DISCOS
Data-Intensive Computing & Systems Laboratory

IHS

Key-Value Computational Solid-State Drive (KV-CSD)

Junhyeok Park

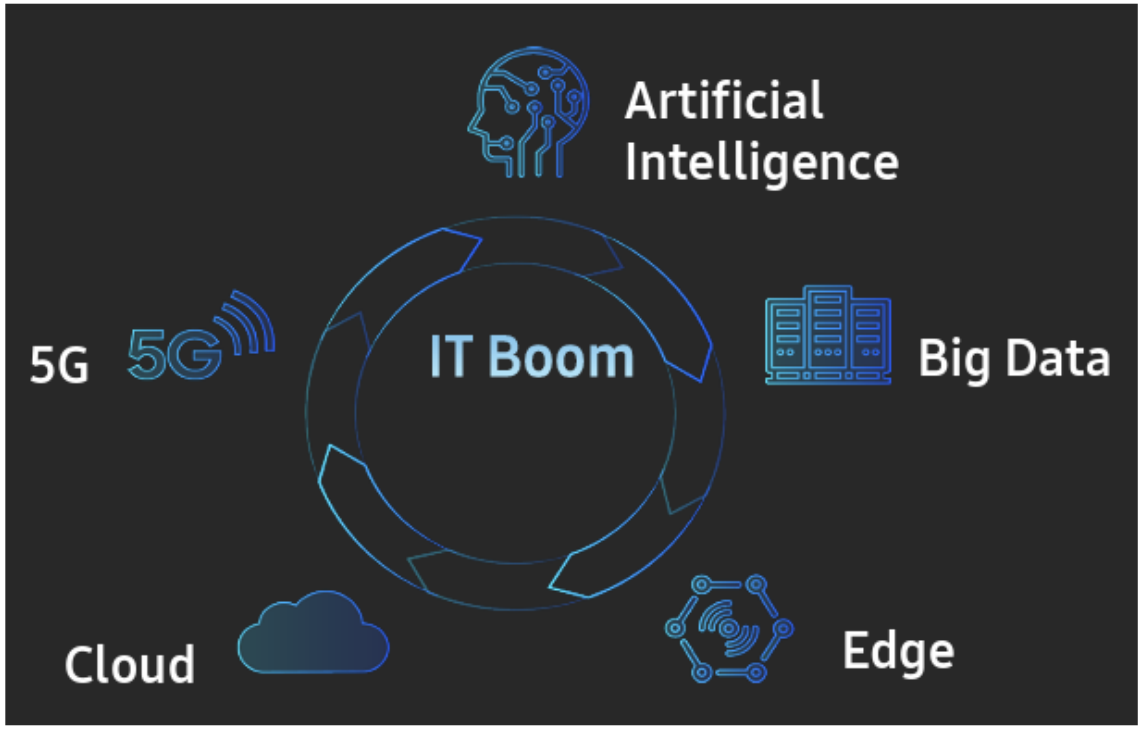
Advised by Prof. Youngjae Kim

Big Data Computing Era

- A rapid adoption of artificial intelligence (e.g., LLM) and cloud computing in modern data centers
  - These applications handle **Big Data**

ChatGPT

Microsoft Azure

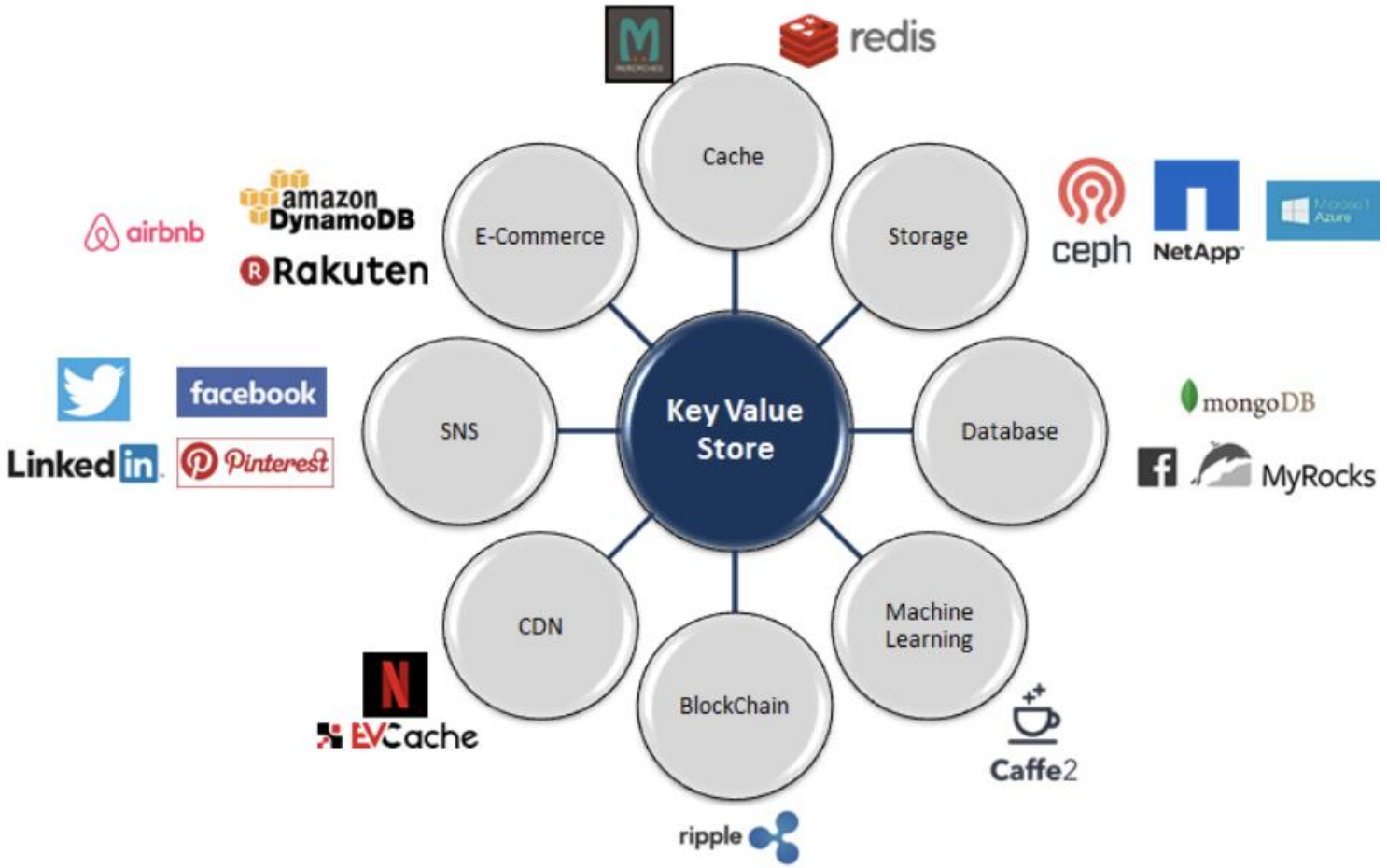


What does Data look like?

- These Big Data applications do not merely handle Blocks; they manage variable-sized **Key-Value Pairs (Objects)**

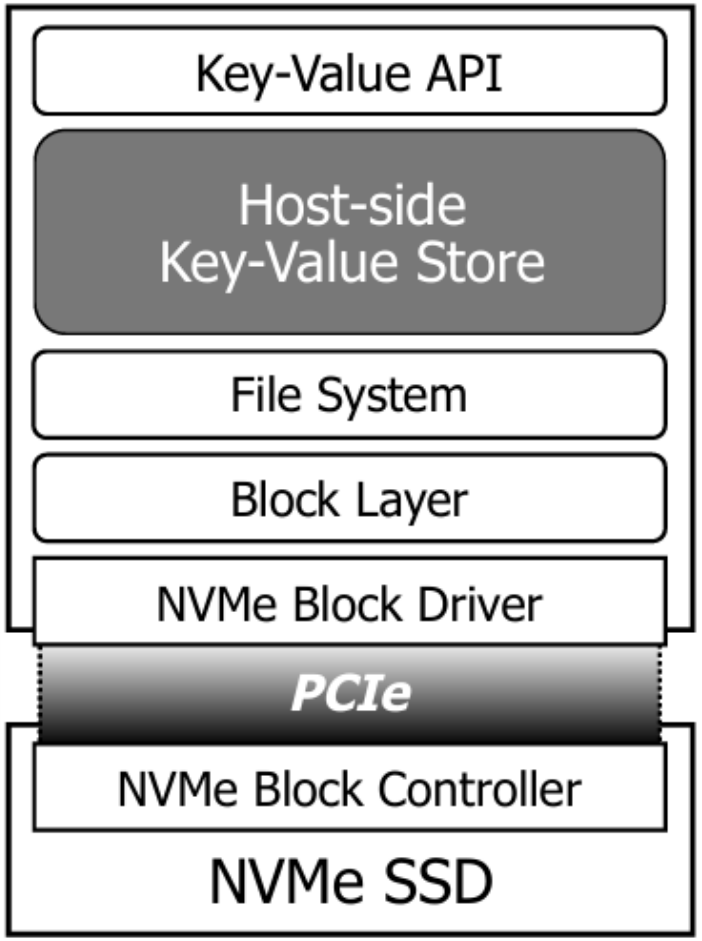
What does Data look like?

- Therefore, these Big Data applications typically operate by employing Key-Value Stores (e.g., RocksDB, Cassandra)



Software Stack Issue

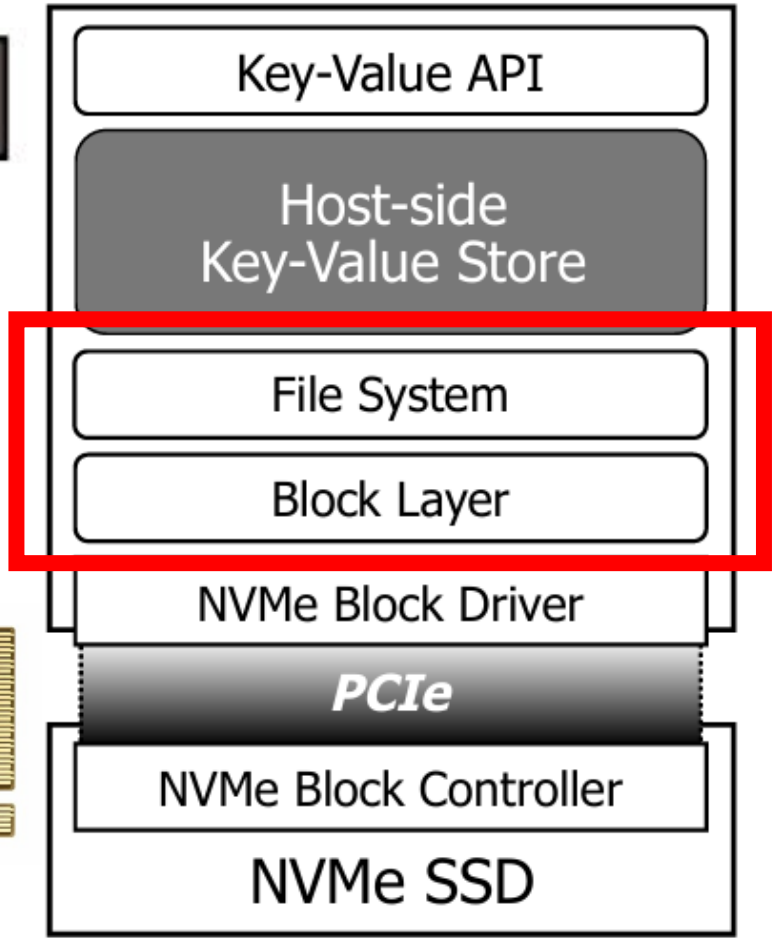
- Key-Value Stores run on top of File System, Block Layer, Block Device Driver and Block Device Controller



Software Stack Issue

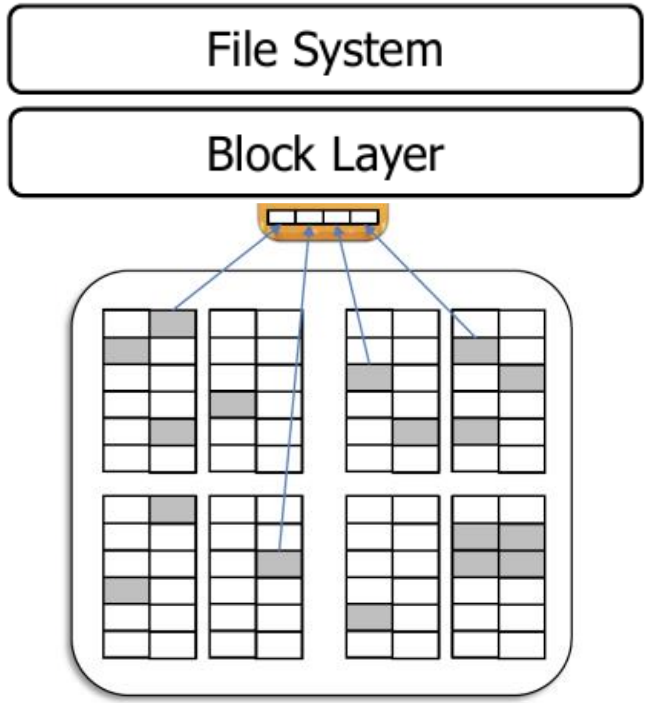
- Key-Value Stores run on top of File System, Block Layer, Block Device Driver and Block Device Controller

Do we really need these layers?



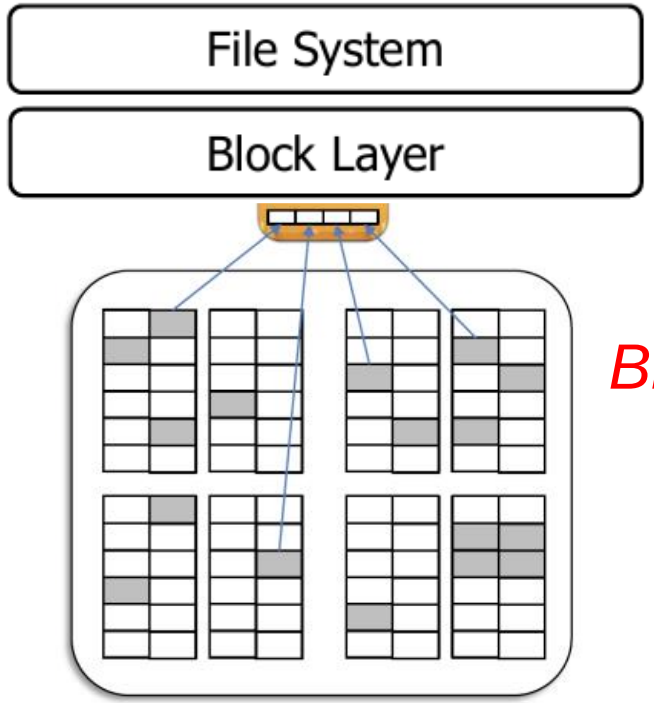
Software Stack Issue

- These layers are in place to follow the **block interface**, which originated from the old HDDs

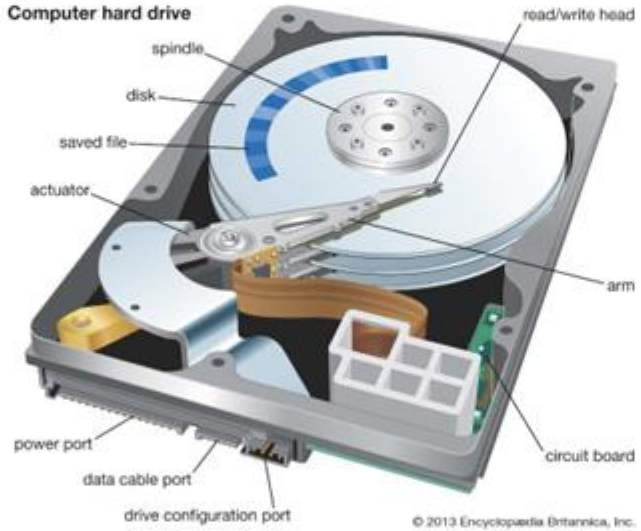


Software Stack Issue

- These layers are in place to follow the **block interface**, which originated from the old HDDs

