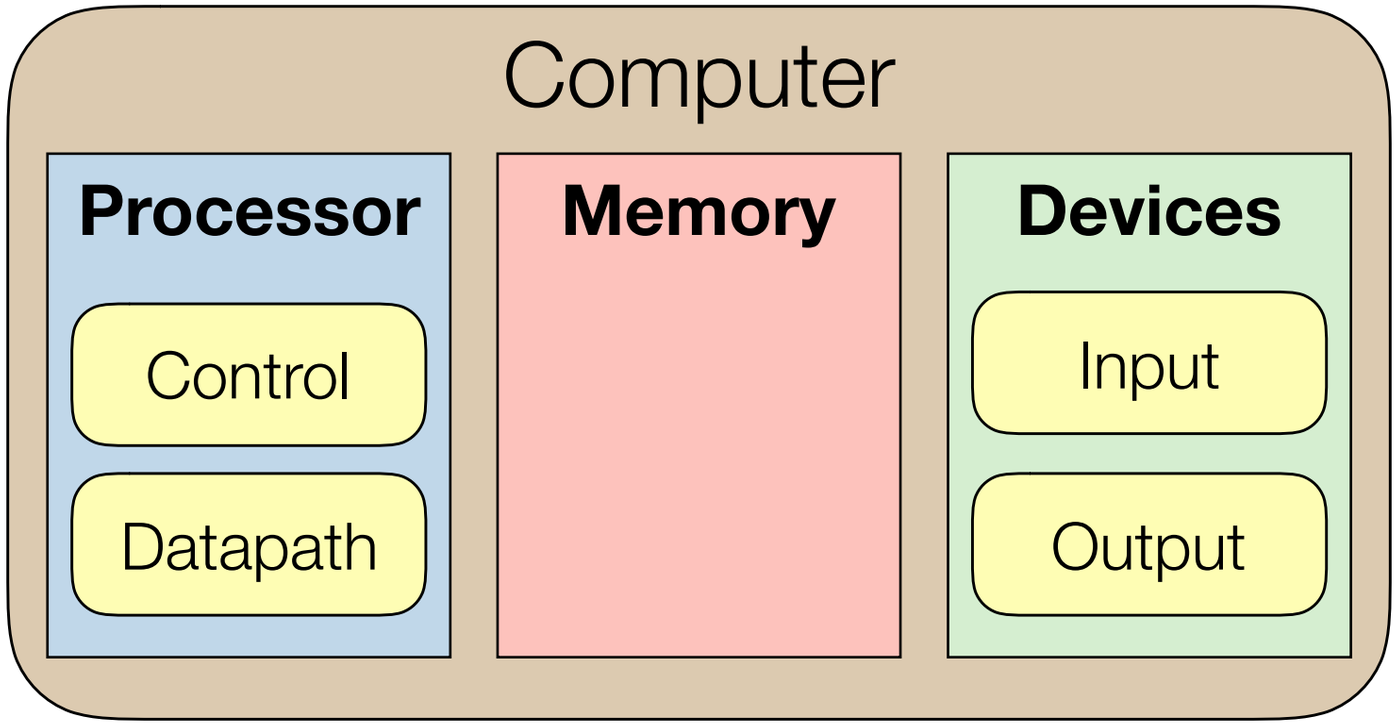
Lecture 23: I/O Performance

Spring 2024
Jason Tang

Topics

- Measuring I/O performance
- Disk I/O performance
- Increasing availability

I/O Design Decisions



- I/O Interface: device drivers, device controller, service queues, interrupt handling
- Design Issues: performance, expandability, standardization, failure prevention
- Impact on Tasks: blocking conditions, priority inversion, access ordering

