Lecture 24

- Busses & Disks

Networks and Buses

- There are two main ingredients to I/O systems.
  - Devices like hard drives that we discussed last time.
  - Buses/Networks connect devices to each other and the processor.
    - Back of the envelope performance metrics
    - Bus organization and Performance
    - Serial vs. Parallel

Back of the Envelope Calculation

- When communicating over a network, typically your communication is broken into a collection of “packets”
  - Each packet carries ~1kB of data
  - Packets are reassembled into the original message at the destination.
- Because the transmission of network packets can be pipelined, the time for a transfer can be estimated as:

Transfer time = latency + transfer_size / bandwidth

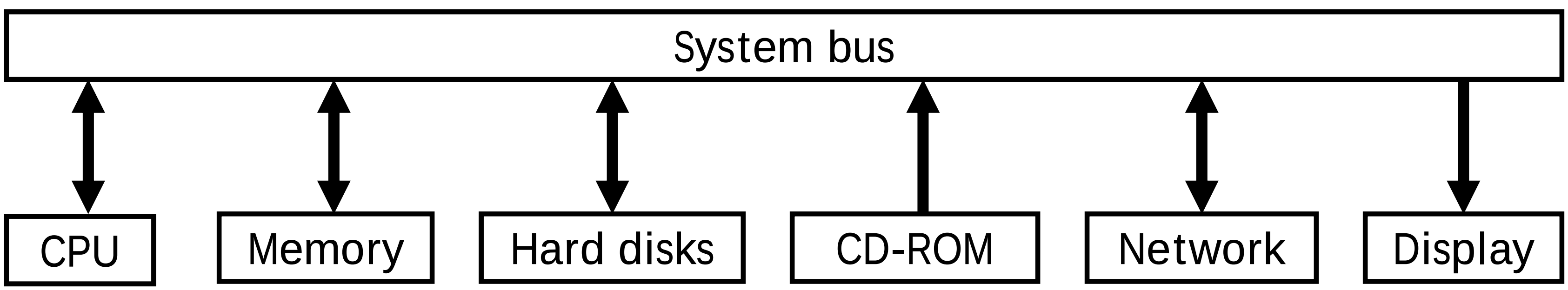
= time + bytes / (bytes/time)

Dominant term for small transfers (latency)

Dominant term for large transfers (transfer_size / bandwidth)

