Fast, Elastic Storage for the Cloud

Ana Klimovic

Stanford University

January 2019

The amount of **data** we are **generating** and **analyzing** is growing **exponentially**

The amount of **data** we are **generating** and **analyzing** is growing **exponentially**

We are storing 30% to 40% more data every year

→ data stored is doubling every 2 to 3 years!

Source: “Drowning in Data.” IEEE Spectrum, September 2018.

The amount of **data** we are **generating** and **analyzing** is growing **exponentially**

We are not just storing, we are analyzing data.

→ 5 ZB of data is predicted to be analyzed in 2025.

Source: International Data Corporation, 2017.

Users are increasingly storing and processing data in the **cloud**

Cloud computing offers high…

Elasticity

Cost-efficiency

Scalability

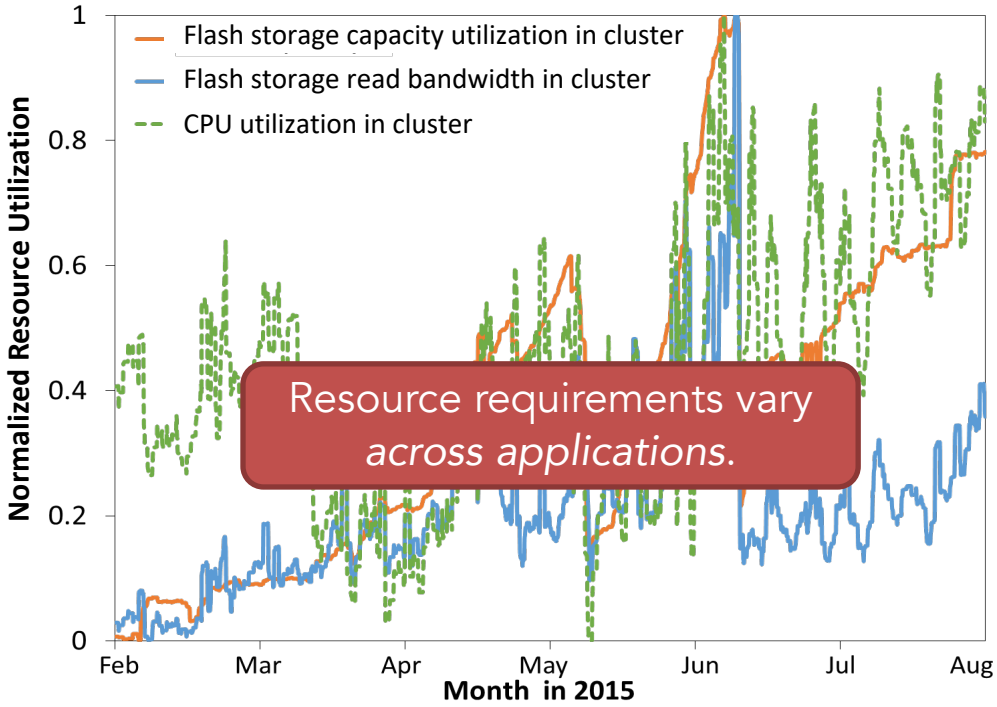
Performance

…only if each application receives the storage and compute resources it needs.

Allocating resources to achieve these goals is hard.

Why is resource allocation difficult?

Consider the resource utilization of a large-scale service at Facebook, normalized over 6 months: [EuroSys'16]

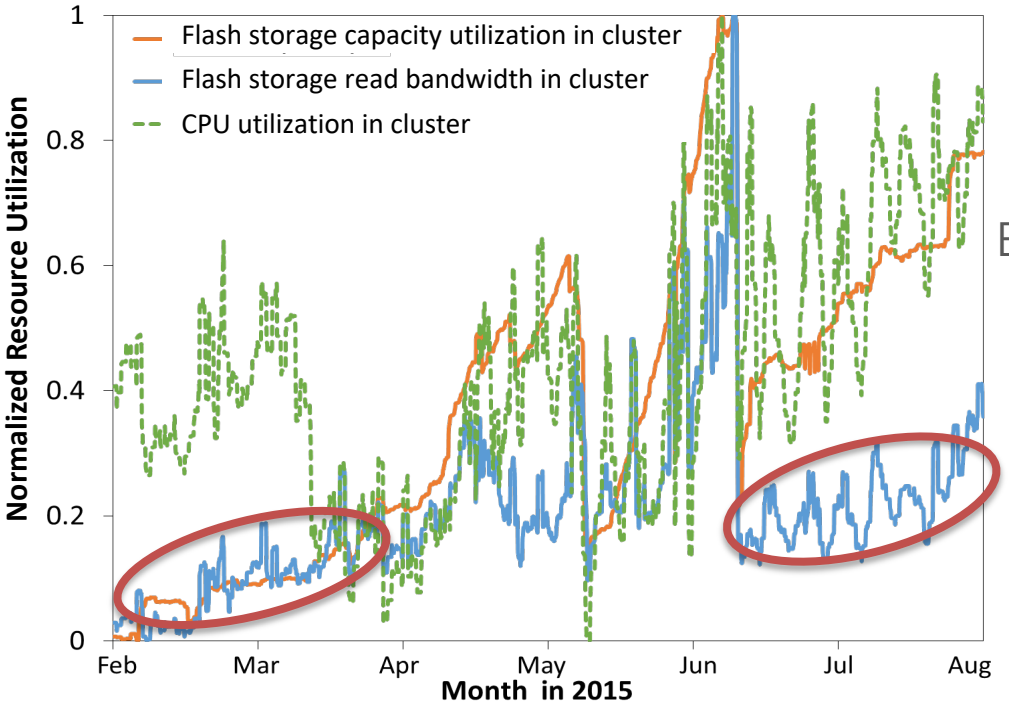


Resource requirements vary *across applications.*

Compute and storage requirements vary dynamically *over time.*

Why is resource allocation difficult?

Consider the resource utilization of a large-scale service at Facebook, normalized over 6 months: [EuroSys'16]



Compute and storage requirements are often *uncorrelated*.

But servers in a cloud facility have fixed ratios of compute and storage.

→ **Lack of flexibility leads to *imbalanced resource usage***

Flash storage is **underutilized**.

At datacenter scale...

~10,000s of servers
~10s of Megawatts

...underutilizing resources
is extremely wasteful.

Huge opportunity to improve resource utilization!
Challenge: maintain high performance

How should we build **resource-efficient** and **high performance** storage systems for diverse, elastic applications in the cloud?

Key insights

Decouple storage from compute resources

- Improve resource allocation flexibility by enabling fast, predictable access to remote data
  - Efficiently integrate network & storage software processing
  - Introduce token-based I/O scheduling for quality of service

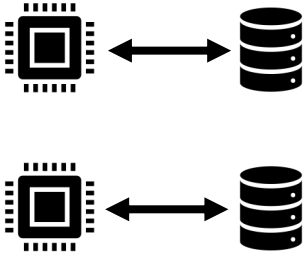
Automate storage resource management

- Allocate resources dynamically based on application requirements, learned via hints or a machine learning model

Decouple storage from compute

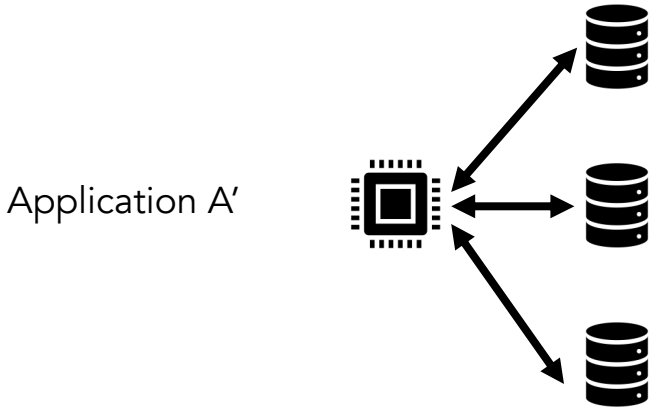
Provide **flexibility** to allocate as many CPU cores an application needs, independent of its storage capacity and storage throughput requirements.

Application A



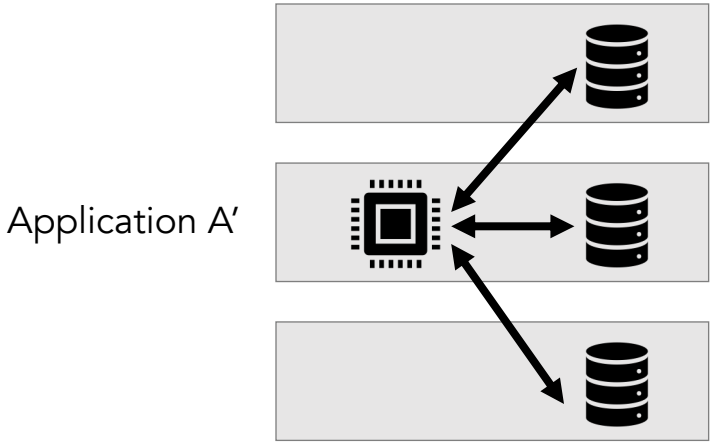
Decouple storage from compute

Provide **flexibility** to allocate as many CPU cores an application needs, independent of its storage capacity and storage throughput requirements.



Decouple storage from compute

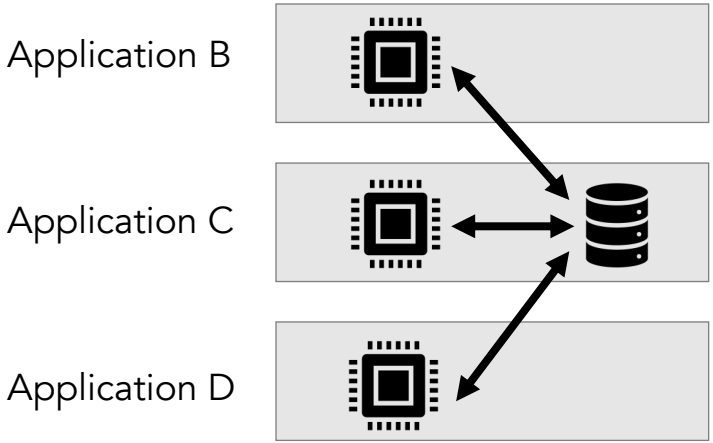
Provide **flexibility** to allocate as many CPU cores an application needs, independent of its storage capacity and storage throughput requirements.



Goal: enable any CPU to use any storage device in a cloud facility that has spare capacity & bandwidth.

Share storage to improve utilization

Share storage devices among applications to increase utilization and improve **cost-efficiency**.



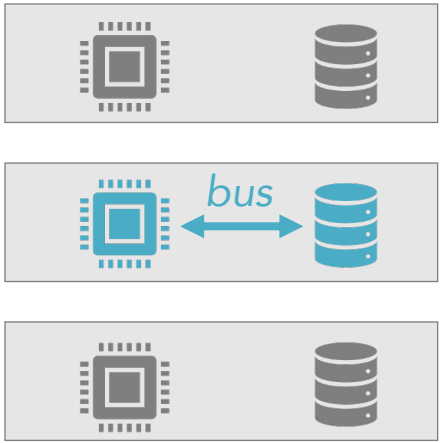
Goal: enable any CPU to use any storage device in a cloud facility that has spare capacity & bandwidth.

Requirement #1: Fast access to remote data

To improve resource allocation flexibility while maintaining high performance, we need **fast, predictable** access to remote data.

Requirement #1: Fast access to remote data

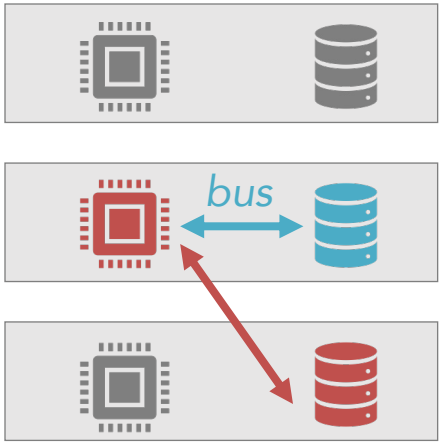
To improve resource allocation flexibility while maintaining high performance, we need **fast, predictable** access to remote data.



Local access to storage

Requirement #1: Fast access to remote data

To improve resource allocation flexibility while maintaining high performance, we need **fast, predictable** access to remote data.

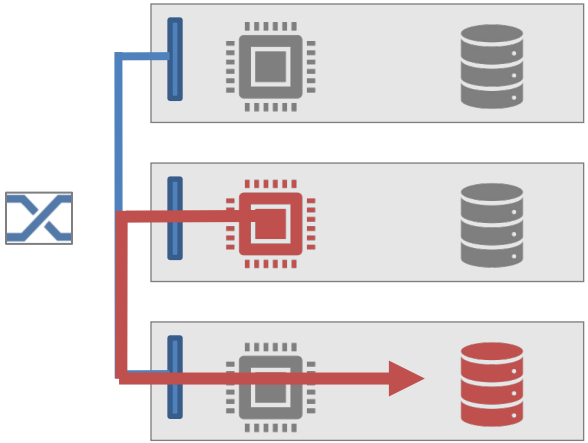


Local access to storage

Remote access to storage

Requirement #1: Fast access to remote data

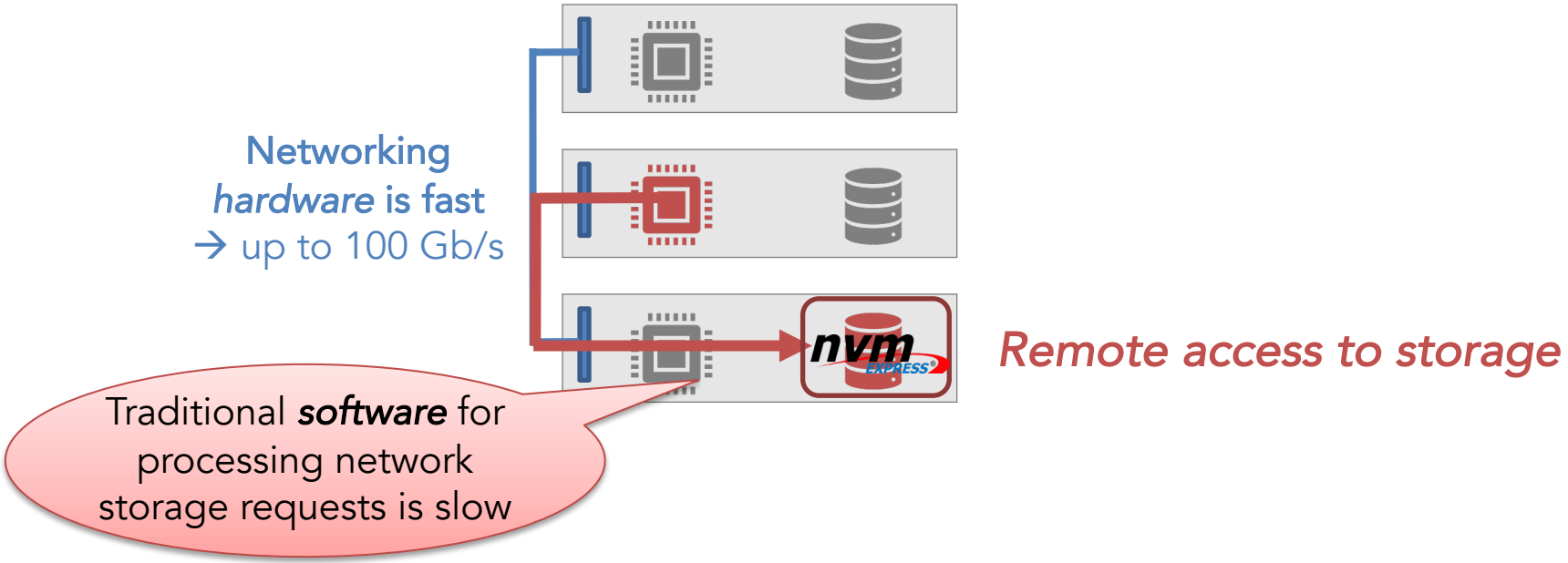
To improve resource allocation flexibility while maintaining high performance, we need **fast, predictable** access to remote data.



