

# Serverless Object Caching

*DS 5110: Big Data Systems (Spring 2023)*

Lecture 7c

Yue Cheng



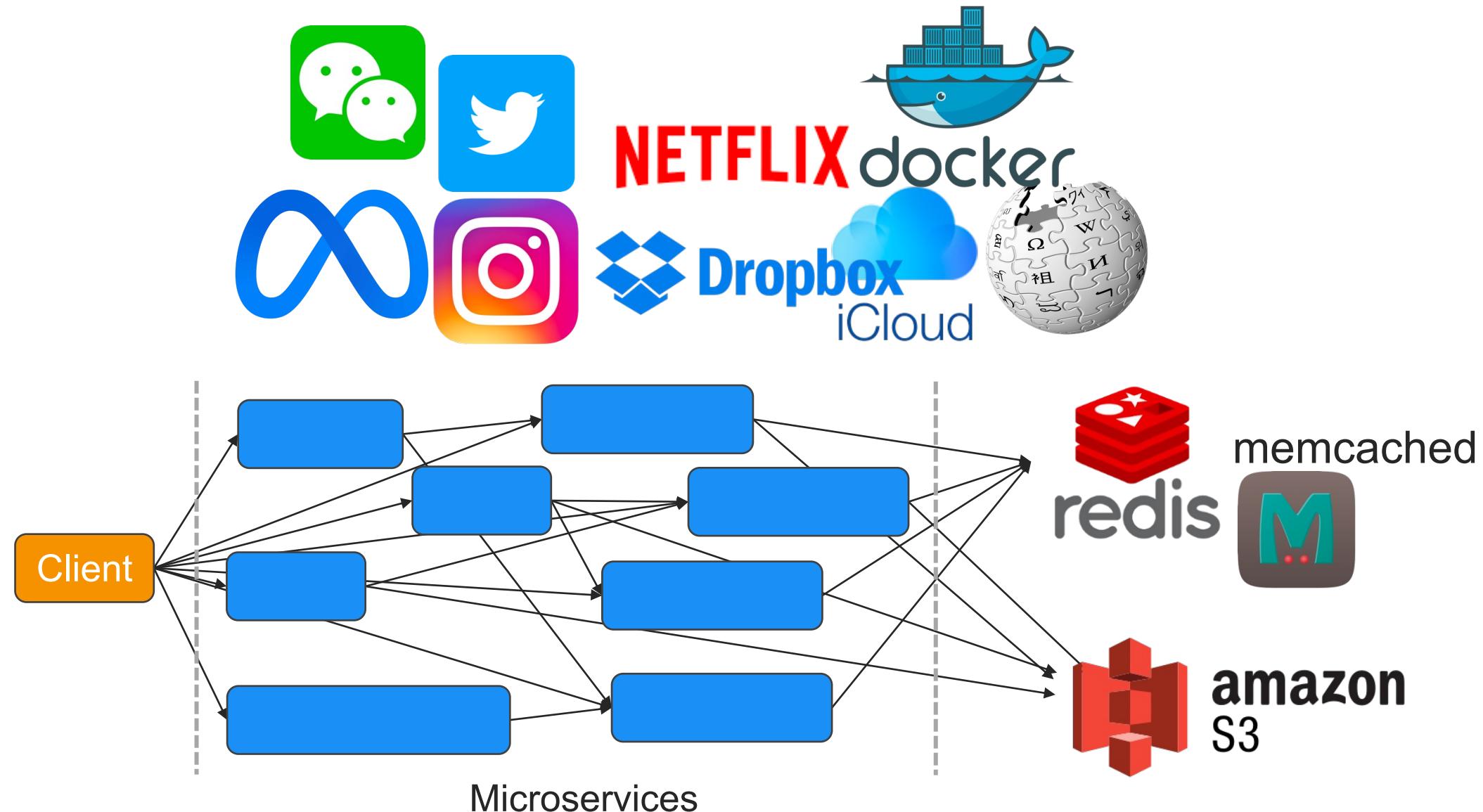
# Challenges of supporting stateful apps on FaaS

**Research Question:** Is FaaS poorly suited for stateful applications because these applications share state?

Case studies:

1. [Programming model] How to design FaaS-centric parallel computing to enable easy programming of 10,000 CPU cores and 15,000 GBs of RAM?
2. [Data storage] How to exploit FaaS **elasticity and pay-per-use** to reduce the \$\$ cost by **100X**?

# Internet-scale web apps are storage-intensive



# Example app: IBM Cloud Container Registry workloads

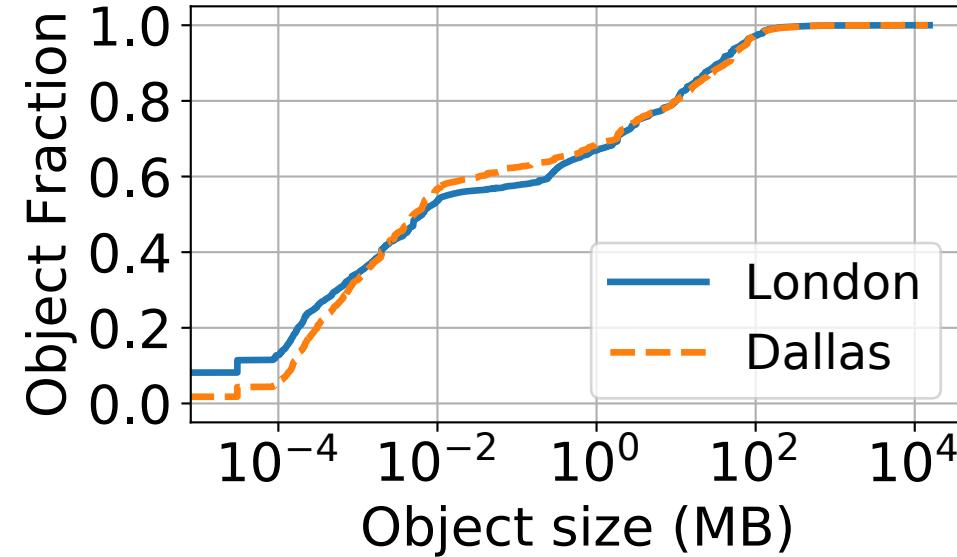
- Collected the workload traces of IBM Cloud Container Registry service for a duration of 75 days across seven datacenters in 2017
- Selected datacenters: Dallas & London

# Example app: IBM Cloud Container Registry workloads

- Object size distribution
- Large objects' reuse patterns
- Storage footprint

# Example app: IBM Cloud Container Registry workloads

- Object size distribution
- Large objects' reuse patterns
- Storage footprint

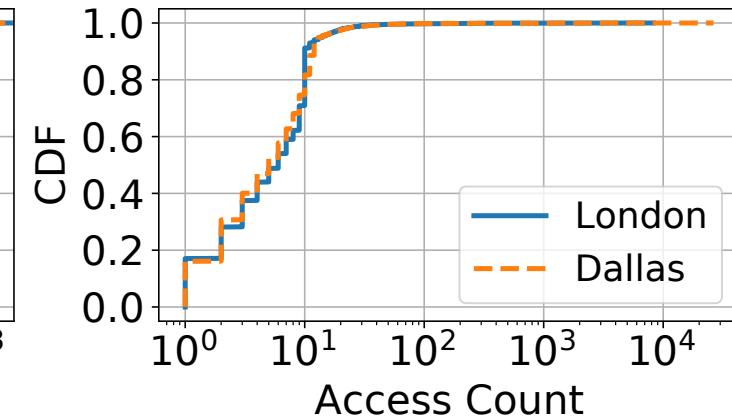
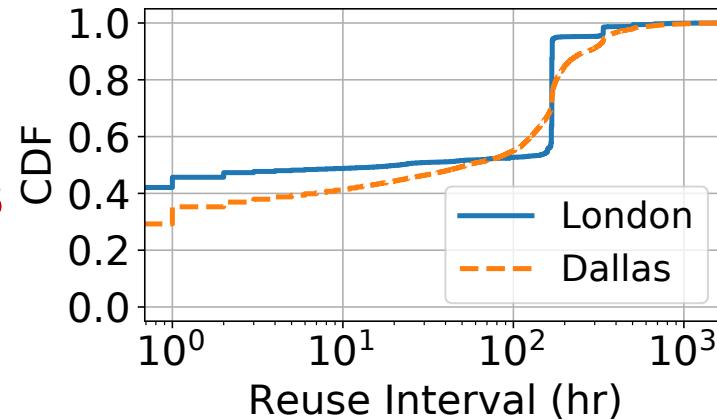


Extreme variability in object sizes:

- Object sizes span over 9 orders of magnitude
- 20% of objects > 10MB

# Example app: IBM Cloud Container Registry workloads

- Object size distribution
- Large objects' reuse patterns
- Storage footprint

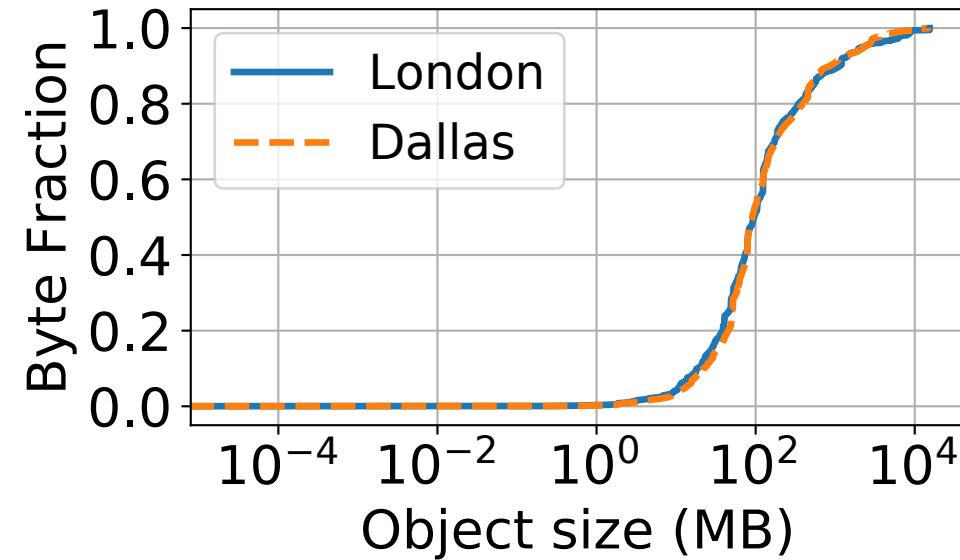


Caching large objects is beneficial:

- > 30% large object being accessed 10+ times
- Around 45% of them get reused within 1 hour

# Example app: IBM Cloud Container Registry workloads

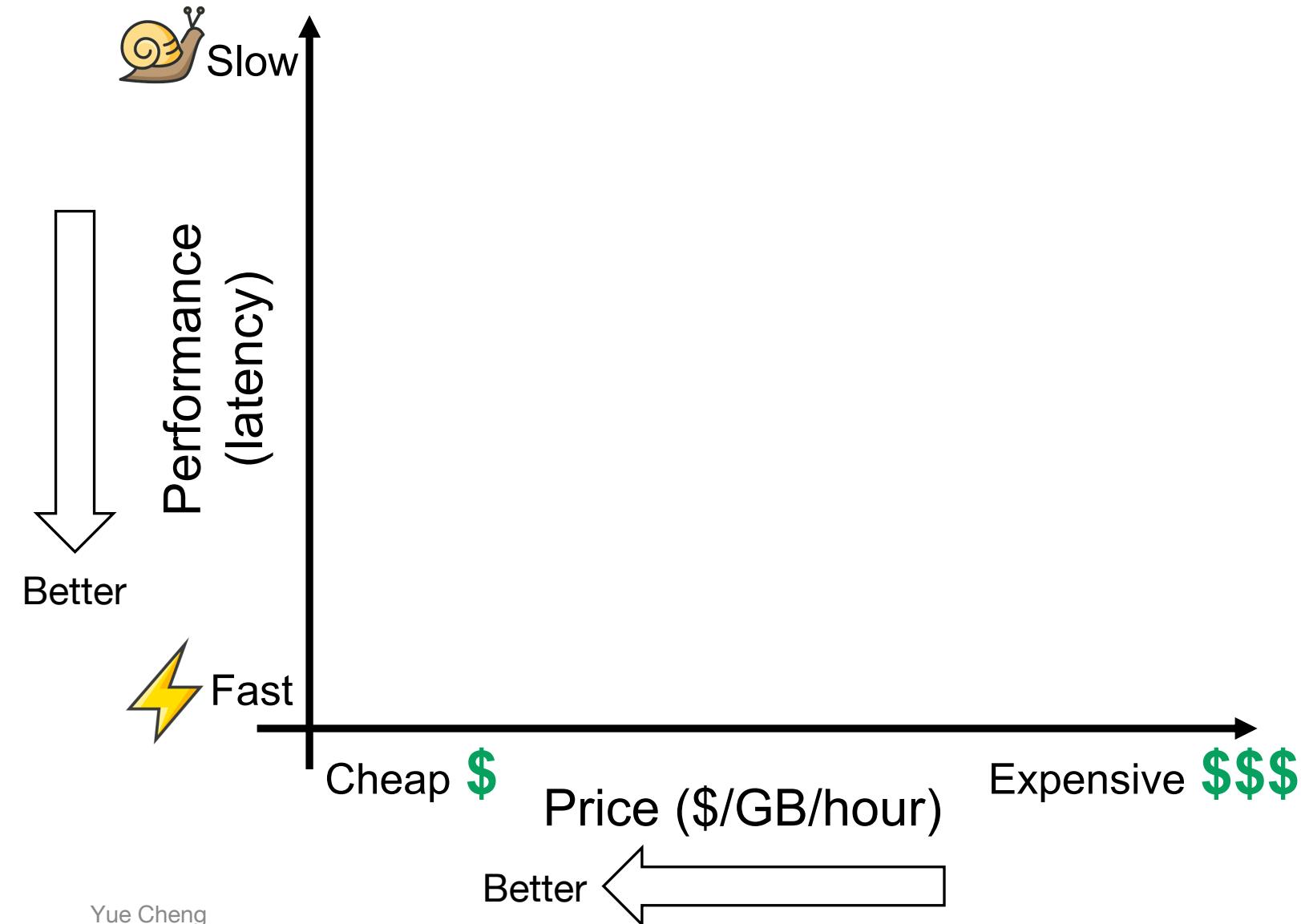
- Object size distribution
- Large objects' reuse patterns
- Storage footprint



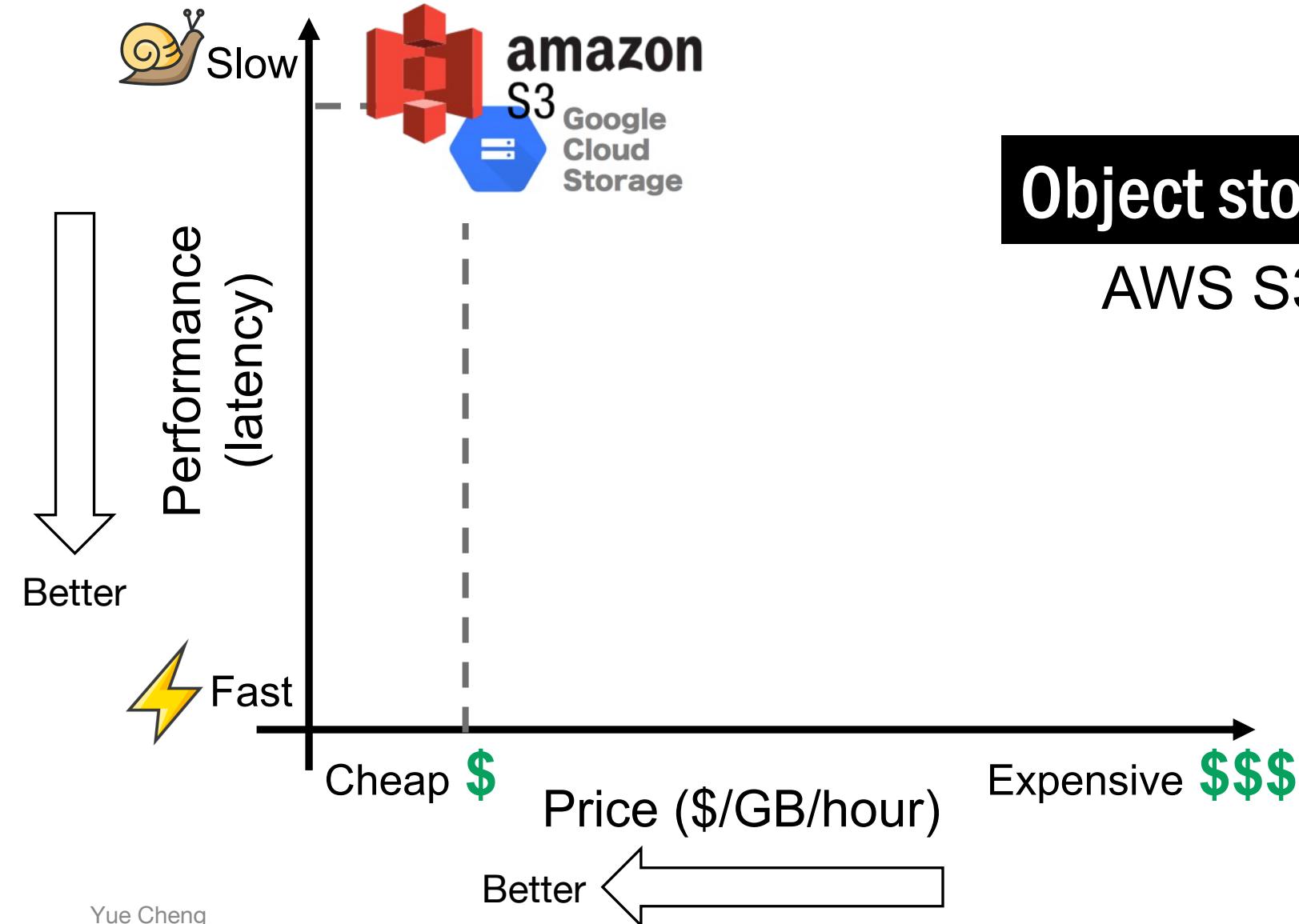
Extreme tension between small and large objects:

- Large objects (>10MB) occupy 95% storage footprint

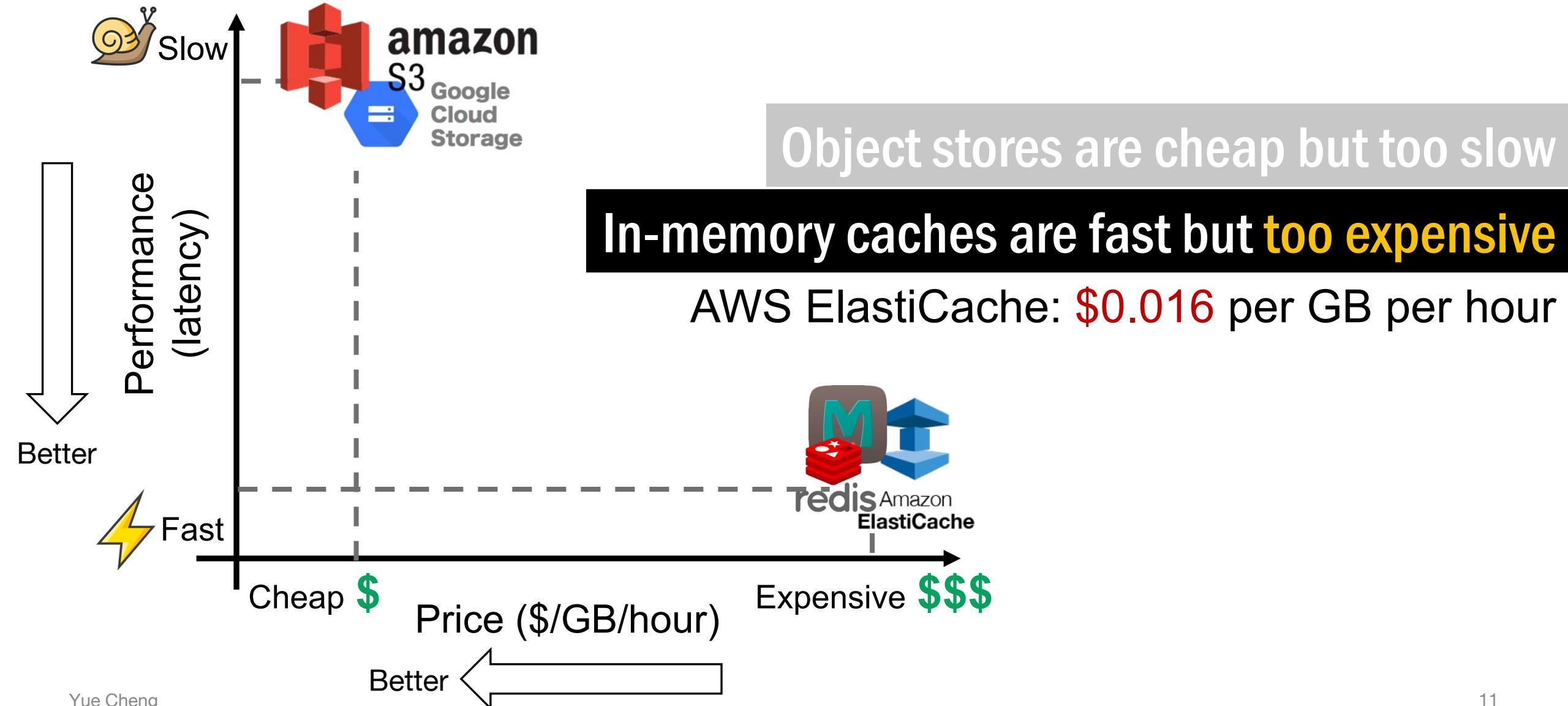
# Today's cloud storage landscape



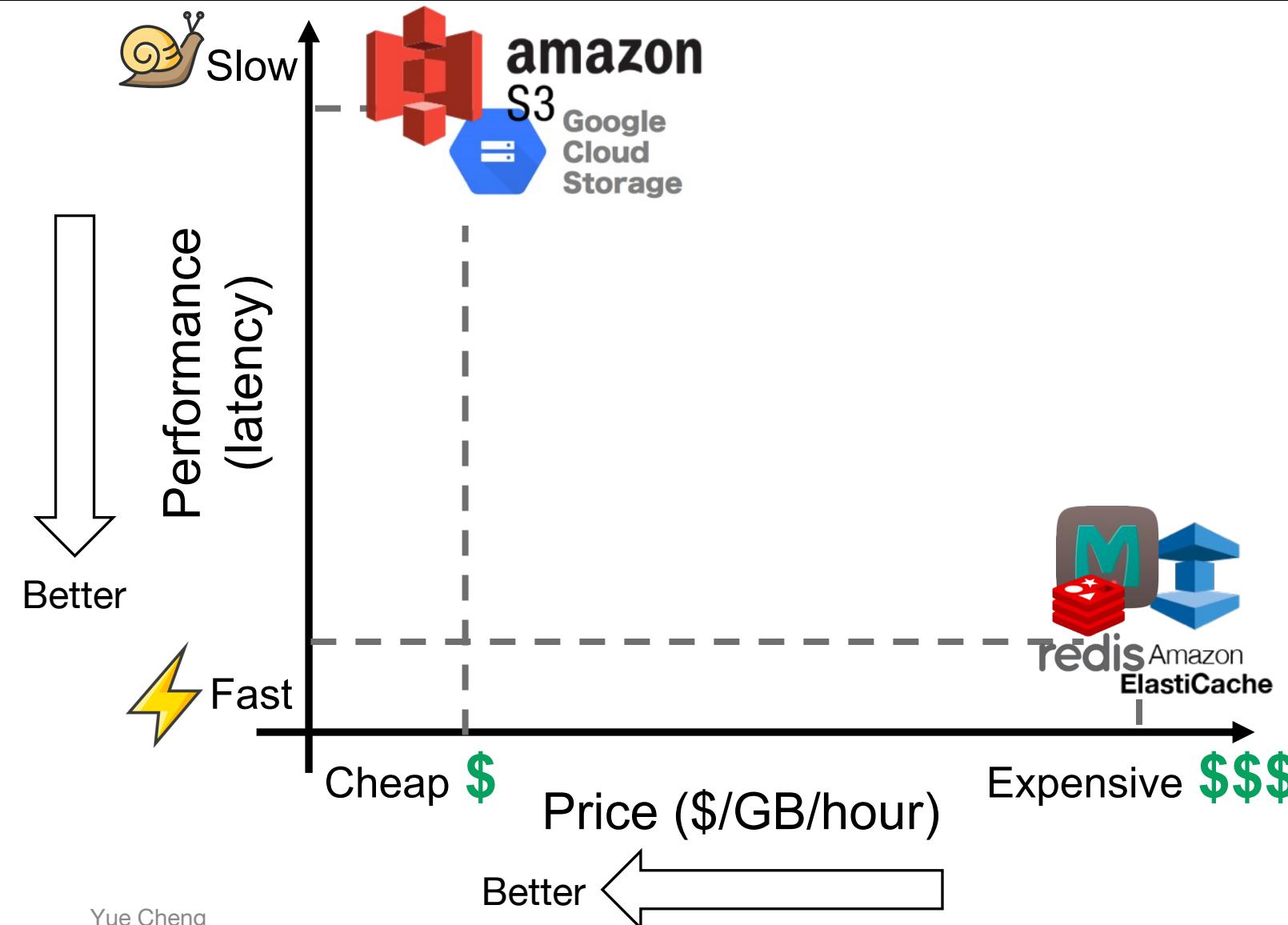
# Today's cloud storage landscape



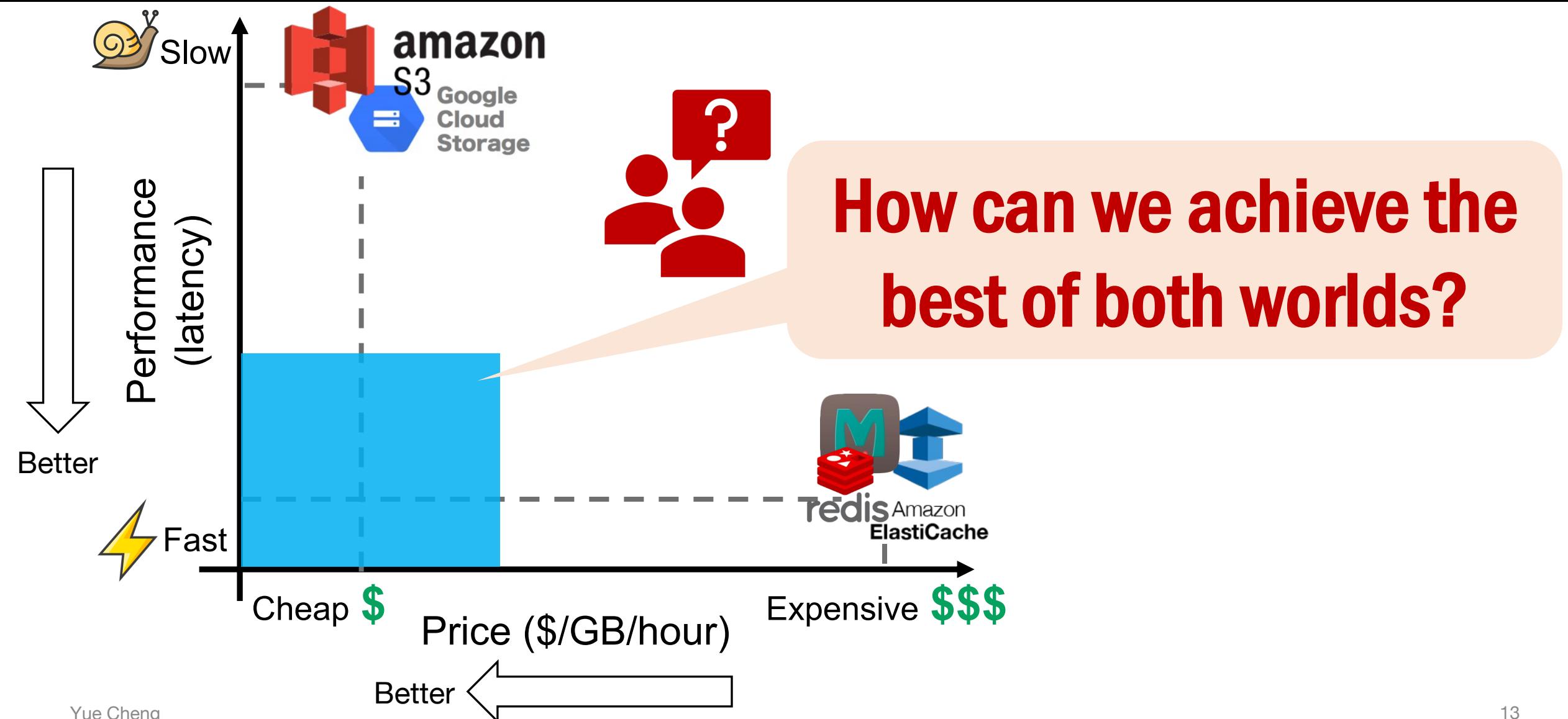
# Today's cloud storage landscape



- **Caching both small and large objects is challenging**
- **Existing solutions either too slow or too expensive**



