Computer Architecture

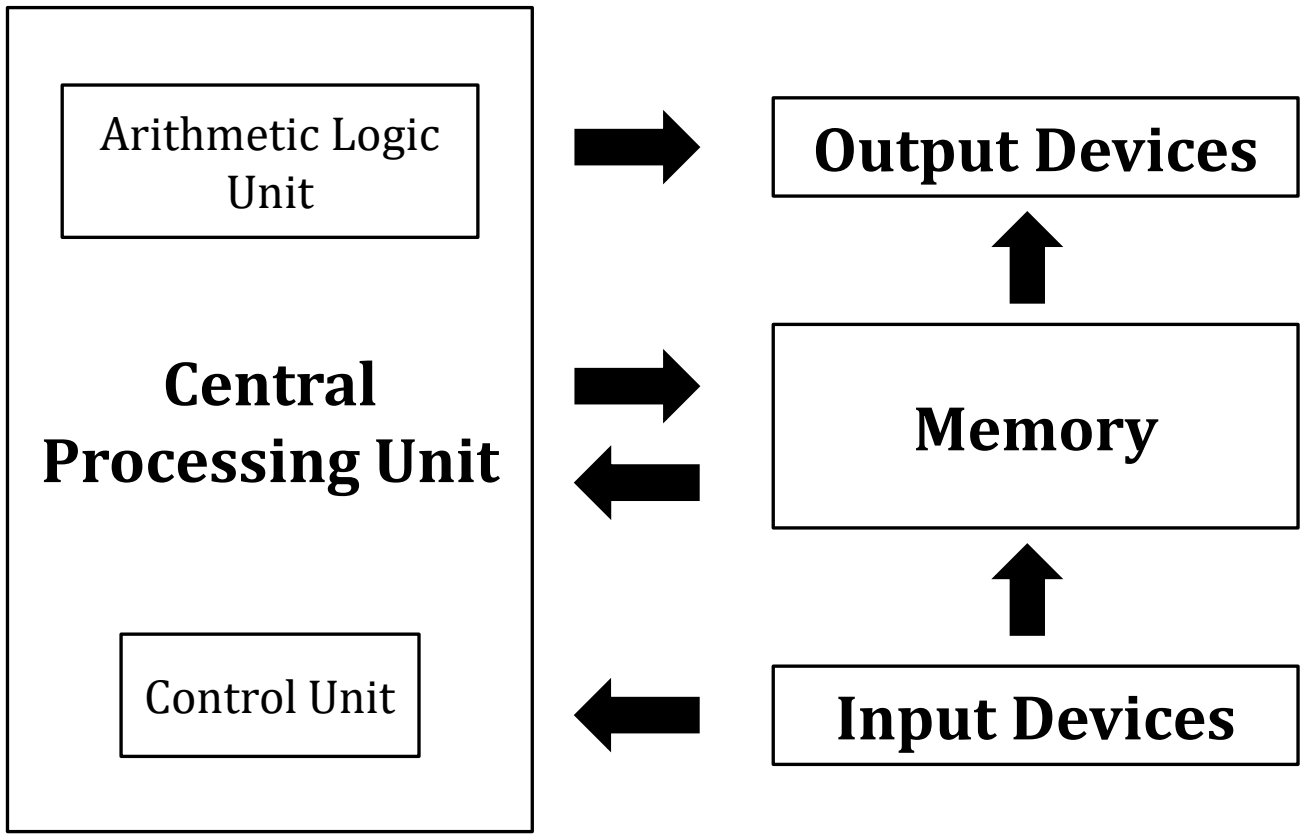
Angshuman Paul

Assistant Professor

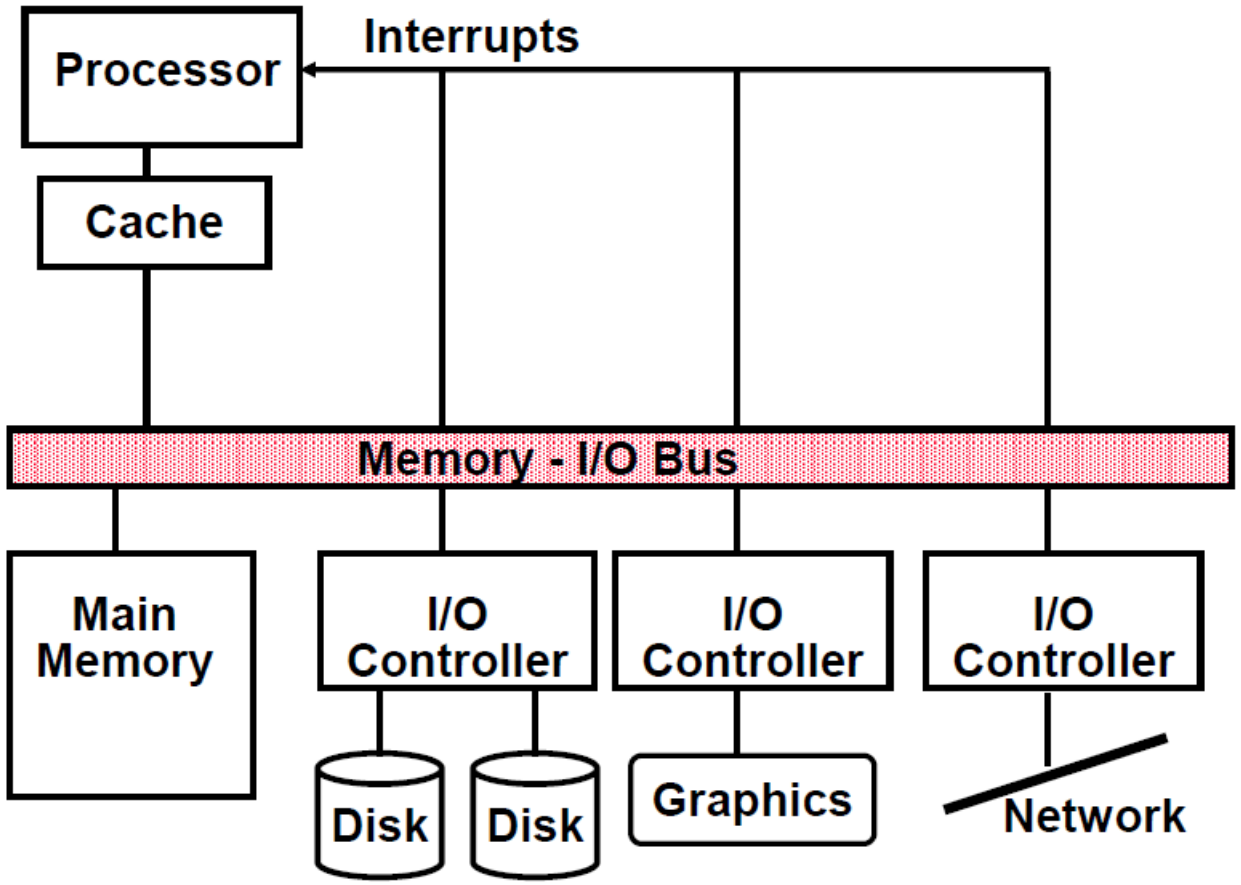
Department of Computer Science & Engineering

Input Output Subsystem

The Components of a Computer



Interfacing



Input Output Subsystem

- I/O Subsystem: used for connecting the computing system with the outside world
  - Through peripheral devices
- Input peripherals : Allows user input, from the outside world to the computer
  - Keyboard, Mouse, scanner etc.
- Output peripherals: Allows information output, from the computer to the outside world
  - Printer, Monitor, speaker, etc.
- Input-Output peripherals: Allows both input (from outside world to computer) as well as output (from computer to the outside world)
  - Touch screen

I/O Performance Measures

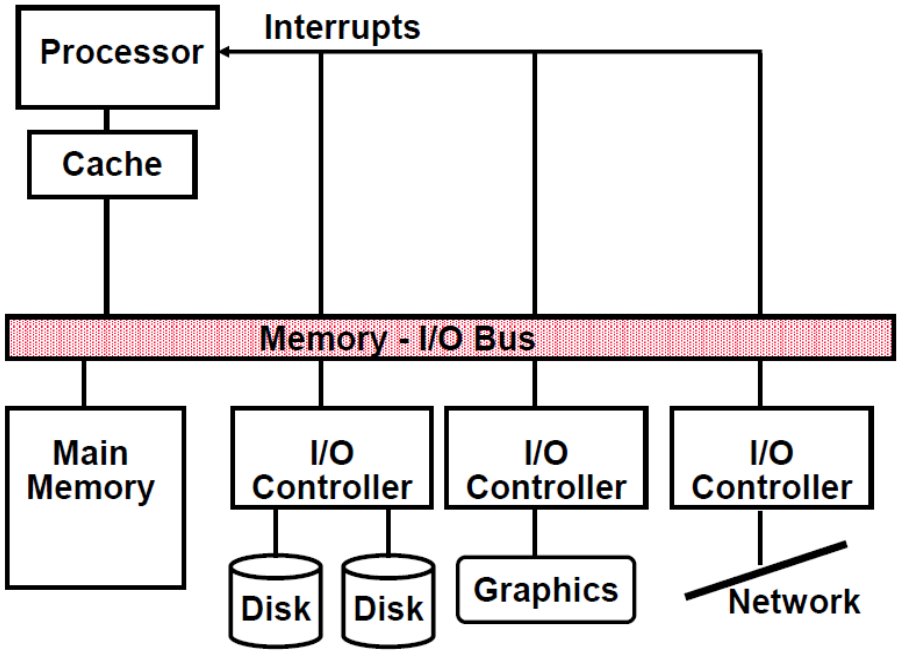
- Throughput
  - Amount of data transfer in unit time (KB/s)
  - IOPS (Input/output operations per second)
    - Random IOPS
      - Read
      - Write
    - Sequential IOPS
      - Read
      - Write
  - IOPS×Transfer size = Amount of data transfer per second

I/O Performance Measures

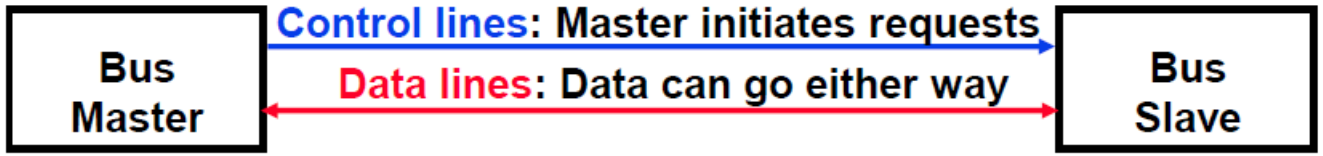
- I/O response time (latency): total elapsed time to accomplish an input or output operation
- Very important performance metric in real-time systems
- Often we need both high throughput and low response time

I/O Interconnection

- **Bus**: a shared communication link (a set of wires) that connects multiple subsystems with widely varying latencies and data transfer rates
- Only one pair of subsystems may interact at a time
- Low cost, low throughput

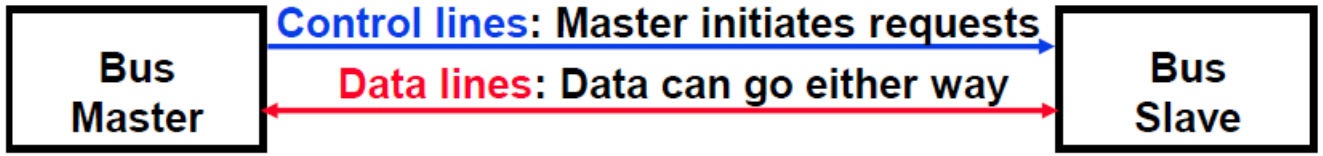


Bus Characteristics



- Control lines
  - Signal requests and acknowledgments
  - Indicate what type of information is on the data lines
- Data lines
  - Data, addresses, and complex commands

Bus Characteristics



- Bus transaction consists of
  - Bus master issues the command (and address): request
  - Bus slave receives (or sends) the data: action

- Defined by what the transaction does to memory
  - Input –inputs data from the I/O device to the memory
  - Output –outputs data from the memory to the I/O device

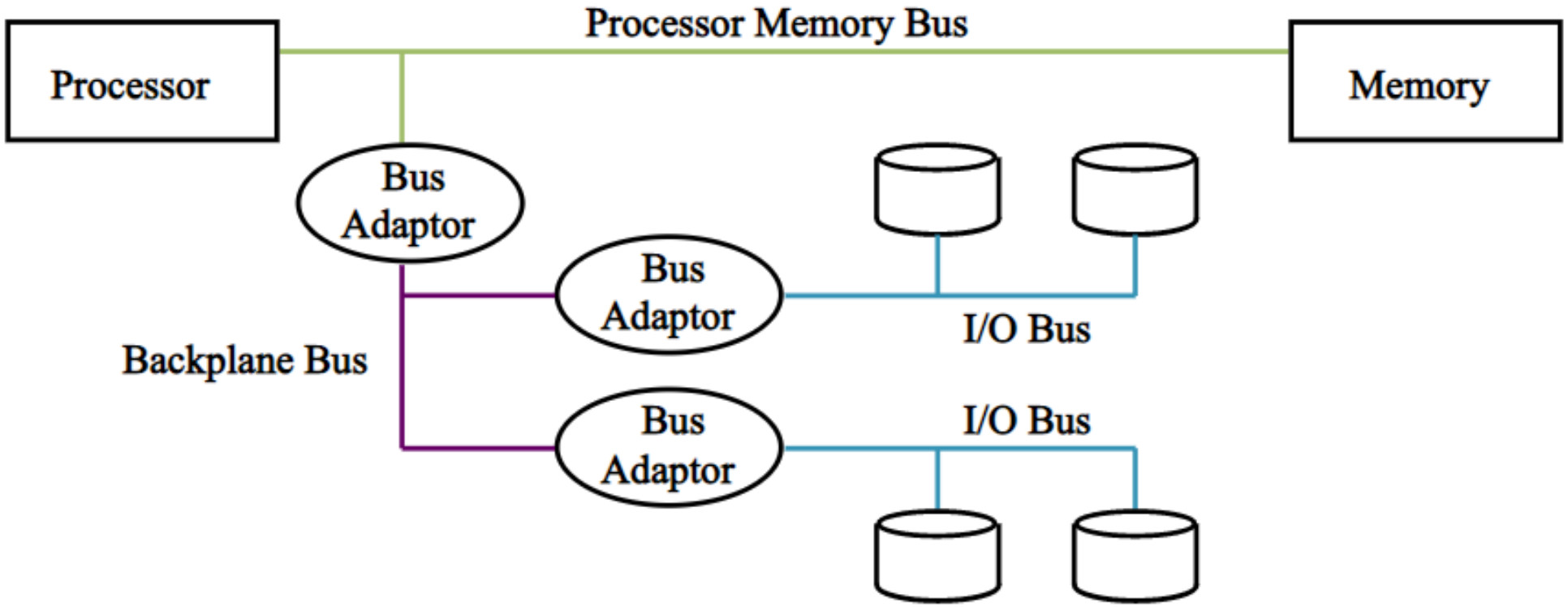
Types of Buses

- Processor-memory bus (proprietary)
  - Short and high speed
  - Matched to the memory system to maximize the memory-processor bandwidth
  - Optimized for cache block transfers

