CS 795 Distributed Systems & Cloud Computing

Fall 2018

Lec 9: Datacenter storage management

Yue Cheng

Announcements

- Office hours on 11/29 and 12/06 open for project wrap-up discussion
  - Or, send me an email to make an appointment for other time that works for you

- Final project presentation
  - When: 4:30—7:30pm, 12/14
  - Where: Room 4201, Engineering Building

An In-Memory Object Caching Framework with Adaptive Load Balancing

Yue Cheng (George Mason)

Aayush Gupta (IBM Research – Almaden)

Ali R. Butt (Virginia Tech)

ACM EuroSys 2015

Memcached is an essential component in datacenters

Local deployment

Cloud deployment

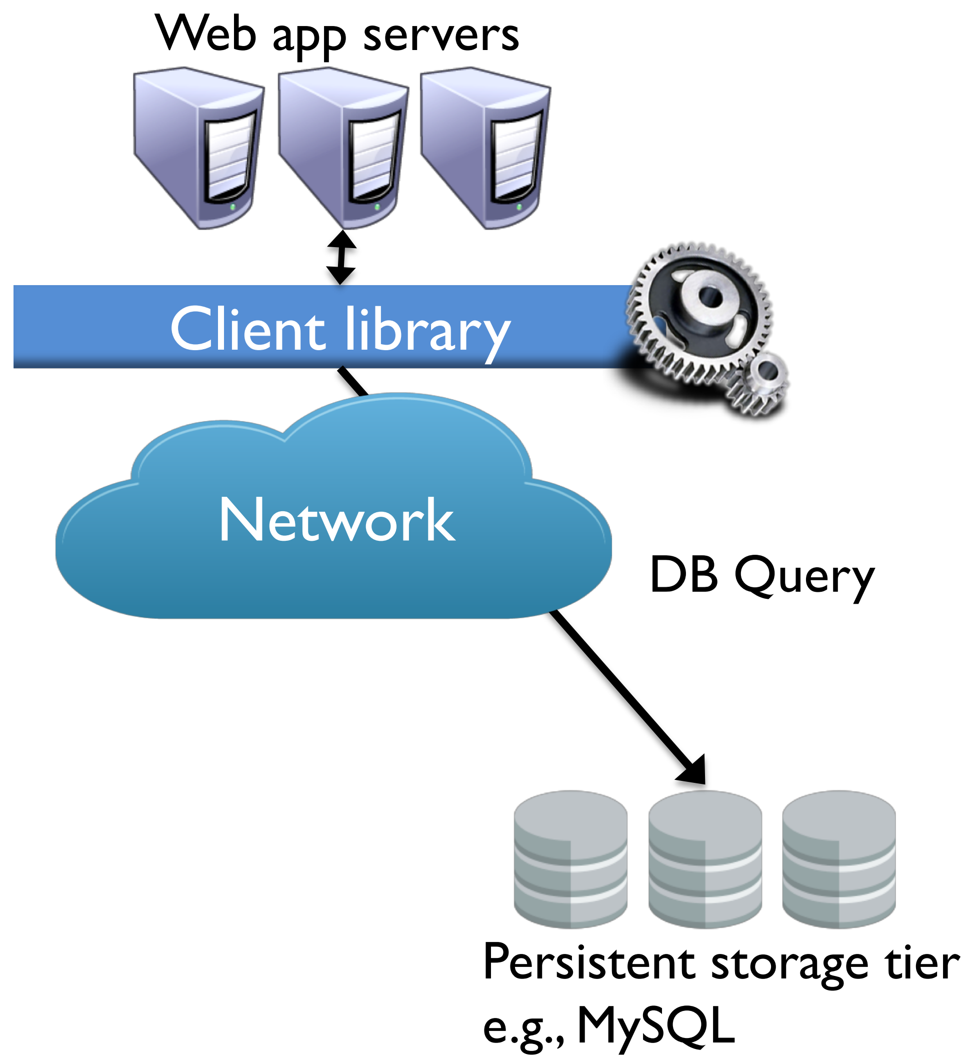
Amazon ElastiCache

Memcached is an essential component in datacenters

Local deployment

Cloud deployment

Amazon ElastiCache



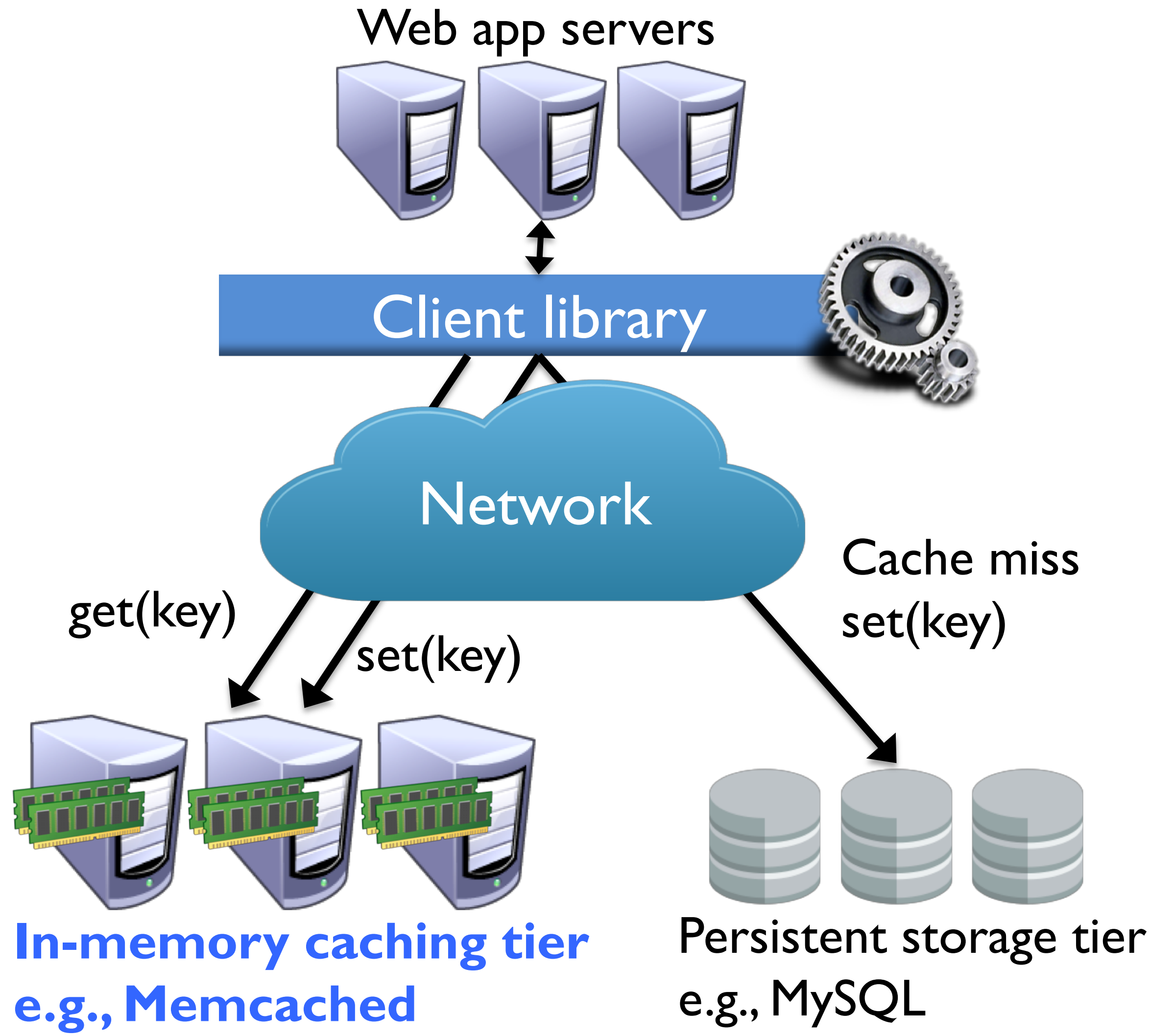
Memcached is an essential component in datacenters

