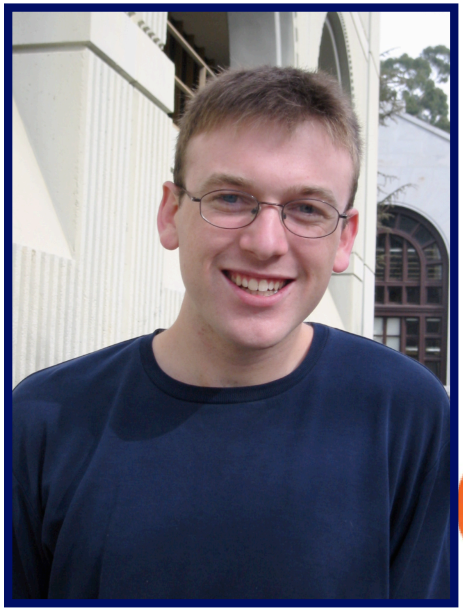
inst.eecs.berkeley.edu/~cs61c

CS61CL : Machine Structures

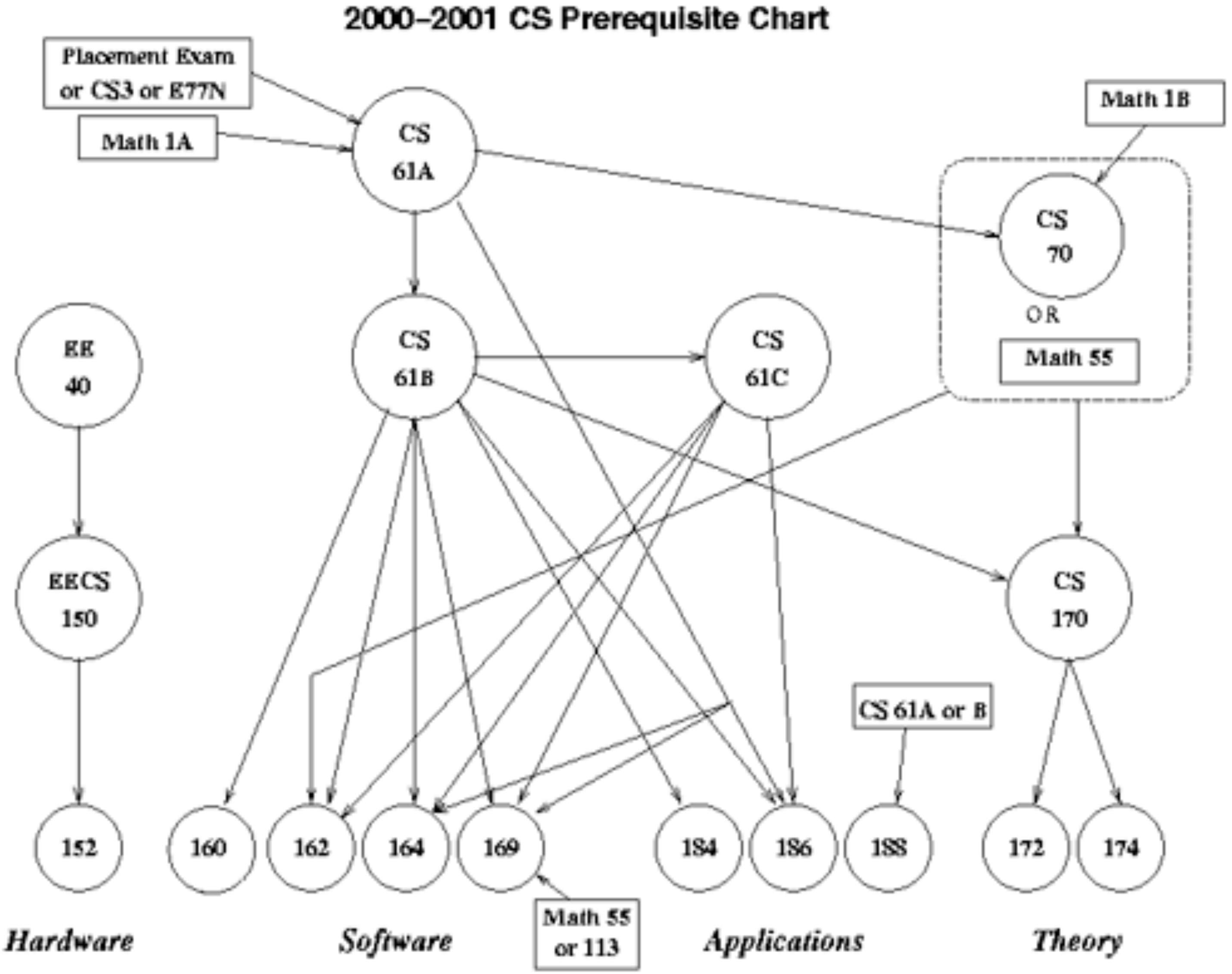
Lecture #1 – Introduction, C

2009-06-21



Jeremy Huddleston

Where does CS61C fit in?



http://hkn.eecs.berkeley.edu/student/cs-prereq-chart1.gif

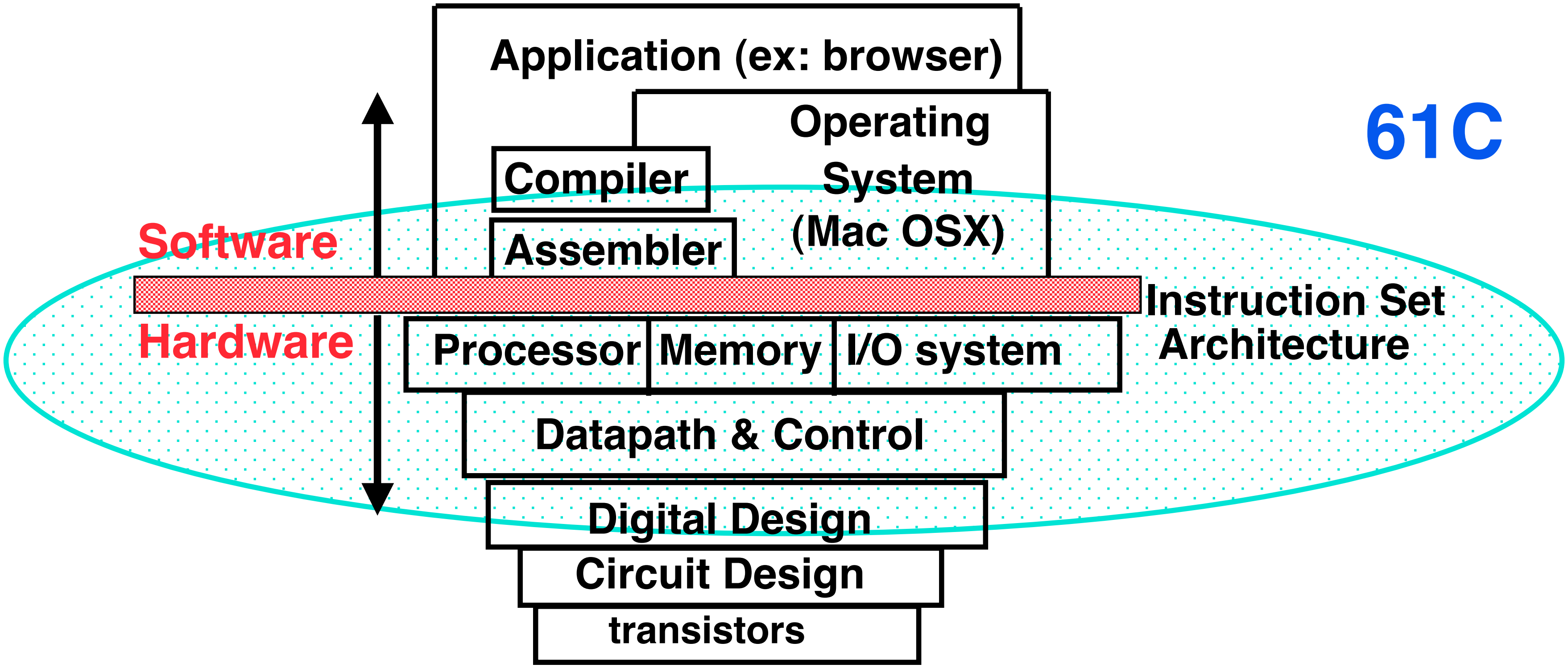
Are Computers Smart?

- To a programmer:
  - Very complex operations / functions:
    - `(map (lambda (x) (* x x)) '(1 2 3 4))`
  - Automatic memory management:
    - `List l = new List;`
  - “Basic” structures:
    - Integers, floats, characters, plus, minus, print commands

Are Computers Smart?

- In real life at the lowest level:
  - Only a handful of operations:
    - `{and, or, not}`
  - <u>No</u> automatic memory management.
  - Only 2 values:
    - {0, 1} or {low, high} or {off, on}

What are “Machine Structures”?