- Caching both small and large objects is challenging
- Existing solutions either too slow or too expensive



# InfiniCache: A cost-effective and high-performance in-memory caching system built atop FaaS

- **Insight #1:** Serverless functions' <CPU, RAM> resources are **pay-per-use**
- **Insight #2:** Serverless providers offer “**free**” function memory caching for tenants

# InfiniCache: A cost-effective and high-performance in-memory caching system built atop FaaS

- **Insight #1:** Serverless functions' <CPU, RAM> resources are **pay-per-use** → **Cheap**
- **Insight #2:** Serverless providers offer “**free**” function memory caching for tenants → **Fast**

# Challenges to build a memory cache using serverless functions

High-level idea: Use Lambda functions to cache data objects

A strawman proposal that directly caches data objects in Lambda functions' memory may not work because of those FaaS limitations:

- **No** guaranteed data availability
- **Banned** inbound network
- **Limited** per-function resources

# Our contribution: InfiniCache

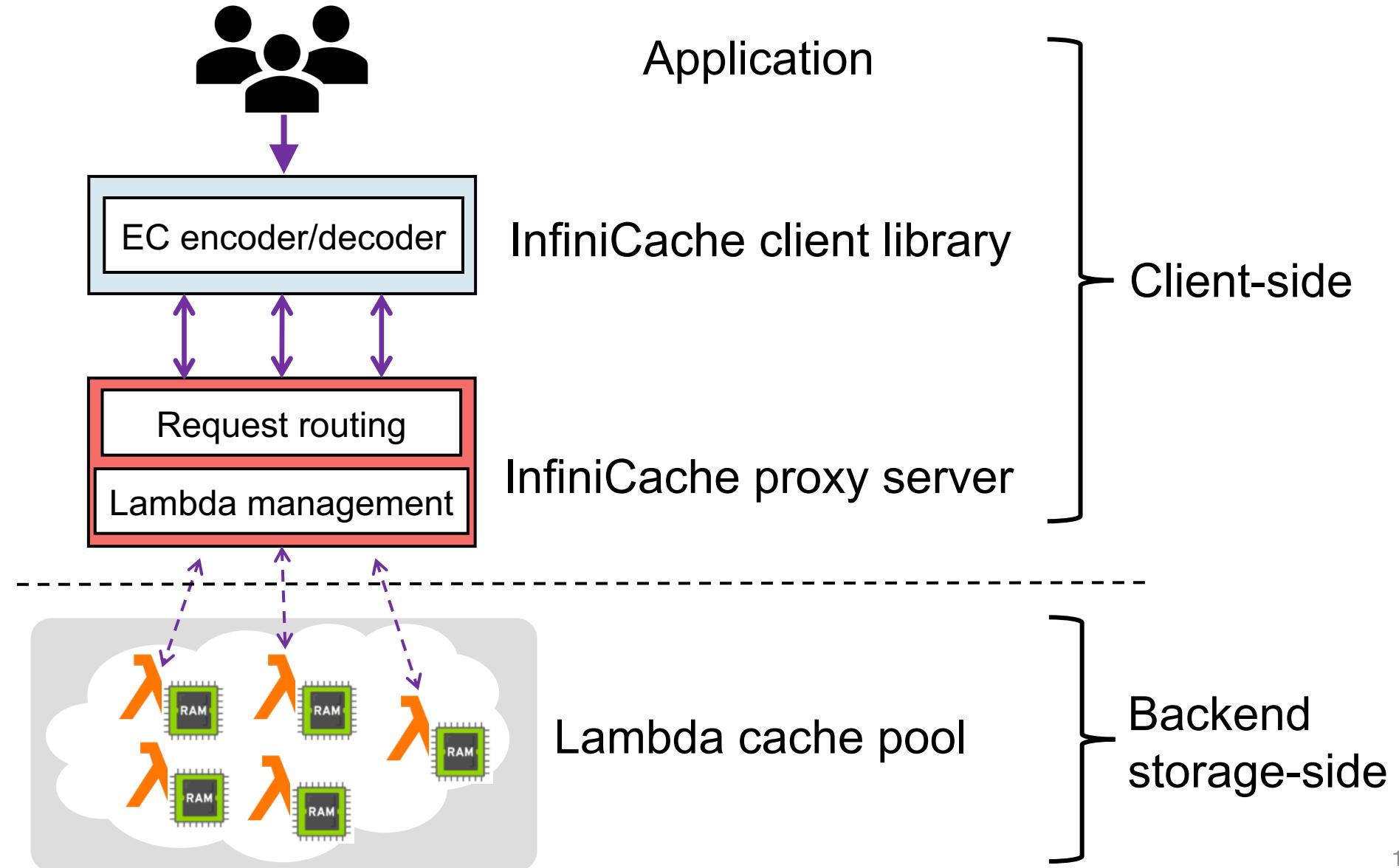
## The first in-memory caching system built atop FaaS

- InfiniCache achieves **high data availability** by using erasure coding and delta-sync periodic data backup across functions
- InfiniCache achieves **high performance** by utilizing the aggregated network bandwidth of multiple functions in parallel
- InfiniCache achieves similar performance to AWS ElastiCache while improving the cost-effectiveness by **31-96X**

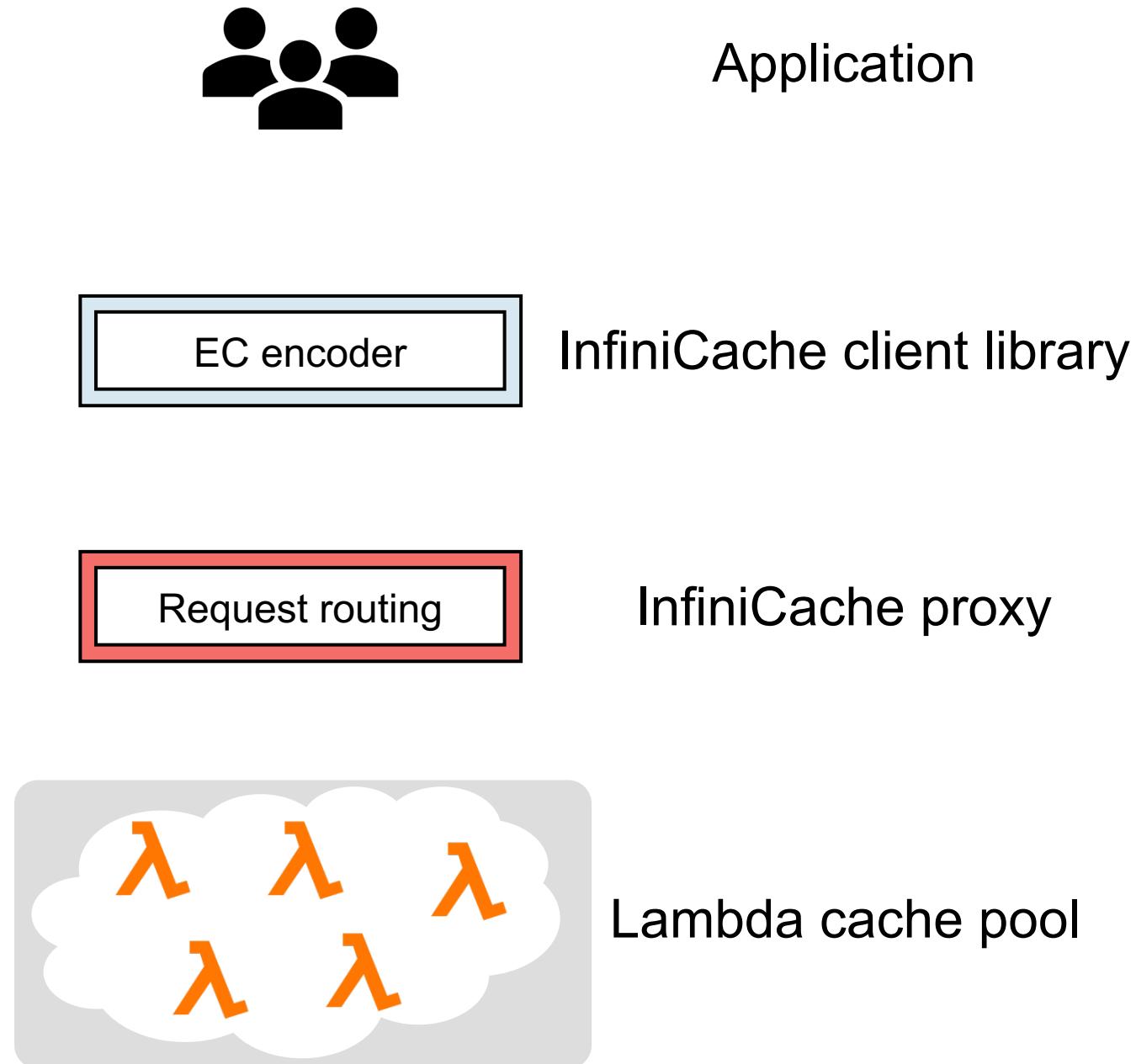
# Background of RAID and erasure coding (RS)

Switch to note

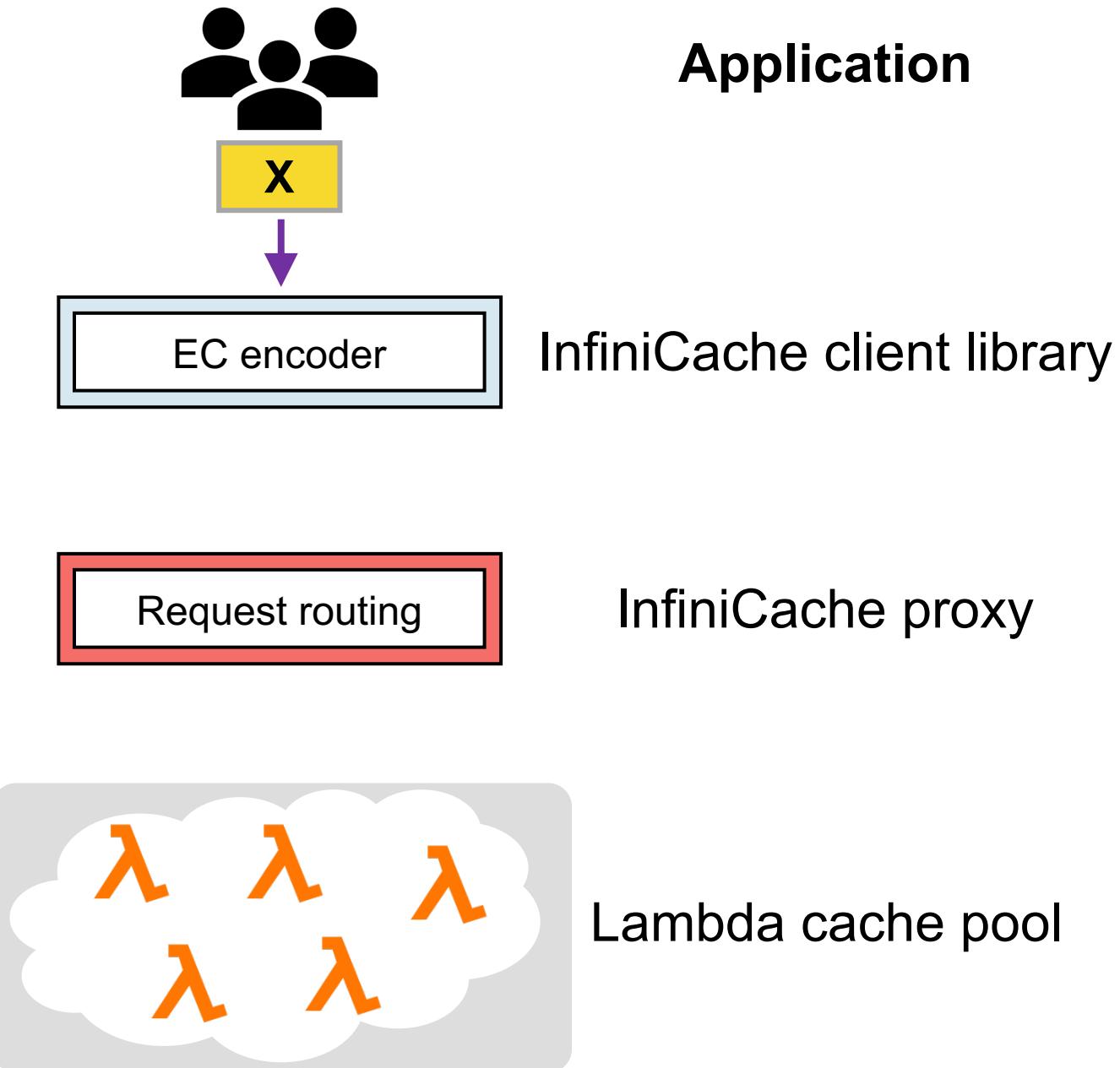
# InfiniCache bird's eye view



# InfiniCache: PUT path

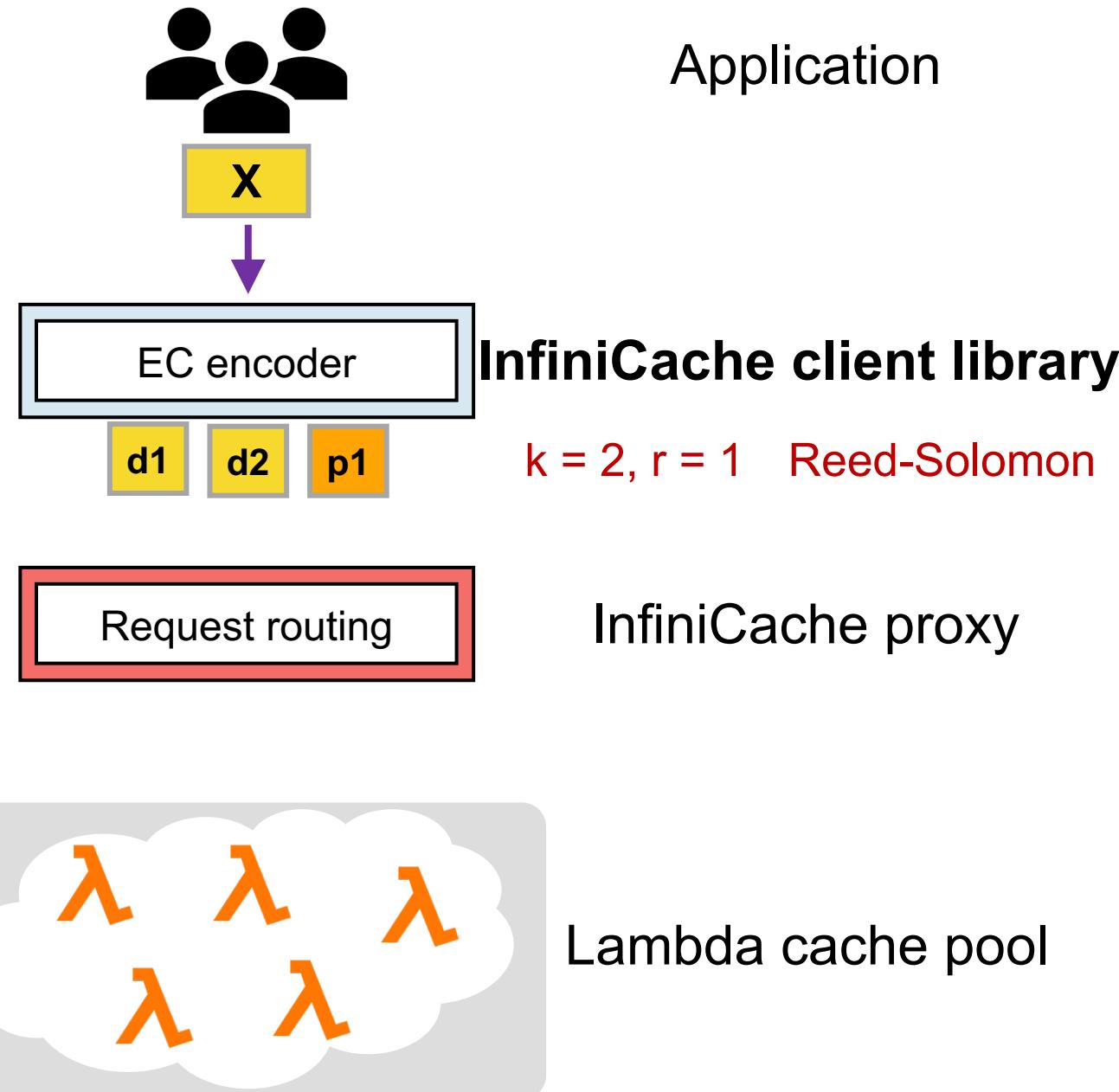


# InfiniCache: PUT path



# InfiniCache: PUT path

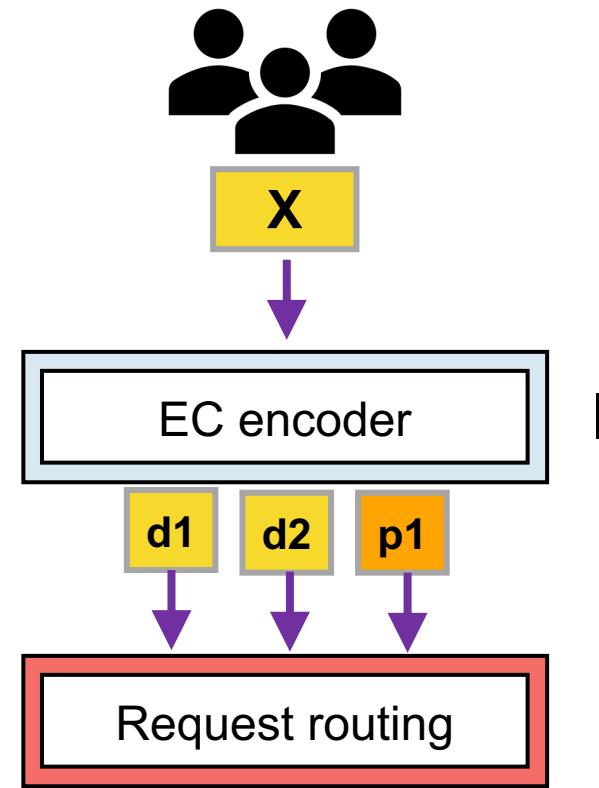
1. Object is split and encoded into  $k+r$  chunks



# InfiniCache: PUT path

1. Object is split and encoded into  $k+r$  chunks

2. Object chunks are sent to the proxy in parallel

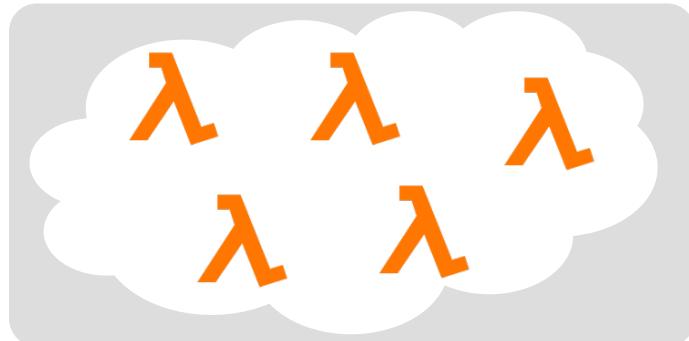


Application

InfiniCache client library

$k = 2, r = 1$  Reed-Solomon

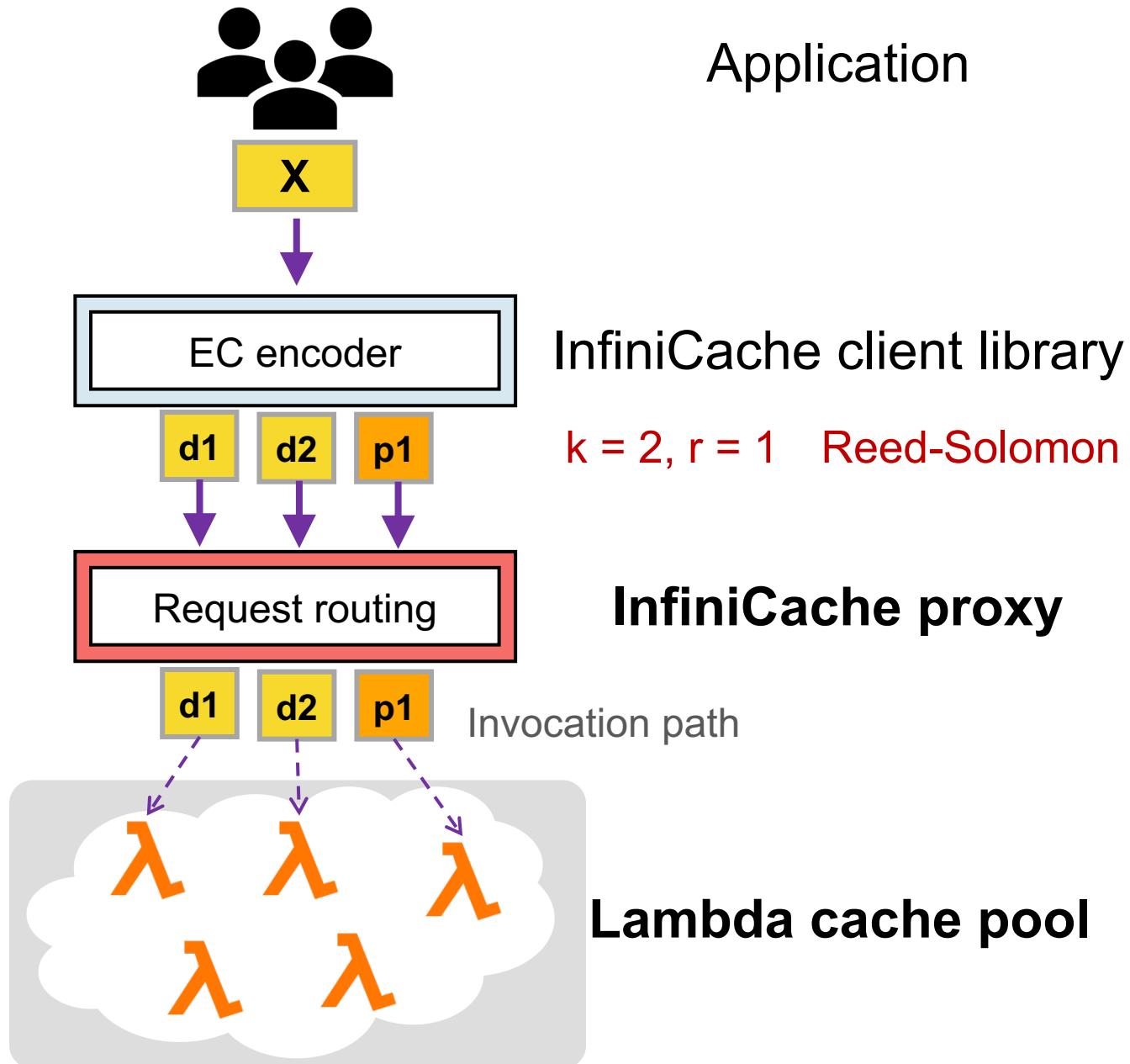
**InfiniCache proxy**



Lambda cache pool

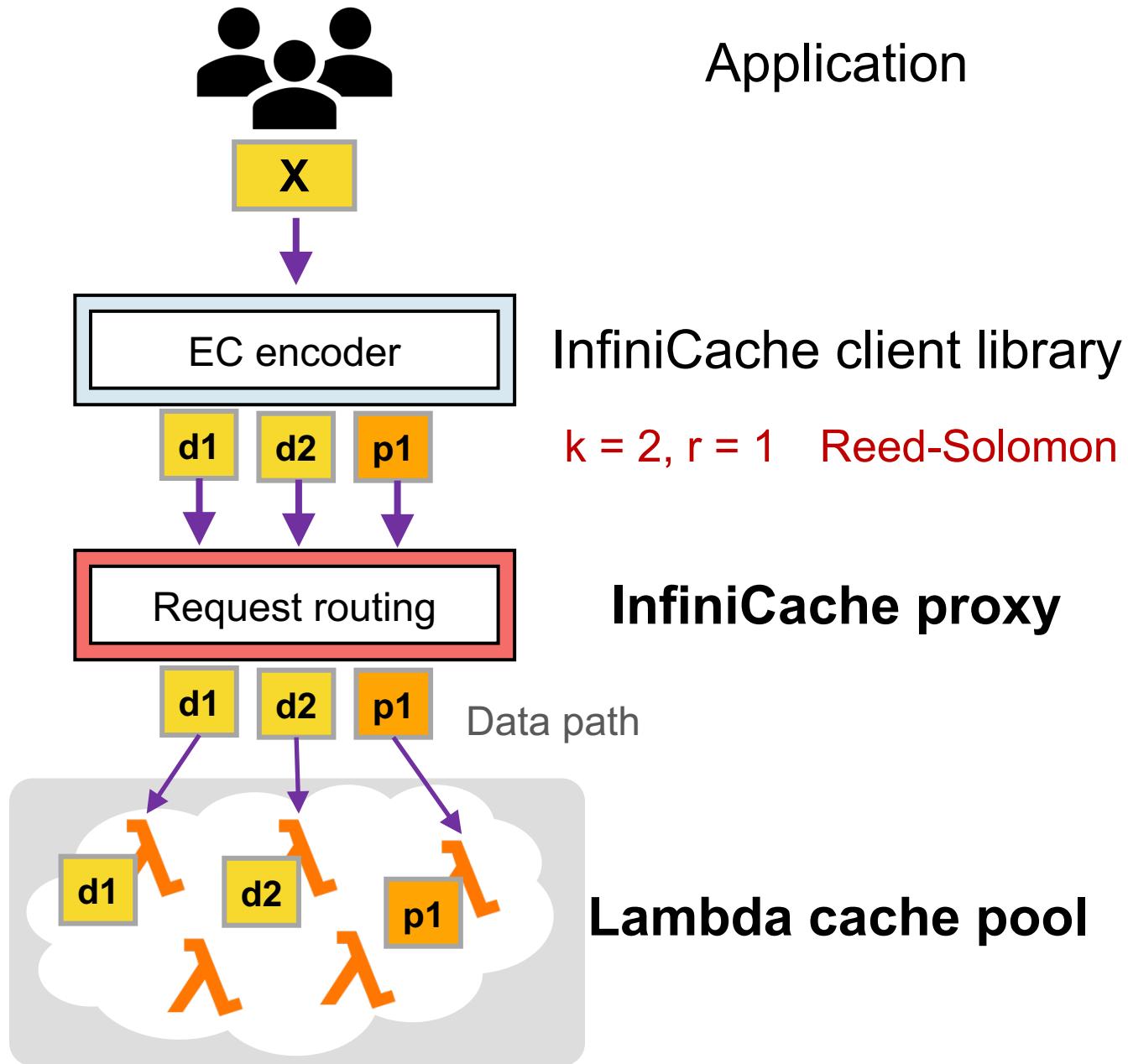
# InfiniCache: PUT path

1. Object is split and encoded into  $k+r$  chunks
2. Object chunks are sent to the proxy in parallel
3. Proxy invokes Lambda cache nodes



