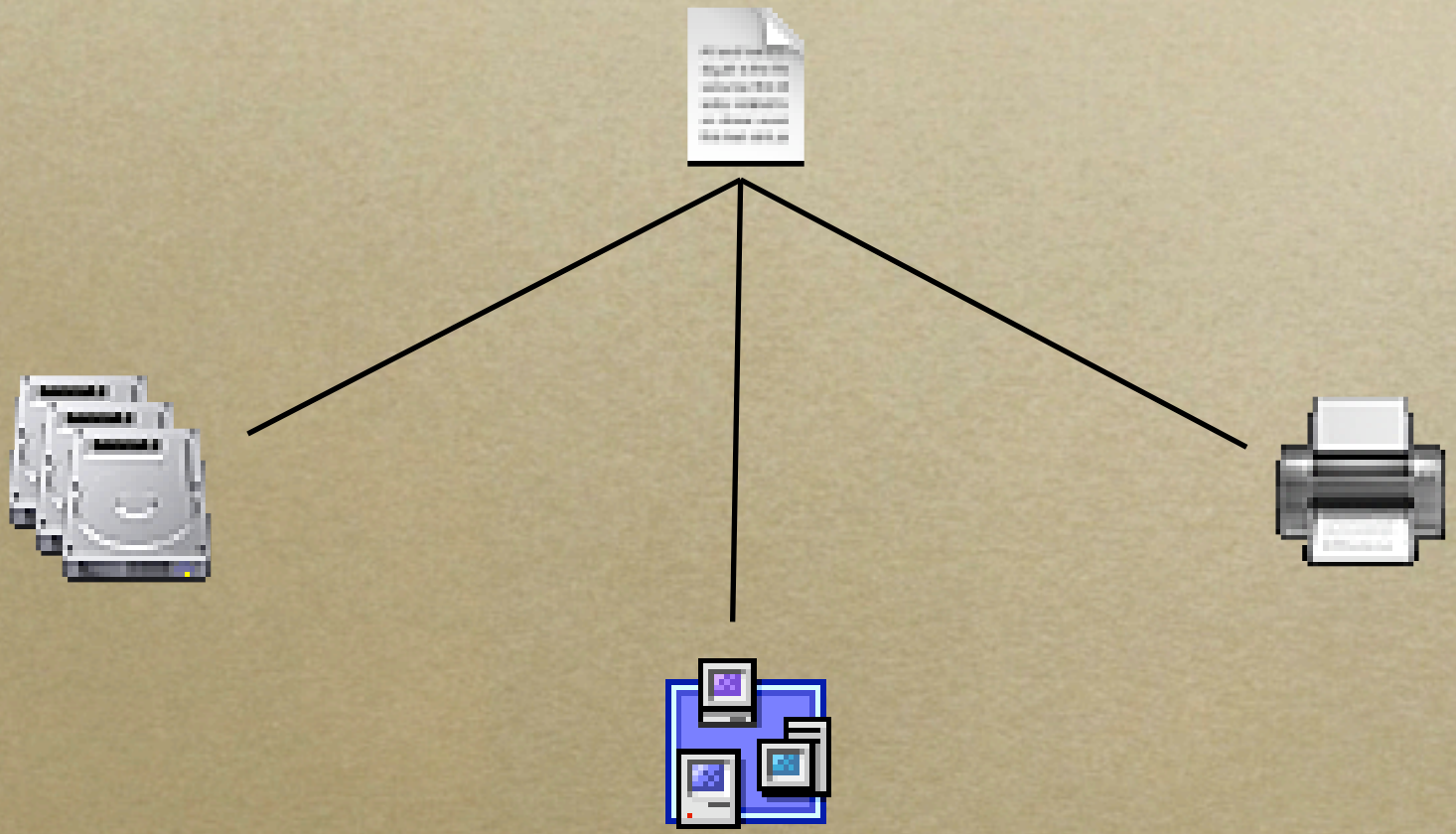
