Operating Systems
COMP2006

Mass Storage and I/O
Lecture 9

Mass Storage and I/O

References:

Silberschatz, Galvin, and Gagne, *Operating System Concepts*, Chapter 10 and 13.

Topics:

- I/O Hardware.
- Application I/O Interface.
- Kernel I/O Subsystem.
- Transforming I/O Requests to Hardware Operations.
- Performance.
- Disk Structure, Scheduling, and Management.
- Swap-Space Management.
- Disk Reliability.

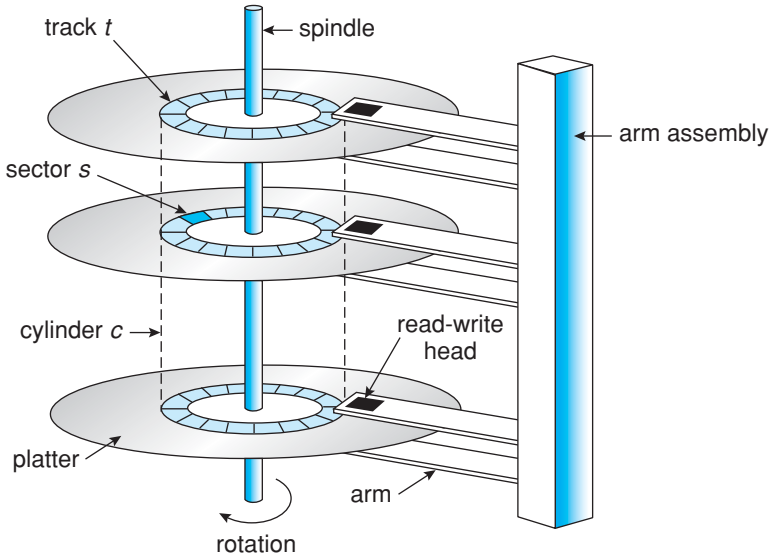
Mass Storage

- Main memory (primary storage) is volatile and too small to store all data and programs permanently
  - the computer system provides a secondary storage to back up main memory.
- Most modern systems use disks as the main on-line storage medium for both programs and data.
- The OS is responsible for the following activities:
  - Free-space management.
  - Storage allocation.
  - Disk scheduling.

Disk Structure

- Disk structure is the lowest level of the file system.
  - Disks are used as secondary storage.
- Disk drives are addressed as large 1-dimensional arrays of logical blocks.
  - A logical block is the smallest unit of data transfer.
  - A logical block is usually 512 bytes; use low-level format for different logical block size.

Disk Structure



- 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.
  - There can be hundreds of sectors per track and tens of thousands of cylinders.

Disk Scheduling

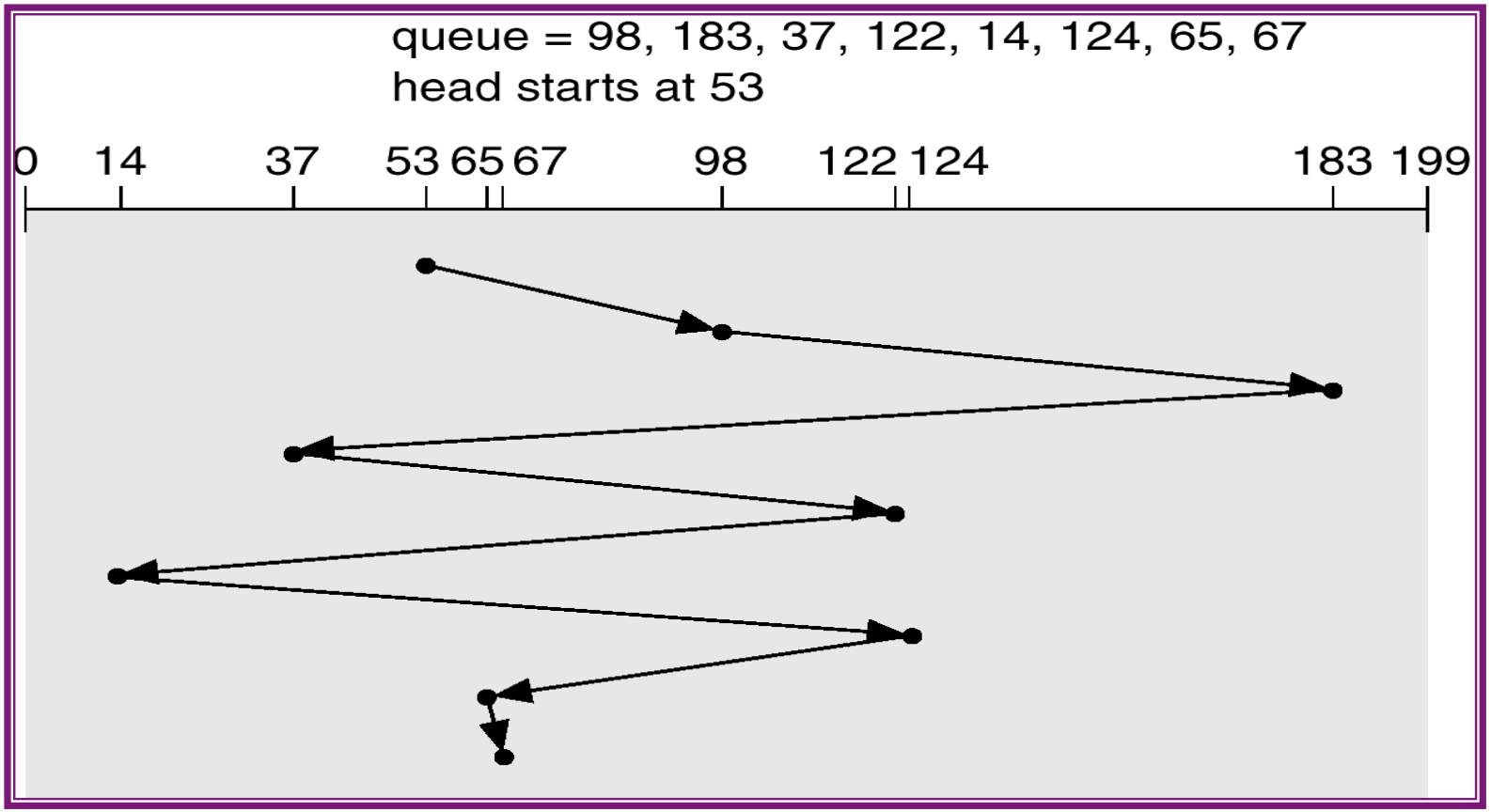
- OS is responsible for using hardware efficiently
  - For the disk drives, this means having a fast *access time* and *disk bandwidth.*
- *Access time* has two major components:
  - *Seek time*: the time for the disk arm to move the heads to the cylinder containing the desired sector.
  - *Rotational latency*: the additional time waiting for the disk to rotate the desired sector to the disk head.
- *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 Scheduling (cont.)

- A process request for disk I/O requires the following information:
  - Is the I/O for input or output?
  - The disk address for the transfer.
  - The memory address for the transfer.
  - The number of sectors to be transferred.
- For multiprogramming, disk requests are placed in a queue, and OS chooses which pending request to service next
  - OS uses a disk scheduling algorithm
- Goal of the algorithms: to reduce the total disk head movement (i.e., seek time)

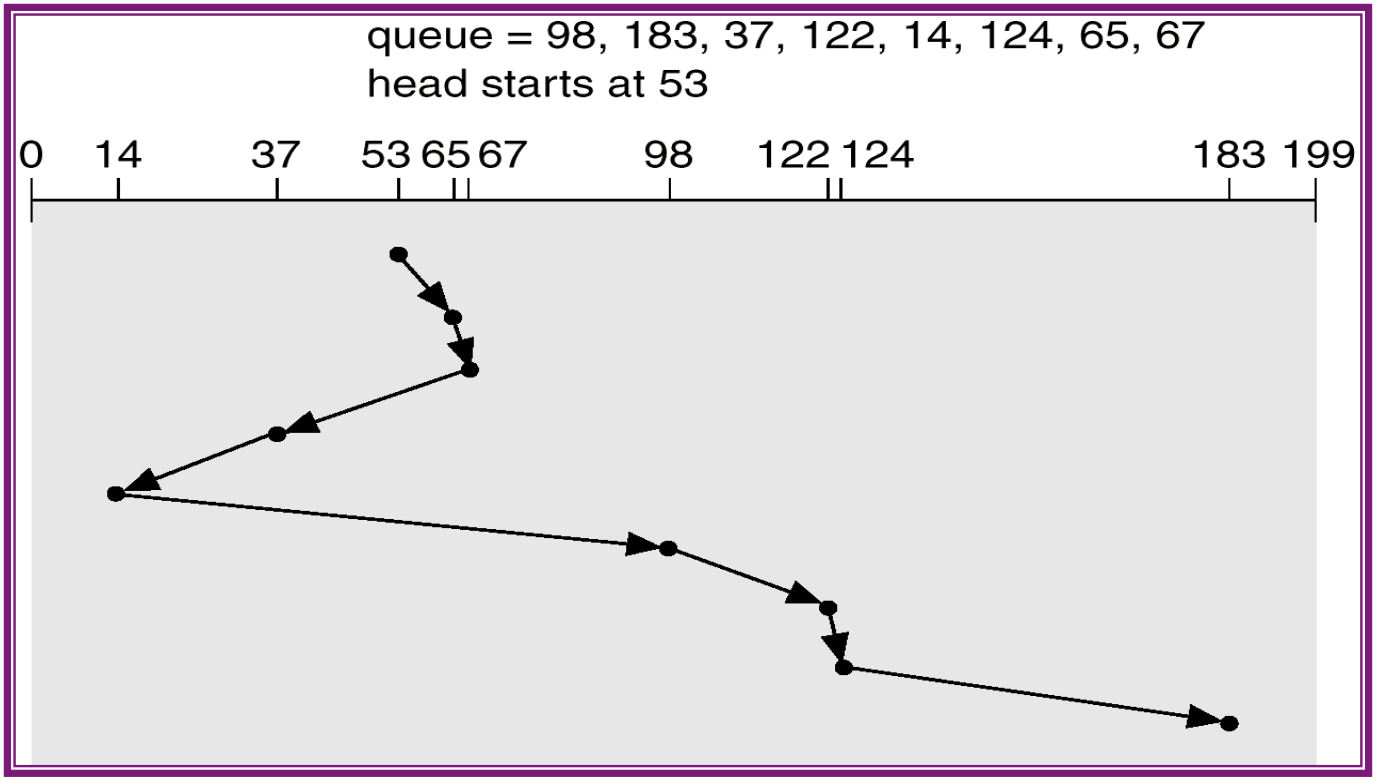
First Come First Served (FCFS)

- FCFS is the simplest disk scheduling algorithm.
  - It is fair, but it does not provide the fastest performance.
- The following illustration shows total head movement of 640 cylinders.



