OctoKV: An Agile Network-Based Key-Value Storage System with Robust Load Orchestration

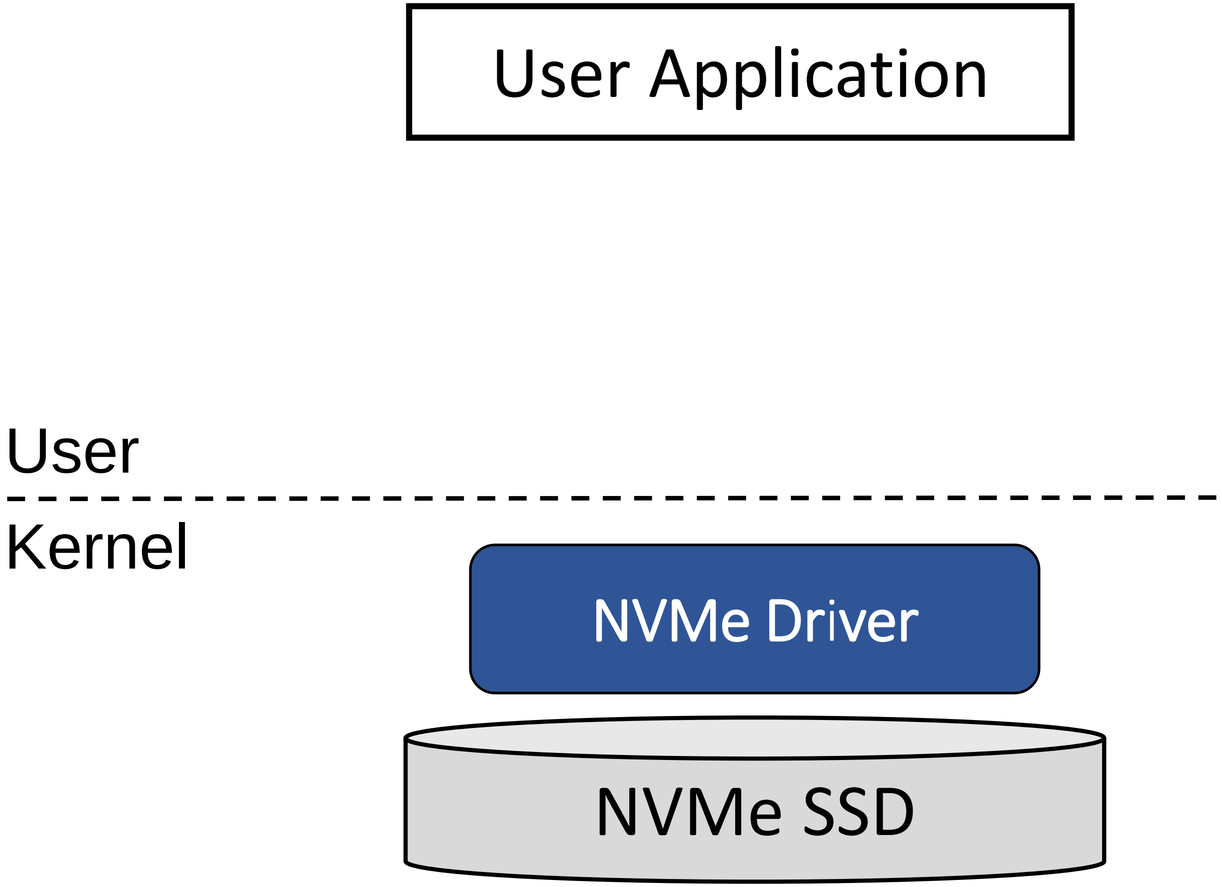
Yeohyeon Park[1], Junhyeok Park[1], Awais Khan[2], Junghwan Park[1], Chang-Gyu Lee[1], Woosuk Chung[3], Youngjae Kim[1]

