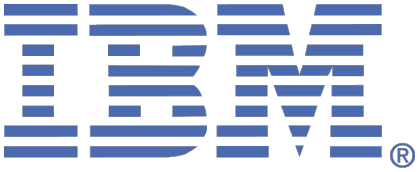


An In-Memory Object Caching Framework with Adaptive Load Balancing

Yue Cheng (Virginia Tech)

Aayush Gupta (IBM Research – Almaden)

Ali R. Butt (Virginia Tech)

In-memory caching in datacenters

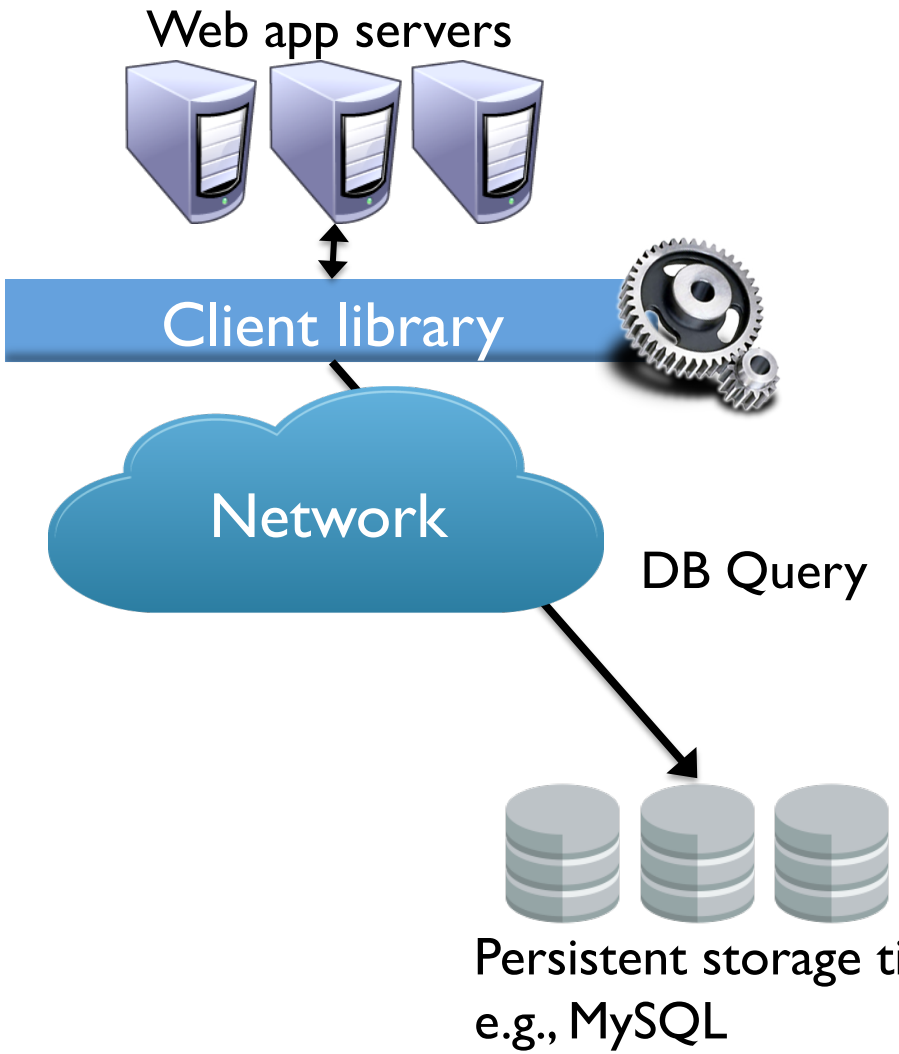
Local deployment

Cloud deployment

In-memory caching in datacenters

Local deployment

Cloud deployment

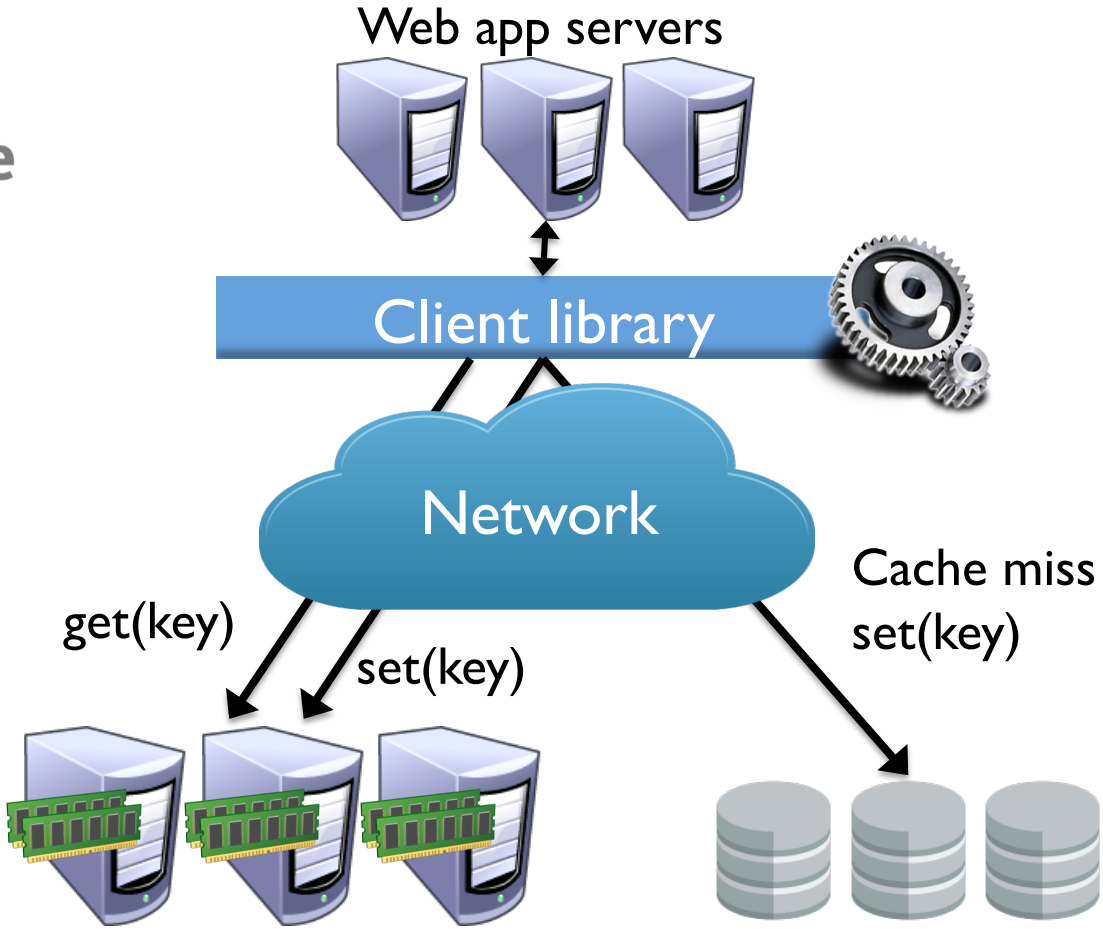


In-memory caching in datacenters

Local deployment

Cloud deployment

Amazon ElastiCache



In-memory caching is desirable

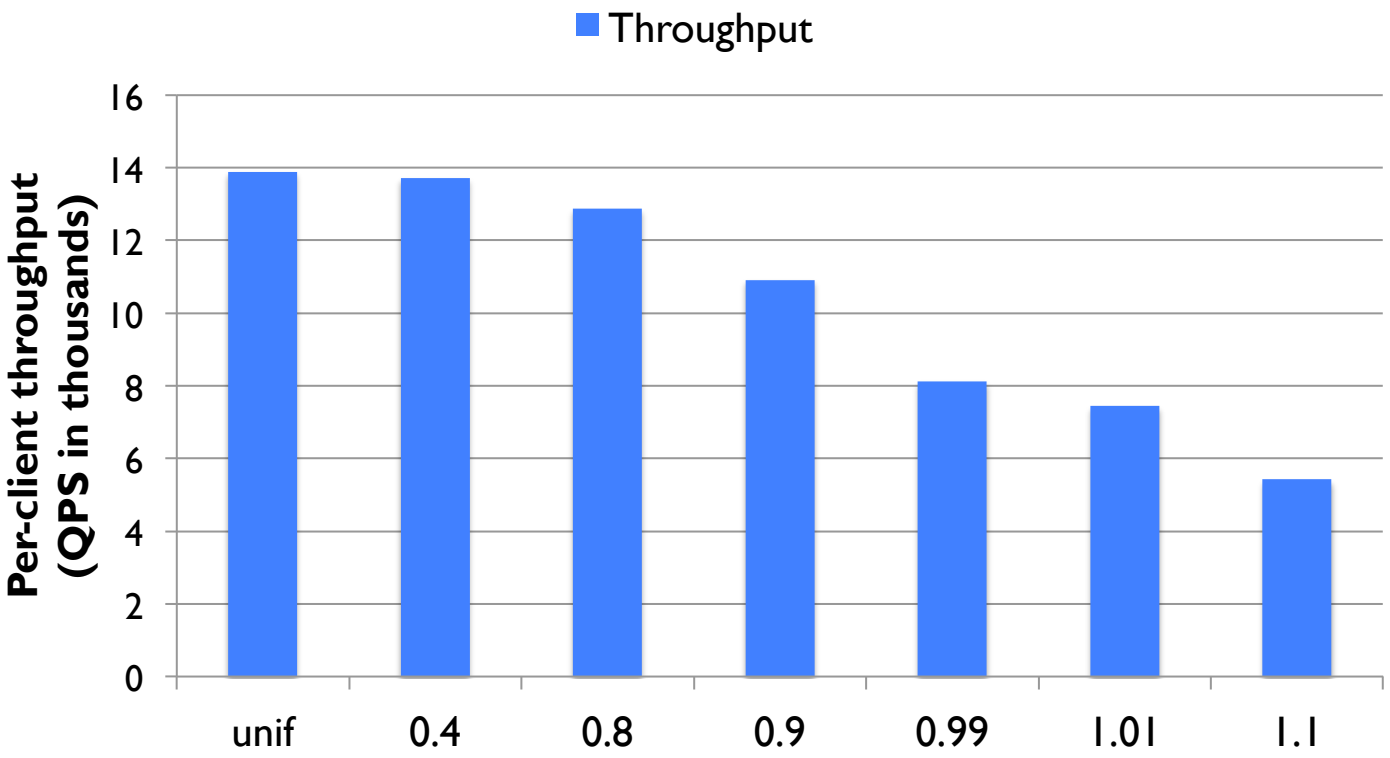
- Offers high performance
- Enables quick deployment
- Provides ease of use
- Supports elastic scale-out

In-memory caching is desirable

- Offers high performance
- Enables quick deployment
- Provides ease of use
- Supports elastic scale-out

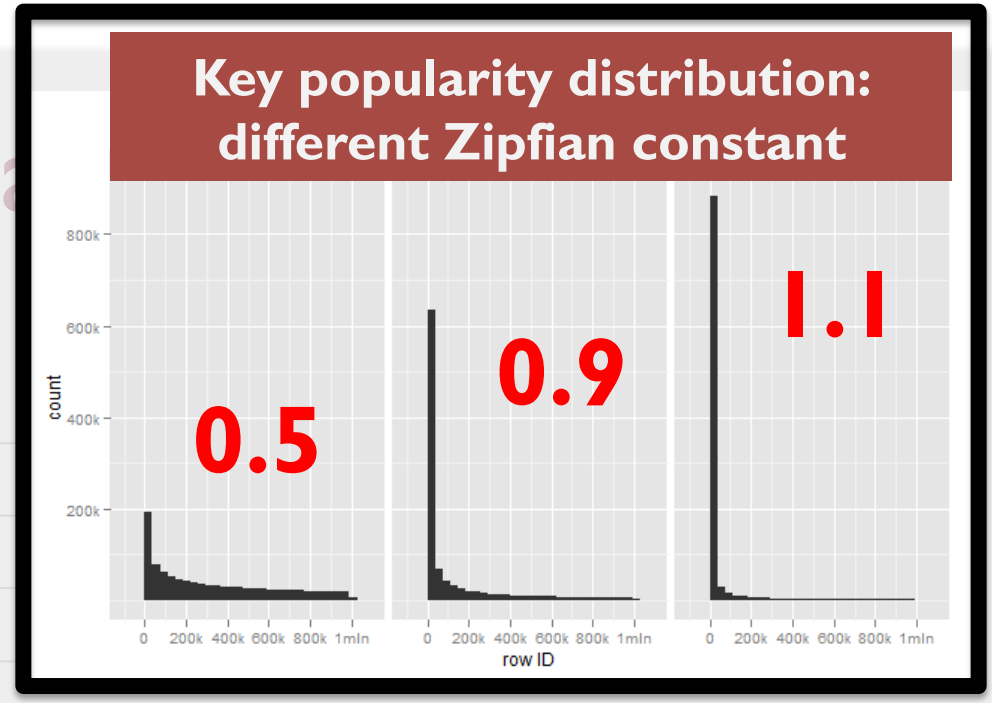
- **Problem:** Load imbalance impacts performance

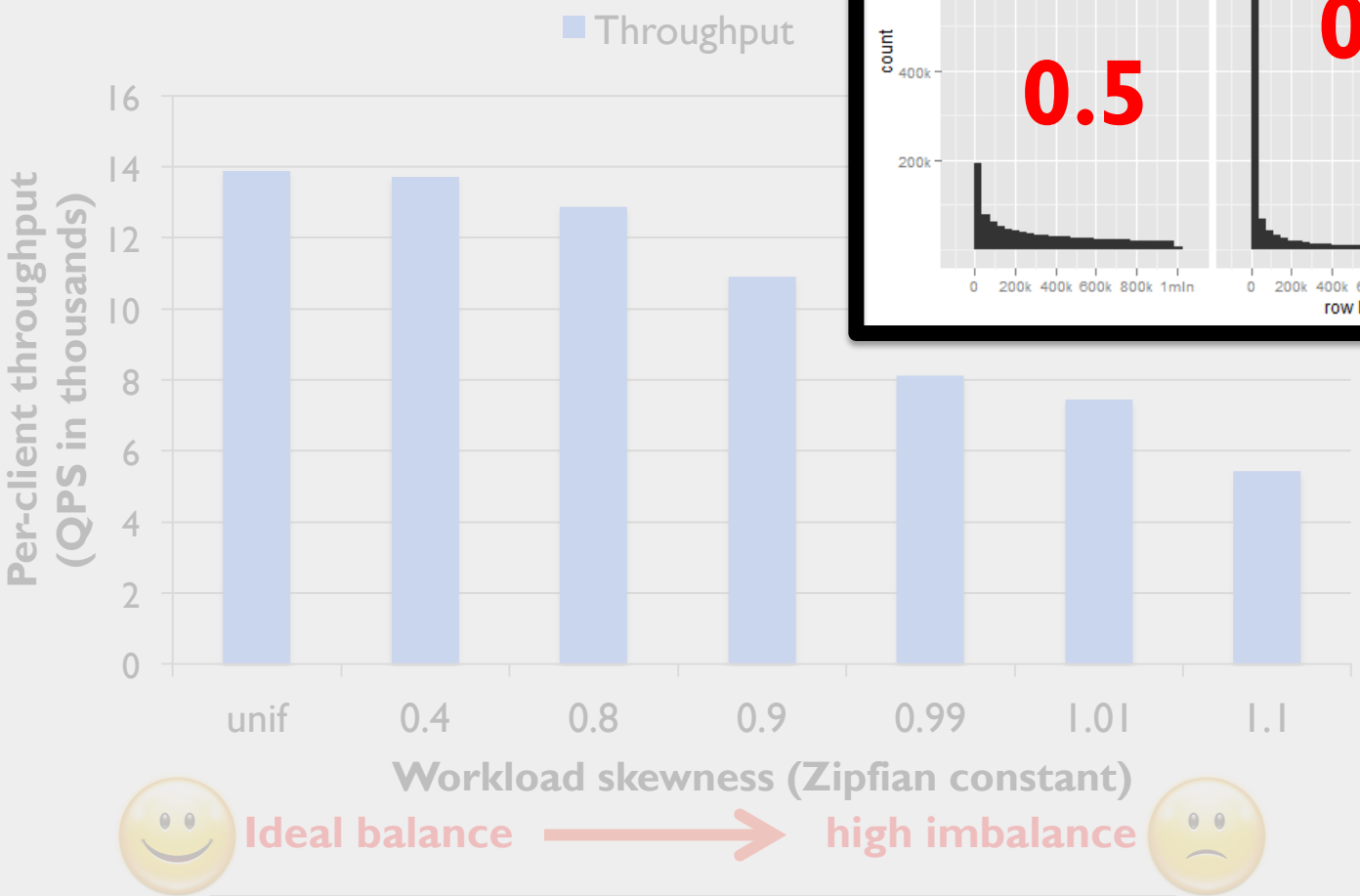
Access load imbalance



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

Access load imba

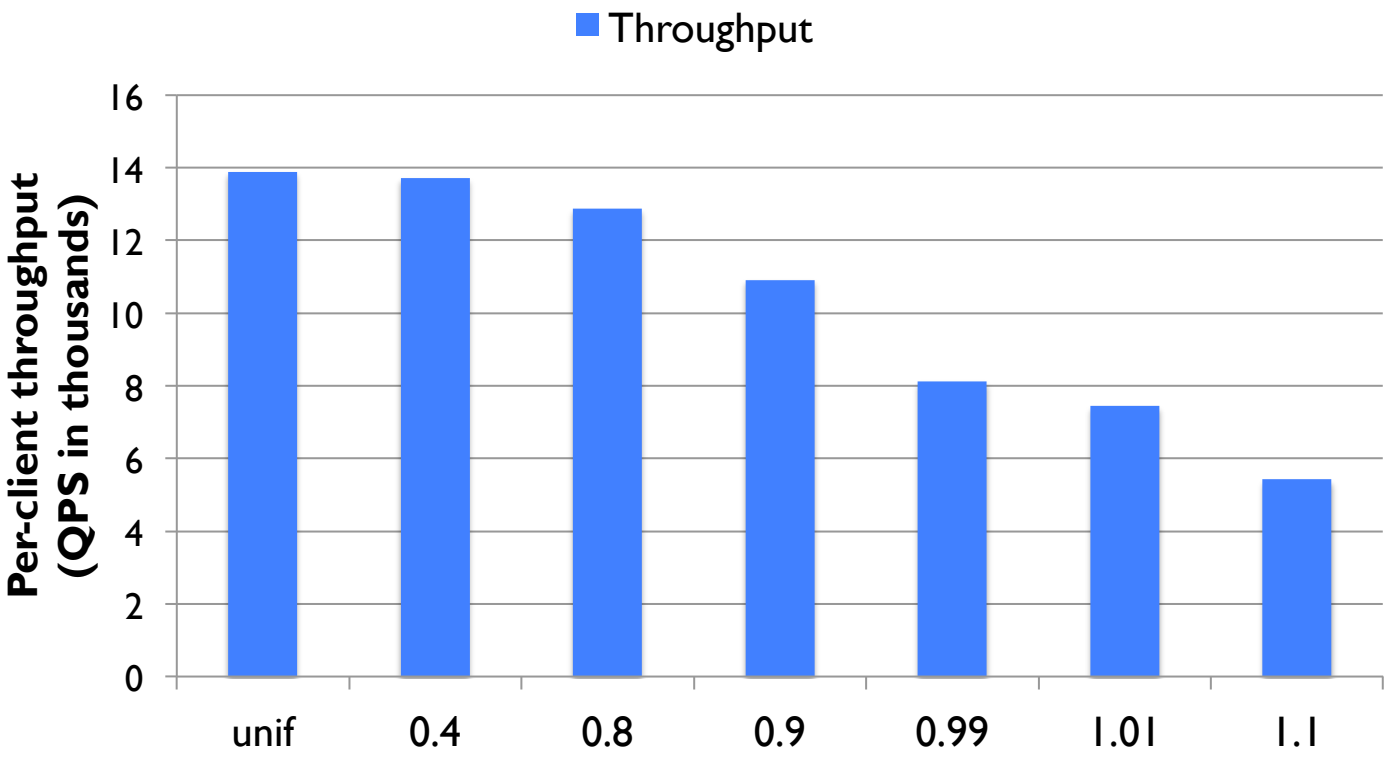




95% GET, 5% SET, Zipfian, 20 cache servers

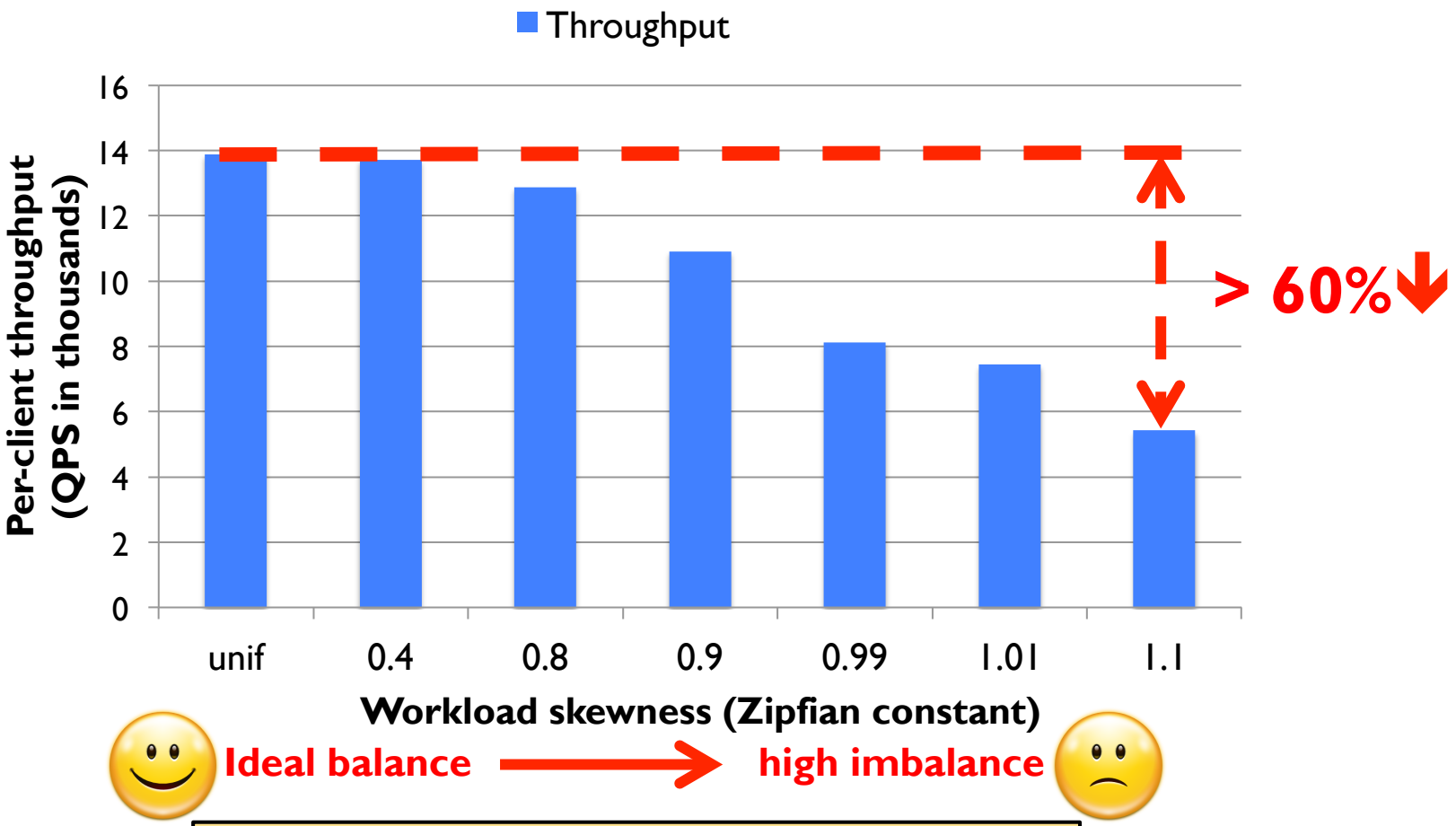
* http://www.percona.com/blog/2012/05/09/new-distribution-of-random-generator-for-sysbench-zipf/

