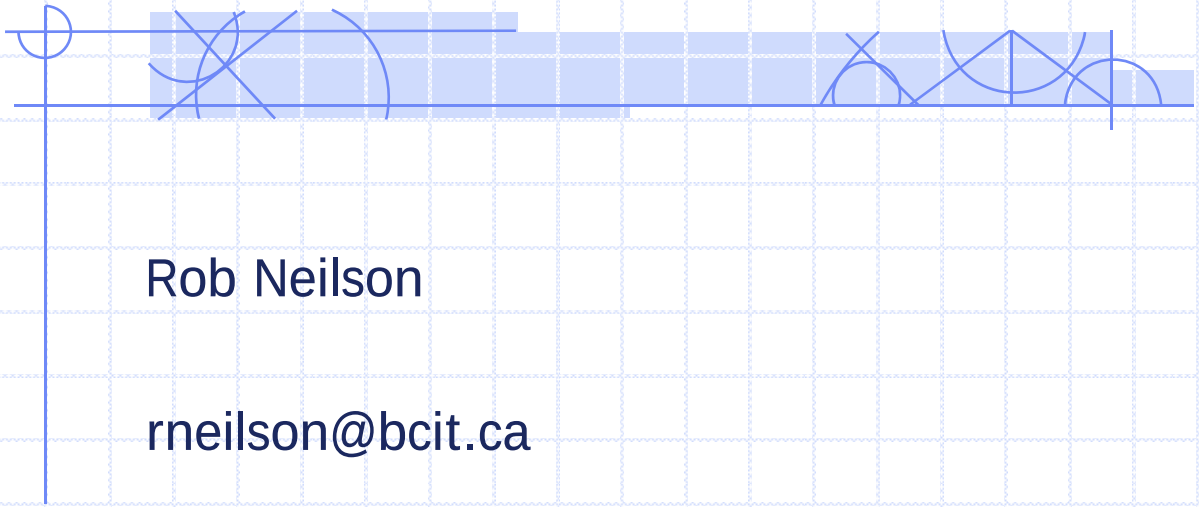
COMP 4735: Operating Systems Concepts

Lecture 3: Computer Hardware Review



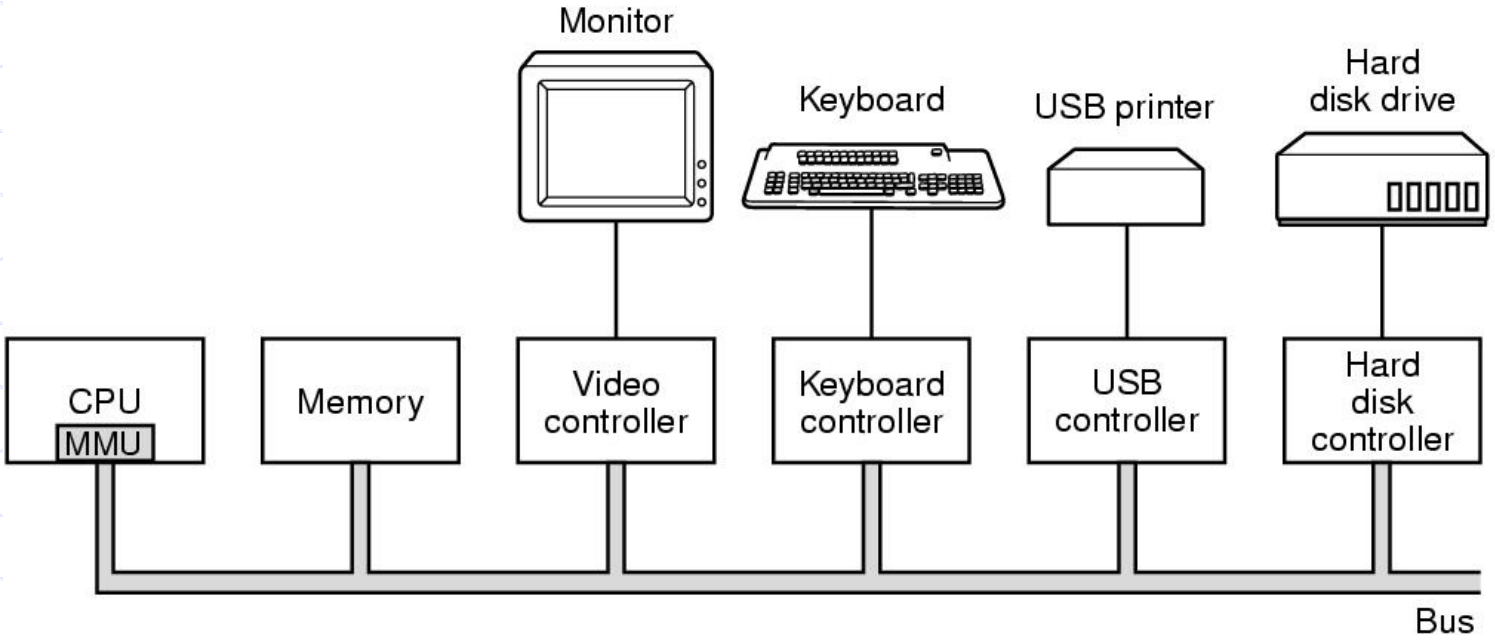
Rob Neilson

rneilson@bcit.ca

Reading

- The following sections should be read before next Monday
  - Textbook Sections: 1.1, 1.2, 1.3 (Theory Part)
    10.1, 11.1 (Case Study Part)
- Yes, there will be a quiz next Monday in lecture
  - A sample quiz will be posted on webct on Friday
  - The sample quiz will be reviewed next Monday in lecture, before the real quiz
  - This quiz (Quiz 1) covers material from all 5 of the sections listed above
    - You might do well to focus on the 'key concepts' presented in lecture

Computer Hardware Review



Some of the components of a simple personal computer.

