Daredevil: Rescue Your Flash Storage from Inflexible Kernel Storage Stack

Junzhe Li, *Ran Shu, Jiayi Lin, Qingyu Zhang, Ziyue Yang, Jie Zhang,*

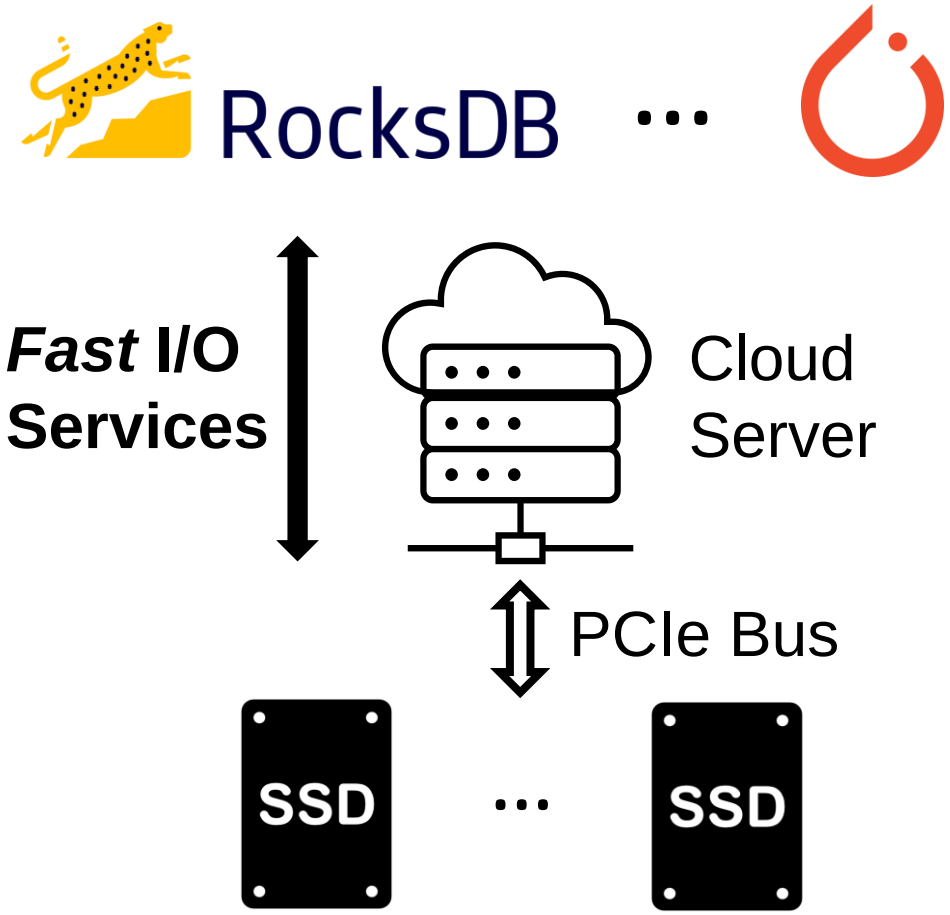
Yongqiang Xiong, Chenxiong Qian

香 港 大 學
THE UNIVERSITY OF HONG KONG

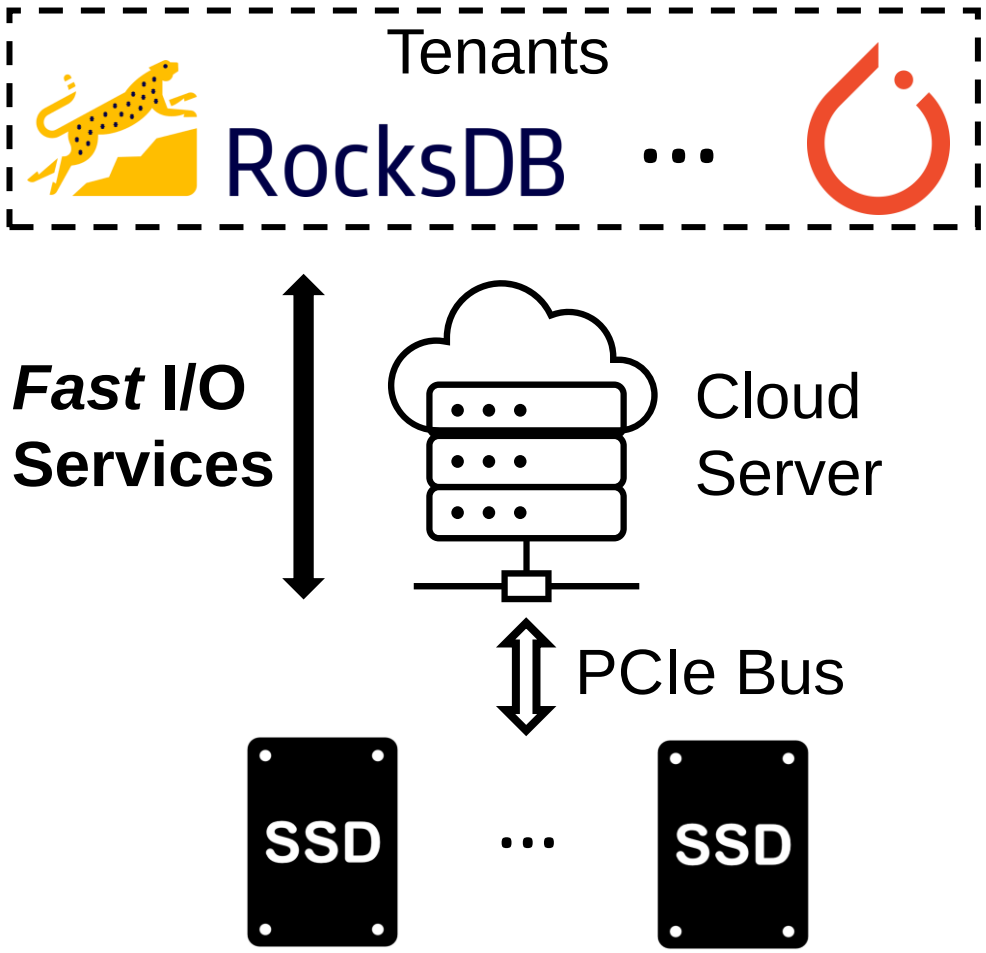
Microsoft

北京大學
PEKING UNIVERSITY

I/O Services: In the Face of Diversity

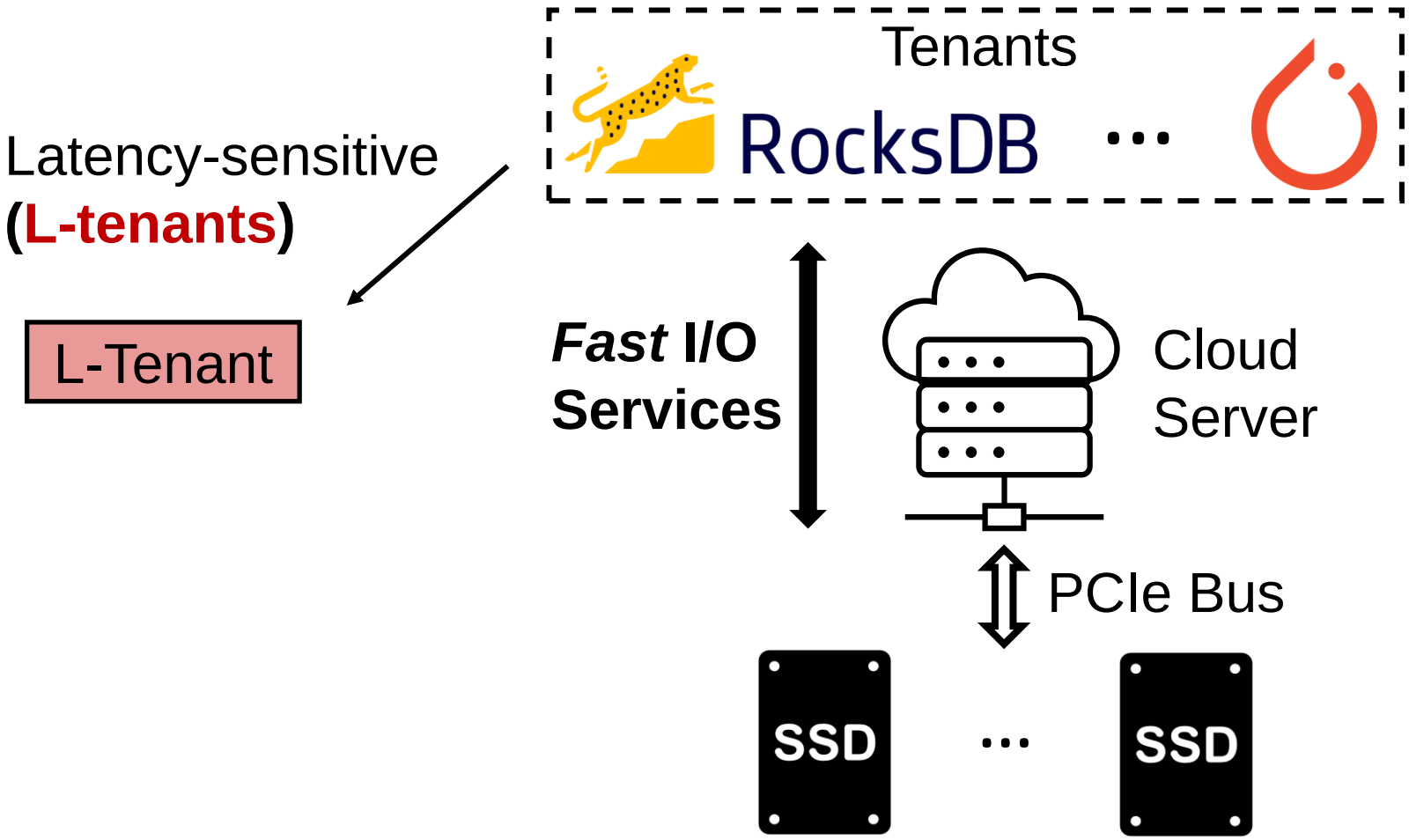


I/O Services: In the Face of Diversity



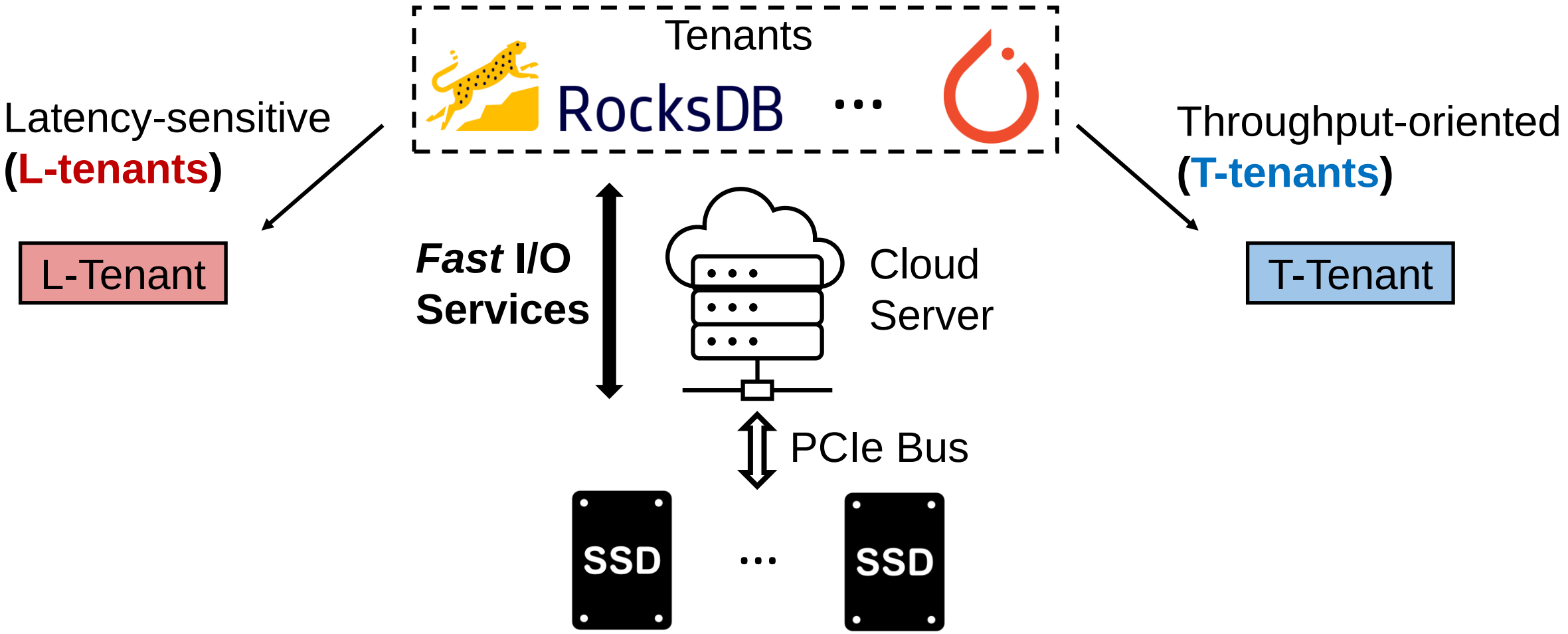
* **Processes** that require I/O services (e.g., containers) are referred to as "***tenants***".

I/O Services: In the Face of Diversity



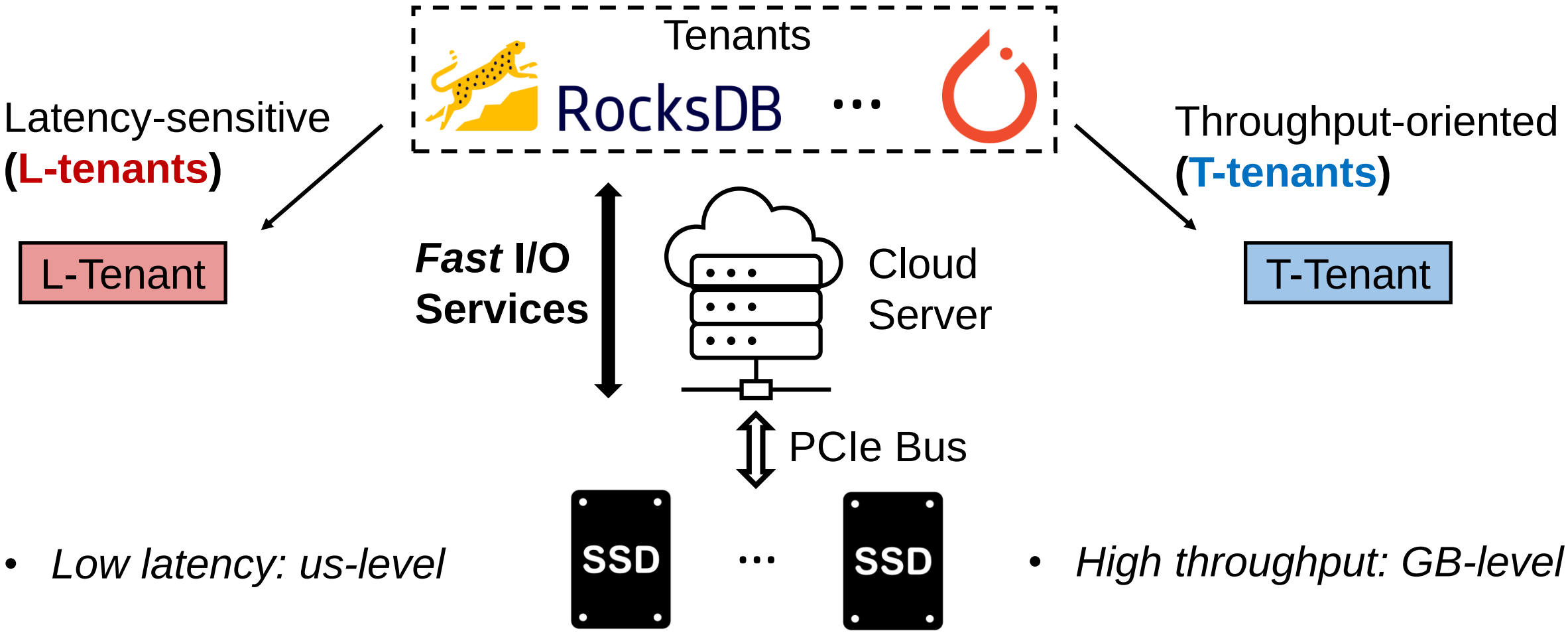
* **Processes** that require I/O services (e.g., containers) are referred to as "***tenants***".

