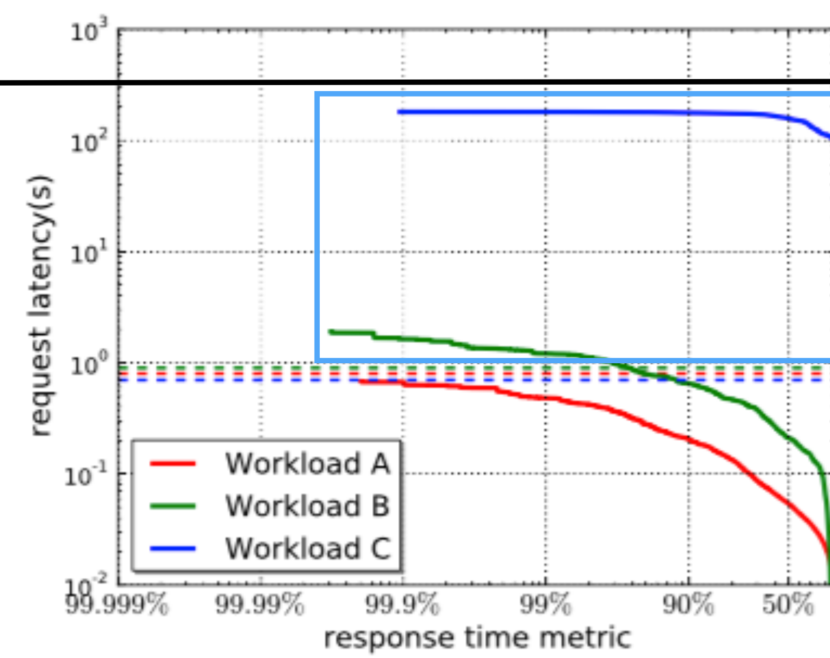
(b) Cake(reactive): blue SLO violated @ 84th-%ile



