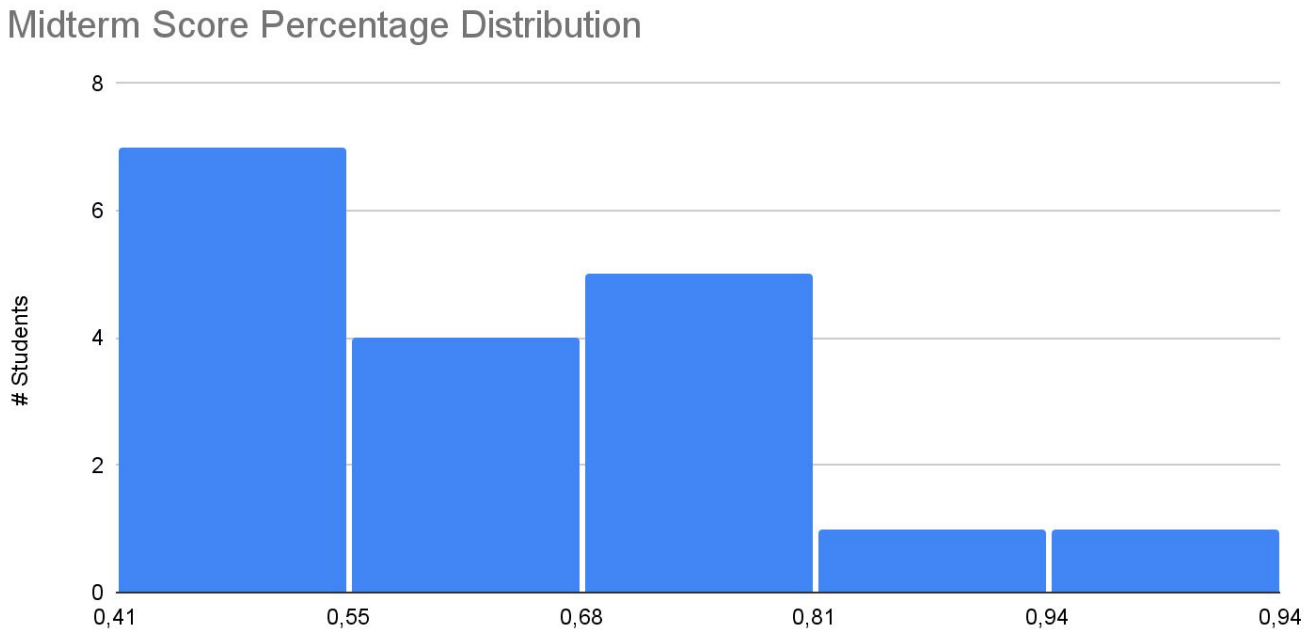
CS 477:

Advanced Operating Systems

Flash Memory & Today's LFS

Administrivia

Midterm marks distribution

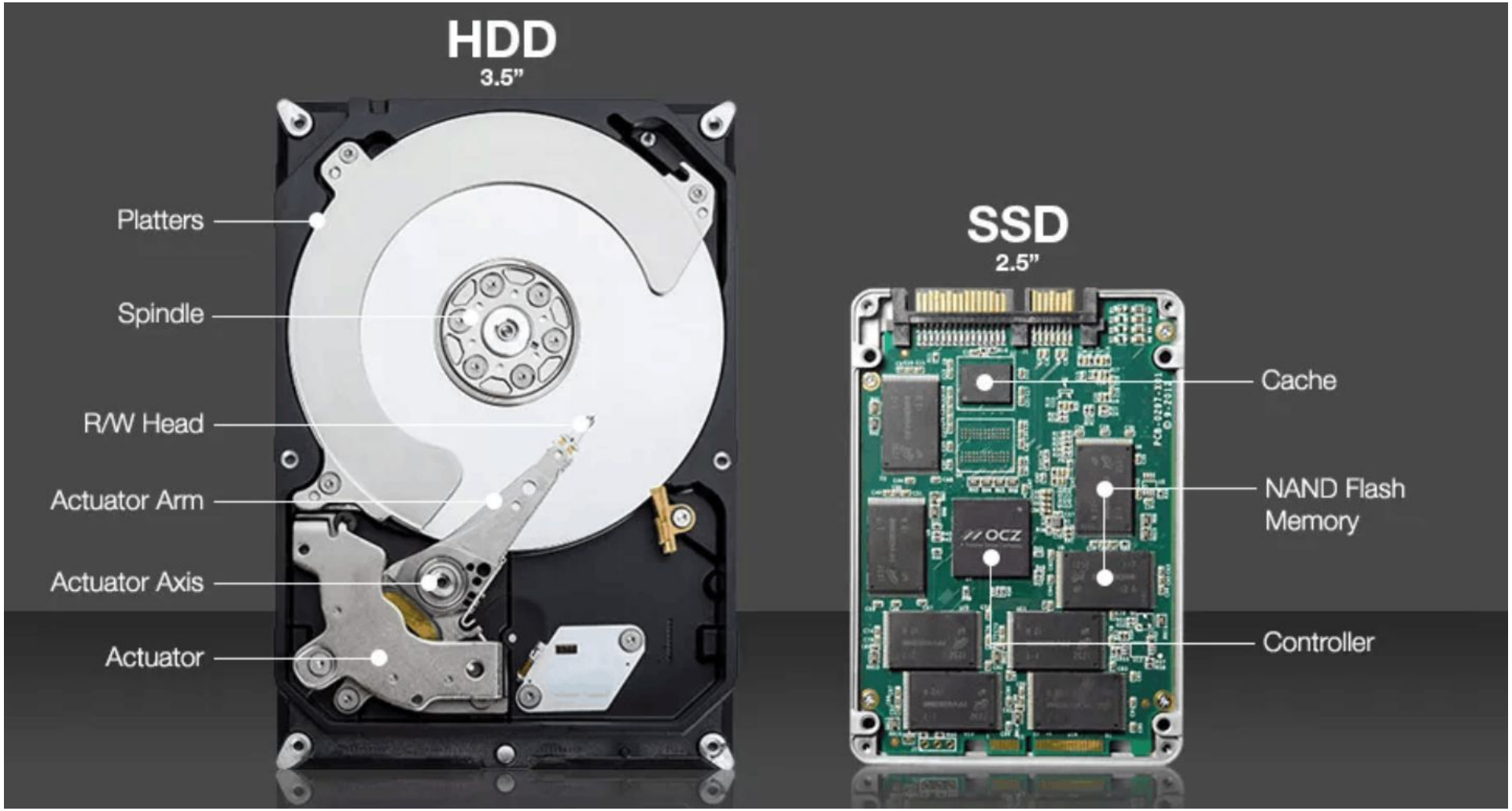


	Min	Avg	Max
Percentage	41%	62%	94%

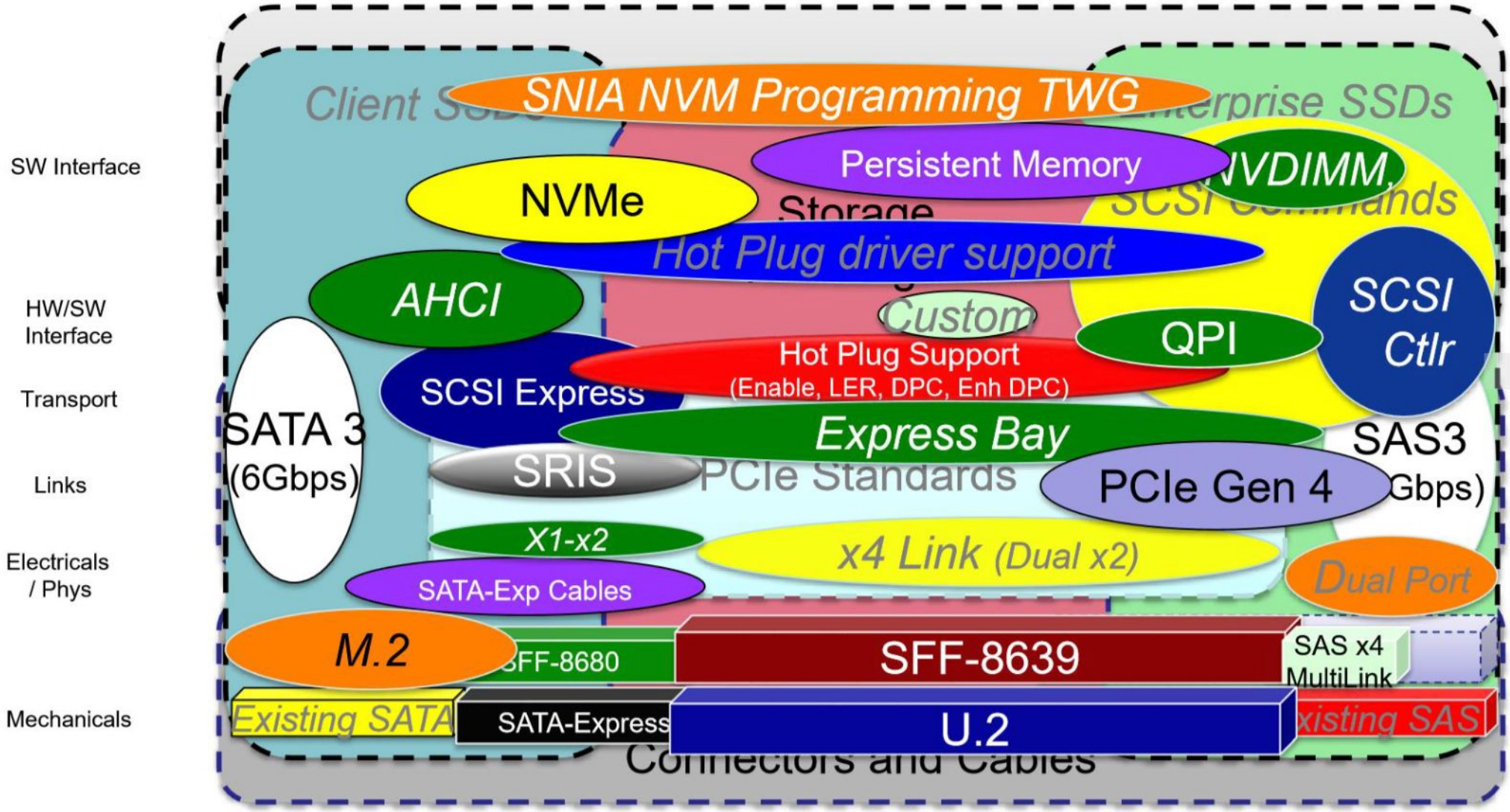
This week

- Solid state drives (SSDs)
  - Flash memory basics
- SSD types
  - Zoned namespace SSDs (ZNS)
  - Key-value SSD (KVSSD)
  - Computational storage (CSD)
- F2FS

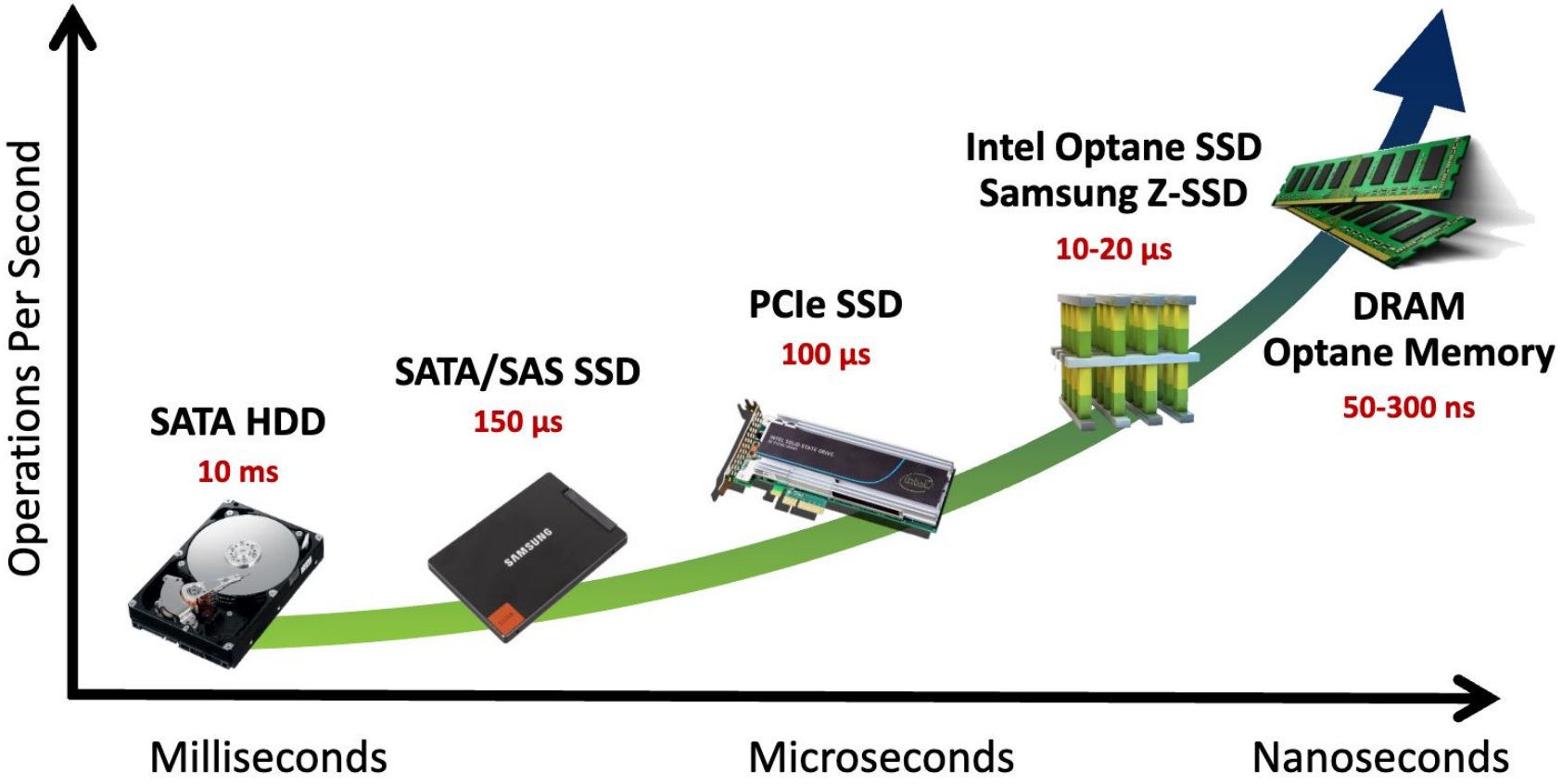
SSD vs. HDD


HDD
3.5”
Platters
Spindle
R/W Head
Actuator Arm
Actuator Axis
Actuator
SSD
2.5”
Cache
NAND Flash Memory
Controller
OCZ