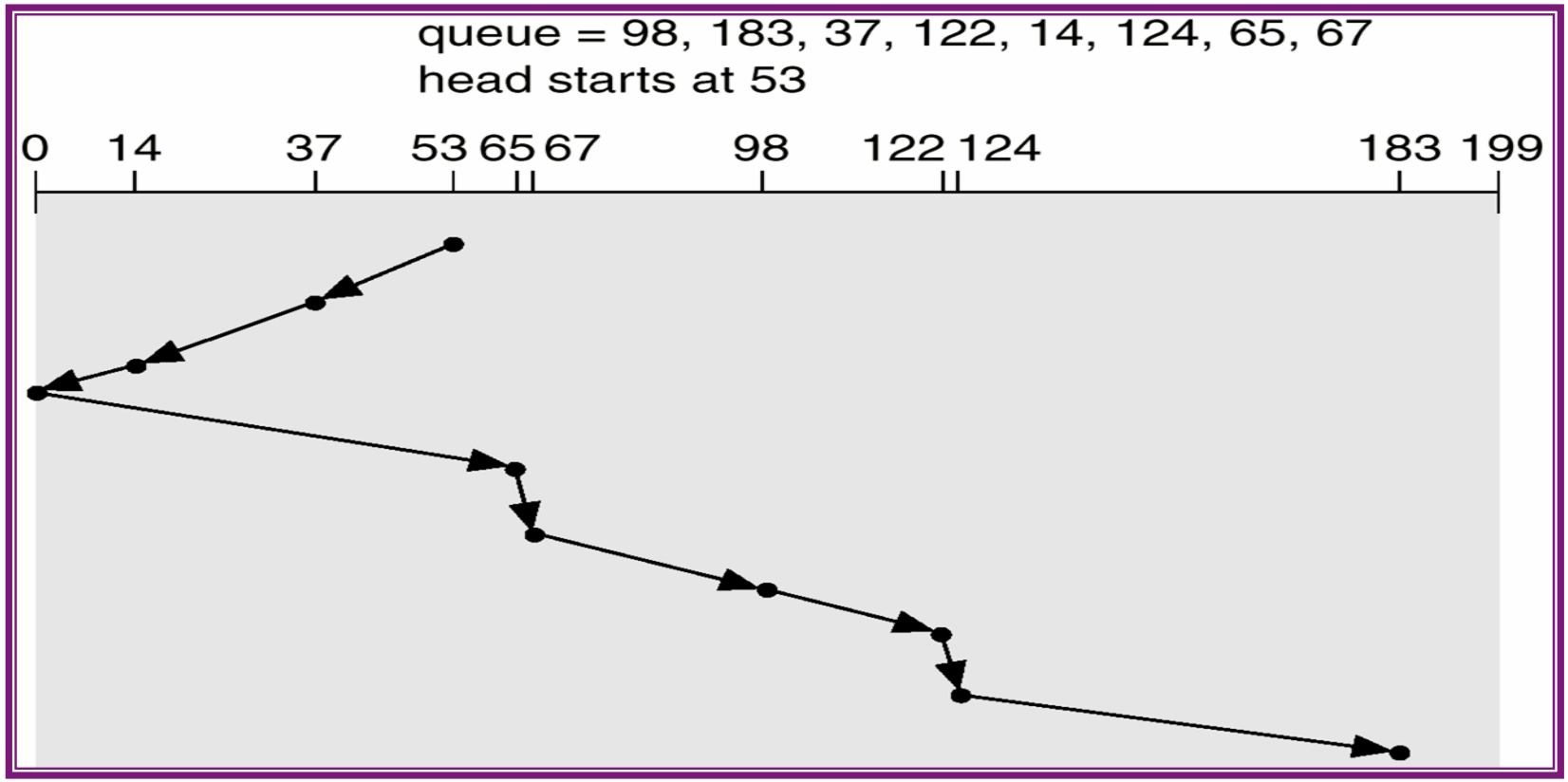
Shortest Seek Time First (SSTF)

- SSTF selects the request with the minimum seek time from the current head position.
  - It is a form of SJF scheduling.
  - It may cause starvation of some requests.
- The following illustration shows total head movement of 236 cylinders.



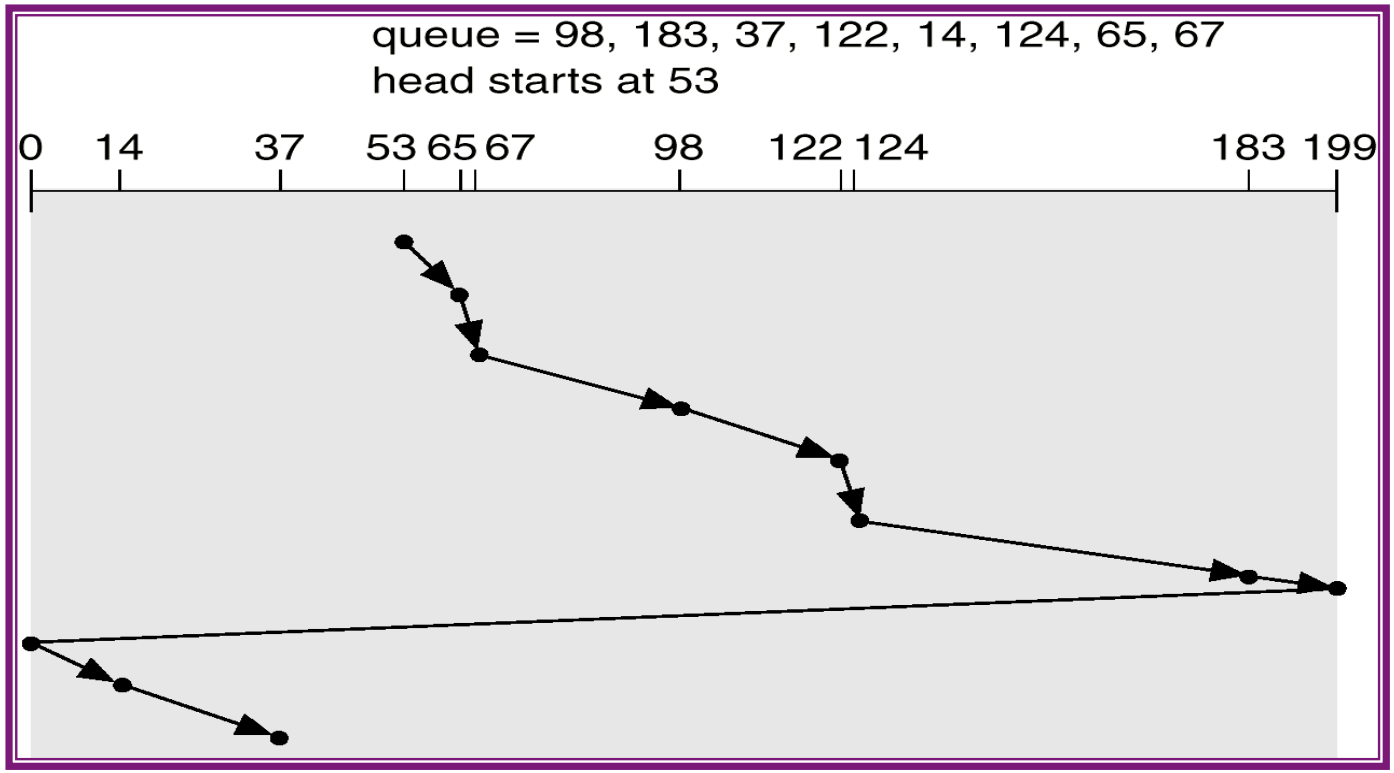
SCAN

- In 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 is sometimes called the elevator algorithm.
- The following illustration shows total head movement of 236 cylinders.



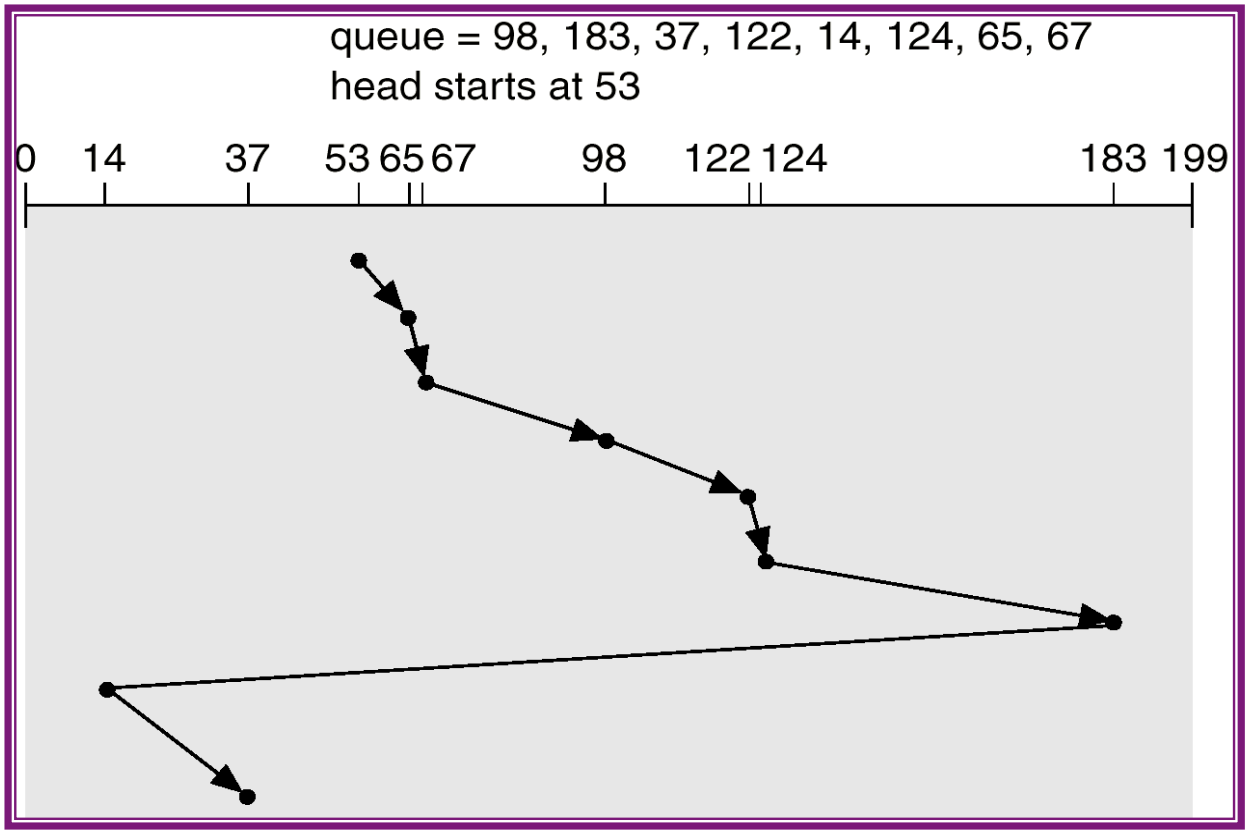
C-SCAN

- In C-SCAN, the head moves from one end of the disk to the other, servicing requests as it goes. When it reaches the other end, it immediately returns to the beginning of the disk, without servicing any requests on the return trip.
  - It provides a more uniform wait time than SCAN.
  - It treats the cylinders as a circular list that wraps around from the last cylinder to the first one.
- The following illustration shows total head movement of 382 cylinders.