(c) bySLO: green SLO violated @ 91st-%ile



(d) EDF: green SLO violated @ 97th-%ile



(e) PS: blue SLO always violated

All policies except PM violates SLOs

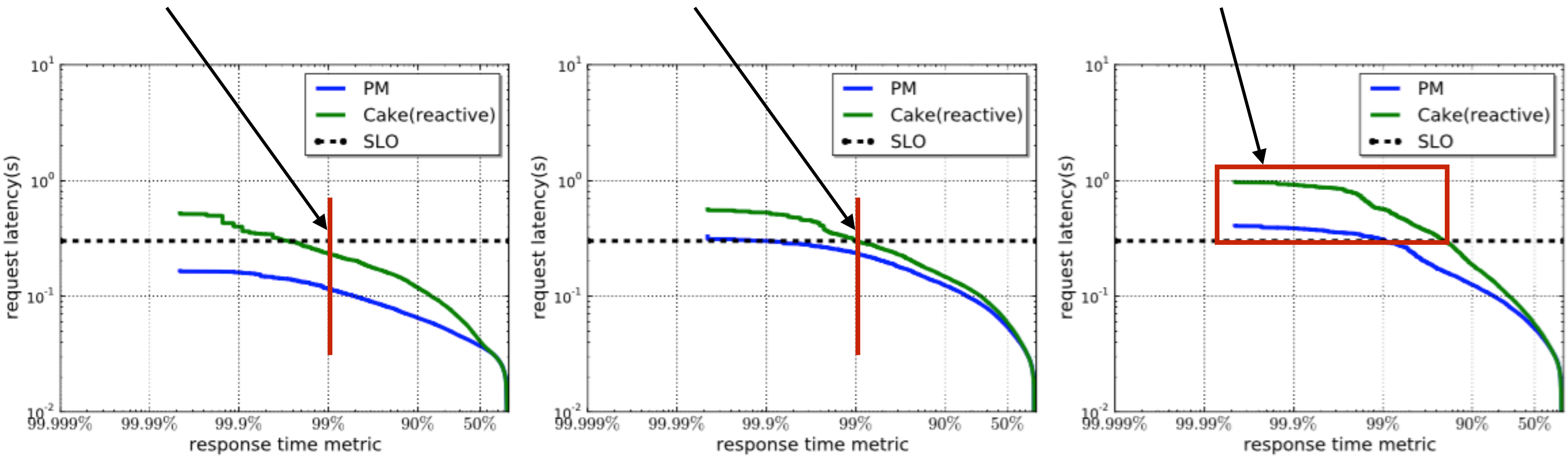
Experiments

Latency Under Bursty Workload

Both satisfies SLO at 99%

Cake violates SLO > 99%, while PM satisfies SLO

Under extreme burstiness, both fails to satisfy SLO



(a) low burstiness, $C_A^2 = 1$, Workload F

(b) high burstiness, $C_A^2 = 20$, Workload G

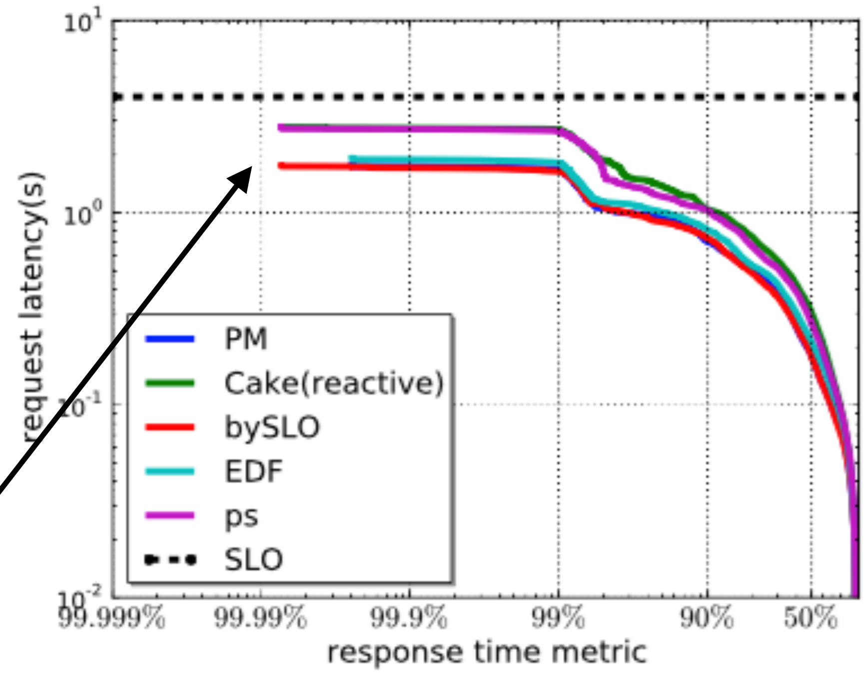
(c) very high burstiness, $C_A^2 = 40$, Workload H

C_A^2 : Squared coefficient of variation of inter-arrival times
Higher value means burstier workload

