InfiniStore: Elastic Serverless Cloud Storage

Jingyuan Zhang, Ao Wang, Xiaolong Ma, **Benjamin Carver**, Nicholas John Newman, Ali Anwar, Lukas Rupprecht, Vasily Tarasov, Dimitrios Skourtis, Feng Yan, Yue Cheng

GEORGE MASON UNIVERSITY
UNIVERSITY of VIRGINIA
UNIVERSITY OF HOUSTON
UNIVERSITY OF MINNESOTA
IBM Research

Storages for data-intensive applications

In-memory key-value (IMKV) stores







File stores





Object stores



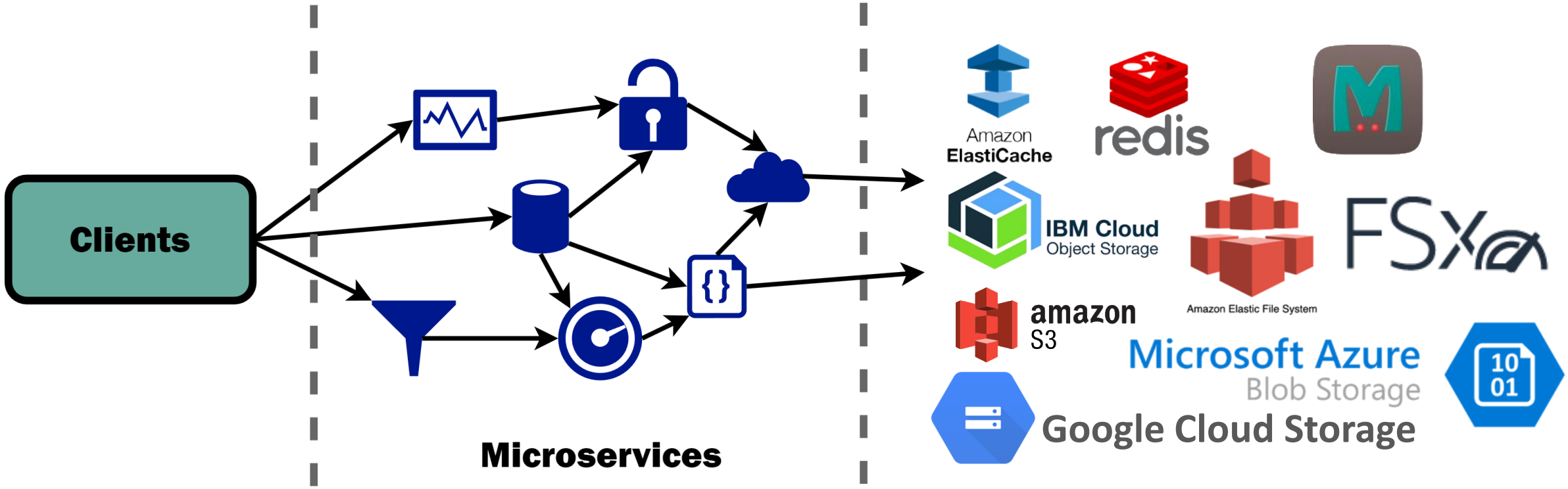




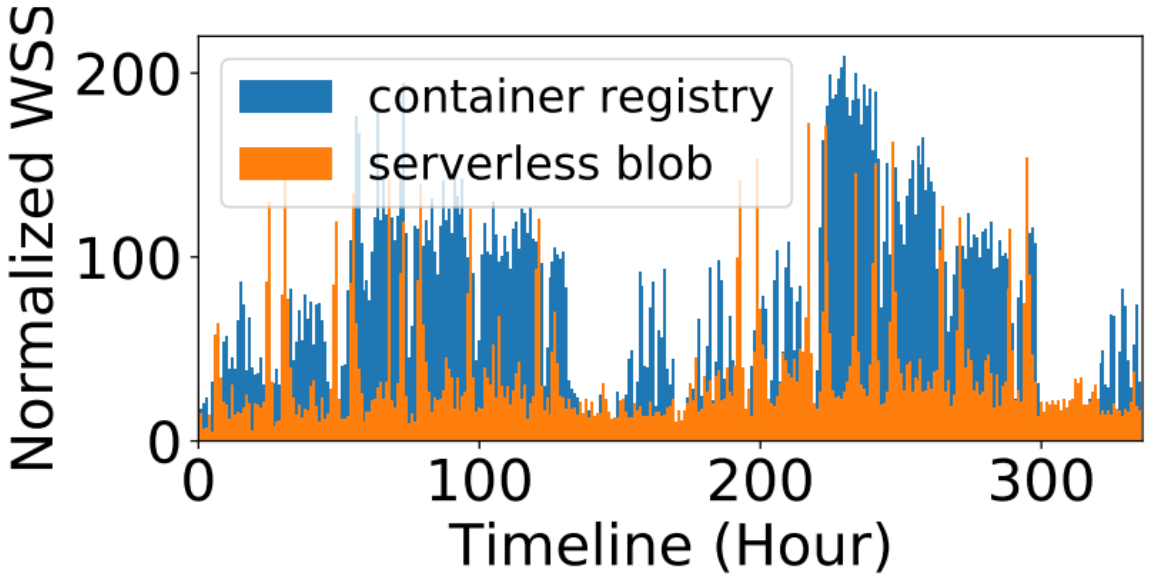


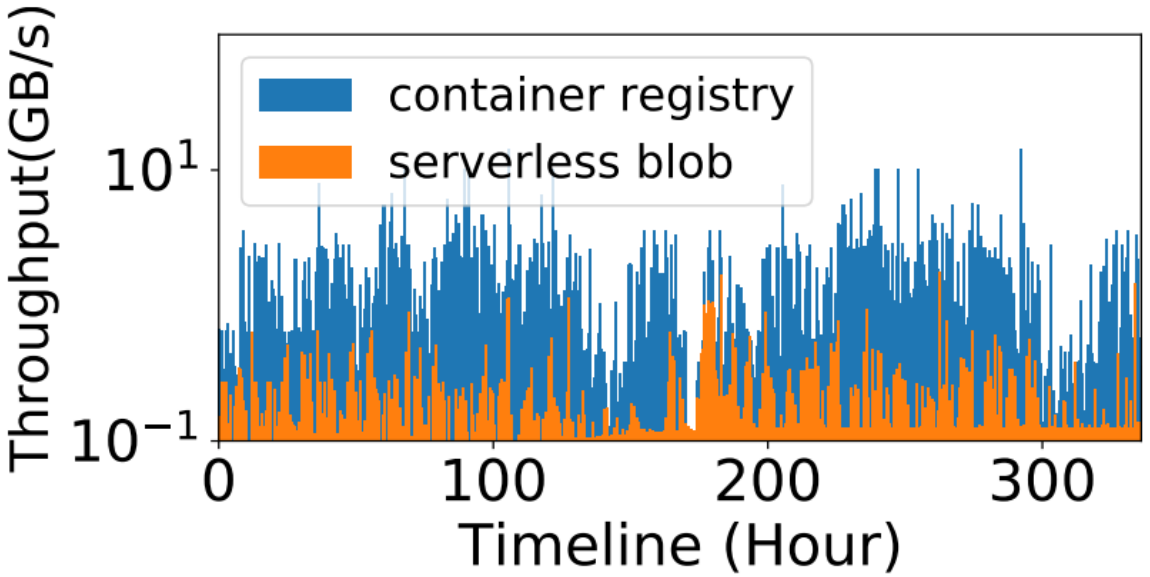


Heterogeneous I/O patterns



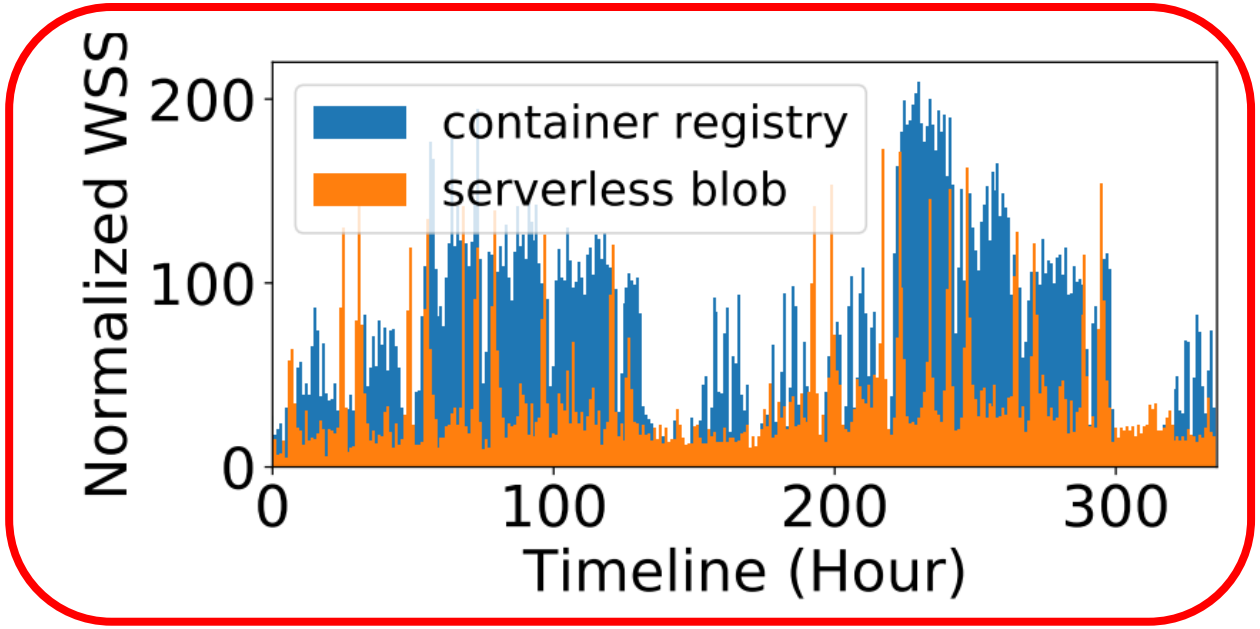
Frequently-changing working set size (WSS)

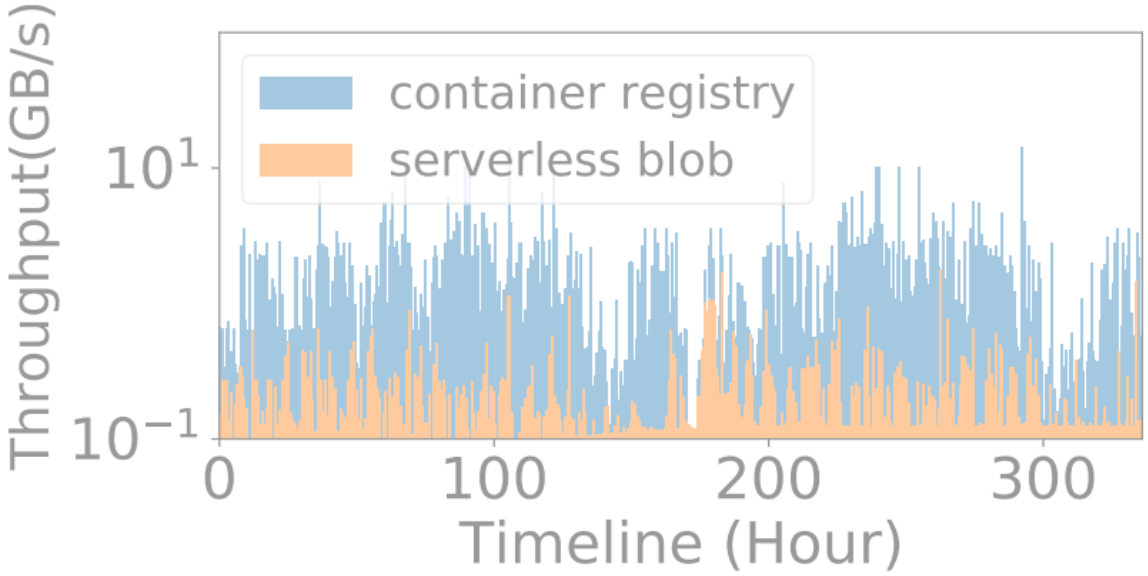




Frequently-changing working set size (WSS)

Maximum 1-hour WSS is 209x larger than the minimum.

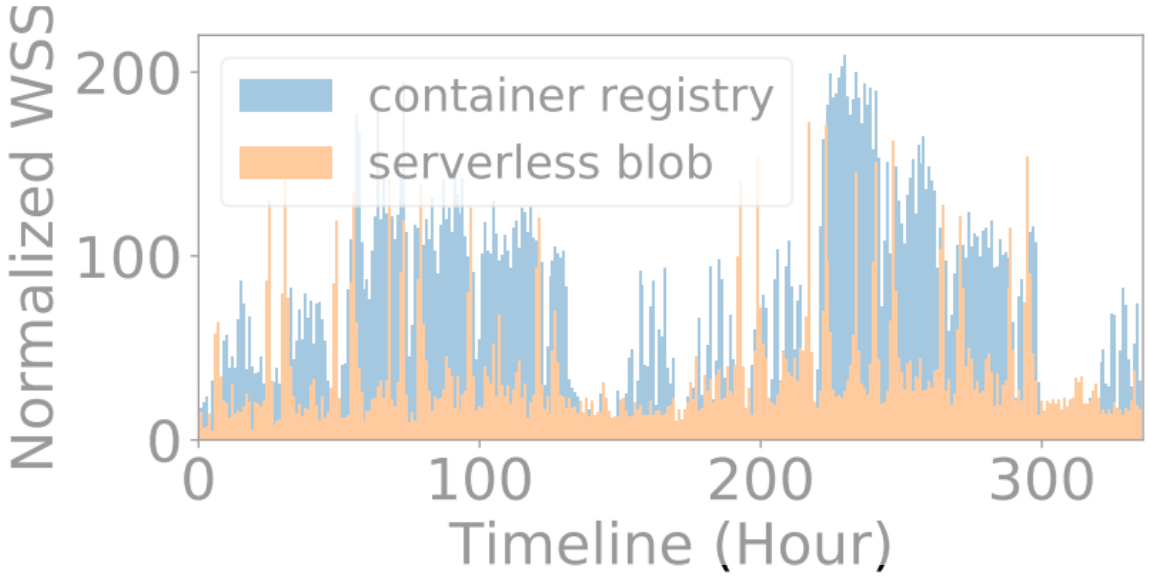


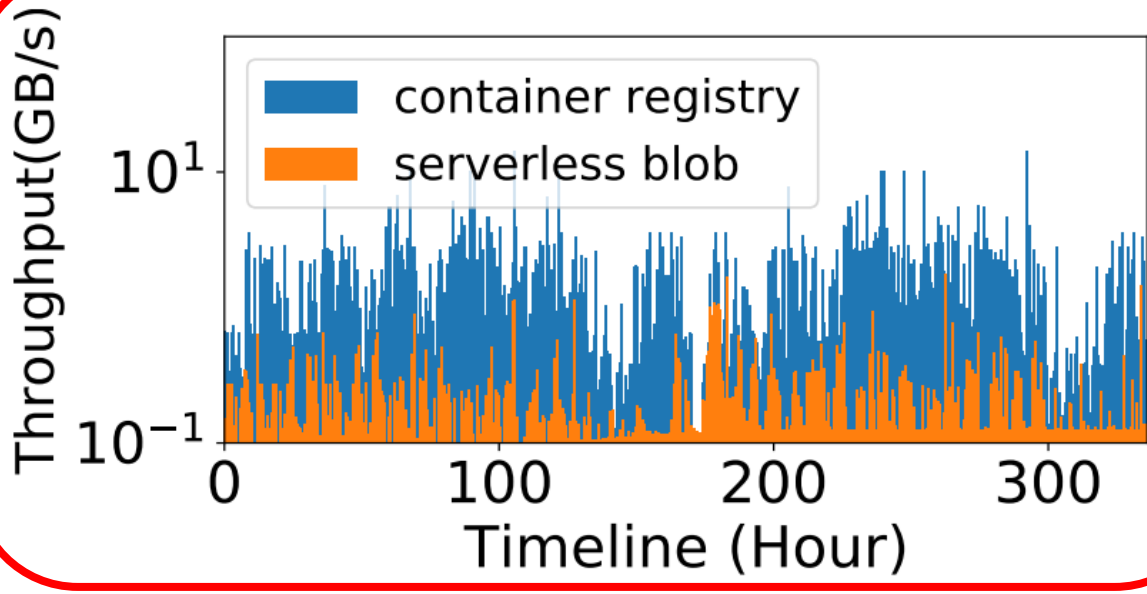


Frequently-changing working set size (WSS)

Maximum 1-hour WSS is 209x larger than the minimum.

Highly dynamic bandwidth provisioning.



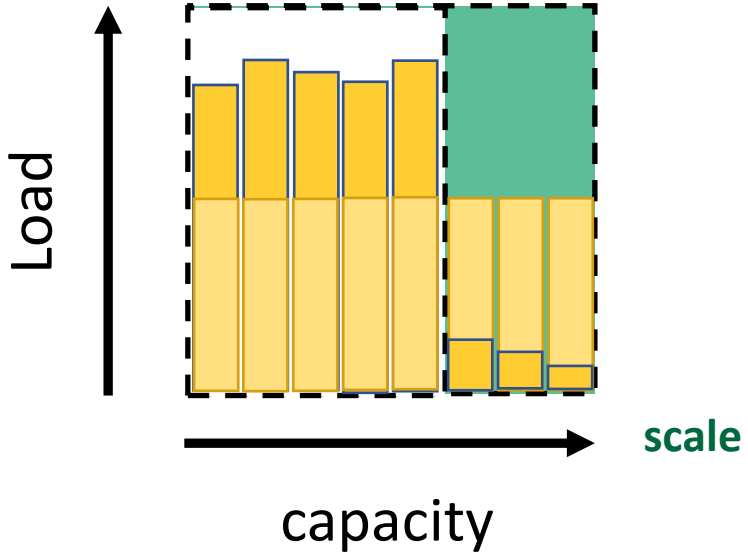


Efficiently scaling

Maximum 1-hour WSS is 209x larger than the minimum.

Highly **dynamic** bandwidth provisioning.

Repartitioning incurs **massive data movement**.