61C

Coordination of many
levels (layers) of abstraction

61C Levels of Representation

High Level Language Program (e.g., C)

Compiler

Assembly Language Program (e.g.,MIPS)

Assembler

Machine Language Program (MIPS)

Machine Interpretation

Hardware Architecture Description (e.g., block diagrams)

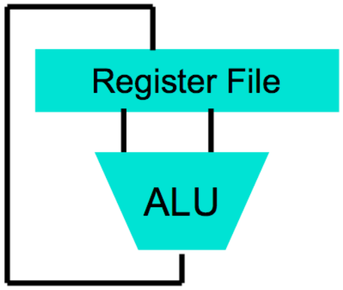
Architecture Implementation

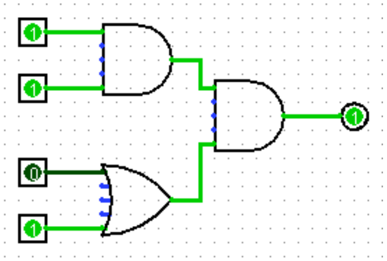
Logic Circuit Description (Circuit Schematic Diagrams)

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

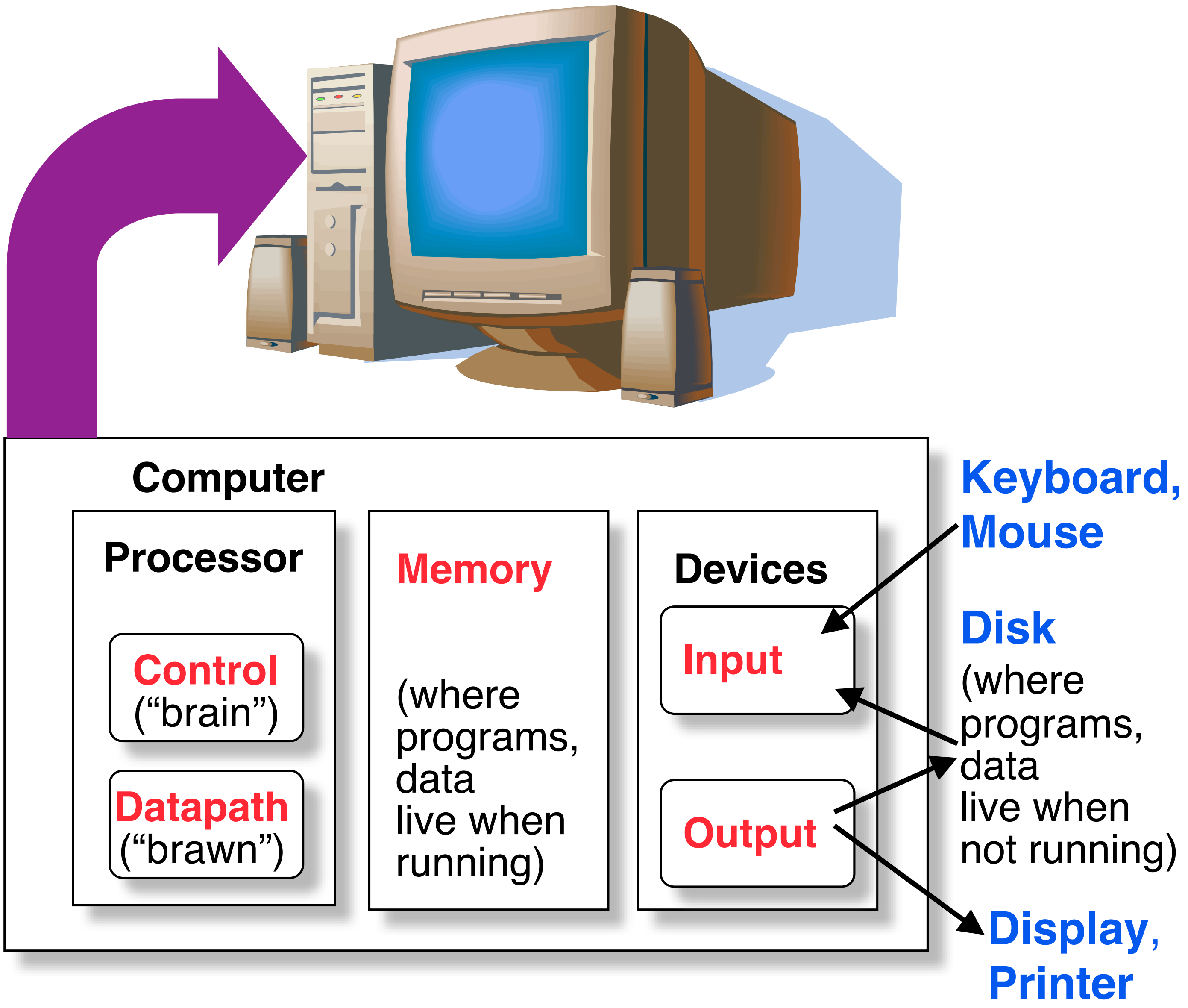
```
lw   $t0, 0($2)
lw   $t1, 4($2)
sw   $t1, 0($2)
sw   $t0, 4($2)
```

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```





Anatomy: 5 components of any Computer



Computer

Processor

Control
("brain")

Datapath
("brawn")

Memory

(where programs, data live when running)

Devices

Input

Output

Keyboard, Mouse

Disk
(where programs, data live when not running)

Display, Printer

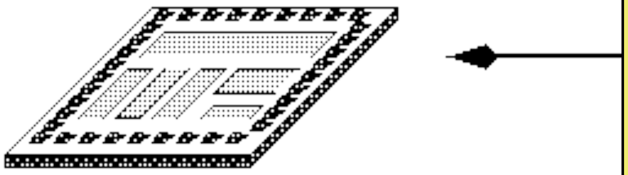
Overview of Physical Implementations

The hardware out of which we make systems.

- Integrated Circuits (ICs)
  - Combinational logic circuits, memory elements, analog interfaces.
- Printed Circuits (PC) boards
  - substrate for ICs and interconnection, distribution of CLK, Vdd, and GND signals, heat dissipation.
- Power Supplies
  - Converts line AC voltage to regulated DC low voltage levels.
- Chassis (rack, card case, ...)
  - holds boards, power supply, provides physical interface to user or other systems.
- Connectors and Cables.

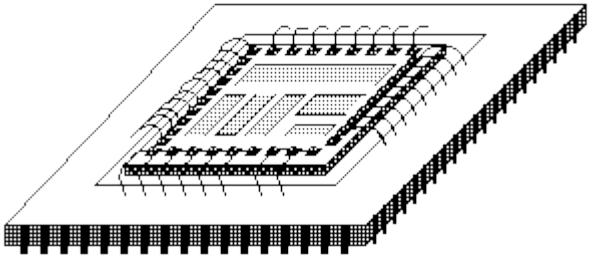
Integrated Circuits (2009 state-of-the-art)

Bare Die



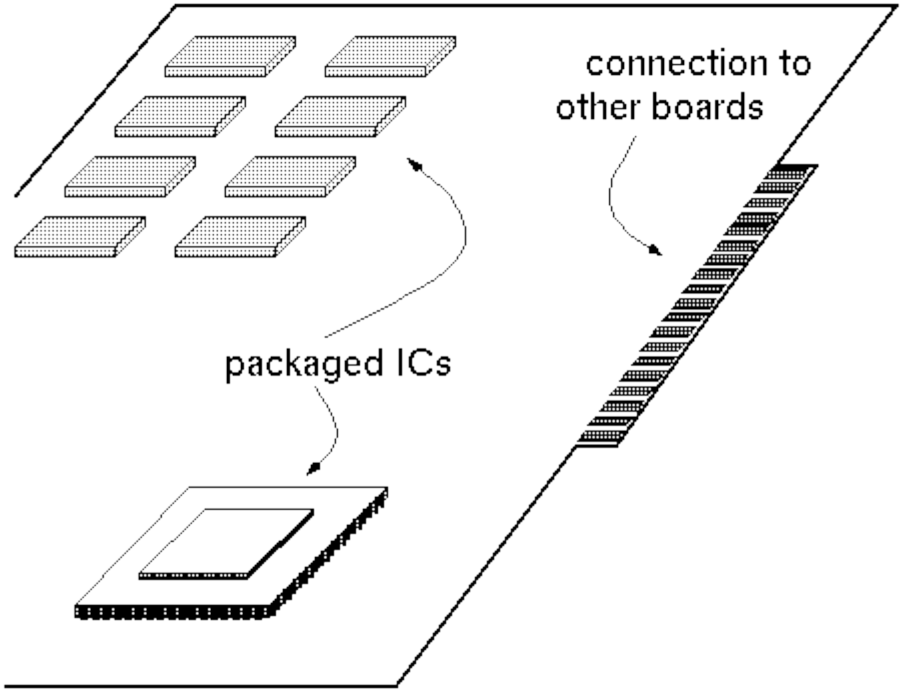
- Primarily Crystalline Silicon
- 1mm - 25mm on a side
- 2009 feature size ~ 45 nm = 45 x 10^{-9} m (red light has a wavelength of ~700nm)
- 500 - 2000M transistors
- 2 - 864 cores
- 3 - 10 conductive layers
- "CMOS" (complementary metal oxide semiconductor) - most common.

