CPSC 457

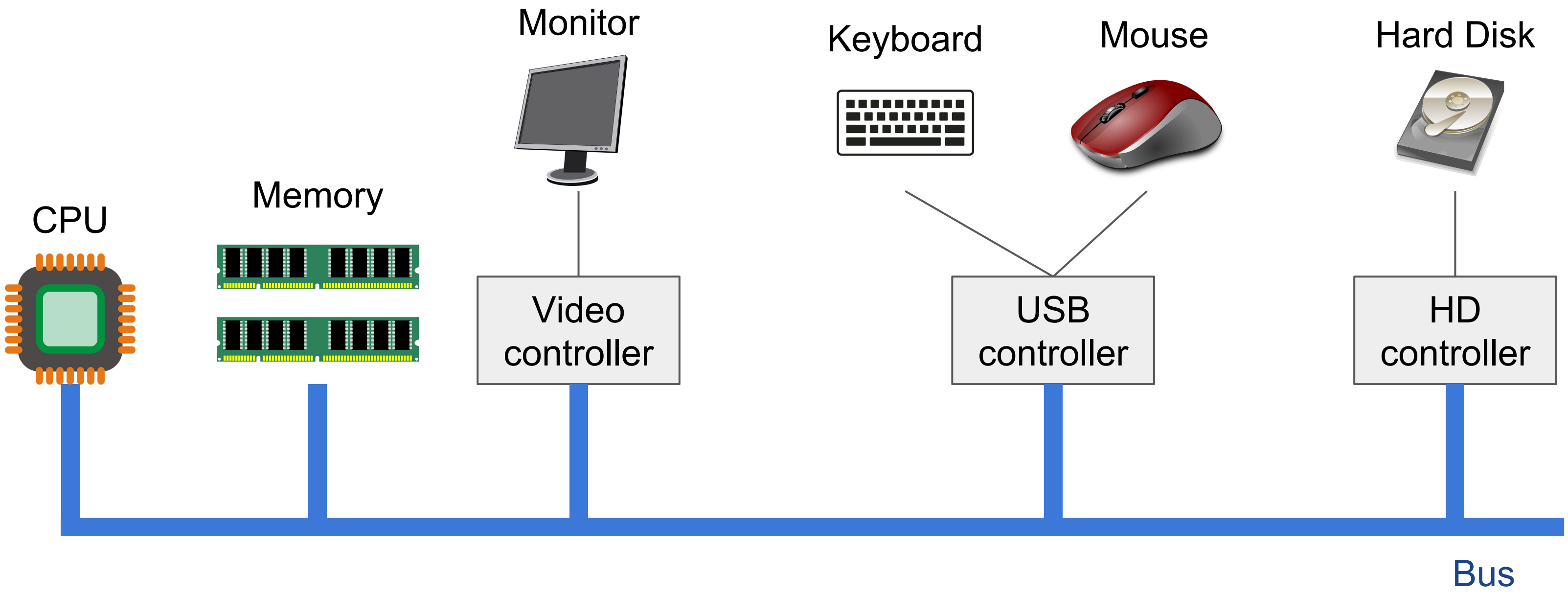
Hardware, booting, cache, kernel

Contains slides from Mea Wang, Andrew Tanenbaum and Herbert Bos

Outline

- hardware review
- caching
- booting the computer
- traps, interrupts
- kernel mode, user mode
- kernel designs
- virtual machines

Hardware review



common components of a desktop computer

CPU

- the “brain” of the computer
- on-board registers for faster computation
  - instead of accessing memory for every instruction
  - accessing information in registers is much faster than memory
- general purpose registers:
  - data & addresses
- special purpose registers:
  - program counter: contains memory address of the next instruction to be fetched
  - stack pointer: points to the top of the current stack in memory
  - status register: interrupt flag, privilege mode, zero flag, carry flag, …
- other (floating point, vector, internal, machine specific, etc)

CPU

- a simple CPU cycle:

