CS 61C: Great Ideas in Computer Architecture

Course Introduction, Number Representation

Instructor: Justin Hsia

Introducing Your Instructor

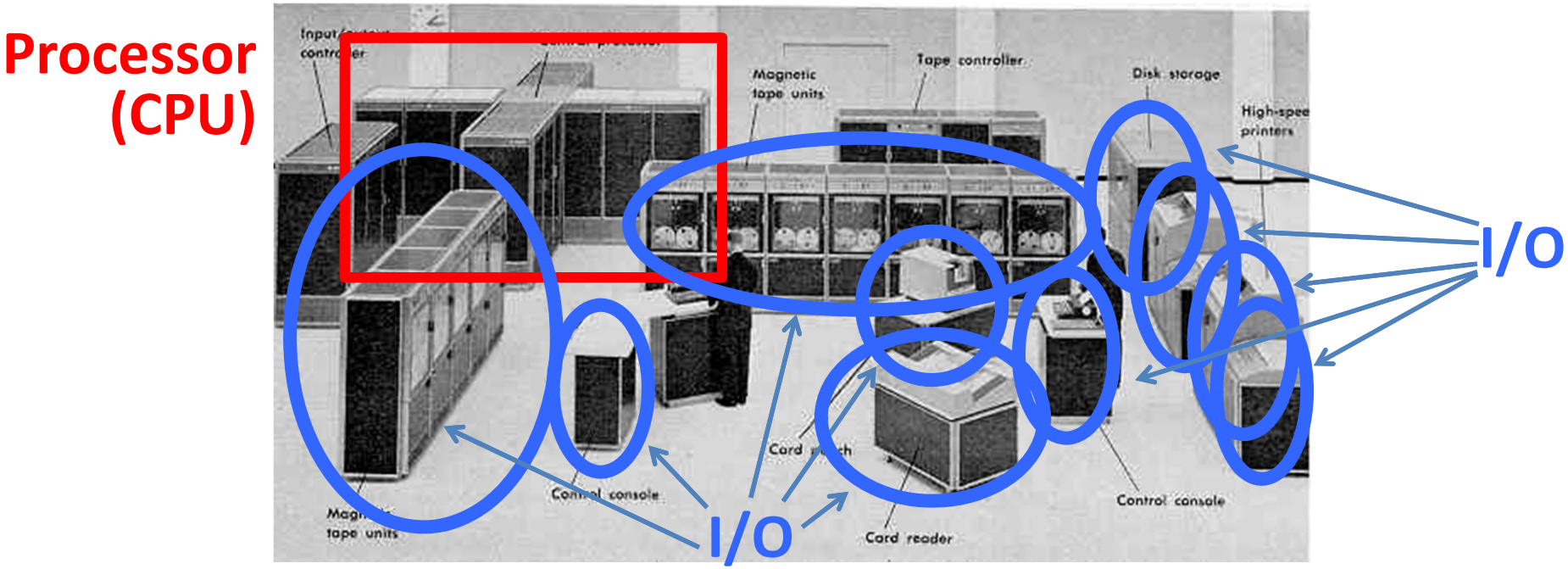
- I'm not a professor – call me Justin
- **Upbringing:** Born and raised in the Bay
- **Education:** I bleed Blue and Gold (Go Bears!)
  - B.S. ME, B.S. EECS, M.S. EECS
  - Ph.D. EECS (expected 2014) in CIR
- **Research:** Synthetic Biology
- **Teaching:** EE128 (Fa08, Fa09), CS61C (Su11)
- **Interests:**



Agenda

- Course Overview
- Administrivia
- Number Representation
  - Number Bases
  - Signed vs. Unsigned
  - Sign Extension

Mainframe Era: 1950s - 1960s



Enabling Tech: Computers

Big Players: "Big Iron" (IBM, UNIVAC)

Cost: $1M, **Target:** Businesses

Using: COBOL, Fortran, timesharing OS

Minicomputer Era: 1970s

Enabling Tech: Integrated circuits
Big Players: Digital, HP
Cost: $10k, **Target:** Labs & universities
Using: C, UNIX OS

PC Era: Mid 1980s - Mid 2000s

Enabling Tech: Microprocessors
Big Players: Apple, IBM
Cost: $1k, **Target:** Consumers (1/person)
Using: Basic, Java, Windows OS

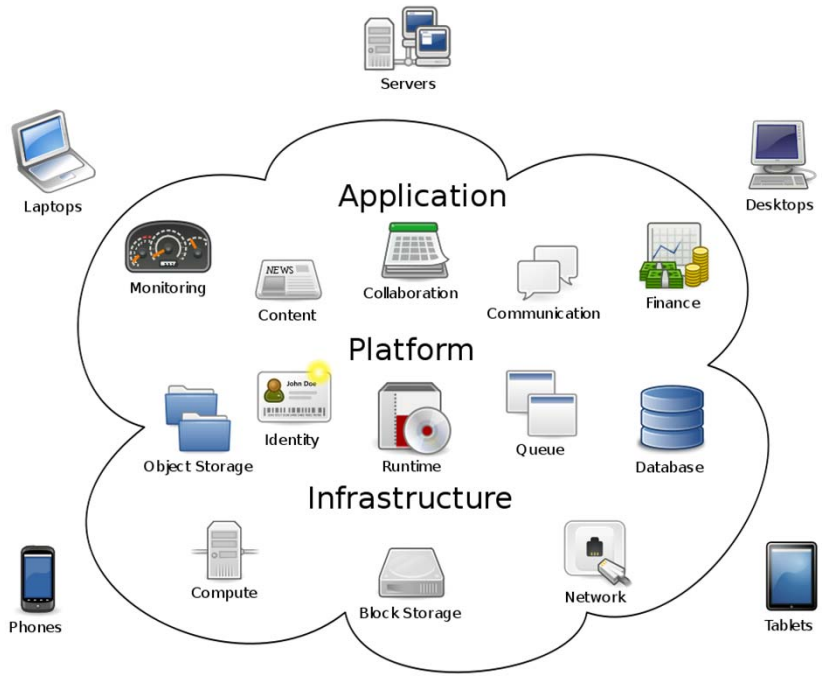
Post-PC Era: Late 2000s - ???

Personal Mobile Devices (PMD):

Enabling Tech: Wireless networking, smartphones
Big Players: Apple, Nokia, …
Cost: $500, **Target:** Consumers on the go
Using: Objective C, Android OS

Post-PC Era: Late 2000s - ???

Cloud Computing:



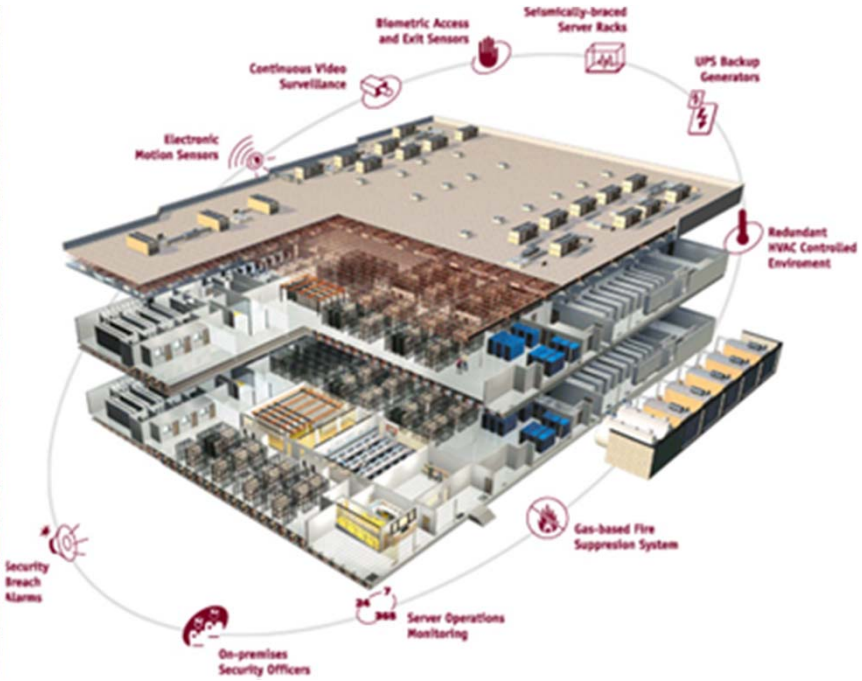
Enabling Tech: Local Area Networks, broadband Internet

Big Players: Amazon, Google, …

Target: Transient users or users who cannot afford high-end equipment

Post-PC Era: Late 2000s - ???