C-LOOK

- In C-LOOK, the disk arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk.
  - C-LOOK is a version of C-SCAN.
  - LOOK is a version of SCAN
- The following illustration shows total head movement of 322 cylinders.



Disk Management

Disk formatting

- Disk must be divided into sectors that the disk controller can read and write → called low level formatting.
  - Low-level formatting fills the disk with a special data structure for each sector
- Data structure for each sector typically consists of a *header*, a *data area* (e.g., 512 bytes), and a *trailer*.
  - Header and trailer contain information used by disk controller, i.e., sector number and ECC.
- To use a disk to hold files, OS needs to record its own data structures on the disk; two steps:
  1) *Partition* the disk into one or more groups of cylinders
     - Each partition can be considered as a separate disk.
  2) *Logical formatting* or 'making a file system'
     - OS stores the initial file-system data structures onto the disk (include a FAT or inode, and empty directory).

Disk Management (cont.)

Boot Block

- A computer needs an initial program to run, i.e., a bootstrap program.
  - Bootstrap initializes all aspects of the system: CPU registers, device controllers, contents of main memory, and starts the OS.
  - Bootstrap finds the OS kernel on disk, loads that kernel into memory, and jumps to an initial address to start the OS execution.
- A small program in ROM (Bootstrap loader) is used to bring in a full bootstrap program from disk.
- Boot-block is a fixed location in hard disk partition where the full bootstrap program is stored.
  - Boot-disk or system disk is the disk that has a boot partition.

Disk Management (cont.)

Bad Blocks

- Disks are prone to failure:
  - If the disk is completely failed, we need to replace it.
  - It usually contains one or more defective sectors.
  - It may also have bad blocks from factory.
- Different ways to handle bad blocks:
  - Handled manually (in IDE controller): scans the disk to find bad blocks and records this information into corresponding FAT.
    - This is done during logical formatting, or using chkdisk.
  - Maintain a list of bad blocks on the disk.
    - Started from low level formatting, and is updated over the life of the disk.
- SCSI uses sector sparing or forwarding to handle bad blocks.
  - During low level format, it sets aside spare sectors not visible to the OS.
  - SCSI disk controller puts the content of a bad sector into a spare sector.

Swap Space Management

- Swap-space: virtual memory uses disk-space as an extension of main memory
  - swap-space management is another task of OS.
- Main goal: to provide the best throughput for the virtual memory system.
  - Remember: disk access is much slower than memory access.

Swap-space use:

- It can be used to hold the entire process image, including code and data segments.
- It can be used to store pages that have been pushed out of main memory.
- It can be used in other various ways by different OS's.

- The size of swap-space varies.
  - It is safer to overestimate size; just waste disk space.
  - If underestimated, it may result in aborted processes or system crashed.

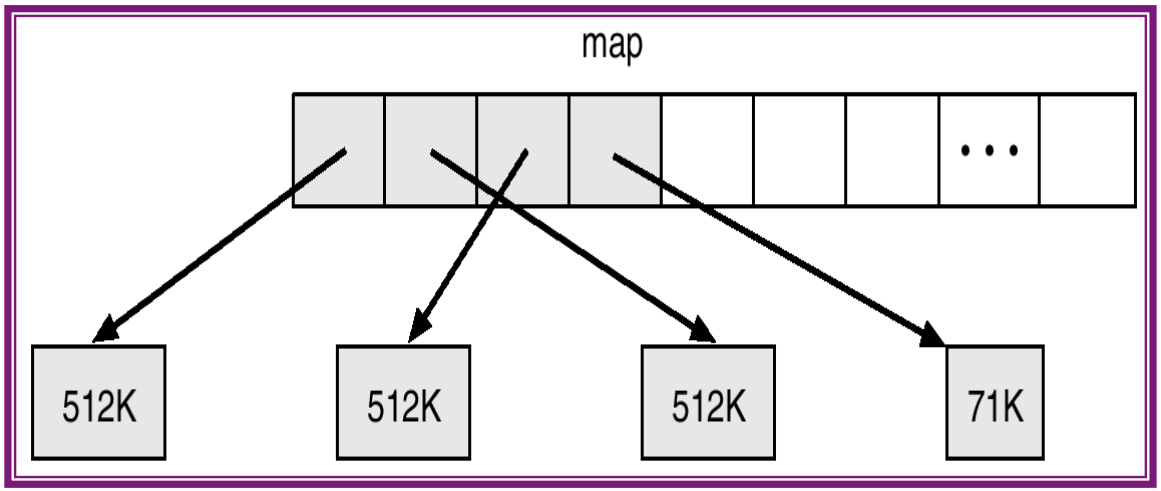
Swap Space Management (cont.)

Swap-space location:

- Two places that a swap space can reside:
  1) Swap space is a large file within the file system.
     - Easy to implement, but inefficient.
  2) Swap space in a separate disk partition.
     - More commonly used.
     - Use algorithms that optimize speed, not storage efficiency.
     - May create internal fragmentation.

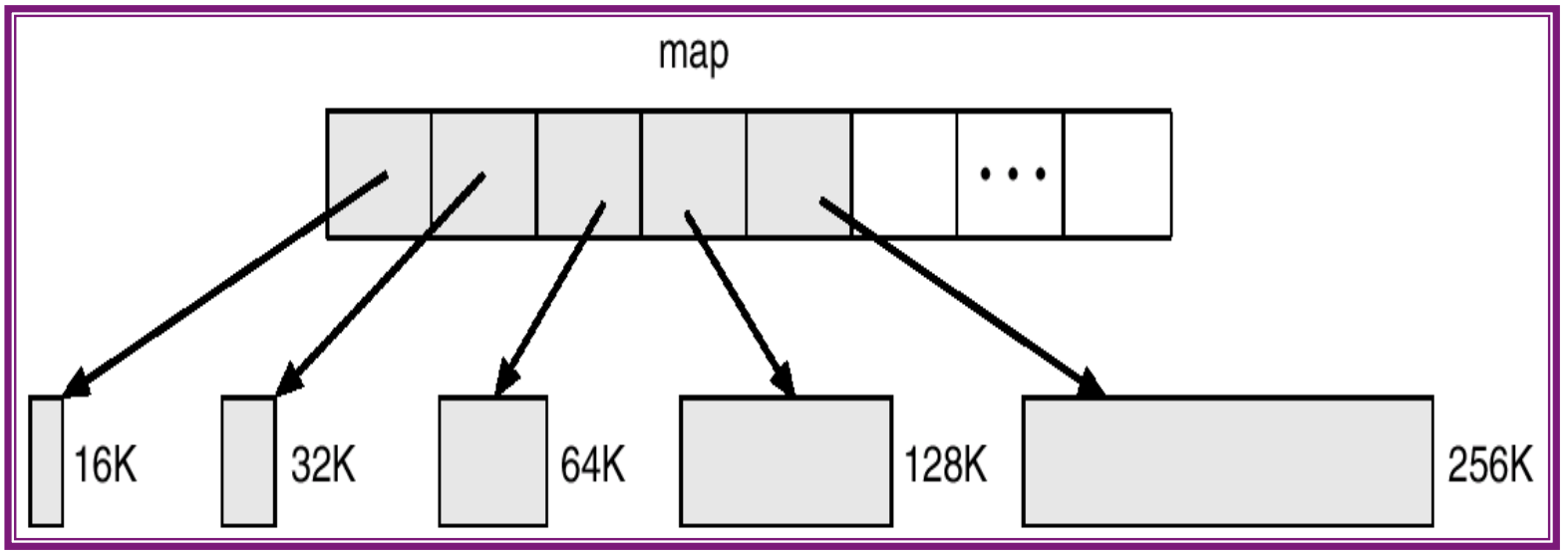
Swap Space Management (cont.)

- Kernel uses swap-map to track swap-space use.
- 4.3BSD allocates swap space when process starts; it holds *text segment*/*pages* (code) and *data segment* of the process.
  - This pre-allocation prevents process from running out of swap space.
  - Optimization: processes with identical text pages share these pages, both in physical memory and swap space.
- **Text segment** is a fixed size → swap space is allocated 512K chunks, except the final chunk (in 1 K increment).



Other Disk Management (cont)

- **Data-segment** can grow over time.
- Map is fixed size containing addresses for varying block sizes.
- For index *i*, a block pointed to by a swap map entry *i* is of size 2^i x 16K, to a max of 2 MB (minimum and maximum size can be changed at boot time).