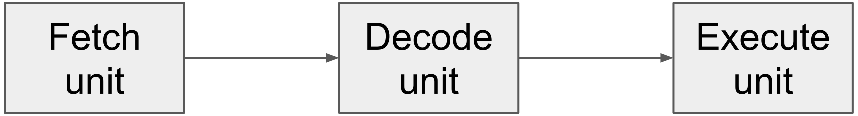
  1. fetch an instruction from memory
  2. decode it to determine its type and operands
  3. execute it

  repeat for the next instruction... until the program finishes

- performance problem: fetching from memory takes longer than executing an instruction
- solution: **pipeline** the operations
  - while executing instruction N,
  - the CPU could be simultaneously decoding instruction N+1,
  - and at the same time also fetching instruction N+2

CPU pipelining

- while executing instruction N, the CPU could be decoding instr. N+1 and fetching instr. N+2.
- we make fetch, decode and execute units to work independently and in parallel
- three stage pipeline:



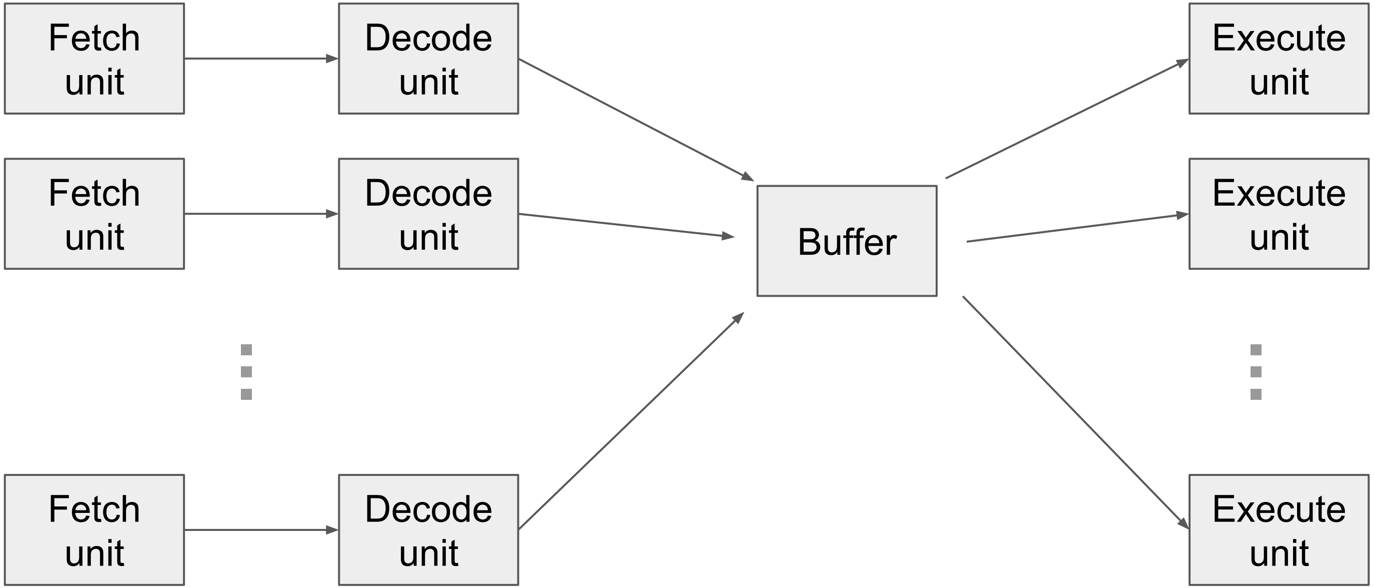
Benefits:

- the CPU can 'execute' more than one instruction at a time
- 'hide' the memory access time

Cons:

- more complexity

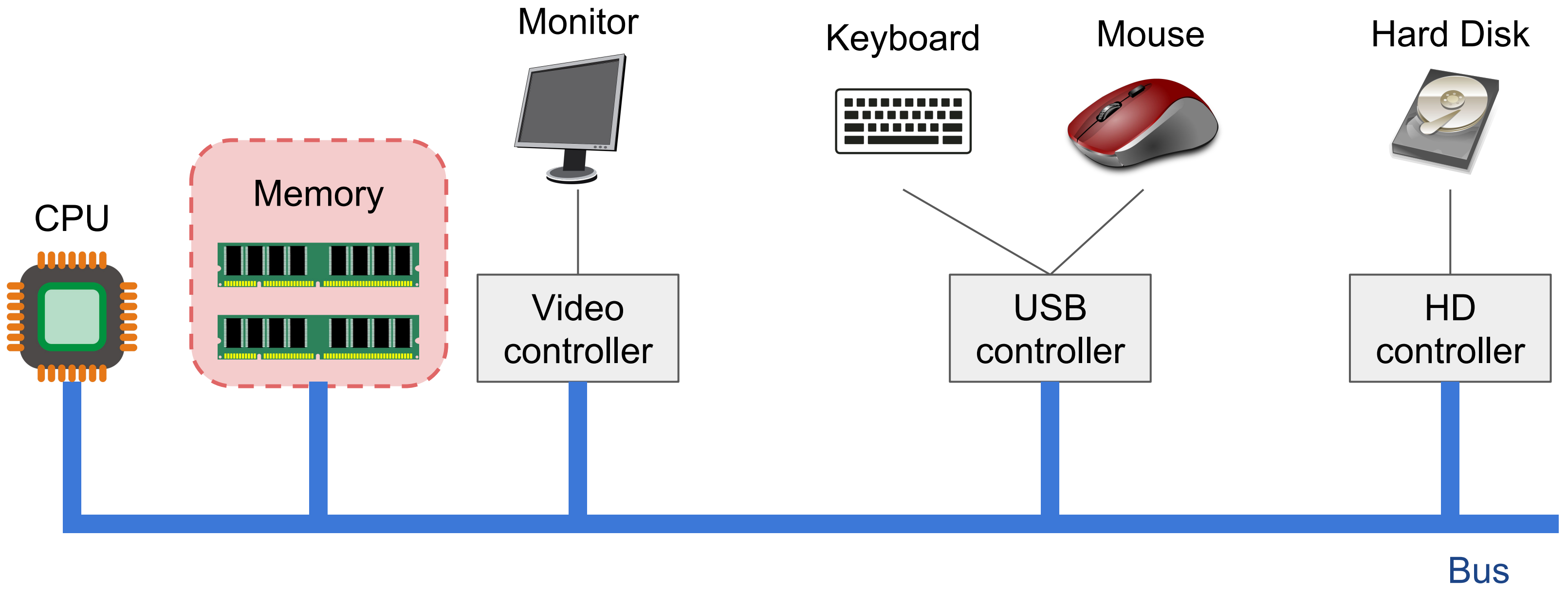
CPU pipelining



superscalar CPU

Memory

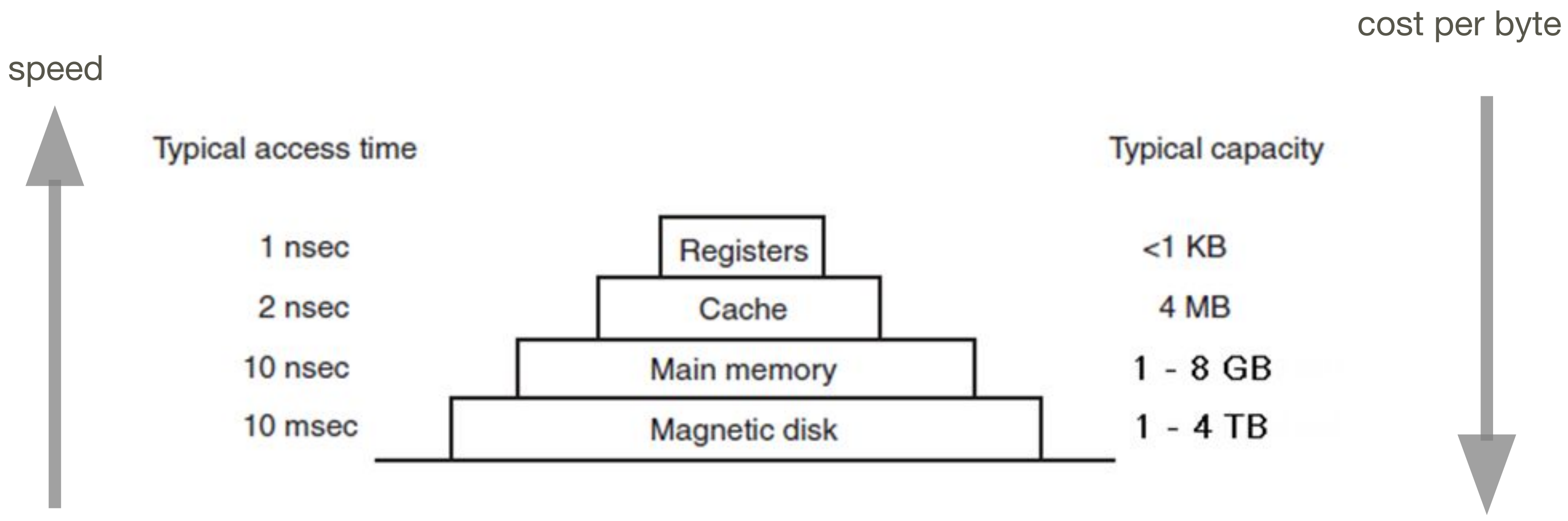
- ideally, memory should be (i) fast, (ii) large and (iii) cheap
- in practice, we can get 2 of the 3, but not all three