Access load imbalance



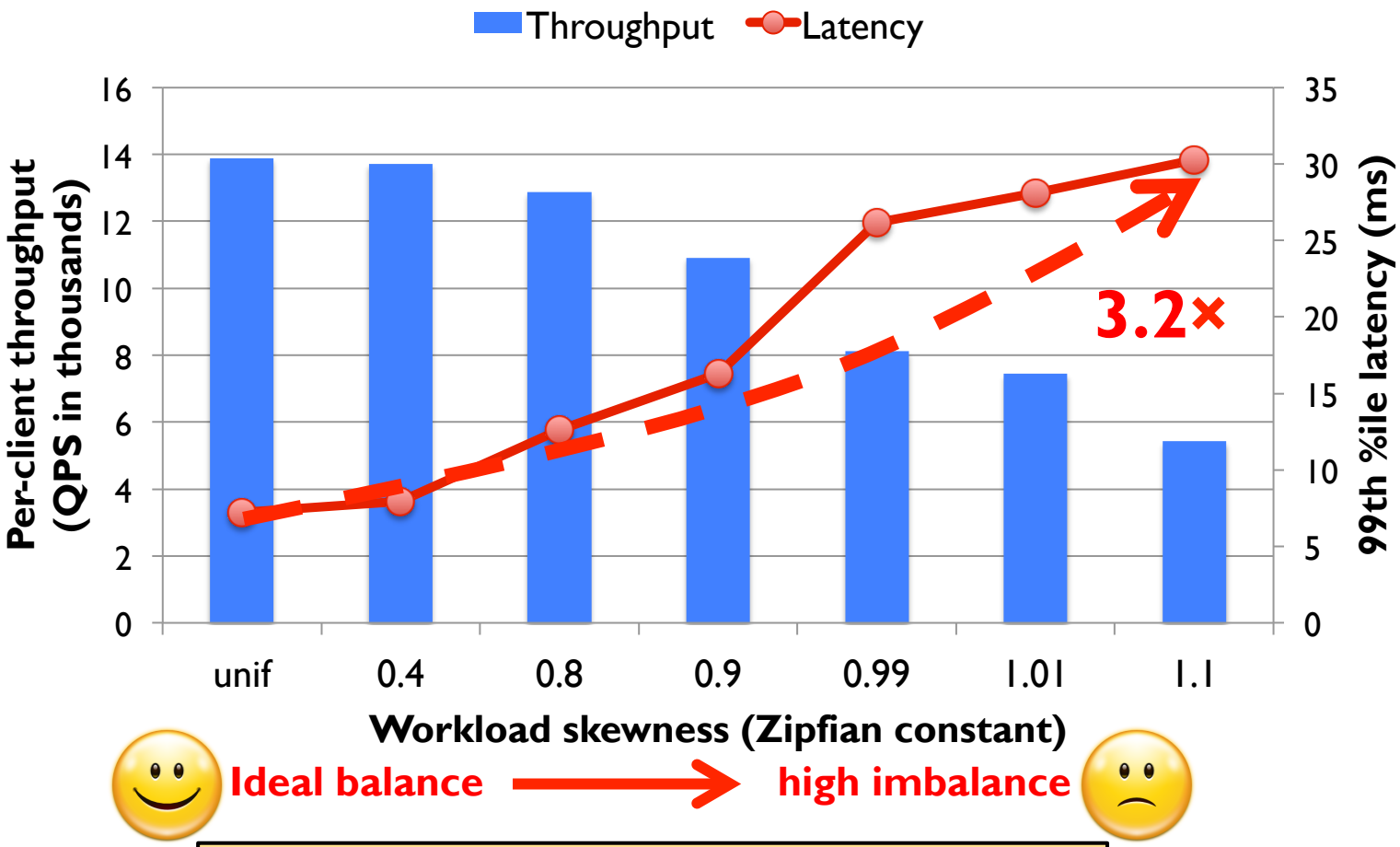
95% GET, 5% SET, Zipfian, 20 cache servers

Access load imbalance



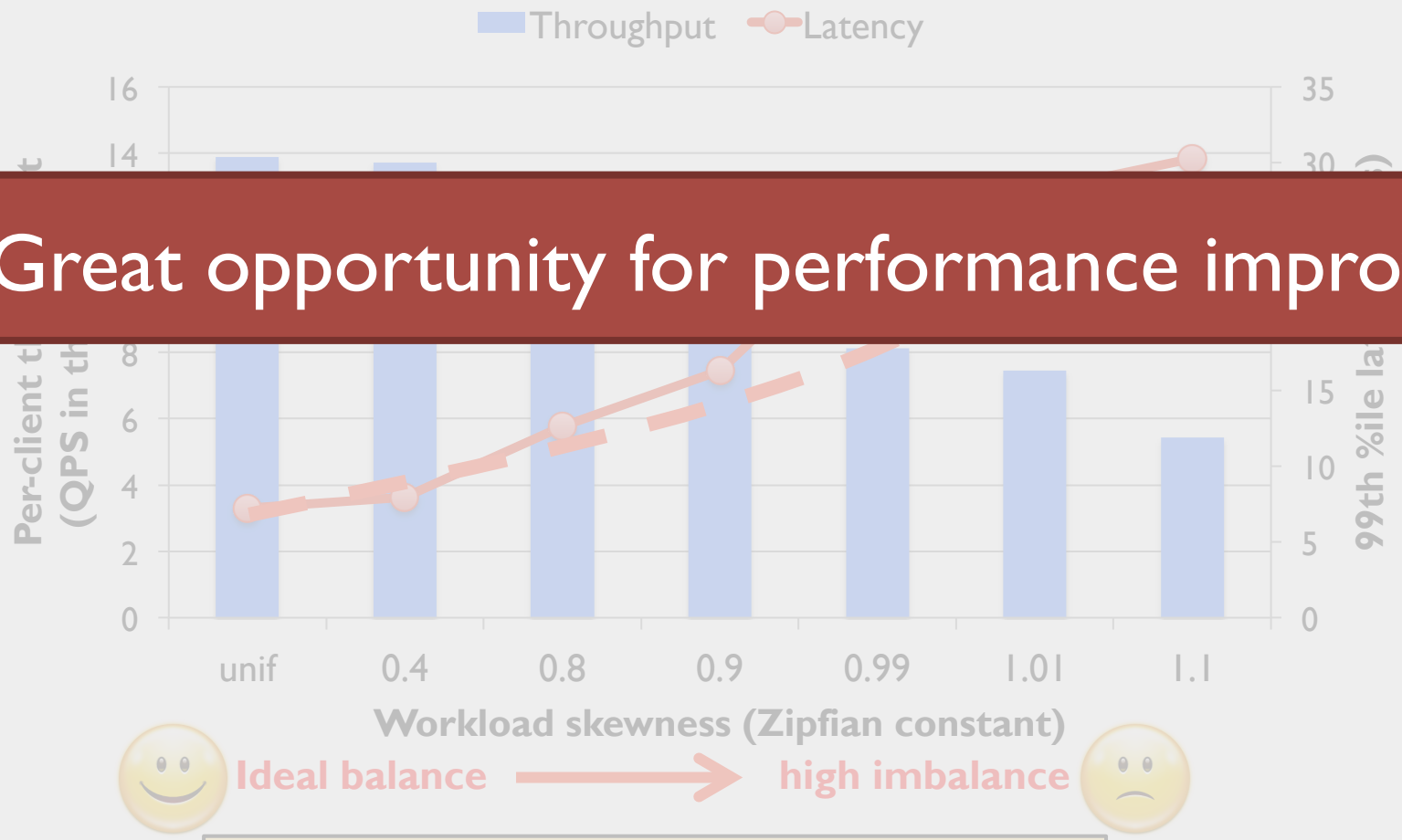
95% GET, 5% SET, Zipfian, 20 cache servers

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 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

