ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

Rearchitecting Linux Storage Stack for μs Latency and High Throughput




Jaehyun Hwang *Cornell University*

Midhul Vuppalapati *Cornell University*

Simon Peter *UT Austin*

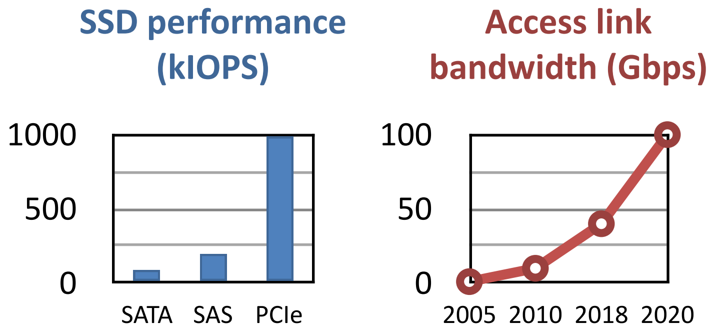
Rachit Agarwal *Cornell University*

Widespread belief: Linux can't achieve *μs*-scale latency & high throughput

Widespread belief: Linux can't achieve μs-scale latency & high throughput

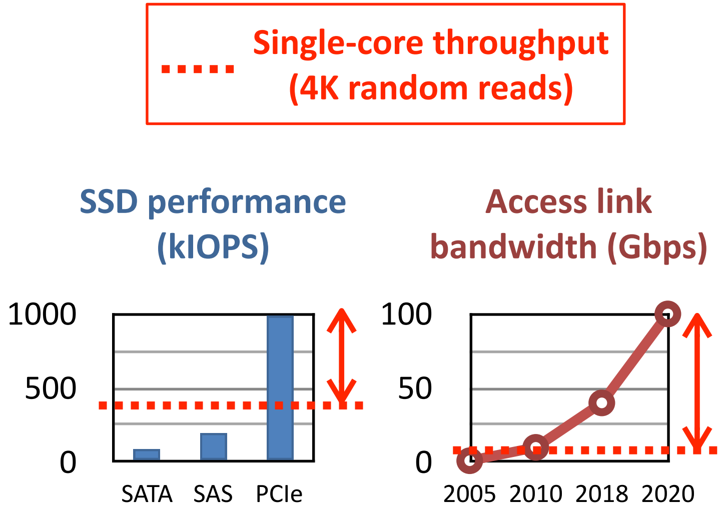
Widespread belief: Linux can't achieve μs-scale latency & high throughput

1. Adoption of high performance H/W, but stagnant single-core capacity



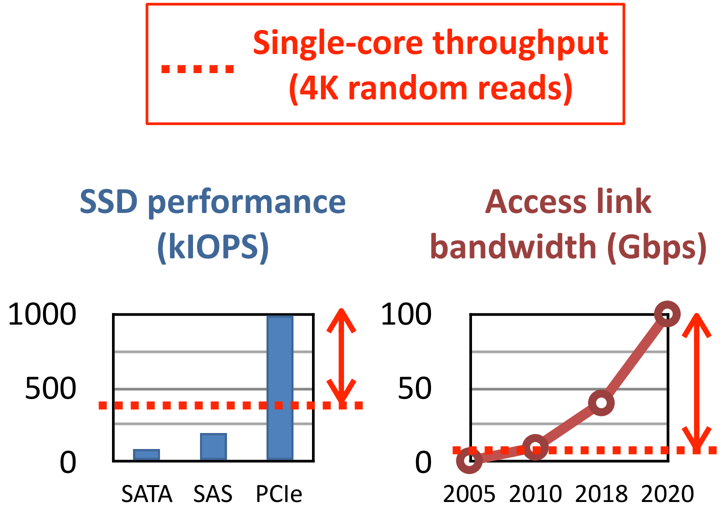
Widespread belief: Linux can't achieve μs-scale latency & high throughput

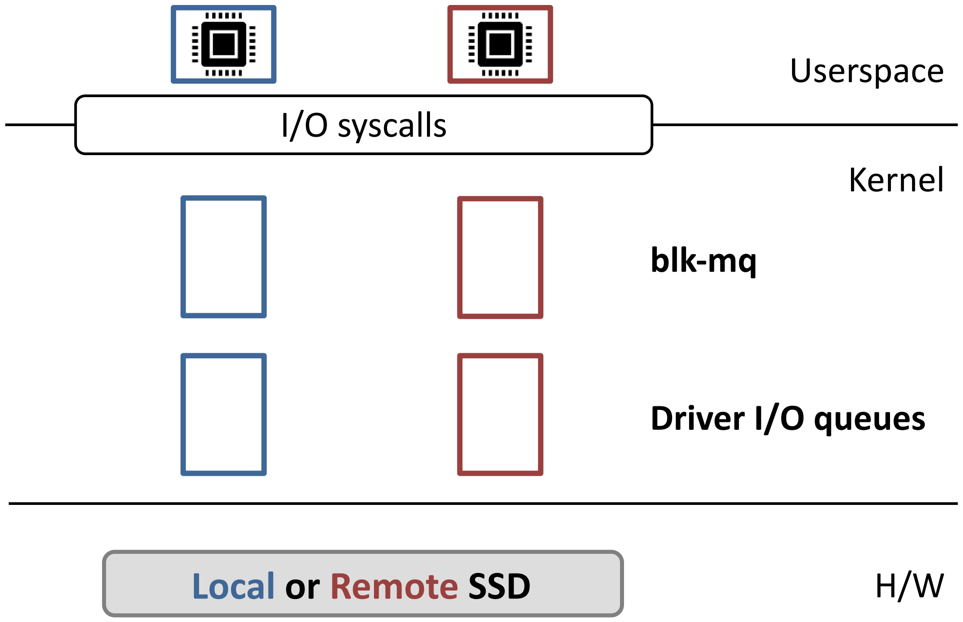
1. Adoption of high performance H/W, but stagnant single-core capacity



Widespread belief: Linux can't achieve μs-scale latency & high throughput

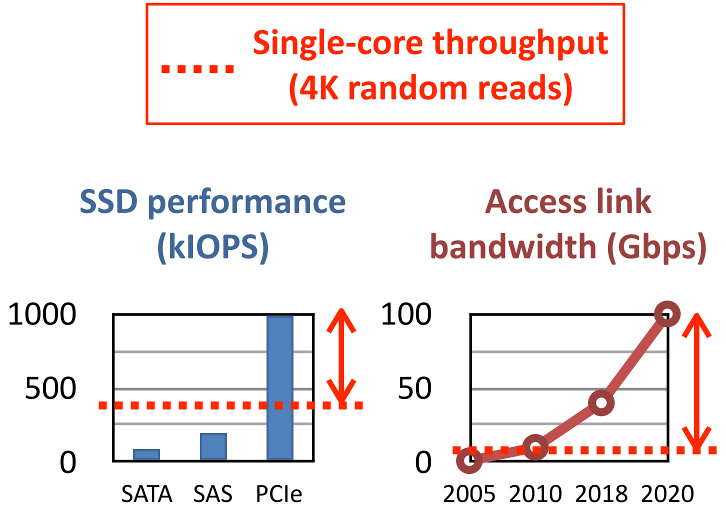
1. Adoption of high performance H/W, but stagnant single-core capacity

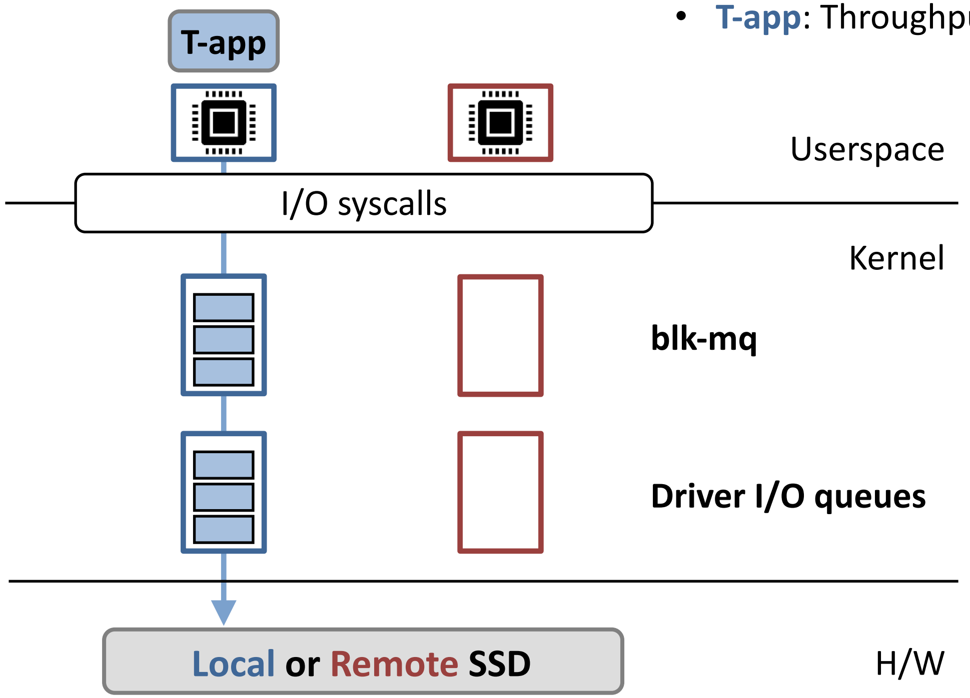




Widespread belief: Linux can't achieve μs-scale latency & high throughput

1. Adoption of high performance H/W, but stagnant single-core capacity

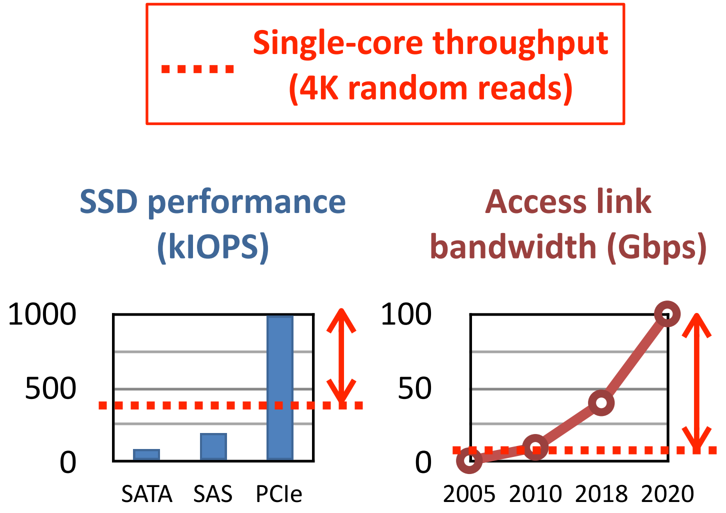


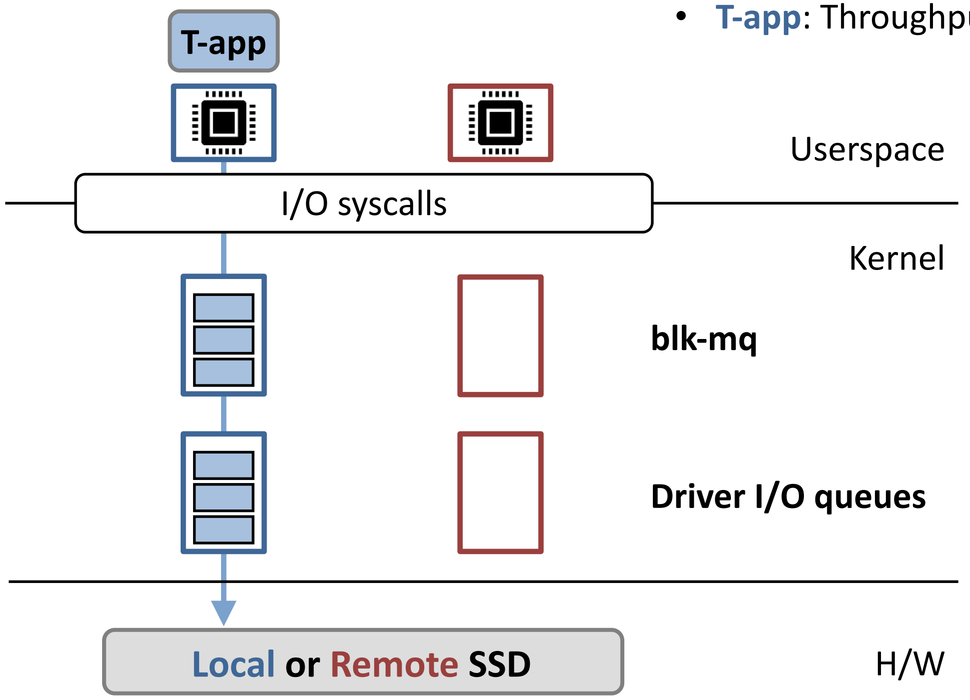


- **T-app**: Throughput-bound app

Widespread belief: Linux can't achieve μs-scale latency & high throughput

1. Adoption of high performance H/W, but stagnant single-core capacity





- T-app: Throughput-bound app

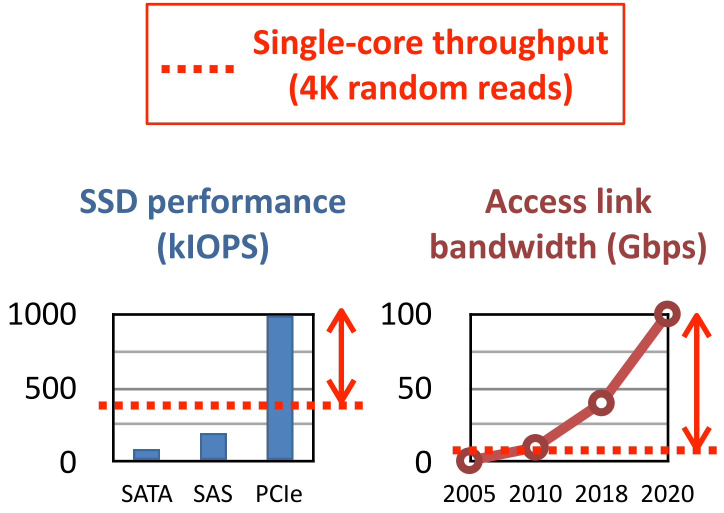
Static data path ⇢ hard to utilize all cores

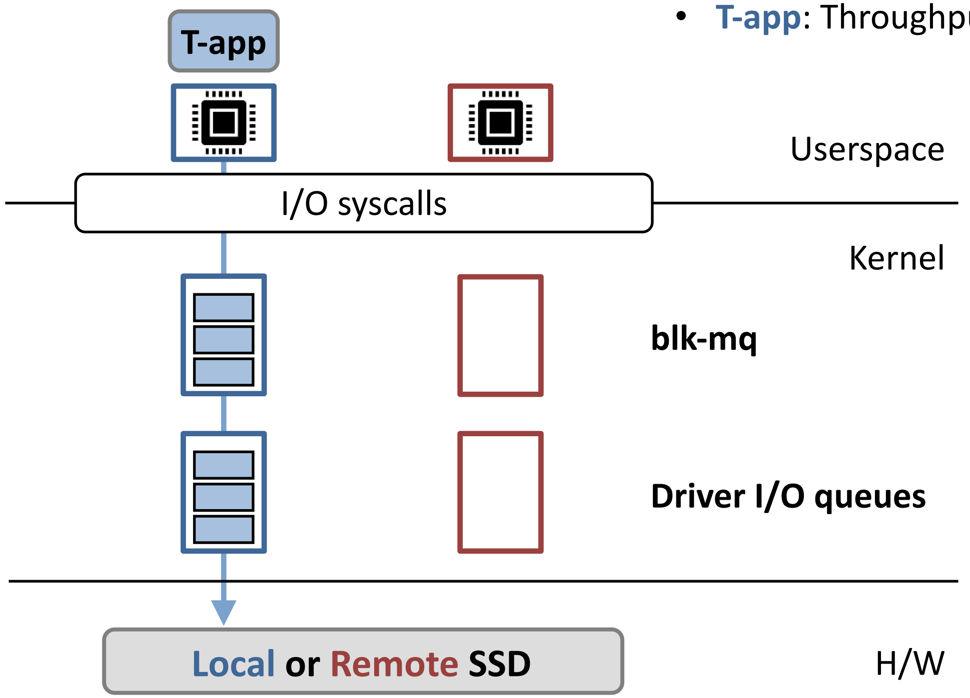
Widespread belief: Linux can't achieve μs-scale latency & high throughput

1. Adoption of high performance H/W, but stagnant single-core capacity

2. Co-location of apps with different performance goals

- T-app: Throughput-bound app



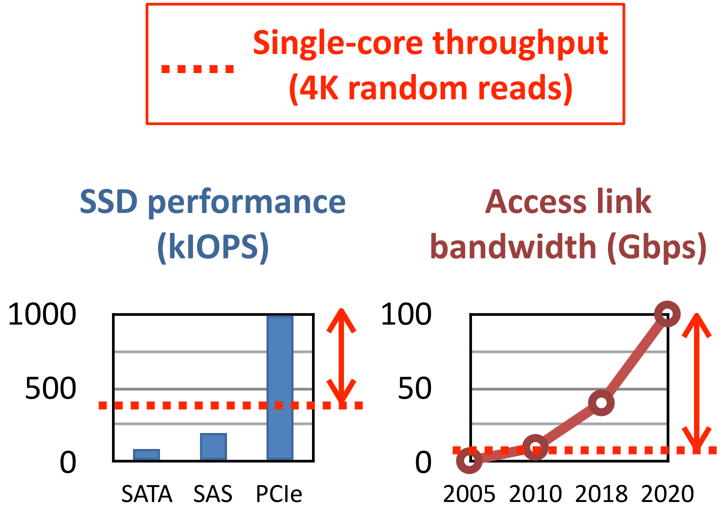


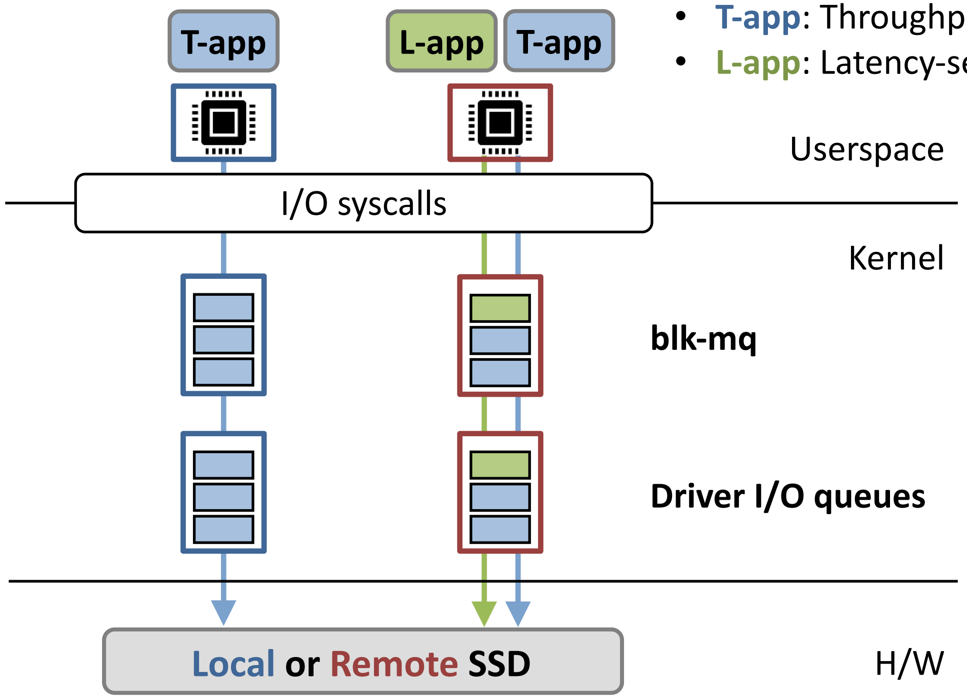
Static data path ⇢ hard to utilize all cores

Widespread belief: Linux can't achieve μs-scale latency & high throughput

1. Adoption of high performance H/W, but stagnant single-core capacity

2. Co-location of apps with different performance goals





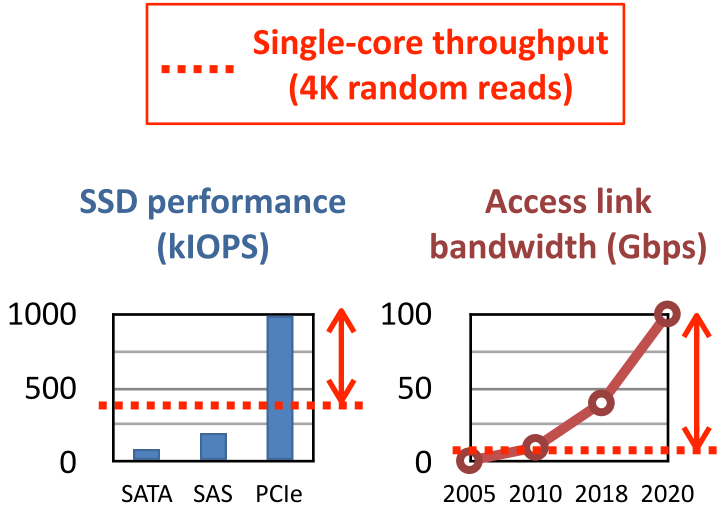
- T-app: Throughput-bound app
- L-app: Latency-sensitive app

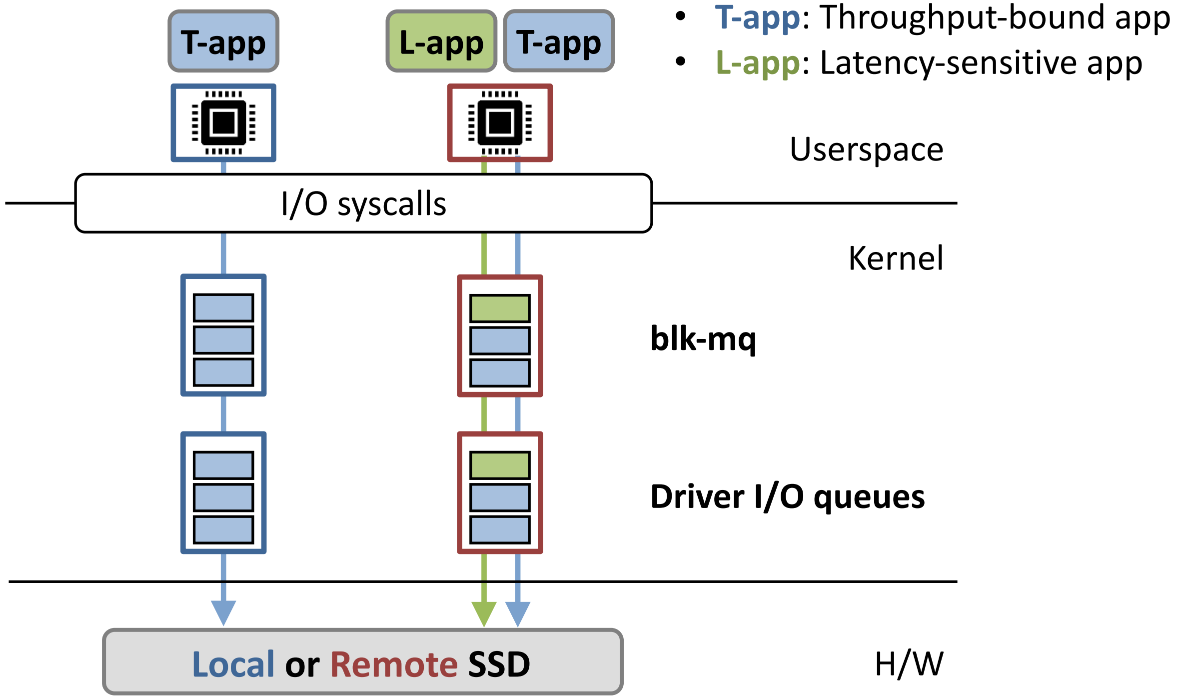
Static data path ⇢ hard to utilize all cores

Widespread belief: Linux can't achieve μs-scale latency & high throughput

1. Adoption of high performance H/W, but stagnant single-core capacity

2. Co-location of apps with different performance goals





Static data path ⇢ hard to utilize all cores

High latency due to HoL blocking

Performance of Existing Storage Stacks

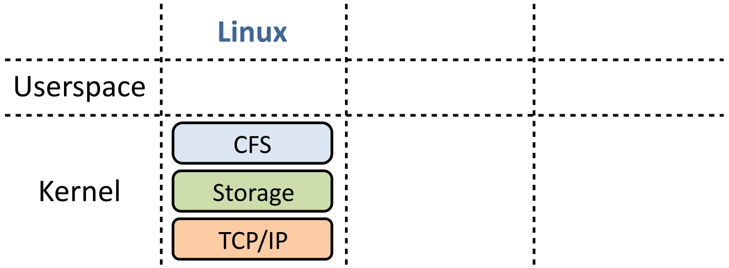
Applications accessing in-memory data in remote servers (single-core case)

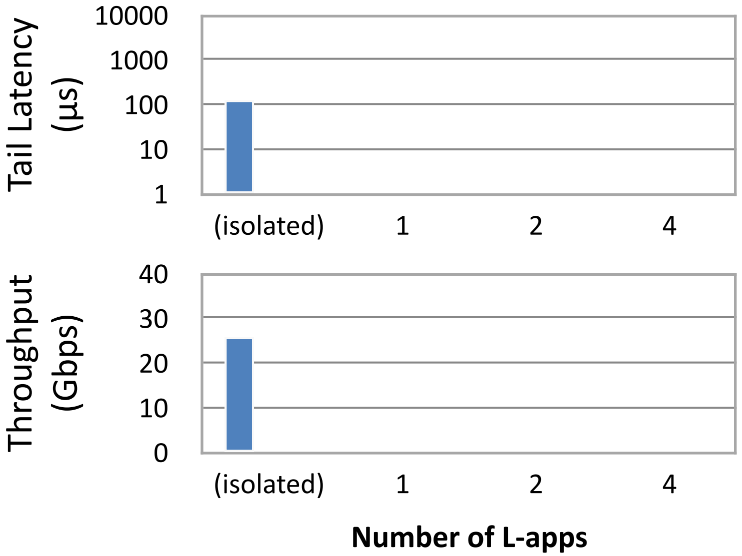
L-app: 4KB, **T-app**: 64–128KB

Performance of Existing Storage Stacks

Applications accessing in-memory data in remote servers (single-core case)

L-app: 4KB, **T-app**: 64–128KB

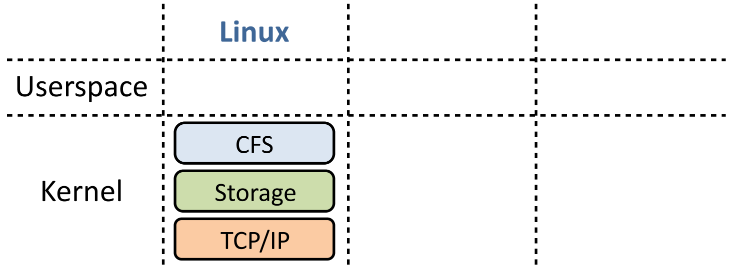


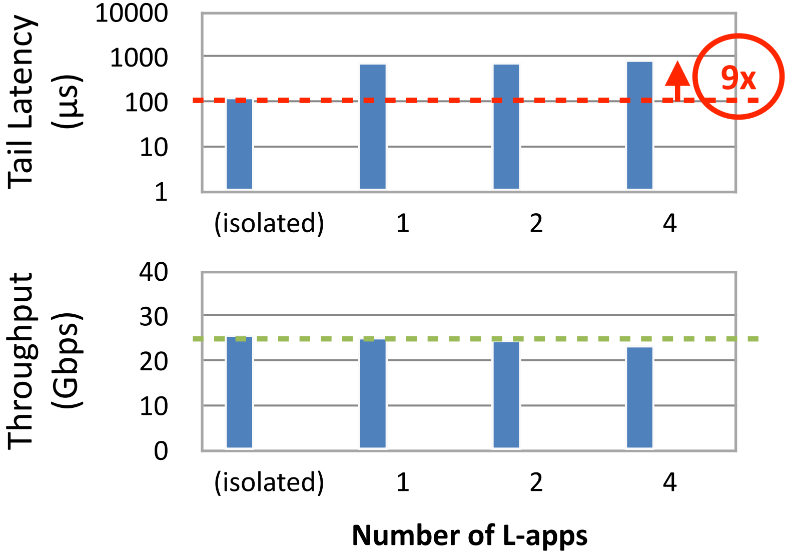


Performance of Existing Storage Stacks

Applications accessing in-memory data in remote servers (single-core case)

L-app: 4KB, **T-app**: 64–128KB





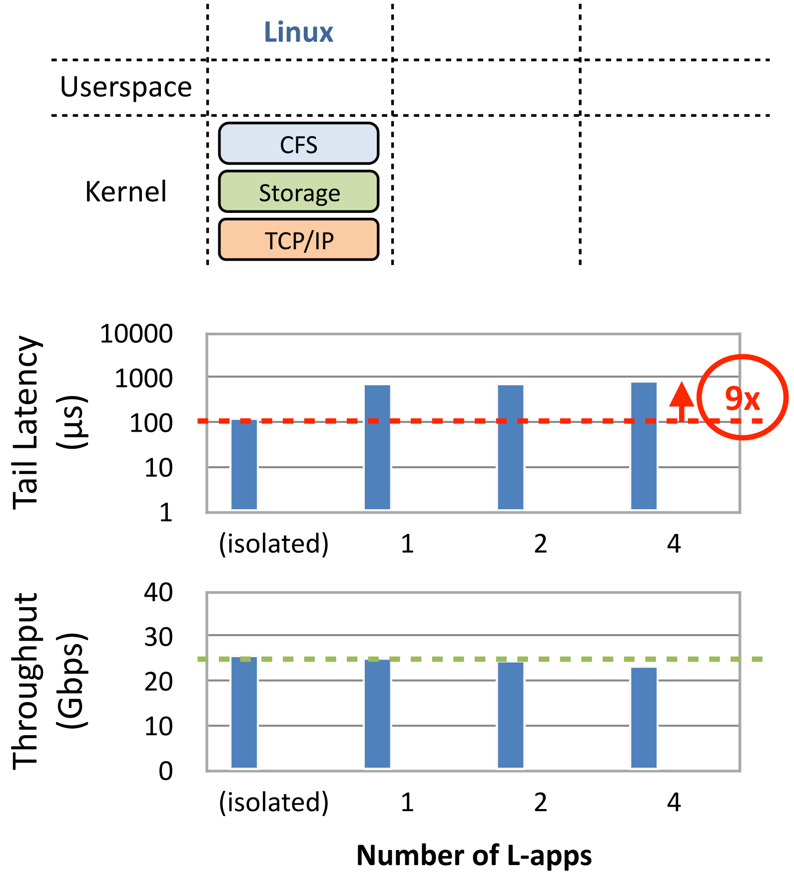
	µs-scale Latency	High Throughput
Linux	✗	✓

Performance of Existing Storage Stacks

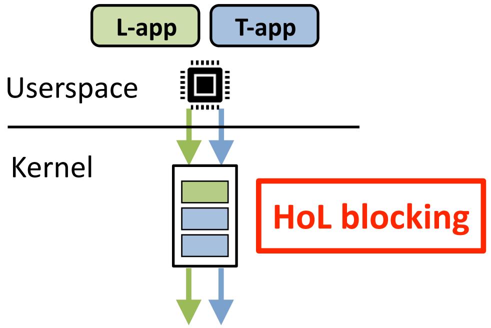
Applications accessing in-memory data in remote servers (single-core case)

L-app: 4KB, **T-app**: 64–128KB

*Detailed root cause analysis is in the paper



	µs-scale Latency	High Throughput
Linux	✗	✓

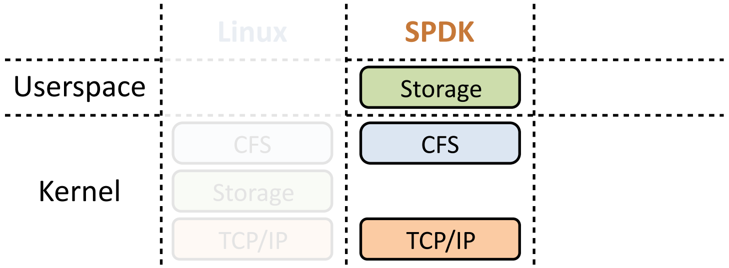


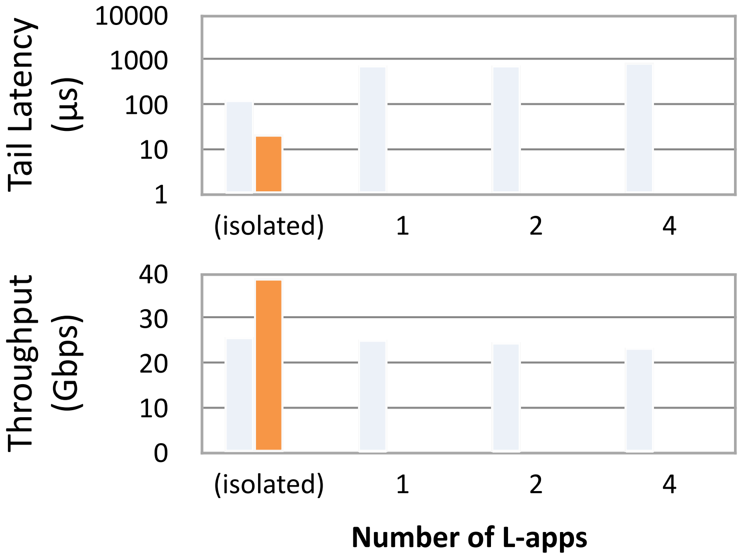
Performance of Existing Storage Stacks

Applications accessing in-memory data in remote servers (single-core case)

L-app: 4KB, **T-app**: 64–128KB

*Detailed root cause analysis is in the paper





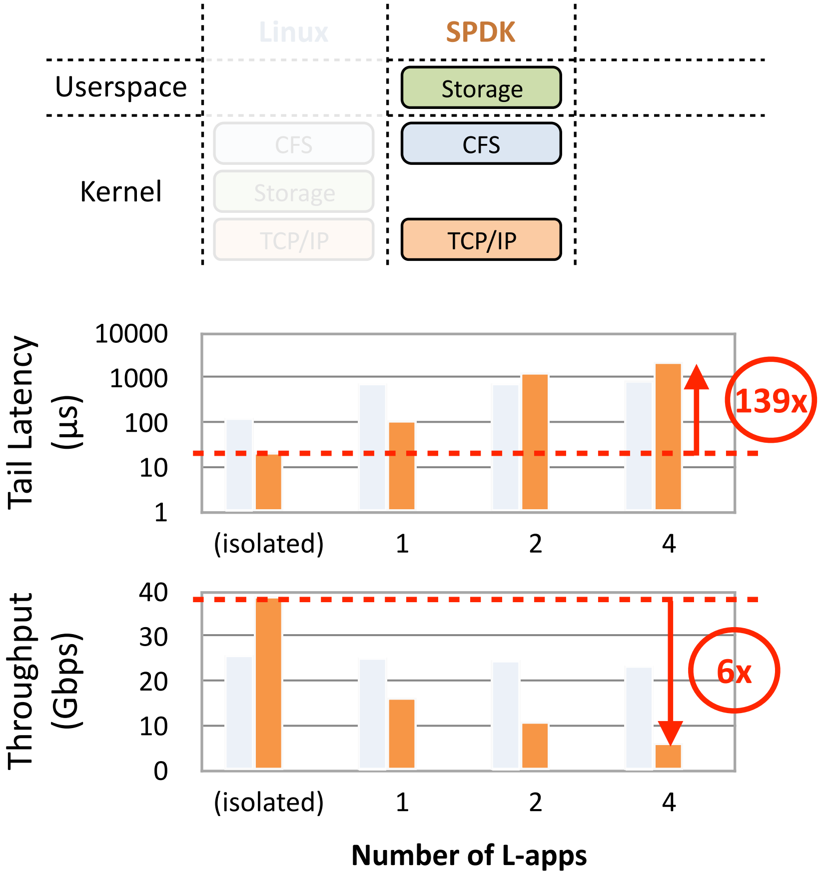
	μs-scale Latency	High Throughput
Linux	✗	✓