- I/O bus (standard, e.g., SCSI, USB, Firewire)
  - Usually is lengthy and slower
  - Needs to accommodate a wide range of I/O devices
  - Connects I/O to the processor-memory bus or backplane bus

- Backplane bus (standard/ proprietary, e.g., ATA, PCIexpress)
  - The backplane is an interconnection structure within the chassis
  - Used as an intermediary bus connecting I/O busses to the processor-memory bus

All Types of Buses



https://slideplayer.com/slide/4925906/

Types of Buses

- Synchronous bus (e.g., processor-memory buses)
  - Includes a clock in the control lines and has a fixed protocol for communication that is relative to the clock
  - Advantage: involves very little logic and can run very fast
  - Disadvantages
    - Every device communicating on the bus must use same clock rate
    - To avoid clock skew, they cannot be long if they are fast

- Asynchronous bus (e.g., I/O buses)
  - It is not clocked, so requires a handshaking protocol and additional control lines (ReadReq, Ack, DataRdy)
  - Advantages:
    - Can accommodate a wide range of devices and device speeds
    - Can be lengthened without worrying about clock skew or synchronization problems
  - Disadvantage: slower

Bus Arbitration

- Multiple devices may need to use the bus at the same time
- There must be some way to arbitrate multiple requests
- Bus arbitration schemes usually try to balance:
  - Bus priority –the highest priority device should be serviced first
  - Fairness –even the lowest priority device should never be completely locked out from the bus

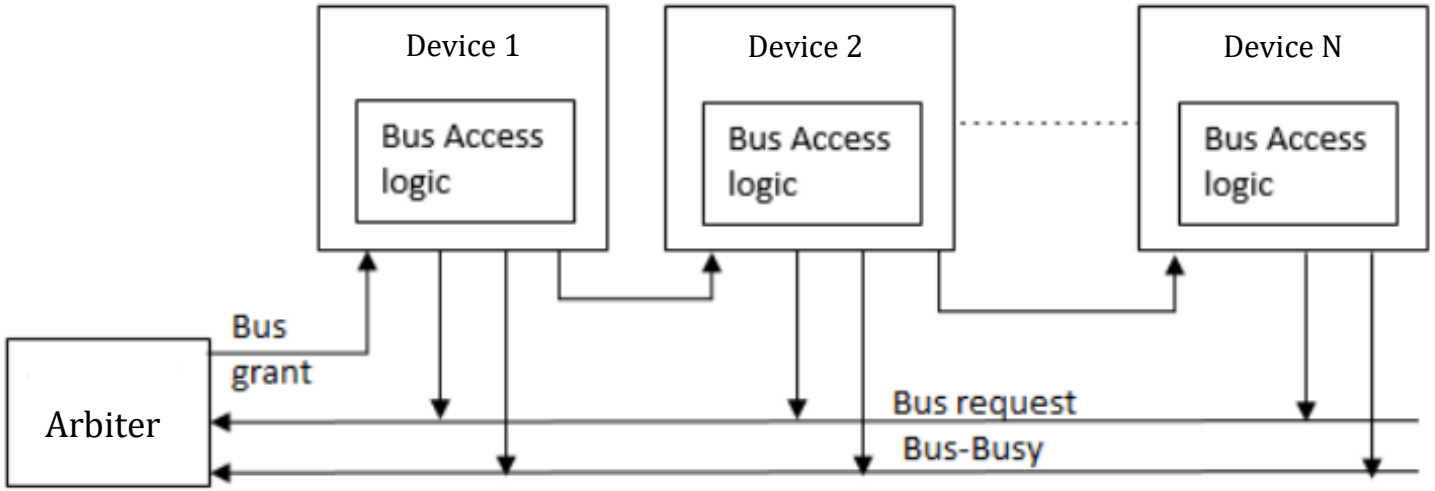
Bus Arbitration Schemes

- Centralized: Only single bus arbiter performs the required arbitration and it can be either a processor or a separate controller.
  - Daisy chain arbitration
  - Centralized parallel arbitration

- Distributed
  - Distributed arbitration by self-selection
    - Each device wanting the bus places a code indicating its identity on the bus
  - Distributed arbitration by collision detection
    - Device uses the bus when its not busy and if a collision happens (because some other device also decides to use the bus) then the device tries again later (Ethernet)

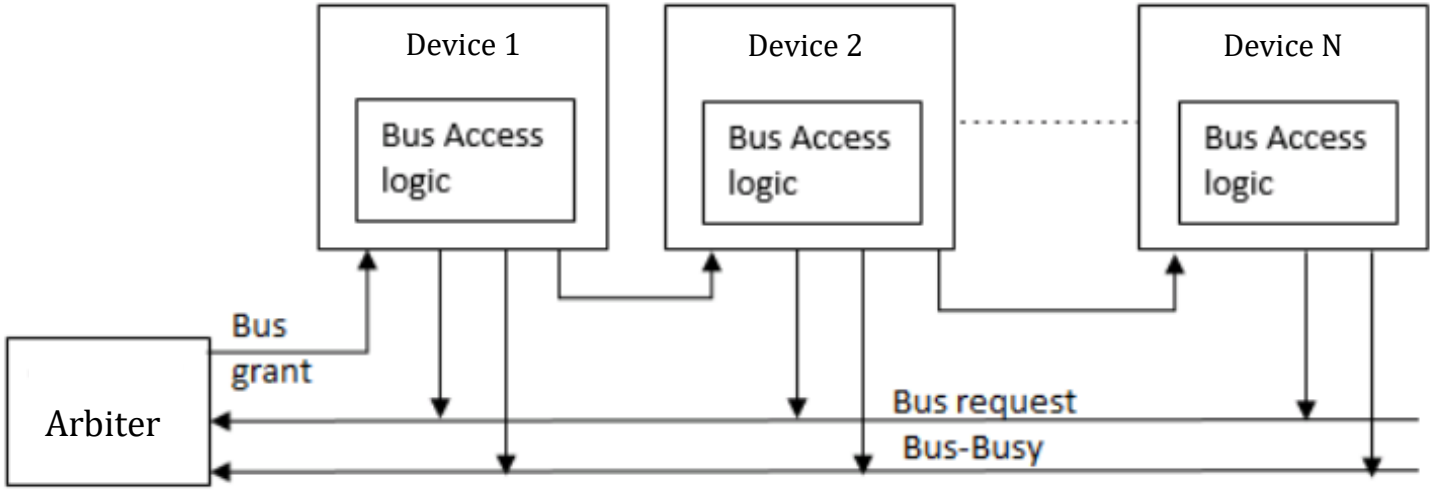
The Daisy Chain Arbitration

- The daisy chain consists of wires connecting the devices in a pre-defined order
- This order (position) of master effectively sets the priority of the devices



The Daisy Chain Arbitration

- All bus master use the same line for bus request.
- If the bus busy line is inactive, the bus controller gives the bus grant signal.
- Bus grant signal is propagated serially through all masters starting from nearest one.
- The bus master which requires system bus, stops this signal, activates the bus busy line and takes control of system bus.



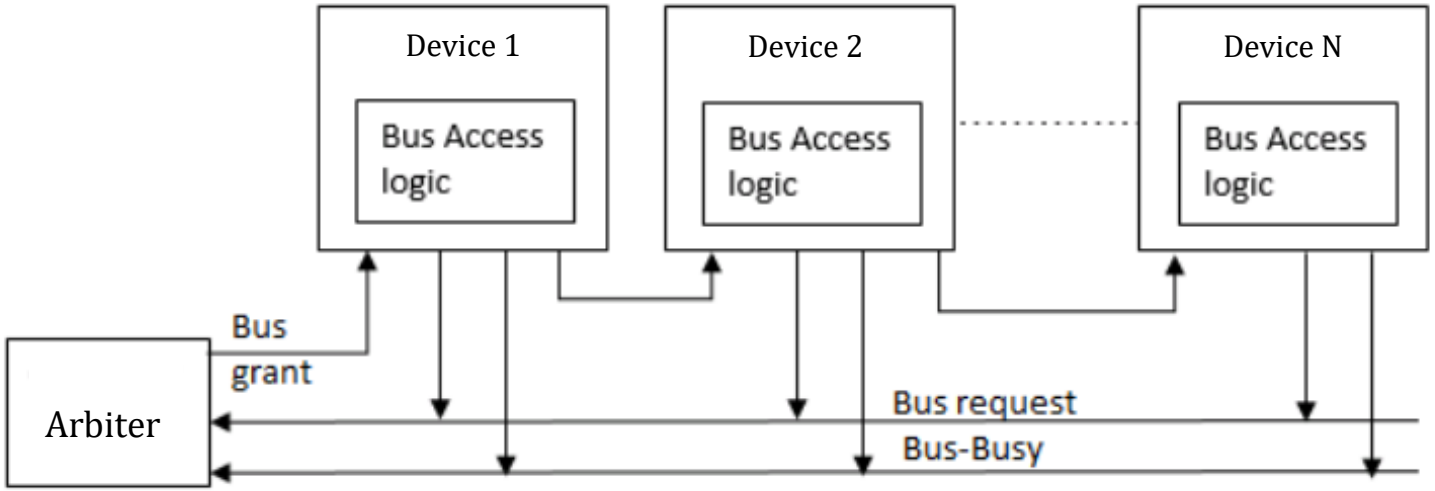
The Daisy Chain Arbitration

Advantages:

- Simple design
- Less no. of control lines.

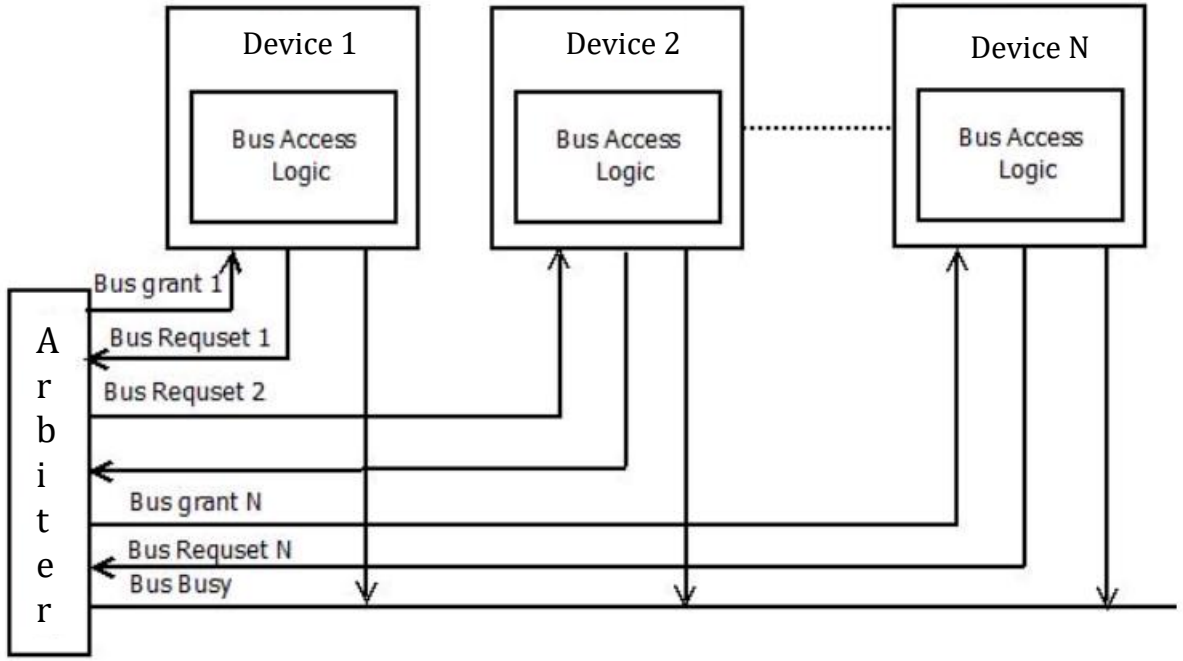
Disadvantages:

- Priority depends on the physical location of master.
- Propagation delay due to serially granting of bus.
- Failure of one of the devices may fail entire system.
- Fairness cannot be guaranteed