Impact of I/O on System Performance

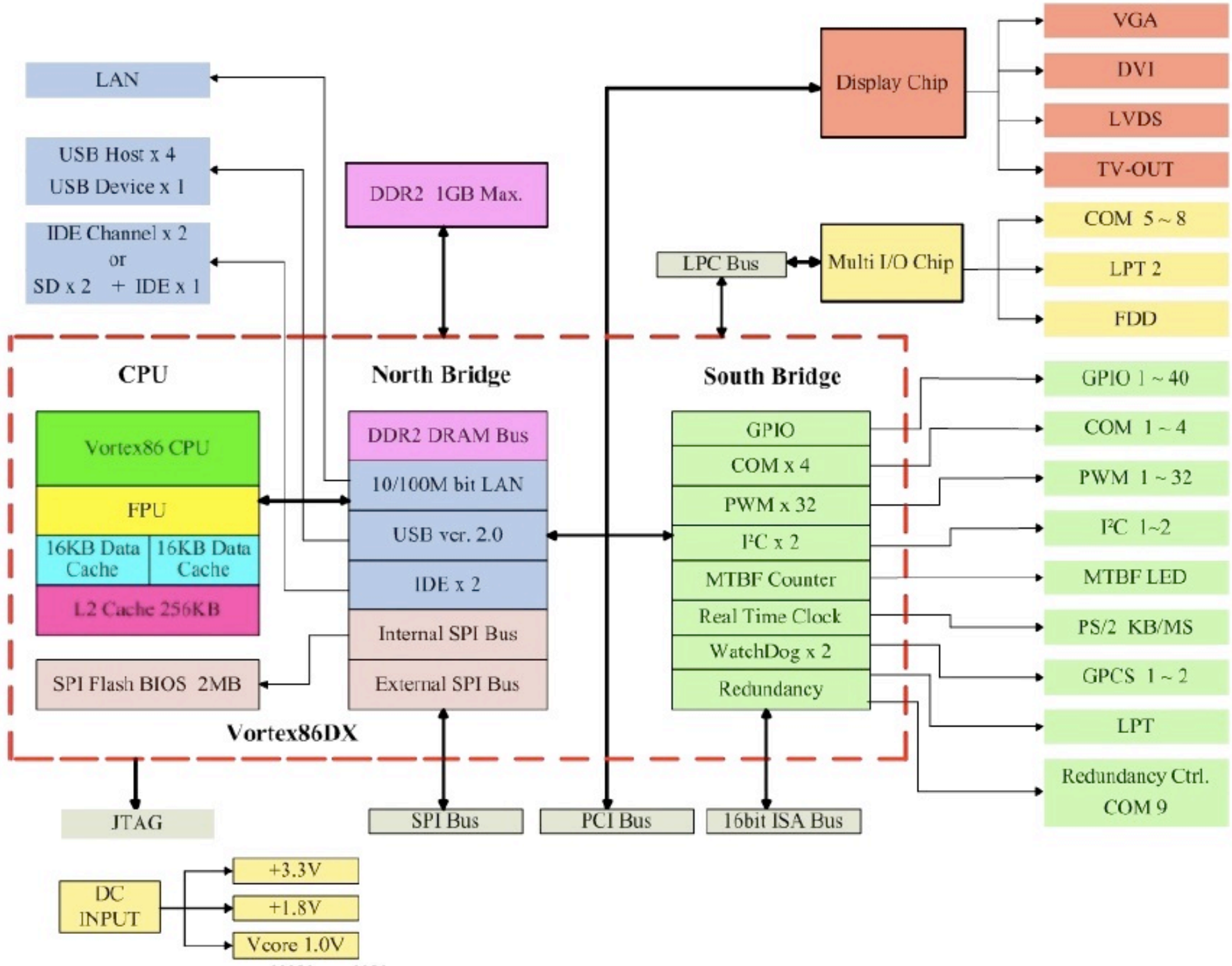
- Example: A particular application takes 100 seconds to execute, where 90 seconds is CPU time and the rest is I/O time. If CPU time is halved every year, but I/O time does not decrease, how much faster will this application run after five years?

Year	CPU Time	I/O time	Total Time	% I/O Time
0	90 s	10 s	100 s	10.00%
1	45 s	10 s	55 s	18.18%
2	22.5 s	10 s	32.5 s	30.77%
3	11.25 s	10 s	21.25 s	47.06%
4	5.625 s	10 s	15.625 s	64.00%
5	2.8125 s	10 s	12.8125 s	78.05%

Typical I/O System

- Usually, a bus connects the processor to devices
    - Modern systems have multiple buses, for different types of devices
- Processors and devices use bus protocols and interrupts to communicate

Typical x86 I/O Layout



https://linuxdevices.org/x86-system-on-chip-adds-vga-graphics/

Modern I/O Device Examples

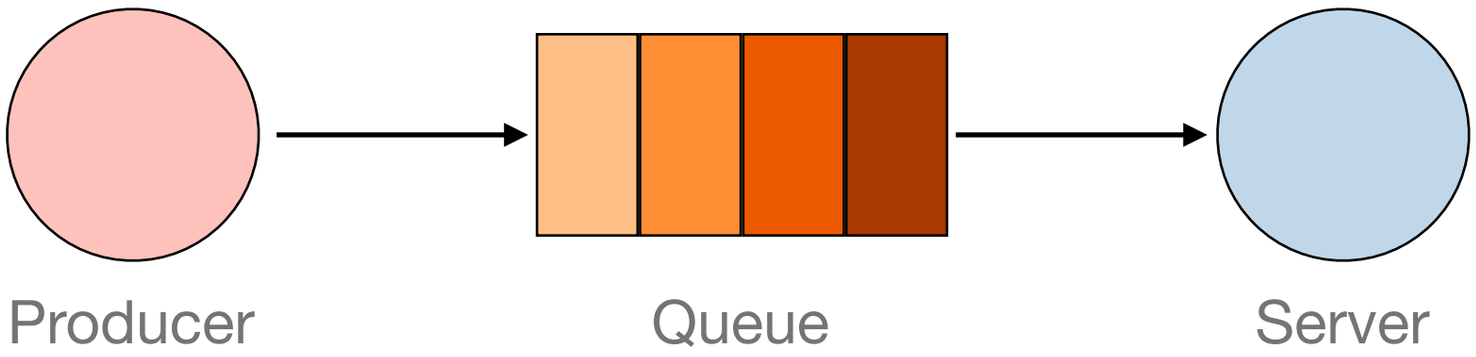
Device	Interconnect	Maximum Data Rate
Keyboard	USB 3.0	3.2 Gb/s
Hard Disk	Serial ATA 3.2	16 Gb/S
Graphics	PCI Express 4.0	64 Gb/S
Local Area Networking	Ethernet 100GbE	100 Gb/S
High Performance Computing	InfiniBand HDR 12x	600 Gb/S

