CHAPTER 11 AND 12 - MASS-STORAGE STRUCTURE & I/O-SYSTEMS

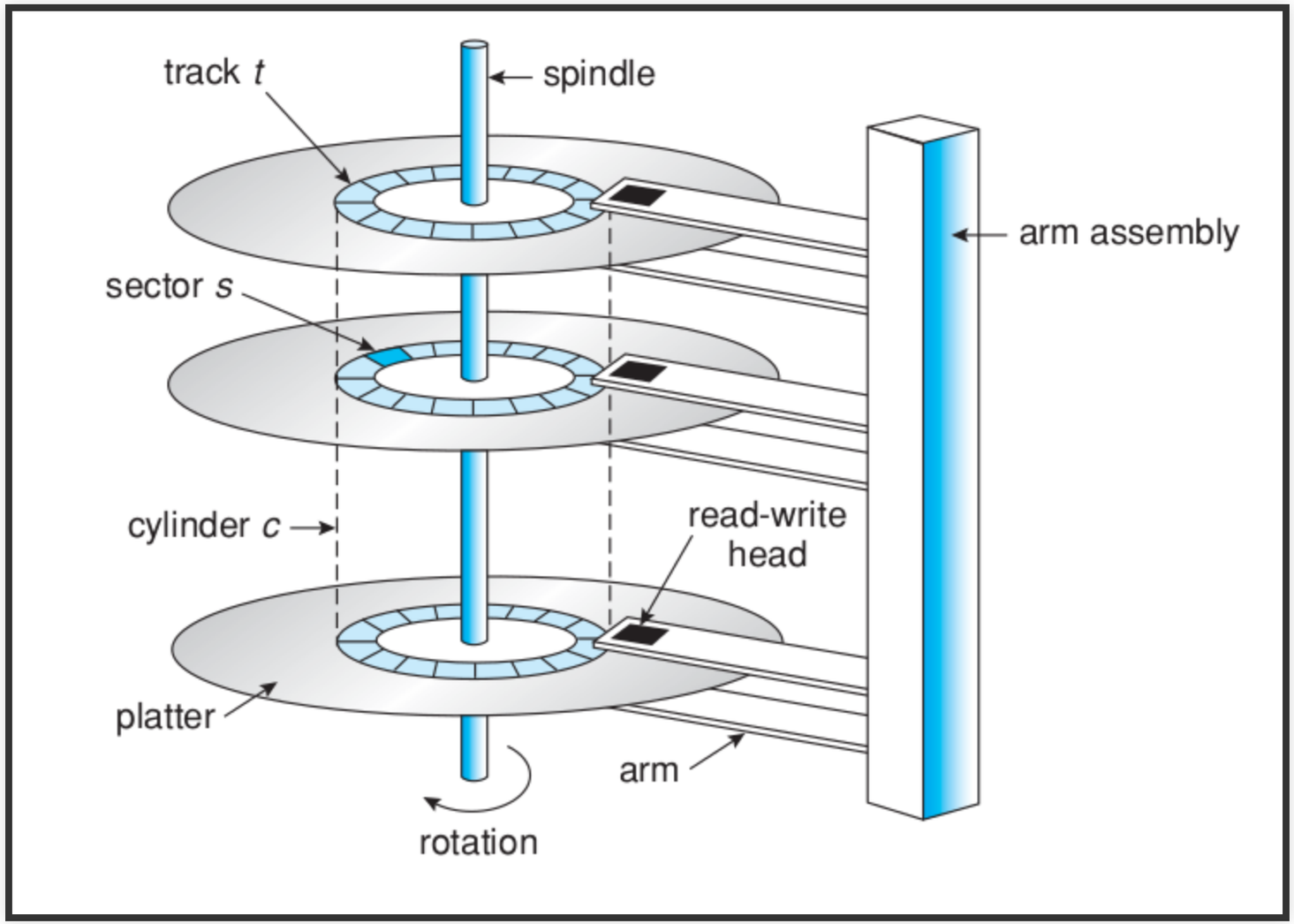
OBJECTIVES

- Describe physical structure of secondary storage devices and its effects on the uses of the devices
- Explain the performance char. of mass-storage devices
- Evaluate I/O scheduling algorithms
- Discuss OS services provided for mass storage, incl. RAID
- Explore the structure of an OS's I/O subsystem
- Discuss principles of I/O hardware and its complexity
- Provide details of the performance aspects of I/O hardware and software

OVERVIEW OF MASS-STORAGE STRUCTURE

- Hard disk drives (HDDs)
- Nonvolatile memory (NVM) devices

HARD DISK DRIVES



HARD DISK DRIVES

- Magnetic disks (still) provide bulk of secondary storage of modern computers
  - Drives rotate at 60 to 250 times per second (RPM)
  - **Transfer rate** is rate at which data flow between drive and computer
  - Positioning time (random-access time) is both
    - **seek time**: time to move disk arm to desiredcylinder
    - **rotational latency** time for desired sector to rotate under the disk head

HARD DISK DRIVES

- Head flies just above disc (microns)
- Head crash results from disk head making contact with the disk surface
  - That's bad
- Some storage media can be removable (disk, CD, DVD)
- Drive attached to computer via I/O bus
  - Busses vary, including EIDE, ATA, SATA, USB, Fibre Channel, SCSI, SAS, Firewire
  - **Host controller** in computer uses bus to talk to **disk controller** built into drive or storage array

MAGNETIC DISKS

- Platters range from .85” to 14” (historically)
  - Commonly 3.5”, 2.5”, and 1.8”
- Range from 30GB to 3TB per drive

MAGNETIC DISKS - PERFORMANCE

- Performance
  - Transfer Rate – theoretical – 6 Gb/sec
  - Effective Transfer Rate – real – 1Gb/sec
  - Seek time from 3ms to 12ms
  - Average seek time measured or calculated based on 1/3 of tracks
  - Latency based on spindle speed

THE FIRST COMMERCIAL DISK DRIVE

NONVOLATILE MEMORY DEVICES

- Electrical instead of mechanical
- Growing in importance
- Controller and flash NAND die semiconductor chips
- DRAM with battery backup
- Flash-memory-based NVM used in a disk-drive-like container ⇒ solid-state disk (SSD)

SOLID-STATE DISKS

- Nonvolatile memory used like a hard drive
  - Many technology variations
- Can be more reliable than HDDs
- More expensive per MB
- Uses less power
- Maybe have shorter life span?
- Less capacity → But much faster
- No moving parts, so no seek time or rotational latency

NAND SEMICONDUCTOR CHARACTERISTICS

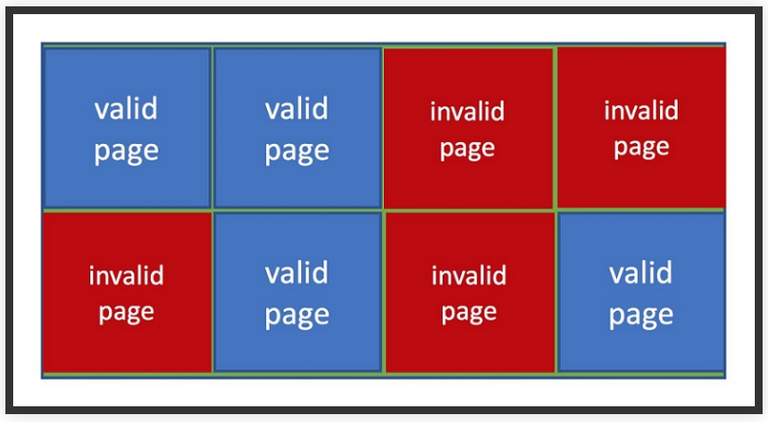
- Can be read and written in a “page” increment (similar to a sector),
- Data cannot be overwritten — rather, the NAND cells have to be erased first.
- Erasure, occurs in a “block” increment that is several pages in size
  - Takes much more time than a read (the fastest operation) or a write (slower than read, but much faster than erase).

LIFESPAN

- NAND semiconductors also deteriorate with every erase cycle
  - After approximately 100,000 program-erase cycles they stop retaining data
- NAND NVM lifespan is not measured in years but **Drive Writes Per Day (DWPD)**
  - Measure how many times the drive capacity can be written per day before the drive fails.
- Example, a 1 TB NAND drive with a 5 DWPD rating is expected to have 5 TB per day written to it for the warranty period without failure.

NAND FLASH CONTROLLER ALGORITHMS

NAND FLASH CONTROLLER ALGORITHMS



NAND FLASH CONTROLLER ALGORITHMS

Consider a full SSD with a pending write request - some individual pages could contain invalid data.

Where would garbage collection store valid data?

NAND FLASH CONTROLLER ALGORITHMS

To solve this problem and improve write performance, the NVM device uses **over-provisioning**.

Device sets aside a number of pages (frequently 20 percent of the total) as an area always available to write to.

Over-provisioning space can also help with **wear leveling**: Controller tries to avoid that by using various algorithms to place data on less-erased blocks

VOLATILE MEMORY

- DRAM is frequently used as a mass-storage device.
- RAM drives (which are known by many names, including RAM disks) act like secondary storage
  - Created by device drivers that carve out a section of the system's DRAM and present it to the rest of the system as it if were a storage device.

MAGNETIC TAPE

- Was early secondary-storage medium
  - Evolved from open spools to cartridges
- Relatively permanent and holds large quantities of data
- Access time slow
- Random access ~1000 times slower than disk

MAGNETIC TAPE

- Mainly used for backup, storage of infrequently-used data, transfer medium between systems
- Kept in spool and wound or rewound past read-write head
- Once data under head, transfer rates comparable to disk
  - 140MB/sec and greater
- 200GB to 1.5TB typical storage

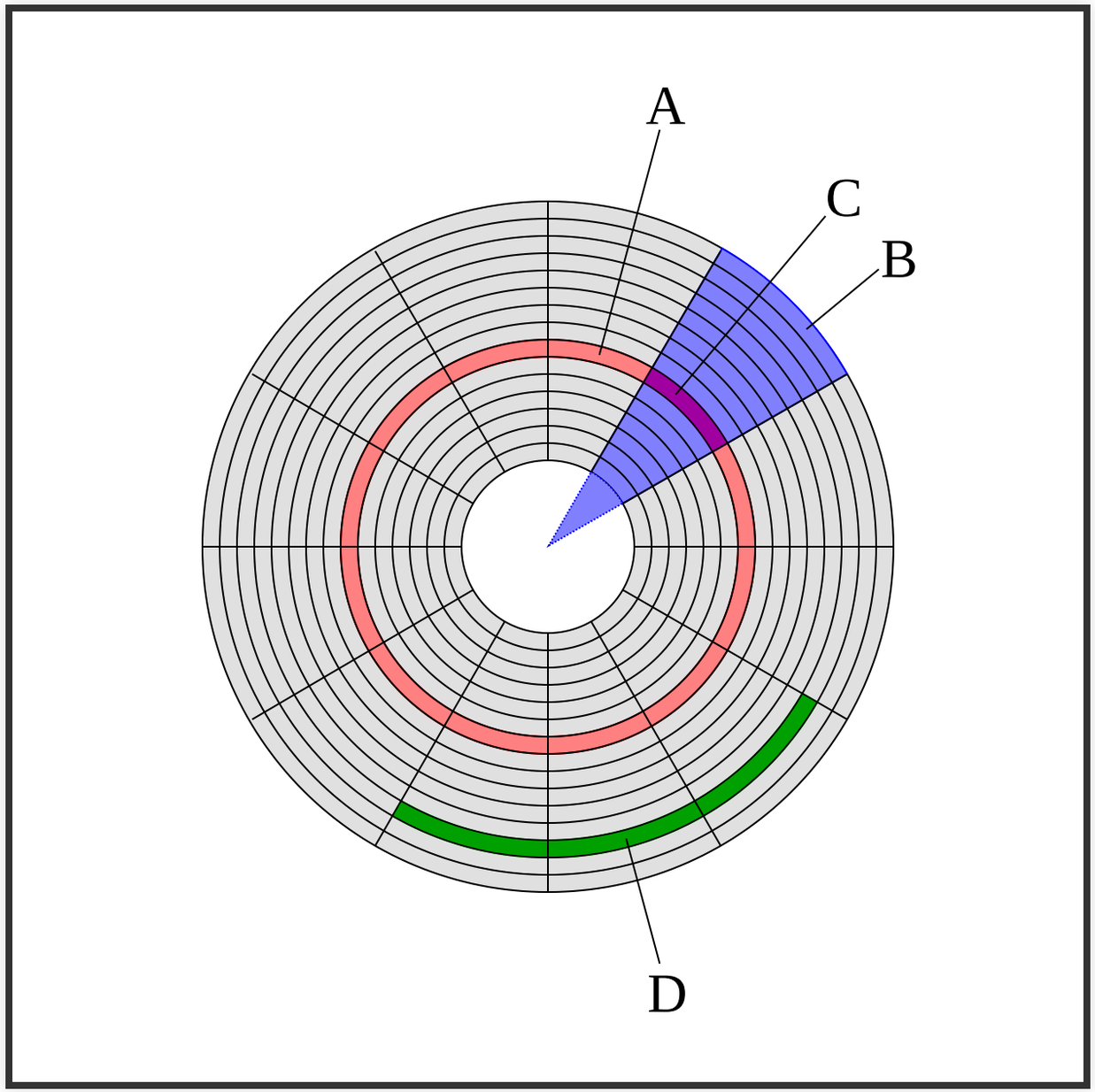
ADDRESS MAPPING AND DISK STRUCTURE

- Disk drives are addressed as large 1-dimensional arrays of logical blocks, where the logical block is the smallest unit of transfer
  - Low-level formatting creates logical blocks on physical media

ADDRESS MAPPING AND DISK STRUCTURE

- The 1-dimensional array of logical blocks is mapped into 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

DISK STRUCTURE



DISK STRUCTURE

- Logical to physical address should be easy
  - Except for bad sectors
  - Non-constant # of sectors per track via constant angular velocity

HDD SCHEDULING

DISK SCHEDULING

- The operating system is responsible for using hardware efficiently — for the disk drives, this means having a fast access time and disk bandwidth
- Minimize seek time
- Seek time ≈ seek distance
- 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
- We can improve access time and the bandwidth by managing the order in which storage I/O requests are serviced

DISK SCHEDULING

- I/O request includes
  - input or output mode
  - File handle
  - Memory address
  - Amount of data to transfer

Absolute knowledge of head location and physical block/cylinder locations is generally not possible on modern drives

DISK SCHEDULING

The current goals of disk scheduling include fairness, timeliness, and optimizations, such as bunching reads or writes that appear in sequence, as drives perform best with sequential I/O.