Remote access to storage

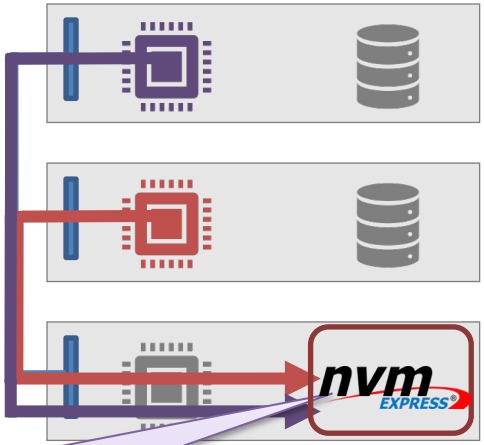
Requirement #1: Fast access to remote data

To improve resource allocation flexibility while maintaining high performance, we need **fast, predictable** access to remote data.



Requirement #1: Fast access to remote data

To improve resource allocation flexibility while maintaining high performance, we need **fast, predictable** access to remote data.



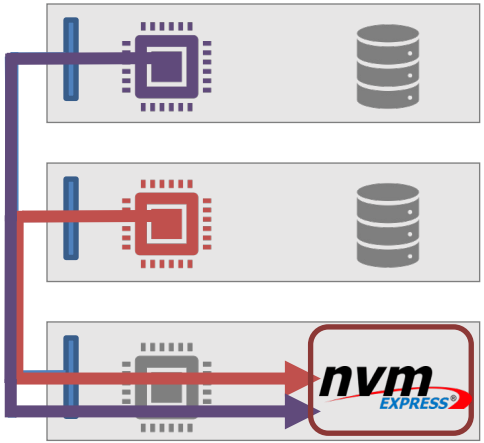
shared

Remote access to storage

Request interference can cause unpredictable performance

Requirement #1: Fast access to remote data

To improve resource allocation flexibility while maintaining high performance, we need **fast, predictable** access to remote data.

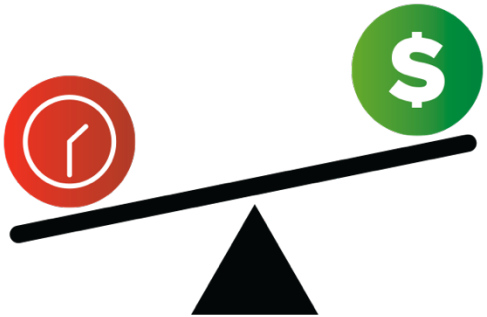


How do we decide which resources to allocate to each application and where to place data?

Automate storage resource management

- Automatically decide how many storage devices to allocate, where to place data, and on what type of storage technology

- Dynamically adjust allocations as application load varies



Requirement #2: Learn application resource requirements to automate resource allocation

Learn the application's characteristics (e.g., capacity, throughput, latency requirements) and translate to a performance-cost efficient resource allocation.

- Leverage high-level hints about application characteristics

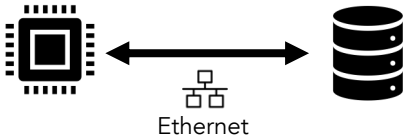
→ use heuristic policies to decide resource allocation

- Leverage performance data collected across application runs

→ build a machine learning model that recommends a near-optimal resource allocation

Two major requirements for cloud storage

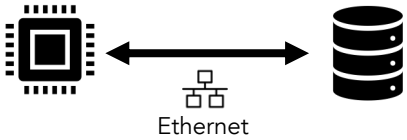
1. Fast access to remote data



2. Automatic, elastic allocation of storage resources

Contributions

1. Fast access to remote data



2. Automatic, elastic allocation of storage resources

[OSDI'14] **IX**: Dataplane OS for Fast Networking

[EuroSys'16] Flash Storage Disaggregation

[HotStorage'17] Rack-Scale Disaggregated Storage

[ASPLOS'17] **ReFlex**: Remote Flash ≈ Local Flash

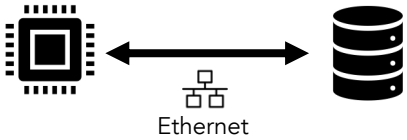
[ATC'18a] Understanding Ephemeral Storage for Serverless Analytics

[OSDI'18] **Pocket**: Elastic Ephemeral Storage for Serverless Analytics

[ATC'18b] **Selecta**: ML-based Heterogeneous Cloud Storage Configuration

Focus of this talk

1. Fast access to remote data



2. Automatic, elastic allocation of storage resources

[OSDI'14] **IX**: Dataplane OS for Fast Networking

[EuroSys'16] Flash Storage Disaggregation

[HotStorage'17] Rack-Scale Disaggregated Storage

[ASPLOS'17] **ReFlex**: Remote Flash ≈ Local Flash

[ATC'18a] Understanding Ephemeral Storage for Serverless Analytics

[OSDI'18] **Pocket**: Elastic Ephemeral Storage for Serverless Analytics

[ATC'18b] **Selecta**: ML-based Heterogeneous Cloud Storage Configuration

Fast access to remote flash

Flash storage

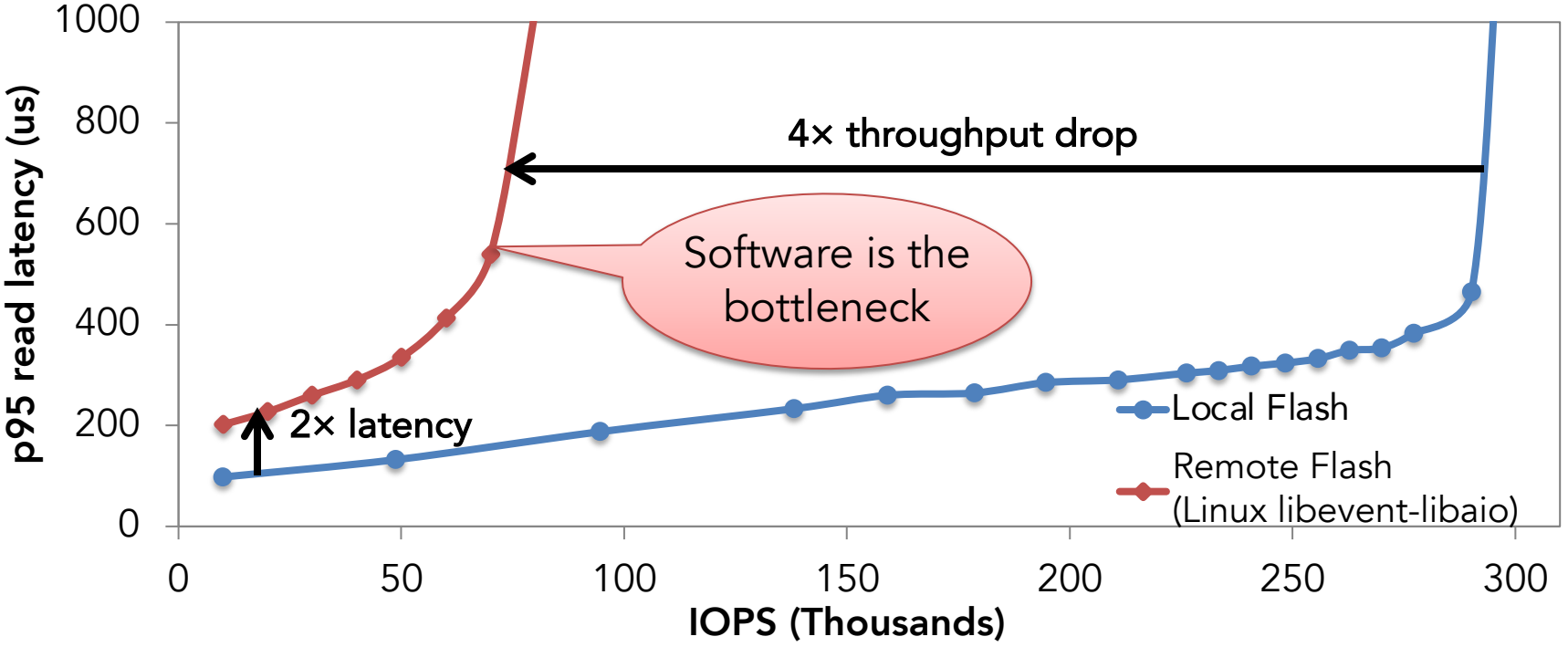
NVMe Flash:

- 70 µs read latency → 100x faster than disk
- 1,000,000 IOPS → 1000x higher than disk

What if we use existing remote storage approaches to access remote flash?

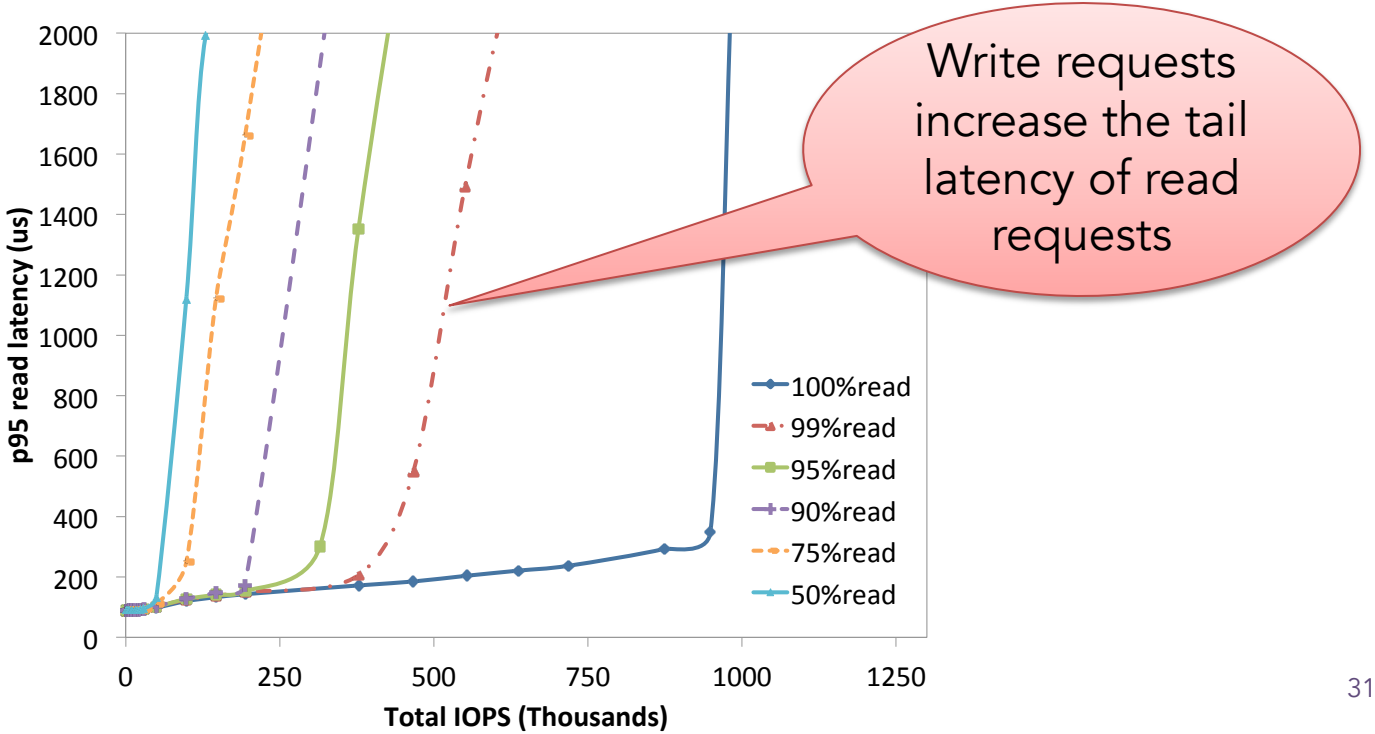
Issue #1: Existing software is too slow

Existing remote storage solutions have significant overheads.



Issue #2: Unpredictable performance on shared flash

Write requests from one tenant can **interfere** with read requests from another tenant, leading to unpredictable performance on shared flash



ReFlex: Remote Flash ≈ Local Flash

A software system for fast, predictable access to remote flash storage

ReFlex: Remote Flash ≈ Local Flash

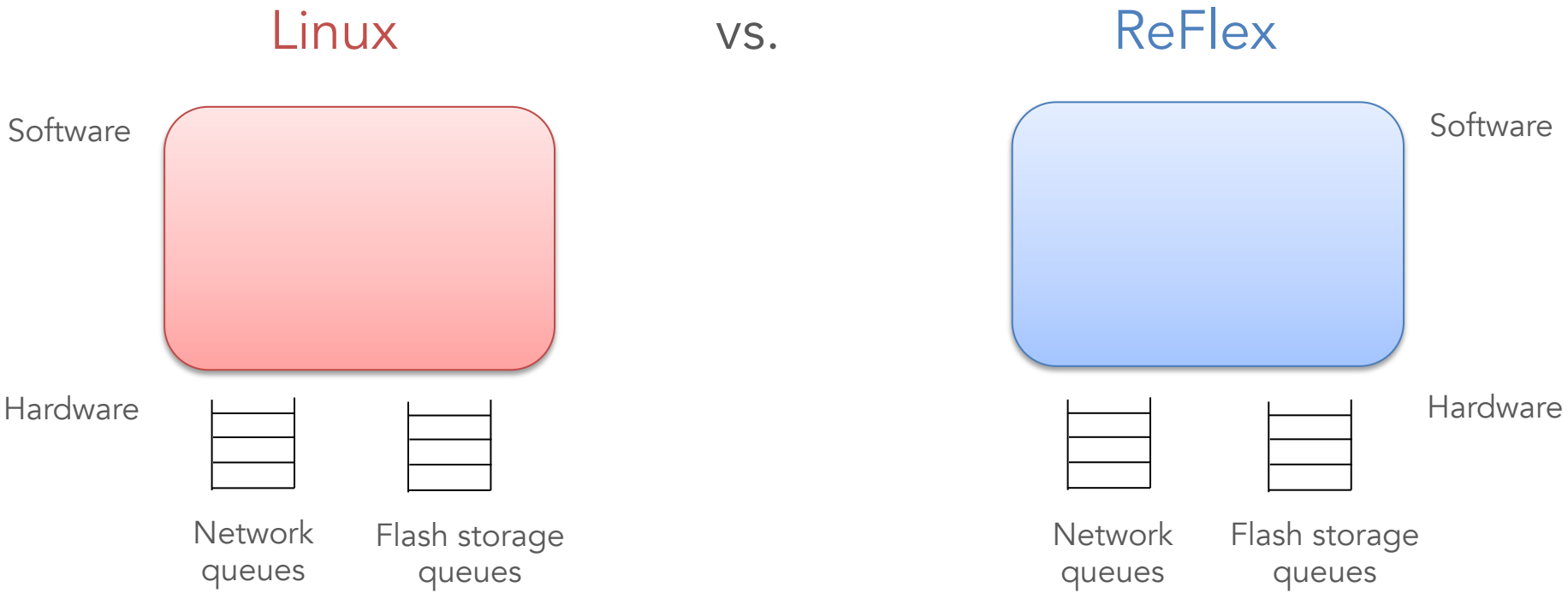
A software system for fast, predictable access to remote flash storage