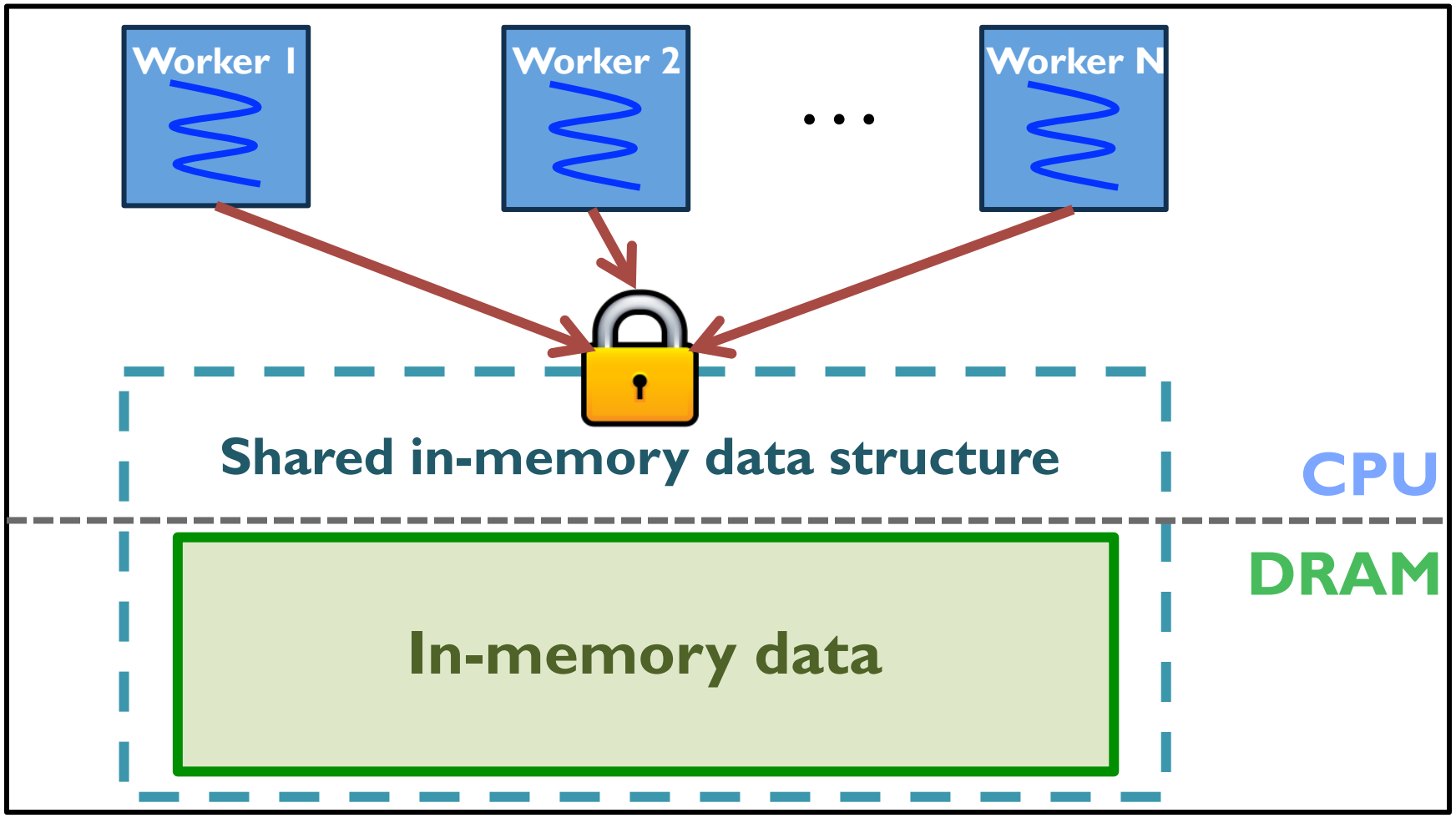
MBal Cache Design

MBal load balancer design

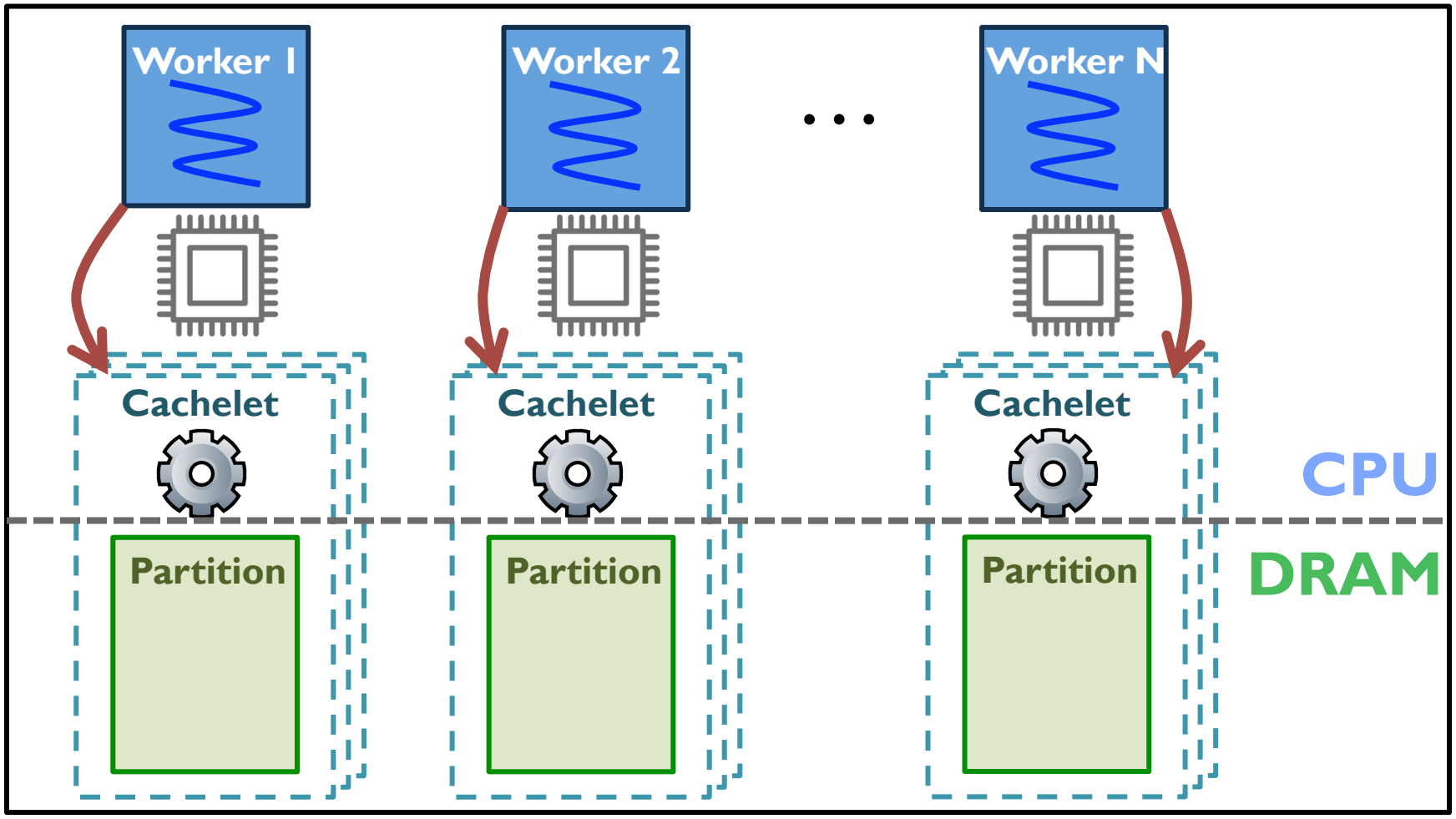
Evaluation

Related work

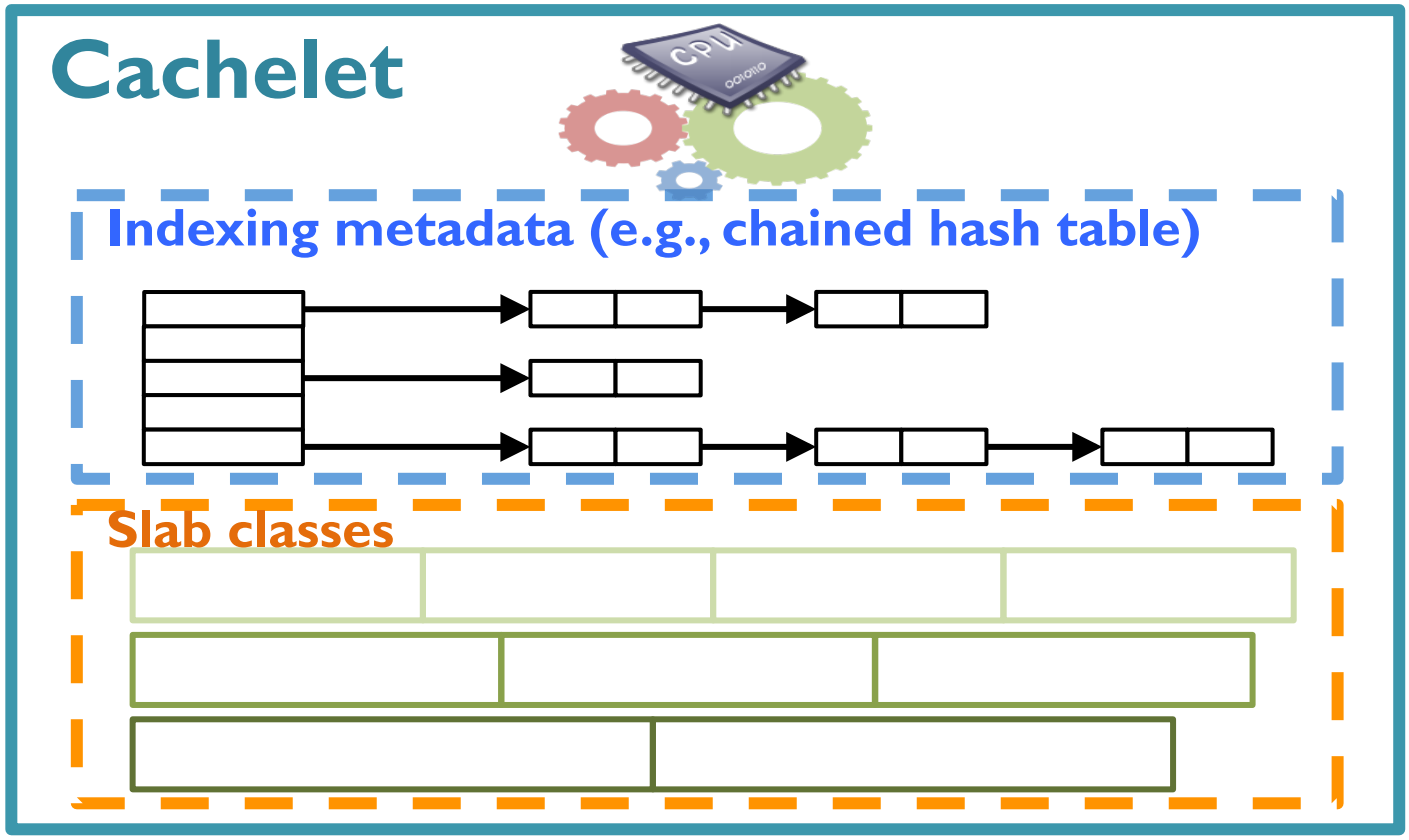
A typical in-memory cache design



MBal: Fine-grained resource partitioning

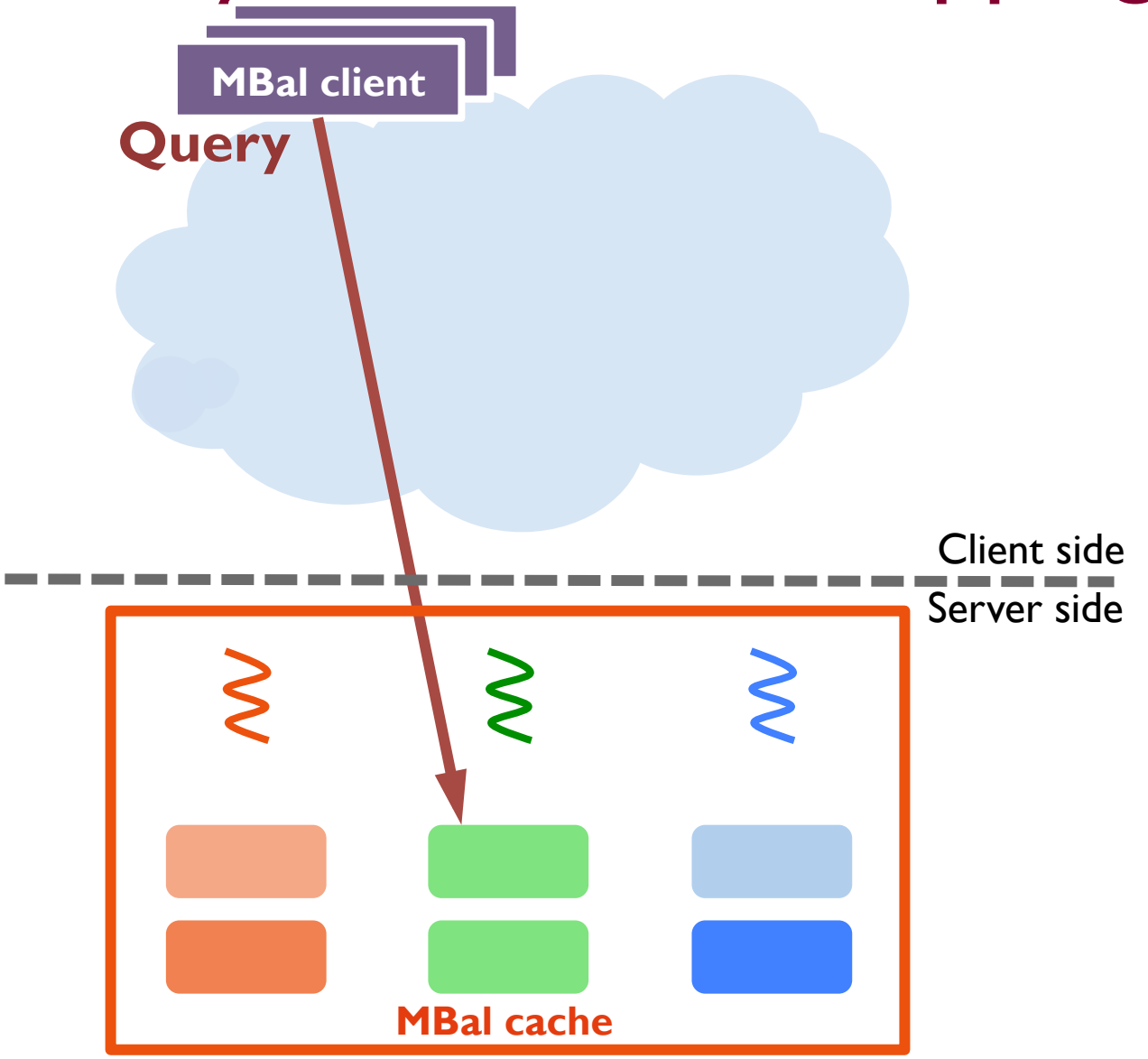


MBal cachelet: Resource encapsulation

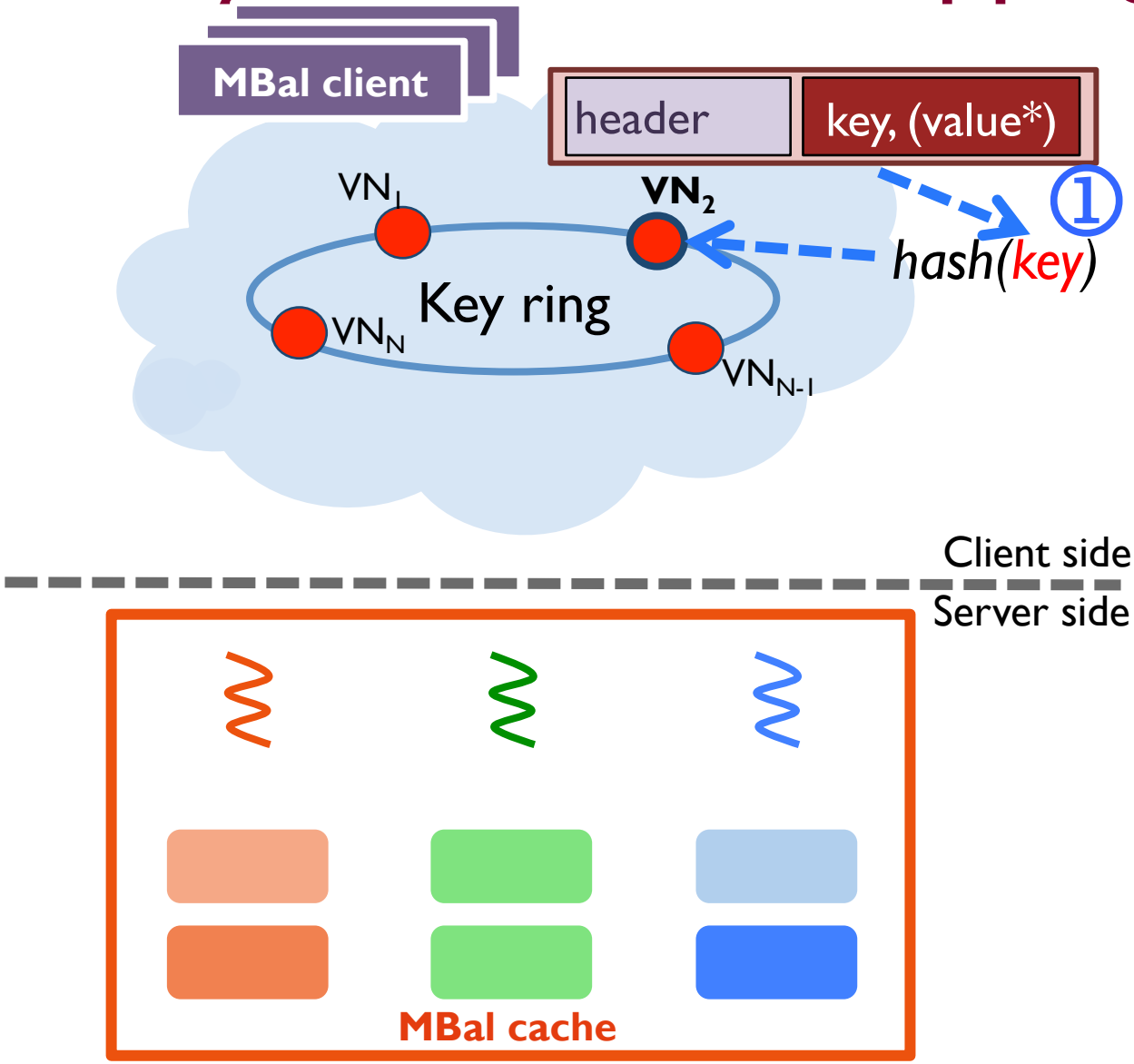


- Cachelet
  - Encapsulates resources
  - Avoids lock contention

Key-to-cachelet mapping

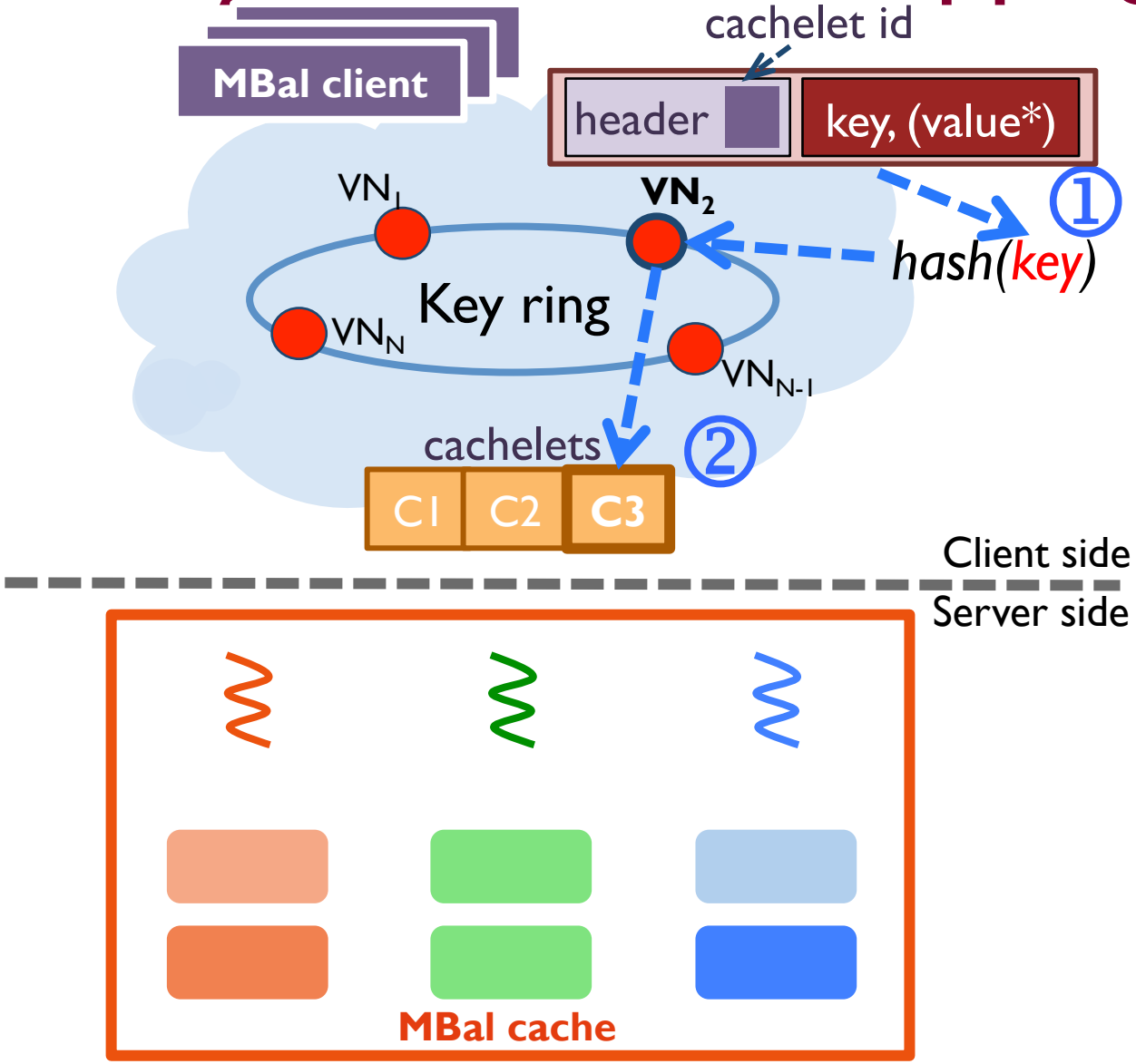


Key-to-cachelet mapping



①Compute VN # with hash

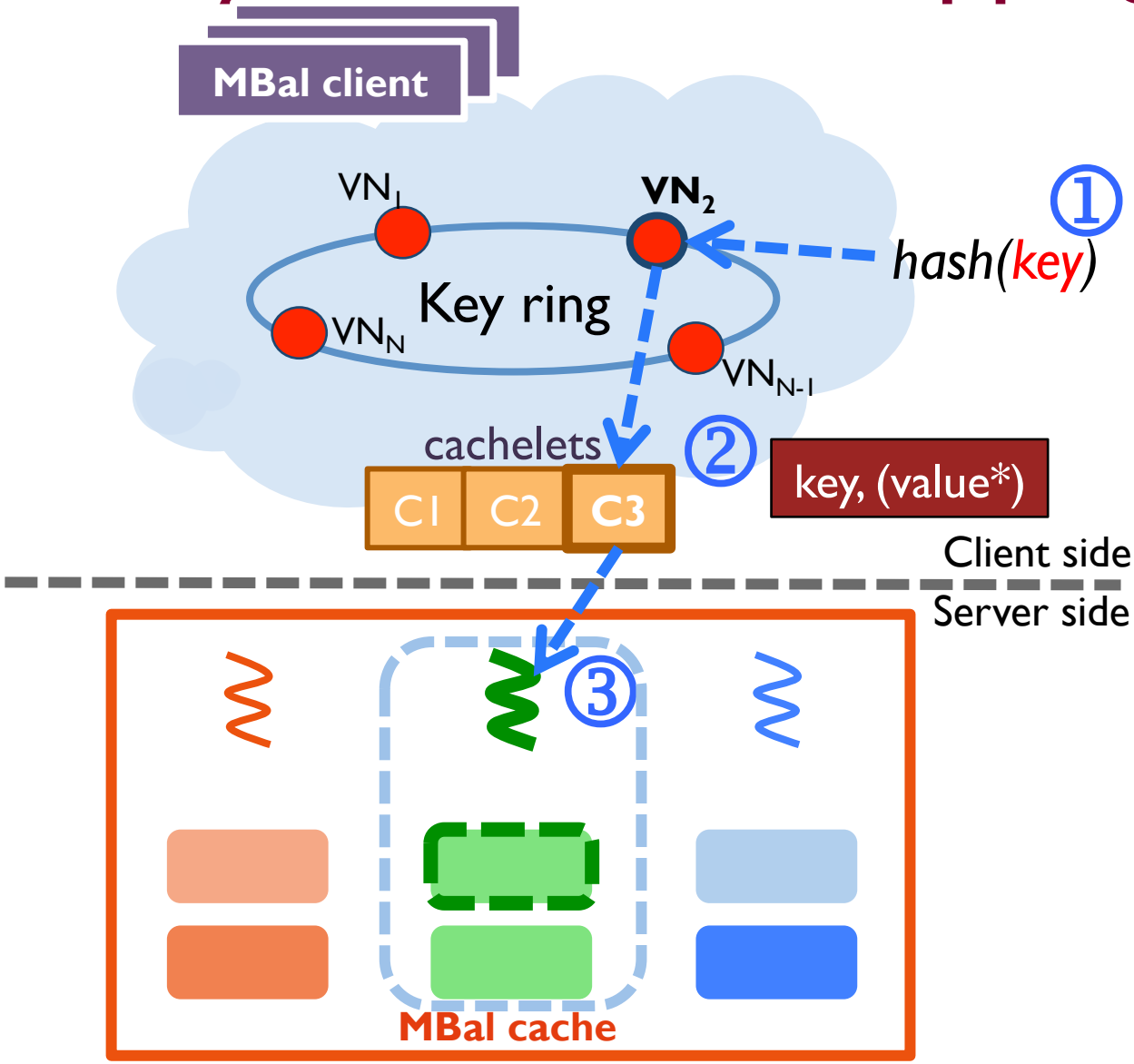
Key-to-cachelet mapping



① Compute VN # with hash

②Map VN # to Cachelet ID

Key-to-cachelet mapping



① Compute VN # with hash

② Map VN # to Cachelet ID

③Map Cachelet ID to the worker thread

Outline

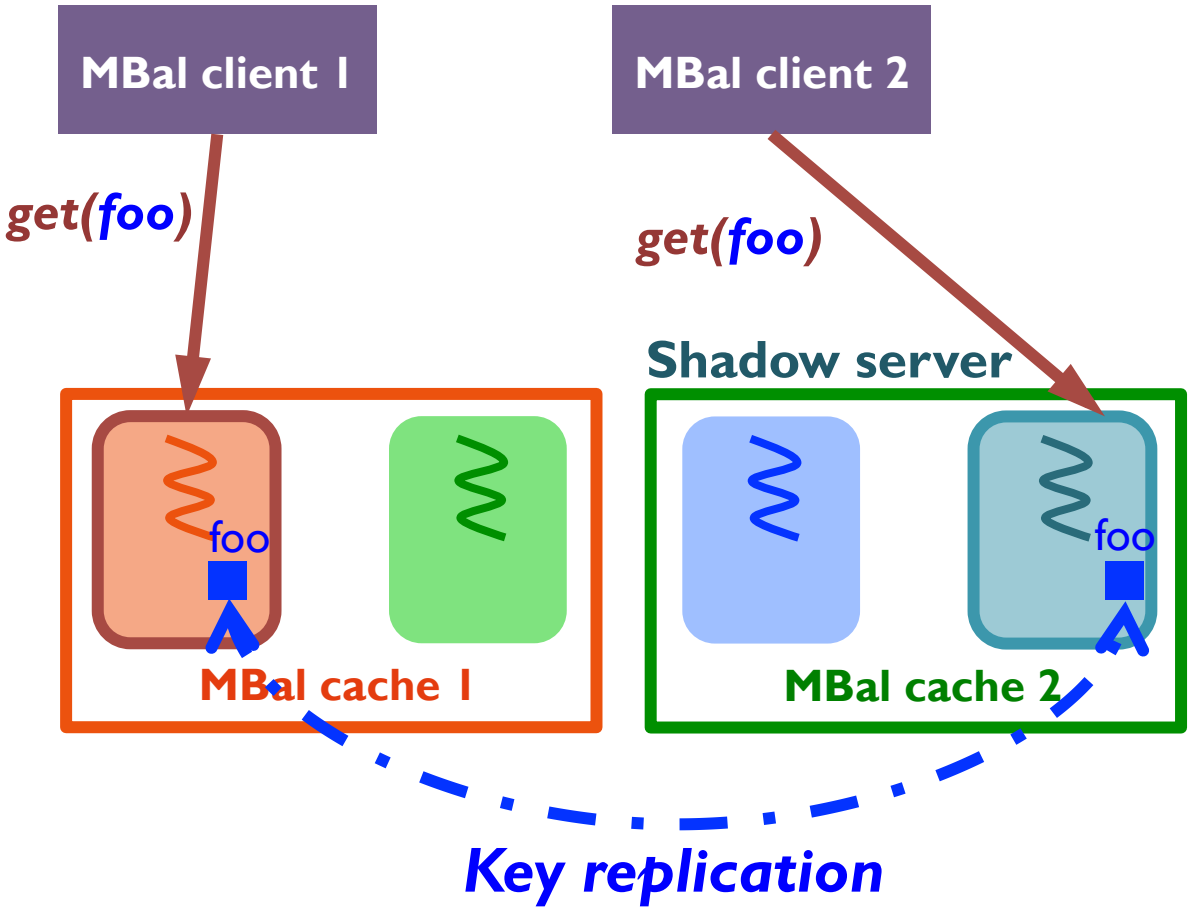
MBal cache design

MBal Multi-Phase Load Balancer

Evaluation

Related work

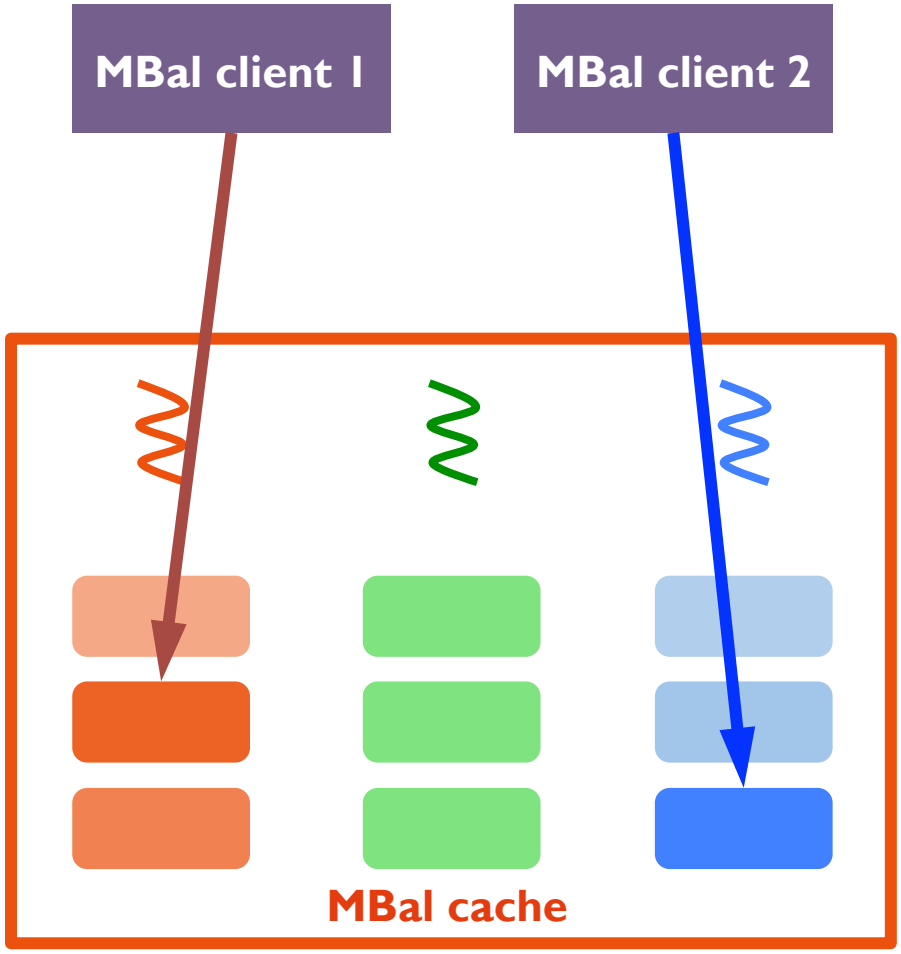
Phase 1: Key replication



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

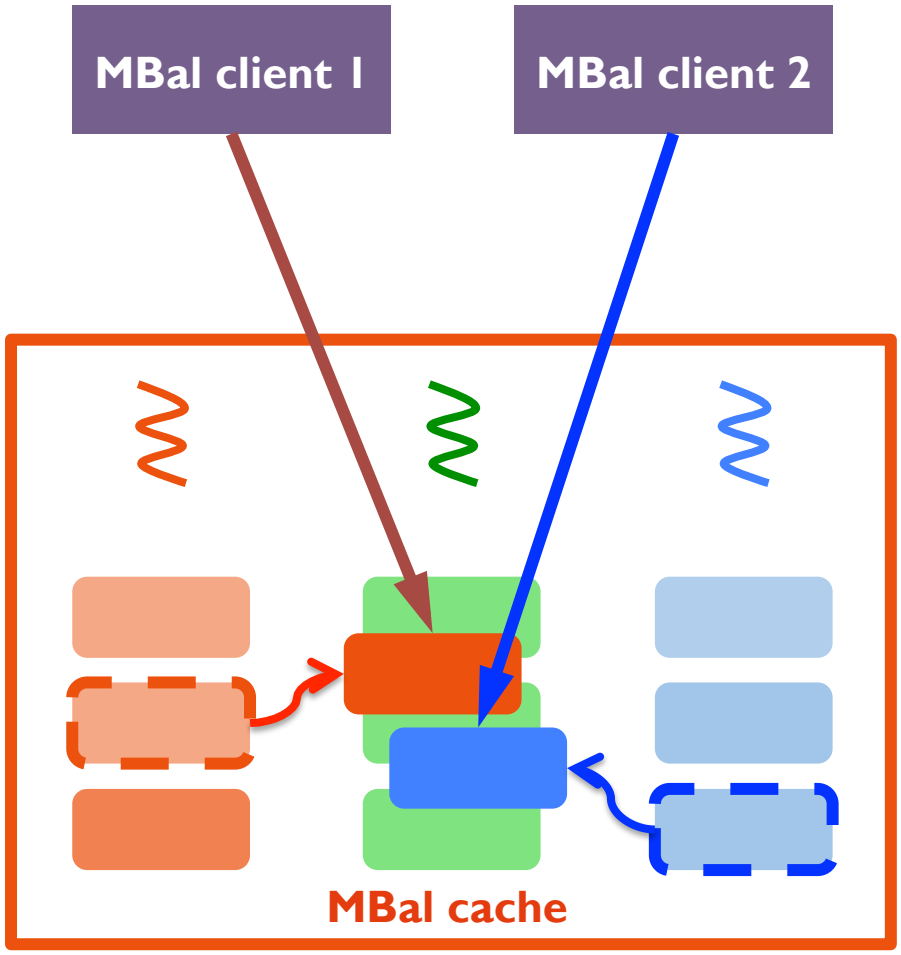
* SPORE [SoCC'13]