DISK SCHEDULING

- 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 only make sense when a queue exists
- Drive controllers have small buffers and can manage a queue of I/O requests (of varying “depth”)
- Several algorithms exist to schedule the servicing of disk I/O requests
- The analysis is true for one or many platters

DISK SCHEDULING

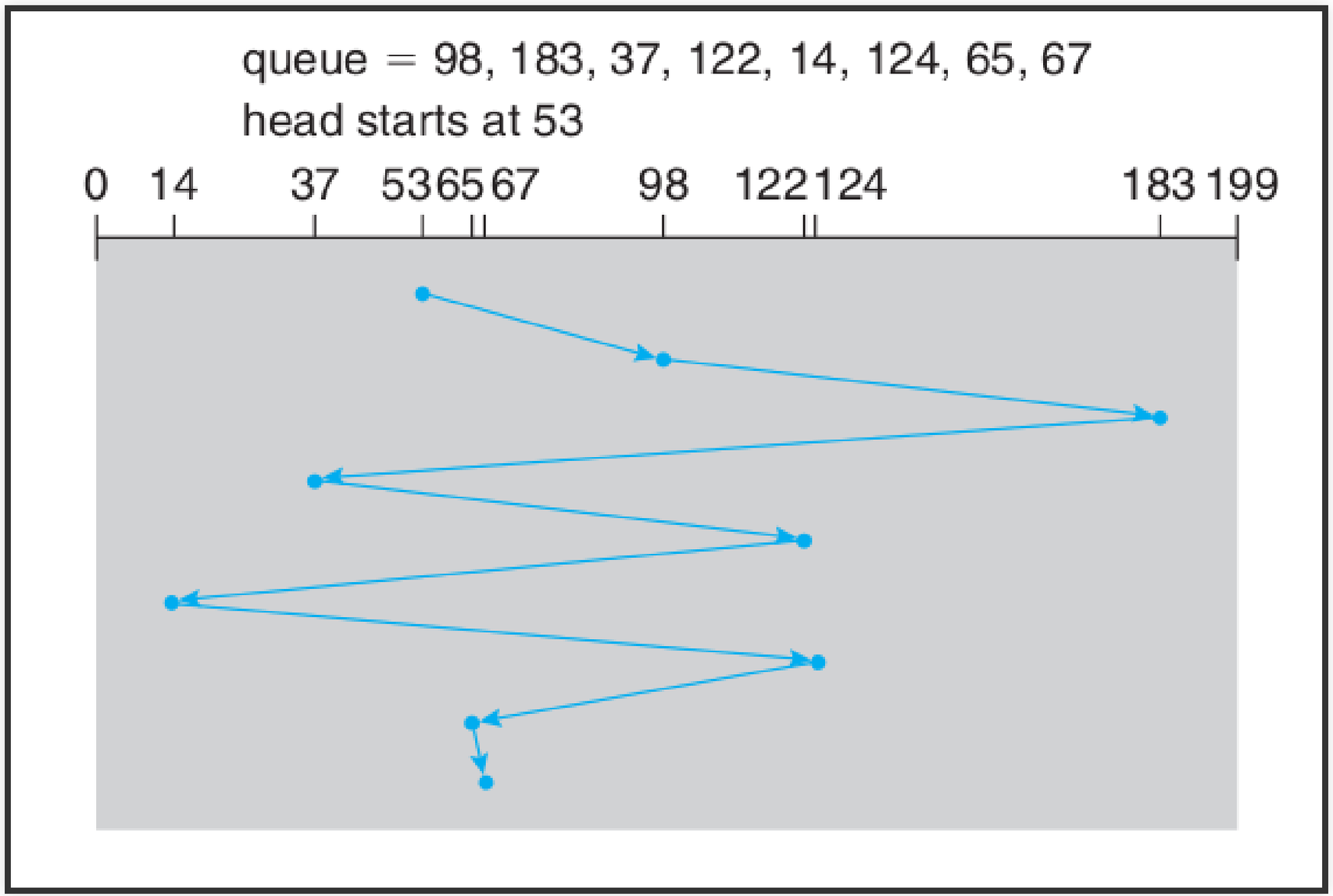
- We illustrate scheduling algorithms with a request queue (0-199)

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

Head pointer 53

FCFS

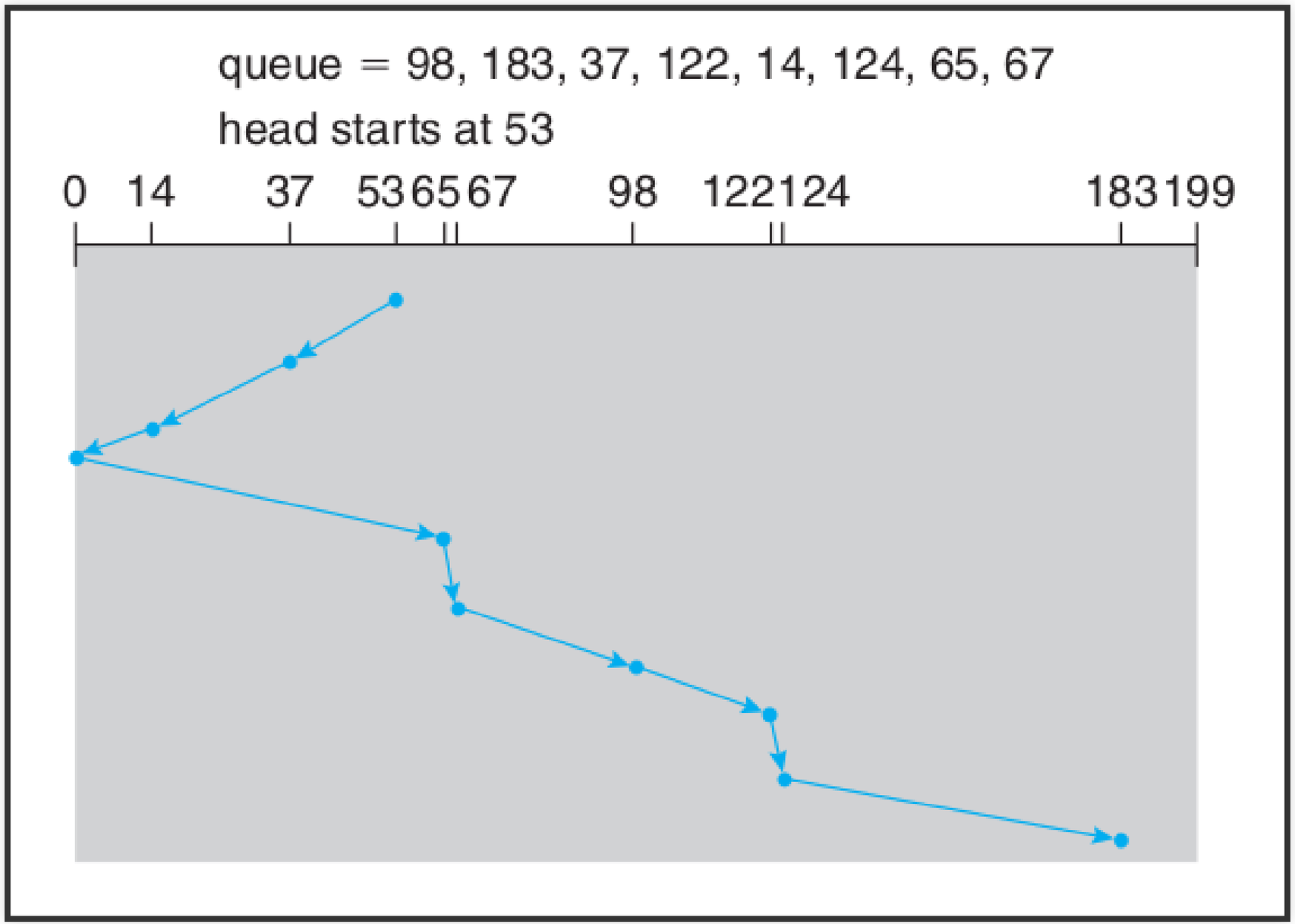
Illustration shows total head movement of 640 cylinders



SCAN

- 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.
- SCAN algorithm Sometimes called the elevator algorithm
- Illustration shows total head movement of 208 cylinders
- But note that if requests are uniformly dense, largest density at other end of disk and those wait the longest

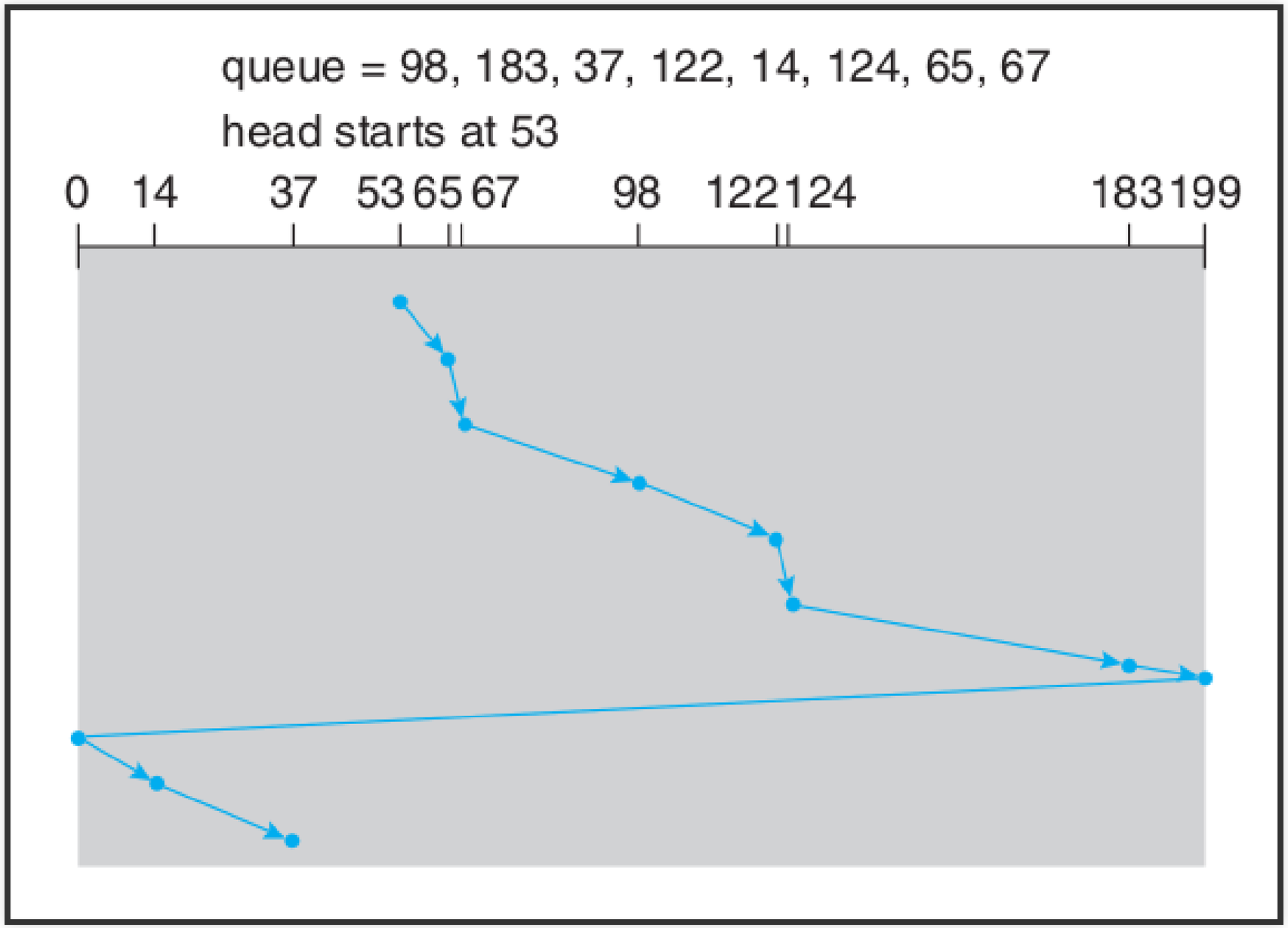
SCAN


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

C-SCAN

- Provides a more uniform wait time than SCAN
- 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 cylinder to the first one
- Total number of cylinders?

C-SCAN


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

SELECTING AN ALGORITHM

- SCAN and C-SCAN perform better for systems that place a heavy load on the disk
  - Less starvation
- Performance depends on the number and types of requests
- Requests for disk service can be influenced by the file-allocation method
  - And metadata layout

SELECTING AN ALGORITHM

- 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
- What about rotational latency?
  - Difficult for OS to calculate
- How does disk-based queuing effect OS queue ordering efforts?

NVM SCHEDULING

⚠ Disk-scheduling algorithms just discussed apply to mechanical platter-based storage like HDDs.

Their focus: Minimizing disk head movement

NVM SCHEDULING

NVM devices do not contain moving disk heads and commonly use a simple FCFS policy.

Linux NOOP scheduler uses an FCFS policy but modifies it to merge adjacent requests.

Random access I/O is much faster on NVM.

IMPACT OF GARBAGE COLLECTION

Consider an NVM device under random read and write load.

IMPACT OF GARBAGE COLLECTION

⚠ one write request eventually causes a page write (the data), one or more page reads (by garbage collection), and one or more page writes (of good data from the garbage-collected blocks) ⇒ **write amplification**

ERROR DETECTION AND CORRECTION

Frequent in memory, networking, and storage.

ERROR DETECTION

- Memory systems have long detected certain errors by using parity bits.
- For each byte store extra bit
- Single bit error detected - double might go undetected
- In networking: Cyclic Redundancy Check

ERROR CORRECTION

- An error-correction code (ECC) not only detects the problem, but also corrects it.
  - Uses extra storage and algorithm
  - disks drives use per-sector ECC and flash drives per-page ECC.
- If correctable error occured: report *soft error* and fix
- If too much is changed and nonrecoverable: *hard error*

STORAGE DEVICE MANAGEMENT

DISK FORMATTING

- Low-level formatting, or physical formatting — Dividing a disk into sectors that the disk controller can read and write
  - Each sector can hold header information, plus data, plus error correction code (ECC)
  - Usually 512 bytes or 4 Kb of data but can be selectable

DISK FORMATTING

- To use a disk to hold files, the operating system still needs to record its own data structures on the disk

  1. **Partition** the disk into one or more groups of cylinders, each treated as a logical disk
  2. **Volume** creation and management.
  3. **Logical formatting** or “making a file system”

Mounting a file system is making the file system available for use by the system and its users.