Note: this is extremely simplified ... here there is a single bus connecting the I/O and storage devices to the CPU, which does the computation

CPU

- Computers have one or more CPU's, which do the actual computing
  - Basic computational cycle: fetch – decode – execute
  - Each processor has an instruction set, which is a set of commands it knows how to process

CPU Registers

- CPU's have registers to store commands and operands and other information

  - Some typical special purpose registers:

    PC (program counter) - contains address of next instruction to be fetched

    IR (instruction register) - contains the current instruction being executed

    AR (address register) - memory (RAM) address being read or written

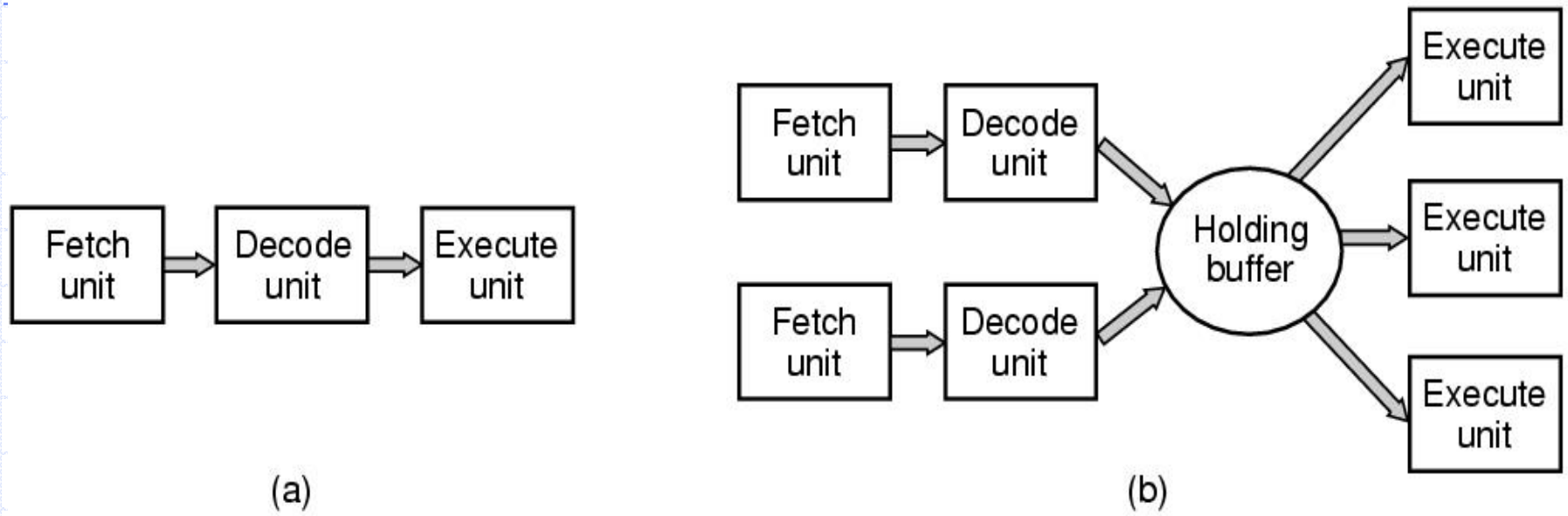
    PSW (Program Status Word) - contains bits that are set and cleared by the OS
    - the bits indicate current CPU state (eg: CPU mode)

    SP (stack pointer) - points to the top of the stack in memory
    - stack holds the local variables, parameters etc for the function being executed

  - There are also a number of general purpose registers:

    store data that is needed for a command

CPU Pipelining



(a) A three-stage pipeline
- uses separate units to fetch/decode/execute at same time

(b) A superscalar CPU
- has multiple processing units to execute instructions at the same time
- consequence is that instructions can be executed out of order