Datacenters and Warehouse Scale Computers (WSC):



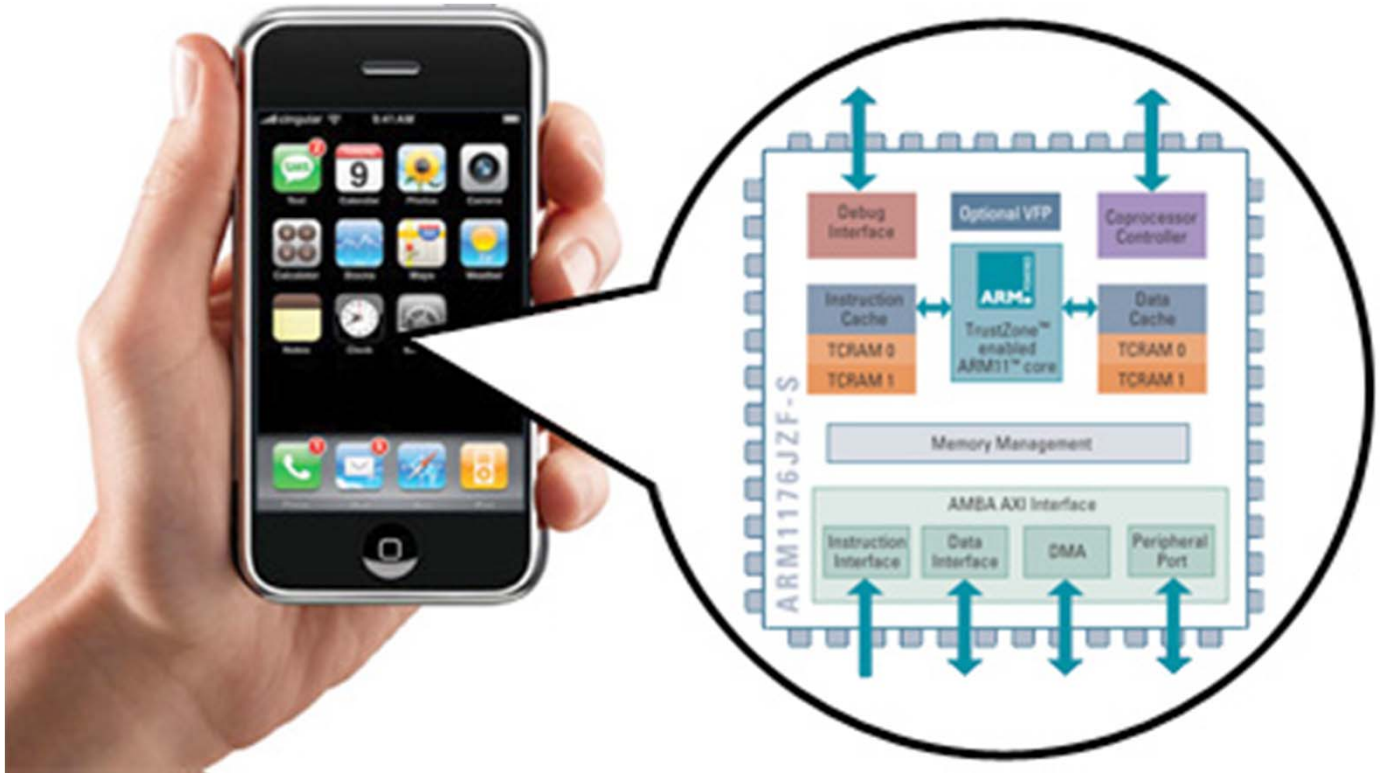
Enabling Tech: Local Area Networks, cheap servers