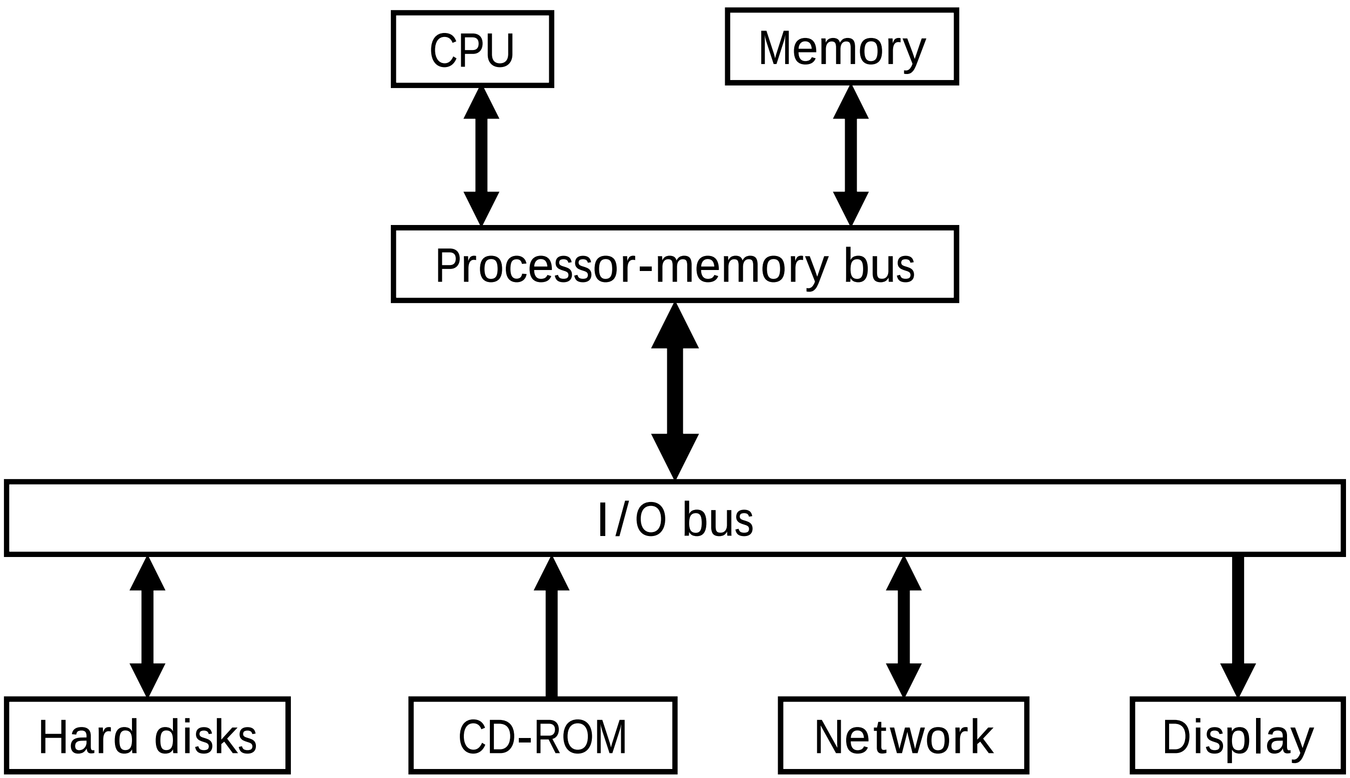
Computer buses

- Every computer has several small “networks” inside, called buses, to connect processors, memory, and I/O devices.
- The simplest kind of bus is linear, as shown below.
  - All devices share the same bus.
  - Only one device at a time may transfer data on the bus.
- Hmmm….do you like this?



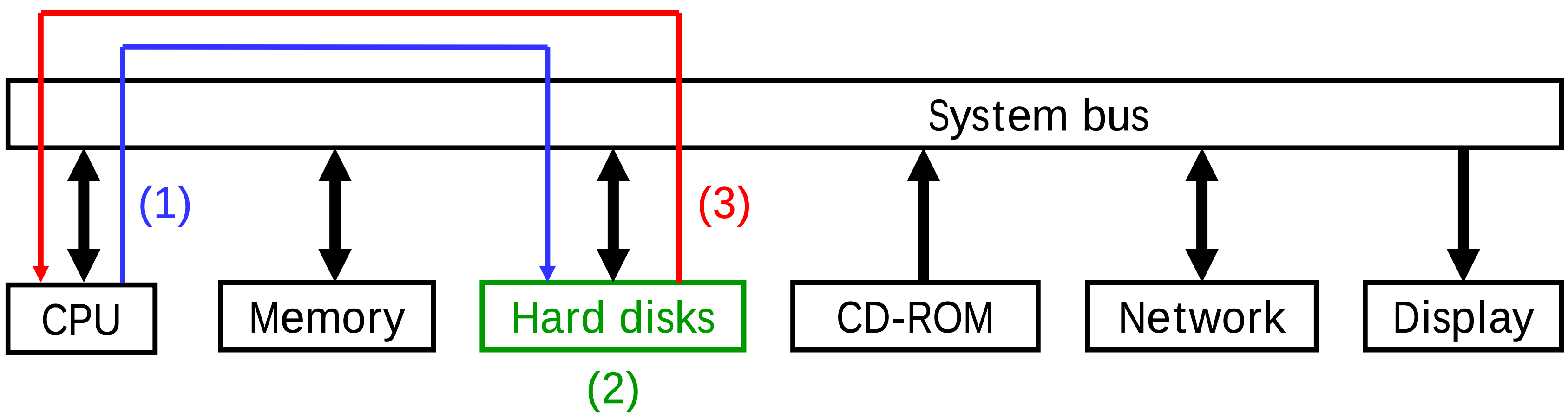
Hierarchical buses

- We could split the bus into different segments.
  - Since the CPU and memory need to communicate so often, a shorter and faster processor-memory bus can be dedicated to them.
  - A separate I/O bus would connect the slower devices to each other, and eventually to the processor.