Storage interfaces

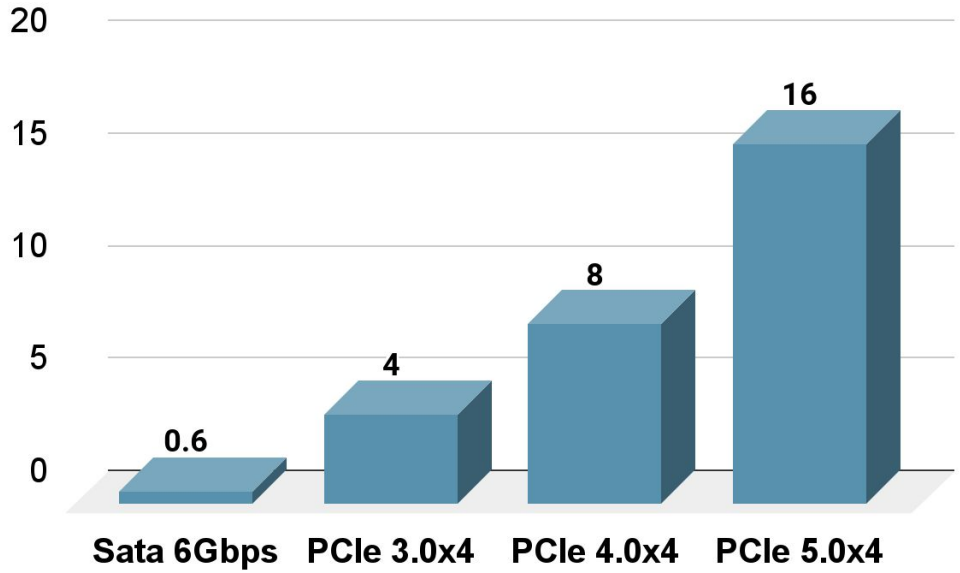


Operations inching closer to the processor speed



NVMe (NVM Express)

- The industry standard interface for high-performance NVM storage
  - NVMe 1.0 in 2011 by NVM Express Workgroup
  - NVMe 1.2 in 2014
- PCIe-based
- Lower latency
  - Direct connection to CPU
- Scalable bandwidth
  - 1 GB/s per lane (PCI Gen 3); 2 for PCIe Gen 4



NVMe SSD form factors

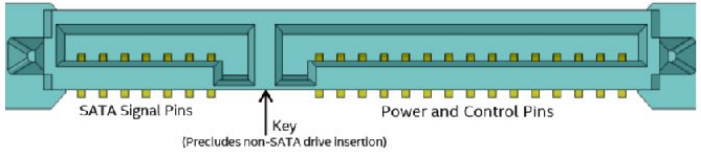
Add-in-card (AIC)

2242 2260 2280

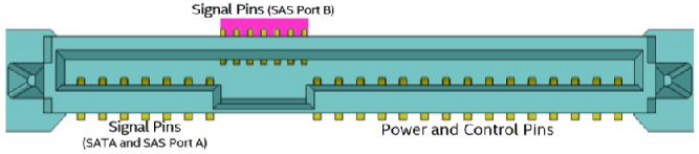
M.2 (PCIe: Up to x4)

U.2
(SFF-8639:
Up to x4)

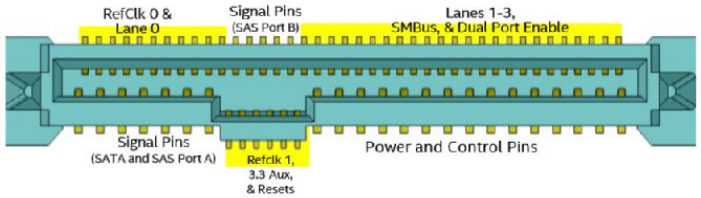
SATA



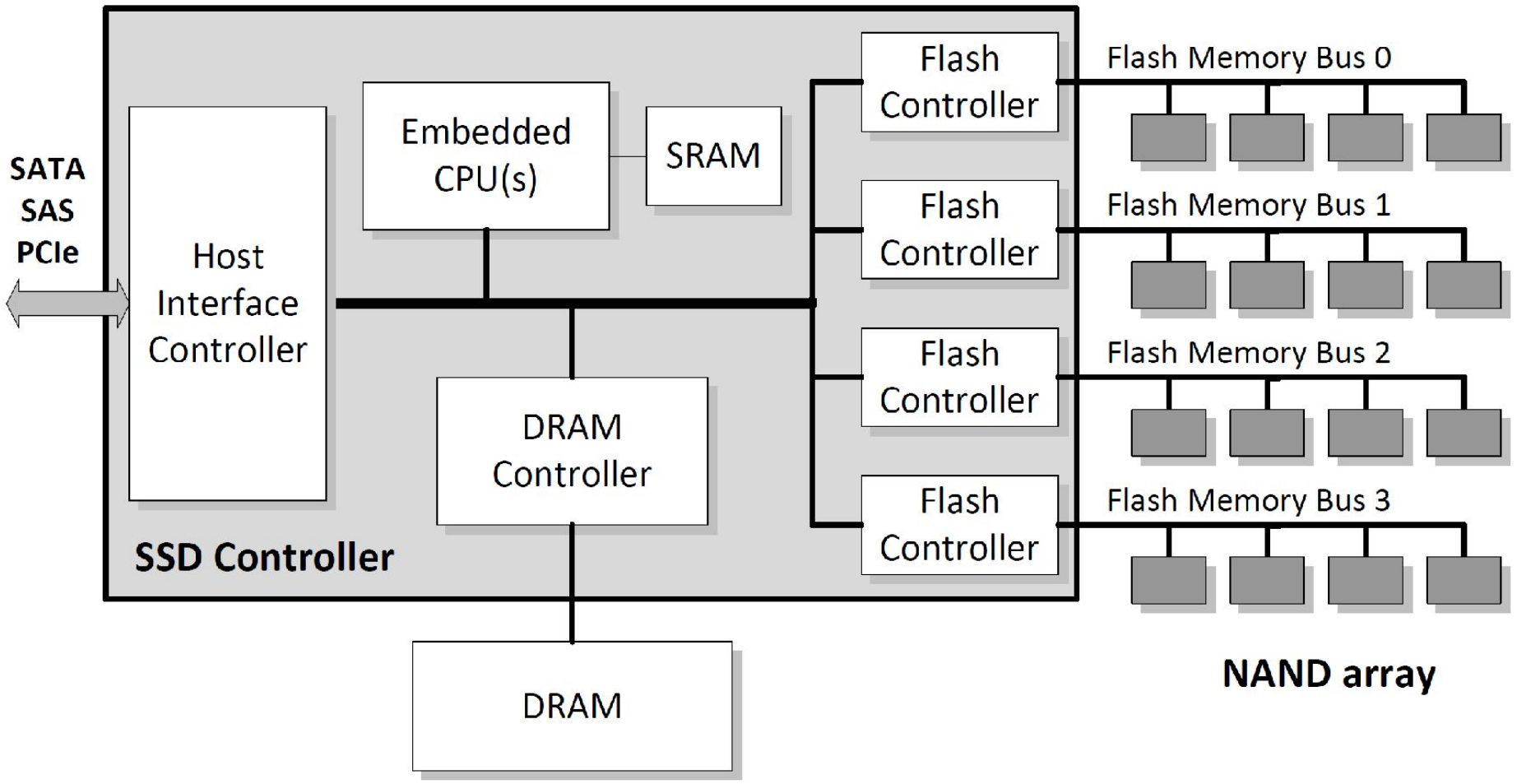
SAS



U.2



SSD internals



The unwritten contract

Several assumptions are no longer valid

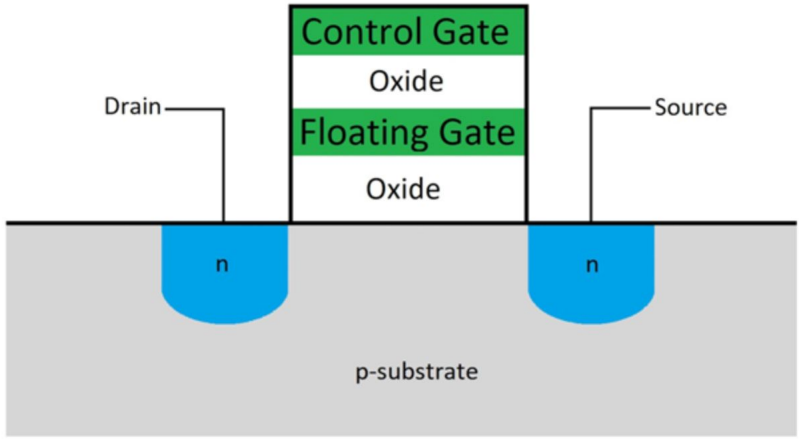
Assumptions	Disks	SSDs
Sequential accesses much faster than random	✓	✗
No write amplification	✓	✗
Little background activity	✓	✗
Media does not wear down	✓	✗
Distant LBNs lead to longer access time	✓	✗

This week

- Solid state drives (SSDs)
  - **Flash memory basics**
- SSD types
  - Zoned namespace SSDs (ZNS)
  - Key-value SSD (KVSSD)
  - Computational storage (CSD)
- F2FS

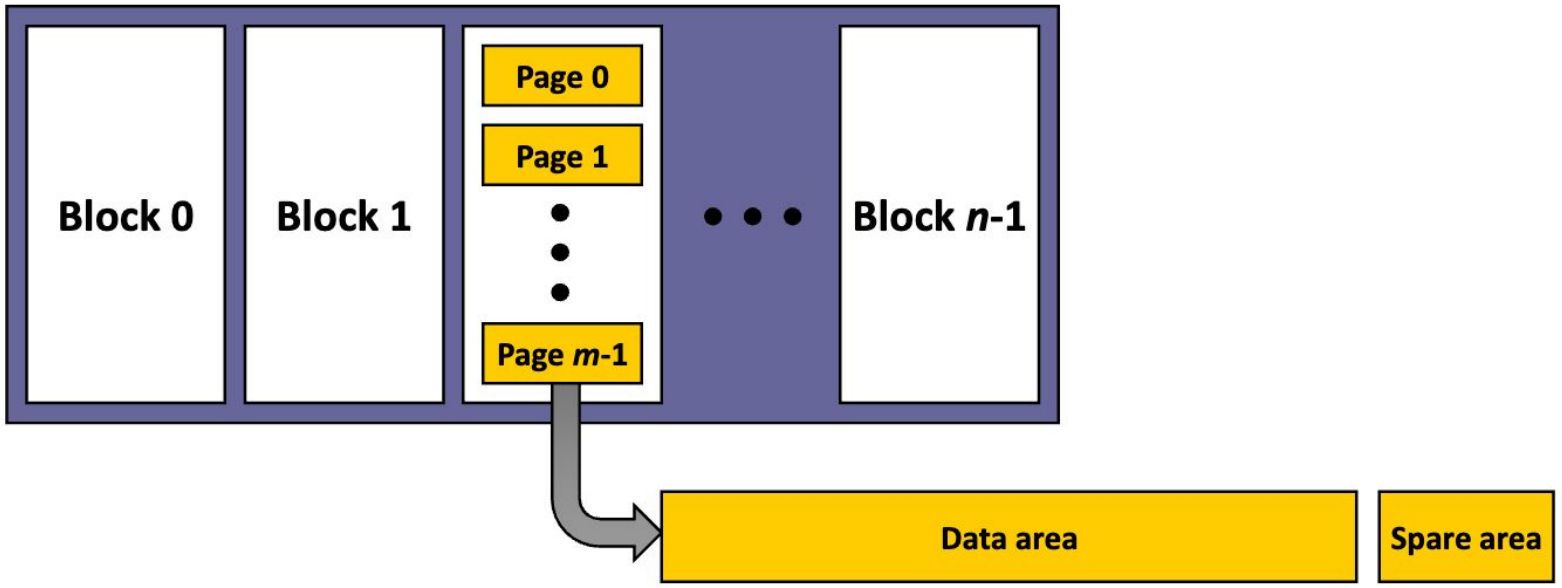
Flash memory basics: NAND transistor

- Data is stored in memory cells
  - Cells have floating-gate transistors that can capture electrons for an extended period, but not indefinitely
  - Chip pushes electrons into the oxide layer and into the “silicone” gate
  - Float gate ensures that electricity keeps circulating even if the power is switched off
  - Floating gate selects the zero or one state based on the memory’s state before it was turned off



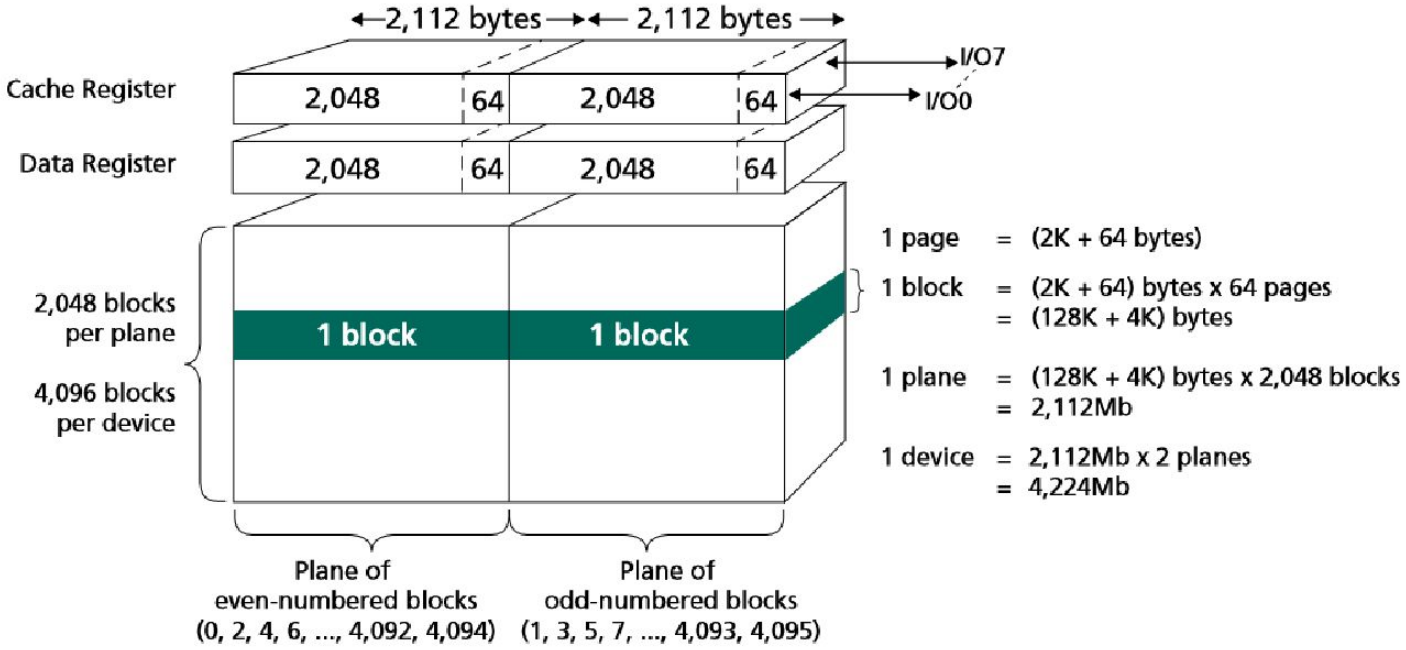
Logical view of NAND flash

- A collection of blocks
- Each block has a number of pages
- The size of a page or block depends on the technology in use