# InfiniCache: PUT path

1. Object is split and encoded into  $k+r$  chunks
2. Object chunks are sent to the proxy in parallel
3. Proxy invokes Lambda cache nodes
4. Proxy streams object chunks to Lambda cache nodes



# InfiniCache: GET path



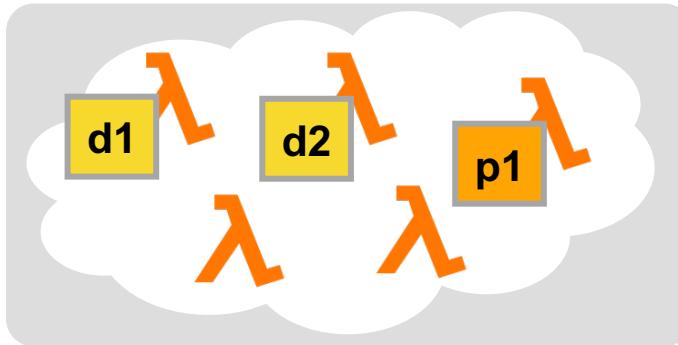
Application

EC decoder

InfiniCache client library

Request routing

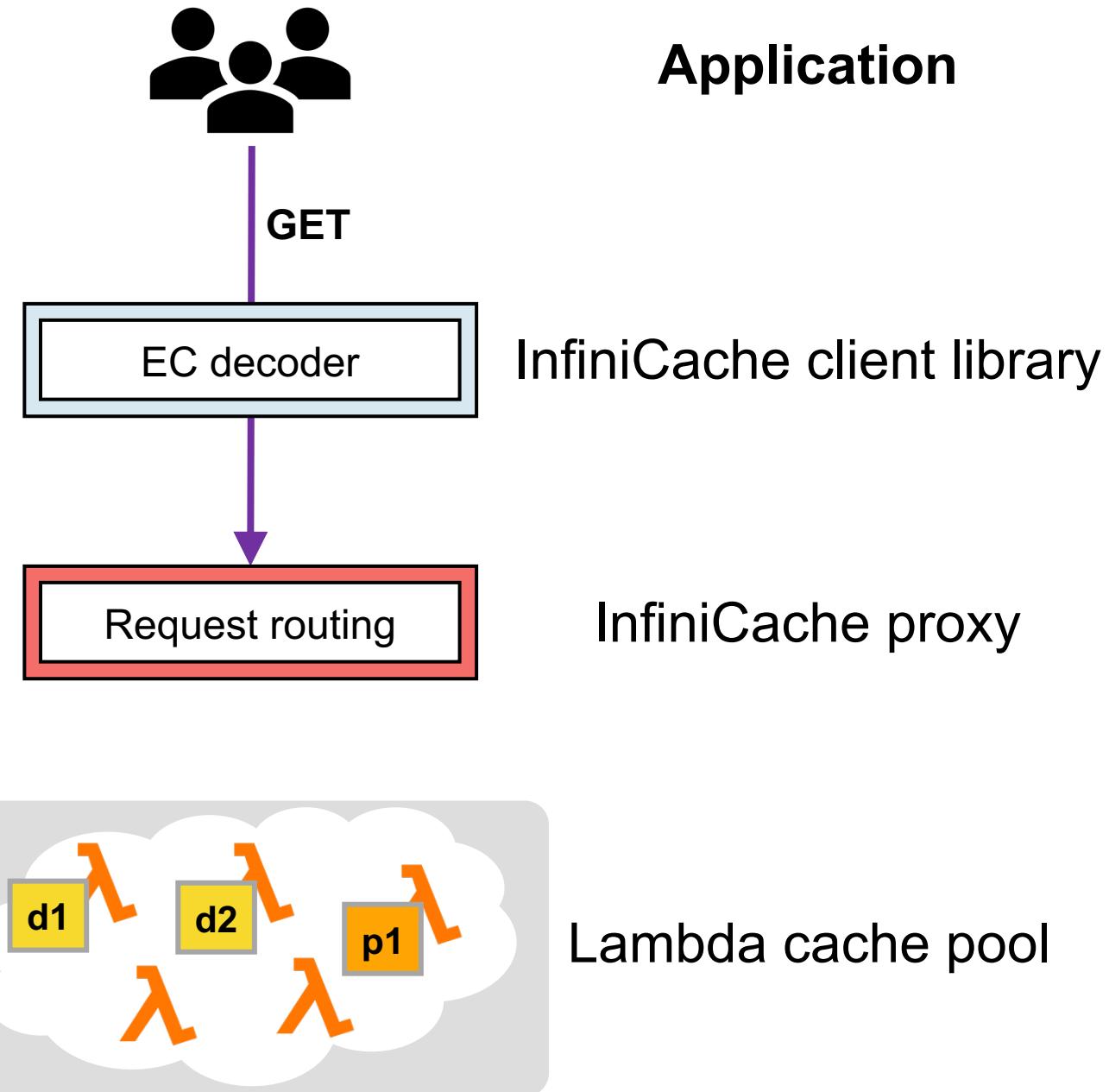
InfiniCache proxy



Lambda cache pool

# InfiniCache: GET path

1. Client sends GET request

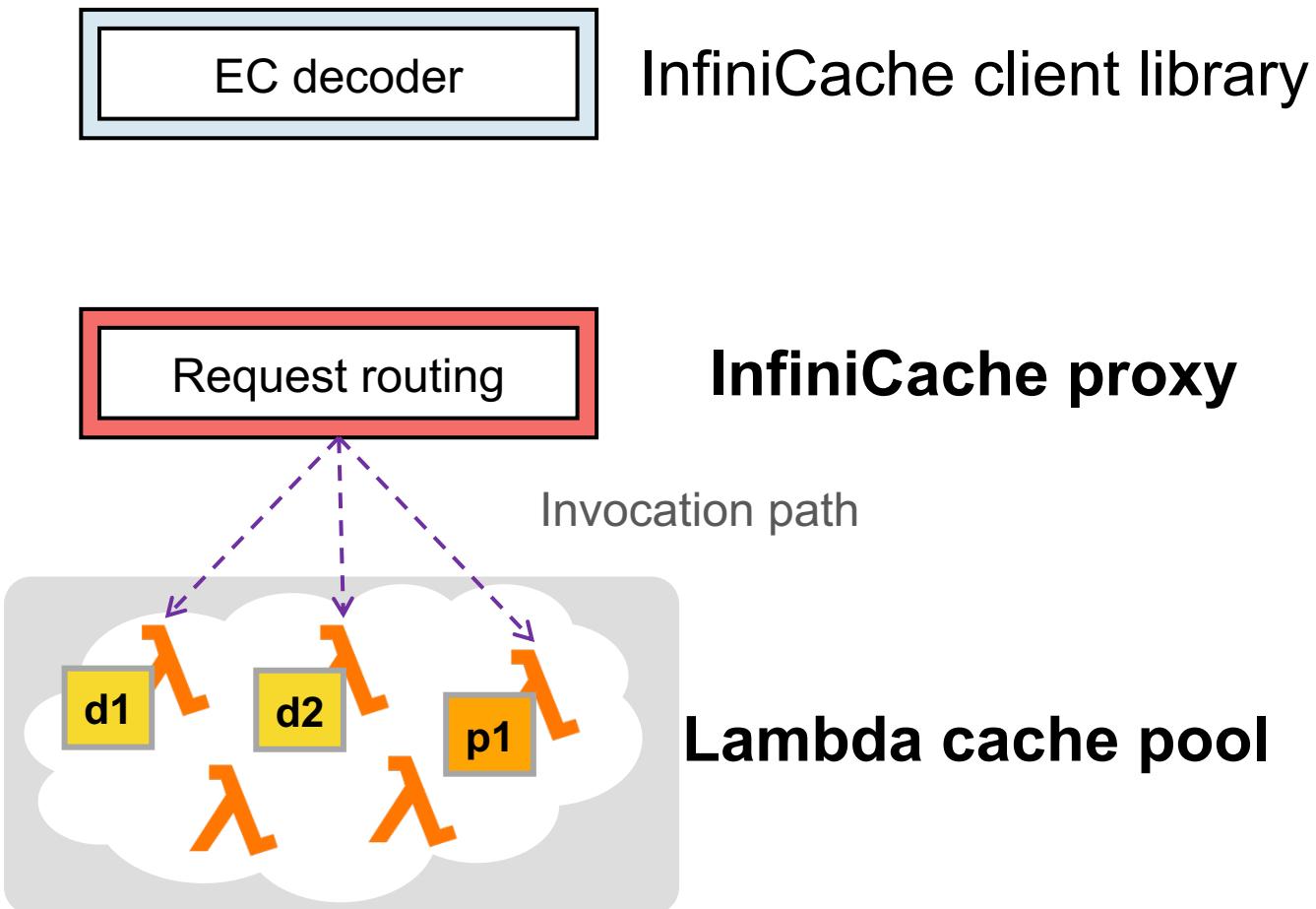


# InfiniCache: GET path



Application

1. Client sends GET request
2. Proxy invokes associated Lambda cache nodes

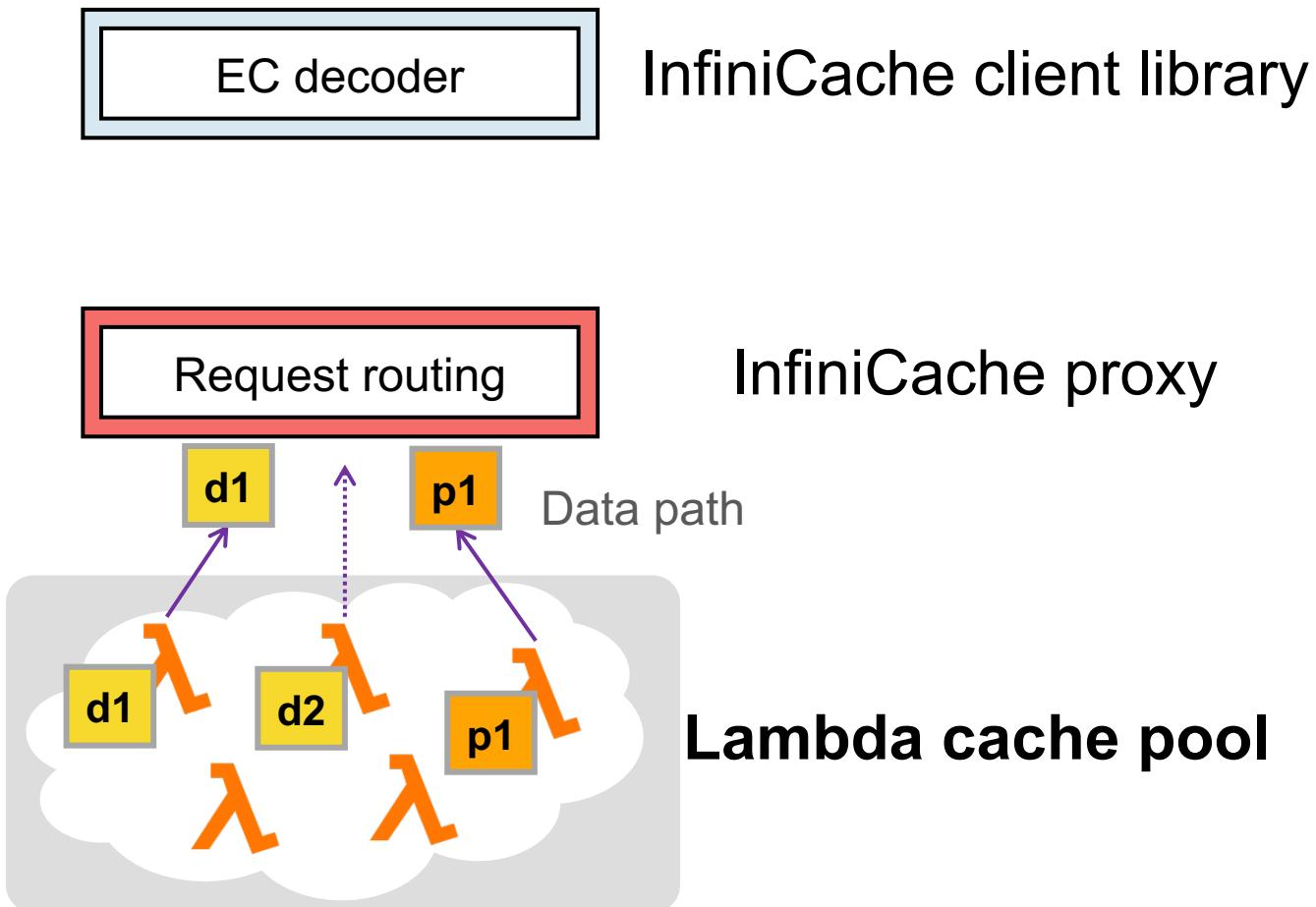


# InfiniCache: GET path



Application

1. Client sends GET request
2. Proxy invokes associated Lambda cache nodes
3. Lambda cache nodes transfer object chunks to proxy

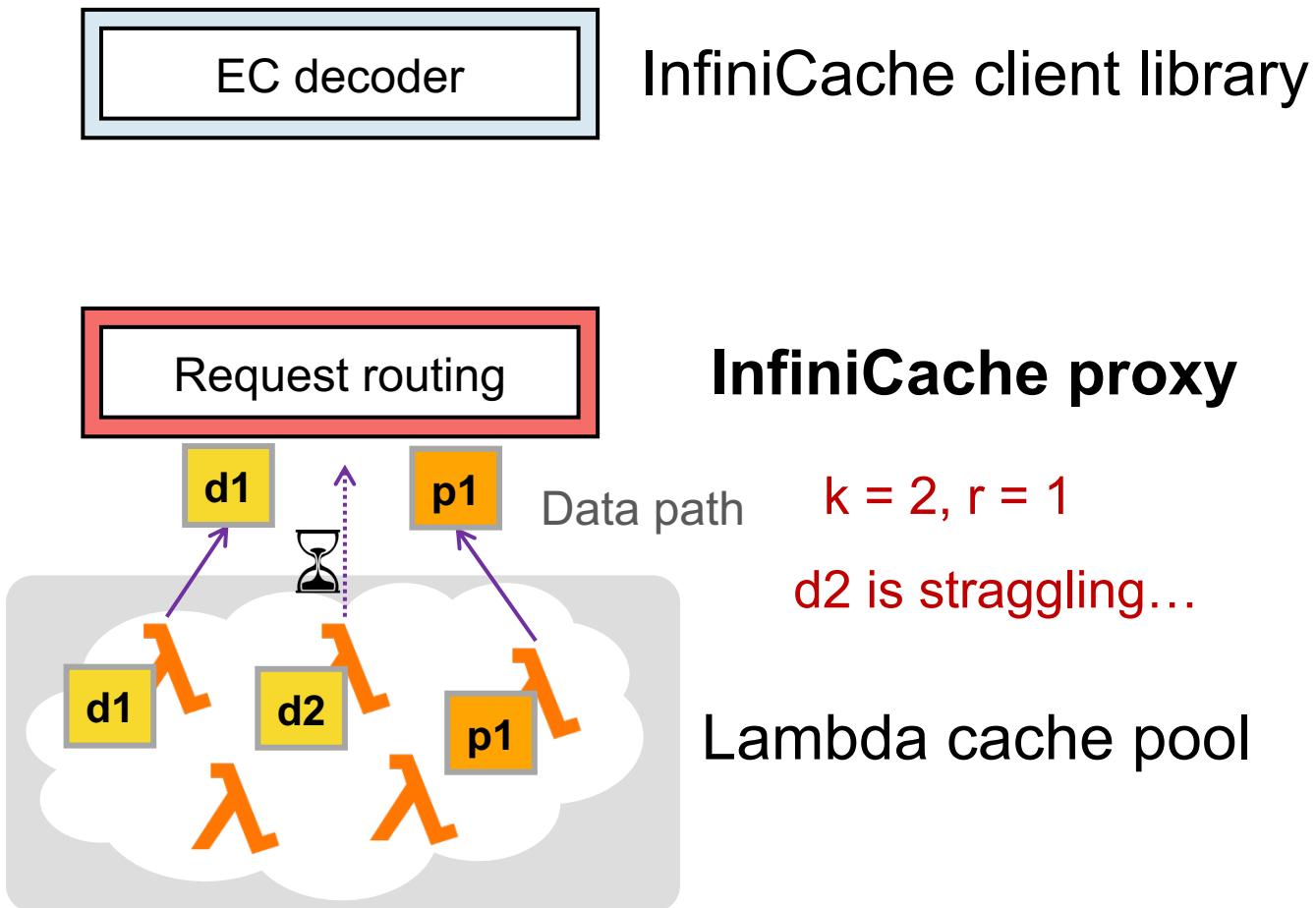


# InfiniCache: GET path



Application

1. Client sends GET request
2. Proxy invokes associated Lambda cache nodes
3. Lambda cache nodes transfer object chunks to proxy
  - **First-d optimization:** Proxy drops straggler Lambda

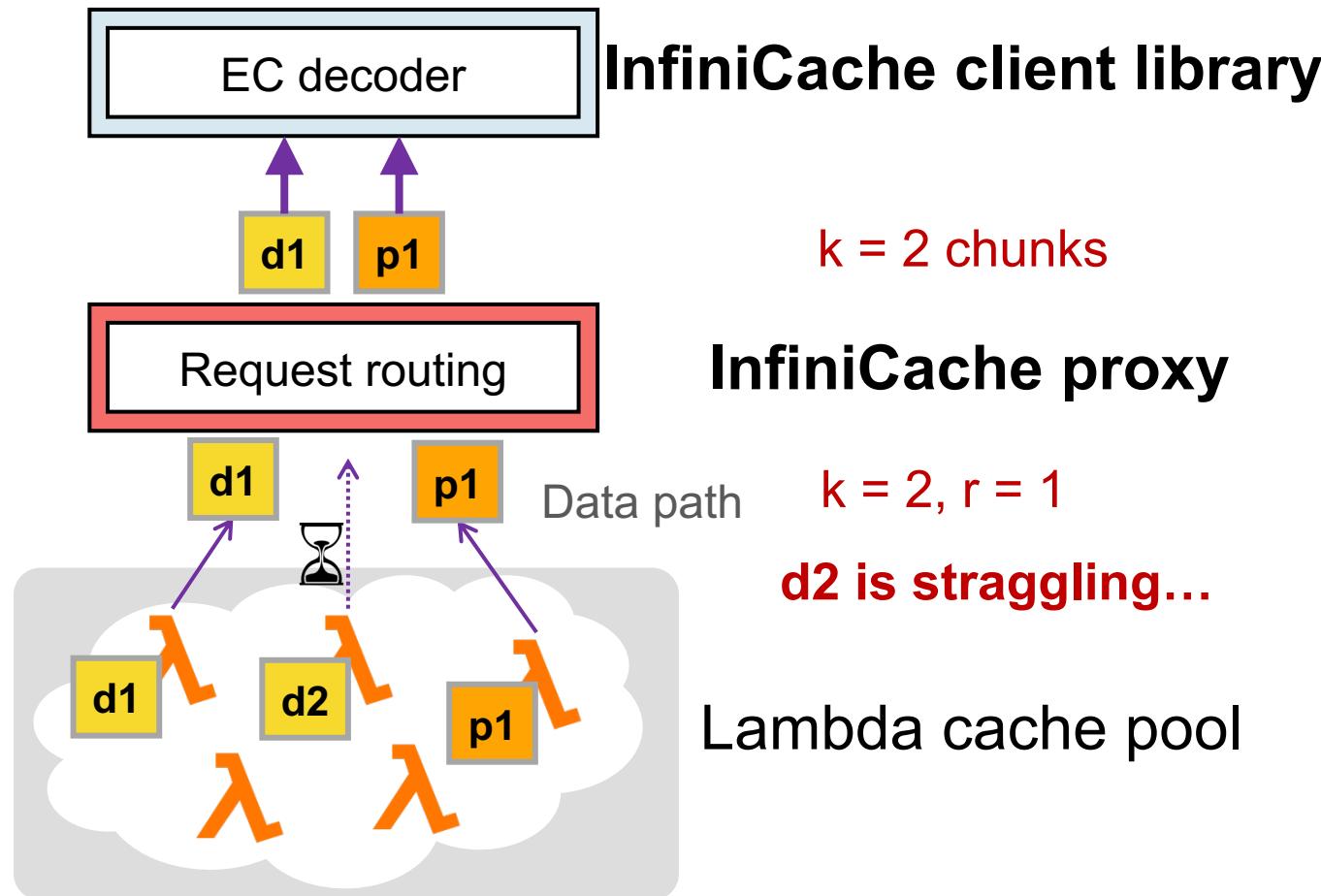


# InfiniCache: GET path



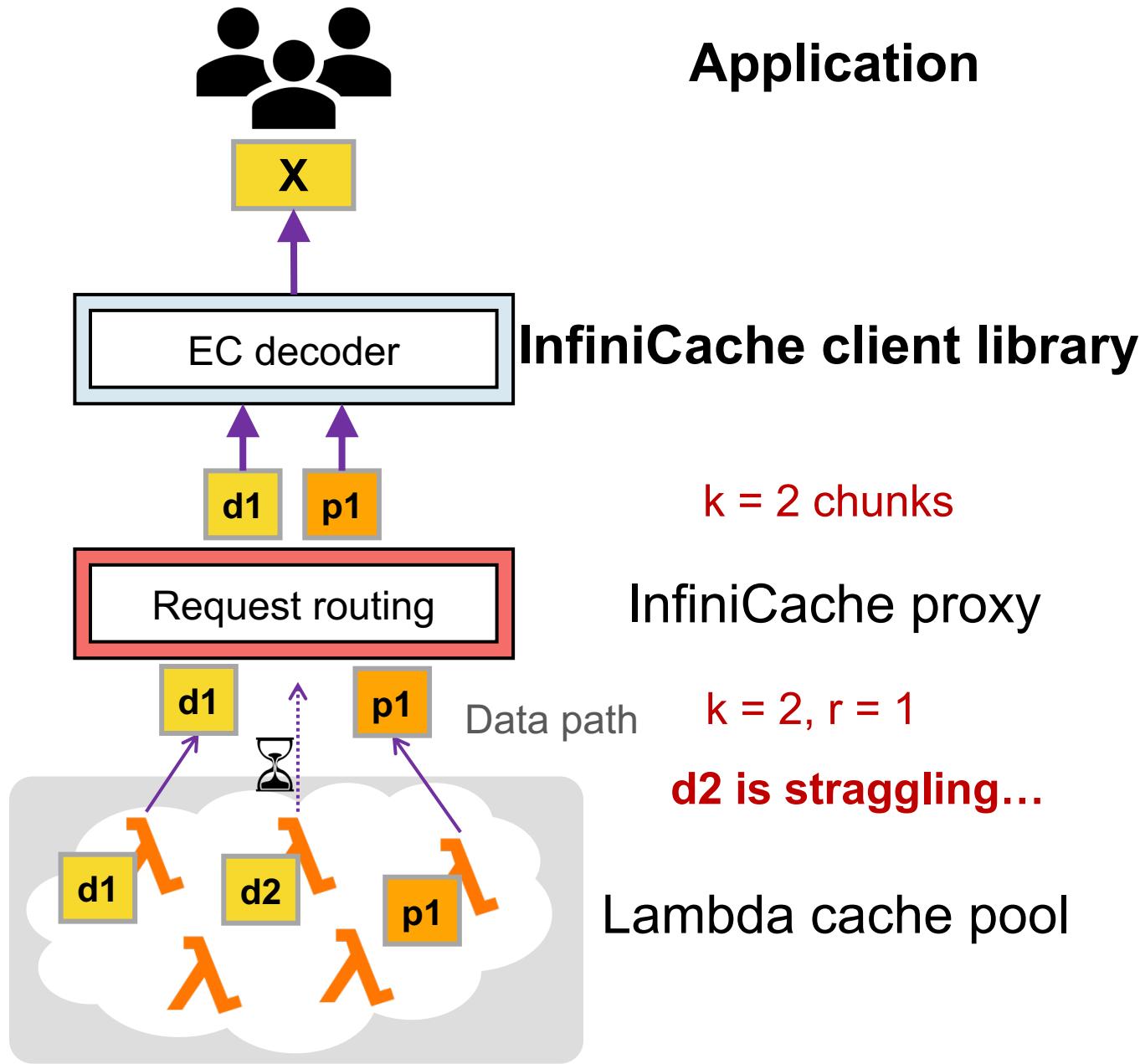
Application

1. Client sends GET request
2. Proxy invokes associated Lambda cache nodes
3. Lambda cache nodes transfer object chunks to proxy
4. Proxy streams  $k=2$  chunks in parallel to client



