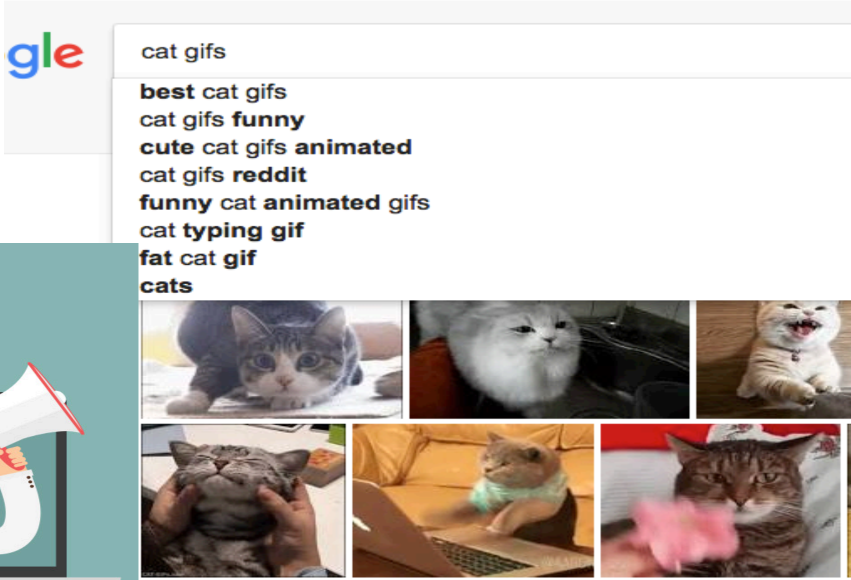
µSuite & µTune: Auto-Tuned Threading for OLDI Microservices

Akshitha Sriraman, Thomas F. Wenisch

University of Michigan

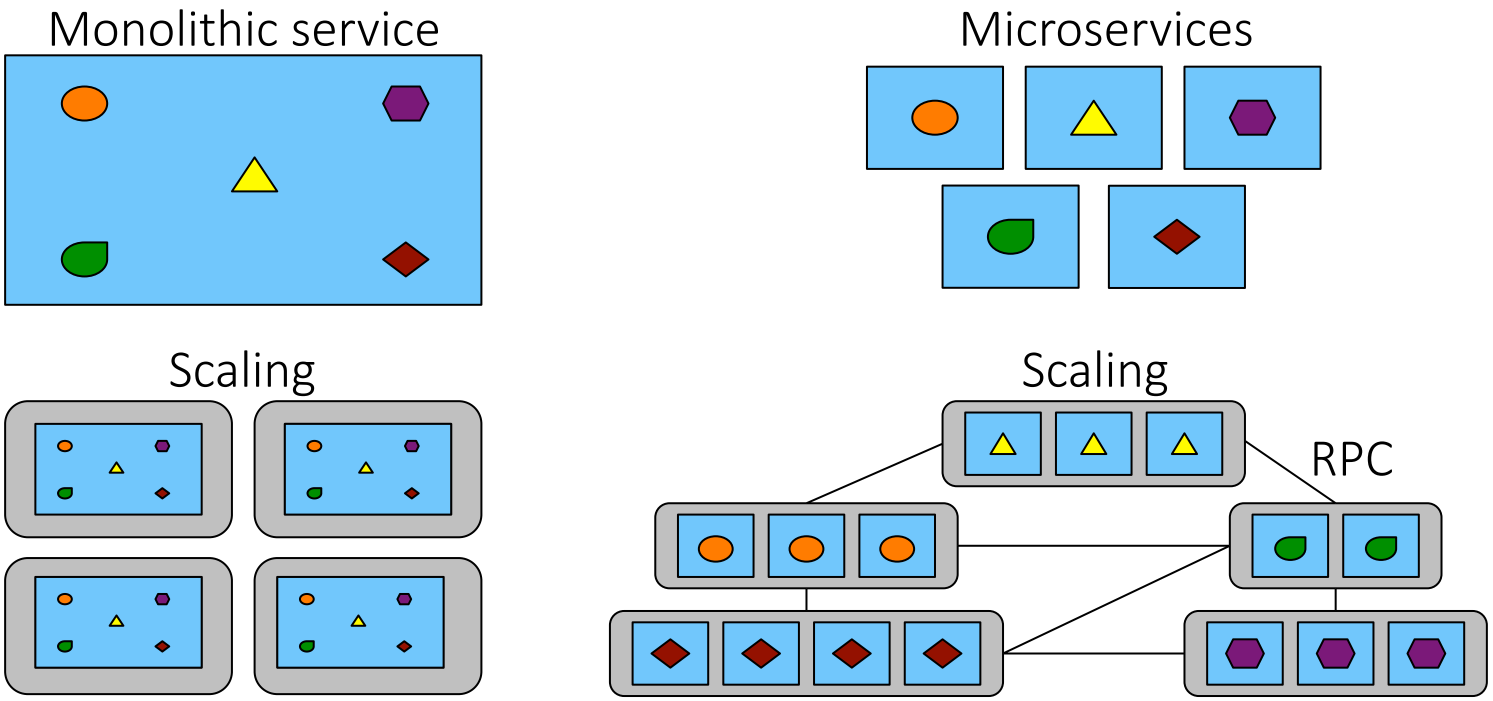
Applications Driving Architectures

On-Line Data Intensive (OLDI) Services



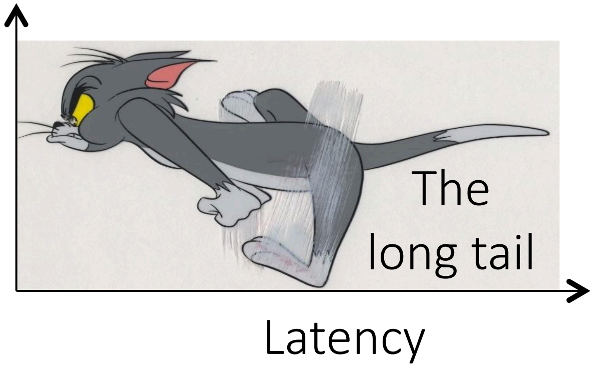
Must meet stringent Service Level Objectives (SLOs)

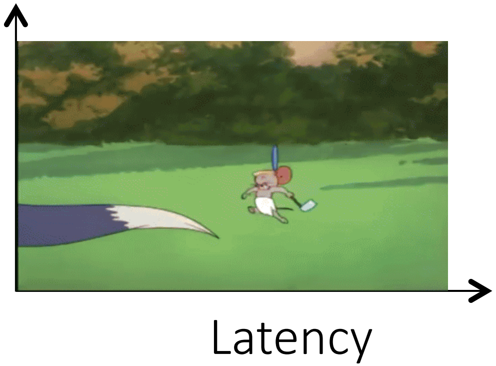
OLDI: From Monoliths to Microservices



From >100ms SLOs to sub-ms SLOs

Tail Latency



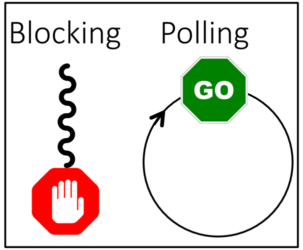


- SLOs are impacted by the 99th+% (tail) latency
- Negatively affects user experience

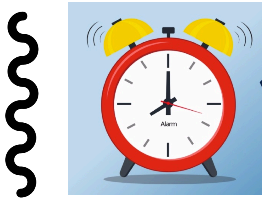
Goal: Minimize microservice tail latency

Threading Effects on Tails for Monoliths

- Our focus: Sub-ms overheads due to threading design



Lock contention



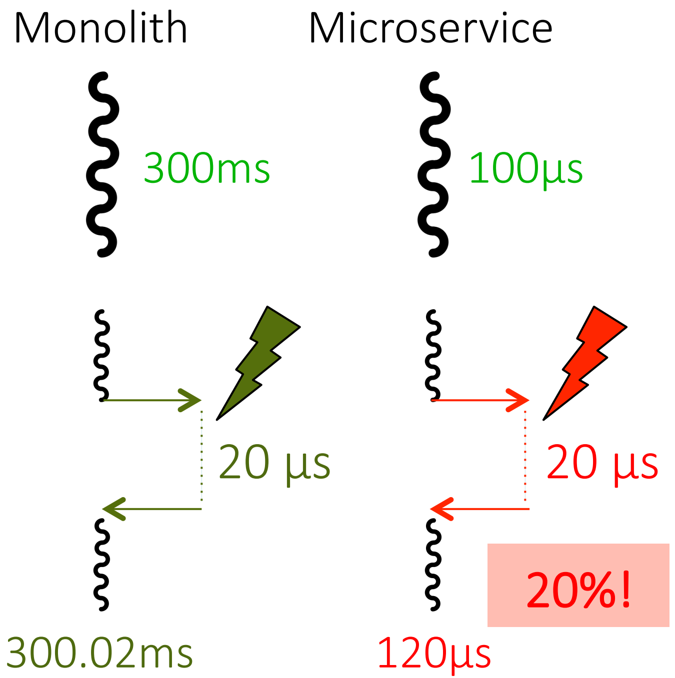
Thread wakeups

Spurious context switch

Threading-induced OS/network overheads are minor for monoliths

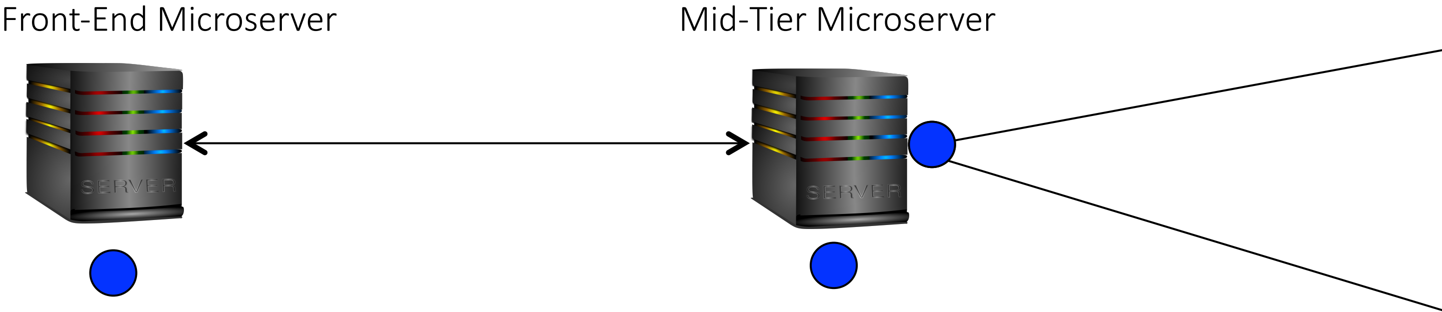
Threading Effects on Microservice Tails

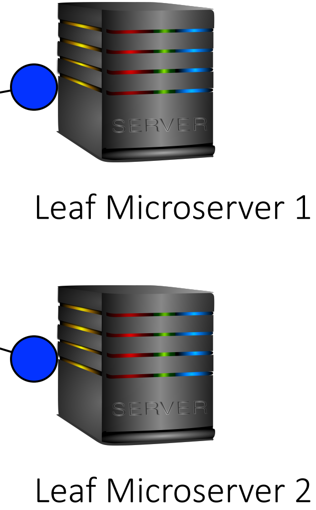
- Threading can significantly impact microservice SLOs



Prior threading conclusions must be revisited for microservices

Mid-tier Faces More Threading Overheads

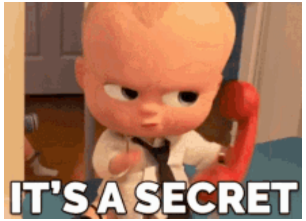




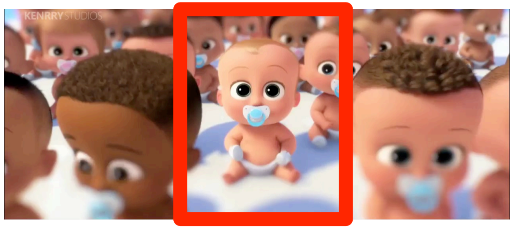
- Mid-tier – subject to more threading overheads
  - Manages RPC fan-out to many leaves
  - RPC layer interactions dominate computation

Threading overheads must be characterized for ***mid-tier*** microservices

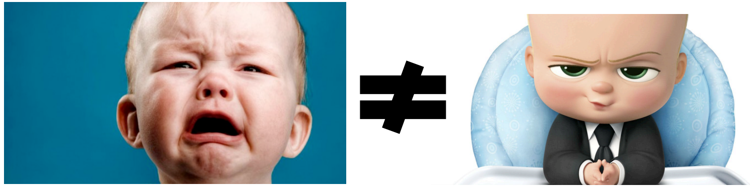
Need for a Microservice Benchmark Suite



Closed-source
[Ayers '18]



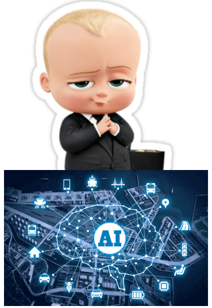
Only one workload
[Hsu '15]



Not representative
[Zhu '16]

Monolithic Architectures
[Ferdman '12]

Only leaf nodes
[Lo '14]



Domain-specific
[Hauswald '15]

No open-source benchmark sufficiently represents microservices

Contributions

μSuite: Benchmark suite of OLDI services composed of microservices [1]

↓

Taxonomy of threading models: Implications of threading designs [2]

↓

μTune: Load adaptation system to tune threading models & improve tails [2]

↓

Achieve **1.9x** tail latency speedup over state-of-the-art adaptations [2]

[1] A. Sriraman, T.F. Wenisch. μSuite: A Benchmark Suite for Microservices. International Symposium on Workload Characterization **(IISWC) 2018**.

