BTS: Exploring Effects of Background Task-Aware Scheduling for Key-Value CSDs

Yeohyeon Park, Chang-Gyu Lee, Seungjin Lee,
Inhyuk Park, Soonyeal Yang, Woosuk Chung, Youngjae Kim

7th International Parallel Data Systems Workshop (PDSW'22)

Outline

- Background
  - Computational Storage Device (CSD)
  - Intel SPDK
- Motivation
- Proposed Architecture
  - BTS : Background Task-Aware Scheduler
  - Execution Flow
- Evaluation
- Conclusion and Q&A

Background

Computational Storage

- What is computational storage device (CSD)?

Computational Storage

- What is computational storage device (CSD)?
  - Computational storage devices (CSD) can run computational tasks inside the storage device, reducing data transfer between the host and the device.

Computational Storage

- What is computational storage device (CSD)?
  - Computational storage devices (CSD) can run computational tasks inside the storage device, reducing data transfer between the host and the device.

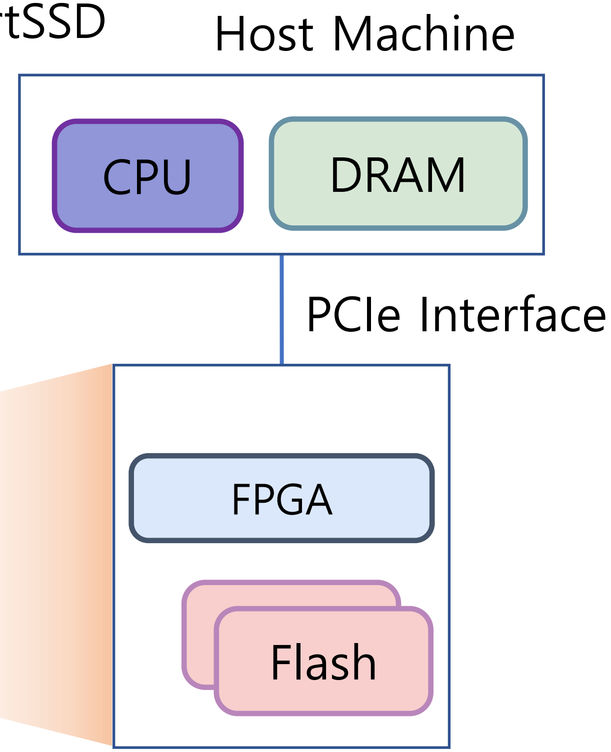
- CSD **without OS**

- CSD **with OS**

Computational Storage

- What is computational storage device (CSD)?
  - Computational storage devices (CSD) can run computational tasks inside the storage device, reducing data transfer between the host and the device.

- CSD **without OS**
  - Samsung SmartSSD
  - INSIDER [1]
  - PolarDB [2]



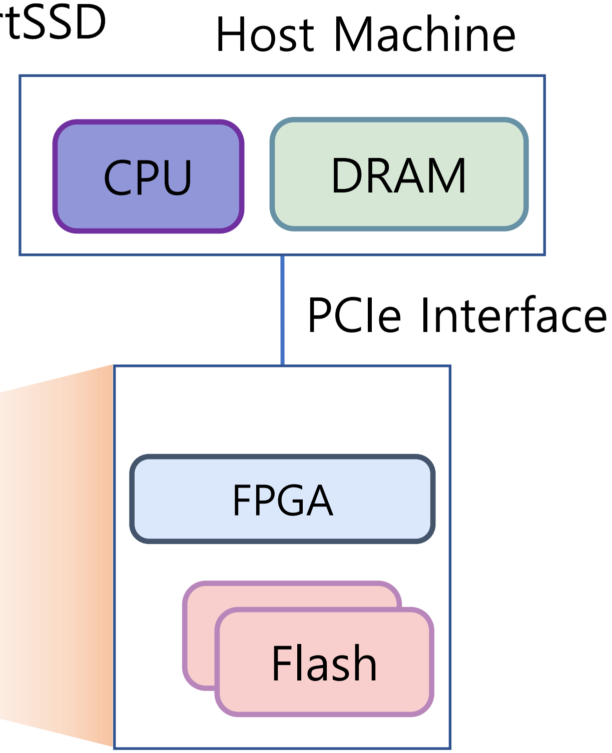
- CSD **with OS**

[1] Z. Ruan et. al., “INSIDER: Designing In-Storage Computing System for Emerging High-Performance Drive,” USENIX ATC ’19
[2] W. Cao et. al., “POLARDB Meets Computational Storage: Efficiently Support Analytical Workloads in Cloud-Native Relational Database,” FAST ’14

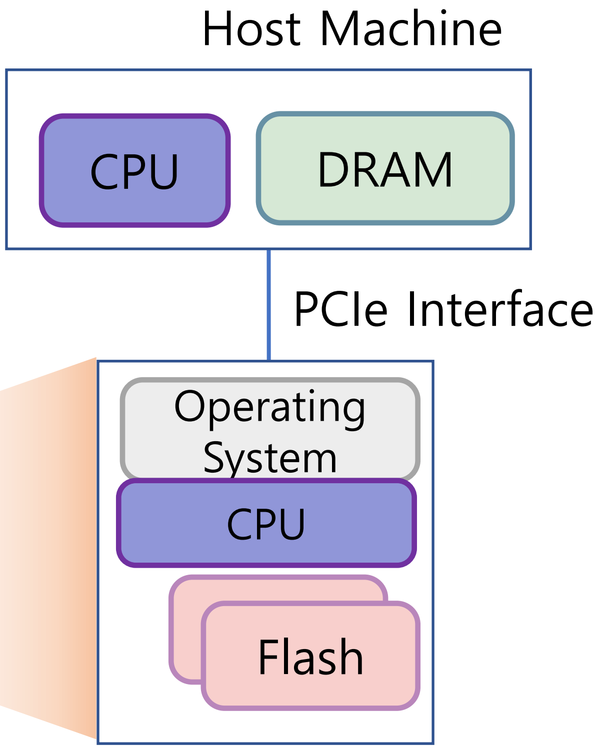
Computational Storage

- What is computational storage device (CSD)?
  - Computational storage devices (CSD) can run computational tasks inside the storage device, reducing data transfer between the host and the device.

- CSD **without OS**
  - Samsung SmartSSD
  - INSIDER [1]
  - PolarDB [2]



- CSD **with OS**
  - SK hynix KV-CSD
  - NGD System Newport SSD



[1] Z. Ruan et. al., “INSIDER: Designing In-Storage Computing System for Emerging High-Performance Drive,” USENIX ATC ’19
[2] W. Cao et. al., “POLARDB Meets Computational Storage: Efficiently Support Analytical Workloads in Cloud-Native Relational Database,” FAST ’14

CSD **with OS**

- Pros
  - Programmability, and manageability

- Cons
  - OS overhead due to frequent interrupts and context switches

[1] Intel. SPDK. https://spdk.io/.

CSD **with OS**

- Pros
  - Programmability, and manageability

- Cons
  - OS overhead due to frequent interrupts and context switches

To reduce OS overhead, CSD can adopt Intel SPDK[1].

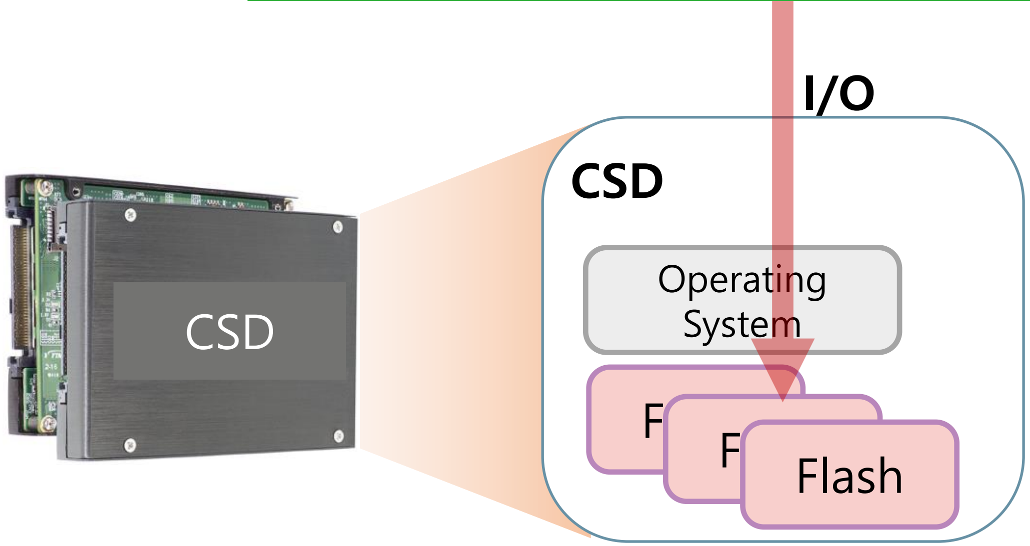
[1] Intel. SPDK. https://spdk.io/.

CSD **with OS**

- ❑ Pros
  - ➢ Programmability, and manageability

- ❑ Cons
  - ➢ OS overhead due to frequent interrupts and context switches

To reduce OS overhead, CSD can adopt Intel SPDK[1].



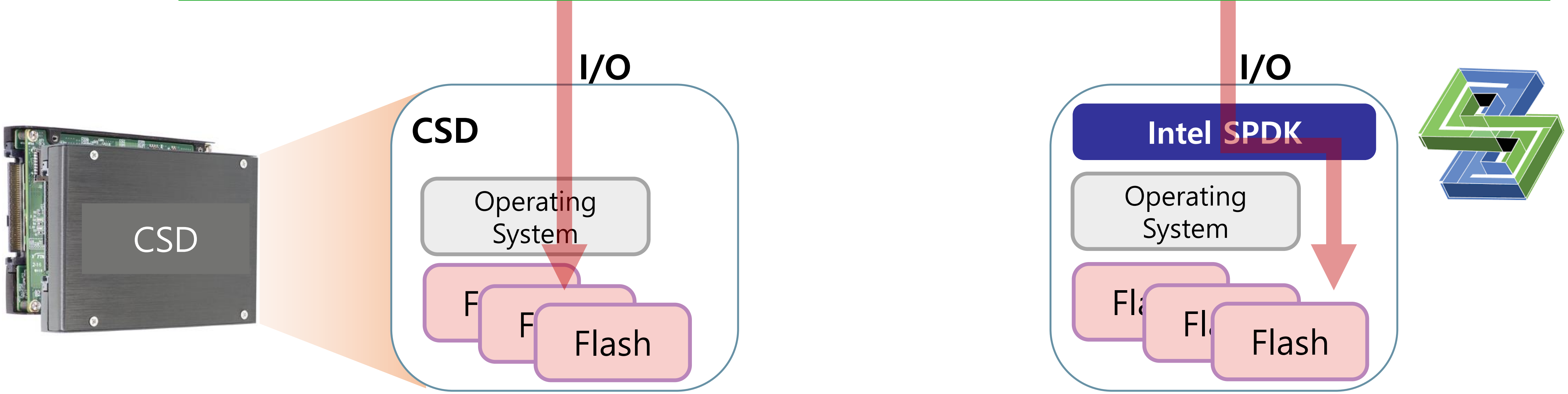
[1] Intel. SPDK. https://spdk.io/.

CSD **with OS**

- Pros
  - Programmability, and manageability

- Cons
  - OS overhead due to frequent interrupts and context switches

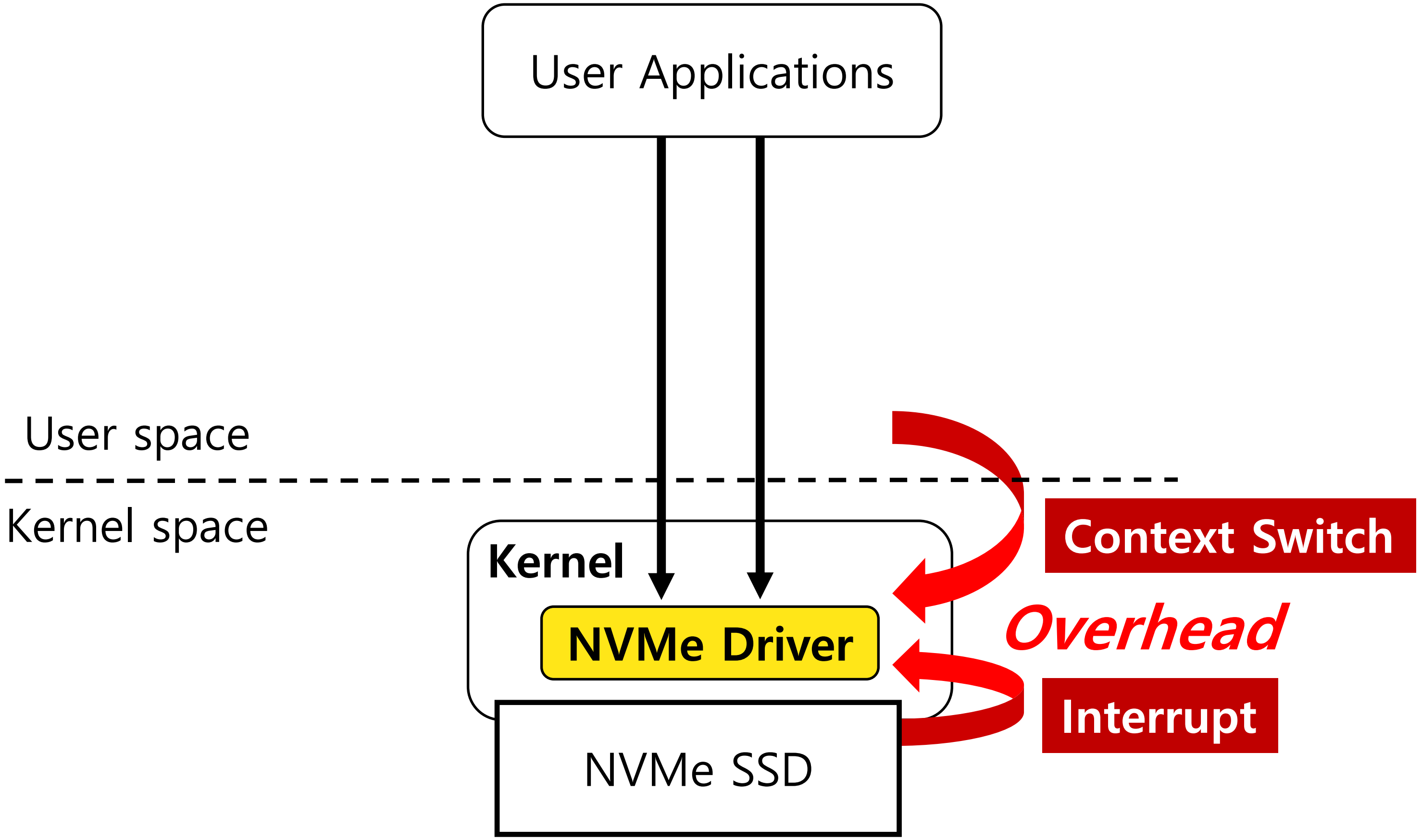
To reduce OS overhead, CSD can adopt Intel SPDK[1].



[1] Intel. SPDK. https://spdk.io/.

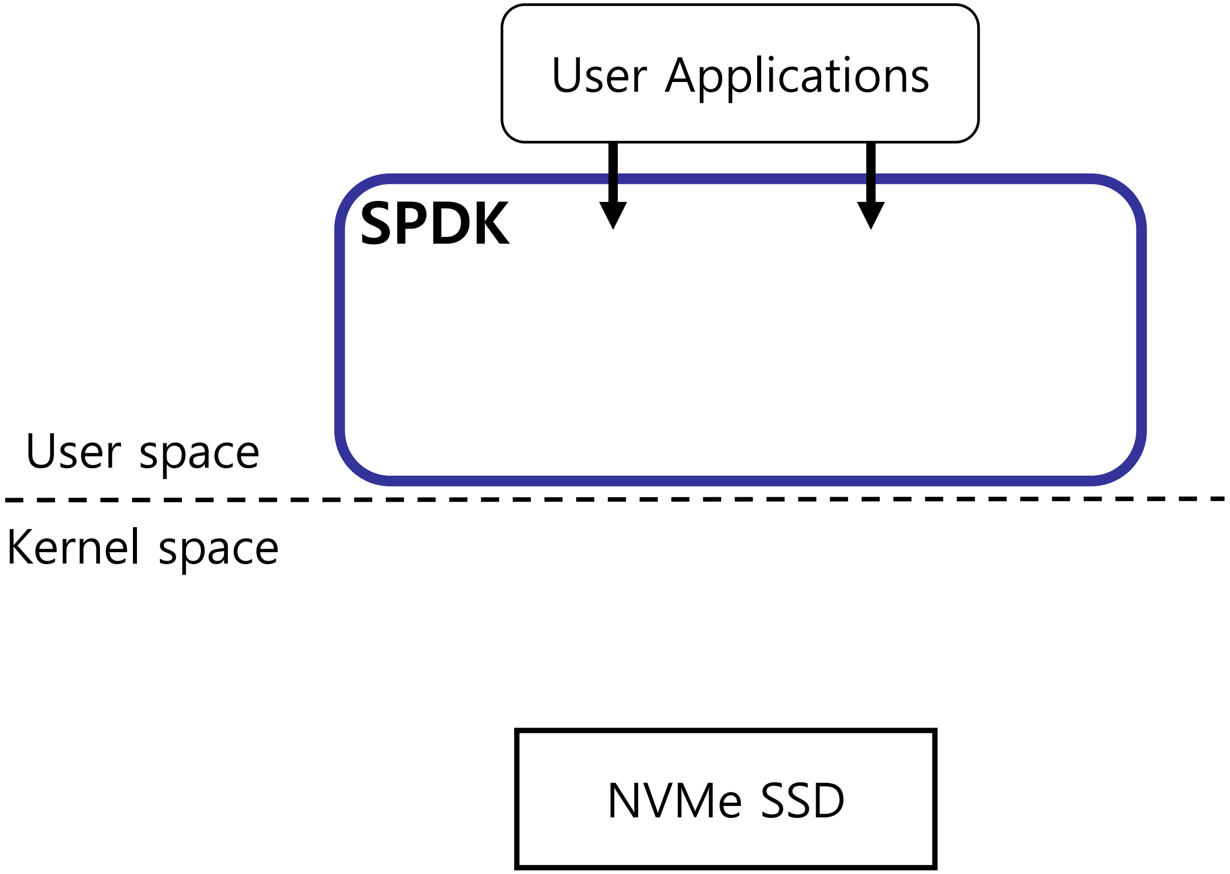
Traditional I/O Stack

- Kernel-level NVMe Driver



Intel SPDK[1]

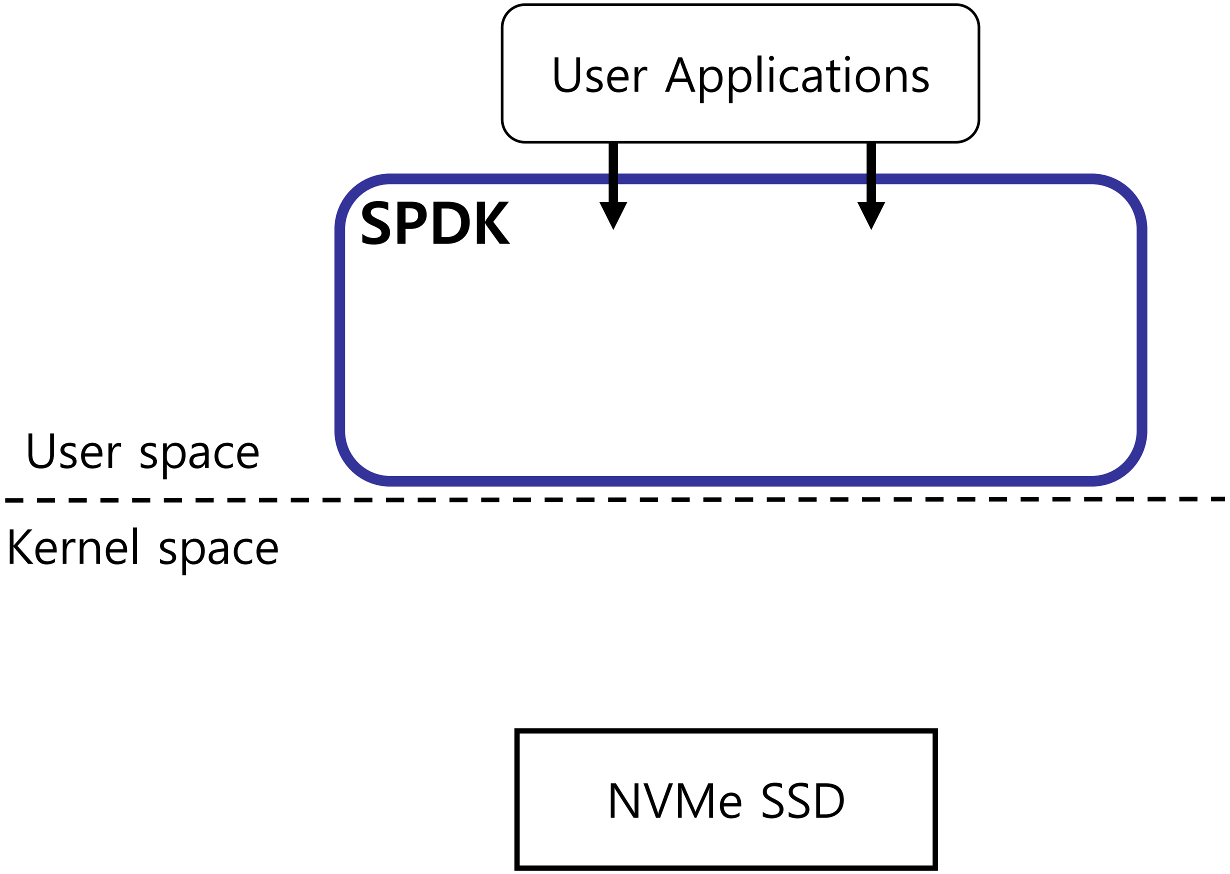
- User-level NVMe Driver



[1] Intel. SPDK. https://spdk.io/.

