Chapter 11: Mass Storage Structure

CS 3423 Operating Systems
Fall 2019
National Tsing Hua University

Outline

- Overview of Mass Storage Structure
- Disk Structure
- Disk Attachment
- Disk Scheduling
- Disk Management
- Swap-Space Management
- RAID Structure
- Stable-Storage Implementation

Objectives

- To describe the physical structure of secondary storage devices and its effects on the uses of the devices
- To explain the performance characteristics of mass-storage devices
- To evaluate disk scheduling algorithms
- To discuss operating-system services provided for mass storage, including RAID

Overview of Mass Storage Structure

- Magnetic disks, hard disk drives (HDD)
  - bulk of secondary storage of modern computers
- Solid State Drive (SSD)
  - uses nonvolatile memory (NVM) for bulk storage
- Storage Arrays
  - Multiple HDD or SSDs forming an array
- Storage Area Networks

The First Commercial Disk Drive

1956
IBM RAMDAC computer included the IBM Model 350 disk storage system

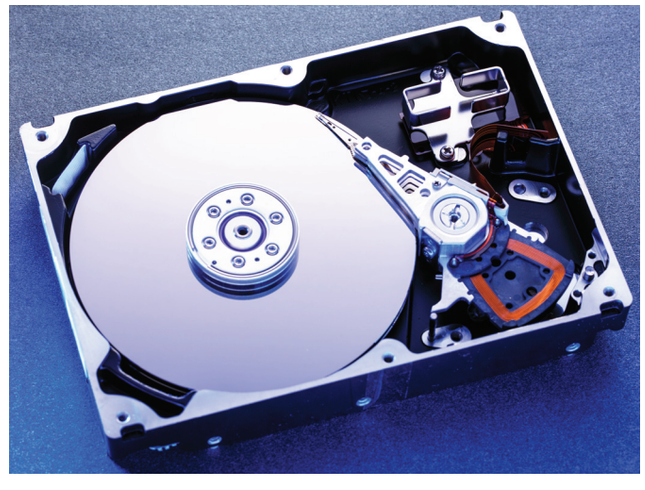
5M (7 bit) characters
50 x 24” platters
Access time = < 1 second

Common HDDs

1.8 inch

2.5 inch

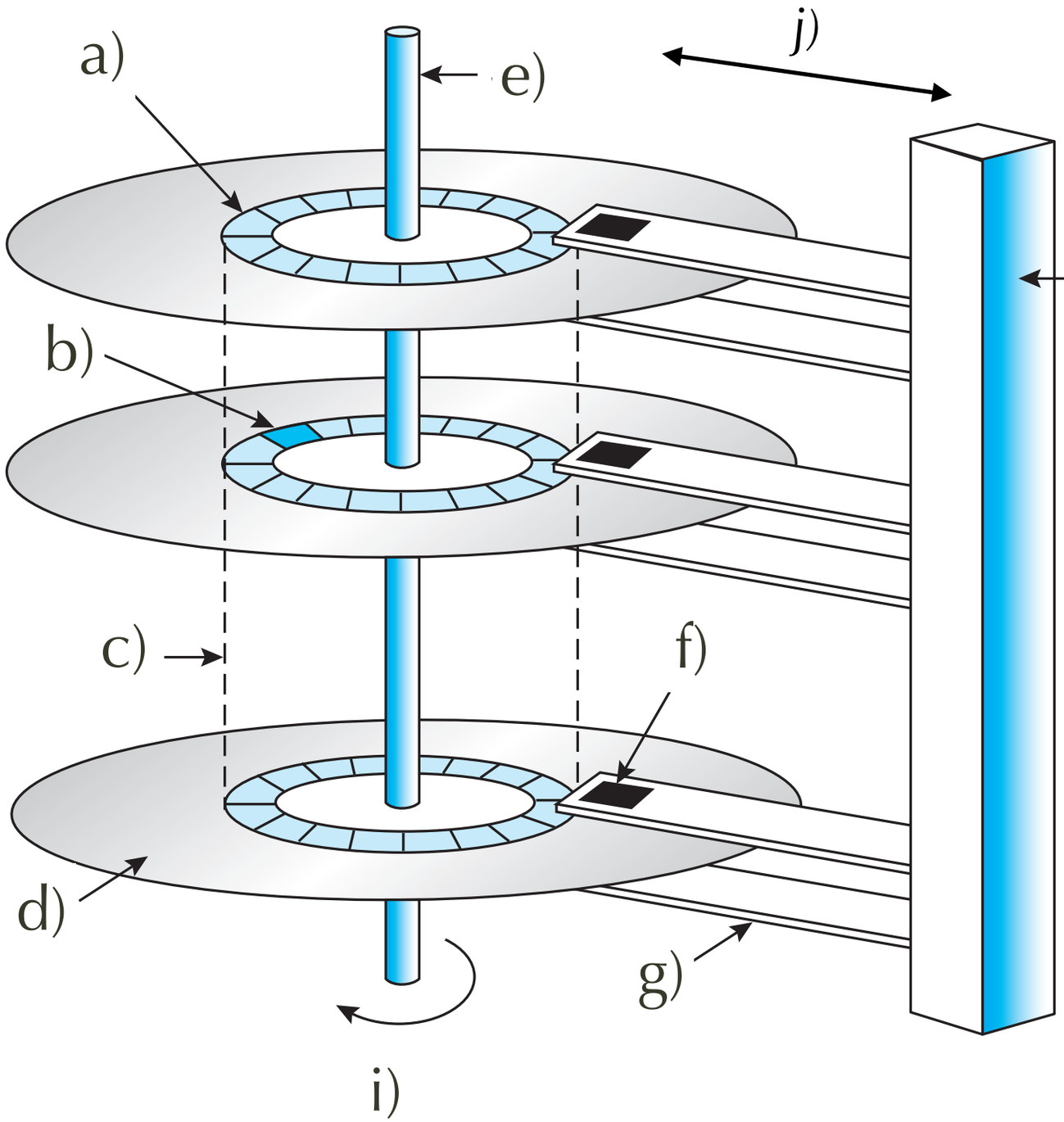
3.5 inch



- Platters range from .85" to 14" (historically)
  - Commonly 3.5", 2.5", and 1.8"
- Drives rotate at 60 to 250 times per second
  - 5400, 7200, 1000, 15000 RPM
- Range from 30GB to >8TB per drive

Moving-head Disk Mechanism

a) track
b) sector
c) cylinder
d) platter



e) rotation
f) read-write head
g) arm
h) arm assembly

HDD Mechanism

- Read/write heads gliding over both sides of platter surface
  - Several microns over thin air (helium)
  - Disk head contacting disk surface => head crash
- Disk organization
  - **tracks** = Concentric rings on a given platter
  - **cylinder** = set of tracks on all platters at a given arm position (radius)
  - a track is divided into multiple **sectors** = minimum transfer unit
- Random access requires
  - **seek**: moving read/write head in or out to target cylinder
    (all arms move in and out together as one unit)
  - **rotation**: disk spins to the target sector within a track

Hard Disk Performance

- Positioning (random-access) time
  = seek time +rotational latency
- seek time = time to move disk arm to desired cylinder
  - from 3ms to 12ms – 9ms common for desktop drives
  - Average seek time measured or calculated based on 1/3 of tracks
- rotational latency = time to rotate to desired sector
  - Average latency = ½ latency
  - 1 / (RPM / 60) = 60 / RPM

Spindle [rpm]	Average latency [ms]
4200	7.14
5400	5.56
7200	4.17
10000	3
15000	2

(From Wikipedia)

Hard Disk Performance

- **Access Latency** = **Average access time**
  - = average seek time + average latency
  - For fastest disk 3 ms + 2 ms = 5 ms
  - For slow disk 9 ms + 5.56 ms = 14.56 ms
- Transfer Rate –
  - theoretical – 6 Gb/sec
  - Effective Transfer Rate – real – 1Gb/sec
- Average I/O time
  - = average access time + (amount to transfer / transfer rate) + controller overhead

Example of HDD Performance

- Configuration
  - 4KB block, 7200 RPM, 5ms seek
  - 1Gb/sec transfer rate, .1ms controller overhead
- Transfer time for 1 GB
  - = 4KB / 1Gb/s * 8Gb / GB * 1GB / 1024^2KB = 32 / (1024^2) = 0.031 ms
- Average I/O time
  - 5ms (seek) + 4.17ms (read) + 0.1ms (overhead) + transfer time
  - Average I/O time for 4KB block = 9.27ms + .031ms = 9.301ms