MASCOTS'23

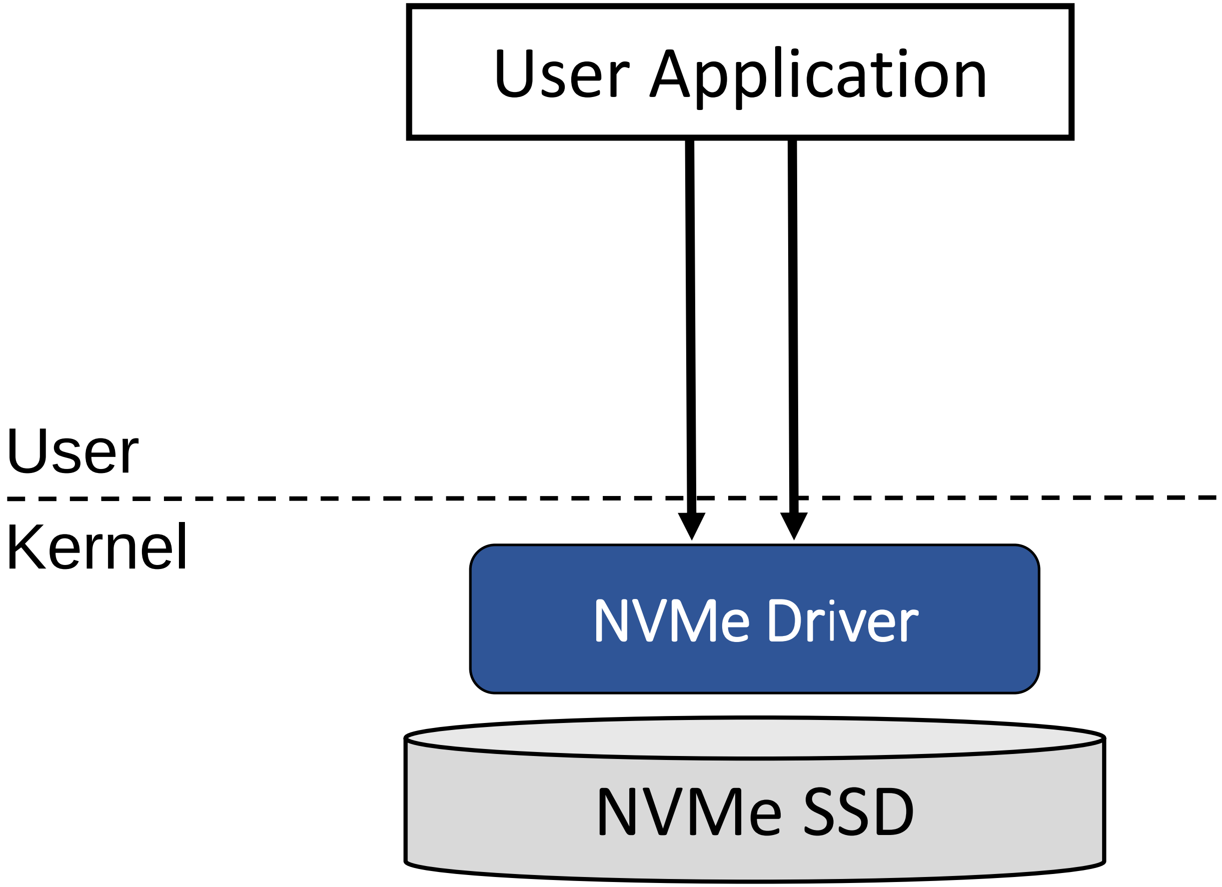
Presenter: Junhyeok Park

Content

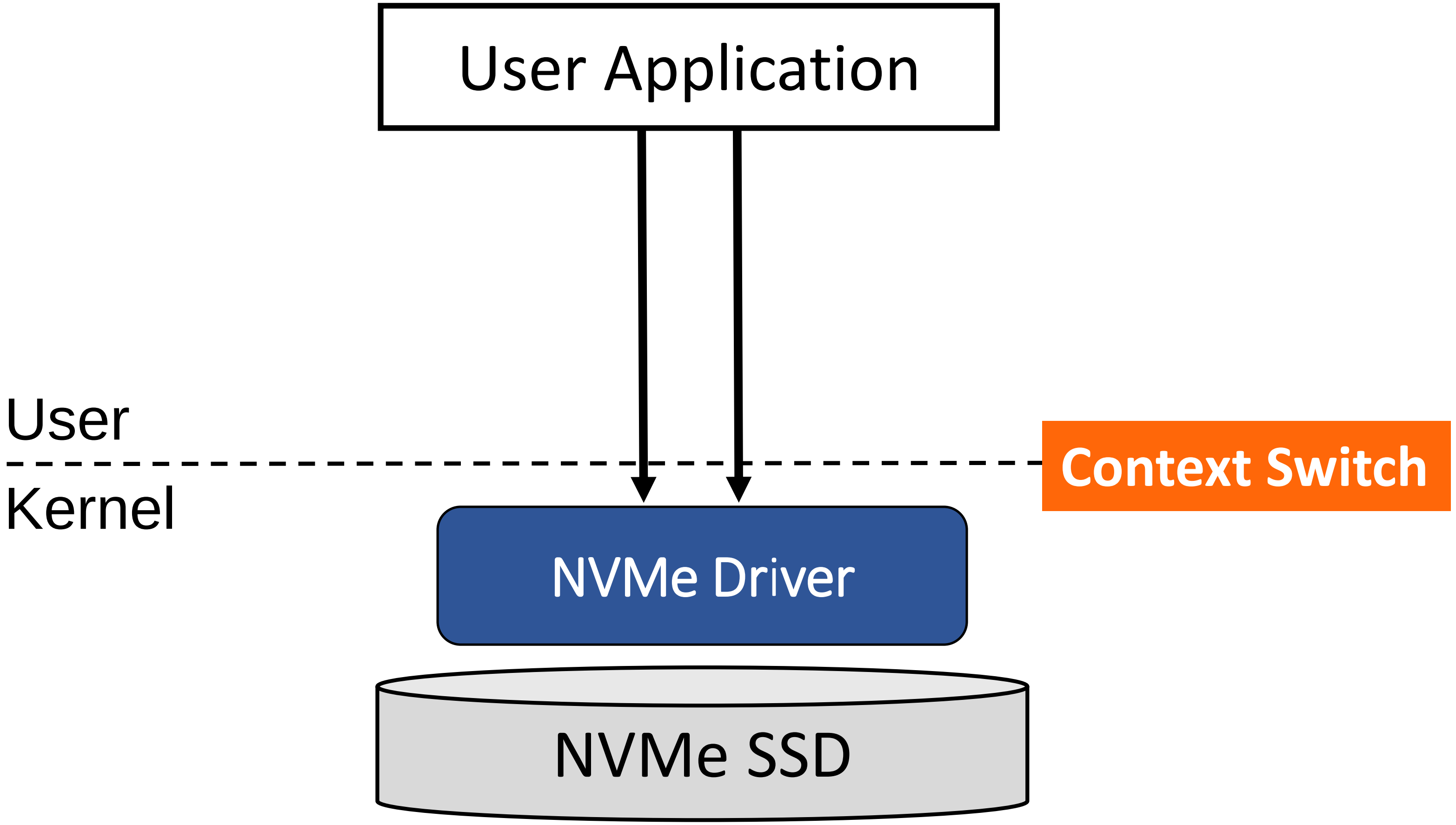
(1) User Space NVMe Driver



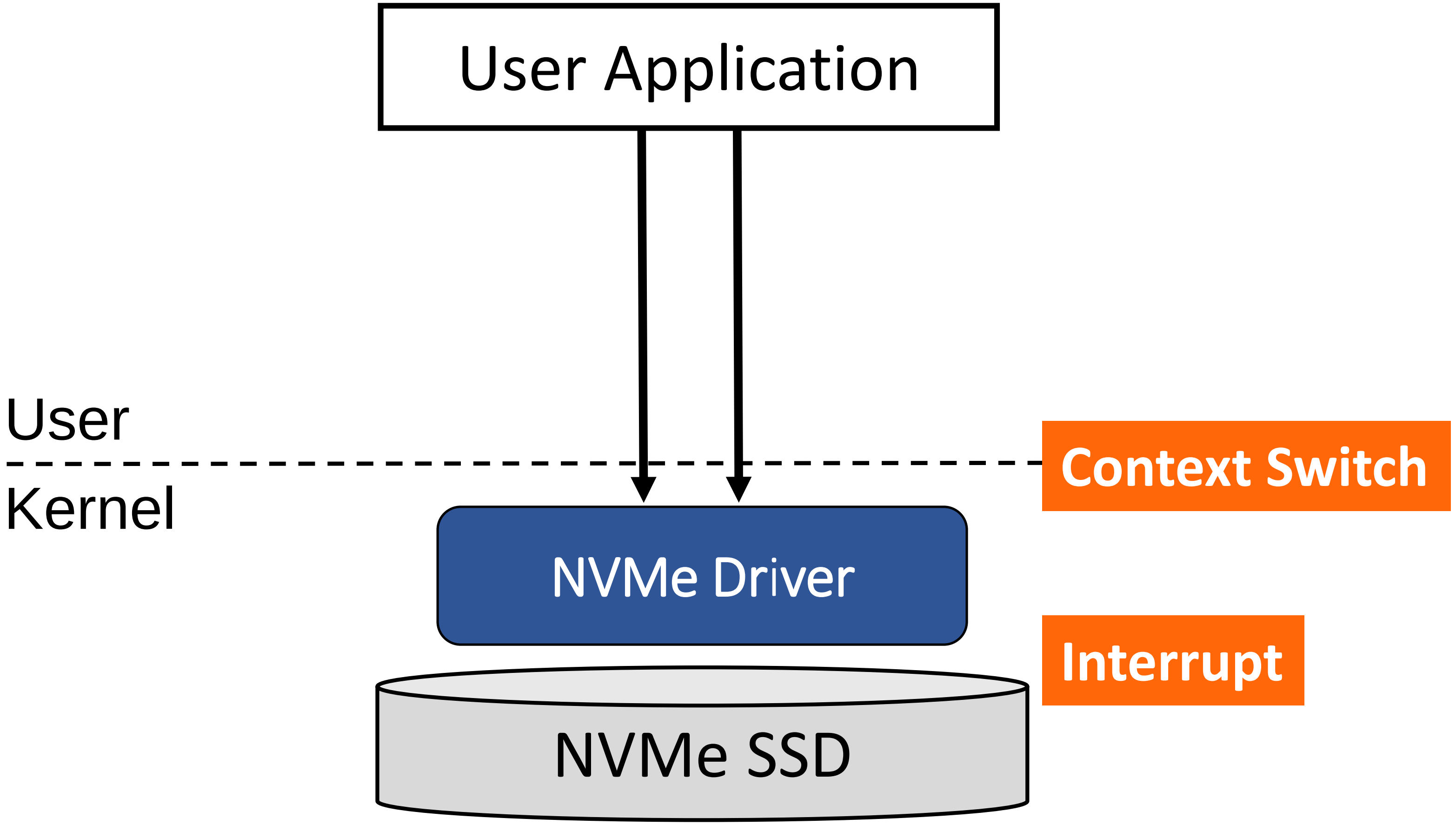
(1) User Space NVMe Driver



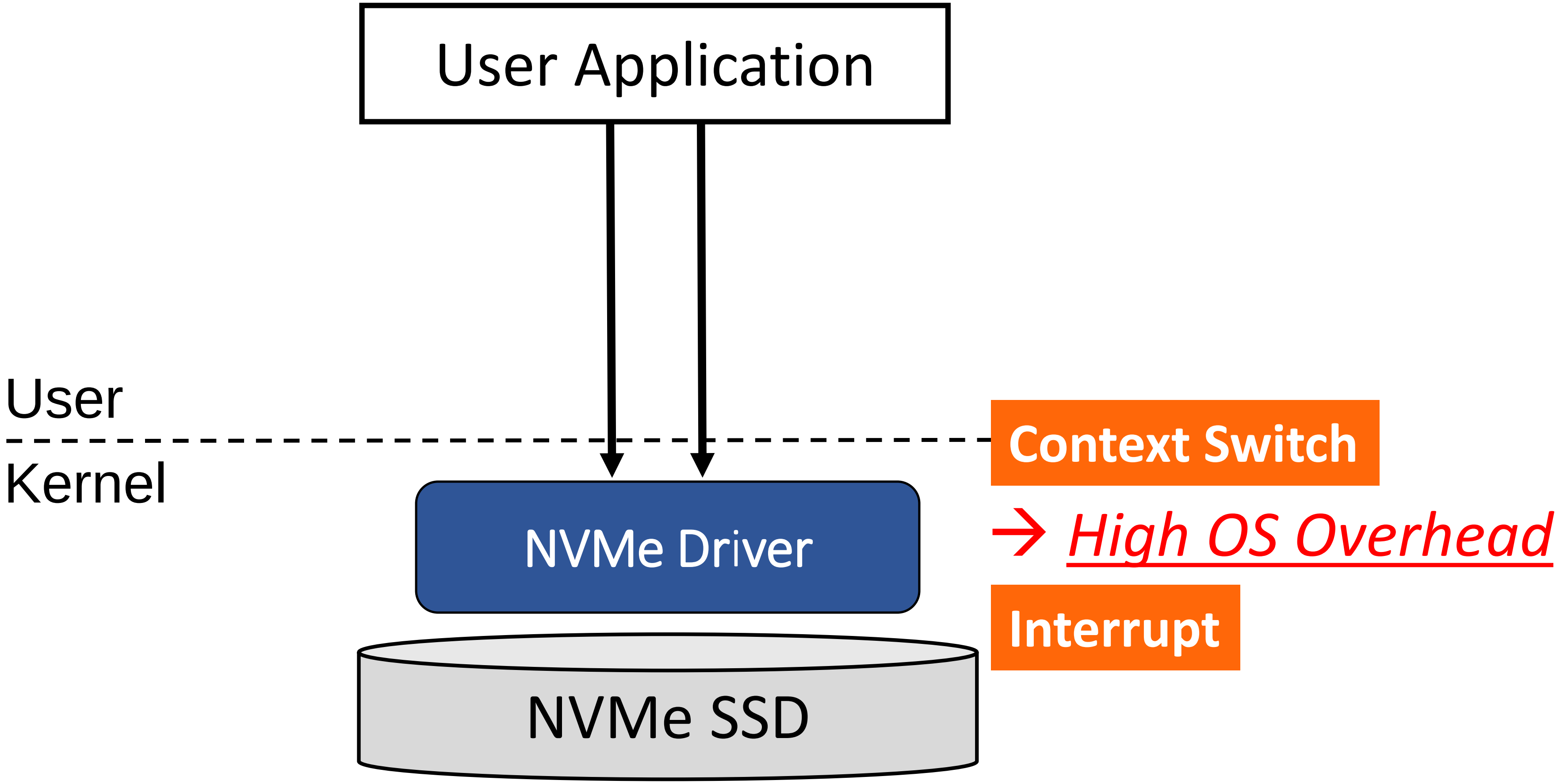
(1) User Space NVMe Driver



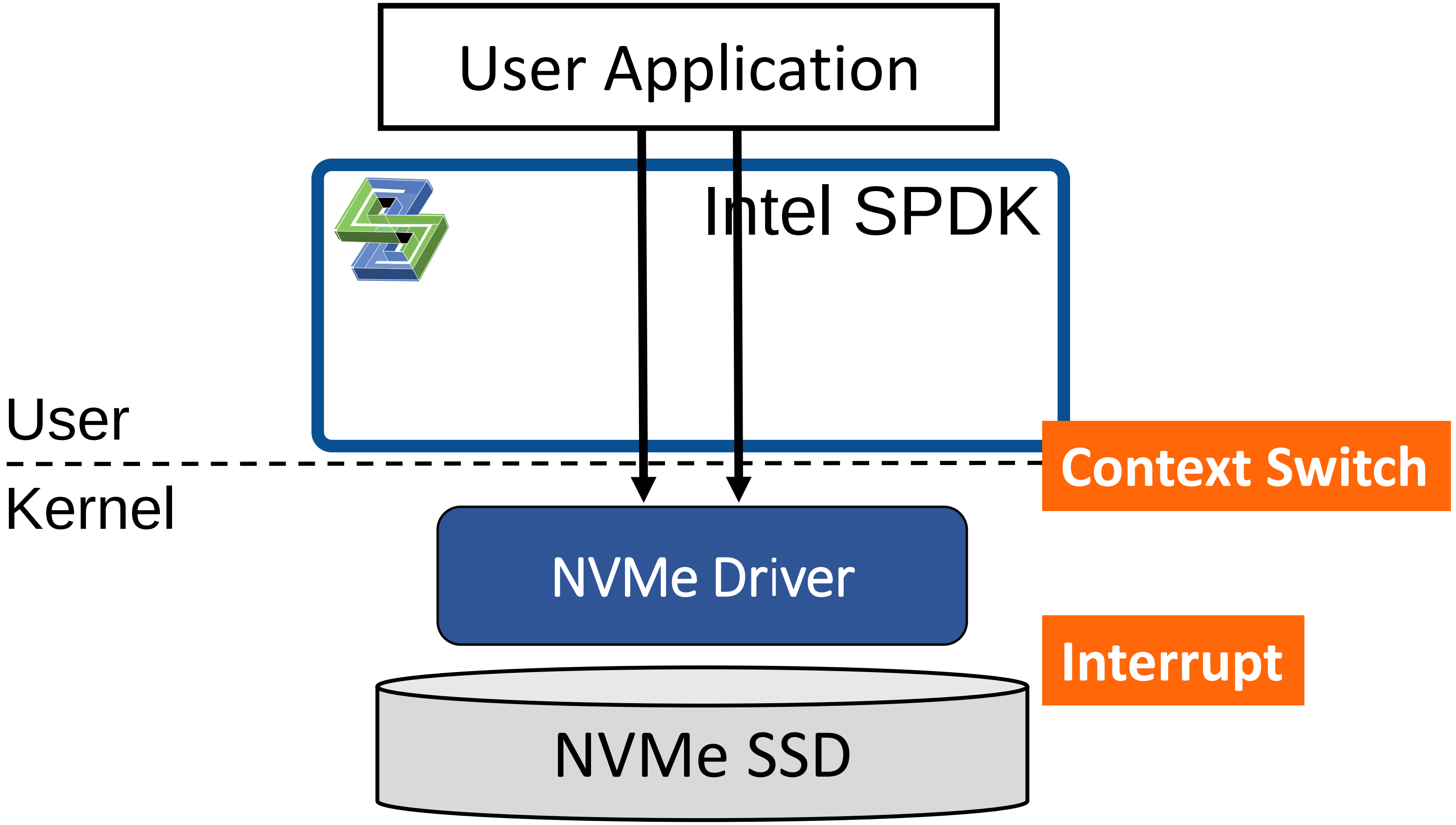
(1) User Space NVMe Driver



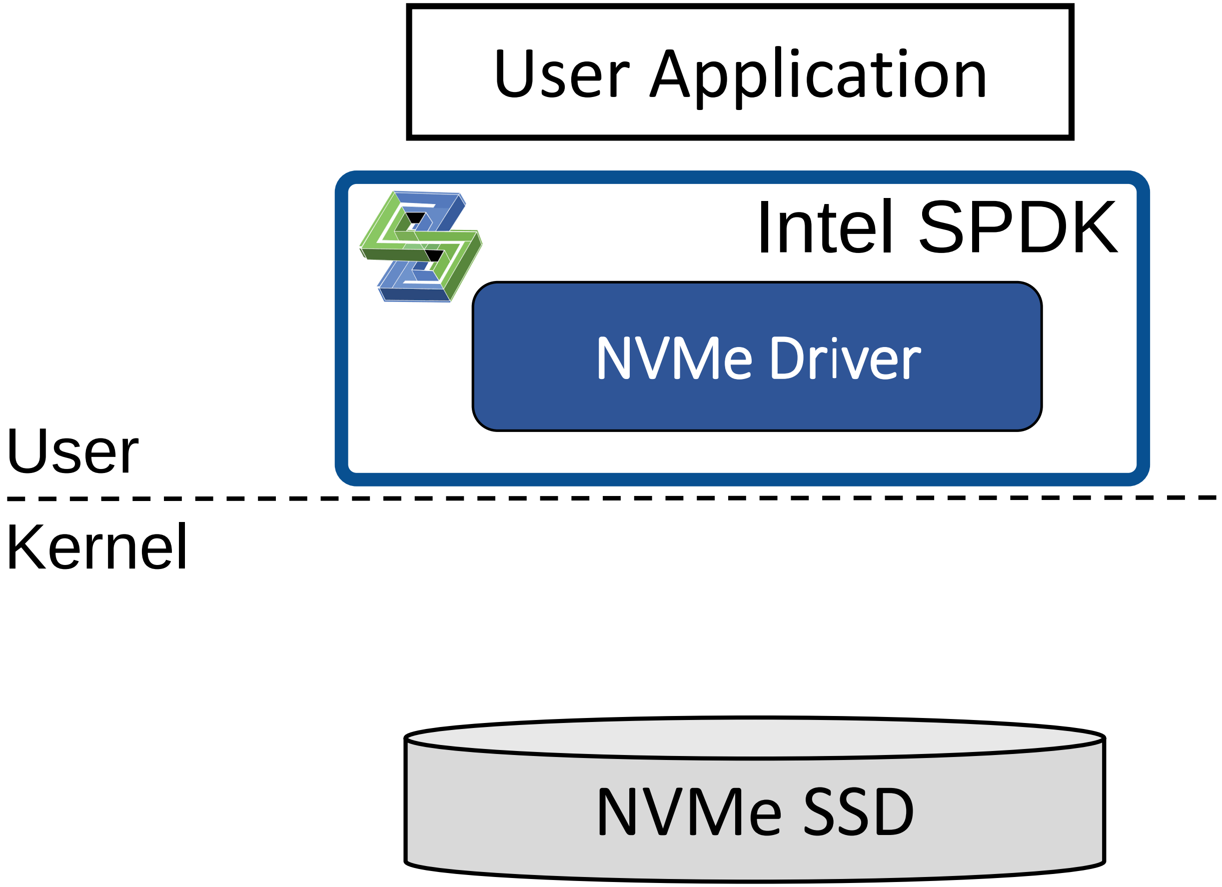
(1) User Space NVMe Driver



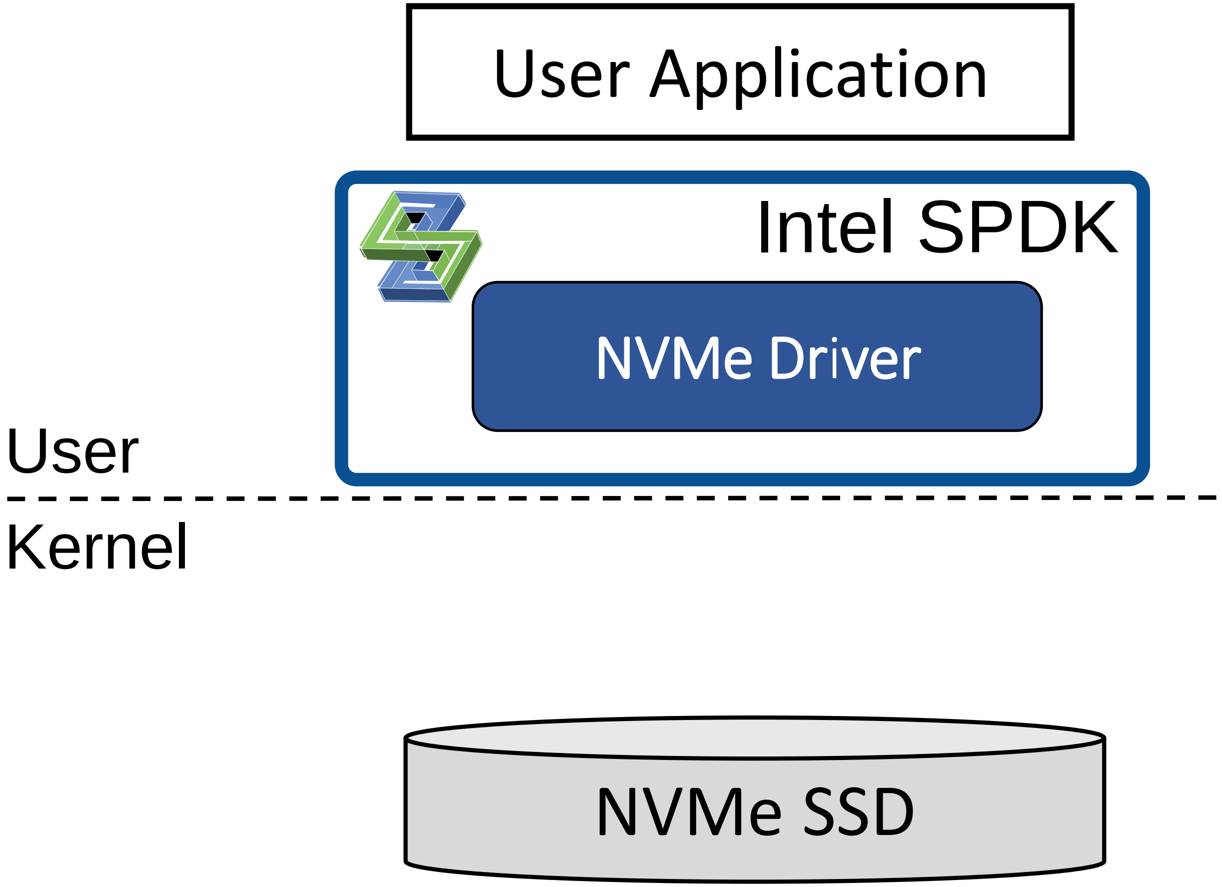
(1) User Space NVMe Driver



(1) User Space NVMe Driver

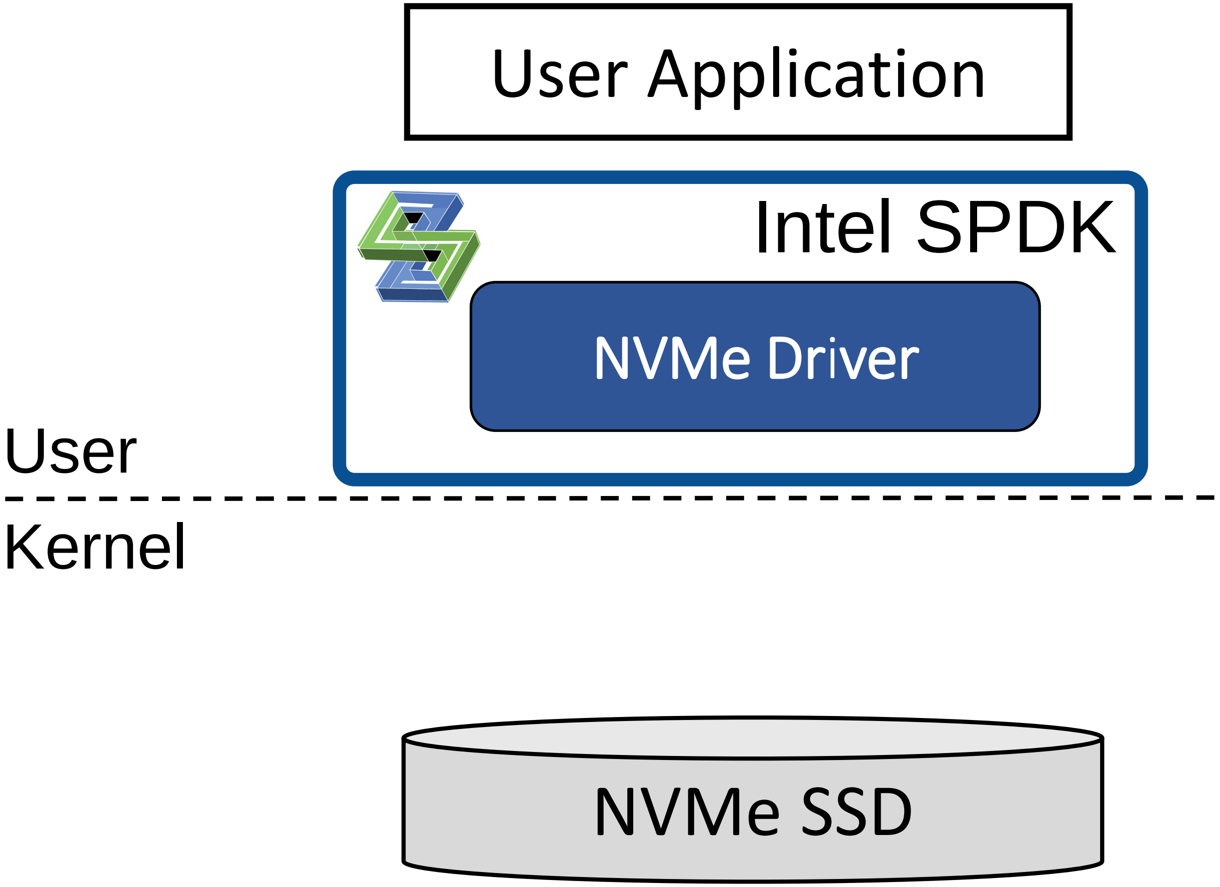


(1) User Space NVMe Driver



1. User level NVMe driver

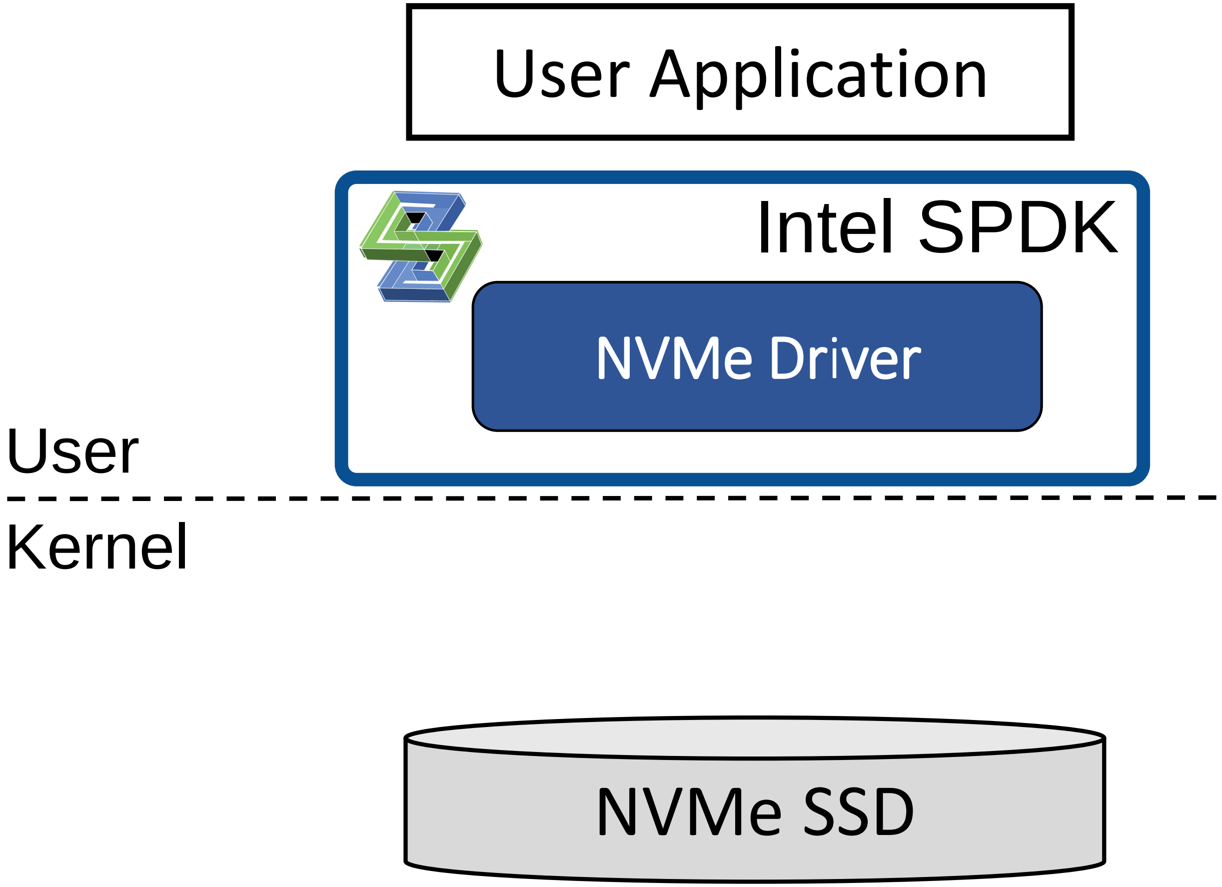
(1) User Space NVMe Driver



1. User level NVMe driver

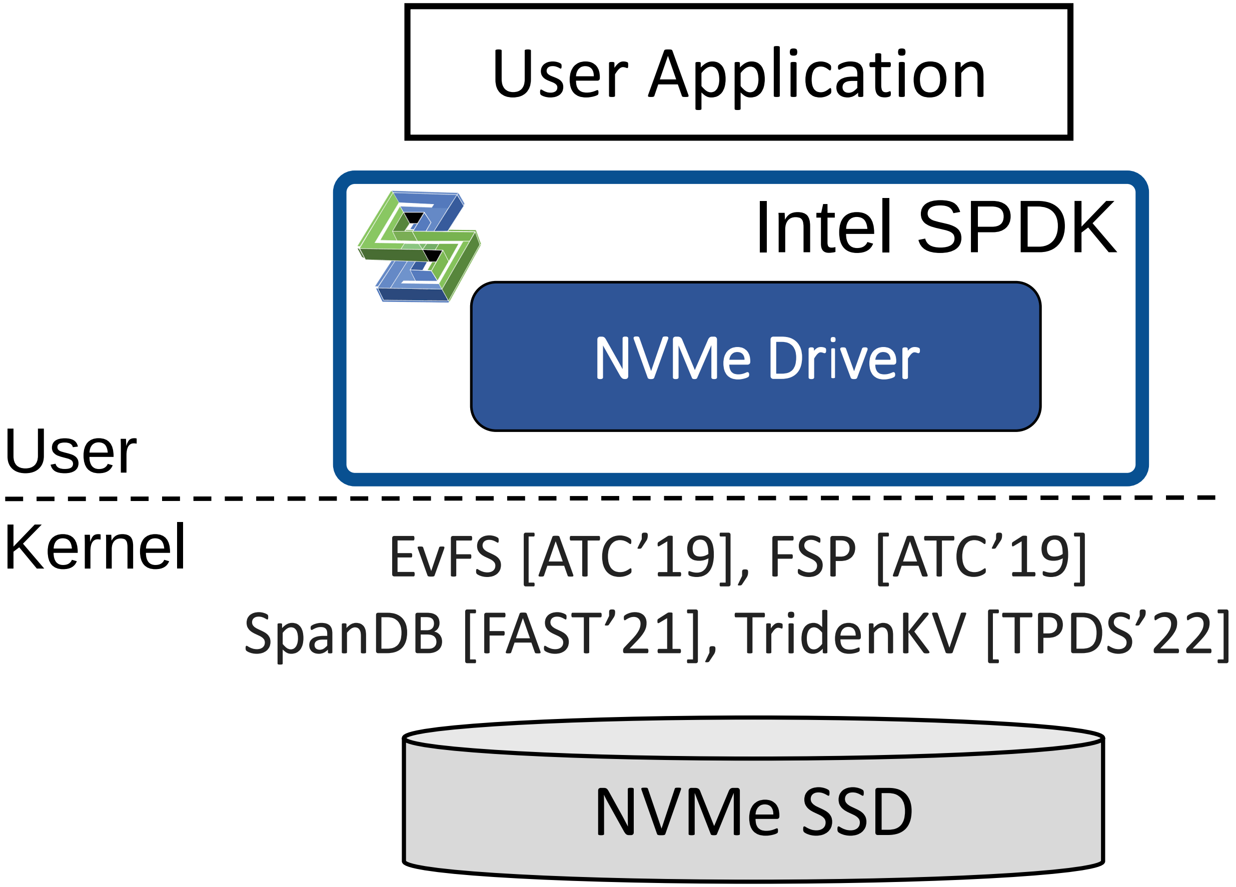
2. Bind I/O to a specific core

(1) User Space NVMe Driver



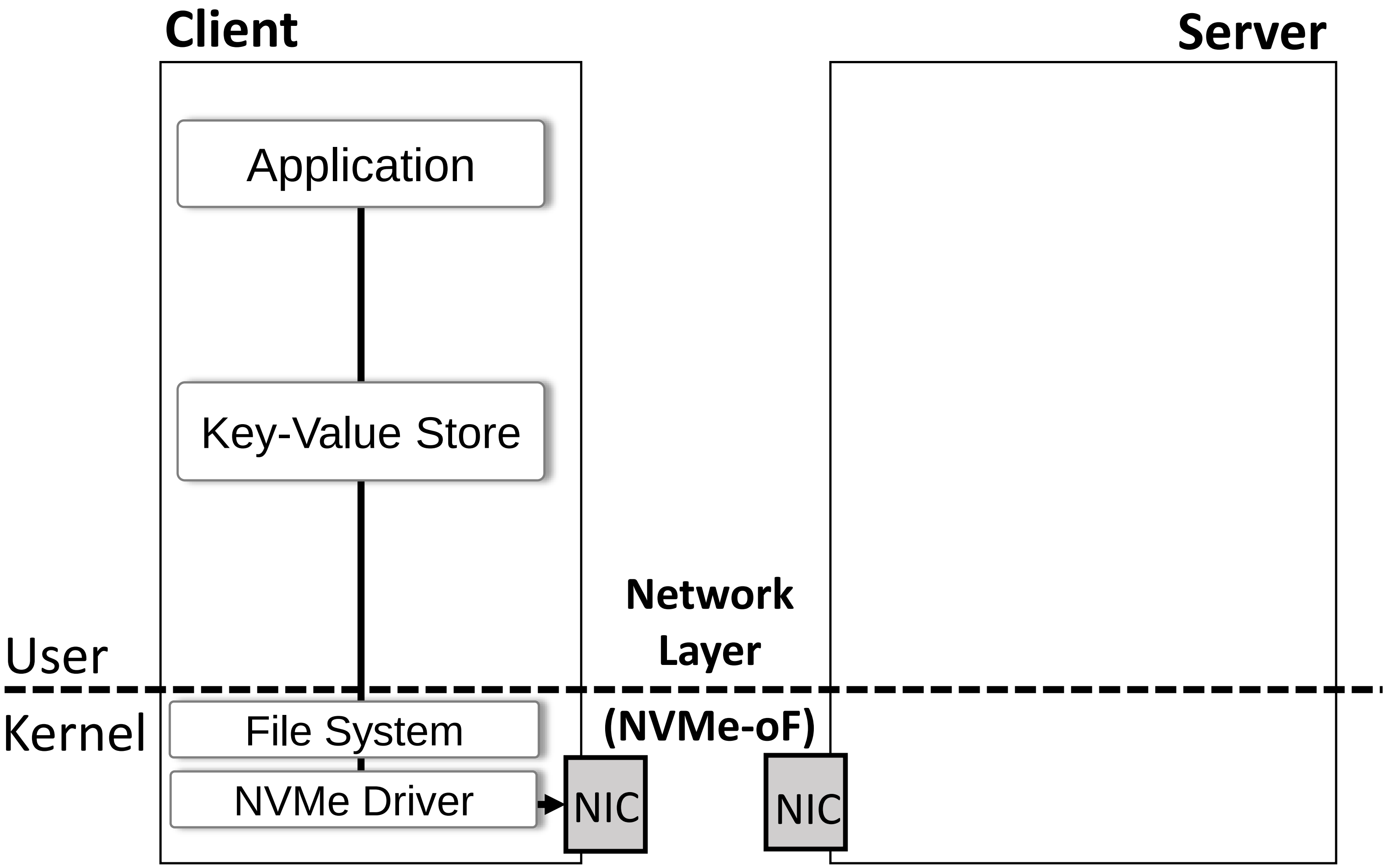
1. User level NVMe driver
2. Bind I/O to a specific core
3. Use polling for completion

(1) User Space NVMe Driver

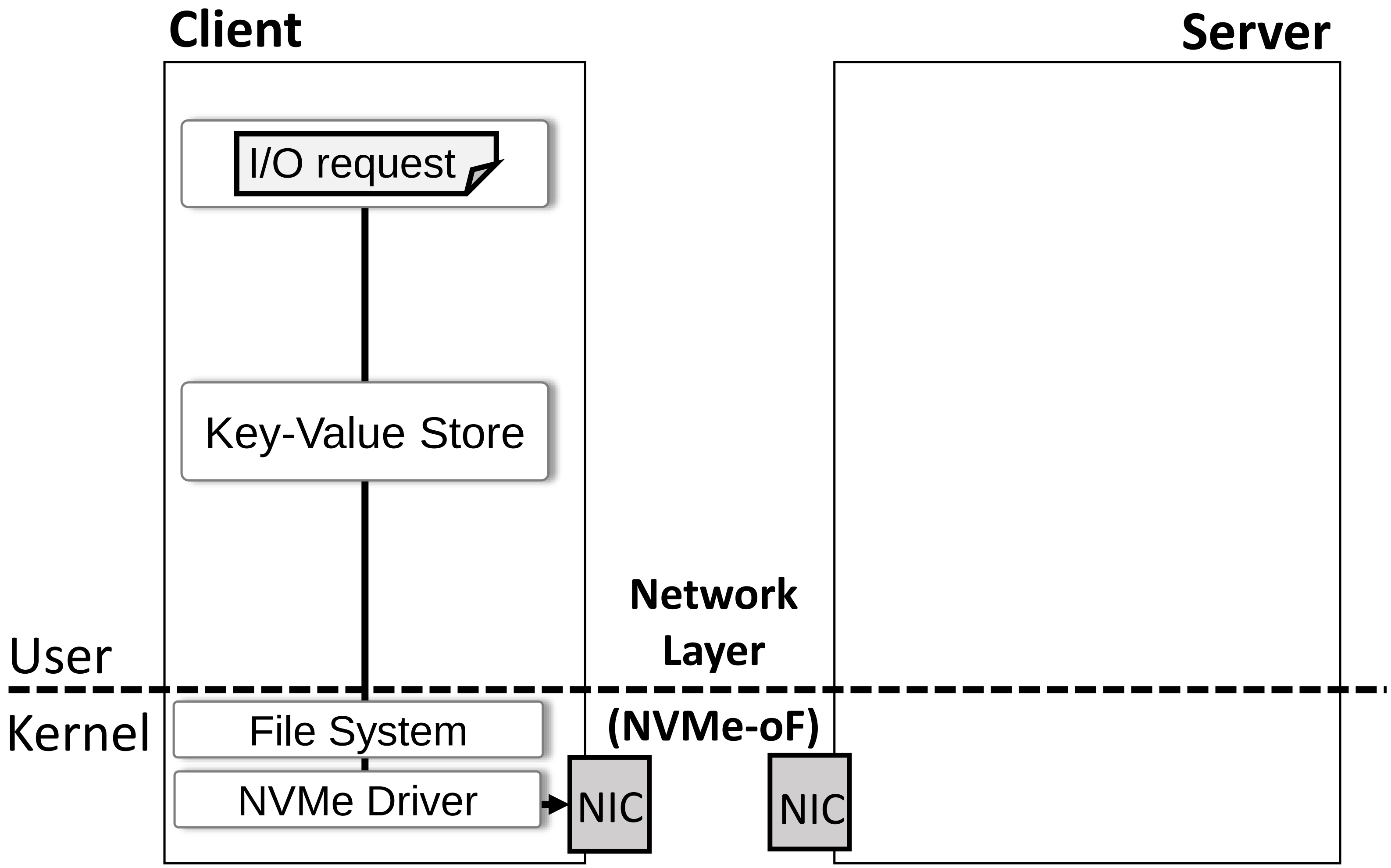


1. User level NVMe driver
2. Bind I/O to a specific core
3. Use polling for completion

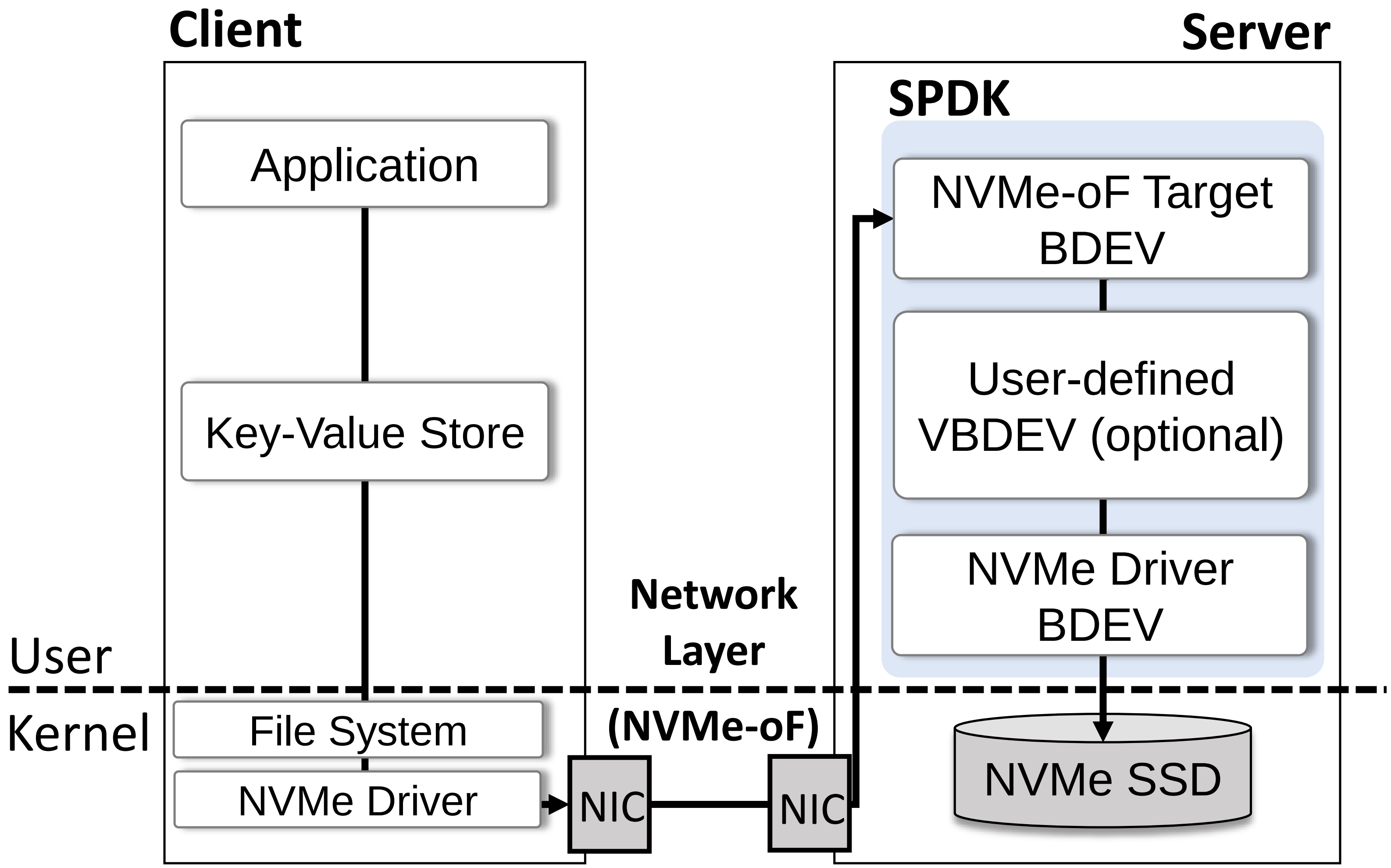
(2) SPDK-based Network-based Block Storage



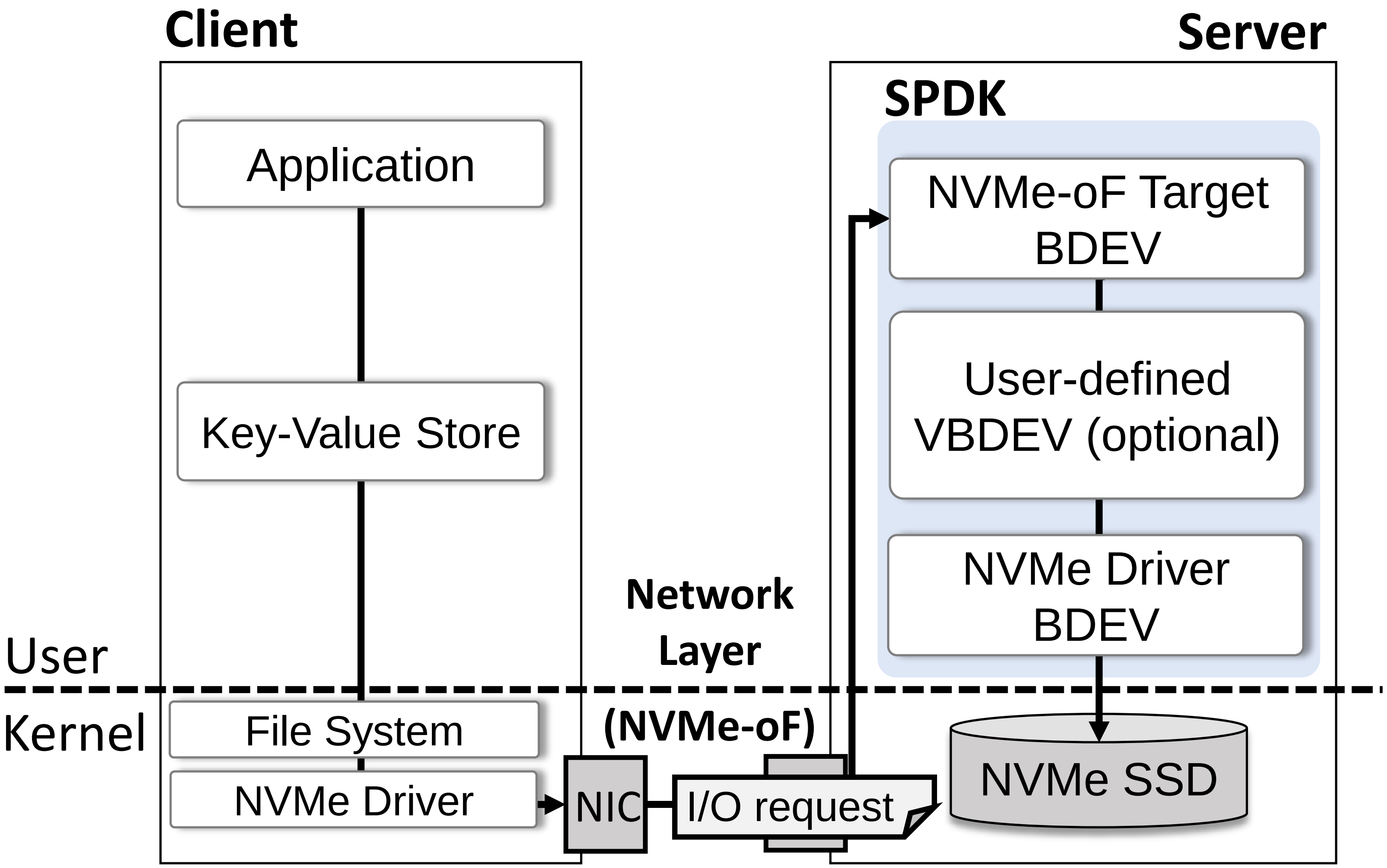
(2) SPDK-based Network-based Block Storage



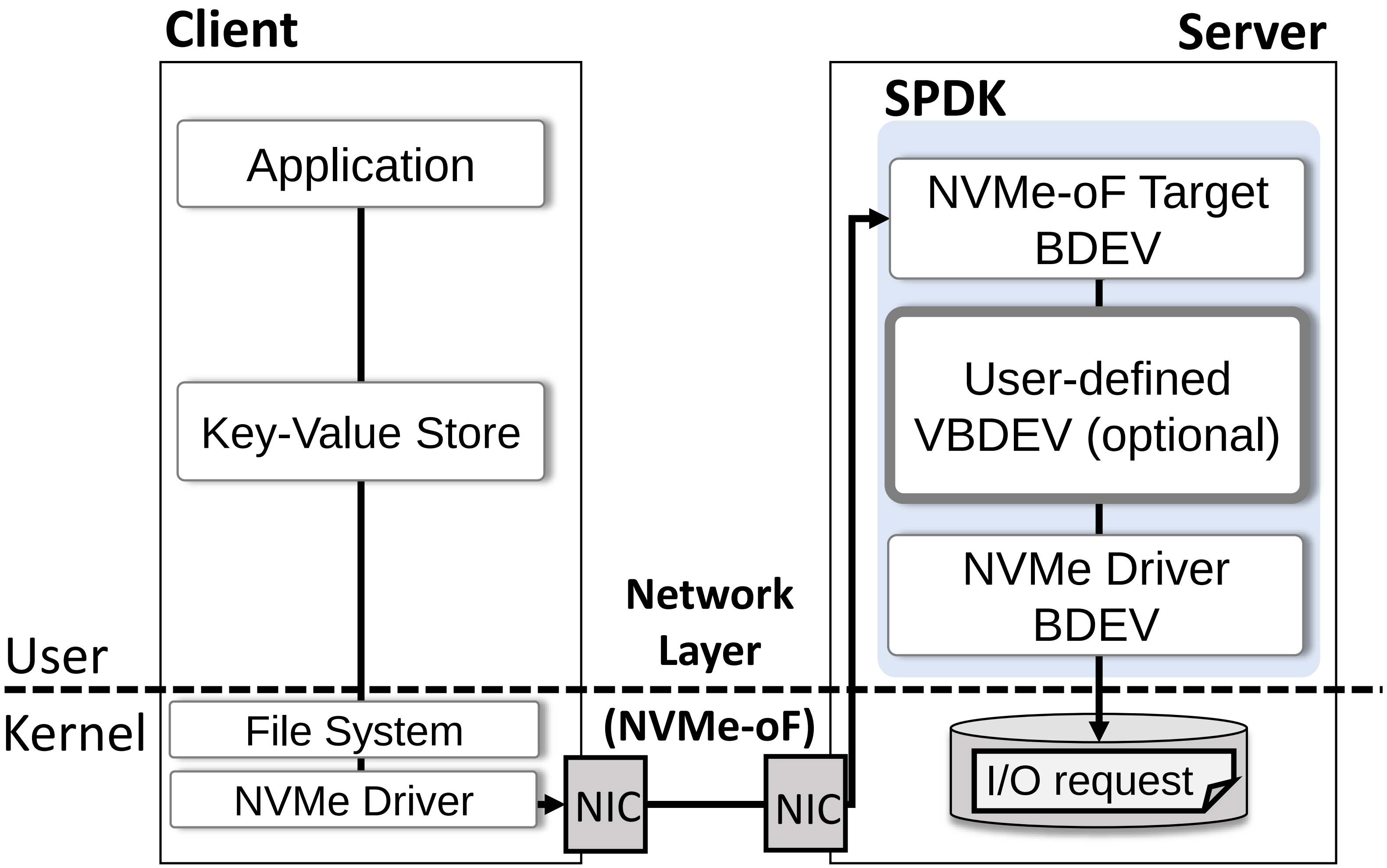
(2) SPDK-based Network-based Block Storage



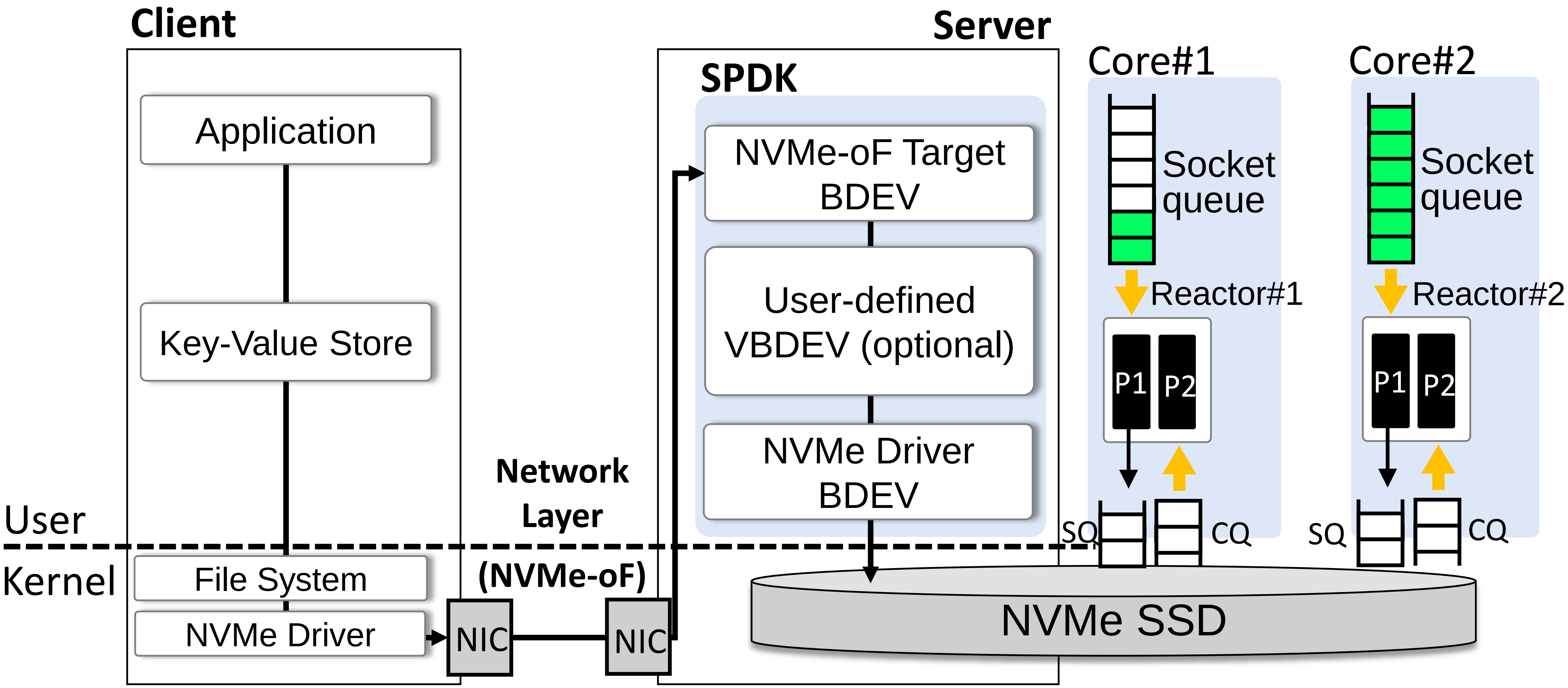
(2) SPDK-based Network-based Block Storage



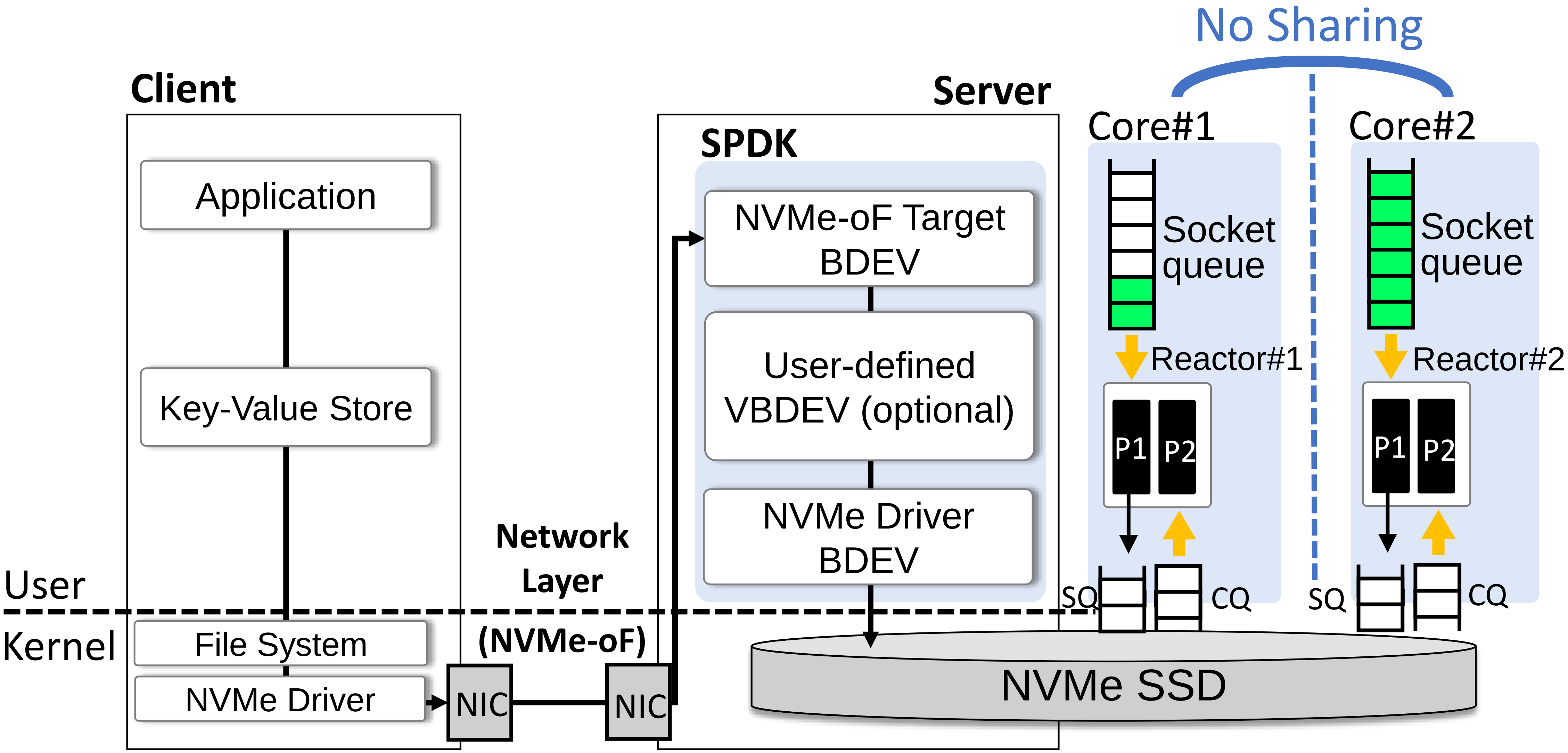
(2) SPDK-based Network-based Block Storage