Intel SPDK[1]

❑ User-level NVMe Driver

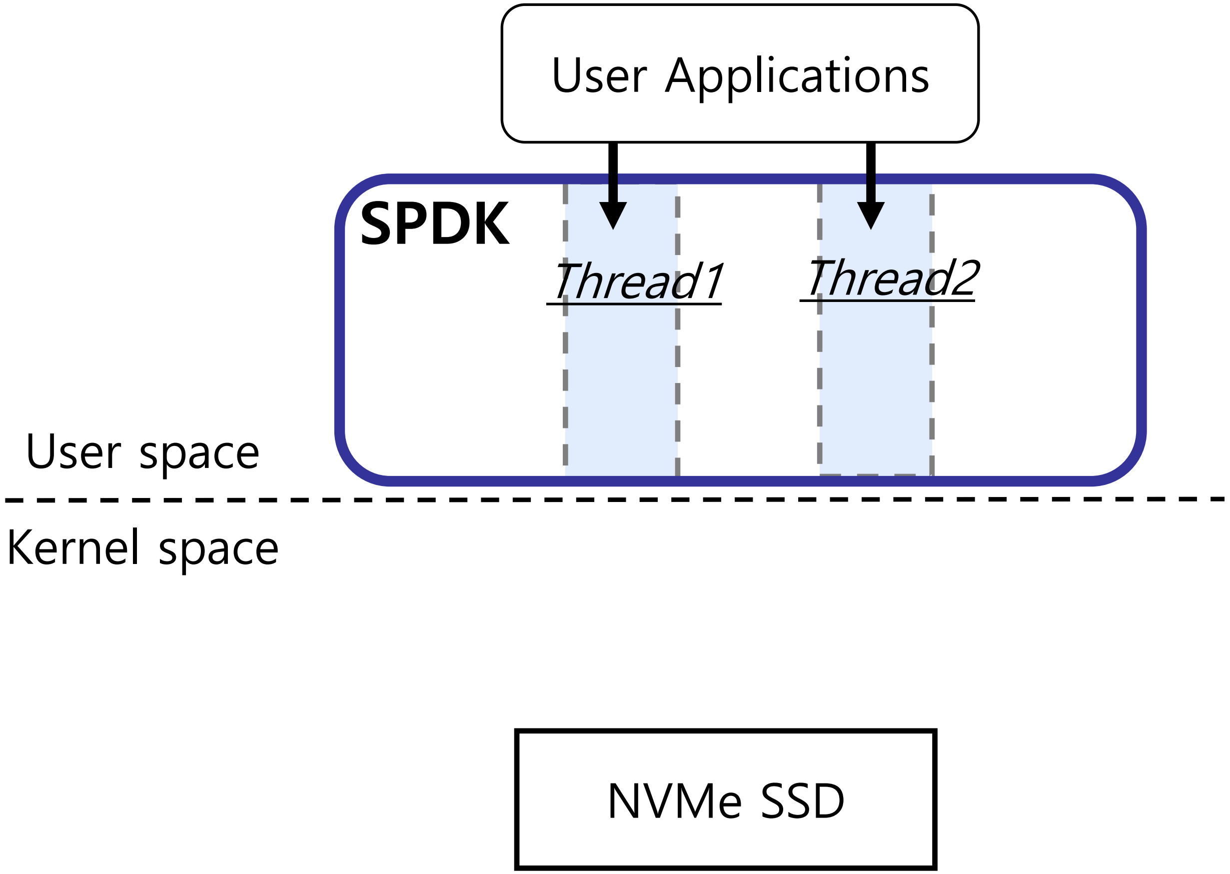


1. User level NVMe device driver

[1] Intel. SPDK. https://spdk.io/.

Intel SPDK[1]

❑ User-level NVMe Driver



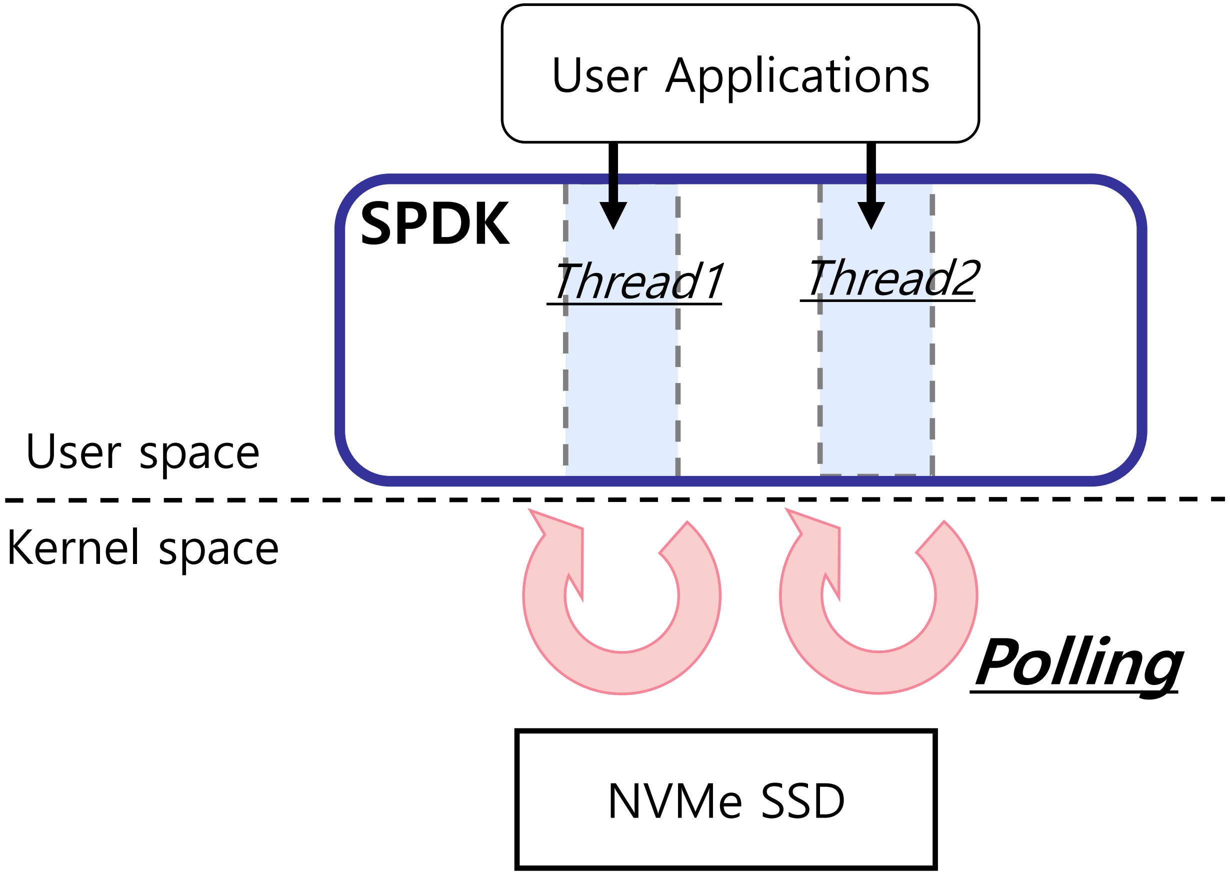
1. User level NVMe device driver

2. Binds I/O at the core

[1] Intel. SPDK. https://spdk.io/.

Intel SPDK[1]

❑ User-level NVMe Driver



1. User level NVMe device driver

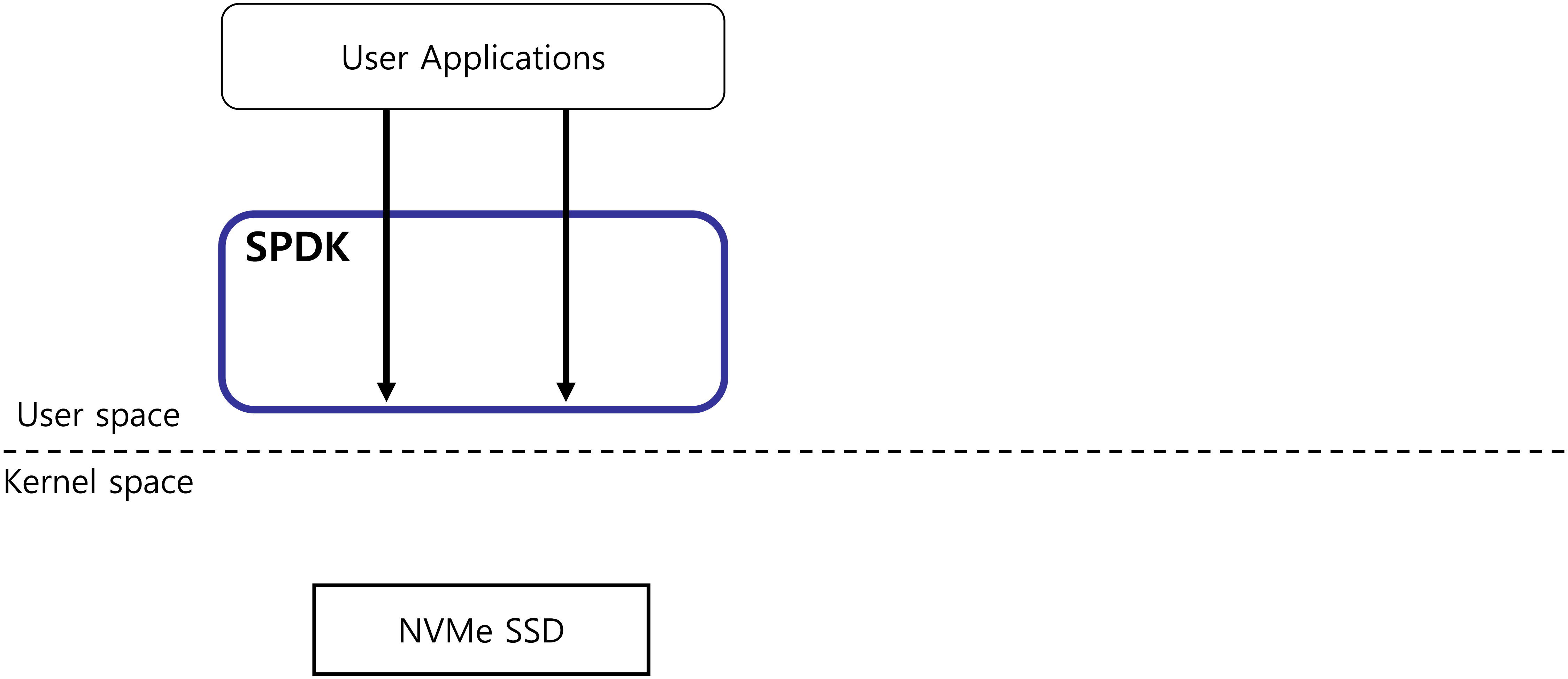
2. Binds I/O at the core

3. Uses polled mode

[1] Intel. SPDK. https://spdk.io/.

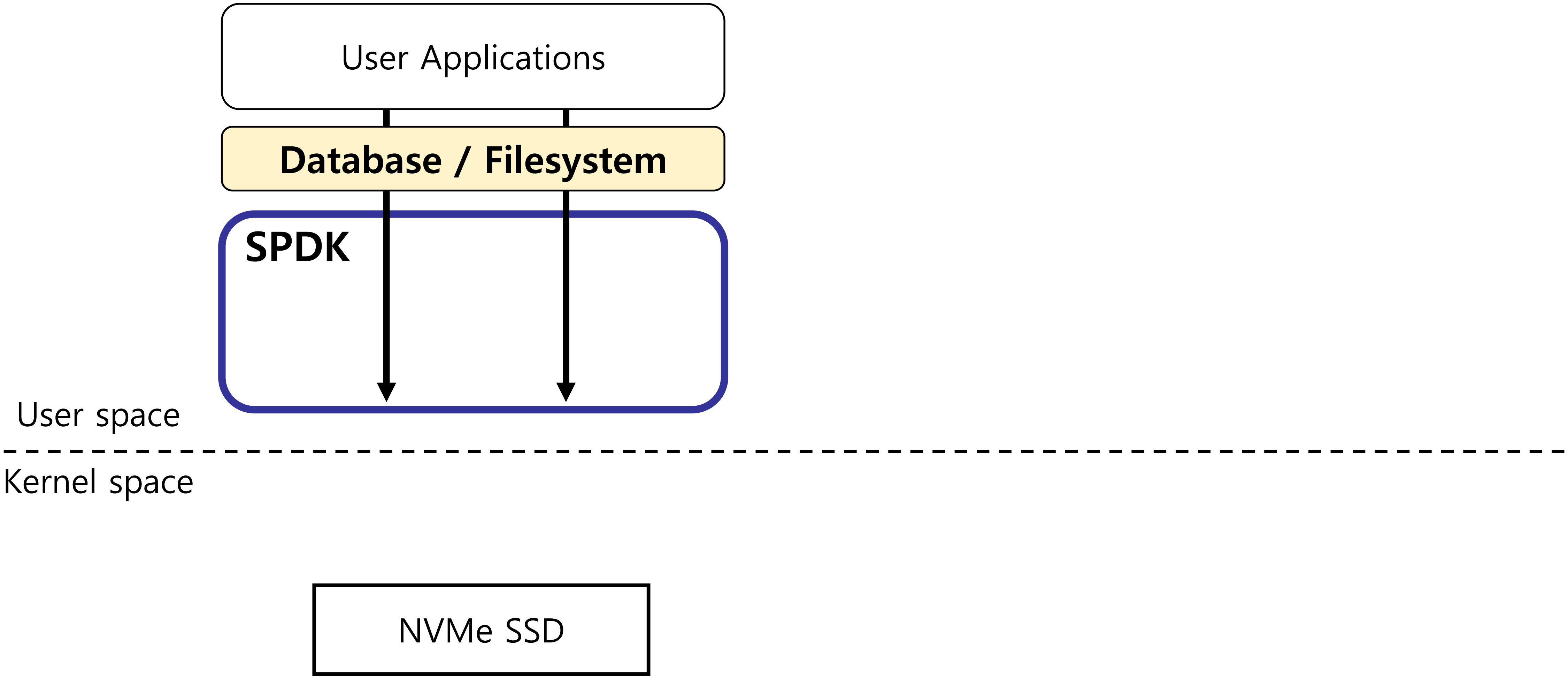
Use Case

- ❑ Storage Applications



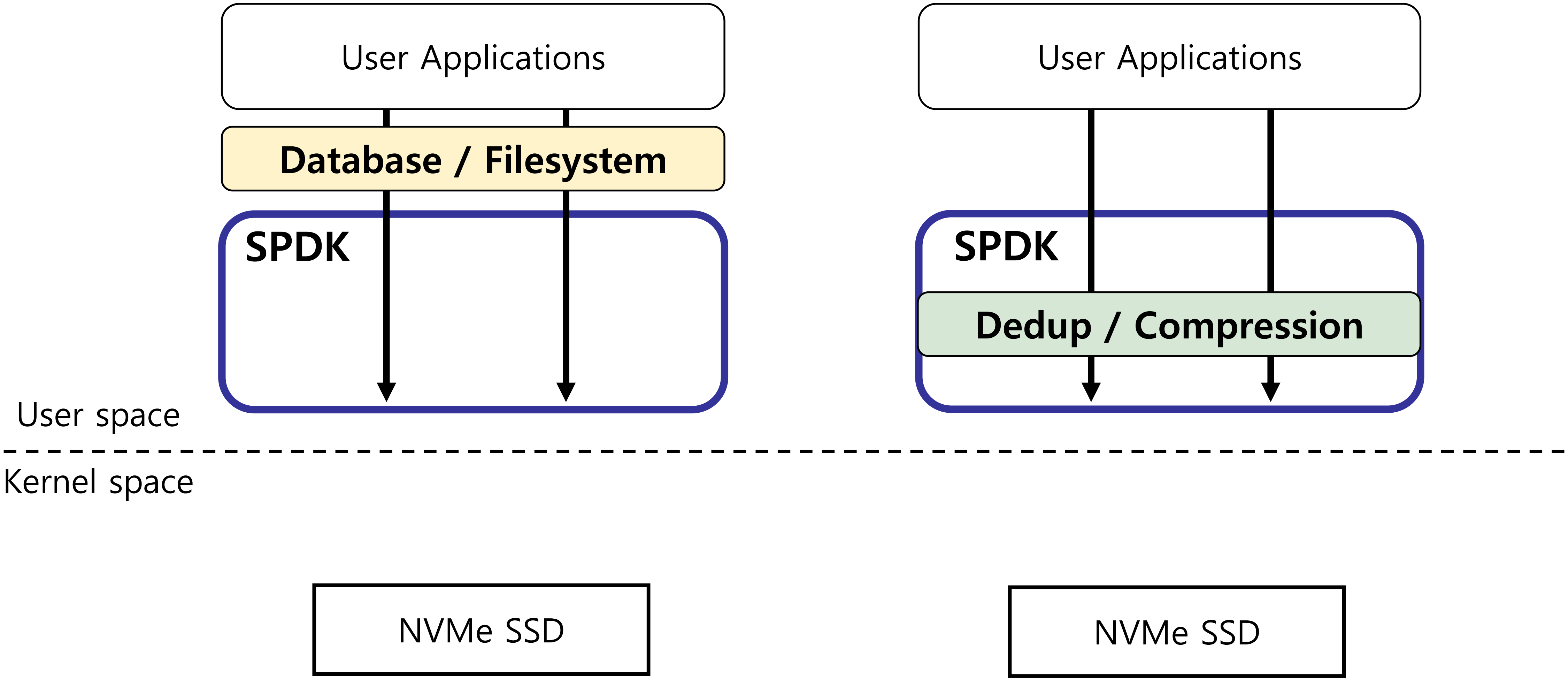
Use Case

- Storage Applications



Use Case

- ❑ Storage Applications
- ❑ Storage Services (Deduplication)



Research Problem

However, SPDK has a problem in that foreground I/O and background service tasks compete for CPU cores.

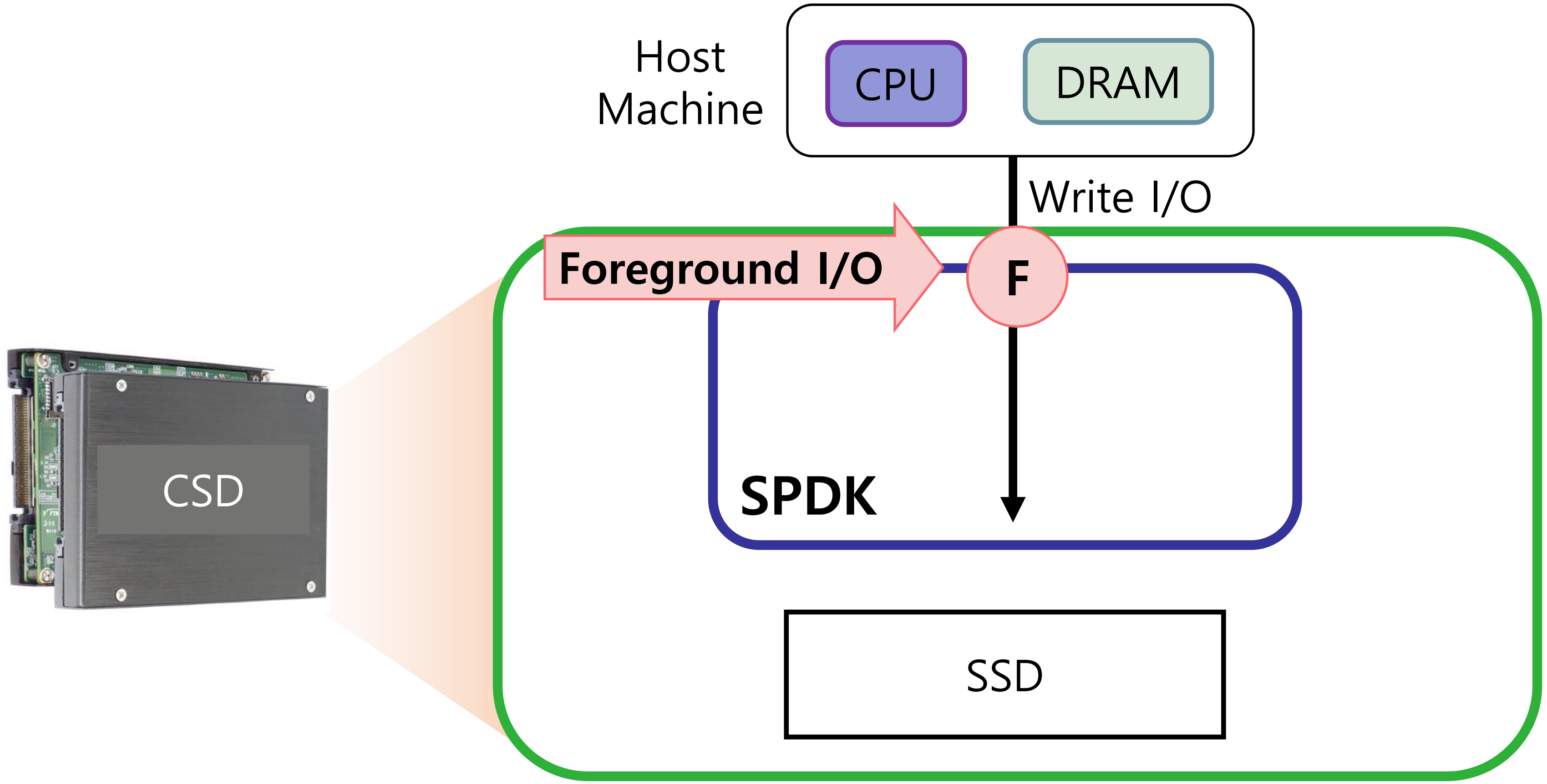
Research Problem

However, SPDK has a problem in that foreground I/O and background service tasks compete for CPU cores.

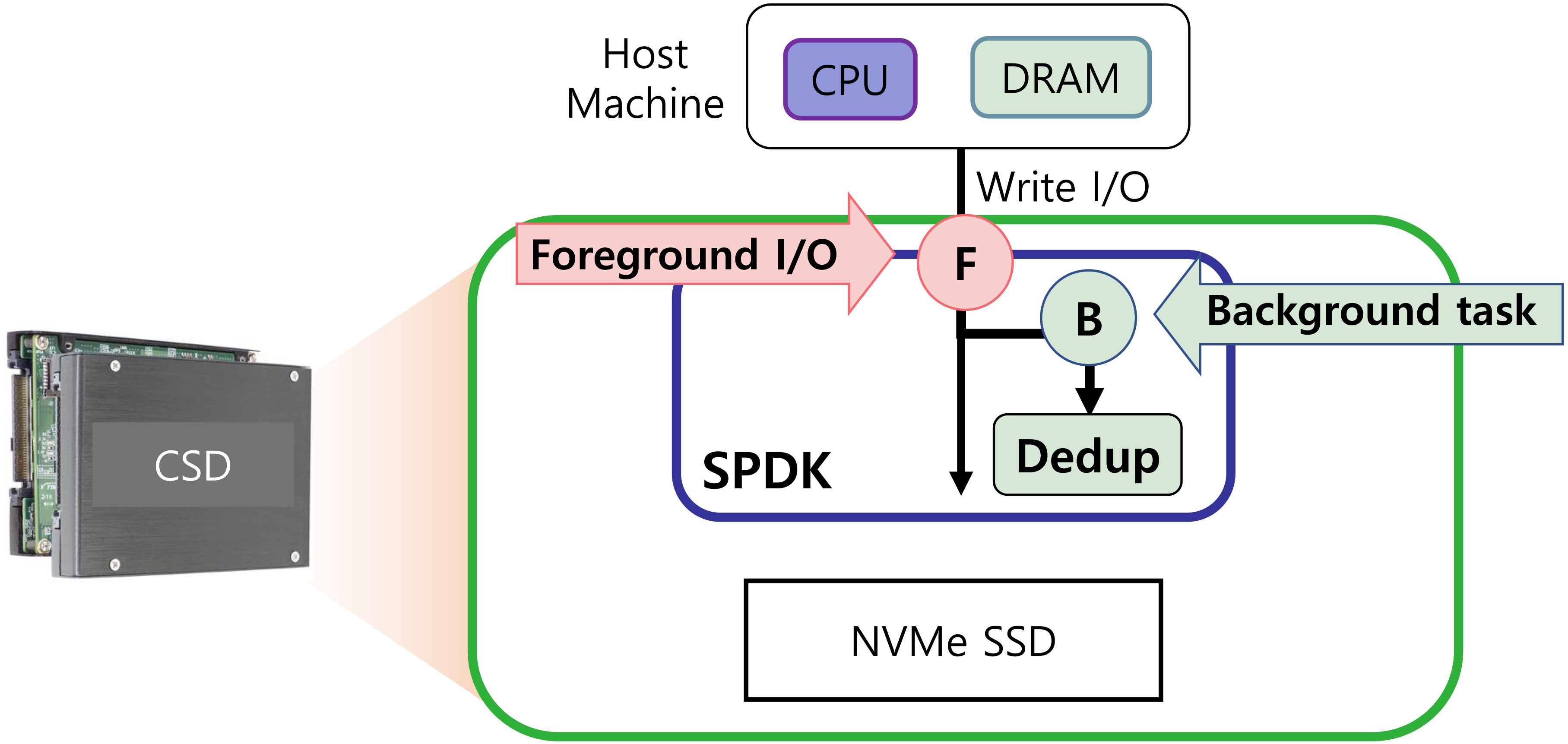
Contention of these tasks for CPU cores increases the response time of foreground I/O.