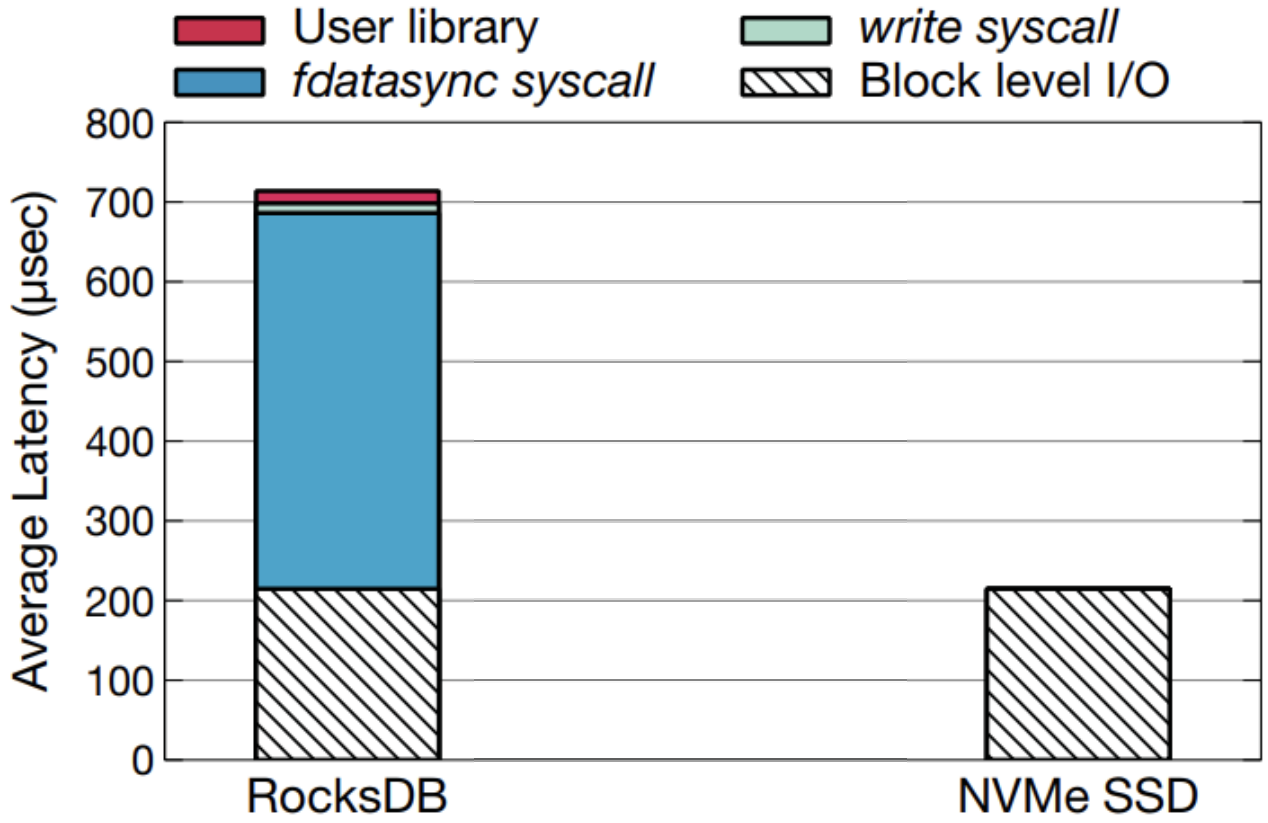


Block Interface from legacy storage



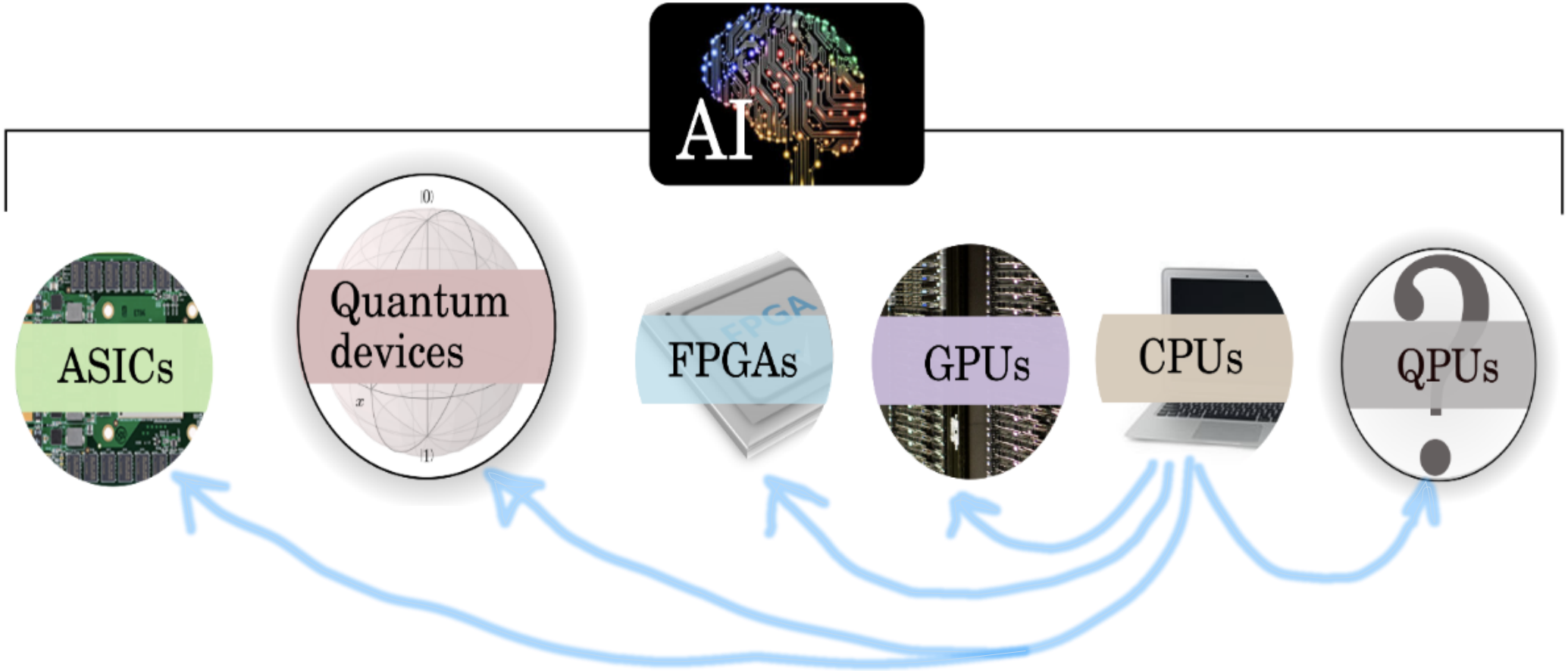
Software Stack Issue

- The problem is that these layers account for a **significant portion** of the total response time in RocksDB



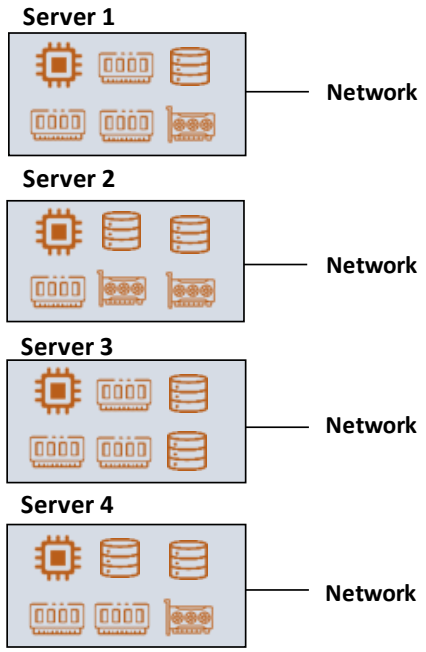
Software Stack Issue

- Utilizing the host node's resources (for File System and Block Layer) solely to comply with block semantics contradicts the heterogeneous computing paradigm

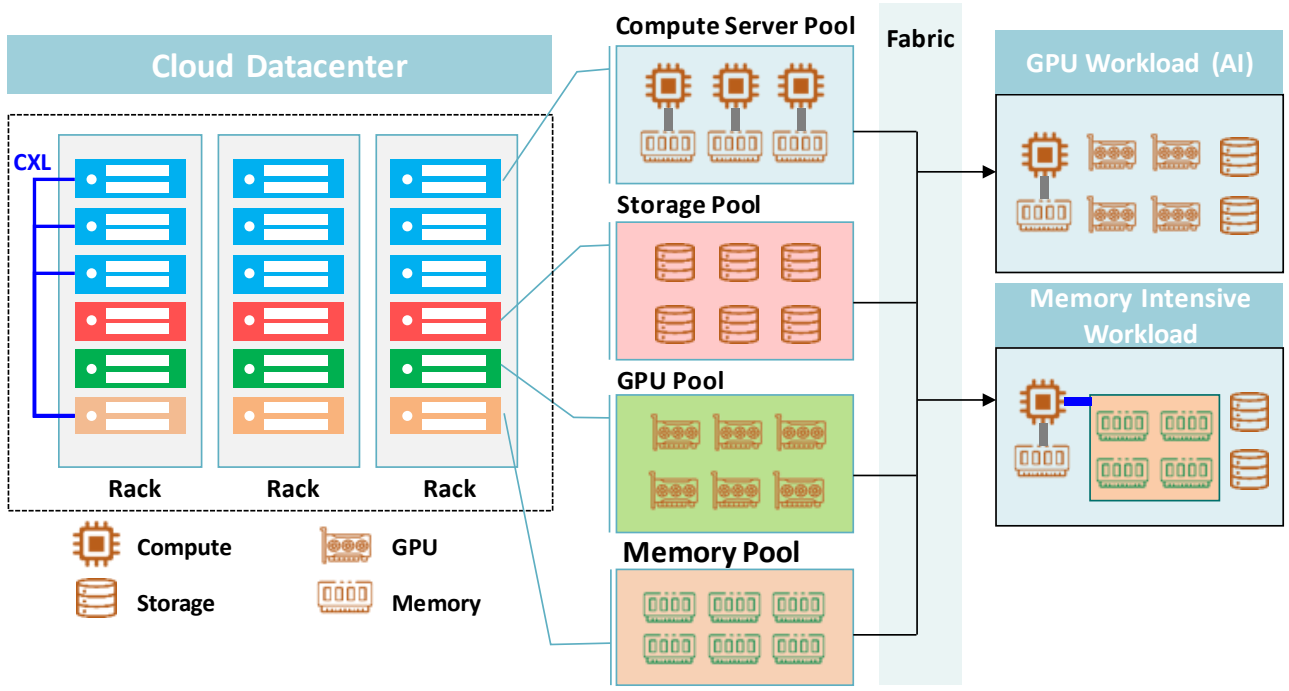


Software Stack Issue

- Heavy Software Stack impedes the optimal disaggregation and scaling out of compute and memory resources in data centers



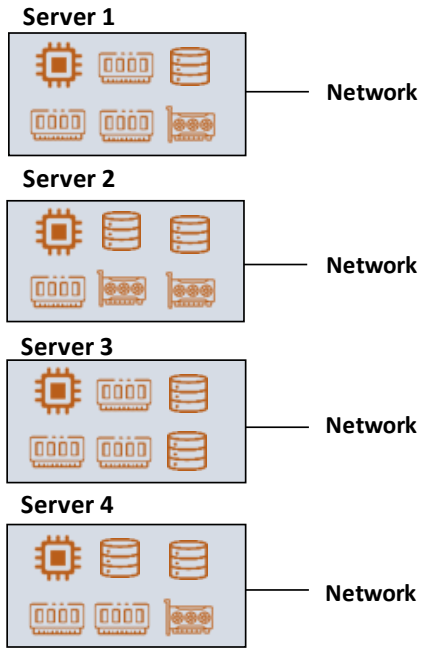
[Fixed Configuration Infrastructure]



[Composable Disaggregated Infrastructure]

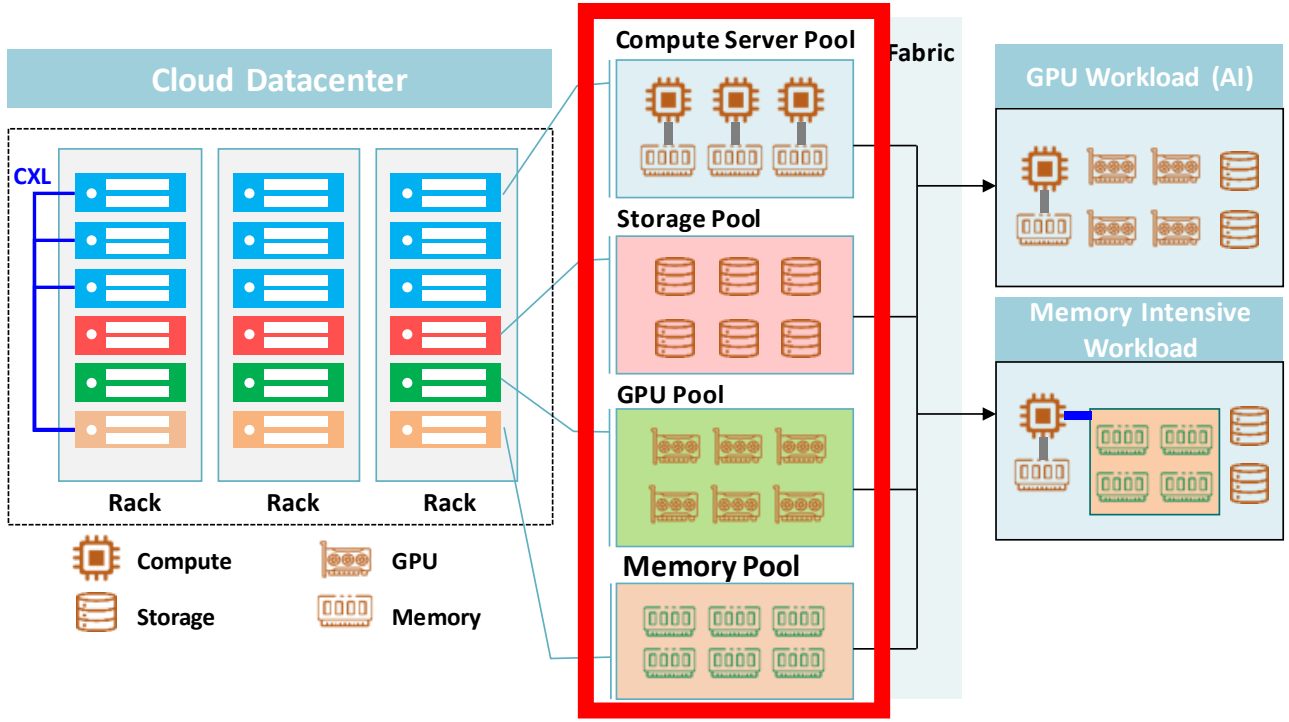
Software Stack Issue

- Heavy Software Stack impedes the optimal disaggregation and scaling out of compute and memory resources in data centers



[Fixed Configuration Infrastructure]

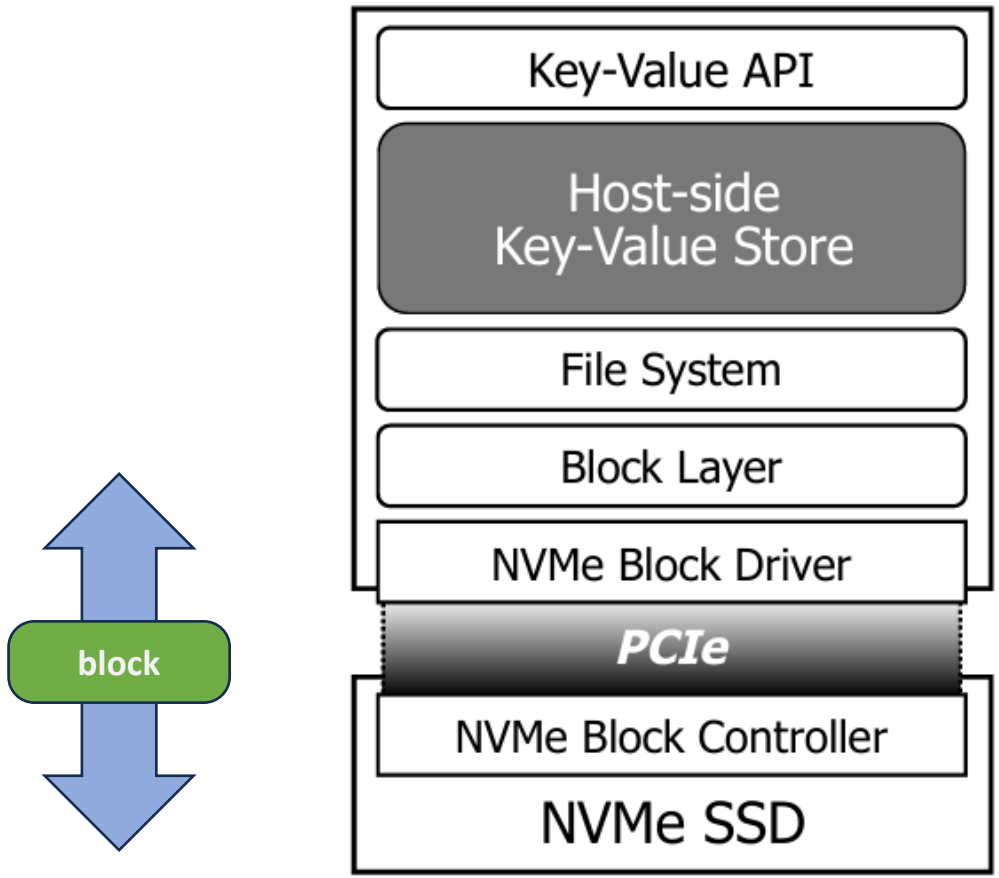
High loads
Hard to scale out



[Composable Disaggregated Infrastructure]

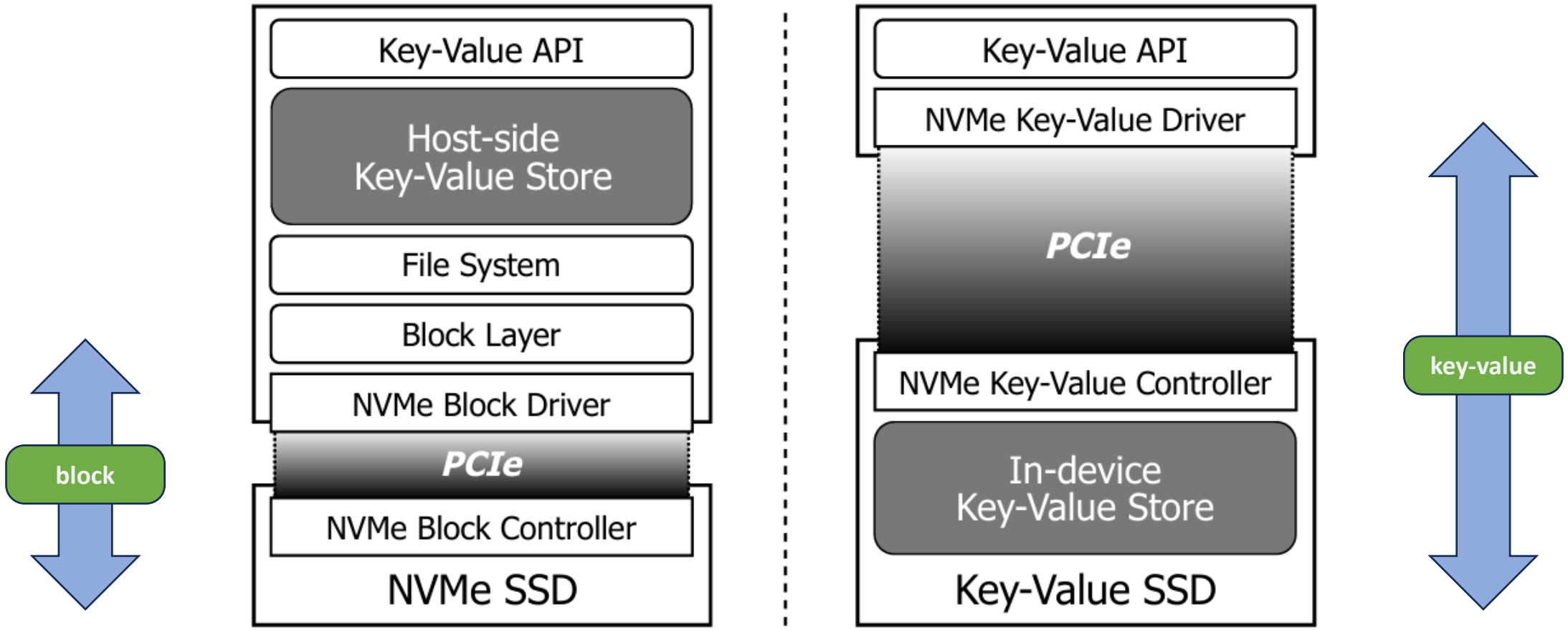
Key Idea

- What about streamlining these layers from the storage stack?
  - By making a key-value pair as the unit of data communication interface



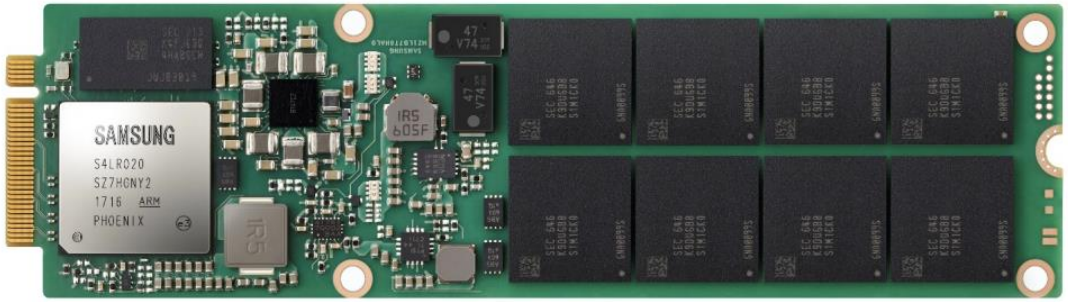
Key Idea

- What about streamlining these layers from the storage stack?
  - By making a key-value pair as the unit of data communication interface