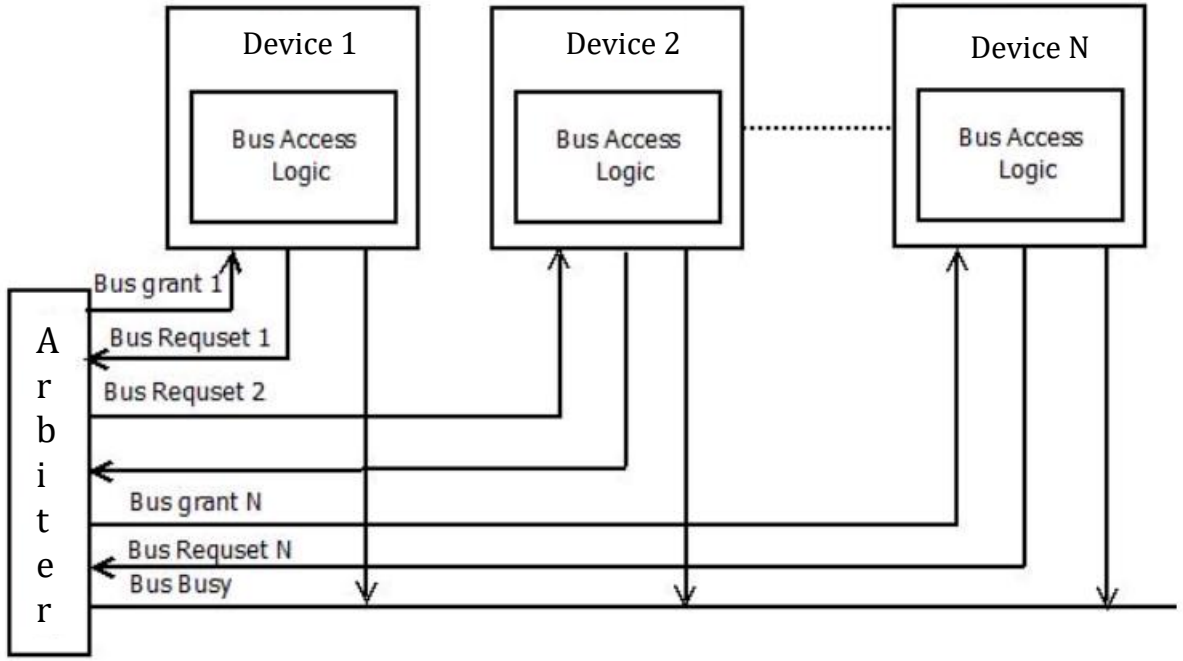
The Centralized Parallel Arbitration

- All bus masters have their individual bus request and bus grant lines
  - The arbiter thus knows which master has requested
- Priorities of the masters are predefined (or set dynamically) so on simultaneous bus requests, the bus is granted based on the priority, provided the bus busy line is not active



The Centralized Parallel Arbitration

- Also known as: Independent request arbitration
- **Advantages:**
  - Bus arbitration is fast
  - Speed independent of no. of devices connected
- **Disadvantage:** No. of control lines required is more. Hence connecting a large number of bus masters is difficult
- Example: PCI bus



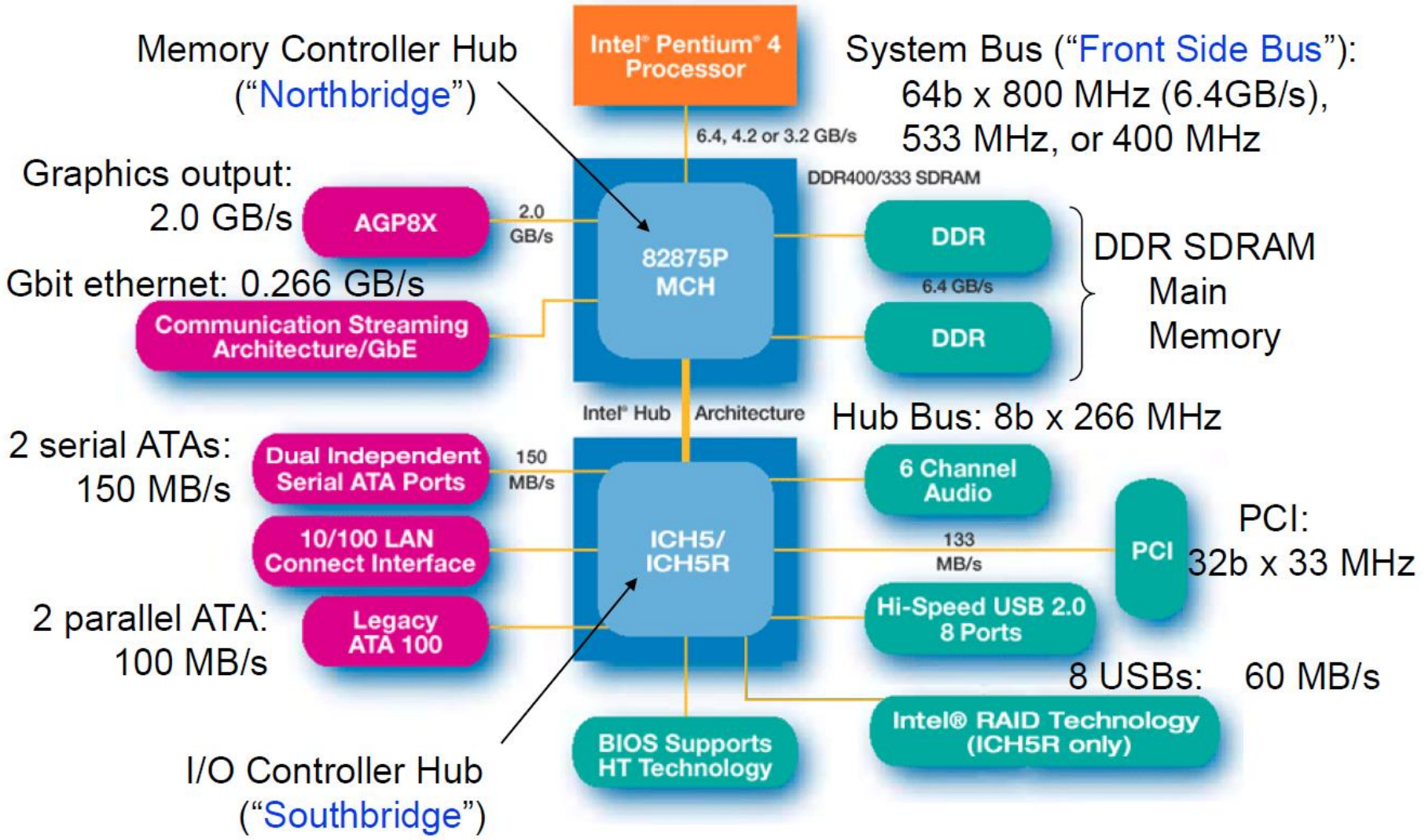
Distributed Arbitration by Self-selection

- Each device can send their identities and requests through several control lines
- Each device can see which other devices are also requesting
- Priorities (identities) of different devices are known to each other
- All devices collectively decides who gets the bus based on priorities
- Example: NuBus in Mac

Distributed arbitration by Collision Detection

- Each device can see if the bus is free
- If multiple device try to access bus, there will be collision
- If there is collision, back off
  - Try again after some time
  - Random delay for different devices, so that colliding devices do not try again at the same time
- If no collision, use the bus
- Example: ethernet

Example: Pentium 4 Buses



Technology in Existence

ATA: AT Attachment

- 16 bits of data in parallel
- Peak bandwidth: 133MB/s

SCSI: Small Computer System Interface

- Synonymous with high-end IO
- Fast bus speeds: up to 160Mhz QDR (four data transfers per clock)
- Many variants up to SCSI Ultra-640: 640MB/s
- Scalable: up to 16 devices per SCSI bus
- Expensive

Technology in Existence

Peripheral Component Interconnect (PCI)/ PCI Express

- The fastest general-purpose expansion option
- Graphics cards
- Network cards
- High-performance disk controllers (RAID)
- Current generation in PCI Express (PCIe)

The Serial Revolution

- Wider busses: increased bandwidth
- Problems: crosstalk, clock skew
- If we have n lines in a bus, you need to wait for the slowest one
- All devices must use the same clock
- This limits bus speeds
- High speed serial lines have been replacing wide buses

High Speed Serial: Differential Signaling

- Two wires carry different voltages $+V$ and $-V$
- Decision of $0/1$ as bit value is made based on the voltages carried by the pair of wires
- If voltage 1> voltage 2, bit value 1 is carried by the pair of wires
- If voltage 1< voltage 2, bit value 0 is carried by the pair of wires
- Max bandwidth per pair $\sim$6Gb/s
- More immune to noise

Serial Interfaces

- USB: Universal Serial Bus
  - Replaces Serial and parallel ports
  - USB 2.0: Single differential pair
  - USB 4.0: Up to 40 Gb/s
- Firewire
  - IEEE 1394
  - 400 Mb/s
- SATA: Serial ATA
- SAS: Serial attached SCSI
- PCIe (PCI Express)

Parallel vs Serial

	PCI	PCIexpress	ATA	Serial ATA
Total # wires	120	36	80	7
# data wires	32 – 64 (2-way)	2 x 4 (1-way)	16 (2-way)	2 x 2 (1-way)
Clock (MHz)	33 – 133	635	50	150
Peak BW (MB/s)	128 – 1064	300	100	375 (3 Gbps)

Commutation with the Peripheral Devices

- Considerations:
  - The format of the data must be understood by the I/O device
    - A display device and a speaker works with different format of data
  - Interface hardware of the device
  - Timing information
  - System software takes care of these details

Accessing I/O Devices

- An I/O device (device controller): Viewed as a set of registers/ port by CPU

- CPU uses these ports to
  - Write commands and parameters
  - Read status from specific registers
  - Read/ write data

Types of I/O Access

- Memory-mapped I/O
  - Portions of the memory address space are assigned to each I/O device
  - Read and writes to those memory addresses are interpreted as commands to the I/O devices
  - Load/stores to the I/O address space can only be done by the OS

- Port-mapped I/O (Also called I/O mapped I/O)
  - Separate dedicated address space
  - Special class of CPU instructions designed specifically for performing I/O
  - Must specify both the device and the command

How Do I/O Devices Communicate with CPU?

- Polling
  - Controlled by processor
  - The processor periodically checks the status of an I/O device to determine its need for service
  - Wastage of a lot of clock cycles

- Interrupt
  - The I/O device issues an interrupt to indicate that it needs attention
  - Asynchronous in nature
  - No wastage of clock cycle for checking the status

Modes of I/O Data Transfer

Programmed I/O

- Each data transfer is initiated by an I/O instruction written in the program

- Upon execution of the instruction, the processor gets direct control of the I/O operation
  - Sensing device status
  - Sending a read or write command
  - Transferring the data

- When the processor issues a command to the I/O module, it must wait until the I/O operation is complete

- If the processor is faster than the I/O module, this is wasteful of processor time

Modes of I/O Data Transfer

Interrupt Driven I/O

- Asynchronous with respect to instructions
- The CPU issues a command to the I/O
  - The CPU keeps on doing other useful works until the I/O device is ready
- Once the I/O device is ready to exchange data, it issues an interrupt request
- Upon receipt of interrupt
  - The CPU executes data transfer
  - Subsequently resumes normal CPU operation

Interrupt Service Routine (ISR)

- Also known as interrupt handler
- It is a software routine that hardware invokes in response to an interrupt
- ISR examines an interrupt and determines how to handle it

Vectored Interrupts: Interrupt Vector Table

- Devices that use vectored interrupts are assigned interrupt vectors
- Interrupt vector: address of ISR corresponding to an interrupt
- Interrupt vector table (IVT)
  - A data structure consisting of interrupt requests and corresponding interrupt vectors
- A device requesting an interrupt needs to identify itself using a special signal
- This information is used as a pointer to IVT for finding ISR

Vectored Interrupts

- When the a device interrupts the CPU, it finds location of ISR from IVT
- Subsequently, CPU branches to the particular ISR
- When the CPU branches, it saves the address of the next instruction on the stack.
- At the end of the service routine, the CPU returns the execution to where the program was interrupted

Non-vectored Interrupts

- The CPU does not know which device caused the interrupt
- To know this information, polling each I/O interface is required
- The address of the ISR is sent along with the interrupt

Modes of I/O Data Transfer

Programmed I/O and Interrupt Driven I/O

- Requires intervention of CPU

- Alongside other factors, I/O transfer rate also depends on how fast a processor can service an I/O device

Modes of I/O Data Transfer

Direct Memory Access (DMA)

- The DMA controller can transfer data directly to/from memory without involving the CPU

- CPU initiates the DMA transfer by supplying
  - I/O device address
  - The operation to be performed
  - The memory address of destination/source
  - The number of bytes to transfer

- DMA controller takes over the control of buses and performs the transfer

- Once complete, DMA controller interrupts the CPU to notify the completion of I/O

DMA Stale Data Problem

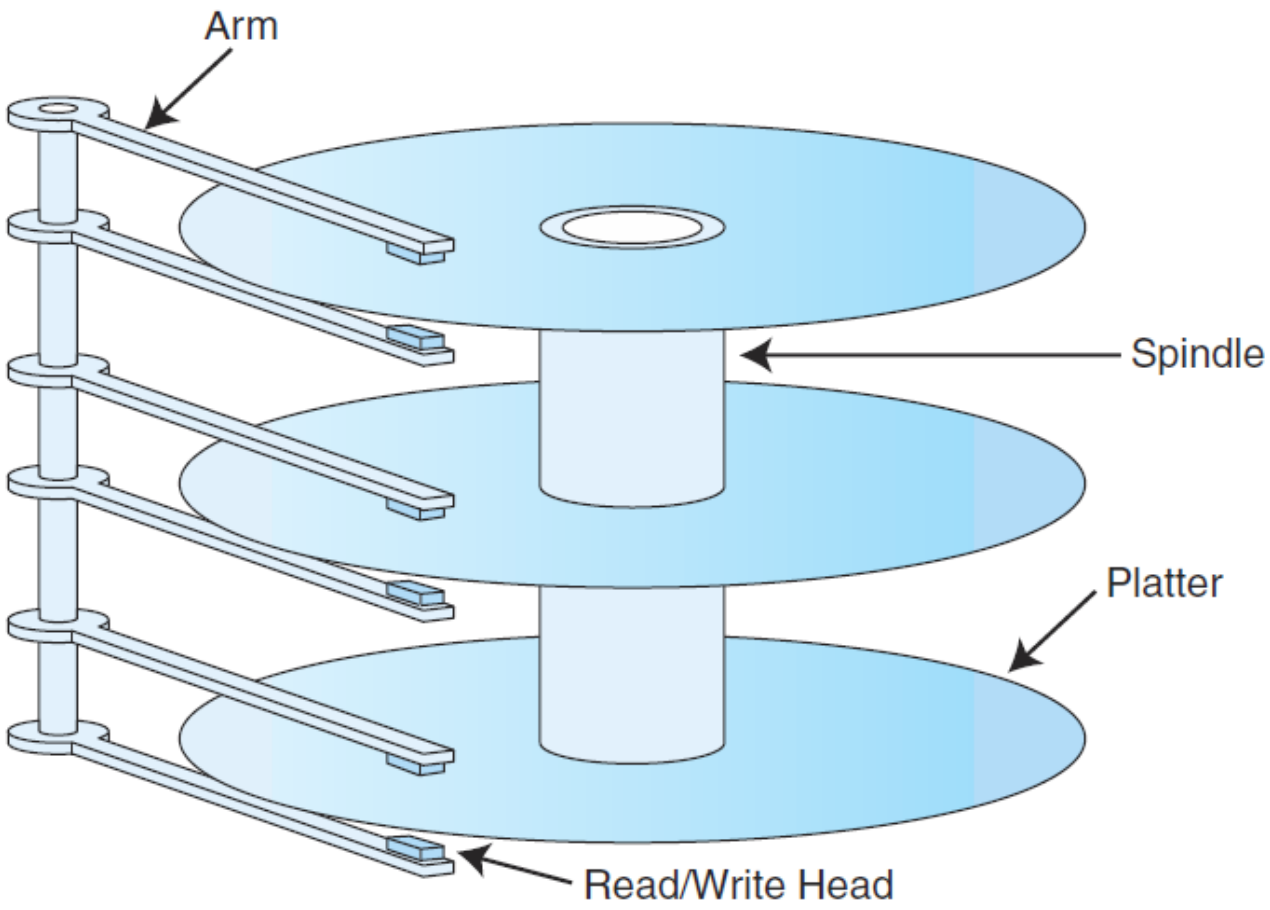
- In systems with caches, there can be two copies of a data item
  - One in the cache
  - One in the main memory
- For a DMA read (from disk to memory)
  - The processor will be using stale data if that location is also in the cache
- For a DMA write (from memory to disk) and a write-back cache
  - The I/O device will receive stale data if the data is in the cache and has not yet been written back to the memory