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

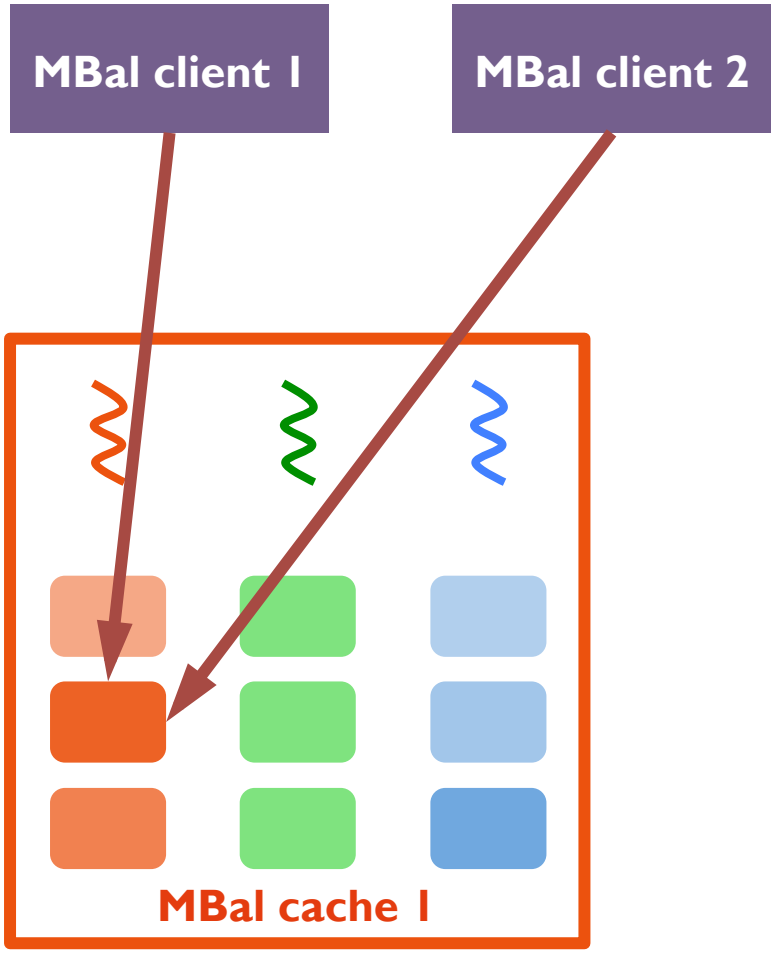
Phase 2: Server-local cachelet migration



Server-local migration

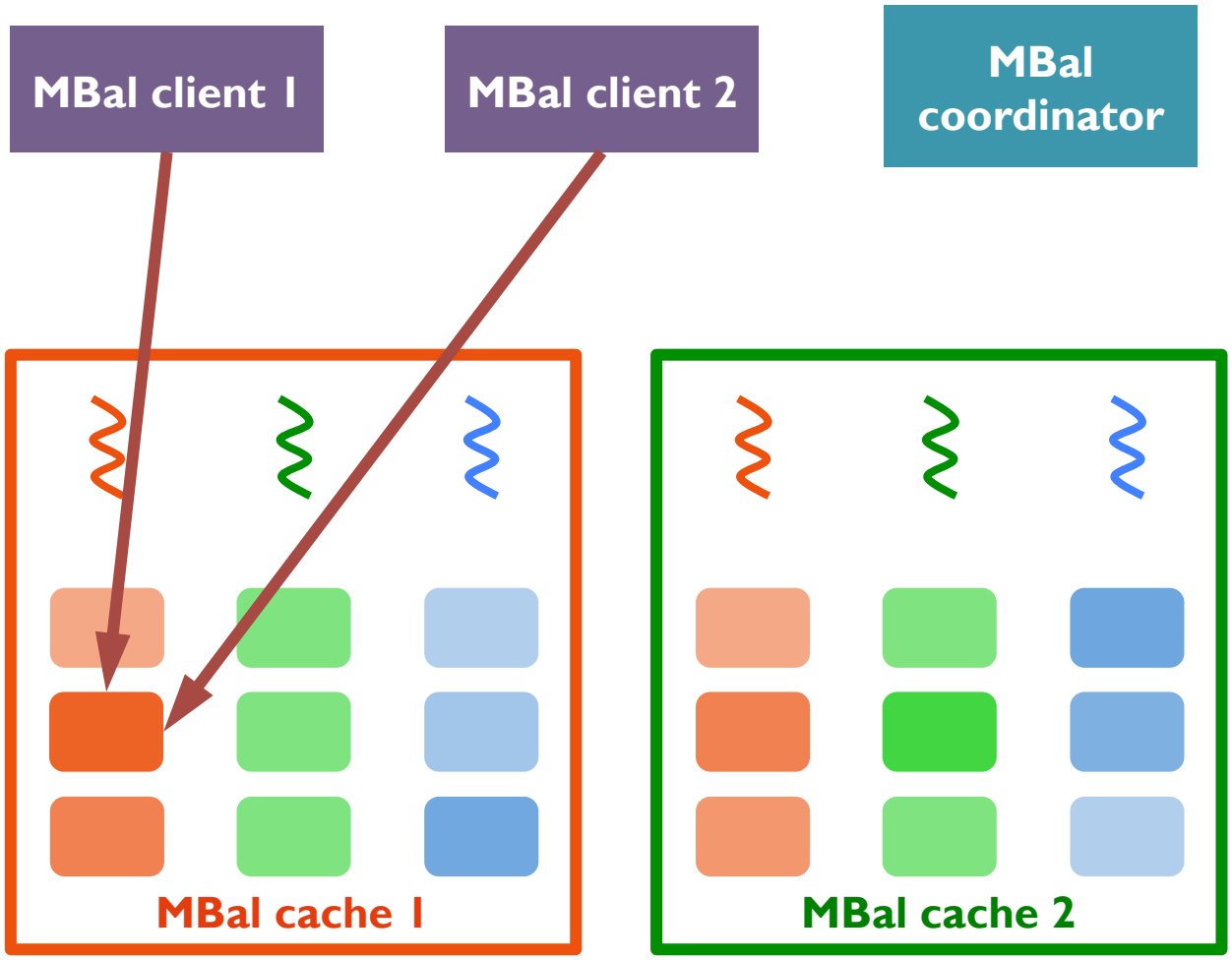
- TRIGGER?
  - # hot keys > $REPL_{HIGH}$
  - Enough local headroom
- ACTION?
  - Migrate/swap cachelet(s) within a server
  - ILP
- FEATURES?
  - Coarse-grained
  - Temporary

Phase 3: Coordinated cachelet migration



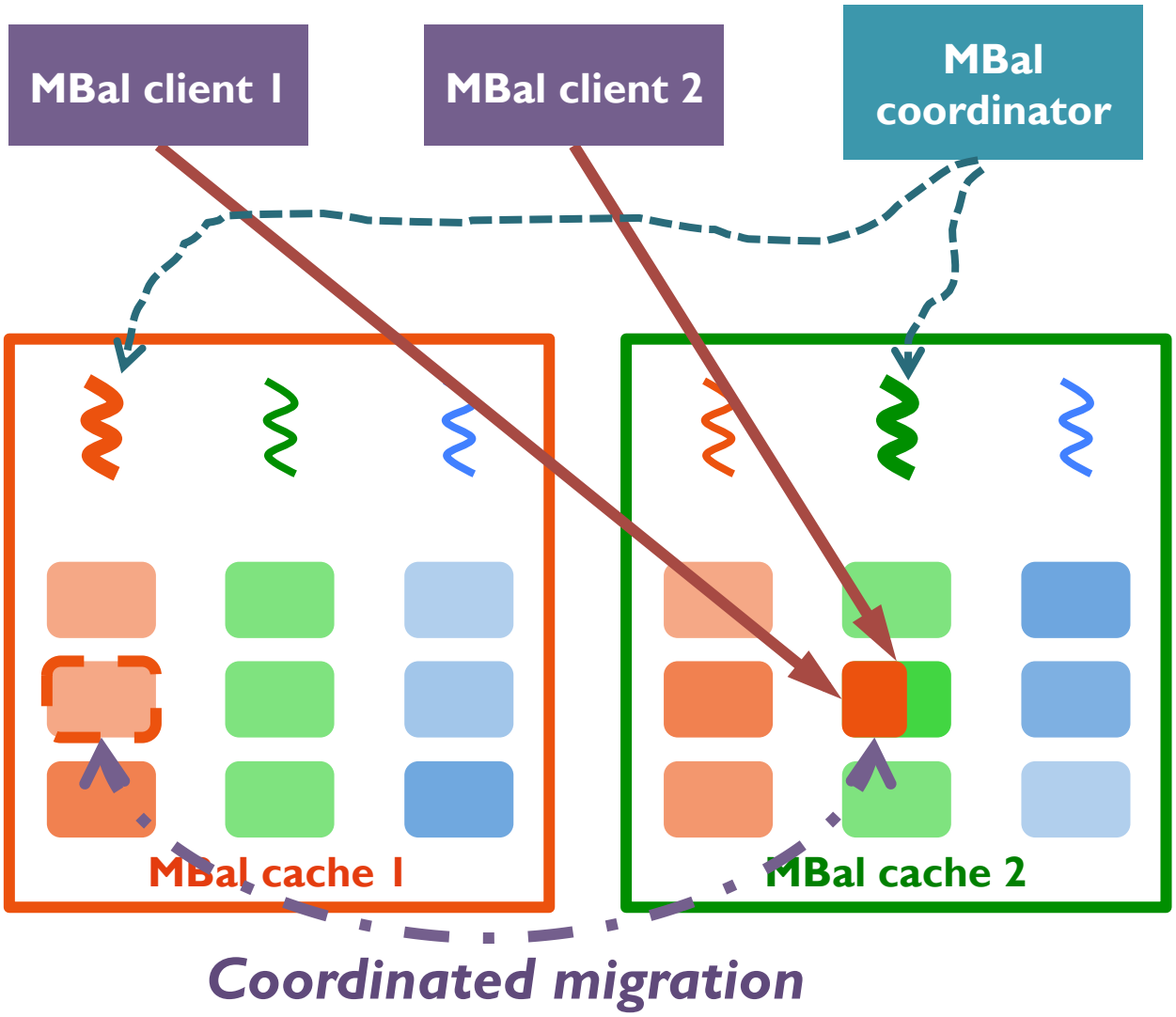
- TRIGGER?
  - # hot keys > $REPL_{HIGH}$
  - Not enough local headroom
- ACTION?
  - Migrate/swap cachelet(s) across servers
  - ILP
- FEATURES?
  - Coarse-grained
  - Permanent

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

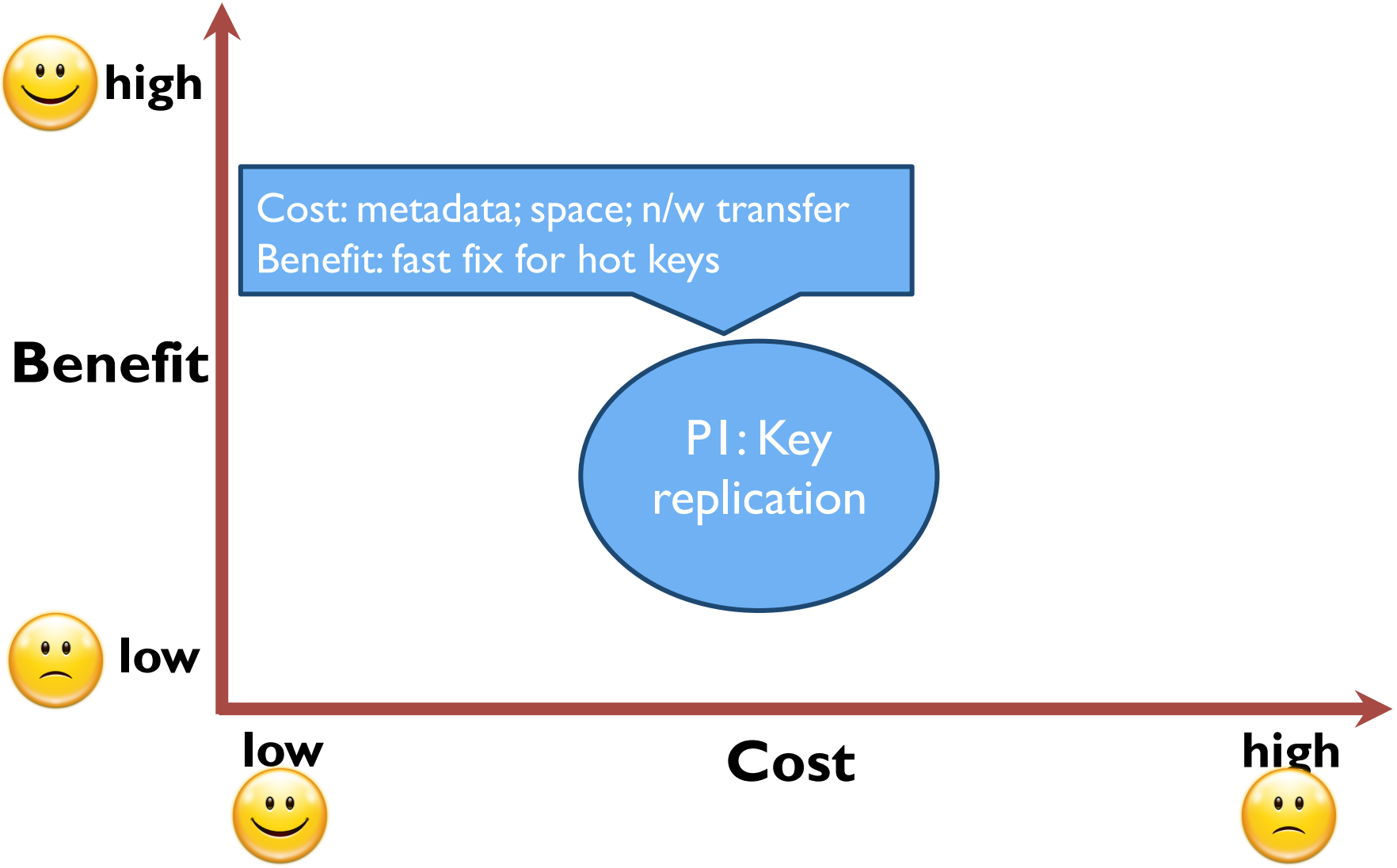


- TRIGGER?
  - # hot keys > $REPL_{HIGH}$
  - Not enough local headroom
- ACTION?
  - Migrate/swap cachelet(s) across servers
  - ILP
- FEATURES?
  - Coarse-grained
  - Permanent

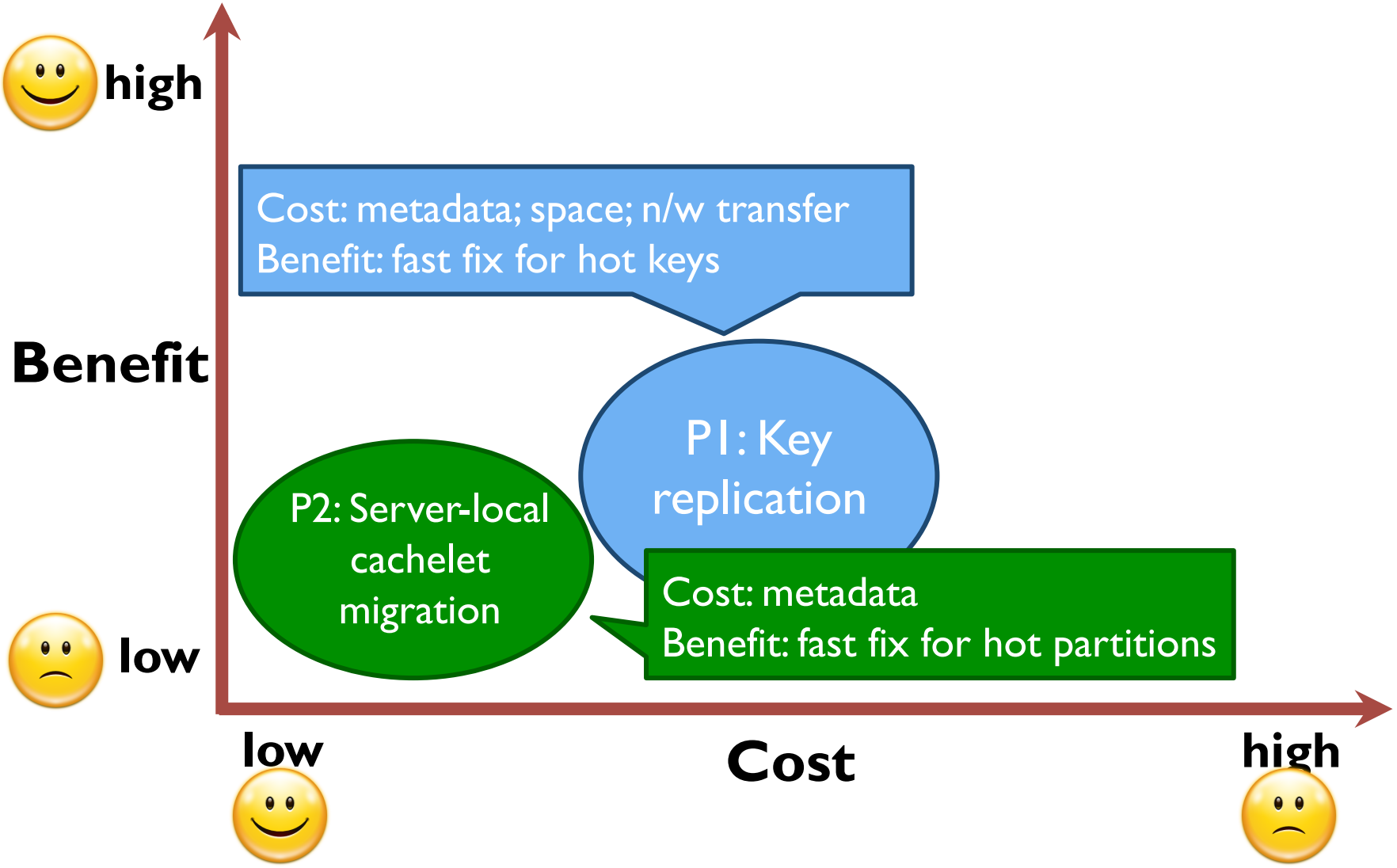
Client-side mapping change

- Phase 2: Server-local cachelet migration
  - Clients are informed of cachelet migration when cache home worker receives requests about that migrated cachelet
- Phase 3: Coordinated cachelet migration
  - Once migration is done, source worker informs coordinator about the mapping change
  - Clients ping coordinator periodically

