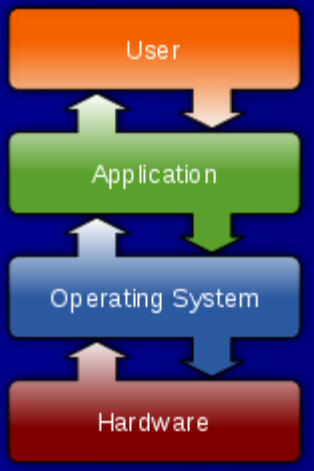
ENCE360 Operating Systems



Input/Output

MOS Ch 5

Outline

- Introduction ←
- Hardware
- Software
- Specific Devices

Introduction

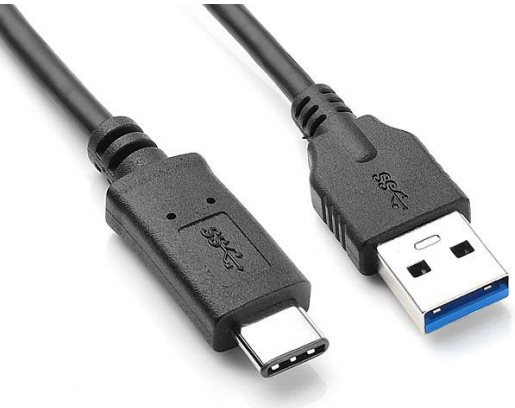
- One OS function is to control devices
  - significant fraction of code (80-90% of Linux)
- Want all devices to be simple to use
  - convenient
  - E.g: stdin/stdout, pipe, re-direct
- Want to optimize access to device
  - efficient
  - devices have very different needs

Input/Output Problems

- **Wide variety of peripherals**
  - Delivering different amounts of data
  - At different speeds
  - In different formats
- **All slower than CPU and RAM**
- Need I/O modules

Throughput	Type of Service
10 bps	keyboard
64 kbps	Integrated Services Digital
2.4 Mbps	Bluetooth 3.0
480 Mbps	USB 2.0
1 Gbps	Gigabit Ethernet
1 Gbps	IEEE 802.11ac WiFi
3.2 Gbps	FireWire 3.0
10 Gbps	USB 3.1
16 Gbps	eSATA 3.2
40 Gbps	USB-C Thunderbolt 3
31 GBps	PCI Express 4.0

USB-C



More Speed

40 Gbps

More Pixels

Two 4k

More Power

Up to 100w

More Protocols

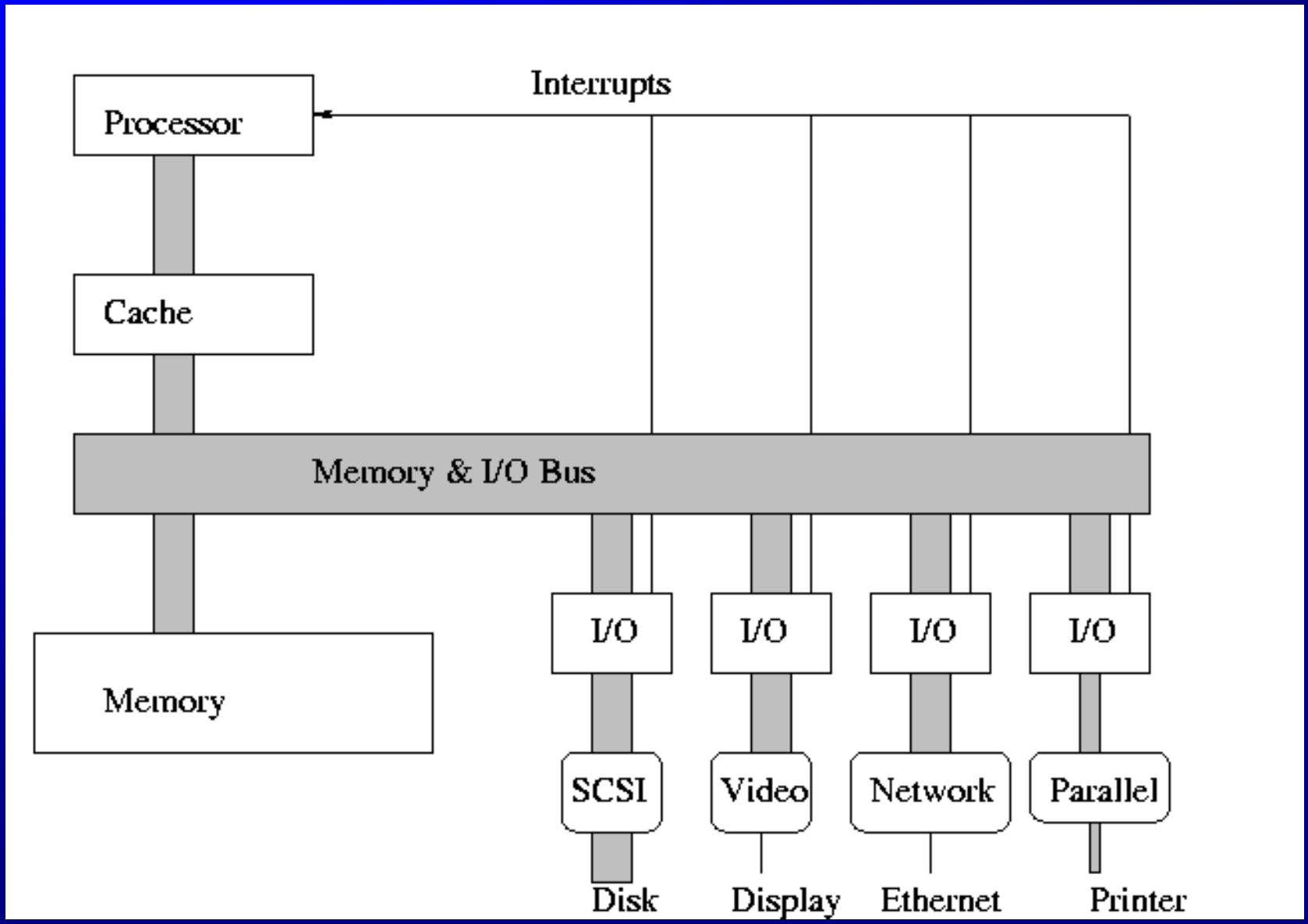
THUNDERBOLT.

SUPERSPEED USB

DisplayPort

PCI EXPRESS

Generic System Architecture



devices are getting faster

GPUs: Nvidia GeForce GTX TITAN Z

- 5,760 CUDA cores
  - Equivalent to ~100 CPUs
- 12 GB of GDDR5 memory
  - I/O speed can dominate

Outline

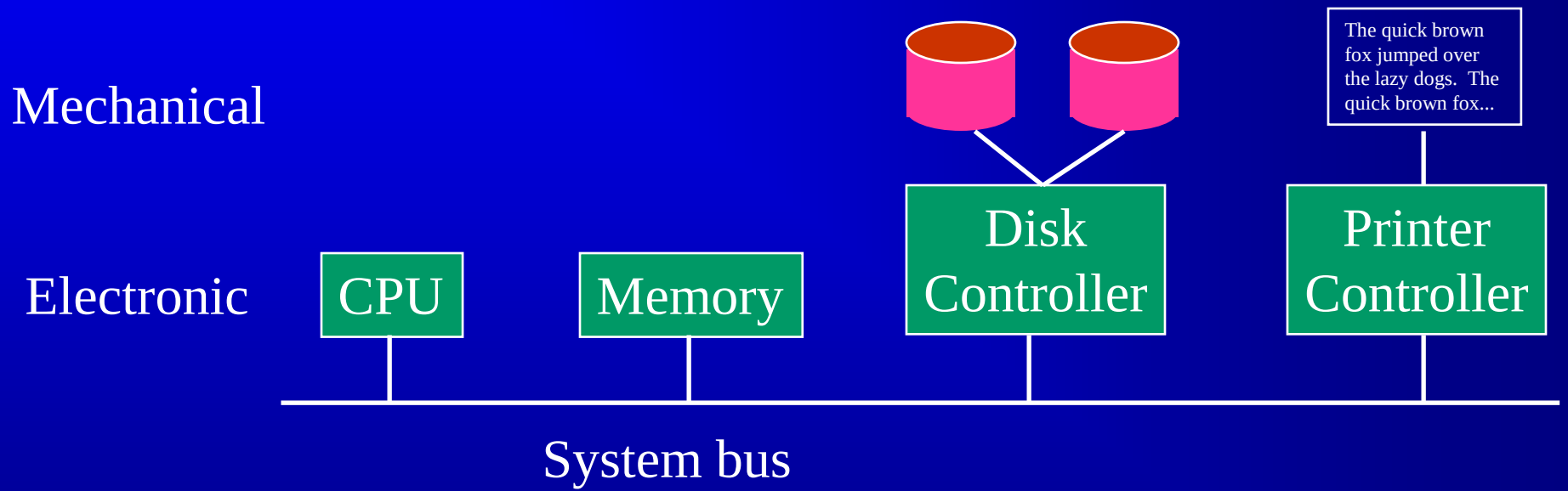
- Introduction (done)
- **Hardware** ←
- Software
- Specific Devices

Hardware

- Device controllers
- Types of I/O devices
- Direct Memory Access (DMA)

Device Controllers

- Mechanical and electronic component



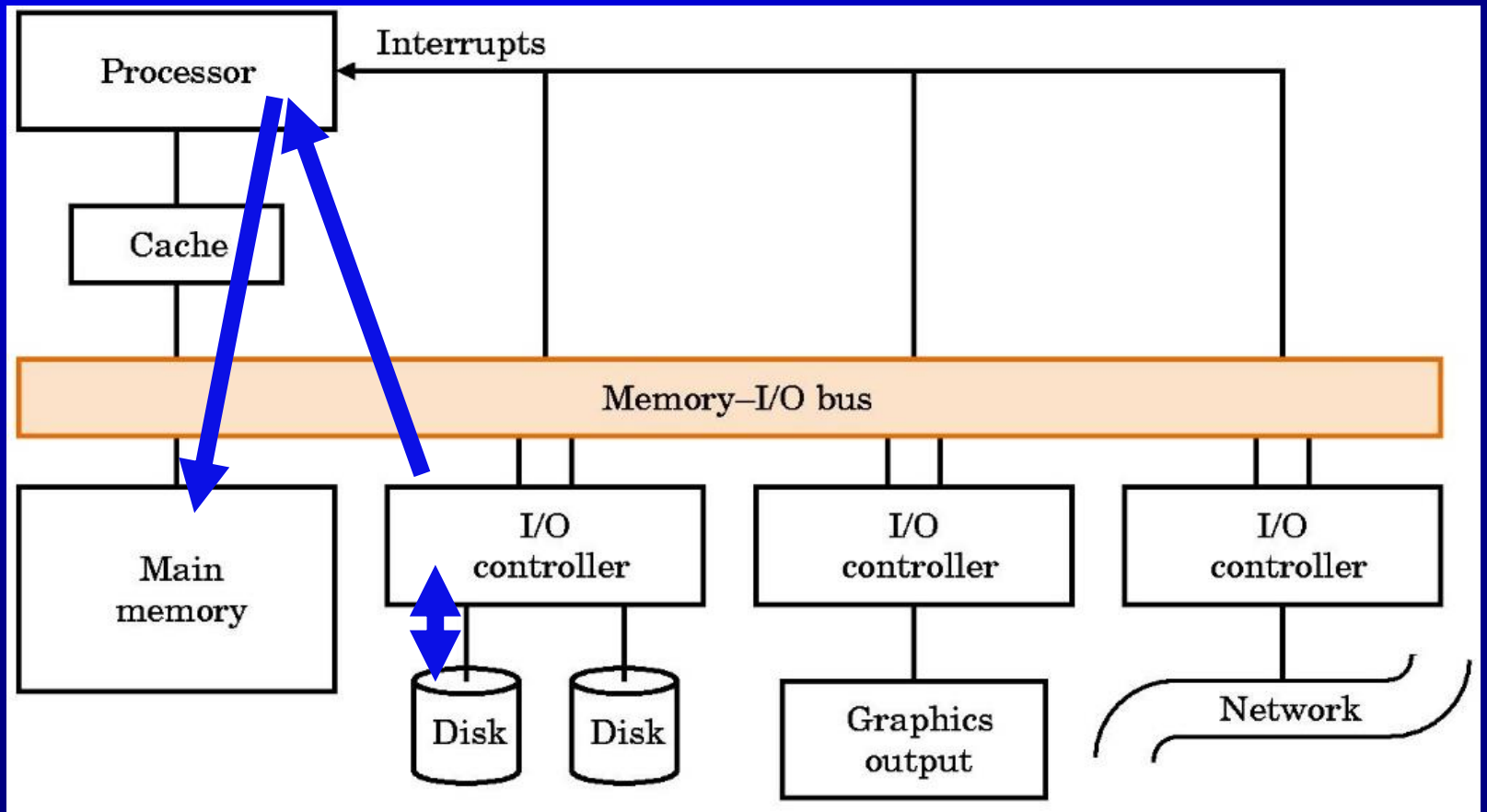
- OS deals with electronic
  - device controller

I/O Device Types

- block - access is independent
  - ex- disk
- character - access is serial
  - ex- printer, network
- other
  - ex- clocks (just generate interrupts)

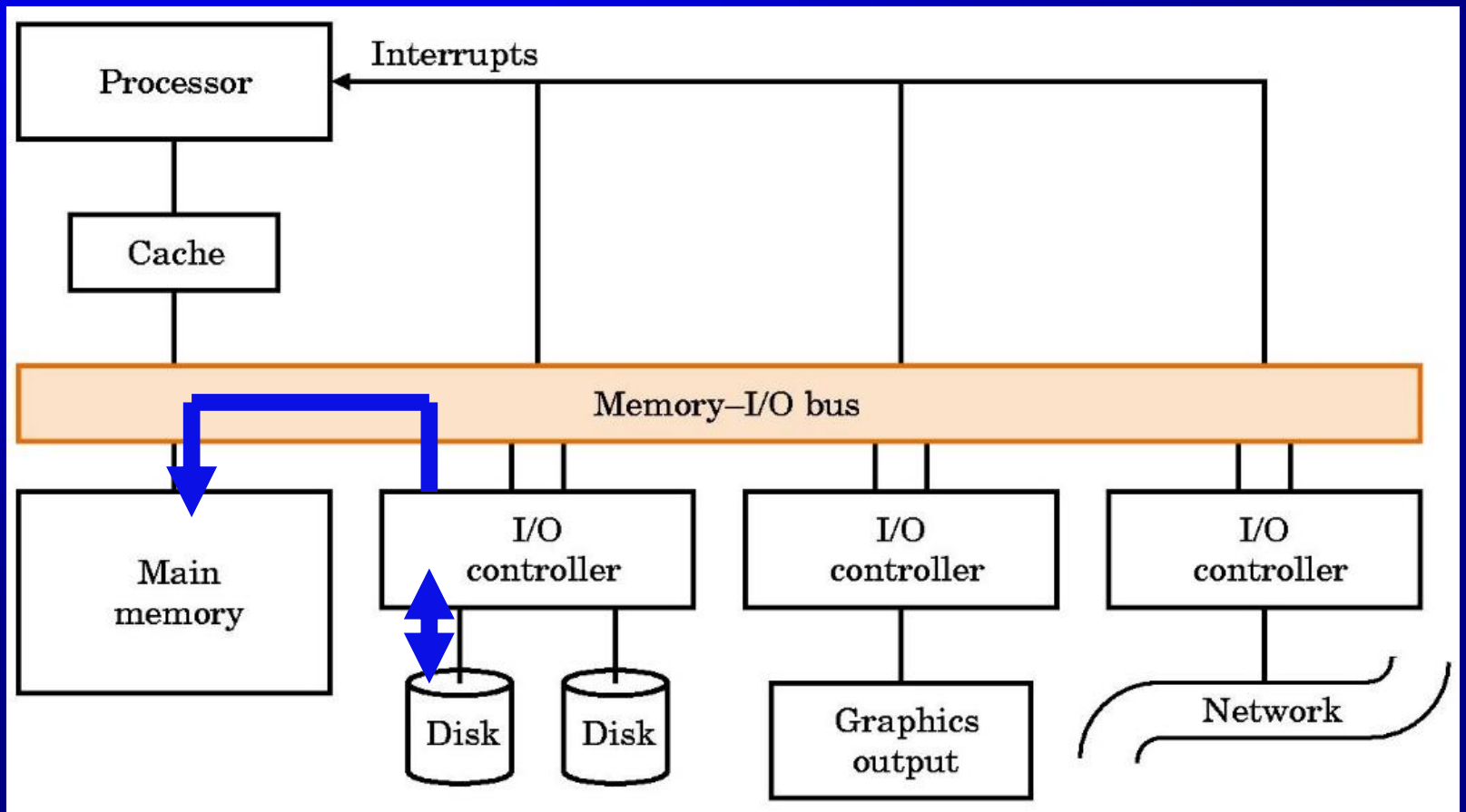
Data transfer between I/O and memory

- Simplest approach is for the CPU to perform the data transfer through load and store operations (assuming memory-mapped I/O)
  - CPU determines either through polling or interrupts that the transfer is done and another can occur