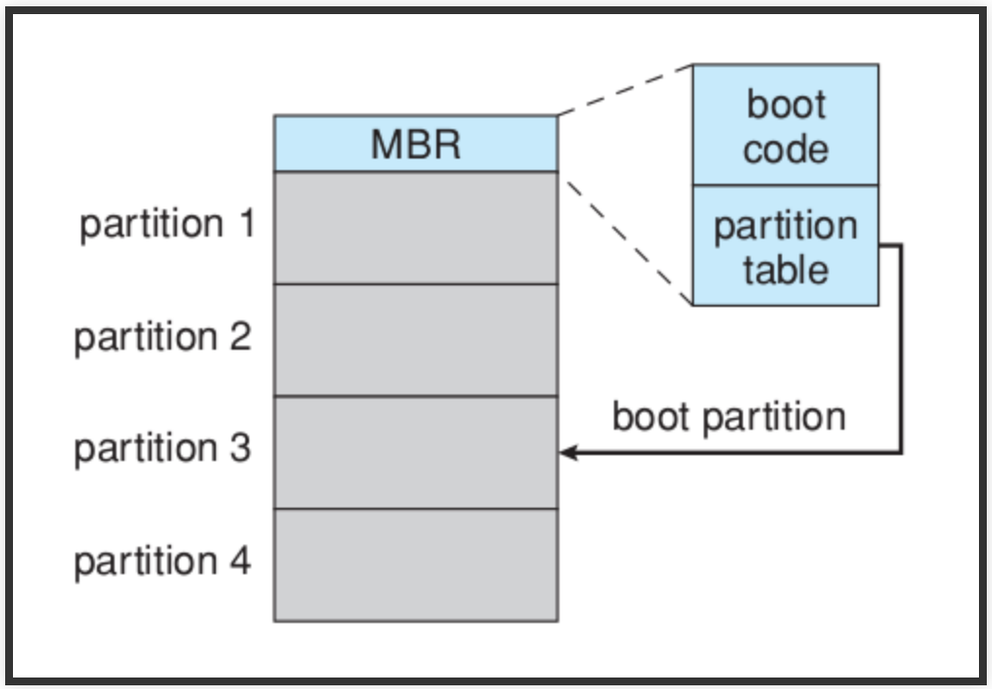
DISK FORMATTING

- To increase efficiency most file systems group blocks into clusters
  - Disk I/O done in blocks
  - File I/O done in clusters
- Raw disk access for apps that want to do their own block management, keep OS out of the way
  - Databases for example

BOOT BLOCK

- Boot block initializes system
  - The bootstrap is stored in ROM
  - Bootstrap loader program stored in boot blocks of boot partition

BOOTING FROM A DISK IN WINDOWS



BAD BLOCKS

- Methods such as **sector sparing** used to handle bad blocks
- Example
  - OS tries to read logical block 87
  - Controller calculates ECC → finds sector is bad
    - It reports to the operating system as an I/O error.
  - Maintains bad blocks list
  - Next boot → command run to replace sector wih spare
  - Next read to 87 → controller finds spare sector

BAD BLOCKS NVM DEVICES

Controller maintains a table of bad pages and never sets those pages as available to write to, so they are never accessed.

SWAP-SPACE MANAGEMENT

SWAP-SPACE MANAGEMENT

- Swap-space — Virtual memory uses disk space as an extension of main memory
  - Less common now due to memory capacity increases
- Swap-space can be carved out of the normal file system, or, more commonly, it can be in a separate disk partition (raw)

SWAP-SPACE MANAGEMENT

- 4.3BSD allocates swap space when process starts; holds text segment (the program) and data segment
- Kernel uses swap maps to track swap-space use

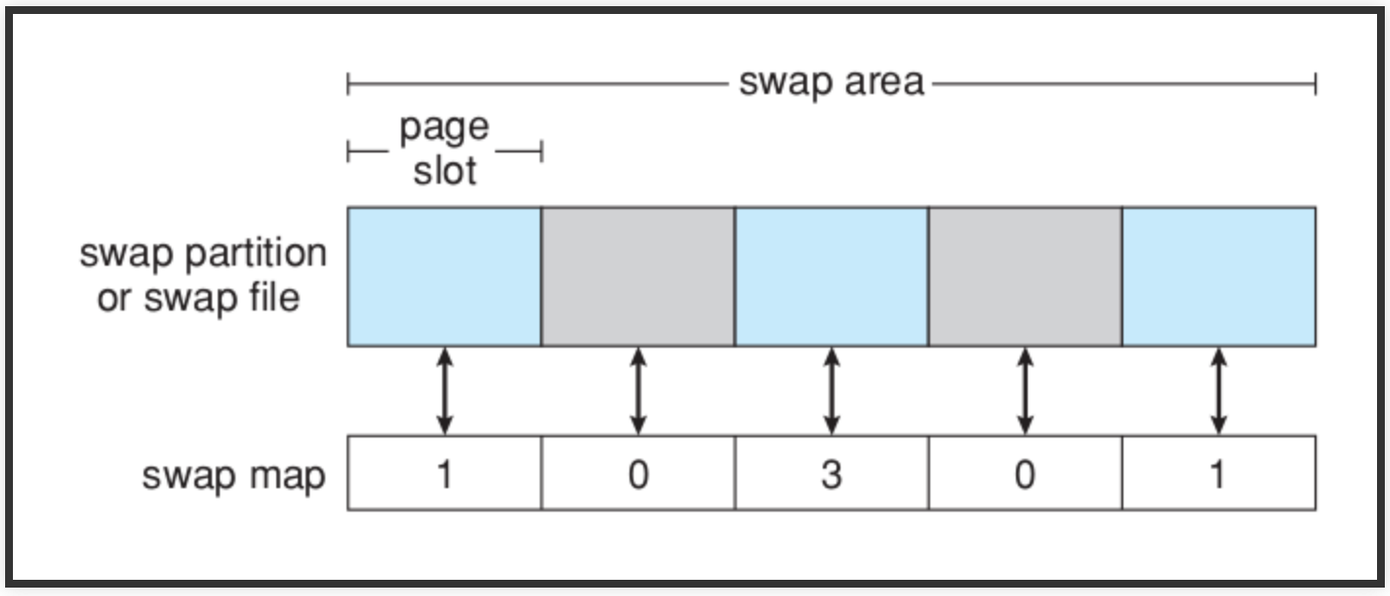
SWAP-SPACE MANAGEMENT

- Solaris 2 allocates swap space only when a dirty page is forced out of physical memory, not when the virtual memory page is first created
  - File data written to swap space until write to file system requested
  - Other dirty pages go to swap space due to no other home
  - Text segment pages thrown out and reread from the file system as needed

SWAP-SPACE MANAGEMENT

- What if a system runs out of swap space?
- Some systems allow multiple swap spaces

DATA STRUCTURES FOR SWAPPING ON LINUX SYSTEMS



STORAGE ATTACHMENT

DISK ATTACHMENT

- **Host-attached storage** accessed through I/O ports talking to I/O busses
- SCSI itself is a bus, up to 16 devices on one cable, SCSI initiator requests operation and SCSI targets perform tasks
  - Each target can have up to 8 logical units (disks attached to device controller)

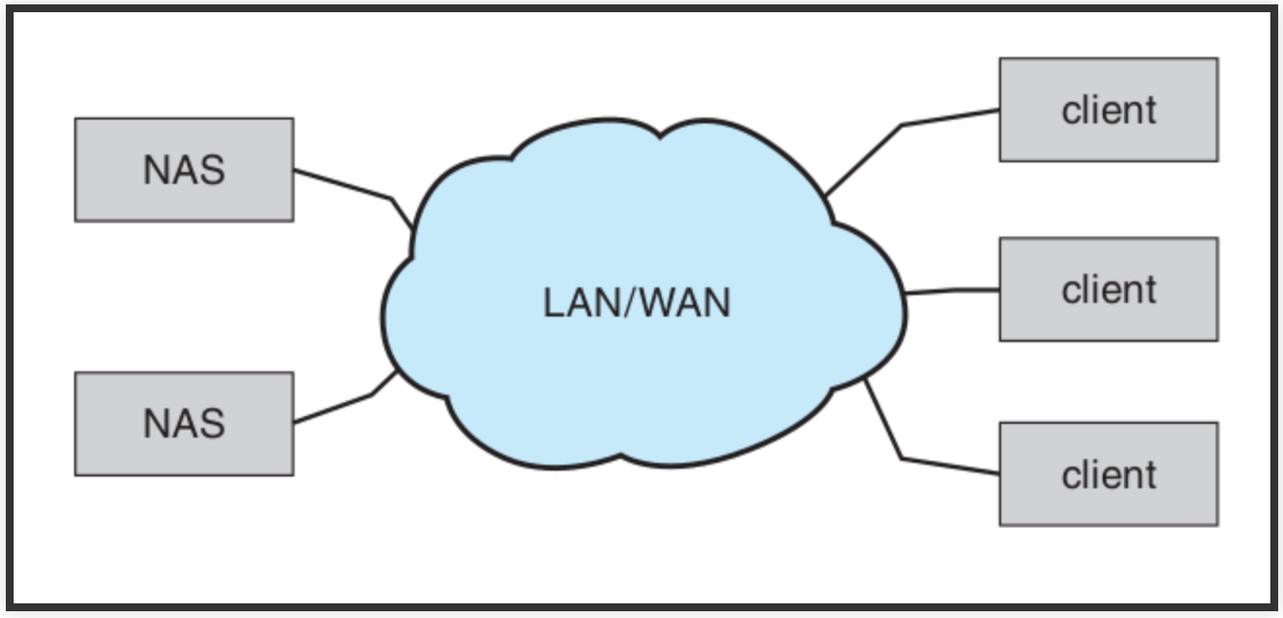
DISK ATTACHMENT

- FC is high-speed serial architecture
  - Can be switched fabric with 24-bit address space – the basis of storage area networks (SANs) in which many hosts attach to many storage units
- I/O directed to bus ID, device ID, logical unit (LUN)

NETWORK-ATTACHED STORAGE

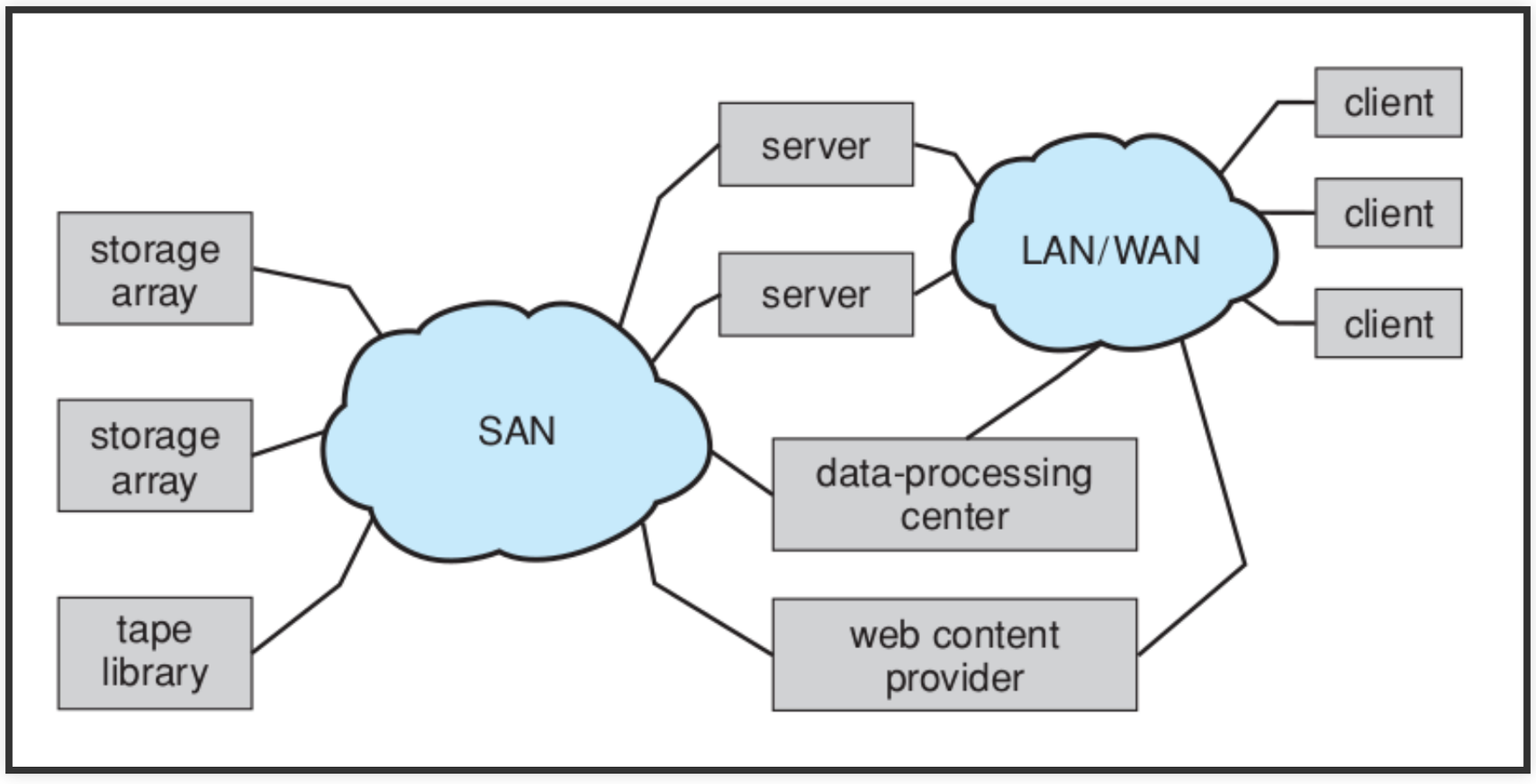
- Network-attached storage (NAS) is storage made available over a network rather than over a local connection (such as a bus)
  - Remotely attaching to file systems
- NFS and CIFS are common protocols
- Implemented via remote procedure calls (RPCs) between host and storage over typically TCP or UDP on IP network
- iSCSI protocol uses IP network to carry the SCSI protocol
  - Remotely attaching to devices (blocks)

NETWORK-ATTACHED STORAGE



STORAGE AREA NETWORK

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



STORAGE ARRAY

- Can just attach disks, or arrays of disks
- Storage Array has controller(s), provides features to attached host(s)
  - Ports to connect hosts to array
  - Memory, controlling software (sometimes NVRAM, etc)
  - A few to thousands of disks
  - RAID, hot spares, hot swap (discussed later)
  - Shared storage → more efficiency

STORAGE AREA NETWORK

- SAN is one or more storage arrays
  - Connected to one or more Fibre Channel switches
- Hosts also attach to the switches
- Storage made available via LUN Masking from specific arrays to specific servers
- Easy to add or remove storage, add new host and allocate it storage
  - Over low-latency Fibre Channel fabric

CLOUD STORAGE

Similar to network-attached storage, cloud storage provides access to storage across a network.