I/O Services: In the Face of Diversity



* **Processes** that require I/O services (e.g., containers) are referred to as "***tenants***".

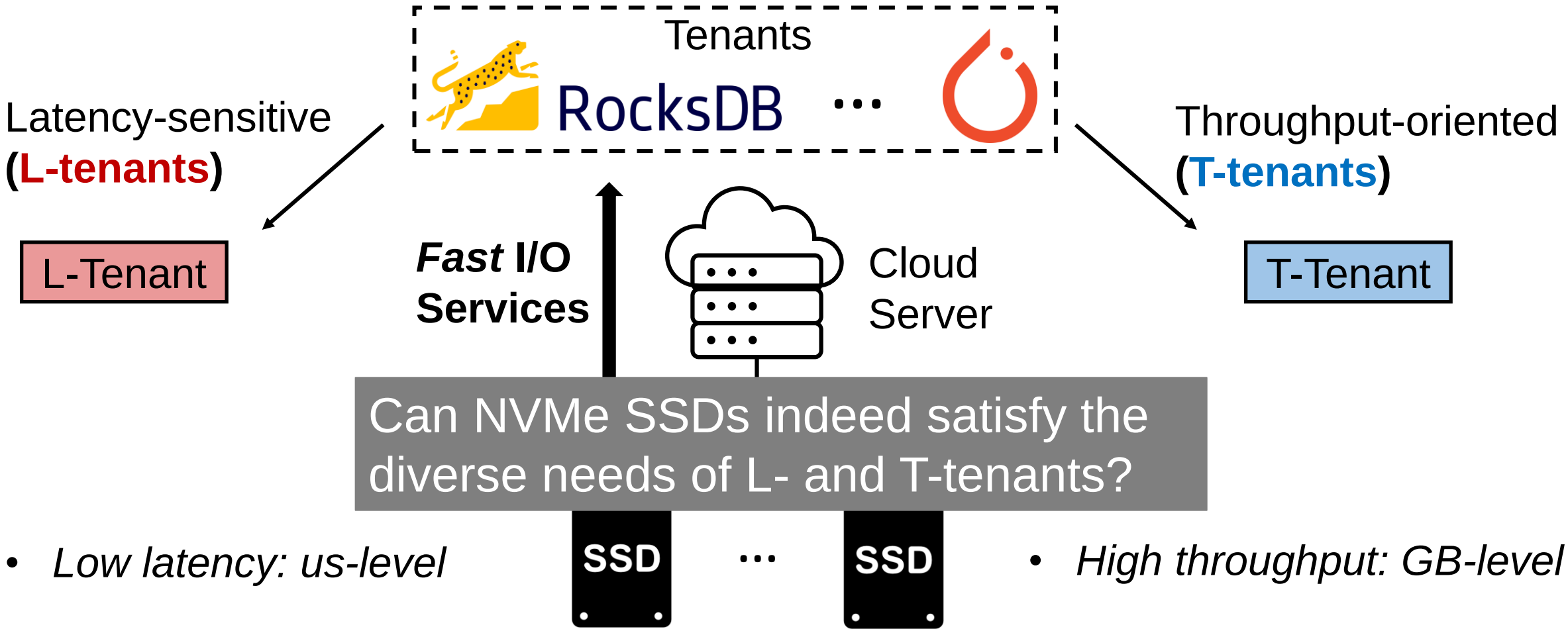
I/O Services: In the Face of Diversity



- *Low latency: us-level*
- *High throughput: GB-level*

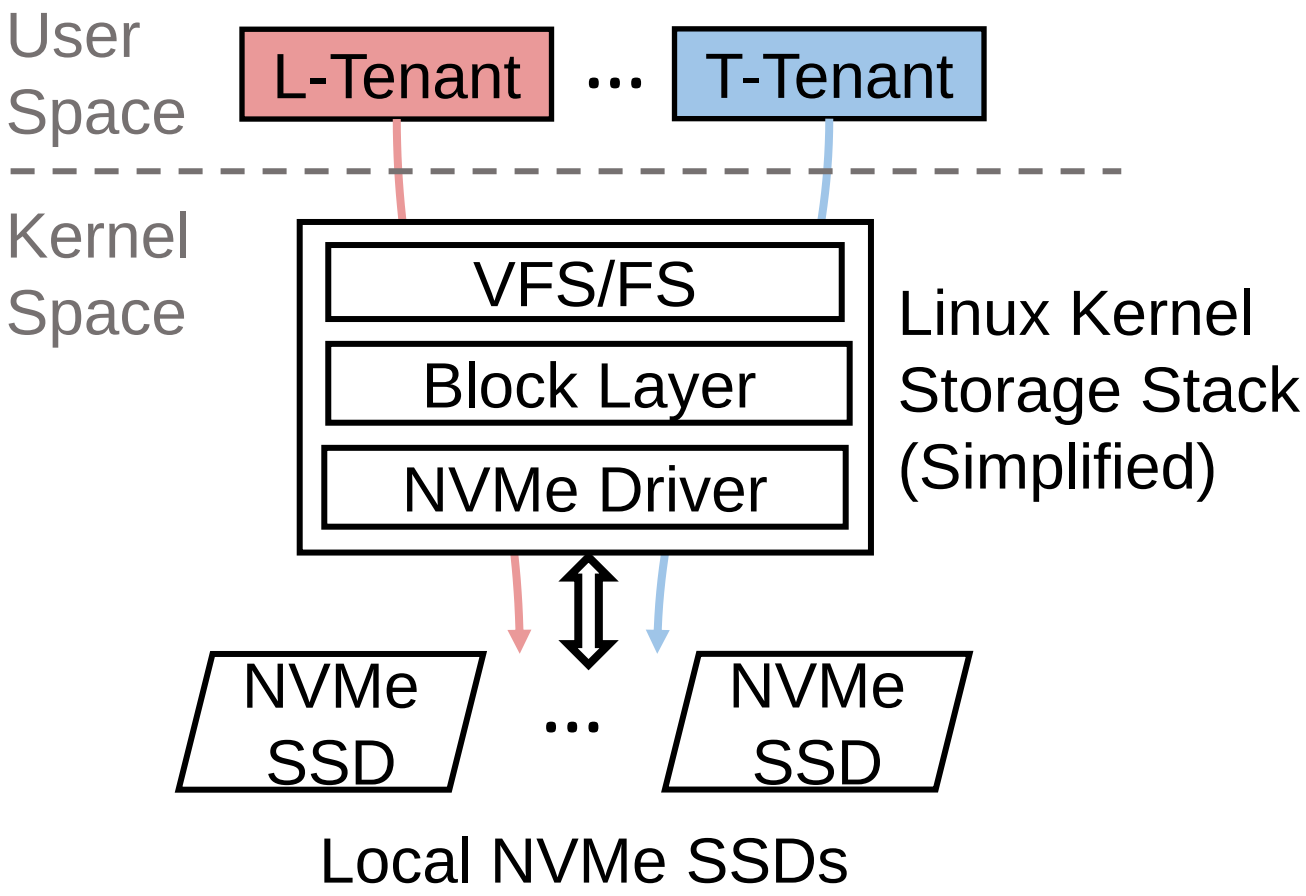
* **Processes** that require I/O services (e.g., containers) are referred to as "***tenants***".

I/O Services: In the Face of Diversity

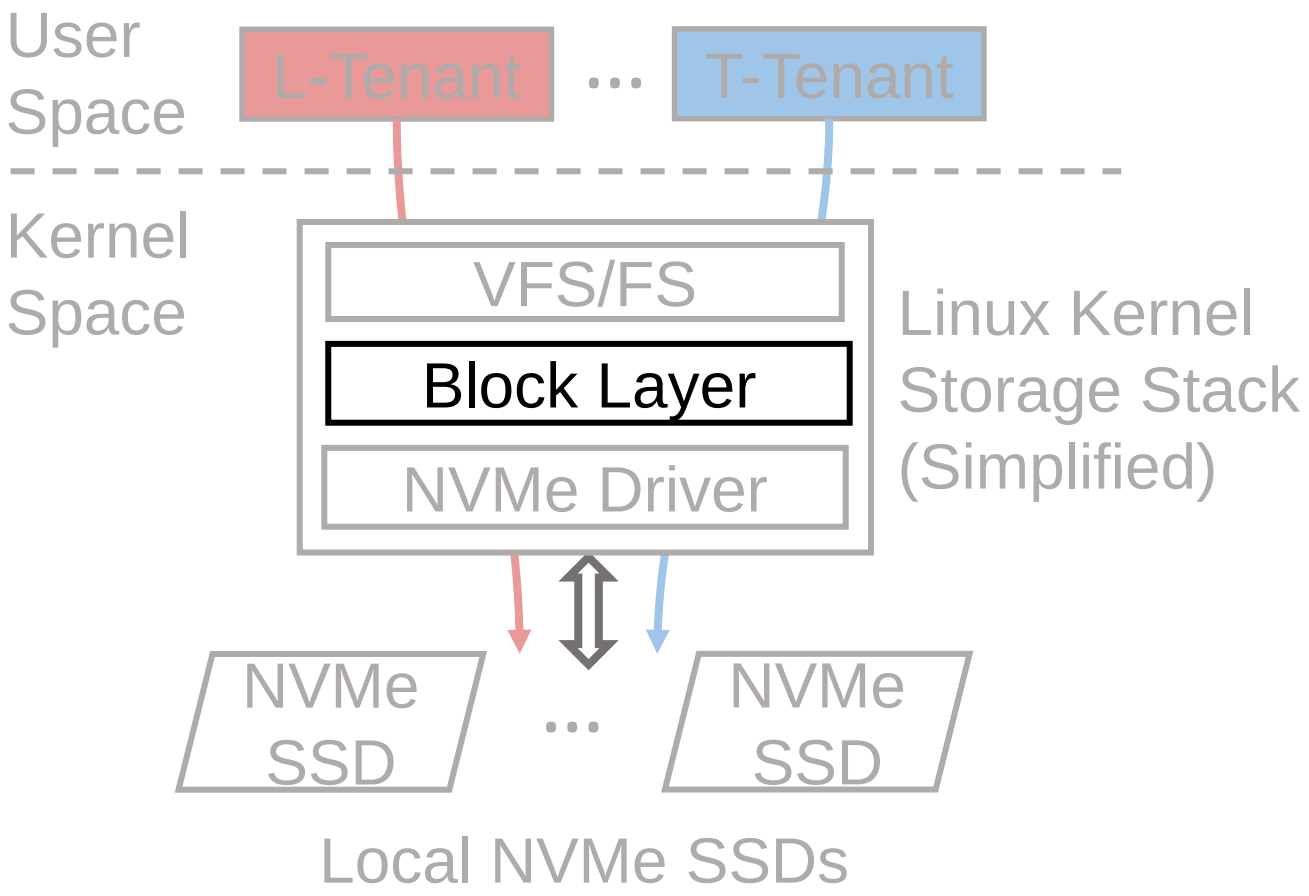


- *Low latency: us-level*
- *High throughput: GB-level*

I/O Services: In the Eyes of the Kernel



I/O Services: In the Eyes of the Kernel



Block layer – Where generic I/O scheduling operations happen.

Kernel Storage Stack: *blk-mq* Structure

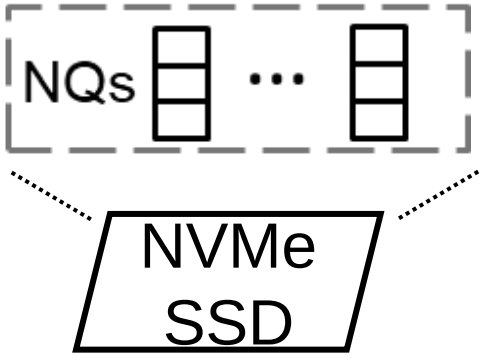
NVMe SSDs support multiple ***NVMe I/O queues (NQs)***

- Used for kernel-SSD I/O interactions
- Parallel access supported

Kernel Storage Stack: *blk-mq* Structure

NVMe SSDs support multiple ***NVMe I/O queues (NQs)***

- Used for kernel-SSD I/O interactions
- Parallel access supported

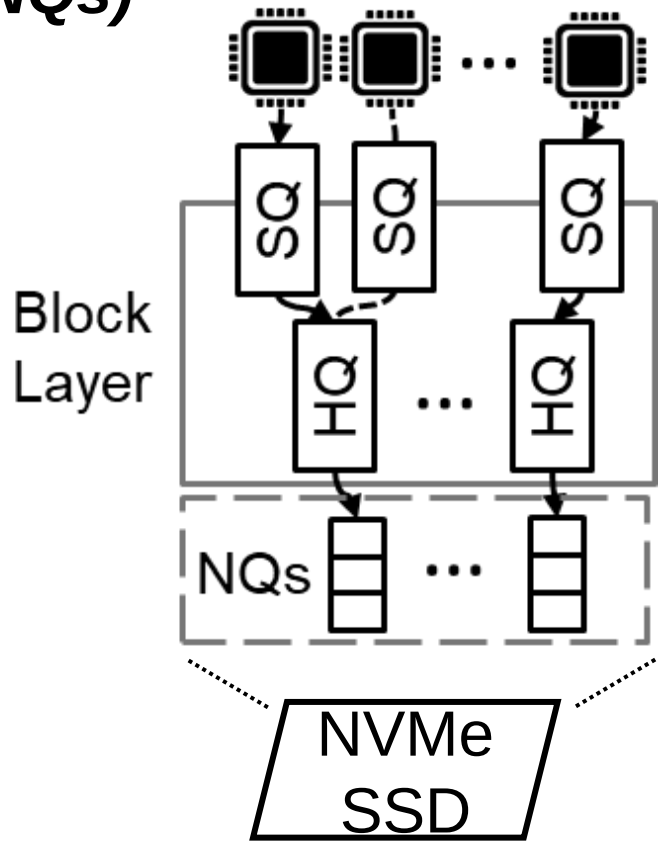


Kernel Storage Stack: *blk-mq* Structure

NVMe SSDs support multiple ***NVMe I/O queues (NQs)***

- Used for kernel-SSD I/O interactions
- Parallel access supported

Multi-Queue Block IO Queueing Mechanism (*blk-mq)*:



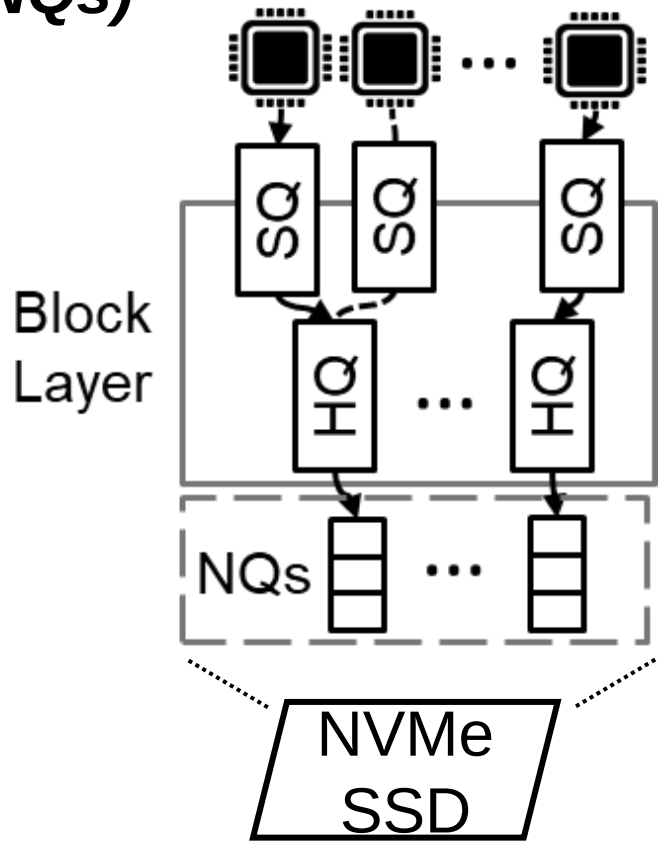
Kernel Storage Stack: *blk-mq* Structure

NVMe SSDs support multiple ***NVMe I/O queues (NQs)***

- Used for kernel-SSD I/O interactions
- Parallel access supported

Multi-Queue Block IO Queueing Mechanism (*blk-mq)*:

- Software queues (**SQs**): one per CPU core



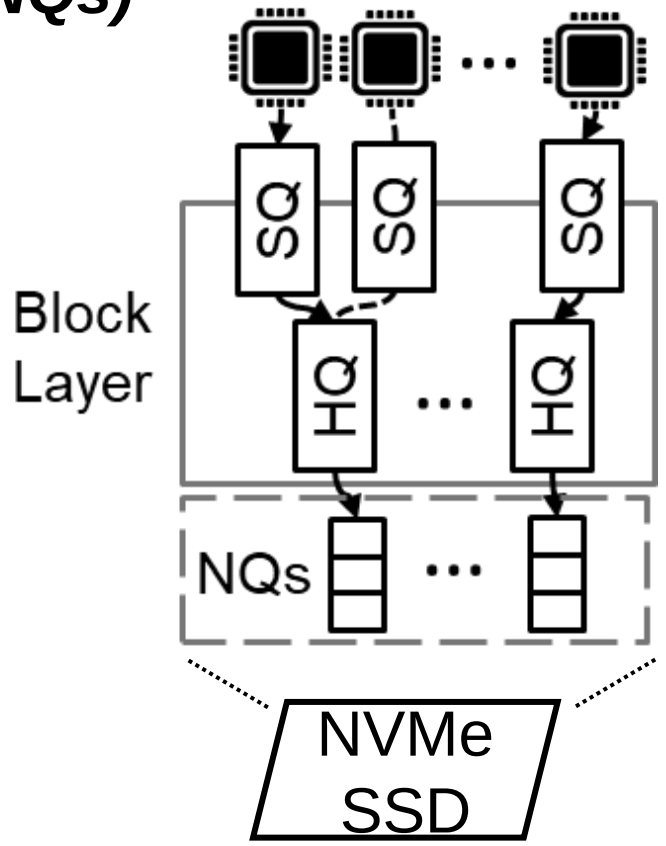
Kernel Storage Stack: *blk-mq* Structure

NVMe SSDs support multiple ***NVMe I/O queues (NQs)***

- Used for kernel-SSD I/O interactions
- Parallel access supported

Multi-Queue Block IO Queueing Mechanism (*blk-mq)*:

- Software queues (**SQs**): one per CPU core
- Hardware queues (**HQs**): one per NQ



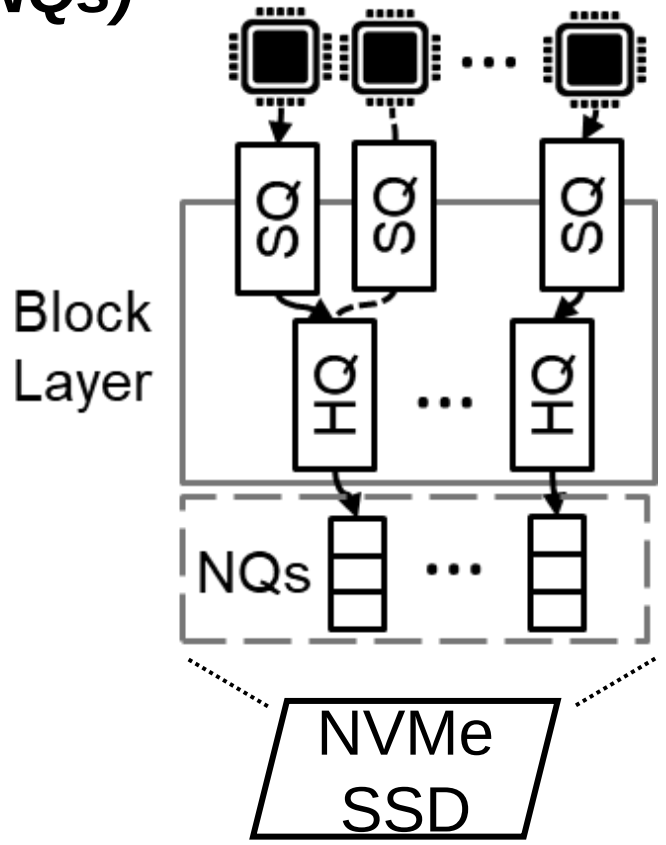
Kernel Storage Stack: *blk-mq* Structure

NVMe SSDs support multiple ***NVMe I/O queues (NQs)***

- Used for kernel-SSD I/O interactions
- Parallel access supported

Multi-Queue Block IO Queueing Mechanism (*blk-mq)*:

- Software queues (**SQs**): one per CPU core
- Hardware queues (**HQs**): one per NQ
- **Static bindings** between SQs and HQs



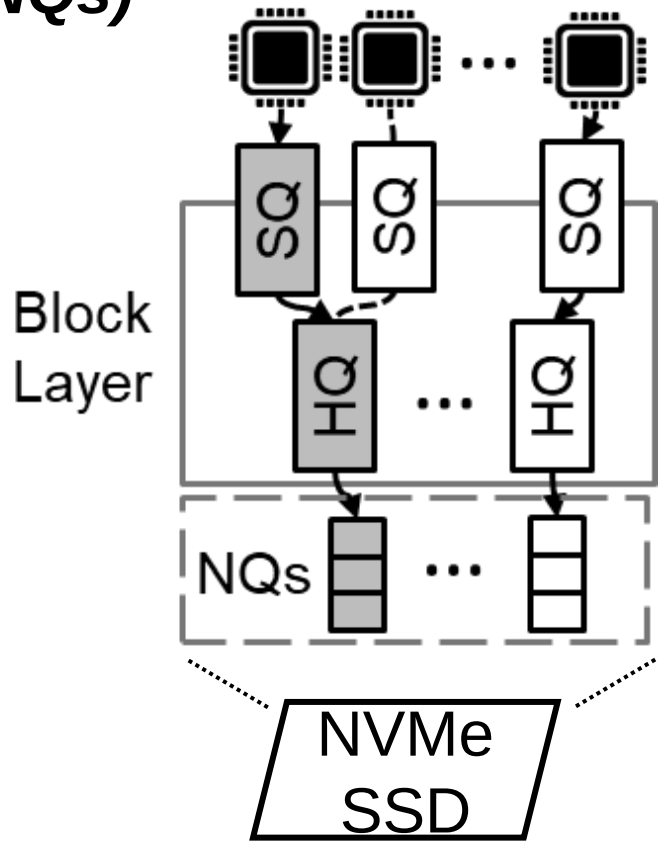
Kernel Storage Stack: *blk-mq* Structure

NVMe SSDs support multiple ***NVMe I/O queues (NQs)***

- Used for kernel-SSD I/O interactions
- Parallel access supported

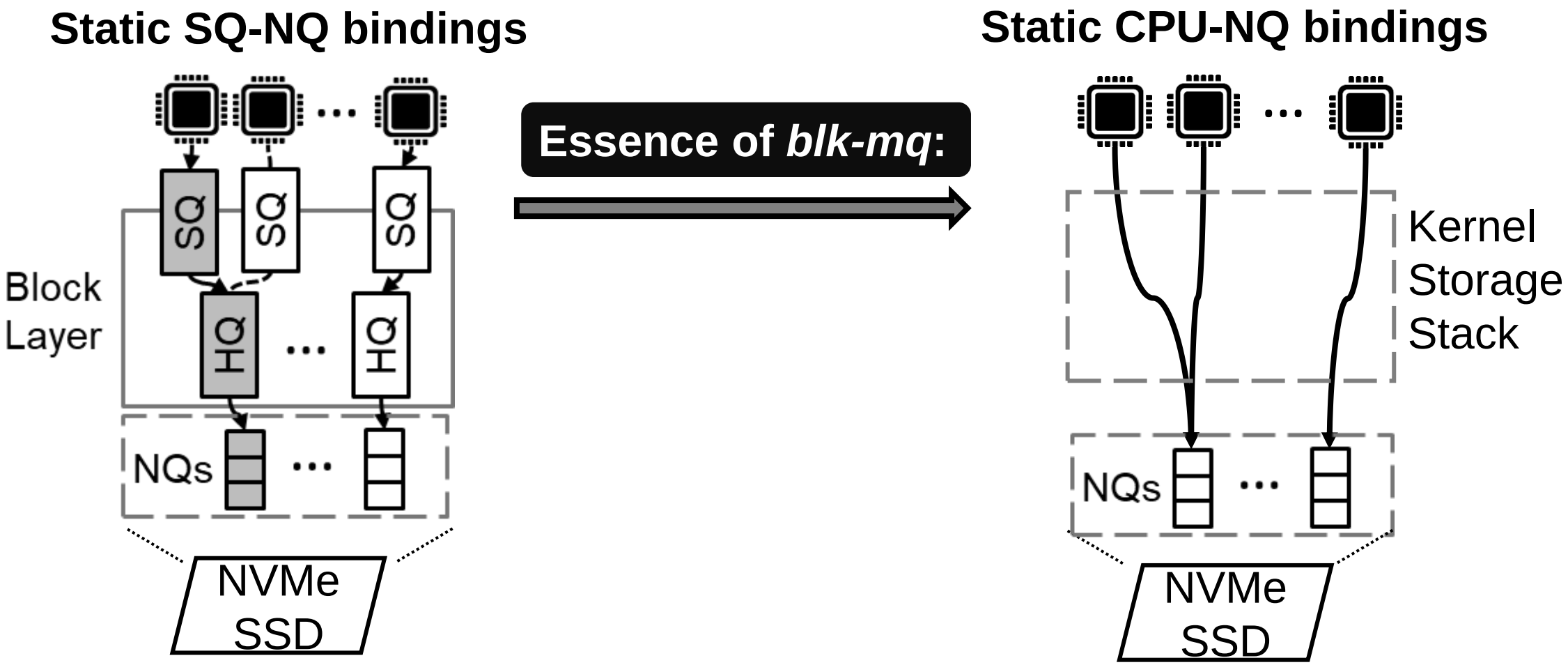
Multi-Queue Block IO Queueing Mechanism (*blk-mq)*:

- Software queues (**SQs**): one per CPU core
- Hardware queues (**HQs**): one per NQ
- **Static bindings** between SQs and HQs



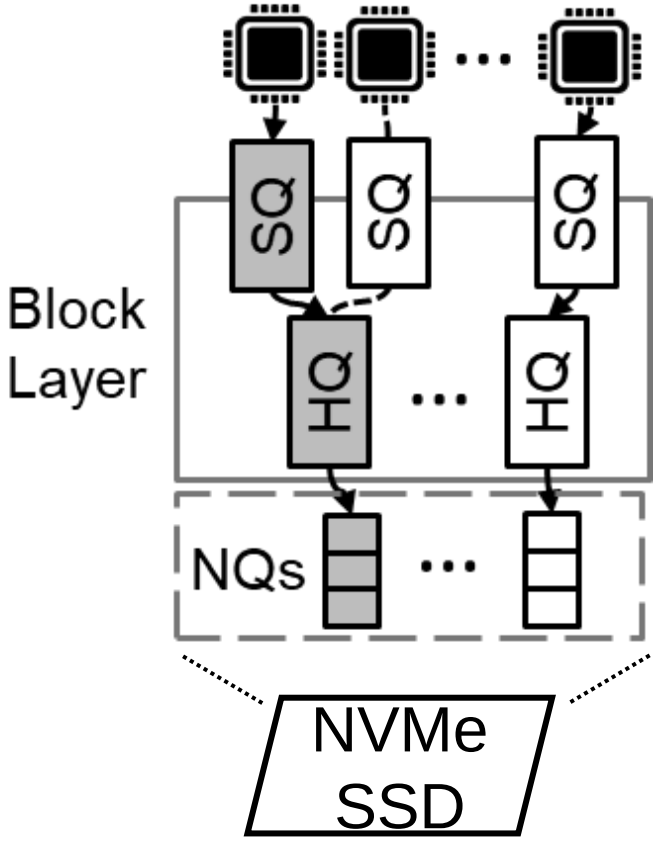
I/O flow: CPU core → SQ → HQ → NQ

Kernel Storage Stack: *blk-mq* Structure



Kernel Storage Stack: *blk-mq* Structure

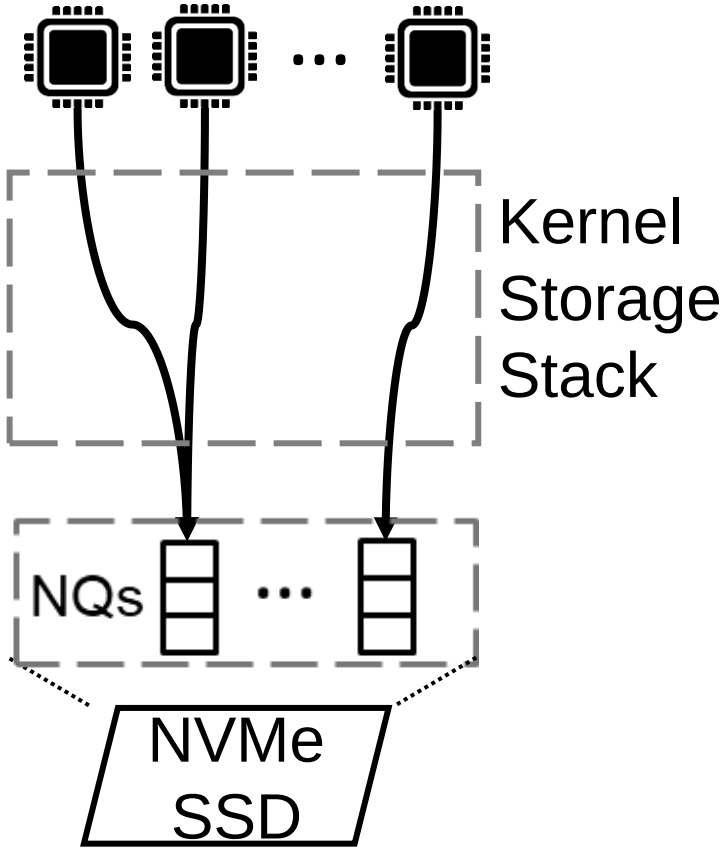
Static SQ-NQ bindings



Essence of *blk-mq*:

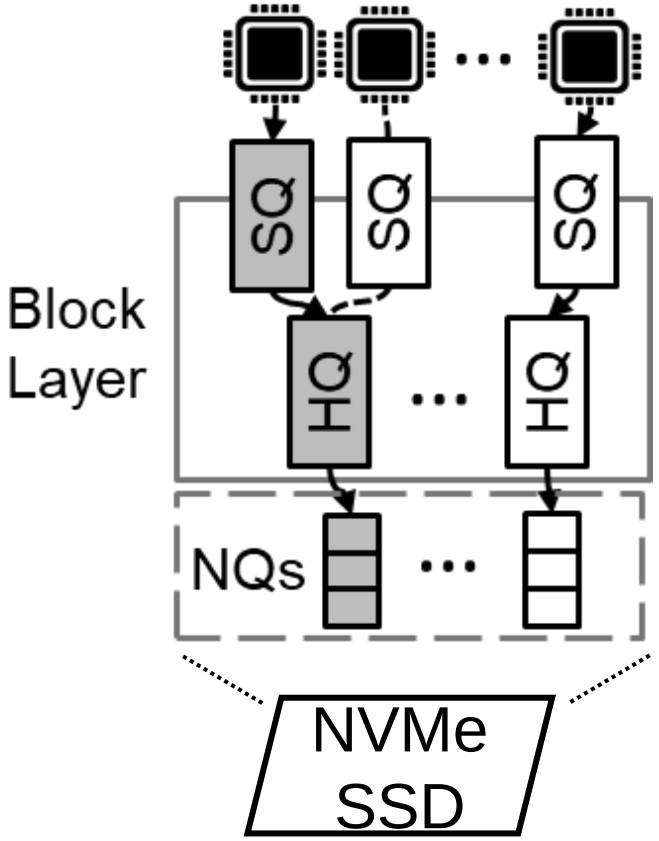
- Maintenance
- Parallelism/Concurrency

Static CPU-NQ bindings



Kernel Storage Stack: *blk-mq* Structure

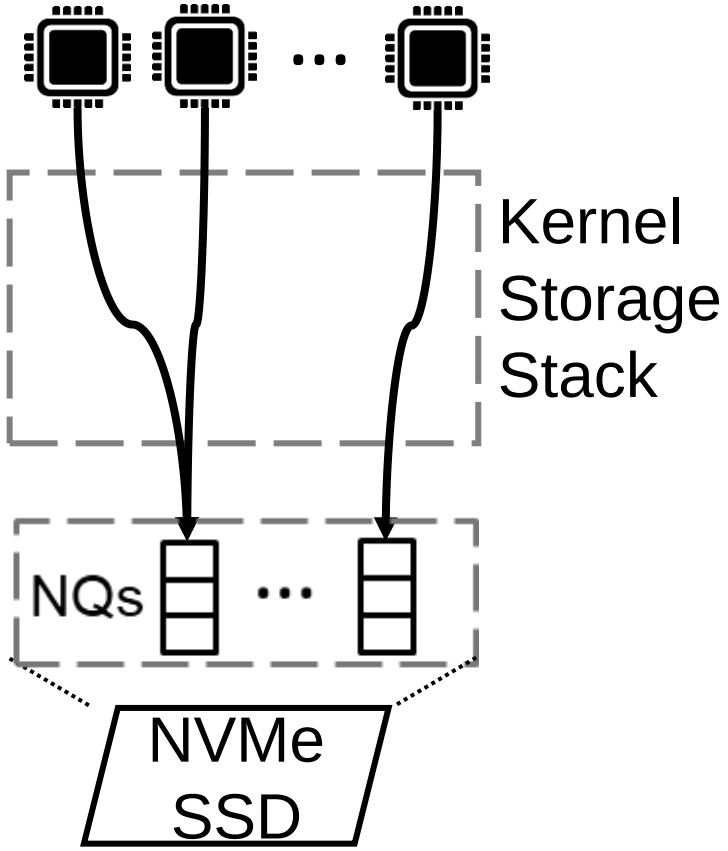
Static SQ-NQ bindings



Essence of *blk-mq*:

- Maintenance
- Parallelism/Concurrency
- **But…! *Troublesome*** in cloud servers

Static CPU-NQ bindings

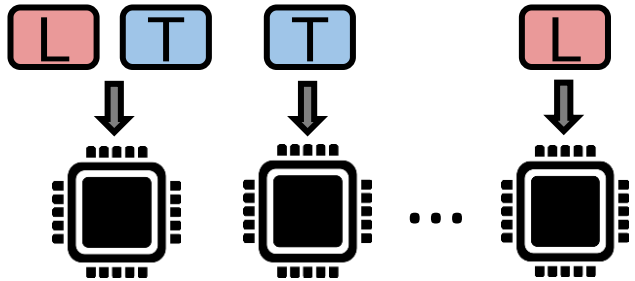


I/O Services: Within the *blk-mq* Structure

T : Throughput-oriented tenants (T-tenant)

L : Latency-sensitive tenants (L-tenant)

Common CPU **sharing among L- and T-tenants**.

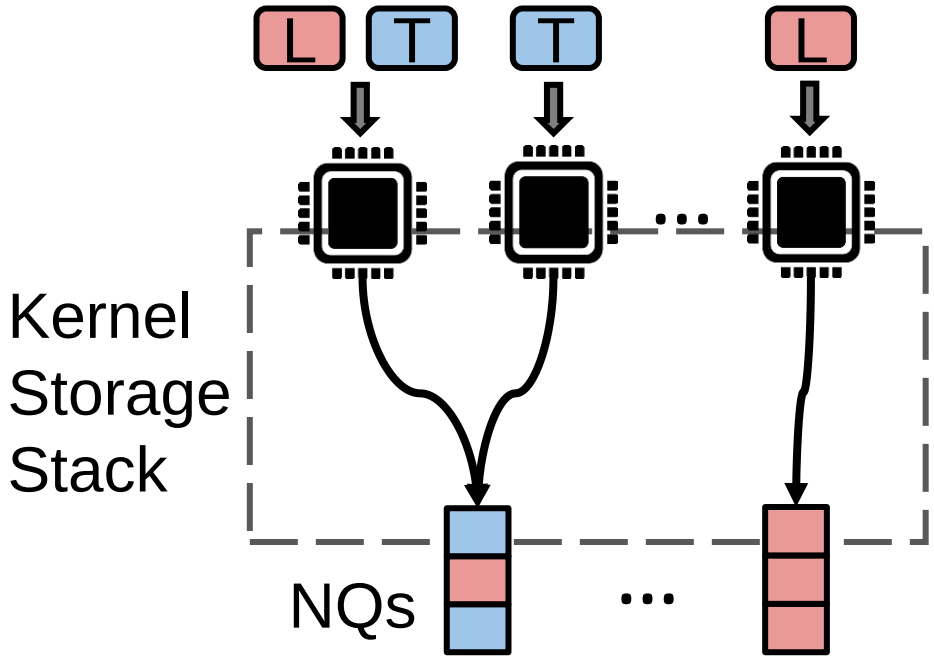


I/O Services: Within the *blk-mq* Structure

T : Throughput-oriented tenants (T-tenant)

L : Latency-sensitive tenants (L-tenant)

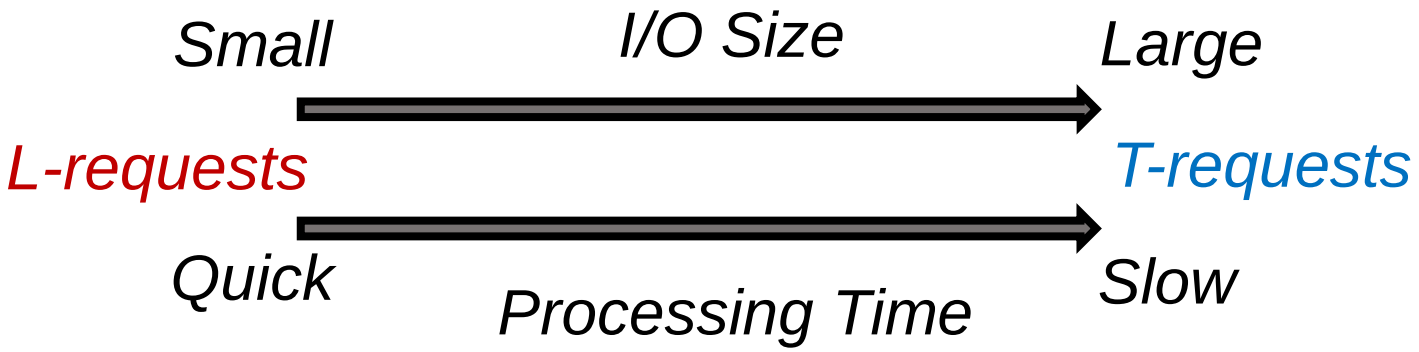
Common CPU **sharing among L- and T-tenants**.



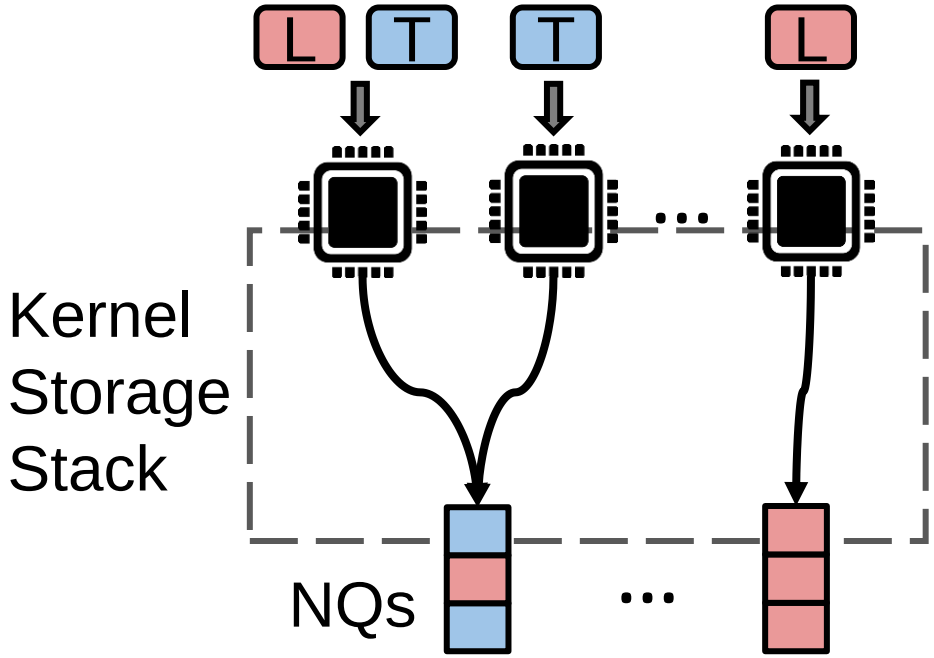
I/O Services: Within the *blk-mq* Structure

T : Throughput-oriented tenants (T-tenant)

L : Latency-sensitive tenants (L-tenant)



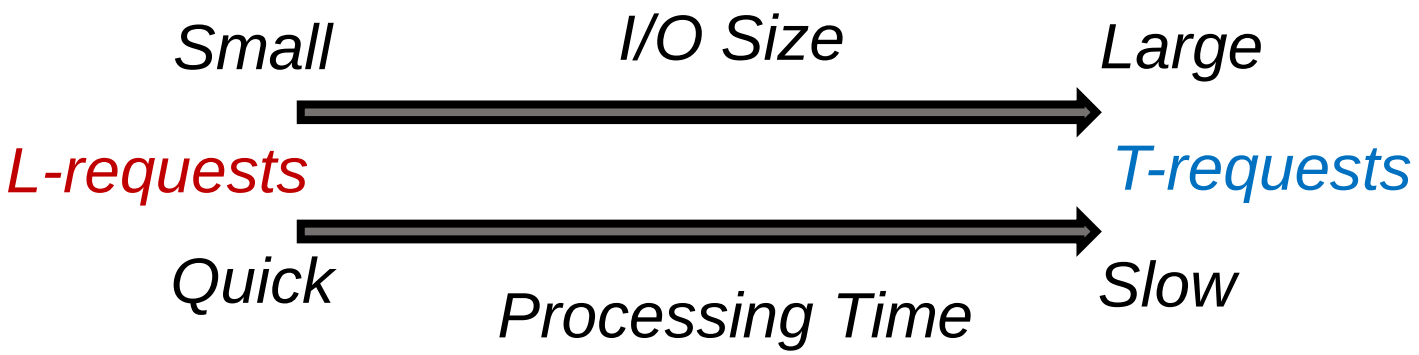
Common CPU **sharing among L- and T-tenants**.



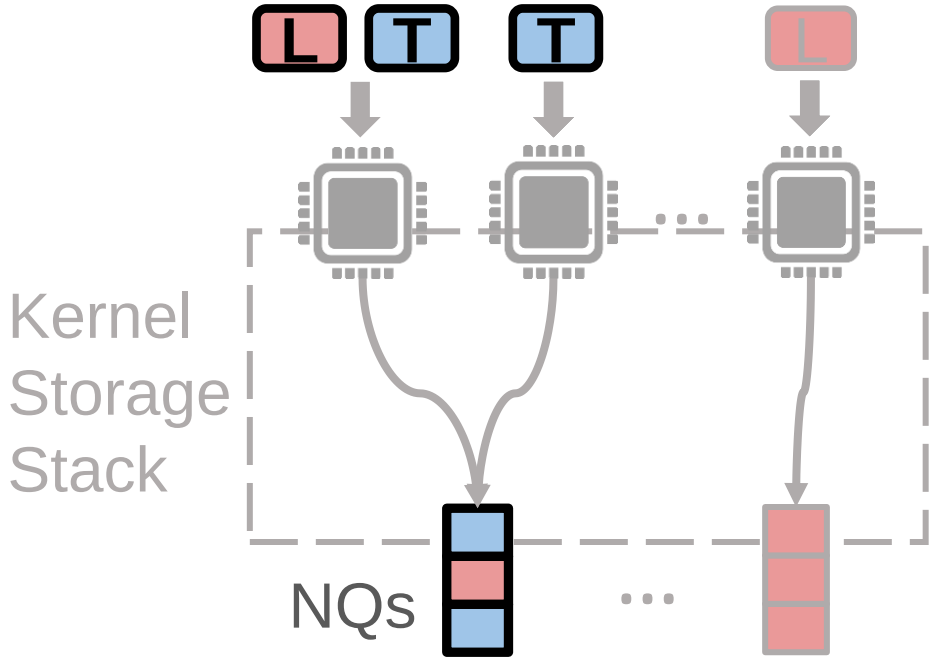
I/O Services: Within the *blk-mq* Structure

T : Throughput-oriented tenants (T-tenant)

L : Latency-sensitive tenants (L-tenant)



Common CPU **sharing among L- and T-tenants**.