Cost: $200M clusters + maintenance costs

Target: Internet services and PMDs

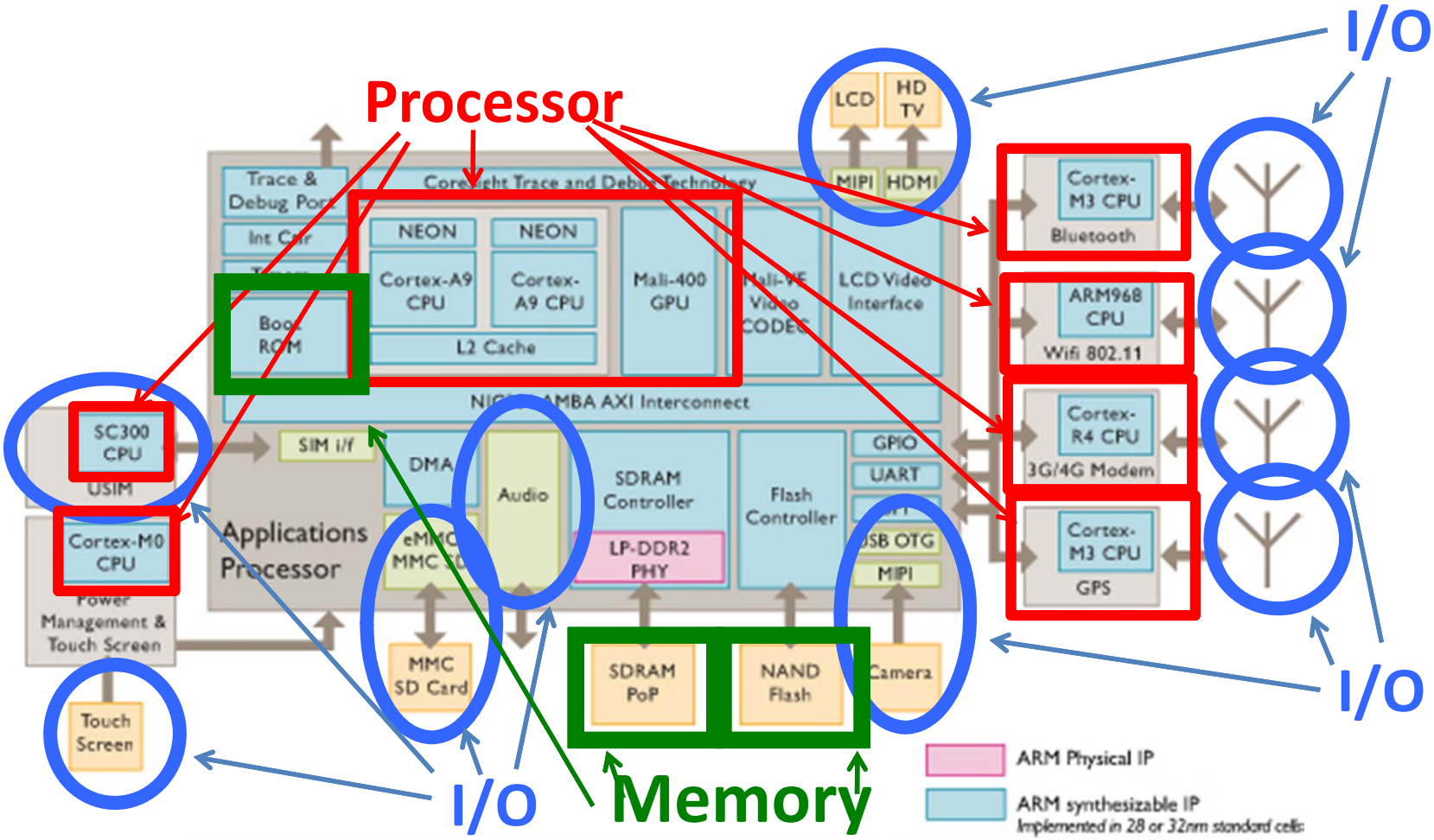
Usages: MapReduce, Ruby on Rails

Advanced RISC Machine (ARM) instruction set inside the iPhone



You will learn how to design and program a related RISC computer: MIPS

iPhone Innards

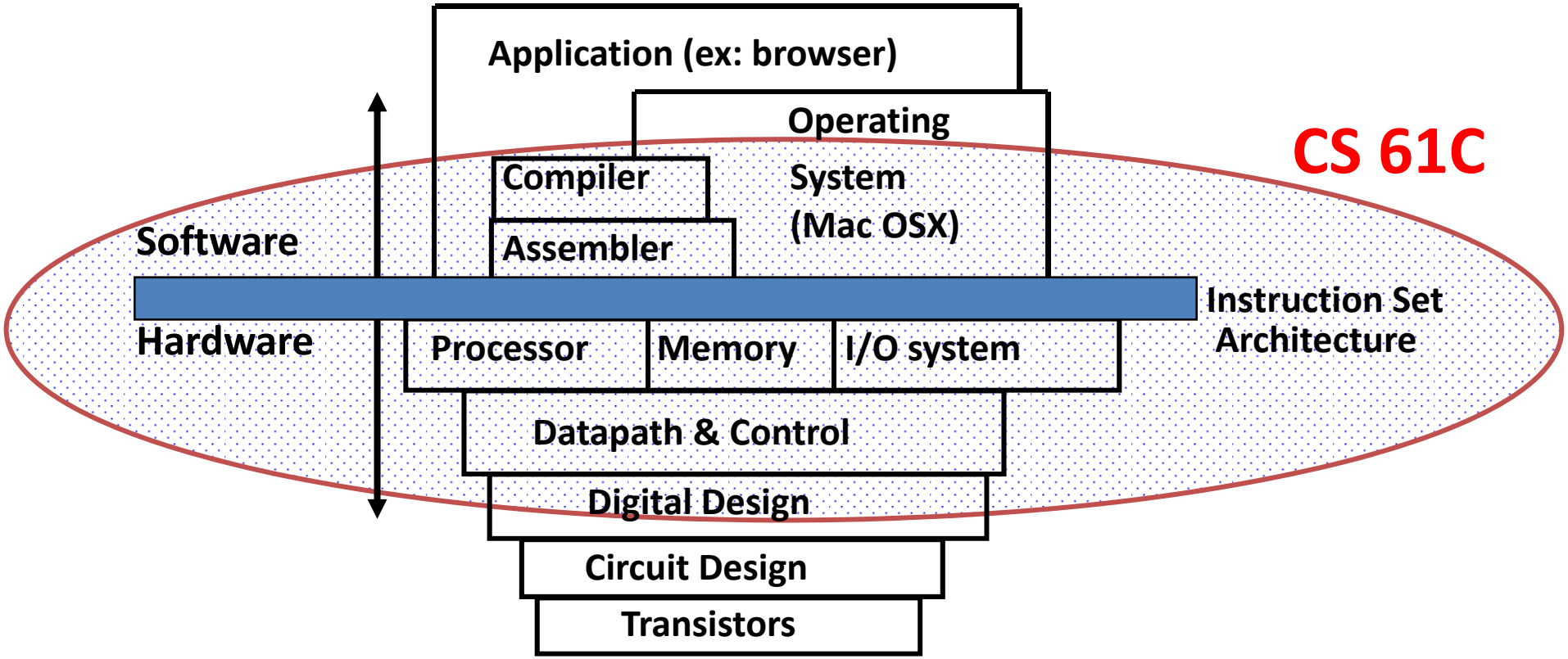


You will learn about multiple processors, data level parallelism, caches in 61C

What is CS 61C about?

- It is about the hardware-software interface
  - What does the programmer need to know to achieve the highest possible performance?
- Use low-level programming languages (closer to underlying hardware)
  - Allows us to talk about key hardware features in higher-level terms
  - Allows programmers to harness underlying hardware parallelism for high performance

Machine Structures



New-School: Parallelism

Software

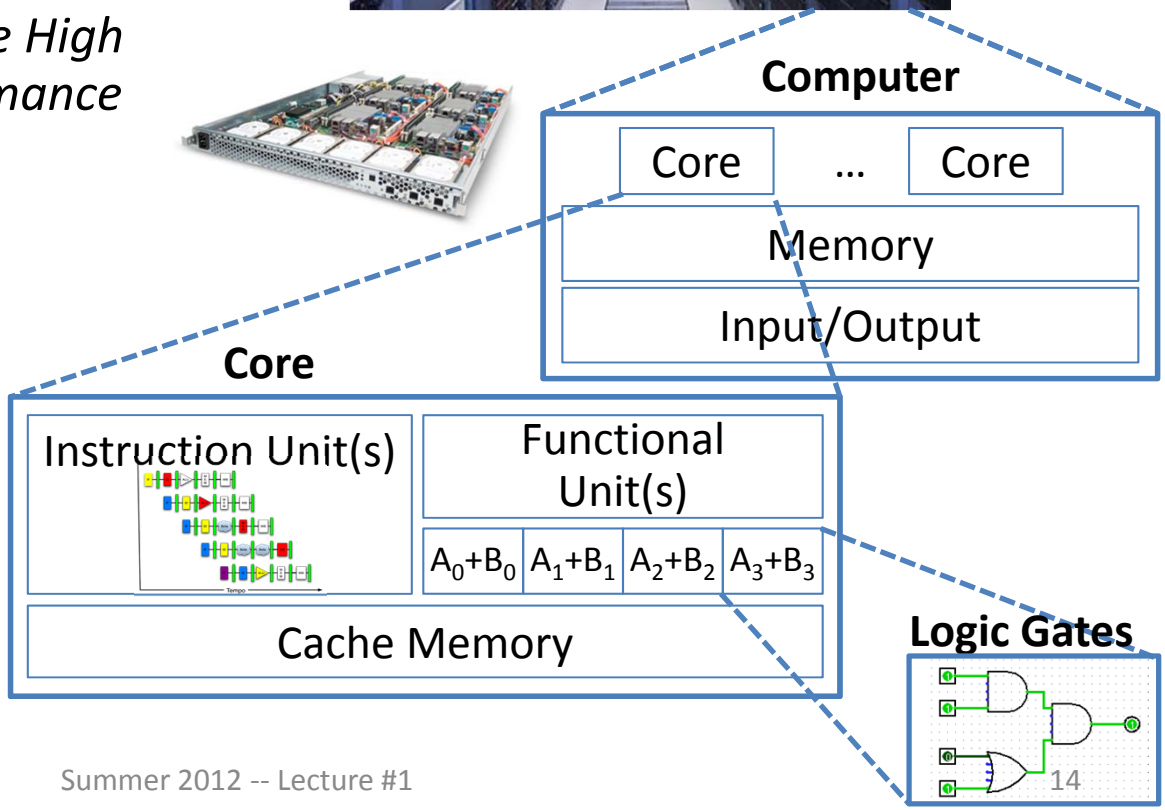
- Parallel Requests
  - Assigned to computer
  - e.g. search "Katz"
- Parallel Threads
  - Assigned to core
  - e.g. lookup, ads
- Parallel Instructions
  - > 1 instruction @ one time
  - e.g. 5 pipelined instructions
- Parallel Data
  - > 1 data item @ one time
  - e.g. add of 4 pairs of words
- Hardware descriptions
  - All gates functioning in parallel at same time

Leverage Parallelism & Achieve High Performance

Hardware

Warehouse Scale Computer

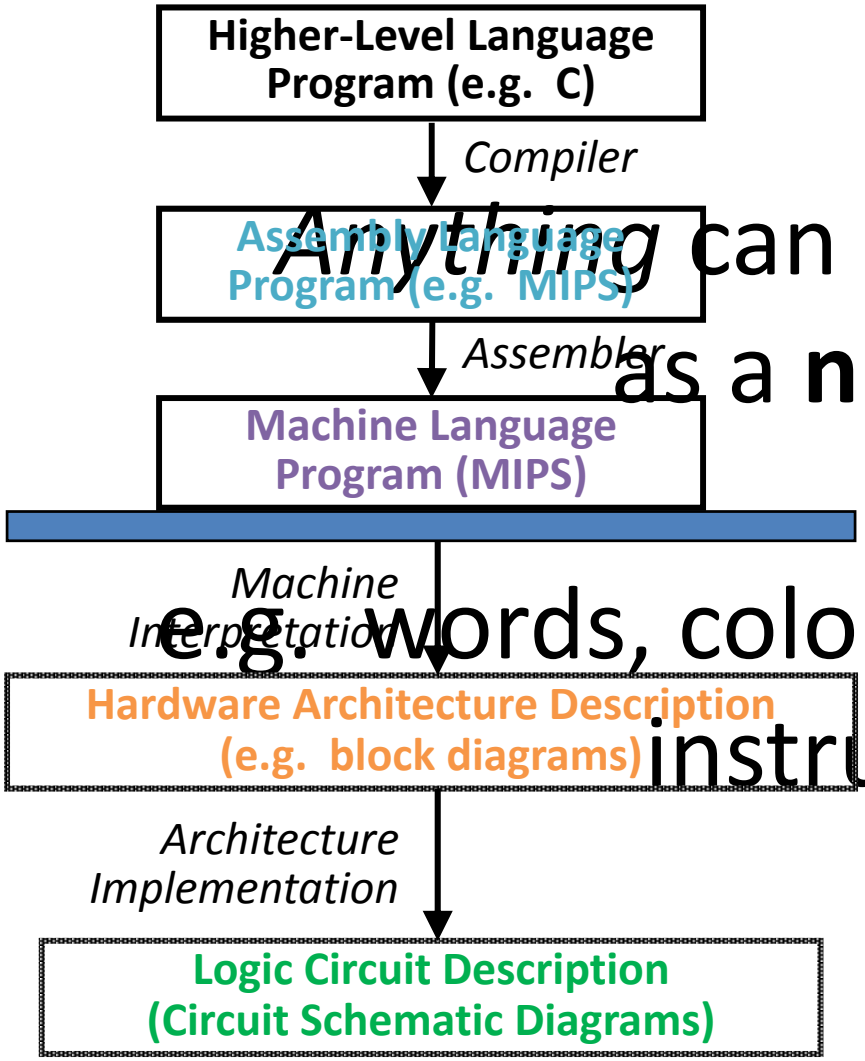
Smart Phone



Six Great Ideas in Computer Architecture

1. Layers of Representation/Interpretation
2. Moore's Law
3. Principle of Locality/Memory Hierarchy
4. Parallelism
5. Performance Measurement & Improvement
6. Dependability via Redundancy