Memory

- main memory: random-access memory (RAM)
- consists of an array of words, and each word has its own address (memory address)
- memory operations:
  - load: moves a word from memory to CPU register
  - store: moves the content of a register to memory
- both load and store are slow operations compared to the speed of the CPU

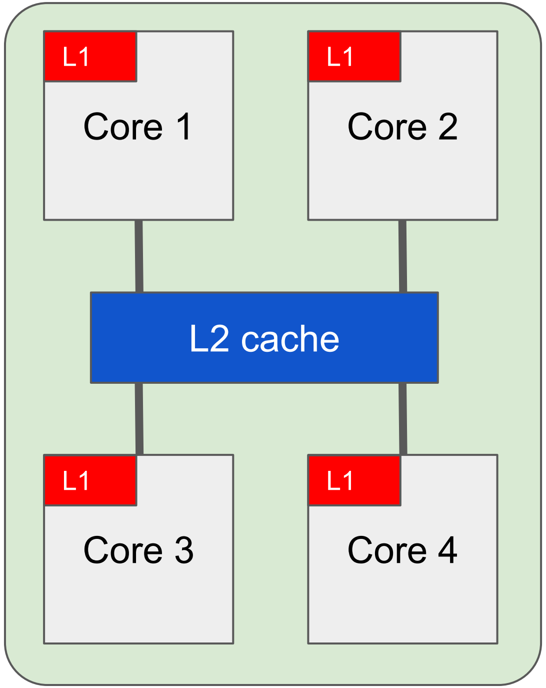
Typical memory hierarchy



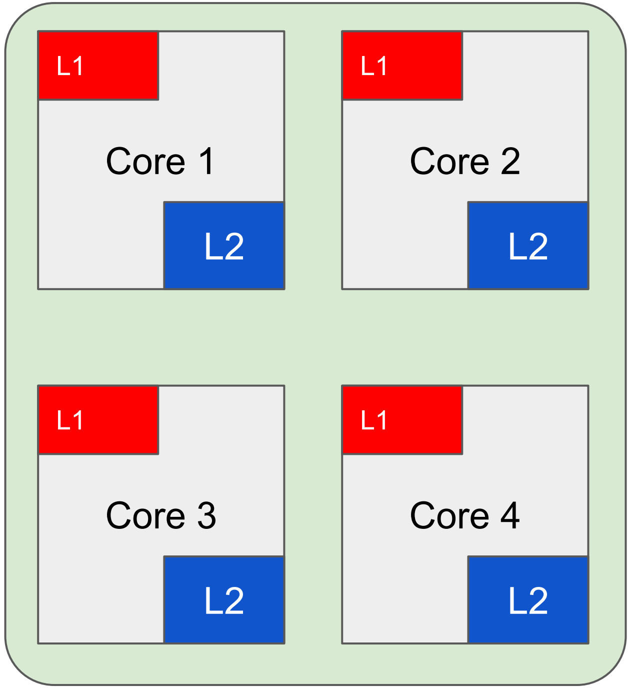
Caching

- CPU caching
  - most heavily used data from memory is kept in a high-speed cache located inside or very close to the CPU
  - when CPU needs to get data from memory, it first checks the cache
  - **cache hit**: the data needed by the CPU is in the cache
    cache miss: CPU needs to fetch the data from main memory
- CPU caches:
  - L1 cache (16KB): inside the CPU, usually feeds decoded instructions into CPU execution engine
  - L2 cache (xMB): stores recently used memory words, slower than L1
  - L3 and even L4 becoming common

Caches on multicore CPUs



(a) A quad-core chip with a shared L2 cache.



