CS61C: Great Ideas in Computer Architecture (a.k.a. Machine Structures)

Dr. Nick Weaver
Lecturer

- My primary specialty is Network Security and Network Measurement
  - Although a reformed hardware person, originally specializing in FPGAs
- I will sprinkle a fair bit of security stuff throughout the lectures
  - Security is not an afterthought, but needs to be engineered in from the start: Since this class covers everything from the transistor to the cloud, I'll make security notes along the way

Course Information

- Course Web: http://www-inst.eecs.bekeley.edu/cs61c/
- Instructors:
  - Nicholas Weaver
- Teaching Assistants: (see webpage)
- Major tutoring support from CS-Mentors
- Textbooks: Average 15 pages of reading/week (can rent!)
  - Patterson & Hennessey, Computer Organization and Design, 5/e (RISC-V)
  - Kernighan & Ritchie, The C Programming Language, 2nd Edition
  - Barroso & Holzle, The Datacenter as a Computer, 2nd Edition

Course Grading

- EPA: Effort, Participation and Altruism (5%)
- Homework (5%)
- Labs (5%, Individually or in groups of 2)
- Projects (25%) (Projects done and submitted ***individually***)
  - A Learn to C Project (C)
  - Code Generation/Assembly (RISC-V & C)
  - Computer Processor Design (Logisim)
  - Parallelize for Performance, SIMD, MIMD (C)
  - Concurrency (Go)
- Two midterms (15% each)
- Final (30%)
  - NO clobbering

Piazza

- Piazza is an official channel
  - We will post announcements in it and we expect that, by posting announcements, you will read them
  - You can use this as a discussion forum to ask open questions
  - If you have private questions of the instructors and staff:
    do not use email, use a private question in Piazza

Gradescope...

- We will use Gradescope for all assignments
  - Midterms/Final with a regrade window
  - Homeworks
  - Lab checkoffs
    - You will be given ***unique*** codes that you need to enter ***exactly***

Scaling the class...

- I may be House Slytherin... But there is still a lot of Hufflepuff:
  - "I will teach them all and treat them all the same"
- We want to scale to support every student who wants to take this class
  - Lectures will be webcast
    - Why the iClicker is only one dimension of EPA
  - Attend ***any*** discussion & lab section
    - Lab checkoffs designed to be super-fast
  - Lots of TA & Tutoring Staff
  - Use Piazza for questions! DO NOT USE EMAIL!

Tried-and-True Technique: Peer Instruction

- Increase real-time learning in lecture, test understanding of concepts vs. details
- As complete a “segment” ask multiple-choice question
  - 1-2 minutes to decide yourself
  - 2 minutes in pairs/triples to reach consensus.
  - Teach others!
  - 2 minute discussion of answers, questions, clarifications
- You can get transmitters from the ASUC bookstore
  - The WiFi is generally not good enough for REEF soft-clickers

EECS Grading Policy

- "A typical GPA for a lower division course will fall in the range 2.8 - 3.3, depending on the course and the students who enroll. For example, a GPA of 3.0 would result from 35% A's, 45% B's, 13% C's, and 7% D's and F's. Introductory courses specifically designed for a broad audience may fall outside of this range."
  - http://www.eecs.berkeley.edu/Policies/ugrad.grading.shtml
  - We will curve the class to the target range if the grades are below the target curve
    - I believe in hard tests but a reasonable curve:
      Don't get discouraged if you find a test hard! They are ***supposed*** to be hard!
- Job/Intern Interviews: They grill you with technical questions, so it's what you say, not your GPA
  - Failure is always an option. If you want a laugh, my undergrad transcript is in my office

My goal as a lecturer

- To make your experience in CS61C as enjoyable & informative as possible
  - Humor, enthusiasm & technology-in-the-news in lecture
  - Fun, challenging projects & HW
  - Pro-student policies

- To maintain Cal & EECS standards of excellence
  - Projects & exams will be as rigorous as every year.
- Score 7.0 on HKN:
  - Please give feedback! Why are we not 7.0 for you? We will ***listen***!

EPA!

- **E**ffort
  - Attending prof and TA office hours, completing all assignments, turning in HW, doing reading quizzes
- **P**articipation
  - Attending lecture and voting using the clickers
    - Note: We will ensure that this is just one dimension of EPA, and we will ignore the lowest dimension
  - Asking great questions in discussion and lecture and making it more interactive
- **A**ltruism
  - Helping others in lab or on Piazza: Be Excellent to Each Other
- EPA! points have the potential to bump students up to the next grade level! (but actual EPA scores are internal)

Checkoff Tags...

- For both lab checkoffs and EPA we are handing out small tags
  - They have a 32b value in hexadecimal (e.g. **`56789ABC`**) along with the assignment they are good for.
  - Enter that value in Gradescope to get credit
  - Codes are effectively ***unique***: Sharing codes results in ***negative*** points!
- For EPA, a TA can give you a tag for ***participation***:
  - EG, interacting in discussion, attending office hours, being seen being helpful to other students in lab
    - You ***can not*** get an EPA tag just for showing up to discussion!
  - EPA tags are specified in 2-week periods
    - You can ***only*** redeem up to 2 EPA tags for each 2-week period

Late Policy…
Slip Days!

- Assignments due at 11:59:59 PM PT
- You have 3 slip day tokens (NOT hour or min)
- Every day your project is late (even by a ***millisecond***) we deduct a token
- After you've used up all tokens, it's 33.3333333% deducted per day.
  - No credit if more than 3 days late
  - Cannot be used on homeworks!
- No need for sob stories, ***just use a slip day***!

Policy on Assignments and Independent Work

- ALL PROJECTS WILL BE DONE AND SUBMITTED INDIVIDUALLY
- With the exception of laboratories and assignments that explicitly permit you to work in groups, all homework and projects are to be YOUR work and your work ALONE.
- You are encouraged to discuss your assignments with other students, and extra credit will be assigned to students who help others, particularly by answering questions on Piazza, but we expect that what you hand in is yours.
- It is NOT acceptable to copy (or even "start with") solutions from other students or the Web
- It is NOT acceptable to use PUBLIC GitHub archives (giving your answers away)
- We have tools and methods, developed over many years, for detecting this. You WILL be caught, and the penalties WILL be severe.
- Both Giver and Receiver are equally culpable and suffer equal penalties
  - If it is from a previous semester, the previous semester's students will ***also be reported to the student conduct office***

Use Git and Push Often…

- You will be using BitBucket and/or GitHub to host your projects for submission…
  - So use it for your normal workflow too
- Push your work back on a regular basis
  - It really prevents screwups:
    "Ooops, go back" is the reason for version control
    - Your computer should be able to ***blow up***, and you should only lose a couple hours work!
  - It gives a timestamp we can trust of when you wrote your code
    - Very useful if flagged for cheating
- Also, for any C coding, use valgrind
  - C is ***not memory safe***,
    valgrind will catch most of these errors when you make them

Debugging...

- We are all very helpful during office hours...
  - But we will ***not*** simply debug your project for you
- In order to receive project assistance you ***must***:
  - Have a test case which shows the problem
  - Have the debugger running and at a breakpoint that shows the problem
  - If it is a memory problem (segfault etc) you must also have the project running in valgrind to indicate where the problem is

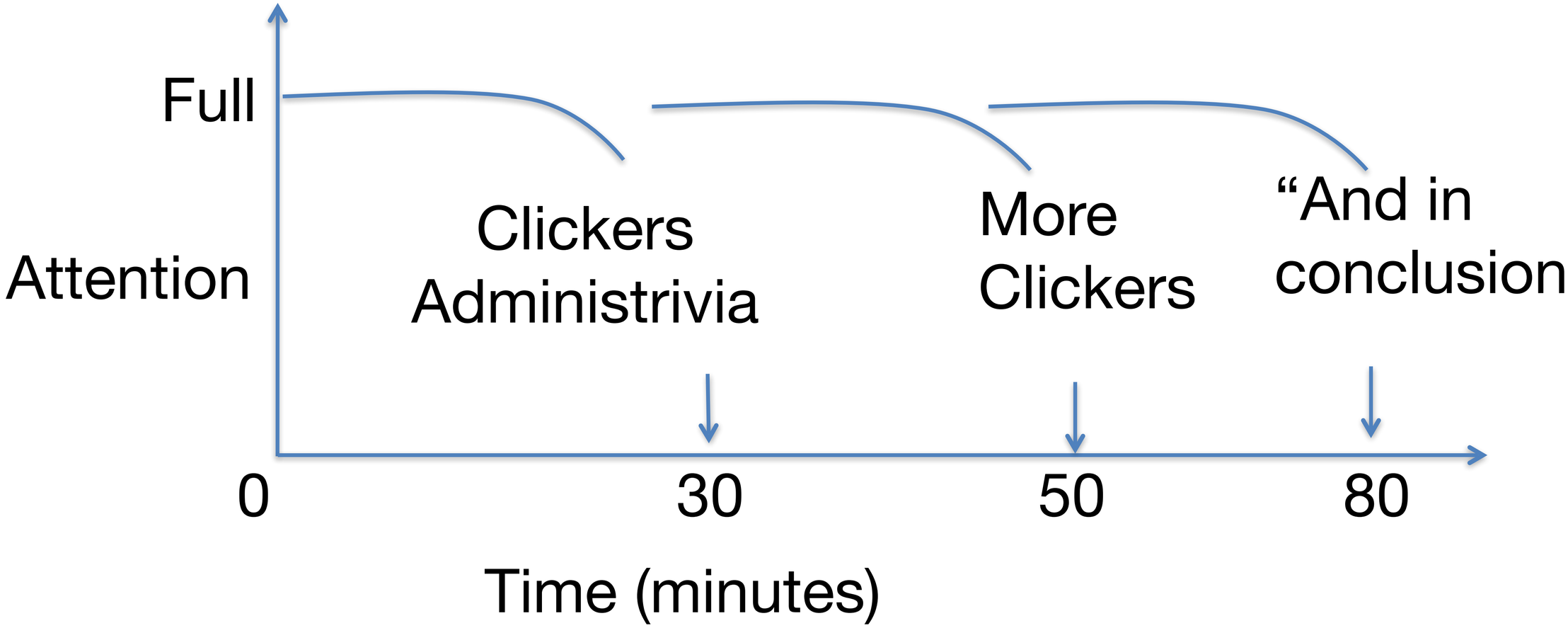
Intellectual Honesty Policy: Detection and ***Retribution***