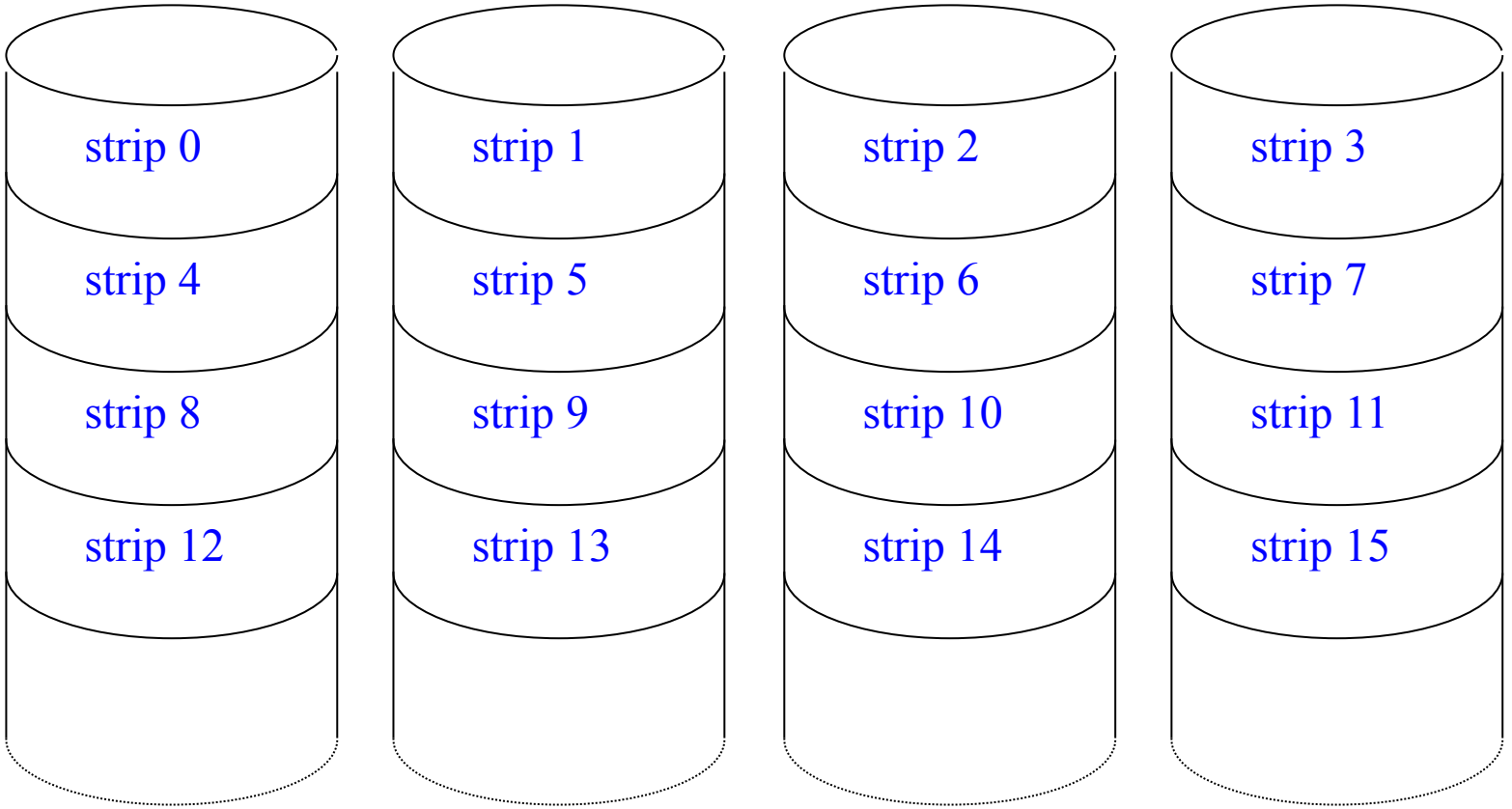
RAID Structure

- Disks are among the least reliable component of a system.
- Recovering from a disk crash may take hours.
  - Thus, improving the reliability of disk system is important.
- Disk performance is critical in improving the overall system performance.
  - disk performance has improved considerably less than that for CPU and main memory.
- Use multiple disks working cooperatively, called RAID – *redundant array of independent disks*.

RAID Structure (cont.)

- RAID improves speed.
  - Disk striping uses a group of disks as one storage unit.
  - A block is broken into several sub-blocks with one sub block stored on each disk.
  - It is fast because access to sub-blocks can be done in parallel.
- RAID improves reliability.
  - Multiple disk drives provides **reliability** via **redundancy.**
  - Mirroring or shadowing keeps duplicates of each disk.
  - Block interleaved parity uses much less redundancy.
- Three common characteristics of RAID:
  - RAID is a set of physical disk drives viewed by the OS as a single logical drive.
  - Data is distributed across an array of physical drives.
  - Redundant disk is used to store parity information, which guarantees data recoverability in case of disk failure.
- RAID has been an industry standard, and consists of Level 0 through Level 6; RAID 2 and RAID 4 are not commercially available.

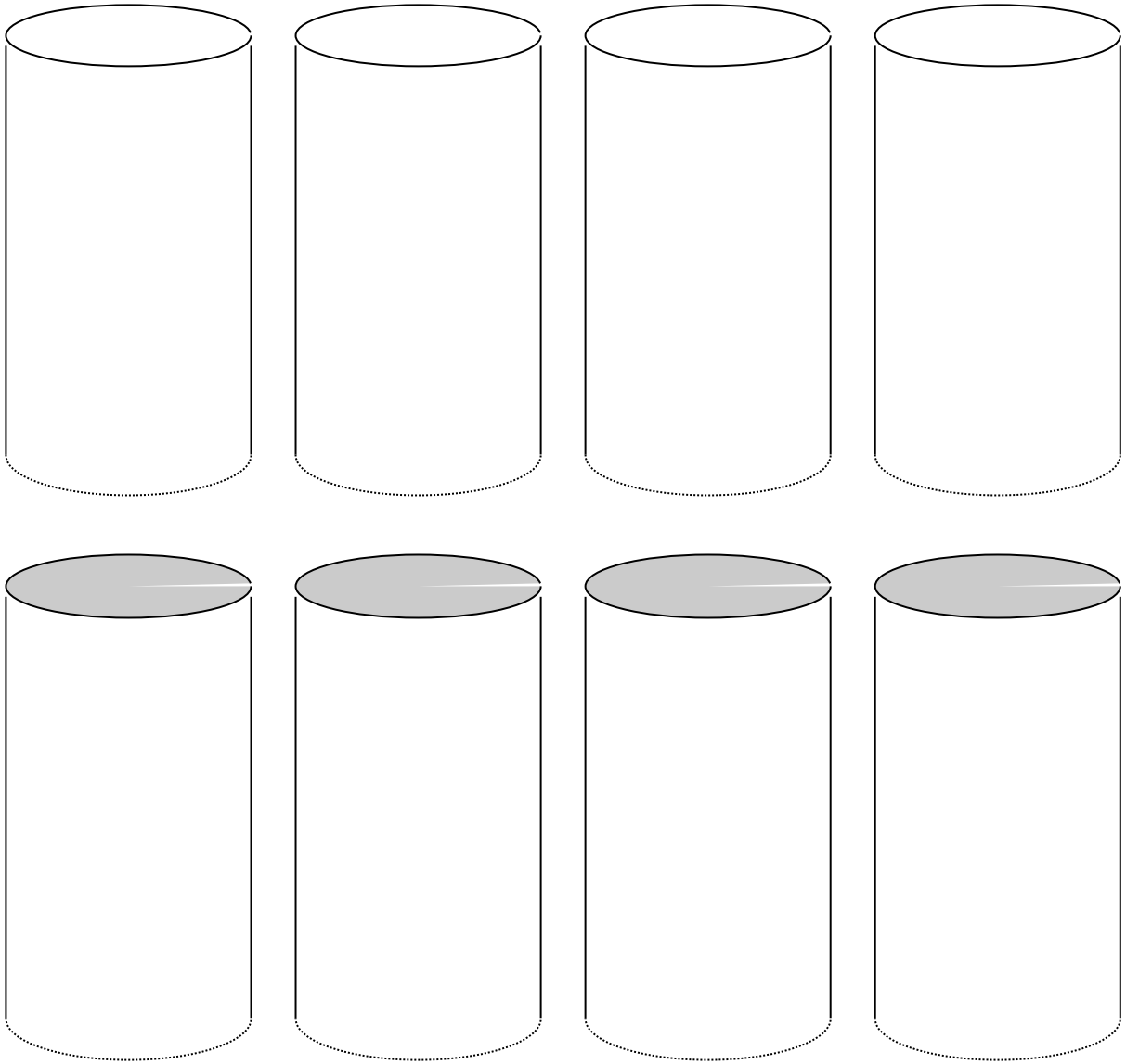
RAID Level 0


strip 0
strip 1
strip 2
strip 3
strip 4
strip 5
strip 6
strip 7
strip 8
strip 9
strip 10
strip 11
strip 12
strip 13
strip 14
strip 15

RAID Level 0

- Goal: to improve performance and capacity, NOT reliability.
- Disks are divided into blocks.
  - User and system data are distributed across all the disks in the array.
  - Different requests for different blocks located in different disks can be issued in parallel.
- For *n*-disk array, the first *n* logical strips are physically stored as the first strip on each of the *n* disks, the second *n* strips are distributed as the second strip on each disk and soon.
  - A single request from multiple contiguous strips can be accessed in parallel (from up to *n* strips); this increases the effective transfer rate, thus higher data transfer capacity.
  - For multiple requests, RAID 0 reduces queuing time for each request; thus higher I/O request rate.