(b) A quad-core chip with separate L2 caches.

Caching

- the goal of caching is to increase performance of slower memory/device by adding a small amount of fast memory (cache)
- improving read performance:
  - keep copy of information obtained from slow storage in cache
  - next time we need the information, check the cache first
- improving write performance
  - write info to fast storage, and eventually write to slow storage
- caching is a very useful concept in general
- many uses: disk cache, DNS, database
- cache storage is fast but expensive, so it's usually much smaller than the slow storage

Caching

- cache storage is fast but expensive, so it's usually much smaller than the slow storage
- some general caching issues:
  - when to put a new item into the cache
  - which cache line to put the new item in
  - which item to remove from the cache when cache is full
  - where to put a newly evicted item in the larger memory
  - multiple cache synchronization
  - how long is the data in cache valid (expiration)
- different answers based on the application

Memoization

- similar concept to caching
- optimization technique used to speed up programs, by storing results of expensive computations

```
def fib_slow(n):
  if n < 2:
    return n
  else:
    return fib_slow(n-1)
         + fib_slow(n-2)
```

Memoization

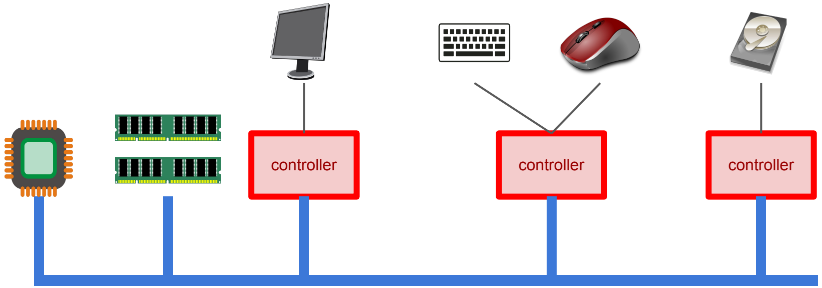
- similar concept to caching
- optimization technique used to speed up programs, by storing results of expensive computations

```
def fib_slow(n):
  if n < 2:
    return n
  else:
    return fib_slow(n-1)
         + fib_slow(n-2)
```

```
cache = {}
def fib_fast(n):
  if n not in cache.keys():
    if n < 2:
      cache[n] = n
    else:
      cache[n] = fib_fast(n-1)
               + fib_fast(n-2)
  return cache[n]
```

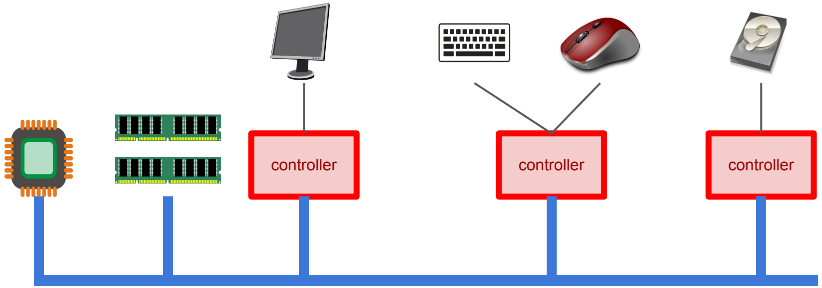
Hardware review - I/O

- I/O devices usually implemented in two parts: device controller and the device
- **device controller**
  - a chip or a set of chips that physically control the device
  - controlling the device is complicated, and CPU could be doing other things, so the controller presents a simpler interface to the OS
  - there are many different types of controllers
- **device**
  - connects to the computer through the controller
  - follows some agreed standard for communication