Nonvolatile Memory Technologies: NAND flash

- Flash memory
  - "block" = minimum erase unit, contains several page
  - "page" = minimum read/write unit.
  - Erase (set to all 1's) , Write (change bits from 1 to 0)
- Issues with Flash: Writing
  - high power, long latency, increases flash's wear & tear
  - => **flash translation layer** (FTL) for tracking logical-to-physical mapping and erase state
- Limited rewrite cycles (~100,000 cycles)
  - => **wear leveling** to even out wear on some "hot" blocks

Forms of Solid State Storage Devices

- Removable storage
  - USB thumb drives, SD card, Compact Flash
  - contains controller and FTL and wear leveling
- Solid state disks
  - drop-in replacement for HDD
  - controller + buffer is for performance optimization
- hybrid = SSD as cache for HDD

Solid-State Disks (SSD)

- Pros (relative to hard disk drives, HDD)
  - more robust (mobile) and much faster
  - No moving parts, so no seek time or rotational latency
- Cons (relative to HDD)
  - More expensive per MB, lower capacity
  - Maybe have shorter life span
- Busses can be too slow -> connect directly to PCI for example

RAM Drives

- Use DRAM instead of NVM
  - Need to be powered!
  - Doesn't actually have to be a separate device; could be just RAM implemented as file system
- Why?
  - much faster than flash or other NVM, useful for temporary file system
  - e.g., /tmp, of type tmpfs (RAM drive)

Computer-Disk Interface

- Host talks through I/O ports to I/O busses to disk
  - Host controller (on computer) talks to Disk controller in drive or storage array
- HDD attached to computer via I/O bus
  - EIDE, ATA, SATA, USB, Fibre Channel, SCSI, SAS, Firewire, Thunderbolt
- SSD may be attached directly to system bus
  - PCI bus, NVMe (NVM express) => faster!
  - could also attach to HDD interface for compatibility

Logical block

- basic unit of transfer
  - maps to a physical sector or a flash page
- HDD
  - Sector 0 = first sector of the first track on **outermost** cylinder
  - Sequential thru tracks on a **cylinder** from outer to inner track
  - Bad sectors => skip
  - Constant **angular** velocity (CAV) vs. constant **linear** velocity (CLV)
- SSD
  - map (chip, block, page) => (array of logical blocks)

HDD Scheduling

- OS objectives for disk access
  - fast access time
  - disk bandwidth
- What OS can do
  - minimize seek time ≈ seek distance
- Disk bandwidth is
  - total number of bytes transferred ÷
    total time from first request to completion of last transfer

HDD Scheduling (Cont.)

- Disk I/O requests can be made by
  - OS, System processes, User processes
- Parameters to I/O requests
  - input / output mode
  - disk address, memory address
  - number of sectors to transfer
- OS maintains queue of requests (per disk or device)
  - Idle disk can immediately work on I/O request
  - busy disk means work must queue
  - Optimization algorithms for when a queue exists

Disk Scheduling (Cont.)

- Drive controllers have small buffers
  - For managing I/O requests of varying "depth"
  - For one or many platters
- Scheduling algorithm servicing disk I/O requests
  - First-come first serve (FCFS)
  - Shortest Seek Time First (SSTF)
  - Elevator algorithm (SCAN) and LOOK
  - C-SCAN and C-LOOK

FCFS

queue = 98, 183, 37, 122, 14, 124, 65, 67

head starts at 53

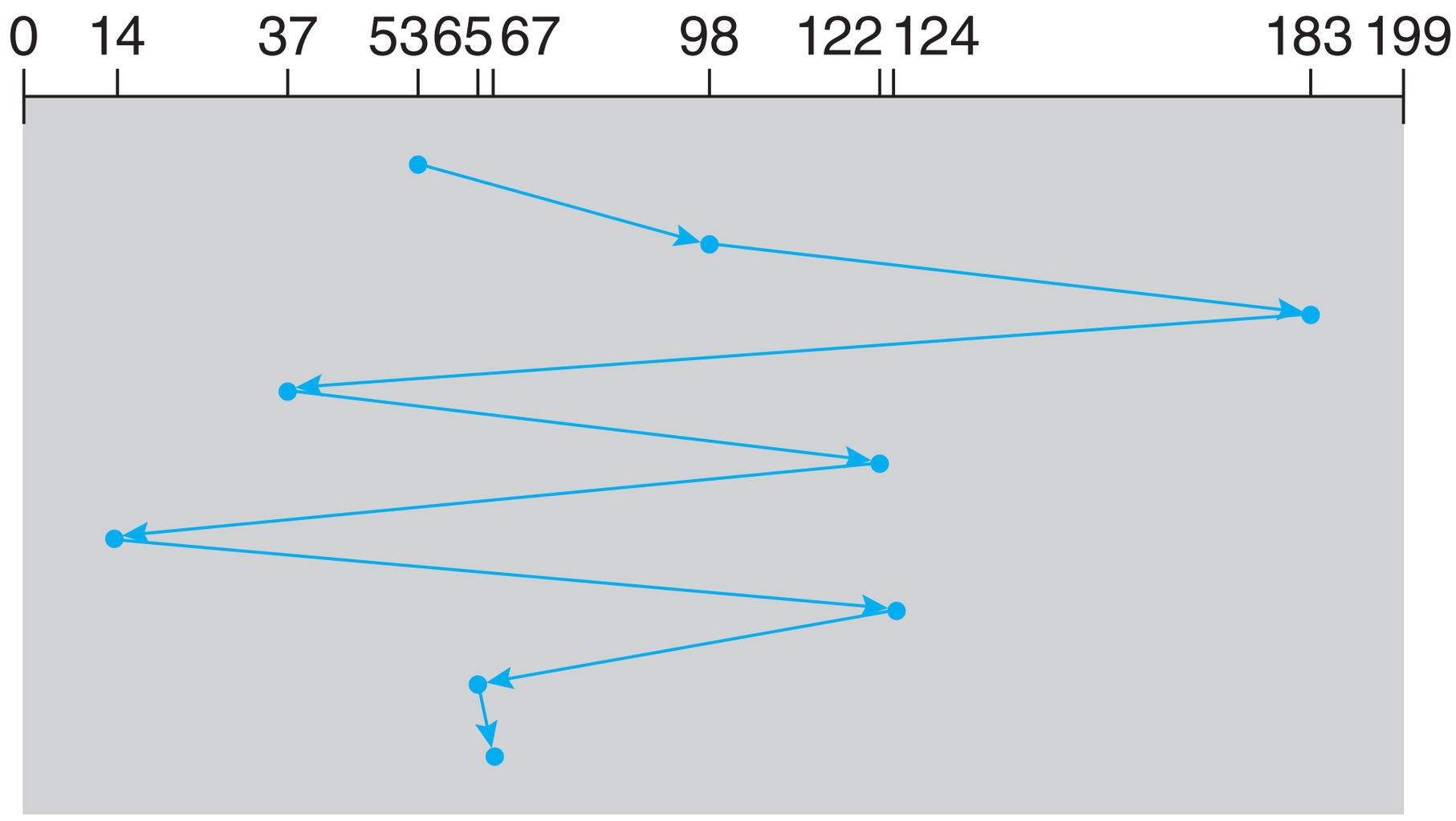


Illustration shows total head movement of 640 cylinders

SSTF

- Shortest Seek Time First
  - selects request w/ min seek time from current head position
  - a form of SJF scheduling
- Issue
  - tends to favor middle cylinders over innermost and outermost ones
  - may cause starvation of some requests

SCAN (elevator algorithm)