Experiments

Mis-behaving Workloads

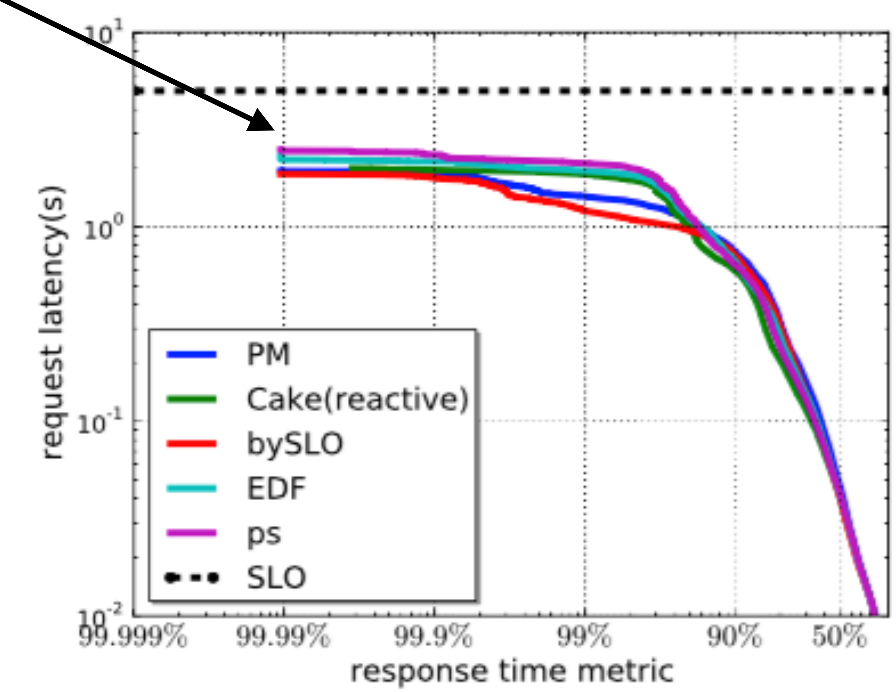
Workload D

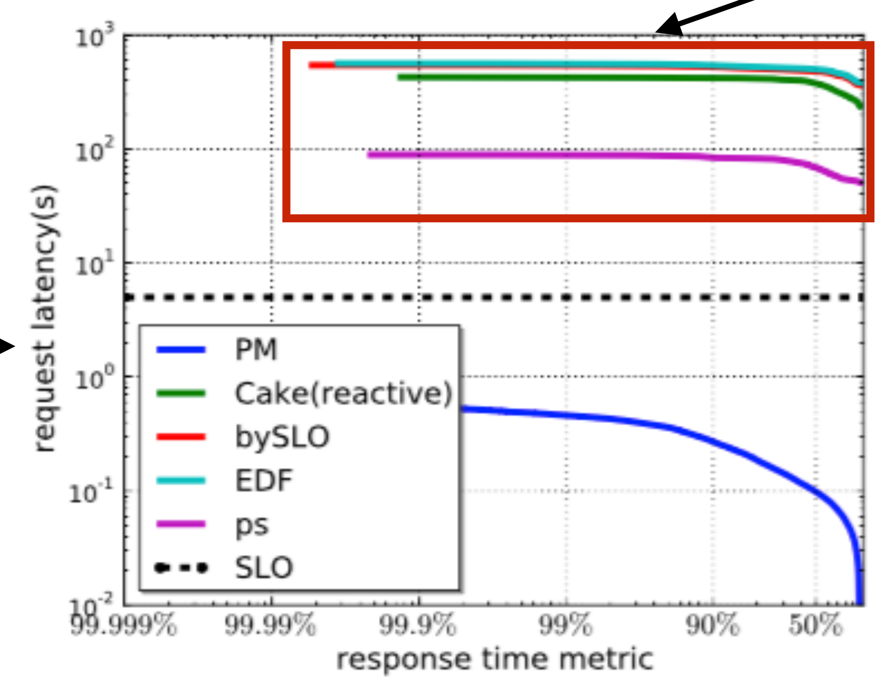


Workload D mis-behaves (hogs network)