Measuring I/O Performance

- I/O system performance depends upon:
  - Hardware: CPU, memory systems, buses, controller, device itself
  - Software: operating system, application itself

- Two common I/O performance metrics (similar to CPU metrics):
  - Response Time: latency
  - Throughput: I/O bandwidth

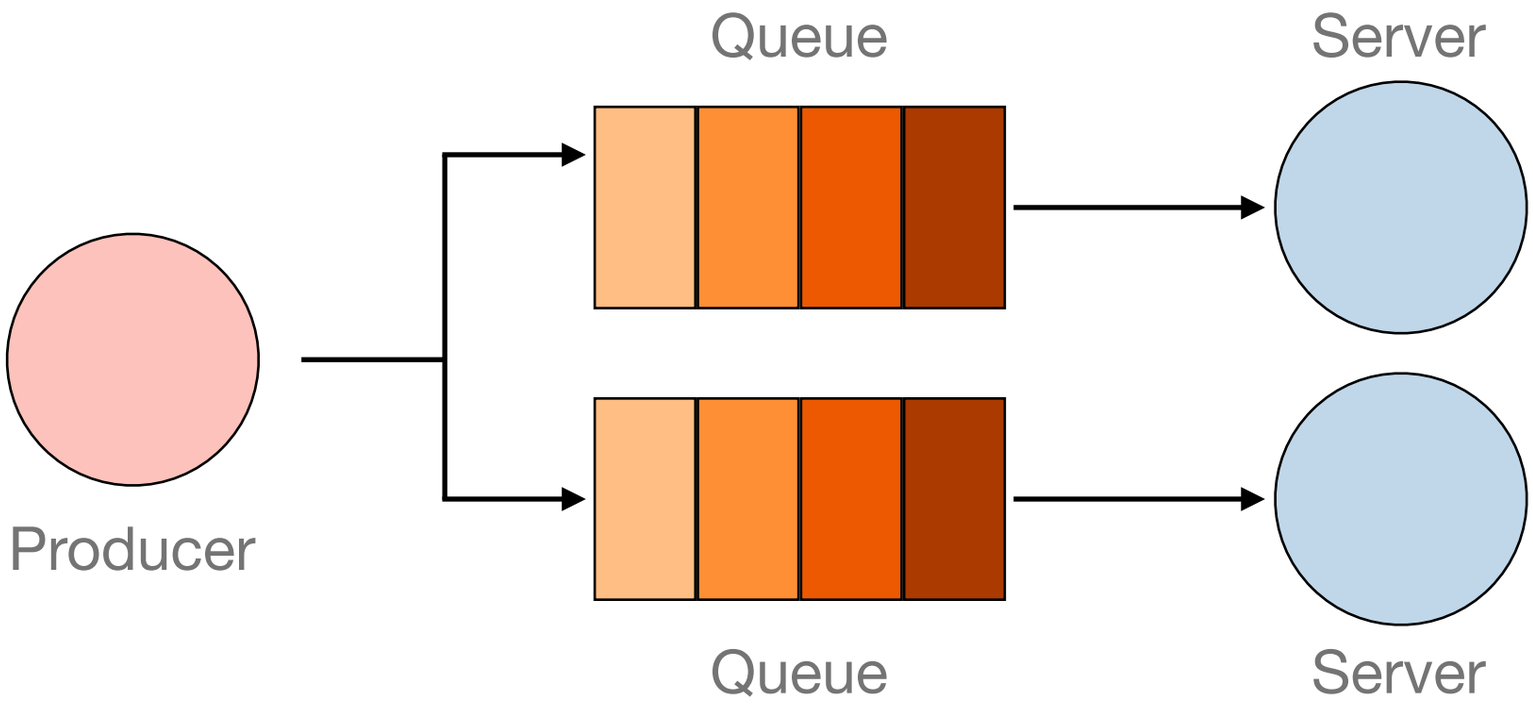
Producer/Server Model



- Throughput: number of tasks completed per unit time
  - To maximize throughput, queue should never be empty and server should never be idle
- Response time: elapsed time from when a task is enqueued until task begins
  - To minimize response time, queue should be empty and server is idle

