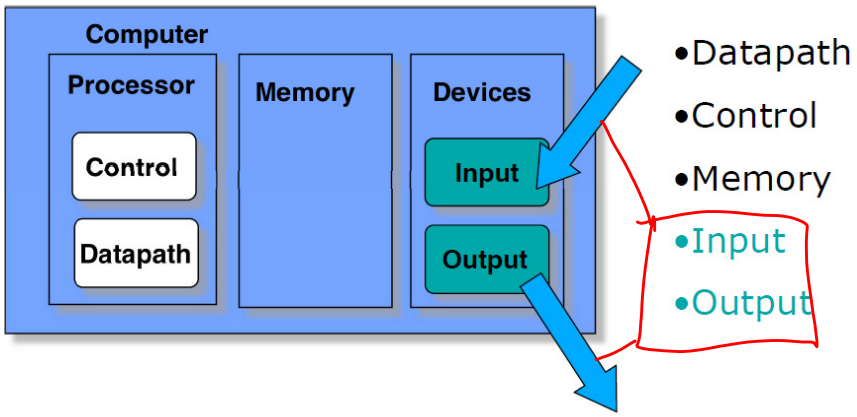
Input/Output



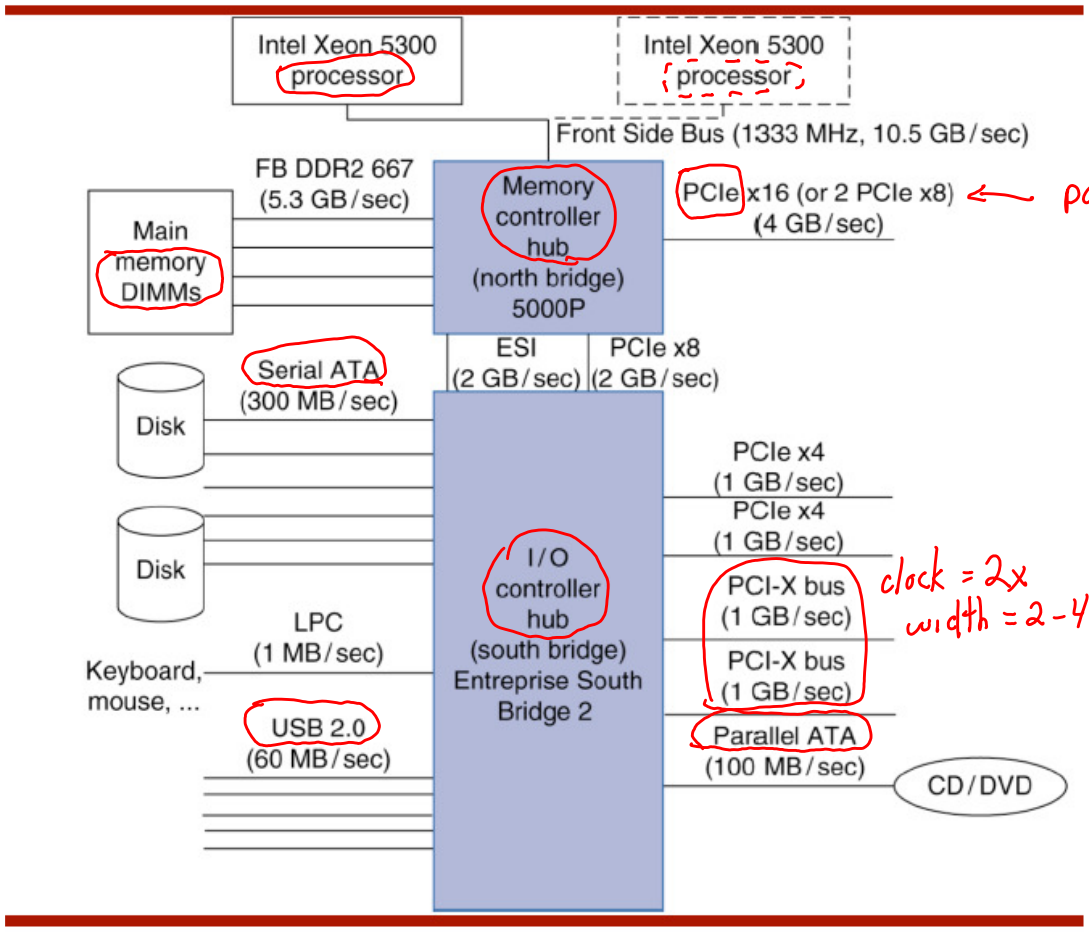
- I/O devices can be characterized by
  - Behaviour: input/output, storage
  - Partner: human or machine ← who's at the other end?
  - Data rate: bytes/sec, transfers/sec ← which is it better at?

Device	Behavior	Partner	Data Rate (KB/sec)
Keyboard	Input	Human	0.01
Mouse	Input	Human	0.02
Line Printer	Output	Human	1.00
Laser Printer	Output	Human	100.00
Graphics	Output	Human	100,000.00
Network-LAN	Communication	Machine	10,000.00
Floppy disk	Storage	Machine	50.00
Optical Disk	Storage	Machine	10, 000.00
Magnetic Disk	Storage	Machine	30,000.00

$\times 10^4$

$\times 10^3$

Typical x86 PC I/O System



point-to-point, lanes

crossbar

any-to-any

parallel

grouped, bit serial

clock = 2x

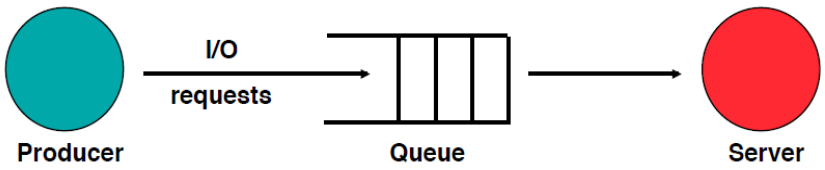
width = 2-4x

- Performance measures
  - Latency (response time)
  - Throughput (bandwidth)
- Dependability is important
  - Resilience in the face of failures
  - Particularly for storage devices
- Expandability

- Computer classes
  - Desktop: response time and diversity of devices
  - Server: throughput, expandability, failure resilience
  - Embedded: cost and response time

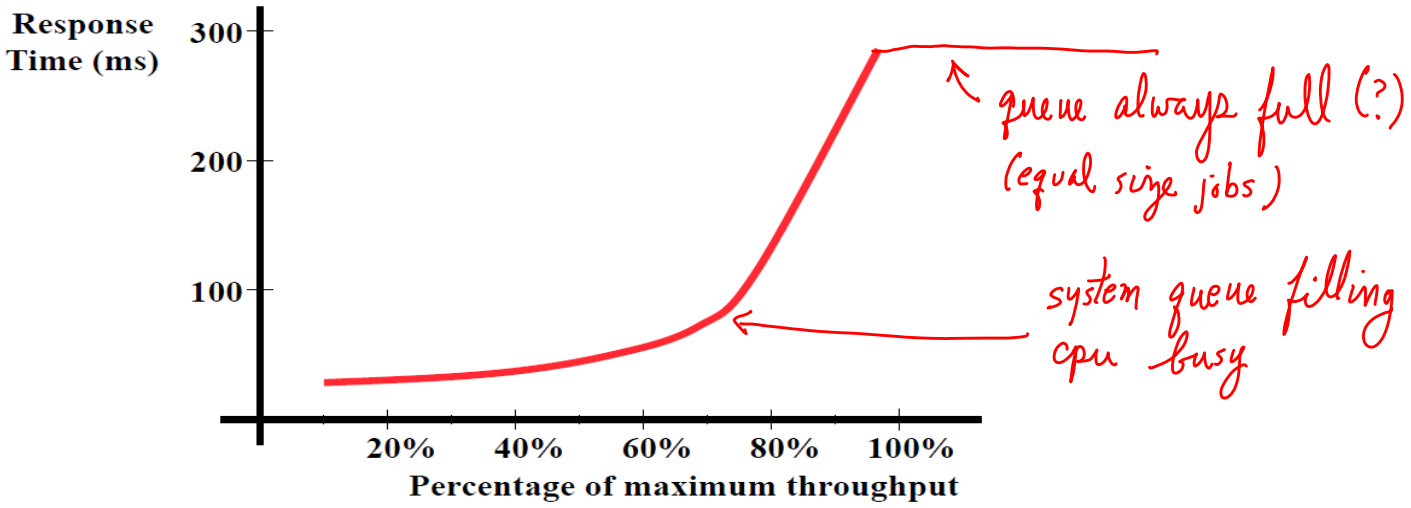
- Throughput
  - Aggregate measure of amount of data moved per unit time, averaged over a window
    - Measure in bytes/sec or transfers/sec
  - Sometimes referred to as bandwidth
    - Examples: Memory bandwidth, disk bandwidth

- Response time
  - Response time to do a single I/O operation
    - Measured in seconds or cycles
  - Sometimes referred to as latency
    - Example: Write a block of bytes to disk
    - Example: Send a data packet over the network

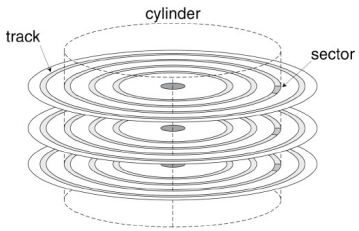


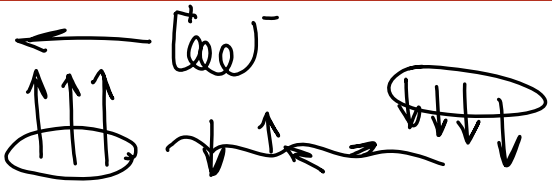
- Throughput is the number of tasks completed by server per unit time
- Response time is the elapsed time between tasks entering queue and tasks completed by the server ← enter queue, exit system
- Tradeoff between throughput and response time
  - Highest possible throughput is when the server is always busy and the queue is never empty
  - Fastest response time is when the queue is empty and the server is idle when the task arrives

