Operating Systems

Associate Prof. Yongkun Li
中科大-计算机学院 副教授
http://staff.ustc.edu.cn/~ykli

Chapter 8
Mass Storage

Topics in Part 3 (Storage Management)

User Space

Processes

Operating System Kernel

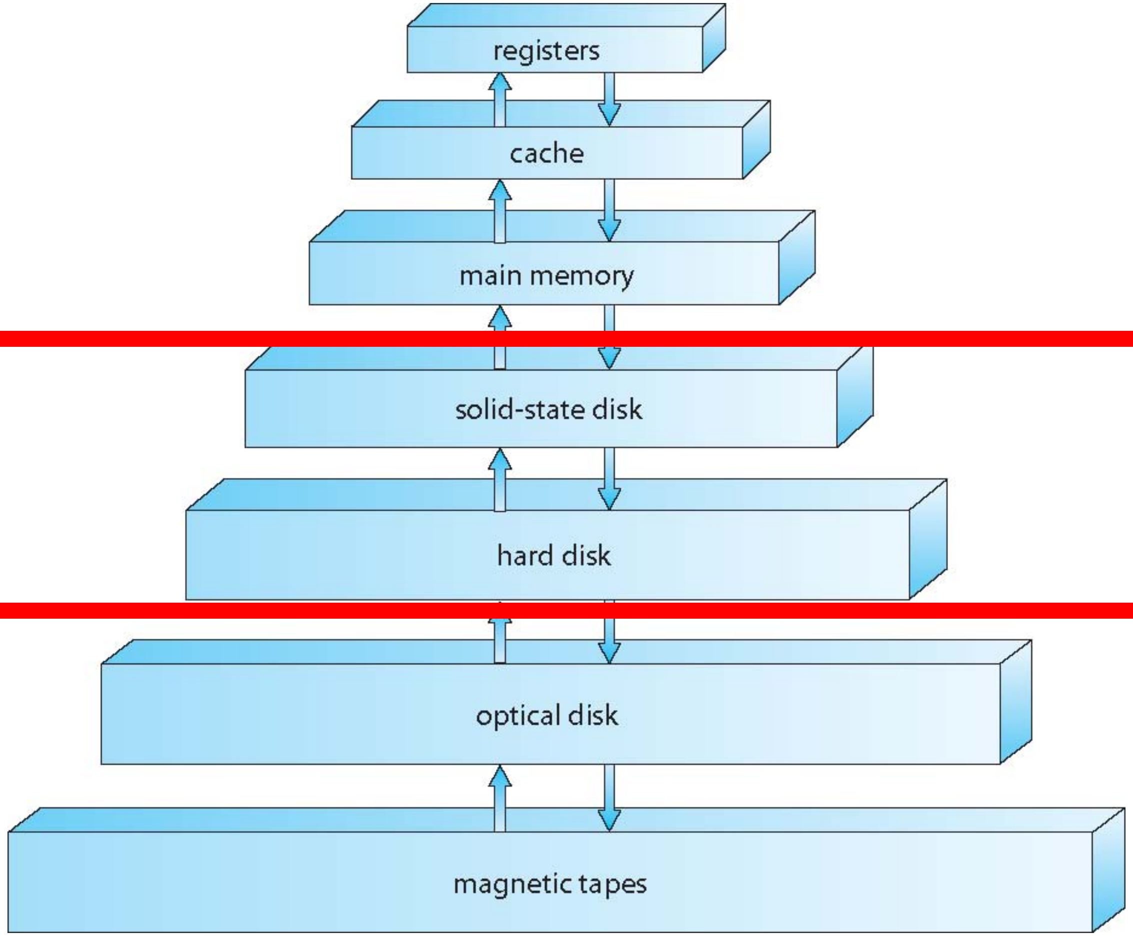
File system Implementation

FAT32, EXT2/3 KV, Distributed FS, Graph System…

File System Operations

Devices

Storage Hierarchy



Topics in Ch 7 (Mass Storage)

Disk Structure

Disk Scheduling

SSD Structure

SSD Features/Issues

RAID

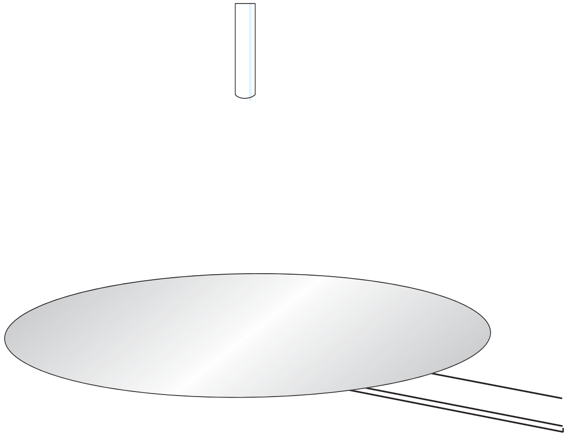
Erasure Coding

Research Problems

Topics

- Disk structure
- Disk scheduling
- Solid-state drives (SSDs)
- RAID
- Erasure coding

Hard Disk Structure – Physical view



Physical address (cylinder, track, sector)

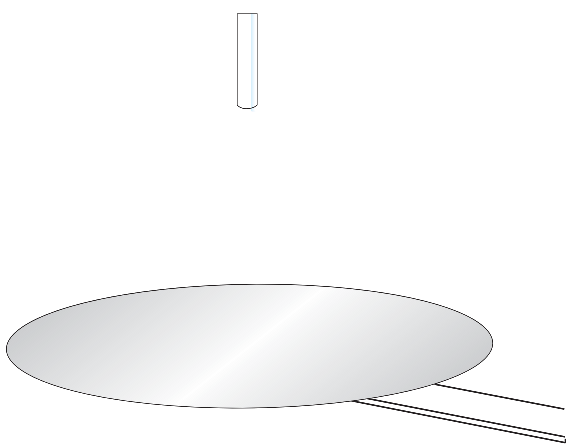
Track:
The surface of a platter is divided into tracks
Sector:
Track is divided into sectors (512B data + ECC)
Cylinder:
Set of tracks that are at one arm position

Access: Seek + Rotate

Seek time:
move disk arm to desired cylinder

Rotational latency:
spin at 5400/7200/10K/15K RPM

Hard Disk Structure – Physical view



<u>Constant liner velocity (CLV)</u>

- Uniform density of bits per track, outer track hold more sectors
- Variable rotation speed to keep the same rate of data moving
- **<u>CD-ROM/DVD-ROM</u>**

<u>Constant angular velocity (CAV)</u>

- Constant rotation speed
- Higher density of bits in inner tracks
- **<u>Hard disks</u>**

Hard Disk Structure – Logical view

<u>How to use?</u>

Large 1-D arrays of logical blocks (usually 512 bytes)

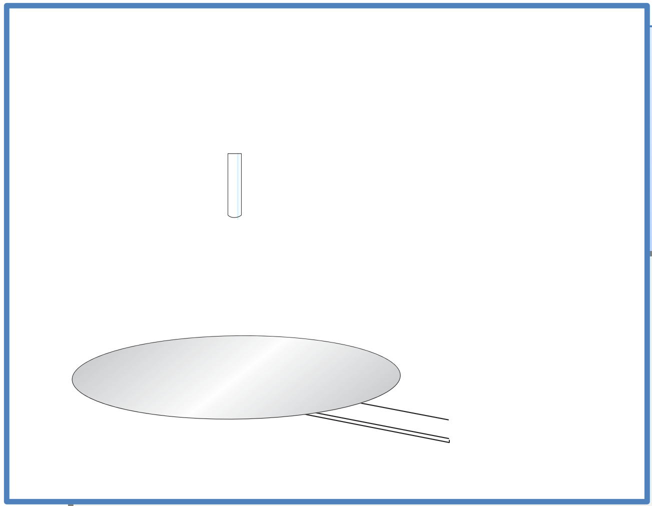
<u>Address mapping</u>

Logical block number -> (cylinder #, track #, sector #)

<u>Disk management is required</u>

- ➢ **<u>Disks are prone to failures: defective sectors are common (bad blocks)</u>**
  - ✓ Need to handle defective sectors: bad block management
- ➢ **<u>Disk formatting</u>**

Disk Management



Bad Block Management

- ✓ Maintain a list of bad blocks (initialized during low-level formatting) and preserve an amount of spare sectors
- ✓ **Sector sparing/forwarding:** replace a bad sector logically with one spare sector
  - Problem: invalidate disk scheduling algorithm
  - Solution: spare sectors in each cylinder + spare cylinder
- ✓ **Sector slipping:** remap to the next sector (data movement is needed)

Disk Management

Disk Formatting

Step 1: Low-level formatting/physical formatting

- ✓ Divide into sectors so disk controller can read/write
- ✓ Fills the disk with a special data structure for each sector (data area(512B), header and trailer (sector number & ECC))
  - The controller automatically does the ECC processing whenever a sector is read/written
- ✓ Done at factory, used for testing and initializing (e.g., the mapping). It is also possible to set the sector size (256B, 512B, 1K, 4K)

Disk Management

Disk Formatting

Step 2: How to use disks to hold files after shipment?

- ➢ **Choice 1: File system**
- ✓ Partition into one or more groups of cylinders (each as a separate disk)
- ✓ Logical formatting: creating a FS by storing the initial FS data structures
- ✓ I/O optimization: Disk I/O (via blocks) & file system I/O (via clusters), why?
  - More sequential access, fewer random access

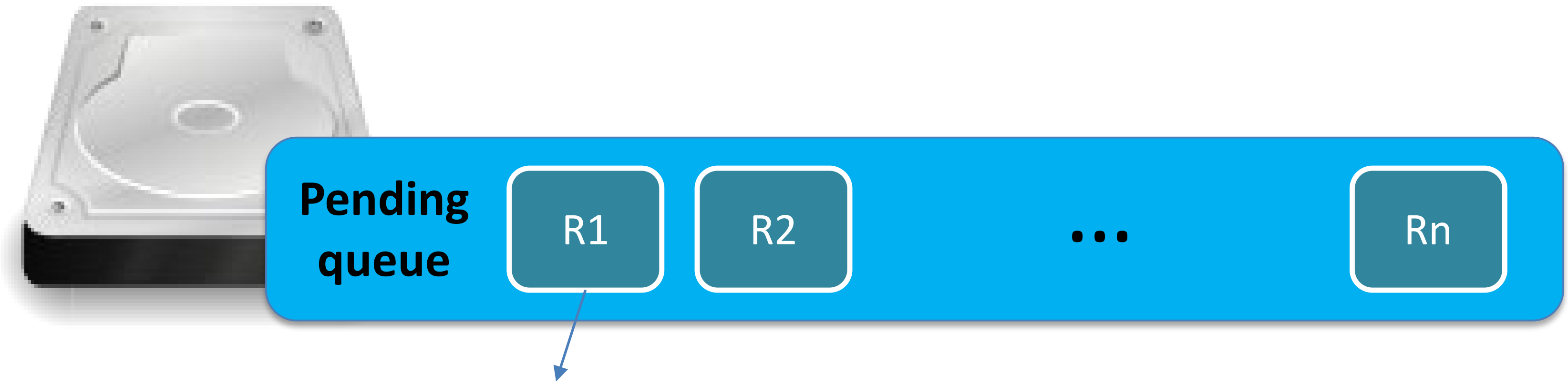
- ➢ **Choice 2: Raw disk**
- ✓ Use disk partition as a large sequential array of logical blocks, without FS
- ✓ Raw I/O: bypass all FS services (buffer cache, prefetching…), be able to control exact disk location

Topics

- Disk structure
- **Disk scheduling**
- Solid-state drives (SSDs)
- RAID & Erasure codes
- Problems with EC

Why needed?

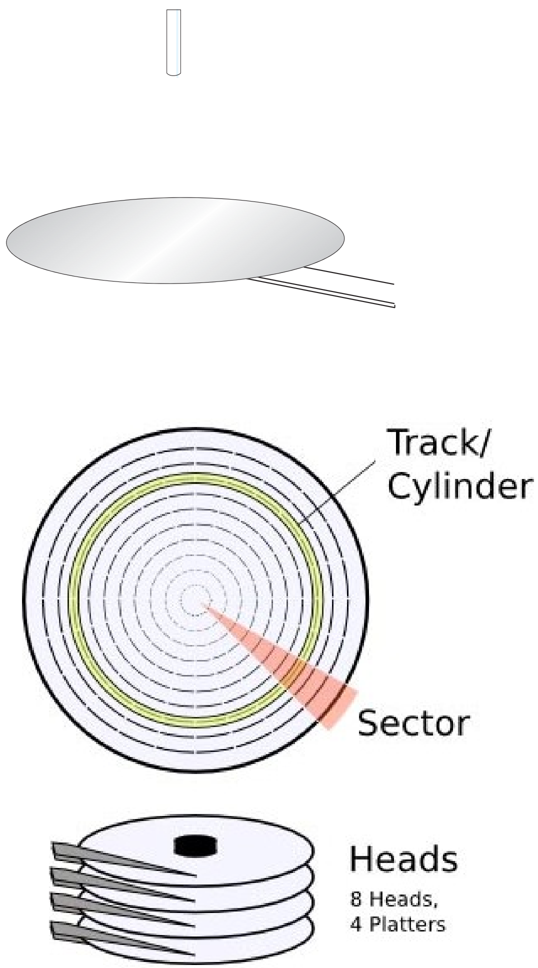
- Requests are placed in the queue of pending requests for that drive if the drive/controller is busy



Read/write, disk address, memory address, number of sectors to be transferred

Request ordering significantly affects the access performance (seek + rotate), so scheduling is needed

What is disk scheduling



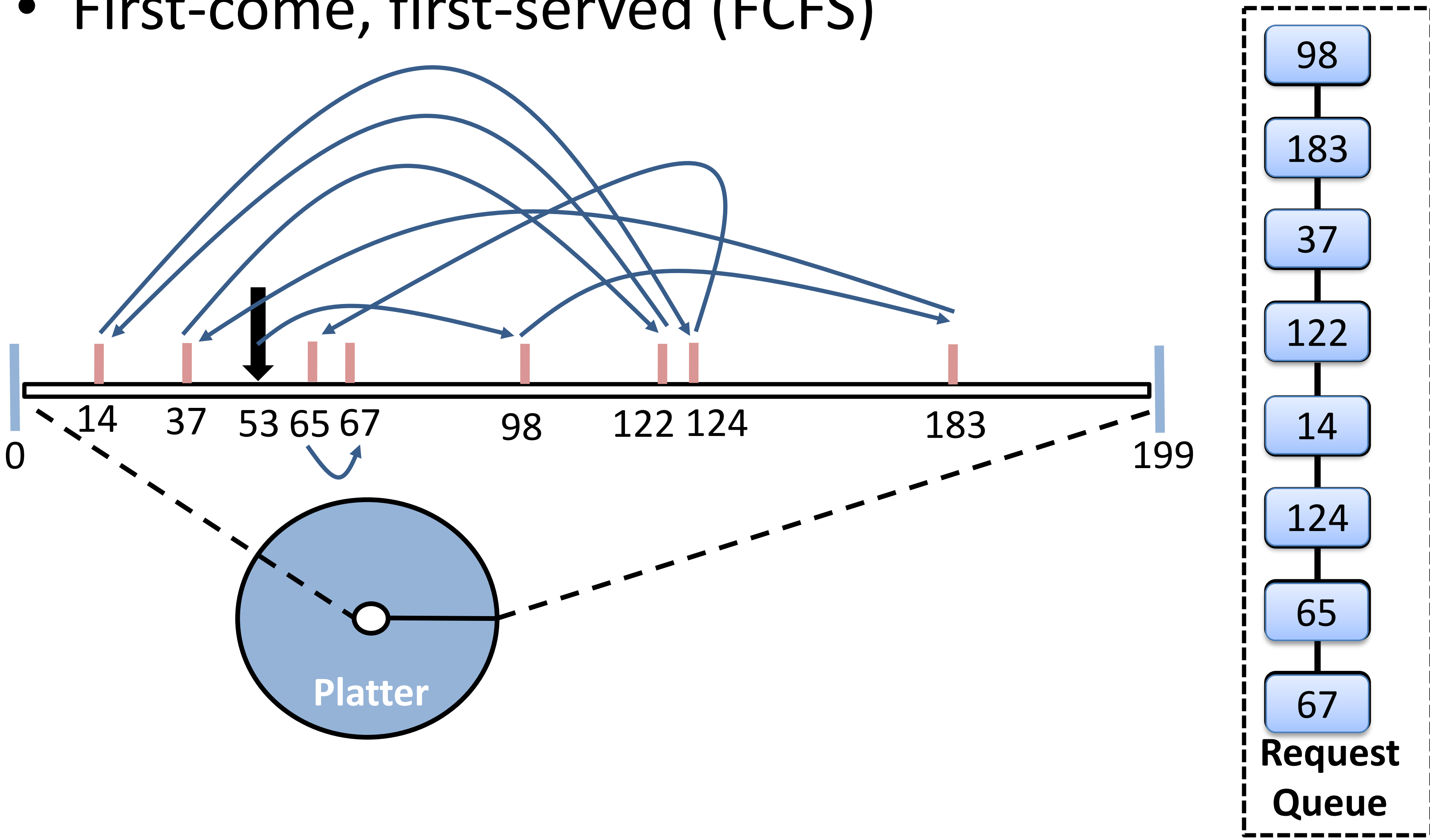
- I/O access procedure
  - Seek: move the head to the desired cylinder
  - Rotate: spin to the target sector on the track
- Disk scheduling
  - Choose the next request in the pending queue to service so as to minimize the **seek time**
- Scheduling algorithms

FCFS Scheduling

- First-come, first-served (FCFS)
  - Intrinsically fair, but does not provide the fastest service

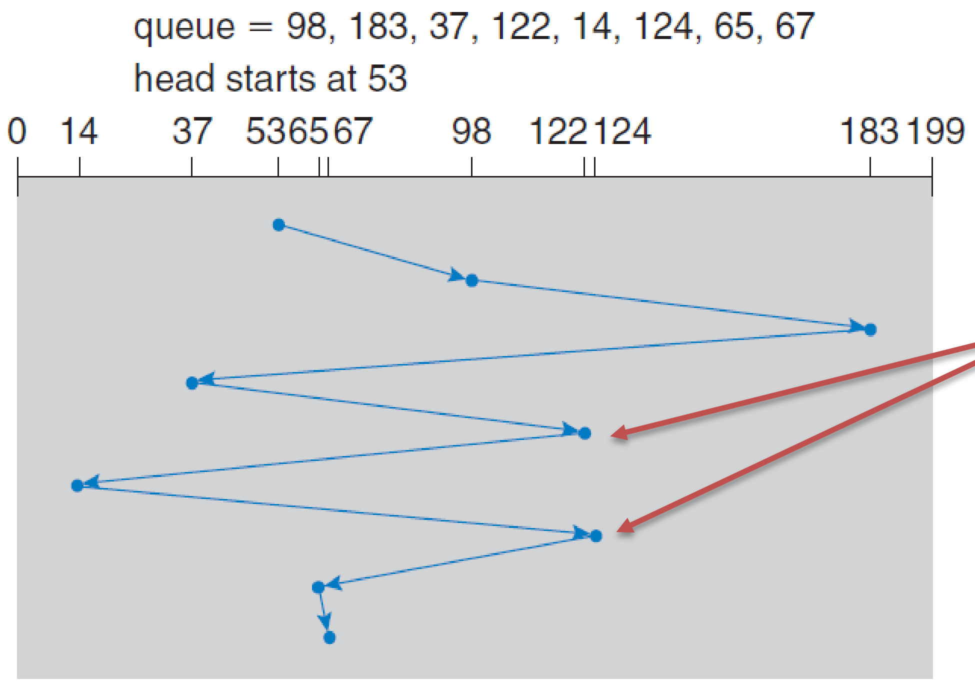
FCFS Scheduling

- First-come, first-served (FCFS)



FCFS Scheduling

- Scheduling diagram



Total head movement (640 cylinders)

Wild swing is very common

E.g.: 122 to 14, then to 124

How to reduce the head movement?