Low response time is user-desirable but is system-undesirable

Producer/Server Model



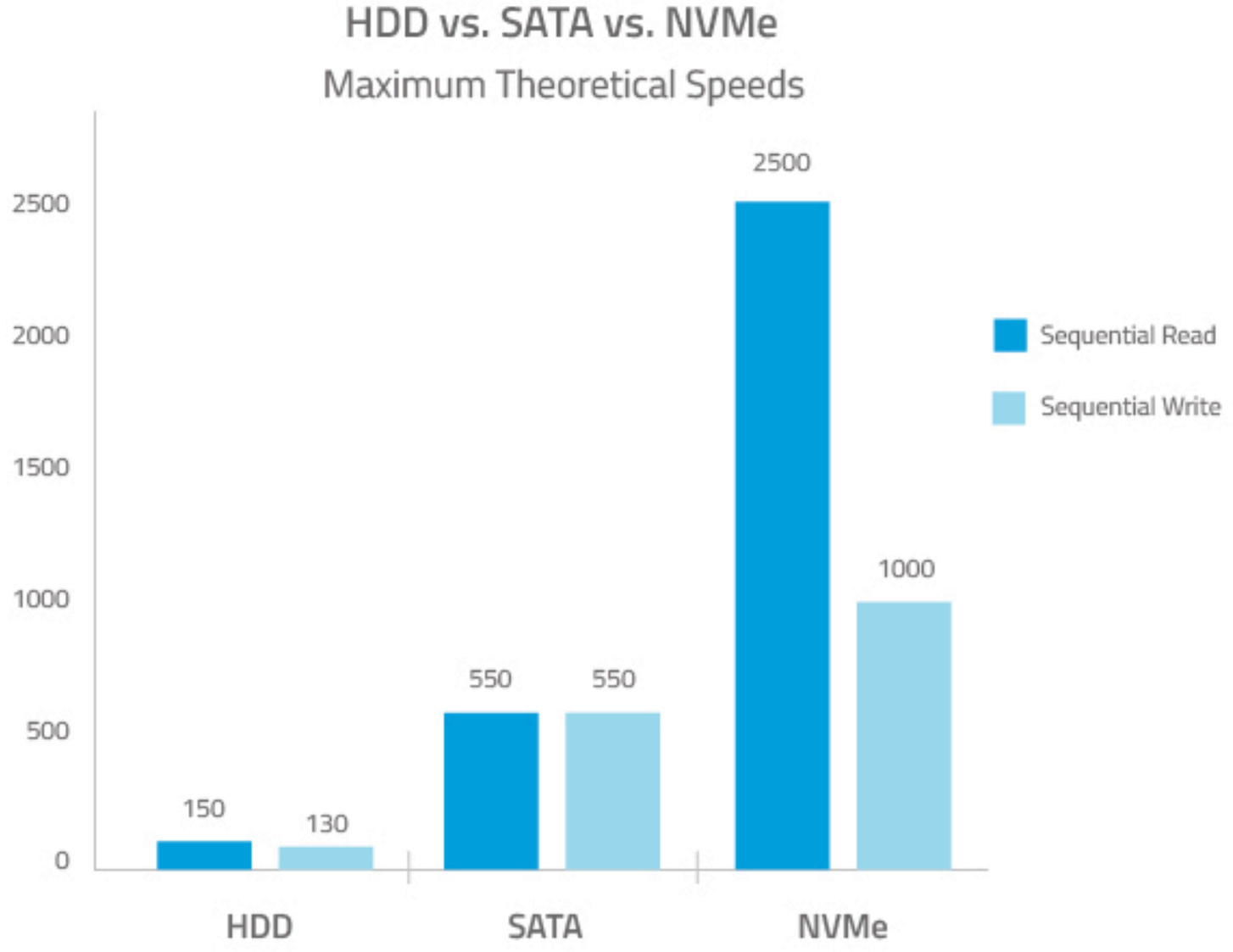
- Throughput can be increased by adding more hardware to problem
- Response time is harder to reduce, limited by mechanical subsystem

Disk Access Patterns

- Supercomputer:
  - Few large files, one large read and many small writes
  - Data Rate: MB/s between memory and disk
- Database Server:
  - Frequent small changes to large shared database
  - I/O Rate: Number of disk accesses / second