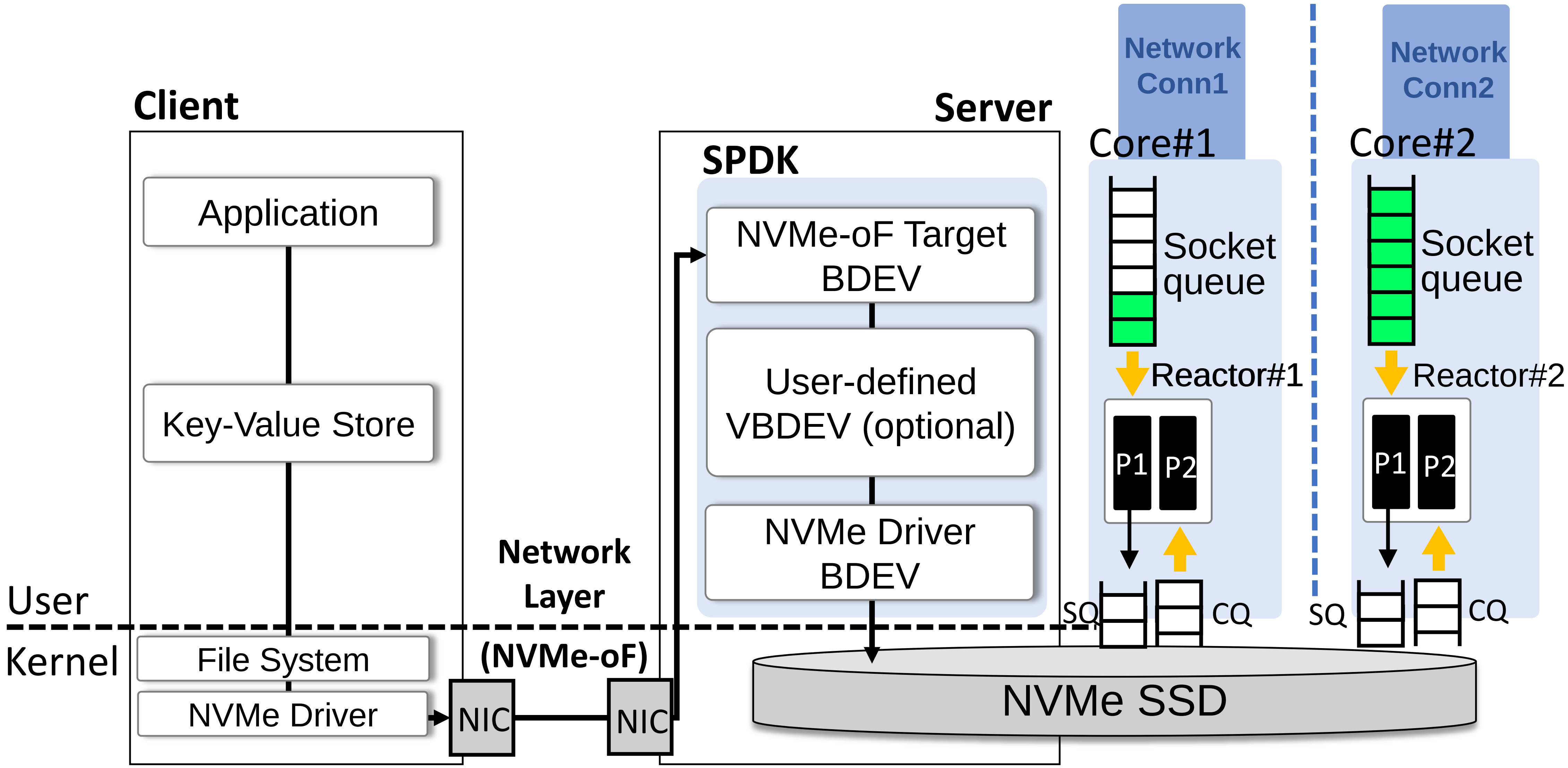
(2) SPDK-based Network-based Block Storage



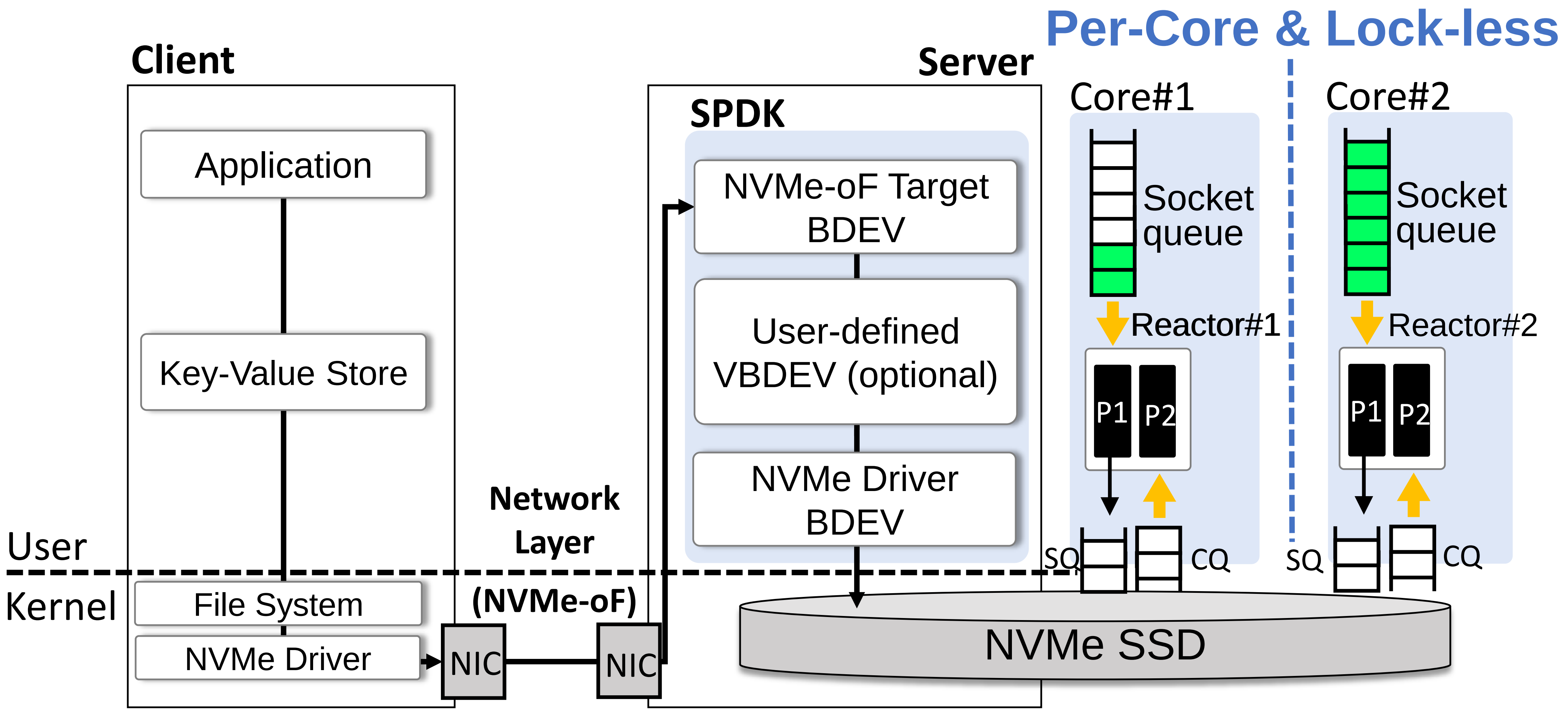
(2) SPDK-based Network-based Block Storage



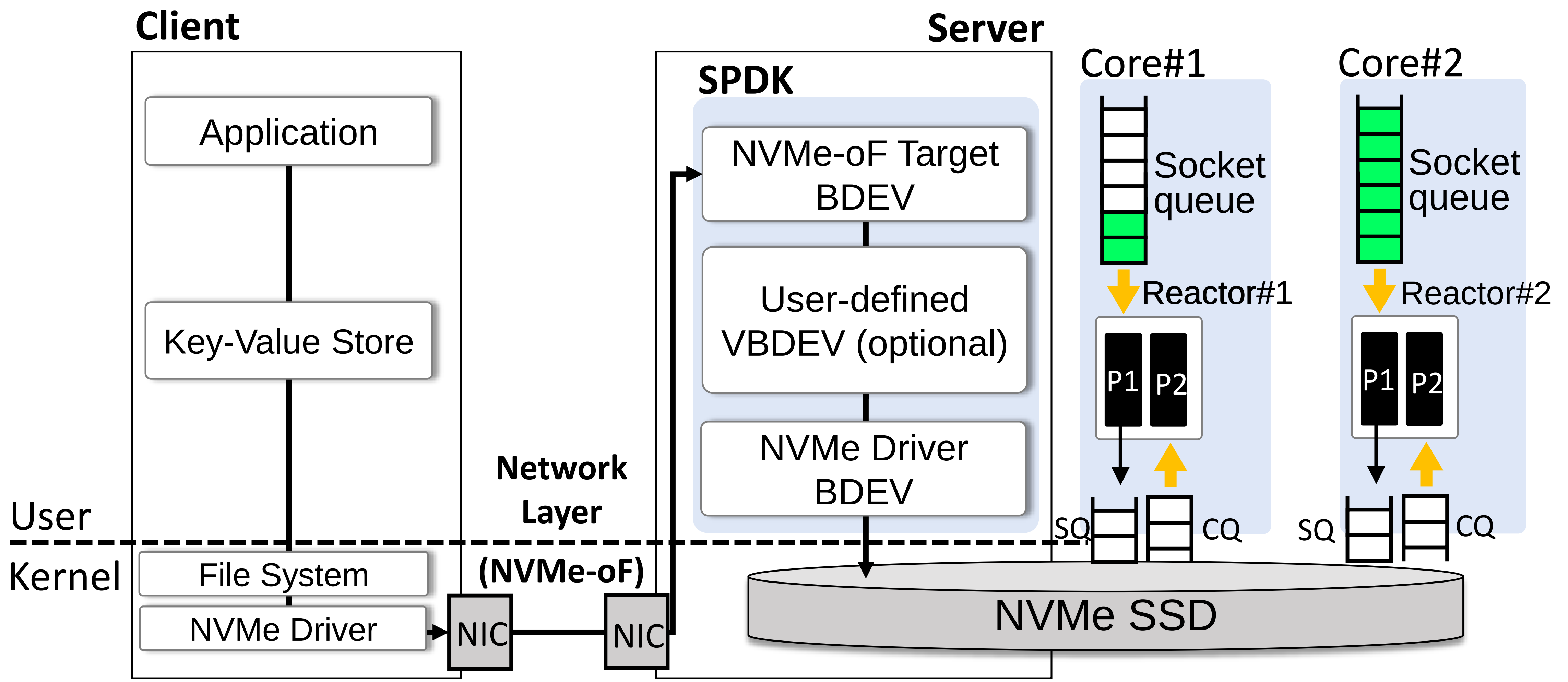
(2) SPDK-based Network-based Block Storage



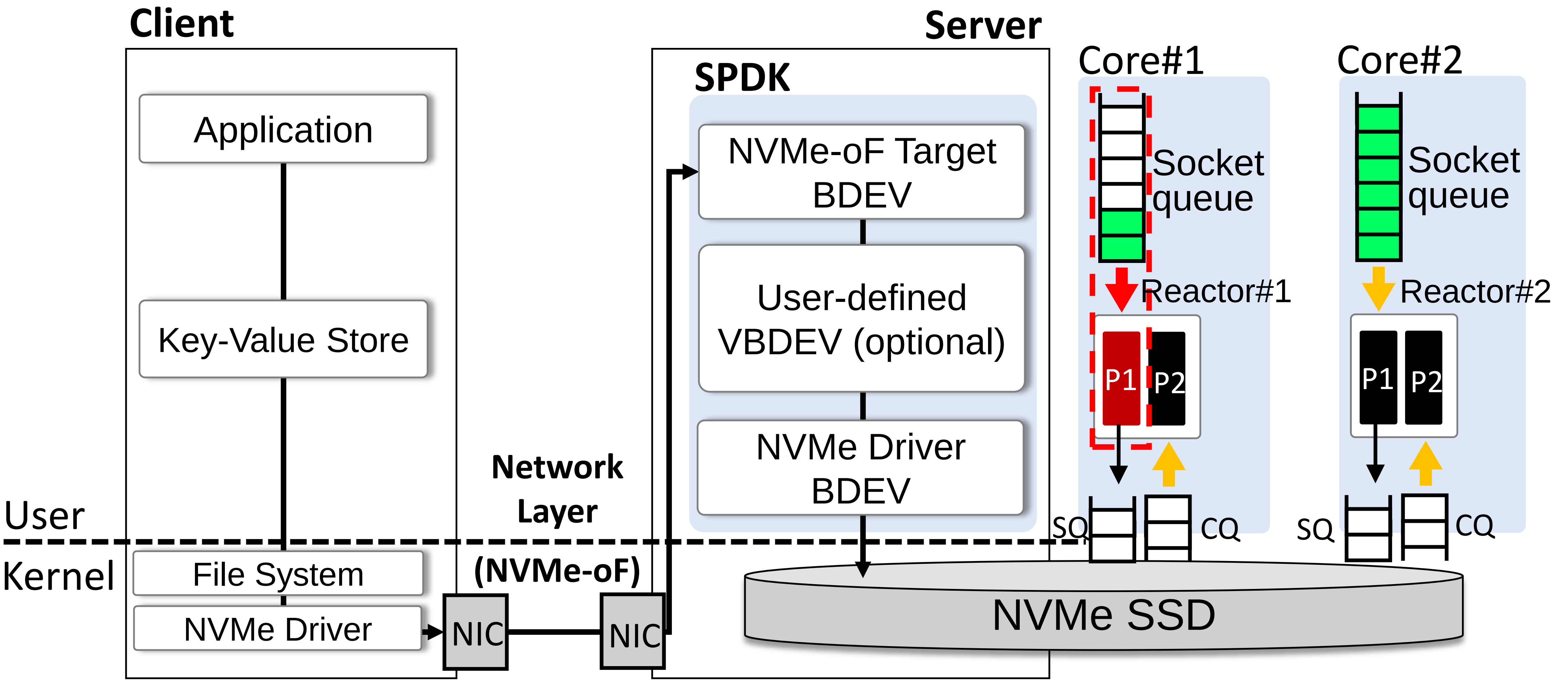
(2) SPDK-based Network-based Block Storage



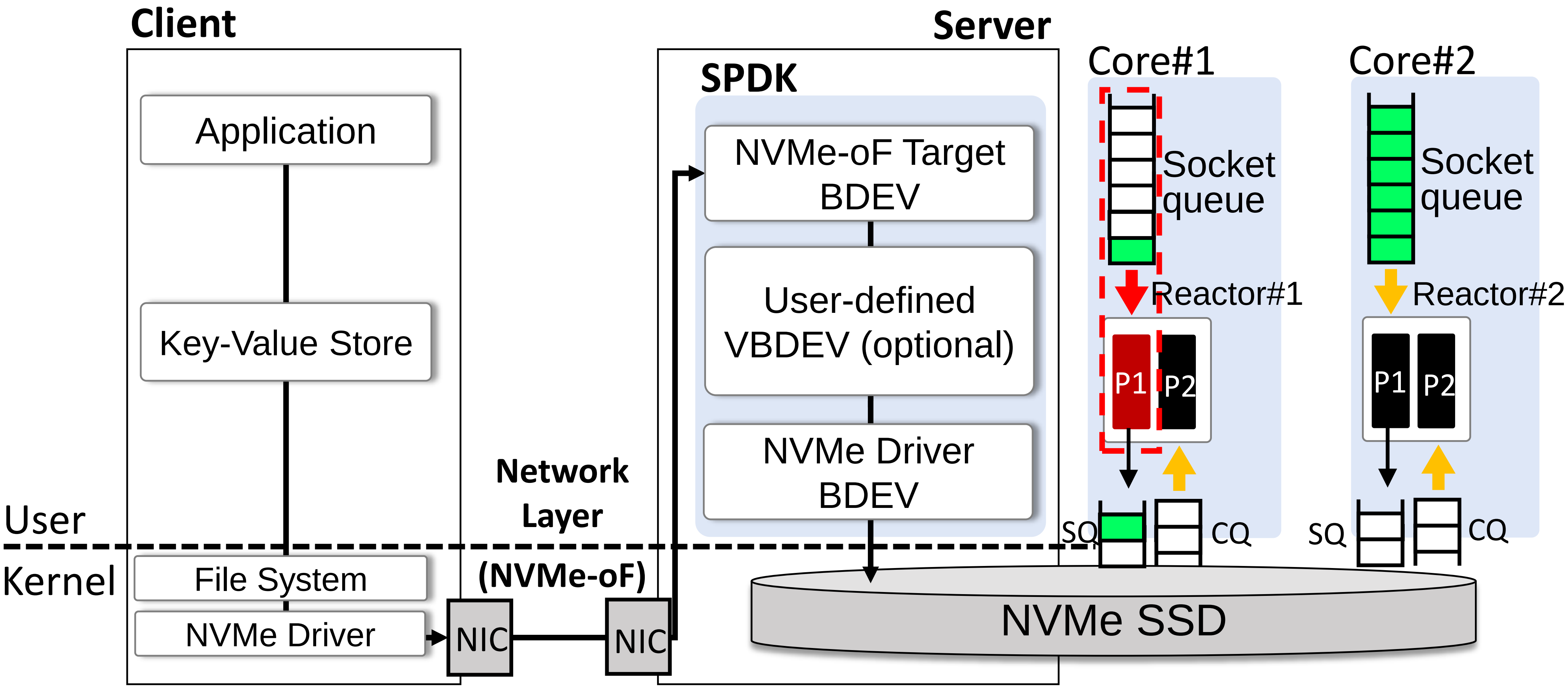
(2) SPDK-based Network-based Block Storage



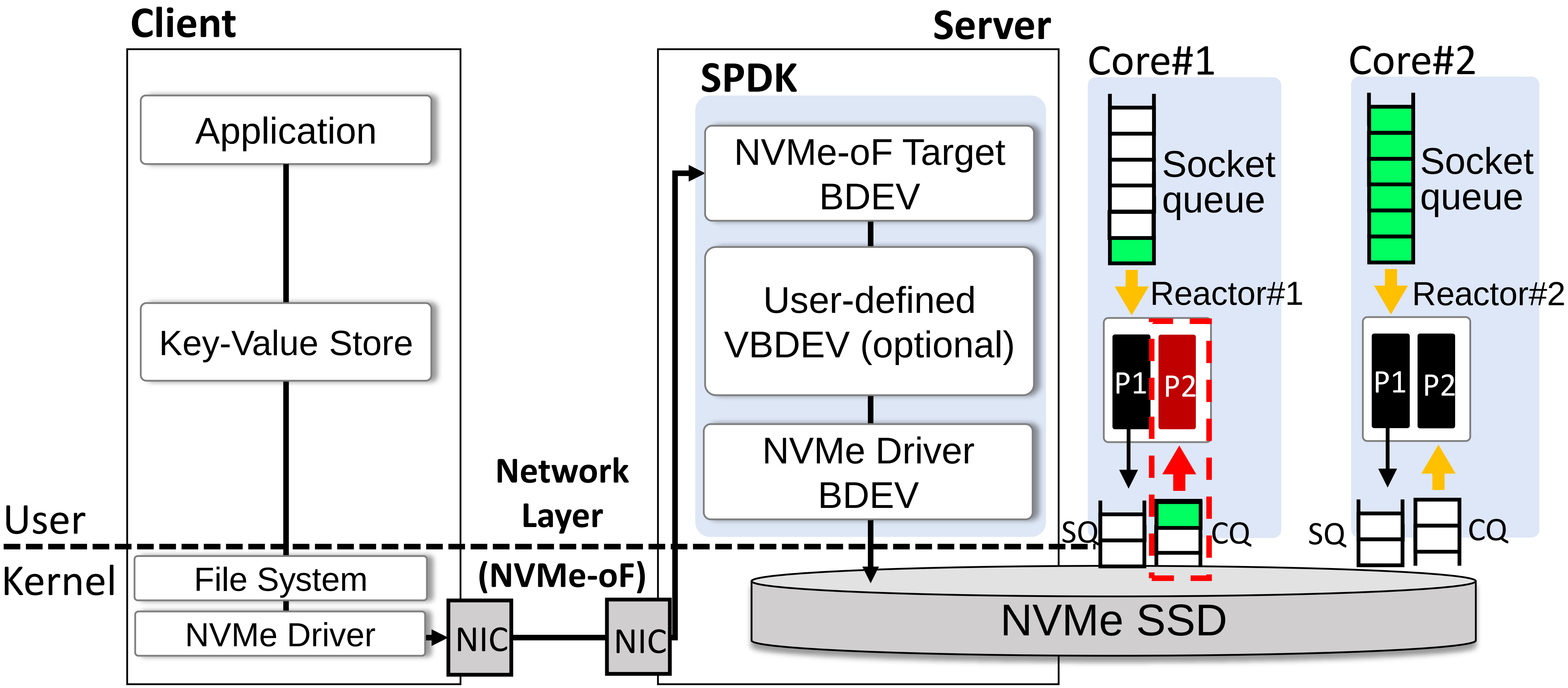
(2) SPDK-based Network-based Block Storage



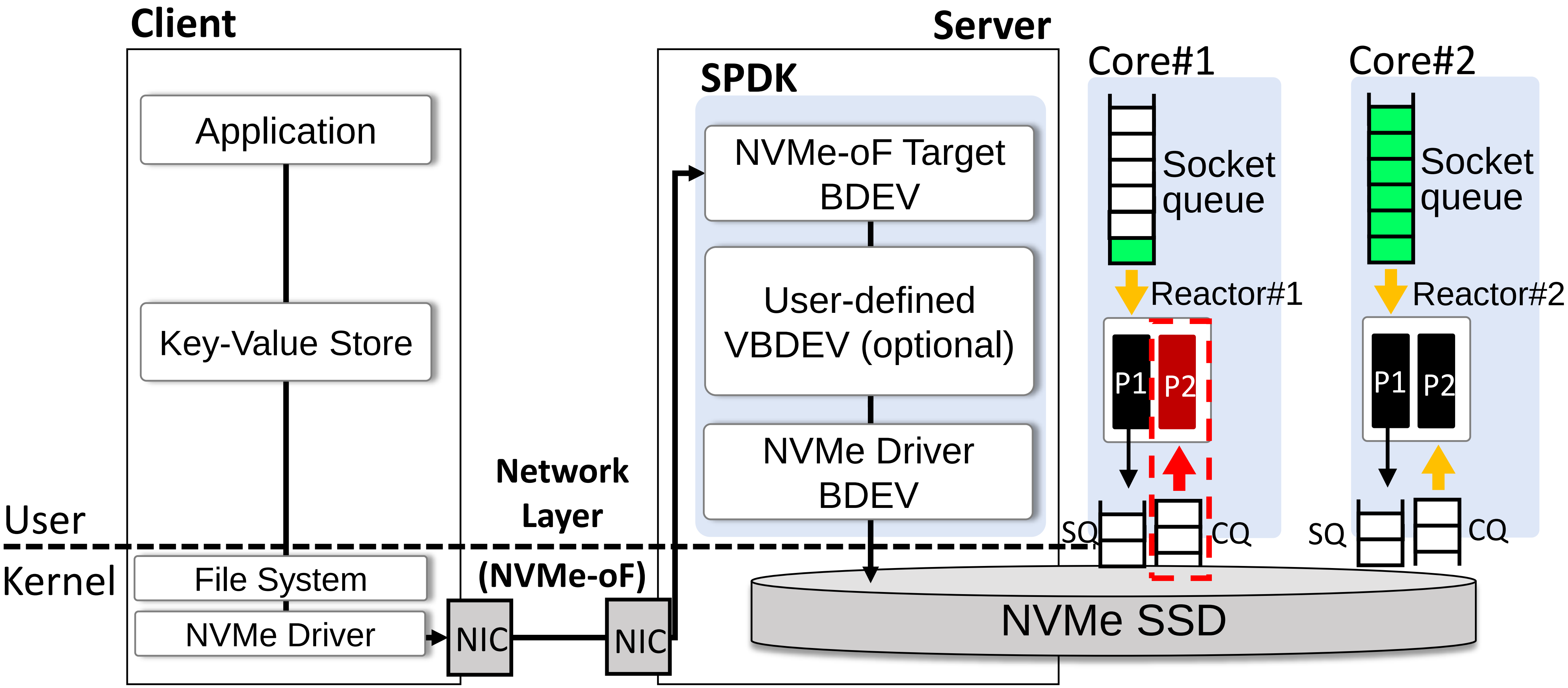
(2) SPDK-based Network-based Block Storage



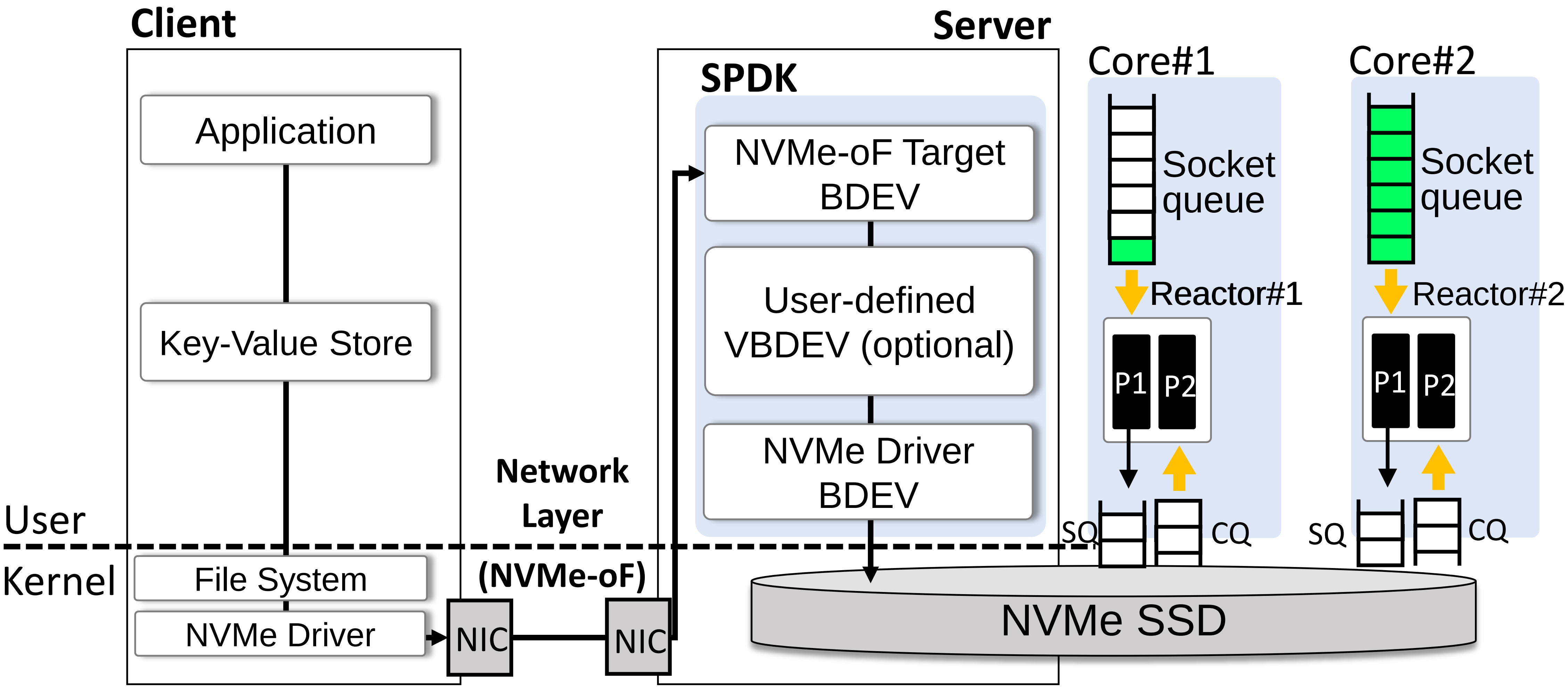
(2) SPDK-based Network-based Block Storage



(2) SPDK-based Network-based Block Storage



(2) SPDK-based Network-based Block Storage



Content

- Background
- Problem Definition
- Motivational Experiments
- OctoKV: Design and Implementation
- Evaluation
- Conclusion