[2] A. Sriraman, T.F. Wenisch. μTune: Auto-Tuned Threading for OLDI Microservices Operating Systems Design and Implementation **(OSDI) 2018**.

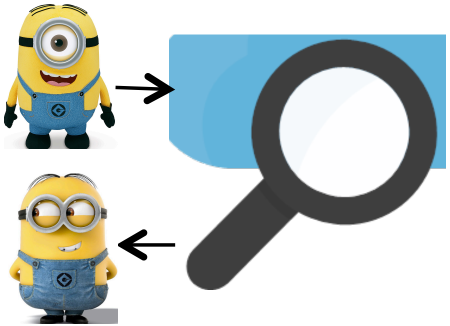
Outline

- µSuite: Description of services & microservices
- Show how µSuite facilitates future research

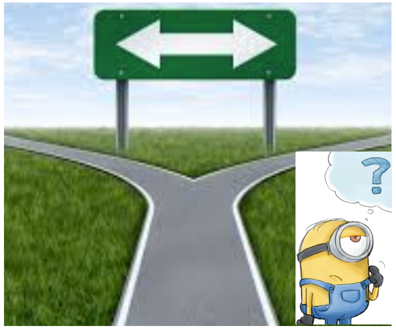
- A taxonomy of threading models
  - Characterize threading effects on microservice tails
- µTune: Dynamic load adaptation system that improves tail latency
- Evaluation

µSuite

HDSearch



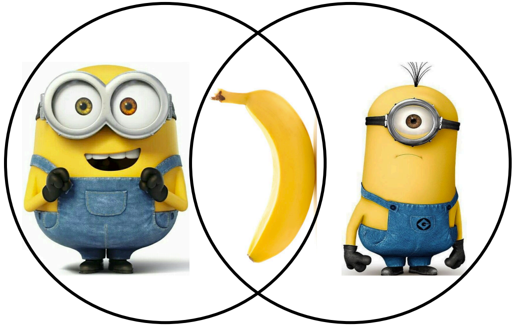
Leaf compute bound



Router

Variability in scale-out

Set Algebra

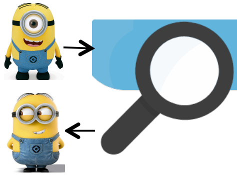


Variability in leaf compute

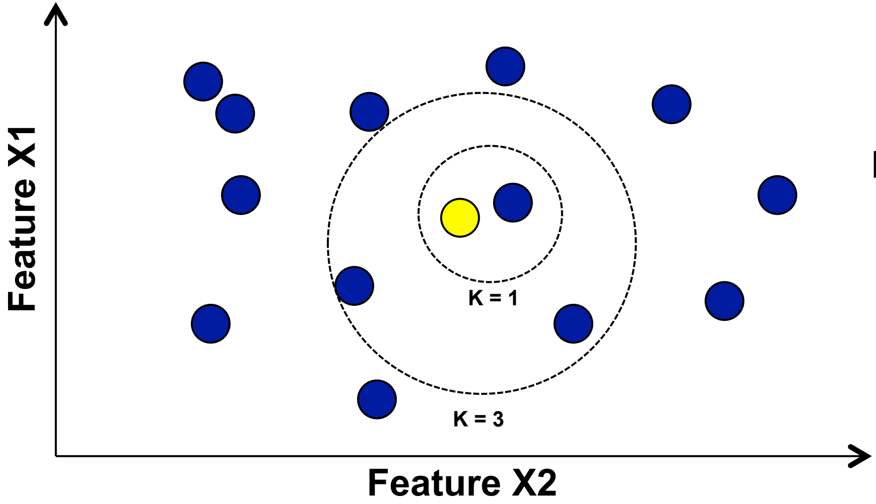
Recommend

Variability in mid-tier compute

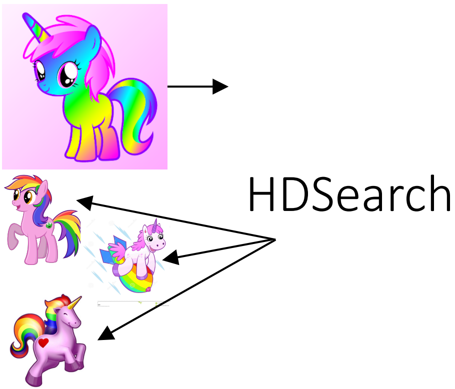
Benchmark 1: HDSearch



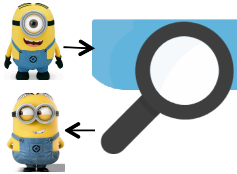
- Content-based search for image similarity
- Leaf compute bound - mid-tier has high threading overheads



K = # nearest neighbors

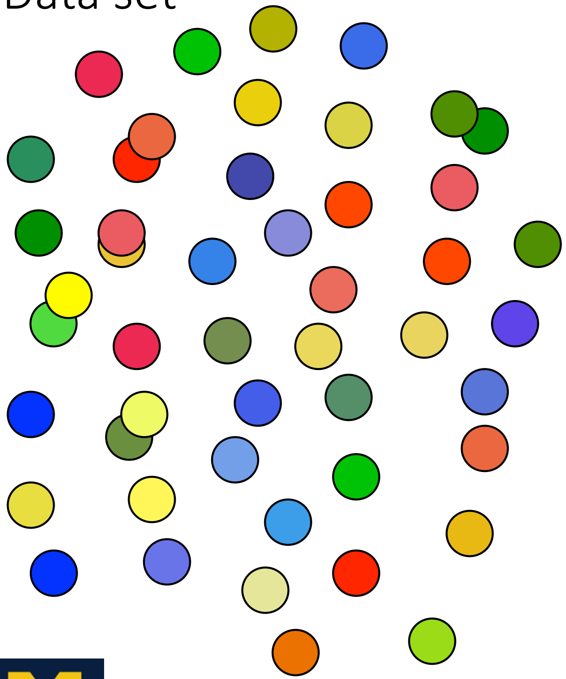


HDSearch: Locality Sensitive Hashing



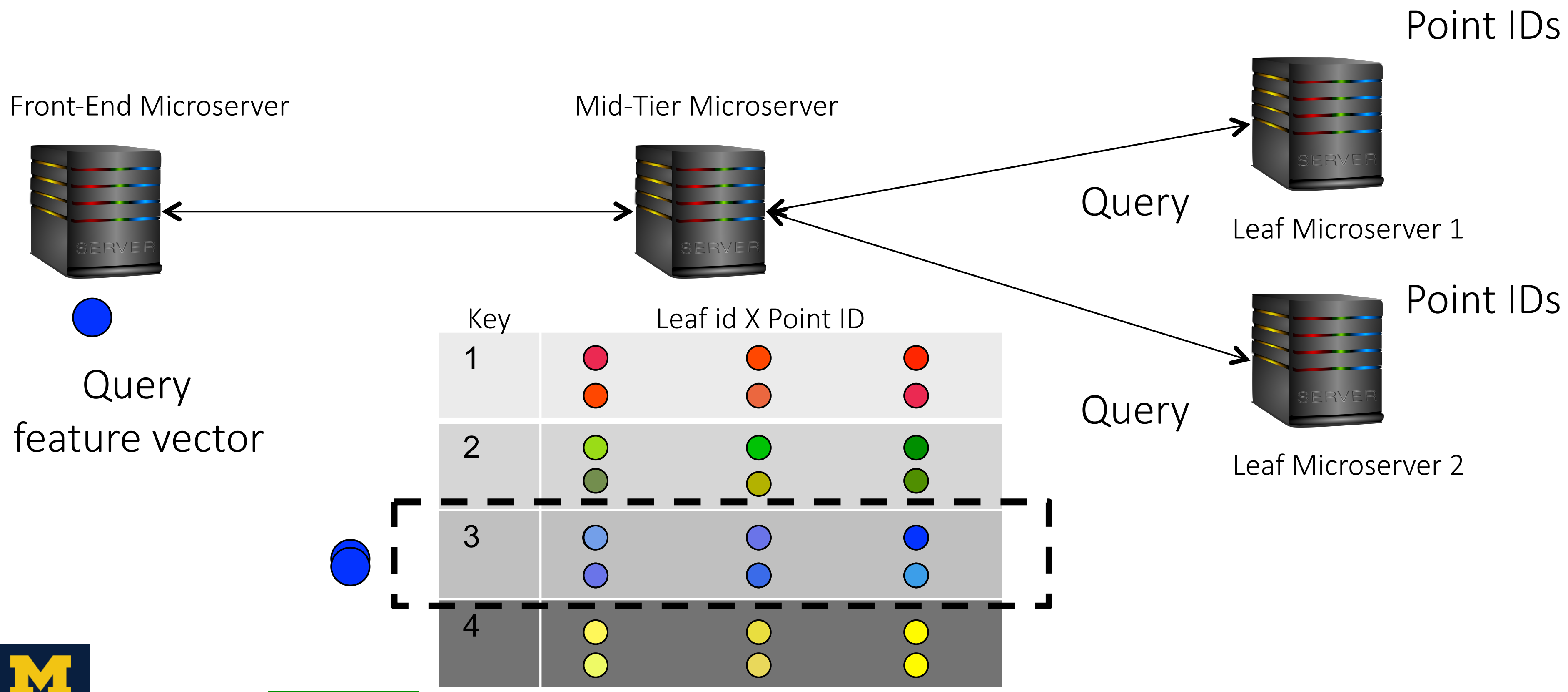
Reduces nearest neighbor computation time