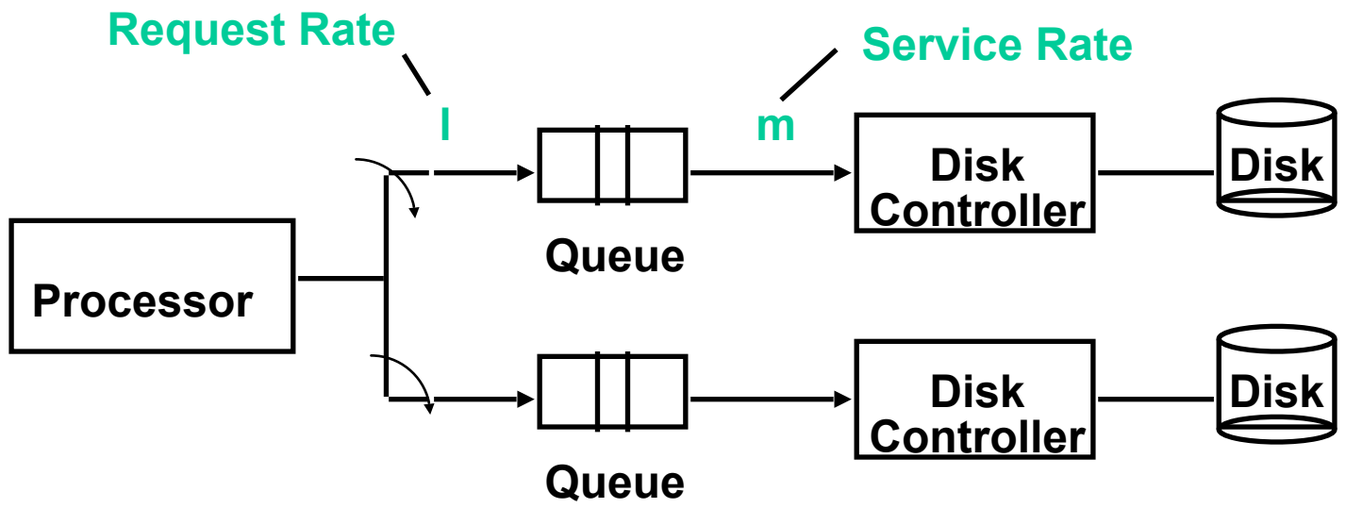
Disk Performance

- Different ways to measure “disk performance”: sequential read, sequential write, random read, random write, power consumption, responsiveness, etc.



https://www.atpinc.com/blog/nvme-vs-sata-ssd-pcie-interface

Hard Disk I/O Performance



- Hard Disk Access Time = Seek Time + Rotational Latency + Transfer Time + Controller Time + Queuing Delay
  - A hard disk platter rotates on average 1/3 when seeking
  - For solid state disks, seek and transfer times are much faster, and there is no rotational latency

Hard Disk I/O Performance Example

- Example: A 5400 RPM hard disk transfers 4 MB/s. The average seek time is 12 ms. The controller overhead is 1 ms. Assume that queue is idle (thus no service time). What is its average disk access time for a single sector?

  - Usually 512 bytes per sector

$$
\begin{aligned}
&= 12\ ms + \frac{1}{3} \times (5400\ RPM)^{-1} + \frac{512\ B}{4\ MB/s} + 1\ ms + 0\ ms \\
&= 12\ ms + \frac{1}{3} \times (90\ RPS)^{-1} + \frac{2\ B}{15625\ B/s} + 1\ ms + 0\ ms \\
&= 12\ ms + \frac{1}{270}\ s + \frac{2}{15625}\ s + 1\ ms + 0\ ms \\
&= 12\ ms + 3.7\ ms + 0.128\ ms + 1\ ms + 0\ ms \\
&= 16.828\ ms
\end{aligned}
$$

Reliability and Availability

- Reliability: how long a system performs as intended (is it broken?)
- Availability: percent time that a system is an operable state (can I use it?)
  - Reliability and availability are related, but are **not** the same
- Mean Time Between Failure (MTBF): how long until system breaks, on average
- Mean Time to Repair (MTTR): how quickly to fix system