Unlike NAS , the storage is accessed over the Internet or another WAN to a remote data center

Access is API based

RAID STRUCTURE

RAID STRUCTURE

- RAID – redundant array of independent disks
  - multiple disk drives provides reliability via redundancy
- Increases the mean time to failure
- Mean time to repair – exposure time when another failure could cause data loss
- Mean time to data loss based on above factors

RAID RELIABILITY

- 100 disks with mean time to failure 100,000 hours
  - 100,000/100 = 1000 hours → 41.66 days → Not very long
- If only one copy of data stored → dataloss frequently
- **Mirroring** Duplicate disk completely
  - Still have inconsistent data if power failure occurs while writing

RAID STRUCTURE

- If mirrored disks fail independently, consider disk with 100,000 hours mean time to failure and 10 hour mean time to repair
  - Mean time to data loss is $100{,}000^2 / (2 * 10) = 500 * 10^6$ hours, or 57,000 years!
- Frequently combined with NVRAM to improve write performance
- RAID is arranged into six different levels

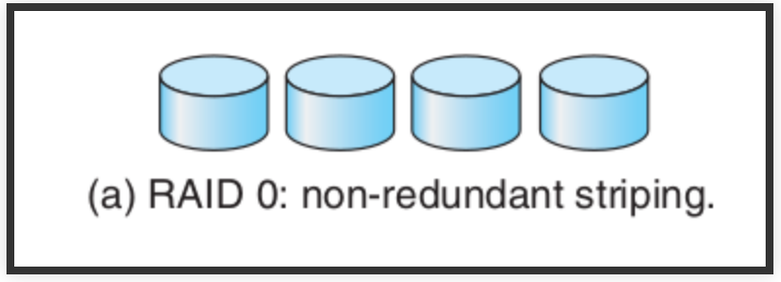
RAID STRUCTURE

- Several improvements in disk-use techniques involve the use of multiple disks working cooperatively
- Disk striping uses a group of disks as one storage unit
- RAID schemes improve performance and improve the reliability of the storage system by storing redundant data

STRIPING

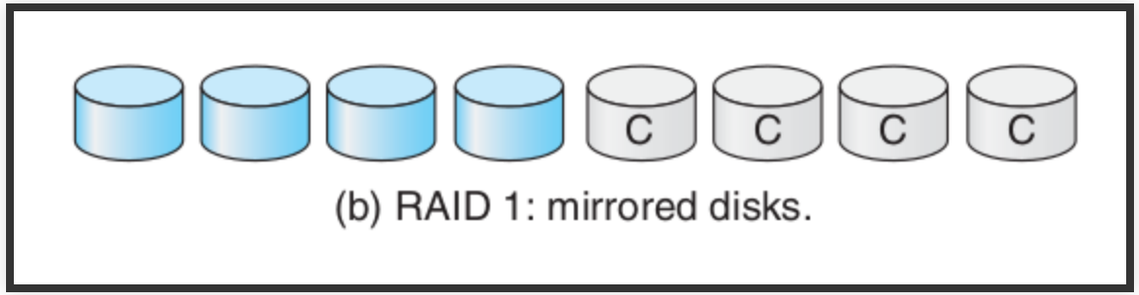
- **Data striping** splitting the data accross multiple discs
- **Bit-level striping** Divede bits of each byte across discs
- **Block-level striping** Blocks from a file divided across discs
  - Most common
- Increases parallelism
  - Increases throughput
  - Reduce response times

RAID LEVEL 0



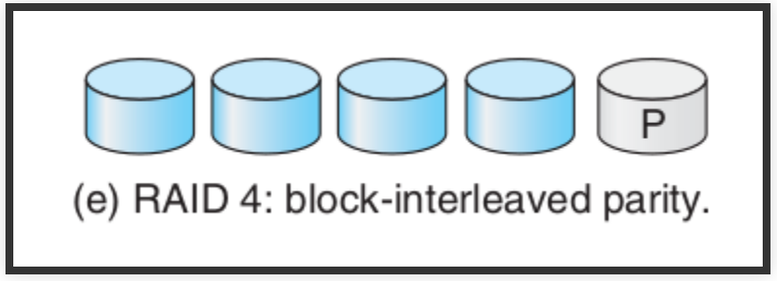
(a) RAID 0: non-redundant striping.

RAID LEVEL 1



(b) RAID 1: mirrored disks.

RAID LEVEL 4



(e) RAID 4: block-interleaved parity.

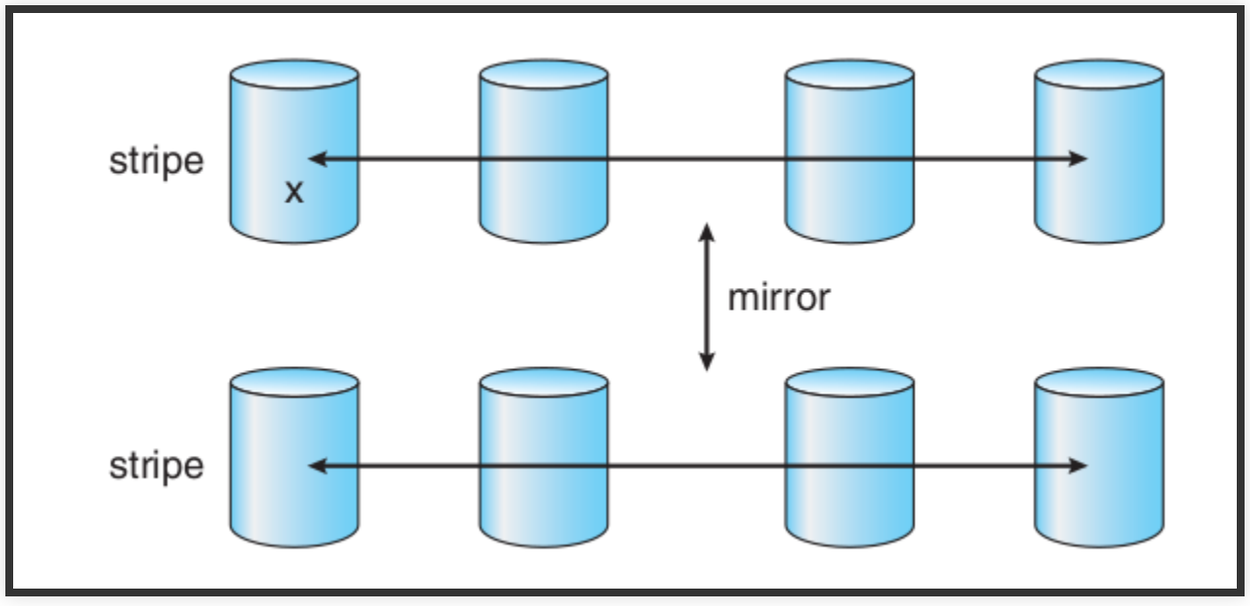
RAID LEVEL 5

(f) RAID 5: block-interleaved distributed parity.

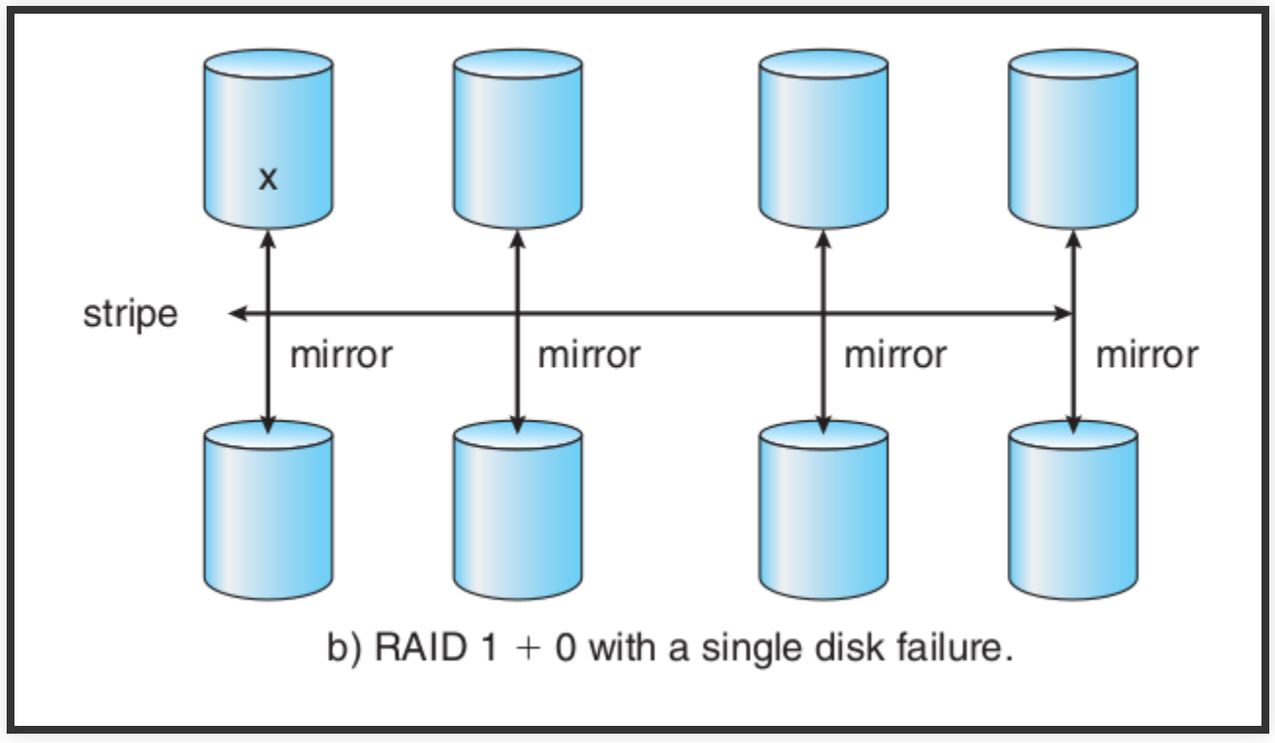
RAID LEVEL 6

(g) RAID 6: P + Q redundancy.

RAID (0 + 1)



RAID (1 + 0)



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

OTHER FEATURES

- Regardless of where RAID implemented, other useful features can be added
- Snapshot is a view of file system before a set of changes take place (i.e. at a point in time)
- Replication is automatic duplication of writes between separate sites
  - For redundancy and disaster recovery
  - Can be synchronous or asynchronous

OTHER FEATURES

- Hot spare disk is unused, automatically used by RAID production if a disk fails to replace the failed disk and rebuild the RAID set if possible
  - Decreases mean time to repair

PROBLEMS WITH RAID

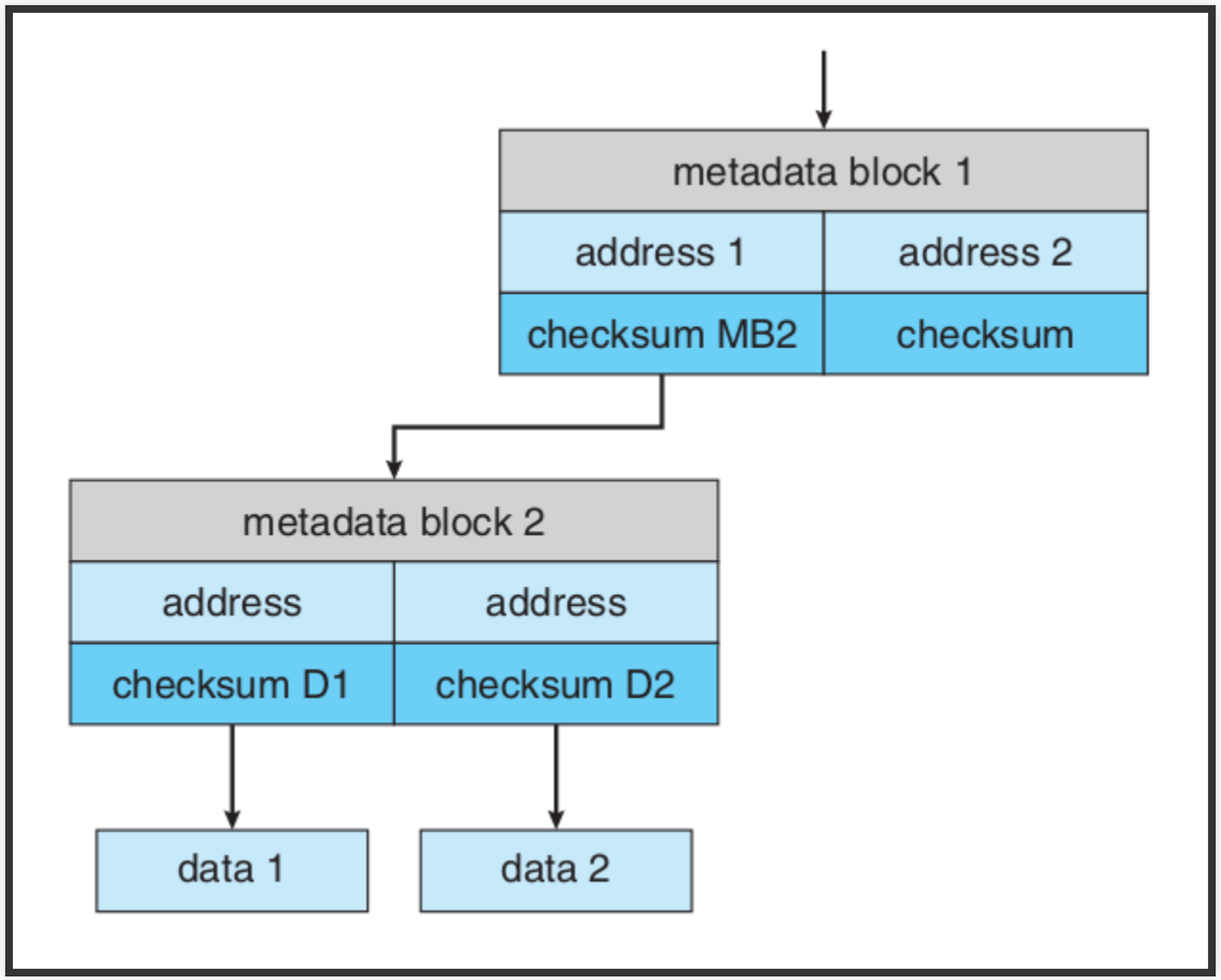
RAID protects against physical media errors, but not other hardware and software errors.

A failure of the hardware RAID controller, or a bug in the software RAID code, could result in total data loss.

