




Computer Organisation and Architecture

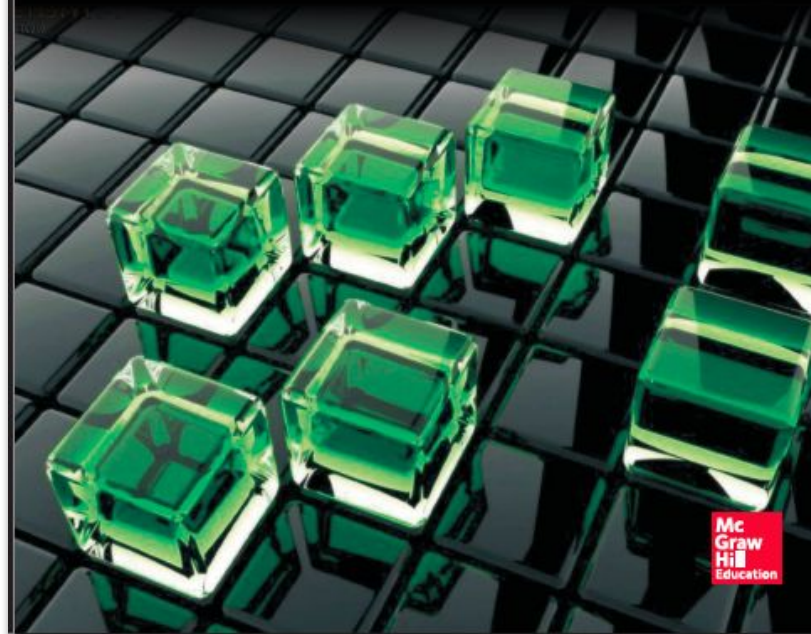
**Smruti Ranjan Sarangi,
IIT Delhi**

Chapter 12 I/O and Storage Devices

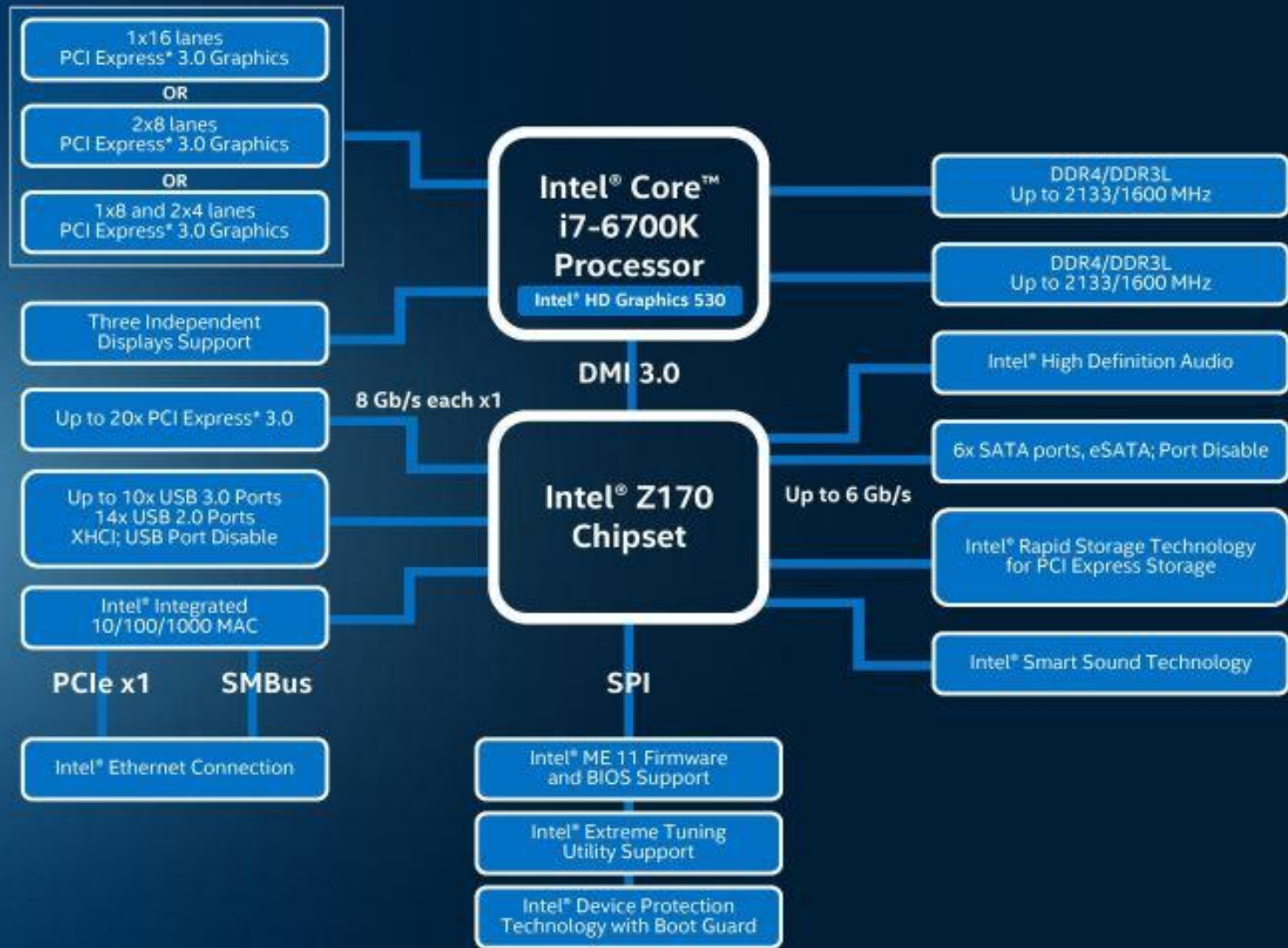
Also available as  Book

Computer Organisation and Architecture

Smruti Ranjan Sarangi



These slides are meant to be used along with the book: Computer Organisation and Architecture, Smruti Ranjan Sarangi, McGrawHill 2015
Visit: <http://www.cse.iitd.ernet.in/~srsarangi/archbooksoft.html>



I/O Devices

- * I/O Devices

- * Hard disk
- * Network card
- * Printer, scanner, camera, speakers

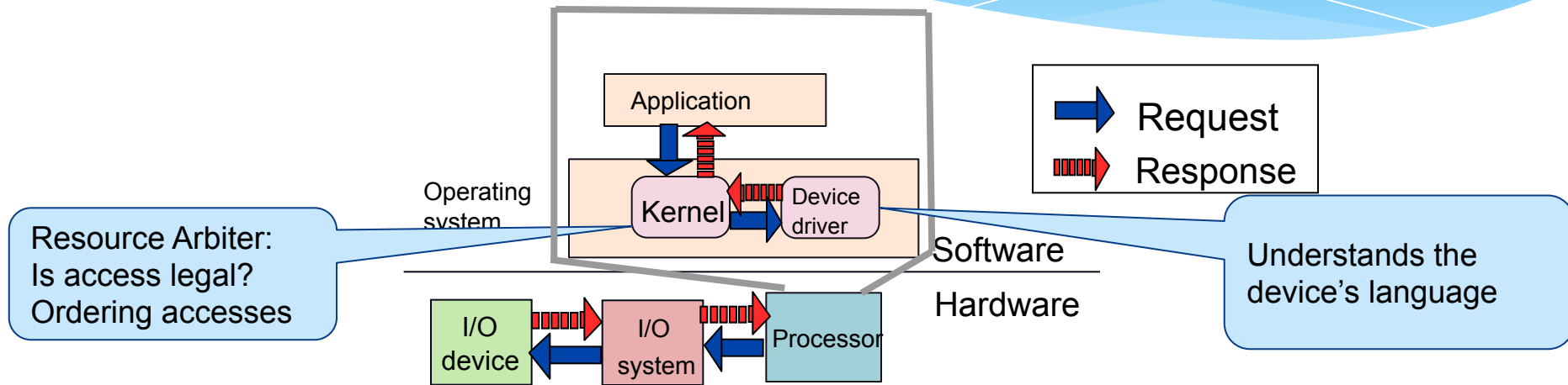
- * They are connected to the motherboard with I/O ports

- * A set of metallic pins for attaching with external connectors
- * Example : USB port, DVI Port.

Software's View of I/O Devices

- * **Linux** defines two **system calls**
 - * `read (int file_descriptor, void *buffer, int num_bytes)`
 - * `write (int file_descriptor, void *buffer, int num_bytes)`
- * All **devices** are perceived to be **files** in the `/dev` **file system**
 - * We can **read bytes** from them, or write bytes to them
 - * Two kinds of **devices** : **character** (keyboard, mouse), and **block** (hard disk, network card)

Operating System's I/O Stack

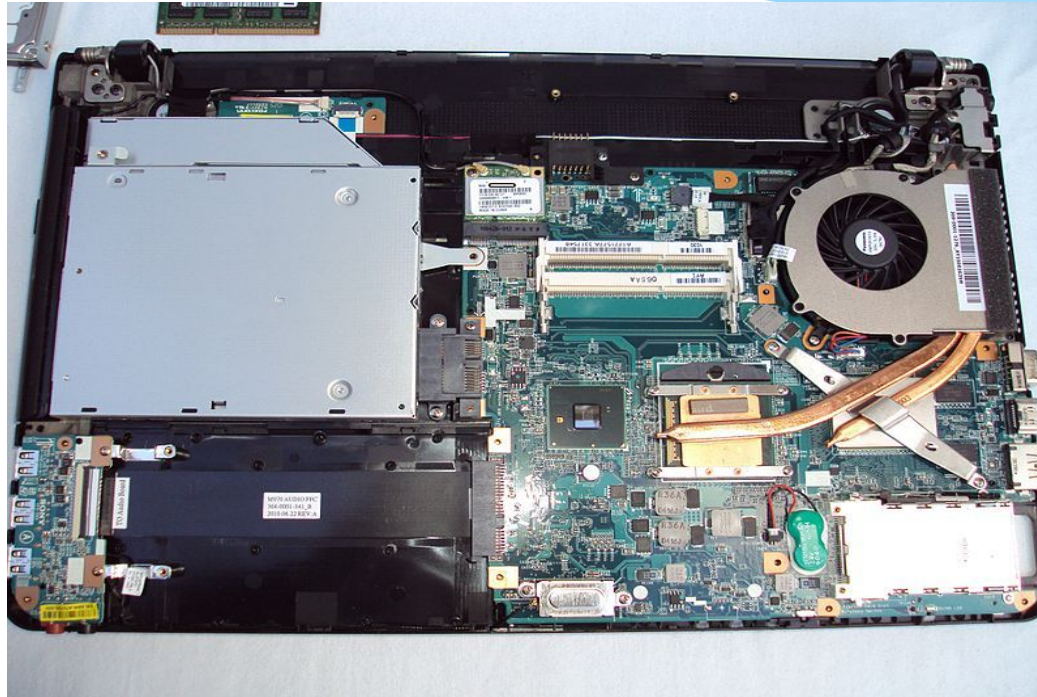


- * A request goes through the **kernel**, **device driver**, **processor**, and **I/O system**
- * **I/O devices** are connected to the **motherboard** via add-on **cards**, or directly

A Network Card

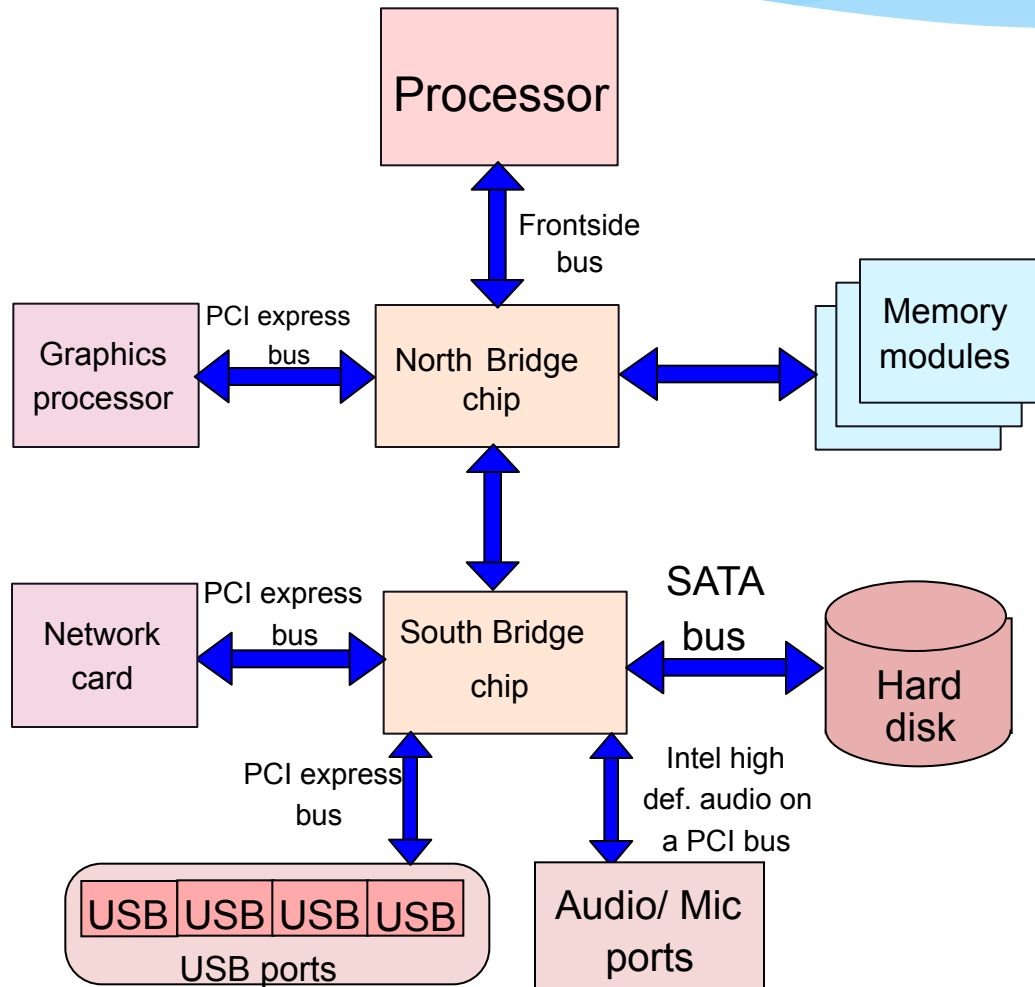


Chipset and Motherboard



- * A **motherboard** is a printed circuit board connecting the **processors** and auxiliary **chips**
- * The additional **chips** comprise the **chipset**

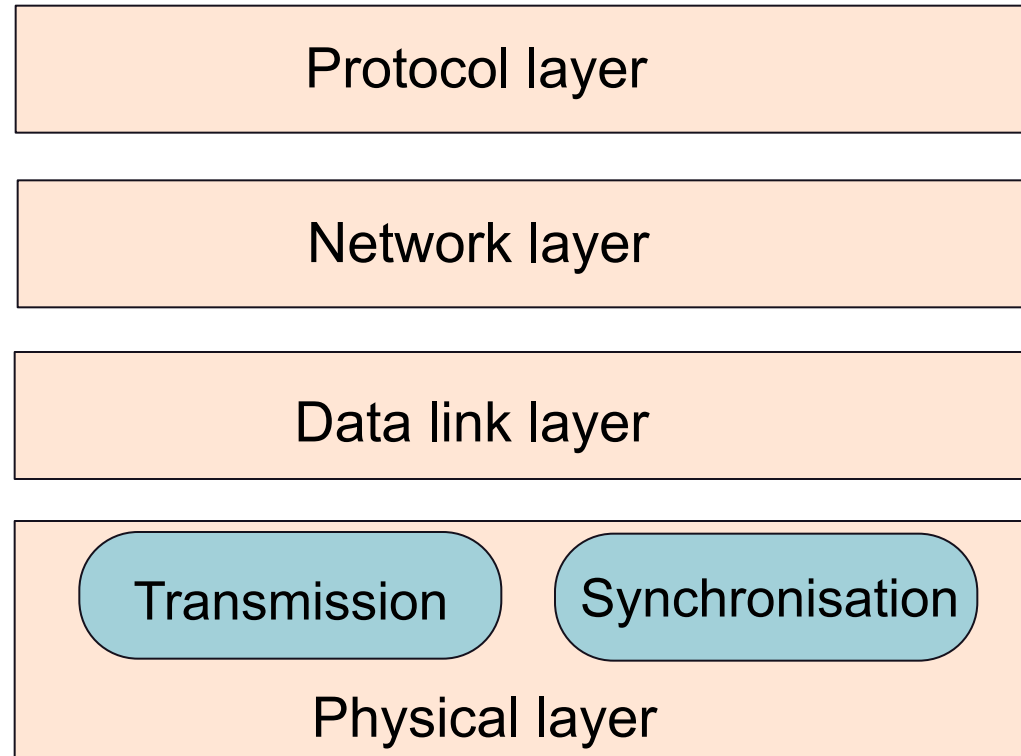
Architecture of the Motherboard



I/O Buses

- * The different components on the **motherboard** are connected with **I/O buses**
- * **I/O buses** are also used to connect external devices to the **motherboard**
- * An **I/O bus** is a set of **wires** that carries **data** and **control** signals between a set of **devices**. These devices use the **bus** to transmit data and **control signals** between each other.