$$Availability = \frac{MTBF}{MTBF + MTTR}$$

http://www.barringer1.com/ar.htm

Increasing MTBF

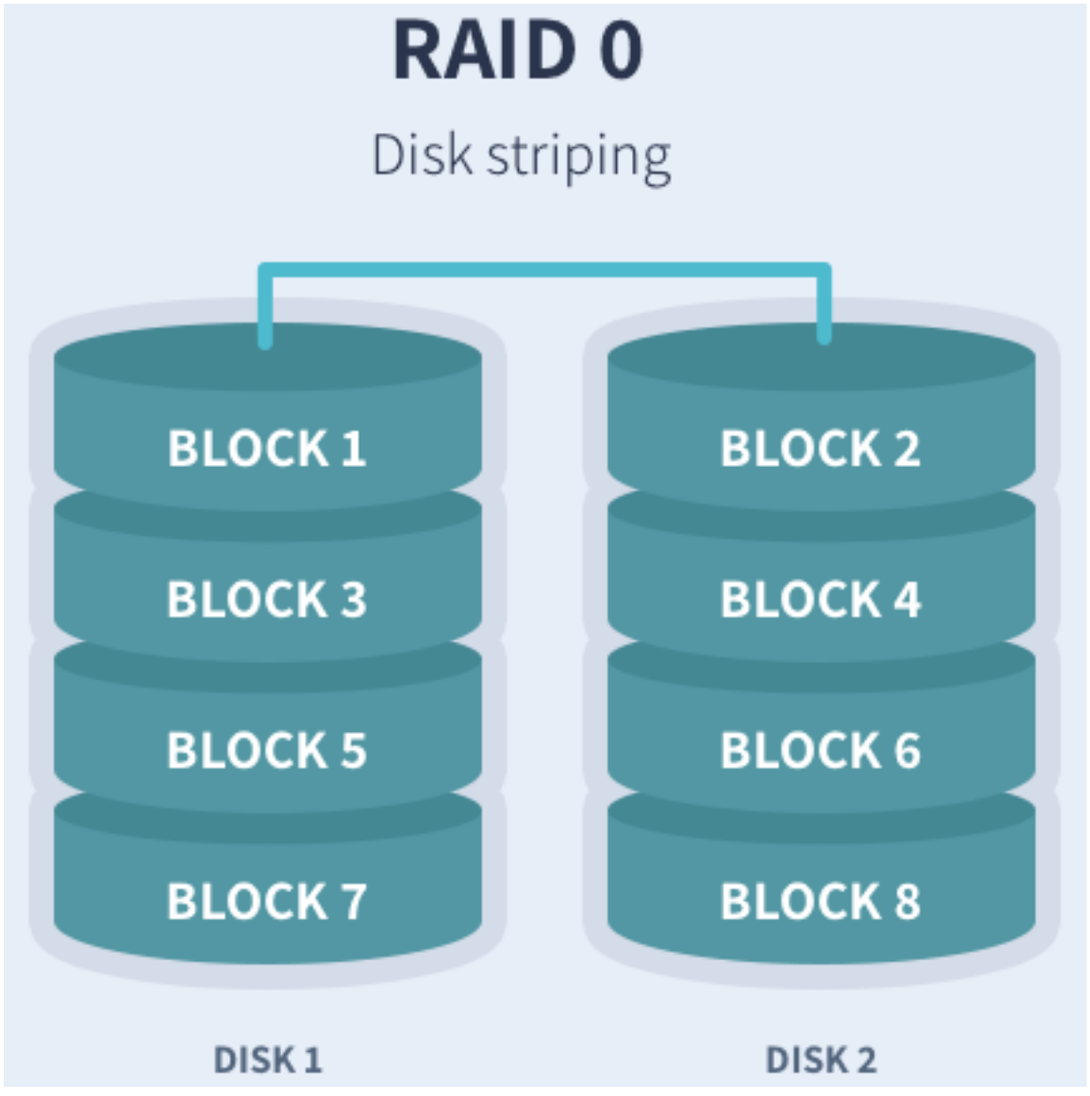
- Many ways to increase MTBF (and thus improve availability):
  - Prevent faults by improving manufacturing
  - Use redundancy to allow system to continue processing despite faults
  - Predict faults, allowing component to be replaced *before* failure
- Can thus design hardware to increase availability

RAID

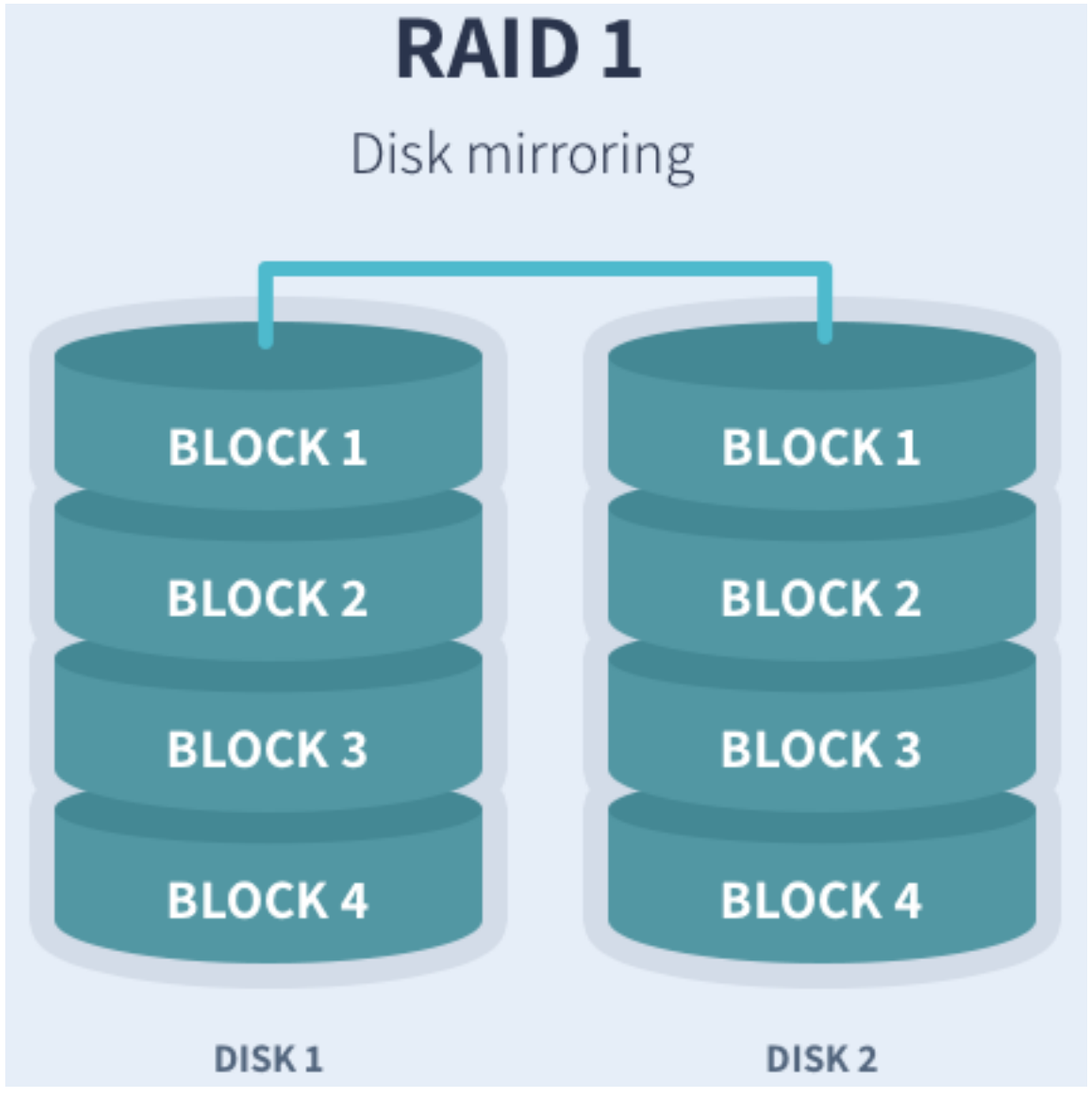
- Redundant Array of Inexpensive Disks (RAID): use multiple disks to store data
- Different configurations, all resulting in improving some aspect of performance
  - RAID 0: striping
  - RAID 1: mirroring
  - RAID 3: parity
  - RAID 5: striping + parity