Great Idea #1: Levels of Representation/Interpretation

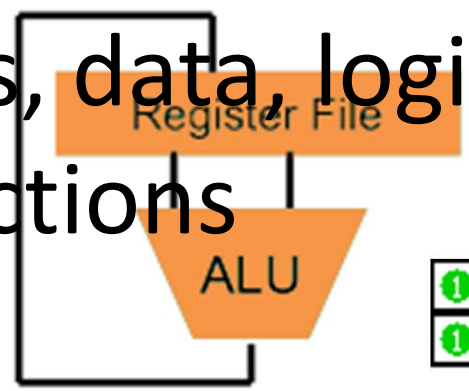


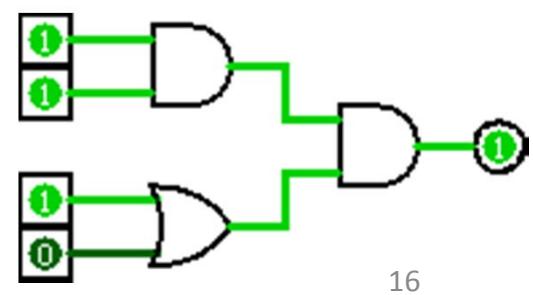

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;

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
```

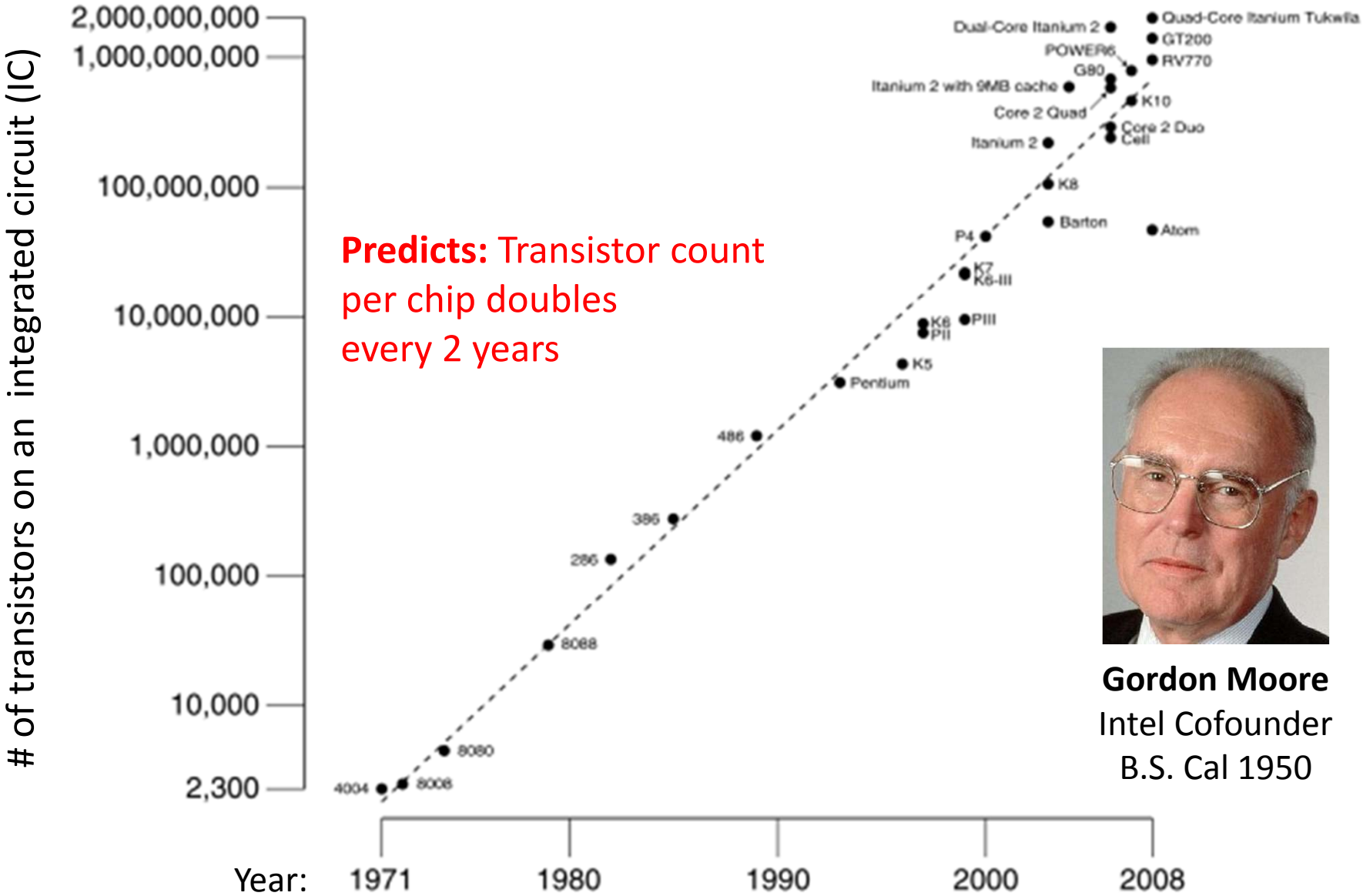
Anything can be represented as a **number!**

e.g. words, colors, data, logic, and instructions





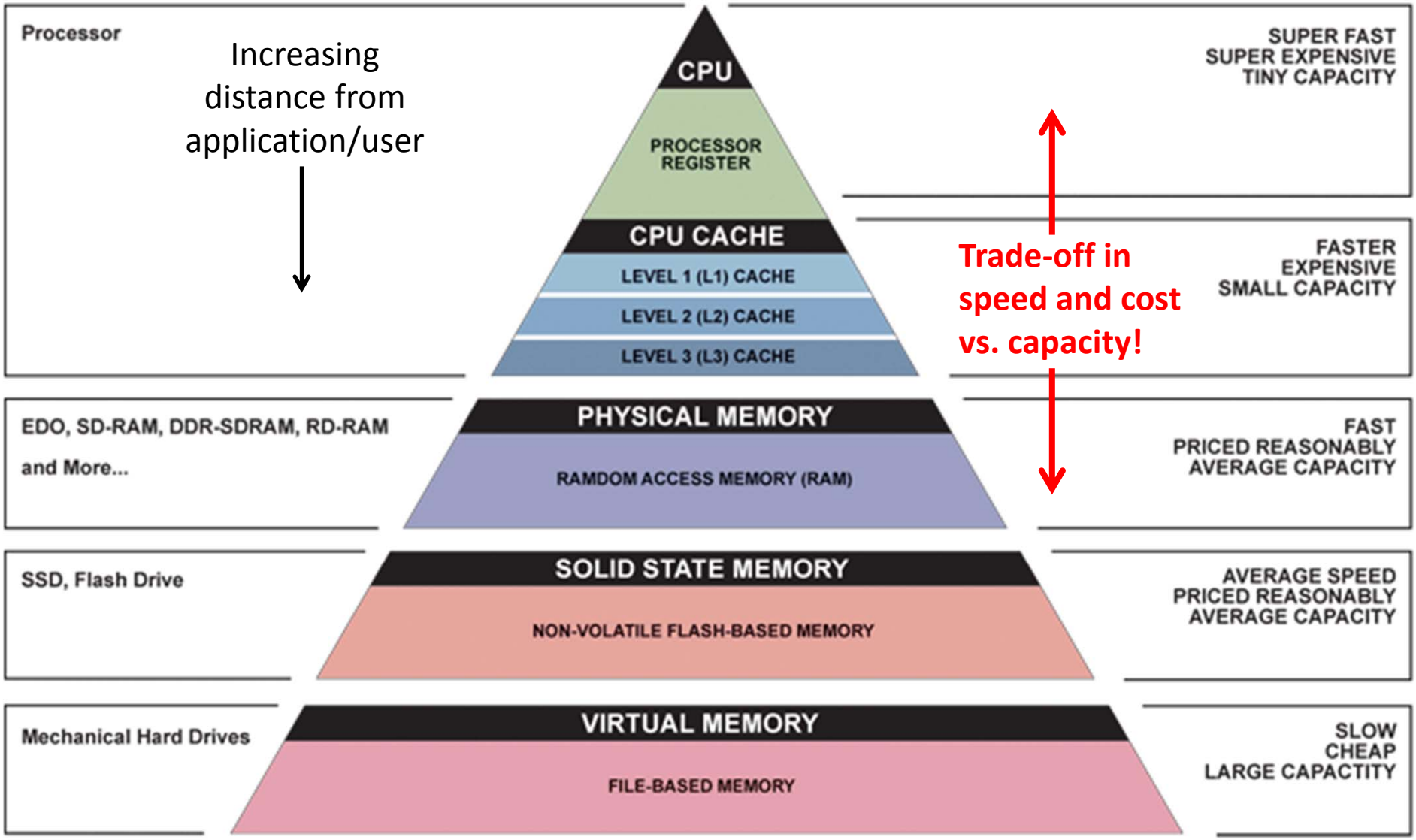
Great Idea #2: Moore's Law



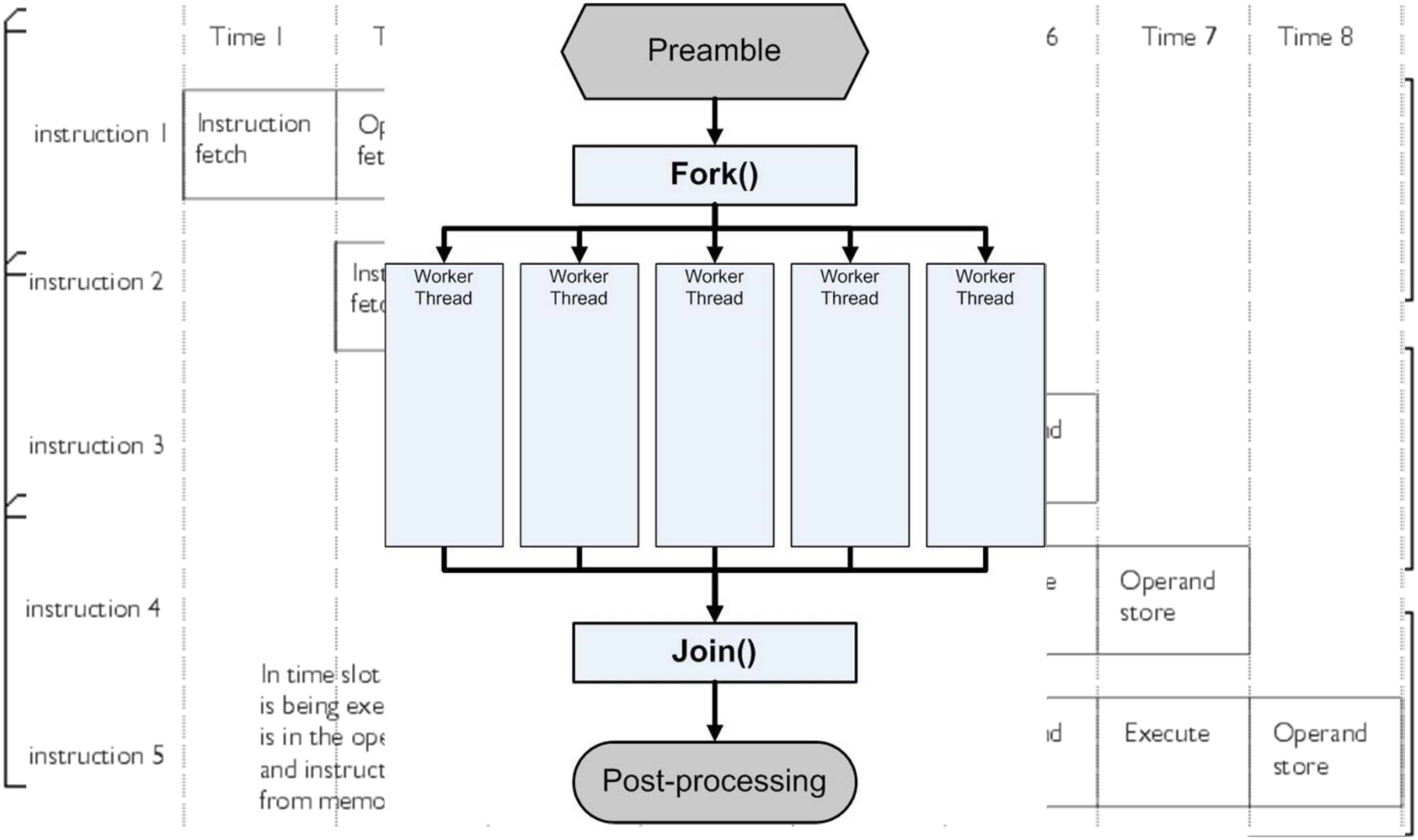
Predicts: Transistor count per chip doubles every 2 years

Gordon Moore
Intel Cofounder
B.S. Cal 1950

Great Idea #3: Principle of Locality/ Memory Hierarchy



Great Idea #4: Parallelism



Great Idea #5: Performance Measurement and Improvement

- Allows direct comparisons of architectures and quantification of improvements
- It is all about *time to finish* (latency)
  - Includes both *setup* and *execution*.
- Match application and hardware to exploit:
  - Locality
  - Parallelism
  - Special hardware features, like specialized instructions (e.g. matrix manipulation)

Aside: Amdahl's Law

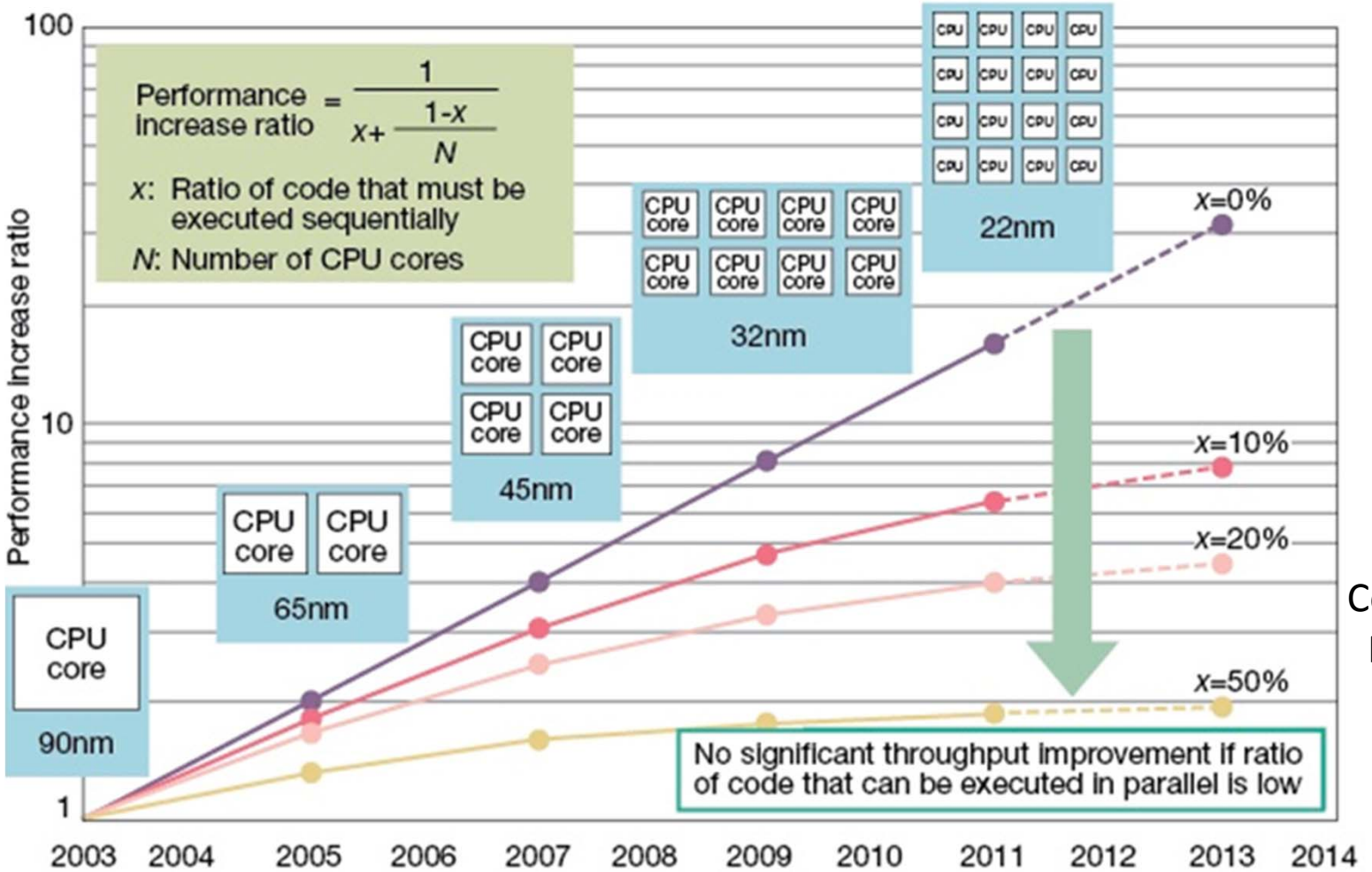
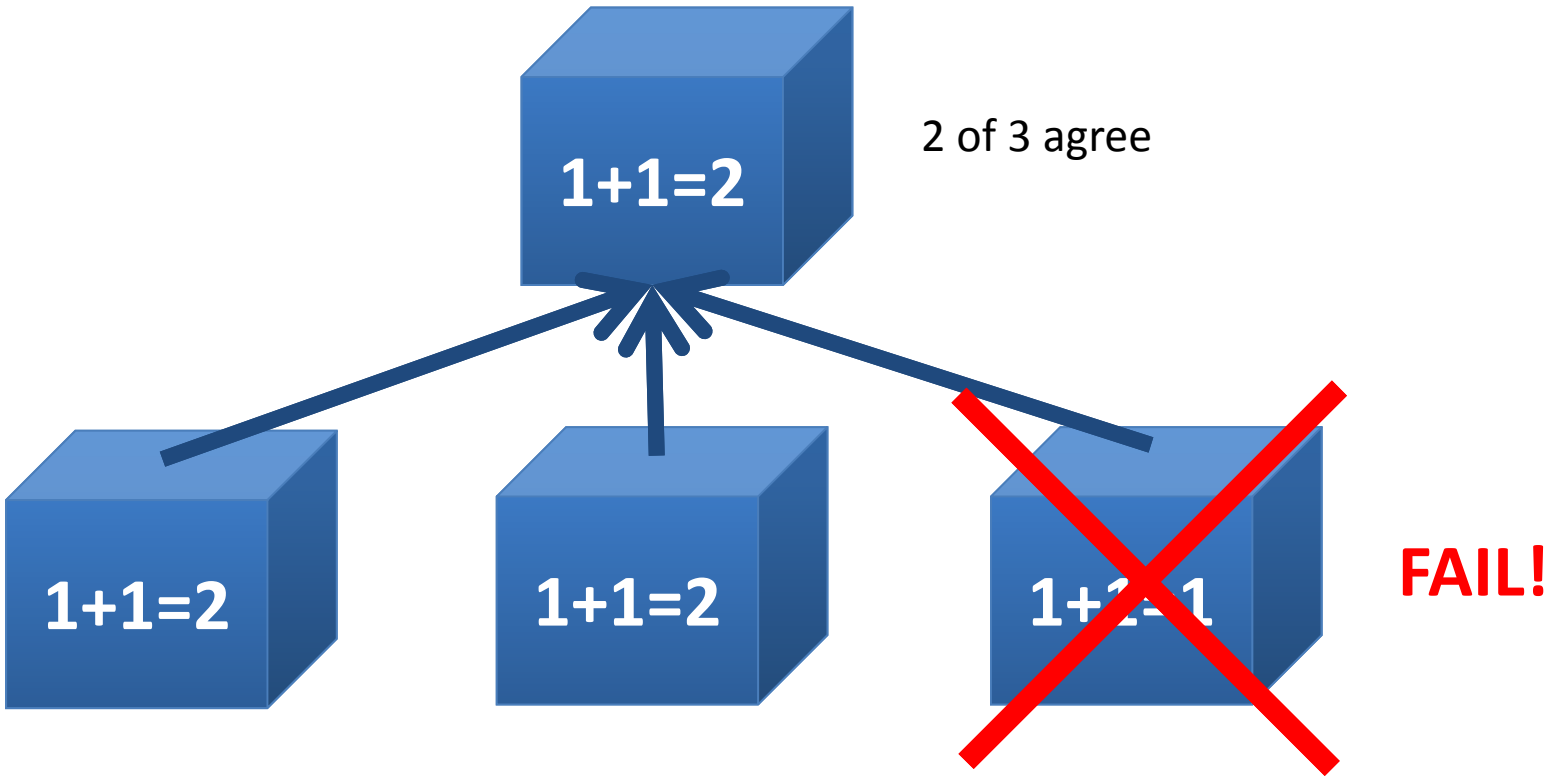


Fig 3 Amdahl's Law an Obstacle to Improved Performance

Gene Amdahl
Computer Pioneer
Ph.D. Wisconsin
1952

Great Idea #6: Dependability via Redundancy

- Redundancy so that a failing piece doesn't make the whole system fail



Great Idea #6: Dependability via Redundancy

- Applies to everything from datacenters to storage to memory
  - Redundant datacenters so that can lose 1 datacenter but Internet service stays online
  - Redundant disks so that can lose 1 disk but not lose data (Redundant Arrays of Independent Disks/RAID)
  - Redundant memory bits of so that can lose 1 bit but no data (Error Correcting Code/ECC Memory)
- Increasing transistor density reduces the cost of redundancy

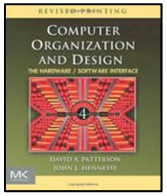
Agenda

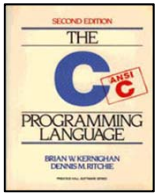
- Course Overview
- Administrivia
- Number Representation
  - Number Bases
  - Signed vs. Unsigned
  - Sign Extension

Course Information

This information can also be found on the course syllabus

- **Website:** http://inst.eecs.berkeley.edu/~cs61c/su12
- **Instructor:** Justin Hsia
- **Teaching Assistants:** Brandon Lee, Paul Ruan, Raphael Townshend, Sung Roa Yoon
- **Textbooks:** average 15 pages of reading/week
  - Patterson & Hennessey, *Computer Organization and Design*, Revised 4th Edition (not ≤ 3rd Edition, not Asian 4th Edition)
  - Kernighan & Ritchie, *The C Programming Language*, 2nd Edition
  - Barroso & Holzle, *The Datacenter as a Computer*, 1st Edition (free!)
- **Piazza** (http://piazza.com) is the class forum
  - Every announcement, discussion, clarification happens there





Course Assignments and Grading

- **Homework** (10%)
- **Labs** (10%)
- **Projects** (30%)
  1. MIPS Instruction Set Simulator (C)
  2. Performance Tuning of a Parallel Application -- Matrix Multiply using cache blocking, SIMD, MIMD (OpenMP)
  3. Computer Processor Pipelined Design (Logisim)