# InfiniCache: GET path

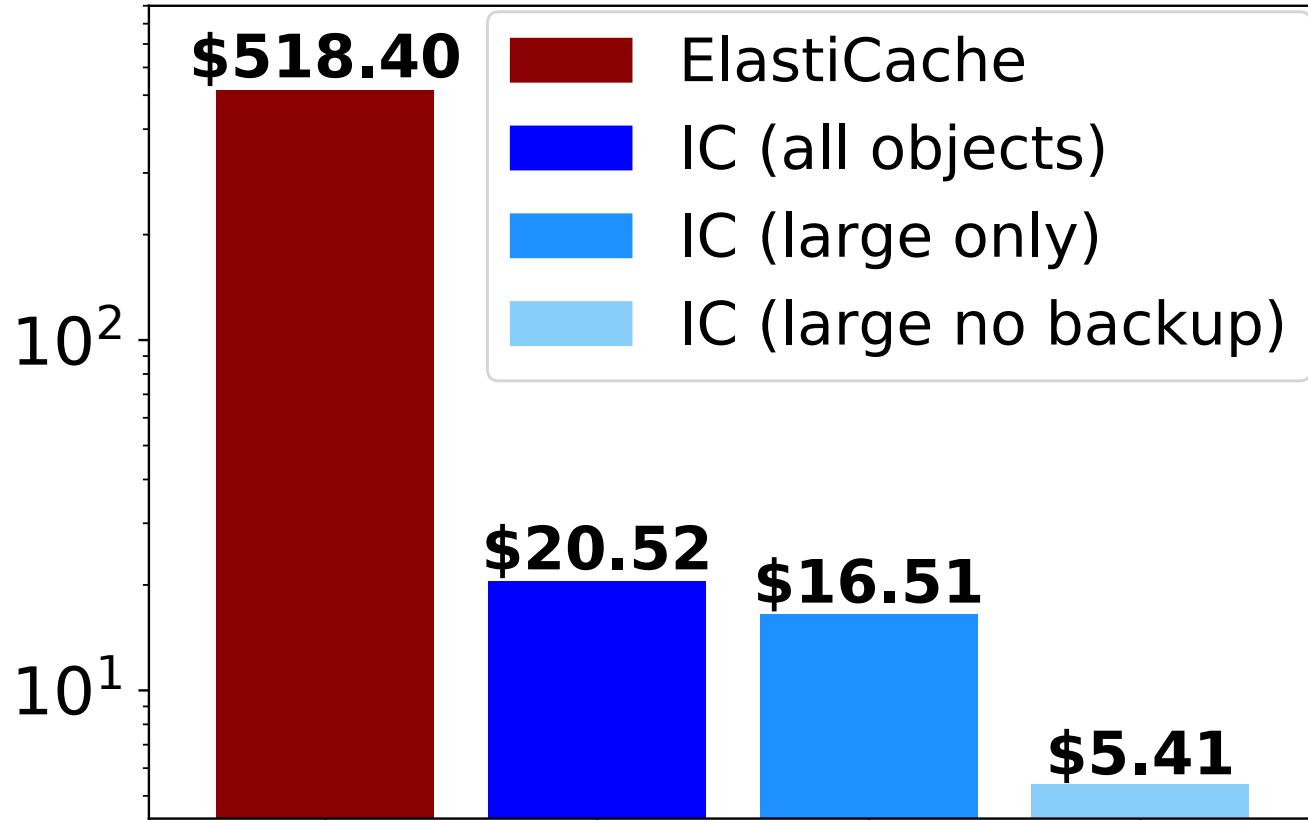
1. Client sends GET request
2. Proxy invokes associated Lambda cache nodes
3. Lambda cache nodes transfer object chunks to proxy
4. Proxy streams  $k=2$  chunks in parallel to client
5. Client library decodes  $k$  chunks



# Maximizing data availability

- Erasure-coding
- Periodic warm-up
- Smart delta-sync backup

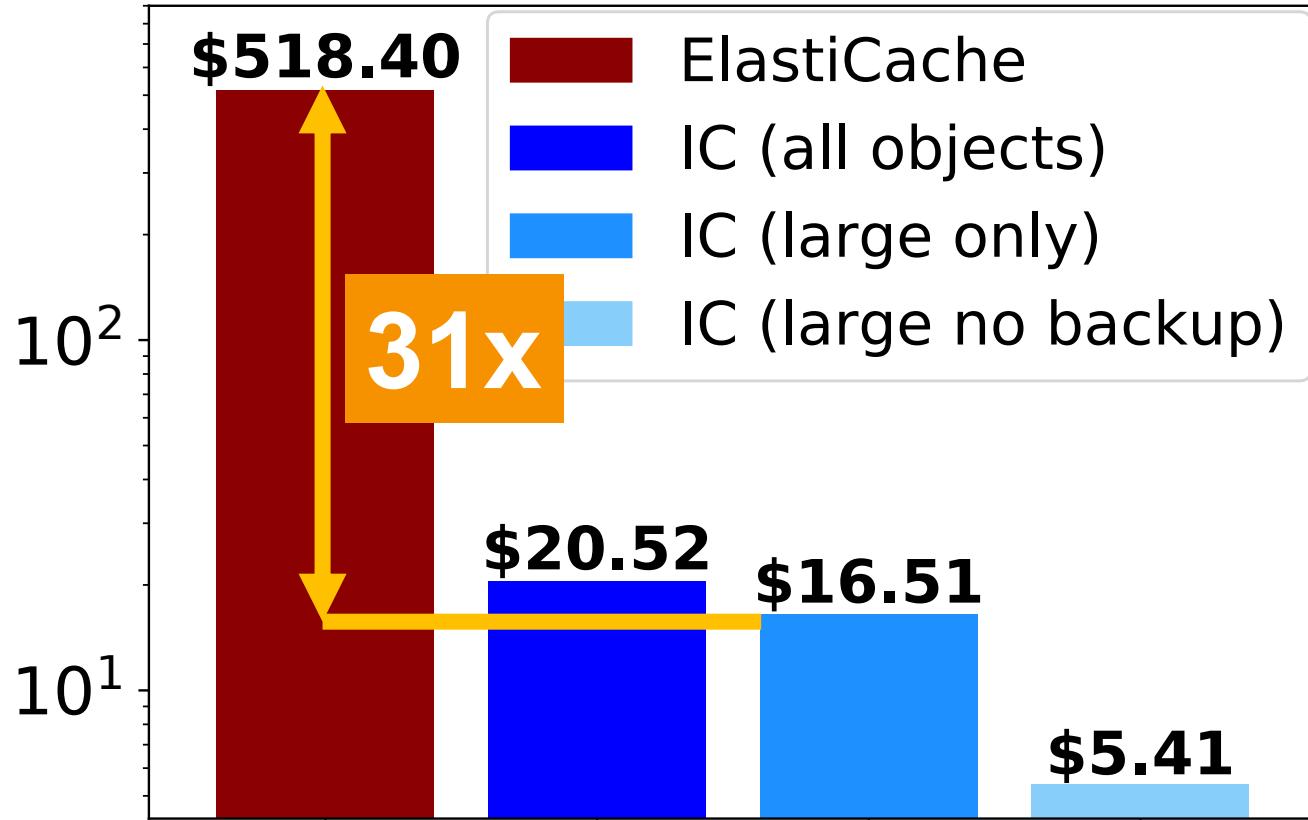
# Cost effectiveness of InfiniCache



## Workload setup

- All objects
- Large object only
  - Object larger than 10MB
- Large object w/o backup

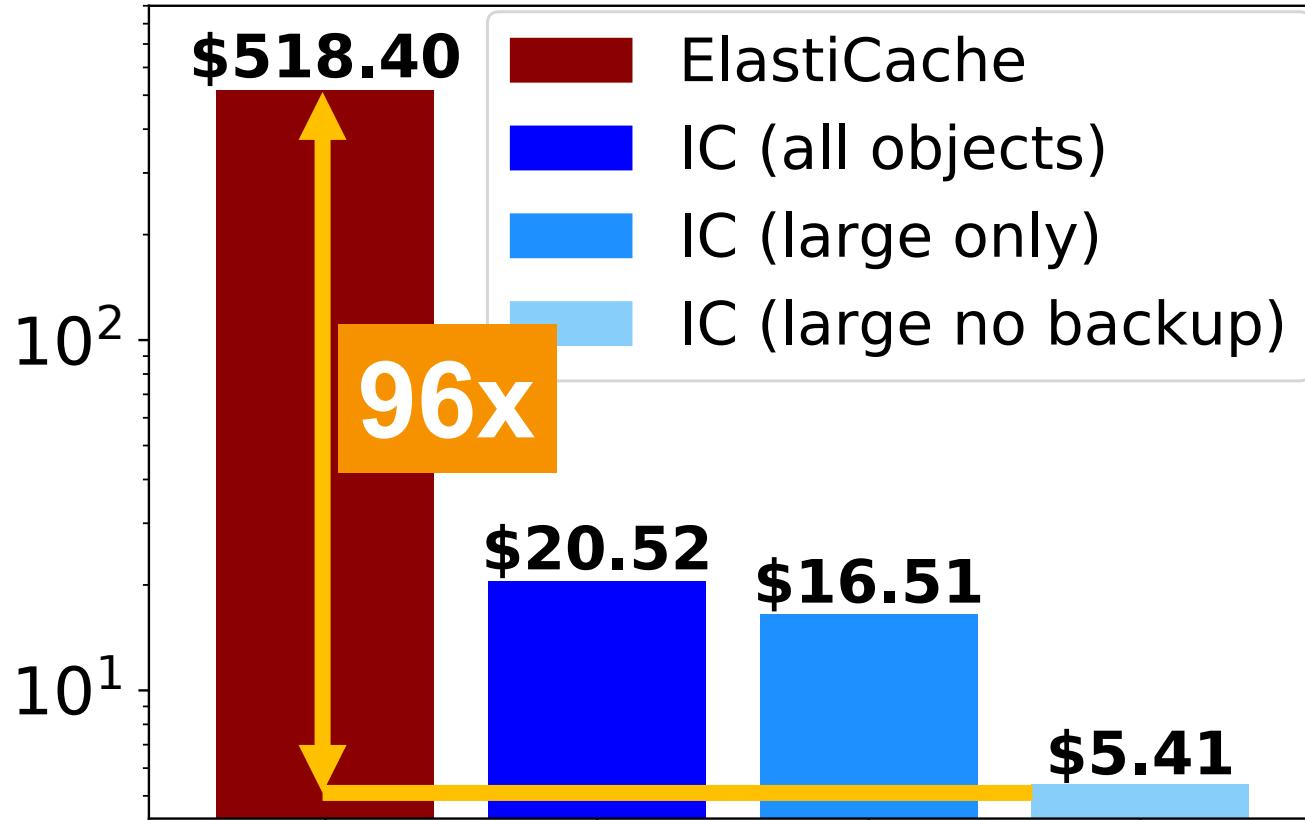
# Cost effectiveness of InfiniCache



Workload setup

- All objects
- **Large object only**
  - Object larger than 10MB
- Large object w/o backup

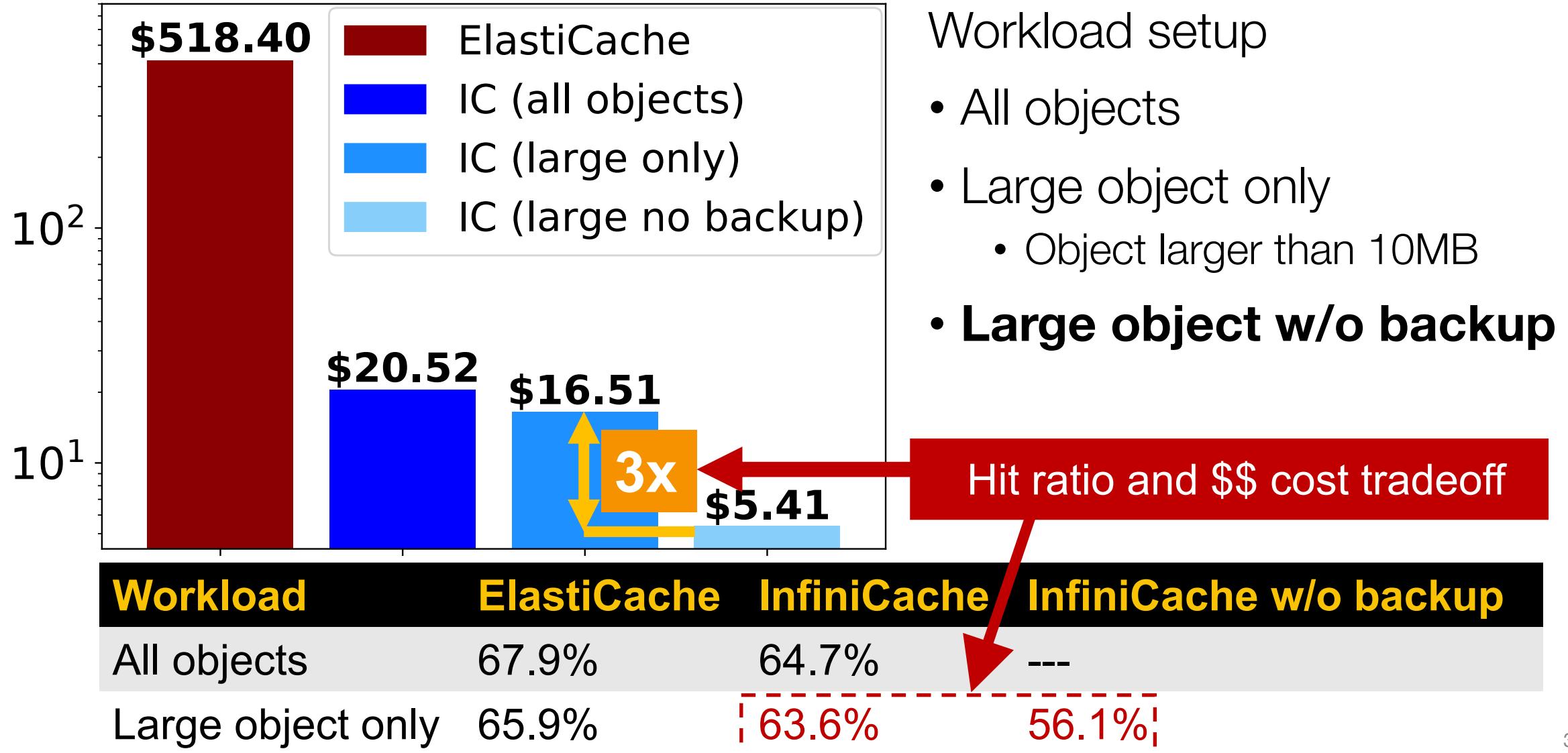
# Cost effectiveness of InfiniCache



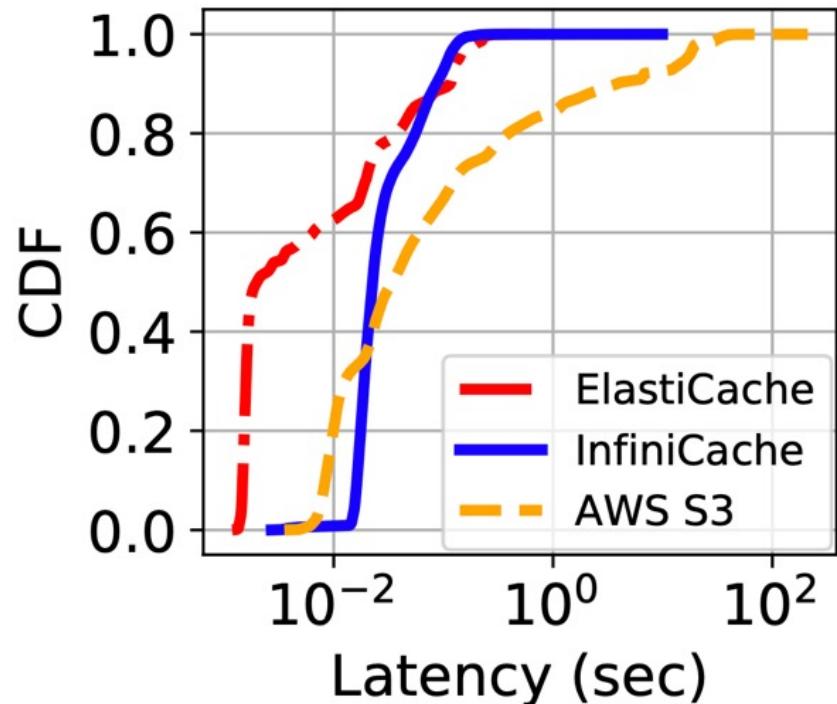
## Workload setup

- All objects
- Large object only
  - Object larger than 10MB
- **Large object w/o backup**

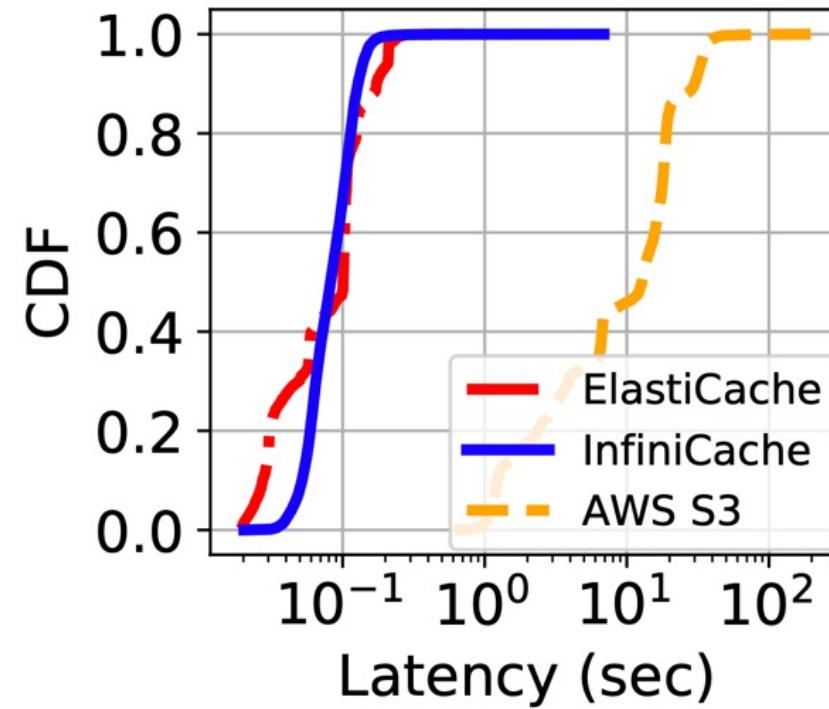
# Cost effectiveness of InfiniCache



# Performance of InfiniCache

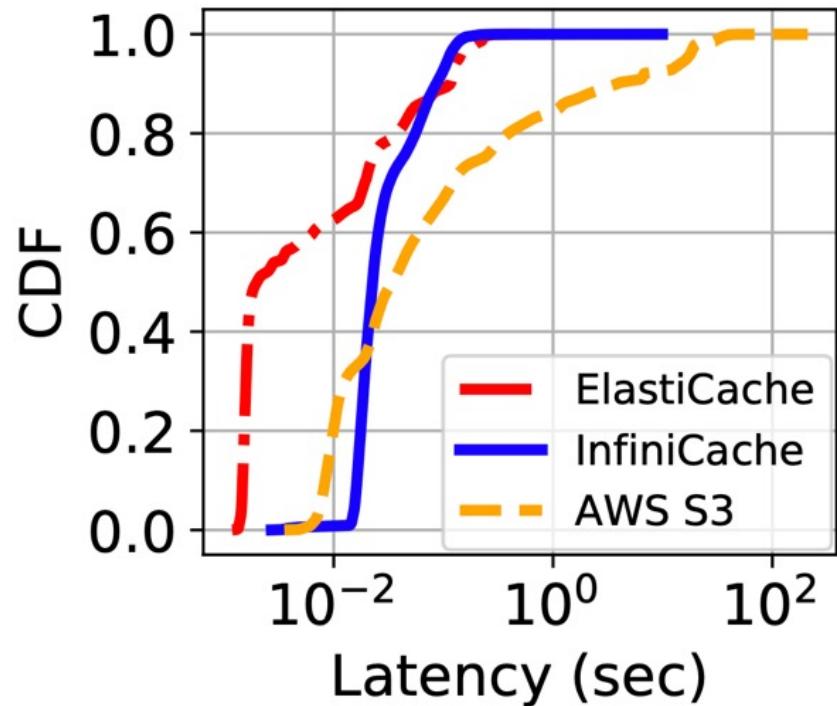


All objects

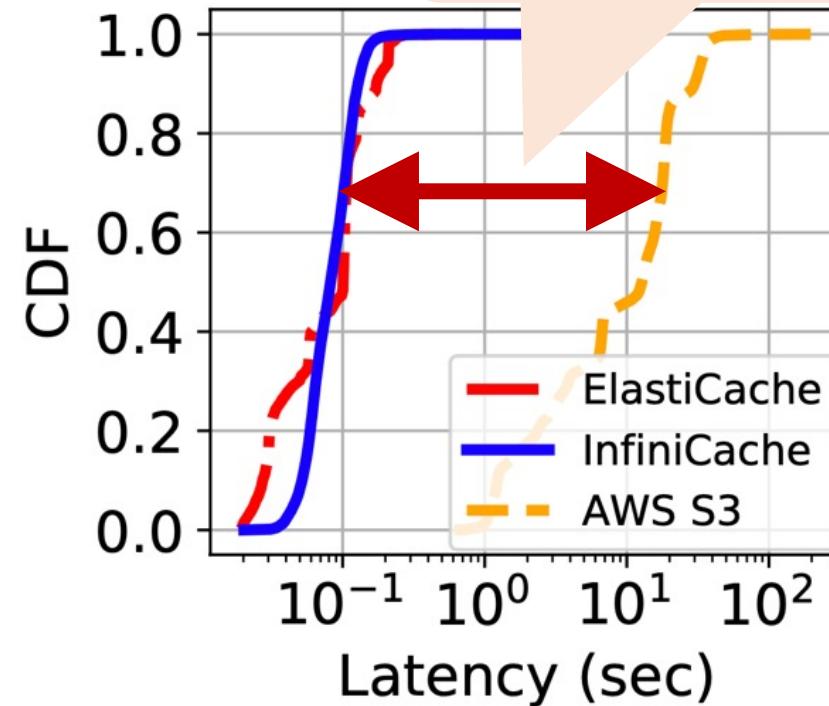


Large objects only

# Performance of InfiniCache



All objects



Large objects only

> 100 times improvement

# The future of cloud programming

FaaS is ~~poorly~~ suited for stateful applications, **but to achieve so, it requires research and innovation**

Making stateful applications **FaaS-aware**

Making FaaS platforms **embrace state**

- Hard things become easy → **more fun programming**
- Easy things become better → **save more \$\$\$**

A step towards a **programmable cloud**

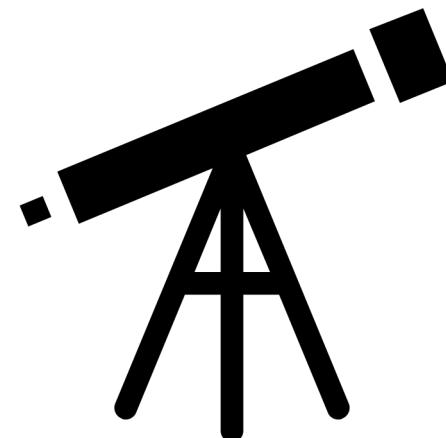


# Our vision towards a programmable cloud

Serverless will change the way we write software and the way that programming infrastructure works

Cloud-native programming models

- **Language-integrated programming models** that enable developers to take advantage of **millions of CPU cores and petabytes of RAM** as easy as **writing centralized code**



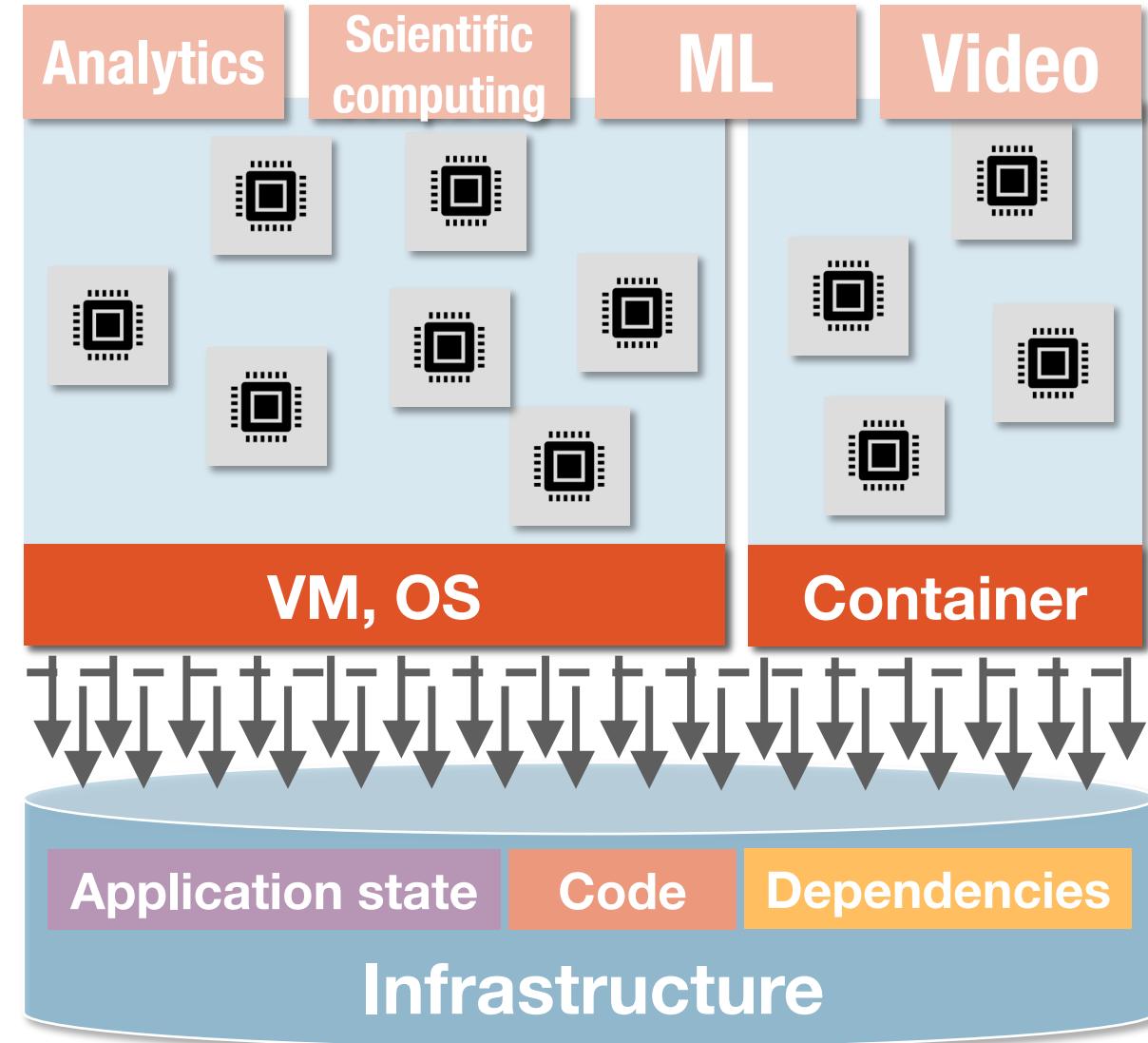
# Ongoing research: An end-to-end, full-stack approach

New stateful FaaS applications

New programming frameworks  
atop FaaS platforms

New infrastructure techniques  
within FaaS platforms

New FaaS-aware OS/hardware  
techniques



# I am looking for students to join my lab

- If you are interested in that kind of systems research that we are doing, come talk to us:

<https://ds2-lab.github.io/>

# FAQ – Conclusion

Q: How to make programming more productive

A: Cloud-native, language-integrated programming models are the future.