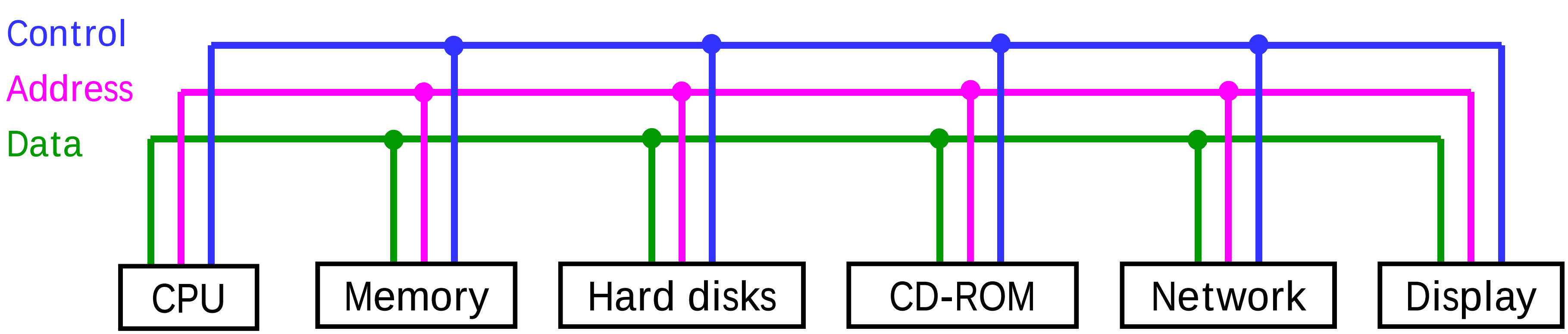
Basic bus protocols

- Although physically, our computer may have a hierarchy of buses (for performance), logically it behaves like a single bus
- Last time we discussed how I/O reads and writes can be programmed like loads and stores, using addresses.
- Two devices might interact as follows.
  1. An initiator sends an address and data over the bus to a target.
  2. The target processes the request by “reading” or “writing” data.
  3. The target sends a reply over the bus back to the initiator.
- The bus width limits the number of bits transferred per cycle.



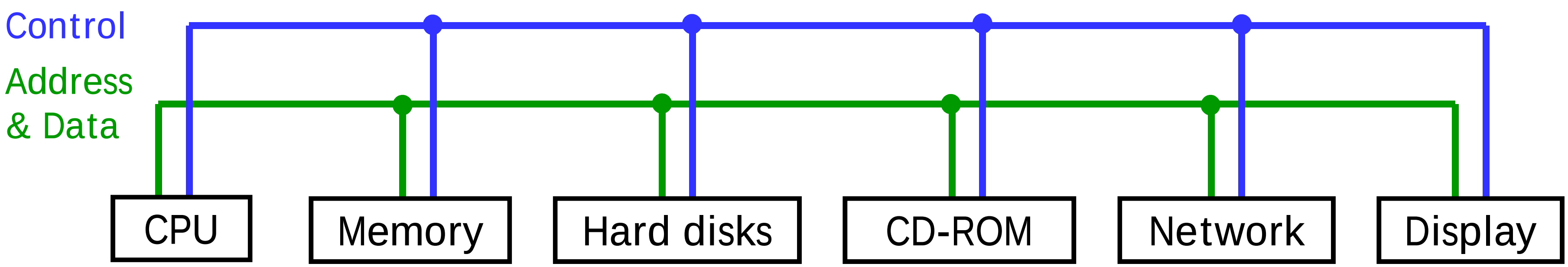
What is the bus anyway?

- A bus is just a bunch of wires which transmits three kinds of information.
  - Control signals specify commands like “read” or “write.”
  - The location on the device to read or write is the address.
  - Finally, there is also the actual data being transferred.
- Some buses include separate control, address and data lines, so all of this information can be sent in one clock cycle.



Multiplexed bus lines

- Unfortunately, this could lead to many wires and wires cost money.
  - Many buses transfer 32 to 64 bits of data at a time.
  - Addresses are usually at least 32-bits long.
- Another common approach is to multiplex some lines.
  - For example, we can use the same lines to send both the address and the data, one after the other.
  - The drawback is that now it takes *two* cycles to transmit both pieces of information.