DMA Stale Data Problem

The coherency problem may be solved by

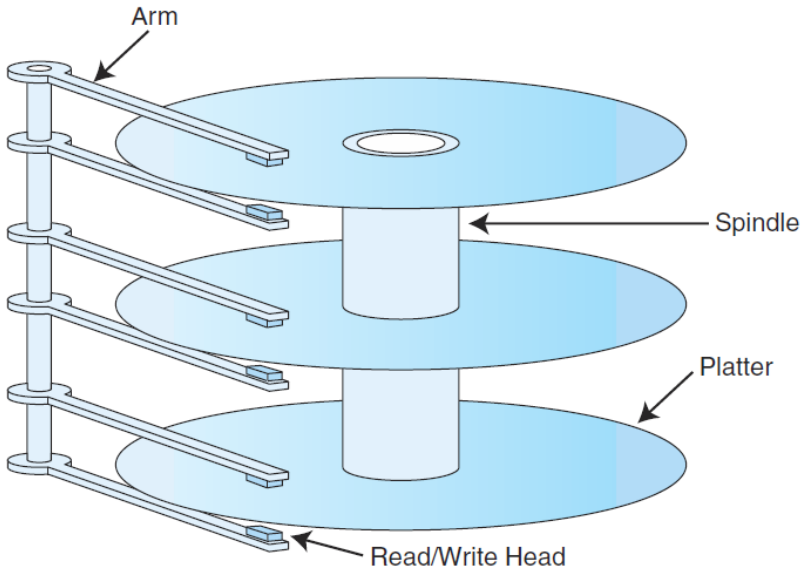
1. Routing all I/O activity through the cache
   - Expensive and a large negative performance impact
2. Having the OS selectively invalidate the cache for an I/O read or force write-backs for an I/O write (flushing)
3. Providing hardware to selectively invalidate or flush the cache
   - Need a hardware snooper

Peripherals: Hard Disk Drive



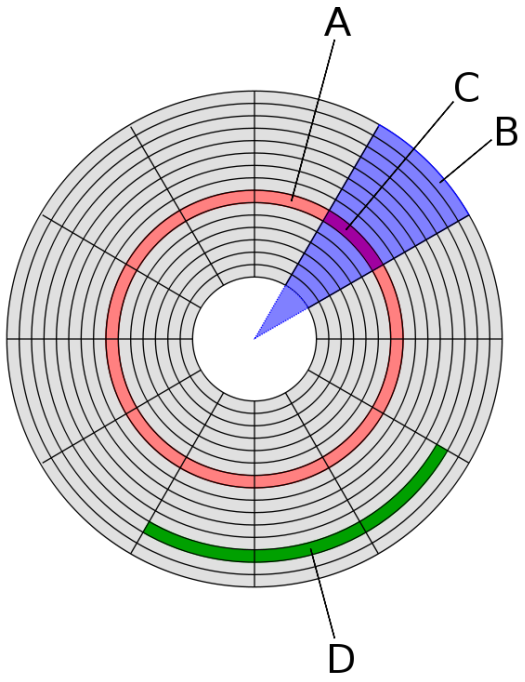
Peripherals: Hard Disk Drive

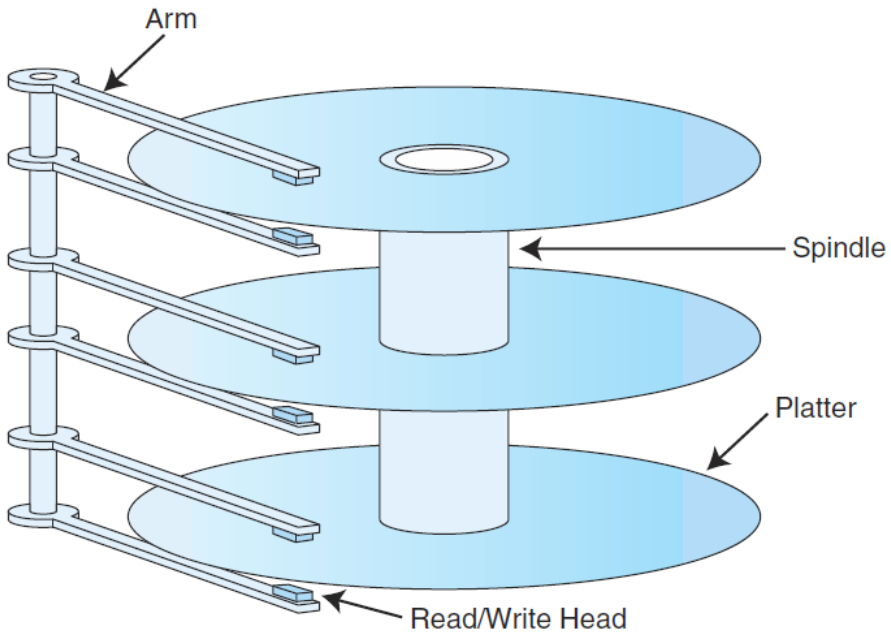
- Platters rotate
- Patter is covered with magnetic film
- Each platter may store information on both sides
- Information is stored in the surface of the platter
- Information (bit value 0/1) is stored using magnetization
- Bit value 0/1 depends on polarity of magnetization



https://en.wikipedia.org/wiki/Hard_disk_drive_platter

Peripherals: Hard Disk Drive



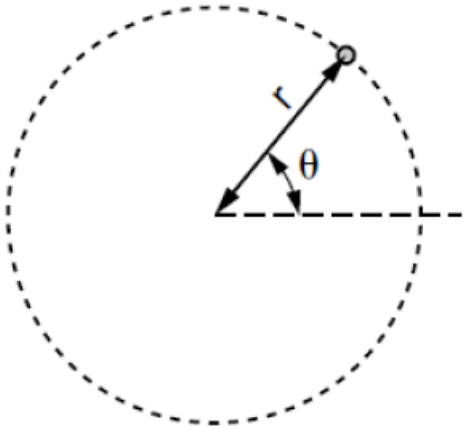


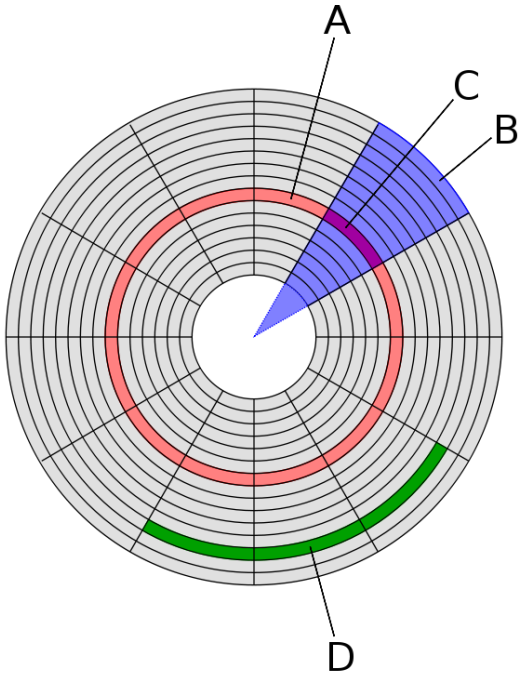
- Circular tracks on the disks
- Read write head may move close to center or towards the periphery
  - From track to track (A in the figure)

https://en.wikipedia.org/wiki/Track_(disk_drive)

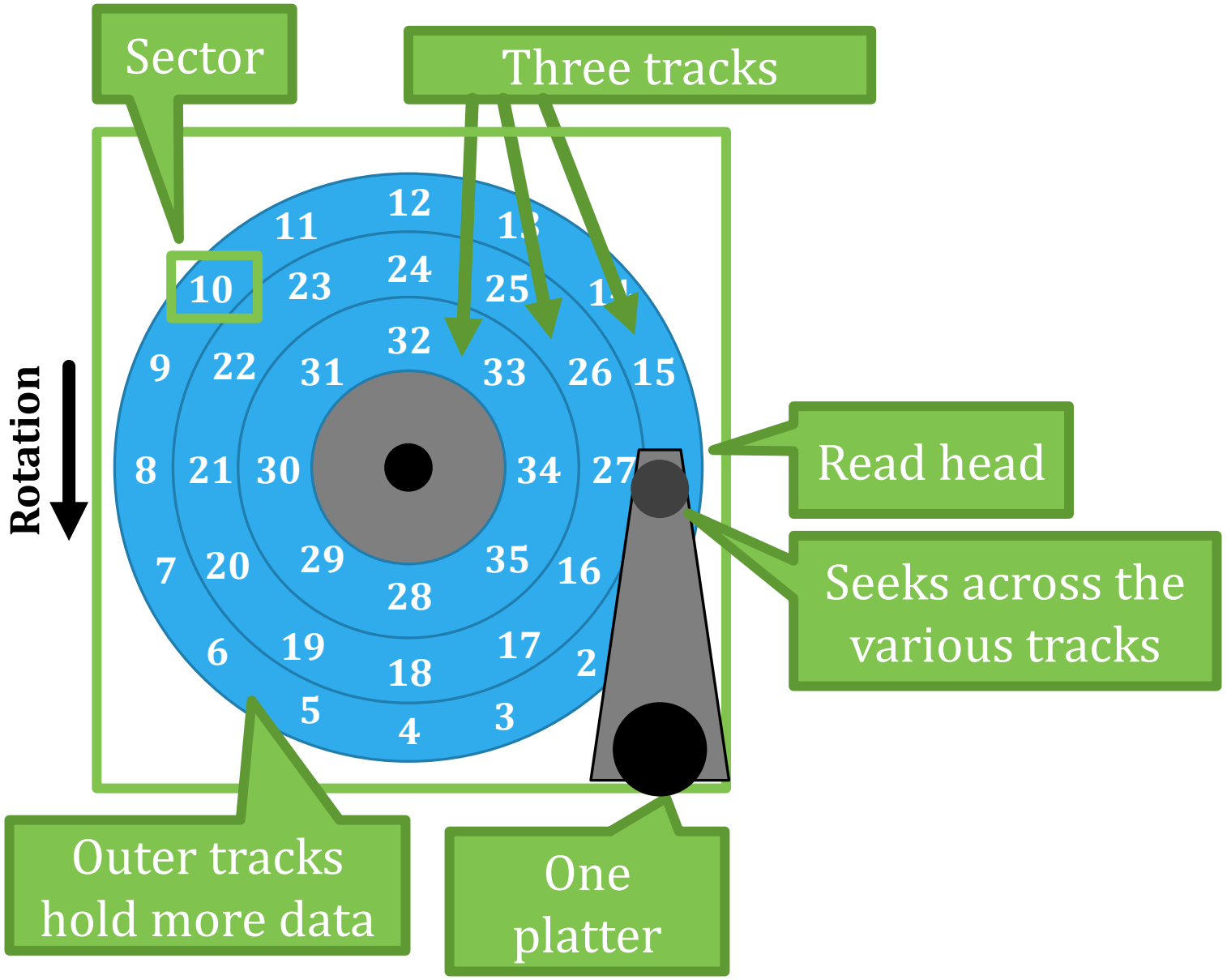
Peripherals: Hard Disk Drive

- A: Track
- C: Track Sector
  - A track is a big area to store data
  - Hence tracks are divided into sectors
- B: Disk Sector
  - Collection of track sectors in a specific angular position
- D: Cluster
  - A group of track sectors
  - The grouping by which disk files are organized