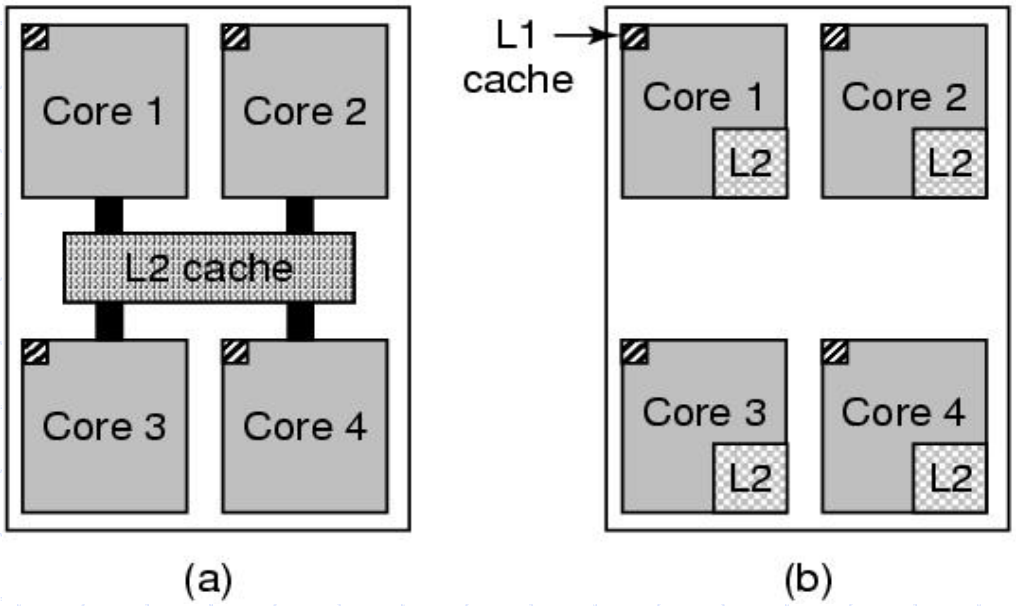
- *Important for OS Designers as anything done on parallel can appear as multiprocessing - and may need to be controlled/managed in the OS*

Multi-threaded Chips

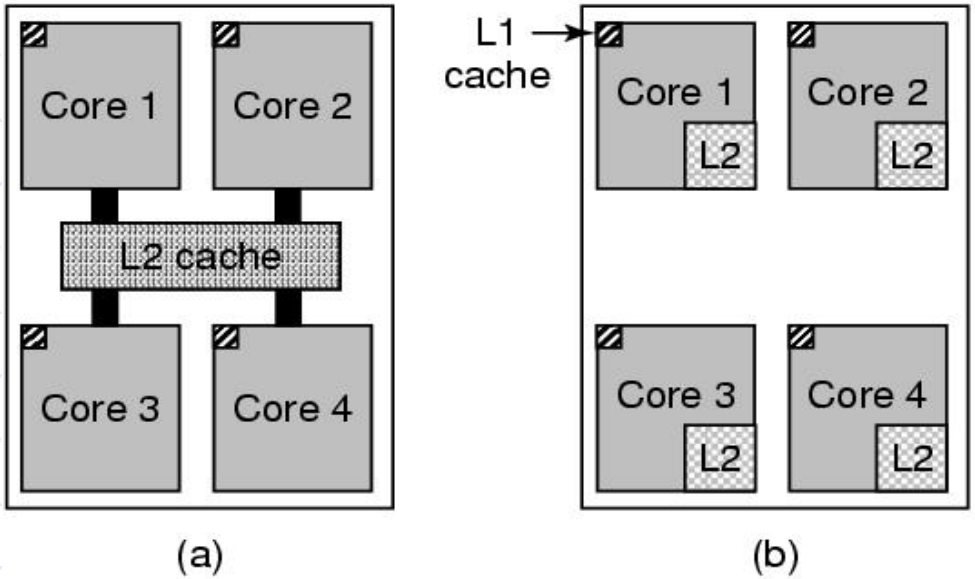
- ***hyperthreading***/multithreading can occur inside a processor (if the processor was designed to support it)
- this requires that the CPU contains internal units (sets of registers) to process the instructions from each process or thread
- *not parallel execution* - but appears that way to OS

Multi-Core Chips

- each ***core*** is a separate processing unit
- a separate process or thread can execute (simultaneously) on each of the processors



