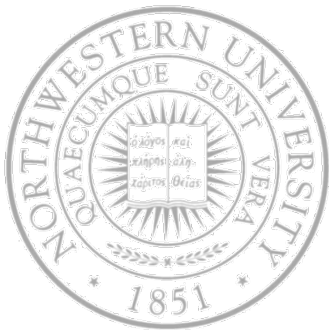


Architectural Support for Operating Systems



Today

- Computer system overview

Next time

- OS components & structure

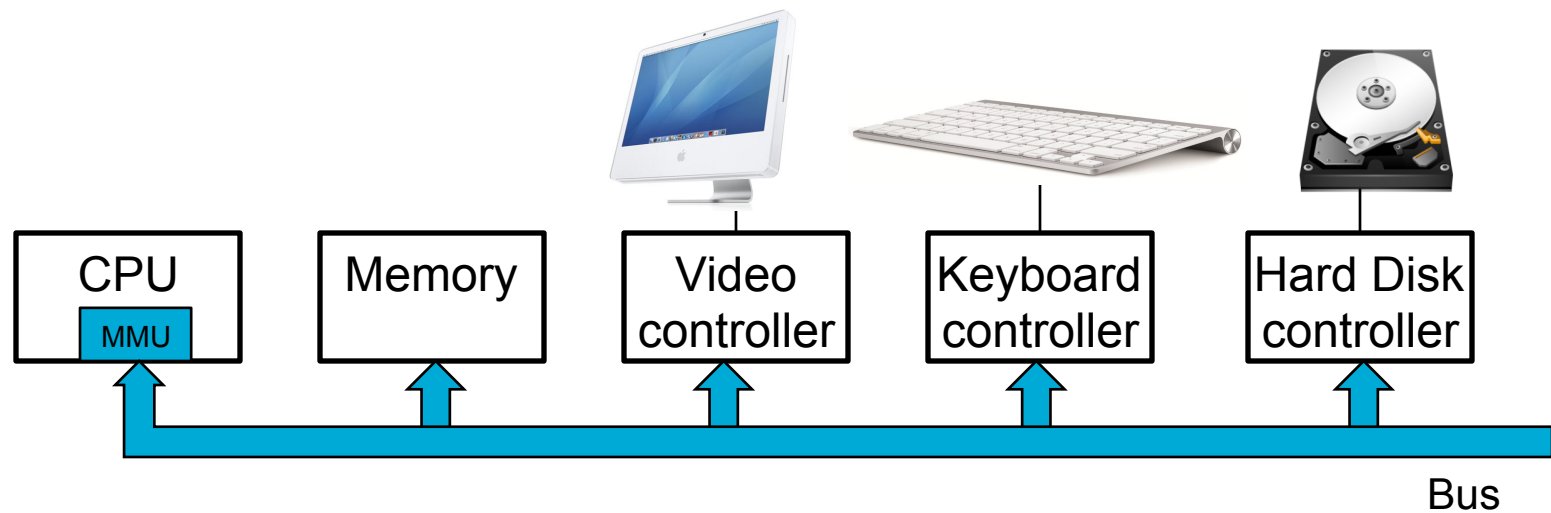
Announcements and reminders

- Google group for discussion – go to the course webpage and register Email:
- Project 1 is to be done on your own, all others are to be done in teams of 2
- Project 1 is out!
 - TA session on Wed. 6PM in TLab!
- Project competitions and awards

- And now a short quiz

Computer architecture and OS

- OS is intimately tied to the hardware it runs on
 - The OS design is impacted by it
 - The OS needs result on new architectural features
- Abstract model of a simple computer