Q: Is FaaS well suited for stateful applications?

A: Not exactly. It requires research to make FaaS embrace state.

Q: Is FaaS poorly suited for stateful applications?

A: Not really! It requires research to rethink the application+system design.

Q: I'm still not convinced... Why bother?

A: Gain more **fun** (ease of use, zero server maintenance, etc.)

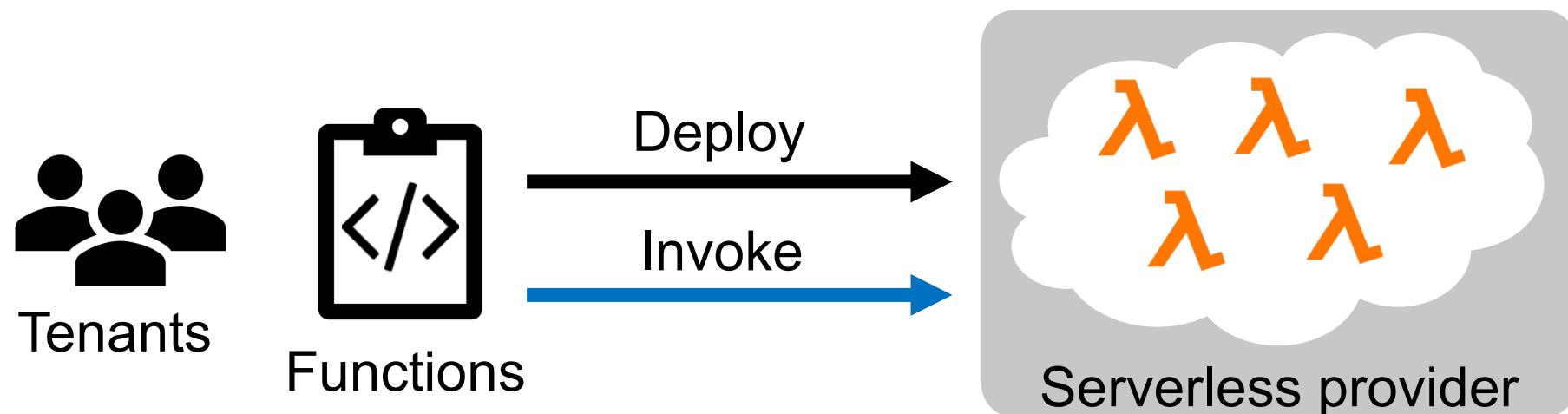
Gain more **profit** (pay per use, best bang for bucks, no idled VMs anymore)



# Backup slides

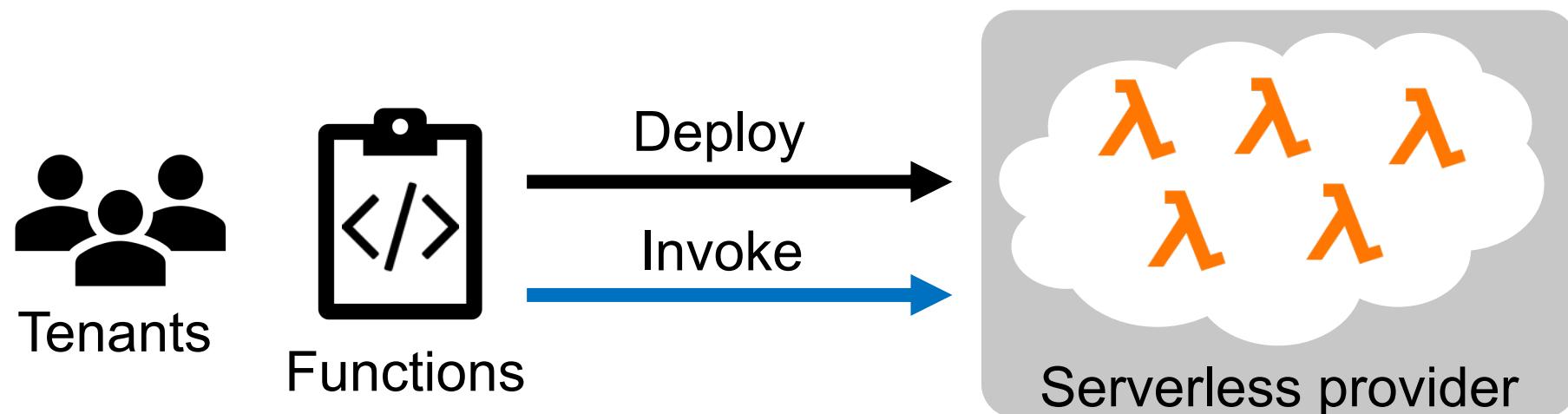
# A primer on Serverless Computing

- Serverless computing enables cloud tenants to launch short-lived tasks (i.e., Lambda functions) with **high elasticity** and **fine-grained resource billing**



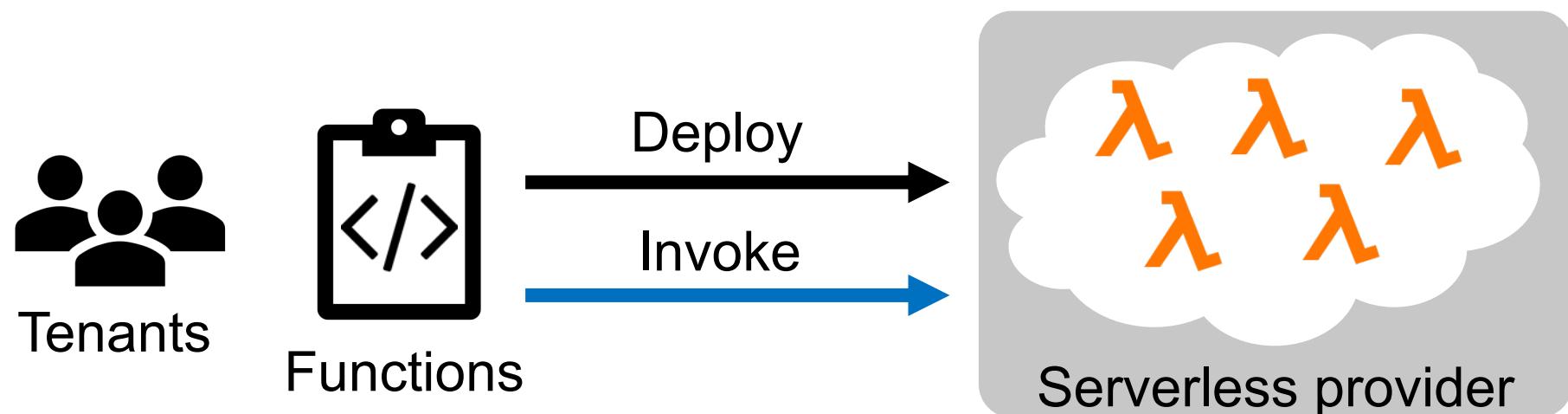
# A primer on Serverless Computing

- Serverless computing enables cloud tenants to launch short-lived tasks (i.e., Lambda functions) with **high elasticity** and **fine-grained resource billing**
- Function: basic unit of deployment. Application consists of multiple serverless functions



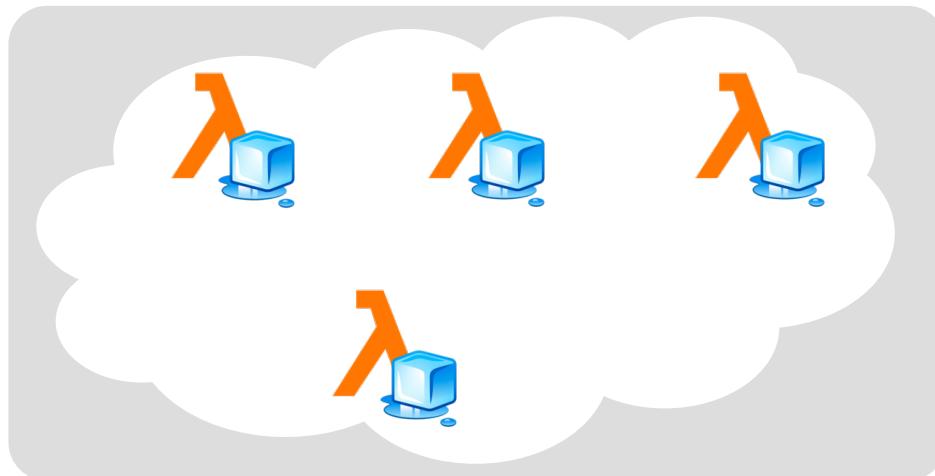
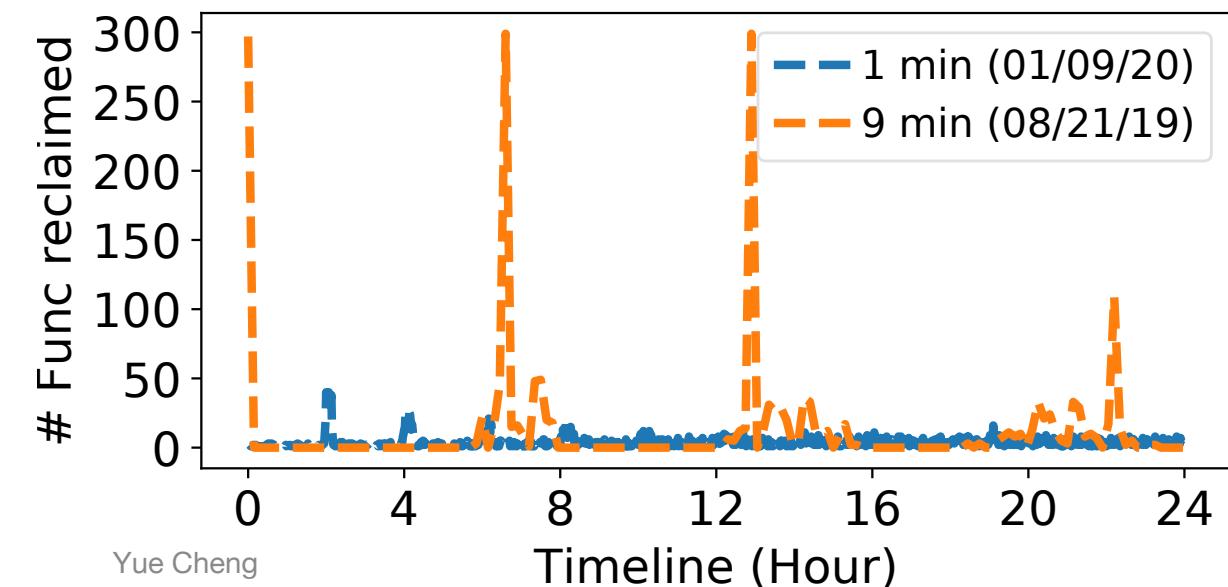
# A primer on Serverless Computing

- Serverless computing enables cloud tenants to launch short-lived tasks (i.e., Lambda functions) with **high elasticity** and **fine-grained resource billing**
- Function: basic unit of deployment. Application consists of multiple serverless functions
- Popular use cases: Backend APIs, data processing...



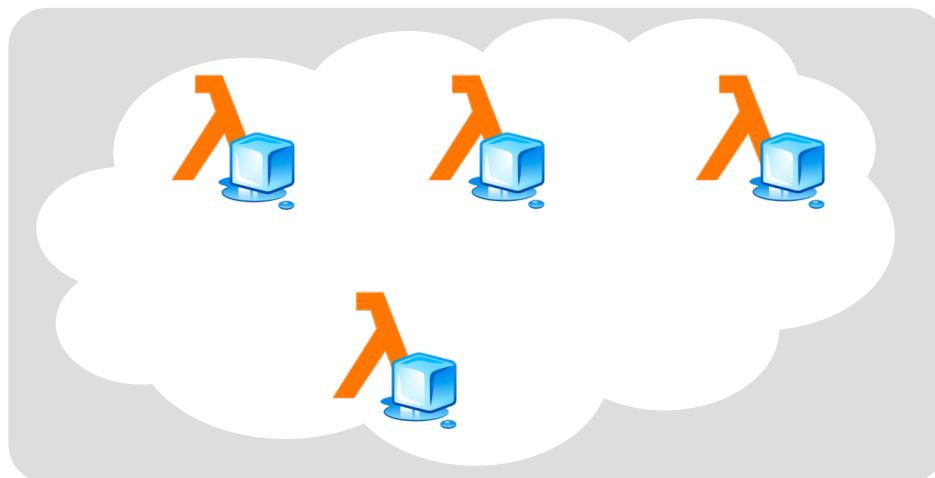
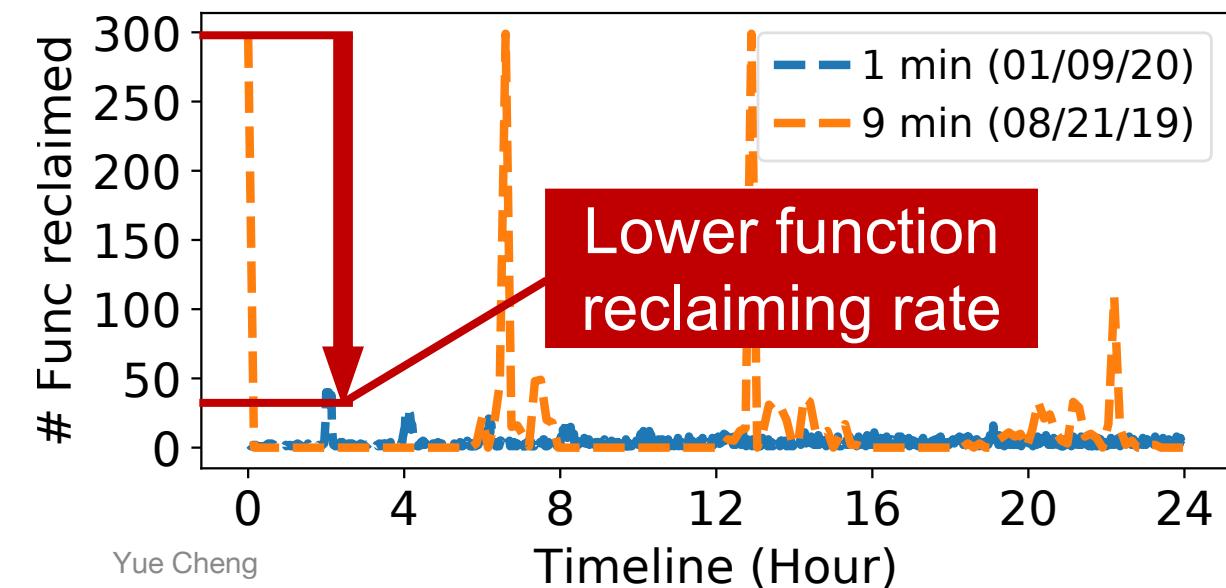
# Maximizing data availability: Periodic warm-up

1. Lambda nodes are cached by AWS when not running
  - AWS may reclaim cold Lambda functions after they are idling for a period



# Maximizing data availability: Periodic warm-up

1. Lambda nodes are cached by AWS when not running
  - AWS may reclaim cold Lambda functions after they are idling for a period



# Experimental setup

- InfiniCache
  - 400 1.5GB Lambda cache nodes
  - Client running on one c5n.4xlarge EC2 VM
  - Warm-up interval: 1 minute; backup interval: 5 minutes
  - Under one AWS VPC
- Production workloads
  - The first 50 hours of the Dallas datacenter traces from IBM Docker registry workloads
  - All objects: including small and large objects
  - Large object only: objects > 10MB

# Key question: How to make programmers productive?

# Key question: How to make programmers productive?

But, where will apps be run?

# Key question: How to make programmers productive?

But, where will apps be run? In the



!

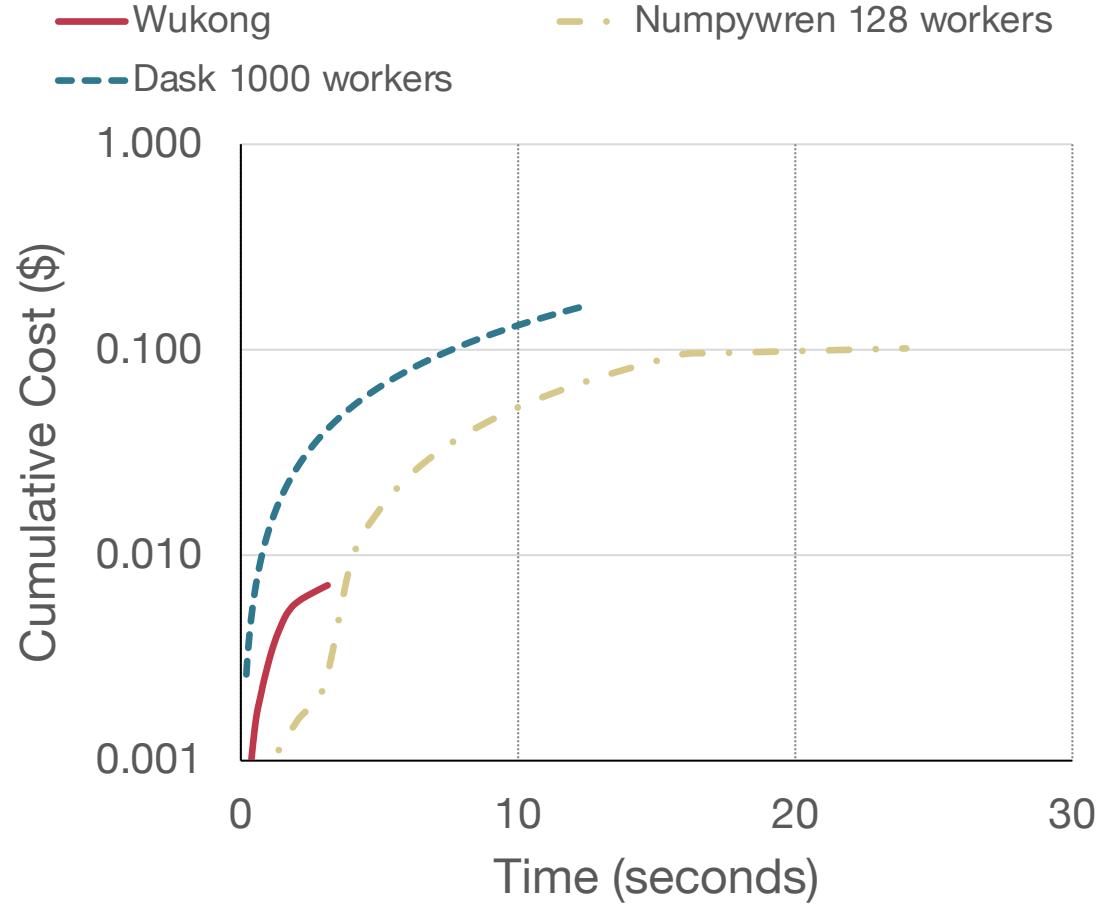
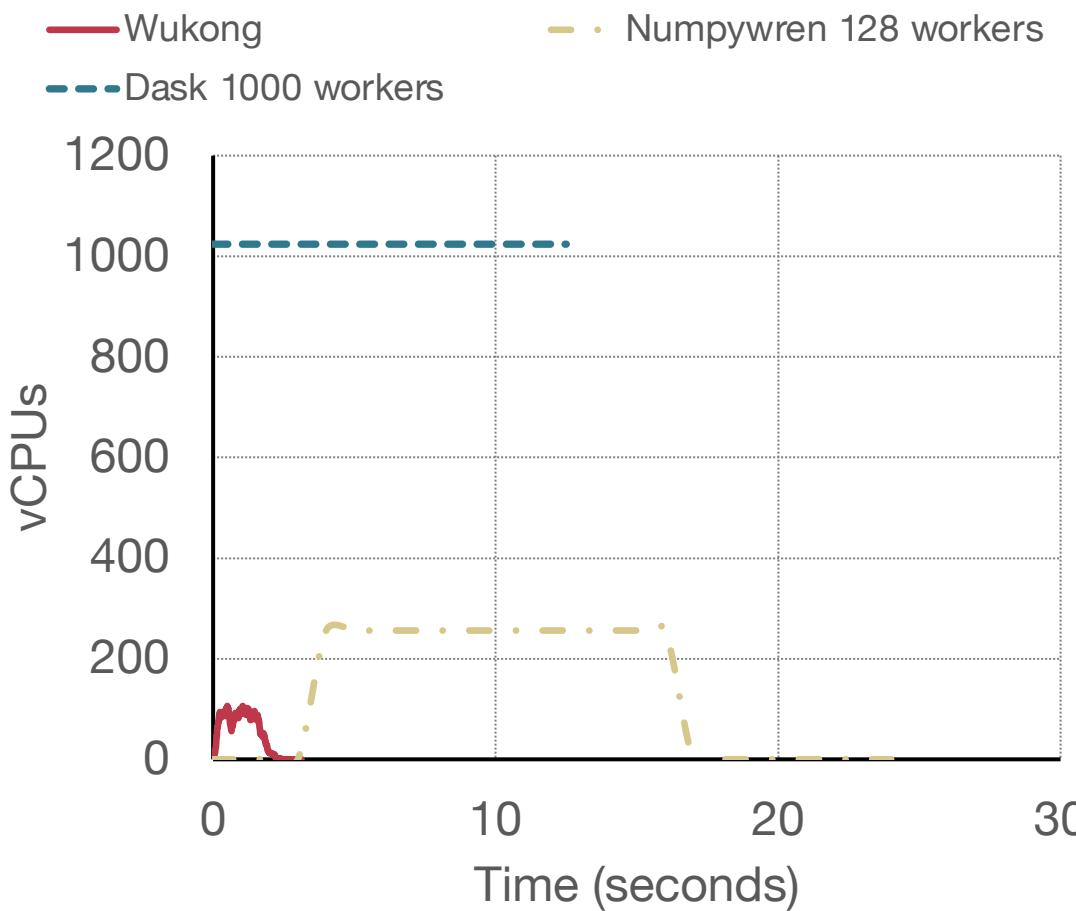
# Key question: How to make programmers productive?