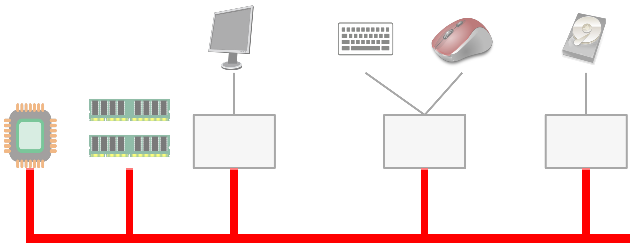
Hardware review - I/O

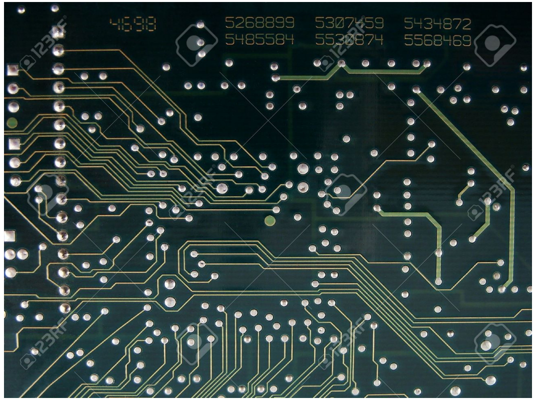
- **device driver**
  - is the software that talks to a controller, issues commands and accepts responses
  - usually written by the controller manufacturer, follows some abstraction
  - needed so that an OS knows how to communicate with a controller
  - drivers can be implemented as kernel modules, loaded on demand



Buses

- a communication system for transferring data between different computer components
- modern computer systems have multiple busses, eg. cache, memory, PCI, ISA, etc
- each has a different transfer rate and function
- OS must be aware of all of them for configuration and management
- for example, collecting information about the I/O devices
- assigning interrupt levels and I/O addresses
- much of this is done during the boot process



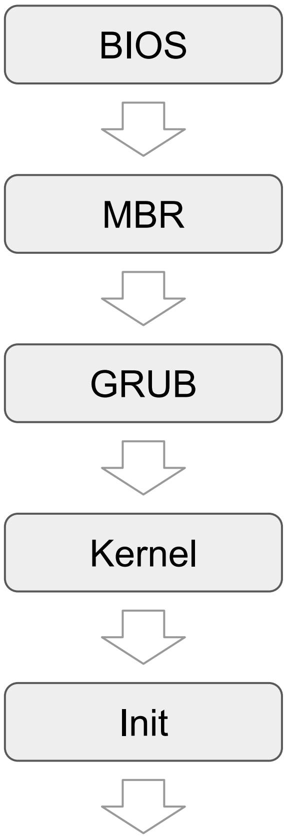


Booting

http://www.youtube.com/watch?v=C2Ph8zwpNyI&feature=related

Booting a system

- when the computer is booted, the BIOS is started
  (Basic Input Output System) is a program stored on motherboard
- check the RAM, keyboard, other devices by scanning the ISA and PCI buses
- record interrupt levels and I/O addresses of devices, or configure new ones
- determine the boot device (ie. try list of devices stored in CMOS)
- read & run primary boot loader program from first sector of boot device
- read & run secondary boot loader from potentially another device
- read in the OS from the active partition and start it
- OS queries the BIOS to get the configuration information and initialize all device drivers in the kernel
- OS creates a device table, and necessary background processes, then waits for I/O events