Performance of Existing Storage Stacks

Applications accessing in-memory data in remote servers (single-core case)

L-app: 4KB, **T-app**: 64–128KB

*Detailed root cause analysis is in the paper



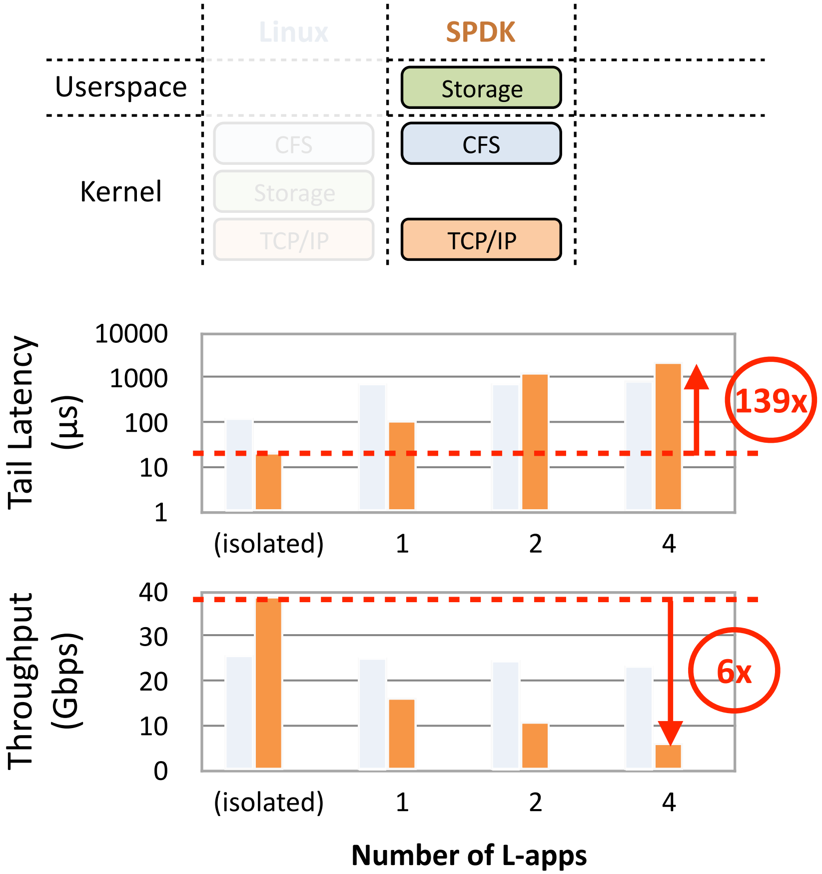
	µs-scale Latency	High Throughput
Linux	✗	✓
SPDK	✗	✗

Performance of Existing Storage Stacks

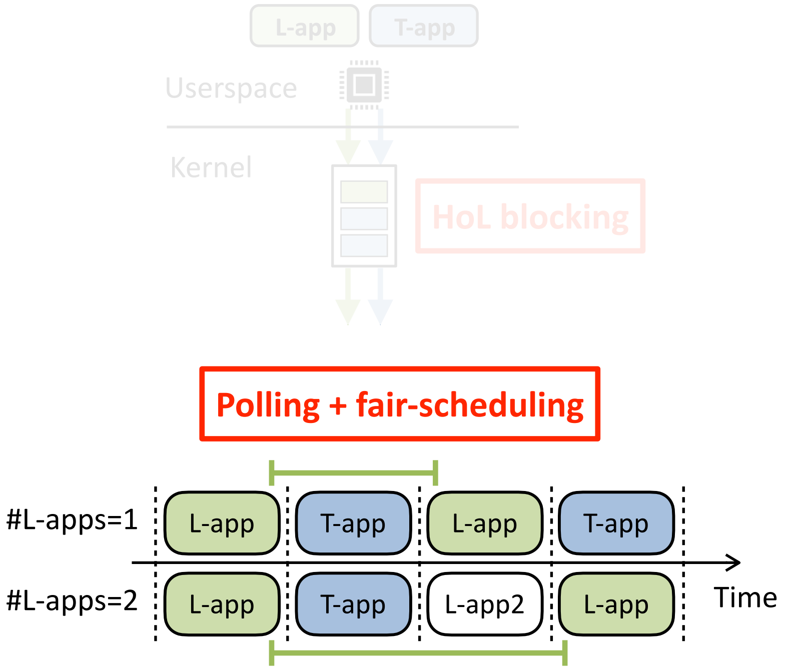
Applications accessing in-memory data in remote servers (single-core case)

L-app: 4KB, **T-app**: 64–128KB

*Detailed root cause analysis is in the paper



	μs-scale Latency	High Throughput
Linux	✗	✓
SPDK	✗	✗

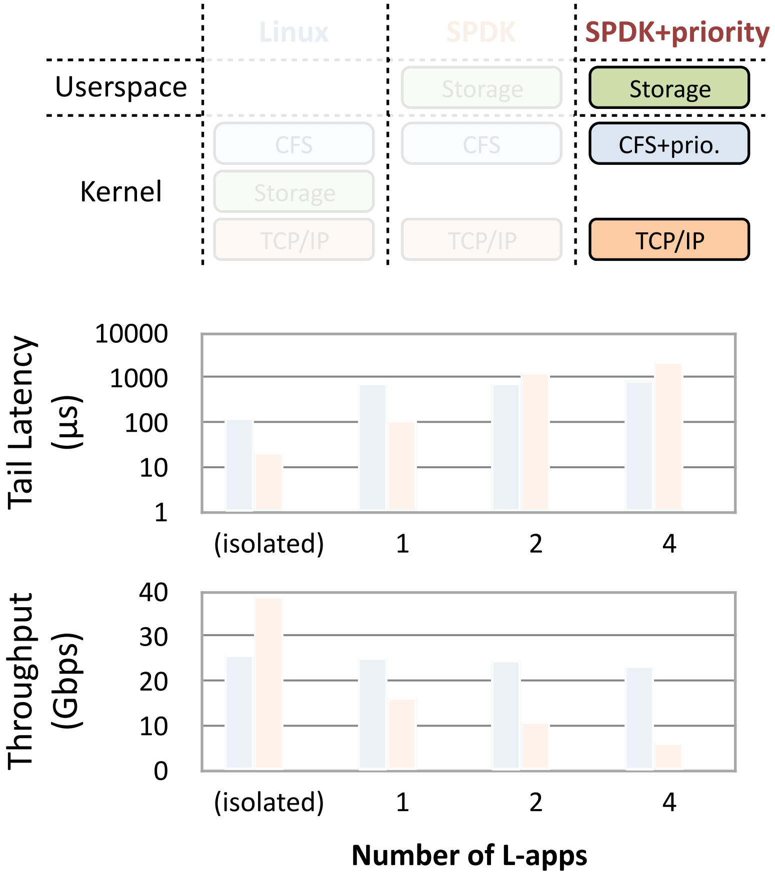


Performance of Existing Storage Stacks

Applications accessing in-memory data in remote servers (single-core case)

L-app: 4KB, **T-app**: 64–128KB

*Detailed root cause analysis is in the paper



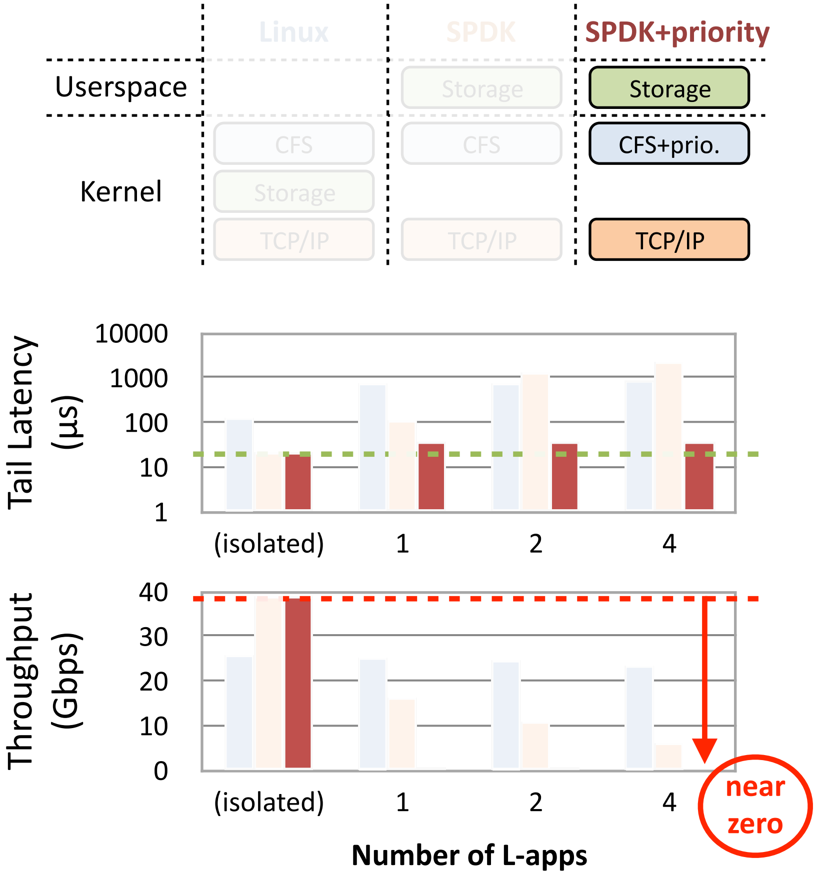
	μs-scale Latency	High Throughput
Linux	✗	✓
SPDK	✗	✗

Performance of Existing Storage Stacks

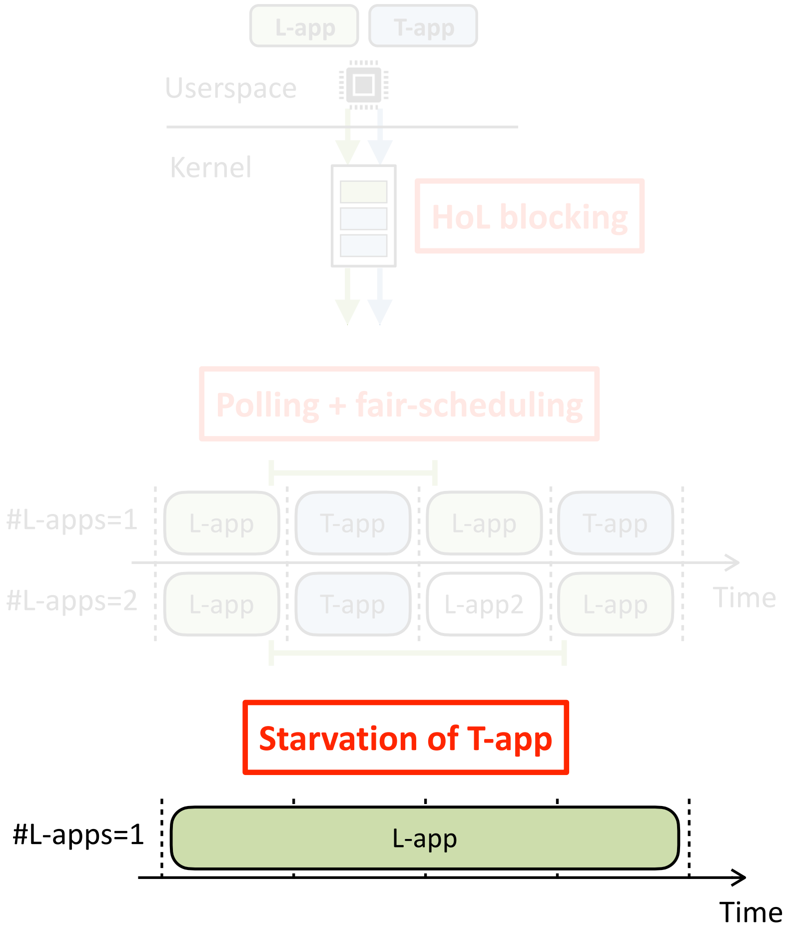
Applications accessing in-memory data in remote servers (single-core case)

L-app: 4KB, **T-app**: 64–128KB

*Detailed root cause analysis is in the paper



	µs-scale Latency	High Throughput
Linux	✗	✓
SPDK	✗	✗
SPDK+ priority	✓	✗

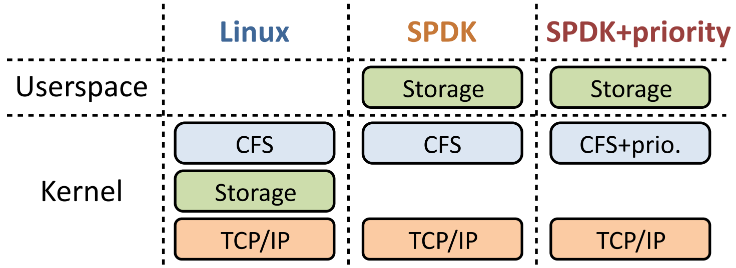


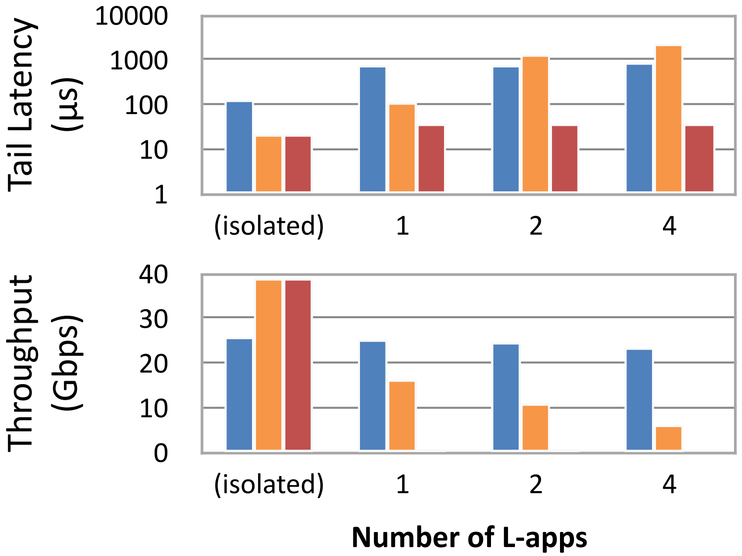
Performance of Existing Storage Stacks

Applications accessing in-memory data in remote servers (single-core case)

L-app: 4KB, **T-app**: 64–128KB

*Detailed root cause analysis is in the paper





	µs-scale Latency	High Throughput
Linux	✗	✓
SPDK	✗	✗
SPDK+ priority	✓	✗

Low latency or high throughput, but not both

blk-switch Summary

blk-switch Summary

- **Linux can achieve *μ*s latency while achieving near-H/W capacity throughput!**
  - Without changes in applications, kernel CPU scheduler, kernel TCP/IP stack, and network hardware

blk-switch Summary

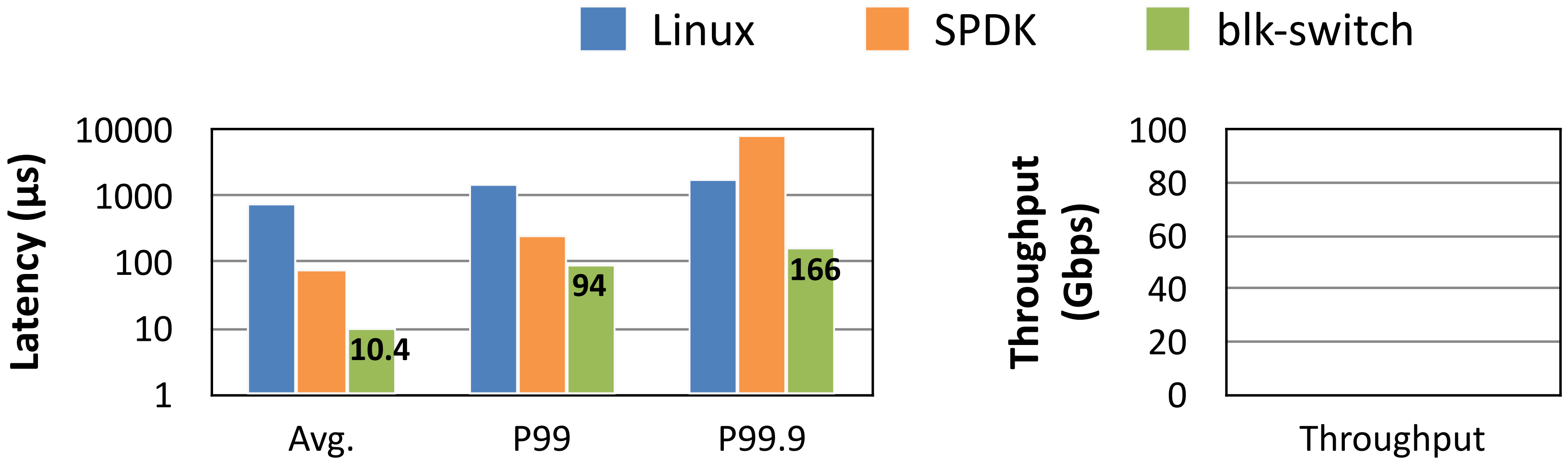
- **Linux can achieve μs latency while achieving near-H/W capacity throughput!**
  - Without changes in applications, kernel CPU scheduler, kernel TCP/IP stack, and network hardware

- For example, **blk-switch** achieves:
  - Even with tens of applications (6 L-apps + 6 T-apps on 6 cores)
  - Complex interference at compute, storage, and network stacks (remote storage access over 100Gbps)

blk-switch Summary

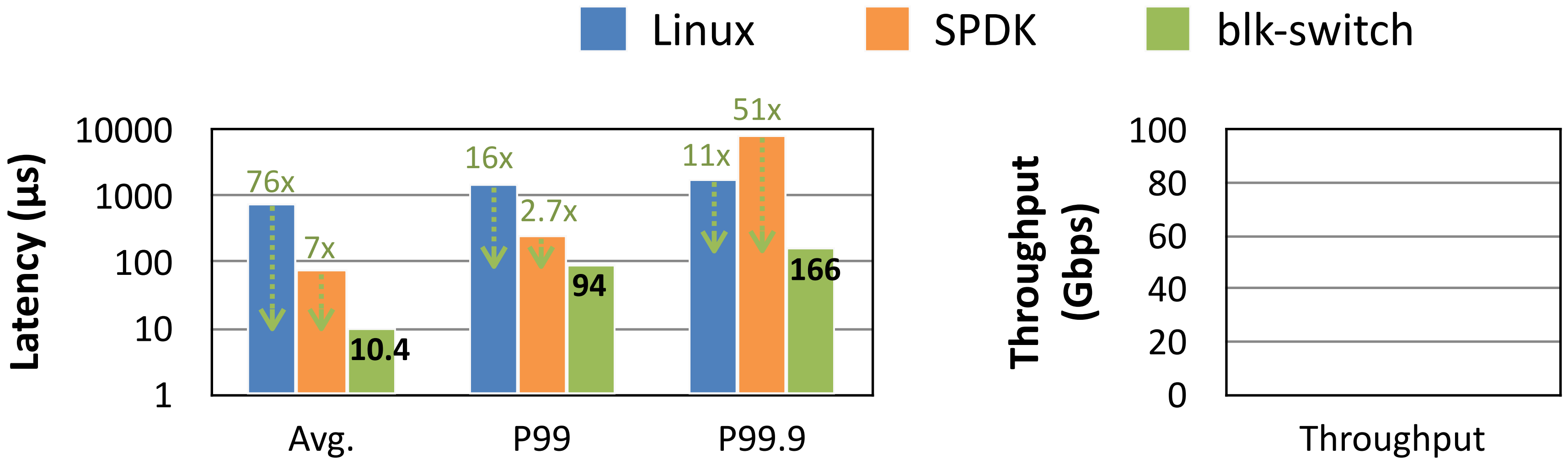
- **Linux can achieve *μs latency* while achieving near-H/W capacity throughput!**
  - Without changes in applications, kernel CPU scheduler, kernel TCP/IP stack, and network hardware

- For example, **blk-switch** achieves:
  - Even with tens of applications (6 L-apps + 6 T-apps on 6 cores)
  - Complex interference at compute, storage, and network stacks (remote storage access over 100Gbps)



blk-switch Summary

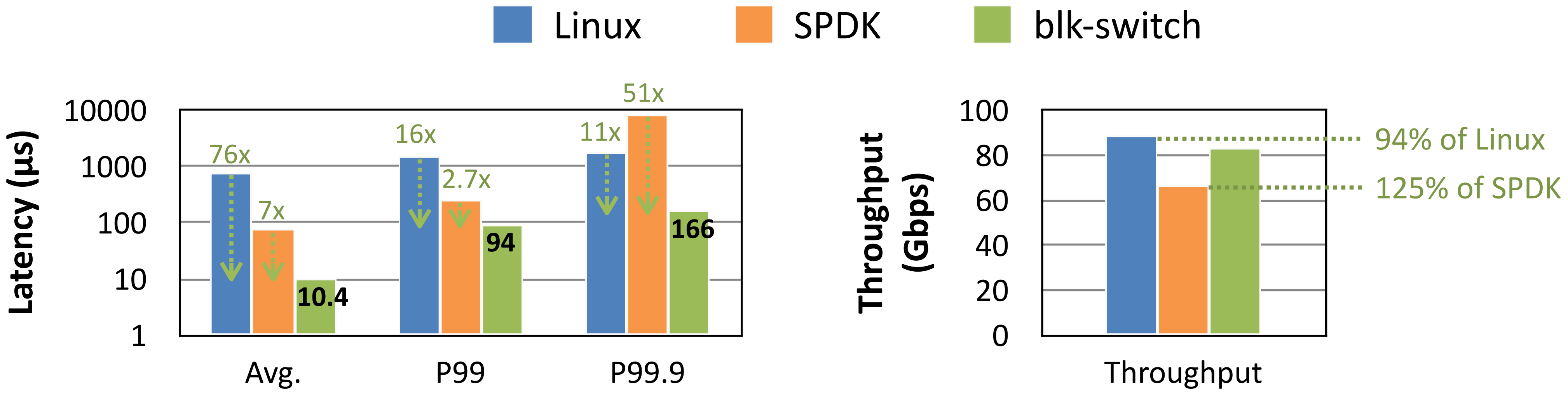
- **Linux can achieve μs latency while achieving near-H/W capacity throughput!**
  - Without changes in applications, kernel CPU scheduler, kernel TCP/IP stack, and network hardware
- For example, **blk-switch** achieves:
  - Even with tens of applications (6 L-apps + 6 T-apps on 6 cores)
  - Complex interference at compute, storage, and network stacks (remote storage access over 100Gbps)



blk-switch Summary

- **Linux can achieve μs latency while achieving near-H/W capacity throughput!**
  - Without changes in applications, kernel CPU scheduler, kernel TCP/IP stack, and network hardware

- For example, **blk-switch** achieves:
  - Even with tens of applications (6 L-apps + 6 T-apps on 6 cores)
  - Complex interference at compute, storage, and network stacks (remote storage access over 100Gbps)



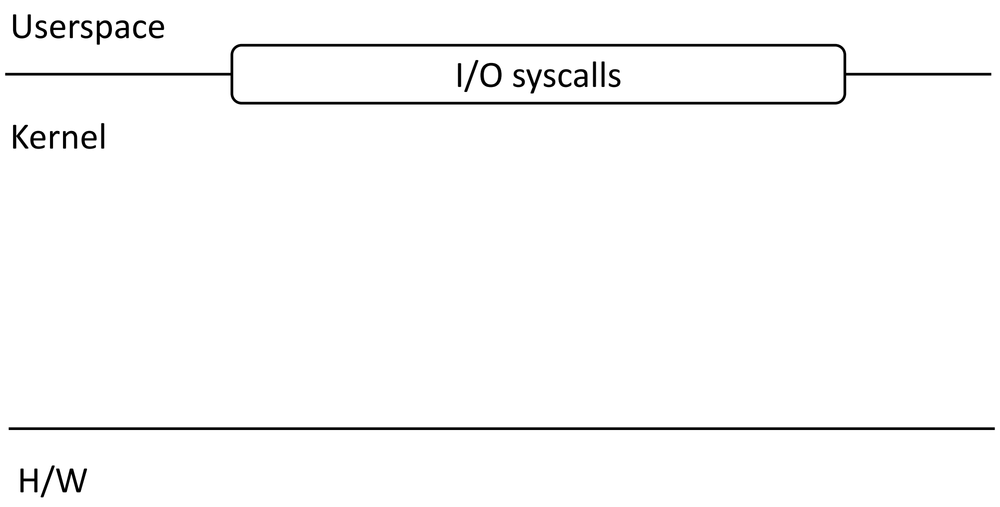
blk-switch Key Insight

blk-switch Key Insight

- **Observation:** Today's Linux **storage stack** is conceptually similar to **network switches!**

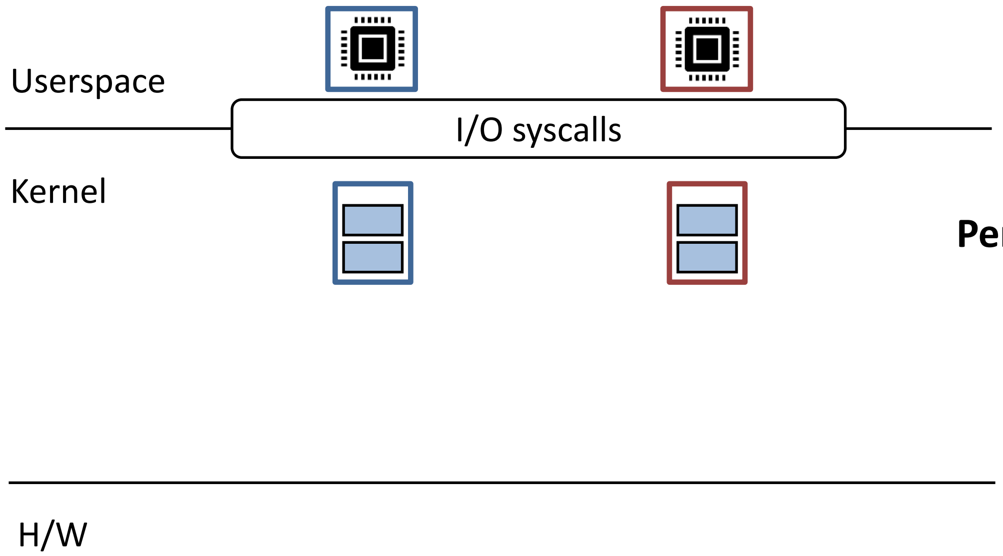
blk-switch Key Insight

- **Observation:** Today's Linux **storage stack** is conceptually similar to **network switches!**



blk-switch Key Insight

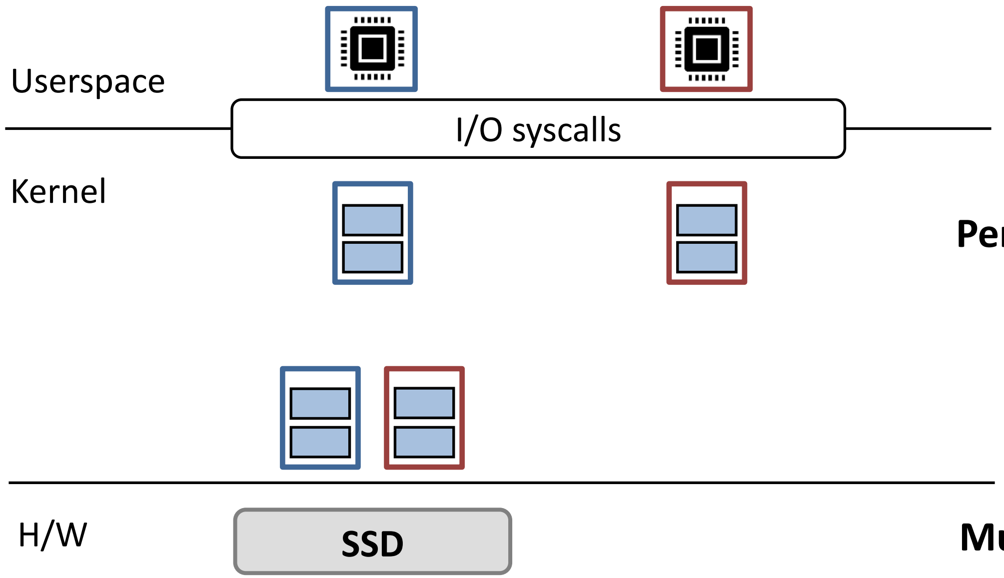
- **Observation:** Today's Linux **storage stack** is conceptually similar to **network switches!**



Per-core block layer queues

blk-switch Key Insight

- **Observation:** Today's Linux **storage stack** is conceptually similar to **network switches!**

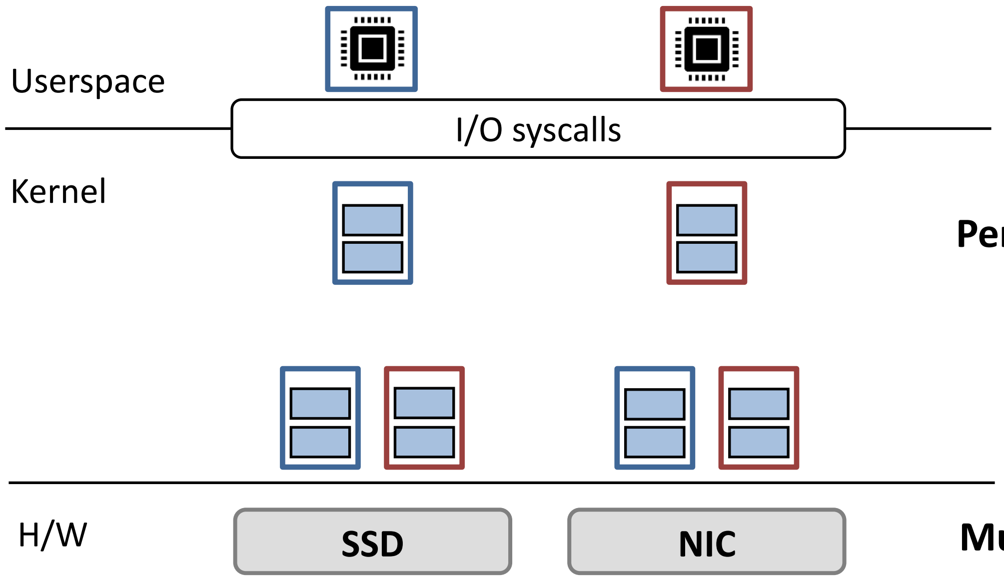


Per-core block layer queues

Multi-queue storage/network hardware

blk-switch Key Insight

- **Observation:** Today's Linux **storage stack** is conceptually similar to **network switches!**

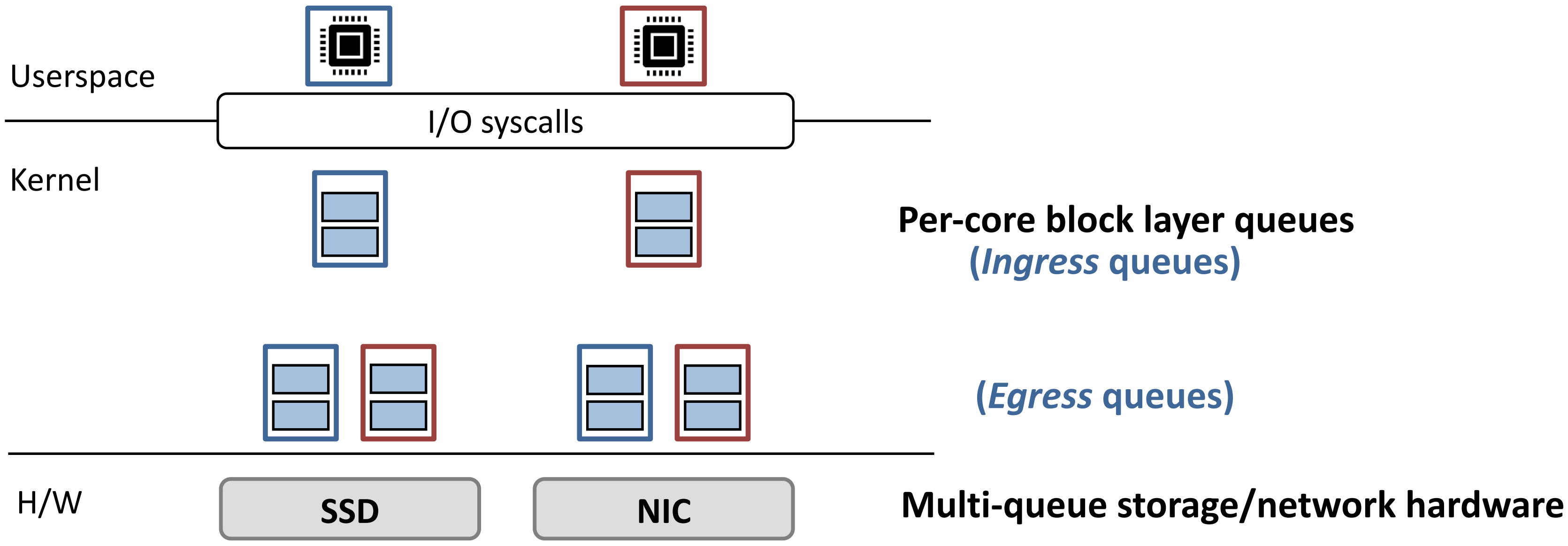


Per-core block layer queues

Multi-queue storage/network hardware

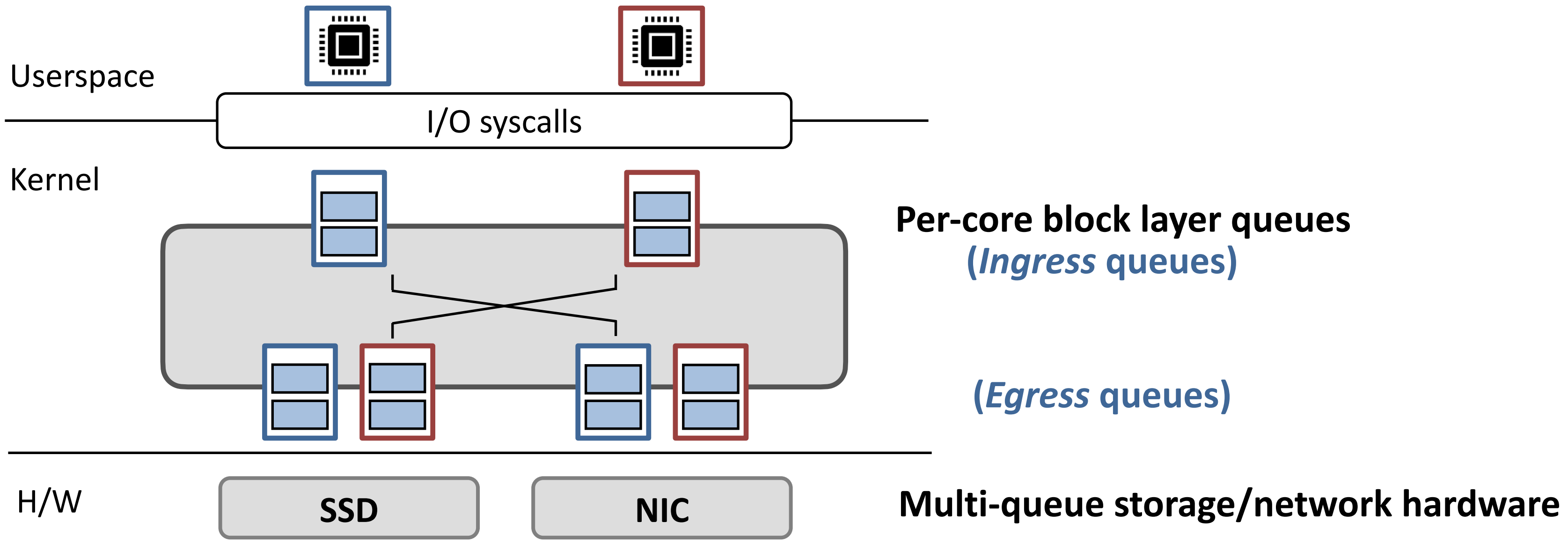
blk-switch Key Insight

- **Observation:** Today's Linux **storage stack** is conceptually similar to **network switches!**