Layers in the I/O System



Layers in the I/O System

- * Physical Layer

- * **Transmission Sublayer** → Defines the electrical specifications of the bus, and the methods for encoding data
 - * **Synchronisation Sublayer** → The timing of signals

- * Data link layer

- * Framing, buffering, error correction, transactions

Layers - II

- * Network Layer

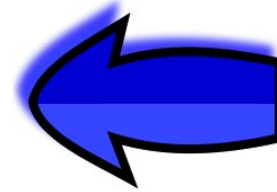
- * I/O device addressing, location, routing

- * Protocol Layer

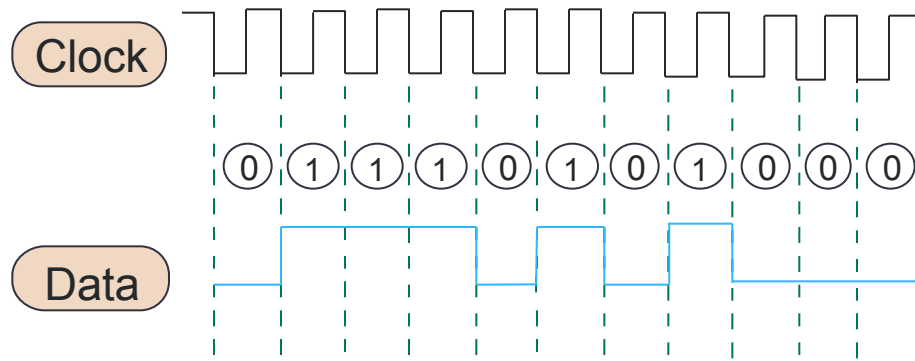
- * End to end request processing
 - * Interrupts, and polling
 - * DMA (Direct Memory Access)

Outline

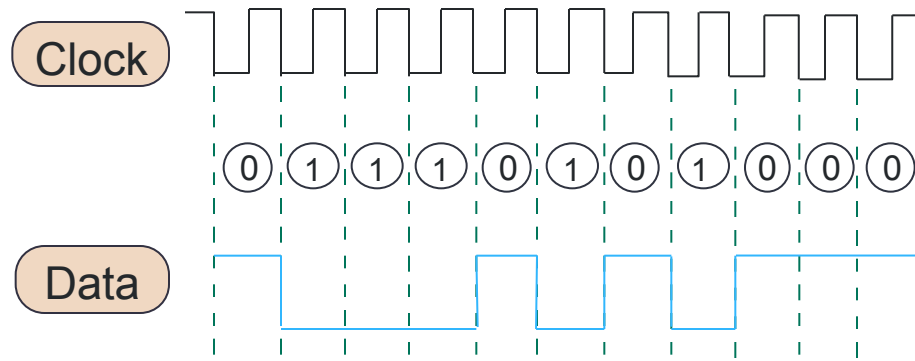
- * Overview
- * Physical Layer
- * Data Link Layer
- * Network Layer
- * Protocol Layer
- * Case Studies
- * Storage Media



Active High and Active Low



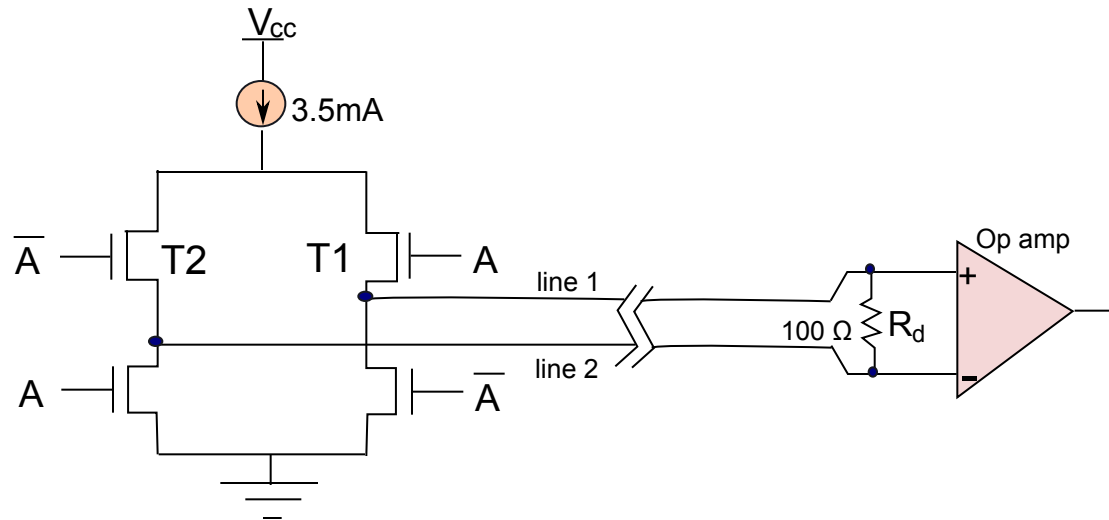
(a) Active high signalling



(b) Active low signalling

LVDS Signalling

Low Voltage Differential Signalling



- * If ($A = 1$) there is a **voltage difference** of 350 mV across the op Amp
- * Else there is a **voltage difference** of -350 mV
- * Smaller is the **voltage swing**, faster is the **circuit**

Terminology

- * **Physical Bit**

- * A value of 0 or 1 on the **bus**

- * **Logical Bit**

- * A function of a sequence of **physical** bits. Logical bits are passed to the next layer.

- * **Bit Period**

- * Time it takes to transmit a single bit

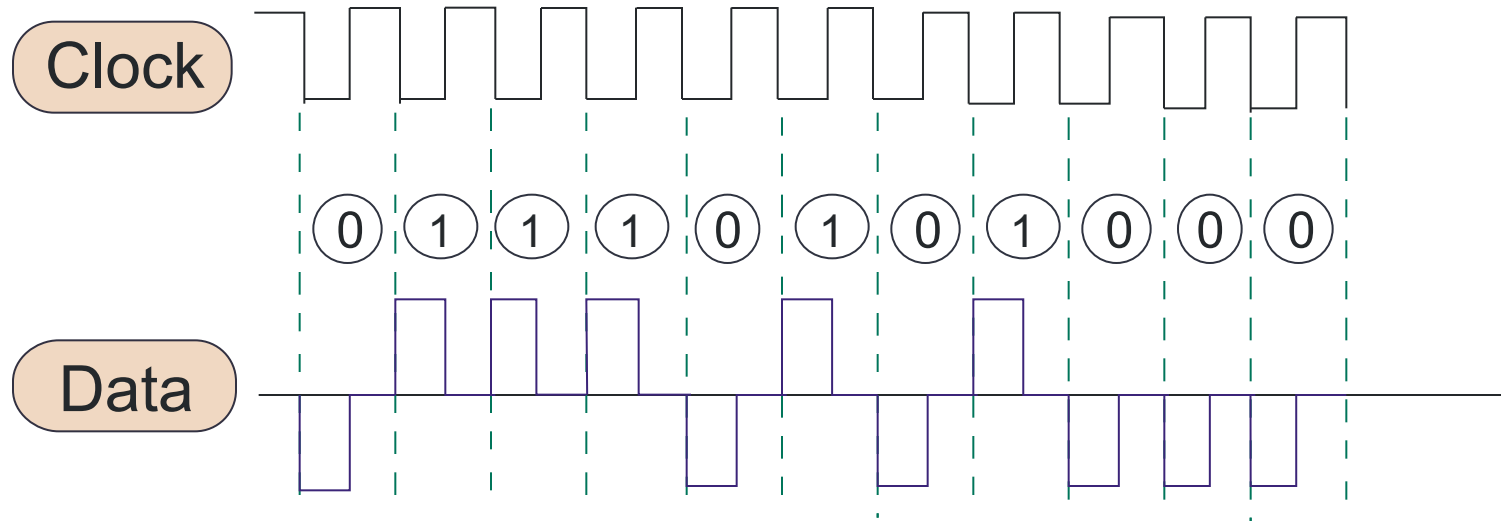
- * **I/O Clock Period**

- * One clock cycle of the I/O clock

Binary vs Ternary Signalling

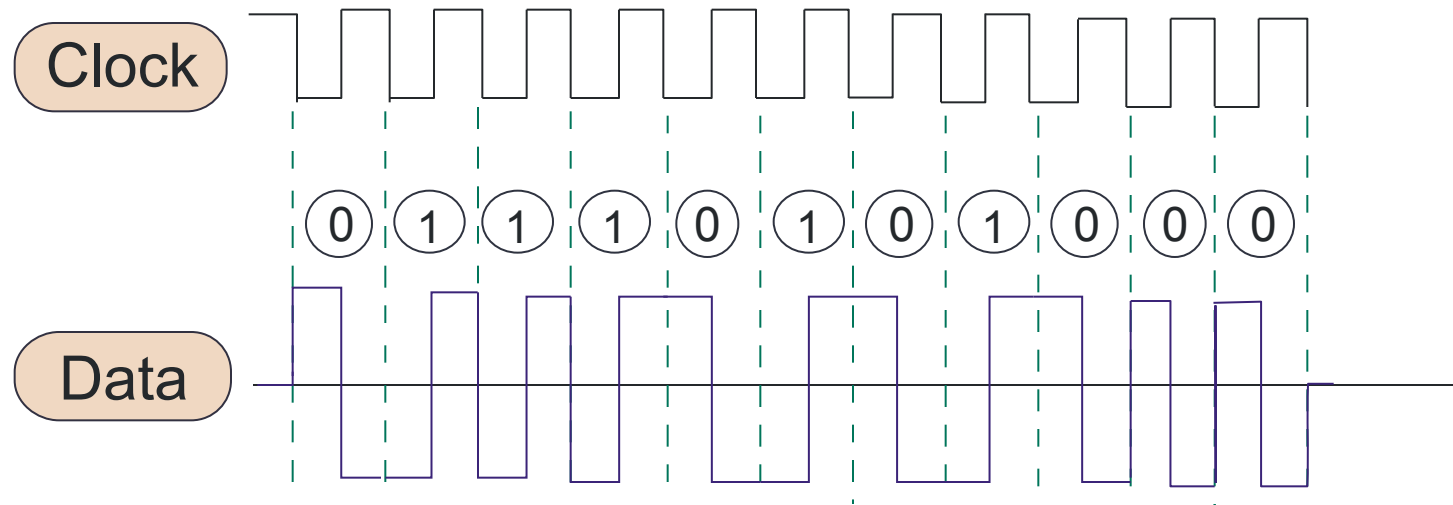
- * **Binary Signalling** → The physical bit can either take a value of 0 or 1
- * **Ternary Signalling** → We have three physical bits :
0, 1, and idle
 - * The **idle** state of the **bus** corresponds to the case where no **signal** is transmitted
 - * **LVDS** naturally supports **ternary signalling**. When the **voltage difference** is less than the threshold, the bus is in the **idle** state.

Return to Zero (RZ)



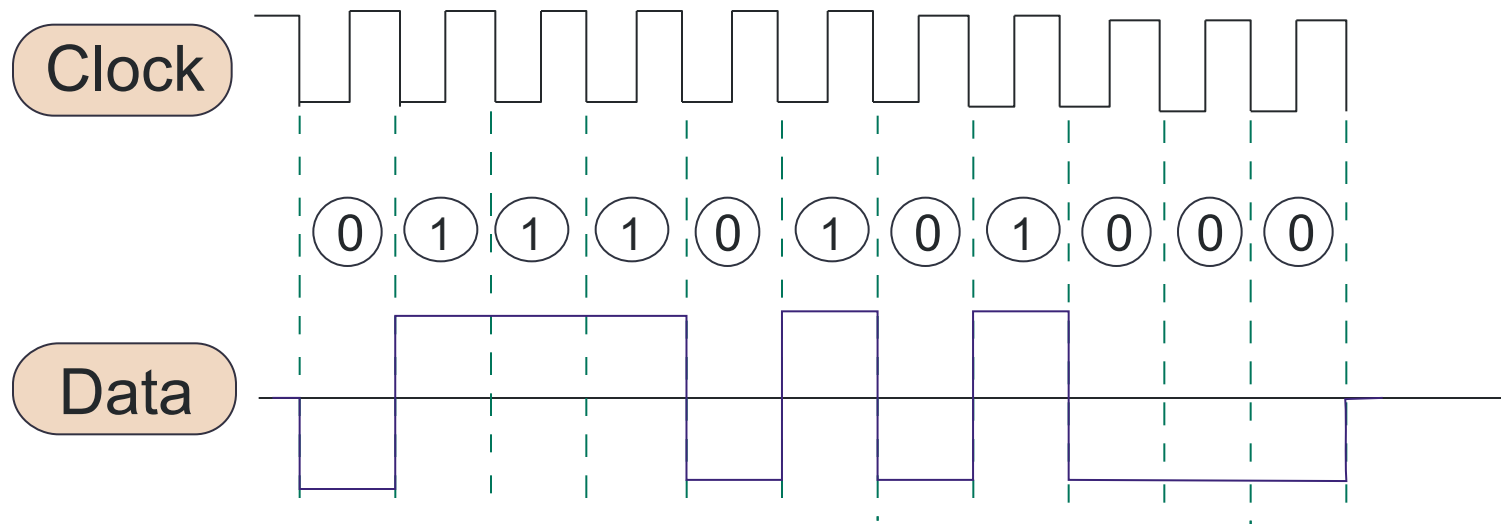
- * Logical 0 : 0 \rightarrow idle
- * Logical 1 : 1 \rightarrow idle