Local deployment

Cloud deployment

Amazon ElastiCache

airbnb

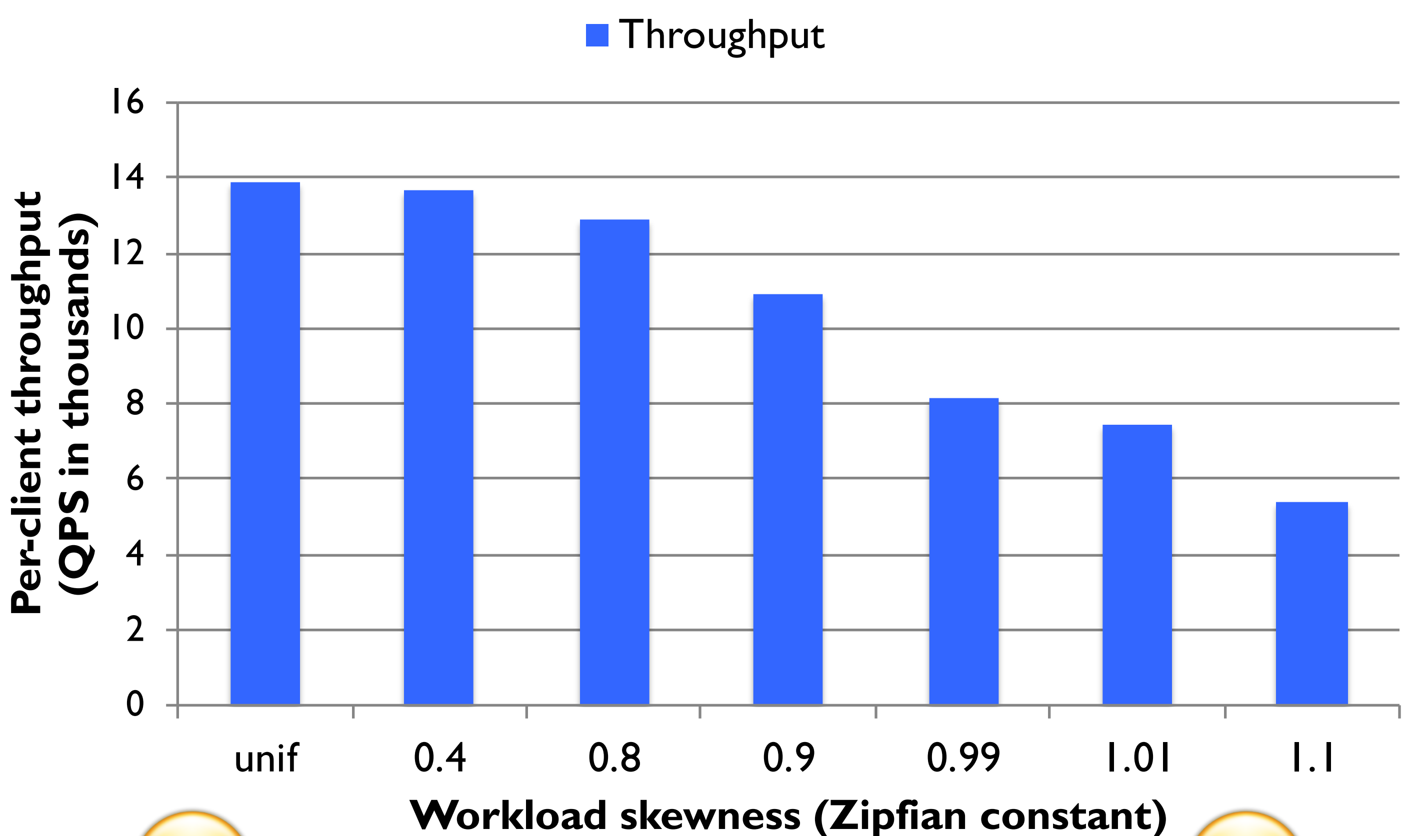


Memcached is desirable

- Offers high performance
- Enables quick deployment
- Provides ease of use

- **Problem:** Load imbalance impacts performance

Access load imbalance

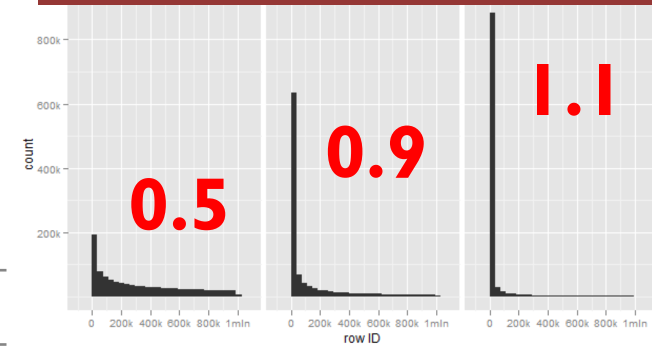


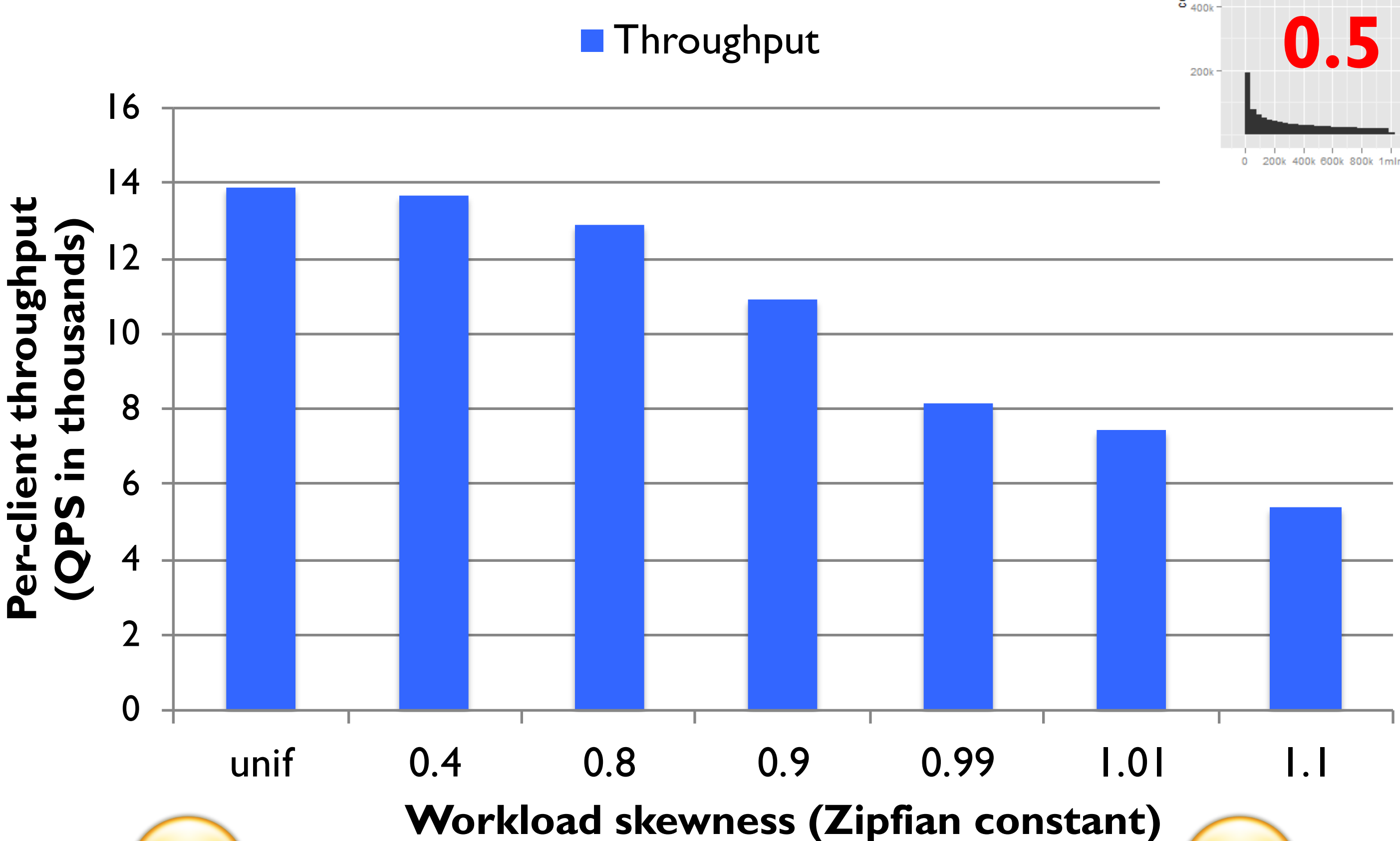
Ideal balance → high imbalance

95% GET, 5% SET, Zipfian, 20 cache servers

Access load imbalance

Key popularity distribution: different Zipfian constant

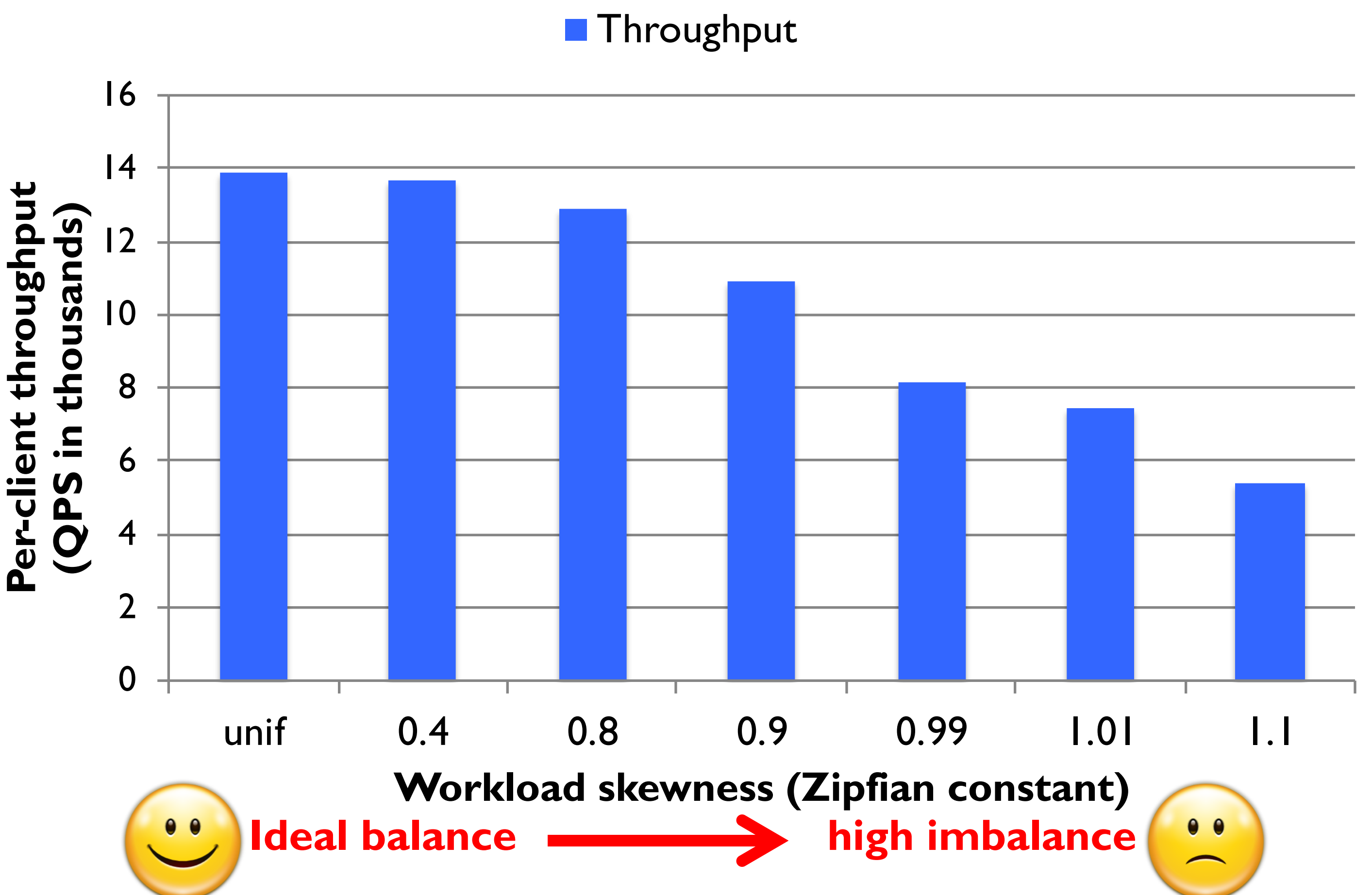




Ideal balance → high imbalance

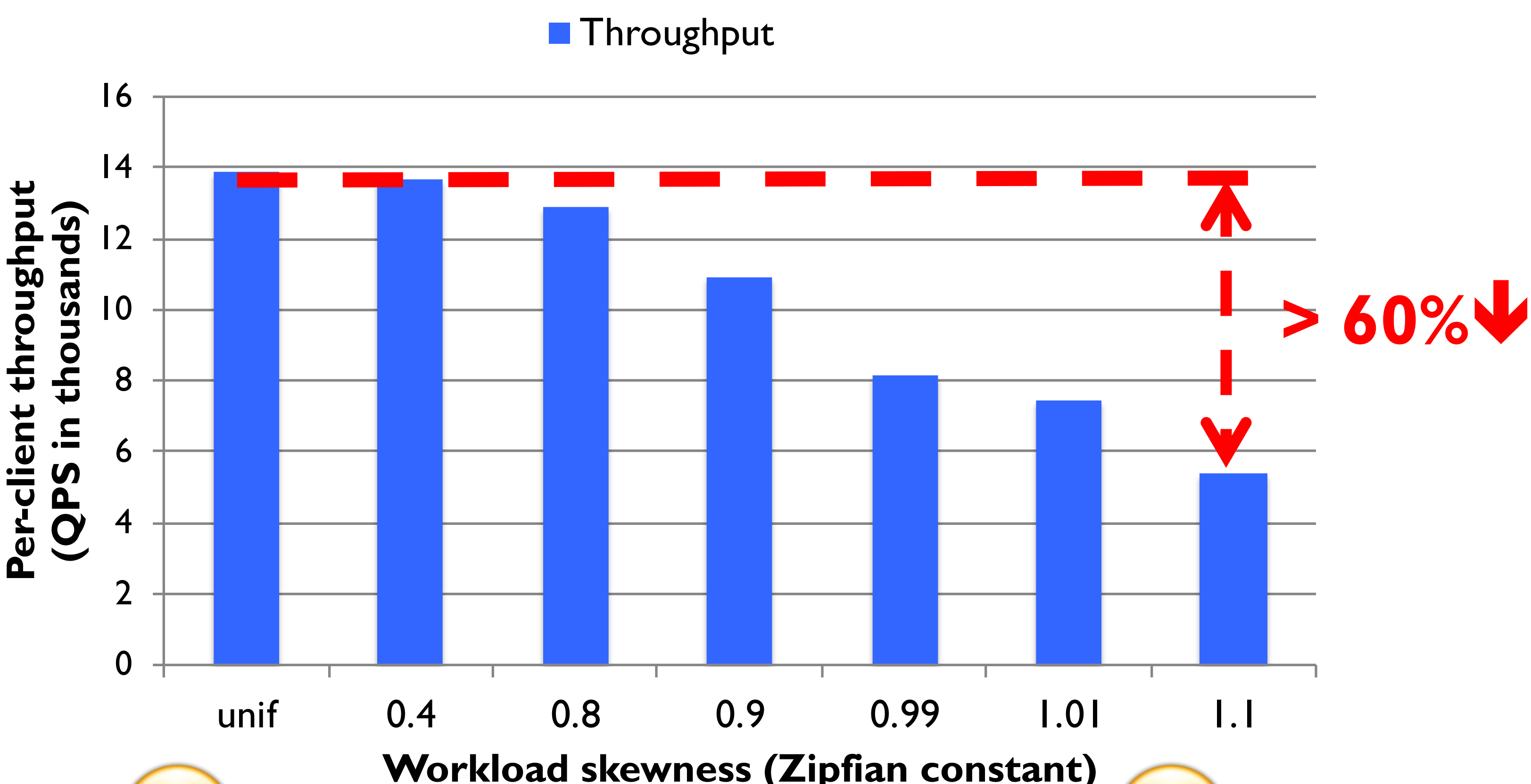
95% GET, 5% SET, Zipfian, 20 cache servers

Access load imbalance



95% GET, 5% SET, Zipfian, 20 cache servers

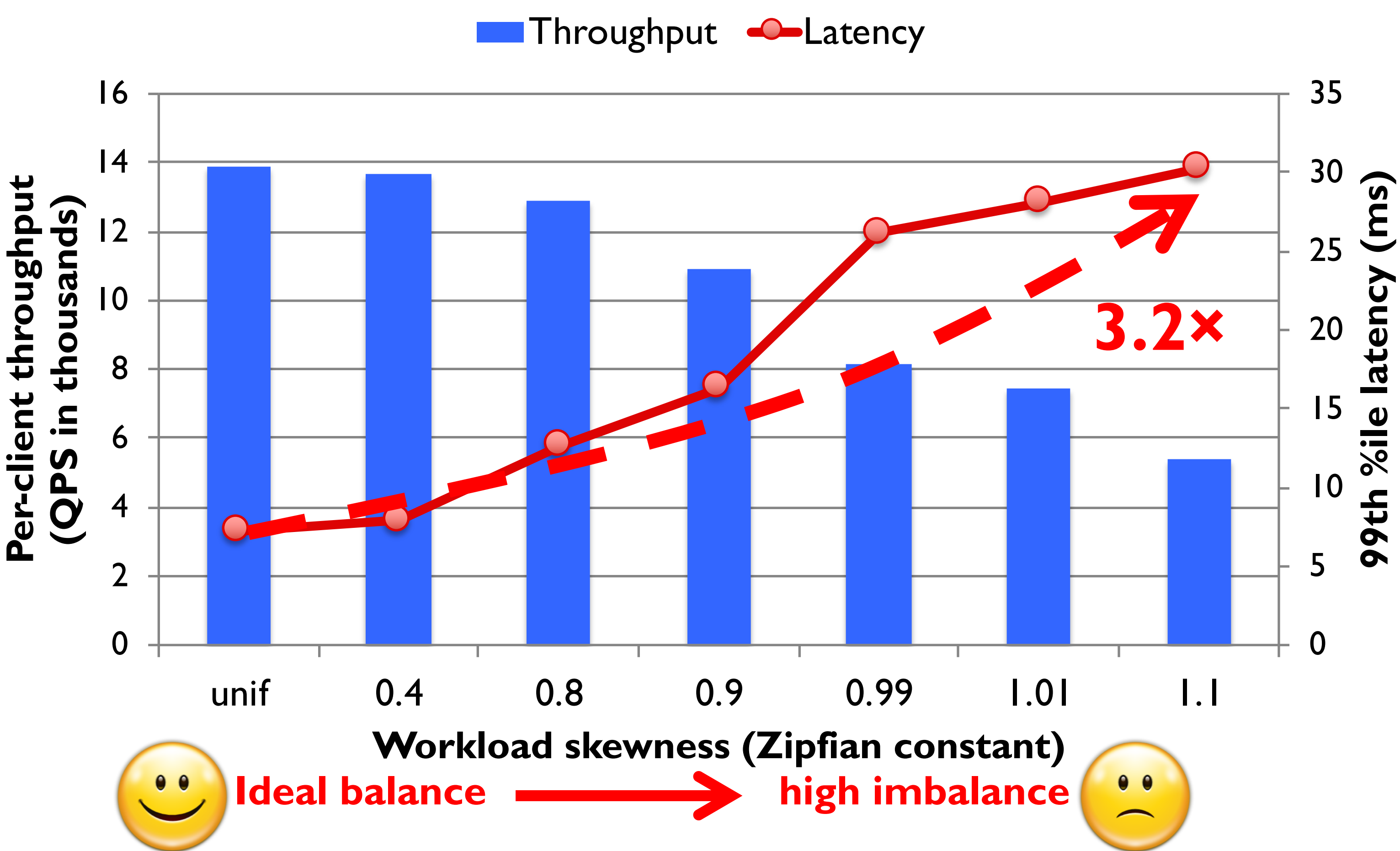
Access load imbalance



Ideal balance → **high imbalance**

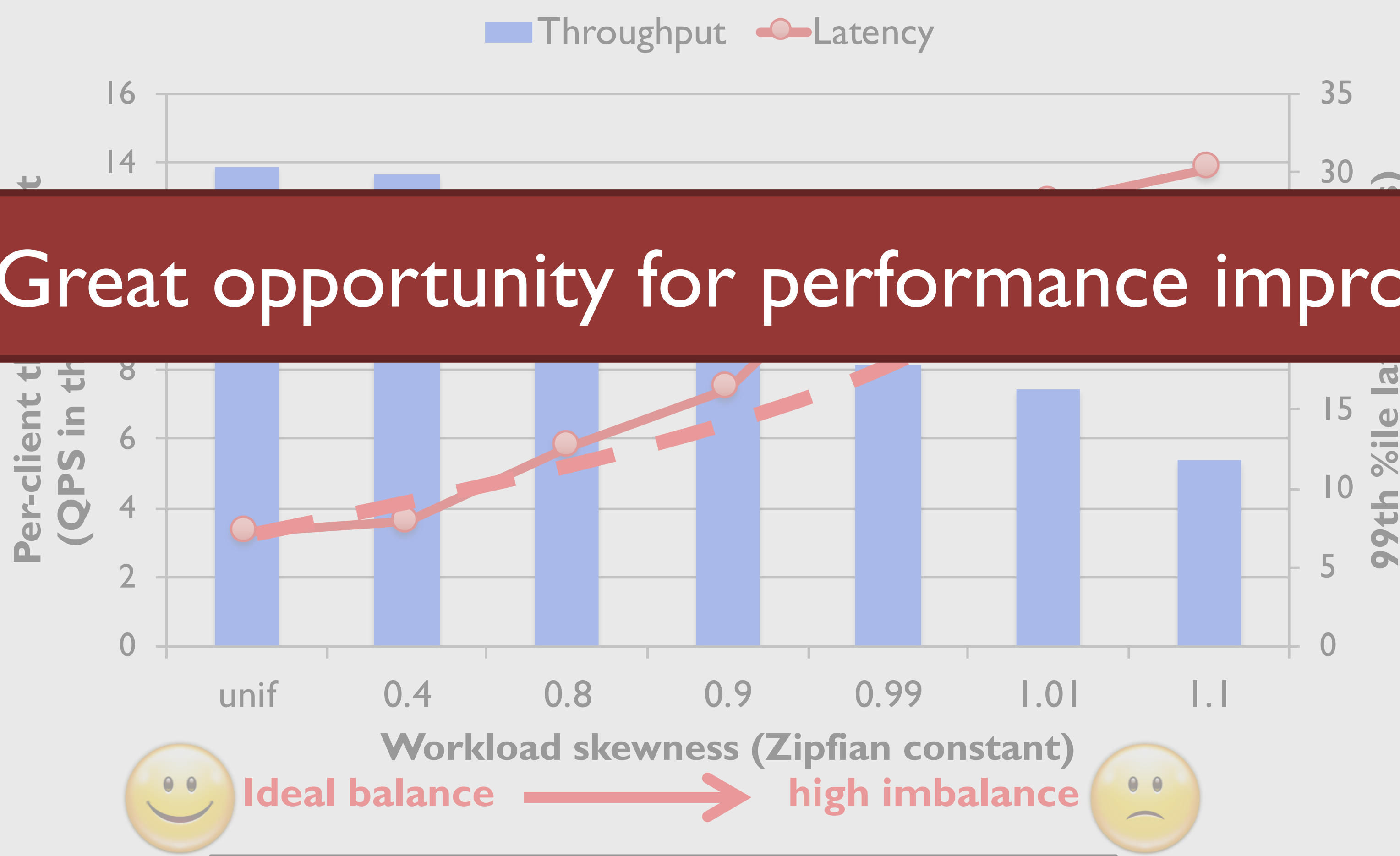
95% GET, 5% SET, Zipfian, 20 cache servers

Access load imbalance



95% GET, 5% SET, Zipfian, 20 cache servers

Access load imbalance



Great opportunity for performance improvement

95% GET, 5% SET, Zipfian, 20 cache servers

Our contribution: **MBal**

Revisiting in-memory cache design

A novel holistic in-memory caching framework with adaptive Multi-phase load Balancing

- Synthesizes different load balancing techniques
  - Key replication
  - Server-local cachelet migration
  - Coordinated cachelet migration
- Improves scale-up gains
- Mitigates load imbalance