- **Midterm** (20%): Friday, July 13 @ 9am
- **Final** (25%): Thursday, August 9 @ 9am
- **Participation and Altruism** (5%)

Projects

Software

- Parallel Requests
  Assigned to computer
  e.g. search "Katz"
- Parallel Threads
  Assigned to core
  e.g. lookup, ads
- Parallel Instructions
  >1 instruction @ one time
  e.g. 5 pipelined instructions
- Parallel Data
  > 1 data item @ one time
  e.g. add of 4 pairs of words
- Hardware descriptions
  All gates functioning in parallel at same time

Leverage Parallelism & Achieve High Performance

Hardware

MapReduce

Warehouse Scale Computer

Smart Phone

Project 1: MIPS Emulator

Computer

Core … Core

Memory

Input/Output

Project 2: Matrix Multiply

Core

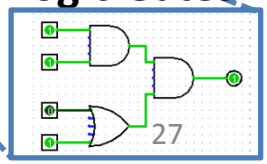
Instruction Unit(s)

Functional Unit(s)

A_0+B_0 A_1+B_1 A_2+B_2 A_3+B_3

Cache Memory

Logic Gates



Project 3: CPU Design

Participation and Altruism

- Participation
  - Asking great questions in discussion and lecture and making it more interactive
  - Attending office hours, completing all assignments (on time is a plus!)
- Altruism
  - Helping others in lab, discussion, and on Piazza

- This is a subjective score (internal)
  - *Reward* for helping, not a *penalty* for not helping
  - The point is to encourage class-wide learning!

Late Policy – Slip Days

- Assignments due at 11:59:59pm (timestamped)
- You have 3 slip day tokens
  - Token used for every day your project or homework is late (even by a second)
  - Keep in mind that projects are worth more
- After tokens, it's 33% deducted per day.
  - No credit if more than 2 days late

- No need for sob stories, just use a slip day!

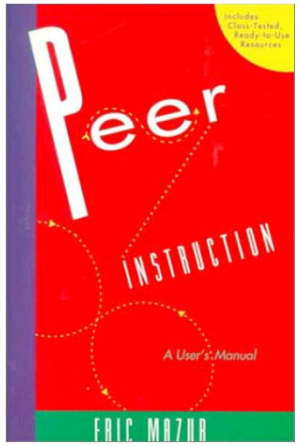
Pedagogic Comments

- Deep learning does not happen in lecture
  - Learn by doing: labs, discussions, and assignments
- Engaging the material outside of class and lab is critical to your success
  - Study groups, testing out your own questions
  - Talking with others helps solidify your own understanding
- You learn best from your mistakes
  - Don't be afraid to be wrong; only you lose if you remain silent

Peer Instruction

- Increase real-time learning in lecture, test understanding of concepts vs. details
  `mazur-www.harvard.edu/education/pi.phtml`

- Multiple choice question at end of a "segment"
  - 1 minute to decide yourself
  - 2 minutes in pairs to reach consensus
  - Learn by teaching!



- Save flash cards for voting (get in discussion section)

1	2	3	4

Question: Which statement is FALSE about Great Ideas in Computer Architecture?

- ☐ To offer a dependable system, you must use components that almost never fail
- ☐ Memory hierarchy goal is to look as fast as most expensive memory, as big as cheapest
- ☐ Moore's Law means computers will continue to get put twice as many transistors/chip every ≈ 2 years without fail
- ☐ **The levels of interpretation/representation mean that we can represent anything as 0's and 1's**

Comments on Summer Version

- Summer is EXTREMELY hectic!
  - Double the standard pace
  - Less time to synthesize material