Motivation

Executing Background Tasks in SPDK



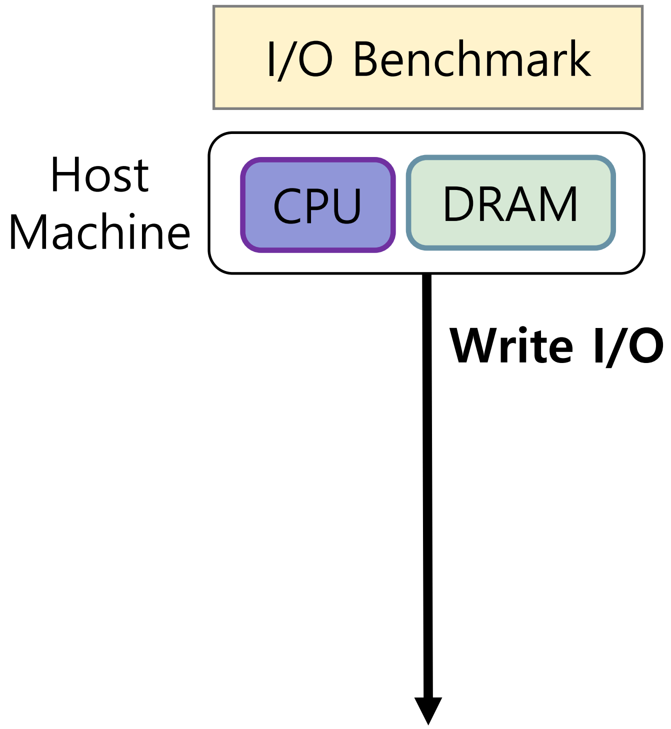
Executing Background Tasks in SPDK



In SPDK, background tasks are derived from foreground I/O.

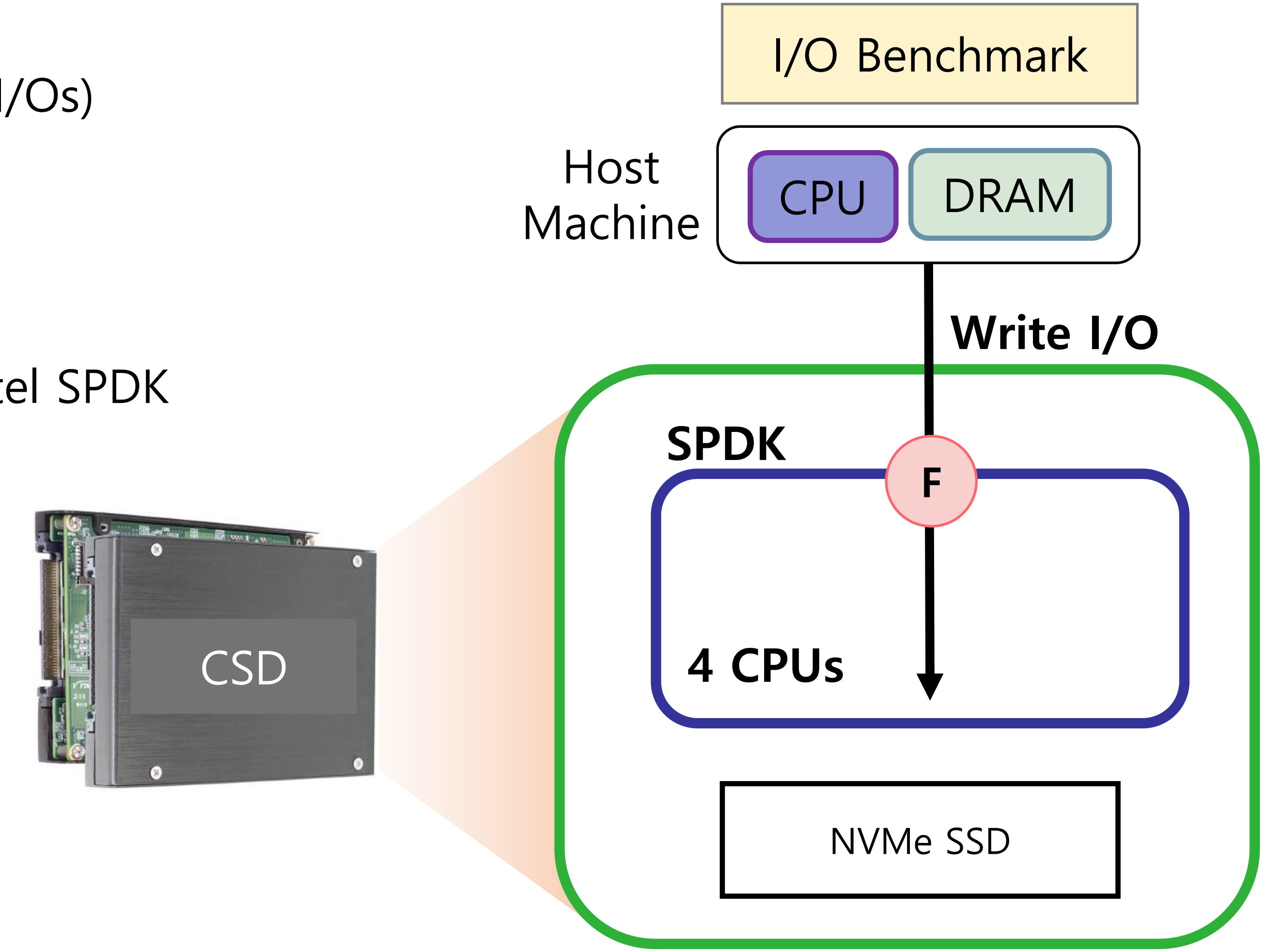
Experimental Setup

- Host machine
  - Running a db_bench
    (Two I/O threads issue write I/Os)
  - I/O request size = 16KB
- CSD
  - 4 Core device
  - Running a Linux OS using Intel SPDK
- Background task
  - Offline deduplication
  - Fingerprinting using a SHA-1 hash algorithm
  - Light deduplication : 1KB chunk size, SHA1 16 times
  - Heavy dedulication : 0.5KB chunk size, SHA1 32 times
- Comparisons
  - Only foreground I/O
  - Foreground I/O + Background task (light)
  - Foreground I/O + Background task (heavy)



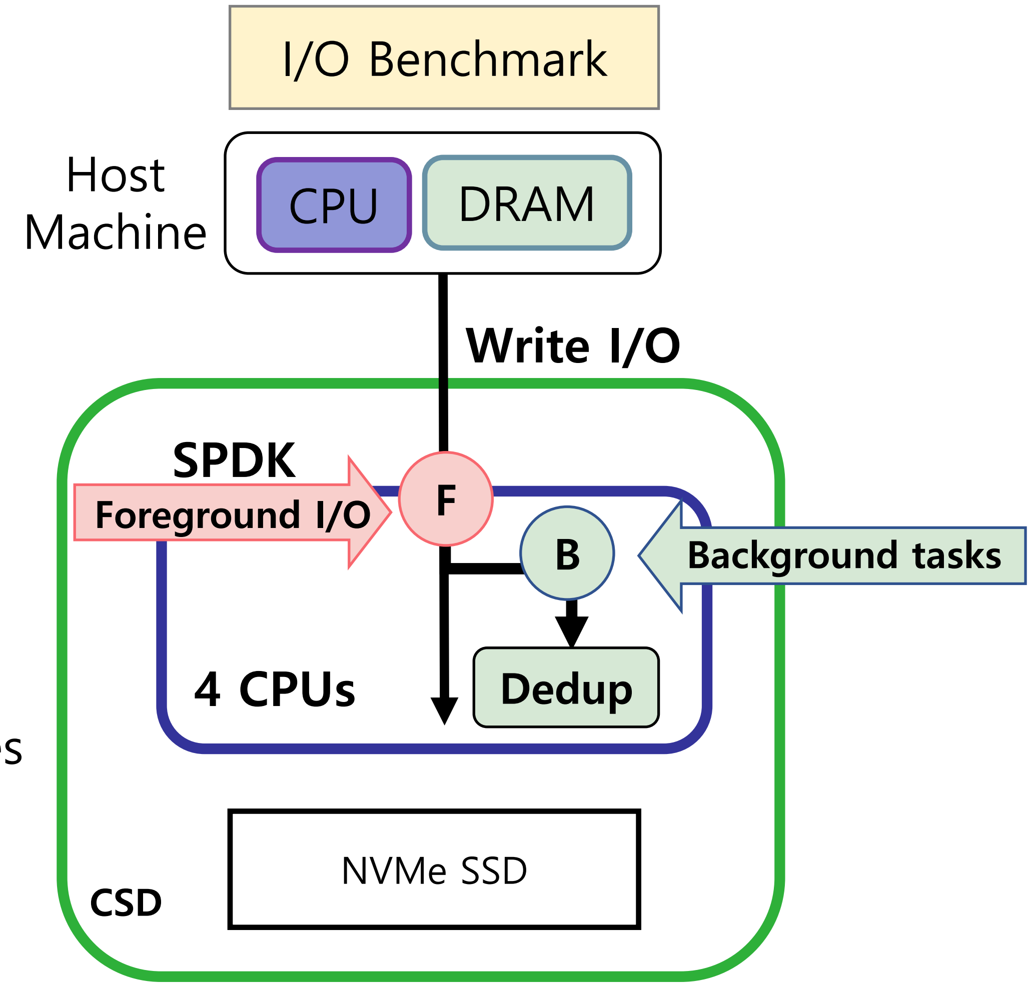
Experimental Setup

- Host machine
  - Running a db_bench (Two I/O threads issue write I/Os)
  - I/O request size = 16KB
- CSD
  - 4 Core device
  - Running a Linux OS using Intel SPDK



Experimental Setup

- ❑ Host machine
  - ❑ Running a db_bench
    (Two I/O threads issue write I/Os)
  - ❑ I/O request size = 16KB
- ❑ CSD
  - ❑ 4 Core device
  - ❑ Running a Linux OS using Intel SPDK
- ❑ Background task
  - ❑ Offline deduplication
  - ❑ Fingerprinting using a SHA-1 hash algorithm
  - ❑ Light deduplication : 1KB chunk size, SHA1 16 times
  - ❑ Heavy dedulication : 0.5KB chunk size, SHA1 32 times



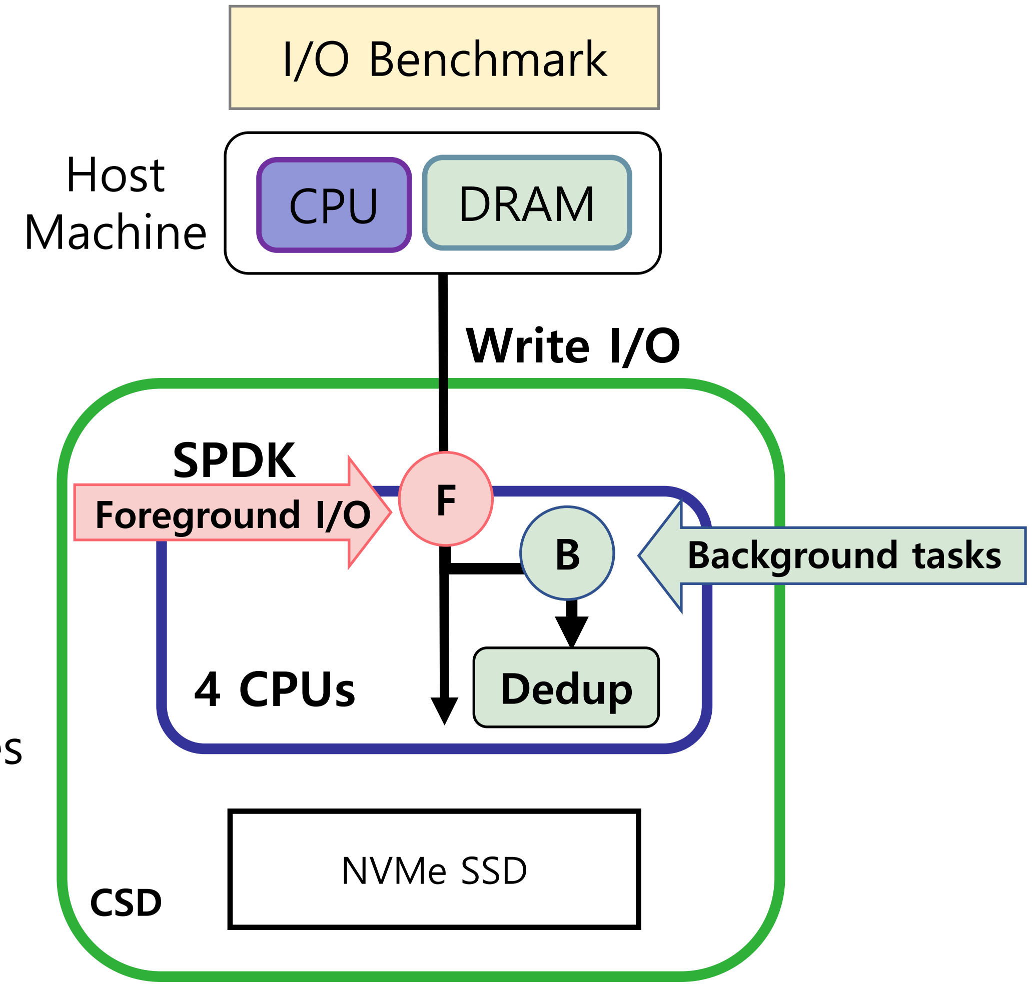
Experimental Setup

- Host machine
  - Running a db_bench
    (Two I/O threads issue write I/Os)
  - I/O request size = 16KB

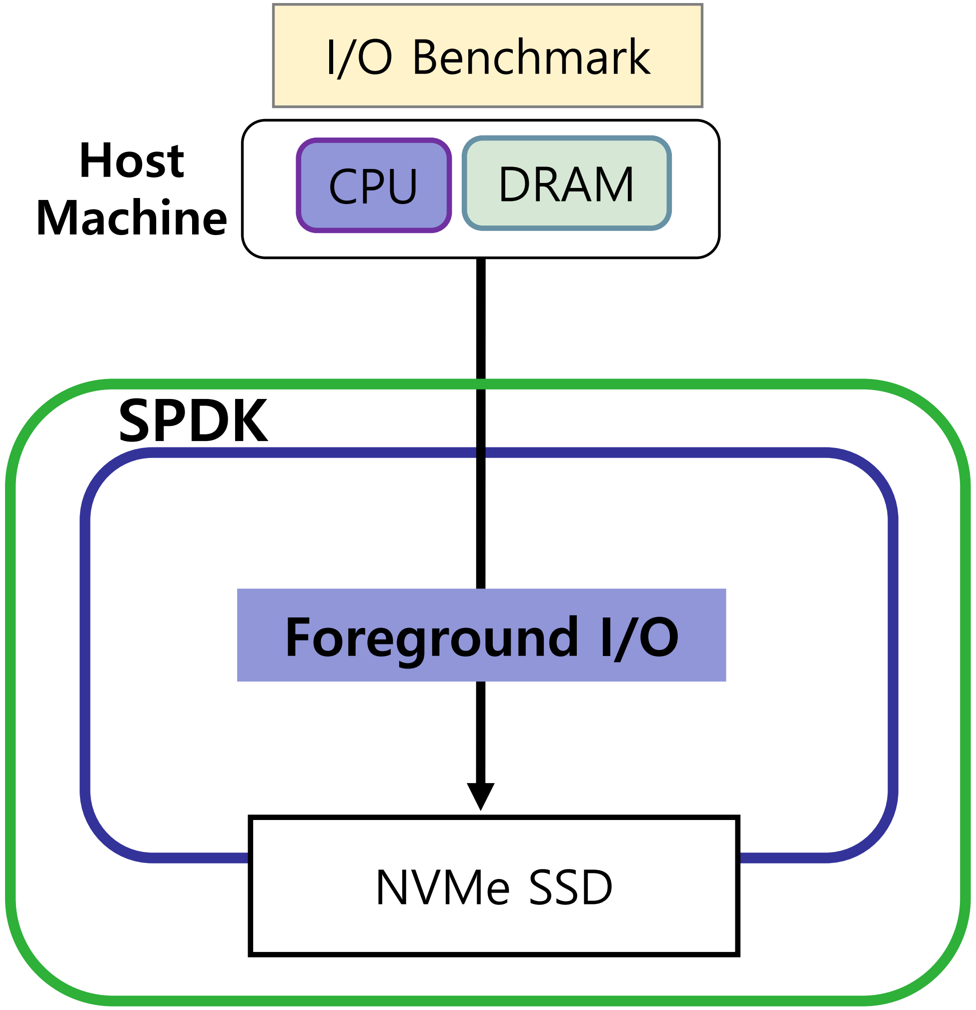
- CSD
  - 4 Core device
  - Running a Linux OS using Intel SPDK

- Background task
  - Offline deduplication
  - Fingerprinting using a SHA-1 hash algorithm
  - Light deduplication : 1KB chunk size, SHA1 16 times
  - Heavy dedulication : 0.5KB chunk size, SHA1 32 times

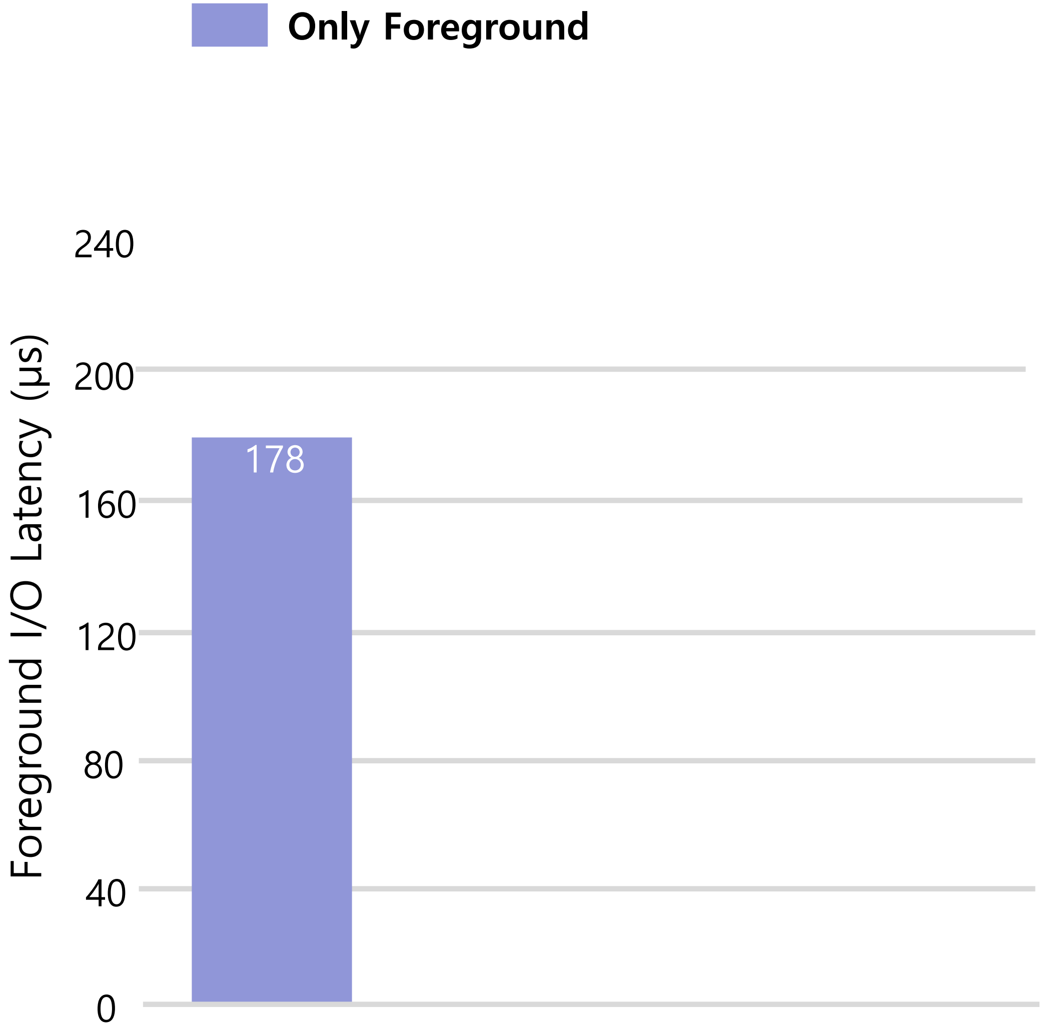
- Comparisons
  - Only foreground I/O
  - Foreground I/O + Background task (light)
  - Foreground I/O + Background task (heavy)