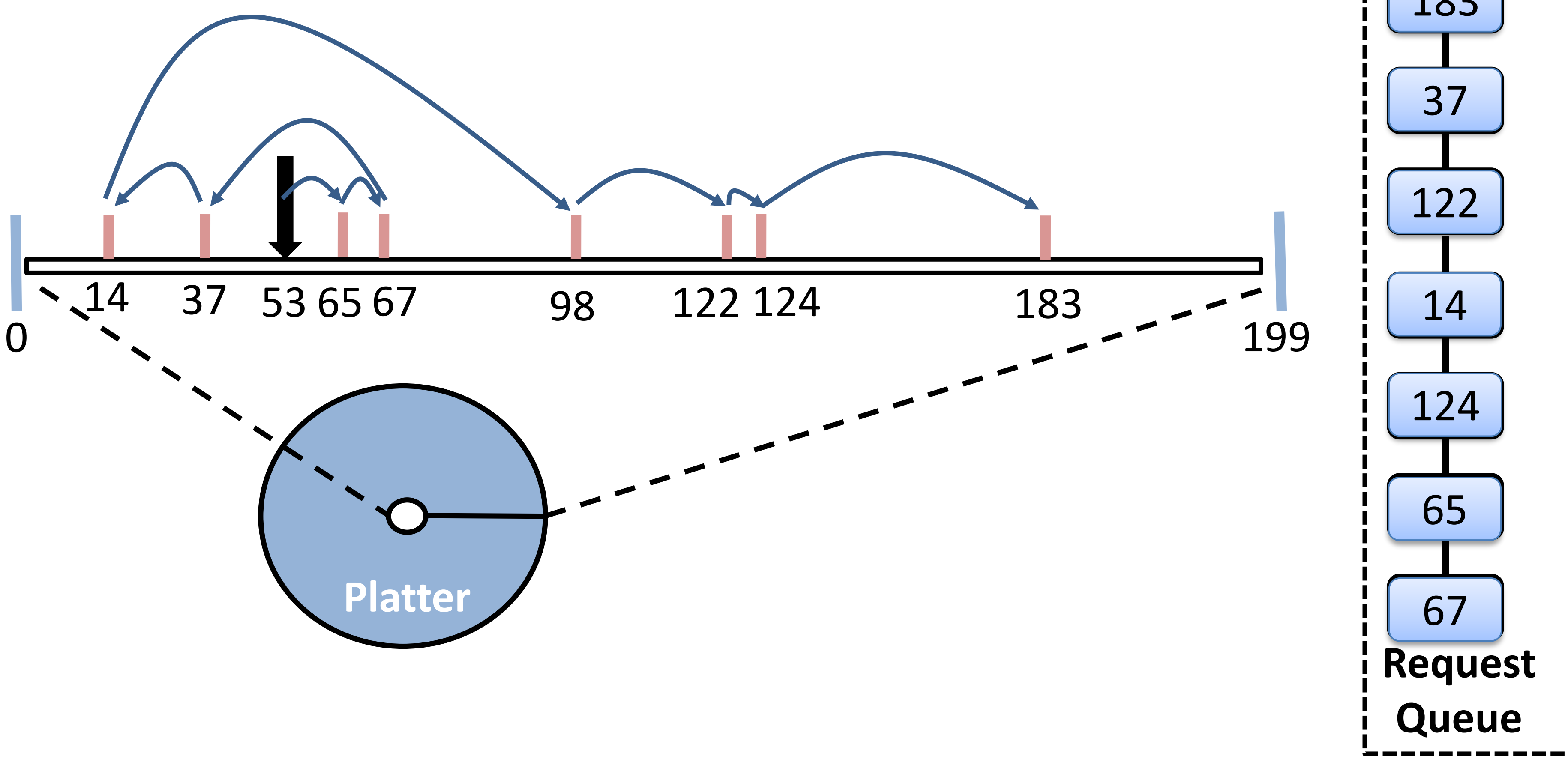
Handle nearby requests first

SSTF Scheduling

- Shortest seek time first (SSTF)
  - Choose the request with the least seek time
  - Choose the request closest to the current head position

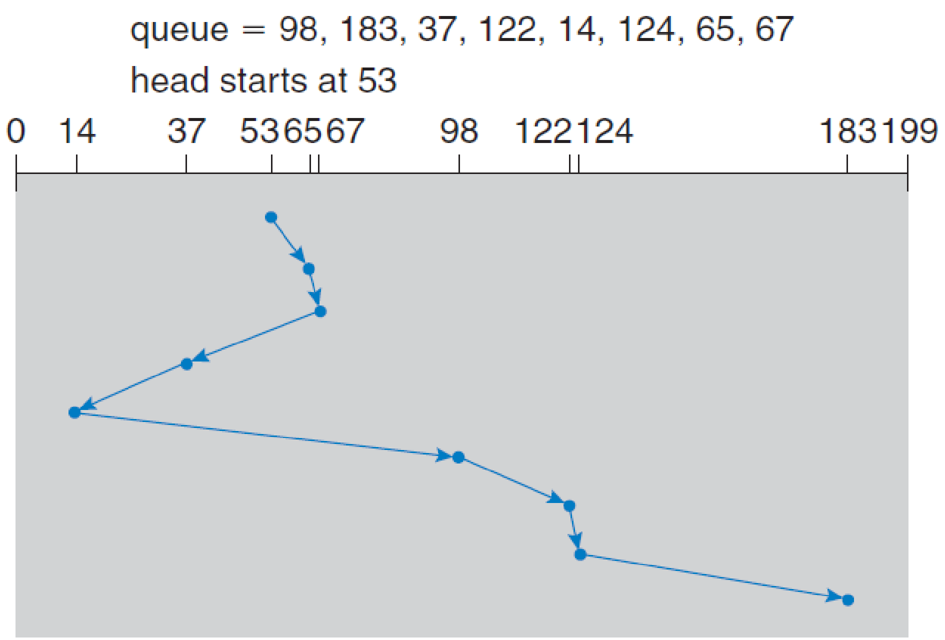
SSTF Scheduling

- Shortest seek time first (SSTF)



SSTF Scheduling

- Scheduling diagram



Total head movement: 236 cylinders (it is 640 for FCFS)

Essentially a form of SJF scheduling

It is not optimal

The sequence of 53-37-14-65… could reduce the head movement to 208

It may cause starvation

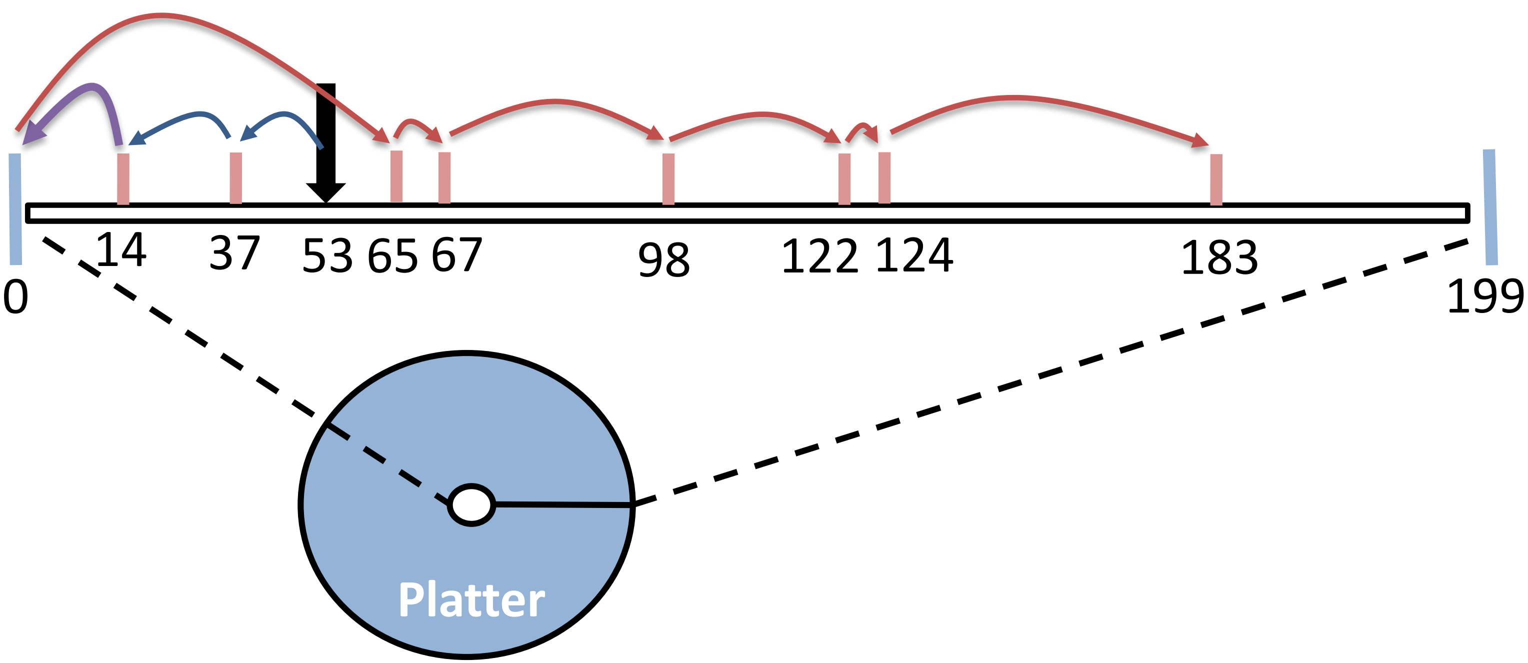
SCAN Scheduling

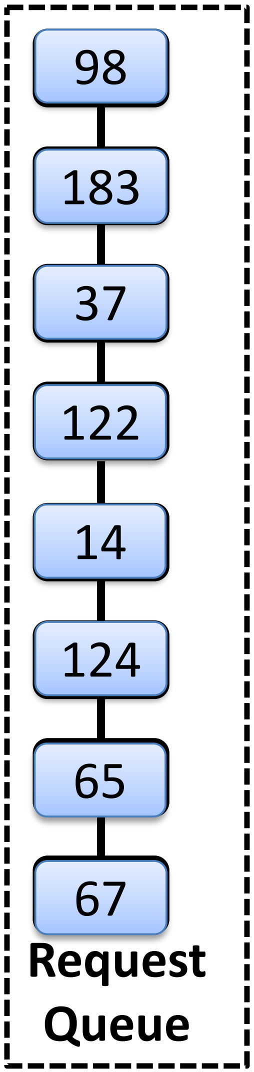
- Scan back and forth
  - Starts at one end, moves toward the other end
  - Service the requests as it reaches each cylinder
  - Reverse the direction
  - **Elevator algorithm**

SCAN Scheduling

- Scan back and forth

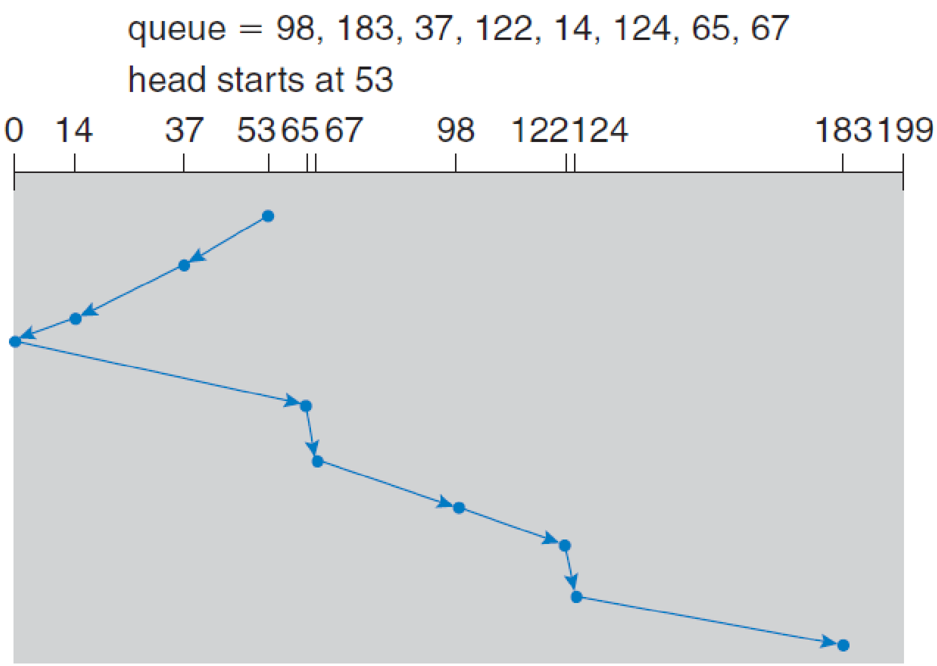
Suppose the head is moving from 53 to 0





SCAN Scheduling

- Scheduling diagram



Any problem?

Assume a uniform request distribution

The heaviest density of requests is at the other end of the disk

They need to wait for a long time

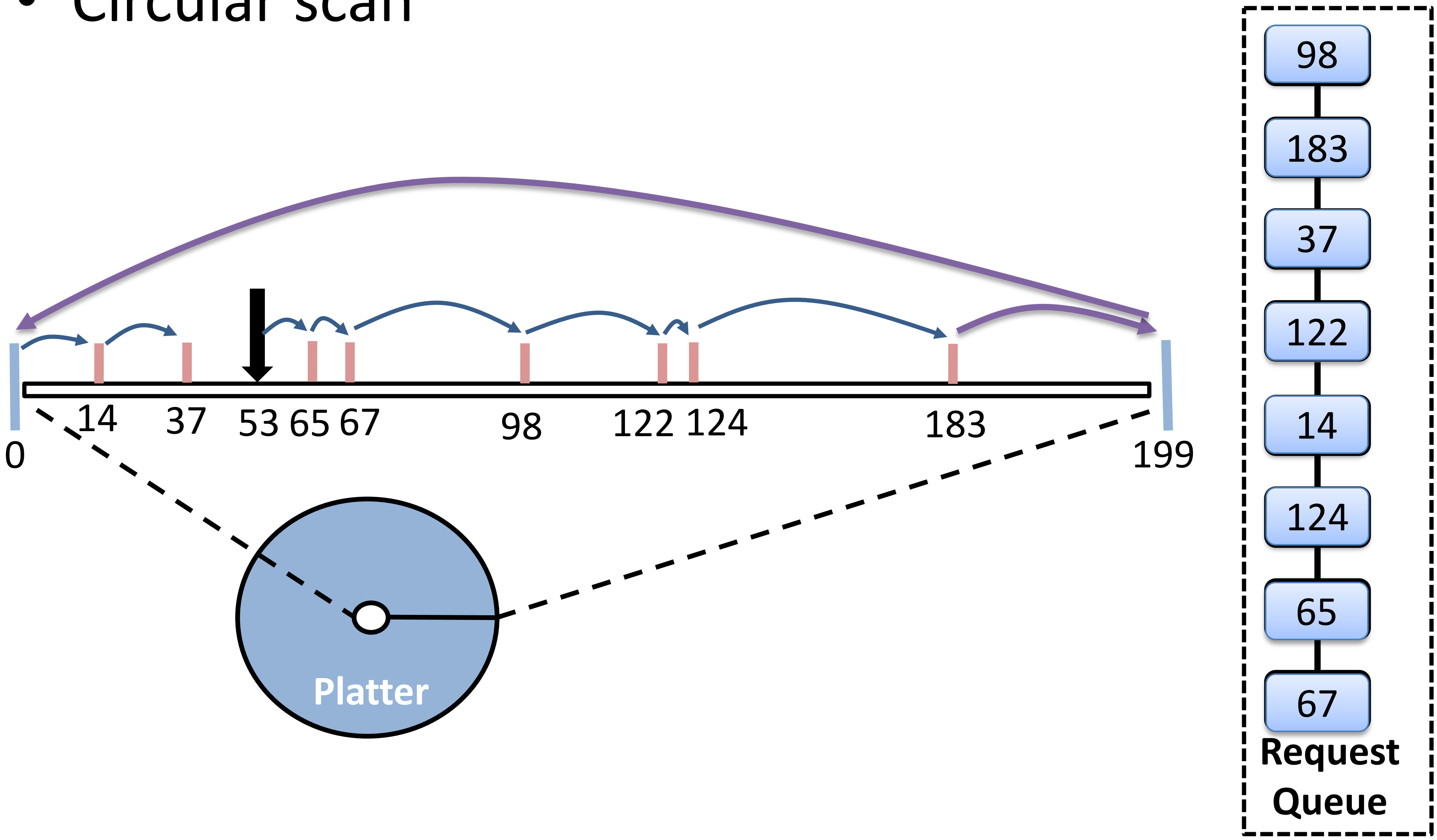
Can we do something about this?

C-SCAN Scheduling

- Circular Scan back and forth
  - A variant of SCAN: immediately return when reaches the end
  - Aim for providing a more uniform wait time

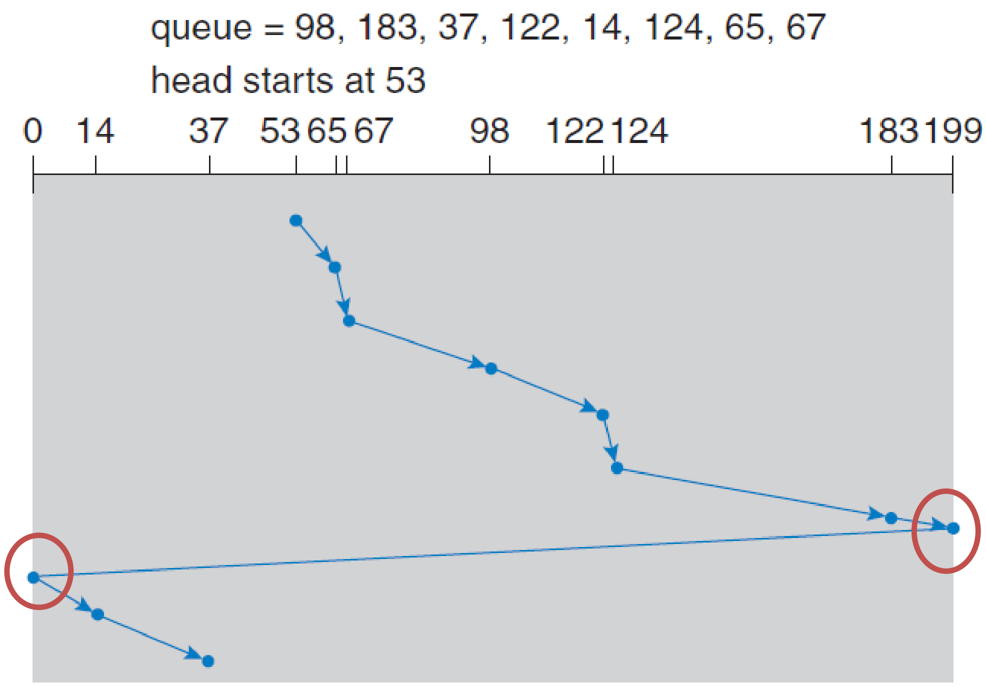
C-SCAN Scheduling

- Circular scan



C-SCAN Scheduling

- Scheduling diagram



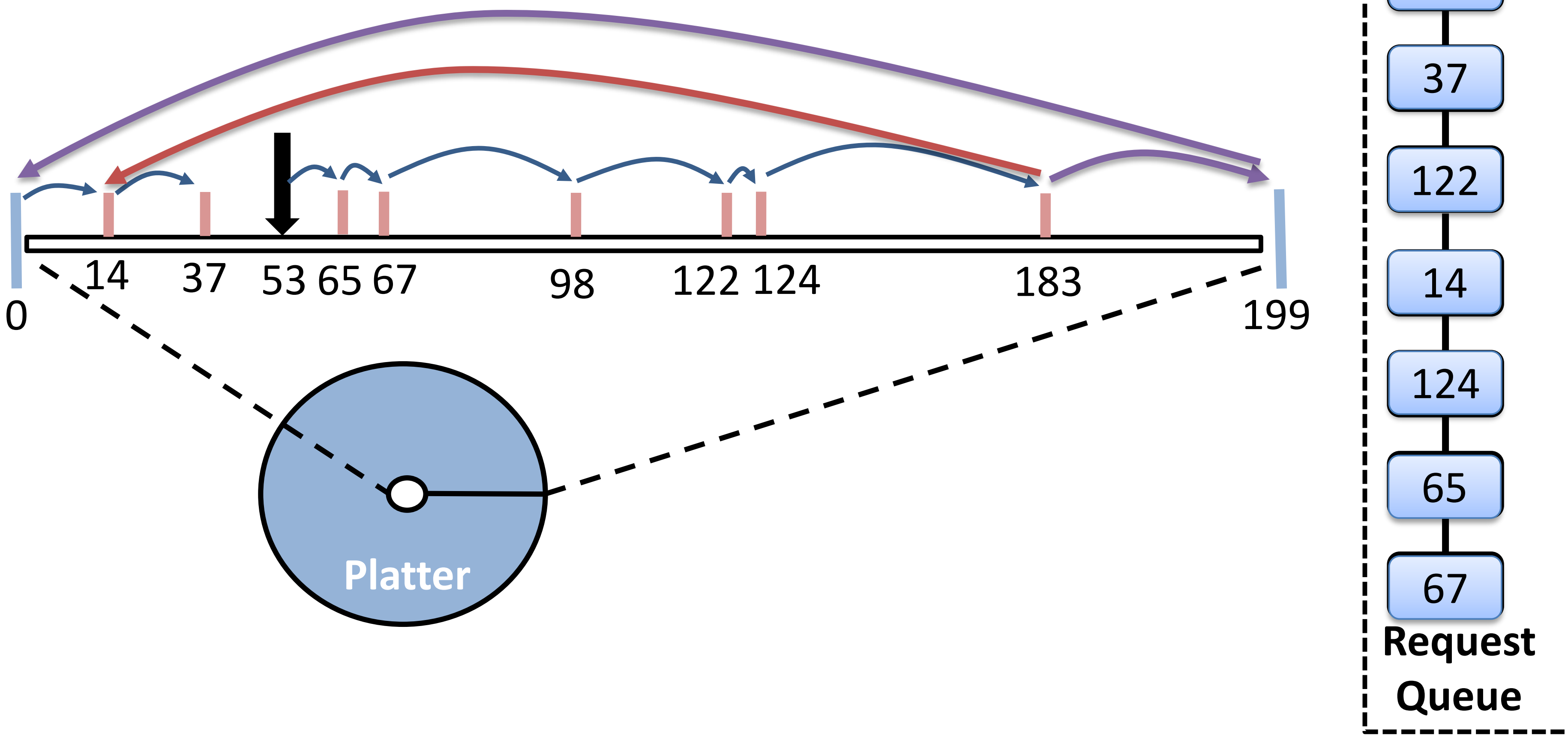
Unnecessary

No need to move across the full width of the disk, but only need to reach the final request