Chip in Package



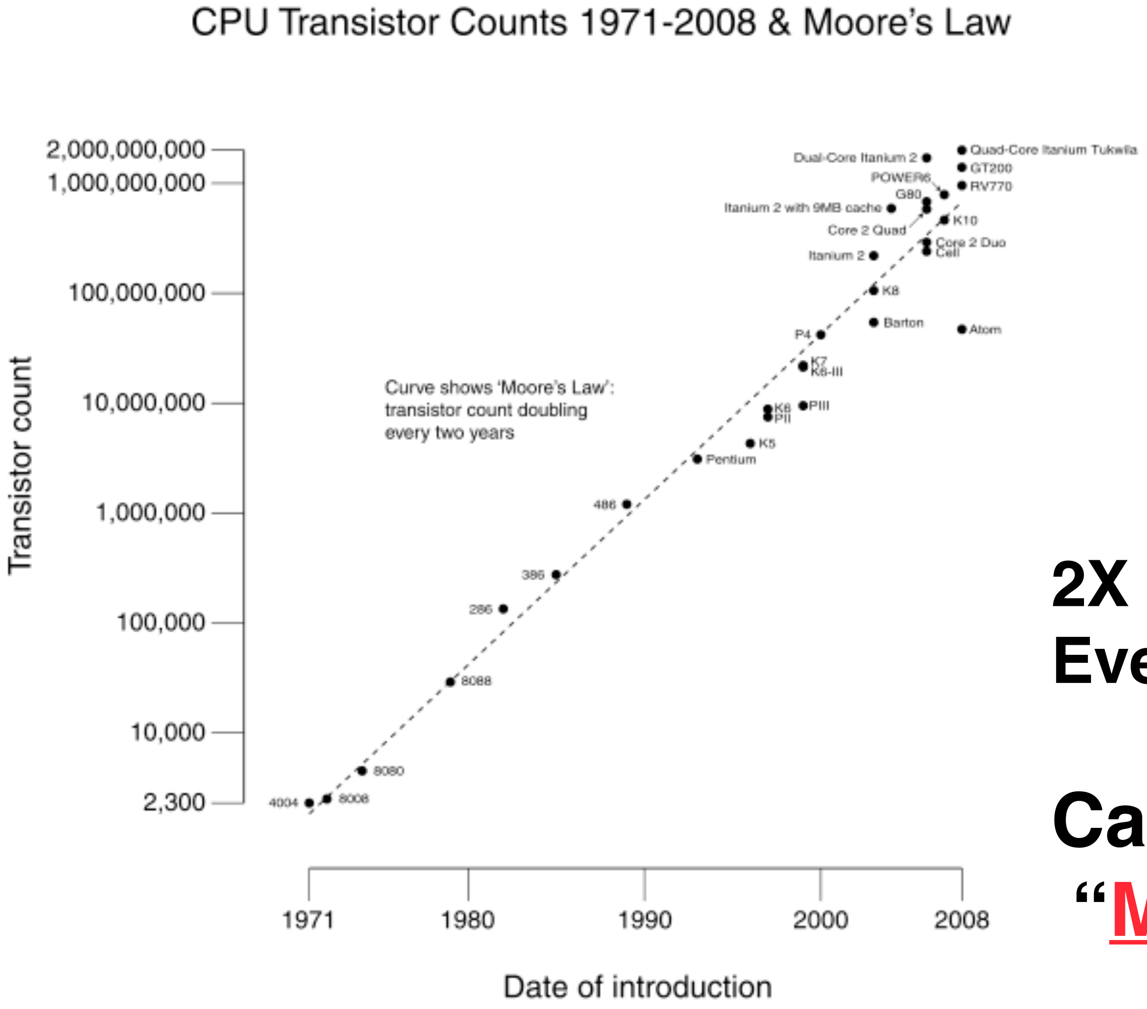
- Package provides:
  - spreading of chip-level signal paths to board-level
  - heat dissipation.
- Ceramic or plastic with gold wires.

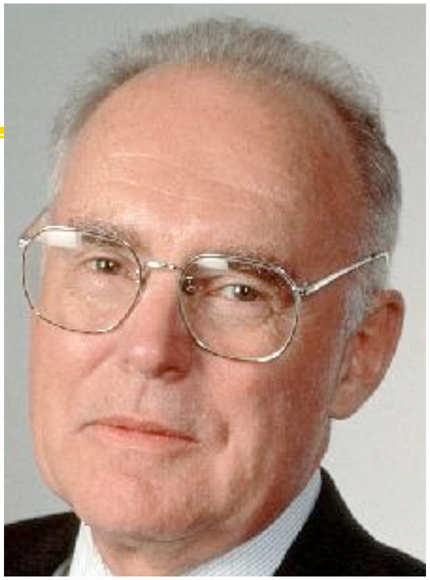
Printed Circuit Boards



- fiberglass or ceramic
- 1-20 conductive layers
- 1-20 in on a side
- IC packages are soldered down.
- Provides:
    - Mechanical support
    - Distribution of power and heat.

Technology Trends: Microprocessor Complexity


CPU Transistor Counts 1971-2008 & Moore's Law



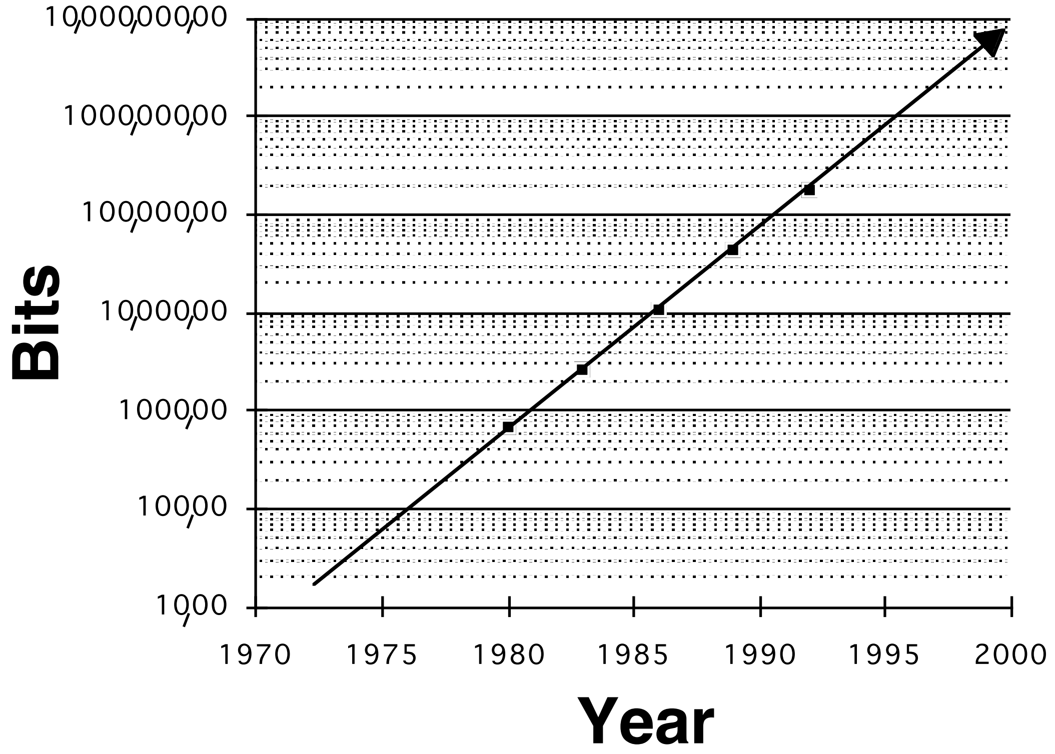
Gordon Moore
Intel Cofounder
B.S. Cal 1950!

2X Transistors / Chip Every 1.5 - 2 years

Called "Moore's Law"

Technology Trends: Memory Capacity

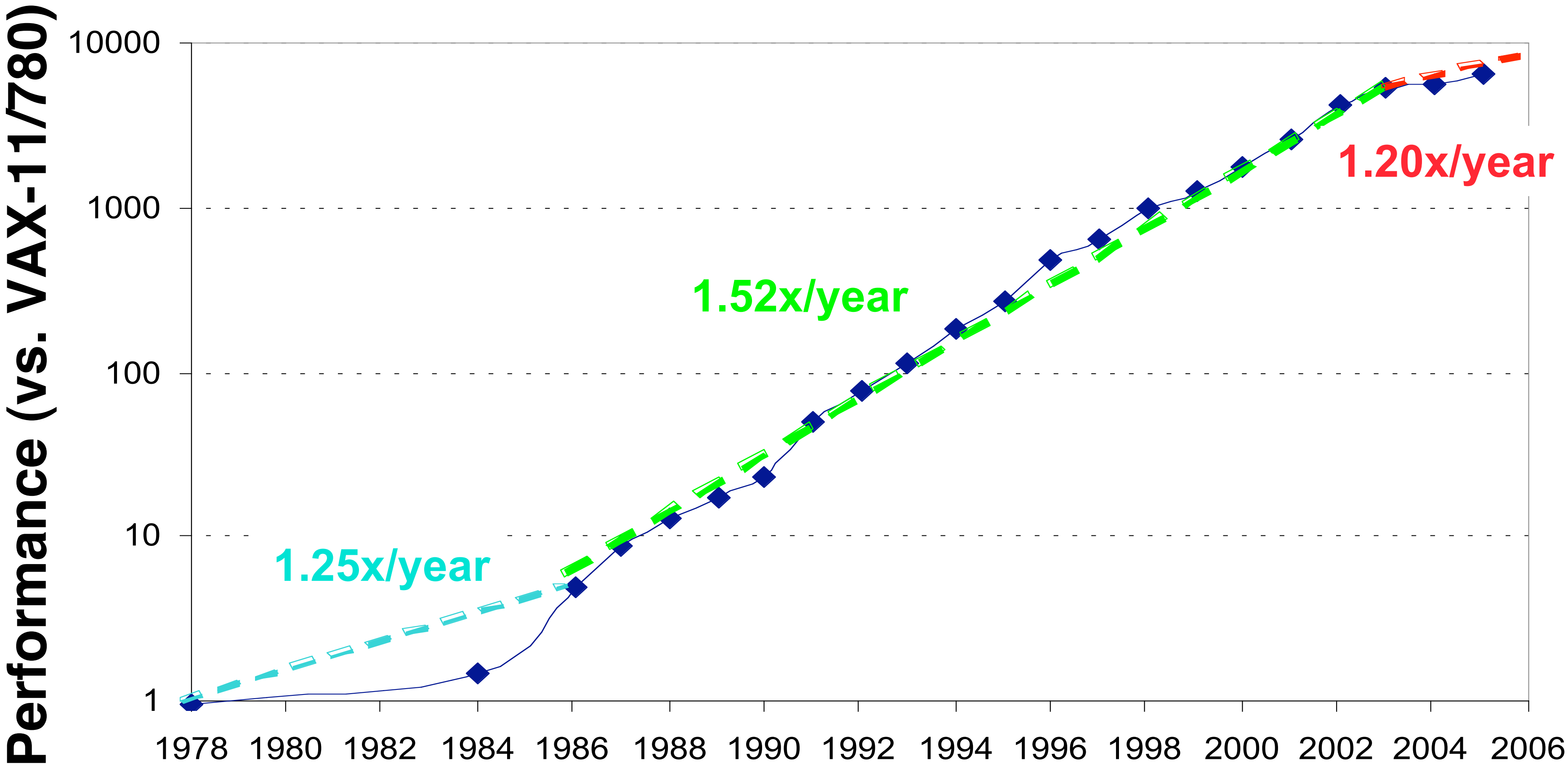
1950: Alan Turing predicted ~1G by 2000



year	size (Mbi)
1986	1
1988	4
1991	16
1995	64
1997	128
1999	256
2001	512
2003	1024 (1 Gbi)
2005	2048 (2 Gbi)
2007	4096 (4 Gbi)
2009	8192 (8 Gbi)

- Now 1.4X/yr, or 2X every 2 years.
- Over 10,000 X since 1980!