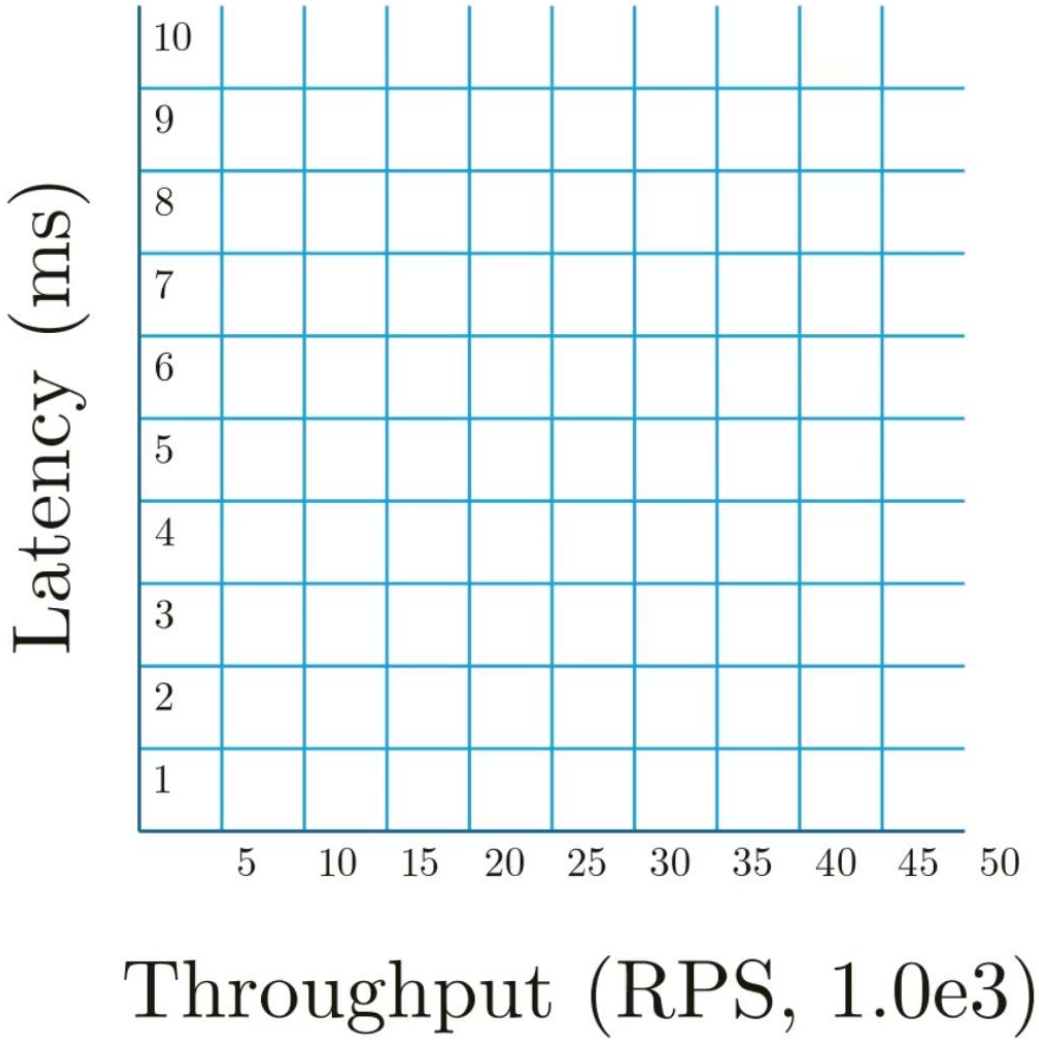


ElastiPut: Elastic Throughput

ElastiPut = Throughput (RPS) / Latency

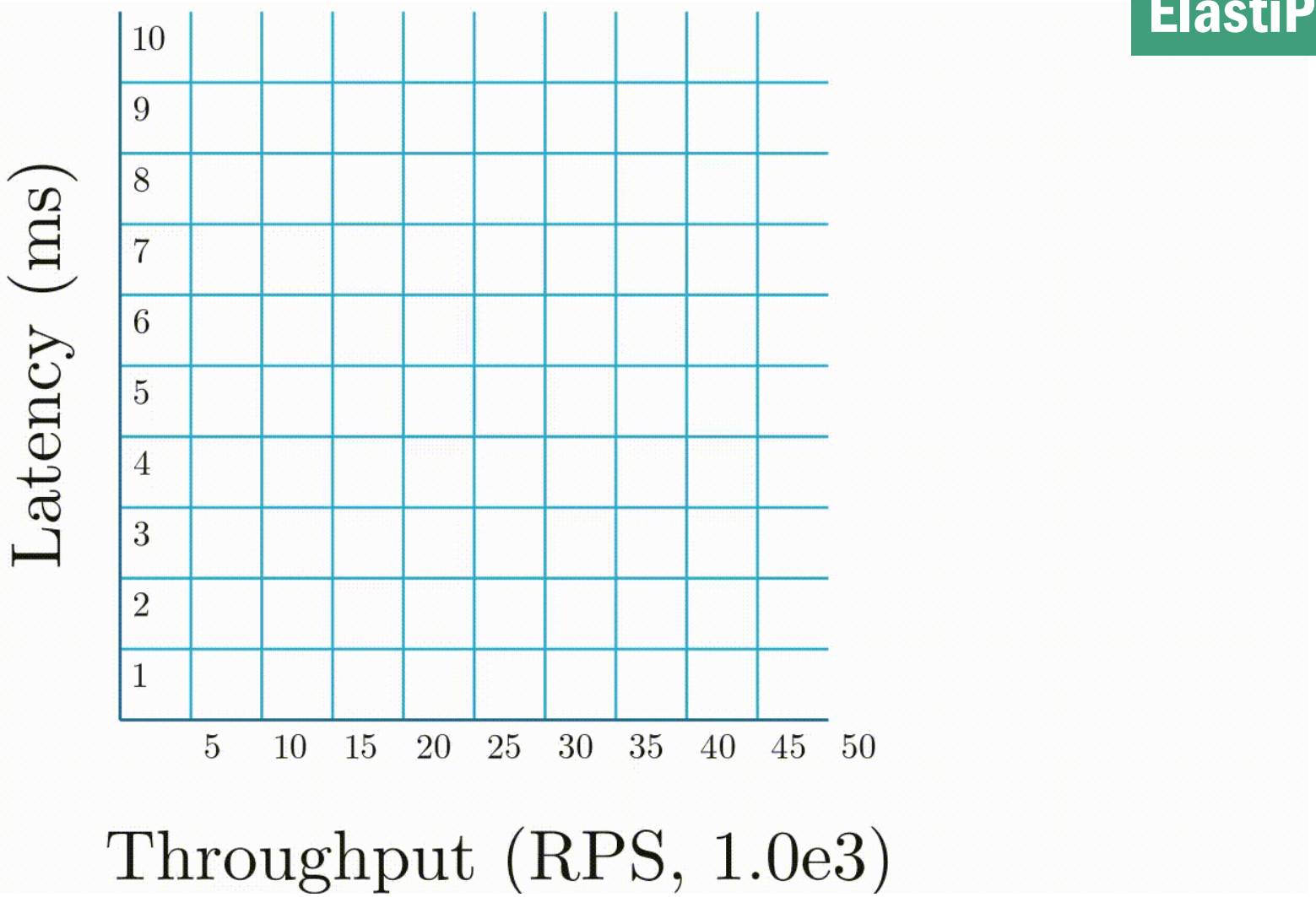
ElastiPut: Elastic Throughput

ElastiPut = Throughput (RPS) / Latency



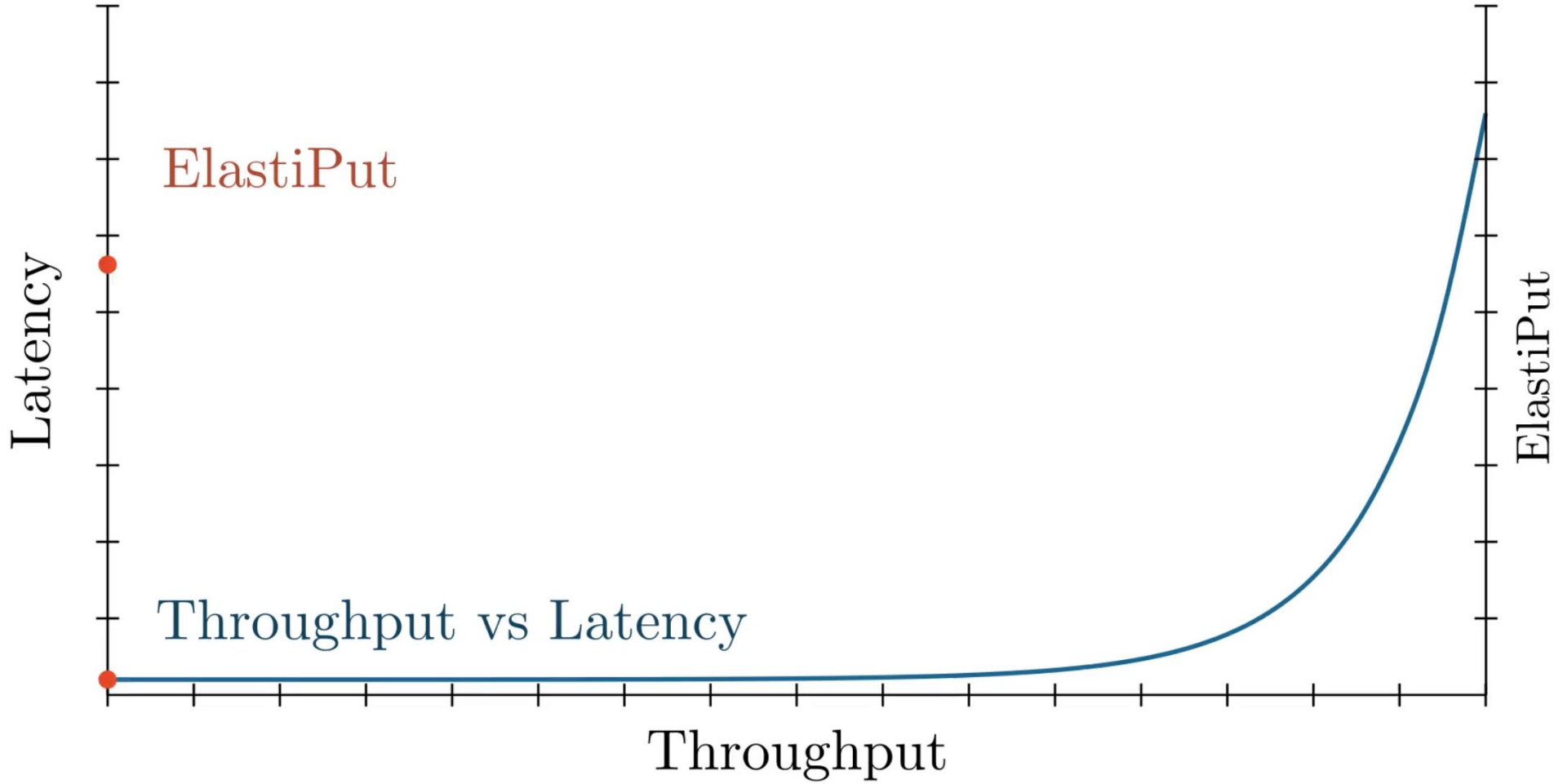
ElastiPut: Elastic Throughput

ElastiPut = Throughput (RPS) / Latency



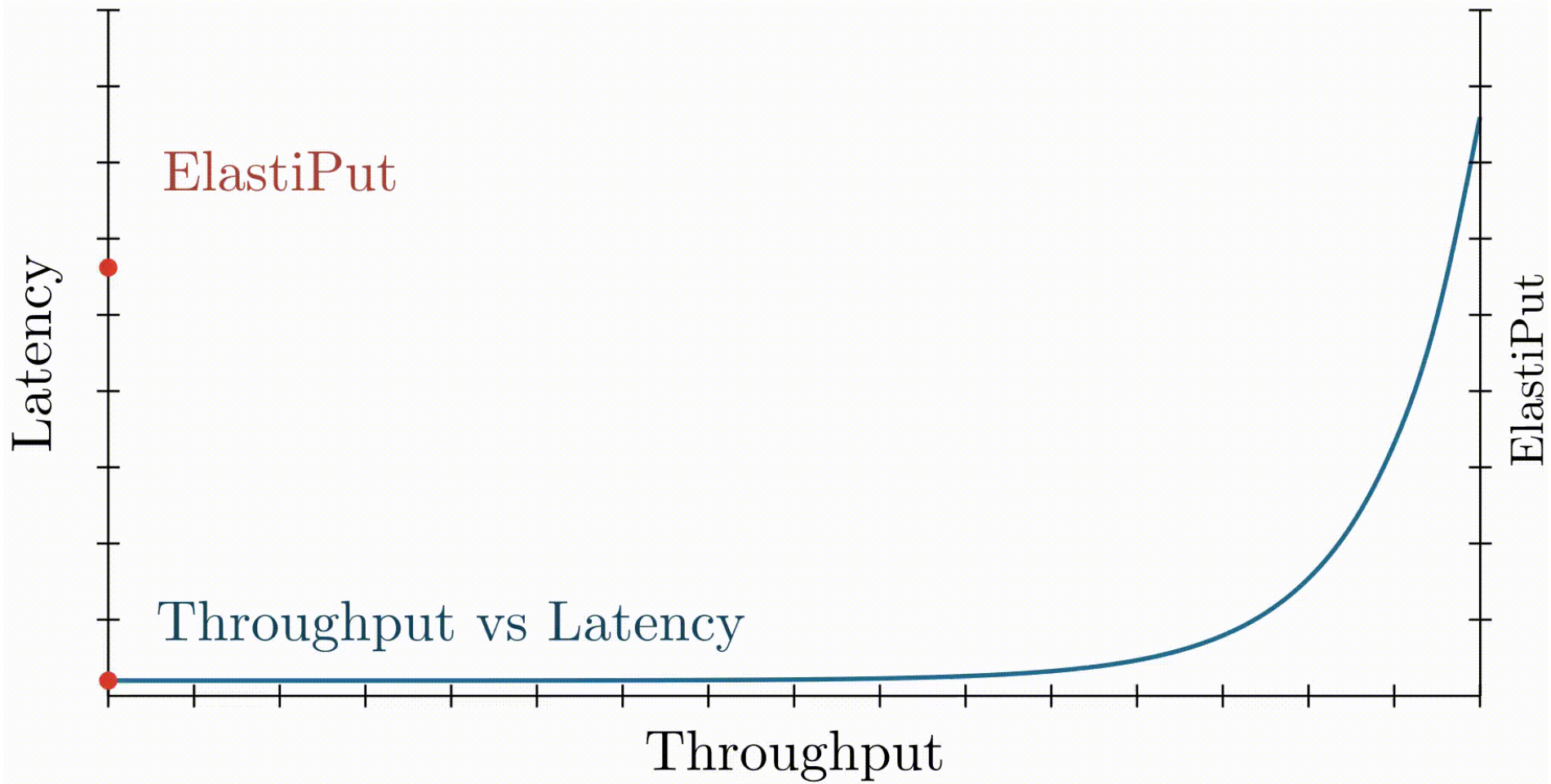
ElastiPut: Elastic Throughput

ElastiPut = Throughput (RPS) / Latency

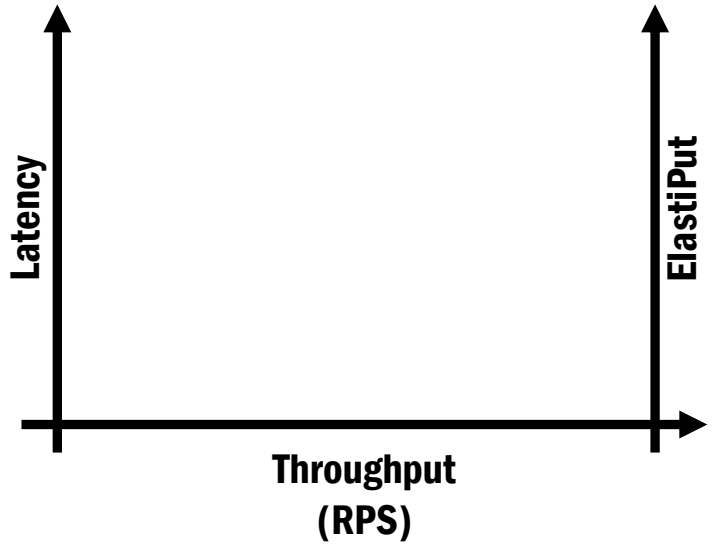


ElastiPut: Elastic Throughput

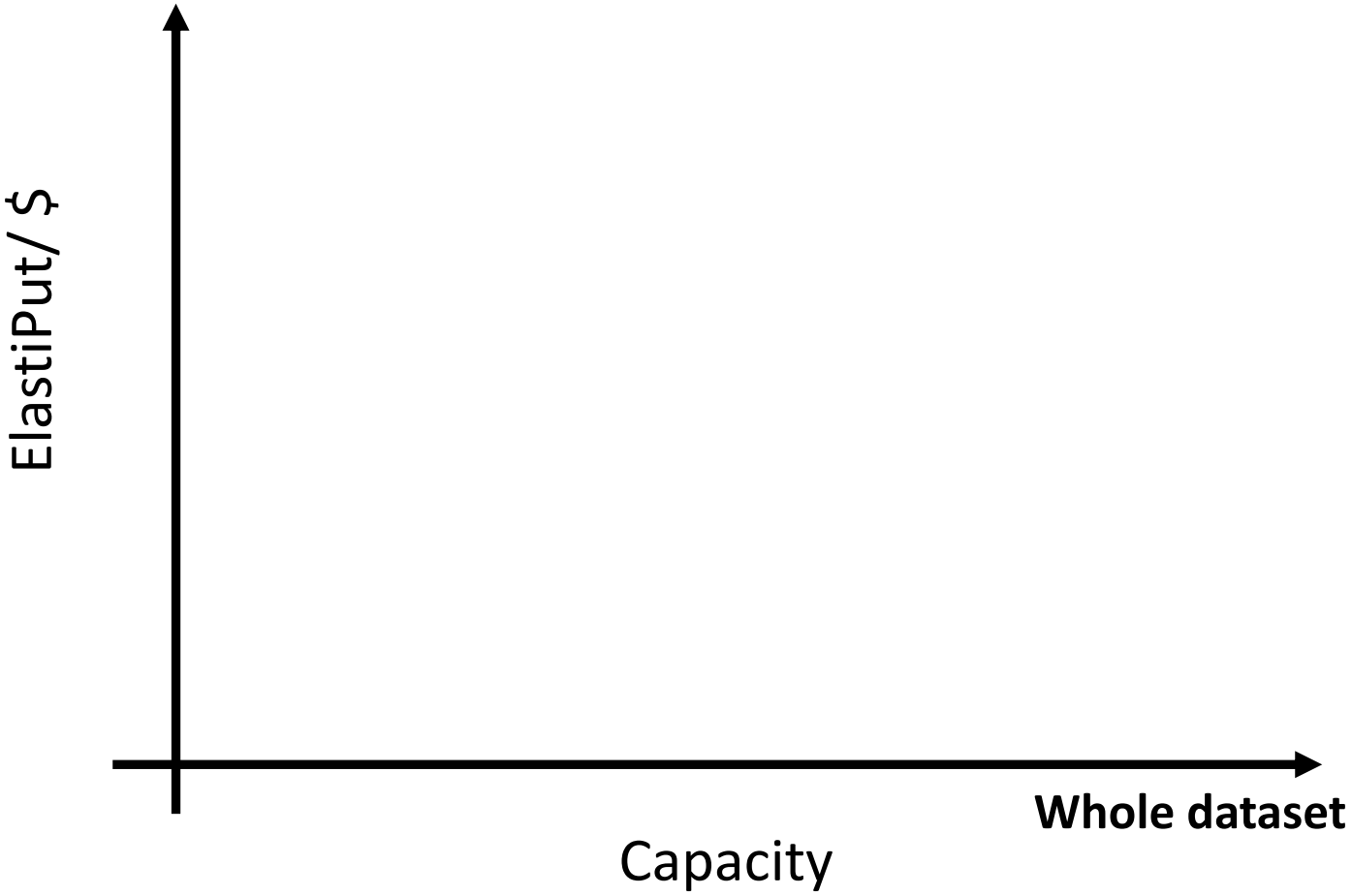
ElastiPut = Throughput (RPS) / Latency



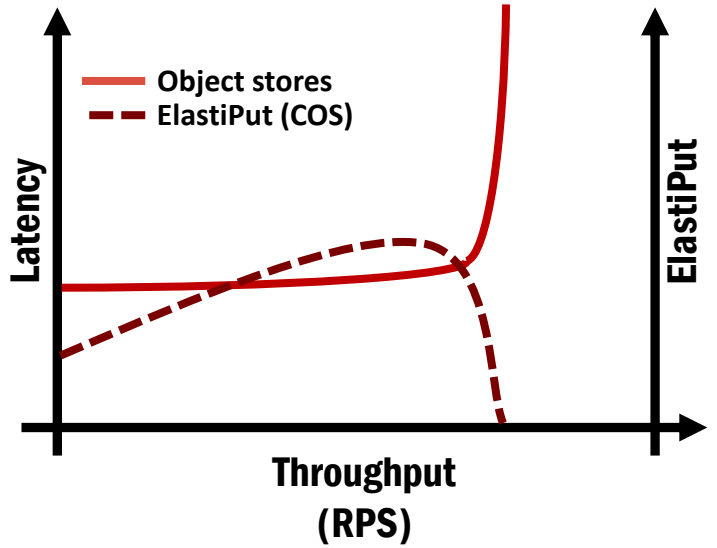
Cost-effectiveness of existing cloud storage services



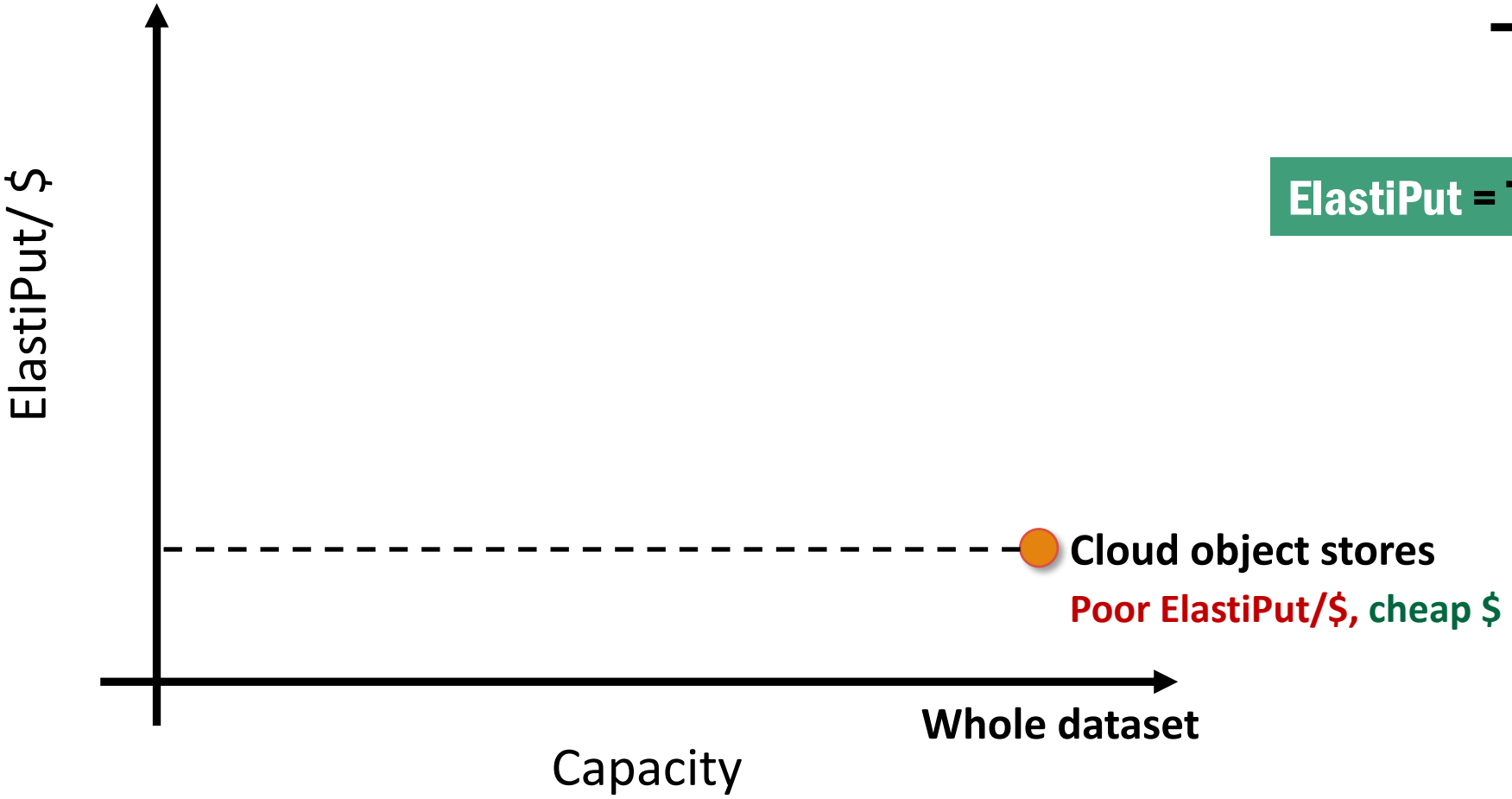
ElastiPut = Throughput (RPS) / Latency



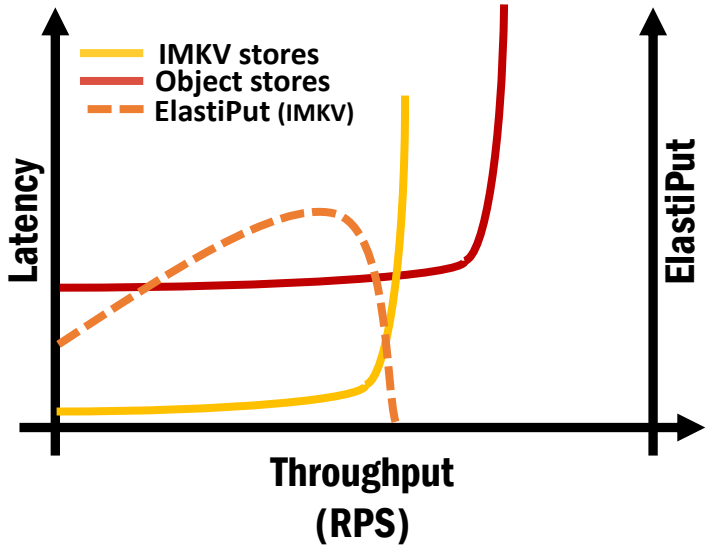
Cost-effectiveness of existing cloud storage services