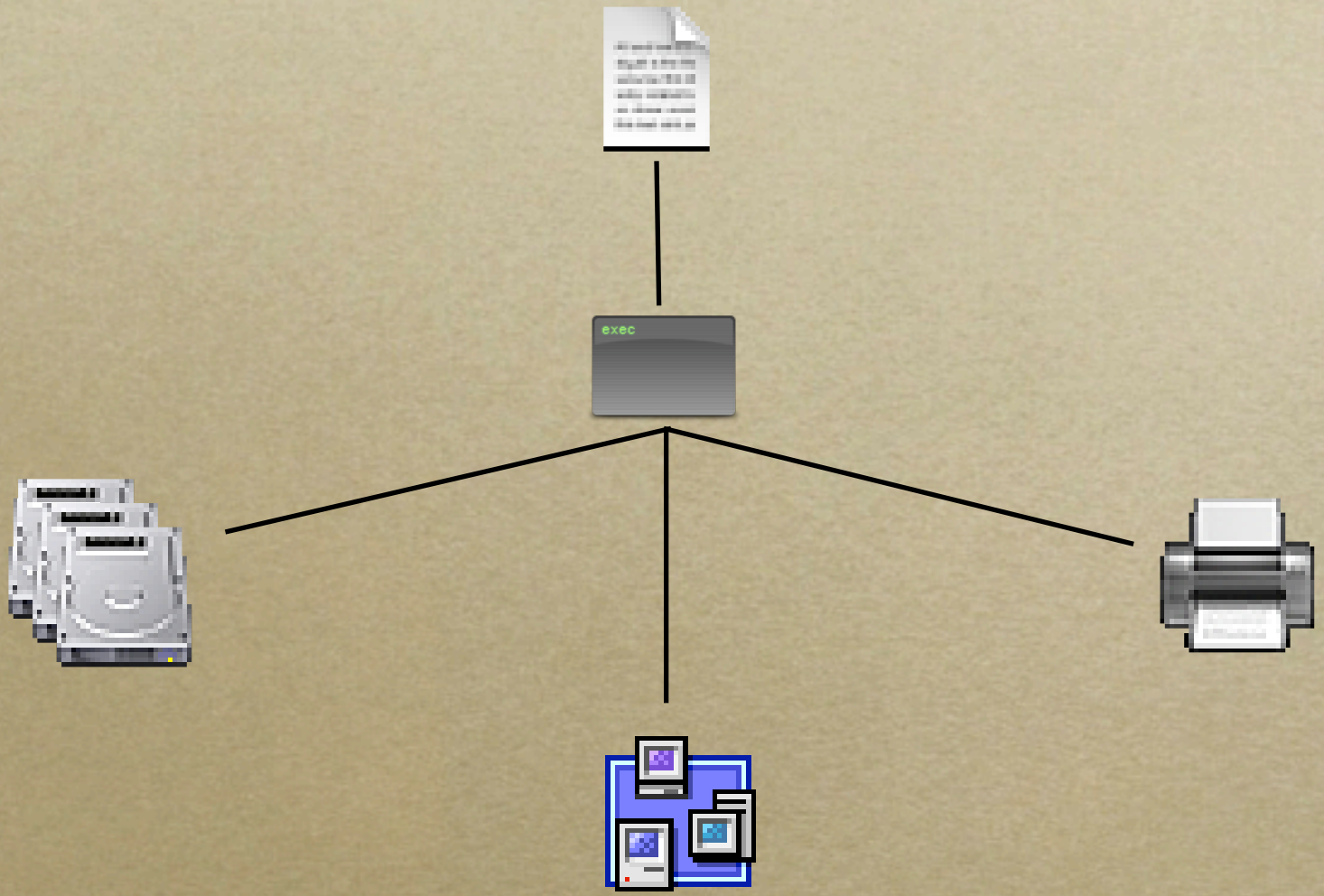
CS 415
Operating Systems Practicum