Motivation

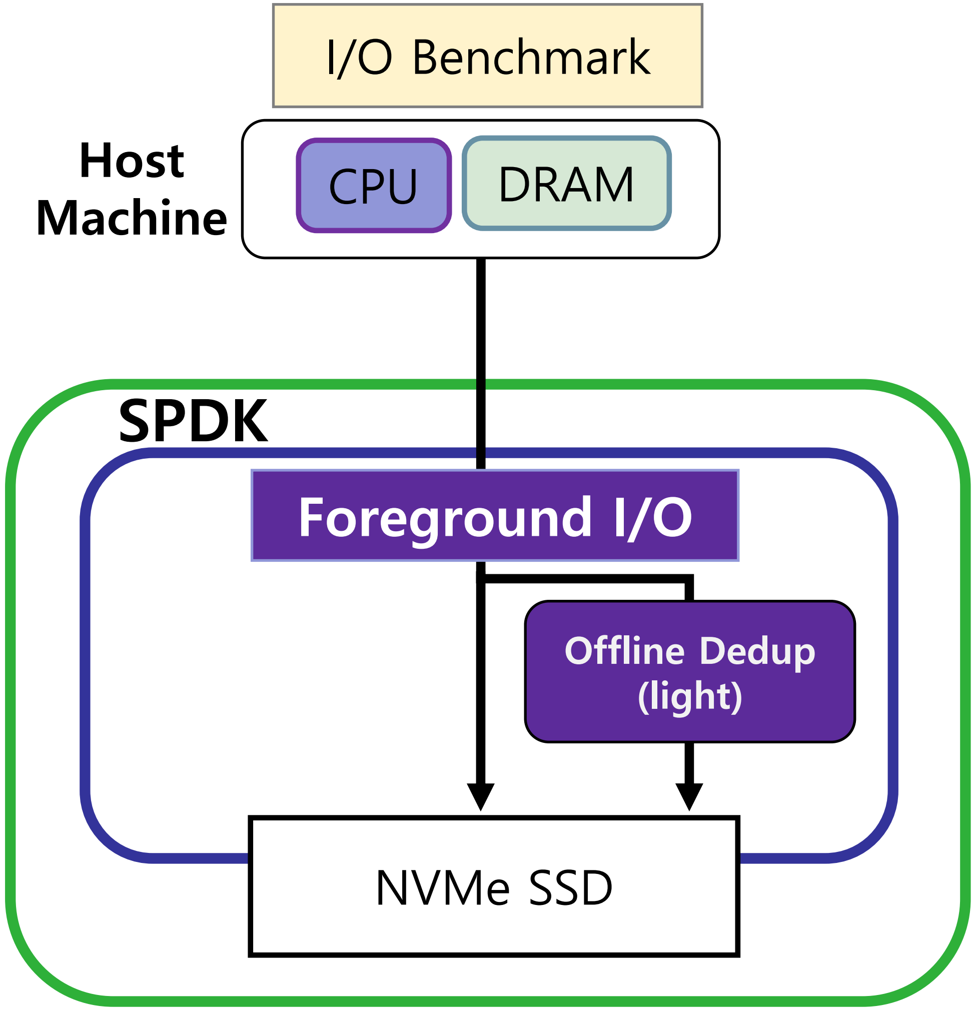


I/O Benchmark
Host Machine
CPU
DRAM
SPDK
Foreground I/O
NVMe SSD

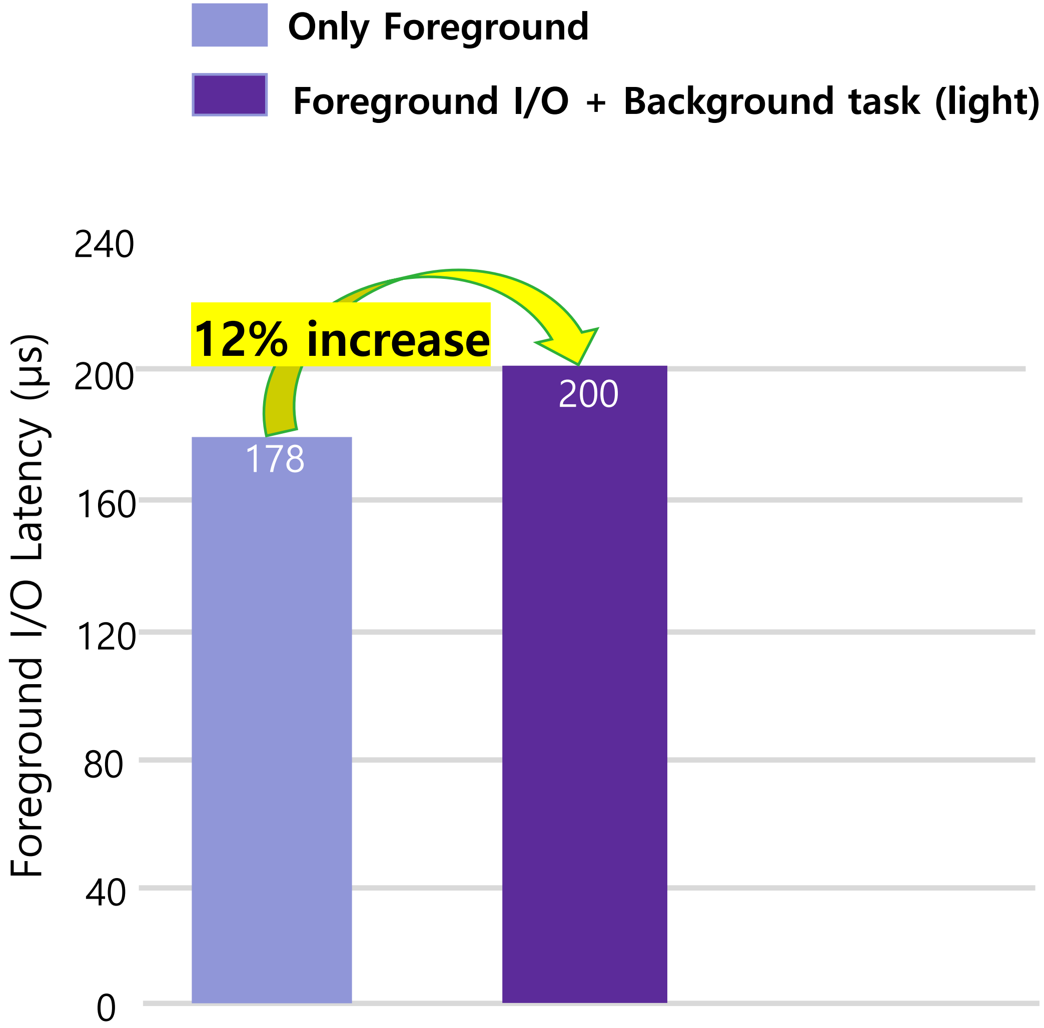


Only Foreground
Foreground I/O Latency (μs)
240
200
160
120
80
40
0
178

Motivation

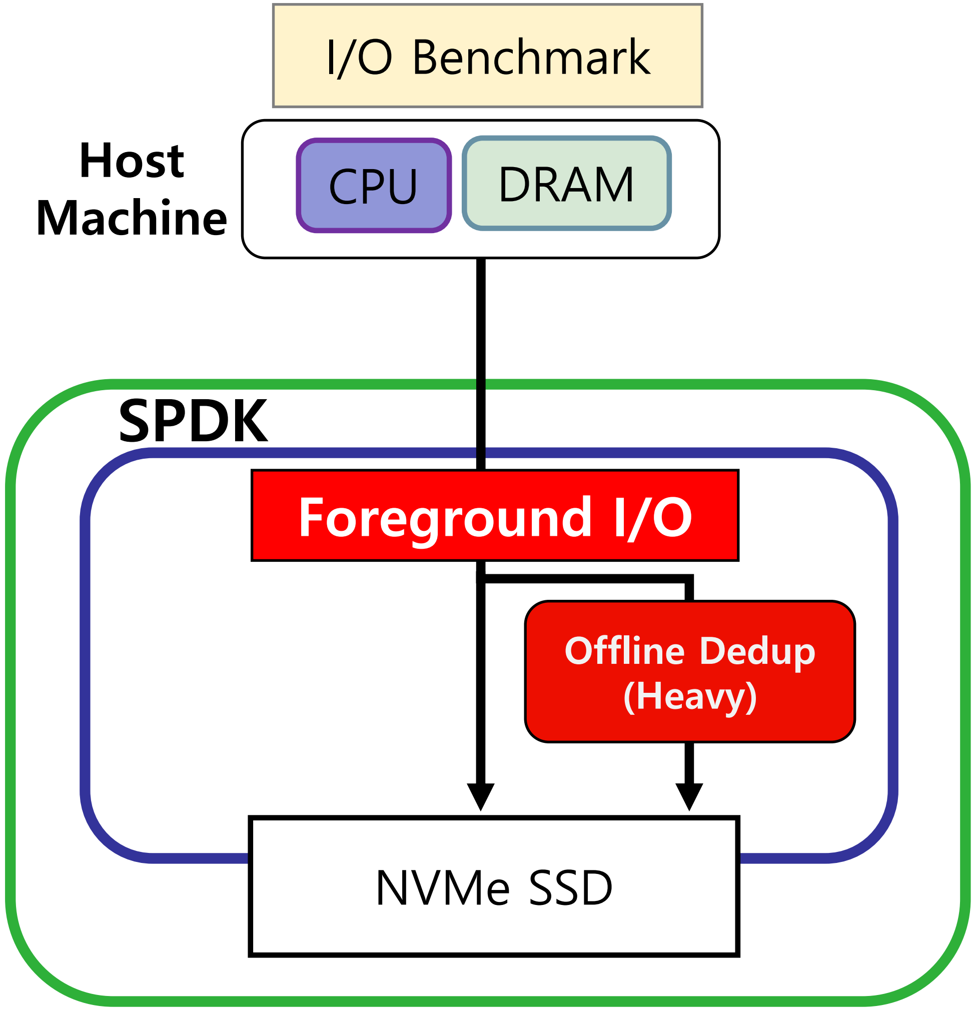


I/O Benchmark
Host Machine
CPU
DRAM
SPDK
Foreground I/O
Offline Dedup (light)
NVMe SSD

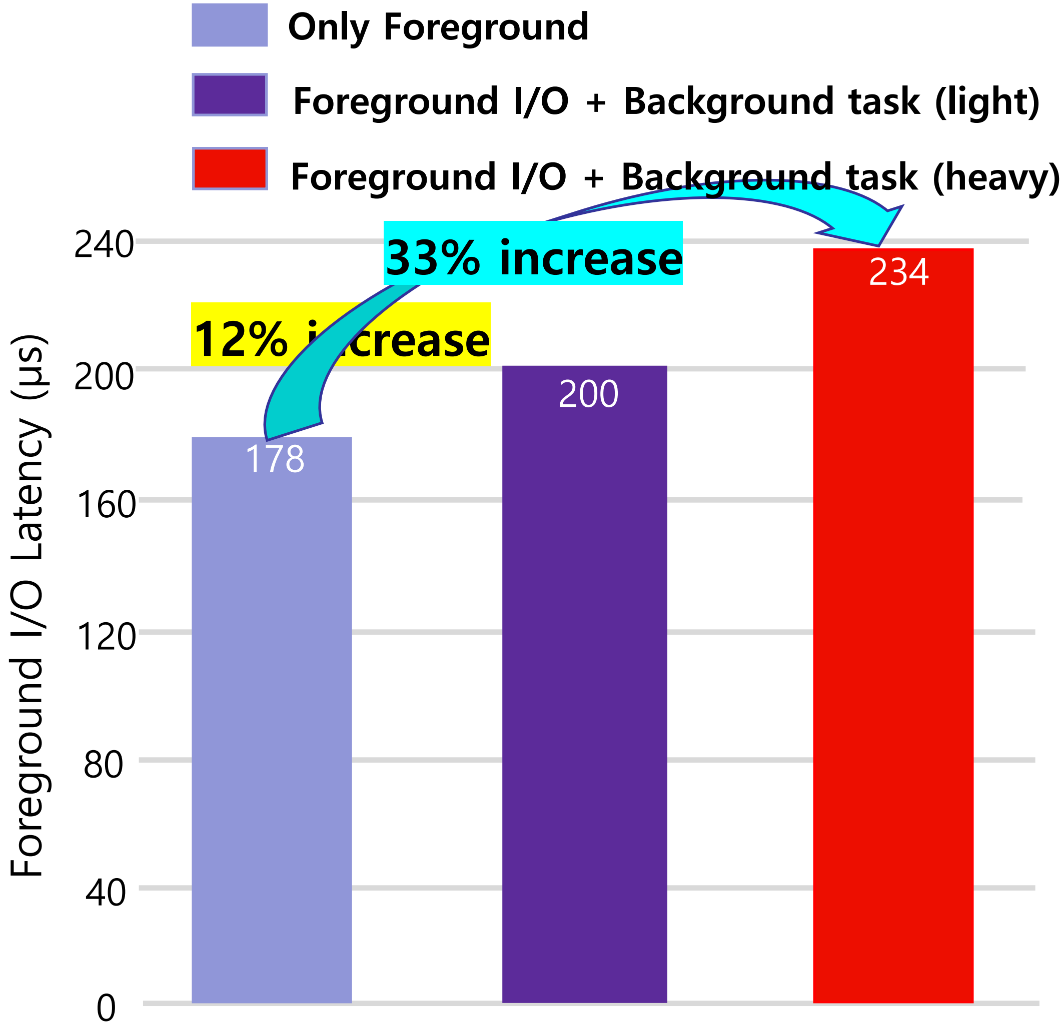


Only Foreground
Foreground I/O + Background task (light)
Foreground I/O Latency (μs)
240
200
160
120
80
40
0
12% increase
178
200

Motivation

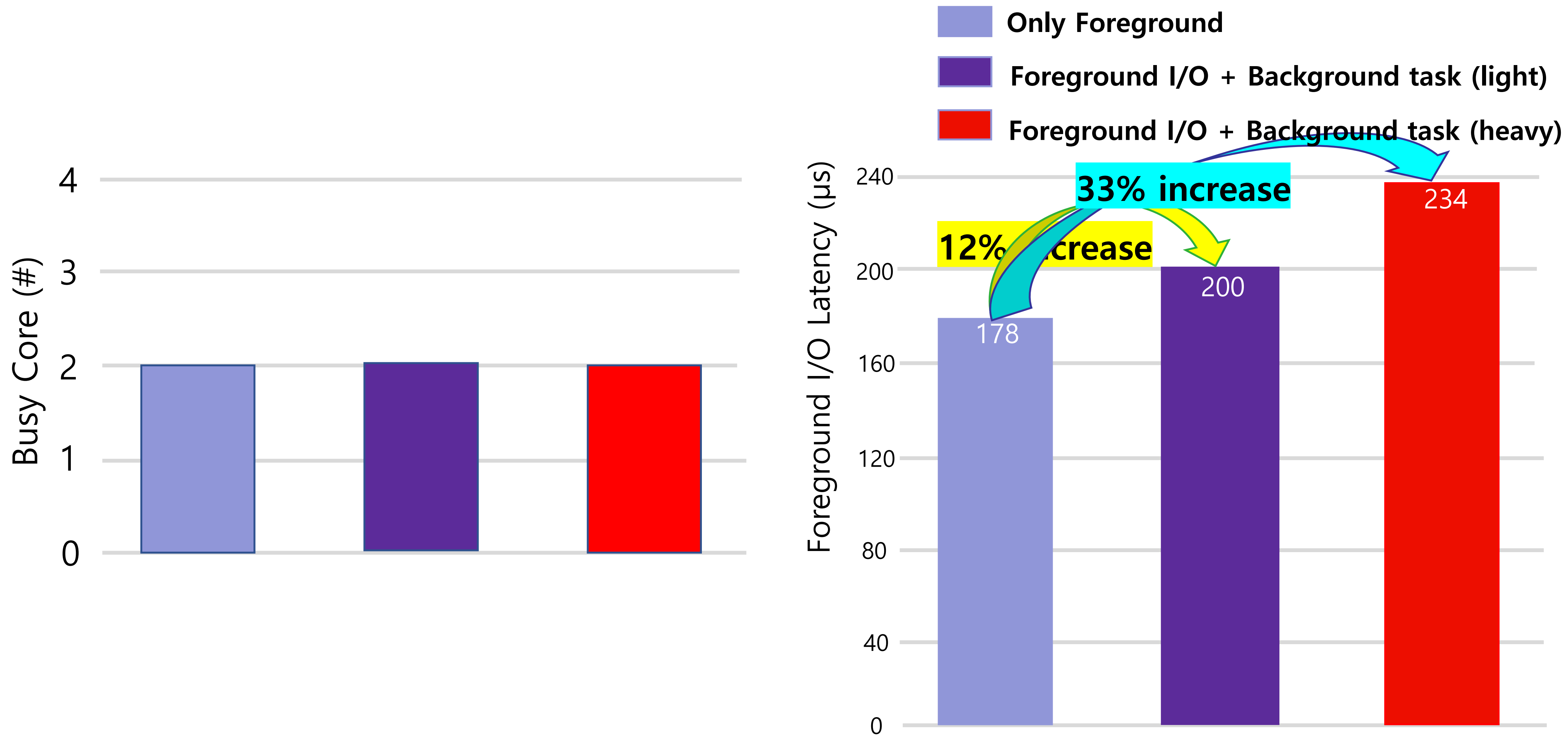


I/O Benchmark
Host Machine
CPU
DRAM
SPDK
Foreground I/O
Offline Dedup (Heavy)
NVMe SSD

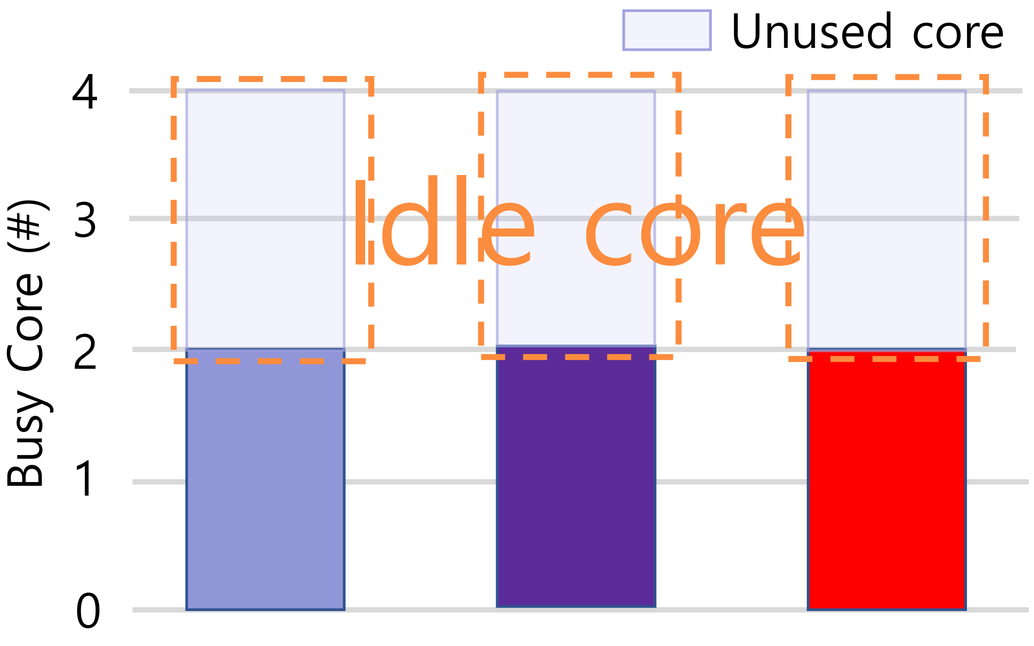


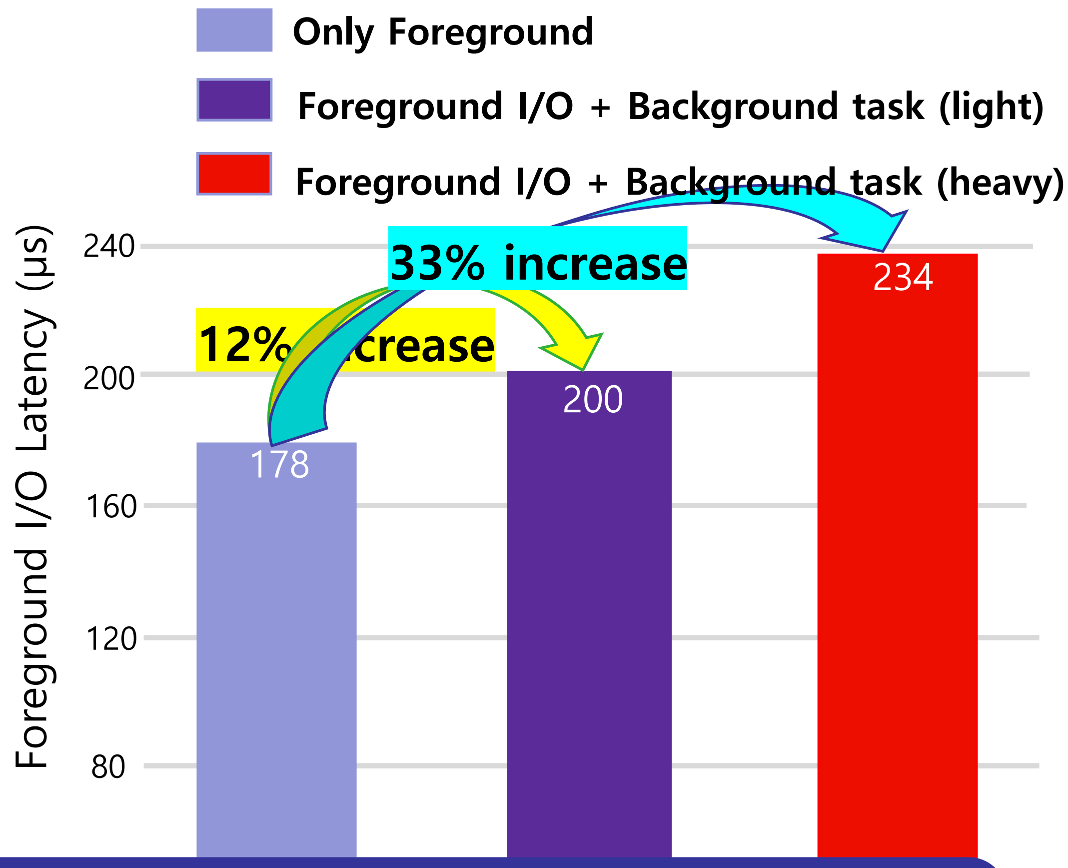
Only Foreground
Foreground I/O + Background task (light)
Foreground I/O + Background task (heavy)
33% increase
12% increase
Foreground I/O Latency (μs)
240
200
160
120
80
40
0
178
200
234

Motivation


Only Foreground
Foreground I/O + Background task (light)
Foreground I/O + Background task (heavy)
Busy Core (#)
4
3
2
1
0
Foreground I/O Latency (µs)
240
200
160
120
80
40
0
33% increase
12% increase
178
200
234

Motivation





Only two cores are fully-utilized.
That is, the remaining cores are under-utilized.