- Head moves from one end to other end, then reverse direction

queue = 98, 183, 37, 122, 14, 124, 65, 67

head starts at 53

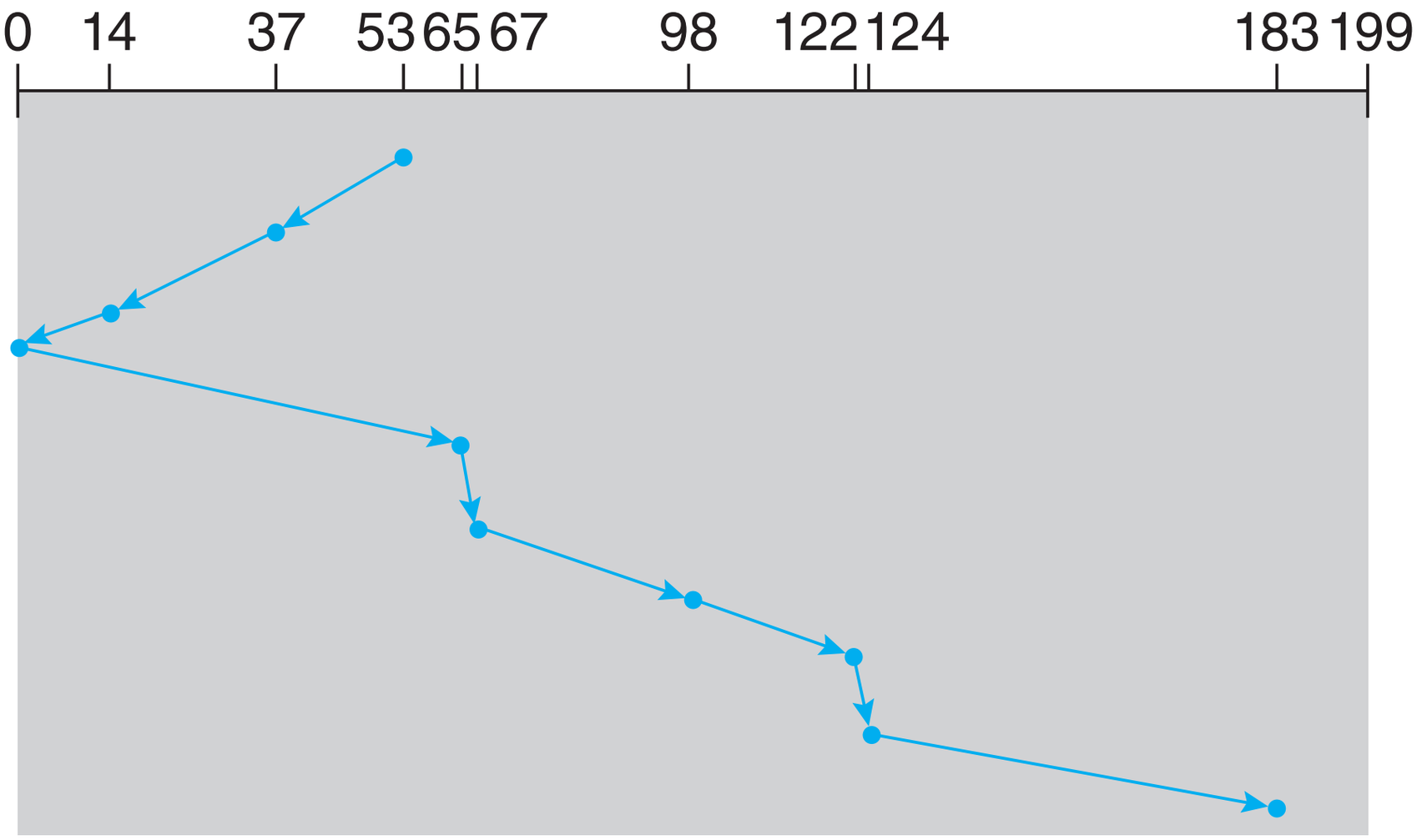


Illustration shows total head movement of 208 cylinders

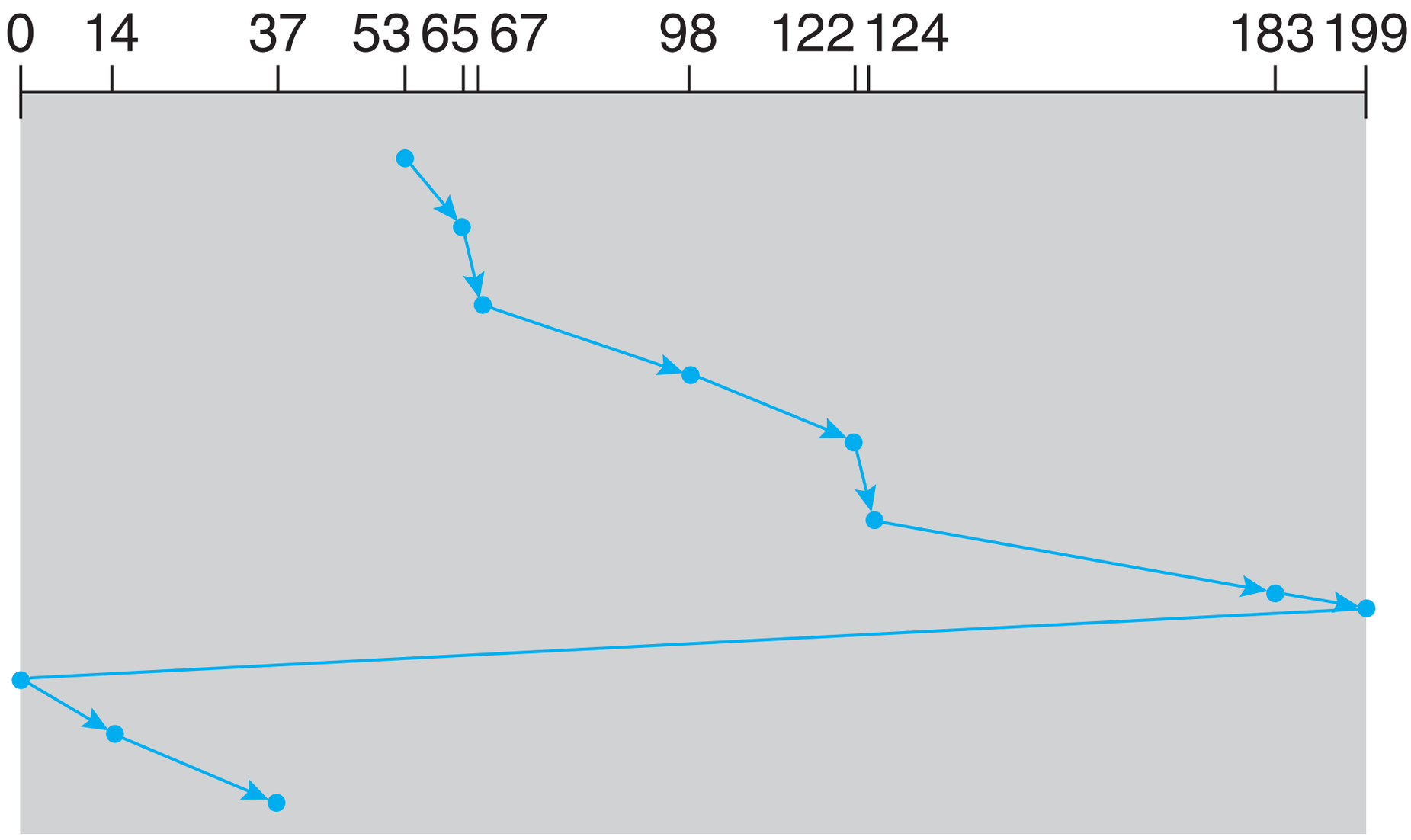
C-SCAN

- Head moves from one end of disk to the other,
- immediately wraps to beginning and start over
  - think 'C' = "circular"
- Properties
  - Provides a more uniform wait time than SCAN
  - No starvation
- Treats the cylinders as a circular list that wraps around from the last cylinder to the first one

C-SCAN (Cont.)

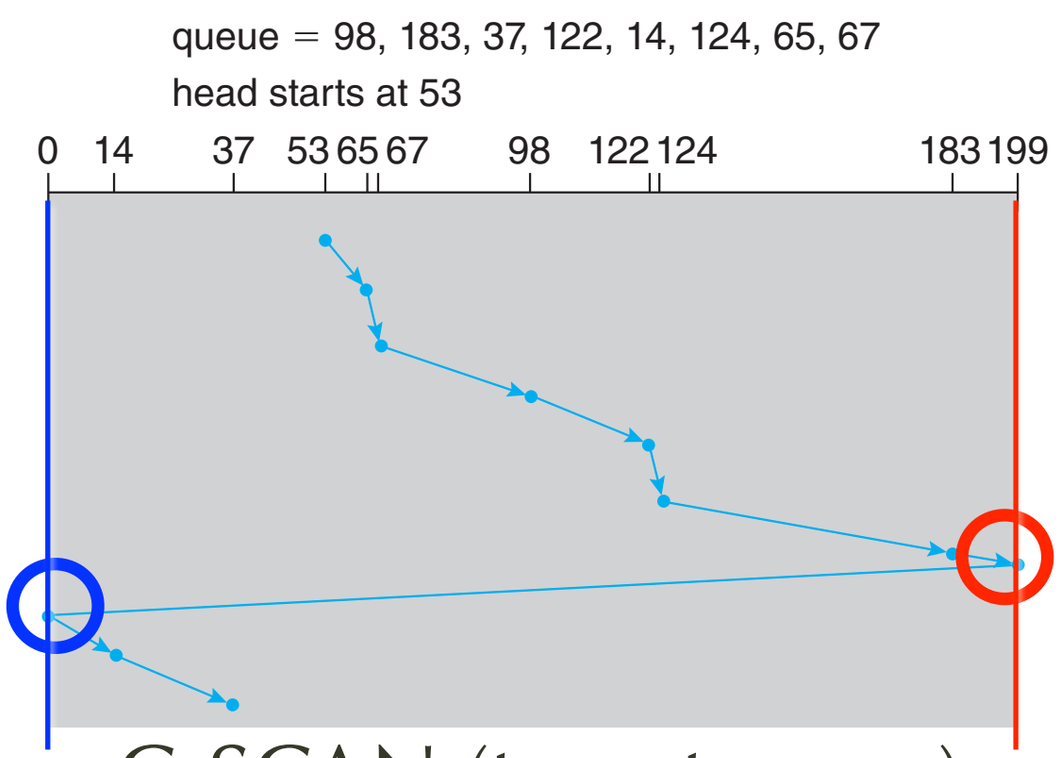
queue = 98, 183, 37, 122, 14, 124, 65, 67