Initially both workloads satisfy SLOs

Workload C



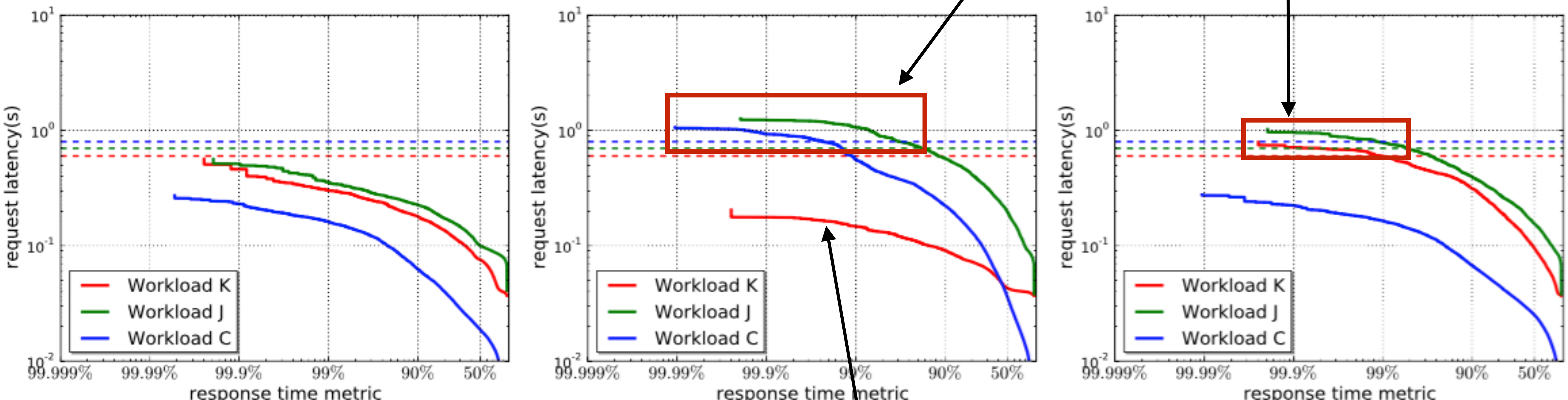


All policies except PM no longer satisfy SLO

Experiments

Network Bottlenecked Workloads

SLO violated for bySLO
and no network QoS



(a) PM: all workloads' SLOs satisfied

(b) bySLO: green SLO violated @ 95th-%ile

(c) no QoS: green SLO violated @ 98th-%ile

PM assigns workload K lower network priority thus improving Workload C and J's tail latencies

Workload K runs on Ramdisk.
Workload K has tightest SLO(highest priority)
Performs unnecessarily better than SLO.

Scenario: Workload K runs on Ramdisk(better storage latency), and others run normal disk
SLO Level(C > J > K)

Summary

PriorityMeister

- Proactive end-to-end tail latency QoS system
- Combines **priorities** and **rate limits**
- Automatically configures itself
- Performs well under real world bursty workloads

Comments

- Pros
  - Unique system that provides good tail latency
  - Allows multi-tenant, multi-resources
  - Extensive experiments
- Cons
  - Prior computation of trace is not always possible
  - No mention on fault tolerance
  - No mention on how to ensure throughput SLO

Discussion

- What do we lose at cost of better tail-latency?
- What semantic meanings does rate(r) and bucket size(b) of leaky token bucket model have?
- Is it possible to combine reactive(Cake) and proactive(PriorityMeister) approach? Would it perform better than both?
- Experiment shows bySLO(just assigning higher priority for lower SLO) performs really well. Why so?

Thank You!

Backup Slides

How enforcers work?

Enforcers

- Storage enforcer
  - # tokens == Amount of storage time consumed by request
  - Queues on top of NFS
- Network enforcer
  - # tokens == Number of transmitted bytes by request
  - Enforce priority with network QoS level

Linear Programming

- Estimate $\beta_W(t)$: Maximize interference with higher priority workloads
- Let's estimate $t=\beta_W^{-1}(y)$ instead
- For queue q, define $t_{in}^q, t_{out}^q, R_k^q, R_k'^q$

Time constraints

$$t_{in}^q \leq t_{out}^q \qquad t_{out}^{q'} = t_{in}^q$$

Flow constraints

$$R_k^q \leq R_k'^q$$

Rate limit constraints

$$R_k'^q - R_k^{q^*} \leq r_i \times (t_{out}^q - t_{in}^{q*}) + b_i$$, q*: workload k's first queue

Work conservation constraints

$$\sum_k (R_k'^q - R_k^q) = B_q \times (t_{out}^q - t_{in}^q)$$

Objective function: $\max(t_{out}^{q_n} - t_{in}^{q_1})$, where $(R_w'^{q_n} - R_w^{q_1}) = y$

Experiments

Traces

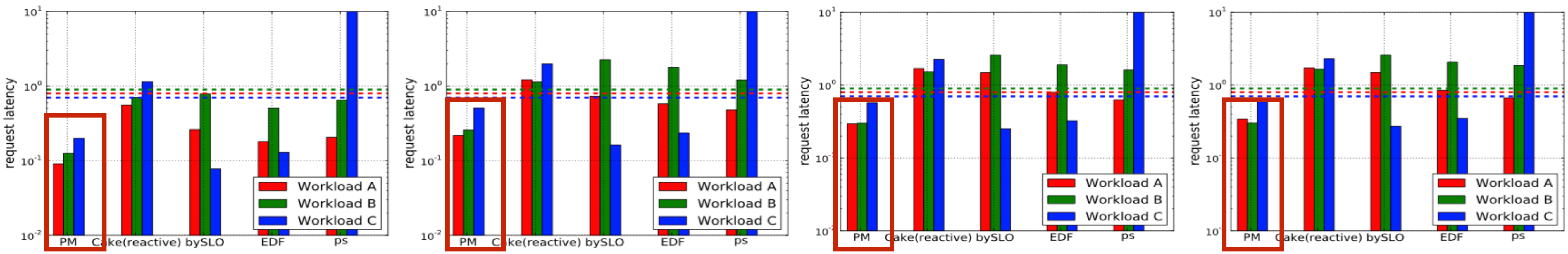
Workload label	Workload source	Estimated storage load	Estimated network load	Interarrival Variability, C_A^2
Workload A	DisplayAds production trace	5%	5%	1.3
Workload B	MSN storage production trace	5%	5%	14
Workload C	LiveMaps production trace	55%	5%	2.2
Workload D	Exchange production trace (behaved)	10%	5%	23
Workload E	Exchange production trace (misbehaved)	$> 100\%$	15%	145
Workload F	Synthetic low burst trace	25%	5%	1
Workload G	Synthetic high burst trace	25%	5%	20
Workload H	Synthetic very high burst trace	25%	5%	40
Workload I	Synthetic medium network load trace 1	35%	20%	1
Workload J	Synthetic medium network load trace 2	45%	25%	1
Workload K	Synthetic ramdisk trace	N/A	35%	3.6
Workload L	Synthetic large file copy	N/A	N/A	N/A