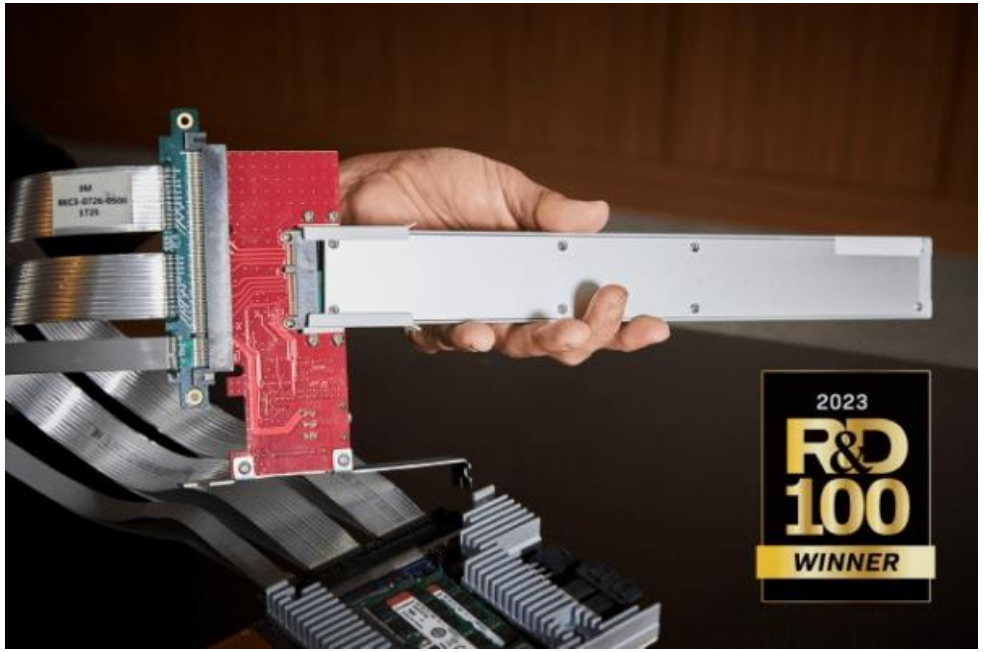


Key-Value Solid-State Drive (KV-SSD)

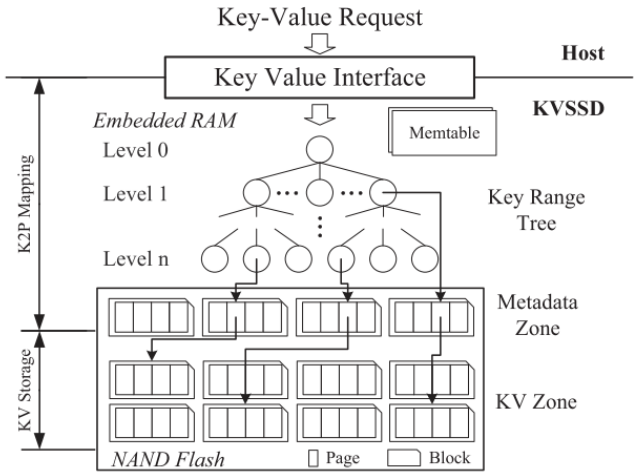
Samsung KVSSD

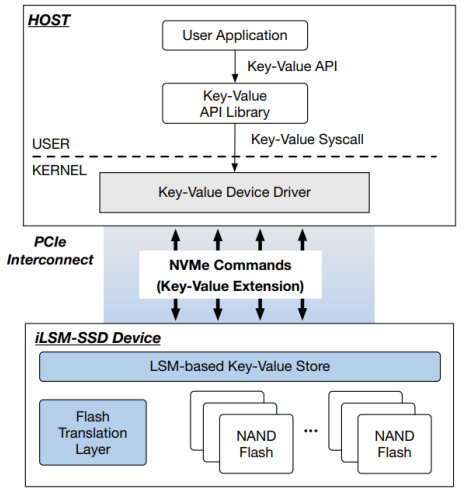


SK hynix KV-CSD



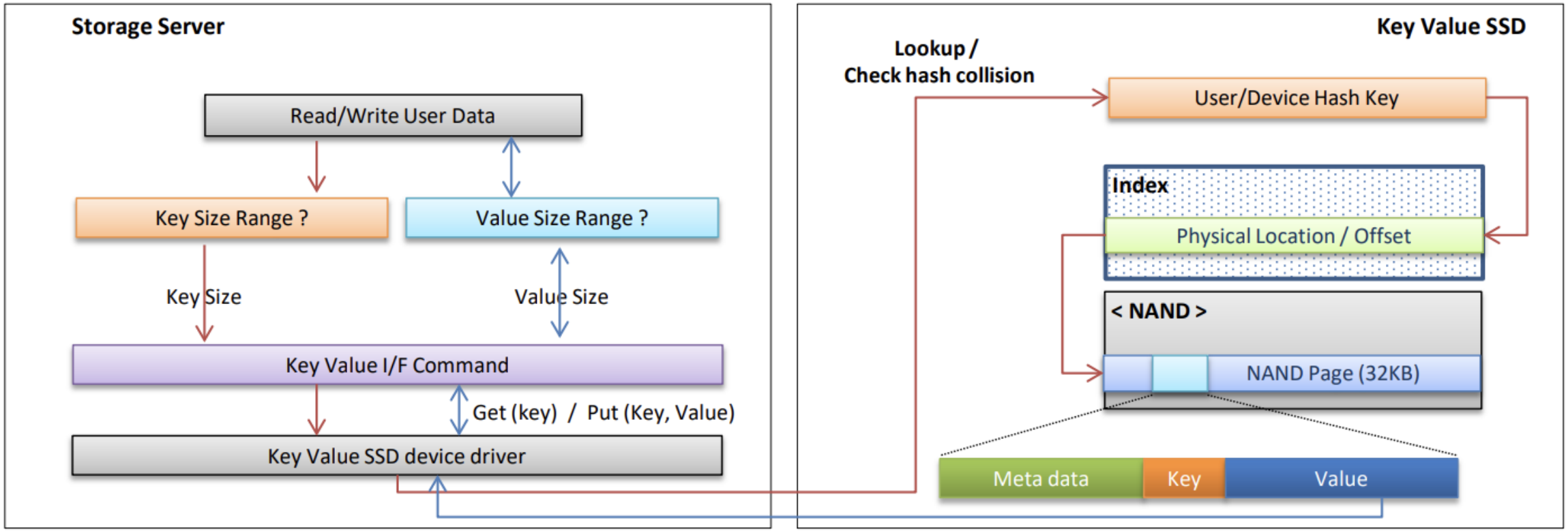
Academia



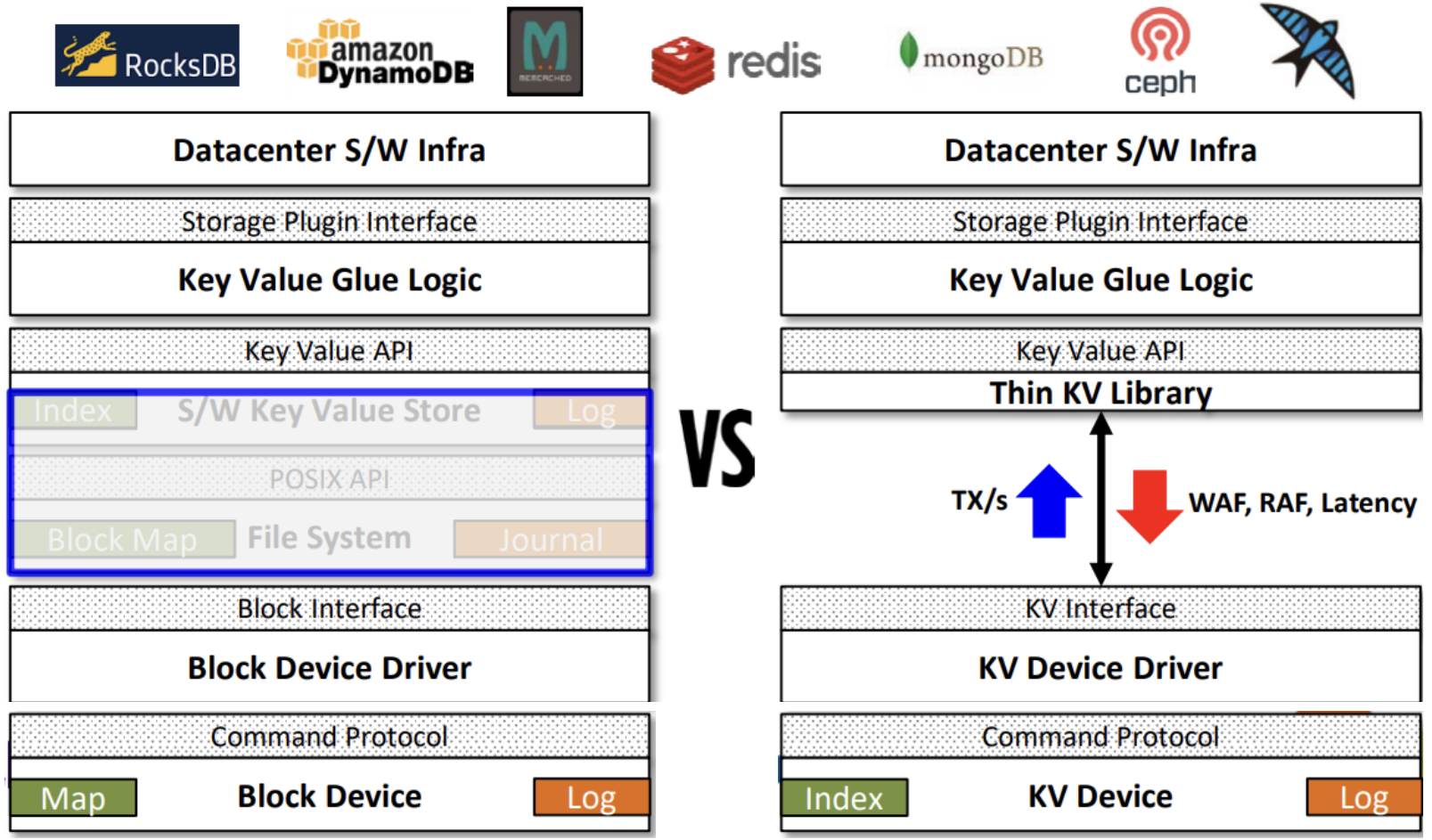


Key-Value Solid-State Drive (KV-SSD)

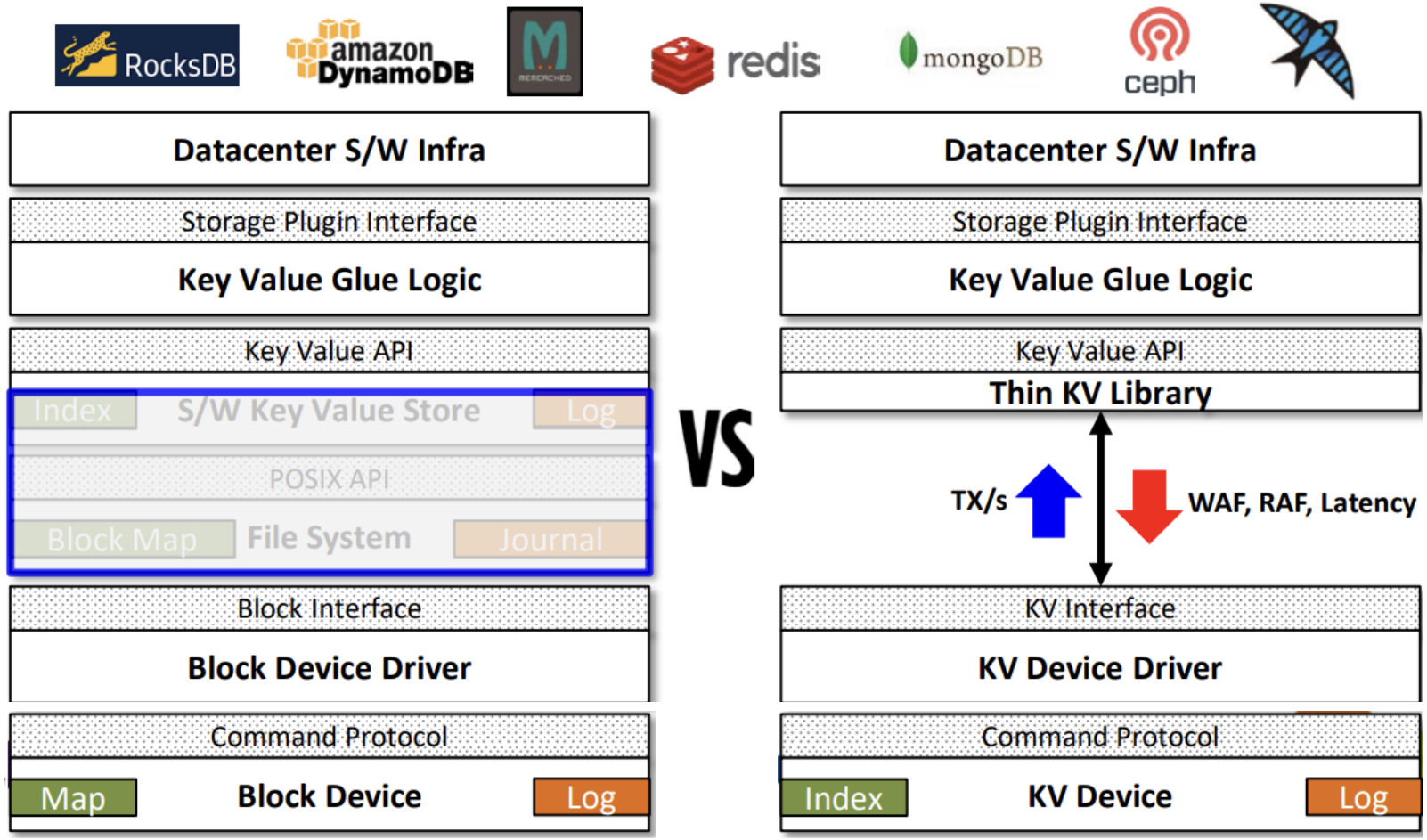
- SSD with a native key-value interface, rather than a block interface
  - SSD supports key-value store operations like PUT, GET, SEEK, and NEXT
  - SSD maintains Key-to-Page mapping index structures like Hash or LSM-tree



Key-Value Solid-State Drive (KV-SSD)



Key-Value Solid-State Drive (KV-SSD)



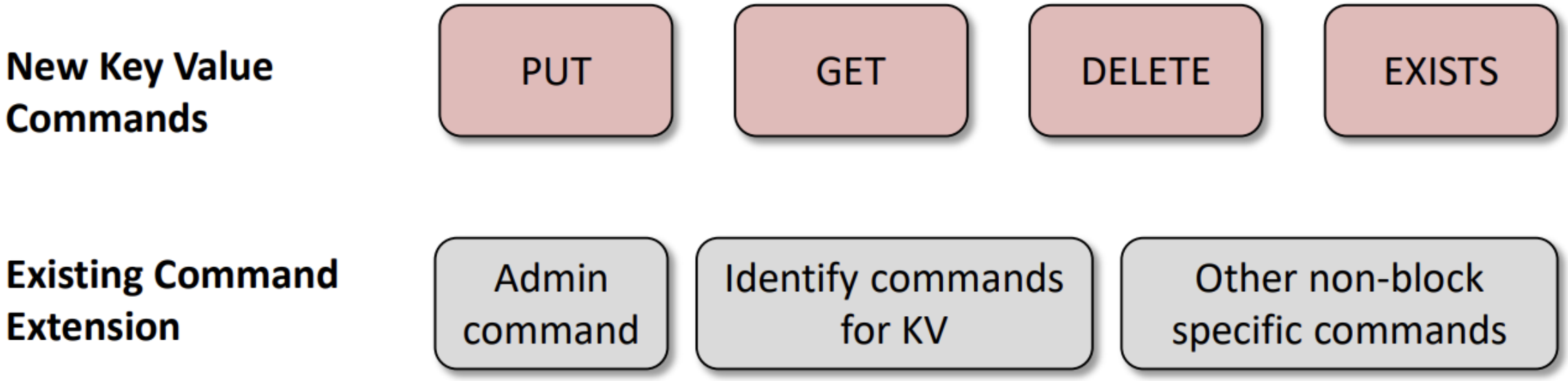
- ***Thin Storage Stack*** *leads to optimal performance for key-value store operations*
- ***Offloaded Key-Value Store*** *reduces the loads of host node, leading to full disaggregation*

Design Space of KV-SSDs

- KV-SSD has multiple design choices
  - In-storage Key-Value Store Implementation
  - **Hash-based KV-SSD**
    - Simple design
    - Low compute power required
    - Limited operations and relatively low write performance
    - Ex) Samsung KVSSD (2019), KAML (HPCA ’17), Dotori (VLDB ’23), KVrangeDB (TOS ’23)
  - **LSM-tree-based KV-SSD**
    - Complex design
    - High compute power required
    - Various operations and relatively high write performance
    - Ex) SK hynix KV-CSD (2023), iLSM-SSD (MASCOTS ’19), LightStore (ASPLOS ’19)