ElastiPut = Throughput (RPS) / Latency



Cost-effectiveness of existing cloud storage services



ElastiPut = Throughput (RPS) / Latency

ElastiPut/ $

IMKV stores

Good ElastiPut/$, expensive $$$

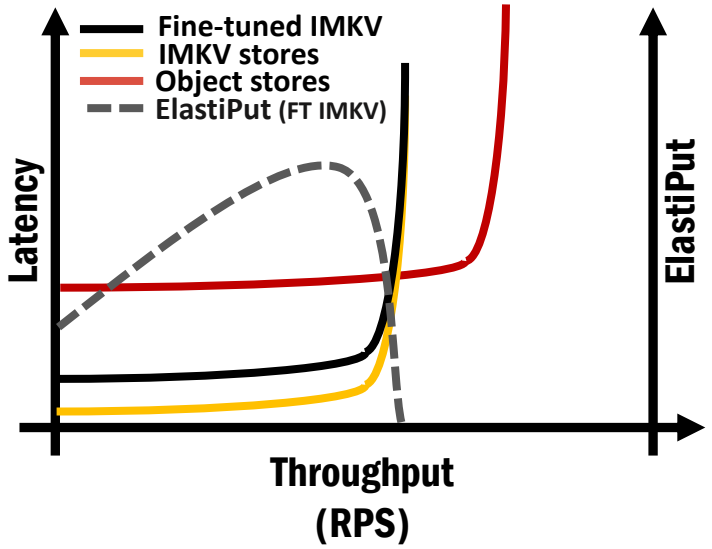
Cloud object stores

Poor ElastiPut/$, cheap $

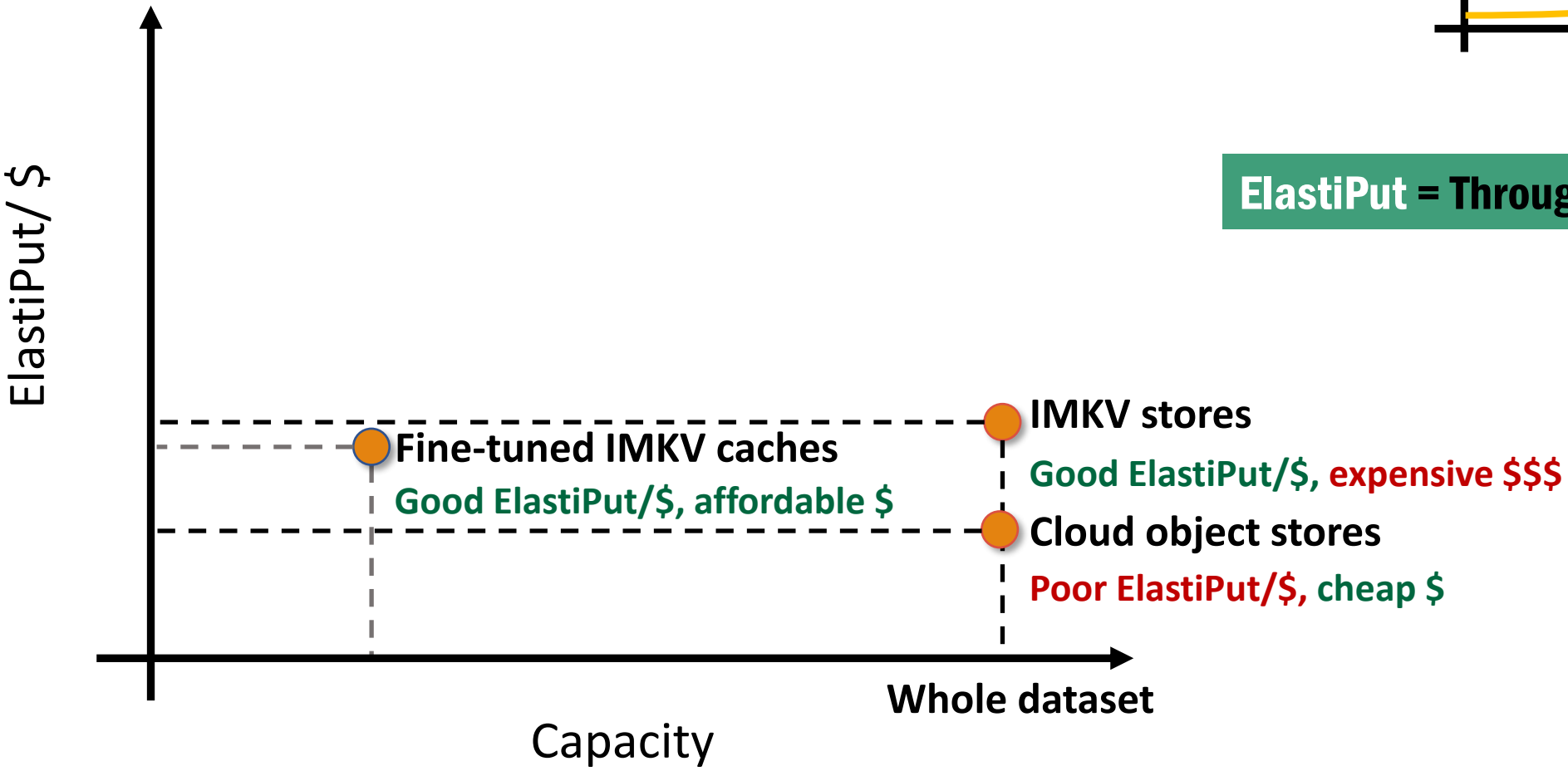
Whole dataset

Capacity

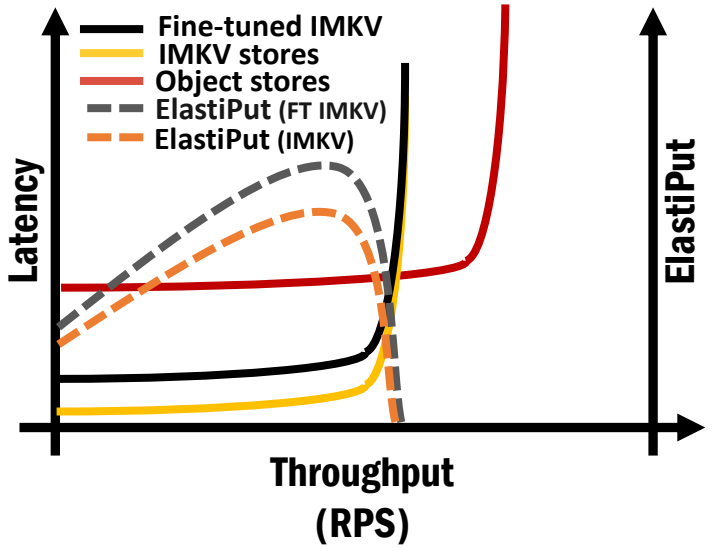
Cost-effectiveness of existing cloud storage services



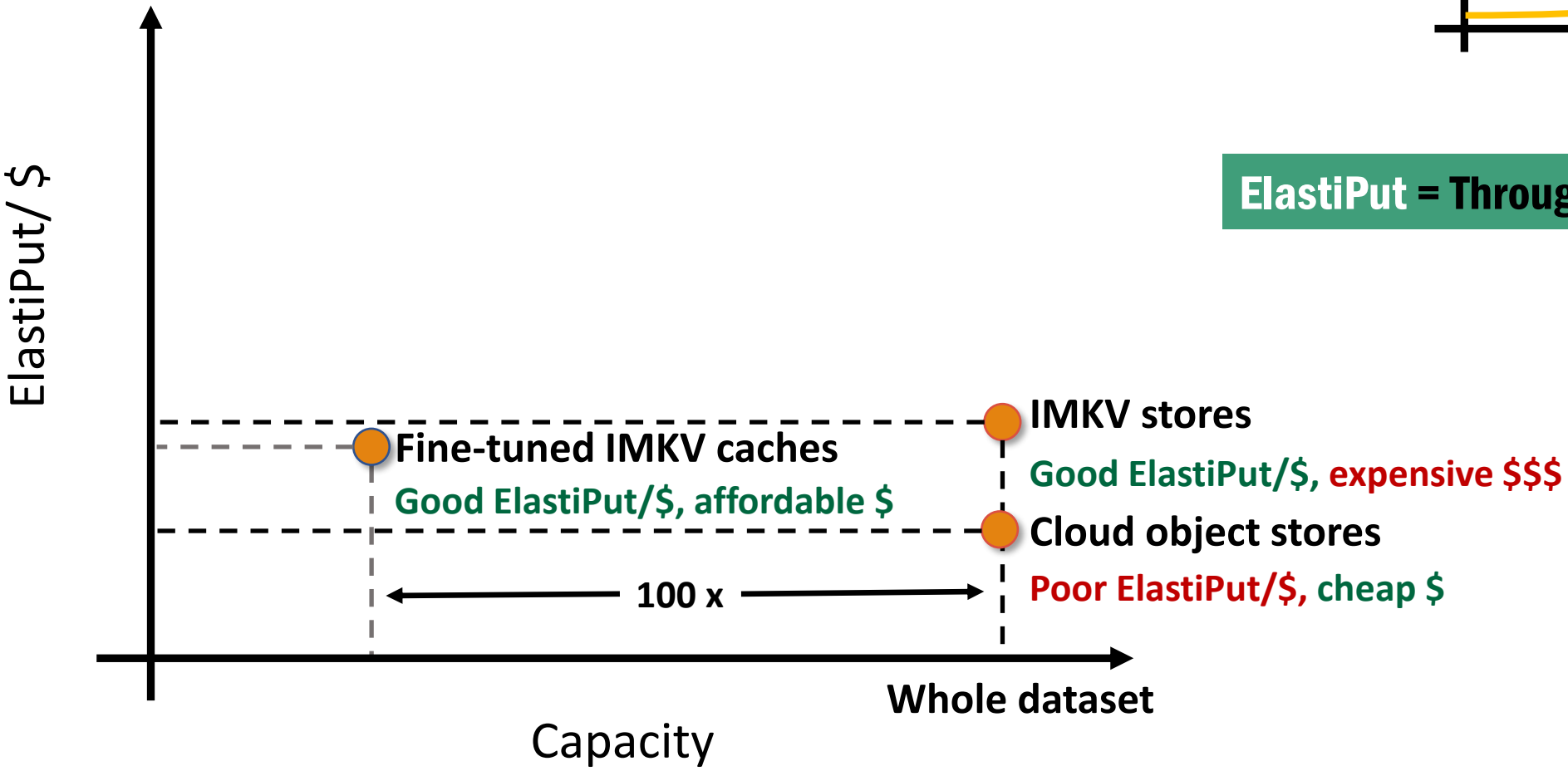
ElastiPut = Throughput (RPS) / Latency



Cost-effectiveness of existing cloud storage services



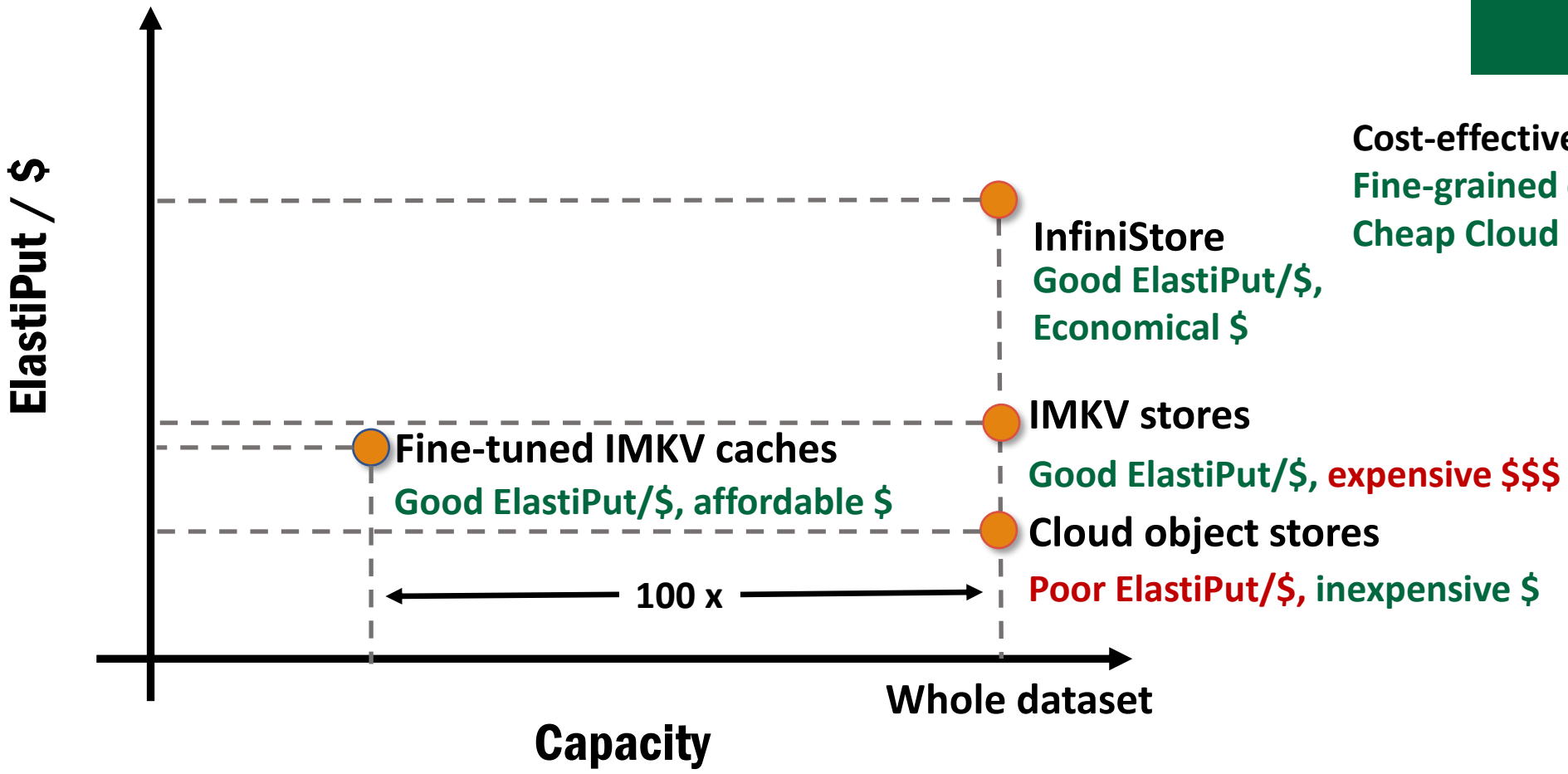
ElastiPut = Throughput (RPS) / Latency



InfiniStore

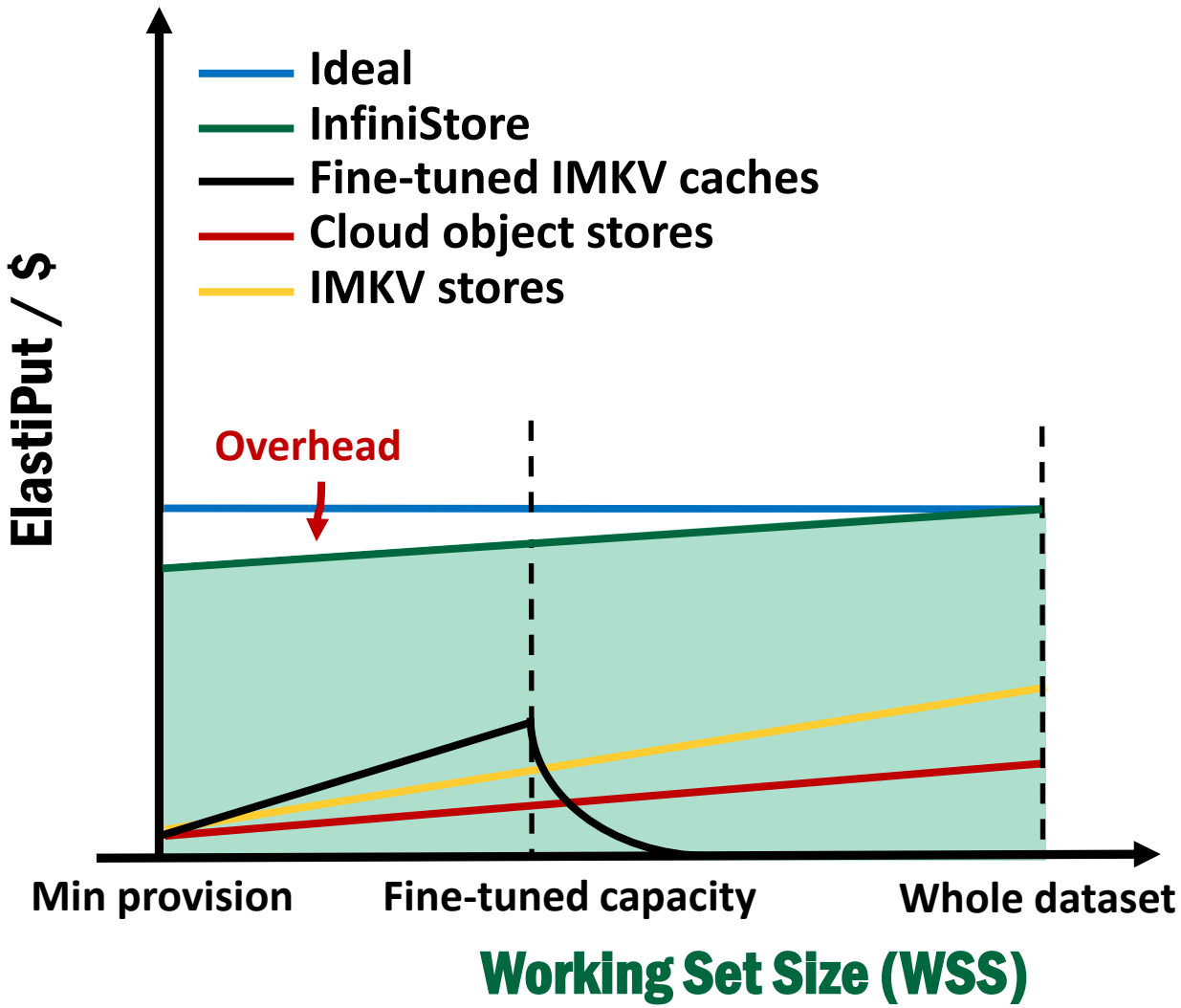
ElastiPut = Throughput (RPS) / Latency

Two-tier storage



Cost-effective:
Fine-grained elastic ServerlessMemery
Cheap Cloud Object Store (S3)

InfiniStore



ElastiPut = Throughput (RPS) / Latency

Two-tier storage

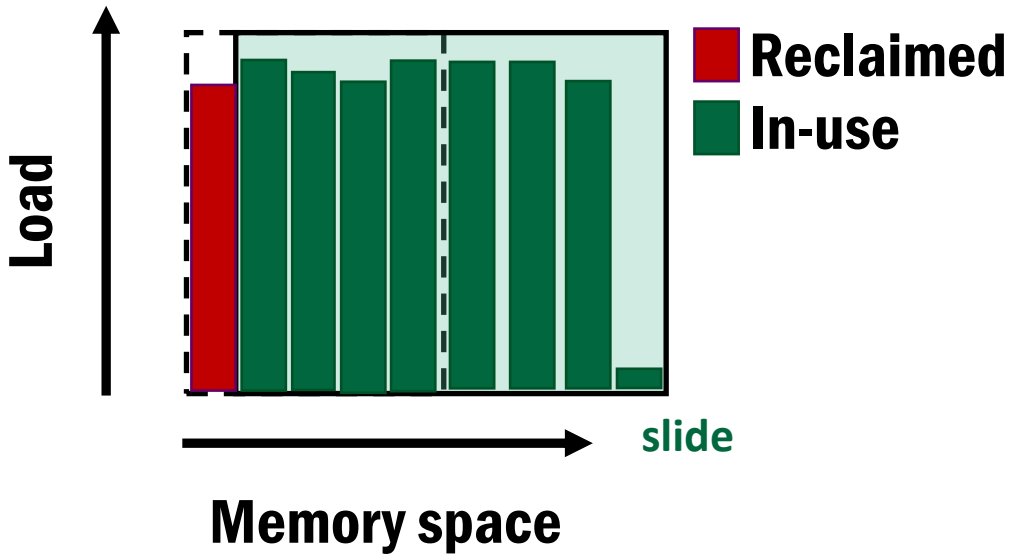
Fine-grained proportional capacity and bandwidth scaling