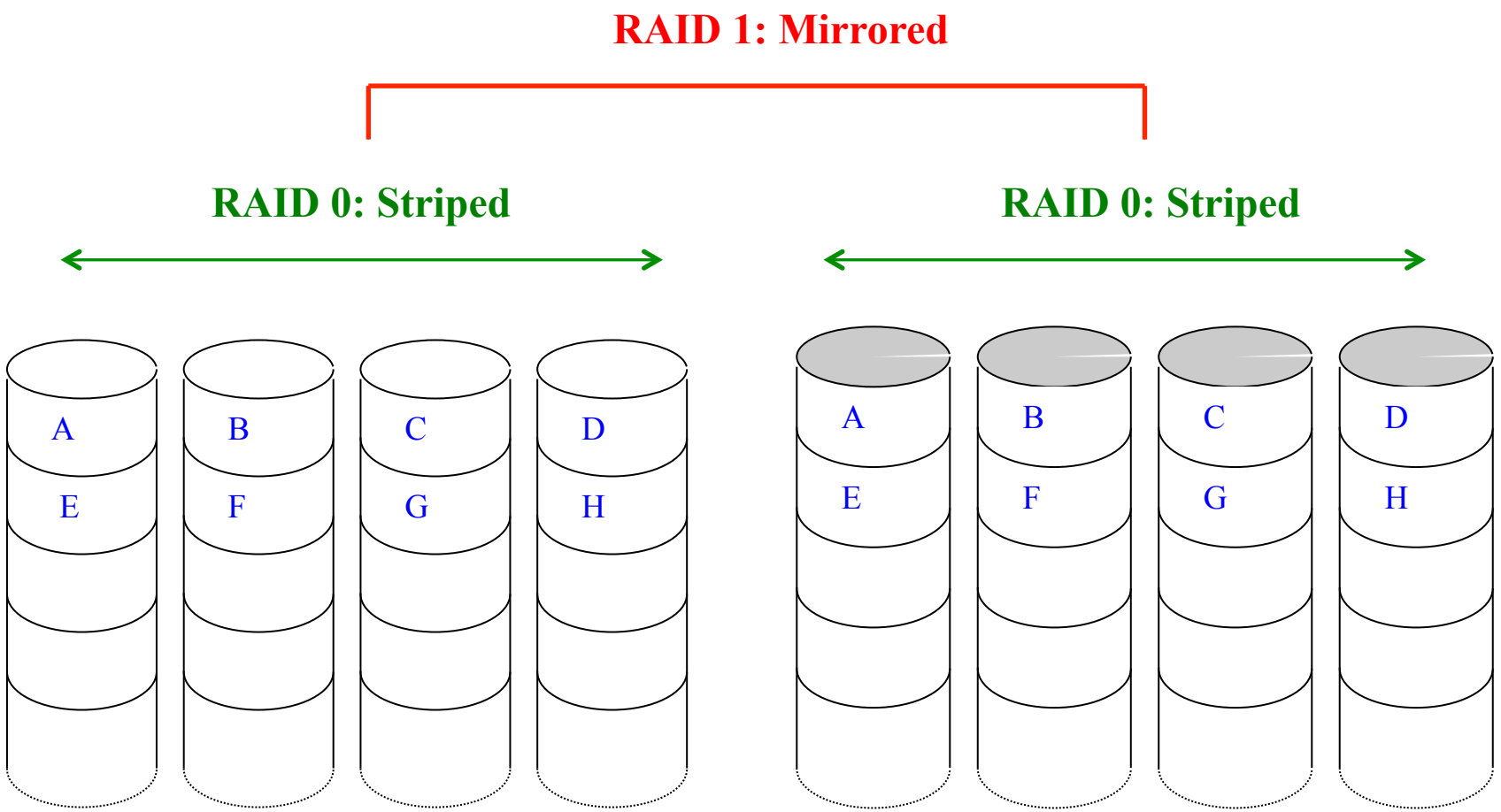
RAID Level 1 (mirrored)



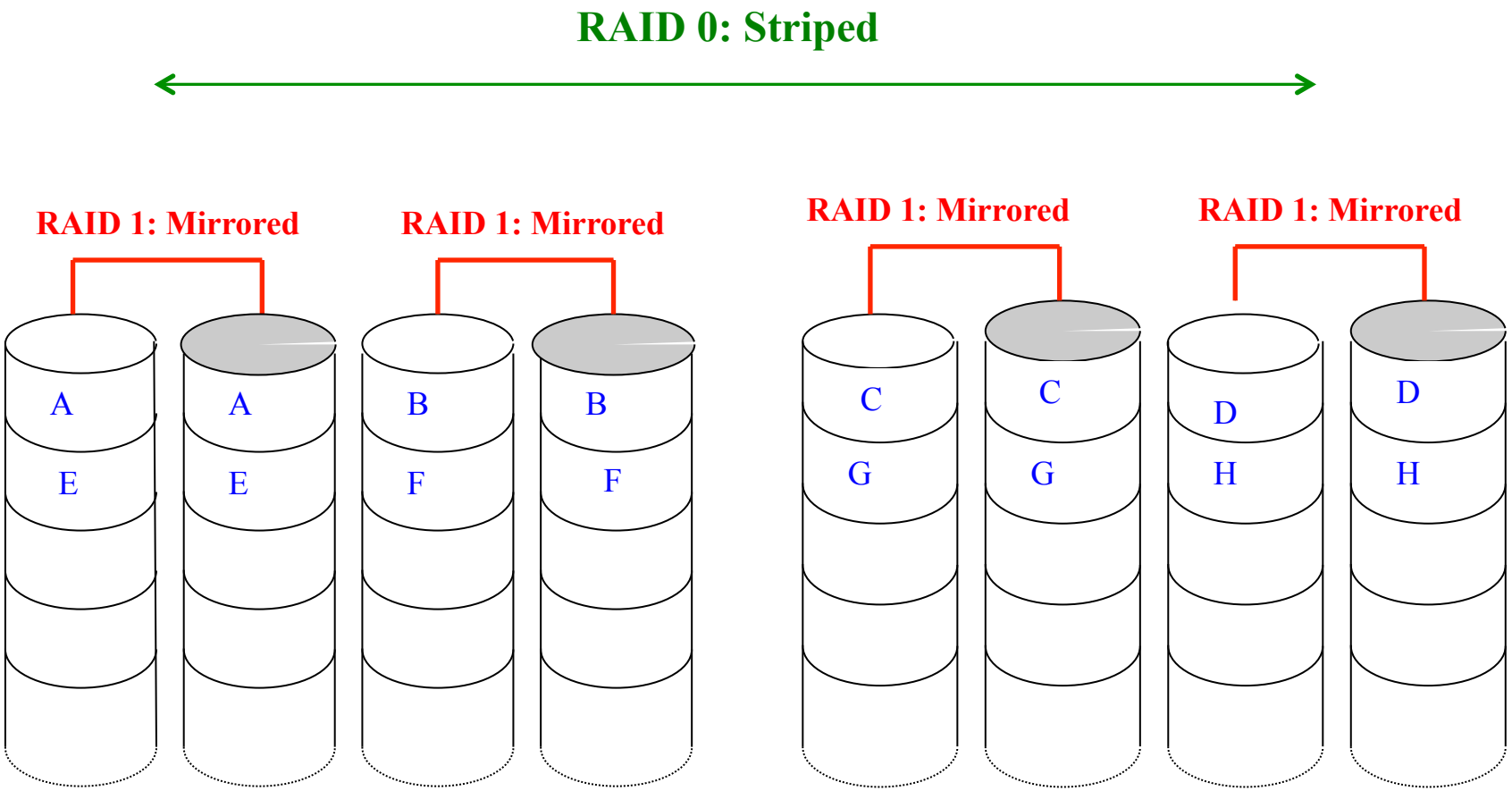
RAID Level 1 (mirrored)

- RAID 1 improves reliability by simply duplicating all the data.
  - It requires twice disk space of logical disk that it supports.
  - When one drive fails, the data may still be accessed from the second drive.
- RAID 0+1: A set of disks are striped, and then the stripe is mirrored to another equivalent stripe.
  - Read request can be serviced by either of the two disks.
  - A write request requires updating both corresponding strips.
  - One disk fail makes an entire stripe unavailable, leaving only the mirrored stripe
- RAID 1+0: Disks are mirrored in pairs and the resulting mirrored pairs are striped.
  - When one disk fails, its mirrored disk is still available, and also the remaining disks.

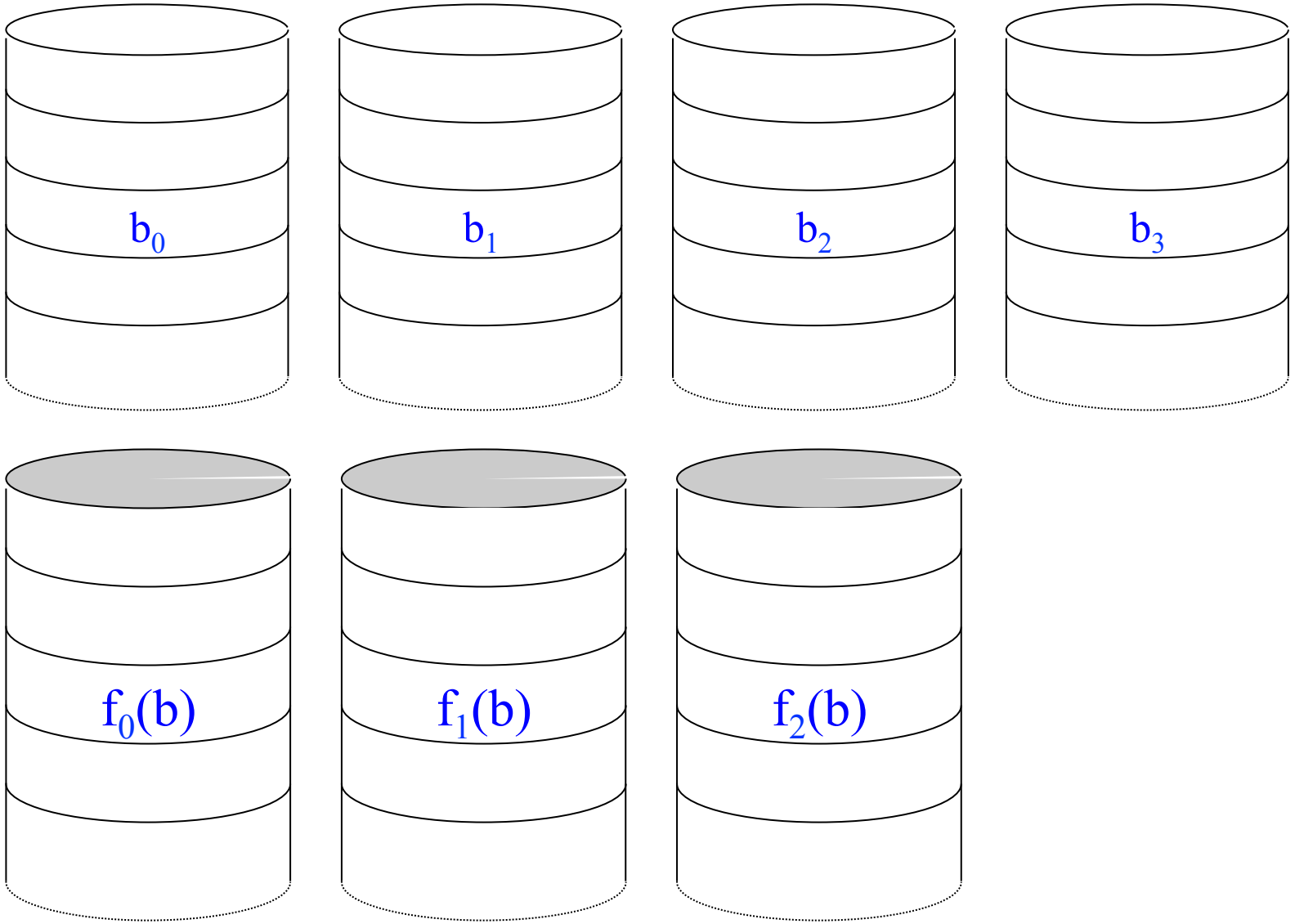
RAID 0+1


RAID 1: Mirrored
RAID 0: Striped
RAID 0: Striped
A
E
B
F
C
G
D
H
A
E
B
F
C
G
D
H

RAID 1+0


RAID 0: Striped
RAID 1: Mirrored
RAID 1: Mirrored
RAID 1: Mirrored
RAID 1: Mirrored
A
E
A
E
B
F
B
F
C
G
C
G
D
H
D
H