Outline

MBal cache design

MBal load balancer design

Evaluation

Related work

Outline

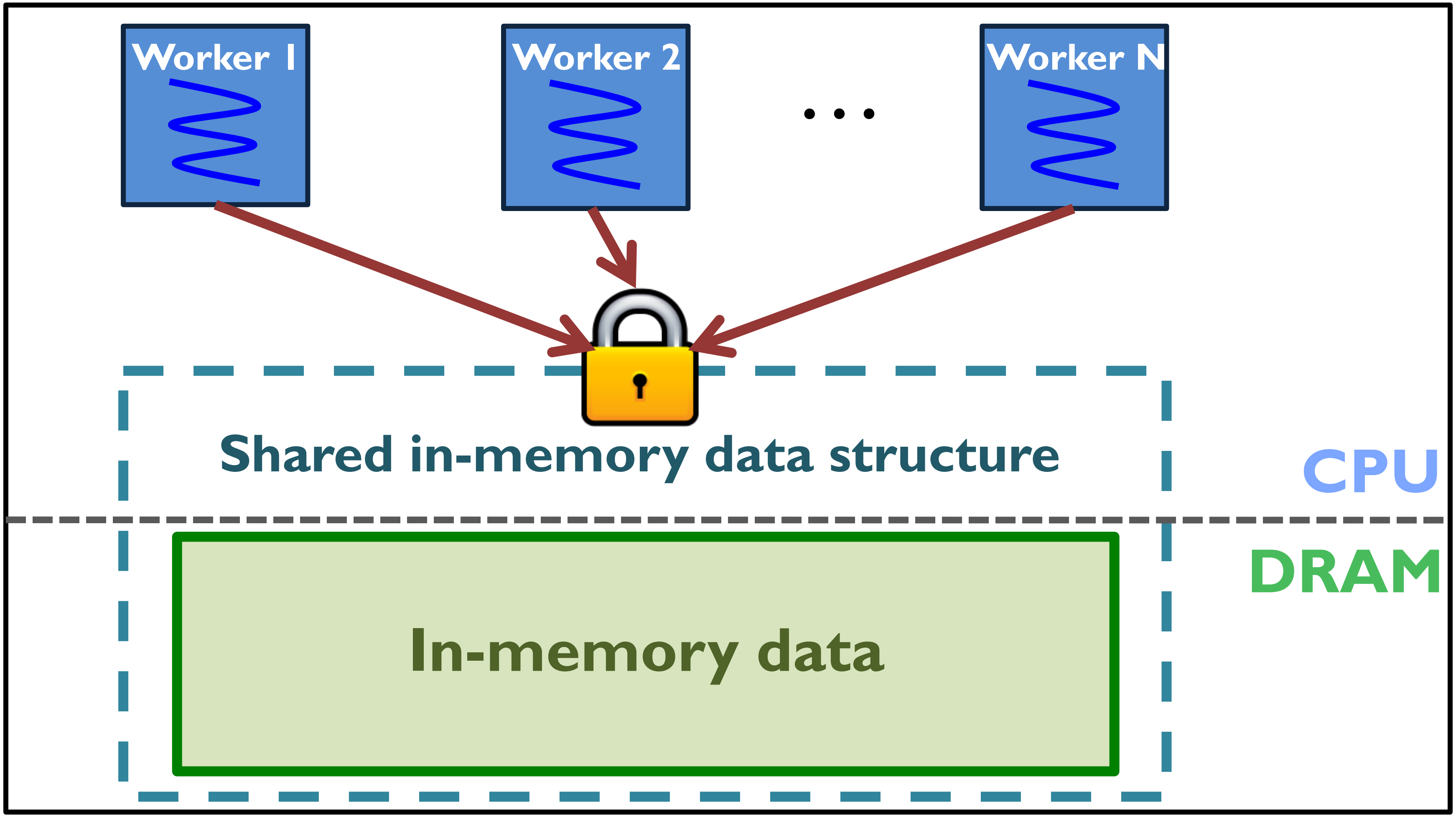
MBal Cache Design

MBal load balancer design

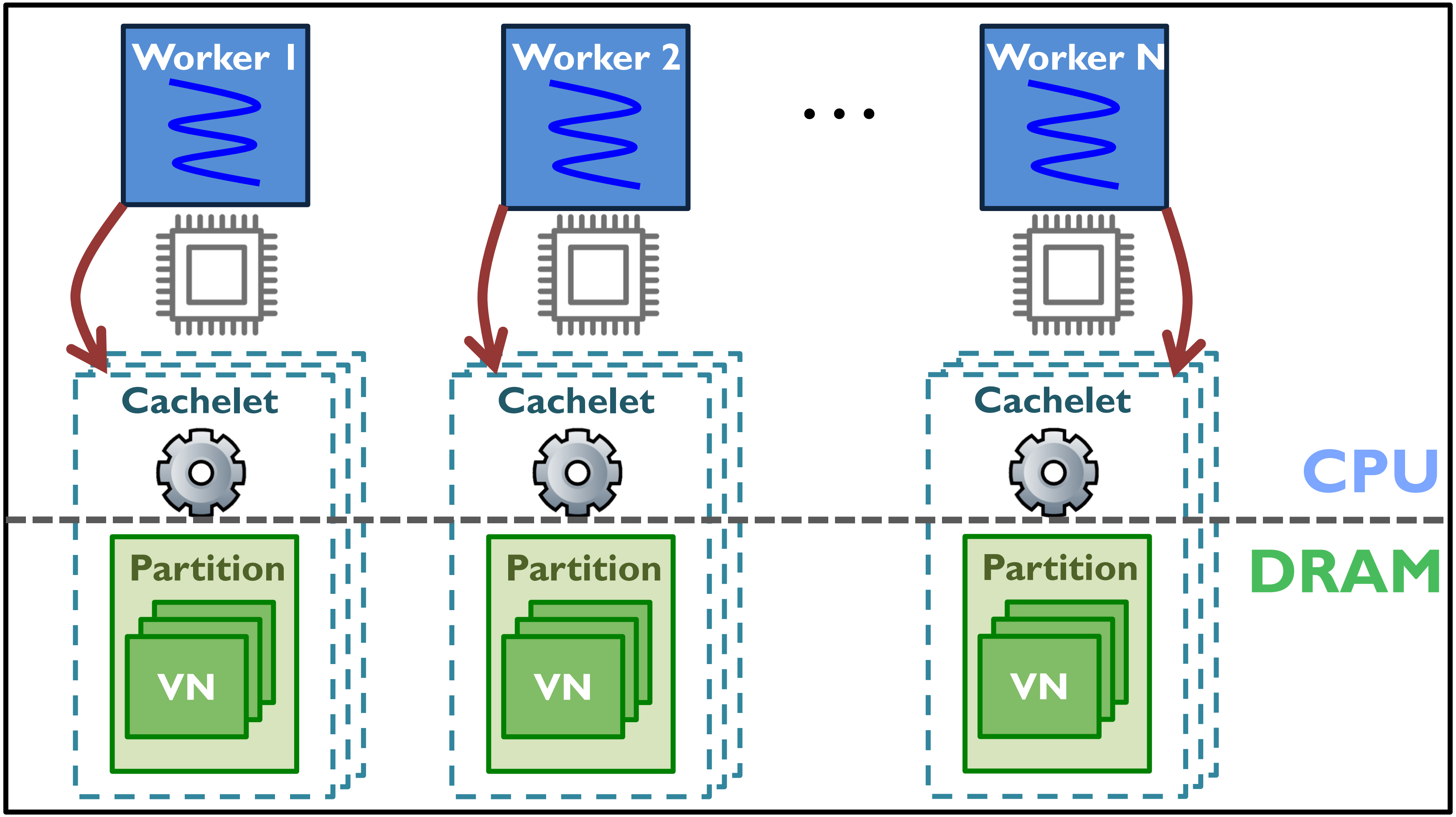
Evaluation

Related work

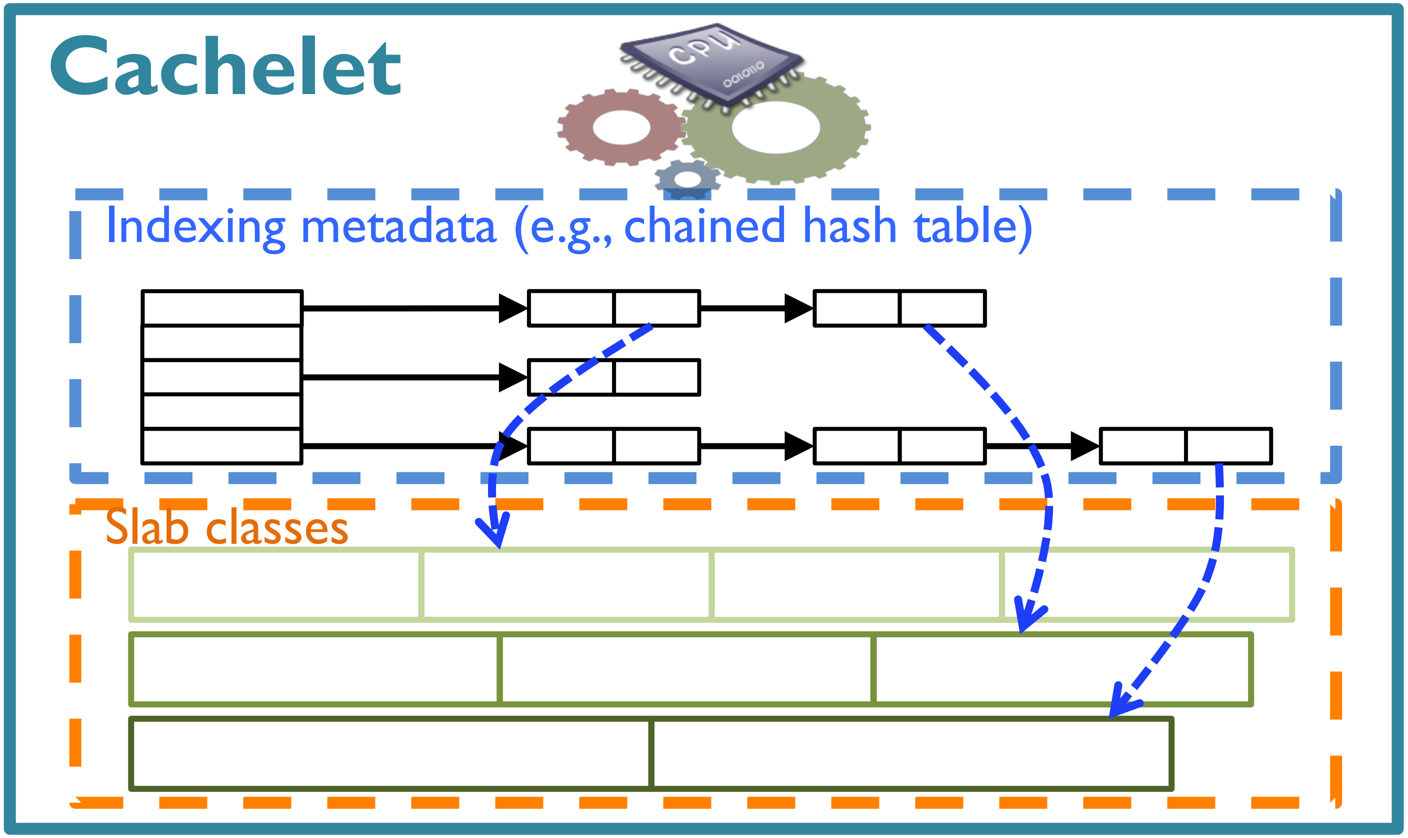
In-memory data structure in Memcached



Fine-grained data structures in MBal

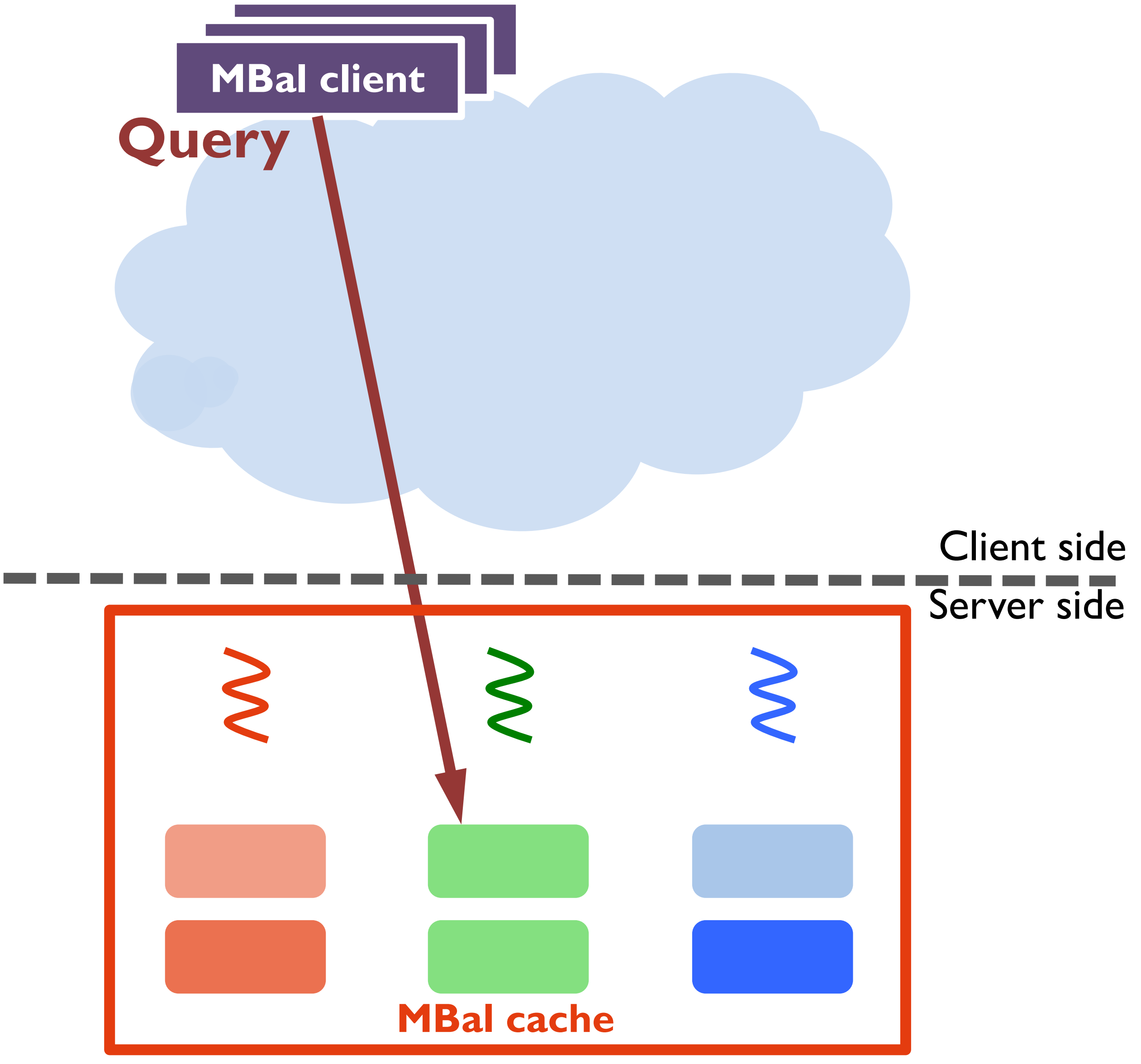


MBal cachelet: a partition with associated resources

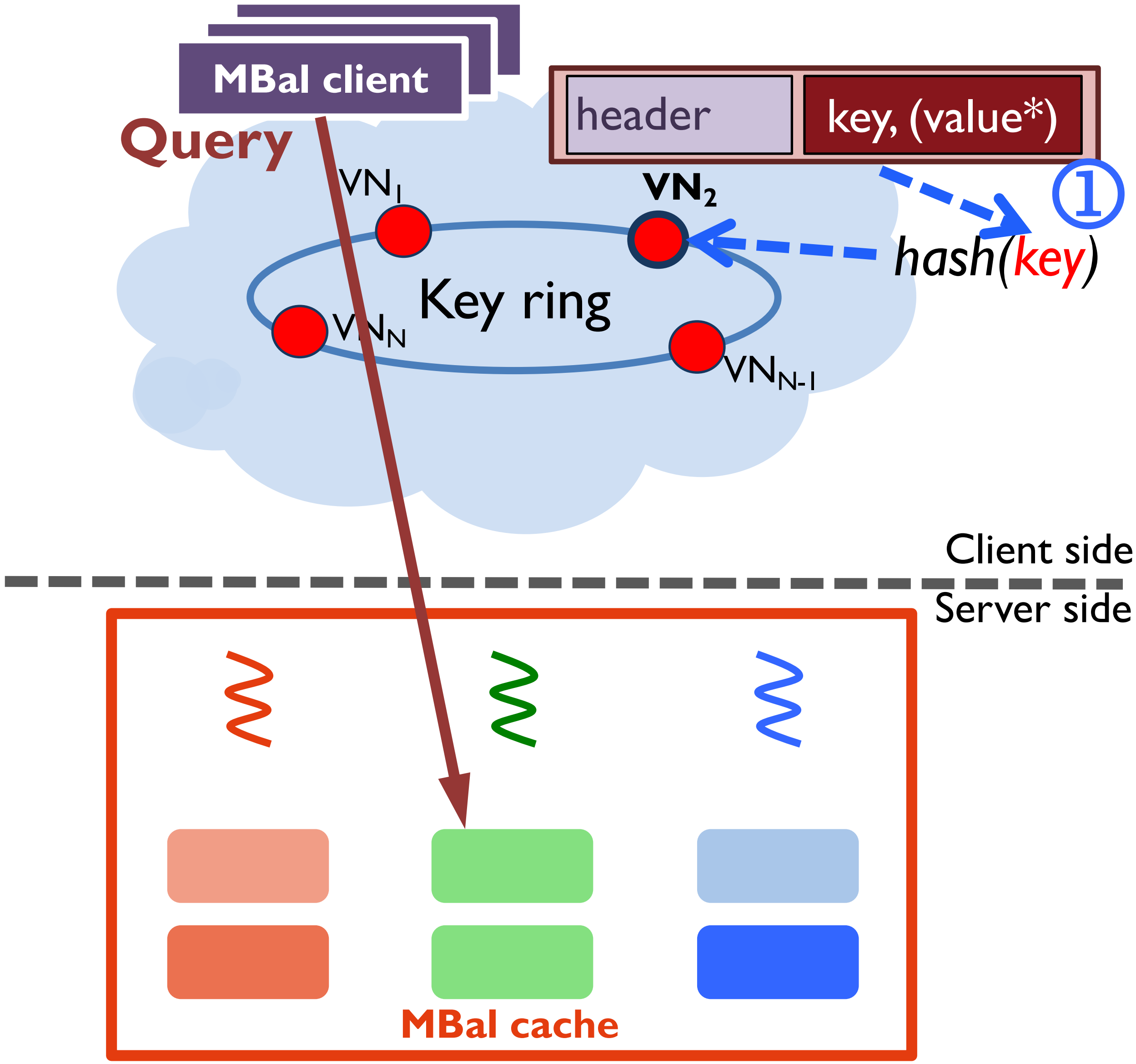


- Cachelet encapsulates resources
- Avoid lock contention

Key-to-thread mapping

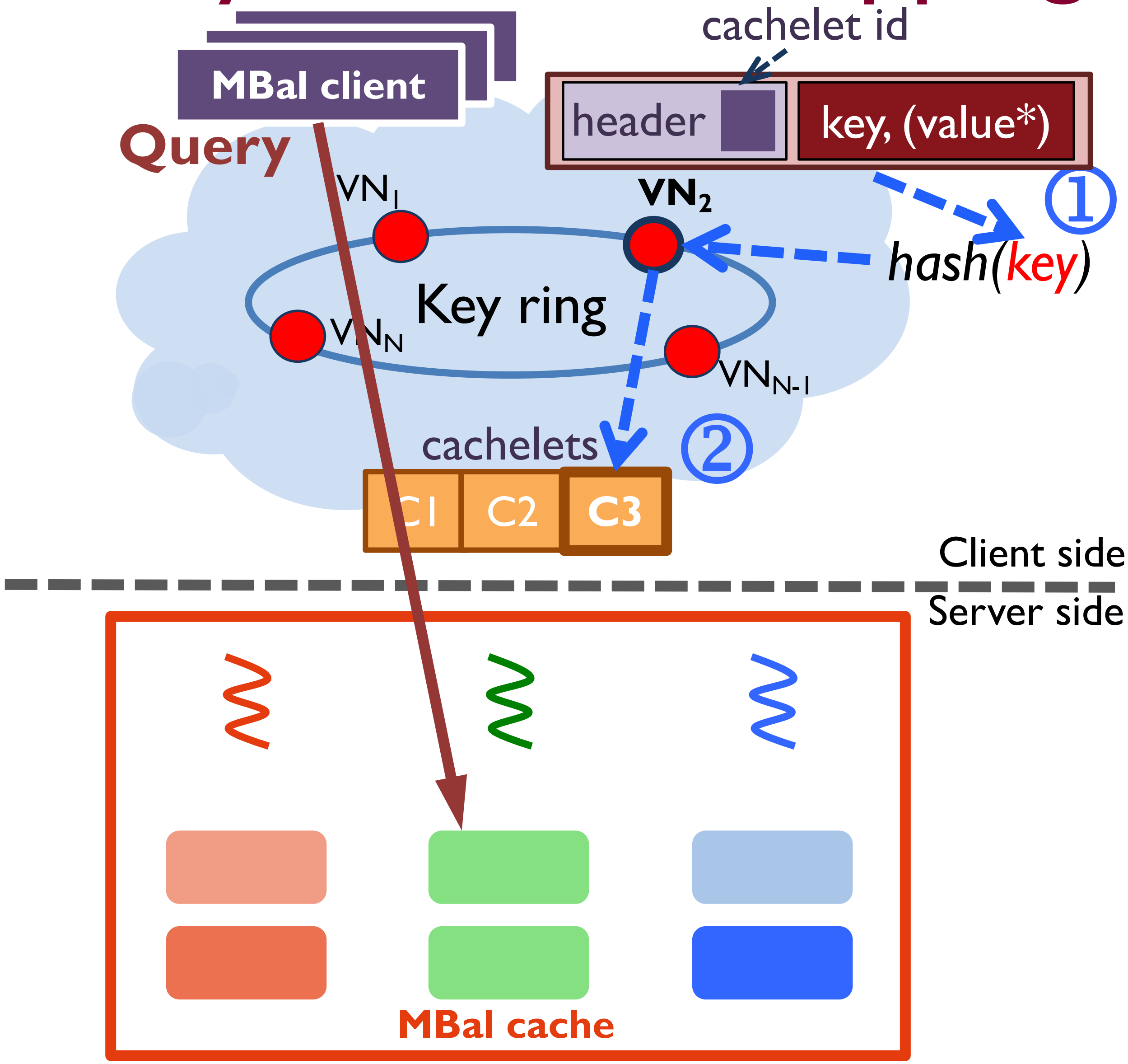


Key-to-thread mapping



① Compute VN # with hash

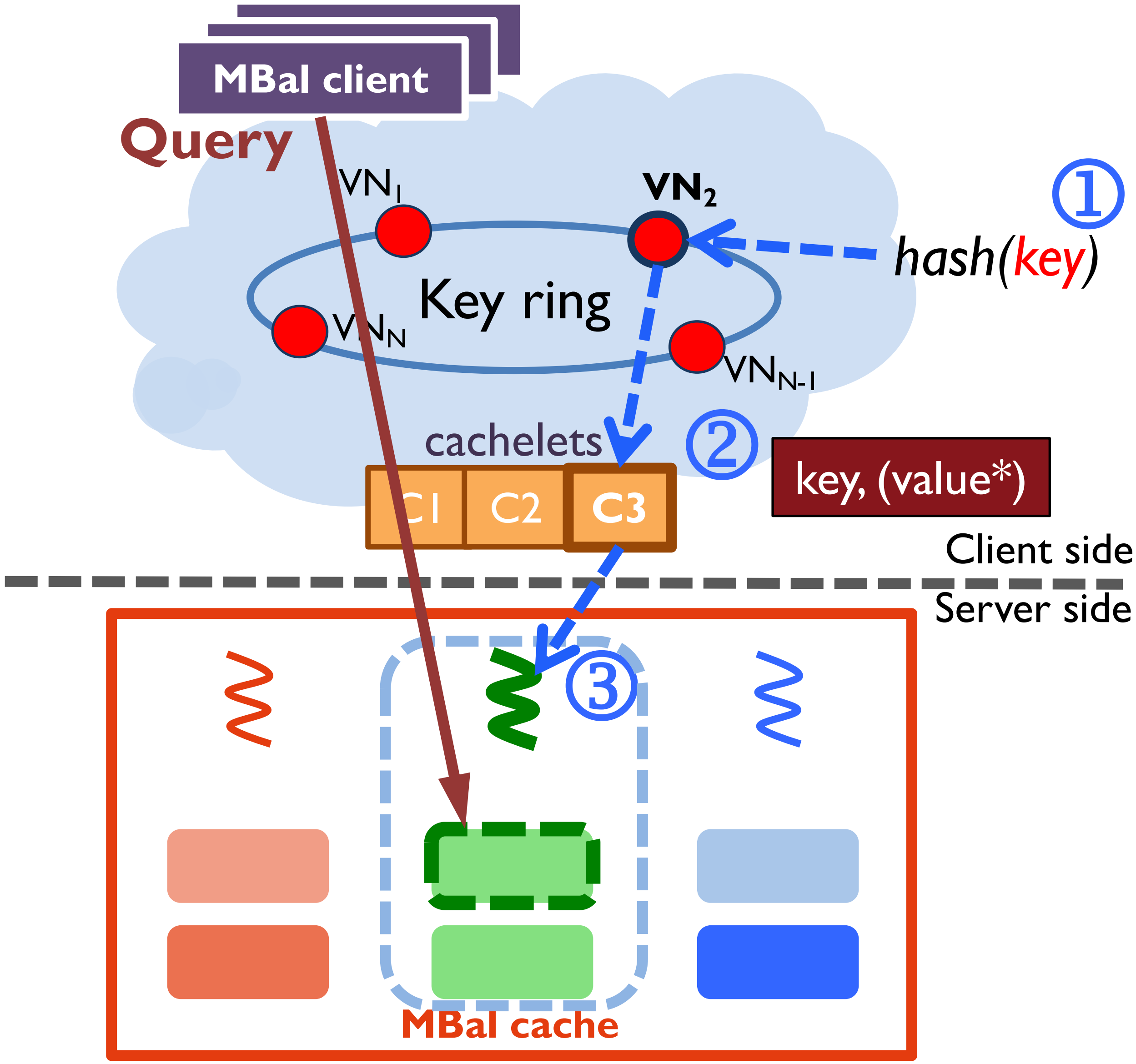
Key-to-thread mapping



① Compute VN # with hash

② Map VN # to Cachelet ID

Key-to-thread mapping



① Compute VN # with hash

② Map VN # to Cachelet ID

③ *Map Cachelet ID to the worker thread*

Outline

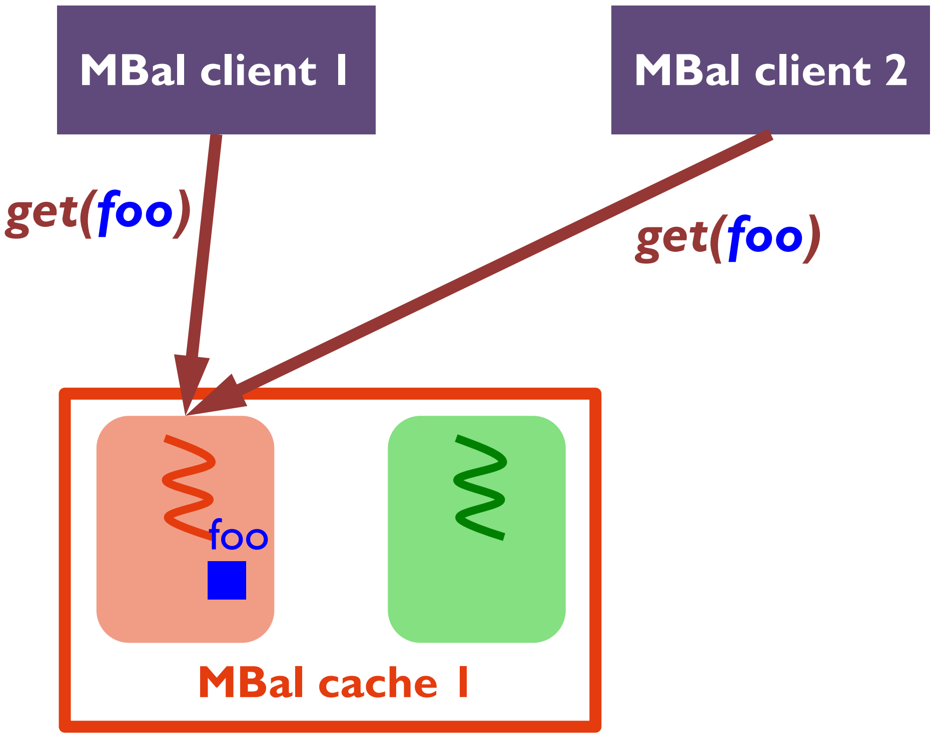
MBal cache design

MBal Multi-Phase Load Balancer

Evaluation

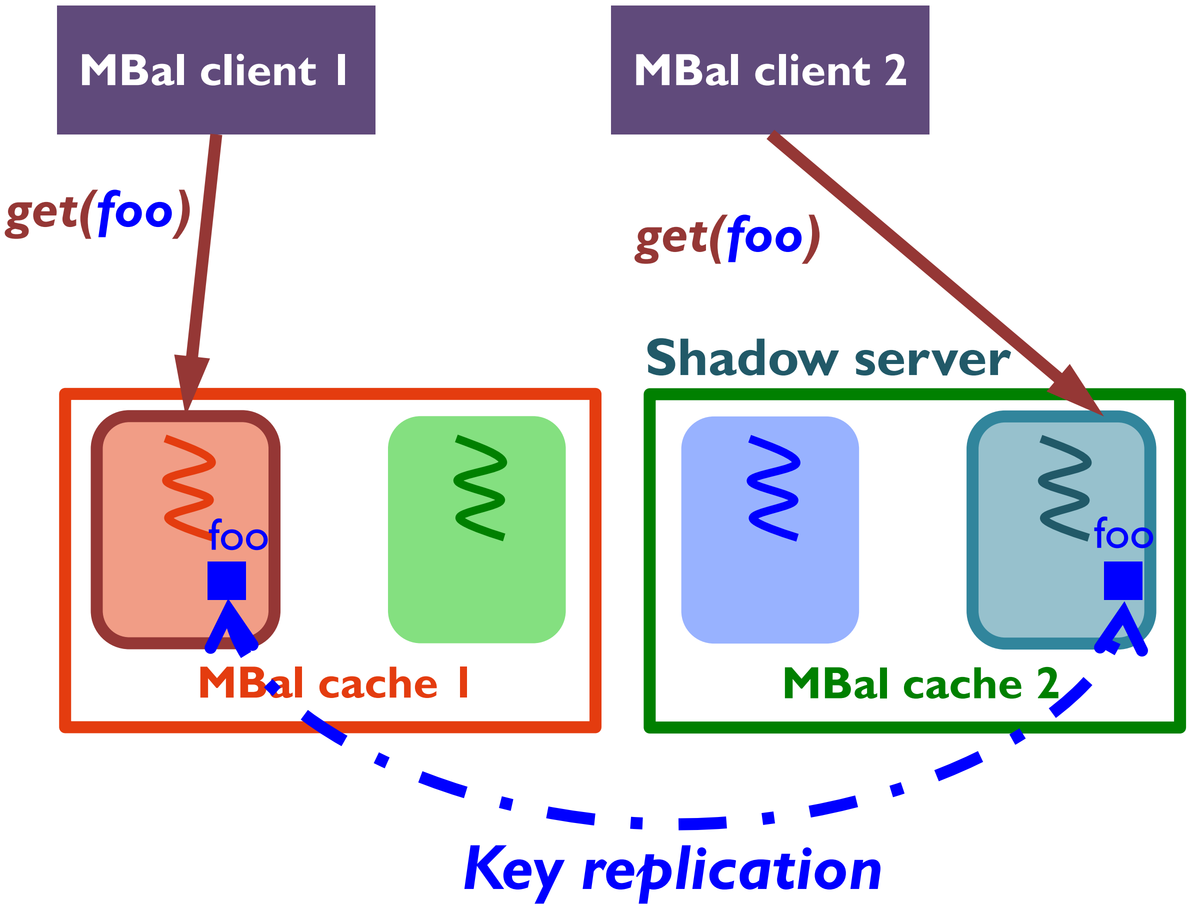