head starts at 53

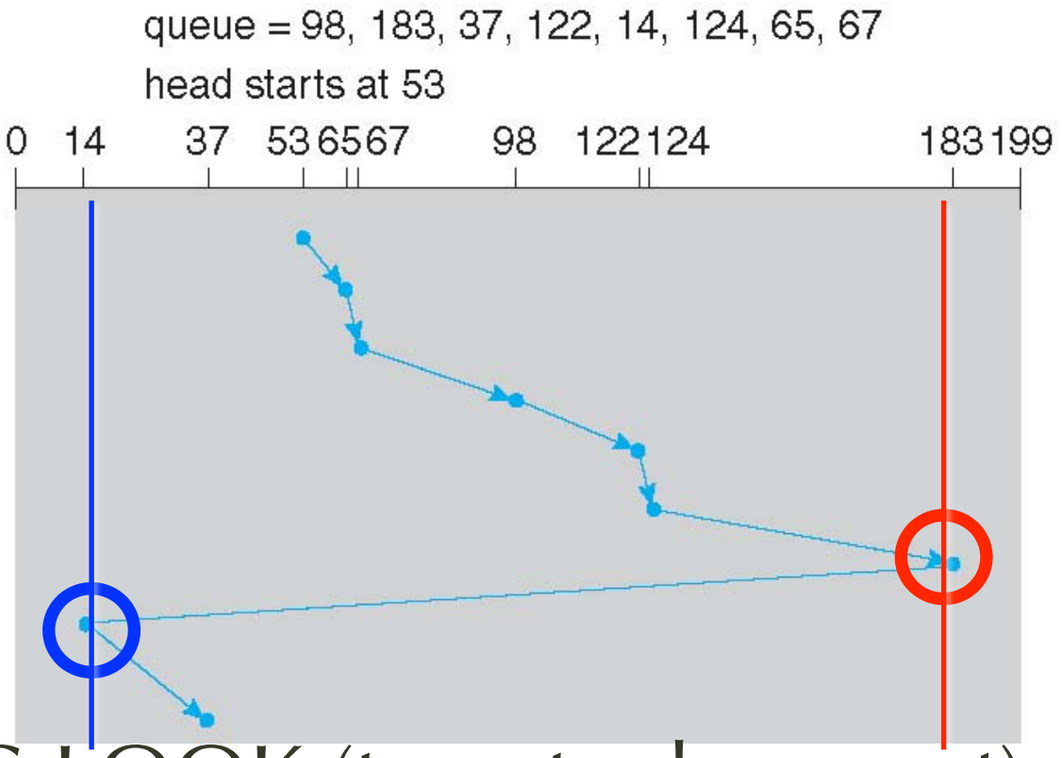


C-LOOK

- LOOK a version of SCAN
- C-LOOK a version of C-SCAN
- Disk arm goes only as far as the last request in each direction (instead of end of disk)



C-SCAN (to extremes)



C-LOOK (to actual request)

Selecting a Disk-Scheduling Algorithm

- SSTF
  - common, has a natural appeal
- SCAN and C-SCAN
  - good for systems that place a heavy load on disk
  - Less starvation
- Performance depends on
  - number and types of requests
  - the file-allocation method
  - metadata layout

Selecting a Disk-Scheduling Algorithm

- Separating policy from mechanism
  - disk-scheduling algorithm as a separate module of OS
- Default choice
  - SSTF or LOOK is reasonable
- Issues
  - Rotational latency is difficult for OS to calculate
  - Disk-based queueing affecting OS queue ordering efforts?

NVM Scheduling

- No seek or ration => FCFS type works
- Need to minimize write
  - Writing is much slower than reading, need erase
  - actually, read page, modify in memory, erase block, write page
  - also need garbage collection, wear leveling
  - need to avoid **write-amplification**

Error Detection and Correction

- Bit errors happen to disk
- Error Detection
  - Parity (even or odd), checksum
  - CRC (cyclic redundancy check)
- ECC
  - Error Correction Code / Error Correctable Code not only detects but also corrects errors
  - examples: Hamming code

Disk Management: Low-Level Formatting

- Also called physical formatting
- Dividing a disk into sectors
  - so that the disk controller can read and write
    - Usually 512 bytes of data but can be selectable
- Each sector can hold
  - header and trailer
    - sector or page number, checksum or ECC
  - data area

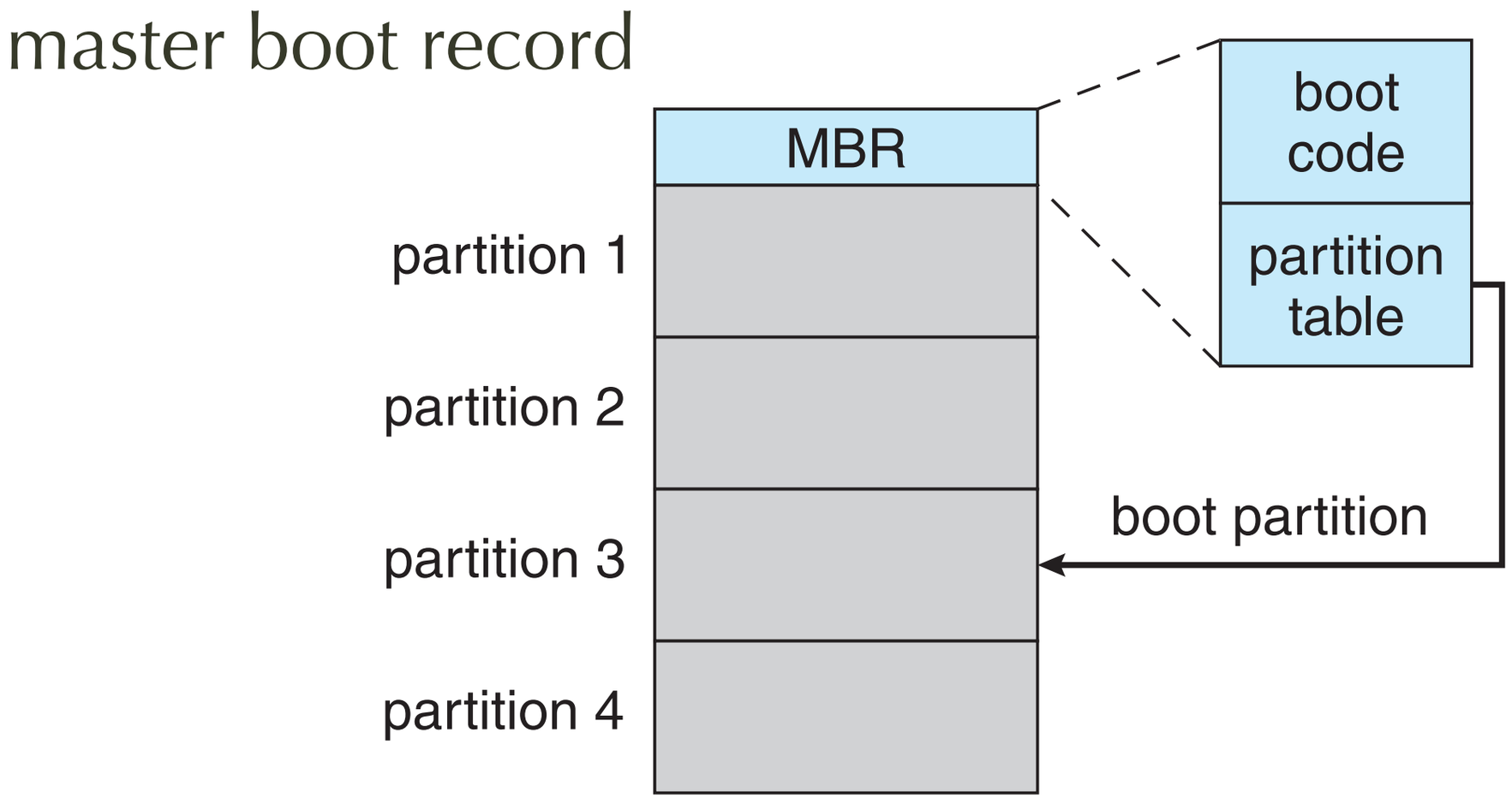
Disk Management: Logical formatting

- Partition
  - divide the disk into one or more groups of cylinders
  - each treated as a logical disk
- Logical formatting
  - means "making a file system" (ch. 13-15)
  - To increase efficiency, most FS group blocks into clusters
    - Disk I/O done in blocks
    - File I/O done in clusters = group of blocks