Issue #1: Existing software is too slow

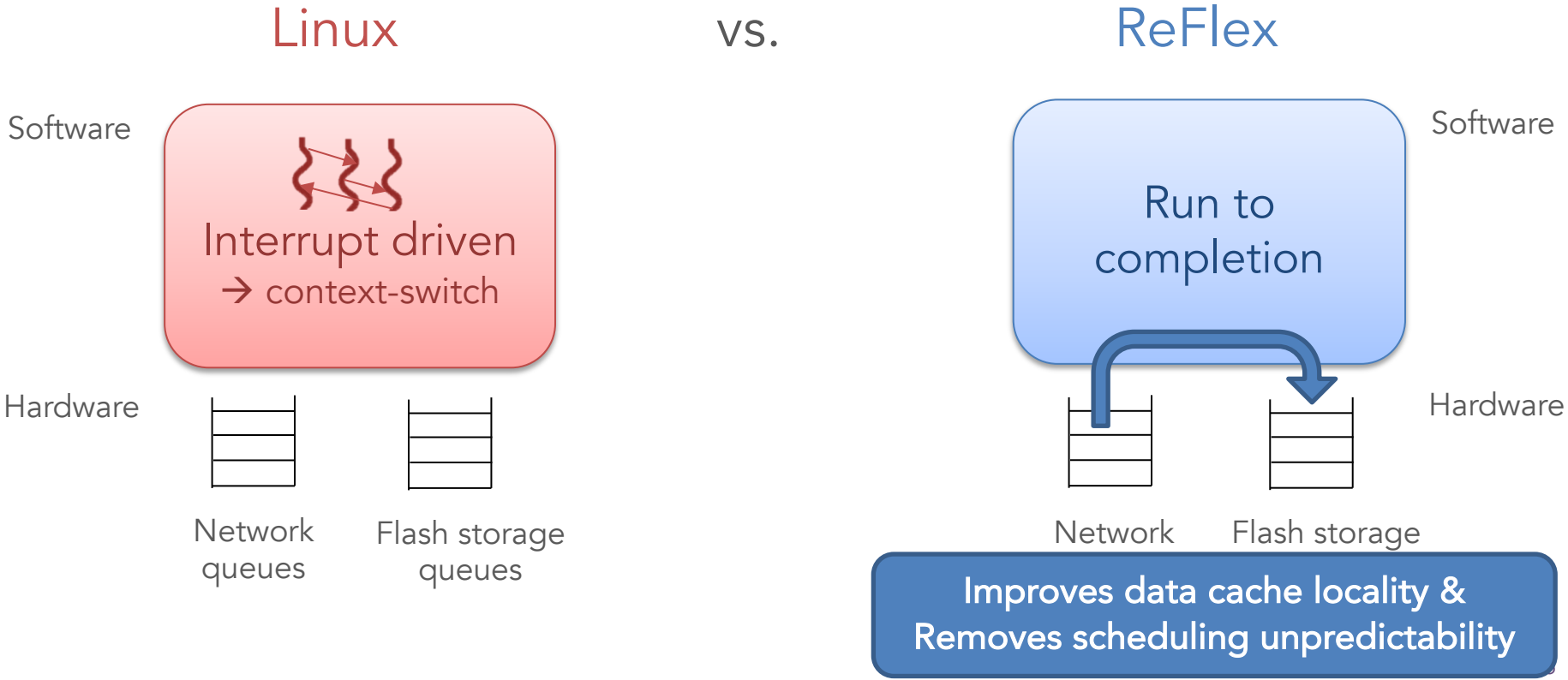
→ Custom network-storage OS, designed for low latency and high throughput

- ✓ Run to completion
- ✓ Adaptive batching
- ✓ Reduce data copies
- ✓ Direct access to hardware

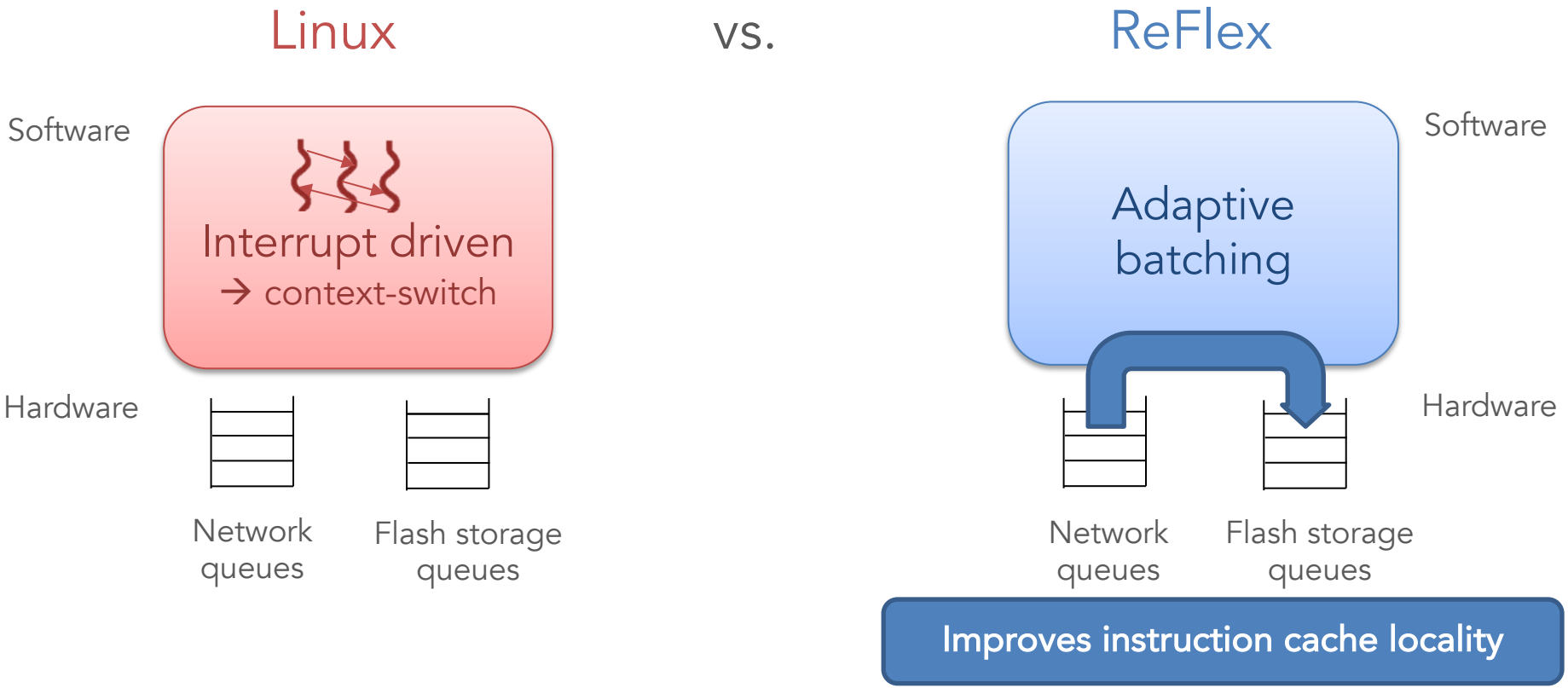
How does ReFlex achieve high performance?



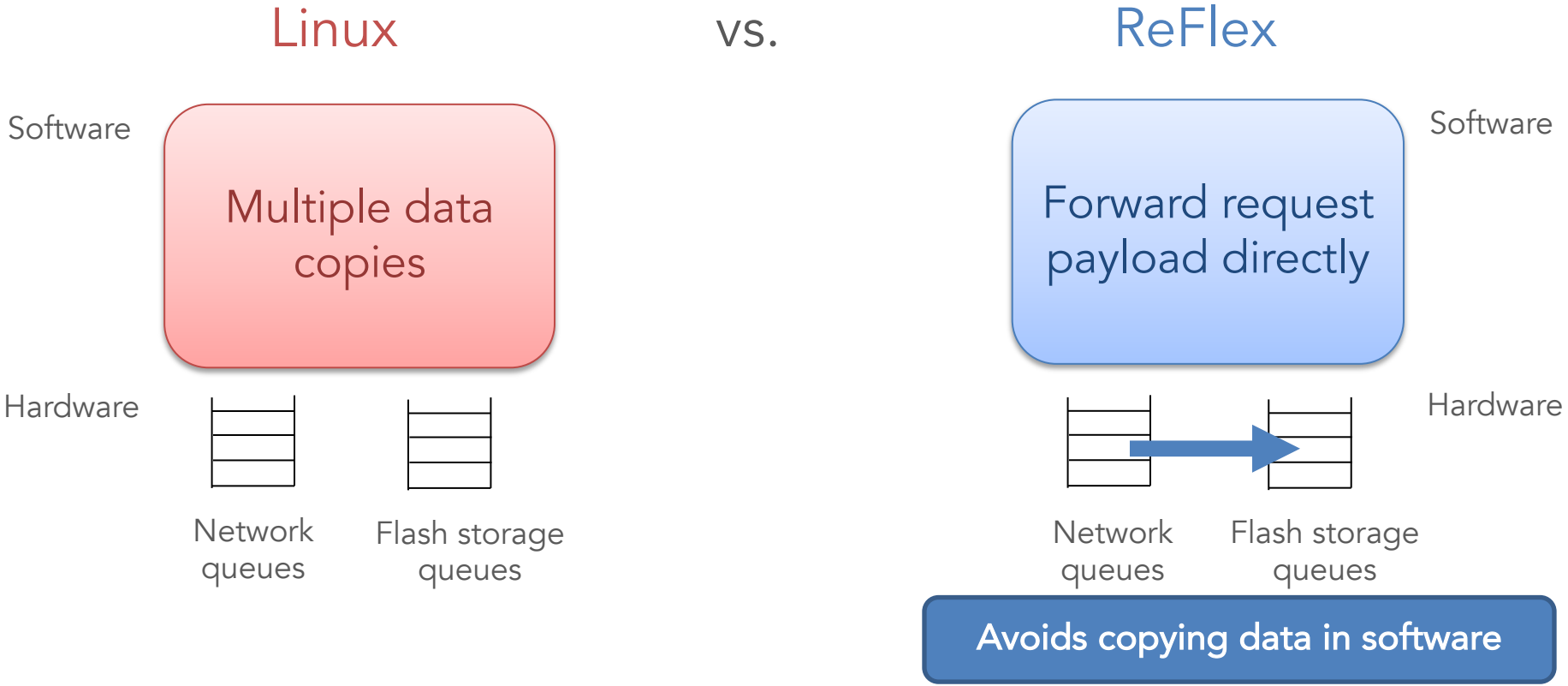
How does ReFlex achieve high performance?



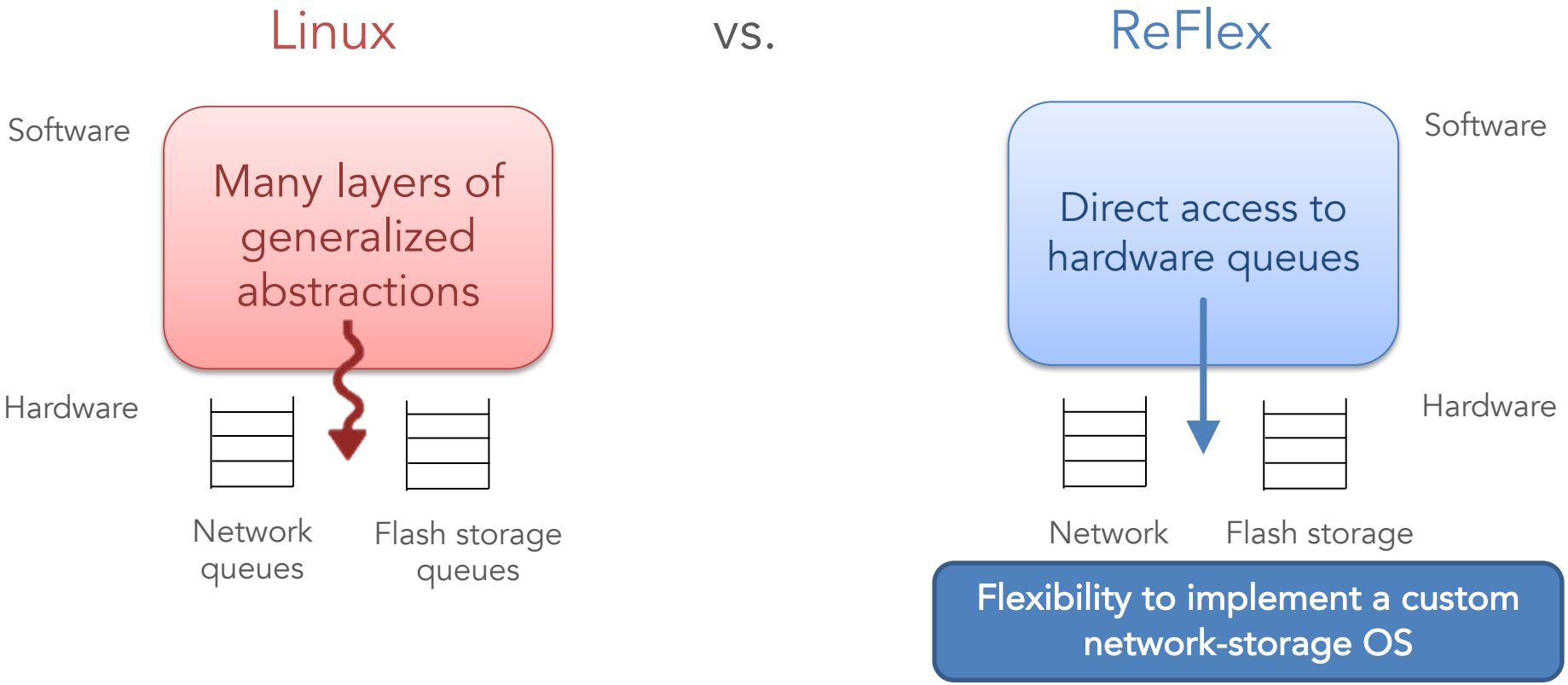
How does ReFlex achieve high performance?