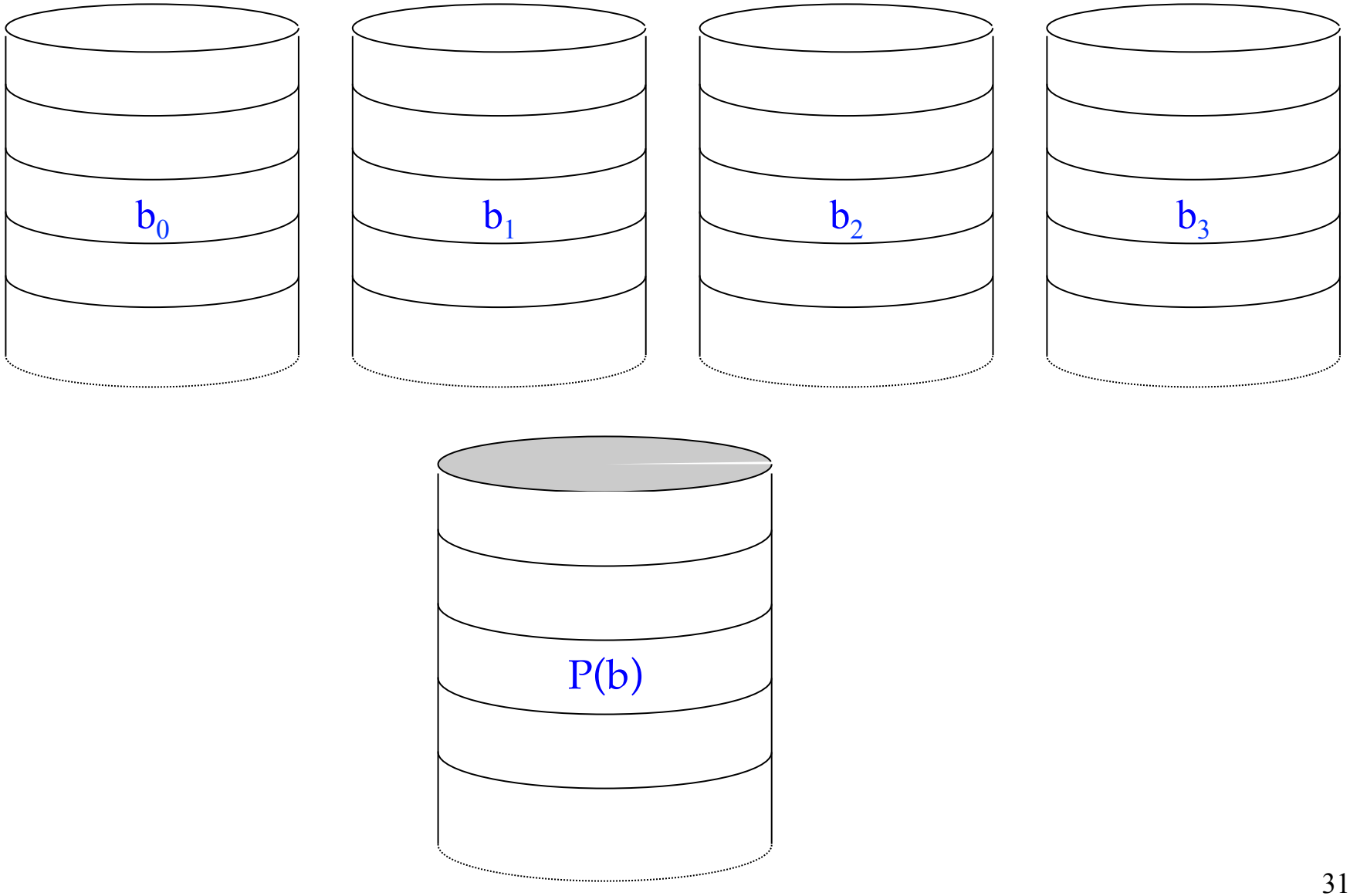
RAID Level 2



RAID Level 2 (cont.)

- Level 2 uses parallel access technique so that all disks participate in the execution of every I/O request.
- Strips are very small, often as small as a single byte or word.
- Error-correcting code is calculated across corresponding bits on each data disk.
  - Bits of the code are stored in the corresponding bit positions on multiple parity disks.
  - Typically, use Hamming code that can correct 1 error and detect 2 errors.
- The number of redundant disk is proportional to the *log* value of the number of data disks.
- RAID 2 should be used only if there are many disk errors.

RAID Level 3 (Bit-interleaved parity)



RAID Level 3 (Cont.)

- RAID 3 is similar to RAID 2, but RAID 3 requires only 1 redundant disk.
  - RAID 3 uses simple parity bit computed for the set of individual bits in the same position on all of the data disks.
  - RAID 3 employs parallel data access with data distributed in small strips.

Reliability

- When there is a drive failure, parity drive is accessed and data is reconstructed from the remaining devices.

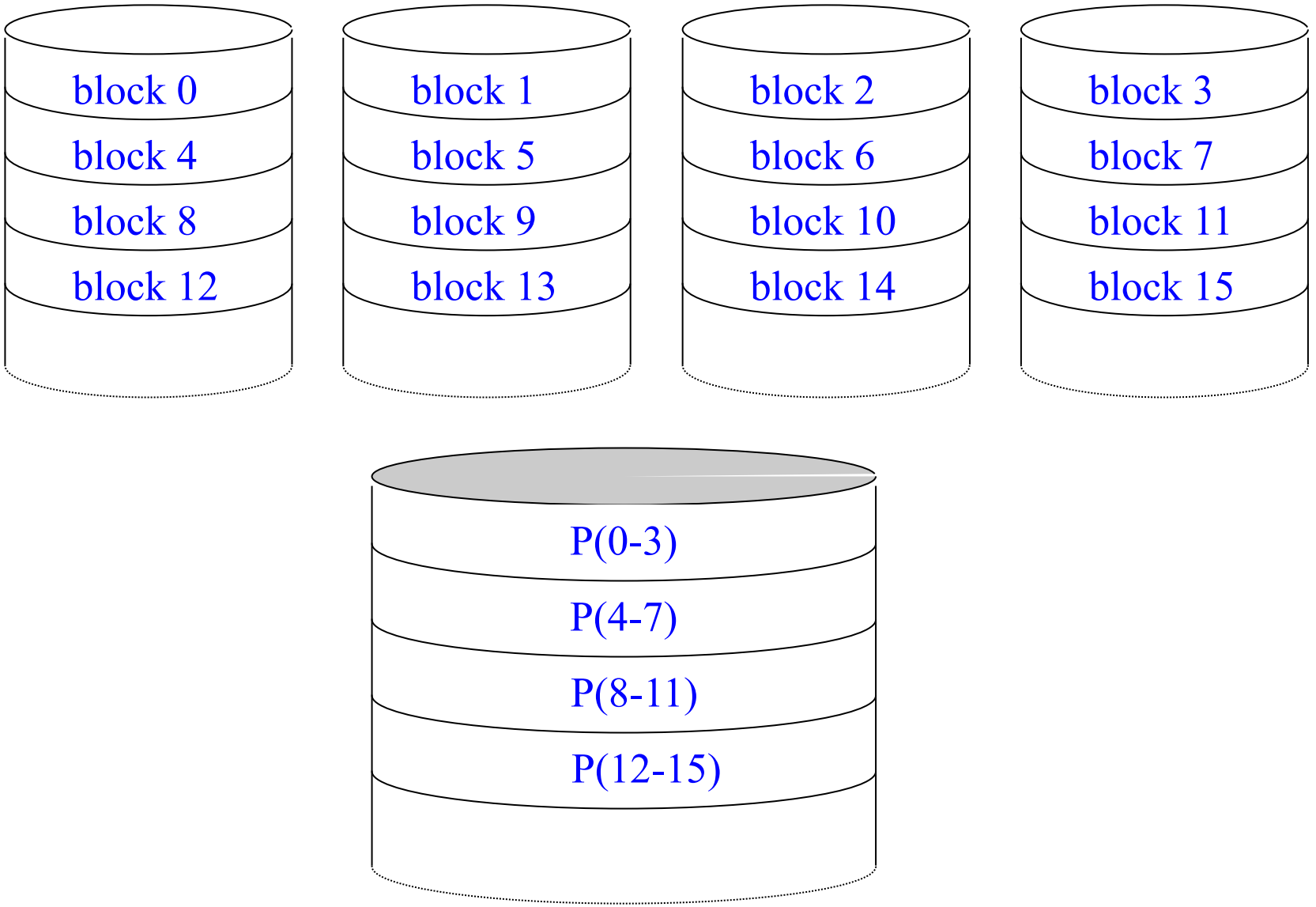
  Example - consider RAID 3 with 5 disks: D_0 to D_3 for data, and D_4 for parity.
  - Parity calculation: $D_4(i) = D_3(i) \oplus D_2(i) \oplus D_1(i) \oplus D_0(i)$
  - If drive D_1 fails, reconstruction is done as:

    $D_1(i) = D_4(i) \oplus D_3(i) \oplus D_2(i) \oplus D_0(i)$

Performance

- Data is stripped in very small strips: very high data transfer rates; good for large transfers.
- Only one I/O request can be executed at a time → does not improve I/O rate

RAID Level 4 (Block-level parity)