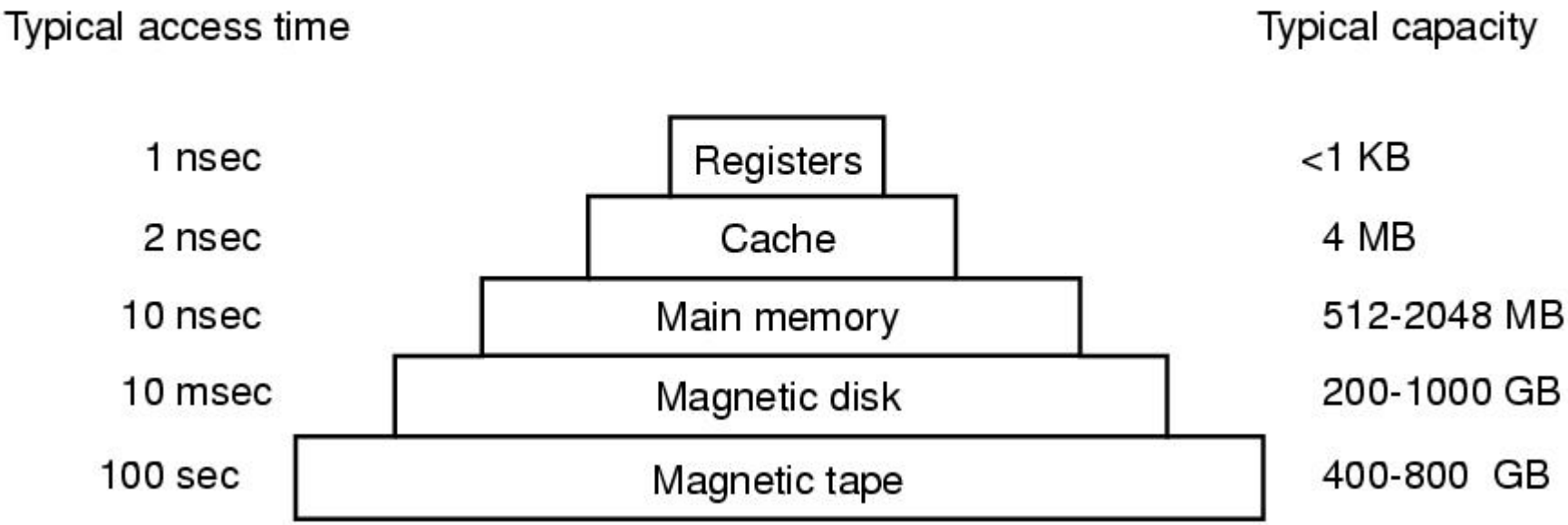
Cache



- CPU's contain a couple levels of cache
  - ***L1 cache*** is small and holds current instructions
  - ***L2 cache*** is a bit slower, but larger to hold more hits

- multi-core chip architects need to decide whether to share L2 cache or allocate it to each core (processor)

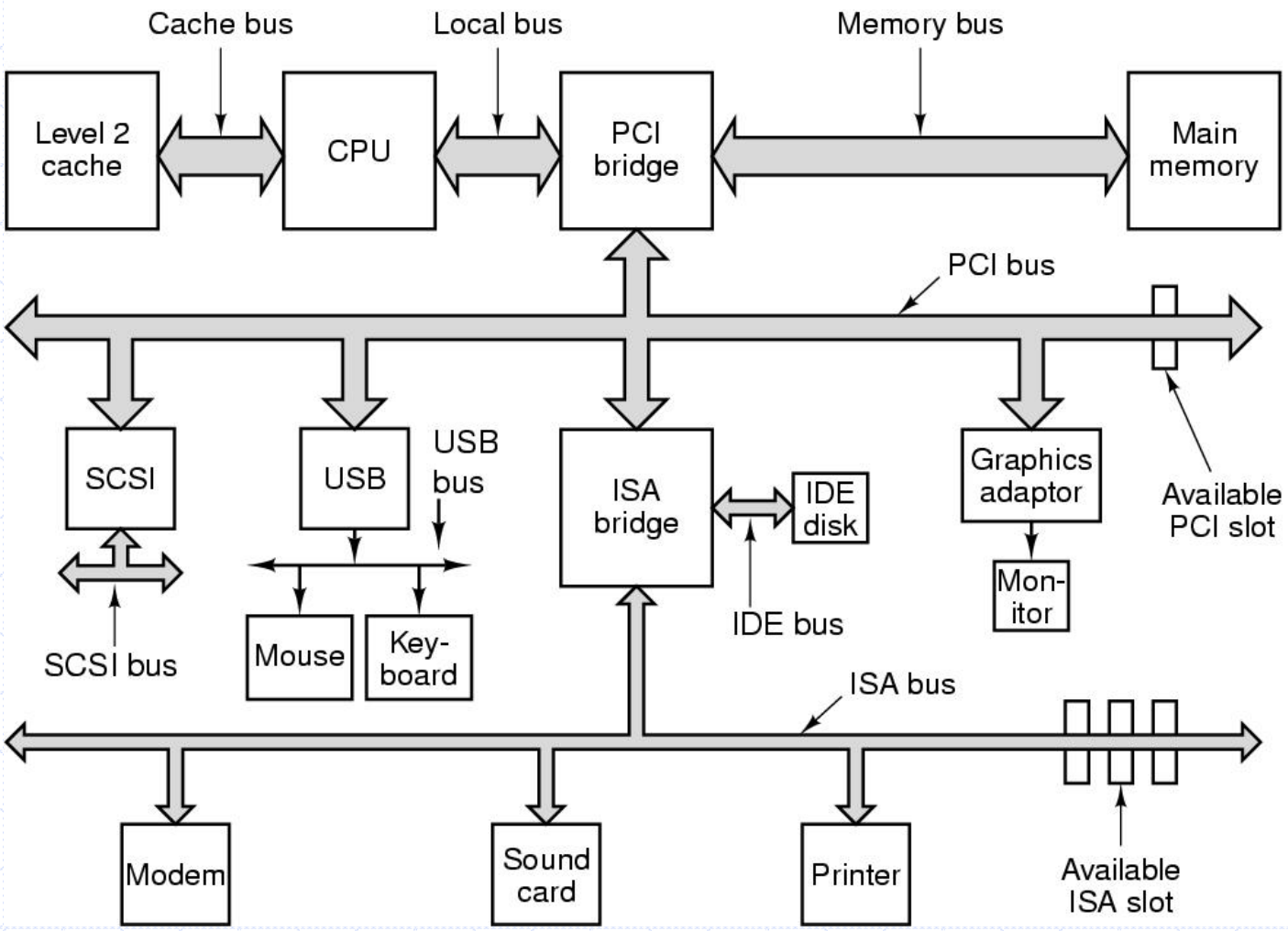
Memory



A typical memory hierarchy.