Other QoS Policies

- Proportional sharing (ps)
  - Each workload gets equal share of storage time
- Cake
  - Dynamically adjust proportional shares to meet latency SLOs
- Earliest Deadline First (EDF)
  - Deadline = workload's SLO
- Prioritization by SLO (bySLO)
  - Simply assign workload priorities in order of workload latency SLOs

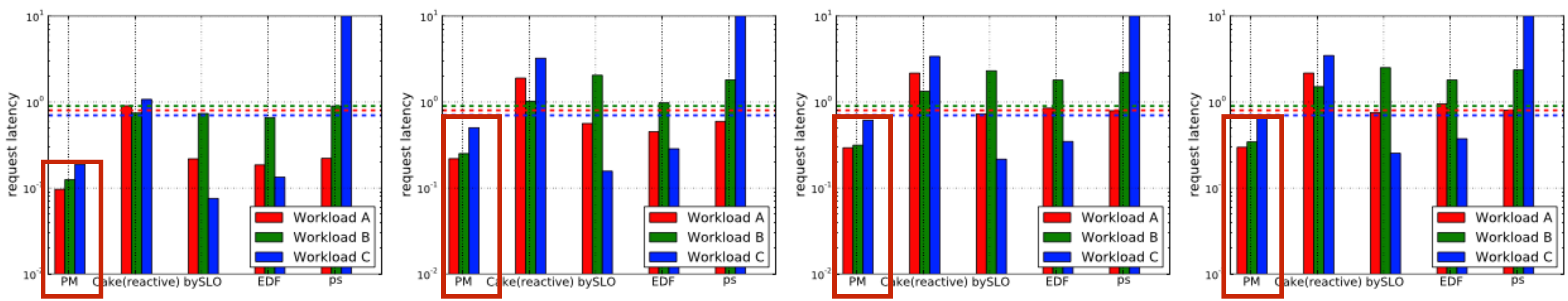
Latency Under Estimator Inaccuracy

Accurate Estimator (same as tail latency performance experiment)



(a) 90th-%ile of latency (b) 99th-%ile of latency (c) 99.9th-%ile of latency (d) 99.99th-%ile of latency

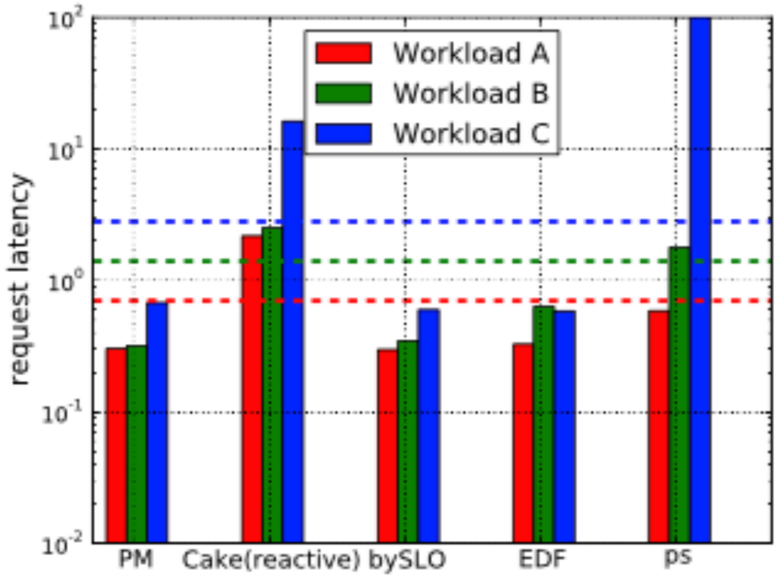
Inaccurate Estimator (Token counting does not reflect reality well)