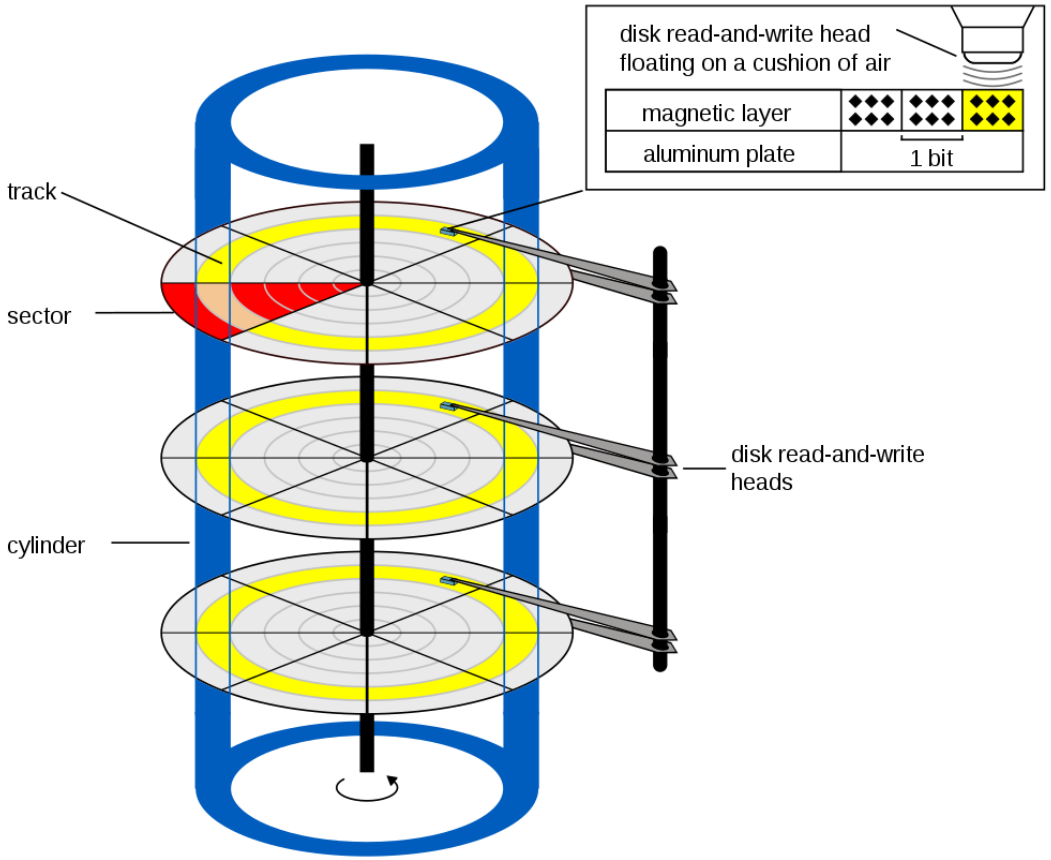
Geometry Example



Peripherals: Hard Disk Drive

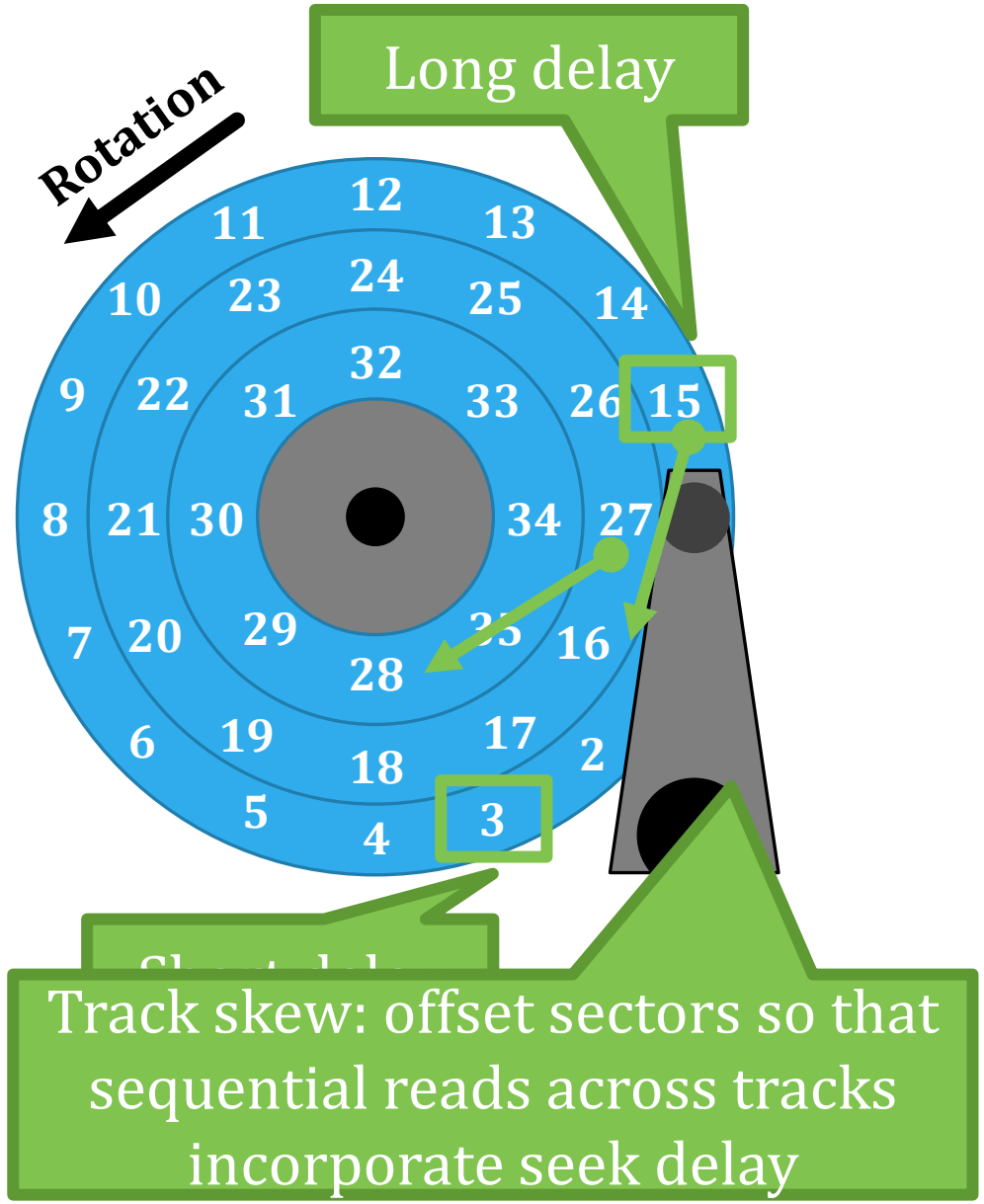
Cylinder

- Same tracks of different platters form an imaginary cylinder like structure
- Data is stored cylinder by cylinder
- All tracks on a cylinder are written and then the R/W head moves to the next cylinder
- This reduces movement of R/W head and increases the speed of read and write operations



https://en.wikipedia.org/wiki/Cylinder-head-sector

Types of Delay With Disks



1. Rotational Delay
   - Time to rotate the desired sector to the read head
   - Related to RPM
2. Seek delay
   - Time to move the read head to a different track
3. Transfer time
   - Time to read or write bytes

Time to Read/ Write a Sector

- Three components
  - Seek time
    - Entire seek often takes several milliseconds (4–10 ms)
  - Rotation time
    - Depends on disk's rotational speed: Rotations Per Minute (RPM)
    - 7200 RPM is common, 15000 RPM is high end
    - For 7200 RPM, we will have 8.3 ms / rotation
    - Time for half rotation: 4.15 ms
  - Transfer time
    - Transfer is fast
    - More than 100 MB/s is typical for maximum transfer rate
- Time to read/ write = seek time + rotation time + transfer time

Workload Performance

- Seeks are slow
- Rotations are slow
- Transfers are fast

- What kind of workload is faster for disks?
  - Sequential: access sectors in order (transfer dominated)
  - Random: access sectors arbitrarily (seek + rotation dominated)

Workload Performance

	Cheetah	Barracuda
Capacity	300 GB	1 TB
RPM	15,000	7,200
Avg Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s
Platters	4	4
Cache	16 MB	32 MB

- Throughput for sequential workload
  - Cheetah: 125 MB/s
  - Barracuda: 105 MB/s

https://compas.cs.stonybrook.edu/~nhonarmand/courses/fa17/cse306/slides/15-disks.pdf

Workload Performance

	Cheetah	Barracuda
Capacity	300 GB	1 TB
RPM	15,000	7,200
Avg Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s
Platters	4	4
Cache	16 MB	32 MB

- Throughput for random workload
  - Assume size of each random read is 16KB

Workload Performance

	Cheetah	Barracuda
RPM	15,000	7,200
Avg Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s

- Time to read/ write = seek time + rotation time + transfer time
- Seek = 4 ms
- Full rotation = 60 / 15,000 = 4 ms
  - Half rotation = 2 ms
- Transfer = 16 KB / 125 MBps = 128 us
- Total time required= (4+2+0.128) ms =6.128 ms

Workload Performance

	Cheetah	Barracuda
RPM	15,000	7,200
Avg Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s

- Time to read/ write = seek time + rotation time + transfer time
- Seek = 4 ms
- Full rotation = 60 / 15,000 = 4 ms
  - Half rotation = 2 ms
- Transfer = 16 KB / 125 MBps = 128 us
- Total time required= (4+2+0.128) ms =6.128 ms
- Throughput = 16 KB / (6.128 ms) = 2.6 MBps

Assume size of each random read is 16KB

Workload Performance

	Cheetah	Barracuda
RPM	15,000	7,200
Avg Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s

- Time to read/ write = seek time + rotation time + transfer time
- Seek = 9 ms
- Full rotation = 60 / 7,200 = 8.3 ms
  - Half rotation = 4.15 ms
- Transfer = 16 KB / 105 MBps = 152 us
- Total time required= (9+4.15+0.152) ms =13.302 ms

Workload Performance

	Cheetah	Barracuda
RPM	15,000	7,200
Avg Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s

- Time to read/ write = seek time + rotation time + transfer time
- Seek = 9 ms
- Full rotation = 60 / 7,200 = 8.3 ms
  - Half rotation = 4.15 ms
- Transfer = 16 KB / 105 MBps = 152 us
- Total time required= (9+4.15+0.152) ms =13.302 ms
- Throughput = 16 KB / (13.302 ms) = 1.2 MBps

Assume size of each random read is 16KB

Challenges of Hard Disk Drive

- Hard drives are relatively fast, persistent storage
- Shortcomings:
  - How to cope with disk failure?
    - Mechanical parts break over time
    - Sectors may become silently corrupted
  - Capacity is limited
    - Managing files across multiple physical devices is cumbersome

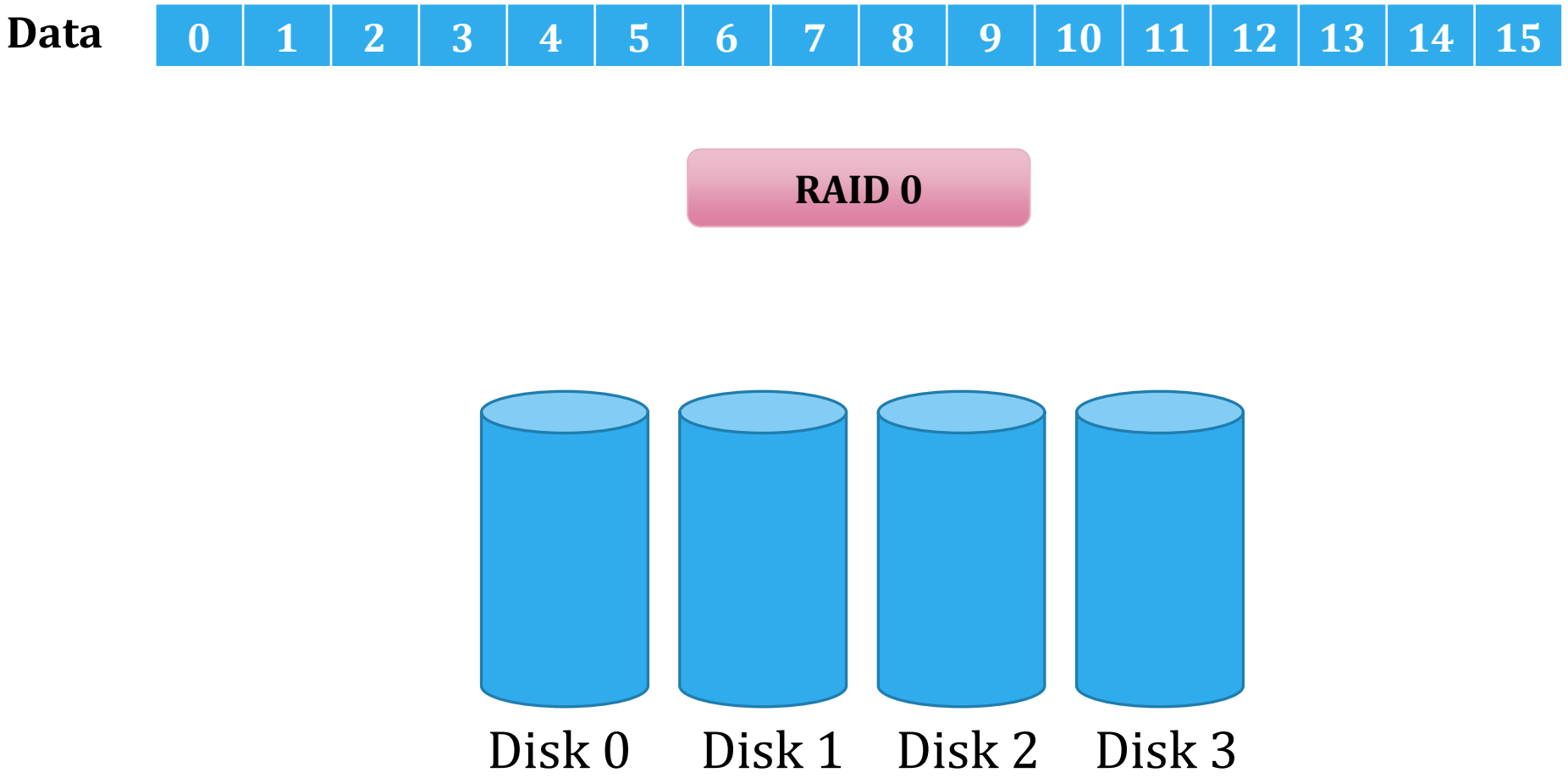
https://en.wikipedia.org/wiki/Cylinder-head-sector

Redundant Array of Independent Disks (RAID)

