Mass Storage

Last Update: May 30, 2023

Objectives and Outline

Objectives

- Describe the physical structure of secondary storage devices and the resulting effects on the uses of the devices
- Explain the performance characteristics of mass-storage devices
- Discuss operating-system services provided for mass storage, including RAID

Outline

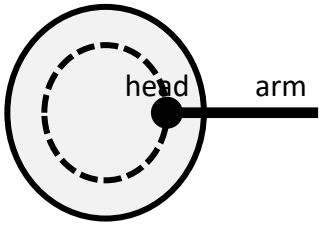
- Overview
- Disk Structure
- Disk Attachment
- Disk Scheduling
- Disk Management
- Swap-Space Management
- RAID Structure

Mass Storage

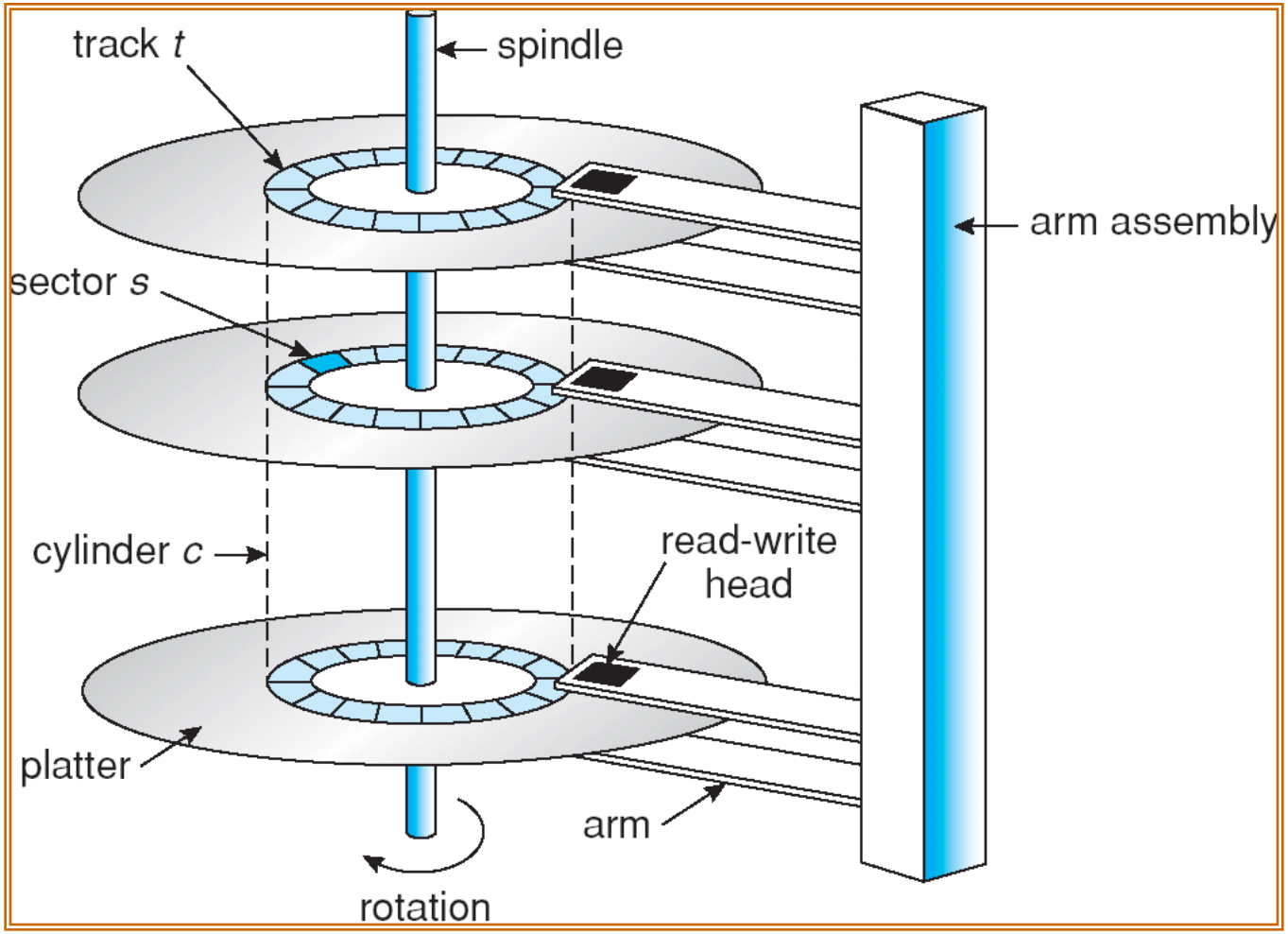
- *Mass Storage*: persistent storage; large volume of data can be stored permanently (powering off will not cause loss of data)
  - Secondary storage: always online;
    - hard disk drives (HDDs);
    - flash-based solid state disks (SSDs).
  - Tertiary storage: not always online:
    - CDs,
    - tapes, etc.

Overview of Mass Storage Systems: Magnetic Disks

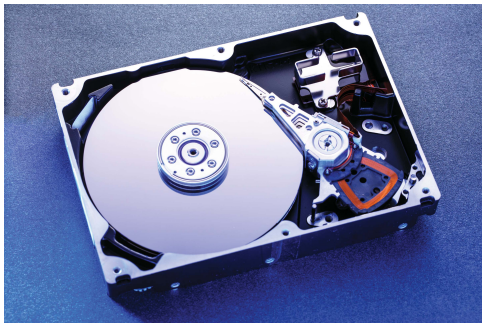
- Magnetic disks (HDDs) provide bulk of secondary storage of modern computers
- Drives rotate at 3600 to 15000 times per minute (RPM)
- Transfer rate is data-rate at which data flow between drive and computer (memory)
  - *Number of bytes transferred per second*
- Positioning time (random-access time):
  - seek time: time to move disk head (arm) to the desired cylinder (track) (ms)
  - rotational latency: time for desired sector (block) to rotate under the disk head (ms)



Moving-head Disk Mechanism



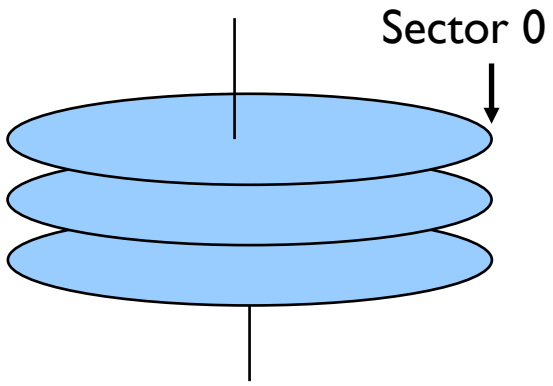
HDD



- Platters range from .85” to 14” (historically)
  - Commonly 3.5”, 2.5”, and 1.8
- GBs or TBs capacity.
- peek transfer rate: 100s MB/s, or GB/s
- seek time: 3 ms – 12 ms.
- rotational latency: depends on rpm (rotations per minute).
  - Example:
    - Assume RPM = 7200 (rotations per minute - rpm)
    - Then: 7200/60 = 120 rotations per second.
    - Then, one rotation time = 1/120 = 8.33 ms
    - Average rotational latency = 8.33/2 = 4.16 ms

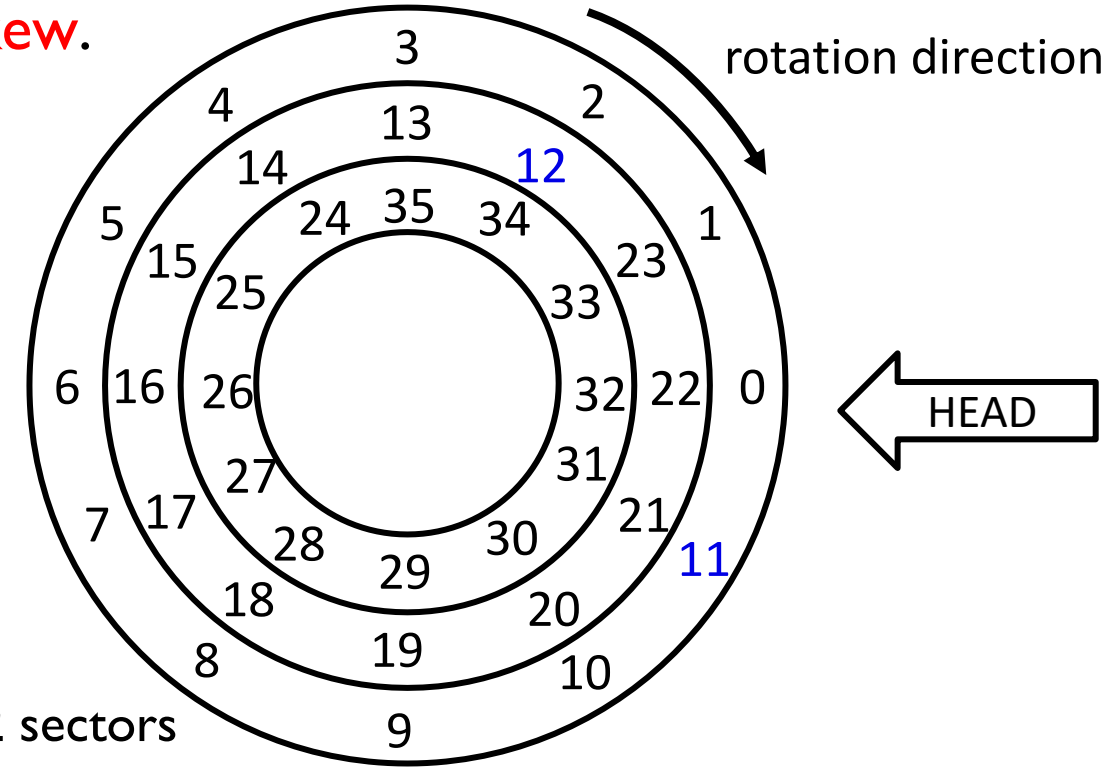
Disk Structure

- A disk drive is addressed as a large 1-dimensional array of blocks, where the logical block is the smallest unit of transfer.
- Each block has a number (logical block address – LBA).
- The 1-dimensional array of logical blocks is mapped onto the sectors of the disk sequentially.
  - Sector 0 is the first sector of the first track on the outermost cylinder.
  - Mapping proceeds in order through that track, then the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost.



Track skewing

- To improve sequential access performance, the sector number on the next track starts so that the time to move the head from current track to the next one is accommodated.
- This is called track skew.
- For example: time to move to next track can be:
  ~ 0.3ms - 1.5 ms.



Multi-zone disks

- Current disks have thousands of tracks in a surface.
  - Inner tracks shorter (can have less sectors),
  - Outer tracks longer (can have more sectors).
- Constant angular velocity (CAV) disk spins at the same speed.
- Tracks can be divided into zones. A zone has consecutive tracks.
  - In a zone, the number of sectors per track is the same.
  - Outer zones have more sectors.
  - Therefore surface transfer-rate of outer zones is higher.
- For example:
  - Innermost zone surface transfer-rate can be 54 MB/s.
  - Outermost zone surface transfer-rate can be 128 MB/s.

Track buffer

- Disk drive has a cache. It is called track buffer. Size: 8-16 MB.
- It can cache, for example, an entire track.
- Even a single sector is read from a track, the disk can decide to read the whole track and cache it (it is under the head anyway).
- If next sector to read is in the same track, we have a hit.
- Writes may also be buffered in the track first.
  - Write back: Ack immediately when data is on track. Write later to surface. (faster(
  - Write through: Ack the request when write is done on surface. (slower)

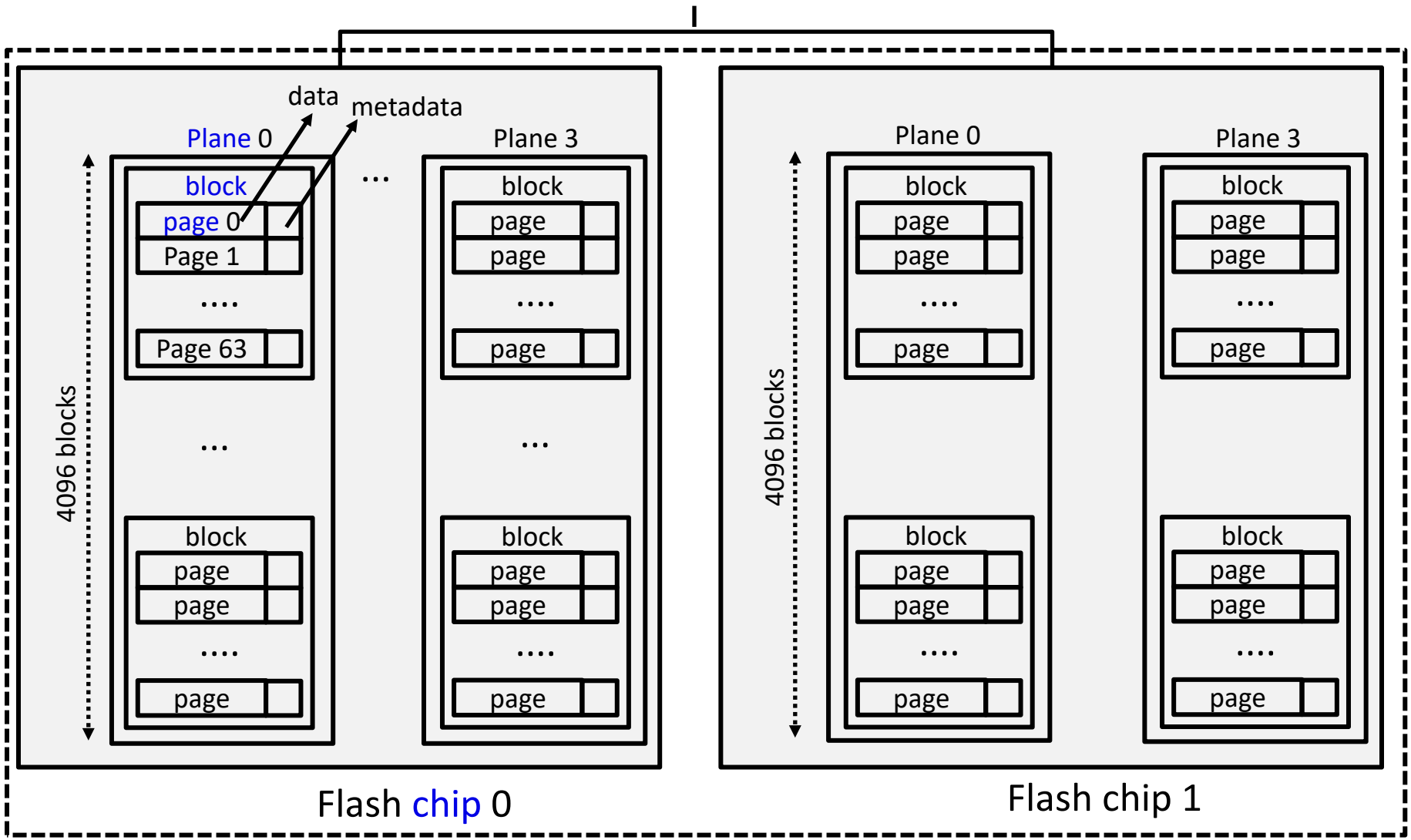
Address Mapping

- In HDDs, the mapping of logical block numbers (LBAs) to physical sector numbers is simple (just sequentially mapped – direct mapping).
  - Block 0: sector 0; block 1: sector 1; ...., or
  - Block 0: sector 0,1,2,3; Block 1: sectors 4,5,67;....
- The exact location of sectors on the disk (cylinder number, track number, sector in track) is known by the disk controller (sometimes, by the disk driver).
  - Modern hard disks are complex. CLV or CAV disks.
  - CLV: constant linear velocity
  - CAV: constant angular velocity

Address Mapping in SSDs

- SSDs are usually implemented with NAND Flash chip technology.
- OS (file system) considers an SSD-disk also as a 1-dimensional array of blocks, numbered as 0,1,2....
- Physically,
  - SSD has physical pages to store those file system blocks.
    - All SSD pages are uniquely numbered.
  - SSD keeps a number of physical pages in a physical SSD block.
    - A physical SSD block can contain, for example, 128 SSD pages.