Head-of-line (HOL) blocking from T-requests!

I/O Services: Evidencing HOL Blocking

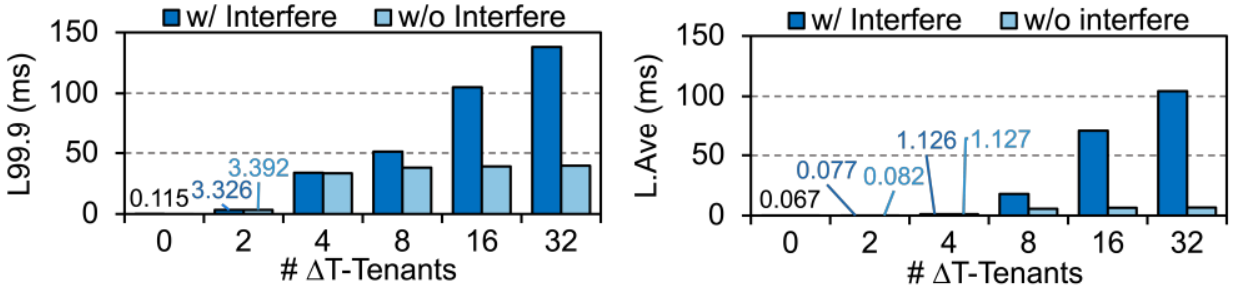
Experiment:

- w/ Interfere: L- and T-tenants served **within *the same NQs*.**
- w/o Interfere: L- and T-tenants served **by *separate NQs*.**

I/O Services: Evidencing HOL Blocking

Experiment:

- w/ Interfere: L- and T-tenants served **within *the same NQs*.**
- w/o Interfere: L- and T-tenants served **by *separate NQs*.**



(a) L-tenant 99.9^{th} tail latency. **(b)** L-tenant average latency.

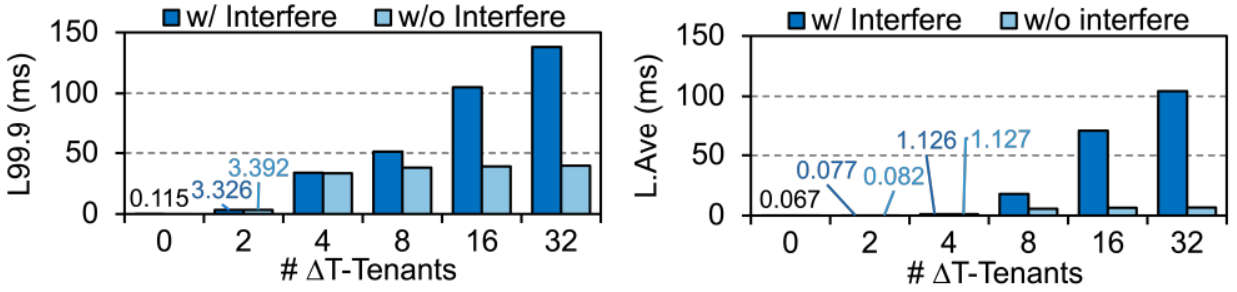
Figure 2. I/O latency of L-tenants with T-tenants interfering within the same NQs (*w/ Interfere*) and using separate NQs (*w/o Interfere*).

3x/15x increase in tail/average latency with HOL T-requests.

I/O Services: Evidencing HOL Blocking

Experiment:

- w/ Interfere: L- and T-tenants served **within *the same NQs*.**
- w/o Interfere: L- and T-tenants served **by *separate NQs*.**



(a) L-tenant 99.9^{th} tail latency. **(b)** L-tenant average latency.

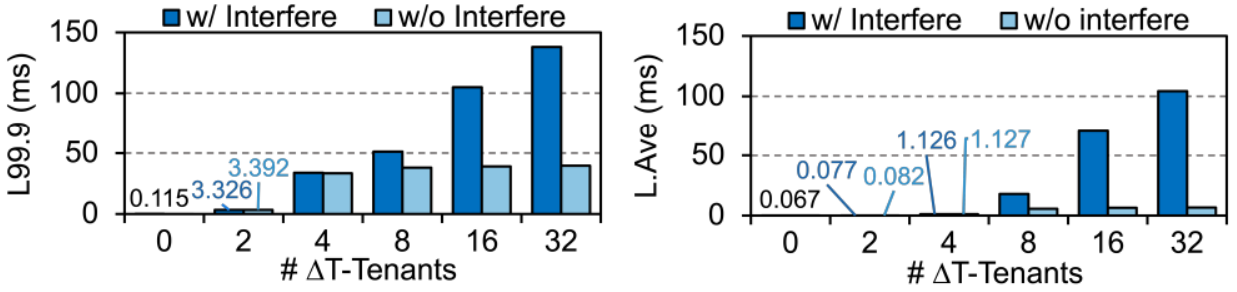
Figure 2. I/O latency of L-tenants with T-tenants interfering within the same NQs (*w/ Interfere*) and using separate NQs (*w/o Interfere*).

The multi-tenancy issue: In NVMe SSDs, the performance of L-requests can be severely impacted by the HOL T-requests within the same NQs.

I/O Services: Evidencing HOL Blocking

Experiment:

- w/ Interfere: L- and T-tenants served **within *the same NQs*.**
- w/o Interfere: L- and T-tenants served **by *separate NQs*.**



(a) L-tenant 99.9^{th} tail latency. **(b)** L-tenant average latency.

Figure 2. I/O latency of L-tenants with T-tenants interfering within the same NQs (*w/ Interfere*) and using separate NQs (*w/o Interfere*).

The multi-tenancy issue: In NVMe SSDs, the performance of L-requests can be severely impacted by the HOL T-requests within the same NQs.

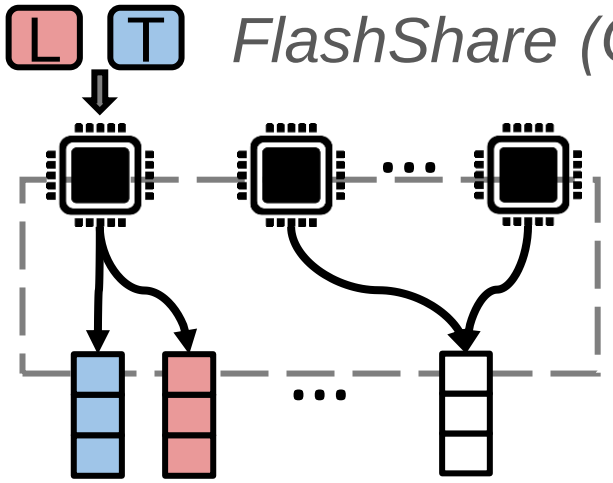
How can we solve this issue within the kernel storage stack?

Static *blk-mq*: Constrained Optimization

Clear solution: **NQ-level separation** of L- and T-requests.

Static *blk-mq*: Constrained Optimization

Clear solution: **NQ-level separation** of L- and T-requests.

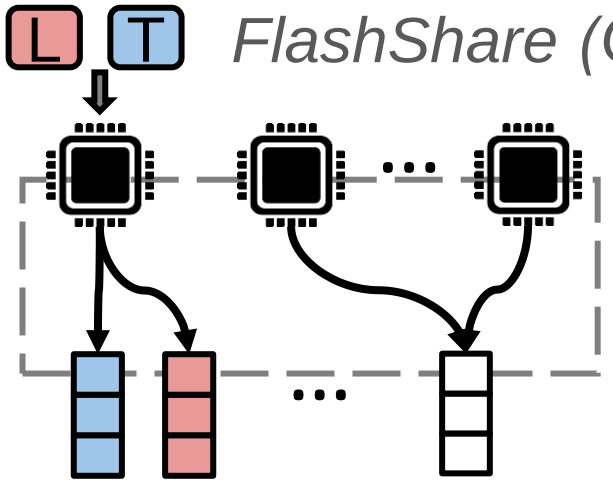


FlashShare (OSDI'18), D2FQ (FAST'21)

NQ overprovision:

Static *blk-mq*: Constrained Optimization

Clear solution: **NQ-level separation** of L- and T-requests.



FlashShare (OSDI'18), D2FQ (FAST'21)

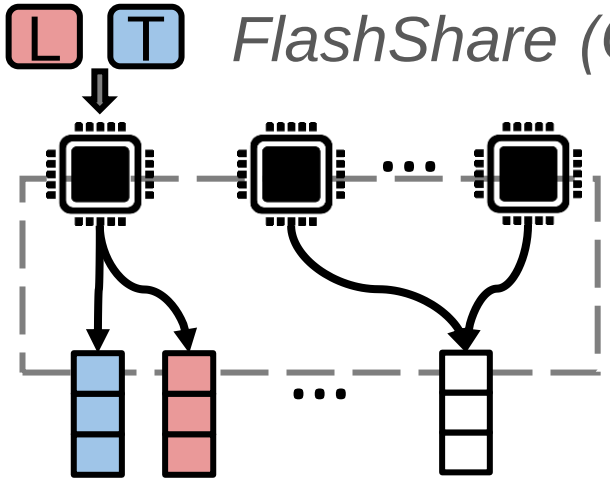
NQ overprovision:

😎 *Simple & Direct*

☹ *Underutilization & HW constraints*

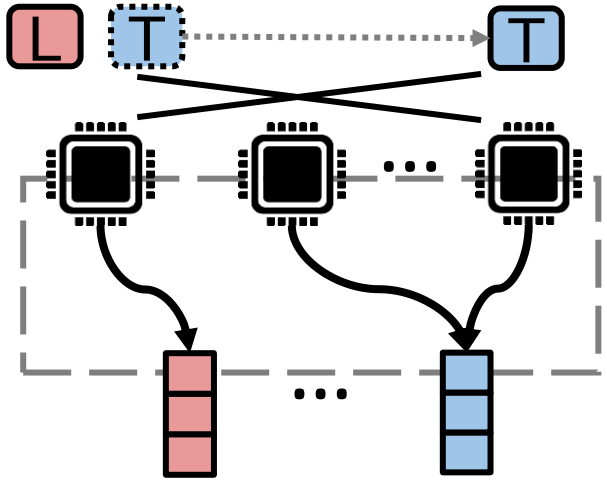
Static *blk-mq*: Constrained Optimization

Clear solution: **NQ-level separation** of L- and T-requests.



FlashShare (OSDI'18), D2FQ (FAST'21)

blk-switch (OSDI'21)



NQ overprovision:

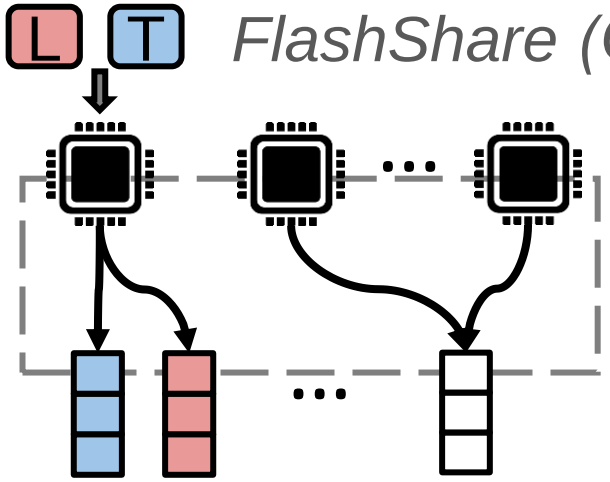
😎 *Simple & Direct*

☹ *Underutilization & HW constraints*

Cross-core scheduling:

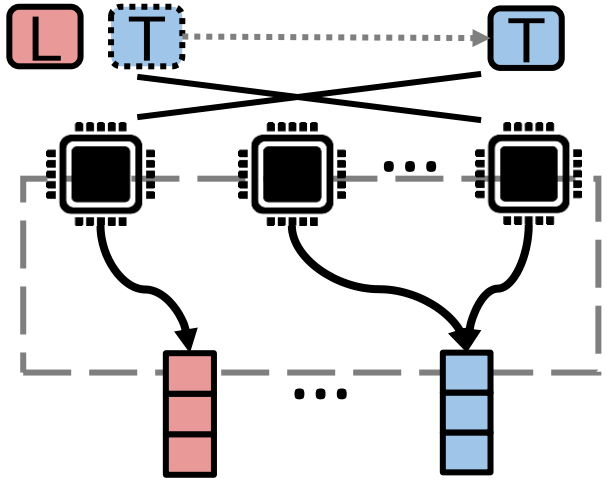
Static *blk-mq*: Constrained Optimization

Clear solution: **NQ-level separation** of L- and T-requests.



FlashShare (OSDI'18), D2FQ (FAST'21)

blk-switch (OSDI'21)



NQ overprovision:

😎 *Simple & Direct*

☹ *Underutilization & HW constraints*

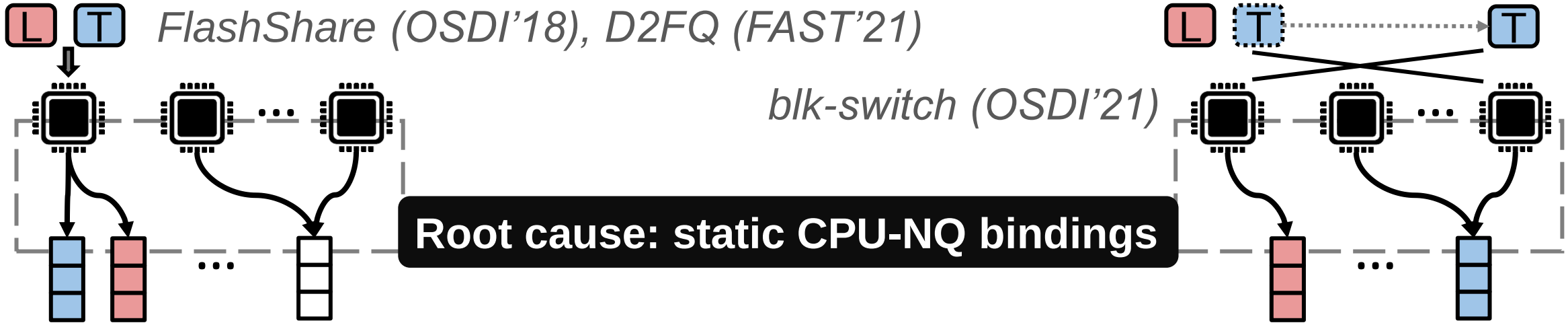
Cross-core scheduling:

😎 *Utilization & Generality*

☹ *CPU intervention*

Static *blk-mq*: Constrained Optimization

Clear solution: **NQ-level separation** of L- and T-requests.