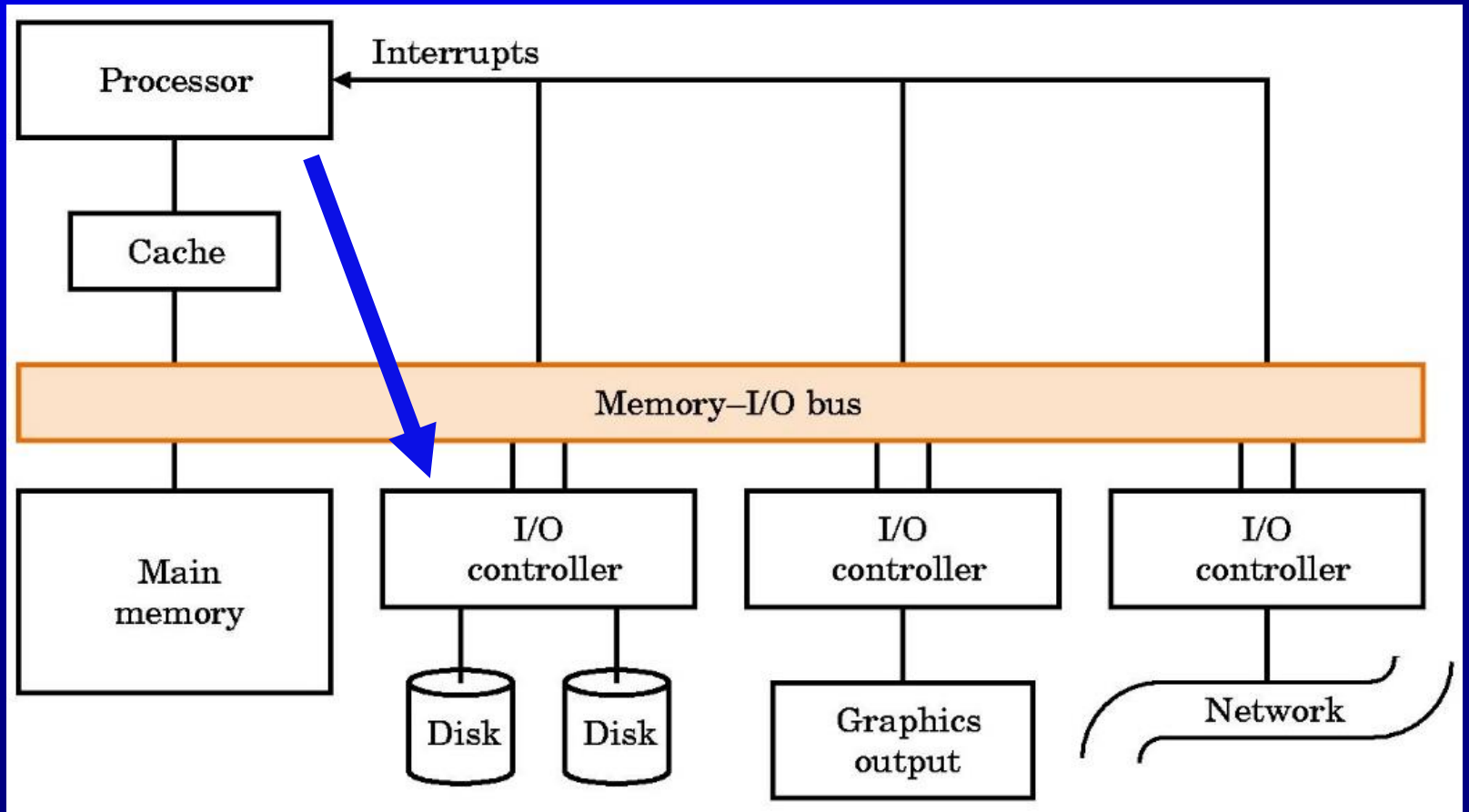
Data transfer between I/O and memory

- A common higher performance alternative is *direct memory access (DMA)* in which the device directly transfers data to/from memory without CPU intervention



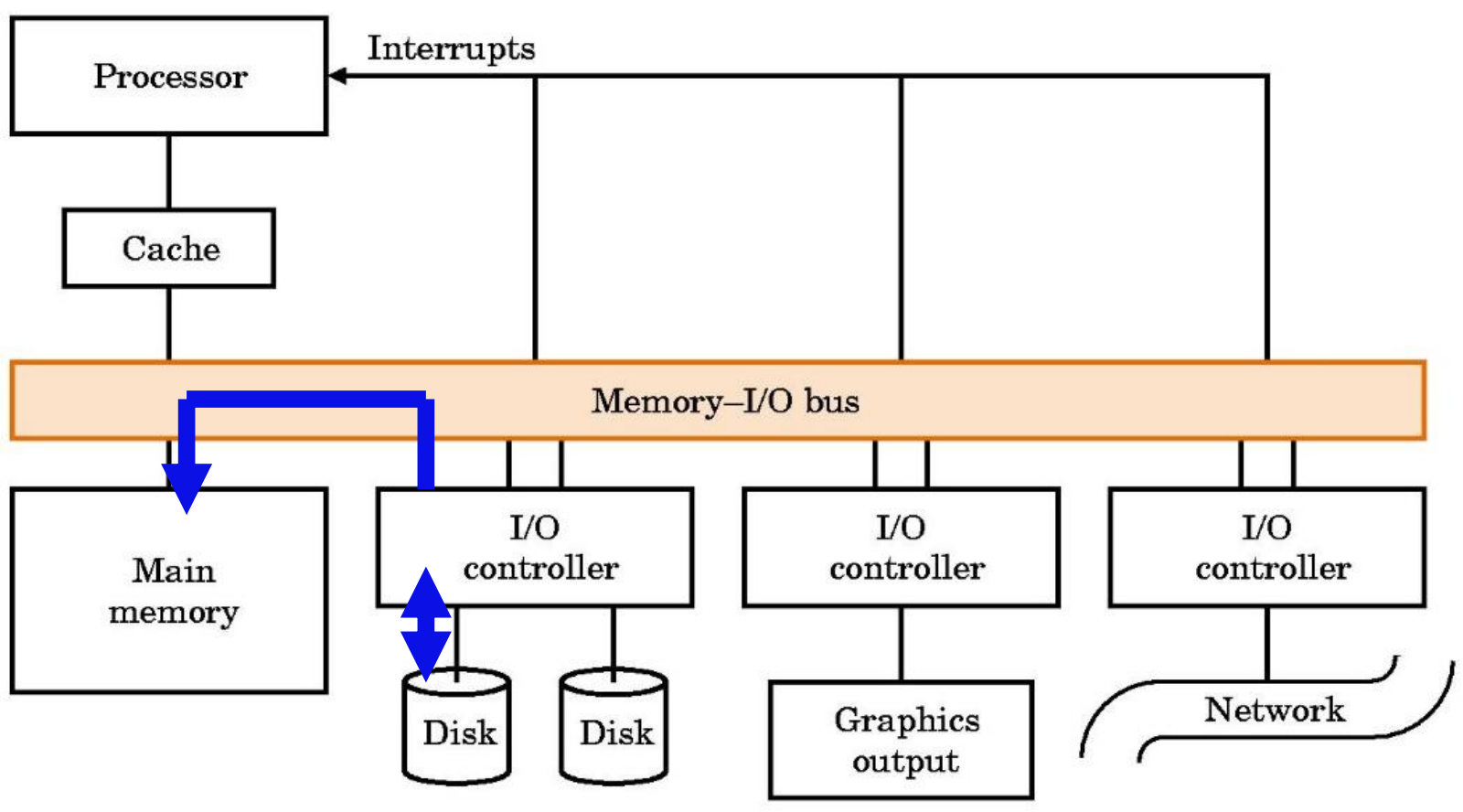
DMA operation

- Processor provides to the DMA device the number of bytes to transfer, the memory address, and the operation to be performed



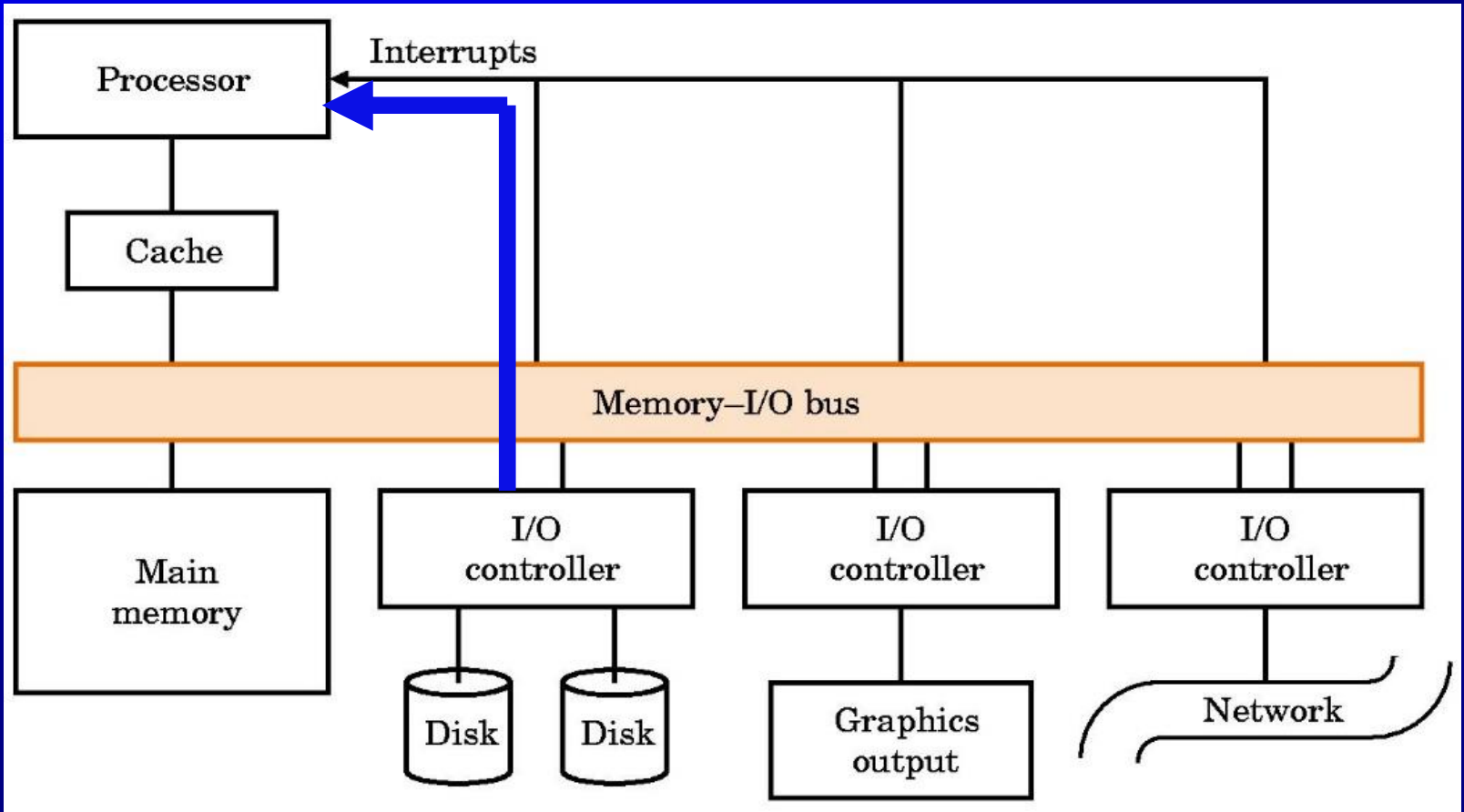
DMA operation

- The device performs the transfer over the memory-I/O bus to or from memory



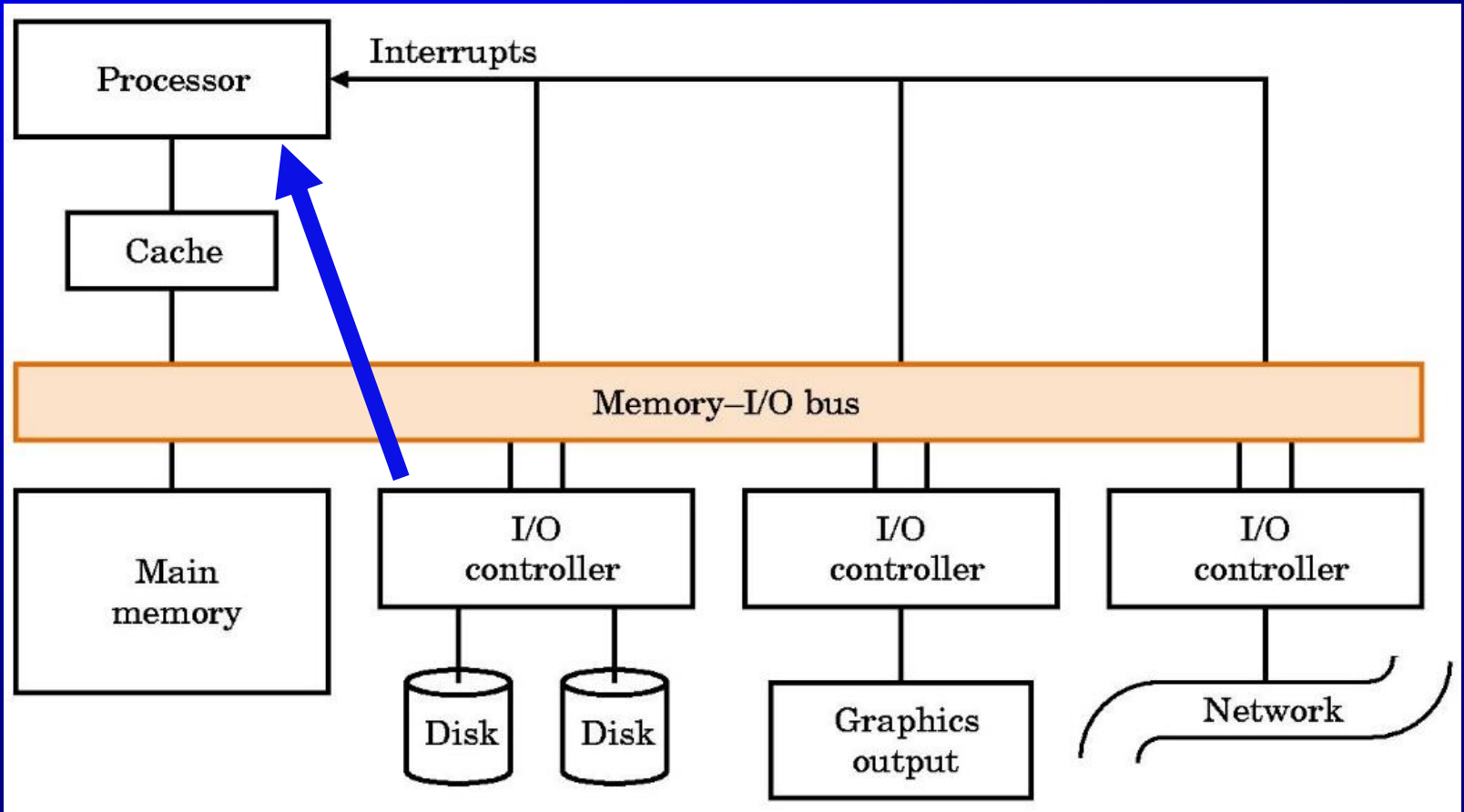
DMA operation

- The device interrupts the processor



DMA operation

- The processor reads the status registers to determine if the operation completed without errors



Direct Memory Access

- Interrupt driven and programmed I/O require active CPU intervention
  - **Transfer rate is limited**
  - **CPU is tied up**
- Direct Memory Access (DMA) controller is the answer

DMA Function

- DMA controller takes over from CPU for I/O
- Additional Module (hardware) on bus

DMA Operation

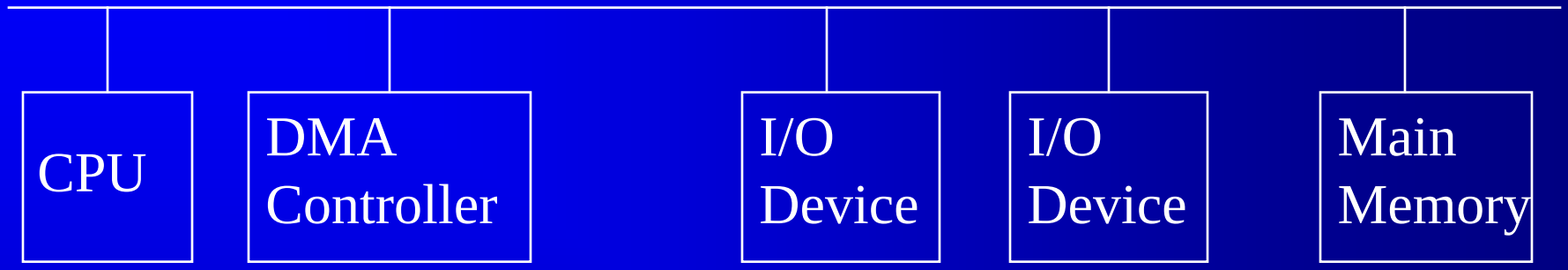
- CPU tells DMA controller:-
  - Read/Write
  - Device address
  - Starting address of memory block for data
  - Amount of data to be transferred
- CPU carries on with other work
- DMA controller deals with transfer
- DMA controller sends interrupt when finished

DMA Transfer
Cycle Stealing

- DMA controller takes over bus for a cycle
- Transfer of one word of data
- Not an interrupt
  - CPU does not switch context
- CPU suspended just before it accesses bus
  - i.e. before an operand or data fetch or a data write
- Slows down CPU but not as much as CPU doing transfer

DMA Configurations (1)