  - Falling behind just a little can be fatal
- No MapReduce project
- Starts deceptively slowly (first two weeks)
  - If the course begins to overwhelm you, don't wait, contact me or your TA immediately!

Policy on Assignments and Independent Work

- With the exception of Project 2, all homework and projects are to be YOUR work and your work ALONE.
- You are encouraged to discuss your assignments with other students (*ideas*), but we expect that what you hand in is yours.
- It is NOT acceptable to copy solutions from other students.
- It is NOT acceptable to copy (or start your) solutions from the Web.

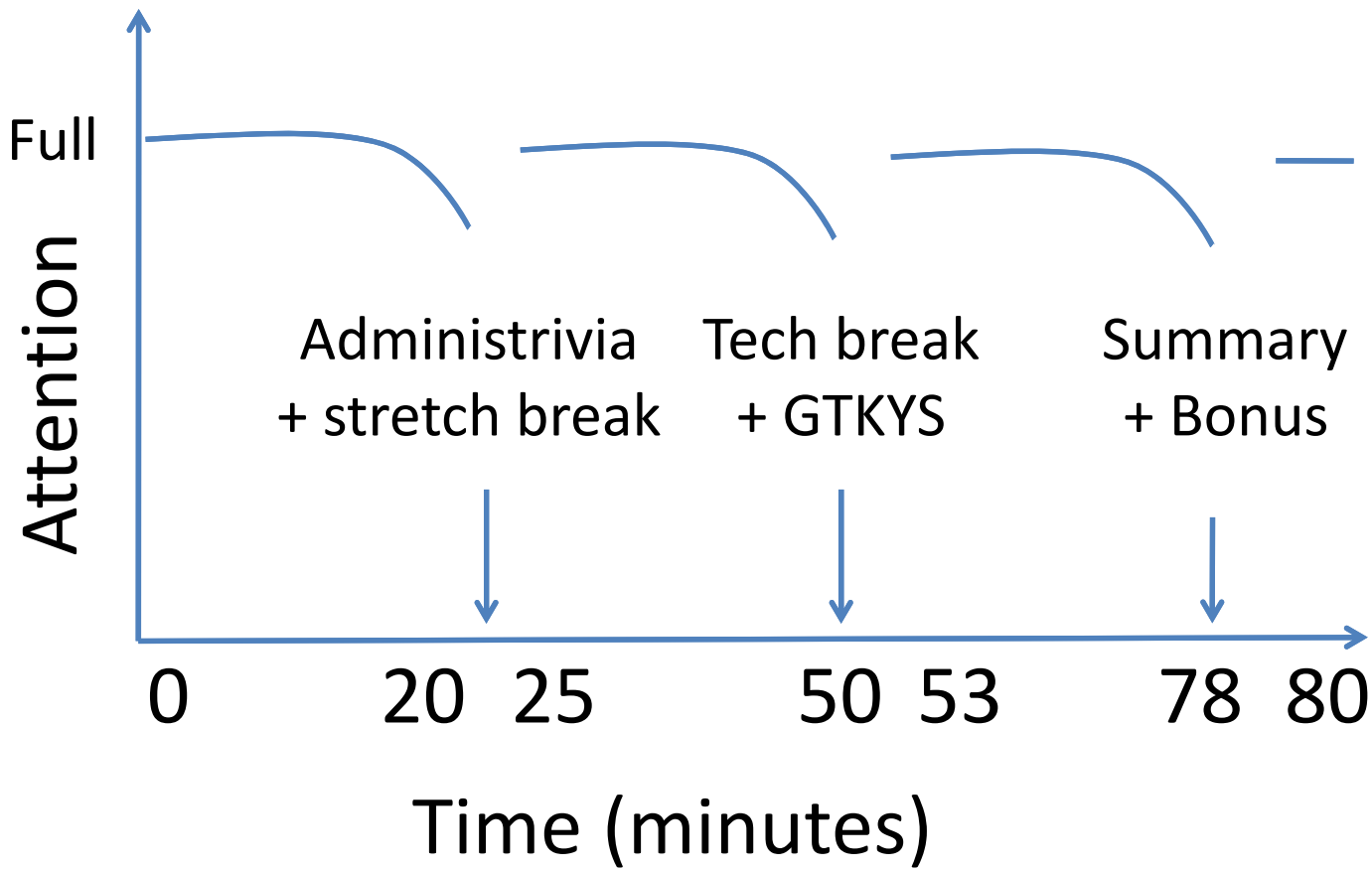
Policy on Assignments and Independent Work

- We have tools and methods, developed over many years, for detecting this. You WILL be caught, and the penalties WILL be severe.
- The *cheater* receives -100% and the *enabler* receives 0% for the assignment. Letter to your university record documenting the incidence of cheating.
  - Possible automatic F in the course
- People are caught every semester of 61C

Hooked on Gadgets

- Gadgets reduce focus and learning
  - Bursts of info (e.g. e-mails, IMs, etc.) are *addictive*
  - Heavy multitaskers have more trouble focusing and shutting out irrelevant information
  - This applies to all aspects of life, not just lecture
- NO audio allowed (mute phones & computers)
- Non-disruptive use okay
  - Stick to side and back seats
  - Stop/move if asked by fellow student

Architecture of a Lecture



Last Things…

- Discussions and labs start immediately
  - Yes, that means today!
  - Switching sections: if you find another 61C student willing to swap discussion AND lab, talk to your TAs
- HW0 due this Tuesday, June 19th
  - Find a small digital image of yourself
- HW1 due this Sunday, June 24th
- No Justin OH this week (presenting research)

Get To Know Your Staff

- Category: **Cal**

Agenda

- Course Overview
- Administrivia
- Number Representation
  - Number Bases
  - Signed vs. Unsigned
  - Sign Extension

Number Representation

- Great Idea #1: Levels of Interpretation/Representation

- Inside a computer, everything stored as a sequence of 0's and 1's (bits)
  - Even this is an abstraction!
- How do we represent numbers in this format?
  - Let's start with integers

Number Bases

- **Key terminology:** digit (*d*) and base (*B*)
- Value of *i*-th digit is $d \times B^i$ where *i* starts at 0 and increases from right to left
  - *n* digit number $d_{n-1} d_{n-2} \ldots d_1 d_0$
  - value = $d_{n-1} \times B^{n-1} + d_{n-2} \times B^{n-2} + \ldots + d_1 \times B^1 + d_0 \times B^0$
- In base B, each digit is one of B possible symbols
- Base is notated either as a prefix or subscript

Commonly Used Number Bases

- **Decimal** (base 10)
  - Symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
  - Notation: 9472_{ten} = 9472
- **Binary** (base 2)
  - Symbols: 0, 1
  - Notation: 101011_{two} = 0b101011
- **Hexadecimal** (base 16)
  - Symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
  - Notation: $2A5D_{hex}$ = 0x2A5D

Decimal	Binary	Hex