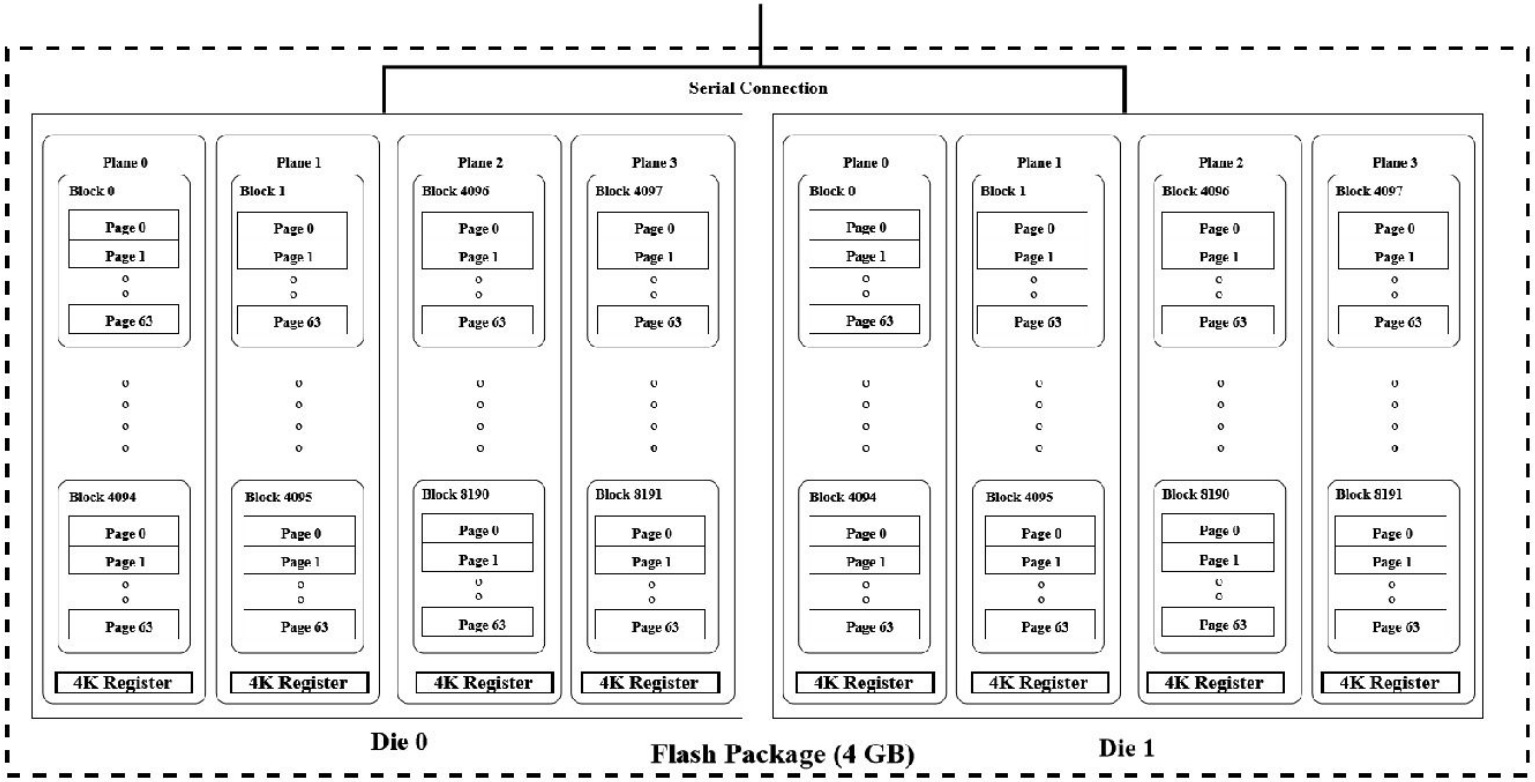
Plane

- Each plane has its own plane register or cache register
- Pages can be reprogrammed or read at once
- Optional feature: 1, 2, 4, 8 .. planes



Chip/die

- Each chip has multiple dies (can be stacked)
- + extra circuits, chip enable signal, ready/busy signal



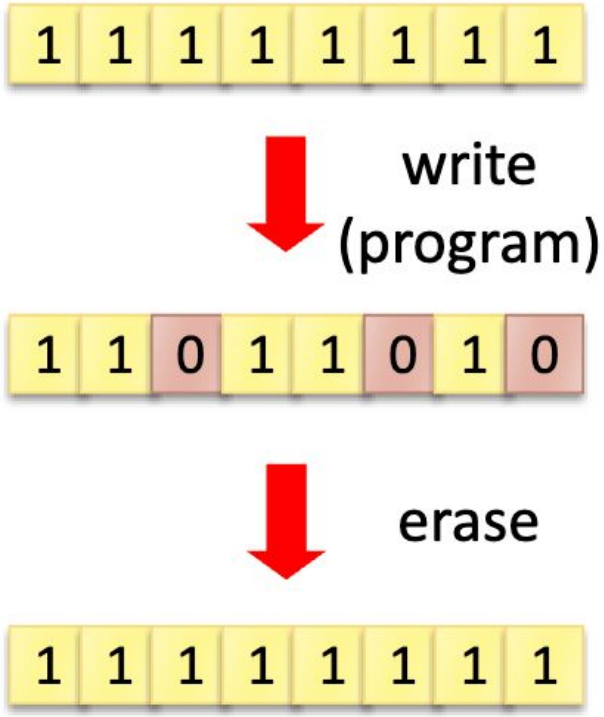
NAND flash types

- SLC NAND: Single level cell (1 bit/cell)
  - Most expensive, used in servers
- MLC NAND: multi-level cell (2 bits/cell)
  - Used where endurance is not important
- TLC NAND: triple-level cell (3 bits/cell)
  - Cheapest option on the market
- QLC NAND: quad-level cell (4 bits/cell)

https://www.kingston.com/en/blog/pc-performance/difference-between-slc-mlc-tlc-3d-nand#:~:text=MLC%20has%20a%20higher%20data,lower%20endurance%20compared%20to%20SLC.

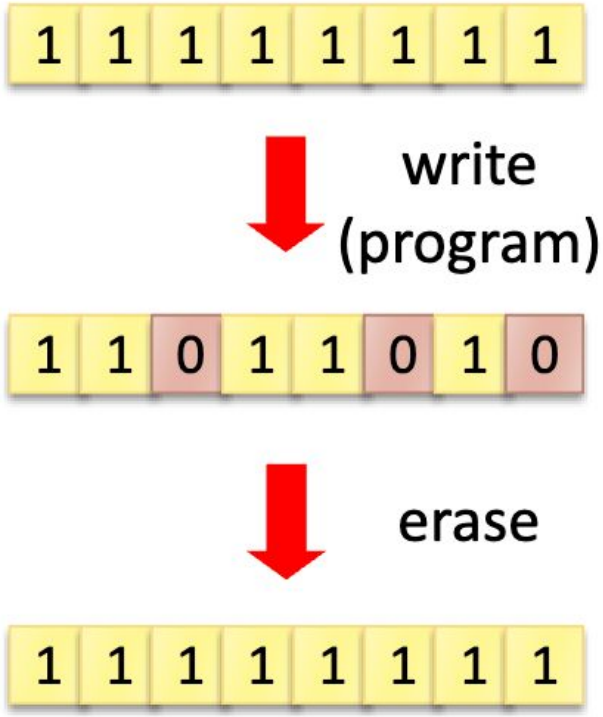
Flash memory characteristics: erase-before-write

- In-place update (overwrite) not allowed
- Pages must be erased before programming new data
- The erase unit is much larger than the read/write unit
  - Read/write unit: page (4KB, 8KB, 16KB, …)
  - Erase unit: block (64-512 pages)
- Handling live pages in a block marked for deletion?



Flash memory characteristics: erase-before-write

- In-place update (overwrite) not allowed
- Pages must be erased before programming new data
- The erase unit is much larger than the read/write unit
  - Read/write unit: page (4KB, 8KB, 16KB, …)
  - Erase unit: block (64-512 pages)
- Handling live pages in a block marked for deletion?
  - Copy the live pages to a spare area and then mark the block for garbage collection



Flash memory characteristics: limited lifetime

- NAND flash blocks has limited programmed and erased (P/E) cycle:
  - SLC: 50,000 – 100,000
  - MLC: 3,000 – 10,000
  - eMLCs (enterprise MLCs): 10,000 – 30,000
  - TLCs: 1,000 – 3,000
  - QLC: 100 – 1,000
- High voltage applied to cell degrades the oxide layer
  - Electrons trapped in oxide; break down of the oxide layer

Flash memory characteristics: asymmetric rd/wr lat

- Reading a page is faster than programming it
- Usually more than 10x
  - MLC: read 45us, program 1350us, erase 4ms
- Programming a page should go through multiple steps of program and verify steps
- With shrinking technology, read/write latency tends to increase
- MLC and TLC make it even worse

How does storage stack interact with SSDs?

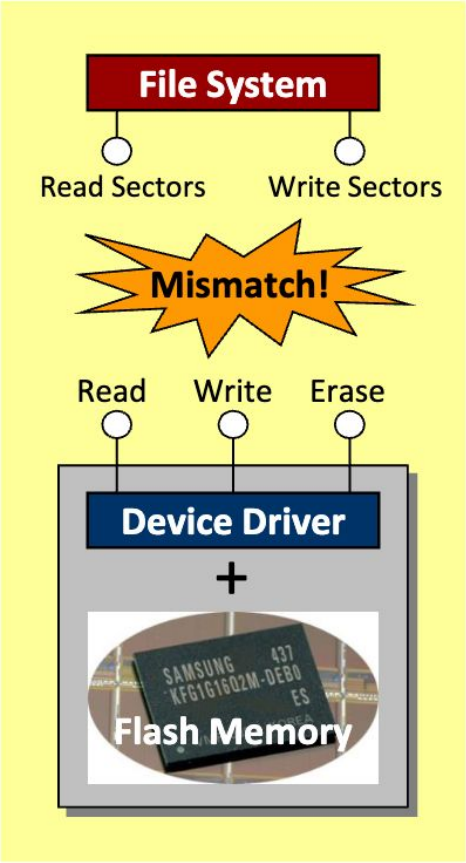
Storage abstraction

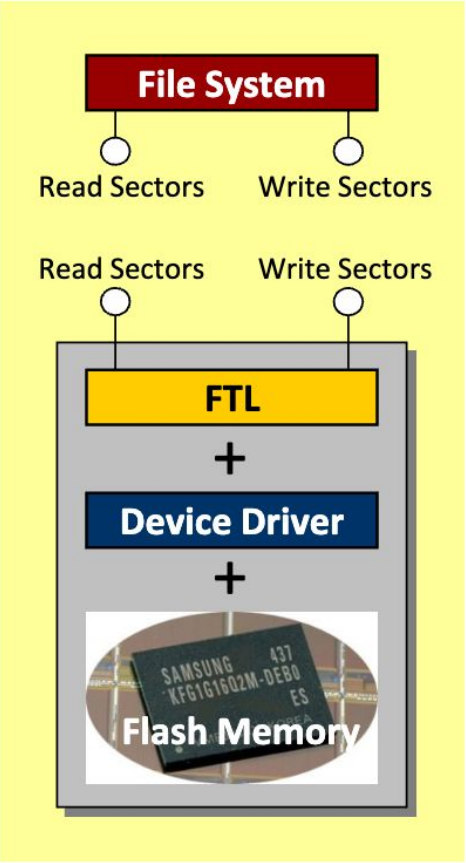
- Abstraction given by block device drivers:

- Operations:
  - Identify(): returns N
  - Read (start sector #, # of sectors, buffer address)
  - Write (start sector #, # of sectors, buffer address)

FTL: flash translation layer

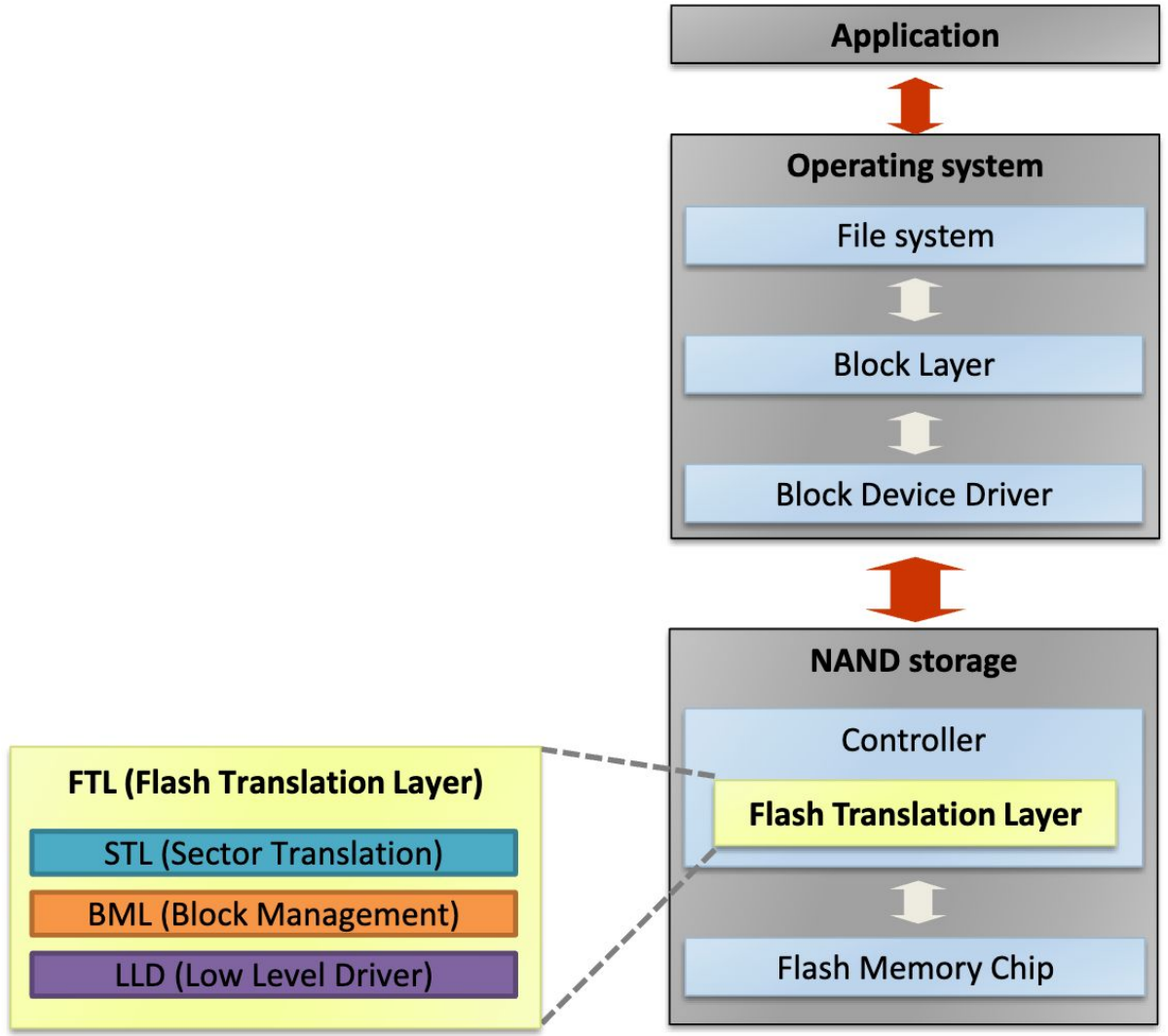
A flash layer that makes NAND flash fully emulate the traditional block devices