But, where will apps be run? In the



My research aims to enable easy-to-use, cloud-native, language-integrated programming abstraction to support stateful applications

# What about elasticity and cost: TSQR

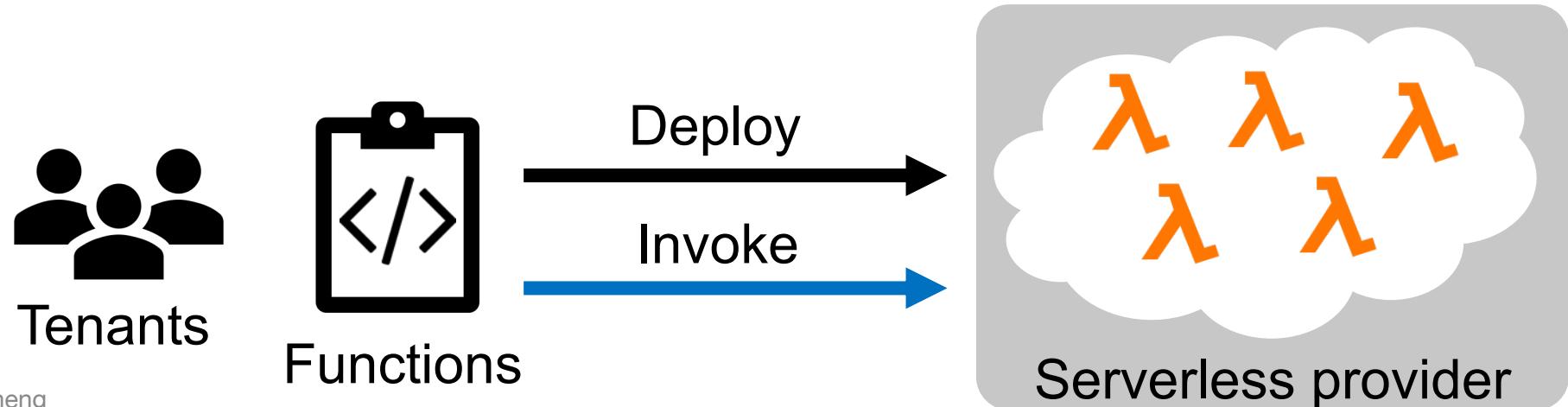


- Caching both small and large objects is challenging
- Existing solutions either too slow or too expensive

**Requires rethinking about a new cloud  
cache/storage model that is both fast and cheap!**

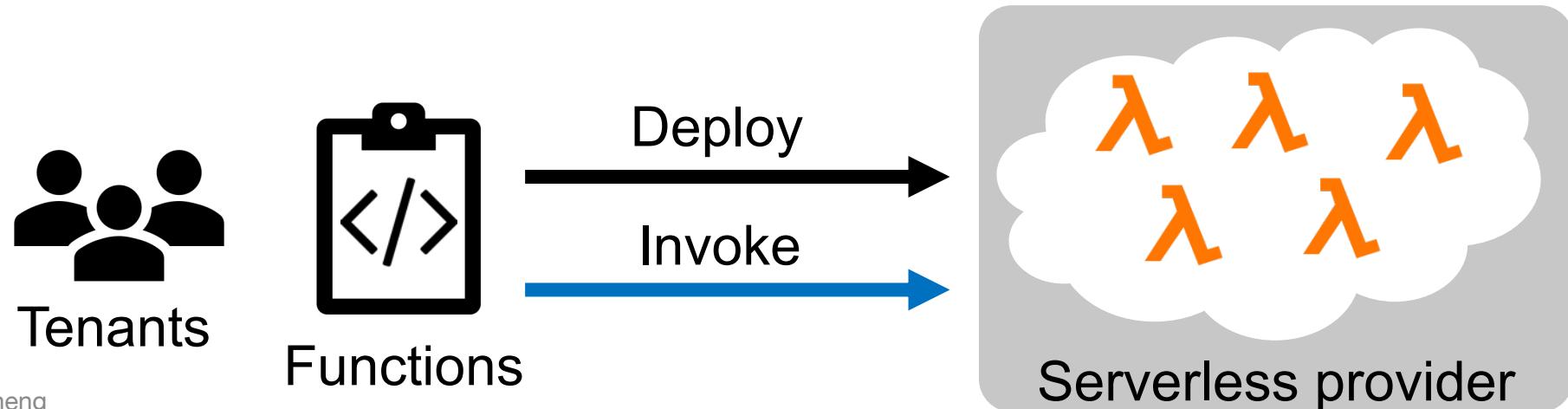
# FaaS is desirable

- Pay-per-use pricing model
  - AWS Lambda: \$0.2 per 1M invocations  
\$0.00001667 for every GB-sec



# FaaS is desirable

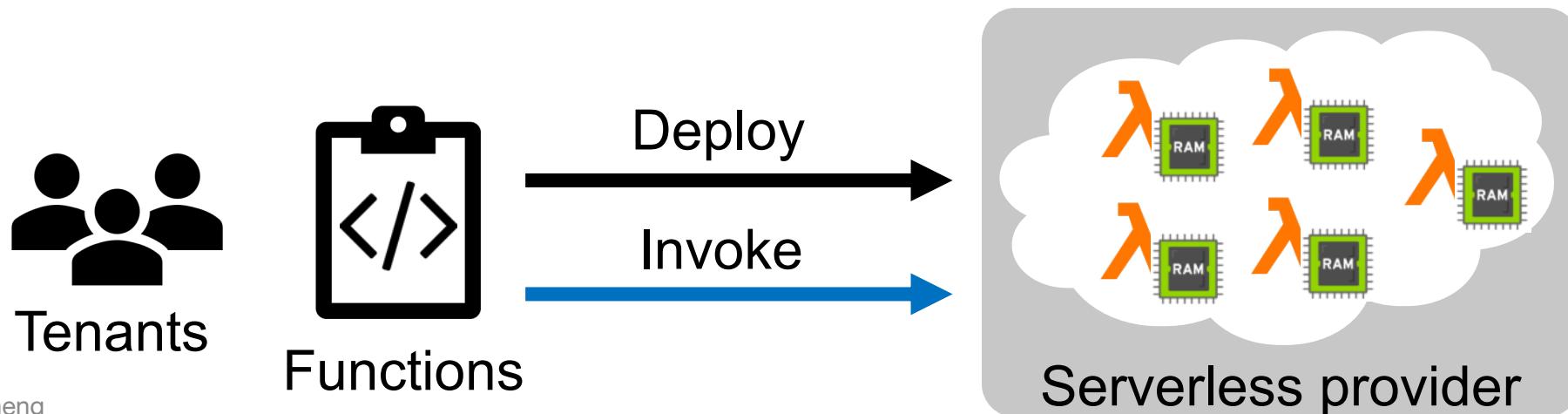
- Pay-per-use pricing model
  - AWS Lambda: \$0.2 per 1M invocations  
\$0.00001667 for every GB-sec
- Short-term function caching
  - Provider caches triggered functions in memory without charging tenants



# FaaS is desirable

- Pay-per-use pricing model
  - AWS Lambda: \$0.2 per 1M invocations  
\$0.00001667 for every GB-sec
- Short-term function caching
  - Provider caches triggered functions in memory without charging tenants

**Goal: Build a cost-effective, high-performance memory cache that exploits the FaaS features**



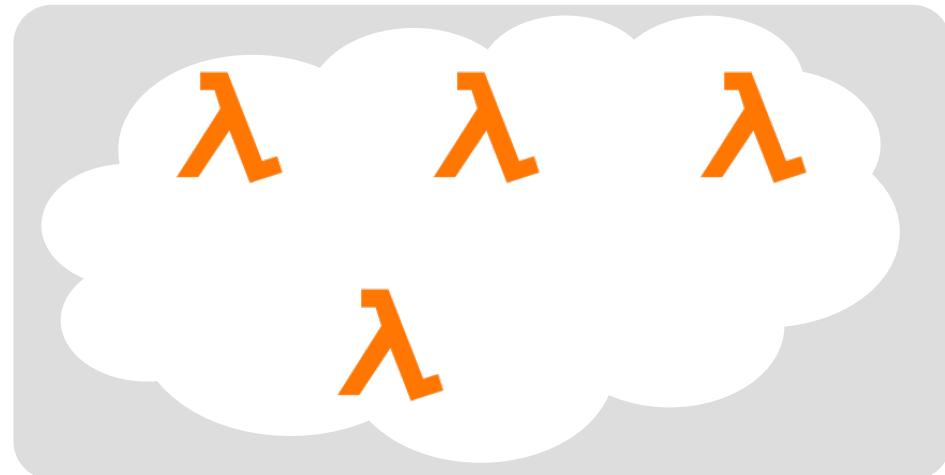
# Maximizing data availability

- Erasure-coding
- Periodic warm-up
- Periodic delta-sync backup

# Maximizing data availability: Periodic warm-up

1. Lambda nodes are cached by AWS when not running

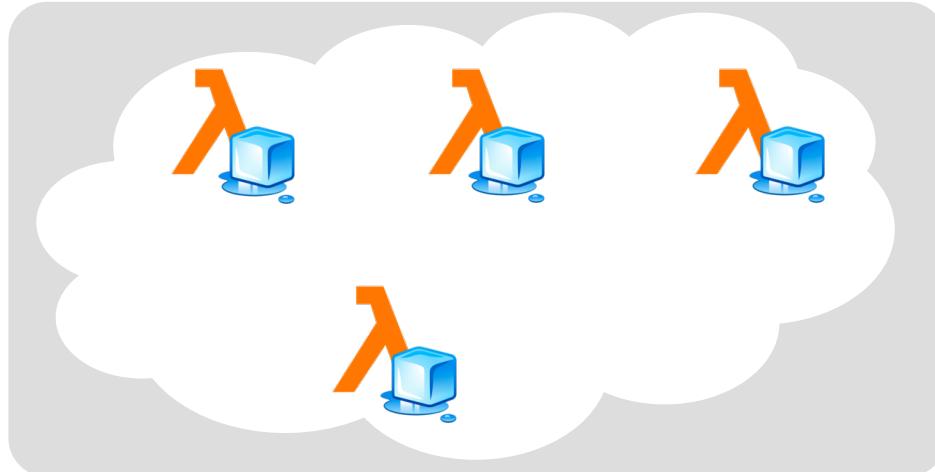
Proxy



# Maximizing data availability: Periodic warm-up

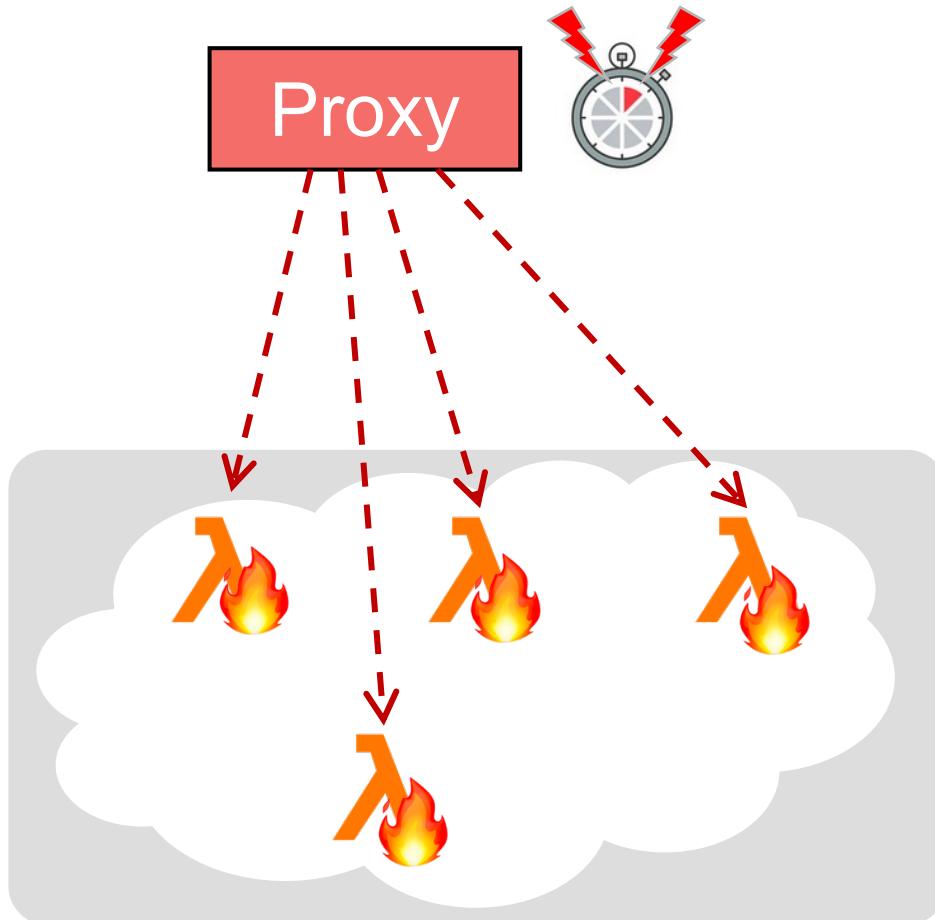
1. Lambda nodes are cached by AWS when not running
  - AWS may reclaim cold Lambda functions after they are idling for a period of time

Proxy



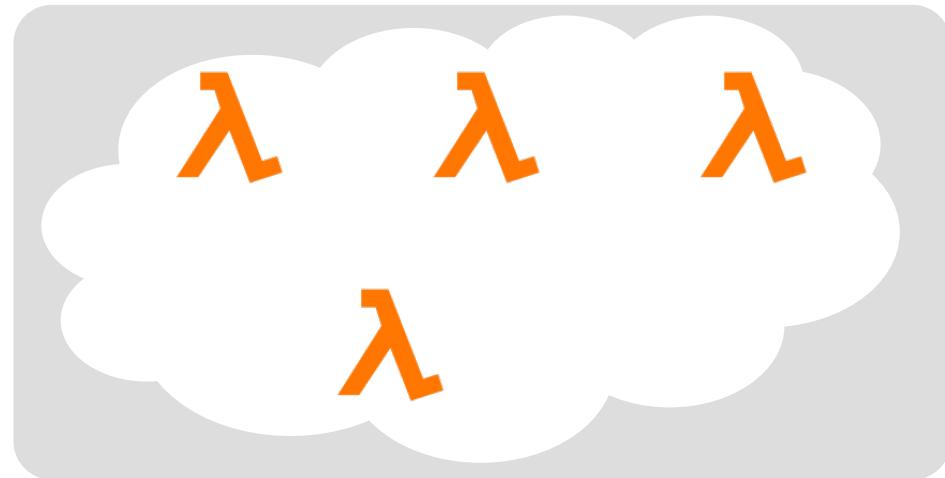
# Maximizing data availability: Periodic warm-up

1. Lambda nodes are cached by AWS when not running
2. Proxy periodically invokes sleeping Lambda cache nodes to extend their lifespan

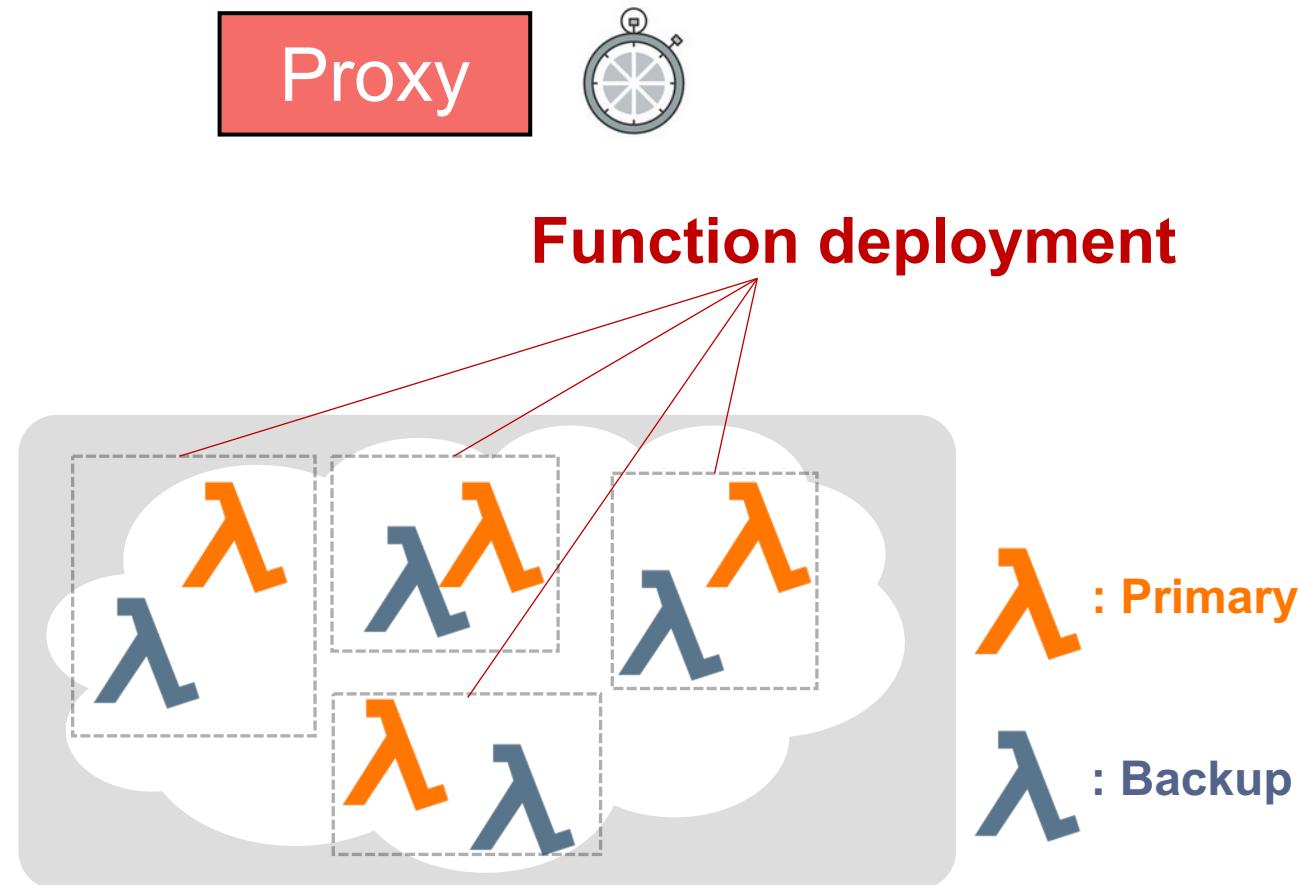


# Maximizing data availability: Periodic backup

Proxy

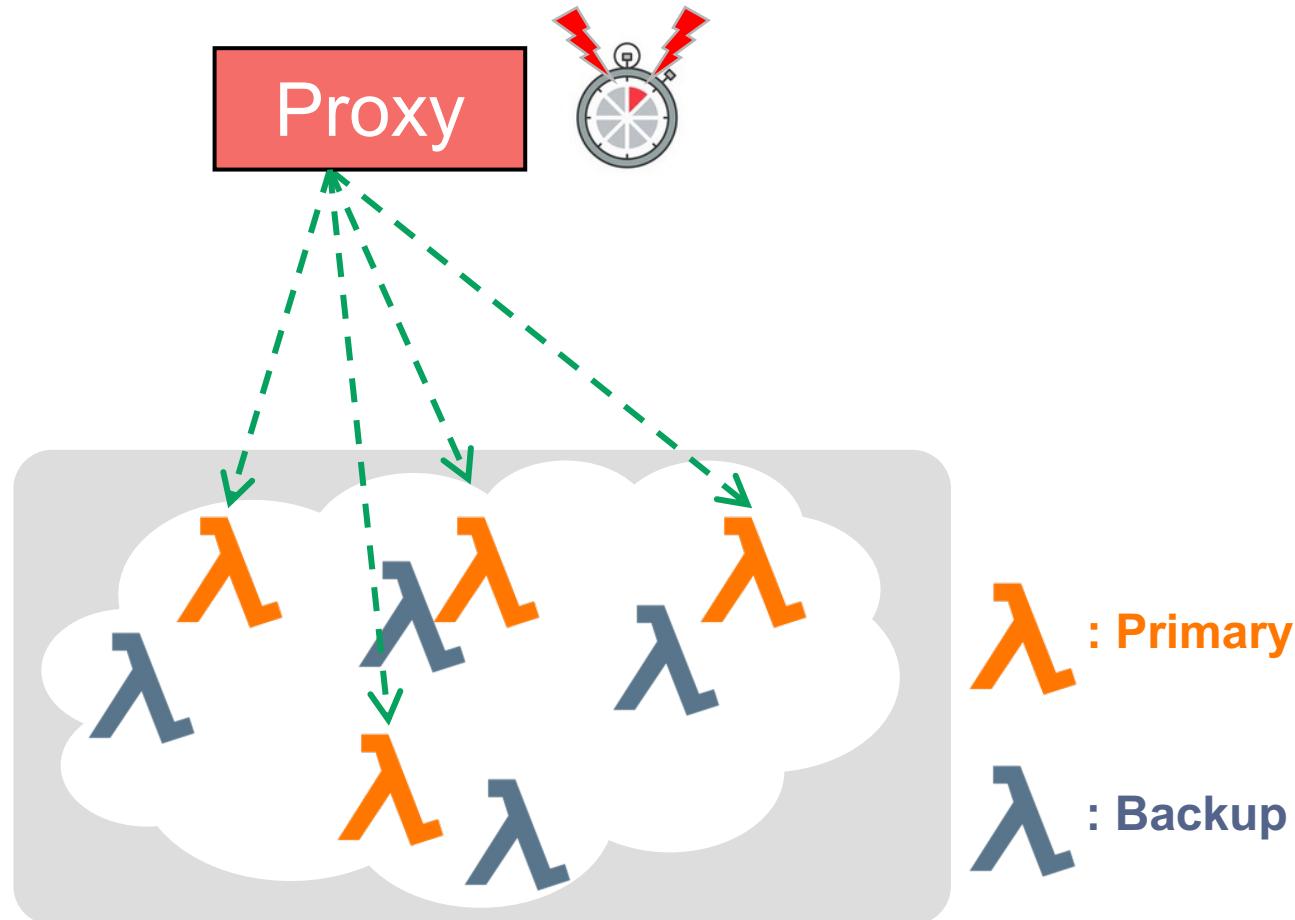


# Maximizing data availability: Periodic backup



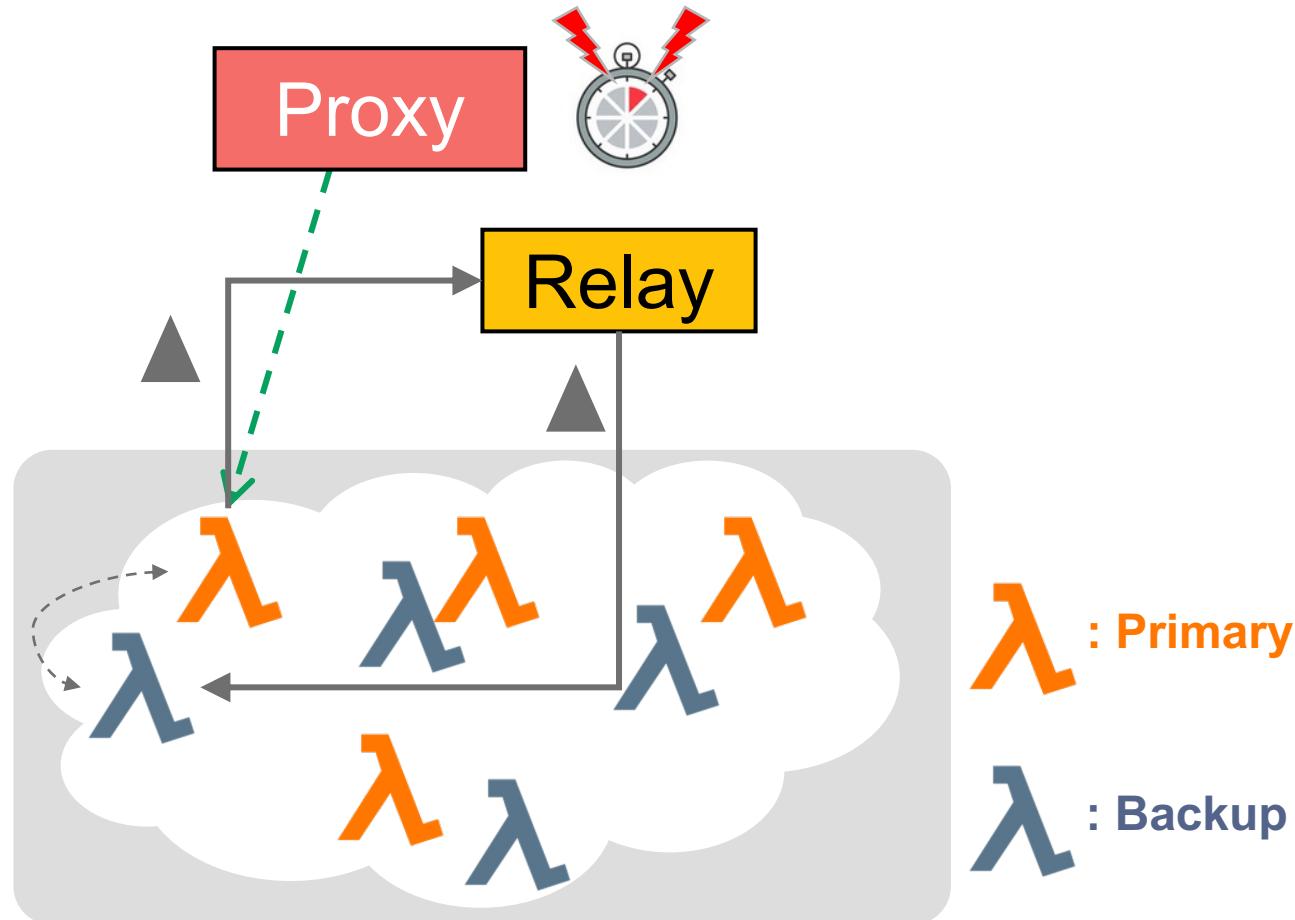
# Maximizing data availability: Periodic backup