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Number Base Examples

- Examples:

3 2 1 0

$\mathbf{9472}_{ten} = 9000 + 400 + 70 + 2$

$9x1000 + 4x100 + 7x10 + 2x1$

$9x10^3 + 4x10^2 + 7x10^1 + 2x10^0$

$\mathbf{9472}_{ten} = 2x16^3 + 5x16^2 + 0x16^1 + 0x16^0$

$= \mathbf{2500}_{hex}$

0xA15 = 0b 1010 0001 0101

Bits Can Represent Anything

- *n* digits in base *B* can represent at most B^n things!
  - Each of the *n* digits is one of *B* possible symbols
  - Have more things? Add more digits!
- Example: Logical values (1 bit) – 0 is False, 1 is True
- Example: Characters
  - 26 letters require 5 bits ($2^5 = 32 > 26$)
- Example: Students in this class (7 bits)

- For convenience, can group into *nibbles* (4 bits) and *bytes* (8 bits)

So What Number Is It Anyway?

- Here we are using binary bit patterns to represent numbers
  - Strictly speaking they are called *numerals* and have no meaning until you interpret them
  - Is CAB a word (taxi) or a number (3243_{ten})?
  - Is 0x999999 a number or a color (RGB)?
- Keep in mind that the same bit pattern will mean different things depending on how you choose to interpret it

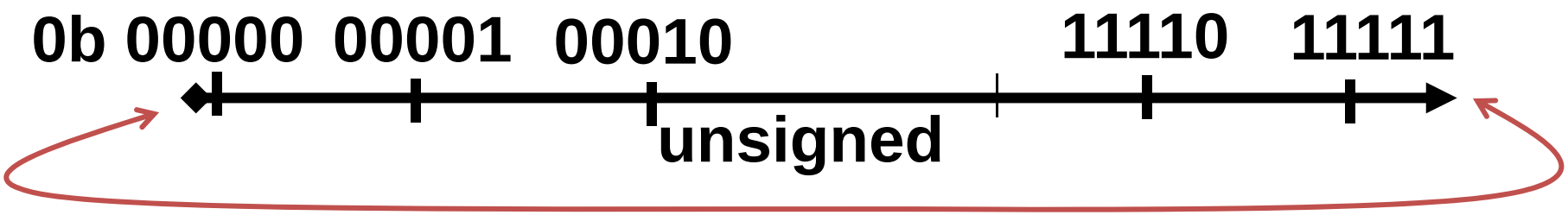
Unsigned Integers

Represent non-negative (unsigned) integers using base 2:

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = 0_{ten}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 1_{ten}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = 2_{ten}$
... ...
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{two} = 2{,}147{,}483{,}645_{ten}$
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = 2{,}147{,}483{,}646_{ten}$
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = 2{,}147{,}483{,}647_{ten}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = 2{,}147{,}483{,}648_{ten}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 2{,}147{,}483{,}649_{ten}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = 2{,}147{,}483{,}650_{ten}$
... ...
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{two} = 4{,}294{,}967{,}293_{ten}$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = 4{,}294{,}967{,}294_{ten}$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = 4{,}294{,}967{,}295_{ten}$

Overflow

- Numbers really have ∞ digits, but hardware can only store a finite number of them (fixed)
  - Usually ignore *leading zeros*
  - Leftmost is *most significant bit* (MSB)
  - Rightmost is *least significant bit* (LSB)
- Overflow is when the result of an arithmetic operation can't be represented by the hardware bits

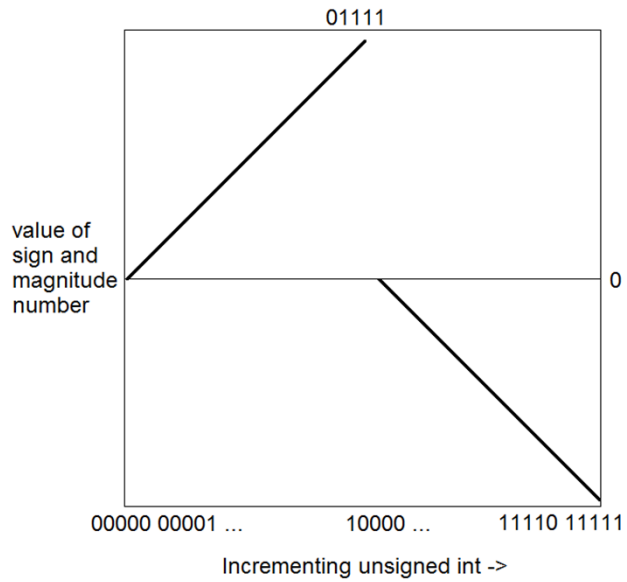


Signed Integers

- Programs often need to deal with negative numbers, so how do we encode these?
- *n* bits can represent 2^n different things
  - Ideally, want the range evenly split between positive and negative
- Can we encode them in such a way that we can use the same hardware regardless of whether the numbers are signed or unsigned?

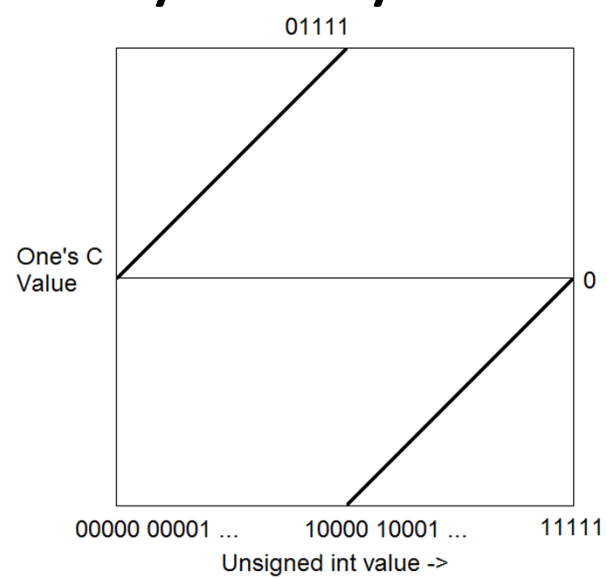
Sign and Magnitude

- MSB gives sign: 0 is positive, 1 is negative, rest of bits treated as unsigned (magnitude)