InfiniStore

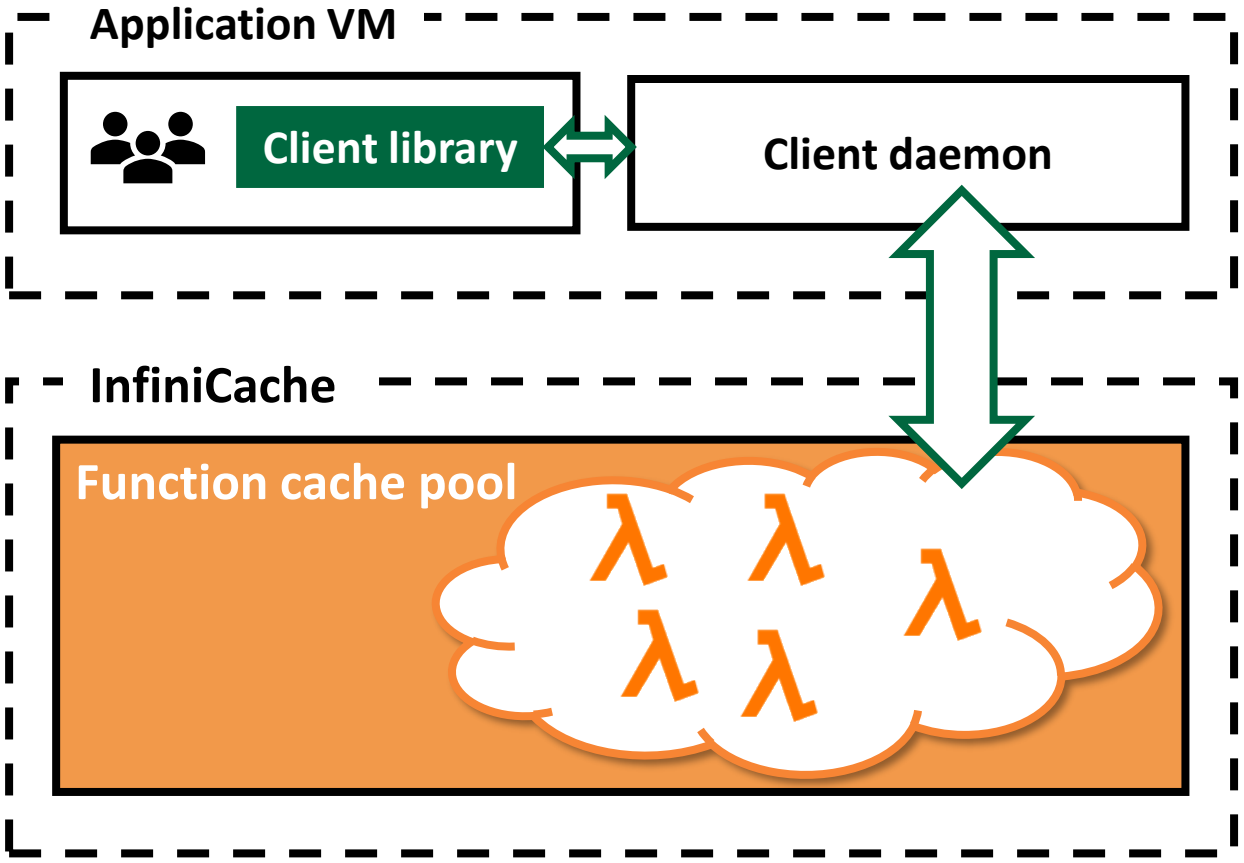
Two-tier storage

Fine-grained proportional capacity and bandwidth scaling

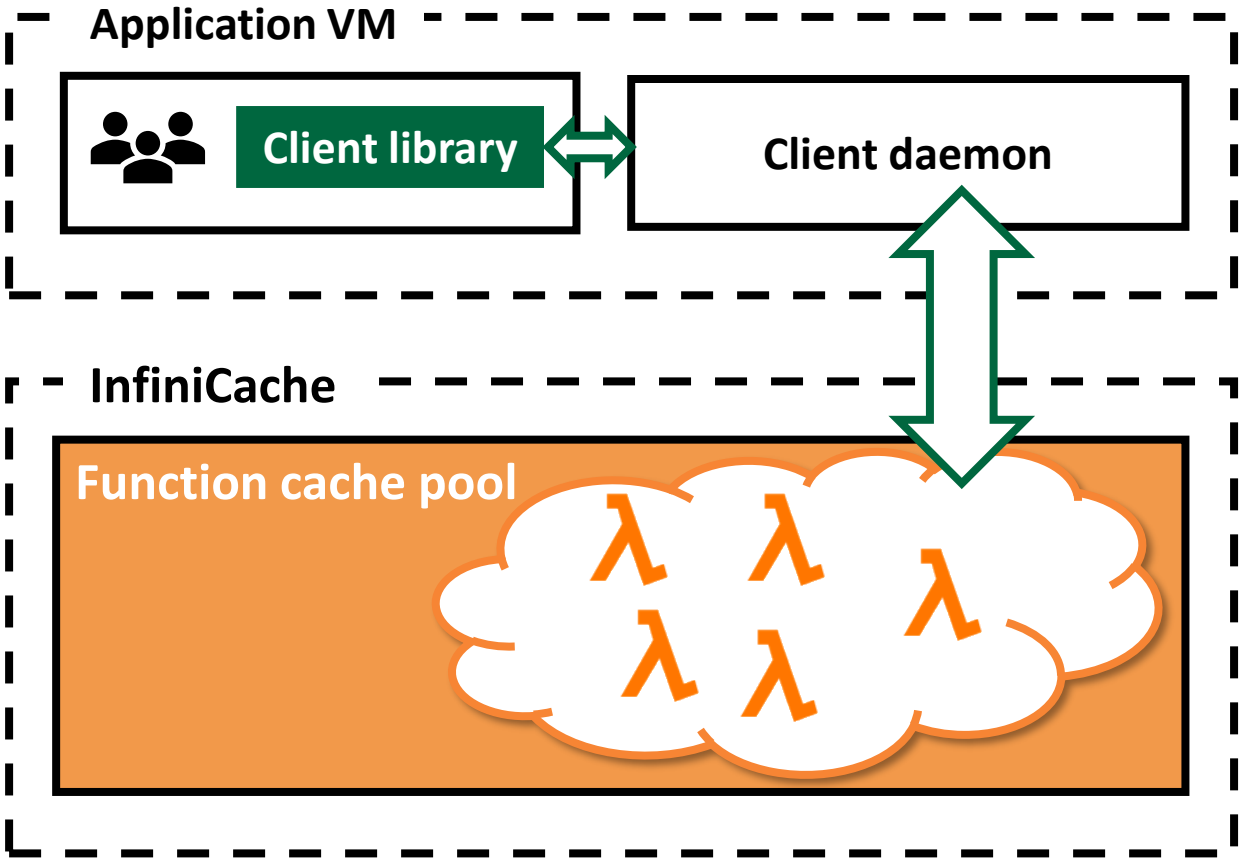
Sliding-window memory management



Background: InfiniCache review



Background: InfiniCache review

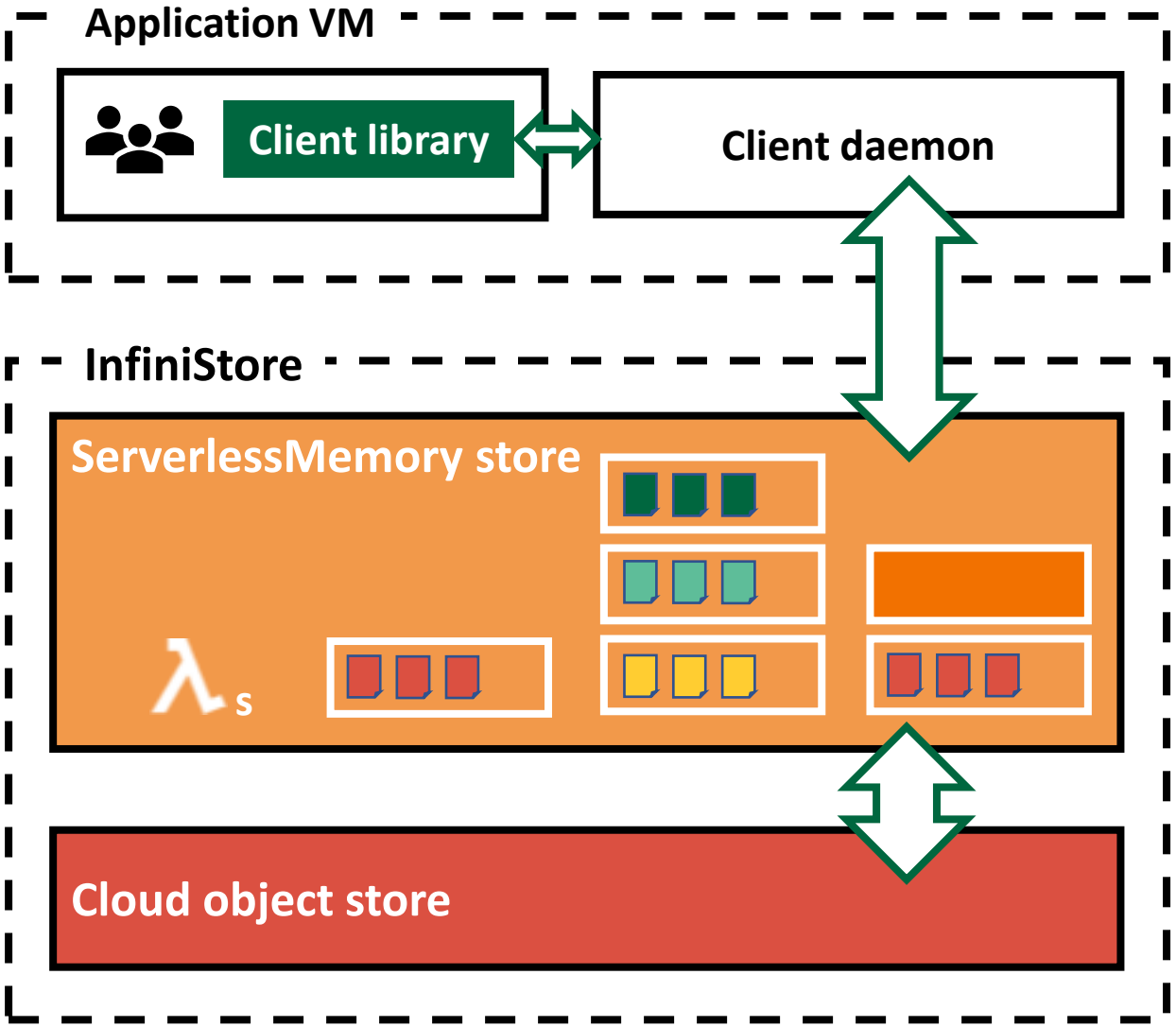


Limited max capacity

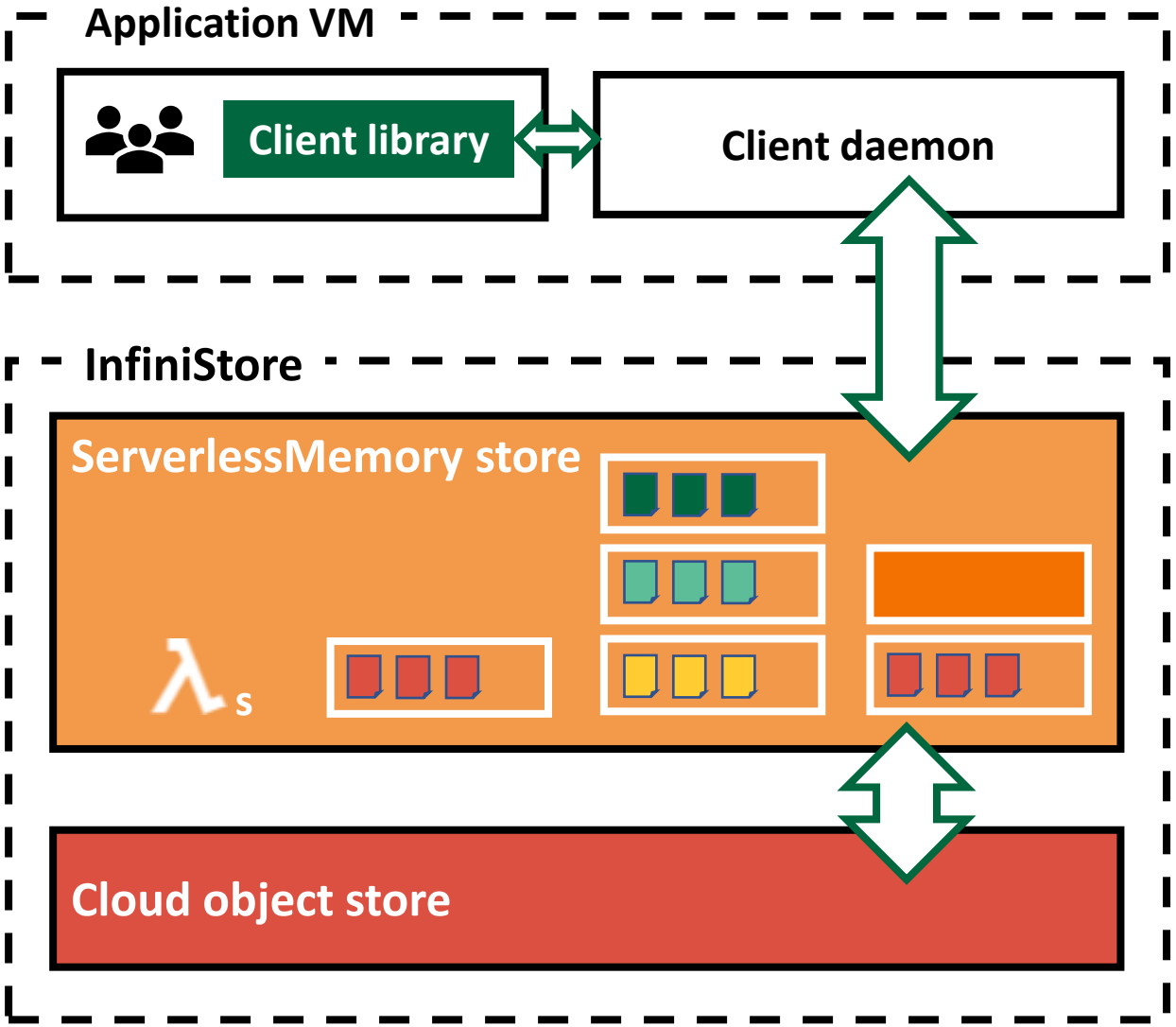
Scaling requires data rebalancing and repartitioning

Limited durability

InfiniStore: overview



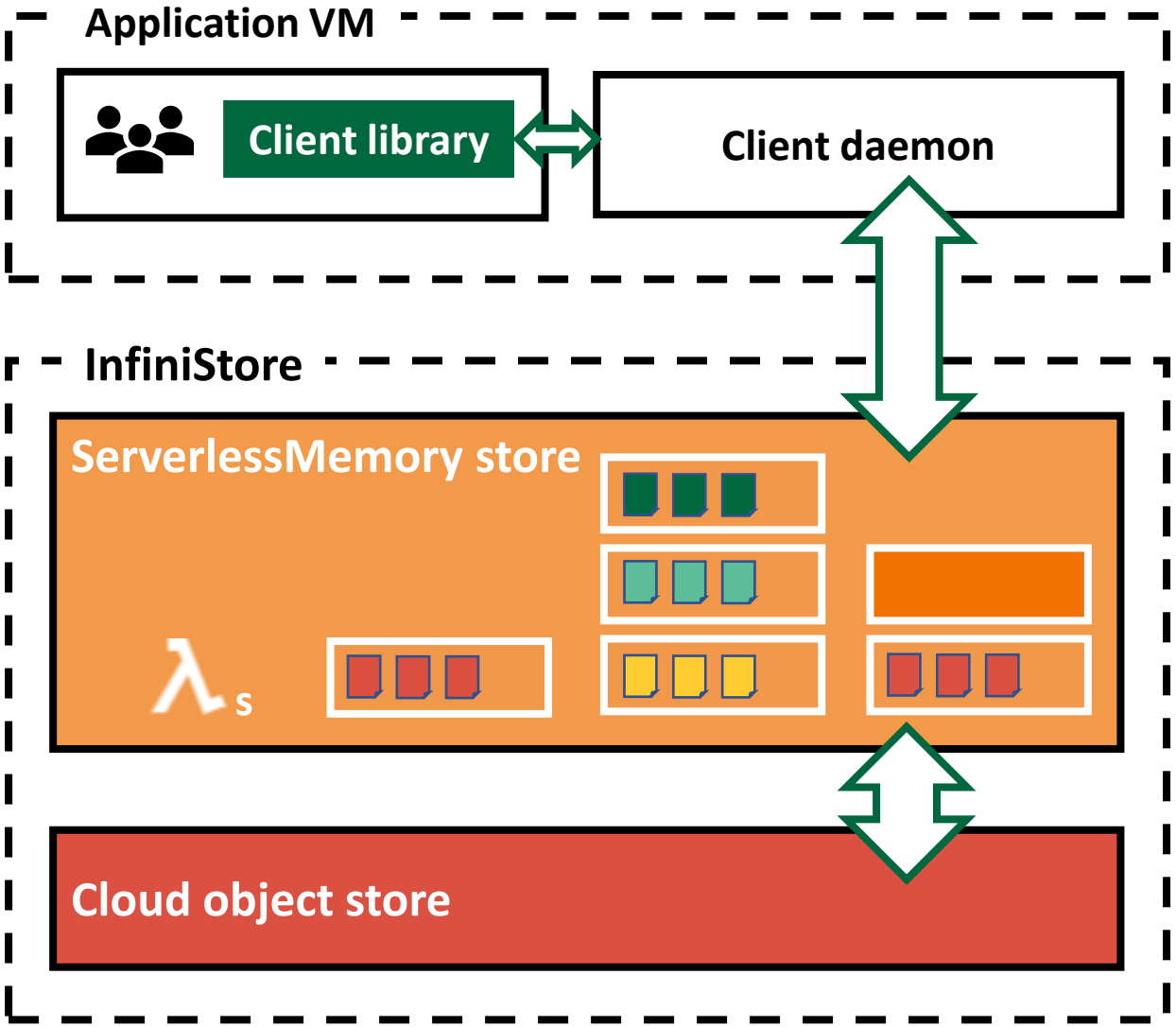
InfiniStore: overview



ServerlessMemory store

Periodic warmup of Lambdas

InfiniStore: overview

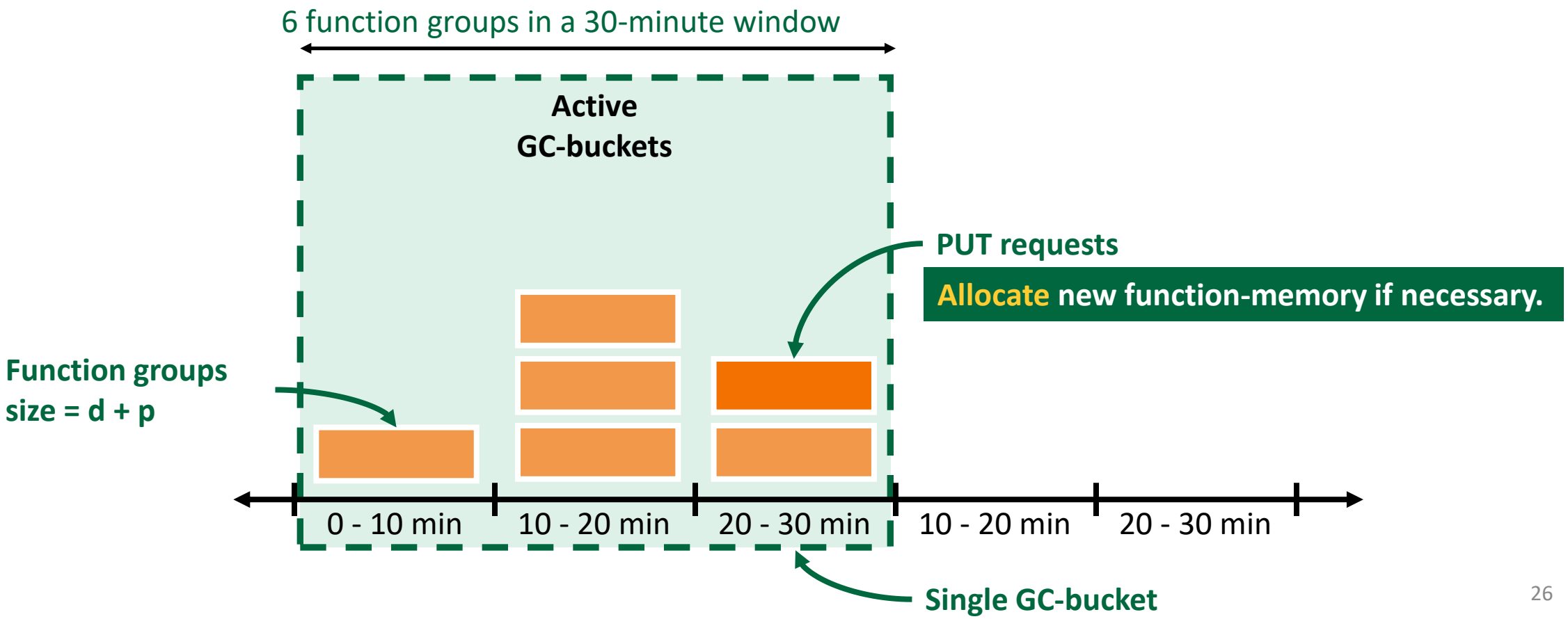


ServerlessMemory store

Sliding-window memory management

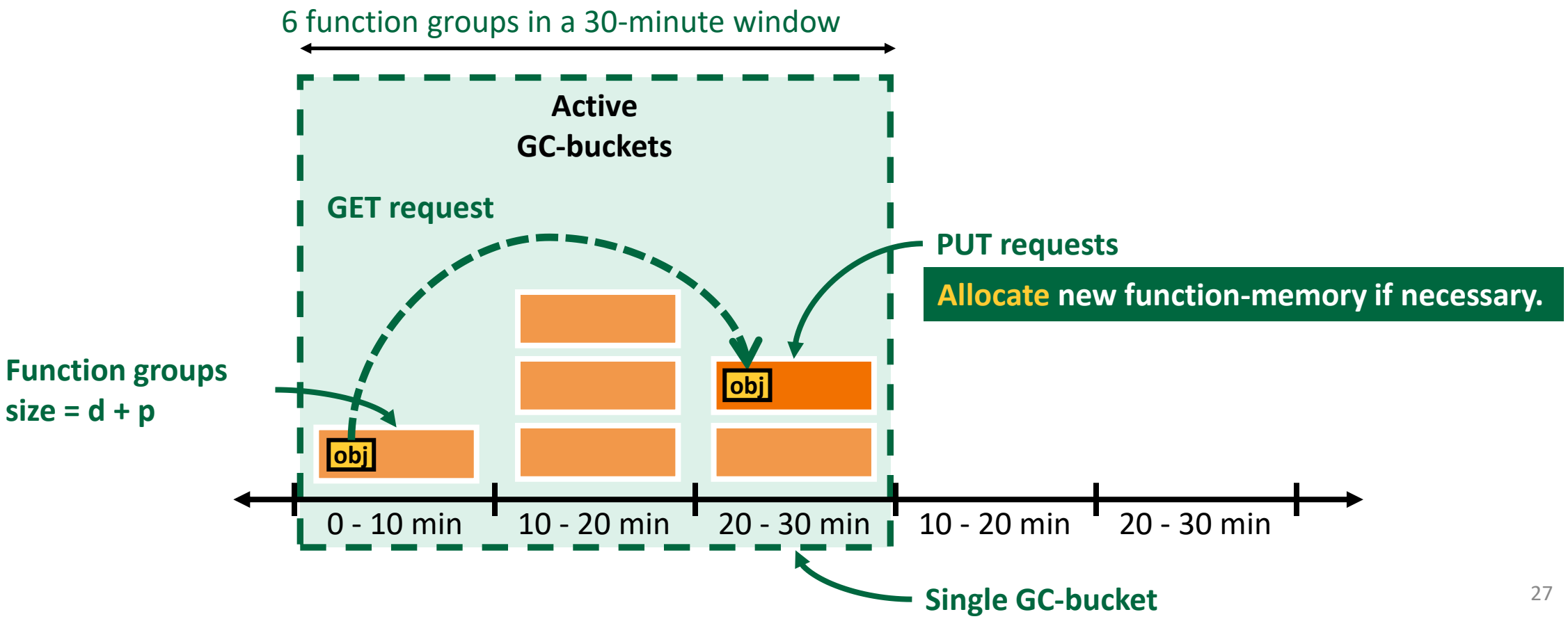
ServerlessMemory: sliding-window memory management