Manchester Encoding



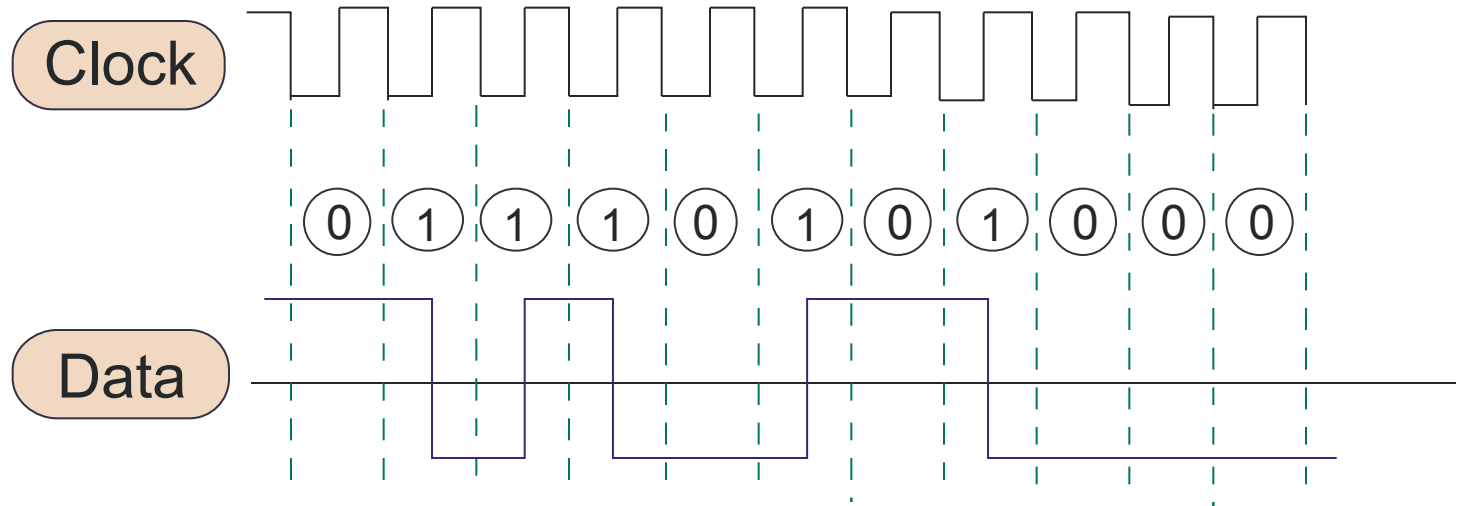
- * Logical 1 : 0 \rightarrow 1
- * Logical 0 : 1 \rightarrow 0

Non Return to Zero (NRZ)



- * Logical 0 : physical 0
- * Logical 1 : physical 1

Non Return to Zero Inverted (NRZI)

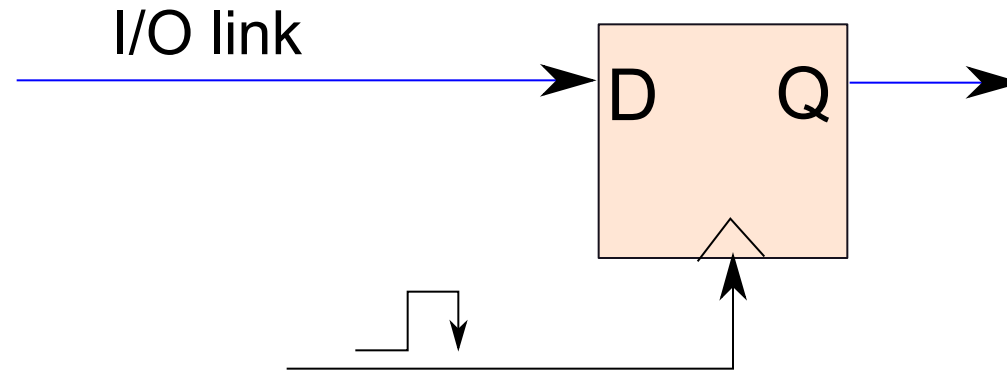


- * Logical 0 : no transition
- * Logical 1 : transition

Synchronisation Sublayer

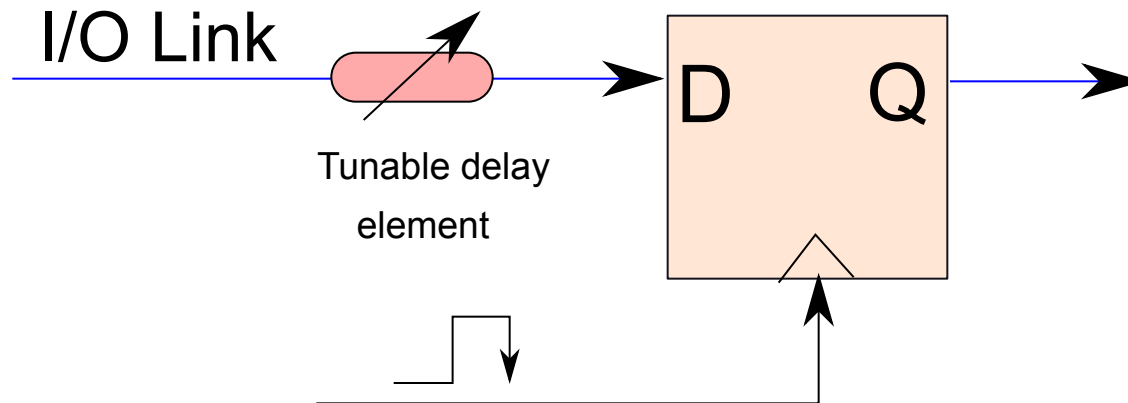
- * How does the receiver know, when to read the data ?
 - * It might not have the **same clock** as the sender
 - * The signal might have a **variable amount of delay**
 - * The receiver might **sample** the clock in the **keepout region**
 - * The signal cannot be **sampled** in a certain **interval** of time around the **clock edge** (recall setup time, and hold time)

Synchronous Buses



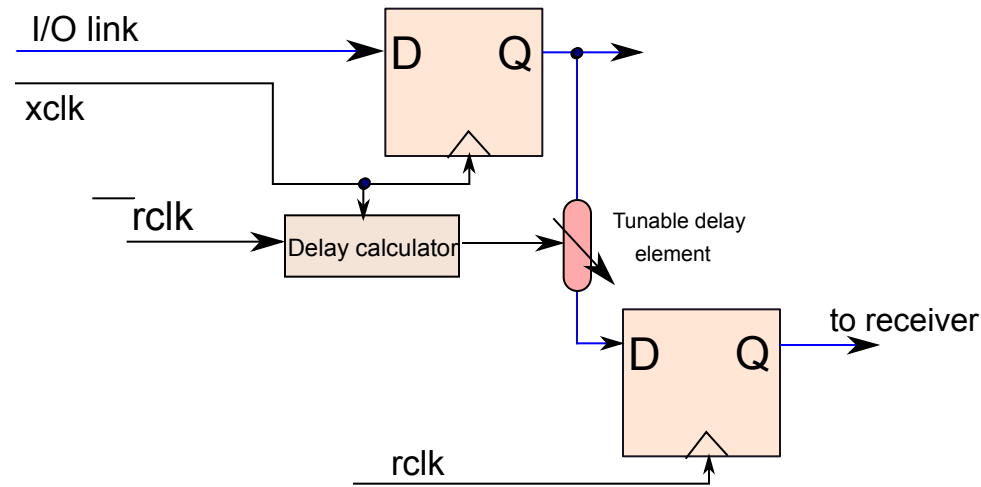
- * Sender and receiver have the same **clock**
- * The **signal** arrives at the **receiver** before its **clock** enters the **keepout region**
- * Figure shows the receiver circuit

Mesochronous Buses



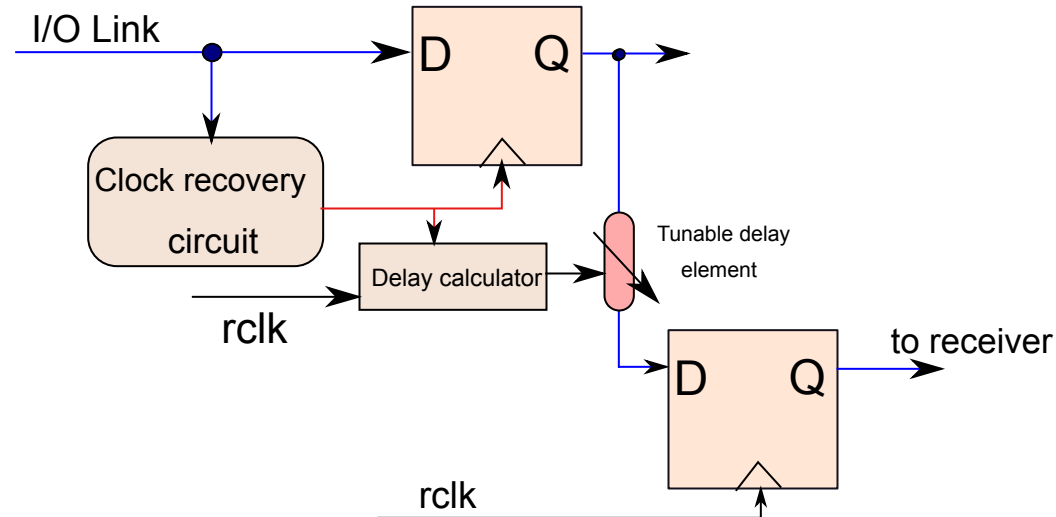
- * Fixed (known) **propagation delay**
- * Use a **delay element** to keep transitions out of the **keepout region**

Source Synchronous Bus



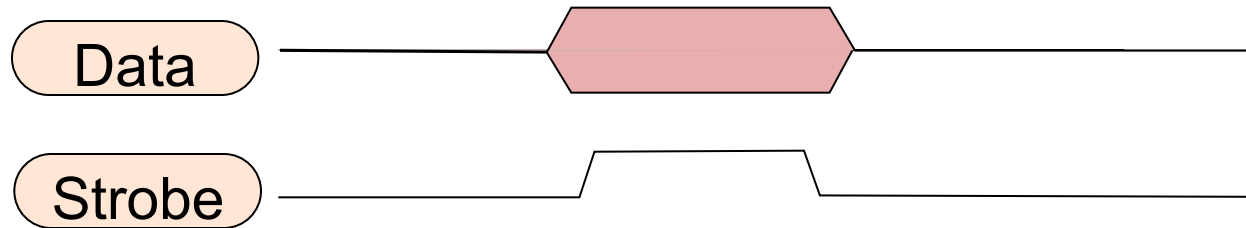
- * Sender sends **clock** along with the **signal**
- * Clock data in with the sender's clock (xclk)
- * Transfer the **data** to the receiver's clock domain using a **tunable delay element**

Asynchronous Bus



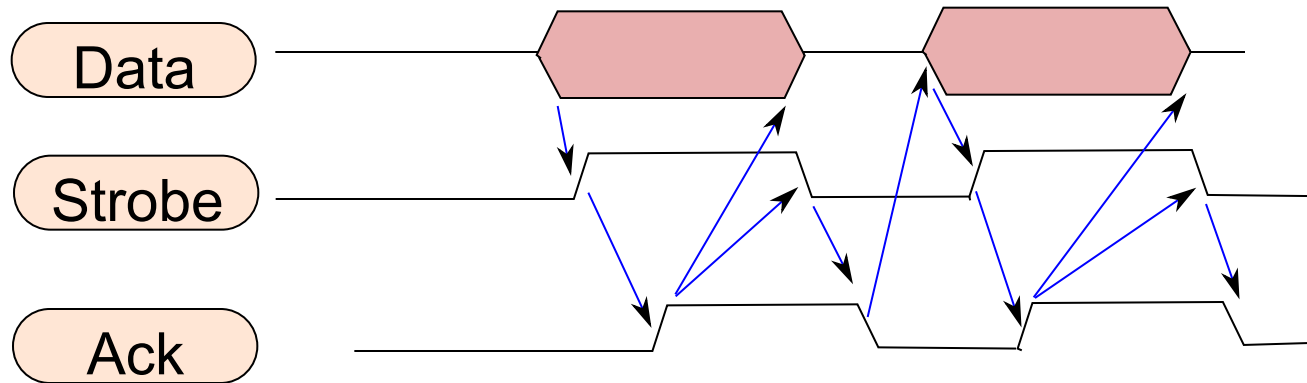
- * No guarantee of timing
- * Recover the **clock** from the **transitions** in the **data**
- * Transfer to the receiver's **clock domain** using a delay element

Communication with Strobe Signals



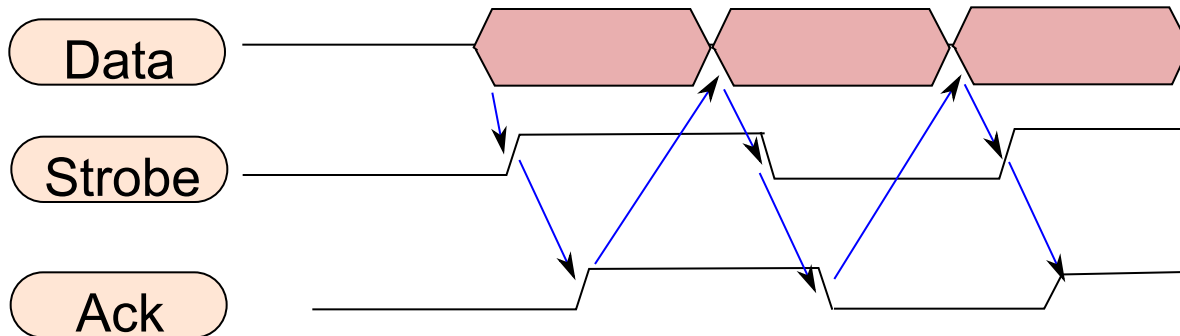
- * The **strobe signal** indicates the availability of data.
- * The **sender** has no way of knowing that the **receiver** has read the data

4 Phase Handshake



- * Receiver asserts the **ack signal** after it has read data.
- * The sender deasserts the **strobe**, and stops sending data
- * The receiver resets the **ack signal**

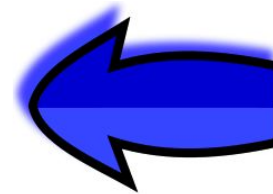
2 Phase Handshake



- * Instead of **asserting**, and **deasserting** signals, we toggle their values
- * No need to **pause** between **transmitting** bits.
- * The **sender** starts sending the next bit after it sees the **ack** line to be **toggled**

Outline

- * Overview
- * Physical Layer
- * Data Link Layer
- * Network Layer
- * Protocol Layer
- * Case Studies
- * Storage Media



Framing

- * Create a **frame** of data from logical bits
- * A **frame** is an atomic unit of data (**data packet**)
- * How do we detect frames ?
 - * **Demarcation** by inserting **pauses** → wastes bandwidth
 - * **Bit count** → Count the number of bits. **What if we miss a bit ?**