  - Examples: 0b 1000 0010 = -2, 0b 0000 0111 = 7
- Two zeros! 0b00…0 (+0) and 0b10…0 (-0)
- Cannot reuse unsigned hardware



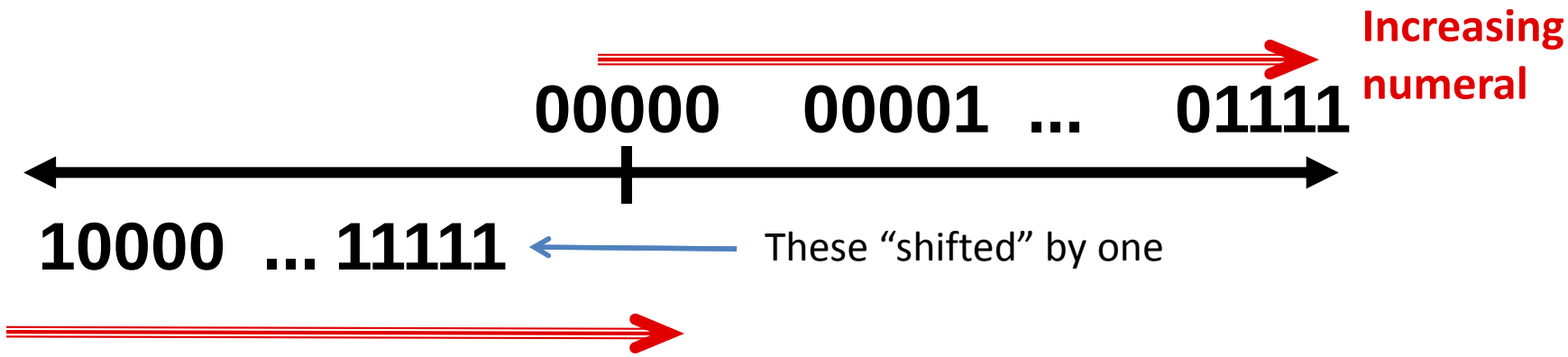
One's Complement

- To negate: complement the bits
  - Example: +7 = 0b 0000 0111, -7 = 0b 1111 1000
- Leading 0's if positive, leading 1's if negative
- Incrementing the numeral nearly always increments the number, with one exception:
  - The two zeros: 0b00…000 and 0b11…111



Two's Complement

- Minor modification of one's complement
  - "Shift" representation of negative numbers down by one to remove duplicate zero



- Using this representation, incrementing the numeral *always* increments the integer
- To negate: complement the bits and add 1

Two's Complement

Sign Bit

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = 0_{ten}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 1_{ten}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = 2_{ten}$
… …
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{two} = 2{,}147{,}483{,}645_{ten}$
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = 2{,}147{,}483{,}646_{ten}$
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = 2{,}147{,}483{,}647_{ten}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = -2{,}147{,}483{,}648_{ten}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = -2{,}147{,}483{,}647_{ten}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = -2{,}147{,}483{,}646_{ten}$
… …
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{two} = -3_{ten}$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = -2_{ten}$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = -1_{ten}$

Two's Complement Summary

- Used by all modern hardware
- Roughly evenly split between positive and negative
  - One more negative # because positive side has 0
- Can still use MSB as sign bit
- To negate: Flip the bits and add one
  - Example: +7 = 0b 0000 0111, -7 = 0b 1111 1001

Two's Complement Review

- Suppose we had 4 bits. What integers can be represented in two's complement?

a)	-15 to +15	←	need 5 bits
b)	-7 to +7	←	one's complement
c)	0 to +15	←	unsigned
d)	-8 to +7	←	two's complement
e)	-16 to +15	←	need 5 bits

Sign Extension

- Want to represent the same number using more bits than before
  - Easy for positive #s (add leading 0's), more complicated for negative #s
  - Sign and magnitude: add 0's *after* the sign bit
  - One's complement: copy MSB
  - Two's complement: copy MSB
- Example:
  - Sign and magnitude: 0b 11 = 0b 1001
  - One's/Two's complement: 0b 11 = 0b 1111

Summary (1/2)

- CS61C: Learn 6 Great Ideas in Computer Architecture to enable high performance programming via parallelism
  1. Layers of Representation/Interpretation
  2. Moore's Law
  3. Principle of Locality/Memory Hierarchy
  4. Parallelism
  5. Performance Measurement and Improvement
  6. Dependability via Redundancy

Summary (2/2)

- Number Representation: How to represent positive and negative integers using binary
  - Unsigned: Interpret numeral in base 2
  - Signed: Two's Complement
  - Sign extension must preserve signed number