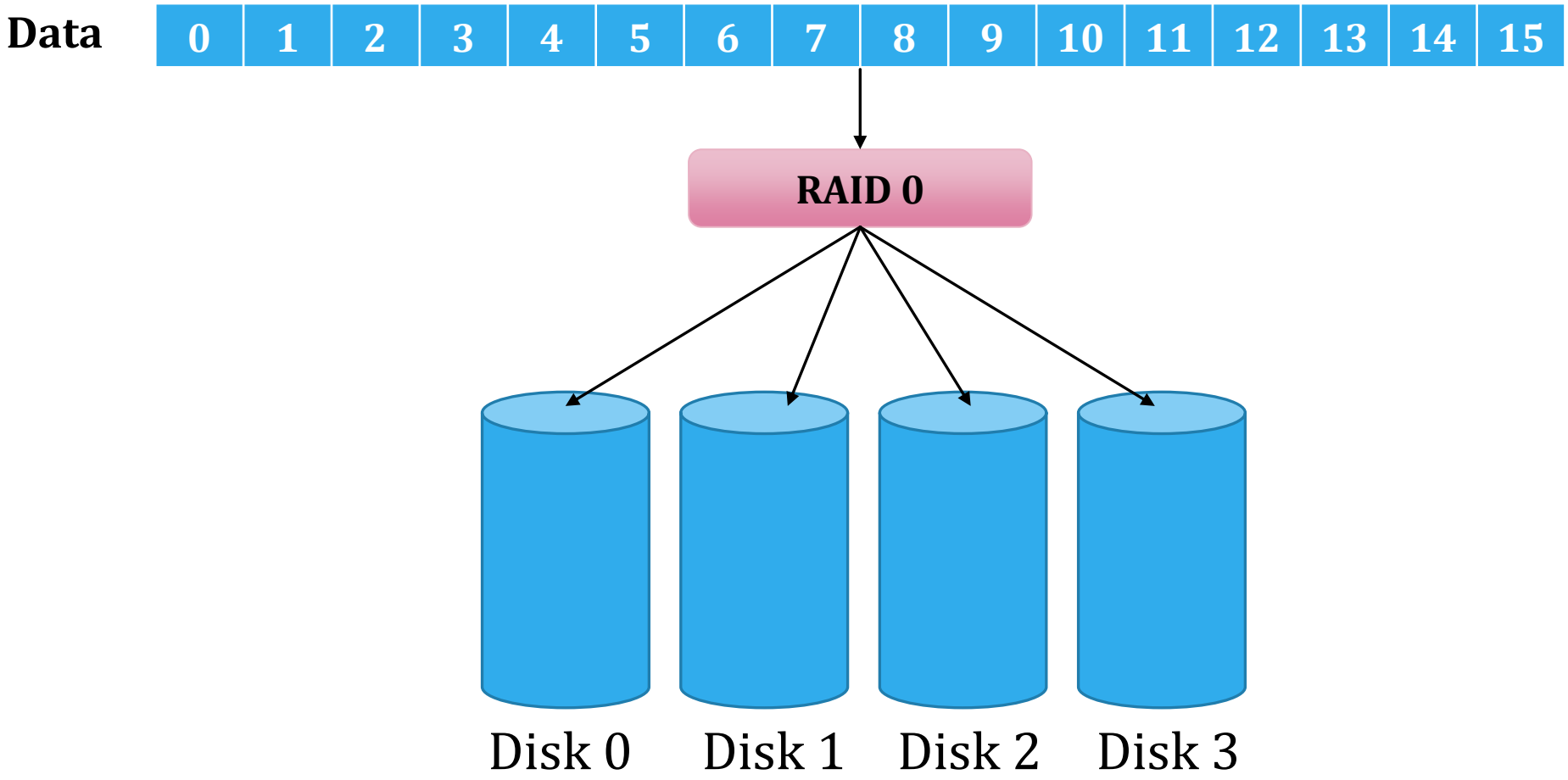
- Use of multiple disks to create an illusion of a large, faster, more reliable disk
- Externally, RAID looks like a single disk
  - i.e. RAID is **transparent**
  - Data blocks are read/written as usual
  - No need for software to explicitly manage multiple disks or perform error checking/ recovery
- Internally, RAID is a complex computer system
  - Disks managed by a dedicated CPU + software