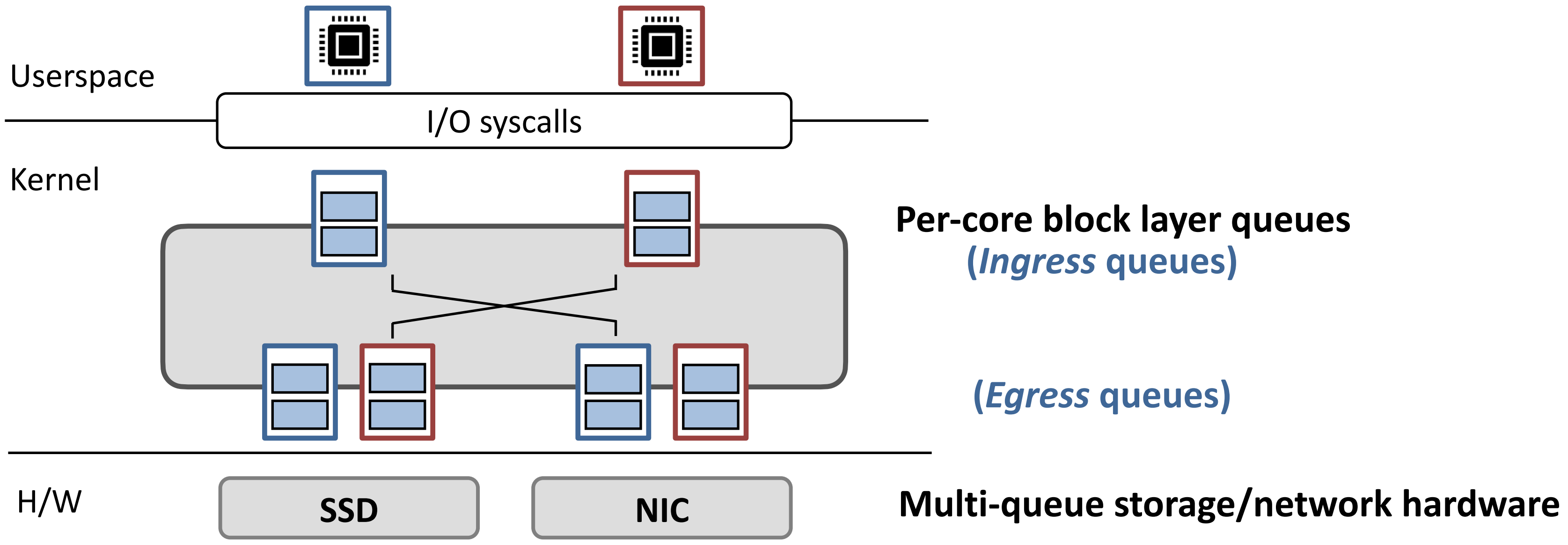
blk-switch Key Insight

- **Observation:** Today's Linux **storage stack** is conceptually similar to **network switches!**



blk-switch Key Insight

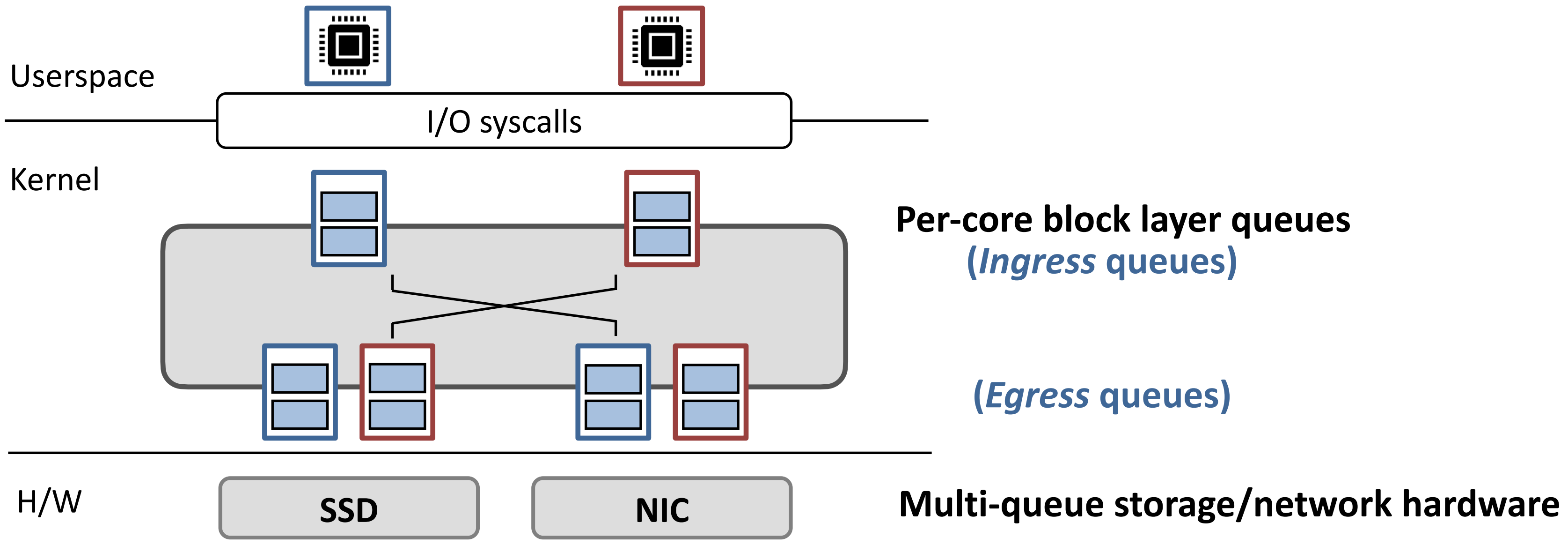
- **Observation:** Today's Linux **storage stack** is conceptually similar to **network switches!**



- **blk-switch**: Switched Linux storage stack architecture

blk-switch Key Insight

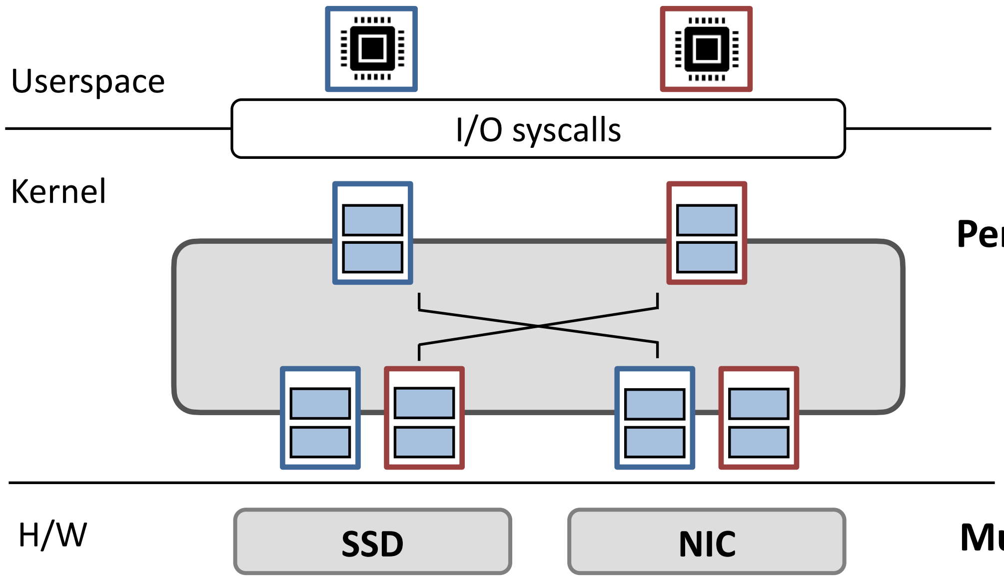
- **Observation:** Today's Linux **storage stack** is conceptually similar to **network switches!**



- **blk-switch**: Switched Linux storage stack architecture
  - Enables decoupling request processing from application cores

blk-switch Key Insight

- **Observation:** Today's Linux **storage stack** is conceptually similar to **network switches!**



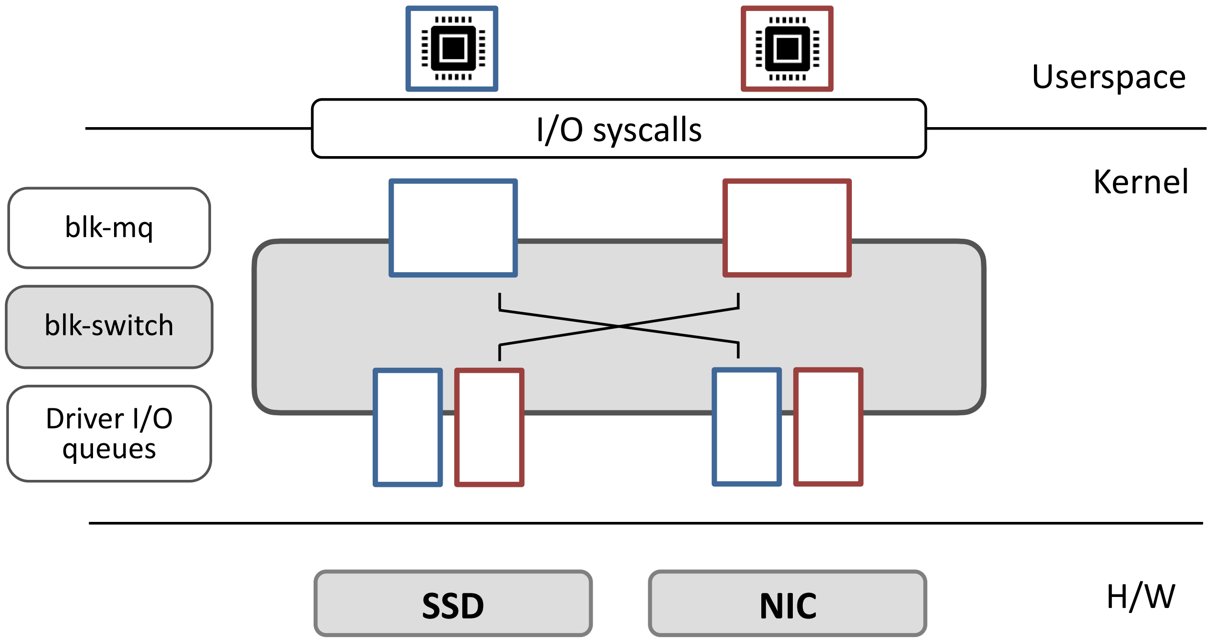
Per-core block layer queues
(*Ingress* queues)

(*Egress* queues)

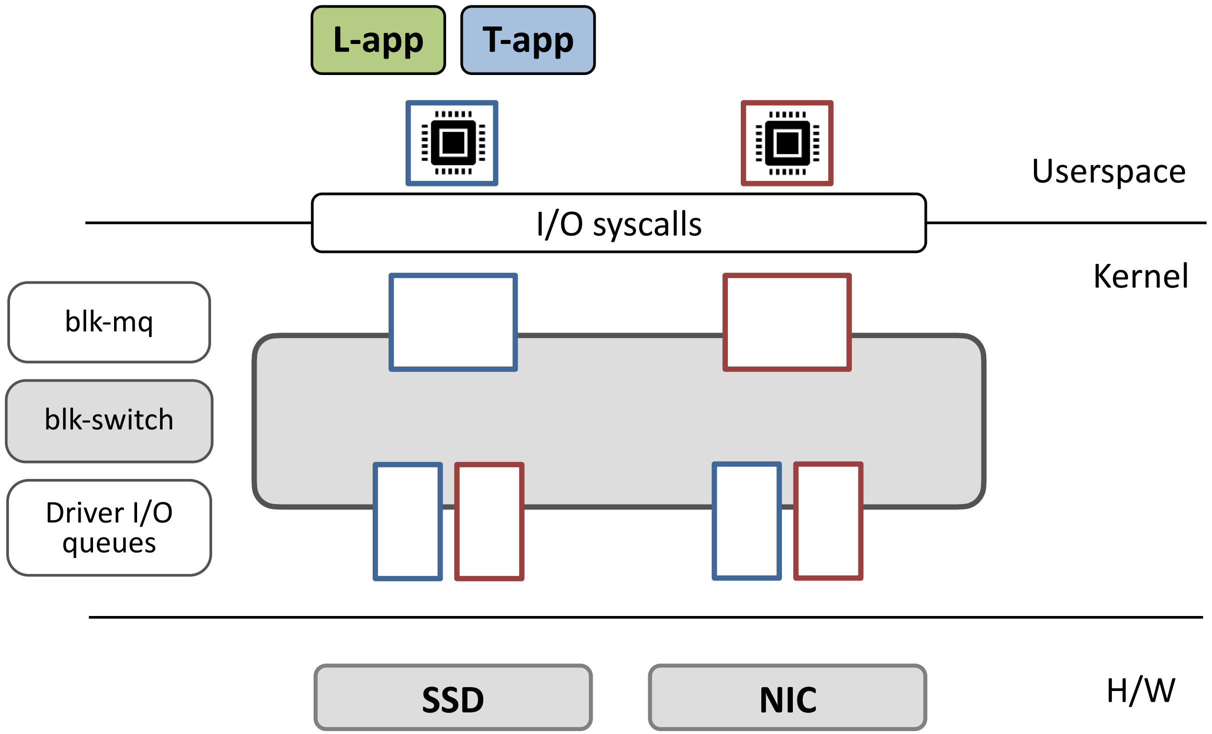
Multi-queue storage/network hardware

- **blk-switch**: Switched Linux storage stack architecture
  - Enables decoupling request processing from application cores
  - Multi-egress queues, prioritization, and load balancing

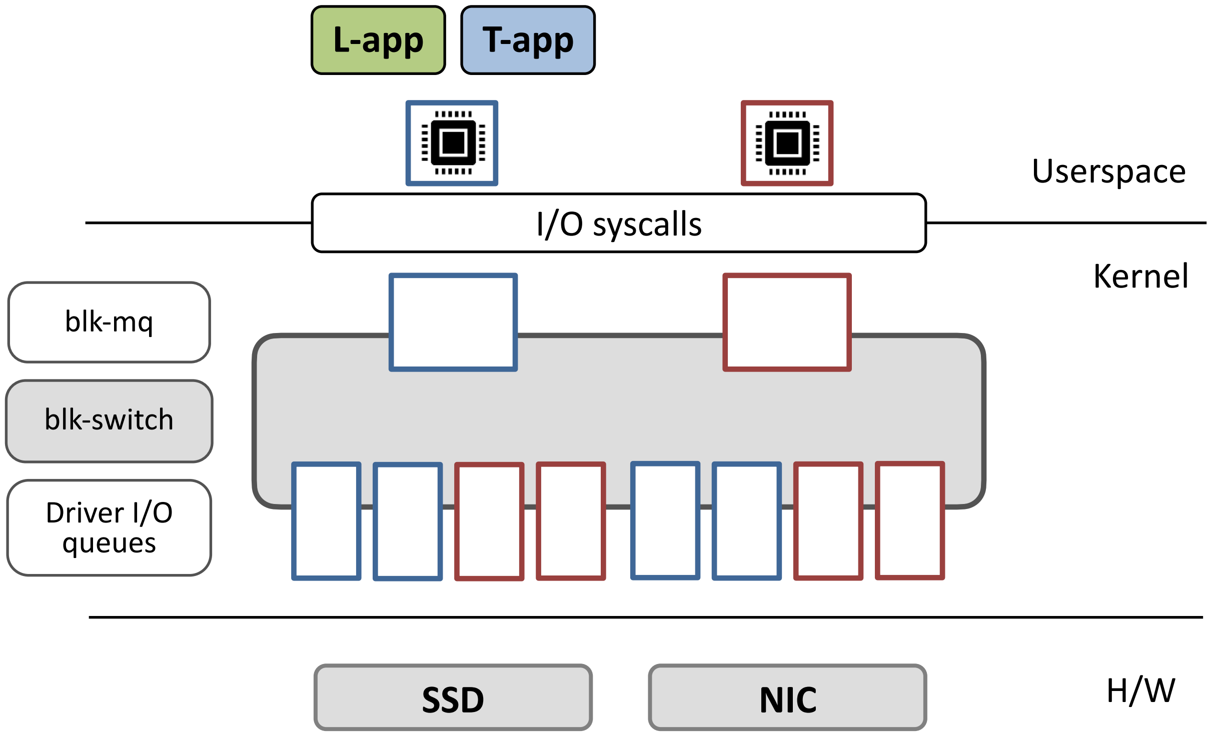
A deeper dive into blk-switch architecture



A deeper dive into blk-switch architecture

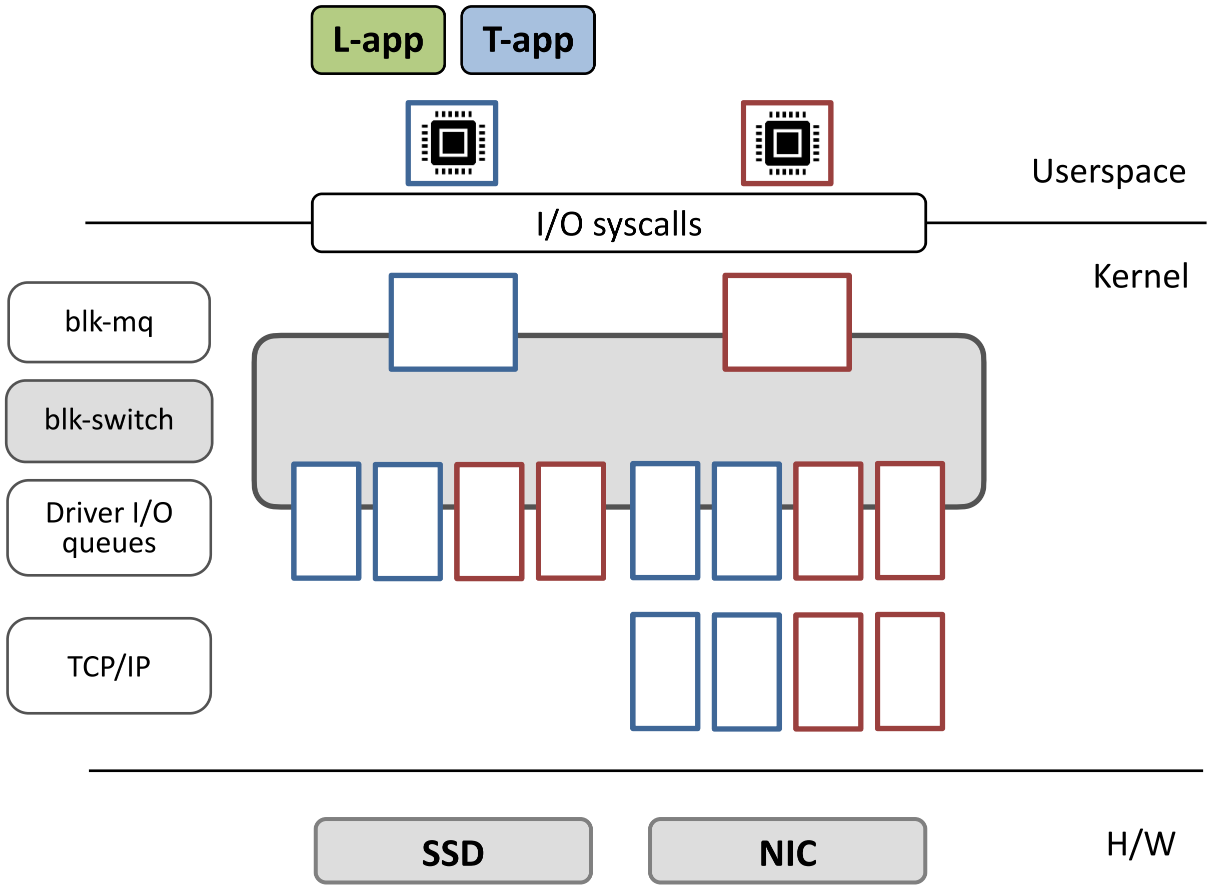


A deeper dive into blk-switch architecture



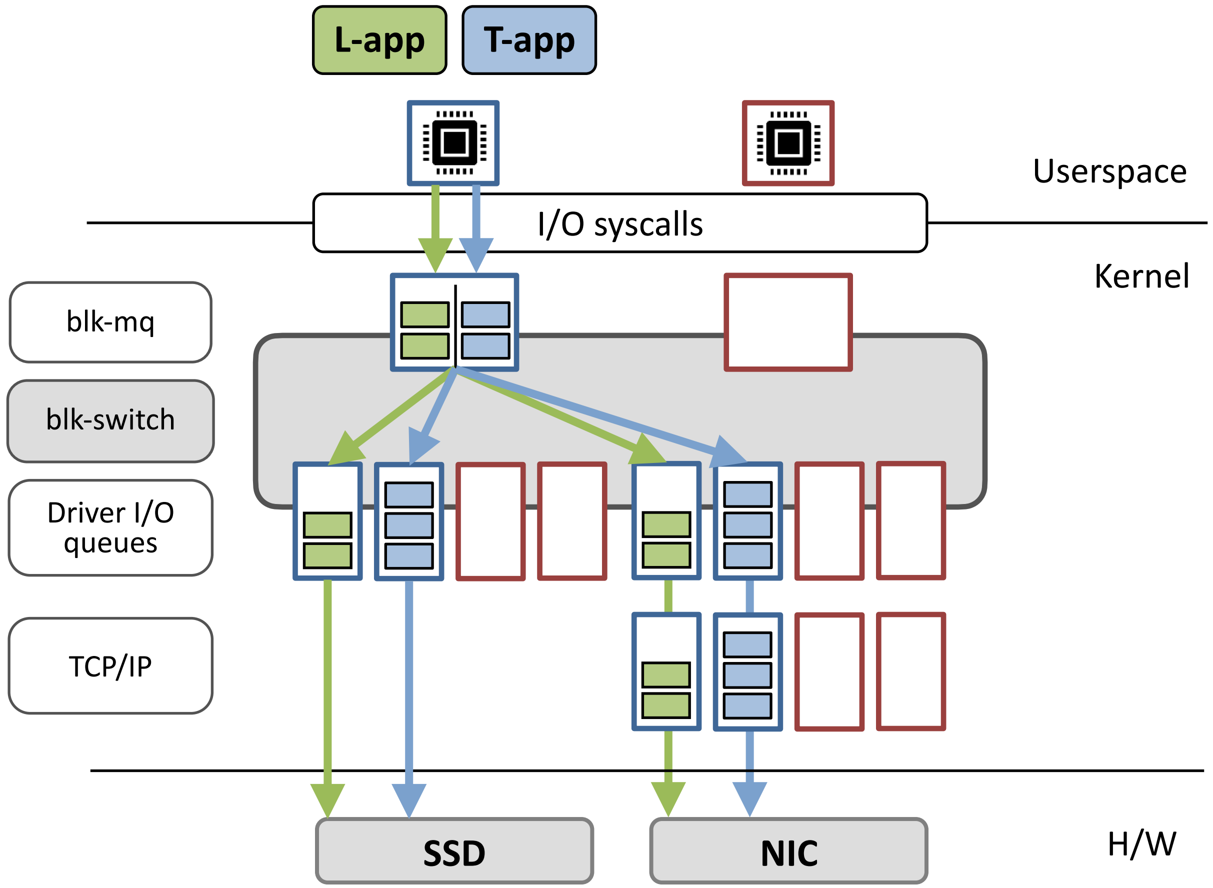
1. *Egress* queue per-(core, app-class)

A deeper dive into blk-switch architecture



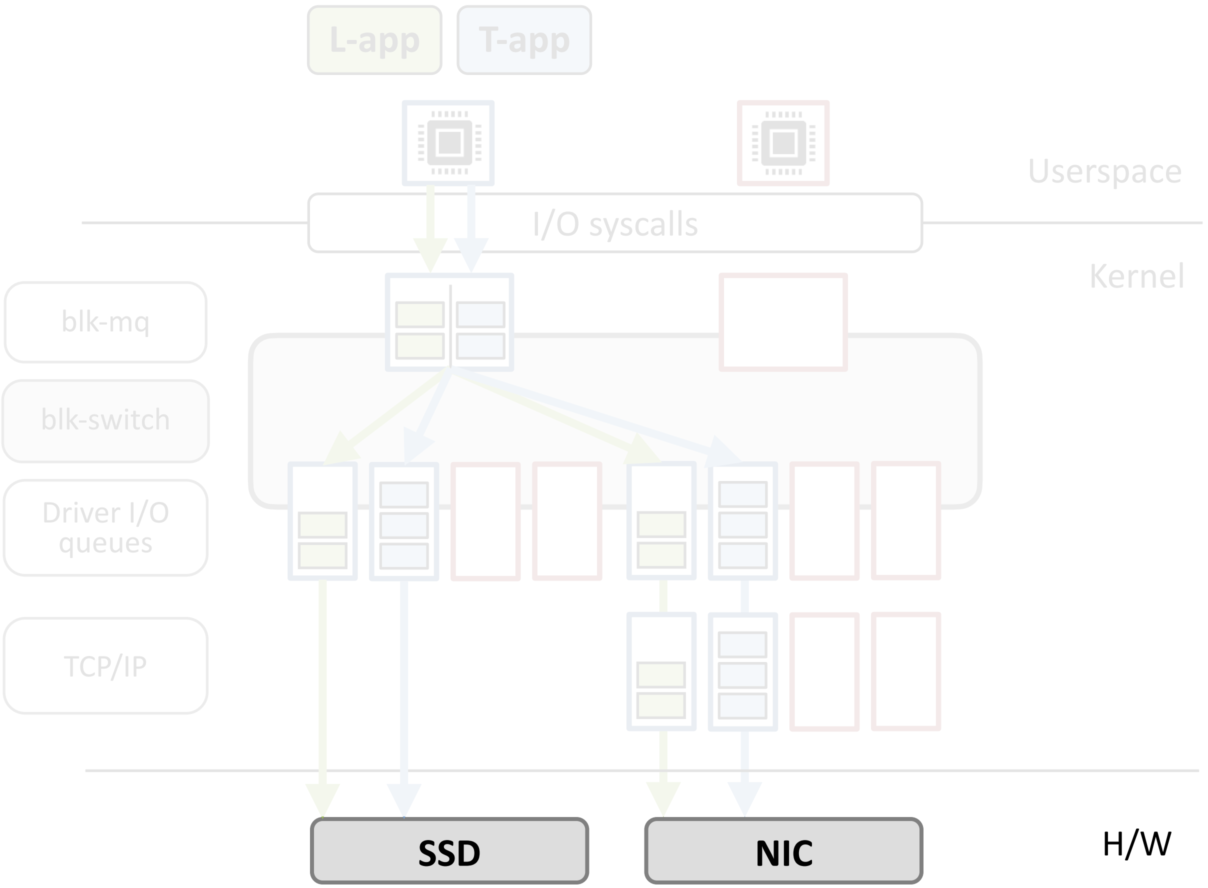
1. *Egress* queue per-(core, app-class)

A deeper dive into blk-switch architecture



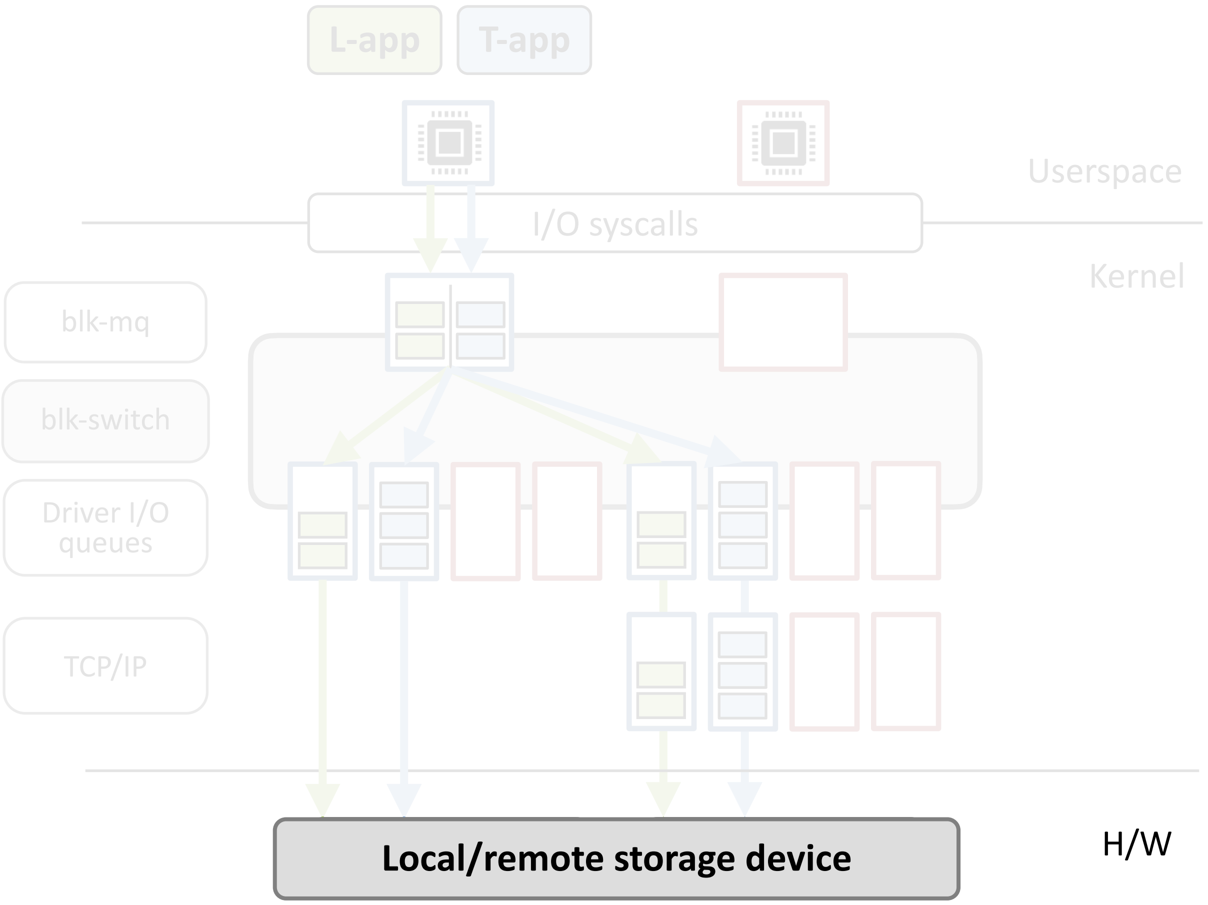
1. *Egress* queue per-(core, app-class)

A deeper dive into blk-switch architecture



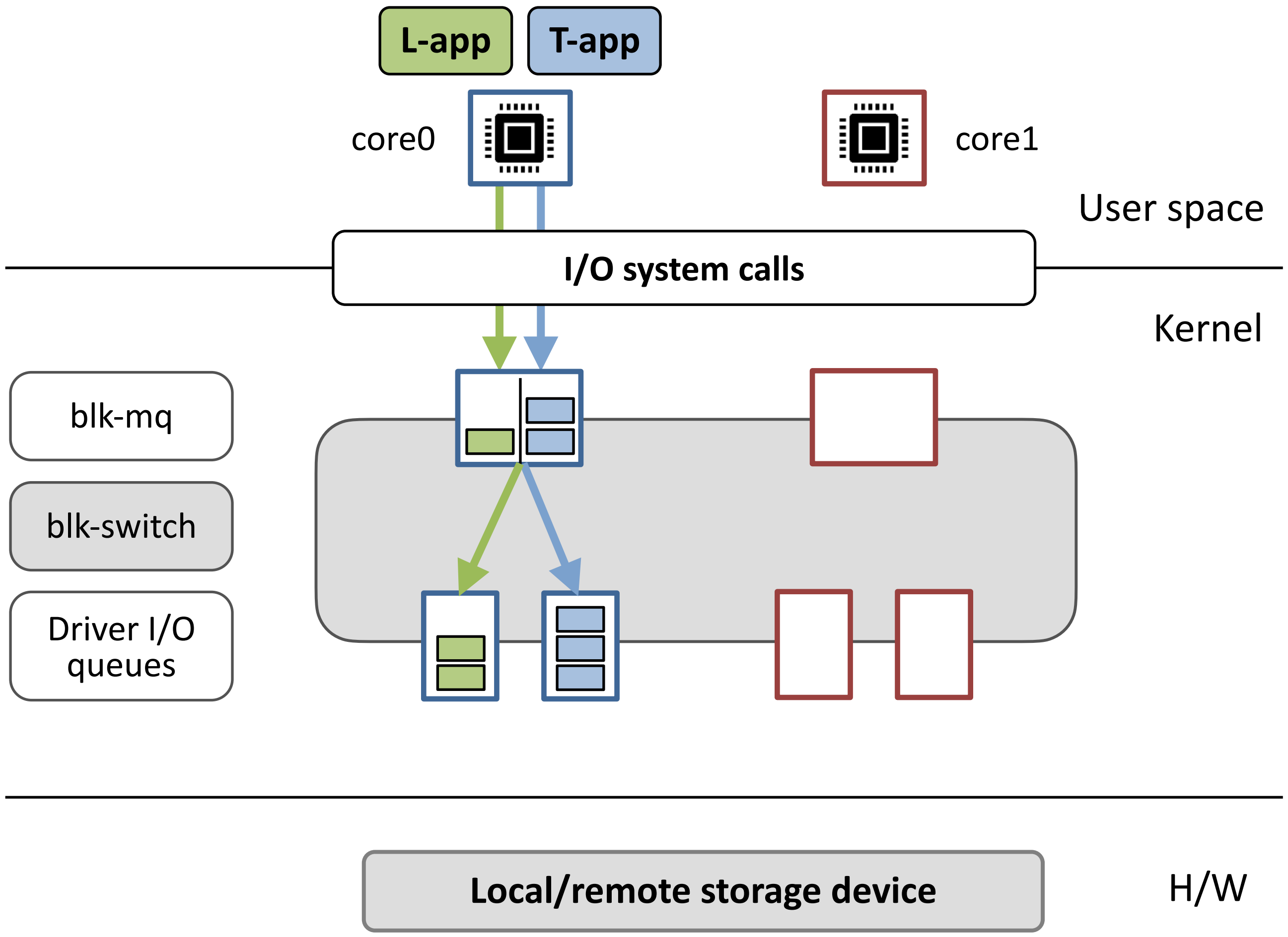
1. *Egress* queue per-(core, app-class)

