Flash Storage Disaggregation

Ana Klimovic[1], Christos Kozyrakis[1,4],
Eno Thereska[3,5], Binu John[2] and Sanjeev Kumar[2]

1 Stanford

2 facebook

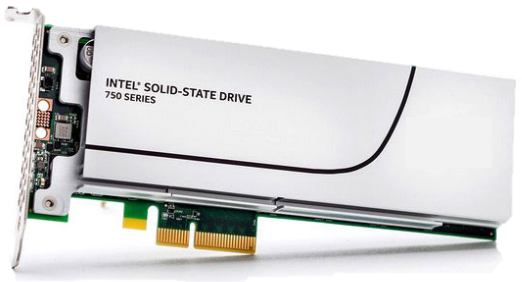
3 confluent
Real-time streams powered by Apache Kafka.

4 EPFL

5 Imperial College London

Flash is underutilized

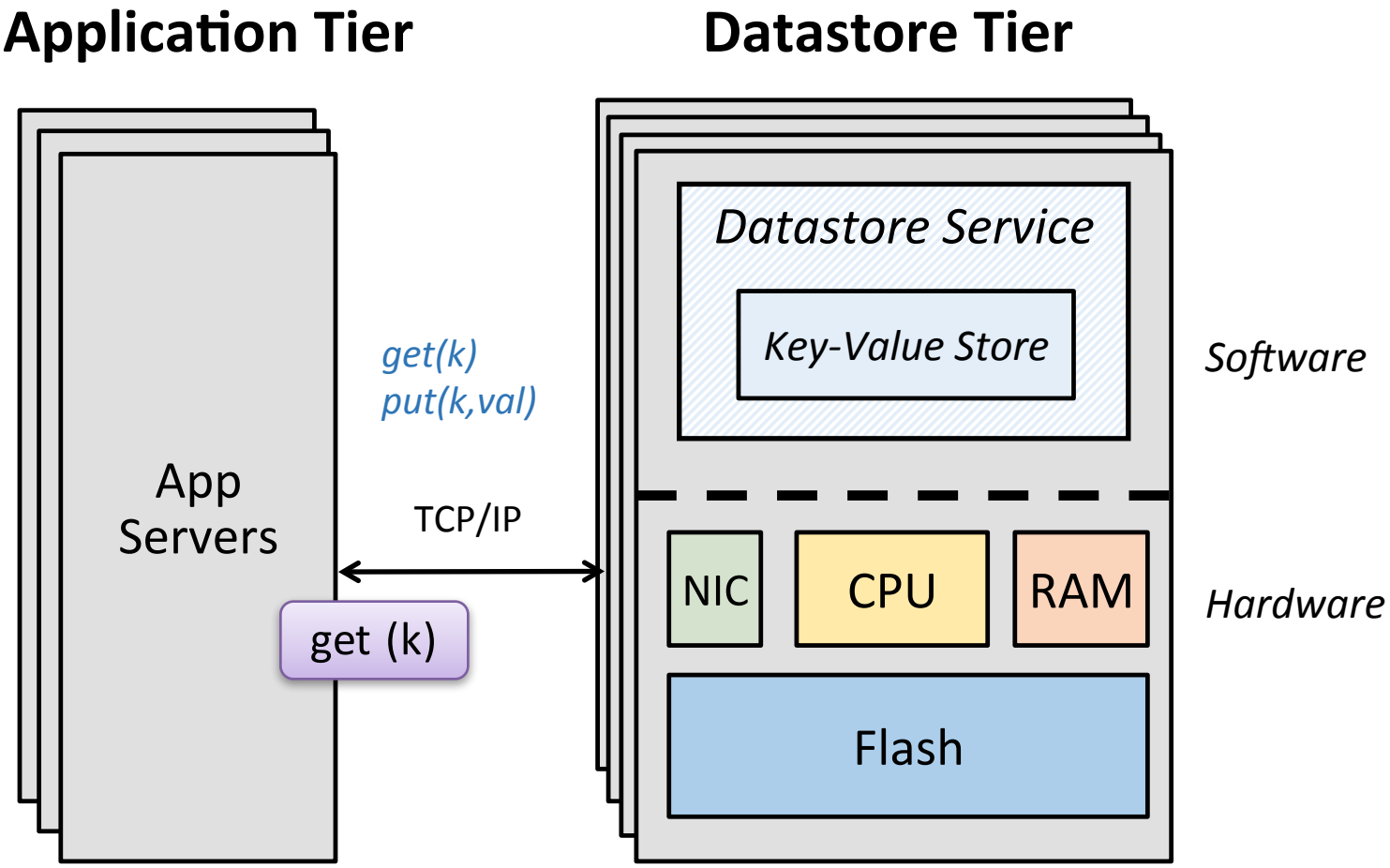
- Flash provides higher throughput and lower latency than disk



PCIe Flash:
- 100,000s of IOPS
- 10s of µs latency

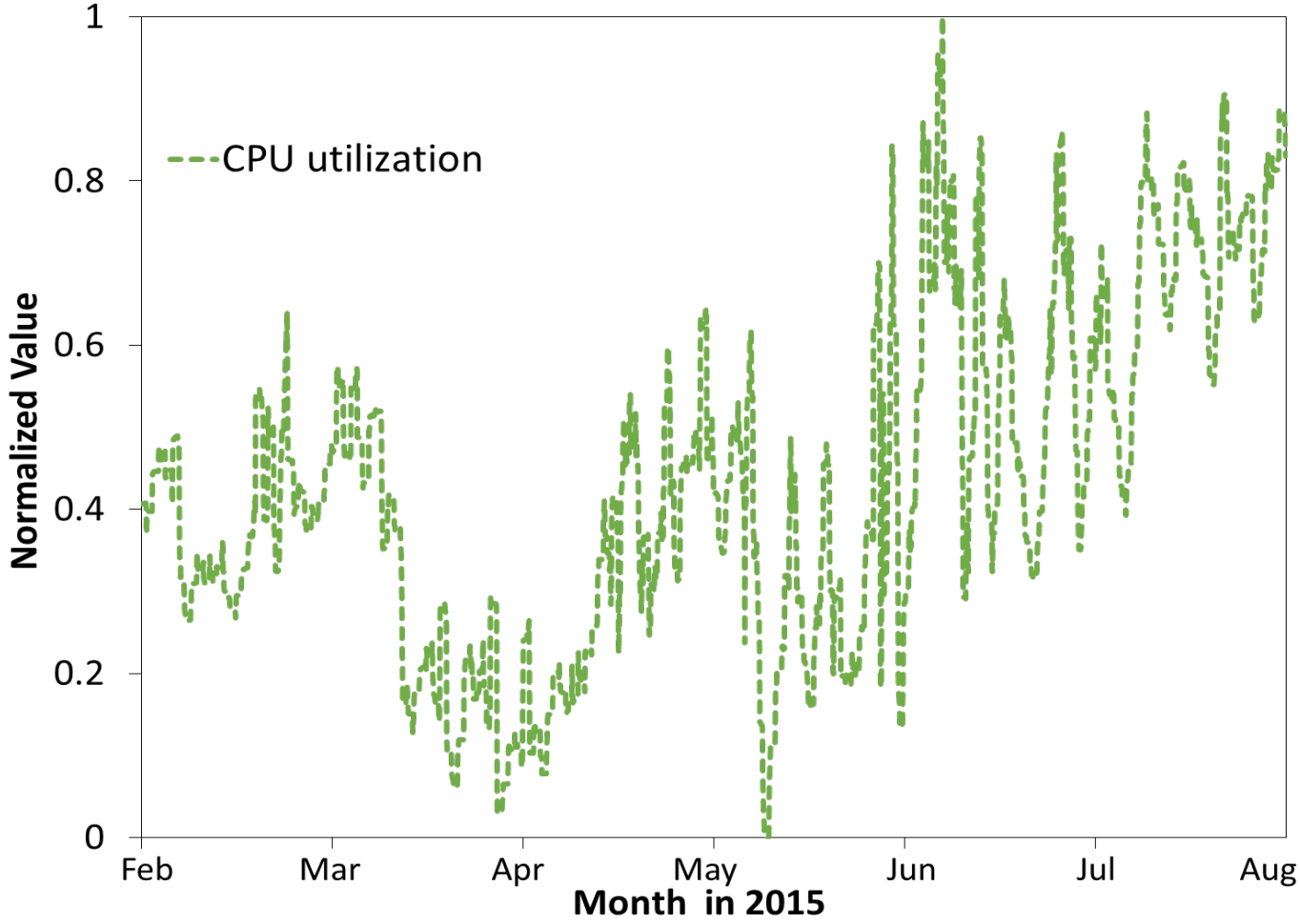
- Flash is underutilized in datacenters due to imbalanced resource requirements

Datacenter Flash Use-Case



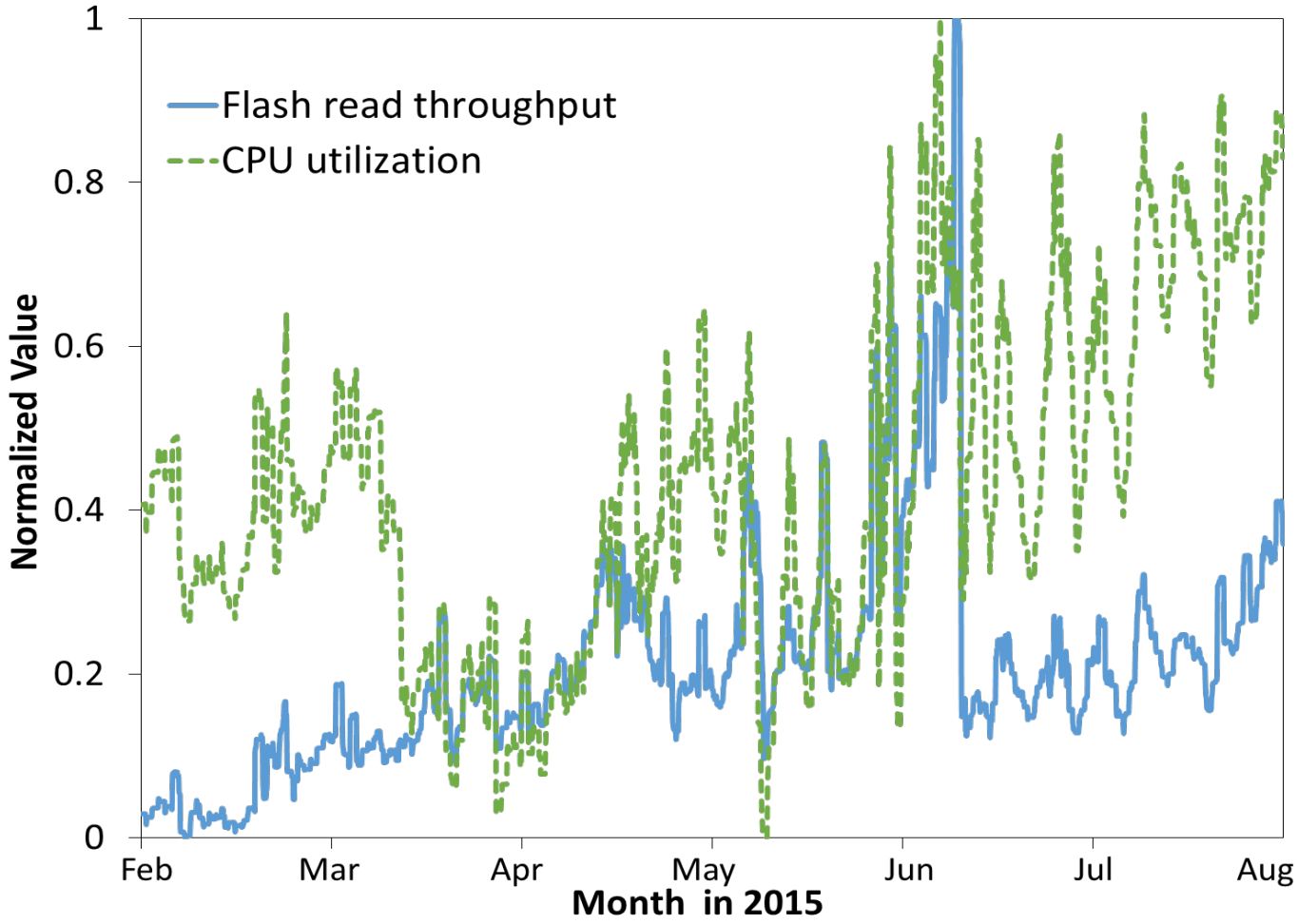
Imbalanced Resource Utilization

- Sample utilization of Facebook servers hosting a Flash-based key-value store over 6 months



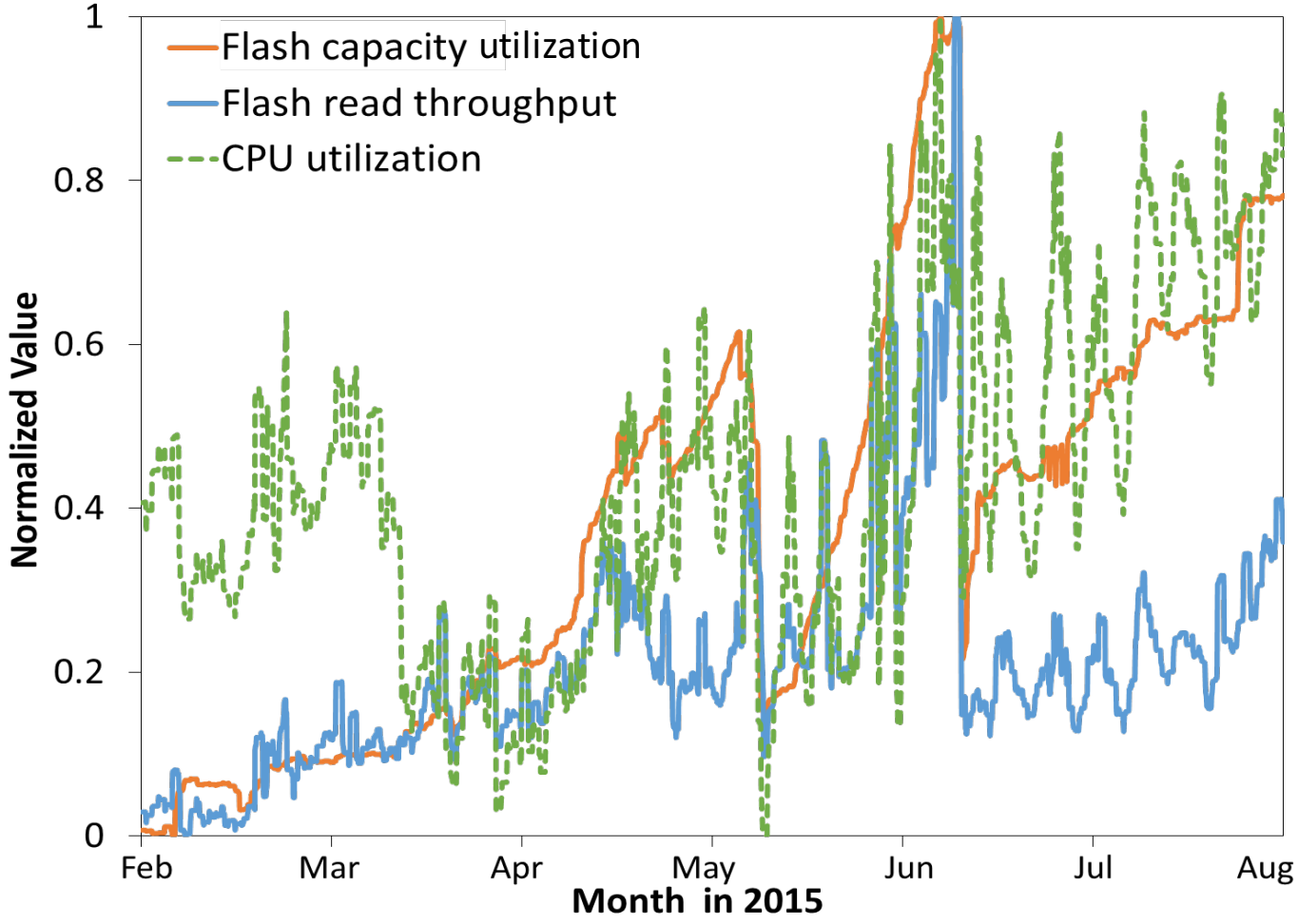
Imbalanced Resource Utilization

- Sample utilization of Facebook servers hosting a Flash-based key-value store over 6 months