Improved SCAN and C-SCAN: LOOK and C-LOOK

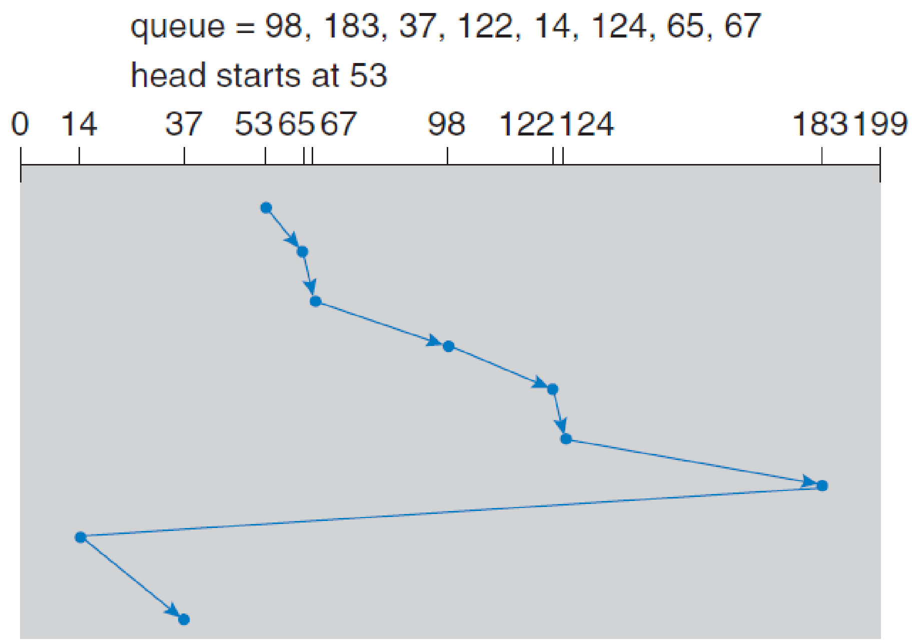
C-LOOK Scheduling

- Goes only as far as the final request
  - Look for a request before moving



C-LOOK Scheduling

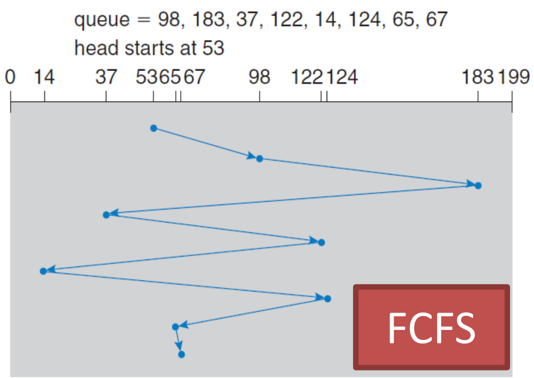
- Scheduling diagram

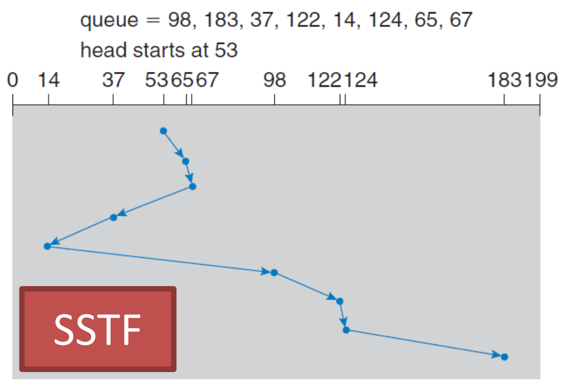


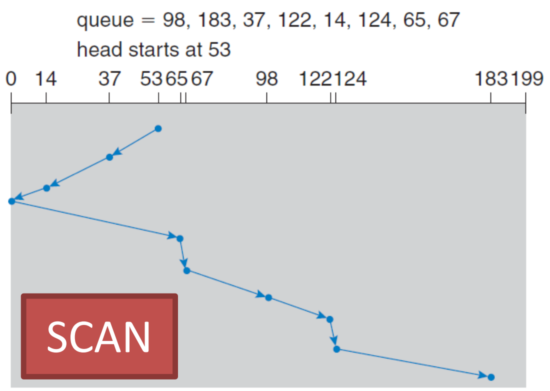
<u>Look for a request before continuing to move in a given direction</u>

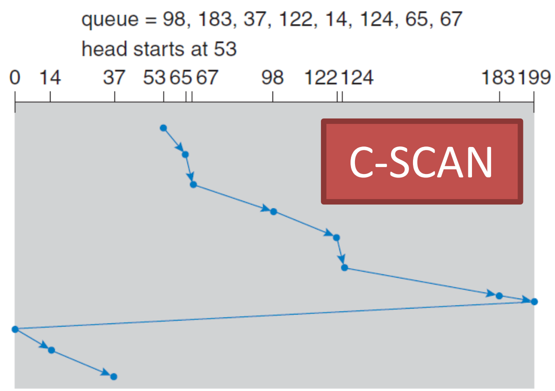
<u>Fewer head movements than SCAN/C-SCAN</u>

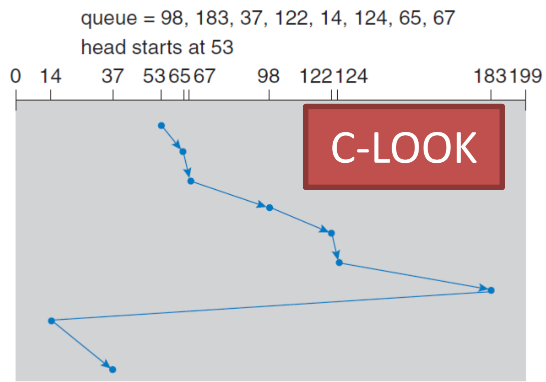
Summary of scheduling algorithms











SSTF outperforms FCFS, but may suffer from starvation

SCAN and C-SCAN perform better for heavy load systems, and they are less likely to cause starvation

Selection of a scheduling algorithm

Disk Performance

Number and types of requests

File allocation method
Large sequential I/O or small random I/O

Location of directories and index blocks (metadata I/O)

Implementing scheduling in OS is necessary to satisfy other constraints (e.g., priority defined by OS)

Write disk scheduling as a separate module of the OS
Can be easily replaced with different alg. (default: SSTF/LOOK).

Topics

- Disk structure
- Disk scheduling
- **Solid-state drives (SSDs)**
- RAID
- Erasure coding

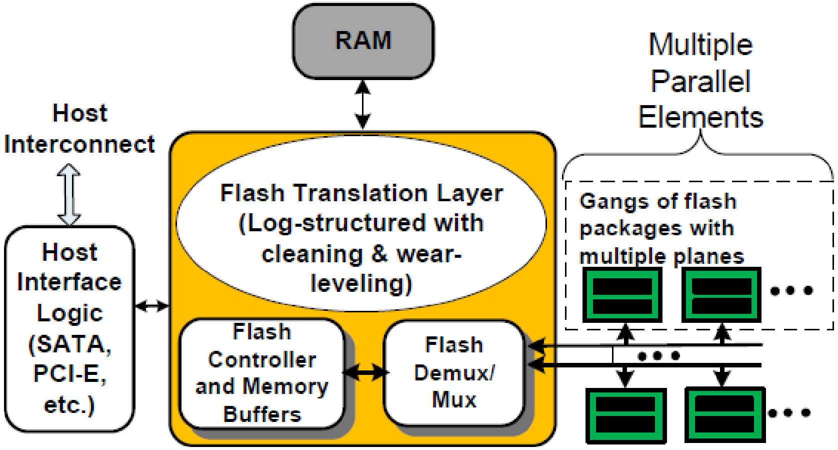
- Solid-state drives (SSDs)
  - SSD architecture
  - SSD operations
  - Flash translation layer
  - Research topics

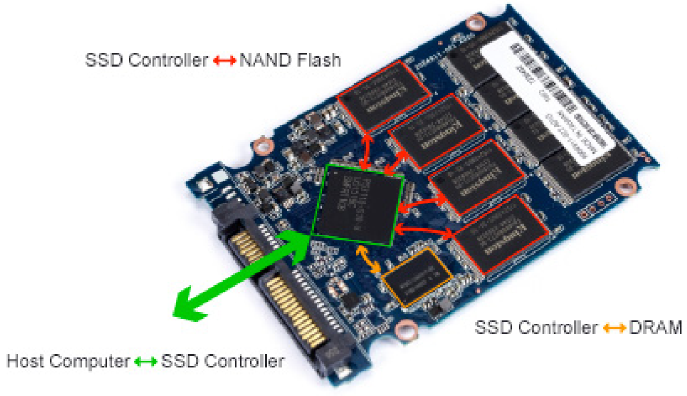
SSDs are widely used

Advantages of flash-based SSDs: non-volatility, shock resistance, high speed and low energy consumption;

SSD Architecture

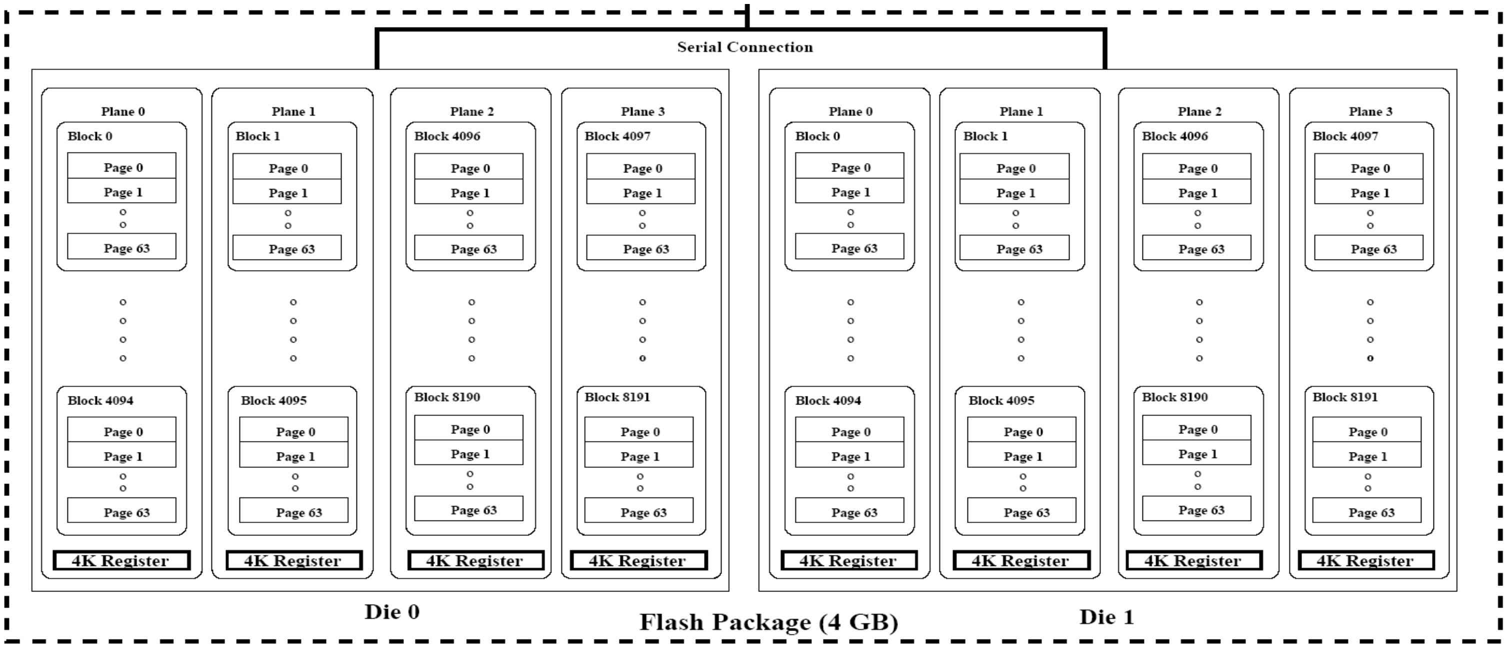
- SSD components
  - Multiple flash packages, controller, RAM





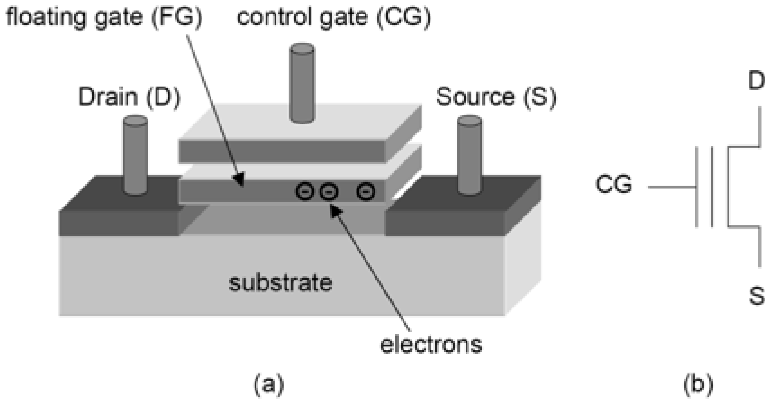
Flash Package

- Package > die/chip > plane > block > page



Samsung K9XXG08UXM (SLC) (2 dies, 4 planes, 2048 blocks, 64 pages)

Flash Cell



(a) Floating gate memory cell and (b) its schematic symbol

- Each cell stores one bit (or multiple bits)
- Program operation can only change the value from 1 to 0 (erase operation changes the value from 0 to 1)
  - **No overwritten**
- The floating gate becomes thinner as the cell undergoes more program-erase cycles
  - **Decreasing reliability**

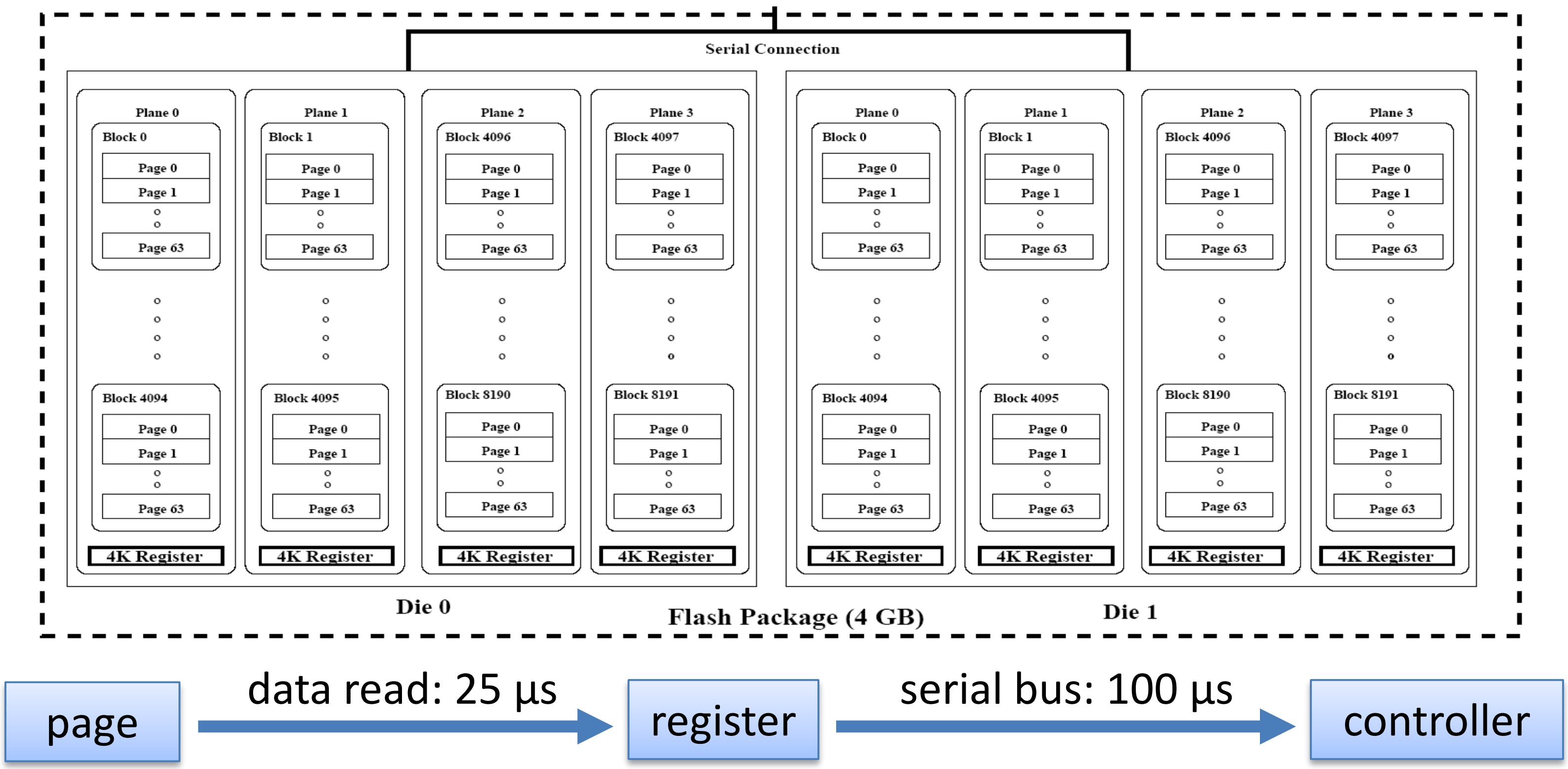
Flash Types

- NAND flash and NOR flash
  - NAND flash: denser capacity, only allow access in units of pages, faster erase operation
  - Most SSD products are based on NAND flash
- NAND flash: SLC and MLC
  - SLC: each cell stores one bit
    - Longer life time, lower access latency, higher cost
  - MLC: each cell stores two (or three) bits
    - Higher capacity

- Solid-state drives (SSDs)
  - SSD architecture
  - **SSD operations**
  - Flash translation layer
  - Research topics

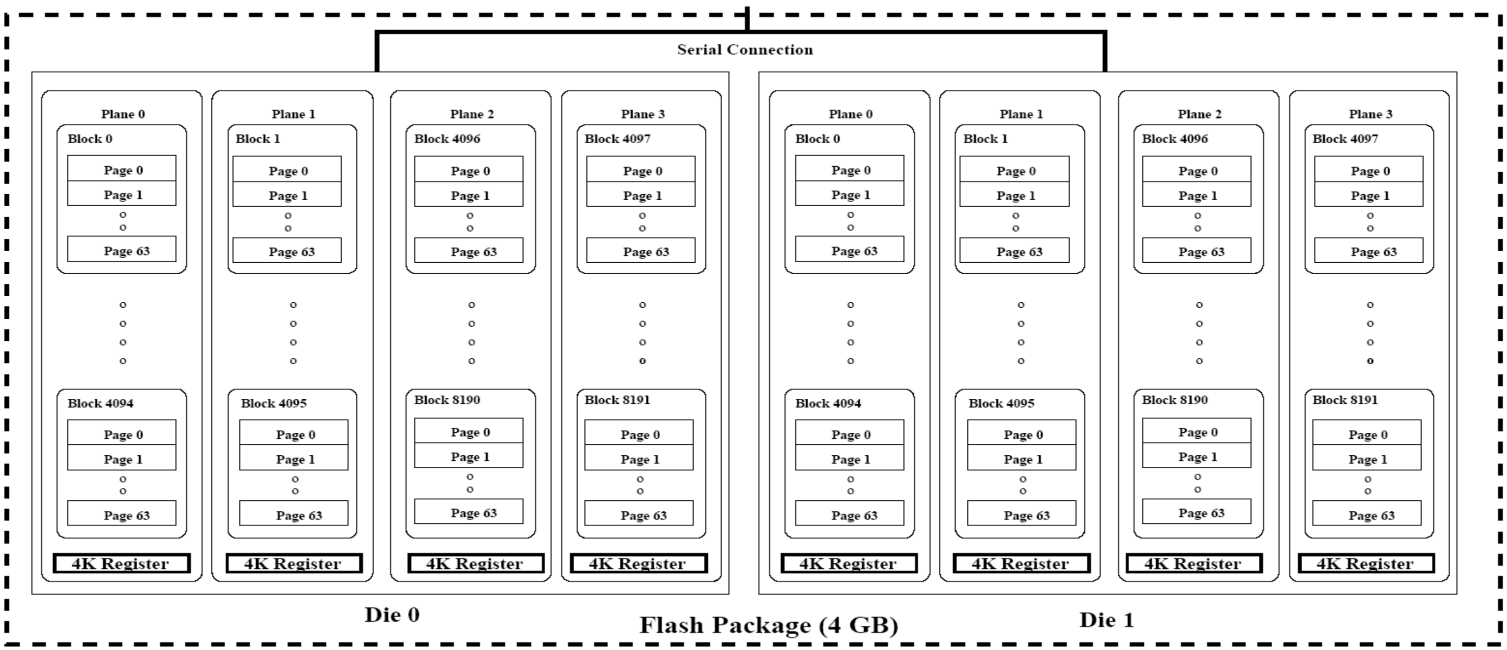
Read

- Read: in unit of pages (4KB)



Write

- Write: in unit of pages (4KB)



page ← program: 200 µs ← register ← serial bus: 100 µs ← controller

Erase

- Erase
  - In unit of blocks (64/128 pages)
  - Change all bits to 1
  - Much slower than read/write: 1.5ms
- Each block can only tolerate limited number of P/E cycles
  - SLC: 100K, MLC: 10K, TLC (several K to several hundred)
- The number of maximum P/E cycles decreases when
  - More bits are stored in one cell
  - The feature size of flash cell decreases (72nm, 34nm, 25nm)

