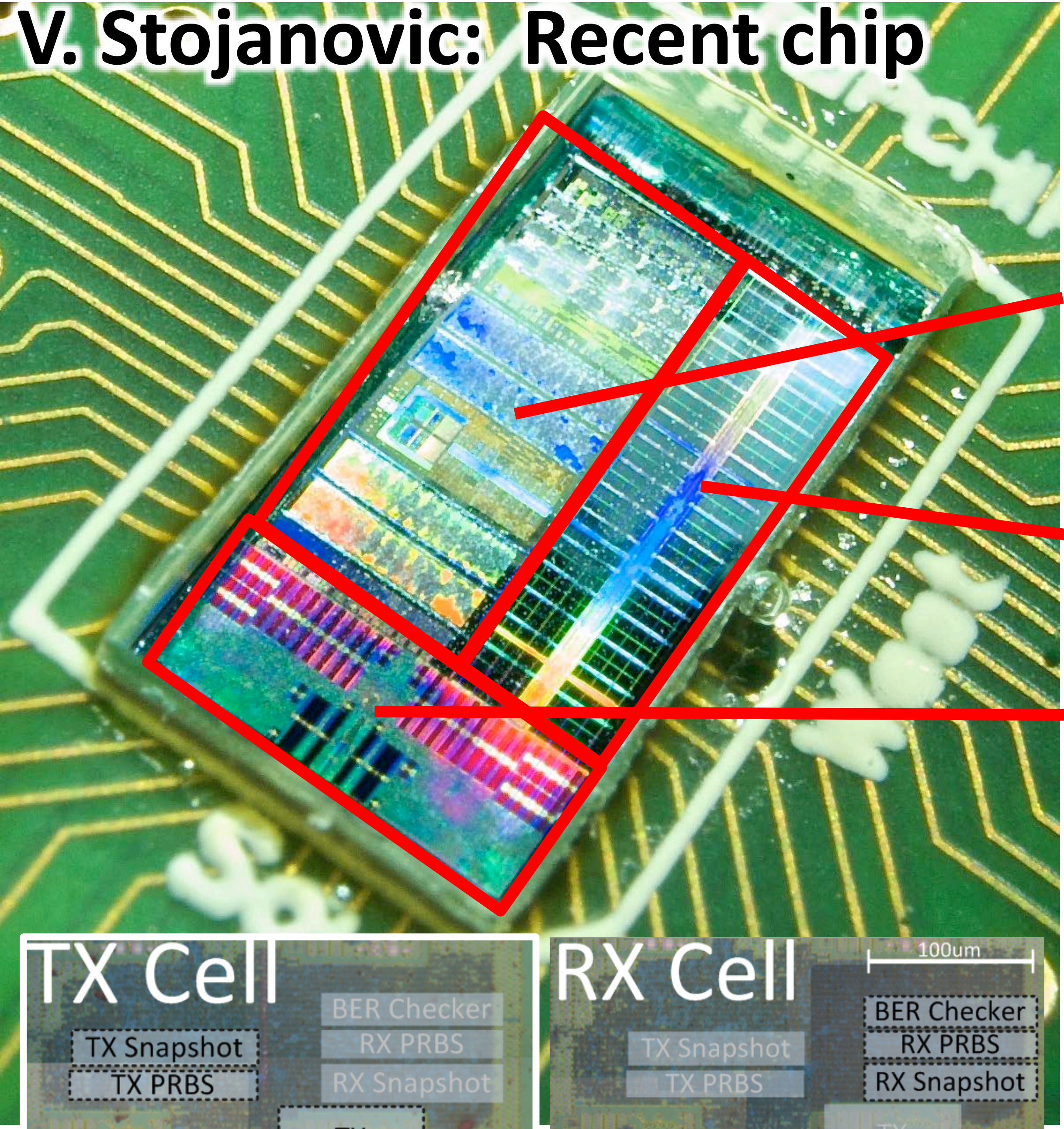
CS 61C Great Ideas in Computer Architecture (a.k.a. Machine Structures) Lecture 1: *Course Introduction*

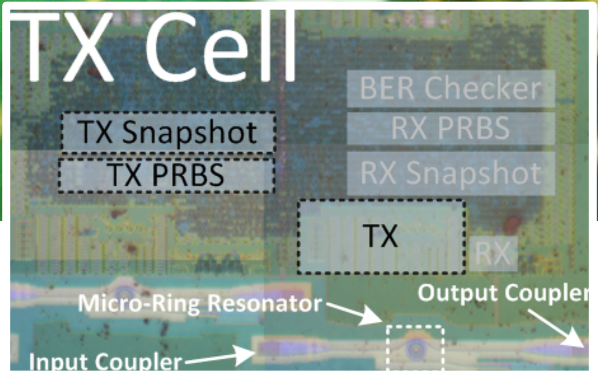
Instructors:

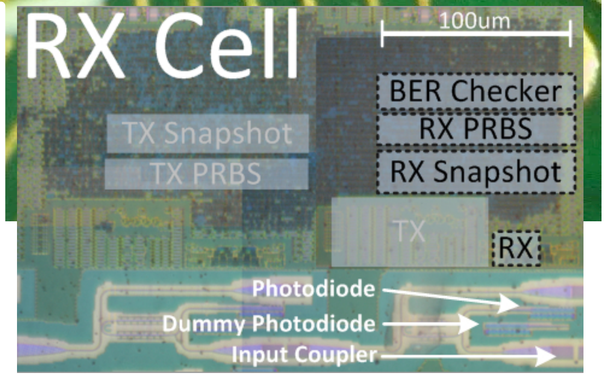
Professor John Wawrzynek (call me "John")
Professor Vladimir Stojanovic (call me "Vladimir")
(lots of help from **TAs,** esp. **Head TAs Fred and William**)

`http://inst.eecs.berkeley.edu/~cs61c/`

V. Stojanovic: Recent chip


Monolithically-Integrated Silicon Photonic Links
1MB SRAM Memory Structure for Testing
Dual-Core RISC-V Processor with Vector Accelerators





3mm X 6mm Chip
Fabricated in 45nm SOI
75m+ transistors

Monolithically-Integrated Silicon Photonic Links

1MB SRAM Memory Structure for Testing

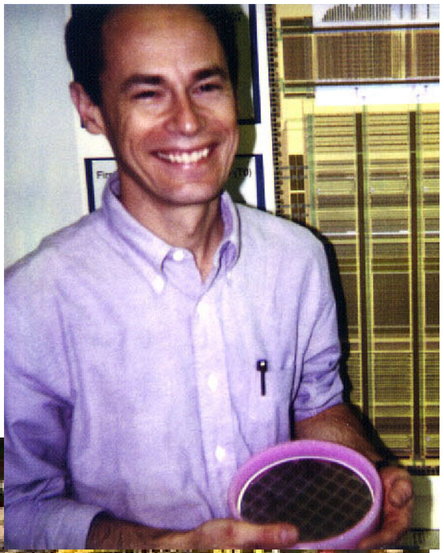
Dual-Core RISC-V Processor with Vector Accelerators

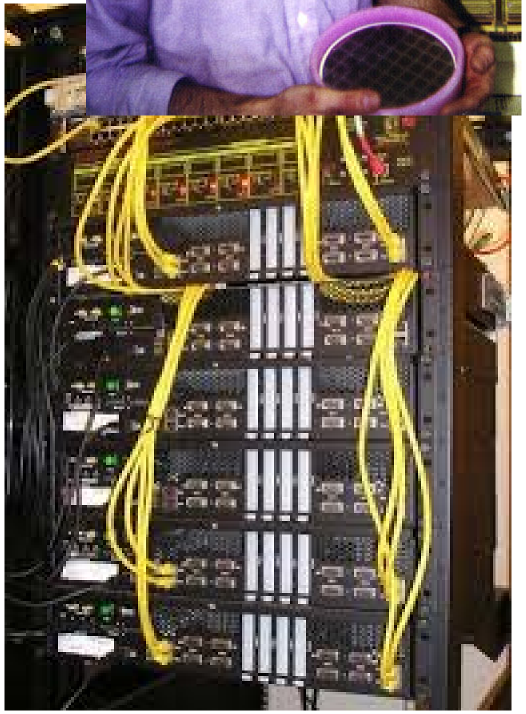
About me:
PhD Stanford – High-speed I/O
At Berkeley since 2013
At MIT since 2005

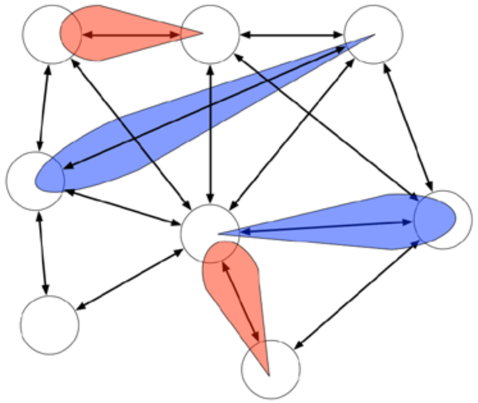
IC design, Sig. processing,
Chip design with new devices