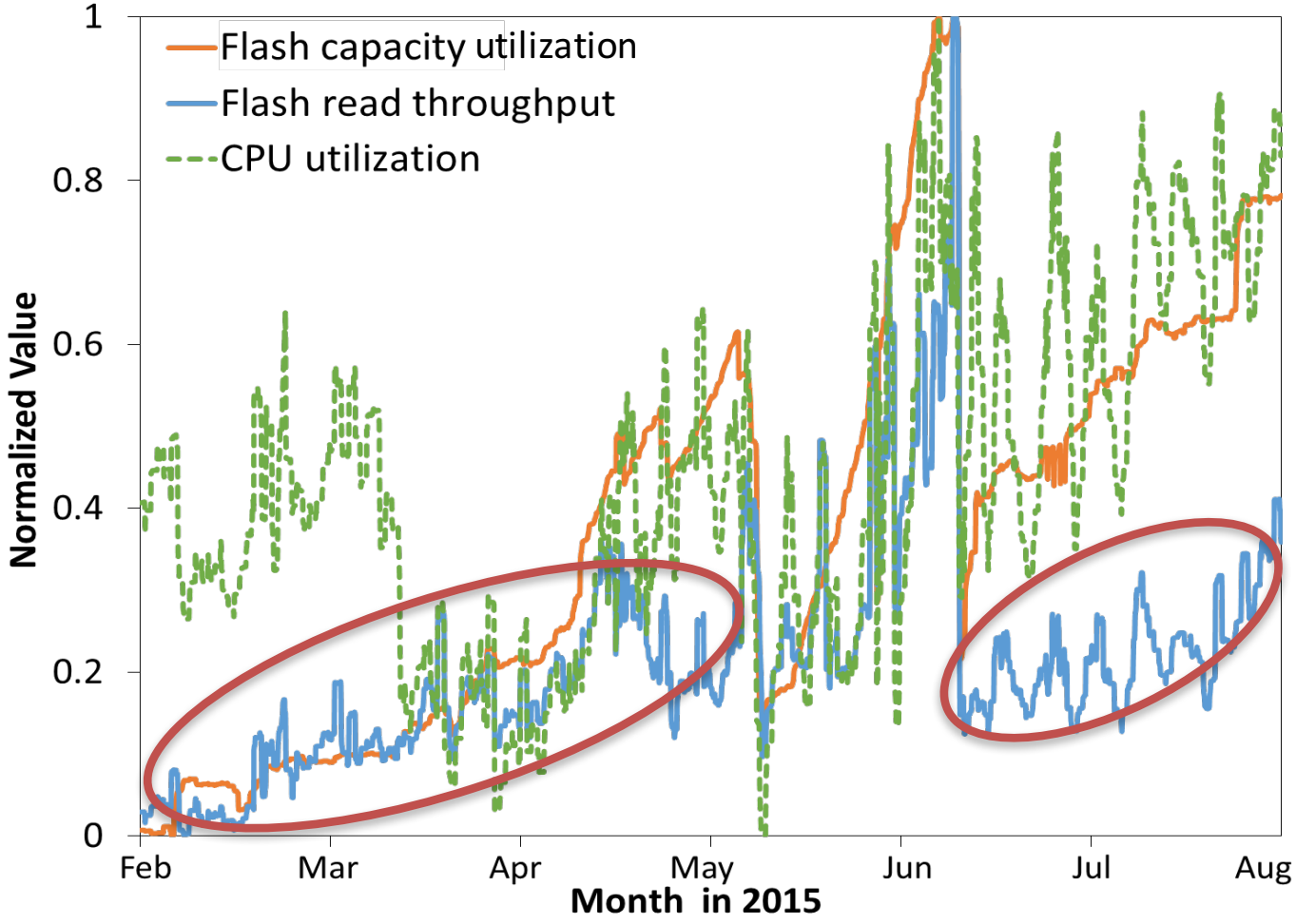
Imbalanced Resource Utilization

- Sample utilization of Facebook servers hosting a Flash-based key-value store over 6 months