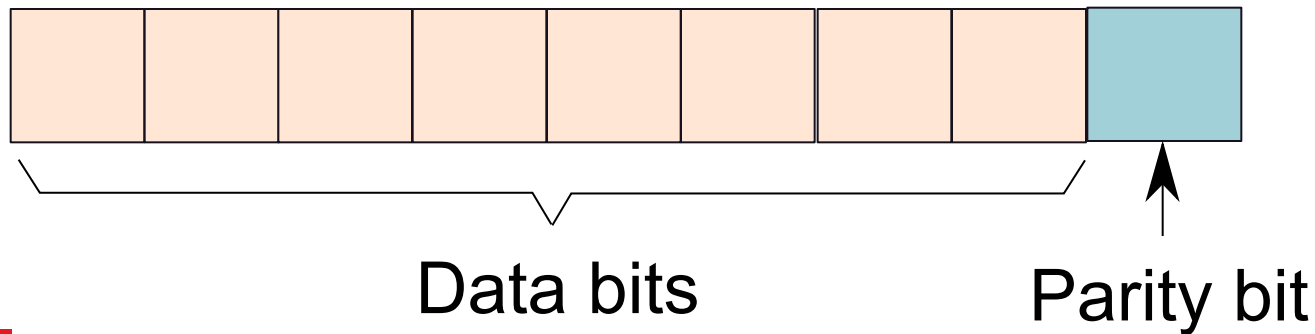
* Frame Detection

- * Bit/Byte Stuffing → Insert special symbols at the beginning and end of frames.
- * For example, insert the sequence : 0xDEADBEEE, at the beginning. If DEADBEEE occurs inside the message, repeat the symbol two times. (Similar to '\ ' in C++/ Java)

Error Detection/ Correction

- * Single **error detection**. Have a **parity** bit.

$$P = D_1 \oplus D_2 \oplus \dots \oplus D_8$$



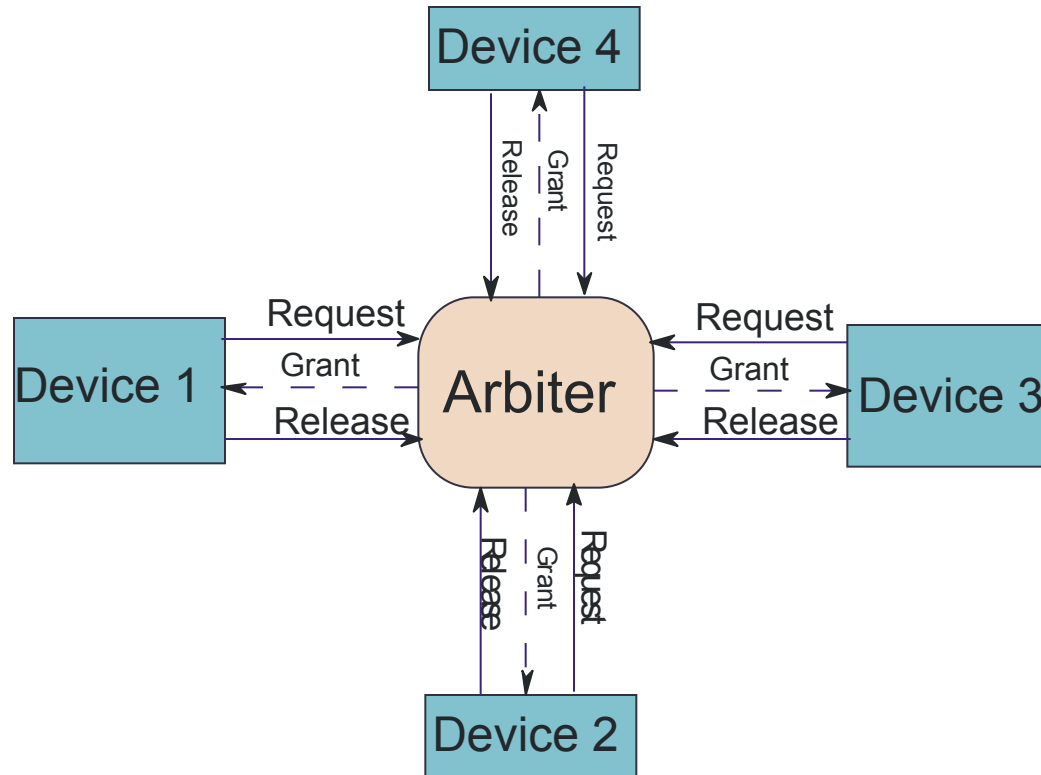
Other Error Detection/Correction Schemes

- * **Single Bit Error Correction** → Achieved by having **multiple parity bits** (taught in classes on **data communication**, and **coding theory**)
- * **SECDED** (Single Error Correct, Double Error Detect) → Taught in courses on coding theory
- * **CRC** (Cyclic redundancy check) → Used to check a burst of errors.

Arbitration

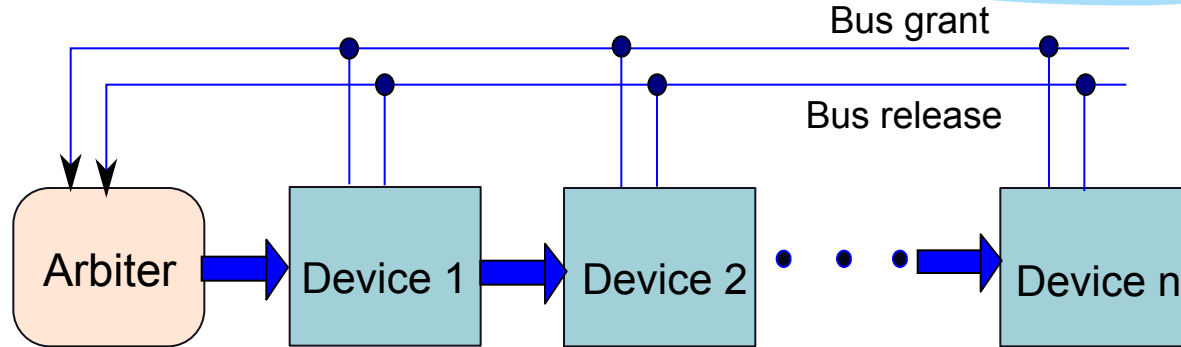
- * In a **multidrop bus**, multiple **transmitters** are connected to a single bus
- * How do they get access to the bus ? Use a method called **arbitration**
- * Two methods for **arbitration** → **centralised**, and **daisy chain**

Centralised Arbitration



* request → grant → release

Daisy Chain Arbitration

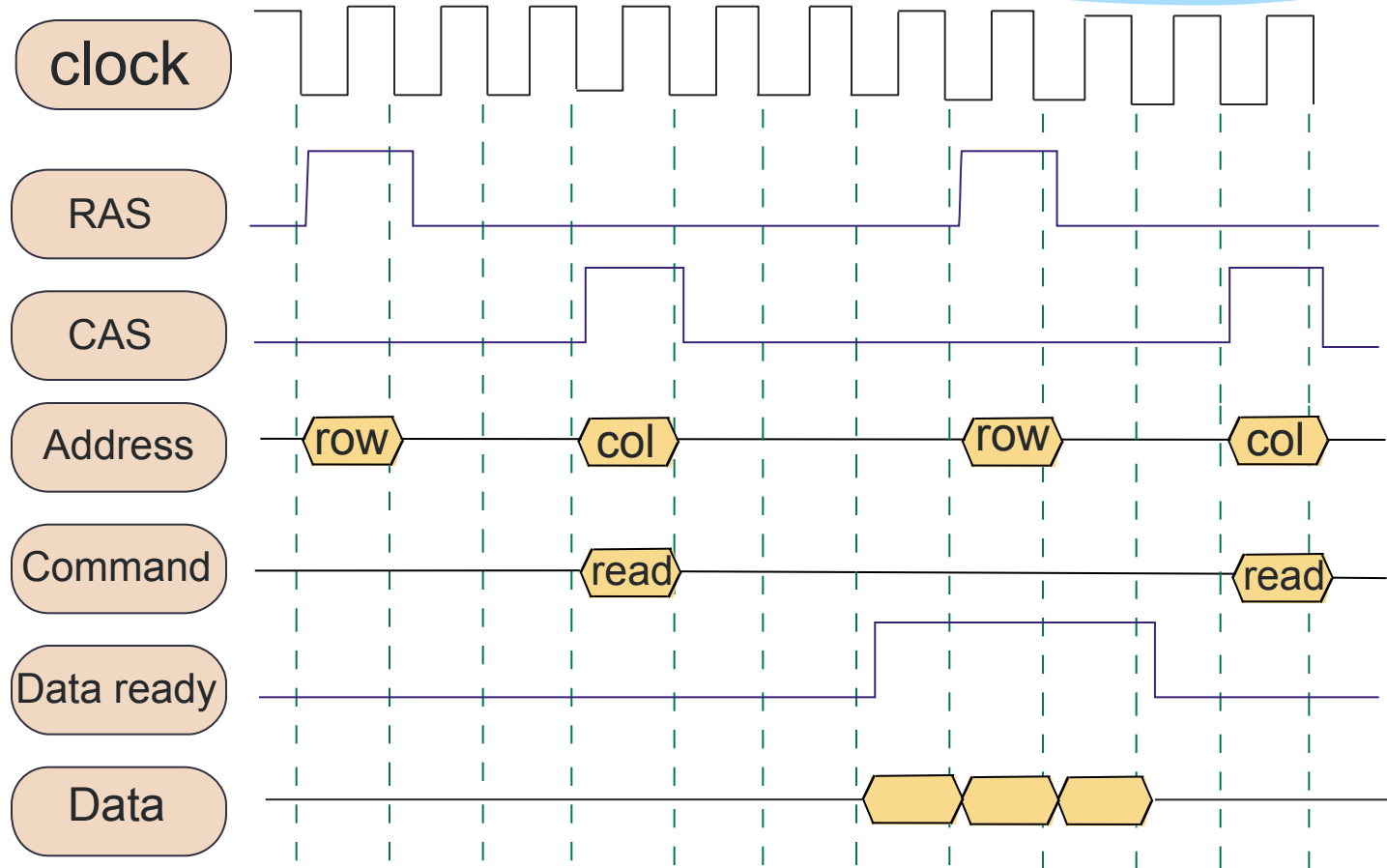


- * Device interested to transmit asserts bus grant
- * The arbiter injects a token.
- * The token passes from device to device.
- * The last device sets the release signal, and destroys the token.

Transaction Oriented Bus

- * A **request** from the **sender** to the **receiver** has a given **semantics**. It consists of an **atomic sequence** of **messages**.
- * This is called a **transaction**.
- * **Bus** that allows both sides to **communicate** at the same **time** → **full duplex**
- * Only one side at a time → **half duplex**

DRAM Read Transaction

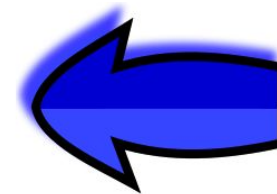


Split Transaction Buses

- * A **transaction** unnecessarily ties up the **bus**
- * A transaction might also have long **pauses** in the **middle**.
- * Use a split transaction bus. Typically consists of a **request** sub-**transaction**, and **response** sub-**transaction**.
- * Other messages can be sent between request and response.
- * Flexible, increases bandwidth, simpler

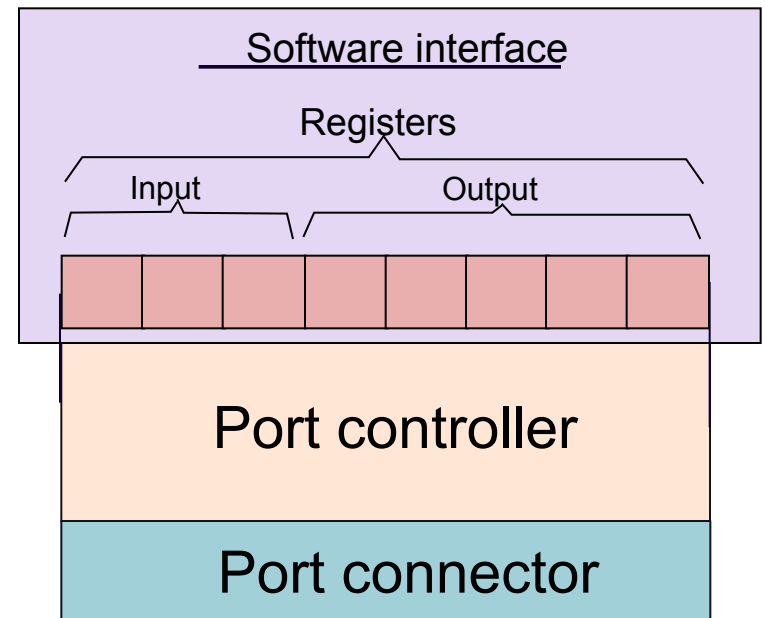
Outline

- * Overview
- * Physical Layer
- * Data Link Layer
- * Network Layer
- * Protocol Layer
- * Case Studies
- * Storage Media



I/O Port Addressing

- * Every device on the **motherboard** exposes a set of **I/O ports**.
- * An **I/O port** in this case, is a **software** entity.
- * Each **software I/O port** is a wrapper on the **actual hardware I/O port**



Software Interface

- * Each I/O port exposes a set of 8-32 bit registers to software
- * Software writes to the registers. The port controller automatically sends the information to the I/O device.
- * Similarly, to read data the processor reads the registers of the port controller.
- * Example : Intel processors define 64K, 8 bit I/O ports. Each port has a 16 bit port number.

I/O Address Space

- * Set of all **I/O ports** that are accessible to **software**
- * x86 **processors** have two instructions to access I/O ports → **in** and **out**

Instruction	Semantics
In r1,<io port>	$r1 \leftarrow \text{Contents of } \langle \text{i/o port} \rangle$
Out r1, <io port>	$\text{Contents of } \langle \text{i/o port} \rangle \leftarrow r1$

I/O Mapped I/O

- * A **request** contains an **I/O port address**
- * The **processor** sends it to the **Northbridge chip**.
- * The **Northbridge chip** forwards it to the **Southbridge chip** (if necessary)
- * The **Southbridge chip** forwards it to the destination. (there might be several more hops)
- * Each **chip** maintains a small routing table.
- * The **response** follows the reverse **path**.