Data set

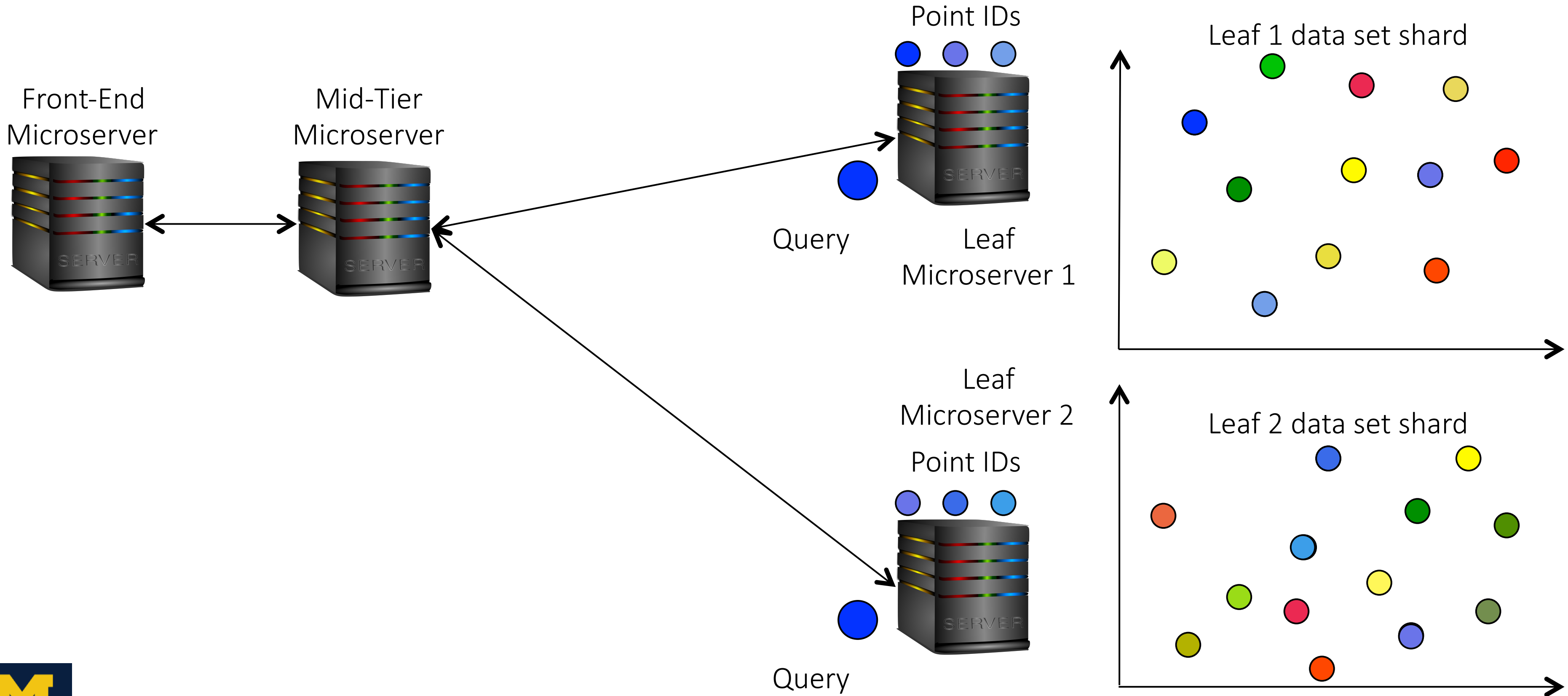


Key	Potentially near-by point IDs
1	
2	
3	
4	

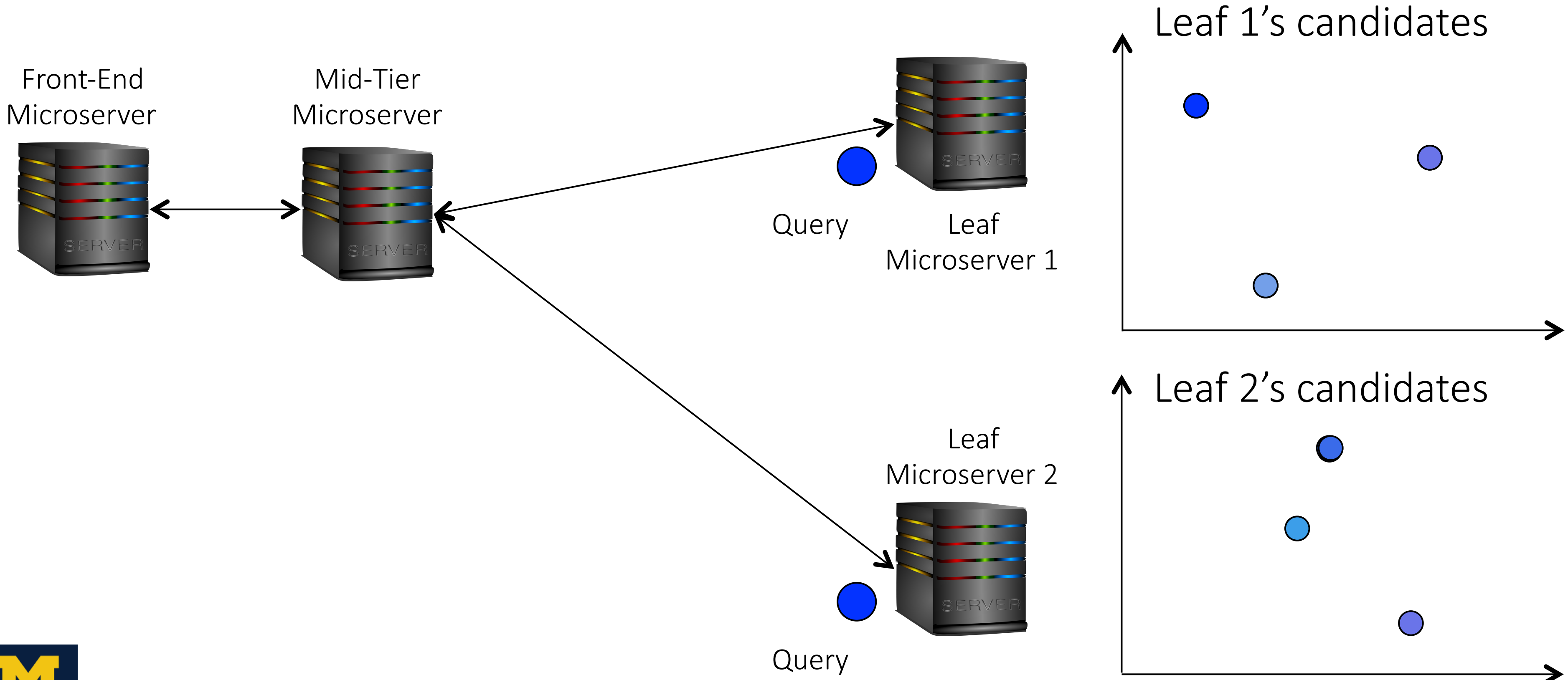
HDSearch: Operation



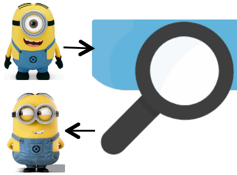
HDSearch: Operation

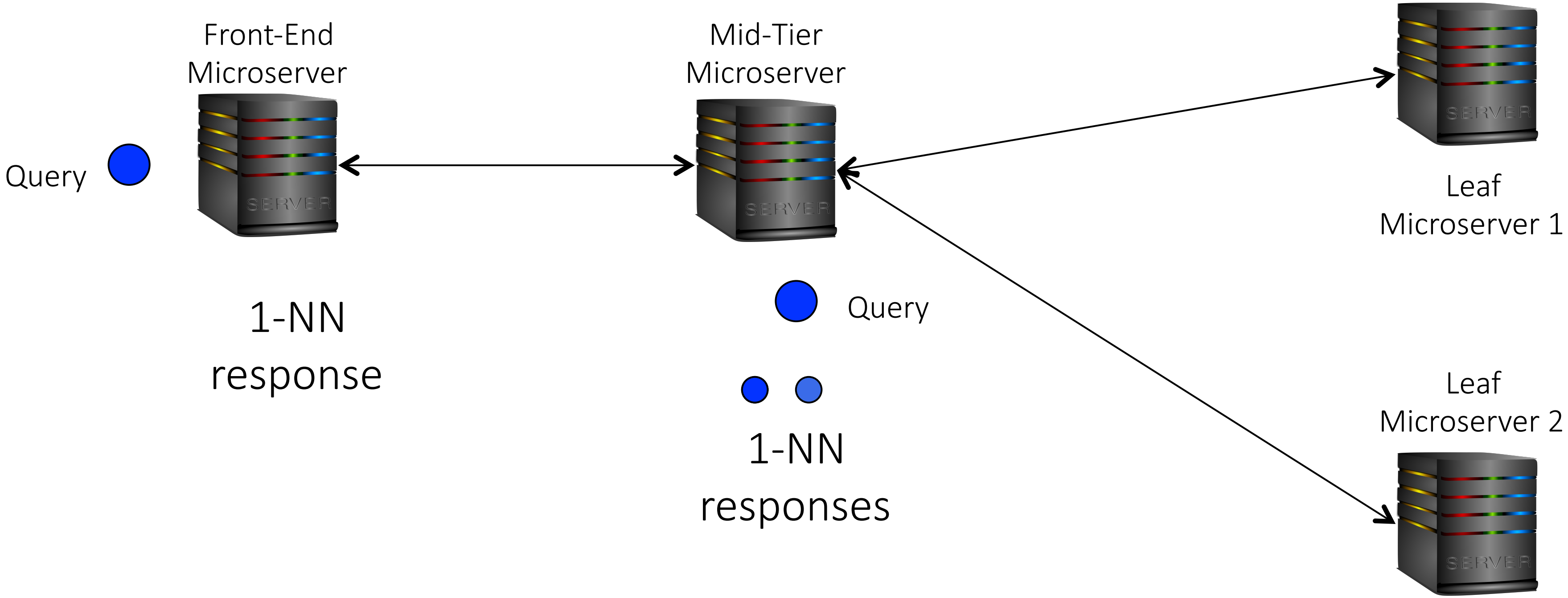


HDSearch: Operation



HDSearch: Operation

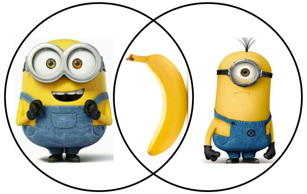




Other µSuite Services

Benchmark 2: Router

- Fault tolerance by replication
- GET:SET asymmetry
- Varied scale-out per request



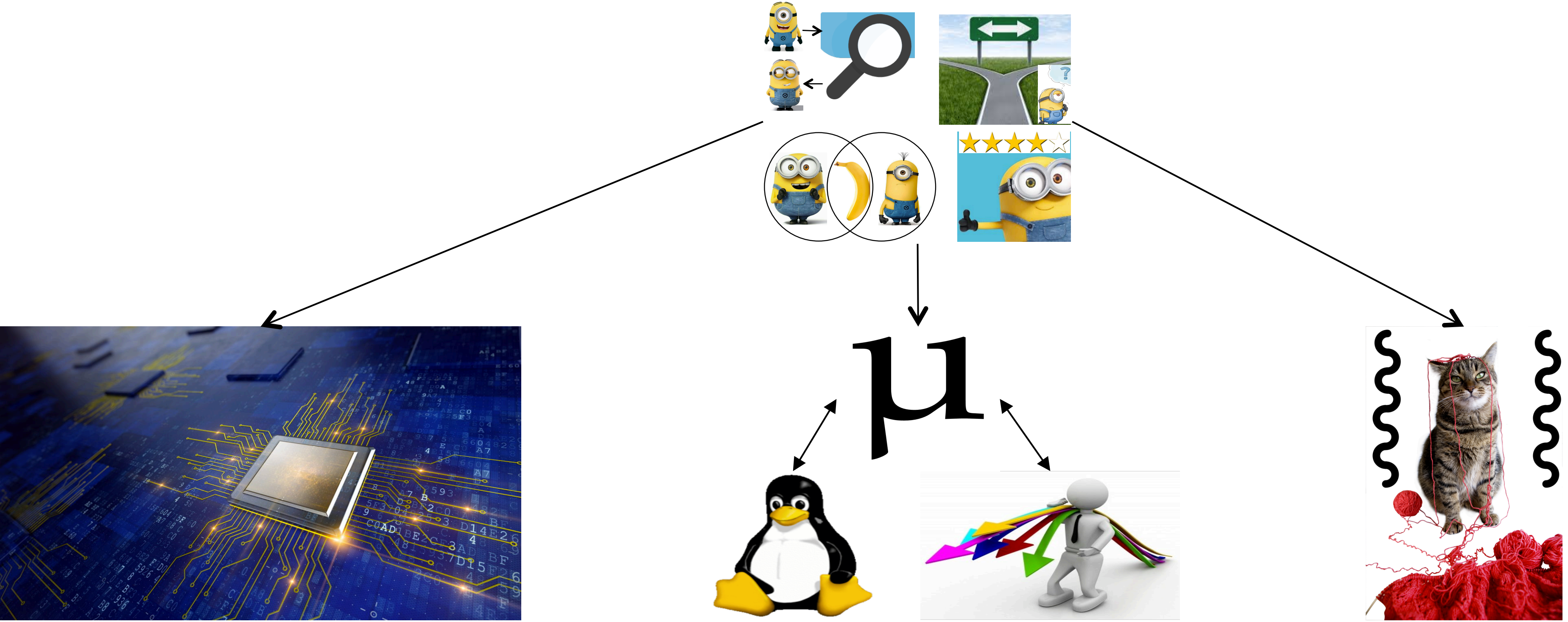
Benchmark 3: Set Algebra

- Inverted index of posting lists
- Large variability in leaf compute

Benchmark 4: Recommend

- Collaborative filtering
- Mid-tier does little work

μSuite Can Facilitate Future Research



Contributions

µSuite: Benchmark suite of OLDI services composed of microservices [1]

↓

Taxonomy of threading models: Implications of threading designs [2]

↓

µTune: Load adaptation system to tune threading models & improve tails [2]

↓

Achieve 1.9x tail latency speedup over state-of-the-art adaptations [2]

[1] A. Sriraman, T.F. Wenisch. µSuite: A Benchmark Suite for Microservices. International Symposium on Workload Characterization **(IISWC) 2018**.

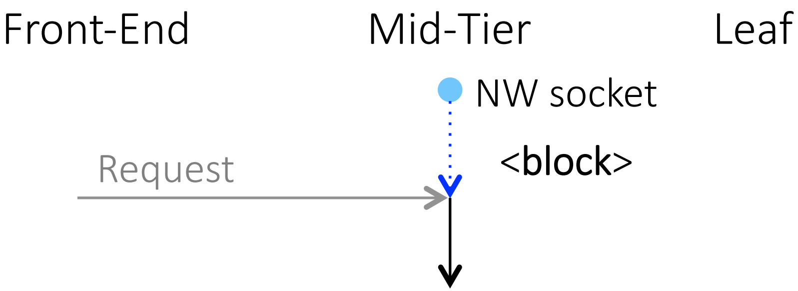
[2] A. Sriraman, T.F. Wenisch. µTune: Auto-Tuned Threading for OLDI Microservices Operating Systems Design and Implementation **(OSDI) 2018.**

Threading Designs

- Taxonomy of threading models
- Threading dimensions:
  - Block vs. Poll
  - In-Line vs. Dispatch
  - Synchronous vs. Asynchronous