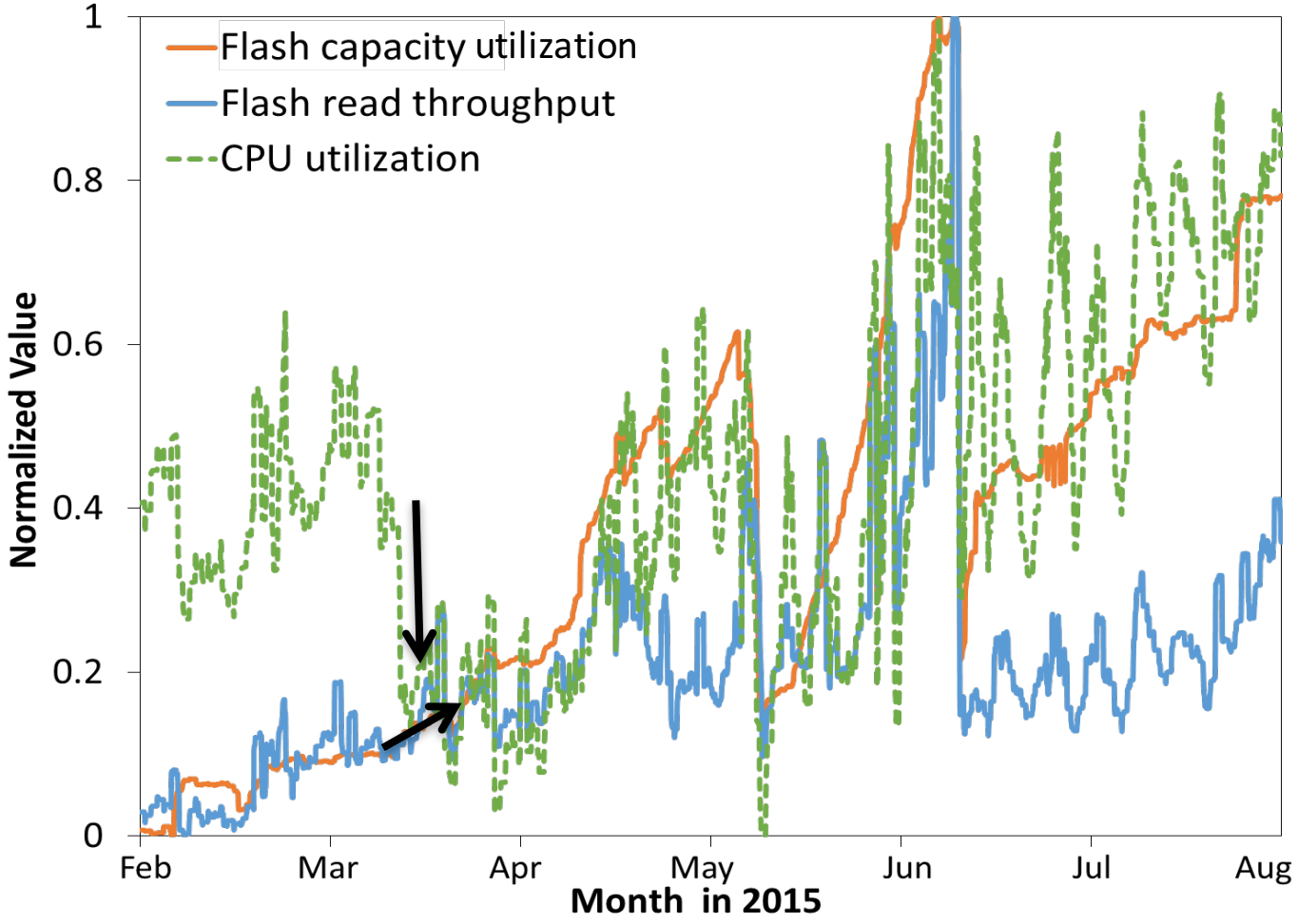
Imbalanced Resource Utilization

- Flash capacity and IOPS are underutilized for long periods of time

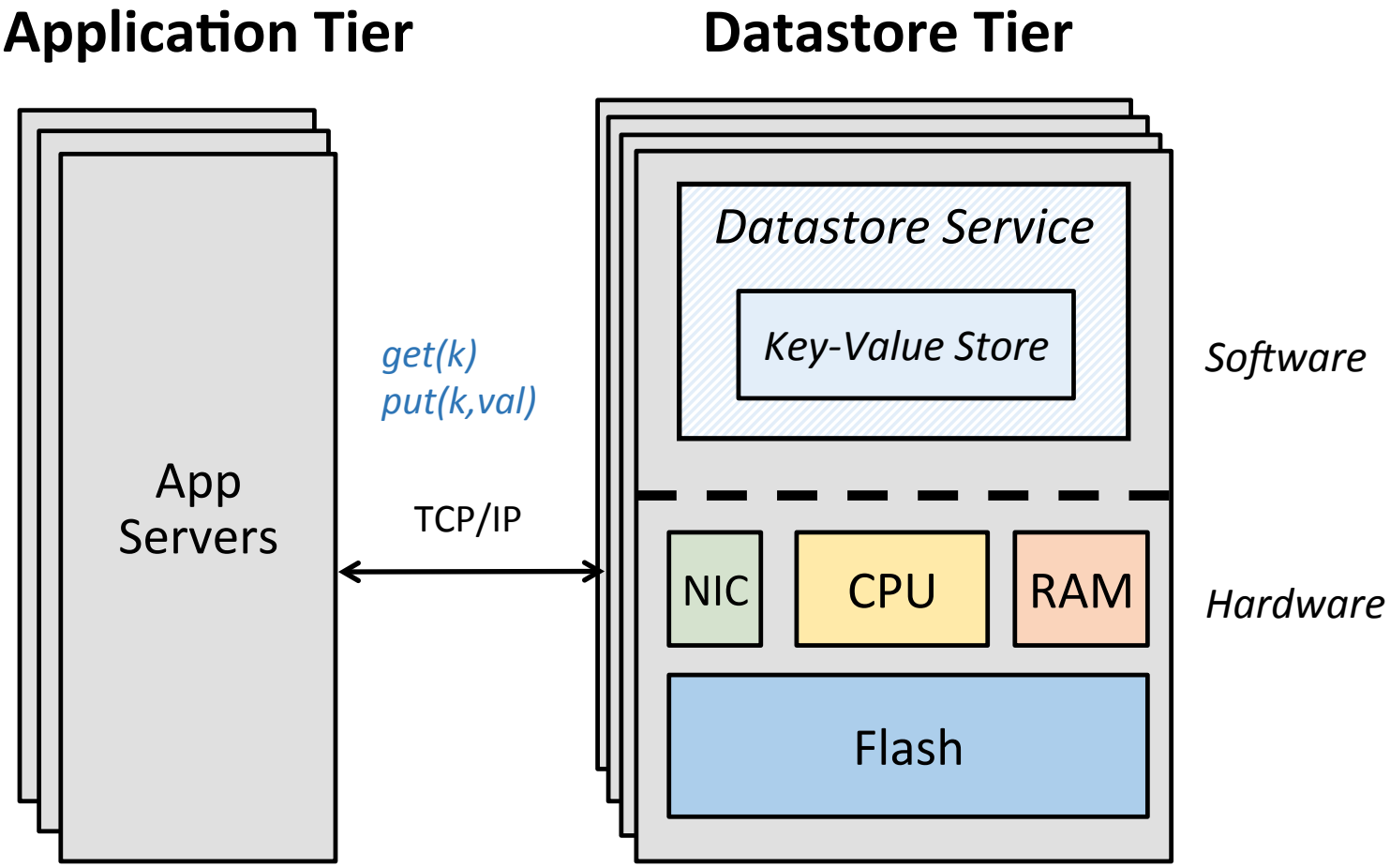


Imbalanced Resource Utilization

- CPU and Flash utilization vary with separate trends

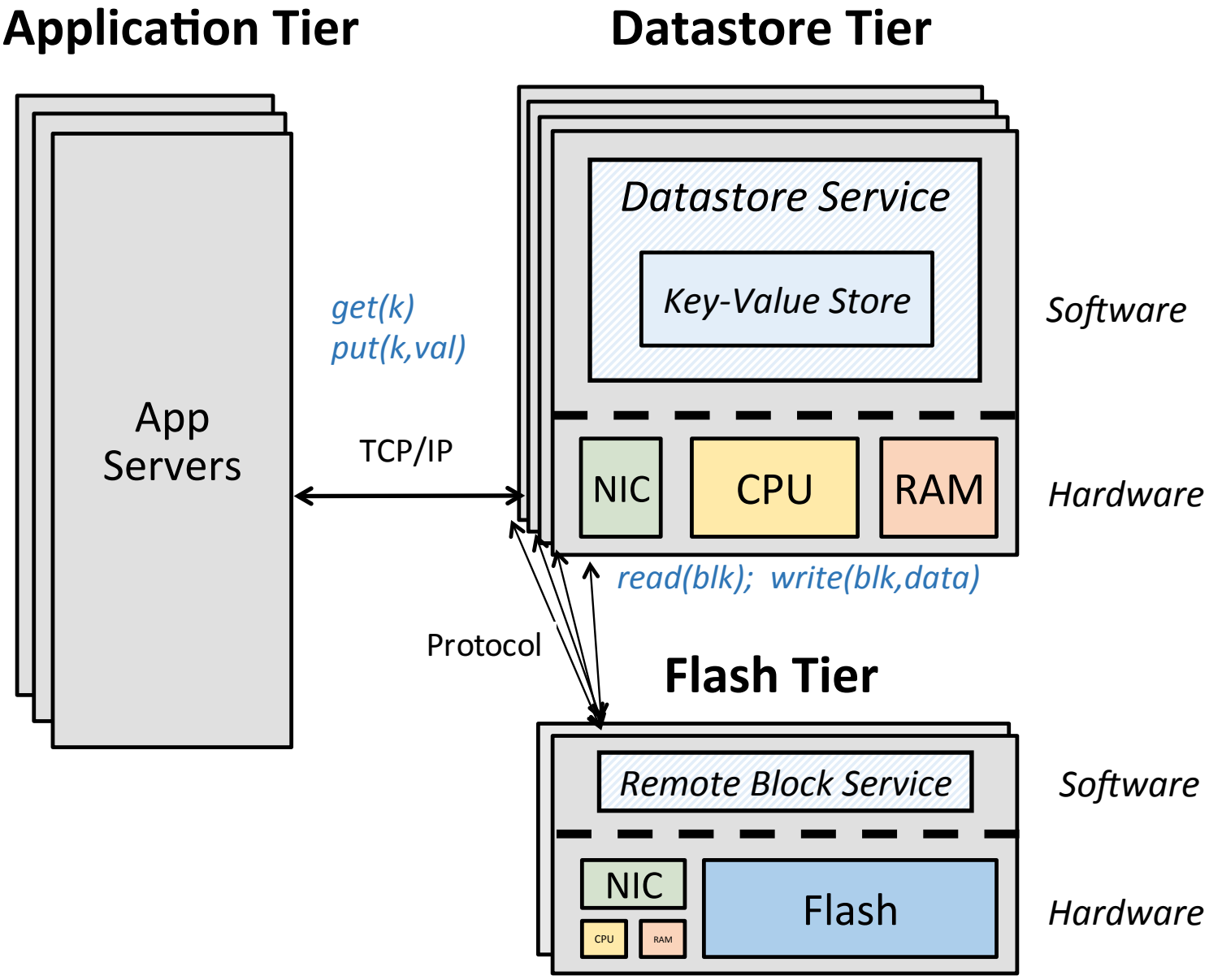


Local Flash Architecture



Provision Flash and CPU in a dependent manner.

Disaggregated Flash Architecture



Contributions

For real applications at Facebook, we analyze:

1. What is the performance overhead of remote Flash using existing protocols?
2. What optimizations improve performance?
3. When does disaggregating Flash lead to resource efficiency benefits?

Flash Workloads at Facebook

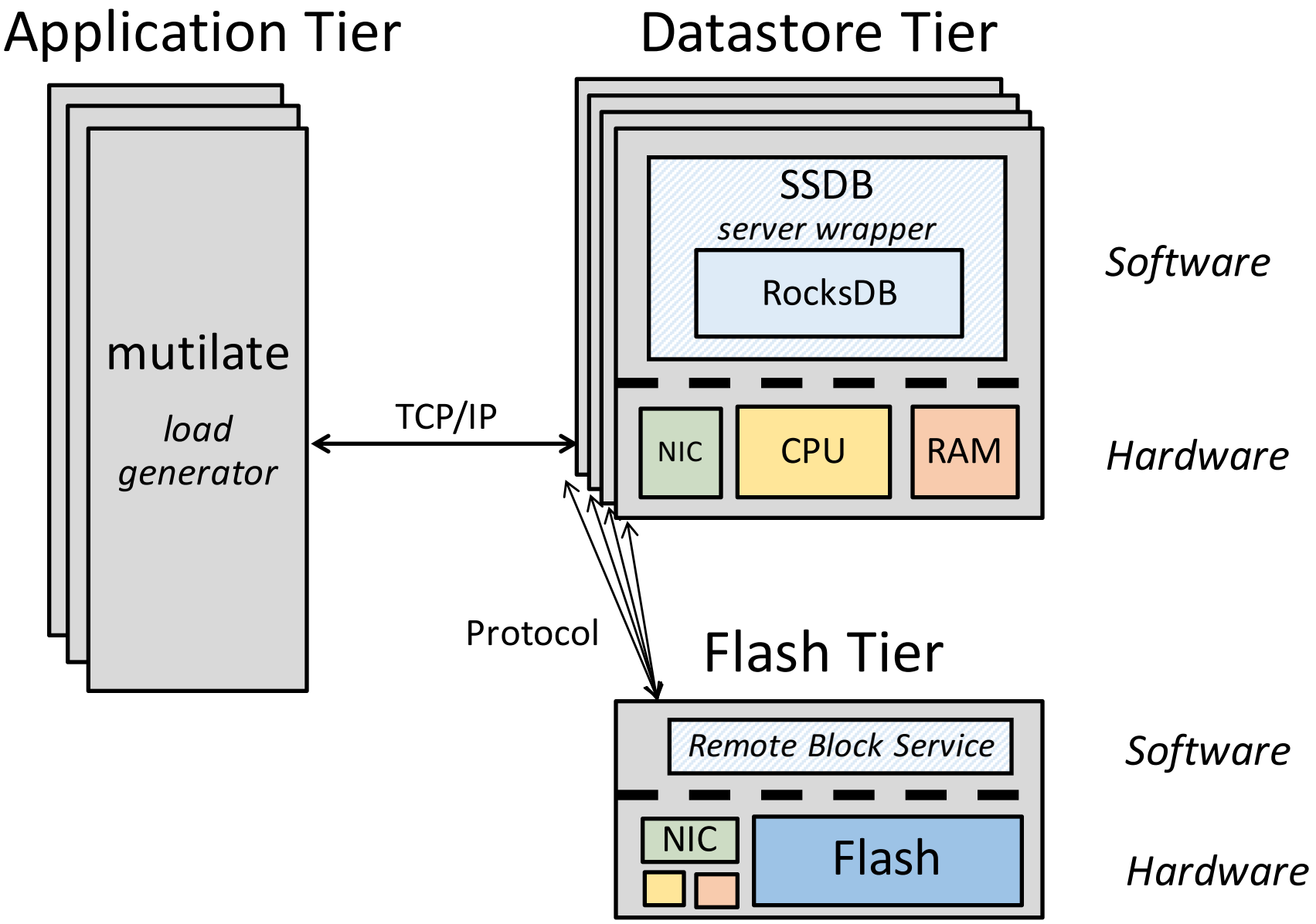
- Analyze IO patterns of real Flash-based Facebook applications
- Applications use RocksDB, a key-value store with a log structured merge tree architecture