Related work

Phase 1: key replication



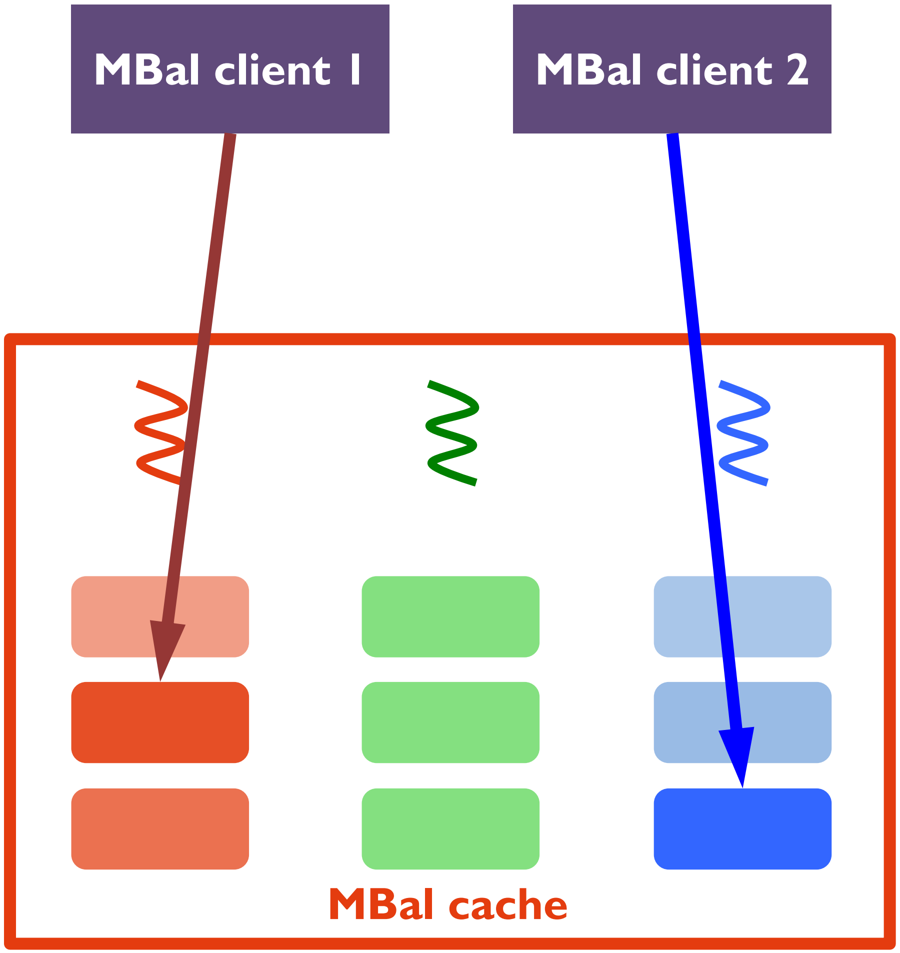
- TRIGGER?
  - EWMA access > threshold
- ACTION?
  - Randomly pick a shadow server
  - replicate hot keys
  - Proportional sampling
- FEATURES?
  - Fine-grained
  - Temporary

Phase 1: key replication



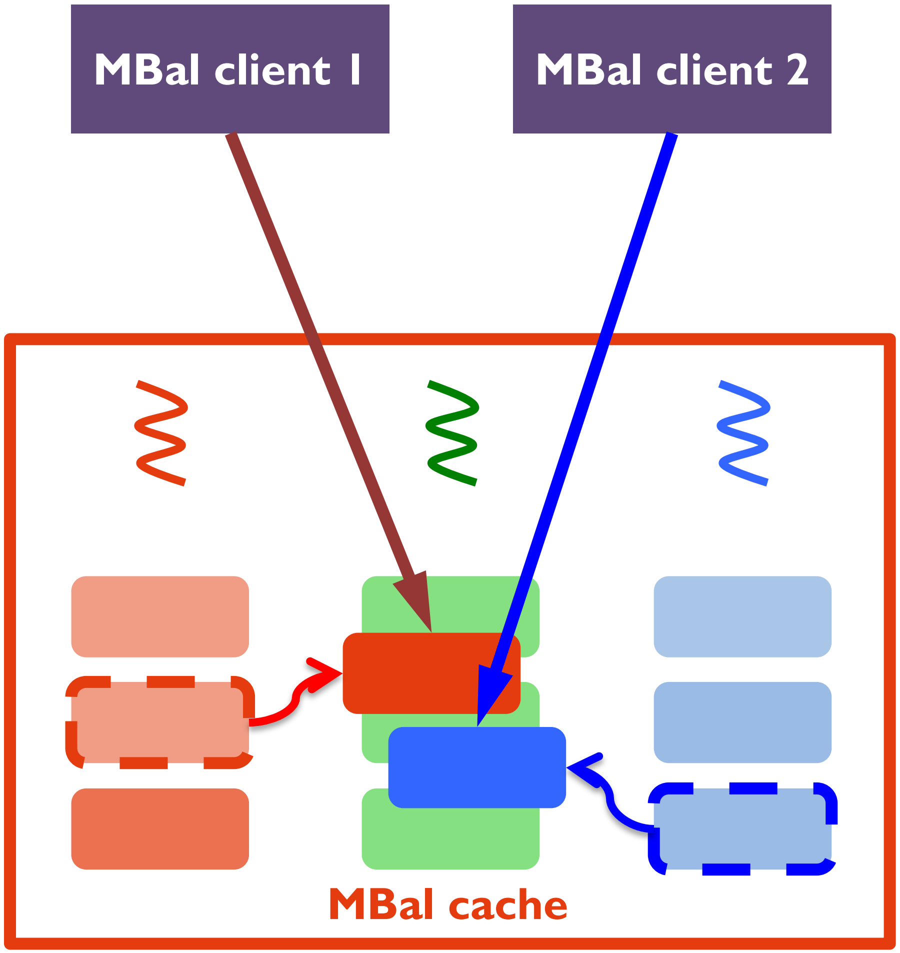
- TRIGGER?
  - EWMA access > threshold
- ACTION?
  - Randomly pick a shadow server
  - replicate hot keys
  - Proportional sampling
- FEATURES?
  - Fine-grained
  - Temporary

Phase 2: server-local cachelet migration



- TRIGGER?
  - # hot keys > $REPL_{HIGH}$
  - Enough local headroom
- ACTION?
  - Migrate/swap cachelet(s) within a server
  - ILP
- FEATURES?
  - Coarse-grained
  - Temporary

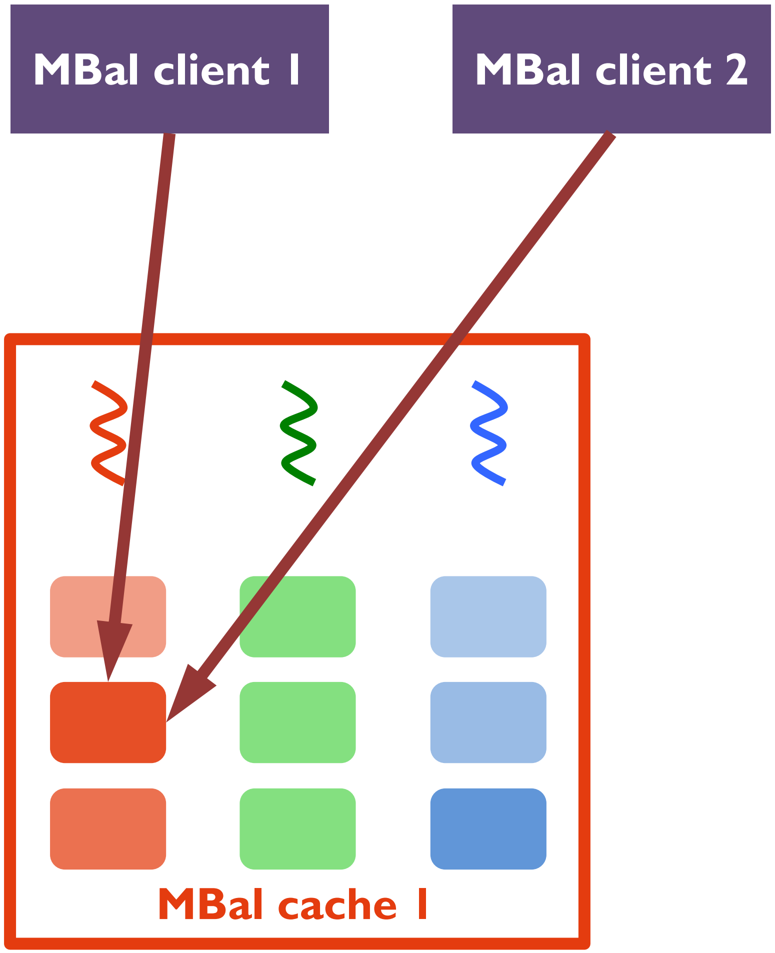
Phase 2: server-local cachelet migration



Server-local migration

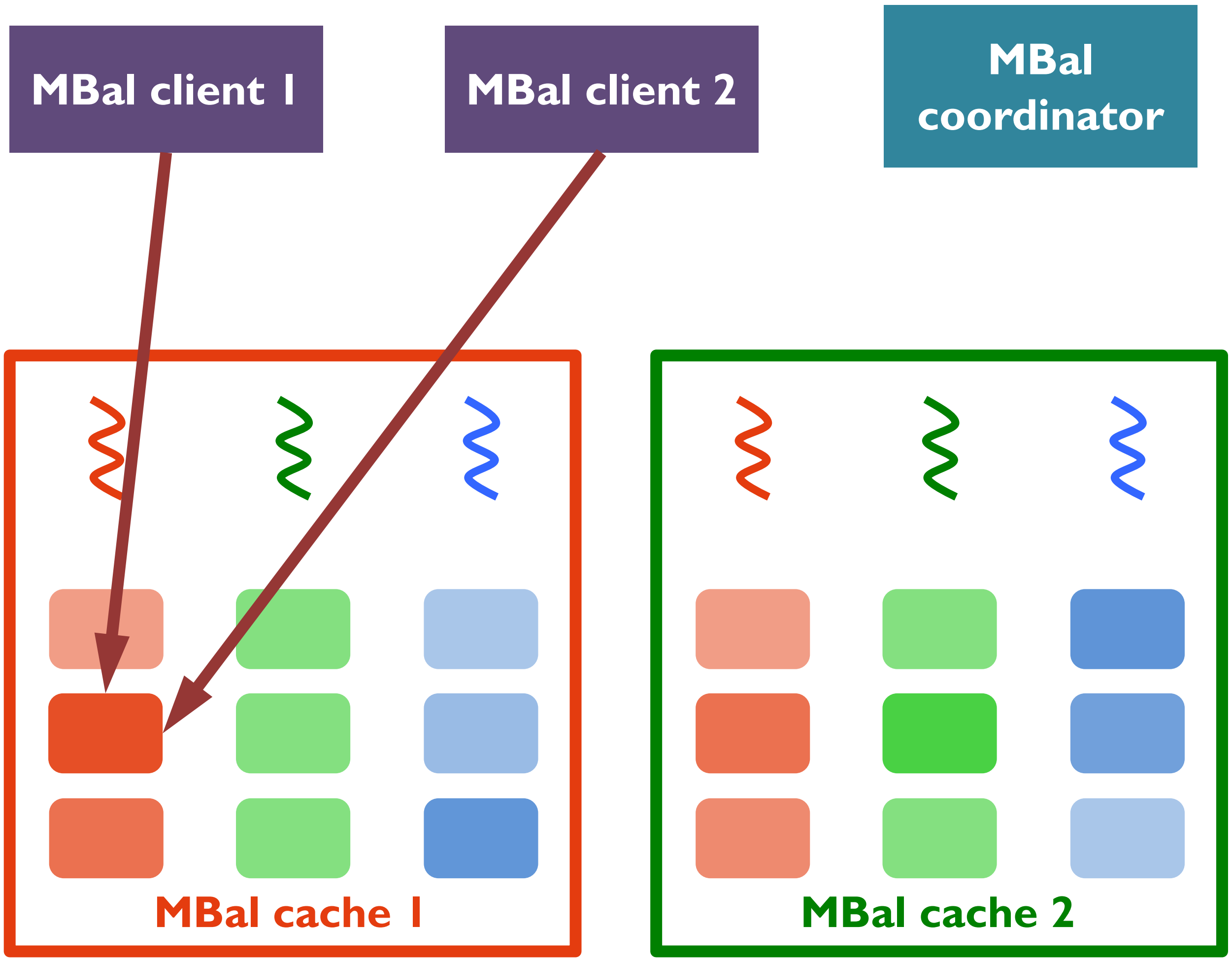
- TRIGGER?
  - # hot keys > $REPL_{HIGH}$
  - Enough local headroom
- ACTION?
  - Migrate/swap cachelet(s) within a server
  - ILP
- FEATURES?
  - Coarse-grained
  - Temporary