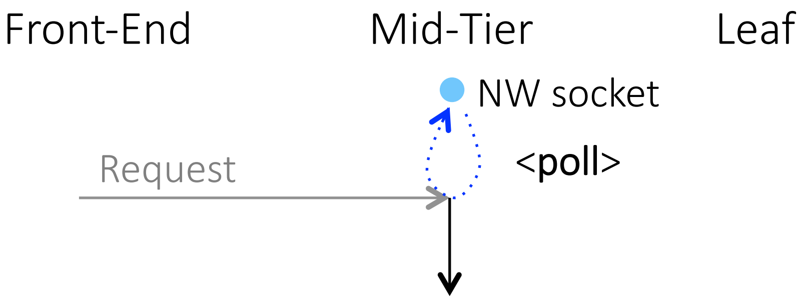
Threading Dimensions: Block vs. Poll

Block or Interrupt-Driven



- Low cost: avoids fruitless poll-loops
- High thread wakeup latency

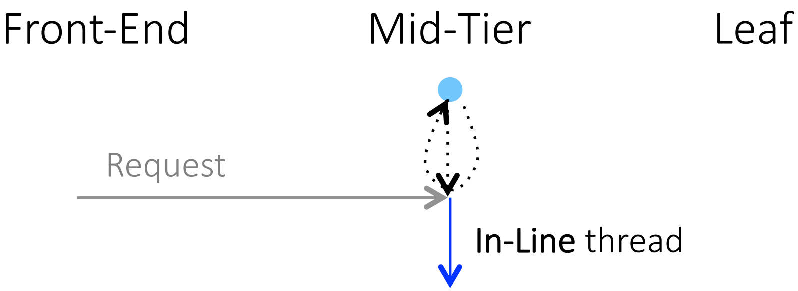
Poll



- Low latency: avoids thread wakeups
- Many poll threads cause contention

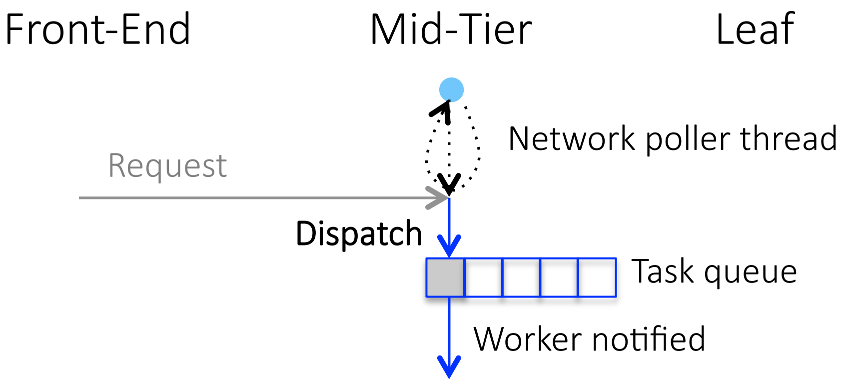
Threading Dimensions: In-Line vs. Dispatch

In-Line



- Better for short queries: no hand-off
- Many in-line threads may contend

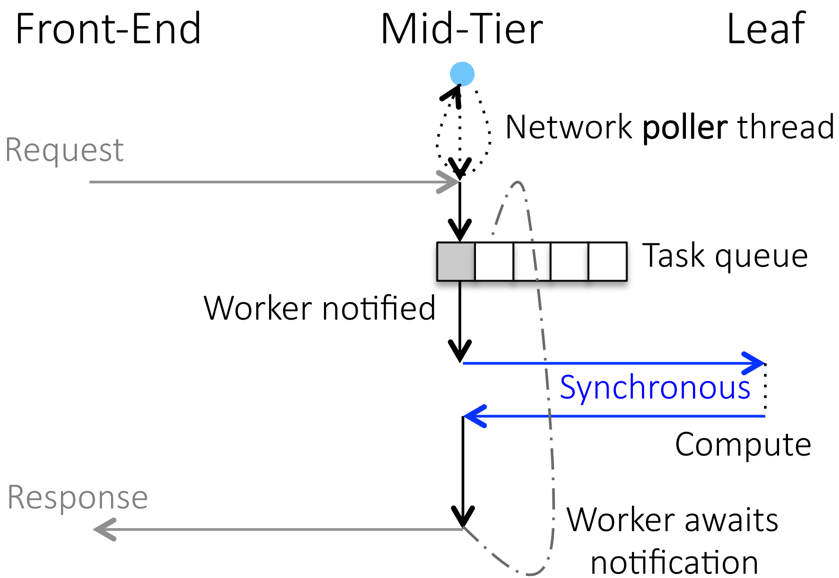
Dispatch



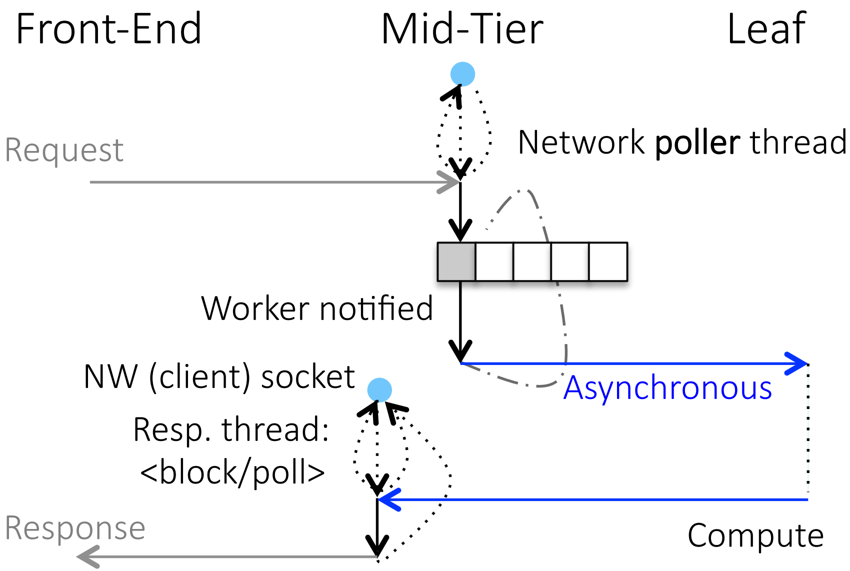
- Better network poller locality
- Harder to program: thread-safety

Threading Dimensions: Sync. vs. Async.

Synchronous

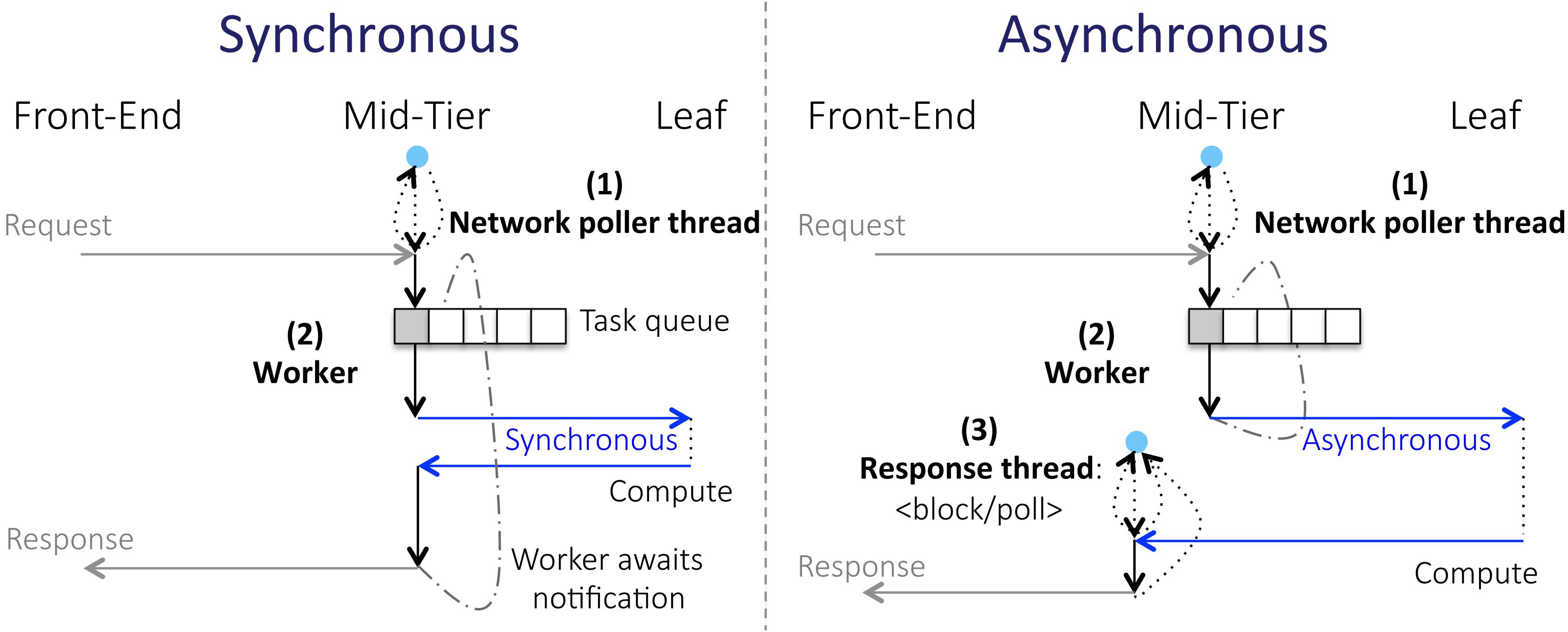


Asynchronous



Synchronous & asynchronous designs are built separately

Threading Dimensions: Thread Pools



A Taxonomy of Threading Models

Synchronous

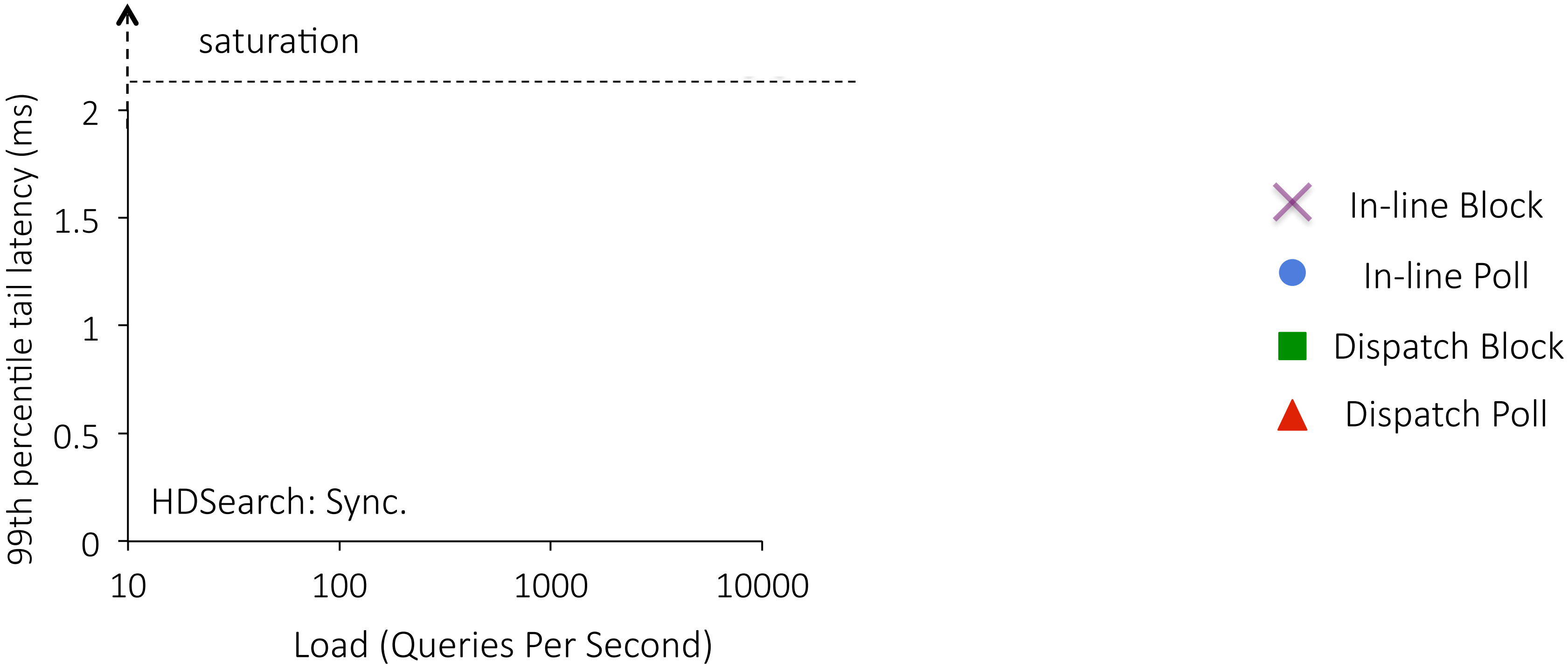
	Block	Poll
In-line	SIB	SIP
Dispatch	SDB	SDP