- **Lambda functions are organized by Function Groups (FG) and GC-buckets**



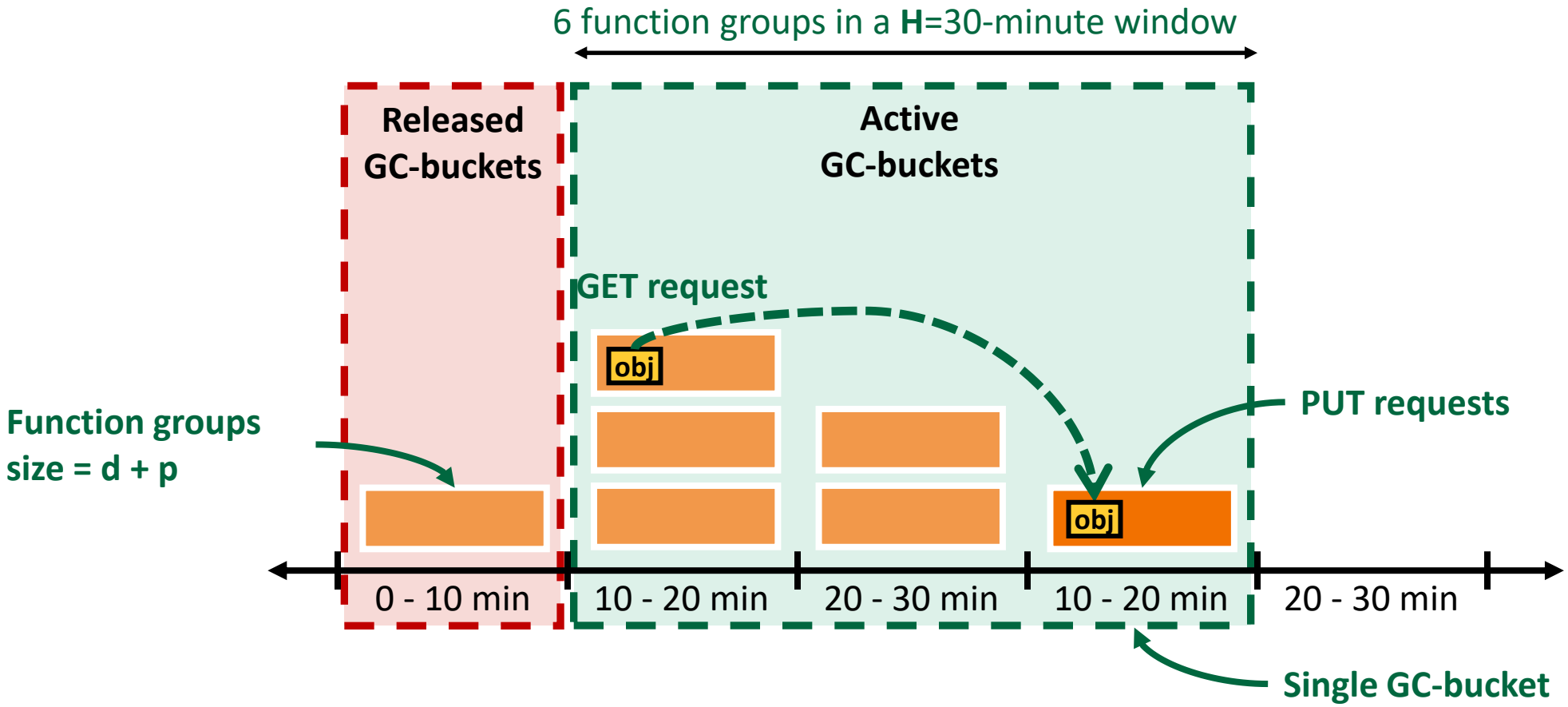
ServerlessMemory: sliding-window memory management

- **Lambda functions are organized by Function Groups (FG) and GC-buckets**



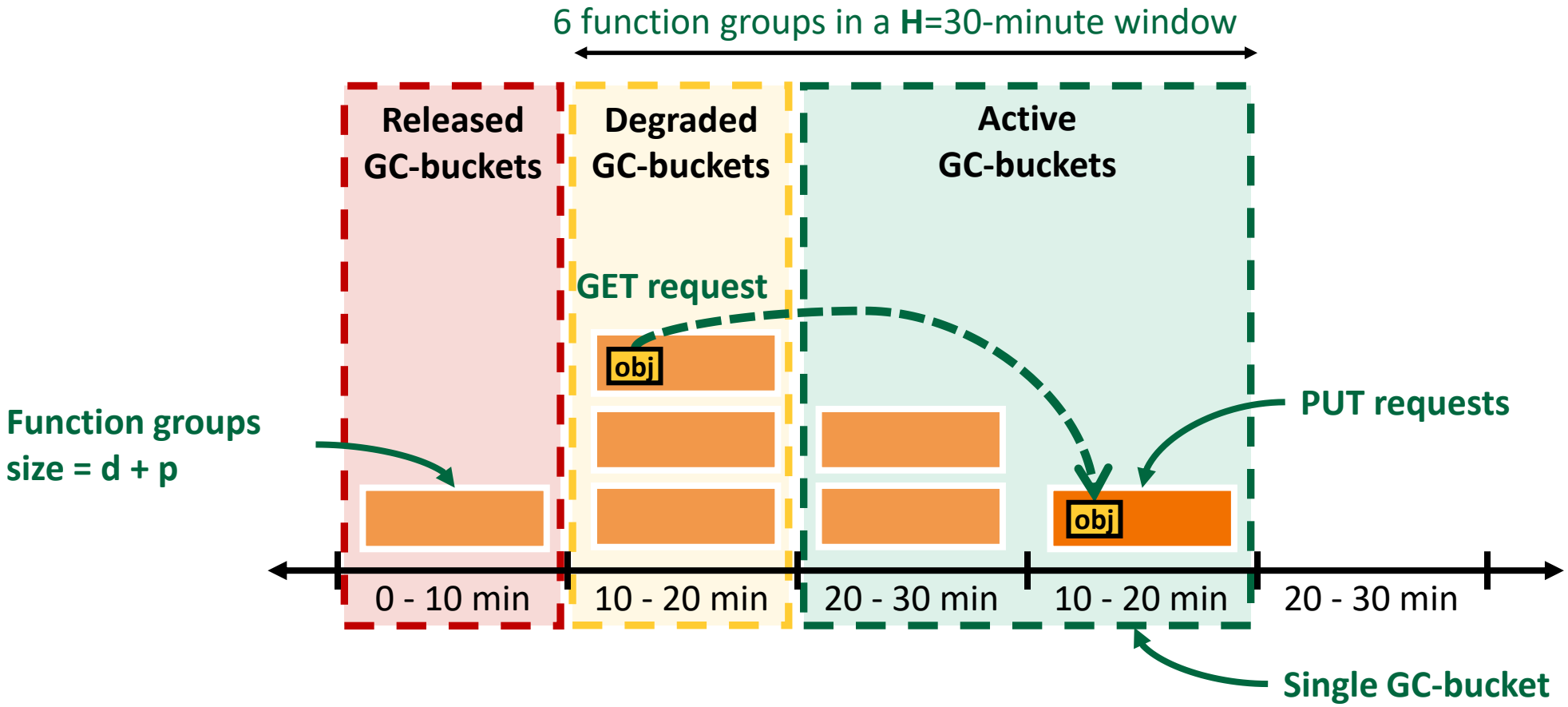
ServerlessMemory: sliding-window memory management

- Lambda functions are organized by **Function Groups (FG)** and **GC-Buckets**
- Inspired by **mark-compact garbage collection** (GC) in programming languages



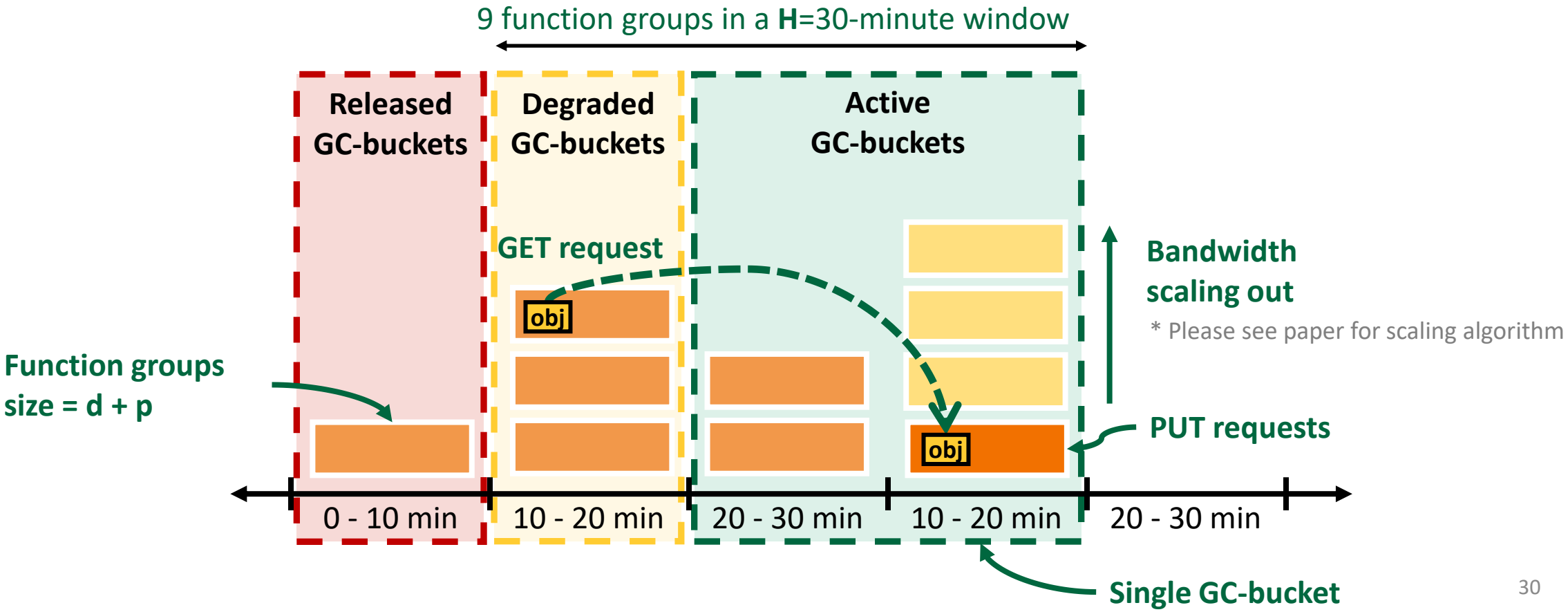
ServerlessMemory: sliding-window memory management

- Lambda functions are organized by **Function Groups (FG)** and **GC-Buckets**
- Inspired by **mark-compact garbage collection** (GC) in programming languages



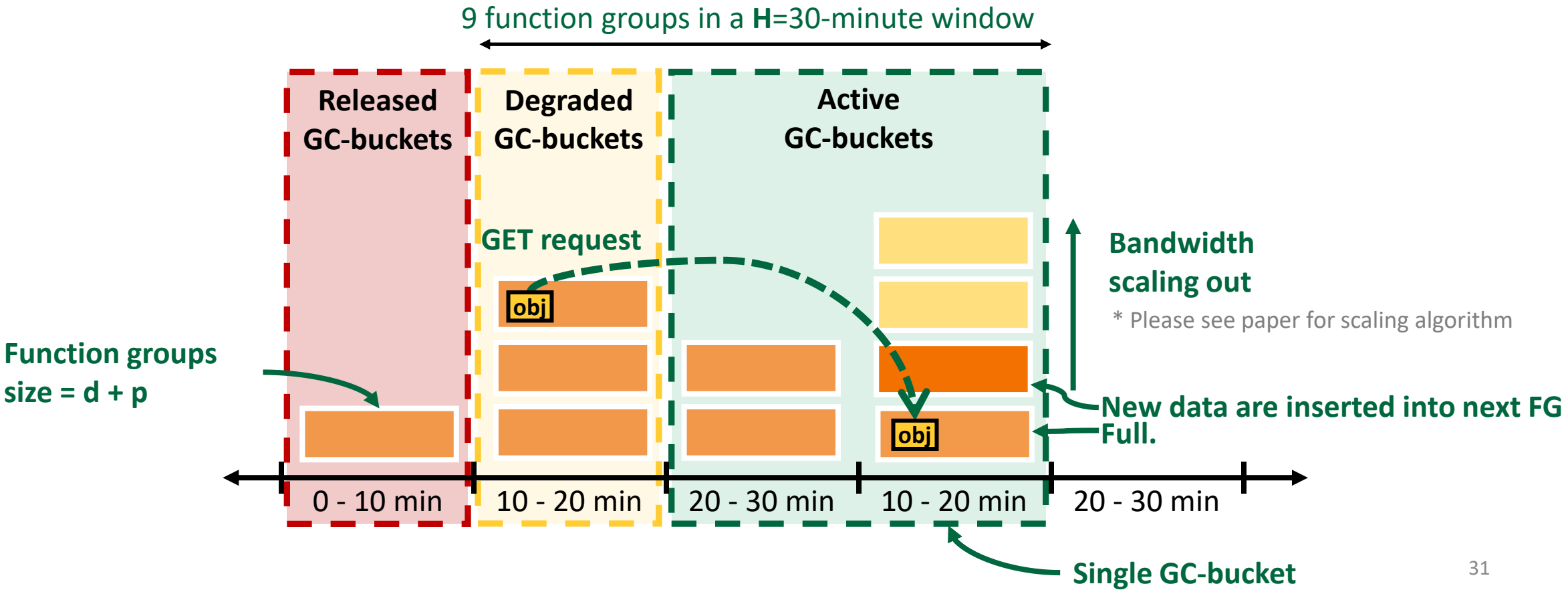
ServerlessMemory: sliding-window memory management

- Lambda functions are organized by **Function Groups (FG)** and **GC-Buckets**
- Inspired by **mark-compact garbage collection** (GC) in programming languages



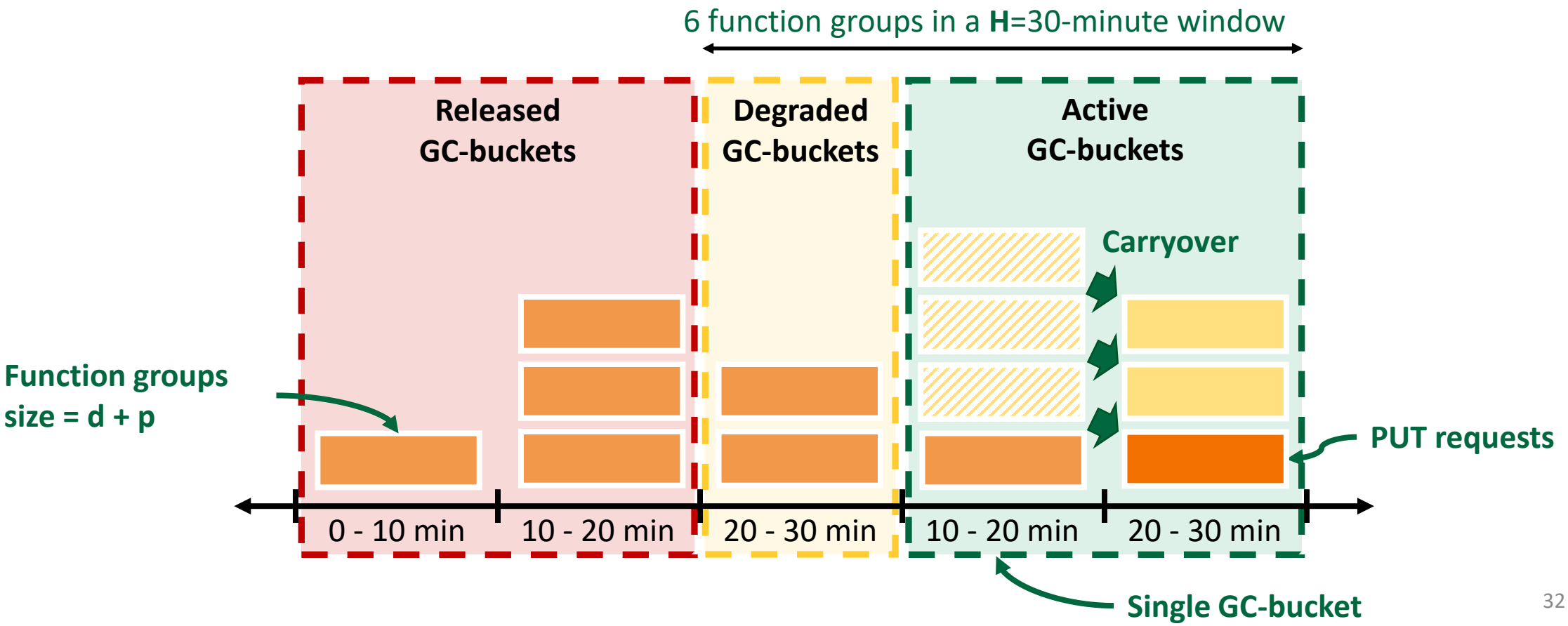
ServerlessMemory: sliding-window memory management

- Lambda functions are organized by **Function Groups (FG)** and **GC-Buckets**
- Inspired by **mark-compact garbage collection** (GC) in programming languages

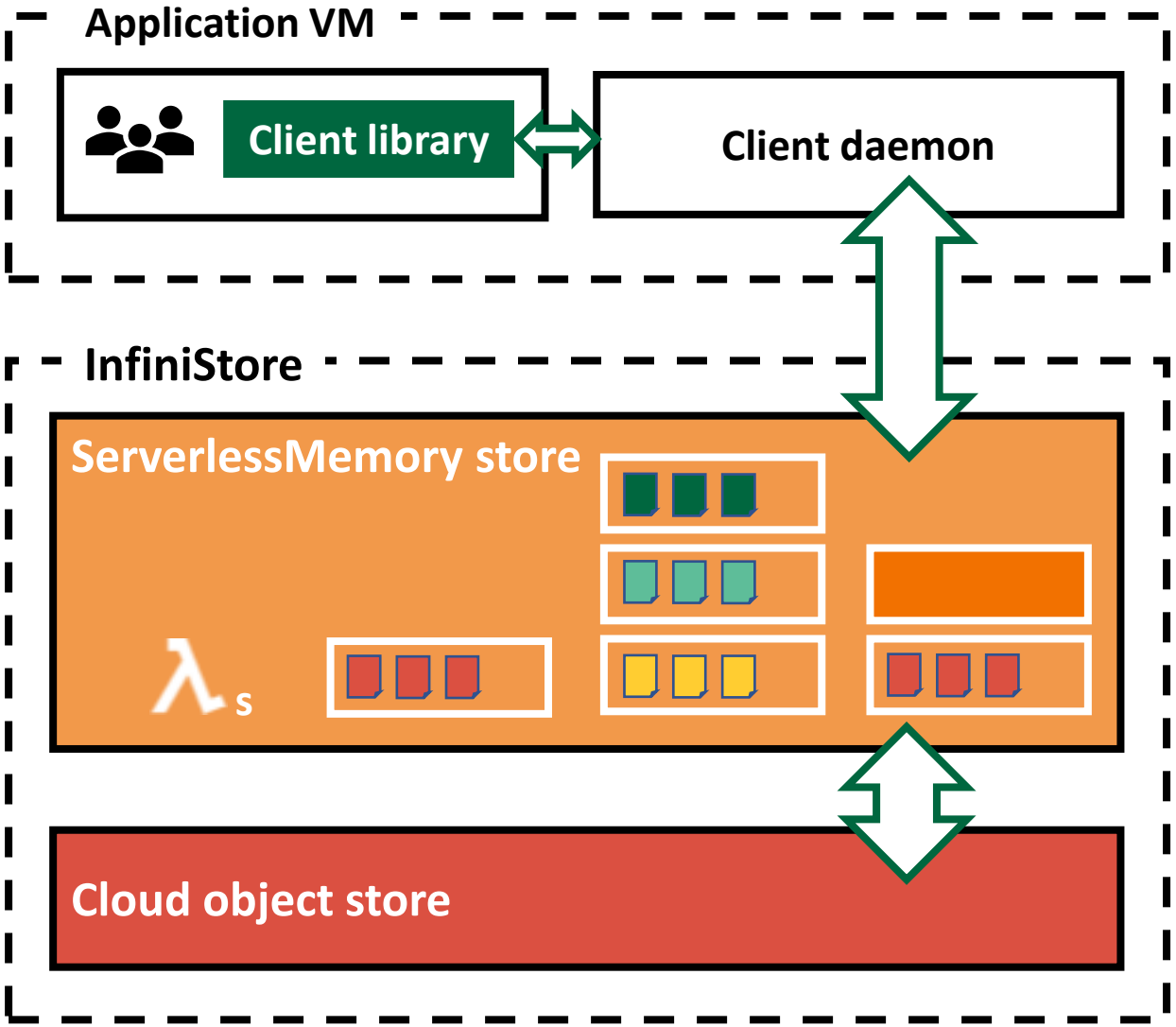


ServerlessMemory: sliding-window memory management

- Lambda functions are organized by **Function Groups (FG)** and **GC-Buckets**
- Inspired by **mark-compact garbage collection** (GC) in programming languages