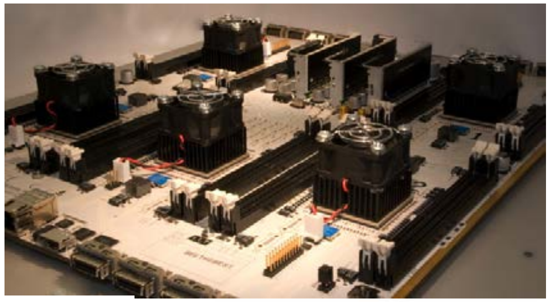
John Wawrzynek – Professor in EECS

- JPL/NASA – space craft data system
- PhD Caltech – electronic music
- Berkeley faculty since 1989
  - IC design, signal processing systems
  - High performance computer design
  - Wireless system design









bina TECHNOLOGIES

ANDES NETWORKS

BEEcube

Agenda

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

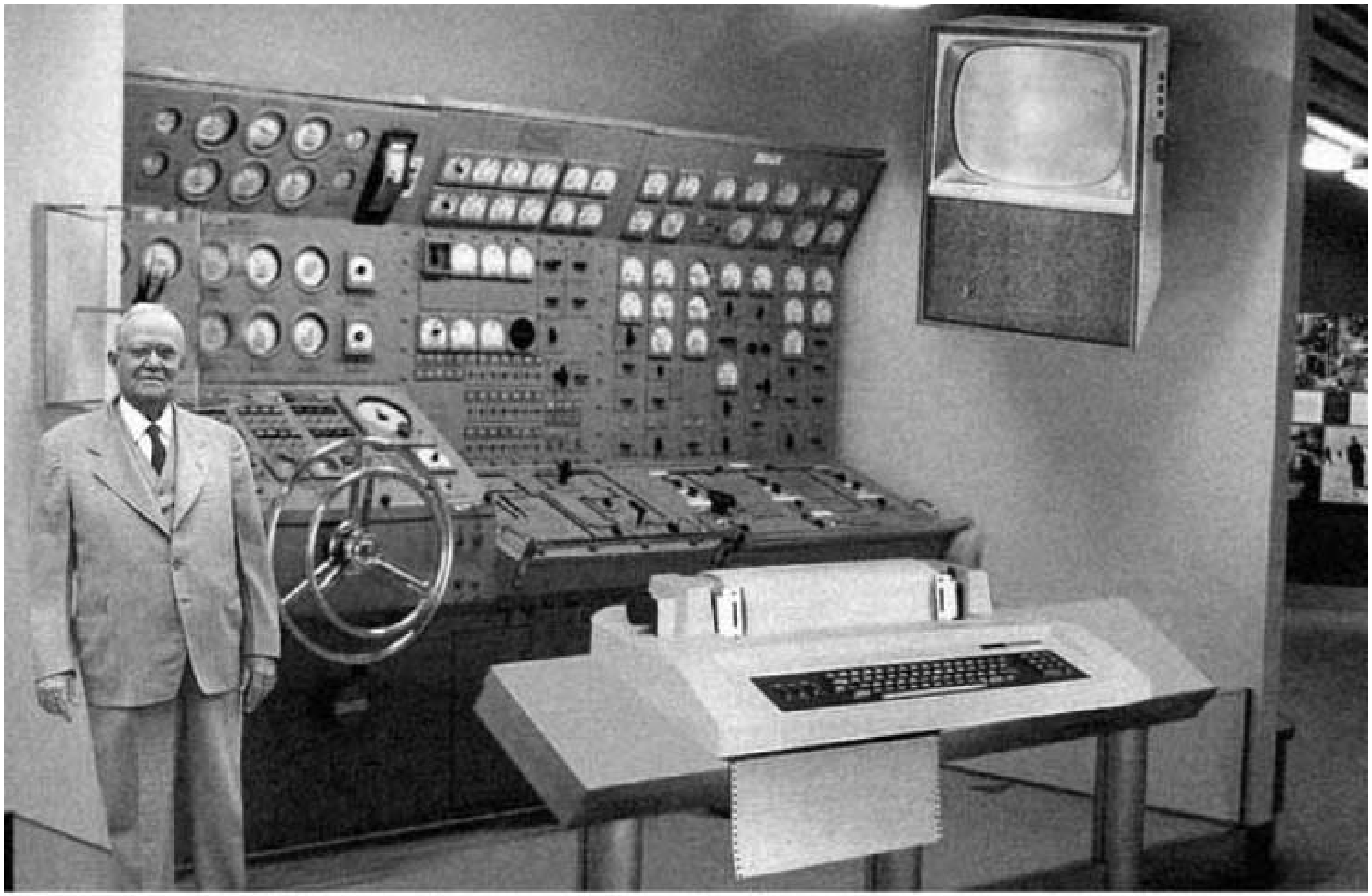
Agenda

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

CS61C is NOT really 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!
  - Allows us to talk about key hardware features in higher level terms