Computer Architecture Review

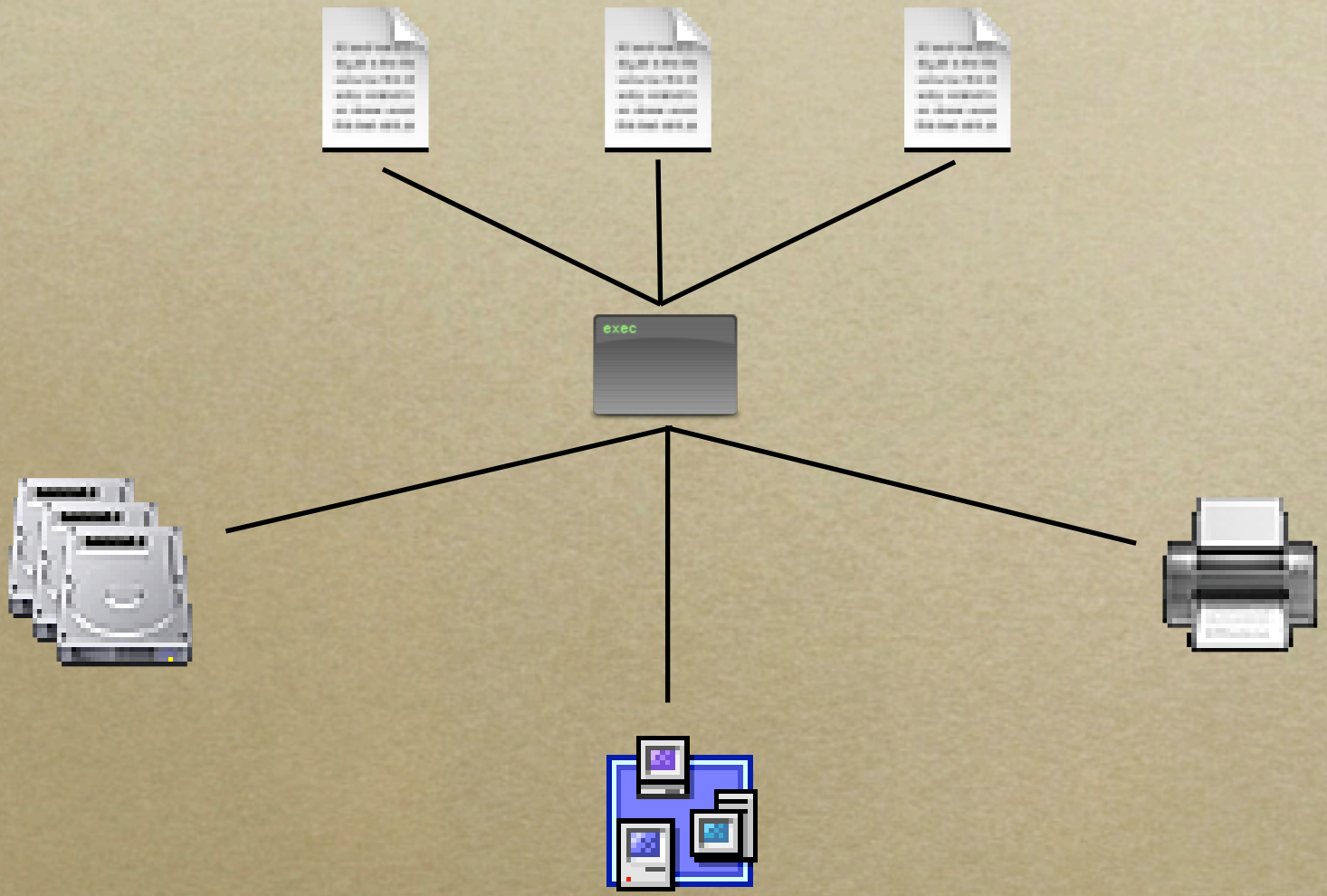
The Dawn of Computing



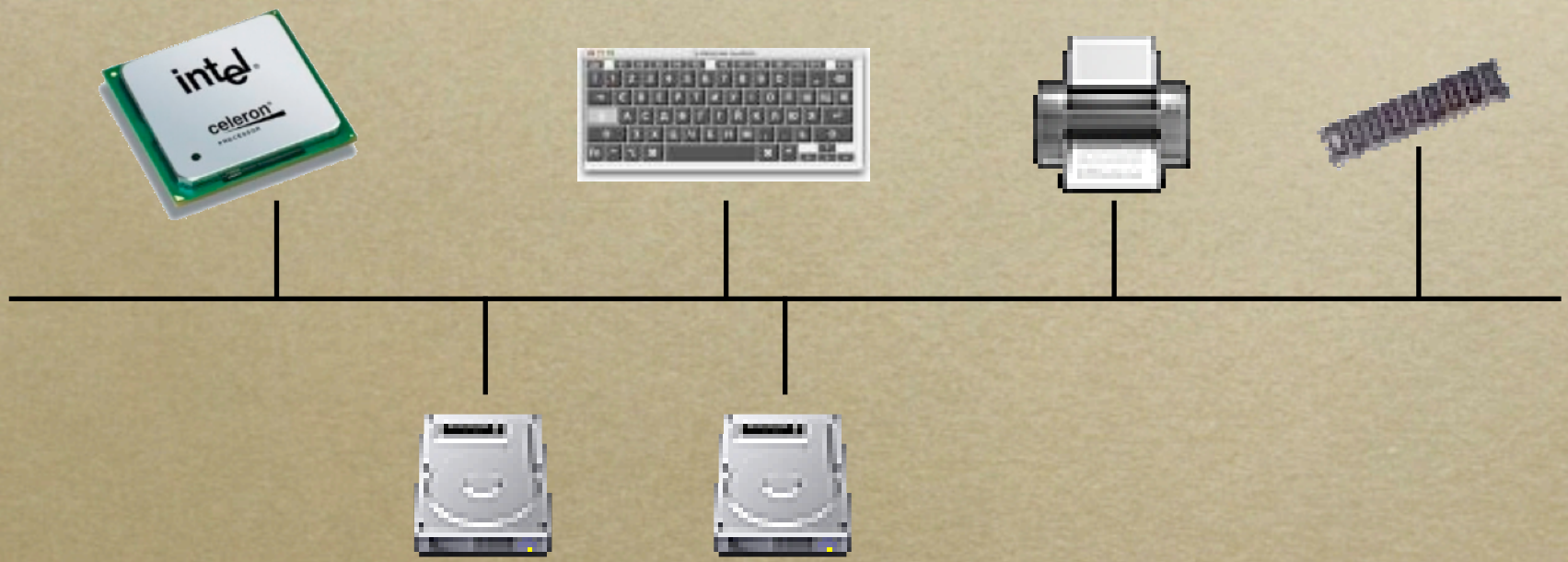
The OS!