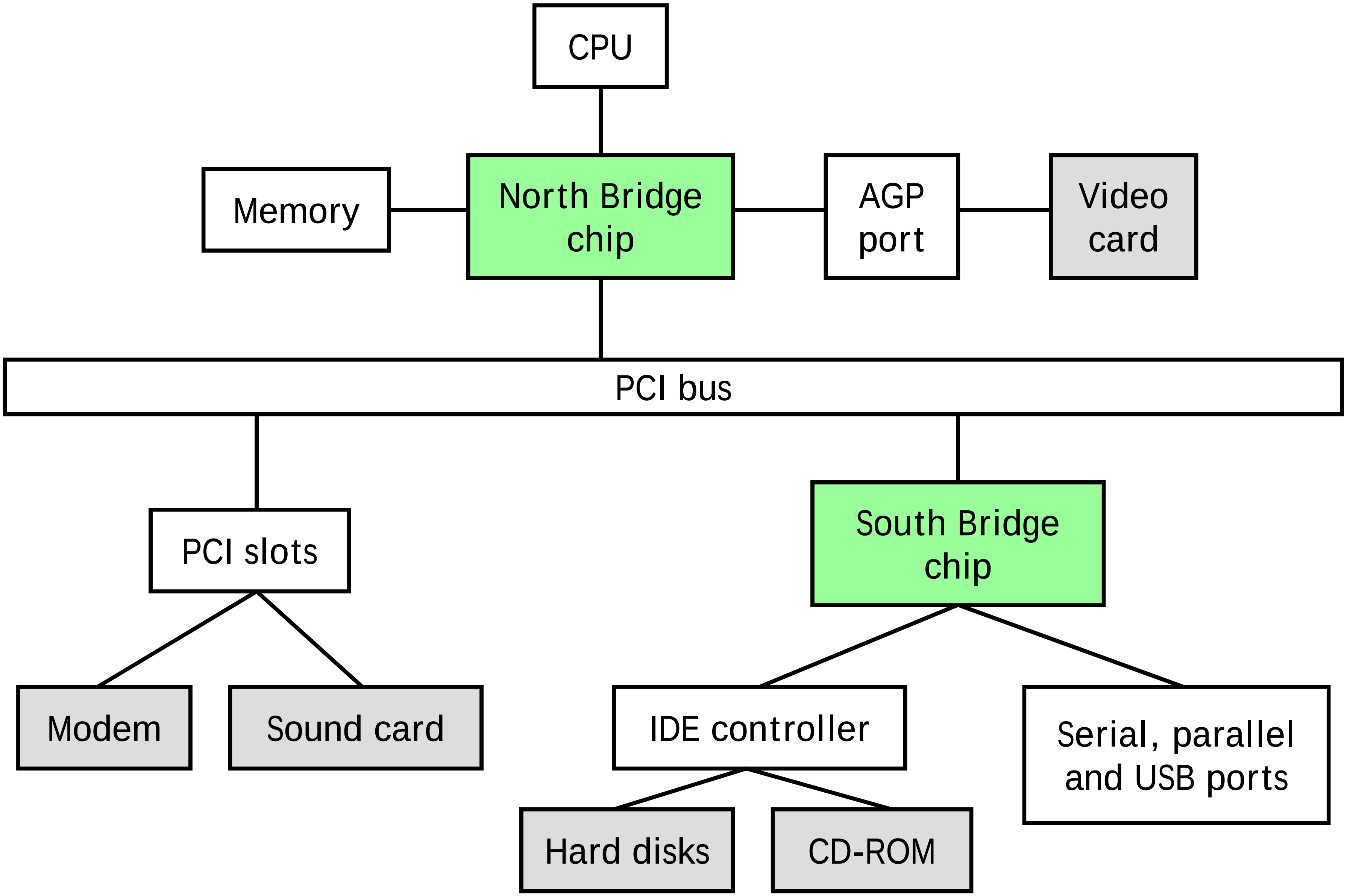
Example bus problems

- I/O problems always start with some assumptions about a system.
  - A CPU and memory share a 32-bit bus running at 100MHz.
  - The memory needs 50ns to access a 64-bit value from one address.
- Then, questions generally ask about the latency or throughput.
  - How long does it take to read one address of memory?
  - How many random addresses can be read per second?
- You need to find the total time for a single transaction.
  1. It takes one cycle to send a 32-bit address to the memory.
  2. The memory needs 50ns, or 5 cycles, to read a 64-bit value.
  3. It takes two cycles to send 64 bits over a 32-bit wide bus.
- Then you can calculate latencies and throughputs.
  - The time to read from one address is eight cycles or 80ns.
  - You can do 12.5 million reads per second, for an effective bandwidth of $(12.5 \times 10^6$ reads/second$) \times (8$ bytes/read$) = 100$MB/s.