InfiniStore: overview



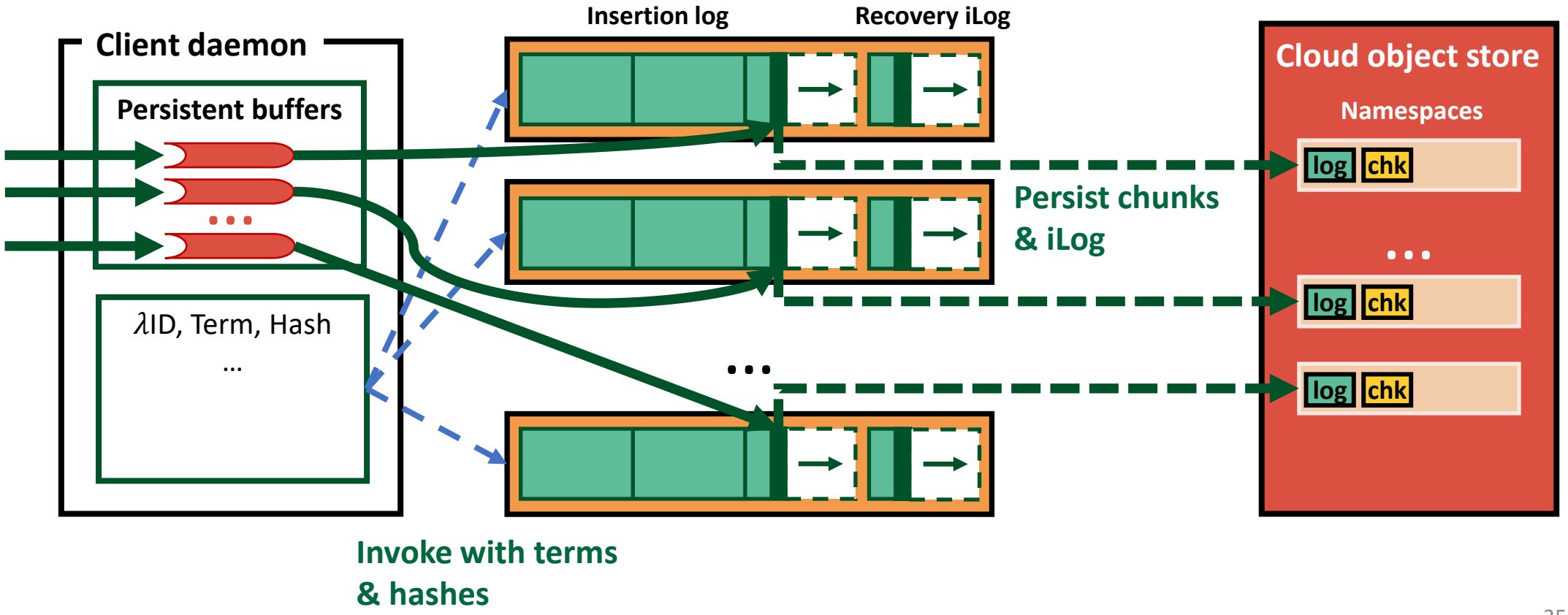
Sliding-window memory management

Durability

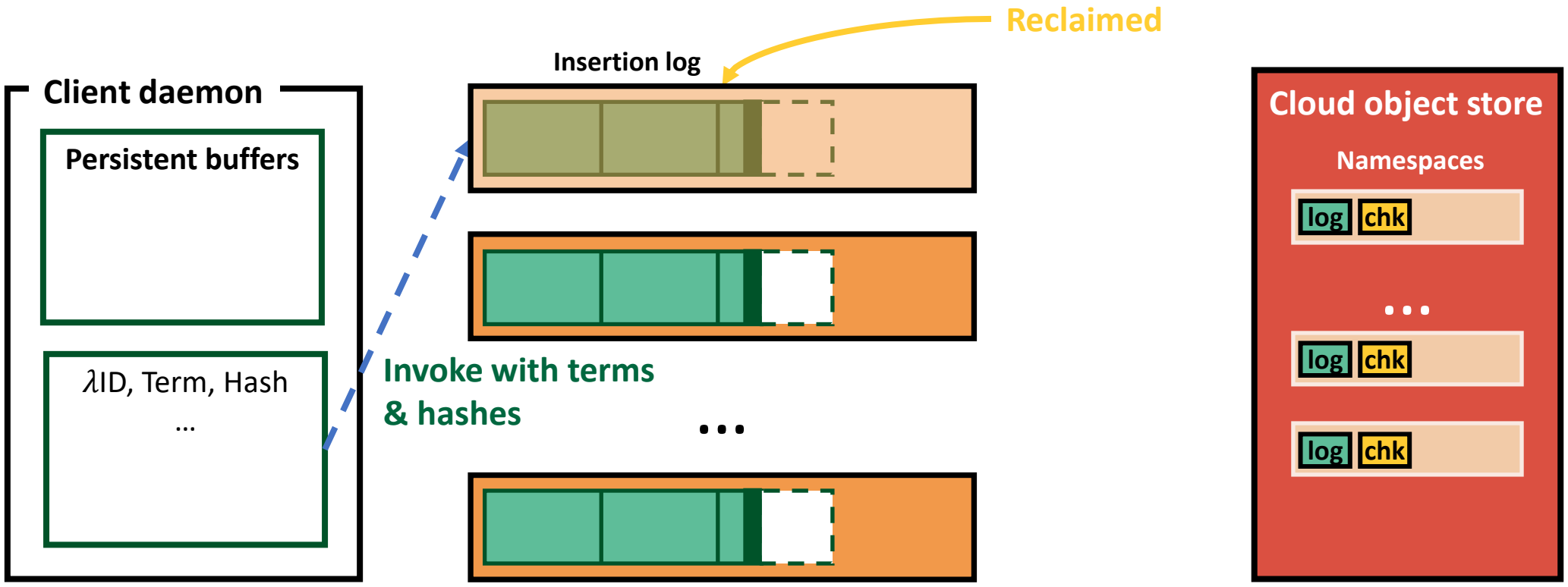
Parallel recovery

Low latency PUTs: see the paper

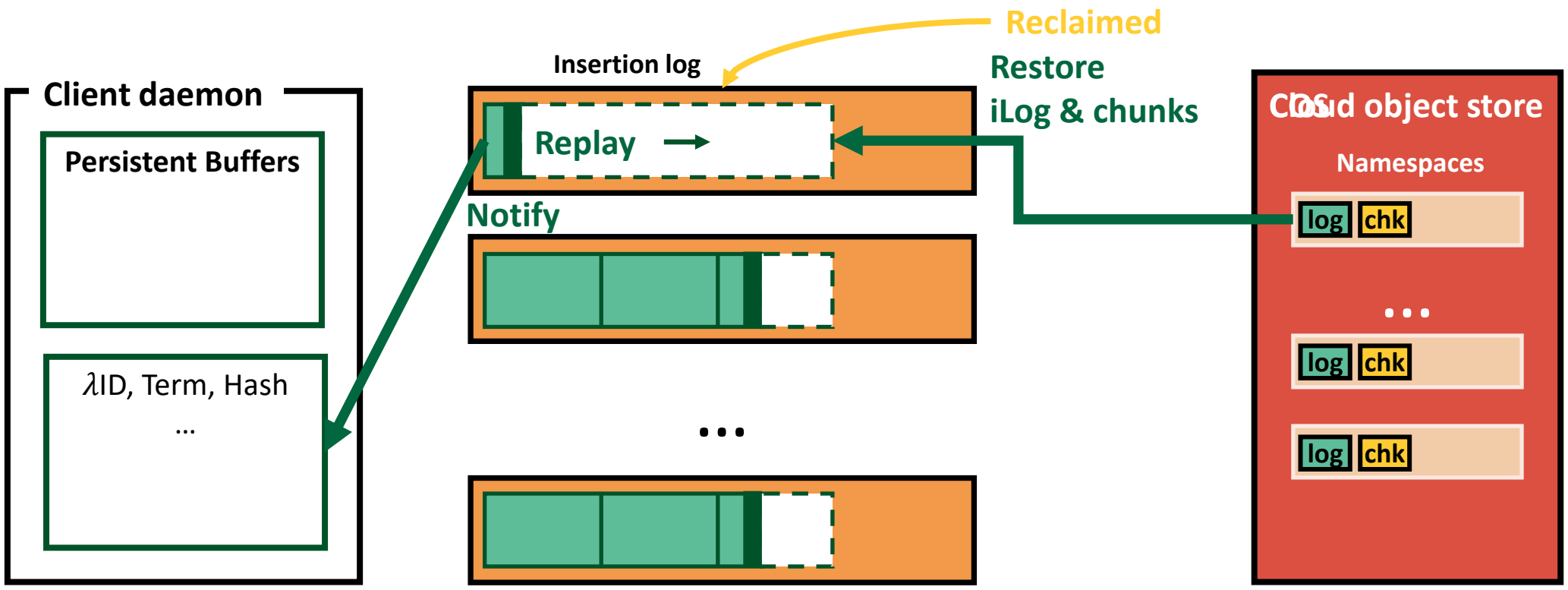
Durability: overview



Durability: parallel recovery

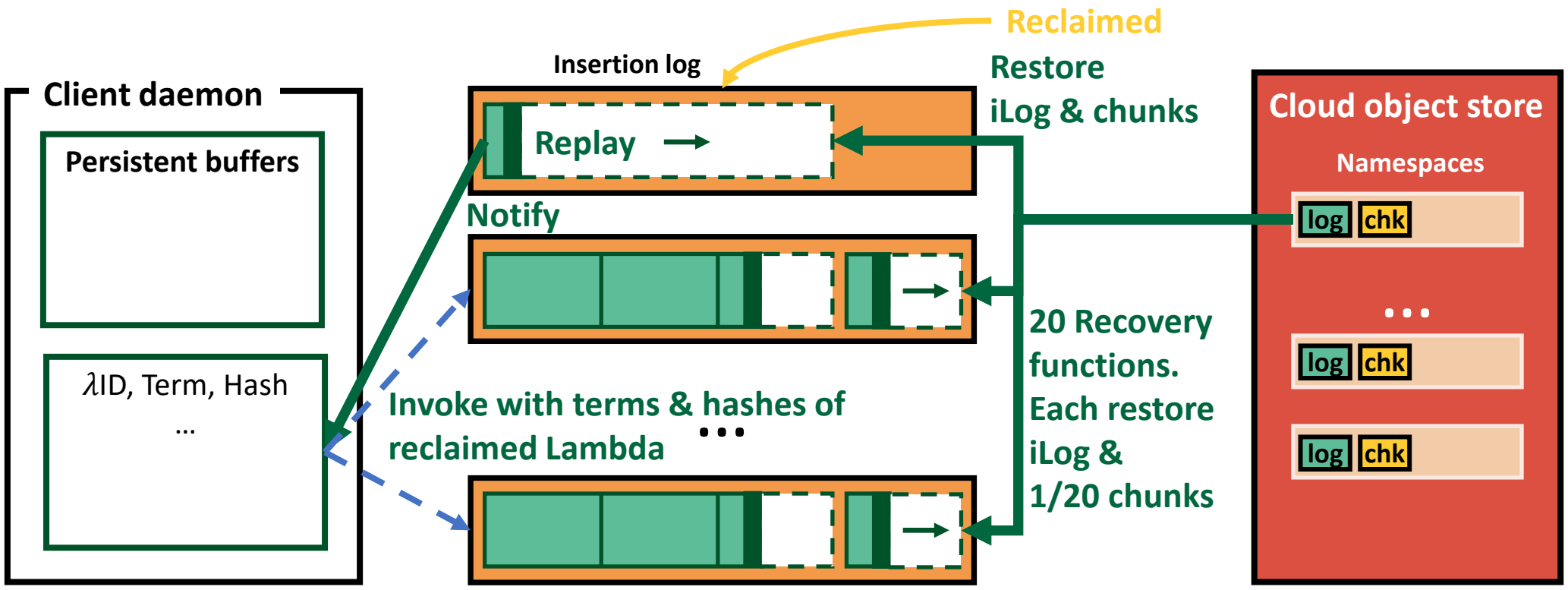


Durability: parallel recovery



Slow: 1.5 GB / 80 MB/s = 18.75 seconds

Durability: parallel recovery



Fast: 1.5 GB x 0.90 / 20 / 80 MB/s = 0.85 seconds

Evaluations

Applications: Workload Replay

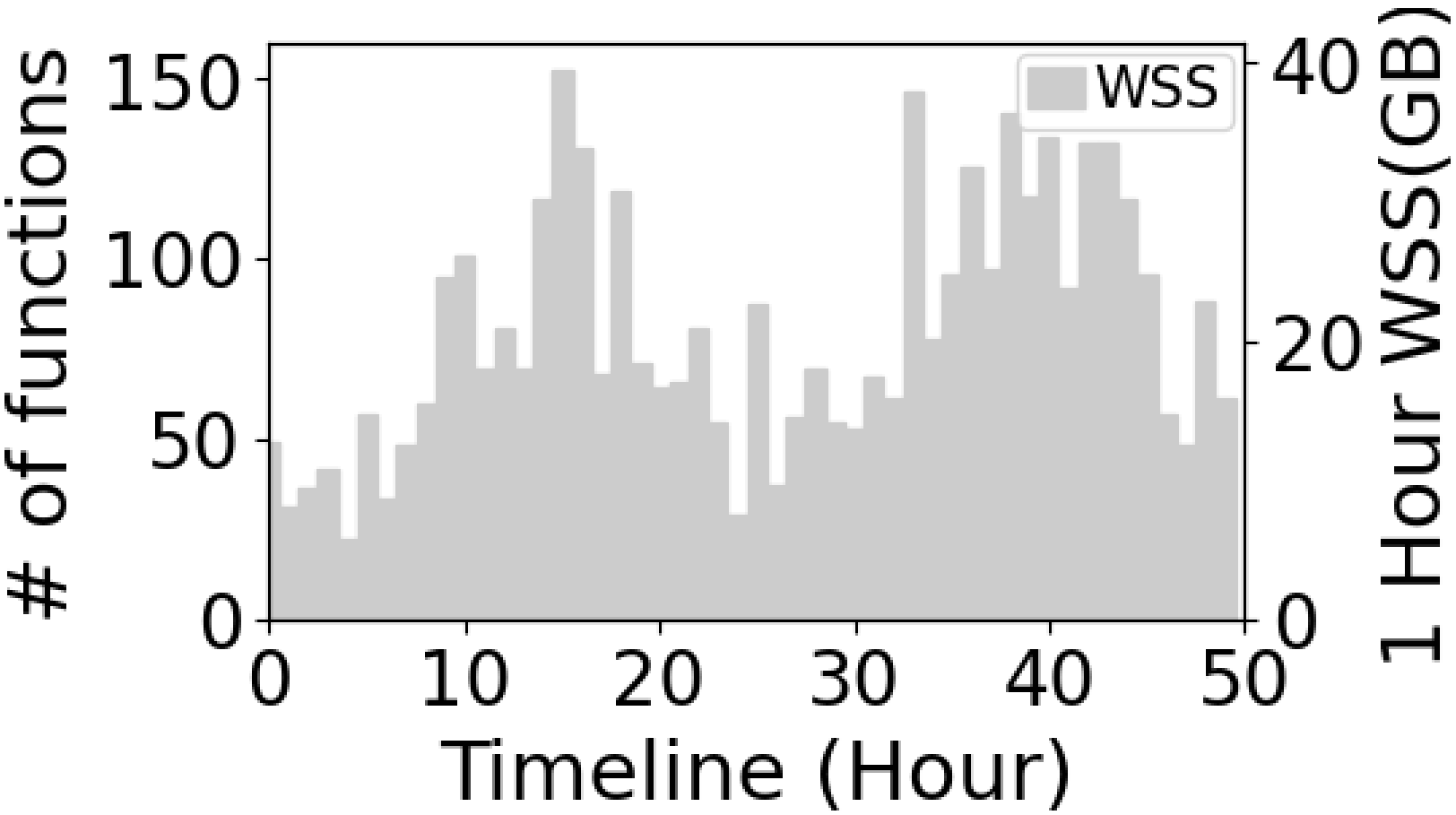
50-hour IBM container registry workload

- Replay of the first 50-hours of a 75-day IBM container-registry trace from the Dallas datacenter.

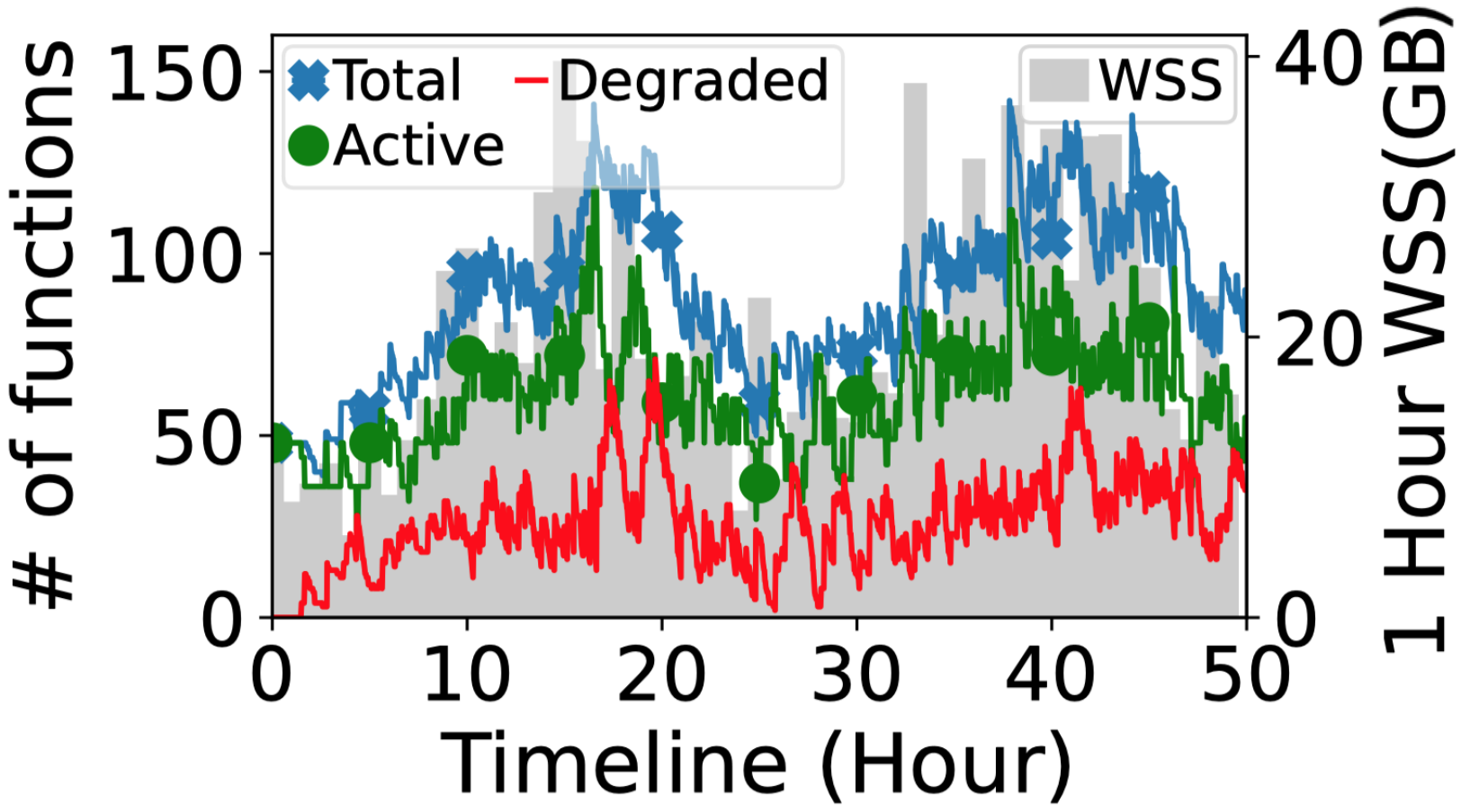
5-hour Azure blob access workload

- Select 5-hours (hours 179 – 184) of the most bursty traces from 37 serverless applications