- We view those who would cheat as “attackers”
  - This includes sharing code on homework or projects, midterms, finals, etc…
  - But we (mostly) assume rational attackers: Benefit of attack > ***Expected*** cost
    - Cost of launching attack + cost of getting caught * probability of getting caught
- We take a detection and response approach
  - We use many tools to detect violations
    - "Obscurity is not security", but obscurity can help. Just let it be known that "We Have Ways"
  - We will go to DEFCON 1 (aka "launch the nukes") ***immediately***
    - “Nick doesn’t make threats. ***He keeps promises***”
    - Punishment can be up to an F in the class but, at minimum, ***negative points***:
      - You will do better if you don't do your work at all than if you cheat
      - ***All*** incidents will be reported to the office of student conduct

Stress Management & Mental Health...

- We'll try to not over-stress you too much
  - But there really is a lot to cover and this really is a demanding major
  - There are 5 projects!
- If you feel overwhelmed, please use the resources available
  - Academically: Ask on Piazza, Tutoring, Office hours, the Slack channel, Guerrilla sections, etc...
  - Non-Academic: Take advantage of University Health Services if you need to
    - ***Nick did!*** Zoloft (an antidepressant) and therapy saved my life, twice.

Architecture of a typical Lecture



Agenda

- What you need to know about this class
- Thinking about Machine Structures
- Great Ideas in Computer Architecture
- Number Representation

Agenda

- What you need to know about this class
- Thinking about Machine Structures
- Great Ideas in Computer Architecture
- Number Representation

CS61C is not about C Programming

- It is about the hardware-software interface
  - What does the programmer need to know to achieve the highest possible performance
- C is close to the underlying hardware, unlike languages like Scheme, Python, Java, Go, Perl, R, Prolog...
  - Allows us to talk about key hardware features in higher level terms
  - Allows programmer to explicitly harness underlying hardware parallelism for high performance
- Also allows programmer to shoot oneself in the foot in ***amazingly*** spectacular ways
  - One of my personal goals in this class is for you to develop a ***rational hatred*** of C

Other Pedagogical Choices In This Class...

- Our assembly language is RISC-V...
  - If you ever have to program in assembly, it will probably be x86 or 32b ARM
  - But RISC processors are much simpler, so we use a RISC (Reduced Instruction Set Computer)
    - And why RISC-V? "All RISC processors are effectively the same except for one or two bad design decisions that 'seemed like a good idea at the time'", and RISC-V is new enough that we don't know of any that only ***seemed*** like a good idea...
- Our hardware design is in Logisim (schematics)
  - If you ever do hardware design, you'll only use schematics for circuit boards, everything else is Verilog or VHDL...
    - But its harder to get up to speed on those

And Still More

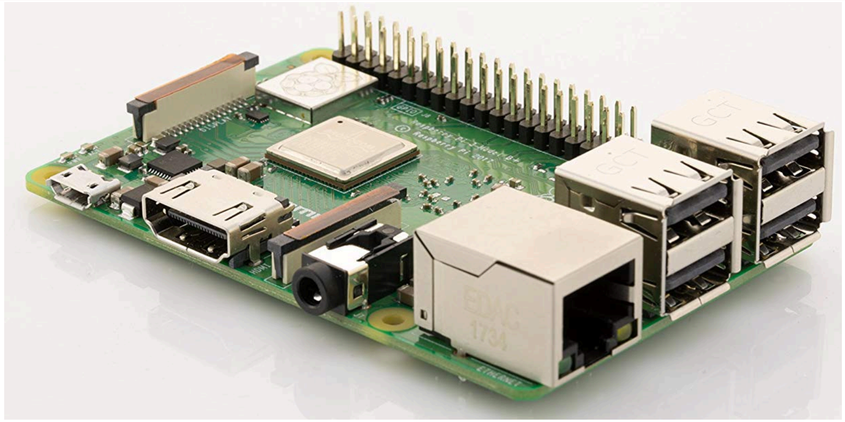
- Intel x86
  - A much uglier CISC assembly...
  - We won't do ***much*** in it, but you may have to write a little in-line assembly for the performance programming project
- Go ("golang" if you are doing Google searches)
  - A very modern language focused on concurrency:
    Being able to do many different things at the same time
  - Will be used for the final project, so look at https://tour.golang.org/ to get a feel for it.

Modern 61C: From the small...

To the Big...

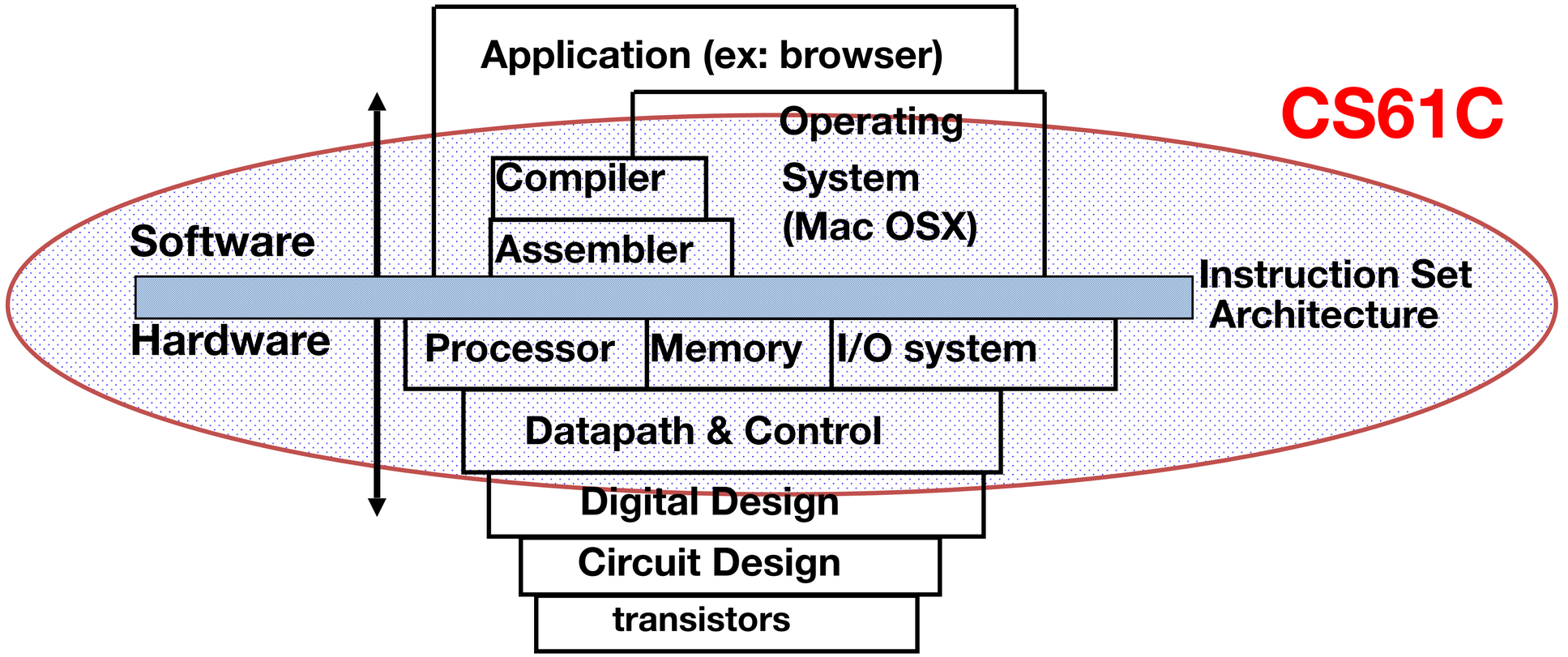

warehouse-scale
computer
My other computer
is a data center

And Parallelism is ***Everywhere***



- Raspberry Pi III B+
  - Quad core processor at 1.4 GHz
    - Each core is 2-issue superscalar
  - Plus 1/2 GB RAM, 4x USB, Gigabit Ethernet, HDMI...
  - $39!
- Compare with a Cray-1 from 1975:
  - 8 MB RAM, 80 MHz processor, 300MB storage, $5M+
- Or modern high end servers:
  - You can get a 2u server which supports 4 processors
    - And each processor can have 20+ cores, so 80 processor cores!