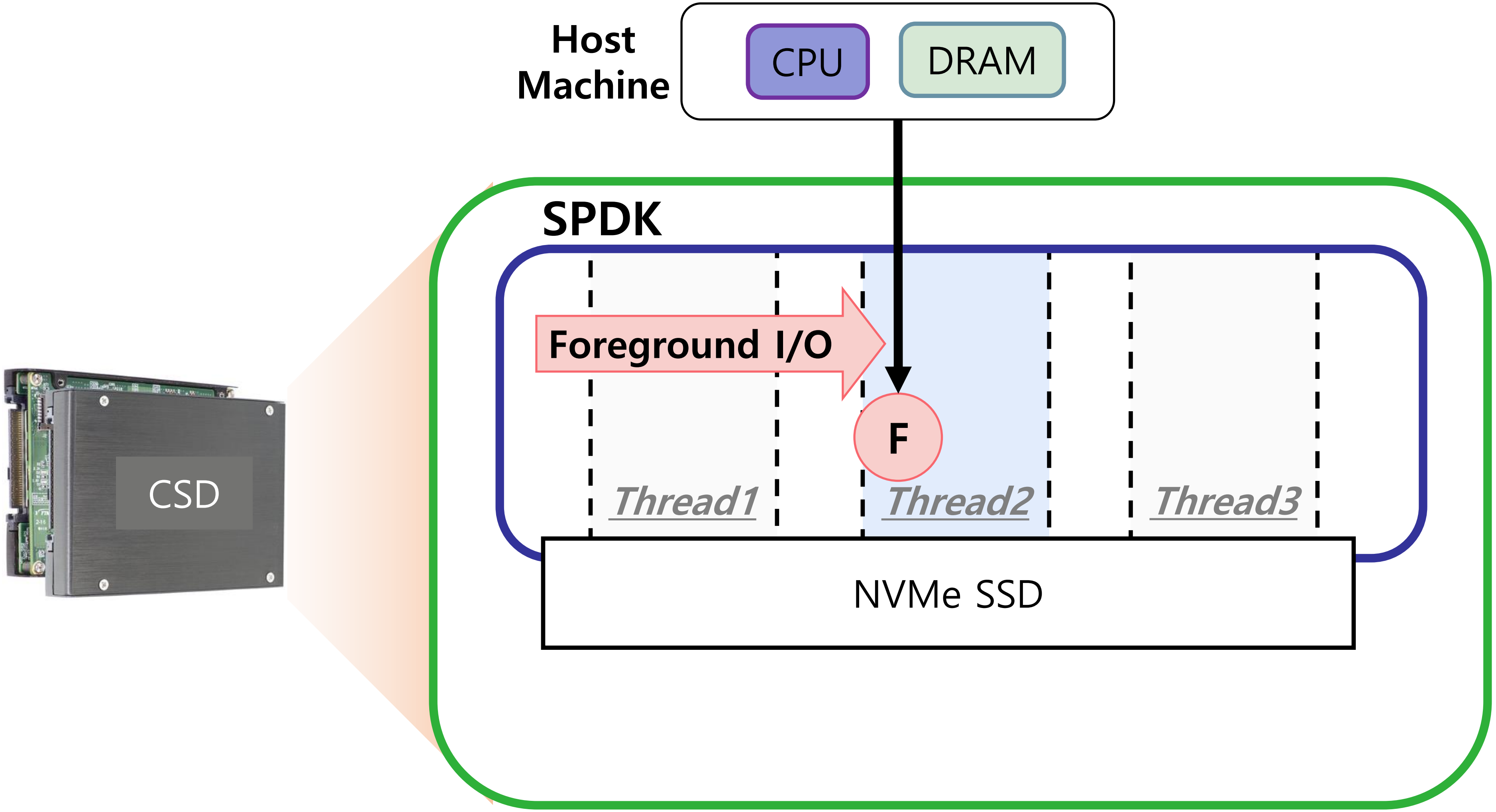
Problem Definition

① SPDK is not aware of background tasks.
② SPDK cannot perform dynamic task scheduling considering the load of each CPU core.

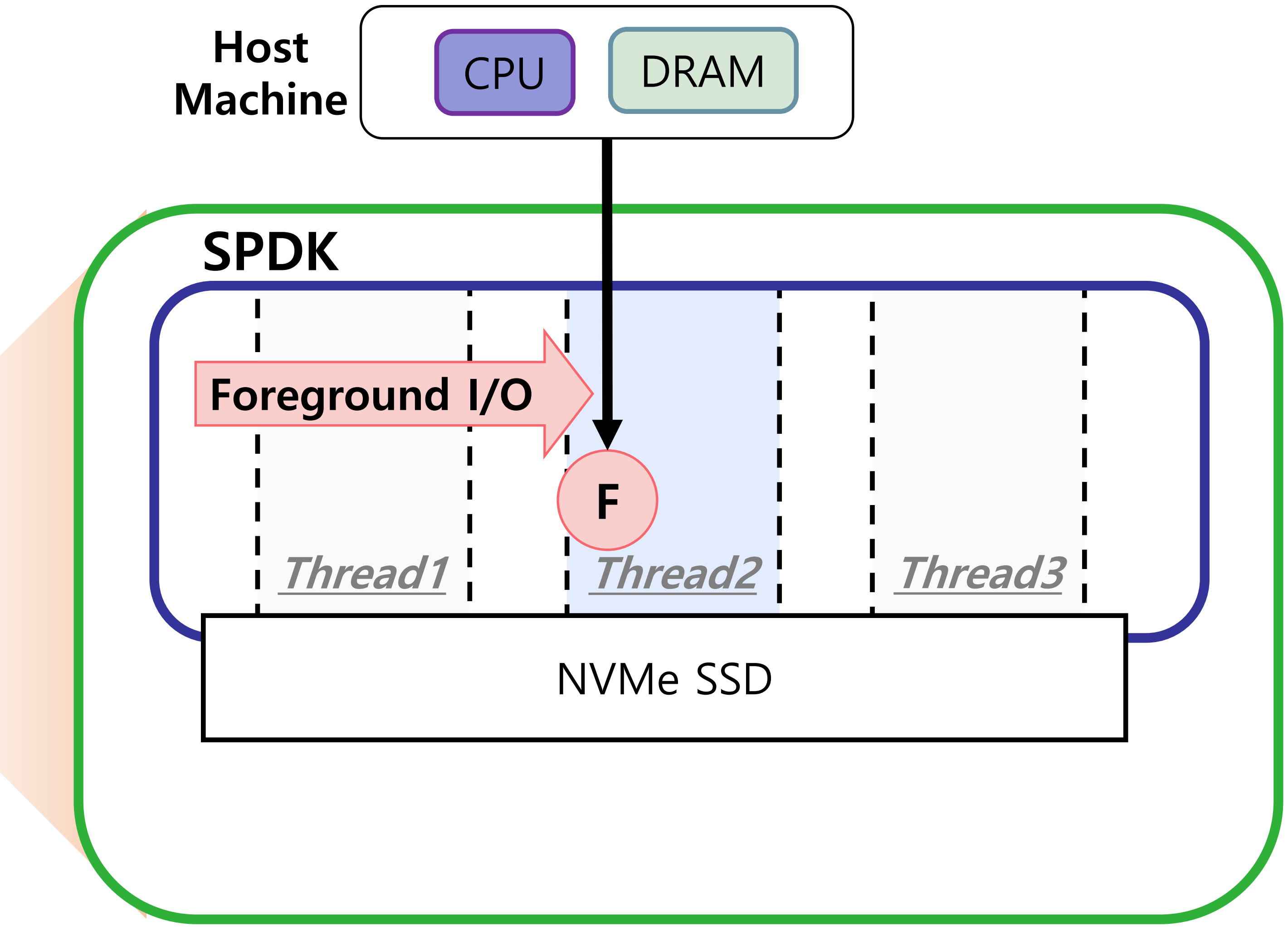
→ BTS : Background Task-aware Scheduler

BTS : Background Task-Aware Scheduler

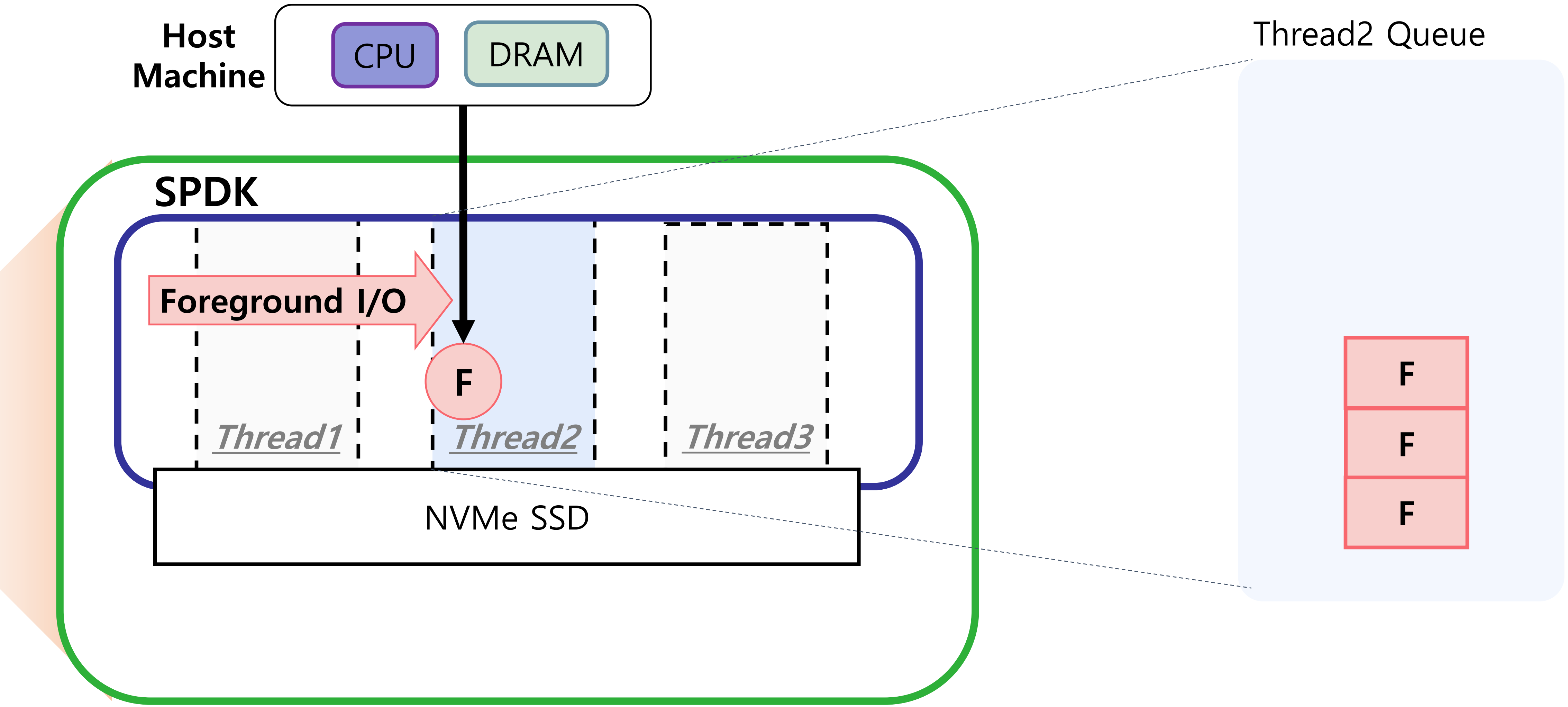
SPDK Problem



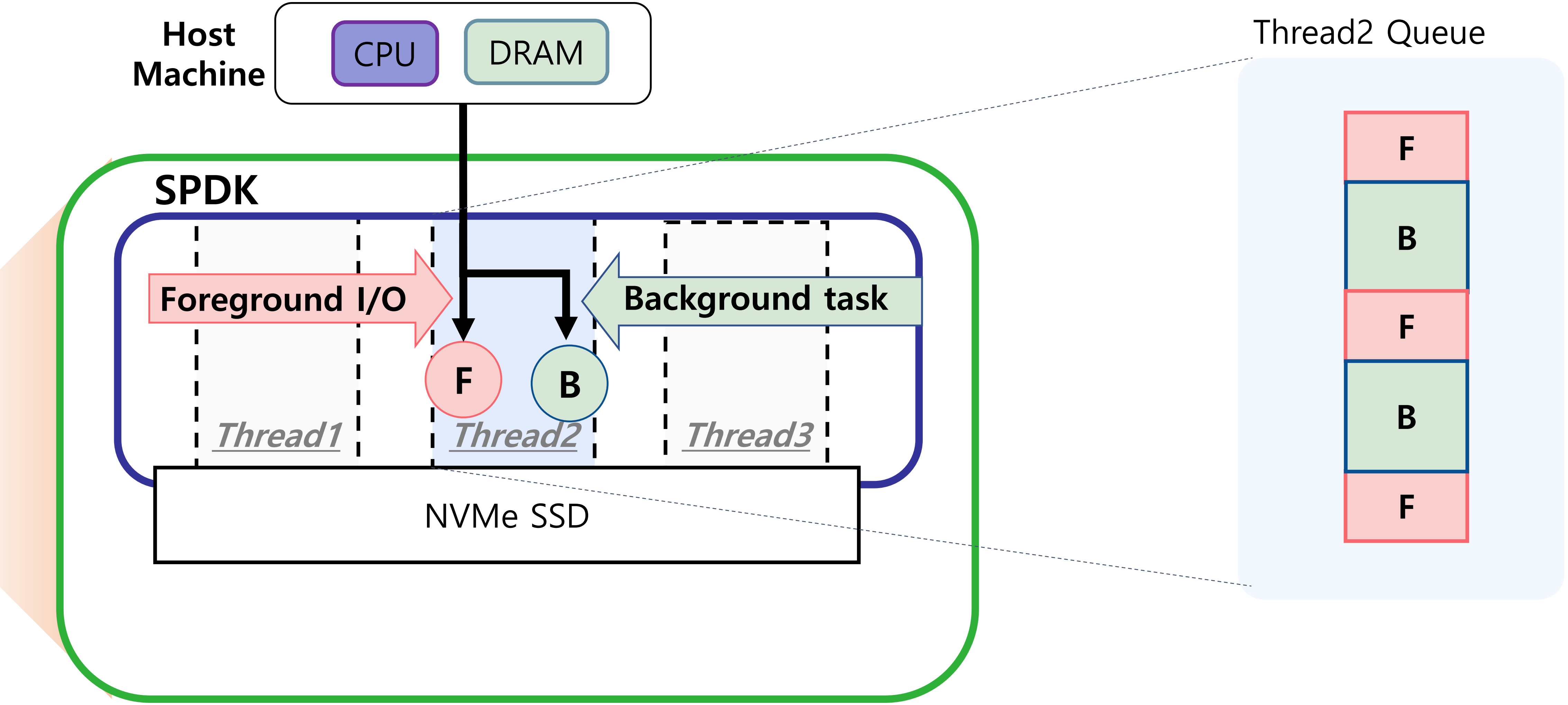
SPDK Problem



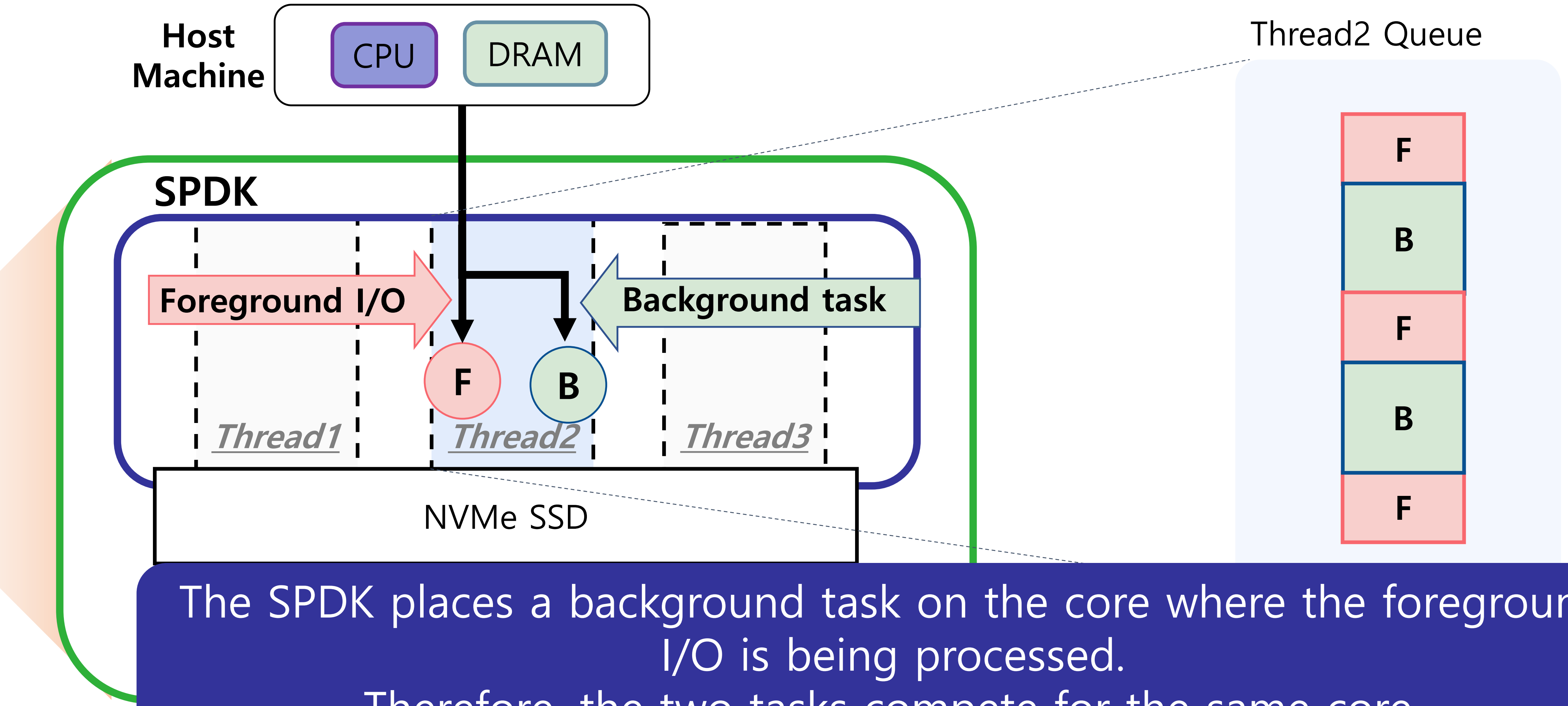
SPDK Problem



SPDK Problem

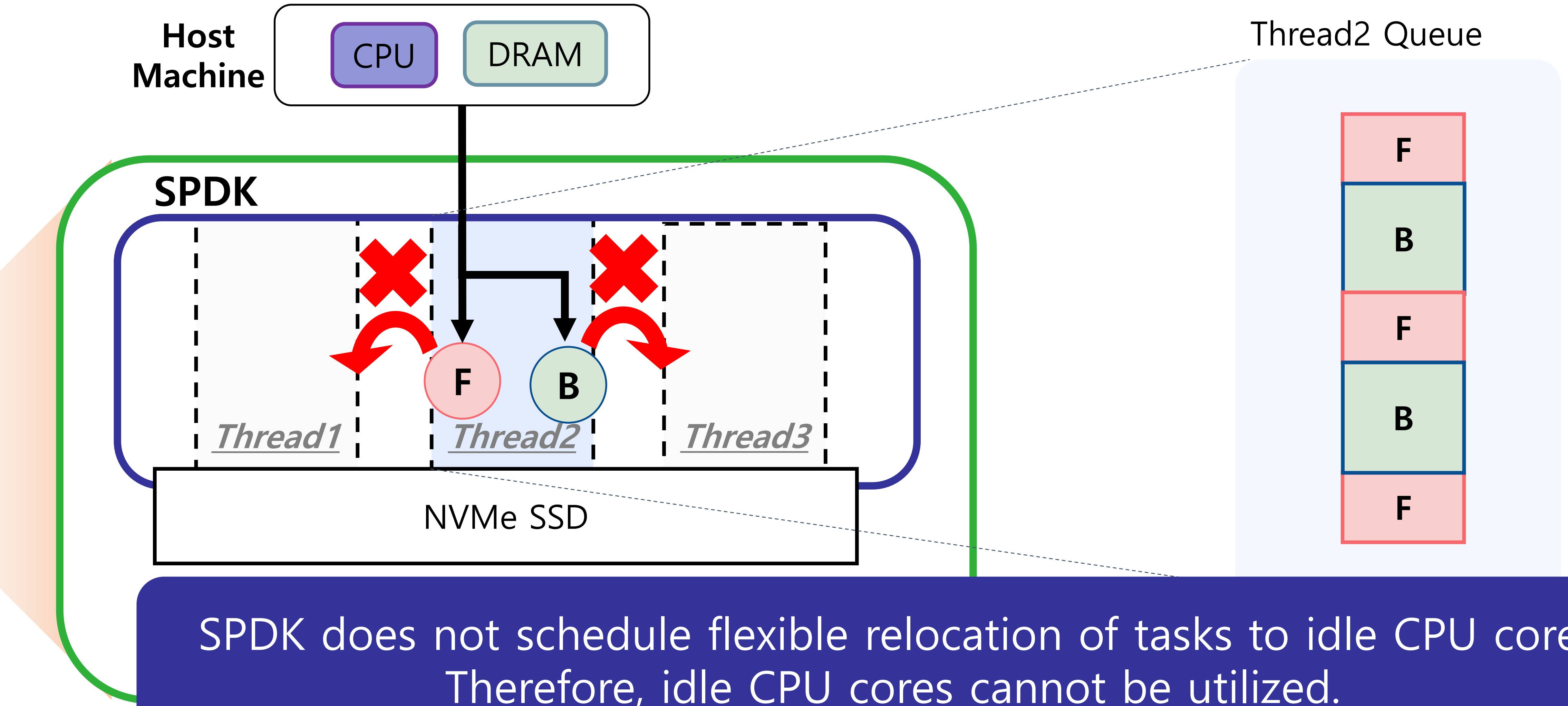


SPDK Problem



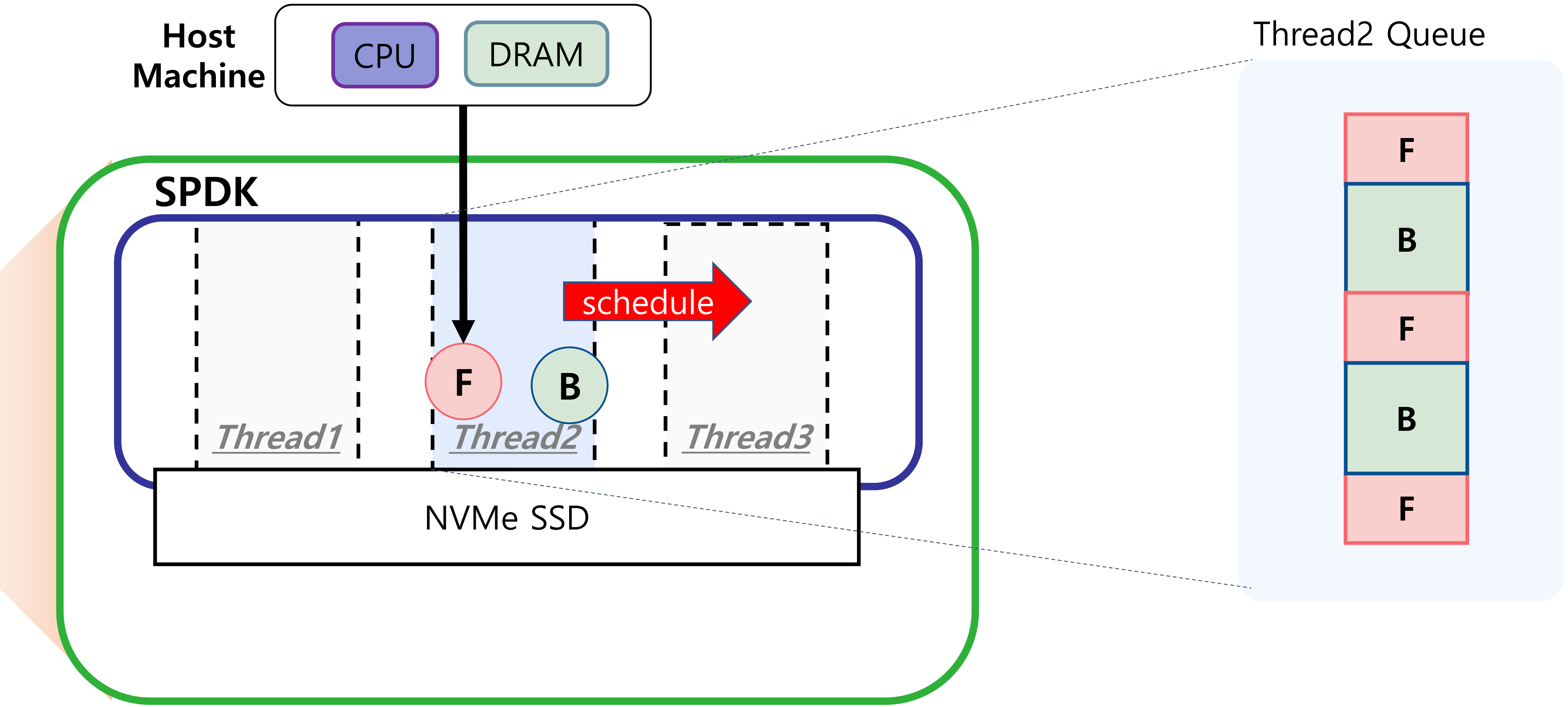
The SPDK places a background task on the core where the foreground I/O is being processed.
Therefore, the two tasks compete for the same core.