RAID 0

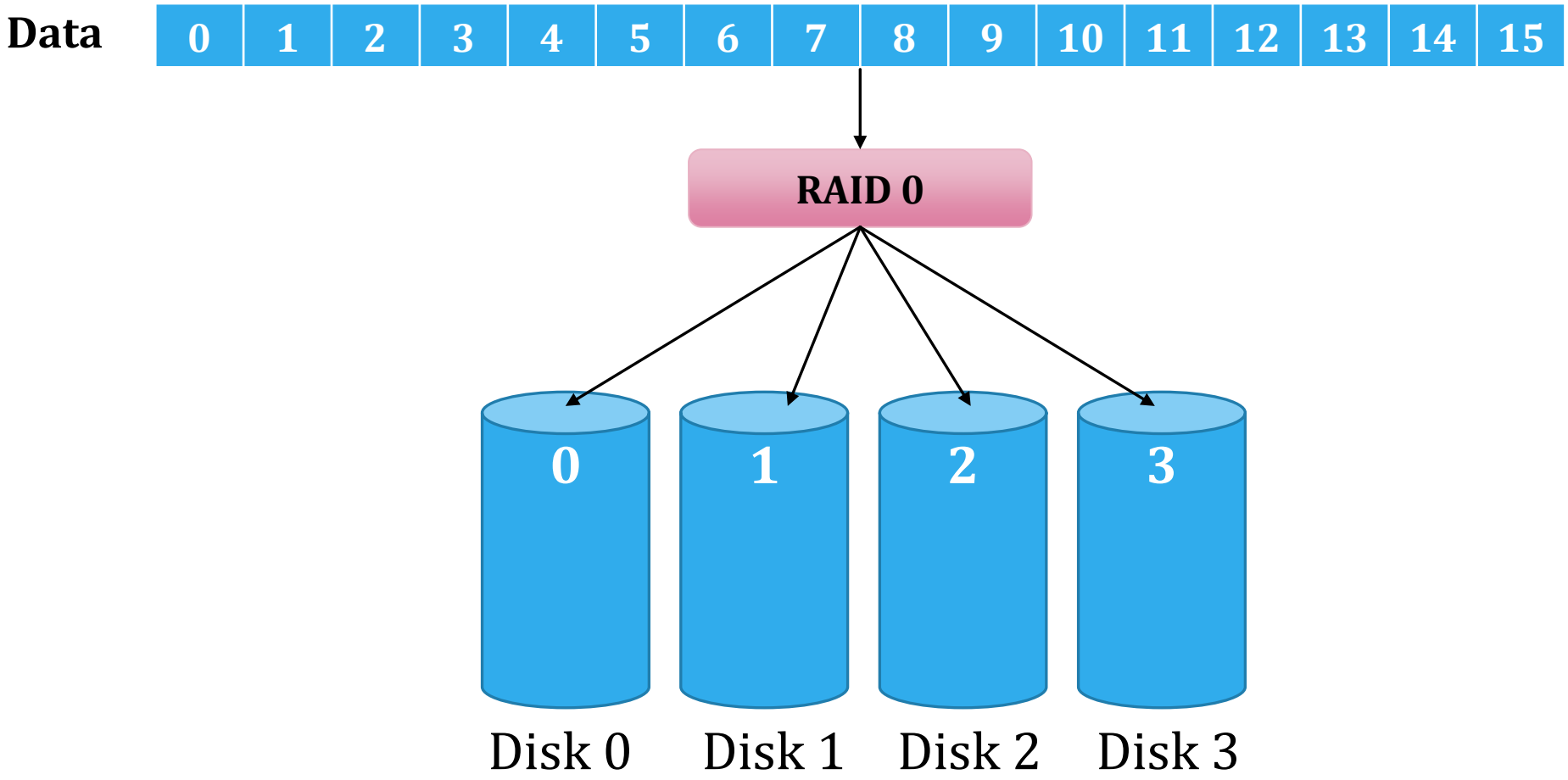
- Striping



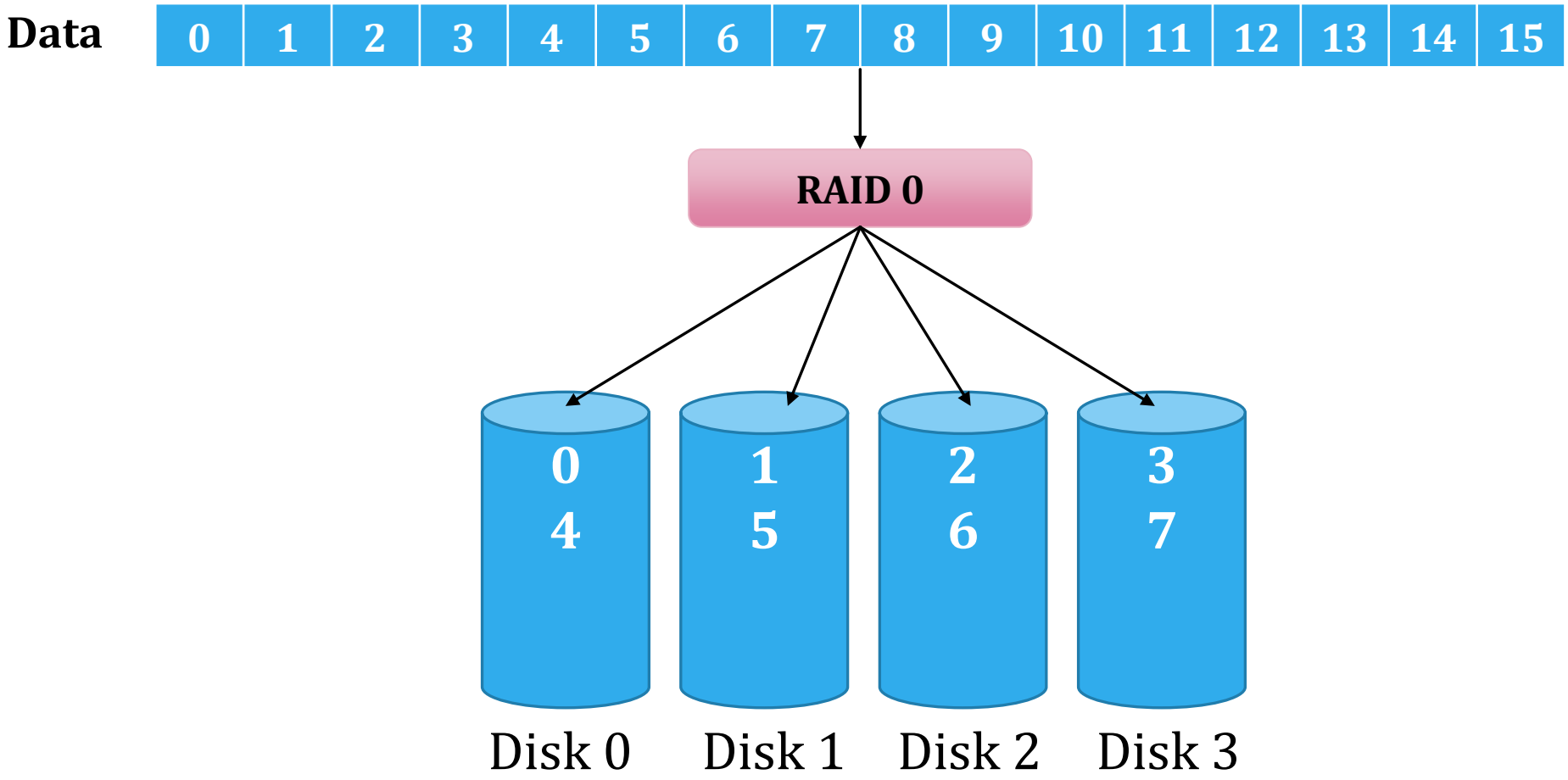
RAID 0


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

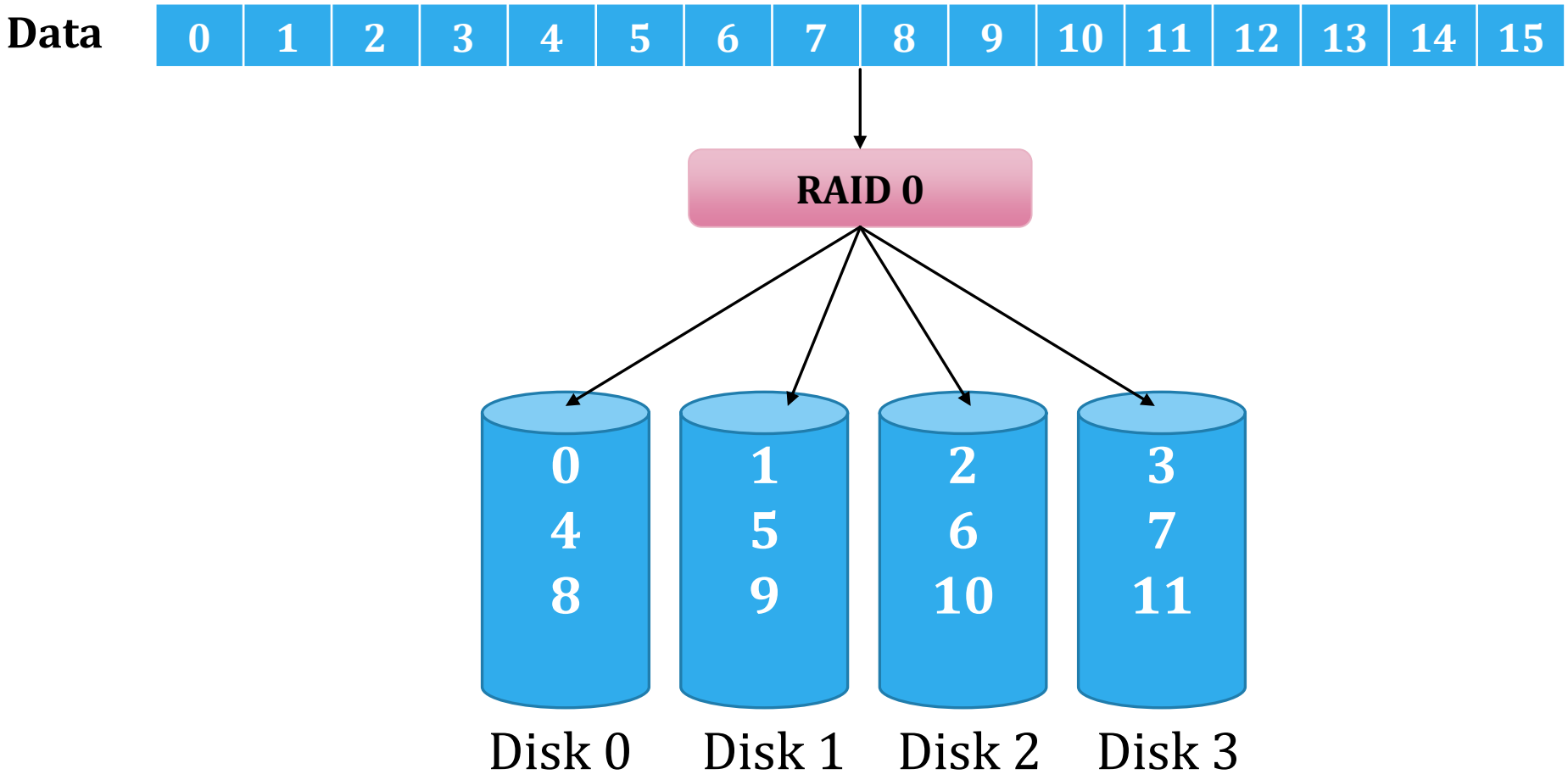
RAID 0


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

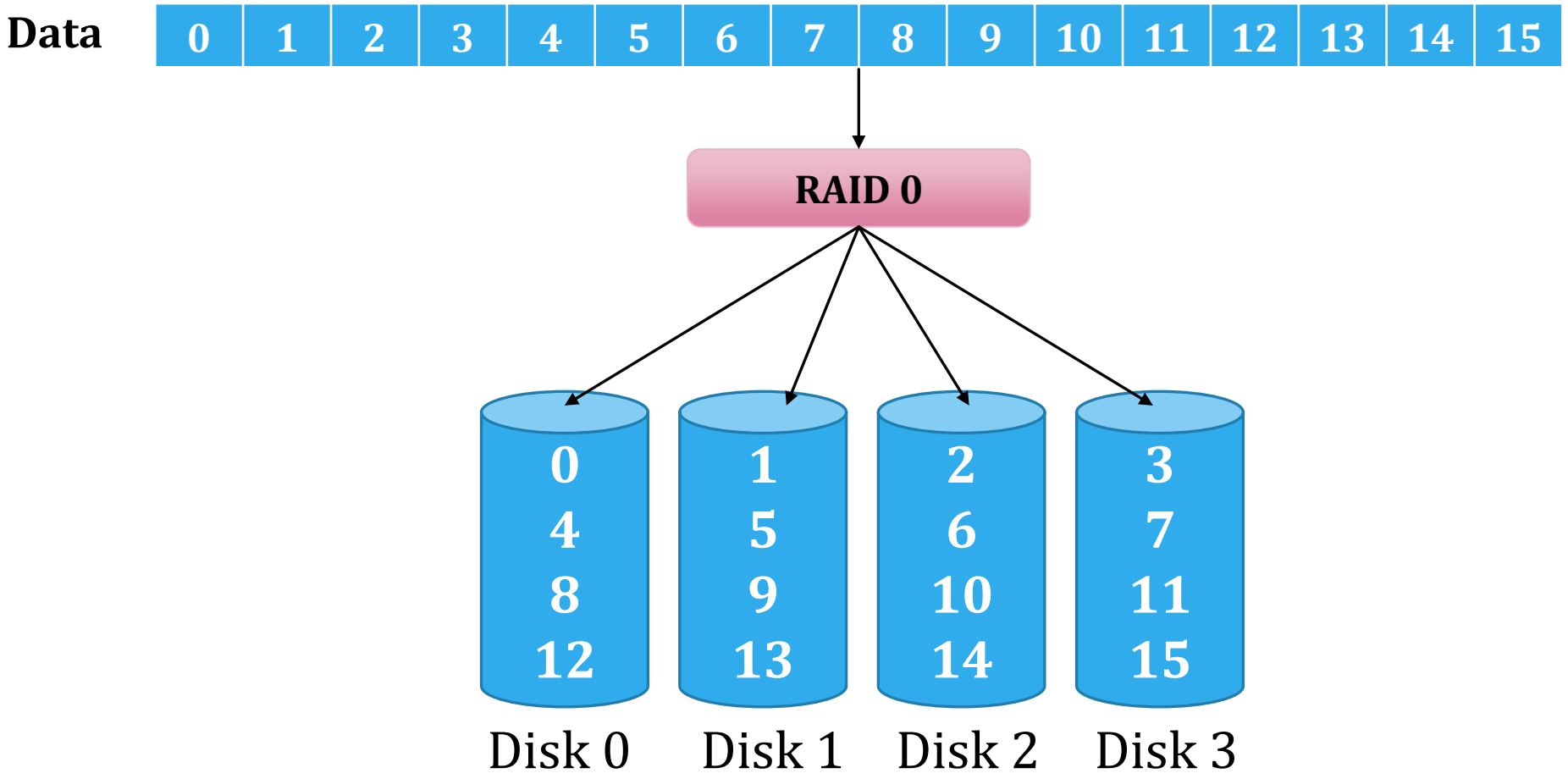
RAID 0


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

RAID 0


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

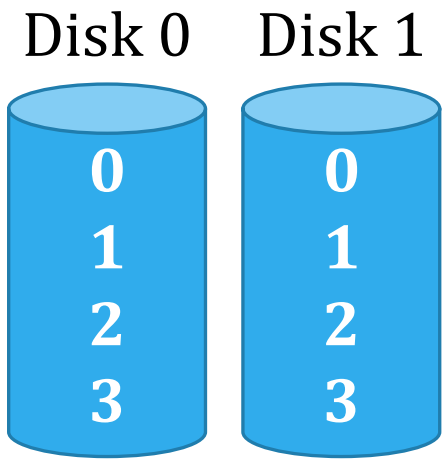
RAID 0


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

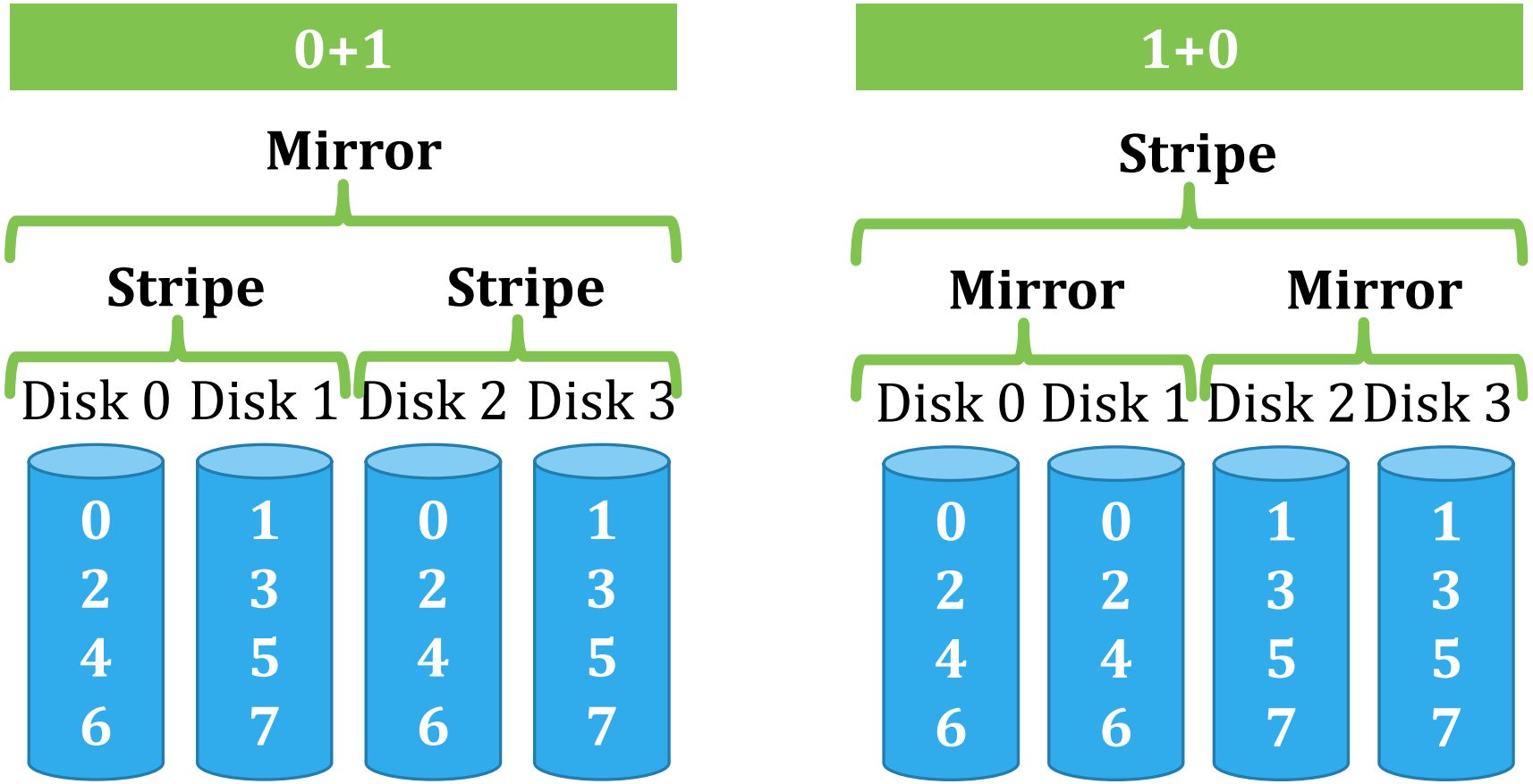
RAID 1: Mirroring

- RAID 0 offers high performance, but zero error recovery

- Key idea in RAID 1: make two copies of all data



RAID 0+1 and 1+0 Examples

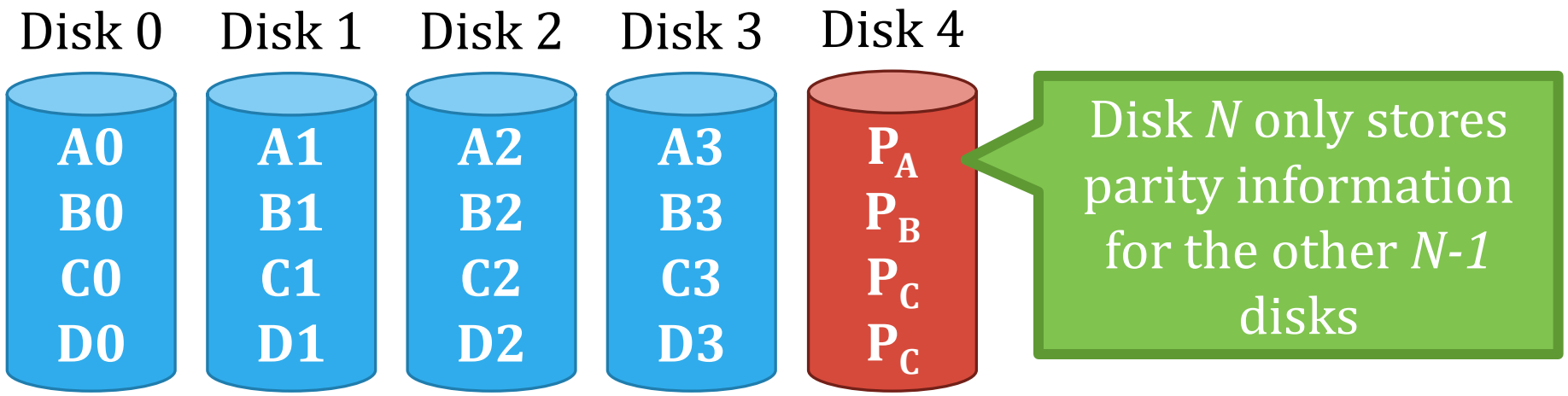


- Combines striping and mirroring
- Superseded by RAID 4, 5, and 6

Decreasing the Cost of Reliability

- RAID 1 offers highly reliable data storage
- But it results in wastage of storage capacity
- Can we achieve the same level of reliability without wasting so much capacity?
  - Yes!
  - Use information coding techniques to build light-weight error recovery mechanisms

RAID 4: Parity Drive



Disk 0	Disk 1	Disk 2	Disk 3	Disk 4
0	0	1	1	0 ^ 0 ^ 1 ^ 1 = 0
0	1	0	0	0 ^ 1 ^ 0 ^ 0 = 1
1	1	1	1	1 ^ 1 ^ 1 ^ 1 = 0
0	1	1	1	0 ^ 1 ^ 1 ^ 1 = 1

Parity calculated using XOR

https://cbw.sh/static/class/5600/slides/9_Storage_Devices.pptx

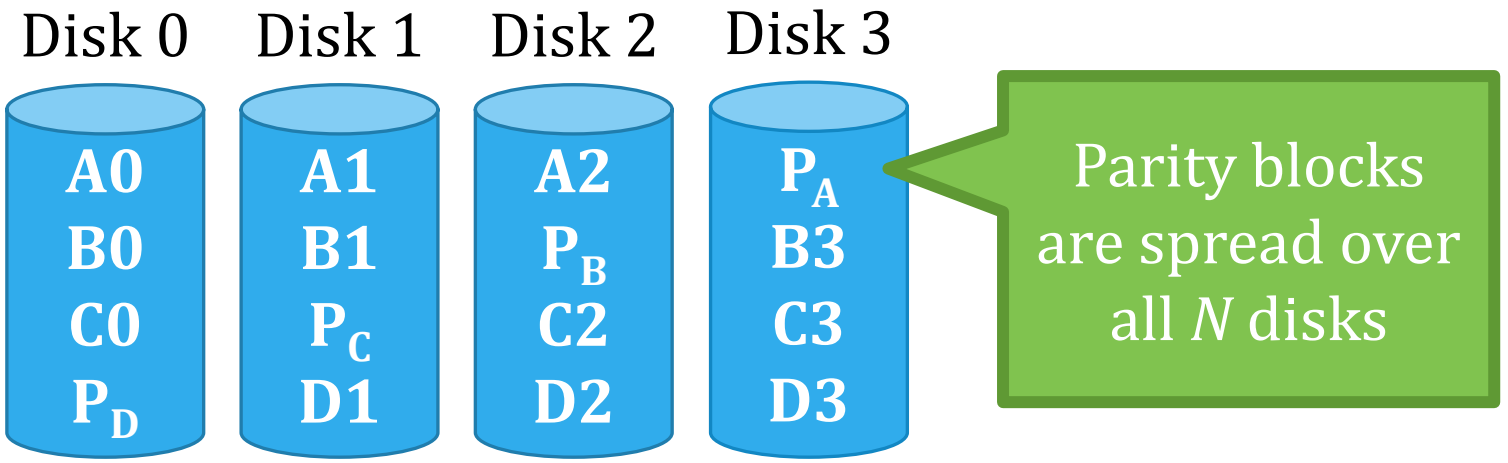
RAID 4: Analysis

- If we want to read block A0 from disk 0
  - Read it
  - If there is an error in the disk, read in blocks A1, A2, A3, and parity block and calculate correct data