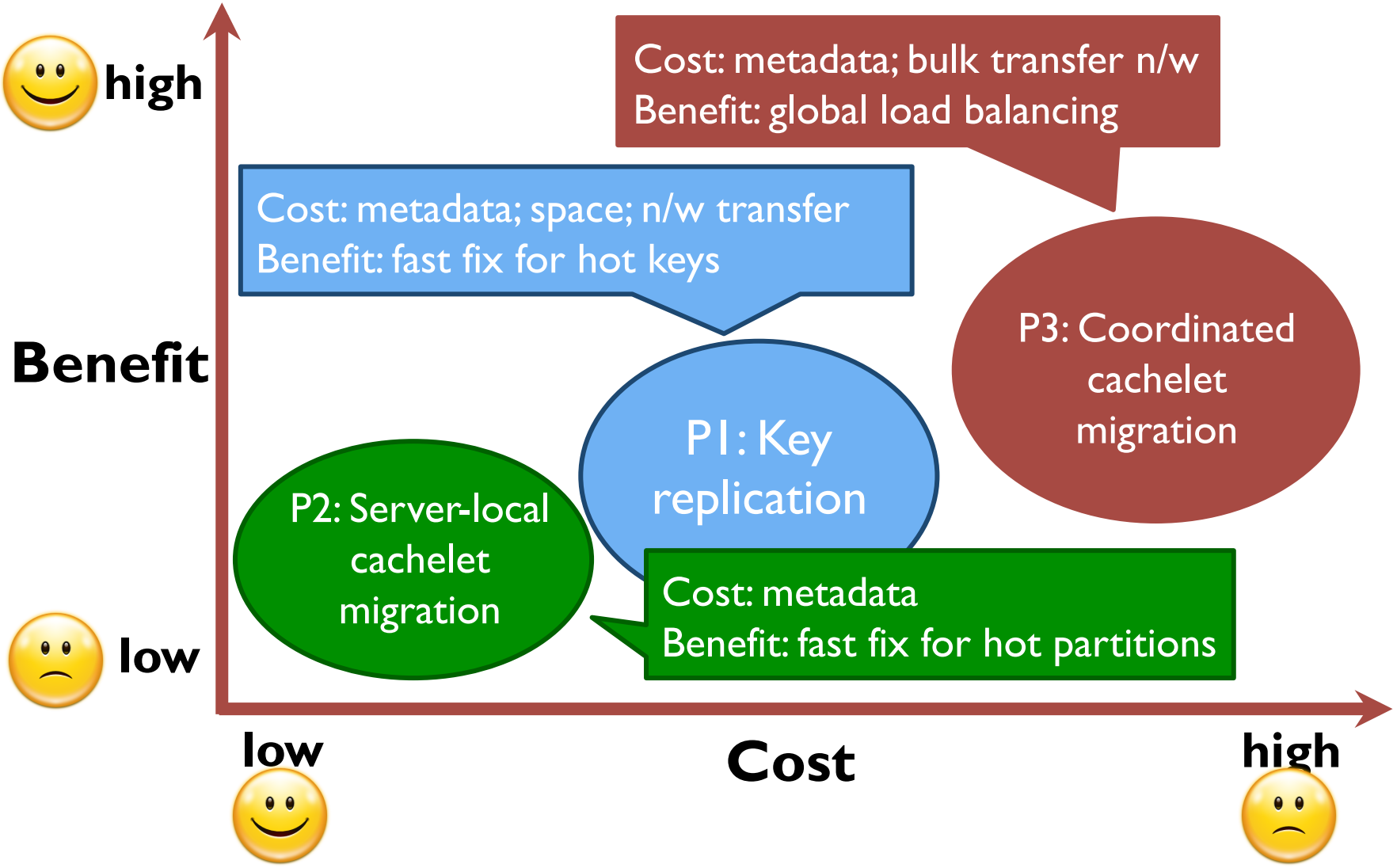
MBal: Cost/benefit trade-offs



MBal: Cost/benefit trade-offs



MBal: Cost/benefit trade-offs



Outline

MBal cache design

MBal load balancer design

Evaluation

Related work

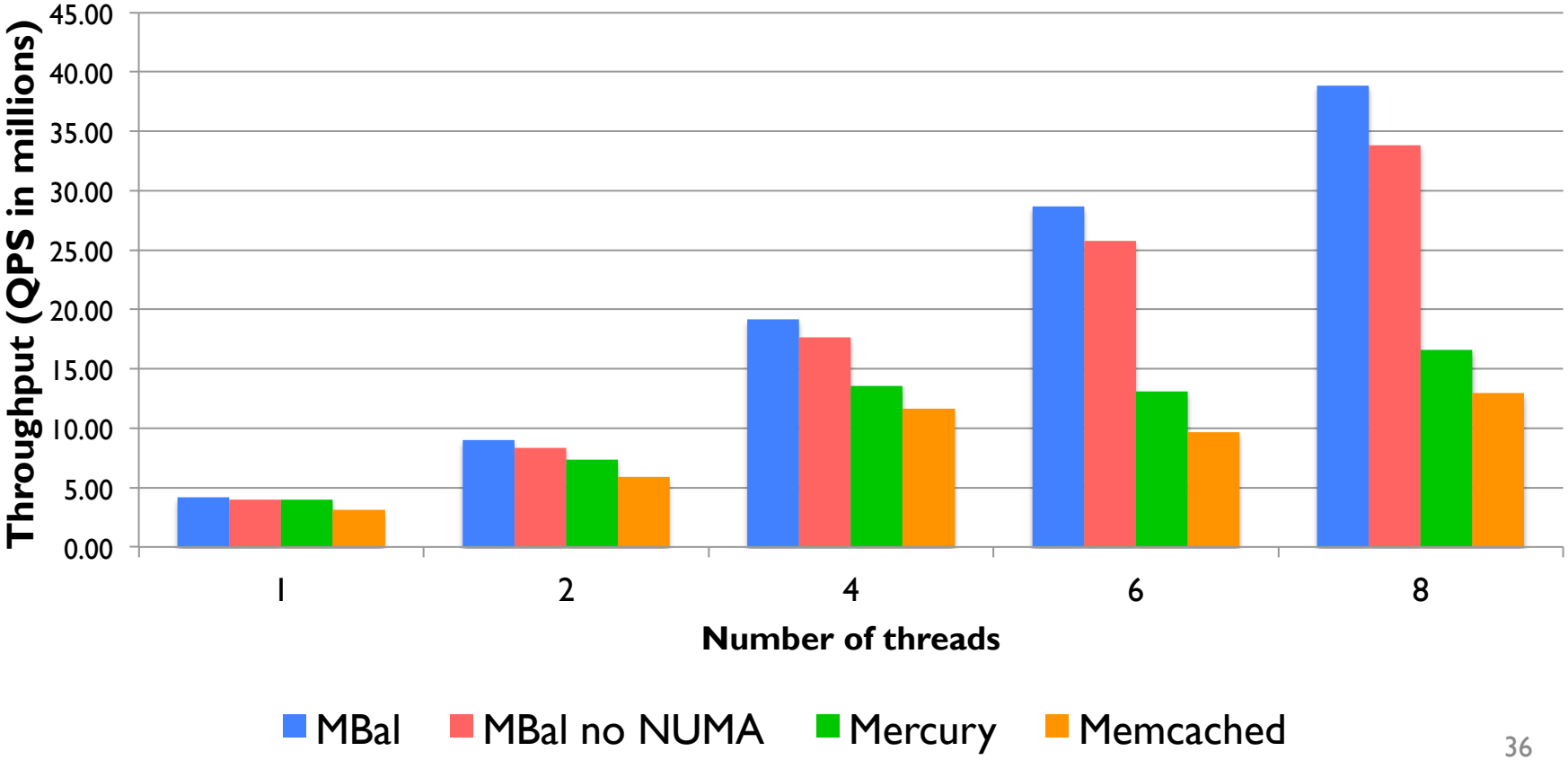
Methodology

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

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

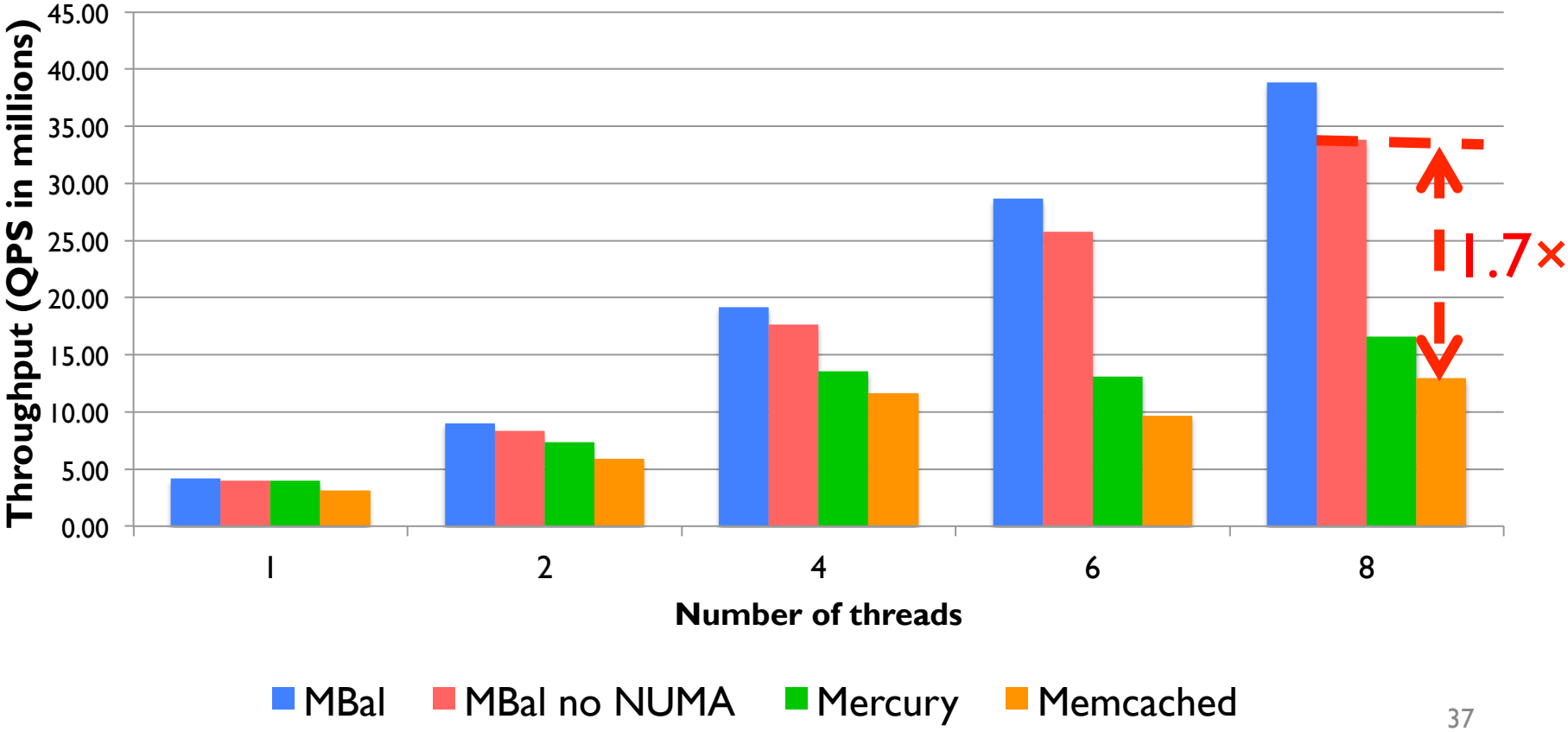
MBal evaluation – micro-benchmark

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



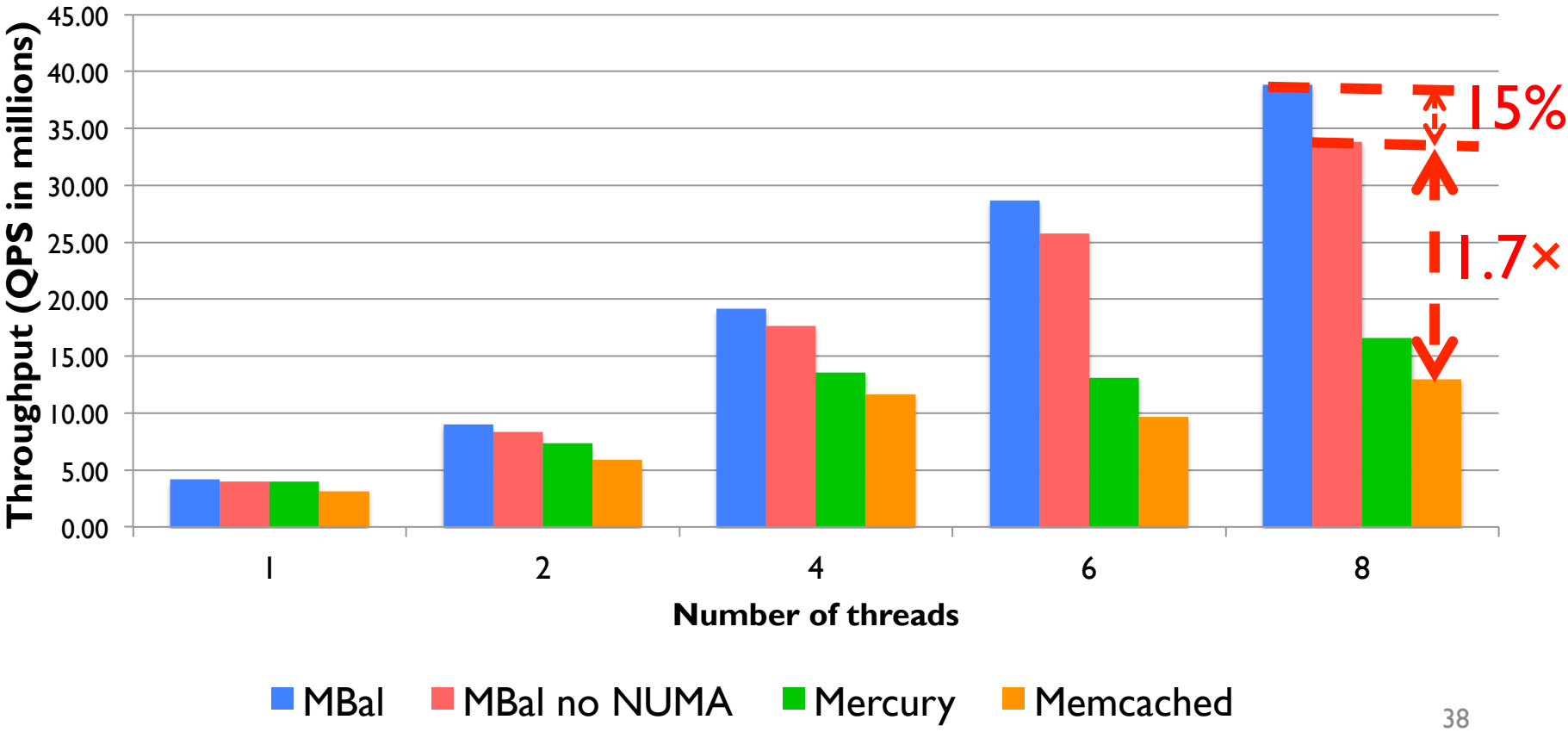
MBal evaluation – micro-benchmark

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



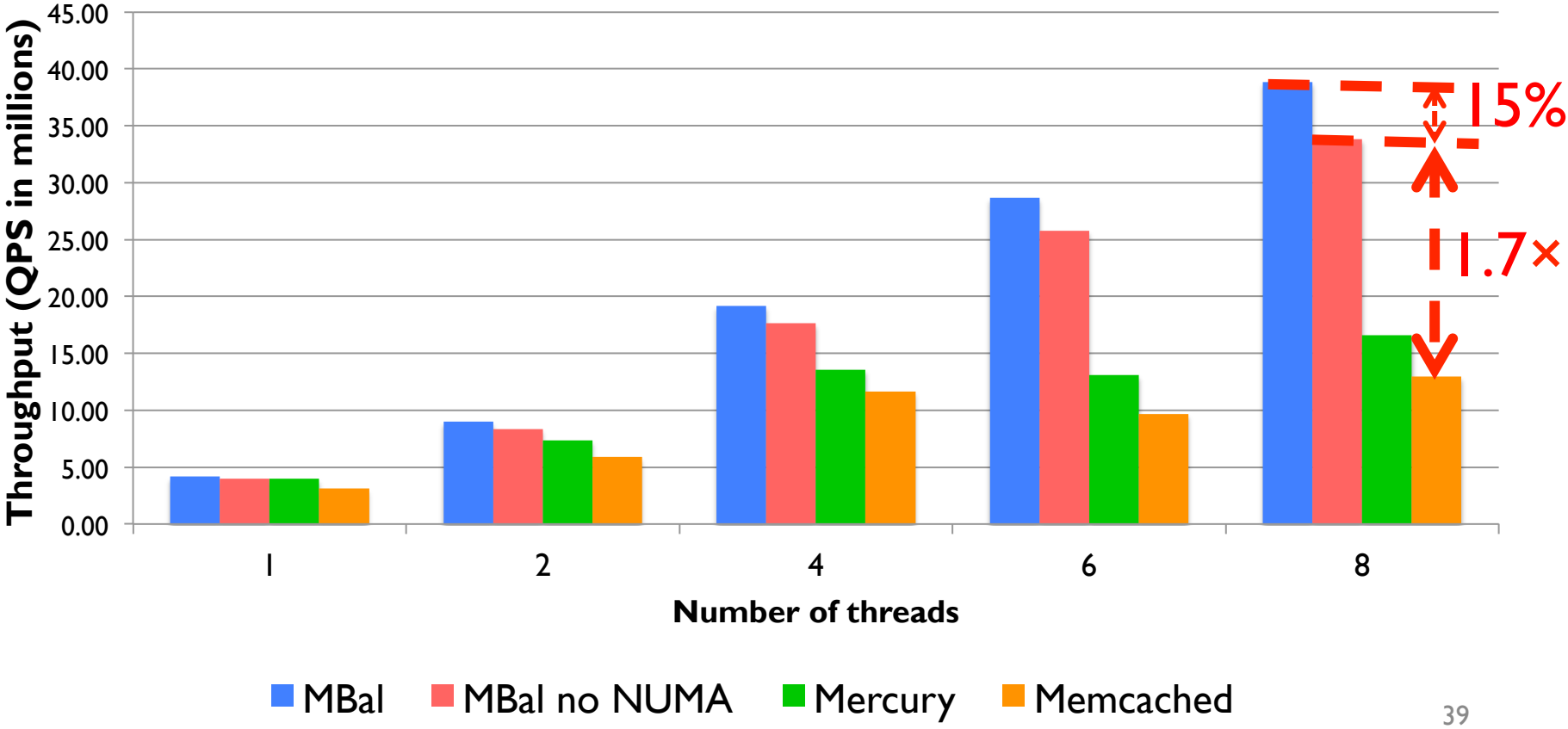
MBal evaluation – micro-benchmark

- 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 uses fine-grained cachelet design
- ✓ MBal eliminates bucket-level lock contention