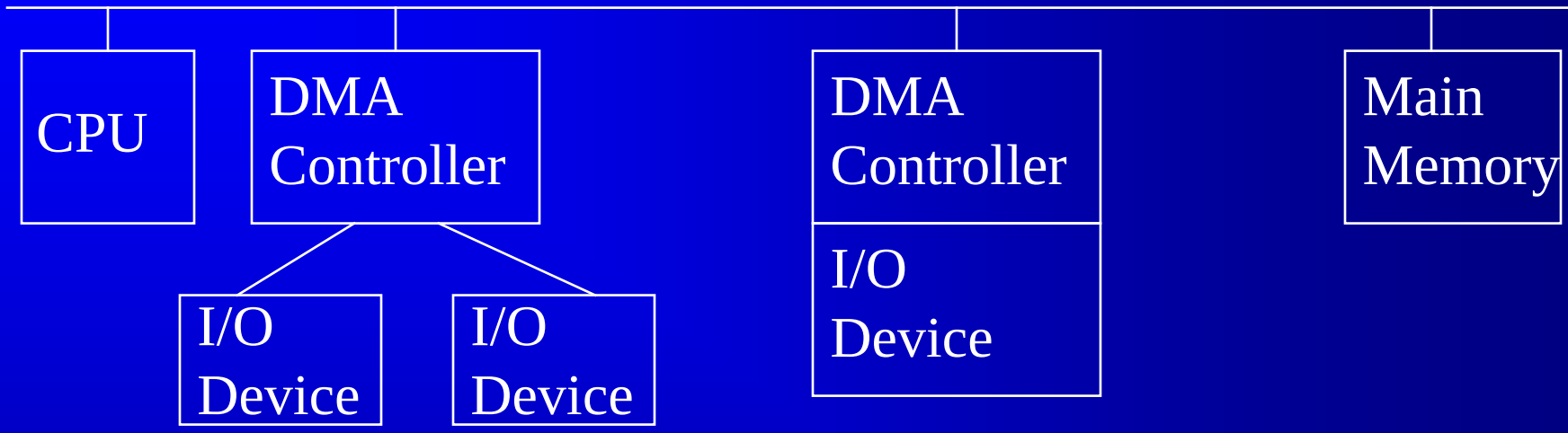
DMA shares bus with I/O units



- Single Bus, Detached DMA controller
- Each transfer uses bus twice
  - **I/O → DMA → memory**
- CPU is suspended twice

DMA Configurations (2)

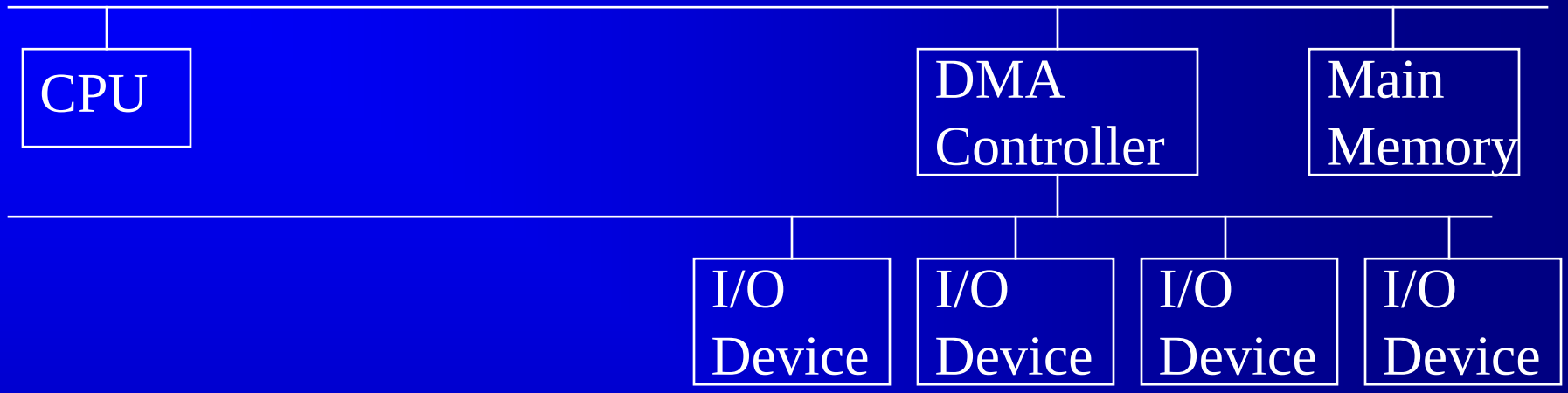
DMA shares bus only with other DMA, CPU, memory



- **Single** Bus, **Integrated** DMA controller
- Controller may support >1 device
- Each transfer uses bus **once**
  - **DMA → memory**
- **CPU is suspended once**

DMA Configurations (3)

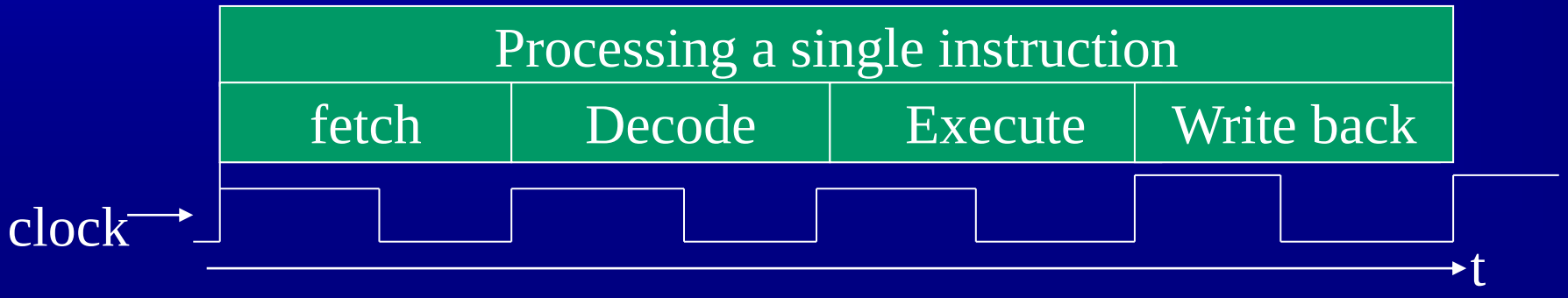
DMA has a separate I/O bus



- Separate I/O Bus
- Bus supports all DMA enabled devices
- Each transfer uses bus once
  - **DMA → memory**
- CPU is suspended once

DMA differs from an interrupt

- DMA can interrupt the processor in the middle of an instruction (before fetch, after decoding, after execution)
- It does not interfere with the running program
- The CPU is idled (waiting for access to the bus)
- The system is just becoming slower



DMA

- driver operation to input sequence of chars

```
write_reg(mm_buf, m);
write_reg(count, n);
write_reg(opcode, read);
block to wait for interrupt;
```

  - writing opcode triggers DMA controller
  - DMA controller issues interrupt after n chars in memory
- I/O processor (channel)
  - extended DMA controller
  - executes I/O program in own memory

Direct Memory Access (DMA)

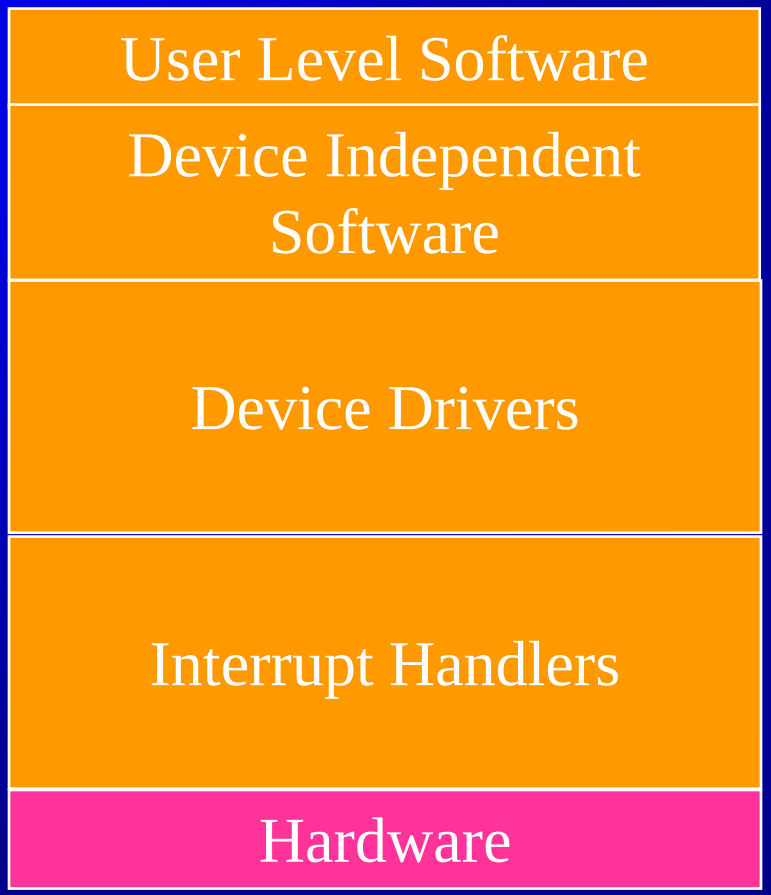
- Very Old
  - Controller reads from device
  - OS polls controller for data
- Old
  - Controller reads from device
  - Controller interrupts OS
  - OS copies data to memory
- DMA
  - Controller reads from device
  - Controller copies data to memory
  - Controller interrupts OS

Outline

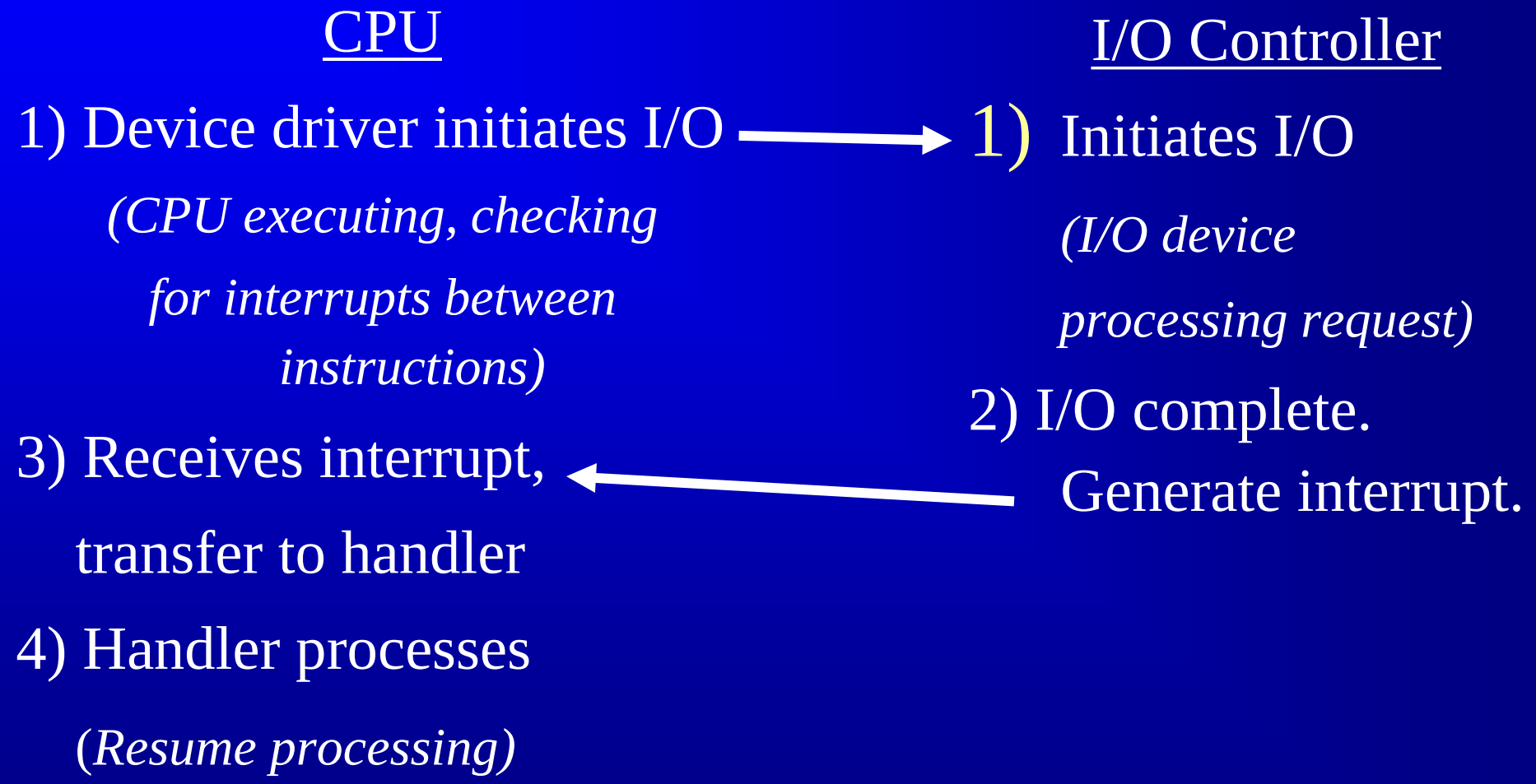
- Introduction (done)
- Hardware (done)
- **Software** ←
- Specific Devices

I/O Software Structure

- Layered



Interrupt Handlers



Interrupt Handler

- Make interrupt handler as small as possible
  - interrupts disabled
  - **Split into two pieces**
- **First part** does minimal amount of work
  - defer rest until later in the rest of the device driver
  - "**top-half**" handler
- **Second part** does most of work
  - Windows: "deferred procedure call" (DPC)
  - Linux: "**bottom-half**" handler

- Implementation specific
  - 3rd party vendors

Interrupt Handler

- Within top half Interrupt Handler, all Interrupts are Blocked
- Interrupt Handlers must finish up quickly so as not to keep interrupts blocked for long
- Top half: the function actually responds to the interrupt – within the Interrupt Handler
- Bottom-half (Linux) / DPC (Windows): a routine that is deferred by the top half to be executed later, at a safer time - during bottom-half, all interrupts are enabled

Device Drivers

- Device dependent code
  - includes interrupt handler
- Accept abstract requests
  - ex: “read block `n`”
- See that they are executed by device hardware
  - registers
  - hardware commands
- After error check
  - pass data to device-independent software

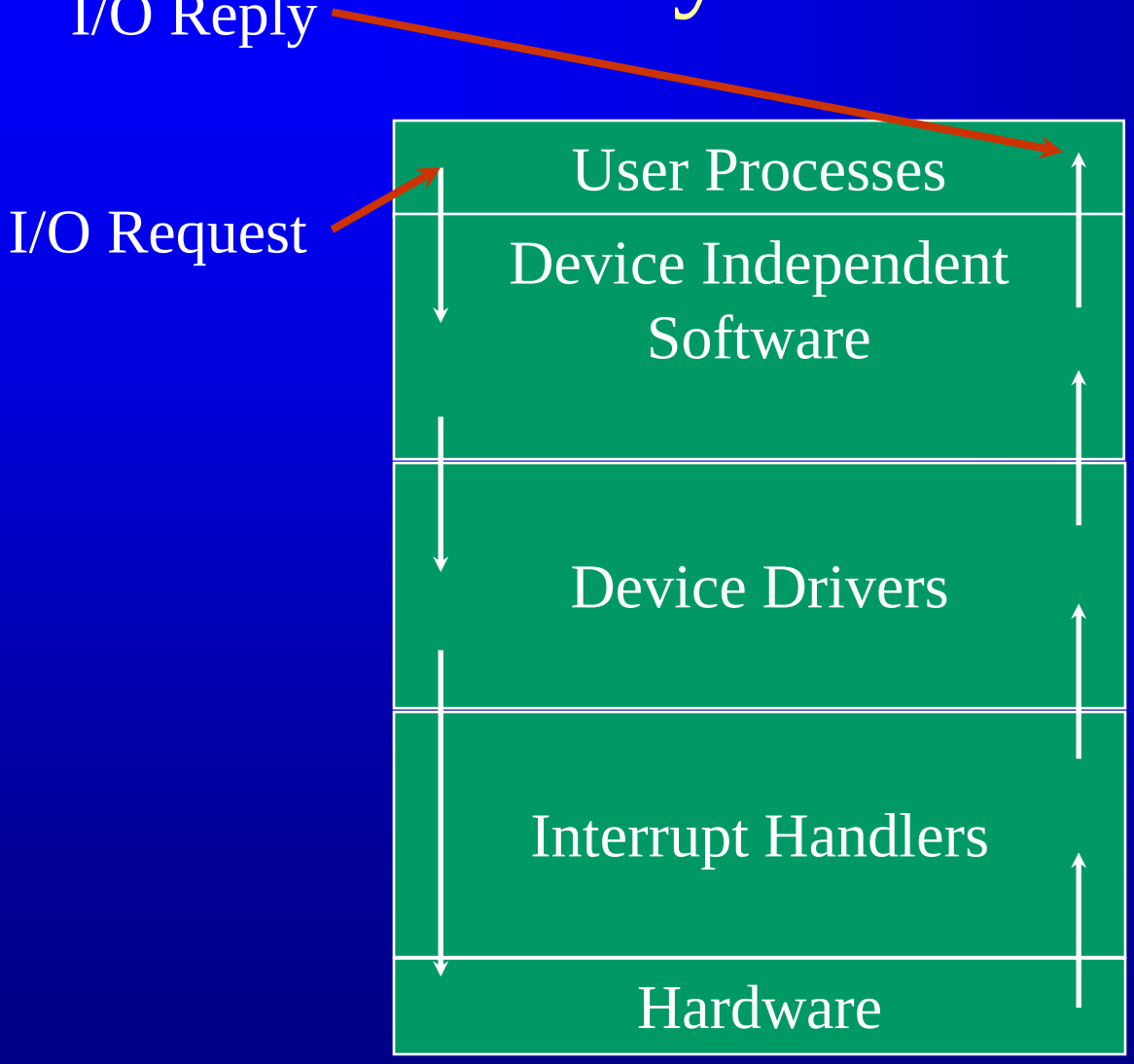
Device-Independent I/O Software

- Much driver code independent of device
- Exact boundary is system-dependent
  - sometimes inside for efficiency
- Perform I/O functions common to all devices
- Examples:
  - naming protection block size
  - buffering storage allocation error reporting