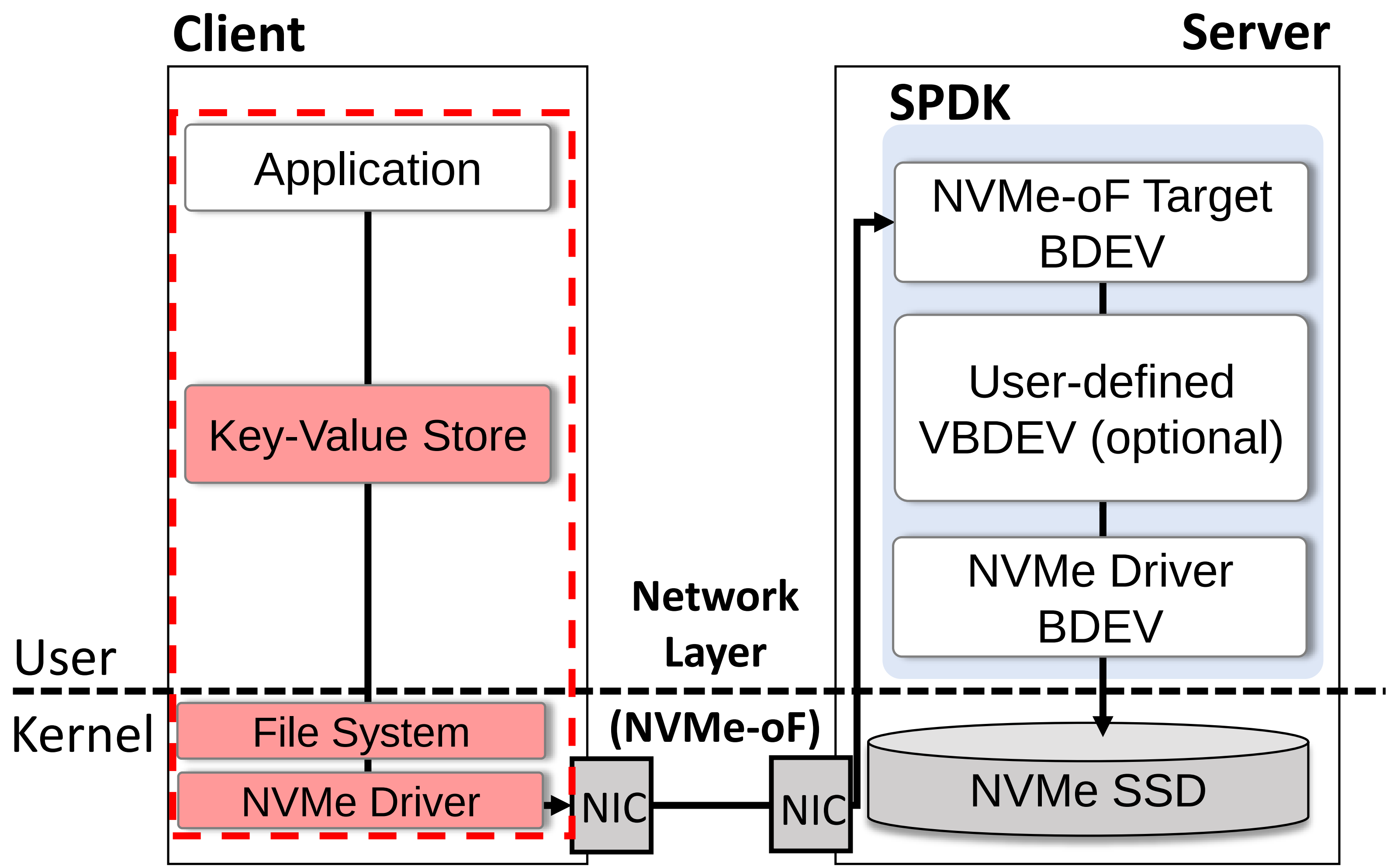
Problem Definition

SPDK-based Network-based Block Storage has the following two problems

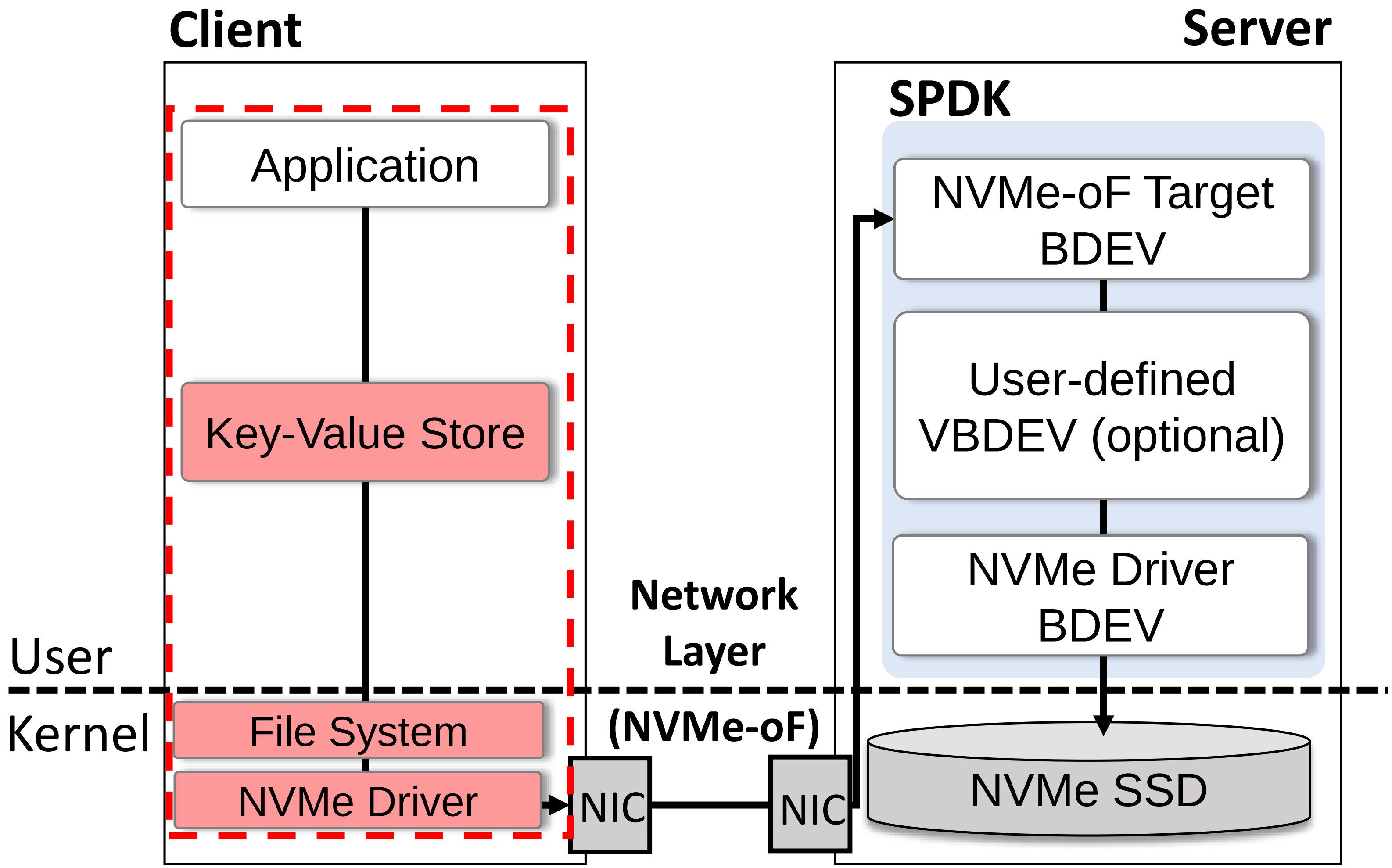
(1) High I/O Stack Overhead Problem

(2) Core Load Imbalance Problem

Problem#1: High I/O Stack Overhead

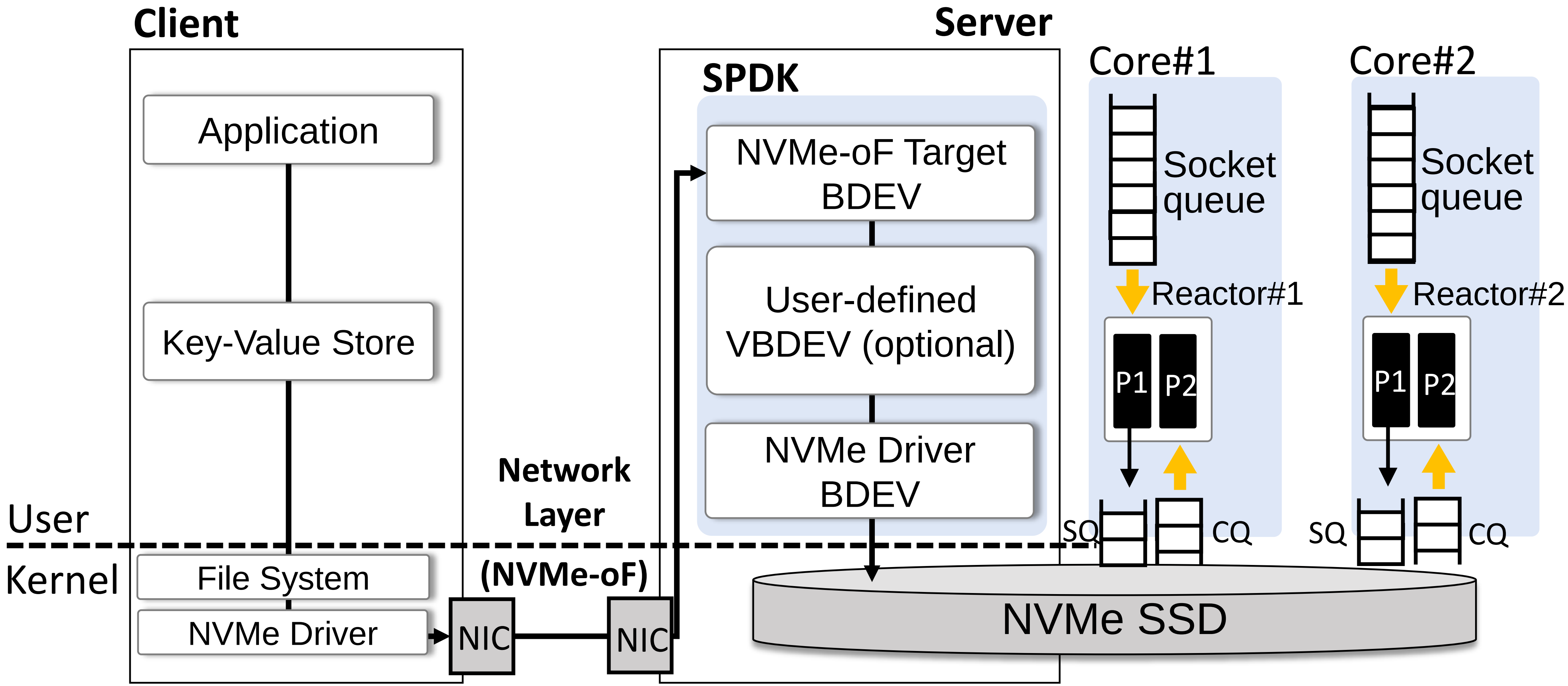


Problem#1: High I/O Stack Overhead

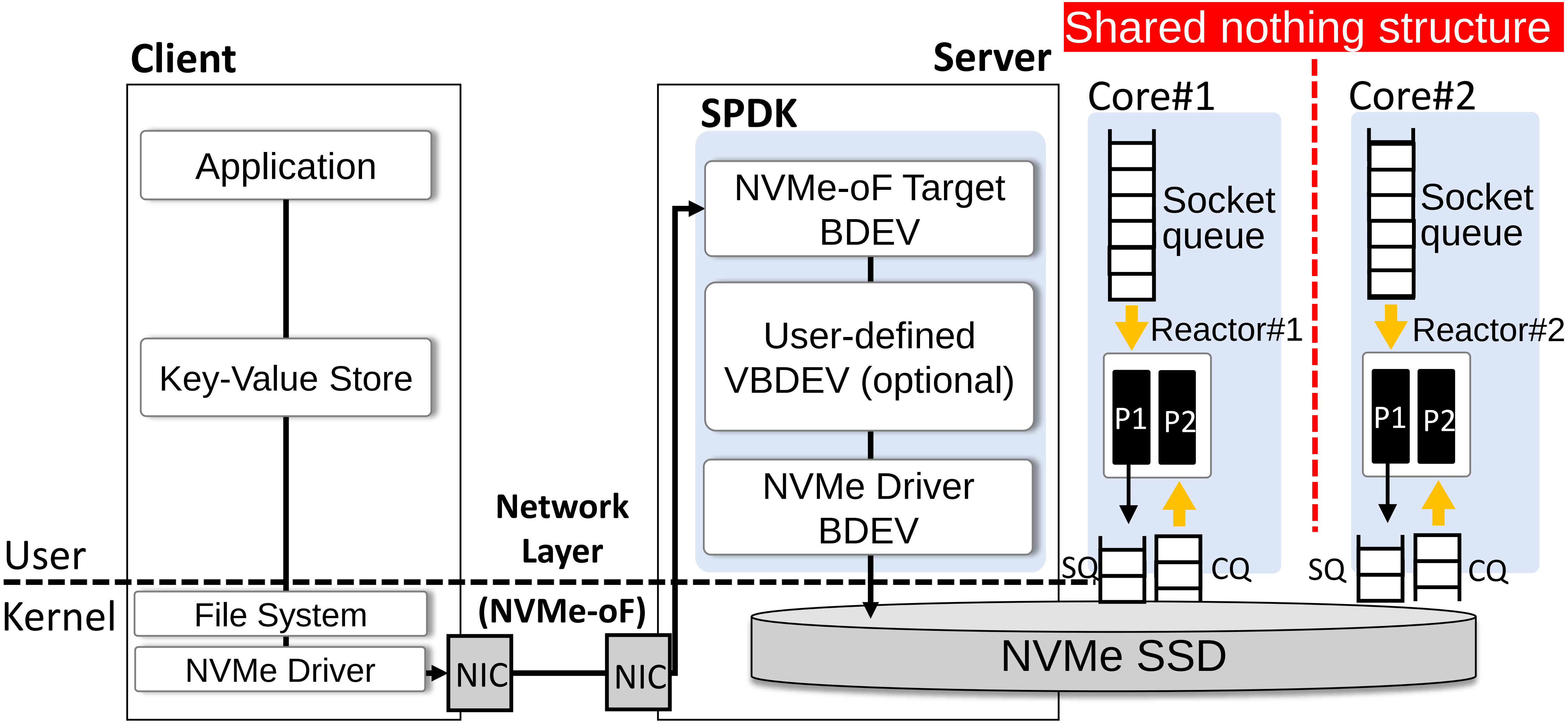


(1) KV to file, file to block address translation overhead
(2) User-to-kernel context switch overhead

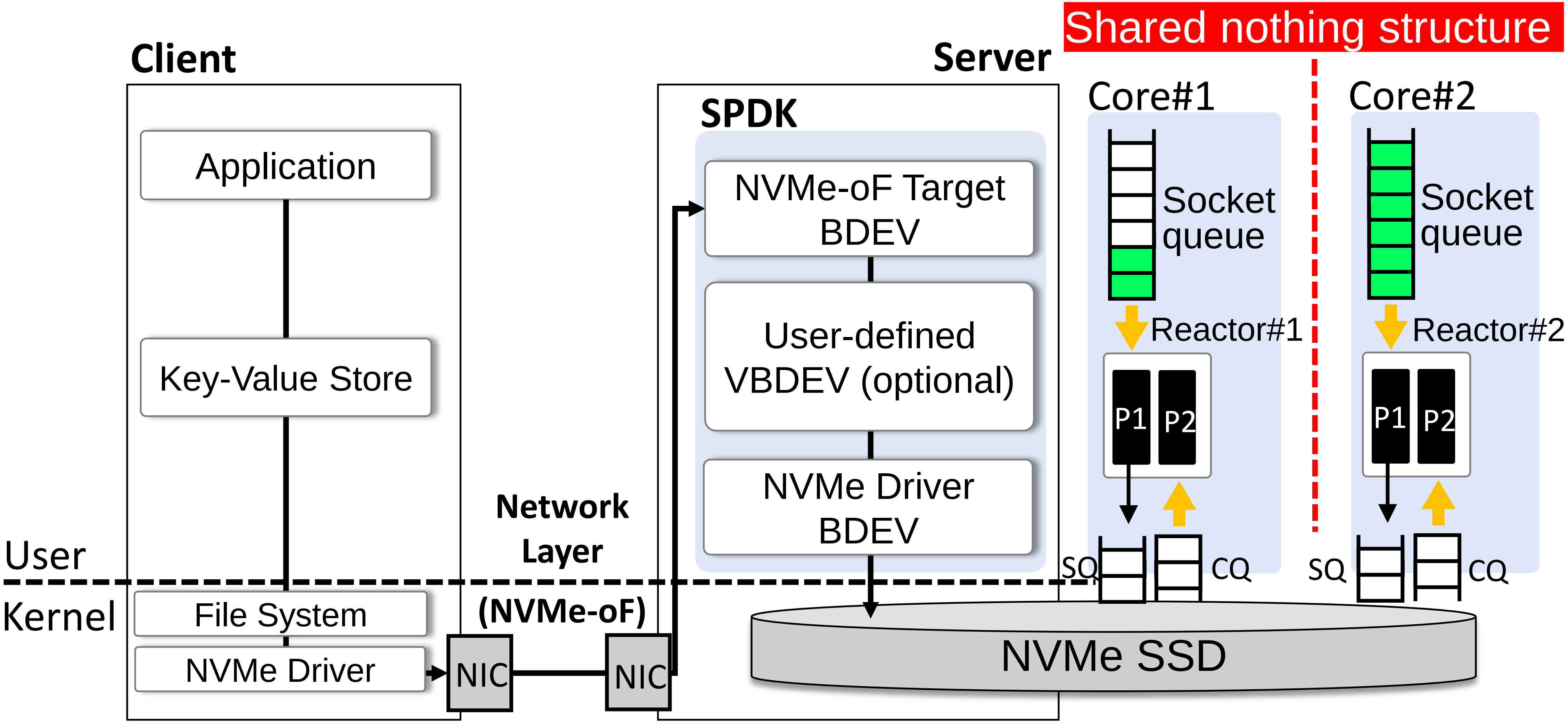
Problem#2: Core Load Imbalance



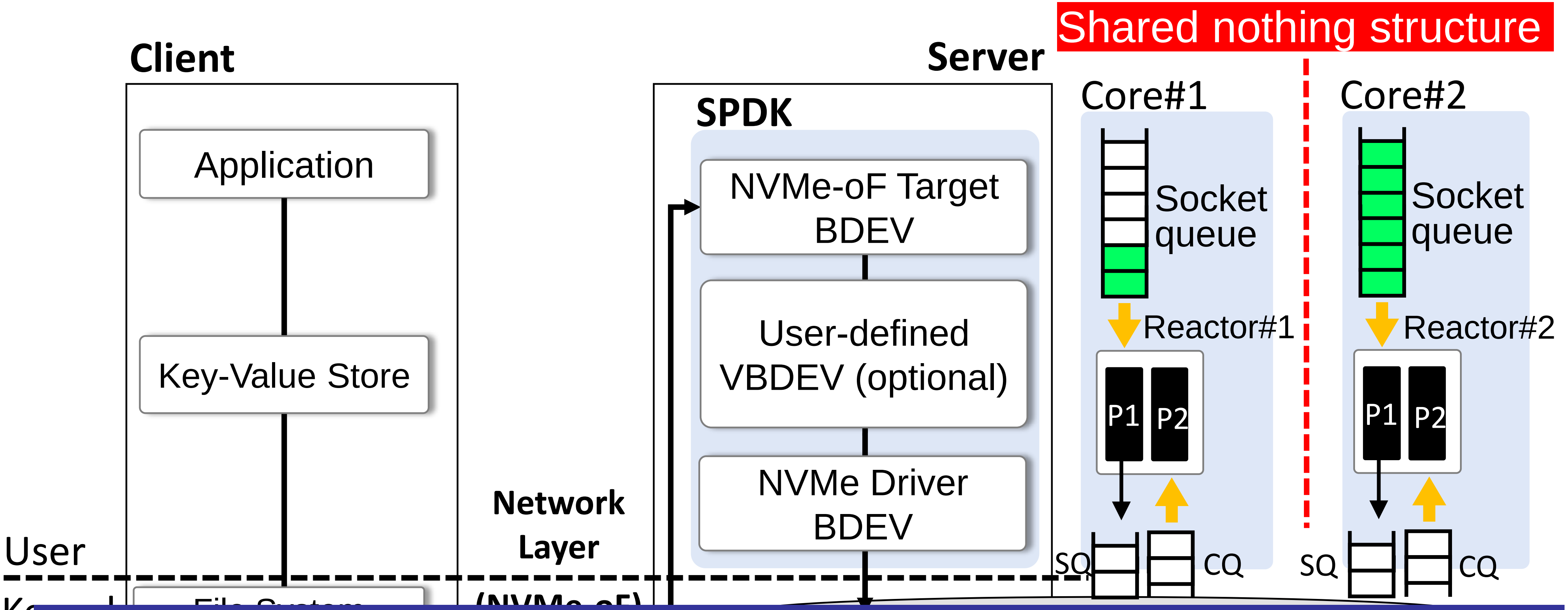
Problem#2: Core Load Imbalance



Problem#2: Core Load Imbalance



Problem#2: Core Load Imbalance



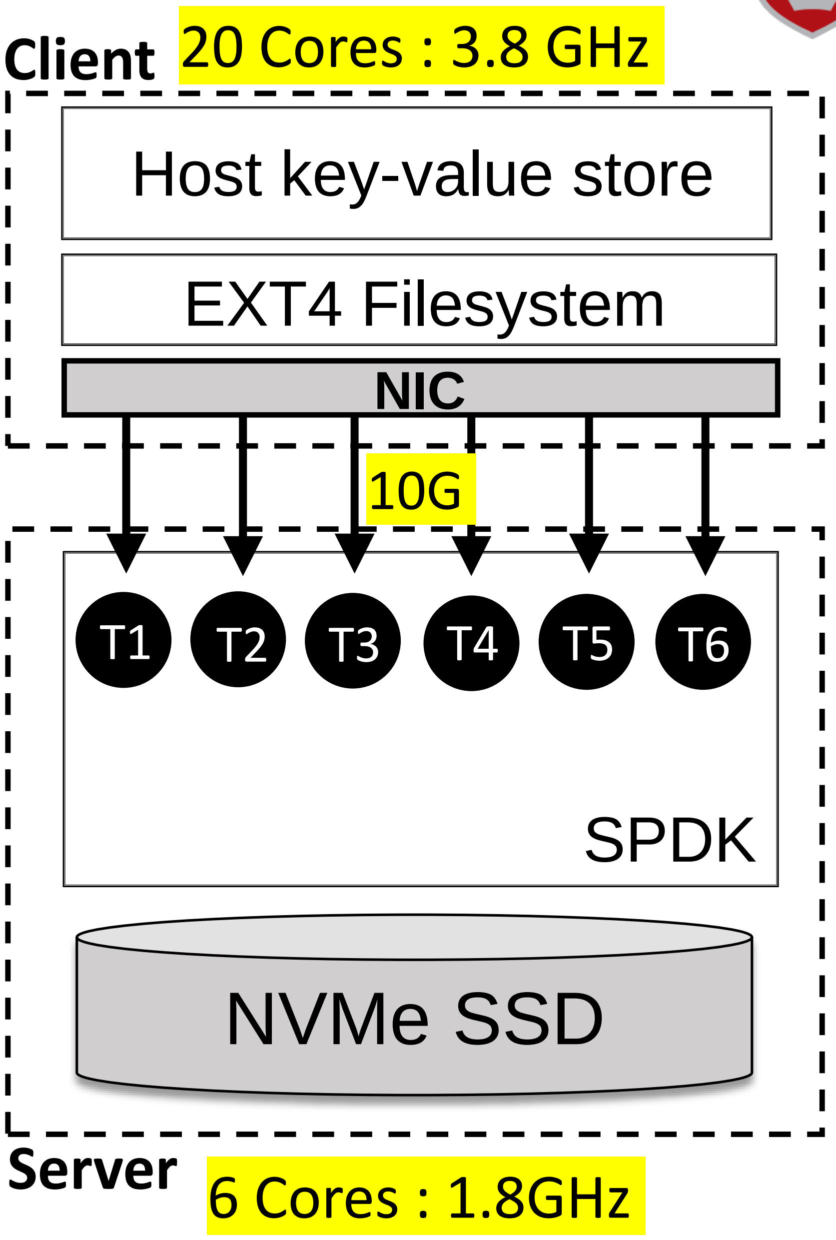
There is a lack of structures for sharing data
between CPU cores in SPDK, resulting in load imbalance

Content

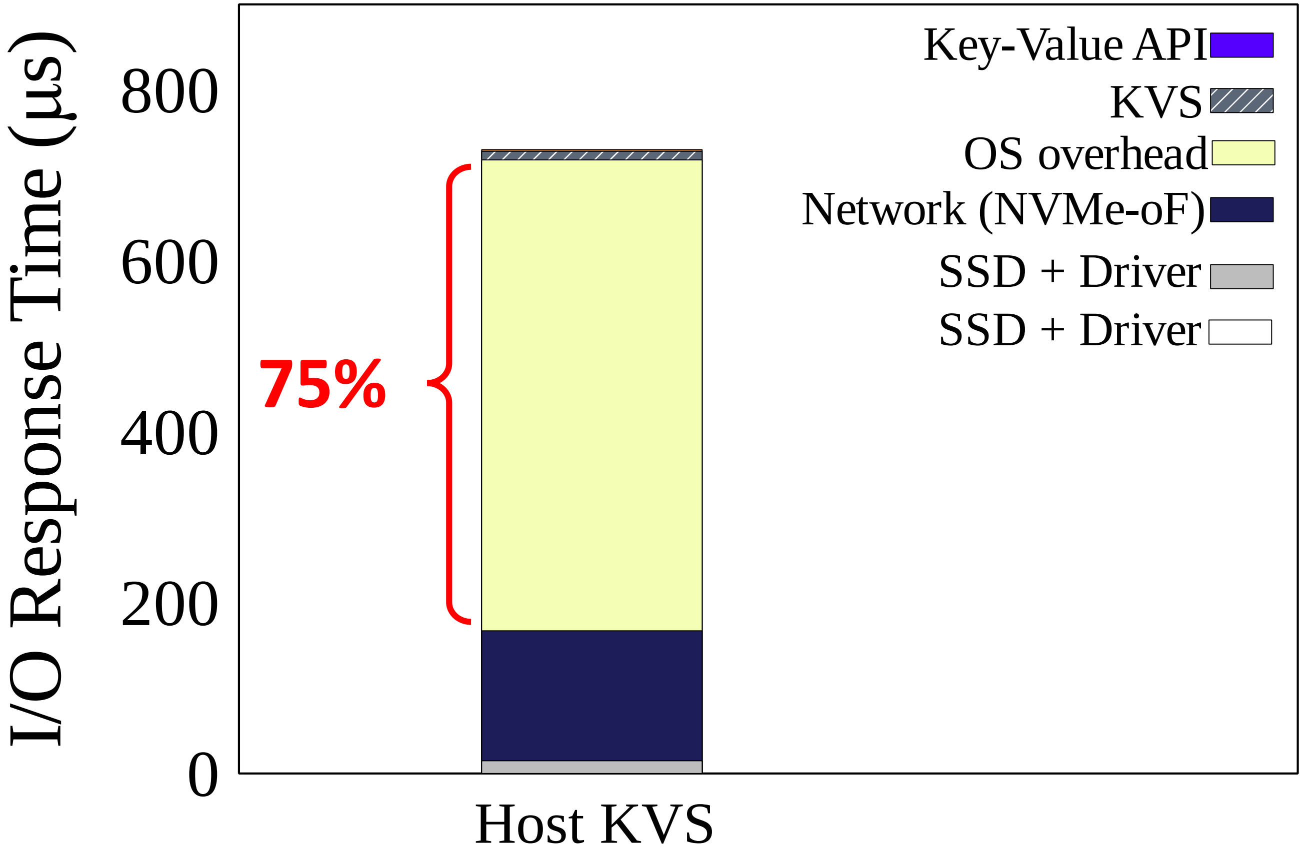
- Background
- Problem Definition
- Motivational Experiments
- OctoKV: Design and Implementation
- Evaluation
- Conclusion

Problem#1: High I/O Stack Overhead

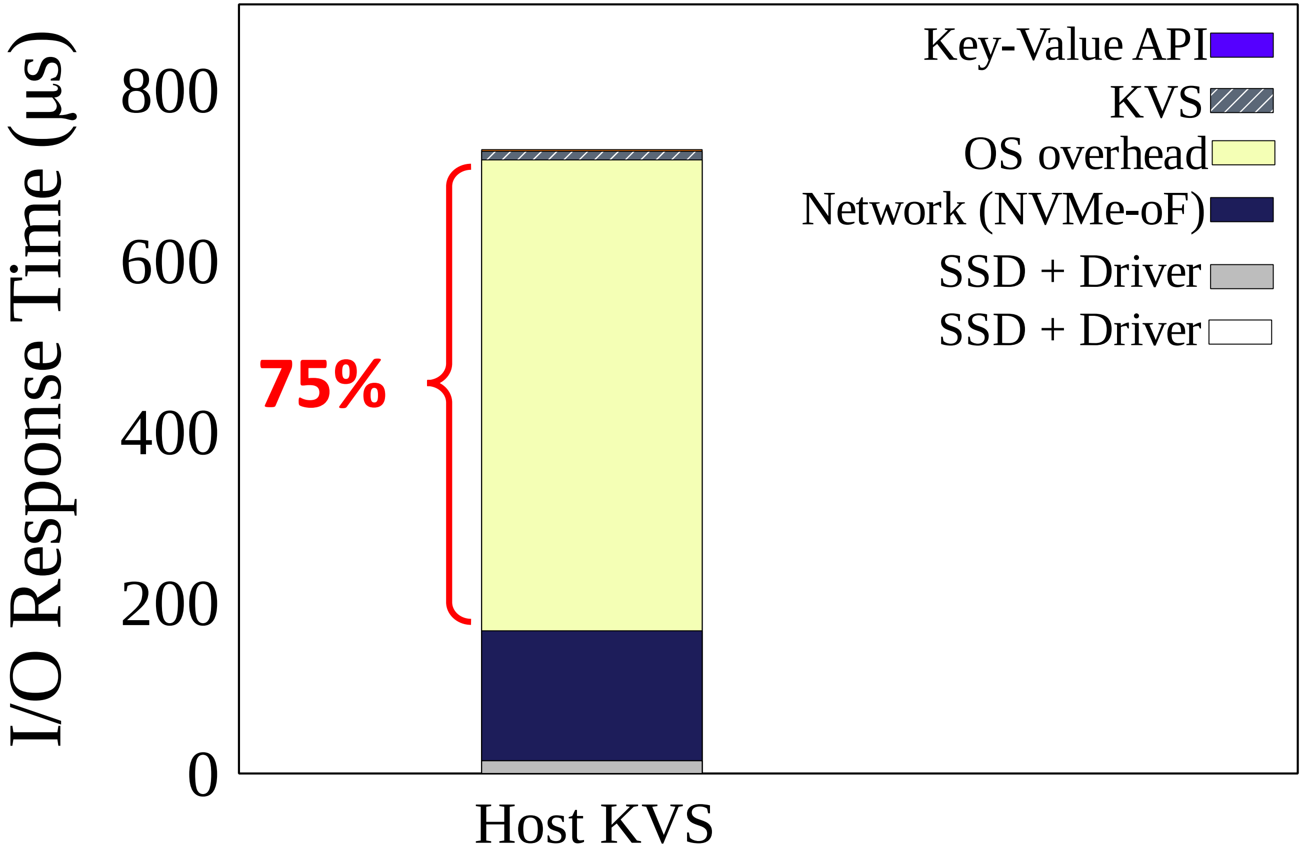
- Client
  - 20 CPU cores
  - Running a host hash-based key-value store
- Server
  - 6 CPU cores
  - Running a Linux OS using Intel SPDK
- Workloads
  - Running a db_bench
  - I/O request size = 16KB



Problem#1: High I/O Stack Overhead



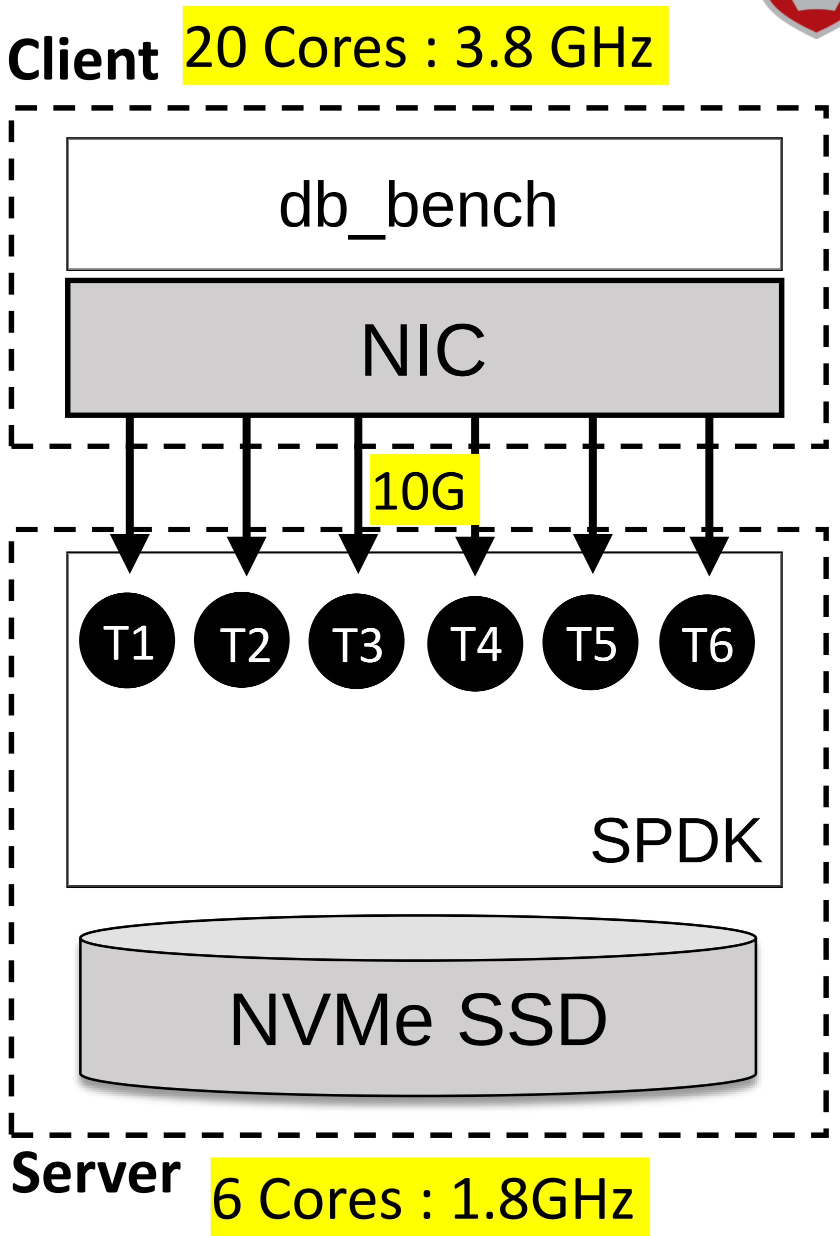
Problem#1: High I/O Stack Overhead



Running key-values store on top of the file system
in a disaggregated architecture has significant I/O overhead

Problem#2: Core Load Imbalance

- Workloads
  - § Running a db_bench
    - 1) Light workload
      - *7 I/O threads issue write I/Os*
    - 2) Heavy workload
      - *12 I/O threads issue write I/Os*
  - § I/O request size = 16KB



Problem#2: Core Load Imbalance

Thread Queue Depth for each core/SPDK thread

Queue Depth	Core#1	Core#2	Core#3	Core#4	Core#5	Core#6	Avg	Stdev
Light Workload (Put)	2.00	2.21	0.75	1.58	0.67	0.33	1.26	0.78
Heavy Workload (Put)	5.25	5.48	2.00	2.06	2.13	2.13	3.18	1.70
Light Workload (Get)	3.95	4.23	1.27	1.36	2.00	1.82	2.43	1.31
Heavy Workload (Get)	6.06	6.54	2.69	2.62	2.92	2.65	3.91	1.86

Problem#2: Core Load Imbalance

Thread Queue Depth for each core/SPDK thread

Queue Depth	Core#1	Core#2	Core#3	Core#4	Core#5	Core#6	Avg	Stdev
Light Workload (Put)	2.00	2.21	0.75	1.58	0.67	0.33	1.26	0.78
Heavy Workload (Put)	5.25	5.48	2.00	2.06	2.13	2.13	3.18	1.70
Light Workload (Get)	3.95	4.23	1.27	1.36	2.00	1.82	2.43	1.31
Heavy Workload (Get)	6.06	6.54	2.69	2.62	2.92	2.65	3.91	1.86

deep depth

shallow depth

Problem#2: Core Load Imbalance

Thread Queue Depth for each core/SPDK thread

Queue Depth	Core#1	Core#2	Core#3	Core#4	Core#5	Core#6	Avg	Stdev
Light Workload (Put)	2.00	2.21	0.75	1.58	0.67	0.33	1.26	0.78
Heavy Workload (Put)	5.25	5.48	2.00	2.06	2.13	2.13	3.18	1.70
Light Workload (Get)	3.95	4.23	1.27	1.36	2.00	1.82	2.43	1.31
Heavy Workload (Get)	6.06	6.54	2.69	2.62	2.92	2.65	3.91	1.86