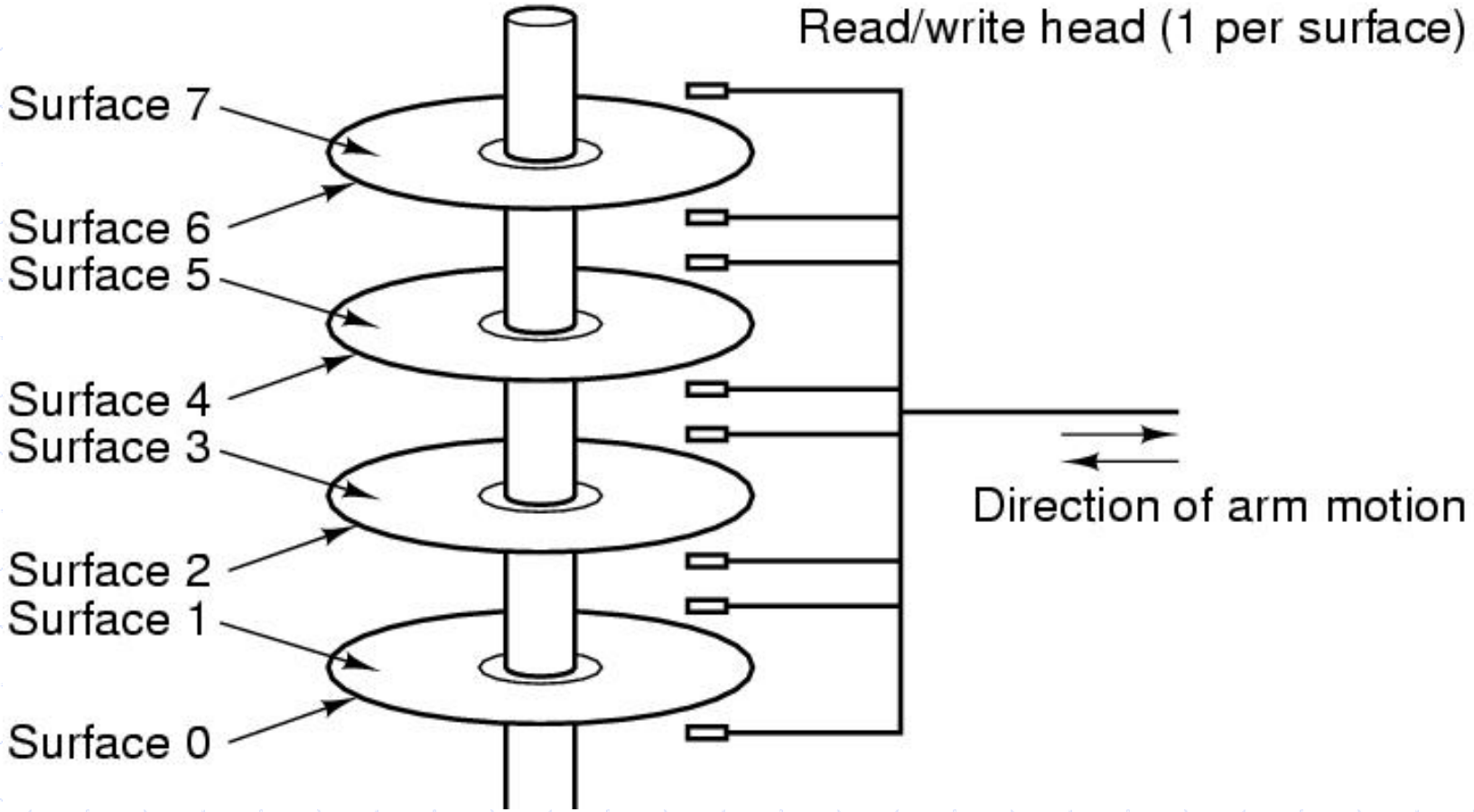
The numbers are very rough approximations.