Throughput vs. Response Time

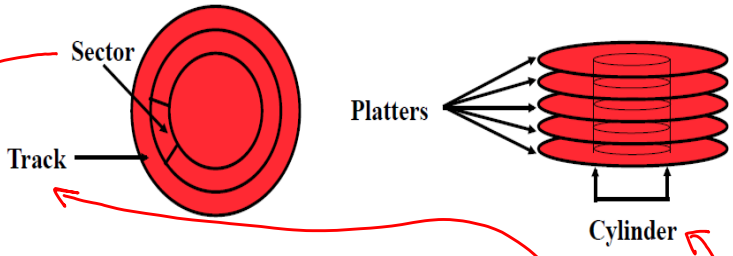


- Nonvolatile, rotating magnetic storage





OR



- >=1 platter per disk with disk heads on both sides
- Platters are divided in concentric tracks
- Each track is divided in sectors
  - Unit of transfer to/from disk (i.e., disk block)
  - Some disks have a constant number of sectors per track
  - Others keep constant bit density which places more sectors on outer track
- A common track across multiple platters is referred to as a cylinder

arm in same seek location

- Basic operation
  - Rotating platter coated with magnetic surface
  - Moving read/write head used to access the disk
- Features of hard disks
  - Platters are rigid (ceramic or metal)
  - High density since head can be controlled more precisely
  - High data rate with higher rotational speed
  - Can include multiple platters
- Incredible improvements
  - Example of I/O device performance being technology driven
  - Capacity: 2x every year
  - Transfer rate: 1.4x every year
  - Price approaching 1$/GB = 10^9 bytes

> growing gap?

- Each read or write has three major components
  - *Seek time* is the time to position the arm over the proper track
  - *Rotational latency* is the wait for the desired sector to rotate under the read/write head
  - *Transfer time* is the time required to transfer a block of bits (sector) under the read/write head

- Note that these represent only the "raw performance" of the disk
  - Other components: I/O bus, controllers, other caches, interleaving, etc.

↑ and ↑ ↑ and ↑ and ...

- Industry definition is that seek time is the time for all possible seeks divided by the number of possible seeks
  - In practice, locality reduces this to 25-33% of this number ?
  - Note that some manufacturers report minimum seek times rather than average seek times

e.g.

- Average rotational latency
  - Average rotational latency = 0.5 rotation / RPM
- Example: 7200 RPM

$$Average\ rotational\ latency = \frac{0.5\ rotation}{7200\ RPM} = \frac{0.5\ rotation}{7200\ RPM\ /(60\ \text{sec}/\ \text{min})} = 4.2ms$$

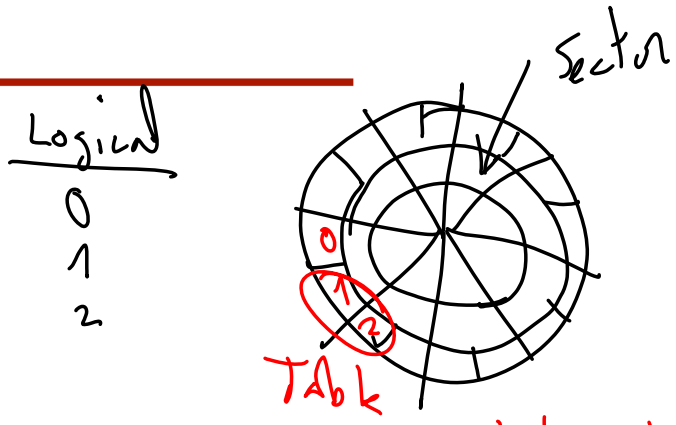
- Transfer time is the time required to transfer a block of bits
- A factor of the transfer size (Block), rotational speed, and recording density
  - Transfer size is usually a sector
- Most drives today use caches to help buffer the effects of seek time and rotational latency

← drive controller HW onboard disk aka drive controller HW on interface board in CPU (controller)

Typical Hard Drive

- Typical hard disk drive
  - Rotation speed: 3600, 5200, 7200, or 15000 RPM
  - Tracks per surface: 500-2,000 tracks
  - Sectors per track: 32-128 sectors
  - Sectors size: 512 B-1024 KB (BLOCK)
  - Minimum seek time is often approximately 0.1 ms
  - Average seek time is often approximately 5-10 ms
  - Access time is often approximately 9-10 ms
  - Transfer rate is often 50-200 MB/s ++

Average Access Example



- Consider the Seagate 36.4 GB Ultra2 SCSI
  - Rotation speed: 10,000 RPM
  - Sector size: 512 B
  - Average seek time: 5.7 ms
  - Transfer rate: 24.5 MB/s
  - Controller overhead of 1 ms
- What is the average read time?

start end
* 160 2357

$$Average\ rotational\ latency = \frac{0.5\ rotation}{10000\ RPM} = \frac{0.5\ rotation}{10000\ RPM\ /\ (60\ sec/\ min)} = 3ms$$

$$Average\ transfer\ time = \frac{0.5\ KB}{24.5\ MB/\ s} = 0.02\ ms$$

$$Average\ access\ time = seek + rotational + transfer + overhead$$
$$= 5.7\ ms + 3\ ms + 0.02\ ms + 1\ ms = 9.72\ ms$$

(+ Bus?)

25%

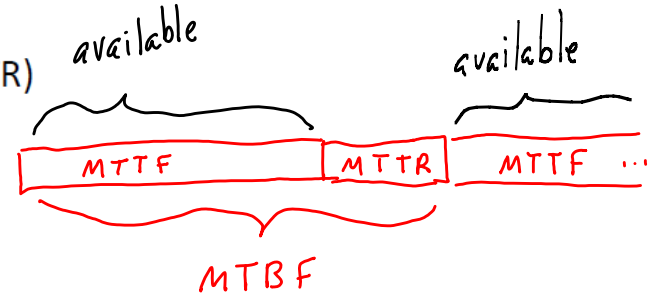
$$Expected\ seek\ time = 0.25 \times 5.7\ ms = 1.43\ ms$$
$$Expected\ access\ time = seek + rotational + transfer + overhead$$
$$= 1.43\ ms + 3\ ms + 0.02\ ms + 1\ ms = 5.45\ ms$$

- Note that the effects of the rotational delay are even more pronounced

Dependability Measures

- Reliability: mean time to failure (MTTF)
- Service interruption: mean time to repair (MTTR)
- Mean time between failures
  - MTBF = MTTF + MTTR
- Availability = MTTF / (MTTF + MTTR)

MTBF



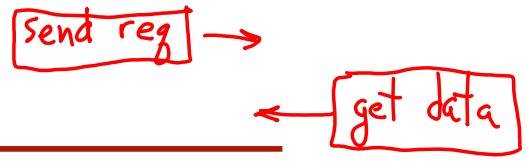
- Improving availability
  - Increase MTTF: fault avoidance, fault tolerance, fault forecasting
  - Reduce MTTR: improved tools and processes for diagnosis and repair

- How would you improve the availability of a storage system?
  - Assume multiple disks...

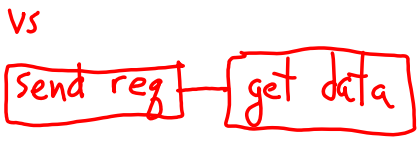
I/O System Design Example: Transaction Processing

- Examples: Airline reservation, bank ATM, inventory system, e-business
  - Many small changes to shared data space
    - Each transaction: 2-10 disk I/Os, ~2M-5M CPU instructions per disk I/O
  - Demands placed on system by many different users

- Important Considerations
  - Both throughput and response times are important
    - High throughput needed to keep cost low (transactions/sec)
    - Low response time is also very important for the users
  - Terrible locality
  - Requires graceful handling of failures
    - Redundant storage & multiphase operations



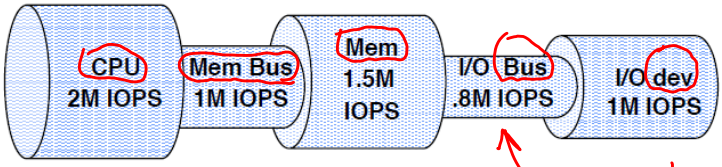
I/O Performance Factors



- Overall performance is dependent upon a many factors
  - CPU
    - How fast can the processor operate on the data?
  - Memory system bandwidth and latency
    - Multilevel caches
    - Main memory
  - System interconnection
    - I/O and memory buses
    - I/O controllers
  - I/O devices (disks)
  - Software efficiency
    - I/O device handler instruction path length, OS overhead, etc

I/O System Design

- Satisfying latency requirements
  - For time-critical operations
  - If system is unloaded
    - Add up latency of components
- Maximizing throughput at steady state (loaded system)
  - Find "weakest link" (lowest-bandwidth component)
  - Configure to operate at its maximum bandwidth
  - Balance remaining components in the system



Bottleneck

$\left(\frac{2\ \text{disk ops}}{T}\right)\left(\frac{128\ B}{\text{disk op}}\right)$ $t_{op} = t_{ms} + 2_{ms} + (128B/4MB/s)$ $\left(\frac{400\ Mi/s}{CPU}\right)\left(\frac{T}{3.2\ Mi}\right) = \frac{125T/s}{CPU}$ $1/125$ sec

- Analyze a multiprocessor to be used for transaction processing:
  - Transaction = two 128-byte disk accesses + 3.2 M instructions
  - Database file must be TPS x 10^9 bytes
    - Where TPS is the transactions/second achieved
  - HW cost: system $4,000 + $3,000 per CPU
  - CPU performance: 400 million instructions per second
    - Each processor can be connected with any number of disks
  - Disk controller delay = 2 msec
  - Can choose between two disk types, but can't mix them

$\left(\frac{3.2\ Mi}{T}\right)\left(\frac{2\ disk}{T}\right)\left(\frac{128\ B}{disk}\right)$

file size depends on Transaction rate, a portion of Total Job rate

Disk size	Cost	Capacity	Avg seek time	Rotation speed	Transfer rate
3.5 inch	$200	50 GB	8 msec	5400 RPM	4 MB/s
2.4 inch	$120	25 GB	12 msec	7200 RPM	2 MB/s

- What is the highest TPS you can process for $40,000, and with what configuration?