Old School Machine Structures



New School 61C: From the Data Center to the Gate

- Parallel Requests

  Assigned to computer

  e.g., Search “cats”
- Parallel Threads

  Assigned to core

  e.g., Lookup, Ads
- Parallel Instructions

  >1 instruction @ one time

  e.g., 5 pipelined instructions
- Parallel Data

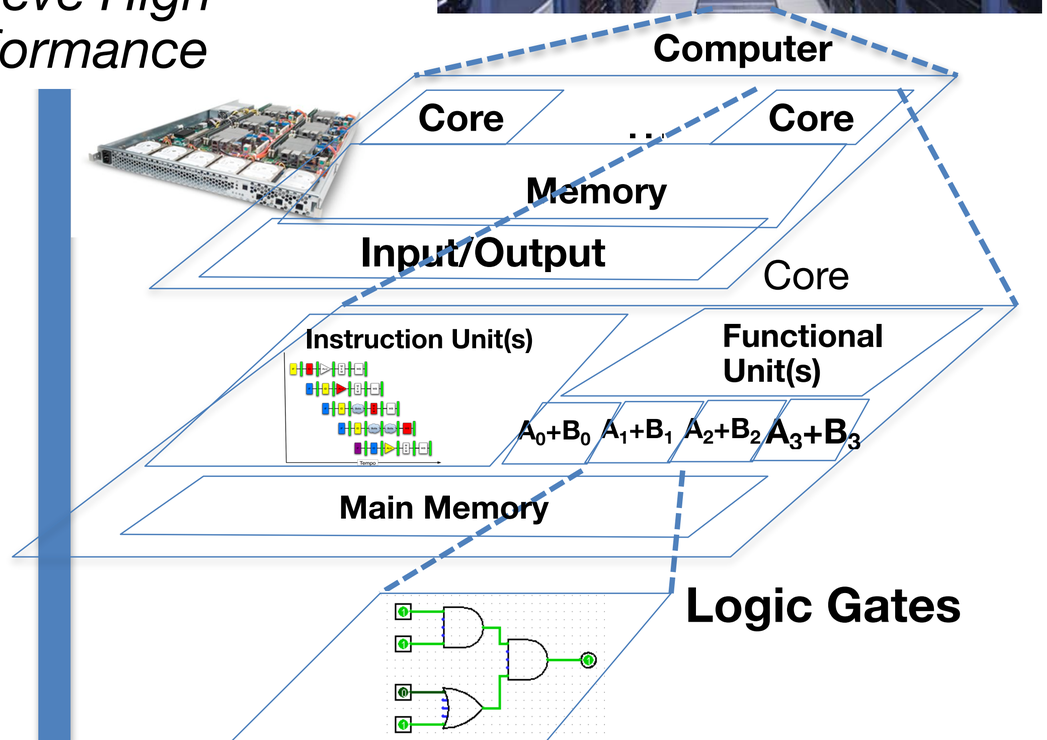
  >1 data item @ one time

  e.g., Add of 4 pairs of words
- Hardware descriptions

  All gates functioning in parallel at same time

Software *Hardware*

Harness Parallelism & Achieve High Performance

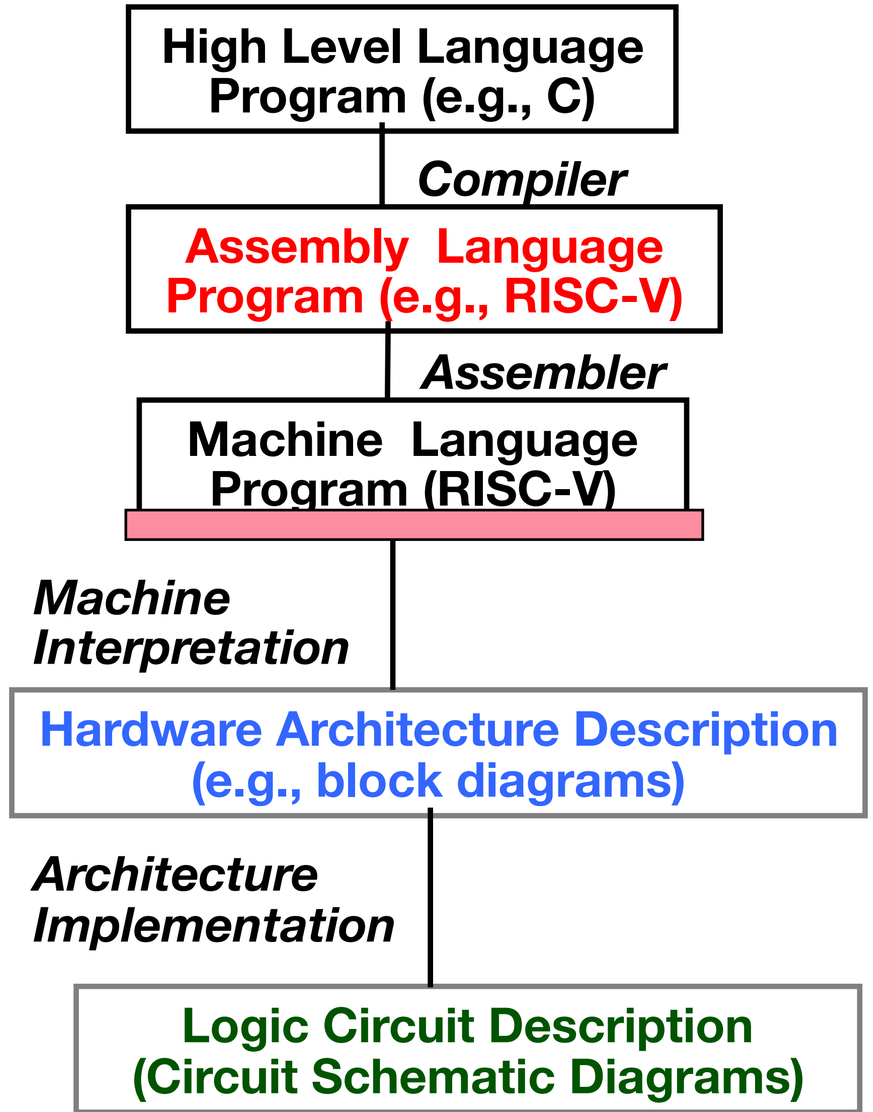


5 Great Ideas in Computer Architecture

1. Abstraction
   (Layers of Representation/Interpretation)
2. Moore’s Law (Designing through trends)
3. Principle of Locality (Memory Hierarchy)
4. Parallelism & Amdahl's law (which limits it)
5. Dependability via Redundancy

Great Idea #1: Abstraction (Levels of Representation/Interpretation)

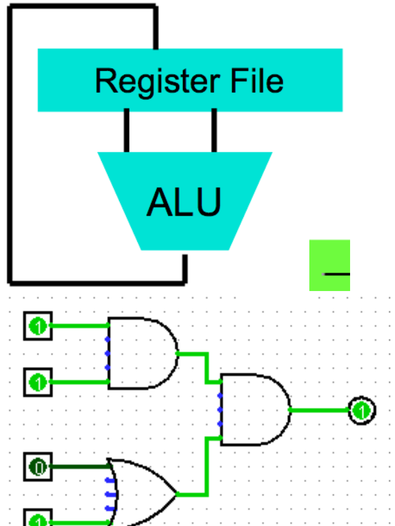

```
lw   t0, t2, 0
lw   t1, t2, 4
sw   t1, t2, 0
sw   t0, t2, 4
```