A deeper dive into blk-switch architecture



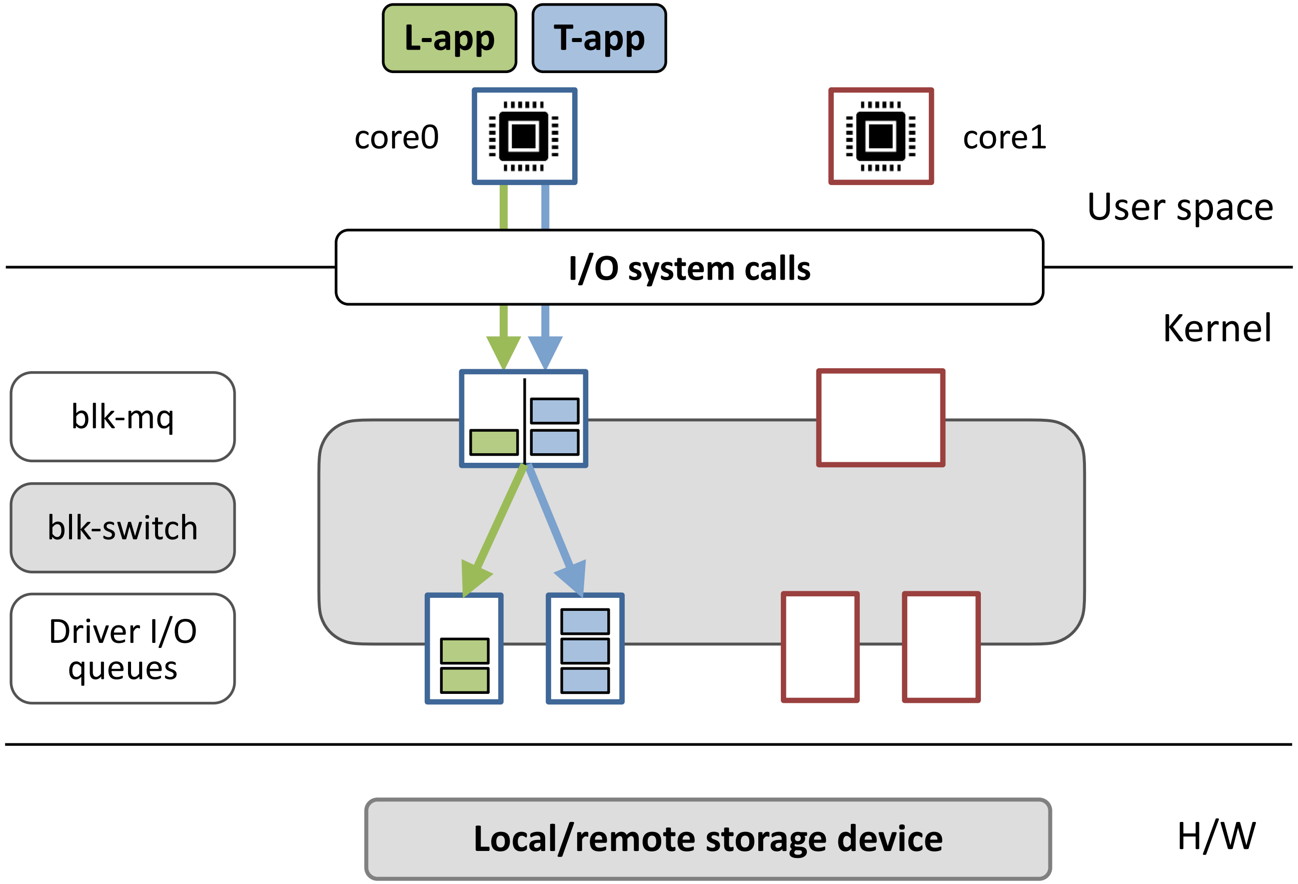
1. *Egress* queue per-(core, app-class)

A deeper dive into blk-switch architecture



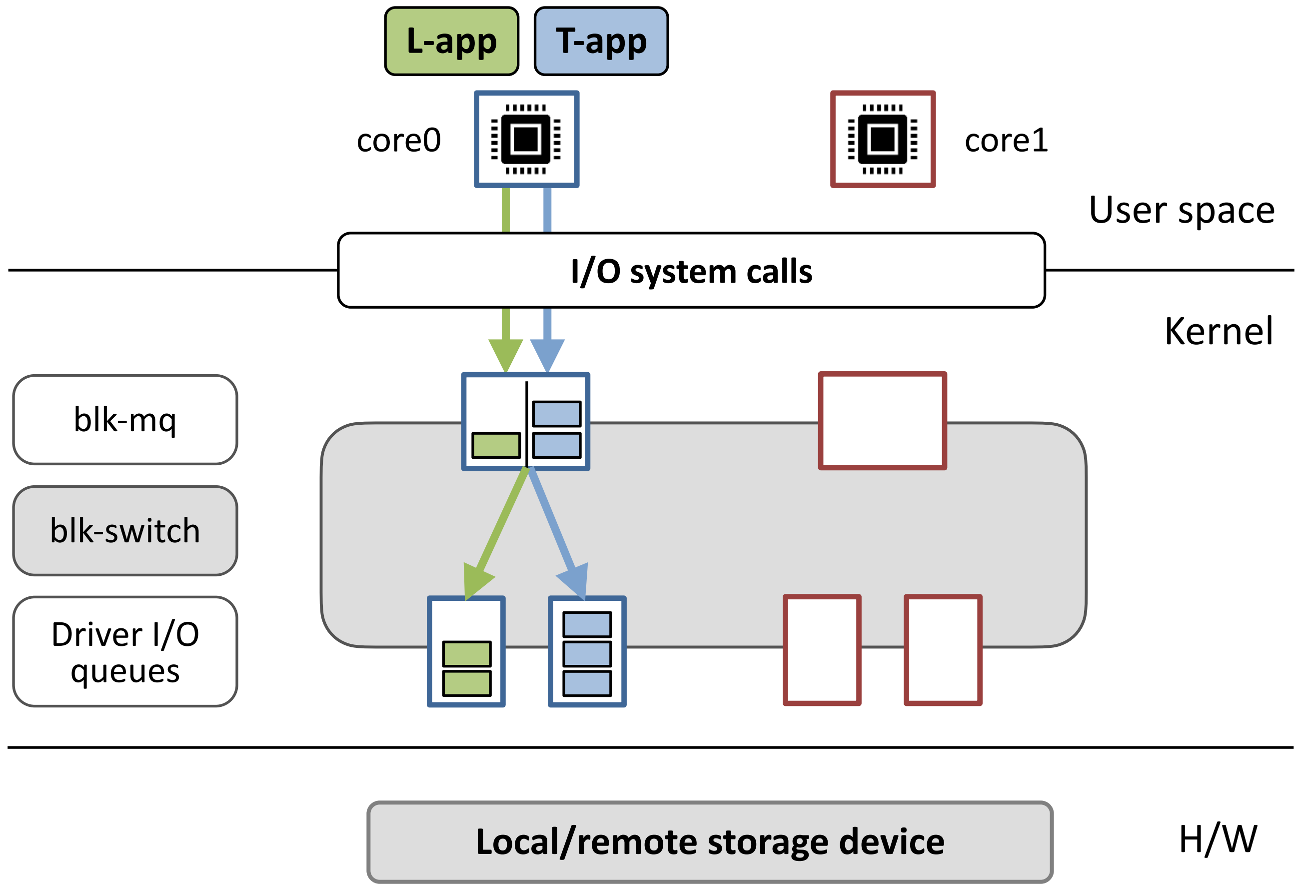
1. *Egress* queue per-(core, app-class)

A deeper dive into blk-switch architecture



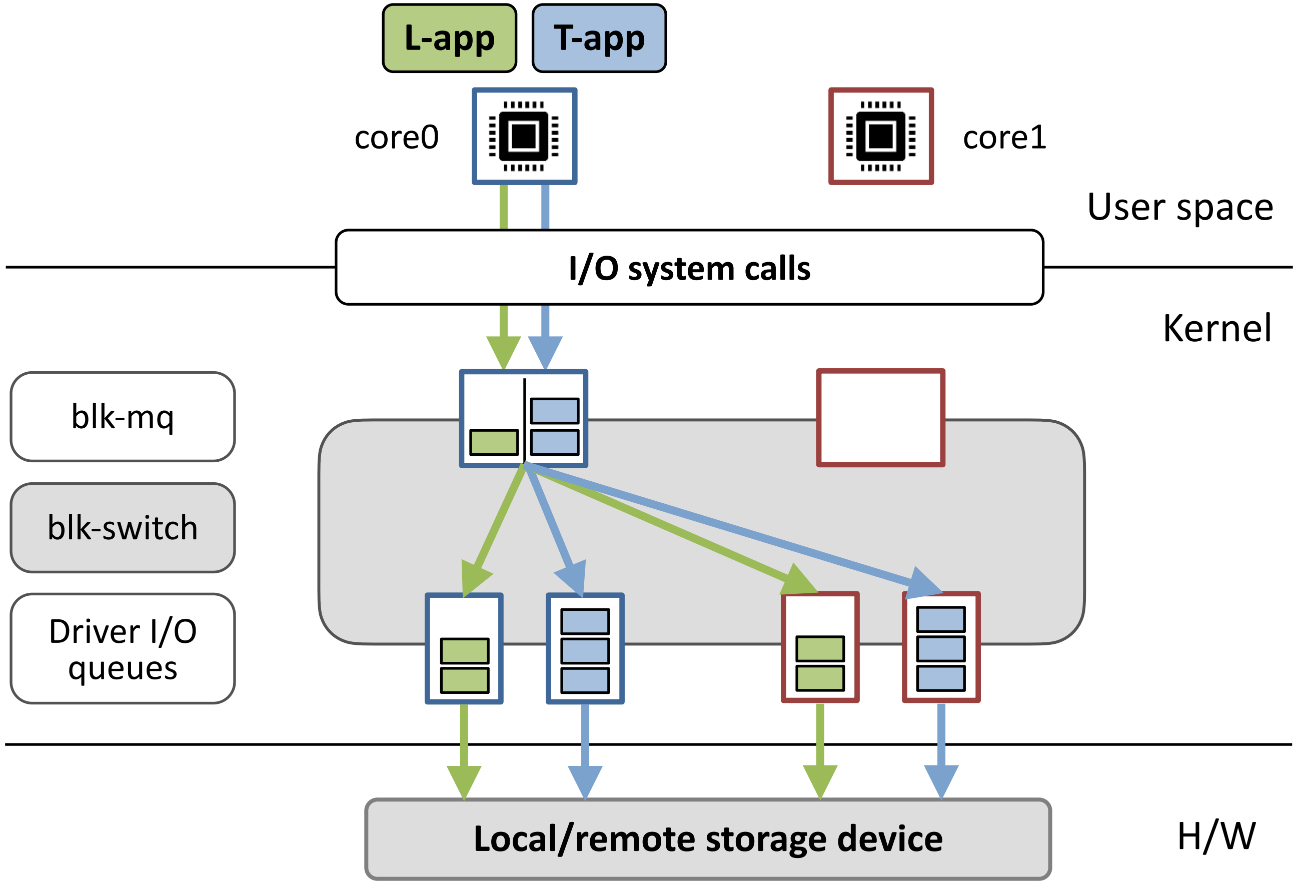
1. *Egress* queue per-(core, app-class)

A deeper dive into blk-switch architecture



1. *Egress* queue per-(core, app-class)
2. *Flexible* mapping from ingress to egress queues

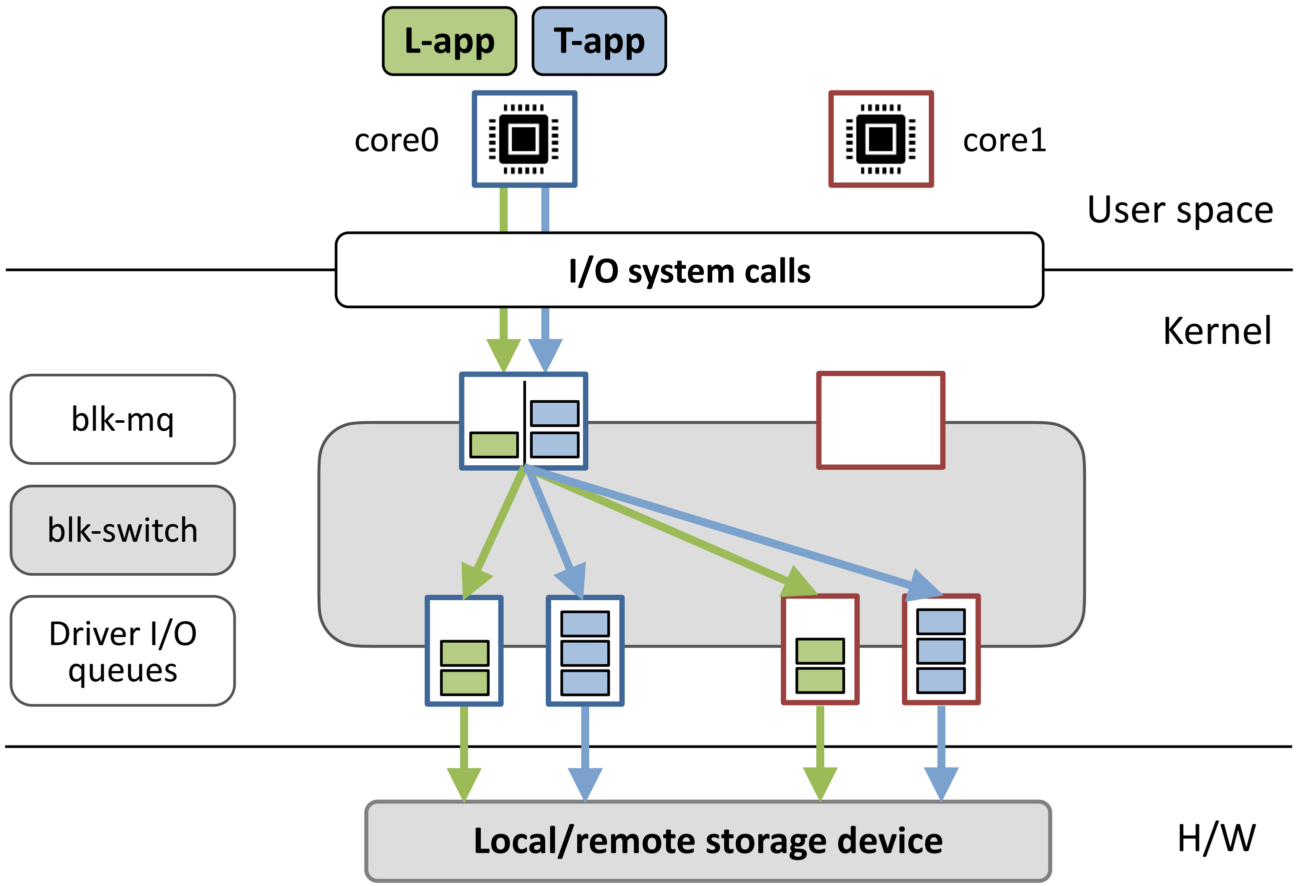
A deeper dive into blk-switch architecture



1. *Egress* queue per-(core, app-class)

2. *Flexible* mapping from ingress to egress queues

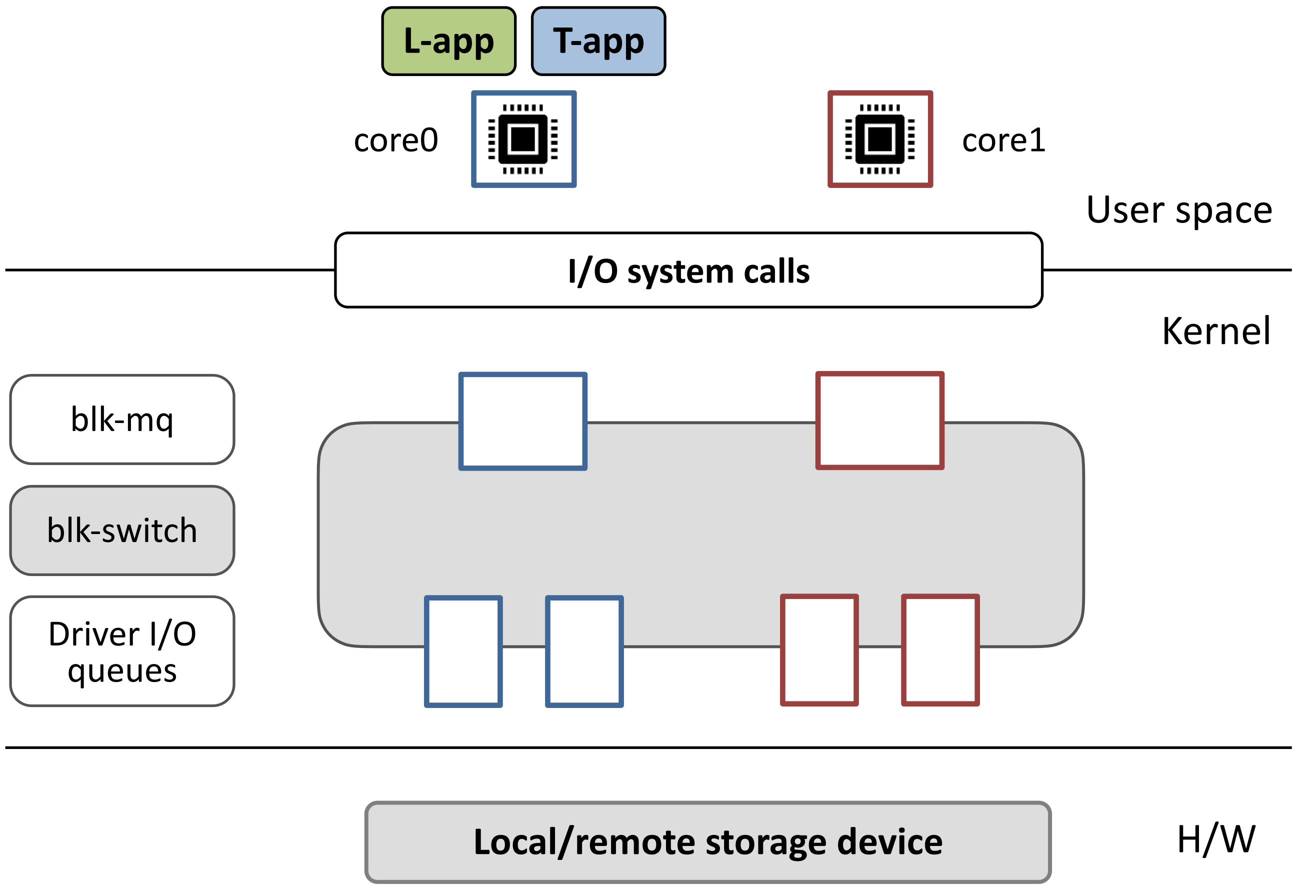
A deeper dive into blk-switch architecture



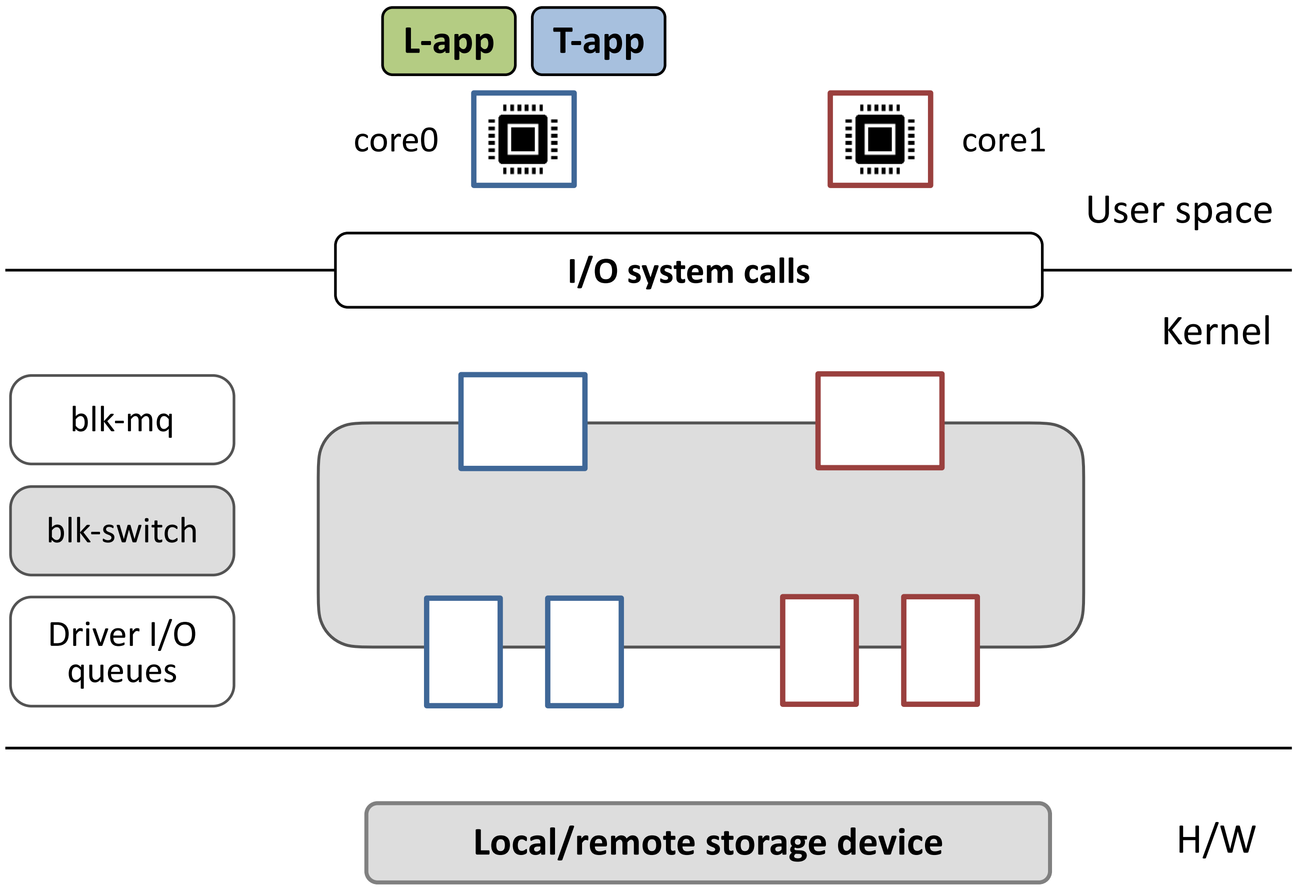
1. *Egress* queue per-(core, app-class)
2. *Flexible* mapping from ingress to egress queues

Decoupling request processing from application cores: “Static ⇢ Flexible”