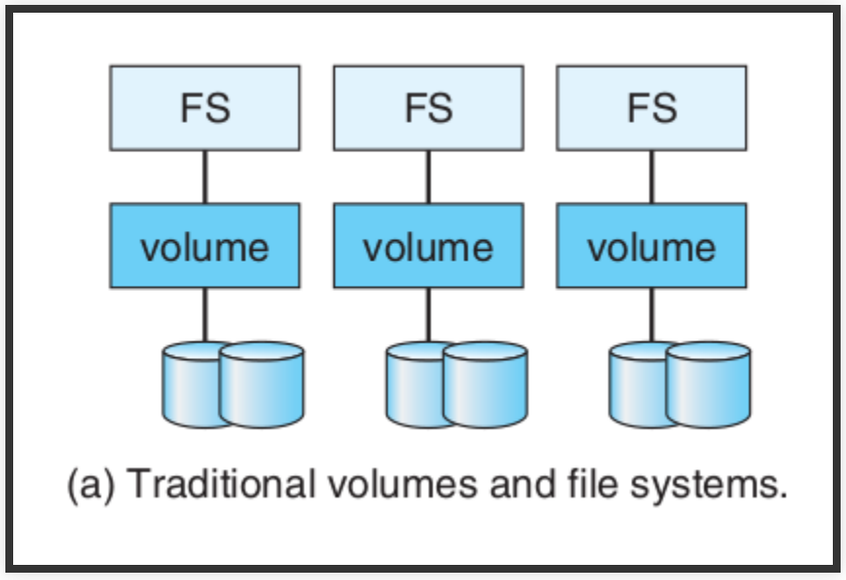
EXTENSIONS

- RAID alone does not prevent or detect data corruption or other errors, just disk failures
- Solaris ZFS adds checksums of all data and 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 CHECKSUMS ALL METADATA AND DATA

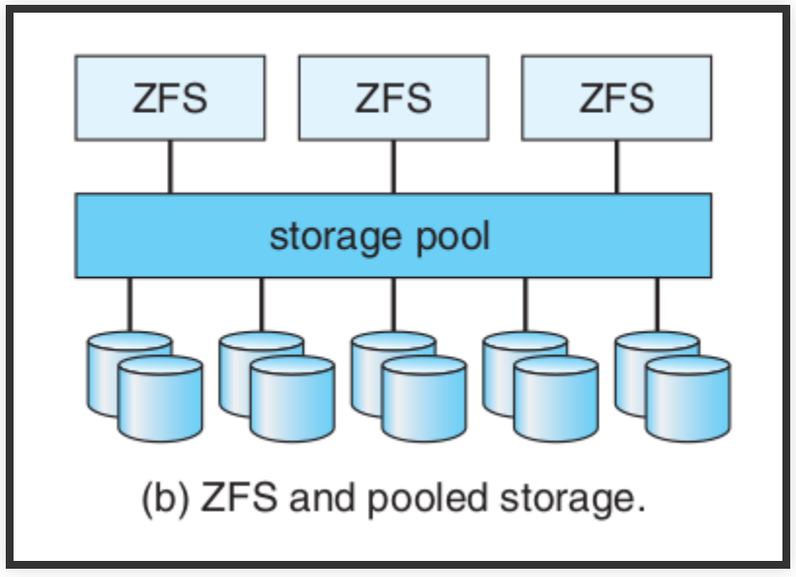


TRADITIONAL STORAGE



(a) Traditional volumes and file systems.

POOLED STORAGE



(b) ZFS and pooled storage.

STABLE-STORAGE IMPLEMENTATION

STABLE-STORAGE IMPLEMENTATION

- Write-ahead log scheme requires stable storage
- Stable storage means data is never lost (due to failure, etc)
- To implement stable storage:
  - Replicate information on more than one nonvolatile storage media with independent failure modes
  - Update information in a controlled manner to ensure that we can recover the stable data after any failure during data transfer or recovery

STABLE-STORAGE IMPLEMENTATION

- Disk write has 1 of 3 outcomes
  1. Successful completion - The data were written correctly on disk
  2. Partial failure - A failure occurred in the midst of transfer, so only some of the sectors were written with the new data, and the sector being written during the failure may have been corrupted
  3. Total failure - The failure occurred before the disk write started, so the previous data values on the disk remain intact

STABLE-STORAGE IMPLEMENTATION

- If failure occurs during block write, recovery procedure restores block to consistent state
  - System maintains 2 physical blocks per logical block and does the following:
    1. Write to 1st physical
    2. When successful, write to 2nd physical
    3. Declare complete only after second write completes successfully
  - Systems frequently use NVRAM as one physical to accelerate

I/O SYSTEMS OVERVIEW

I/O SYSTEMS OVERVIEW

- I/O management is a major component of operating system design and operation
  - Important aspect of computer operation
  - I/O devices vary greatly
  - Various methods to control them
  - Performance management
  - New types of devices frequent

I/O SYSTEMS OVERVIEW

- Ports, busses, device controllers connect to various devices
- Device drivers encapsulate device details
  - Present uniform device-access interface to I/O subsystem

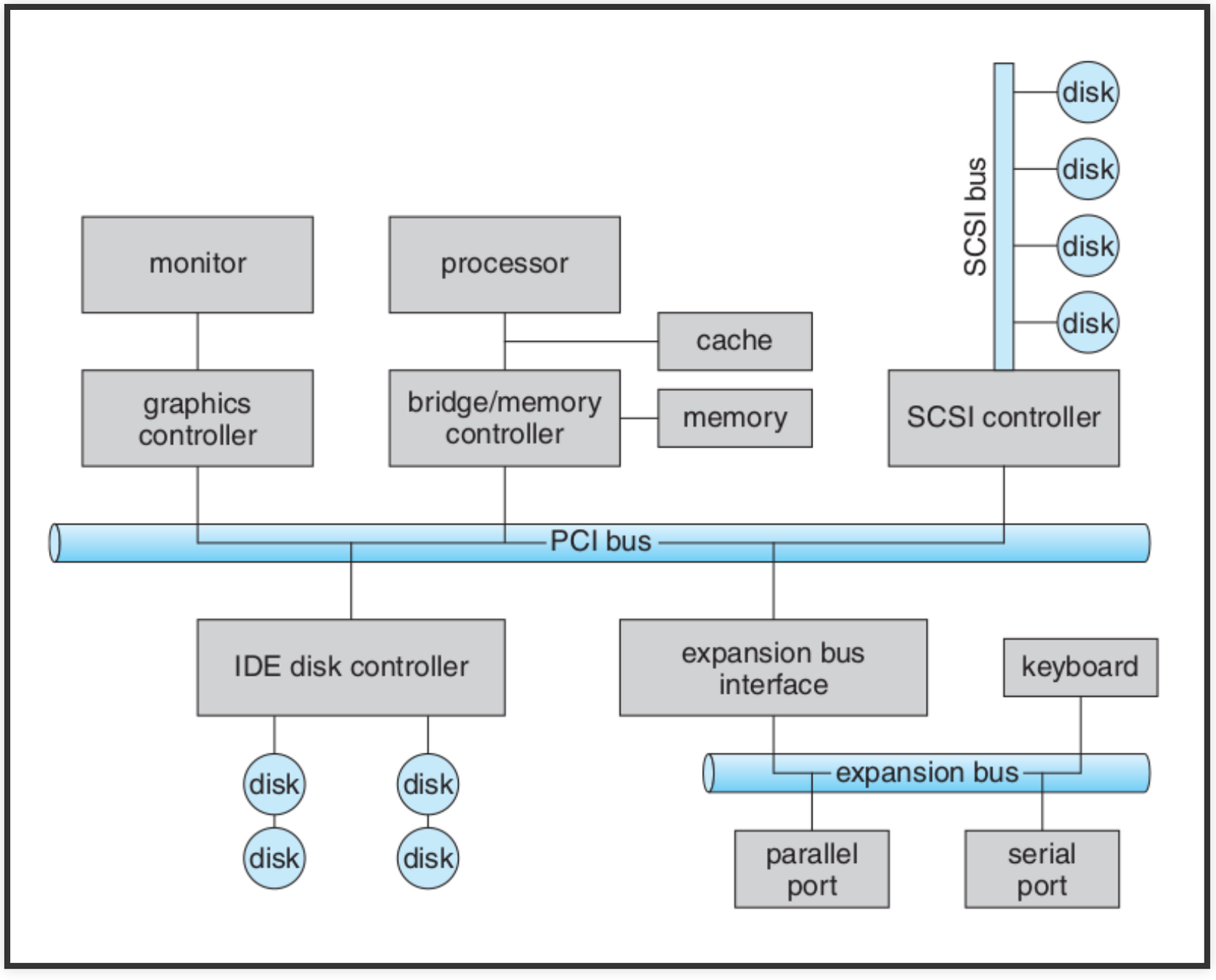
I/O HARDWARE

I/O HARDWARE

- Incredible variety of I/O devices
  - Storage
  - Transmission
  - Human-interface

I/O HARDWARE

A TYPICAL PC BUS STRUCTURE



I/O HARDWARE

- I/O instructions control devices
- Devices usually have registers where device driver places commands, addresses, and data to write, or read data from registers after command execution
  - Data-in register, data-out register, status register, control register
  - Typically 1-4 bytes, or FIFO buffer

I/O HARDWARE

- Devices have addresses, used by
  - Direct I/O instructions
  - Memory-mapped I/O
    - Device data and command registers mapped to processor address space
    - Especially for large address spaces (graphics)

DEVICE I/O PORT LOCATIONS ON PCS (PARTIAL)

I/O address range (hexadecimal)	device
000–00F	DMA controller
020–021	interrupt controller
040–043	timer
200–20F	game controller
2F8–2FF	serial port (secondary)
320–32F	hard-disk controller
378–37F	parallel port
3D0–3DF	graphics controller
3F0–3F7	diskette-drive controller
3F8–3FF	serial port (primary)

POLLING

- For each byte of I/O
  1. Read busy bit from status register until 0
  2. Host sets read or write bit and if write copies data into data-out register
  3. Host sets command-ready bit
  4. Controller sets busy bit, executes transfer
  5. Controller clears busy bit, error bit, command-ready bit when transfer done

POLLING

- Step 1 is busy-wait cycle to wait for I/O from device
  - Reasonable if device is fast
  - But inefficient if device slow
  - CPU switches to other tasks?
    - But if miss a cycle data overwritten / lost

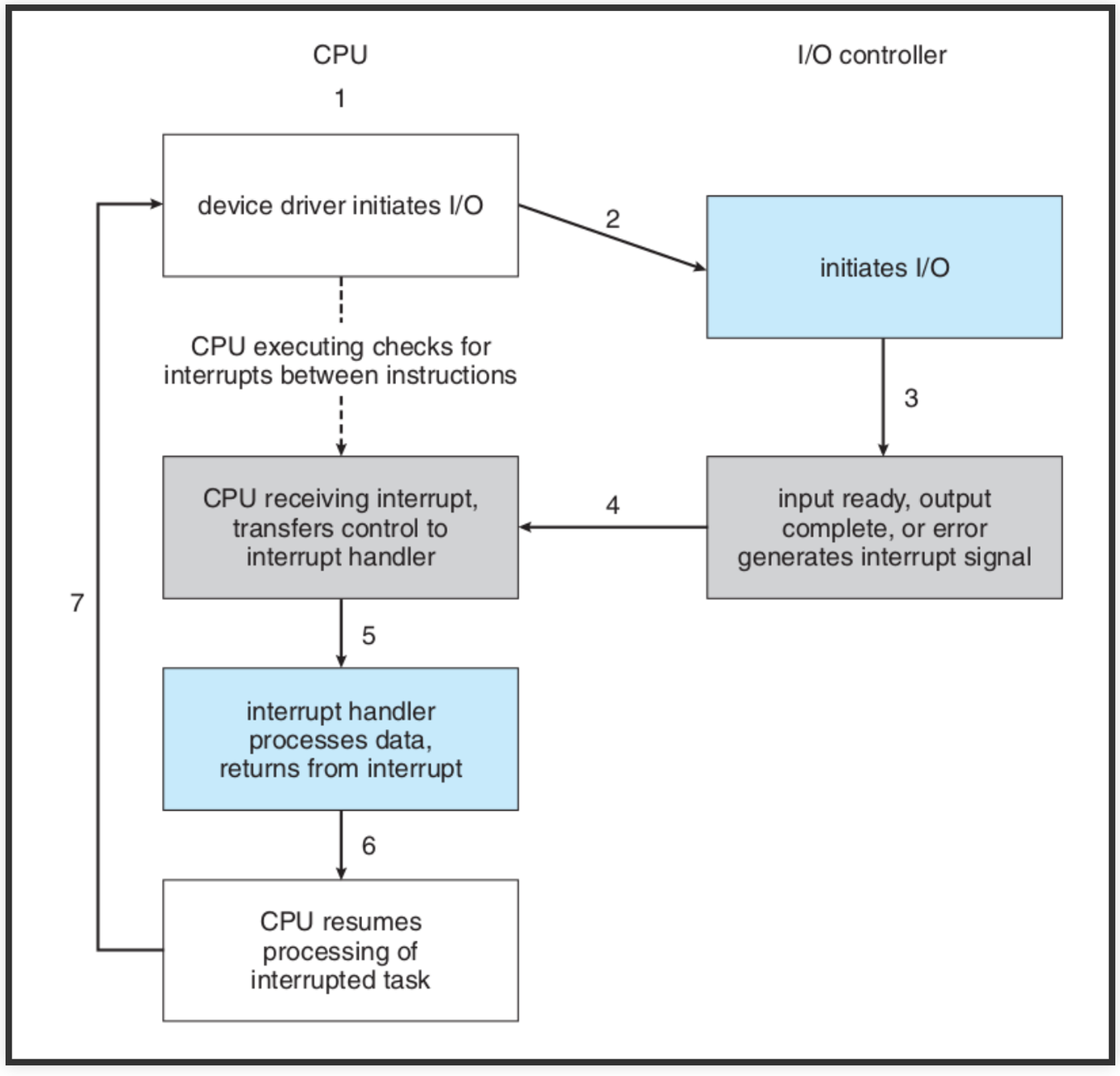
INTERRUPTS

- Polling can happen in 3 instruction cycles
  - Read status, logical-and to extract status bit, branch if not zero
  - How to be more efficient if non-zero infrequently?
- CPU Interrupt-request line triggered by I/O device
  - Checked by processor after each instruction
- Interrupt handler receives interrupts
  - Maskable to ignore or delay some interrupts

INTERRUPTS

- Interrupt vector to dispatch interrupt to correct handler
  - Context switch at start and end
  - Based on priority
  - Some nonmaskable
  - Interrupt chaining if more than one device at same interrupt number

INTERRUPT-DRIVEN I/O CYCLE



INTEL PENTIUM PROCESSOR EVENT-VECTOR TABLE

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19–31	(Intel reserved, do not use)
32–255	maskable interrupts

INTERRUPTS

- Interrupt mechanism also used for exceptions
  - Terminate process, crash system due to hardware error
- Page fault executes when memory access error
- System call executes via trap to trigger kernel to execute request
- Multi-CPU systems can process interrupts concurrently
  - If operating system designed to handle it
- Used for time-sensitive processing, frequent, must be fast