	IOPS/TB	IO size
Read	2K – 10K	10KB – 50KB
Write	100 – 1K	500KB – 2MB

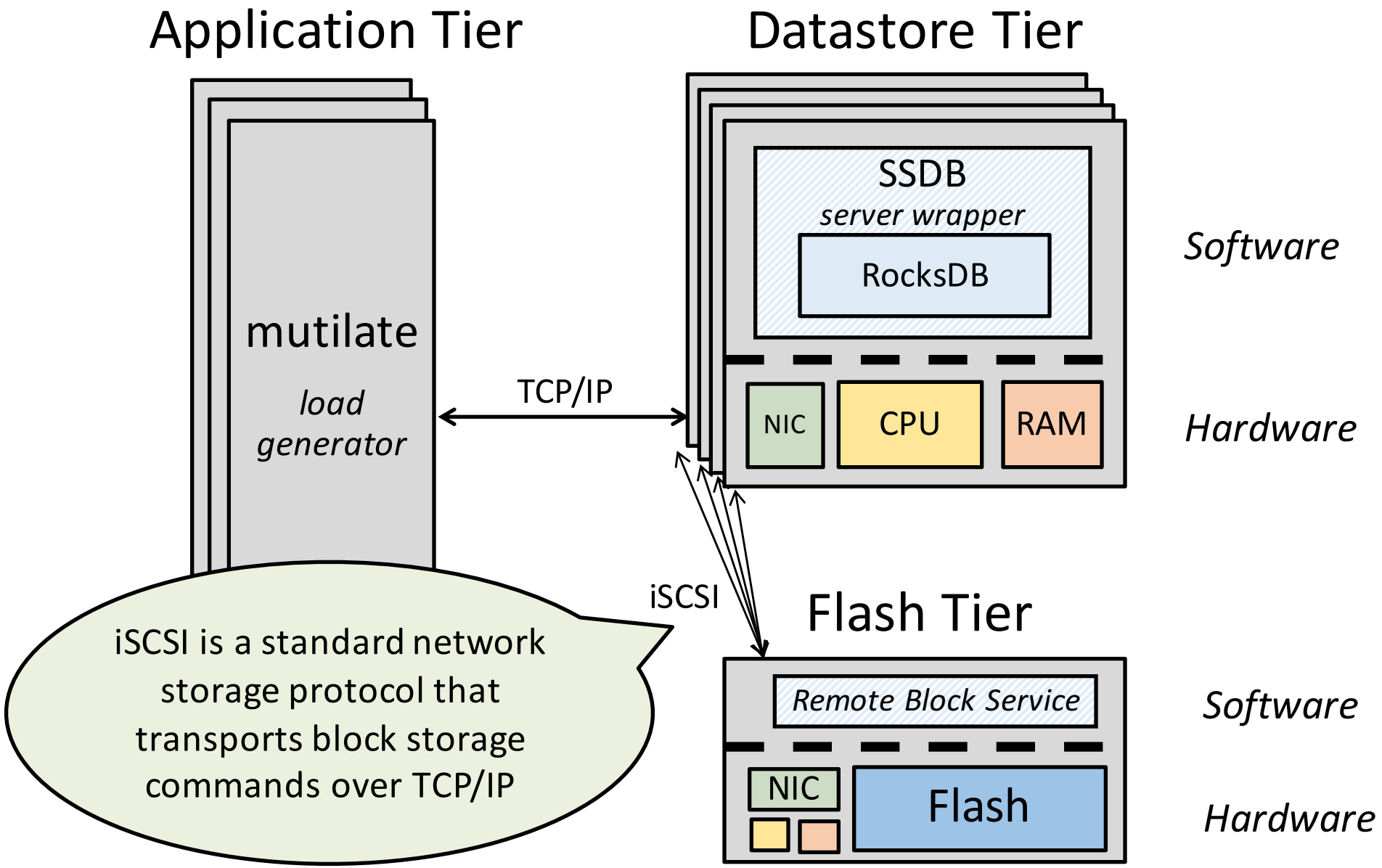
Lots of random reads

Large, bursty writes

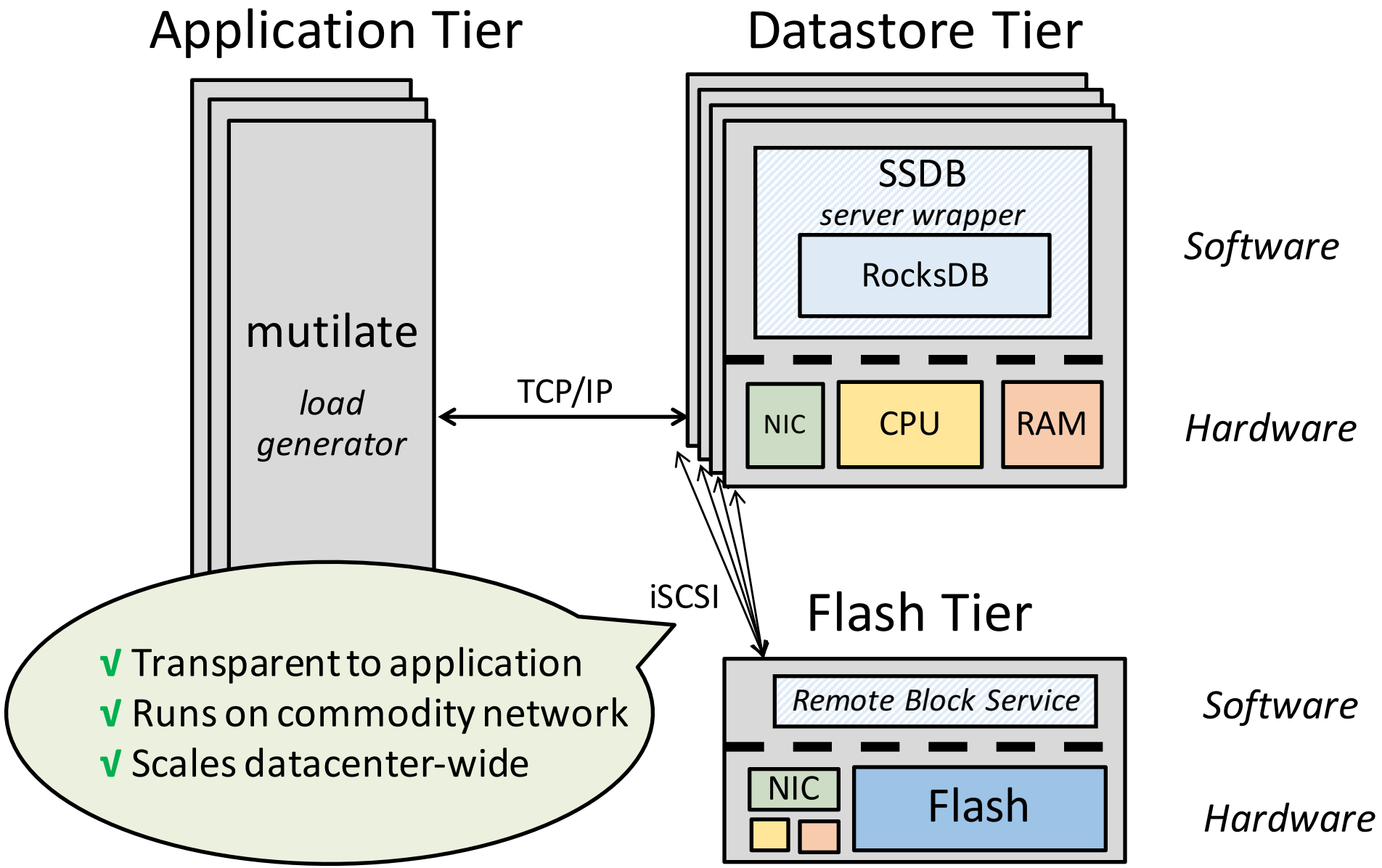
Workload Analysis



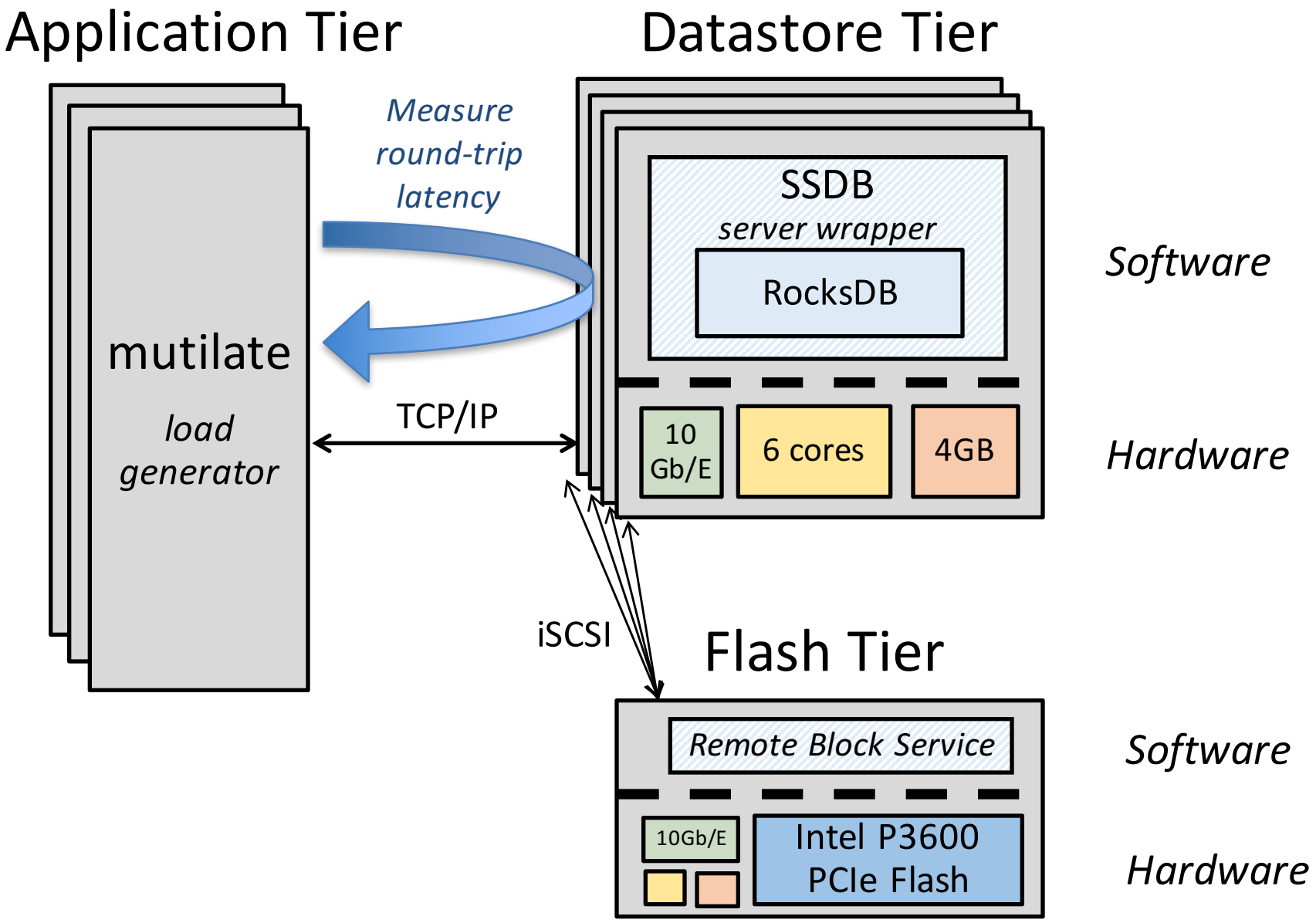
Workload Analysis



Workload Analysis

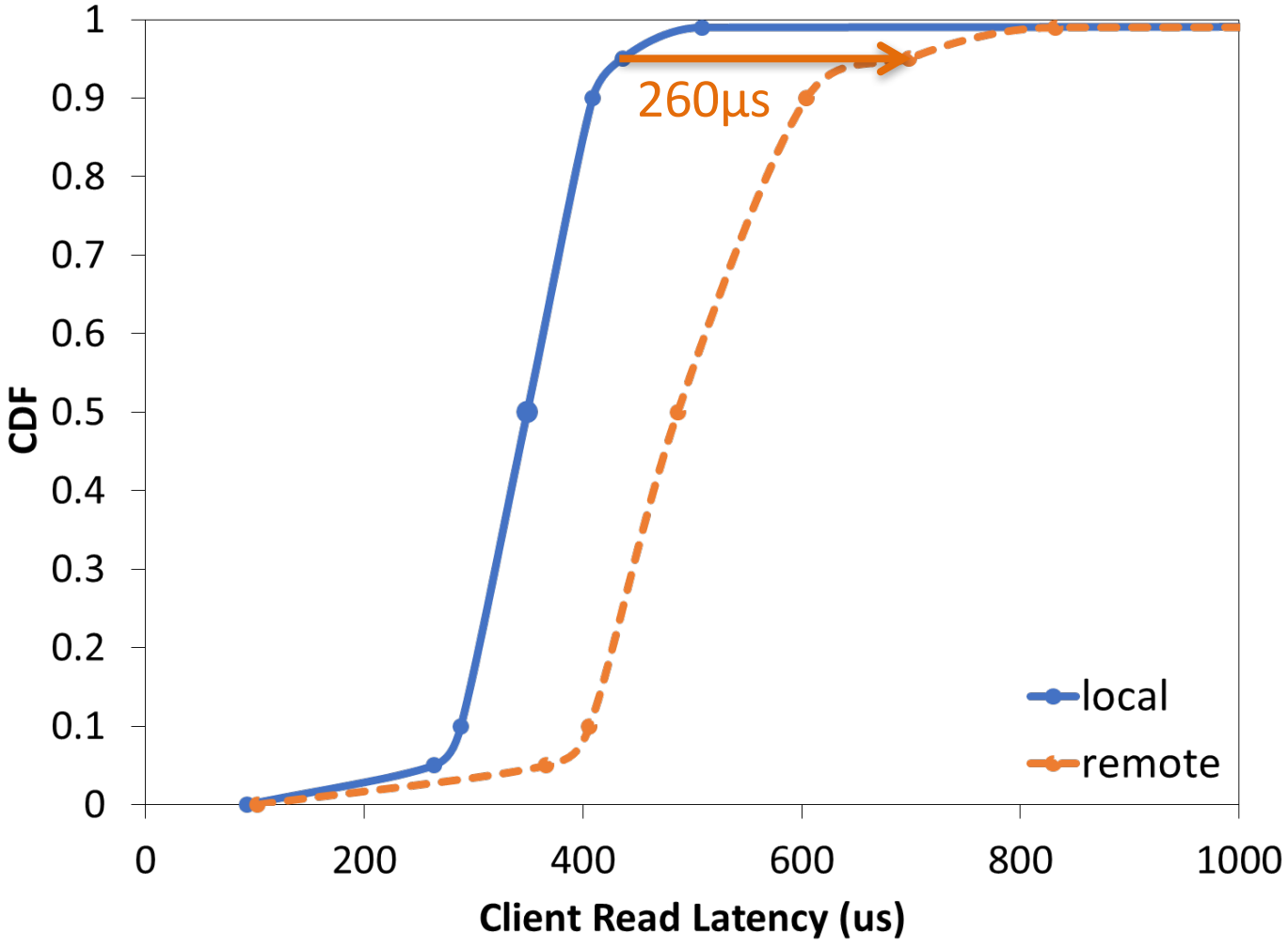


Workload Analysis



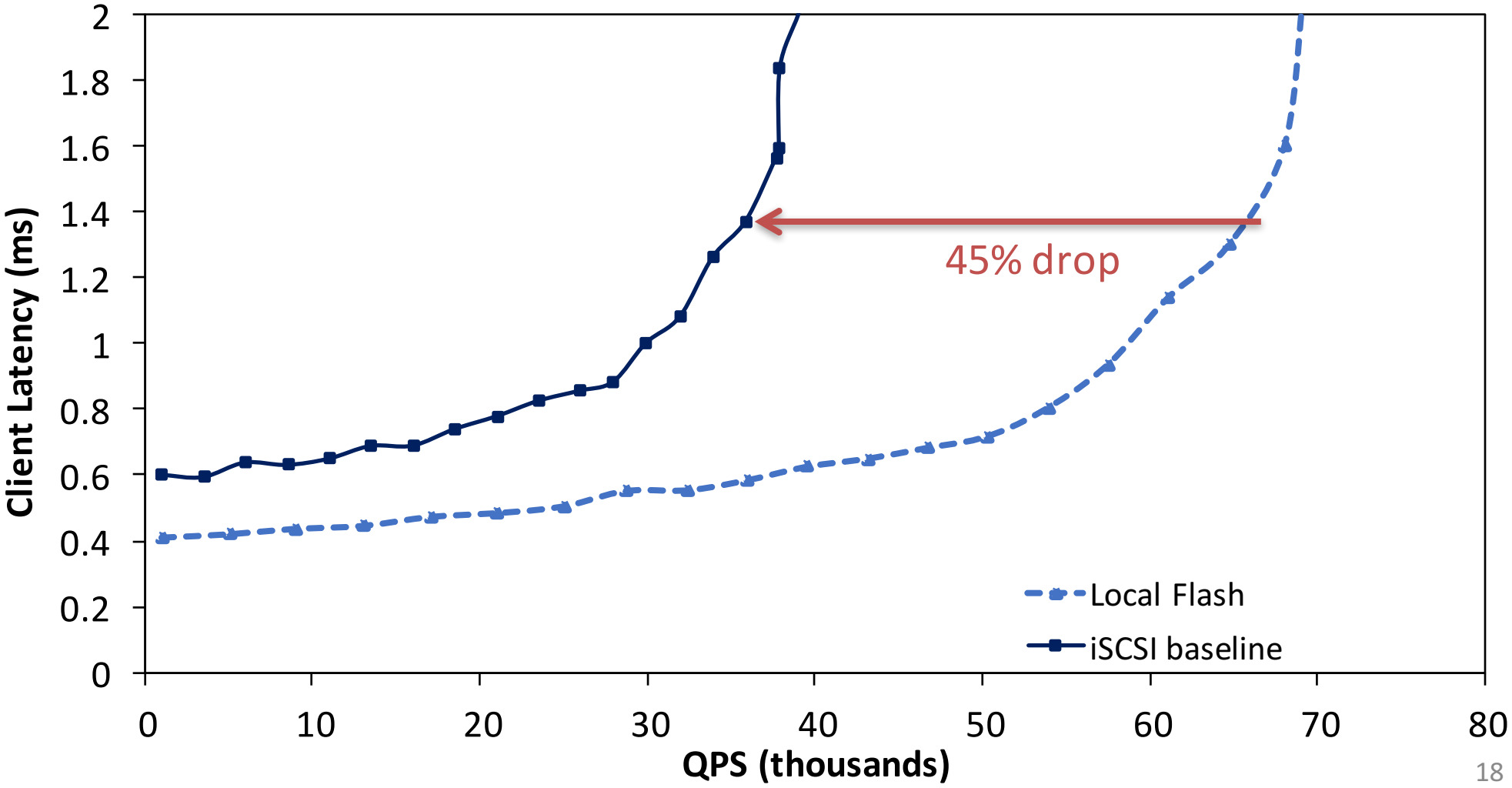
Unloaded Latency

- Remote access with iSCSI adds 260µs to p95 latency, tolerable for our target application (latency SLO ~5ms)



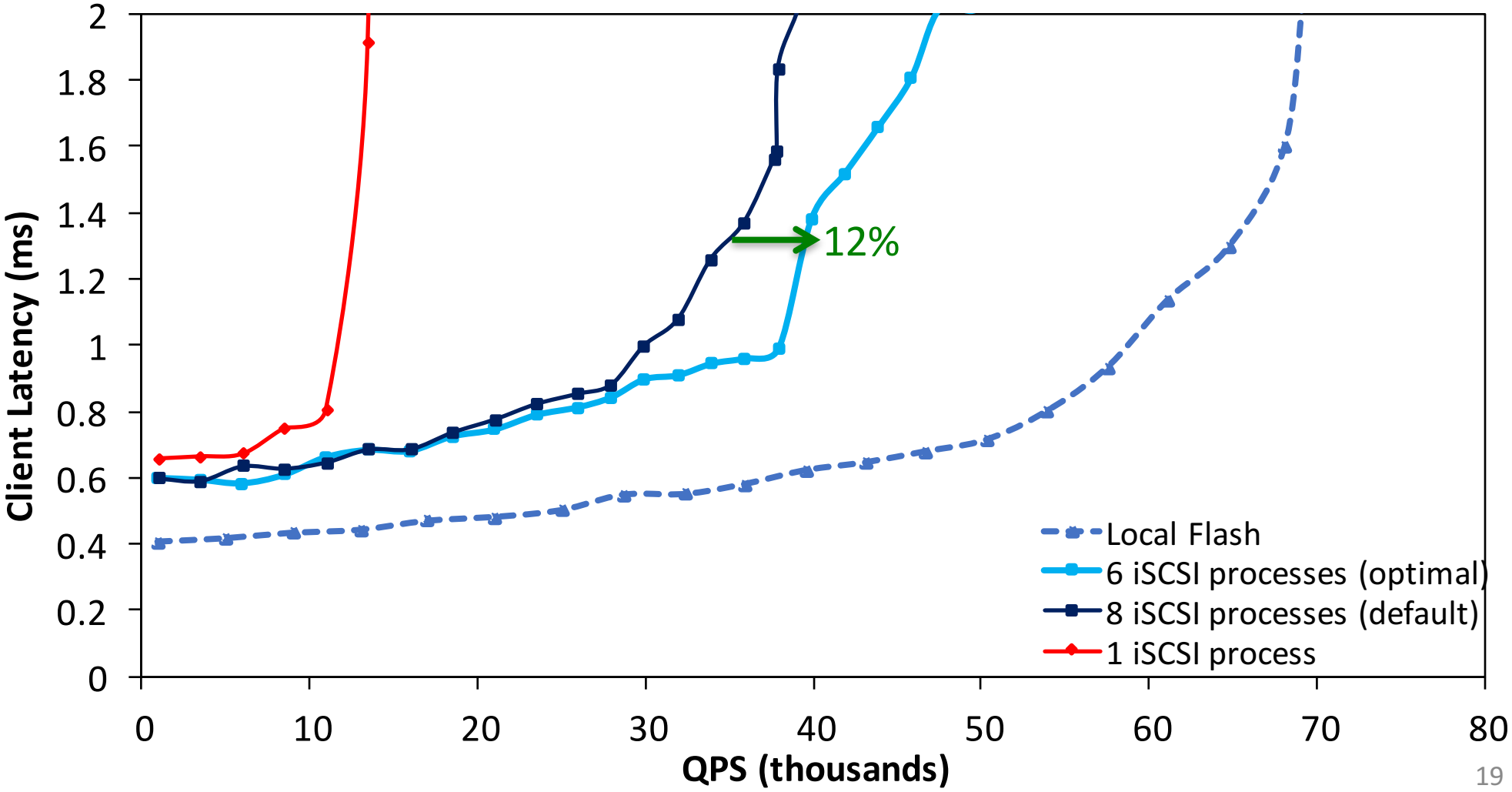
Application Throughput

- 45% throughput drop with “out of the box” iSCSI Flash
- Need to optimize remote Flash server for higher throughput