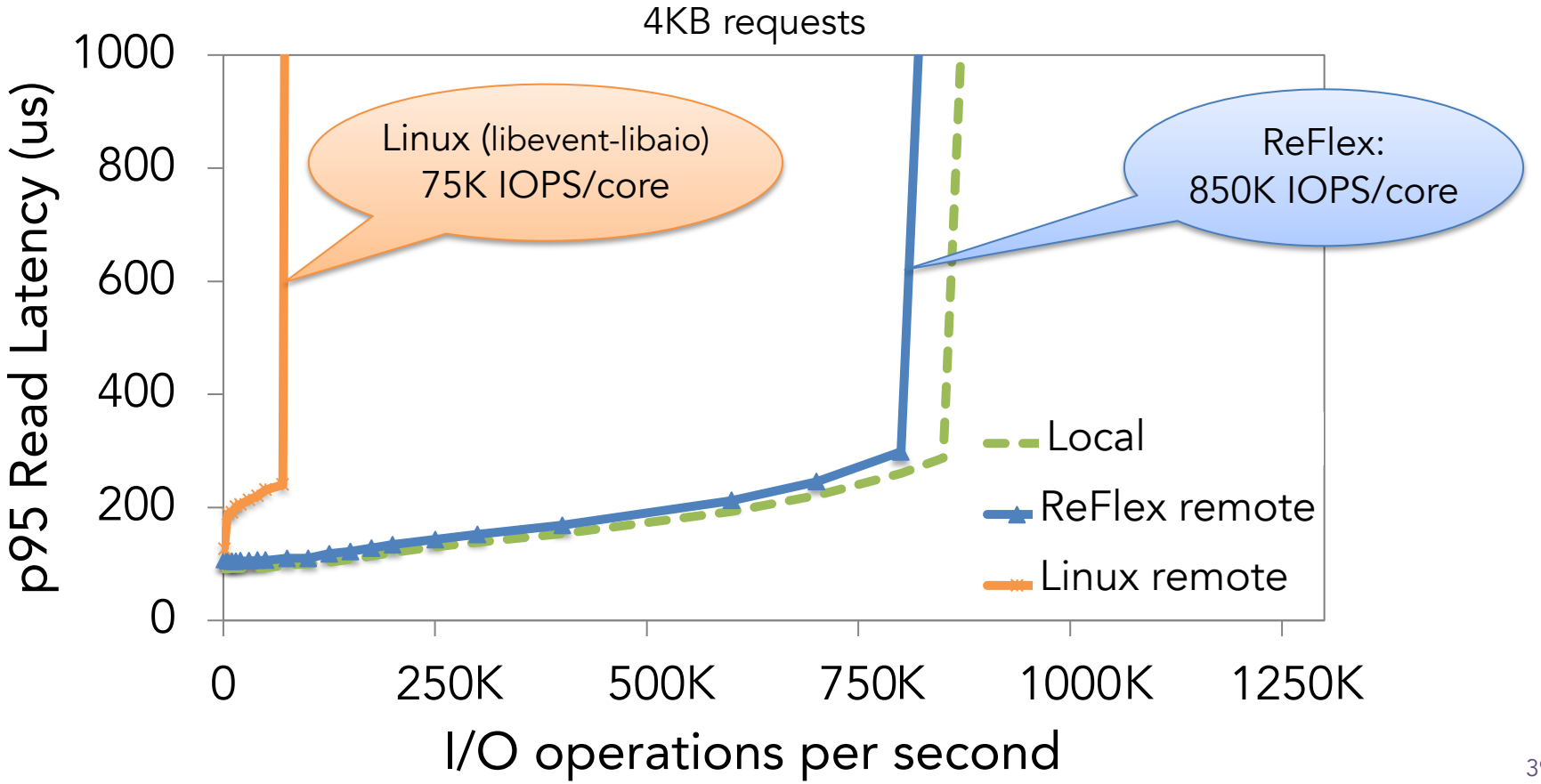
How does ReFlex achieve high performance?



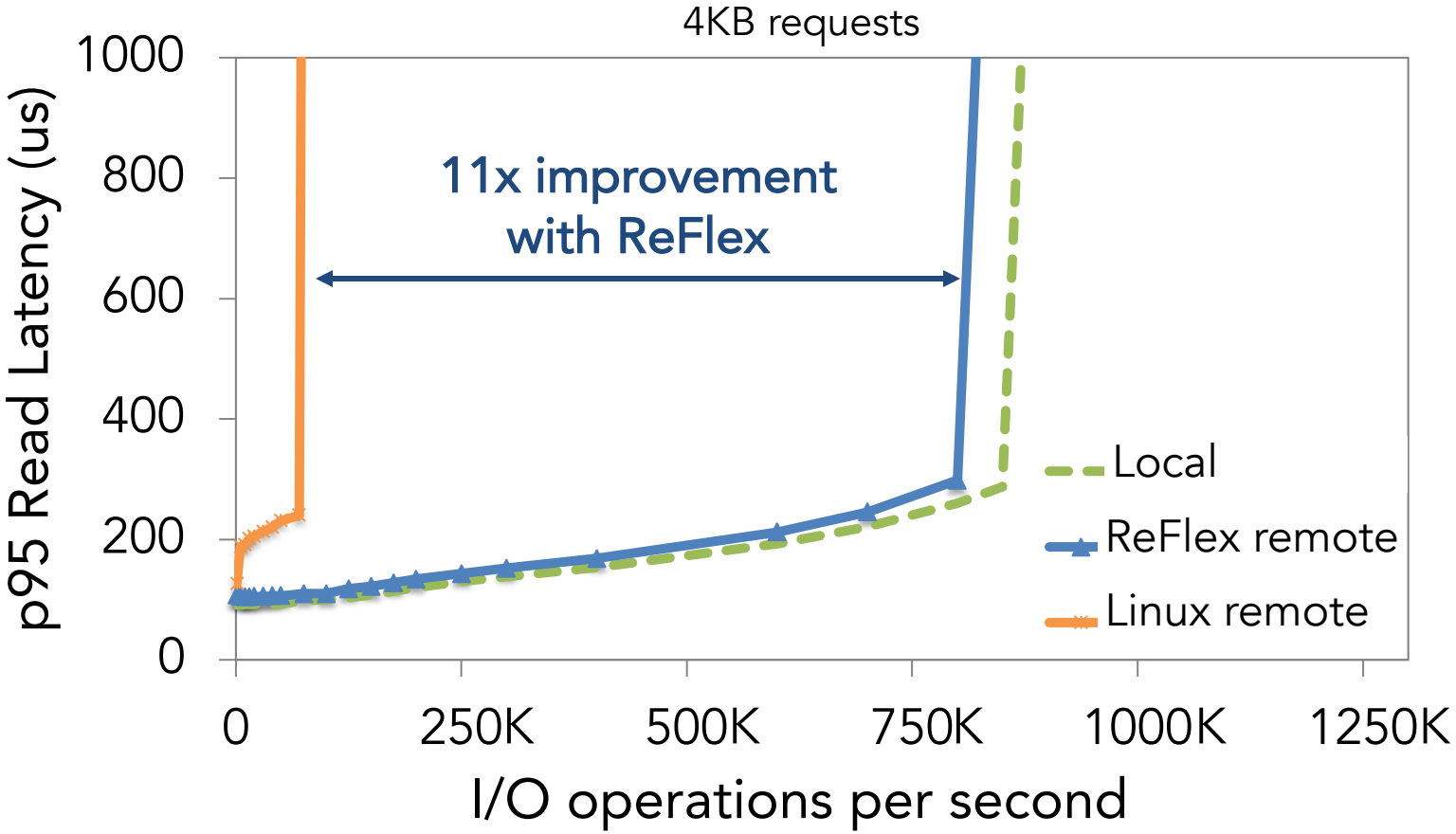
How does ReFlex achieve high performance?



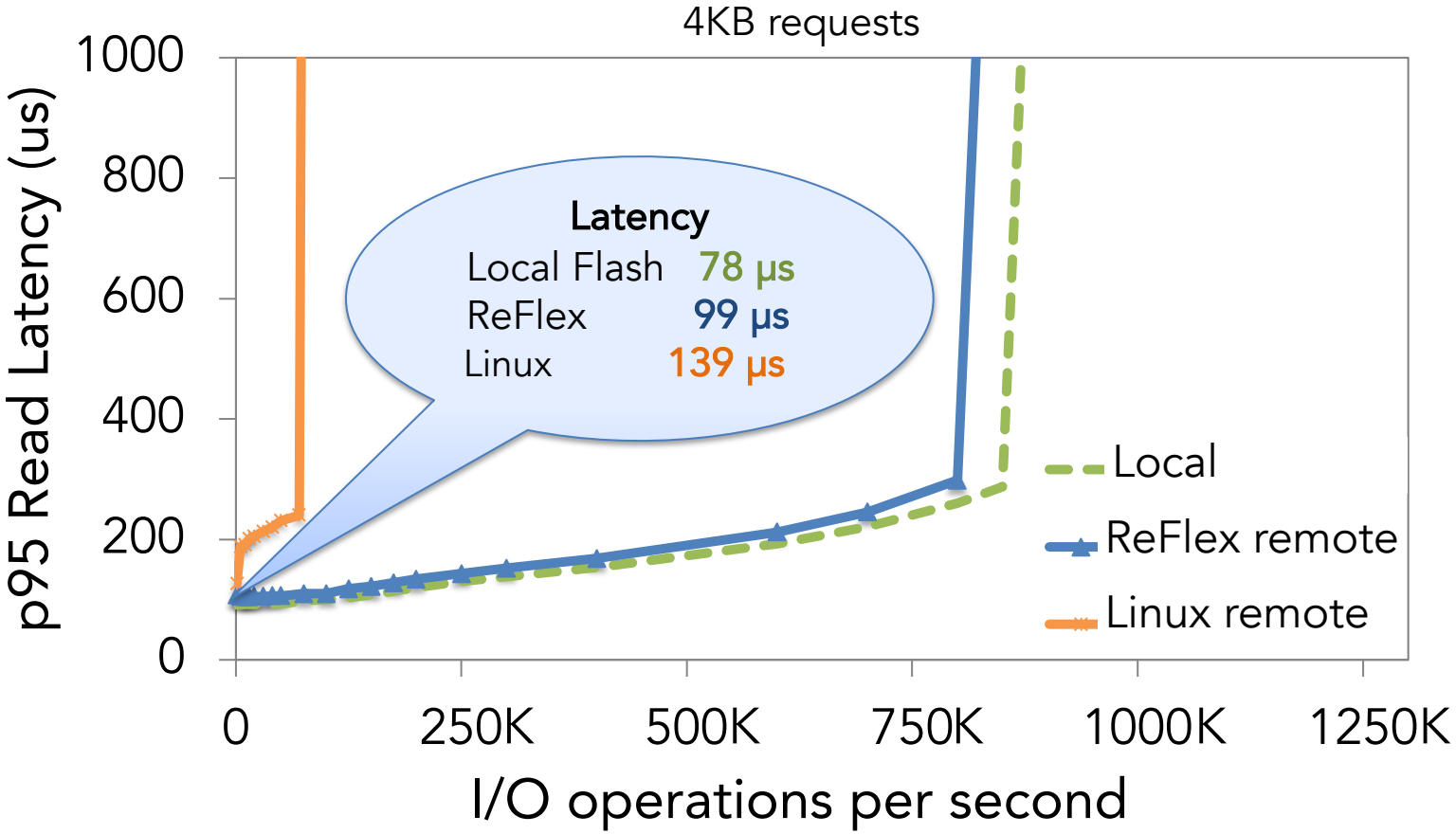
ReFlex performance: throughput per core



ReFlex performance: throughput per core



ReFlex performance: latency



ReFlex: Remote Flash ≈ Local Flash

A software system for fast, predictable access to remote flash storage

Issue #1: Existing software is too slow

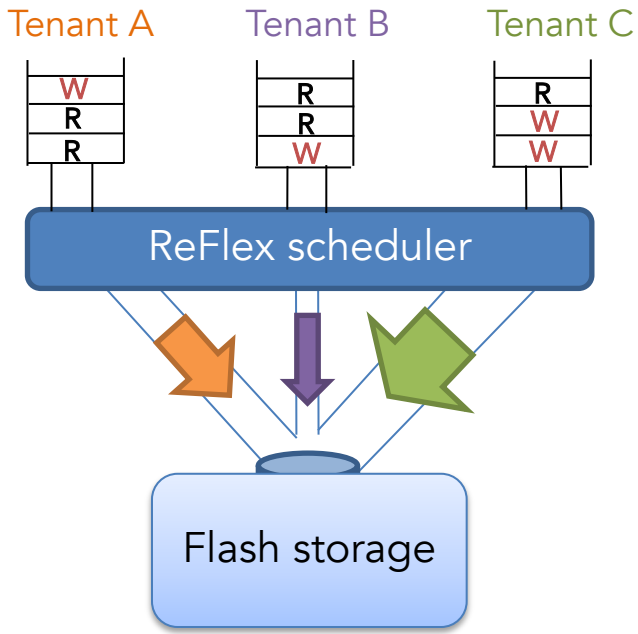
→ Custom network-storage OS, designed for low-latency and high throughput

Issue #2: Unpredictable performance on shared flash storage

→ Novel I/O scheduler provides tail latency and throughput guarantees

How does ReFlex provide predictable performance?

ReFlex schedules requests to satisfy the performance objectives specified by each tenant.



Goal: Provide **tail latency** and **throughput** guarantees to tenants sharing flash.

Challenge: Account for the performance properties of different request types (e.g., read vs. write, 4 KB vs. 1 MB request)

Solution: profile device to derive a request cost model that captures how each type of request impacts tail latency and throughput

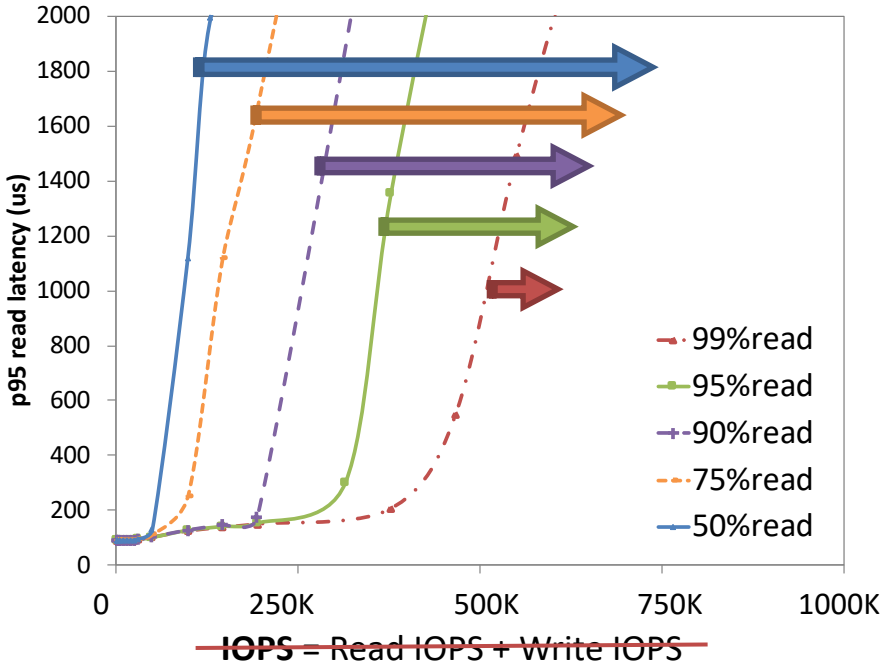
How does ReFlex provide predictable performance?

Step 1: Build a request cost model

→ account for different request types

For this device,

write cost = 10 x read cost



~~**IOPS** = Read IOPS + Write IOPS~~

Weighted IOPS = (1 x Read IOPS) + (10 x Write IOPS)

How does ReFlex provide predictable performance?

Step 1: Build a request cost model
→ account for different request types

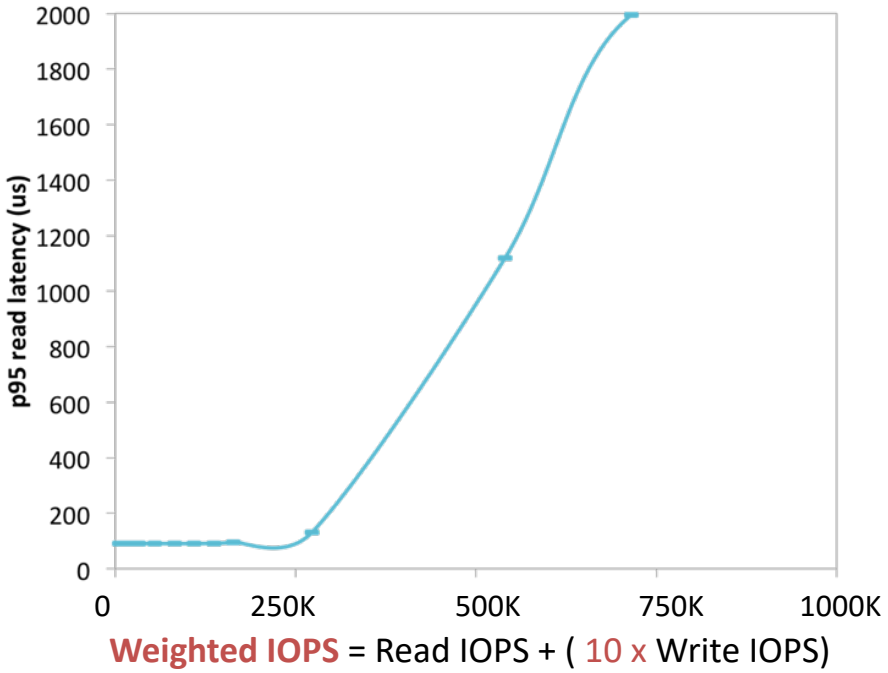
Step 2: Schedule requests

Example scenario

Tenant A:
- 1ms tail latency
- 200K IOPS

Tenant B:
- Best-effort (use slack)

For this device,
write cost = 10 x read cost



How does ReFlex provide predictable performance?