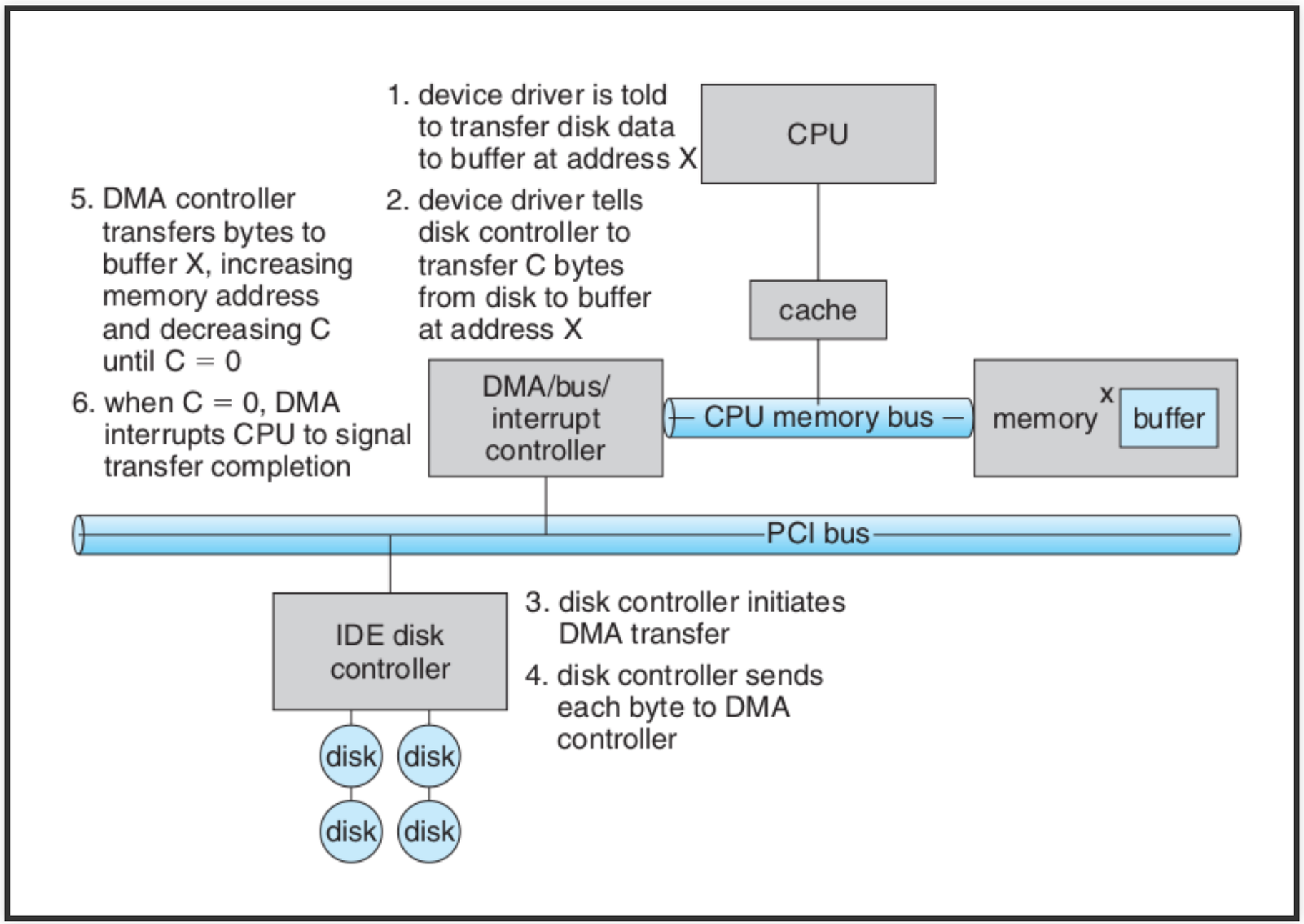
DIRECT MEMORY ACCESS

- Used to avoid programmed I/O (one byte at a time) for large data movement
- Requires DMA controller
- Bypasses CPU to transfer data directly between I/O device and memory

DIRECT MEMORY ACCESS

- OS writes DMA command block into memory
  - Source and destination addresses
  - Read or write mode
  - Count of bytes
  - Writes location of command block to DMA controller
  - Bus mastering of DMA controller – grabs bus from CPU
  - When done, interrupts to signal completion

SIX STEP PROCESS TO PERFORM DMA TRANSFER



APPLICATION I/O INTERFACE

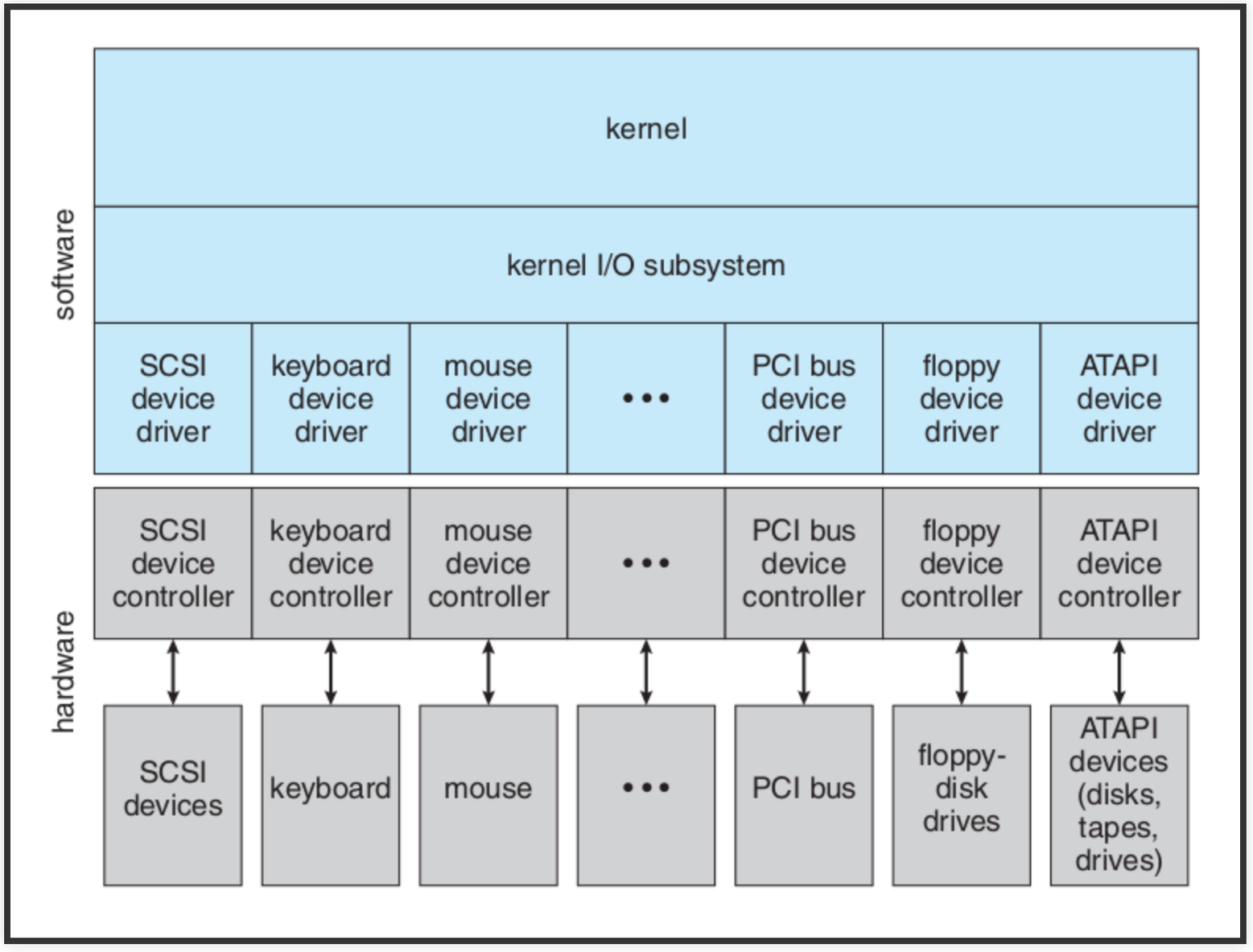
APPLICATION I/O INTERFACE

- I/O system calls encapsulate device behaviors in generic classes
- Device-driver layer hides differences among I/O controllers from kernel
- New devices talking already-implemented protocols need no extra work
- Each OS has its own I/O subsystem structures and device driver frameworks

APPLICATION I/O INTERFACE

- Devices vary in many dimensions
  - Character-stream or block
  - Sequential or random-access
  - Synchronous or asynchronous (or both)
  - Sharable or dedicated
  - Speed of operation
  - read-write, read only, or write only

A KERNEL I/O STRUCTURE



CHARACTERISTICS OF I/O DEVICES

aspect	variation	example
data-transfer mode	character block	terminal disk
access method	sequential random	modem CD-ROM
transfer schedule	synchronous asynchronous	tape keyboard
sharing	dedicated sharable	tape keyboard
device speed	latency seek time transfer rate delay between operations	
I/O direction	read only write only read–write	CD-ROM graphics controller disk

CHARACTERISTICS OF I/O DEVICES

- Subtleties of devices handled by device drivers
- Broadly I/O devices can be grouped by the OS into
  - Block I/O
  - Character I/O (Stream)
  - Memory-mapped file access
  - Network sockets

CHARACTERISTICS OF I/O DEVICES

- For direct manipulation of I/O device specific characteristics, usually an escape / back door
  - Unix ioctl() call to send arbitrary bits to a device control register and data to device data register

BLOCK DEVICES

- Block devices include disk drives
  - Commands include read, write, seek
  - Raw I/O, direct I/O, or file-system access
  - Memory-mapped file access possible
    - File mapped to virtual memory and clusters brought via demand paging
  - DMA

CHARACTER DEVICES

- Character devices include keyboards, mice, serial ports
  - Commands include get(), put()
  - Libraries layered on top allow line editing

NETWORK DEVICES

- Varying enough from block and character to have own interface
- Unix and Windows include socket interface
  - Separates network protocol from network operation
  - Includes select() functionality
- Approaches vary widely (pipes, FIFOs, streams, queues, mailboxes)

CLOCKS AND TIMERS

- Provide current time, elapsed time, timer
- Normal resolution about 1/60 second
- Some systems provide higher-resolution timers
- Programmable interval timer used for timings, periodic interrupts
- ioctl() (on UNIX) covers odd aspects of I/O such as clocks and timers

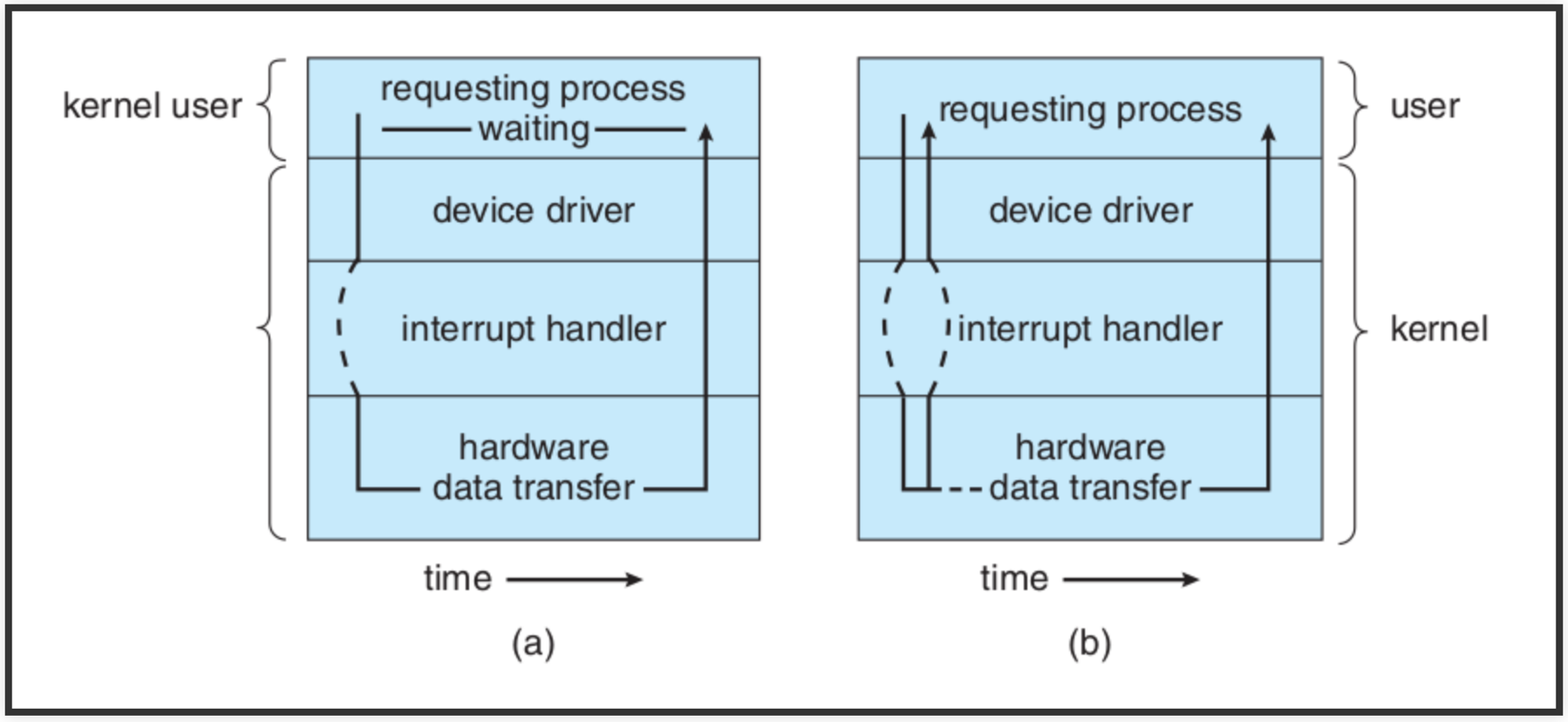
BLOCKING AND NONBLOCKING I/O

- Blocking - process suspended until I/O completed
  - Easy to use and understand
  - Insufficient for some needs

BLOCKING AND NONBLOCKING I/O

- Nonblocking - I/O call returns as much as available
  - User interface, data copy (buffered I/O)
  - Implemented via multi-threading
  - Returns quickly with count of bytes read or written
  - select() to find if data ready then read() or write() to transfer
- Asynchronous - process runs while I/O executes
  - Difficult to use
  - I/O subsystem signals process when I/O completed

TWO I/O METHODS



KERNEL I/O SUBSYSTEM

SCHEDULING

- Some I/O request ordering via per-device queue
- Some OSs try fairness
- Some implement Quality Of Service (i.e. IPQOS)