Block Management

- Some apps can do own block management
  - e.g., databases, keep OS out of the way
- Boot block
  - initializes the system
  - Bootstrap loader (bootloader) stored in boot blocks of boot partition
- Bad blocks handling
  - sector sparing

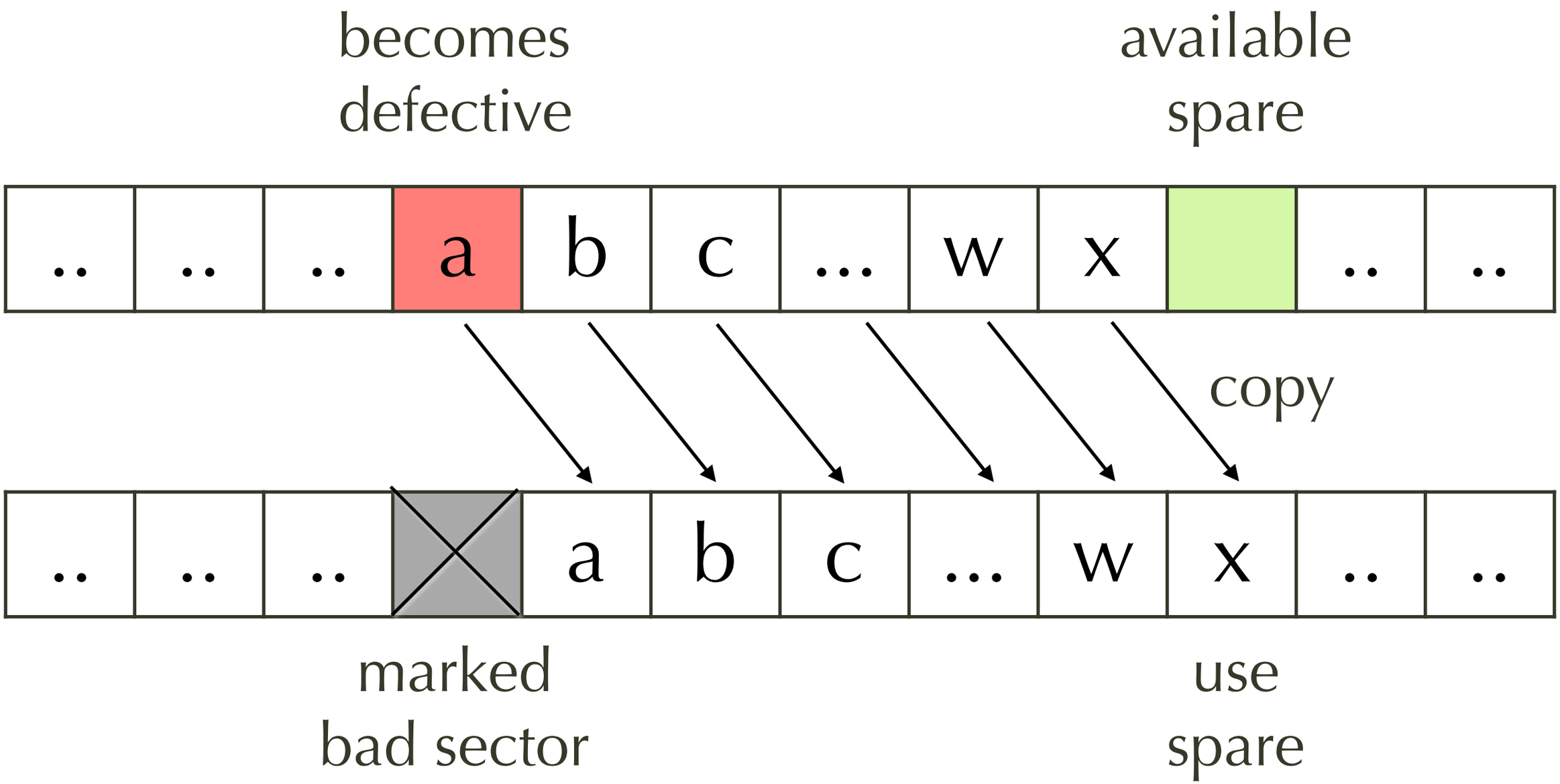
Booting from a Disk in Windows



Bad Blocks and Spare Sectors

- Defective sectors
  - some bits cannot be saved reliably (stuck 0, 1, etc)
  - found during low-level formatting or disk check
- Spare sectors
  - set-aside by low level formatter to replace defective sectors over time
- sector slipping
  - shift sector contents so the spare preserves contiguity

sector slipping

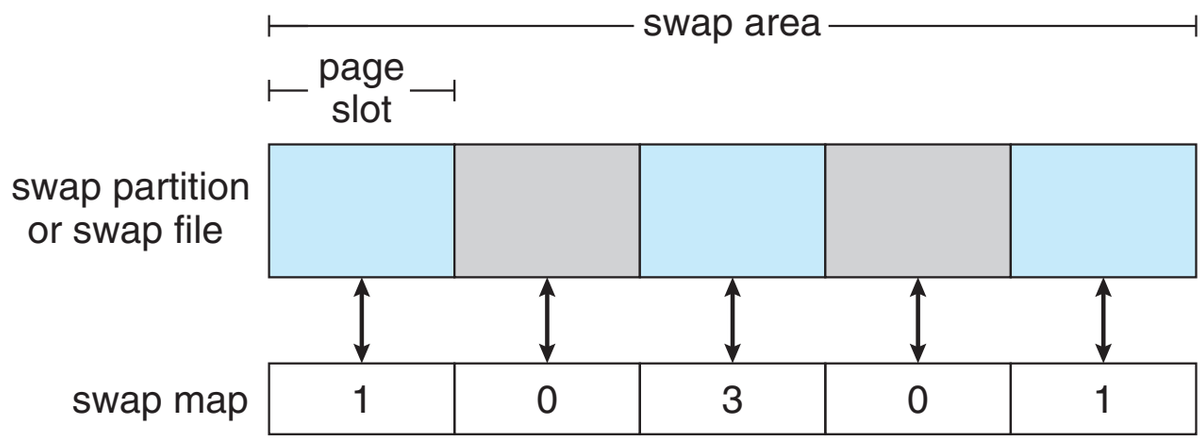


Swap-Space Management

- Virtual memory uses swap space on disk
  - as an extension of main memory
  - Less common now due to memory capacity increases
- Swap-space options
  - be carved out of the normal file system,
  - (more common) a separate disk partition (raw)
- Issues
  - What if a system runs out of swap space?
  - Some systems allow multiple swap spaces

Ways of Swap-Space Management

- Swap space allocation
  - allocates swap space when process starts (4.3 BSD. original Unix)
  - allocates swap space only when a dirty page is forced out of physical memory (Solaris 2)
- Swap space usage
  - holds text segment (the program) and data segment (4.3 BSD)
  - swap space for anonymous memory only (Solaris, Linux);
    text segment pages thrown out and reread from the file system as needed
- Swap map = array of counters for each page slot in swap area
  - Linux, 4.3BSD