Solution

Part 1: pick a disk

- First calculate how many TPS each disk can sustain
- access time = seek time + rotational delay + transfer + controller
  - 3.5" disk time = 8 + 1/2(1/5400 RPM) + 128B / 4MB/s + 2 = 15.6 msec
  - 2.4" disk time = 12 + 1/2(1/7200 RPM) + 128B / 2MB/s + 2 = 18.2 msec
- Need 2 accesses per transaction, so TPS = 1/(2*time)
  - 3.5" TPS = 1/(2*15.6 msec) = 32.0 TPS
  - 2.4" TPS = 1/(2*18.2 msec) = 27.4 TPS

(1 Trans)/(2 access)

- But the database size on each disk = TPS x 10^9 bytes
  - 3.5" size = 50 GB > max 32 TPS (fits!) (I/O limited)
  - 2.4" size = 25 GB = max 25 TPS (doesn't fit!) (capacity limited)
    Must reduce TPS to 25 so that file fits
- Which has better cost/performance?
  - $/TPS for 3.5" = $200/32TPS = 6.25 $/TPS
  - $/TPS for 2.4" = $120/25TPS = 4.8 $/TPS
- Pick the 2.4" disk

Part 2: pick a CPU configuration

- TPS limit for each CPU =
  400 MIPS / (3.2 M instructions/transaction) = 125 TPS

[(1 Trans)/3.2 M instructions][400 M / Sec]

- To fully utilize the CPU TPS, the number of disks that each can accommodate is
  #disks/CPU = (125 TPS/CPU) / (25 TPS/disk) = 5

disks per CPU

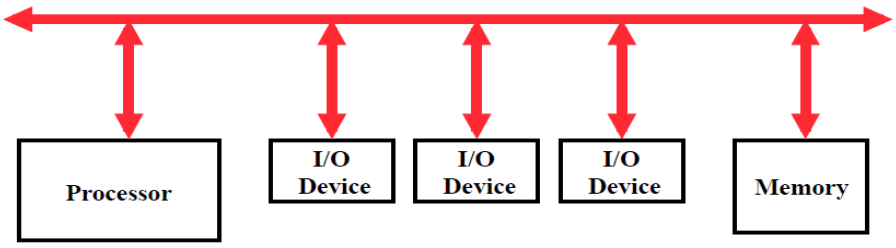
- So a system with n CPUs and $5n$ disks costing $40,000 means
  $4000 + 3000n$ + $120*5$n$ = $40000
  or n = 10

← Budget, solve for n

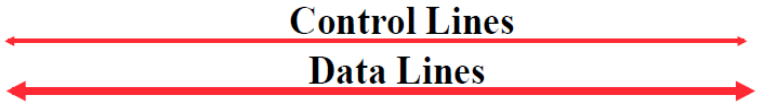
- The system has 10 CPUs, 50 2.4" disks, a total account file size of (50 x 25GB) = 1250 GB, and can process (50 x 25) = 1250 TPS.

Buses

- A bus is a shared communication link that connects multiple devices
- Single set of wires connects multiple "subsystems" as opposed to a point to point link which only connects two components together
- Wires connect in parallel, so 32 bit bus has 32 wires of data



- Advantages
  - Broadcast capability of shared communication link
  - Versatility
    - New device can be added easily
    - Peripherals can be moved between computer systems that use the same bus standard
  - Low Cost
    - A single set of wires is shared multiple ways
- Disadvantages
  - Communication bottleneck
    - Bandwidth of bus can limit the maximum I/O throughput
  - Limited maximum bus speed
    - Length of the bus
    - Number of devices on the bus
    - Need to support a range of devices with varying latencies and transfer rates



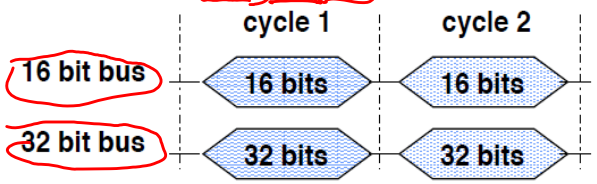
- Bus Components
  - Control Lines
    - Signal begin and end of transactions
    - Indicate the type of information on the data line
  - Data Lines
    - Carry information between source and destination
    - Can include data, addresses, or complex commands

- Processor-Memory Bus (or *front-side* bus or *system* bus)
  - Short, high-speed bus
  - Connects memory and processor directly
  - Designed to match the memory system and achieve the maximum memory-to-processor bandwidth (cache transfers)
  - Designed specifically for a given processor/memory system (proprietary)

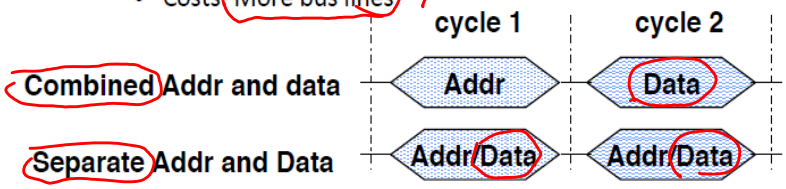
- I/O Bus (or peripheral bus)
  - Usually long and slow
  - Connect devices to the processor-memory bus
  - Must match a wide range of I/O device performance characteristics
  - Industry standard

- Synchronous Bus
  - Includes a clock in control lines
  - Fixed protocol for communication relative to the clock
  - Advantages
    - Involves very little logic and can therefore run very fast
  - Disadvantages
    - Every decision on the bus must run at the same clock rate
    - To avoid clock skew, bus cannot be long if it is fast
  - Example: Processor-Memory Bus
- Asynchronous Bus
  - No clock control line
  - Can easily accommodate a wide range of devices
  - No clock skew problems, so bus can be quite long
  - Requires handshaking protocol

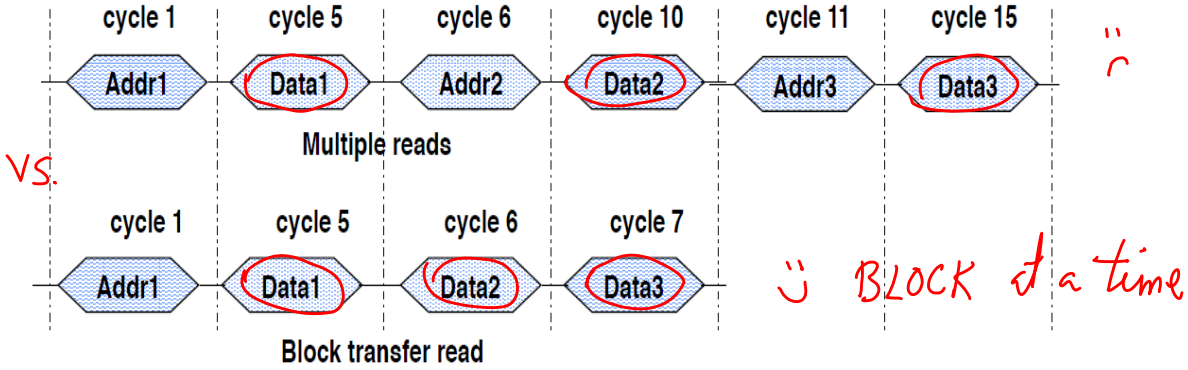
- Several factors account for bus bandwidth
  - Wider bus width
    - Increasing data bus width => more data per bus cycle
    - Cost: More bus lines



  - Separate address and data lines
    - Address and data can be transmitted in one bus cycle if separate address and data lines are available
    - Costs: More bus lines



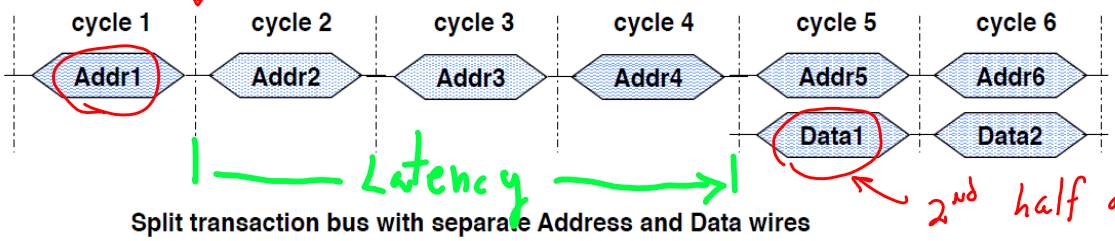
  - Block transfers
    - Transfer multiple words in back-to-back bus cycles
    - Only one address needs to be sent at the start
    - Bus is not released until the last word is transferred
    - Costs: Increased complexity and increased response time for pending requests



BLOCK at a time

  - Split transaction "pipelining the bus"
    - Free the bus during time between request and data transfer
    - Costs: Increased complexity and higher potential latency

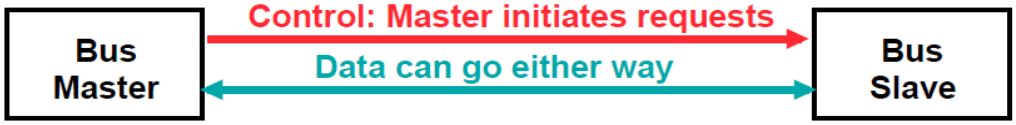
no waiting, next Transaction starts immediately