Asynchronous

	Block	Poll
In-line	AIB	AIP
Dispatch	ADB	ADP

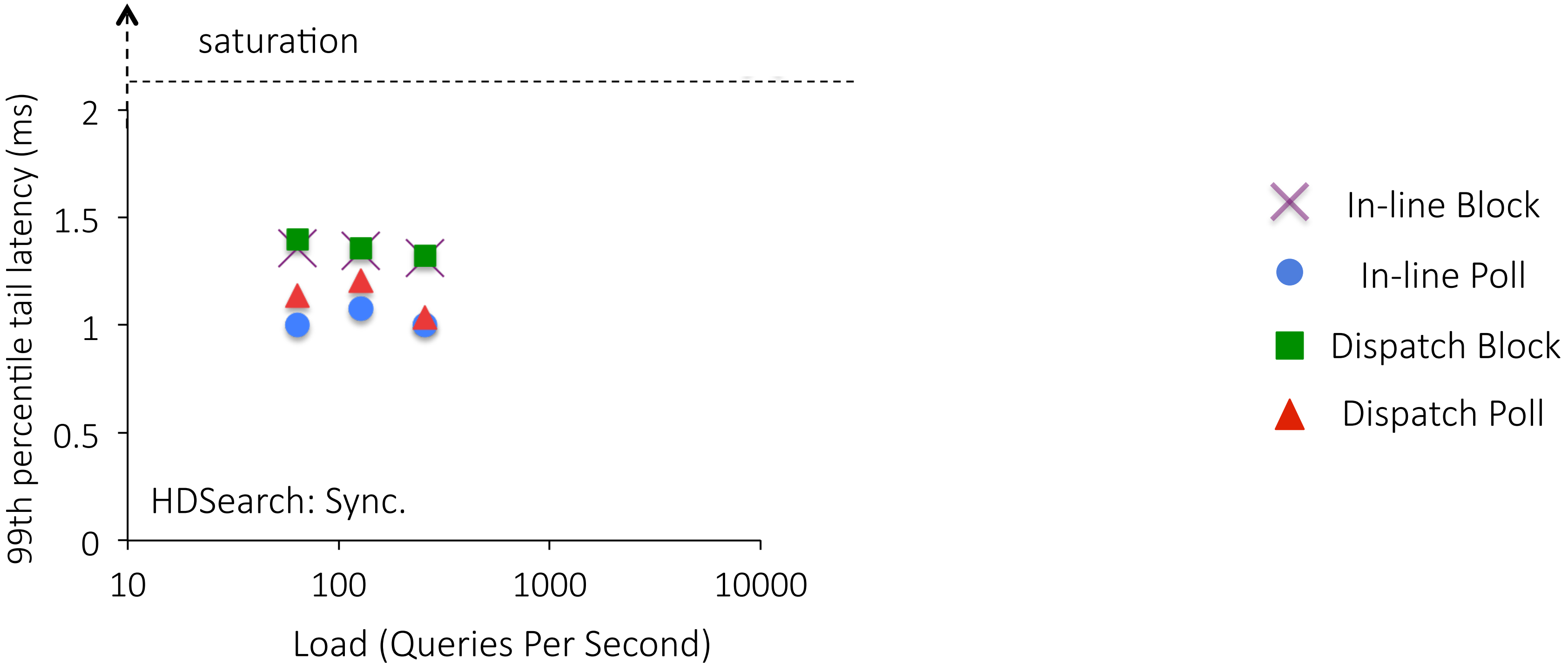
Characterize varying thread pool sizes for each functionality

Latency Tradeoffs Across Threading Models



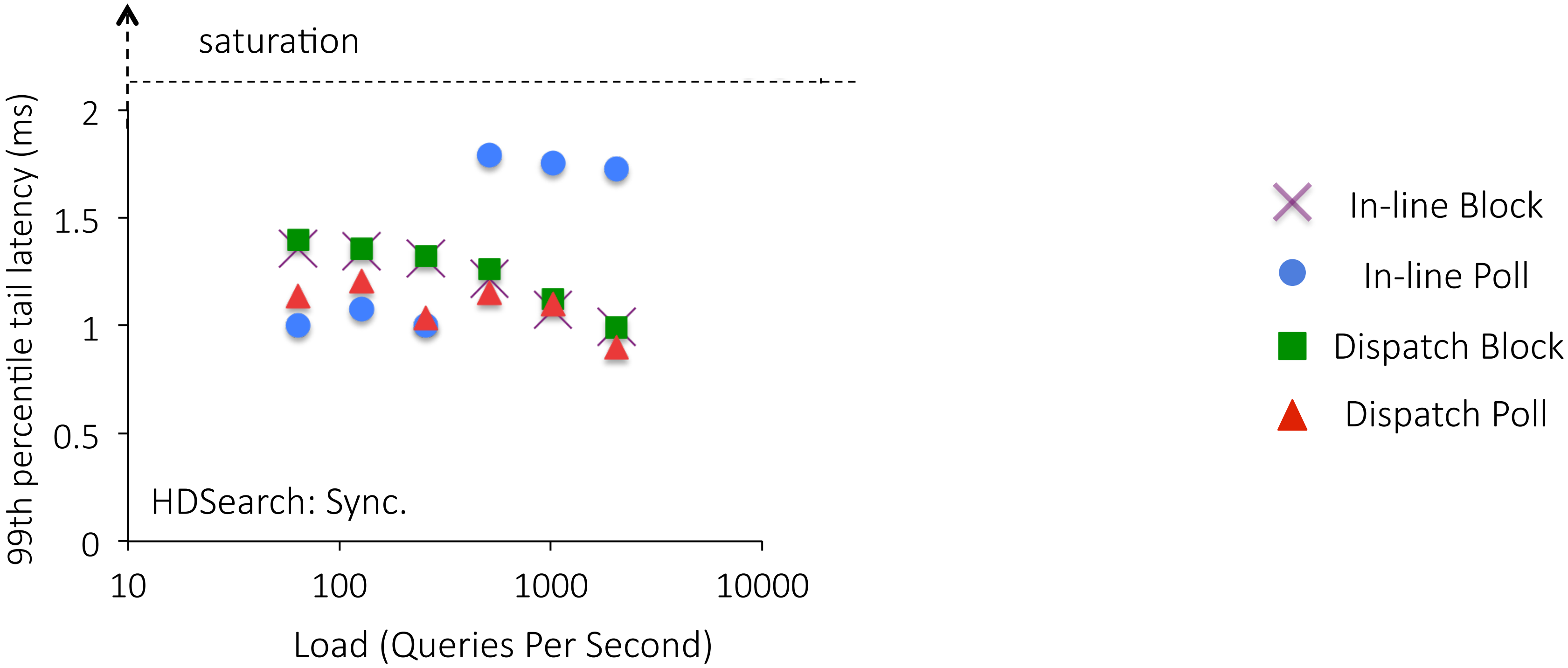
In-line Poll has lowest low-load latency: Avoids thread wakeup delays

Latency Tradeoffs Across Threading Models



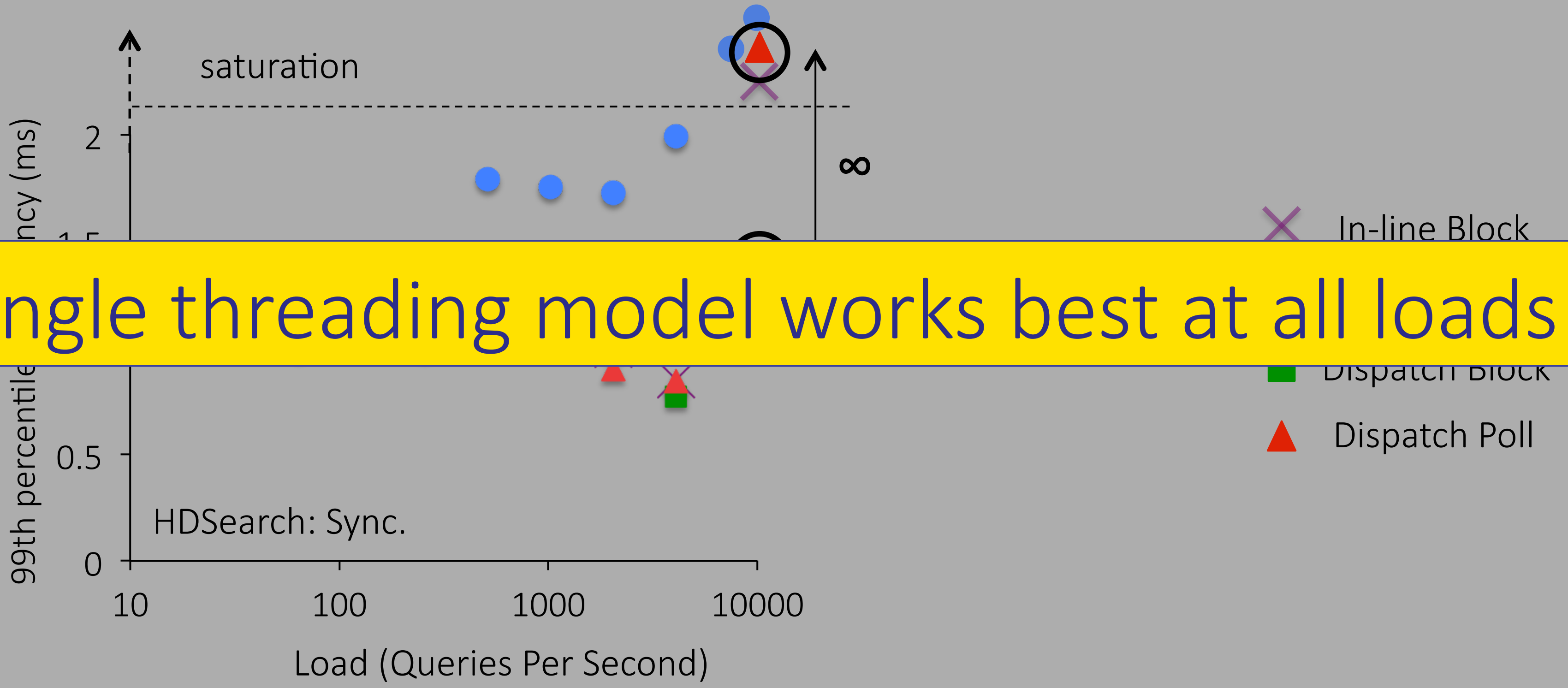
In-Line Poll faces contention; Dispatch Poll with one network poller is best

Latency Tradeoffs Across Threading Models



Dispatch Block is best at high load as it does not waste CPU

Latency Tradeoffs Across Threading Models



No single threading model works best at all loads

Need for Automatic Load Adaptation: μTune

- Threading choice can significantly affect tail latency
- Threading latency trade-offs are not obvious
- Most software face latency penalties due to static threading

Opportunity: Exploit trade-offs among threading models at run-time

Contributions

μSuite: Benchmark suite of OLDI services composed of microservices [1]

↓

Taxonomy of threading models: Implications of threading designs [2]

↓

μTune: Load adaptation s/m to tune threading models & improve tails [2]

↓

Achieve 1.9x tail latency speedup over state-of-the-art adaptations [2]

[1] A. Sriraman, T.F. Wenisch. μSuite: A Benchmark Suite for Microservices. International Symposium on Workload Characterization **(IISWC) 2018**.

[2] A. Sriraman, T.F. Wenisch. μTune: Auto-Tuned Threading for OLDI Microservices Operating Systems Design and Implementation **(OSDI) 2018**.

M CSE

µTune

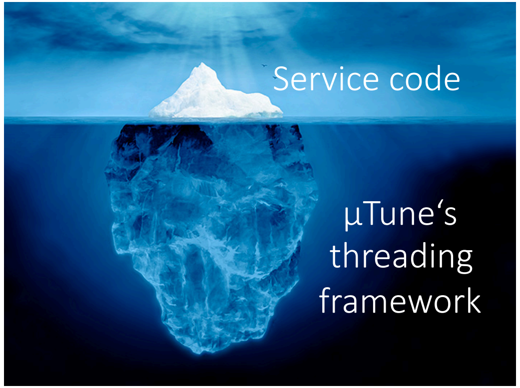
- Load adaptation: Vary threading model & pool size at run-time
- Abstract threading model boiler-plate code from RPC code

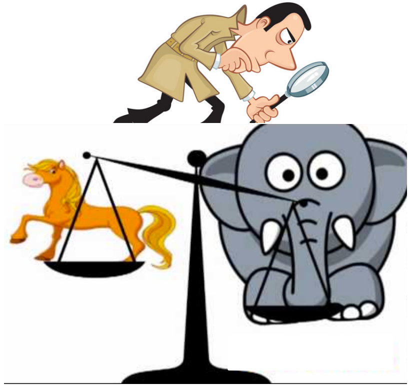
App layer	Microservice functionality: ProcessReq(), InvokeLeaf(), FinalizeResp()
µTune	µTune automatic load adaptation system
Network layer	RPC layer