Storage Attachment

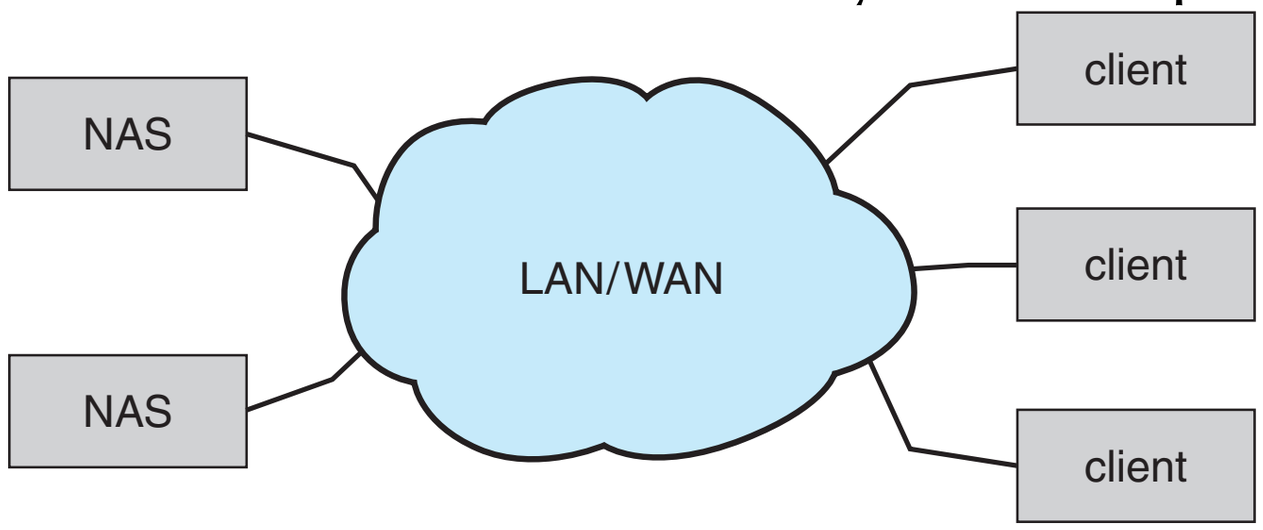
- host-attached storage
  - storage is attached to computer's I/O ports
- network-attached storage (NAS)
  - storage on (mainly local) network via RPC interface
- cloud storage
  - storage over **wide area** network
- storage-area networks and storage arrays

Interface for Host-Attached Storage

- Ports for HDD
  - SATA (Serial ATA) - most common today
  - IDE - popular before
- Plug-and-play
  - USB, FireWire, Thunderbolt, ...
- SCSI: ("skuzzy") Small Computer System
  - bus, up to 16 devices on one cable
  - SCSI initiator (host) requests operation
  - SCSI targets (device) perform tasks
  - target's device controller can control up to 8 logical units (disks)

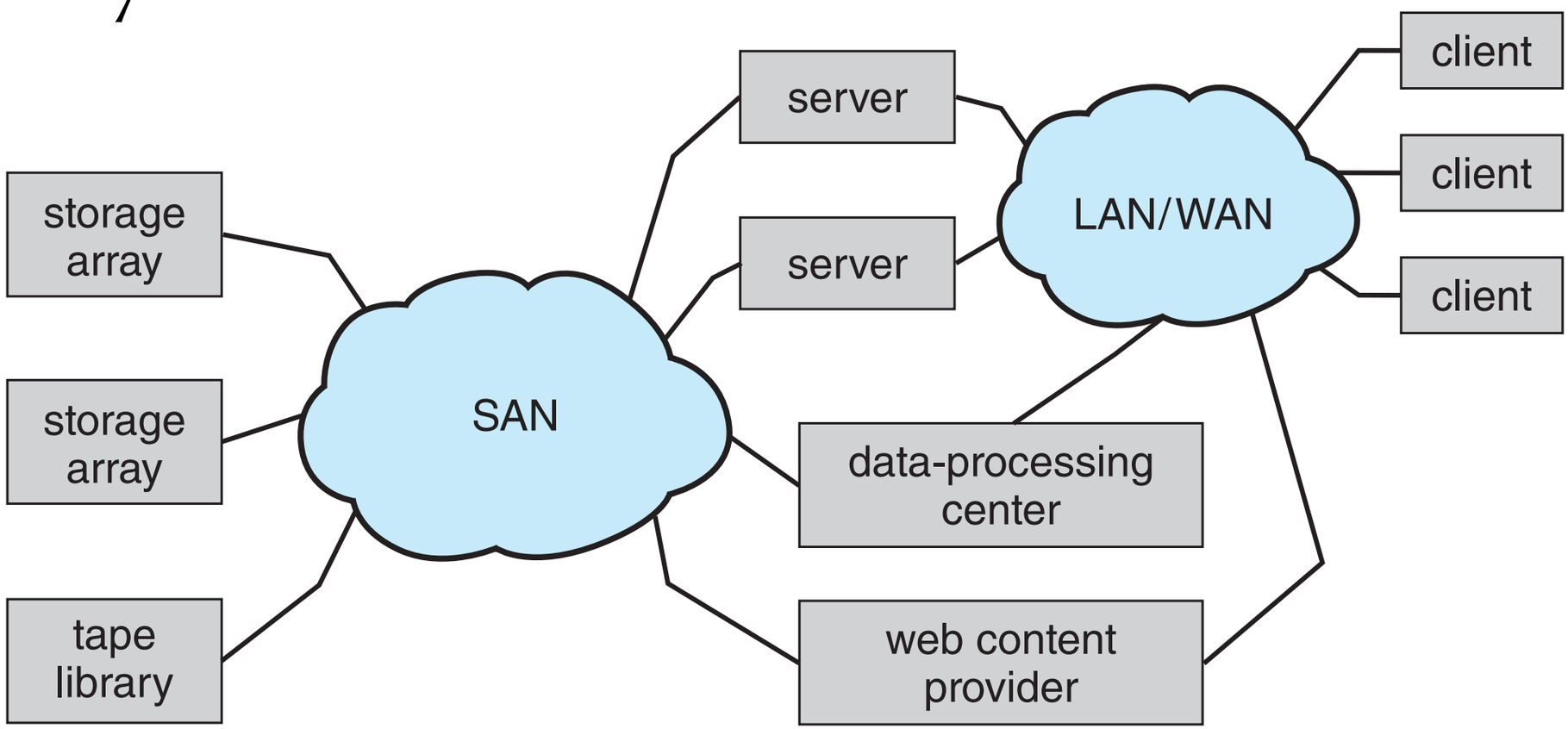
Network-Attached Storage (NAS)

- Storage made available over a network rather local bus
  - Remotely attaching to file systems
  - Examples: NFS (Unix, Linux) and CIFS (Windows)
- Implementation:
  - remote procedure calls (RPCs) between host and storage
  - TCP or UDP on IP network
  - iSCSI protocol uses IP network to carry the SCSI protocol



Storage Area Network

- Common in large storage environments
- Multiple hosts attached to multiple storage arrays - flexible

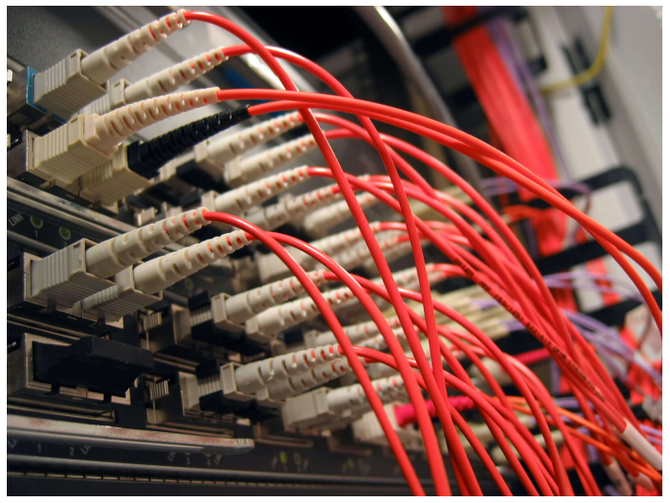


Storage Area Network (Cont.)

- Pros
  - Easy to add or remove storage
  - Easy to add new host and allocate it storage
- Storage networks vs. communications networks
  - Storage networks: (low-latency) Fibre Channel fabric
  - Combined storage+communication: iSCSI, FCOE

Interface for Storage Area Network

- Most common: Fiber Channel
  - high-speed serial architecture
  - Can be switched fabric with 24-bit address space –
    - many hosts to many storage units
    - I/O directed to bus ID, device ID, logical unit (LUN)
- iSCSI gaining popularity - simplicity
- InfiniBand - special-purpose bus architecture



Storage Array

- A unit between hosts and arrays of disks
  - Ports to connect hosts to array
  - From a few disks to thousands of disks
  - Memory, controlling software (sometimes NVRAM, etc)
- Features
  - RAID, hot spares, hot swap
  - Shared storage -> more efficiency
  - Snapshots, clones, thin provisioning, replication, deduplication, etc

RAID Structure

- RAID
  - Organize multiple disk drives as one logical disk
  - Redundant Array of *Independent* Disks
  - (previously, I = *Inexpensive*)
- Two goals:
  - Redundancy: Improves reliability
  - Striping: Improves performance
- Commentary
  - This is more computer architecture, rather than OS topic!