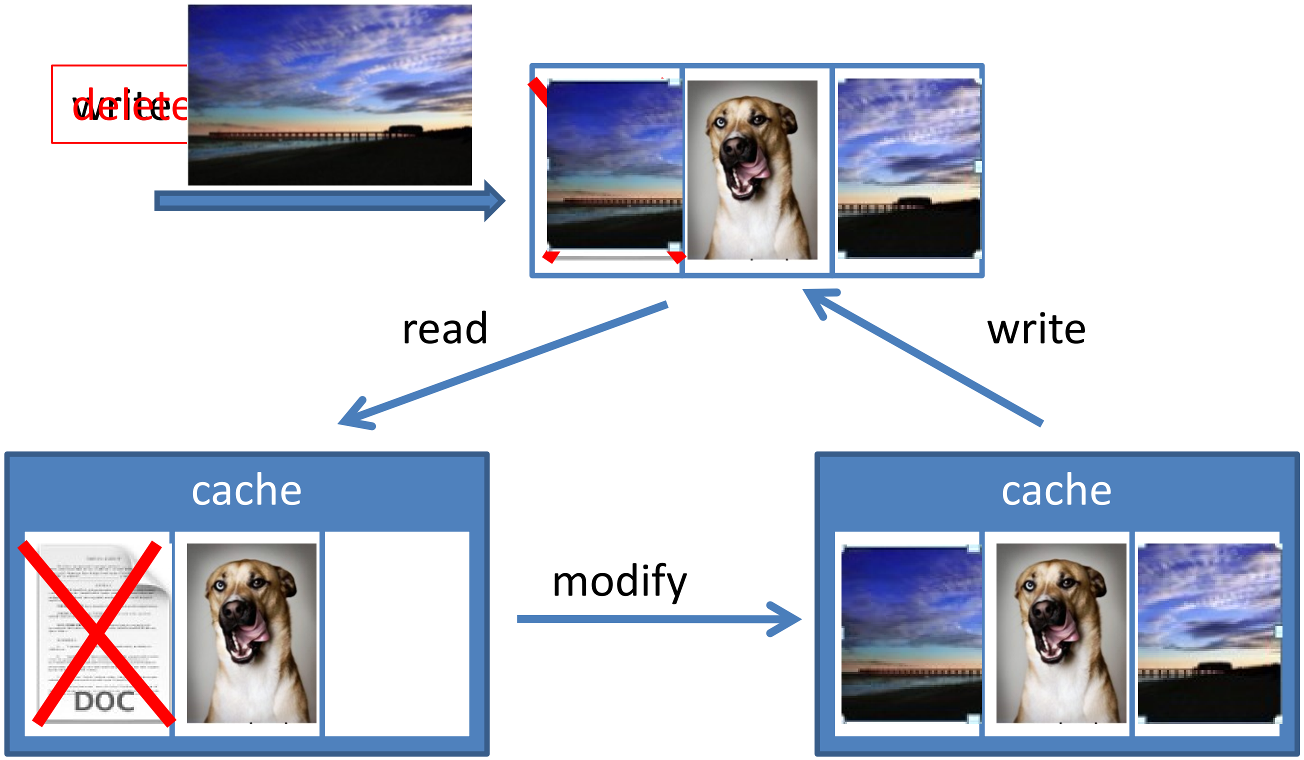
Overwrite & Delete

- Delete
  - Simply mark the page as invalid

- Overwrite/update
  - Does not support in-place overwrite
  - Data can only be programmed to clean pages

- How about read-modify-write?

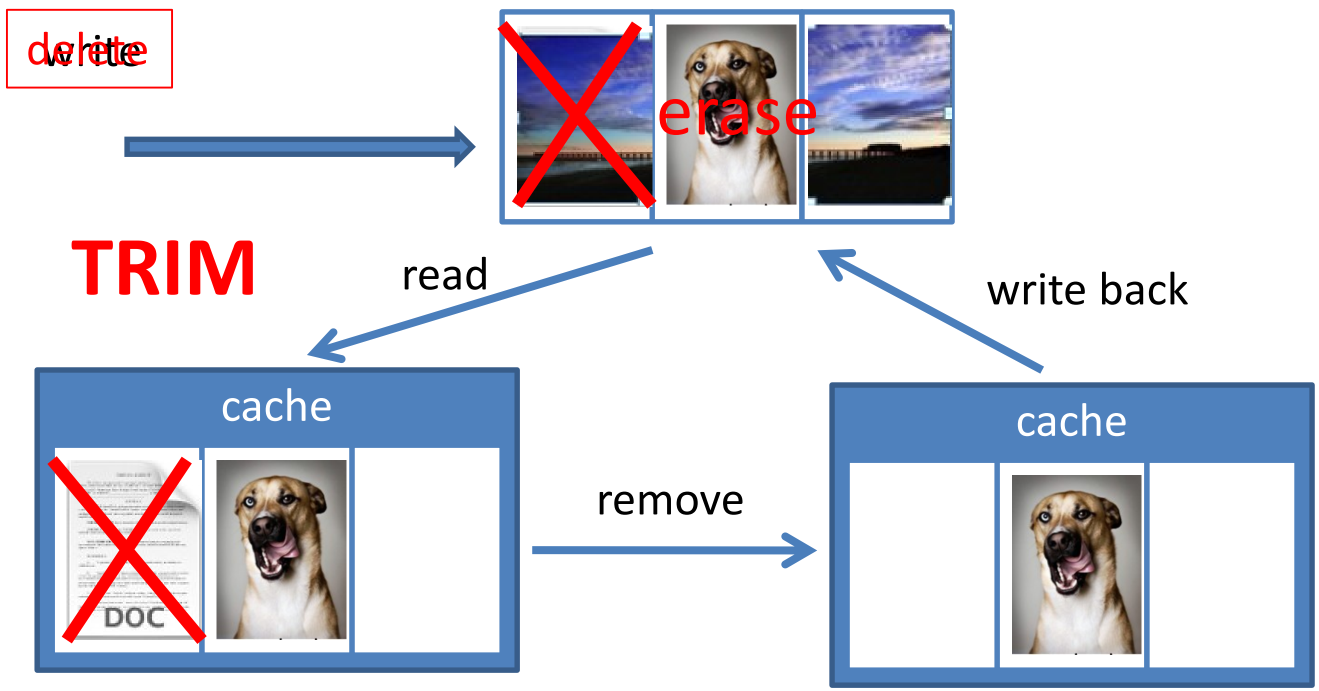
Read-Modify-Write



RMW may require a lot of read and write operations, so it is very slow

Trim

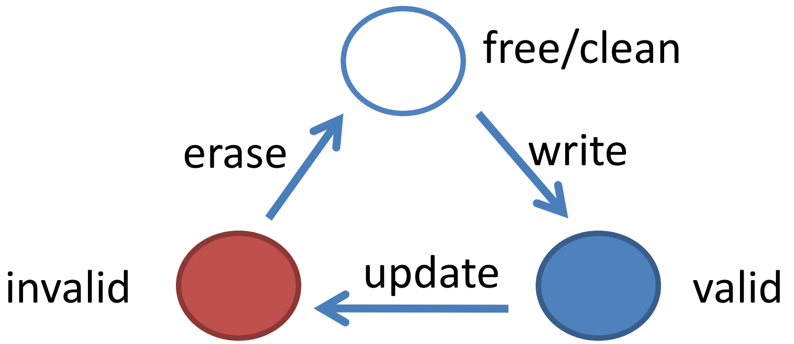
- Improve write performance degraded by RMW
  - The OS also sends a TRIM command to SSD after delete pages
  - Requires both OS and SSD to support



TRIM avoids slow RMW operation during write, so it increases write performance

Software layer in controller

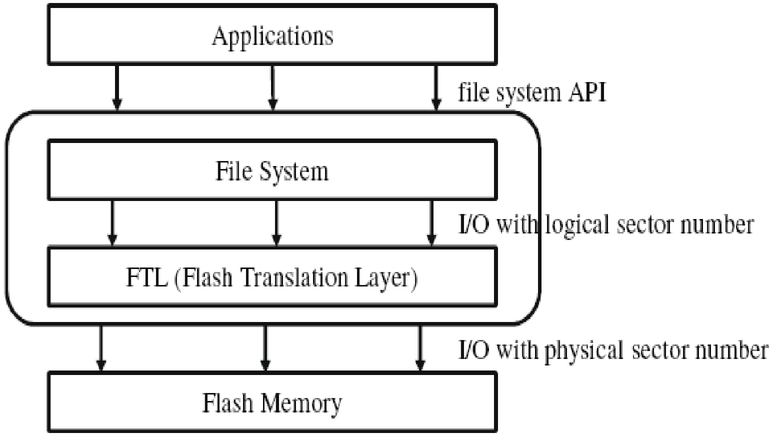
- How to further improve write performance?
  - Address mapping is needed
- Page states
  - Garbage collection is also necessary
- Flash translation layer



- Solid-state drives (SSDs)
  - SSD architecture
  - SSD operations
  - Flash translation layer
  - Research topics

Flash Translation Layer

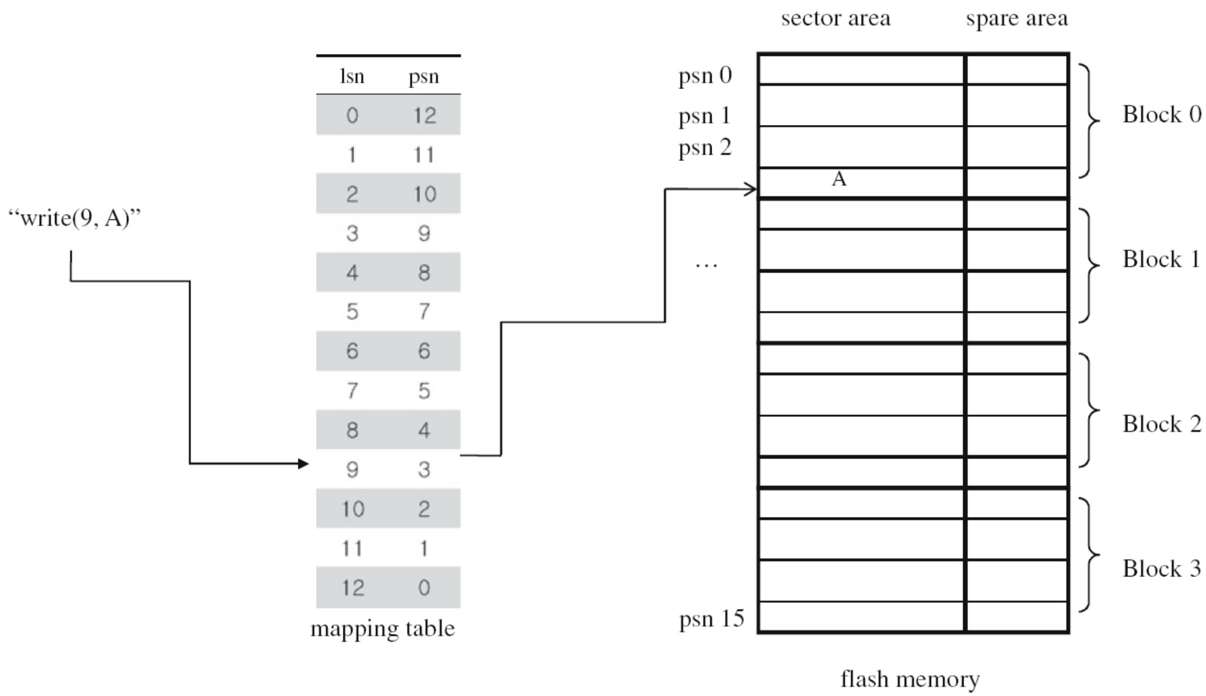
- Three functionalities
  - Address mapping
  - Garbage collection
  - Wear-leveling



Address Mapping

- Sector mapping
- Block mapping
- Hybrid mapping
- Log-structured mapping

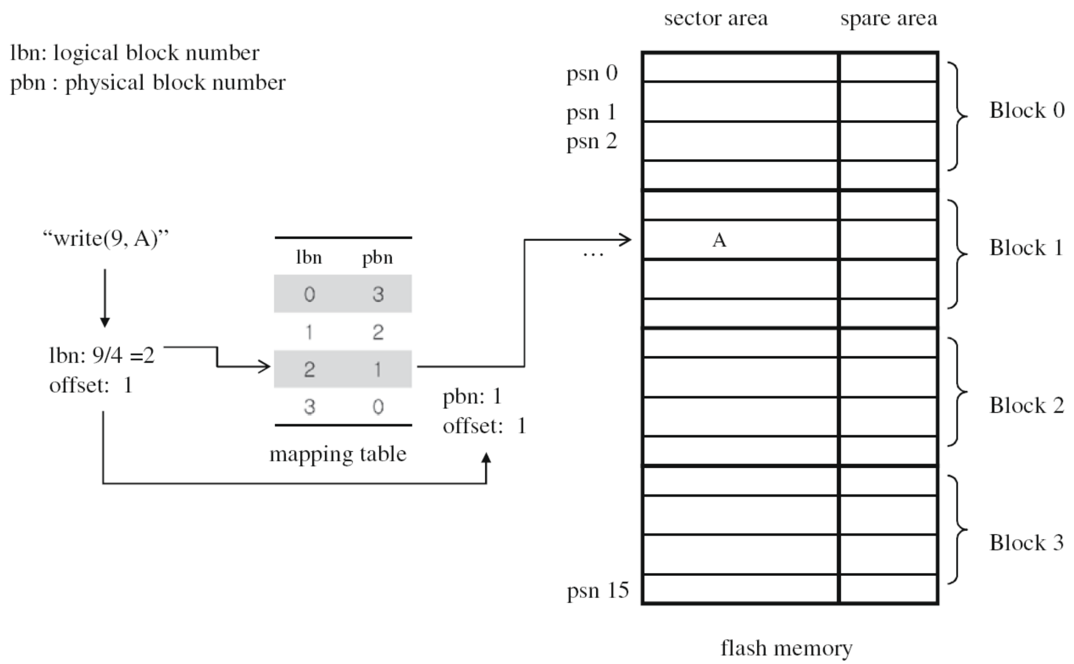
Sector Mapping



Mapping table is large: requires a large amount of RAM

Block Mapping

- The logical sector offset is the same with the physical sector offset

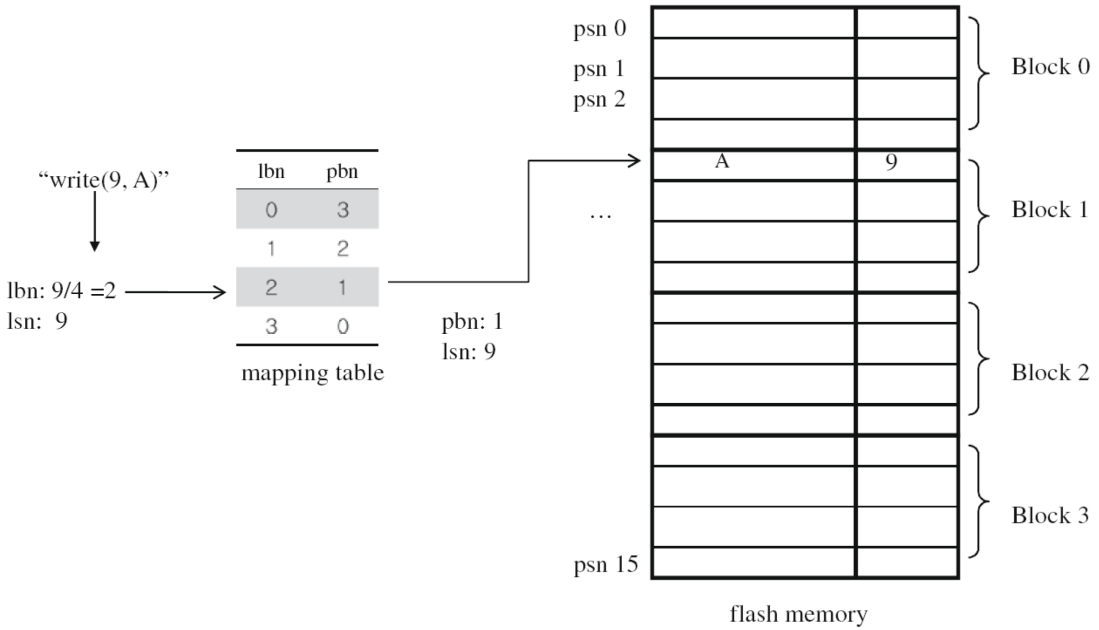


Smaller mapping table

If the FS issues writes with identical lsn, many erases

Hybrid Mapping

- First use block mapping, then use sector mapping in each block

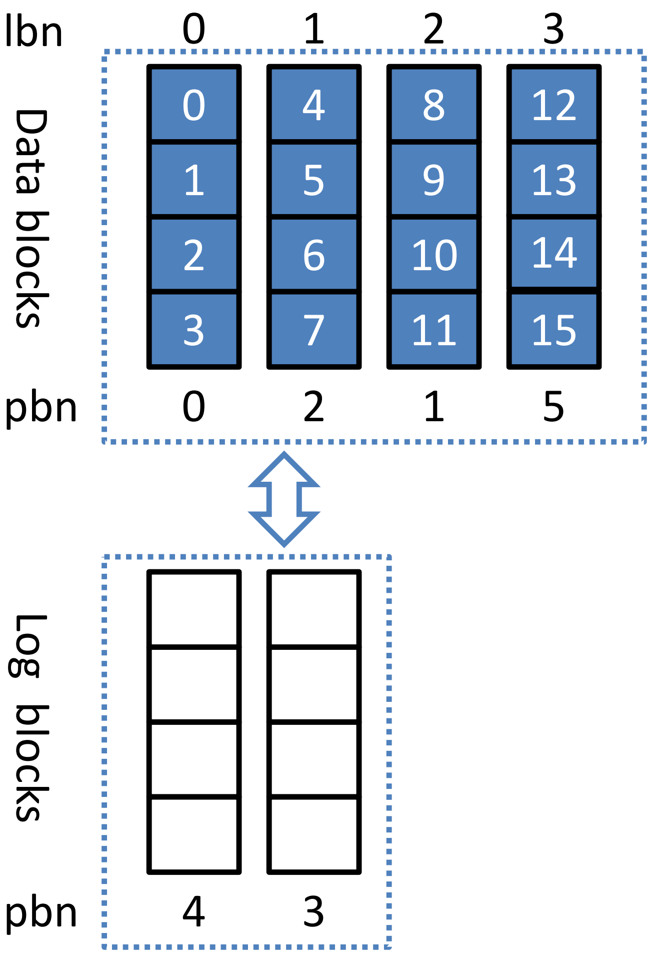


Small mapping table

Avoid a lot of erase operations

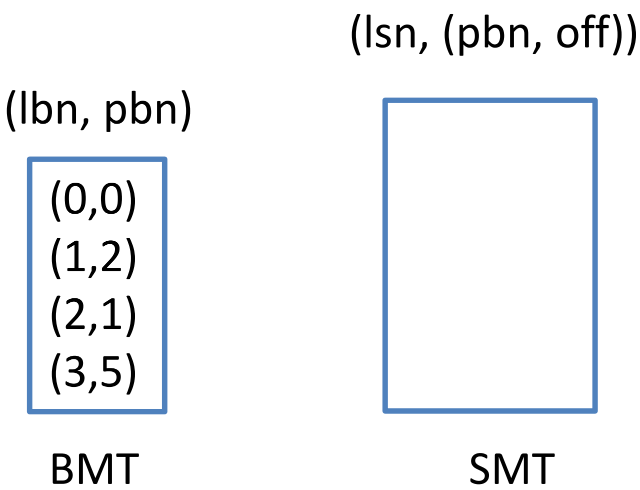
Longer time to identify the location of a page