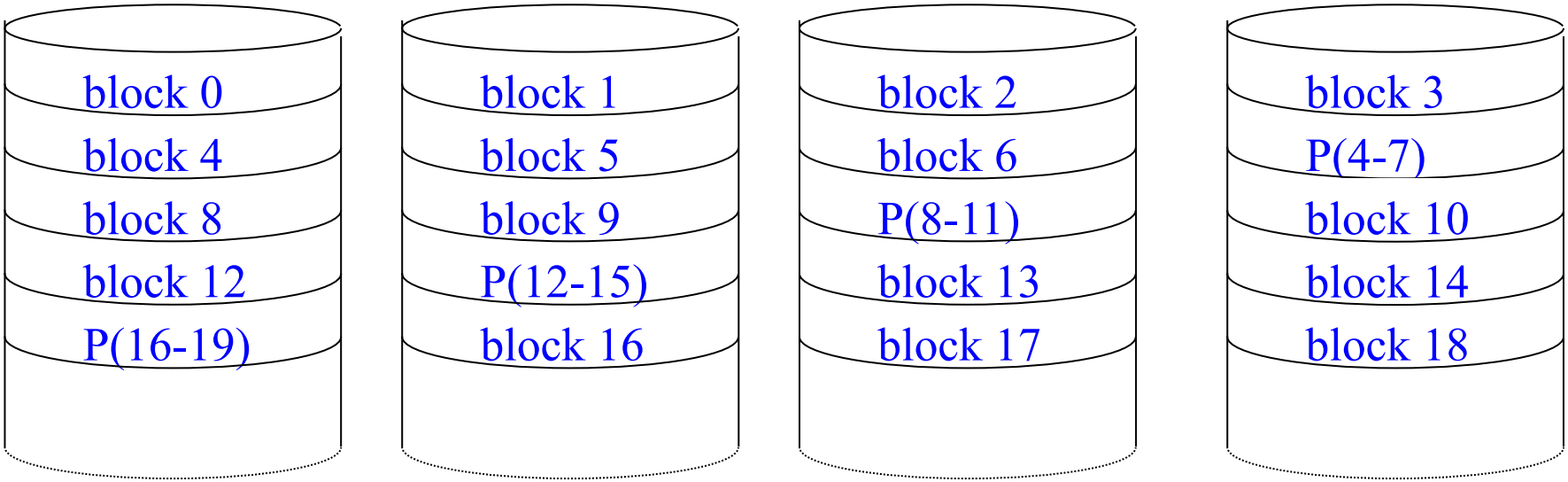
RAID Level 4 (Block-level parity)

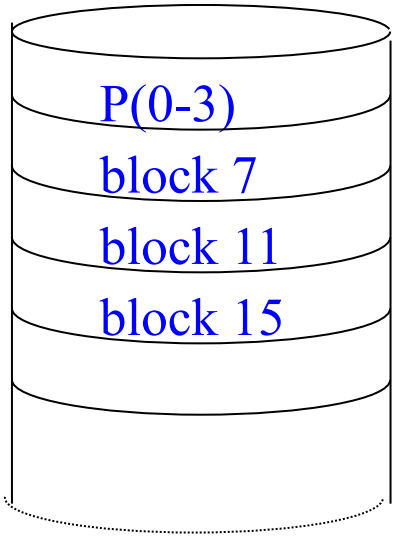
- RAID 4 uses relatively large data strips.
- RAID 4 uses independent access technique.
  - Each member disk operates independently, thus separate I/O request can be satisfied in parallel.
  - Good for applications with high I/O request rates; NOT for application with high data transfer rates.
- In RAID 4, parity bits are stored in a parity disk for reliability.

Performance

- For small data that does not involve all corresponding strips on all disks, a strip write involves 2 reads and 2 writes:
  - Read old data strip, and read old parity strip.
  - Write new data, and write new parity.
- For large size that involves all corresponding strips on all disks, write uses only the new data bits, and thus parity drives can be updated in parallel with data drives → NO extra reads or writes.

RAID Level 5 (Block-level distributed parity)





- RAID 5 is similar to RAID 4, but RAID 5 distributes the parity strips across all disks.
  - This avoids the potential I/O bottleneck in parity disk found in RAID 4.

RAID Level 6

- RAID 6 is similar to RAID 5
  - It stores extra redundant information to handle multiple disk failures
    - Use error correcting code instead of parity in RAID 5
    - Need 2 bits of redundant data for every 4 bits of data → more expensive than RAID 5
      - This can tolerate 2 disk failures
  - It is also called P + Q redundancy

I/O Hardware

- A computer operates many kinds of I/O devices.
  - Storage devices (disks, tapes).
  - Transmission devices (network cards, modems).
  - Human-interface devices (screen, keyboard, mouse).
  - The devices vary in functionalities, storage sizes and access speeds.
- OS needs to provide a wide range of functionality to applications.
  - Need a variety of methods to control the devices; these form the I/O subsystem of the kernel.
  - This subsystem separates the rest of the kernel from the complexity of managing I/O devices.

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 operation	
I/O direction	Read only Write only Read-write	CD-ROM Graphic controller Disk

I/O System Management

- Two main jobs of a computer: I/O and Processing.
  - For some cases, the main job is I/O, e.g., browsing a web, editing a file
- The role of an OS:
  - To manage and control I/O operations and I/O devices.
  - To provide the simplest interface possible to the rest of the system.
  - To optimize I/O for maximum concurrency.
- Devices vary widely, and thus we need to know:
  - How OS gives a convenient, uniform I/O interface to applications.
  - How to design OS so that new devices can be attached to the computer without rewriting OS.
    - To address the details and differences among the devices, OS uses device driver modules.
    - Device driver presents a uniform device-access interface to the I/O subsystem.
      - Similar to system calls that provide a standard interface between the application and OS

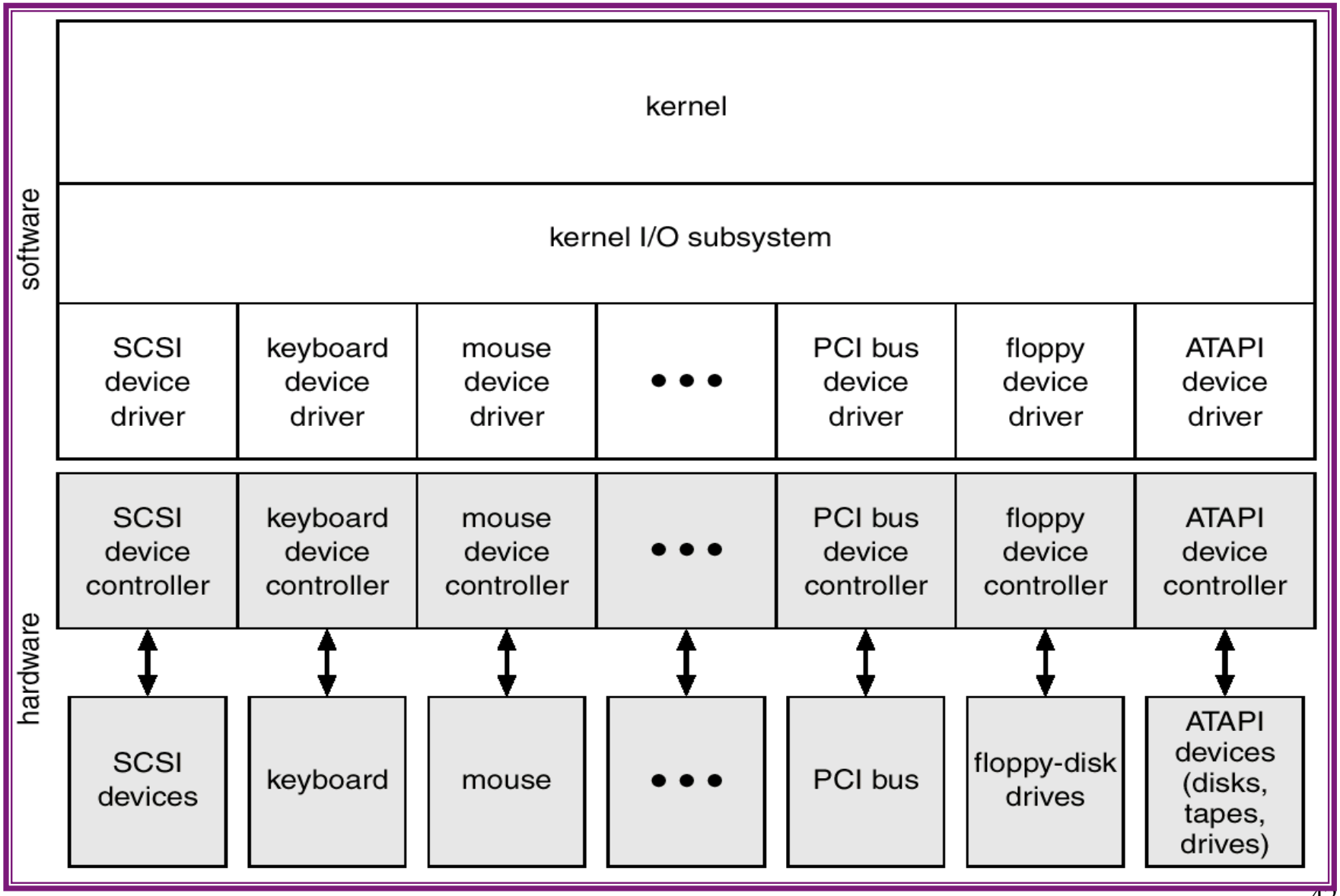
I/O Systems Management (cont.)

- One purpose of an OS is to hide the peculiarities of specific hardware devices from the user.
- I/O system consist of
  - Buffer-caching system.
  - General device-driver interface.
  - Drivers for specific hardware devices.
- Only the device drivers know the peculiarities of the specific device to which it is assigned.

Application I/O interface

- I/O system calls encapsulate device behaviors in generic ways.
- Device-driver layer hides the actual differences among I/O controllers from kernel.
- Each OS has its own standards for the device drivers interface
  - A device may have multiple device drivers, e.g., for Linux, Windows, etc.

Kernel I/O structure



Network Devices

- A *network* device differs significantly from block and character devices.
  - Therefore, it has its own interface.
- Unix and Windows include socket interface that separates network protocol from network operation.
- Approaches for network devices vary widely
  - E.g., Unix: pipes, FIFOs, streams, queues, mailboxes.

Blocking and Non-blocking I/O

- A system-call for I/O can be *blocking* or *non-blocking*.
- Blocking – process suspended until I/O completed.
  - Easy to use and understand.
  - Insufficient for some needs, e.g., keyboard and mouse.
- Non-blocking – I/O call returns as much data as available, without waiting for I/O to complete.
  - It returns quickly with count of bytes read or written.
  - It is implemented using multi-threading.
  - It is used in user interface (keyboard, mouse), data copy, etc.
- *Asynchronous* I/O – process runs while I/O executes.
  - This is an alternative to non-blocking I/O.
  - I/O subsystem signals process when I/O completed.

Kernel I/O Subsystem

- Kernel provides many services related to I/O:
  - I/O scheduling.
  - Buffering.
  - Caching.
  - Spooling.
  - Device reservation.
  - Error handling.

I/O scheduling

- Scheduling can be used:
  - To improve overall system performance.
  - To share device access fairly among processes.
  - To reduce average waiting time for I/O to complete.