User-Space I/O Software

- Ex: `count = write(fd, buffer, bytes);`
- Put parameters in place for system call
- Can do more: formatting
  - `printf()`, `gets()`
- Spooling
  - spool directory
  - daemon (program or process that sits idly in the background until it is invoked to perform its task - from Greek mythology meaning "guardian spirit")
  - e.g.: print spoolers, e-mail handlers, schedulers

I/O System Summary



Make I/O call; Format I/O; Spooling

Naming, protection, blocking, buffering, allocation

Setup device registers; check status

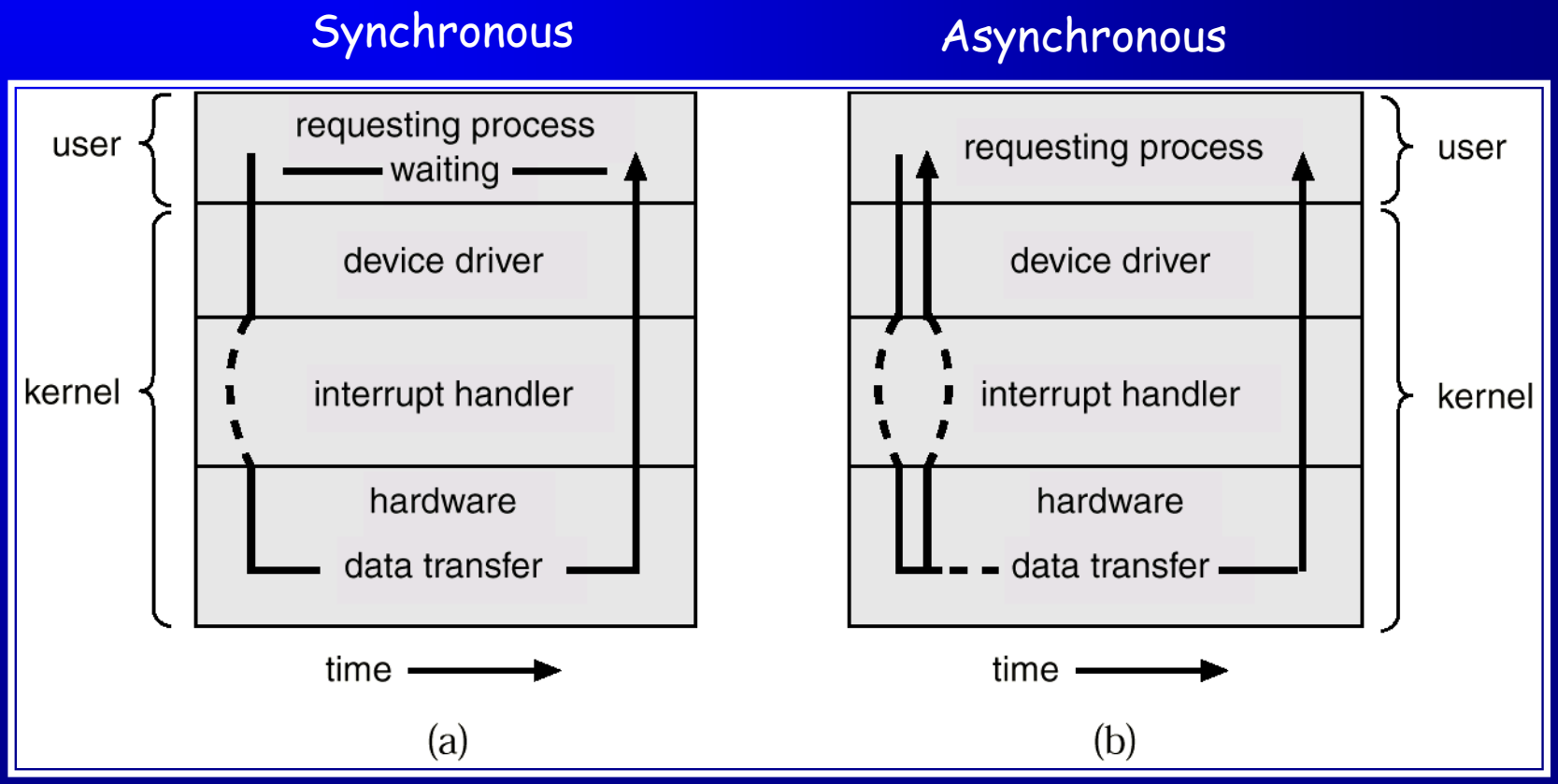
Wakeup driver when I/O completed

Perform I/O operation

I/O Structure: Syn vs. Asyn

- **Synchronous:** After I/O starts, control returns to user program only upon I/O completion.
  - Wait instruction idles the CPU until the next interrupt
  - Wait loop (contention for memory access).
  - At most one I/O request is outstanding at a time, no simultaneous I/O processing.
- **Asynchronous:** After I/O starts, control returns to user program without waiting for I/O completion.
  - *System call* – request to the operating system to allow user to wait for I/O completion.
  - *Device-status table* contains entry for each I/O device indicating its type, address, and state.
  - Operating system indexes into I/O device table to determine device status and to modify table entry to include interrupt. **(Most OS work under this mode.)**

Two I/O Methods



Generic Device I/O

- When a device asserts an interrupt:
  - The system reads the IRQ (Interrupt ReQuest) lines to determine what type of device handler (driver) to invoke.
  - The device driver scans the I/O registers of all devices of that type to determine which unit has the interrupt bit set.
  - The device driver uses other information in the device registers to perform an I/O operation (read/write/…).
  - When the I/O operation completes, the device driver return control to the kernel.

IRQs in the PC EISA (not shared) environment (c.f. PCI – shared):

- IRQ0 = interval timer
- IRQ1 = **keyboard**
- IRQ2 = reserved
- IRQ3 = COM2 and COM4
- IRQ4 = COM1 and COM3
- IRQ5 = LPT
- IRQ6 = Floppy drives
- IRQ7 = LPT
- IRQ8 = Real-time clock (RTC)
- IRQ9 = **Video card**
- IRQ10 - 12 = reserved
- IRQ13 = Math **coprocessor**
- IRQ14 = Primary **IDE** drives
- IRQ15 = Secondary **IDE** drives

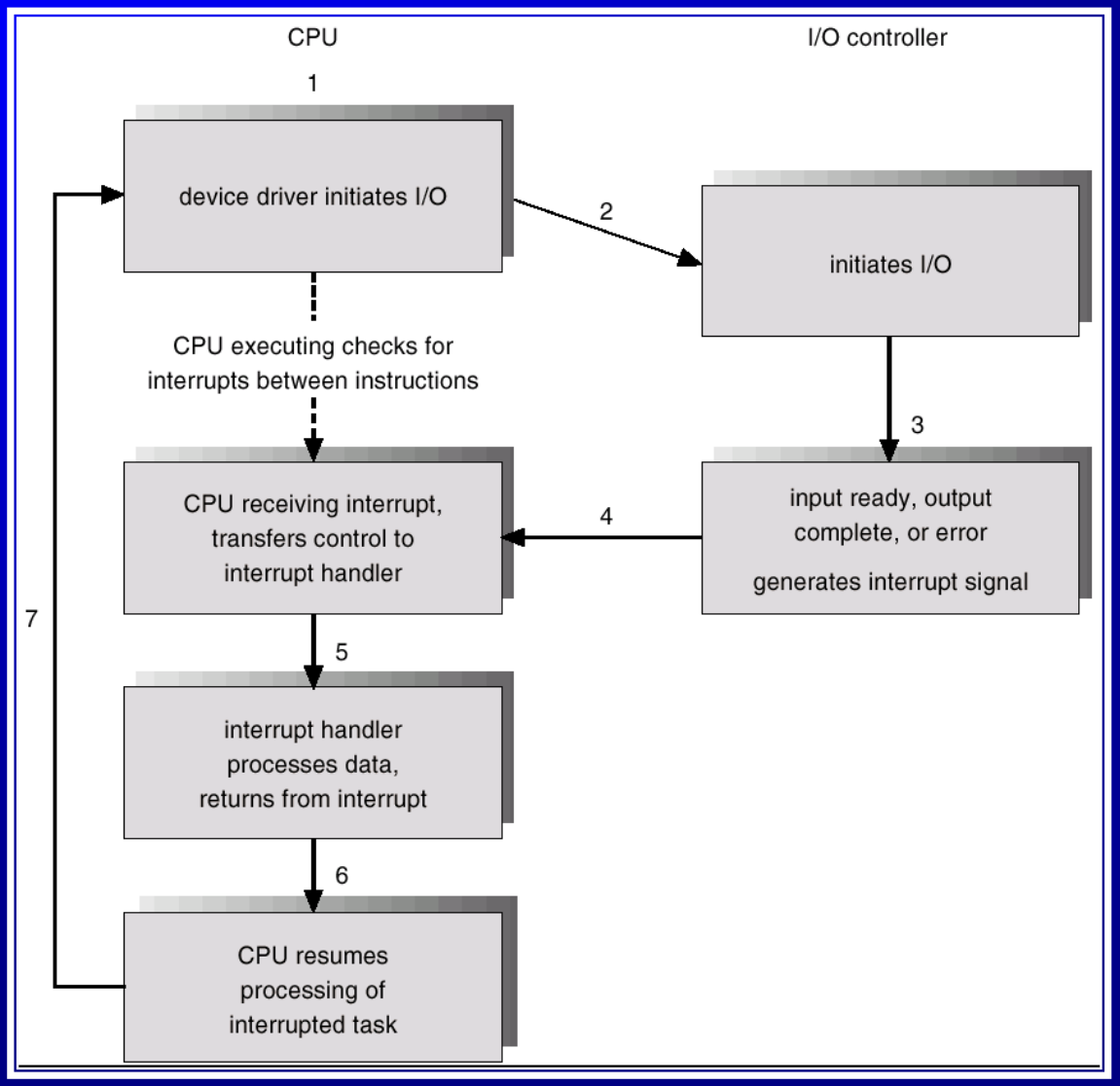
Generic Device I/O

- When a system requires I/O:
  - The system invokes the device driver and passes it a structure that defines the requested operation.
  - The device driver uses the information in the kernel tables to contact the appropriate device.
  - The device driver sets/clears bits in the device registers to cause the device to perform an I/O operation (read/write/…).
  - Unless this is a blocking operation, the device driver returns control to the kernel, and the operation is performed (somewhat) asynchronously.

Interrupts

- CPU Interrupt request line triggered by I/O device
- Interrupt handler receives interrupts
- Maskable to ignore or delay some interrupts
- Interrupt vector to dispatch interrupt to correct handler
  - Based on priority
  - Some unmaskable

Interrupt-Driven I/O Cycle



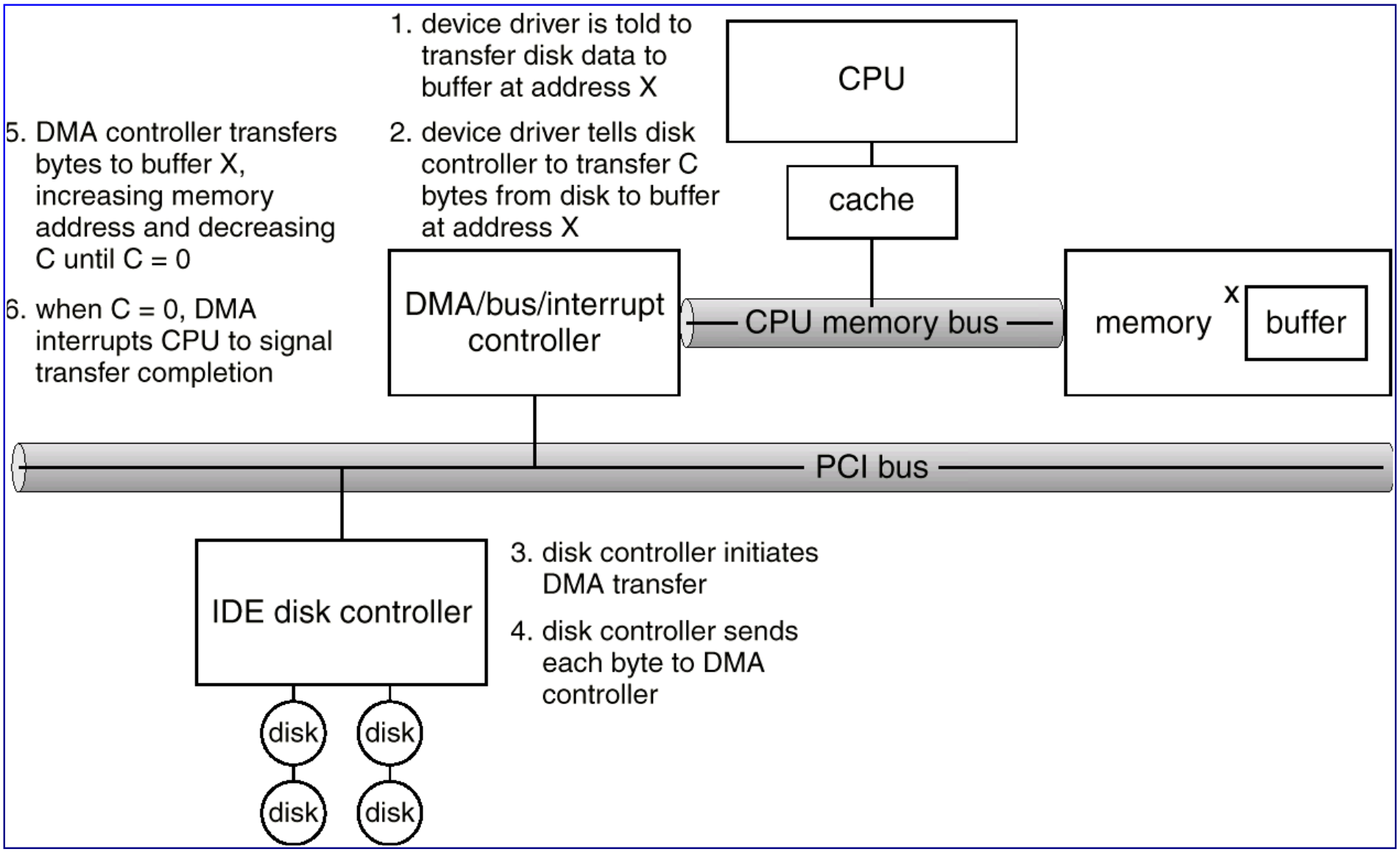
Intel Pentium Processor Event-Vector Table

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
19Ð31	(Intel reserved, do not use)
32Ð255	maskable interrupts

Direct Memory Access

- Used to avoid programmed I/O for large data movement
- Requires DMA controller
- Bypasses CPU to transfer data directly between I/O device and memory

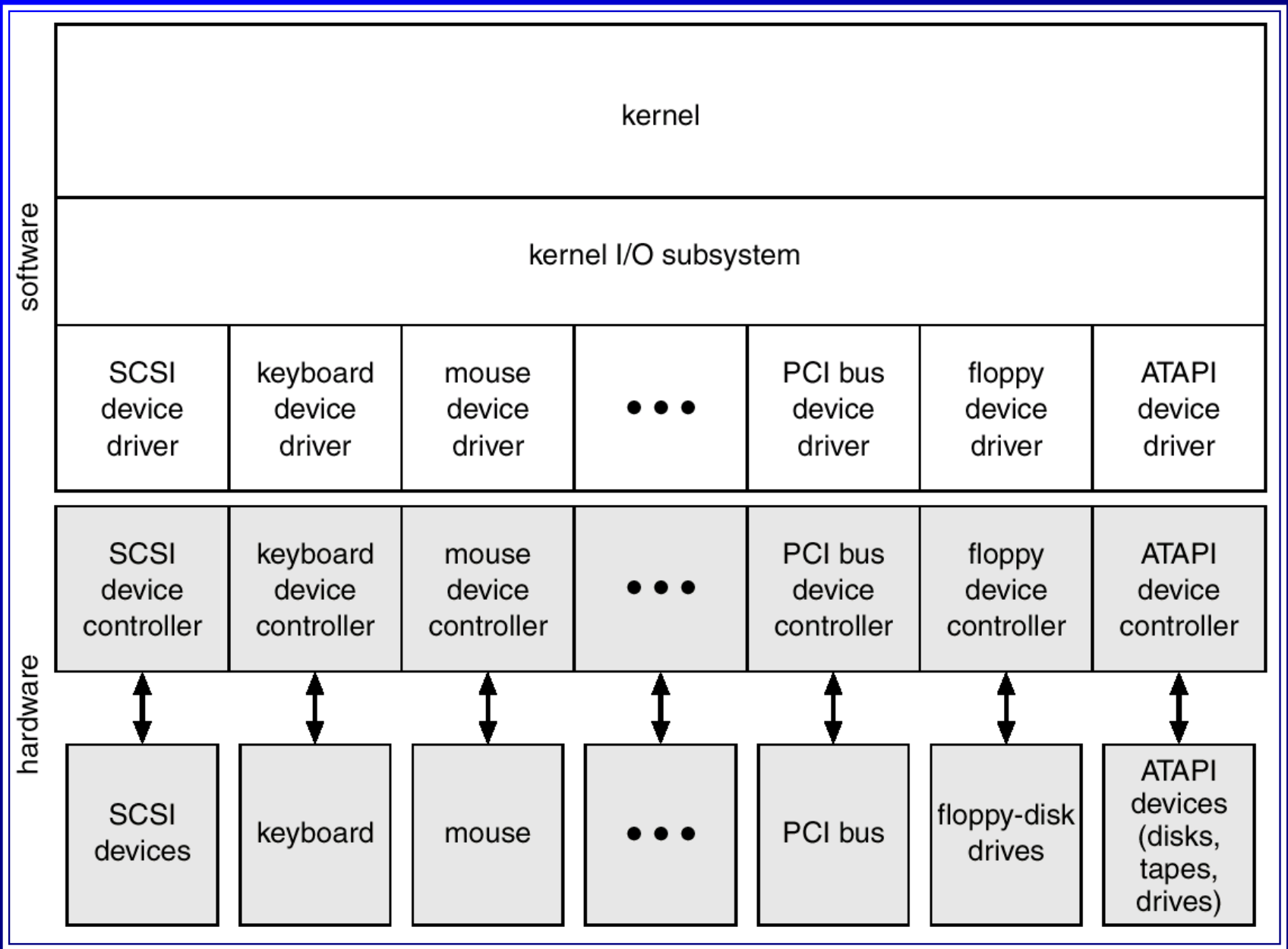
Six Step Process to Perform DMA Transfer