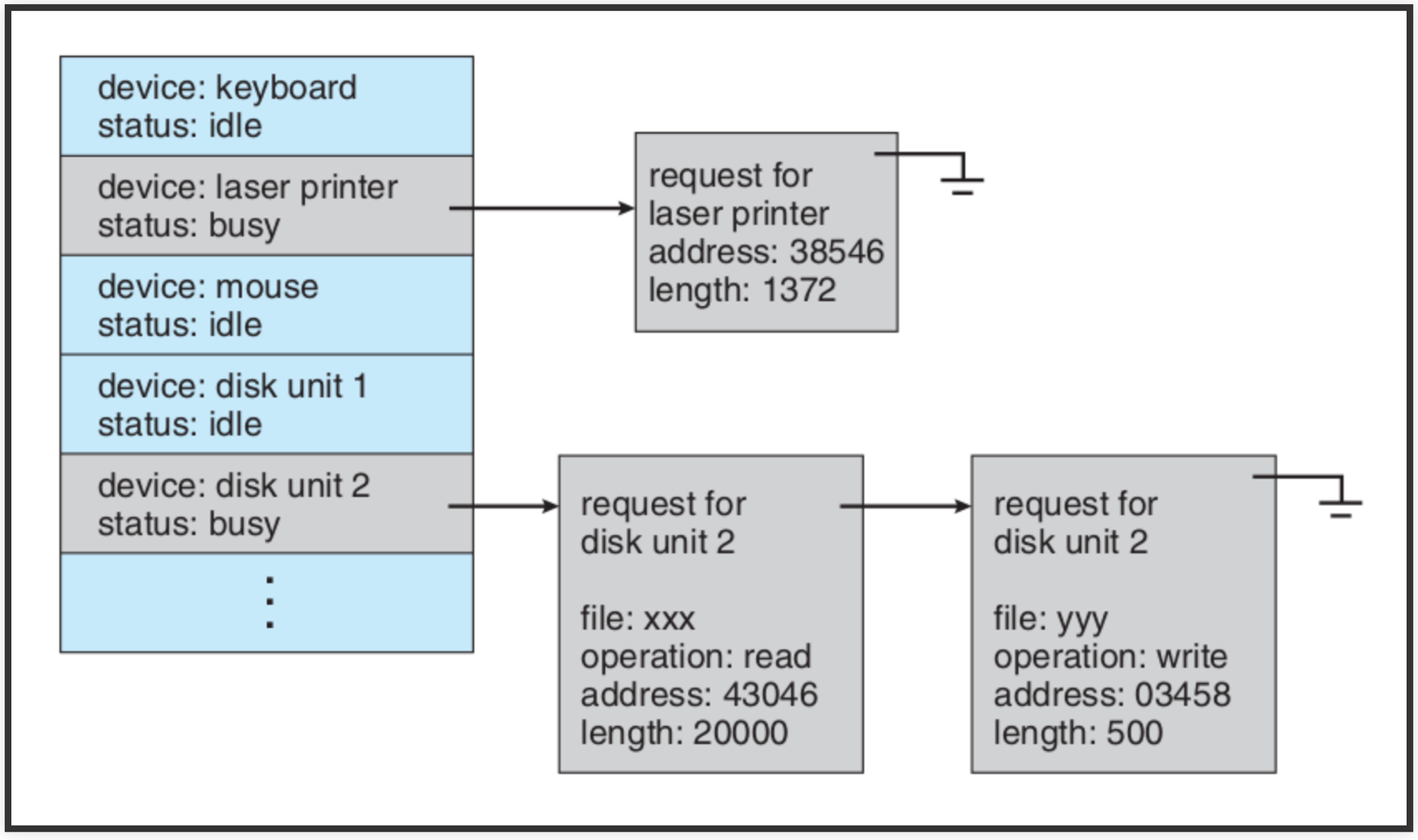
BUFFERING

- store data in memory while transferring between devices
  - To cope with device speed mismatch
  - To cope with device transfer size mismatch
  - To maintain “copy semantics”

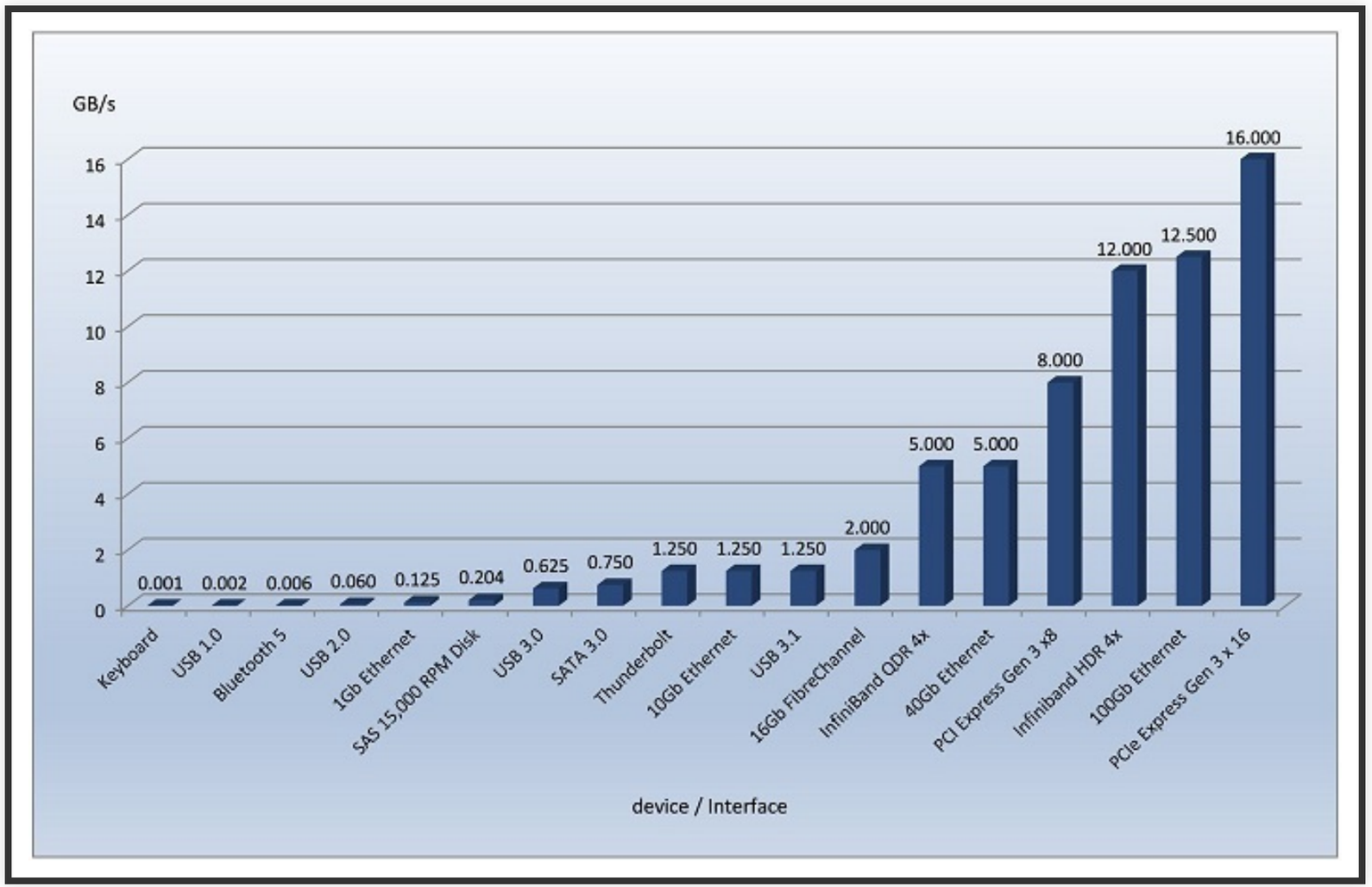
DOUBLE BUFFERING

- Double buffering – two copies of the data
  - Kernel and user
  - Varying sizes
  - Full / being processed and not-full / being used
  - Copy-on-write can be used for efficiency in some cases

DEVICE-STATUS TABLE



COMMON PC AND DATA-CENTER I/O DEVICE AND INTERFACE SPEEDS



KERNEL I/O SUBSYSTEM - CACHING

- **Caching**: Faster device holding copy of data
  - Always just a copy
  - Key to performance
  - Sometimes combined with buffering

A buffer, of course, is a memory area that stores data being transferred between two devices or between a device and an application.

KERNEL I/O SUBSYSTEM - SPOOLING

- **Spooling**: hold output for a device
  - If device can serve only one request at a time
  - i.e., Printing

KERNEL I/O SUBSYSTEM - DEVICE RESERVATION

- **Device reservation**: provides exclusive access to a device
  - System calls for allocation and de-allocation
  - Watch out for deadlock

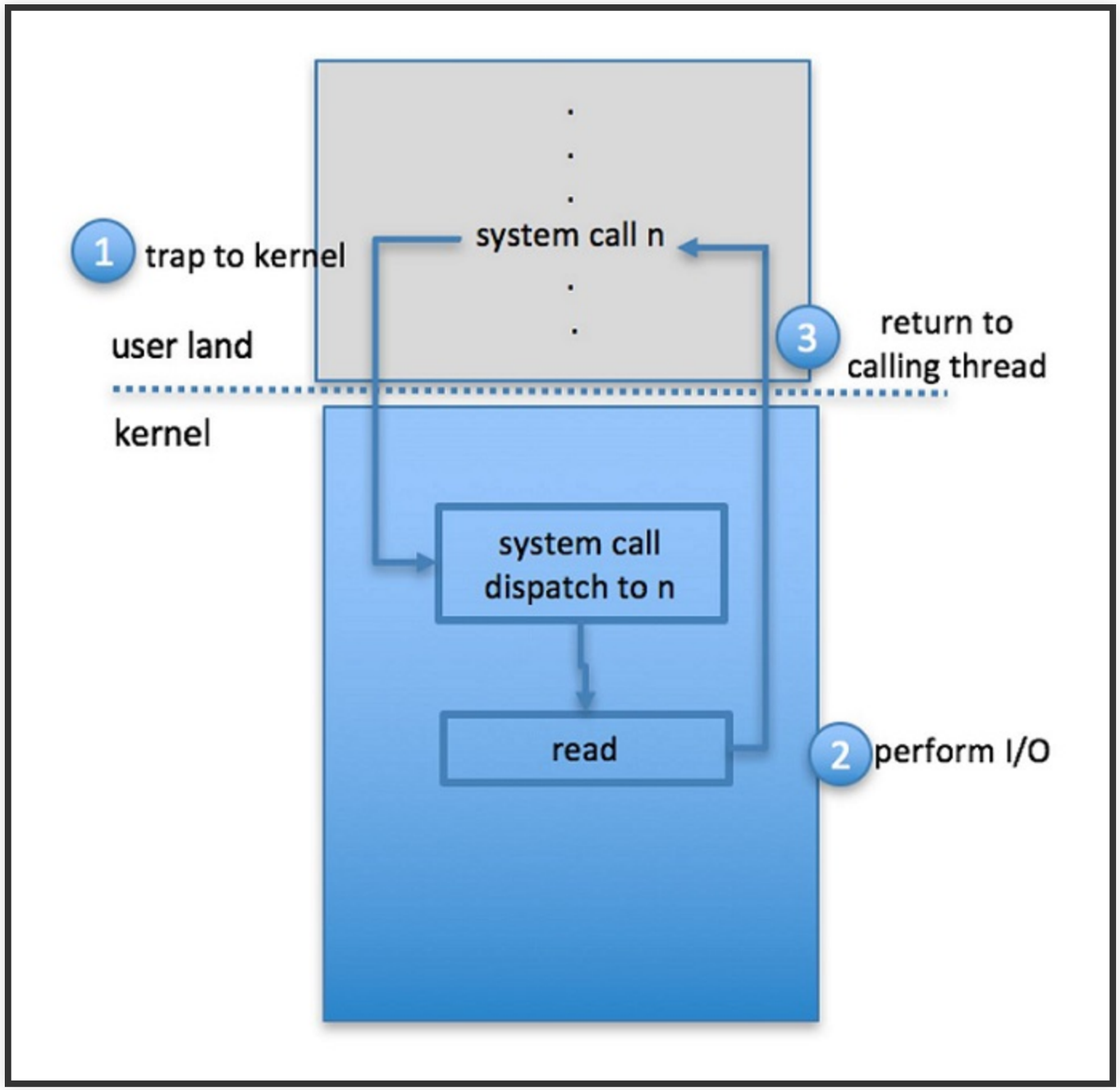
ERROR HANDLING

- OS can recover from disk read, device unavailable, transient write failures
  - Retry a read or write, for example
  - Some systems more advanced – Solaris FMA, AIX
    - Track error frequencies, stop using device with increasing frequency of retry-able errors
- Most return an error number or code when I/O request fails
- System error logs hold problem reports

I/O PROTECTION

- User process may accidentally or purposefully attempt to disrupt normal operation via illegal I/O instructions
  - All I/O instructions defined to be privileged
  - I/O must be performed via system calls
    - Memory-mapped and I/O port memory locations must be protected too

SYSTEM CALL TO PERFORM I/O



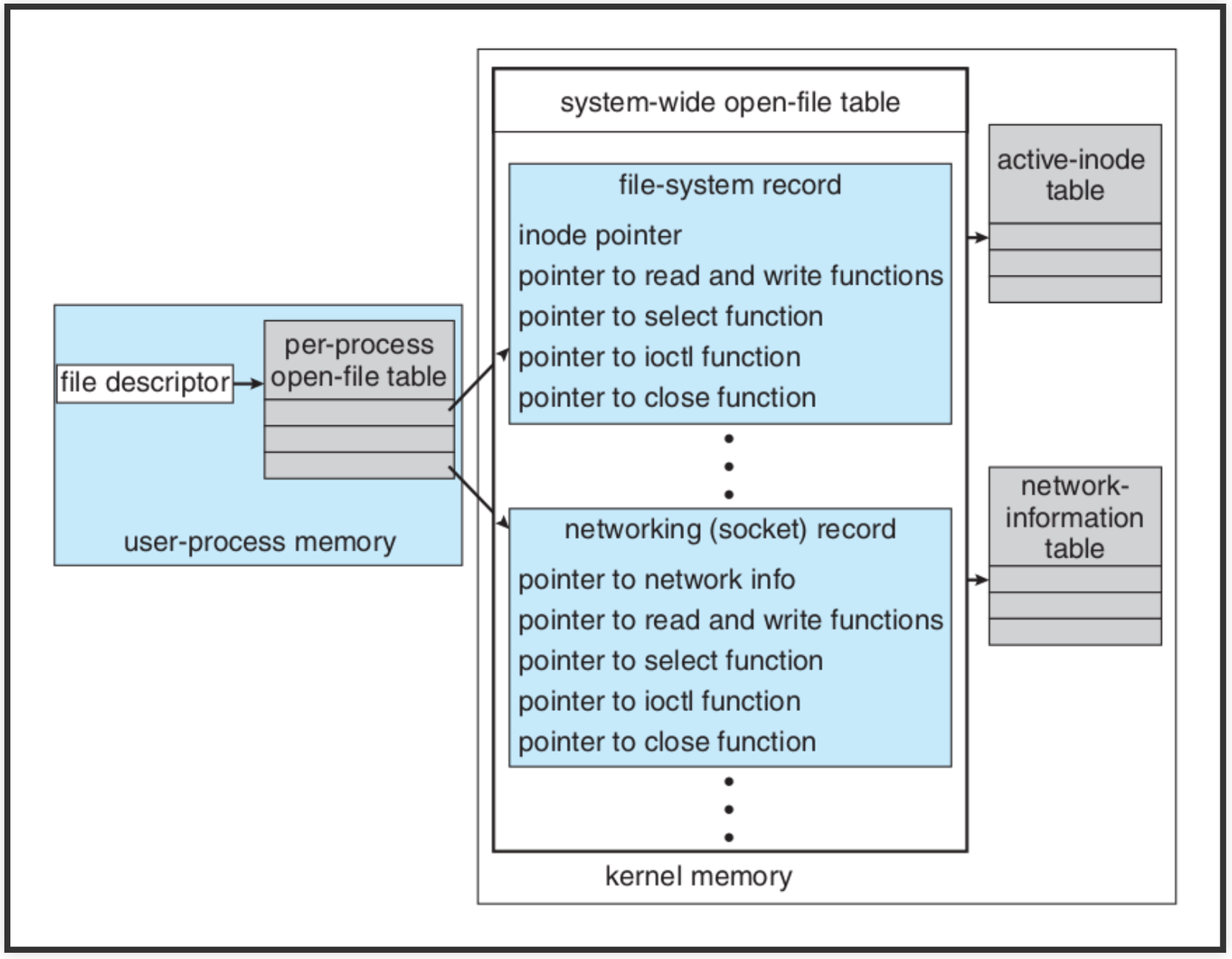
KERNEL DATA STRUCTURES

- Kernel keeps state info for I/O components, including open file tables, network connections, character device state
- Many, many complex data structures to track buffers, memory allocation, “dirty” blocks

KERNEL DATA STRUCTURES

- Some use object-oriented methods and message passing to implement I/O
  - Windows uses message passing
    - Message with I/O information passed from user mode into kernel
    - Message modified as it flows through to device driver and back to process
    - Pros / cons?