RAID: improving reliability

- Mean time to **failure** of disks:
  - say 100K hours (11.4 years)
  - However, 1 out of 100 disks fail in 100K/100 hours => 41.67 days!
- Solution: mirroring
  - Make exact copy of disk: every write => write same data to multiple disks
  - One disk fails => can get data from other disks
- Mean time to repair
  - average time to (discover and) repair or replace+restore the failed disk
  - => vulnerable, because mirror failure could cause data loss
- Mean time to data loss

Example
Mean time to data loss

- Disk assumptions
  - Mirrored disks fail independently
  - Disk with 100,000 hour mean time to failure
  - Disk has 10 hour mean time to repair
- Mean time to data loss
  - $100{,}000^2 / (2 * 10)$
    $= 500 * 10^6$ hours,
    or 57,000 years!

Mirroring Issues

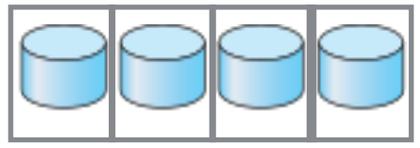
- Mirrored disks not always fail independently
  - power failure => affect both disks
  - disks from same batch => may fail for same defect
  - as disks age => likelihood of failure both increase
- Potential solutions
  - Write to NVRAM or SSD as write-back cache

RAID: Improving Performance by data striping

- Observation: multiple disks => parallelism!
  - Data striping = split data onto multiple disks, access in parallel
- Bit-level striping
  - access different bits on different disks
    e.g., 8 disks: each disk i access bit[i] of a given byte
    => 8x throughput! (though latency about same)
- Block-level striping (more common)
  - access different blocks on different disks

RAID Levels

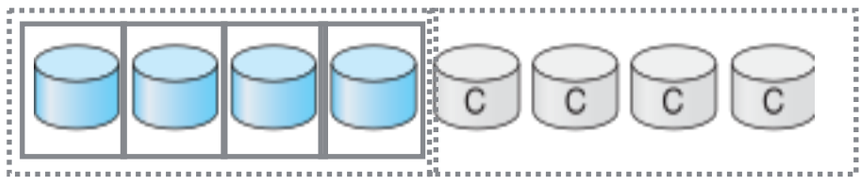
- RAID 0: non-redundant striping



  - performance: minimum 2 disks (or else no parallelism!)
- RAID 1: Mirroring or shadowing

  - redundancy: keeps duplicate of each disk
  - minimum 2 disk (or else no redundancy!)
- RAID 1+0: Striped mirrors

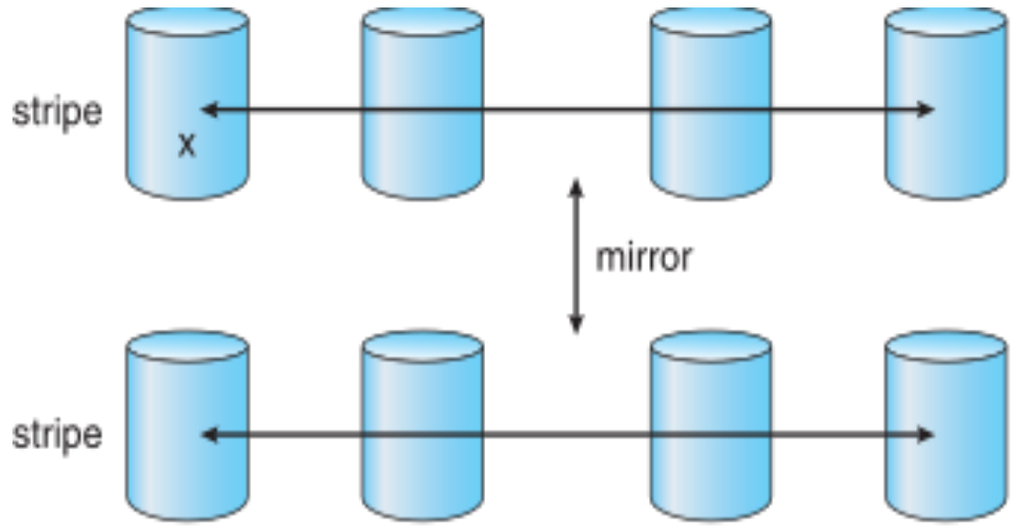
  - better choice than 0+1
- RAID 0+1:mirrored stripes



  - less fault tolerant than 1+0

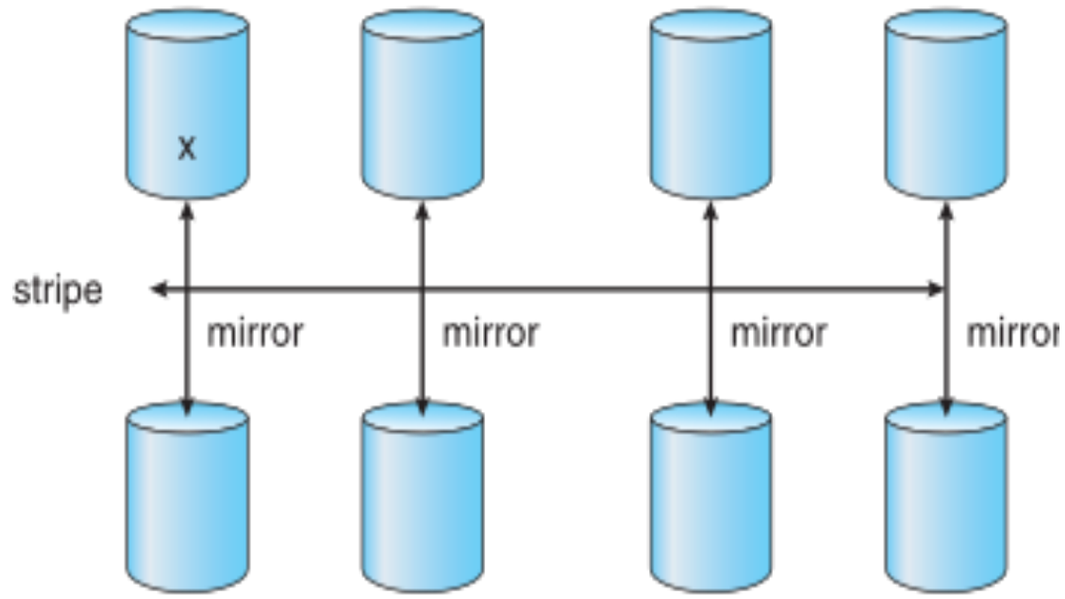
RAID (0 + 1) and (1 + 0)

0+1
mirrored stripes



a) RAID 0 + 1 with a single disk failure.

1+0
striped mirrors



b) RAID 1 + 0 with a single disk failure.

RAID Levels (cont'd)

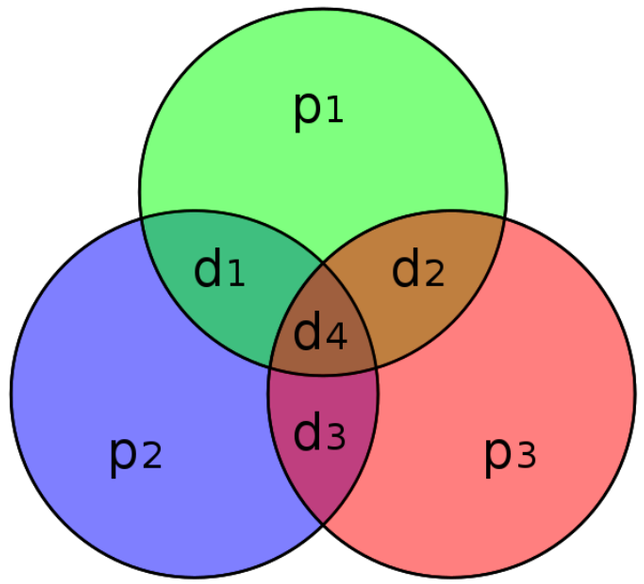
- RAID 2:

  - memory-style error-correcting codes (ECC)
  - bit-level striping w/ dedicated disk for Hamming code
  - saves one disk compared to mirroring
    - not common now, since drives have own ECC
- RAID 3: bit-interleaving parity

  - use parity instead of ECC => save even more disks
  - why? disks have own ECC and knows which disk fails
  - construct failed bit from parity

Hamming [7,4] code

- "payload": 4 data bits d1..d4
- total: 7 bits
- 3 "parity" bits: p1..p3
  - p1 = parity for {d1,d2,d4}
  - p2 = parity for {d1,d3,d4}
  - p3 = parity {d2,d3,d4}
- detect and <u>correct</u> single-bit error