Simple interface: Developer defines only three functions

µTune: Goals & Challenges

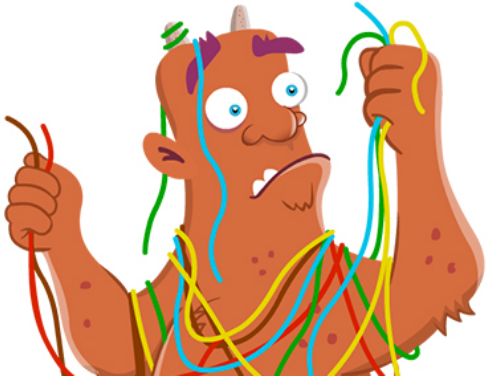
Simple interface





Quick load change detection

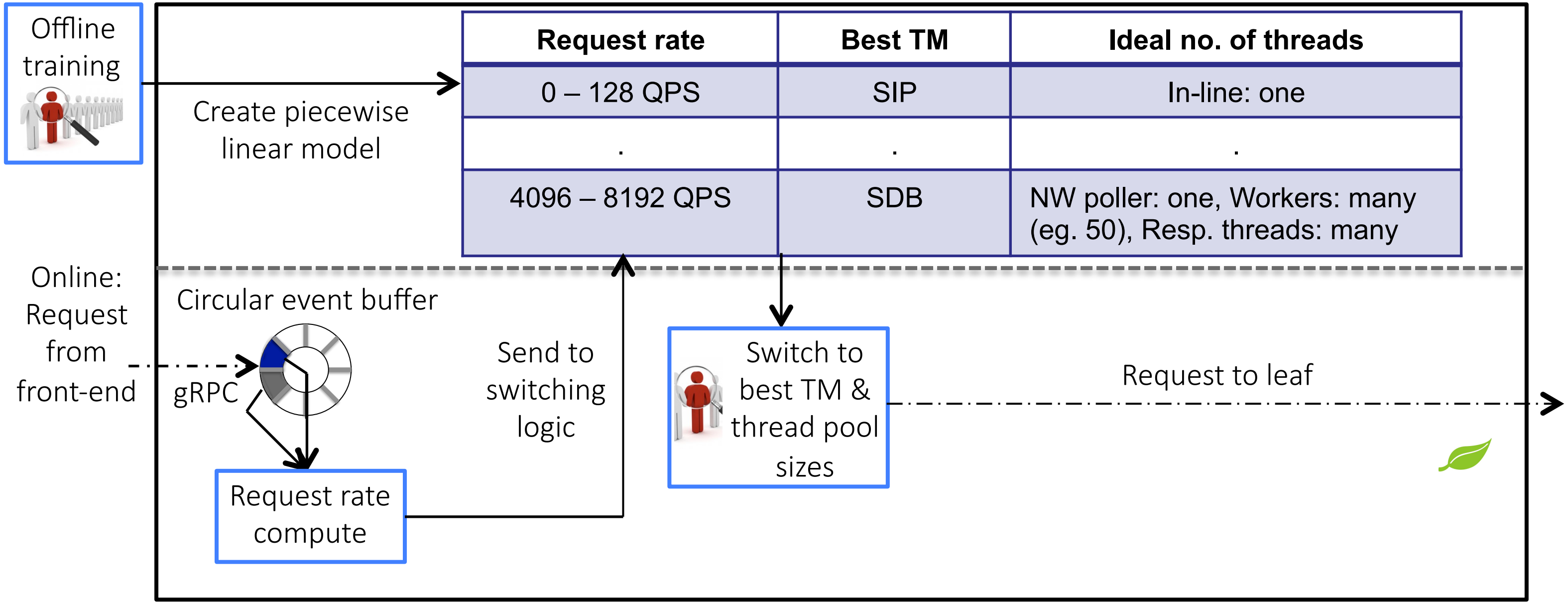
Fast threading model switches



Scale thread pools

µTune System Design: Auto-Tuner

- Dynamically picks threading model & pool sizes based on load

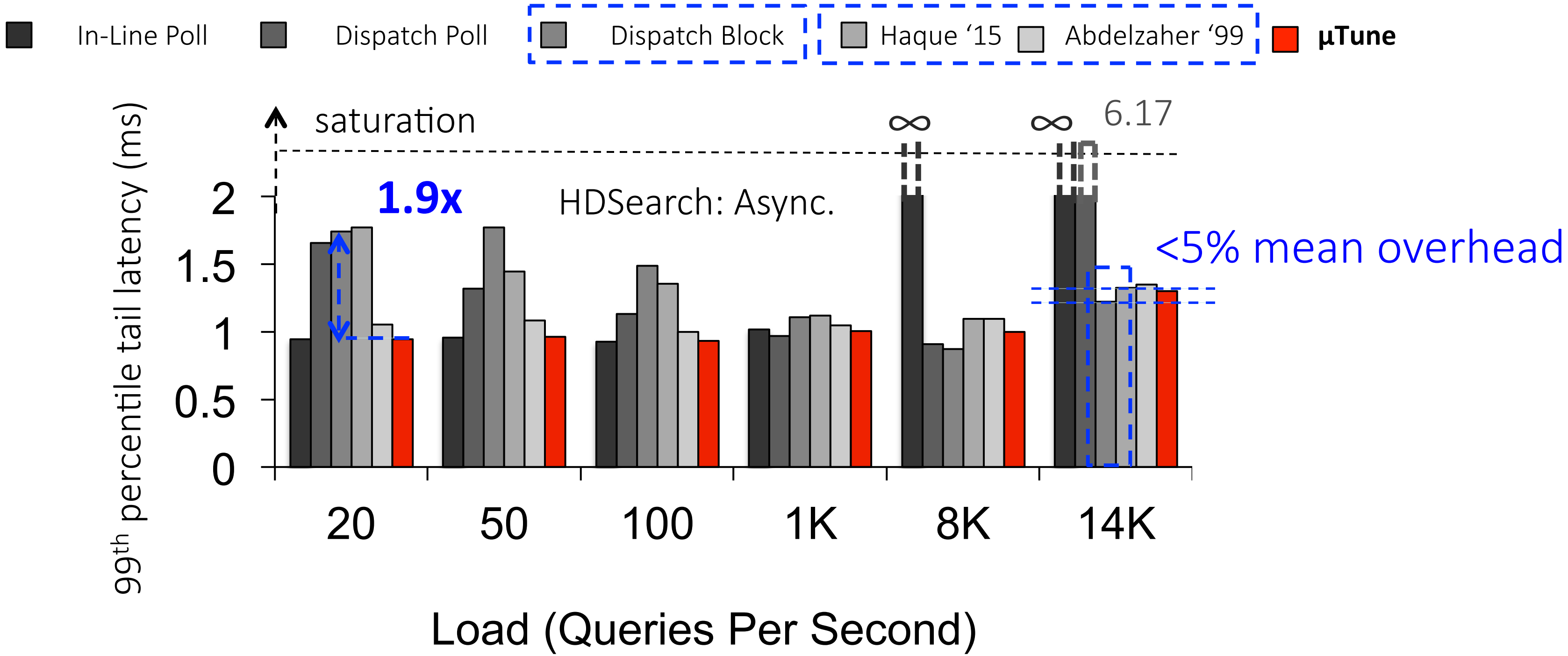


Request rate	Best TM	Ideal no. of threads
0 – 128 QPS	SIP	In-line: one
.	.	.
4096 – 8192 QPS	SDB	NW poller: one, Workers: many (eg. 50), Resp. threads: many

Experimental Setup

- µSuite: Three service tiers:
  - Load generator, a mid-tier, 4 or 16 leaf microservers
- State-of-the-art load generation mechanisms [Zhang ‘16]:
  - Closed-loop: Saturation throughput
  - Open-loop (arrivals from exponential distribution): Latency
- Study µTune’s adaptation in two load scenarios:
  - Steady-state
  - Transients

Evaluation: µTune's Load Adaptation



Converges to best threading model & pool sizes to improve tails by up to **1.9x**

Conclusion

- µSuite – benchmark suite of microservices
  - µSuite can facilitate future research
- Taxonomy of threading models
  - Optimal threading model is load dependent
- µTune – threading model framework + load adaptation system

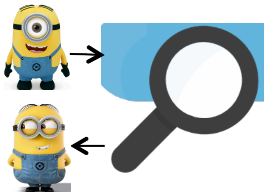
A. Sriraman, T.F. Wenisch. µTune: Auto-Tuned Threading for OLDI Microservices Operating Systems Design and Implementation **(OSDI) 2018.**

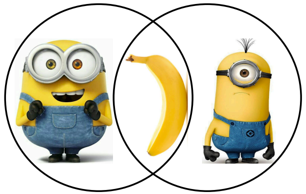
A. Sriraman, T.F. Wenisch. µSuite: A Benchmark Suite for Microservices. International Symposium on Workload Characterization **(IISWC) 2018**.

M CSE

µSuite & µTune: Auto-Tuned Threading for OLDI Microservices

Akshitha Sriraman, Thomas F. Wenisch



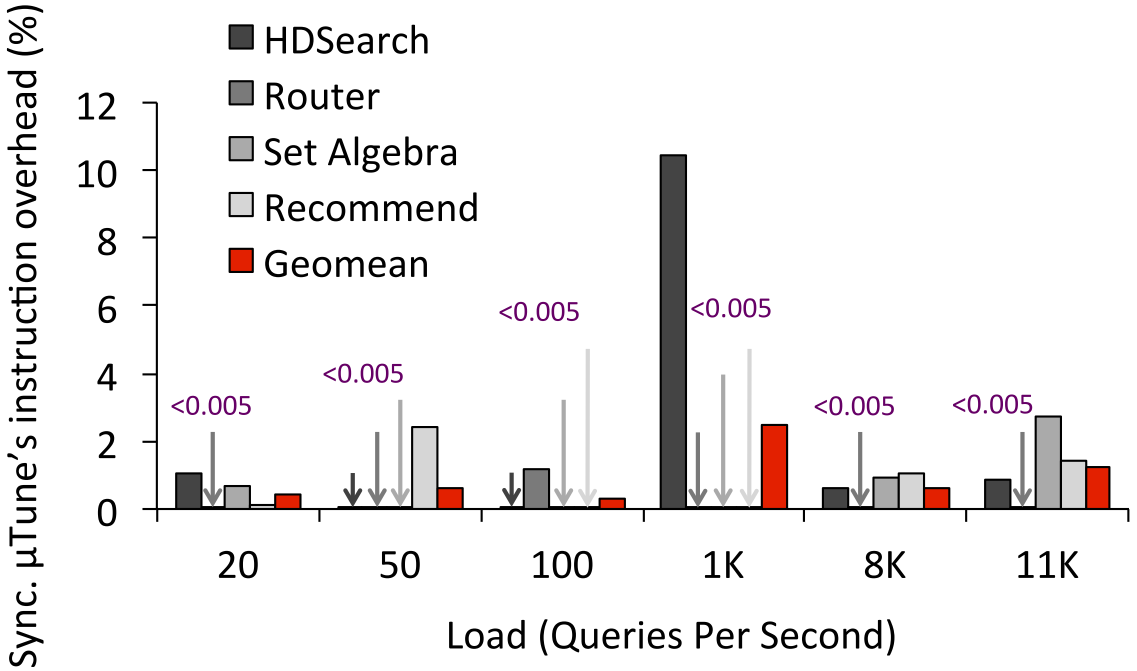


https://github.com/wenischlab/MicroSuite

https://github.com/wenischlab/MicroTune

BACKUP SLIDES

Instruction Overhead



Sync. µTune's instruction overhead for steady-state load: <5% mean overhead

Comparison With State-of-the-Art

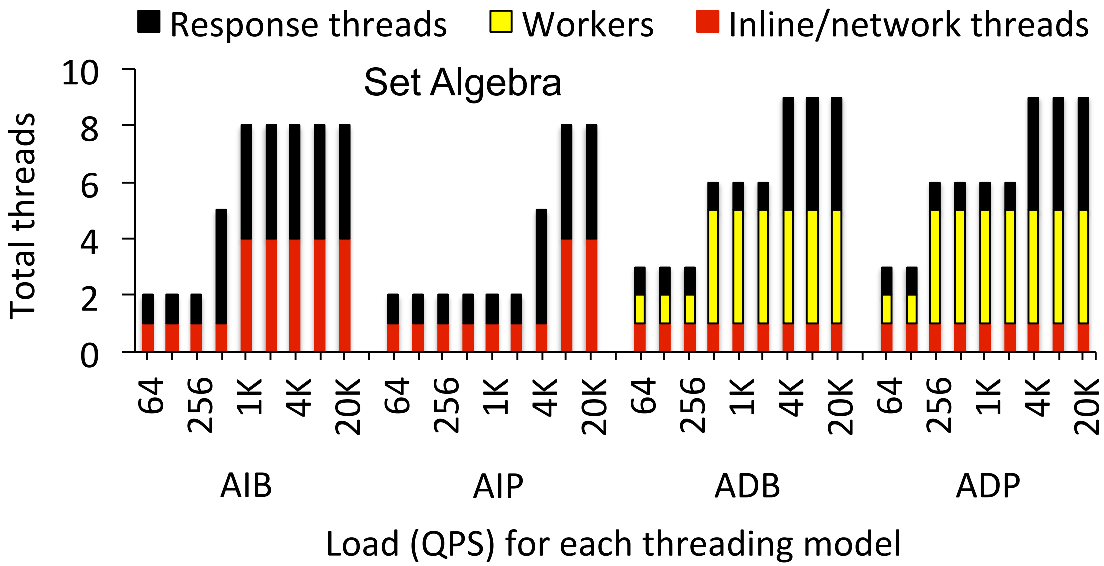
- Few-to-Many Parallelism:
  - Adapting thread pool sizes
- Langendoen et al.
  - Adapting poll vs. block
- Abdelzaher et al.
  - Time window-based load detection