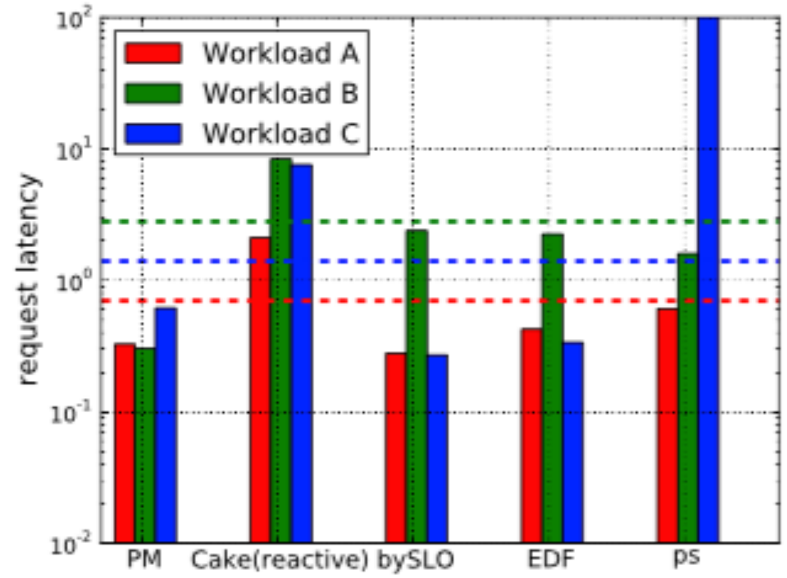
(a) 90th-%ile (b) 99th-%ile (c) 99.9th-%ile (d) 99.99th-%ile