Memory Mapped I/O

- * **Problems** with I/O mapped I/O
 - * The **programmer** needs to be aware of the addresses of the I/O ports.
 - * The same **device** might have different port addresses across different motherboards
 - * **Programs** will cease to work.
 - * It is hard to **transfer** a block of data (need to send it instruction by instruction)
 - * **Solution** : **memory mapped I/O**

Memory Mapped I/O

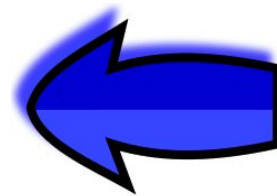
- * Define a **virtual layer** between I/O ports, and the **application**.
- * The **operating system** can use the paging mechanism to map I/O ports to **memory addresses**.
- * Whenever, we **write** to a **memory address** that is mapped to an I/O port, the **TLB** directs the request to the **I/O system**.
- * Similarly, for reading data, the response comes from the **I/O System**

Advantages

- * The **operating system** creates the **mapping** between the I/O ports and the memory addresses
- * The same **I/O program** runs on multiple machines
- * Reading and writing to **I/O devices** requires normal load/store instructions
- * Easy to **read** and **write** a large block of data using **block** load/store operations available in some **architectures** (e.g, x86)

Outline

- * Overview
- * Physical Layer
- * Data Link Layer
- * Network Layer
- * Protocol Layer
- * Case Studies
- * Storage Media



Protocol Layer

- * Defines the **interaction** between the **host** (processor) and **I/O devices**
- * The **protocol** layer consists of high level commands that are sent over the **three lower layers**
- * Three methods :
 - * (1) Polling, (2) Interrupts, (3) DMA (Direct Memory Access)

Polling

* Example :

- * Assume that the **application** running on the **processor** wants to print a **page**.
- * It needs to first find if the **printer** is free before sending it the contents of the **page**.
- * It keeps querying the **printer** for its status. If its **status** is **busy**, the program waits for some time, and queries again.
- * This method is known as **polling**.
- * Simple, yet **inefficient** (traffic, power, computational time)

Interrupts

- * The **host** tells the **device** to notify it if there is a change in its **status**
- * In this case, if the **printer** is busy, then the host lets the **printer** know that it is interested in printing one more **page**.
- * The **printer** sends it an **interrupt**, once it is **free**.
- * The **host** processes the **interrupt**, and the **application** subsequently sends the **print** job

DMA (Direct Memory Access)

- * Now, let us assume that the **application** is aware that the **printer** is free.
- * It needs to **transfer** several MB of data to the **printer** for **printing**.
 - * If it **transfers** data byte by byte, the **processor** will be tied up for the entire duration
 - * Even if it uses **memory mapped I/O**, the entire **operation** will require a large amount of **CPU** time.
 - * **Best solution** : **outsourcing**

DMA Engine

- * Assign the work of transferring data between **main memory** and the **I/O devices** to the **DMA engine**
 - * The **DMA** (direct memory access) unit is typically a part of the **Northbridge chip**
 - * It has access to **main memory**, and to **I/O devices**. It can seamlessly **transfer** data between them.
 - * Once it is done, it sends an **interrupt** to the **processor**.
 - * The **processor** programs the **DMA** engine with the addresses in memory, size of data, and **I/O address** locations

DMA Modes

- * **Burst mode**

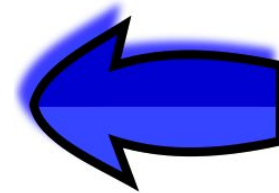
- * Locks the **FSB** and the buses to the **I/O device** till the entire transaction is over.

- * **Cycle stealing mode**

- * **Split-transactions**
 - * **Transfers** data in smaller chunks. DMA traffic typically has a much lower priority than regular traffic.

Outline

- * Overview
- * Physical Layer
- * Data Link Layer
- * Network Layer
- * Protocol Layer
- * Case Studies
- * Storage Media



PCI Express

- * **Motherboards** needs a **bus** to connect the I/O elements
 - * There were many buses in use in the late nineties. Two of them were very popular.
 - * **PCI** (Peripheral Component Interconnect)
 - * **AGP** (Accelerated Graphics Port)
- * A **standardisation** effort led to the **PCI-Express** (PCI-X) bus
- * A **PCI Lane** is a :
 - * **High speed** serial bus.
 - * Does not use **parallel links** because of the possibility of different amounts of **delay** across the links.
Synchronisation across links is **difficult**.

PCI Express (Peripheral Component Interconnect Express)	
Usage	As a mother board bus
Specification	PCI Express specifications (link)
Topology	
Connection	Point to point with multiple lanes
Lane	A single bit full duplex channel with data striping
Number of Lanes	1 to 32
Physical Layer	
Signalling	LVDS based differential signalling
Encoding	8 bit/10 bit
Timing	Source synchronous
Data Link Layer	
Frame Size	1byte
Error Correction	32 bit CRC
Transactions	Split transaction bus
Bandwidth	250 MB/s per lane
Network Layer	
Routing Nodes	Switches

USB

- * **USB** (Universal Serial Bus)

- * De facto **standard** for connecting all types of **peripherals** to a computer (as of today)
- * Scanners, printers, cell phones, pen drives, ...

- * This is also a serial bus to allow high speed signalling.

- * Each **USB port** (host) can be connected to a set of **devices** arranged as a tree.

- * Each **internal node** is known as a **hub**
- * We can **connect** a total of 127 **devices** (including hubs). **max depth** = 5

USB Physical Layer

- * A USB connector has 4 pins
 - * $V_{cc} \rightarrow 5V$ DC
 - * D^+ and D^- : differential pair (3.3V)
 - * **Gnd pin**
 - * Micro and Mini USB ports have an additional pin \rightarrow ID (helps differentiate between device and host)
- * Uses NRZI signalling with dummy bits (for clock recovery)

USB Data Link Layer

- * Four kinds of packets
 - * **Control** → control messages to configure devices
 - * **Interrupt** → interrupts
 - * **Bulk** → Large amount of data transfer (printing a page)
 - * **Isochronous** → Data transfer at a fixed rate (web camera)
- * Implements a **split transaction** bus

Network Layer

- * Each **USB device** is assigned a **7 bit** id by the **host**
 - * Every **USB device** defines a set of **I/O ports**, limited to
 - * 16 **IN** ports
 - * 16 **OUT** ports
 - * Thus each **port** has a unique 11 bit **address**
 - * Software **drivers** talk to the **device** by sending messages to the corresponding **port**.
- * Protocol Layer
 - * Stream pipe (unstructured), message pipe (structured with a handshaking mechanism)

USB Summary

USB (Universal Serial Bus)

Usage	Connecting peripheral devices → keyboards, mice, web cameras, pen drives
-------	--

Topology

Connection	Point-to-point, Serial
Width	Single bit, half-duplex

Physical Layer

Signalling	LVDS based differential signalling
Encoding	NRZI (transition represents a logical 0)
Timing	Asynchronous (a 0 added after 6 continuous 1s for clock recovery)

USB Summary - II

Data Link Layer

Frame Size	46 – 1058 bits
Error Correction	CRC
Transactions	Split Transaction Bus
Bandwidth	192 KB/s (low speed) 1.5 MB/s (full speed) 60 MB/s (high speed)

Network Layer

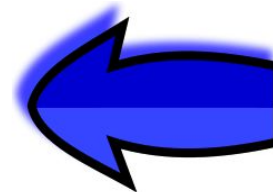
Address	7 bit device id, 4 bit end point id
Routing	Using a tree of hubs
Hub	1 upstream port, up to 4 downstream ports
USB Network	Can support a maximum of 127 devices

Protocol Layer

Connections	Message Pipe (structured), and stream pipe (unstructured)
Type of Traffic	Control, Interrupt, Bulk, Isochronous

Outline

- * Overview
- * Physical Layer
- * Data Link Layer
- * Network Layer
- * Protocol Layer
- * Case Studies
- * Storage Media

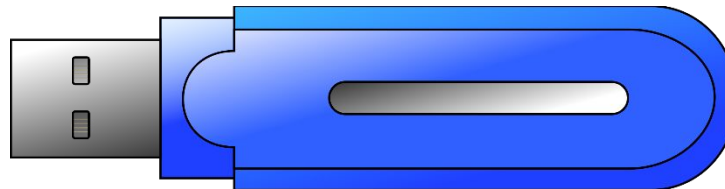




Hard Disks



Optical Drives



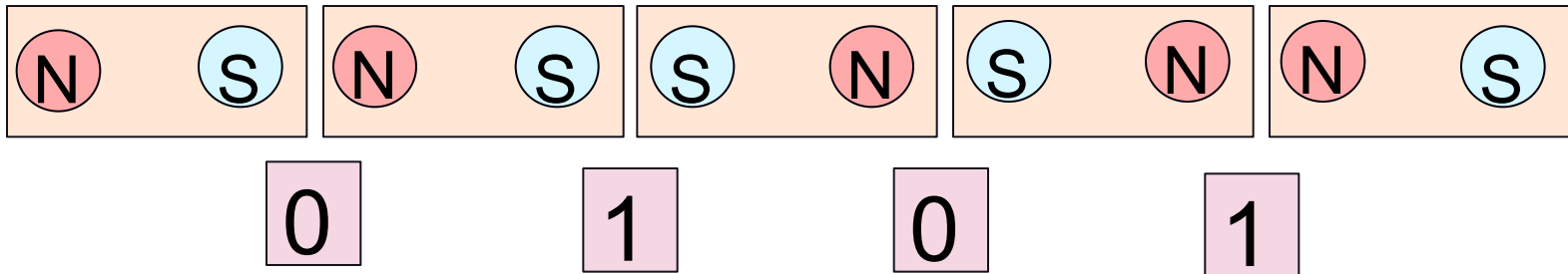
Solid State Drives



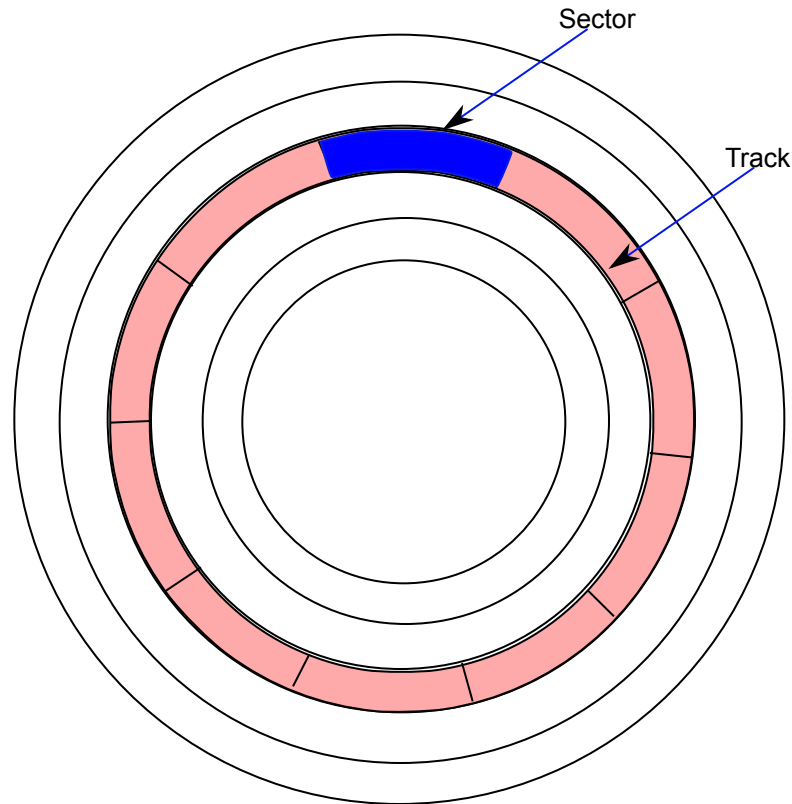
Hard Disks

Data Storage in Hard Disks

- * **Magnetic** storage (sequence of tiny magnets)
- * NRZI Encoding
 - * with **dummy** bits