Kernel I/O Subsystem (cont.)

Buffering

- Buffer is a memory area that stores data being transferred between two devices or between a device and an application.
- Buffering is done for 3 reasons:
  - To cope with device speed mismatch.
    - If the system is receiving a file from modem (slow) to store in a hard disk (fast).
  - To cope with device transfer size mismatch.
    - In computer network, buffers are for message fragmentation and reassembly.
  - To maintain *copy semantics*.
    - The version of data written to disk must be the version at the time of the application system call, NOT the most recent content of the application's buffer, if there is any changes.

Kernel I/O Subsystem (cont.)

Caching

- A cache is a part of fast memory to hold copies of data item.
- A cache is different from a buffer:
  - Buffer may hold the only existing copy of data item.
  - Cache holds a copy of data item on a faster storage (e.g., memory) that resides elsewhere (e.g., disk).
- Cache is important to improve the access time to the data item that is accessed frequently.

Kernel I/O Subsystem (cont.)

Spooling

- A spool is a buffer that holds output for a device (e.g., printer) that cannot accept interleaved data stream.
  - Each application's output is spooled to a separate disk file.
  - When an application finishes printing, the spooling system queues the corresponding spool file for output to the printer.
- OS provides a control interface that enables users and system administrators to display the queue, to remove unwanted jobs, suspend printing, etc.

Kernel I/O Subsystem (cont.)

Device reservation

- It provides exclusive access to a device.
- It uses system calls for device allocation and de-allocation.
- OS needs to handle possible deadlock.

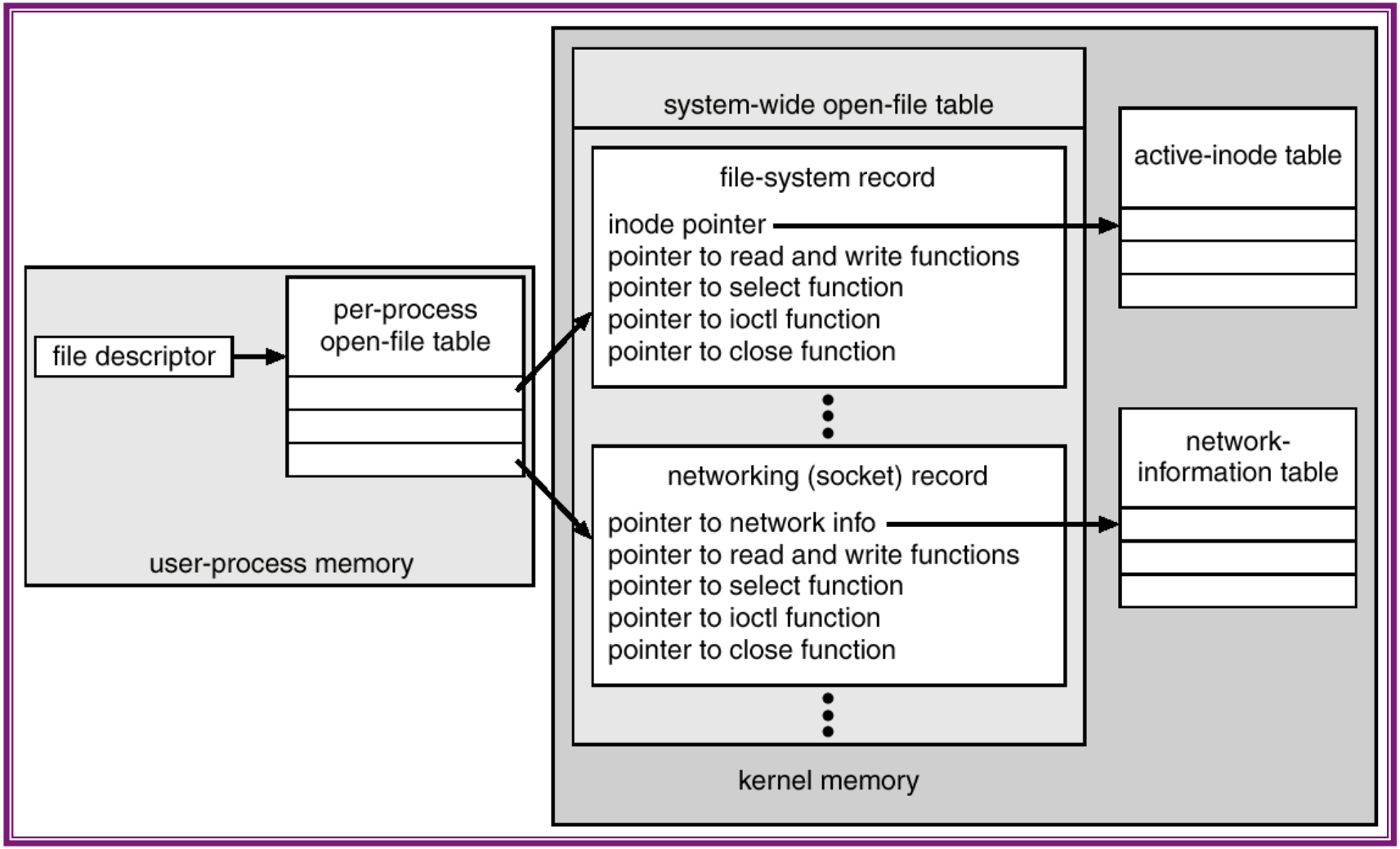
Error Handling

- Devices and I/O transfers can fail in many ways, either for transient reasons, or for permanent reasons.
- OS can recover from disk read, device unavailable, transient failures.
- Most OS returns an error number or code when I/O request fails.
  - E.g., Linux returns an integer value *errno*.

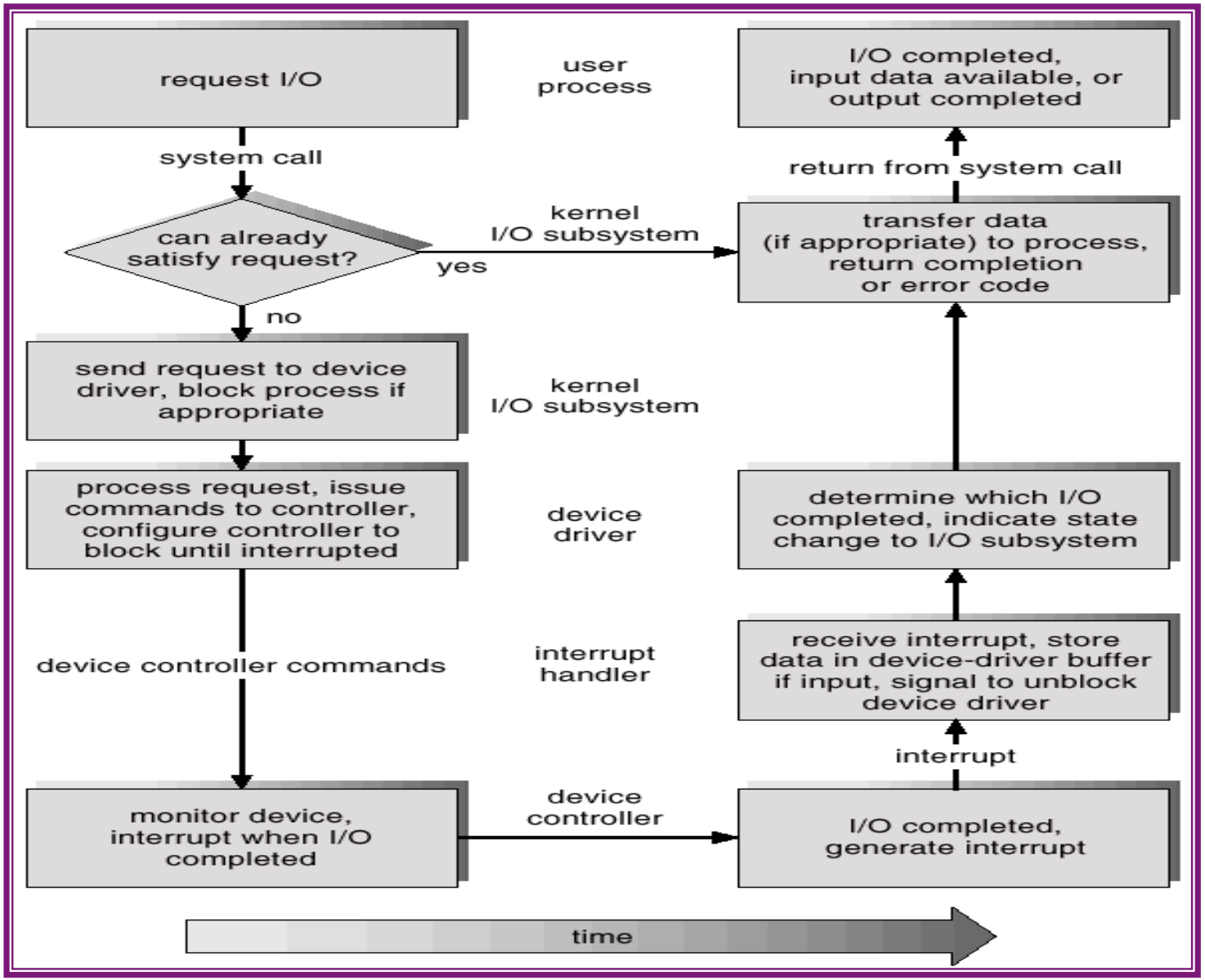
Kernel Data Structure

- Kernel keeps the state information about I/O components, including open file tables, network connections, character device state, etc.
- Many complex data structures are used to track buffers, memory allocation, dirty blocks, etc.
- Some OS uses object-oriented methods and message passing to implement I/O.

UNIX I/O Kernel Structure



Life Cycle of An I/O Request



I/O Performance

- I/O is a major factor in the overall system performance.
  - CPU frequently executes device-driver code.
  - OS must schedule processes fairly and efficiently as they block and unblock.
  - I/O involves many context switches due to interrupts and data copying.
  - I/O for computer network is especially very time consuming; it needs a lot of context switches

Improving performance

How to improve I/O performance?

- Reduce the total number of context switches.
- Reduce the number of times that data must be copied in memory while passing between device and application.
- Reduce the frequency of interrupts by using large transfers, smart controllers, polling (if busy waiting can be minimized).
- Increase concurrency by using DMA controllers or channels to offload simple data copying from the CPU.
- Move processing primitives into hardware, to allow their operation in device controllers concurrent with the CPU and bus operation.
- Balance CPU, memory subsystem, bus, and I/O performance, because overload in any one area may cause idleness in others.