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

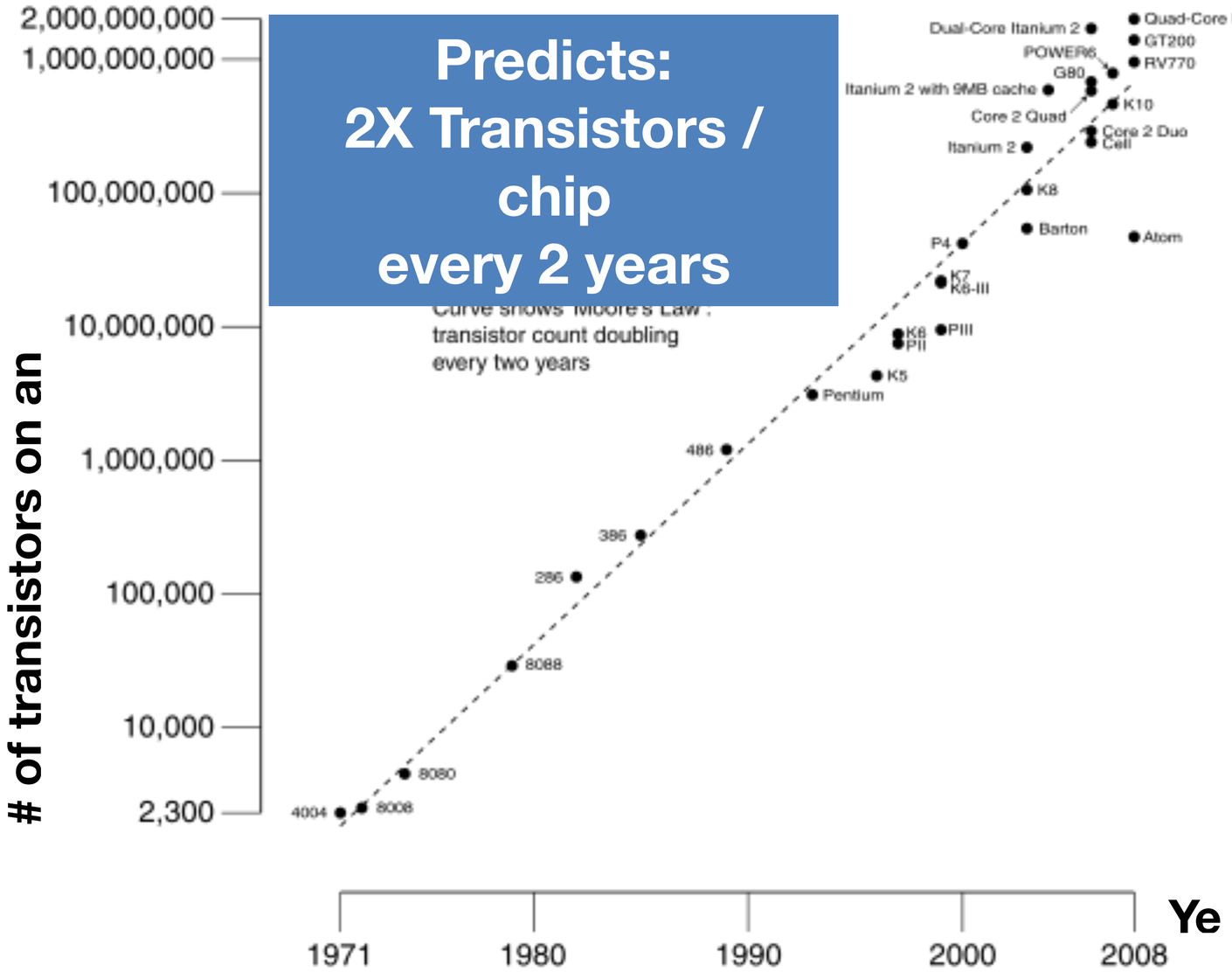
Anything can be represented as a *number*, i.e., data or instructions

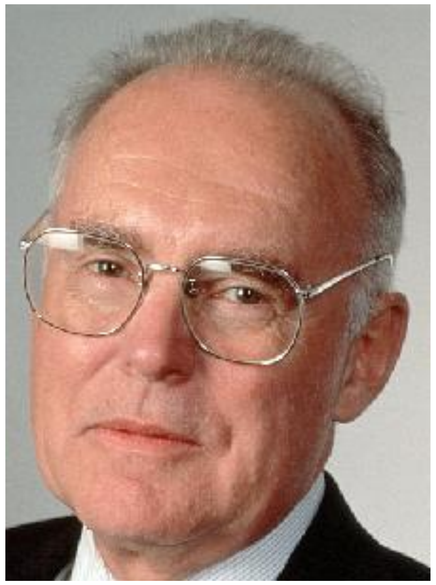
```
0000 1001 1100 0110 1010 )0
1010 1111 0101 1000 0000 .0
1100 0110 1010 1111 0101 )1
0101 1000 0000 1001 1100 .1
```



#2: Moore's Law

Predicts:
2X Transistors / chip
every 2 years

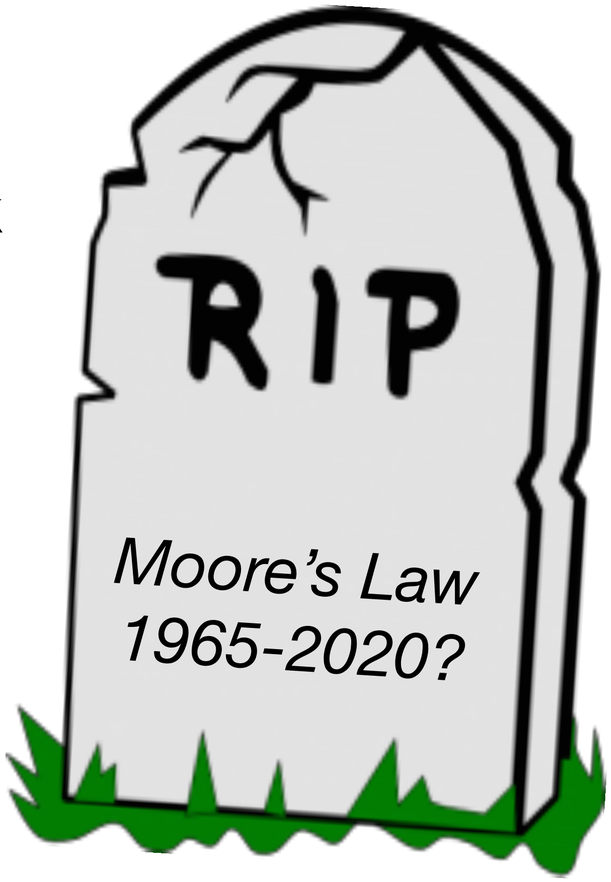




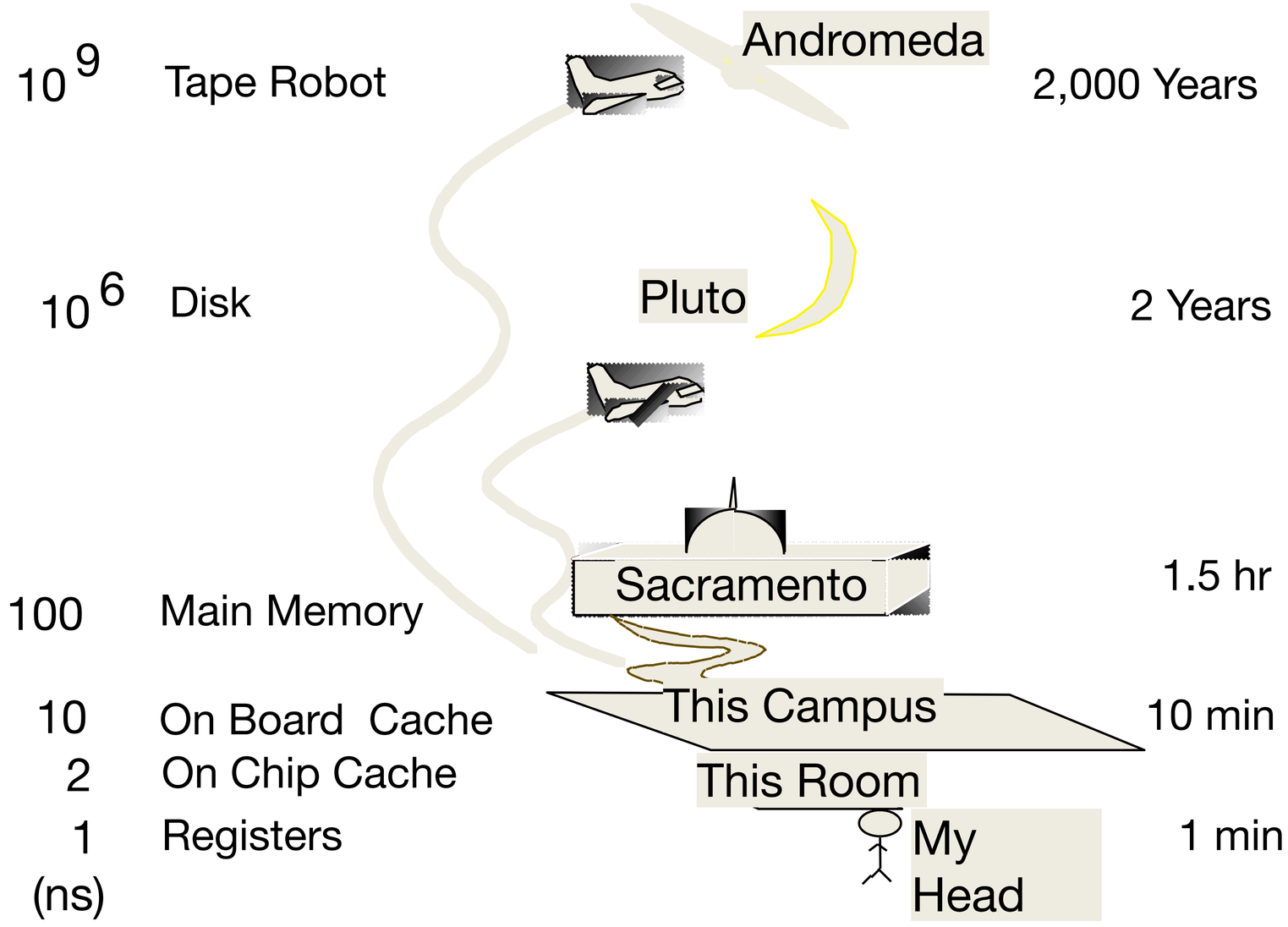
Gordon Moore
Intel Cofounder
B.S. Cal 1950!