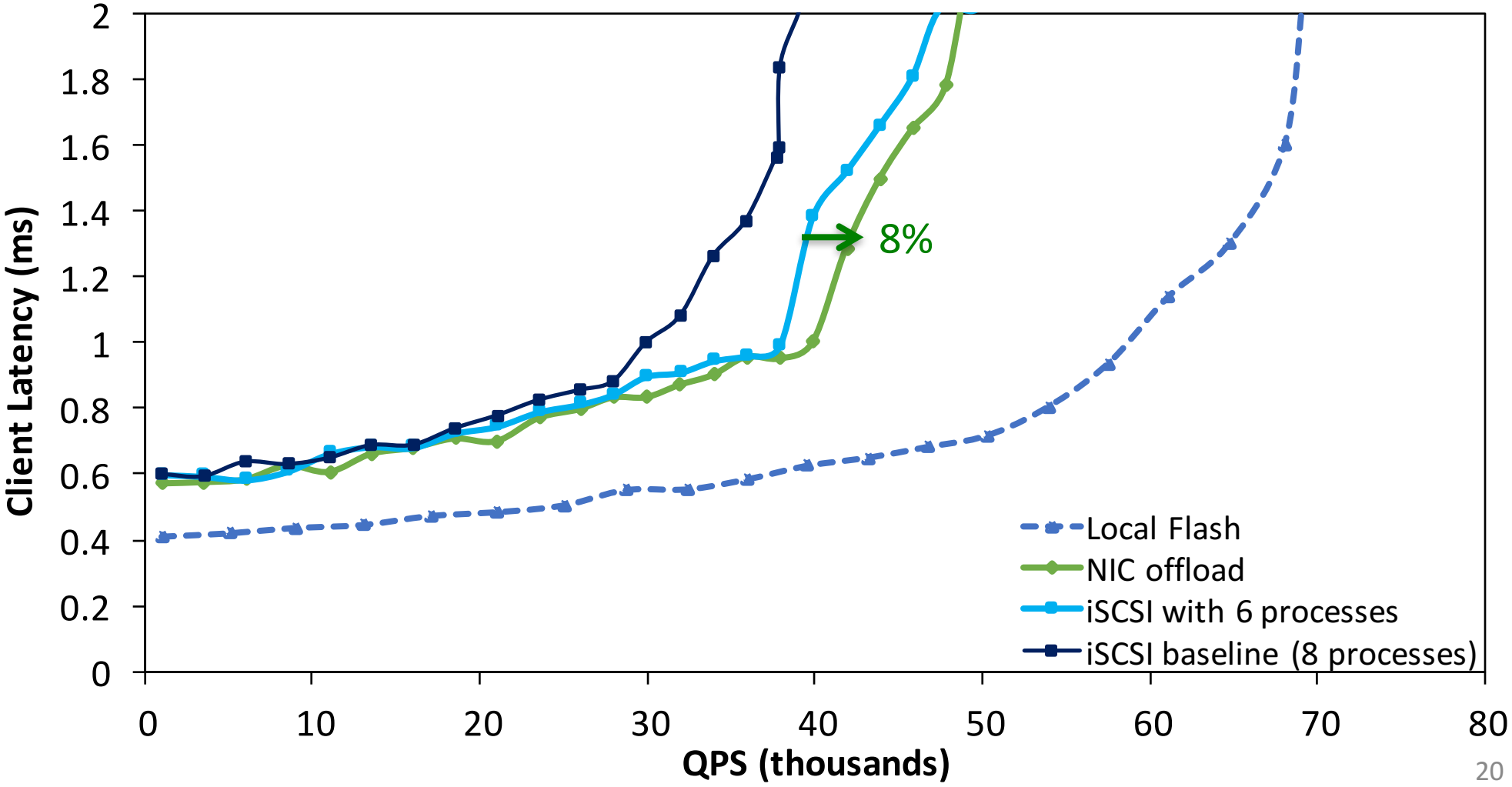
Multi-process iSCSI

- Vary number of iSCSI processes that issue IO
- Want enough parallelism, avoid scheduling interference



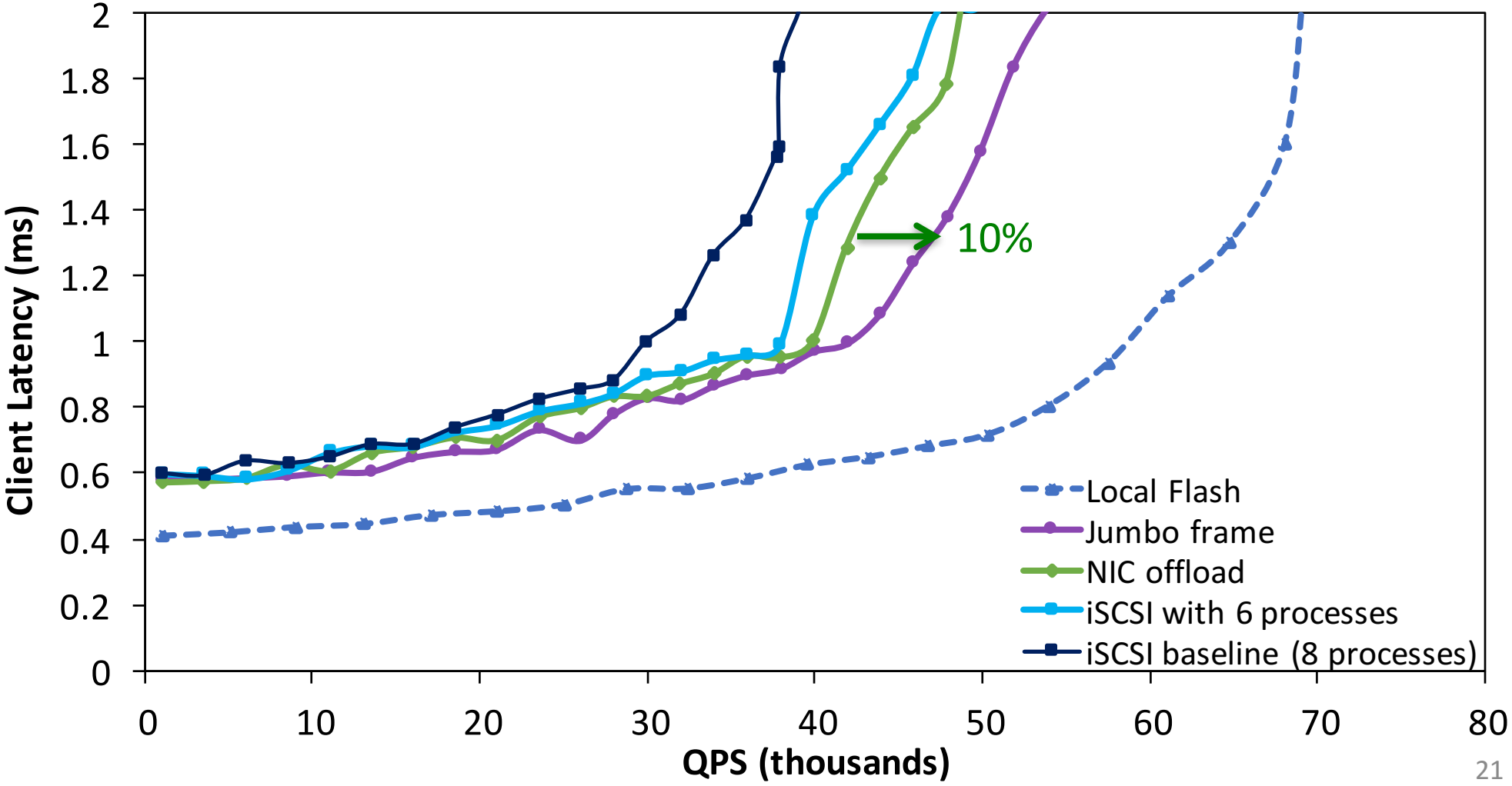
NIC offloads

- Enable NIC offloads for TCP segmentation (TSO/LRO) to reduce CPU load on Flash server and datastore server



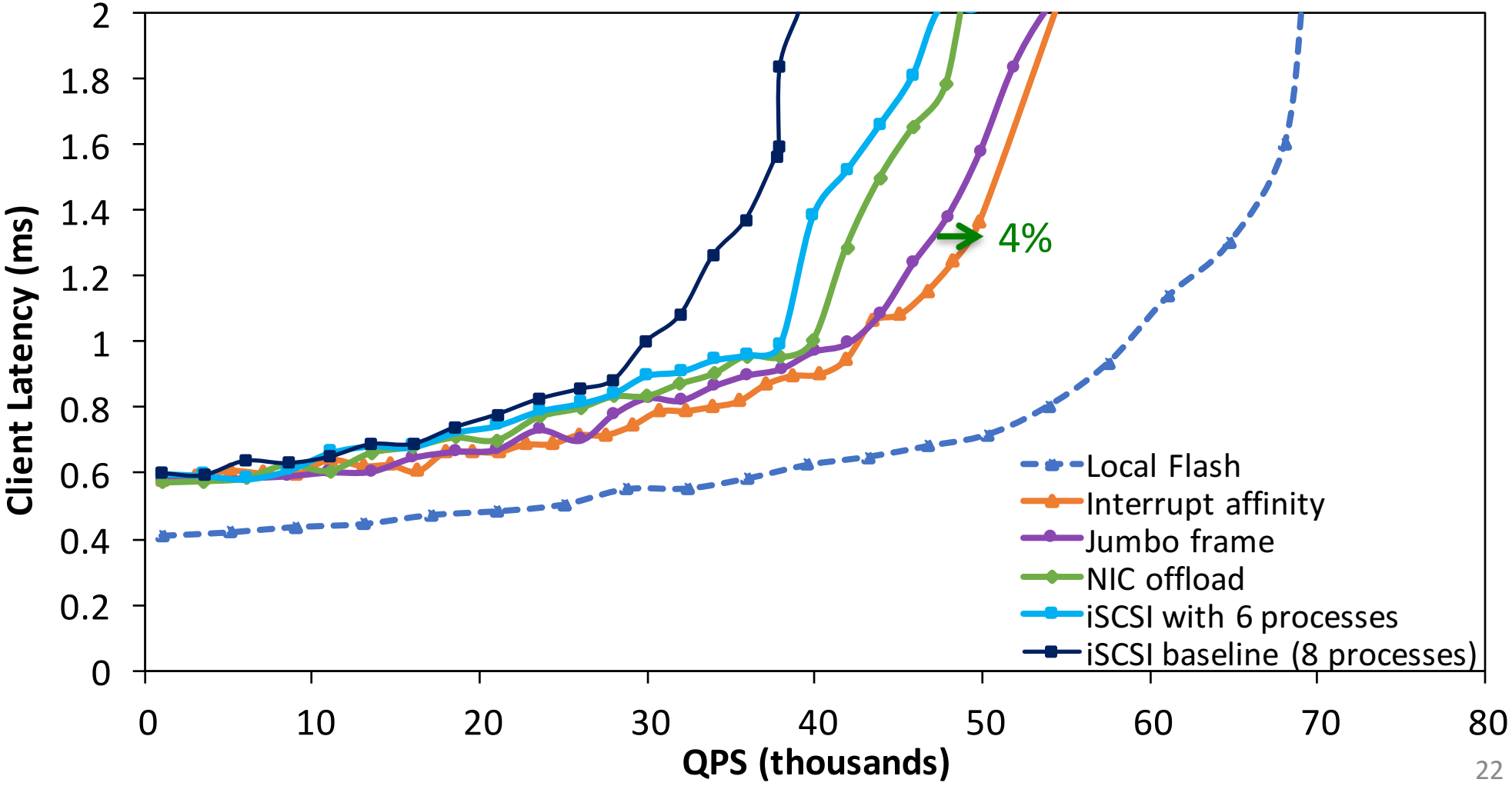
Jumbo Frames

- Jumbo frames further reduce overhead by reducing segmentation altogether (max MTU 9kB)