SPDK Problem

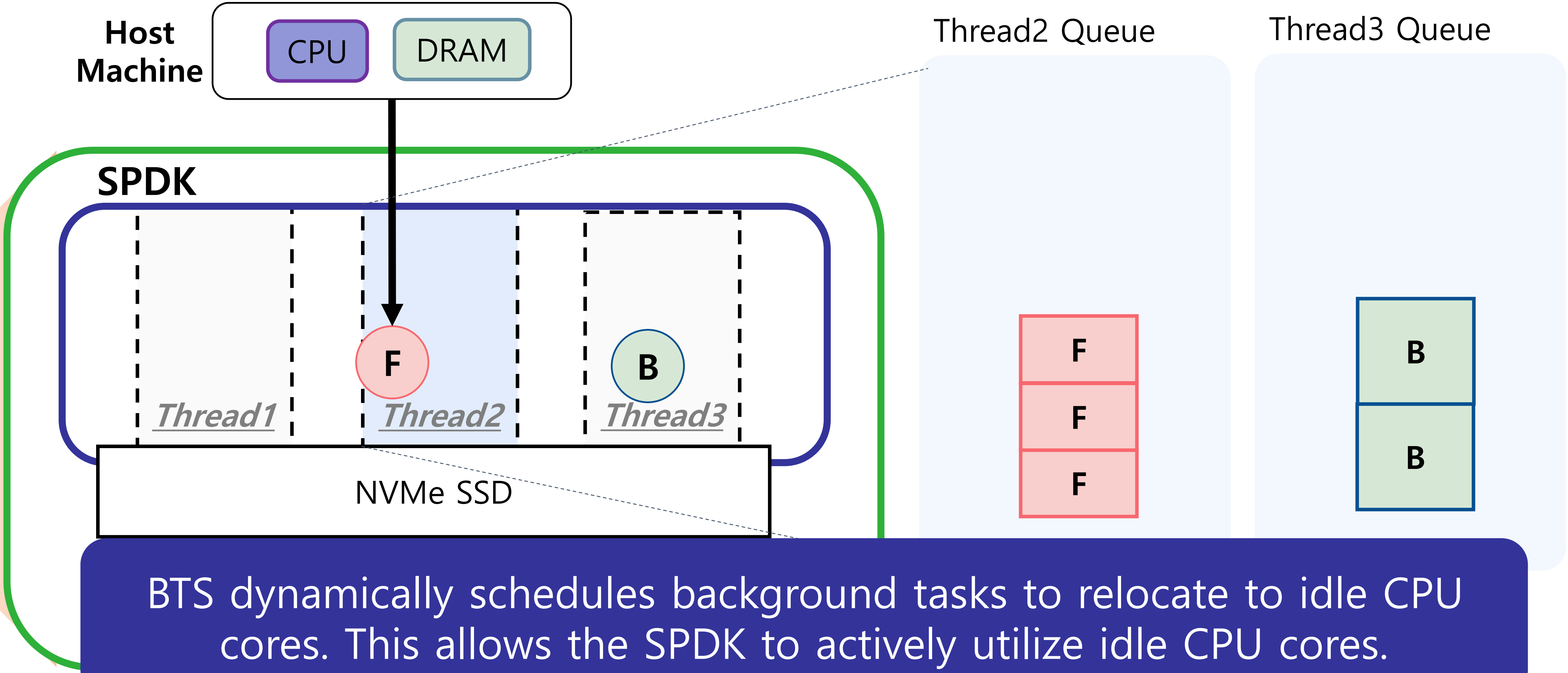


SPDK does not schedule flexible relocation of tasks to idle CPU core.
Therefore, idle CPU cores cannot be utilized.

SPDK Problem



SPDK Problem

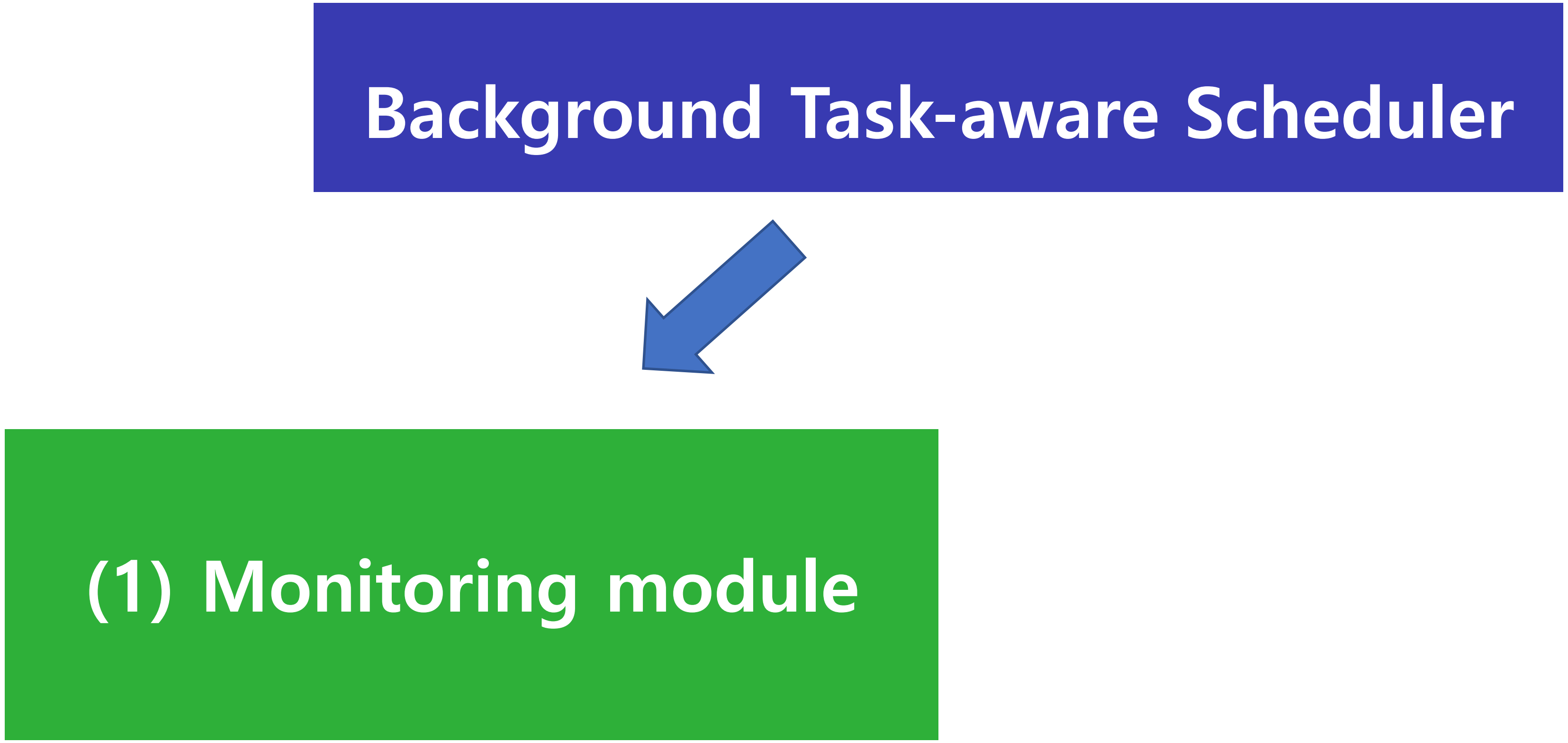


BTS dynamically schedules background tasks to relocate to idle CPU cores. This allows the SPDK to actively utilize idle CPU cores.

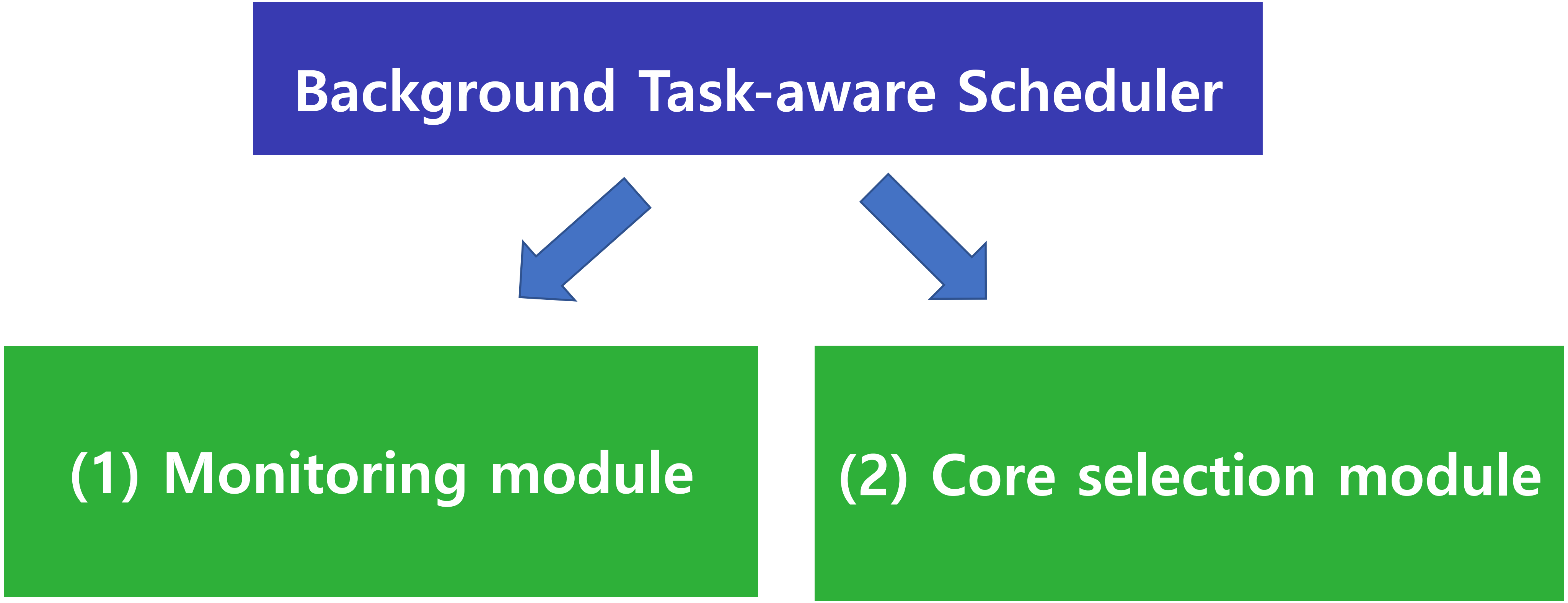
BTS Scheduler

Background Task-aware Scheduler

BTS Scheduler

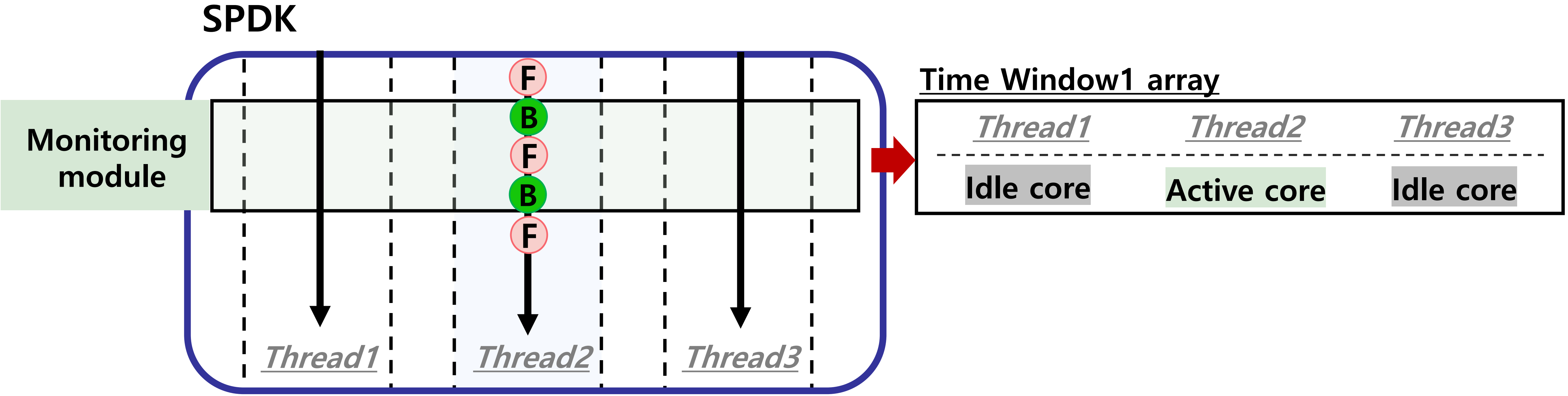


BTS Scheduler



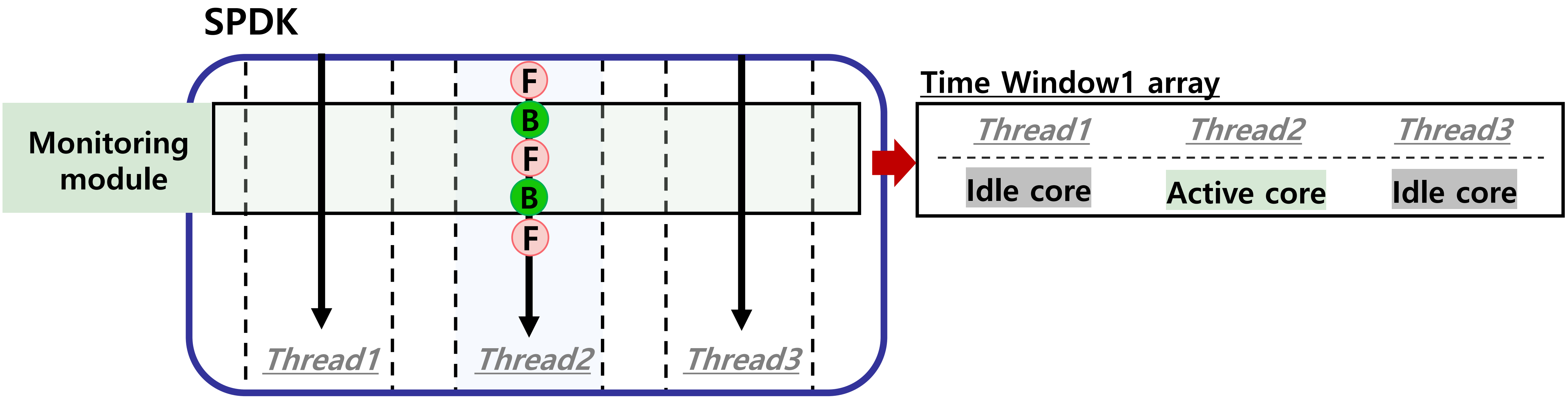
1) Monitoring Module

- Monitors the utilization of each core
  - Because SPDK randomly changes the core that handles foreground I/O



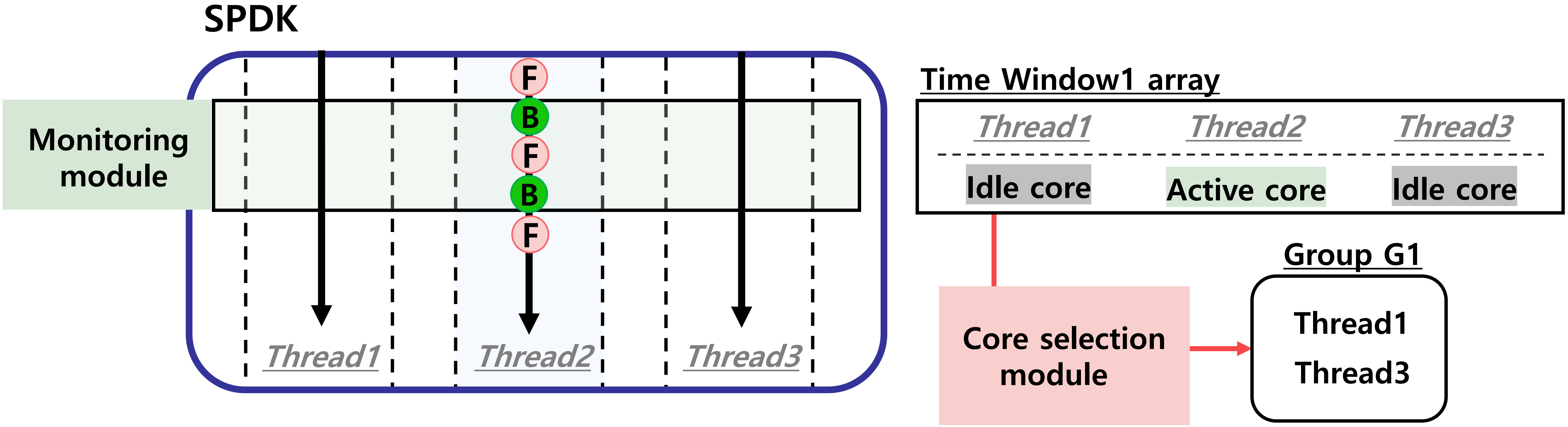
1) Monitoring Module

- Monitors the utilization of each core
  - Because SPDK randomly changes the core that handles foreground I/O
- The monitoring module periodically tracks the utilization of active cores
  - Active core is CPU core that processes at least one foreground I/O



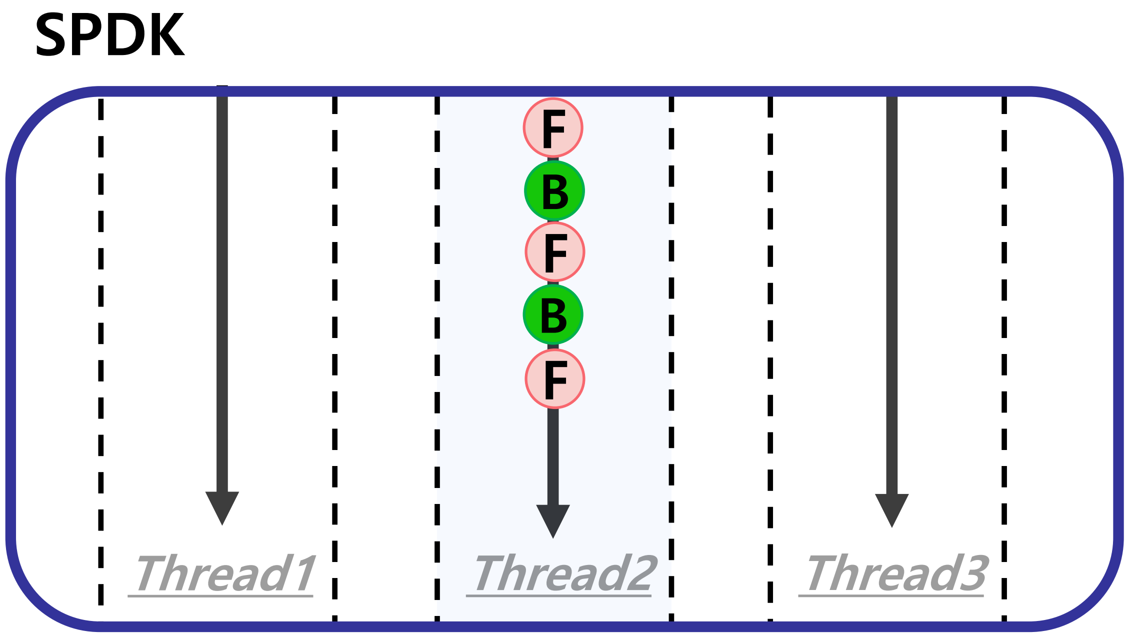
2) Core Selection Module

- ❑ Selects a core with low utilization and move background task to that core
- ❑ Builds an idle core group (G) based on the CPU utilization of each core



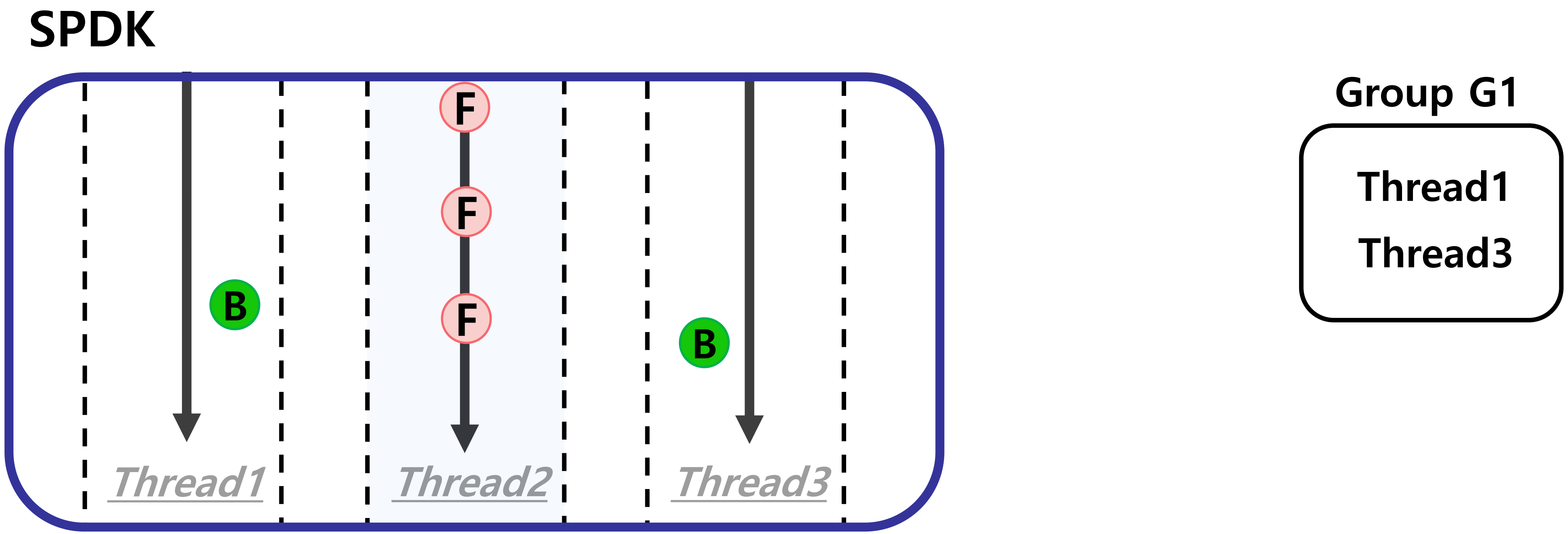
2) Core Selection Module

- ❑ Selects a core with low utilization and move background task to that core
- ❑ Builds an idle core group (G) based on the CPU utilization of each core
- ❑ Selects idle cores to execute background tasks from the idle core group (G)