Load Transients

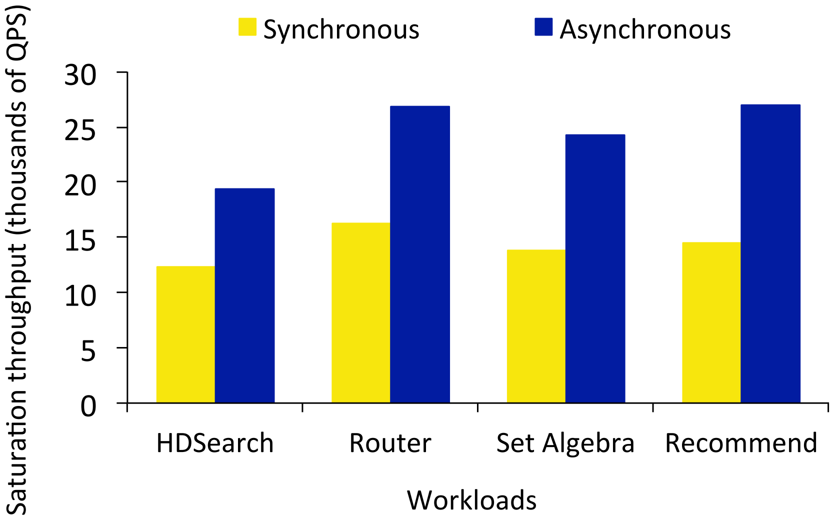
	Synchronous	100 QPS (0 - 30s)	8K QPS (30s - 31s)	100 QPS (31 - 61s)	Asynchronous	100 QPS (0 - 30s)	13K QPS (30s - 31s)	100 QPS (31 - 61s)
HDSearch	SIP	0.99	>1s	>1s	AIP	0.95	>1s	>1s
	SDB	1.49	1.07	1.40	ADB	1.48	1.10	1.40
	FM	1.35	13.00	1.32	FM	1.28	4.73	1.33
	IPI	1.59	1.10	1.50	IPI	NA	NA	NA
	TBD	1.03	8.69	1.02	TBD	1.06	2.63	1.08
	μTune	1.01	1.09	0.99	*μTune*	0.98	1.13	0.96
Router	SIP	1.10	>1s	>1s	AIP	1.01	>1s	>1s
	SDB	1.31	0.83	1.36	ADB	1.35	1.13	1.31
	FM	1.33	9.40	1.40	FM	1.30	12.95	1.30
	IPI	1.4	1.10	1.38	IPI	NA	NA	NA
	TBD	1.13	4.51	1.11	TBD	1.03	6.24	1.01
	μTune	1.12	0.88	1.13	*μTune*	0.99	1.02	0.98
Set Algebra	SIP	0.95	>1s	>1s	AIP	1.04	>1s	>1s
	SDB	1.30	0.92	1.32	ADB	1.26	0.99	1.23
	FM	1.30	12.00	1.25	FM	1.28	4.14	1.27
	IPI	1.20	0.94	1.12	IPI	NA	NA	NA
	TBD	1.00	8.45	1.03	TBD	1.09	6.62	1.1
	μTune	0.97	0.92	1.03	*μTune*	1.06	1.1	1.06
Recommend	SIP	1.00	>1s	>1s	AIP	1.03	>1s	>1s
	SDB	1.26	0.96	1.22	ADB	1.37	1.30	1.32
	FM	1.23	>1s	>1s	FM	1.28	8.61	1.20
	IPI	1.13	1.02	1.13	IPI	NA	NA	NA
	TBD	1.02	4.96	1.03	TBD	1.06	6.00	1.07
	μTune	1.00	1.00	1.00	*μTune*	1.06	1.39	1.04

M CSE

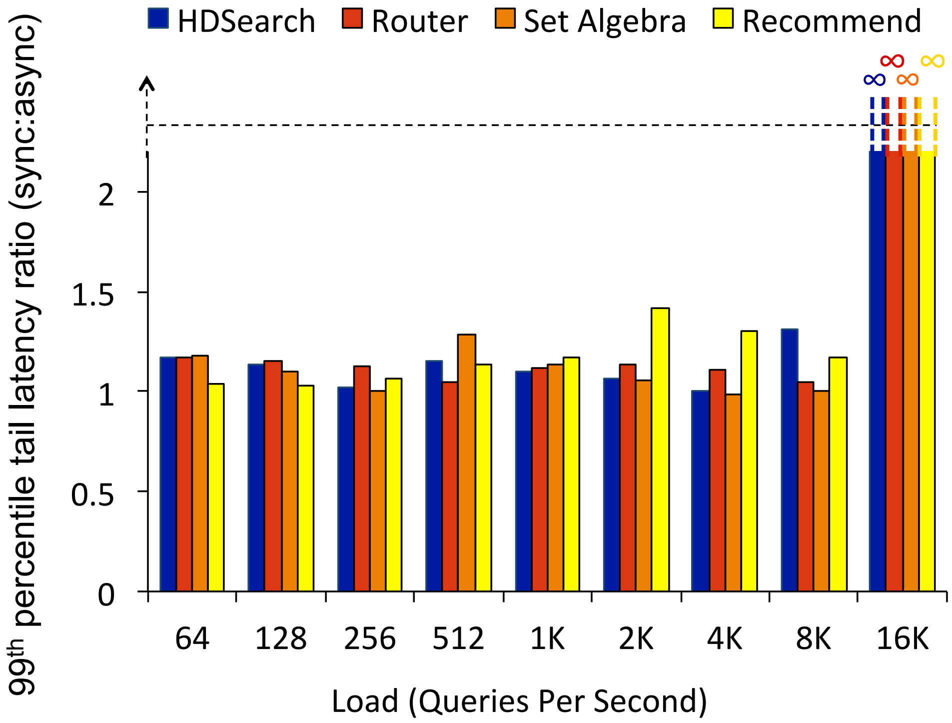
Thread Pool Sizes



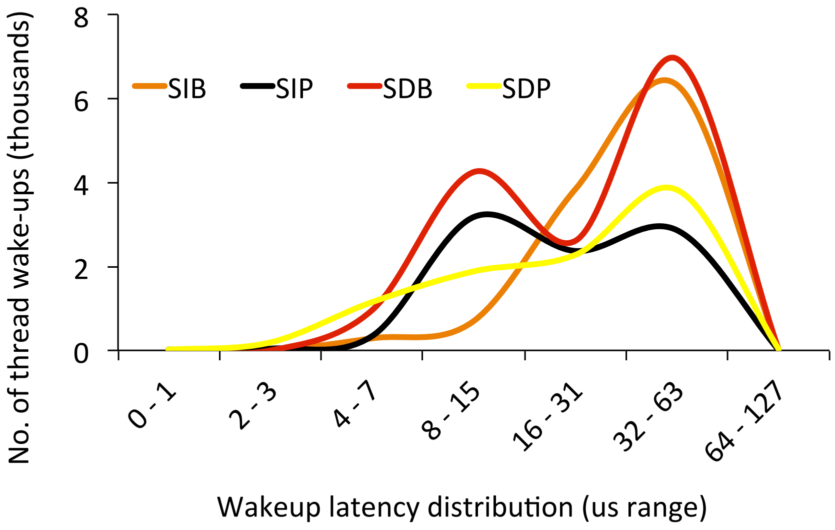
Sync vs. Async: Saturation Throughput



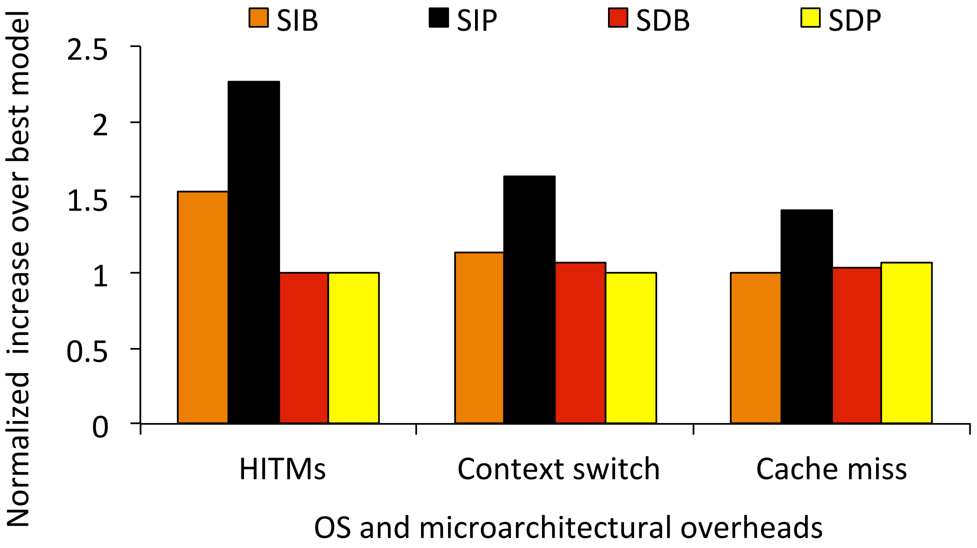
Sync. Vs. Async.: Tail Latency



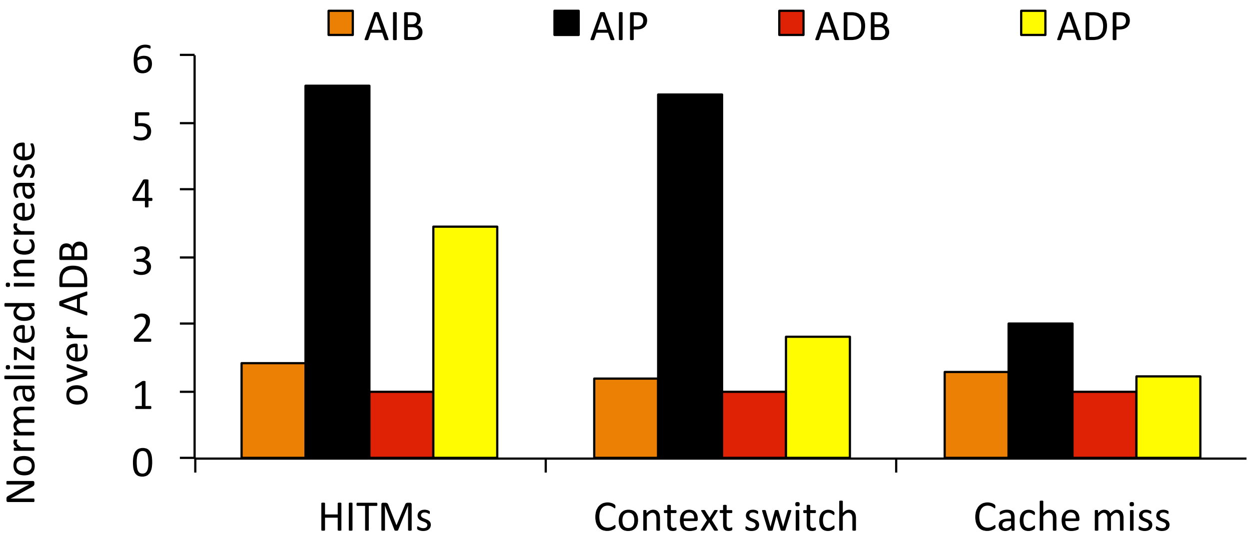
Thread Wakeup Delays



OS & Microarchitectural Effects



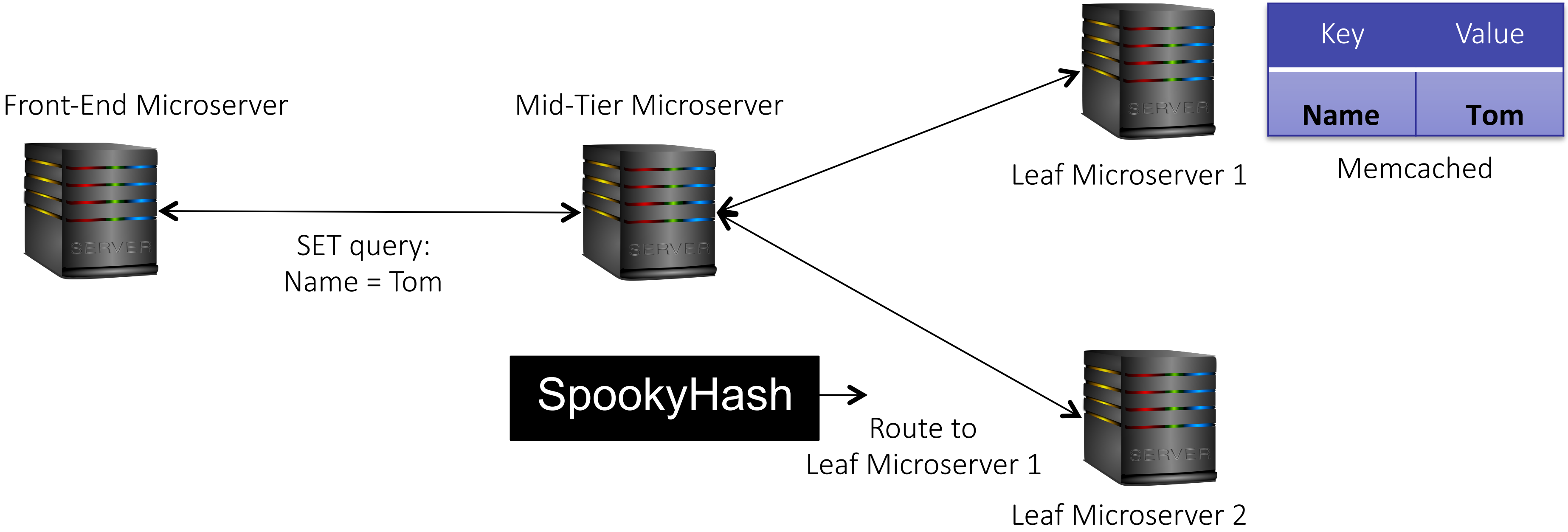
Async. OS & Microarchitectural Effects



Router

- Routes key-value stores to Memcached
- Replication-based protocol routing for fault-tolerance
  - SETs go to multiple leaves
  - GETs go to a single leaf
- More scalable – a subset of leaves are contacted
  - May face more threading overheads due to GET/SET asymmetry

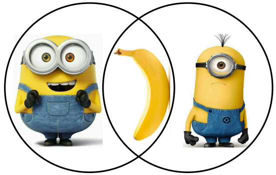
Router: Operation



Making Router a Benchmark

- Query set:
  - Set of {key, value} pairs from a Twitter data set [Ferdman '12]
  - GET:SET distributions mimic YCSB's workload A (50:50)