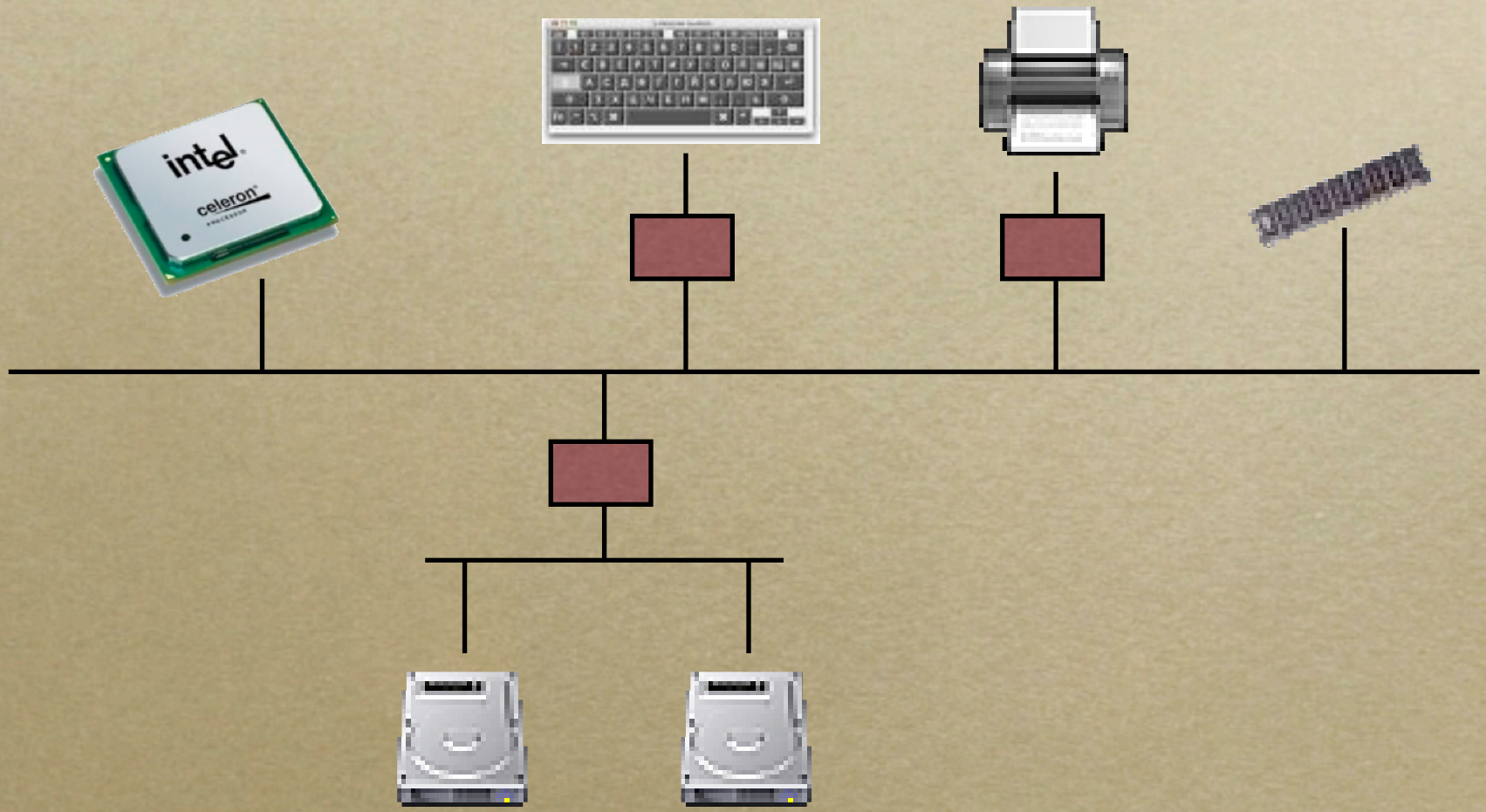
Multitasking



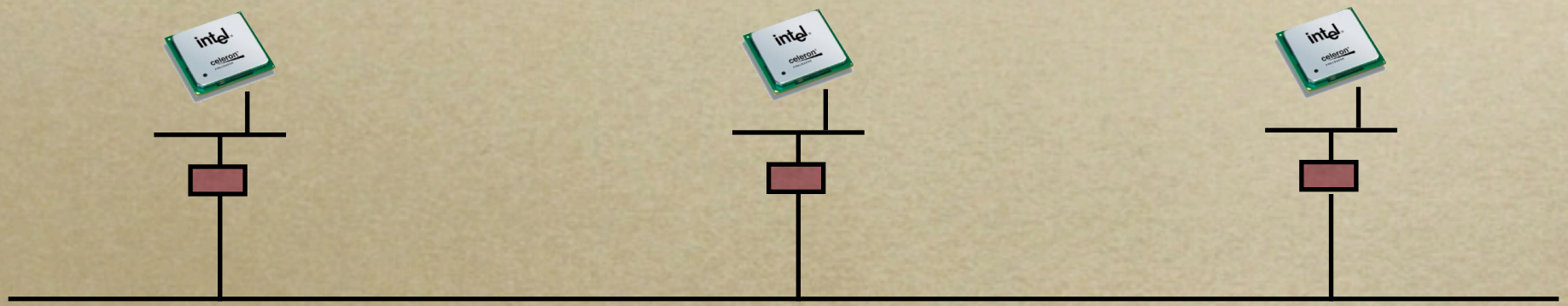
So what's under the hood?



Well... not quite



Networks



A CPU

- Registers:
  - The CPU's short term memory.
- Arithmetic Logic Unit:
  - Where most of the work gets done.
- Floating Point Unit:
  - Handles the decimal calculations.
- Cache:
  - Reduces memory access times.

The Pipeline

- A lot of computation goes into a single instruction.
- Can some of this computation be done in parallel?
- Set up an assembly line.
  - Each stage processes a little and passes it on.
  - Less work per stage means stages go by faster.
- Why not have lots and lots of stages?
  - What happens if we don’t know what will happen next?
  - What happens if one instruction needs data from an earlier instruction?

The Pipeline

- Avoiding delays:
  - Branch Prediction.
  - Instruction Reordering.
- Currently, most pipelines are 10-15 stages in length.
  - Fetch the instruction
  - Decode/Dispatch the instruction.
  - Get necessary data.
  - Perform necessary calculations.
  - Write the results to registers/memory.

The Memory Hierarchy

- Registers: 8-64 integers/floats at a time.
  - Available immediately.
- L1 Cache: ~32KB Data, ~32KB Instructions.
  - Short access time (2-3 cycles).
- L2 Cache: 1-2 MB.
  - Moderate access time (~10-20 cycles).
- Main Memory: up to 4GB or more.
  - Long access time (on the order of 100 cycles).
- Prefetching is used to increase cache hits.