Log-structured Mapping



Data blocks: block mapping

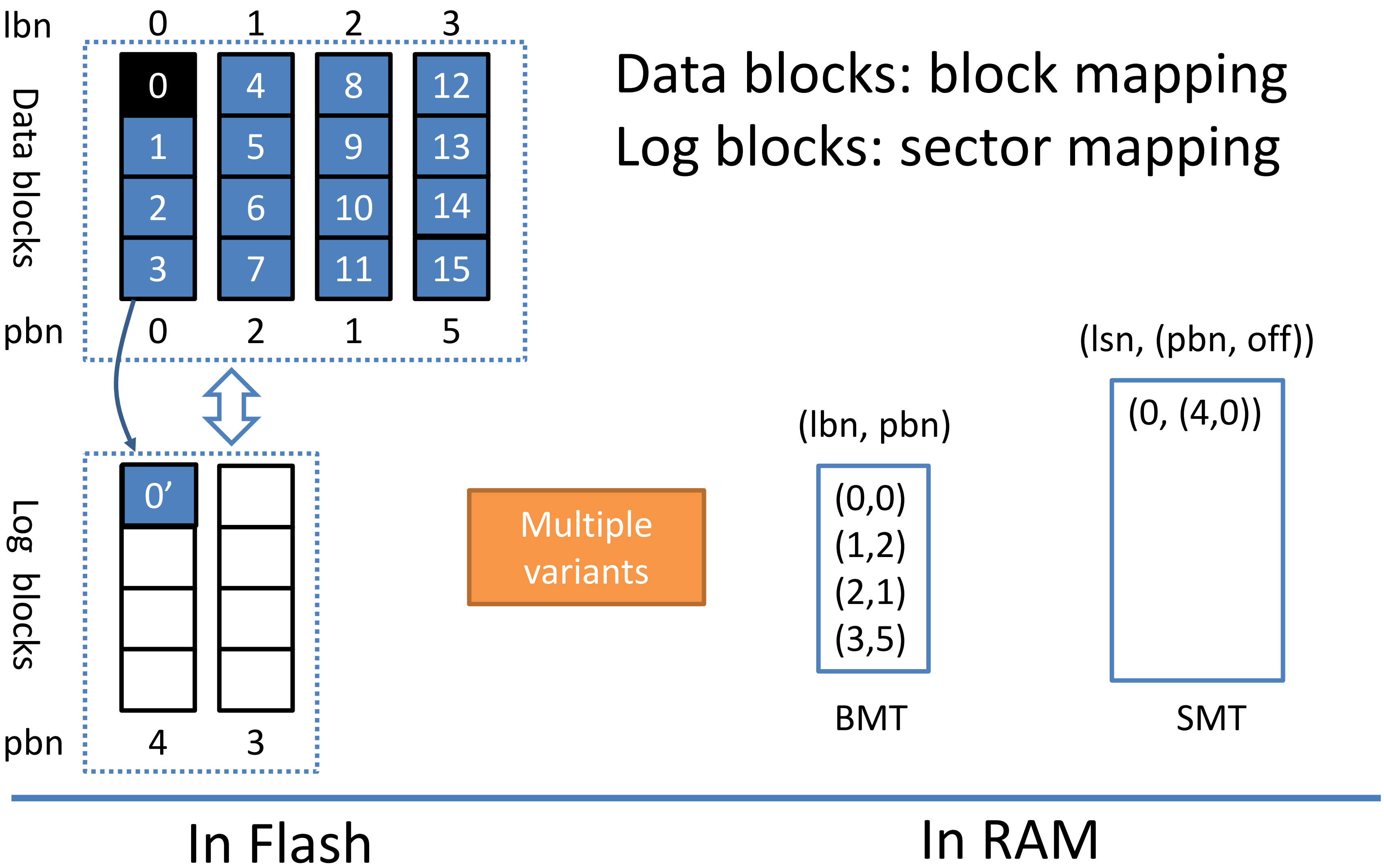
Log blocks: sector mapping



In Flash

In RAM

Log-structured Mapping



Data blocks: block mapping

Log blocks: sector mapping

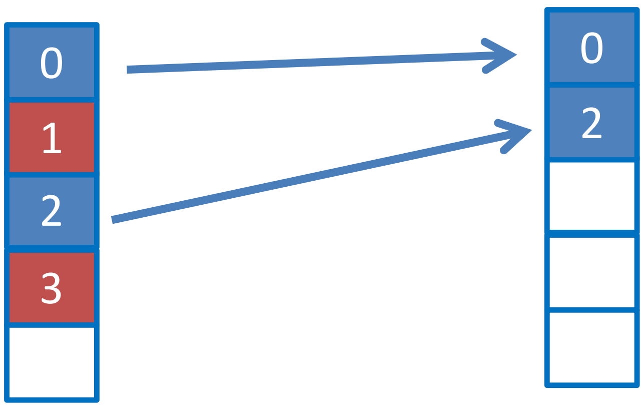
Short summary

- The performance of address mapping is workload dependent
  - Block mapping is suitable for sequential workloads
  - Sector mapping is suitable for random workloads
  - Log-structured mapping is suitable for workloads with large sequential and small random requests

- **Tradeoff exists**

Garbage Collection

- Due to the existence of invalid pages, GC must be called to reclaim storage
  - Choose a candidate block
  - Write valid pages to another free block
  - Erase the original block



Design Issues of GC Algorithms

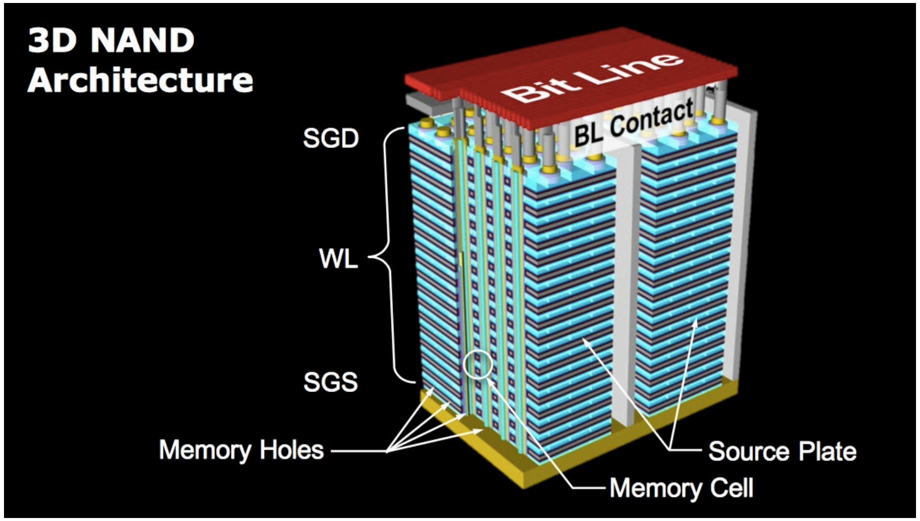
- **Tradeoff** in GC design
  - Efficiency: minimize writes
  - Wear-leveling: erase every block as even as possible
  - Tradeoff
  - GC is considered together with wear-leveling

- Algorithms
  - Greedy, random, and their variants
  - Hot/cold identification

Other Technologies

- Non-volatile memory (NVRAM)
  - PCM, STTRAM, ReRAM, etc…
  - Byte-addressable and non-volatile
  - 3D XPoint

- 3D NAND flash



- Solid-state drives (SSDs)
  - SSD architecture
  - SSD operations
  - Flash translation layer
  - Research topics

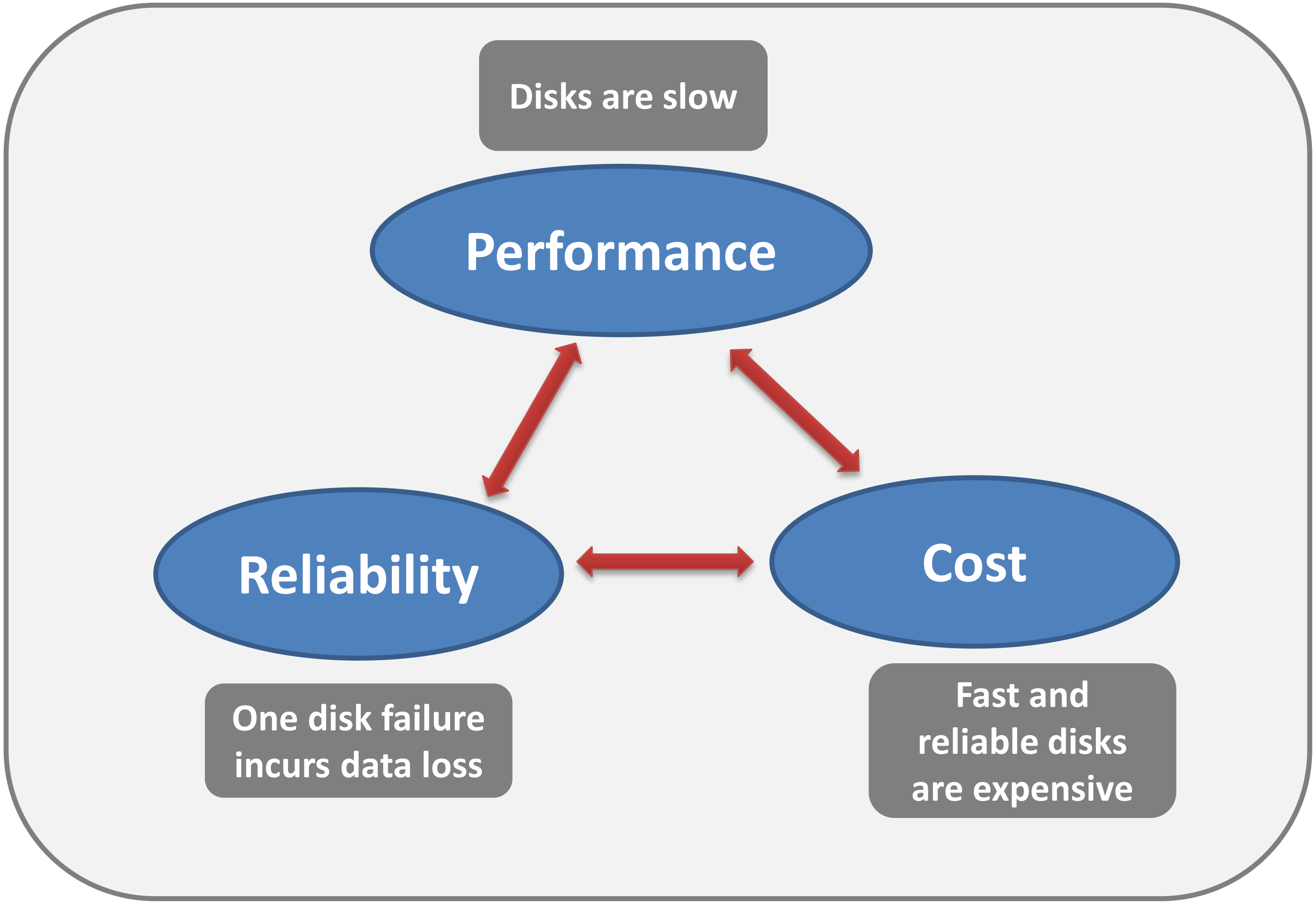
Topics related to SSDs

- Device-level optimization
  - GC/read/write performance
  - Large-capacity SSDs: large address mapping table
- Open-channel SSD
  - Expose the internals of SSDs to the host and allow host to manage it directly
- Computational Storage
- SSD-based system optimization
  - SSD RAID
  - Leverage SSDs in key-value stores, graph systems…
  - Hybrid architecture

Topics

- Disk structure
- Disk scheduling
- Solid-state drives (SSDs)
- **RAID & Erasure codes**
- Problems with EC

RAID Motivation

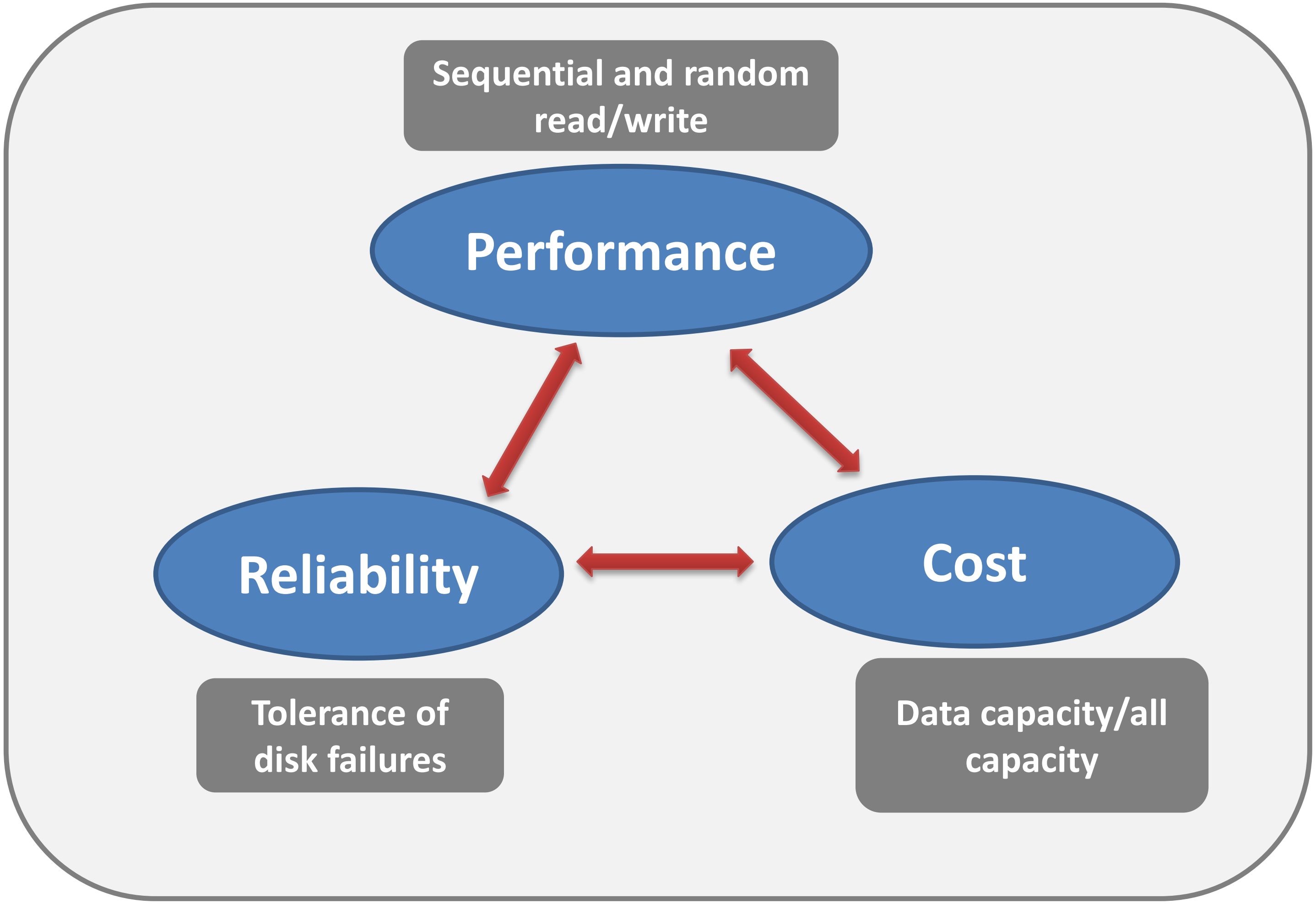


RAID Introduction

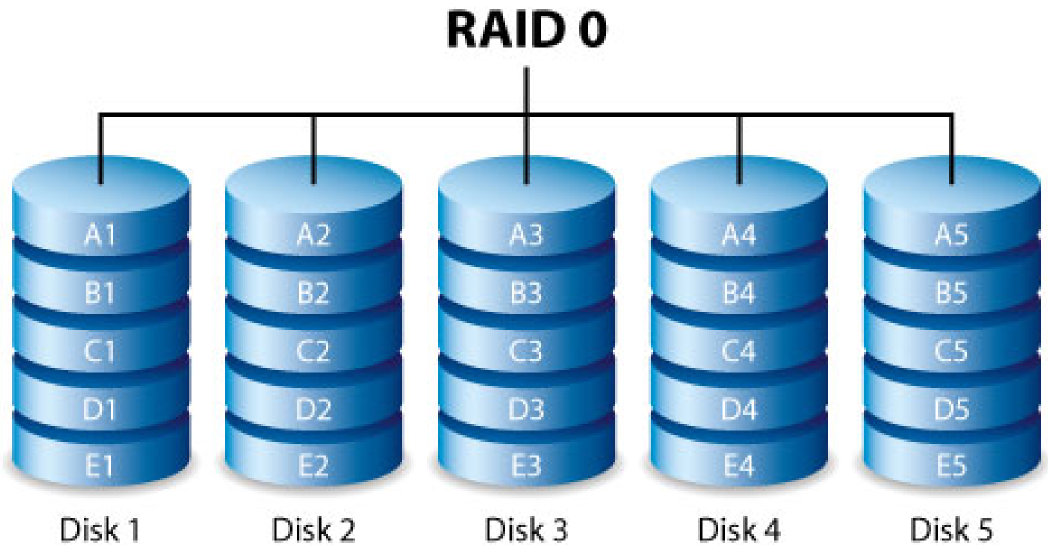
RAID: Redundant Array of Inexpensive (independent) Disks

- ✓ In the past
  - ➢ Combine small and cheap disks as a **cost-effective** alternative to large and expensive disks
- ✓ Nowadays
  - ➢ Higher performance
  - ➢ Higher reliability via redundant data
  - ➢ Larger storage capacity
- ✓ Many different levels of RAID systems
  - ➢ Different levels of redundancy, capacity, cost…

RAID Evaluation



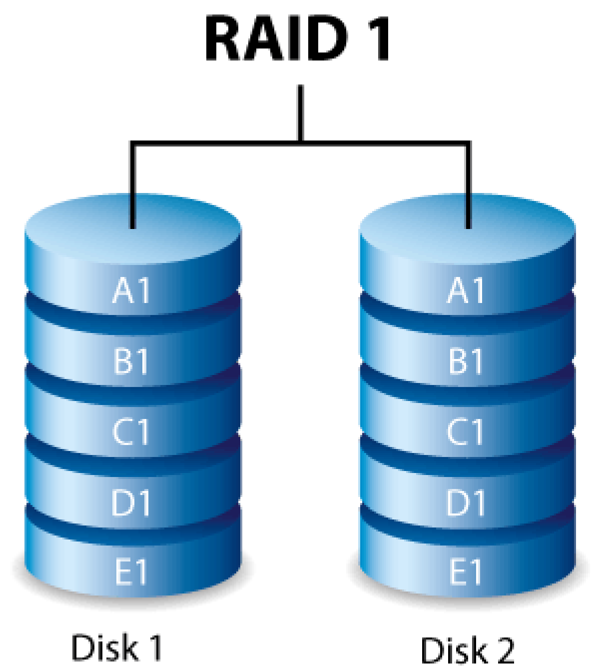
RAID 0



- Block-level striping, no redundancy
- Provides higher data-transfer rate
- Does not improve reliability. **Once a disk fails, data loss may happen (MTTF: mean time to failure)**

RAID 1

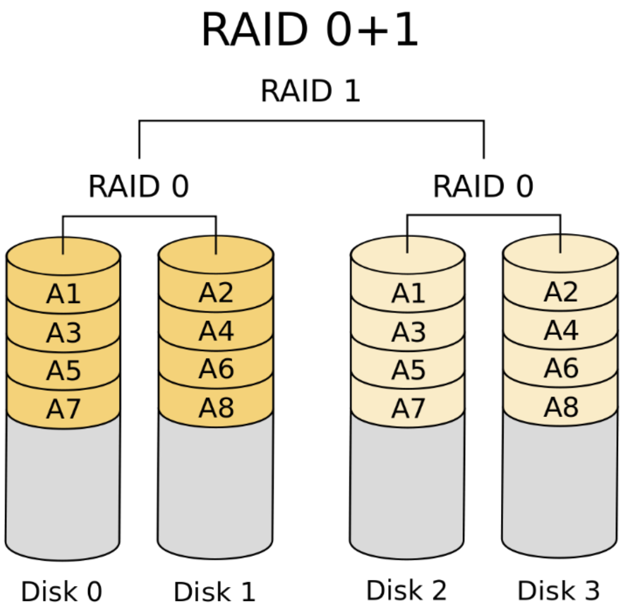
- How to improve reliability?
- Data mirroring (RAID1)
  - ✓ Two copies of the data are held on two physical disks, and the data is always identical.
  - ✓ Replication
- High storage cost
  - ✓ Twice as many disks are required to store the same data when compared to RAID 0.
  - ✓ Even worse storage efficiency with more copies