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

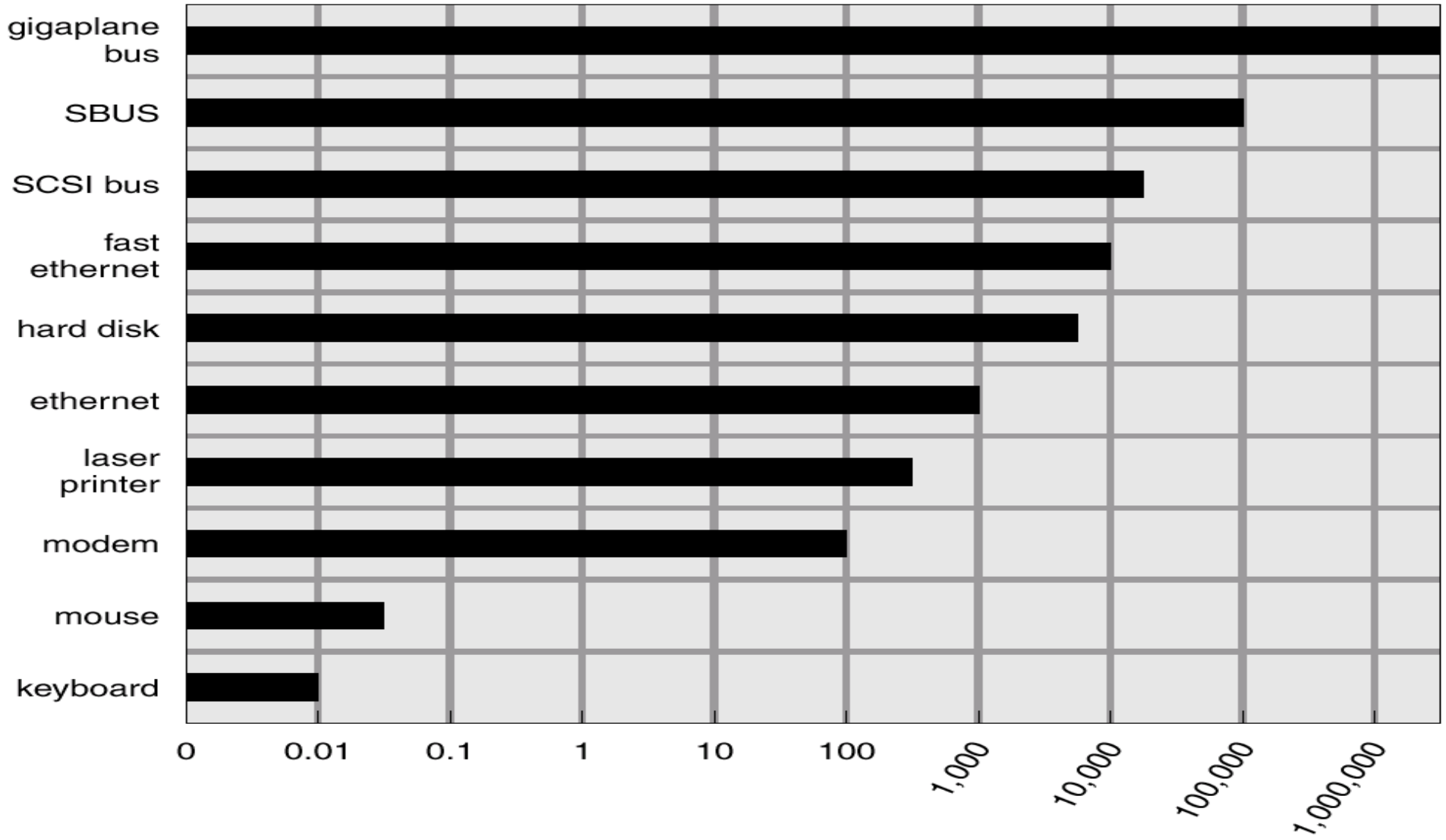
A Kernel I/O Structure



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 readÐwrite	CD-ROM graphics controller disk

Device-Transfer Rates



Block and Character Devices

- Block devices include disk drives
  - Commands include read, write, seek
  - Raw I/O or file-system access
  - Memory-mapped file access possible
- 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
- Linux 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
- If programmable interval time used for timings, periodic interrupts
- `ioctl` (on Linux) covers odd aspects of I/O such as clocks and timers

Kernel I/O Subsystem

- Scheduling
  - Some I/O request ordering via per-device queue
  - Some OSs try fairness
- 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”

Kernel I/O Subsystem

- **Caching -** fast memory holding copy of data
  - Always just a copy
  - Key to high performance of modern OS
- **Spooling -** hold output for a device
  - If device can serve only one request at a time
  - i.e., Printing
- **Device reservation -** provides exclusive access to a device
  - System calls for allocation and deallocation
  - Watch out for deadlock

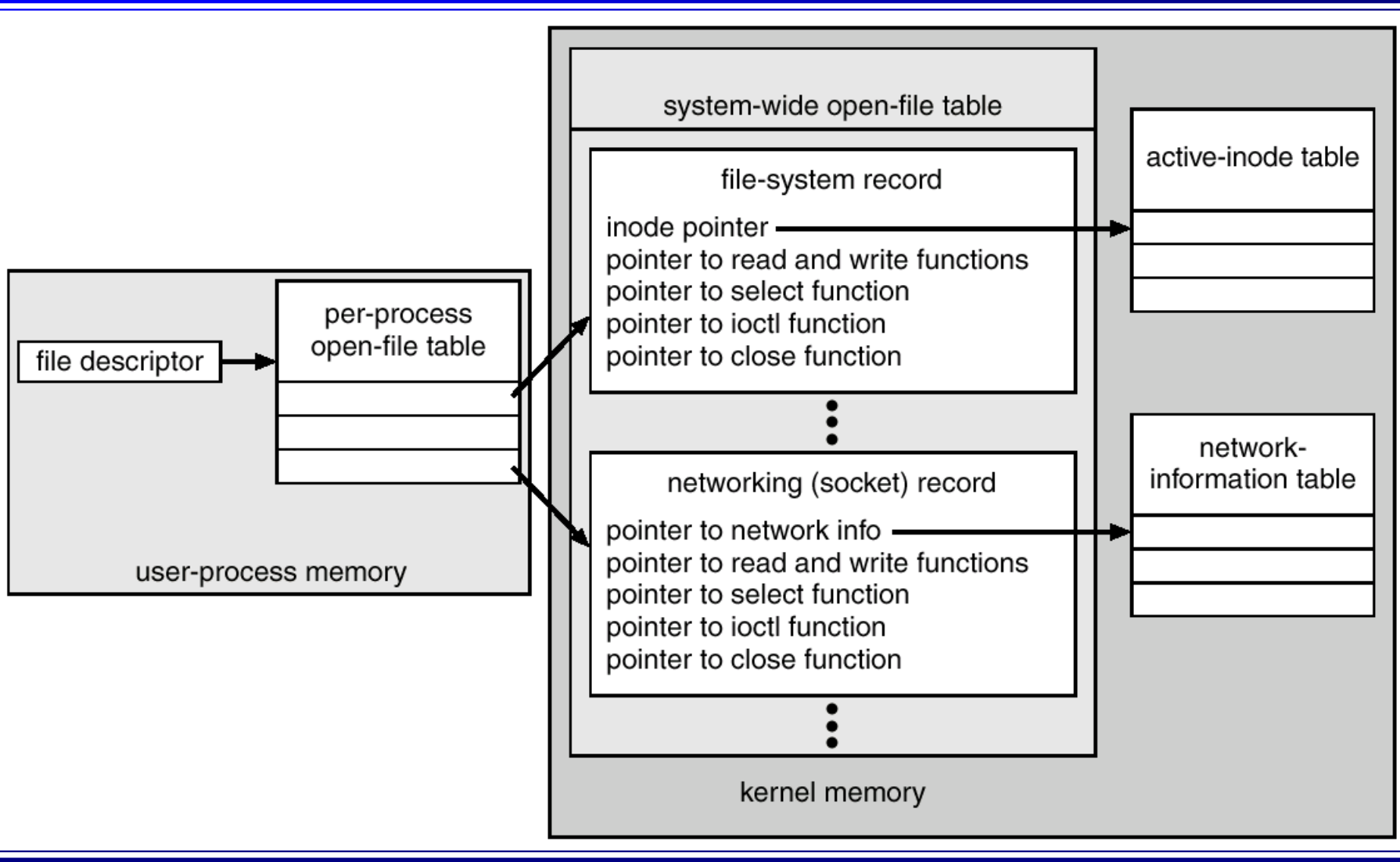
Error Handling

- OS can recover from disk read, device unavailable, transient write failures
- Most return an error number or code when I/O request fails
- System error logs hold problem reports

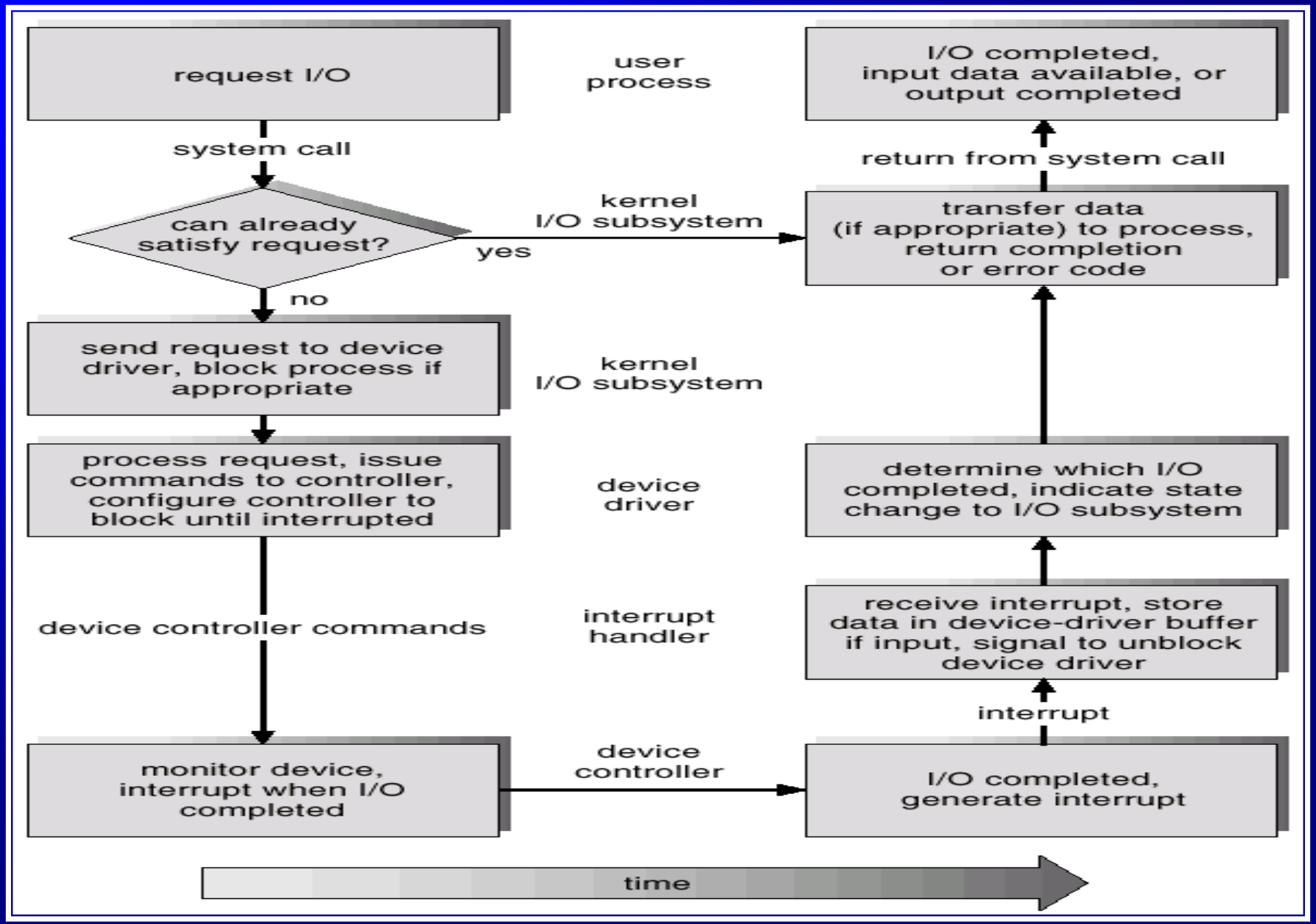
Kernel Data Structures

- 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
- Some use object-oriented methods and message passing to implement I/O

Linux I/O Kernel Structure



Life Cycle of An I/O Request



STREAMS

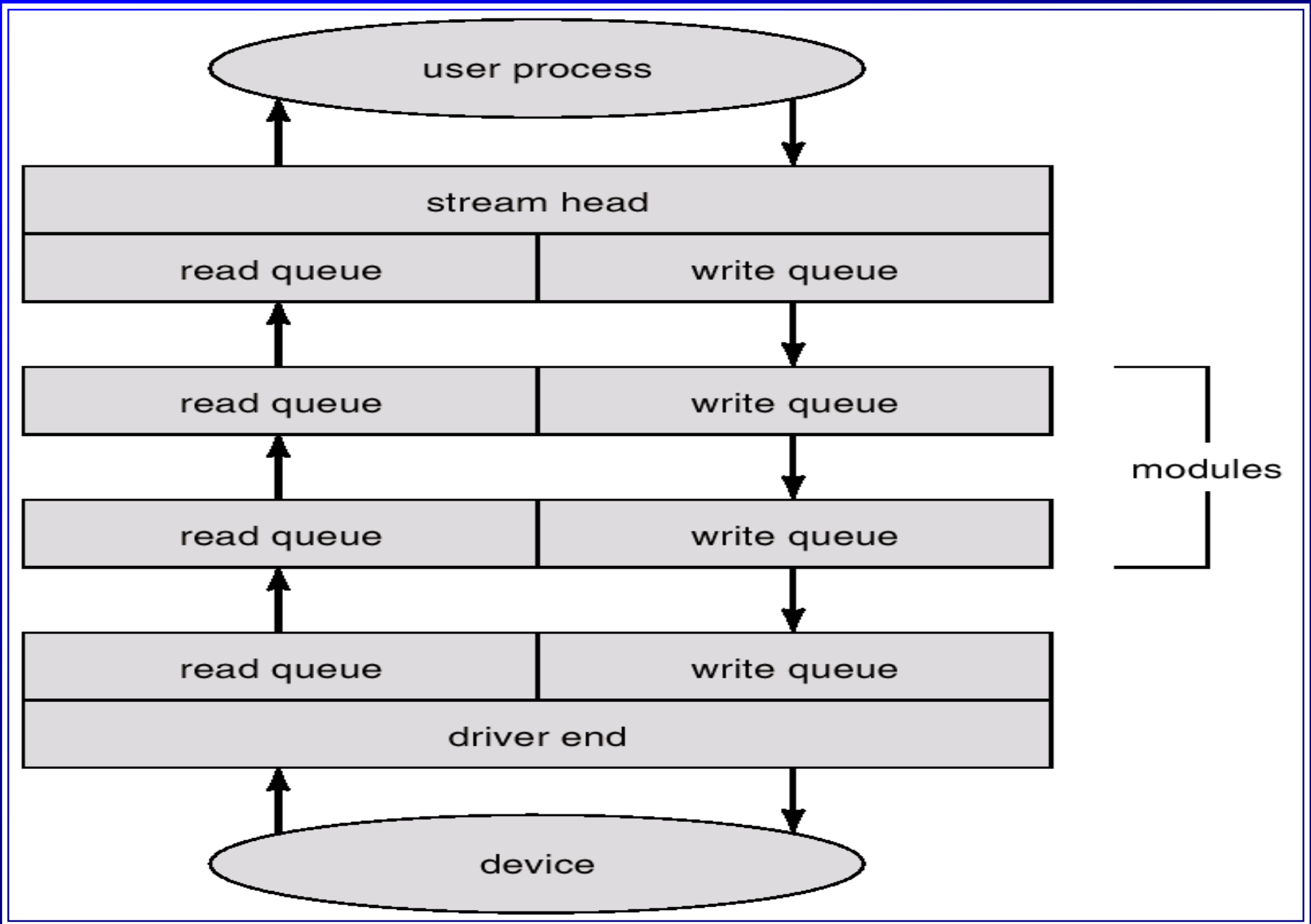
- **STREAM** – a full-duplex communication channel between a user-level process and a device

- A STREAM consists of:
  - **STREAM head** interfaces with the user process
  - **driver end** interfaces with the device
  - zero or more STREAM modules between them.