2nd half of Transaction

Split transaction bus with separate Address and Data wires

Accessing the Bus

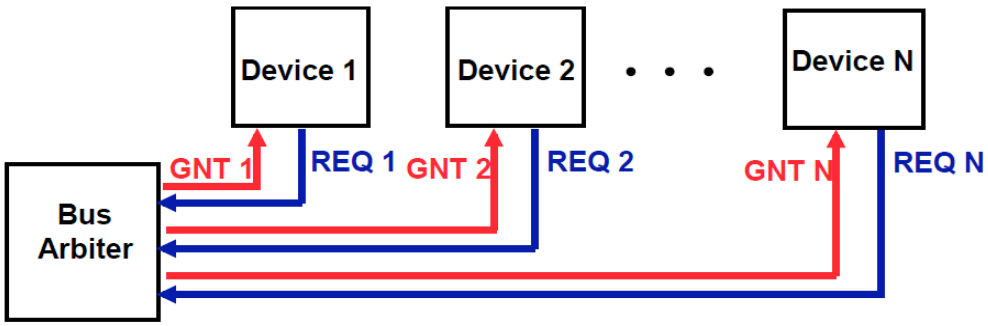


- How is the bus reserved by a device that wishes to use it?
- Master-slave arrangement
  - Only the bus master can control access to the bus
  - The bus master initiates and controls all bus requests
  - A slave responds to read and write requests
- A simple system
  - Processor is the only bus master
  - All bus requests must be controlled by the processor
  - Major drawback is the processor must be involved in every transfer

Multiple Masters

- With multiple masters, arbitration must be used so that only one device is granted access to the bus at a given time
- Arbitration
  - The bus master wanting to use the bus asserts a bus request
  - The bus master cannot use the bus until the request is granted : wait for permission "grant"
  - The bus master must signal the arbiter when finished using the bus
- Bus arbitration goals
  - Bus priority – Highest priority device should be serviced first
  - Fairness – Lowest priority devices should not starve

Centralized Parallel Arbitration

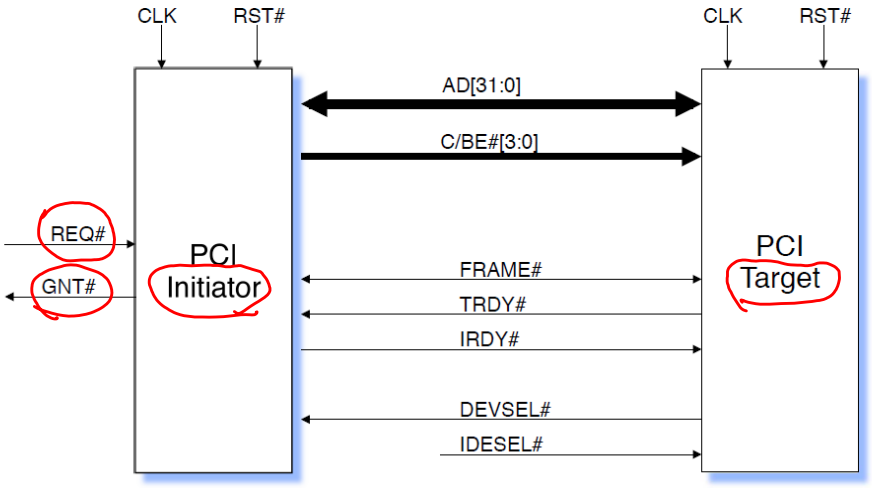


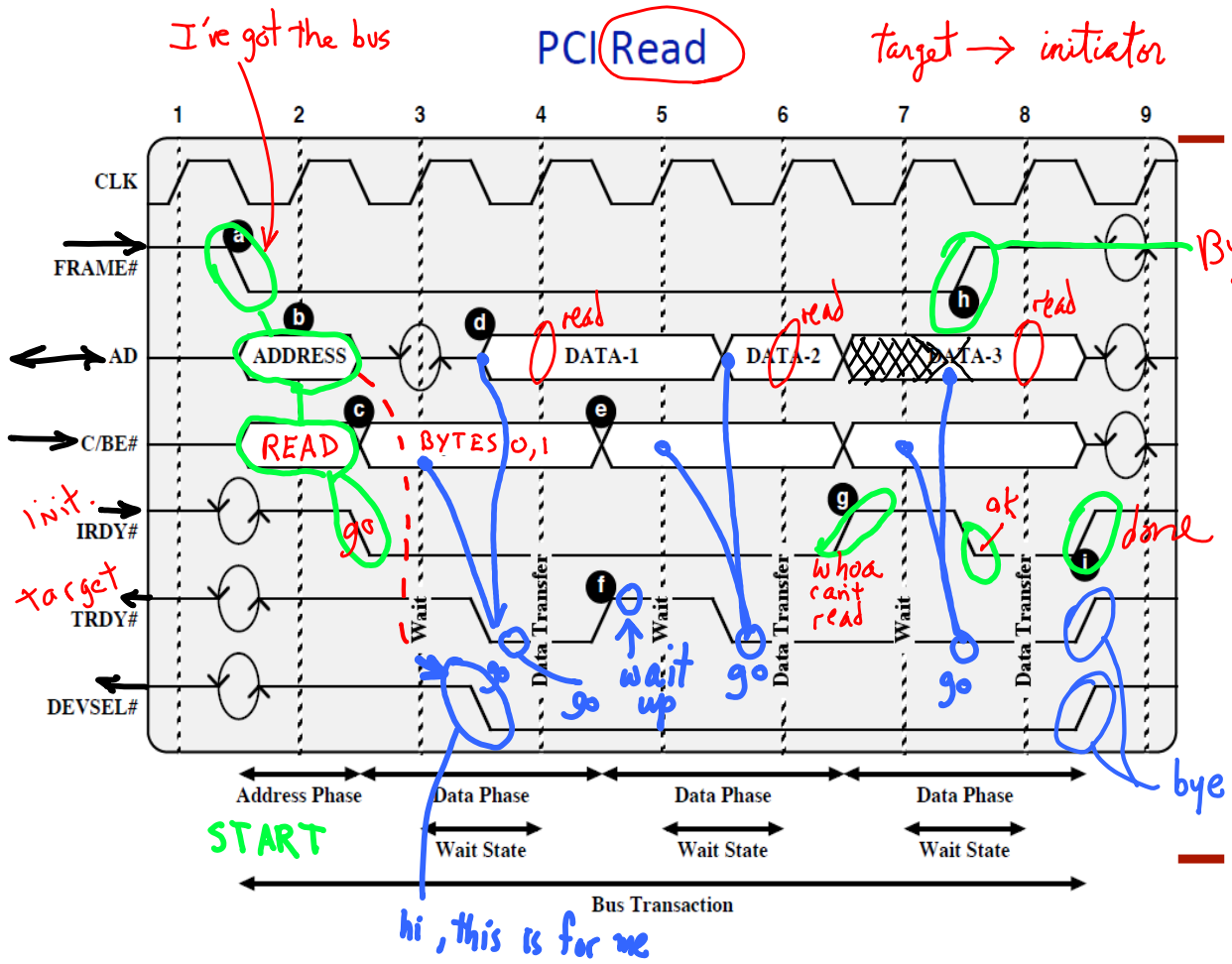
- Advantages
  - Centralized control where all devices submit request
  - Any fair priority scheme can be implemented (FCFS, round-robin)
- Disadvantages
  - Potential bottleneck at central arbiter

Case Study: PCI

- Peripheral Component Interconnect (PCI) peripheral backplane bus standard
- Clock Rate: 33 MHz (or 66 MHz in PCI Version 2.1) [CLK]
- Central arbitration (REQ#, GNT#)
  - Overlapped with previous transaction
- Multiplexed Address/Data
  - 32 lines (with extension to 64) [AD]
- General Protocol
  - Transaction type (bus command is memory read, memory write, memory read line, etc) [C/BE#]
  - Address handshake and duration [FRAME#, TRDY#]
  - Data width (byte enable) [C/BE#]
  - Variable length data block handshake between Initiatory Ready and Target Ready [IRDY#, TRDY#]
- Maximum bandwidth is 132 MB/s (533 MB/s at 64 bit/ 66 MHz)

32 bit PCI Signals





PCI Read Steps 1

PCI Read Steps 2

a) Once a bus master has gained control of the bus, it initiates the transaction by asserting FRAME. This line remains asserted until the last data phase. The initiator also puts the start address on the address bus, and the read command on the C/BE lines.

b) The target device recognizes its address on the AD lines.

c) The initiator ceases driving the AD bus. A turnaround cycle (marked with two circular arrows) is required before another device may drive any multiple-source bus. Meanwhile, the initiator changes the C/BE lines to designate which AD lines are to be used for data transfer (from 1-4 bytes wide). The initiator also asserts IRDY to indicate that it is ready for the first data item.

d) The selected target asserts DEVSEL to indicate that it has recognized its address and will respond. It places the requested data on the AD lines and asserts TRDY to indicate that valid data is present on the bus.

e) The initiator reads the data at the beginning of clock 4 and changes the byte enable lines as needed in preparation for the next read.

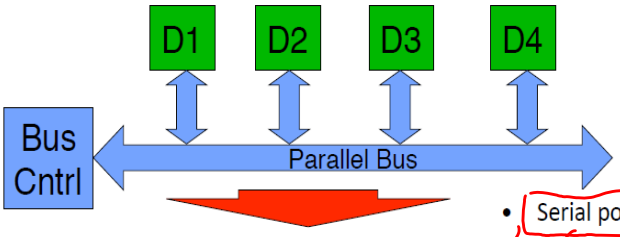
f) In this example, the target needs some time to prepare the second block of data for transmission. Therefore, it deasserts TRDY to signal the initiator that there will not be new data during the coming cycle. Accordingly, the initiator does not read the data lines at the beginning of cycle 5 and does not change the byte enable on that cycle. The block of data is read at the beginning of cycle 6.

g) During clock 6, the target places the third data item on the bus. However, in this example the initiator is not yet ready to read the data item (i.e. temporarily buffers are full). It therefore deasserts IRDY. This will cause the target to hold the data for an extra cycle.