Combinations

- RAID 0 provides reliability and RAID 1 provides reliability

- RAID 0+1 (RAID01)
  - ✓ First data striping
  - ✓ Then data mirroring



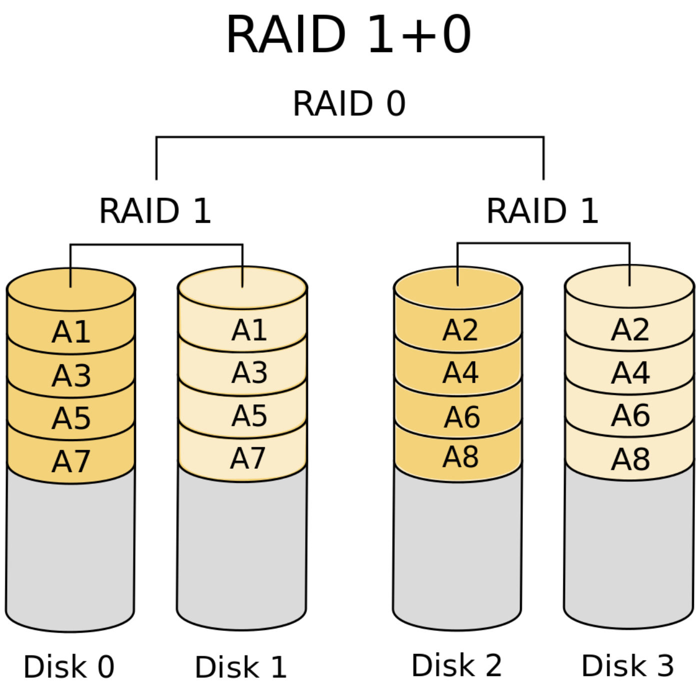
Same storage cost as RAID 1

Combinations

- RAID 0 provides reliability and RAID 1 provides reliability

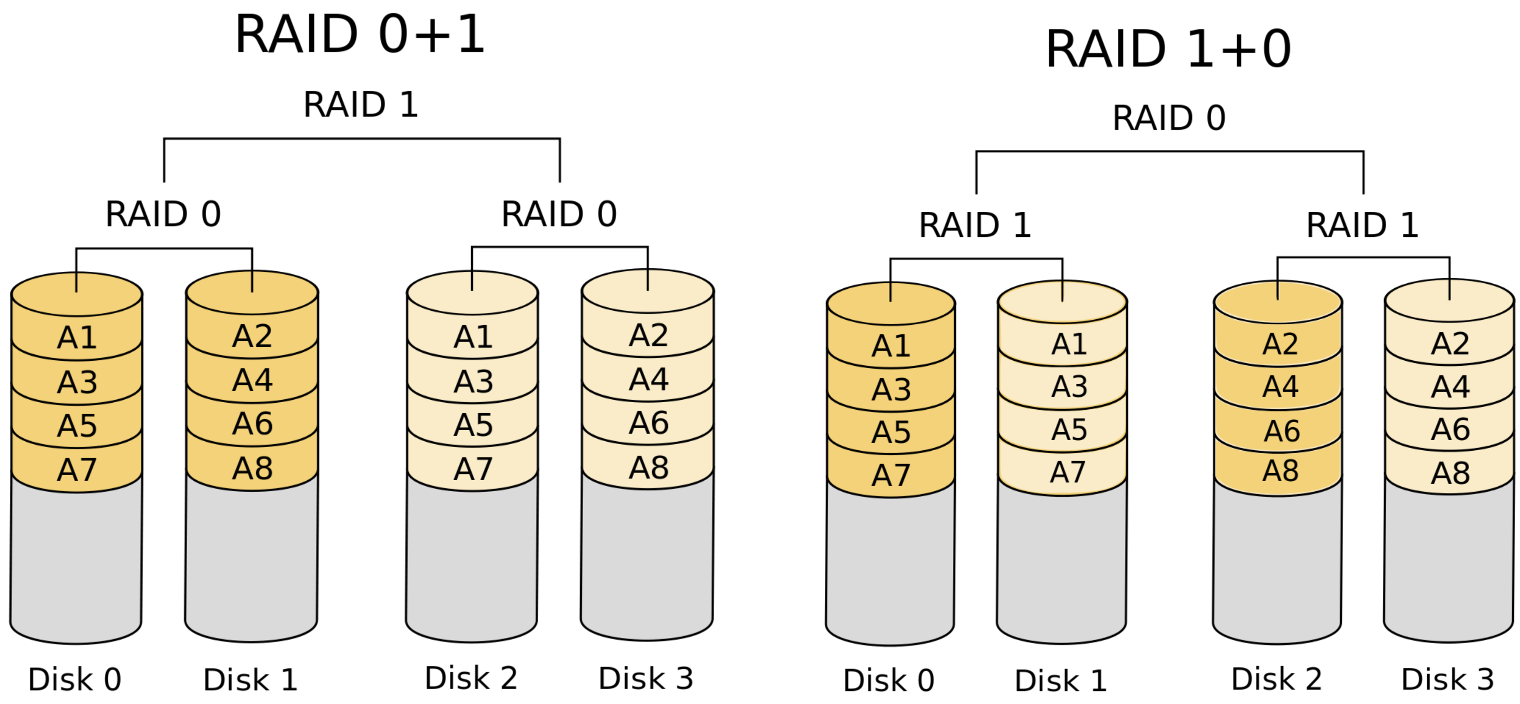
- RAID 0+1 (RAID01)
  - ✓ First data striping
  - ✓ Then data mirroring

- RAID 1+0 (RAID10)
  - ✓ First data mirroring
  - ✓ Then data striping



Same storage cost

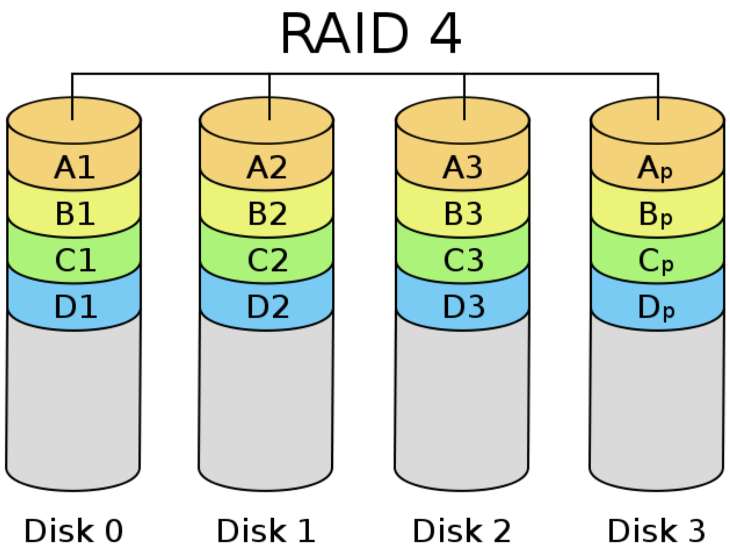
RAID01 vs RAID10



Both suffer from high storage cost

RAID 4

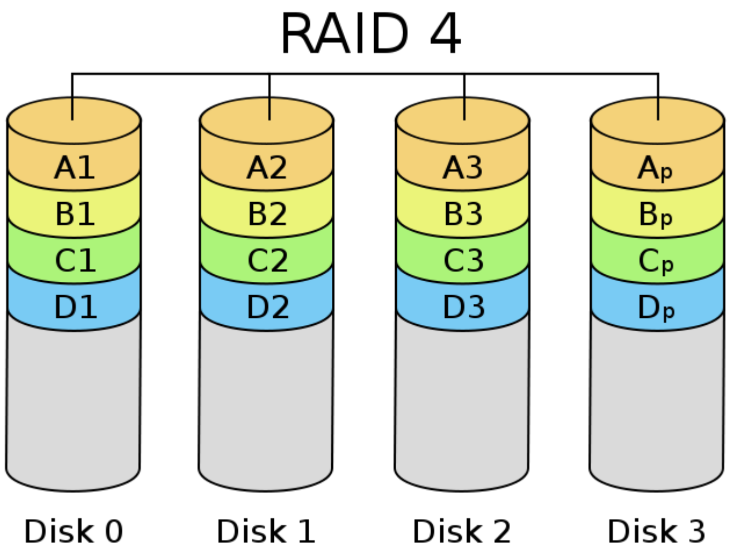
- Balance the tradeoff between reliability and storage cost?
  - Redundancy with parities
- **Parity generation:** Each parity block is the XOR value of the corresponding data disks
- **Block-level data striping**
  - Data and parity blocks are distributed across disks
  - Dedicated parity disk
- Any problem?



$$A_p = A1 \otimes A2 \otimes A3$$

How to update data

- Suppose A1 will be updated to A1’
  - Both A1 and Ap need to be updated
  - Read-modify-write (RMW)

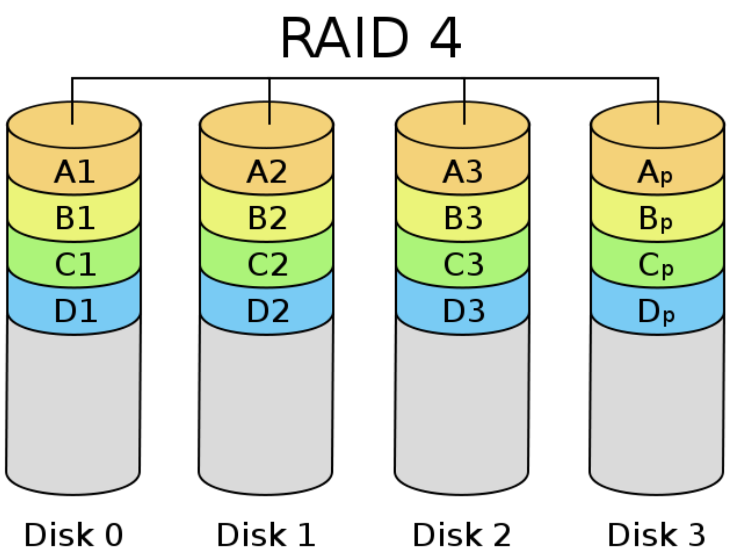


RMW: $A_p' = A_p \otimes A1 \otimes A1'$

$$A_p' = A1 \otimes A2 \otimes A3 \otimes A1 \otimes A1'$$
$$= A2 \otimes A3 \otimes A1'$$

How to update data

- Suppose A1 will be updated to A1'
  - Both A1 and Ap need to be updated
  - Read-modify-write (RMW)
- How about updating both A1 and A2 simultaneously?
  - RMW?
  - Read-reconstruct-write (RRW)
- Selection of RMW/RRW

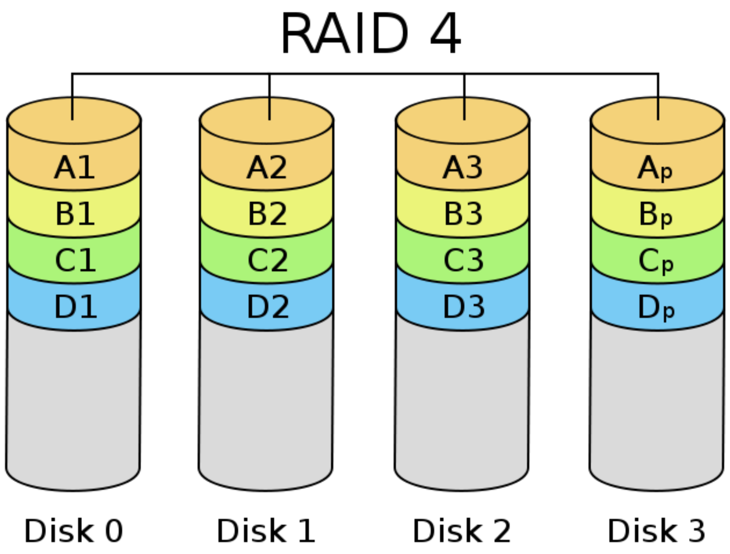


RRW: $A_p' = A3 \otimes A1' \otimes A2'$

Both RMW and RRW incur extra reads and writes

Problems of RAID 4

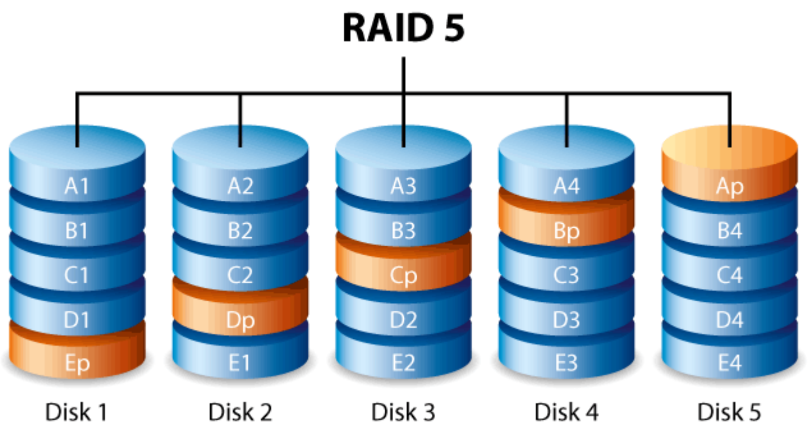
- Problems of RAID 4
- Disk bandwidth are not fully utilized
  - Parity disk will not be accessed under normal mode
- Parity disk may become the bottleneck
  - E.g., updating A1, B2, C3



Read: A1, B2, C3, **Ap, Bp, Cp**
Write: A1’ B2’, C3’, **Ap’, Bp’, Cp’**

RAID 5

- Similar to RAID 4
  - One parity per stripe
- Key difference
  - Uniform parity distribution
- RAID 5 is an ideal combination of
  - good performance
  - good fault tolerance
  - high capacity
  - storage efficiency



$$A_P = A_1 \oplus A_2 \oplus A_3 \oplus A_4$$

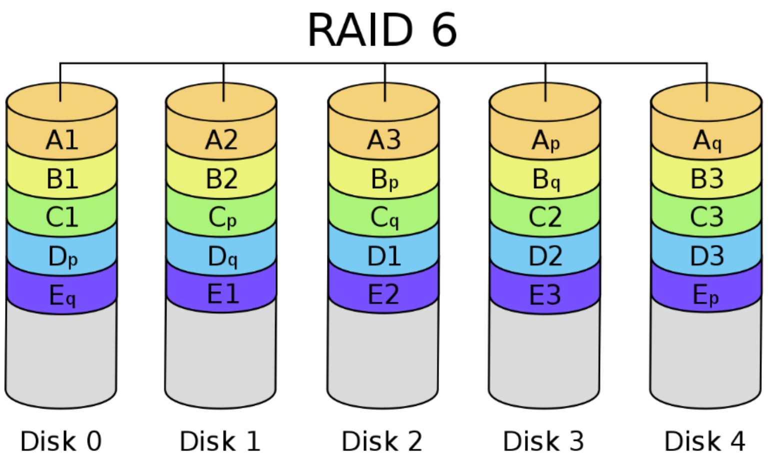
$$\vdots$$

$$E_P = E_1 \oplus E_2 \oplus E_3 \oplus E_4$$

Parity update overhead still exist

RAID 6

- How to tolerate more disk failures?
- RAID-6 protects against two disk failures by maintaining two parities
- Encoding/decoding operations:
  - ➢ Based on **Galois field**



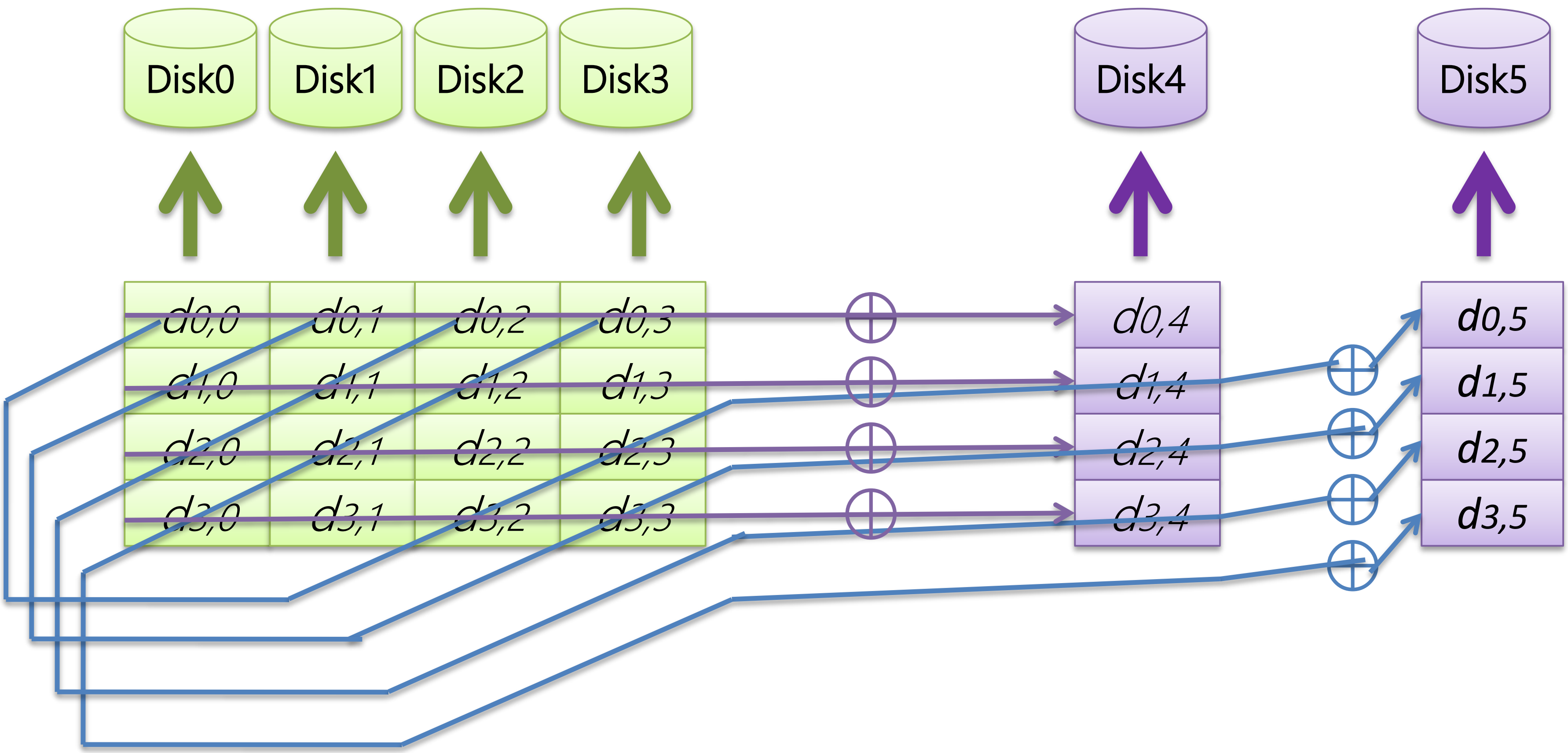
$$A_P = A_1 \oplus A_2 \oplus A_3 \oplus A_4$$

$$A_q = c^0 A_1 \oplus c^1 A_2 \oplus c^2 A_3 \oplus c^3 A_4$$

Parity update overhead becomes larger

RDP Code

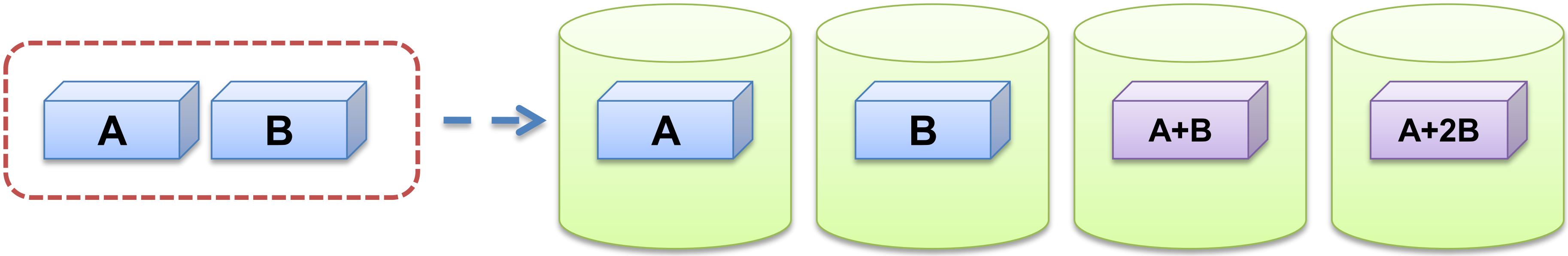
- An RDP code example with 6 disks



Erasure Codes

- Erasure codes
  - Different redundancy levels
    - **2-fault tolerant**: RDP, EVENODD, X-Code
    - **3-fault tolerant**: STAR
    - **General-fault tolerant**: Cauchy Reed-Solomon (CRS)

- Generate **m** code blocks from **k** data blocks, so as to tolerate any **m** disk failures



Summary on Erasure Codes

- The motivation to introduce erasure codes in large-scale storage systems

The need to reduce the tremendous cost of storage

- In practice, erasure codes have seen widely deployment
  - Google File System [Ford, OSDI'10]
  - Windows Azure Storage [Huang, ATC'12]
  - Facebook [Borthakur, Hadoop User Group Meeting 2010]
  - …

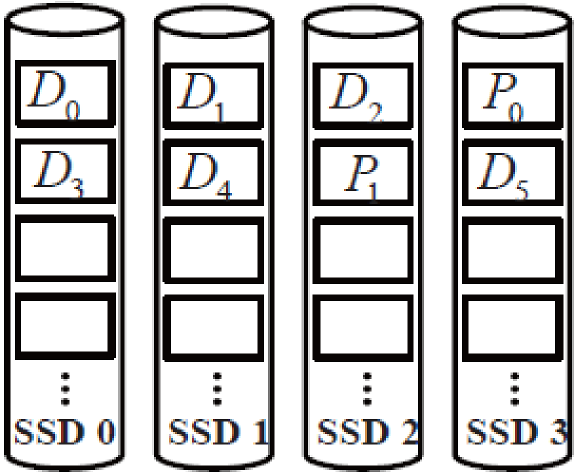
Topics

- Problems with RAID/EC
  - Optimizing parity updates
  - Recovery
  - Asynchronous coding
  - …

SSD RAID

- RAID provides device-level fault tolerance
  - Each stripe contains data and parity

- Limitation: Parity updates
  - Update data -> update parity
    - Update D_1 to $D_1{}'$
    - RMW: $P_0' = P_0 \oplus D_1 \oplus D_1{}'$
    - RRW: $\mathrm{P}_0' = \mathrm{D}_0 \oplus D_1{}' \oplus D_2$
  - Extra I/Os and GC



Parity chunks:
$P_0 = D_0 \oplus D_1 \oplus D_2$
$P_1 = D_3 \oplus D_4 \oplus D_5$

- SSD RAID
  - Parity update influences both performance and endurance

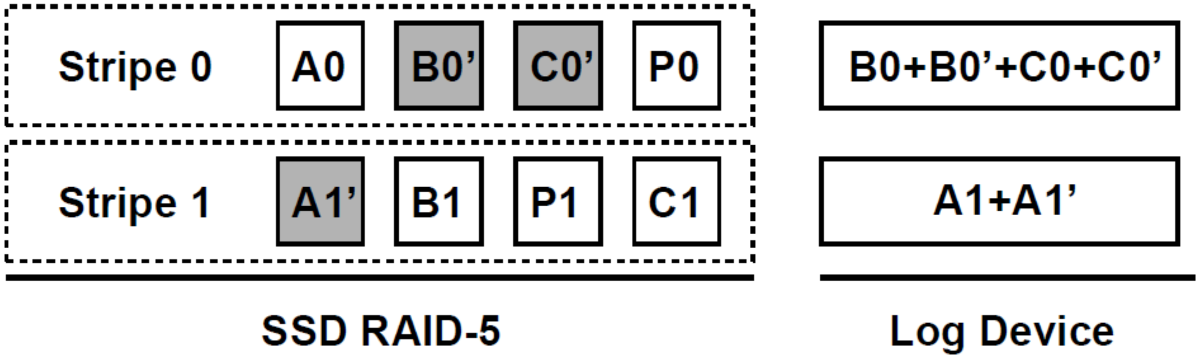
Design tradeoff

- Design trade-off in SSD RAID arrays
  - RAID improves reliability
  - Parity updates incur extra I/Os and GC operations
    - Degrade performance and endurance

How to address the parity update overhead?