NQ overprovision:

😎 *Simple & Direct*

☹ *Underutilization & HW constraints*

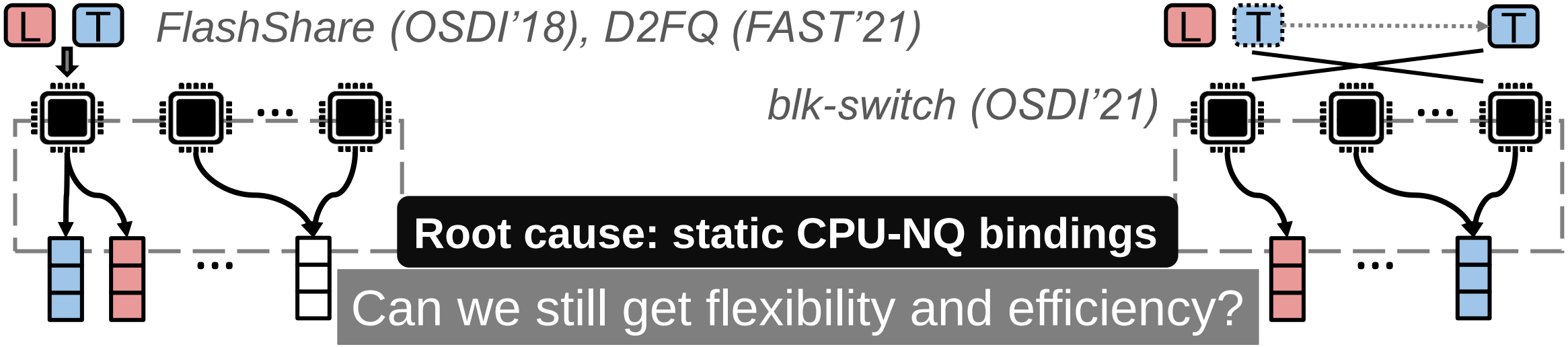
Cross-core scheduling:

😎 *Utilization & Generality*

☹ *CPU intervention*

Static *blk-mq*: Constrained Optimization

Clear solution: **NQ-level separation** of L- and T-requests.



NQ overprovision:

😎 *Simple & Direct*

☹️ *Underutilization & HW constraints*

Cross-core scheduling:

😎 *Utilization & Generality*

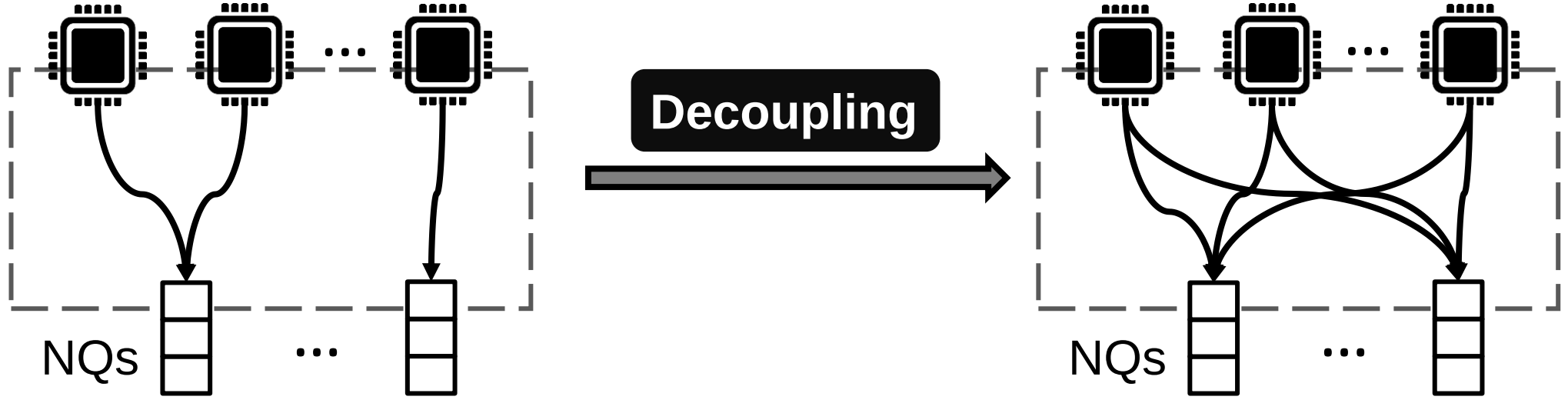
☹️ *CPU intervention*

Daredevil: Here Comes the Rescue

Core idea: **Decoupling** of CPU-NQ bindings.

Daredevil: Here Comes the Rescue

Core idea: **Decoupling** of CPU-NQ bindings.

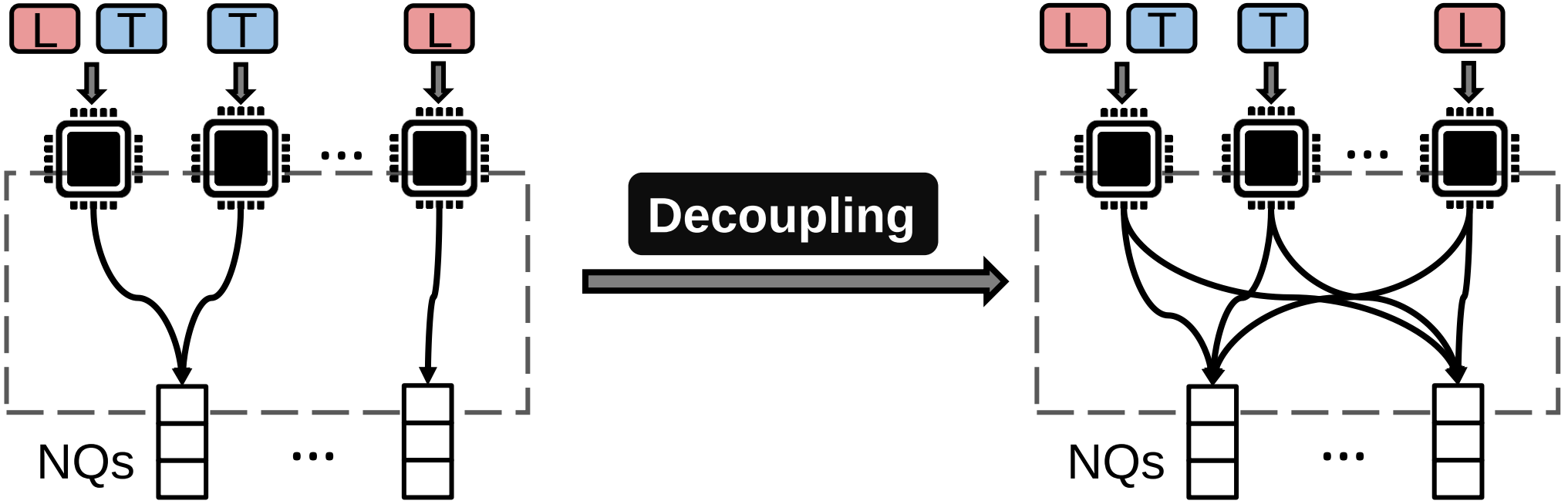


blk-mq based storage stack

Daredevil storage stack

Daredevil: Here Comes the Rescue

Core idea: **Decoupling** of CPU-NQ bindings.

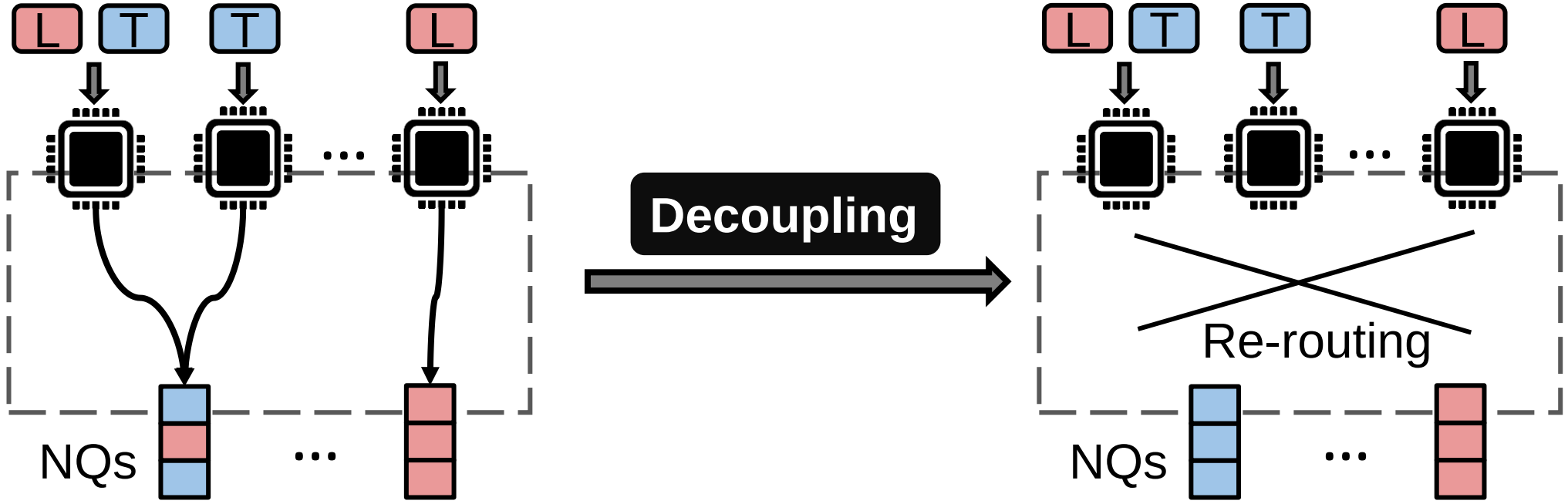


blk-mq based storage stack

Daredevil storage stack

Daredevil: Here Comes the Rescue

Core idea: **Decoupling** of CPU-NQ bindings.

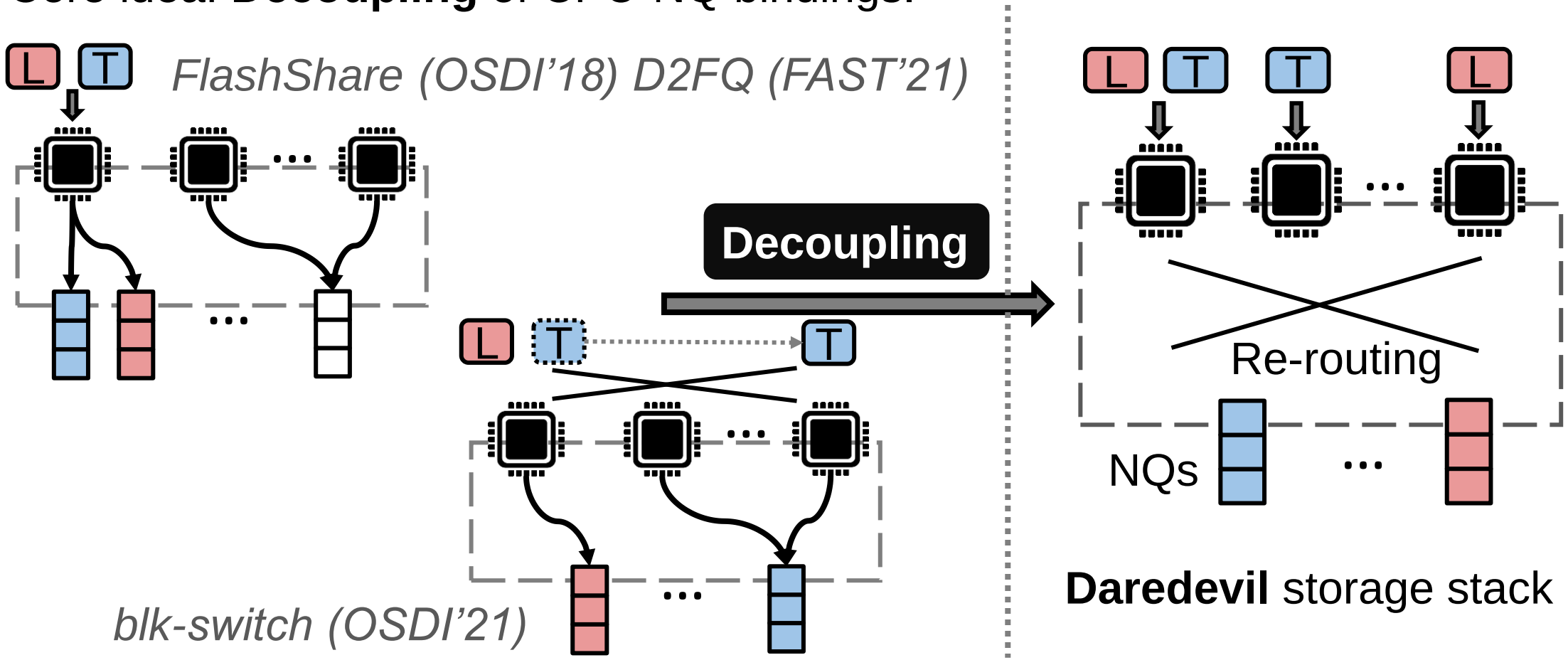


blk-mq based storage stack

Daredevil storage stack

Daredevil: Here Comes the Rescue

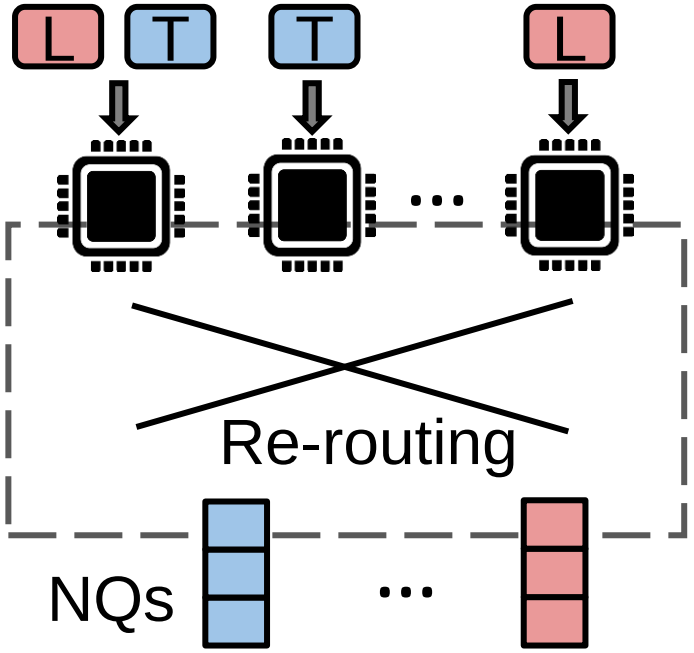
Core idea: **Decoupling** of CPU-NQ bindings.



Daredevil: Here Comes the Rescue

Core idea: **Decoupling** of CPU-NQ bindings.

- **Full-connectivity** between CPU cores and NQs.
- **Independent & flexible** policy for multi-tenancy control.
- **Full utilization** of NQs.



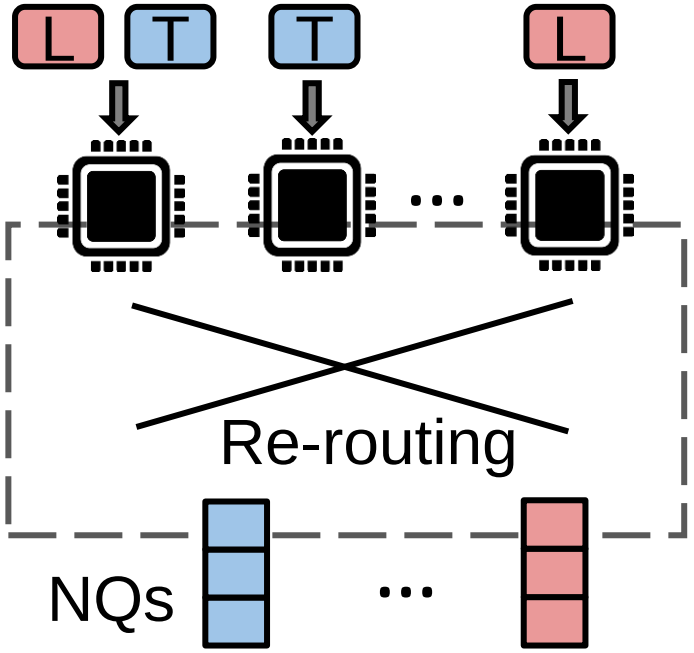
Daredevil storage stack

Daredevil: Here Comes the Rescue

Core idea: **Decoupling** of CPU-NQ bindings.