Traps/Interrupts

- What does the hardware do when something happens?
  - The software does something wrong. (Divide by 0)
  - The software asks for a wakeup call.
  - A packet arrives over the network.
- It could just set a flag and have the software check for it.
  - Processor intensive.
  - Defeats the point of an operating system.
- Instead, the processor pauses what it's doing and and calls a callback.

Traps/Interrupts

- How does the processor know what code to execute?
  - Most architectures define a datastructure for a Interrupt Vector Table.
    - States where callbacks are stored.
    - ... and sometimes how they should be called.
  - x86 uses the first 1k of memory.

Traps/Interrupts

- But what happens if an interrupt is interrupted?
  - Disable interrupts while processing one.
  - Have multiple levels of interrupt.
  - Most architectures keep a short queue of interrupts waiting to be delivered.
- But what about traps?
  - Identical to interrupts, except triggered by code (/ by 0).

IO

- Hard disks are slow. Do we want to wait idly for data to arrive?
  - Signal data availability with an interrupt.
  - Also works with networks. (a packet just arrived)
- The hard drive's connected to the disk controller...
  - A small piece of hardware (the controller) controls the IO device and communicates with the processor via a bus.
  - A small piece of software (a driver) interprets the data sent to the processor by the controller and communicates this information to the OS.
  - Finally, the OS notifies the program that data is available.

Protection Levels

- If the OS is going to be managing multiple programs, how does it stop them from misbehaving?
  - Have multiple access levels. (user, system, etc..)
  - Restrict access to certain instructions in less privileged levels.
- Most architectures have an instruction to give up privileges, but no instruction to regain them.
  - But how can the OS access the privileged instructions?
    - Traps! (The famous INT 21)

Memory Management: Segments

- The 80286 was a 16 bit processor.
  - It could address 2^16 (64k) of memory.
  - 64k is tiny! Couldn't it use more?
    - It divided memory up into 64k segments. Applications that needed more could set a segment register to change which segment their addresses pointed to.
- The 80386 was a 32 bit processor.
  - It could address 2^32 (4 gb) of memory.
  - Segments were still convenient for isolating applications from each other.

Memory Management: Segments

- The x86 architecture specifies several Segment Descriptor Tables which list blocks of memory.
  - The segment register(s) serve as an index into these tables.
  - The OS can allocate a chunk of memory to a process by adding an entry to these tables.
    - The process thinks it has the whole address space to itself.
    - When switching processes, the OS just changes the segment register.
  - This solution gets messy quickly.
    - All the memory has to be allocated upfront
    - Reallocating can lead to fragmentation.

Memory Management: Pages

- Instead, let's break memory up into a large number of chunks (pages) and hand them to processes as needed.
- Create a Page Table
  - When the CPU is told to access an address in memory, it consults this table and translates the virtual address to a physical address.
- A program has access to the whole 4 gb range of addresses.
  - ... even if the computer doesn't have 4 gb of memory.
    - The OS can allocate pages as they're needed.
  - When switching processes, the OS tells the CPU to use a different set of translations.

Memory Management: Pages

- What happens when code tries to access an unallocated page?
  - On the x86, a trap called a Page Fault occurs.
  - The OS can allocate the page and the application is none the wiser that anything happened.
- The Translation Lookaside Buffer
  - The Page Table is stored in main memory. (slow)
  - The CPU keeps a cache of recent translations.
  - When switching processes, this cache needs to be cleared.
    - Major slowdown!

Memory Management: Pages

- The x86 Way:
  - The Page Table datastructure is defined by the architecture.
  - A register points to the active page table.
  - The processor updates the TLB manually.
- The MIPS Way:
  - The OS can store Page Tables as it sees fit.
  - When a TLB miss occurs, a trap occurs and the OS updates the TLB.

Memory Management: DMA

- Interrupts are slow.
  - Pausing running code takes a lot of work.
  - ... doubly so if you need to do anything complex.
- Transferring data from disk to memory involves a lot of interrupts.
- Let the IO controller write directly to memory.
  - This is called Direct Memory Access
- Another twist: Memory Mapped IO.
  - Let the disk controller pretend to be a portion of memory.

Synchronization

- How do you ensure that two processes don't try and modify a variable at the same time?
  - Have a variable to lock?
    - But the variable suffers the same problem.
  - Turn off interrupts?
    - Ugly, but it works.
  - Test and Set.