Technology Trends: Uniprocessor Performance (SPECint)



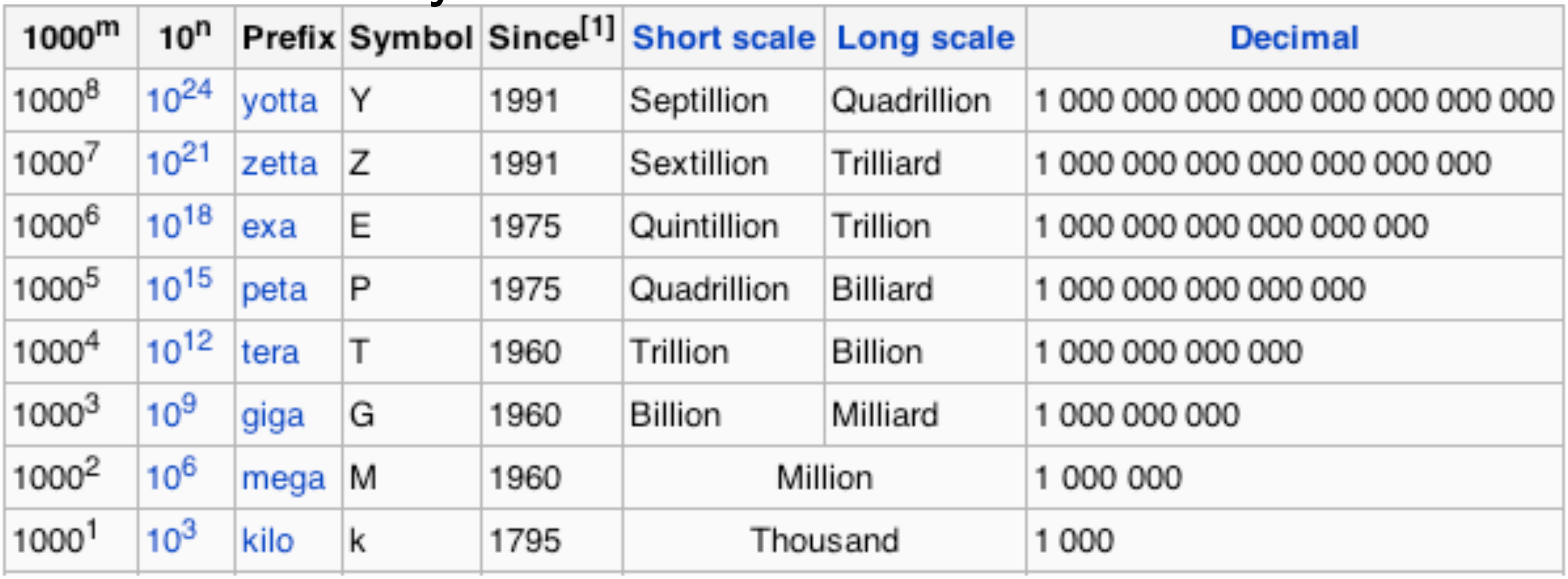
- VAX : 1.25x/year 1978 to 1986
- RISC + x86: 1.52x/year 1986 to 2002
- RISC + x86: 1.20x/year 2002 to present

Computer Technology - Dramatic Change!

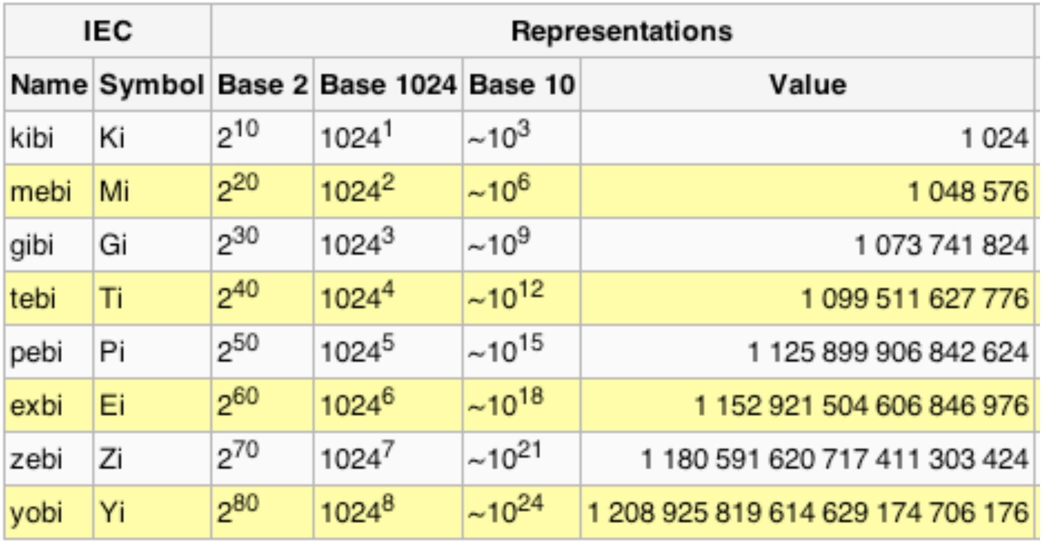
- Memory
  - DRAM capacity: 2x / 2 years (since ‘96); 64x size improvement in last decade.
- Processor
  - Speed 2x / 1.5 years (since ‘85); [slowing!] 100X performance in last decade.
- Disk
  - Capacity: 1.8x / 1 year (since ‘97) 250X size in last decade.

SI Prefixes

Le **S**ystème **I**nternational d'Unités



1000^m	10^n	Prefix	Symbol	Since[1]	Short scale	Long scale	Decimal
1000^8	10^{24}	yotta	Y	1991	Septillion	Quadrillion	1 000 000 000 000 000 000 000 000
1000^7	10^{21}	zetta	Z	1991	Sextillion	Trilliard	1 000 000 000 000 000 000 000
1000^6	10^{18}	exa	E	1975	Quintillion	Trillion	1 000 000 000 000 000 000
1000^5	10^{15}	peta	P	1975	Quadrillion	Billiard	1 000 000 000 000 000
1000^4	10^{12}	tera	T	1960	Trillion	Billion	1 000 000 000 000
1000^3	10^9	giga	G	1960	Billion	Milliard	1 000 000 000
1000^2	10^6	mega	M	1960	Million		1 000 000
1000^1	10^3	kilo	k	1795	Thousand		1 000



IEC		Representations			
Name	Symbol	Base 2	Base 1024	Base 10	Value
kibi	Ki	2^{10}	1024^1	$\sim10^3$	1 024
mebi	Mi	2^{20}	1024^2	$\sim10^6$	1 048 576
gibi	Gi	2^{30}	1024^3	$\sim10^9$	1 073 741 824
tebi	Ti	2^{40}	1024^4	$\sim10^{12}$	1 099 511 627 776
pebi	Pi	2^{50}	1024^5	$\sim10^{15}$	1 125 899 906 842 624
exbi	Ei	2^{60}	1024^6	$\sim10^{18}$	1 152 921 504 606 846 976
zebi	Zi	2^{70}	1024^7	$\sim10^{21}$	1 180 591 620 717 411 303 424
yobi	Yi	2^{80}	1024^8	$\sim10^{24}$	1 208 925 819 614 629 174 706 176

Computer Technology - Dramatic Change!

- State-of-the-art PC when you graduate: (at least…)
  - Processor clock speed: 16 x 4000 MegaHz (16 x 4.0 GigaHz)
  - Memory capacity: 327680 MebiBytes (320 GibiBytes)
  - Disk capacity: 6000 GigaBytes (6 TeraBytes)
- Mega ⇒ Giga ⇒ Tera ⇒ Peta ⇒ Exa ⇒ …

CS61CL: So, what's in it for me?