- **Full-connectivity** between CPU cores and NQs.
- **Independent & flexible** policy for multi-tenancy control.
- **Full utilization** of NQs.

But…At what cost?

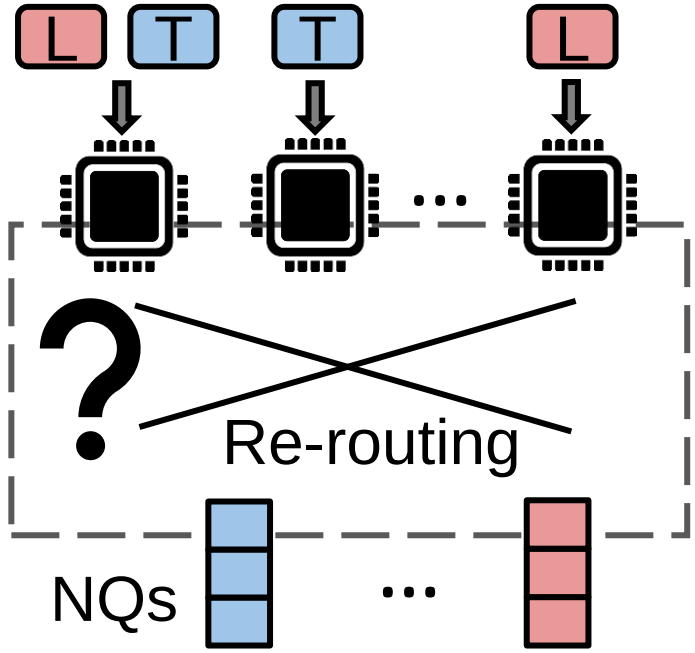


Daredevil storage stack

Daredevil: Here Comes the Rescue

Challenge #1: Light-weight routing decisions for I/O requests.

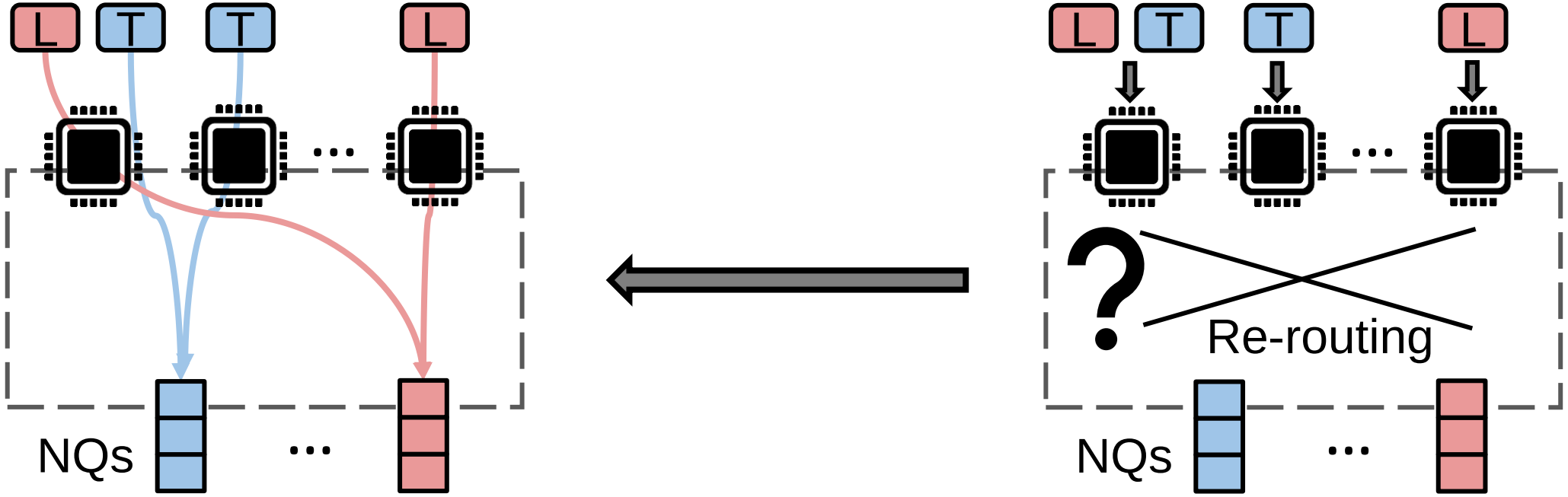
Where should I/O requests go?



Daredevil storage stack

Daredevil: Here Comes the Rescue

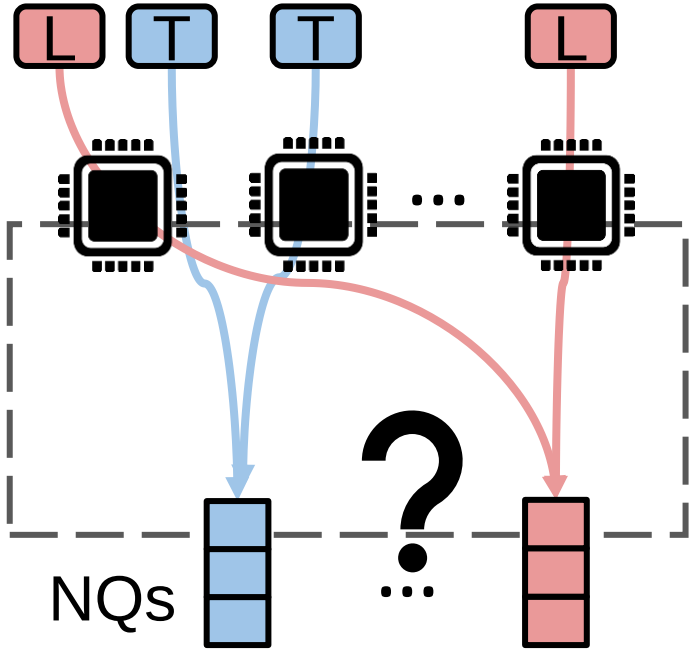
Challenge #1: Light-weight routing decisions for I/O requests.



Solution #1: Tenant-based request routing.

Daredevil: Here Comes the Rescue

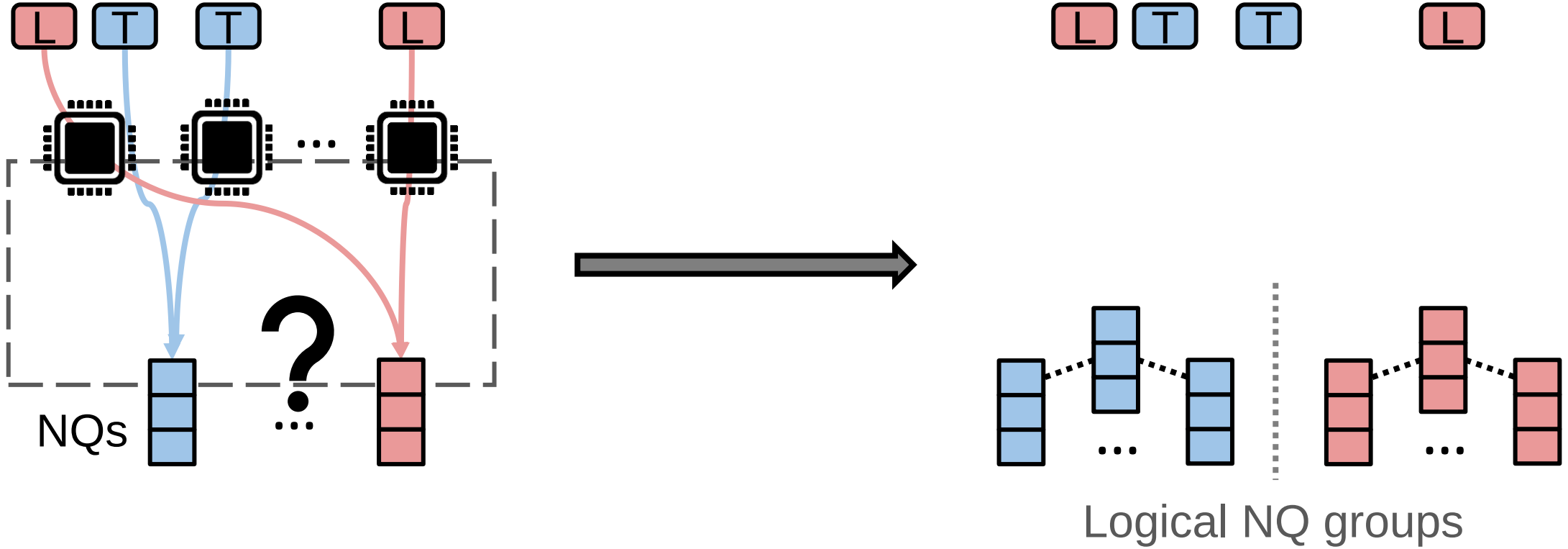
Challenge #2: Performance assurance for NQ-level separation.



How to ensure separation with guaranteed performance?

Daredevil: Here Comes the Rescue

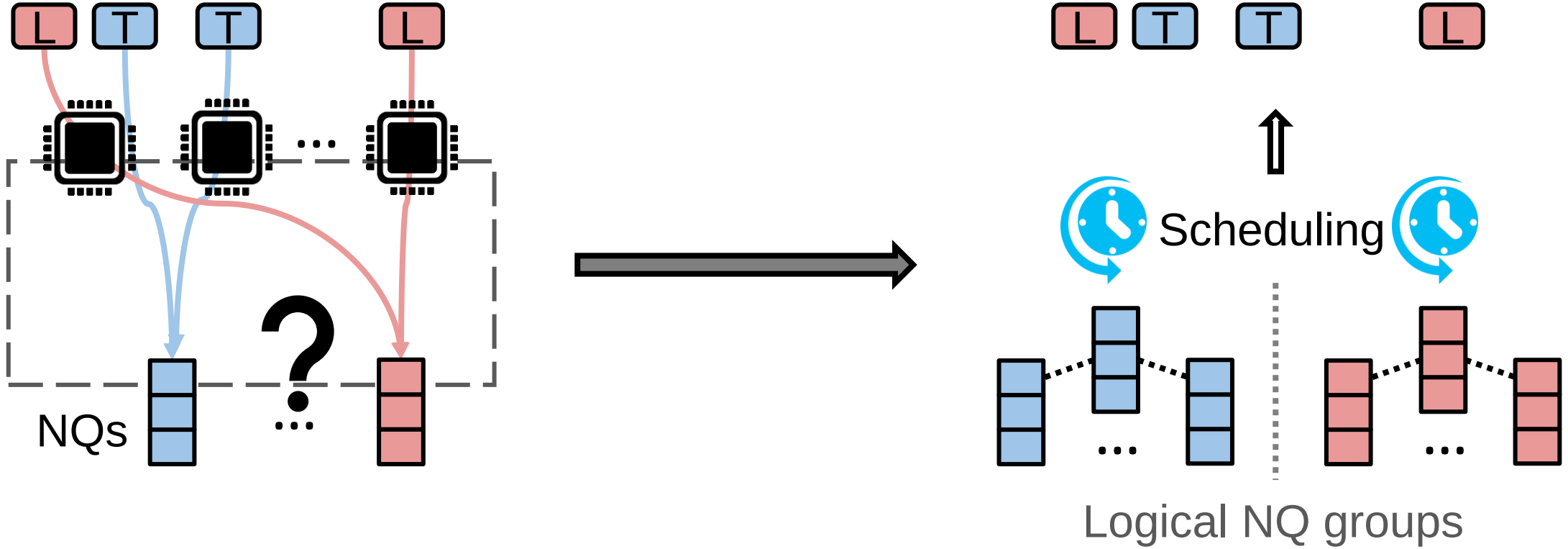
Challenge #2: Performance assurance for NQ-level separation.



Solution #2: Heap-based performance-aware NQ scheduling.

Daredevil: Here Comes the Rescue

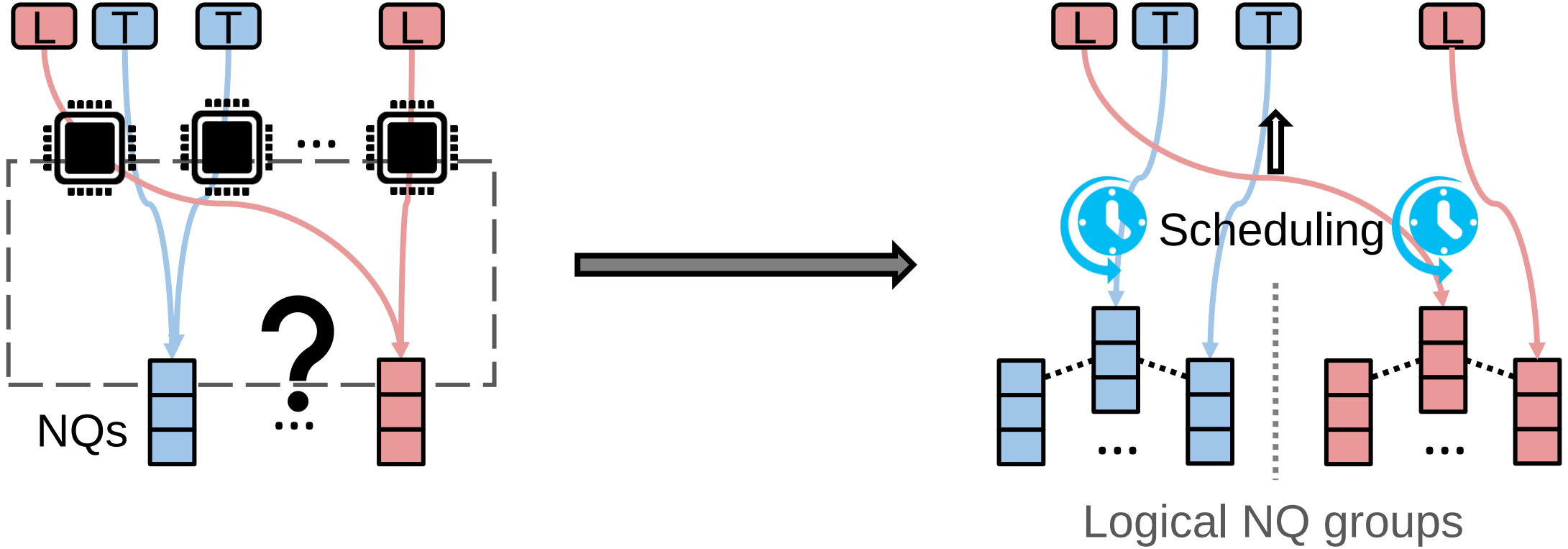
Challenge #2: Performance assurance for NQ-level separation.



Solution #2: Heap-based performance-aware NQ scheduling.

Daredevil: Here Comes the Rescue

Challenge #2: Performance assurance for NQ-level separation.



Solution #2: Heap-based performance-aware NQ scheduling.