Phase 3: coordinated cachelet migration



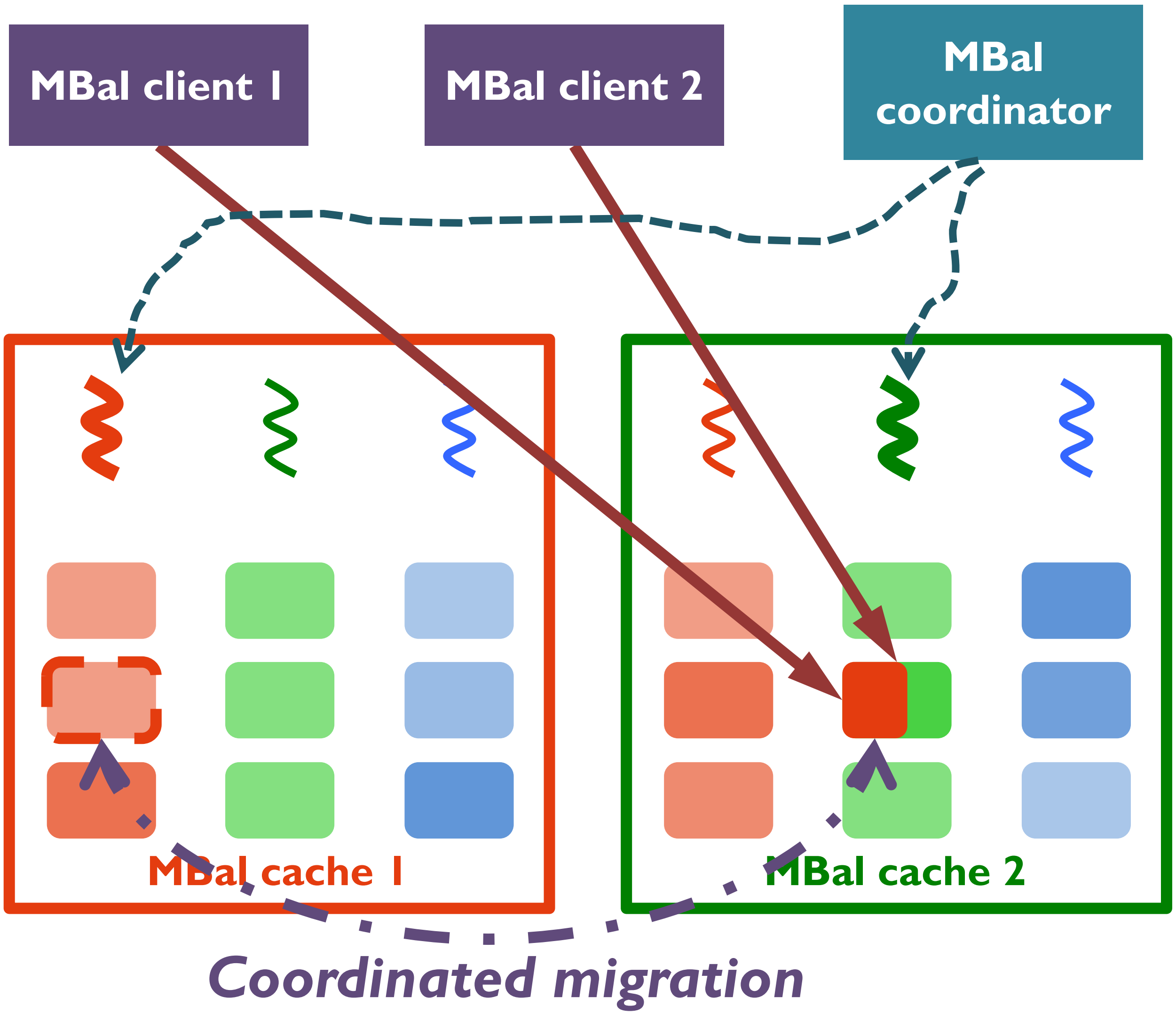
- TRIGGER?
  - # hot keys > $REPL_{HIGH}$
  - Not enough local headroom
- ACTION?
  - Migrate/swap cachelet(s) across servers
  - ILP
- FEATURES?
  - Coarse-grained
  - Permanent

Phase 3: coordinated cachelet migration



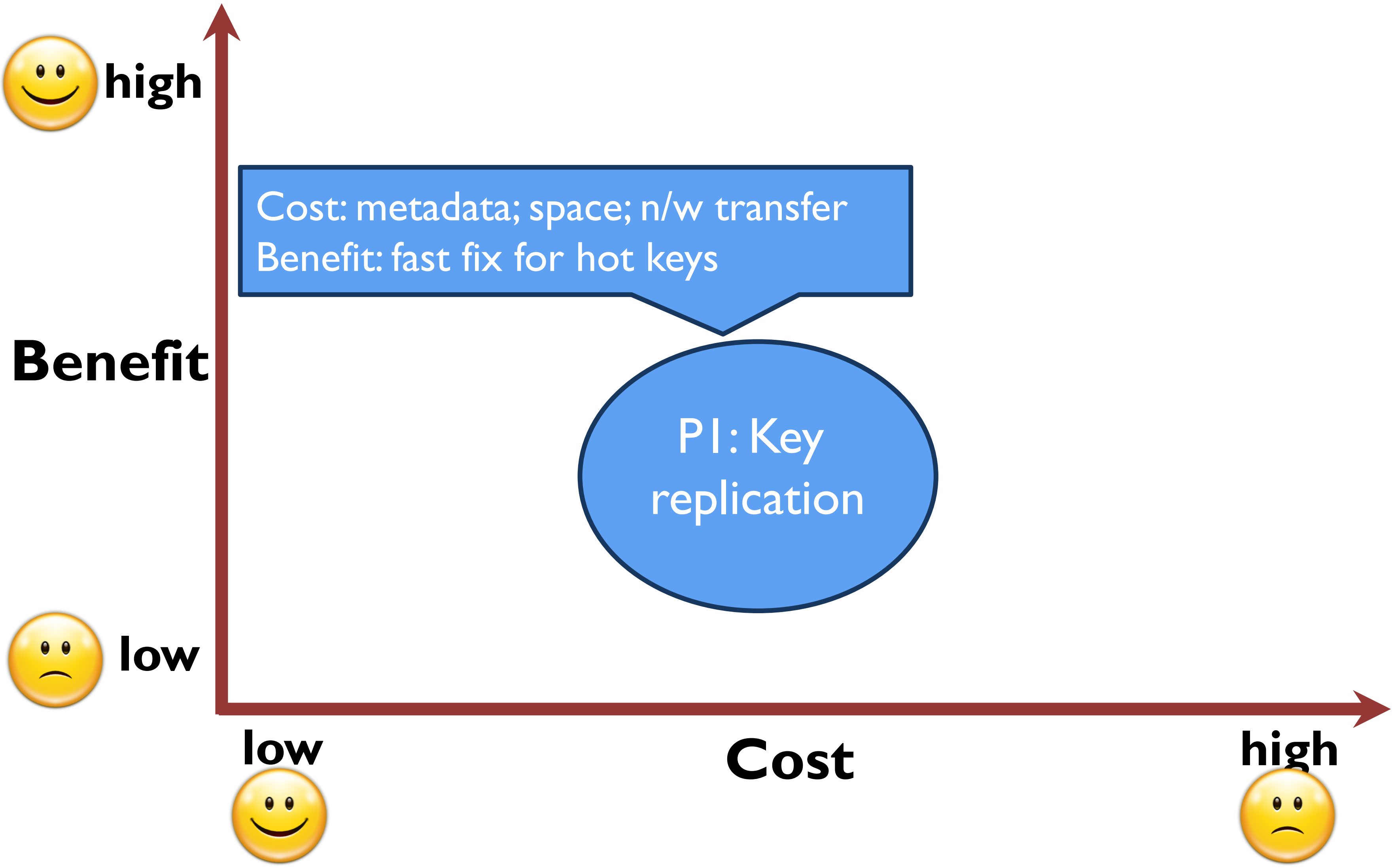
- TRIGGER?
  - # hot keys > $REPL_{HIGH}$
  - Not enough local headroom
- ACTION?
  - Migrate/swap cachelet(s) across servers
  - ILP
- FEATURES?
  - Coarse-grained
  - Permanent

Phase 3: coordinated cachelet migration

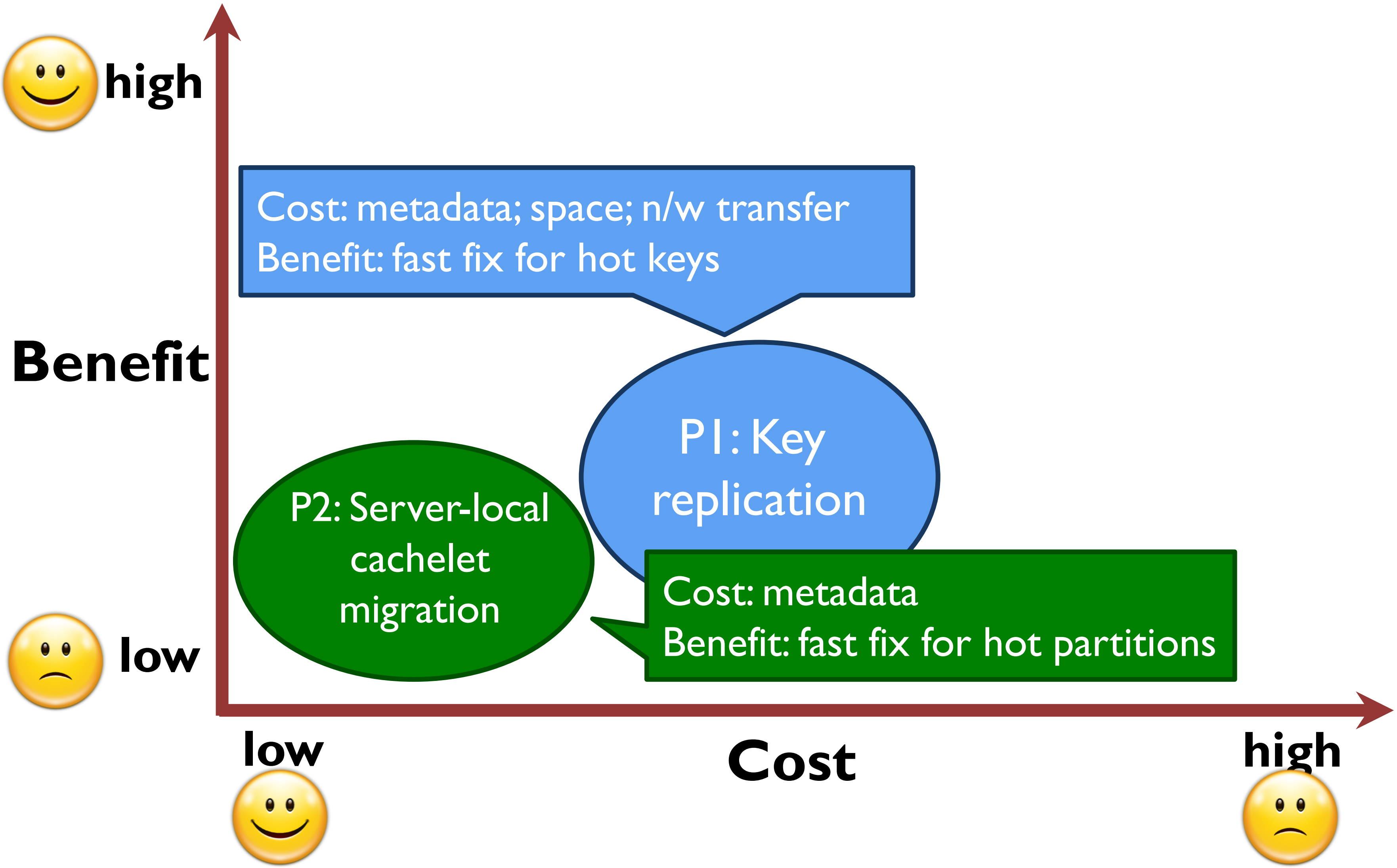


- TRIGGER?
  - # hot keys > $REPL_{HIGH}$
  - Not enough local headroom
- ACTION?
  - Migrate/swap cachelet(s) across servers
  - ILP
- FEATURES?
  - Coarse-grained
  - Permanent

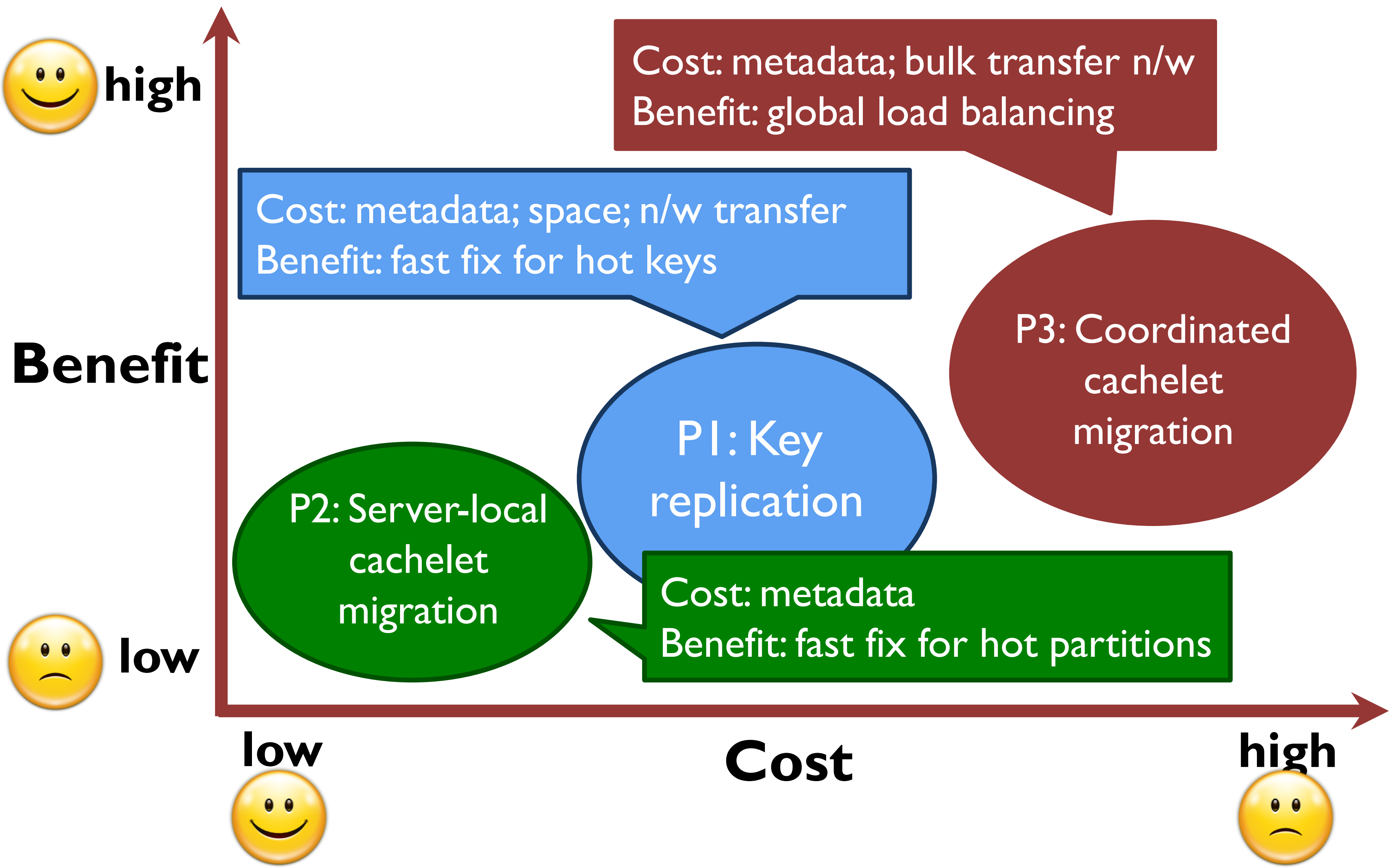
Cost/benefit tradeoffs in MBal



Cost/benefit tradeoffs in MBal



Cost/benefit tradeoffs in MBal



Outline

MBal cache design

MBal load balancer design

Evaluation

Related work

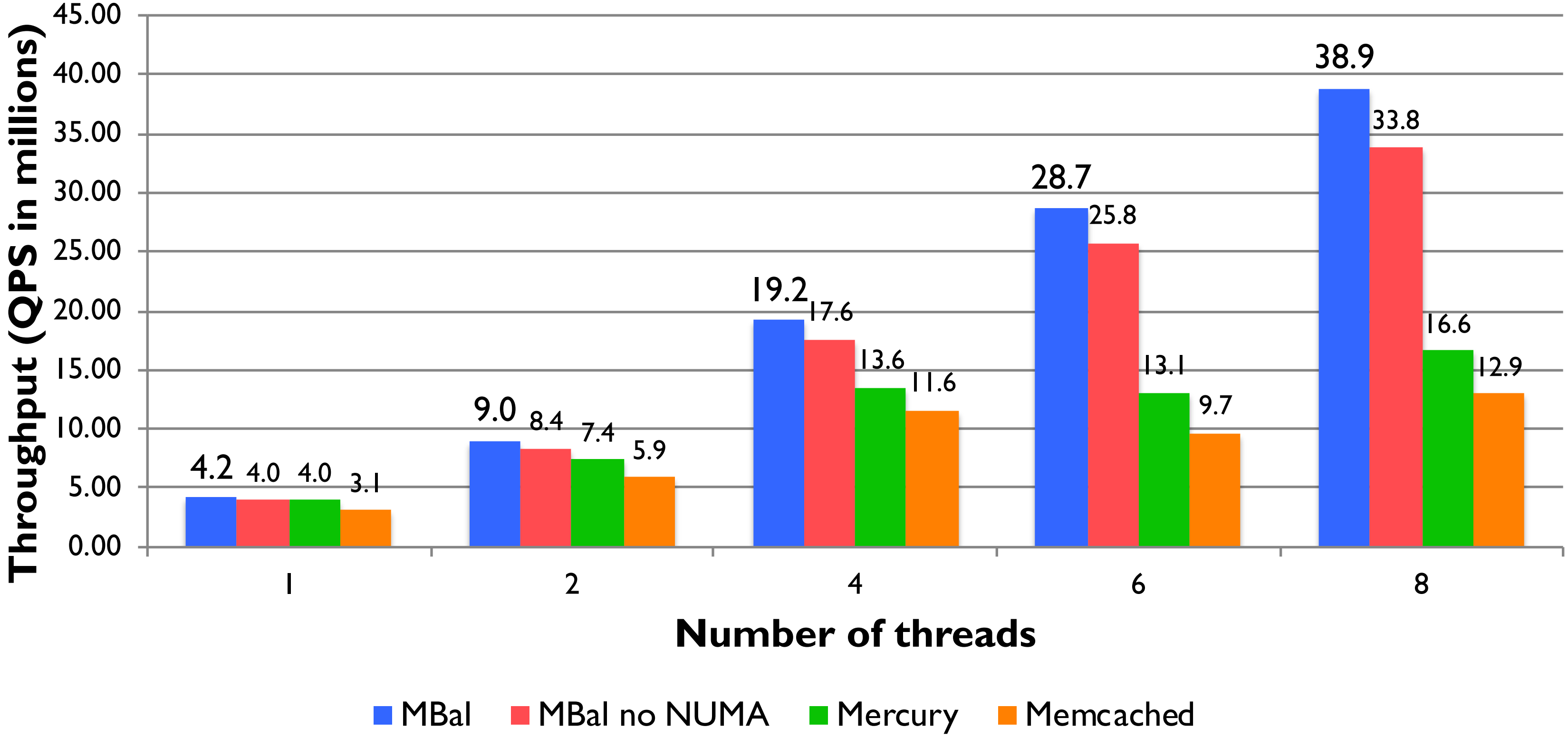
Methodology

- Scale-up cache performance tests
  - Local testbed
  - Single instance (8-core and 32-core server)