  - Allows programmer to explicitly harness underlying hardware parallelism for high performance

Old School CS61C

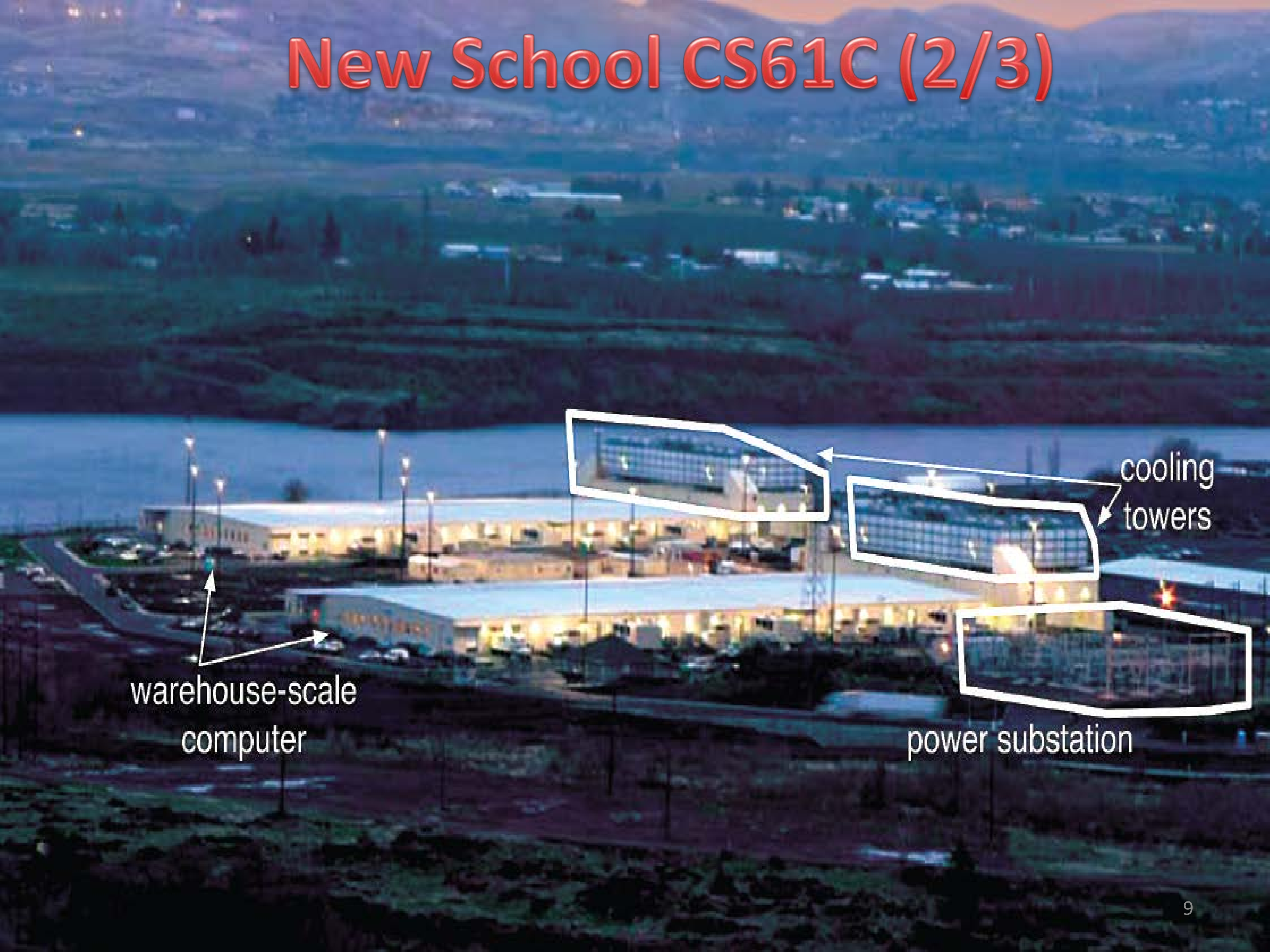


Scientists from the RAND Corporation have created this model to illustrate how a "home computer" could look like in the year 2004. However the needed technology will not be economically feasible for the average home. Also the scientists readily admit that the computer will require not yet invented technology to actually work. but 50 years from now scientific progress is expected to solve these problems. With teletype interface and the Fortran language, the computer will be easy to use.