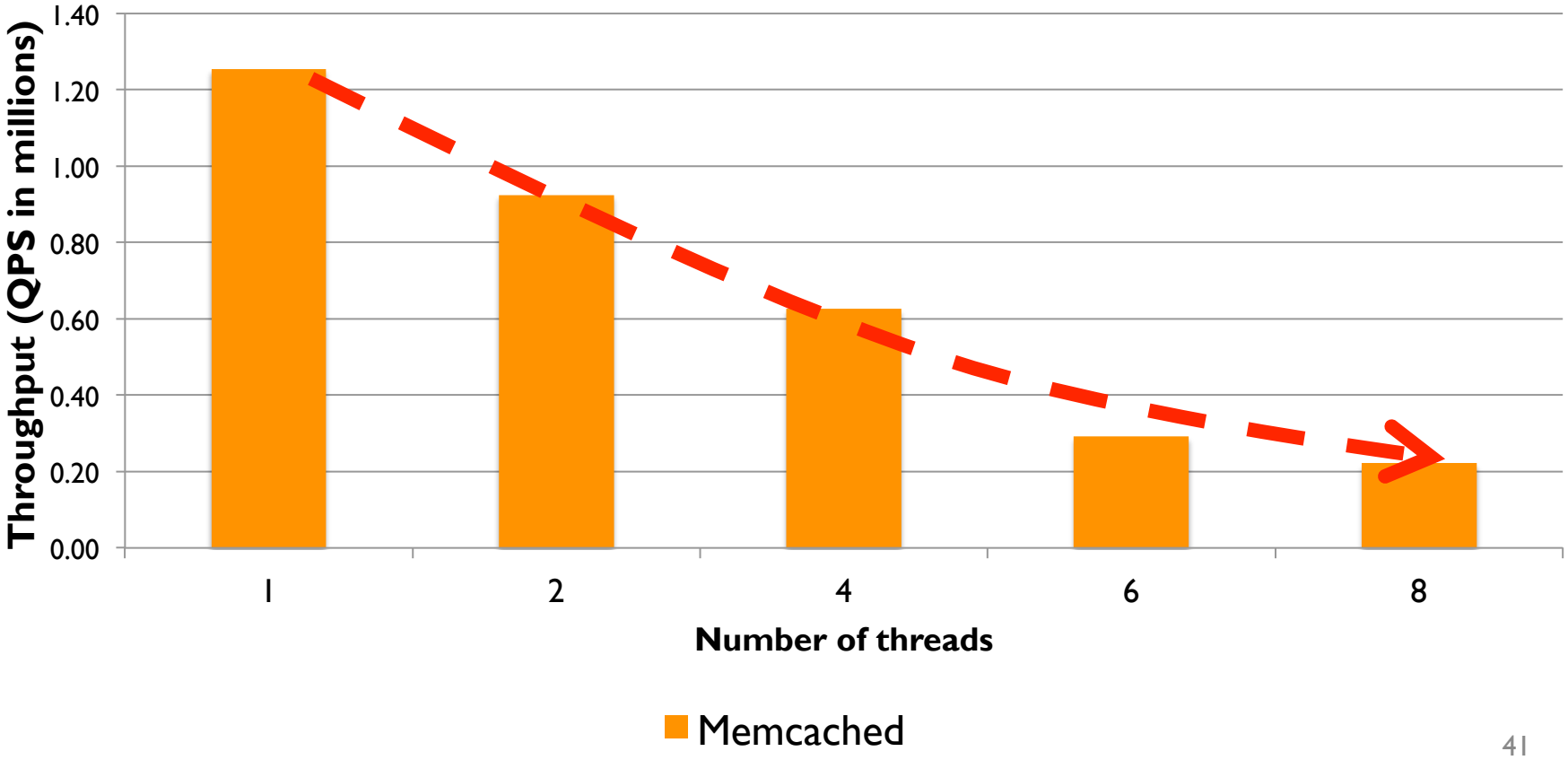


MBal evaluation – micro-benchmark

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

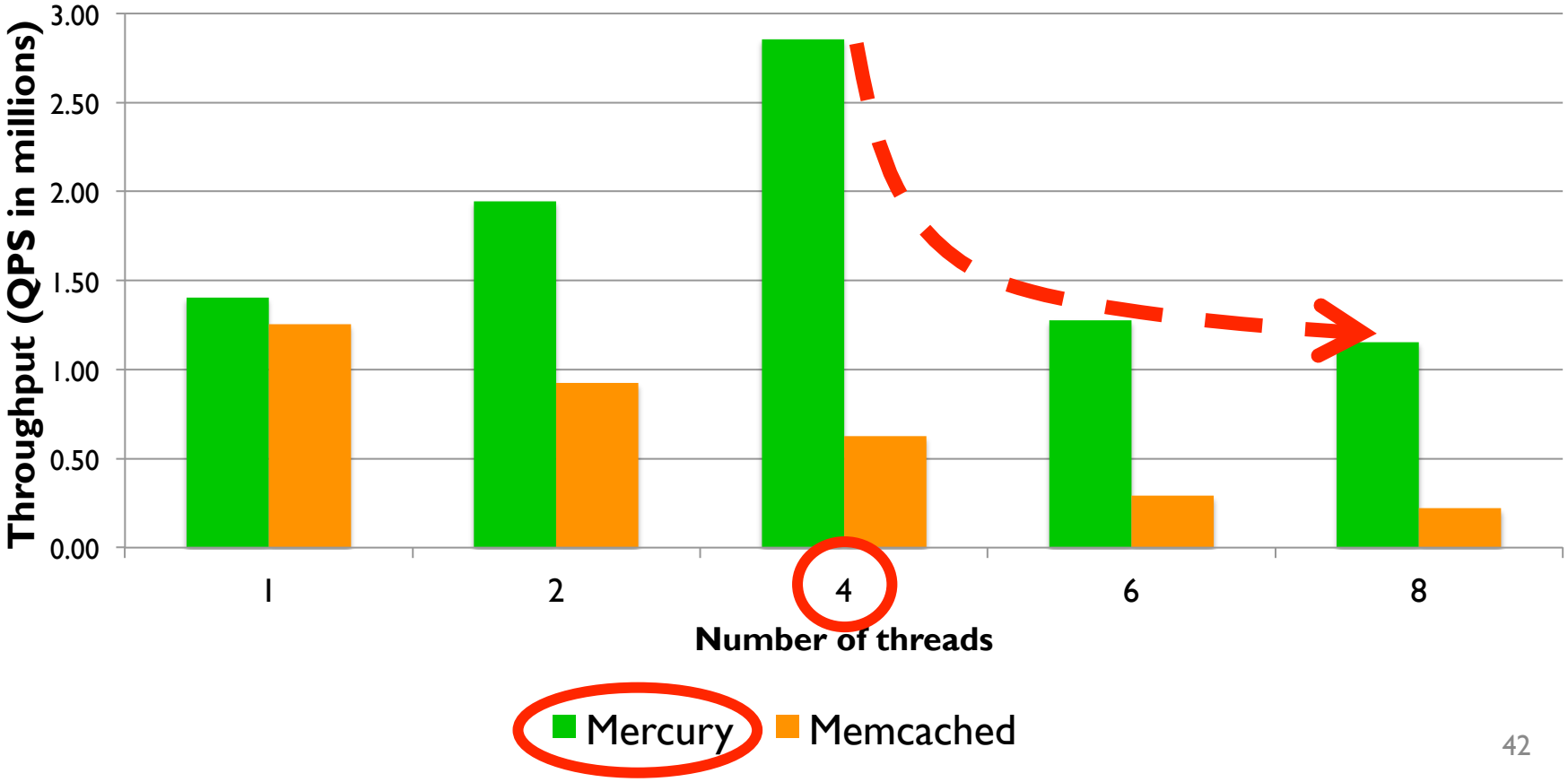
MBal evaluation – micro-benchmark

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



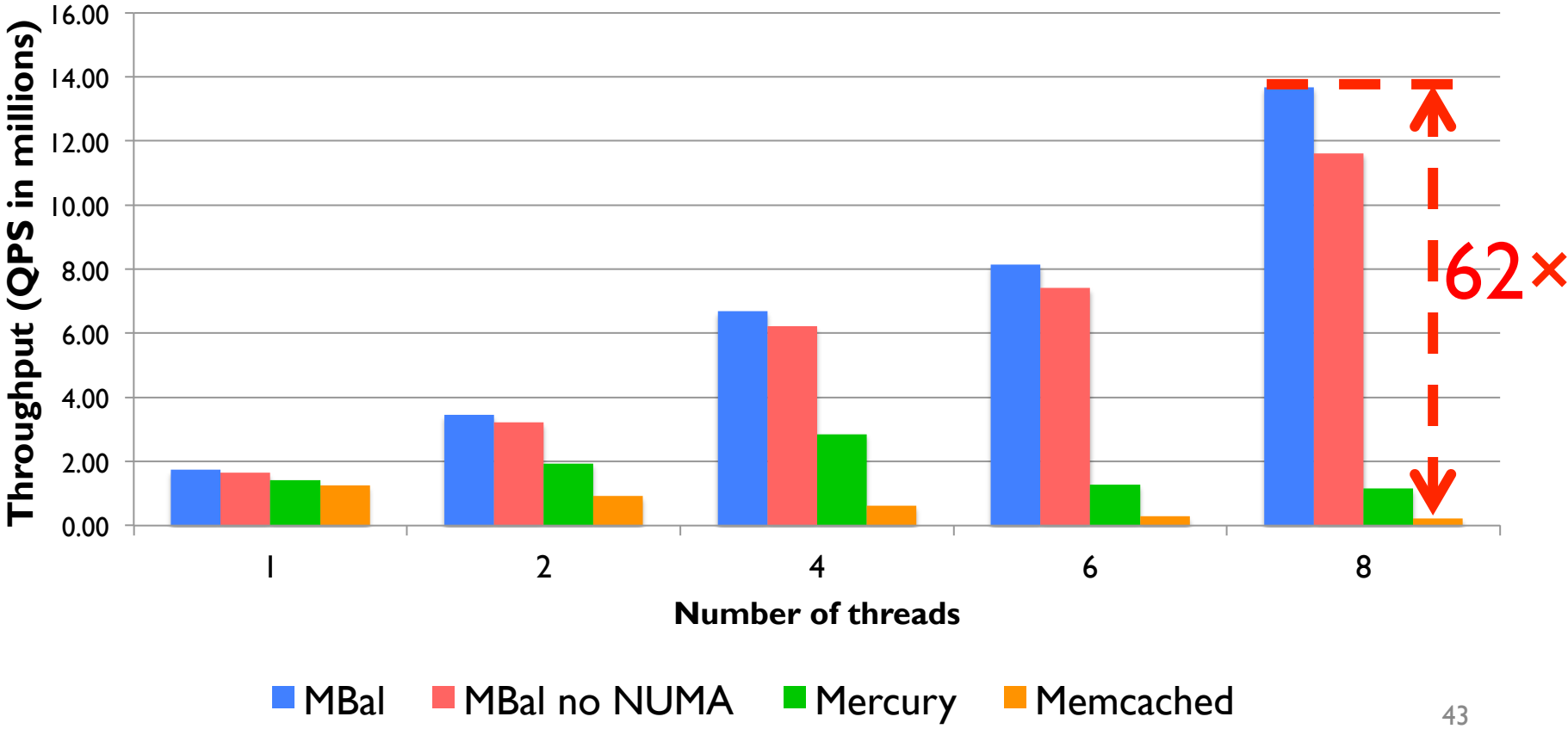
MBal evaluation – micro-benchmark

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



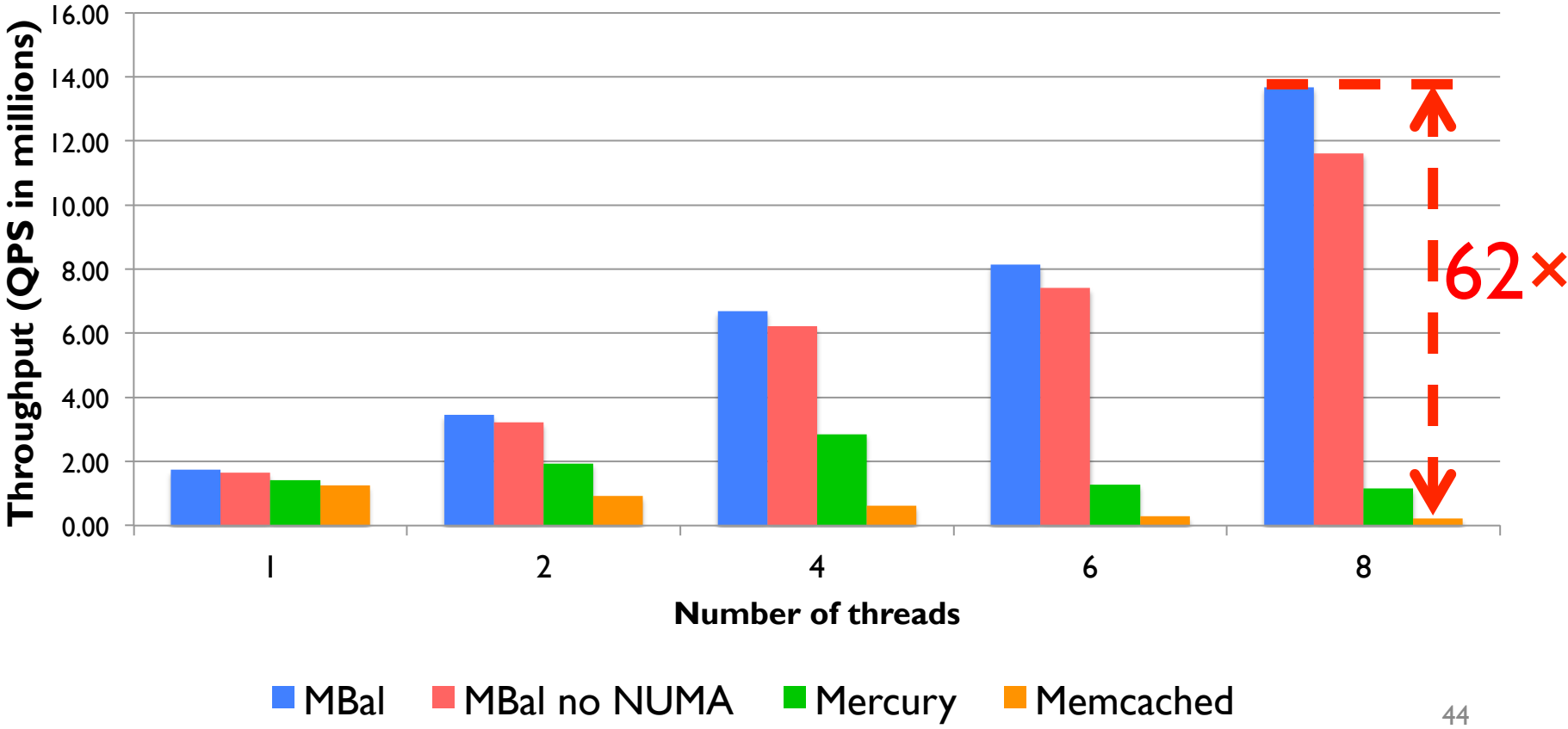
MBal evaluation – micro-benchmark

- 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!



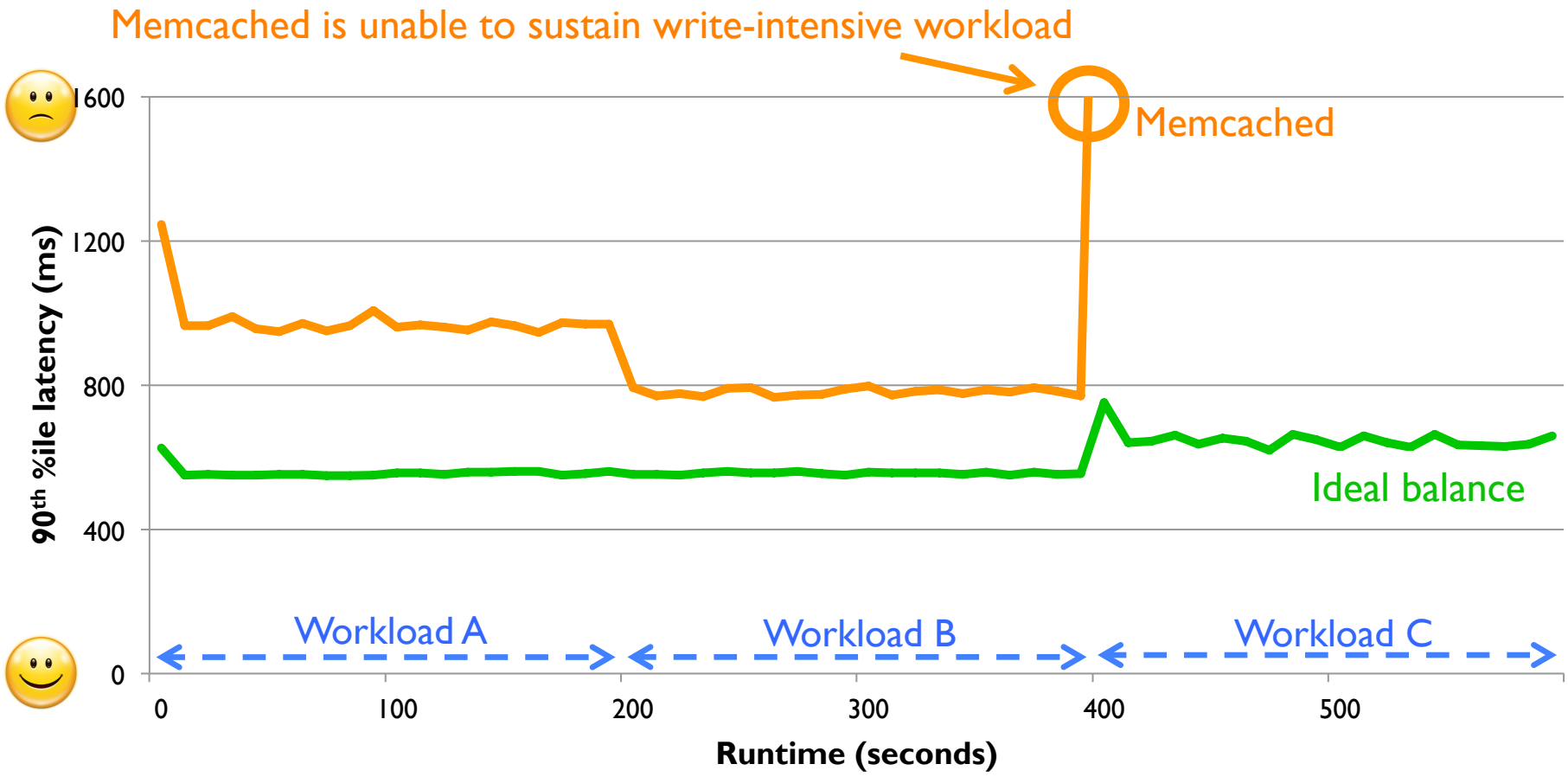
End-to-end load balancer evaluation

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), MBal caches on 20-VM cluster (c3.large)