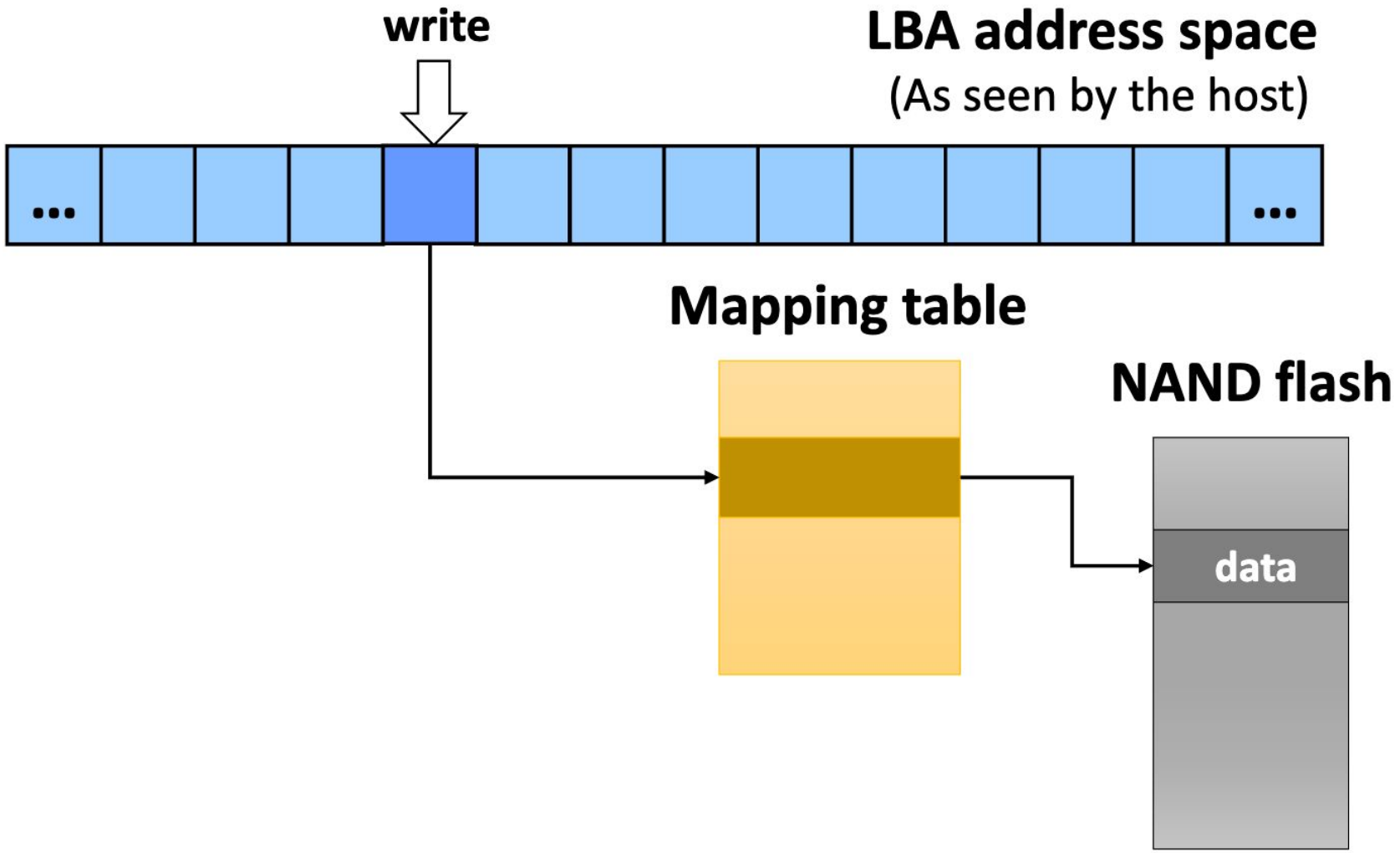


NAND flash types

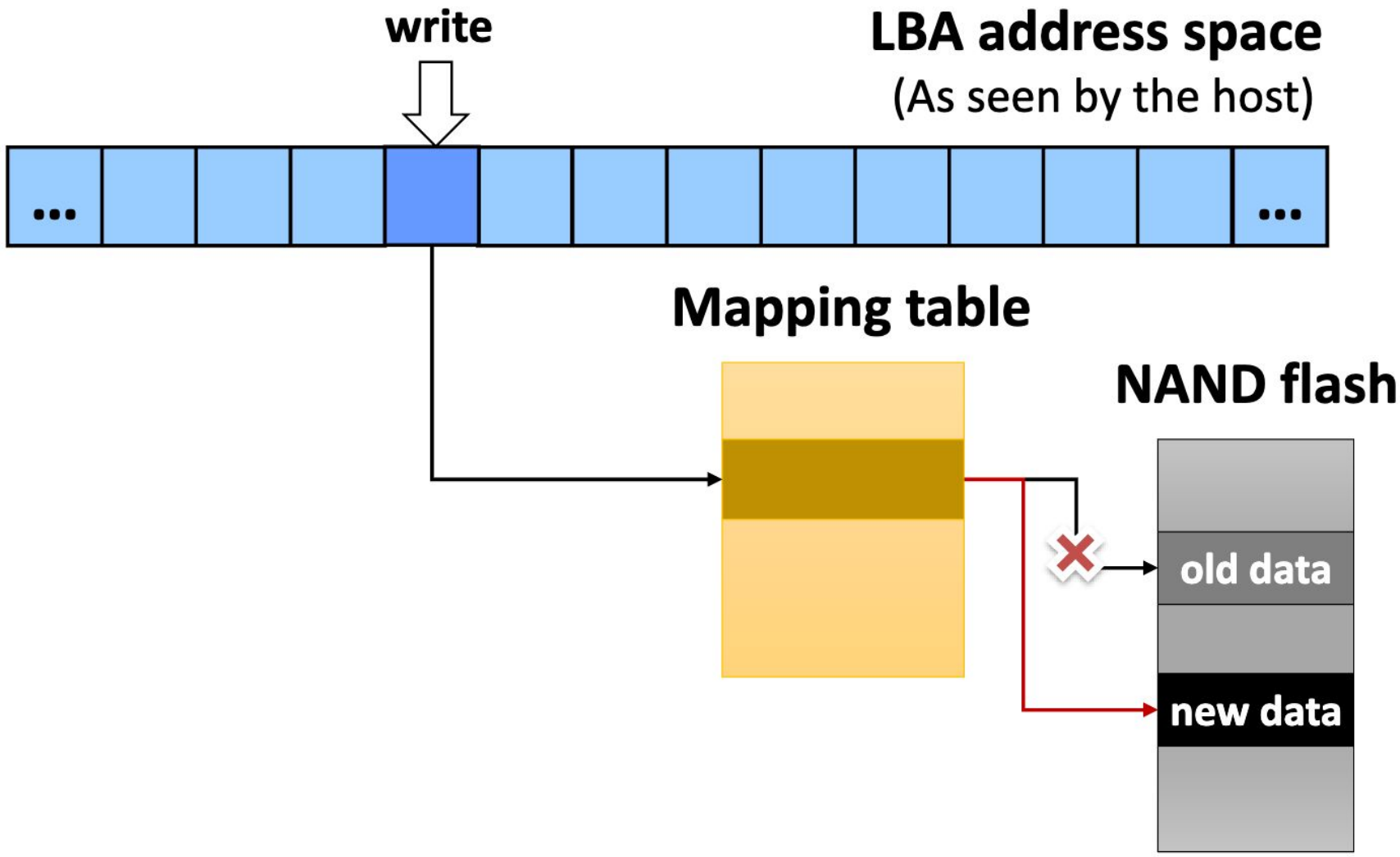
- Sector translation layer
  - Address mapping
  - Garbage collection
  - Wear levelling
- Block management layer
  - Bad block management
  - Error handling
- Low level driver
  - Flash interface



Address mapping



Address mapping: cannot overwrite the same page



Mapping scheme

- Page mapping
  - Fine-granularity page-level mapping table
  - Huge amount of memory space required for the map table
- Block mapping
  - Coarse-granular block-level map table
  - Small amount of memory space required for the map table
- Hybrid mapping
  - Use both page-level and block-level map tables
  - Higher algorithm complexity

Garbage collection

- Garbage collection (GC)
  - Eventually, FTL will run out of blocks to write to
  - GC reclaims free space
  - Actual GC procedure depends on the mapping table
- GC in page-mapping FTL
  - Select victim block(s)
  - Copy all valid pages of victim block(s) to free block
  - Erase victim block(s)
  - Note: At least one free block should be reserved for GC

Write amplification

- Ratio of data written to flash to data written on host
- Write amplification factor (WAF)

  = Bytes written to **flash** / bytes written from **host**

  = (Bytes written from host + bytes written during GC) / bytes written from host
- Generally, WAF > 1
  - Due to valid page copies made from victim block to free block during GC
  - WAF is one of the metrics showing the efficiency of GC

Victim selection policy: Greedy

- Select a block with largest amount of invalid data
- A block with the min. utilization *u*

  u = # valid pages in a block / # pages in a block
- Pros:
  - Least valid data copying costs
  - Simple
- Cons
  - Performs worse when there is a high locality among writes
  - Does not consider wear leveling

Victim selection policy: Cost-benefit

- Select a block with the maximum

  Benefit / cost = (1 - u)/2u x age

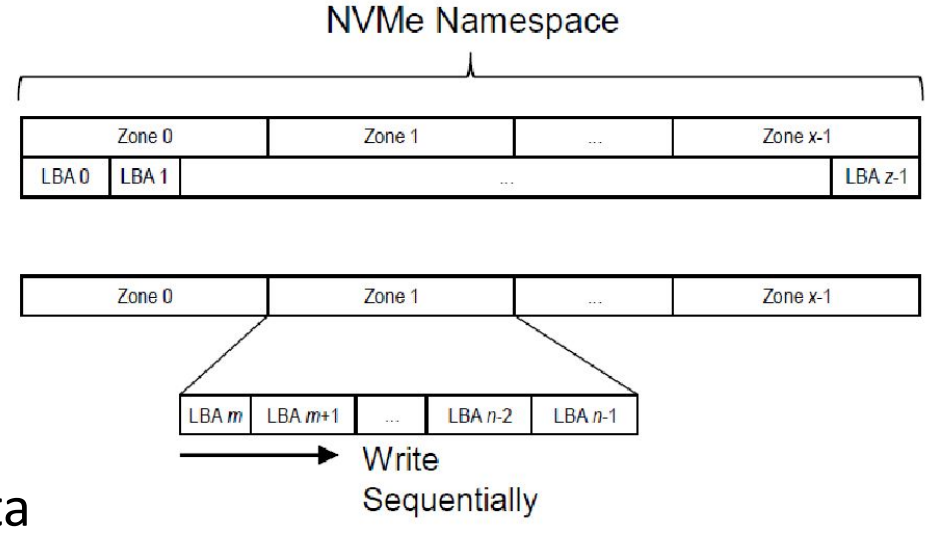
  - u: utilization
  - age: the time since the last modification
- Pros:
  - Performs well with locality
  - Somehow also maintains wear-leveling
- Cons:
  - Computation/data overhead

This week

- Solid state drives (SSDs)
  - Flash memory basics
- **SSD types**
  - **Zoned namespace SSDs (ZNS)**
  - Key-value SSD (KVSSD)
  - Computational storage (CSD)
- F2FS

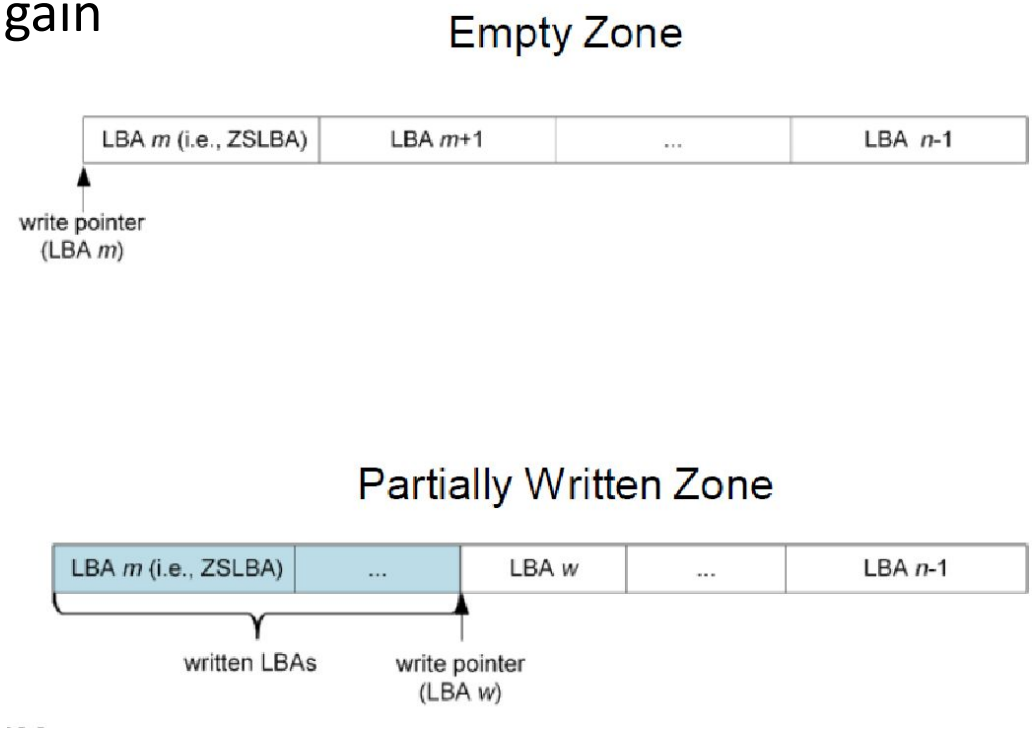
Zoned storage model

- Zones are laid out sequentially in an NVMe namespace
- The zone size is fixed and applies to all zones in the namespace (e.g., 512 MB)
- Inherits the base NVMe command set
  - Built upon the conventional block interface (read, write, flush, and other commands)
  - Adds rules to collaborate with host and device data placement