RAID Levels (cont'd) block-level interleaving

- RAID 4:
  - block-interleaving parity on separate disk
  - like RAID 3 but block level instead of bit

- RAID 5:
  - block-interleaved distributed parity on each disk
  - parity bit goes with data, not separate disk

- RAID 6: P+Q Redundancy
  - store more bits to be able to recover from multi-disk failure

Trade-offs of RAID Levels

- Performance and Reliability, but use more space
  - => RAID 0+1 or RAID 1+0
  - provides high performance and high reliability
- Less redundancy
  - Block interleaved parity (RAID 4, 5, 6)
- RAID within a storage array can still fail if the array fails
  - => automatic replication of the data between arrays is common
- hot-spare disks:
  - automatically replacing a failed disk and having data rebuilt onto them

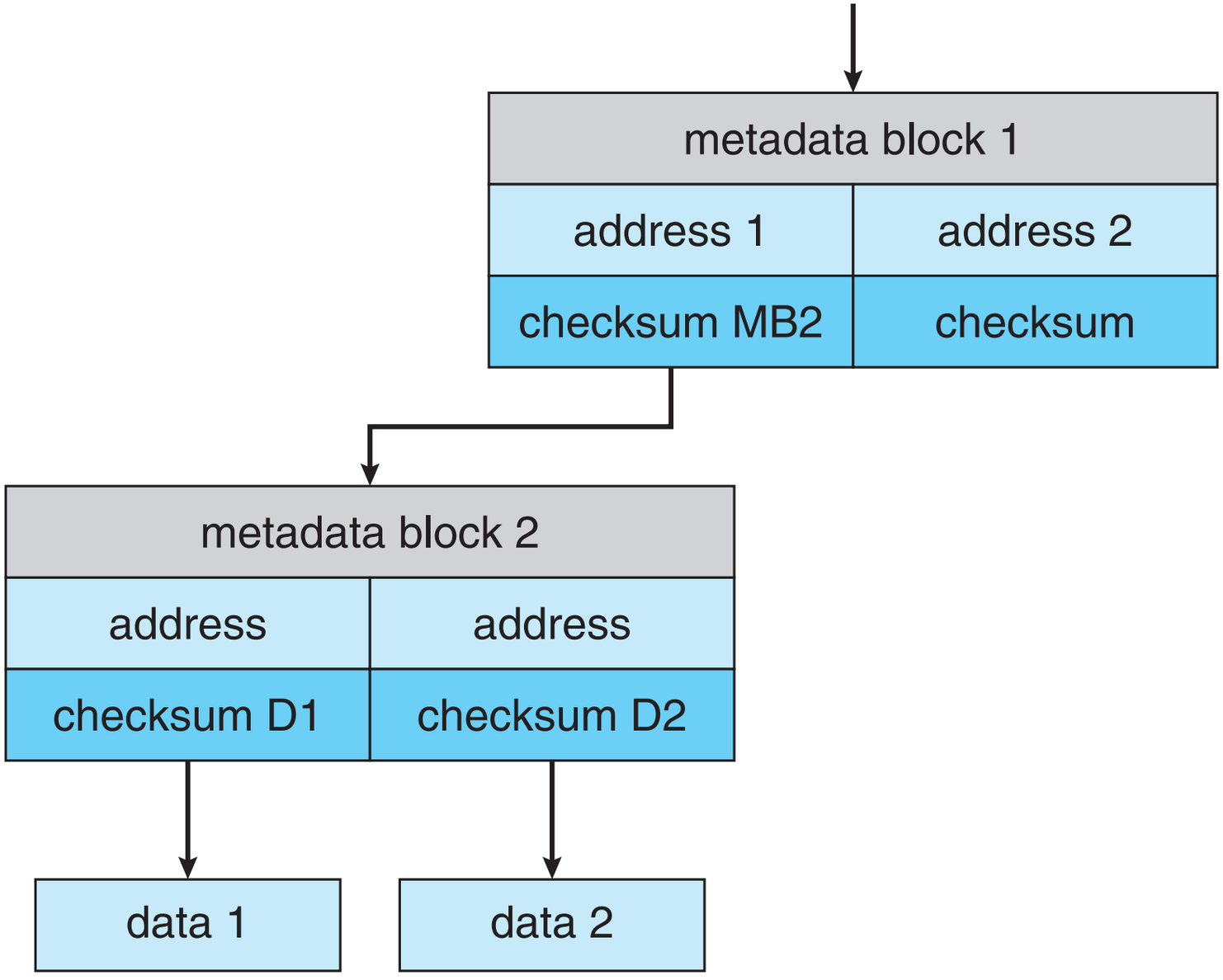
Other Features of Disk Management

- Snapshot
  - a view of file system before a set of changes take place
- Replication
  - automatic duplication of writes between separate sites
    - For redundancy and disaster recovery
    - Can be synchronous or asynchronous
- Hot spare disk
  - Normally unused, automatically used by RAID production if a disk fails
  - replaces the failed disk and rebuild the RAID set if possible
  - Decreases mean time to repair

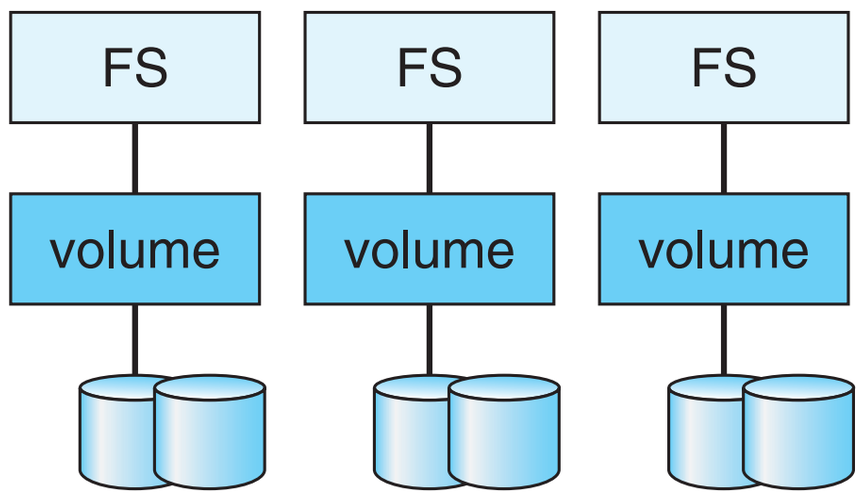
Preventing or detecting data corruption

- handled mainly by OS or FS
  - RAID mainly handles disk failures
- Solaris ZFS adds checksums of all data,metadata
  - Checksums kept with pointer to object
    - to detect if object is the right one and whether it changed
    - Can detect and correct data and metadata corruption
- ZFS also removes volumes, partitions
  - Disks allocated in pools, shared by Filesystems
  - use and release space like malloc() and free()

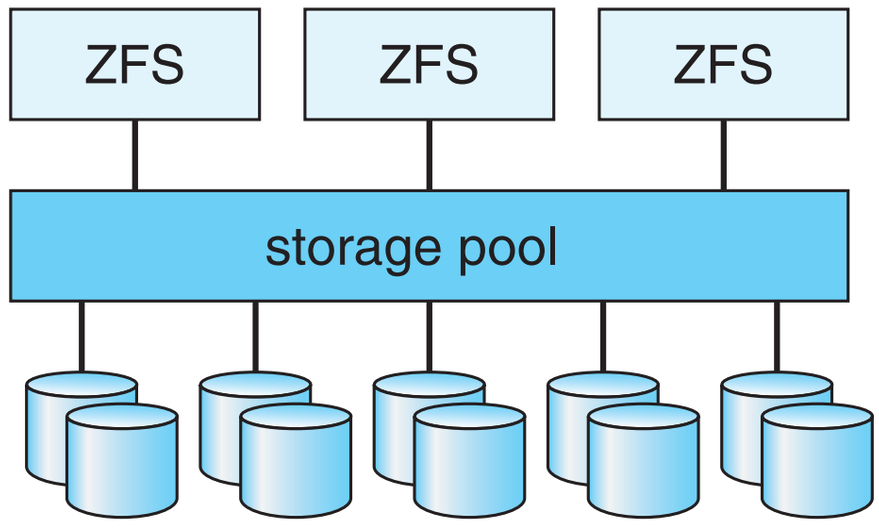
ZFS Checksums All Metadata and Data



Traditional and Pooled Storage



(a) Traditional volumes and file systems.



(b) ZFS and pooled storage.