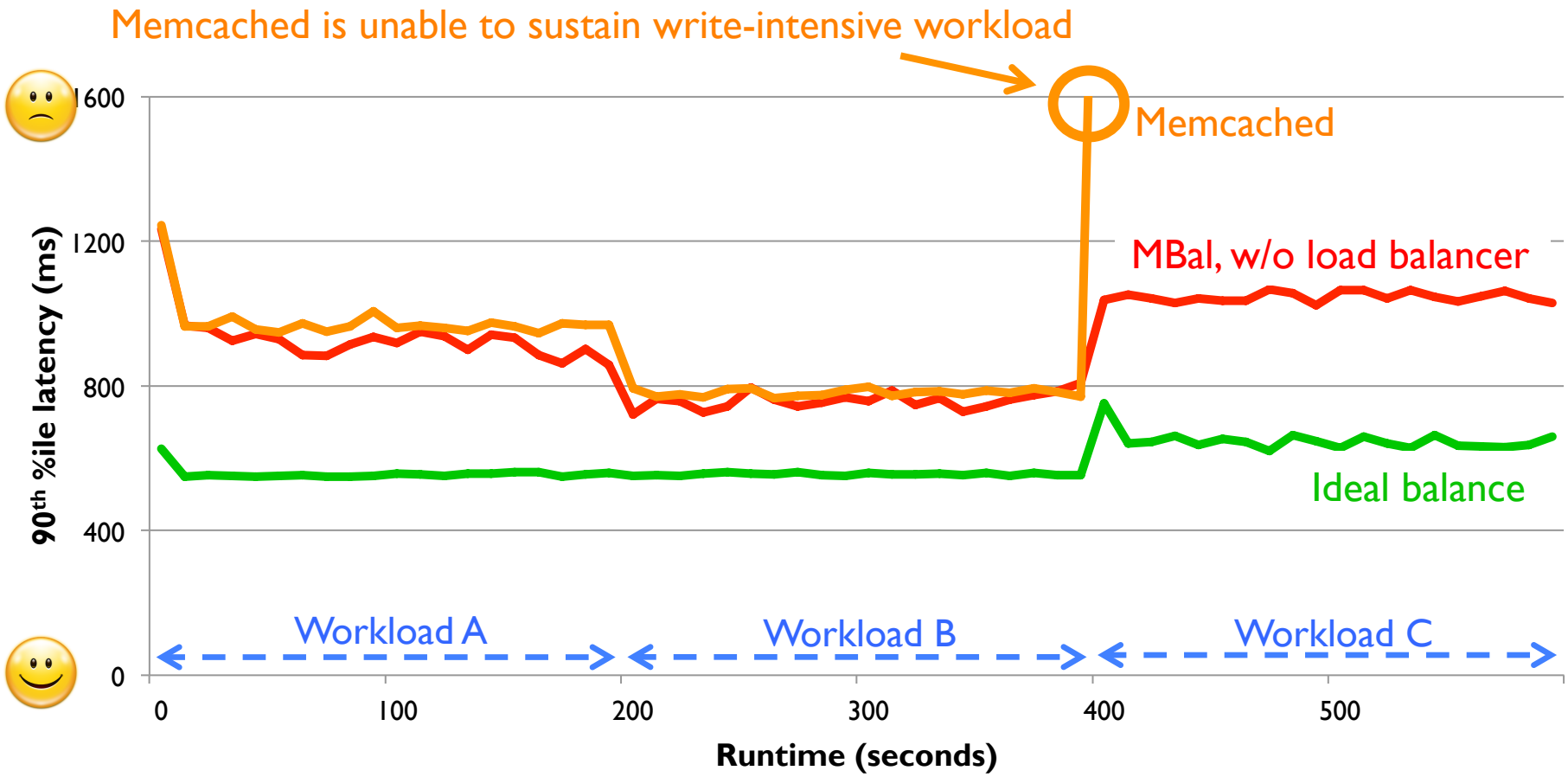
Load balancer evaluation

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



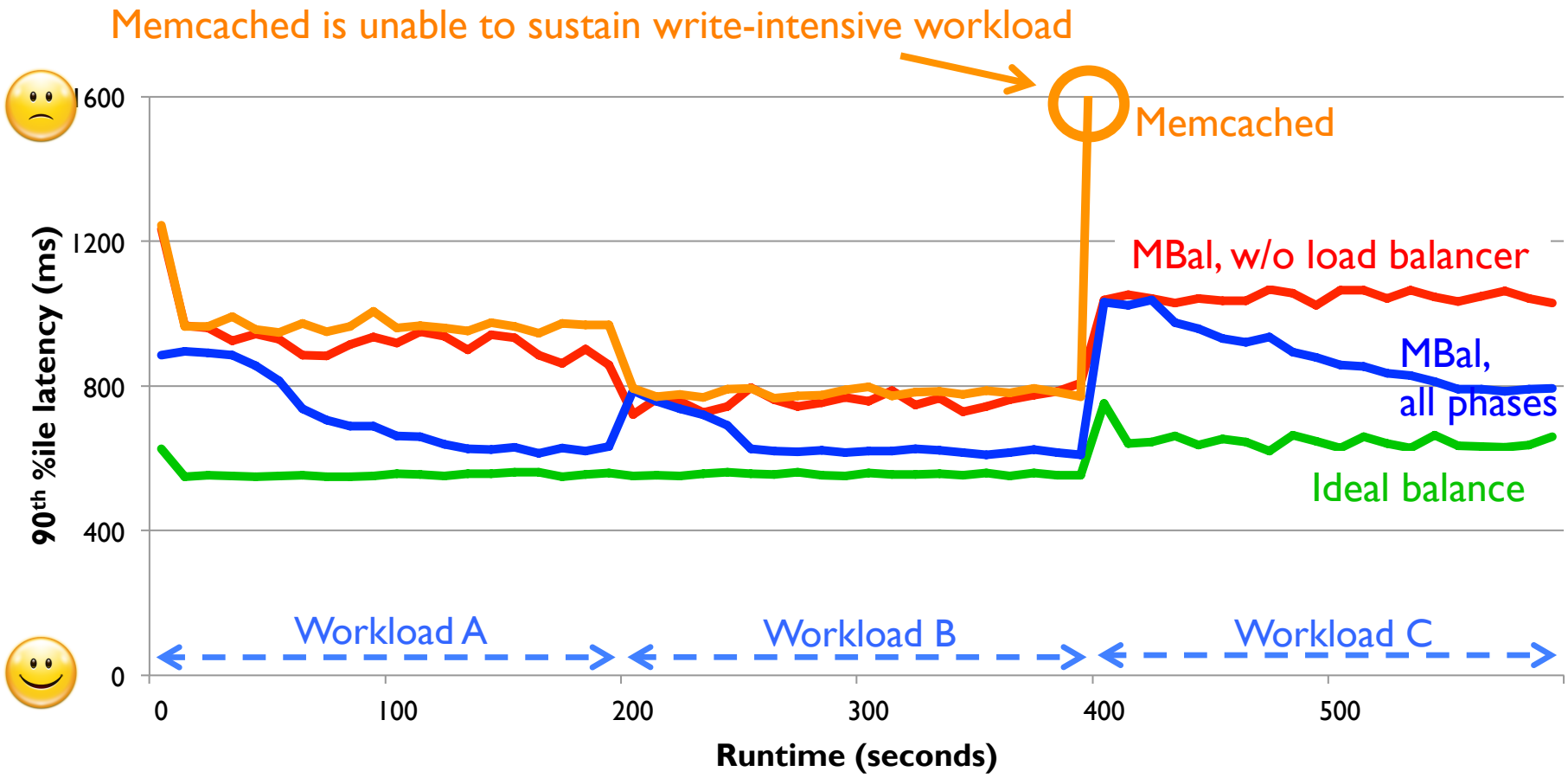
Load balancer evaluation

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



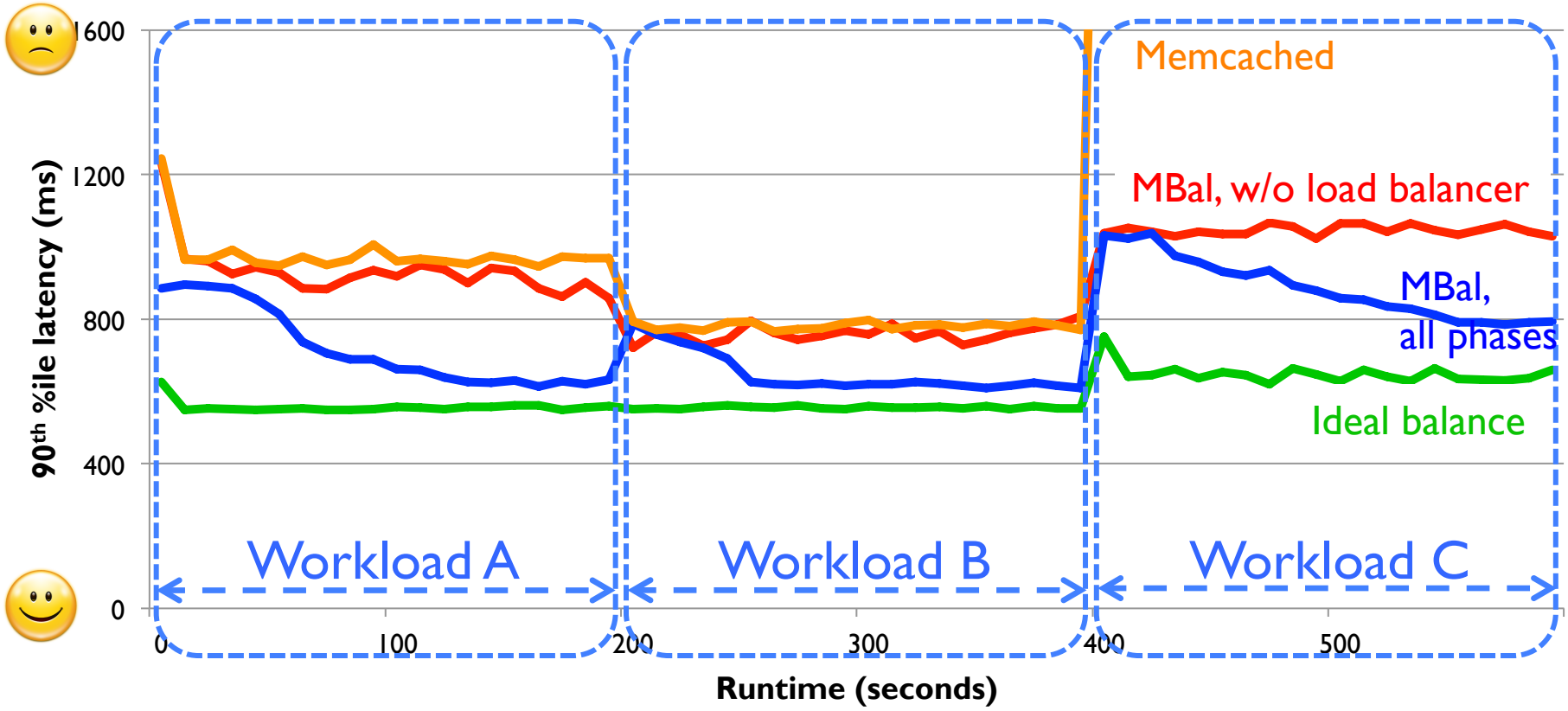
Load balancer evaluation

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



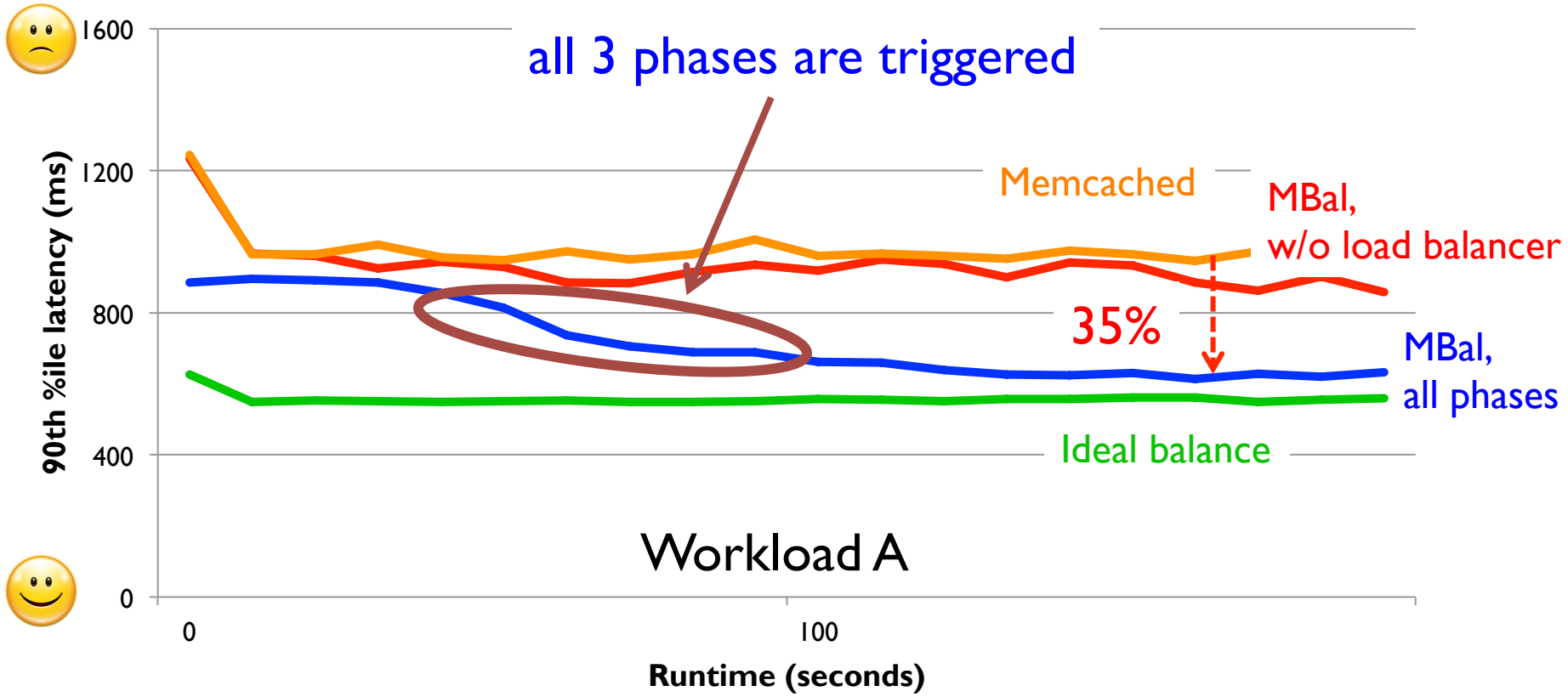
Load balancer evaluation

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



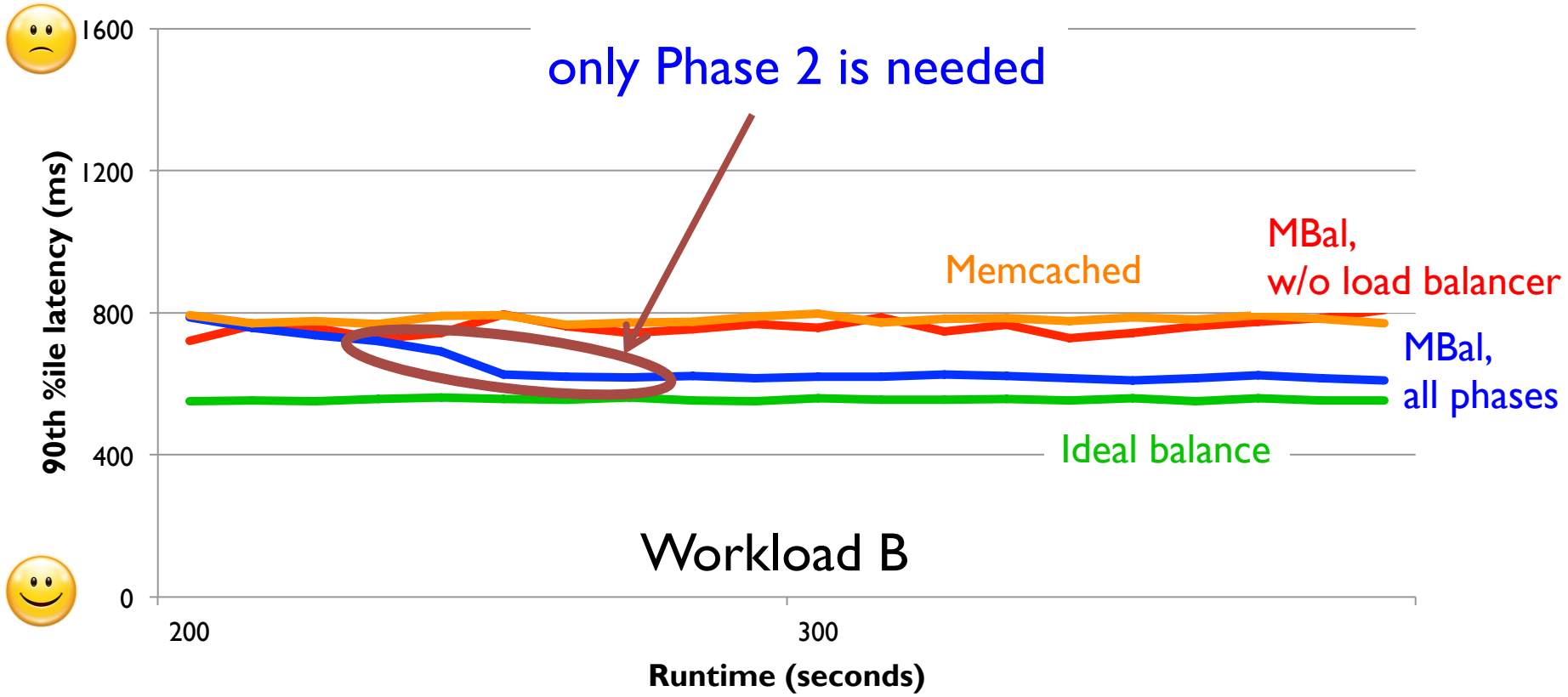
Load balancer evaluation

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



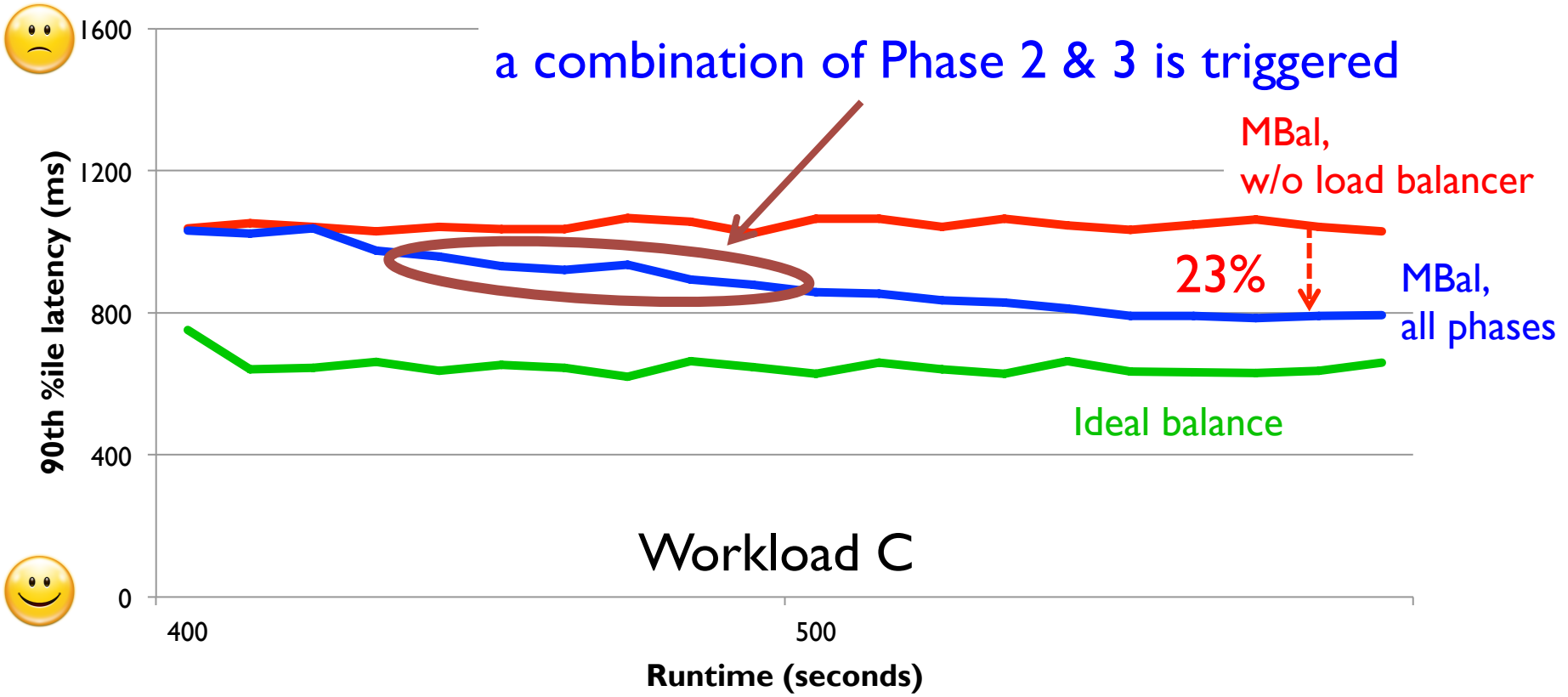
Load balancer evaluation

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

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

Improves "BANG for the buck"

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]

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]

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]

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

Thank you!

http://research.cs.vt.edu/dssl/

Yue Cheng Aayush Gupta Ali R. Butt

Backup Slides

Memcached is desirable

- Quick deployment
- Ease of use

Memcached deployment **in the Cloud**

- Quick deployment
- Ease of use
- Elastic scale-up
- Elastic scale-out

Memcached deployment **in the Cloud**

- Quick deployment
- Ease of use
- Elastic scale-up
- Elastic scale-out

Instance type	vCPU	ECU	N/w (Gbps)	Price/hr
m1.small	1	1	0.1	$0.044
m3.medium	1	3	0.5	$0.070
c3.large	2	7	0.6	$0.105
m3.xlarge	4	13	0.7	$0.280
c3.2xlarge	8	28	1	$0.420
c3.8xlarge	32	108	10	$1.680

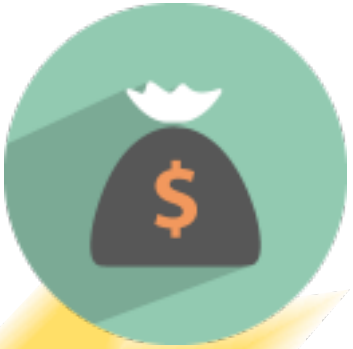
Memcached deployment **in the Cloud**

- Quick deployment
- Ease of use
- Elastic scale-up
- Elastic scale-out

Instance type	vCPU	ECU	N/w (Gbps)	Price/hr
m1.small	1	1	0.1	$0.044
m3.medium	1	3	0.5	$0.070
c3.large	2	7	0.6	$0.105
m3.xlarge	4	13	0.7	$0.280
c3.2xlarge	8	28	1	$0.420
c3.8xlarge	32	108	10	$1.680

“Decision paralysis ...”

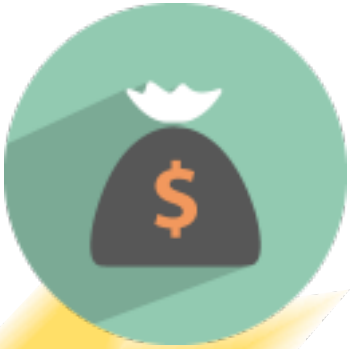
Instance type	vCPU	ECU	N/w (Gbps)	Price/hr
m1.small	1	1	0.1	$0.044
m3.medium	1	3	0.5	$0.070
c3.large	2	7	0.6	$0.105
m3.xlarge	4	13	0.7	$0.280
c3.2xlarge	8	28	1	$0.420
c3.8xlarge	32	108	10	$1.680



Getting the most “BANG for the buck”

"Decision paralysis ..."

Instance type	vCPU	ECU	N/w (Gbps)	Price/hr
m1.small	1	1	0.1	$0.044
m3.medium	1	3	0.5	$0.070
c3.large	2	7	0.6	$0.105
m3.xlarge	4	13	0.7	$0.280
c3.2xlarge	8	28	1	$0.420
c3.8xlarge	32	108	10	$1.680

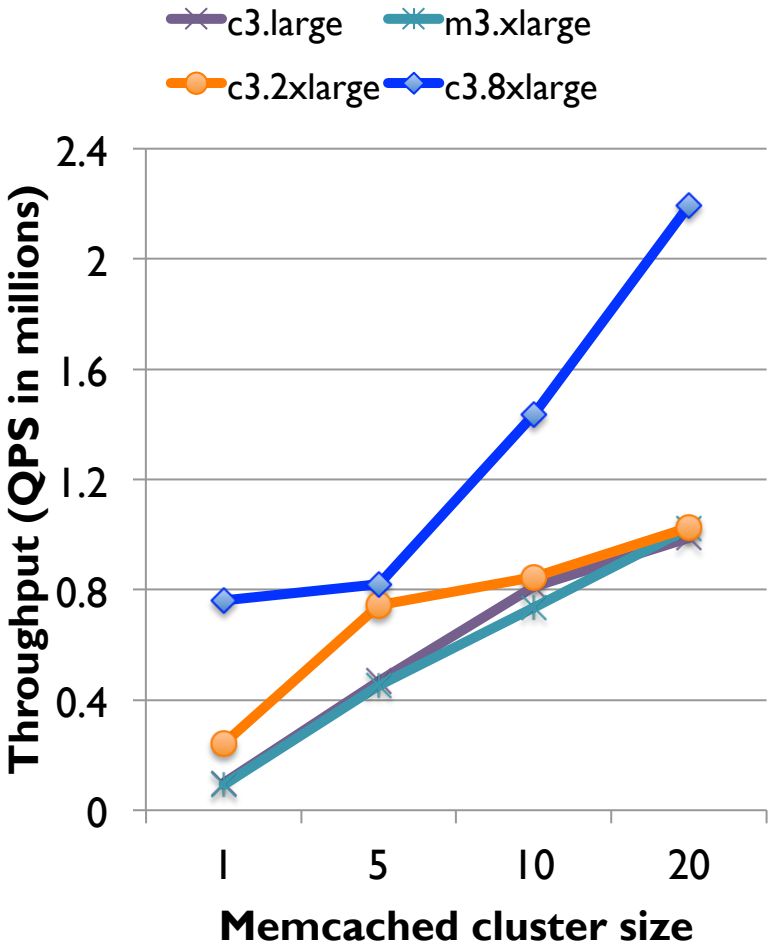


Getting the most "BANG for the buck"

- Desire 1: performance
- Desire 2: $ efficiency

Desire 1: Performance

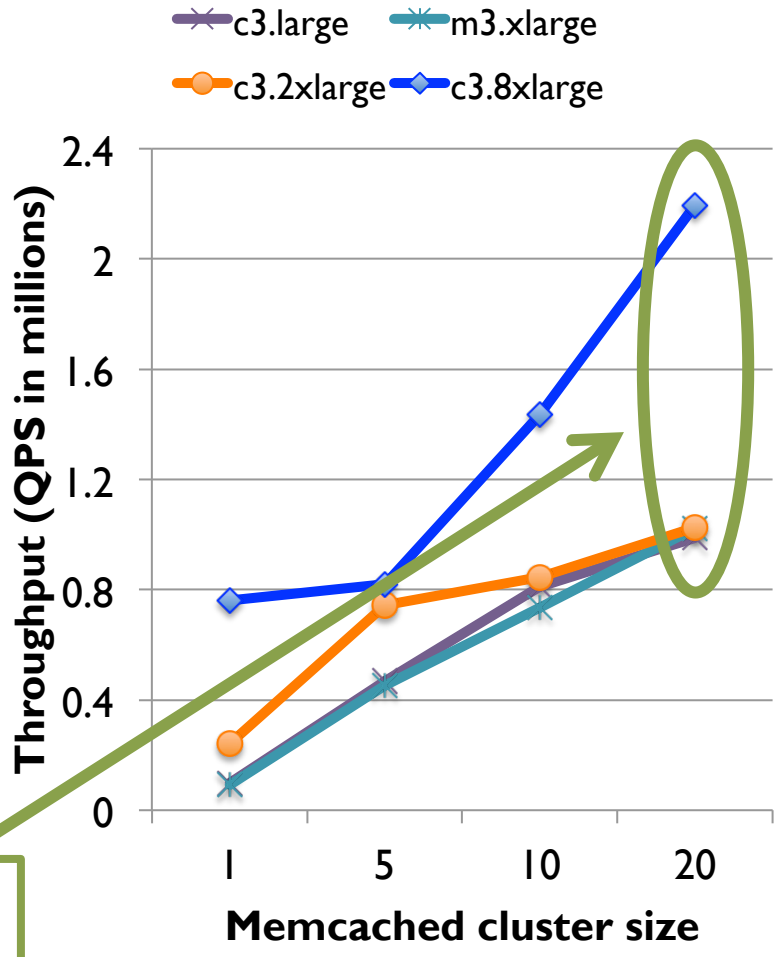
Instance type	vCPU	ECU	N/w (Gbps)	Price/hr
m1.small	1	1	0.1	$0.044
m3.medium	1	3	0.5	$0.070
c3.large	2	7	0.6	$0.105
m3.xlarge	4	13	0.7	$0.280
c3.2xlarge	8	28	1	$0.420
c3.8xlarge	32	108	10	$1.680



95% GET, 5% SET, Uniform

Desire 1: Performance

Instance type	vCPU	ECU	N/w (Gbps)	Price/hr
m1.small	1	1	0.1	$0.044
m3.medium	1	3	0.5	$0.070
c3.large	2	7	0.6	$0.105
m3.xlarge	4	13	0.7	$0.280
c3.2xlarge	8	28	1	$0.420
c3.8xlarge	**32**	**108**	**10**	**$1.680**

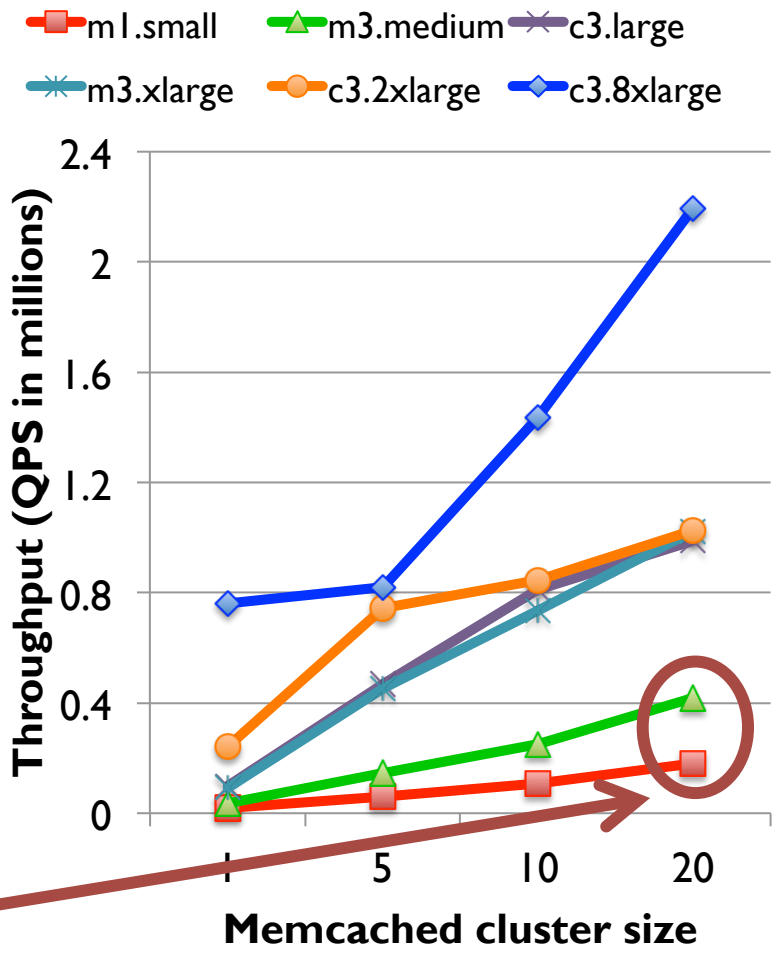


Network is the bottleneck!