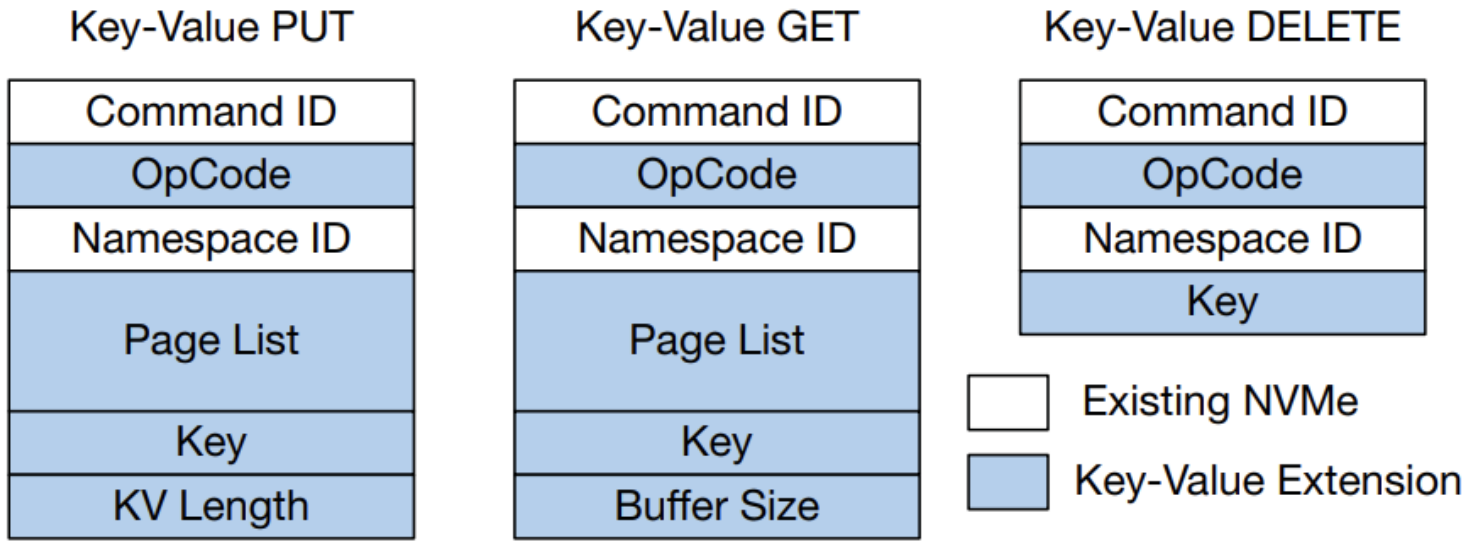
Standardization Efforts for KV-SSDs

- NVMe protocol, a contemporary storage protocol designed for high-speed NAND flash memory, has recently introduced a standard key-value command set for KV-SSDs



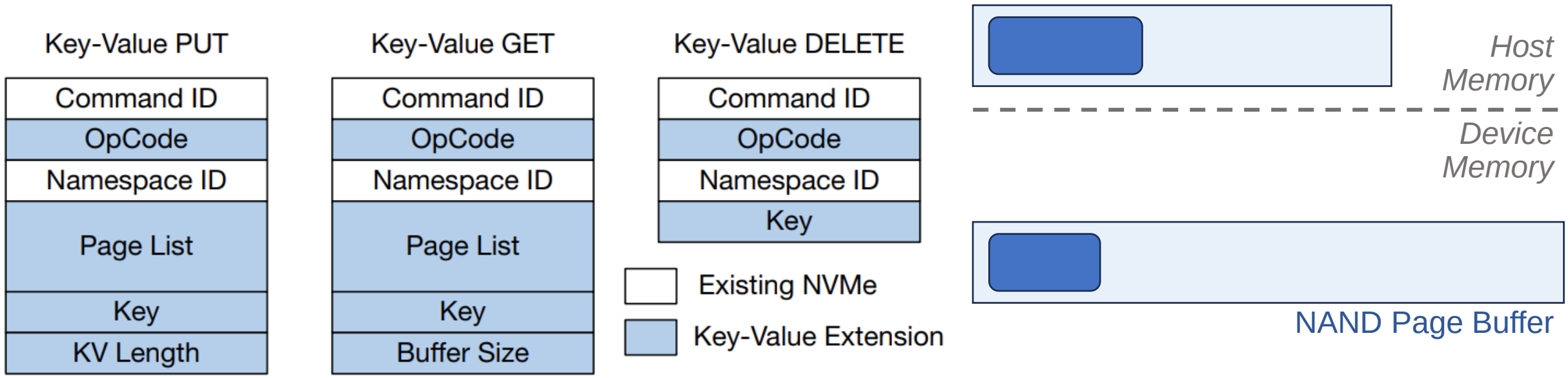
Standardization Efforts for KV-SSDs

- A key and metadata are transferred via NVMe command submission queue entry, and value is transferred via DMA transactions



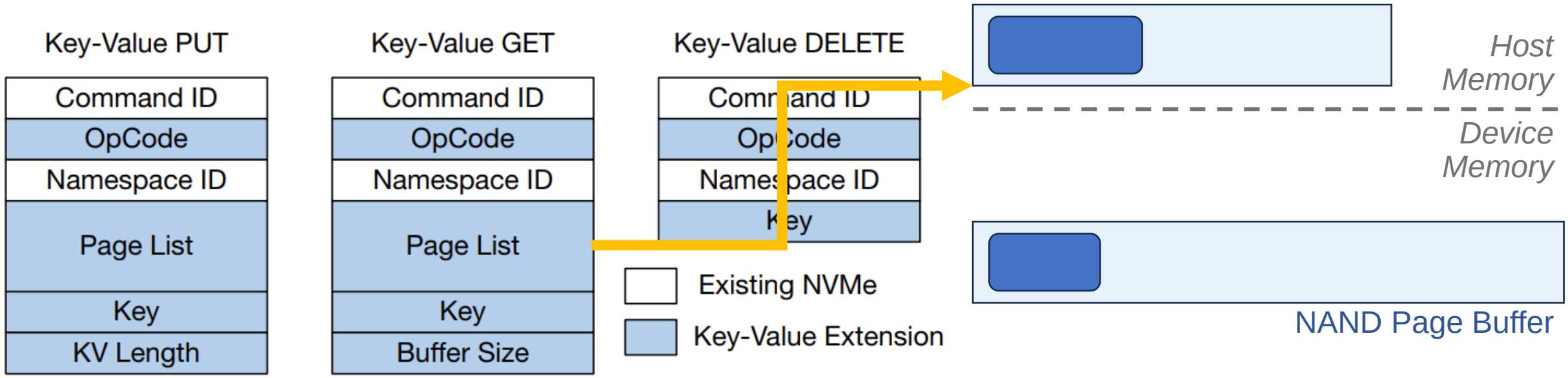
Standardization Efforts for KV-SSDs

- A key and metadata are transferred via NVMe command submission queue entry, and value is transferred via DMA transactions



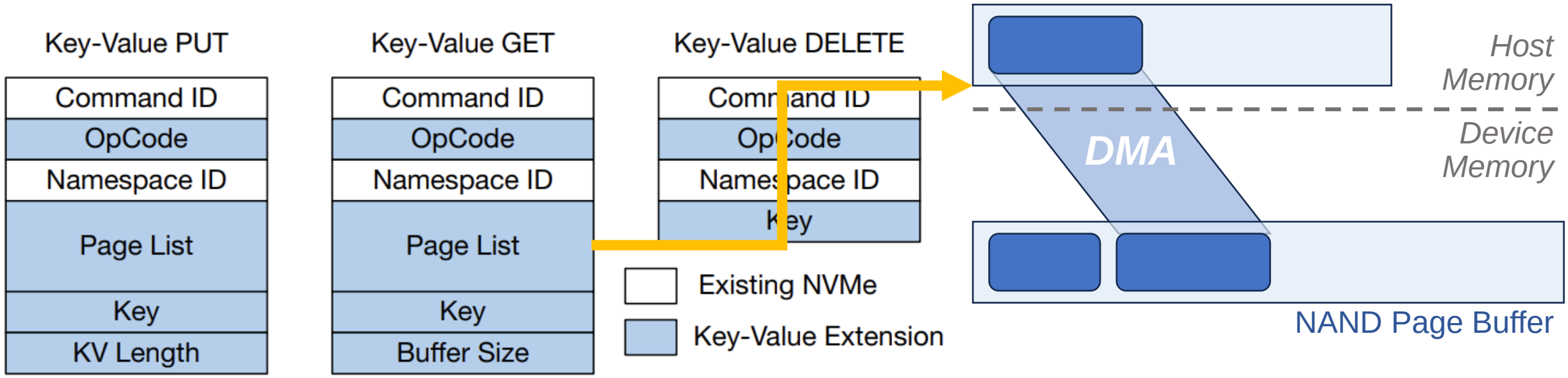
Standardization Efforts for KV-SSDs

- A key and metadata are transferred via NVMe command submission queue entry, and value is transferred via DMA transactions



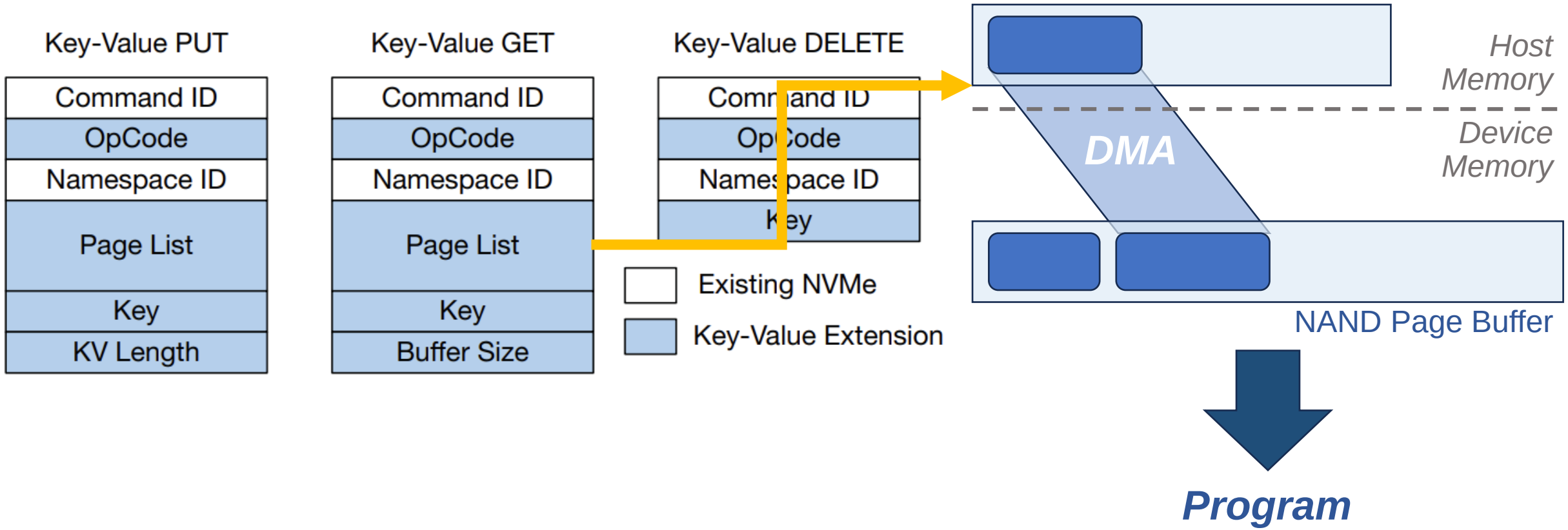
Standardization Efforts for KV-SSDs

- A key and metadata are transferred via NVMe command submission queue entry, and value is transferred via DMA transactions



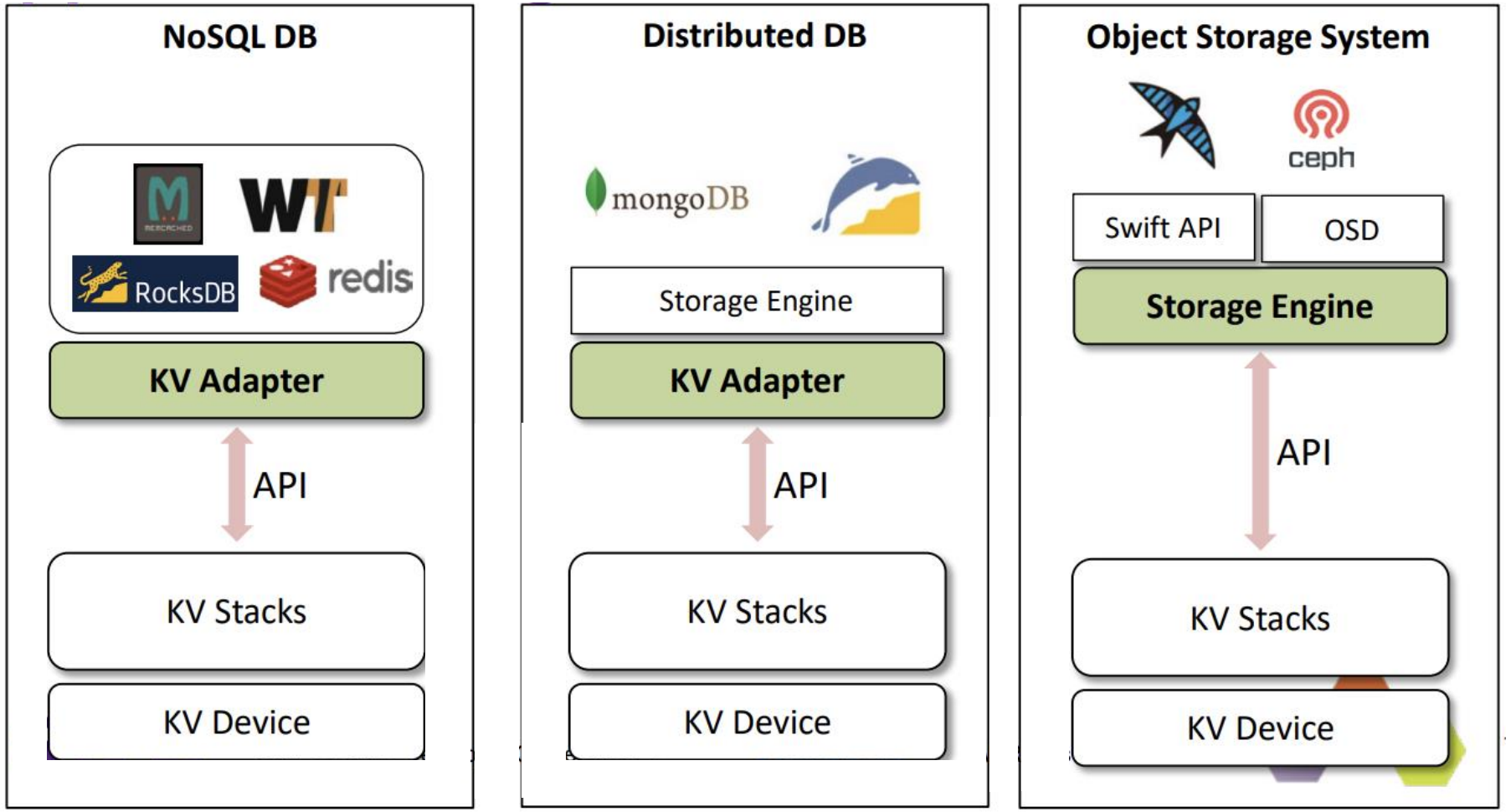
Standardization Efforts for KV-SSDs

- A key and metadata are transferred via NVMe command submission queue entry, and value is transferred via DMA transactions



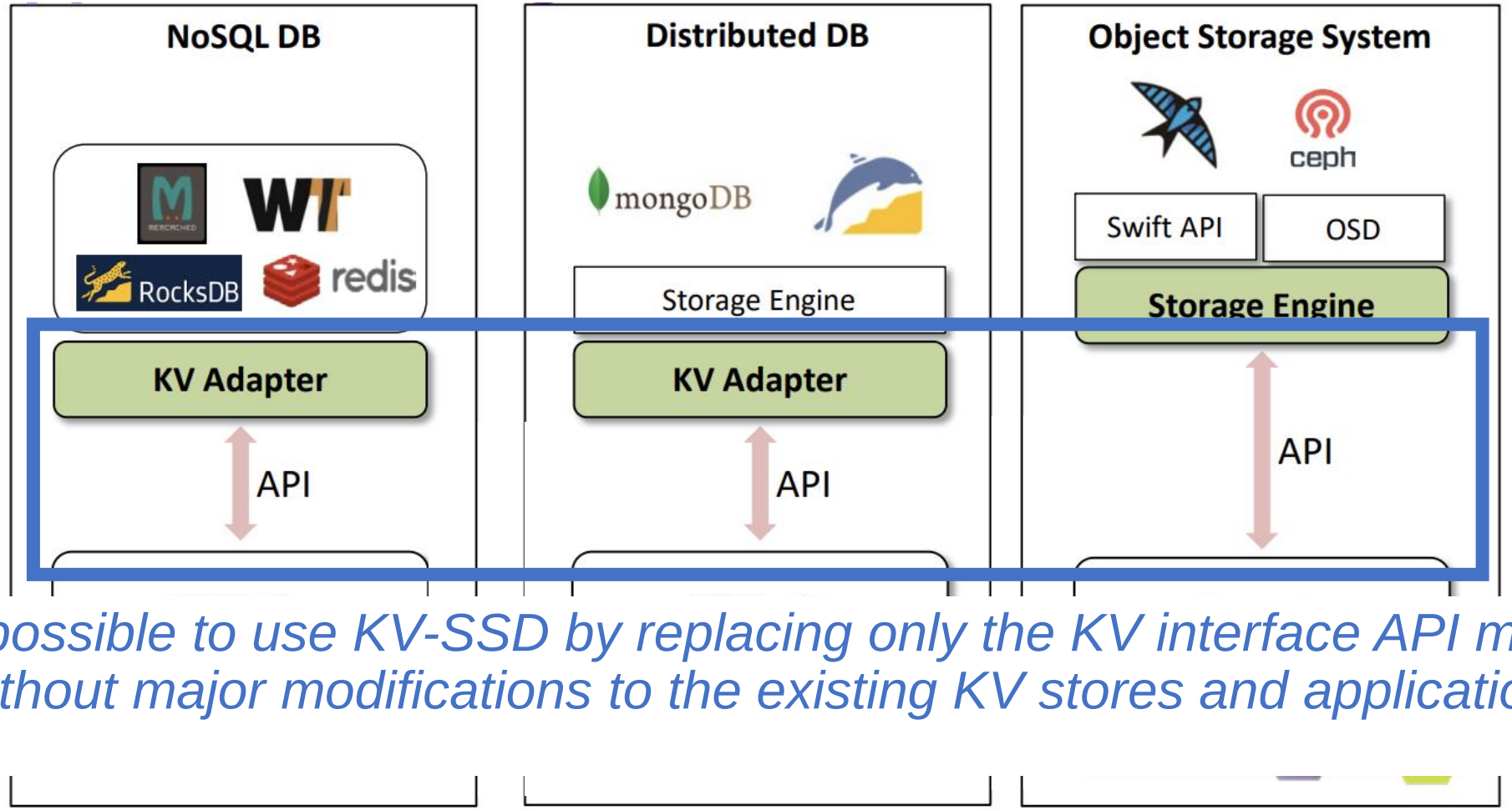
KV-SSD Usecases

- KV-SSD can be used independently, but it is also envisioned to be integrated with existing database and storage systems