Example Flash Package (an SSD)



SSD Properties

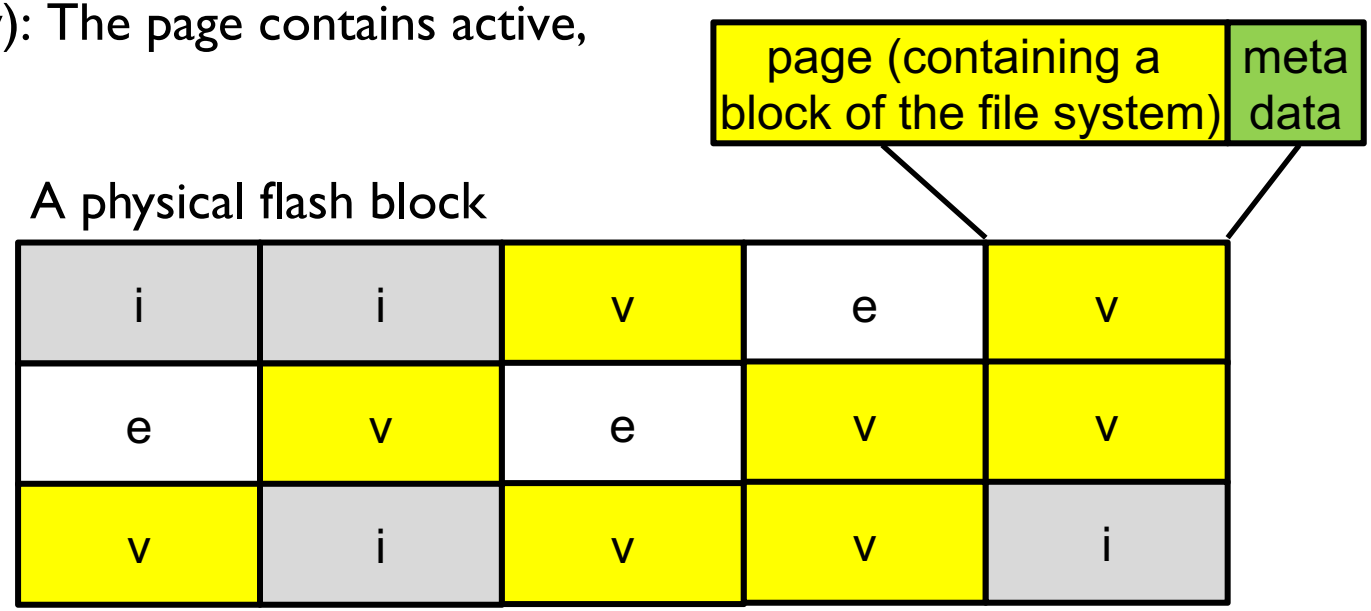
- SSD has different properties than HDD.
  - An SSD page can not be over-written. The block containing the page has to be erased first.
  - This is the constraint of the flash technology.
  - Erase is a costly operation (takes time).
- If a file system block k (data) in an SSD page x is to be modified, it is first read into memory, modified, and then written to another SSD page y. Then the data in the old page x becomes invalid.
- Later, after the SSD block containing page x is erased, a file system block can written on page x again.

Basic Flash Operations

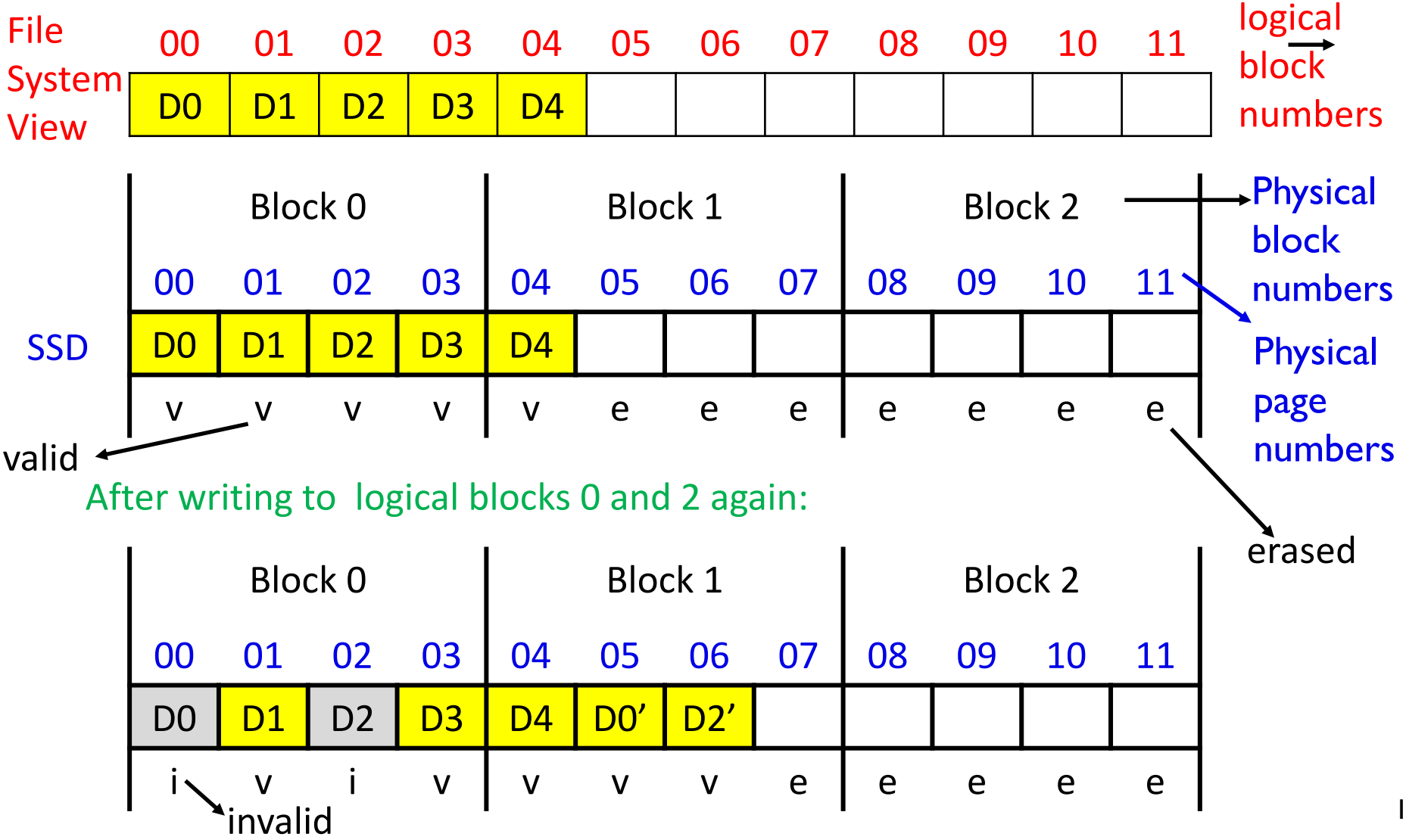
- The following are the low level operations that a flash chip supports.
  - Read (a page). Data is read from an SSD page whose number is specified.
    - Takes 10s of microseconds.
  - Write (a page). It is also called programming a page. Data is written to an SSD page whose number is specified.
    - Takes 100s of microseconds.
  - Erase (a block). Before writing to a page in a physical block, the entire block must be erased first.
    - All data destroyed. All bits become 1.
    - Takes a few ms (for example, 3 ms). Much slower than a read or write.
    - Before erasing a block, SSD needs to read and write the live pages in the block, if any, to some other block(s) first.

Page state

- A page can be in one of 3 states:
  - INVALID (i): the data in the page is old (dead logical block) and no longer needed (it is overwritten into another page).
  - ERASED (e). Page is erased (reset) (set to all `1`s), since the physical block containing the page is erased (reset).
    - The page is programmable (writable) again.
  - VALID (v): The page contains active, live data.



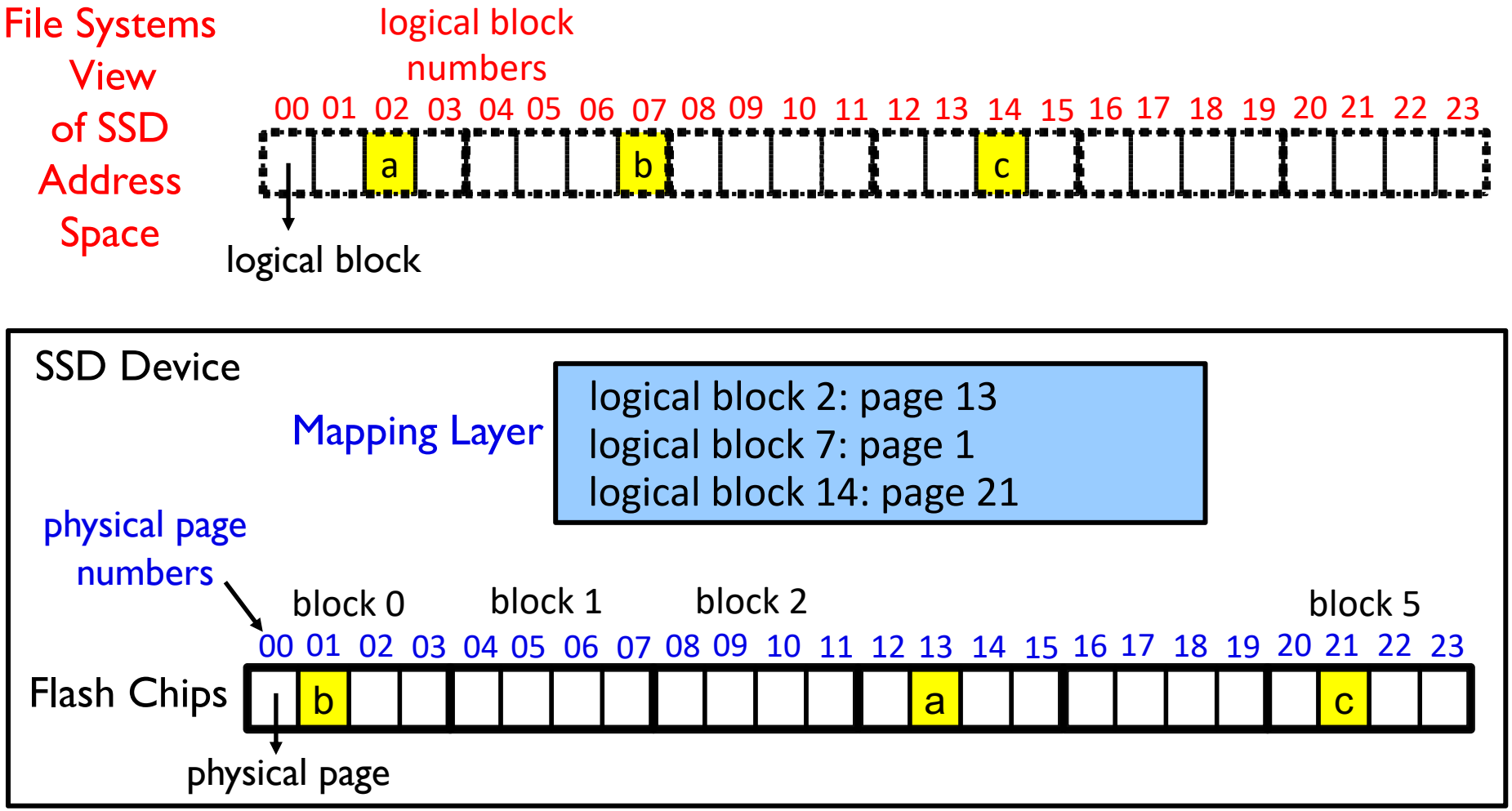
Page state


File System View
00 01 02 03 04 05 06 07 08 09 10 11
logical block numbers
D0 D1 D2 D3 D4
Block 0
Block 1
Block 2
Physical block numbers
SSD
00 01 02 03 04 05 06 07 08 09 10 11
D0 D1 D2 D3 D4
v v v v v e e e e e e e
Physical page numbers
valid
erased
After writing to logical blocks 0 and 2 again:
Block 0
Block 1
Block 2
00 01 02 03 04 05 06 07 08 09 10 11
D0 D1 D2 D3 D4 D0' D2'
i v i v v v v e e e e e
invalid