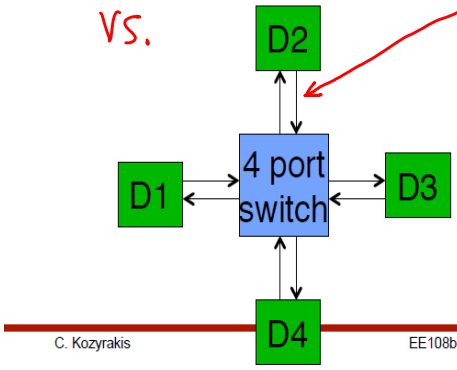
h) The initiator deasserts FRAME to signal the target that the third data transfer is the last, and asserts IRDY to signal that it is ready.

i) Return to the idle state. The initiator deasserts IRDY, and the target deasserts TRDY & DEVSEL.

Trends for Buses
Logical Bus and Physical Switch



VS.



- Serial point-to-point advantages
  - Faster links
  - Fewer chip package pins
  - Higher performance
  - Switch keeps arbitration on chip
- Many bus standards are moving to serial, point to point
- 3GIO, PCI-Express(PCI)
- Serial ATA (IDE hard disk)
- AMD Hypertransport versus Intel Front Side Bus (FSB)

PCI vs. PCI Express

- Same bus protocol
  - Same driver software
- PCI
  - 32-64 shared wires
  - Frequency: 33MHz – 133 MHz
  - Bandwidth: 132 MB/s – 1 GB/s

VS.

- PCI Express
  - 4 wires per direction
  - Frequency: 625 MHz

VS.

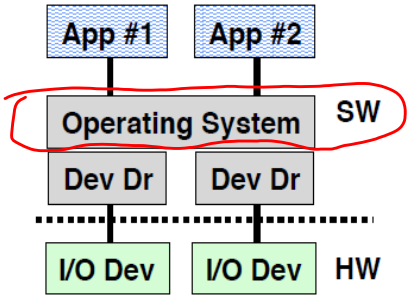
  - Bandwidth: 300 MB/s per direction
- PCI Express Advantage
  - 5-10 x pin bandwidth
  - Multiple links for more bandwidth

4 wires

serial

Operating System Tasks

- The OS is a resource manager and acts as the interface between I/O hardware and programs that request I/O



- Several important characteristics of the I/O system:
  - I/O system is *shared* by multiple programs
  - I/O systems often use *interrupts* to notify CPU
    - Interrupts = externally generated "exceptions"
    - Typically "Input available" or "Output complete" messages
    - OS handles interrupts by transferring control to kernel mode
  - Low-level control of an I/O device is complex
    - Managing a set of concurrent events
    - Requirements for correct device control are very detailed
    - I/O device drivers are the most common area for bugs in an OS!

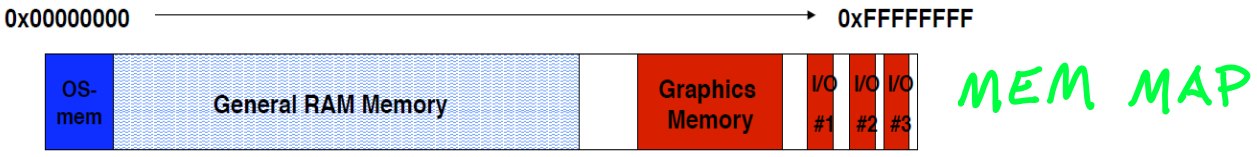
OS Communication

- The operating system should prevent user programs from communicating with I/O device directly
  - Must protect I/O resources to keep sharing fair
  - Protection of shared I/O resources cannot be provided if user programs could perform I/O directly

- Three types of communication are required:
  1. OS must be able to give commands to I/O devices
  2. I/O device must be able to notify OS when I/O device has completed an operation or has encountered an error
  3. Data must be transferred between memory and an I/O device

---

- Memory-mapped I/O:
  - Portions of the address space are assigned to each I/O device
  - I/O addresses correspond to device registers
  - User programs prevented form issuing I/O operations directly since I/O address space is *protected* by the address translation mechanism



MEM MAP

- I/O devices are managed by I/O controller hardware
  - Transfers data to/from device
  - Synchronizes operations with software via

- Command registers
  - Cause device to do something

- Status registers
  - Indicate what the device is doing and occurrence of errors

- Data registers
  - Write: transfer data to a device
  - Read: transfer data from a device

Data Transfer

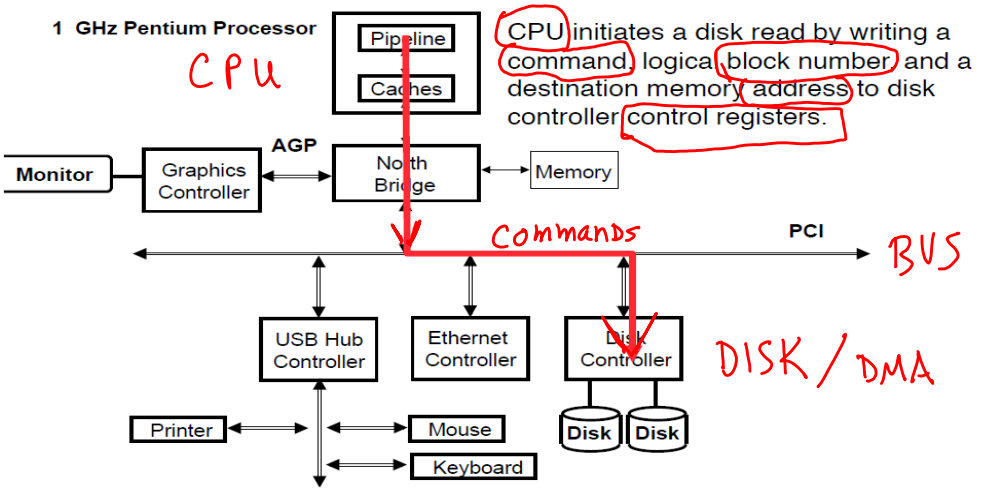
- The third component to I/O communication is the transfer of data from the I/O device to memory (or vice versa)
- Simple approach: "Programmed" I/O
  - Software on the processor moves *all* data between memory addresses and I/O addresses
  - Simple and flexible, but wastes CPU time
  - Also, lots of excess data movement in modern systems
    - Ex. Mem --> NB --> CPU --> NB --> graphics
    - When we want: Mem --> NB --> graphics
- So need a solution to allow data transfer to happen *without* the processor's involvement

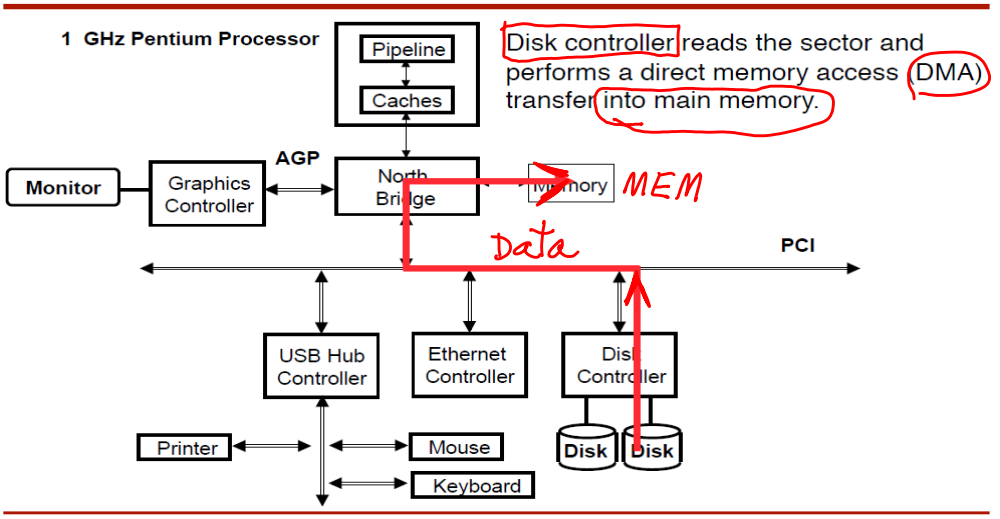
---

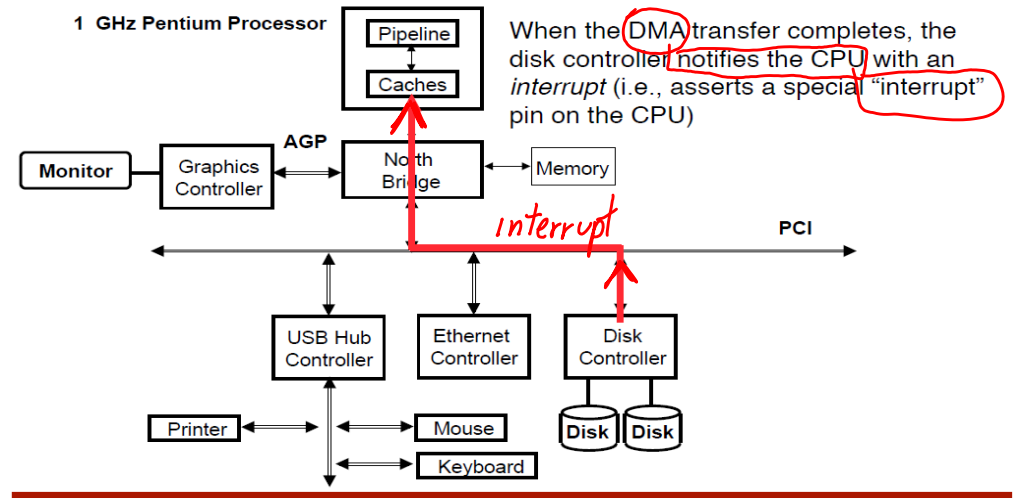
- Direct Memory Access (DMA)
  - Transfer *blocks* of data to or from memory without CPU intervention
  - Communication coordinated by the *DMA controller*
    - DMA controllers are integrated in memory or I/O controller chips
  - DMA controller acts as a bus master in bus-based systems