KV-SSD Usecases

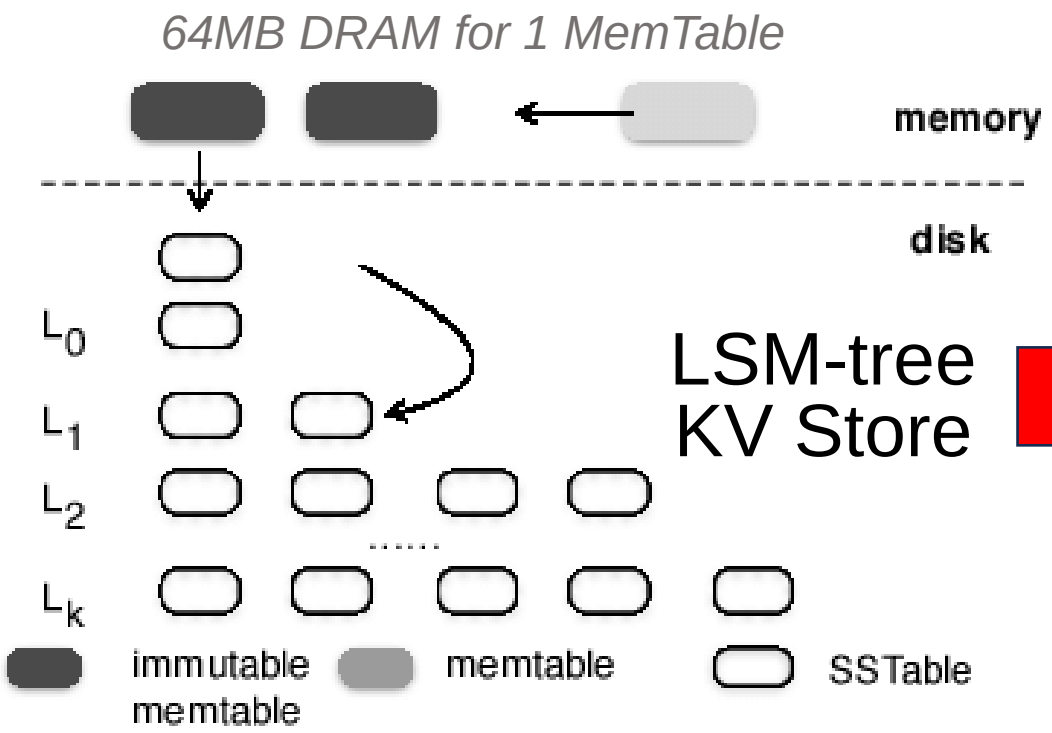
- KV-SSD can be used independently, but it is also envisioned to be integrated with existing database and storage systems



It is possible to use KV-SSD by replacing only the KV interface API module without major modifications to the existing KV stores and applications

Challenges in KV-SSDs

- Resource-constraint nature of SSDs
  - Deploying key-value stores (Hash or LSM-tree) within SSDs requires more memory space than traditional SSDs do

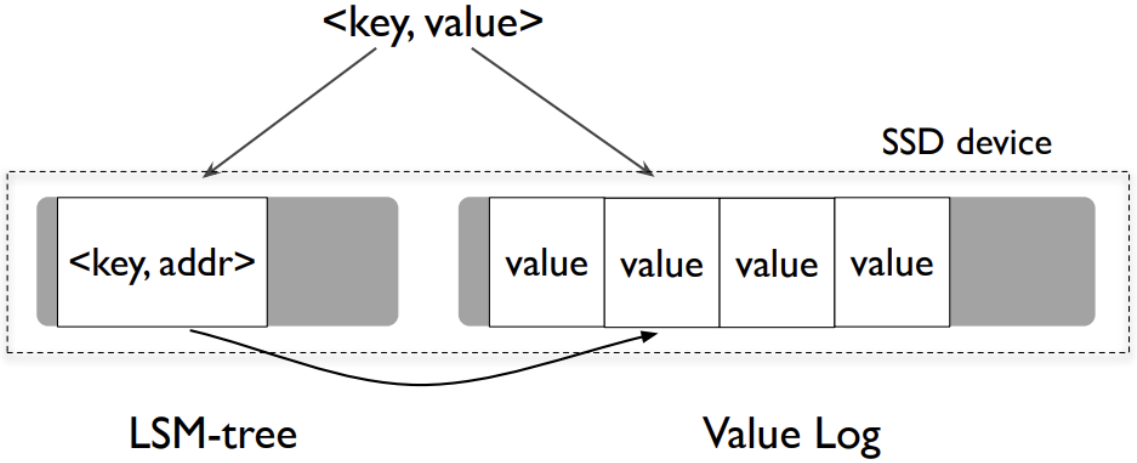


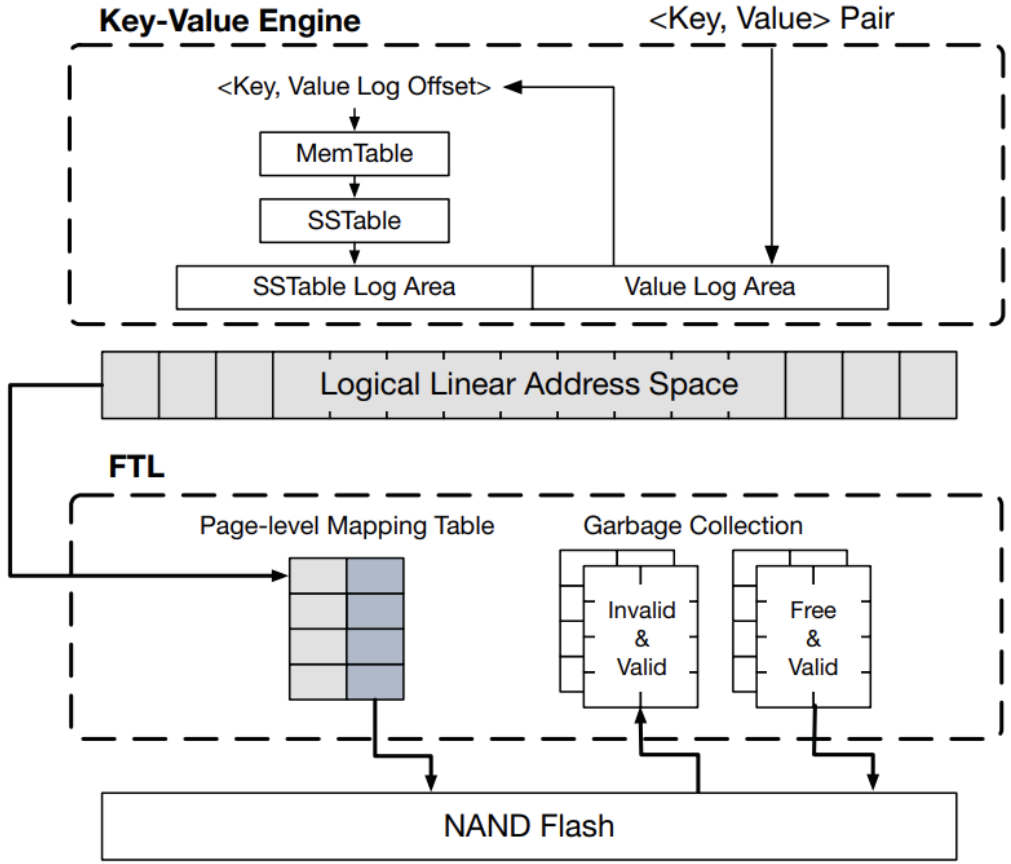
LSM-tree KV Store

※ LSM-tree is a data structure widely used in persistent key-value stores (e.g., RocksDB). It is known for offering high write throughput through append-only sequential write pattern.
※ MemTable is a memory component of LSM-tree. It stores recently inserted key-value pairs. It is periodically flushed to the disk, constructing the disk files called SSTables.

Challenges in KV-SSDs

- Resource-constraint nature of SSDs
  - iLSM-SSD (MASCOTS ’19) introduced in-storage LSM-tree with a key-value separation technique
    - It manages values independently from the LSM-tree, thereby reducing the MemTable size and consequently requiring less compute power and memory space within the SSDs



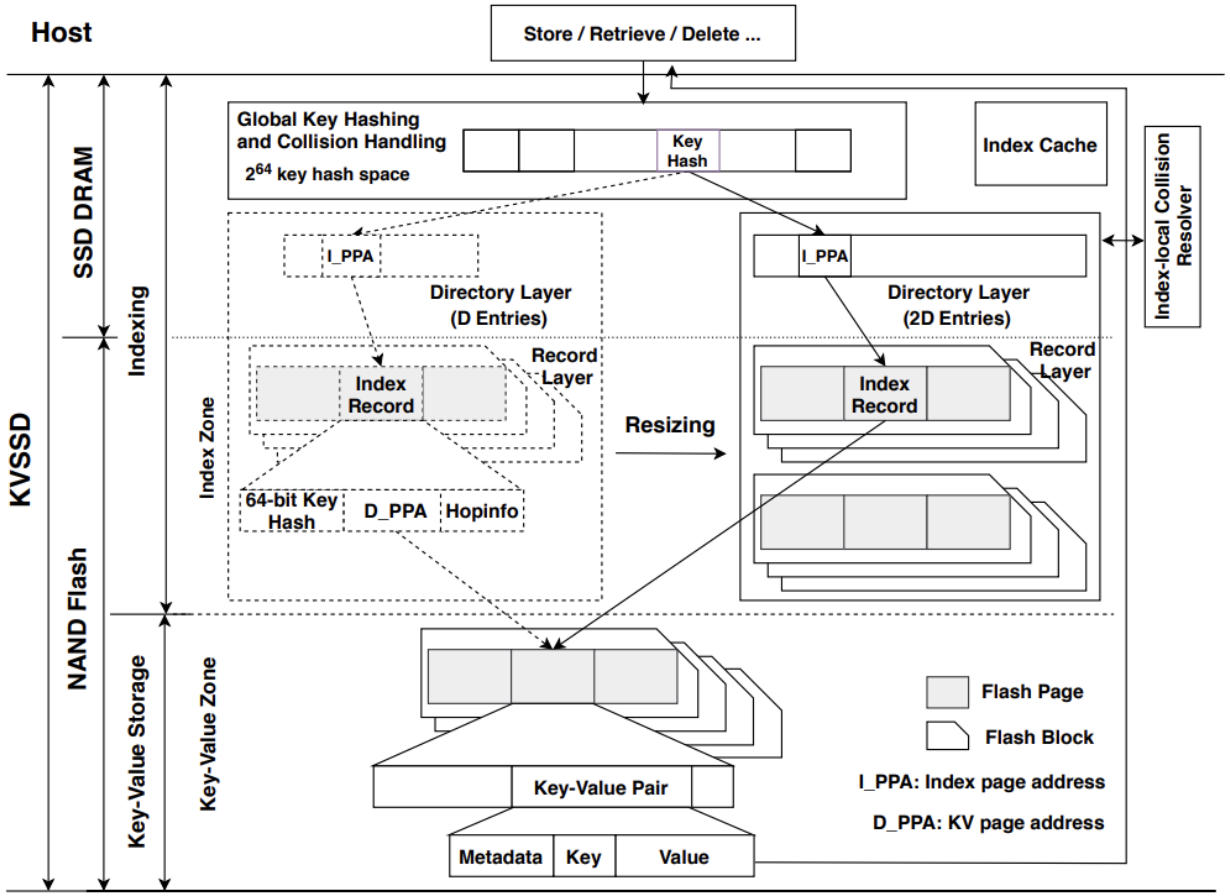


※ A traditional LSM-tree performs compactions for SSTables which are composed of raw key-value pairs. Compations periodically sort, merge and reconstruct SSTables, issuing frequent read and write operations.
※ The key-value separation not only reduces the size of tables, but also the amount of I/O operations occurred during compactions.

Challenges in KV-SSDs

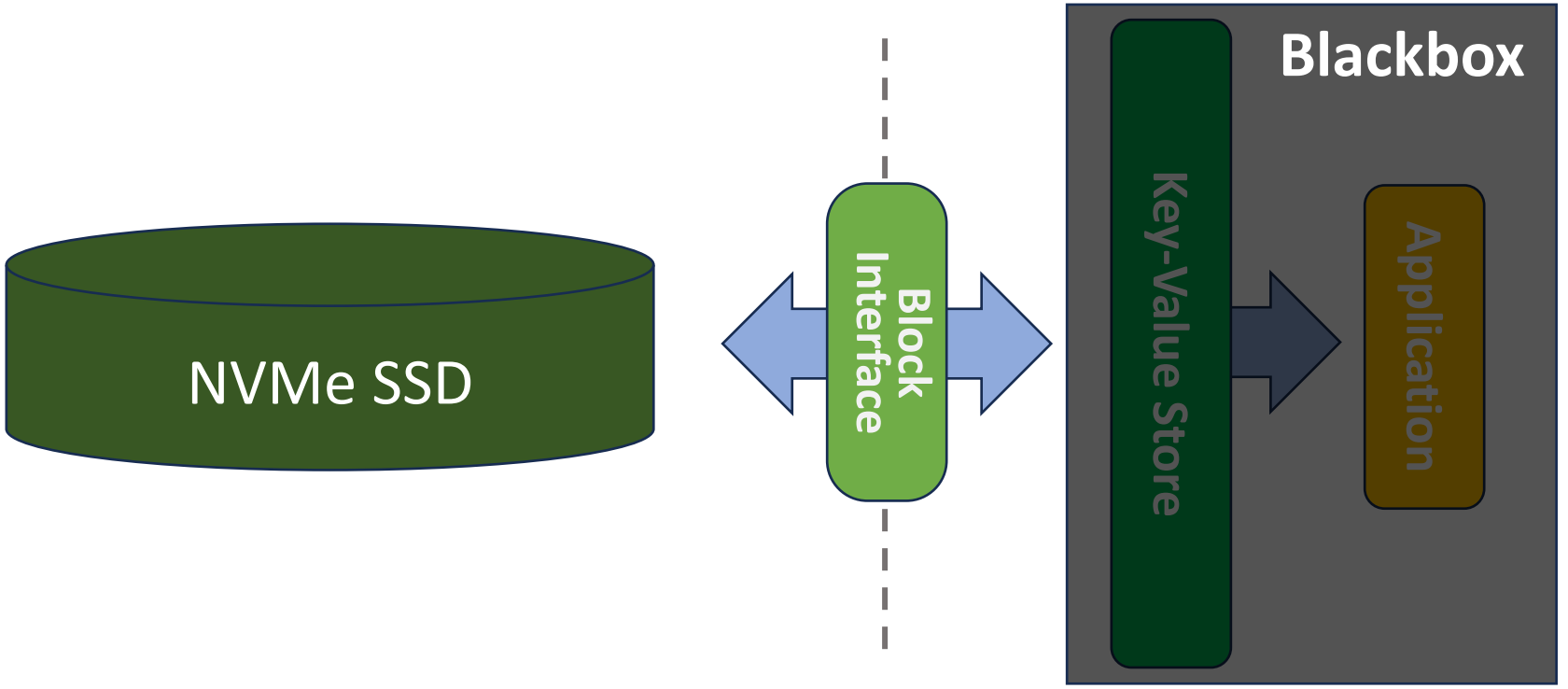
- Resource-constraint nature of SSDs
  - RHIK (HPDC ’23) presented a resizable key-value-aware hash-based indexing scheme that requires a maximum of one NAND flash reads to fetch metadata
    - It maintains a Directory Layer in DRAM
    - Each entry in the Directory Layer points to one Record Layer entry
    - Each entry in the Record Layer is stored in only one NAND page
    - Each record in the entry points to NAND pages where the value is stored

    ➔ Less memory space for indexing



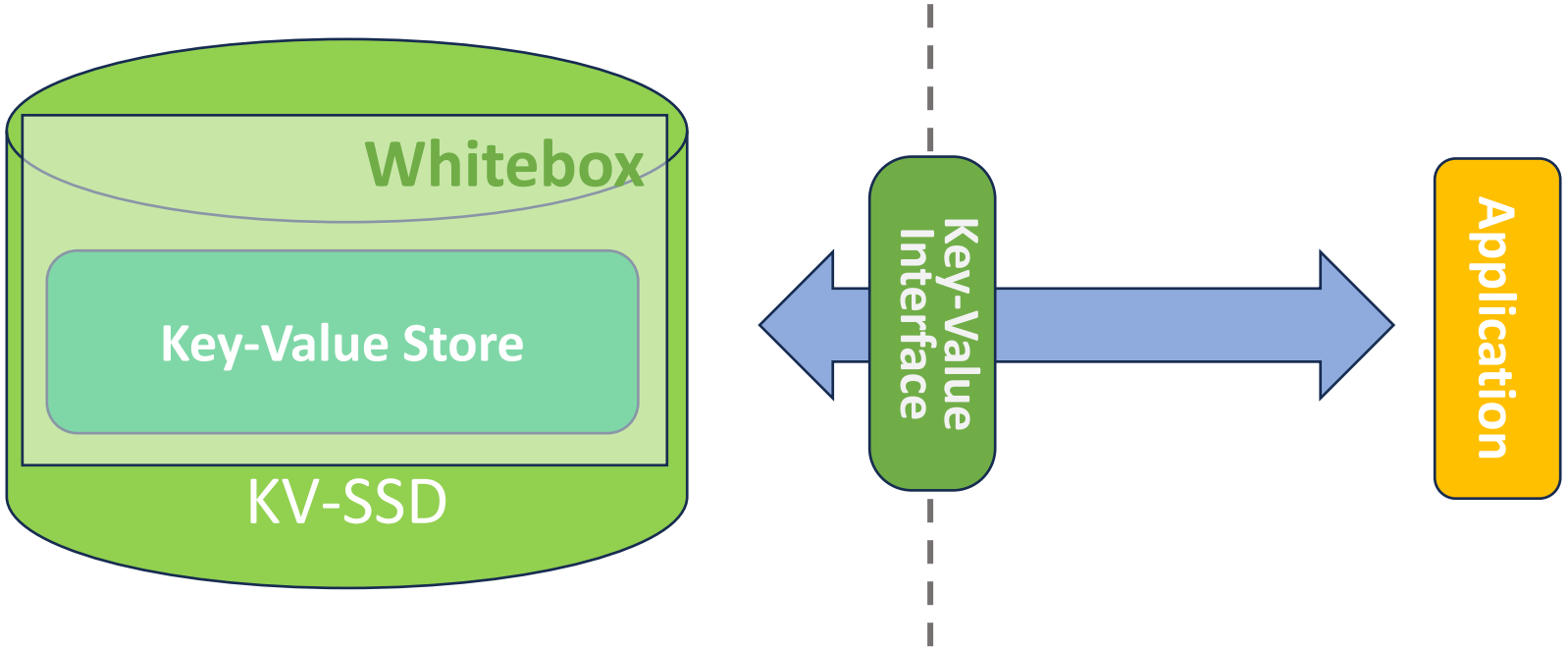
Challenges in KV-SSDs

- How to optimize key-value store functionalities
  - KV-SSD must utilize its awareness of key-value indexing scheme
    - Back then, the storage device had limited opportunities to optimize its operations for improved key-value store performance