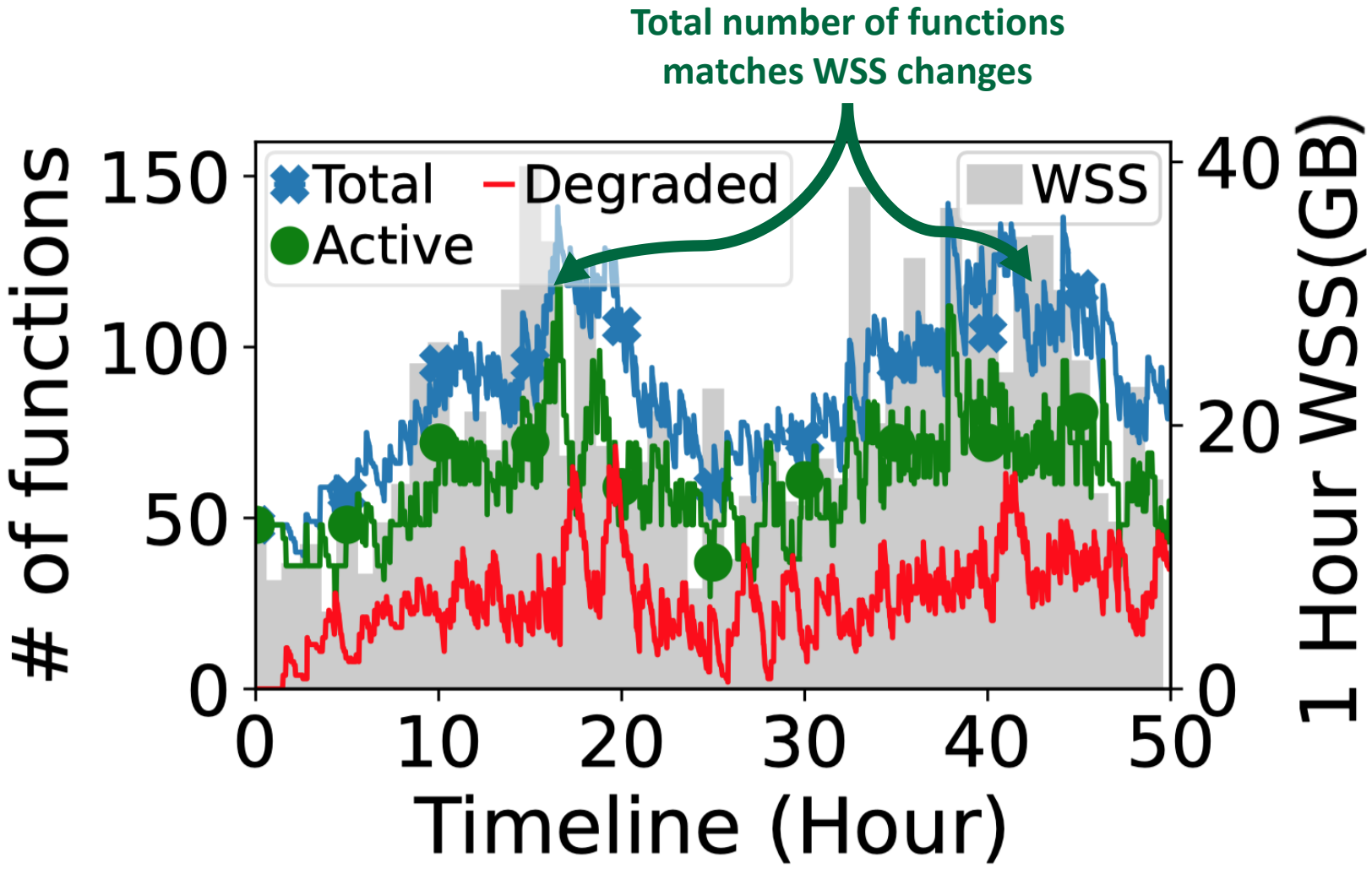
Replay 50-hour IBM container registry workload



Replay 50-hour IBM container registry workload



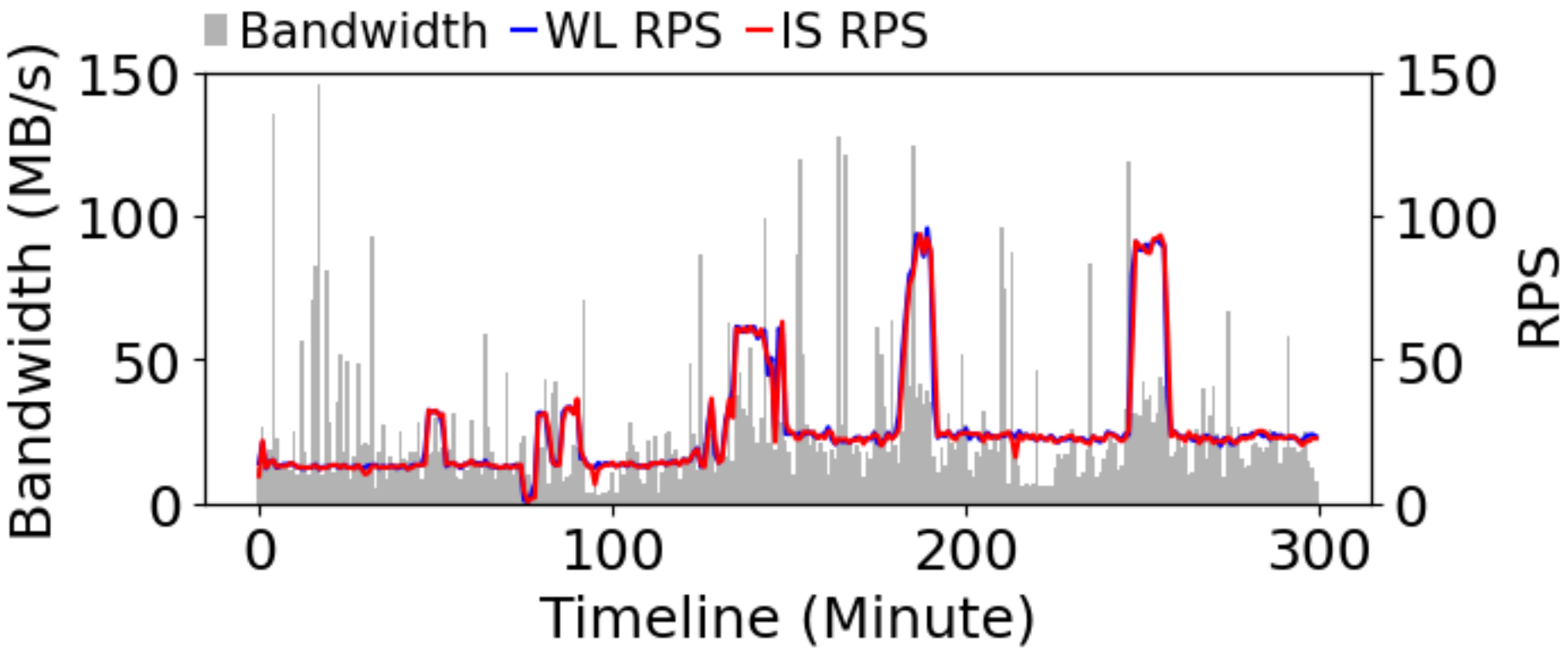
Replay 50-hour IBM container registry workload



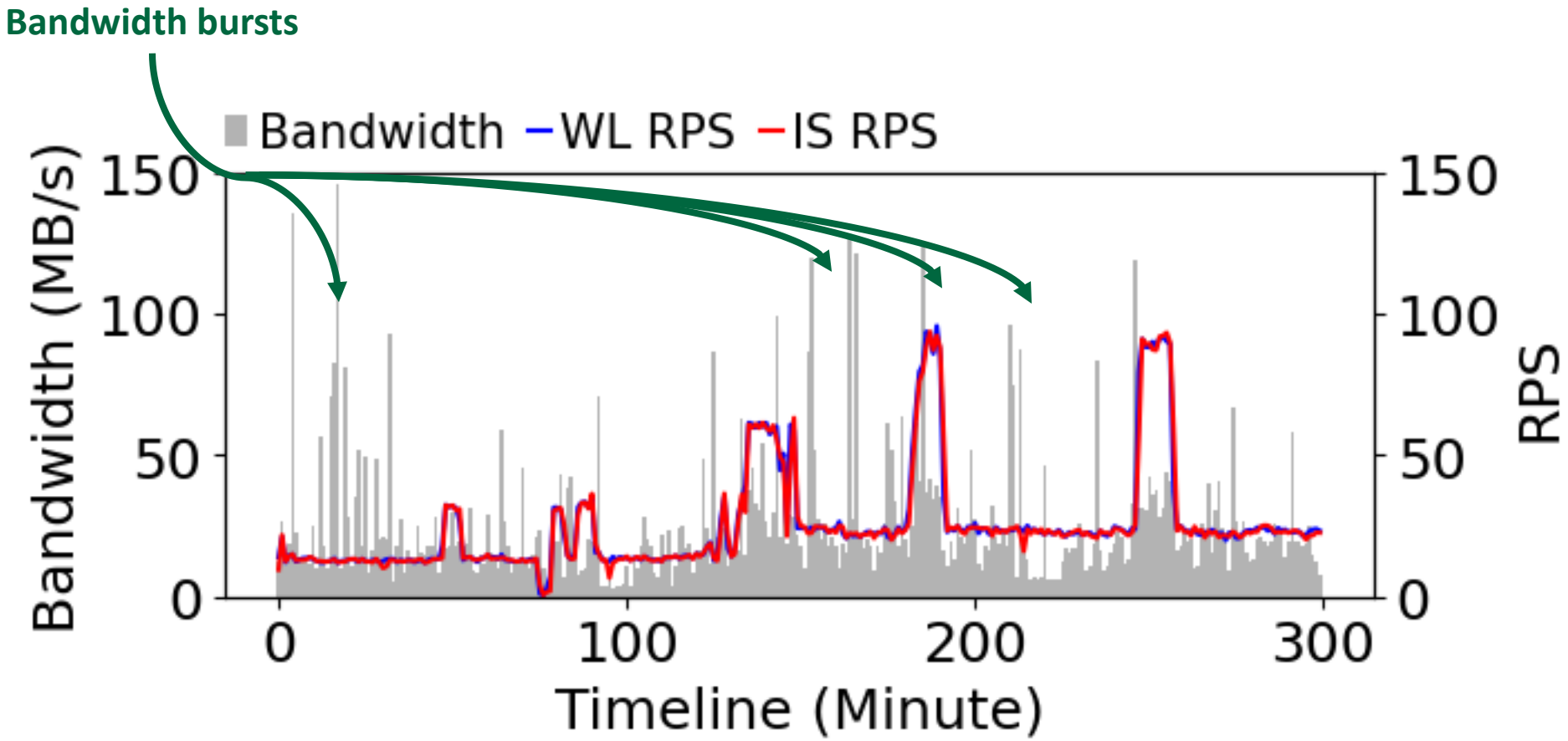
Replay 5-hour Azure blob access workload

IS RPS: throughput achieved while replaying workload with InfiniStore.

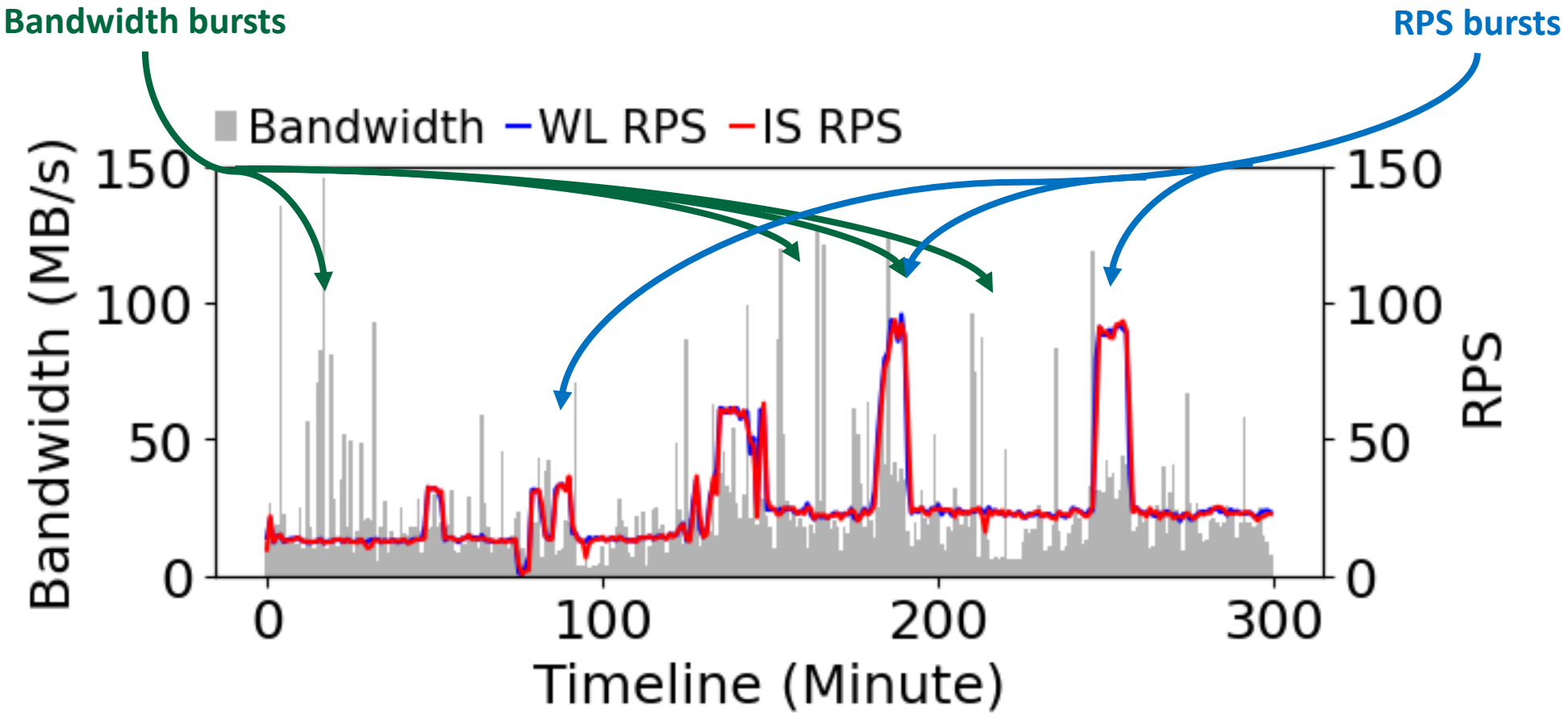
WL RPS: original workload throughput as recorded in traces.



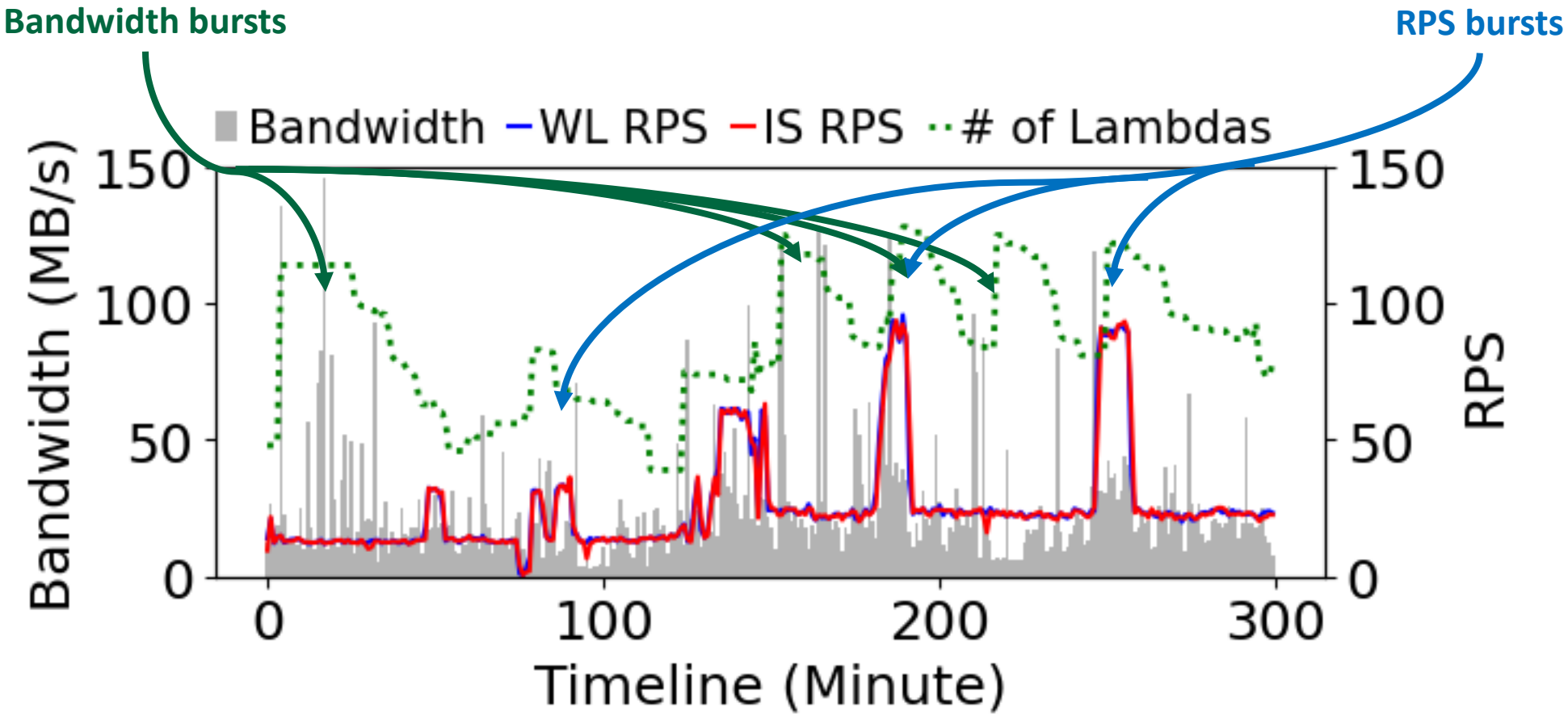
Replay 5-hour Azure blob access workload



Replay 5-hour Azure blob access workload



Replay 5-hour Azure blob access workload



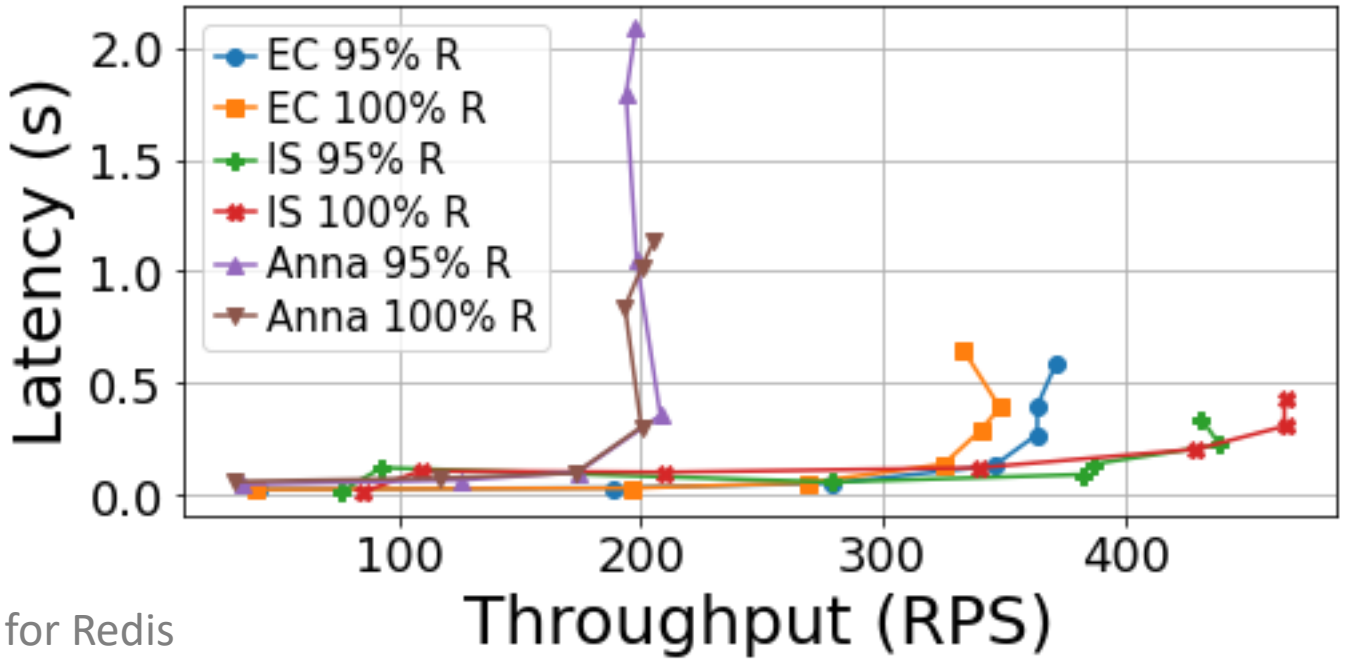
YCSB microbenchmarking

- Stress-test on ElastiPut/$ for fixed-size objects.
- Baselines: EC*, Anna*.
- Bandwidth (YCSB driver): fixed 50 Gbps.
- Bandwidth (EC, Anna): maximum 30 Gbps.
- Read dominant workloads: 95% read, read-only.
- Object sizes: 1 MB, 10 MB, 100 MB.
- Load concurrency: 1, 5, 10, 25, 50, 75, 100.
- Run for 30 seconds.
- 90ile Latency.

* EC: ElastiCache for Redis
* Anna: A KVS for Any Scale, IEEE Transactions on Knowledge and Data Engineering, 2021

YCSB microbenchmarking: 10MB

- **Stress-test on ElastiPut/$ for fixed-size objects**
- **Bandwidth (YCSB driver): fixed 50 Gbps**
- **Bandwidth (EC*, Anna*): maximum 30 Gbps**

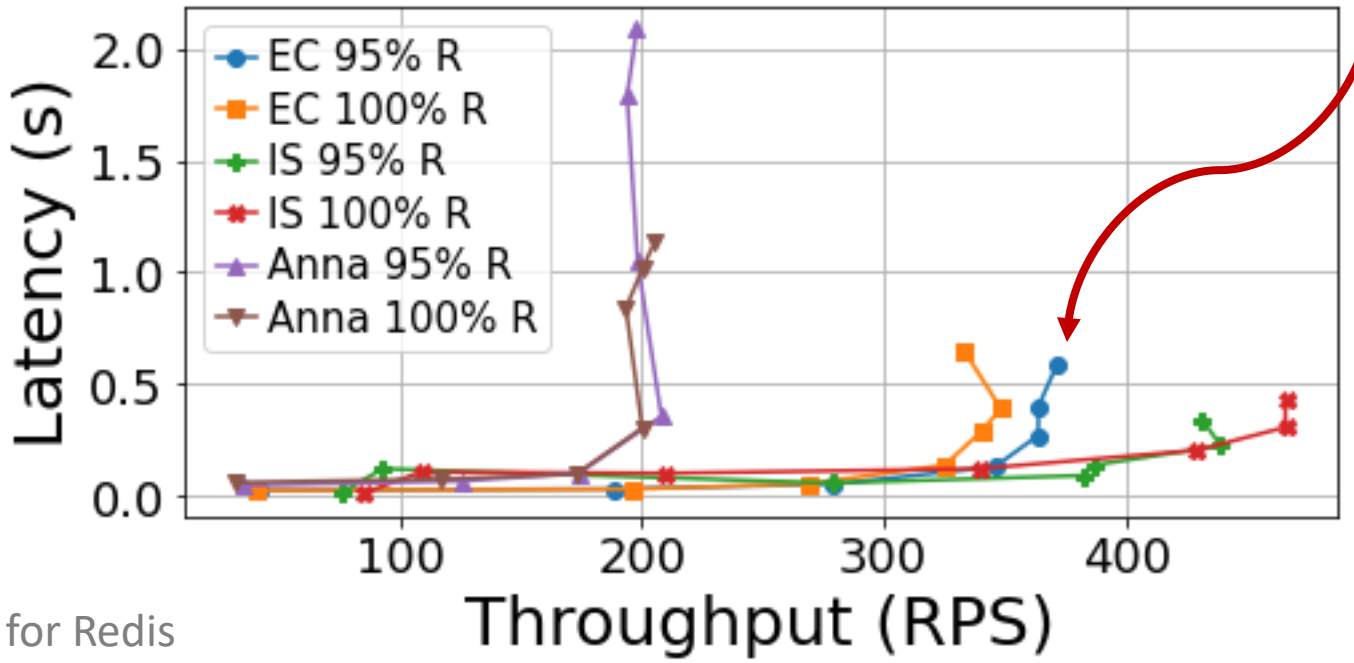


* EC: ElastiCache for Redis
* Anna: A KVS for Any Scale, IEEE Transactions on Knowledge and Data Engineering, 2021

YCSB microbenchmarking: 10MB

- Stress-test on **ElastiPut/$** for fixed-size objects
- Bandwidth (YCSB driver): fixed **50** Gbps
- Bandwidth (EC*, Anna*): maximum **30** Gbps

ElastiPut/$ degradation after cluster bandwidth limit is hit.