Buses



The structure of a Pentium system

Disks

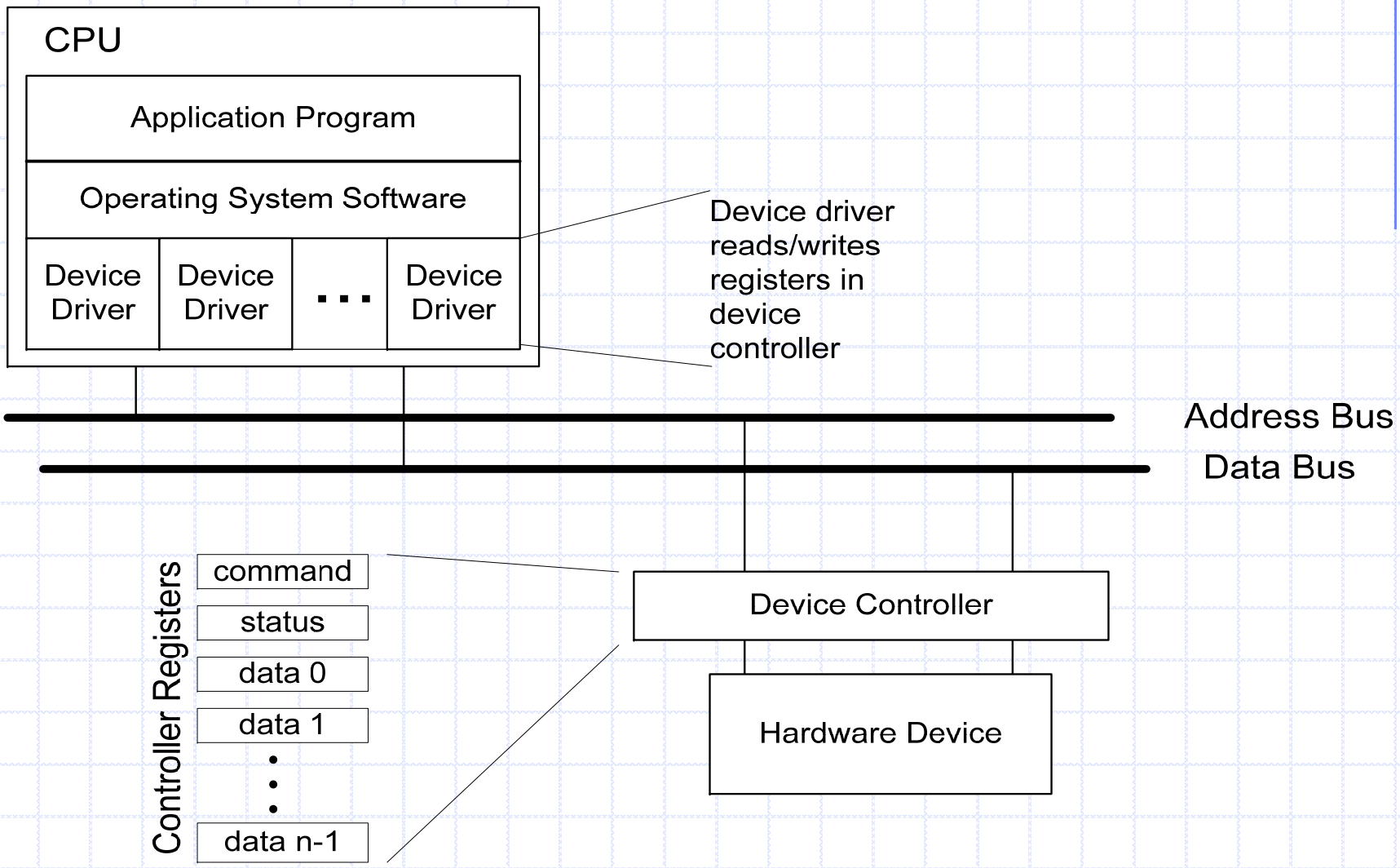


Structure of a disk drive.

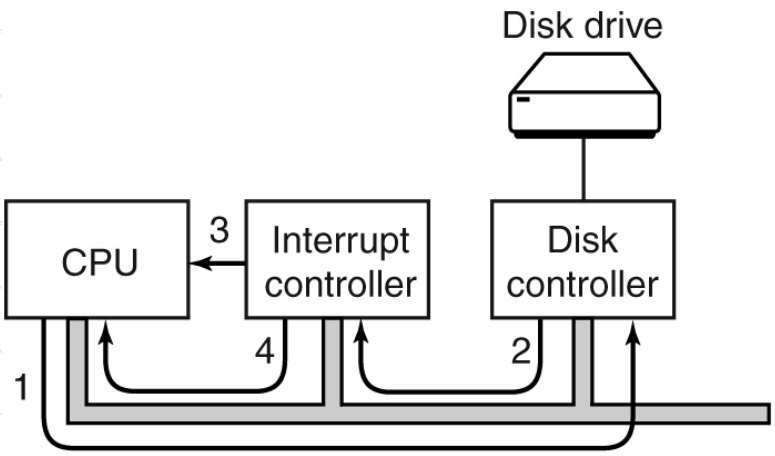
I/O Devices

- pretty much every computer component except the CPU and memory is an I/O device
- the OS has to coordinate access to all these devices
- furthermore, each device comes from the manufacturer with a specific interface
  - ie: each device has different commands and parameters to control reading and/or writing to it
- each device contains a ***controller*** that decodes and performs the commands sent from the CPU
- the software that sends commands and receives responses from the device is called the ***device driver***

Devices, Drivers, and Controllers



Starting an I/O Device



1. **driver writes command** to device control register

   eg: command might be 'start motor', so controller would turn on power to the motor