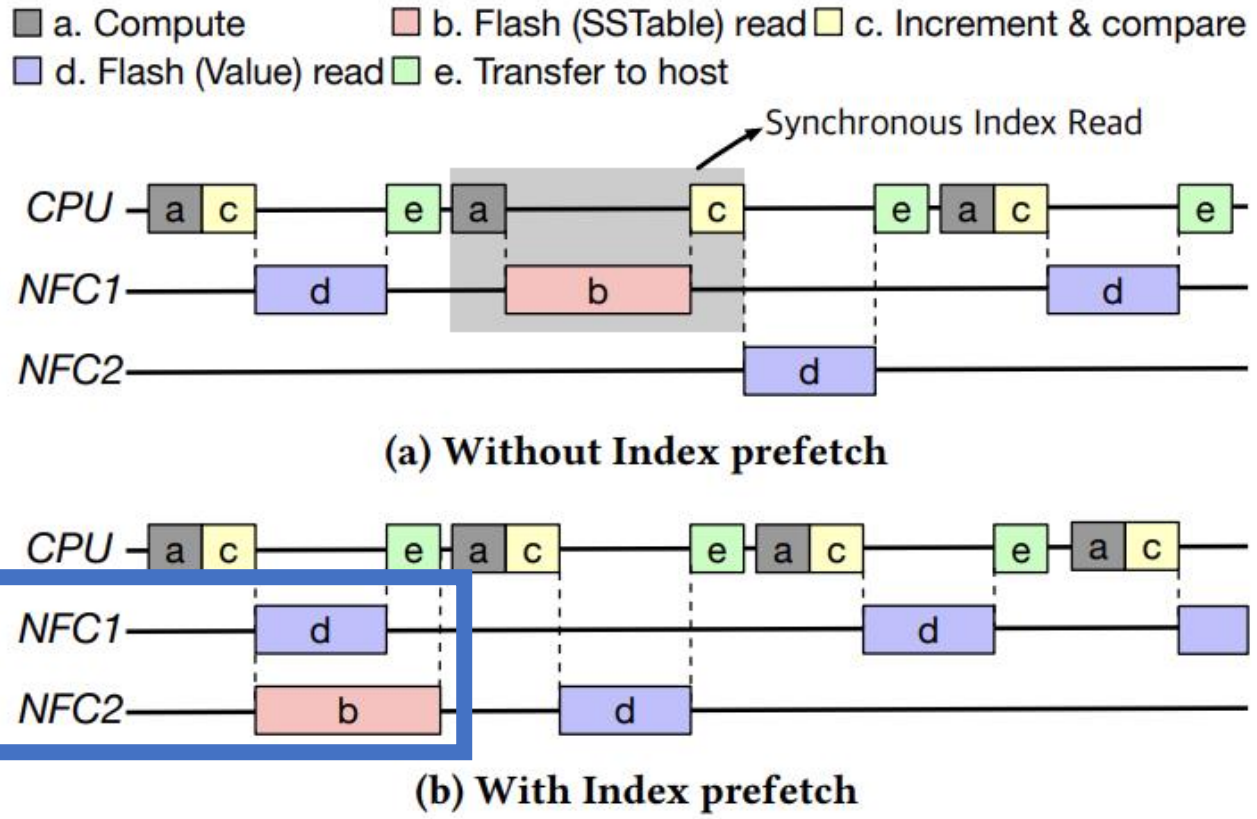
Challenges in KV-SSDs

- How to optimize key-value store functionalities
  - KV-SSD must utilize its awareness of key-value indexing scheme
    - Now the device knows when and how the key-value store operations (e.g., range scan, MemTable flush, compaction) will be performed



Challenges in KV-SSDs

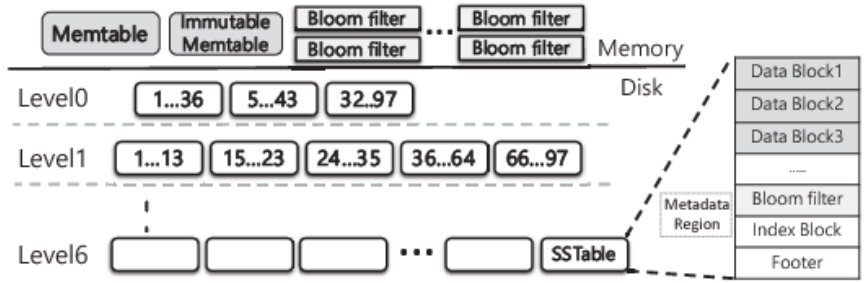
- How to optimize key-value store functionalities
  - IterKVSSD (Systor '23) proposed a prefetching mechanism for index tables and values to improve the efficiency of serving range queries within LSM-tree-based KV-SSDs



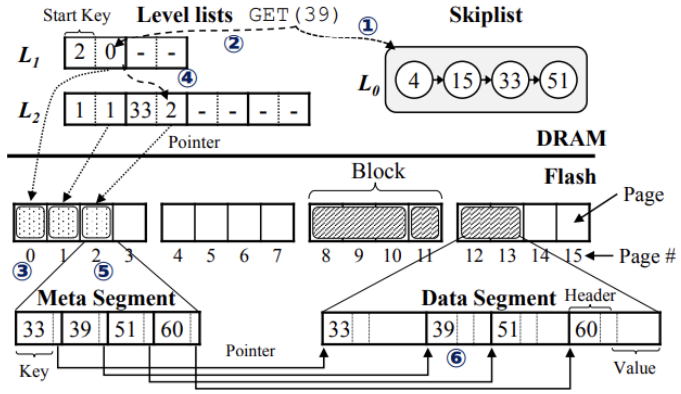
Asynchronous and Parallel NAND reads

Challenges in KV-SSDs

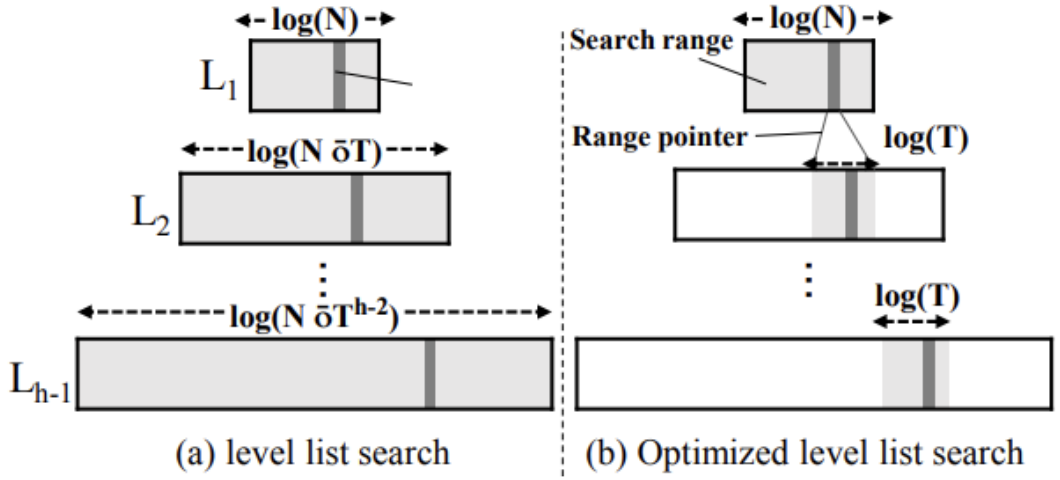
- How to optimize key-value store functionalities
  - PinK (ATC ’20) presented a level-pinning strategy to eliminate Bloom Filters from in-storage LSM-tree, reducing the amount of in-device memory space. Plus, PinK suggests an optimized in-storage LSM-tree search policy



Traditional LSM-tree w/ Bloom Filters



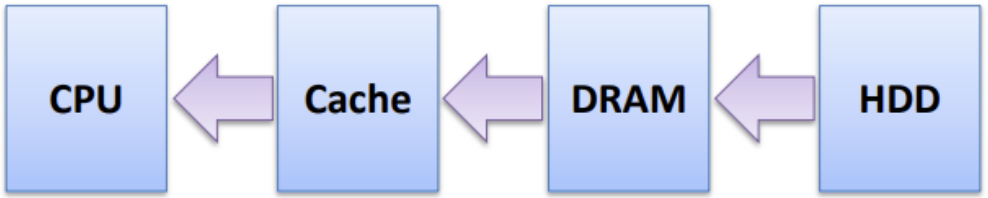
PinK LSM-tree KVSSD w/o Bloom Filters



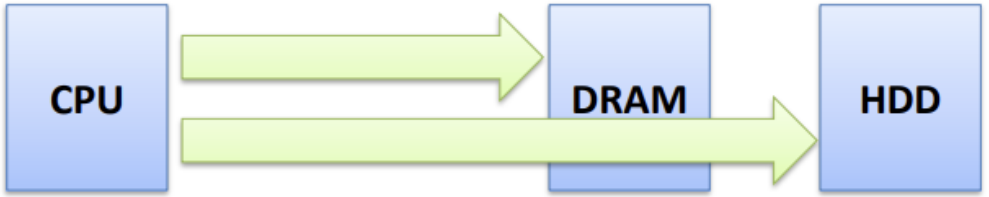
No need to read whole target SSTables

Integration with Computational Storage

- In parallel, there are Computational Storage Devices (CSDs) designed to offload and process some of the data-intensive operations from the host (near-data processing)
  - In memory hierarchy
    - Move data toward ALU to remedy long latency while accessing high-locality data



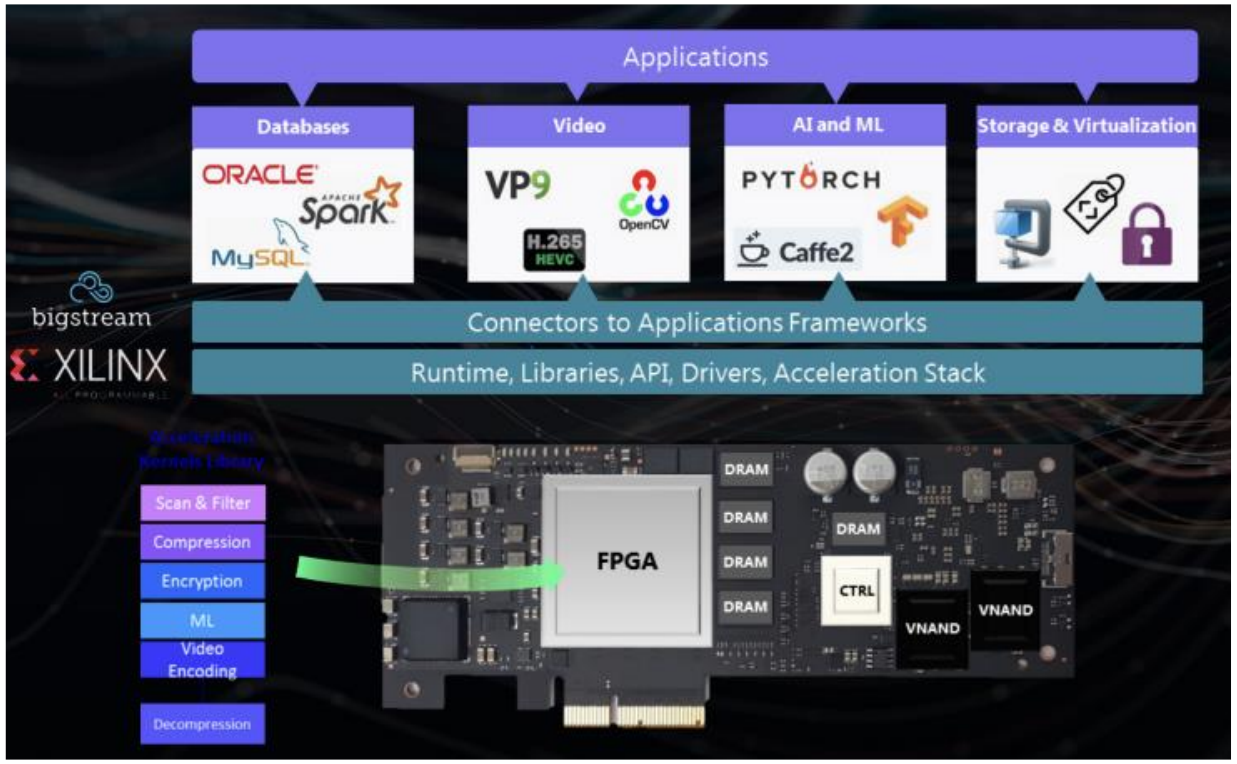
  - In computation hierarchy
    - Move computation toward memory to remedy long latency while accessing low-locality data



※ CSDs have a much longer history compared to KV-SSDs; it's a concept that has been discussed since the 1990s.

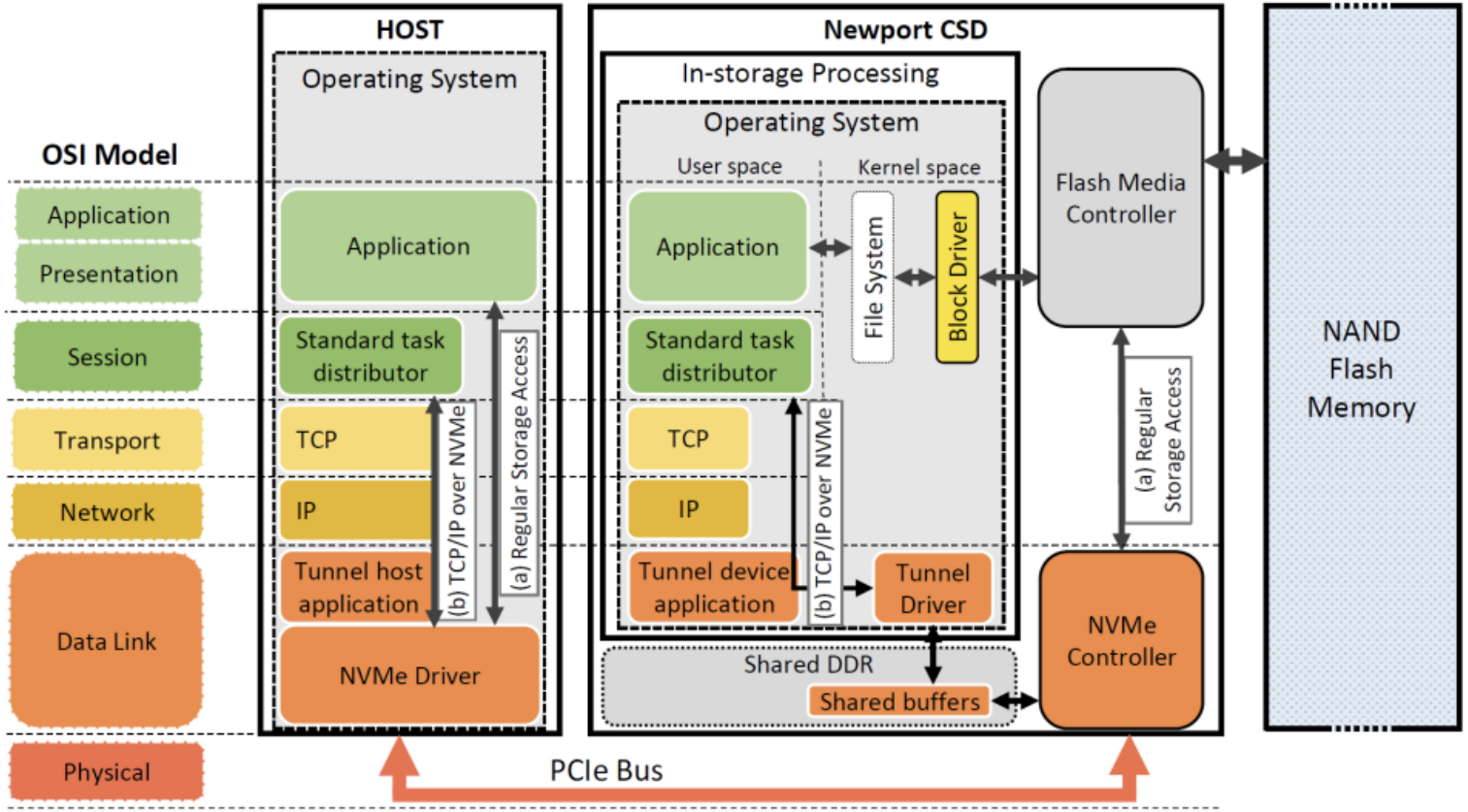
Integration with Computational Storage

- Products like the Samsung SmartSSD and Newport CSD are already on the market, offering the ability to port and run specialized programs (binaries) directly from the host



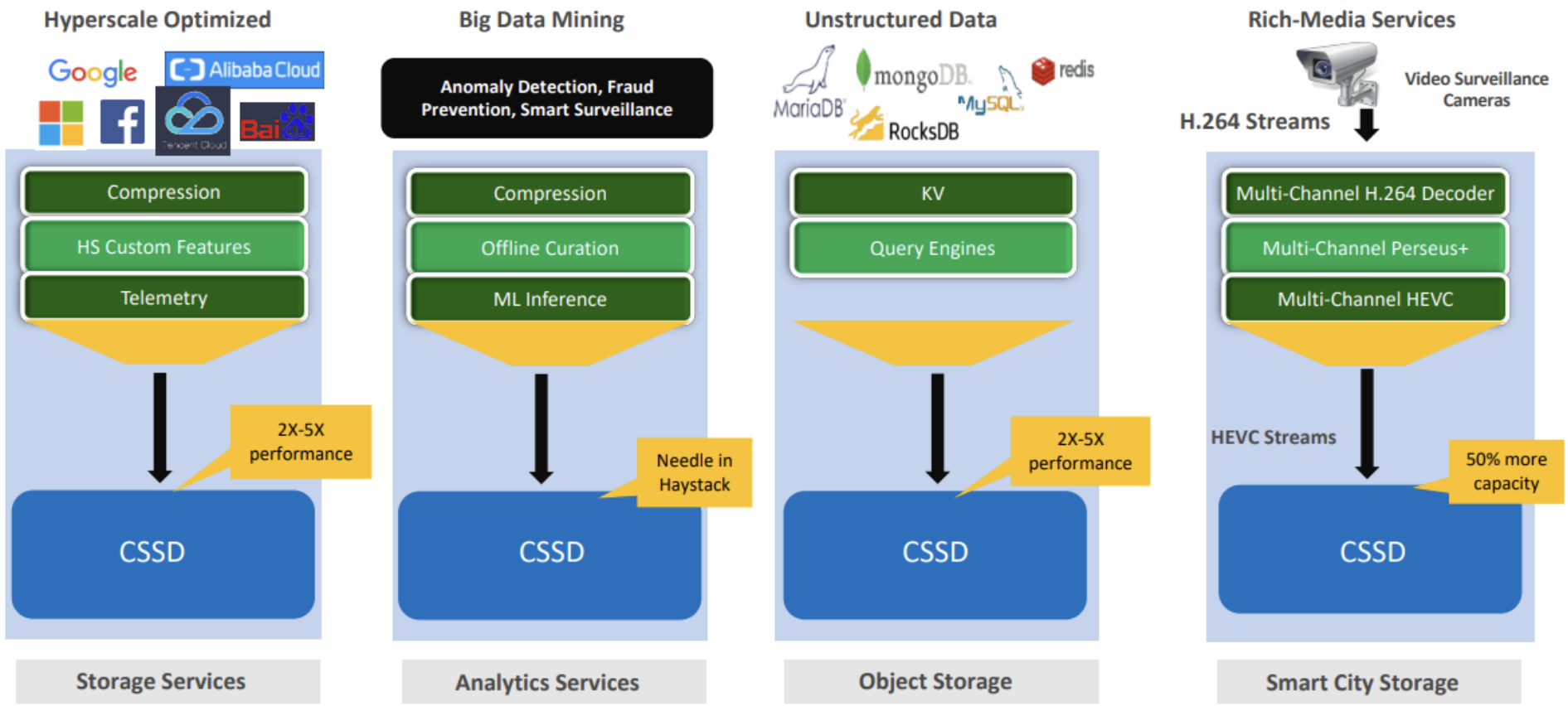
Integration with Computational Storage

- They provide a programmable environment by installing a separate operating system within the device, which enables customers to carry out optimizations tailored to their specific services