blk-switch Prioritization

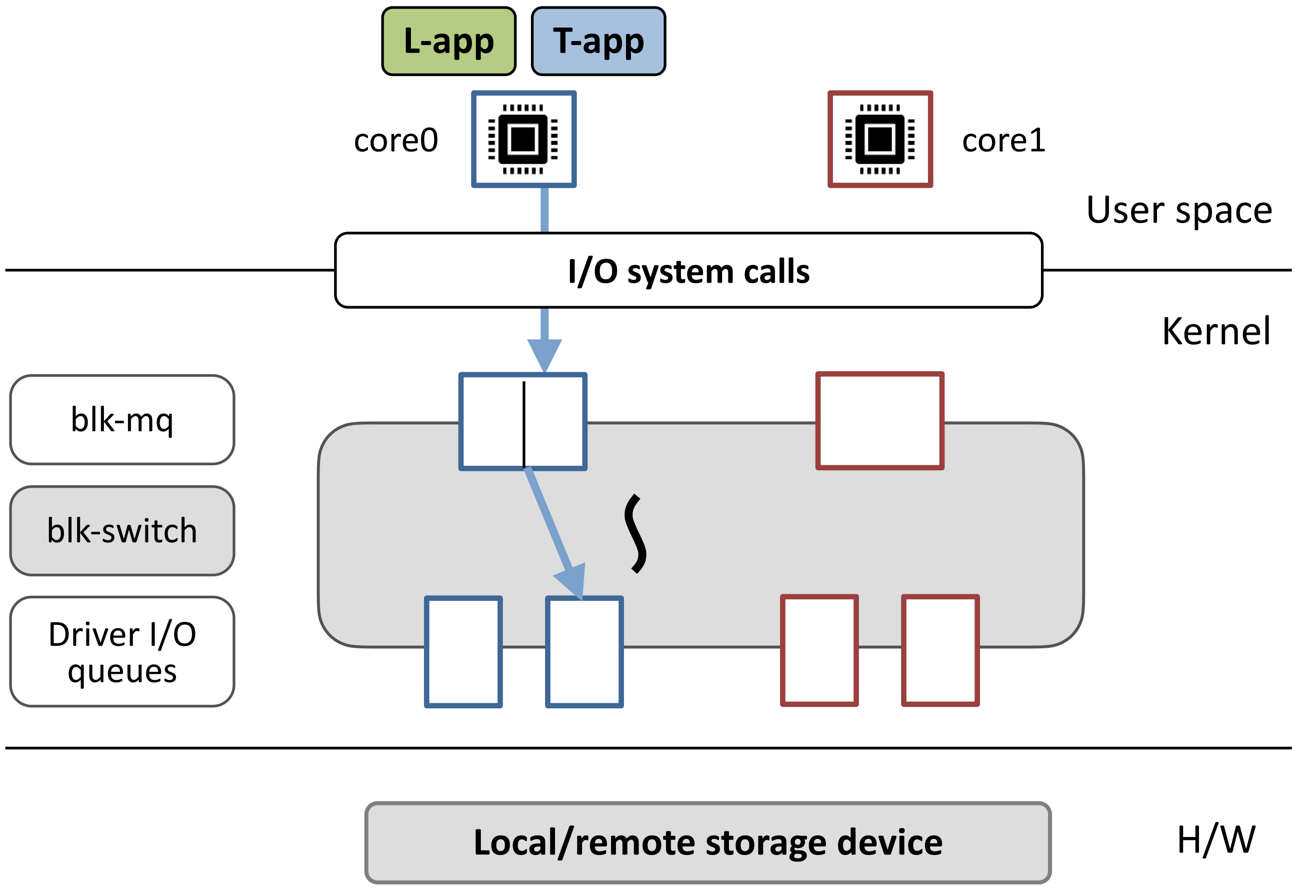


blk-switch Prioritization



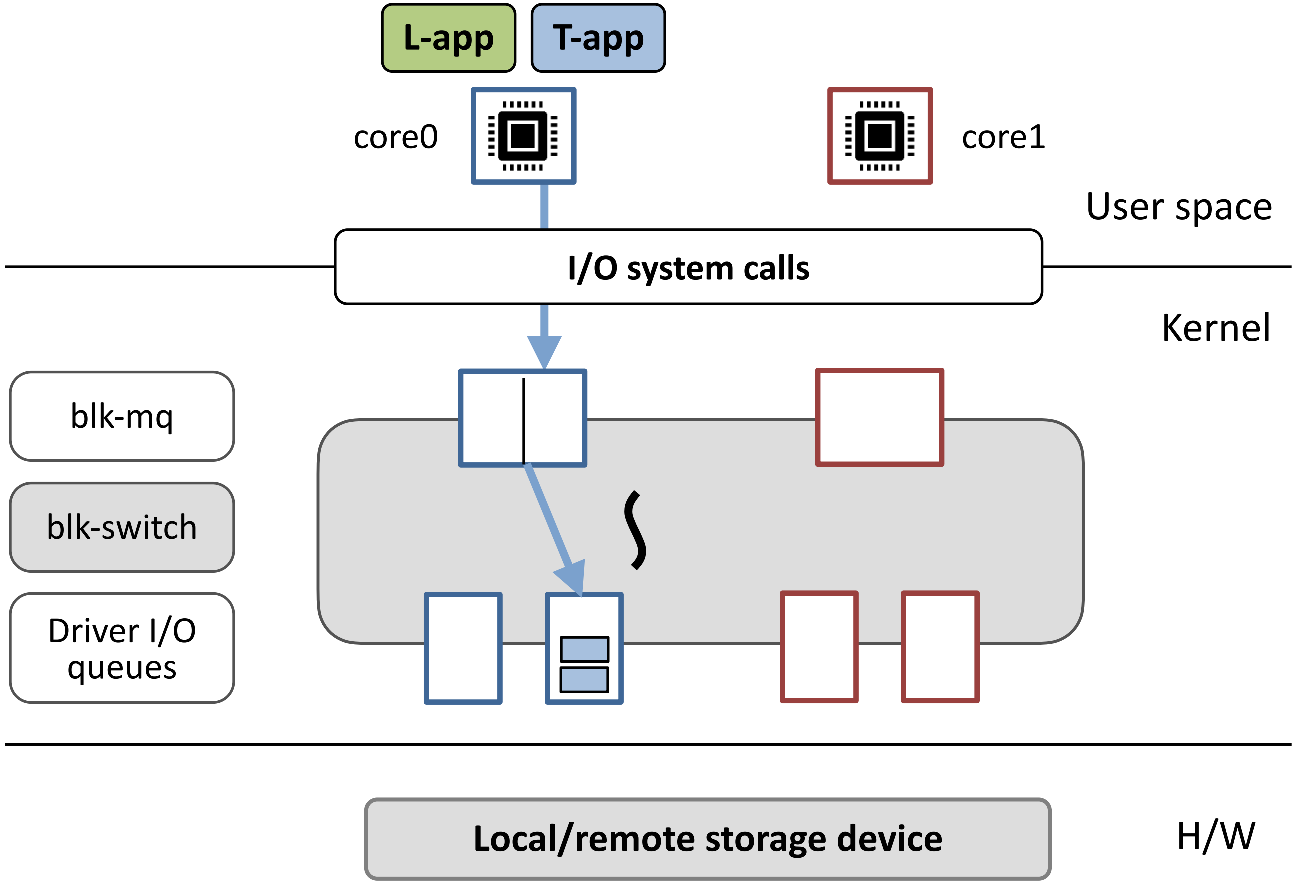
Prioritize **L-app request processing**

blk-switch Prioritization



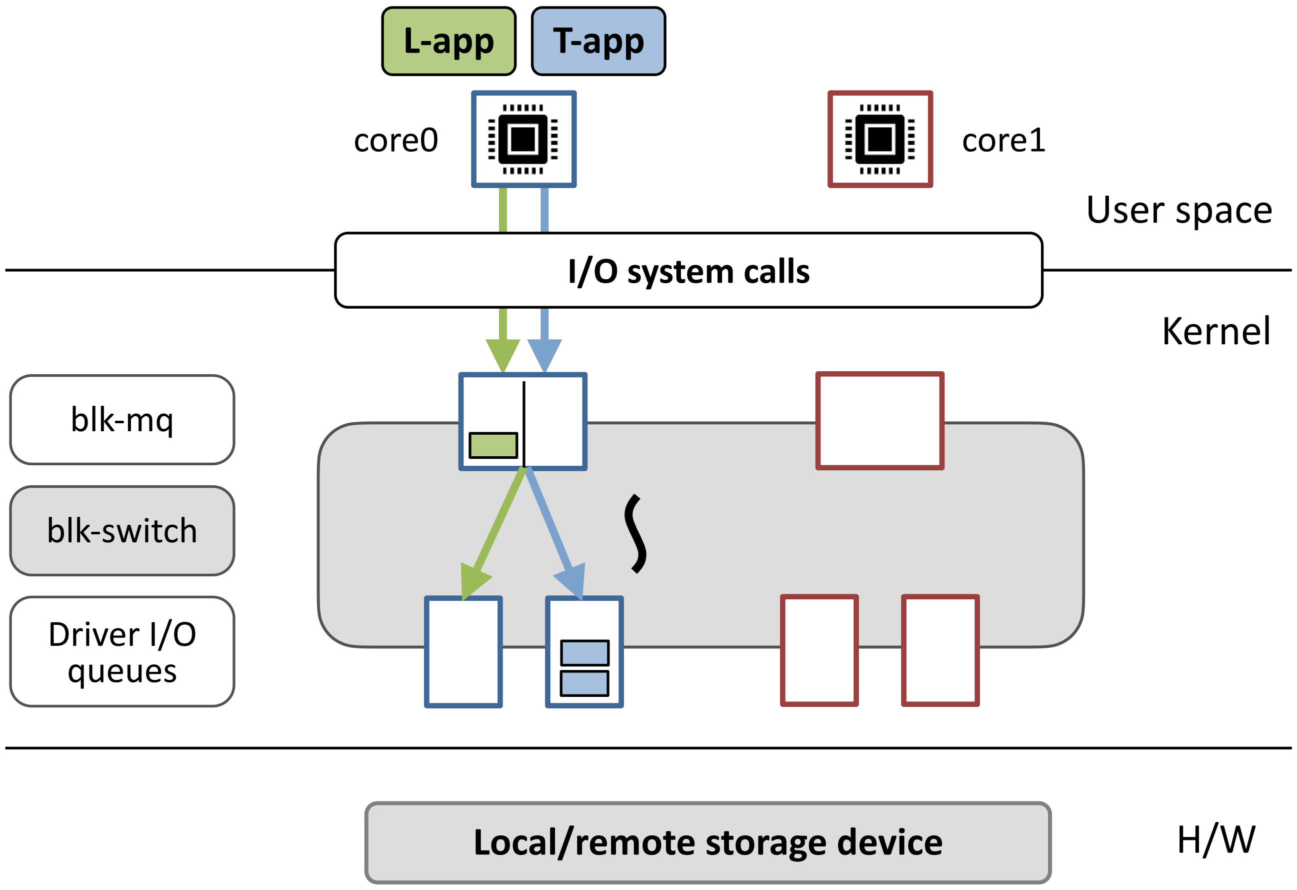
Prioritize **L-app request processing**

blk-switch Prioritization



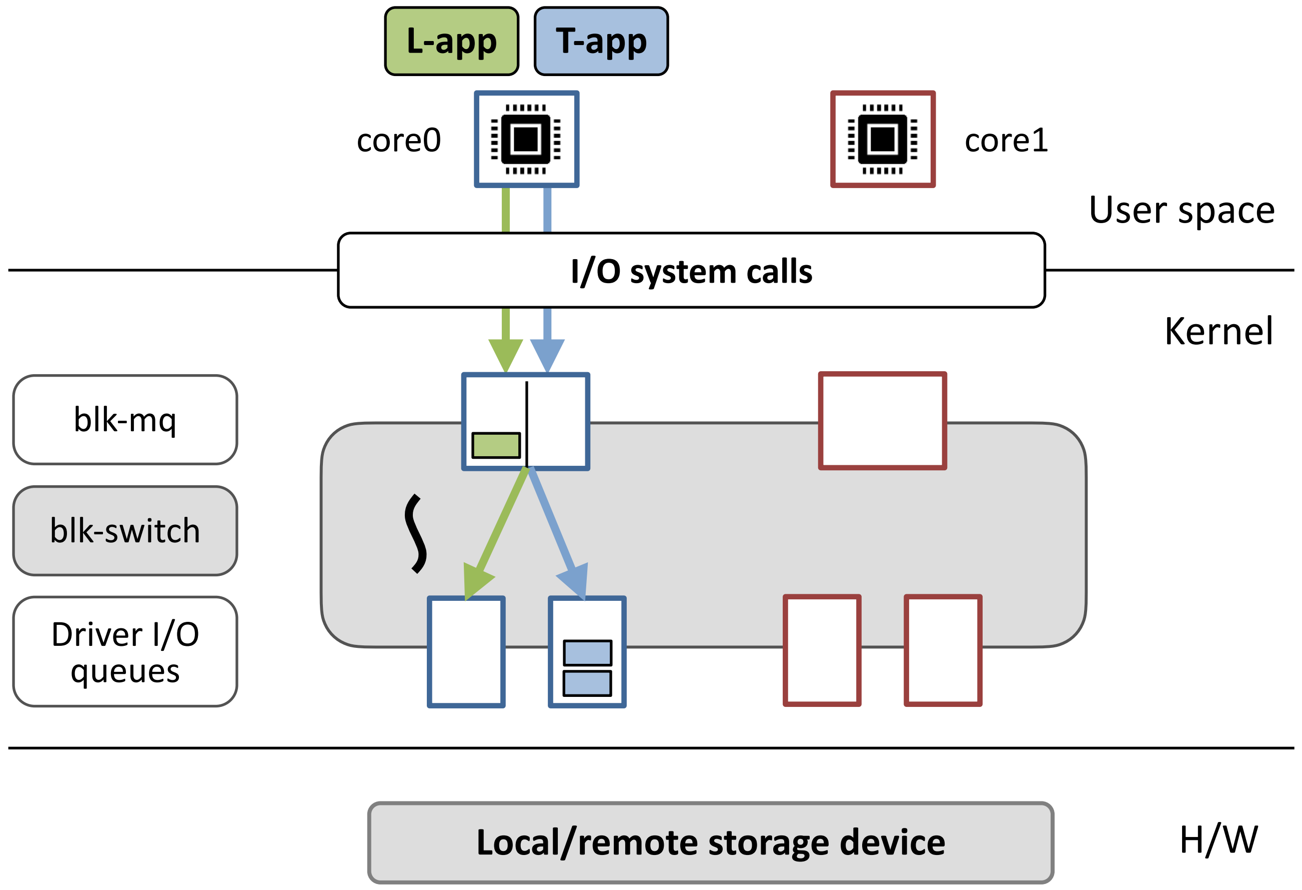
Prioritize **L-app request processing**

blk-switch Prioritization



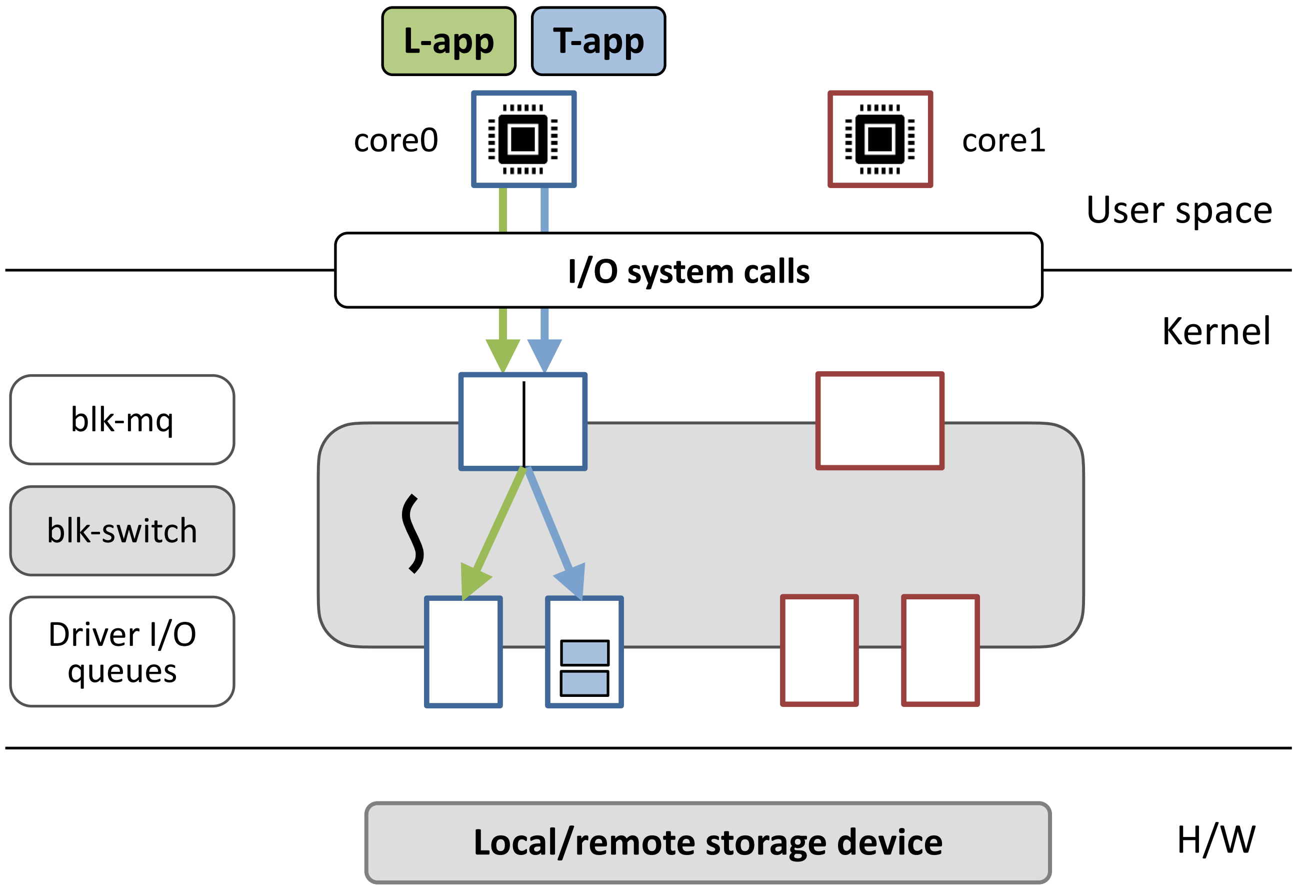
***Prioritize* L-app request processing**

blk-switch Prioritization



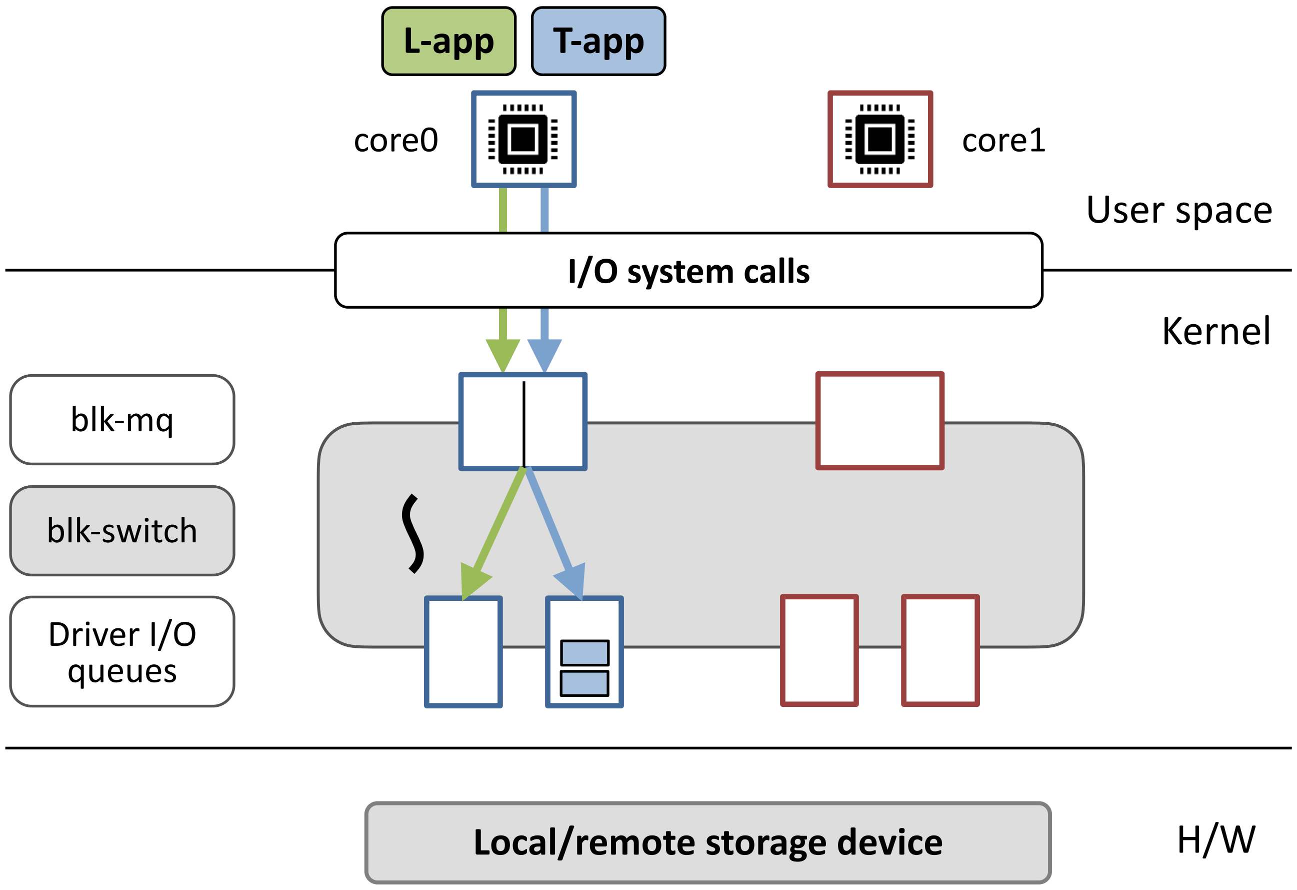
Prioritize **L-app request processing**

blk-switch Prioritization



Prioritize **L-app request processing**

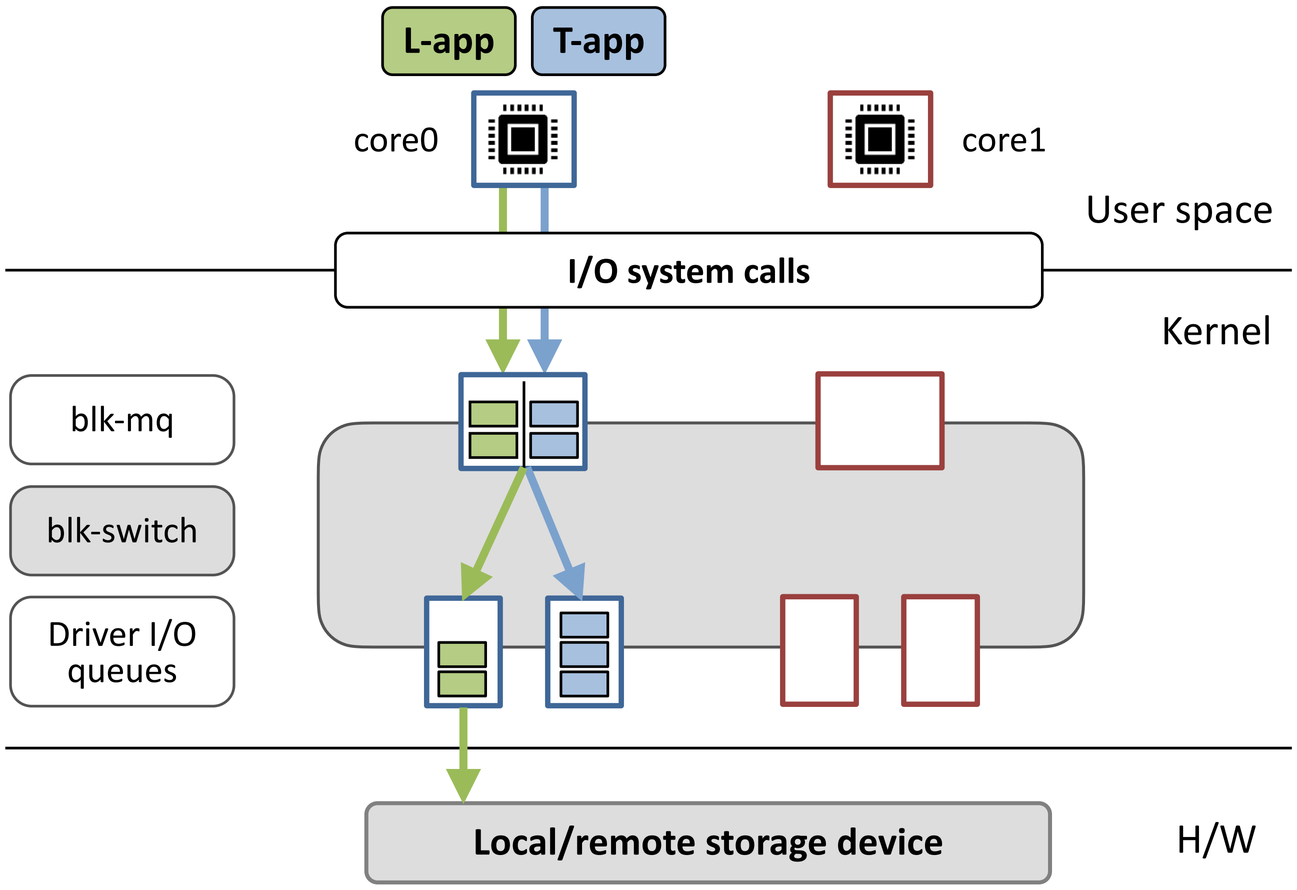
blk-switch Prioritization



Prioritize **L-app request processing**

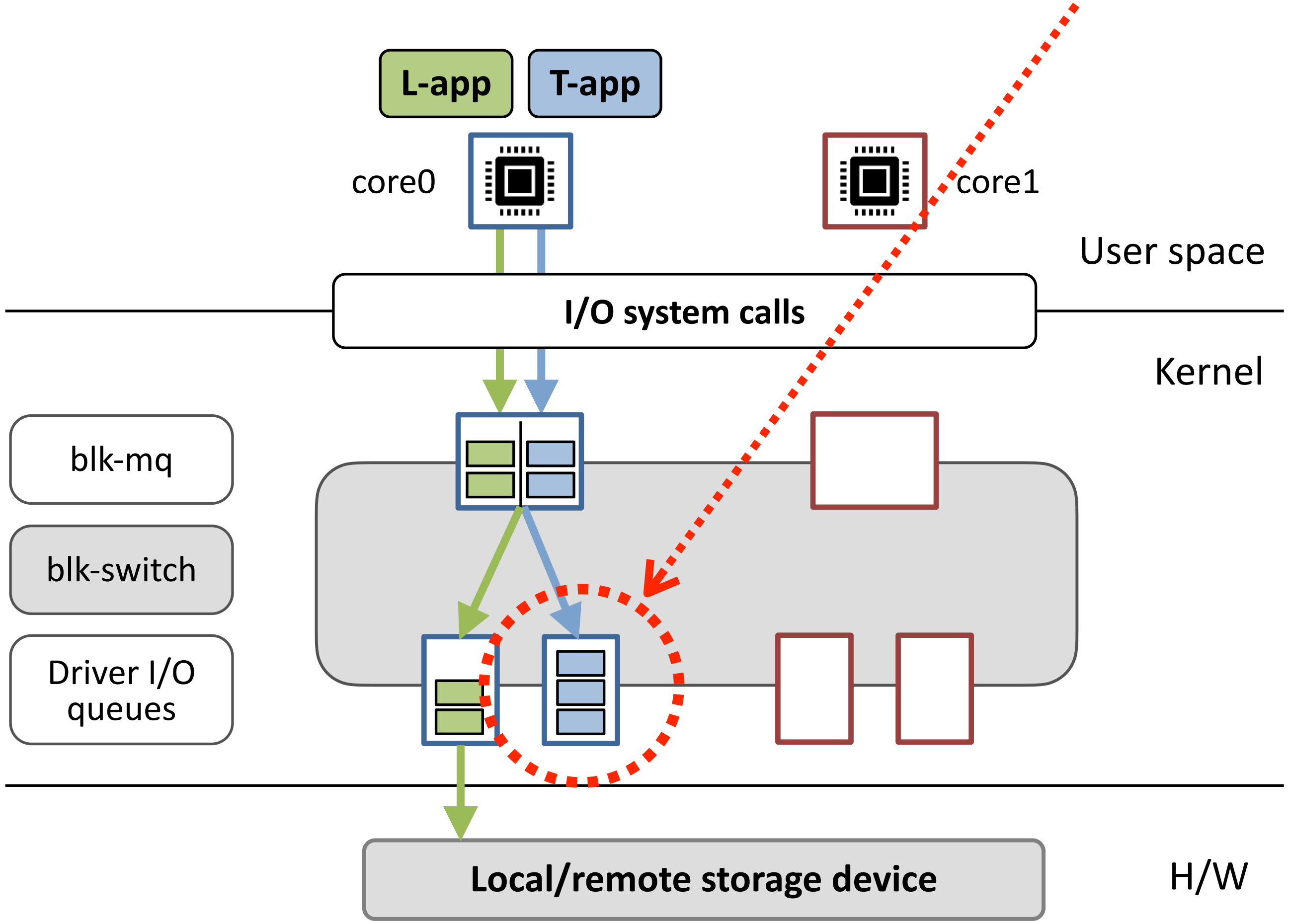
“Multi-egress queues + prioritization”: near optimal latency for L-apps

blk-switch Request Steering for transient loads



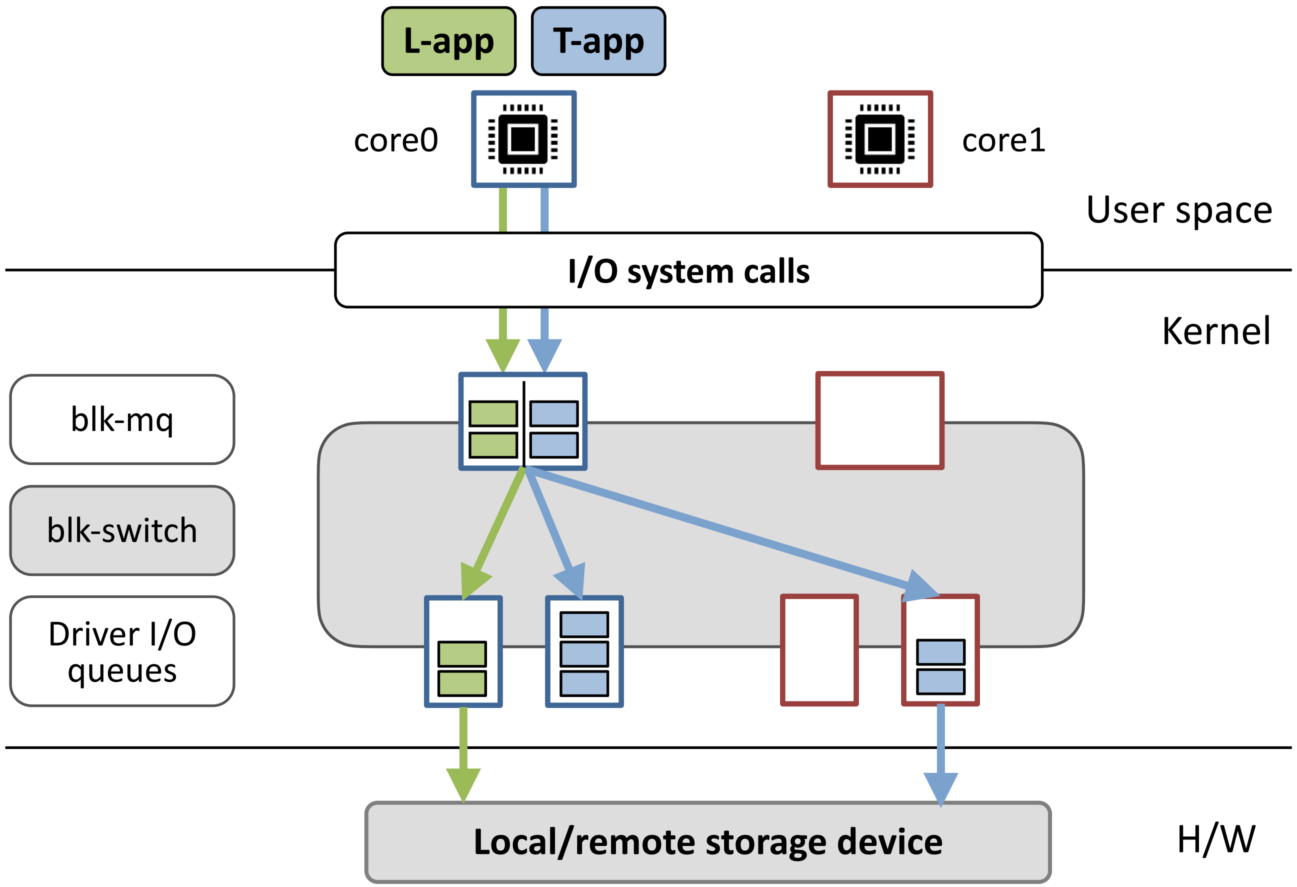
blk-switch Request Steering for transient loads

Challenge: Prioritization of L-apps can lead to transient starvation of T-apps



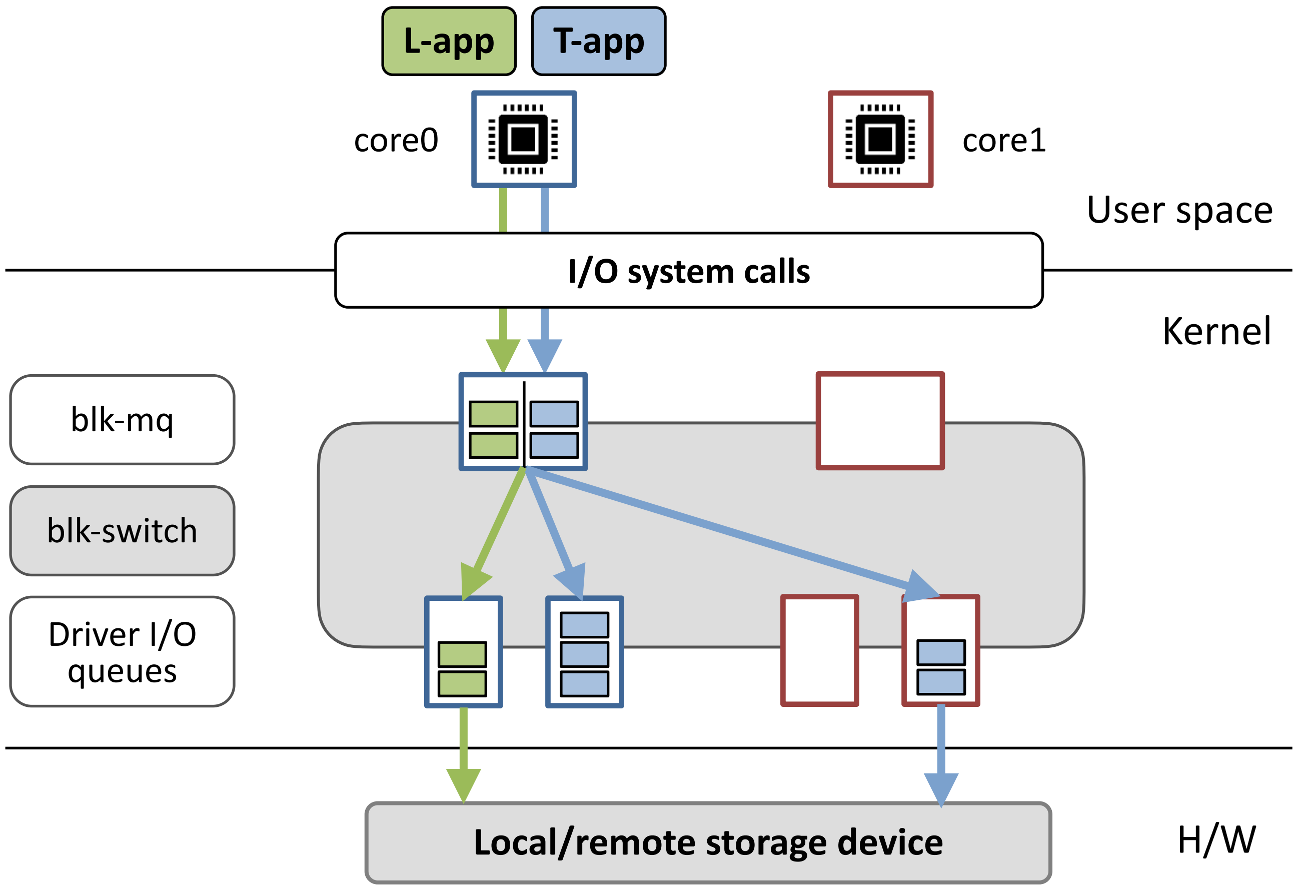
blk-switch Request Steering for transient loads

Challenge: Prioritization of L-apps can lead to transient starvation of T-apps



blk-switch Request Steering for transient loads

Challenge: Prioritization of L-apps can lead to transient starvation of T-apps



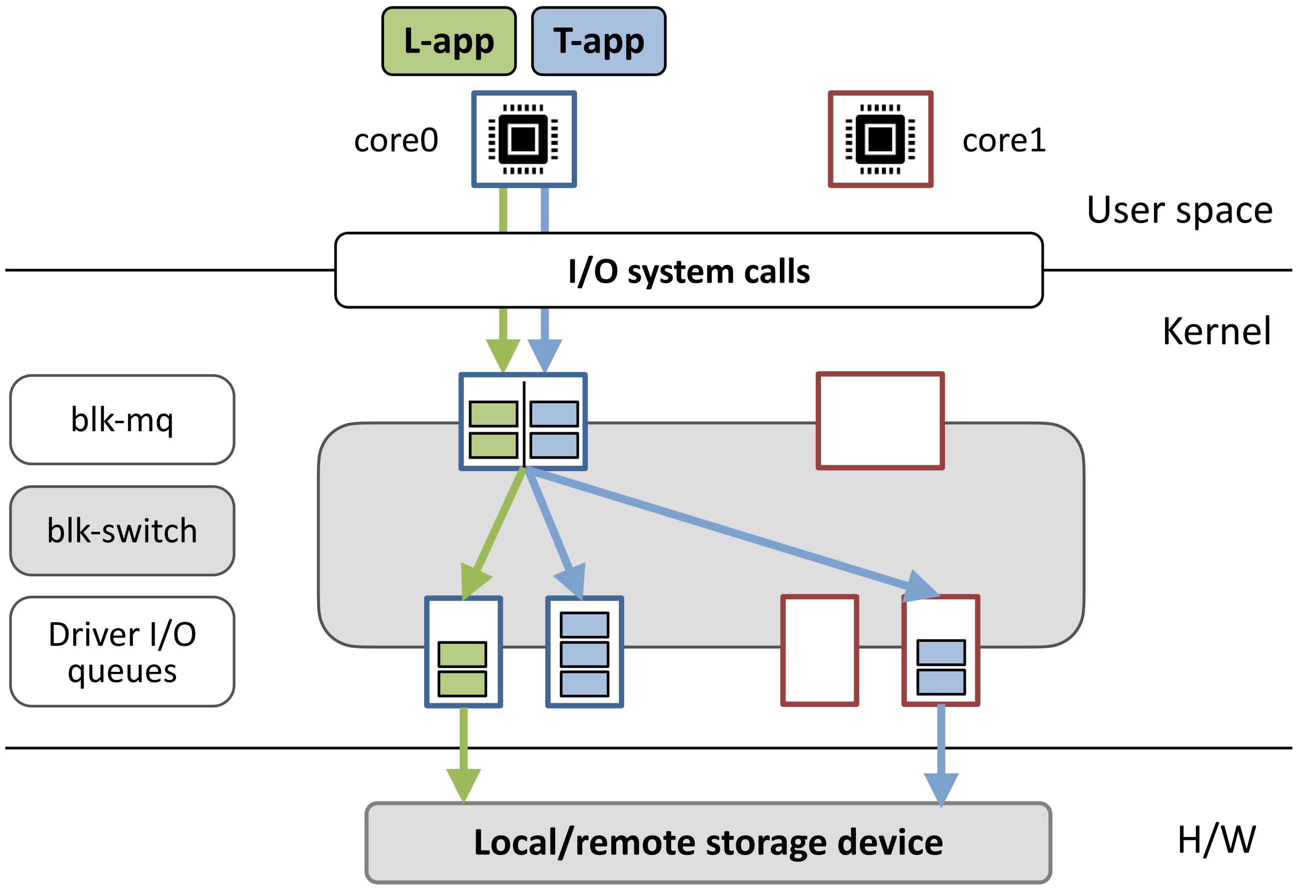
Steer **requests to underutilized cores at per-request granularity**

- Select target cores using known techniques
- Capture only T-app load

(Please see our paper)

blk-switch Request Steering for transient loads

Challenge: Prioritization of L-apps can lead to transient starvation of T-apps



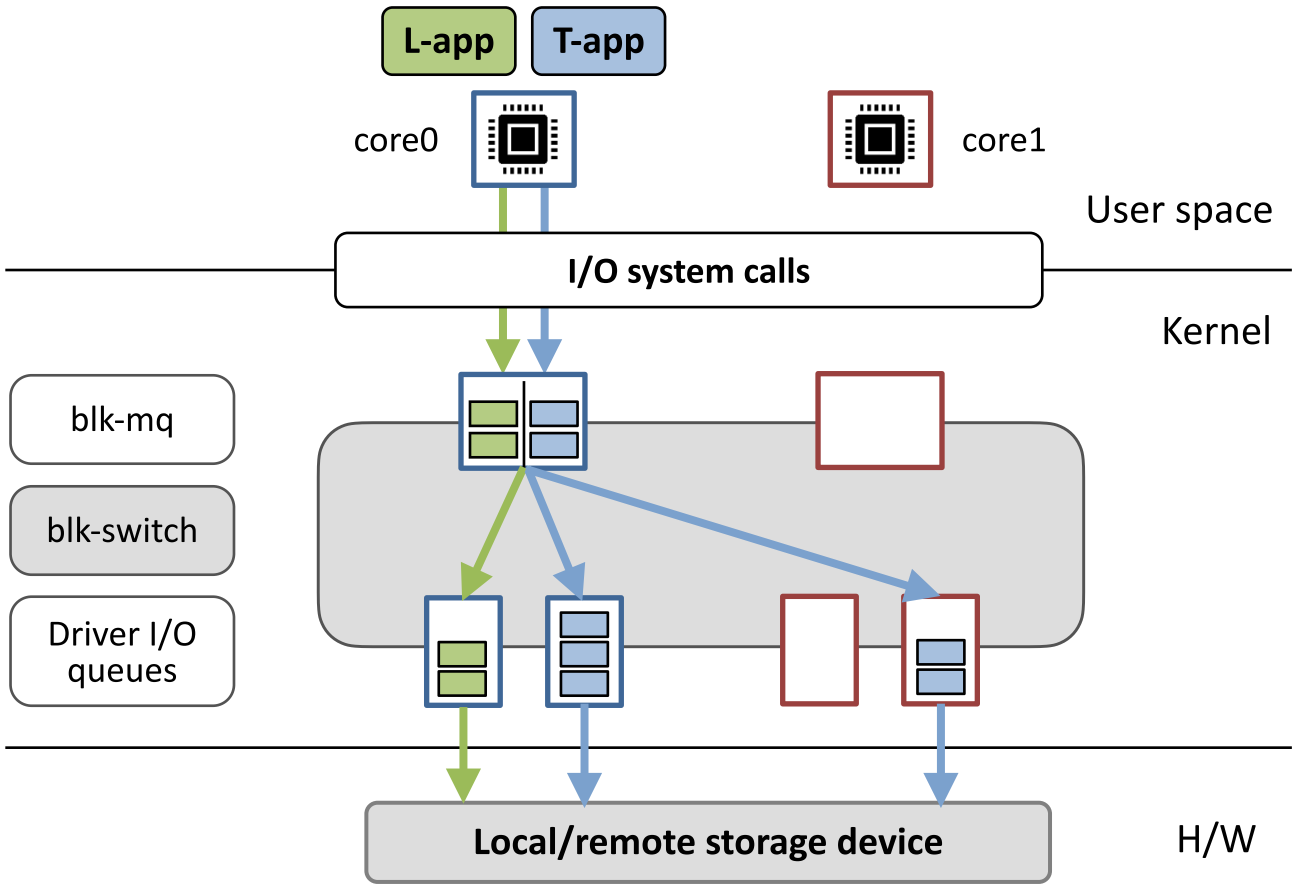
***Steer* requests to underutilized cores at per-request granularity**