- Each module contains a **read queue** and a **write queue**

- Message passing is used to communicate between queues

The STREAMS Structure



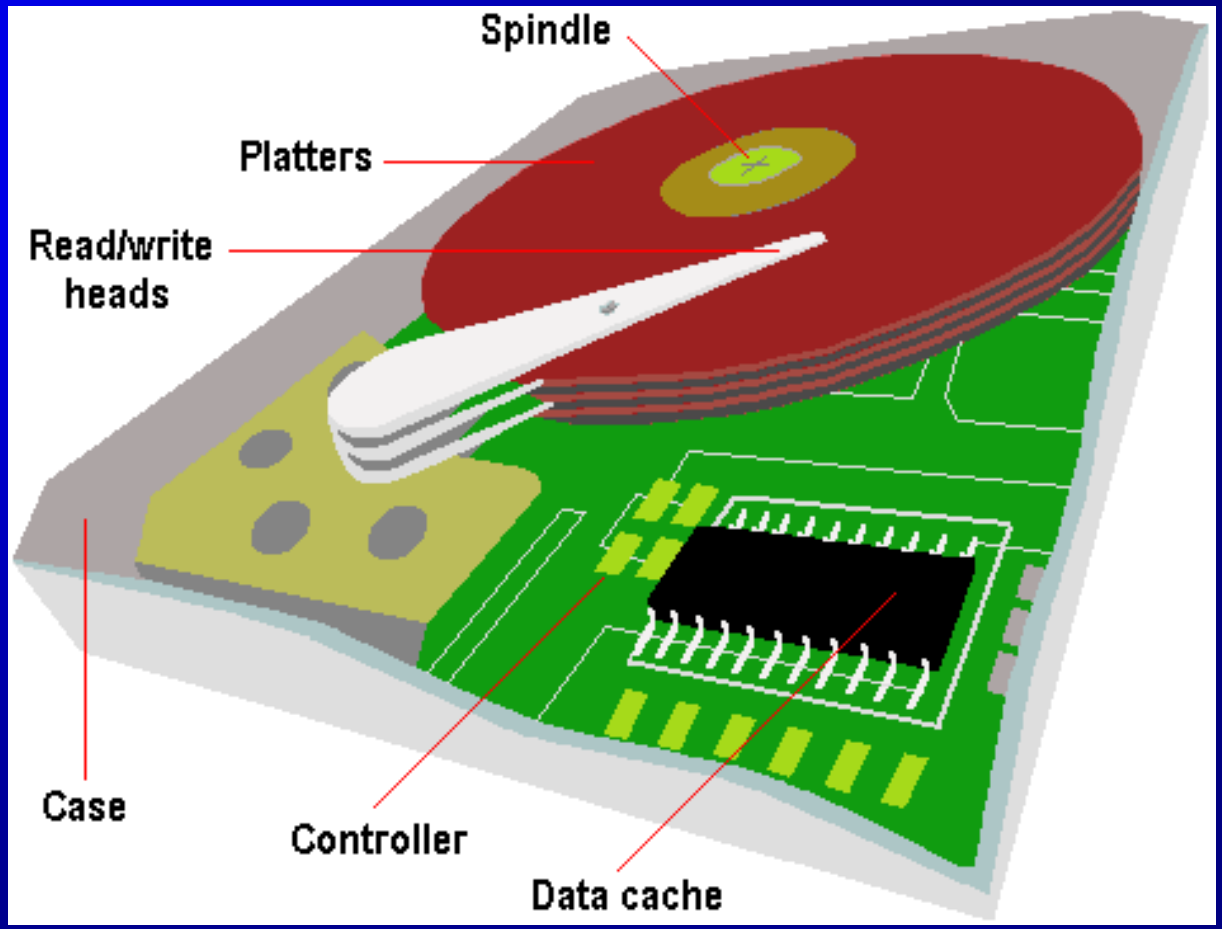
ENCE360 Operating Systems Input/Output Outline

- Introduction (done)
- Hardware (done)
- Software (done)
- **Specific Devices** ←
  - **Hard disk drives**
  - **Clocks**
  - **Bus terminology**
  - **SCSI vs SATA**

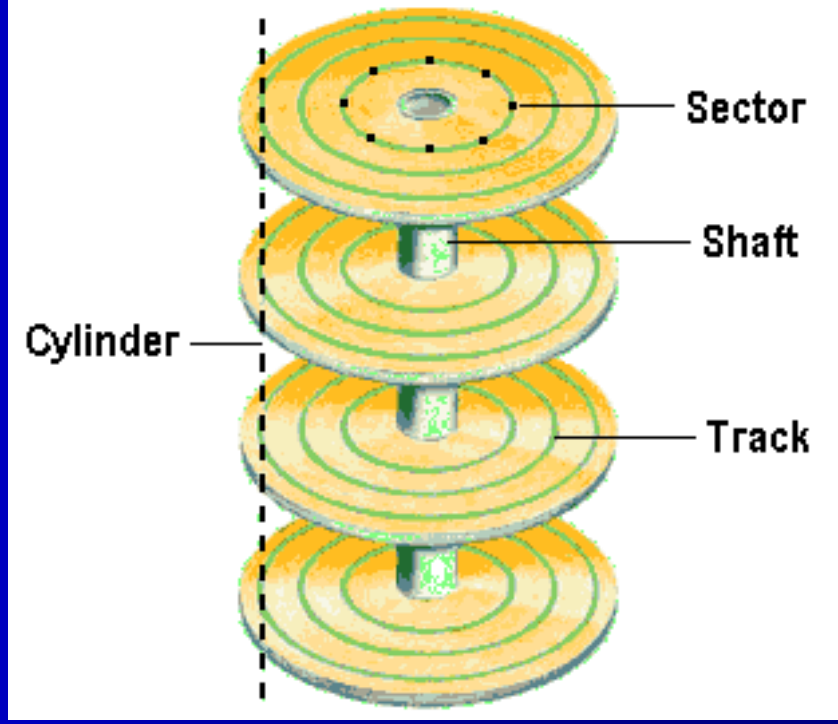
Hard Disk Drives

Hard Disk Drives (HDD)

- Controller often on disk
- Cache to speed access



HDD



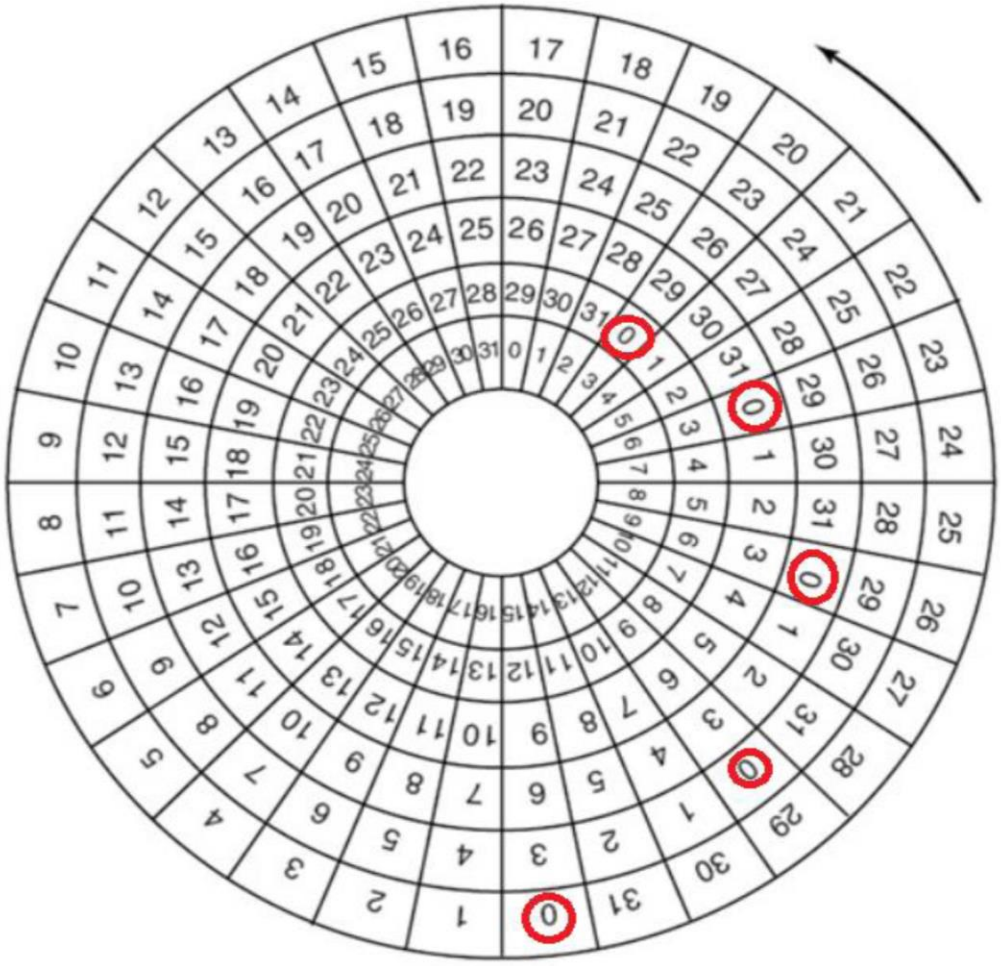
- Platters
- Tracks
- Cylinders
- Sectors
- Disk Arms all move together

E.g. 4TB Hitachi Ultrastar 7K4000 3.5" HDD

- **five 800GB platters spinning at 7,200 rpm**
- **16K cylinders** (~8 billion sectors, 4KB per sector)
- **16 heads with 8 mS seek time**
- **64MB buffer**
- **SATA 6Gb/s interface**
- **2 million hours MTBF (228 years)**

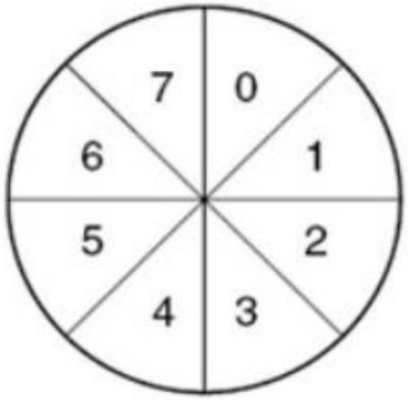
Optimisation 1: cylinder skew

- Allows for time taken to switch cylinders

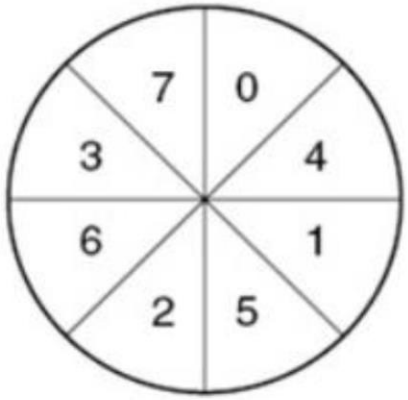


Optimisation 2: sector interleaving

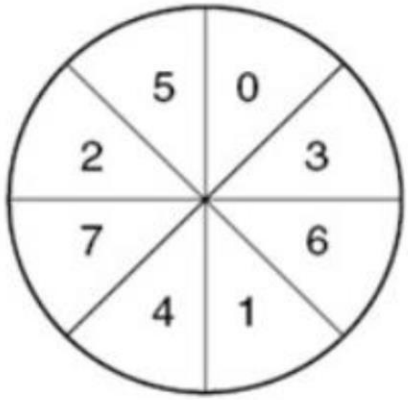
- Allows for time taken to flush buffer



No interleave



single interleave

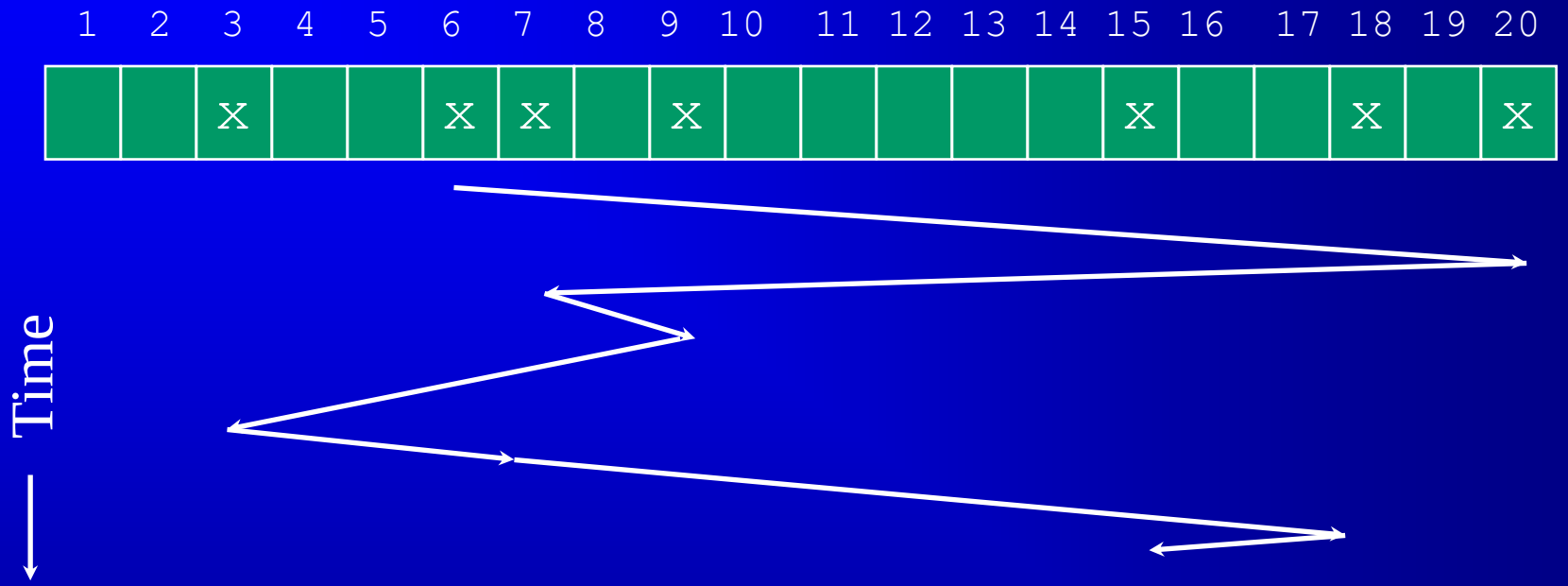


double interleave

Optimisation 3: disk arm Scheduling

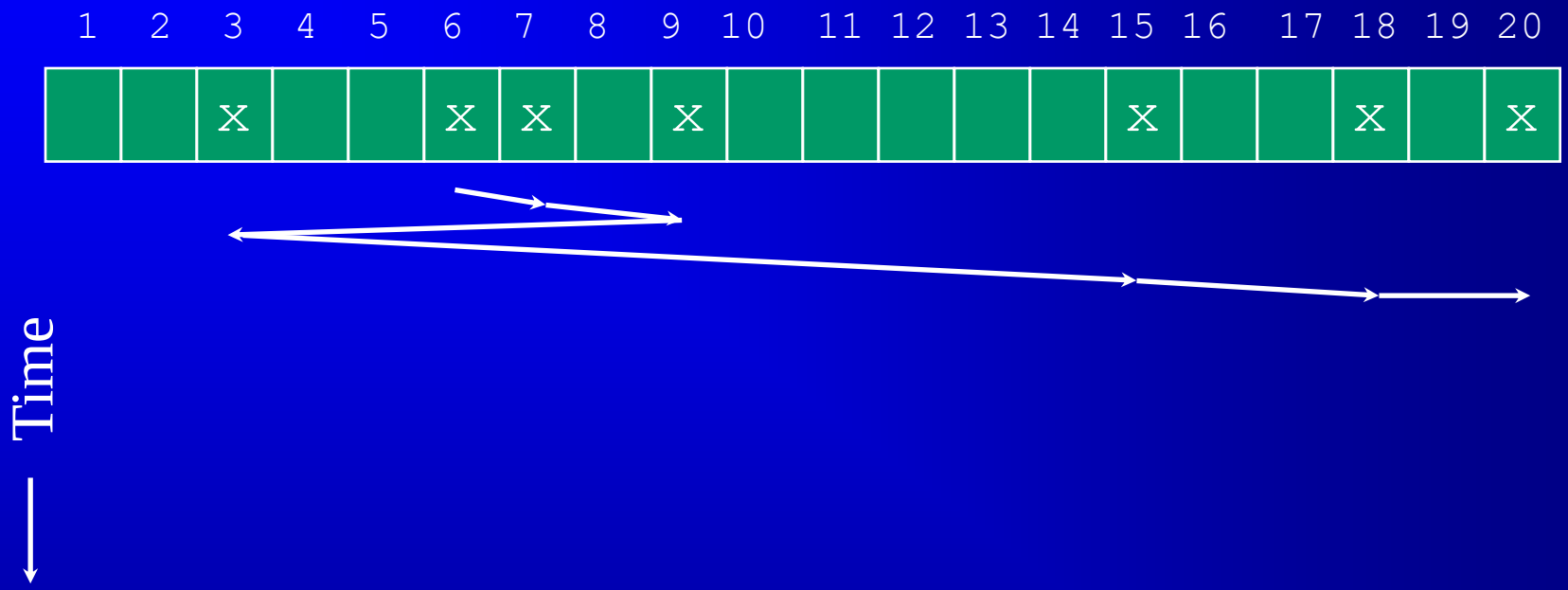
- Read time:
  - seek time (move arm to cylinder)
  - rotational delay (time for sector to rotate under head)
  - transfer time (time to read bits off disk)
- Seek time dominates
- How does disk arm scheduling affect seek?