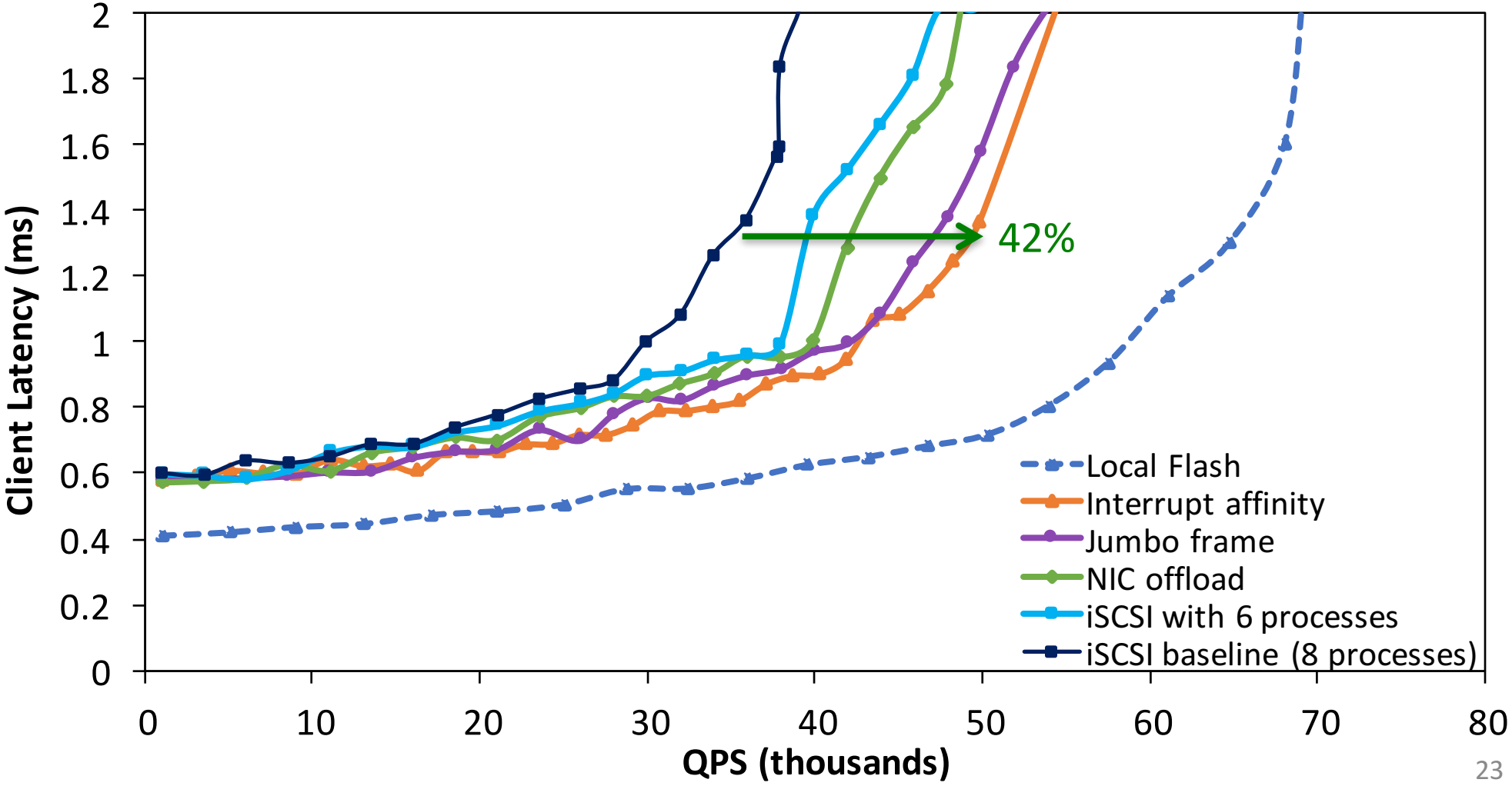
Interrupt Affinity Tuning

- Steer NIC interrupts to core handling TCP connection and Flash interrupts to cores issuing IO commands



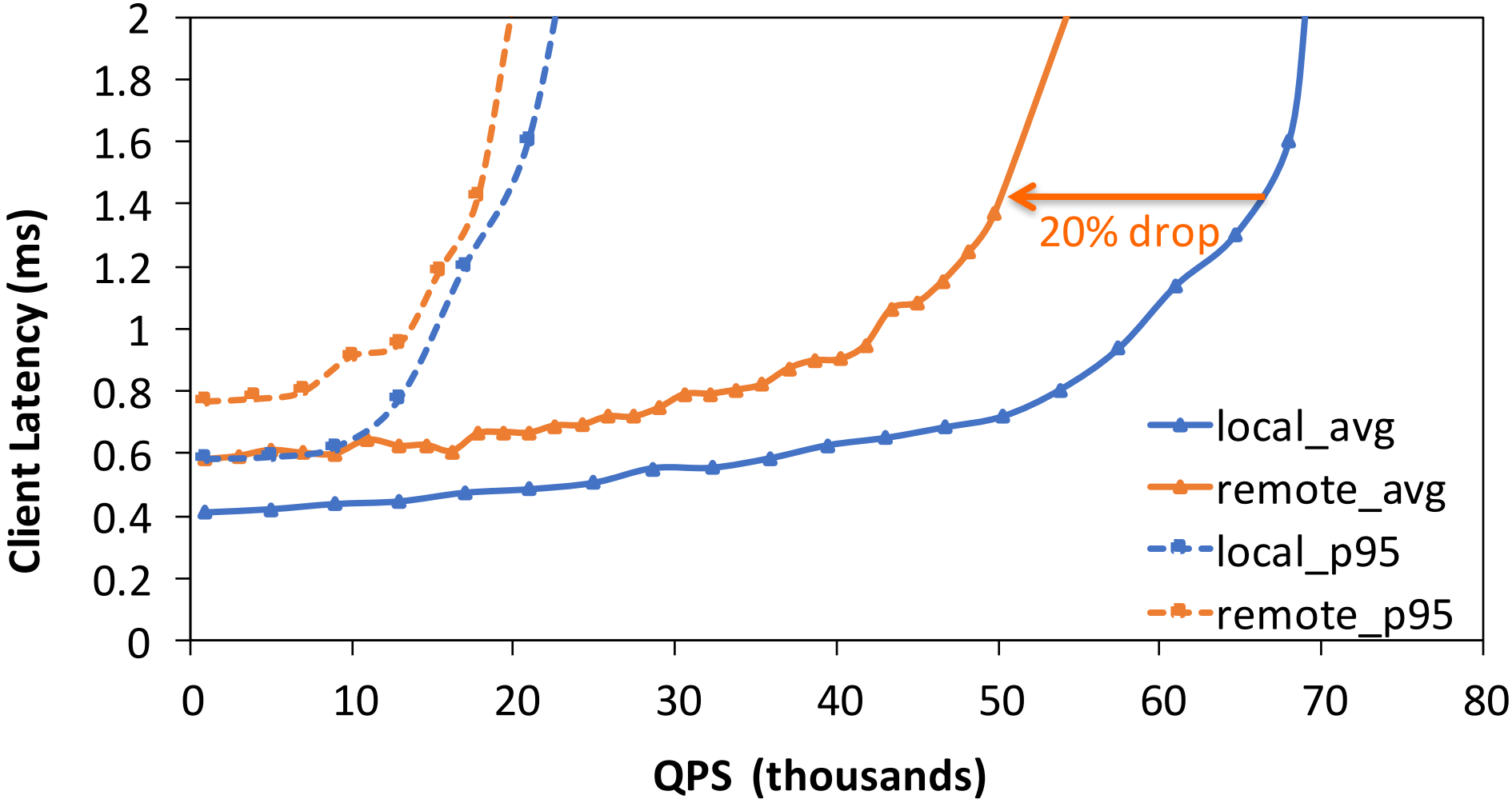
Optimized Application Throughput

- Steer NIC interrupts to core handling TCP connection and Flash interrupts to cores issuing IO commands



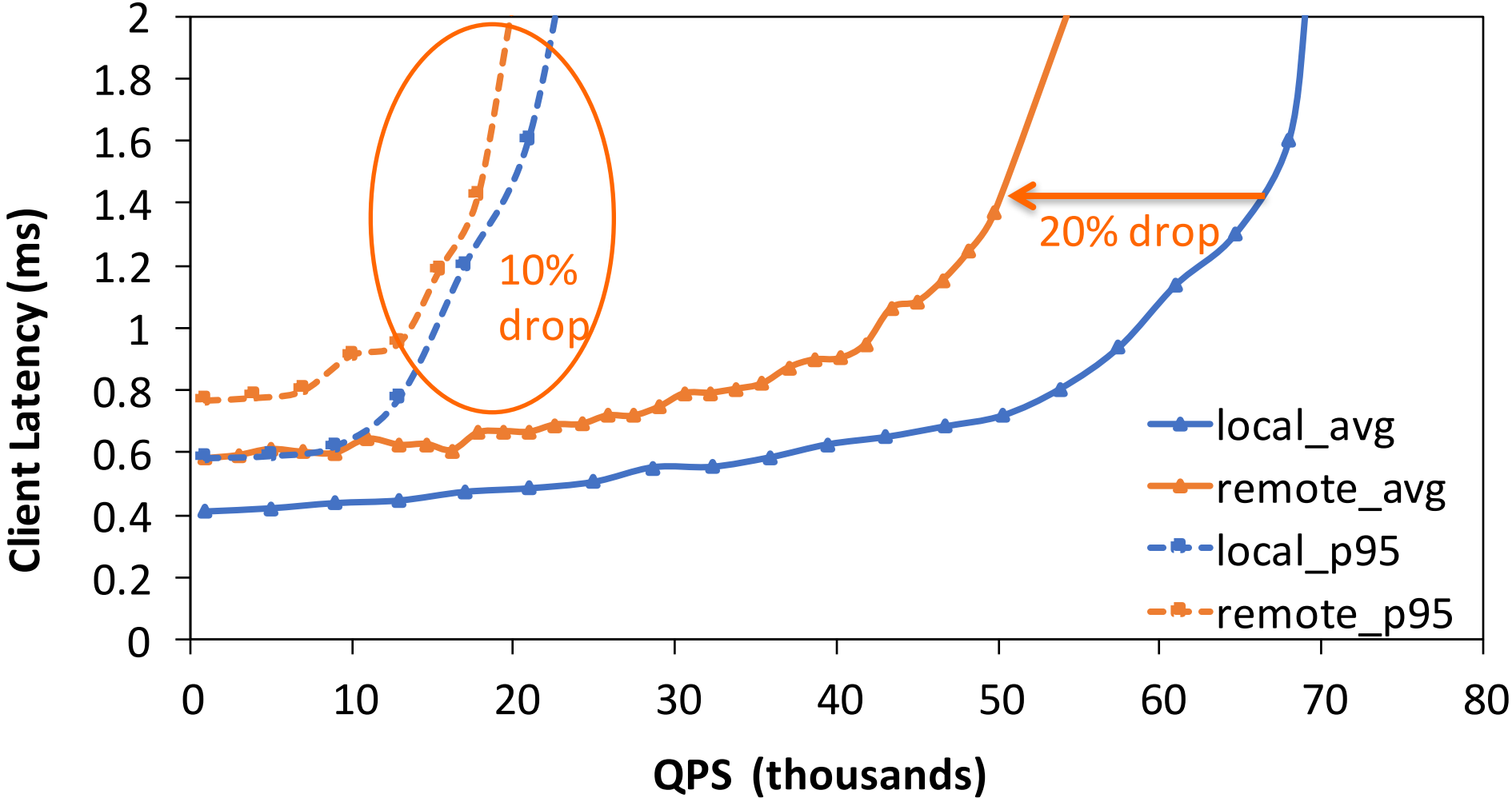
Application Throughput

- 20% drop in application throughput, on average



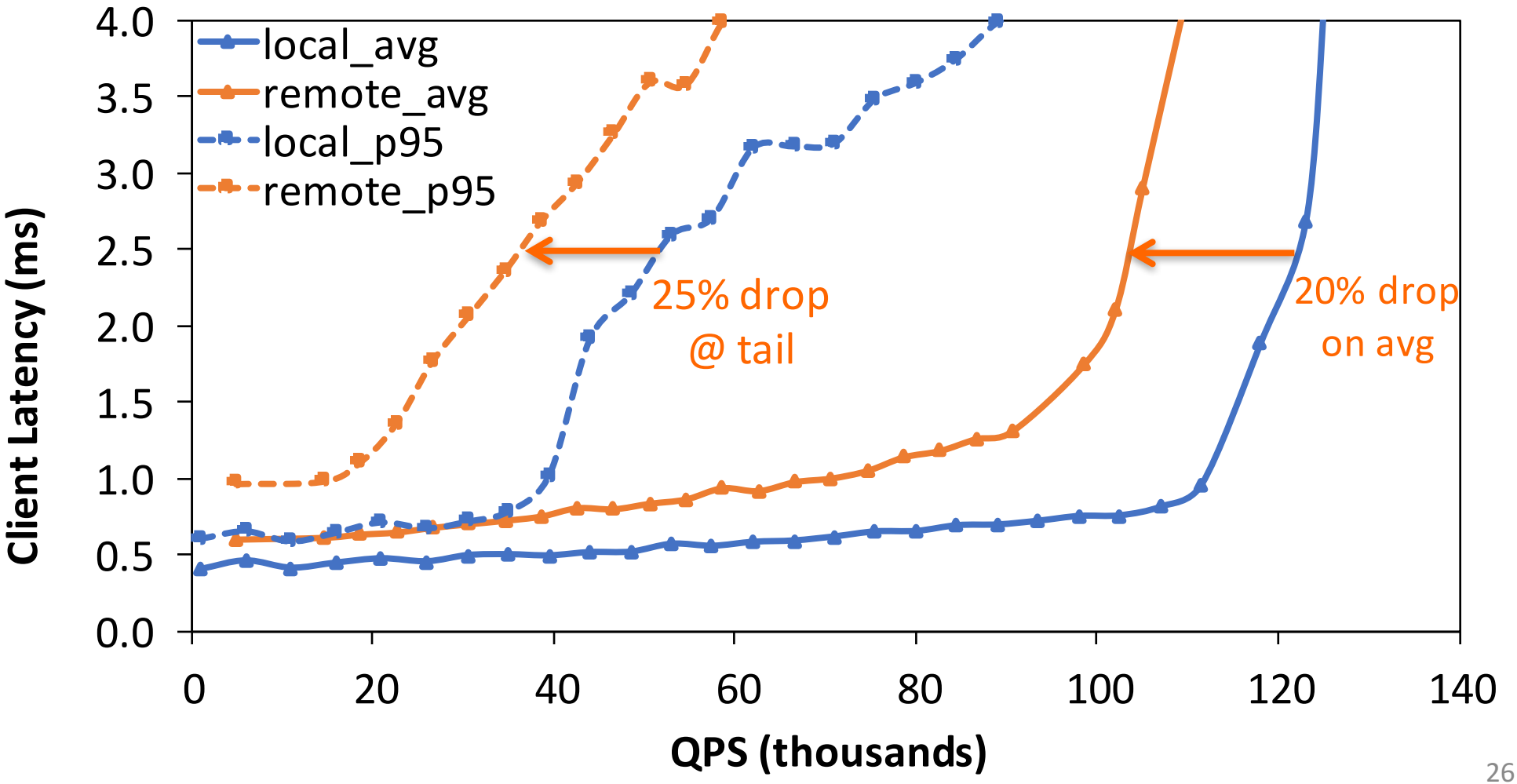
Application Throughput

- At the tail, overhead of remote access is masked by other factors like write interference on Flash



Sharing Remote Flash

- Sharing Flash among 2 or more tenants leads to more write interference → degrades tail performance