New School CS61C (1/2)

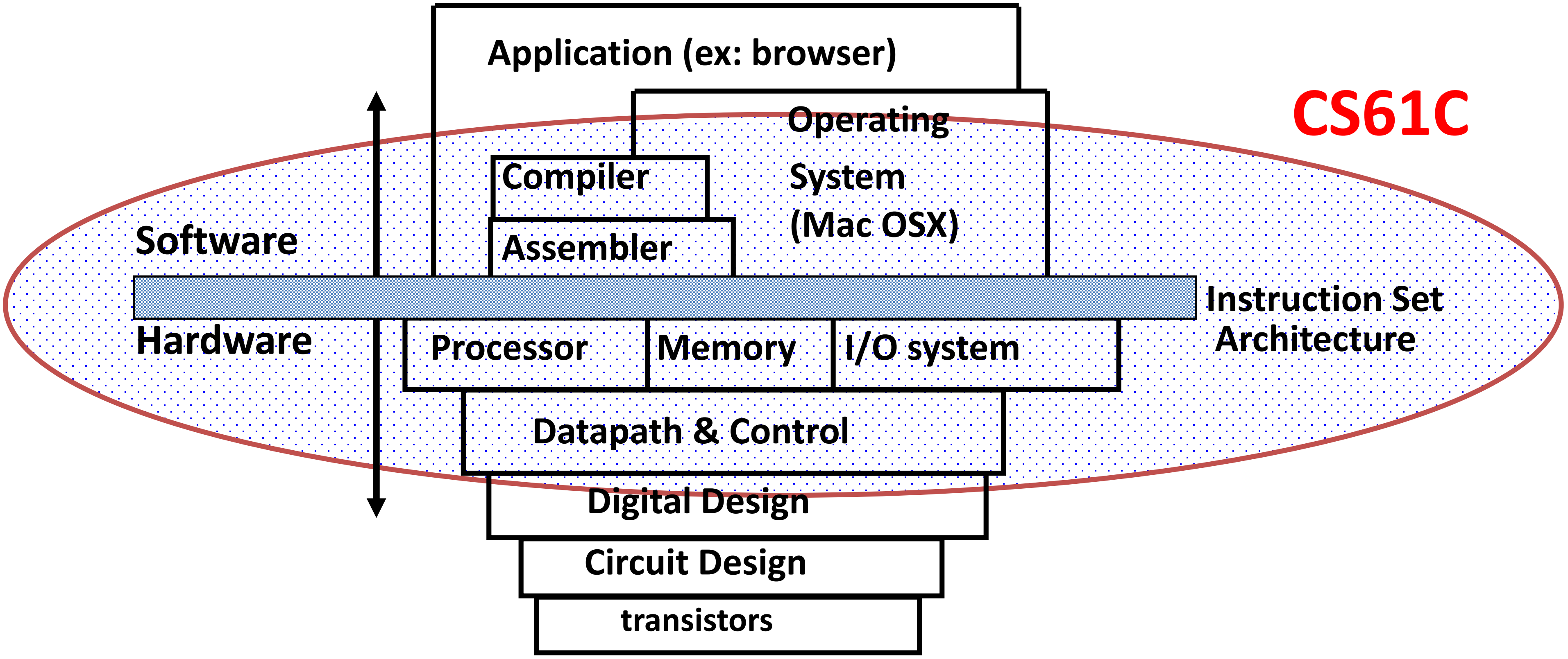
Personal
Mobile
Devices

New School CS61C (2/3)



New School CS61C (3/3)