First-Come First-Served (FCFS)



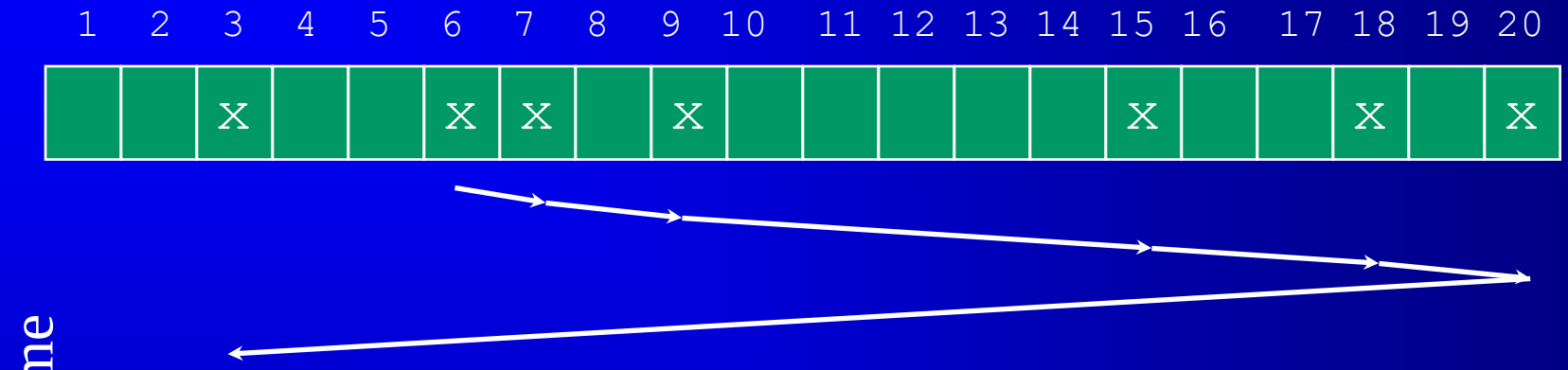
- 14+13+2+6+3+12+3=53
- Service requests in order that they arrive
- Little can be done to optimize
- What if many requests?

Shortest Seek First (SSF)



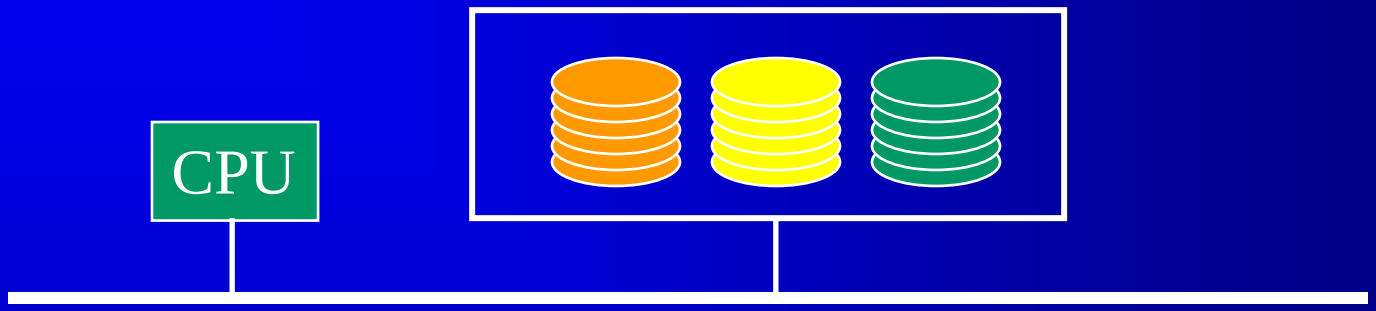
- 1+2+6+12+3+2 = 26
- Suppose many requests?
  - Stay in middle
  - Starvation!

Elevator (SCAN)



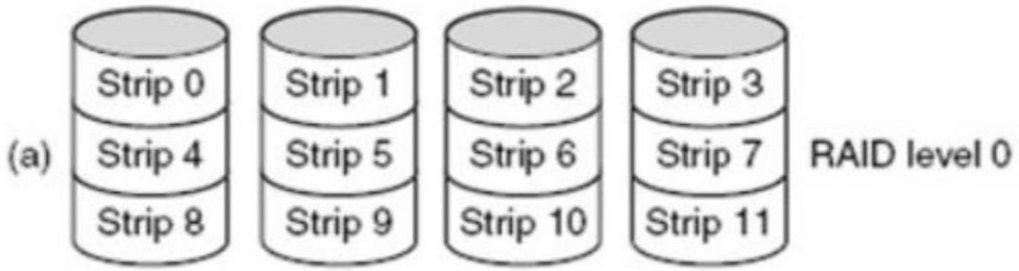
- 1+2+6+3+2+17 = 31
- Usually, a little worse avg seek time than SSF
  - But is more fair, avoids starvation
- C-SCAN (Circular SCAN) has less variance
- Note, seek getting faster, rotational not
  - Someday, change algorithms

Redundant Array of Inexpensive Disks (RAID)

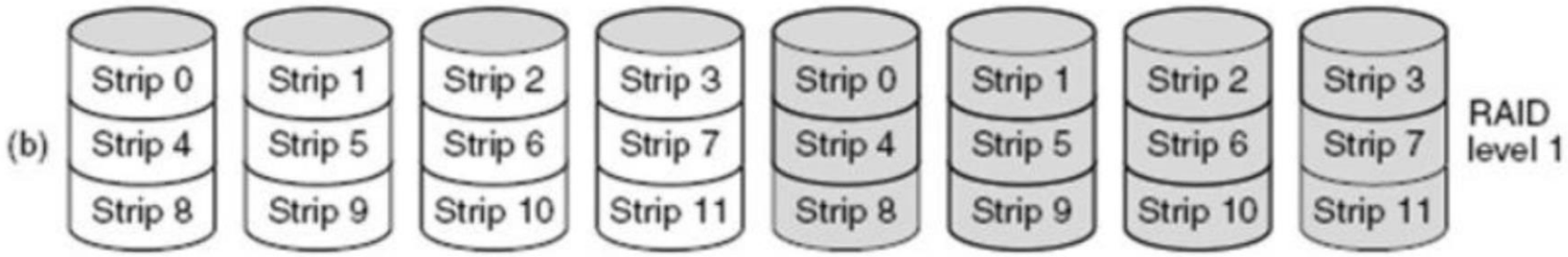


- For speed
  - Pull data in parallel
- For fault-tolerance
  - Example: 38 disks, form 32 bit word, 6 check bits
  - Example: 2 disks, have exact copy on one disk

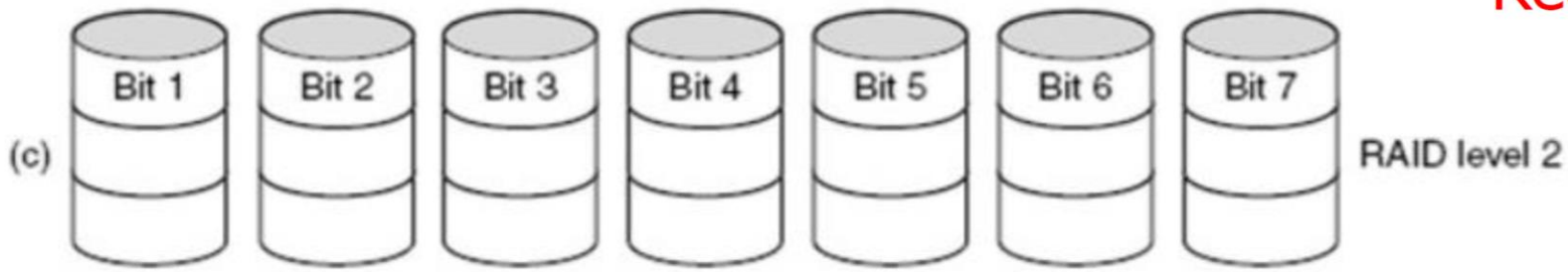
Some RAID configurations



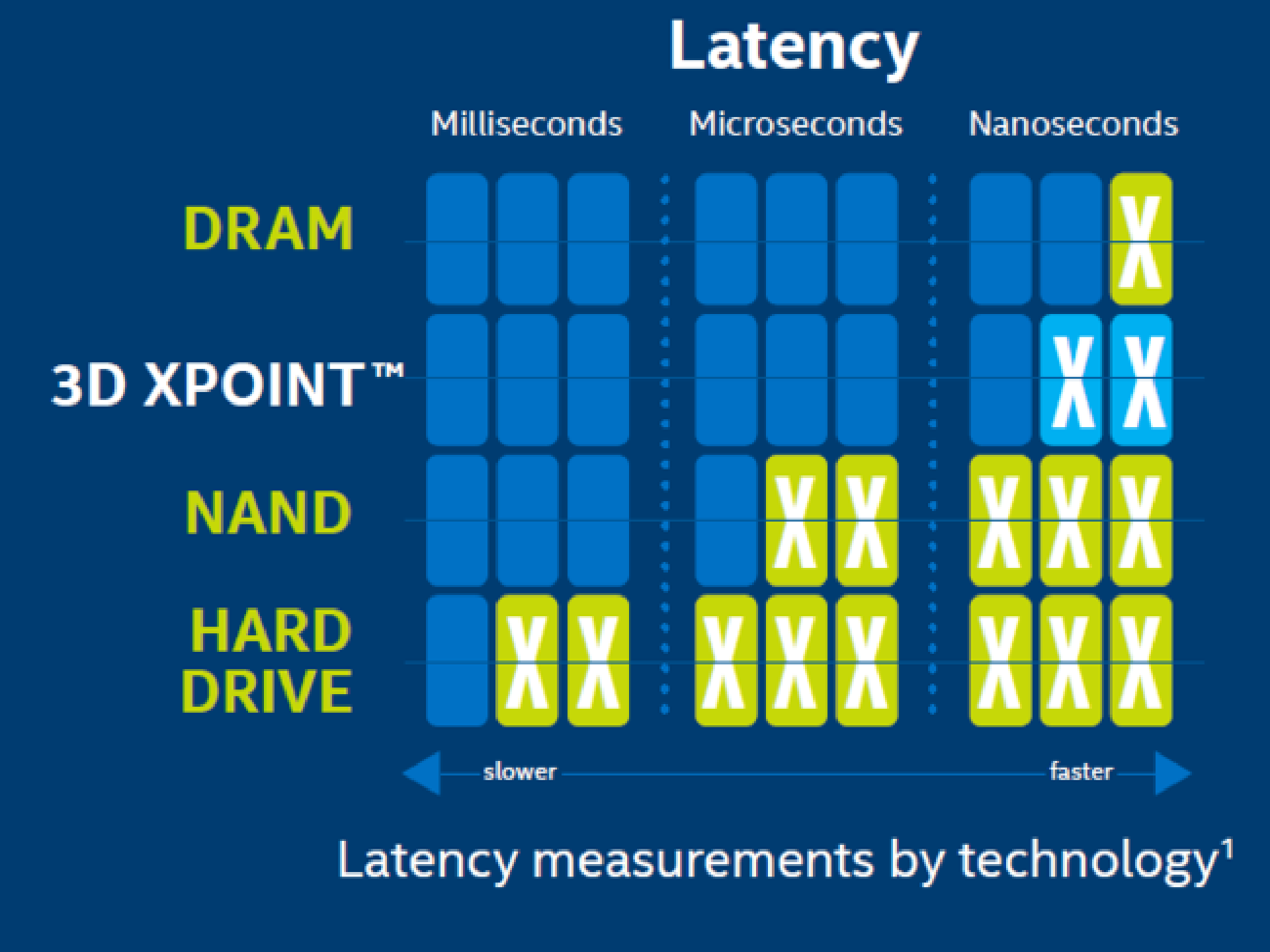
Data sectors "striped" across RAID disks



Redundancy



Super-parallel (requires synchronised disks)

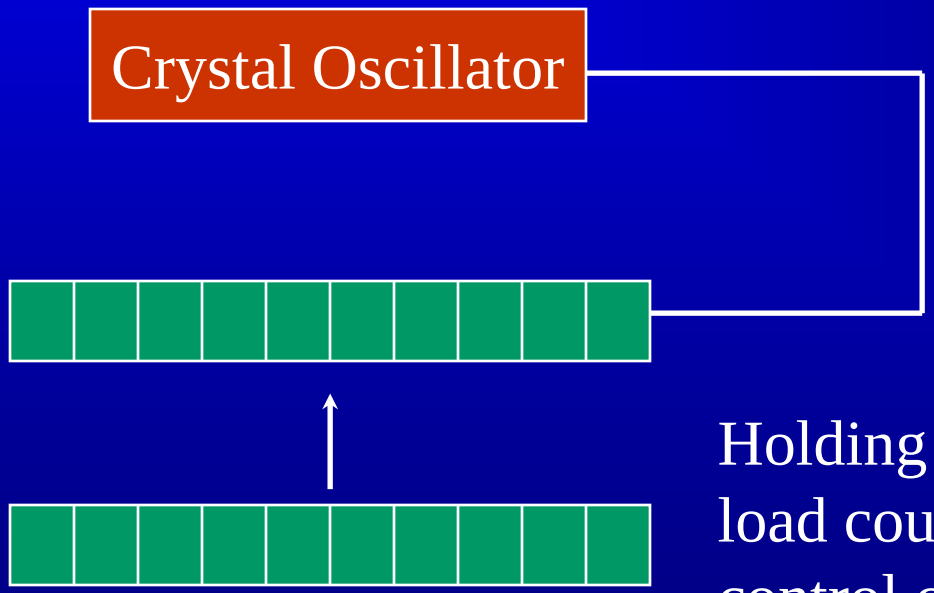


Latency measurements by technology[1]

Clock hardware & software

Clock Hardware

- Time of day to time quantum



Clock Software Uses

- time of day
  - 64-bit, in seconds, or relative to boot
- interrupt after quantum (time interval, or time slice used for batch scheduling systems)
- accounting of CPU usage
  - separate timer or pointer to PCB (Process Control Block)
- `alarm()` system calls
  - separate clock or linked list of alarms with ticks

Bus Architectures

Generic Architecture

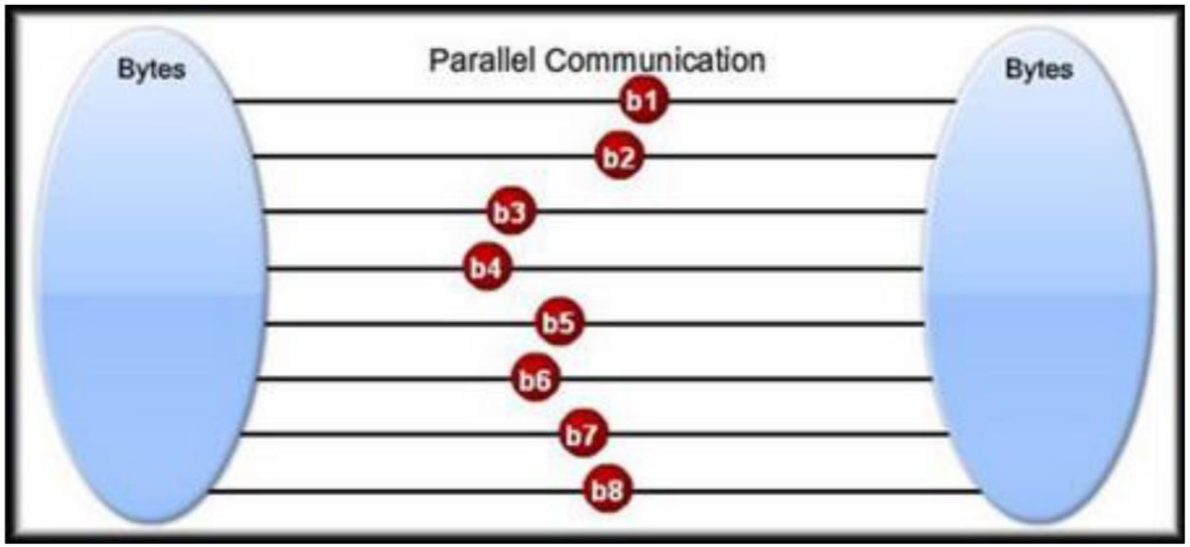
- There are two basic types of buses that we will examine:
  - **Serial bus - contains a single pair of wires for unidirectional data transmission.**
    - USB ports, and network connections are typical examples of serial bus devices.
  - **Parallel bus - contains multiple wires for high-speed bi-directional data transfers.**
    - Disk drives, and graphics cards are typical examples of parallel bus devices.

Serial now beats Parallel Interfaces – why?