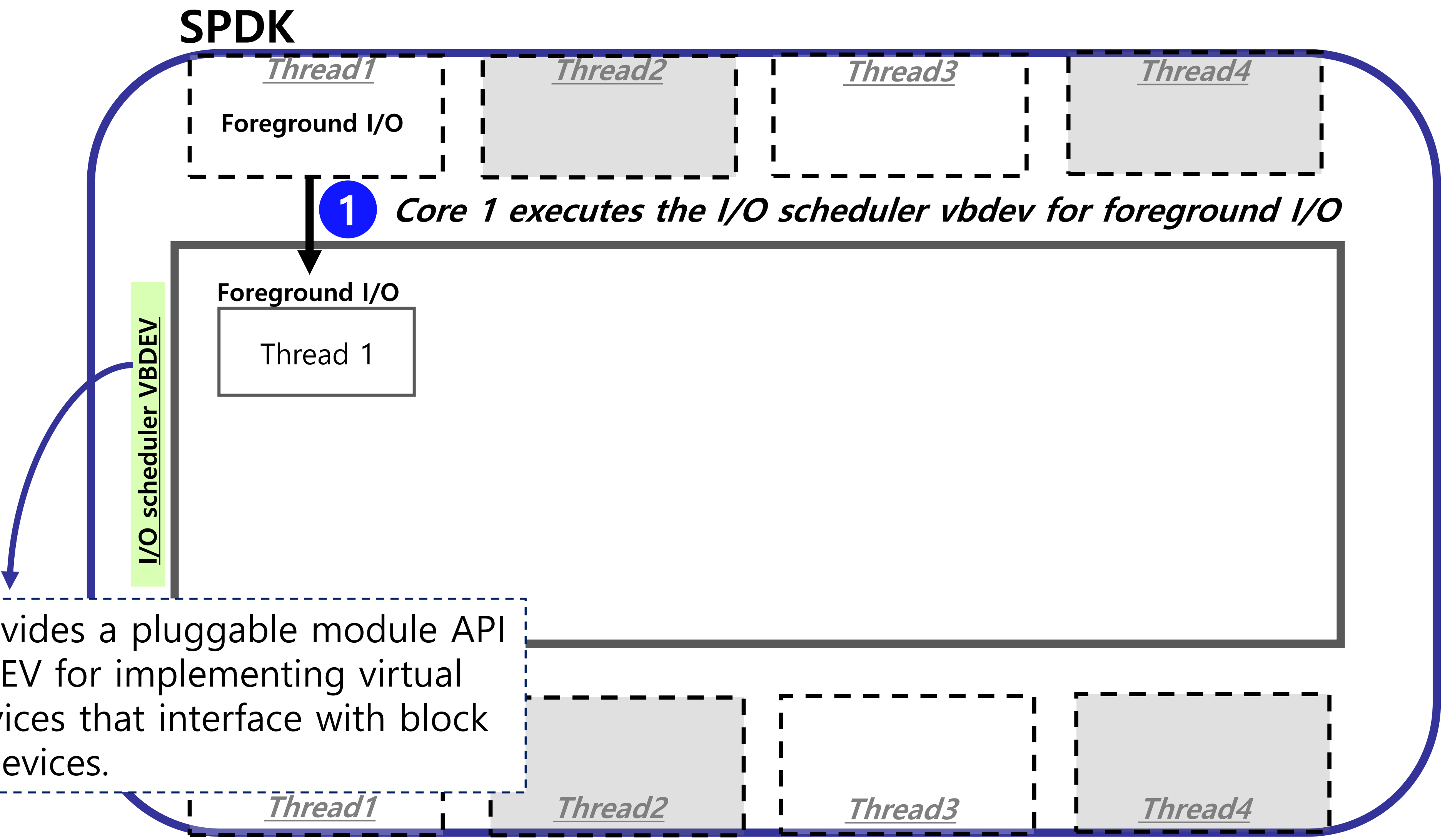
2) Core Selection Module

- ❑ Selects a core with low utilization and move background task to that core
- ❑ Builds an idle core group (G) based on the CPU utilization of each core
- ❑ Selects idle cores to execute background tasks from the idle core group (G)



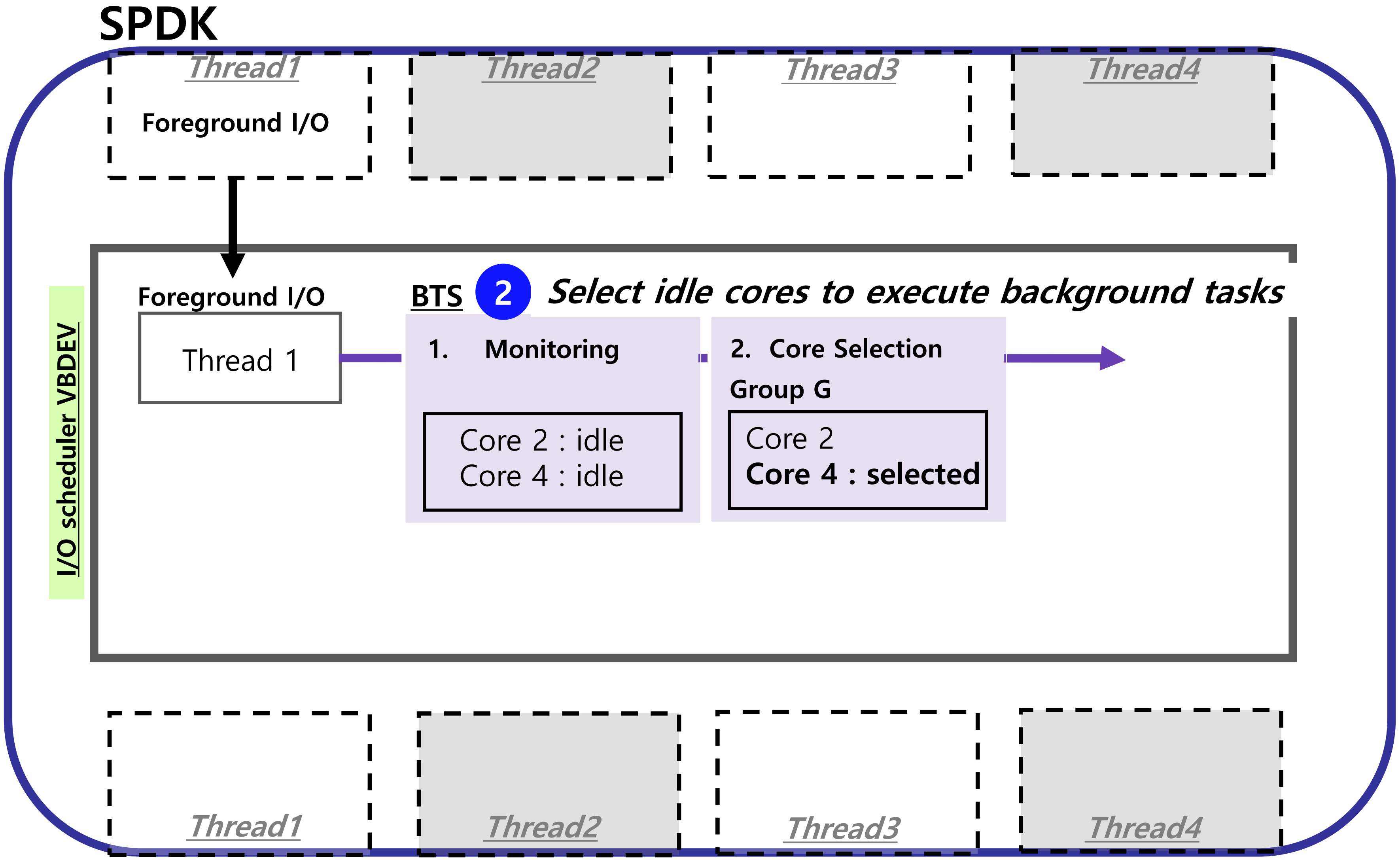
Implementation

Implementation

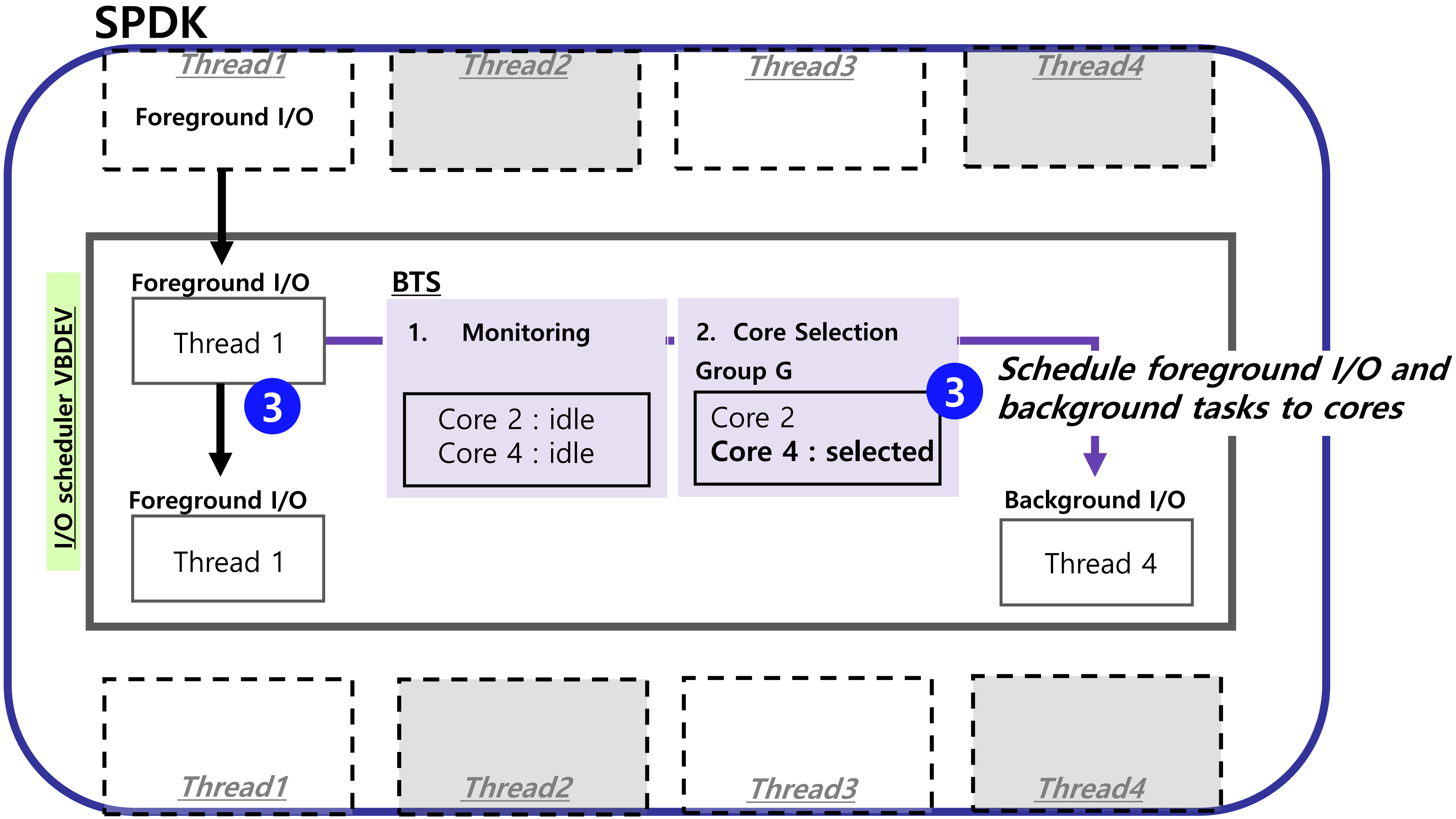


SPDK provides a pluggable module API called BDEV for implementing virtual block devices that interface with block storage devices.