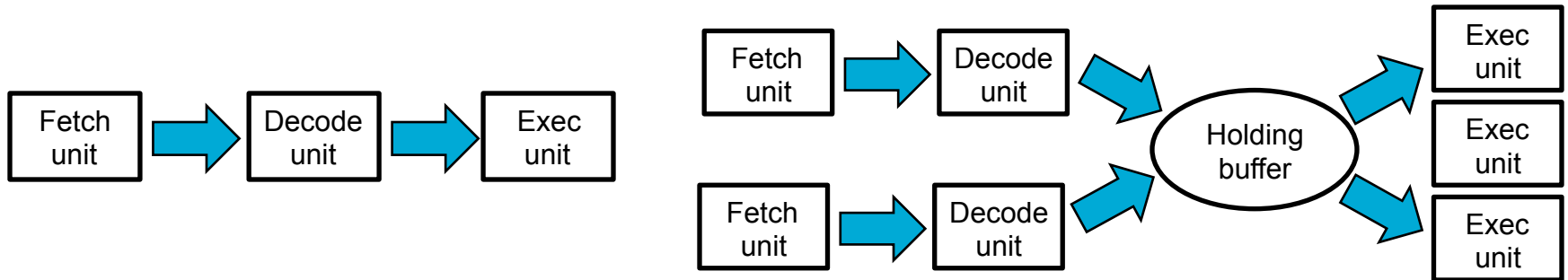
Processor

- The brain with a basic operation cycle
 - Fetch next instruction
 - Decode it to determine type & operands
 - Execute it
- ... and a specific set of instructions
 - E.g. combine ops (ADD), control flow, data movement
 - Architecture specific - Pentium != SPARC
- Since memory access is slow ... registers
 - General regs to hold variables & temp. results
 - Special regs such as Program Counter, Stack Pointer

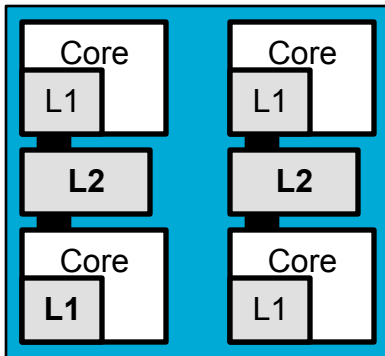


Processor ...

- This model is overly simplistic: pipeline architectures, superscalar, ...



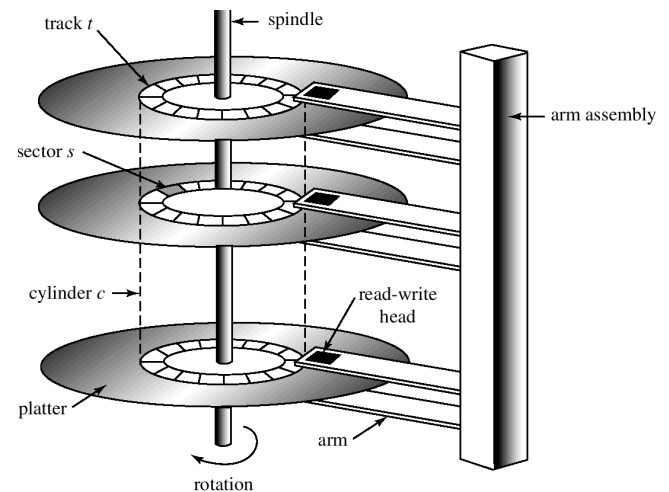
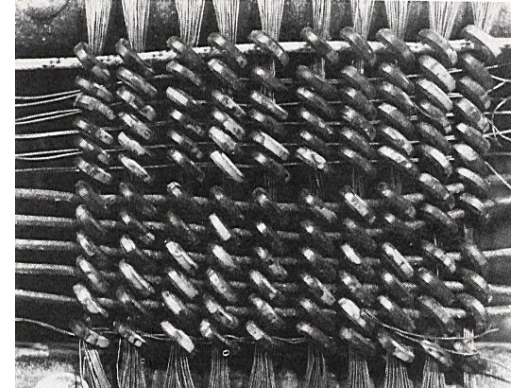
- Multithreading/Hyperthreading and multicore