- Select target cores using known techniques
- Capture only T-app load

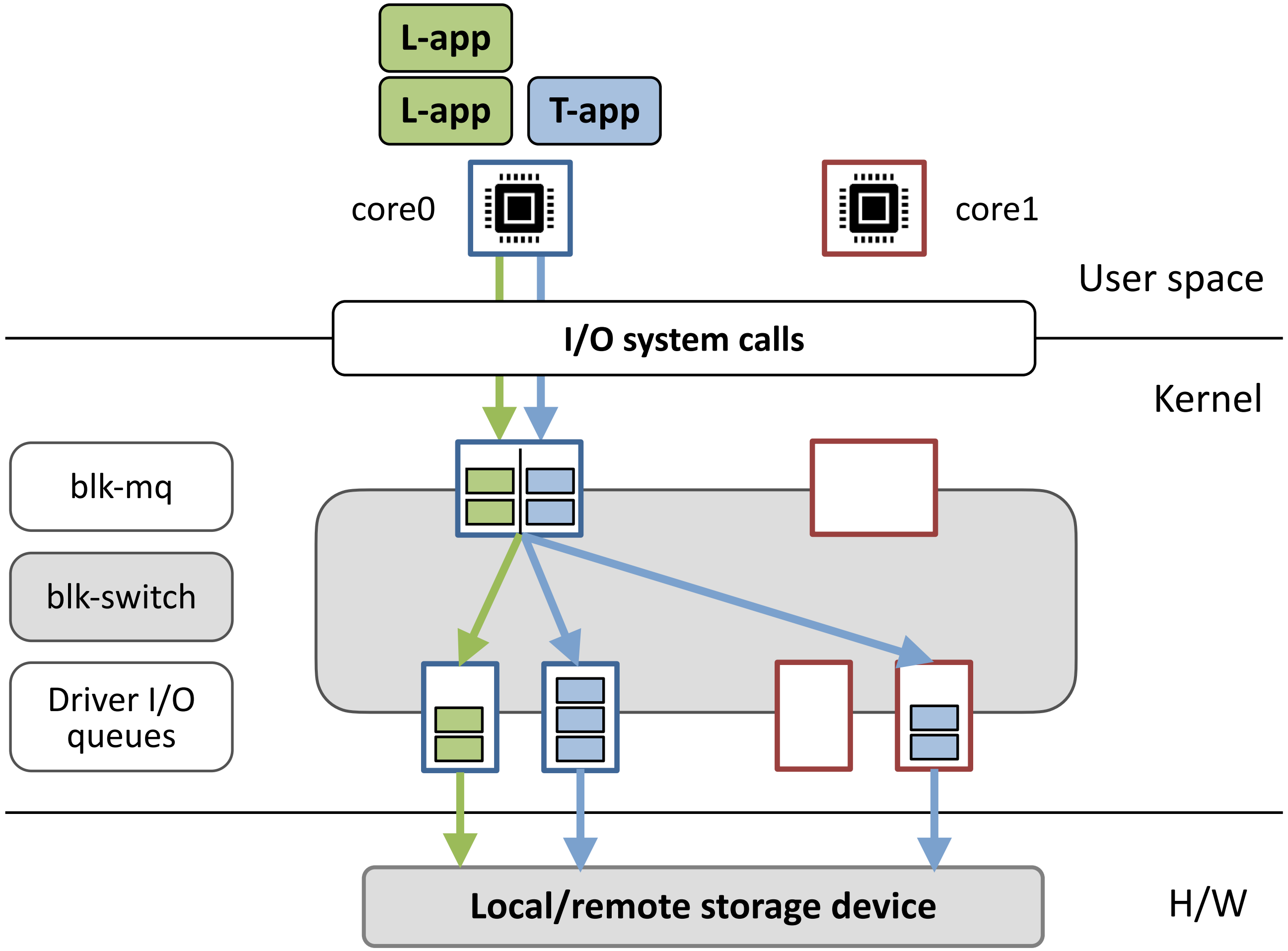
  (Please see our paper)

Request steering allows blk-switch to maintain high throughput, even under transient loads

blk-switch Application Steering for persistent loads

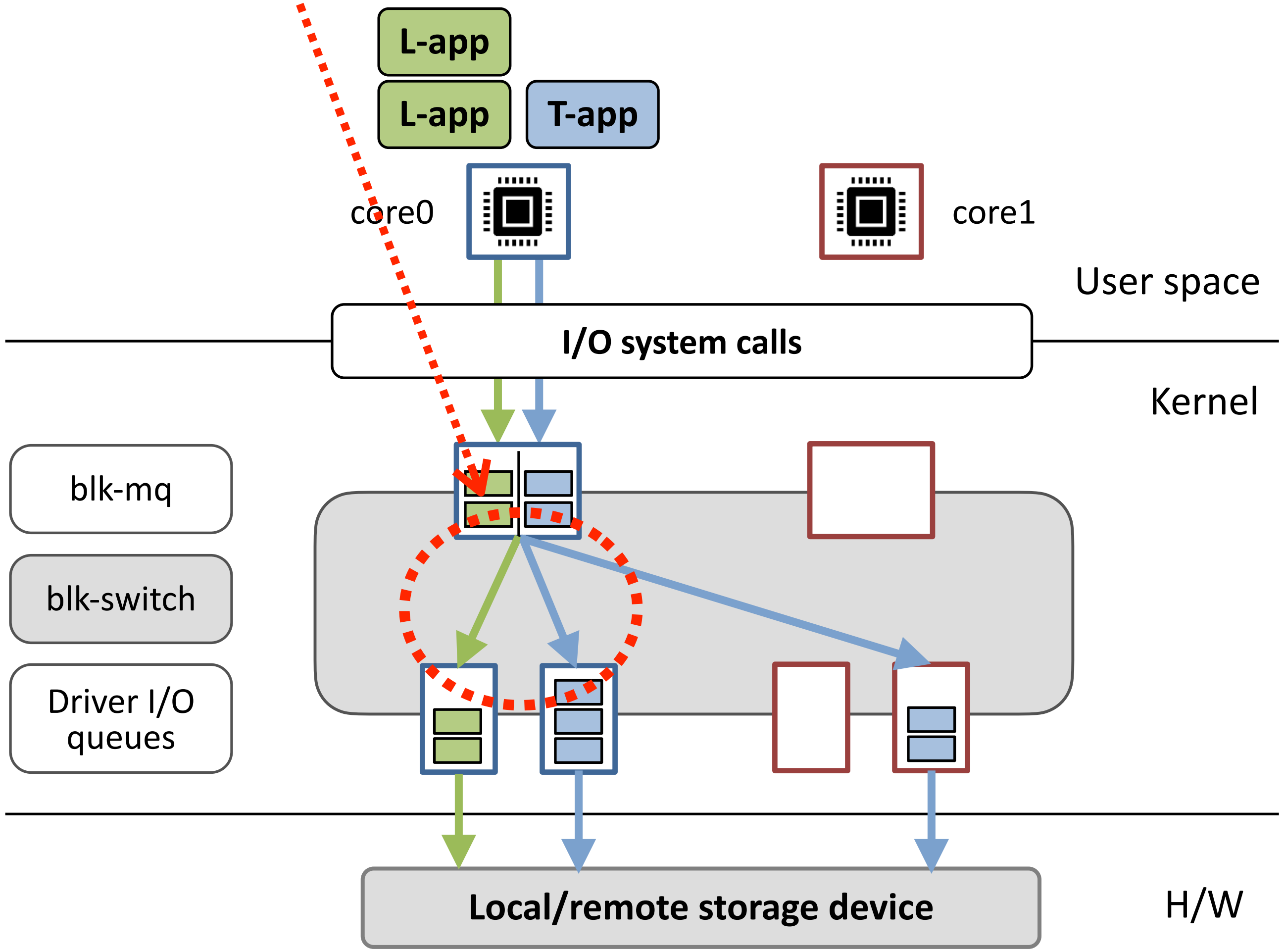


blk-switch Application Steering for persistent loads



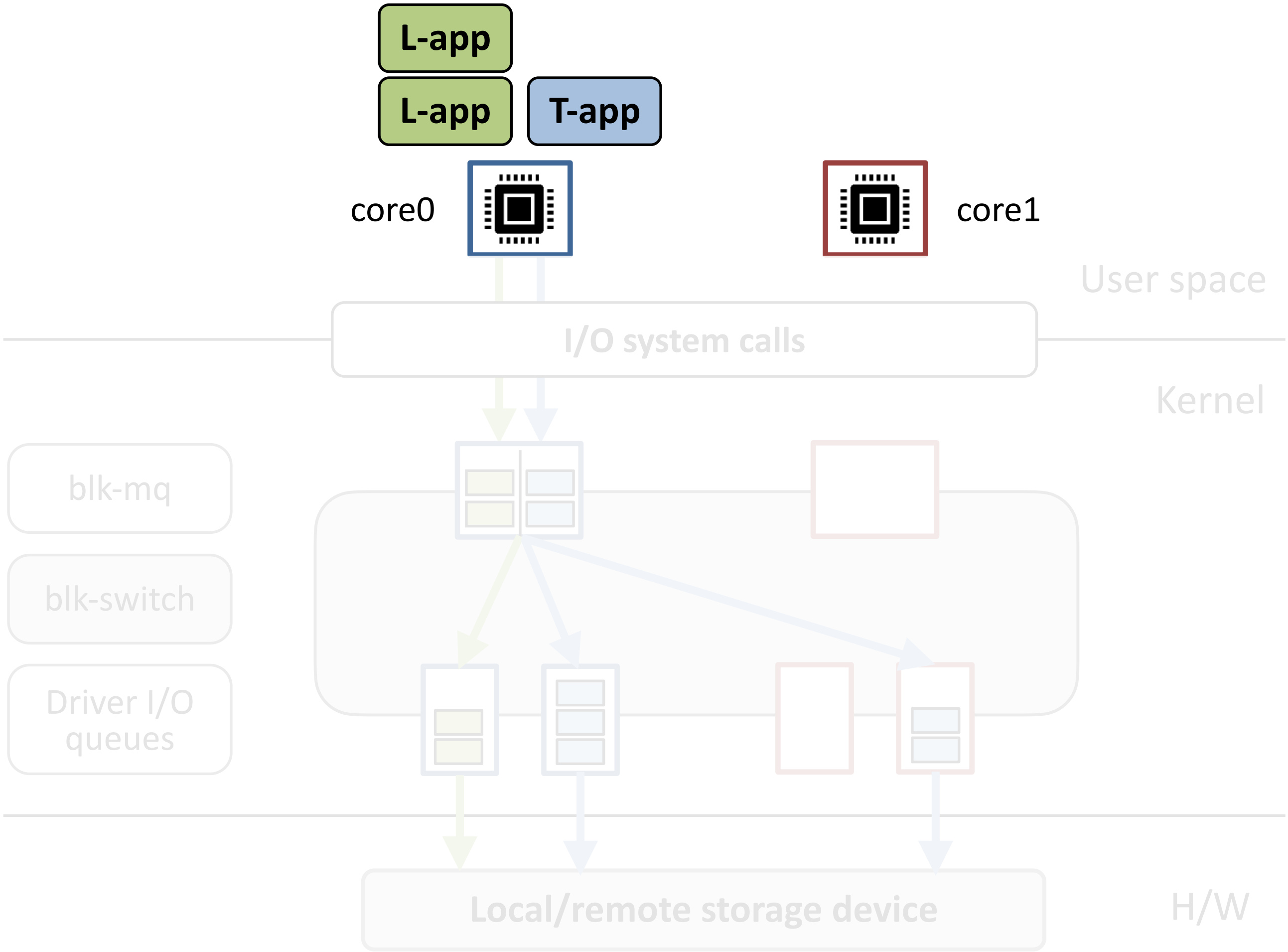
blk-switch Application Steering for persistent loads

Challenge: Persistent loads lead to high system overheads



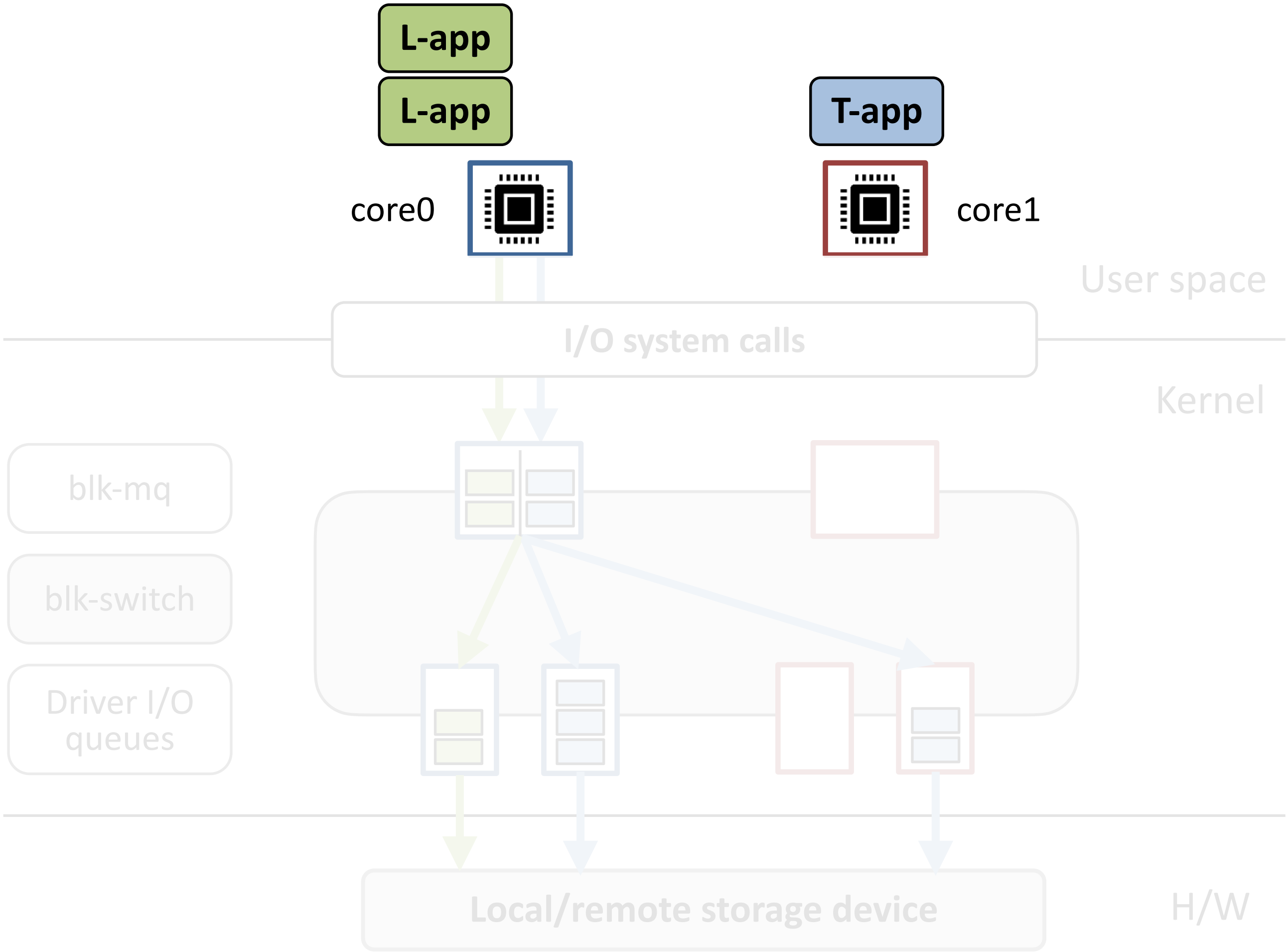
blk-switch Application Steering for persistent loads

Challenge: Persistent loads lead to high system overheads



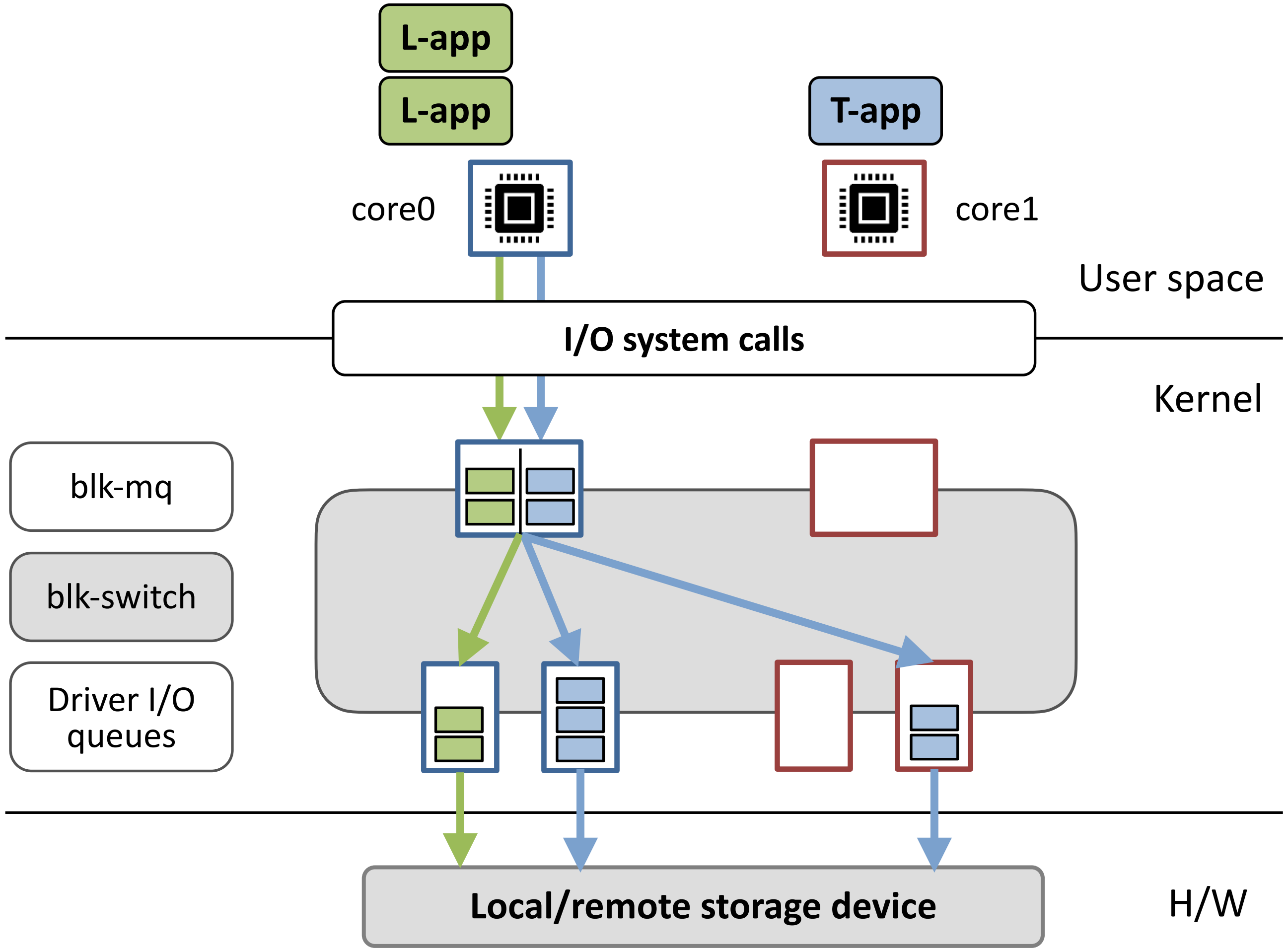
blk-switch Application Steering for persistent loads

Challenge: Persistent loads lead to high system overheads



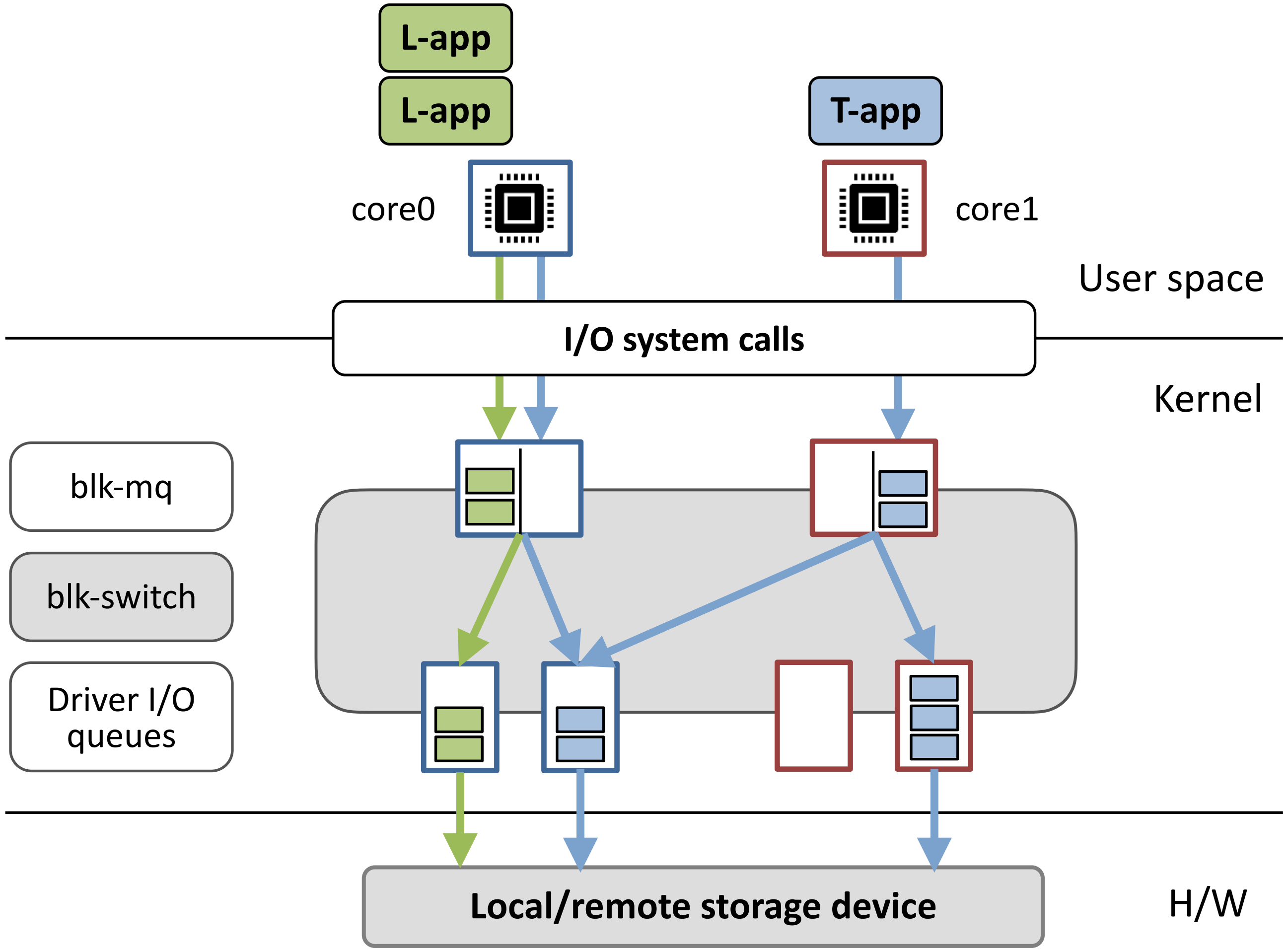
blk-switch Application Steering for persistent loads

Challenge: Persistent loads **lead to high system overheads**



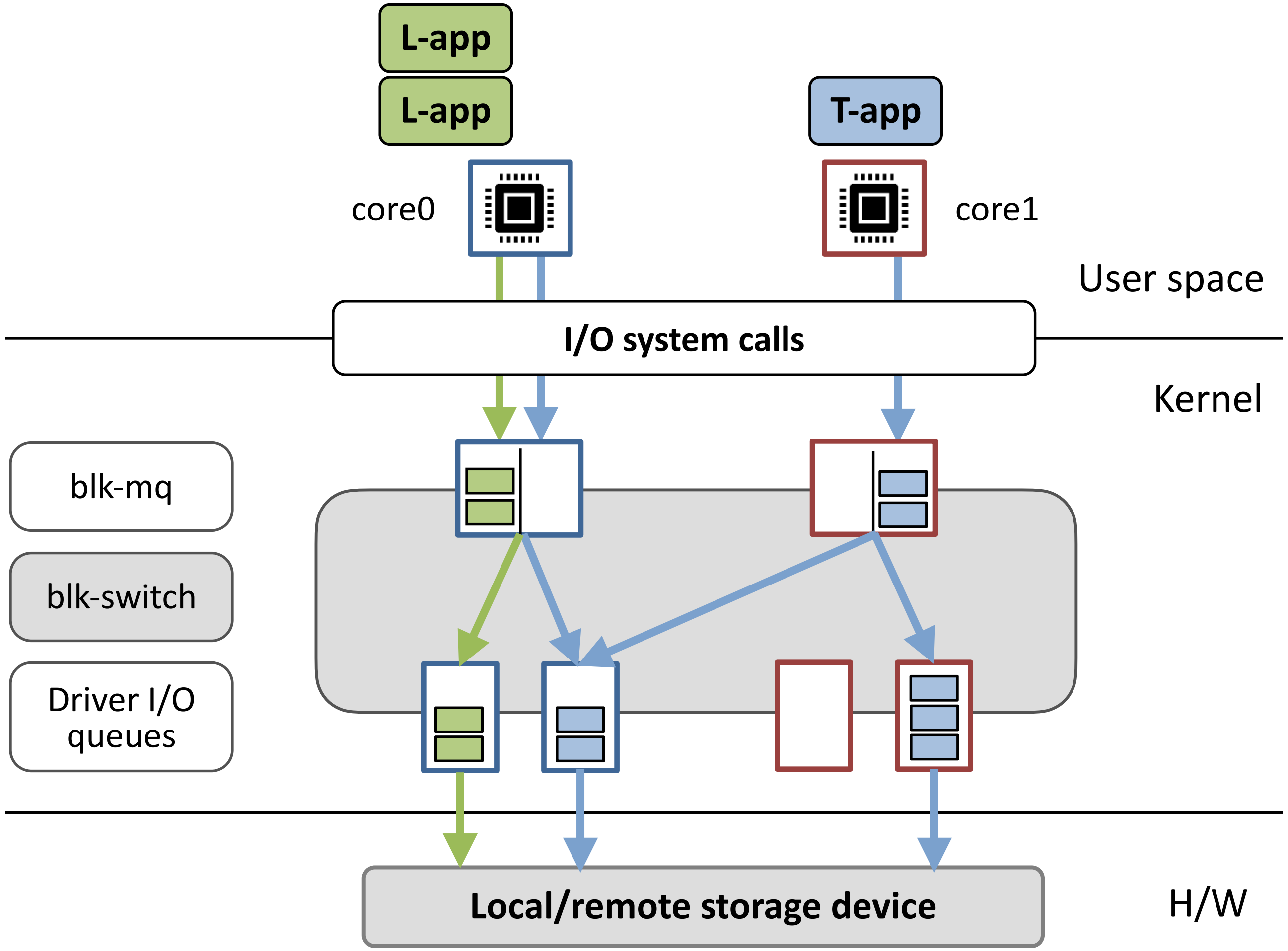
blk-switch Application Steering for persistent loads

Challenge: Persistent loads lead to high system overheads



blk-switch Application Steering for persistent loads

Challenge: Persistent loads lead to high system overheads

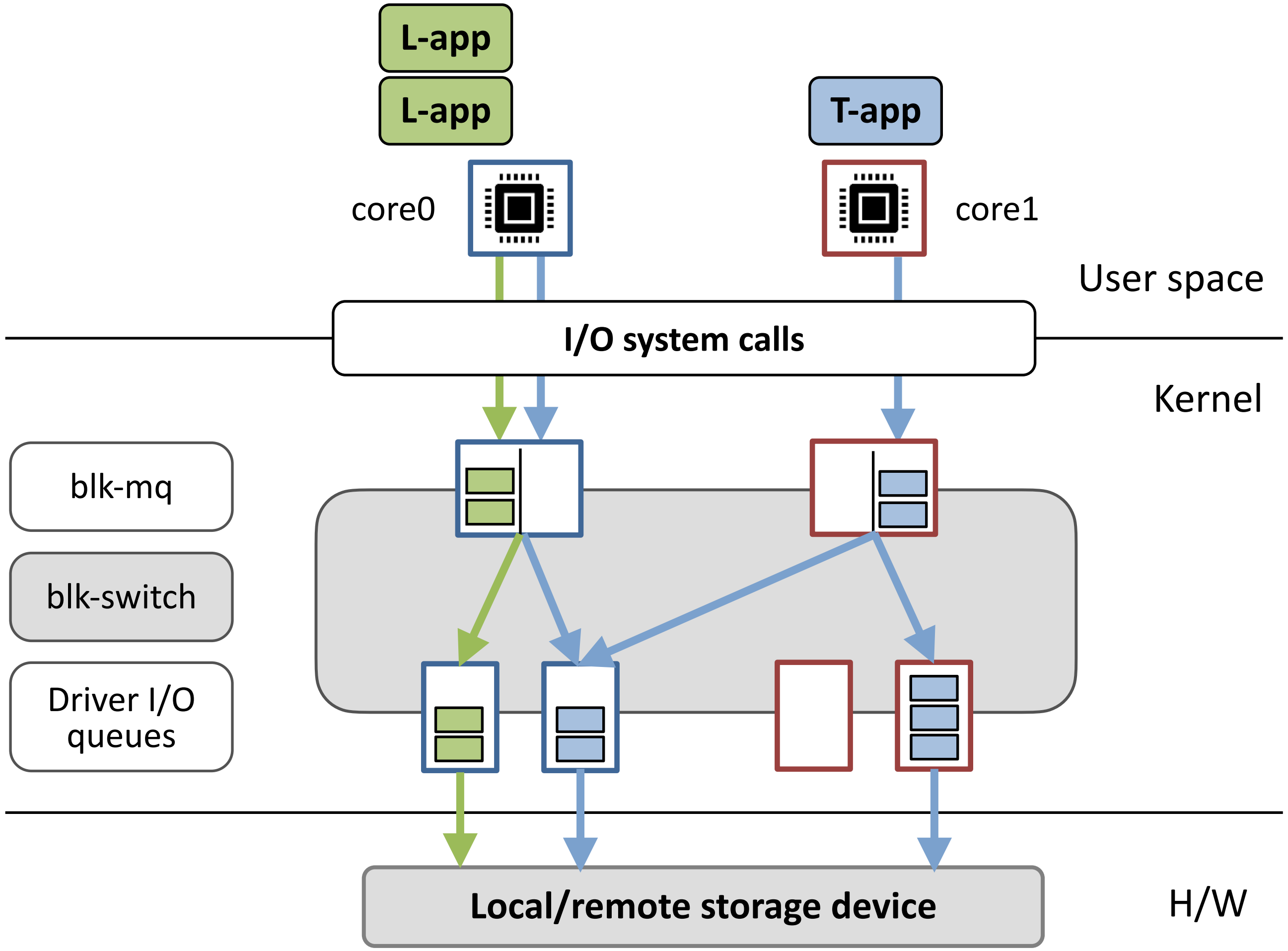


Steer **apps to cores with low average utilization**

- Long-term time scales (e.g., every 10ms)
- Both L-app and T-app load

blk-switch Application Steering for persistent loads

Challenge: Persistent loads lead to high system overheads



Steer **apps to cores with low average utilization**

- Long-term time scales (e.g., every 10ms)
- Both L-app and T-app load

- **High throughput for T-apps even under persistent loads**
- **Even lower latency for L-apps due to fewer context-switches**

blk-switch Evaluation Setup

blk-switch Evaluation Setup

- **Implemented entirely in the Linux kernel with minimal changes (LOC: $\sim$928)**

blk-switch Evaluation Setup

- **Implemented entirely in the Linux kernel with minimal changes (LOC: ∼928)**
- **To stress test blk-switch**
  - Complex interaction among the compute, storage, and network stacks
  - Evaluate “remote storage access” scenarios

blk-switch Evaluation Setup

- **Implemented entirely in the Linux kernel with minimal changes (LOC: $\sim$928)**

- **To stress test blk-switch**
  - Complex interaction among the compute, storage, and network stacks
  - Evaluate “remote storage access” scenarios

- **To push the bottleneck to the storage stack processing**
  - Two 32-core servers connected directly over 100Gbps

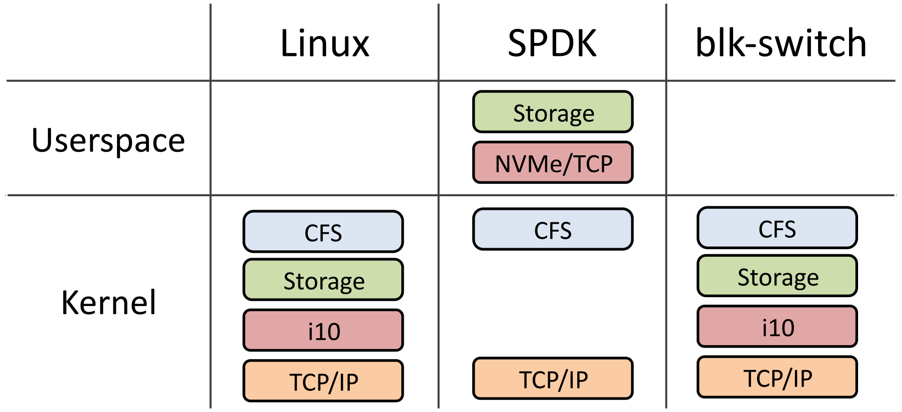
blk-switch Evaluation Setup

- **Implemented entirely in the Linux kernel with minimal changes (LOC: ~928)**

- **To stress test blk-switch**
  - Complex interaction among the compute, storage, and network stacks
  - Evaluate “remote storage access” scenarios

- **To push the bottleneck to the storage stack processing**
  - Two 32-core servers connected directly over 100Gbps

- **To access data on remote servers**
  - Linux/blk-switch use i10 (state-of-the-art remote I/O stack, NSDI’20)
  - SPDK uses userspace NVMe-over-TCP

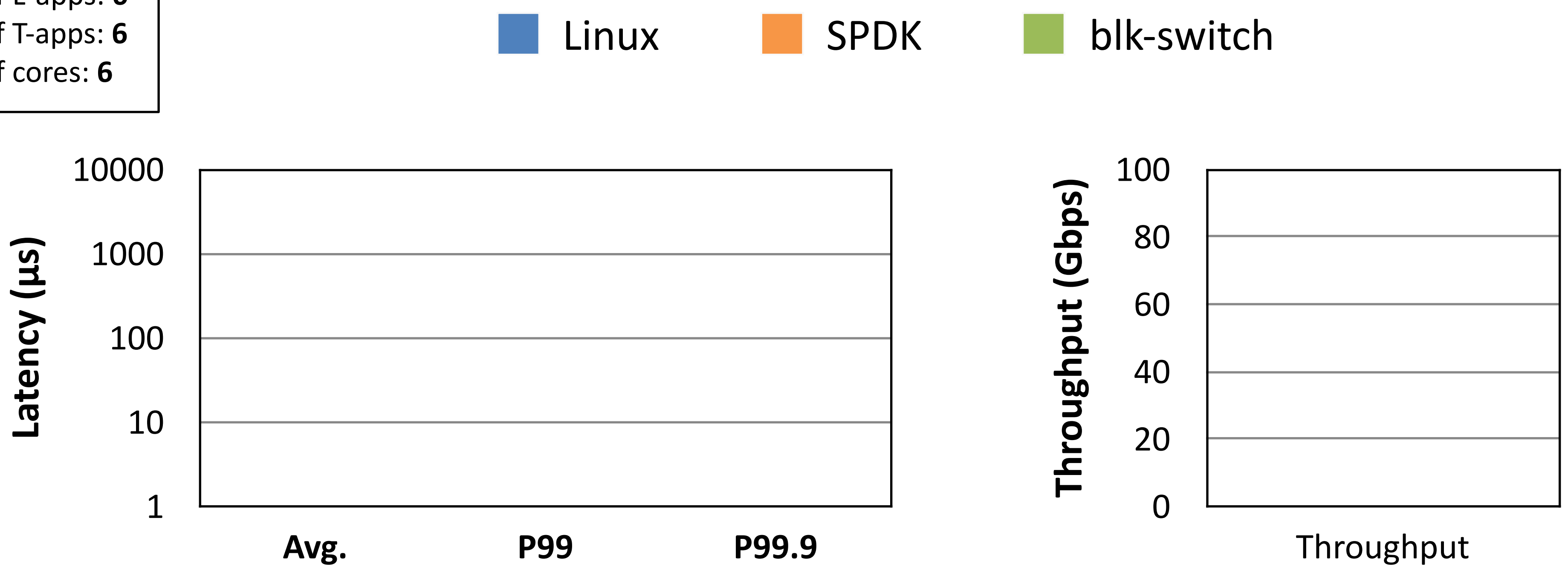


High Contention Scenario (In-memory)