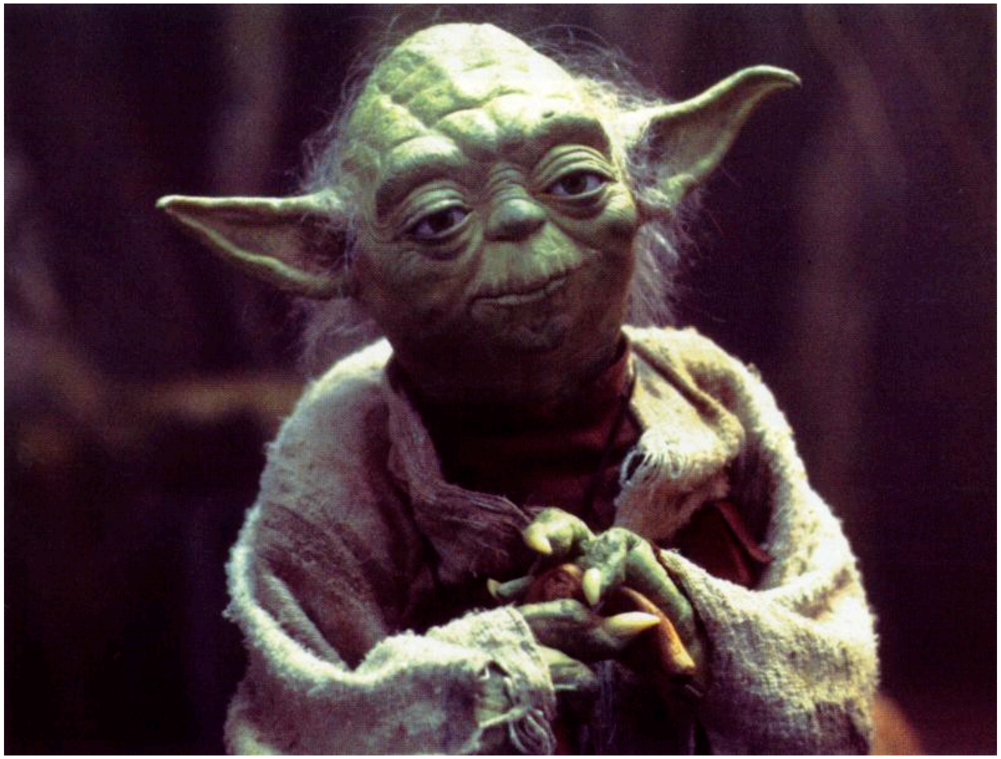
- Learn some of the big ideas in CS & Engineering:
  - Principle of abstraction
    - Used to build systems as layers
  - 5 Classic components of a Computer
  - Data can be anything
    - Integers, floating point, characters, …
    - A program determines what it is
    - Stored program concept: instructions just data
  - Principle of Locality
    - Exploited via a memory hierarchy (cache)
  - Greater performance by exploiting parallelism
  - Compilation v. interpretation through system layers
  - Principles / Pitfalls of Performance Measurement

Others Skills learned in 61C

- Learning C
  - If you know one, you should be able to learn another programming language largely on your own
  - If you know C++ or Java, it should be easy to pick up their ancestor, C
- Assembly Language Programming
  - This is a skill you will pick up, as a side effect of understanding the Big Ideas
- Hardware design
  - We'll learn just the basics of hardware design
  - CS 150, 152 teach this in more detail

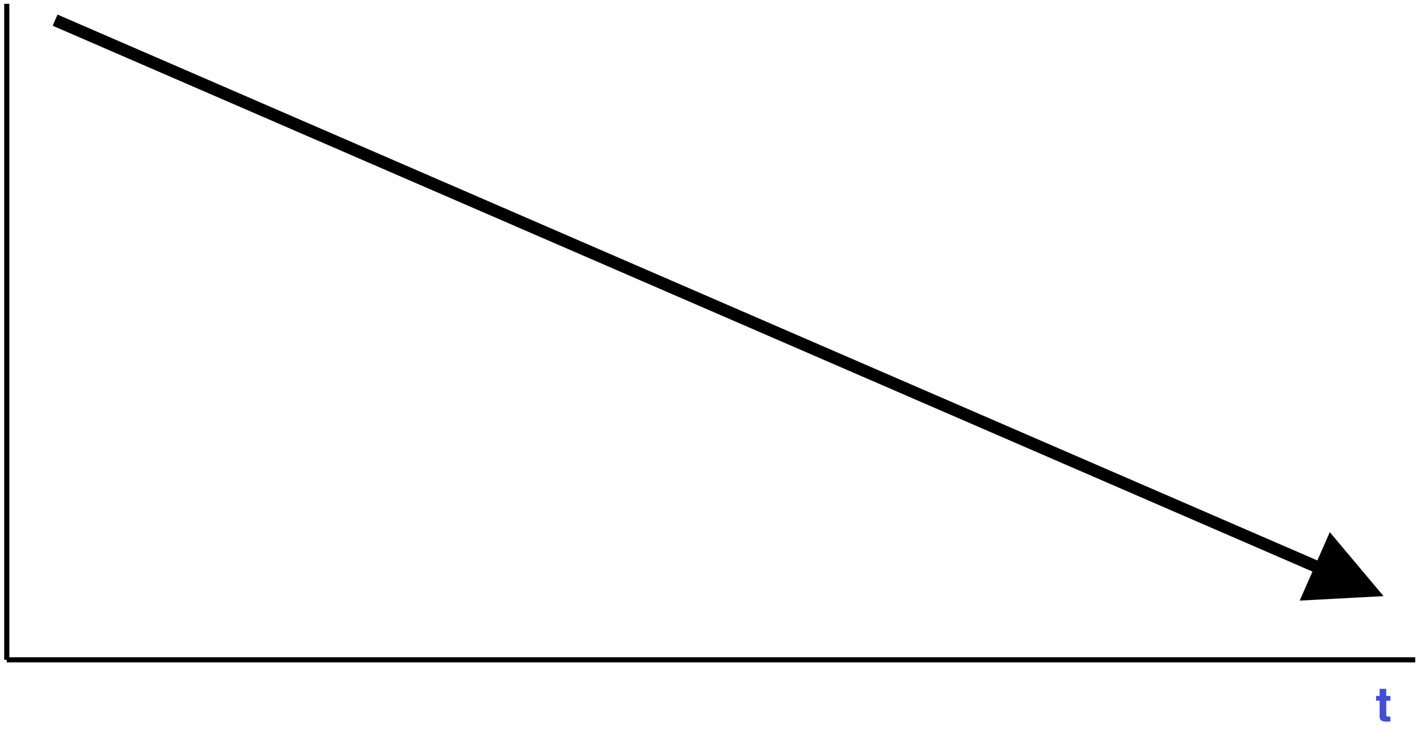
Yoda says…

"Always in motion is the future…"



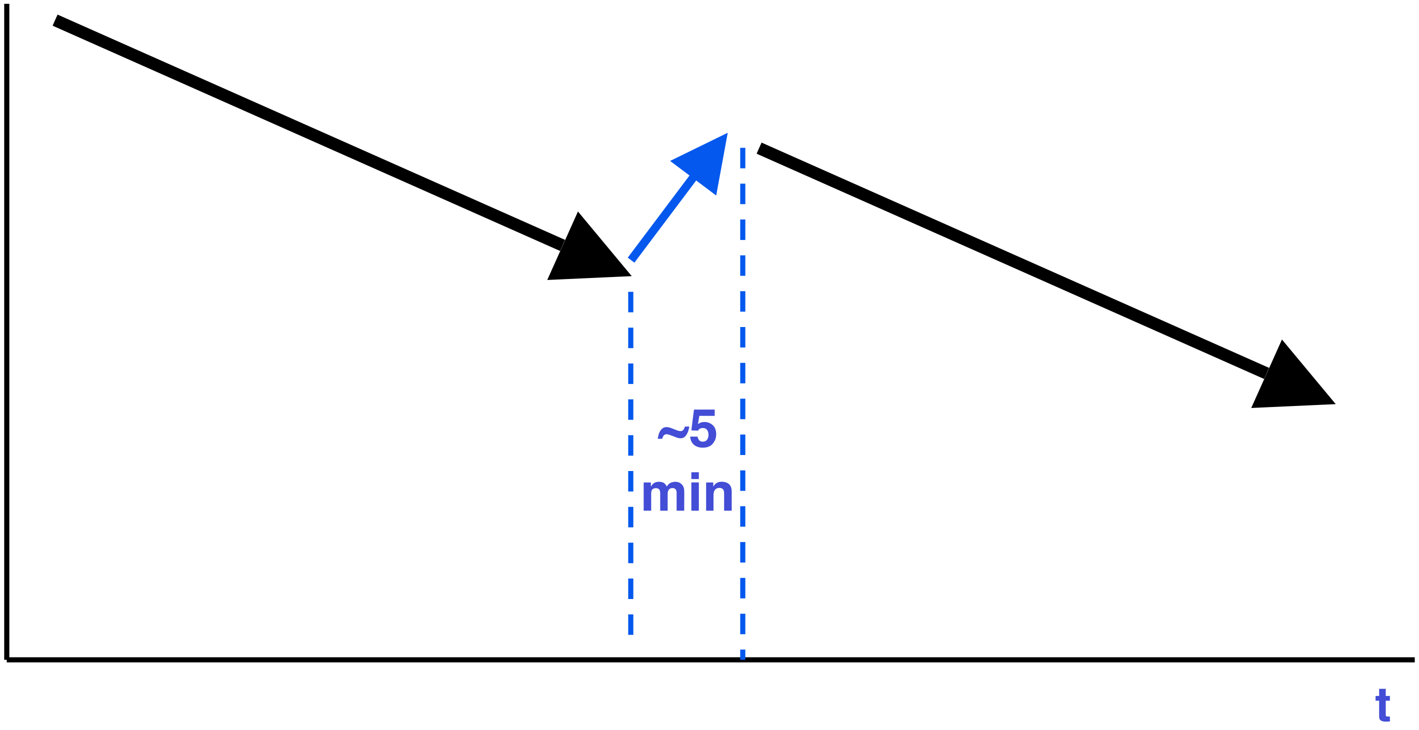
Our schedule may change slightly depending on some factors. This includes lectures, assignments & labs…

What is this?



Attention over time!