- End-to-end load balancer evaluation
  - 20-VM cluster (EC2, c3.large)

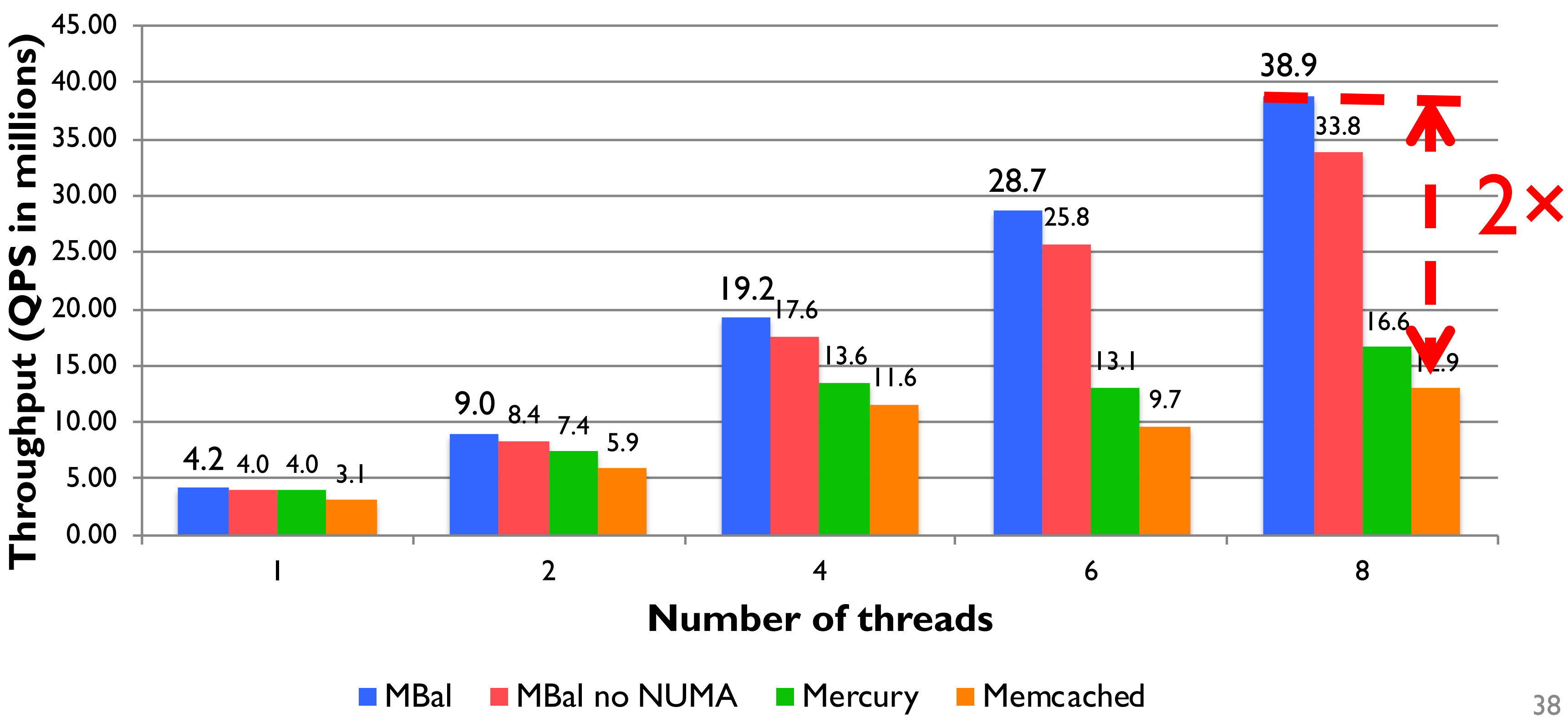
MBal evaluation – micro-benchmark

- 8-core 2.5GHz, 2×10MB L3 LLC, 64GB DRAM
- Uniform workload, 100% GET, 10B key 20B value
- Without network



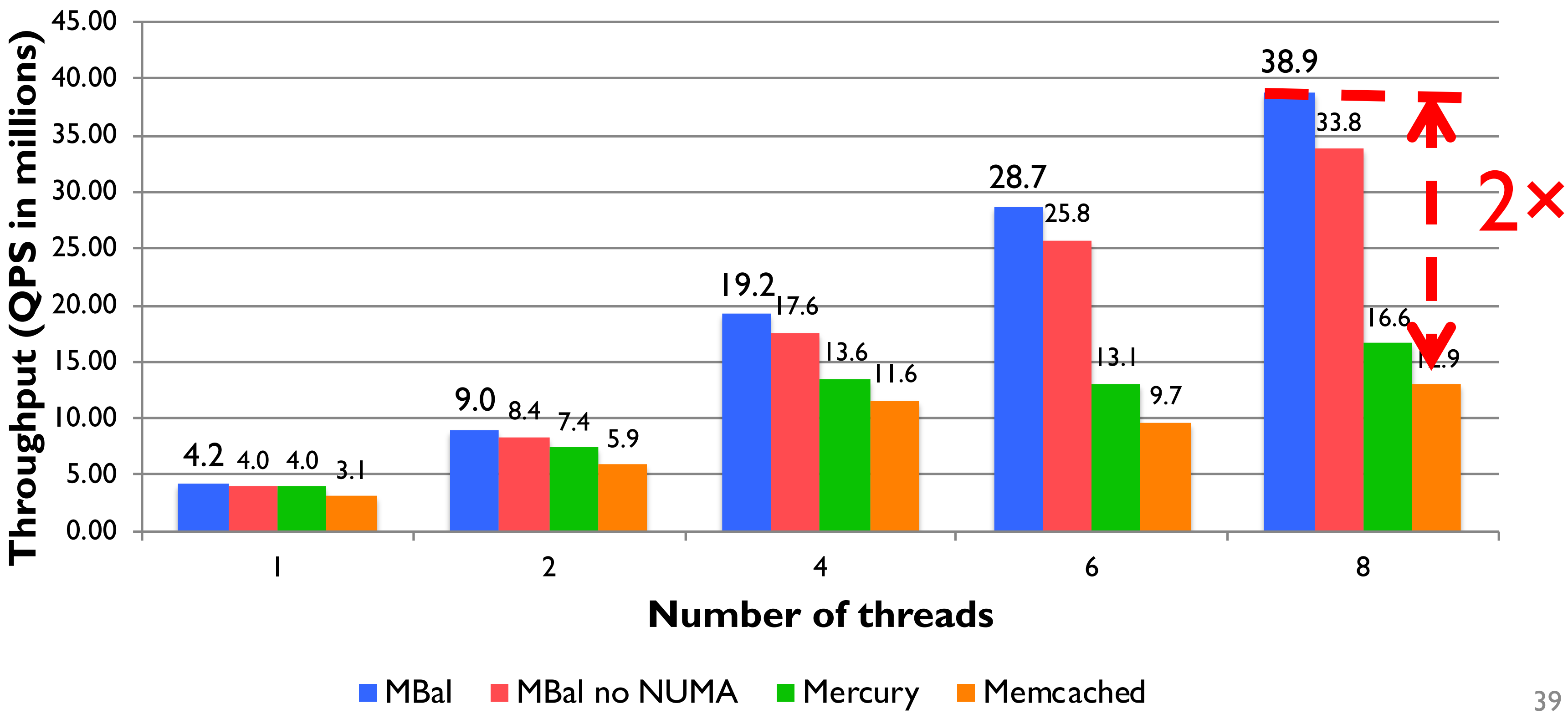
MBal evaluation – micro-benchmark

- 8-core 2.5GHz, 2×10MB L3 LLC, 64GB DRAM
- Uniform workload, 100% GET, 10B key 20B value
- Without network



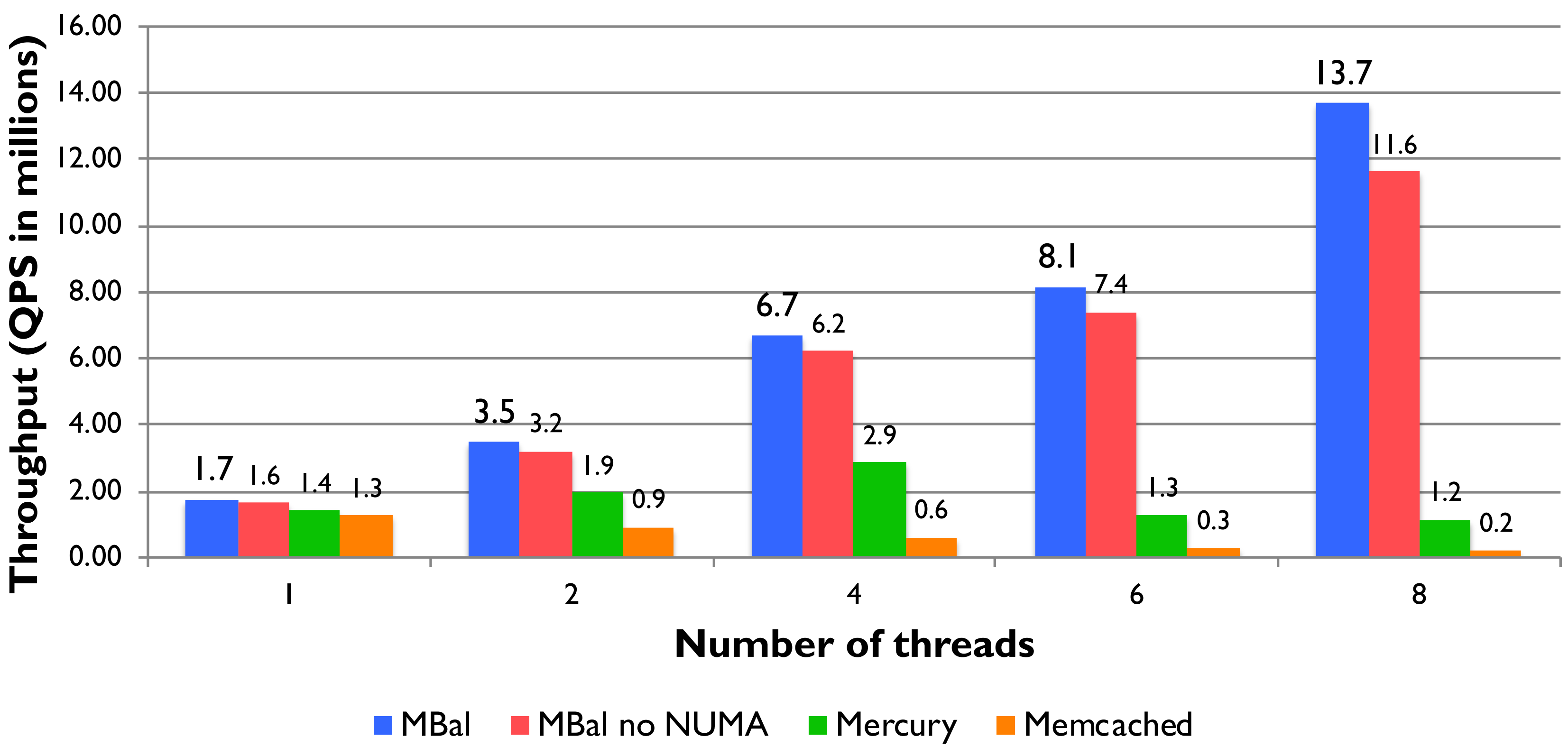
MBal evaluation – micro-benchmark

✓ MBal eliminates bucket-level lock contention!



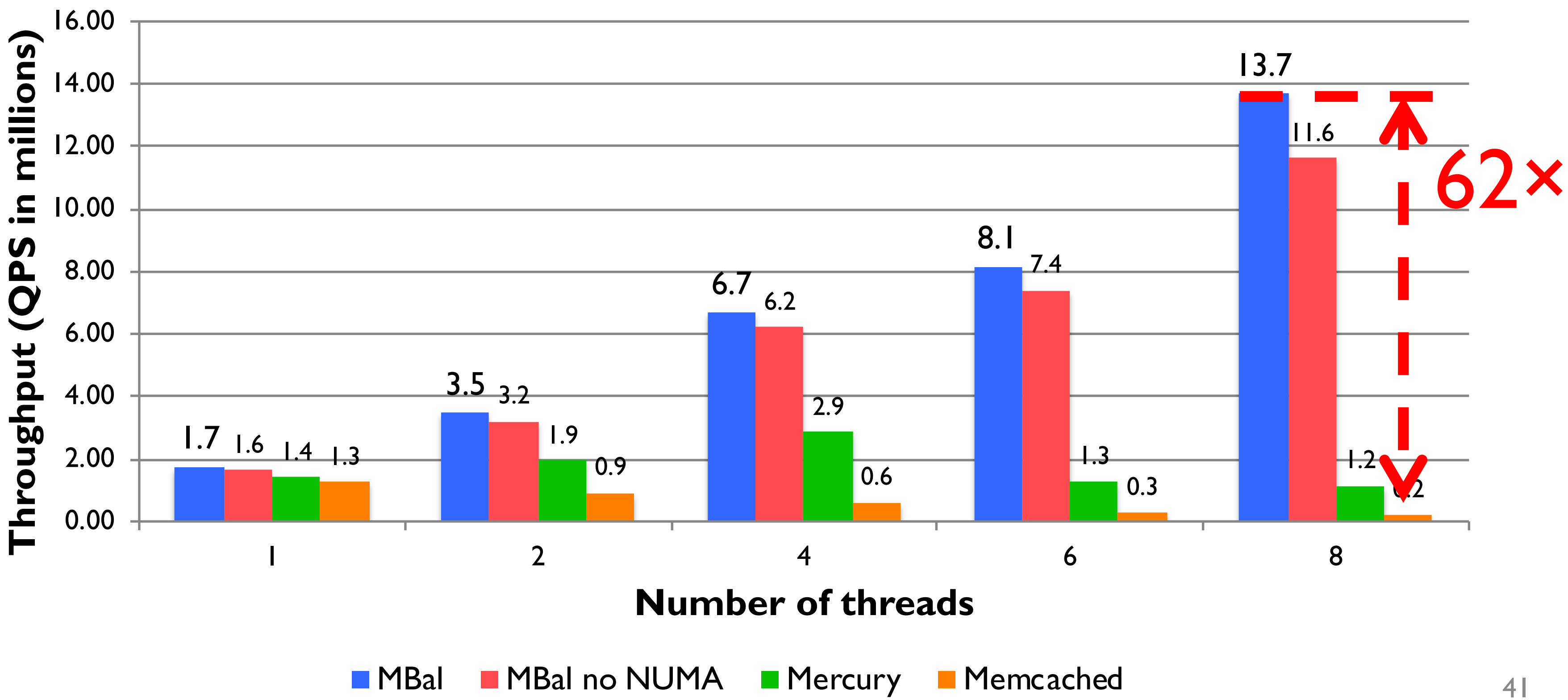
MBal evaluation – micro-benchmark

- 8-core 2.5GHz, 2×10MB L3 LLC, 64GB DRAM
- Uniform workload, 100% SET, 10B key 20B value
- Without network



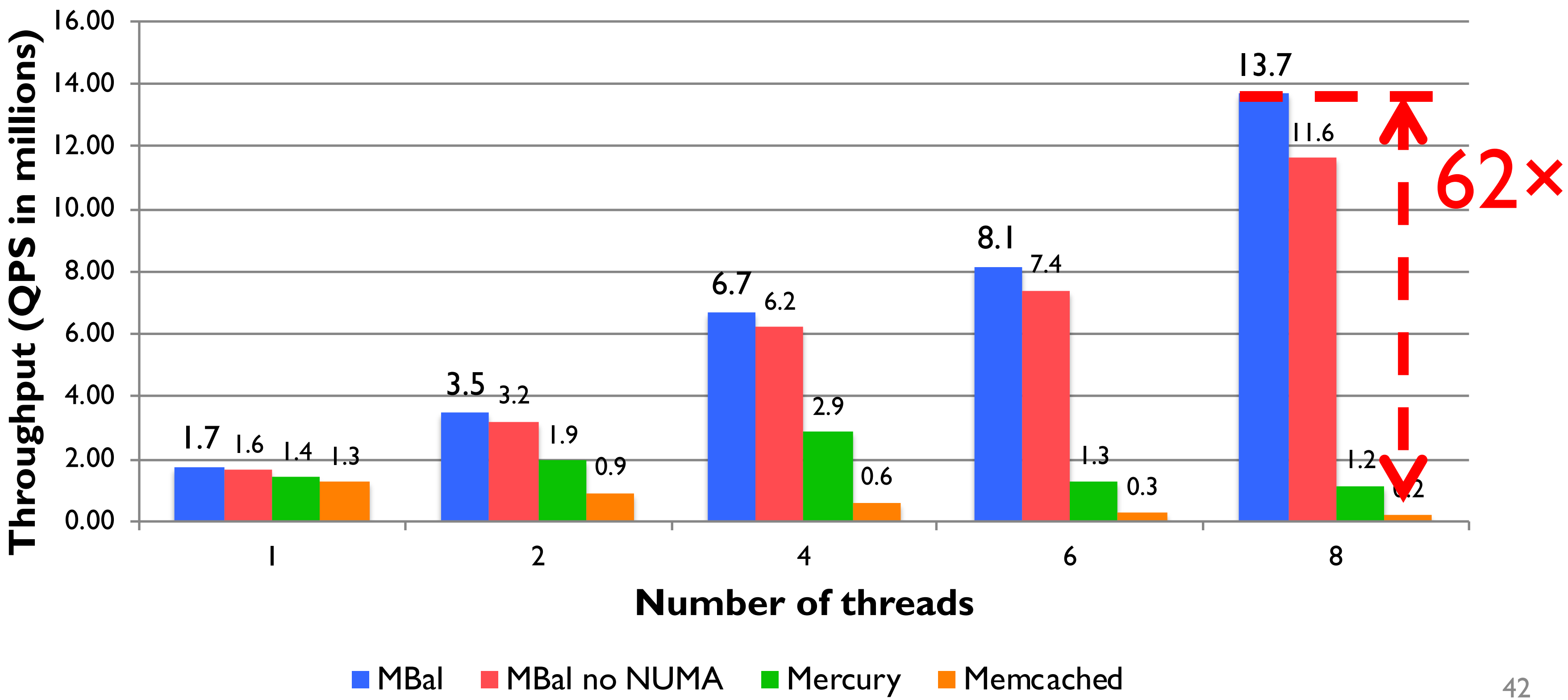
MBal evaluation – micro-benchmark

- 8-core 2.5GHz, 2×10MB L3 LLC, 64GB DRAM
- Uniform workload, 100% SET, 10B key 20B value
- Without network