Address Mapping in SSDs

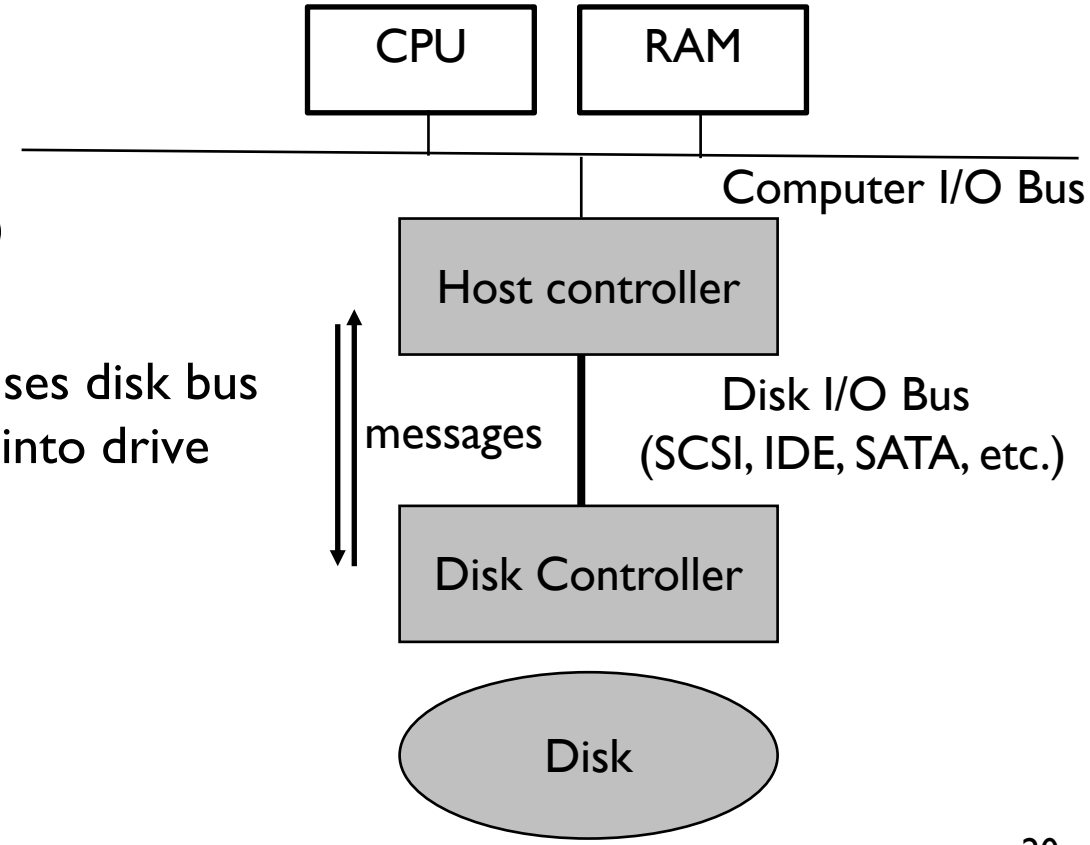
- Therefore, in an SDD disk, the mapping of logical blocks (of the file system) to the physical pages of the SSD (storage device) is not static, is not direct. It is dynamic.
- Internally, SDD keeps a dynamic table for the mapping.

FTL: flexible and dynamic mapping



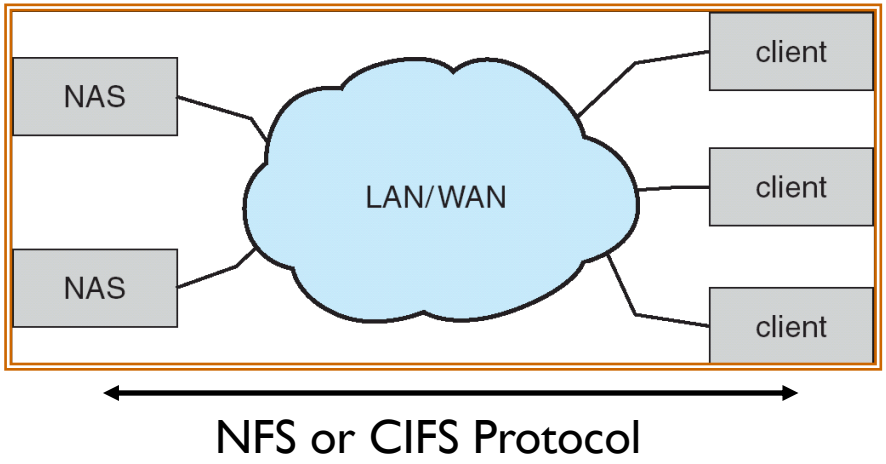
Disk Attachment

- Host-attached storage accessed through I/O ports talking to disk I/O busses
- Attachment technologies and protocols (various disk I/O buses)
  - IDE, EIDE, ATA, SATA
  - USB
  - SCSI
  - Serial Attached SCSI (SAS)
  - Fiber Channel
- Host controller in computer uses disk bus to talk to disk controller built into drive or storage array



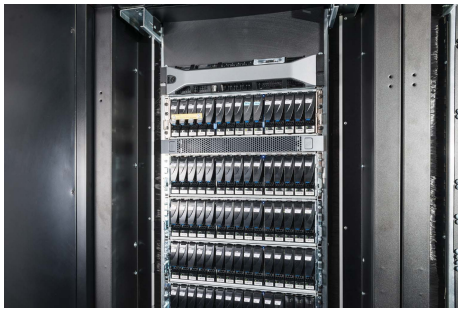
Network Attached Storage

- Network-attached storage (NAS) is storage made available over a network rather than over a local connection (such as a bus)
- NFS and CIFS are common distributed file system protocols used for network attached storage
  - We use those protocols to access remote storage that is connected to a network.
- Implemented via remote procedure calls (RPCs) between host and storage

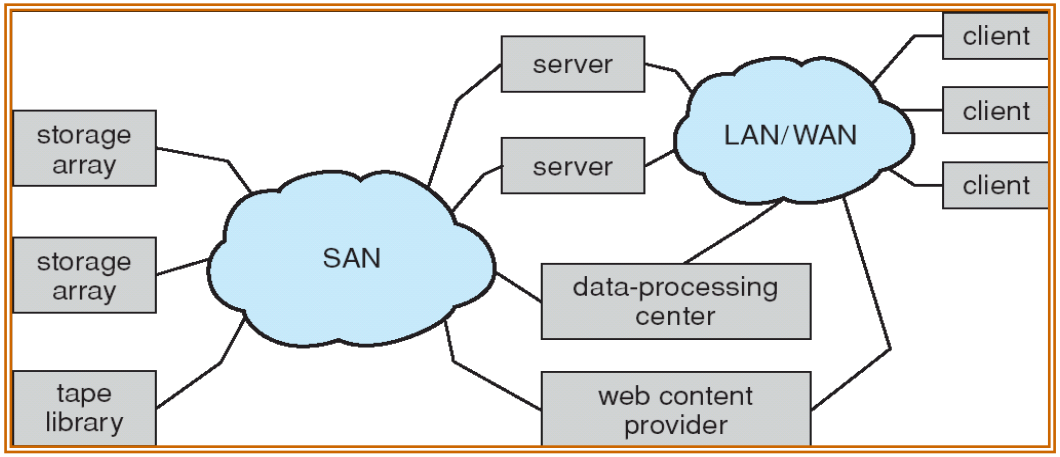


Storage Area Network

- Common in large storage environments (and becoming more common)
- Multiple hosts attached to multiple storage arrays – flexible
- Uses a different communication infrastructure (SAN) than the common networking infrastructure



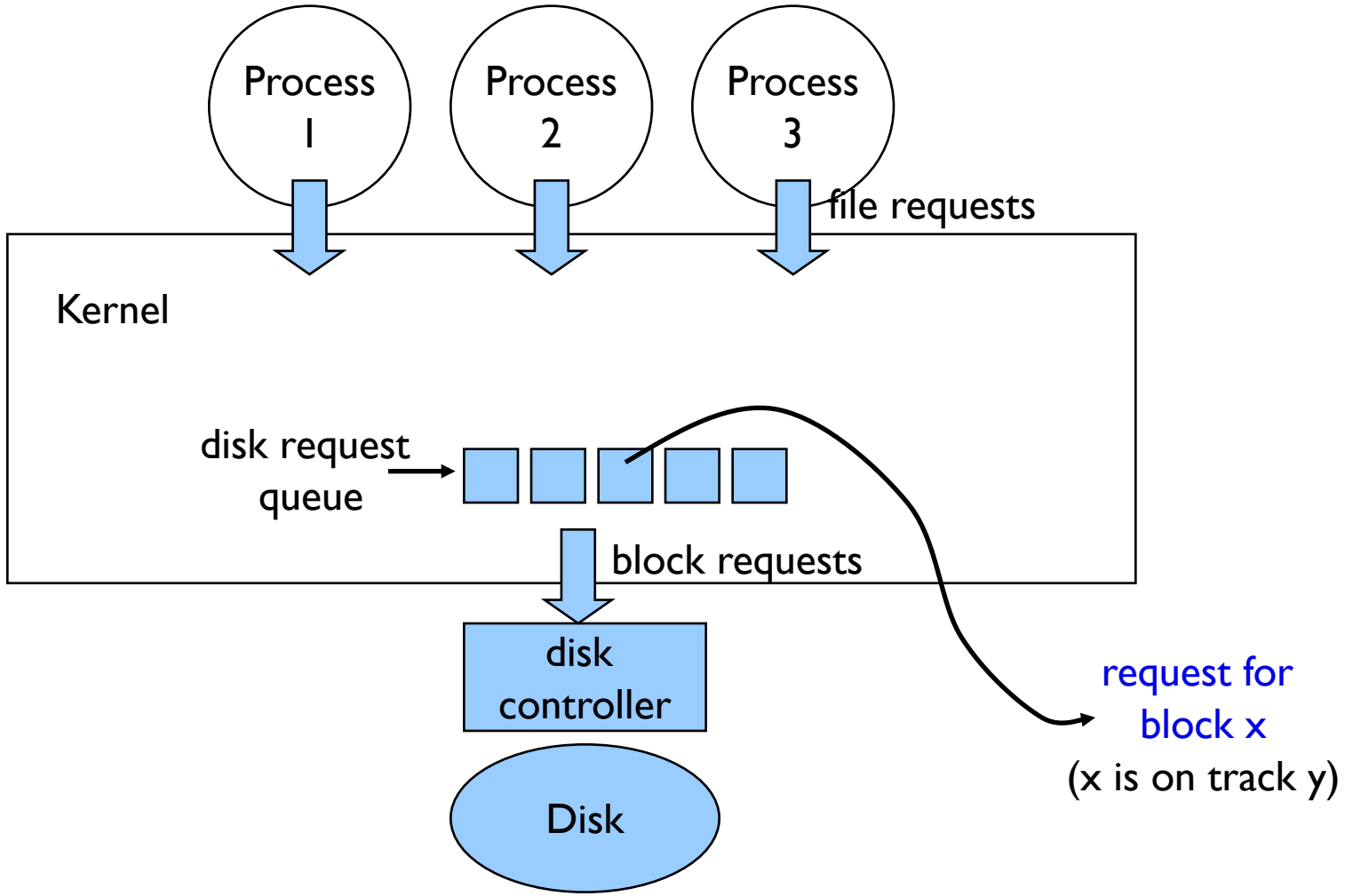
a storage array



Disk Scheduling

- The operating system is responsible for using hardware efficiently — for the disk drives, this means having a fast access time and large disk bandwidth.
- Disk access time has two major components
  - Seek time is the time for the disk to move the head to the track containing the desired sector (block).
  - Rotational latency is the additional waiting time for the disk to rotate the desired sector under the disk head.
- Minimize seek time
- Seek time ≈ seek distance (between tracks)
- Disk bandwidth is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer.

Disk I/O queue



Disk Scheduling

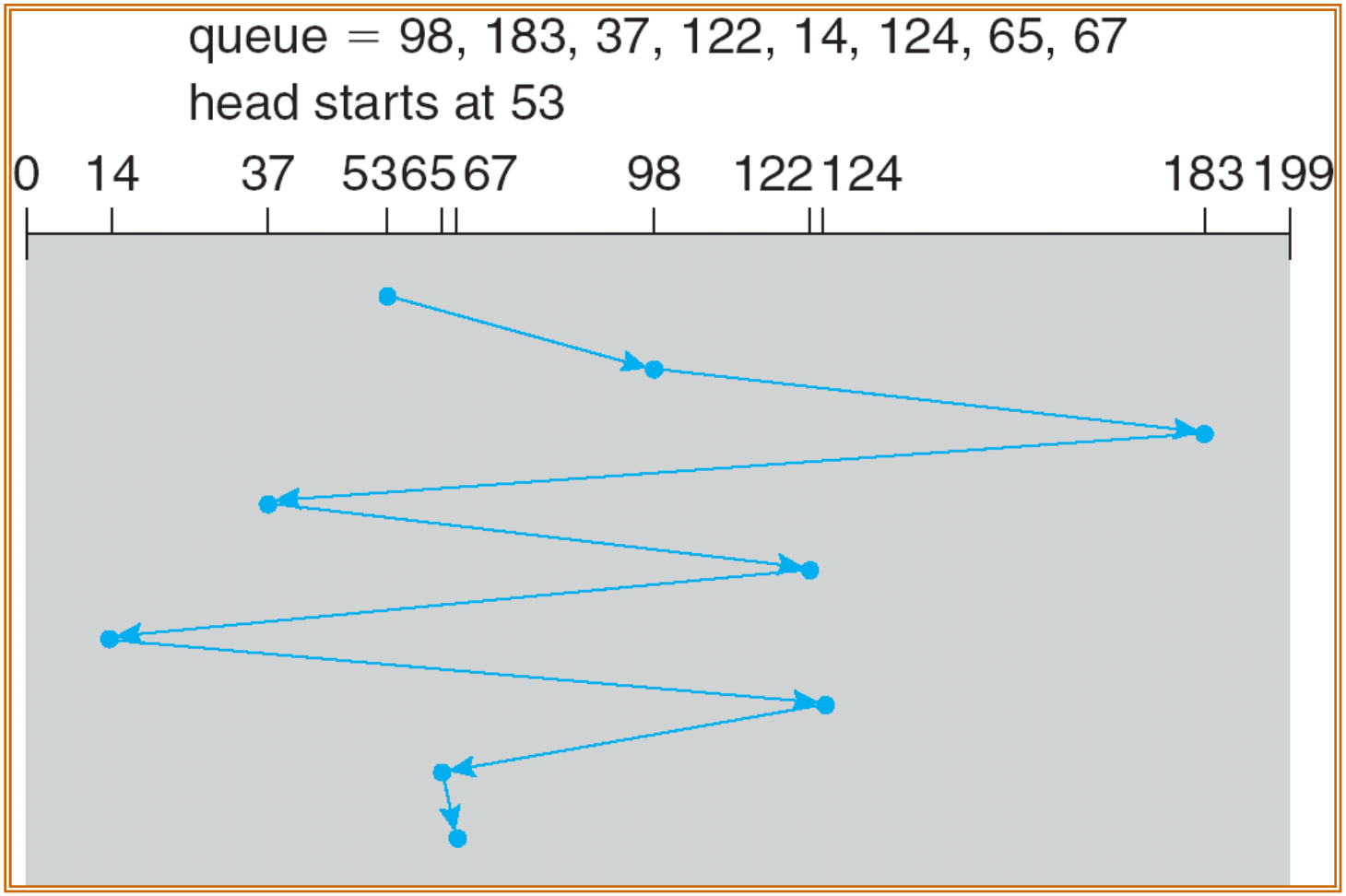
- Several algorithms exist to schedule the servicing of disk I/O requests.
- Assume disk has tracks/cylinders from 0 to 199.
- We illustrate them with a *request queue*. In the queue we have requests for blocks sitting on various tracks.
- In a modern disk, we may have thousands of tracks in a platter.
- We just focus on the track/cylinder numbers.