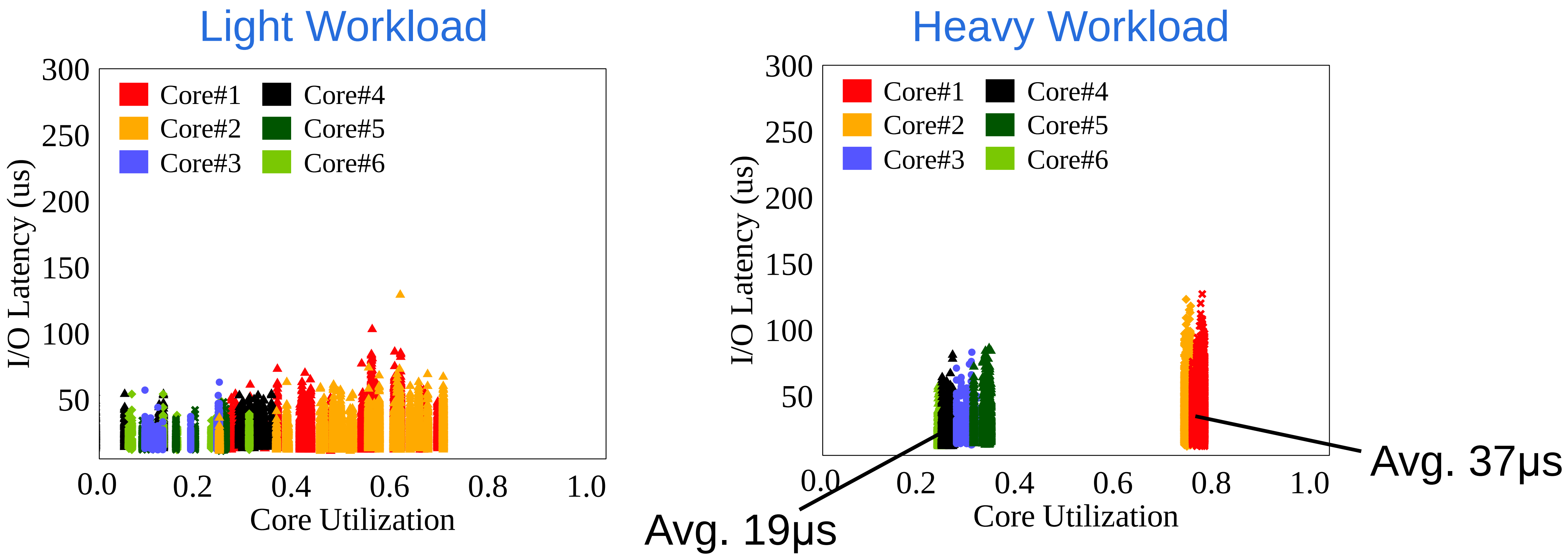
deep depth

shallow depth

The amount of NVMe commands delivered to the core is imbalanced

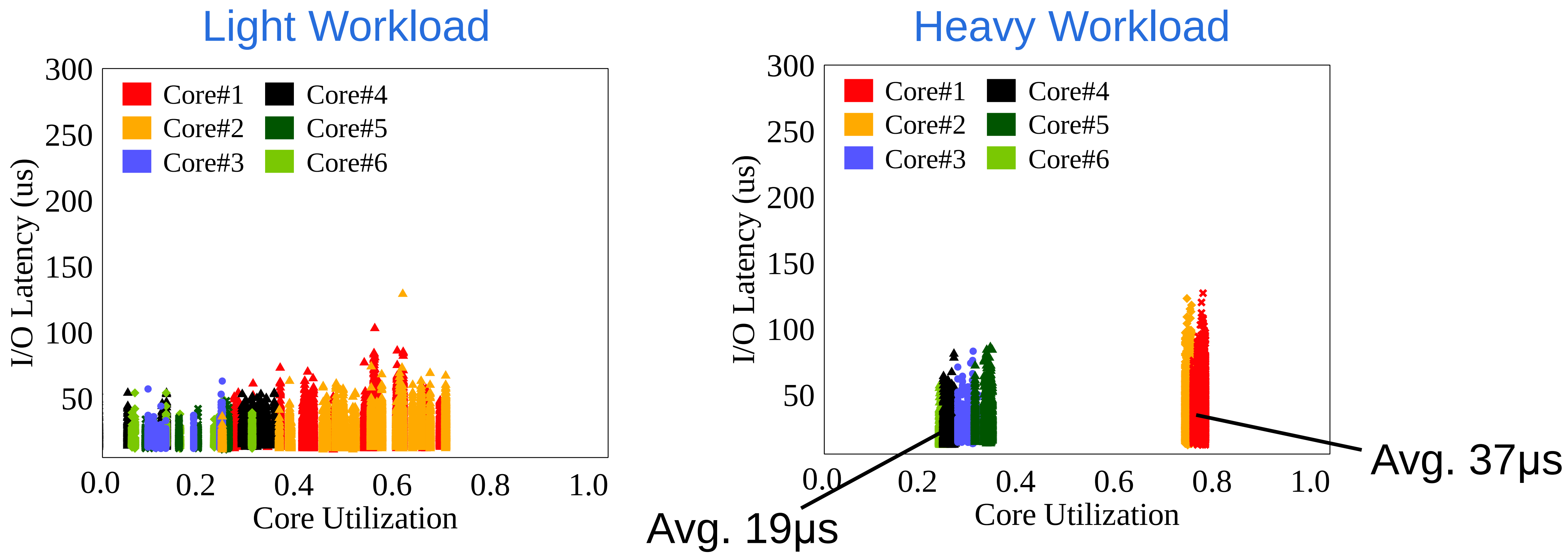
Problem#2: Core Load Imbalance

Core Utilization vs I/O Latency



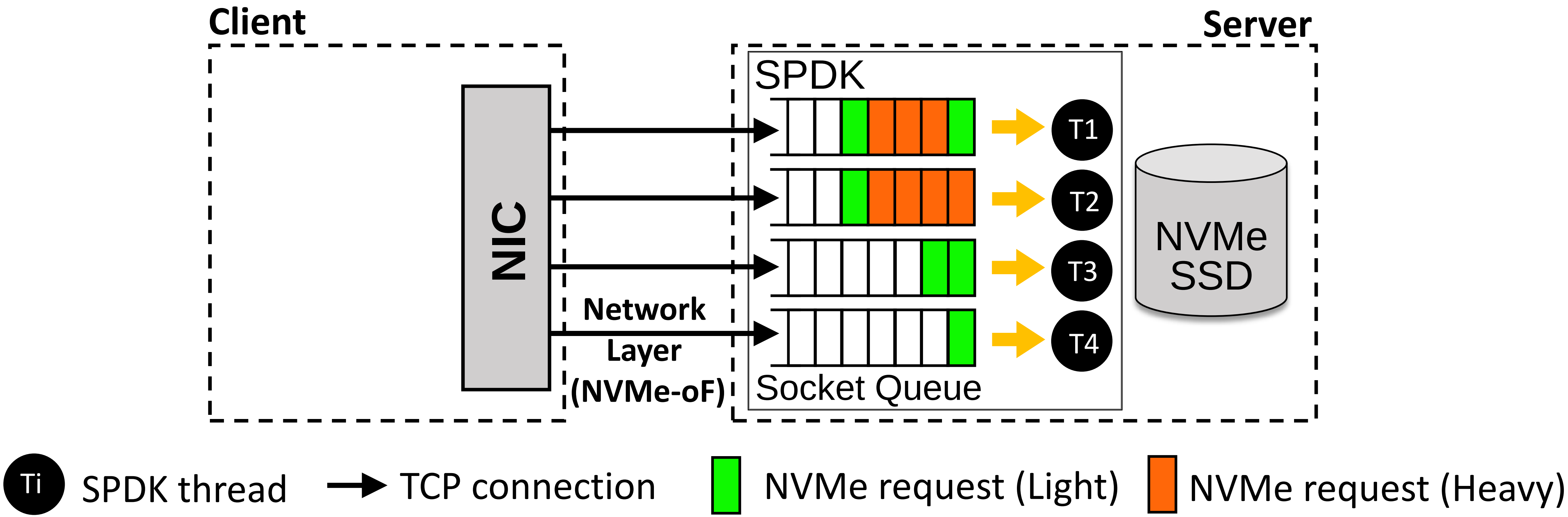
Problem#2: Core Load Imbalance

Core Utilization vs I/O Latency

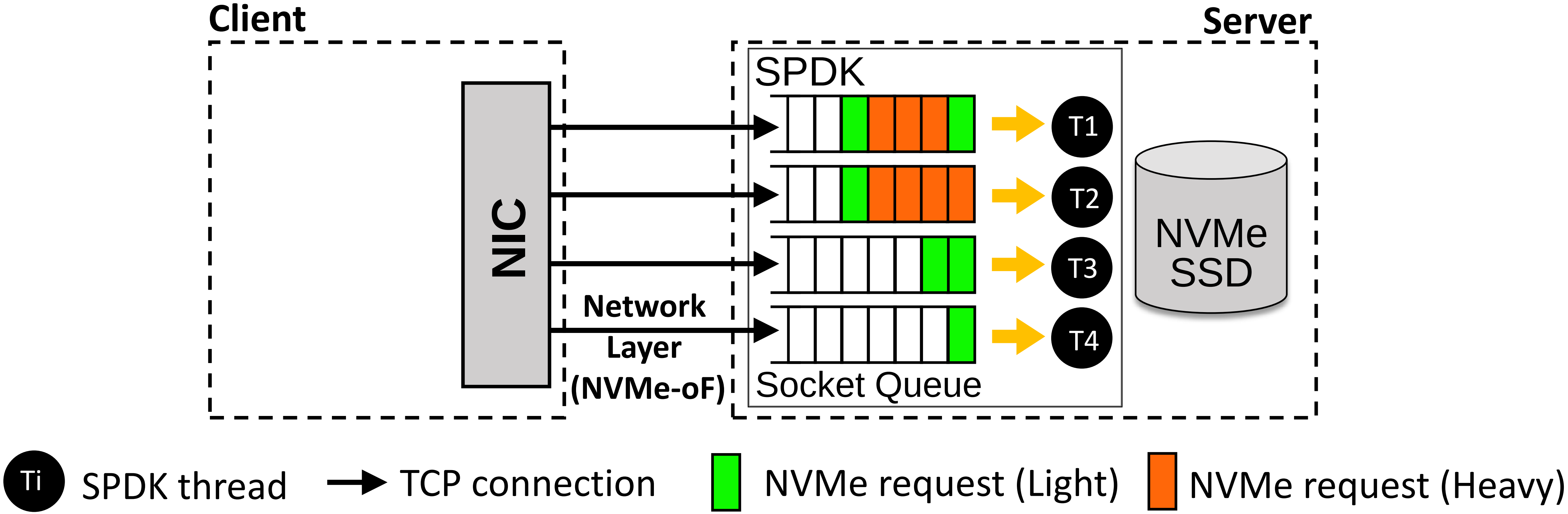


Core utilization between cores forms a bimodal distribution,
I/Os processed on busy cores show high latency

Problem#2: Core Load Imbalance



Problem#2: Core Load Imbalance



Heavy I/O requests increase the CPU load more
and eventually increase the load imbalance problem

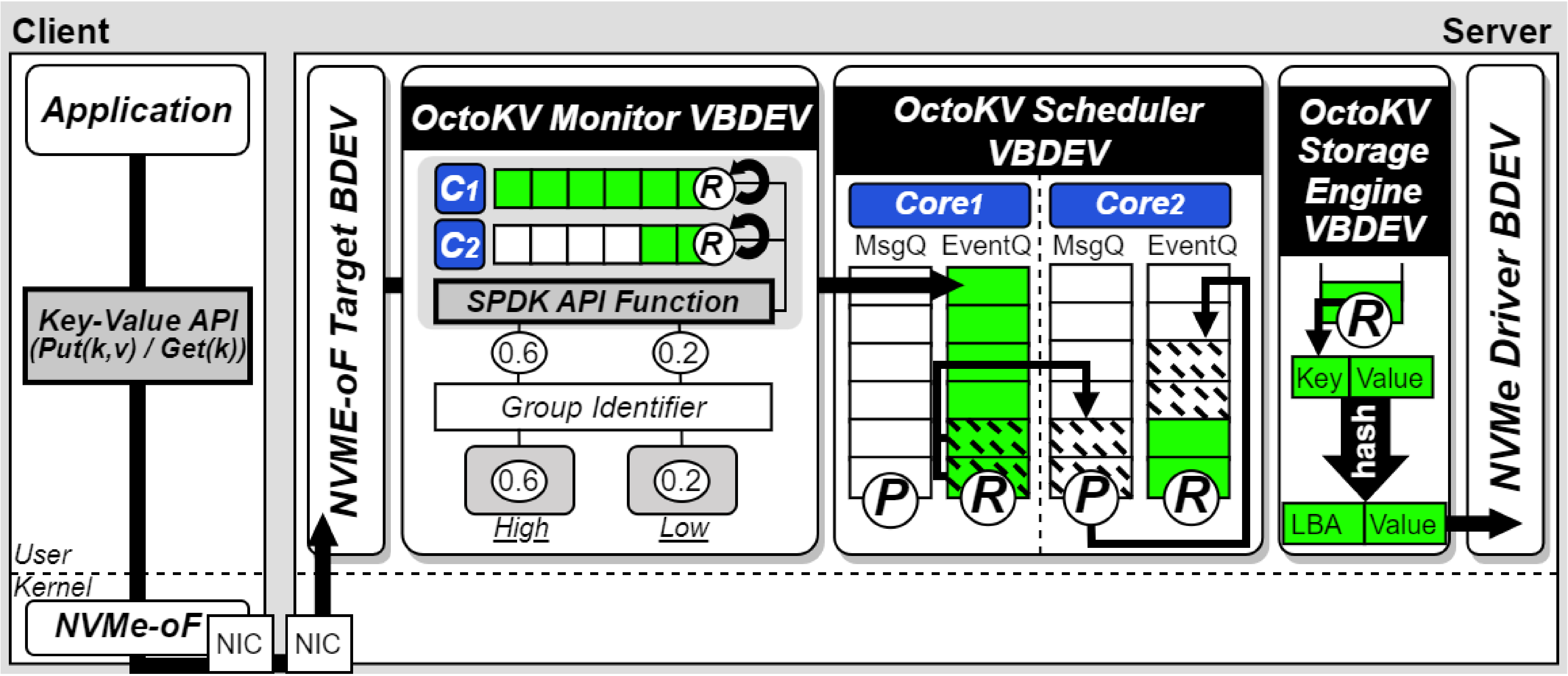
We propose an **OctoKV**

An Agile Network-Based Key-Value Storage System with Robust Load Orchestration

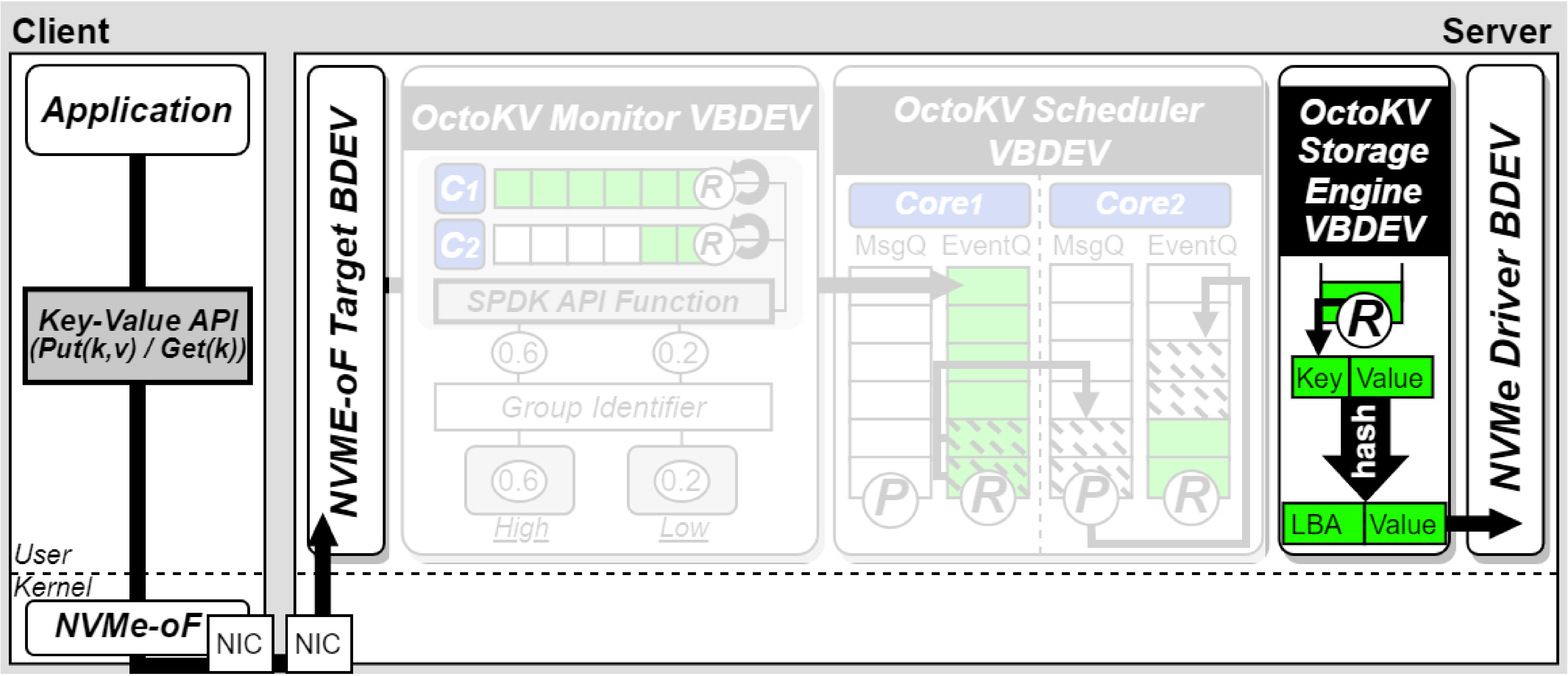
Content

- Background
- Problem Definition
- Motivational Experiments
- OctoKV: Design and Implementation
- Evaluation
- Conclusion