PM works well for both
accurate / inaccurate estimators

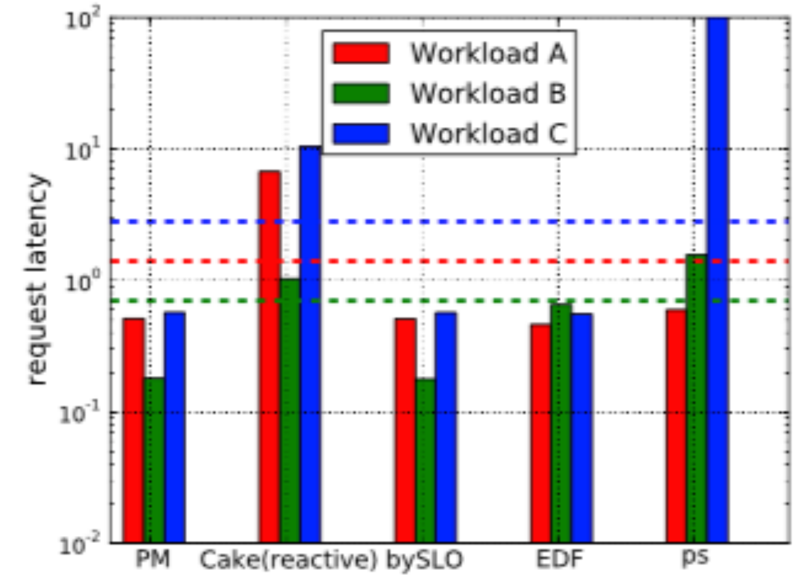
Latency Under Varying SLO Permutations



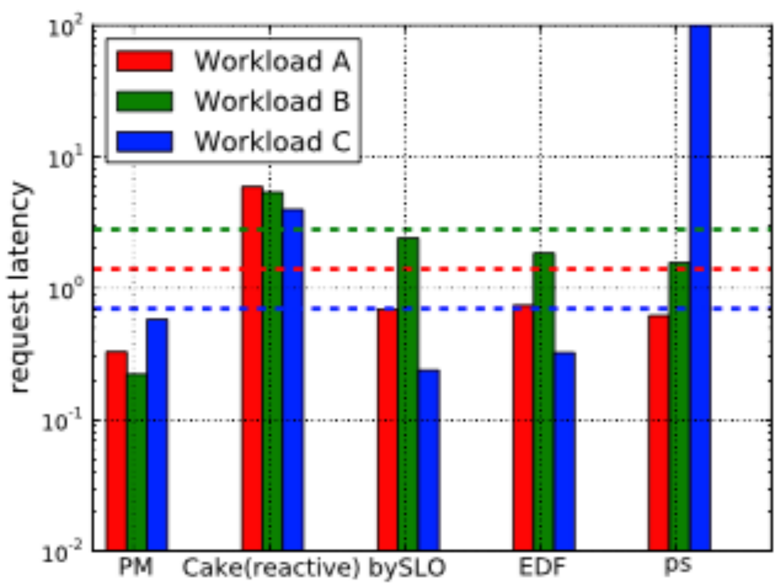
(a)



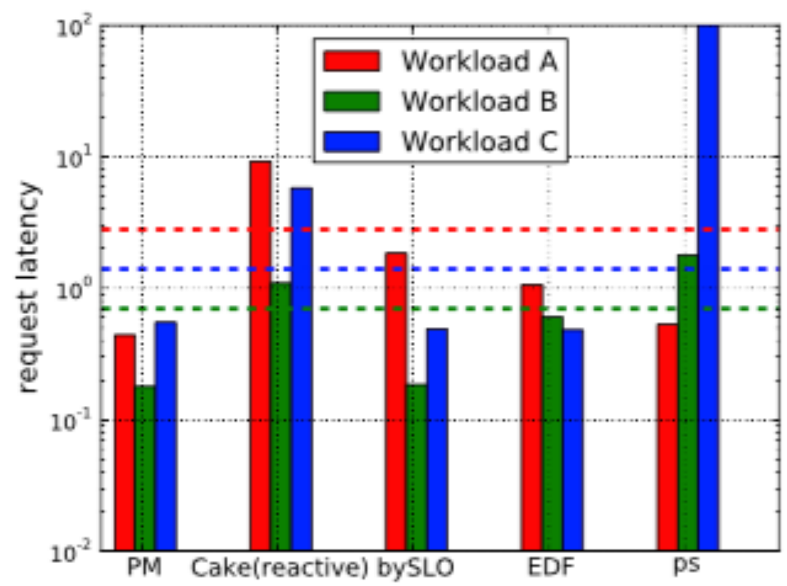
(b)



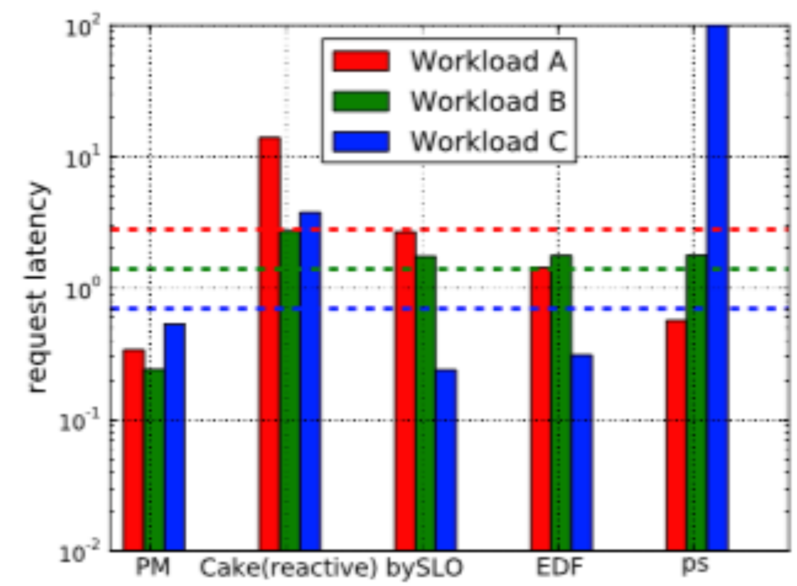
(c)



(d)



(e)



(f)

For different permutations of SLO levels,
only PM satisfies SLOs for all permutation