98, 183, 37, 122, 14, 124, 65, 67 (these are track numbers)

Head pointer: 53 (the head is currently on track 53)

We have 8 requests queued. They are for blocks sitting on tracks 98, 183, …

First Come First Served (FCFS) Algorithm

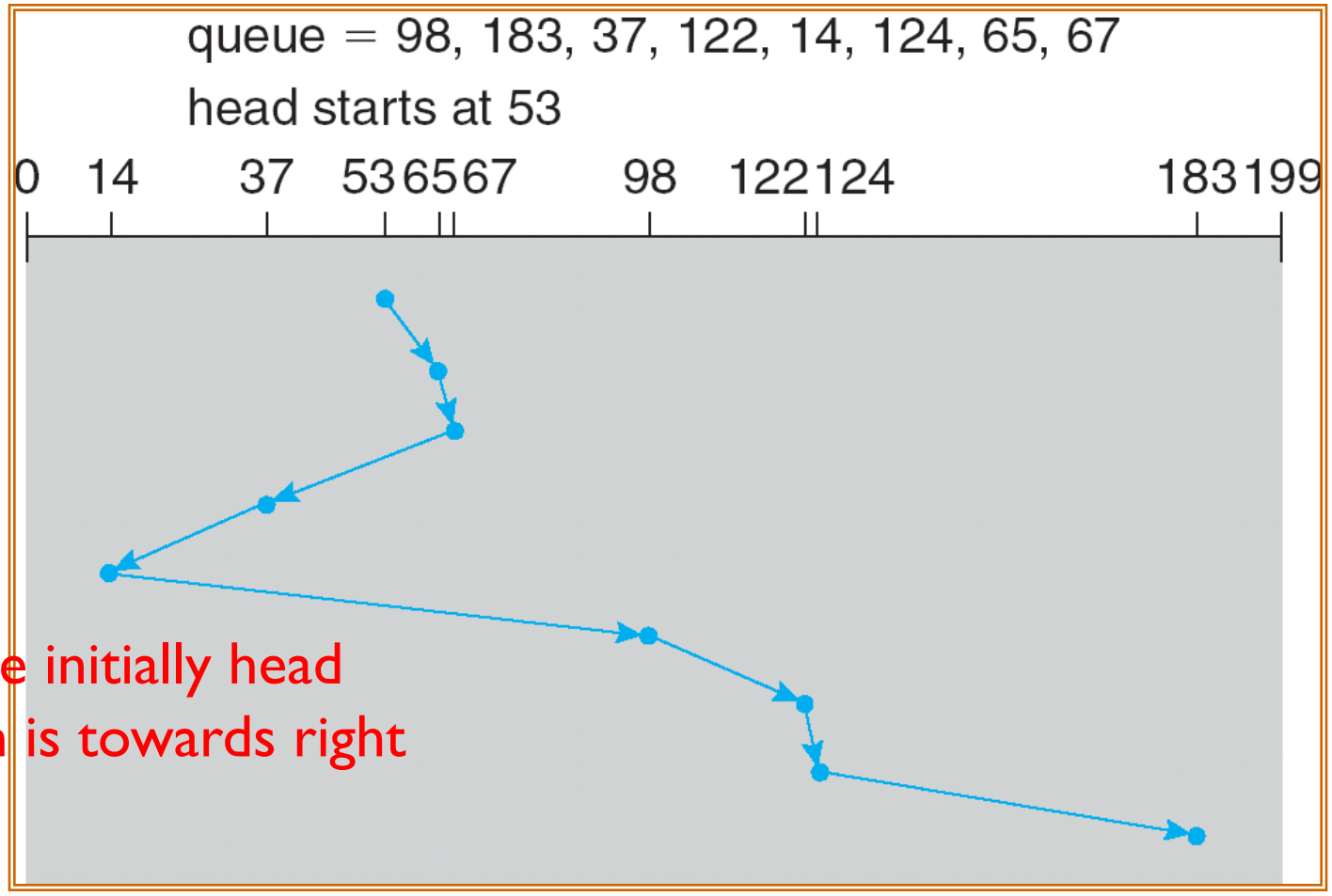


total head movement = 640 tracks

Shortest Seek Time First (SSTF) Algorithm

- Selects the request with the *minimum seek time* from the current head position.
- SSTF scheduling is a form of SJF scheduling; may cause starvation of some requests.

SSTF



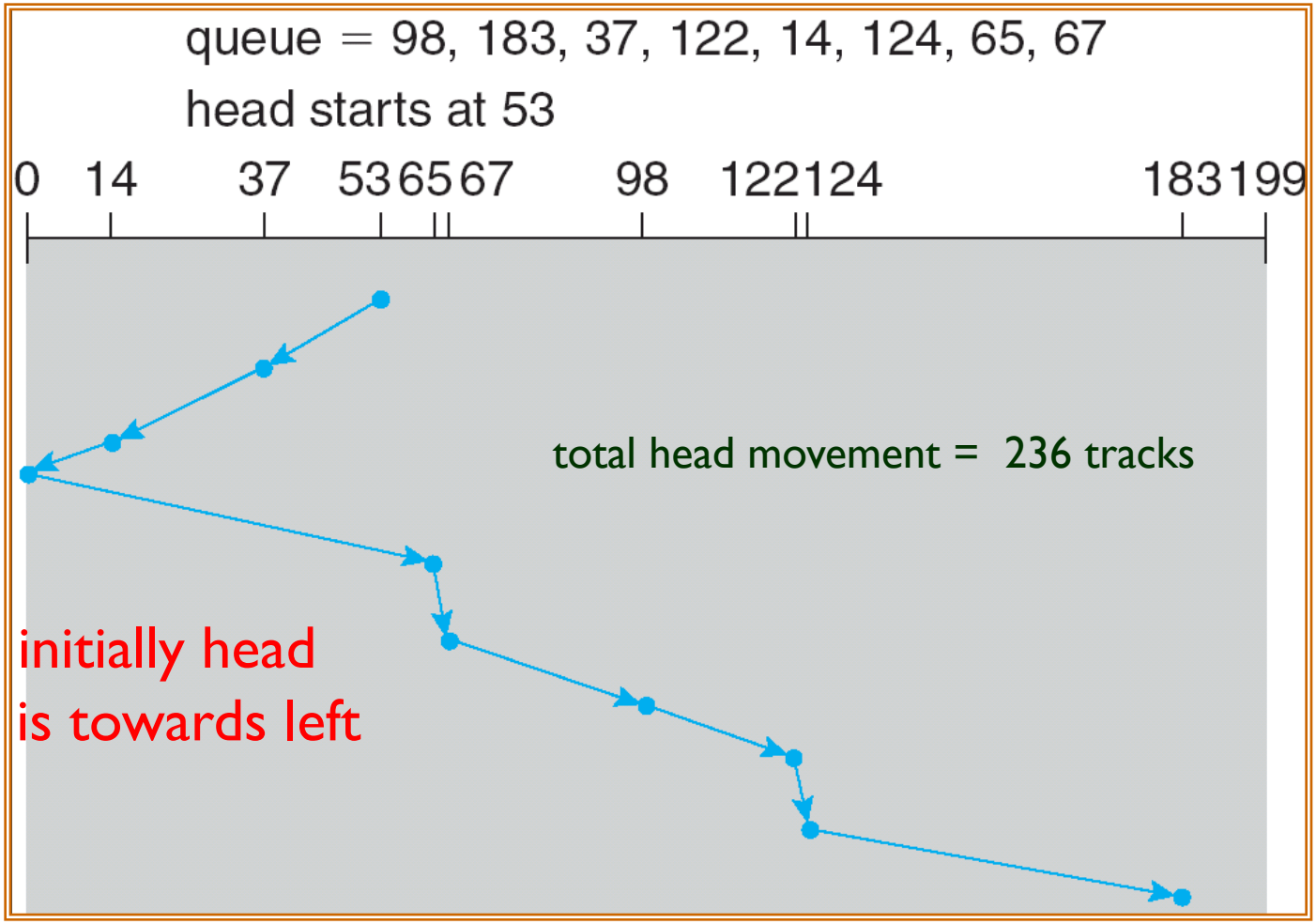
Assume initially head direction is towards right

total head movement = 236 tracks

SCAN/ELEVATOR Algorithm

- The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues.
- Also called the *Elevator algorithm*.
- Several variations of the algorithm exist:
  - SCAN
  - C-SCAN

SCAN

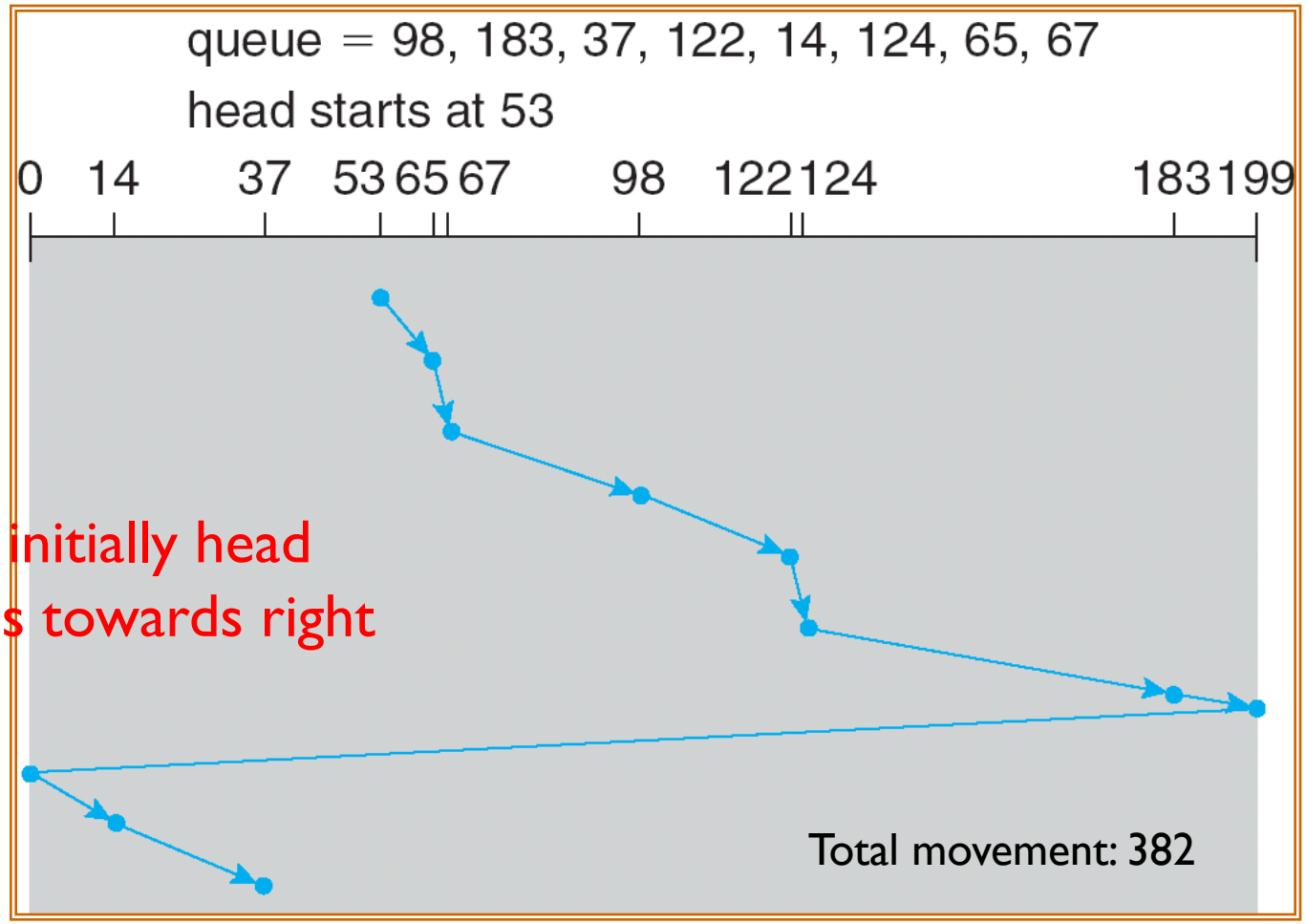


Assume initially head direction is towards left

C-SCAN

- C-SCAN: Circular SCAN
- Provides a *more uniform wait time* than SCAN.
  - Wait time for a request: time between arrival of a request to the queue and the completion of handling the request.
- The head moves from one end of the disk to the other; servicing requests as it goes. When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip.
- Treats the cylinders as a circular list that wraps around from the last track to the first one.