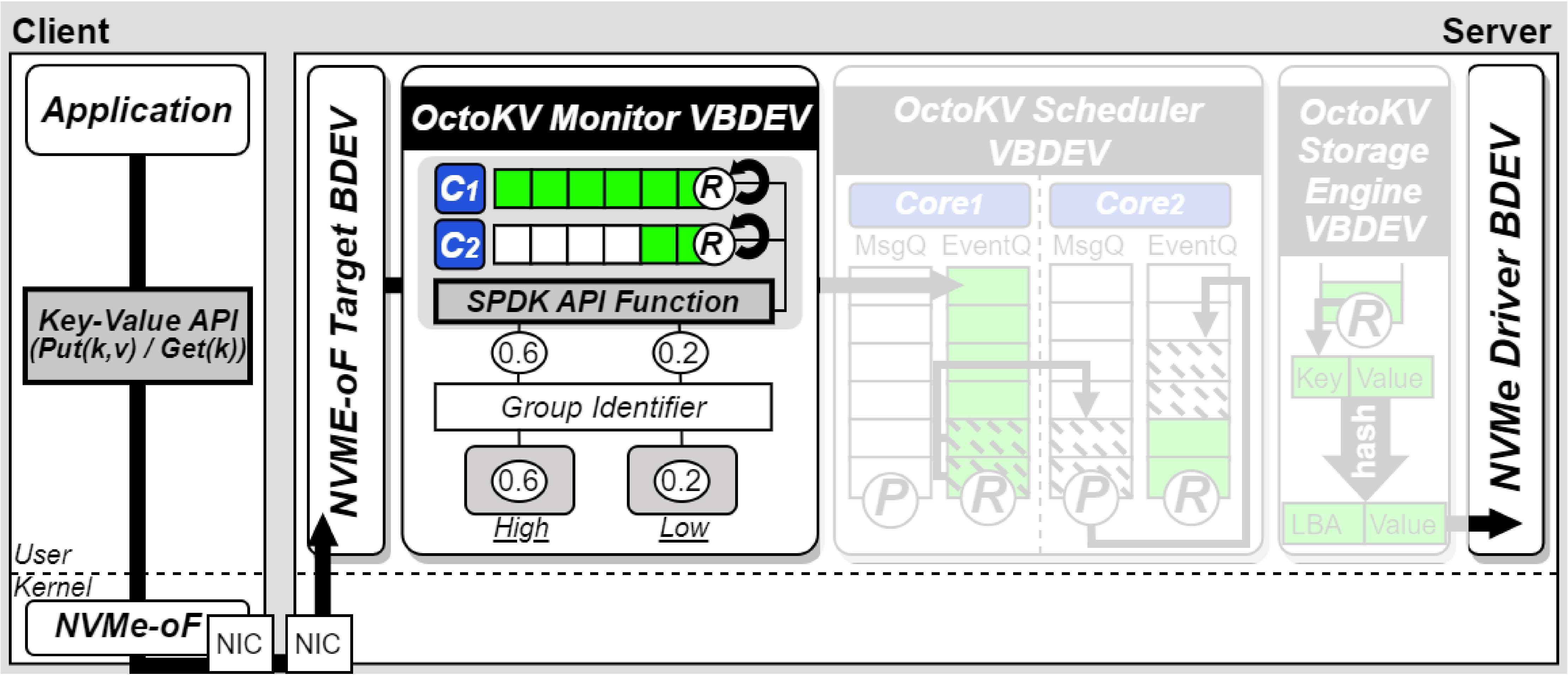
OctoKV Overview



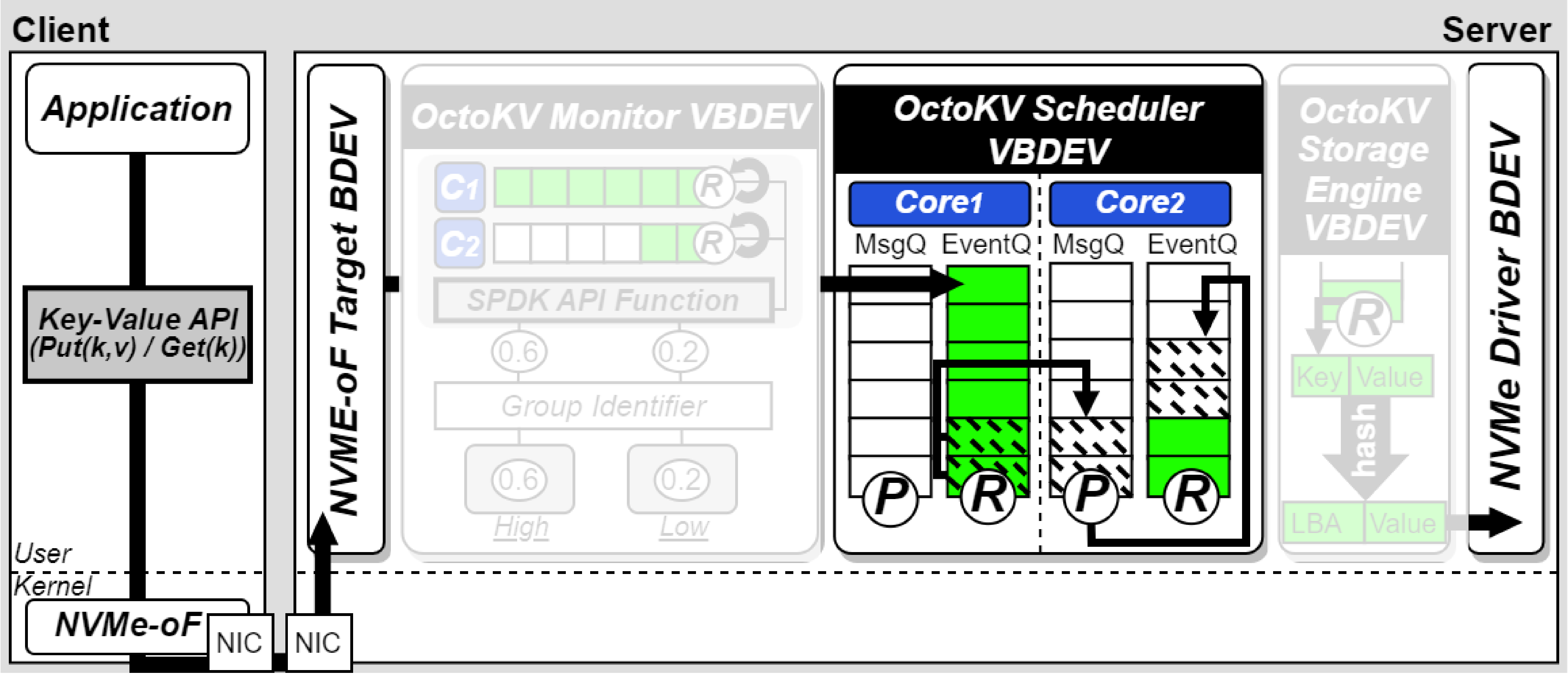
OctoKV Overview



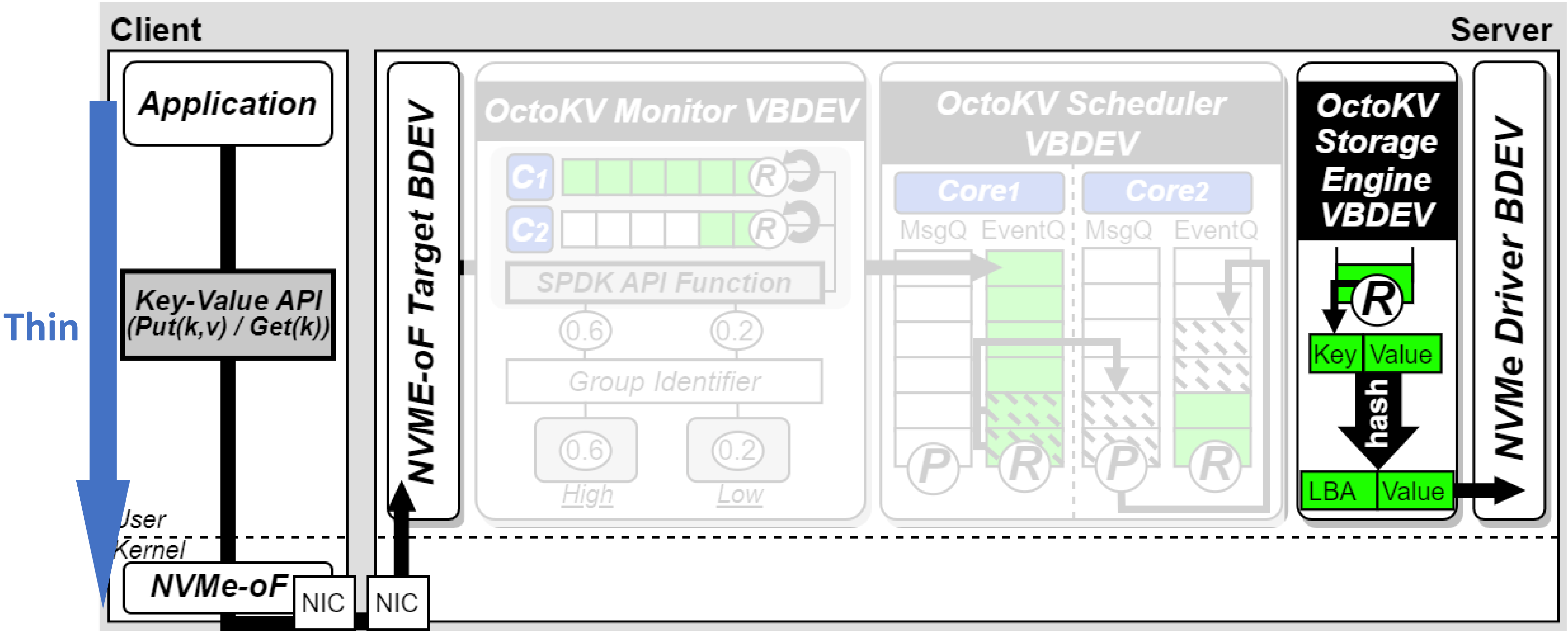
OctoKV Overview



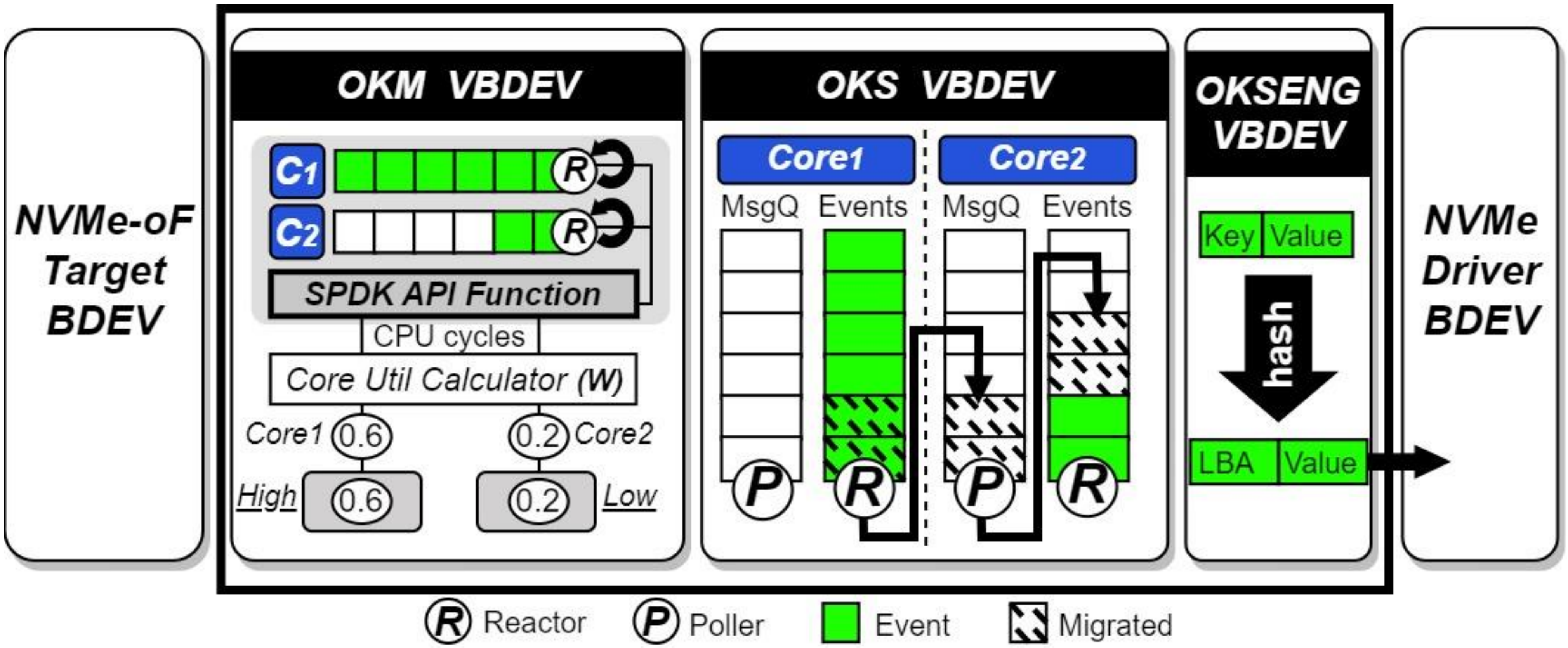
OctoKV Overview



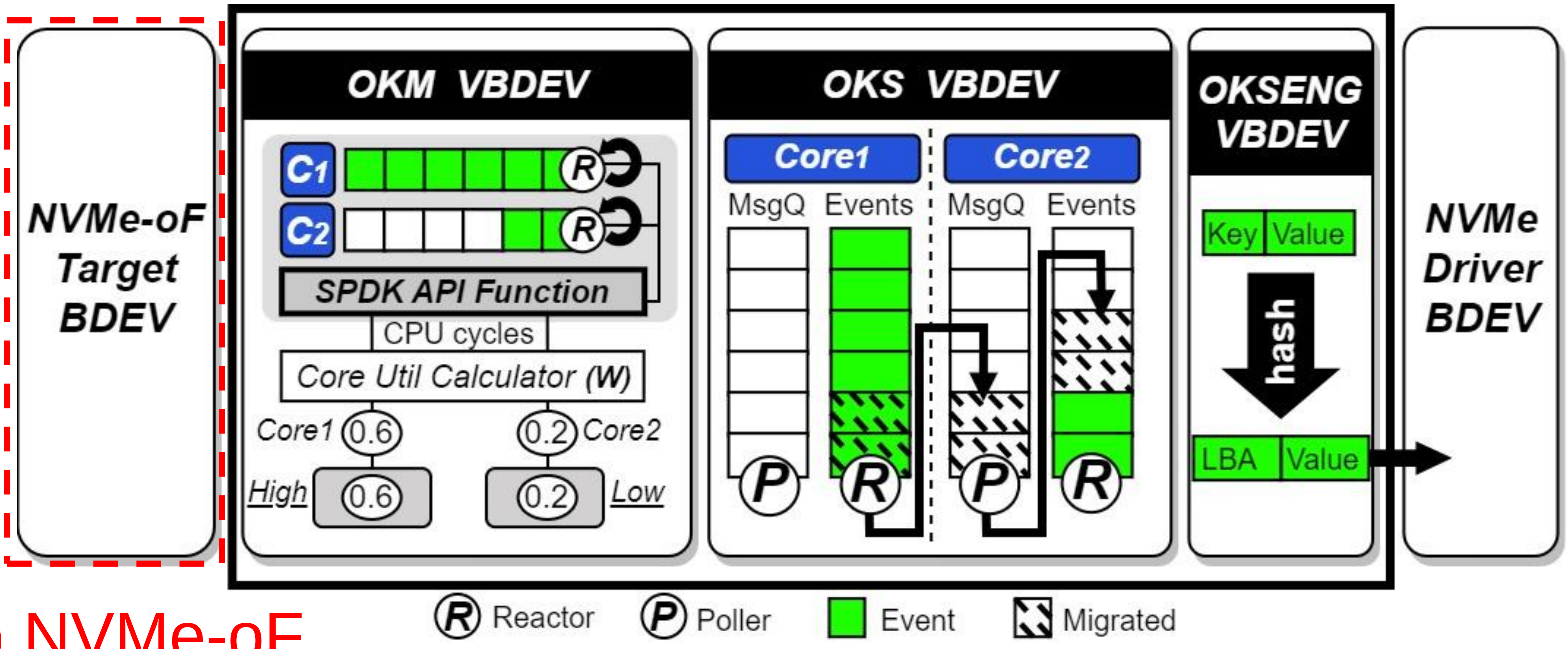
OctoKV Overview



OctoKV Overview



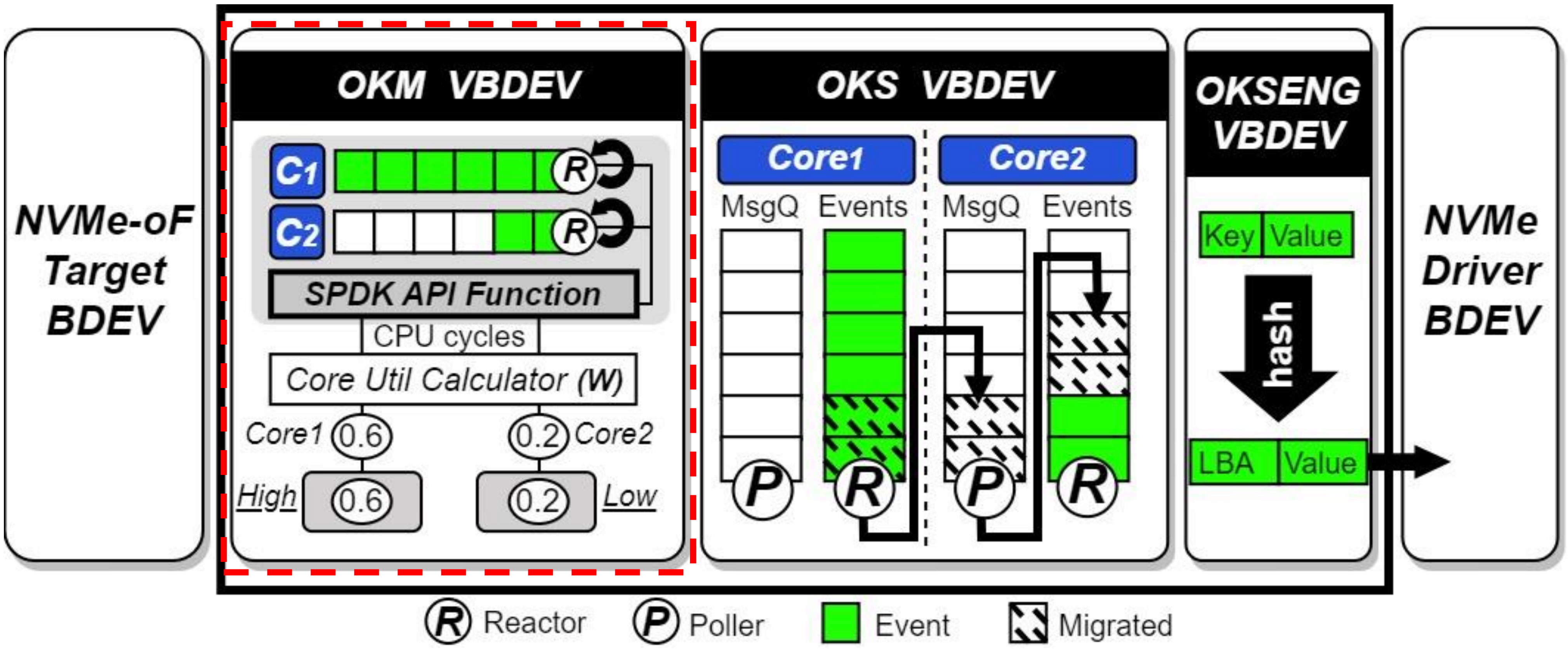
OctoKV Overview



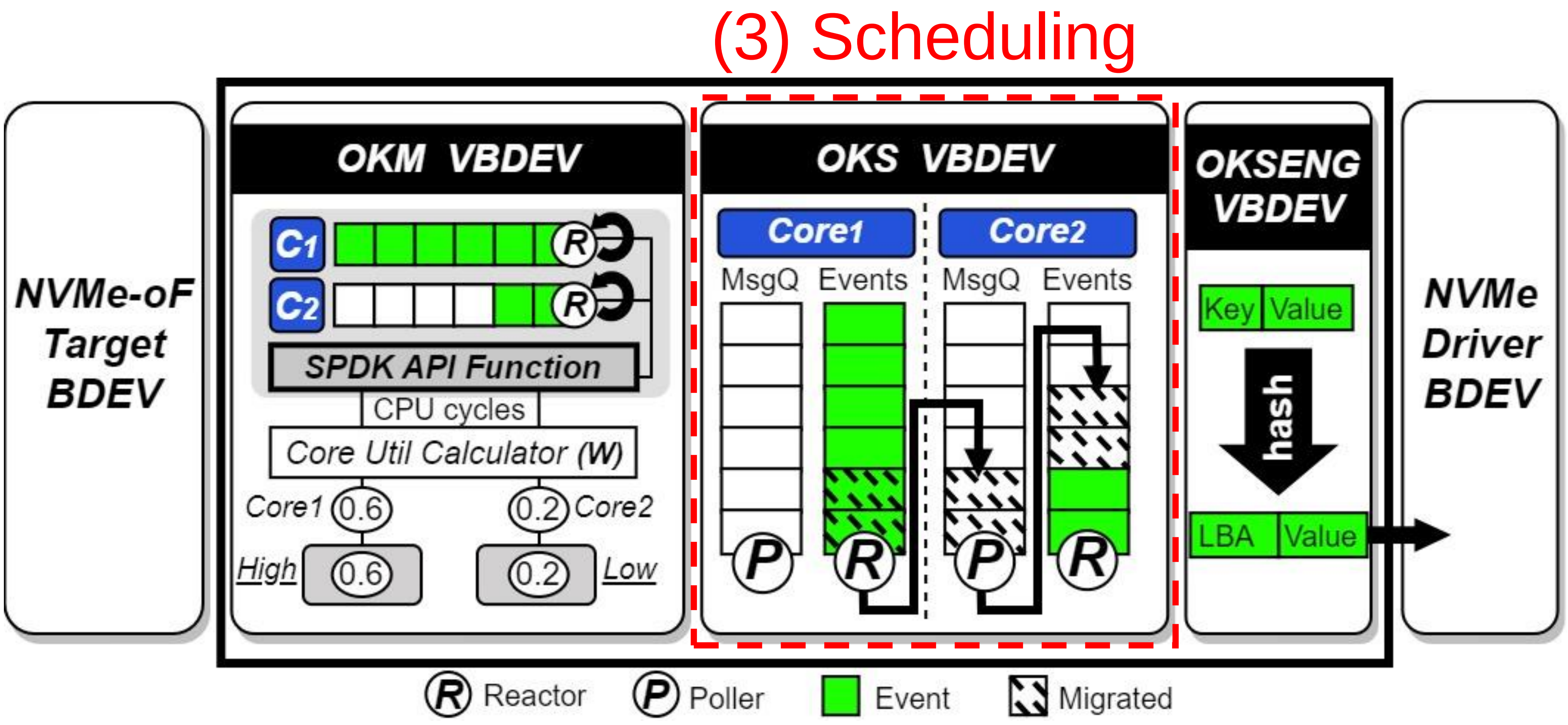
(1) NVMe-oF

OctoKV Overview

(2) Monitoring

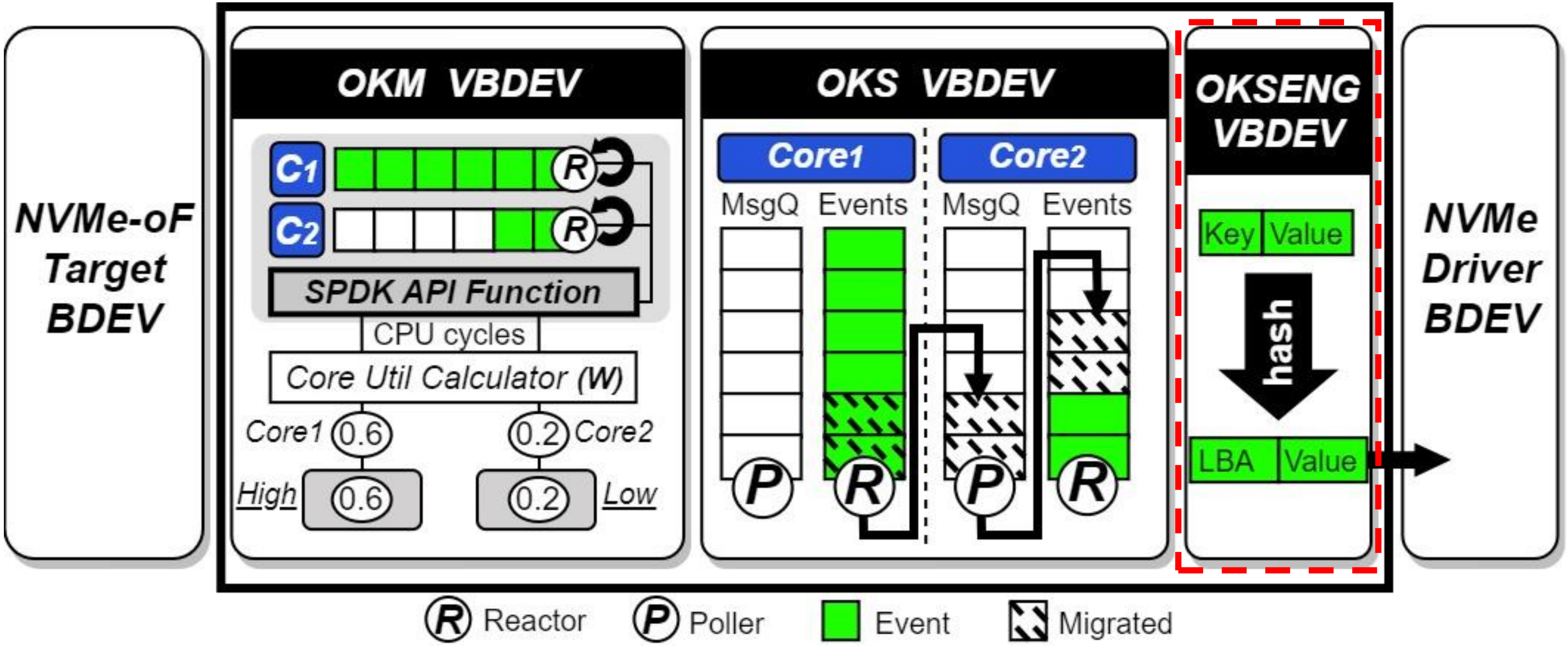


OctoKV Overview

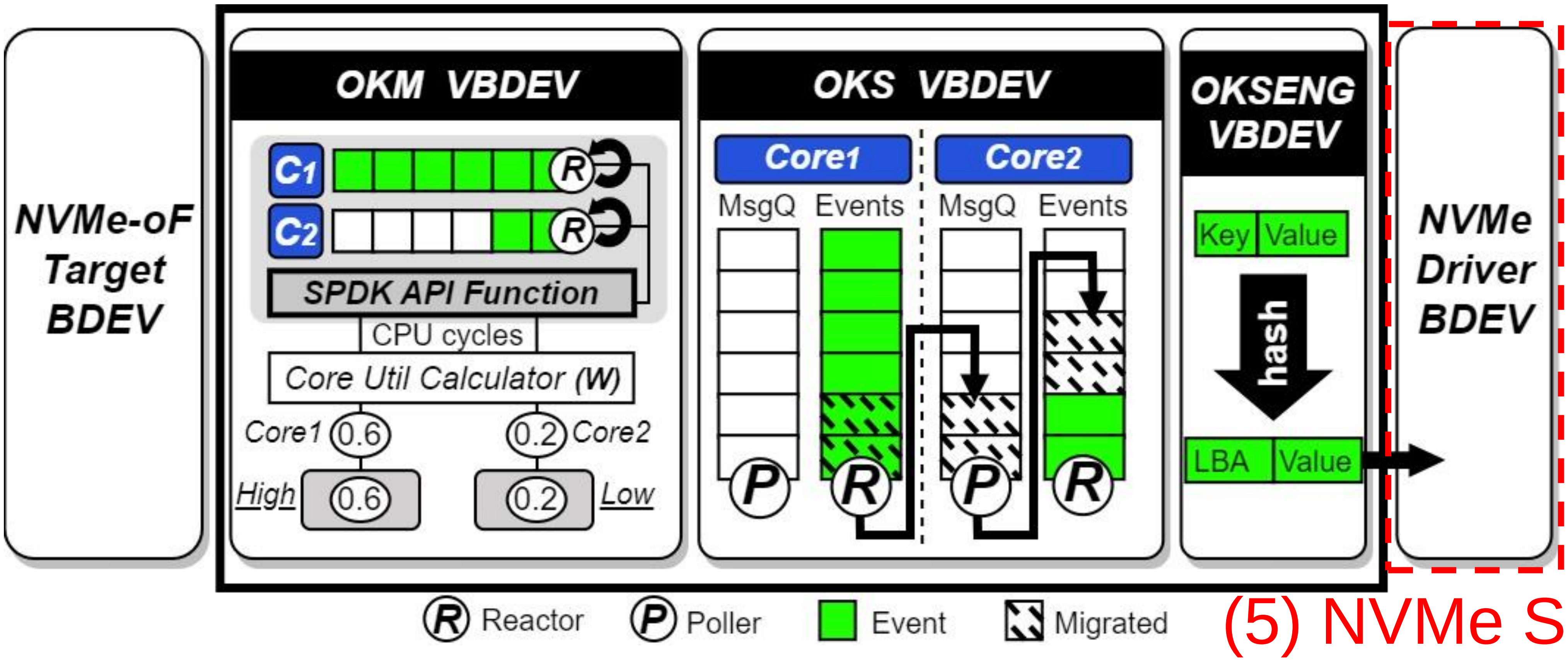


OctoKV Overview

(4) Key-Value Store

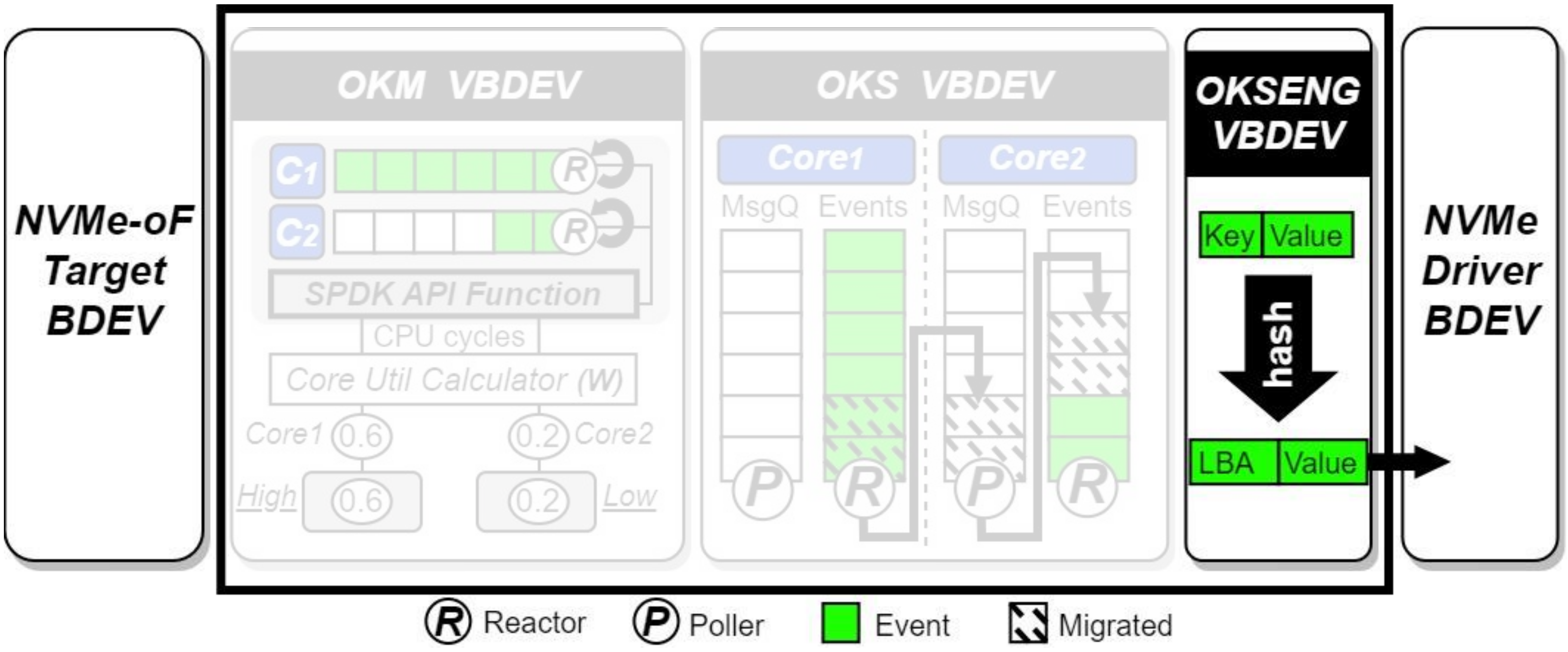


OctoKV Overview



(5) NVMe Submit

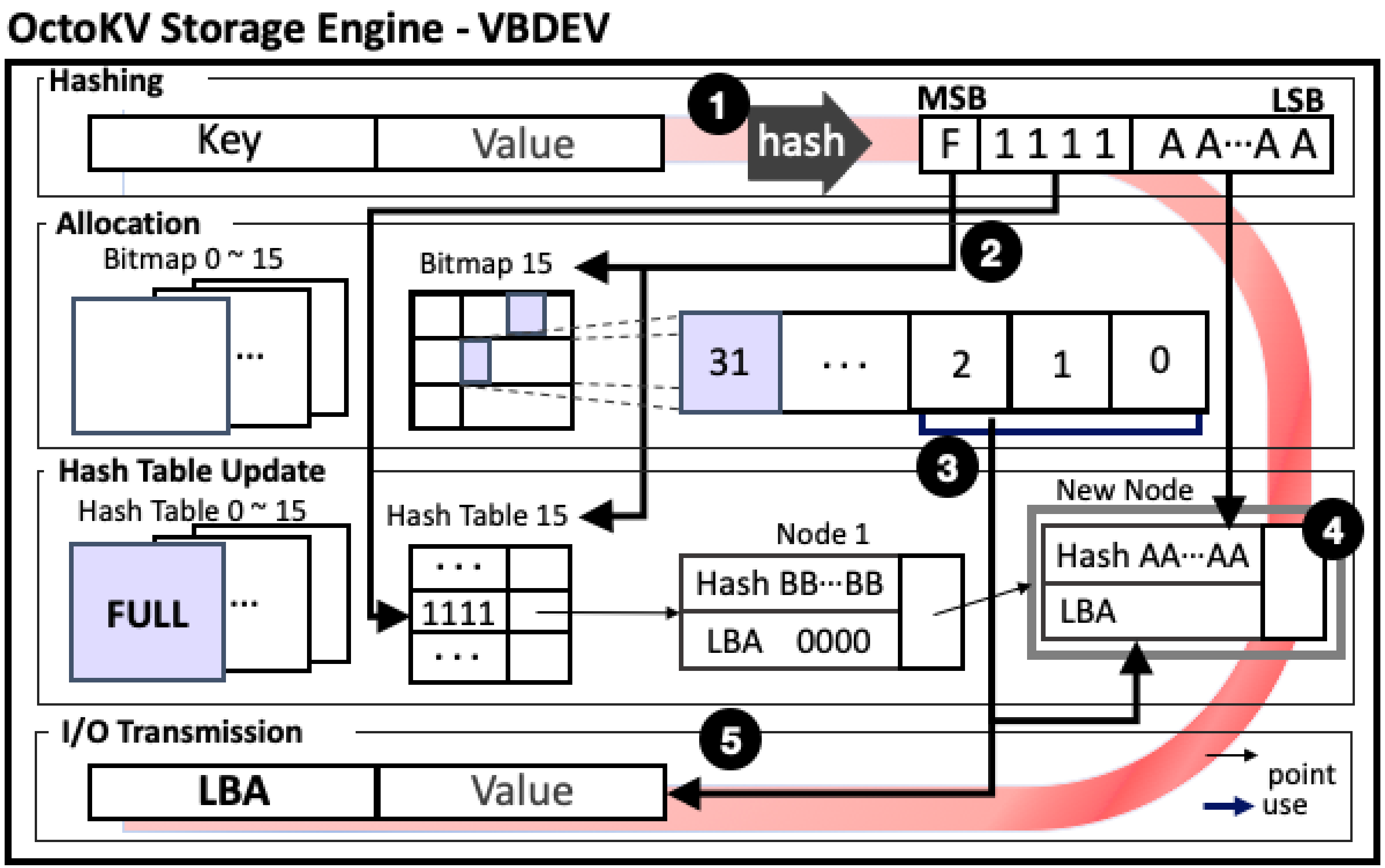
OctoKV: Design and Implementation



Module#1: Storage Engine

Bitmap Array

Hash Table

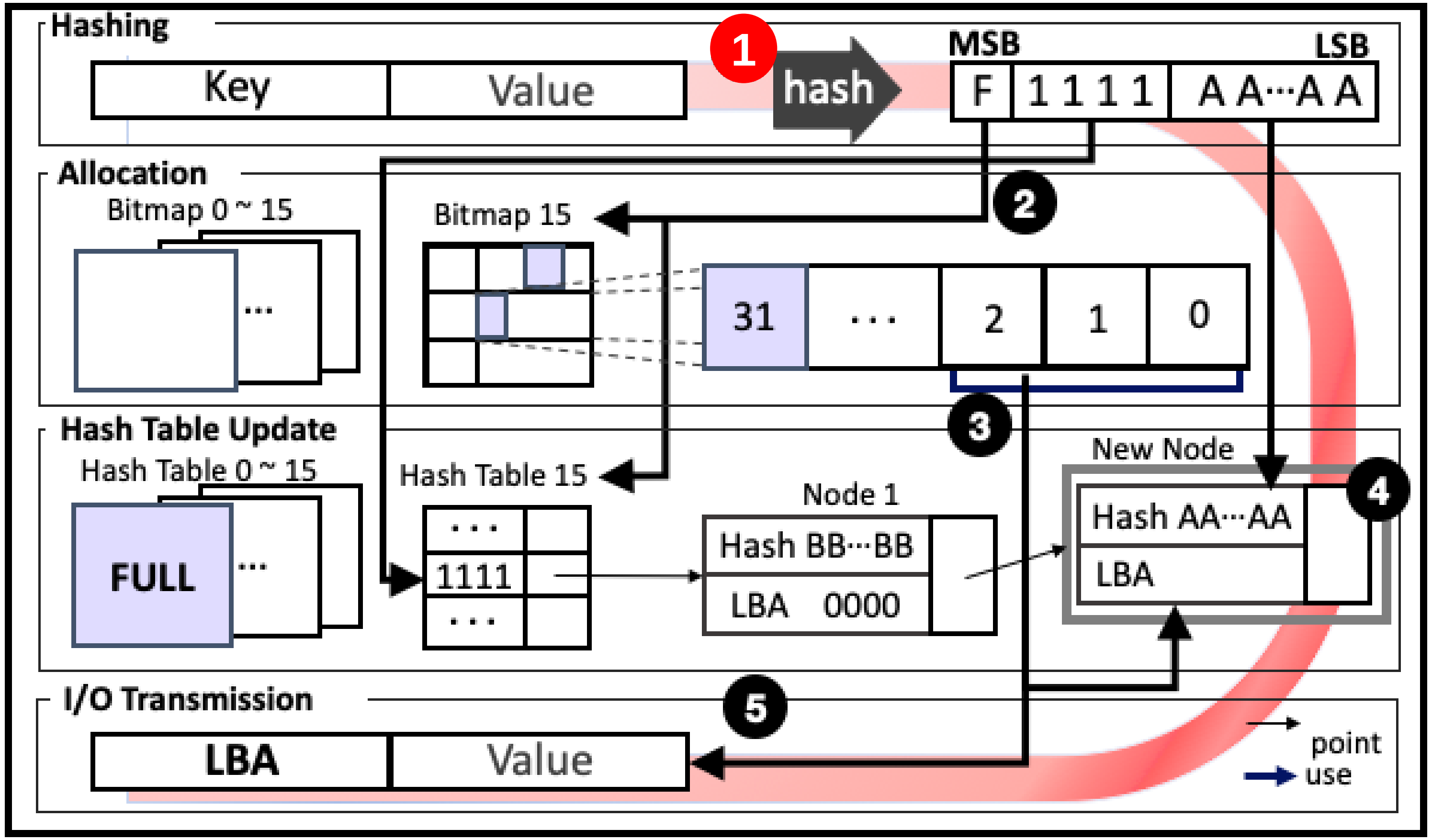


Module#1: Storage Engine

Bitmap Array

Hash Table

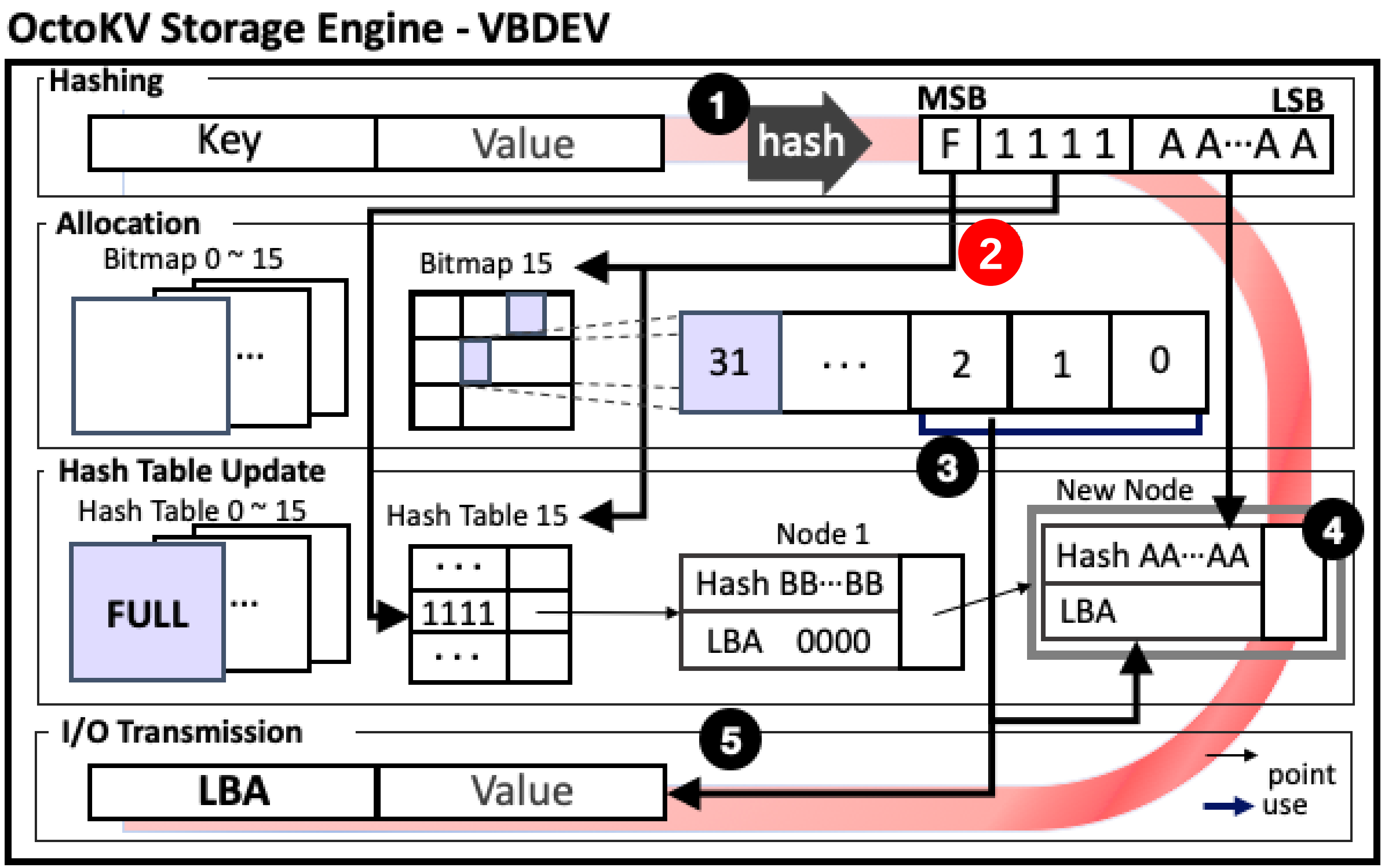
OctoKV Storage Engine - VBDEV



Module#1: Storage Engine

Bitmap Array

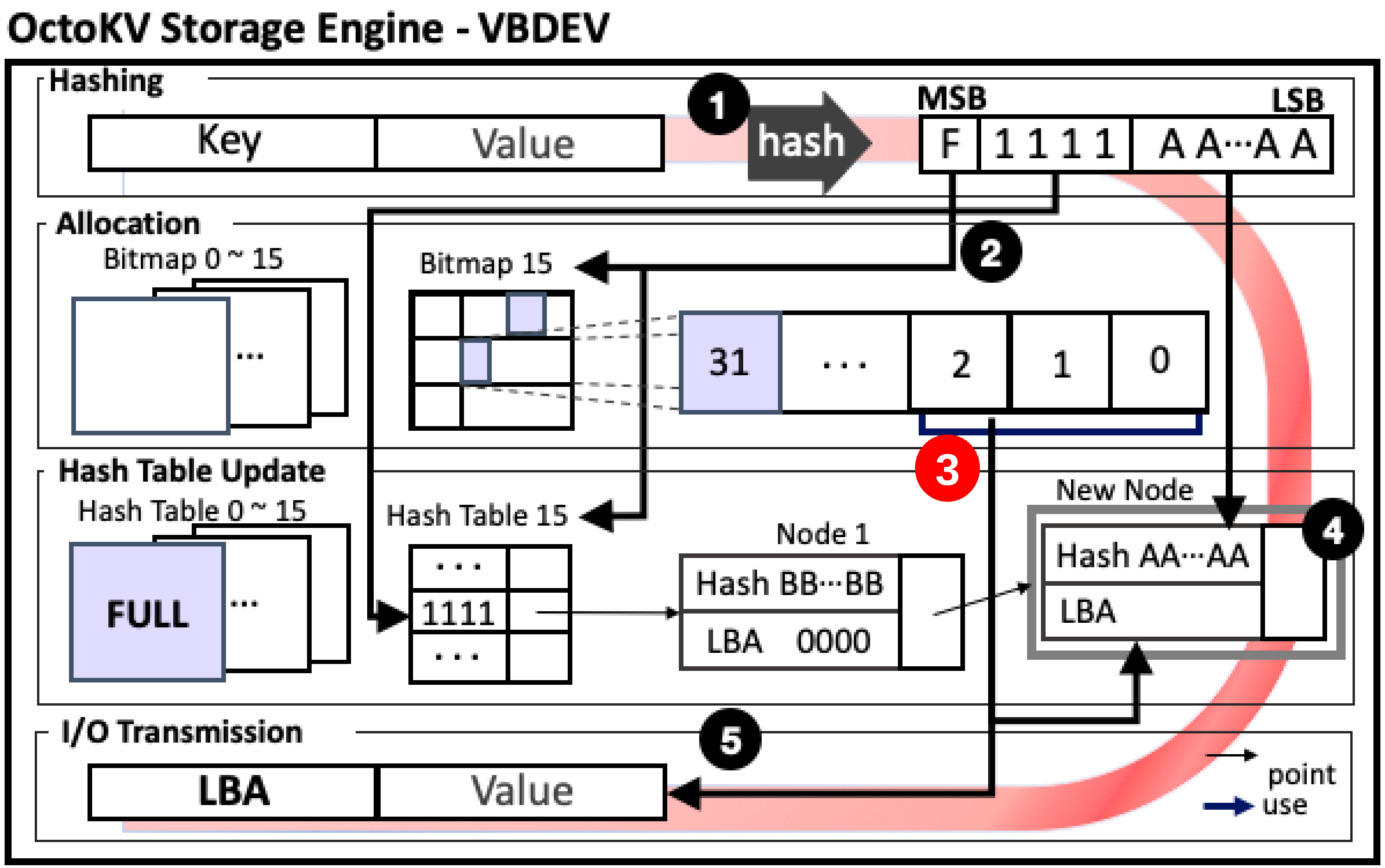
Hash Table



Module#1: Storage Engine

Bitmap Array

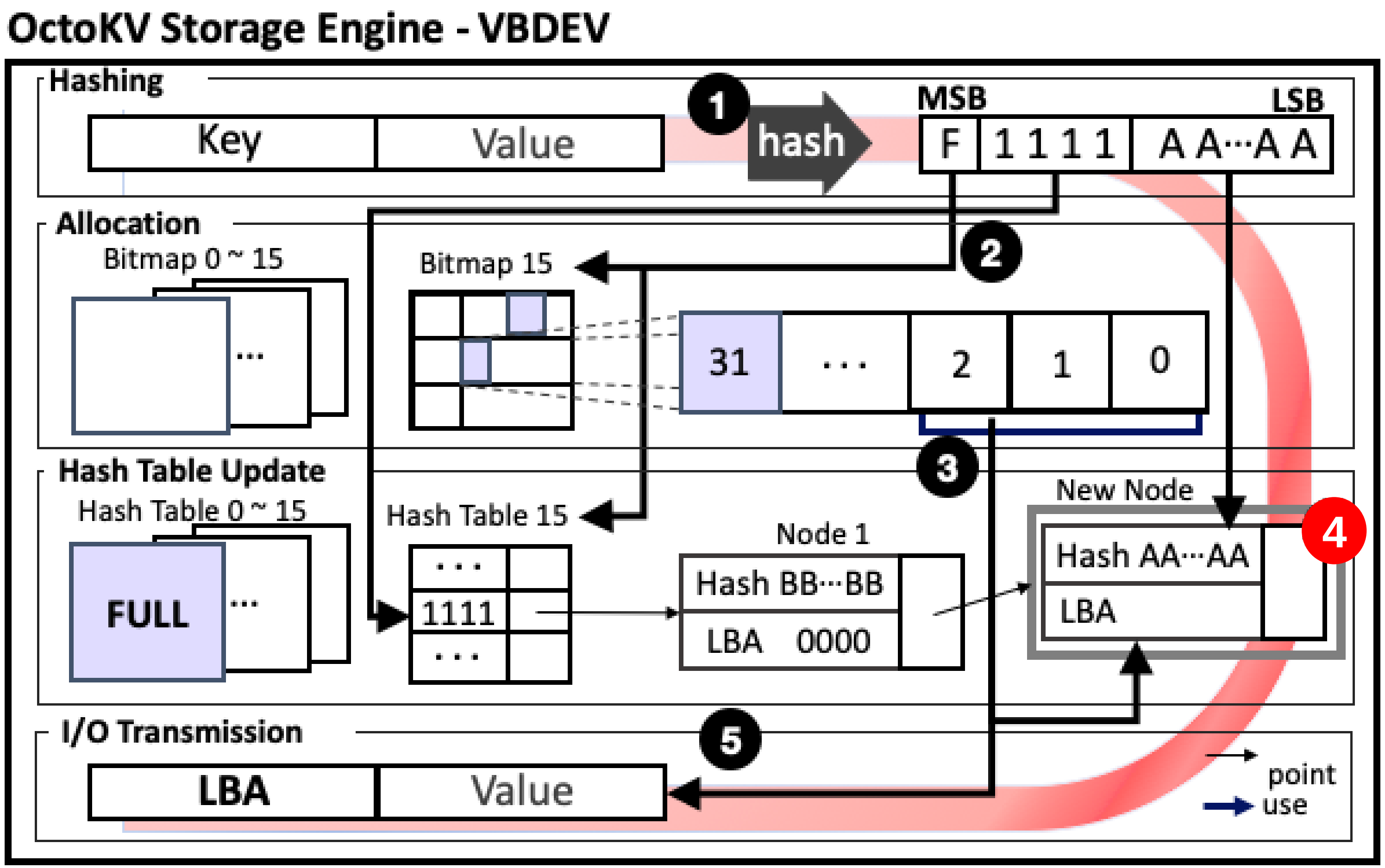
Hash Table



Module#1: Storage Engine

Bitmap Array

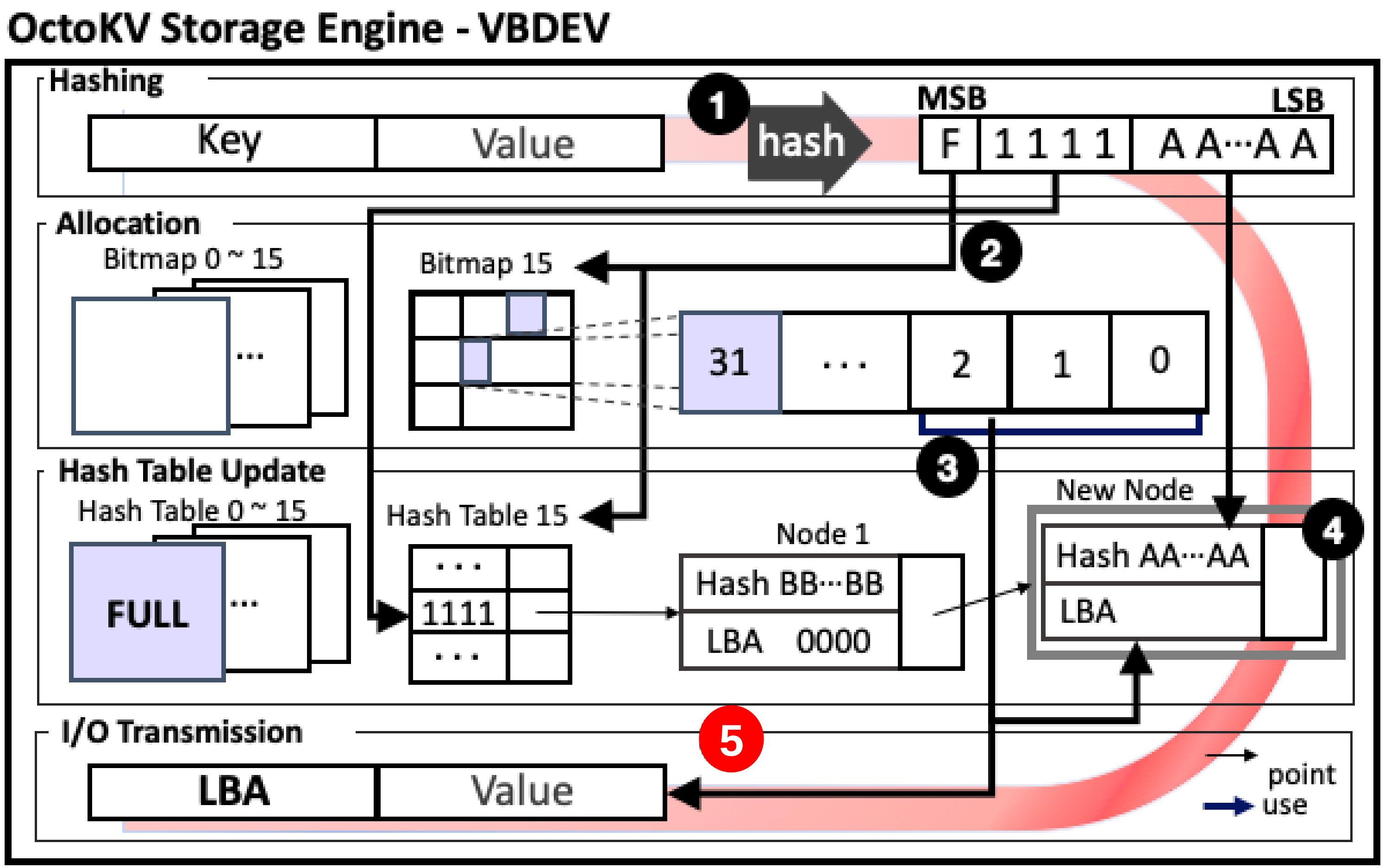
Hash Table



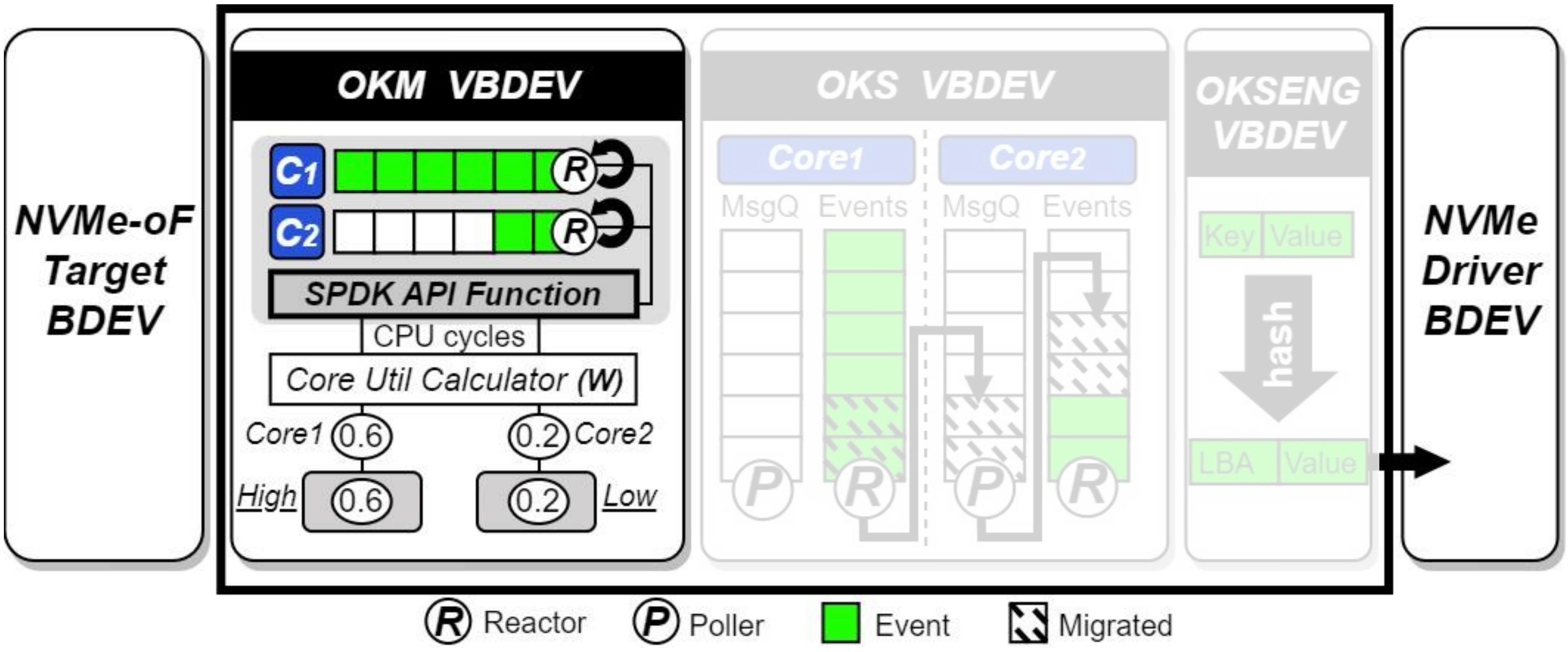
Module#1: Storage Engine

Bitmap Array

Hash Table

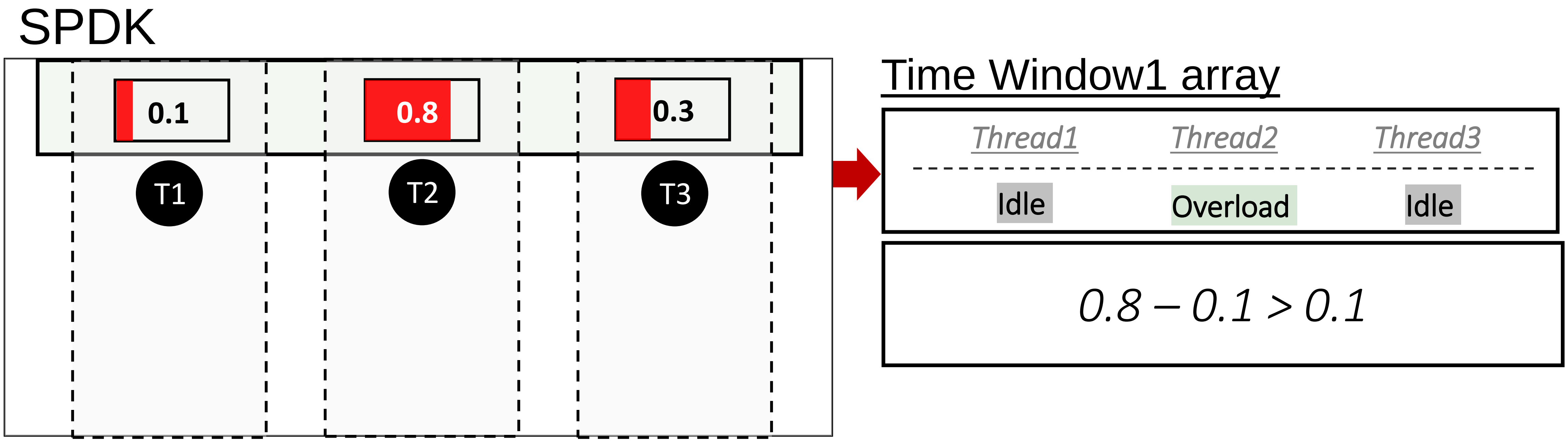


Module#2: Monitoring



Module#2: Monitoring

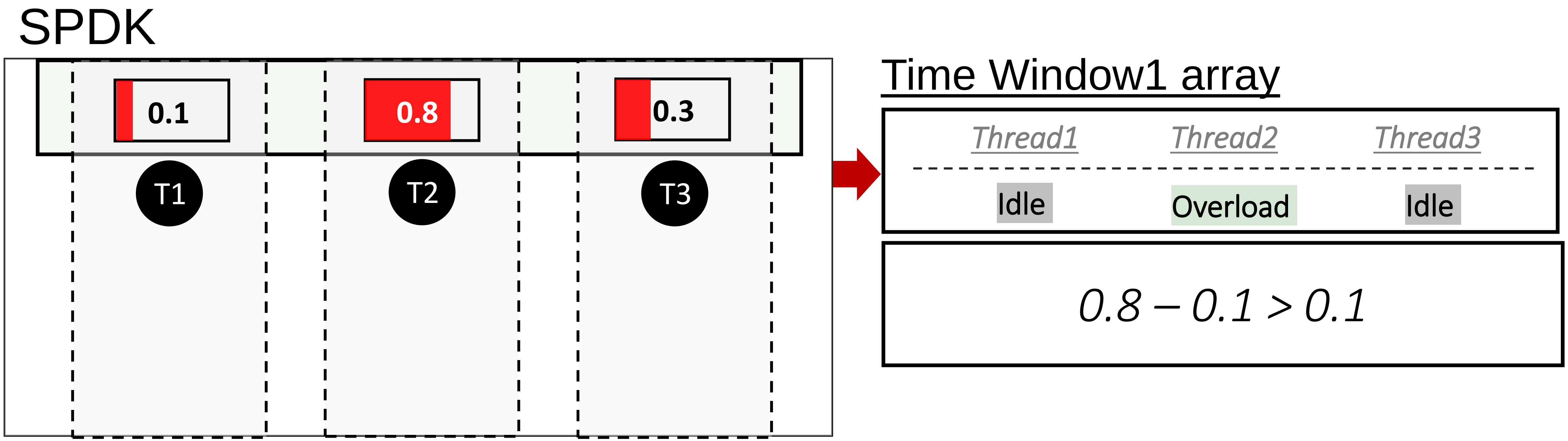
- Monitors the utilization of each core



Module#2: Monitoring