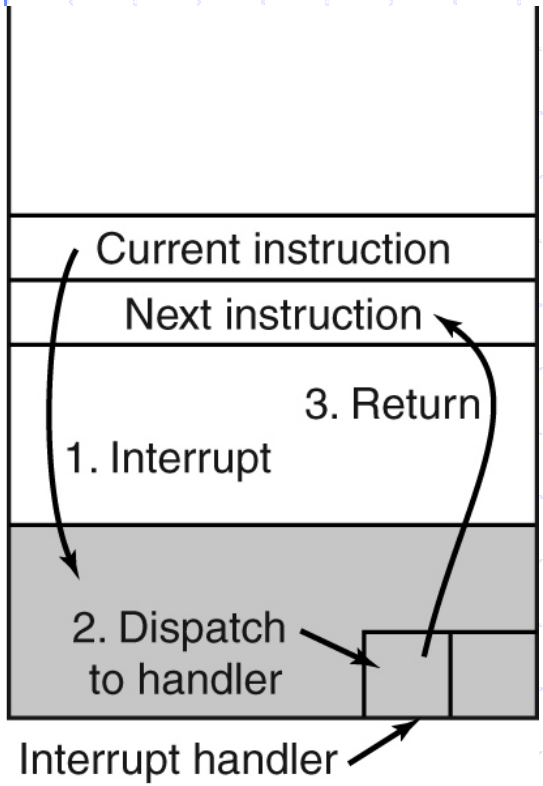
2. **device controller sends interrupt** to CPU to say it is done

   note that the interrupt controller in this case is external to the CPU and processes interrupts from many devices

3. **interrupt controller raises interrupt** line to the CPU (hardware interrupt)

4. **interrupt controller sends device number** to CPU so it knows which driver to load

Processing Interrupts from a Device



In this diagram the CPU is executing some process unrelated to the interrupt.

When the interrupt is received it:

1. **saves registers** such as PC and PSW of the current process on stack, **switches to kernel mode**, and **jumps to an interrupt dispatch routine**
2. the interrupt dispatch routine **reads the device number** for the interrupting device, and **loads the interrupt handler** for the device

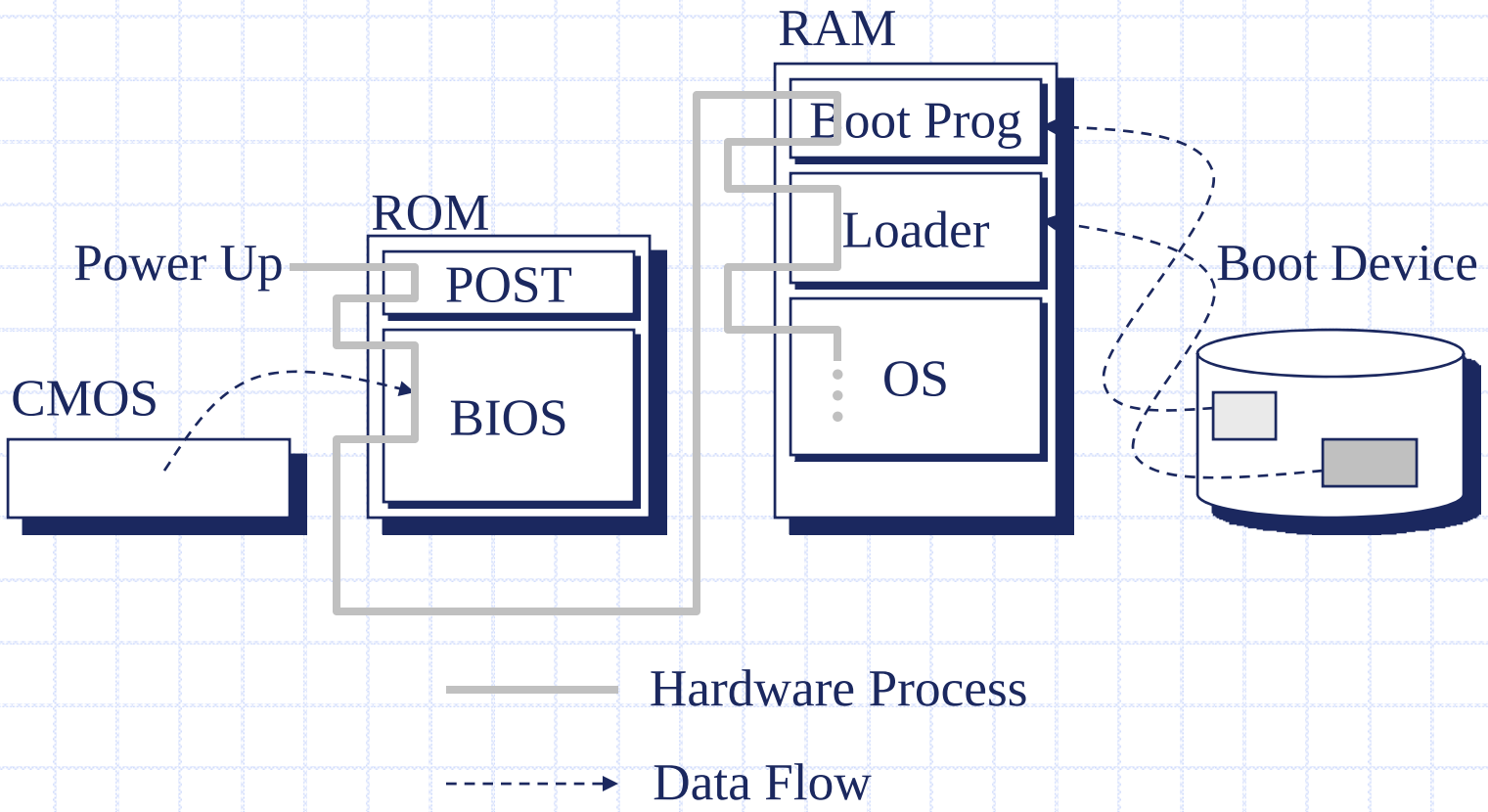
   the interrupt handler then interacts with the device as needed to complete the system call
3. when the handler is finished, the **context** of the original process **is restored from the stack**