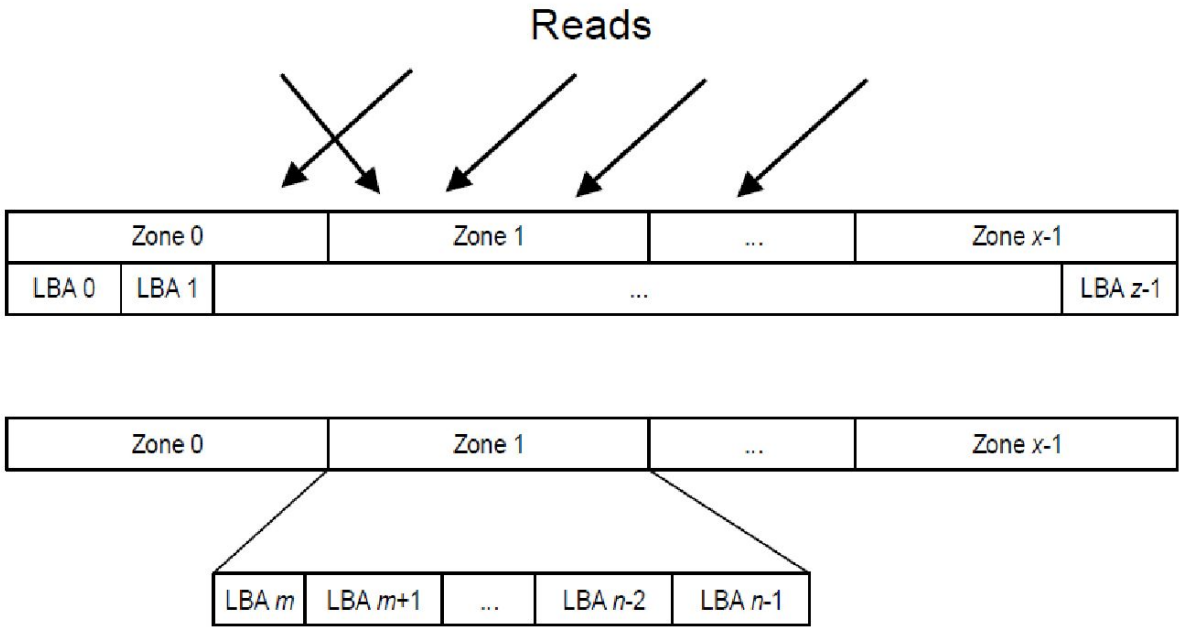
Writing to a zone

- Only supports sequential writing
  - Either write sequentially or reset if written again
- Each zone has a set of associated attributes:
  - Write pointer
  - Zone starting LBA
  - Zone capacity
  - Zone state



Reading from a zone

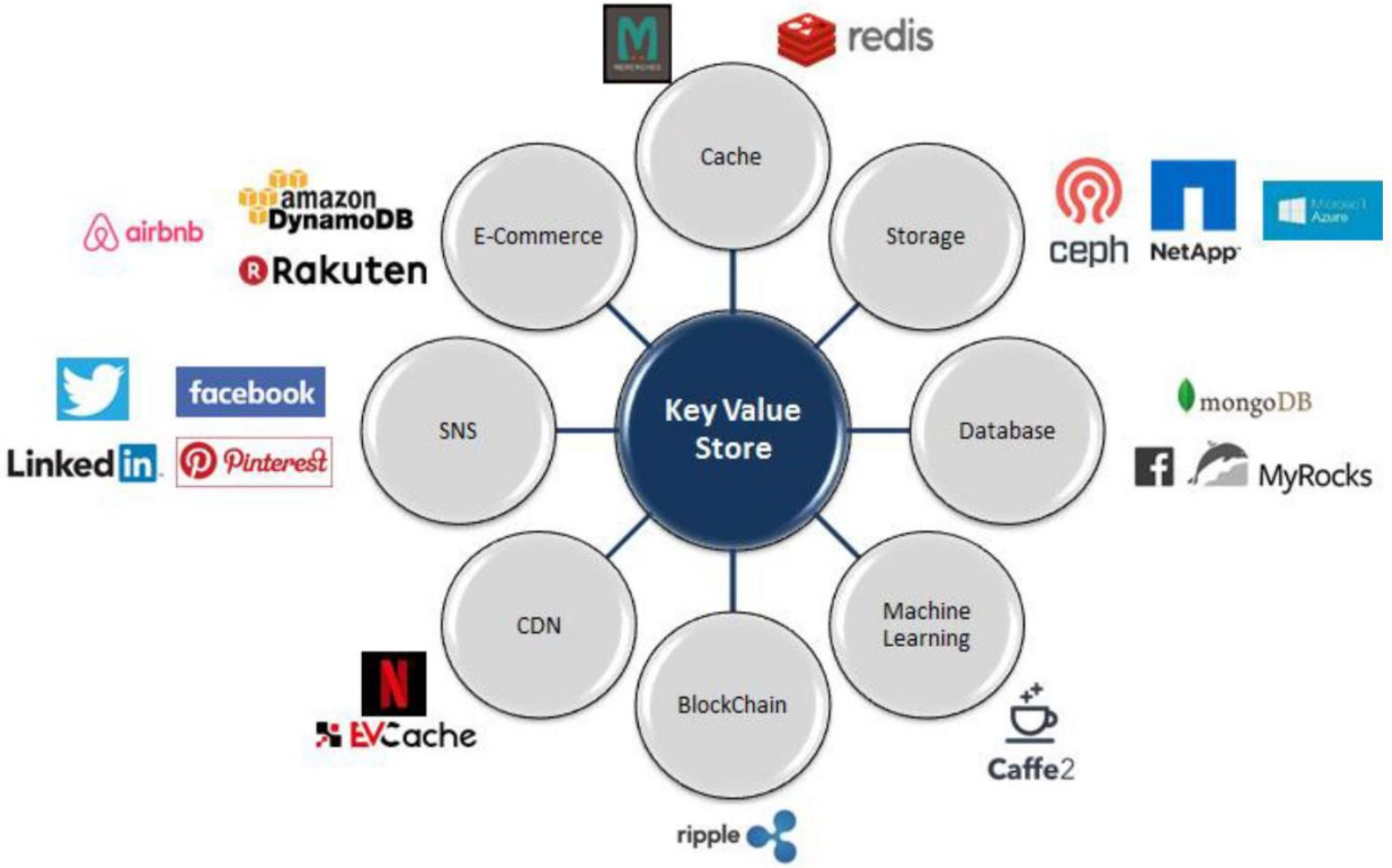
- Writes are required to be sequential within a zone
- Reads may be issued to any LBA within a zone and in any order



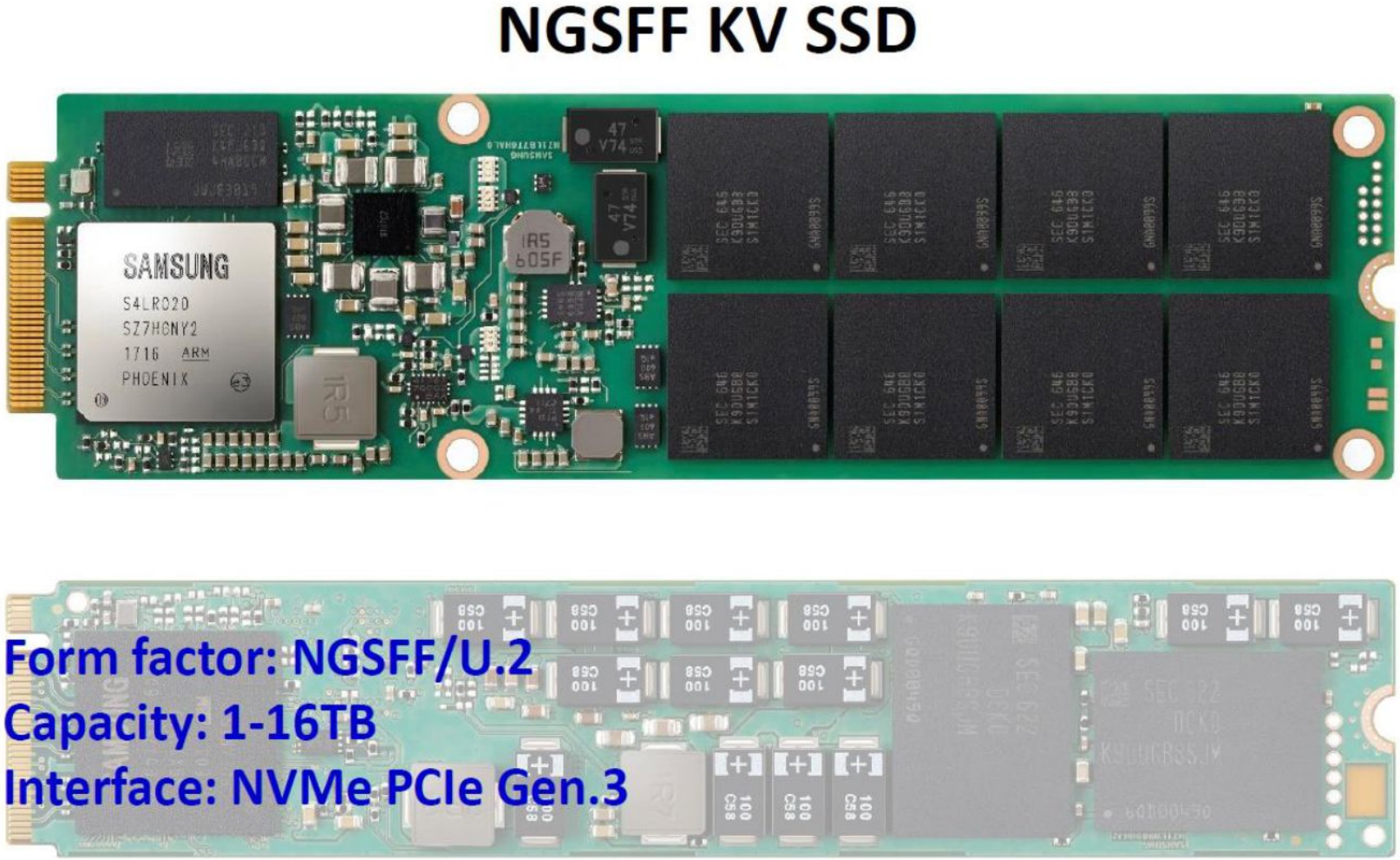
This week

- Solid state drives (SSDs)
  - Flash memory basics
- **SSD types**
  - Zoned namespace SSDs (ZNS)
  - **Key-value SSD (KVSSD)**
  - Computational storage (CSD)
- F2FS