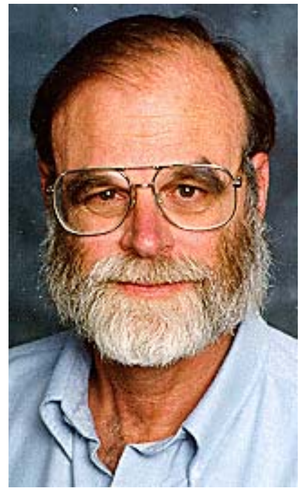
Interesting Times

- Moore's Law meant that the cost of transistors scaled down as technology scaled to smaller and smaller feature sizes.
  - And the resulting transistors resulted in increased single-task performance
- But single-task performance improvements hit a brick wall years ago...
- And now the newest, smallest fabrication processes <14nm, might have greater cost/transistor!!!! So, why shrink????



Jim Gray's Storage Latency Analogy: How Far Away is the Data?

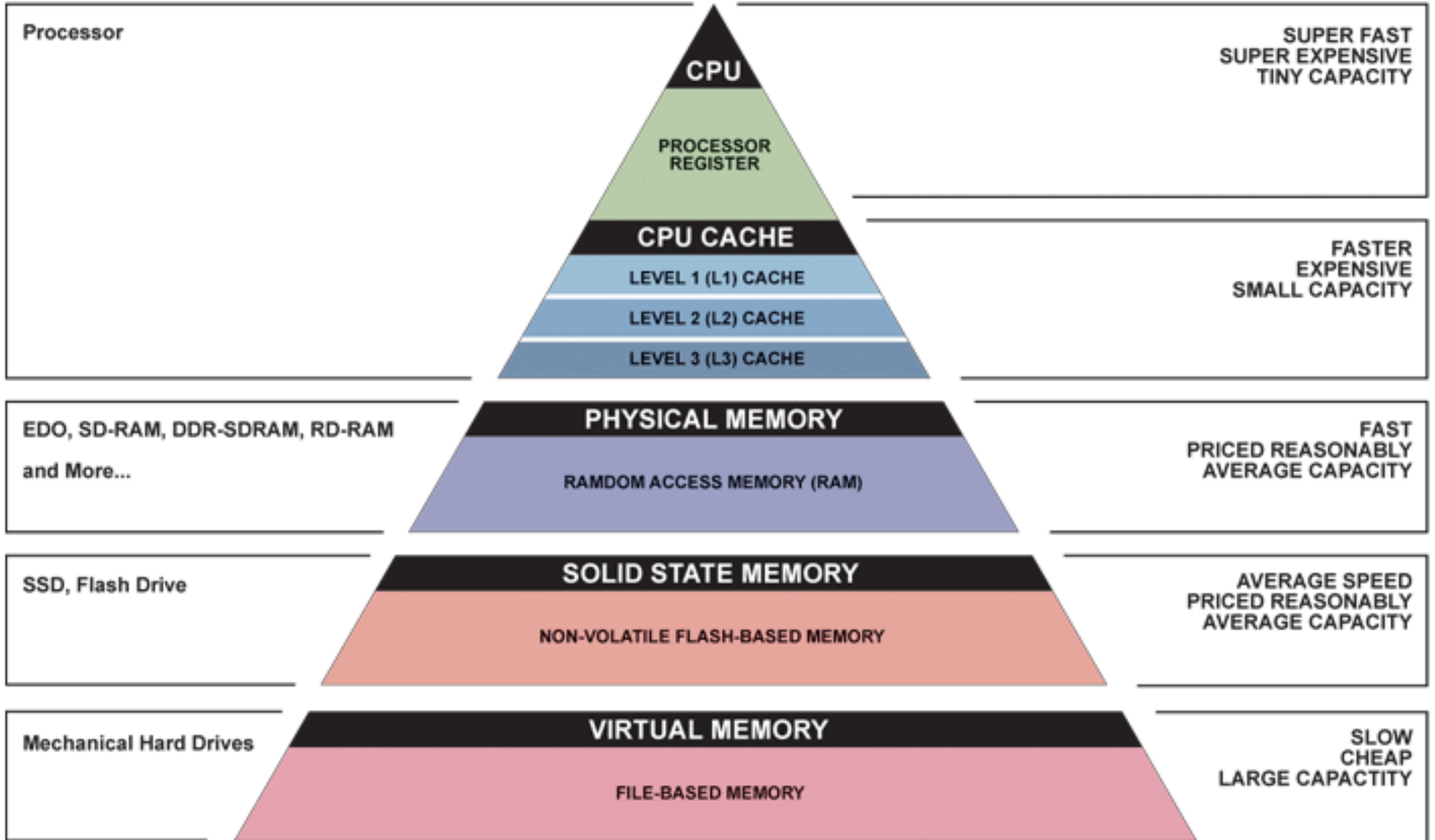




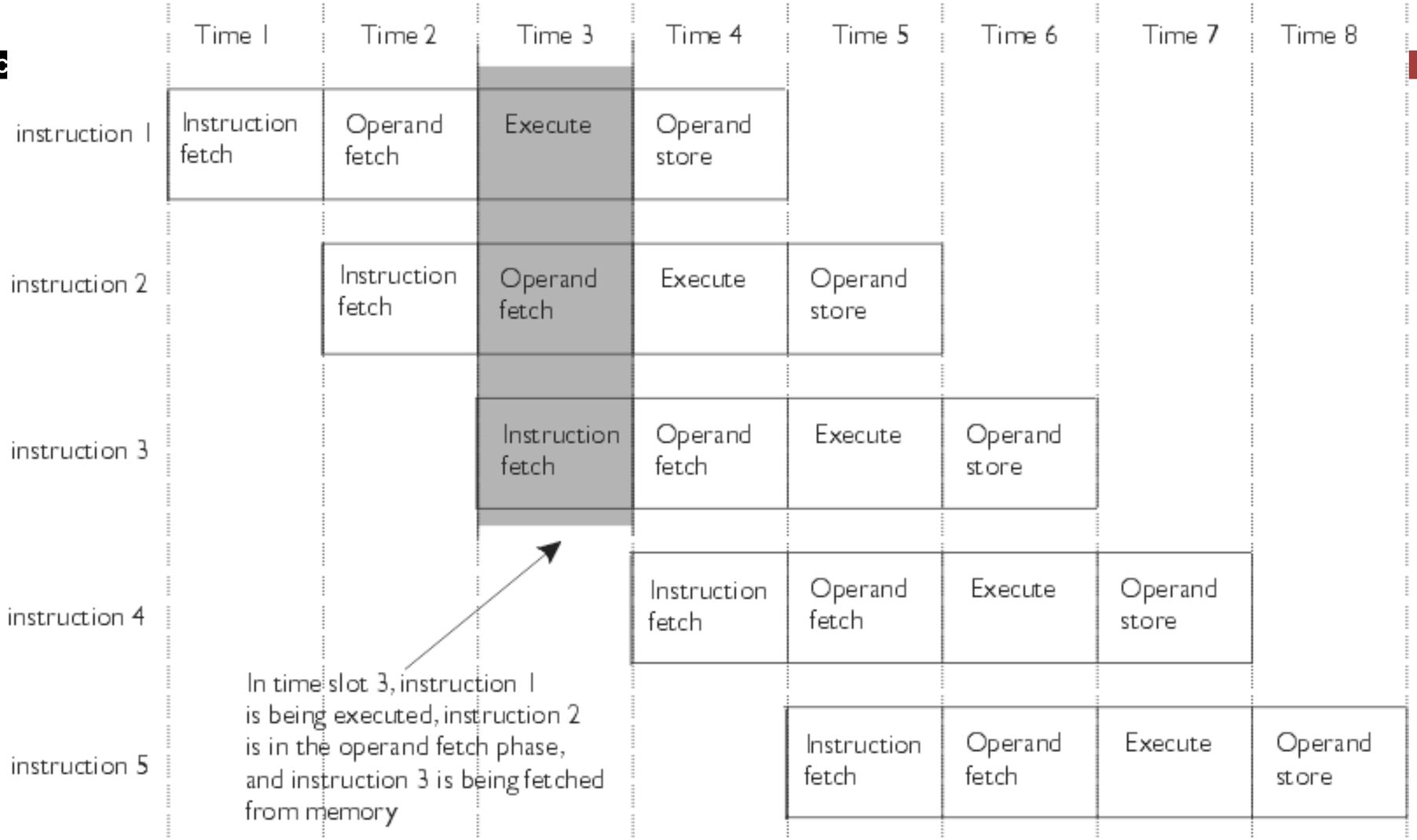
Jim Gray
Turing Award
B.S. Cal 1966
Ph.D. Cal 1969!