Step 1: Build a request cost model
→ account for different request types

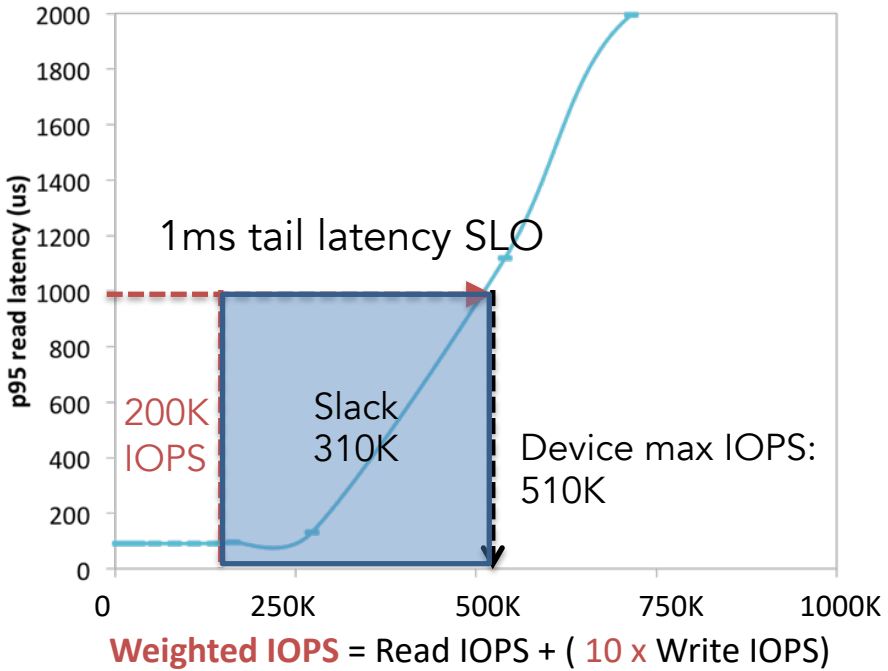
Step 2: Schedule requests

Example scenario

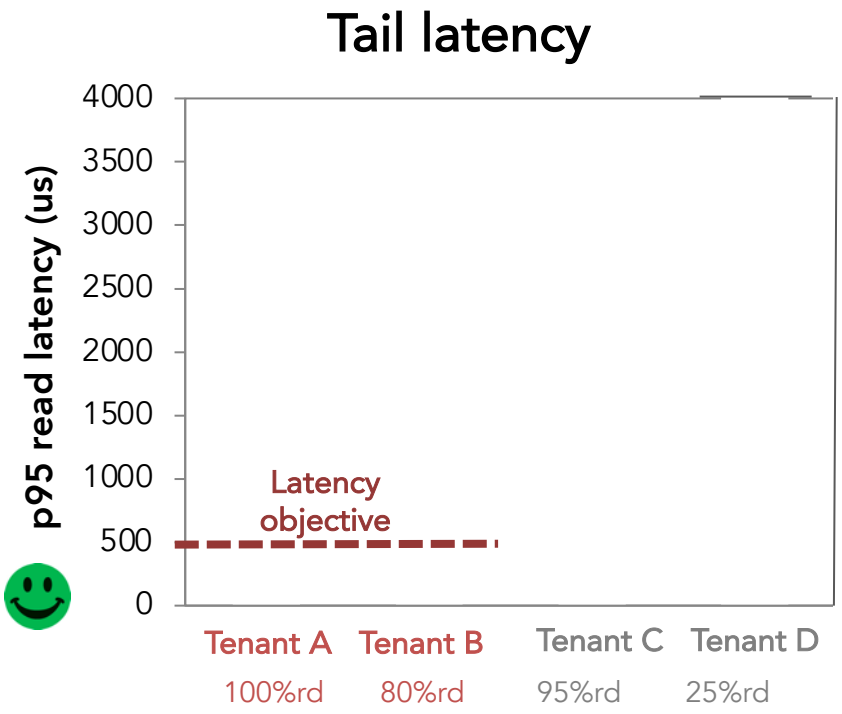
Tenant A:
- 1ms tail latency
- 200K IOPS

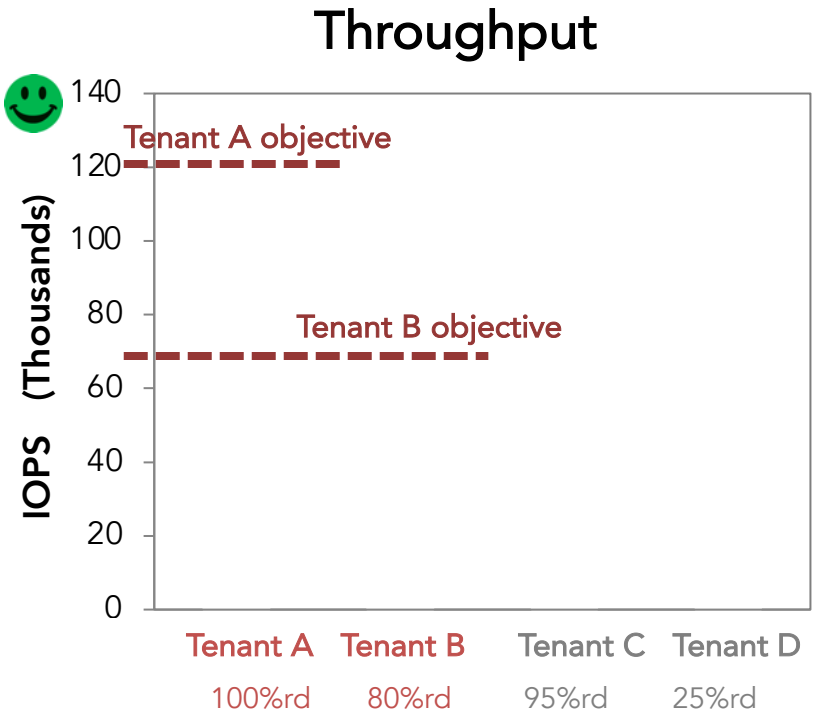
Tenant B:
- Best-effort (use slack)

For this device,
write cost = 10 x read cost



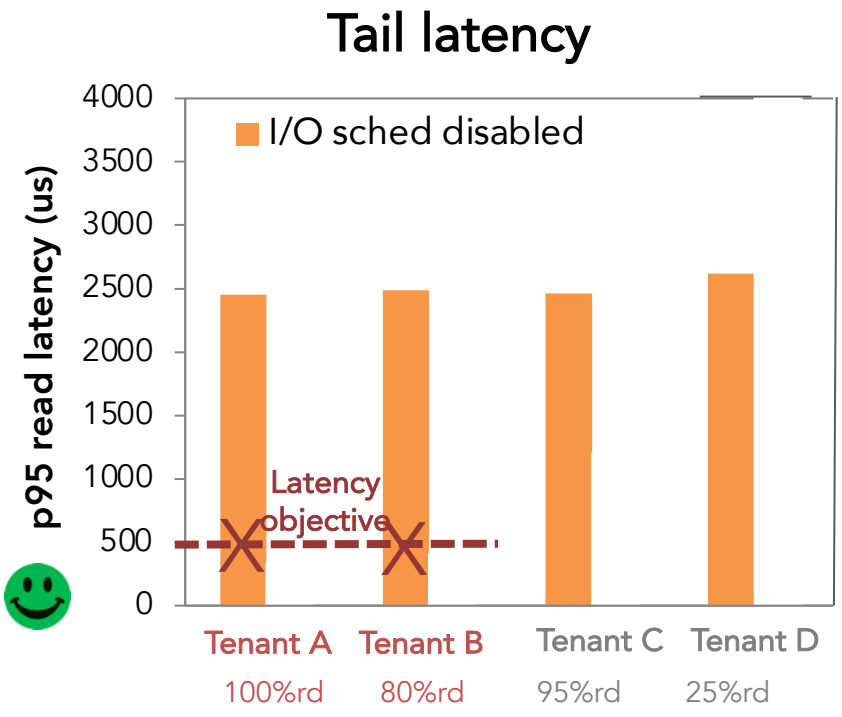
ReFlex performance guarantees on shared Flash

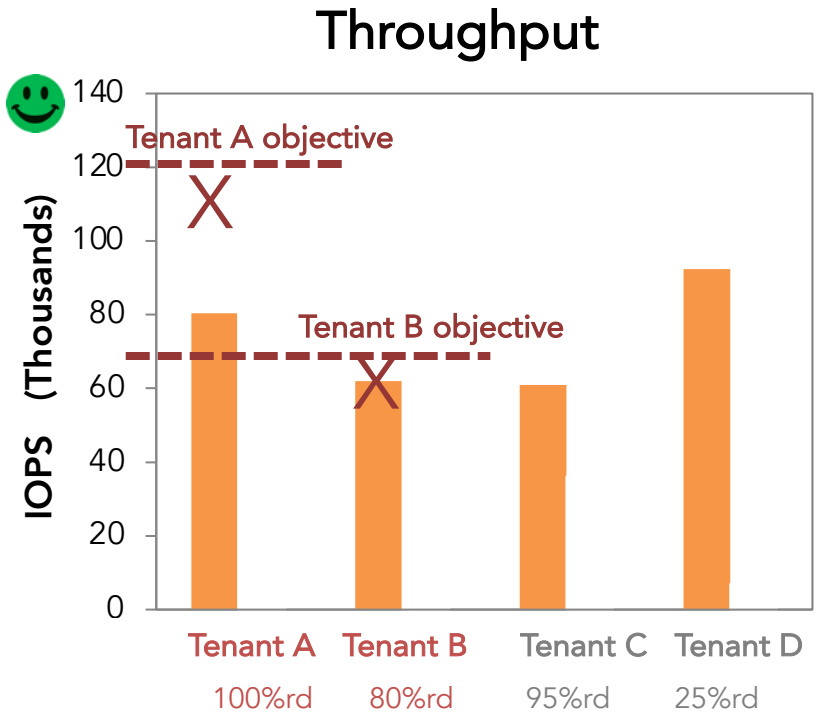




Tenants A & B: latency-critical; Tenant C + D: best effort

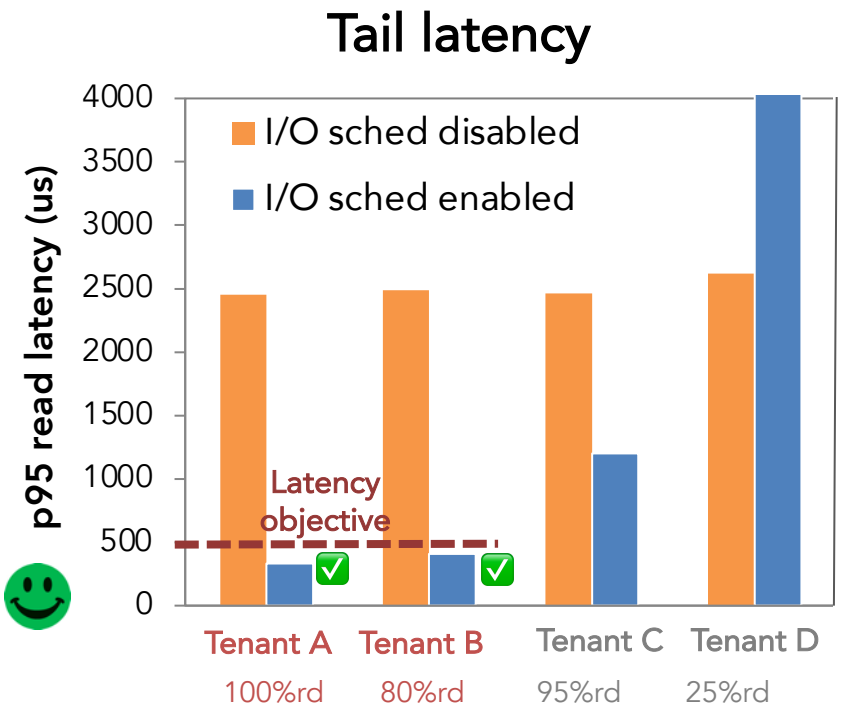
ReFlex performance guarantees on shared Flash

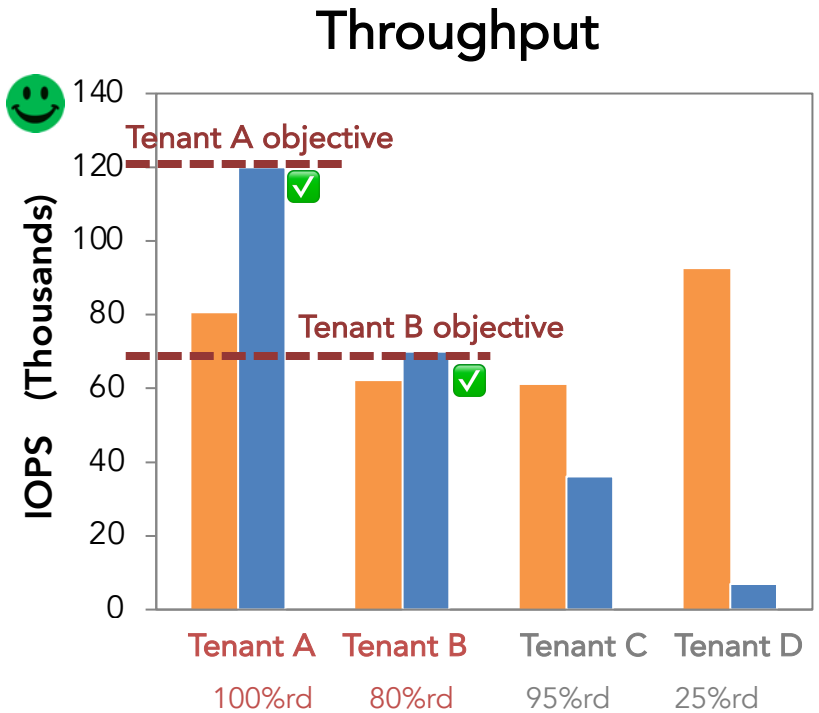




Tenants A & B: latency-critical; Tenant C + D: best effort

ReFlex performance guarantees on shared Flash





Tenants A & B: latency-critical; Tenant C + D: best effort

ReFlex impact

- ReFlex provides high throughput, low latency with the flexibility to use commodity networks

- Broadcom is porting ReFlex to a SoC platform
- ReFlex integrated into Apache Crail storage system → runs on amazon web services


THENEXTPLATFORM

HOME COMPUTE STORE CONNECT CONTROL CODE AI HPC ENTERPRISE

MAKING REMOTE NVM-EXPRESS FLASH LOOK LOCAL AND FAST

March 1, 2017 Jeffrey Burt

Large enterprises are embracing NVM-Express flash as the storage technology of choice for their data intensive and often highly unpredictable workloads. NVM-Express devices bring with them high performance – up to 1 million I/O operations per second – and low latency – less than 100 microseconds. And flash storage now has high capacity, too, making it a natural fit for such datacenter applications.