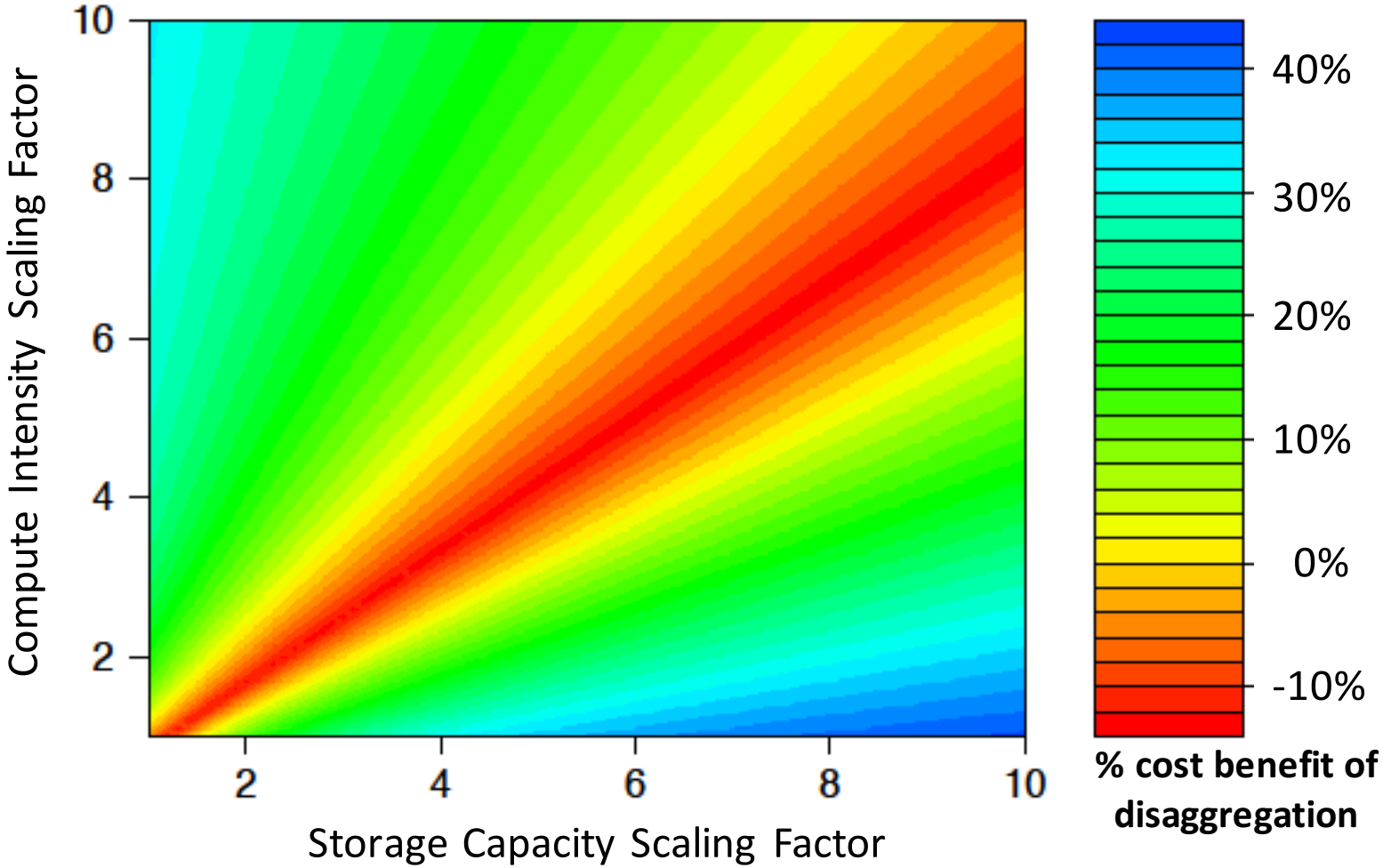


Disaggregation Benefits

- Make up for throughput loss by *cost-effectively* scaling resources with disaggregation
- Improve overall resource utilization
- Formulate cost model to quantify benefits

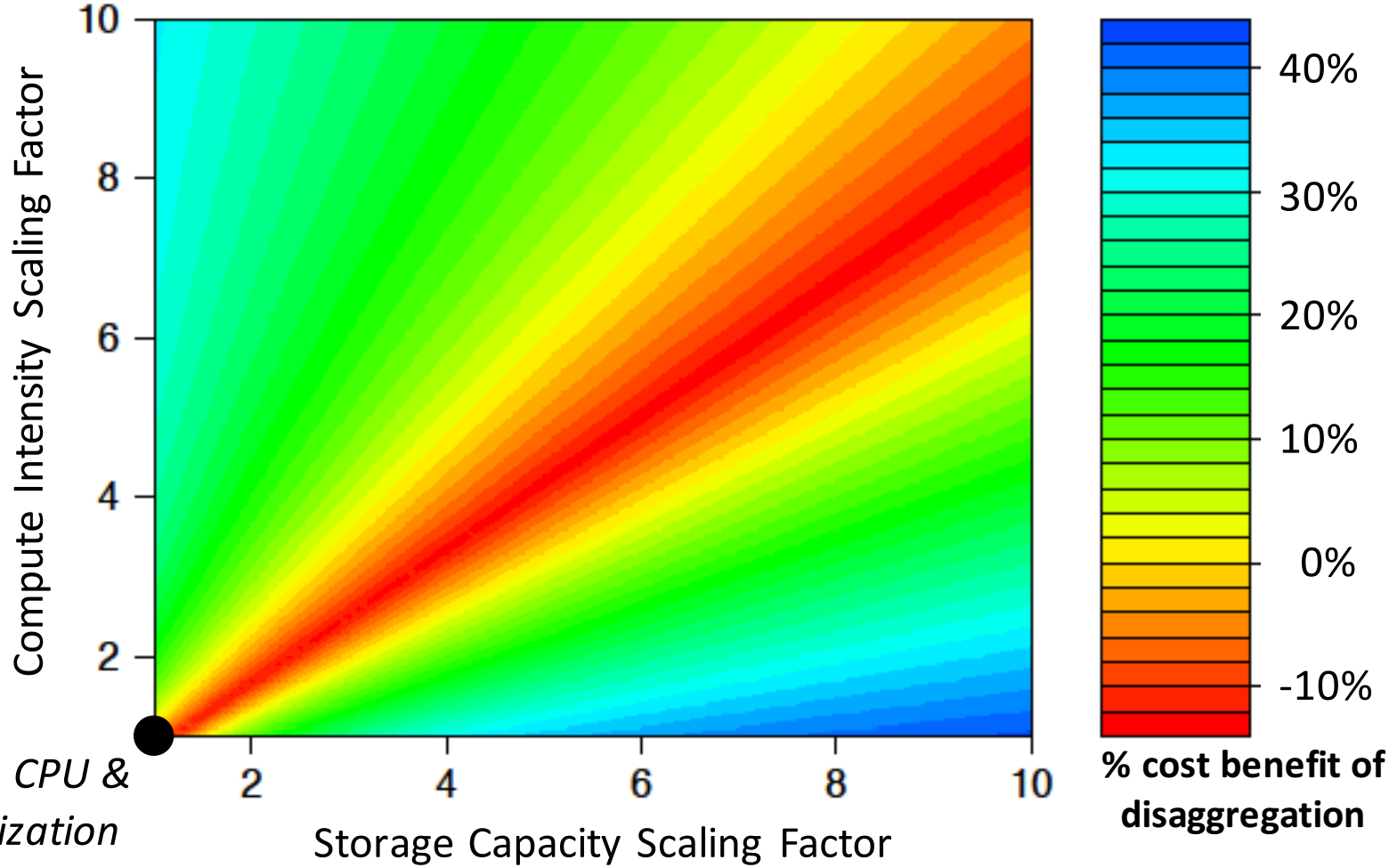
Resource Savings

- Resource savings of disaggregated vs. local Flash architecture as app requirements scale



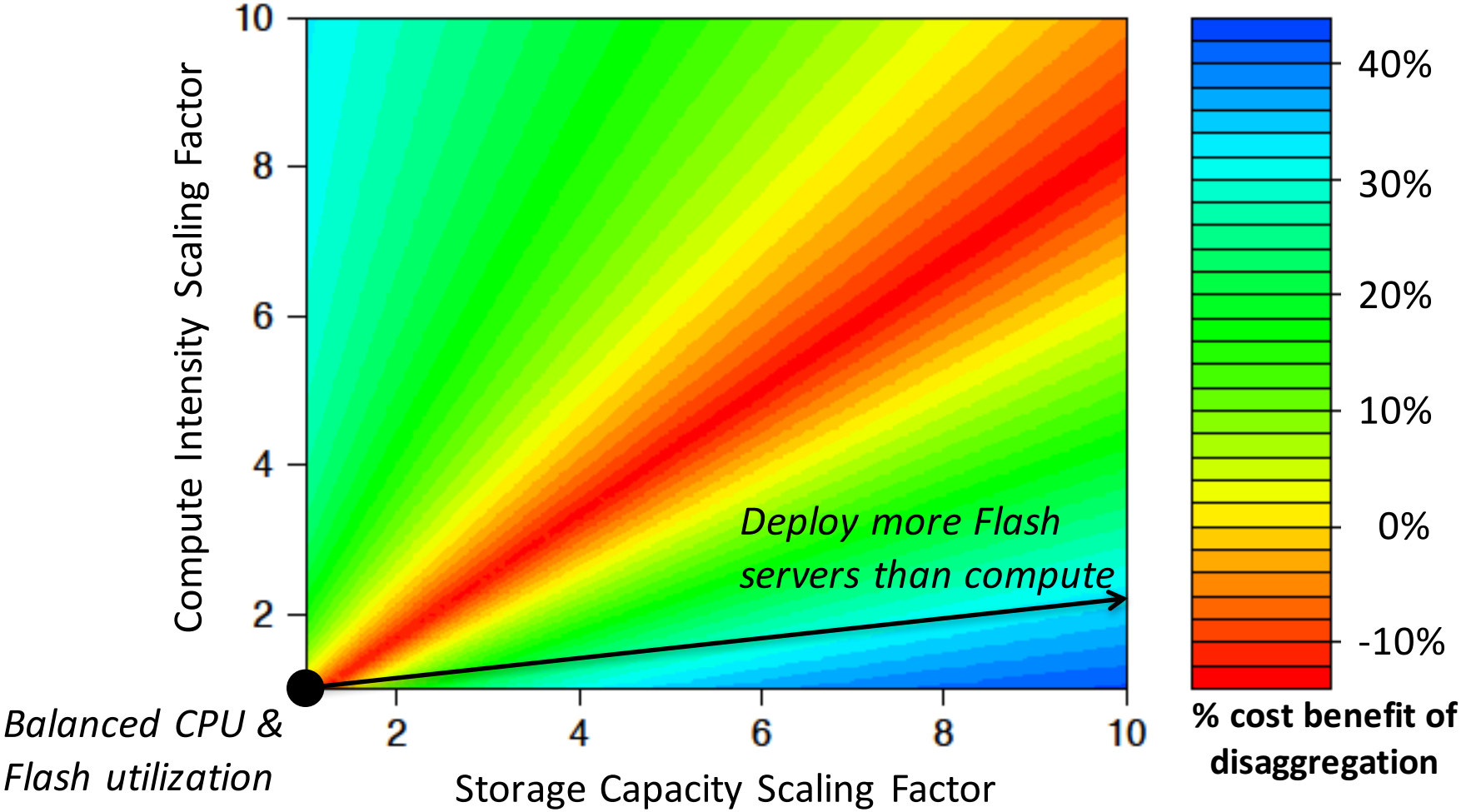
Resource Savings

- Resource savings of disaggregated vs. local Flash architecture as app requirements scale



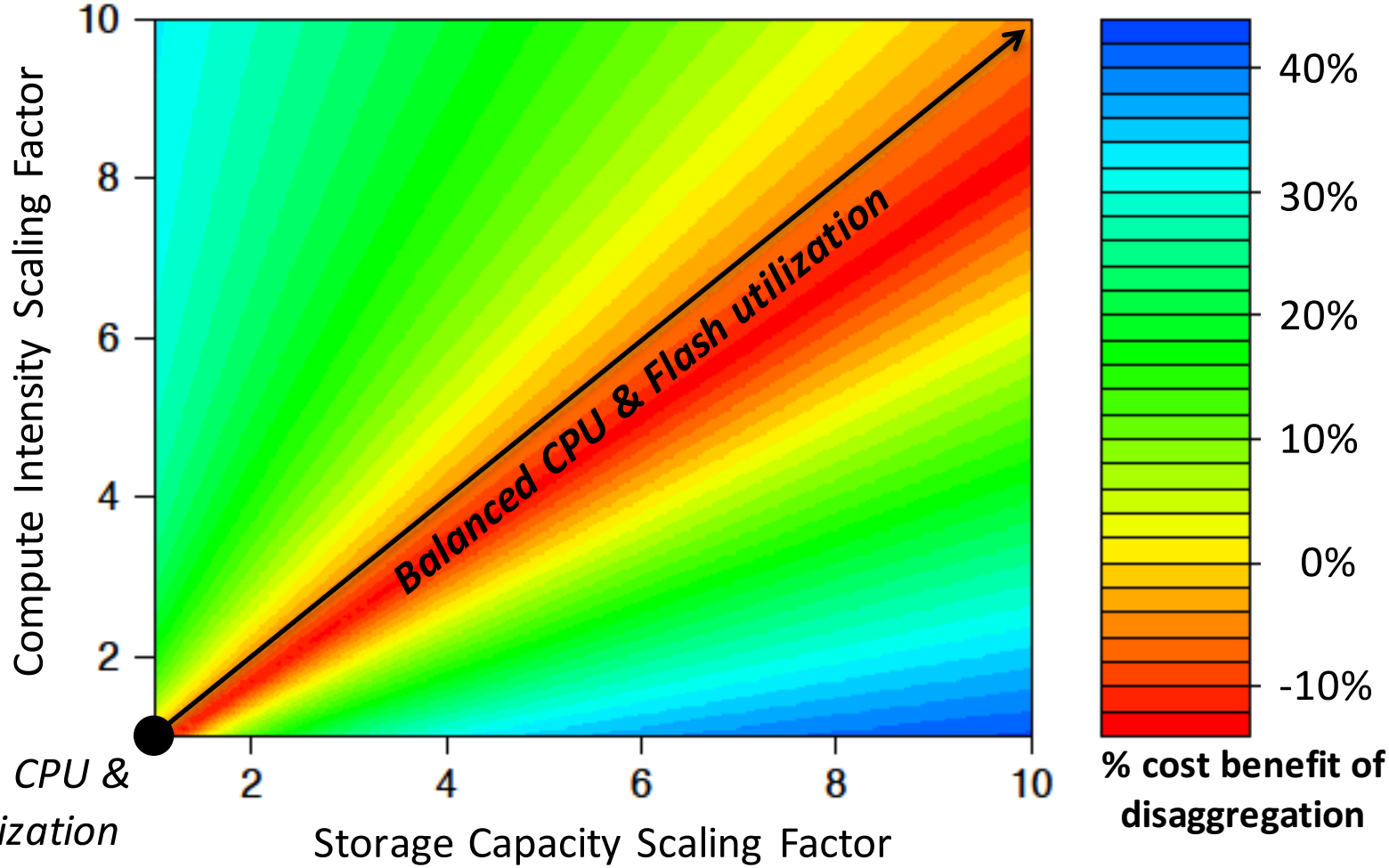
Resource Savings

- When storage scales at higher rate than compute, save resources by deploying Flash without as much CPU