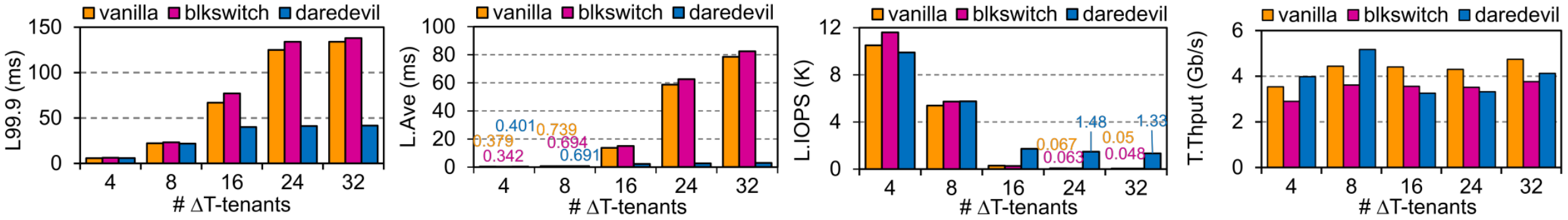
Daredevil: Here Comes the Rescue

Design & implementation details:

- Tenant & outlier cases identification
- Scheduling criteria
- Light-weight concurrent scheduling
- I/O service acceleration

Please refer to our paper for more details.

Evaluation: Efficient Multi-tenancy Support



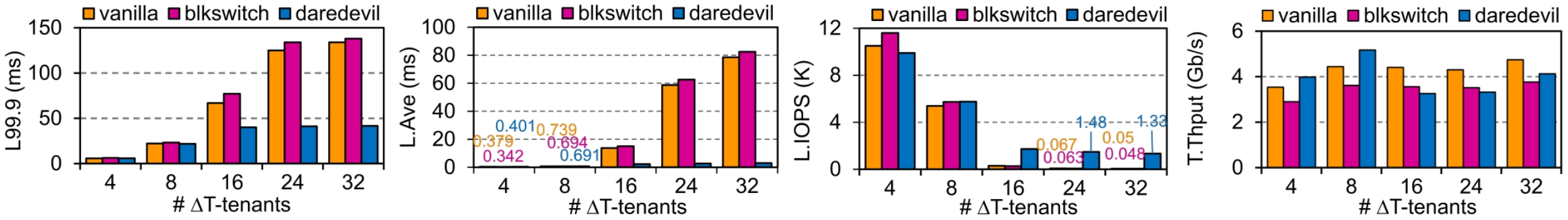
(a) L-tenant 99.9^{th} tail latency. **(b)** L-tenant average latency. **(c)** L-tenant I/O rate. **(d)** T-tenant throughput.

Figure 6. Performance results with increasing T-pressure in SV-M. DAREDEVIL maintains in-time responses for L-tenants even under extreme T-pressure, while the vanilla kernel and `blk-switch` significantly inflate the L-tenants' I/O latency.

Benchmark: FIO-based simulation for L- and T-tenants.

Evaluation: Efficient Multi-tenancy Support

***3x* & 27*x* reduction** in tail & ave latency.



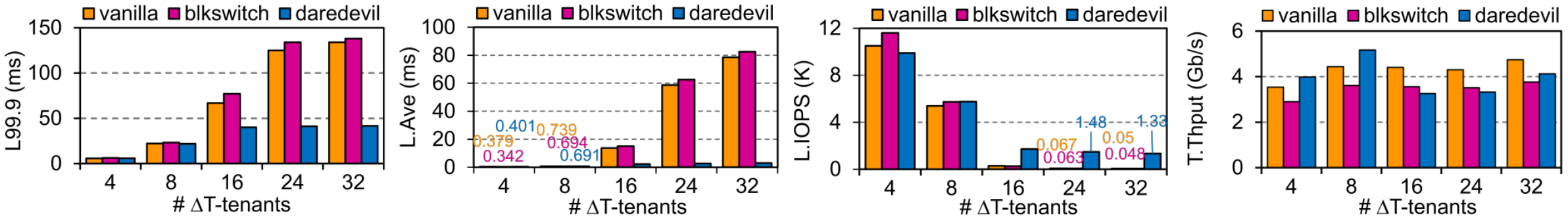
(a) L-tenant 99.9^{th} tail latency. **(b)** L-tenant average latency. **(c)** L-tenant I/O rate. **(d)** T-tenant throughput.

Figure 6. Performance results with increasing T-pressure in SV-M. DAREDEVIL maintains in-time responses for L-tenants even under extreme T-pressure, while the vanilla kernel and `blk-switch` significantly inflate the L-tenants' I/O latency.

Benchmark: FIO-based simulation for L- and T-tenants.

Evaluation: Efficient Multi-tenancy Support

***3x* & *27x* reduction** in tail & ave latency. *No I/O blocking.*



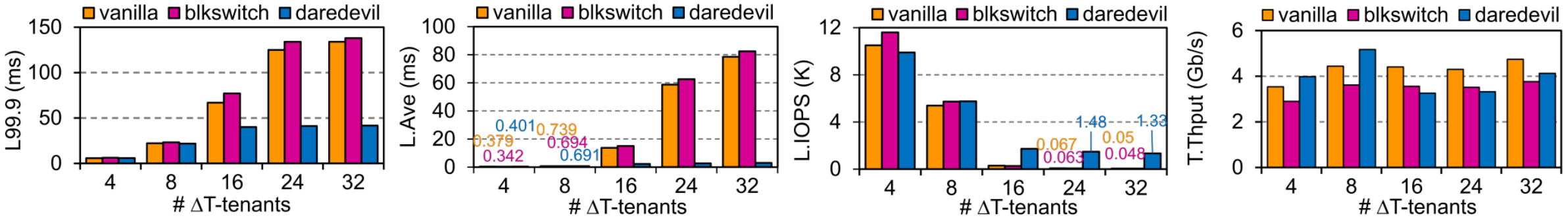
(a) L-tenant 99.9^{th} tail latency. **(b)** L-tenant average latency. **(c)** L-tenant I/O rate. **(d)** T-tenant throughput.

Figure 6. Performance results with increasing T-pressure in SV-M. DAREDEVIL maintains in-time responses for L-tenants even under extreme T-pressure, while the vanilla kernel and `blk-switch` significantly inflate the L-tenants' I/O latency.

Benchmark: FIO-based simulation for L- and T-tenants.

Evaluation: Efficient Multi-tenancy Support

***3x* & *27x* reduction** in tail & ave latency. *No I/O blocking.* *Comparable thput.*



(a) L-tenant 99.9^{th} tail latency. **(b)** L-tenant average latency. **(c)** L-tenant I/O rate. **(d)** T-tenant throughput.

Figure 6. Performance results with increasing T-pressure in SV-M. DAREDEVIL maintains in-time responses for L-tenants even under extreme T-pressure, while the vanilla kernel and `blk-switch` significantly inflate the L-tenants' I/O latency.

Benchmark: FIO-based simulation for L- and T-tenants.

Evaluation: Ablation Study

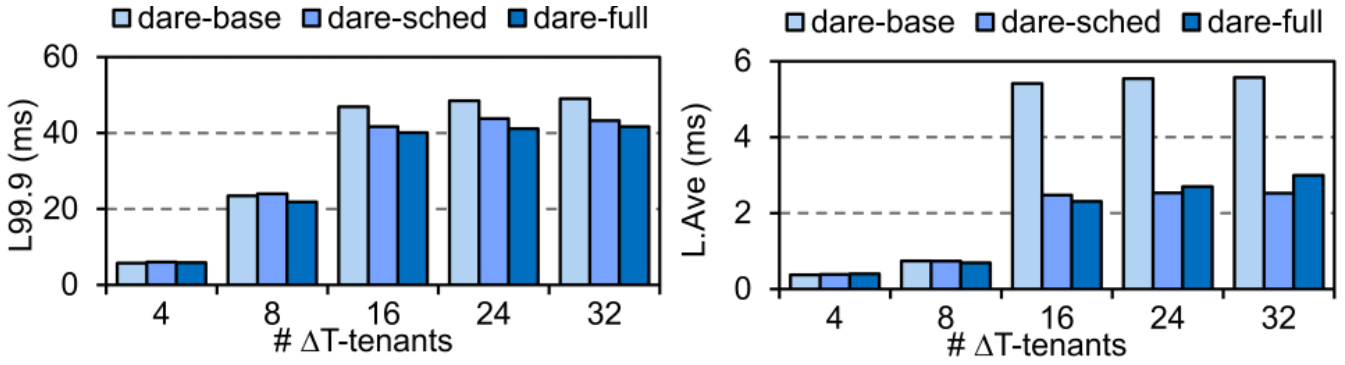
Ablation of Daredevil: What contributions do its optimizations make?

- dare-base: only decoupling and round-robin request routing
- dare-sched: decoupling + NQ scheduling
- dare-full: dare-sched + I/O service acceleration

Evaluation: Ablation Study

Ablation of Daredevil: What contributions do its optimizations make?

- dare-base: only decoupling and round-robin request routing
- dare-sched: decoupling + NQ scheduling
- dare-full: dare-sched + I/O service acceleration



(a) L-tenant 99.9^{th} tail latency with increasing T-pressure.

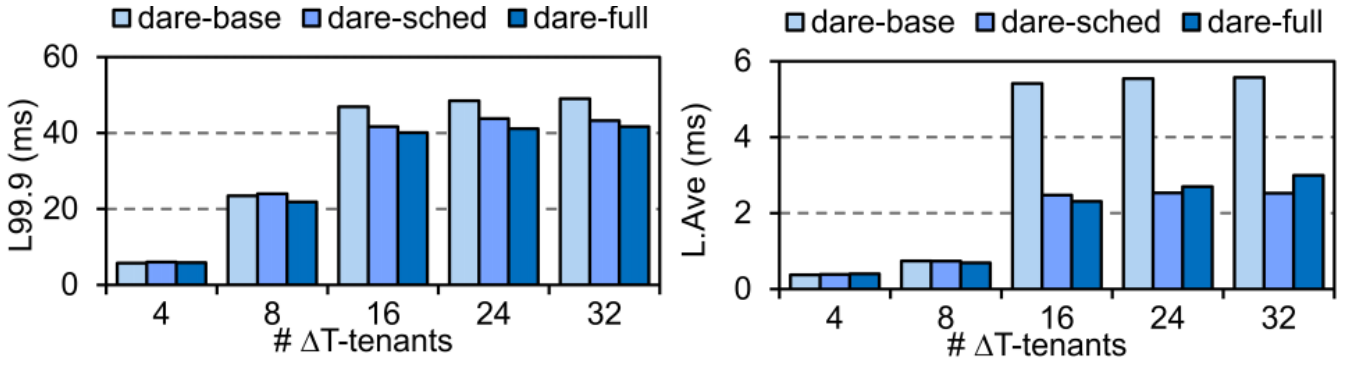
(b) L-tenant average latency with increasing T-pressure.

- Decoupled block layer already achieves low latency.

Evaluation: Ablation Study

Ablation of Daredevil: What contributions do its optimizations make?

- dare-base: only decoupling and round-robin request routing
- dare-sched: decoupling + NQ scheduling
- dare-full: dare-sched + I/O service acceleration



(a) L-tenant 99.9^{th} tail latency with increasing T-pressure.

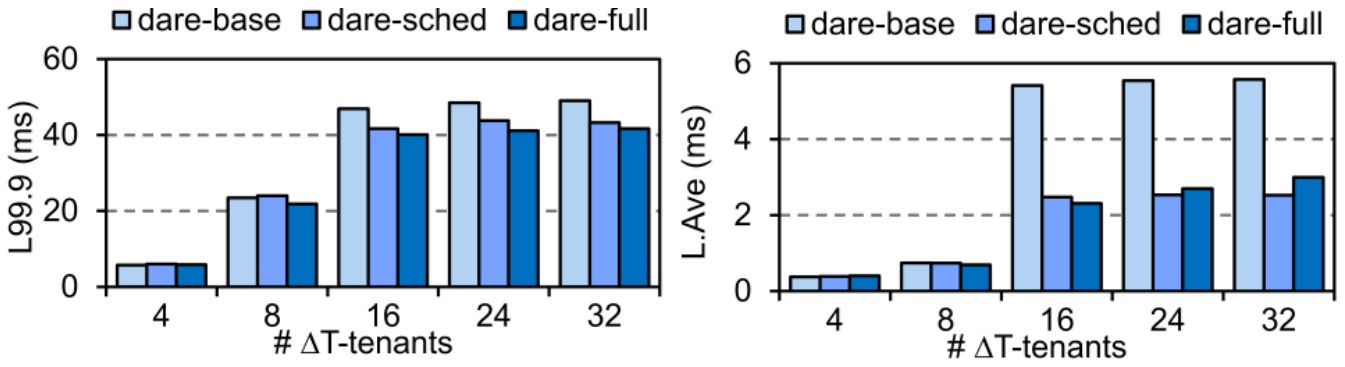
(b) L-tenant average latency with increasing T-pressure.

- Decoupled block layer already achieves low latency.
- NQ scheduling significantly contributes.

Evaluation: Ablation Study

Ablation of Daredevil: What contributions do its optimizations make?

- dare-base: only decoupling and round-robin request routing
- dare-sched: decoupling + NQ scheduling
- dare-full: dare-sched + I/O service acceleration



(a) L-tenant 99.9^{th} tail latency with increasing T-pressure.

(b) L-tenant average latency with increasing T-pressure.

- Decoupled block layer already achieves low latency.
- NQ scheduling significantly contributes.
- I/O service acceleration reduces tail latency.

Discussion

Compatibility with virtual machines (VMs)?

- Not yet: processes inside VMs are invisible to the host.

Beyond NVMe SSDs to new devices?

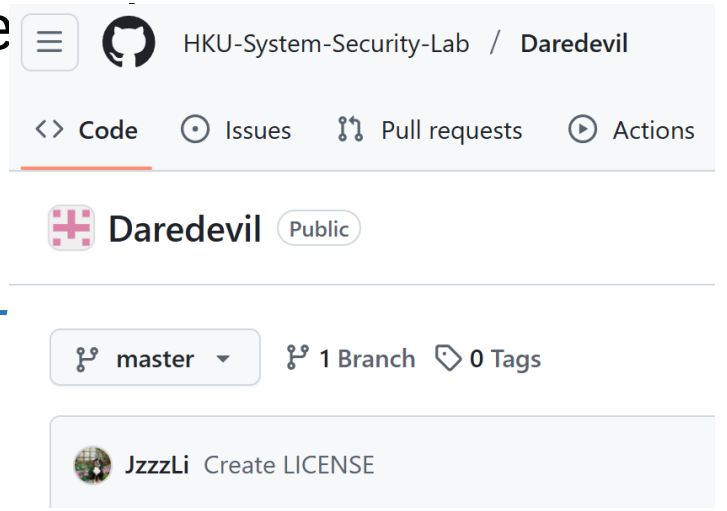
- Possible: the multi-queue feature is maintained for CXL/ZNS SSDs.

Future road after Daredevil:

- Finer-grained performance isolation/consideration with cgroups.
- More comprehensive maintenance with CPU core scheduling.

Conclusion

- Daredevil is a wild research prototype to challenge the static Linux kernel storage stack.
- It achieves *flexibility* and *efficiency* with higher performance multi-tenant I/O services.
- Open sourced at: *https://github.com/HKU-System-Security-Lab/Daredevil*



*Please contact **Junzhe Li** (jzzzli@connect.hku.hk) for any questions!*

Thank you!