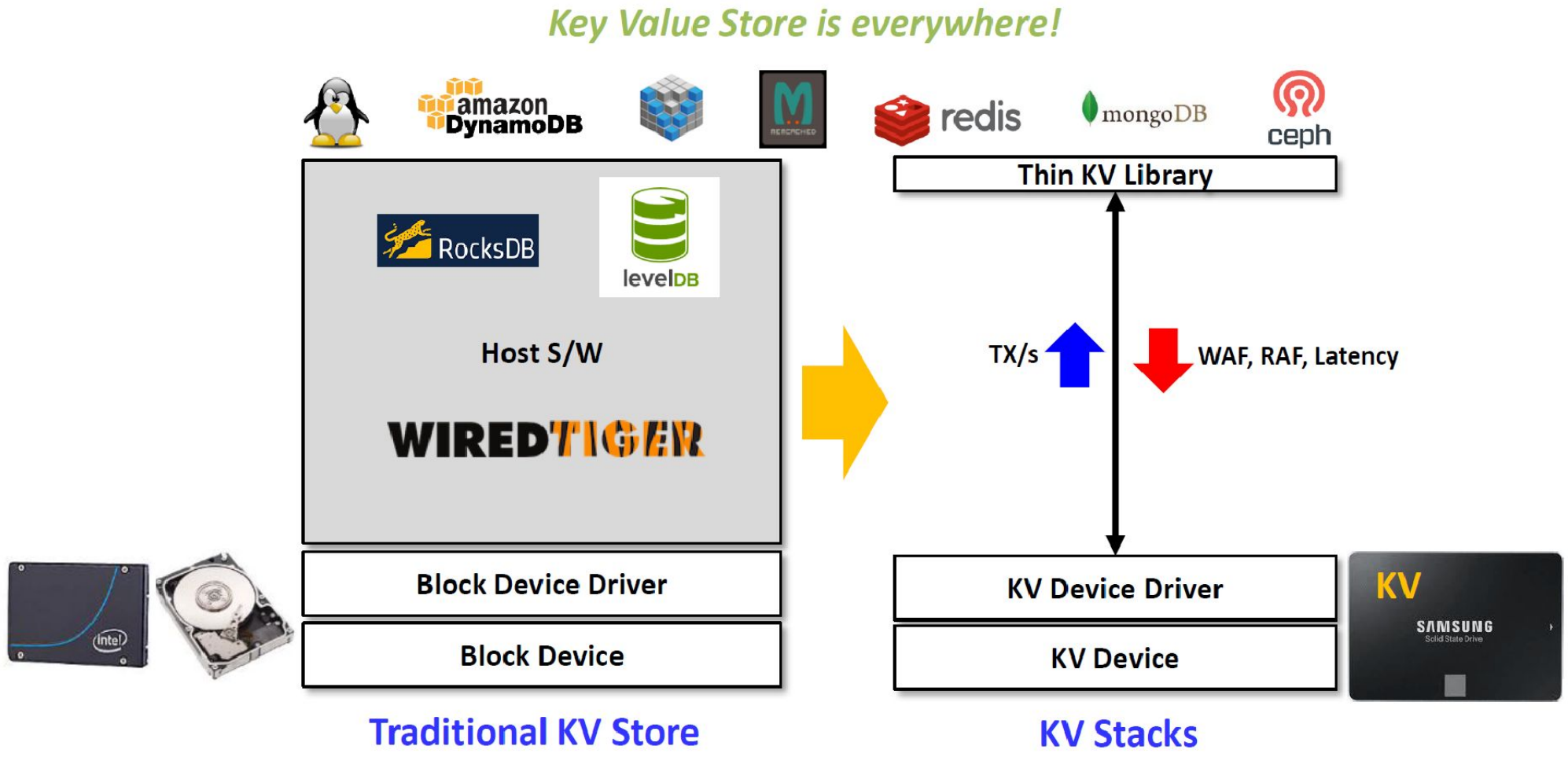
KV stores are commonly used at scale



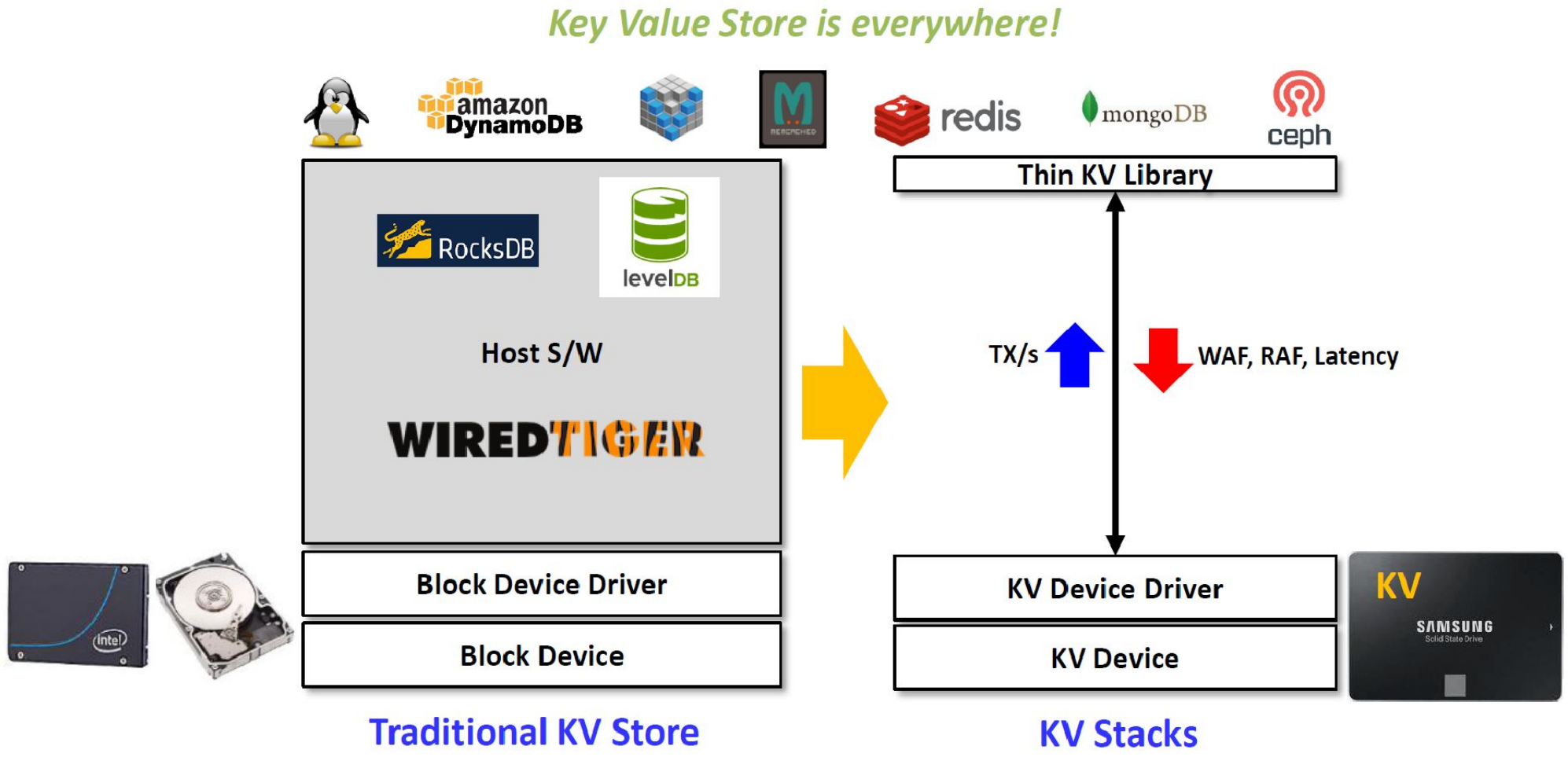
Samsung KVSSD



Key idea

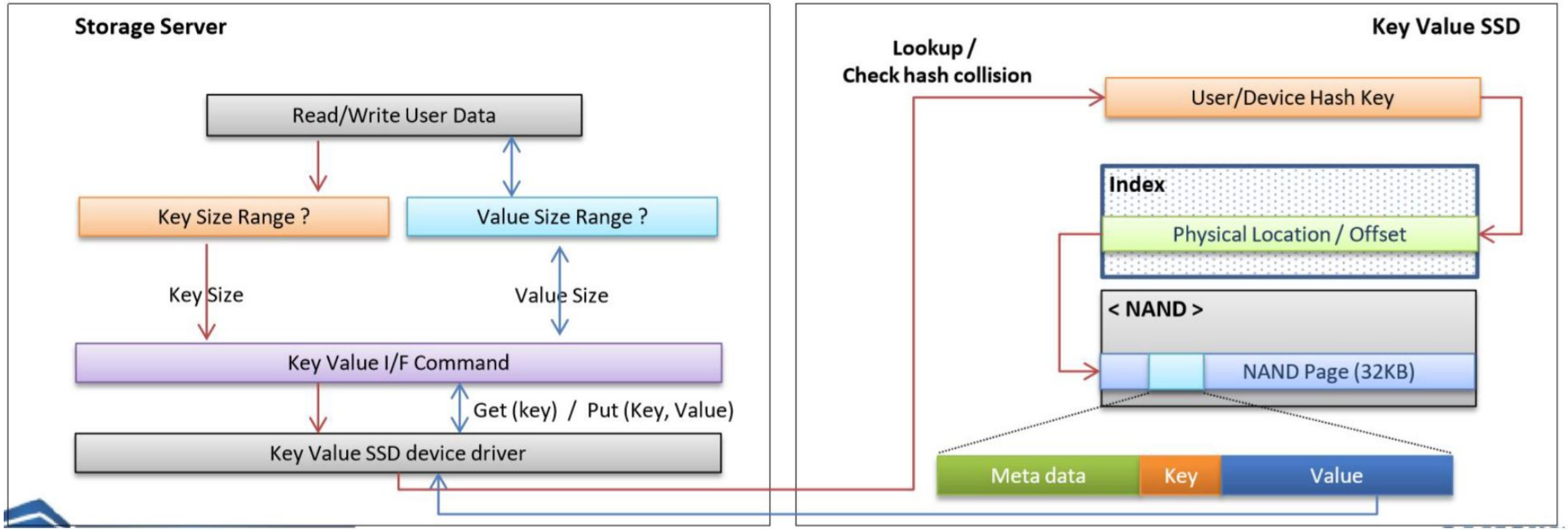

Key Value Store is everywhere!
amazon DynamoDB
redis
mongoDB
ceph
RocksDB
levelDB
Host S/W
WIREDTIGER
Block Device Driver
Block Device
Thin KV Library
TX/s
WAF, RAF, Latency
KV Device Driver
KV Device
KV
SAMSUNG
Traditional KV Store
KV Stacks

Key idea


Key Value Store is everywhere!
amazon DynamoDB
redis
mongoDB
ceph
RocksDB
levelDB
Host S/W
WIREDTIGER
Block Device Driver
Block Device
Thin KV Library
TX/s
WAF, RAF, Latency
KV Device Driver
KV Device
KV
SAMSUNG
Traditional KV Store
KV Stacks

Key idea

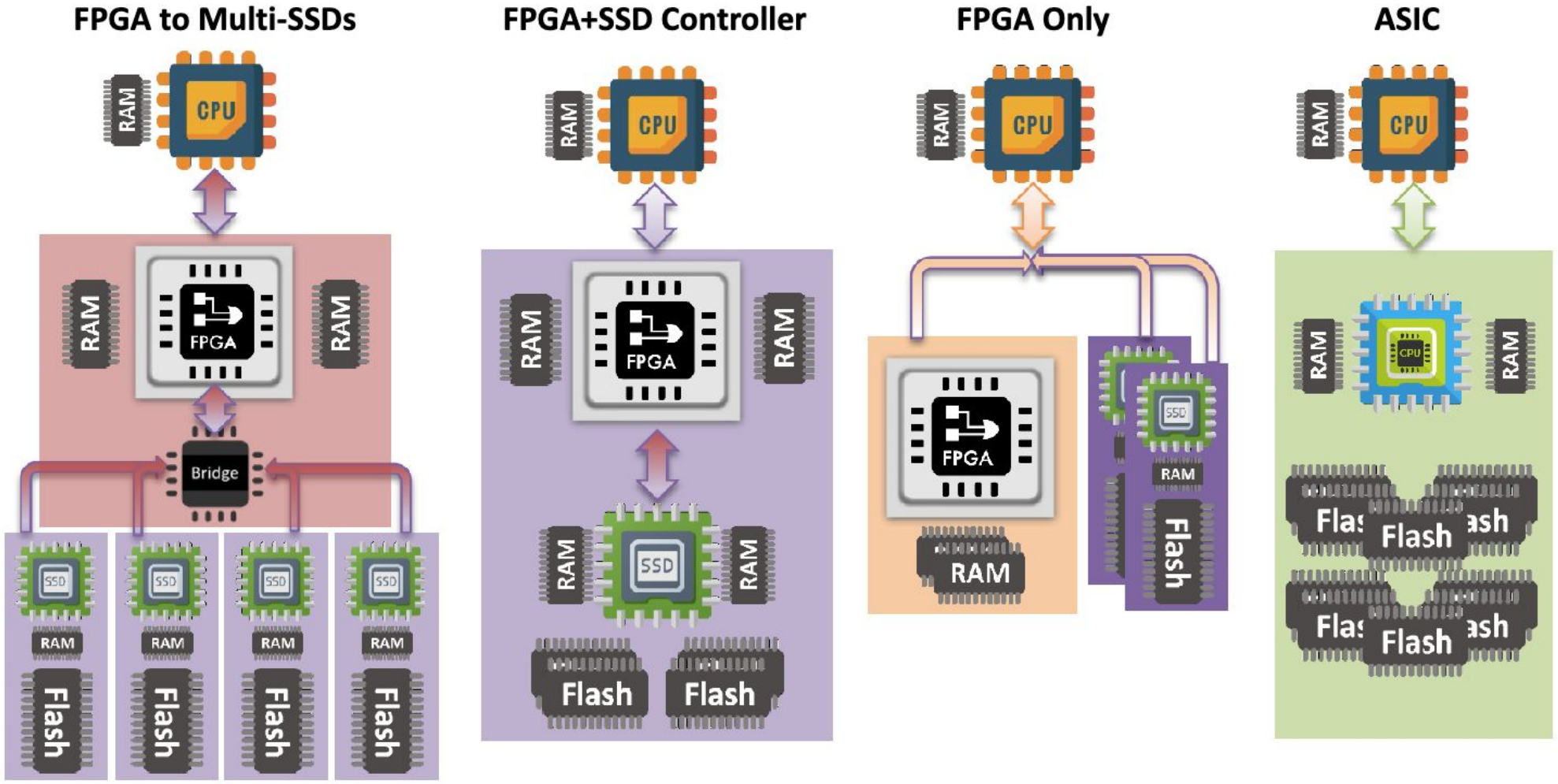
- Key size: 255B, value size: 2MB



This week

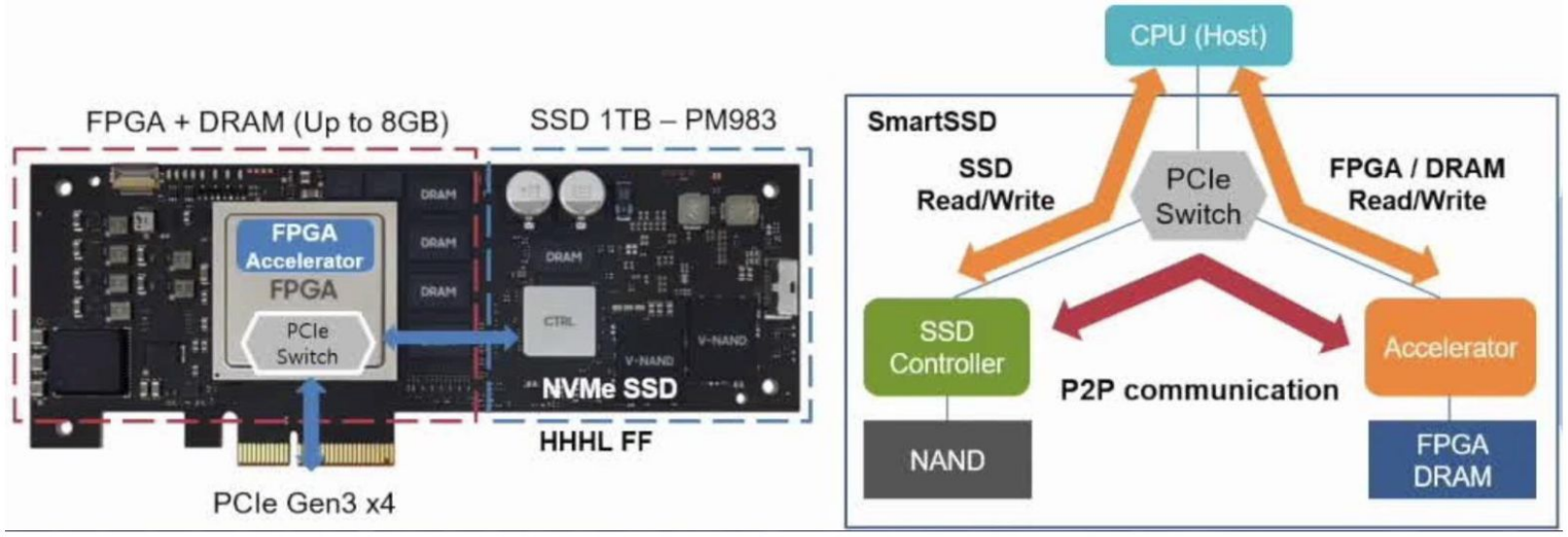
- Solid state drives (SSDs)
  - Flash memory basics
- **SSD types**
  - Zoned namespace SSDs (ZNS)
  - Key-value SSD (KVSSD)
  - **Computational storage (CSD)**
- F2FS