UNIX I/O KERNEL STRUCTURE

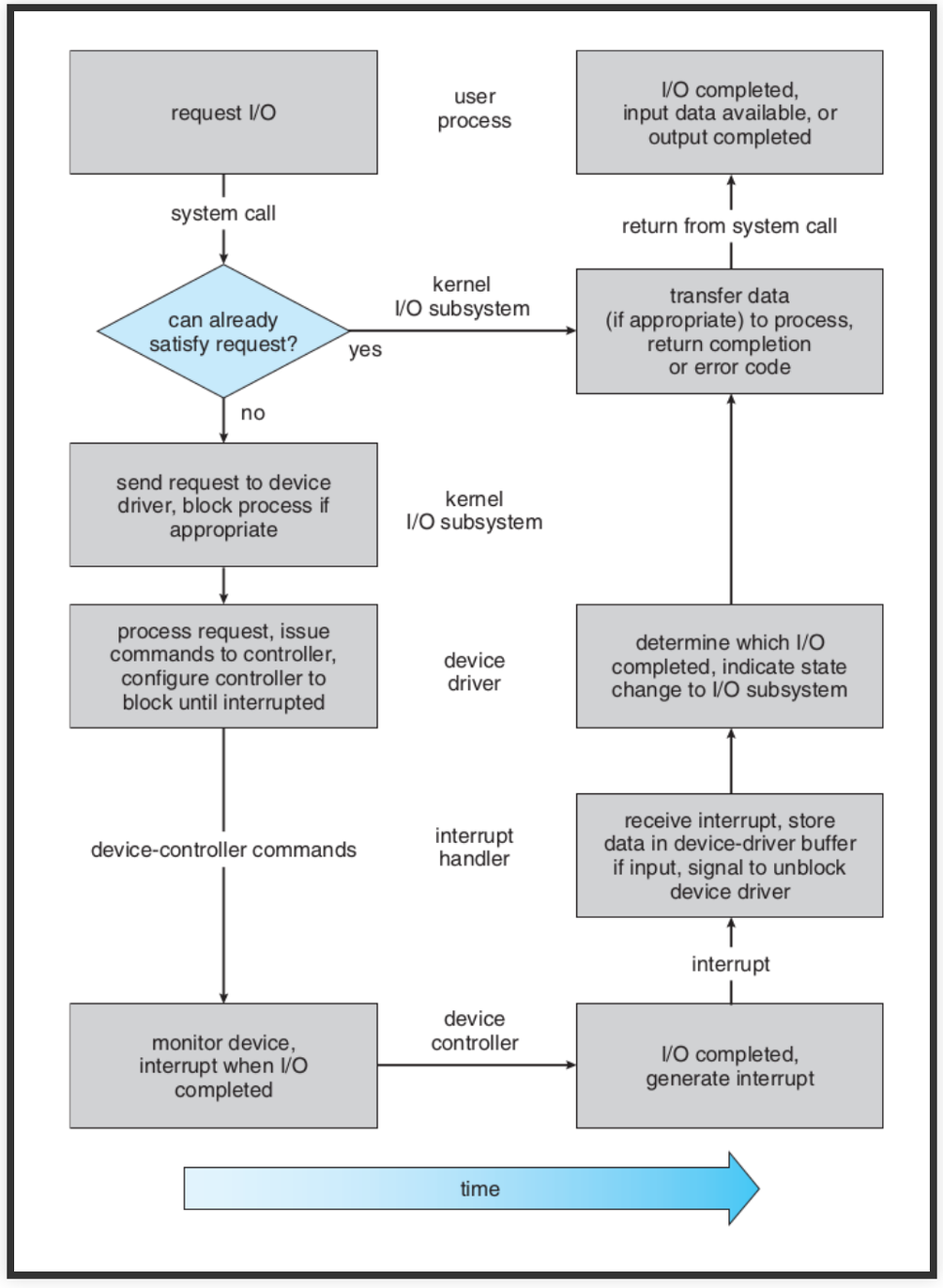


TRANSFORMING I/O REQUESTS TO HARDWARE OPERATIONS

I/O REQUESTS TO HARDWARE OPERATIONS

- Consider reading a file from disk for a process:
  - Determine device holding file
  - Translate name to device representation
  - Physically read data from disk into buffer
  - Make data available to requesting process
  - Return control to process

LIFE CYCLE OF AN I/O REQUEST

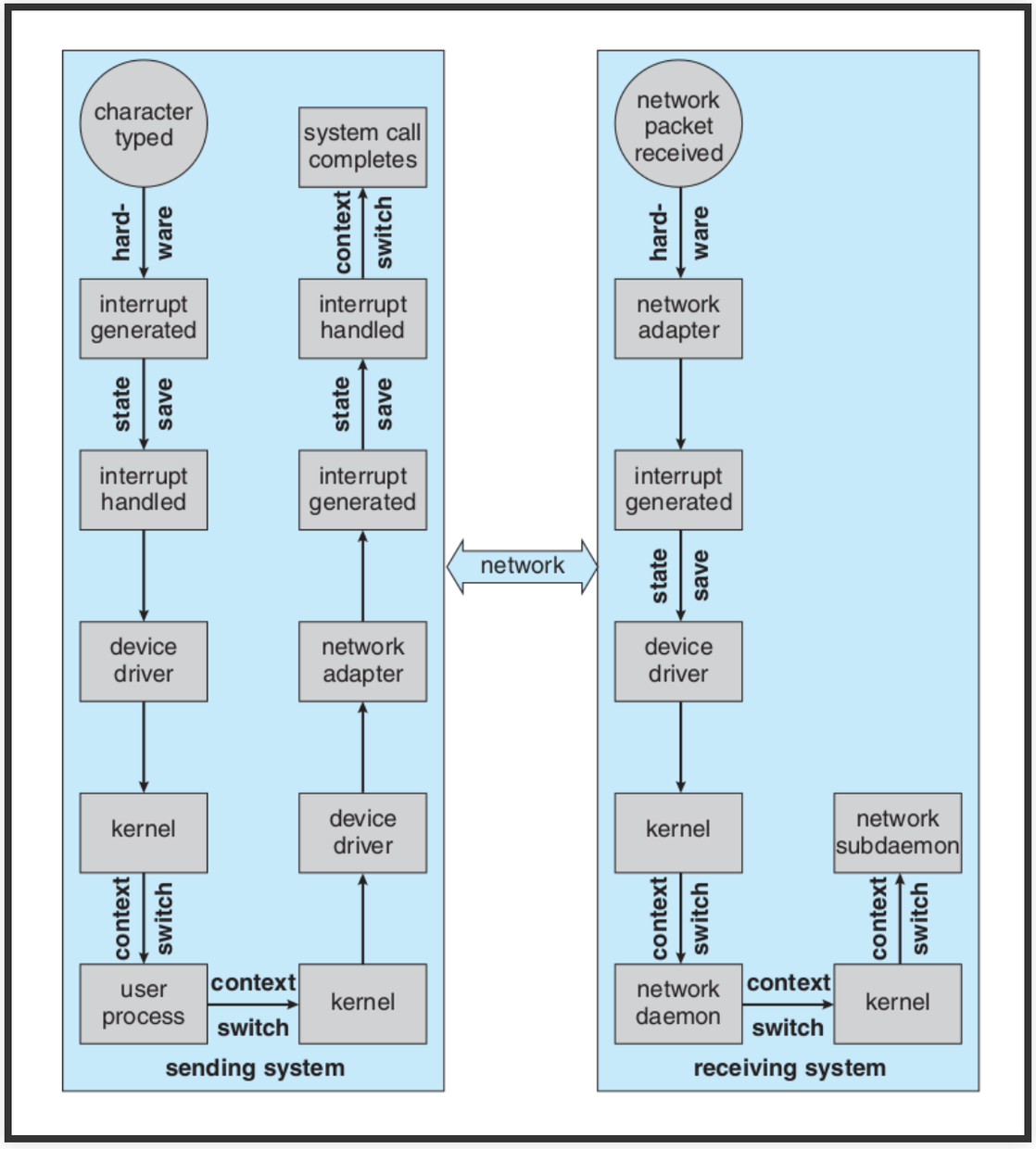

request I/O
user process
I/O completed, input data available, or output completed
system call
return from system call
can already satisfy request?
yes
kernel I/O subsystem
transfer data (if appropriate) to process, return completion or error code
no
send request to device driver, block process if appropriate
kernel I/O subsystem
process request, issue commands to controller, configure controller to block until interrupted
device driver
determine which I/O completed, indicate state change to I/O subsystem
device-controller commands
interrupt handler
receive interrupt, store data in device-driver buffer if input, signal to unblock device driver
interrupt
monitor device, interrupt when I/O completed
device controller
I/O completed, generate interrupt
time

PERFORMANCE

PERFORMANCE

- I/O a major factor in system performance:
  - Demands CPU to execute device driver, kernel I/O code
  - Context switches due to interrupts
  - Data copying
  - Network traffic especially stressful

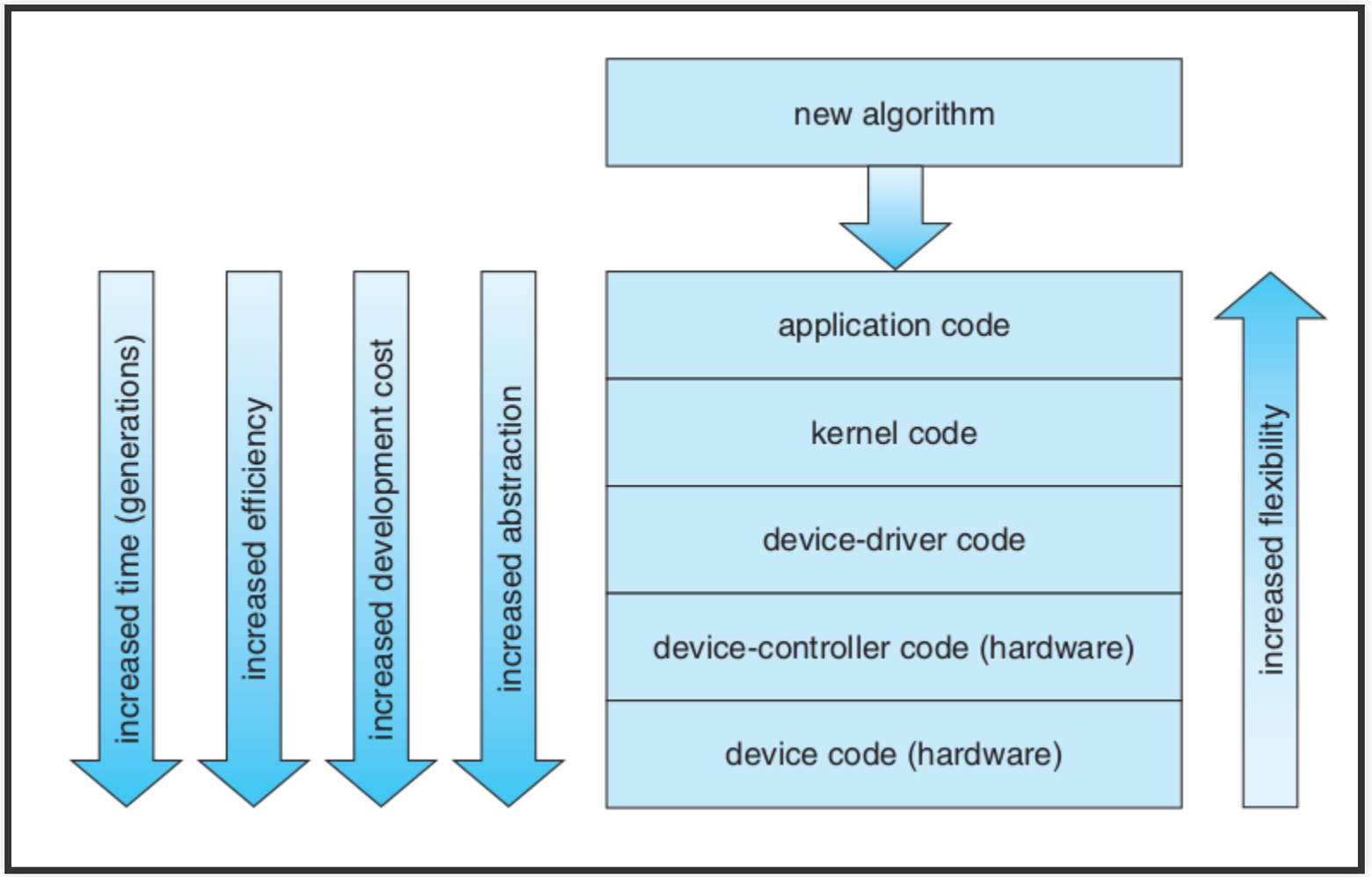
INTERCOMPUTER COM.



IMPROVING PERFORMANCE

- Reduce number of context switches
- Reduce data copying
- Reduce interrupts by using large transfers, smart controllers, polling
- Use DMA
- Use smarter hardware devices
- Balance CPU, memory, bus, and I/O performance for highest throughput
- Move user-mode processes / daemons to kernel threads

DEVICE-FUNCTIONALITY PROGRESSION



QUESTIONS

BONUS

Exam question number 9: **Mass-Storage Structure & I/O-Systems**