Synchronous and asynchronous buses

- A synchronous bus operates with a central clock signal.
  - Bus transactions can be handled easily with finite state machines.
  - However, the clock rate and bus length are inversely proportional; faster clocks mean less time for data to travel. This is one reason why PCs never have more than about five expansion slots.
  - All devices on the bus must run at the same speed, even if they are capable of going faster.
- An asynchronous bus does not rely on clock signals.
  - Bus transactions rely on complicated handshaking protocols so each device can determine when other ones are available or ready.
  - On the other hand, the bus can be longer and individual devices can operate at different speeds.
  - Many external buses like USB and Firewire are asynchronous.

Buses in PCs (basic idea)



External buses

- External buses are provided to support the frequent plugging and un-plugging of devices
  - As a result their designs significantly differ from internal buses

- Two modern external buses, Universal Serial Bus (USB) and FireWire, have the following (desirable) characteristics:
  - Plug-and-play standards allow devices to be configured with software, instead of flipping switches or setting jumpers.
  - Hot plugging means that you don't have to turn off a machine to add or remove a peripheral.
  - The cable transmits power! No more power cables or extension cords.
  - Serial links are used, so the cable and connectors are small.

Serial/Parallel

- Why are modern external buses serial rather than parallel?

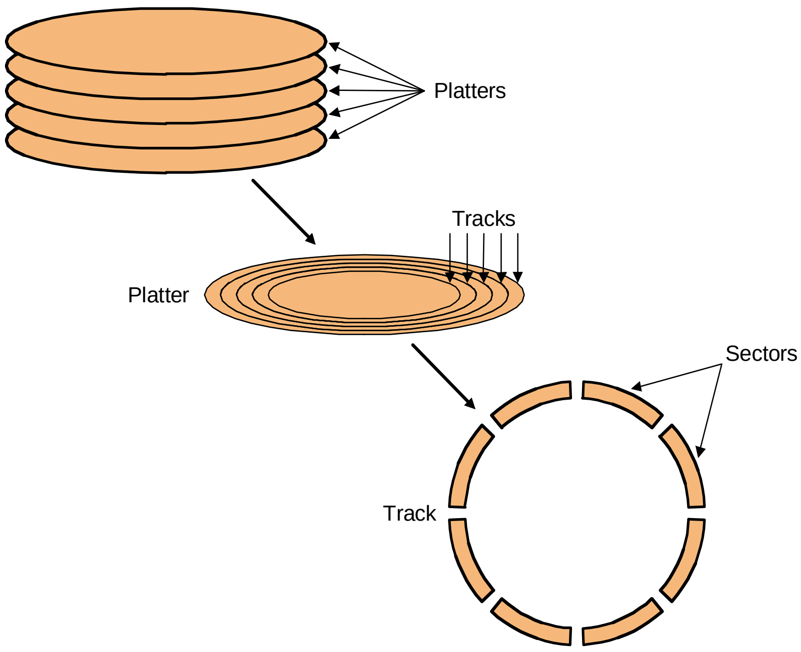
- Generally, one would think that having more wires would increase bandwidth and reduce latency, right?
  - Yes, but only if they can be clocked at comparable frequencies.

- Two physical issues allow serial links to be clocked significantly faster:
  - On parallel interconnects, interference between the signal wires becomes a serious issue.
  - Skew is also a problem; all of the bits in a parallel transfer could arrive at slightly different times.

- Serial links are being increasingly considered for internal buses:
  - Serial ATA is a new standard for hard drive interconnects
  - PCI-Express (aka 3GI/O) is a PCI bus replacement that uses serial links