Computational storage instances



Samsung SmartSSD

- Offload computation directly to storage
  - Data filtering, compression, analytics
- Minimizes data movement, saves CPU cycles, lower latency for data access
- Requires new storage stack design



This week

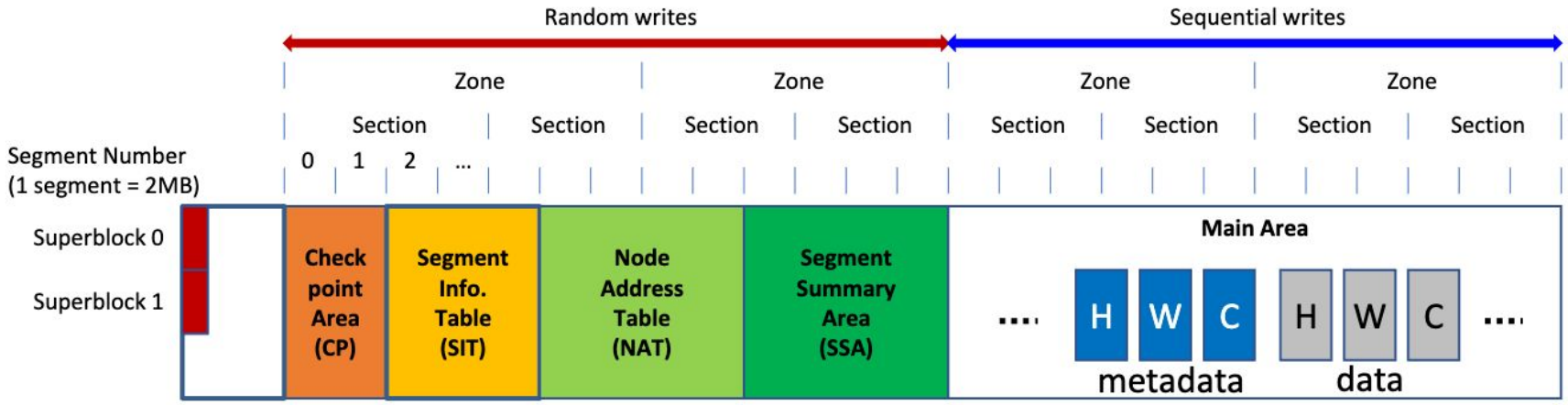
- Solid state drives (SSDs)
  - Flash memory basics
- SSD types
  - Zoned namespace SSDs (ZNS)
  - Key-value SSD (KVSSD)
  - Computational storage (CSD)
- **F2FS**

Flash-friendly file system (F2FS)

- Based on log-structured file system (LFS)
- Design to handle random writes and fsync operations
  - 150% and 70% for mobile and web applications over sequential writes
  - Around 70% more fsync operations
  - Leads to increased device IO latency and reduces device lifetime
- Several techniques used: flash-friendly layout, indexing, multi-head logging, adaptive logging, and fsync acceleration with roll-forward recovery

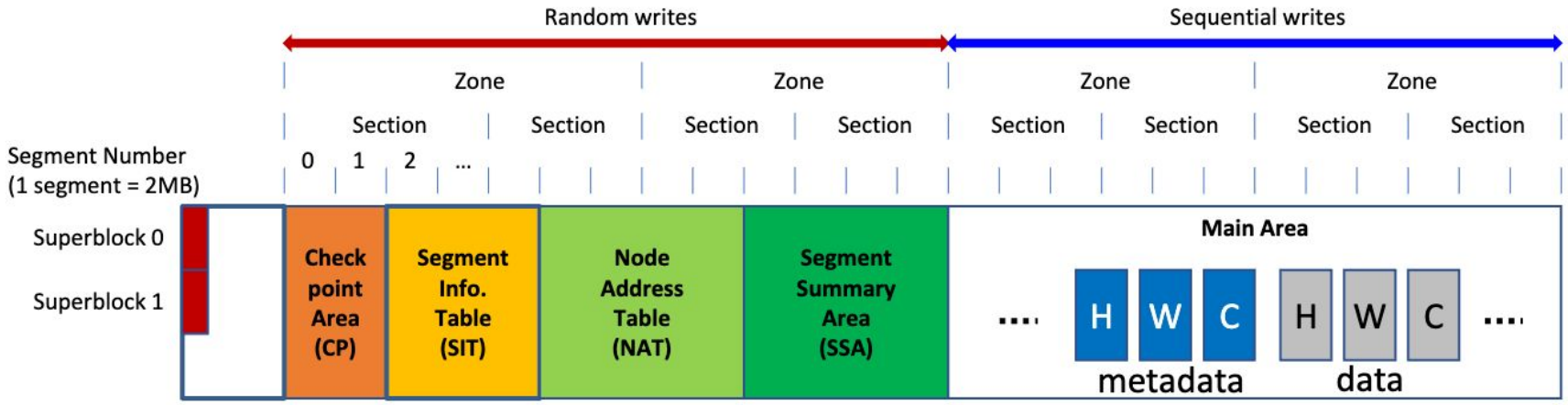
Flash-friendly on-disk layout

- Flash awareness
  - All the metadata located together for locality
  - Start address of the Main area is aligned to the zone size
  - Cleaning is done in a unit of section (FTL's GC unit)
- Cleaning cost reduction
  - Multi-head logging for hot/cold data separation



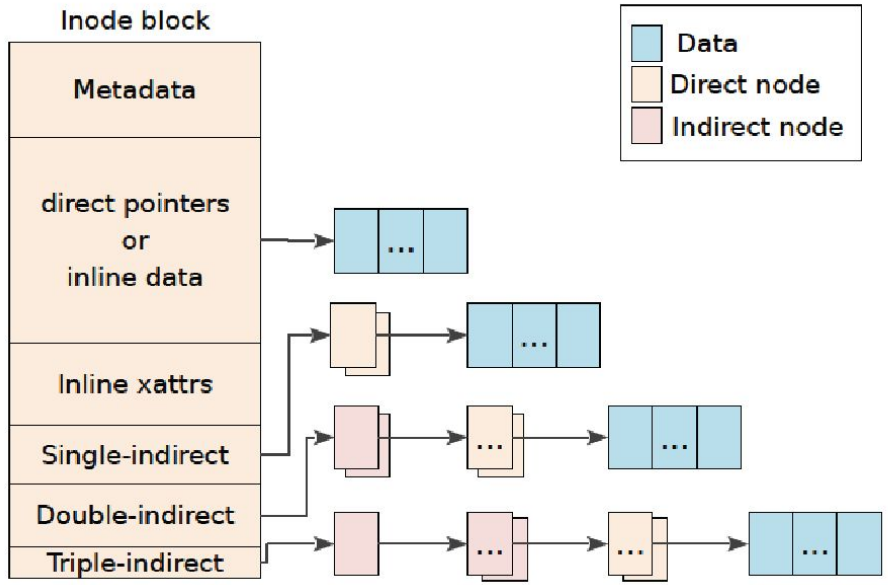
Flash-friendly on-disk layout

- CP: file system status, bitmaps for valid SIT/NAT sets, orphan inode lists & active seg
  - Used for checkpointing pointing
- SIT: Information on valid segments of the main area
- NAT: Address table pointing to all node blocks in the main area
- SSA: Owner info of all blocks (parent inode no and its node/data offsets)
- Main area: 4KB blocks; node: inode or indices of data blocks, data: dir. or user file data

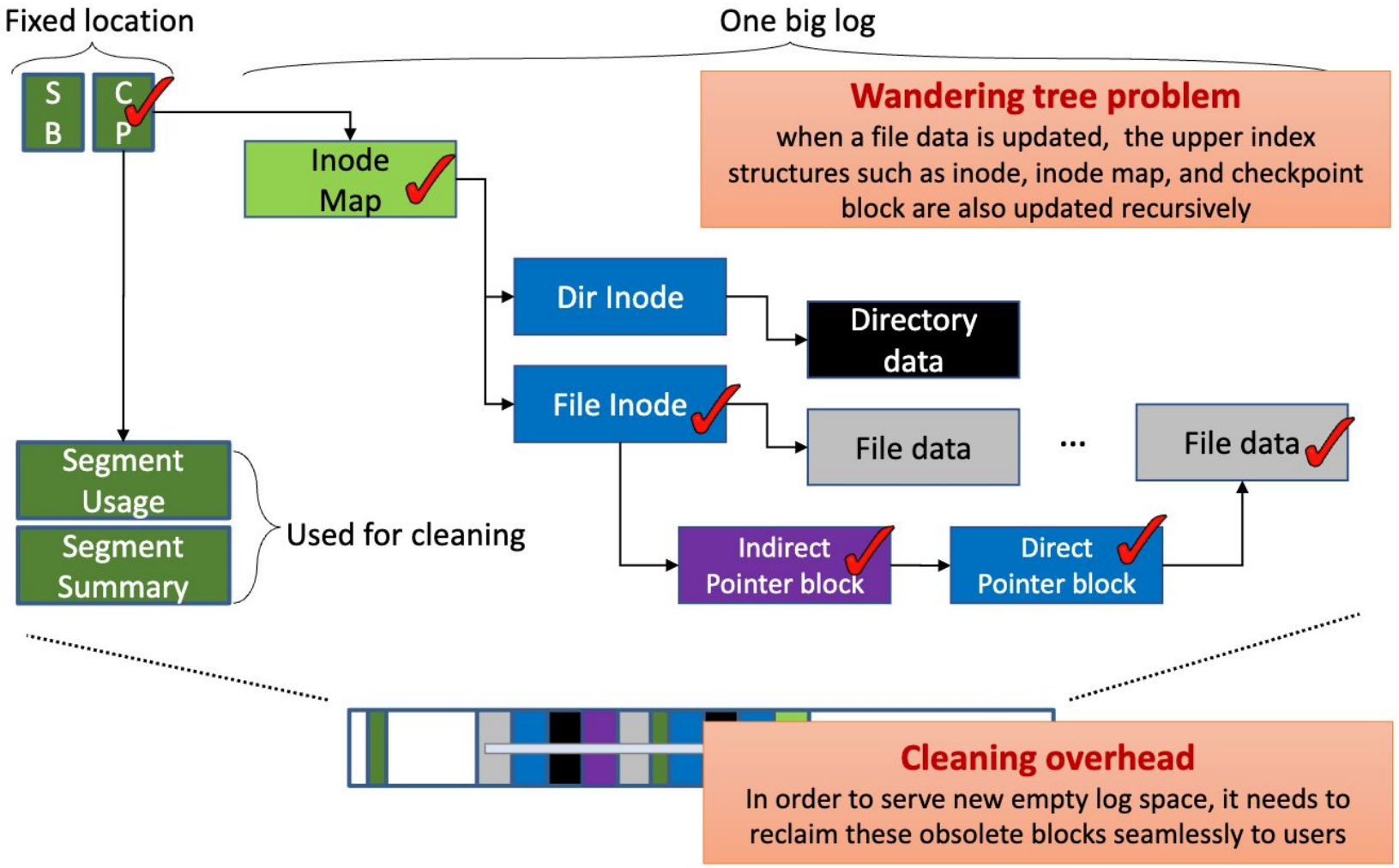


Inode

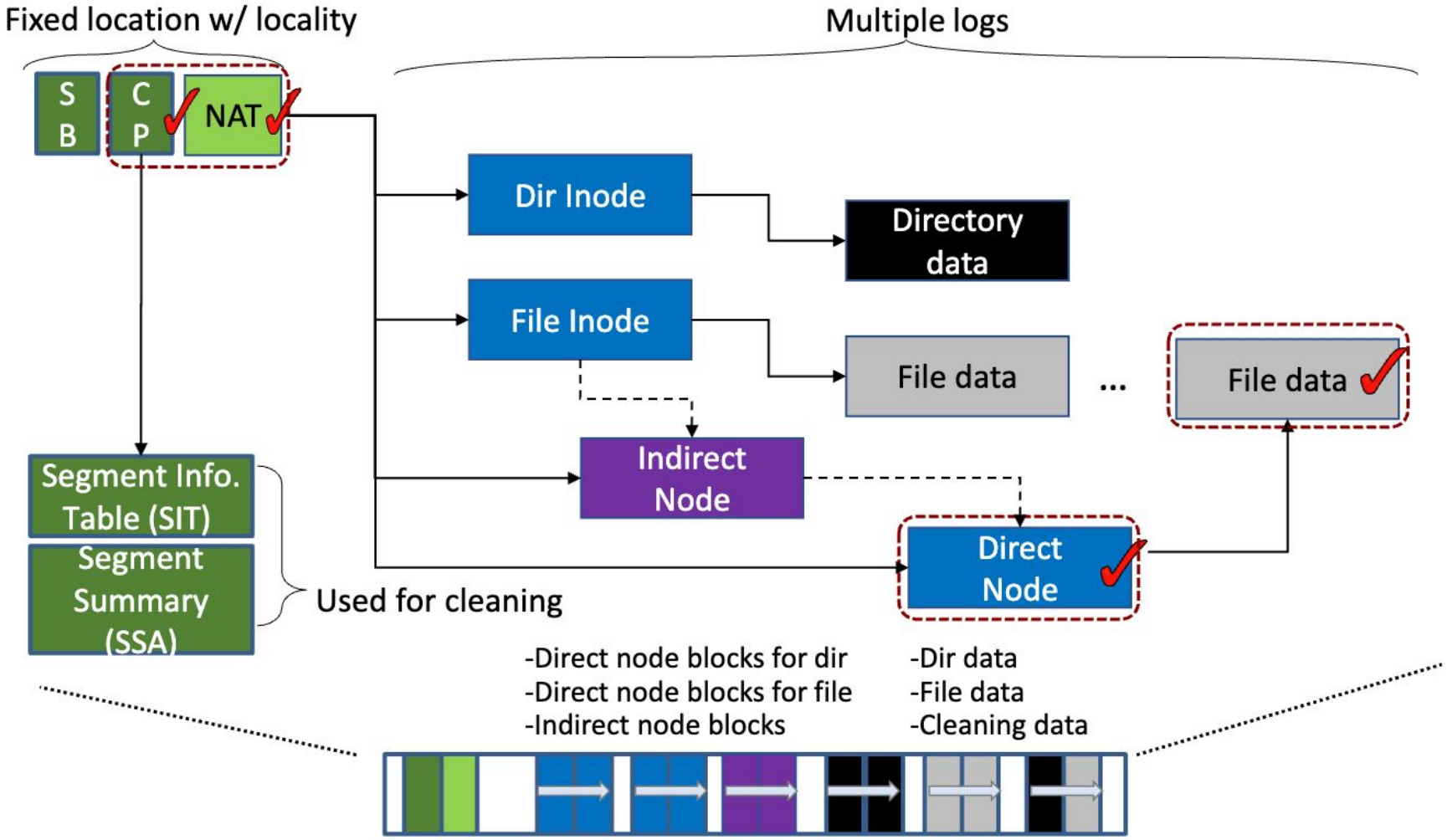
- Unlike LFS, does not maintain inode map
- Uses node structure (unique node ID)
- Uses node ID to use as a physical location by checking the NAT table
- Contains:
  - Metadata, file name, inode no, file size, etc.
- Inline data up to 3,692 bytes
- Inline extended attributes up to 200 bytes