As we have discussed here before, all-flash arrays are quickly becoming mainstream, particularly within larger enterprises, as an alternative to disk drives in environments where tens or hundreds of petabytes of data – rather than the exabytes found with hyperscalers – are more common. NVM-Express will help accelerate the use of flash, and many server makers are banking on that.

BROADCOM.

crail

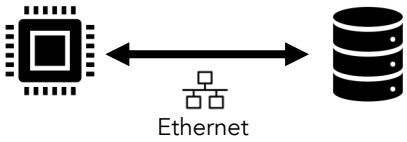
IBM Research

APACHE INCUBATOR

https://github.com/stanford-mast/reflex

Contributions

1. Fast access to remote data



[OSDI'14] **IX**: dataplane OS for fast networking

[EuroSys'16] Flash storage disaggregation

[HotStorage'17] Rack-Scale Disaggregated Storage

[ASPLOS'17] **ReFlex**: Remote Flash ≈ Local Flash

2. Automatic, elastic allocation of storage resources

[ATC'18a] Understanding Ephemeral Storage for Serverless Analytics

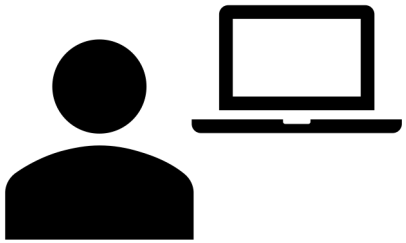
[OSDI'18] **Pocket**: Elastic Ephemeral Storage for Serverless Analytics

[ATC'18b] **Selecta**: ML-based Heterogeneous Cloud Storage Configuration

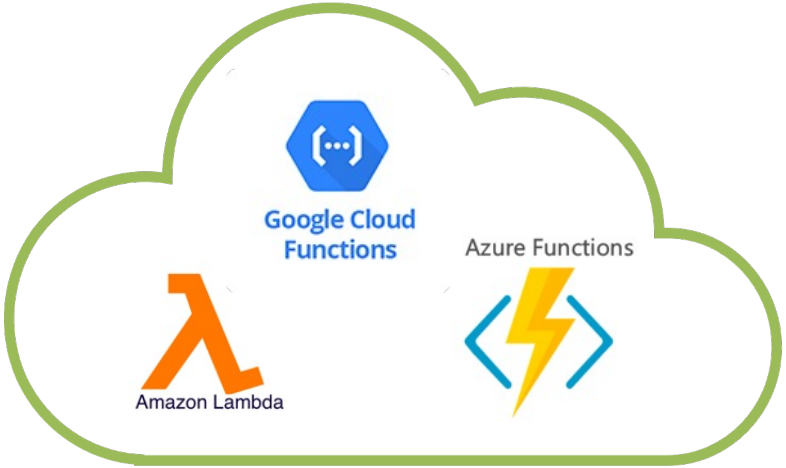
Automatic, elastic allocation of storage resources

Serverless computing

Serverless computing is a new cloud service → users can launch thousands of tiny tasks with **high elasticity** and **fine-grain billing**



Users focus on writing code for their applications
→ *no resource management*

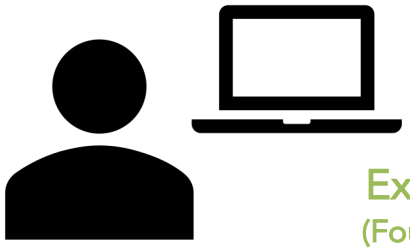


Cloud providers automatically allocate and scale resources.

Serverless analytics

Serverless computing is increasingly used for **interactive analytics**

→ exploit massive task parallelism to get a result in near real-time



gg: The Stanford Builder
(Fouladi et al.)

PyWren
(Jonas et al., SoCC'17)

ExCamera
(Fouladi et al., NSDI'17)

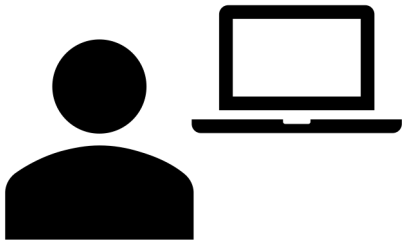
databricks
serverless

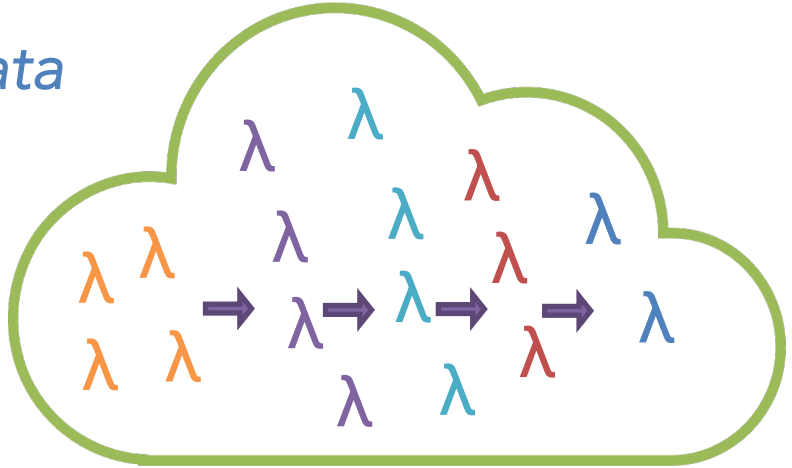
Data sharing in serverless analytics

- Analytics jobs involve multiple stages of execution

Challenge: exchange intermediate data efficiently between tasks

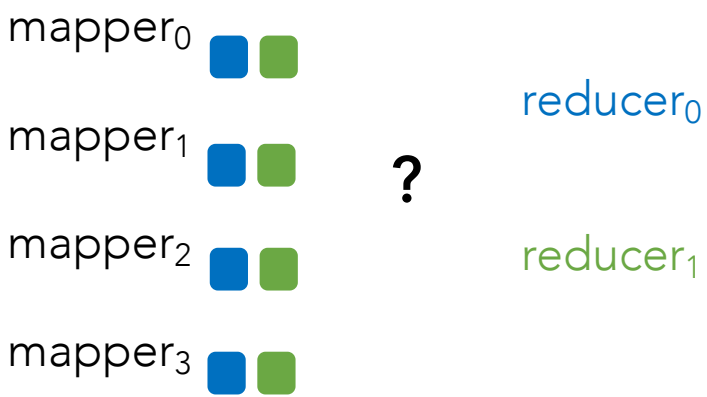
ephemeral data





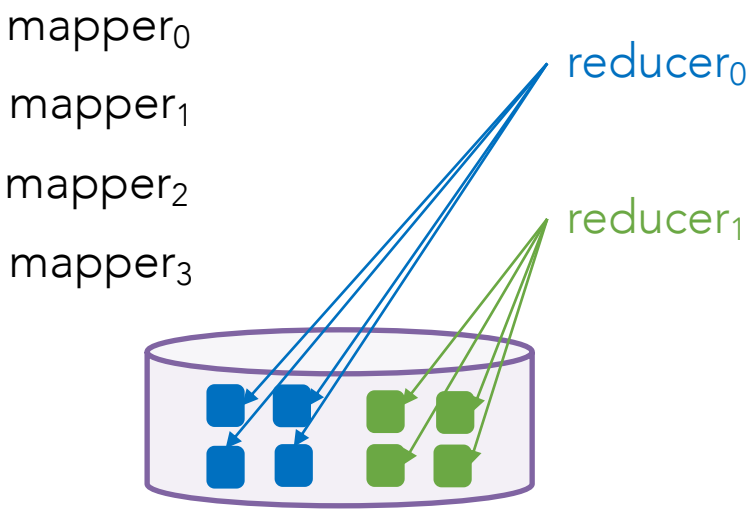
Data sharing in serverless analytics

- Direct communication between serverless tasks is difficult:
  - Tasks are short-lived and stateless



Data sharing in serverless analytics

- The natural approach for sharing ephemeral data is through a **shared remote data store**

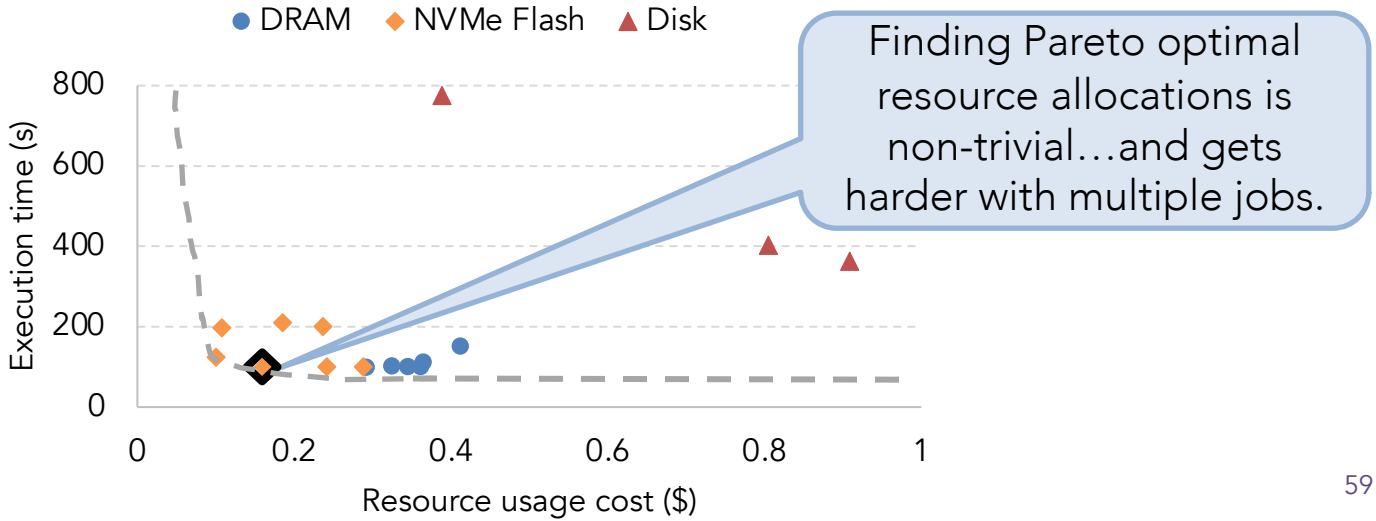


Requirements for ephemeral storage

1. High performance for a wide range of object sizes
2. Cost efficiency

Requirements for ephemeral storage

1. High performance for a wide range of object sizes
2. Cost efficiency
   - Example of performance-cost tradeoff for a serverless video analytics job with different ephemeral data store configurations



Requirements for ephemeral storage

1. High performance for a wide range of object sizes
2. Cost efficiency
3. ~~Fault-tolerance~~

Today's cloud storage systems are not optimized for the requirements of serverless analytics jobs.

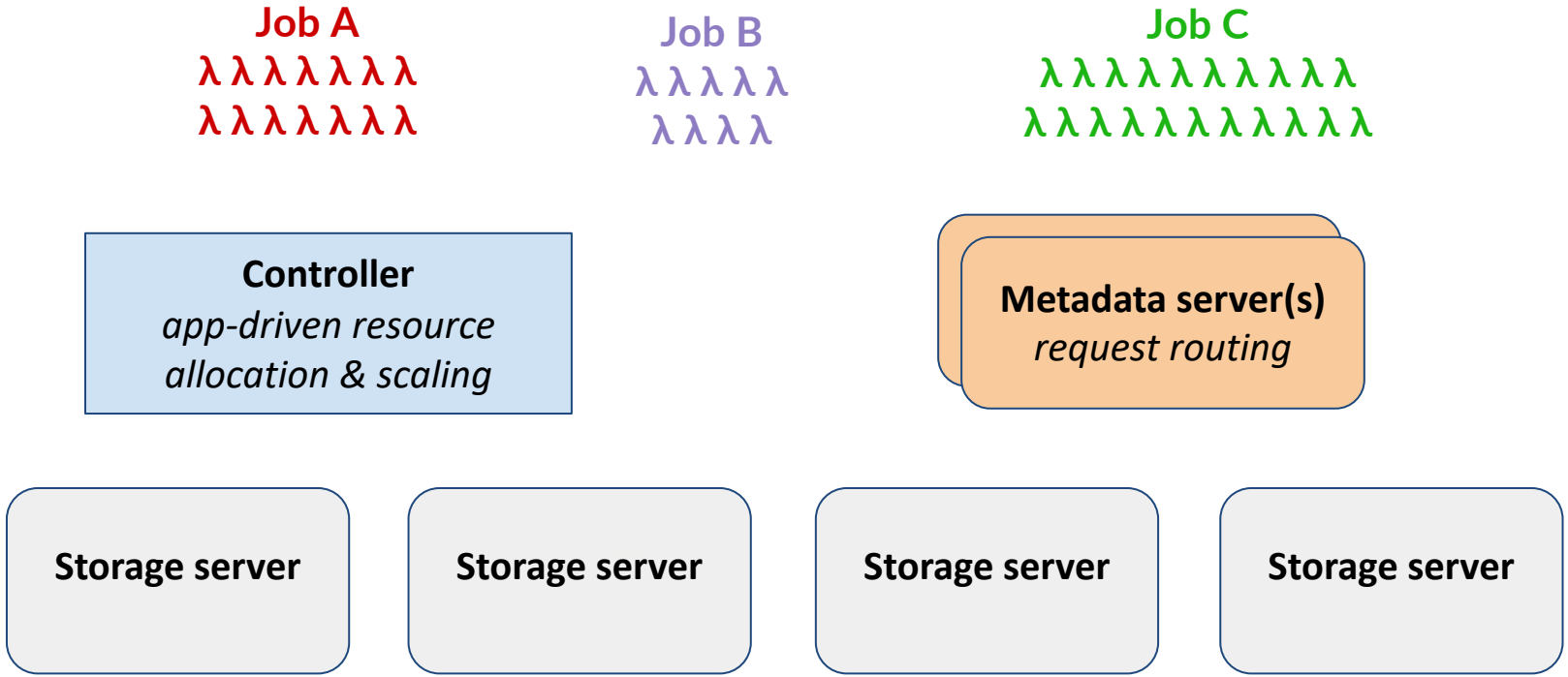
e.g., - S3 has low cost, but is optimized for large objects