Parity Logging

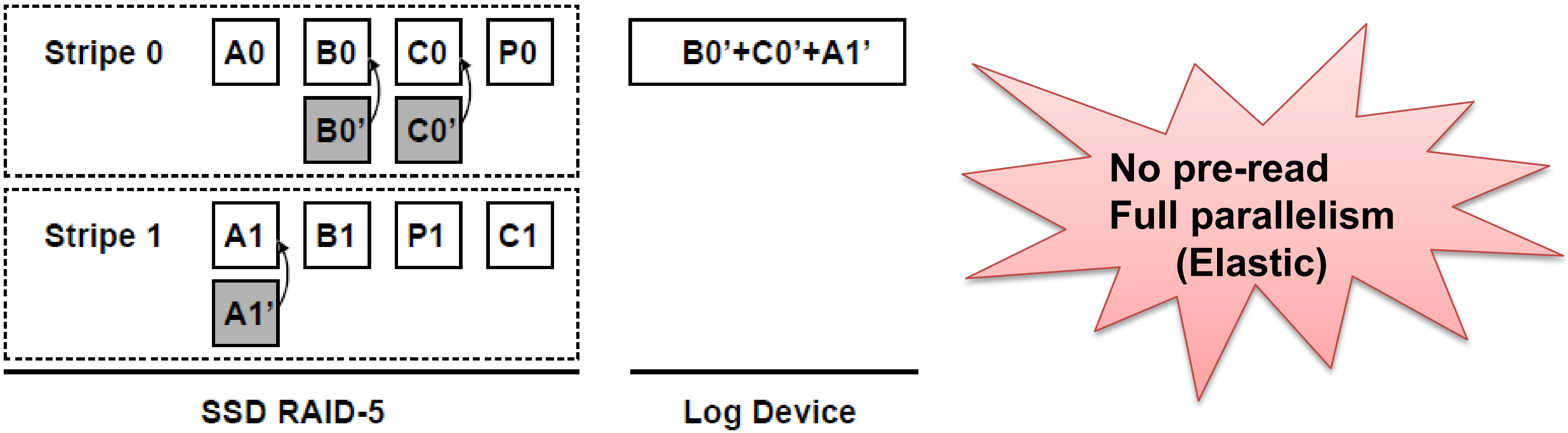
- Original Parity logging
  - Incoming reqs: $\{A_0, B_0, C_0\}, \{A_1, B_1, C_1\}, \{B_0', C_0', A_1'\}$



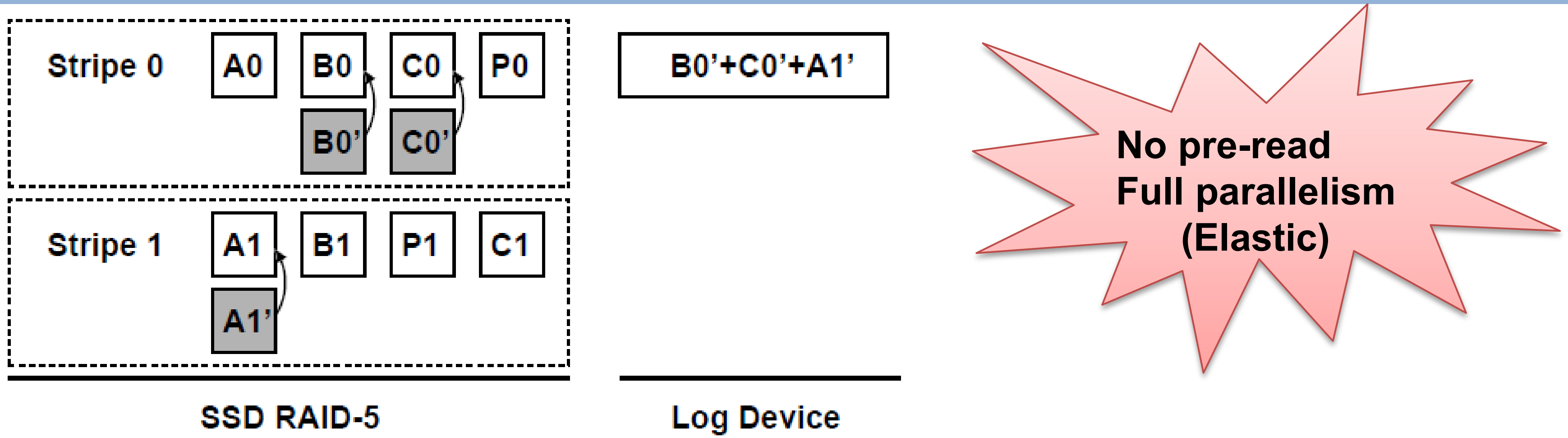
- Drawbacks
  - Pre-read: Extra reads
  - Per-stripe basis: Extra log chunks; Partial parallelism

EPLOG

Our solution: New RAID Design via Elastic Parity Logging (EPLOG)



EPLOG



- Benefits of EPLOG
  - General RAID
  - High endurance: Reduce parity writes to SSDs
  - High performance: Reduce extra I/Os
  - Low-cost deployment: Commodity hardware

✓ Yongkun Li, Helen H. W. Chan, Patrick P. C. Lee, and Yinlong Xu. "Elastic Parity Logging for SSD RAID Arrays." IEEE/IFIP DSN (Regular paper), Toulouse, France, June 2016.

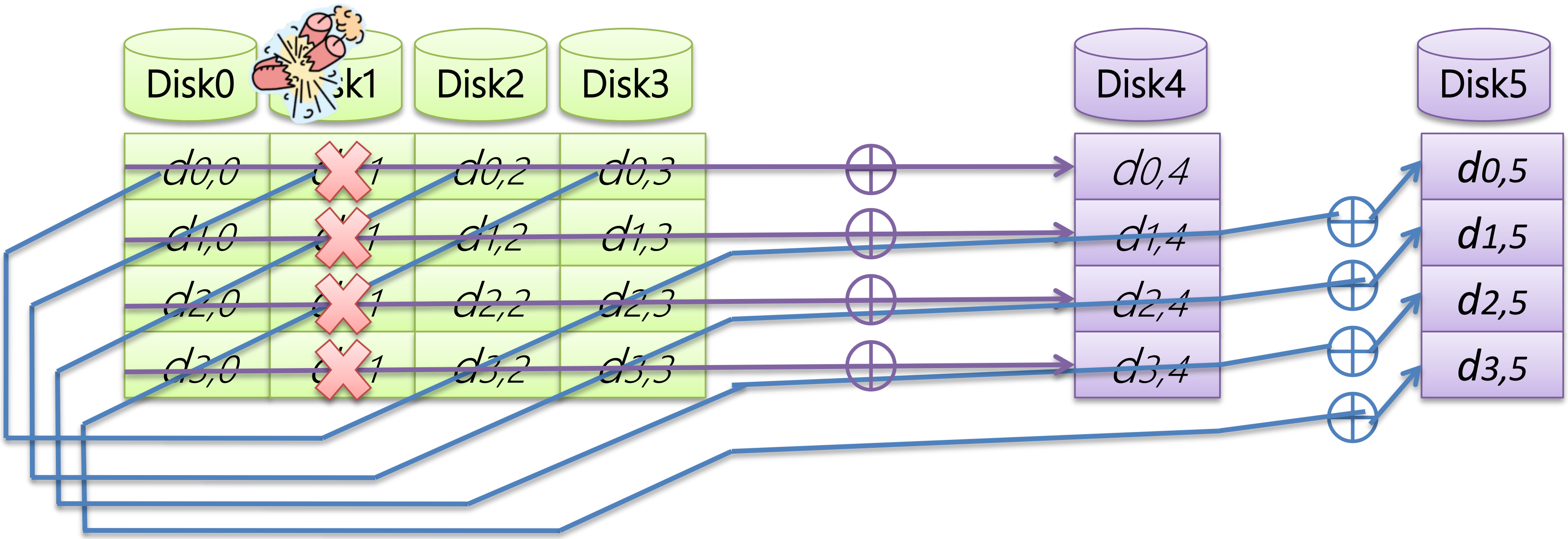
✓ Helen H. W. Chan,Yongkun Li, Patrick P. C. Lee, and Yinlong Xu."Elastic Parity Logging for SSD RAID Arrays: Design, Analysis, and Implementation." IEEE TPDS, volume: 29 , issue: 10 , Oct. 2018.

Topics

- Problems with RAID/EC
  - Optimizing parity updates
  - Recovery
  - Asynchronous coding
  - …

Failure Recovery Problem

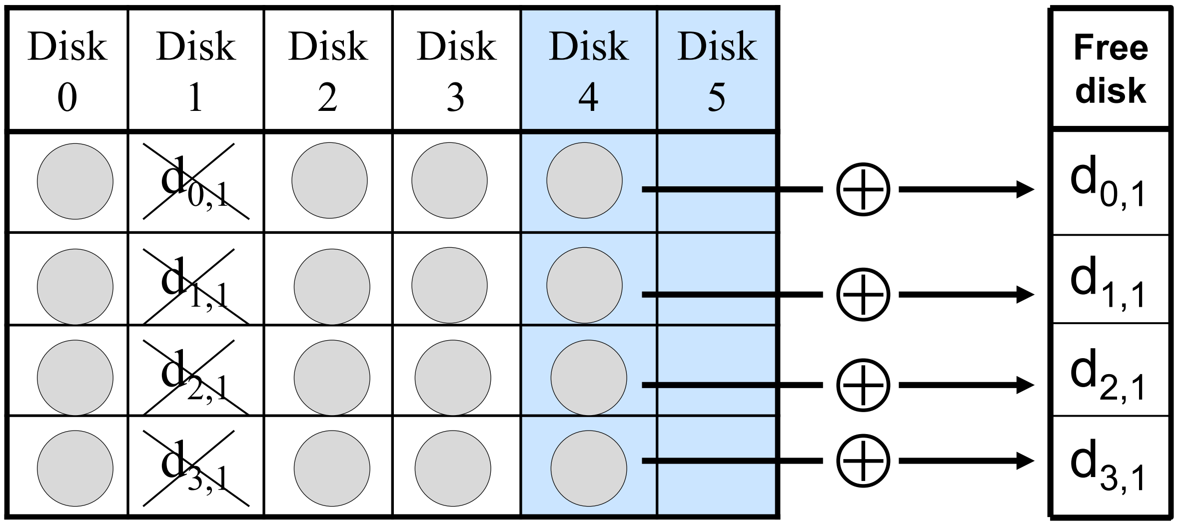
- Recovering disk failures is necessary
  - Preserve the required redundancy level
- Single-disk failure recovery
  - Single-disk failure occurs more frequently than a concurrent multi-disk failure



Suppose Disk 1 fails. How do we recover Disk 1 efficiently?

Optimize Recovery Performance

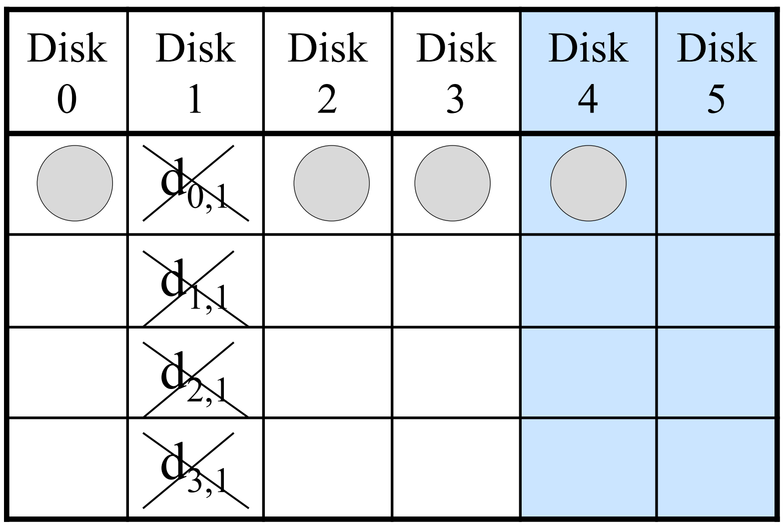
- Traditional method: only use row blocks for repair.



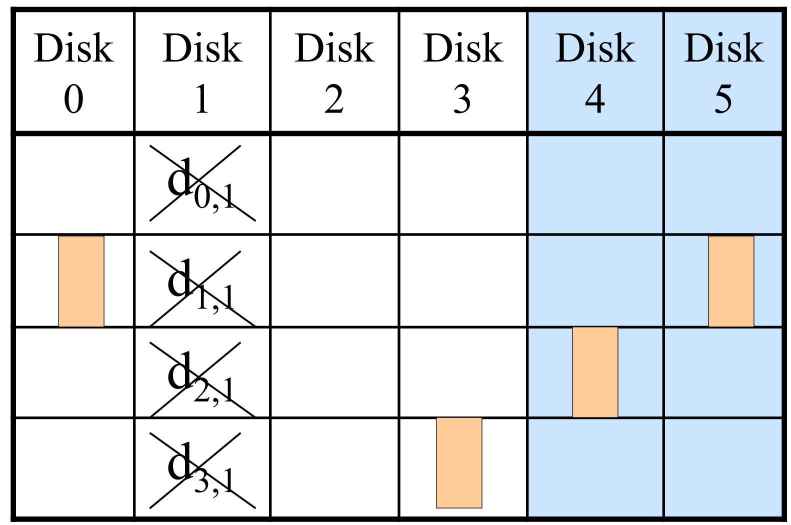
- Example: $\boldsymbol{d_{0,1} = d_{0,0} \oplus d_{0,2} \oplus d_{0,3} \oplus d_{0,4}}$
- Need read $(p-1)^2$=16 blocks

Optimize Recovery Performance

- Recovery choices: row blocks or diagonal blocks



Repair $d_{0,1}$ from row blocks

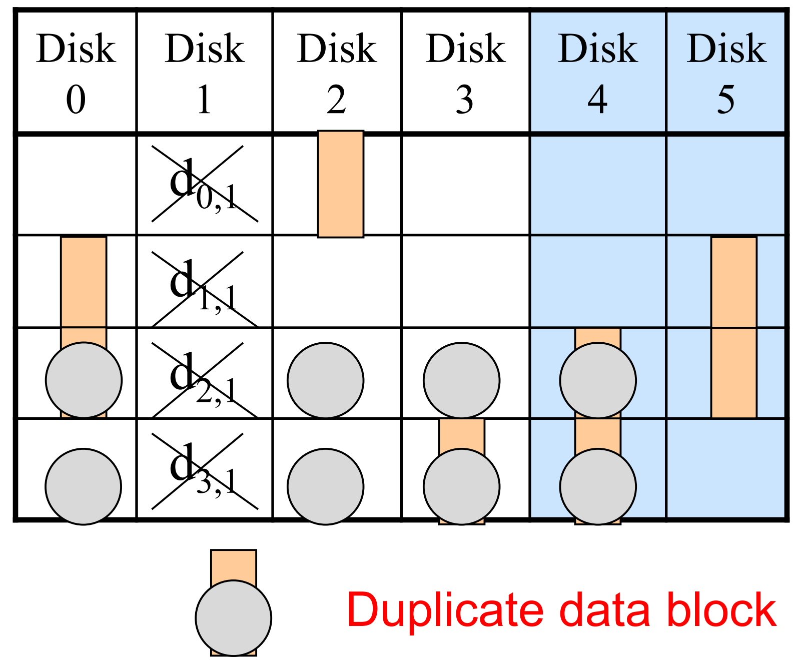


Repair $d_{0,1}$ from diagonal blocks

Single Disk Failure Hybrid Recovery

- Recover Disk 1.

Failure blocks	Recover choices
$d_{0,1}$	diagonal
$d_{1,1}$	diagonal
$d_{2,1}$	row
$d_{3,1}$	row



- ❖ The four blocks are repeated twice.
- ❖ Result: Need read 16–4=12<16 block for recovery.

Xiang, L., Xu, Y., Lui, J., Chang, Q. "Optimal recovery of single disk failure in RDP code storage systems". *ACM SIGMETRICS 2010*.

Hybrid Recovery

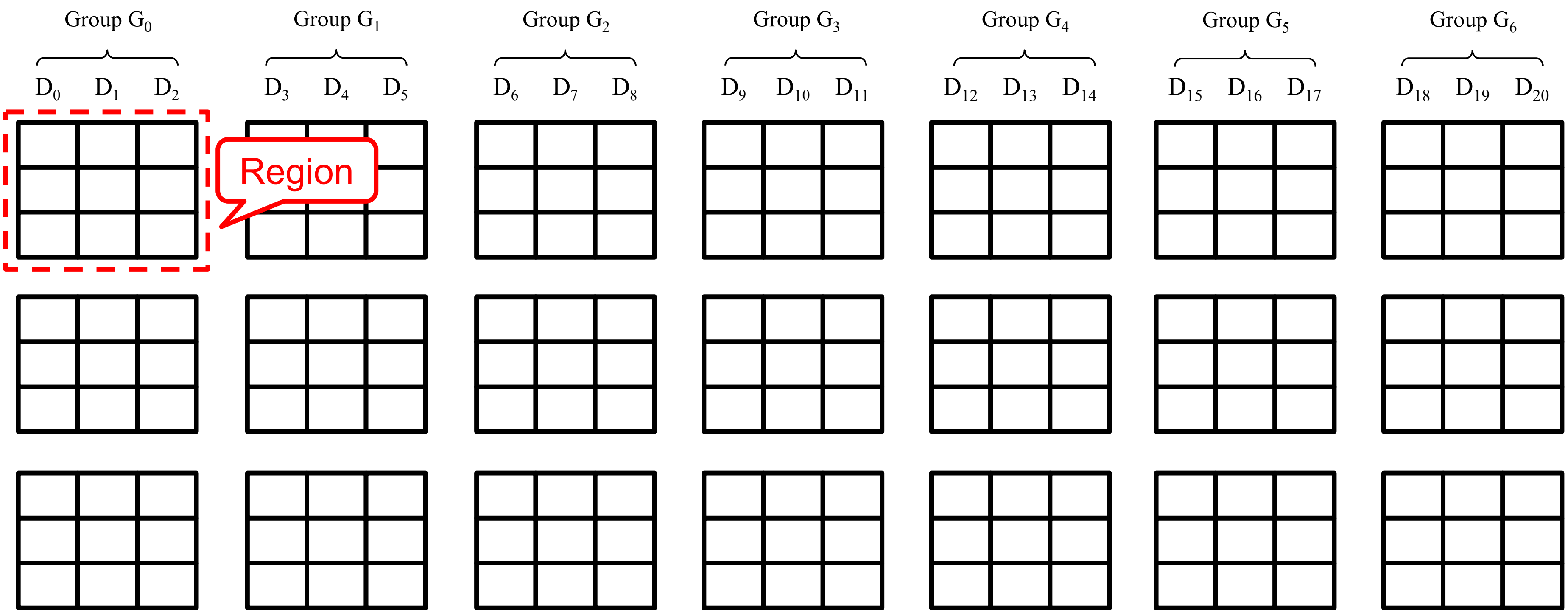
Previous approach leverages the code property, but not change the code

Alternative approach

Can we design new codes which benefit the recovery performance?