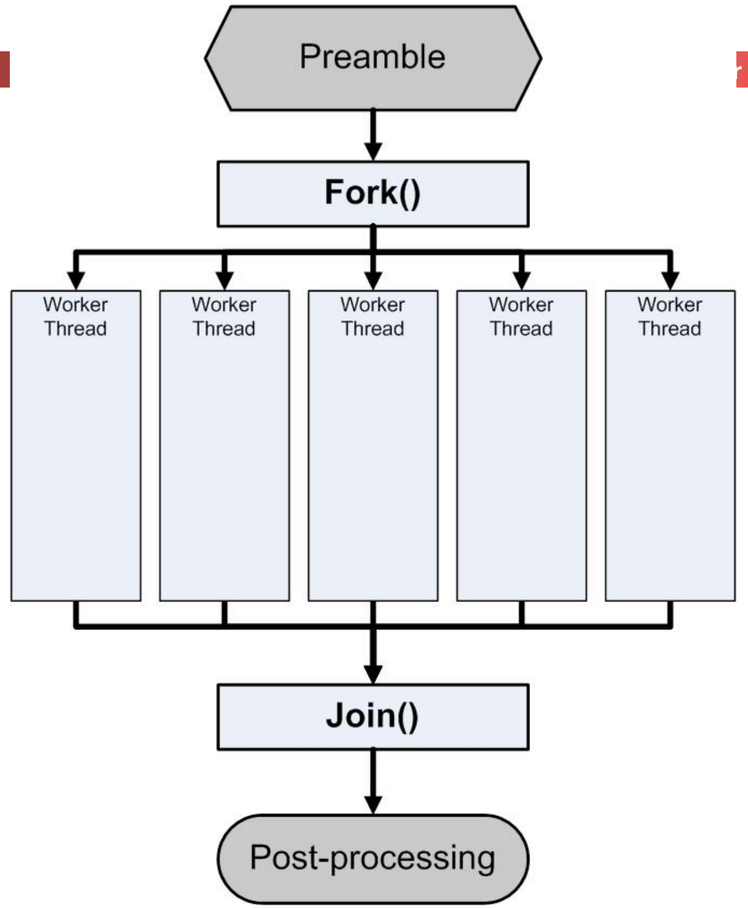
Great Idea #3: Principle of Locality/ Memory Hierarchy

*Memory Hierarchy effectively creates a **large fast cheap** memory.*



Great Idea #4: Parallelism





The Caveat: Amdahl's Law

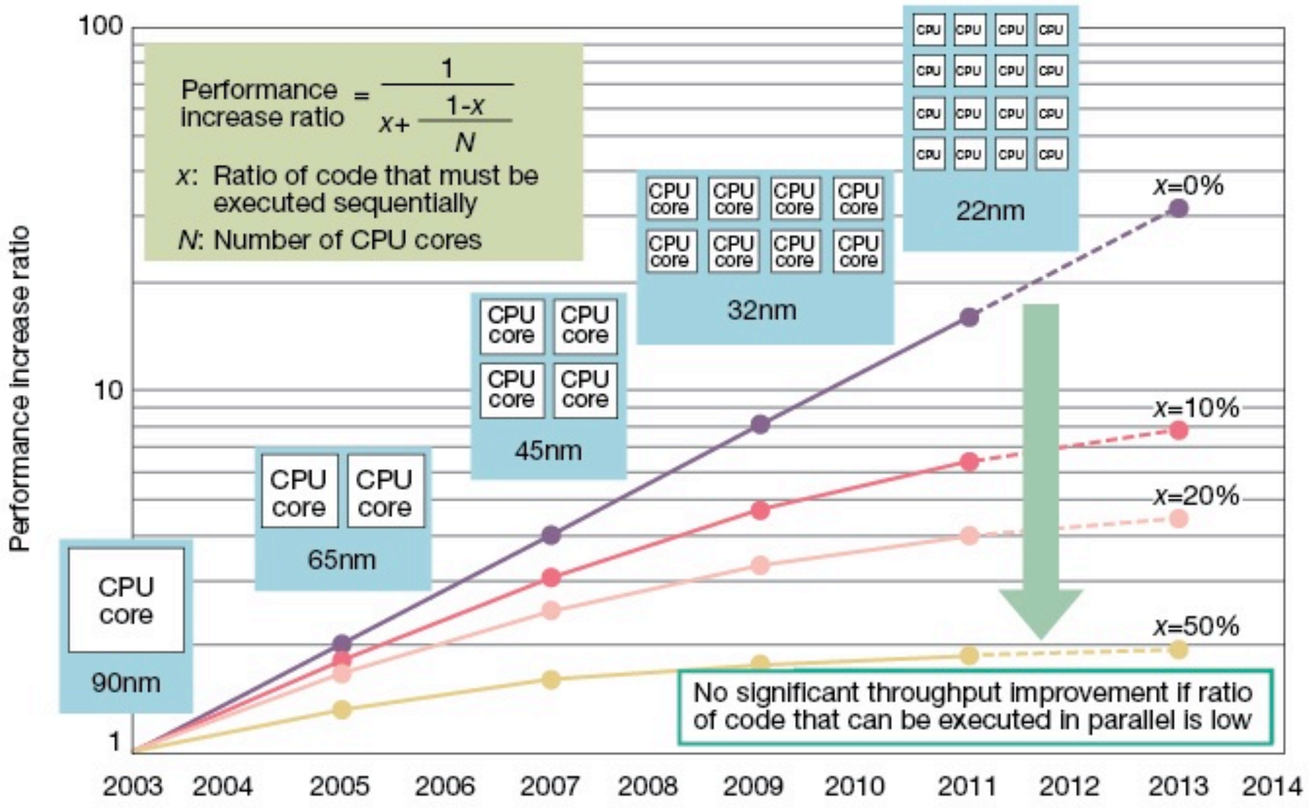


Fig 3 Amdahl's Law an Obstacle to Improved Performance Performance will not rise in the same proportion as the increase in CPU cores. Performance gains are limited by the ratio of software processing that must be executed sequentially. Amdahl's Law is a major obstacle in boosting multicore microprocessor performance. Diagram assumes no overhead in parallel processing. Years shown for design rules based on Intel planned and actual technology. Core count assumed to double for each rule generation.

Gene Amdahl
Computer Pioneer

Great Idea #5: Failures Happen, so...

- 4 disks/server, 50,000 servers
- Failure rate of disks: 2% to 10% / year
  - Assume 4% annual failure rate
- On average, how often does a disk fail?