- Monitors the utilization of each core

  Condition#1: Core Overloading $F_{cutil}(C) > T_{OL}\ (T_{OL} = 0.4)$

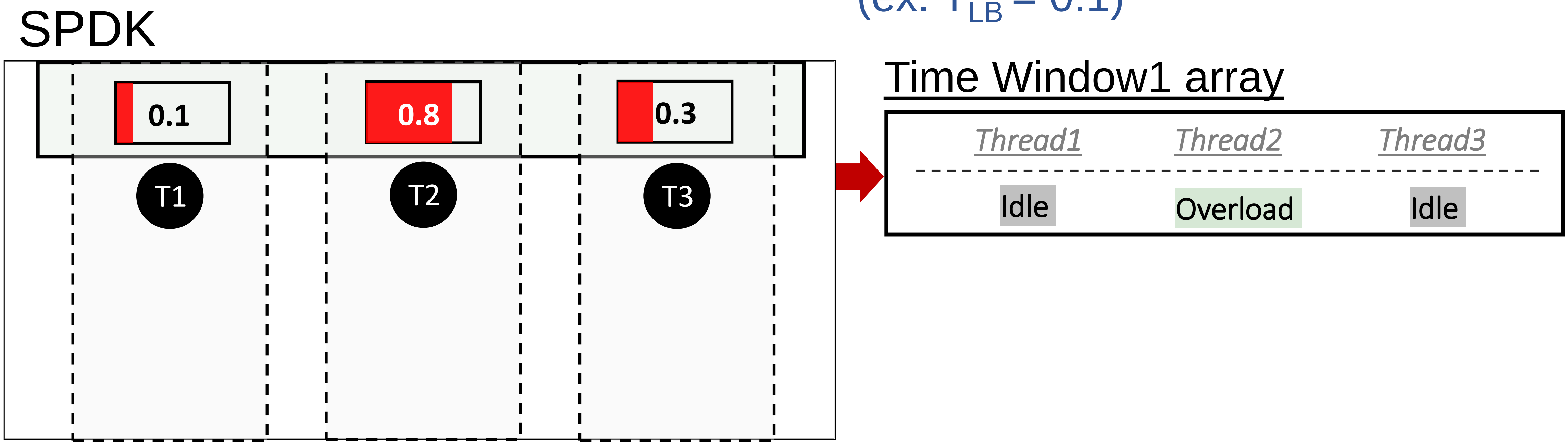


Module#2: Monitoring

- Monitors the utilization of each core

  Condition#1: Core Overloading — $F_{cutil}(C) > T_{OL}$ ($T_{OL} = 0.4$)

  Condition#2: Load Imbalance — $Max\{F_{cutil}(C)\} - Min\{F_{cutil}(C)\} > T_{LB}$ (ex. $T_{LB} = 0.1$)

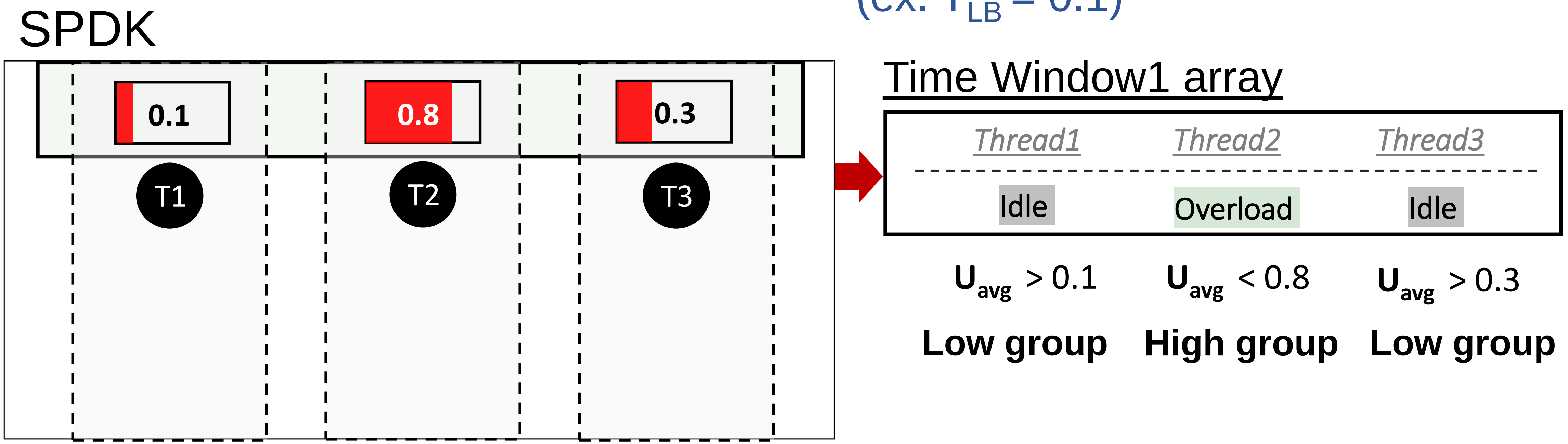


Module#2: Monitoring

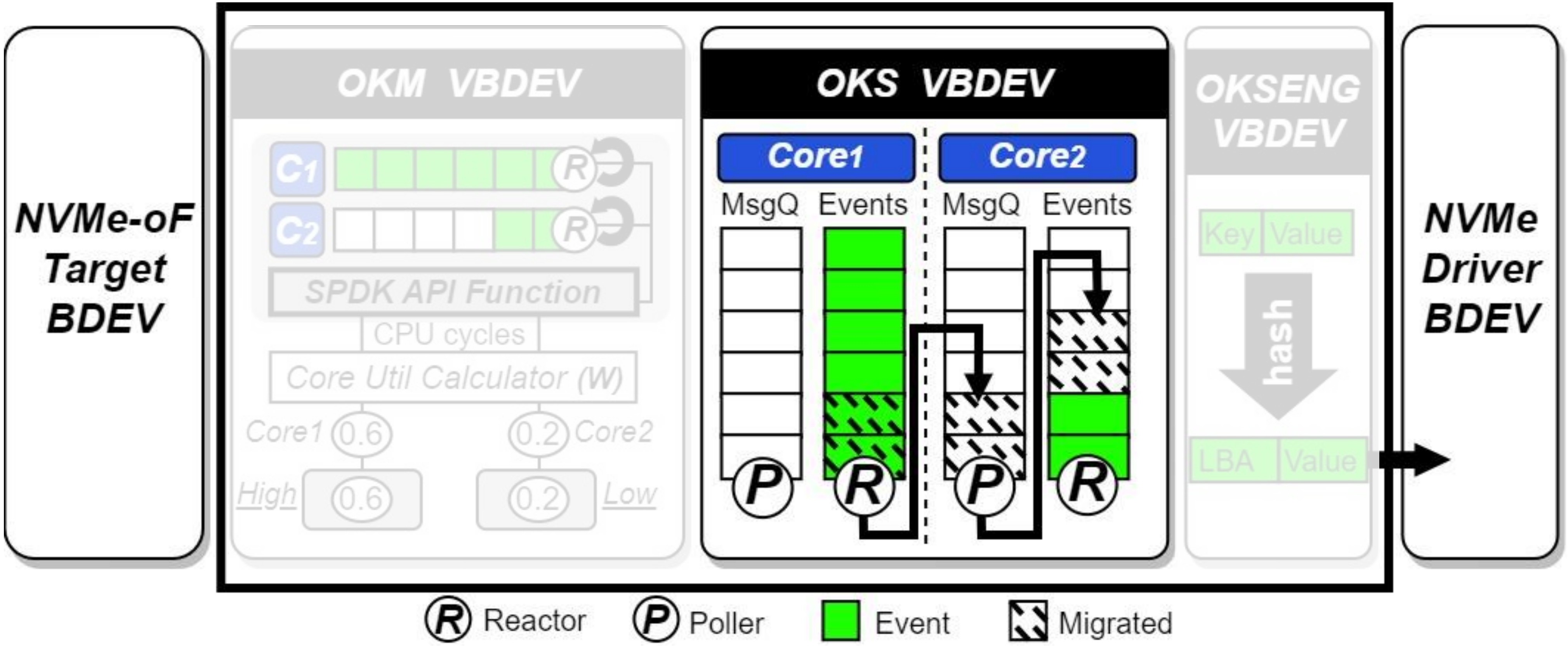
- Monitors the utilization of each core

  Condition#1: Core Overloading — $F_{cutil}(C) > T_{OL}$ ($T_{OL} = 0.4$)

  Condition#2: Load Imbalance — $\mathrm{Max}\{F_{cutil}(C)\} - \mathrm{Min}\{F_{cutil}(C)\} > T_{LB}$ (ex. $T_{LB} = 0.1$)



Module#3: Scheduling



Module#3: Scheduling

- OctoKV Scheduling Module migrates I/O requests from overloaded cores to idle cores
- A single I/O request consists of three stages

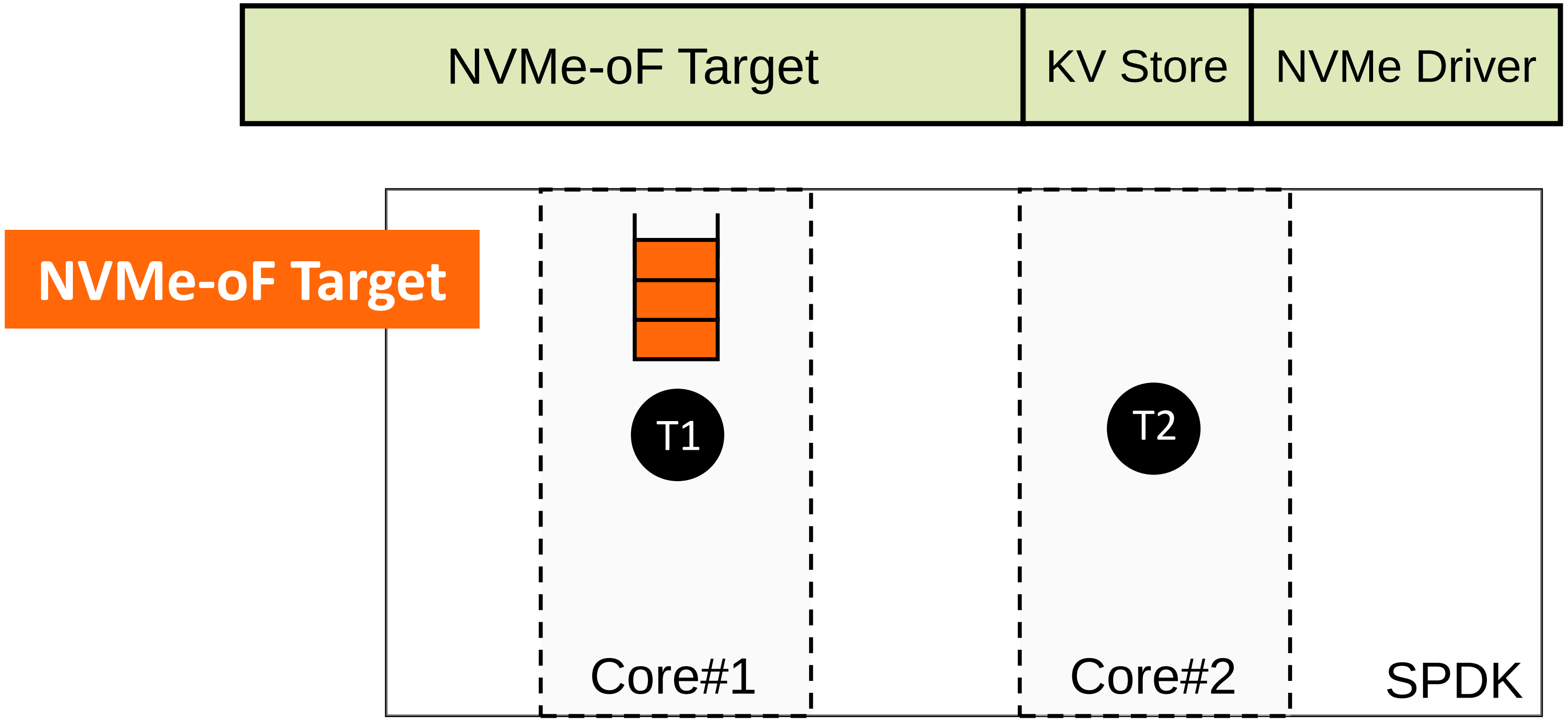
Module#3: Scheduling

- OctoKV Scheduling Module migrates I/O requests from overloaded cores to idle cores
- A single I/O request consists of three stages

NVMe-oF Target	KV Store	NVMe Driver

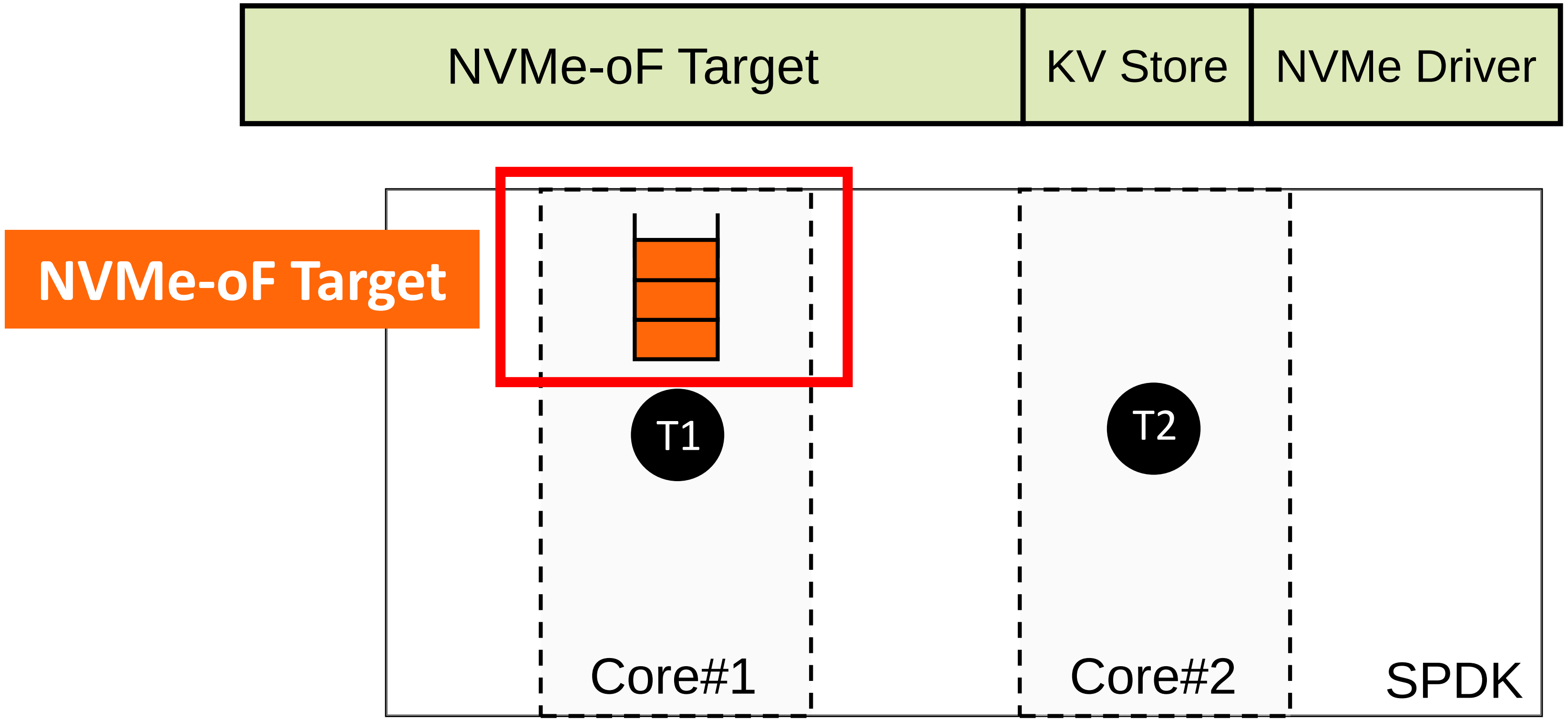
Module#3: Scheduling

- OctoKV Scheduling Module migrates I/O requests from overloaded cores to idle cores
- A single I/O request consists of three stages



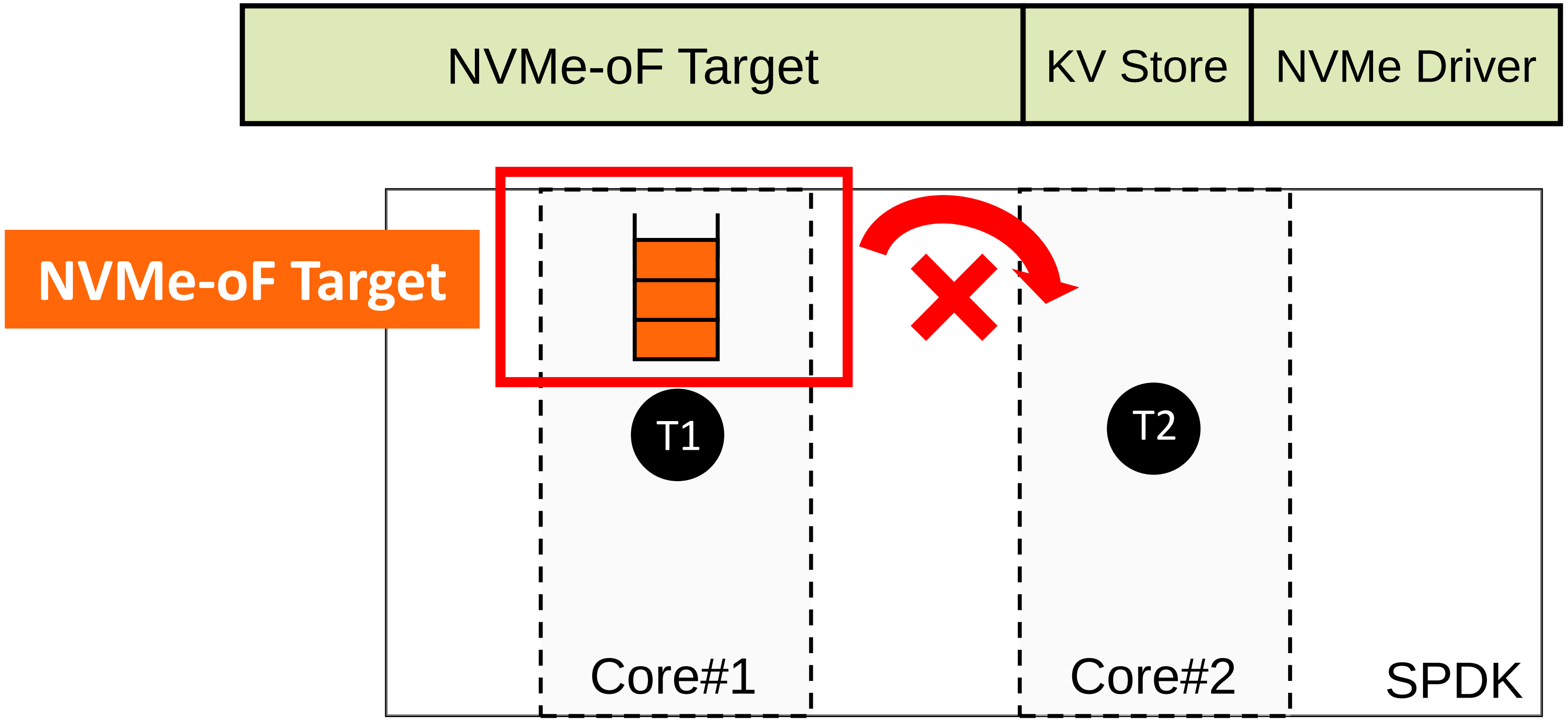
Module#3: Scheduling

- OctoKV Scheduling Module migrates I/O requests from overloaded cores to idle cores
- A single I/O request consists of three stages