Bios (Basic Input / Output System)

BIOS consists of four parts:

1. Power-On-Self-Test (POST)
   - A series of diagnostics routines that test the various system components.
2. BIOS Setup Utility
   - Used to enter, modify and store system configuration data.
3. Internal Diagnostics
   - Used to consist of an extensive set of diagnostic utilities. If available in present BIOS's, it will only contain a Hard Disk Utility, providing low-level format, auto interleave and media analysis.
4. System BIOS
   - Provides the Interrupt Service Routines (ISRs) which are loaded when the system is powered on.
   - The ISRs perform fundamental services necessary to let the system operate correctly.

Bootstrapping an Intel PC



When you turn the power on …

1. The power supply starts the clock generator and asserts #POWERGOOD signal on the bus.
2. CPU #RESET line is asserted (CPU now in real 8086 mode).
3. CPU Registers set to initial values
   - ds=es=fs=gs=ss=0
   - cs=0xFFFF0000
   - eip = 0x0000FFF0 (ROM BIOS POST code).
4. All POST checks are performed with interrupts disabled.
5. IVT (Interrupt Vector Table) initialised at address 0.
6. The BIOS Bootstrap Loader function is invoked via **int 0x19**
   - dl register contains the boot device 'drive number'
   - this loads track 0, sector 1 at physical address 0x7C00 (0x07C0:0000).

Interrupt Vector Table (IVT)

INT	Address	Type	Function
00h	0000:0000h	Processor	Divide error
01h	0000:0004h	Processor	Single step
02h	0000:0008h	Processor	Non-maskable interrupt
03h	0000:000Ch	Processor	Breakpoint instruction
04h	0000:0010h	Processor	Overflow instruction
05h	0000:0014h	BIOS Processor	Print screen Bound range exceeded
06h	0000:0018h	Processor	Invalid opcode
07h	0000:001Ch	Processor	Coprocessor not available
08h	0000:0020h	Hardware Processor Processor	IRQ 0 - System timer Interrupt out of range exception Double exception
09h	0000:0024h	Hardware Processor	IRQ 1 - Keyboard Coprocessor segment overrun
0Ah	0000:0028h	Hardware Processor	IRQ 2 - General adapter use/Cascade Invalid Task State Selector

Interrupt Service Routines

- the code for these is stored at the addresses specified in the IVT
  - eg: bootloader code is at the address for interrupt 19h (0000:0064h)
- to run one of these ISR's, just set the instruction register (IR) to point to the start of the routine
- many ISR's need data (constants etc), which is also supplied by the BIOS

BIOS Data Area

- created by POST at memory location 0040:0000h
  - size is 256 bytes (0040:0000h-0040:00FFh)

Offset	Description	Size	BIOS Service
00h	Base I/O address serial port 1 (COM1)	2 bytes	Int 14h
02h	Base I/O address serial port 2 (COM2)	2 bytes	Int 14h
04h	Base I/O address serial port 3 (COM3)	2 bytes	Int 14h
06h	Base I/O address serial port 4 (COM4)	2 bytes	Int 14h
08h	Base I/O address parallel port 1 (LPT1)	2 bytes	Int 17h
0Ah	Base I/O address parallel port 2 (LPT2)	2 bytes	Int 17h
0Ch	Base I/O address parallel port 3 (LPT3)	2 bytes	Int 17h

BIOS Bootstrap Loader (INT 19h)

- the POST jumps to this address once test is complete
- this is a first stage loader
- it's job is to copy your OS's loader from HDD to memory
- a generalized version would do something like this ...

```
FIXED_LOC:              // Bootstrap loader entry point
        load    R1, =0
        load    R2, =LENGTH_OF_TARGET

        read    BOOT_DISK, BUFFER_ADDRESS
loop:   load    R3, [BUFFER_ADDRESS, R1]
        store   R3, [FIXED_DEST, R1]
        incr    R1
        bleq    R1, R2, loop
        br      FIXED_DEST
```

Second Stage Loader

- this is the loader provided by your OS
  - LILO
  - GRUB
  - NTLDR
- the second stage loader is responsible for loading the kernel image, and starting it running
- second stage loaders are *typically* configurable so you can multi-boot your system
  - except that the NTLDR in XP doesn't really work with linux as it doesn't want to recognize the unix style partitions on the hdd

BIOS Tools and Information

- IVT.EXE ... can be used to dump your IVT
  - written in MASM (Microsoft Macro Assembler)
  - source code is available
- RBILVIEW.EXE
  - has a detailed description of every known BIOS routine
- if you like this type of stuff you will find lots of other good info, and free download tools at

http://www.matrix-bios.nl/

This is the ...

end of the slides