- DMA Steps
  - Processor sets up DMA by supplying:
    - Identity of the device and the operation (read/write)
    - The memory address for source/destination
    - The number of bytes to transfer
  - DMA controller starts the operation by arbitrating for the bus and then starting the transfer when the data is ready
  - Notify the processor when the DMA transfer is complete or on error
    - Usually using an interrupt

Reading a Disk Sector (1)







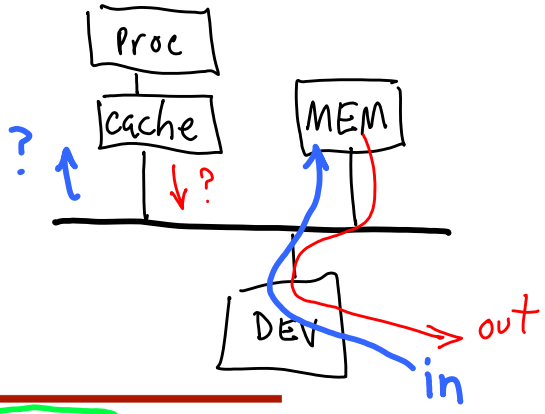
DMA Problems: Virtual Vs. Physical Addresses

- If DMA uses physical addresses
  - Memory access across physical page boundaries may not correspond to contiguous virtual pages (or even the same application!)
- Solution 1: 1 page per DMA transfer
- Solution 1+: chain a series of 1-page requests provided by the OS
  - Single interrupt at the end of the last DMA request in the chain
- Solution 2: DMA engine uses virtual addresses
  - Multi-page DMA requests are now easy
  - A TLB is necessary for the DMA engine initialized by CPU

+

- For DMA with physical addresses, pages must be pinned in DRAM
  - OS should not page to disks pages involved with pending I/O

- A copy of the data involved in a DMA transfer may reside in processor cache
  - If memory is updated: must update or invalidate "old" cached copy
  - If memory is read: Must read latest value, which may be in the cache
    - Only a problem with write-back caches
- This is called the "cache coherence" problem
  - Same problem in multiprocessor systems

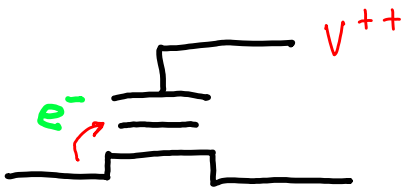


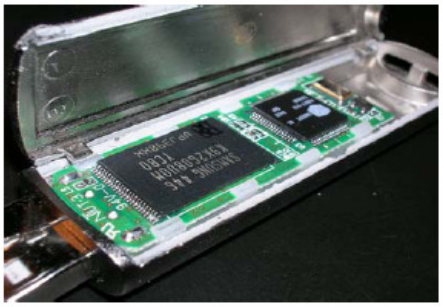
DMA & Coherence

- Solution 1: OS flushes the cache before I/O reads or forces writebacks before I/O writes
  - Flush/write-back may involve selective addresses or whole cache
  - Can be done in software or with hardware (ISA) support
- Solution 2: Route memory accesses for I/O through the cache
  - Search the cache for copies and invalidate or write-back as needed
  - This hardware solution may impact performance negatively
    - While searching cache for I/O requests, it is not available to processor
  - Multi-level, inclusive caches make this easier
    - Processor searches L1 cache mostly (until it misses)
    - I/O requests search L2 cache mostly (until it finds a copy of interest)

Flash Storage

- Nonvolatile semiconductor storage
  - Charge trapped in a floating gate
- General characteristics
  - 100× – 1000× faster than disk
  - Smaller, lower power, more robust
  - But more $/GB (between disk and DRAM)



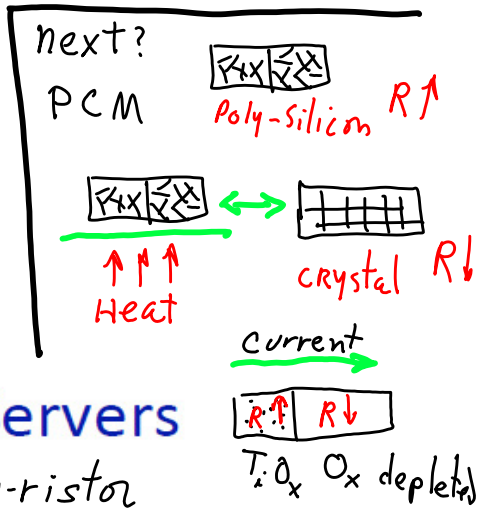


Flash Types

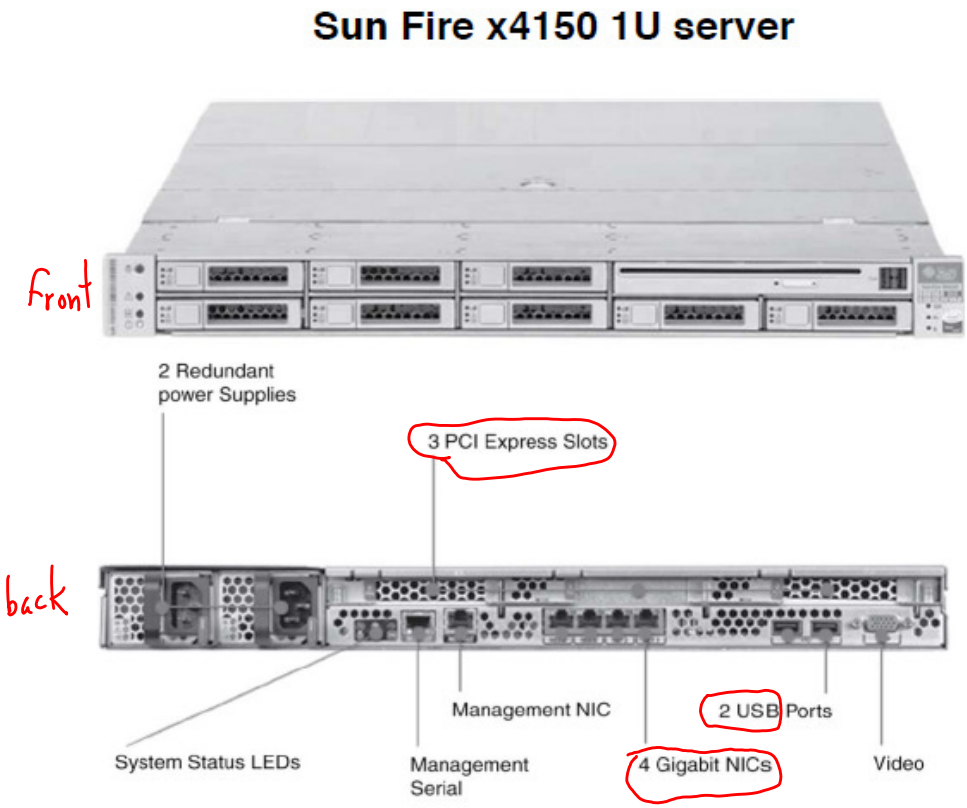
- NOR flash: bit cell like a NOR gate
  - Fast read (~100ns), slow writes (200usec), very slow erase (1sec)
  - 10K to 100K erase cycles
  - Used for instruction memory in embedded systems

- NAND flash: bit cell like a NAND gate
  - Denser (bits/area, ~40% of NOR), cheaper per GB
  - Slow read (50usec), slow writes (200usec), slow erase (2msec)
  - 100K to 1M erase cycles
  - Used for data storage (USB keys, media storage, …)

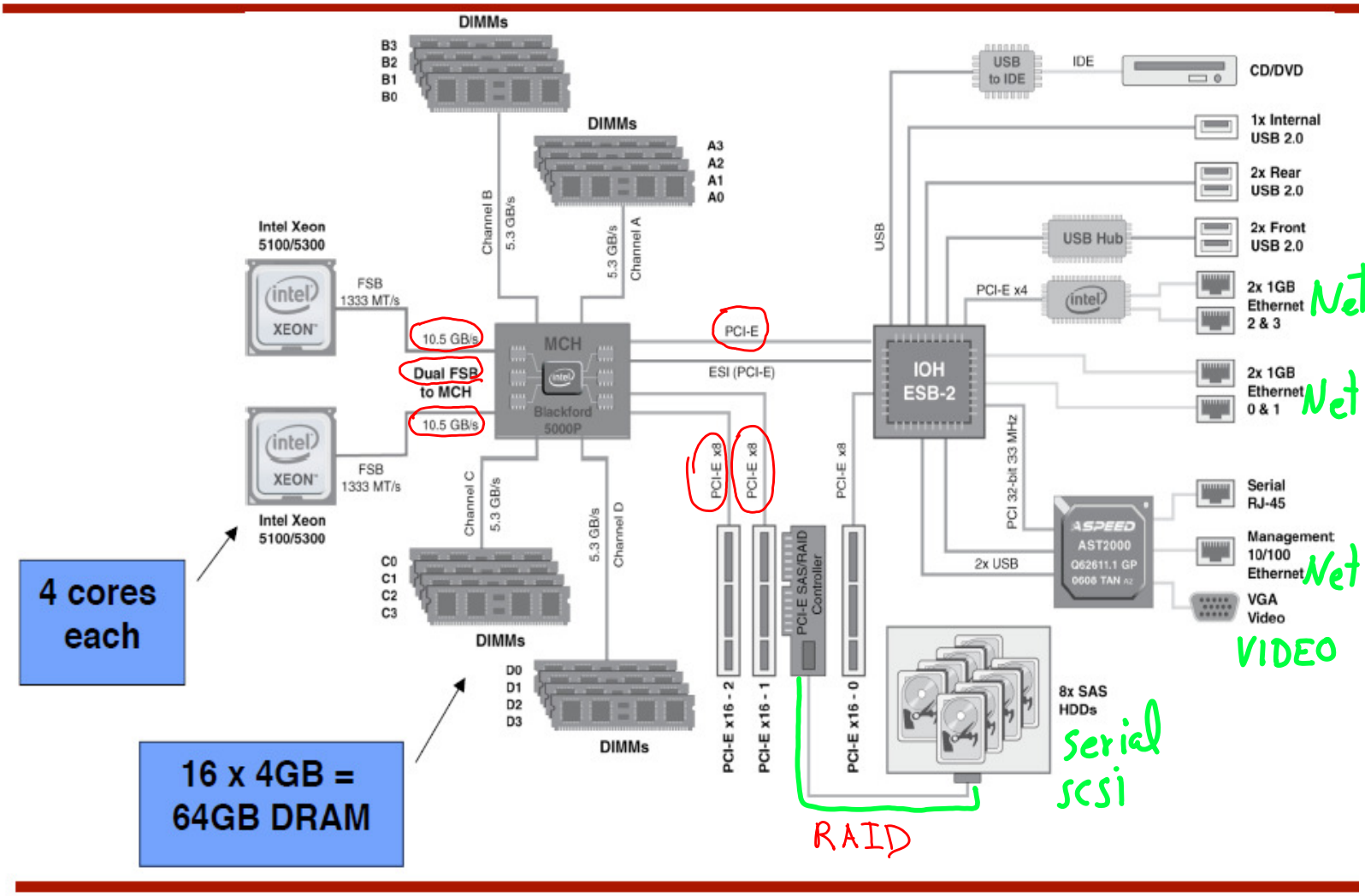
- Flash bits wears out after 1000's of accesses
  - Not suitable for direct RAM or disk replacement
  - Wear leveling: remap data to less used blocks