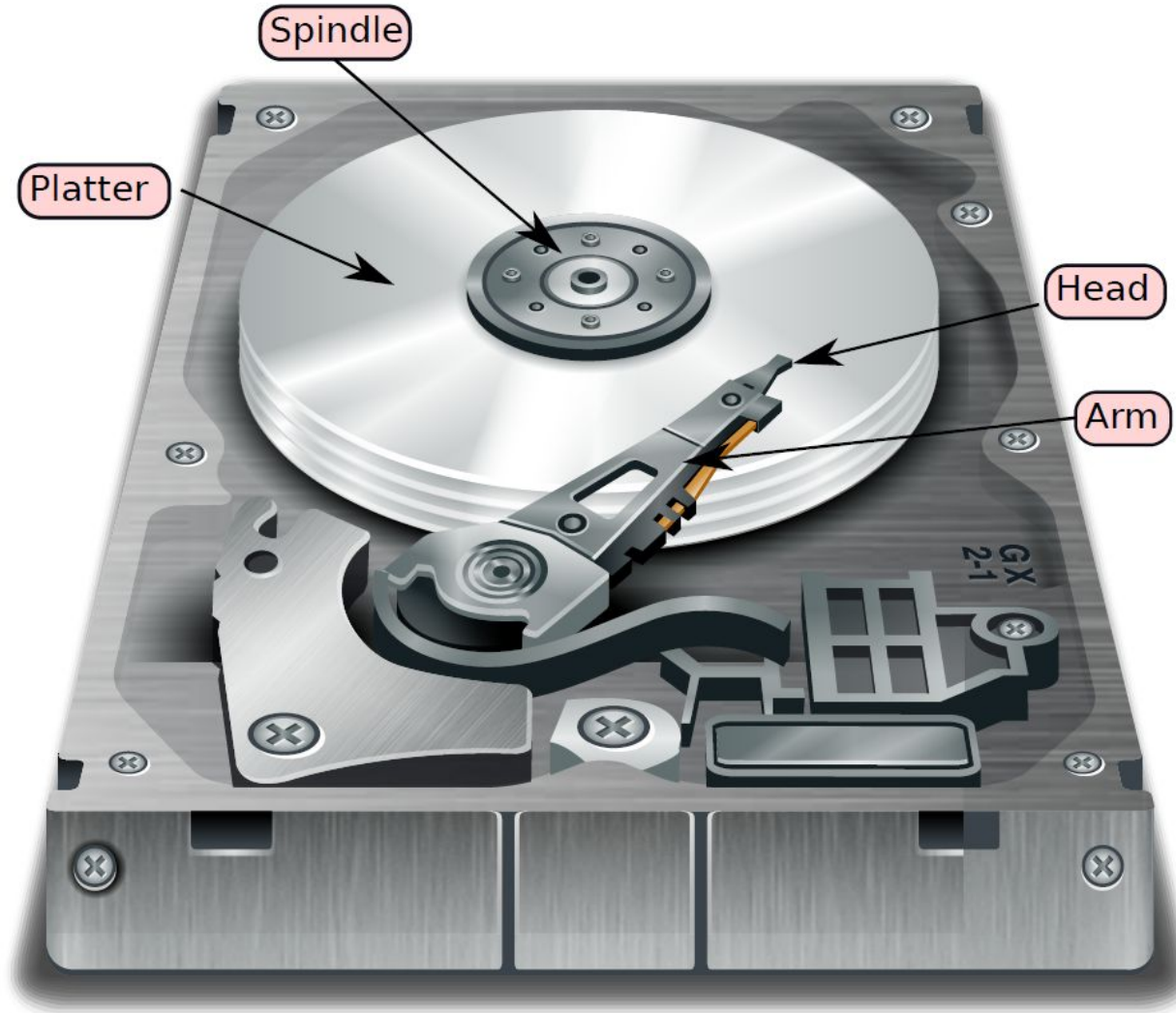


Structure of a Platter

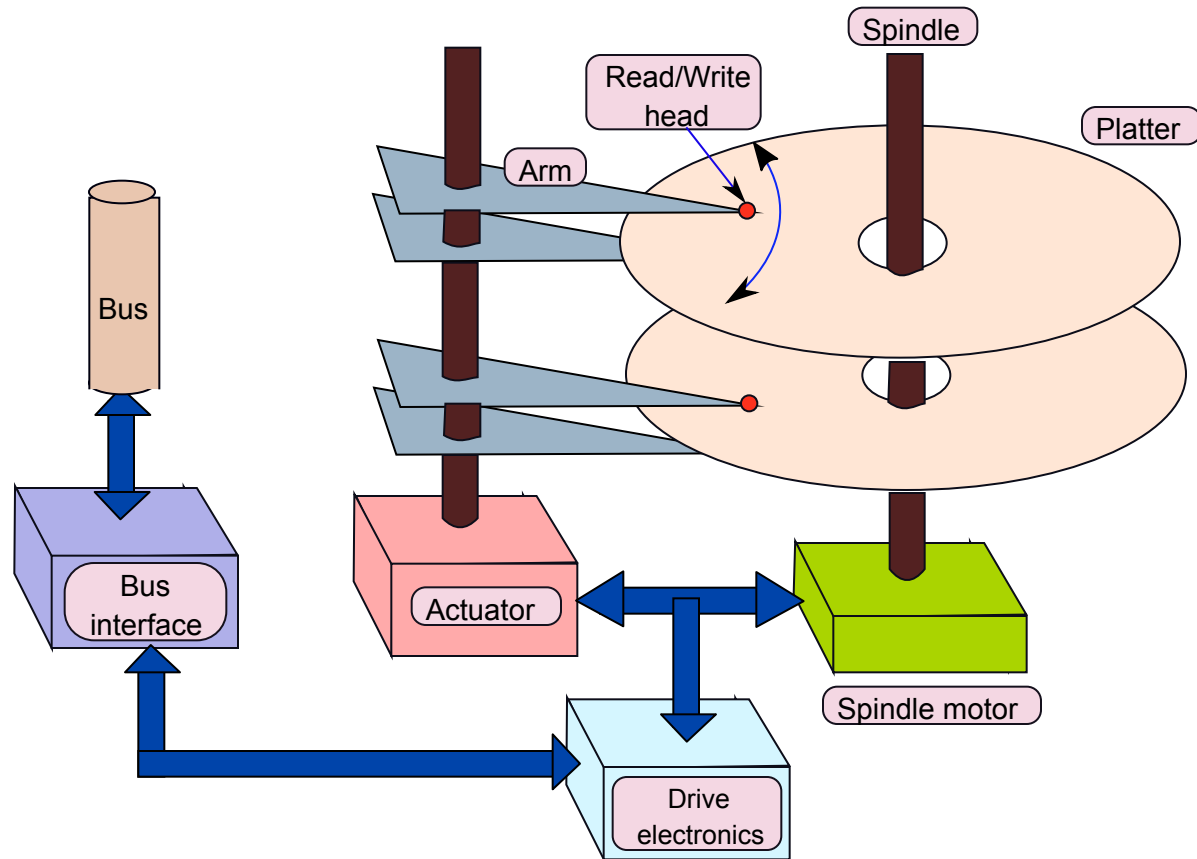


A platter is divided into concentric rings (**track**)
Each track is divided into fixed size **sectors**

Structure of a Hard Disk



Structure in Detail



Accessing a Given Sector

- * Position the **head** on the correct track
 - * seek time
- * Wait for the desired **sector** to come under the **head**
 - * Rotational Latency
 - * Assume that the **hard disk** rotates at constant angular velocity
- * **Read** or **Write** data
 - * Perform error checking, correction, framing
 - * Transfer data to the CPU → (Transfer Time)
 - * The transfer starts when the **head** is on the first bit of the **sector**

$$T_{disk\ access} = T_{seek} + T_{rot.latency} + T_{transfer}$$

Logical vs Physical Block Address

- * Software programs use the **logical block** address to address a **hard disk** block (**sector**)
 - * The **hard disk** internally converts the **logical** address to a **physical** address (recording surface, track, sector)
 - * It dedicates a **recording surface** for storing this information.
 - * It has a small **cache** (DRAM) to store the most recently used **mappings**
 - * The **hard disk** can also use this mechanism to mark **bad sectors** and remap **logical sectors** to healthy **physical sectors**.

RAID – (Redundant Array of Disks)

- * Hard Disks tend to have **high** failure rates
 - * Too **many** mechanical components
 - * High temperature **sensitivity**
- * What do we do?
 - * Do we **lose** all our data
 - * NO
- * Use RAID
 - * **Redundant** disks
 - * To **tolerate** faults, and **recover** from failures

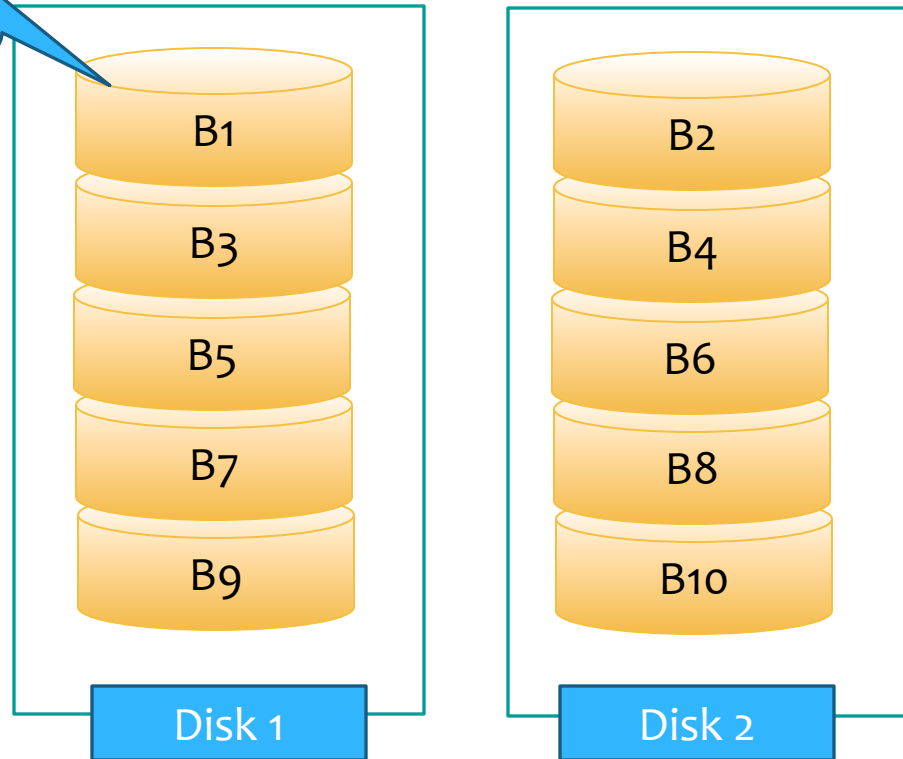
RAID – II

- * What about **bandwidth**?
 - * Assume we want to read two **different** blocks from the same disk?
 - * We need to read them **serially**.
 - * **Low** Bandwidth
- * How can we **increase** bandwidth
 - * Read from multiple disks in **parallel**.
 - * **Solution**: RAID

RAID 0

Block 1
(512 bytes)

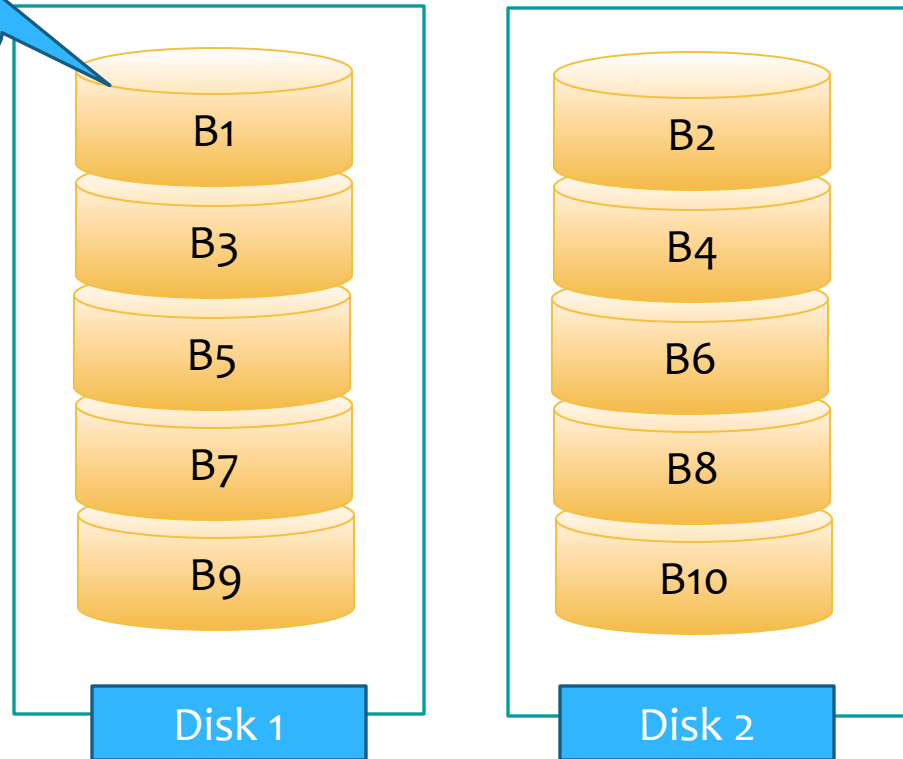
Distributing data
across disks defined
as *data striping*



RAID 0

Block 1
(512 bytes)

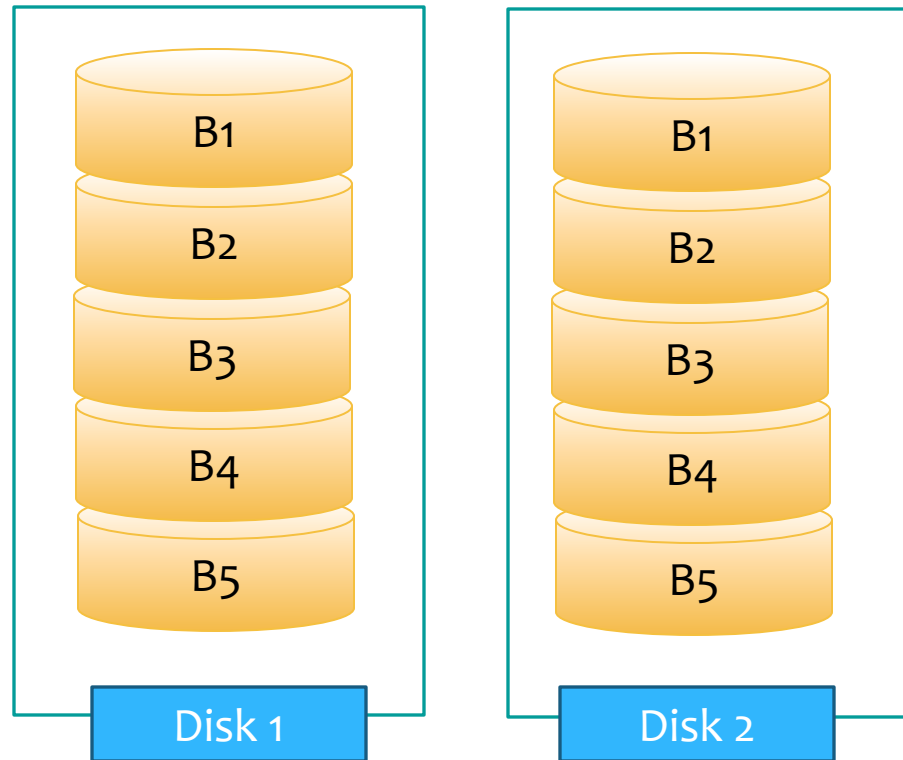
Distributing data
across disks defined
as *data striping*