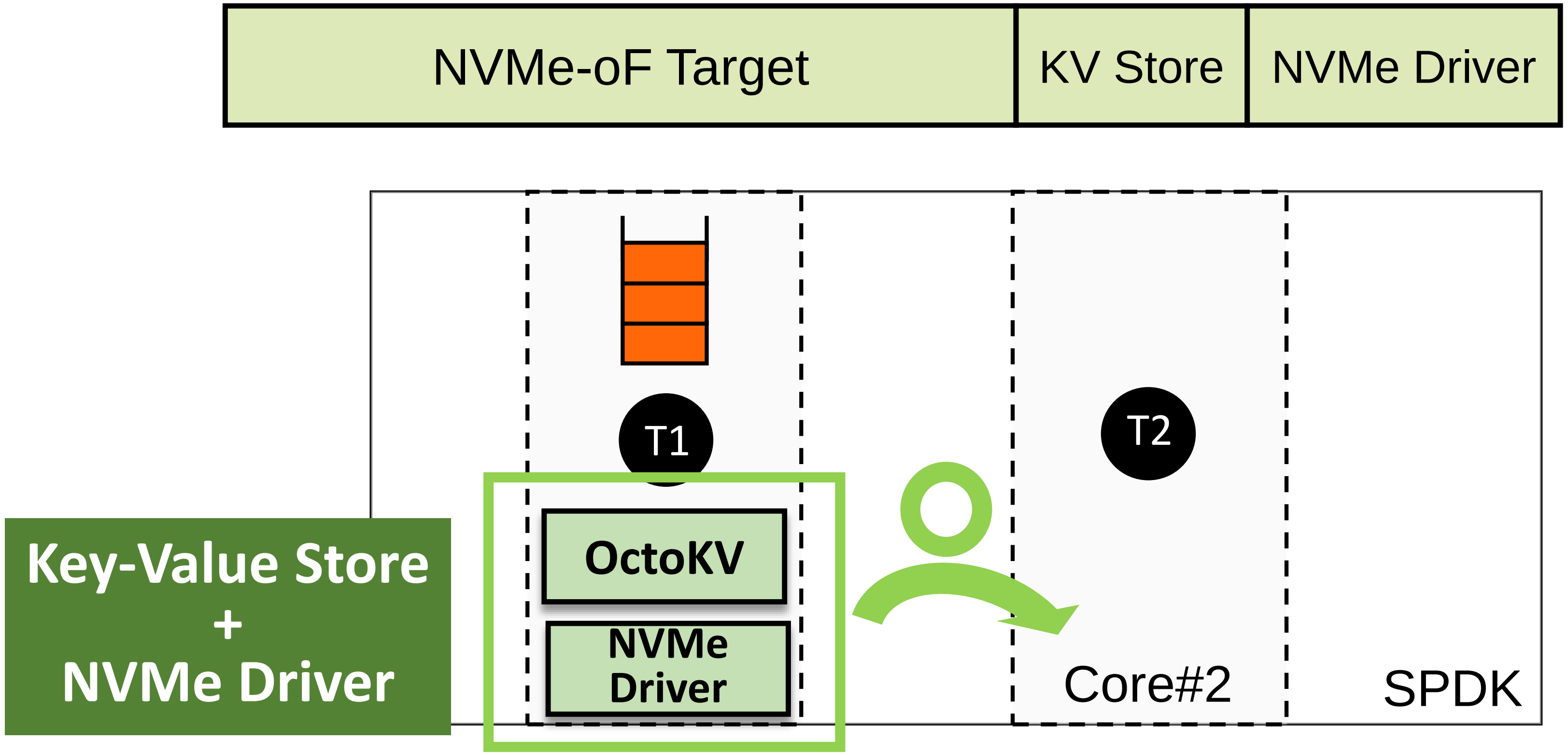
Module#3: Scheduling

- OctoKV Scheduling Module migrates I/O requests from overloaded cores to idle cores
- A single I/O request consists of three stages

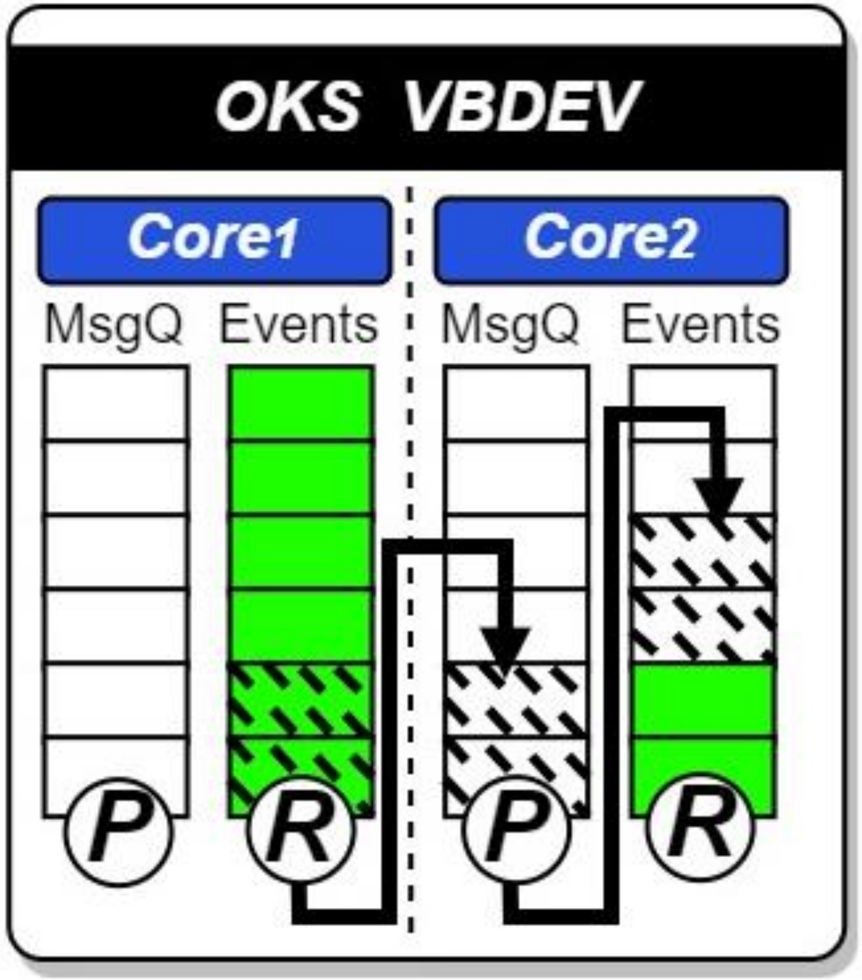


Module#3: Scheduling

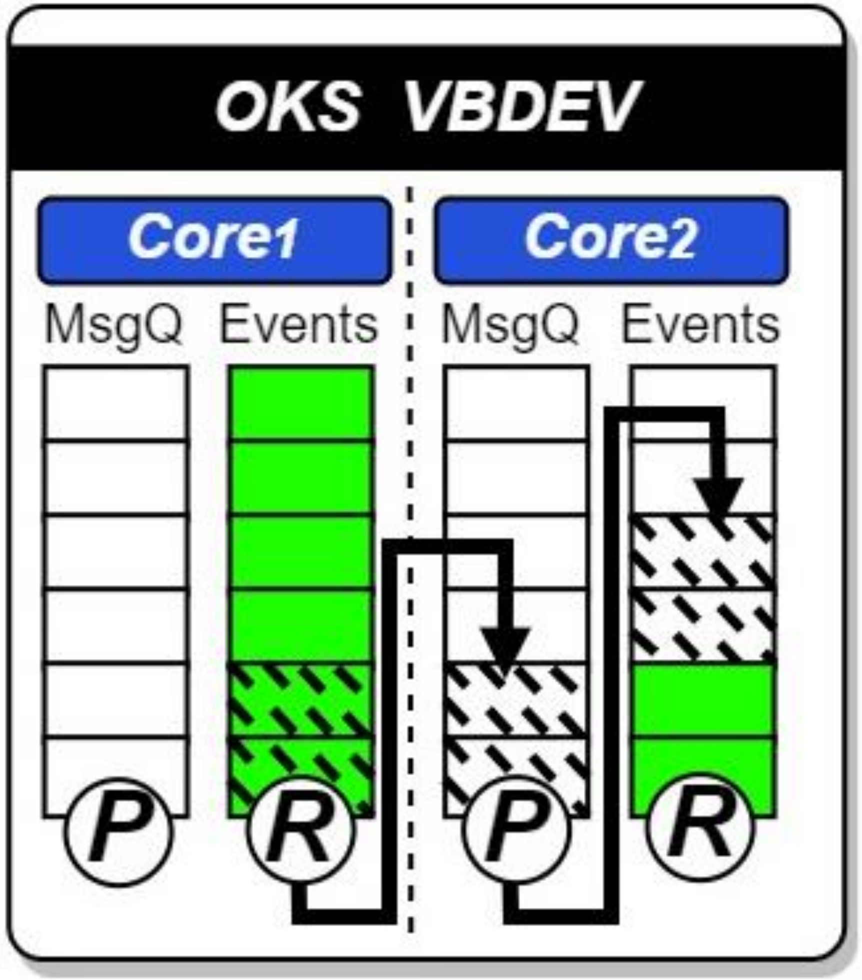
- OctoKV Scheduling Module migrates I/O requests from overloaded cores to idle cores
- A single I/O request consists of three stages

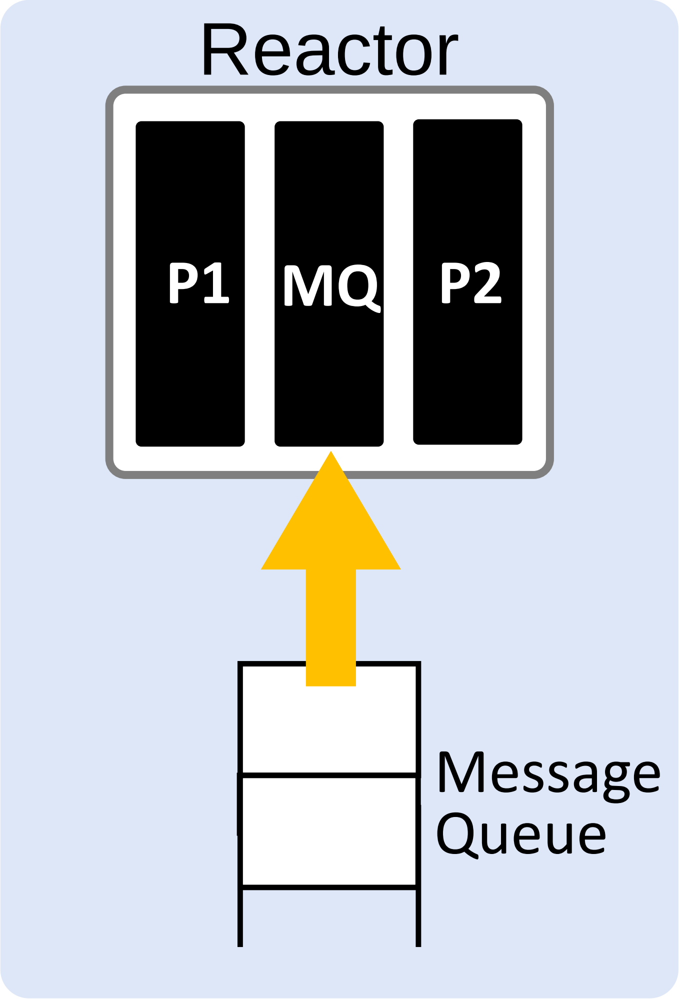


Module#3: Scheduling



Module#3: Scheduling





Module#3: Scheduling

NVMe-oF Target	KV Store	NVMe Driver

80% of Core Utilization (NVMe-oF Target)

20% of Core Utilization (KV Store, NVMe Driver)

High Group

Thread2 : 0.8

Low Group

Thread1 : 0.1

Thread3 : 0.3

➡ Movable core utilization
20% of 0.8 => 0.16

➡ Acceptable core utilization
$(U_{avg} - 0.1) + (U_{avg} - 0.3) => 0.4$

Time Window1 array

Thread1	*Thread2*	*Thread3*
Low Group 0.1	High Group 0.8	Low Group 0.3

All KV stores and NVMe Driver stages in the high group core can be moved to the low group core for processing.

Module#3: Scheduling

- Two heuristic algorithms determine
  how much I/O to migrate to each core of low group

(Ex) *Thread1: 0.1* *Thread3: 0.3* U_{avg} *: 0.4*

Module#3: Scheduling

- Two heuristic algorithms determine
  how much I/O to migrate to each core of low group

(Ex) Thread1: 0.1 Thread3: 0.3 U_{avg} *: 0.4*

RoundRobin (RR)

Thread1 : Thread3 = 1 : 1

Module#3: Scheduling

- Two heuristic algorithms determine
  how much I/O to migrate to each core of low group

(Ex) Thread1: 0.1 Thread3: 0.3 U_{avg} : 0.4

RoundRobin (RR)

<u>*Thread1 : Thread3 = 1 : 1*</u>

Proportional Share (PS)

Thread1 : U_{avg} - 0.1 = 0.3

Thread1 : U_{avg} - 0.3 = 0.1

<u>*Thread1 : Thread3 = 3 : 1*</u>

Content

- Background
- Problem Definition
- Motivational Experiments
- OctoKV: Design and Implementation
- Evaluation
- Conclusion

Evaluation

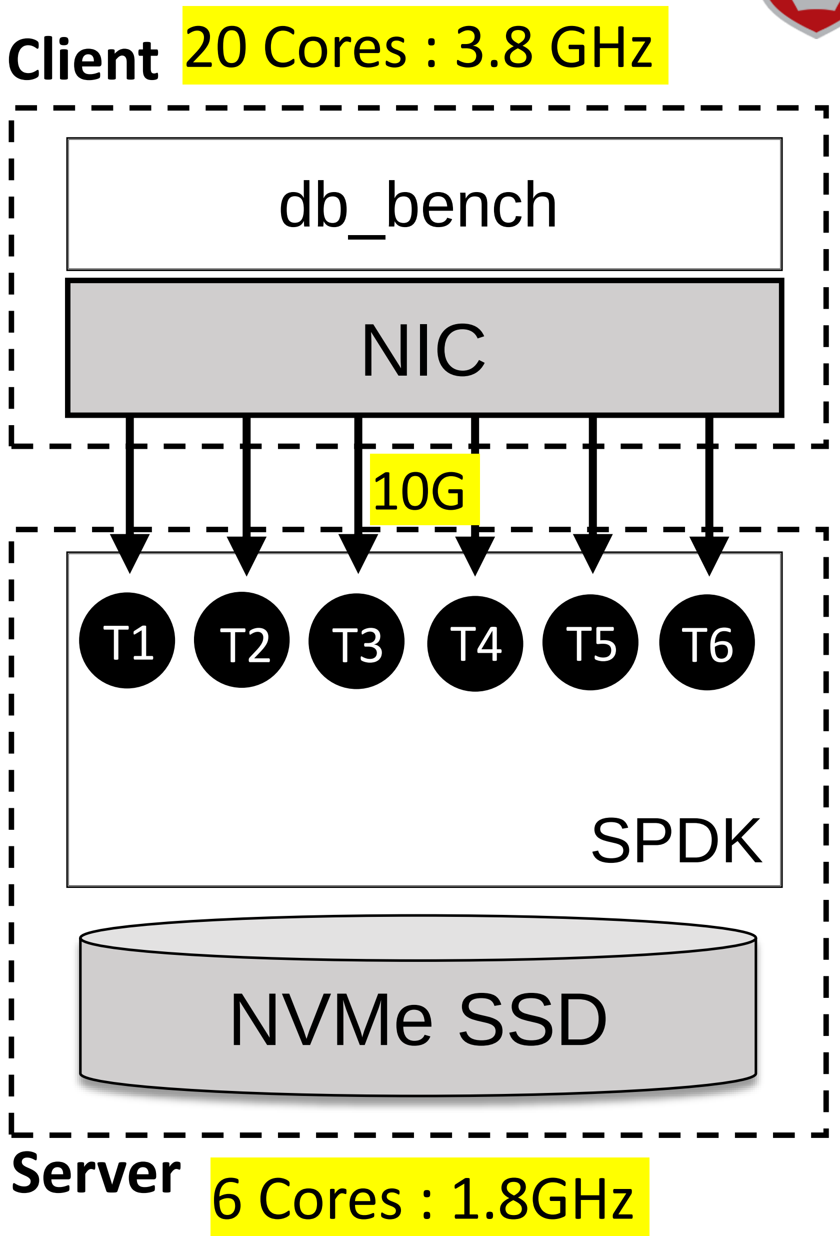
- Client
  - § Running a db_bench benchmark
    1) Light workload

       7 I/O threads issue Put/Get I/Os
    2) Medium workload

       10 I/O threads issue Put/Get I/Os
    3) Heavy workload

       12 I/O threads issue Put/Get I/Os
  - § I/O request size = 16KB

- Server
  - § 6-core device
  - § Running a Linux OS using Intel SPDK



Evaluation
Configurations

Evaluation
Configurations

(1) **Host KVS**

→ A hash-based key-value storage engine running on the client, layered atop the kernel and file systems

Evaluation
Configurations

(1) **Host KVS**

→ A hash-based key-value storage engine running on the client, layered atop the kernel and file systems

(2) **OctoKV**

→ The proposed system with only the key-value storage engine running on the server

Evaluation
Configurations

(1) **Host KVS**

→ A hash-based key-value storage engine running on the client, layered atop the kernel and file systems

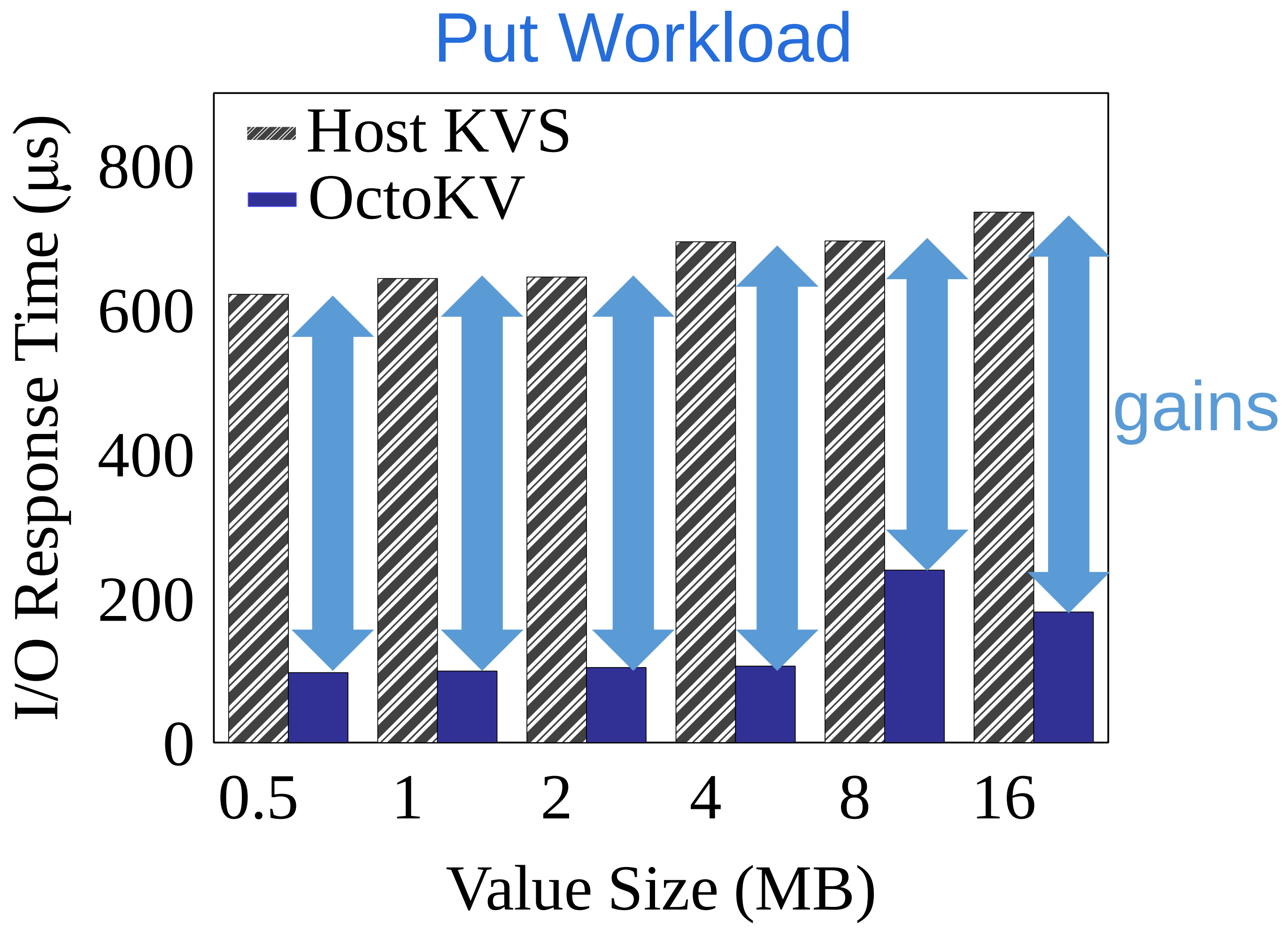
(2) **OctoKV**

→ The proposed system with only the key-value storage engine running on the server

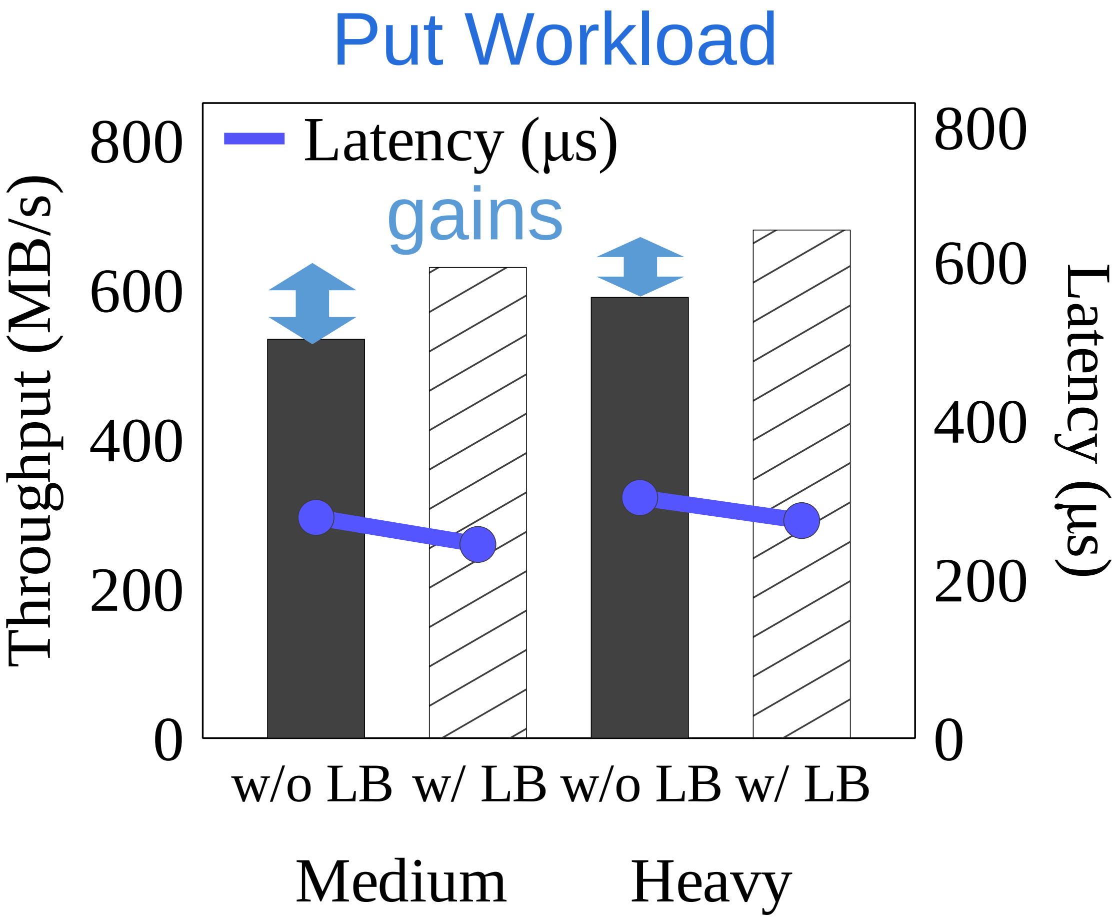
(3) **OctoKV-LB**

→ OctoKV with the load-aware balanced I/O scheduling

Evaluation
I/O Response Time

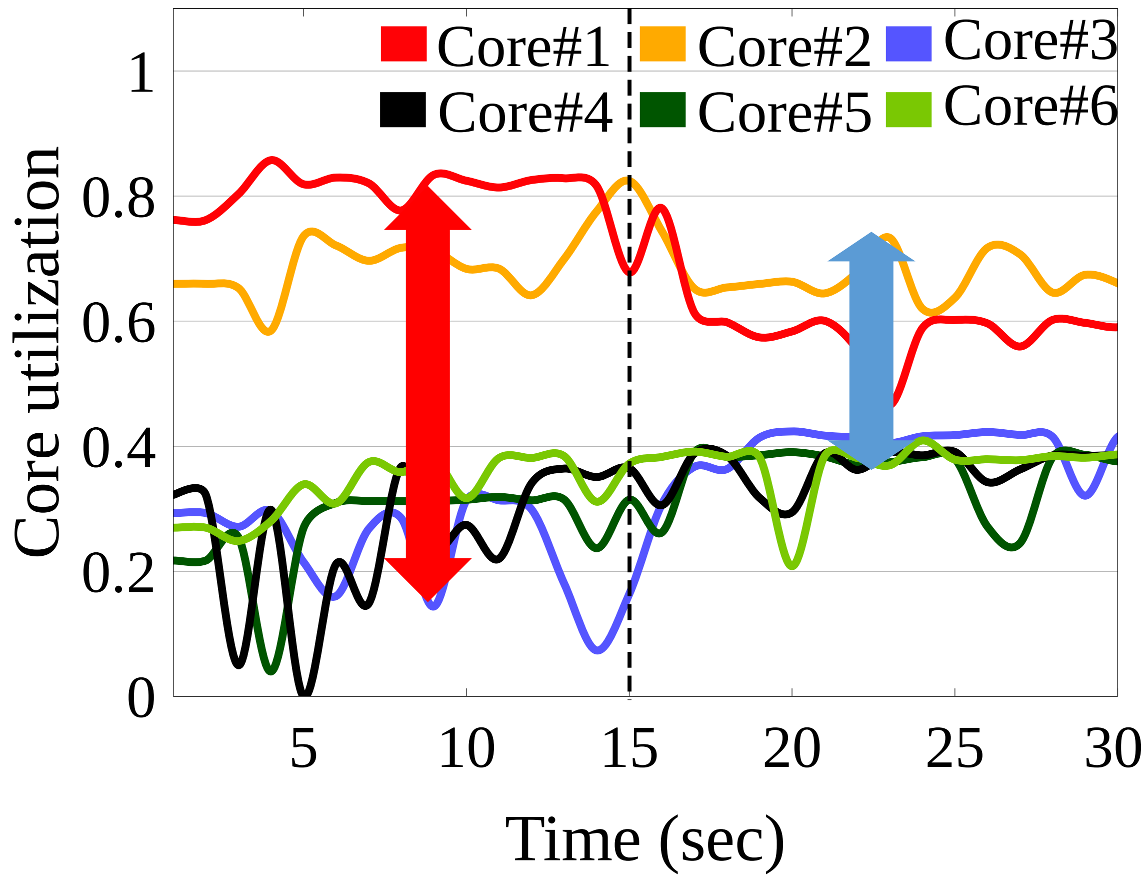


Evaluation
Throughput

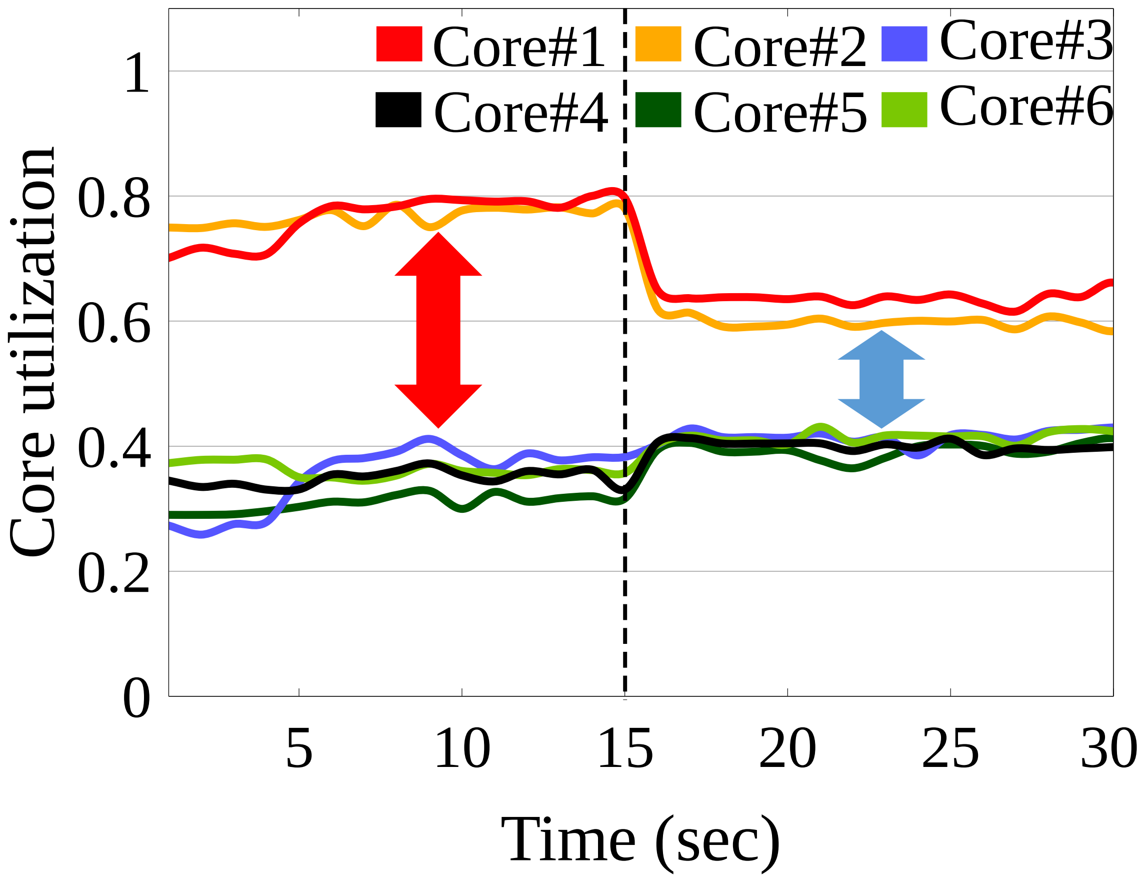


Evaluation
Core Utilization

Medium Workload



Heavy Workload



Content

Conclusion

- OctoKV is a server-side key-value store that leverages the SPDK capabilities for high-performance in disaggregated storage
- OctoKV achieves lower I/O response times in comparison to traditional approaches
- OctoKV has proposed a powerful load-aware balanced I/O scheduling

Thank You ☺

junttang@sogang.ac.kr ***Junhyeok Park***