Another Example: Rack-Mounted Servers



Sun Fire x4150 1U server



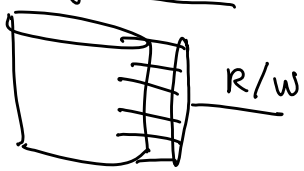
I/O System Design Example

- Given a Sun Fire x4150 system with
  - Workload: 64KB disk reads @
    - Each I/O op requires 200,000 user-code & 100,000 OS instructions
  - Each CPU: 10^9 instructions/sec
  - FSB: 10.6 GB/sec peak
  - DRAM DDR2 667MHz: 5.336 GB/sec
  - PCI-E 8× bus: 8 × 250MB/sec = 2GB/sec
  - Disks: 15,000 rpm, 2.9ms avg. seek time, 112MB/sec transfer rate

- What I/O rate can be sustained?
  - For random reads and for sequential reads ? ? size of reads?
  - I/O rate for CPUs
    - Per core: 10^9/(100,000 + 200,000) = 3,333
    - 8 cores: 26,667 ops/sec
  - Random reads, I/O rate for disks
    - Assume actual seek time is average/4
    - Time/op = seek + latency + transfer
      = 2.9ms/4 + 4ms/2 + 64KB/(112MB/s) = 3.3ms
    - 303 ops/sec per disk, 2424 ops/sec for 8 disks RAM
  - Sequential reads
    - 112MB/s / 64KB = 1750 ops/sec per disk
    - 14,000 ops/sec for 8 disks or SEQ
- PCI-E I/O rate
  - 2GB/sec / 64KB = 31,250 ops/sec
- DRAM I/O rate
  - 5.336 GB/sec / 64KB = 83,375 ops/sec
- FSB I/O rate
  - Assume we can sustain half the peak rate
  - 5.3 GB/sec / 64KB = 81,540 ops/sec per FSB
  - 163,080 ops/sec for 2 FSBs X
- Weakest link: disks
  - 2424 ops/sec random, 14,000 ops/sec sequential
  - Other components have ample headroom to accommodate these rates

Faster Disk I/O

Large disk



k platters

AREA = bits

1 R/W head active

small disks (cheaper)

n R/W heads

⇒ faster I/O

failures ⇑

$P(fail) = P(d_1 \text{ fails}) + P(d_2 \text{ fails}) + \cdots + P(d_n)$

cheaper disks, MTTF ⇓ P(fail) ⇑

failure Rate (n) > n failure Rate (1)

1 disk ($\frac{D\$}{disk}$)

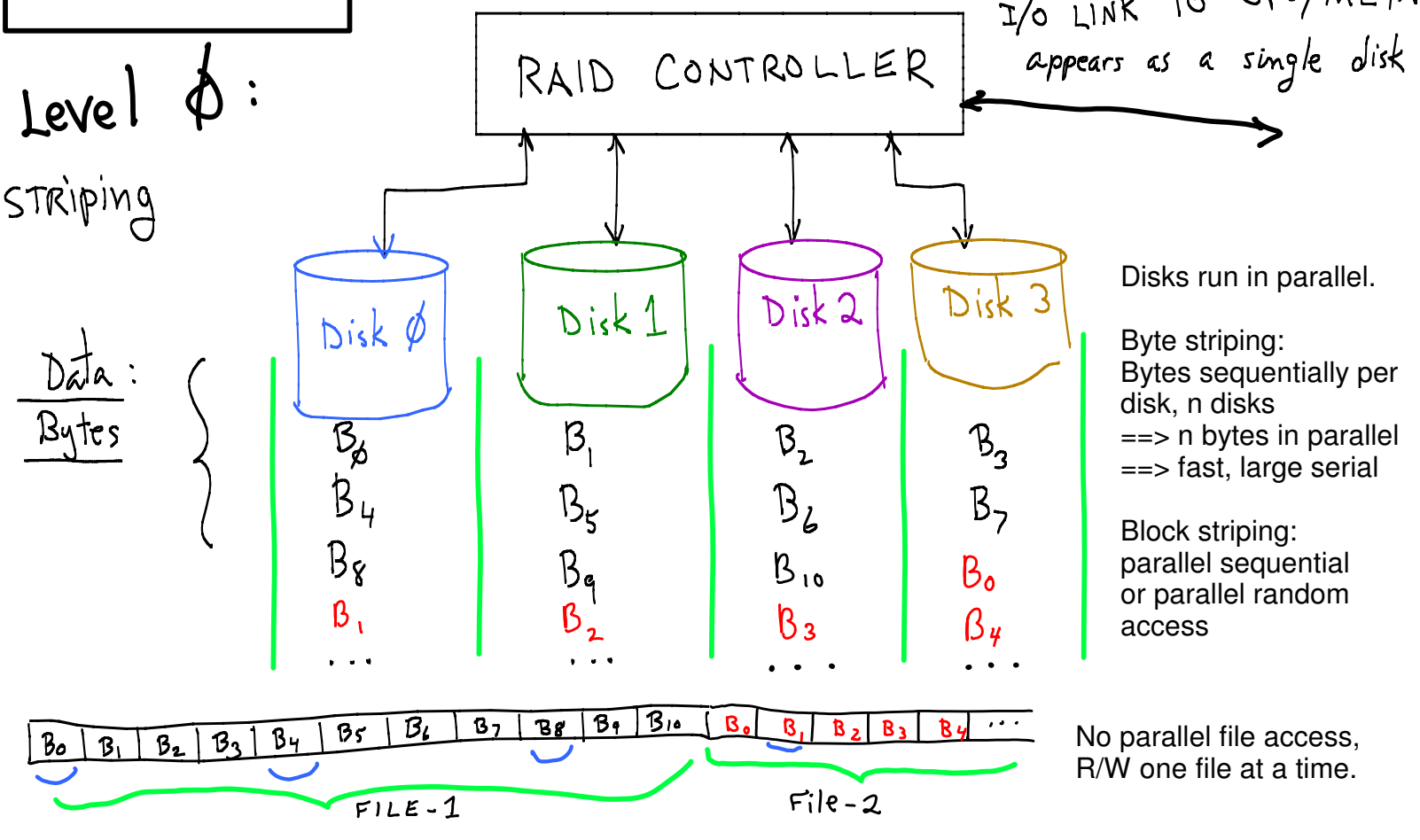
n disks ($\frac{d\$}{disk}$) ? < 1($\frac{D\$}{disk}$)

RAID

Redundant Arrays of Inexpensive Disks

Level 0:

STRIPING



Disks run in parallel.

Byte striping:
Bytes sequentially per disk, n disks
==> n bytes in parallel
==> fast, large serial

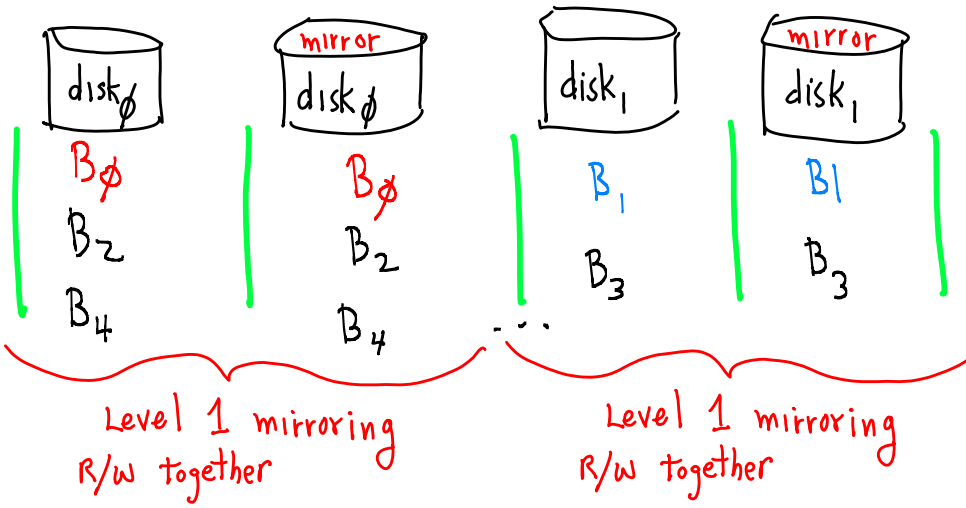
Block striping:
parallel sequential or parallel random access

No parallel file access, R/W one file at a time.