95% GET, 5% SET, Uniform

Desire 1: Performance

Instance type	vCPU	ECU	N/w (Gbps)	Price/hr
m1.small	1	1	0.1	$0.044
m3.medium	1	3	0.5	$0.070
c3.large	2	7	0.6	$0.105
m3.xlarge	4	13	0.7	$0.280
c3.2xlarge	8	28	1	$0.420
c3.8xlarge	32	108	10	$1.680

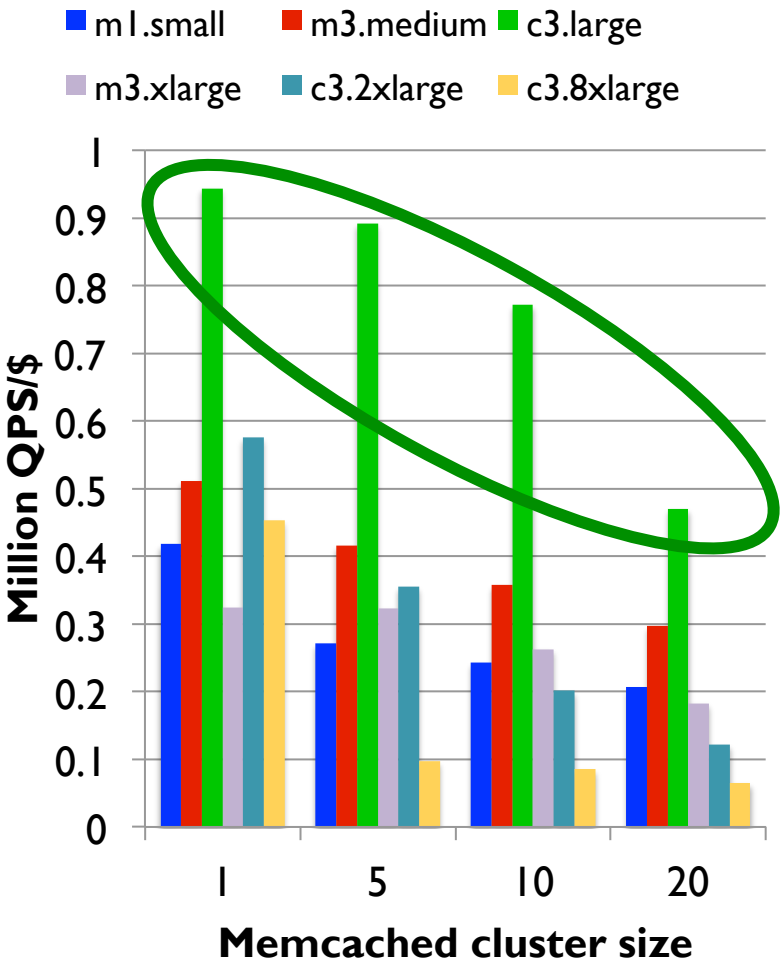


CPU is the bottleneck!

95% GET, 5% SET, Uniform

Desire 2: $ efficiency

Instance type	vCPU	ECU	N/w (Gbps)	Price/hr
m1.small	1	1	0.1	$0.044
m3.medium	1	3	0.5	$0.070
c3.large	**2**	**7**	**0.6**	**$0.105**
m3.xlarge	4	13	0.7	$0.280
c3.2xlarge	8	28	1	$0.420
c3.8xlarge	32	108	10	$1.680



$ efficiency = QPS/$

95% GET, 5% SET, Uniform

Desire 2: $ efficiency

Instance type	vCPU	ECU	N/w (Gbps)	Price/hr
m1.small	1	1	0.1	$0.044
c3.8xlarge	32	108	10	$1.680

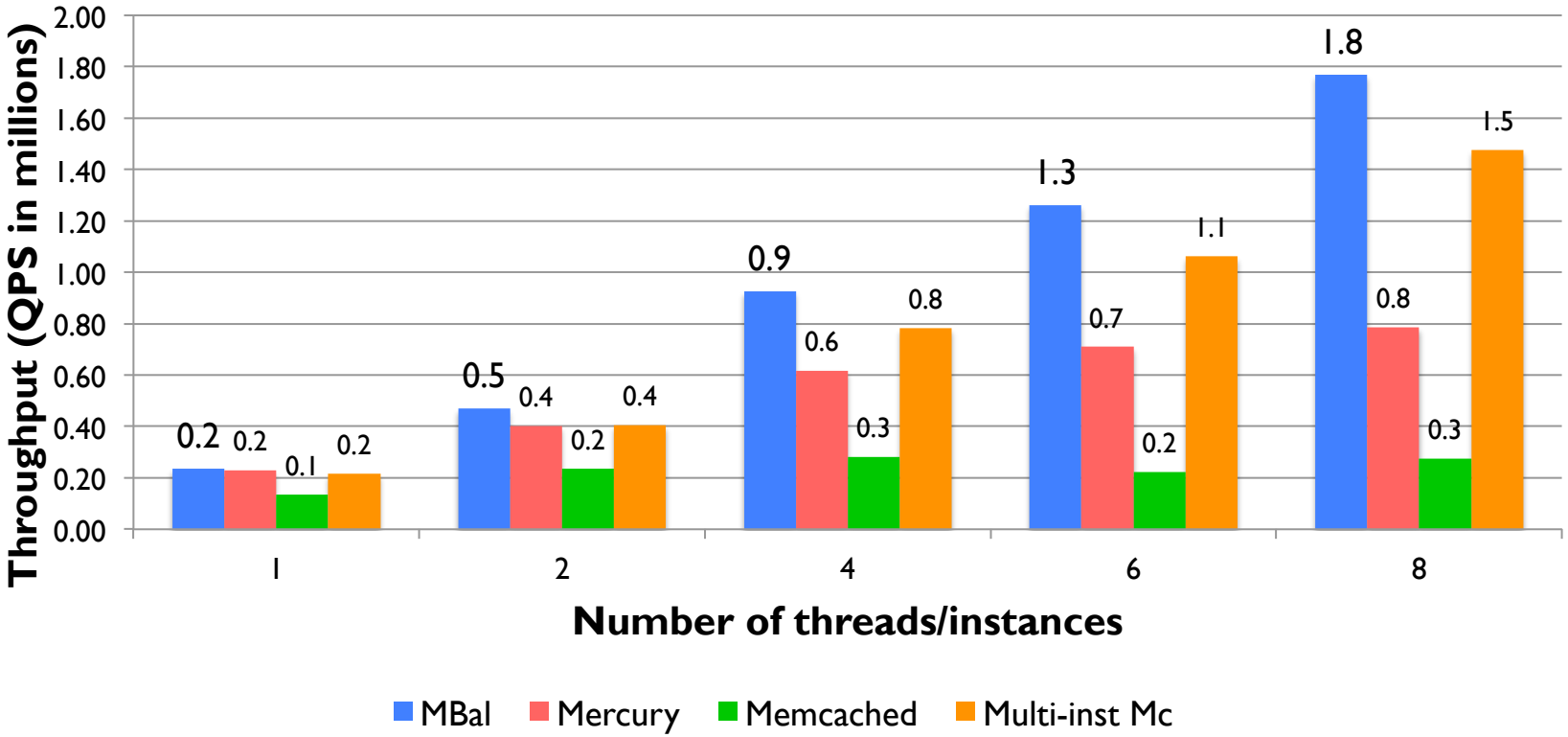
- Adding more resources is NOT a good solution
- Extra CPU capacity is wasted in the cloud
- Instance with modest CPU offers best $ efficiency

$ efficiency = QPS/$

95% GET, 5% SET, Uniform

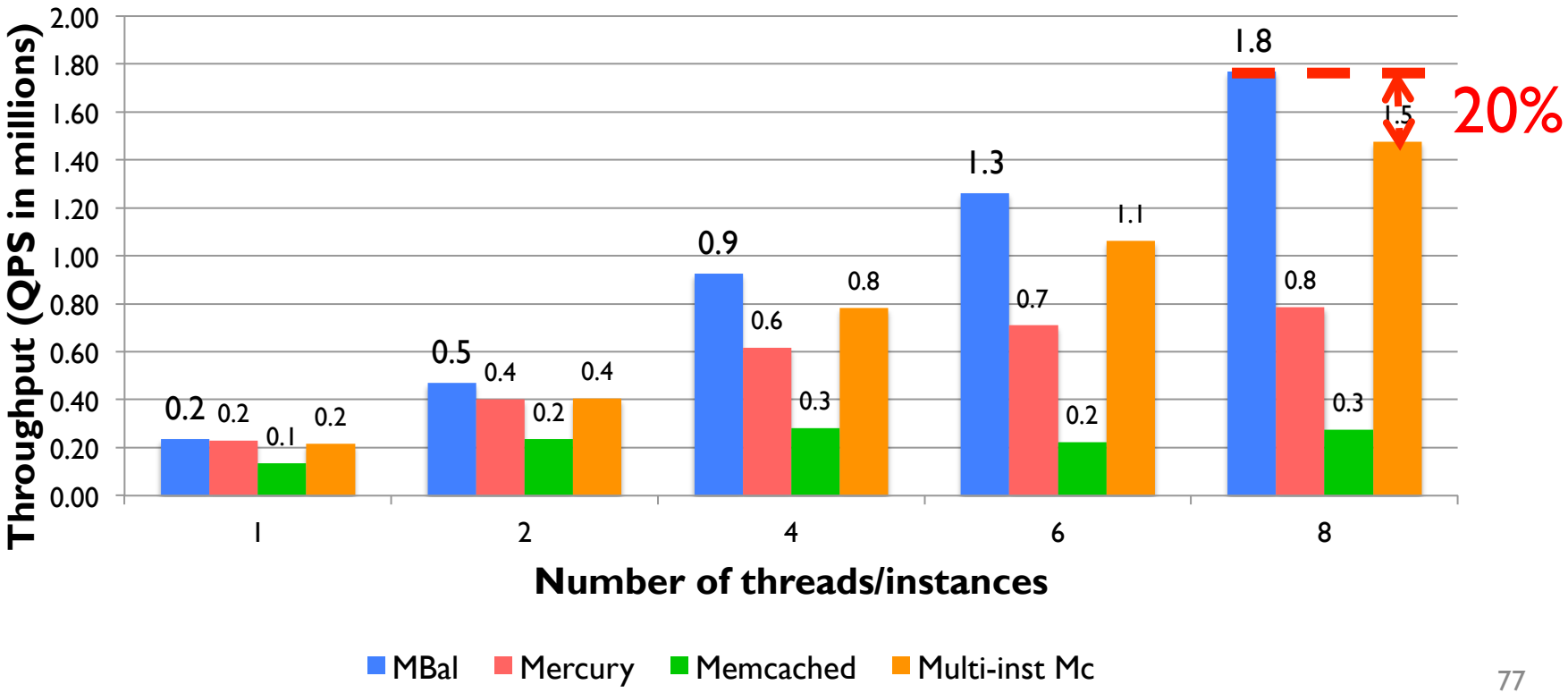
MBal evaluation – complete system

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



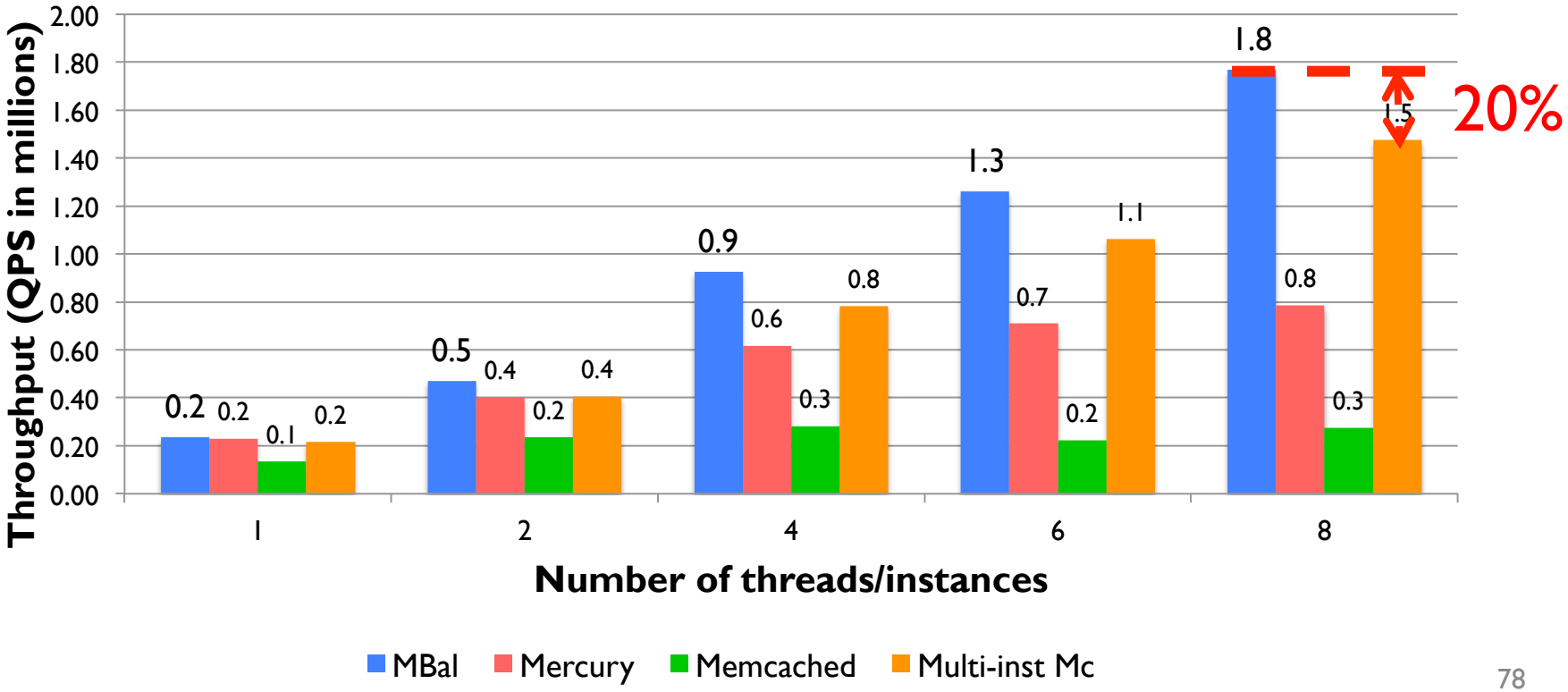
MBal evaluation – complete system

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

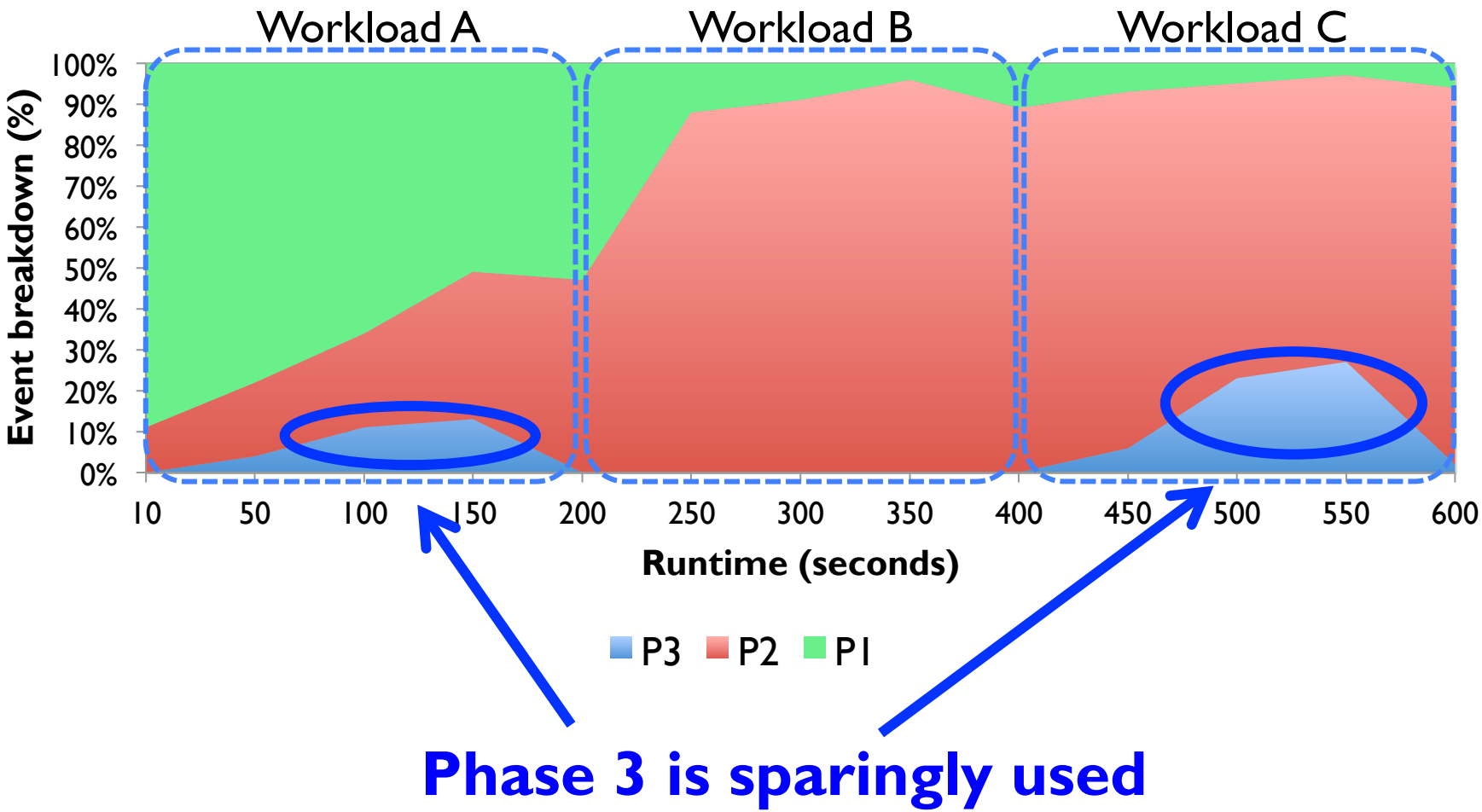


MBal evaluation – complete system

✓ MBal uses lightweight CPU cache-aligned bucket locks!

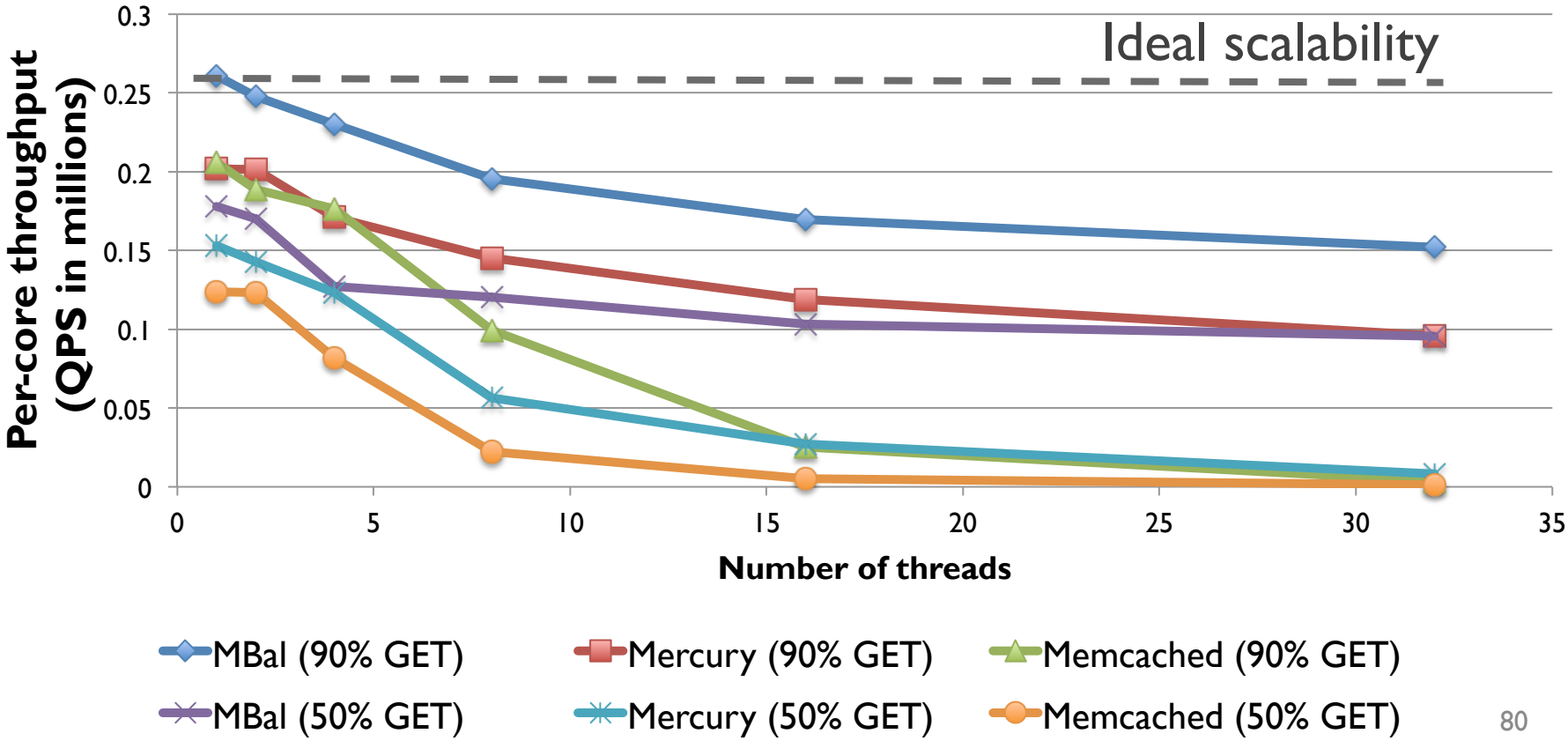


Event breakdown in E2E test



Multi-core scalability

- 32-core 2GHz, 64GB DRAM
- memaslap with MultiGET, 16B key 32B value
- 10GbE network



99th percentile latency vs. throughput