C-SCAN



Assume initially head direction is towards right

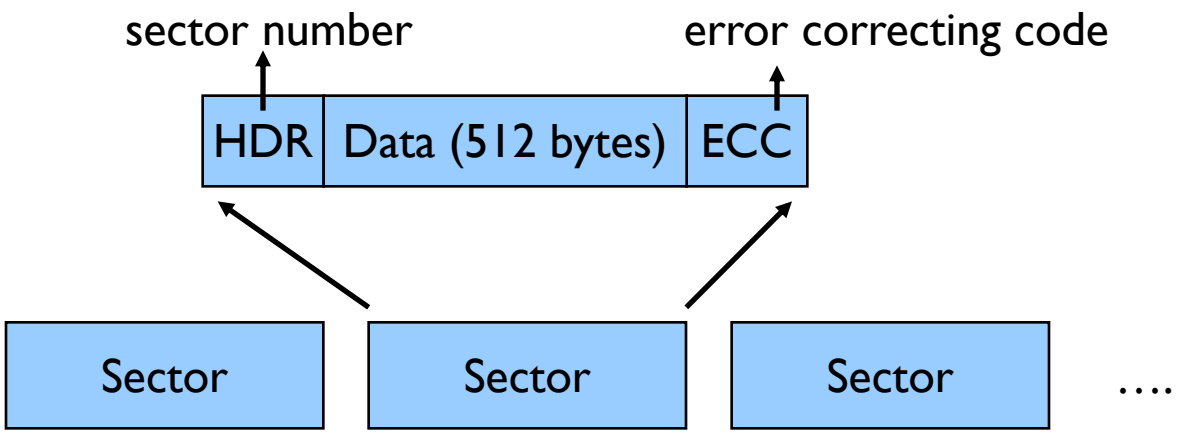
Selecting a Disk-Scheduling Algorithm

- SSTF is common and has a natural appeal.
- SCAN (or a version of it) performs better for systems that place a heavy load on the disk.
- Performance depends on the number and types of requests.
- Requests for disk service can be influenced by the file-allocation method.
- The disk-scheduling algorithm should be written as a separate module of the operating system, allowing it to be replaced with a different algorithm if necessary.
- Either SSTF or SCAN is a reasonable choice as the default algorithm.

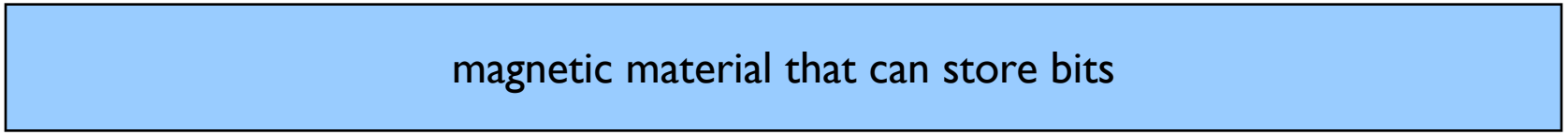
Disk Management

- *Low-level formatting*, or *physical formatting* — Dividing a disk into sectors that the disk controller can read and write.
- To use a disk to hold files, the operating system still needs to record its own data structures on the disk.
- *Partition* the disk into one or more groups of cylinders (volumes).
- *Logical formatting* or "making a file system" on a disk, partition, or volume.

Low Level Formatting



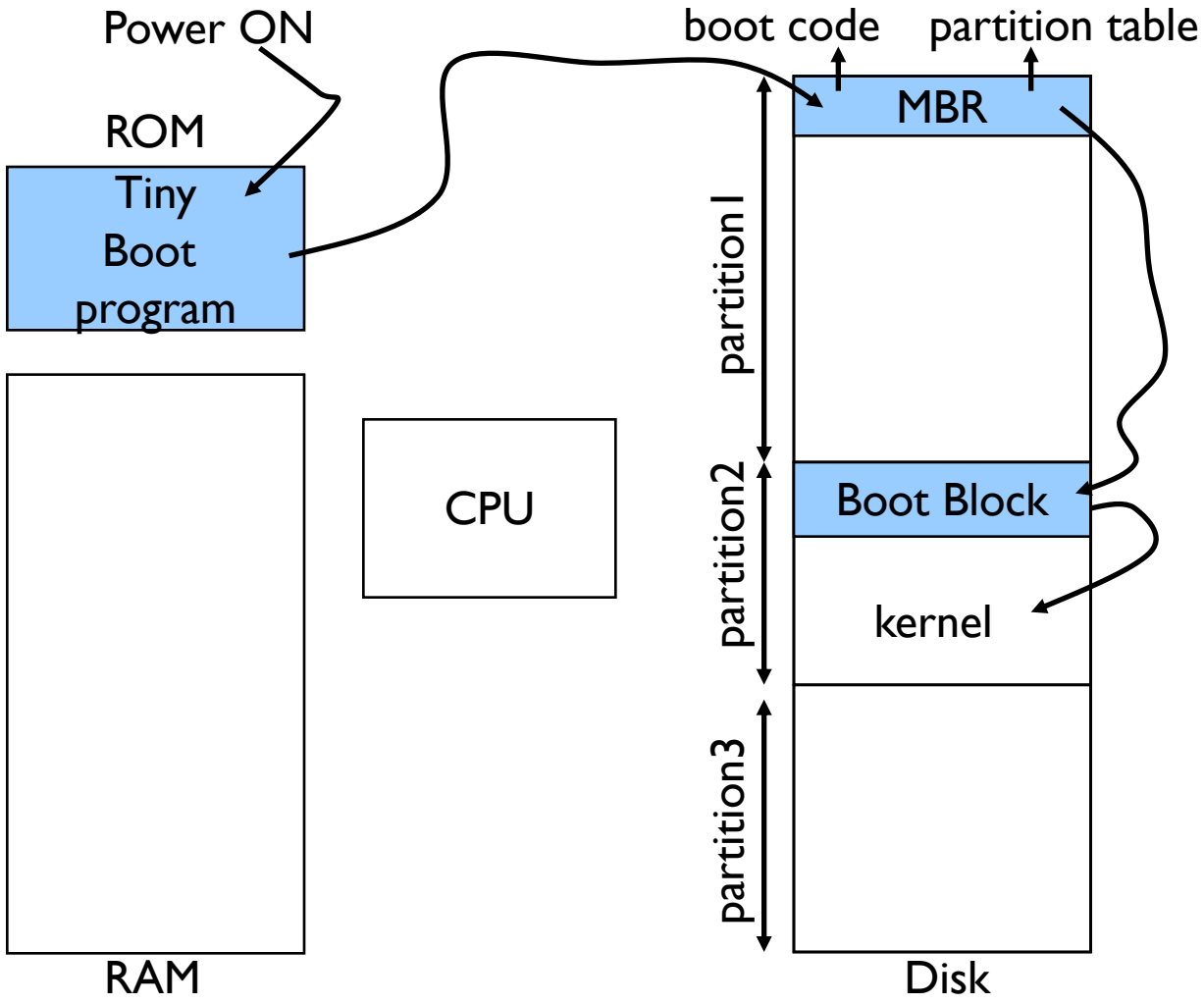
Disk after low level formatting



Disk before low level formatting

Boot Process

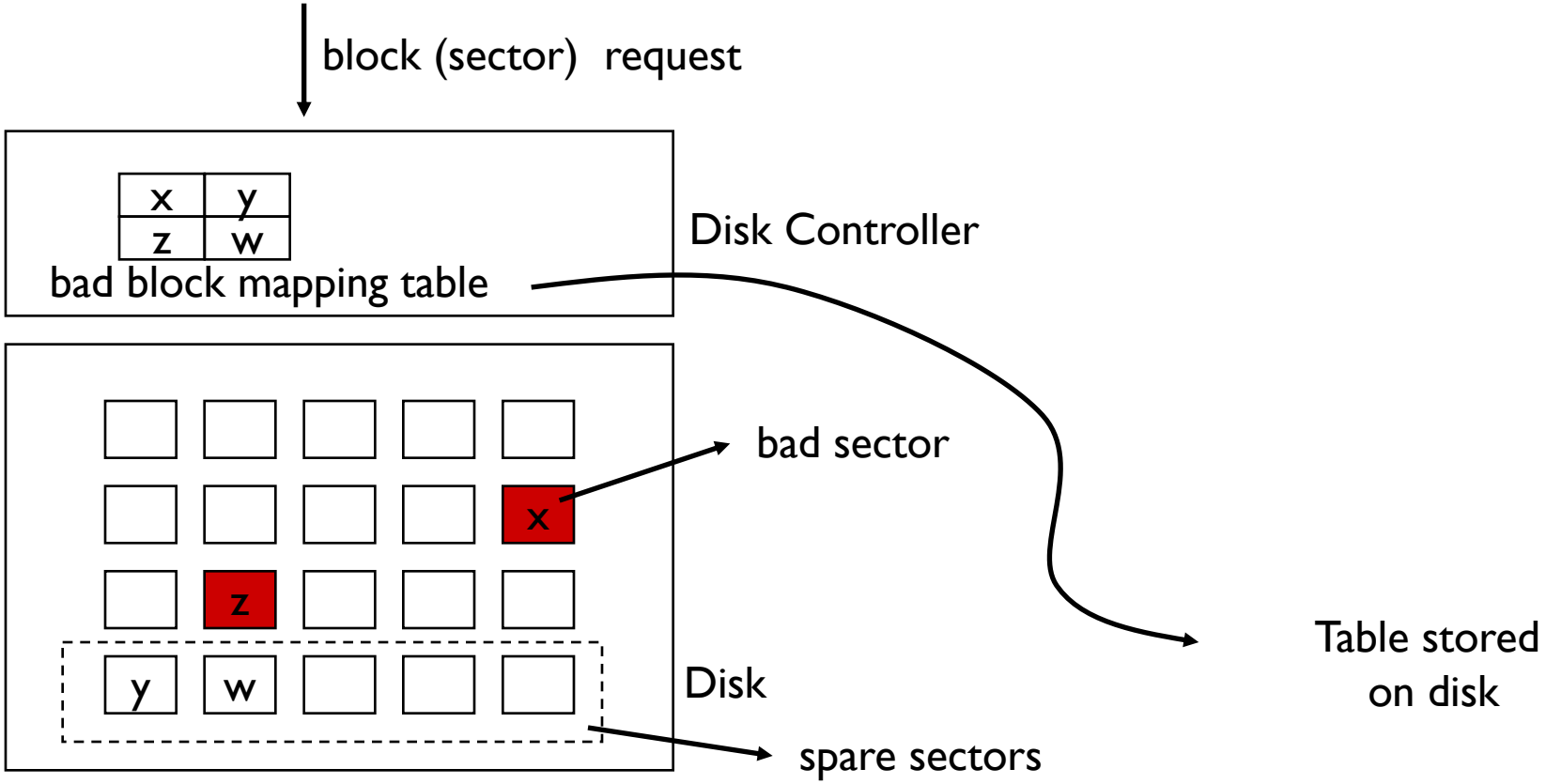
1. Boot code in ROM is run; it brings MBR into memory and starts MBR boot code
2. MBR boot code runs; looks to partition table; learns about the boot partition; brings and starts the boot code in the boot partition
3. Boot code in boot partition loads the kernel sitting in that partition



Bad Blocks

- Disk sectors (blocks) may become defective. Can no longer store data.
  - Hardware defect
  - System should not put data there.
- Possible Strategy:
  - A bad block X can be remapped to a good block Y
    - Whenever OS tries to access X, disk controller accesses Y.
  - Some sectors (blocks) of disk can be reserved for this mapping.
    - This is called sector sparing.

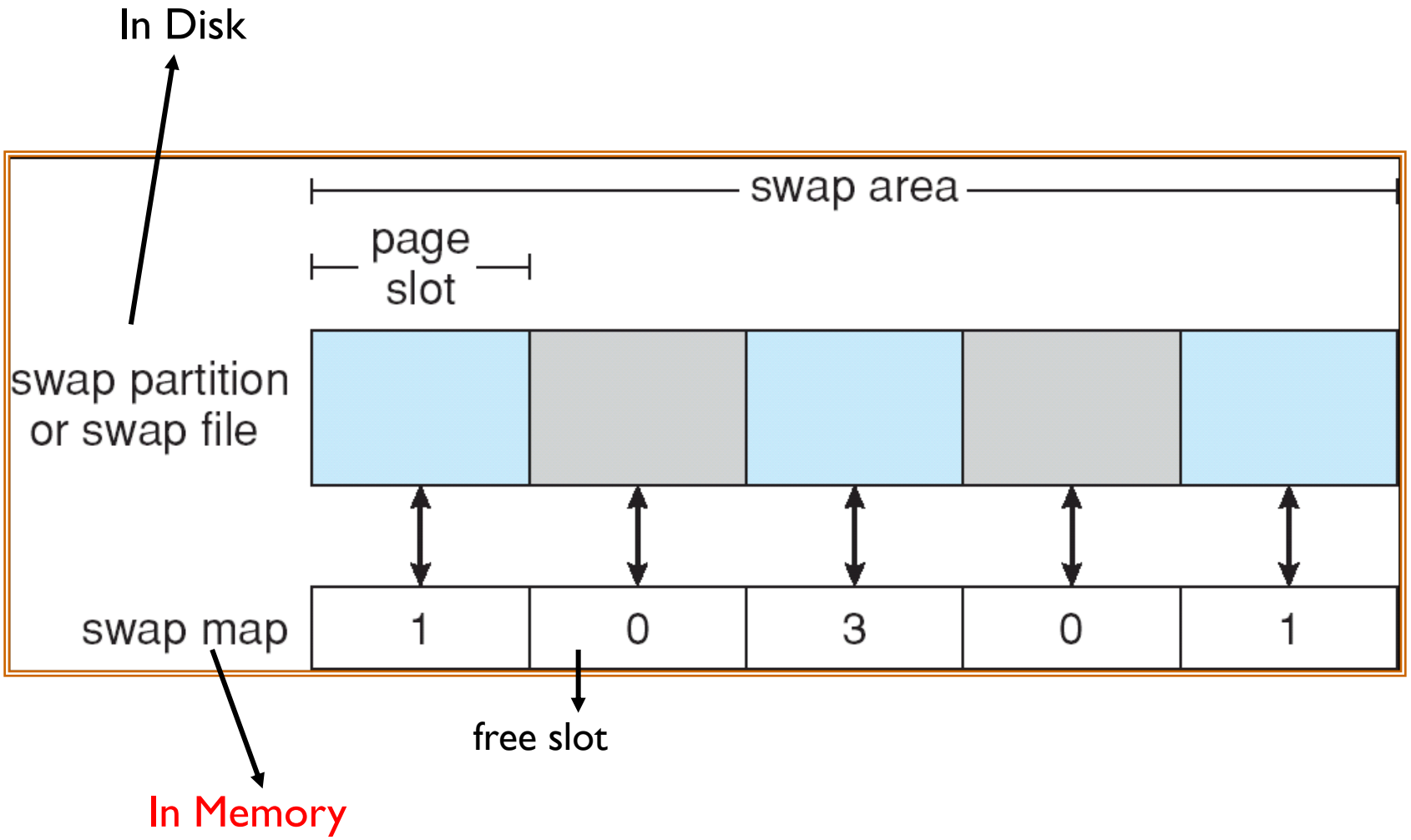
Bad Blocks


block (sector) request
x
y
z
w
bad block mapping table
Disk Controller
bad sector
x
z
y
w
Disk
spare sectors
Table stored
on disk

Swap Space Management

- Swap-space: Virtual memory uses disk space as an extension of main memory. Anonymous pages (not backed by a file) swapped out to swap space. (pages containing heap segment, for example).
- Swap-space can be carved out of the normal file system, or, more commonly, it can be in a separate disk partition.
- Swap-space management
  - Kernel uses *swap maps* to track swap-space use.
  - **Example**: BSD OS allocates swap space when process starts; holds *text segment* (the program) and *data segment*.
  - **Example**: Solaris OS allocates swap space only when a page is forced out of physical memory, not when the virtual memory page is first created.