Memory

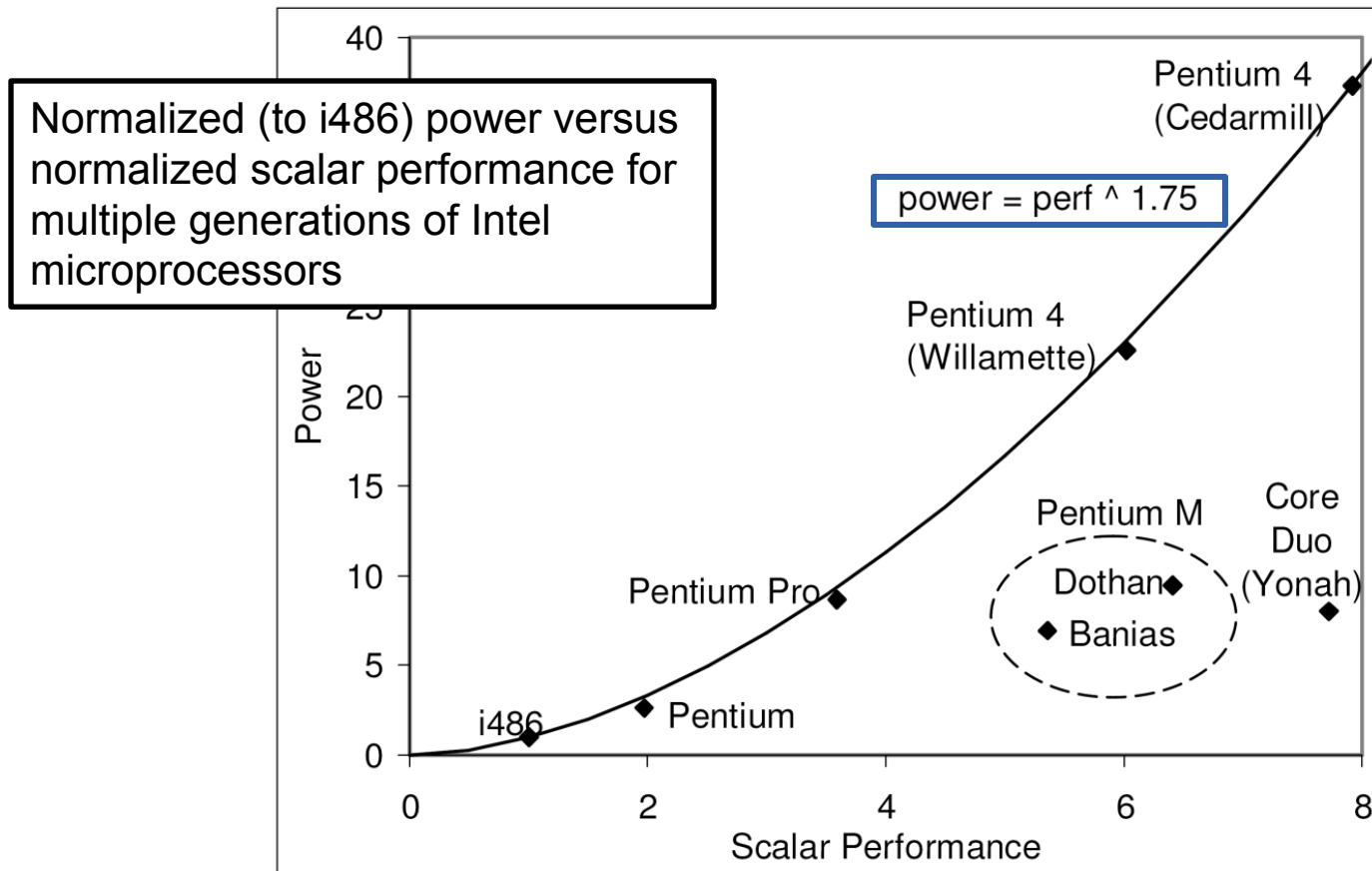
- Ideally – fast, large, cheap and persistent
- Reality – storage hierarchy
 - Registers
 - Internal to the CPU & just as fast
 - 32x32 in a 32 bit machine
 - Cache
 - Split into cache lines
 - If word needs is in cache, get in ~ 2 cycles
 - Main memory
 - Hard disk
 - Magnetic tape
 - *Coherency?*

First core-based memory: IBM 405
Alphabetical Accounting Machine



Architectural trends impact OS design ...

- Processing power
 - Doubling every 18 months (100x per decade)
 - but power is a serious issue



Architectural trends impact OS design ...

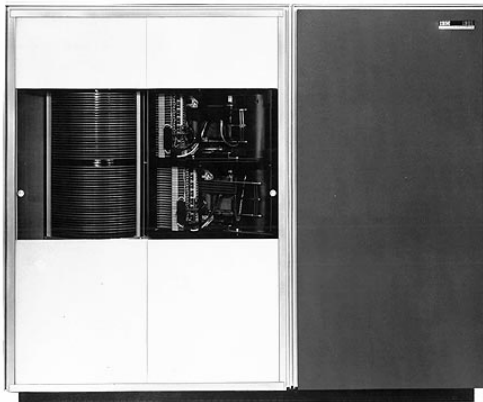
- Primary memory capacity

- Same and for the same reason

1980	64KB	\$405.00 (\$6,480/MB)
1990	8MB	\$851.00 (\$106/MB)
2000	64MB	\$99.89 (\$1.56/MB)
2009	4GB	\$39.99 (\$0.010/MB)*

- Disk capacity

- Double every 12 months (1000x per decade)