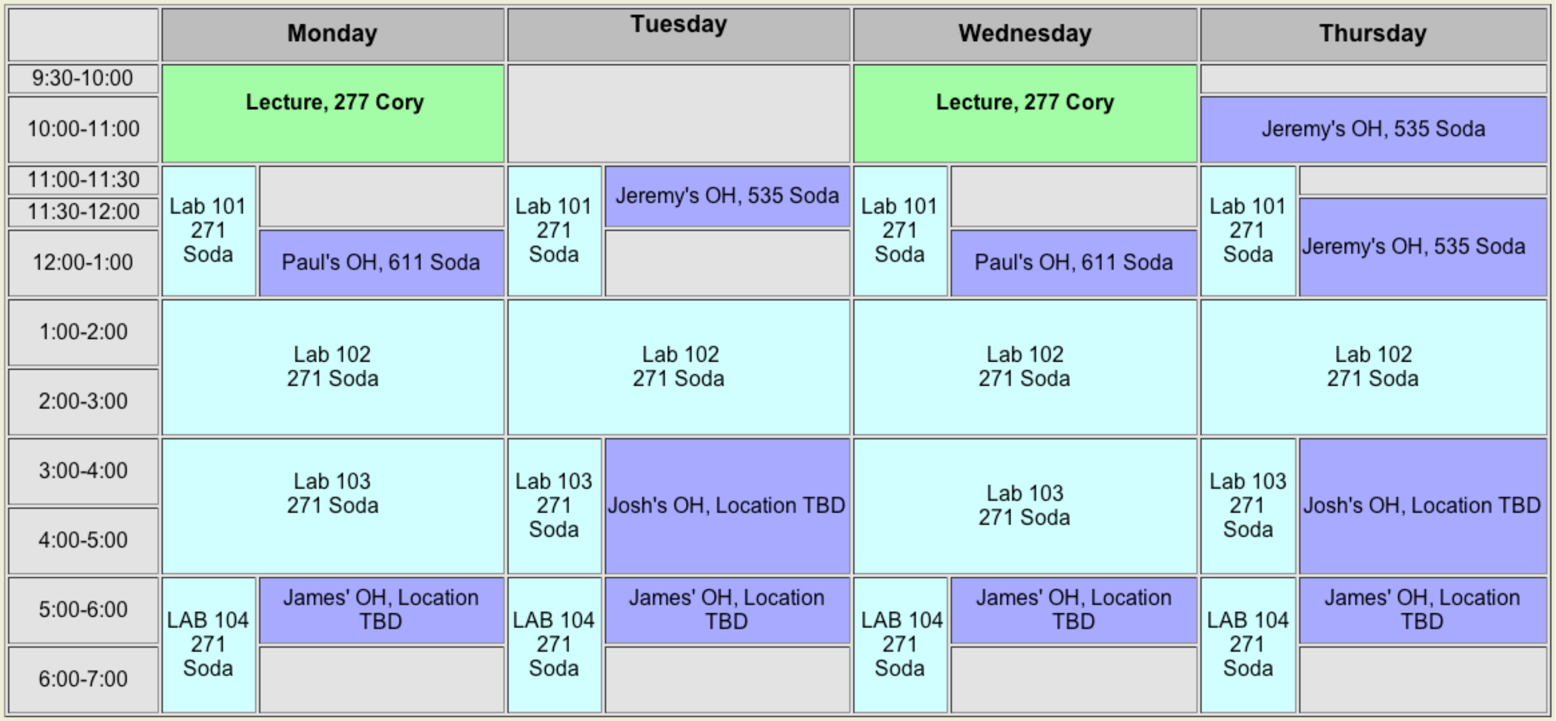
What is this?!



Attention over time!

Lab-based Model

- UC-WISE
- Lecture on M,W only!
- Labs every day
  - Discussion replaced with a 2hr lab at the same time



	Monday		Tuesday		Wednesday		Thursday	
9:30-10:00	Lecture, 277 Cory				Lecture, 277 Cory			
10:00-11:00							Jeremy's OH, 535 Soda	
11:00-11:30	Lab 101 271 Soda		Lab 101 271 Soda	Jeremy's OH, 535 Soda	Lab 101 271 Soda		Lab 101 271 Soda	
11:30-12:00								Jeremy's OH, 535 Soda
12:00-1:00		Paul's OH, 611 Soda				Paul's OH, 611 Soda		
1:00-2:00	Lab 102 271 Soda		Lab 102 271 Soda		Lab 102 271 Soda		Lab 102 271 Soda	
2:00-3:00								
3:00-4:00	Lab 103 271 Soda		Lab 103 271 Soda	Josh's OH, Location TBD	Lab 103 271 Soda		Lab 103 271 Soda	Josh's OH, Location TBD
4:00-5:00								
5:00-6:00	LAB 104 271 Soda	James' OH, Location TBD	LAB 104 271 Soda	James' OH, Location TBD	LAB 104 271 Soda	James' OH, Location TBD	LAB 104 271 Soda	James' OH, Location TBD
6:00-7:00								

Peer Instruction and Just-in-time-learning

- Interact with other students in lab
- Fill out brainstorms in lab
  - Graded for effort, not correctness…
  - Review other students' responses
- Read textbook
  - Reduces examples have to do in class
  - Get more from lecture (also good advice)

Weekly Schedule

- Weekly schedule is on the website
- Office Hours are happening this week
- This week
  - Jeremy’s Th OH Canceled
  - Jeremy has OH Tu and W 11:30-1

Your final grade

- Grading (could change before 1st midterm)
  - 90 = 9% Labs (3 pts per 31-9)
  - 140 = 14% Homework (20 points per 8-1)
  - 320 = 32% Projects (80 points per 4)
  - 150 = 15% Midterm *[can be clobbered]*
  - 300 = 30% Final
  - + Extra credit for EPA. What's EPA?

Extra Credit: EPA!

- **Effort**
  - Attending Dan's and TA's office hours, completing all assignments, turning in HW0
- **Participation**
  - Attending lecture and voting using the PRS system
  - Asking great questions in discussion and lecture and making it more interactive
- **Altruism**
  - Helping others in lab or on the newsgroup
- EPA! extra credit points have the potential to bump students up to the next grade level! (but actual EPA! scores are internal)

Your final grade

- Grade distributions
  - Perfect score is 1 kilopoint.
  - Course average GPA ~ 2.9
  - 25% As, 60% Bs, 18% Cs, 2% D,F
  - No F will be given if all-but-one {hw, lab}, all projects submitted and all exams taken
  - We'll "ooch" grades up but never down

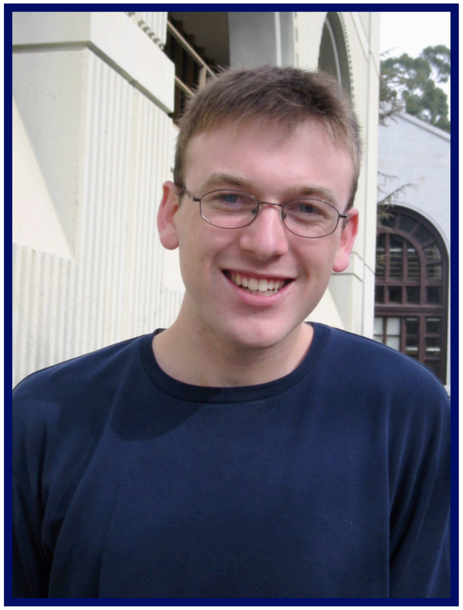
Course Problems…Cheating

- What is cheating?
  - Studying together in groups is encouraged.
  - Turned-in work must be ***completely*** your own.
  - Common examples: running out of time on a assignment and then pick up output, person asks to borrow solution "just to take a look", copying an exam question, …
  - You're not allowed to work on homework/projects/exams with anyone (other than ask Qs walking out of lecture)
  - Both "giver" and "receiver" are equally culpable
- Caught Cheating points: 0 EPA, negative points for that assignment / project / exam (e.g., if it's worth 10 pts, you get -10) In most cases, F in the course.
- Amnesty: If you turn yourself in, 0 for that assignment.
- Every offense will be referred to the Office of Student Judicial Affairs.

`www.eecs.berkeley.edu/Policies/acad.dis.shtml`

My goal as an instructor

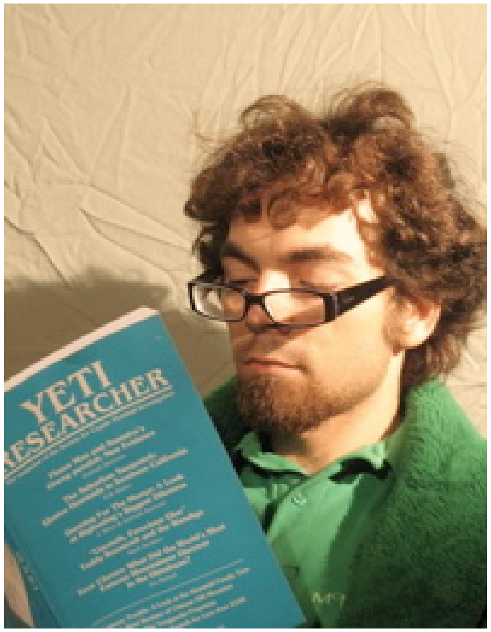
- To make your experience in CS61CL as enjoyable & informative as possible
  - Approachability, share my enthusiasm
  - Fun, challenging projects & HW
  - Pro-student policies (exam clobbering)
- To maintain Cal & EECS standards of excellence
  - Your projects & exams will be just as rigorous as every year. Overall : B- avg
- To be an HKN "7.0" man
  - Please give me feedback so I improve! Why am I not 7.0 for you? I will listen!!
  - Help me help you!