Data Structures for Swapping on Linux



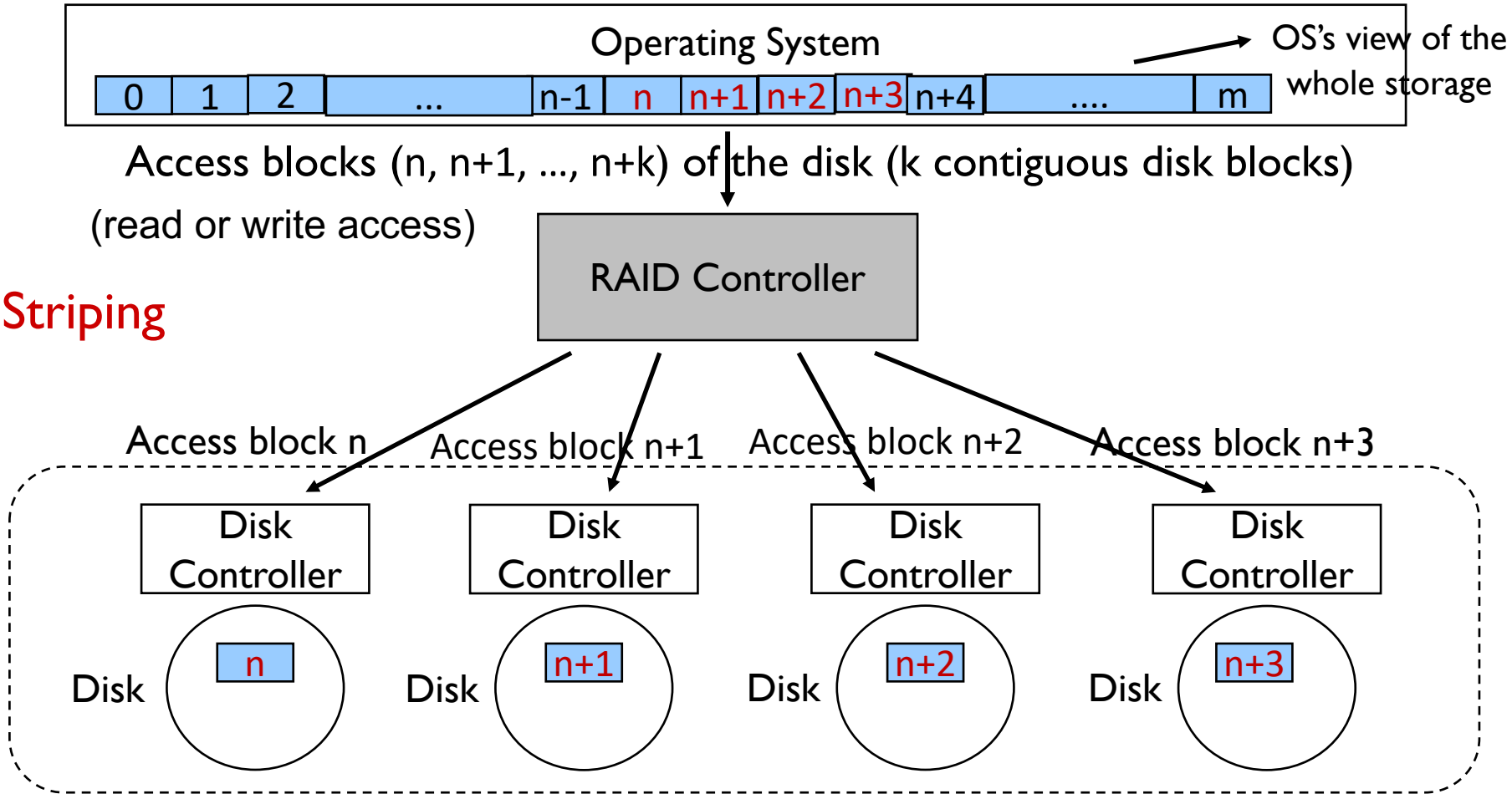
RAID Structure

- RAID: Redundant Array of Independent Disks
- Use of multiple disk drives to store data.
- Multiple disks can be organized in different ways for Reliability and Performance
  - RAID is arranged into different levels/schemes.
- If we have many disks:
  - The probability of one of them failing becomes higher.
  - The probability of all of them failing (at the same time) becomes lower.

RAID

- RAID schemes improve performance and reliability of the storage.
- Reliability: via redundancy
- Performance: via disk striping (also via redundancy)
- Disk striping
  - Disk striping uses a *group of disks* as *one storage unit* and distributes (stripes) the data (blocks) over those disks
  - Reads/writes of blocks can be done in parallel.
- Redundancy by:
  - *Mirroring* or *shadowing* keeps duplicate of each disk.
  - *Use of parity bits* or *ECC (error correction codes)* causes much less redundancy.

RAID Striping example: improves performance



*RAID is seamless to File System. File System sees the RAID system (set of disks) as a single large logical disk (i.e., a **Volume**)*

Different RAID Organizations/Schemes (also called Levels)

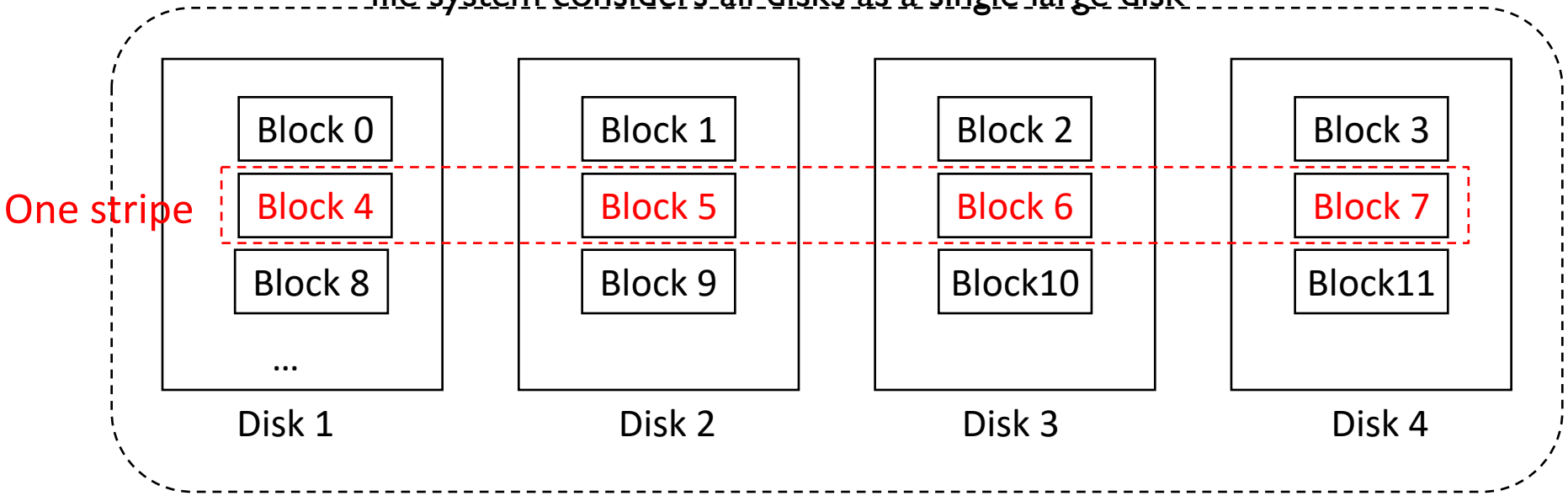
- RAID Level 0: block level striping (no redundancy)
- RAID Level 1: mirroring
- RAID Level 2: bit level striping + error correcting codes
- RAID Level 3: bit level striping + parity
- RAID Level 4: block level striping + parity
- RAID Level 5: block level striping + distributed parity
- ….

RAID 0: Block Level Striping

data: in blocks; adjacent blocks go into different disks

one block can be k sectors

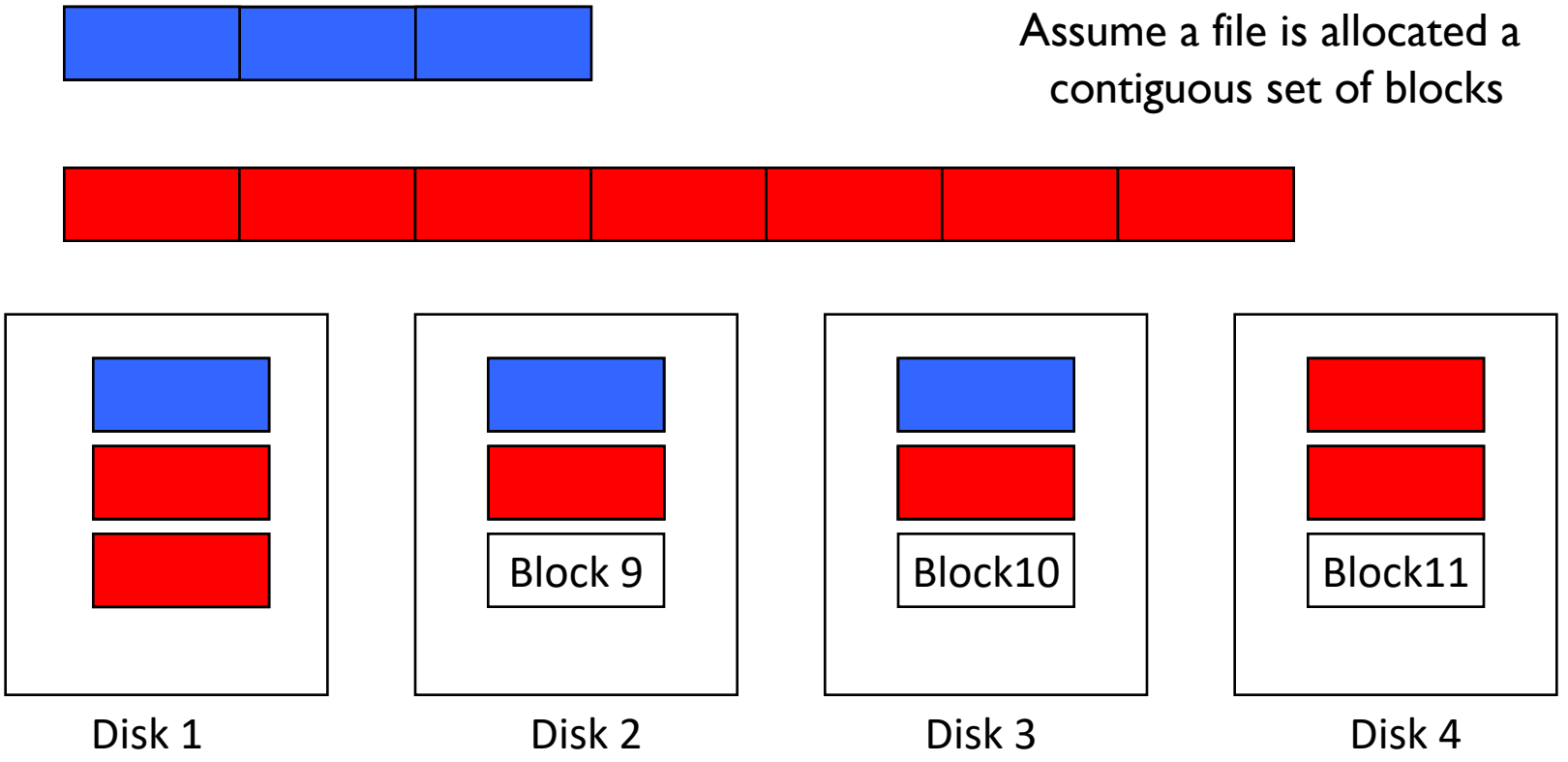
file system considers all disks as a single large disk