- **Parallel**: All information is sent as zero and ones - and this works ok at low frequencies for say splitting 8 bits over 8 wires to send a byte of data at once. For example, these 8 bits travel 300mm in one nanosecond at the slow 1 GHz - and it is easy to cut and connect 8 wires to the same length within an accuracy of 300mm. But at very high frequencies such as 1000GHz, the signal only travels 0.3mm – so if you send these 8 bits in parallel at the same time, unless the wires are exactly the same length, the ones will arrive at different times (“clock skew”) and there is no way of knowing which ones belonged to the current/previous/next byte. But we cannot practically cut and connect wires to such a high accuracy (<0.3mm) to enable perfectly synchronising those 8 bits arriving at exactly the same time - and it only gets worse with higher frequencies.
- **Serial** communication solves this by just sending one bit at a time - with no frequency limit - so there is no need for perfect synchronisation between 8 wires because now there is only essentially one wire. For example Serial SCSI now supports higher data rates than Parallel SCSI, enabling simplified cabling and longer reach.

Serial now beats parallel interfaces – why?

- **Parallel**:
  - at 1GHz: bit "travels" 300mm per tick
  - at 1000GHz: 0.3mm per tick
  - clock skew: different bits arrive at different times depending on cable length



- **Serial**:
  - send one bit at a time
  - no frequency limit
  - e.g. Serial SCSI supports higher data rates than Parallel SCSI

Bus Acronyms

- EISA - Extended Industry Standard Architecture
- ATA – Advanced Technology Attachment (for IBM AT PC)
  uses IDE connector (Integrated Drive Electronics)
- PCMCIA
  1. PC Card Means Confusing The Industry Again
  2. People Cannot Memorize Computer Industry Acronyms
  3. Personal Computer Memory Card International Association
  4. Plastic Connectors May Crack If Adjusted
- SCSI – Small Computer System Interface
- AGP – Accelerated Graphics Port
- PCI – Peripheral Component Interconnect
- SATA - Serial ATA
- eSATA – External SATA
- USB – Universal Serial Bus

Bus Terminology

- **Parallel Bus Interfaces**

  history:

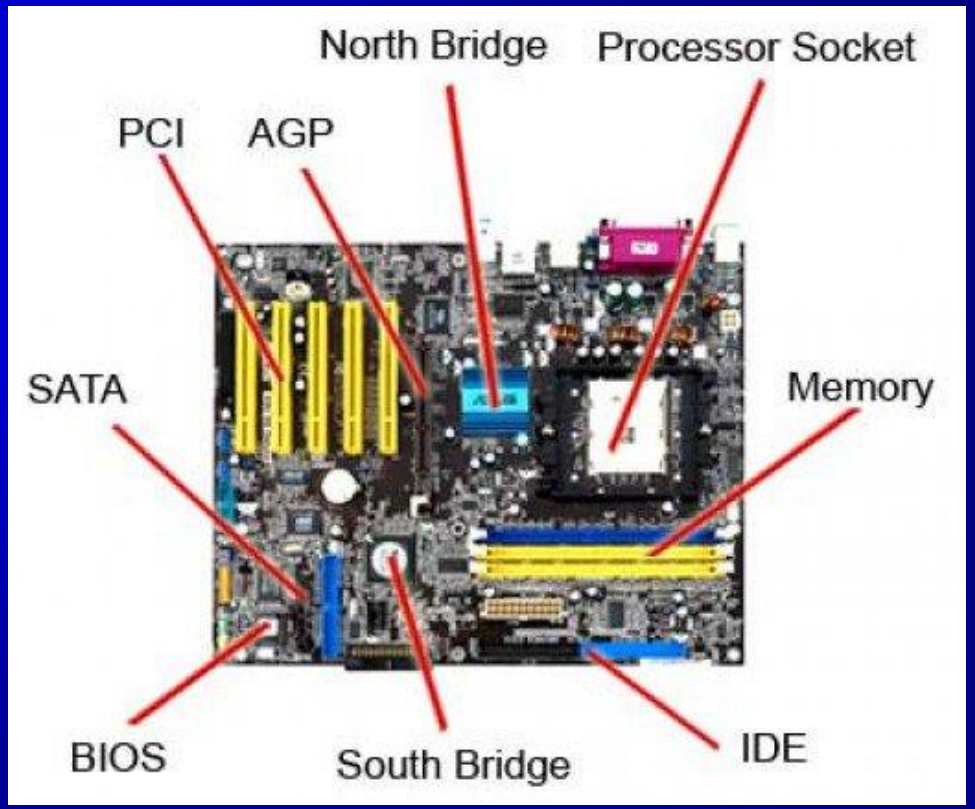
  - **EISA 33 MBps**
  - **ATA 167 MBps**
  - **SCSI 320 MBps**
  - **Express Card (was PC Card, which was PCMCIA) 400 MBps**

  now:

  - **AGP 2.1 GBps**
  - **PCI Express 4.0 31GBps** (2017)

- **Serial Bus Interfaces**

  - **eSATA: 16 Gbps**
  - **USB-C Thunderbolt 3: 40 Gbps**



SATA: Serial ATA

- **SATA 1.0** 1.5 Gbit/s (First generation)
- **SATA 2.0** 3 Gbit/s (Second generation)
- **SATA 3.0** 6 Gbit/s (Third generation)
- **SATA 3.2** **16 Gbit/s** (SATA Express)
- **eSATA:** special connector specified for external devices
- **eSATAp:** power over eSATA or eSATA/USB Combo

- **While even the fastest conventional hard disk drives can barely saturate the original SATA 1.5 Gbps bandwidth, Solid State Disk drives are close to saturating the SATA 3 Gbps limits.**
- **Ten channels of fast flash can actually reach such a high bandwidth that a move from SATA 3 to SATA 3.2 would benefit the flash read speeds.**

Parallel **SCSI**:
Small **C**omputer **S**ystems **I**nterface

- Parallel interface with 8, 16, 32 or 64 bit data lines
- Daisy chained disks
  - 7 for standard SCSI
  - 15 for wide SCSI
- Devices are independent
- Devices communicate with each other and host
- SATA is simpler (cheaper) so now more common

SATA vs SCSI

- **SCSI is a more complex bus than SATA**

  So **SATA based systems are cheaper**

  (SCSI was popular on high-end PCs and servers)

- Desktop computers and notebooks now typically use SATA interfaces for internal hard disk drives - and USB, eSATA for external devices

FireWire vs USB

- **FireWire**, uses a "Peer-to-Peer" architecture in which the peripherals are intelligent and can negotiate bus conflicts to determine which device can best control a data transfer

- Hi-Speed **USB** uses a "Master-Slave" architecture in which the computer handles all arbitration functions and dictates data flow to, from and between the attached peripherals (adding additional system overhead and resulting in slower data flow control)

M.2

- M.2 is a spec for internally mounted computer expansion cards and associated connectors (replaces the mSATA standard/PCI Express Mini Card)
- M.2 is more suitable than mSATA for SSDs and supports PCI Express 4.0 (up to four lanes), SATA 3.0, USB3.0/C
- M.2 modules can integrate multiple functions such as Wi-Fi, Bluetooth, sat nav, NFC, digital radio, WiGig, WWAN and SSDs

Size comparison of an mSATA (left) and an **M.2** SSD (right)

ENCE360 Operating Systems Input/Output Outline

- Introduction (done)
- Hardware (done)
- Software (done)
- Specific Devices (done)
- **Performance** ←

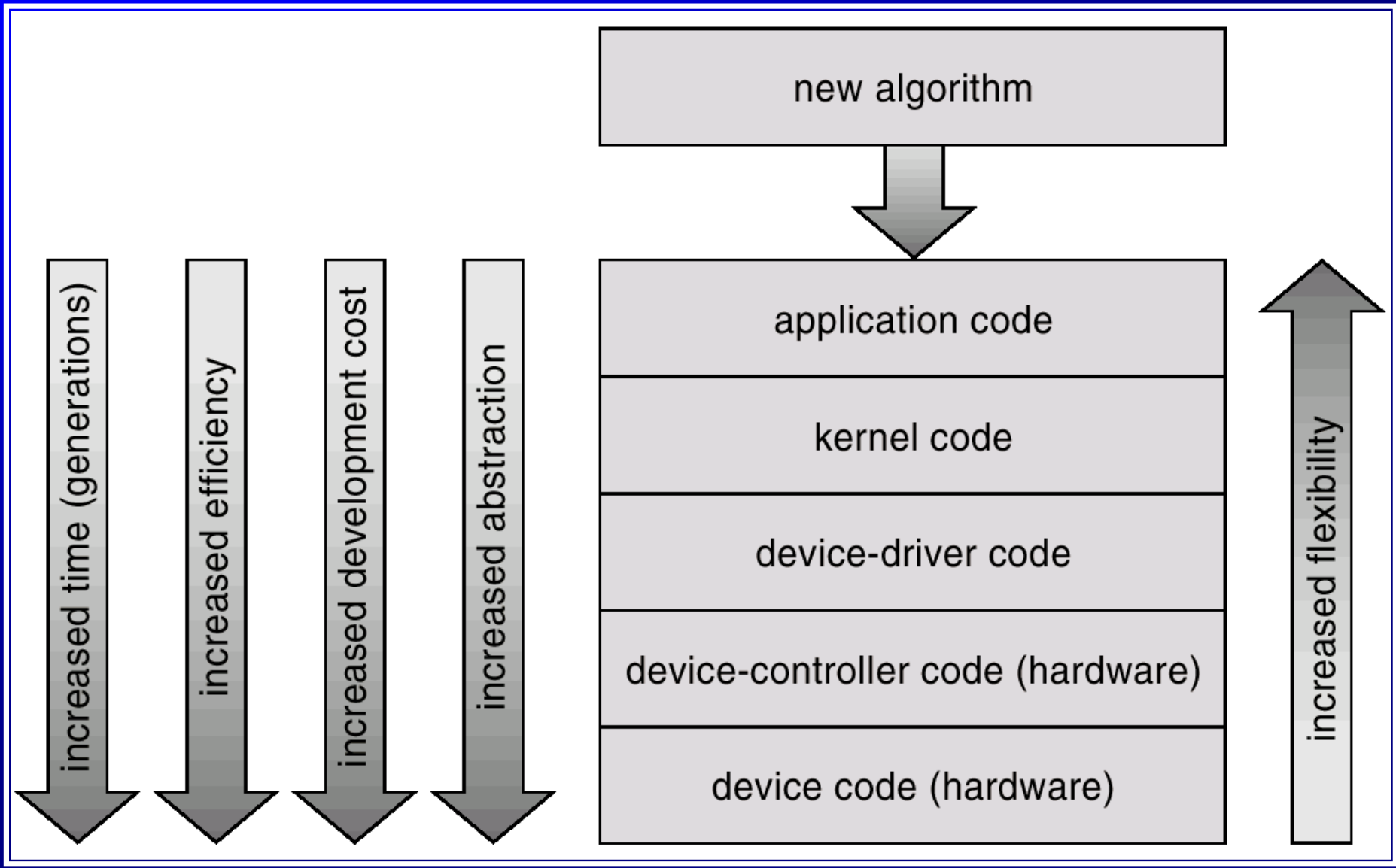
Performance

- I/O is 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

Improving Performance

- Reduce number of context switches
- Reduce data copying
- Reduce interrupts by using large transfers, smart controllers, polling
- Use DMA
- Balance CPU, memory, bus, and I/O performance for highest throughput

Device-Functionality Progression



Example Exam Questions

Example Exam Questions

What percentage of the Linux OS is dedicated to controlling devices?

What are the fastest and slowest I/O devices and what speed?

Example Exam Questions

What percentage of the Linux OS is dedicated to controlling devices?

90%

What are the fastest and slowest I/O devices and what speed?

Keyboard @ 10bps,

USB-C Thunderbolt 3 @ 40 Gbps

(or PCI Express 4.0 @ 31 GBps)

Example Exam Questions

What do the following acrynoms stand for?

PCMCIA, ISA, EISA, PCI, AGP, USB

Example Exam Questions

What do the following acrynoms stand for?
PCMCIA, EISA, PCI, AGP, USB

- **PCMCIA - Personal Computer Memory Card International Association**
- **EISA – Extended Industry Standard Architecture**
- **PCI – Peripheral Component Interconnect**
- **AGP – Accelerated Graphics Port**
- **USB – Universal Serial Bus**

Example Exam Questions

What information does the CPU pass to the DMA controller?

Example Exam Questions

What information does the CPU pass to the DMA controller?

- **Read/Write instruction**
- **Device address**
- **Starting address of memory block for data**
- **Amount of data to be transferred**

Example Exam Questions

Describe DMA Transfer Cycle Stealing?

Example Exam Questions

Describe DMA Transfer Cycle Stealing?

- **DMA controller takes over bus for a cycle**
- **Transfer of one word of data**
- **Not an interrupt - CPU does not switch context**
- **CPU suspended just before it accesses bus**
  - **i.e. before an operand or data fetch or a data write**
- **Slows down CPU but not as much as CPU doing transfer**

Example Exam Questions

Describe three DMA configurations and how many BUS cycles does each use?

Example Exam Questions

Describe three DMA configurations and how many BUS cycles does each use?

- **DMA shares bus with I/O units – 2 cycles**
- **DMA shares bus only with other DMA, CPU, memory – 1 cycle**
- **DMA has a separate I/O bus – 1 cycle**

Example Exam Questions

Why is an Interrupt Handler split into two parts, what are the two parts called in Linux and what do they do?

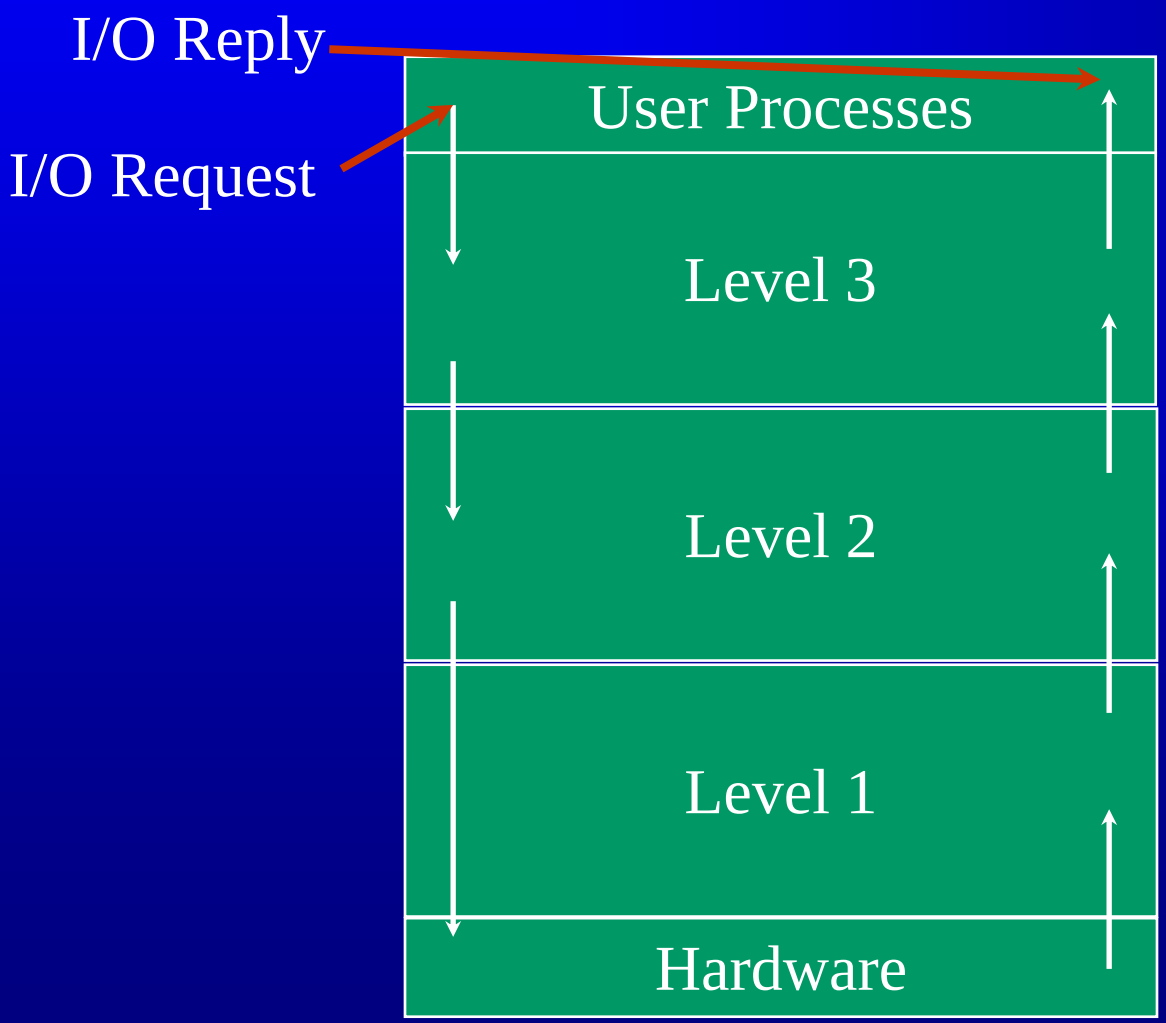
Example Exam Questions

Why is an Interrupt Handler split into two parts, what are the two parts called in Linux and what do they do?

- **Within Interrupt Handler, all Interrupts are Blocked**
- **Interrupt Handlers must finish up quickly so as not to keep interrupts blocked for long**
- **Top half: the function actually responds to the interrupt – within the Interrupt Handler**
- **Bottom-half (Linux) / DPC (Windows): a routine that is deferred by the top half to be executed later, at a safer time - during bottom-half, all interrupts are enabled**

Example Exam Questions

Label levels one to three
with a brief (few words) description of each level.



Example Exam Questions

Label levels one to three
with a brief (few words) description of each level.