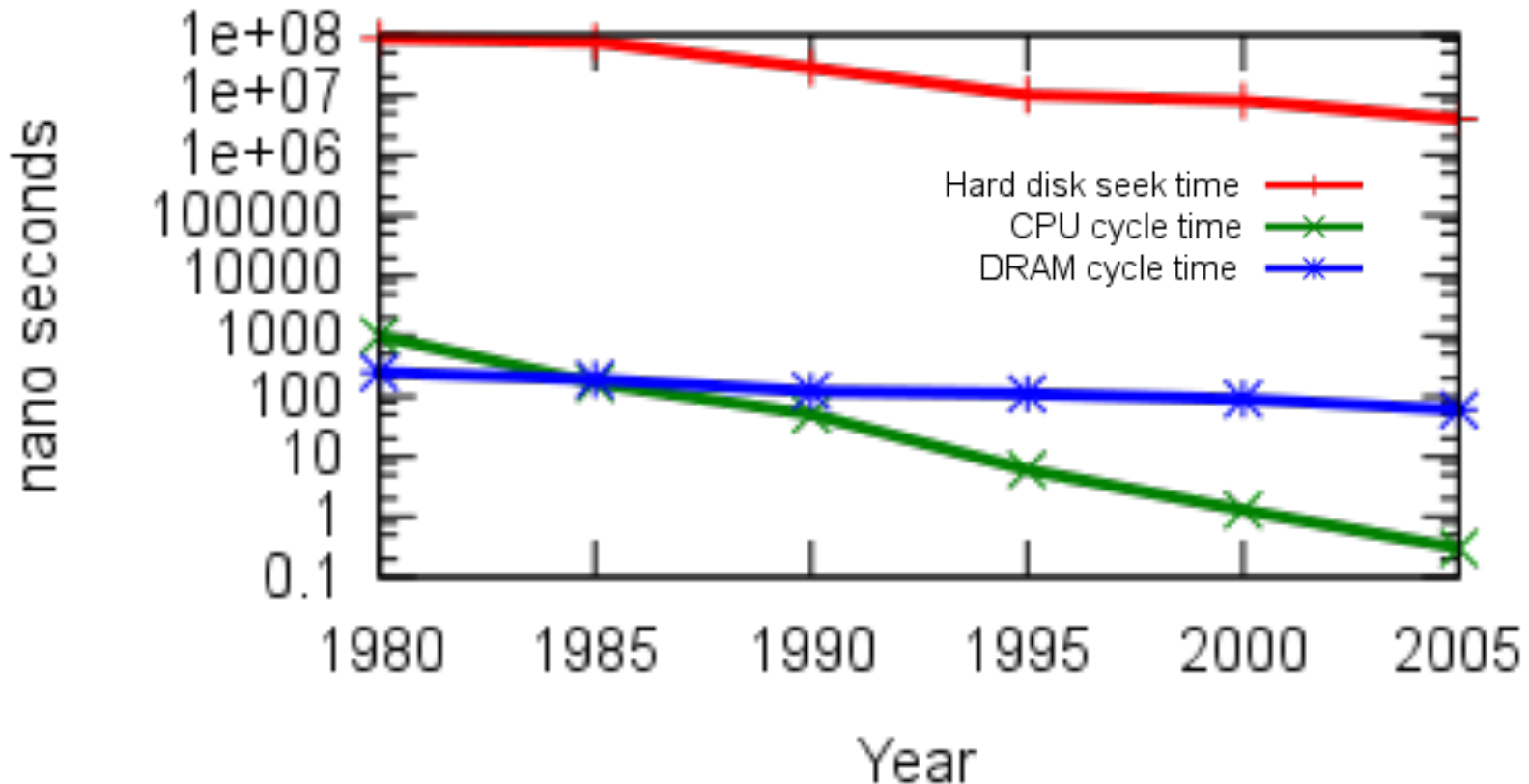
1961 IBM 1301
~26MB ~\$115,500



WD 2TB My Book
Essential ~ \$110

Architectural trends impact OS design ...

- Gap between CPU and I/O speeds



Relative speed of key components – an unbalanced system
FAWN project @ CMU

Architectural trends impact OS design

- Solid state storage (SSD)
 - 10-100k random IOs per second
 - 800 MB/s transfer rates
 - Costly, but quickly riding Moore's law
2011 Crucial 512GB SSD \$750
- Optical bandwidth today
 - Doubling every 9 months (Butter's law)
 - 50% improvement each year for home users (Nielsen's law)
 - Factor of 10,000 every decade
 - 10x as fast as disk capacity!
 - 100x as fast as processor performance!
- What are some of the implications of these trends?
 - E.g.: from mainframes to desktops to cloud computing



Crucial 512 GB m4
2.5-Inch Solid State
Drive SATA 6Gb/s

... and OS needs shape the architecture

- Architectural support can simplify/complicate OS tasks
 - E.g., early PC operating systems (DOS, MacOS) lacked support for virtual memory, partly because hardware lacked necessary hardware support
- Features were built primarily to support OS's:
 - Protected modes of execution (kernel vs. user)
 - Protected instructions
 - System calls (and software interrupts)
 - Memory protection
 - I/O control operations
 - Timer (clock) operation
 - Interrupts and exceptions
 - Synchronization instructions