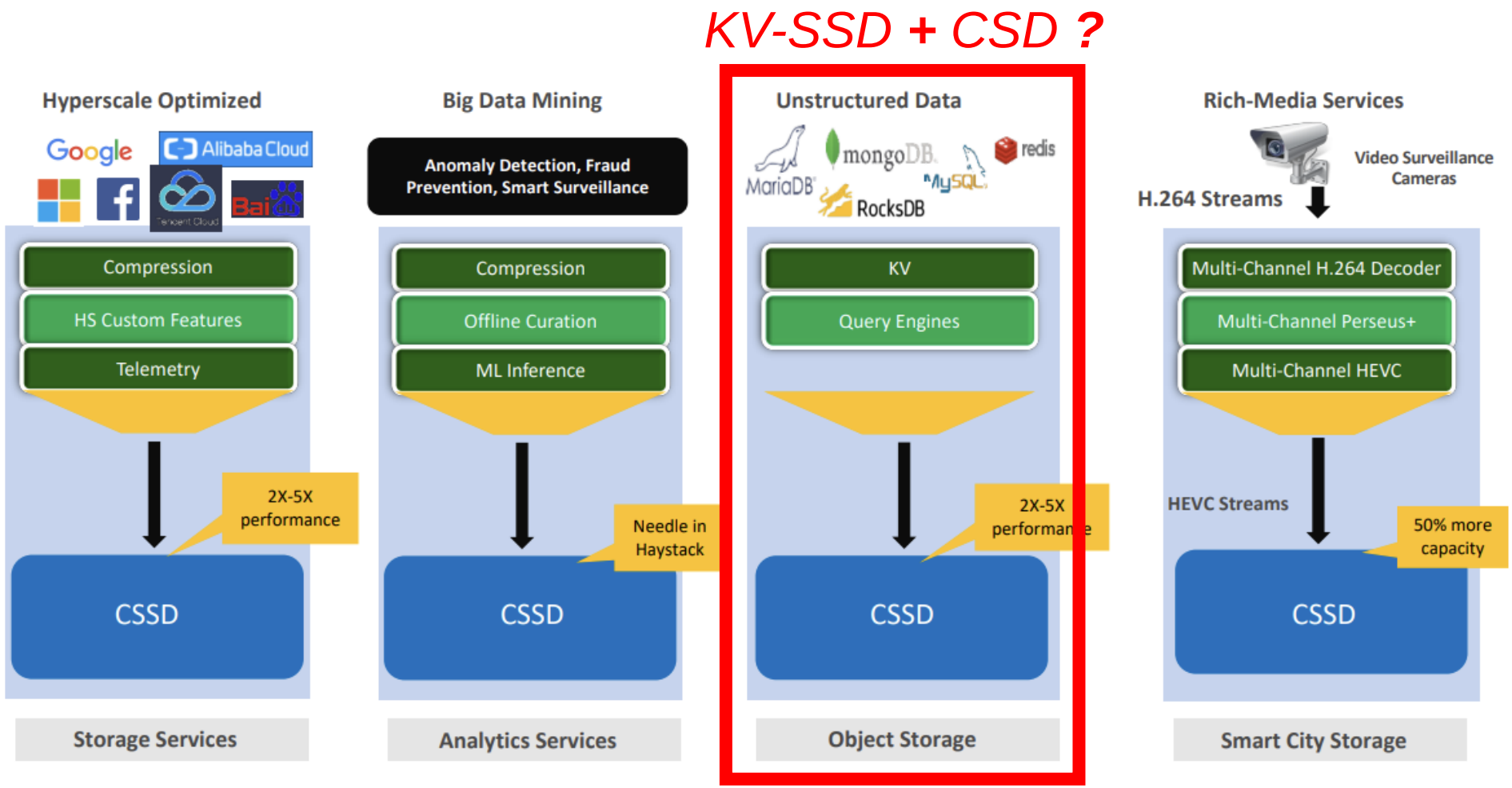
Integration with Computational Storage

- CSDs are similar to KV-SSDs, but they have a different purpose. CSDs target more comprehensive and general scenarios



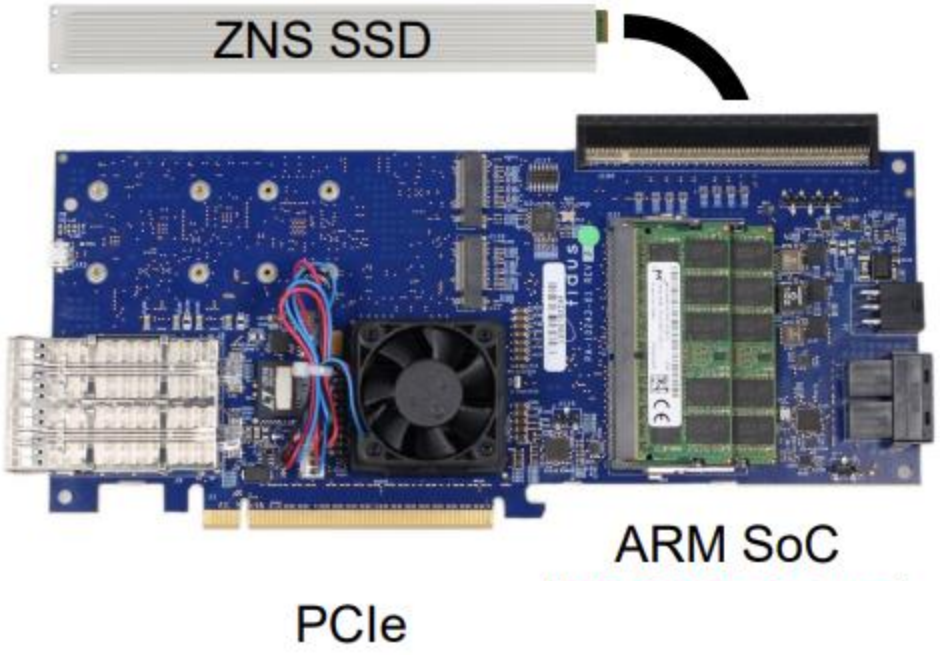
Integration with Computational Storage

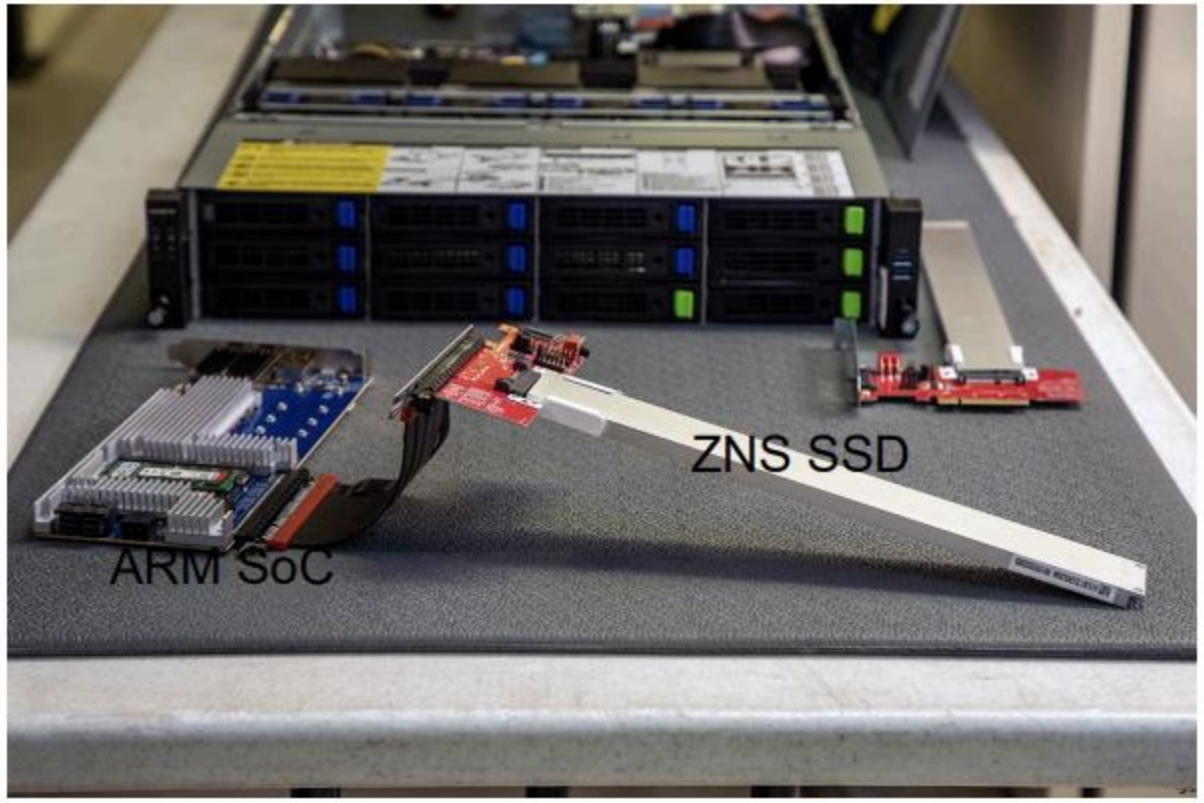
- What about offloading query processing to KV-SSDs?



Key-Value Computational SSD (KV-CSD)

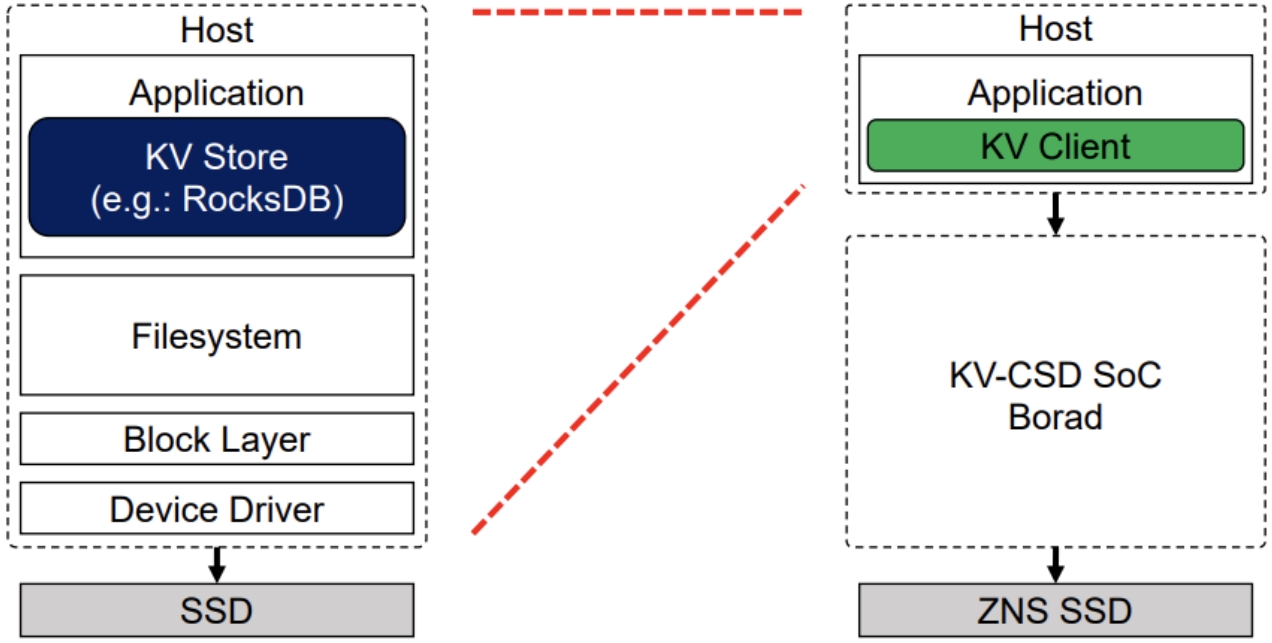
- SK hynix recently announced KV-CSD, which is a fusion and optimization of the existing KV-SSD and CSD concepts





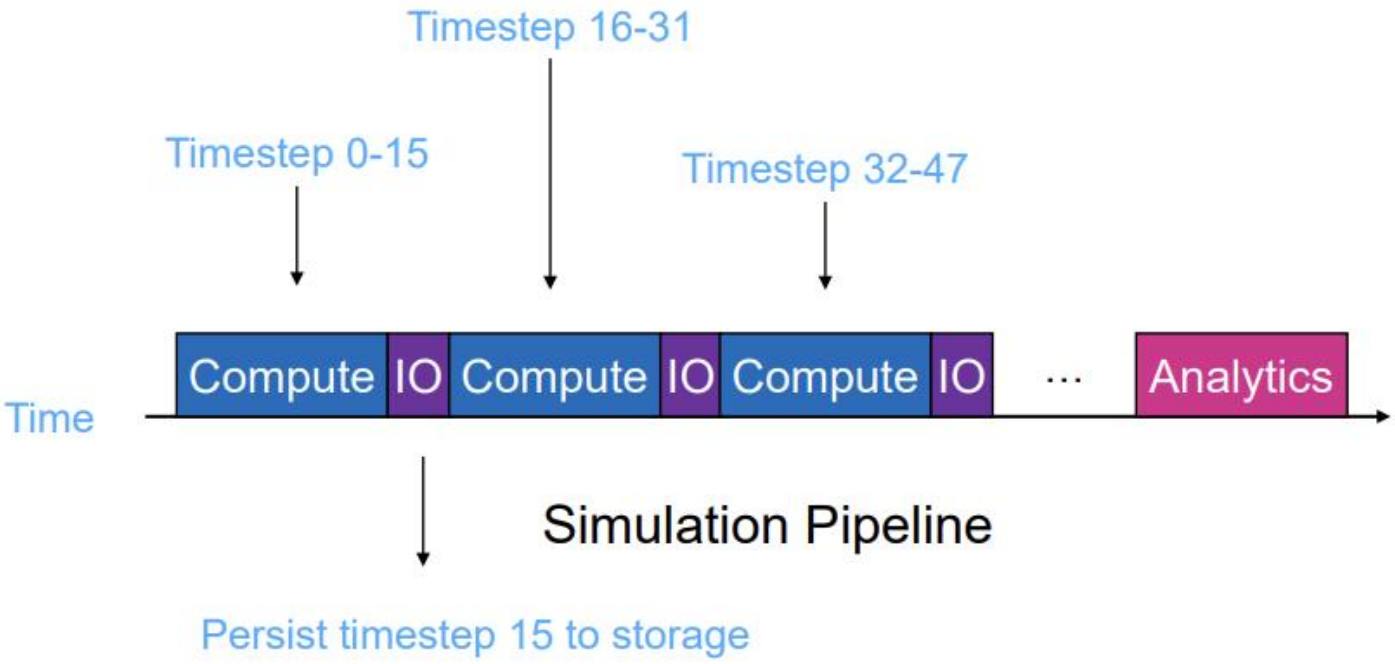
Key-Value Computational SSD (KV-CSD)

- KV-CSD provides a key-value interface through an FPGA-based SoC positioned as an intermediary layer between the storage and the host
  - It aims to offer not only the device-level key-value store operations but also user-defined near-data processing capabilities
  - For the storage device, Zoned Namespaces (ZNS) were used to achieve cost optimization as well



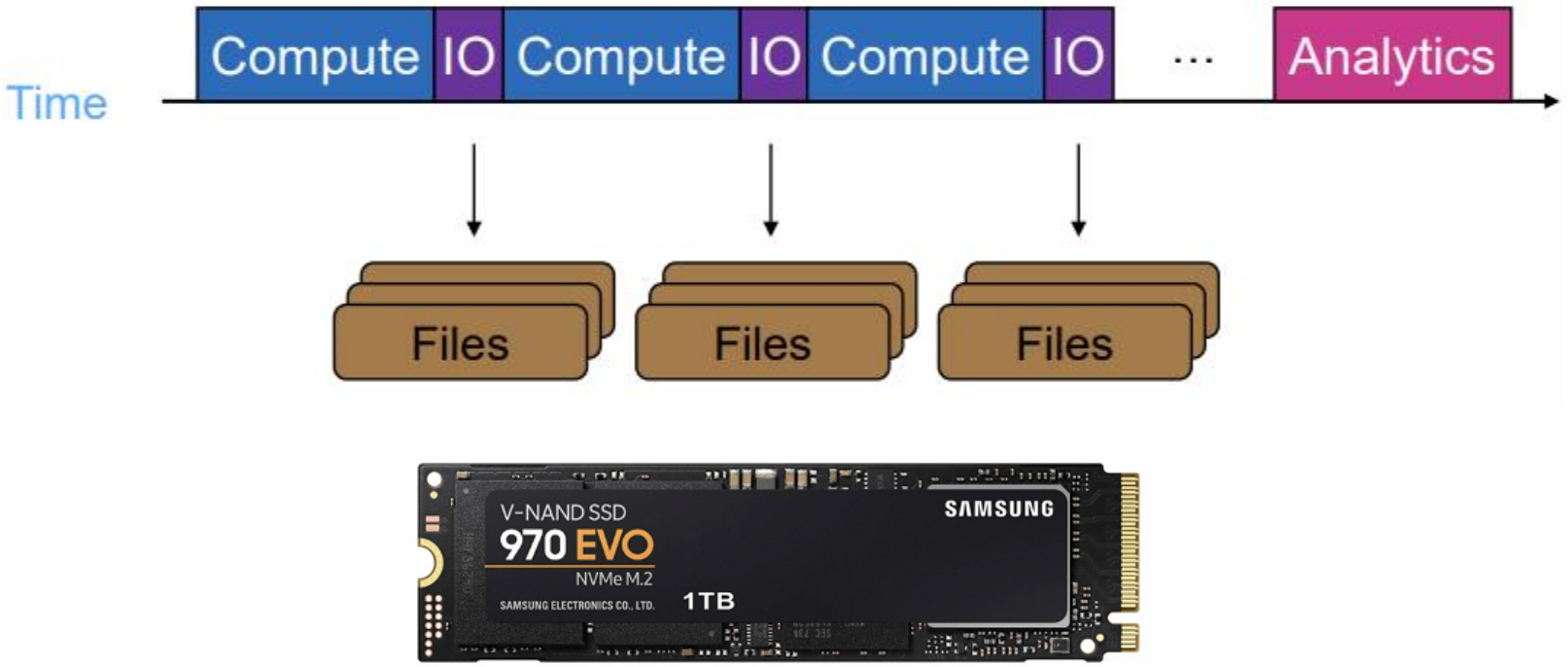
Target Scenarios of KV-CSD

- When we look into scientific analytics applications which are often based on Key-Value / Object Stores, they iterate between compute & I/O phases with time-based bulk-synchronous parallel programs