- Redis offers high performance, but uses DRAM → expensive

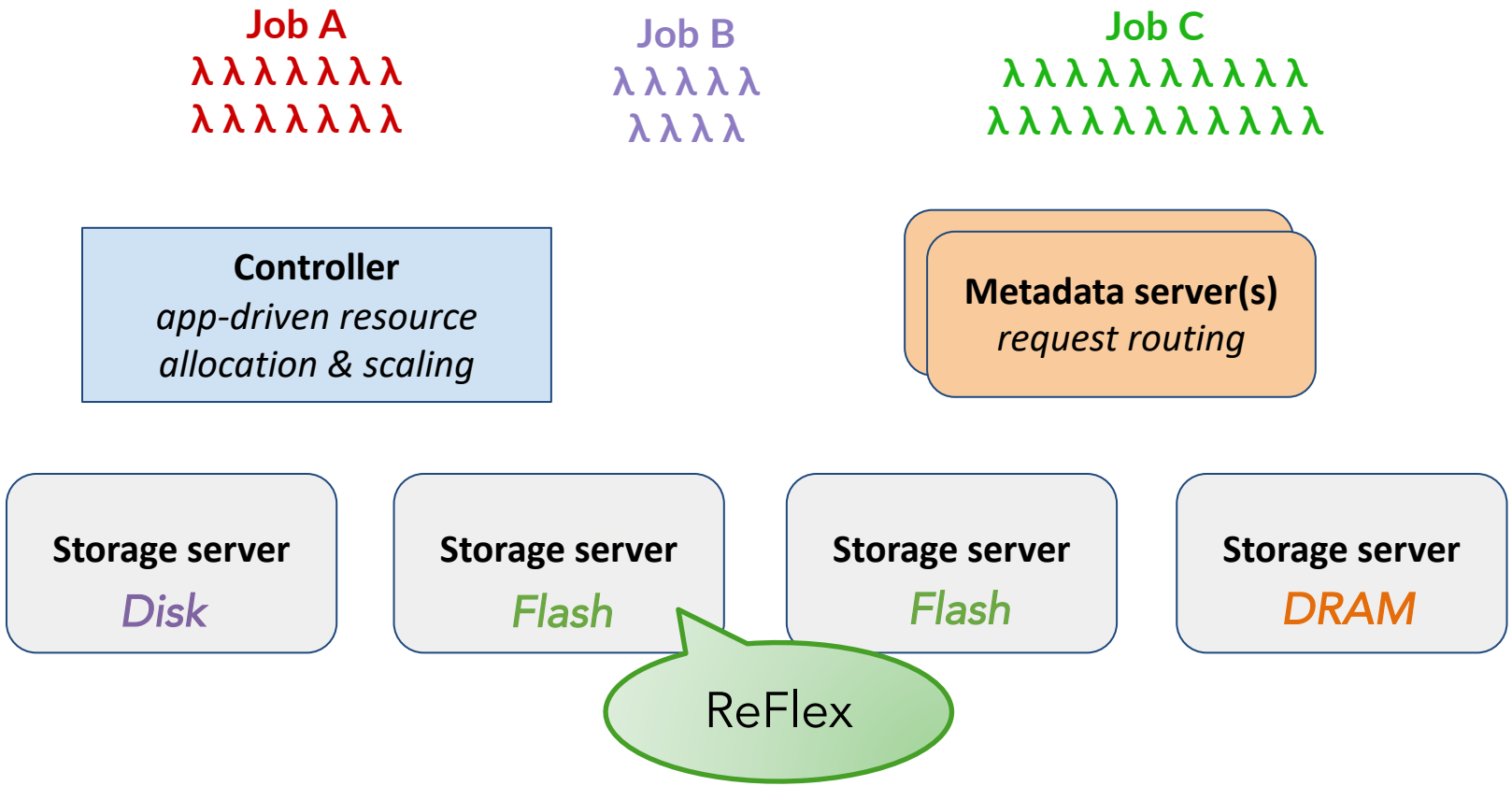
Pocket

- A fast and elastic storage service for ephemeral data sharing in serverless analytics
- Pocket achieves high performance and cost efficiency by:
  1. Leveraging multiple storage technologies
  2. Rightsizing resource allocations for applications
  3. Autoscaling storage resources in the cluster based on usage
- Similar performance to Redis, an in-memory data store, while saving ~60% in cost for various serverless analytics jobs

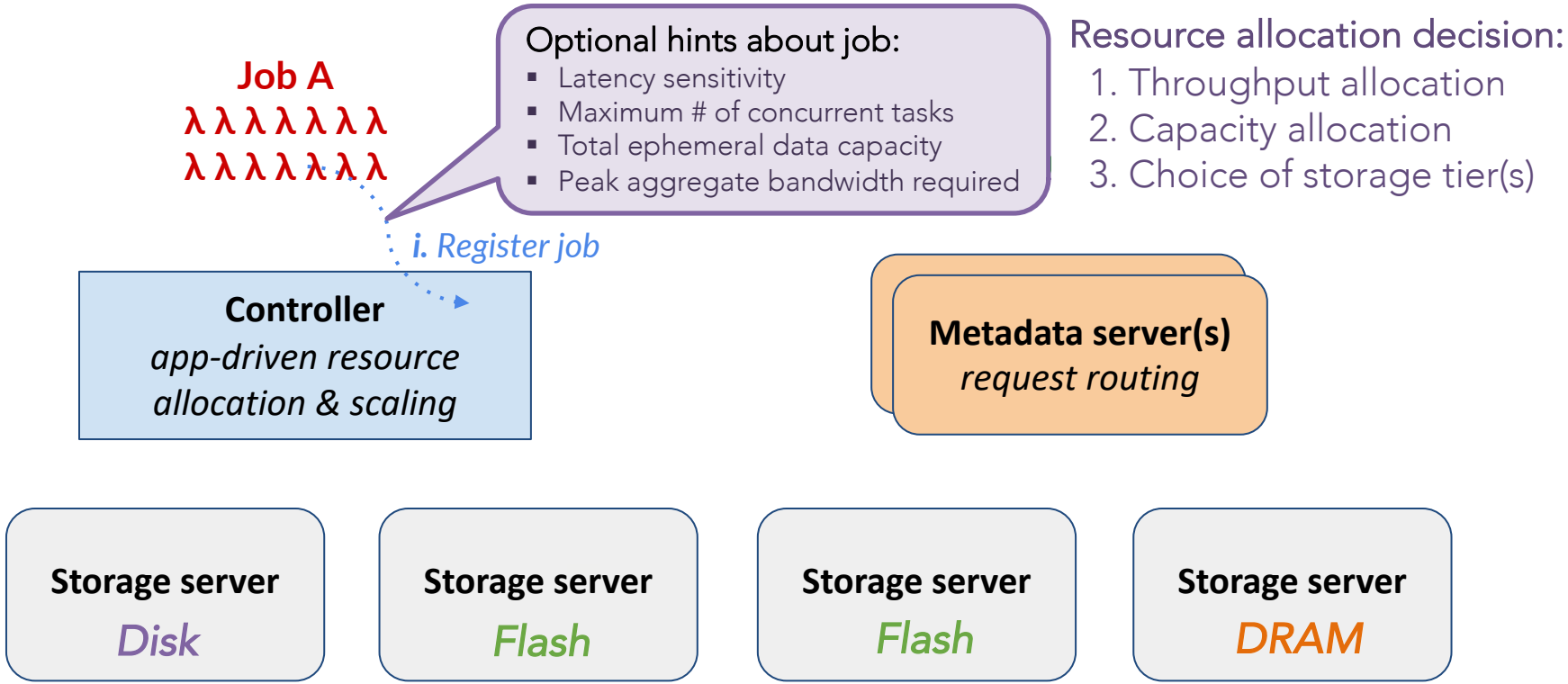
Pocket design



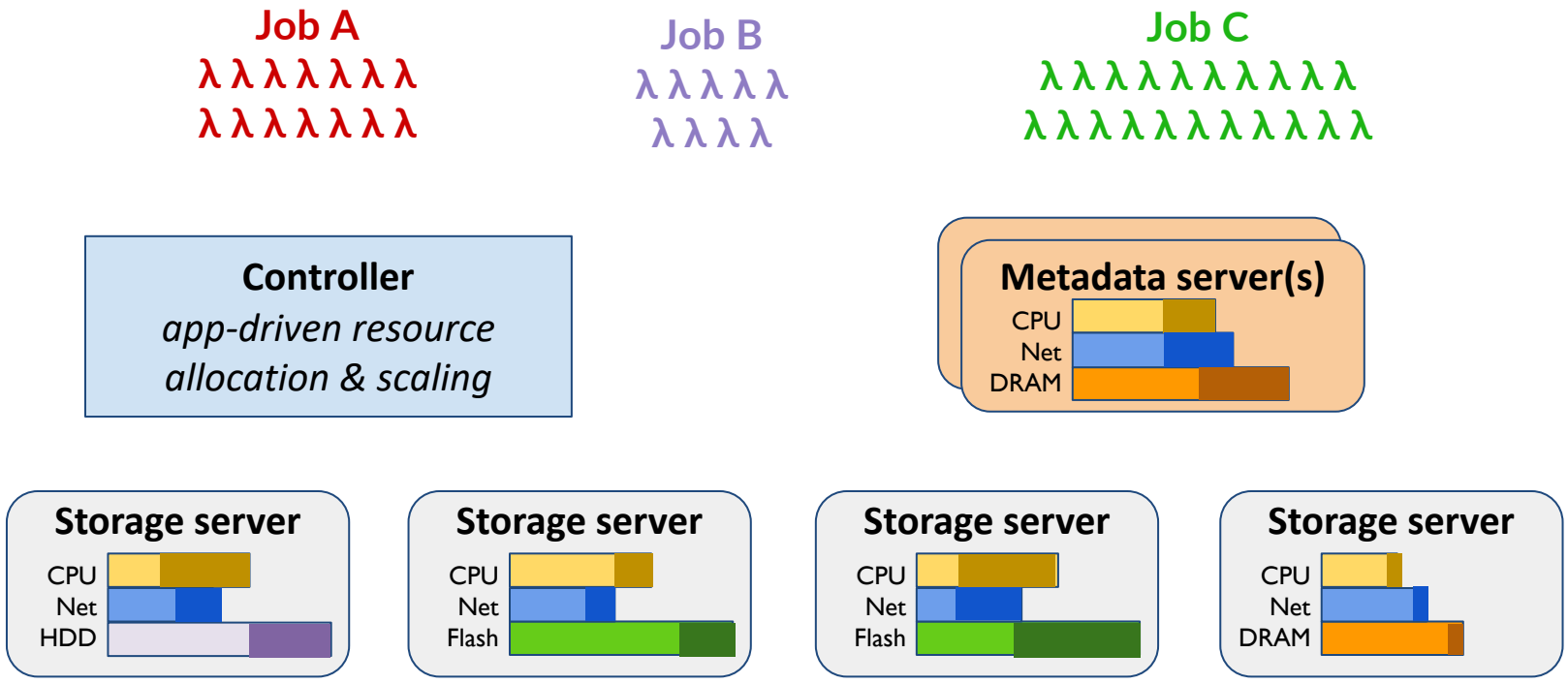
1. Leverage multiple storage technologies



2. Allocate resources based on job requirements



3. Autoscale resources based on utilization



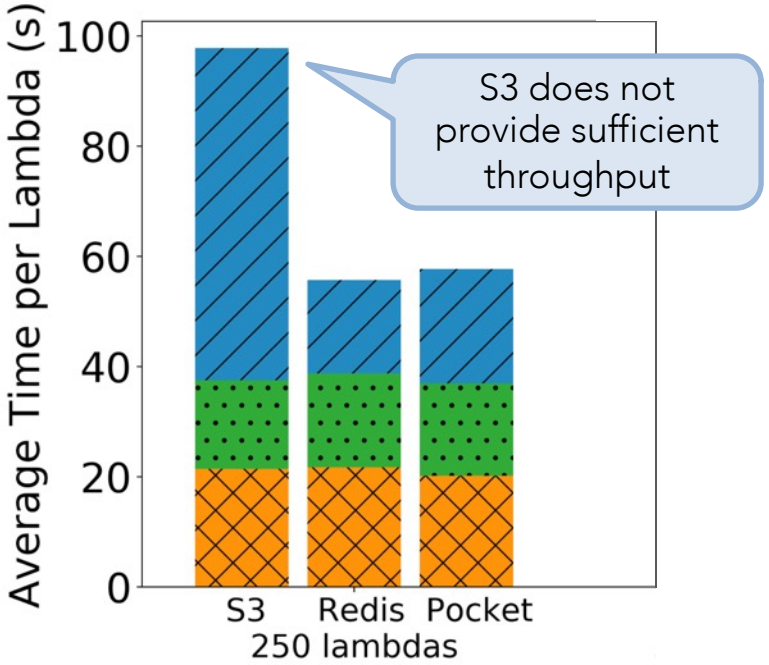
Evaluating Pocket

- Pocket is intended to be managed by cloud providers
- Evaluate Pocket running on Amazon EC2 virtual machines
- Run serverless applications on AWS Lambda:
  - Video analytics (object recognition)
  - MapReduce sort
  - Distributed software compilation

https://github.com/stanford-mast/pocket

Application performance with Pocket

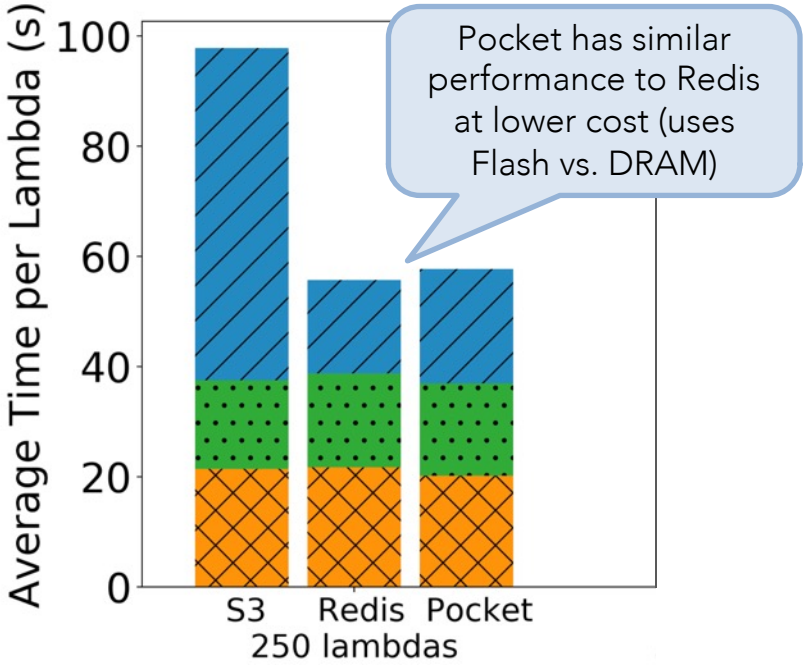
- Compare Pocket to S3 and Redis, which are commonly used today



MapReduce sort job hints	
Ephemeral capacity	100 GB
Latency sensitive	False
Aggregate peak throughput	100 Gb/s

Application performance with Pocket

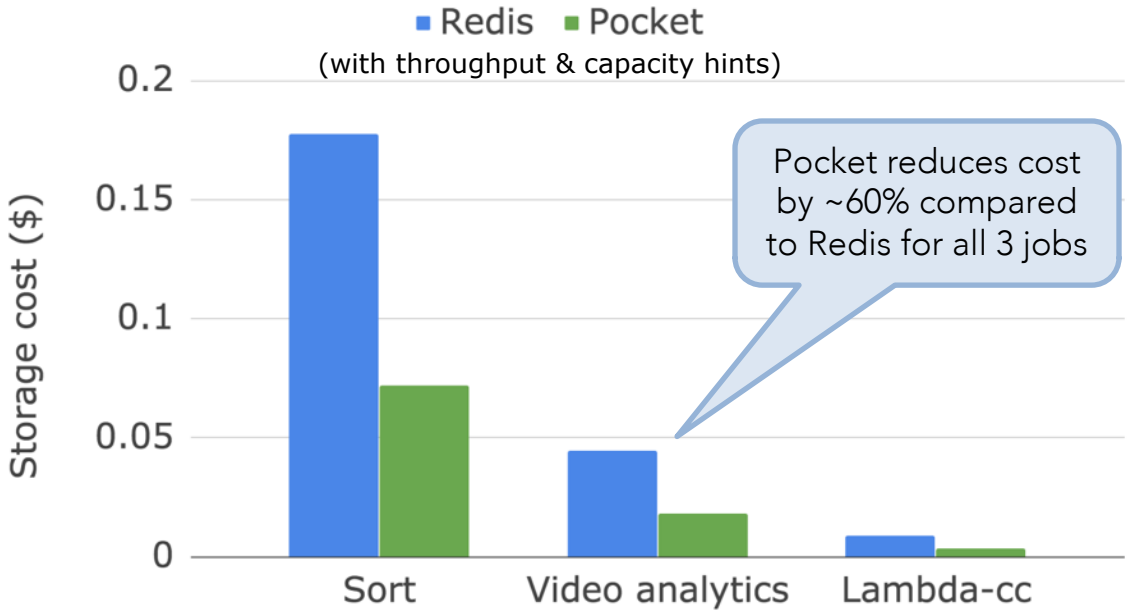
- Compare Pocket to S3 and Redis, which are commonly used today