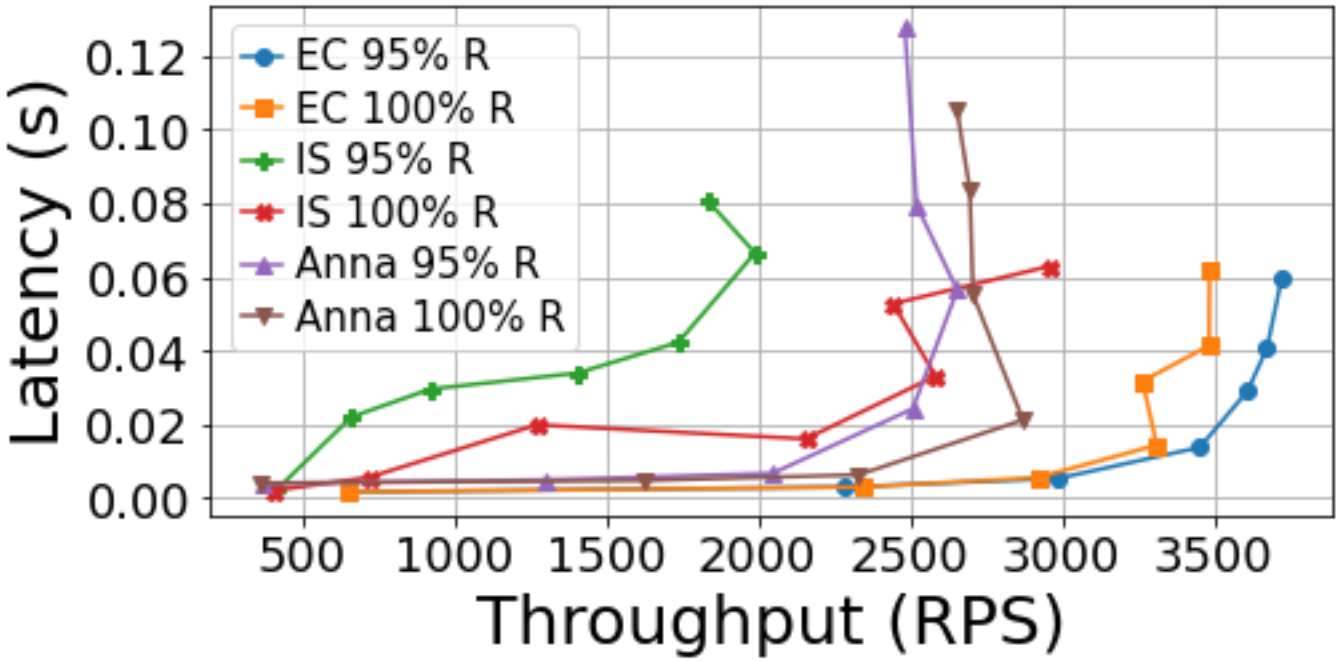


No ElastiPut/$ degradation
(Simulation driver becomes bottleneck)

* EC: ElastiCache for Redis
* Anna: A KVS for Any Scale, IEEE Transactions on Knowledge and Data Engineering, 2021

YCSB microbenchmarking: 1MB

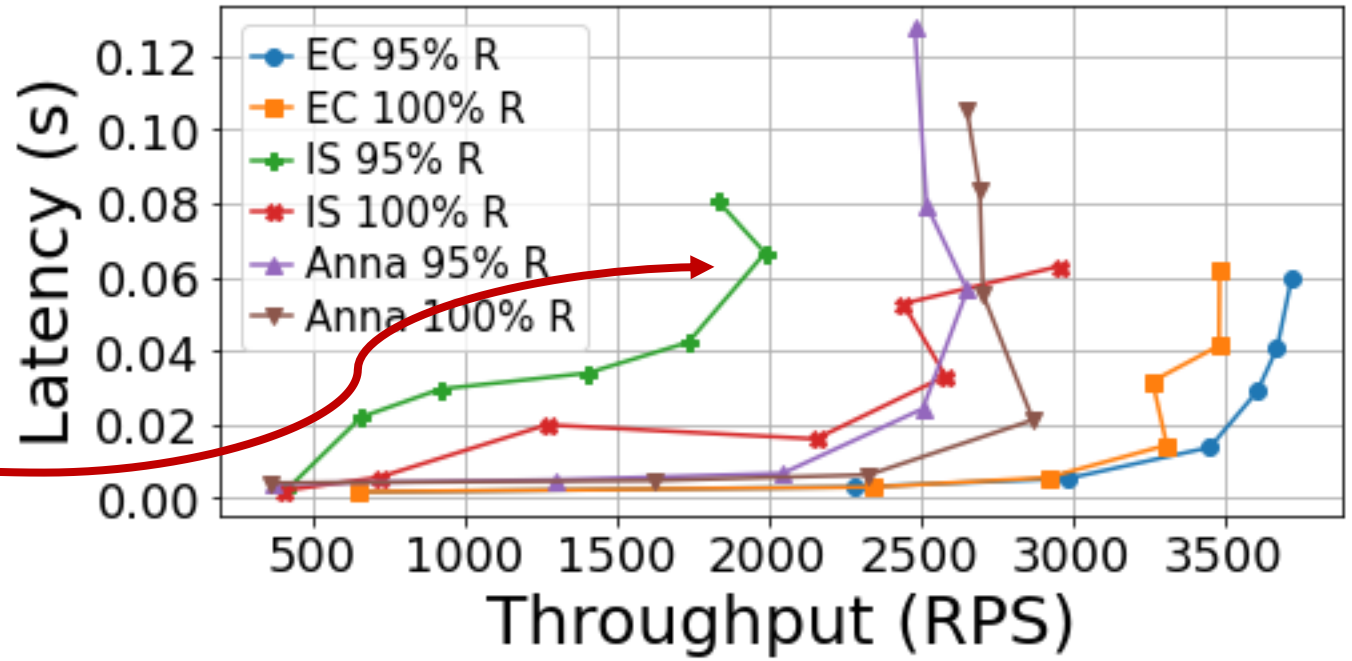
- Stress-test on **ElastiPut/$** for fixed-size objects
- Bandwidth (YCSB driver): fixed **50** Gbps
- Bandwidth (EC*, Anna*): maximum **30** Gbps



* EC: ElastiCache for Redis
* Anna: A KVS for Any Scale, IEEE Transactions on Knowledge and Data Engineering, 2021

YCSB microbenchmarking: 1MB

- Stress-test on **ElastiPut/$** for fixed-size objects
- Bandwidth (YCSB driver): fixed **50** Gbps
- Bandwidth (EC*, Anna*): maximum **30** Gbps

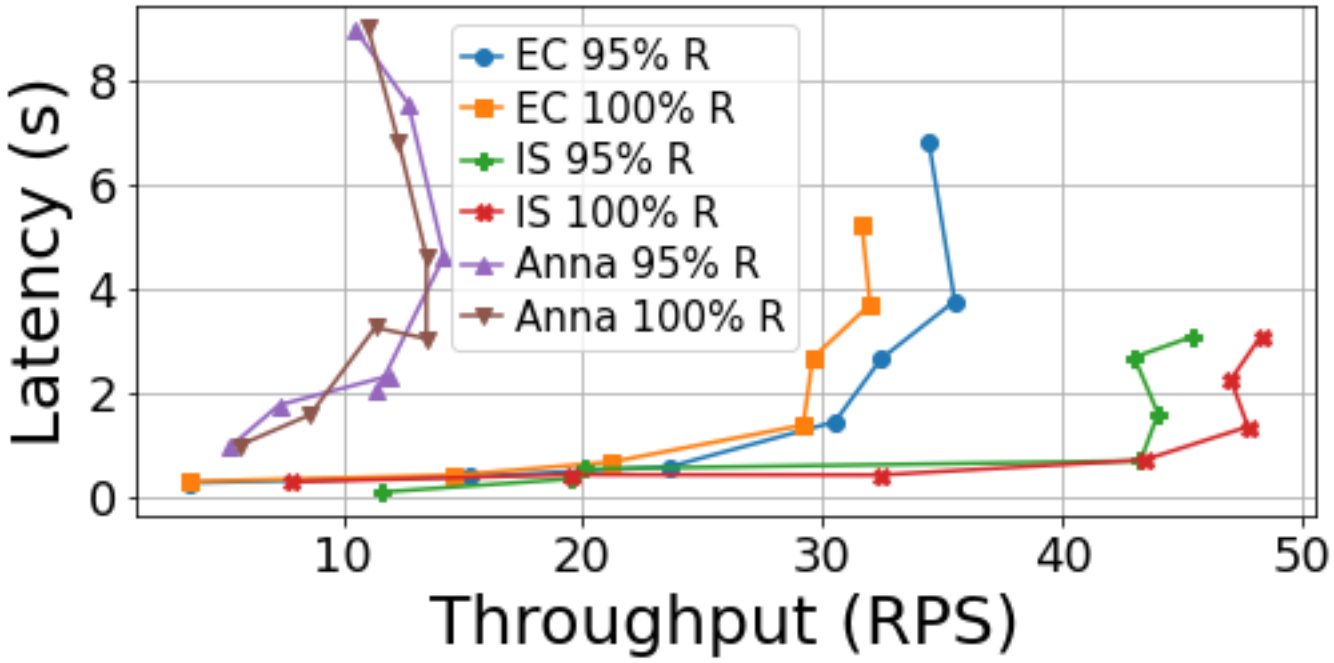


Affected by function invocation delay

* EC: ElastiCache for Redis
* Anna: A KVS for Any Scale, IEEE Transactions on Knowledge and Data Engineering, 2021

YCSB microbenchmarking: 100MB

- Stress-test on **ElastiPut/$** for fixed-size objects
- Bandwidth (YCSB driver): fixed **50** Gbps
- Bandwidth (EC*, Anna*): maximum **30** Gbps



* EC: ElastiCache for Redis
* Anna: A KVS for Any Scale, IEEE Transactions on Knowledge and Data Engineering, 2021

Cost-effectiveness

- **InfiniStore**

Average 85 * 1.5GB Lambda functions (95.8% SMS Hit ratio)

- **InfiniCache**

400 * 1.5GB Lambda functions

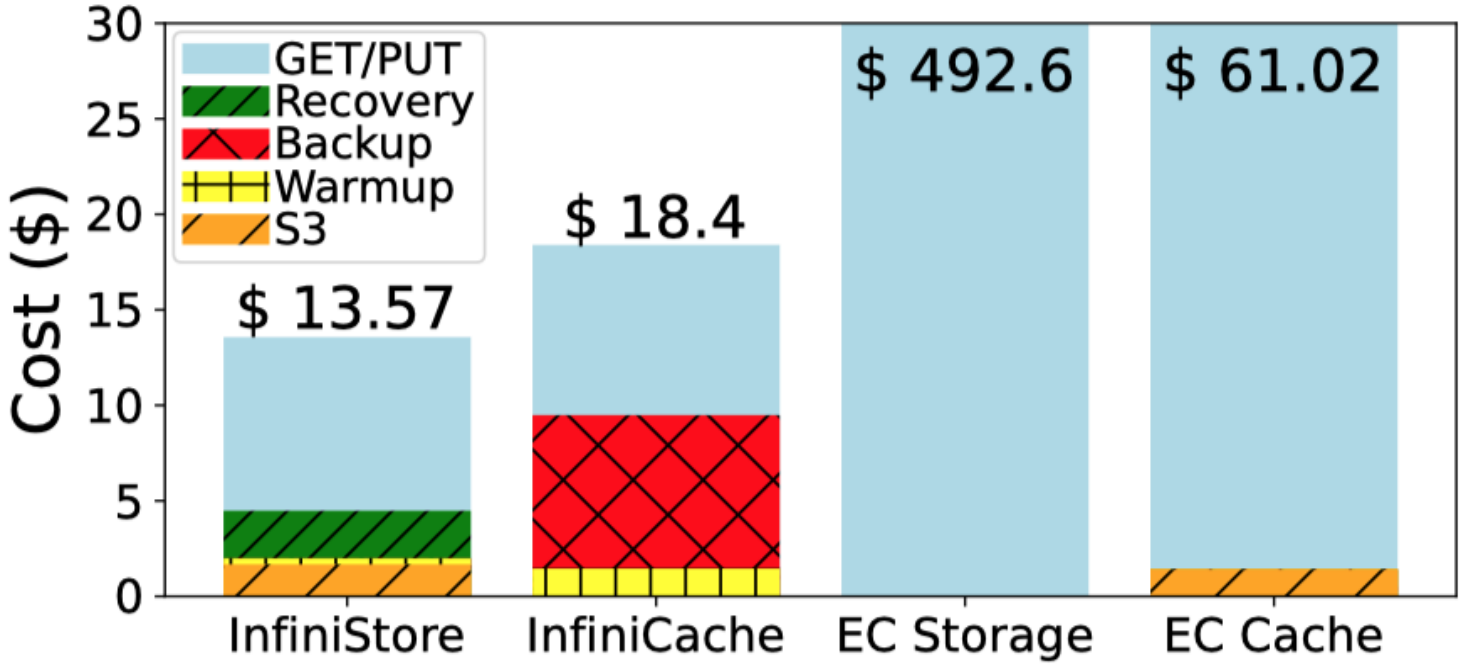
(95.4% Hit ratio)

- **ElasticCache as Storage**

12 cache.r6g.2xlarge with 633.64 GB aggregate memory

- **ElasticCache as Cache**

8 cache.m6g.large with 51.04 GB aggregate memory (93.3% Hit ratio)



Cost-effectiveness

- **InfiniCache**

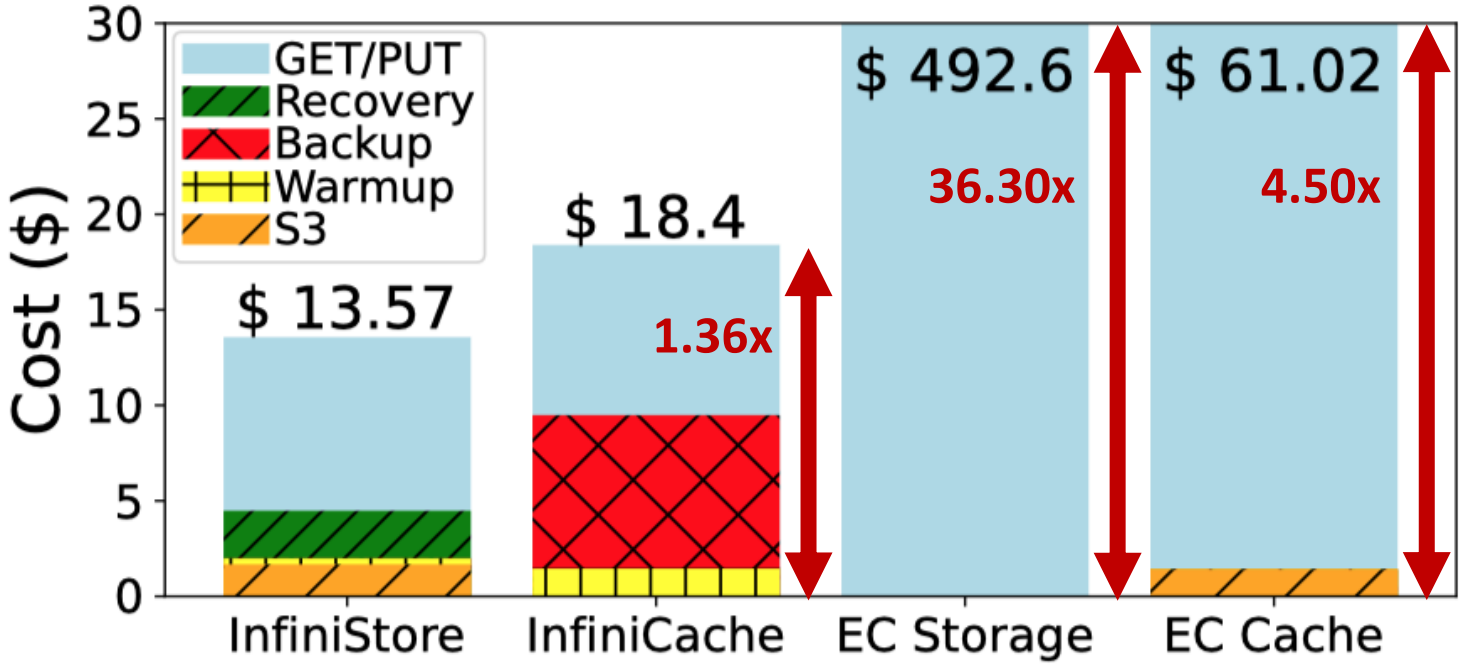
  1.36x $ cost

- **ElasticCache as Storage**

  36.30x $ cost

- **ElasticCache as Cache**

  4.50x $ cost



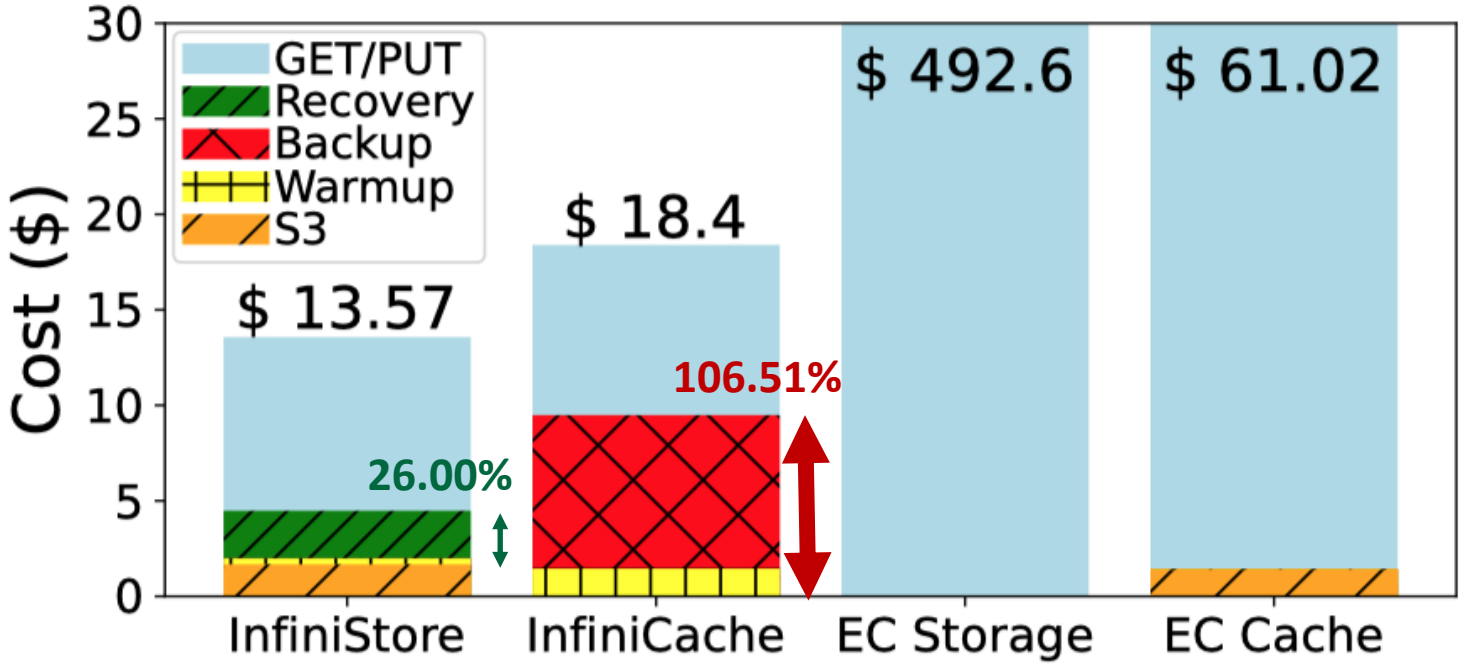
Pay-per-access

- **InfiniCache overhead:**

  106% reqs cost

- **InfiniStore overhead:**

  26.00% reqs cost



Conclusion

- InfiniStore is a multi-layer, elastic, and cost-effective serverless cloud storage. InfiniStore achieves this goal by:
  - Using sliding-window function-memory management to achieve elasticity.
  - Using two layers of inexpensive storage with parallel recovery to achieve durability while remaining cost-effective.

Fork me on GitHub

Thank you!

Any questions?

- Contact: Jingyuan Zhang – jzhang33@gmu.edu
- Source code: https://github.com/ds2-lab/infinistore 
- The extended report is available at: https://arxiv.org/abs/2209.01496