Configurations:
- Number of L-apps: **6**
- Number of T-apps: **6**
- Number of cores: **6**

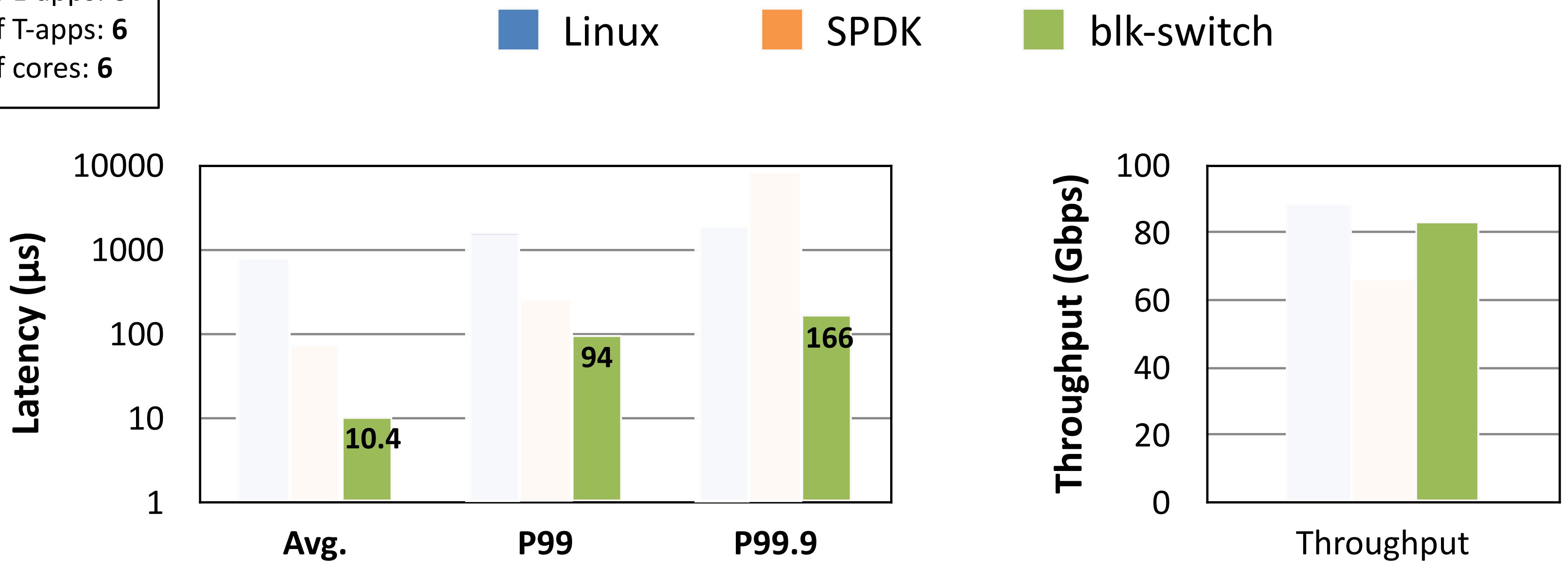
High Contention Scenario (In-memory)

Configurations:
- Number of L-apps: **6**
- Number of T-apps: **6**
- Number of cores: **6**



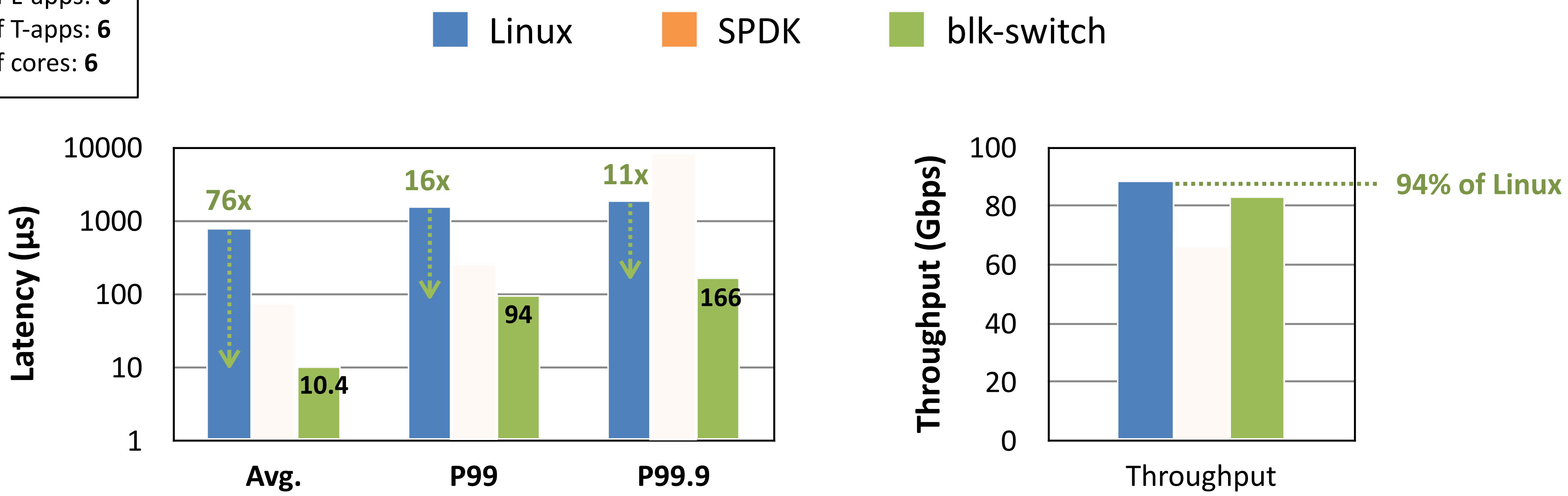
High Contention Scenario (In-memory)

Configurations:
- Number of L-apps: **6**
- Number of T-apps: **6**
- Number of cores: **6**



High Contention Scenario (In-memory)

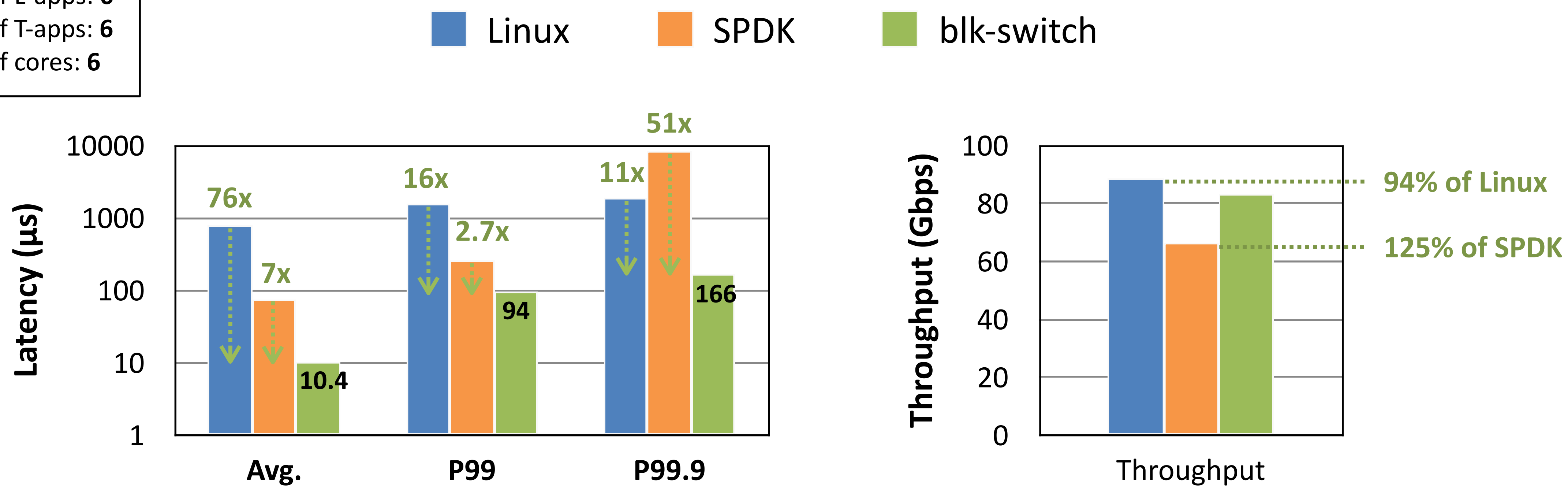
Configurations:
- Number of L-apps: **6**
- Number of T-apps: **6**
- Number of cores: **6**



vs. Linux: low latency by avoiding HoL and high throughput by efficiently using multiple cores

High Contention Scenario (In-memory)

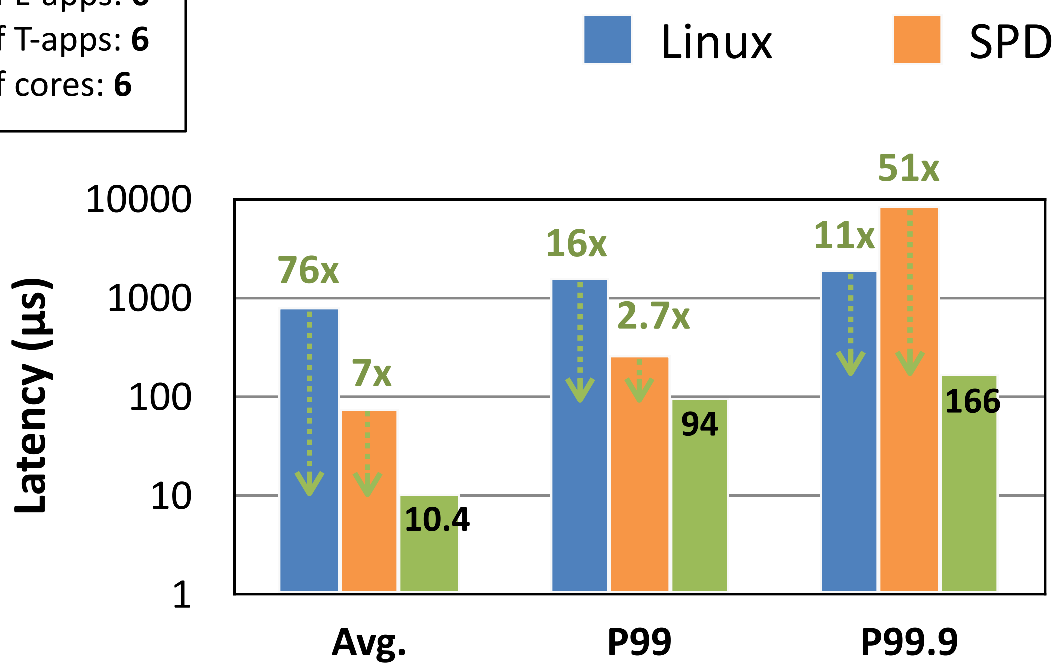
Configurations:
- Number of L-apps: **6**
- Number of T-apps: **6**
- Number of cores: **6**

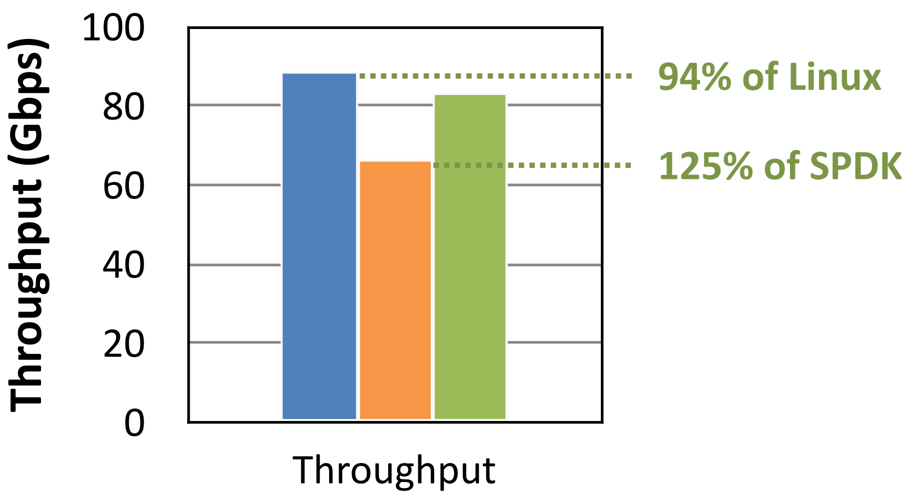


vs. Linux: low latency by avoiding HoL and high throughput by efficiently using multiple cores
vs. SPDK: better latency and throughput by avoiding drawback of polling-based system

High Contention Scenario (In-memory)

Configurations:
- Number of L-apps: **6**
- Number of T-apps: **6**
- Number of cores: **6**



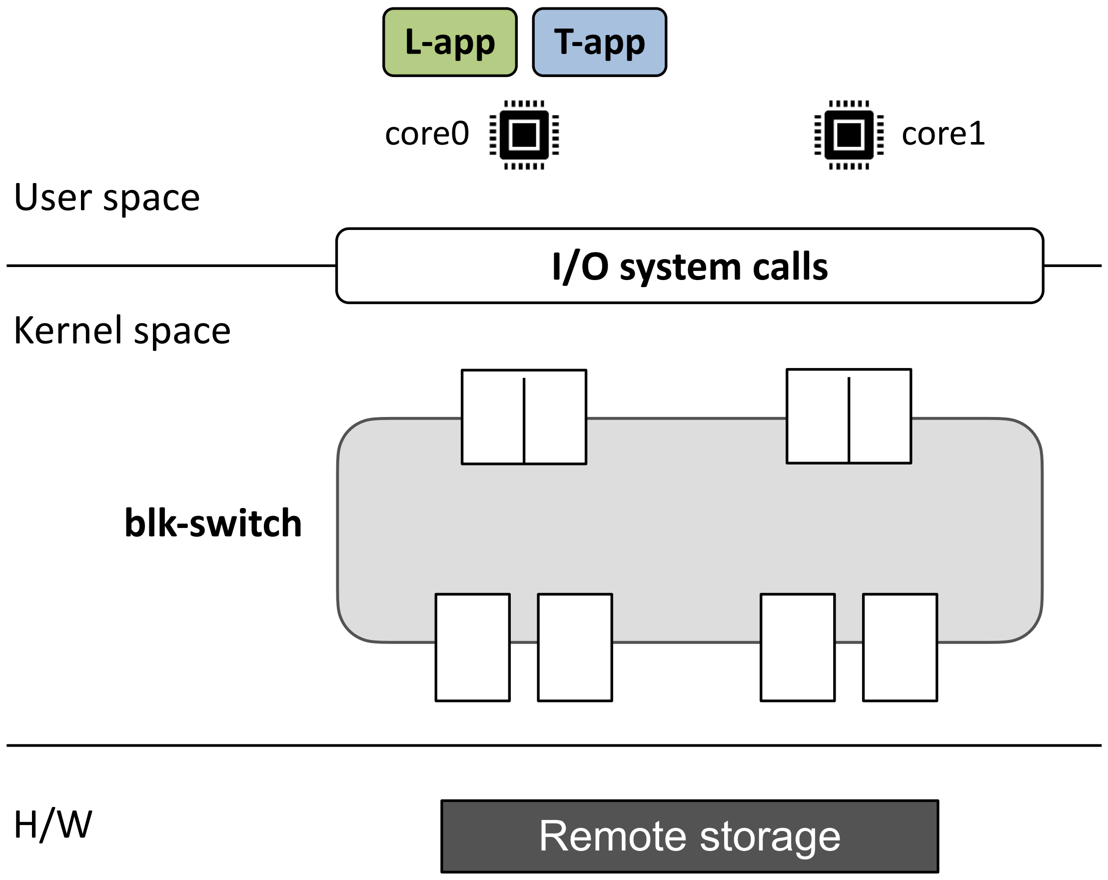


vs. Linux: low latency by avoiding HoL and high throughput by efficiently using multiple cores
vs. SPDK: better latency and throughput by avoiding drawback of polling-based system

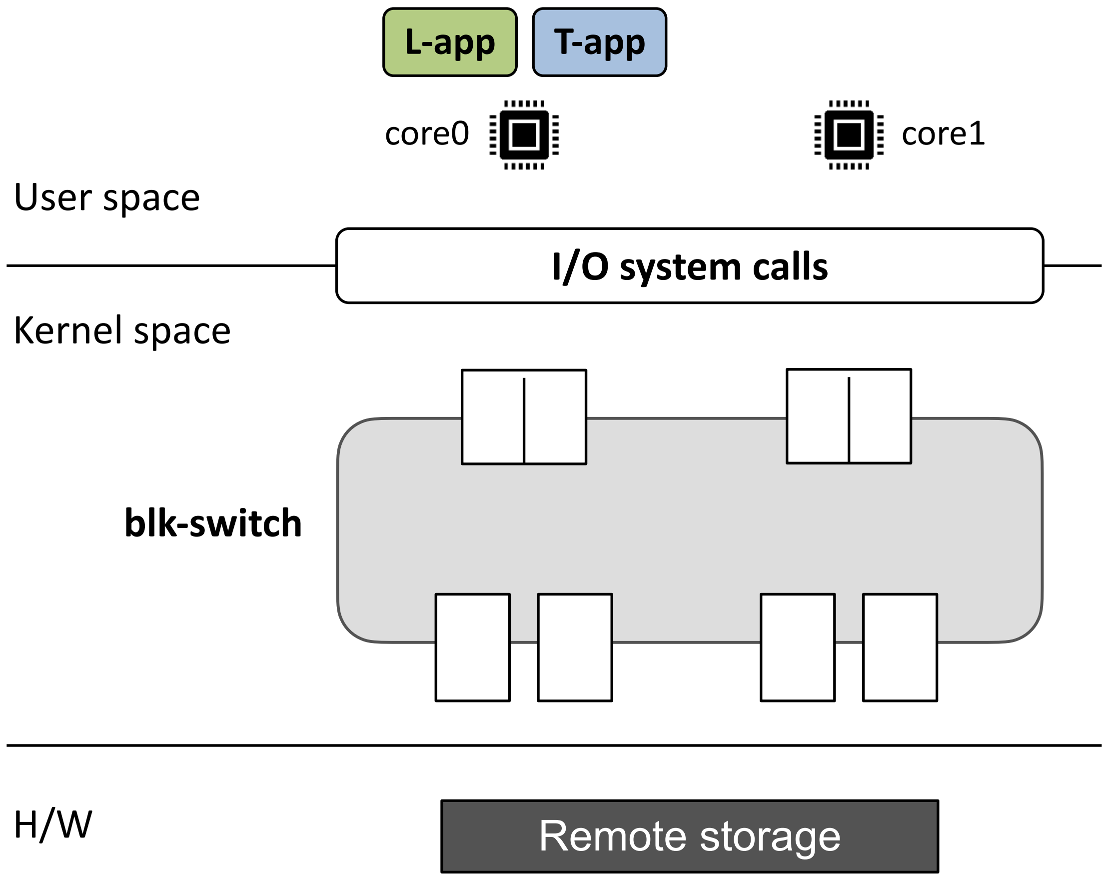
Even with tens of applications contending for host resources, blk-switch achieves both *µs*-scale latency and high throughput!