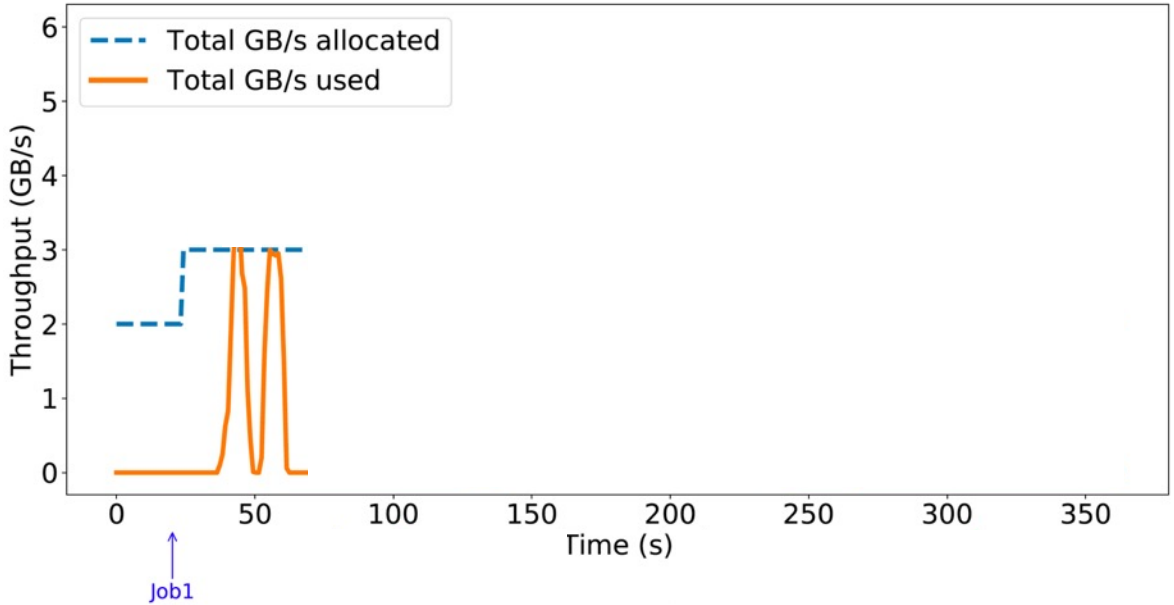
MapReduce sort job hints	
Ephemeral capacity	100 GB
Latency sensitive	False
Aggregate peak throughput	100 Gb/s

Application storage cost with Pocket

- Pocket leverages job attribute hints for cost-effective resource allocation and amortizes VM costs across multiple jobs, offering a pay-what-you-use model

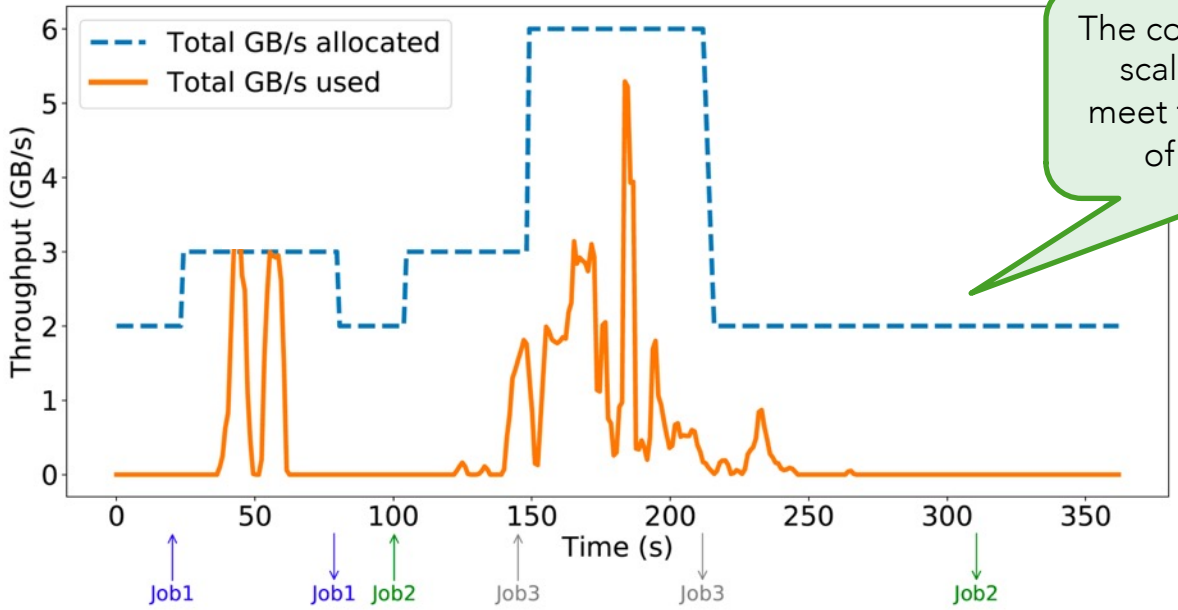


Autoscaling the Pocket cluster



Job hints	Job1: Sort
Latency sensitive	False
Ephemeral data capacity	10 GB
Aggregate throughput	3 GB/s

Autoscaling the Pocket cluster



The controller elastically scales resources to meet the requirements of multiple jobs

Job hints	Job1: Sort	Job2: Video analytics	Job3: Sort
Latency sensitive	False	False	False
Ephemeral data capacity	10 GB	6 GB	10 GB
Aggregate throughput	3 GB/s	2.5 GB/s	3 GB/s

Summary

To build resource-efficient, high performance storage systems, we need to:

Decouple storage from compute resources

- Improve resource allocation flexibility with fast, predictable access to remote data

[OSDI'14] **IX**: Dataplane OS for Fast Networking

[EuroSys'16] Flash Storage Disaggregation

[HotStorage'17] Rack-Scale Disaggregated Storage

[ASPLOS'17] **ReFlex**: Remote Flash ≈ Local Flash

Automate storage resource management

- Allocate resources dynamically based on app requirements, learned via hints or ML

[ATC'18a] Understanding Ephemeral Storage for Serverless Analytics

[OSDI'18] **Pocket**: Elastic Ephemeral Storage for Serverless Analytics

[ATC'18b] **Selecta**: ML-based Heterogeneous Cloud Storage Configuration

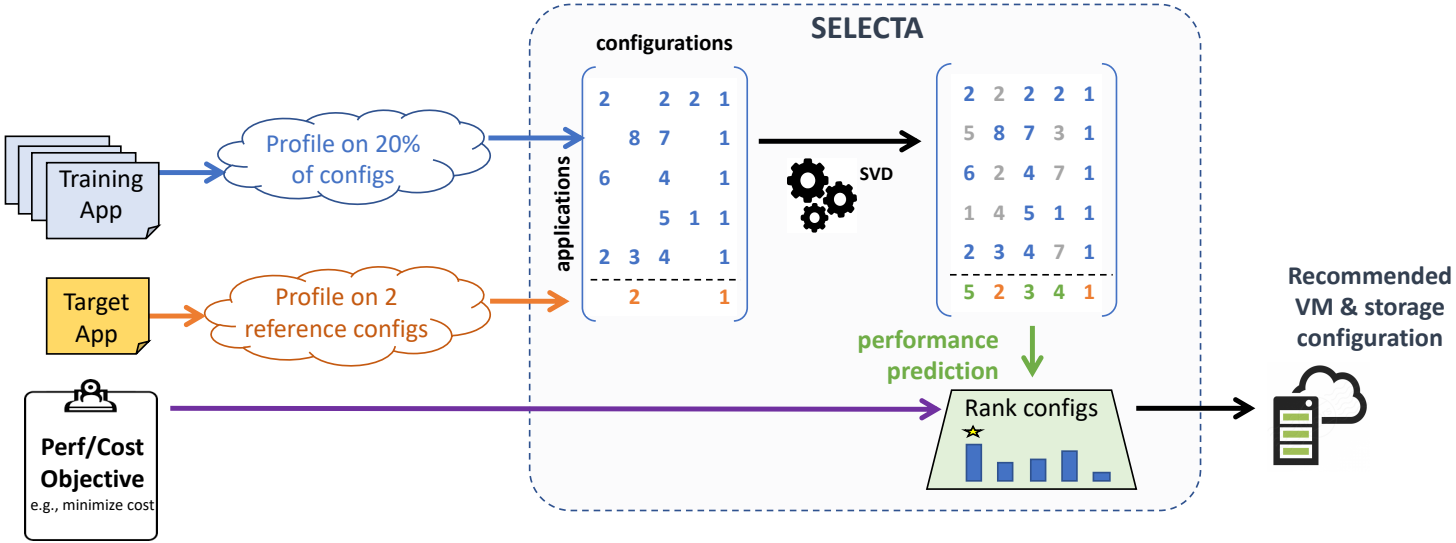
What’s next?

Machine learning for cloud systems

- Use ML to enhance or replace **heuristics** and user **hints**
  - E.g., Automate cluster resource allocation

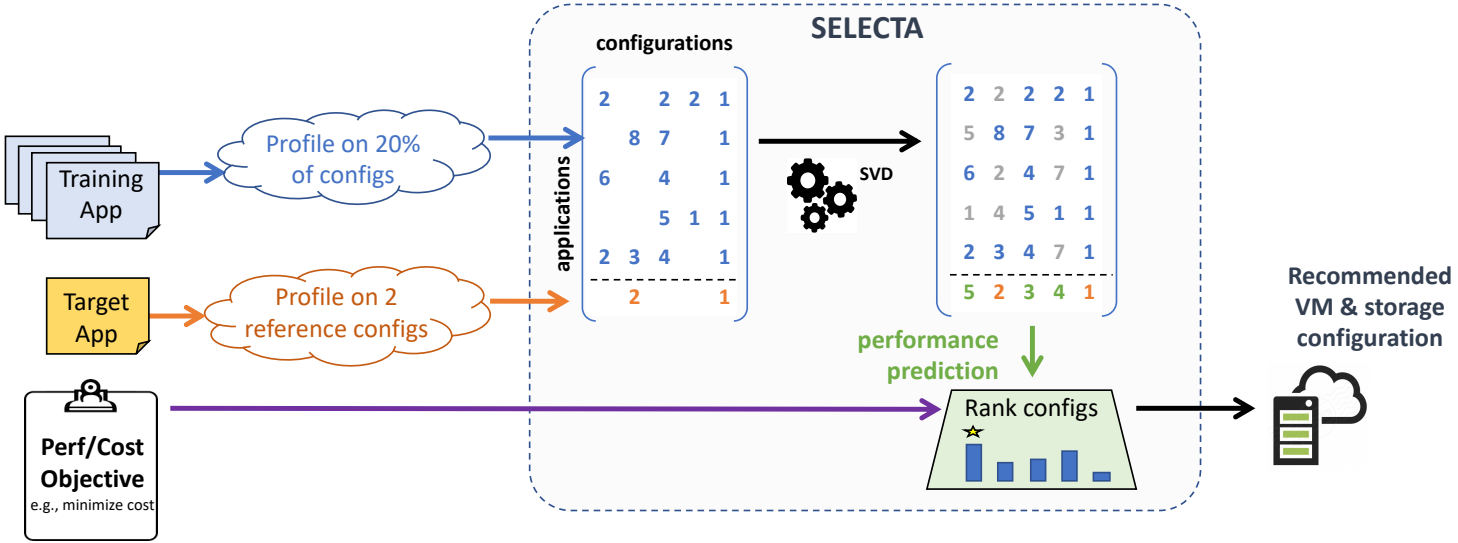
ML-based resource allocation

- *Selecta* [ATC'18] predicts near-optimal configuration for a job using sparse training data across jobs



ML-based resource allocation

- *Selecta* [ATC'18] predicts near-optimal configuration for a job using sparse training data across jobs
  - 94% probability of recommending near-optimal performing configuration
  - 80% probability of recommending near-optimal cost configuration



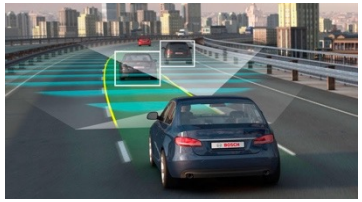
Machine learning for cloud systems

- Use ML to enhance or replace heuristics and user hints
  - Automate cluster resource allocation
  - Predict resource utilization for opportunistic computing
  - Debug performance and security by detecting anomalies

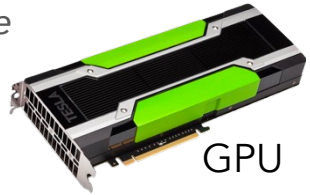
Systems for machine learning

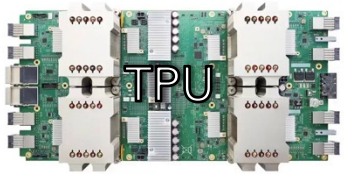
ML-based applications

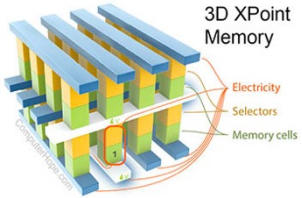





Hardware

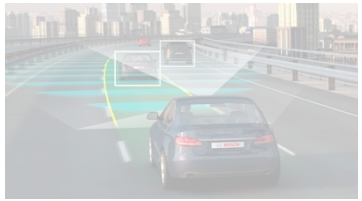




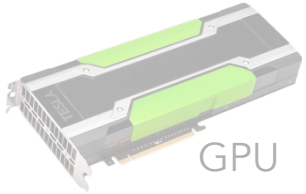


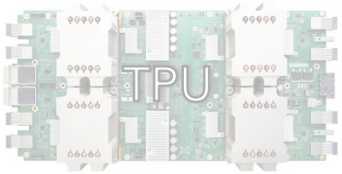
Systems for machine learning

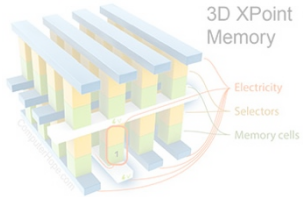





- Near-storage computing for massive datasets
- Distribute computing between the cloud and edge
- Privacy in the cloud with secure enclaves







Acknowledgements

Christos Kozyrakis
Patrick Stuedi
Heiner Litz
Yawen Wang
Adam Belay
Animesh Trivedi
Jonas Pfefferle
Binu John
Eno Thereska
George Prekas
Edouard Bugnion

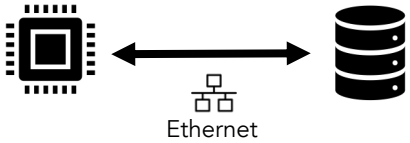
Stanford
University

Microsoft

Conclusion

As we continue exponentially generating and analyzing data in the cloud, we need:

1. Fast access to remote data



2. Automatic, elastic allocation of storage resources

[OSDI'14] **IX**: Dataplane OS for Fast Networking

[EuroSys'16] Flash Storage Disaggregation

[HotStorage'17] Rack-Scale Disaggregated Storage

[ASPLOS'17] **ReFlex**: Remote Flash ≈ Local Flash

[ATC'18a] Understanding Ephemeral Storage for Serverless Analytics

[OSDI'18] **Pocket**: Elastic Ephemeral Storage for Serverless Analytics

[ATC'18b] **Selecta**: ML-based Heterogeneous Cloud Storage Configuration