a) 1 / month
b) 1 / week
c) 1 / day
d) 1 / hour

Coping with Failures

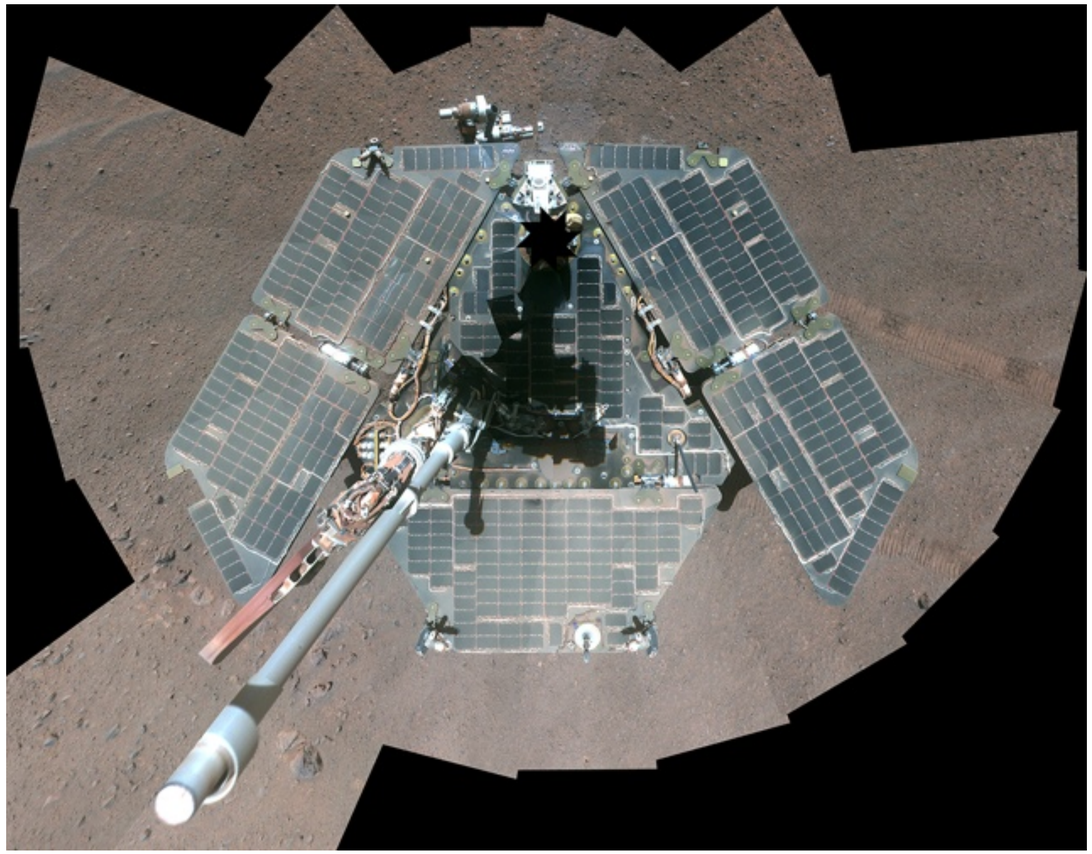
- 4 disks/server, 50,000 servers
- Failure rate of disks: 2% to 10% / year
  - Assume 4% annual failure rate
- On average, how often does a disk fail?

a) 1 / month
b) 1 / week
c) 1 / day
d) 1 / hour

50,000 x 4 = 200,000 disks
200,000 x 4% = 8000 disks fail
365 days x 24 hours = 8760 hours

NASA Fixing Rover's Flash Memory

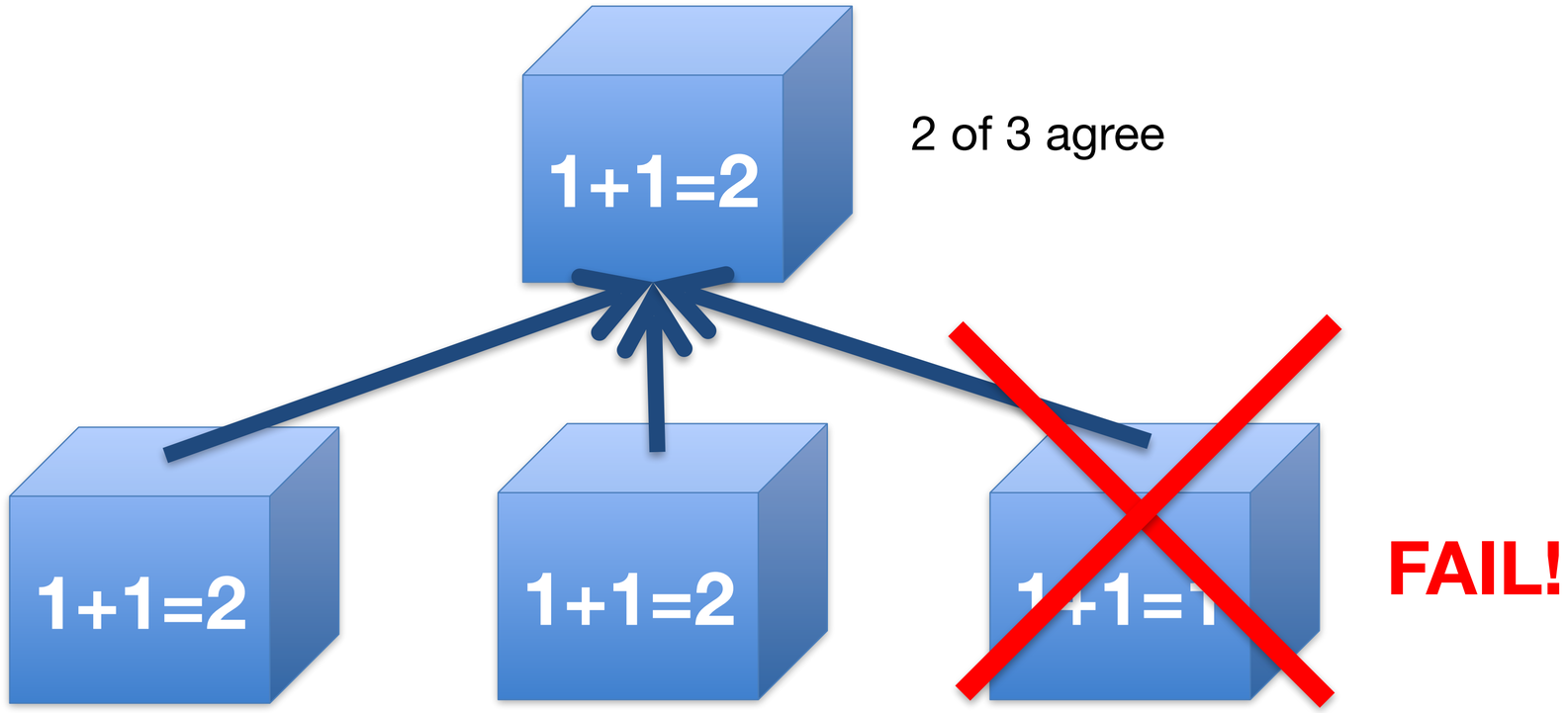
- "Opportunity" (NASA's rover) still active on Mars after >10 years
- But flash memory worn out
- New software update to avoid using worn out memory banks



http://www.engadget.com/2014/12/30/nasa-opportunity-rover-flash-fi

Great Idea #5: Dependability via Redundancy

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



Increasing transistor density reduces the cost of redundancy

Great Idea #5: Dependability via Redundancy

- Applies to everything from datacenters to storage to memory to instructors
  - **Redundant datacenters** so that can lose 1 datacenter but Internet service stays online
  - **Redundant computers** was Google's original internal innovation
  - **Redundant disks** so that can lose 1 disk but not lose data (Redundant Arrays of Independent Disks/RAID)
  - **Redundant memory bits** so that can lose 1 bit but no data (Error Correcting Code/ECC Memory/"Chipkill" memory)
  - **Redundant instructors** (I wish... I'd want to call in a "hot spare" if I have to travel!)

Summary

- CS61C: Learn 5 great ideas in computer architecture to enable high performance programming via parallelism, not just learn C

1. Abstraction
   (Layers of Representation/Interpretation)
2. Moore's Law
3. Principle of Locality/Memory Hierarchy
4. Parallelism
5. Dependability via Redundancy

Within the Computer: Everything is a Number.

- But numbers usually stored with a fixed size
  - 8-bit bytes, 16-bit half words, 32-bit words, 64-bit double words, …
  - And there are really only two primitive "numbers": 0 and 1 is a "bit" (BInary digiT). A byte is 8 bits
- Integer and floating-point operations can lead to results too big/small to store within their representations: *overflow/ underflow*

Number Representation

- Value of i-th digit is $d \times Base^i$ where i starts at 0 and increases from right to left:
- $123_{10} = 1_{10} \times 10_{10}^2 + 2_{10} \times 10_{10}^1 + 3_{10} \times 10_{10}^0$

  $= 1\text{x}100_{10} + 2\text{x}10_{10} + 3\text{x}1_{10}$

  $= 100_{10} + 20_{10} + 3_{10}$

  $= 123_{10}$
- Binary (Base 2), Hexadecimal (Base 16), Decimal (Base 10) different ways to represent an integer
  - We'll use 1_{two}, 5_{ten}, 10_{hex} to be clearer

    (vs. 1_2, 4_8, 5_{10}, 10_{16})

Number Representation

- Hexadecimal digits: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
- $FFF_{hex} = 15_{ten} \times 16_{ten}^{2} + 15_{ten} \times 16_{ten}^{1} + 15_{ten} \times 16_{ten}^{0}$
  $= 3840_{ten} + 240_{ten} + 15_{ten}$
  $= 4095_{ten}$
- $1111\ 1111\ 1111_{two} = FFF_{hex} = 4095_{ten}$

$= 1x2^{11} + 1x2^{10} + 1x2^{8} + 1x2^{7} + 1x2^{6} + 1x2^{5} + 1x2^{4} + 1x2^{3} + 1x2^{2} + 1x2^{1} + 1x2^{0}$

$= 2048_{ten} + 1024_{ten} + 512_{ten} + 256_{ten} + 128_{ten} + 64_{ten} + 32_{ten} + 16_{ten} + 8_{ten} + 4_{ten} + 2_{ten} + 1_{ten}$

$= 4095_{ten}$

(May put blanks every group of binary or hexadecimal digits to make it easier to parse, like commas in decimal)

Signed and Unsigned Integers

- C, C++, and Java have *signed integers*, e.g., 7, -255:

```
int x, y, z;
```

- C, C++ also have *unsigned integers*, which are used for addresses
- 32-bit word can represent 2^{32} binary numbers
- Unsigned integers in 32 bit word represent 0 to 2^{32}-1 (4,294,967,295)

Unsigned Integers

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = 0_{ten}$

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 1_{ten}$

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = 2_{ten}$

... ...