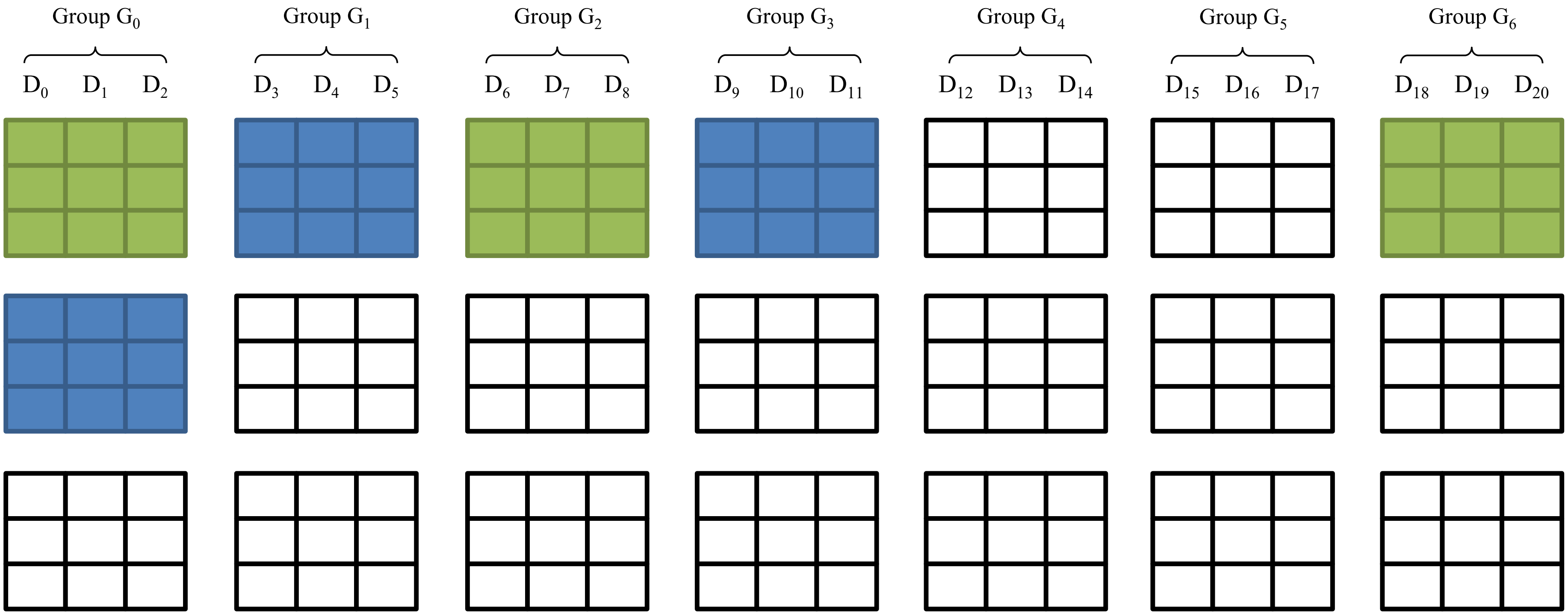
Yes! Our solution: OI-RAID

OI-RAID: An Example



- Divide 21 disks into 7 **groups**
- Divide each disk into 9 **storage units**
- Form a **region** with every 3×3 storage unit array in a group

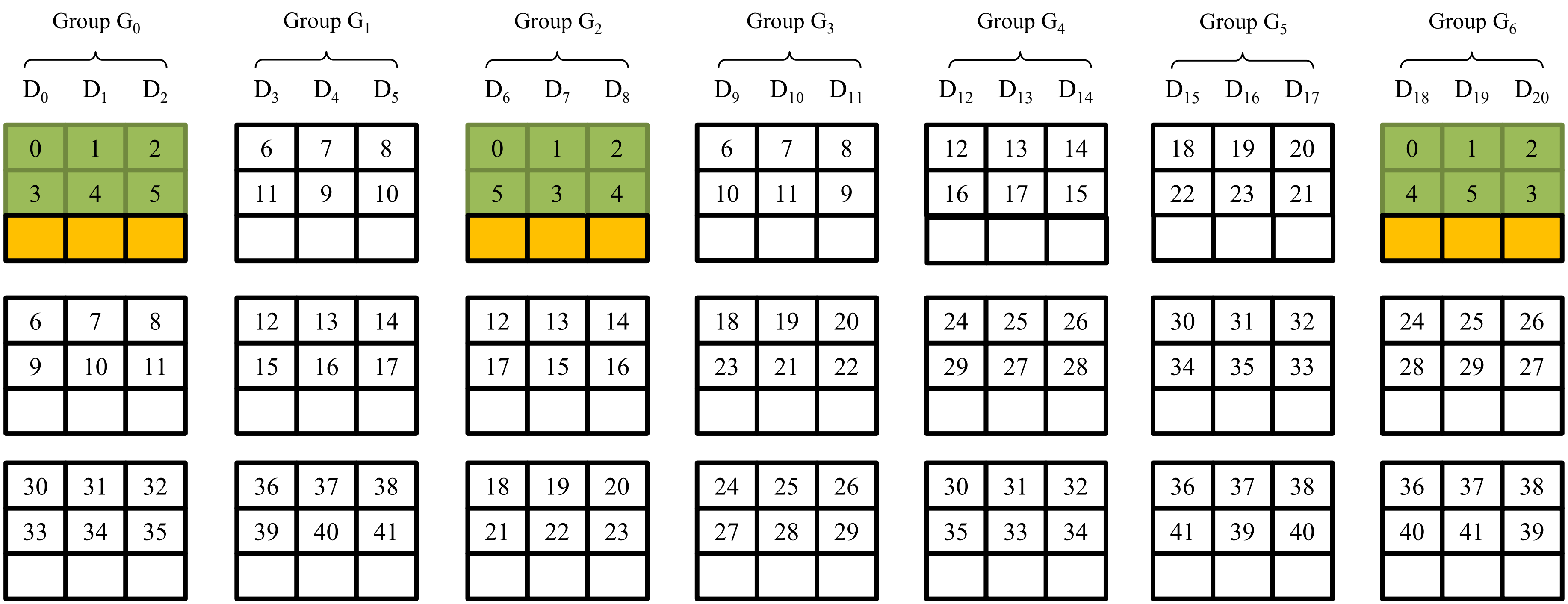
OI-RAID: An Example



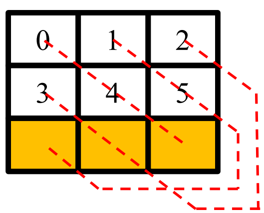
- Group regions into **region sets** based on BIBD
- (7,7,3,3,1)-BIBD:

  Tuple T0：0，2，6　Tuple T1：0，1，3　Tuple T2：1，2，4　Tuple T3：2，3，5
  Tuple T4：3，4，6　Tuple T5：0，4，5　Tuple T6：1，5，6

OI-RAID: An Example



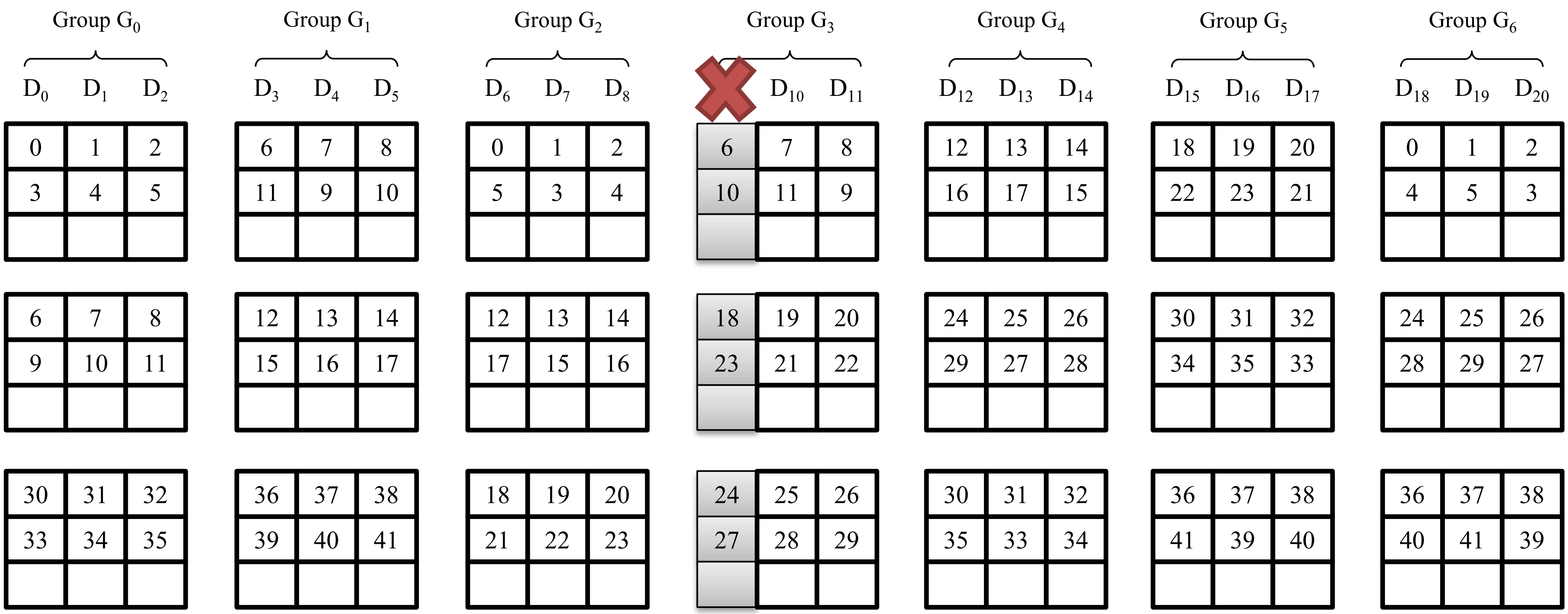
- **Inner layer code:**
  - RAID5 within a region along the diagonal line



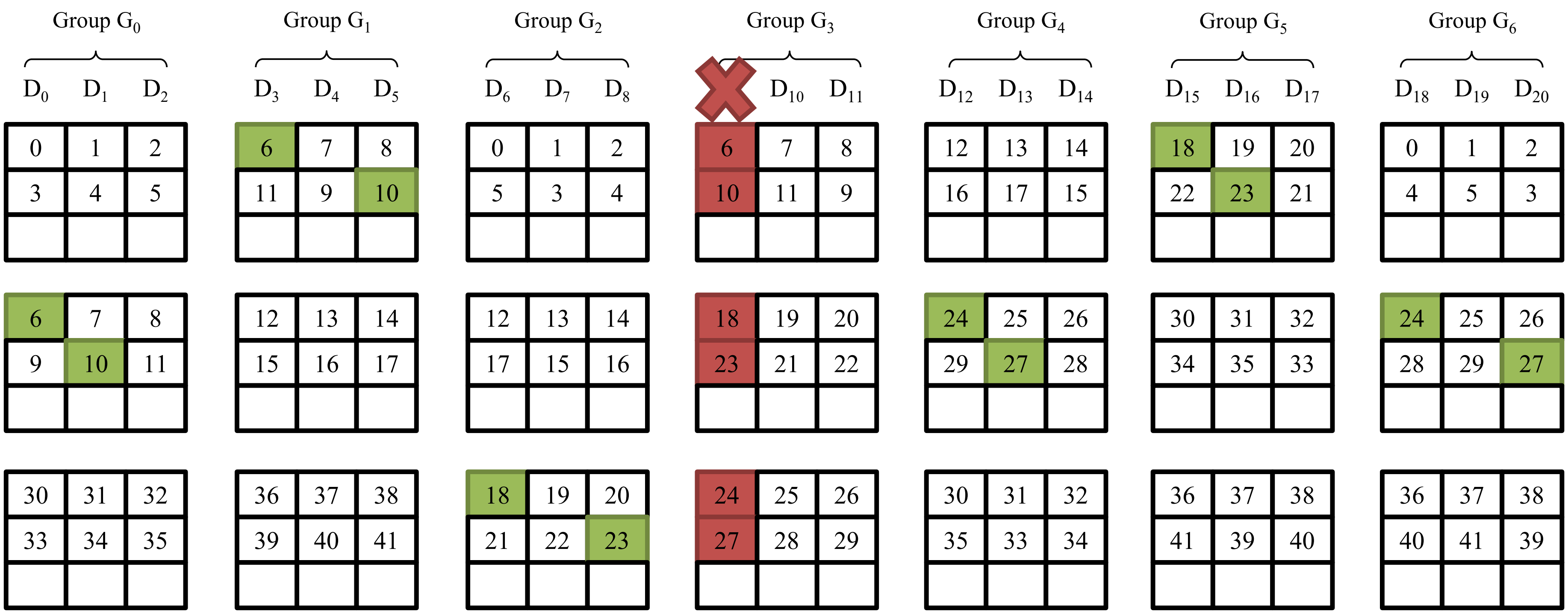
- **Outer layer code**
  - RAID5 within a region set

The two layer code makes OI-RAID tolerate three arbitrary disk failures

Single Failure Recovery



Single Failure Recovery



- To rebuild the 6 failed data units in disk D_9
  - OI-RAID reads only **one** unit from each surviving disk

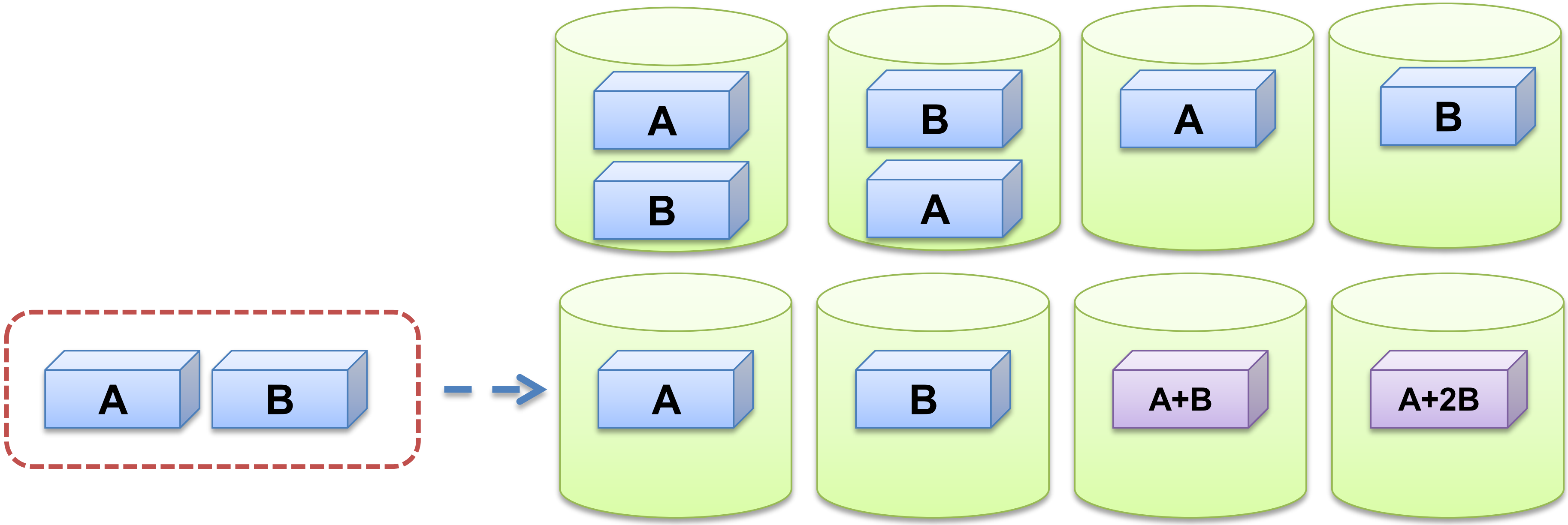
✓ Neng Wang, Yinlong Xu, Yongkun Li, and Si Wu. "OI-RAID: A Two-layer RAID Architecture Towards Fast Recovery and High Reliability." IEEE/IFIP **DSN**, Toulouse, France, June 2016.

✓ Yongkun Li, Neng Wang, Chengjin Tian, Si Wu, Yueming Zhang, Yinlong Xu."A Hierarchical RAID Architecture Towards Fast Recovery and High Reliability." IEEE TPDS, 29(4) , pp. 734 - 747, April 2018.

Topics

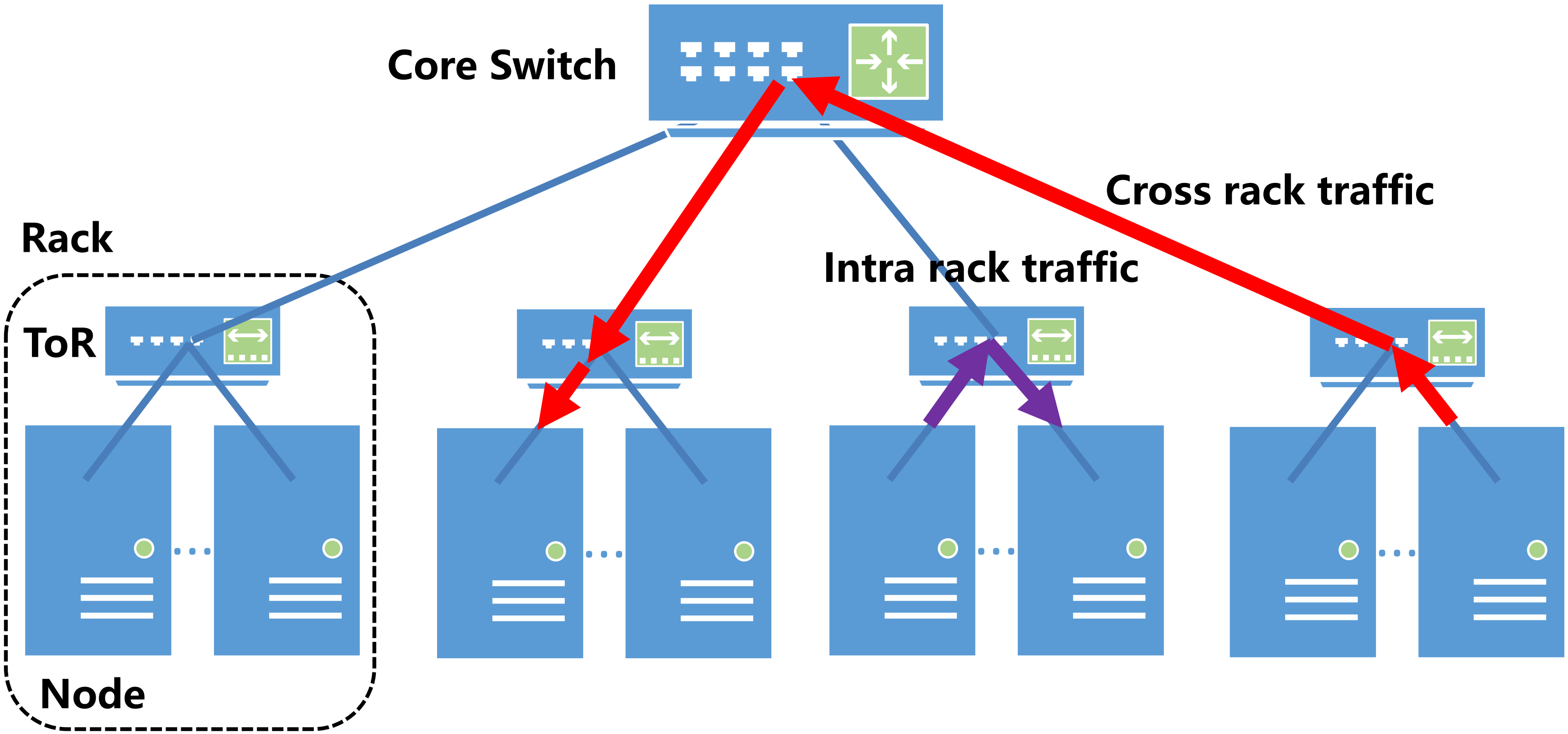
- Problems with RAID/EC
  - Optimizing parity updates
  - Recovery
  - **Asynchronous coding**
  - …

Replication vs. Erasure Coding



- Replication has better **read throughput** while erasure coding has smaller **storage overhead**
- **In practice:**
  - **Data will be frequently accessed in a short time**
  - **Replication to erasure coding**

Clustered File System

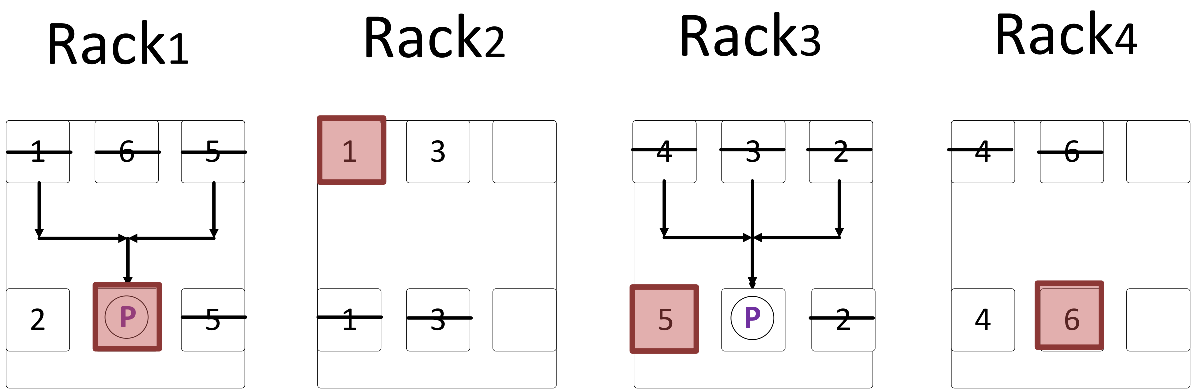


Cross-rack access typically takes ***longer time***

Dynamic Stripe Construction

How to reduce/eliminate cross-rack traffic?

Our solution: Dynamic strip construction



- Encoding speed increases by up to 81%
- Improve frontend map-reduce tasks by 16.4%

Shuzhan Wei, **Yongkun Li***, Yinlong Xu, and Si Wu. "DSC: Dynamic Stripe Construction for Asynchronous Encoding in Clustered File System." IEEE **INFOCOM**, Atlanta, USA, May 2017.

Many Other Problems

- Recovery in heterogeneous systems
- Storage system scaling
- Hybrid system design
  - HDD+SSD
  - NVRAM
- Leveraging SSDs in various systems
- Data consistency
- …

Summary of Ch7

Disk Structure

Disk Scheduling

- ✓ Cylinder, Track, Sector: CLV, CAV
- ✓ Access time
- ✓ FCFS, SSTF, SCAN/C-SCAN, LOOK/C-LOOK

SSD Structure

SSD Features/Issues

- ✓ Structure and features
- ✓ Operations (read/write/erase/GC)

RAID

Erasure Coding

Research Problems

- ✓ RAID structures (RAID0, 1, 4, 5, 6)
- ✓ Parity update

End of Chapter 7