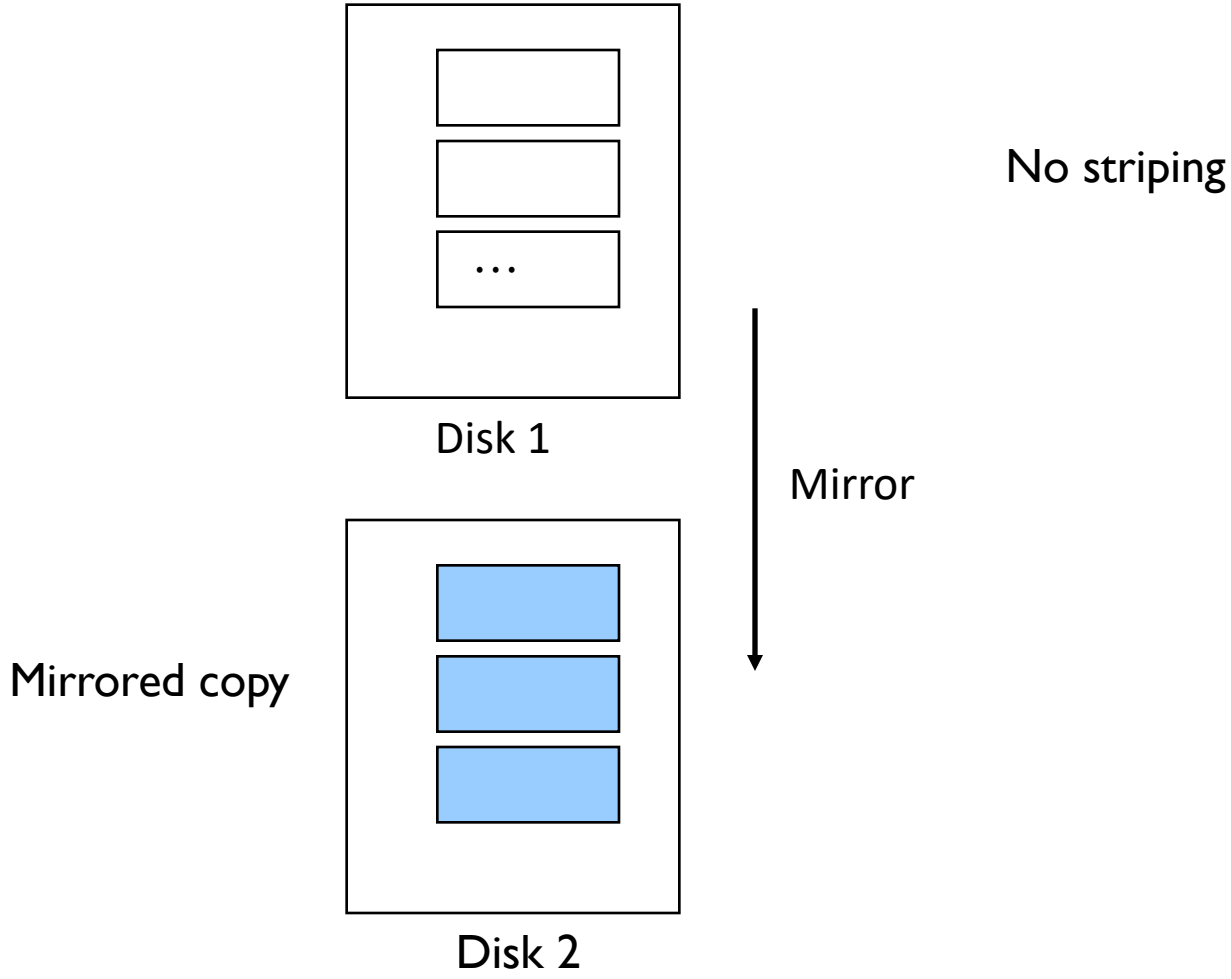
No redundancy;

parallel read/write for large data transfers (larger than block size)

RAID 0


Assume a file is allocated a contiguous set of blocks
Block 9
Block10
Block11
Disk 1
Disk 2
Disk 3
Disk 4

RAID 1: Mirroring



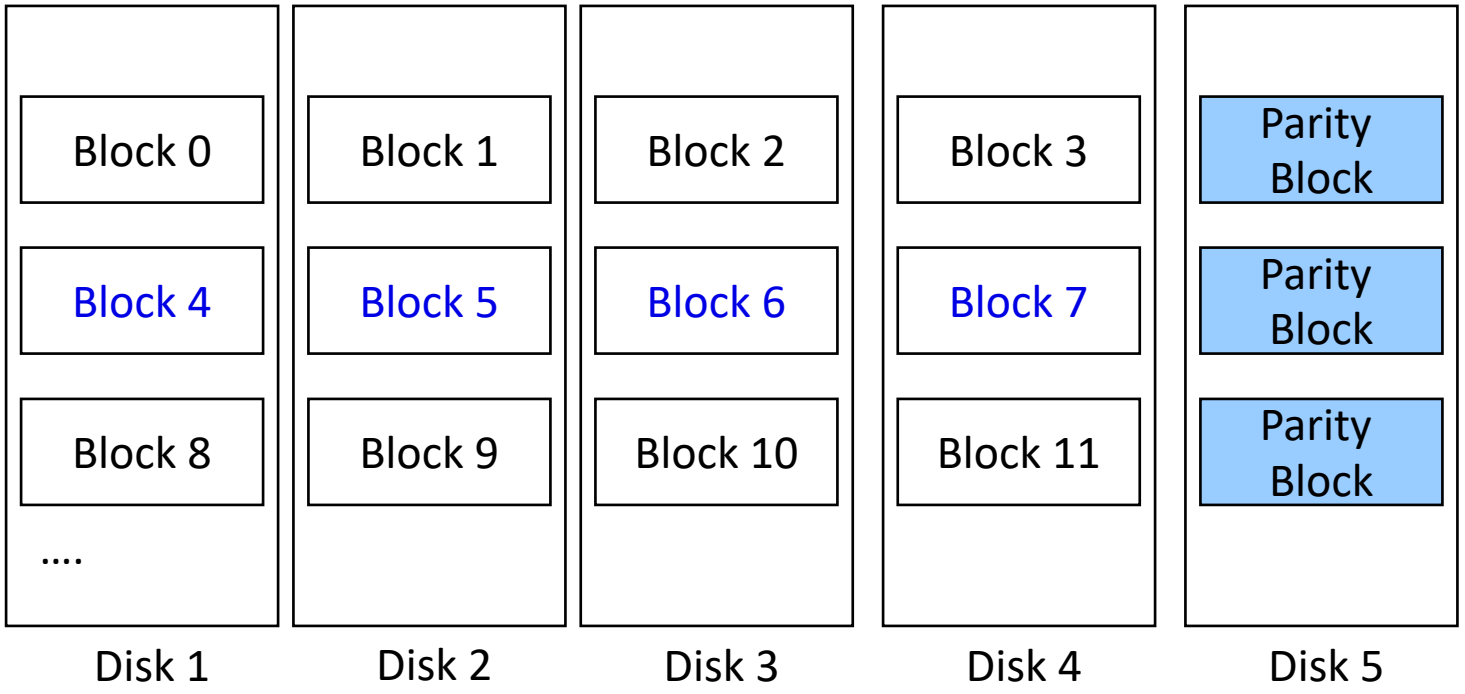
RAID 1

- We are just mirroring the disks (copying one disk to another one).
- Without striping: no performance gain, except for reads (doubled read-rate)
- Reliability provided. If one disk fails, data can be recovered from the other disk.
- If there are originally N ($N \geq 1$) disks; we need N more disks to mirror
  - Quite costly in terms of disks required. This cost is for reliability. We can express the cost as:

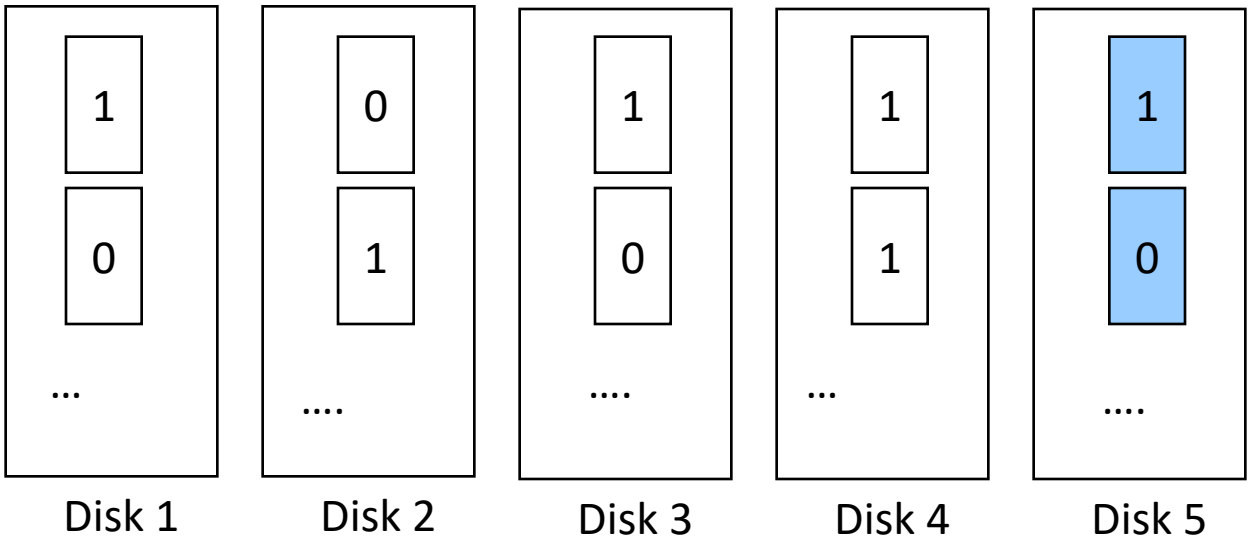
overhead/data = 1/1

RAID Level 4

- Uses parity information. Uses block-level striping.
- Computes the parity of k blocks and store it in the parity disk.
- Can recover from single disk failure.