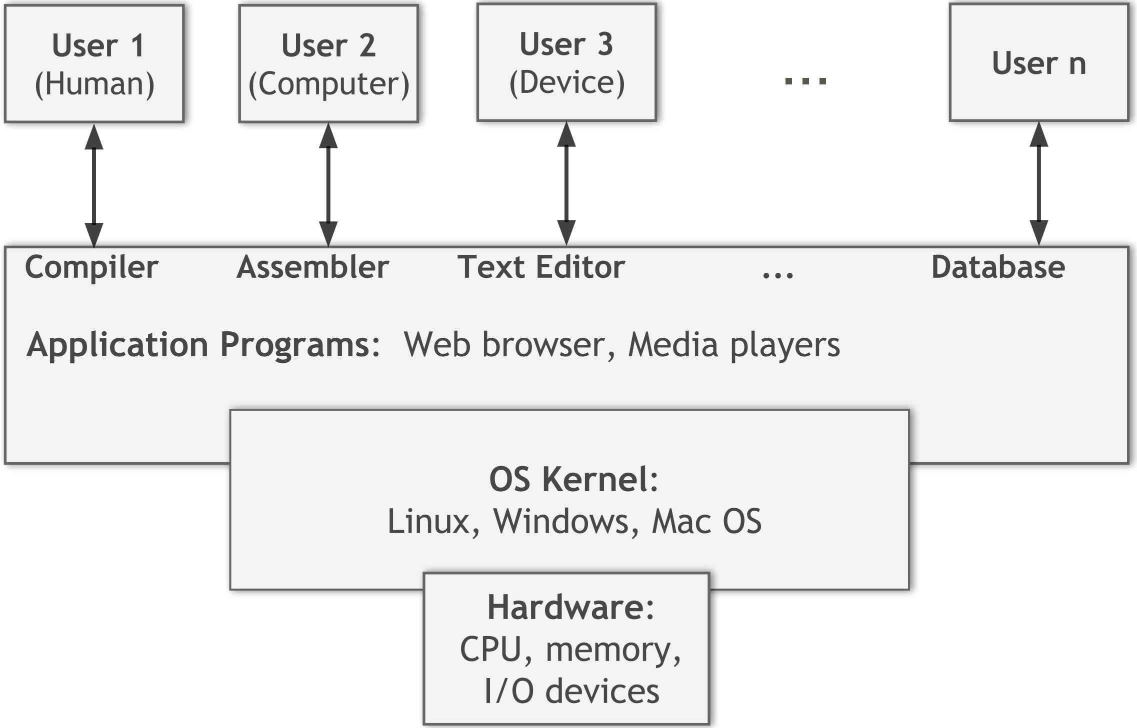
Kernel

- the central part, or the 'heart' of the OS
  - kernel is running at all times on the computer
  - located and started by a bootstrap program (boot loader)
  - provides services to applications via system calls
  - handles all interrupts
  - much of the kernel is a set of routines, some invoked in response to interrupts, others because of application using system calls, etc.
- notes:
  - application programs are not part of OS
  - system programs are part of OS, but not part of kernel
  - "running at all times" mostly means listening and responding to events from hardware

Kernel mode

- most modern CPUs support at least two privilege levels: kernel mode and user mode
- the mode is usually controlled by modifying the status register
- on CPUs without this support, there is only kernel mode
- when CPU is in **kernel mode** (aka unrestricted mode):
  - all instructions are allowed
  - all I/O operations are allowed
  - all memory can be accessed
  - note: most of the kernel runs in kernel mode
- when OS runs a normal application, it runs it in a **user mode:**
  - only subset of operations are allowed
  - eg. accessing the status register is disallowed (of course),
    I/O instructions not allowed, access to some parts of memory not allowed, …
  - illegal instructions result in traps (exceptions)

Kernel vs. user mode



User mode - no access to hardware, limited instruction set

Kernel mode - full access to hardware, full instruction set

User mode

- all applications run in user mode, including ones that came with the OS
  - that means applications cannot talk to hardware… (directly)
  - how do we read/write files then?
- applications must ask the kernel to do I/O
  - cannot be a simple function call, why?
  - application must invoke a **system call**
  - usually accomplished by invoking a **trap** into the operating system
- **trap**
  - often a special instruction (`SWI n`, `INT n`, …)
  - switches from user mode to kernel mode and invokes a **predefined routine**
  - think of it as 'pausing' the application and executing a kernel routine configured by the OS
  - when the kernel routine is done, user mode is restored and application 'resumes'

Review

- Invoking a system call will cause a trap.
  - True or False
- Applications run in user mode.
  - True or False
- Device drivers run in kernel mode.
  - True or False

Summary

- Hardware review
  - Processor
  - Memory & Disks, caching
  - Devices & I/O
  - Buses
- Bootstrapping
  - Traps
  - Kernel mode v.s. user mode

Reference: 1.3, 1.5 - 1.6 (Modern Operating Systems)
1.2 - 1.5 (Operating System Concepts)

Questions?