Meet Your TAs

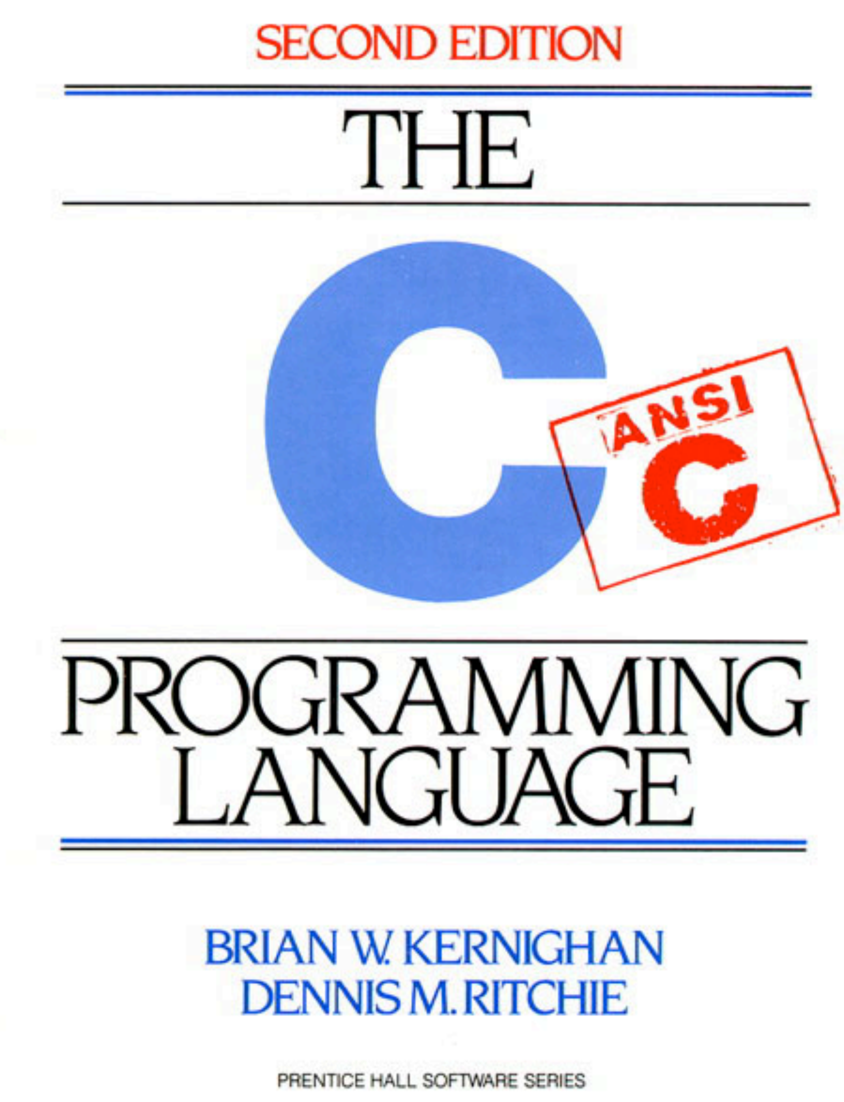
James
Tu

Paul
Pearce



Josh
Hug

Introduction to C



SECOND EDITION

THE

C

ANSI C

PROGRAMMING LANGUAGE

BRIAN W. KERNIGHAN
DENNIS M. RITCHIE

PRENTICE HALL SOFTWARE SERIES

Has there been an update to ANSI C?

- Yes! It's called the "C99" or "C9x" std
  - You need "`gcc -std=c99`" to compile
- References
  - `http://en.wikipedia.org/wiki/C99`
  - `http://home.tiscalinet.ch/t_wolf/tw/c/c9x_changes.html`
- Highlights
  - Declarations anywhere, like Java (#15)
  - Java-like `//` comments (to end of line) (#10)
  - Variable-length non-global arrays (#33)
  - `<inttypes.h>`: explicit integer types (#38)
  - `<stdbool.h>` for boolean logic def's (#35)
  - `restrict` and `inline` keywords for optimization (#30-32)

Disclaimer

- **Important**: You will not learn how to fully code in C in these lectures! You'll still need your C reference for this course.
  - K&R is a must-have reference
    - Check online for more sources
  - "JAVA in a Nutshell," O'Reilly.
    - Chapter 2, "How Java Differs from C"
  - Brian Harvey's course notes
    - On class website

Compilation : Overview

C *compilers* take C and convert it into an architecture specific machine code (string of 1s and 0s).

- Unlike Java which converts to architecture independent bytecode.
- Unlike most Scheme, Python, Ruby environments which interpret the code.
- These differ mainly in when your program is converted to machine instructions.
- For C, generally a 2 part process of compiling .c files to .o (object) files, then linking the object files into executables

Compilation : Advantages

- **Great run-time performance**: generally much faster than interpreted languages or Java for comparable code (because it optimizes for a given architecture)
- **OK compilation time**: enhancements in compilation procedure (`Makefiles`) allow only modified files to be recompiled

Compilation : Disadvantages

- All compiled files (including the executable) are architecture specific, depending on *both* the CPU type and the operating system.
- Executable must be rebuilt on each new system.
  - Called "porting your code" to a new architecture.
- The "change→compile→run [repeat]" iteration cycle is slow

C Syntax: `main`

- To get the main function to accept arguments, use this:

```
int main (int argc, char *argv[])
```

- What does this mean?
  - `argc` will contain the number of strings on the command line (the executable counts as one, plus one for each argument). Here `argc` is 2:

```
$ sort myFile
```

  - `argv` is a pointer to an array containing the arguments as strings (more on pointers later).

C Syntax: Variable Declarations

- Very similar to Java, but with a few minor but important differences
- All variable declarations must go before they are used (at the beginning of the block)*
- A variable may be initialized in its declaration; if not, it holds garbage!
- Examples of declarations:
    - correct:

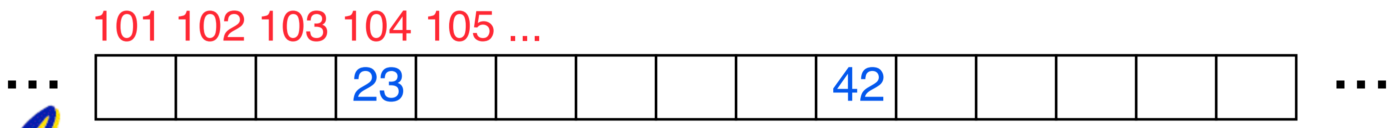
      ```
      {
          int a = 0, b = 10;
                ...
      ```

    - Incorrect:* `for (int i = 0; i < 10; i++)`

*C99 overcomes these limitations

Address vs. Value

- Consider memory to be a single huge array:
    - Each cell of the array has an address associated with it.
    - Each cell also stores some value.
- Don’t confuse the address referring to a memory location with the value stored in that location.

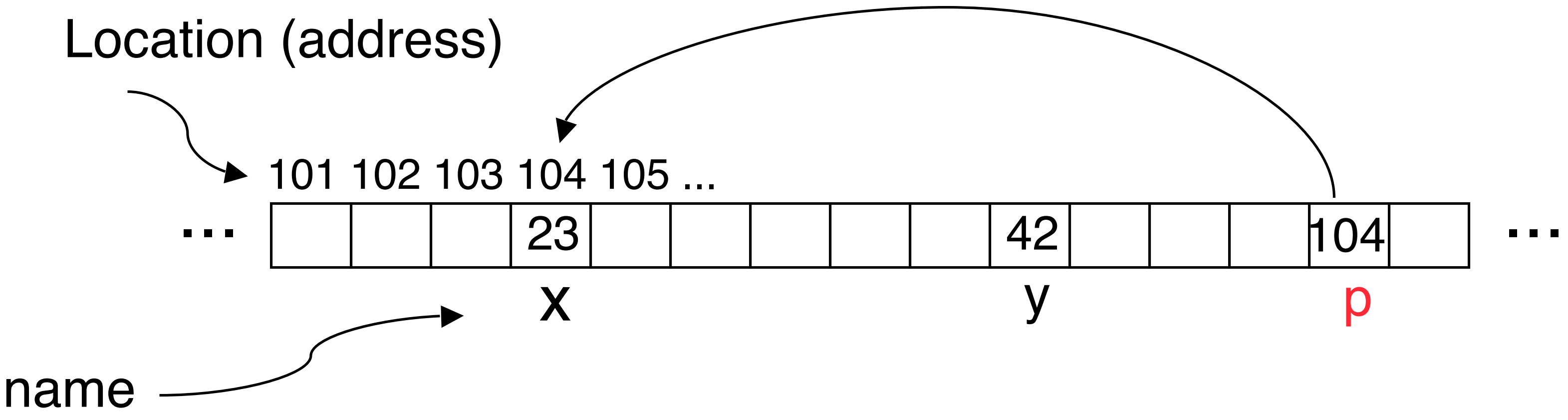


101 102 103 104 105 ...

... 23 42 ...

Pointers

- An address refers to a particular memory location. In other words, it <u>points</u> to a memory location.
- **Pointer**: A variable that contains the <u>address</u> of a variable.