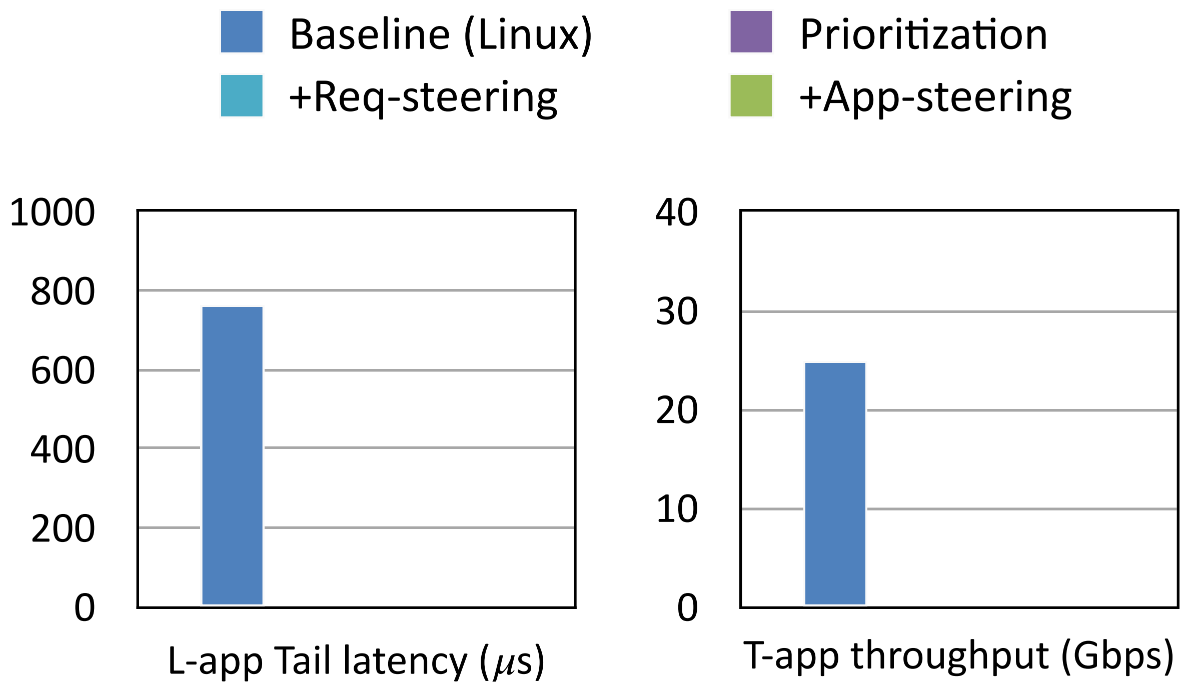
blk-switch Performance Breakdown

blk-switch Performance Breakdown

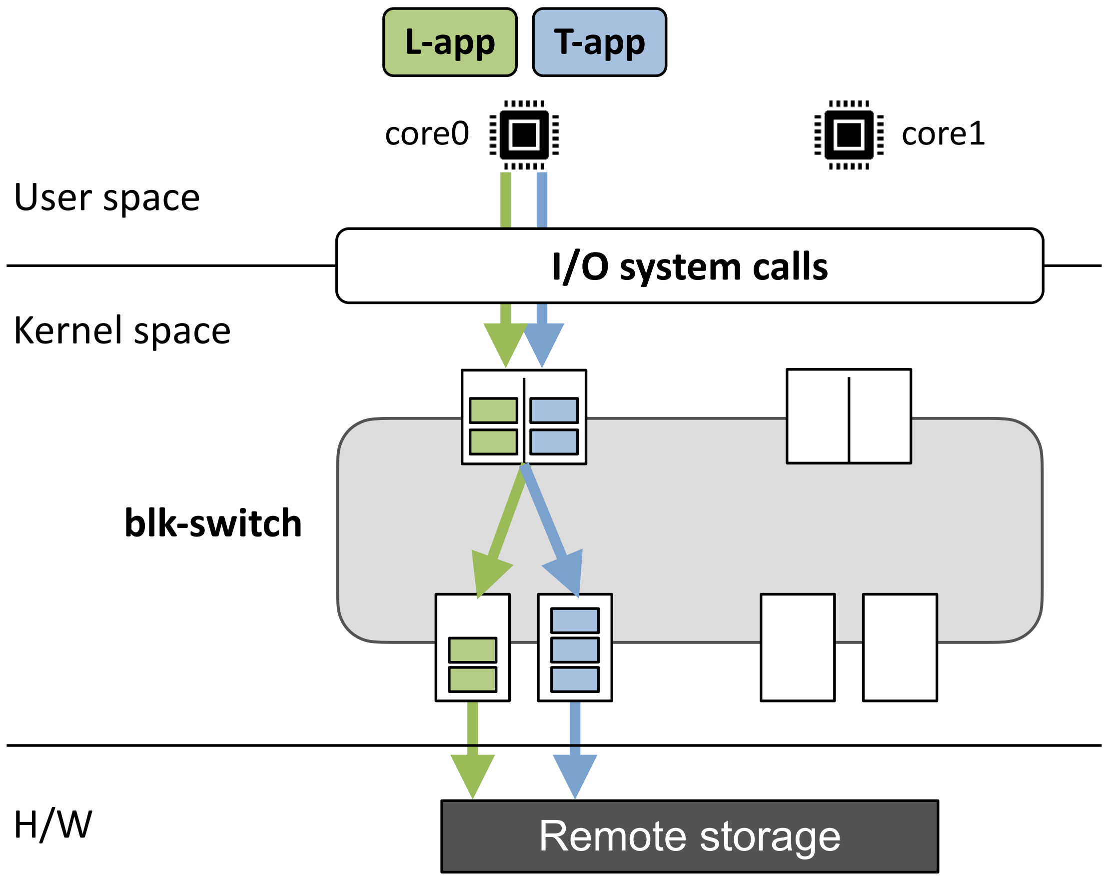


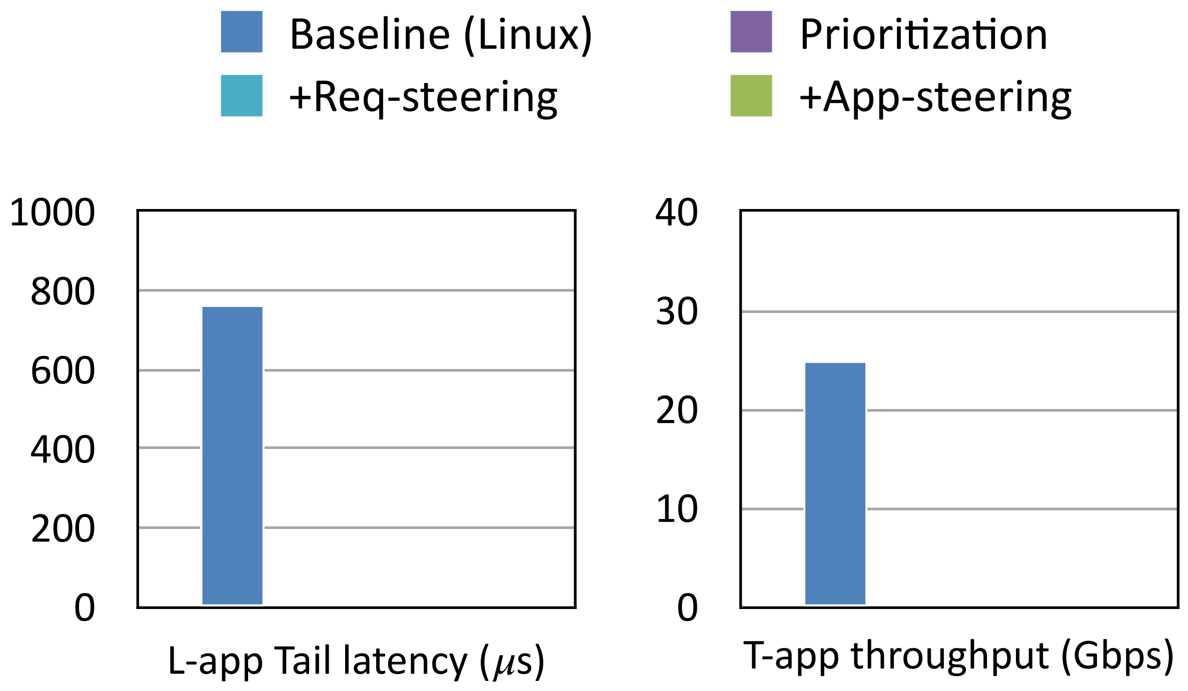
blk-switch Performance Breakdown



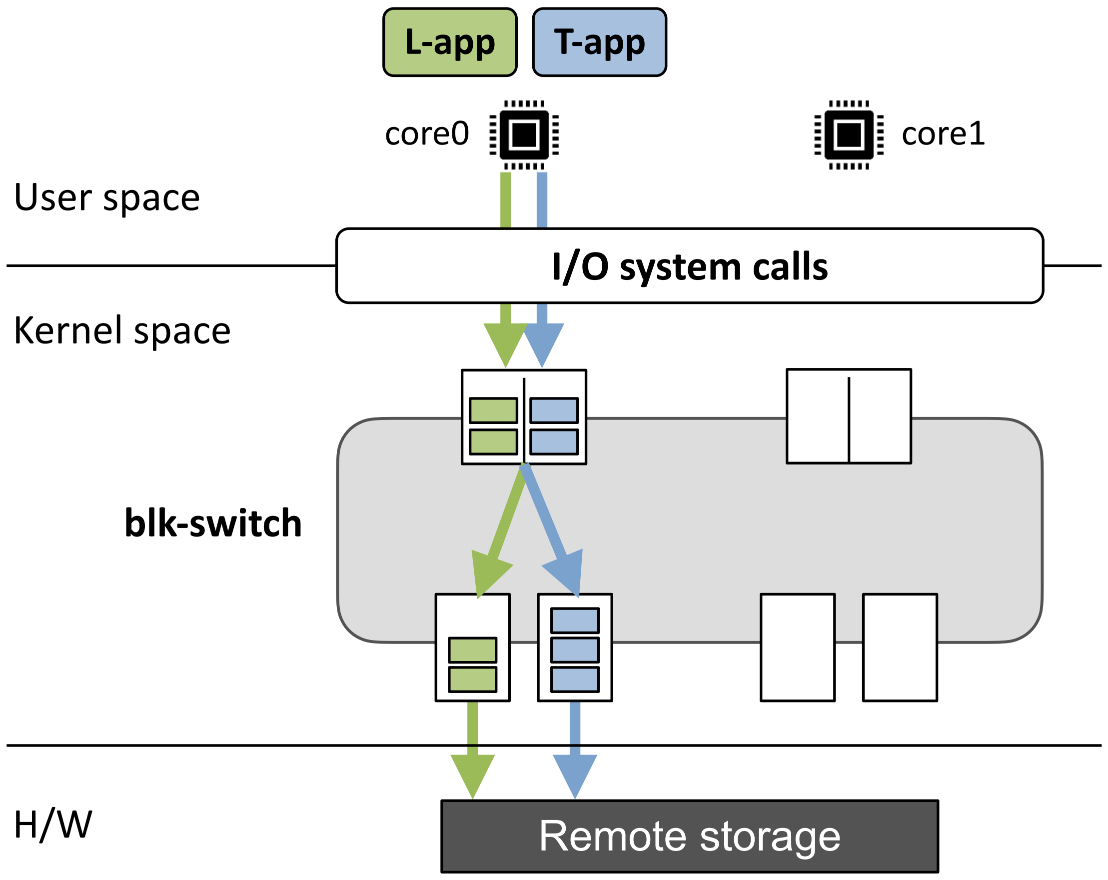


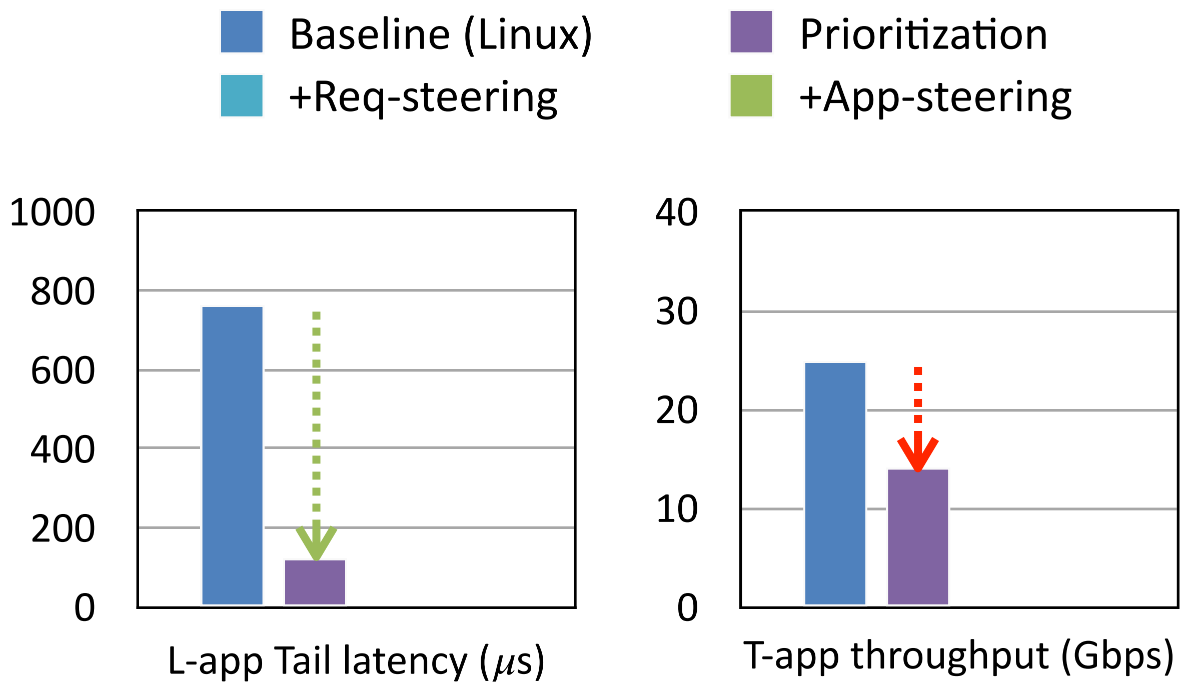
blk-switch Performance Breakdown



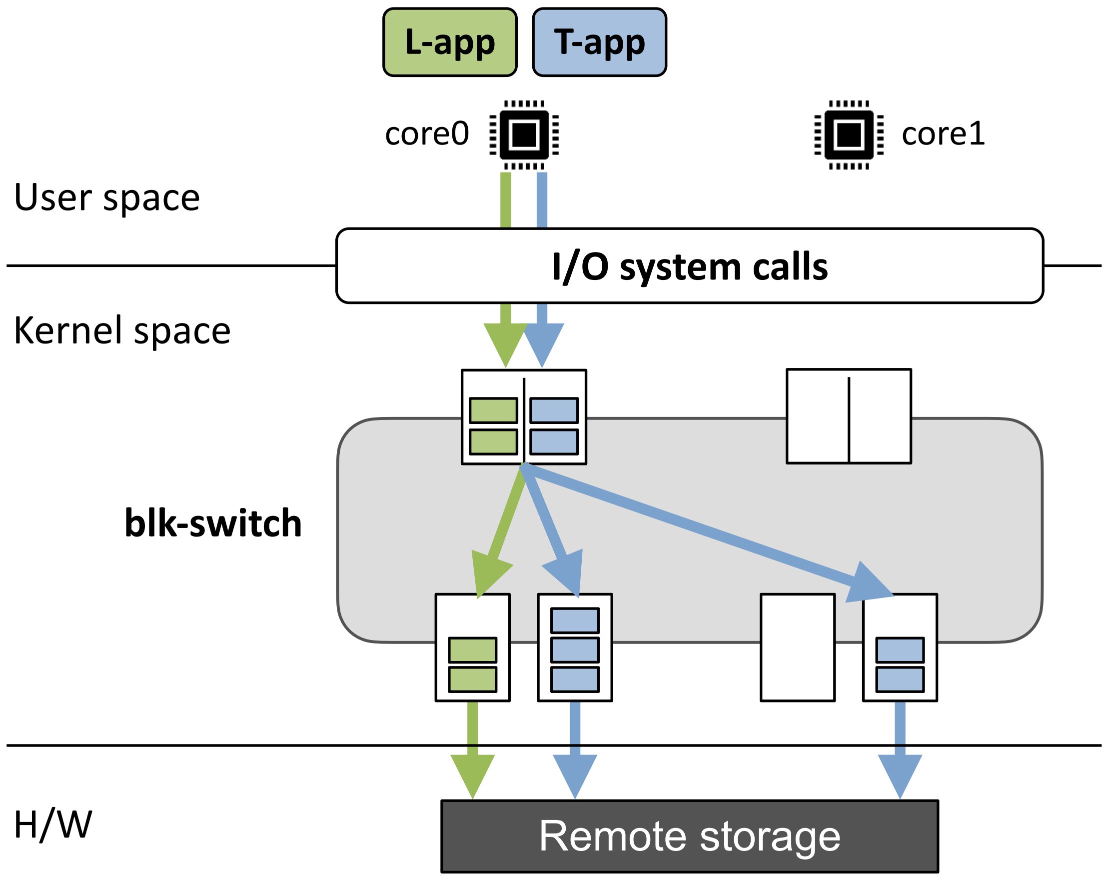


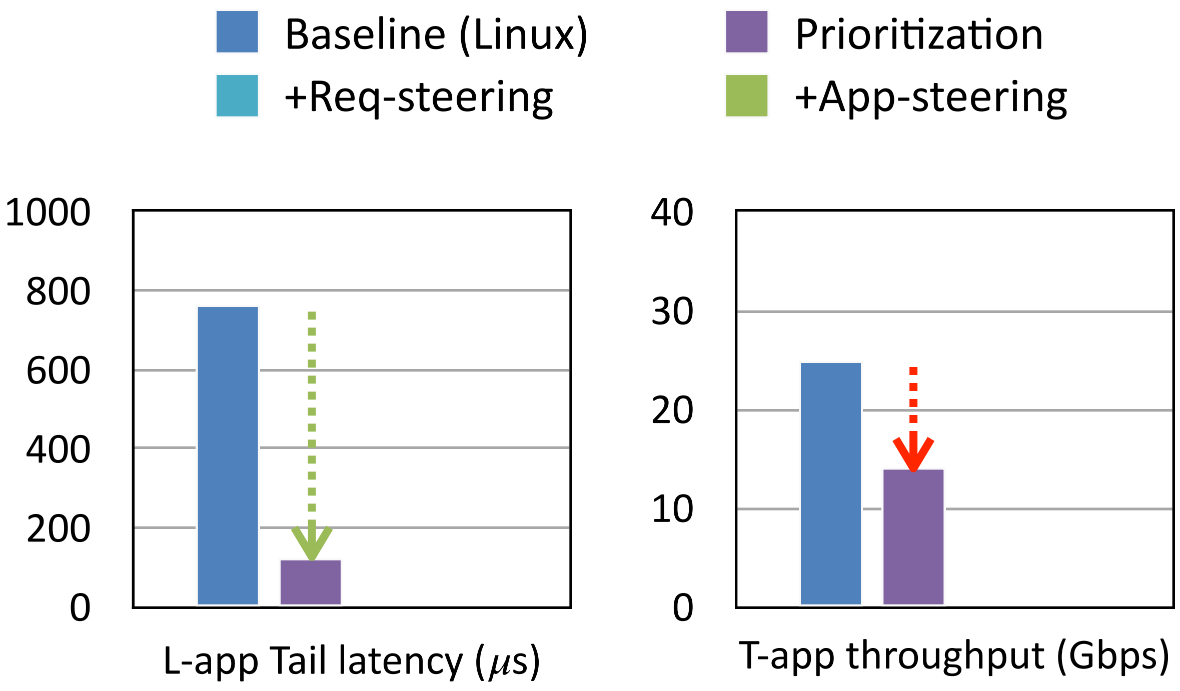
blk-switch Performance Breakdown



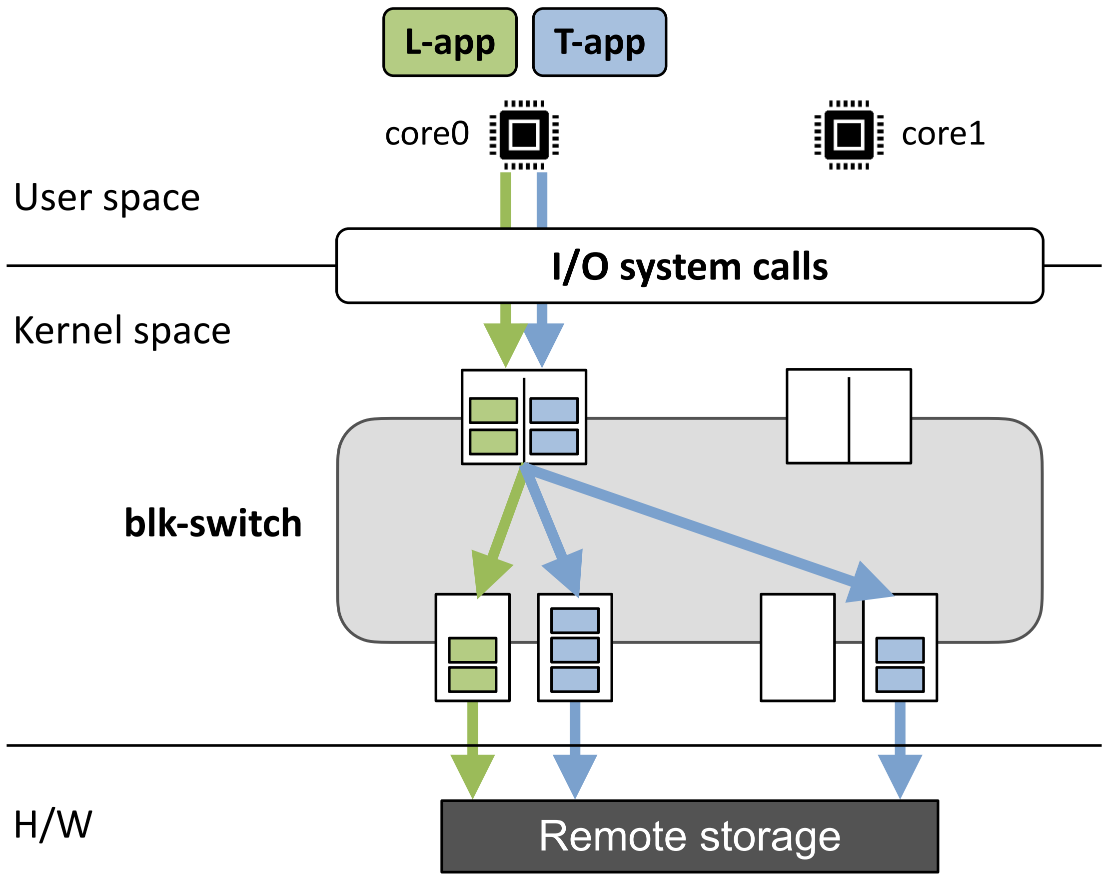


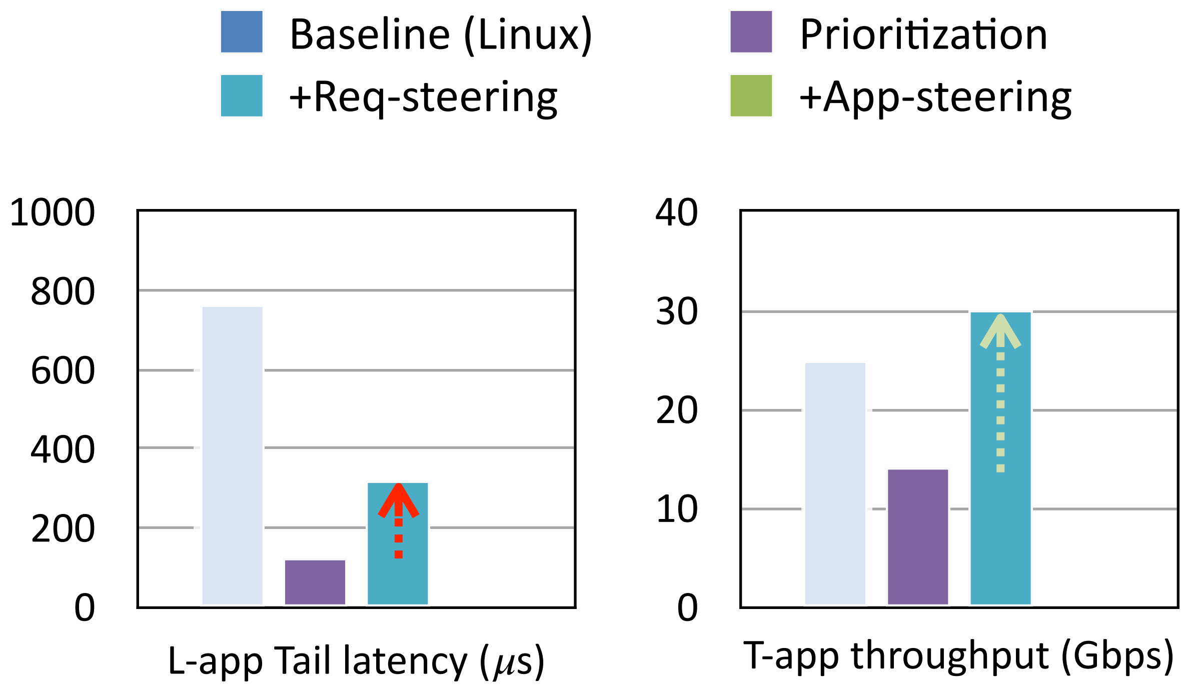
blk-switch Performance Breakdown



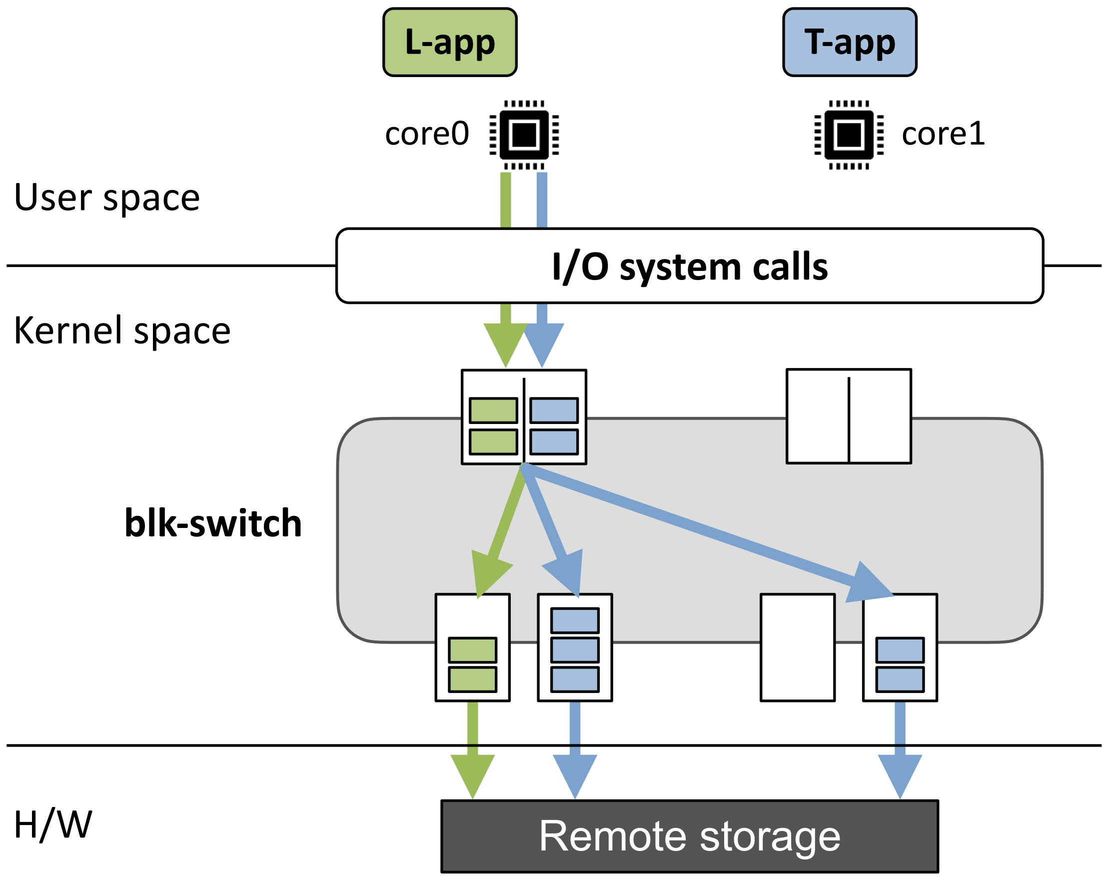


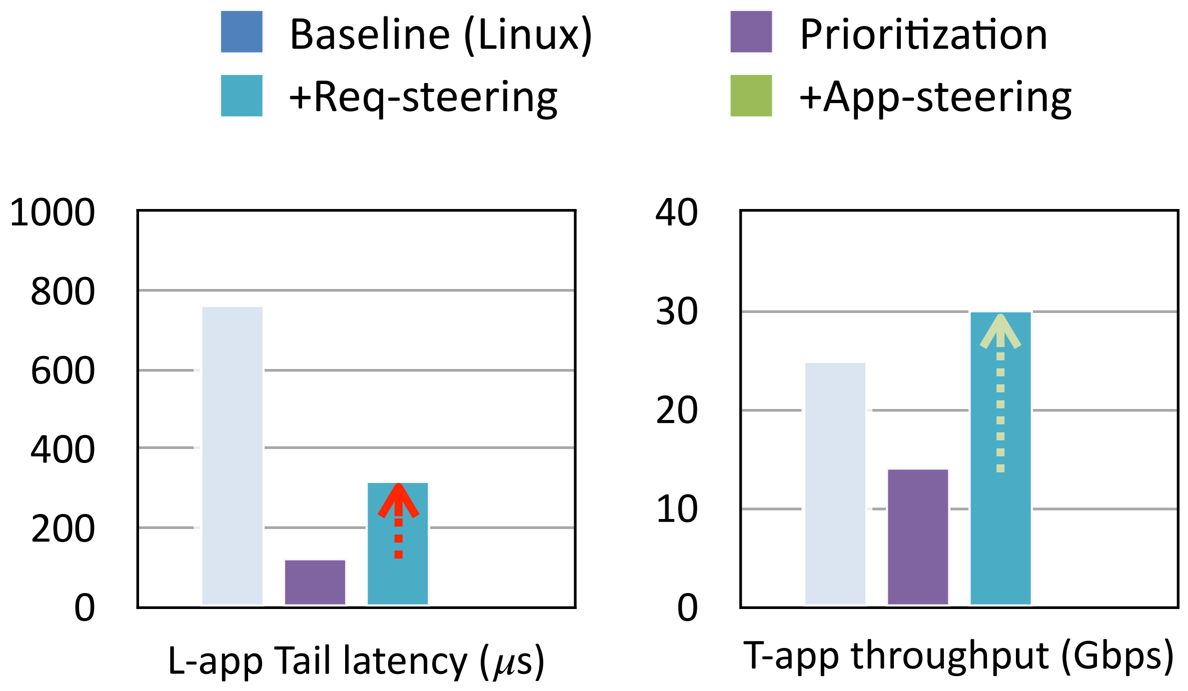
blk-switch Performance Breakdown



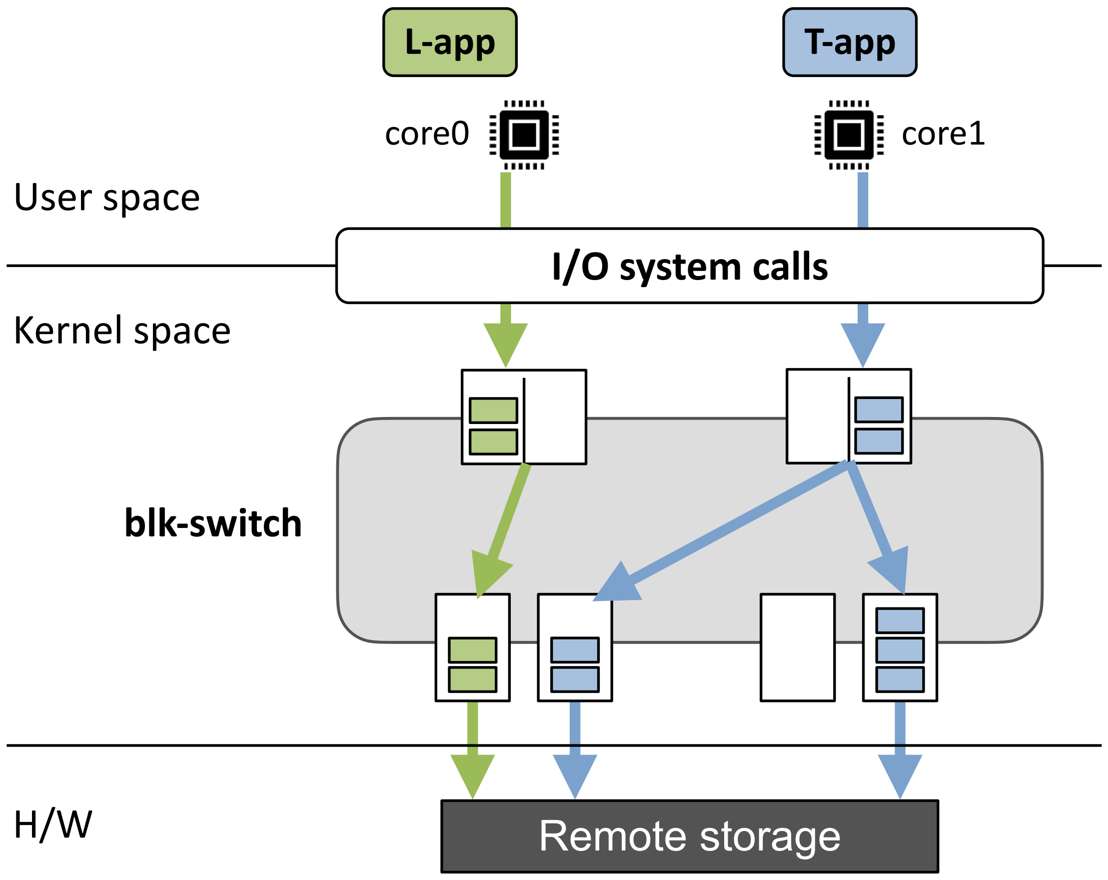


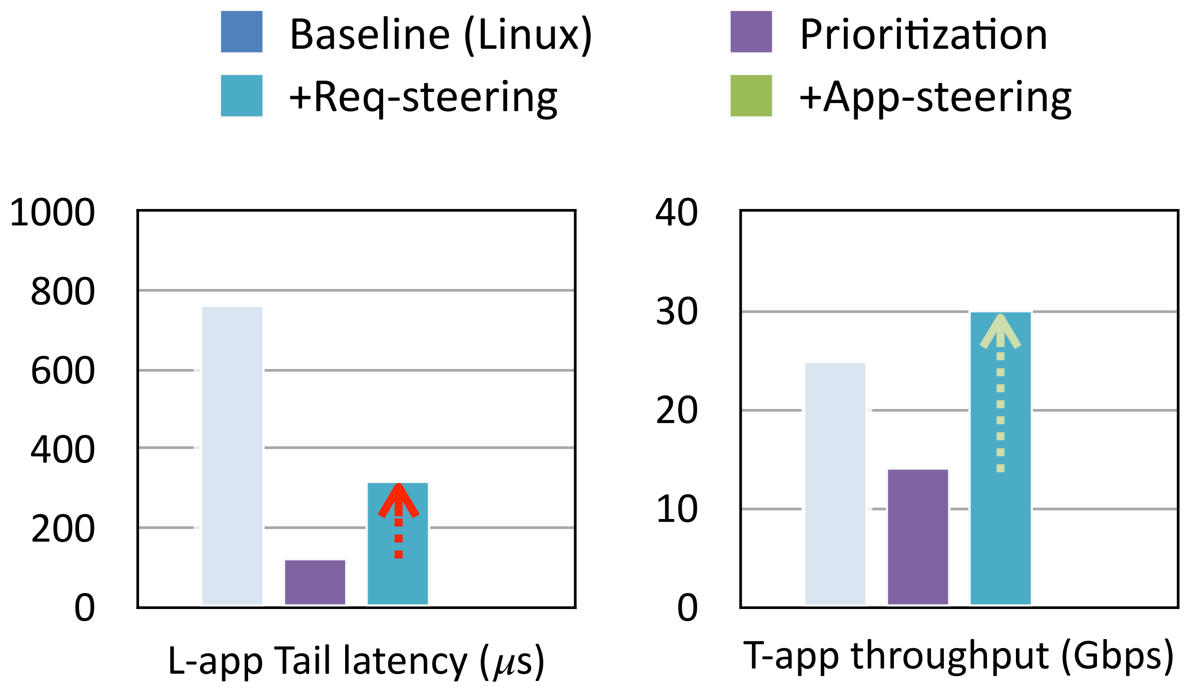
blk-switch Performance Breakdown



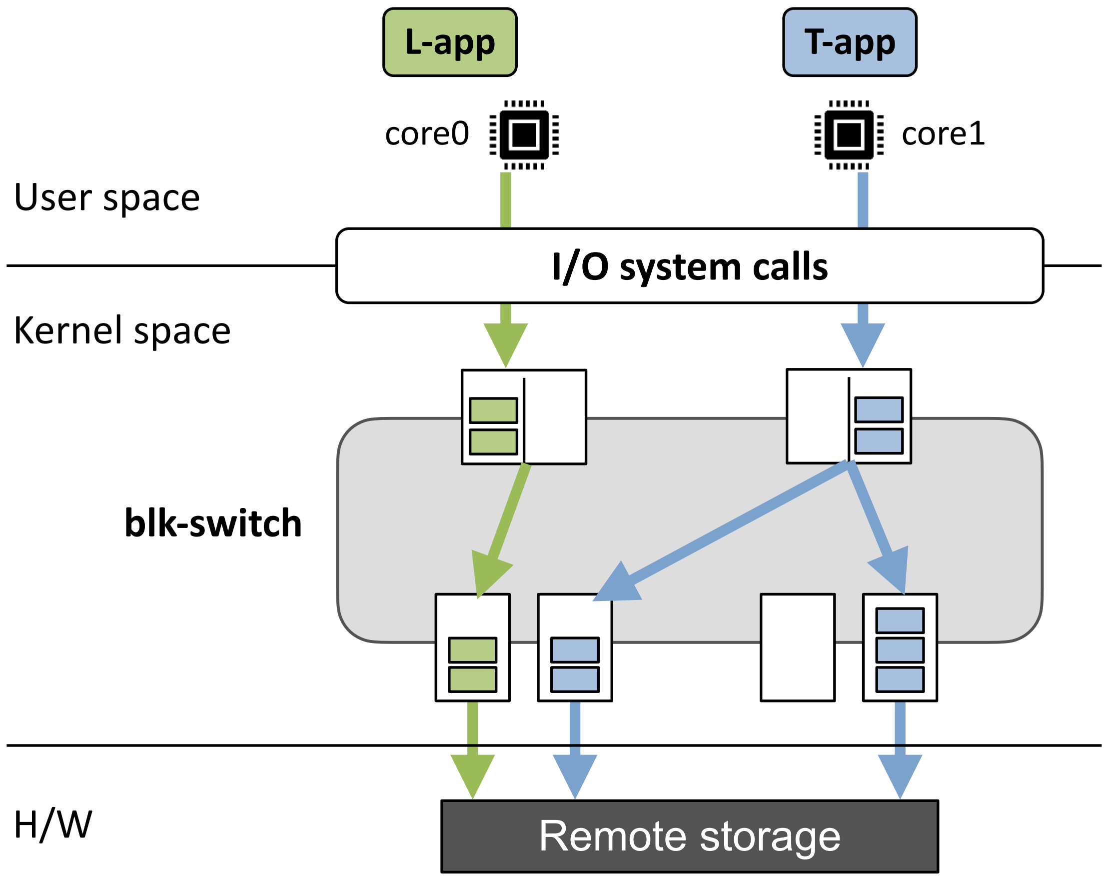


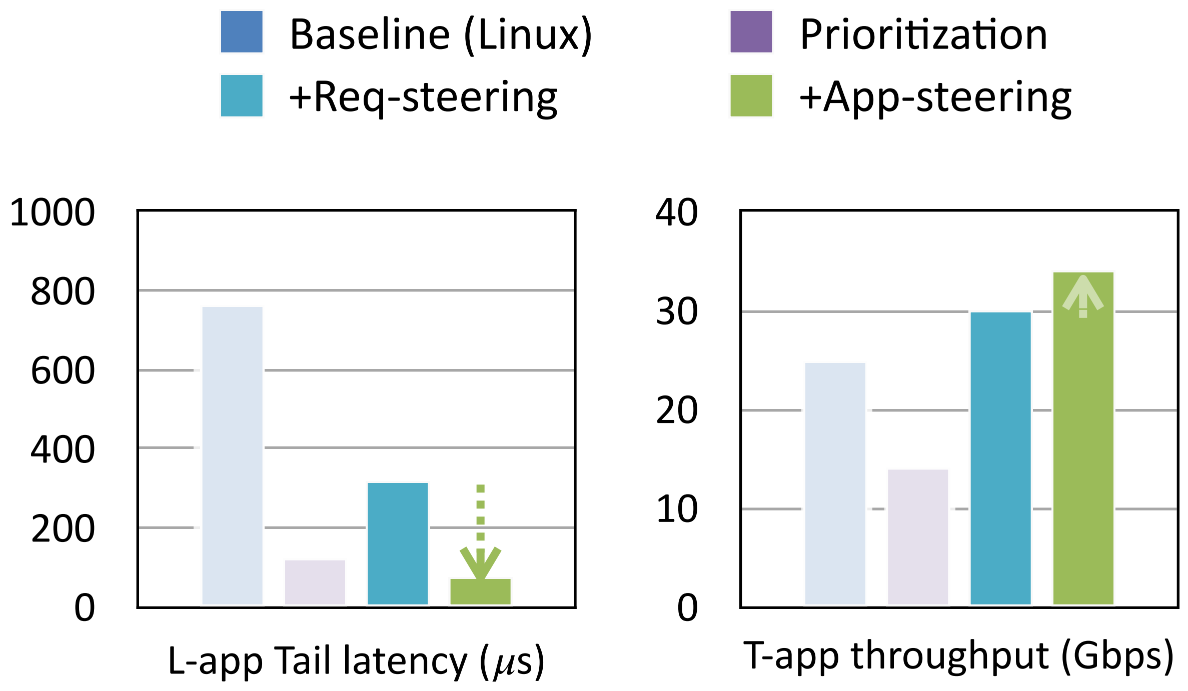
blk-switch Performance Breakdown



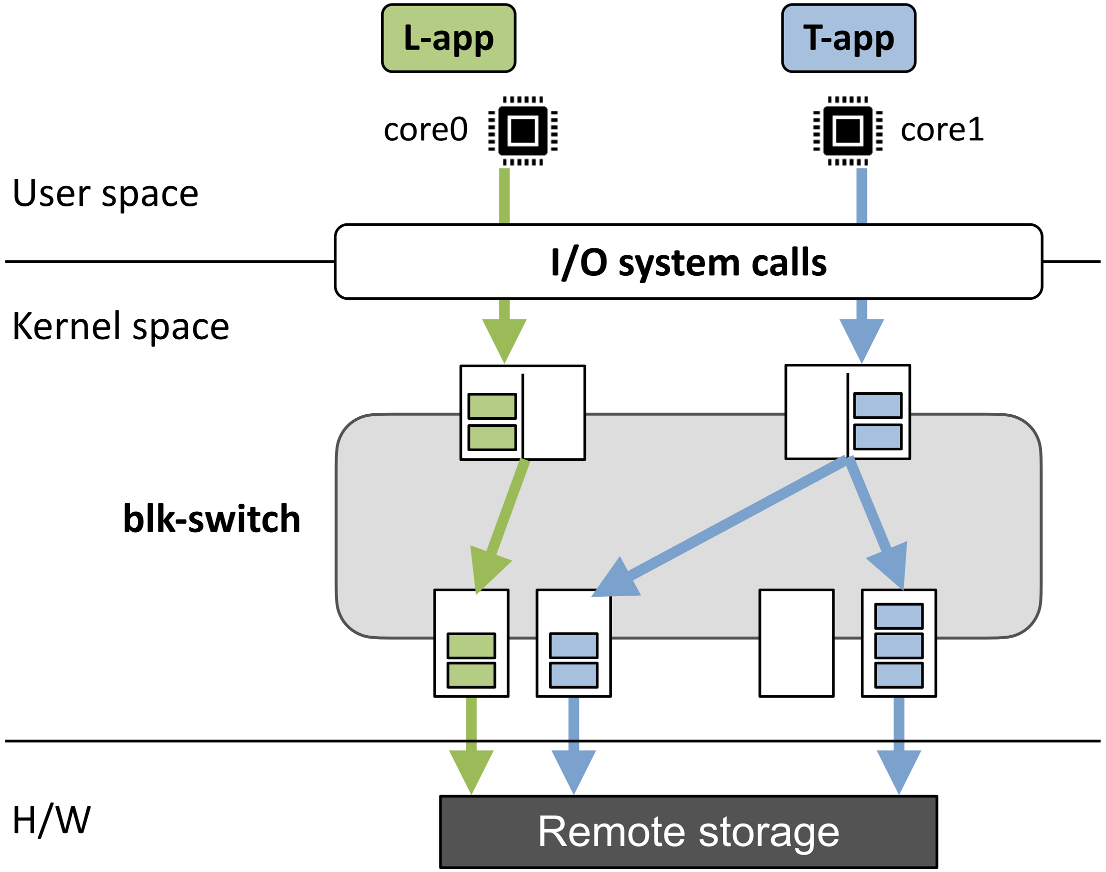


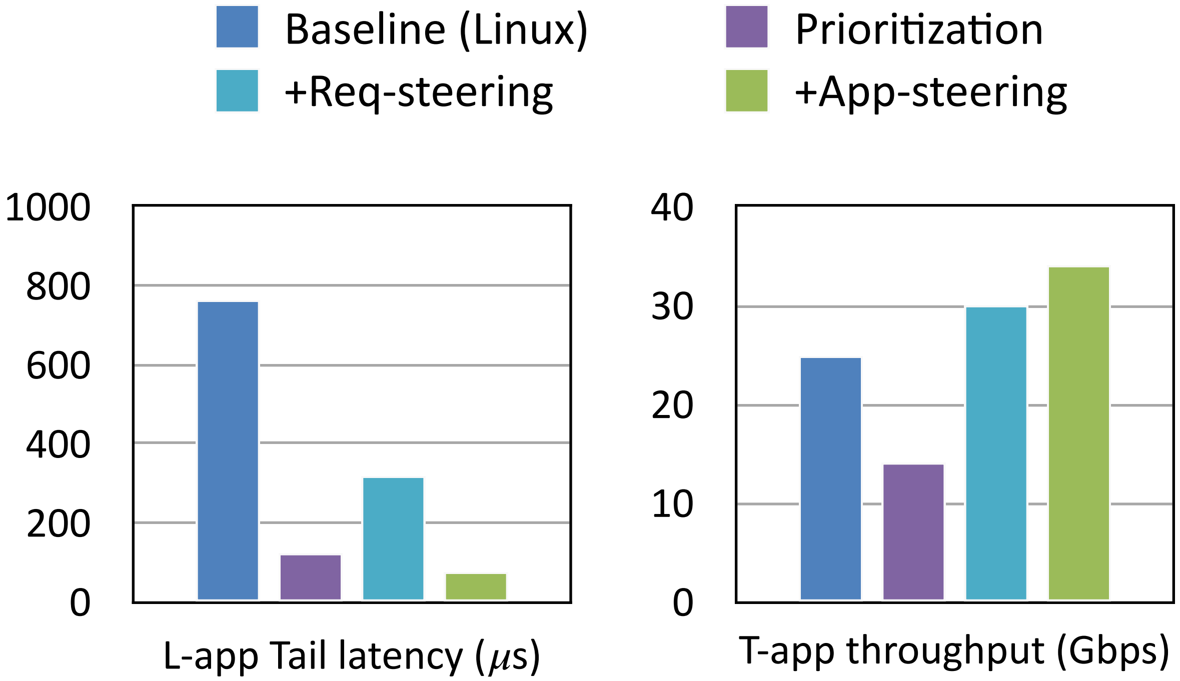
blk-switch Performance Breakdown





blk-switch Performance Breakdown





All design components contribute to achieving μs-scale latency and high throughput

Many more evaluation results in the paper

- **Performance** under different workloads, hardware, and applications
  - Number of L-apps
  - I/O depth of T-apps
  - Single-threaded vs. multi-threaded
  - Storage device access latency
  - Real applications
  - Request size of T-apps
  - Read/write ratios
  - …
- **Performance scalability with number of cores**
- **Performance scalability beyond 100Gbps**

Summary

Summary

- It is possible to achieve **μs-scale latency** and **high throughput** with Linux

Summary

- It is possible to achieve **μs-scale latency** and **high throughput** with Linux
- **blk-switch insight:** Modern storage stack is conceptually similar to network switches
  - Decoupling request processing from application cores
  - Multi-egress queue architecture, prioritization, request steering, and application steering

Summary

- It is possible to achieve **μs-scale latency** and **high throughput** with Linux
- **blk-switch insight:** Modern storage stack is conceptually similar to network switches
  - Decoupling request processing from application cores
  - Multi-egress queue architecture, prioritization, request steering, and application steering
- **blk-switch achieves:**
  - 10s of μs average latency and <190μs tail latency with in-memory storage
  - Near-hardware capacity throughput

Summary

- It is possible to achieve **μs-scale latency** and **high throughput** with Linux
- **blk-switch insight:** Modern storage stack is conceptually similar to network switches
  - Decoupling request processing from application cores
  - Multi-egress queue architecture, prioritization, request steering, and application steering
- **blk-switch achieves:**
  - 10s of μs average latency and <190μs tail latency with in-memory storage
  - Near-hardware capacity throughput

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

https://github.com/resource-disaggregation/blk-switch

Thank you!

Jaehyun Hwang
Cornell University
(jaehyun.hwang@cornell.edu)

Midhul Vuppalapati
Cornell University
(midhul@cs.cornell.edu)

Simon Peter
UT Austin
(simon@cs.utexas.edu)

Rachit Agarwal
Cornell University
(ragarwal@cs.cornell.edu)