- Random writes in RAID 4
  1. Read the target block and the parity block
  2. Calculate the new parity block
  3. Write the target block and the parity block

- RAID 4 has terrible write performance
  - Bottlenecked by the parity drive

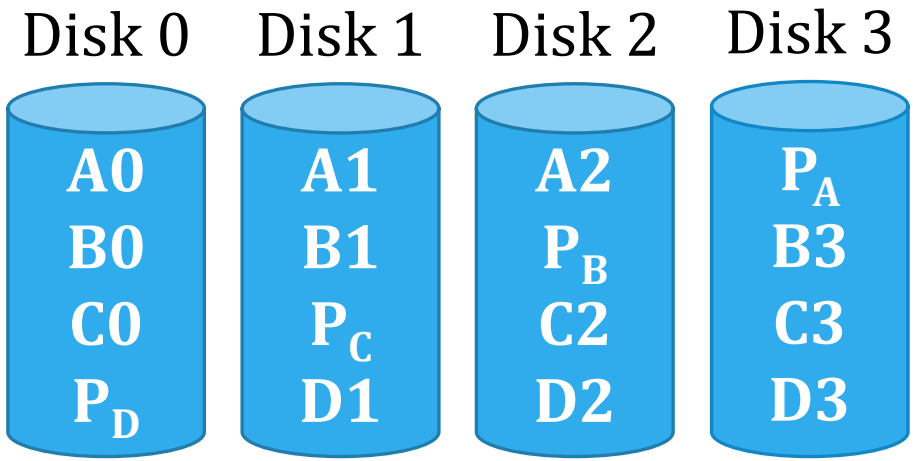
RAID 5: Rotating Parity



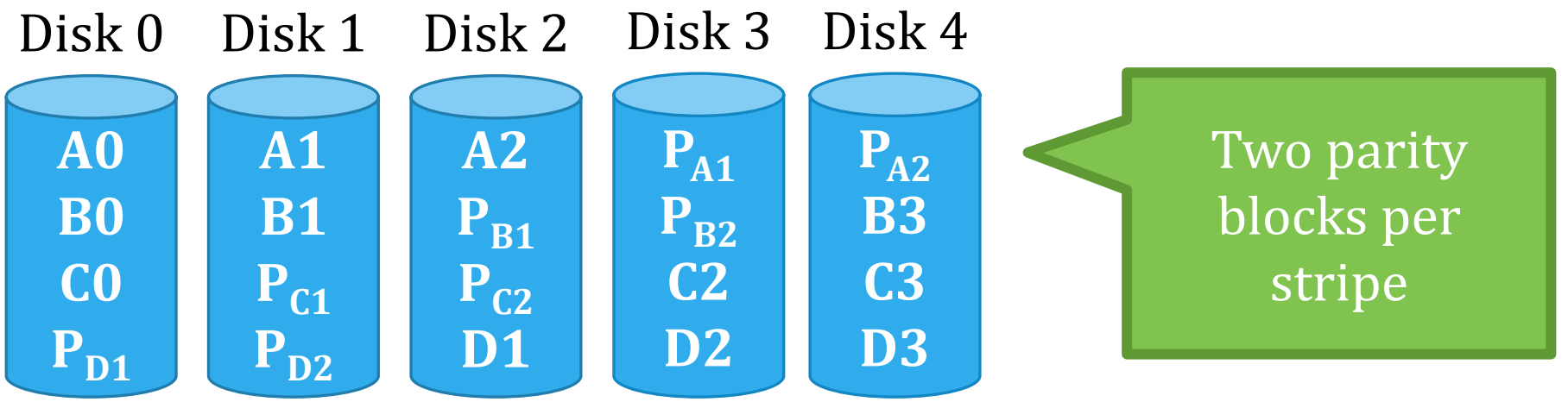
Disk 0	Disk 1	Disk 2	Disk 3
0	1	1	0 ^ 1 ^ 1 = 0
0	0	0 ^ 0 ^ 0 = 0	0
1	1 ^ 1 ^ 1 = 1	1	1
0 ^ 1 ^ 1 = 0	0	1	1

RAID 5: Analysis

- Parity information is spread over all the disk
- No bottleneck due to single parity disk
- Multiple writes may be serviced in parallel
  - For example, writing A0 and B1
- Complex disk controllers
- How many disk failures can it handle?



RAID 6



- We would be able to recover even if any two drives fail
- *N – 2* usable capacity
- Typically implemented using Reed-Solomon codes
- Overhead penalty involved in generating the Reed-Solomon code

Numerical Problems

Question:

Suppose a given bus protocol requires 10 ns for devices to make requests, 15 ns for arbitration, and 25 ns to complete each operation. How many operations can be completed per second?

Numerical Problems

Question:

Suppose a given bus protocol requires 10 ns for devices to make requests, 15 ns for arbitration, and 25 ns to complete each operation. How many operations can be completed per second?

Answer:

The total time for an operation is = (10+15+25) ns = 50 ns.

Operations per second = $\frac{1}{50\times10^{-9}} = 2 \times 10^{7}$

Numerical Problems

Question:

A processor executes 3 billion instructions/s and averages 100,000 OS instructions to handle a disk I/O operation. A disk workload consisting of 64KB reads and writes where the user program executes 200,000 instructions per disk I/O operation. The system has a memory-I/O bus that sustains a transfer rate of 1000 MB/s.

a) Calculate the maximum disk I/O rate (# I/Os per second) of the processor.

b) Calculate the maximum disk I/O rate (# I/Os per second) of the bus.

Numerical Problems

Question:

A processor executes 3 billion instructions/s and averages 100,000 OS instructions to handle a disk I/O operation. A disk workload consisting of 64KB reads and writes where the user program executes 200,000 instructions per disk I/O operation. The system has a memory-I/O bus that sustains a transfer rate of 1000 MB/s.

a) Calculate the maximum disk I/O rate (# I/Os per second) of the processor.

b) Calculate the maximum disk I/O rate (# I/Os per second) of the bus.

Answer:

a) No. of I/Os per second $= \frac{No.of\ instr\ that\ can\ be\ executed\ per\ second}{No.of\ instructions\ required\ for\ an\ I/O} = \frac{3\times10^9}{100000+200000} = 10^4$

The maximum disk I/O rate (# I/Os per second) of the processor is 10^4 I/Os per second

b) Each disk I/O reads/writes 64 KB;

so the maximum I/O rate of the bus is$= \frac{Bus\ b/w}{Bytes\ per\ I/0} = \frac{1000\times10^6}{64\times10^3} = 15{,}625$ I/Os per second

Numerical Problems

Question:

Previous system has a SCSI disk I/O controllers with a DMA transfer rate of 320 MB/s that can accommodate up to 7 disks per controller. The disk drives with a read/write bandwidth of 75 MB/s and an average seek plus rotational latency of 6 ms.

a) What is the maximum sustainable I/O rate ?

Numerical Problems

Question:

Previous system has a SCSI disk I/O controllers with a DMA transfer rate of 320 MB/s that can accommodate up to 7 disks per controller. The disk drives with a read/write bandwidth of 75 MB/s and an average seek plus rotational latency of 6 ms.

a) What is the maximum sustainable I/O rate ?

Answer:

Disk I/O read/write time = seek + rotational time + transfer time

= 6ms + 64KB/(75MB/s) = 6.9ms

The no. of I/Os that each disk can complete in one second $= \frac{1s}{6.9ms} = \frac{1000}{6.9} = 145$

Numerical Problems

Question:

Previous system has a SCSI disk I/O controllers with a DMA transfer rate of 320 MB/s that can accommodate up to 7 disks per controller. The disk drives with a read/write bandwidth of 75 MB/s and an average seek plus rotational latency of 6 ms.

b) How many disks are required to maximally utilize the processor in terms of I/O?

Numerical Problems

Question:

Previous system has a SCSI disk I/O controllers with a DMA transfer rate of 320 MB/s that can accommodate up to 7 disks per controller. The disk drives with a read/write bandwidth of 75 MB/s and an average seek plus rotational latency of 6 ms.

b) How many disks are required to maximally utilize the processor in terms of I/O?

Answer:

The processor supports 10^4 I/O s per second

The no. of I/Os that each disk can complete in one second is145

Hence, to maximally utilize the processor in terms of I/O, we need $\frac{10^4}{145} \approx 69$ disks

Numerical Problems

Question:

Previous system has a SCSI disk I/O controllers with a DMA transfer rate of 320 MB/s that can accommodate up to 7 disks per controller. The disk drives with a read/write bandwidth of 75 MB/s and an average seek plus rotational latency of 6 ms.

c) What is the number of SCSI controllers required to achieve that rate?

Answer:

Disk transfer rate = (transfer size)/(transfer time) = 64KB/6.9ms = 9.27 MB/s

Thus 7 disks won't saturate either the SCSI controller (with a maximum transfer rate of 320 MB/s) or the memory-I/O bus (1000 MB/s).

Thus the number of SCSI controllers we need is ceiling $\left(\frac{69\ disks}{7\ disks\ per\ controller}\right) = 10$.

Numerical Problems

Question:

A given processor requires 1000 cycles to perform a context switch and start an interrupt handler (and the same number of cycles to context-switch back to the program that was running when the interrupt occurred), or 500 cycles to poll an I/O device. An I/O device attached to that processor makes 150 requests per second, each of which take 10,000 cycles to resolve once the handler has been started. By default, the processor polls every 0.5 ms if it is not using interrupts.

a. How many cycles per second does the processor spend handling I/O from the device if interrupts are used?

Answer:

The device makes 150 requests, each of which require one interrupt.

No. cycles taken for interrupt handling

= start the handler + for the handler + context switch

= 1000 + 10,000+ 1000 = 12,000.

Therefore, no. of cycles = 150 x 12000 = 1,800,000 cycles

Numerical Problems

Question:

A given processor requires 1000 cycles to perform a context switch and start an interrupt handler (and the same number of cycles to context-switch back to the program that was running when the interrupt occurred), or 500 cycles to poll an I/O device. An I/O device attached to that processor makes 150 requests per second, each of which take 10,000 cycles to resolve once the handler has been started. By default, the processor polls every 0.5 ms if it is not using interrupts.

b. How many cycles per second are spent on I/O if polling is used (include all polling attempts)? Assume that the processor only polls during time slices when user programs are not running, so do not include any context-switch time in your calculation of polling costs.

Answer:

No. of polling attempt/second = 1000/0.5 = 2000 times/s.

Each polling attempt takes 500 cycles.

No. of cycles spend for polling per second = 500 x 2000 = 1,000,000 cycles

In 150 of the polling attempts were made. Each of which takes 10,000 cycles.

So time taken = 150 x 10,000 = 1,500,000 cycles.

Therefore the total time spent on I/O each second

= 1,000,000 + 1,500,000 = 2,500,000 cycles with polling.

Numerical Problems

Question:

An I/O device transfers 10MB/s of data into the memory of a processor over the I/O bus, which has a total bandwidth of 100MB/s. The 10MB/s of data is transferred as 2500 independent pages, each of which are 4KB in length. If the processor operates at 200 MHz, it takes 1000 cycles to initiate a DMA transaction and 1500 cycles to respond to the device's interrupt when the DMA transfer completes, what fraction of the CPU's time is spent handling the data transfer with and without DMA?

Answer:

Without DMA:

The processor must copy the data into the memory as the I/O device sends it over the bus.

Device data transfer rate = 10MB/s over the I/O bus

Bus bandwidth = 100MB/s

Thus CPU may transfer 100 MB data per second, but it needs to transfer only 10 MB per second

Hence, percentage of CPU cycle used= 10%

Numerical Problems

Question:

An I/O device transfers 10MB/s of data into the memory of a processor over the I/O bus, which has a total bandwidth of 100MB/s. The 10MB/s of data is transferred as 2500 independent pages, each of which are 4KB in length. If the processor operates at 200 MHz, it takes 1000 cycles to initiate a DMA transaction and 1500 cycles to respond to the device's interrupt when the DMA transfer completes, what fraction of the CPU's time is spent handling the data transfer with and without DMA?

Answer:

With DMA:

The processor is free to work on other tasks, except when initiating each DMA and responding to the interrupt at the end of each transfer.

Time taken for DMA transfer = Initiation time + Response time

= (1000+1500) = 2500 cycles/transfer

There are 2500 pages to transfer. So, total no. of cycles spent

= 2500 x 2500 = 6,250,000 (each second).

The processor operates at 200 MHz, means the number of cycles generated per second = 2,000,000,000.

Fraction of CPU cycle for DMA = 6,250,000/200,000,000 = 0.03125 = 3.125 %

Numerical Problems

Question:

A hard disk with one platter rotates at 15,000 rotations/min and has 1024 tracks, each with 2048 sectors. The disk head starts at track 0 (tracks are numbered from 0 to 1023). The disk then receives a request to access a random sector on a random track. The seek time of the disk head is 1ms for every 100 tracks it must cross.

a) What is the average seek time?

Answer:

Since the disk head starts at track 0, it will have to travel 0 tracks to handle a request to track 0, 1 track to handle a request to track 1, and so on, up to 1023 tracks to a request to track 1023.

On average, the head will have to travel half of the way to the outermost track, or 511.5 tracks. At 100 tracks/ms, this gives an average seek time of 5.115 ms.

Numerical Problems

Question:

A hard disk with one platter rotates at 15,000 rotations/min and has 1024 tracks, each with 2048 sectors. The disk head starts at track 0 (tracks are numbered from 0 to 1023). The disk then receives a request to access a random sector on a random track. The seek time of the disk head is 1ms for every 100 tracks it must cross.

b) What is the average rotational latency?

Answer:

At 15,000 r/min, each rotation takes (1 min / 15,000) = 4 ms.

The average rotational latency is half the rotation time, or 2 ms.