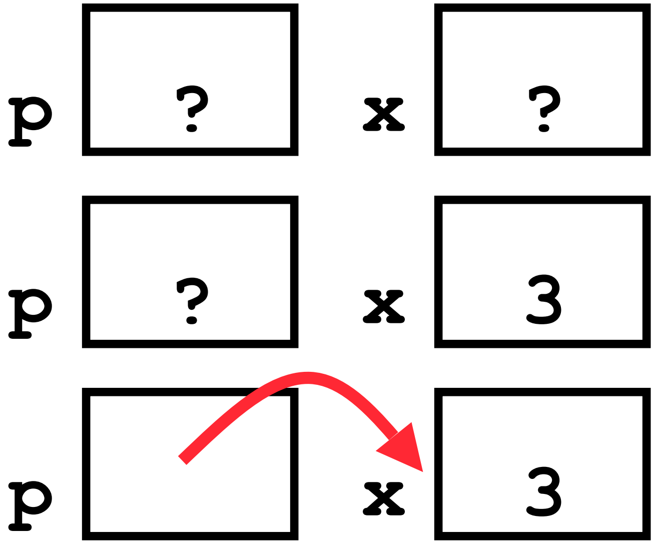


Pointers

- How to create a pointer:
  - `&` operator: get address of a variable

```
int *p, x;
    x = 3;

    p =&x;
```



Note the "*" gets used 2 different ways in this example. In the declaration to indicate that **`p`** is going to be a pointer, and in the **`printf`** to get the value pointed to by **`p`**.

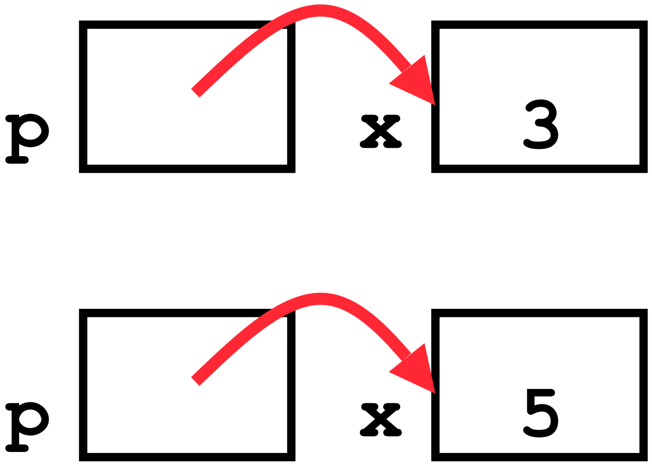
- How get a value pointed to?
  - `*` "dereference operator": get value pointed to

```
printf("p points to %d\n",*p);
```

Pointers

- How to change a variable pointed to?
  - Use dereference * operator on left of =

```
*p = 5;
```



Pointers and Parameter Passing

- **Java and C pass parameters "by value"**
  - **procedure/function/method gets a copy of the parameter, so changing the copy cannot change the original**

```
void addOne (int x) {
   x = x + 1;
}
```

```
int y = 3;
```

```
addOne(y);
```

`y` is still = 3

Pointers and Parameter Passing

- How to get a function to change a value?

```
void addOne (int *p) {
  *p = *p + 1;
}

int y = 3;


addOne(&y);
```

`y` is now = 4

Pointers

- Pointers are used to point to any data type (`int`, `char`, a `struct`, etc.).
- Normally a pointer can only point to one type (`int`, `char`, a `struct`, etc.).
  - `void *` is a type that can point to anything (generic pointer)
  - Use sparingly to help avoid program bugs… and security issues… and a lot of other bad things!

And in conclusion…

- All declarations go at the beginning of each function except if you use C99.
- Only 0 (and NULL) evaluate to FALSE.
- All data is in memory. Each memory location has an address used to refer to it and a value stored in it.
- A pointer is a C version of the address.
  - * "follows" a pointer to its value
  - & gets the address of a value

Reference slides

You ARE responsible for the material on these slides (they're just taken from the reading anyway) ; we've moved them to the end and off-stage to give more breathing room to lecture!

Course Lecture Outline

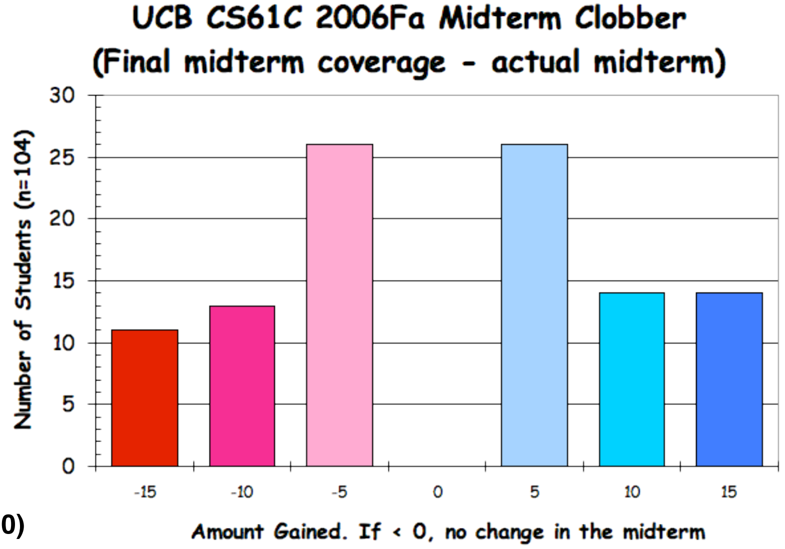
- Basics
  - C-Language, Pointers
  - Memory management
- Machine Representations
  - Numbers (integers, reals)
  - Assembly Programming
  - Compilation, Assembly
- Processors & Hardware
  - Logic Circuit Design
  - CPU organization
  - Pipelining
- Memory Organization
  - Caches
  - Virtual Memory
- I / O
  - Interrupts
  - Disks, Networks
- Advanced Topics
  - Performance
  - Virtualization
  - Parallel Programming

Homeworks, Labs and Projects

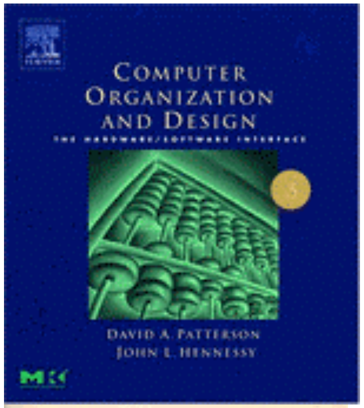
- Lab exercises (due in that lab session unless extension given by TA)
- Homework exercises (~ every week; (HW 0) out now, due in lab Wednesday)
- Projects (every 2 to 3 weeks)
- All exercises, reading, homeworks, projects on course web page
- We will DROP your lowest HW, Lab!
- Only one {Project, Midterm} / week

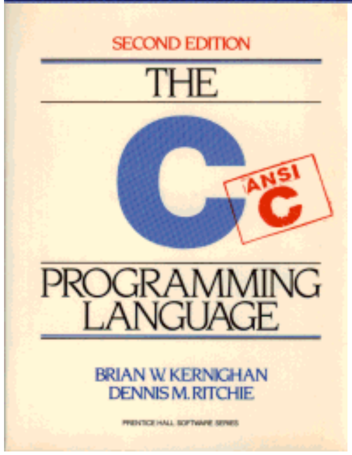
2 Course Exams

- **Midterm: Monday 2009-07-20 In Lecture**
  - Give 1.5 hours for 2.5 hour exam
  - Open everything that can be used during takeoff
  - Review session Fri beforehand, time/place TBA
- **Final: Th 2009-08-13 In "Lecture"**
  - You can *clobber* your midterm grade!
  - (students always LOVE this…)



Texts


Computer Organization and Design
The Hardware/Software Interface
David A. Patterson
John L. Hennessy
MK



- Required: *Computer Organization and Design: The Hardware/Software Interface*, *Third or Fourth Edition*, Patterson and Hennessy (COD). *The second edition is far inferior, and is not suggested.*

- Required: *The C Programming Language*, Kernighan and Ritchie (K&R), 2nd edition

- Reading assignments on web page

Administrivia : You have a question?

- Do not email Jeremy (& expect response)
  - Hundreds of emails in inbox
  - Email doesn’t scale to classes with 100+ students!
- Tips on getting an answer to your question:
  - Ask a classmate
  - Ask Jeremy after or before lecture
  - The newsgroup, ucb.class.cs61c
    - Read it : Has your Q been answered already?
    - If not, ask it and check back
  - Ask TA in section, lab or OH
  - Ask Jeremy in OH
  - Ask Jeremy in lecture (if relevant to lecture)
  - Send your TA email
  - Send your Head TAs email
  - Send Dan email

Administrivia : Lab priority

Rank order of seating priority

1. 61c registered for that section
2. 61c registered for another section
3. 61c waitlisted for that section
4. 61c waitlisted for another section
5. Concurrent enrollment

If low on list for busy section, think of moving to the early or late sections (usually more empty seats)

C vs. Java™ Overview (1/2)

Java	C
Object-oriented (OOP)	No built-in object abstraction. Data separate from methods.
“Methods”	“Functions”
Class libraries of data structures	C libraries are lower-level
Automatic memory management	Manual memory management
	Pointers

C vs. Java™ Overview (2/2)

Java

- High memory overhead from class libraries
- Relatively Slow
- Arrays initialize to zero
- Syntax:

```
 /* comment */
// comment
System.out.print
```

C

- Low memory overhead
- Relatively Fast
- Arrays initialize to garbage
- Syntax: *

```
/* comment */
// comment
printf
```

* You need newer C compilers to allow Java style comments, or just use C99

C Syntax: True or False?

- What evaluates to FALSE in C?
  - 0 (integer)
  - NULL (pointer: more on this later)
  - no such thing as a Boolean*
- What evaluates to TRUE in C?
  - everything else…
  - (same idea as in scheme: only `#f` is false, everything else is true!)

*Boolean types provided by C99's `stdbool.h`

C syntax : flow control

- Within a function, remarkably close to Java constructs in methods (shows its legacy) in terms of flow control
  - `if-else`
  - `switch`
  - `while` and `for`
  - `do-while`