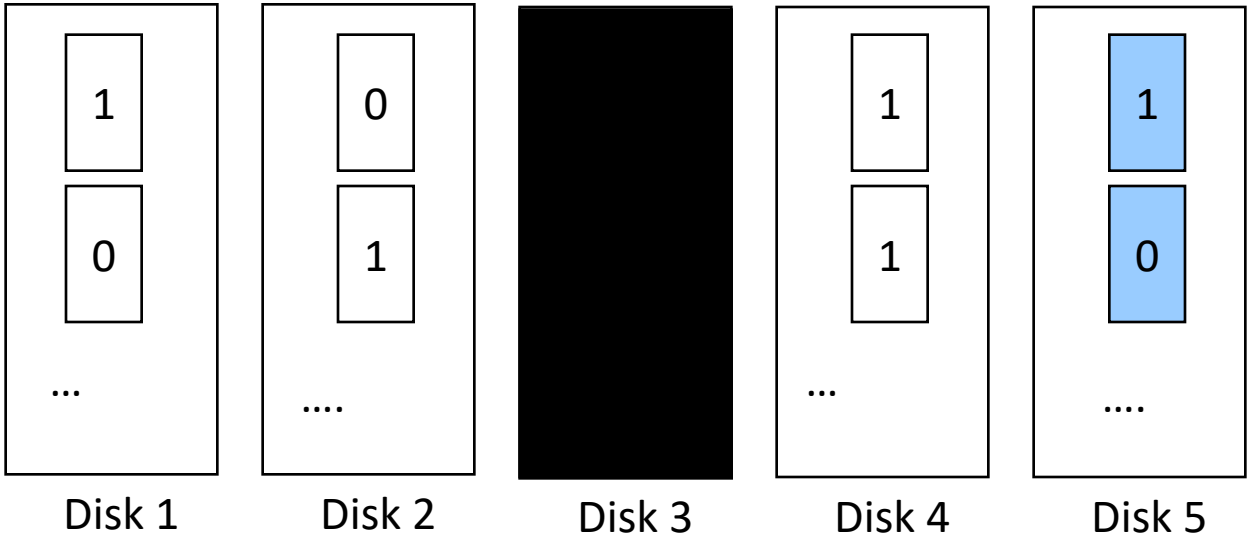
RAID 4: failure example



Even parity is used here

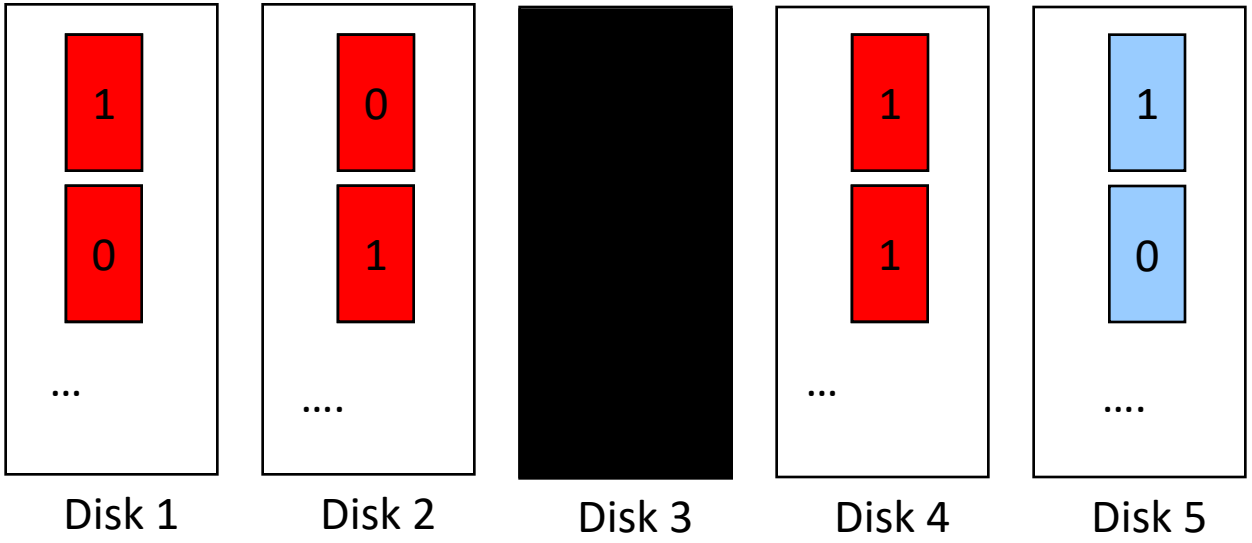
RAID 4: failure example

Let one disk fail! How can we recover its data



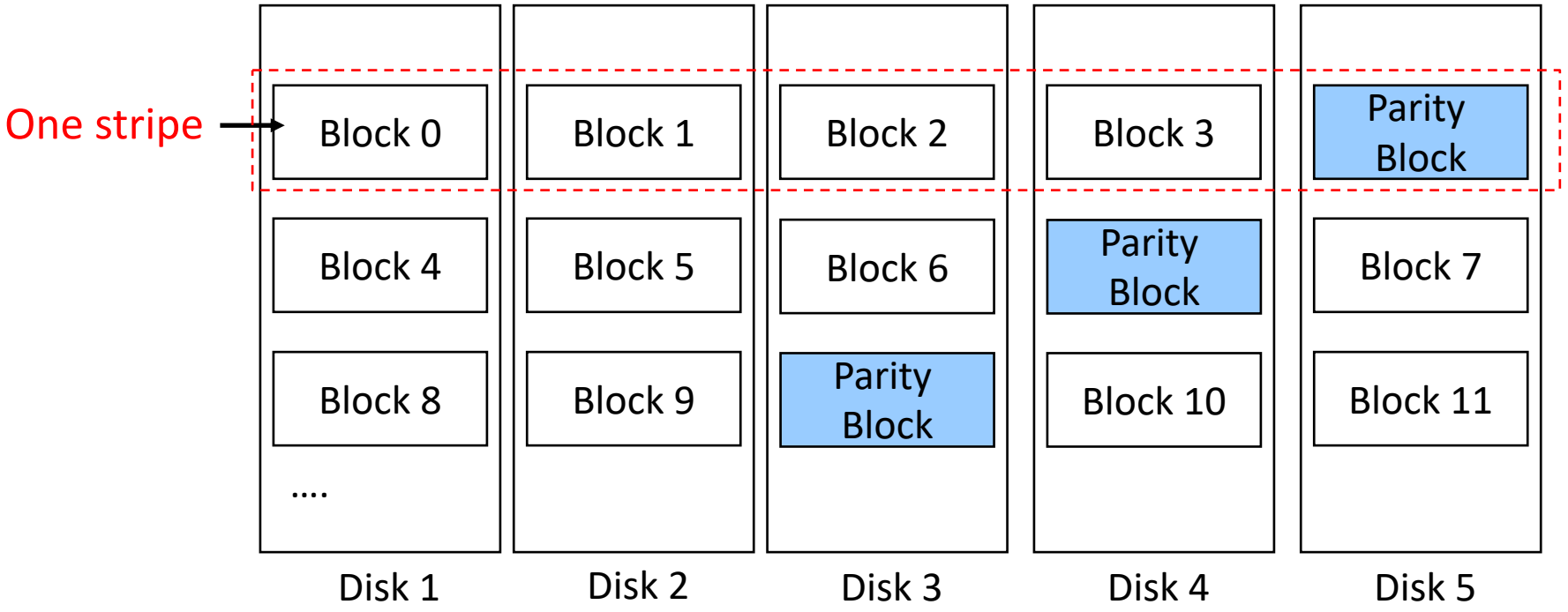
Look to disks 1, 2, 4, and 5. compute the parity and according to that generate the content of disk 3.

RAID 4: failure example



RAID Level 5

- Parity blocks are distributed on other disks. Similar to RAID 4.
- Load on parity disk is distributed in this way. Different methods exist to distribute parity blocks. One way is below (called rotation parity – left asymmetric)



RAID Level 6

- Similar to RAID level 5 but uses not only a single parity bit, but multiple ECC bits to guards against multiple disk failures
- Called also as: P+Q scheme.
- Reed-Solomon codes are used as ECC code.
- Example: 2-bits ECC code can be used for every 4-bits data.
- Can be multi-dimensional.

Abstractions provided by OS

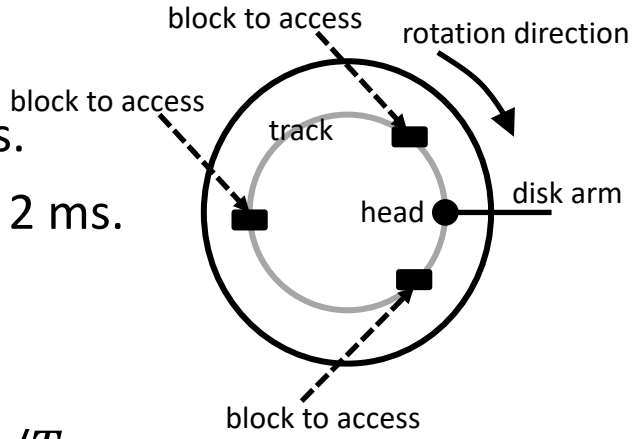
- Major OS jobs are to manage physical devices and to present appropriate abstractions to applications
- For hard disks, the OS provides two abstractions:
  - Raw device: the device is an array of data blocks. Blocks can be read or written by specifying their numbers (addresses).
  - File system: the OS provides file abstraction and interface. OS queues and schedules the interleaved requests from several applications accessing files.

Disk I/O Performance

- Given the performance numbers of a disk, we can calculate the time, $\boldsymbol{T}$, needed to do a disk I/O operation of certain size (A). A : transfer size

$$T = T_{seek} + T_{rotation} + T_{transfer}$$

- $T_{rotation}$ can be found from RPM. Example:
  - RPM = 15000 rpm
  - Time per rotation = (1/15000)x60x1000 = 4 ms.
  - Average rotational latency = $T_{rotation}$ = 4/2 = 2 ms.
- $T_{transfer}$ = A / Max-Transfer-Rate



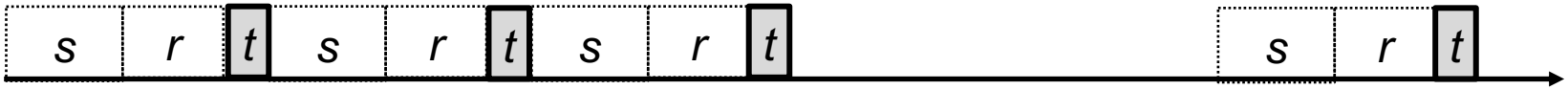
- Number of disk I/Os per second (each A-sized) = $1/T$
- Rate of disk I/O = X= $A \cdot (\frac{1}{T})$ (also called effective-bandwidth or effective datarate or throughput or effective transfer rate).

Disk I/O Performance

$T = T_{seek} + T_{rotation} + T_{transfer} = s + r + t$ $\quad where \quad t = A/peakBW$

$X = (1 / T) \times A$: disk I/O data-rate (MB/s)

Random workload (small transfer size, such as 1 block)



positioning time

Random workload consists of small sized transfers, like 4KB, happening to/from random locations on the disk.

Sequential workload (large transfer size, many many physically contigous blocks)

Sequential workload consists of large sized transfers, like 100 MB of contiguous data on disk need to be transferred to memory.

Disk I/O Performance Example

- If A = 4 KB (random workload) (small random transfers)
  - $T_{transfer}$ = 4 KB / 125 MB/s ~= 30 microseconds (0.03 ms)
  - T = 4 ms (seek t)+ 2 ms (rota.)+ 0.030 ms = 6.03 ms.
  - Number of IOs per second = 1000/6.03 = 166
  - Rate of IO, X = 4 KB x (166) = 0.66 MB/s
- If A = 100 MB (sequential workload) (large sequential transfers)
  - $T_{transfer}$ = 100 MB /125 MB/s = 0.8 s (800 ms)
  - T = 4 ms (seek t) + 2 ms (rota.) + 800 = 806 ms.
  - Number of I/Os per second = 1000/806 = 1.24
  - Rate of IO = X = (1000/806) x 100 MB ~= 125 MB/sec

published performance numbers for a disk

Capacity	300 GB
RPM	15000
Average Seek	4 ms
Max Transfer Rate (Peek Rate)	125 MB/s
Connects via	SCSI
Cache	16 MB

avg rotational latency: 2 ms

Bandwidth of a disk

- Disk bandwidth is measured in bytes per second (for large sequential transfers). We can have two types of disk bandwidth:
  - Sustained bandwidth (SB): average data rate during a large transfer, counting only the time when data stream is actually flowing
    - SB = A/ $T_{transfer}$
    - This is the maximum bandwidth (surface bandwidth), i.e, peak rate
  - Effective bandwidth (EB) – average over the entire I/O time, including positioning time.
    - $EB = A \ / \ (T_{seek} + T_{rotation} + T_{transfer})$

Effective bandwidth < Sustained bandwidth

Latency

- Access latency (L) – amount of time needed to service a very small request.
  - Dominated by the seek time and *rotational latency* (positioning time). We can ignore transfer time.
  - $L = T_{seek} + T_{rotation}$
  - L = positioning time

RAID Performance

- Single request latency: latency of a single I/O request (for small amount of data – 1 block – 4 KB for example). L.
- Steady-state throughput: the total bandwidth of many *concurrent* requests.
- Workloads:
  - Sequential I/O (request for a large contigous chunk – large sequence of adjascent blocks to be read or written)
  - Random I/O: each request is small, and is to different location (block) of the disk. For example: a stream of requests, each at most for 4KB of data (1 block); different blocks.

RAID Performance

- Assuming a single disk can transfer data (effective transfer rate - throughput) at S MB/s under sequential workload and R MB/s under random workload. $S >> R$.
- For example: suppose avg seektime = 7 ms, avg rot. lat. = 3 ms; (max) transfer rate of disk = 50 MB/s; then:
  - T (for 10 MB) = 7 + 3 + 1000(10 / 50) = 210 ms = 0.21 s
  - S = (1/0.21) x 10 MB ~= 48 MB/s (throughput)
  - T (for 10 KB) = 7 + 3 + 10 KB / 50 MB/s ~= 10.2 ms.
  - R = (1/0.0102)* 4 KB ~= 0.380 MB/s (throughput)
- Assume single request latency of a single disk is:
  - $L = T_{seek} + T_{rotation}$

RAID Level 1

- Mirroring applied. Each disk block has a copy on the other disk. $N=2$. There are two disks.
- Single request latency: identical to that of a single disk: L.
- Steady-state sequential throughput (bandwidth):
  - read: S MB/s: with a sophisticated approach we can achieve $2S$ MB/s.
  - write: S MB/s (writing each logical block, will write to two disks.
- Steady-state random throughput (bandwidth):
  - read: $2R$ MB/s: distribute the small reads (1 block) accross all disks.
  - write: R MB/s: each write will cause 2 physical writes.

D0	D1
0	0
1	1
2	2
3	3
4	4

Reliability

- A head crash in a fixed HDD disk generally destroys the data.
- SSDs are more reliable than HDDs.
- RAID is not solving all reliability problems.
  - It is for permanent failures of disks (called full-disk failure).
- There are also other kinds of failures. Disk did not fail completely, but blocks may be in error.
  - Pointers in FS may be wrong.
  - Incomplete writes may happen (corrupt data).
  - Accidental write over file data.
  - ....

Error detection and correction

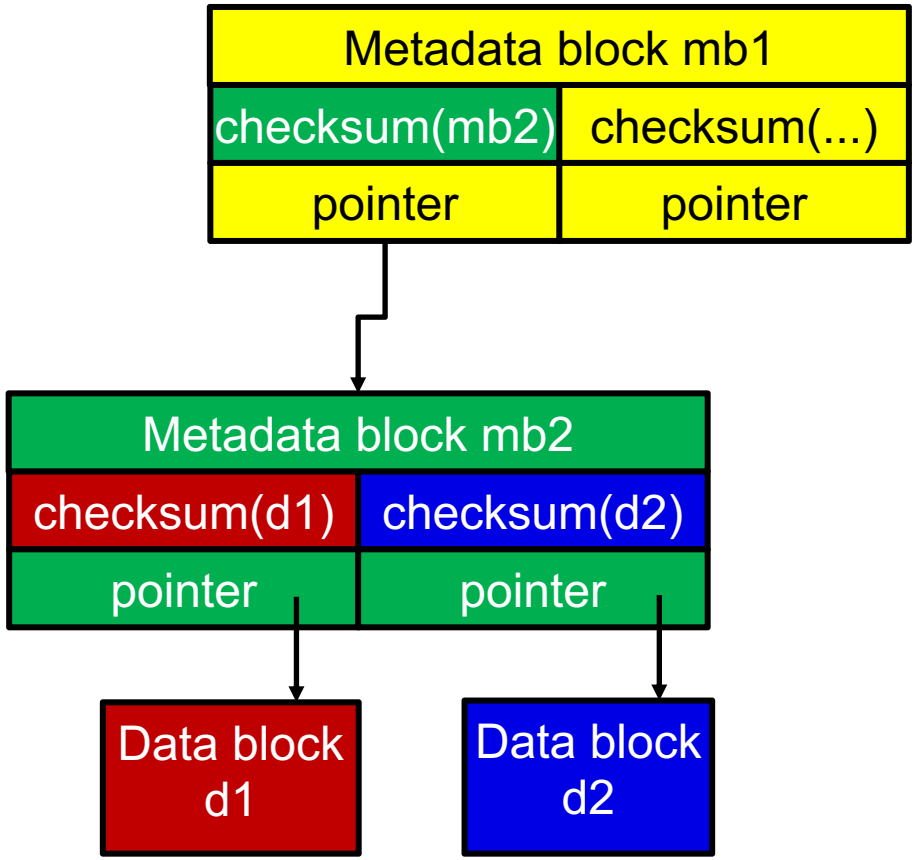
- Between writing a block and reading the block, are some bits changed (bit errors)? We need to detect, if changed.
  - Checksums can be used to detect errors. Uses modular arithmetic to compute, store and compare values on fixed-length words.
  - Checksums are computed over the data of the blocks and are stored in the disk, together with the block or somewhere else.
  - Various checksum algorithms.
  - Upon reading a block, checksum for the block can be computed and compared with the stored checksum for the block.
- ZFS file system, for example, checksums all metadata and data.
- Checksums are stored together with block pointers.

Error detection and correction

- Error correction codes are also used.
- Can be used to detect and also correct errors.
- Correction is done using algorithms.
- Disks use per-sector ECC.
- SSDs use per-page ECC.
- ECC calculated using the data in the sector or block or page.

ZFS file system (Zettabyte File System)

- Uses checksums to solve various problems related to blocks of the file system.
- ZFS maintains internal checksums for all blocks, including data and metadata blocks.
- Checksums are not kept with the blocks that are checksummed. Stored with the pointers to blocks.



References

- Operating System Concepts, Silberschatz et al.
- Modern Operating Systems, Andrew S. Tanenbaum et al.
- OSTEP, Remzi Arpaci-Dusseau et al.
- Operating Systems, Principles and Practice, Anderson et al.