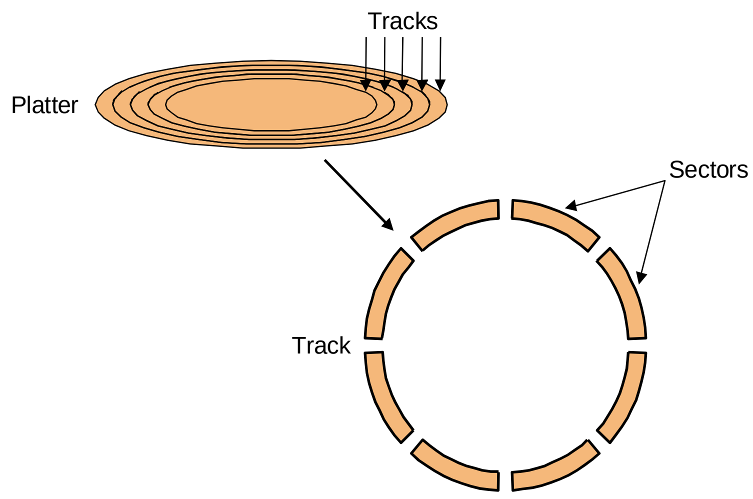
Hard drives

- The ugly guts of a hard disk.
  - Data is stored on double-sided magnetic disks called platters.
  - Each platter is arranged like a record, with many concentric tracks.
  - Tracks are further divided into individual sectors, which are the basic unit of data transfer (traditionally 512B, moving to 4K).
  - Each surface has a read/write head like the arm on a record player, but all the heads are connected and move together.



Accessing data on a hard disk

- Accessing a sector on a track on a hard disk takes a lot of time!
  - Seek time measures the delay for the disk head to reach the track.
  - A rotational delay accounts for the time to get to the right sector.
  - The transfer time is how long the actual data read or write takes.
  - There may be additional overhead for the operating system or the controller hardware on the hard disk drive.
- Rotational speed, measured in revolutions per minute or RPM, partially determines the rotational delay and transfer time.



Estimating disk latencies (seek time)

- Manufacturers often report *average* seek times of 8-10ms.
  - These times average the time to seek from any track to any other track.
- In practice, seek times are often much better.
  - For example, if the head is already on or near the desired track, then seek time is much smaller. In other words, locality is important!
  - Actual average seek times are often just 2-3ms.

Estimating Disk Latencies (rotational latency)

- Once the head is in place, we need to wait until the right sector is underneath the head.
  - This may require as little as no time (reading consecutive sectors) or as much as a full rotation (just missed it).
  - On average, for random reads/writes, we can assume that the disk spins halfway on average.

- Rotational delay depends partly on how fast the disk platters spin.

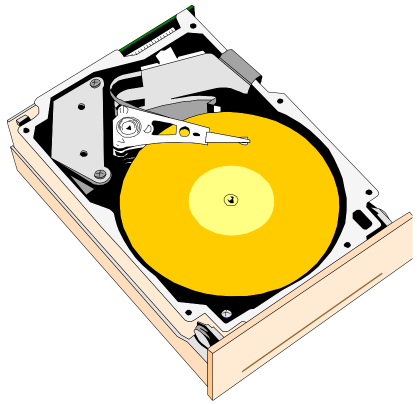
  Average rotational delay = 0.5 x rotations x rotational speed

  - For example, a 5400 RPM disk has an average rotational delay of:

  0.5 rotations / (5400 rotations/minute) = 5.55ms

Estimating disk times

- The overall response time is the sum of the seek time, rotational delay, transfer time, and overhead.
- Assume a disk has the following specifications.
  - An average seek time of 9ms
  - A 5400 RPM rotational speed
  - A 10MB/s average transfer rate
  - 2ms of overheads



- How long does it take to read a random 1,024 byte sector?
  - The average rotational delay is 5.55ms.
  - The transfer time will be about (1024 bytes / 10 MB/s) = 0.1ms.
  - The response time is then 9ms + 5.55ms + 0.1ms + 2ms = 16.7ms. That's 16,700,000 cycles for a 1GHz processor!
- One possible measure of throughput would be the number of random sectors that can be read in one second.

(1 sector / 16.7ms) x (1000ms / 1s) = 60 sectors/second.

Estimating disk times

- The overall response time is the sum of the seek time, rotational delay, transfer time, and overhead.
- Assume a disk has the following specifications.
  - An average seek time of 3ms
  - A 7200 RPM rotational speed
  - A 10MB/s average transfer rate
  - 2ms of overheads
- How long does it take to read a random 1,024 byte sector?
  - The average rotational delay is:
  - The transfer time will be about:
  - The response time is then:
- How long would it take to read a whole track (512 sectors) selected at random, if the sectors could be read in any order?

Parallel I/O

- Many hardware systems use parallelism for increased speed.
  - Pipelined processors include extra hardware so they can execute multiple instructions simultaneously.
  - Dividing memory into banks lets us access several words at once.
- A redundant array of inexpensive disks or RAID system allows access to several hard drives at once, for increased bandwidth.
  - The picture below shows a single data file with fifteen sectors denoted A-O, which are "striped" across four disks.
  - This is reminiscent of interleaved main memories from last week.