https://datapacket.com/blog/advantages-disadvantages-various-raid-levels/

RAID

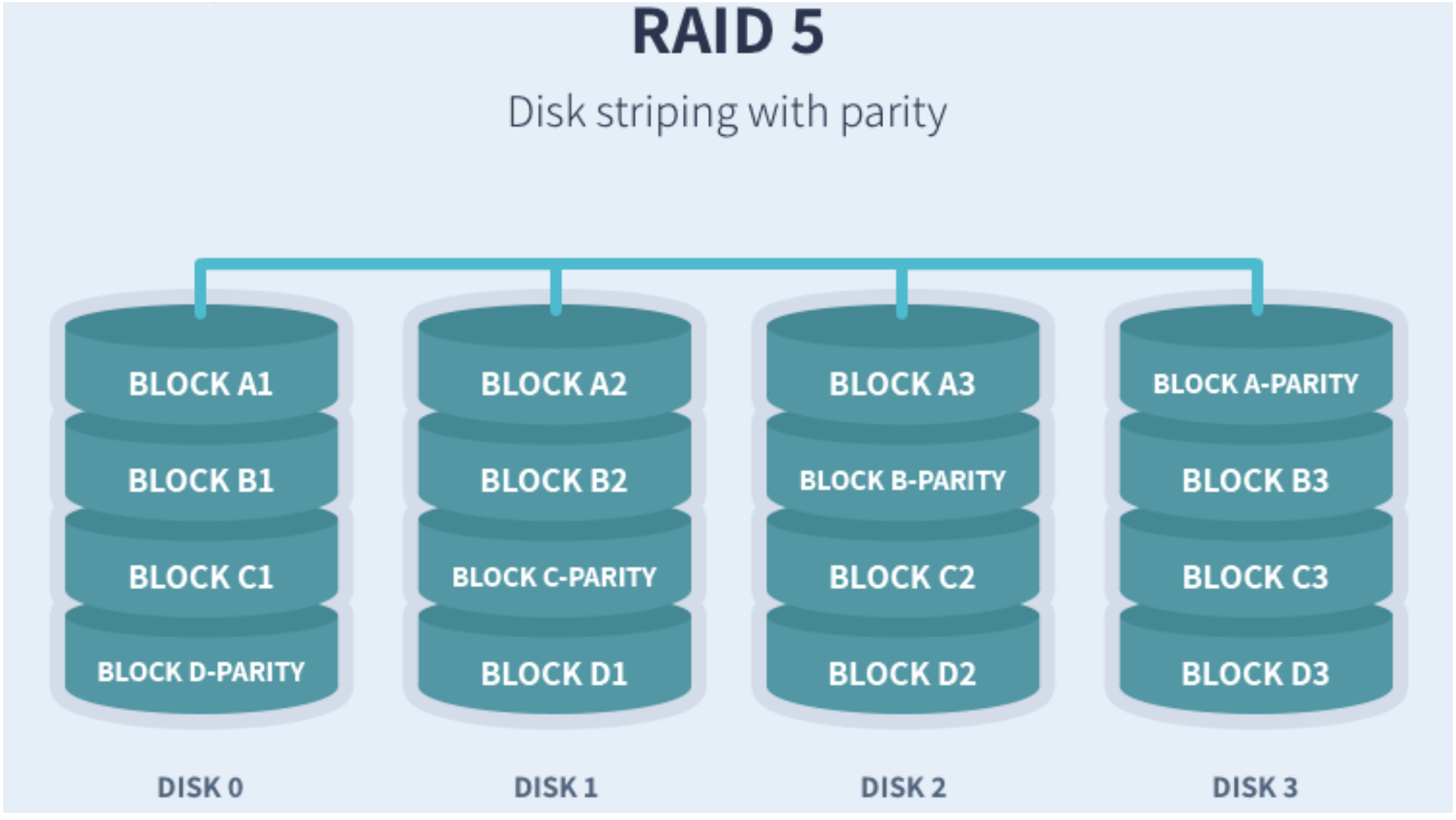


- Increase read and write speed, but failure of one disk loses all data



- Increase read speed (no change to write speed), provide redundancy, but doubles cost of storage