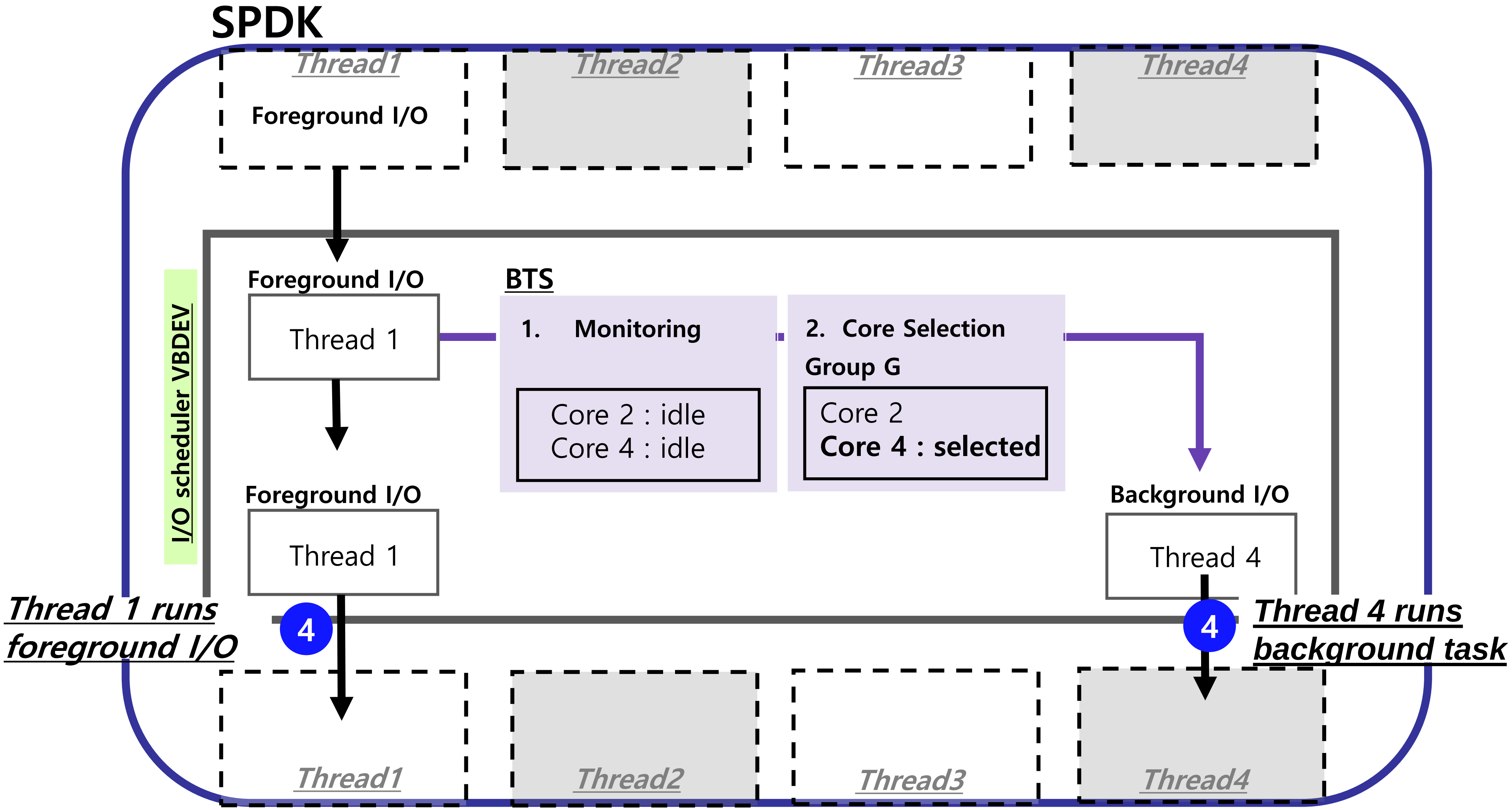
Implementation



Implementation

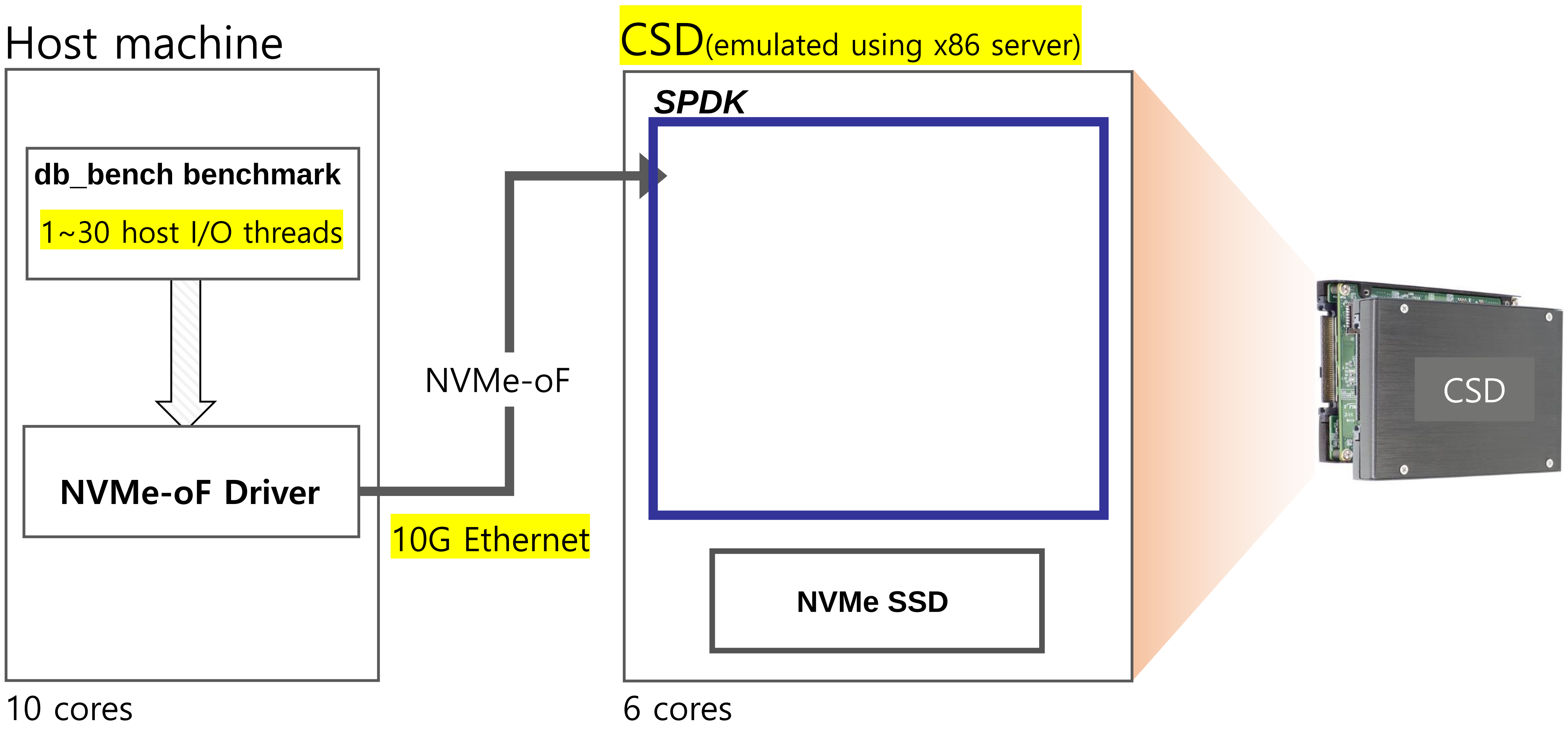


Implementation

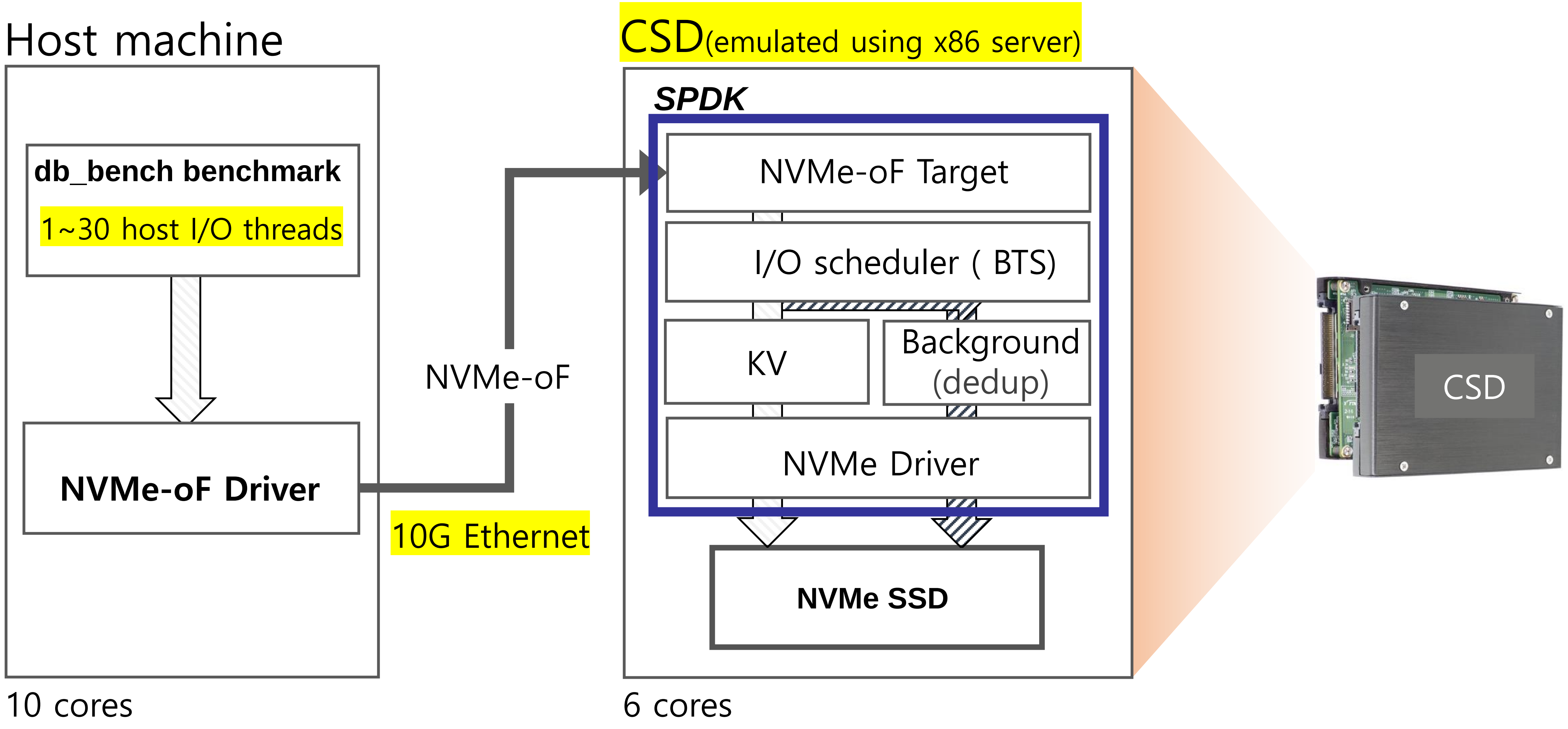


Evaluation

Experimental Setup



Experimental Setup

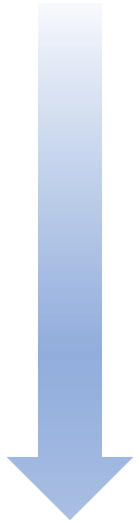


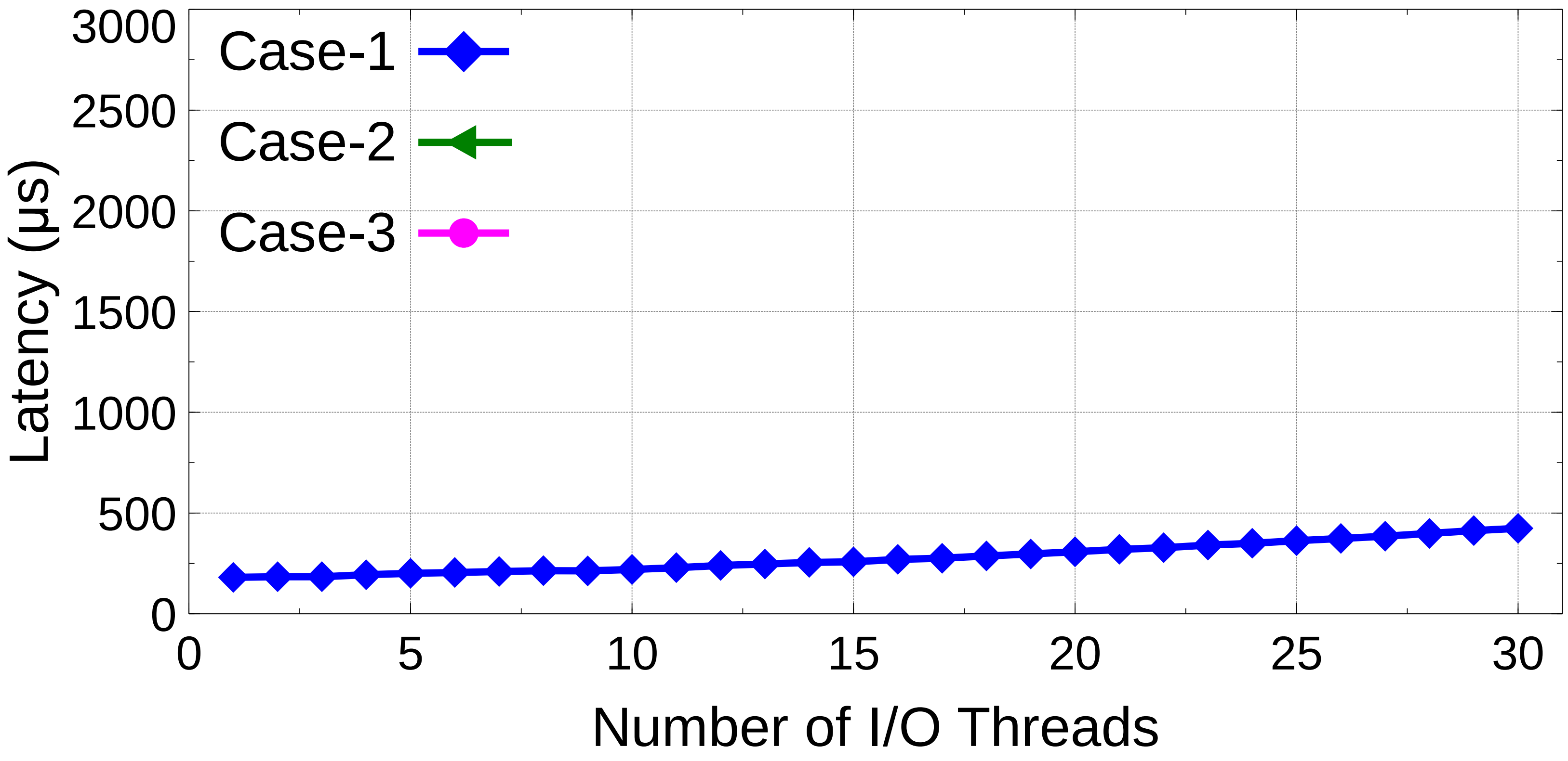
Experimental Setup

- Comparisons
  - Case1: Only foreground I/O
  - Case2: Foreground I/O + Deduplication without BTS
  - Case3: Foreground I/O + Deduplication with BTS

Put() Performance

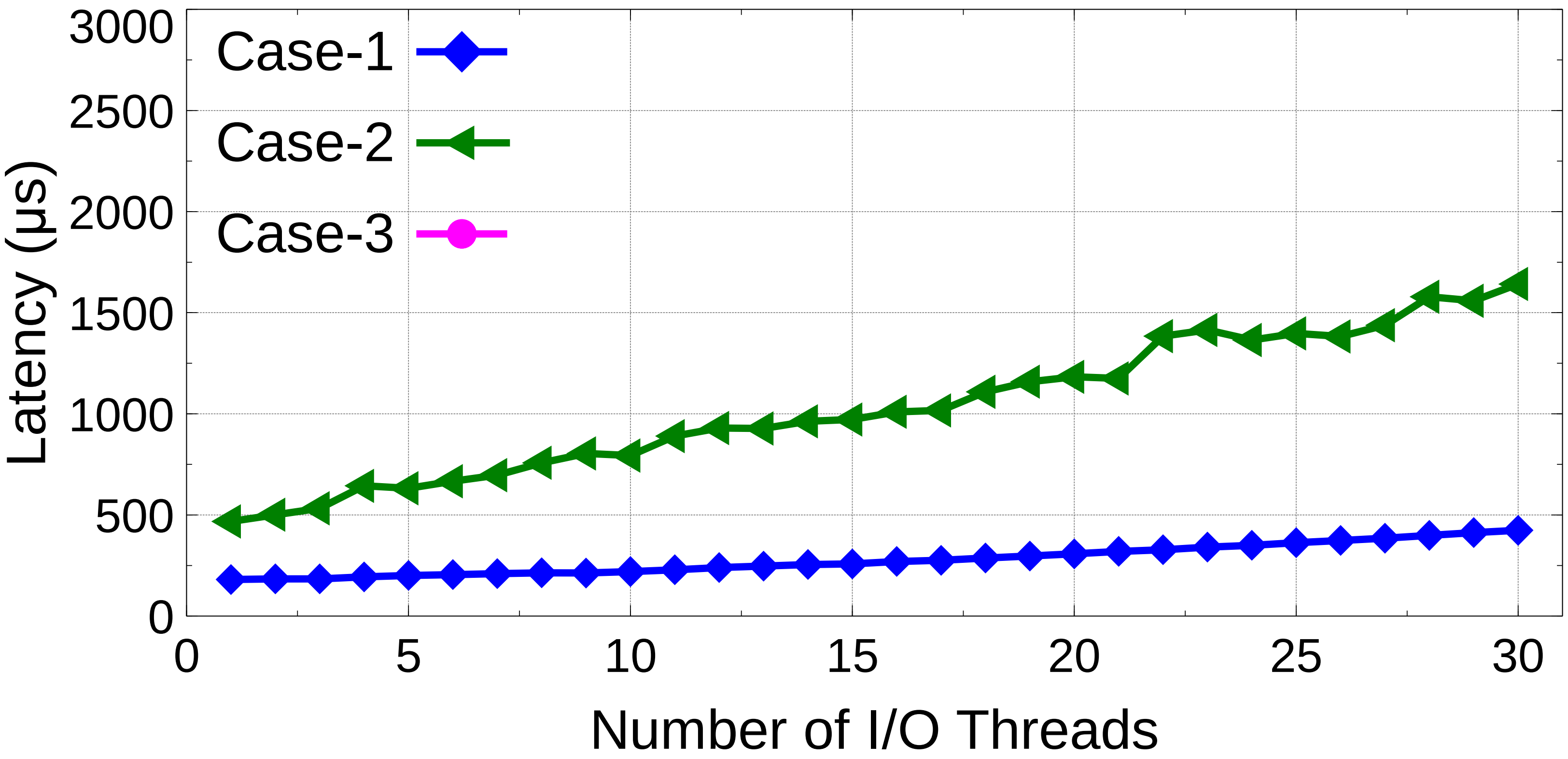
Case 1 : Only foreground I/O
Case 2 : Foreground I/O & Dedup without BTS
Case 3 : Foreground I/O & Dedup with BTS





Put() Performance

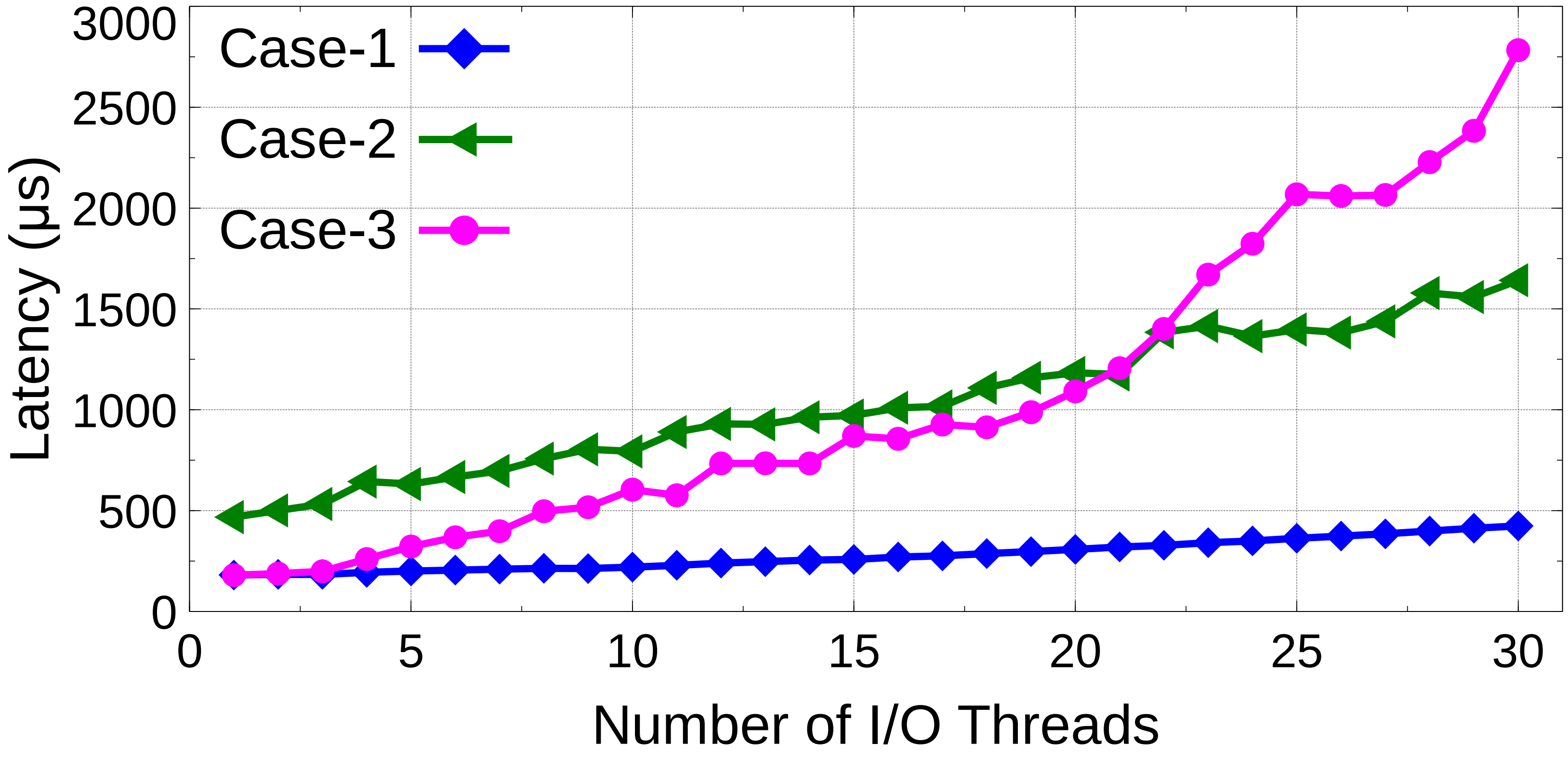
Case 1 : Only foreground I/O
Case 2 : Foreground I/O & Dedup without BTS
Case 3 : Foreground I/O & Dedup with BTS



Put() Performance

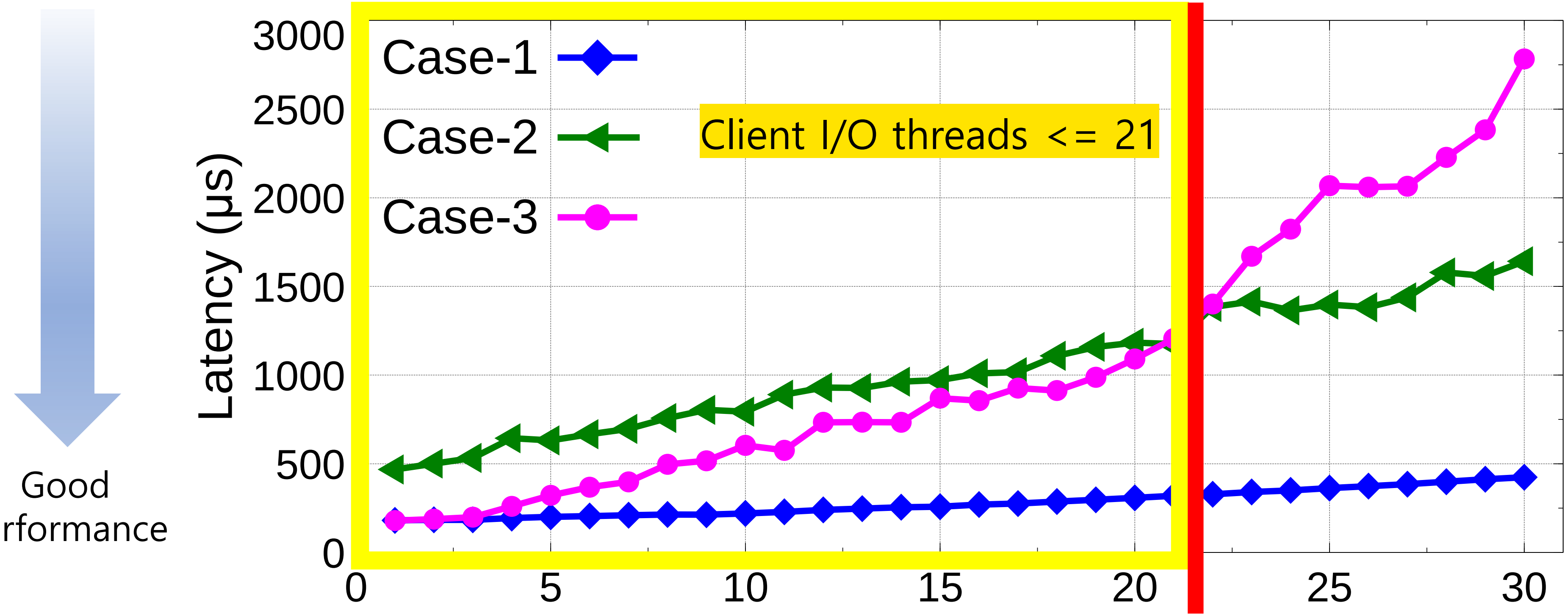
Case 1 : Only foreground I/O
Case 2 : Foreground I/O & Dedup without BTS
Case 3 : Foreground I/O & Dedup with BTS

Good
Performance



Put() Performance

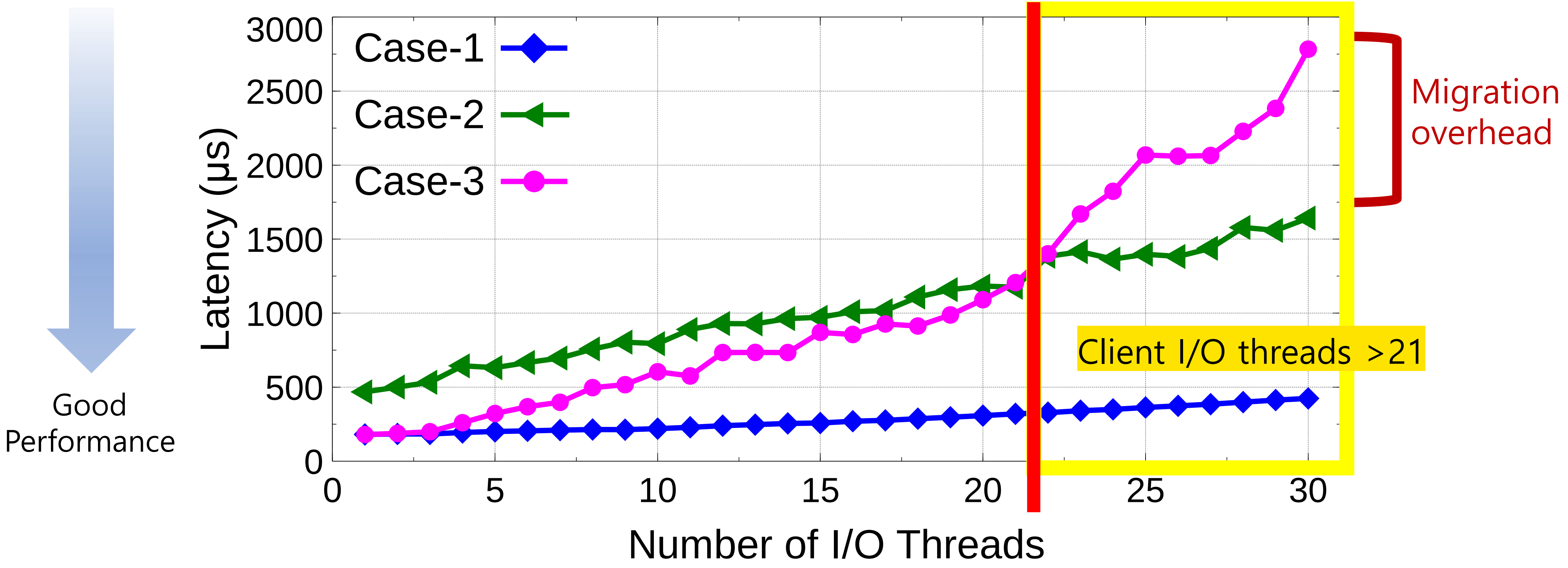
Case 1 : Only foreground I/O
Case 2 : Foreground I/O & Dedup without BTS
Case 3 : Foreground I/O & Dedup with BTS



BTS reduced latency by an average of 47.8% when the number of host I/O threads was less than 10.

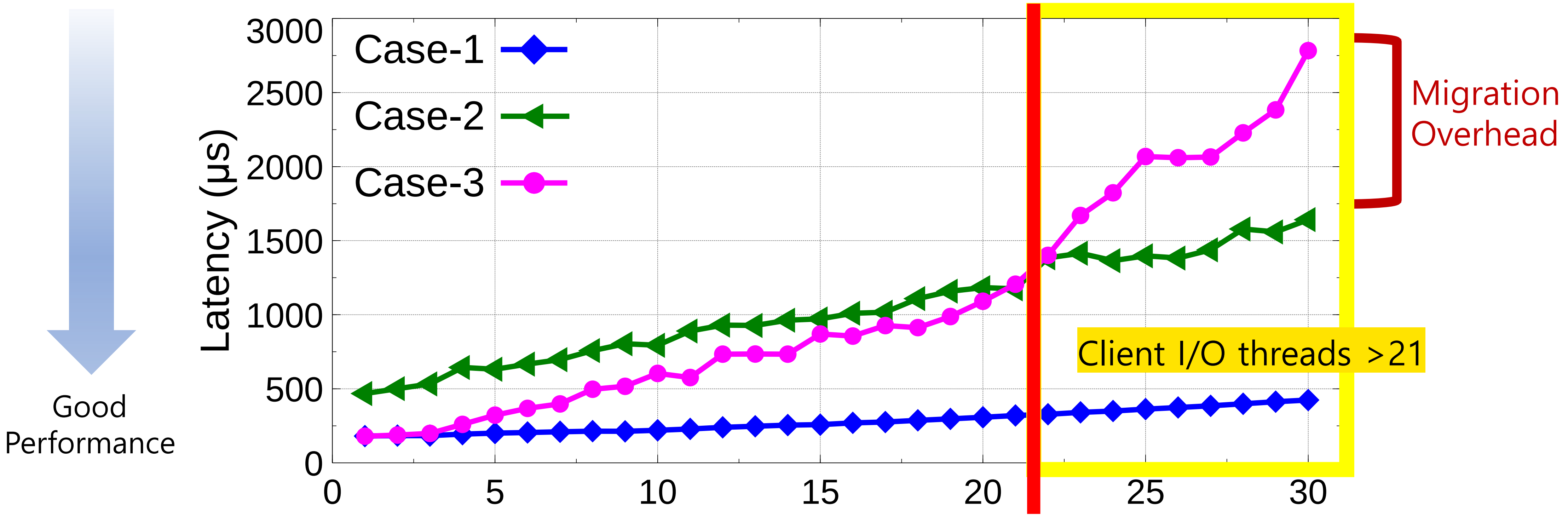
Put() Performance

Case 1 : Only foreground I/O
Case 2 : Foreground I/O & Dedup without BTS
Case 3 : Foreground I/O & Dedup with BTS



Put() Performance

Case 1 : Only foreground I/O
Case 2 : Foreground I/O & Dedup without BTS
Case 3 : Foreground I/O & Dedup with BTS



Under heavy load, the migration overhead for background tasks outweighs the performance gains.

Conclusion

- We have identified a problem with SPDK where background tasks increase the response time of foreground I/O in CSD using SPDK
- We proposed a **Background Task-Aware Scheduler (BTS)** in SPDK for CSD
- Comprehensive evaluation showed that the BTS scheduler is effective when core utilization is rather low

Conclusion

- We have identified a problem with SPDK where background tasks increase the response time of foreground I/O in CSD using SPDK
- We proposed a **Background Task-Aware Scheduler (BTS)** in SPDK for CSD
- Comprehensive evaluation showed that the BTS scheduler is effective when core utilization is rather low
- **BTS can be applied to any storage system using SPDK**

Thank you!

Yeohyeon Park
yeohyeon@sogang.ac.kr