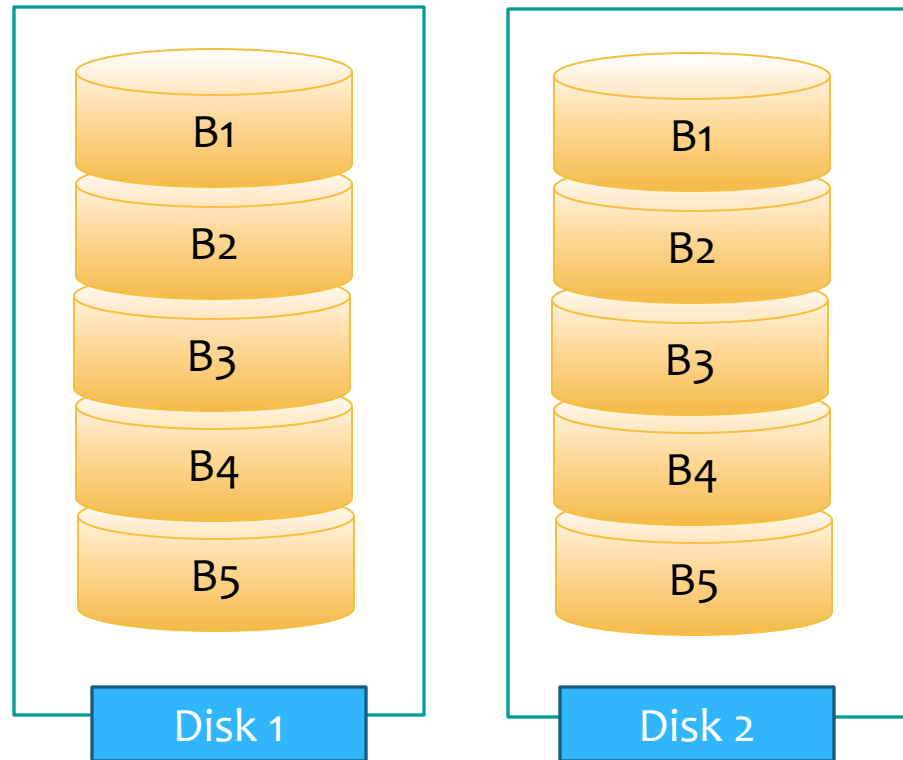
- * **No Reliability**

- * Allows us to **read** even and odd blocks in **parallel**

RAID 1

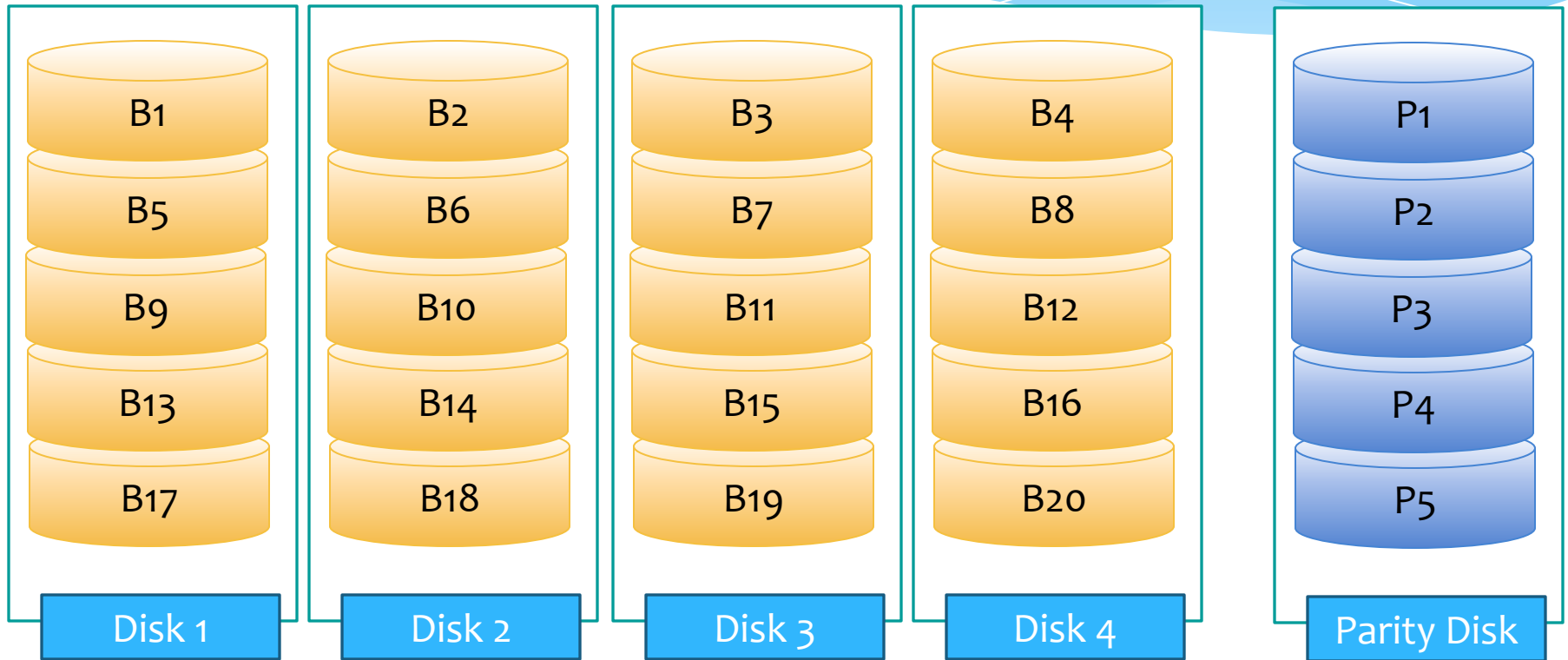


RAID 1

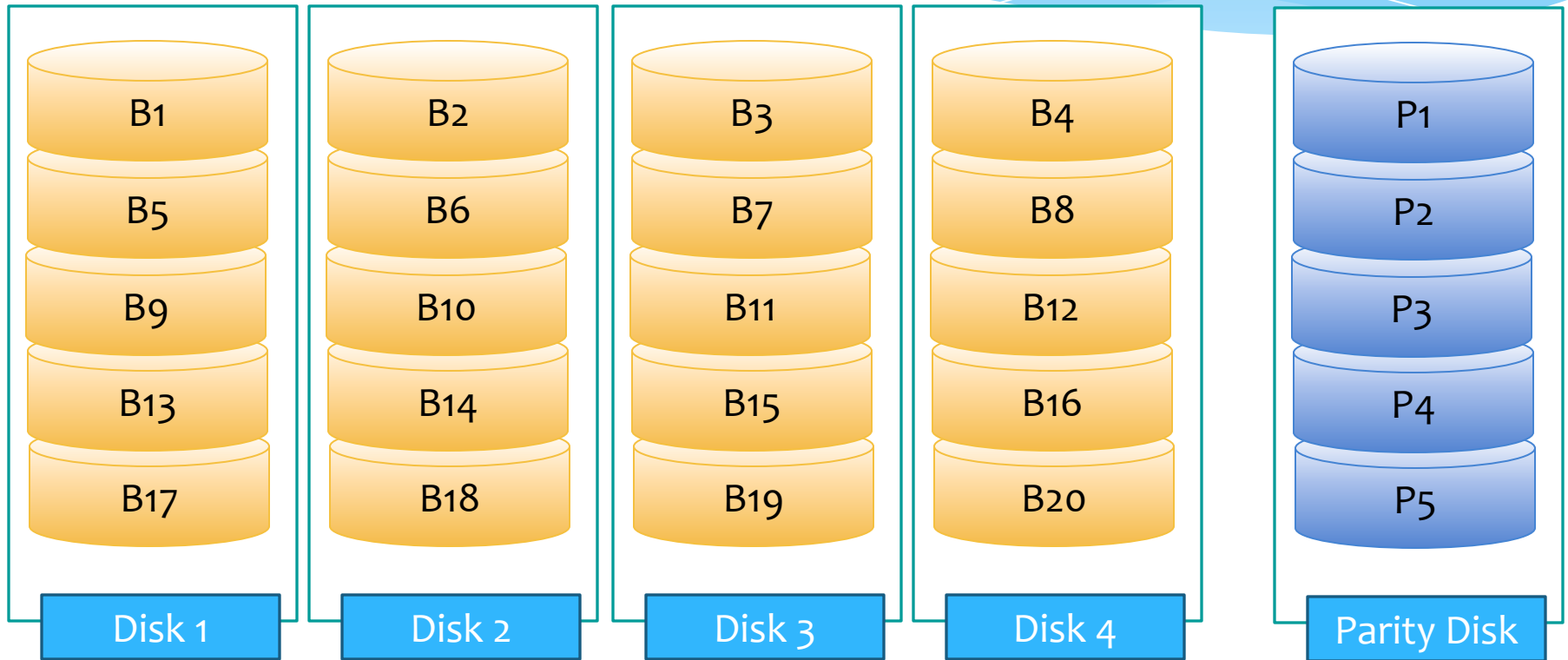


- * **Immune** to one disk failure
- * Can **read** blocks B1 and B2 in parallel
- * 100% overhead in storage

RAID 2, 3, 4

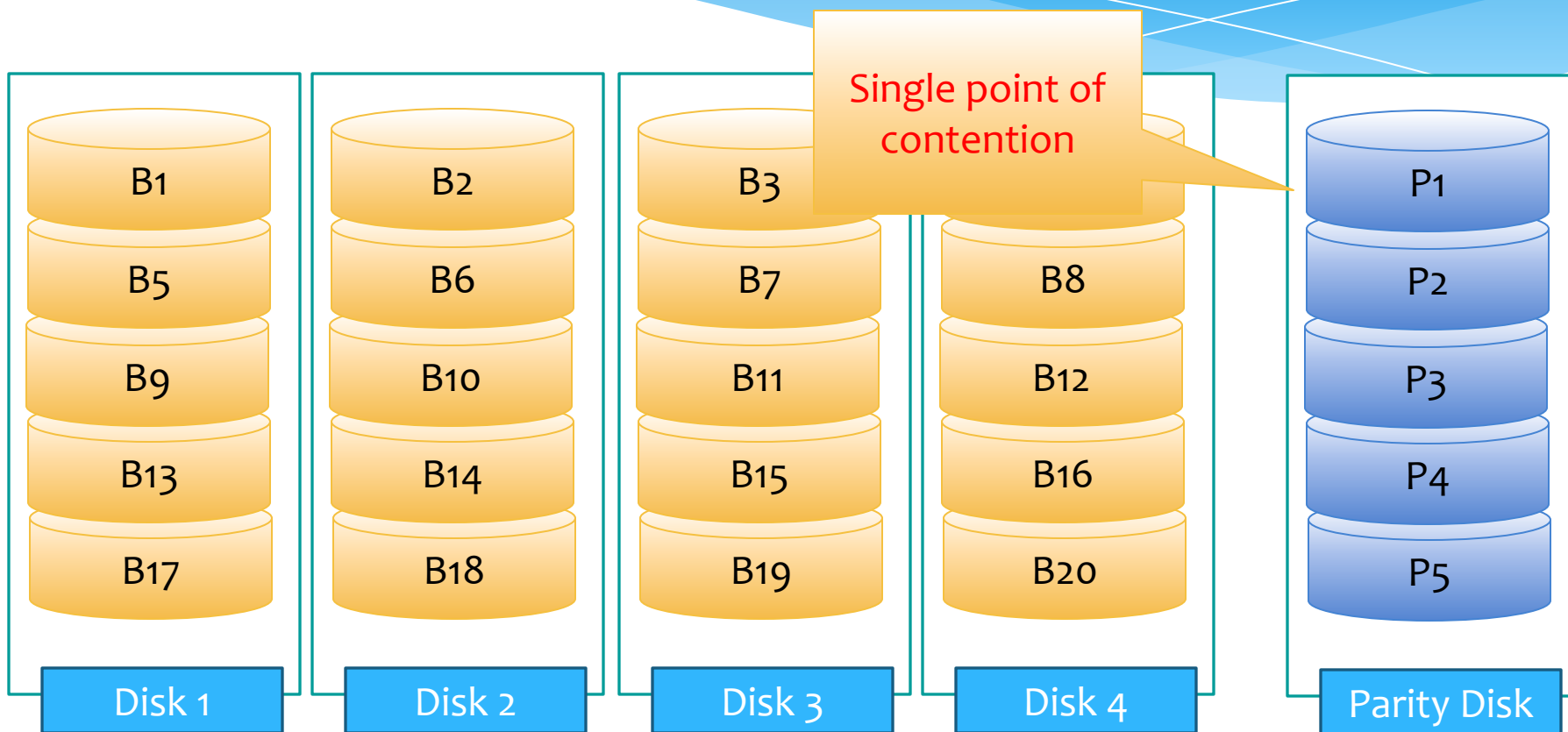


RAID 2, 3, 4



- * **Immune** to one disk failure
- * Parity block $\rightarrow P1 = B1 \oplus B2 \oplus B3 \oplus B4$
- * 25% storage **overhead**

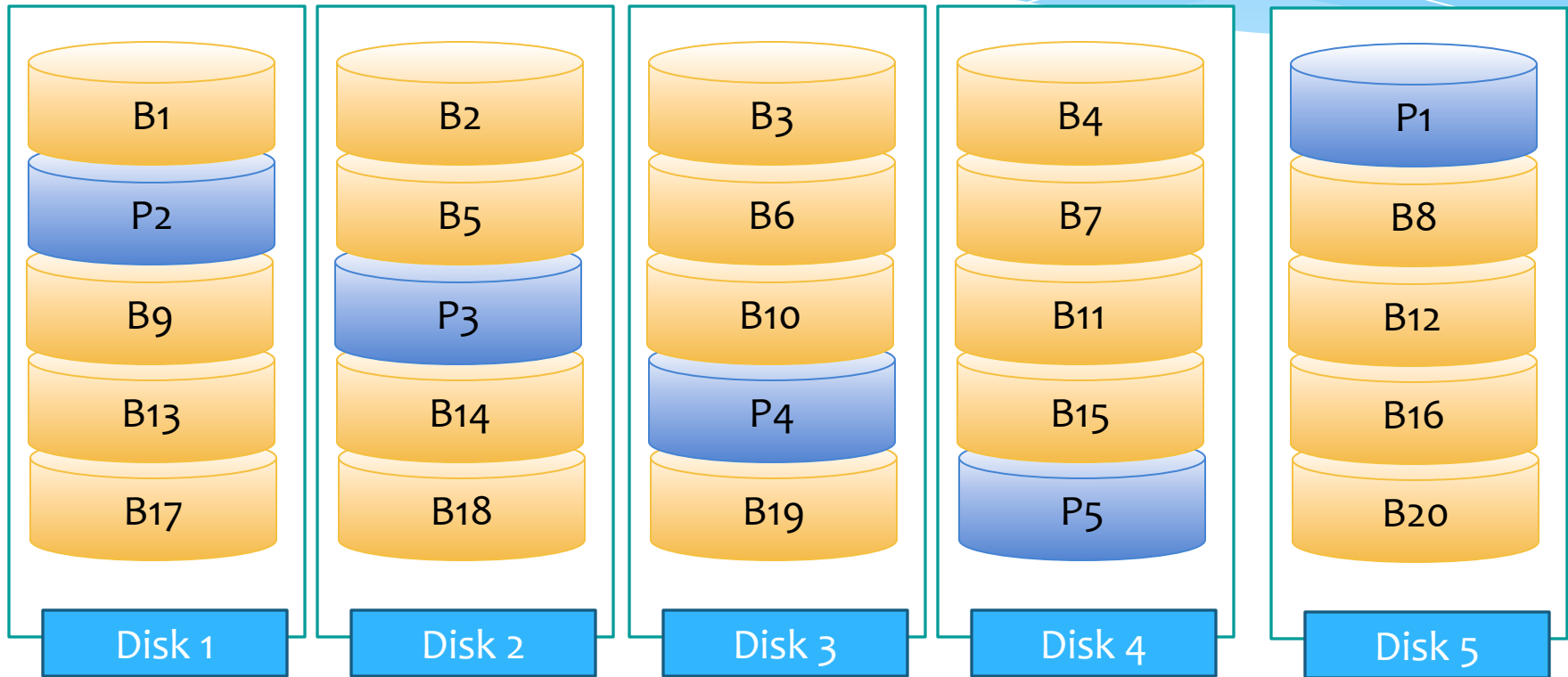
RAID 2, 3, 4



RAID	Block Size
2	1 bit
3	1 byte
4	1 block (512 bytes)

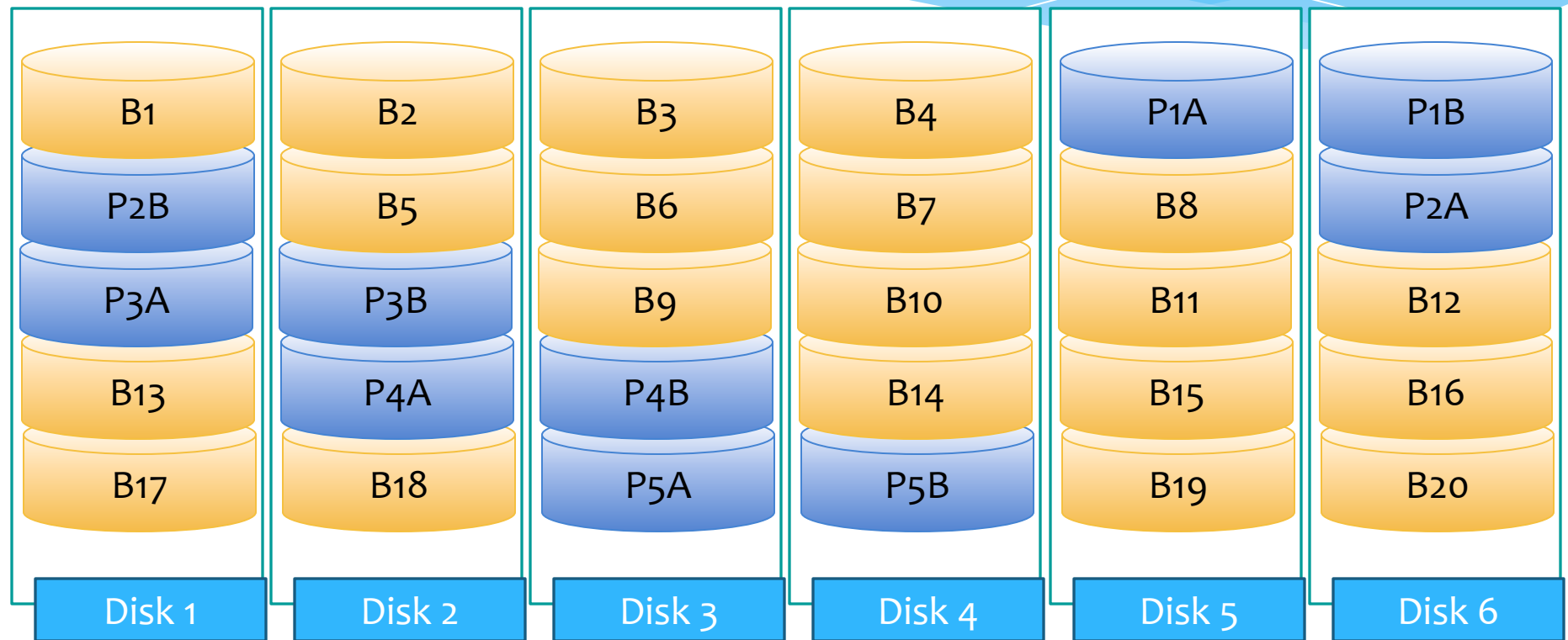
For reading a single block
→ access all the disks