1. Proxy sends out backup commands to Lambda cache nodes periodically

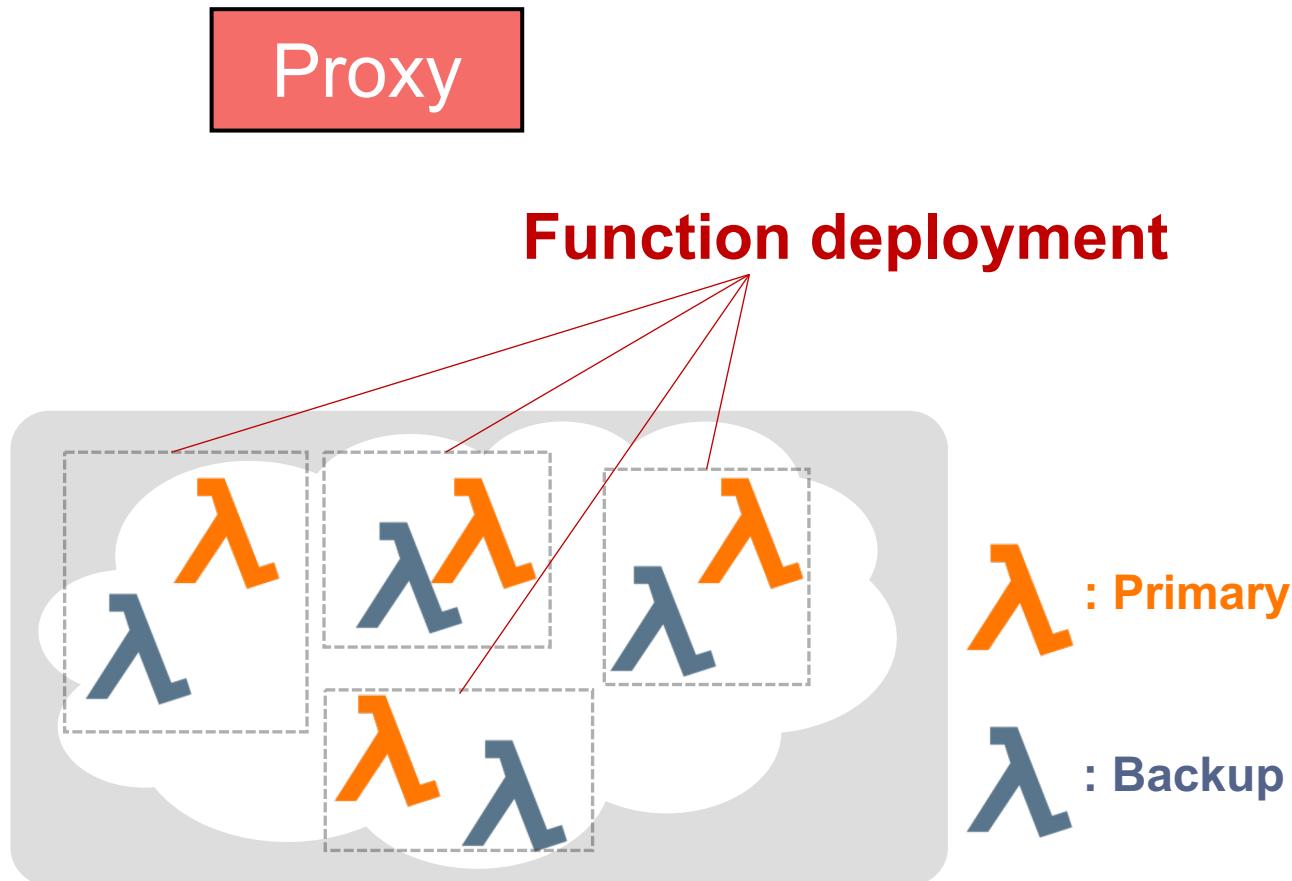


# Maximizing data availability: Periodic backup

1. Proxy sends out backup commands to Lambda cache nodes periodically
2. Lambda node performs delta-sync with its peer replica
  - Source Lambda propagates delta-update  to destination Lambda

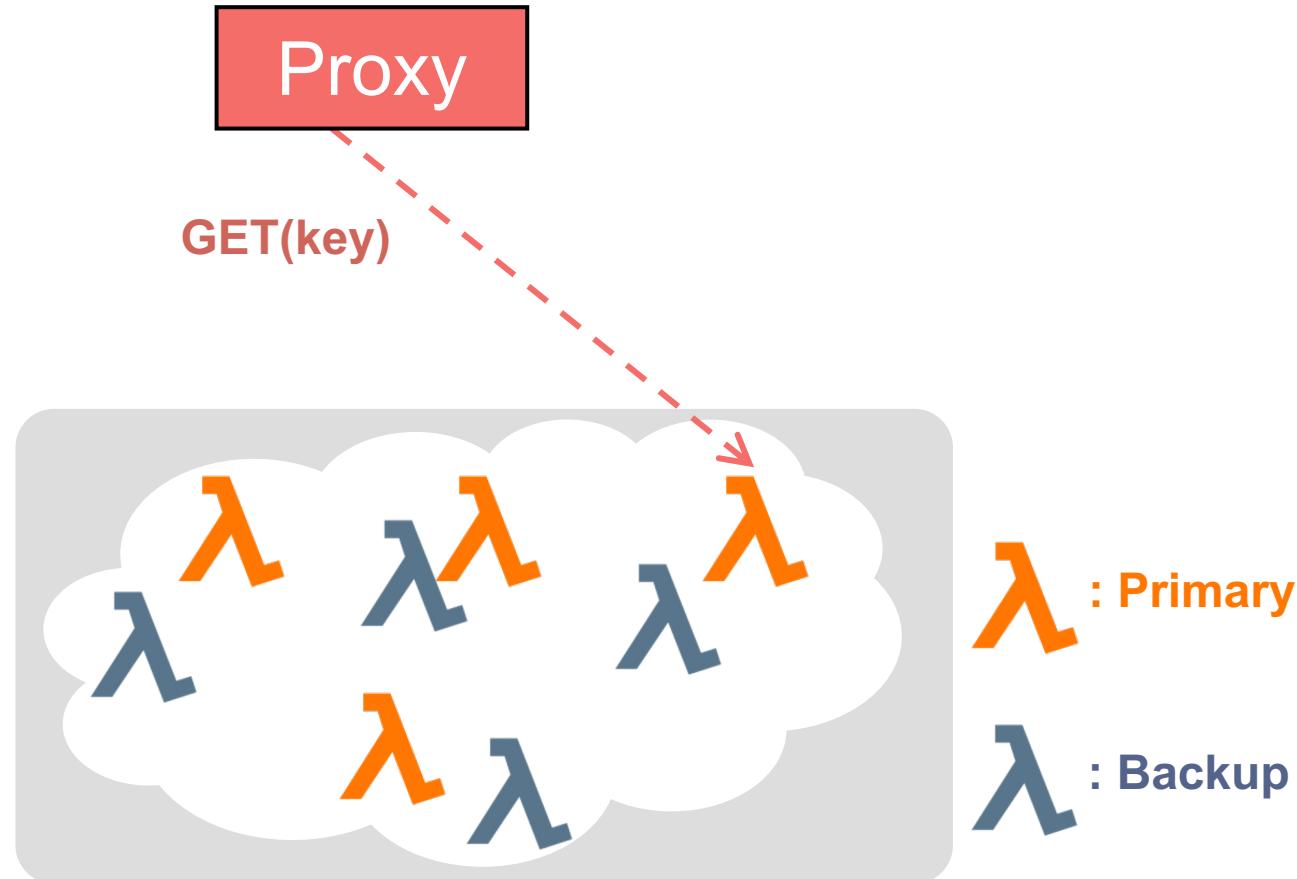


# Maximizing data availability: Seamless failover



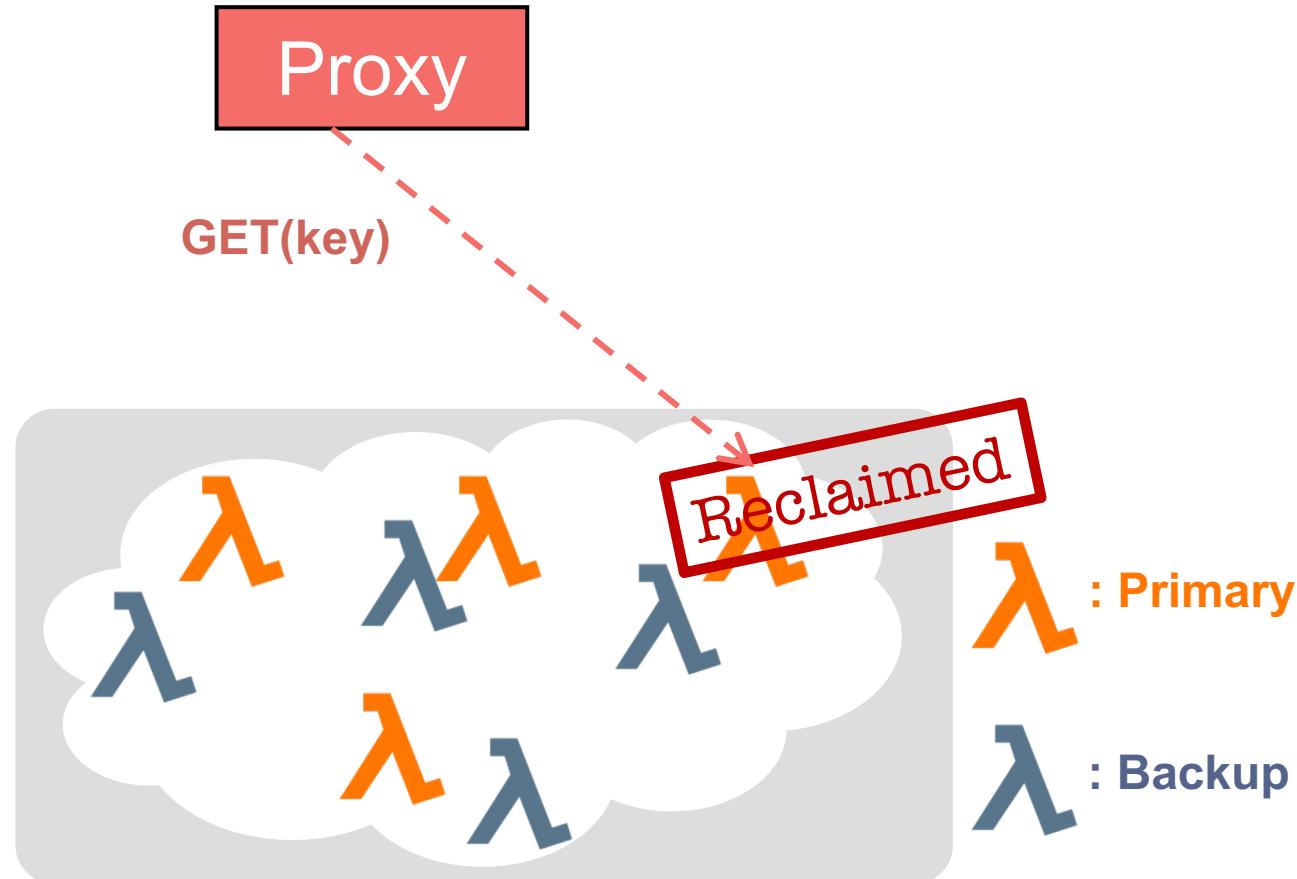
# Maximizing data availability: Seamless failover

1. Proxy invokes a Lambda cache node with a GET request



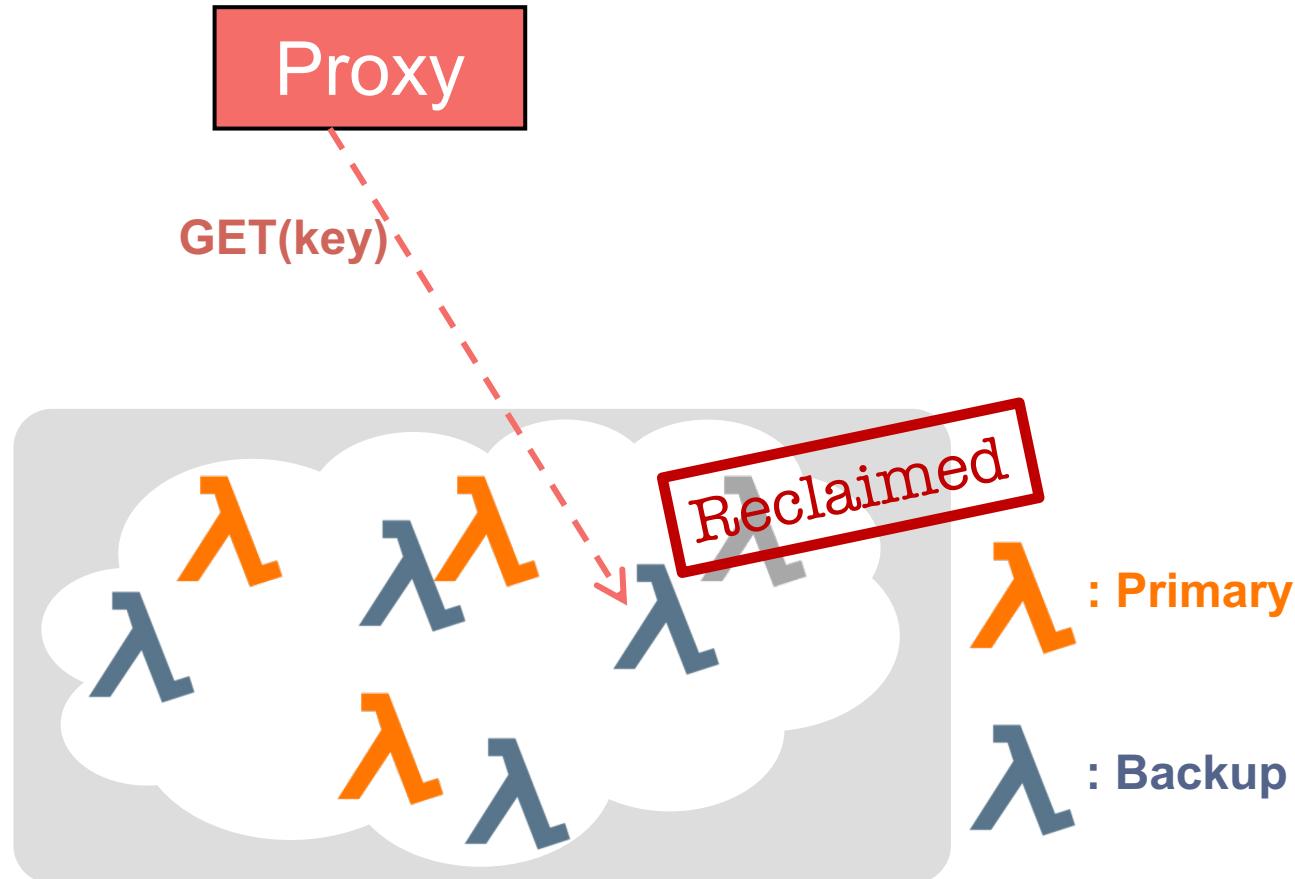
# Maximizing data availability: Seamless failover

1. Proxy invokes a Lambda cache node with a GET request
2. Primary Lambda gets reclaimed



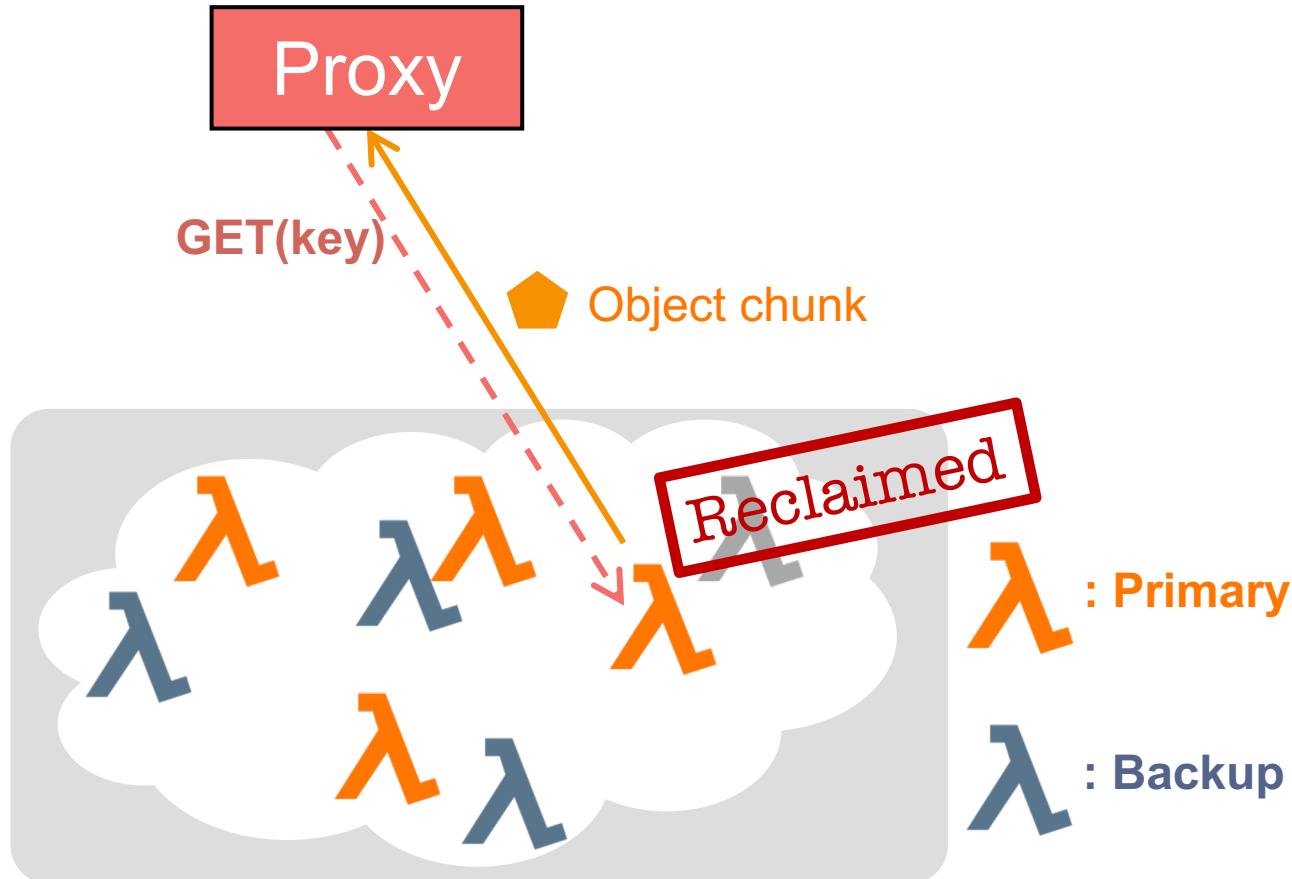
# Maximizing data availability: Seamless failover

1. Proxy invokes a Lambda cache node with a GET request
2. Primary Lambda gets reclaimed
3. The invocation request gets seamlessly redirected to the backup Lambda

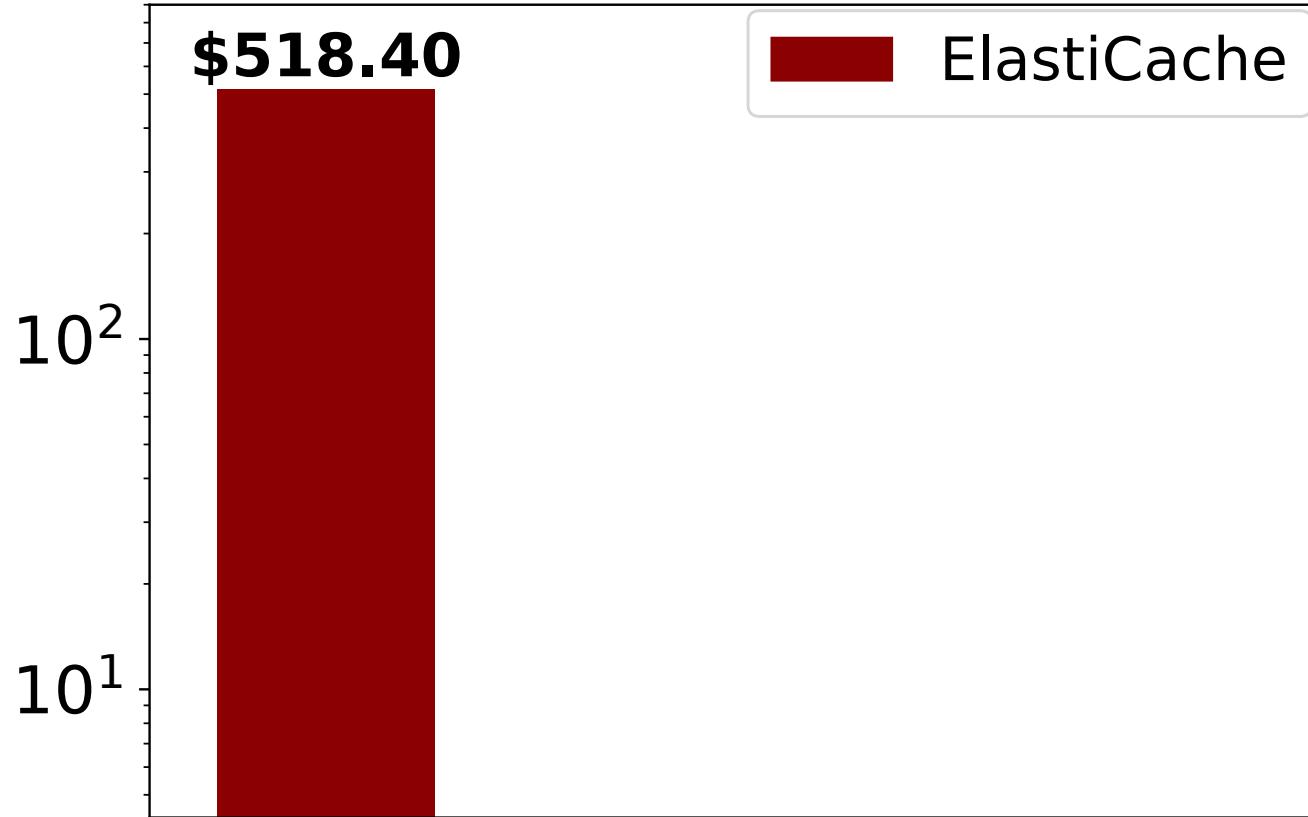


# Maximizing data availability: Seamless failover

1. Proxy invokes a Lambda cache node with a GET request
2. Primary Lambda gets reclaimed
3. The invocation request gets seamlessly redirected to the backup Lambda
  - Backup Lambda becomes the primary
  - Achieves automatic failover by exploiting FaaS **auto-scaling**