Set Algebra

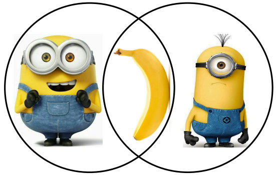


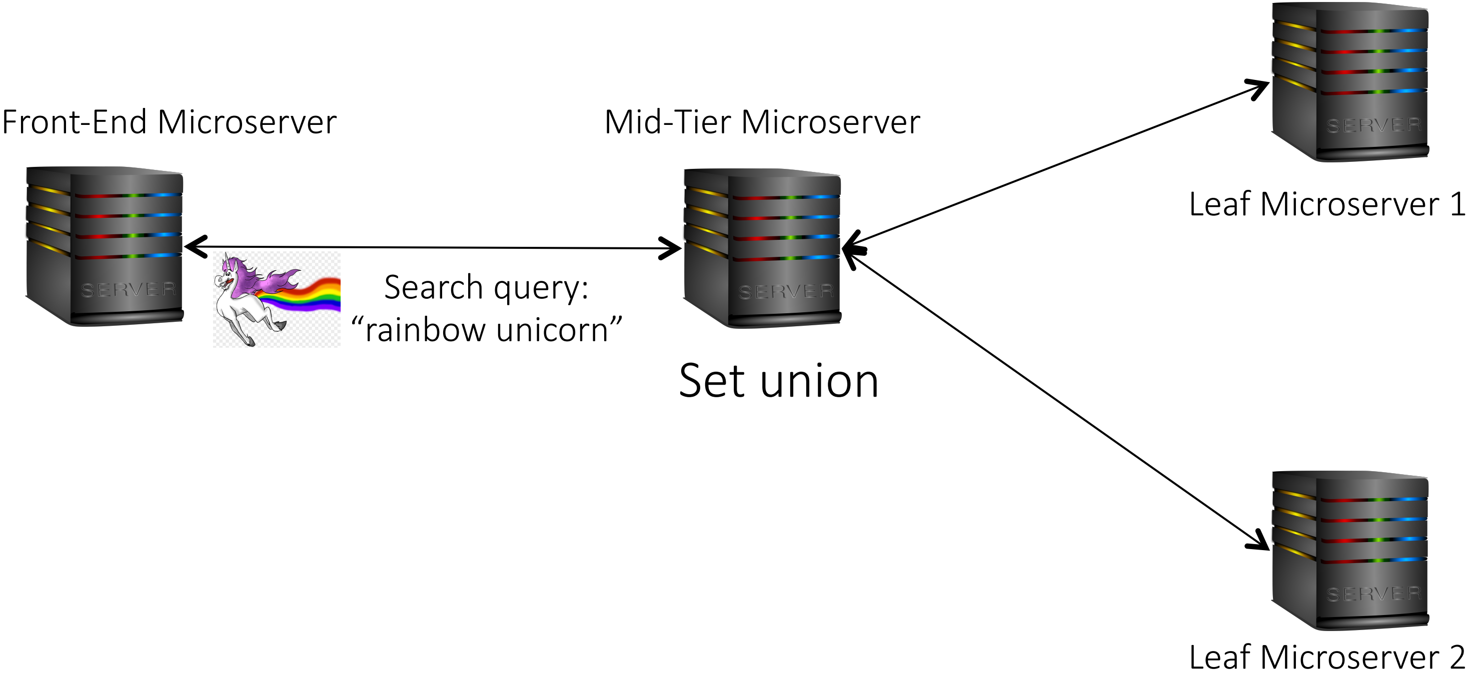
- Document retrieval for web search
  - Set intersections on posting lists
- Inverted index:
  - Map of term to all doc IDs containing term

ID	Term	Doc. IDs
1	Data	1, 2, 3, 4
3	Butterfly	1, 2, 6, 7
3	Rainbow	2, 4, 5
4	Unicorn	2

- Large variability in leaves' compute
  - Helps study overheads with short & long requests

Set Algebra: Operation





Term	Doc ID
Butterfly	1 3
Rainbow	**3**
Unicorn	1 **3**

Inverted index

Set intersection

Term	Doc ID
Butterfly	2 8
Unicorn	**4**
Rainbow	**4** 6

Inverted index

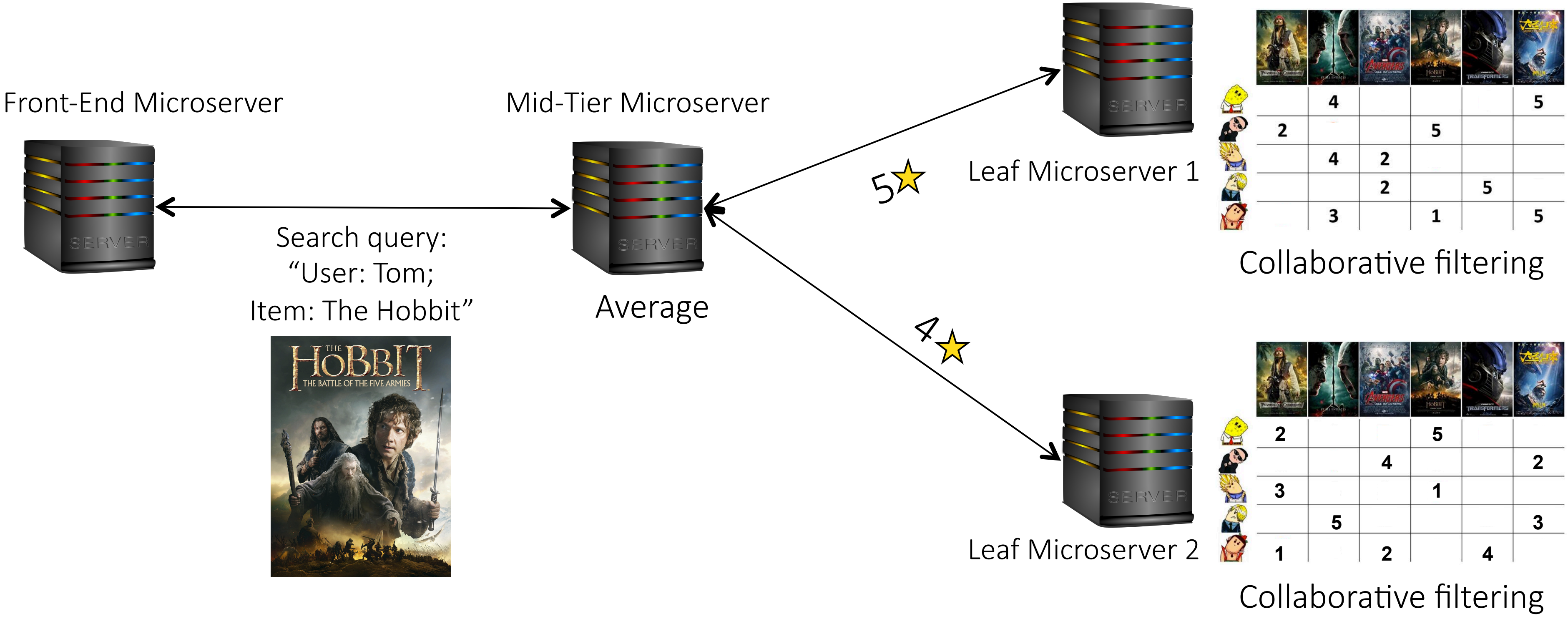
Making Set Algebra a Benchmark

- Data set: inverted index of documents
  - 4.3M documents from Wikipedia: 10 GB
  - Prepared sharded inverted index corpus
  - Test set: Synthetically created using Wikipedia's word probabilities
  - Query: uniformly randomly selected set of <= 10 terms

Recommend

- Predicts user ratings for specific items
  - Uses collaborative filtering
- Mid-Tier does minimal work on the request path
  - Helps study unmasked OS and network effects

Recommend: Operation



	4				5
2			5		
	4	2			
		2		5	
	3		1		5

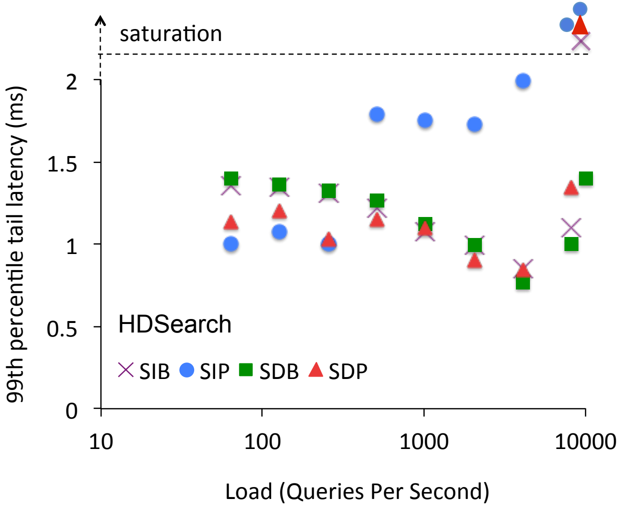
2			5		
		4			2
3			1		
	5				3
1		2		4	

Making Recommend a Benchmark

- Dataset: {user, item, rating} tuples
  - MovieLens movie recommendation data set [Harper '15]
  - Prepared sharded sparse user-item rating matrix
  - Test set of {user, item} query pairs from MovieLens [Harper '15]

Characterizing the Threading Taxonomy

- SIP has lowest latency at low load
  - Avoid two kinds of thread wakeups
- SDP is best at intermediate loads
  - Avoids in-line polling thread contention
- SDB enables highest load
  - Single network thread, many workers



QPS	64	128	256	512	1024	2048	4096	8192	10K
SIB	1.4	1.3	1.3	**1**	**1**	**1**	1.1	1.1	∞
SIP	**1**	**1**	**1**	1.6	1.6	1.9	2.6	∞	∞
SDB	1.4	1.3	1.3	1.1	1.1	1.1	**1**	**1**	**1**
SDP	1.2	1.1	**1**	**1**	**1**	**1**	1.1	1.4	∞

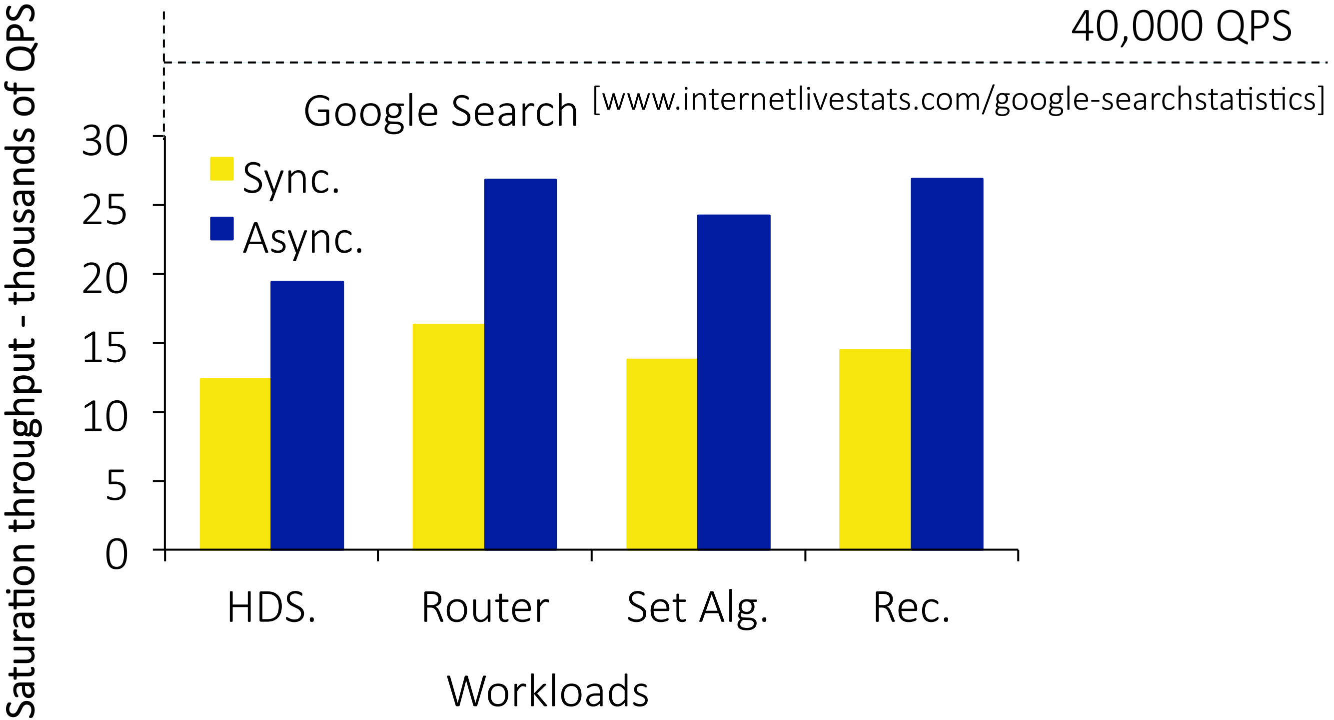
No single threading model is optimal at all loads

Comparison With State-of-the-Art Adaptation

- Few-to-Many (FM) parallelism [Haque '15]
  - Uses offline interval table to select thread pool sizes
- Integrating Polling and Interrupts (IPI) [Langendoen '96]
  - Polls when threads are blocked
  - Uses interrupts when blocked thread returns
- Time-window Based Detection (TBD) [Abdelzaher '99]
  - Track request arrivals in fixed observation time windows

µTune should outperform as it considers both threading models & pool sizes

Sync. Vs. Async.: Saturation Throughput



Async. models are more performant although harder to program