Level 1:

mirrored

1 W to both
or
2 separate R
for pair



Parallel duplicated disks.

2-disk striping, @ Level 1 mirrored

Automatic switch-over on failure

"Always up"

cost 2X

Level 2 =
+ EDEC CODES

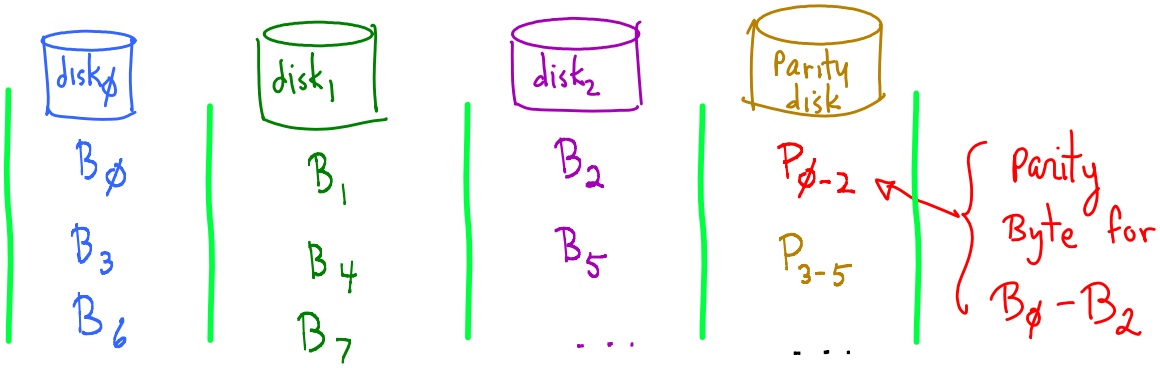
level $\emptyset$ striping

⇒ **Level 10**

level-1 k-mirrored, up to k failures

Level 3:

Parity by Byte



B_0 1101 ...
B_1 0110 ...
B_2 0011 ...

P_{0-2} 1000 ... ← parity by column

⇒ 1-bit error correction for failed disk

⇒ rebuild disk, if entire disk crash (any single disk)

Striping by byte, same as level 0: fast for large serial file access; 1 file R or W at a time.

Why parity works. Parity as XOR. Properties of XOR.

$(0 \oplus A) = A$

$(1 \oplus A) = \overline{A}$

$(A \oplus A') = 1$, if $A \neq A'$, 0 otherwise (detects changes)

$(A \oplus B) = P_{AB}$ (Def'n of Parity)

$A \oplus B \oplus C = P_{AB} \oplus C = P_{ABC}$ (Associativity of Parity)

$A \oplus B = P$

$(A \oplus B) \oplus B = P \oplus B = A \oplus (B \oplus B) = A \oplus \emptyset = A$ } $A = P \oplus B$

disk$_\emptyset$ crashes

$B_\emptyset \oplus B_1 \oplus B_2 = P_{\emptyset-2}$

$B_\emptyset = P_{\emptyset-2} \oplus (B_1 \oplus B_2)$

-- retreive data on the fly

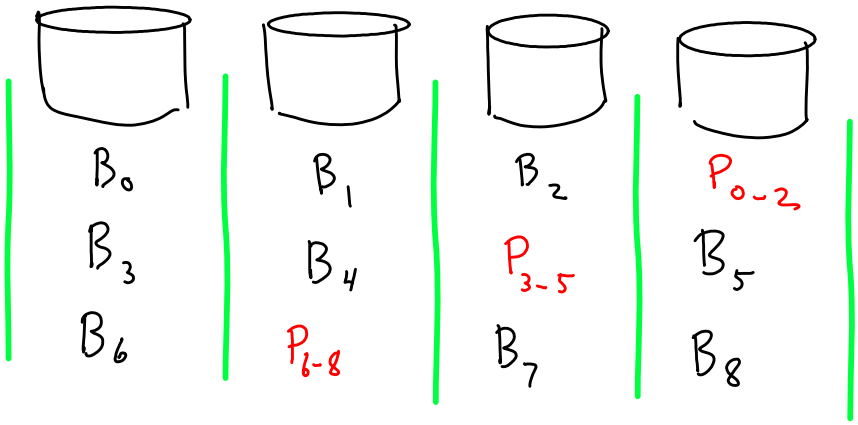
-- When disk_0 is replaced,
-- rebuild disk_0 on the fly

-- handle P disk in same way

Level 4: level 3 + data and Parity by blocks

Independent block accesses, bitwise XOR by blocks (same as by bytes). Multiple, asynchronous, file block reads. Collision on parity disk.

Level 5: level 4 + Parity blocks distributed to all disks



Parallel, independent random block access, multiple file access in parallel.

write (B_0')

Read B_0
Read P_{0-2}

$C = B_0 \oplus B_0'$

Bitwise, if $B_0 \neq B_0'$, $C = 1$

$P_{0-2}' = C \oplus P_{0-2}$ (flips, if $C=1$)

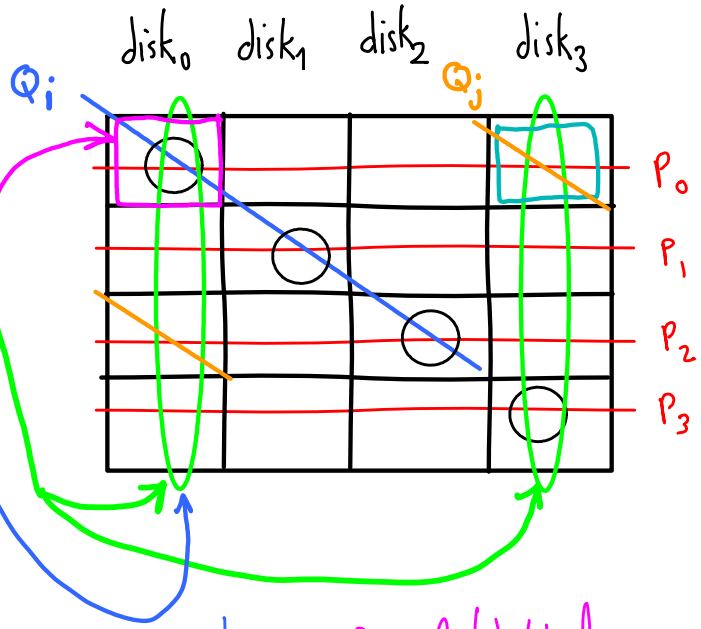
write B_0', P_0'

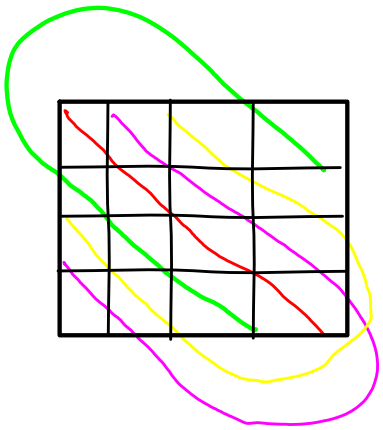
Level 6: level 5 + 2nd parity bit

EDEC: handle 2 disk failures.

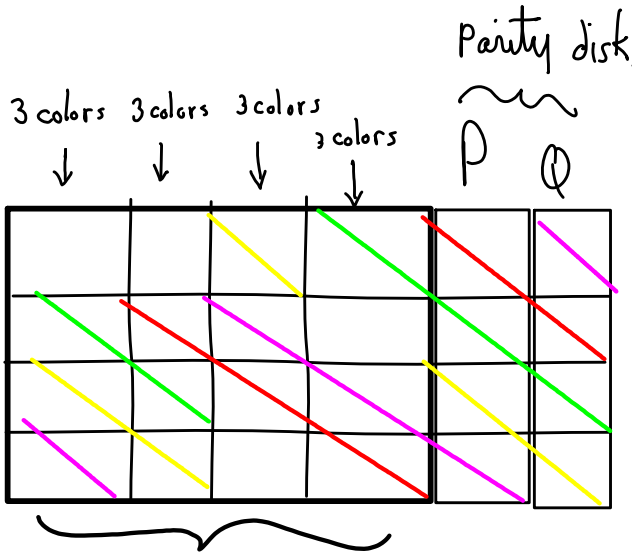
- Compute Parity along diagonals
  - 2 disks fail
- Some Q_i has only 1 fail
- fix block (use same method as above)
- Now row has only a 1-bit error, fix block
- Some other Q_j has 1-bit error → repair
- ...

(see IBM Raid DP)





4 colors ($4\ Q_i$ s)
only 3 colors per any col. Delete one square for each color so that No pair of cols. share 3 colors.



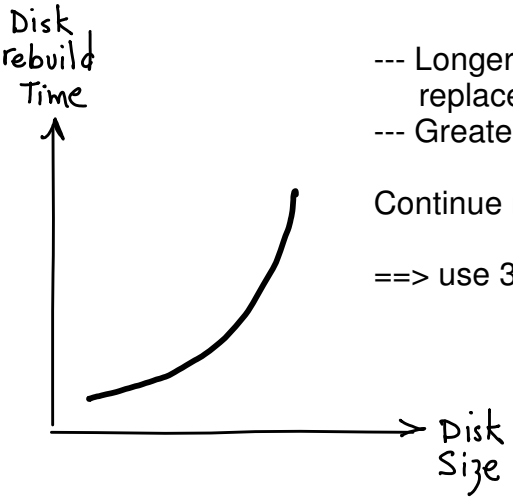
Pick any col., one color is missing.
no pair of cols have same 3 colors.

$\Rightarrow$

- Some color only hit once in both cols. hit.

OR

- Both P + Q hit $\Rightarrow$ rebuild P + Q



--- Longer delays in getting data onto replacement disk,
--- Greater risk of additional failure.

Continue running under error condition:

==> use 3-bit EDEC