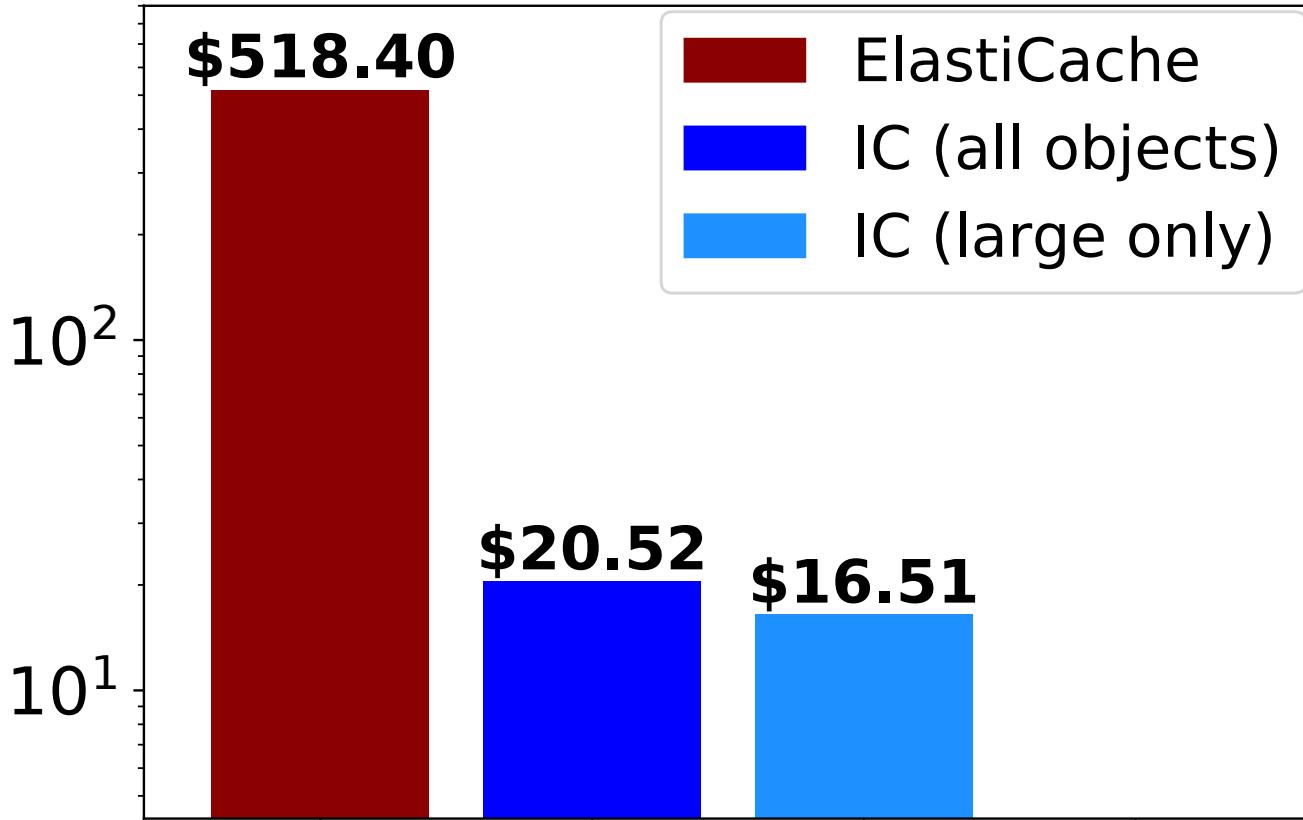
# Cost effectiveness of InfiniCache



## AWS ElastiCache

- One cache.r5.24xlarge with 600GB memory
- \$10.368 per hour

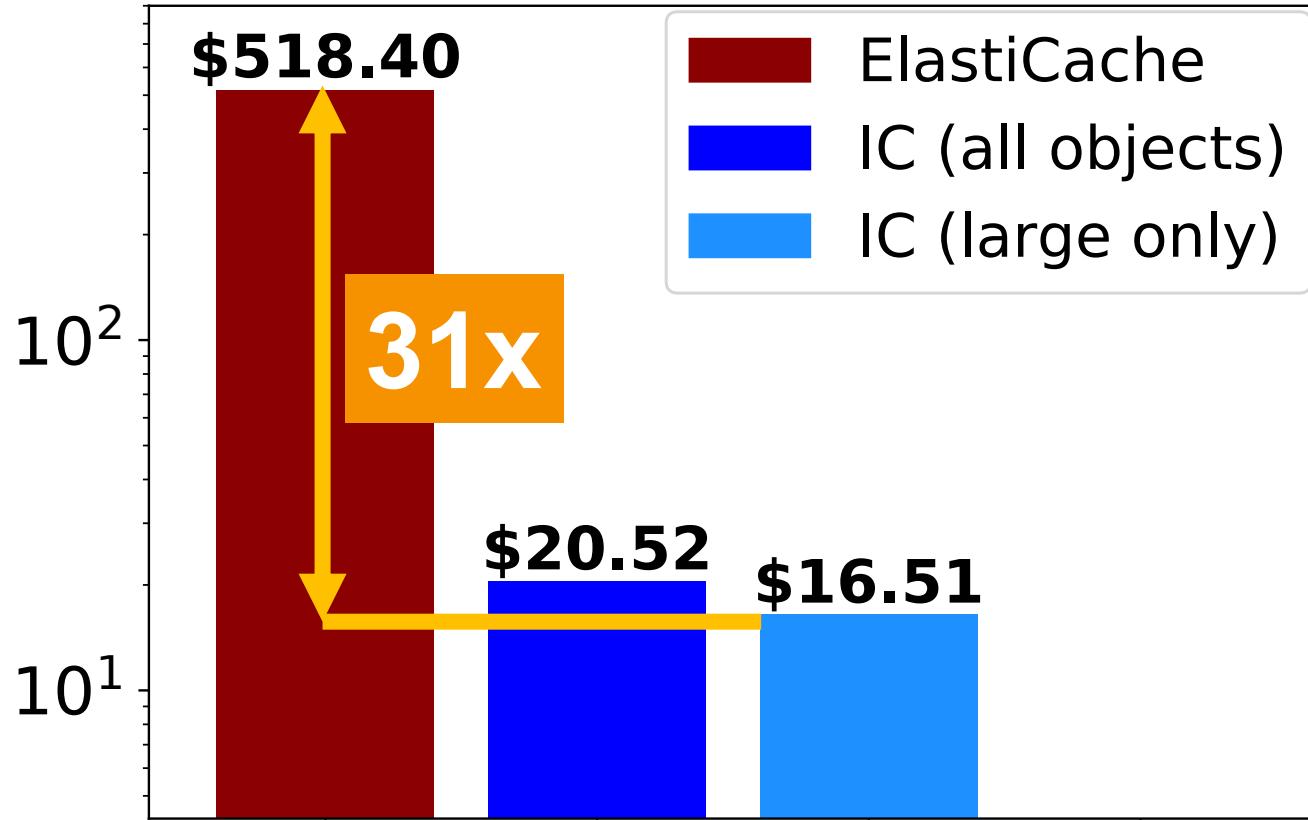
# Cost effectiveness of InfiniCache



## Workload setup

- All objects
- Large object only
  - Object **larger than 10MB**

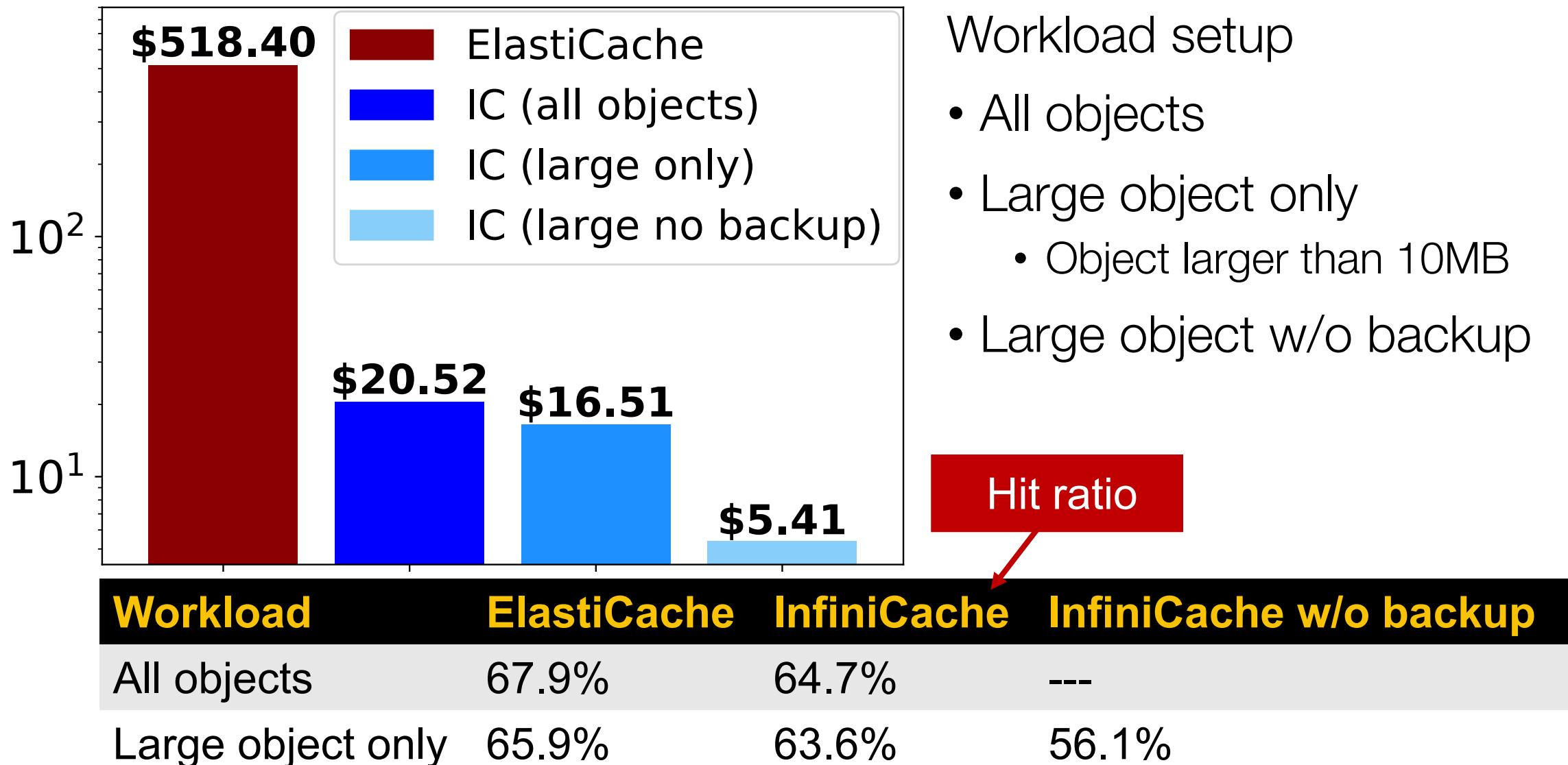
# Cost effectiveness of InfiniCache



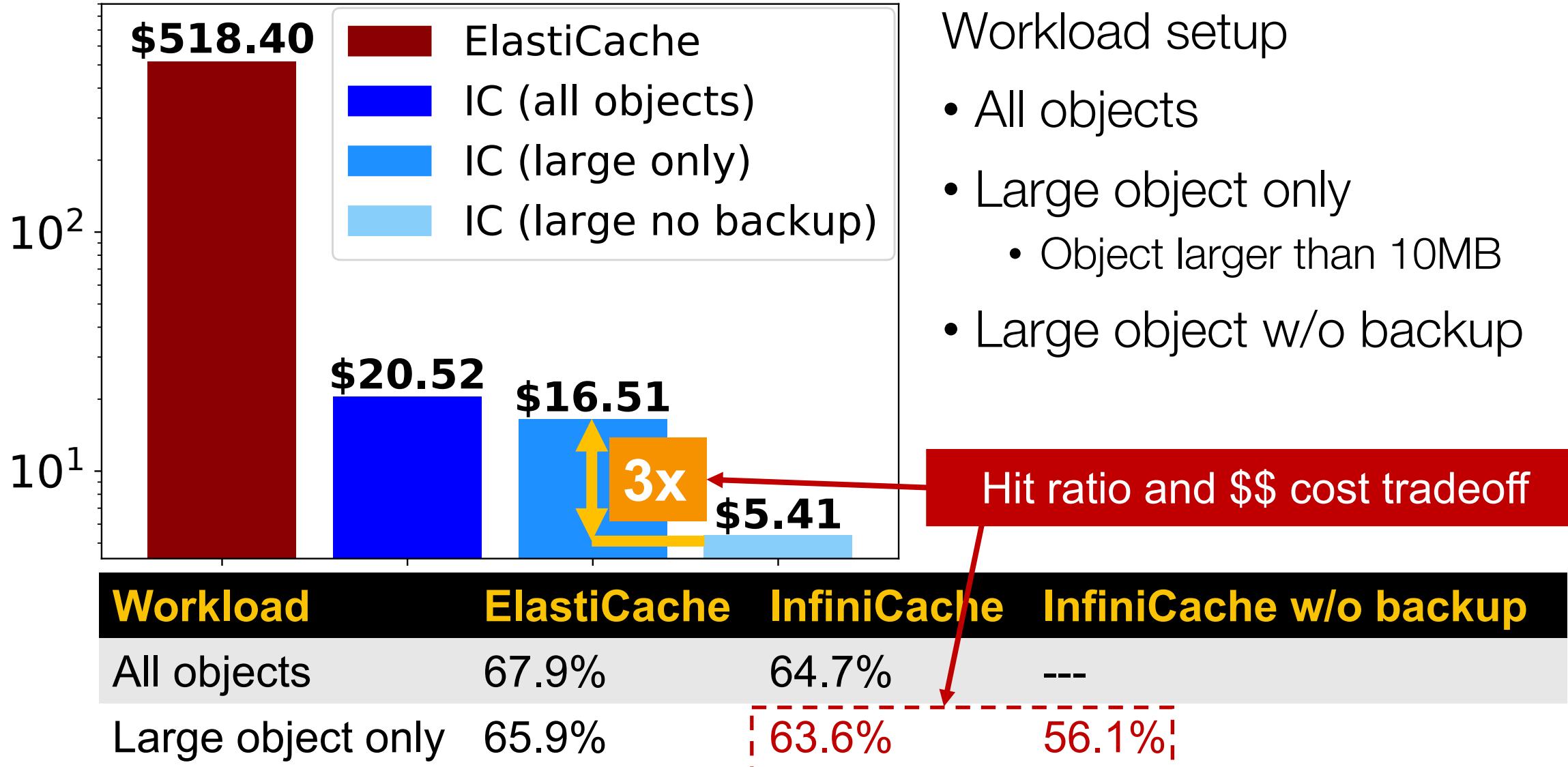
## Workload setup

- All objects
- Large object only
  - Object **larger than 10MB**

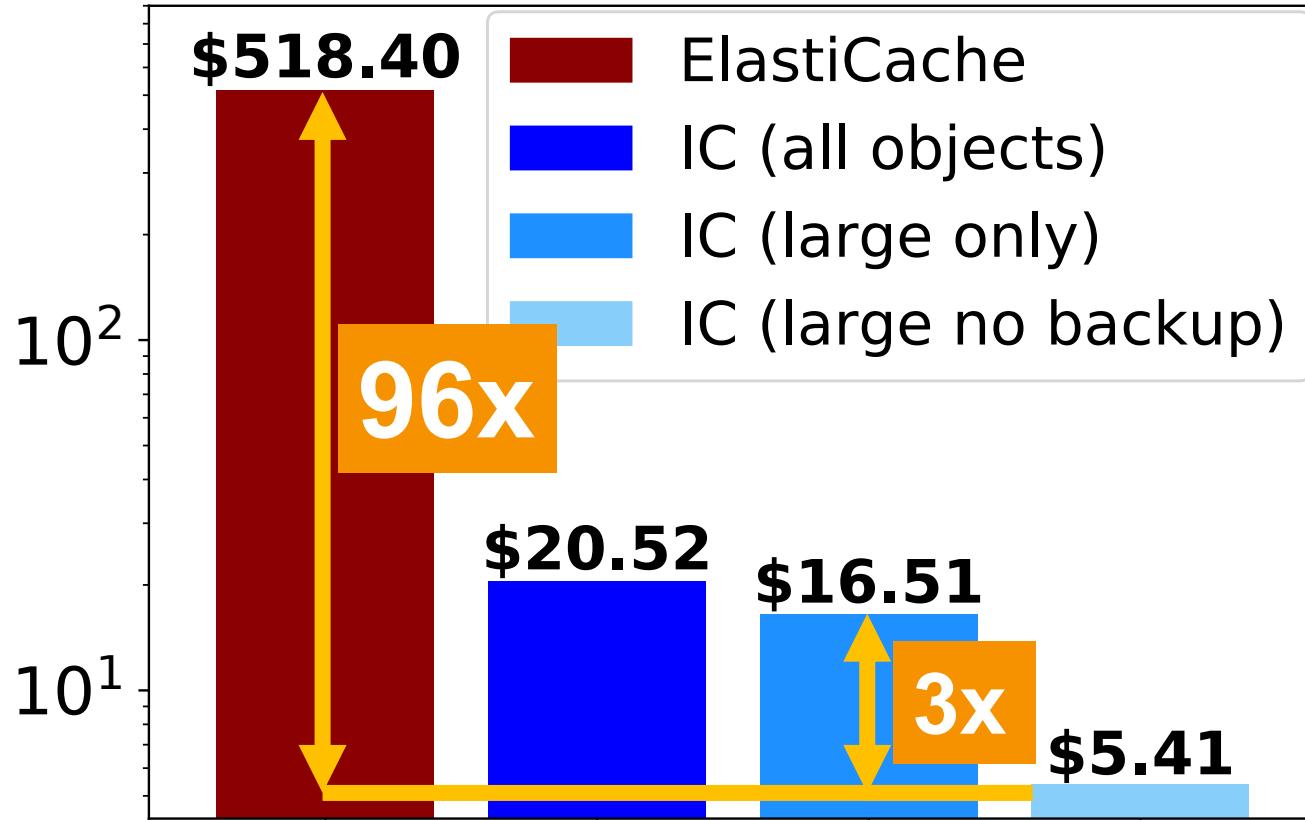
# Cost effectiveness of InfiniCache



# Cost effectiveness of InfiniCache



# Cost effectiveness of InfiniCache

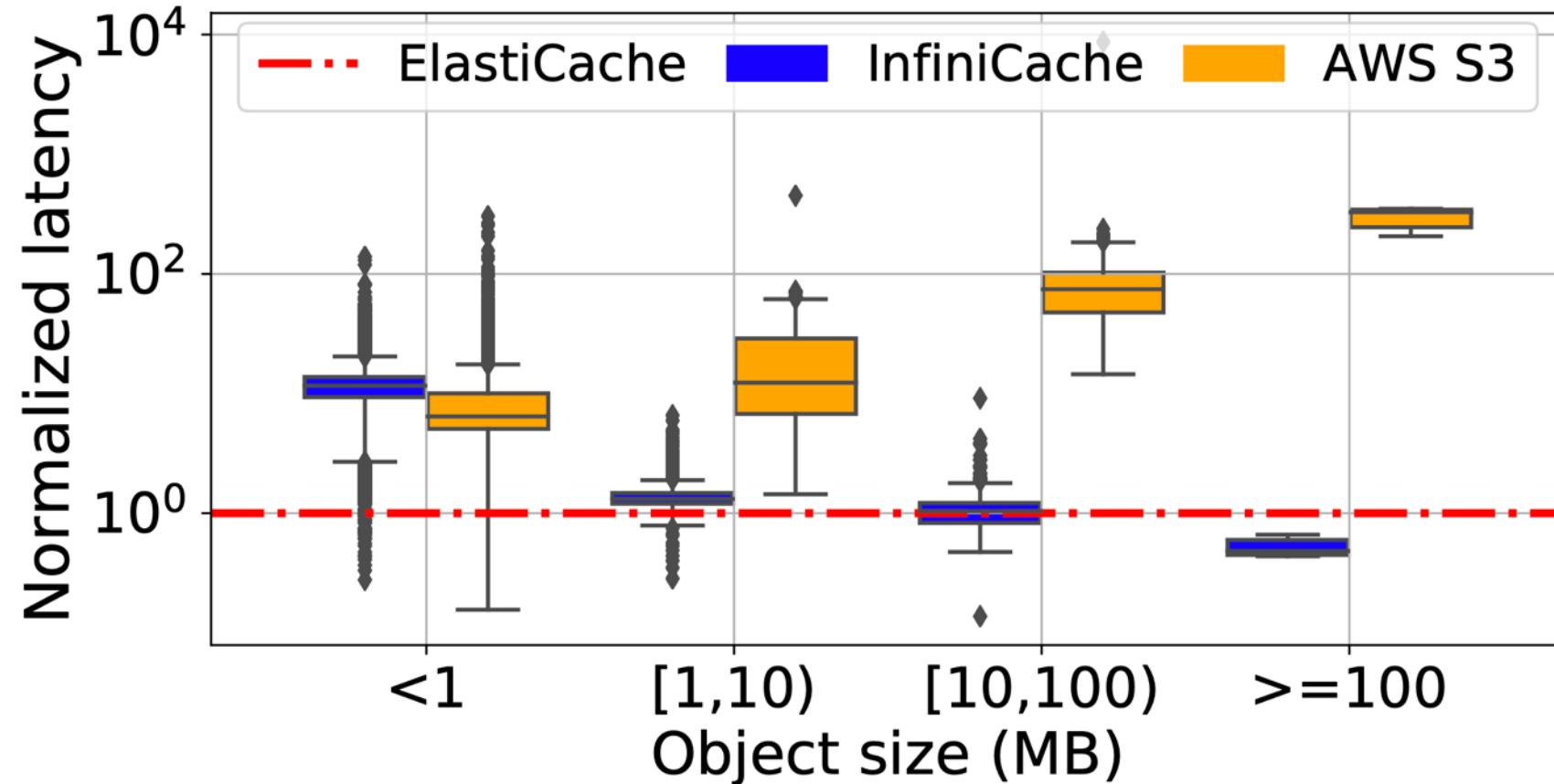


## Workload setup

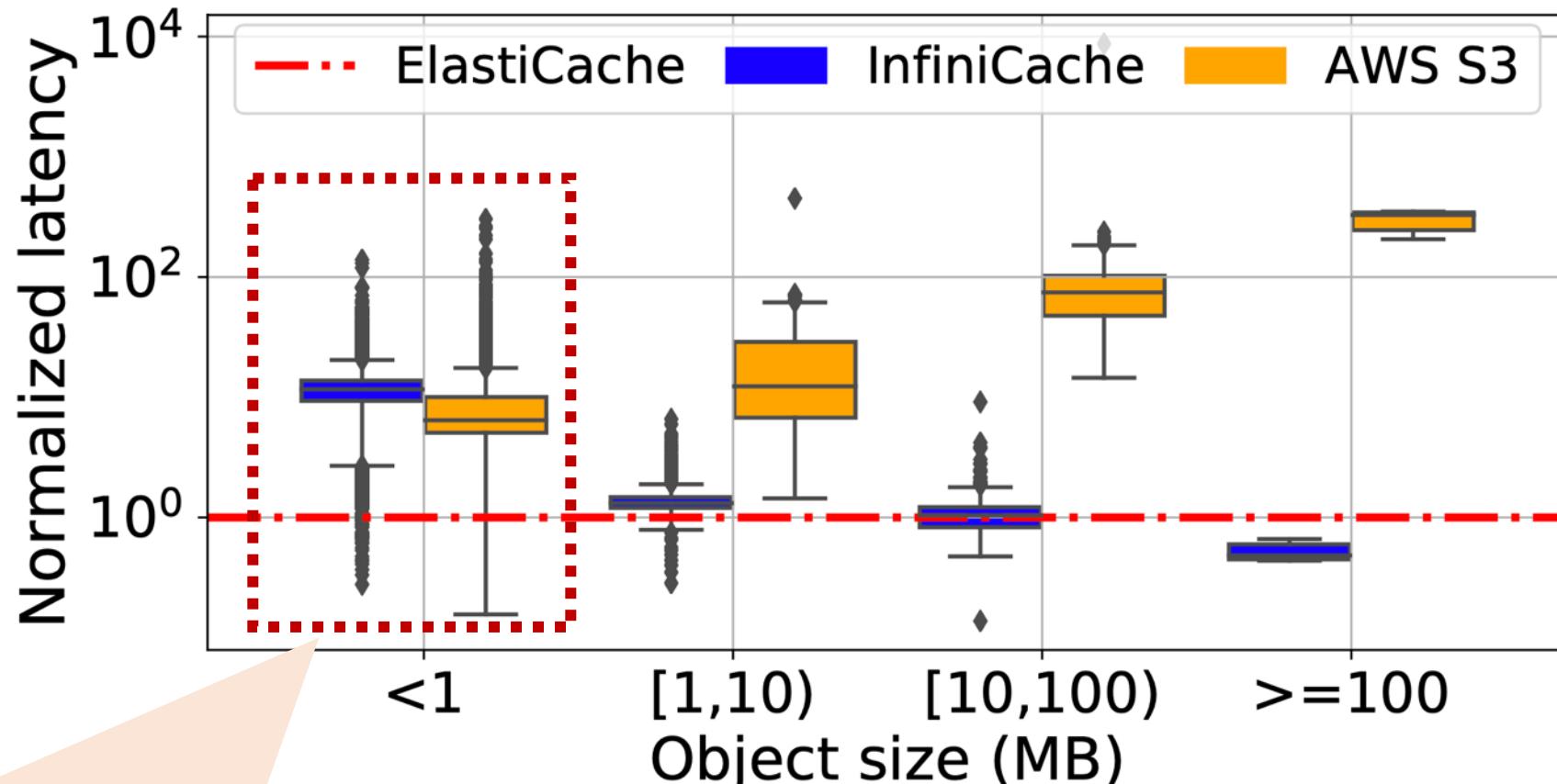
- All objects
- Large object only
  - Object larger than 10MB
- Large object w/o backup

InfiniCache is 31-96x cheaper than ElastiCache because tenants do not pay when Lambdas are not running

# Performance of InfiniCache

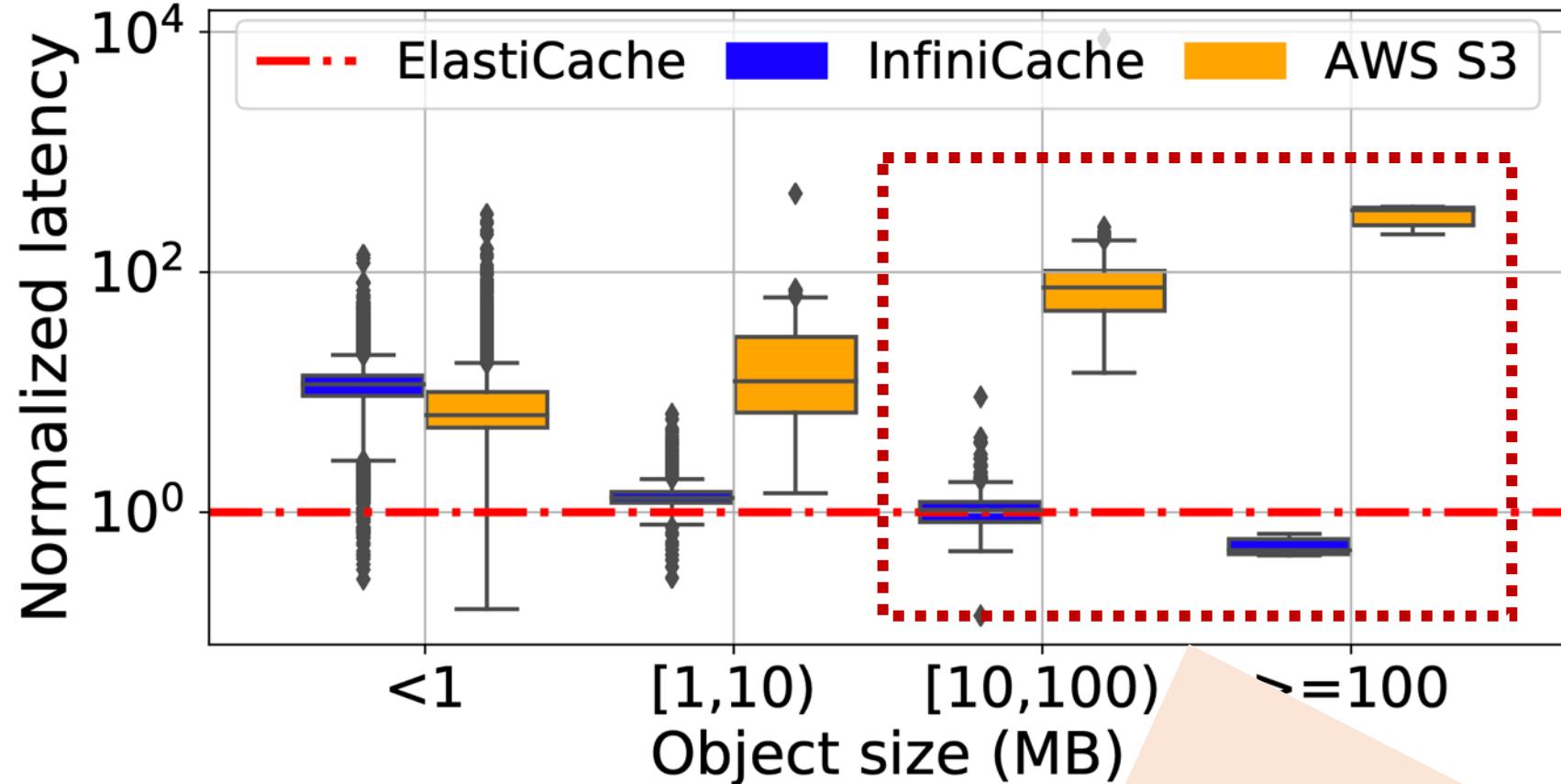


# Performance of InfiniCache



Lambda invocation overhead (~13ms)  
dominates when fetching small objects

# Performance of InfiniCache



**InfiniCache achieves same or higher performance than ElastiCache for large objects**