LFS index structure



F2FS index structure



Multi-head logging

- F2FS maintains 6 major log areas: maximize hot and cold data separation
- Data temperature classification
  - Node > data
  - Direct node > indirect node
  - Directory > user file
- Separate multi-head logs in NAND flash
  - Zone aware log allocation for set-associative mapping FTL
  - Multi-stream interface

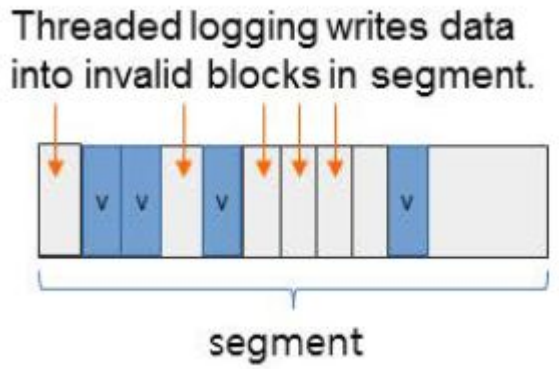
Type	Temp.	Objects
Node	Hot	Direct node blocks for directories
	Warm	Direct node blocks for regular files
	Cold	Indirect node blocks
Data	Hot	Directory entry blocks
	Warm	Data blocks made by users
	Cold	Data blocks moved by cleaning; Cold data blocks specified by users; Multimedia file data

Cleaning

- Cleaning performed at the level of section unit: aligned with FTL's GC unit
- Reclaim scattered and invalidated blocks for further logging
  - Foreground cleaning (reactive) and background cleaning (periodic)
- 3 Step process:
  - Victim selection based on **greedy** (foreground) or **cost-benefit** (background) policy
  - Valid block identification and migration: uses validity bitmap in SIT and migrates parent node blocks to free logs
    - Background cleaning: loads blocks in page cache; marks them dirty; lazily migrates them
  - Post cleaning process: Section becomes pre-free section after migrating all valid blocks; it becomes free only after a checkpoint is made

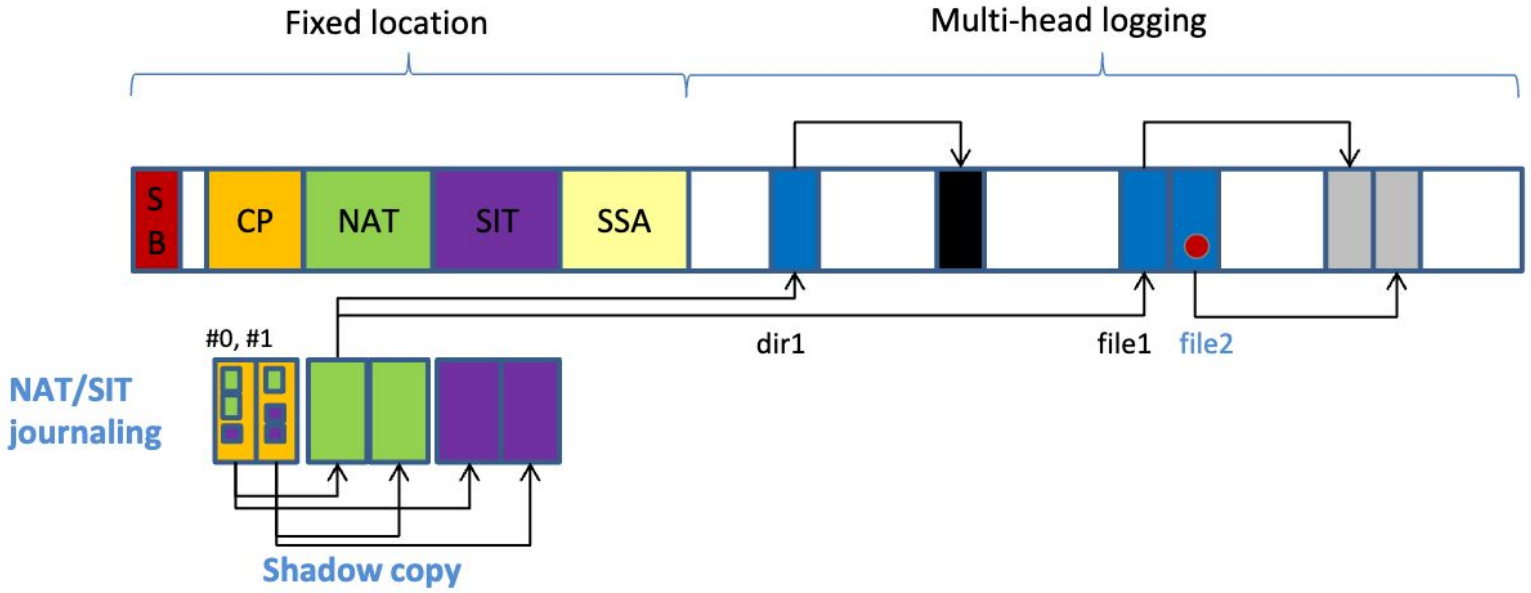
Adaptive logging

- Dynamic write policy, based on the availability of 5% of clean blocks:
  - Append logging (logging to clean segments)
    - Need cleaning operations if there is no free segment
    - Cleaning causes mostly random read and sequential writes
  - Threaded logging (logging to dirty segments)
    - Reuse invalid blocks in dirty segments
    - No need cleaning
    - Causes random writes



Sudden power off recovery

- Checkpoint and rollback mechanism
  - Maintains shadow copy of checkpoint, NAT, SIT blocks
  - Recovers the latest checkpoint
  - Keeps NAT/SIT journal in checkpoint to avoid NAT, SIT writes

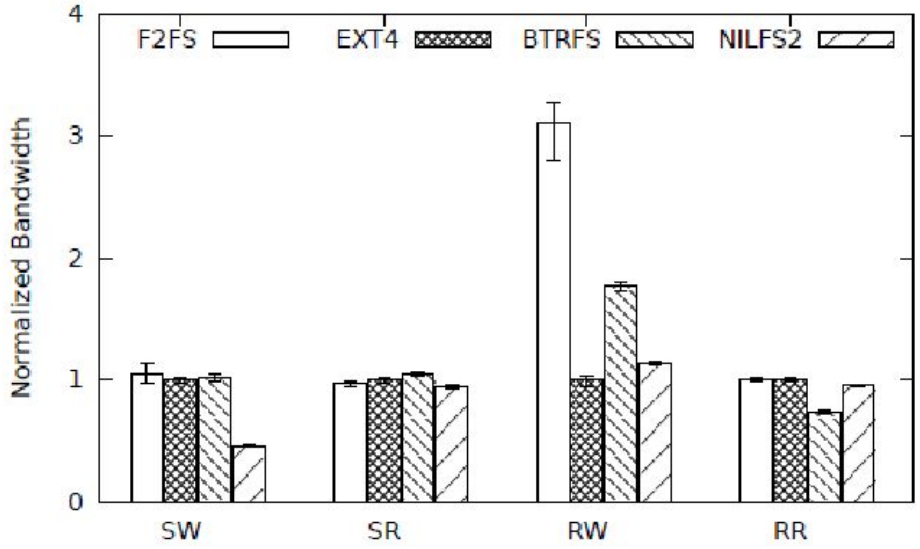


Sudden power off recovery

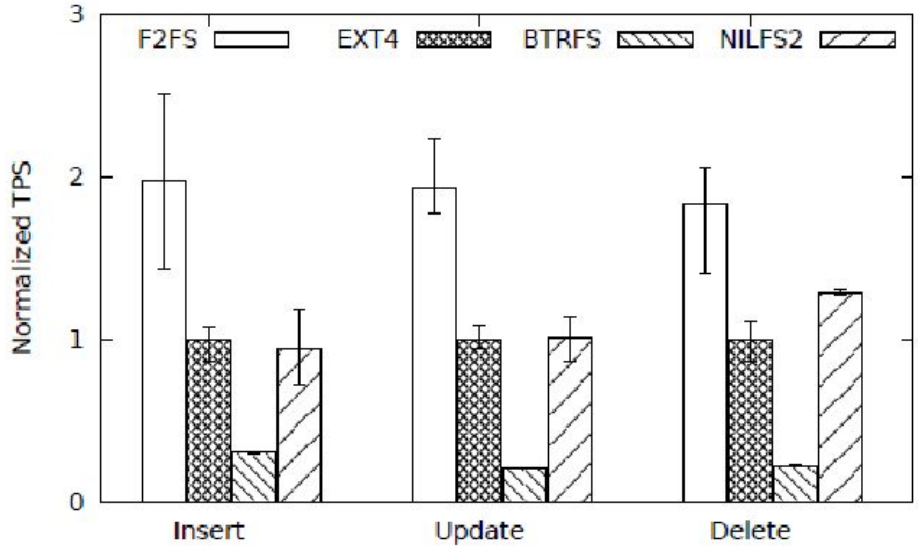
- Fsync handling
  - On fsync, checkpoint is not necessary
  - Direct node blocks are written with fsync mark
- Roll-forward recovery procedure
  - Search marked direct node blocks
  - Per marked node block, identify old and new data blocks by checking the difference between the current and the previous node block
  - Update SIT; invalidate old data
  - Replace new data block writes; update NAT, SIT accordingly
  - Create checkpoint

Mobile benchmark

- In F2FS, more than 90% writes are sequential
  - F2FS reduces writes amount per fsync by using roll-forward recovery
  - Btrfs and Nilfs2 performed poorly than Ext4
    - Btrfs: heavy indexing overhead; Nilfs2: periodic data flush



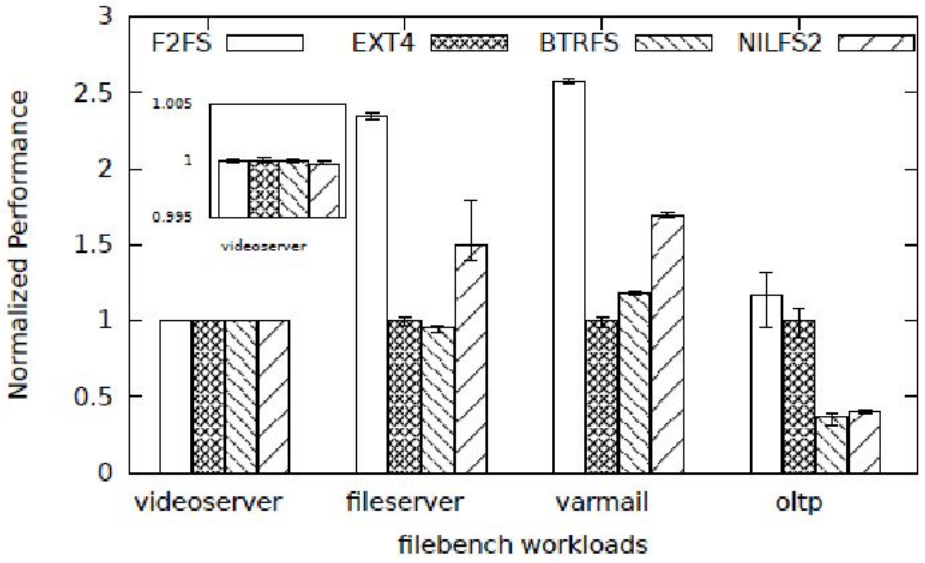
(a) iozone



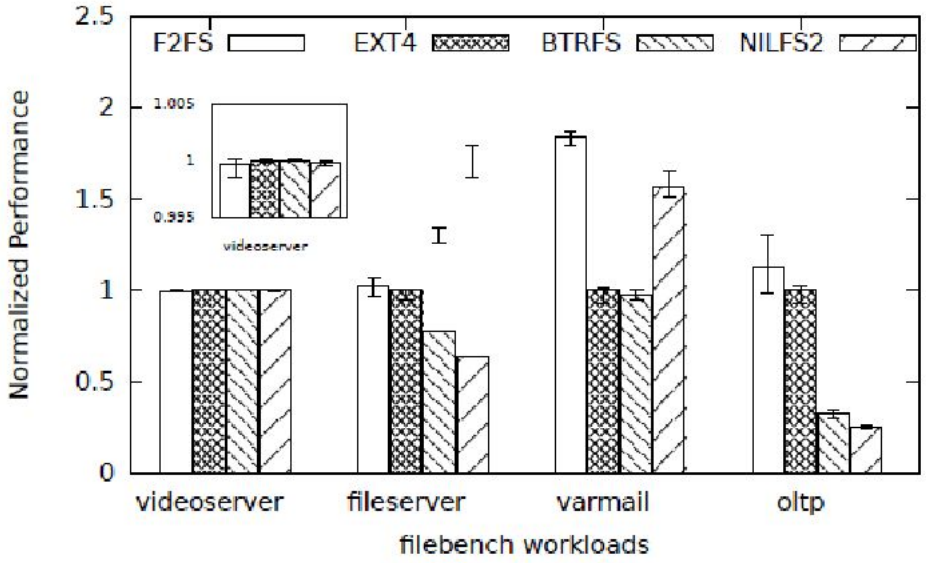
(b) SQLite

Server benchmark

- F2FS better than Ext4 on SATA SSD than PCIe SSD
- Discard size matters in SATA SD due to interface overhead



(a) SATA SSD



(b) PCIe SSD

Summary

- Flash is the widely-deployed storage media
- FTL helps in managing data operations to flash memory
  - Suffers from endurance issues (wear leveling) and write amplification
- Various types of SSDs present to cater various workload characteristics
- F2FS specifically designed for flash memory
- Minimizes random writes with flash-friendly on-disk layout
- Proposes several ways to maintain sequentiality and parallelism