RAID 5



- * **To avoid this problem:** Distribute the parity blocks across the disks
- * Reliable + high bandwidth

RAID 6



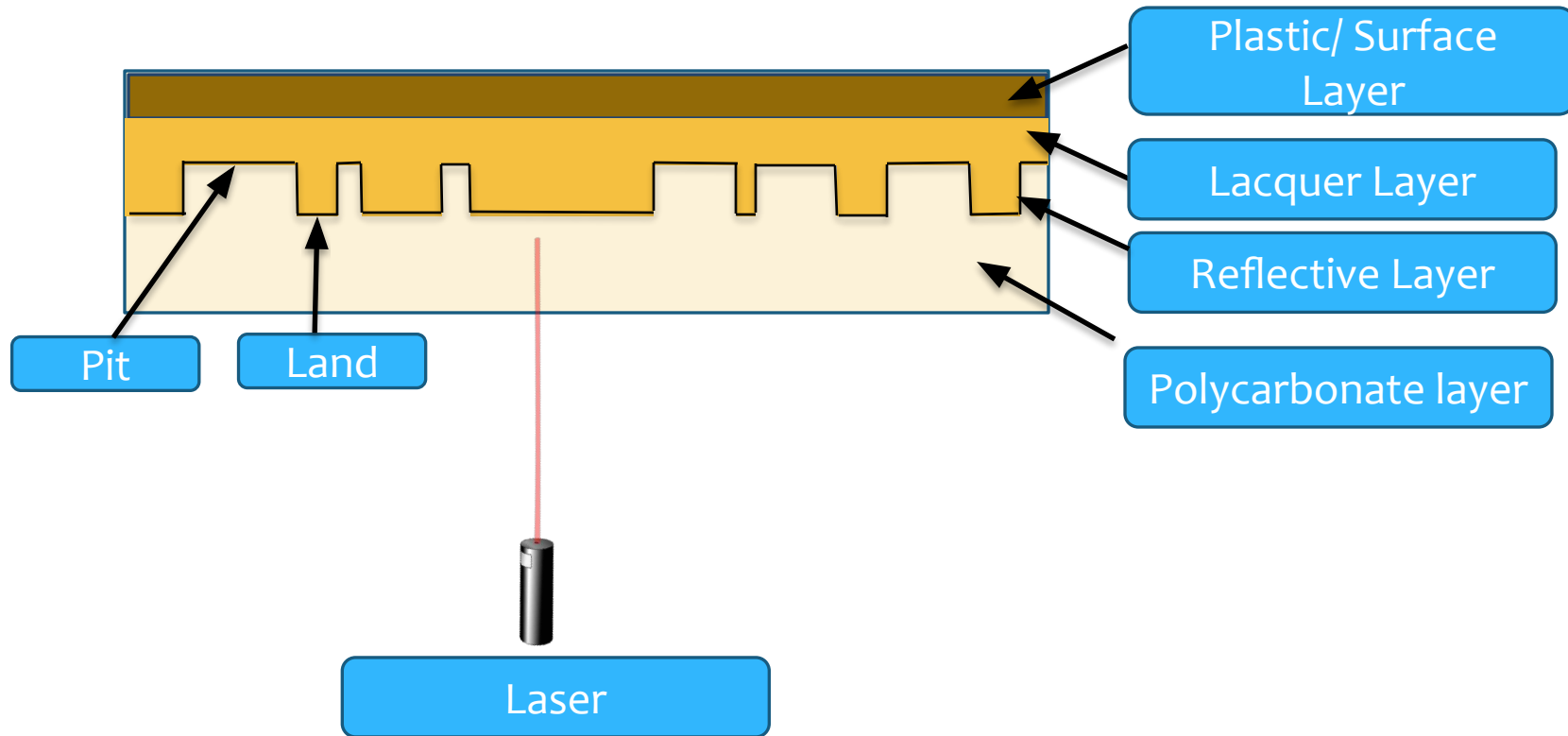
- * **Immune** to two disk failures
- * Have two parity blocks (parity is computed differently)
- * High reliability at the cost of increased storage overhead



Optical Drives

- CD
- DVD
- Blu-Ray

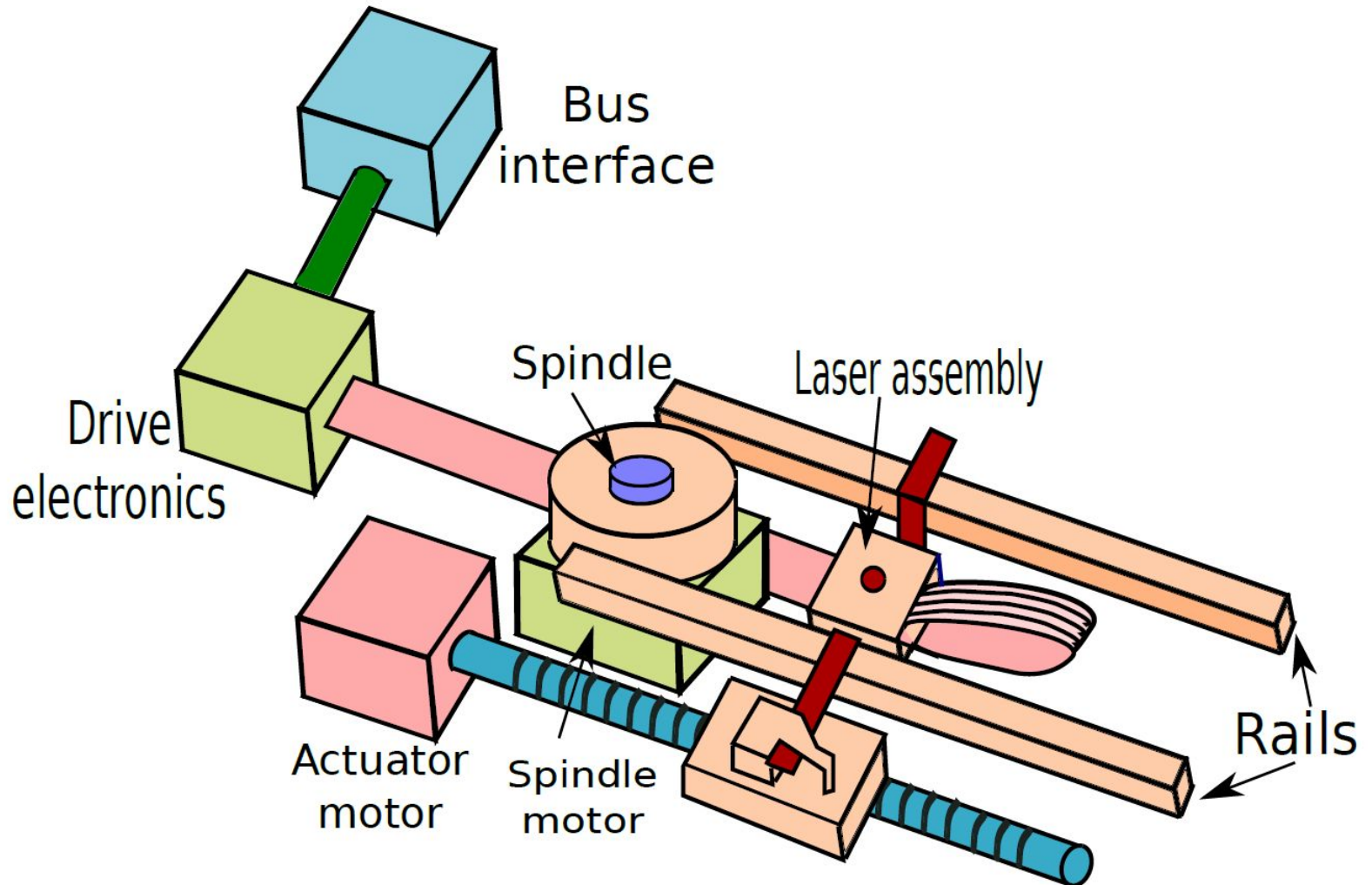
Structure of an Optical Disc



Encoding Data

- * Physical bits:
 - * Lands represent the physical bit 1
 - * Pits represent the physical bit 0
- * Logical bits
 - * Encoded using the **NRZI** encryption scheme
- * Elaborate error protection codes

Design of an Optical Drive

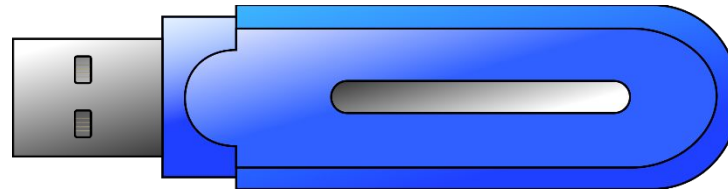


Operation of the Optical Disc Drive

- * Rotates at **constant linear velocity**
 - * $v = \omega r$
 - * $v \rightarrow$ linear velocity
 - * $\omega \rightarrow$ angular velocity
 - * $r \rightarrow$ radius
- * The motor **changes** its speed (rpm) depending on the position of the head
- * It **analyses** the reflected light and figures out the sequences of bits (NRZI scheme)

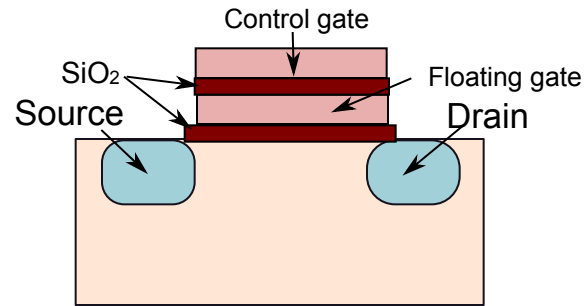
Technology Generations

	CD	DVD	Blu-Ray
Generation	1 st	2 nd	3 rd
Capacity	700 MB	4.7 GB	25 GB
Uses	Audio	Video	High Def. Video
Laser wavelength	780 nm	650 nm	405 nm
Raw 1X transfer rate	153 KB/s	1.39 MB/s	4.5 MB/s

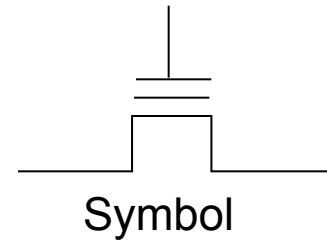


Solid State Drives

Floating Gate (FG) Transistor



(a)



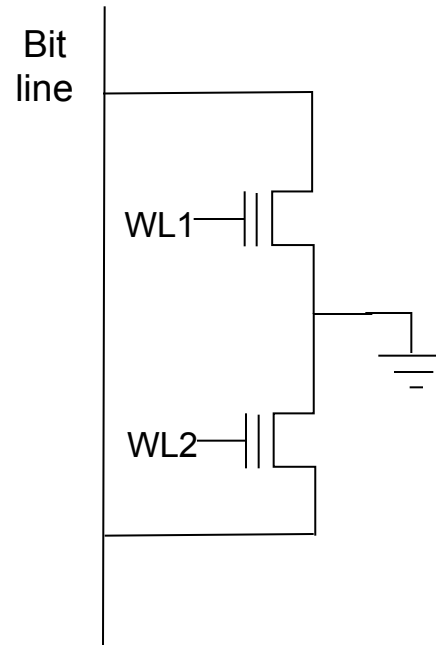
(b)

- * The FG transistor has 2 gates – **control gate** and **floating gate**
- * When we apply a **very high voltage** to the control gate
 - * **Electrons** get deposited in the floating gate
 - * **Increases** the threshold voltage of the transistor
 - * The **FG transistor** is said to be **programmed** (value = 0)
 - * **Not programmed** (value = 1)

Reading the Value in a FG Transistor

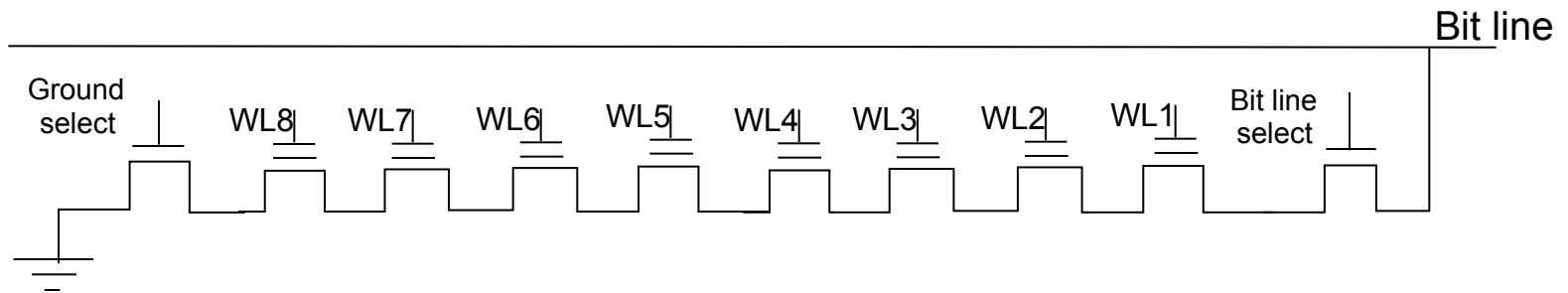
- * Depositing **electrons** increases the threshold voltage
 - * **Increases** from V_t to V_t^+ ($V_t^+ > V_t$)
 - * When the cell contains a **logical 1** $\rightarrow V_t$
 - * When the cell contains a **logical 0** $\rightarrow V_t^+$
- * Set the gate voltage to a value **between** V_t and V_t^+
 - * If a cell has 1, it will **conduct**
 - * Otherwise, it will **not** conduct

NOR Flash



- * Rarely used → low density, low bandwidth, faster
- * Read the value of the bit line using a sense amplifier
- * Operation → Similar to a DRAM array (see Chapter 6)

NAND Flash



- * To **read** a certain transistor, **enable** the rest of the transistors and **ground select** and **bit line select**
- * **High density**
- * 40-60% **more** density than NOR flash

Blocks and Pages

- * A **page** contains 512-4096 bytes of data
 - * Most NAND flash devices **read/write** data at the granularity of pages
 - * Has additional bits for **error correction** (CRC)
- * Blocks contain 32-128 pages
 - * Size: 16 to 512 KB
 - * Can **erase** data at the level of blocks (deprogram)
- * Program Erase Cycle
 - * To **rewrite** a page, we need to erase it first
 - * Converting 0s to 1s (deprogramming) is **slow** and has to be done at **large** granularities

Wear Levelling

- * **Flash memory** can only tolerate a certain number (100k) of program/erase cycles (P/E)
- * The SiO_2 layer breaks down and does not remain an **insulator**, hence the device cannot hold **charge** anymore
- * **Solution :**
 - * Keep **counters** for each block
 - * **Increment** the counter on each P/E cycle
 - * **Aim:** Ensure that the number of P/E cycles for each block is roughly the same.



Wear Levelling and Read Disturbance

- * Wear Levelling

- * Once a counter reaches a **high** value.
- * **Swap** the contents of a frequently accessed block with a less frequently accessed block
- * Keep a high level **mapping** table:
 - * Logical block number → Physical block location
 - * Just **change** the mapping in this table

- * Read disturbance

- * It is **possible** that repeatedly **reading** a transistor causes the contents of the rest of the transistors to **change**
- * **Same approach:** have a counter, **copy** the block to another location, if the counter **exceeds** a threshold.



THE END