OS protection

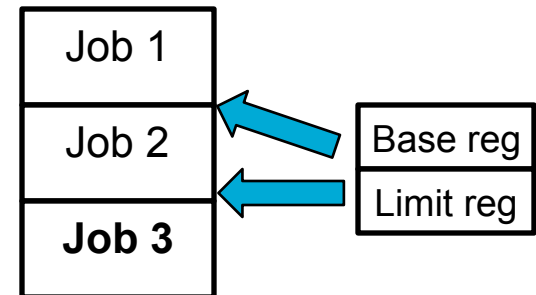
- Multiprogramming & timesharing are useful but
 - How to protect programs from each other & kernel from all?
 - How to handle relocation?
- Some instructions are restricted to the OS
 - e.g. Directly access I/O devices
 - e.g. Manipulate memory state management
- How does the CPU know if a protected instructions should be executed?
 - Architecture must support 2+ mode of operation
 - Mode is set by status bit in a protected register (PSW)
 - User programs execute in user mode, OS in kernel mode
- Protected instructions can only be executed in kernel mode

Crossing protection boundaries

- How can apps. do something privileged?
 - e.g. how do you write to disk if you can't do I/O?
- User programs must call an OS procedure
 - OS defines a sequence of system calls
 - How does the user to kernel-mode transition happen?
- There must be a system call instruction, which ...
 - Causes an exception which vector to a kernel handler
 - Passes a parameter indicating which syscall is
 - Saves caller's state so it can be restored – *Why?*
 - OS must verify caller's parameters – *Why?*
 - Must be a way to go back to user once done

Memory relocation

- OS must protect ...
 - user programs from each other
 - itself from user programs
- Simplest model – base + limit
 - Base (start) of program + limit registers
 - Also solves relocation problem
 - Cost 2 registers + cycle time incr
- More sophisticated alternatives
 - 2 base and 2 limit registers for text & data; allow sharing program text
 - Paging, segmentation, virtual memory



I/O

- I/O Device
 - Device + Controller (simpler I/F to OS; think SCSI)
 - Read sector x from disk y → (disk, cylinder, sector, head), ...
- How does the kernel start an I/O?
 - Special I/O instructions
 - Memory-mapped I/O
- How does it notice when the I/O is done?
 - Polling – *are we done yet?*
 - Interrupts – *let me know when you are done?*
- How does it exchange data with the I/O device?
 - Programmed I/O
 - Direct Memory Access (DMA)

OS control flow

- OSs are event driven
 - Once booted, all entry to kernel happens as result of an event (e.g. signal by an interrupt), which
 - Immediately stops current execution
 - Changes to kernel mode, event handler is called
- Kernel defines handlers for each event type
 - Specific types are defined by the architecture
 - e.g. timer event, I/O interrupt, system call trap

OS control flow ...

- Handling the interrupt
 - Push PC & PSW onto stack and switch to kernel mode
 - Device # is index in interrupt vector - get handler
 - Interrupt handler
 - Stores stack data
 - Handles interrupt
 - Returns to user program after restoring program state

Interrupts and exceptions

- Three main types of events: interrupts & exceptions
 - Exceptions/traps caused by SW executing instructions
 - E.g., a page fault
 - E.g., an attempted write to a read-only page
 - An expected exception is a “trap”, unexpected is a “fault”
 - Interrupts caused by HW devices
 - E.g., device finishes I/O
 - E.g., timer fires

Timers

- How can the OS retain control when a user program gets stuck in an infinite loop?
 - Use a hardware timer that generates a periodic interrupt
 - Before it transfers to a user program, the OS loads the timer with a time to interrupt (how long?)
 - When time's up, interrupt transfers control back to OS
 - OS decides which program to schedule next (which one?)
- Should the timer be privileged?
 - For reading or for writing?

Synchronization

- Issues with interrupts
 - May occur any time, causing code to execute that interferes with the interrupted code
 - OS must be able to synchronize concurrent processes
- Synchronization
 - Guarantee that short instruction sequences (e.g. read-modify-write) execute atomically
 - Two methods
 - Turn off interrupts, execute sequence, re-enable interrupts
 - Have special, complex atomic instructions – test-and-set

Management of concurrency & asynchronous events is the biggest difference bet/ systems-level & traditional application programming.

Summary

- This is far from over – new architectural features are still being introduced
 - Support for virtual machine monitors
 - Hardware transaction support
 - Support for security
 - ...
- Transistors are free so Intel/AMD/... need to find applications that require new hardware that you would want to buy ...