RAID



- Increase read and write speed, but not as fast as RAID 0

- Higher cost for storage, but not as expensive as RAID 1

Parity

- Algorithm that can detect errors in a string of data bits
- In simplest case, append a parity bit after data
    - Set parity bit to 0 when data bits have an even number of bits that are 0, otherwise set parity bit to 1 (so-called even parity)
    - Thus if number of zeroes of (data + even parity bit) is ever odd, then an error occurred
- With single parity bit, can detect, but not correct, a single bit flip
    - Unable to detect double bit flip

More Complex Parity Algorithms

- When multiple parity bits are calculated and stored with data stream, then hardware / software can correct multiple errors
    - Cyclic Redundancy Check (CRC): Ethernet, Zip archives
    - Hamming: Used in Error-correcting Code (ECC) memory
    - Reed-Solomon: Used in optical medium (CD, DVD, Blu-Ray)
    - Bose-Chaudhuri-Hocquenghem (BCH): Used in solid-state drives

Hamming Code

- By increasing parity bits, a Hamming Code allows correcting single bit error correction, and double bit error detection (so-called SEC/DED)

- Hamming code numbers bit positions from right to left, with the rightmost bit having bit position 1 (not 0)

- Check Bits at positions that are a power of 2 are even parity bits; all other bits are data bits

- Check Bits cover overlapping portions of the data

8-bit Data Hamming Code

Bit Position	12	11	10	9	8	7	6	5	4	3	2	1
Encoded Data Bits	d8	d7	d6	d5	p8	d4	d3	d2	p4	d1	p2	p1
p1		X		X		X		X		X		X
p2		X	X			X	X			X	X	
p4	X					X	X	X	X			
p8	X	X	X	X	X							

- Responsible check bits = bit position, as expressed as its binary value

- An extra check bit over all bits can then detect double bit errors

Hamming Code Example

- Let data to encode = 0x9A (binary 1001 1010)

Bit Position	12	11	10	9	8	7	6	5	4	3	2	1
Encoding	1	0	0	1	p8	1	0	1	p4	0	p2	p1
p1		X		X		X		X		X		X
p2		X	X			X	X			X	X	
p4	X					X	X	X	X			
p8	X	X	X	X	X							

  - P1 checks all odd numbered positions = (0, 1, 1, 1, 0) = 1
  - P2 checks bits 11, 10, 7, 6, 3, 2 = (0, 0, 1, 0, 0) = 1
  - P4 checks bits 12, 7, 6, 5, 4 = (1, 1, 0, 1) = 1
  - P8 checks bits 12, 11, 10, 9, 8 = (1, 0, 0, 1) = 0
- Resulting code (the syndrome) = 1001 0101 1011

Hamming Code Example

- If stored data bits are 0111 0010 1110, what is the corrected value?

Bit Position	12	11	10	9	8	7	6	5	4	3	2	1
Encoding	0	1	1	1	0	0	1	0	1	1	1	0
p1		X		X		X		X		X		X
p2		X	X			X	X			X	X	
p4	X					X	X	X	X			
p8	X	X	X	X	X							

  - Calculated parity for P1 = (1, 1, 0, 0, 1, 0) = 1
  - Calculated parity for P2 = (1, 1, 0, 1, 1, 1) = 1
  - Calculated parity for P4 = (0, 0, 1, 0, 1) = 0
  - Calculated parity for P8 = (0, 1, 1, 1, 0) = 1
- Because parity bits P1, P2, and P8 are odd, then bit 11 (1 + 2 + 8) is flipped

SEC/DED Example

Bit Position	12	11	10	9	8	7	6	5	4	3	2	1	0
Encoding	1	0	1	1	1	1	1	0	1	0	0	0	1
$p_{overall}$	X	X	X	X	X	X	X	X	X	X	X	X	X
p1		X		X		X		X		X		X	
p2		X	X			X	X			X	X		
p4	X					X	X	X	X				
p8	X	X	X	X	X								

- Let received data = **0xbc**, parity bits = **0x19**
  - $P_{overall}$ = (1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1) = **0** (correct)
  - P1 = (0, 1, 1, 0, 0, 0) = **0** (correct)
  - P2 = (0, 1, 1, 1, 0, 0) = **1** (error)
  - P4 = (1, 1, 1, 0, 1) = **0** (correct)
  - P8 = (1, 0, 1, 1, 1) = **1** (error)
- $P_{overall}$ indicates 0 errors, but P2 + P8 indicate flips = **Double Errors**