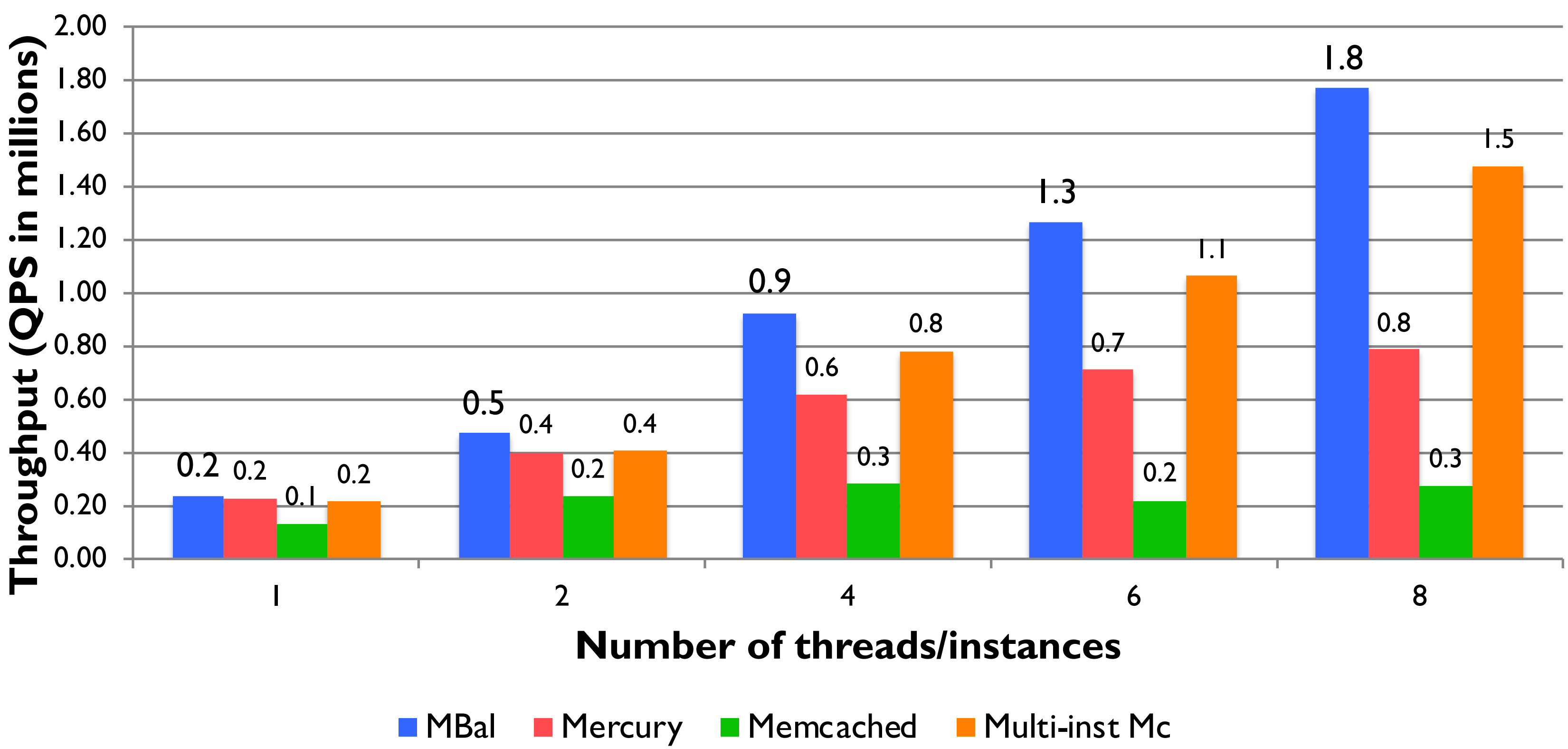
MBal evaluation – micro-benchmark

✓ MBal eliminates global cache lock contention!



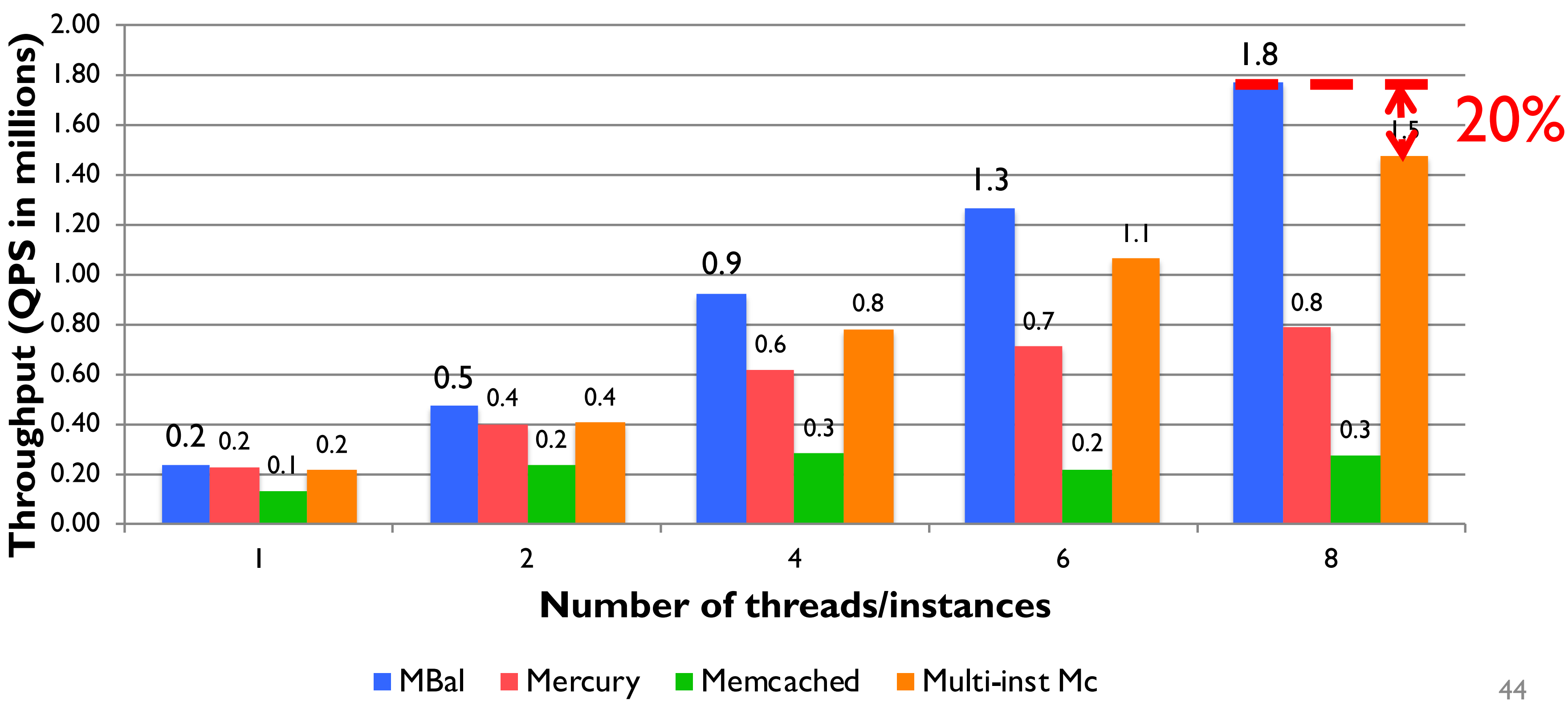
MBal evaluation – complete system

- 8-core 2.5GHz, 2×10MB L3 LLC, 64GB DRAM
- Zipfian workload, 75% GET, 10B key 20B value
- 10Gb Ethernet, MultiGET



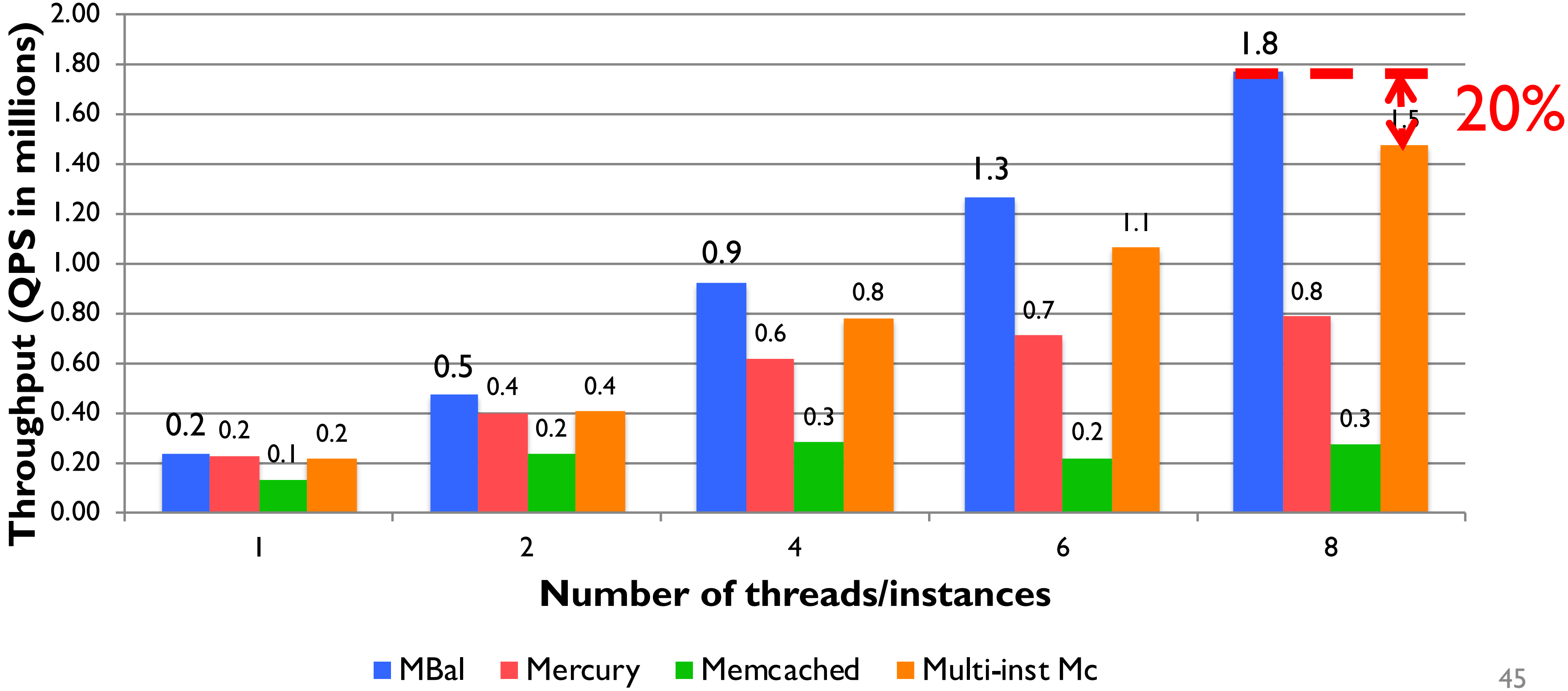
MBal evaluation – complete system

- 8-core 2.5GHz, 2×10MB L3 LLC, 64GB DRAM
- Zipfian workload, 75% GET, 10B key 20B value
- 10Gb Ethernet, MultiGET



MBal evaluation – complete system

✓ MBal uses lightweight CPU cache-aligned bucket locks!



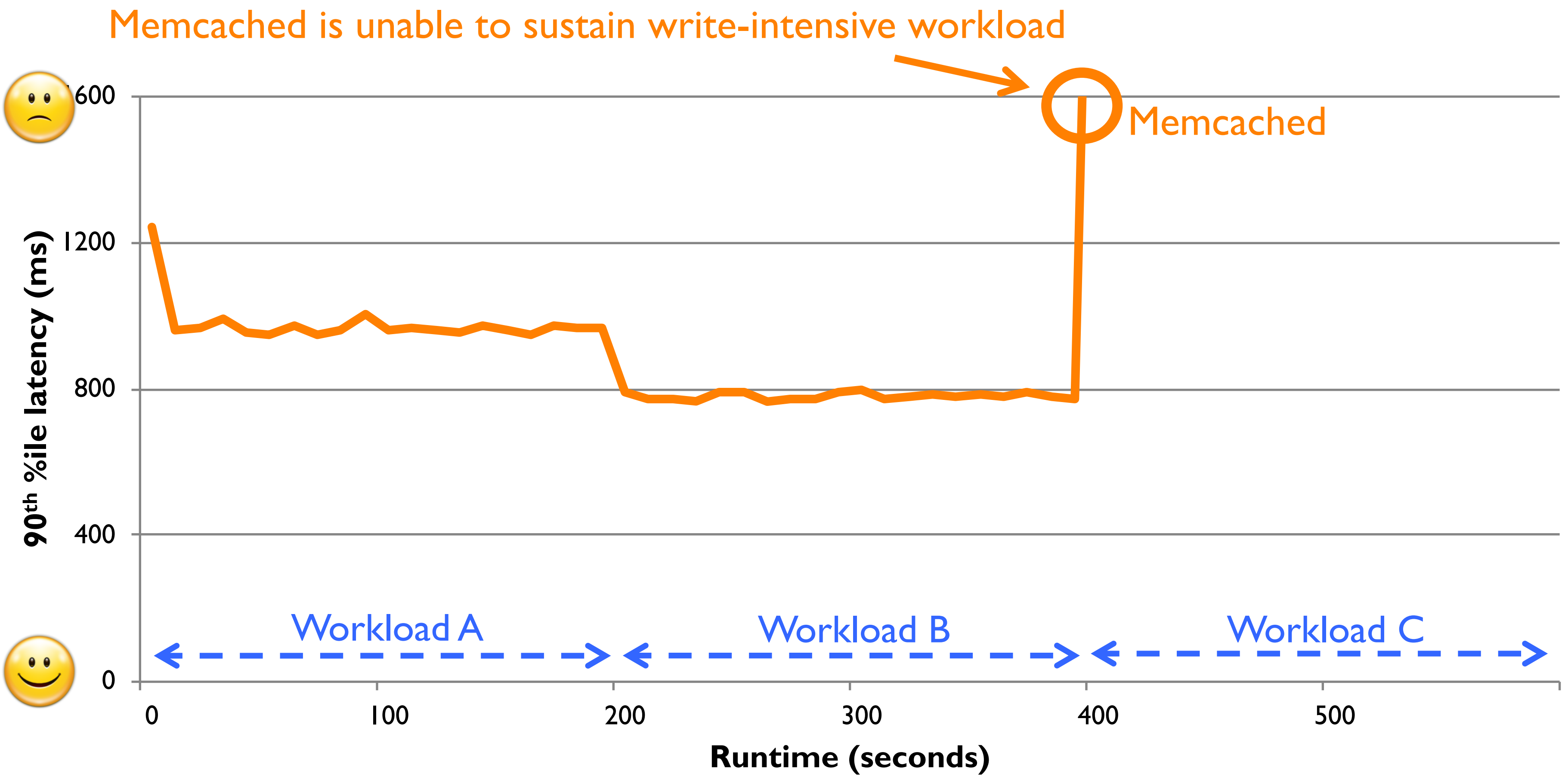
End-to-end load balancer evaluation

Workload	Characteristics	Application scenario
Workload A	**100% read, Zipfian**	**User account status info**
Workload B	**95% read, 5% update, hotspot (95% ops on 5% data)**	**Photo tagging**
Workload C	**50% read, 50% update, Zipfian**	**Session store recording actions**

Amazon EC2, us-west-2b, Clients on 36 instances (c3.2xlarge), 20-node VM cluster (c3.large)

Load balancer evaluation

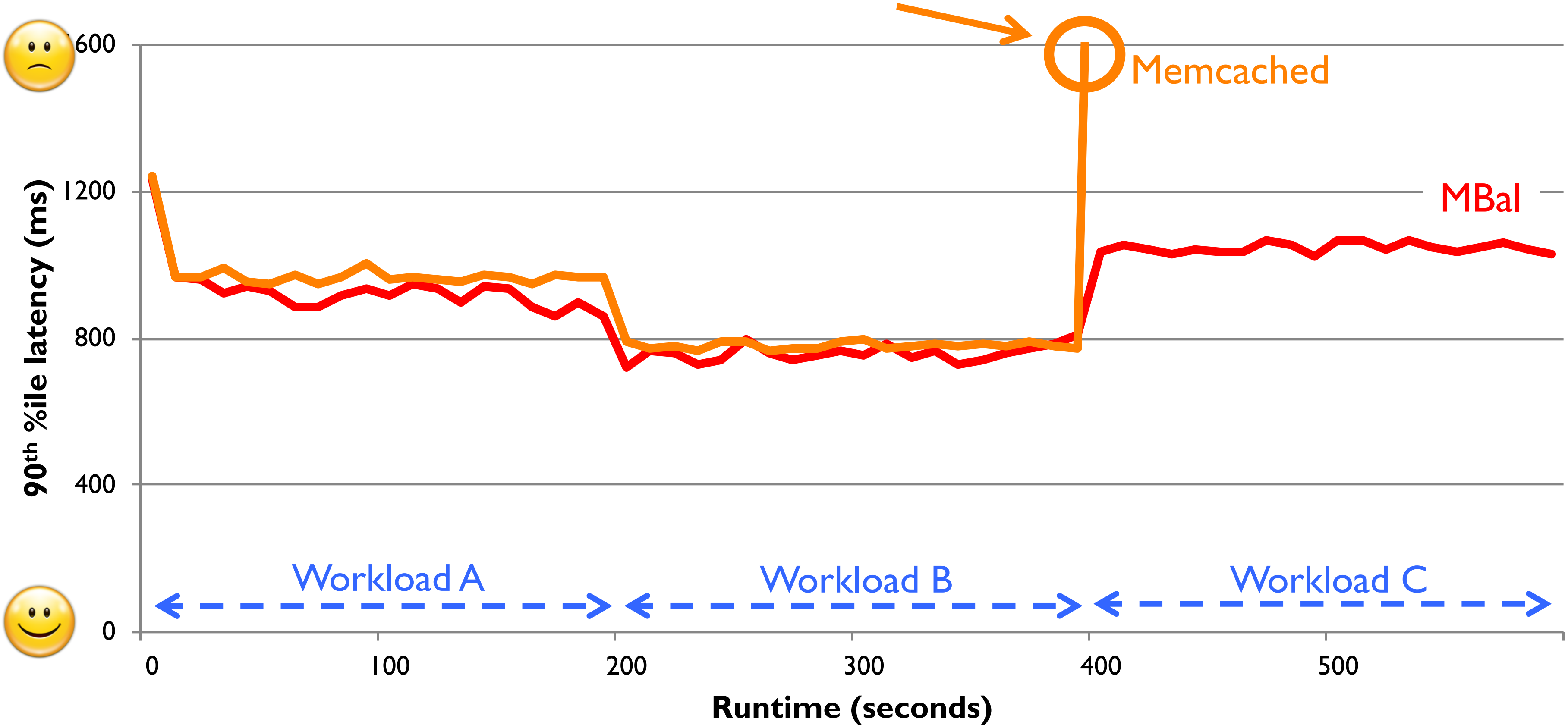
Workload	Characteristics
Workload A	100% read, Zipfian
Workload B	95% read, 5% update, hotspot
Workload C	50% read, 50% update, Zipfian