Resource Savings

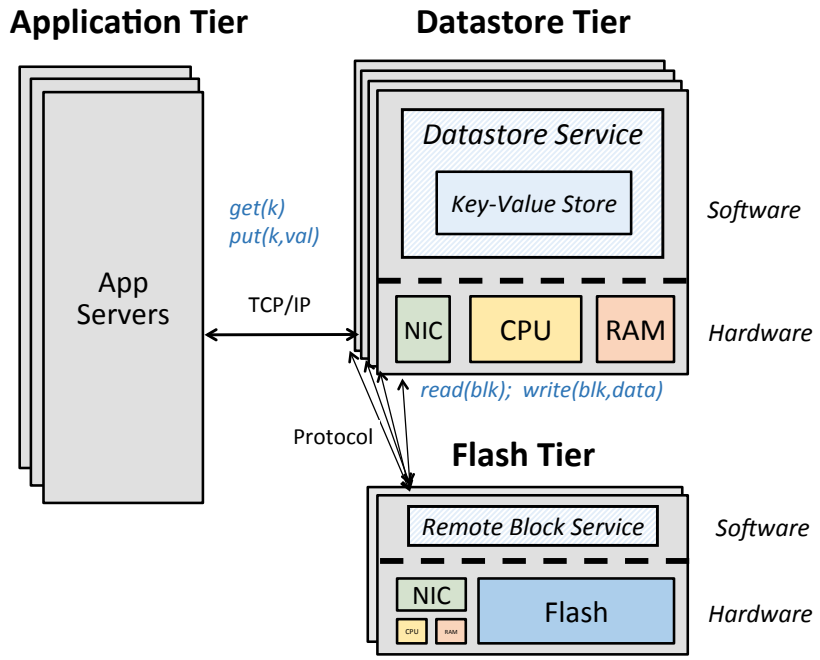
- When compute and storage demands remain balanced, no benefit with disaggregation

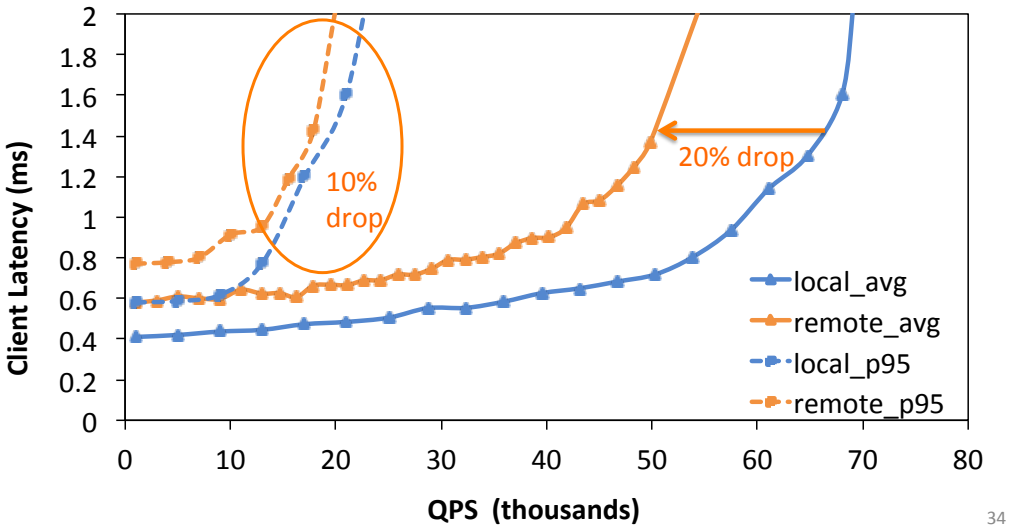


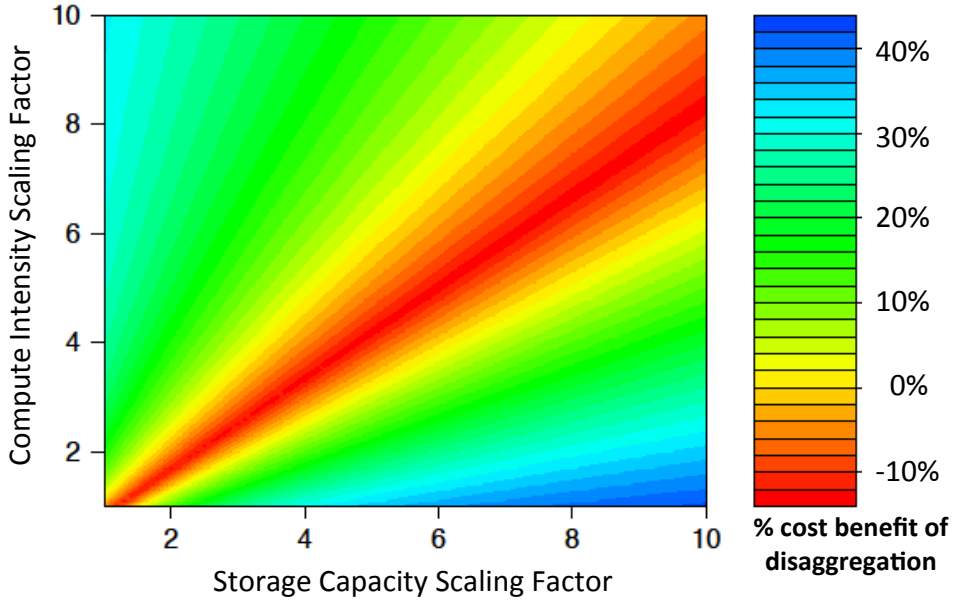
Balanced CPU & Flash utilization

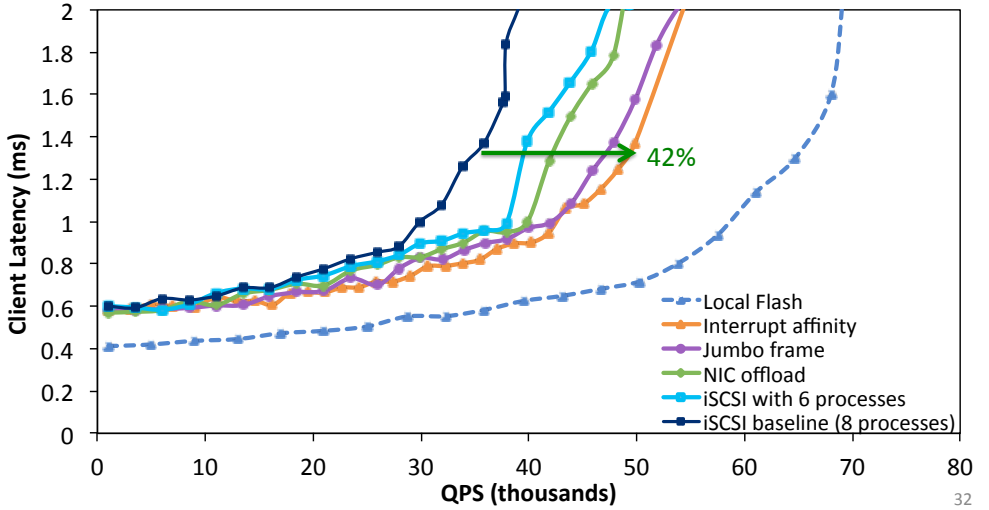
Implications for System Design

- Dataplane:
  - Reduce compute overhead of network (storage) stack
    - Optimize TCP/IP processing
    - Use a light-weight protocol
  - Provide isolation mechanisms for shared remote Flash
- Control plane:
  - Policies for allocating and sharing remote Flash
    - Important to consider write IO patterns of applications


Application Tier
Datastore Tier
App Servers
get(k)
put(k,val)
TCP/IP
Datastore Service
Key-Value Store
Software
NIC
CPU
RAM
Hardware
read(blk); write(blk,data)
Protocol
Flash Tier
Remote Block Service
Software
NIC
Flash
Hardware


Client Latency (ms)
QPS (thousands)
10% drop
20% drop
local_avg
remote_avg
local_p95
remote_p95


Compute Intensity Scaling Factor
Storage Capacity Scaling Factor
40%
30%
20%
10%
0%
-10%
% cost benefit of disaggregation


Client Latency (ms)
QPS (thousands)
42%
Local Flash
Interrupt affinity
Jumbo frame
NIC offload
iSCSI with 6 processes
iSCSI baseline (8 processes)

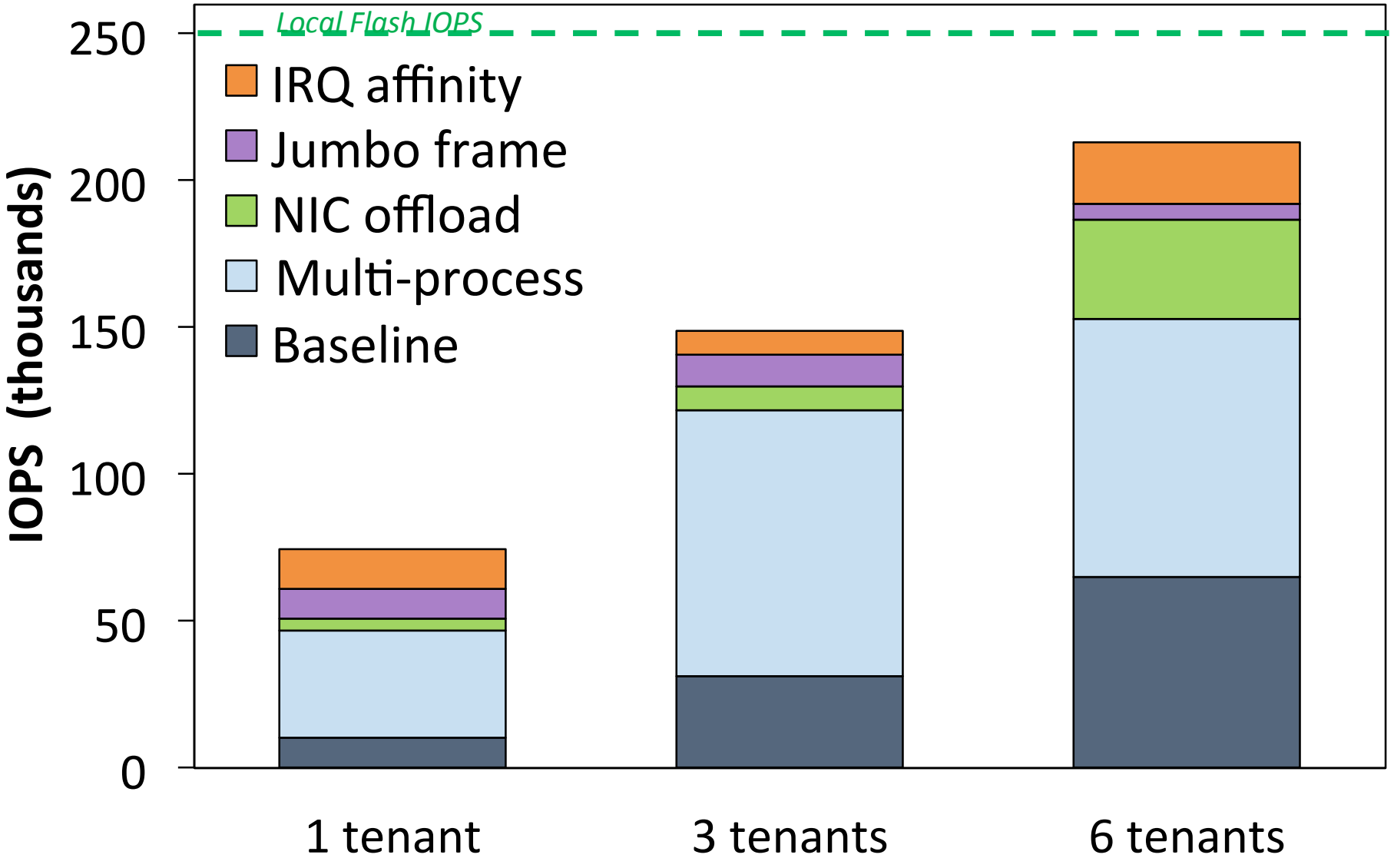
Conclusion

- Disaggregating Flash is beneficial because it allows us to cost-effectively scale resources:
  - Improve overall resource efficiency
  - Compensate for 20% throughput overhead by independently deploying application resources
- System tuning improves performance ~40%, more opportunities if redesign software stack

Backup

Remote Flash IOPS

IO-intensive benchmark: 4kB random reads



Cost Model

$$C_{direct} = \max\left(\frac{\text{GB}_t}{\text{GB}_s}, \frac{\text{IOPS}_t}{\text{IOPS}_s}, \frac{\text{QPS}_t}{\text{QPS}_s}\right) \cdot (f + c)$$

$$C_{disagg} = \max\left(\frac{\text{GB}_t}{\text{GB}_s}, \frac{\text{IOPS}_t}{\text{IOPS}_s}\right) \cdot (f + \delta) + \left(\frac{\text{QPS}_t}{\text{QPS}_s}\right) c$$

where:

f: cost of Flash on a server

c: cost of CPU, RAM and NIC on datastore server

δ: cost of CPU, RAM and NIC on Flash tier server, i.e. resource "tax" for disaggregation

x_s: x provided by a single server, $x = \{\text{GB, IOPS, QPS}\}$

x_t: x required in total for the application

Related Work

- Disaggregated disk storage:
  - Petal [ASPLOS'96], Parallax [HotOS'05], Blizzard [NSDI'14]
- Disaggregated Flash as distributed shared log:
  - CORFU [NSDI'12], FAWN [SOSP'09]
- Disaggregated memory:
  - Memory blade servers (Lim et al.) [ISCA'09]
- Rack-scale disaggregation:
  - Pelican [OSDI'14], HP Moonshot, Intel Rack-Scale