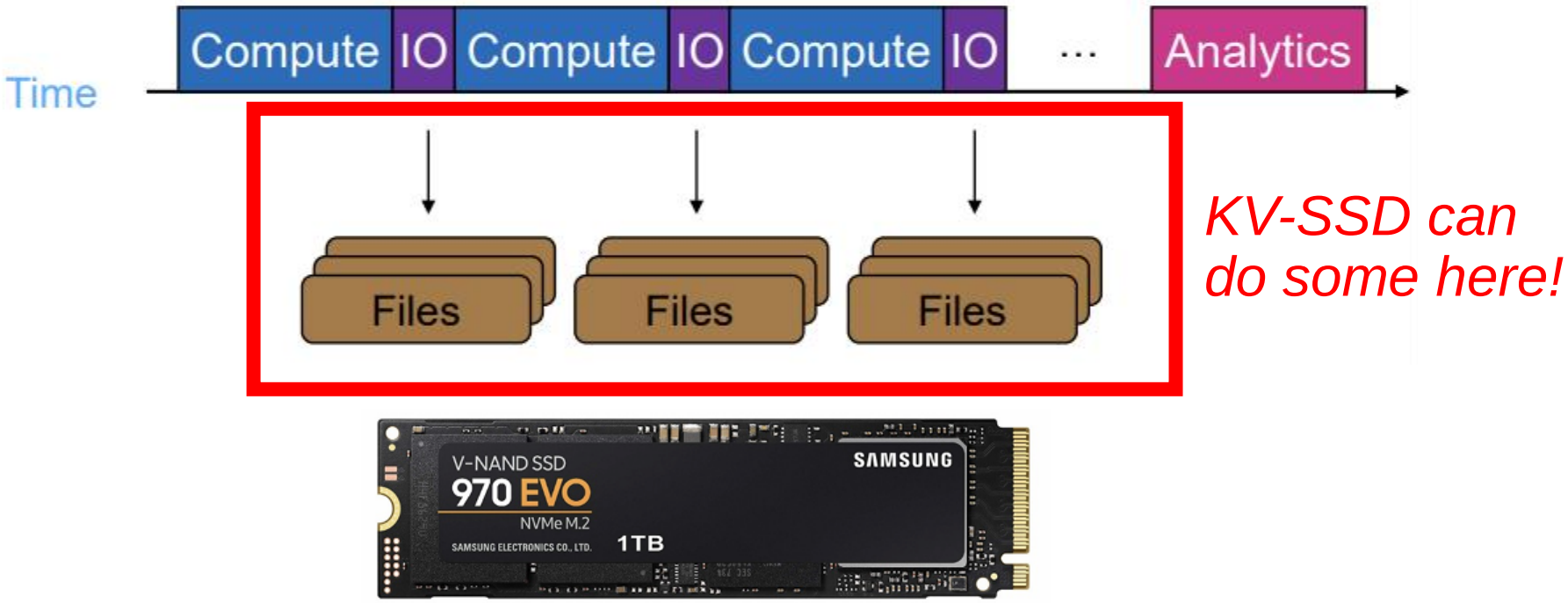
Target Scenarios of KV-CSD

- The I/Os are performed **through File Systems**, and data are stored as one big or many small files per timestep
  - And the data are typically accompanied by metadata that describes data (type, dimension, etc)



Target Scenarios of KV-CSD

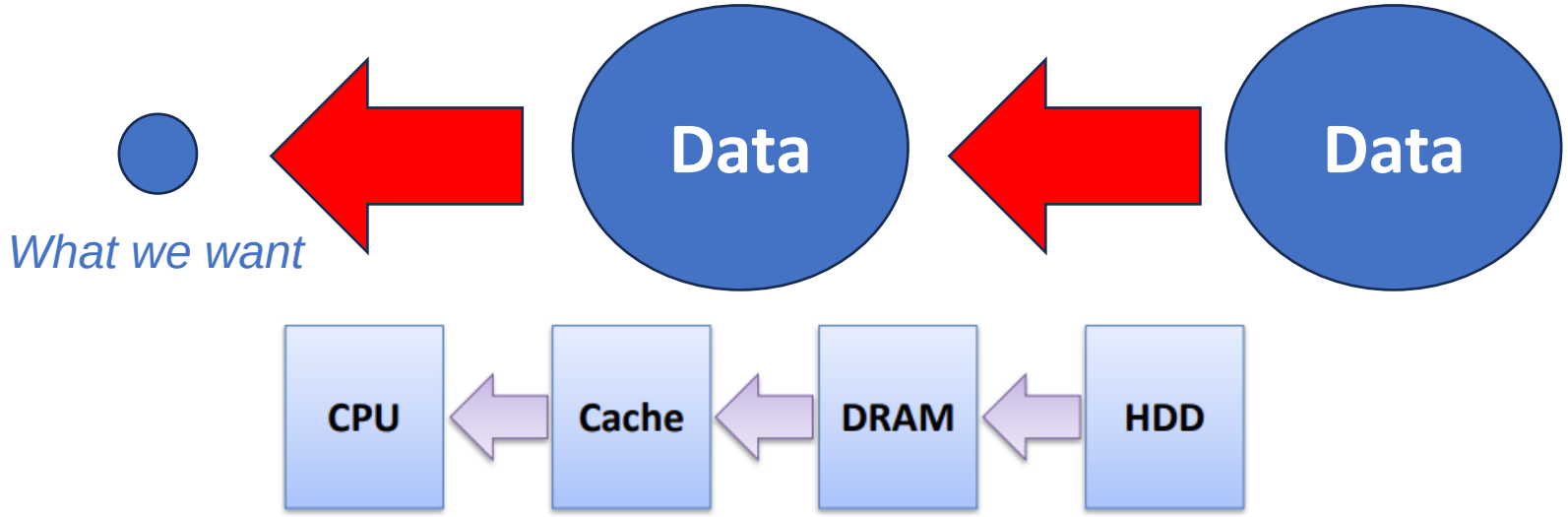
- The I/Os are performed **through File Systems**, and data are stored as one big or many small files per timestep
  - And the data are typically accompanied by metadata that describes data (type, dimension, etc)



KV-SSD can do some here!

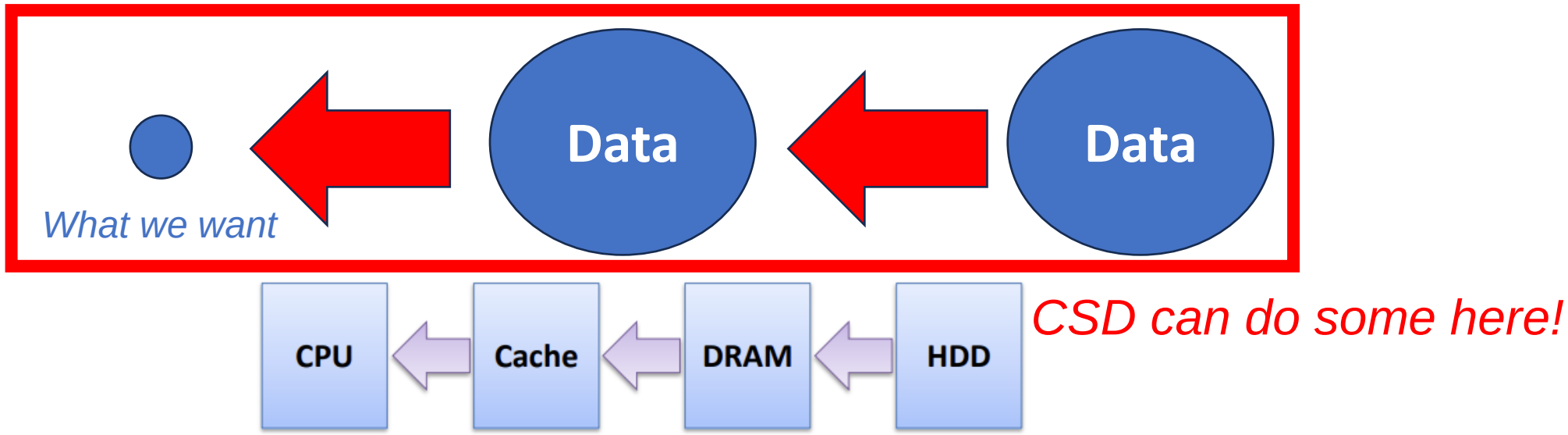
Target Scenarios of KV-CSD

- The applications issue queries on stored data, scanning them and performing computations (join, filter, sort, aggregate) on scanned (loaded onto host's memory) data to get what they really want
  - **Queries often read more data than necessary!**



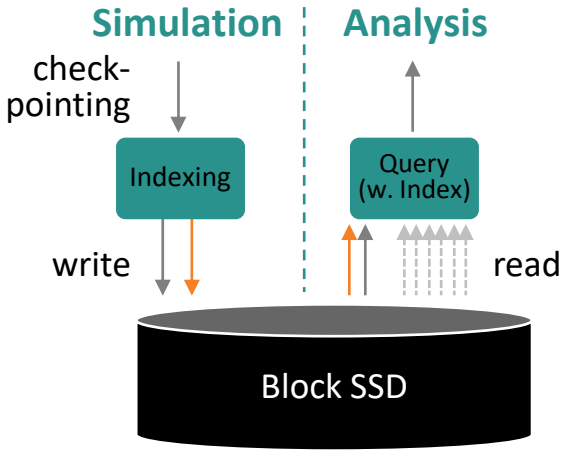
Target Scenarios of KV-CSD

- The applications issue queries on stored data, scanning them and performing computations (join, filter, sort, aggregate) on scanned (loaded onto host's memory) data to get what they really want
  - **Queries often read more data than necessary!**



KV-CSD as Optimal Storage Device

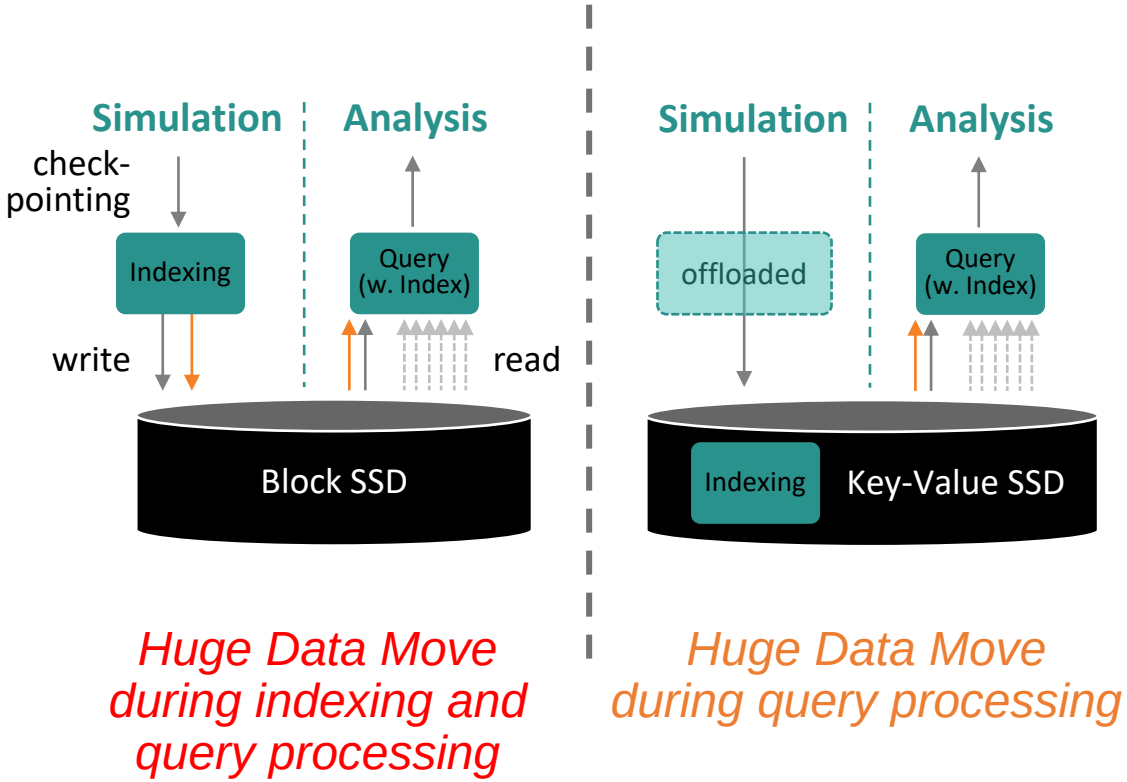
- A concept of KV-SSD+CSD can reduce data movement significantly and accelerate key-value store operation & query performance



Huge Data Move during indexing and query processing

KV-CSD as Optimal Storage Device

- A concept of KV-SSD+CSD can reduce data movement significantly and accelerate key-value store operation & query performance

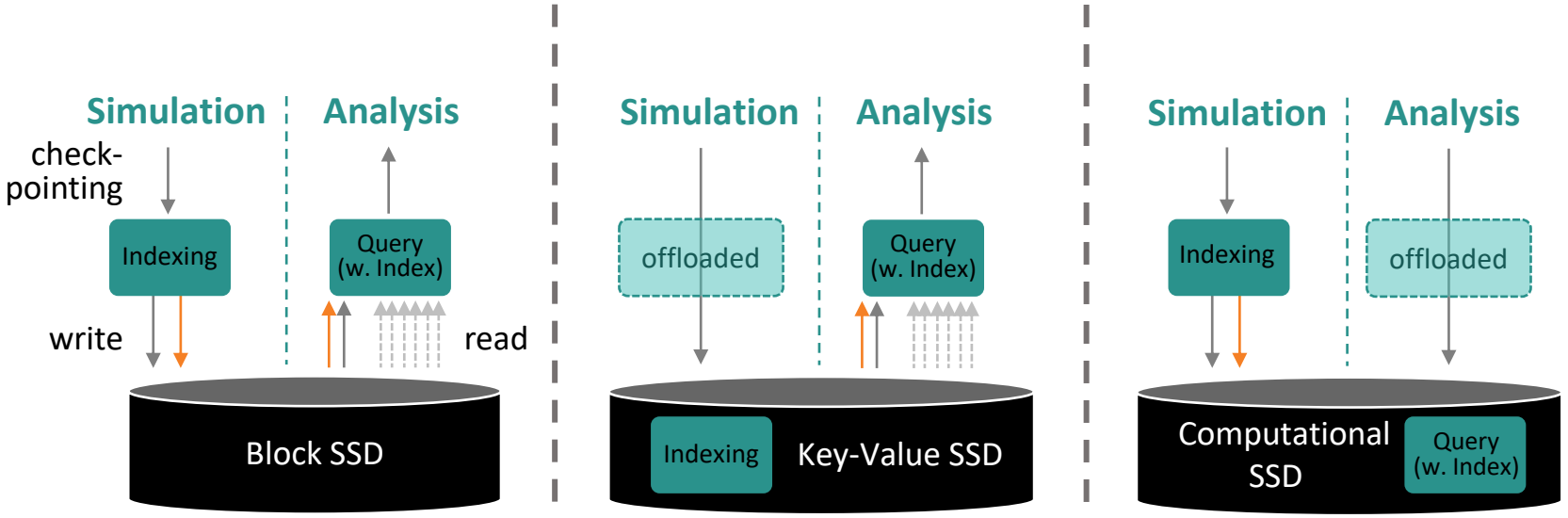


Huge Data Move during indexing and query processing

Huge Data Move during query processing

KV-CSD as Optimal Storage Device

- A concept of KV-SSD+CSD can reduce data movement significantly and accelerate key-value store operation & query performance



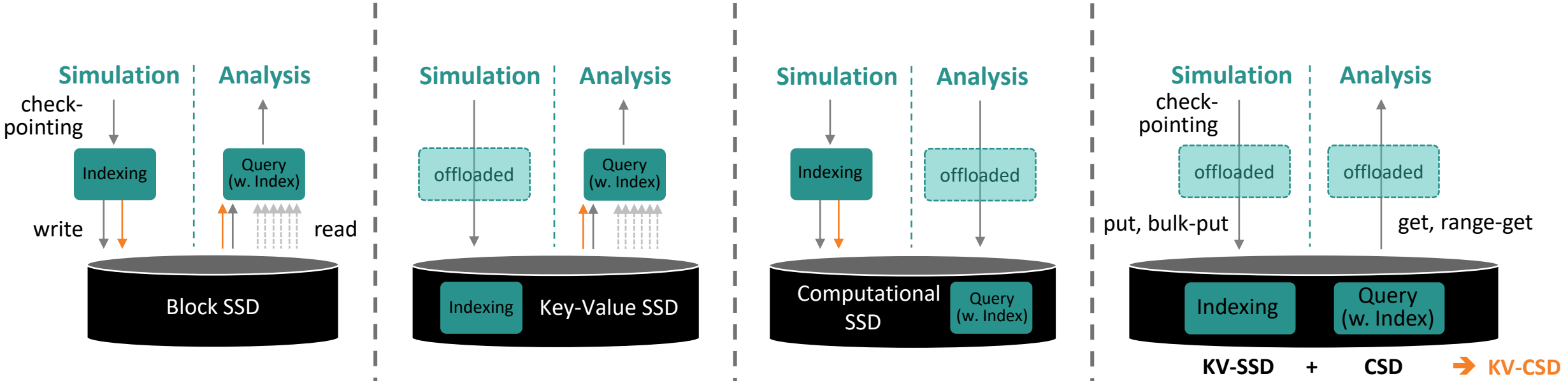
Huge Data Move during indexing and query processing

Huge Data Move during query processing

Huge Data Move during indexing

KV-CSD as Optimal Storage Device

- A concept of KV-SSD+CSD can reduce data movement significantly and accelerate key-value store operation & query performance



Huge Data Move during indexing and query processing

Huge Data Move during query processing

Huge Data Move during indexing

Optimal Data Move

KV-CSD as Optimal Storage Device

Reduce data movement between a host system and a storage system

- Performs data analytics where data reside (near-data processing)
- Efficient data movement increase applications' performance

Save host system resources

- Push-down query execution can save host CPU & memory resources
- It is suitable for heterogeneous and disaggregation paradigm in modern data centers

High interoperability & flexibility

- Based on standard storage interface (e.g., NVMe) and data format (e.g., Apache Arrow & Substrait)
- Break away from fixed and limited pushdown functionality

KV-SSD, KV-CSD Call to Action

- There are still many challenges that KV-SSD/CSD needs to address
  - High cost → more than 200,000₩?
    (One of high-end SSDs, Samsung 990 PRO M.2 NVMe SSD, is currently selling for around 200,000 won (2024))
  - Optimal designs for KV-SSD/CSD
  - Implicitly-assumed-block-interface across storage layers
    (e.g., NVMe protocol's DMA transaction mechanism)
  - Integration with existing storage systems
    (e.g., RocksDB, PrestoDB, Ceph, FS & FDFS)
  - Combining with other next-generation devices
    (e.g., ZNS-SSD, CXL Memory, NVRAM, DPU)

Thank You ☺

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