Load balancer evaluation

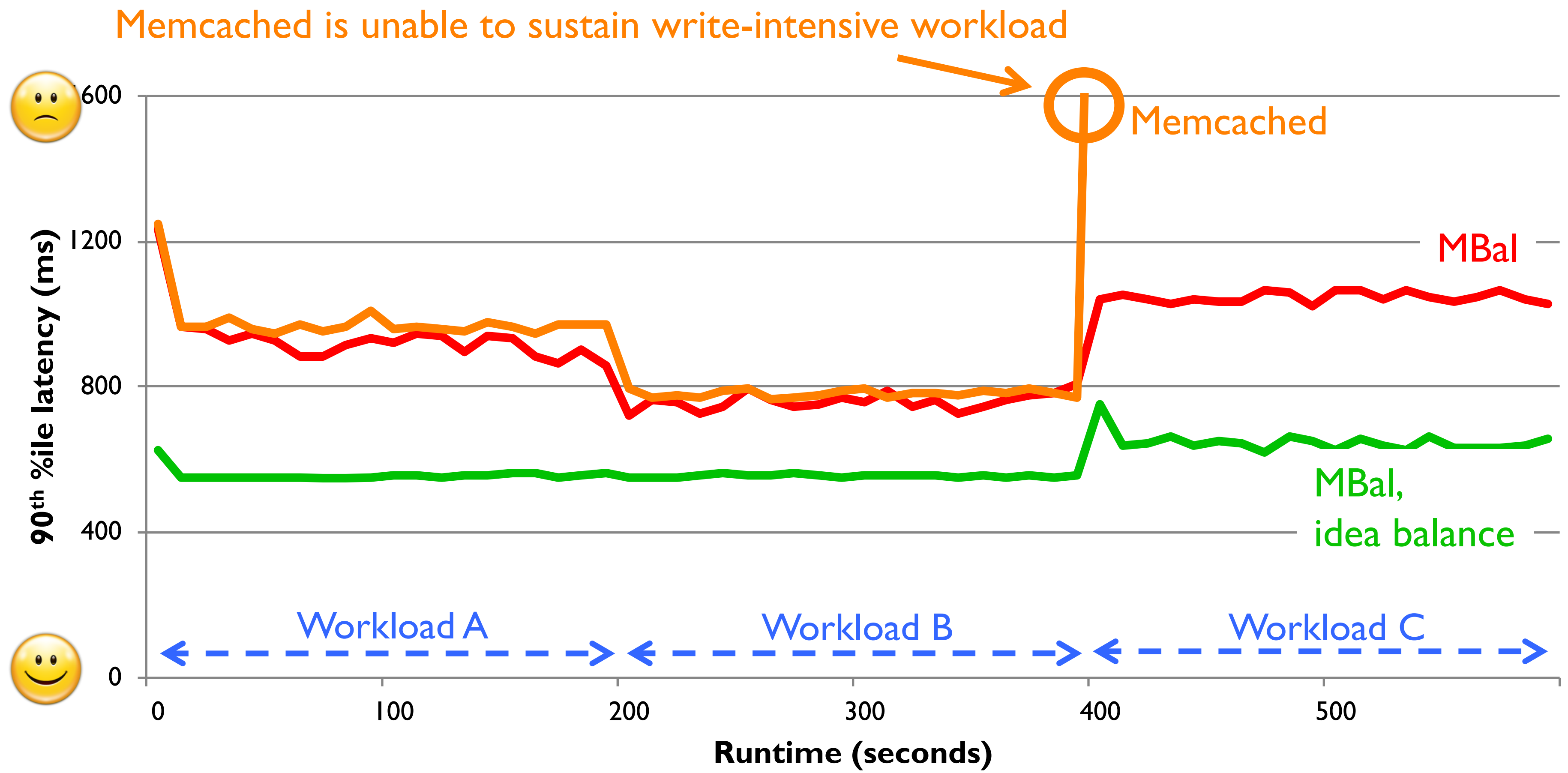
Workload	Characteristics
Workload A	100% read, Zipfian
Workload B	95% read, 5% update, hotspot
Workload C	50% read, 50% update, Zipfian

Memcached is unable to sustain write-intensive workload



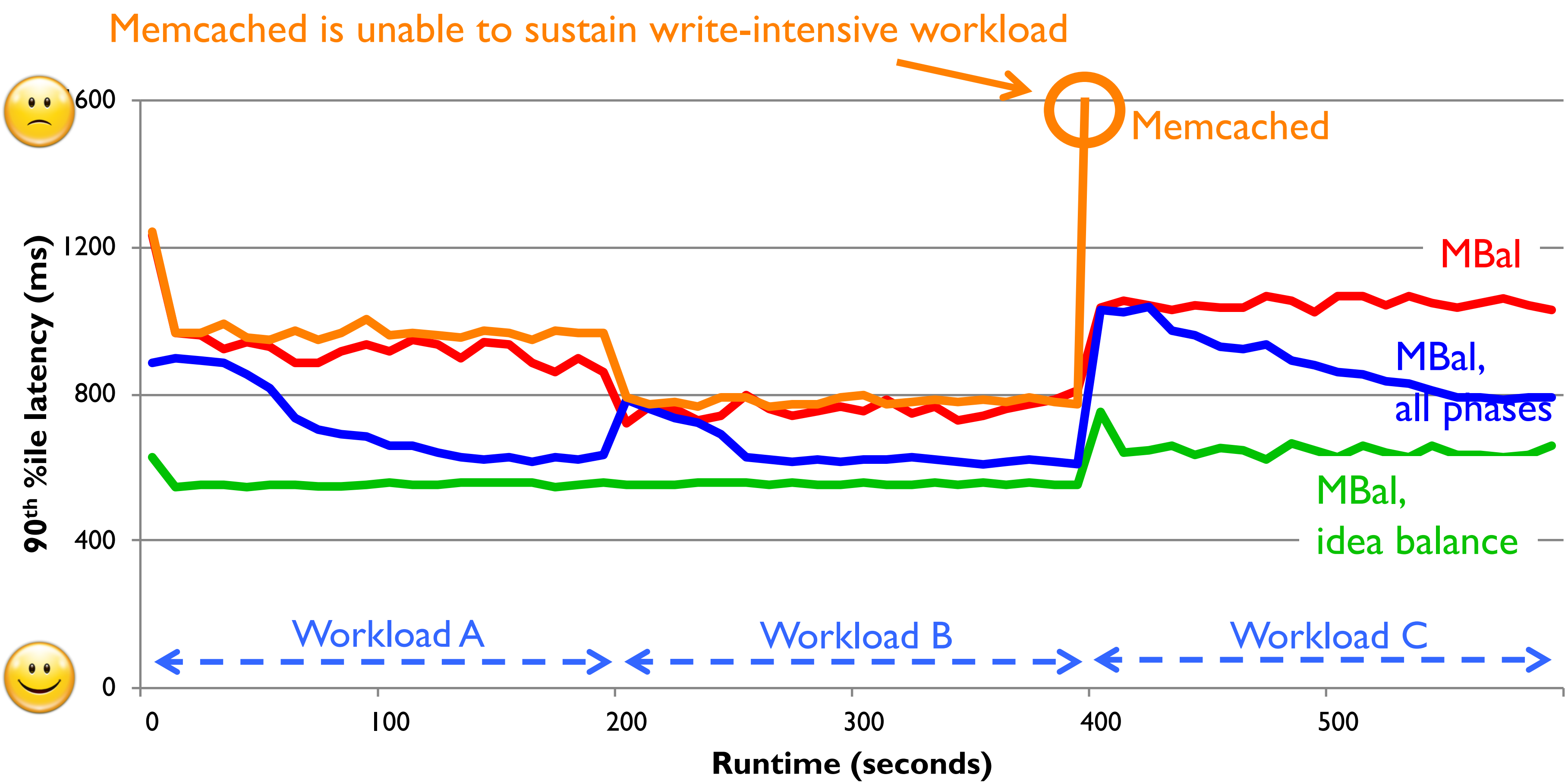
Load balancer evaluation

Workload	Characteristics
Workload A	100% read, Zipfian
Workload B	95% read, 5% update, hotspot
Workload C	50% read, 50% update, Zipfian



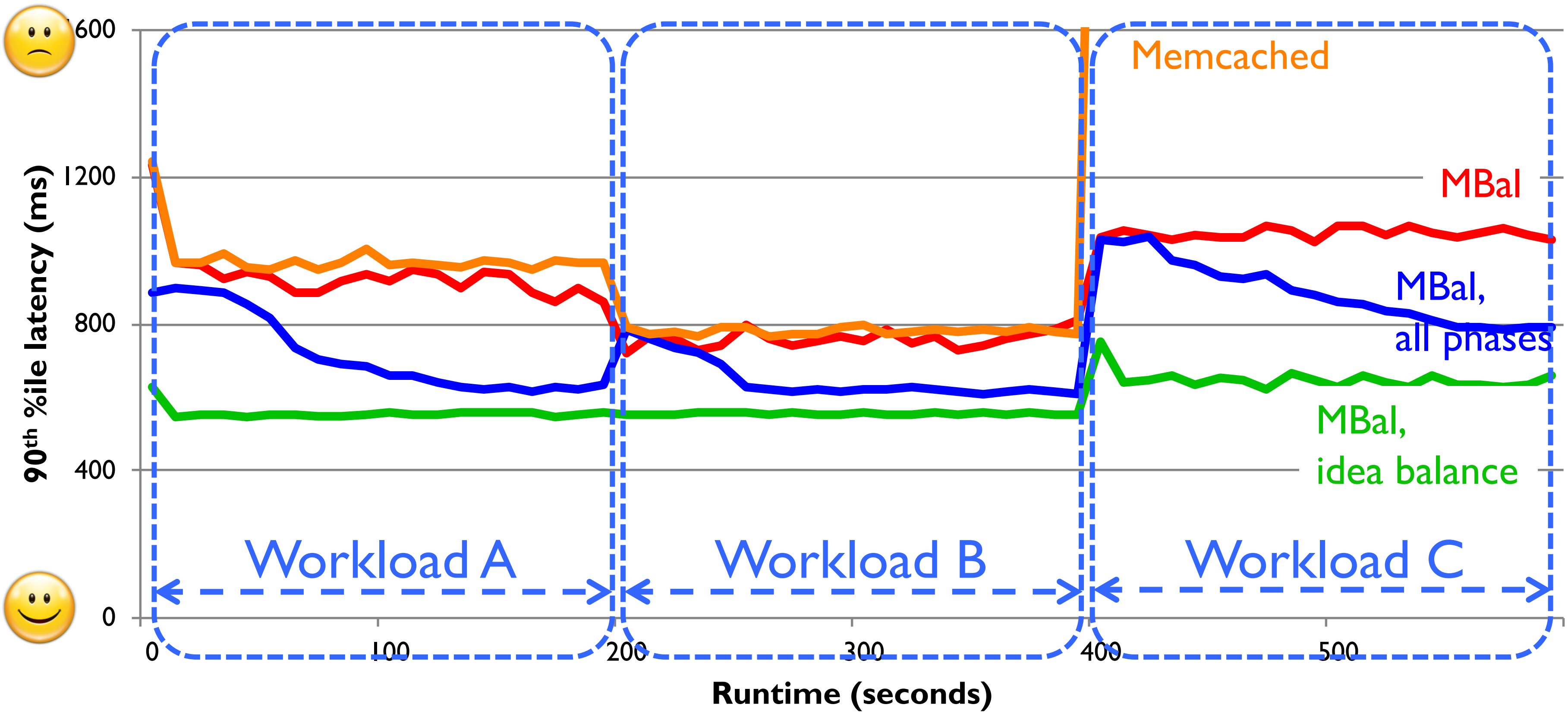
Load balancer evaluation

Workload	Characteristics
Workload A	100% read, Zipfian
Workload B	95% read, 5% update, hotspot
Workload C	50% read, 50% update, Zipfian



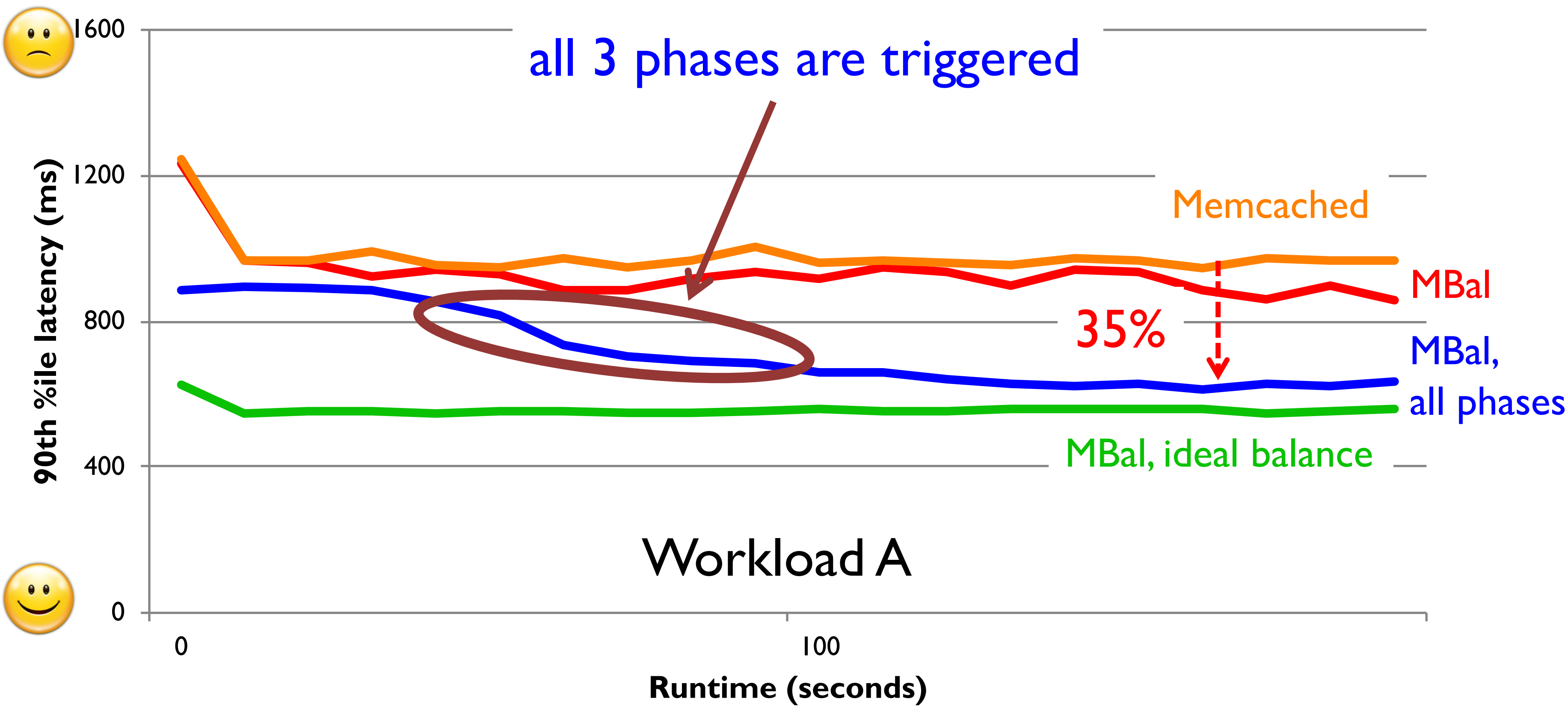
Load balancer evaluation

Workload	Characteristics
Workload A	100% read, Zipfian
Workload B	95% read, 5% update, hotspot
Workload C	50% read, 50% update, Zipfian



Load balancer evaluation

Workload	Characteristics
Workload A	100% read, **Zipfian**
Workload B	95% read, 5% update, hotspot
Workload C	50% read, 50% update, Zipfian

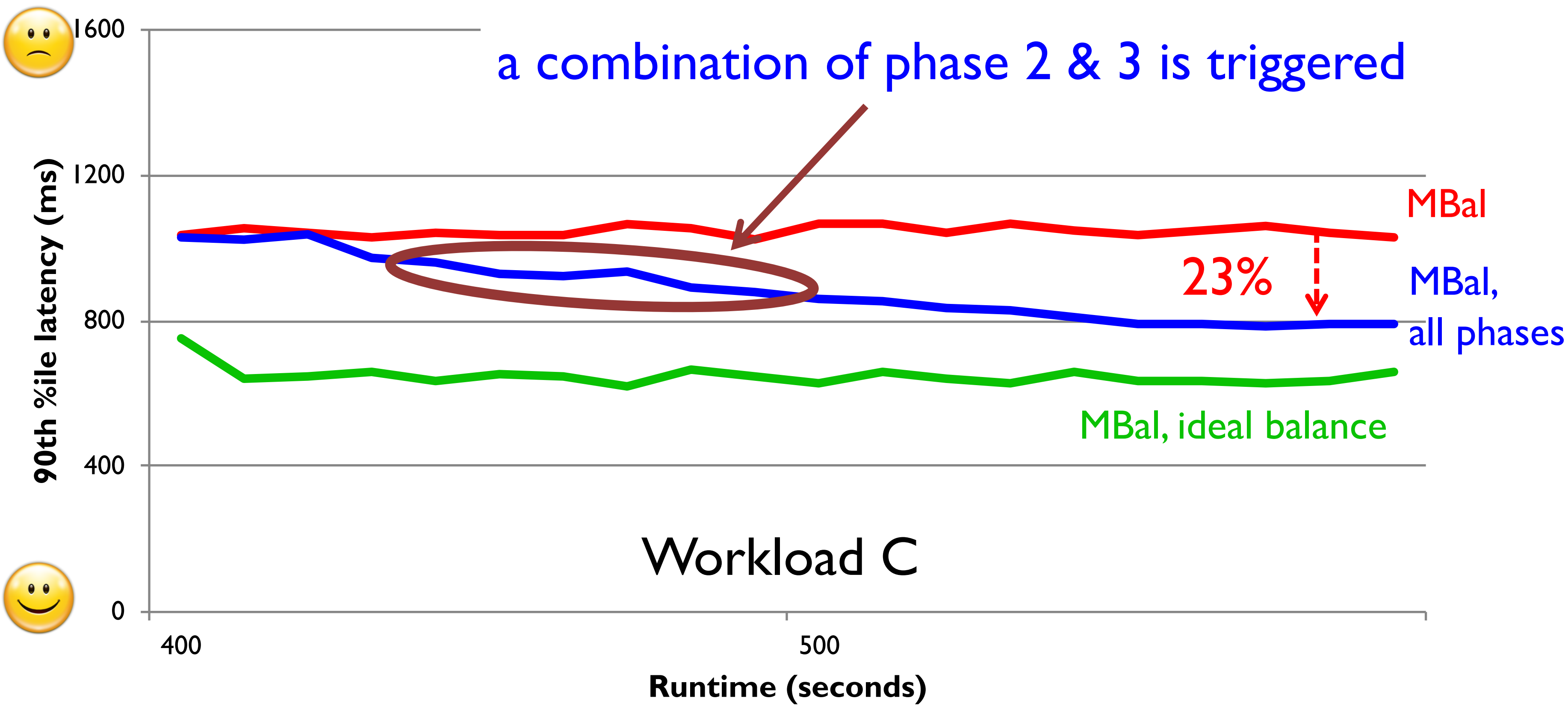


Load balancer evaluation

Workload	Characteristics
Workload A	100% read, Zipfian
Workload B	95% read, 5% update, **hotspot**
Workload C	50% read, 50% update, Zipfian

Load balancer evaluation

Workload	Characteristics
Workload A	100% read, Zipfian
Workload	95% read, 5% update, hotspot
Workload C	50% read, 50% update, **Zipfian**



Summary of results

- MBal fine-grained partitioning design
  - **2×** more QPS for GETs
  - **62×** more QPS for SETs

- MBal multi-phase load balancer
  - **35%** lower tail latency
  - **20%** higher throughput

  → Effectively improves QPS/$

Outline

MBal cache design

MBal load balancer design

Evaluation

Related work

Related work

- High performance in-memory KV store
  - Masstree [**EuroSys'12**], MemC3 [**NSDI'12**], MICA [**NSDI'14**]
- Storage load balancing
  - DHT (Pastry [**Middleware'01**], CFS [**SOSP'01**], Chord [**SIGCOMM'01**]), Proteus [**ICDCS'13**]
- Access load balancing
  - SmallCache [**SoCC'11**], Chronos [**SoCC'12**], SPORE [**SoCC'13**], Streaming Analytics [**Feedback'14**]

Conclusions

- Fine-grained, horizontal partitioning of in-memory data structure
  - eliminates sync overhead
  - enables load balancing
- MBal synthesizes three replication and migration techniques into a holistic system
  - reduces load imbalance
  - Improves tail latency