$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{two} = 2{,}147{,}483{,}645_{ten}$

$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = 2{,}147{,}483{,}646_{ten}$

$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = 2{,}147{,}483{,}647_{ten}$

$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = 2{,}147{,}483{,}648_{ten}$

$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 2{,}147{,}483{,}649_{ten}$

$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = 2{,}147{,}483{,}650_{ten}$

... ...

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{two} = 4{,}294{,}967{,}293_{ten}$

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = 4{,}294{,}967{,}294_{ten}$

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = 4{,}294{,}967{,}295_{ten}$

Signed Integers and Two's-Complement Representation

- Signed integers in C; want ½ numbers <0, want ½ numbers >0, and want one 0
- *Two's complement* treats 0 as positive, so 32-bit word represents 2^{32} integers from -2^{31} (–2,147,483,648) to $2^{31}-1$ (2,147,483,647)
  - Note: one negative number with no positive version
  - Book lists some other options, all of which are worse
  - Every computer uses two's complement today
- *Most-significant bit* (leftmost) is the *sign bit*, since 0 means positive (including 0), 1 means negative
  - Bit 31 is most significant, bit 0 is least significant

Two's-Complement Integers

Sign Bit

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

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

Ways to Make Two's Complement

- *In two's complement the sign-bit has negative weight:*
- So the value of an N-bit word $[b_{N-1}\ b_{N-2}\ \ldots\ b_1\ b_0]$ is:

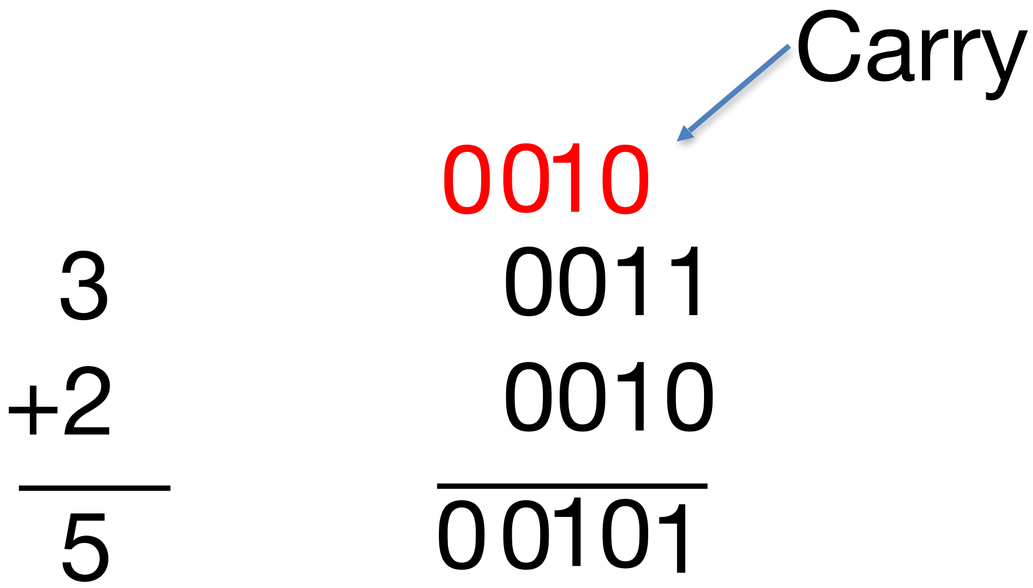
$$-2^{N-1} \times b^{N-1} + 2^{N-2} \times b^{N-2} + \ldots + 2^1 \times b^1 + 2^0 \times b^0$$

- For a 4-bit number, $3_{ten}=0011_{two}$, its two's complement $-3_{ten} = 1101_{two}$ $(-1000_{two} + 0101_{two} = -8_{ten} + 5_{ten})$

- Here is an easier way:
  - Invert all bits and add 1
  - Computers circuits do it like this, too

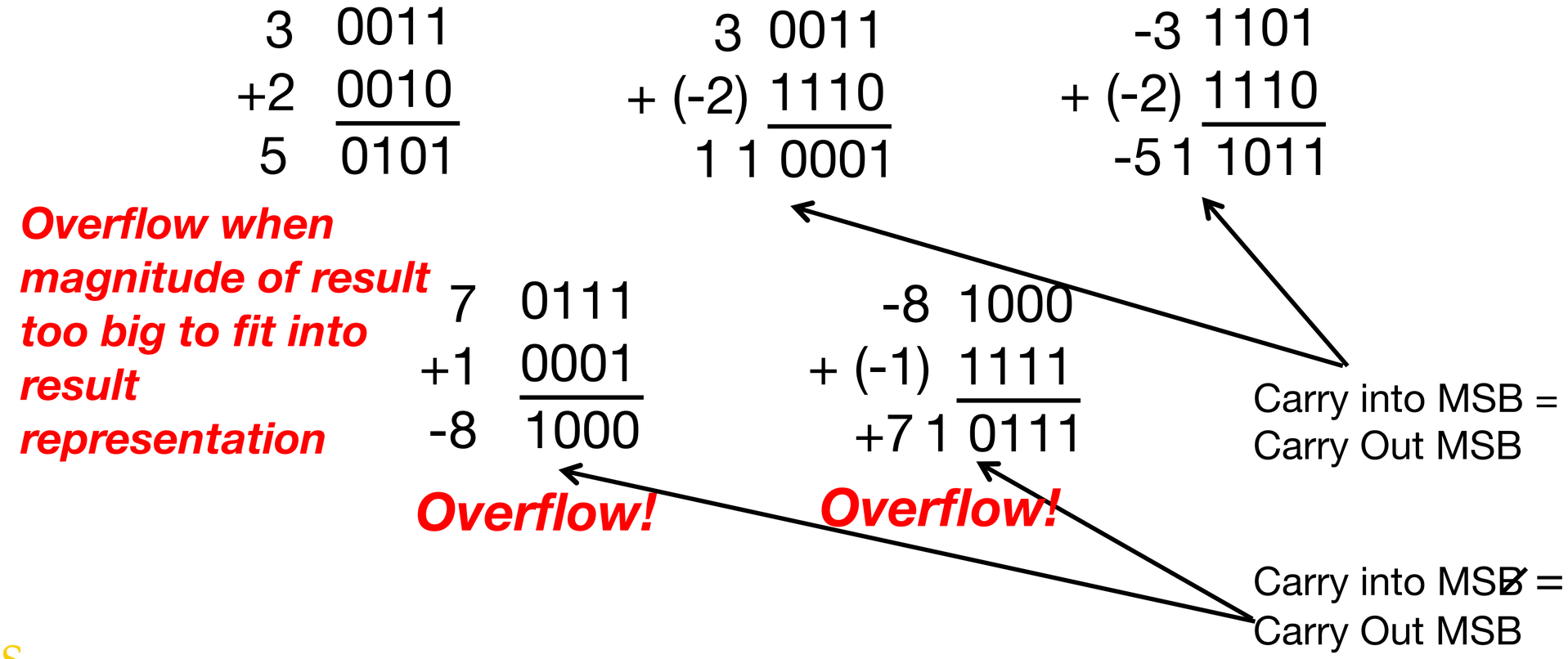
3_{ten}	0011_{two}
Bitwise Invert	1100_{two}
+	1_{two}
-3_{ten}	1101_{two}

Binary Addition Example



Two's-Complement Examples

- Assume for simplicity 4 bit width, -8 to +7 represented



Suppose we had a 5-bit word.
What integers can be represented in two's complement?

- -32 to +31
- 0 to +31
- -16 to +15
- -15 to +16

Suppose we had a 5-bit word.
What integers can be represented in two's complement?

- ☐ -32 to +31
- ☐ 0 to +31
- ☐ -16 to +15
- ☐ -15 to +16

Summary: Number Representations

- Everything in a computer is a number, in fact only 0 and 1.
- Integers are interpreted by adhering to fixed length
- Negative numbers are represented with Two's complement
- Overflows can be detected utilizing the carry bit.