Old School Machine Structures

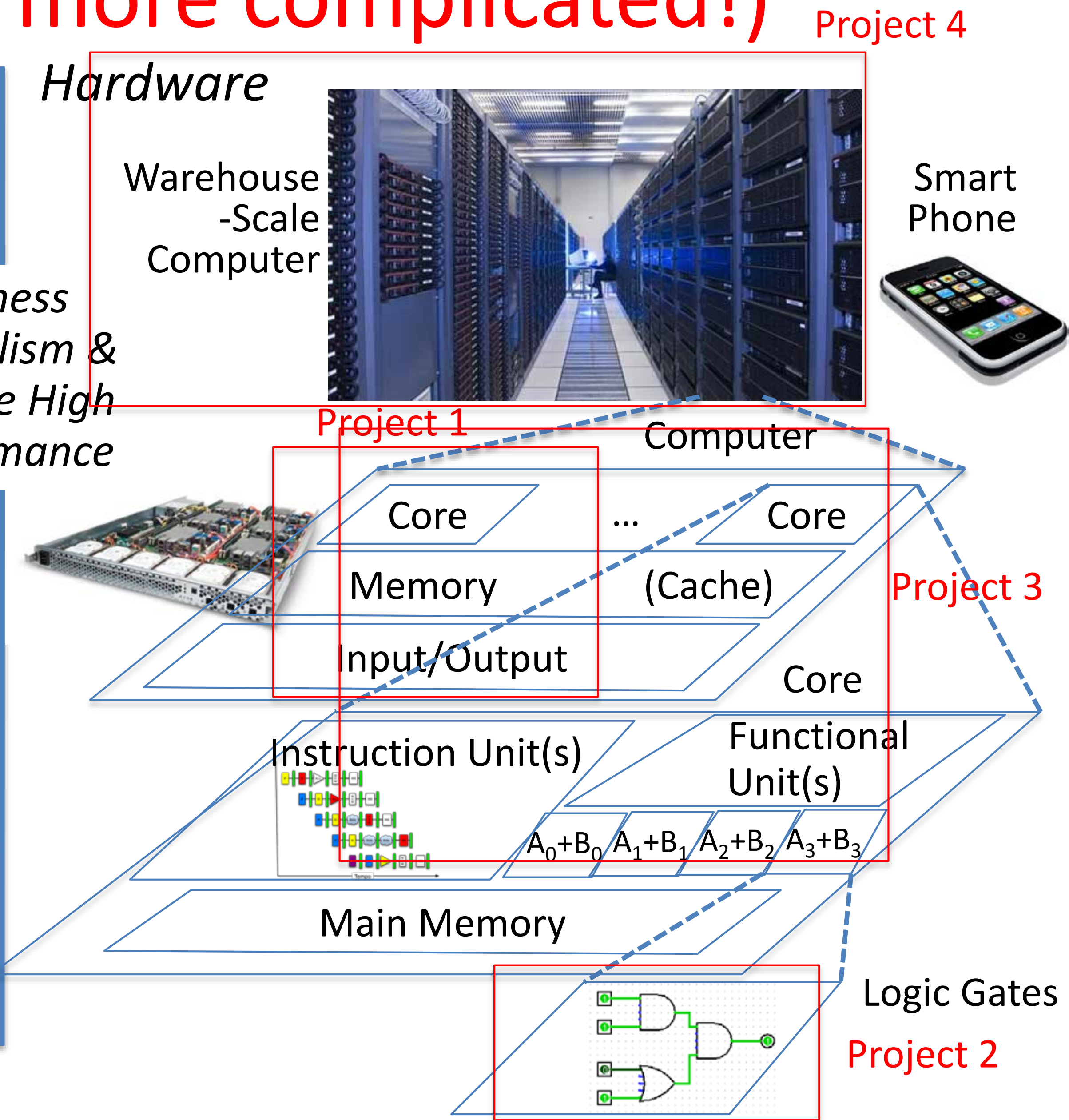


New-School Machine Structures (It’s a bit more complicated!)

Software | *Hardware*

- 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

Harness Parallelism & Achieve High Performance



Agenda

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

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
5. Dependability via Redundancy

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

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

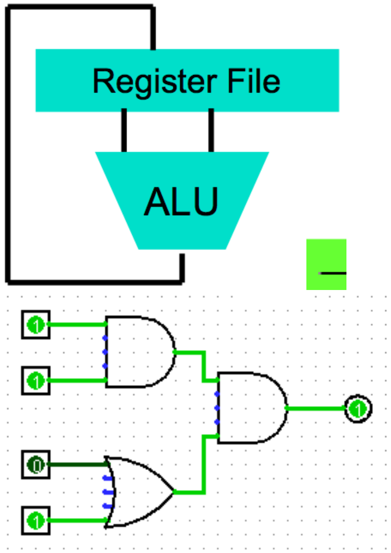
Logic Circuit Description (Circuit Schematic Diagrams)

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

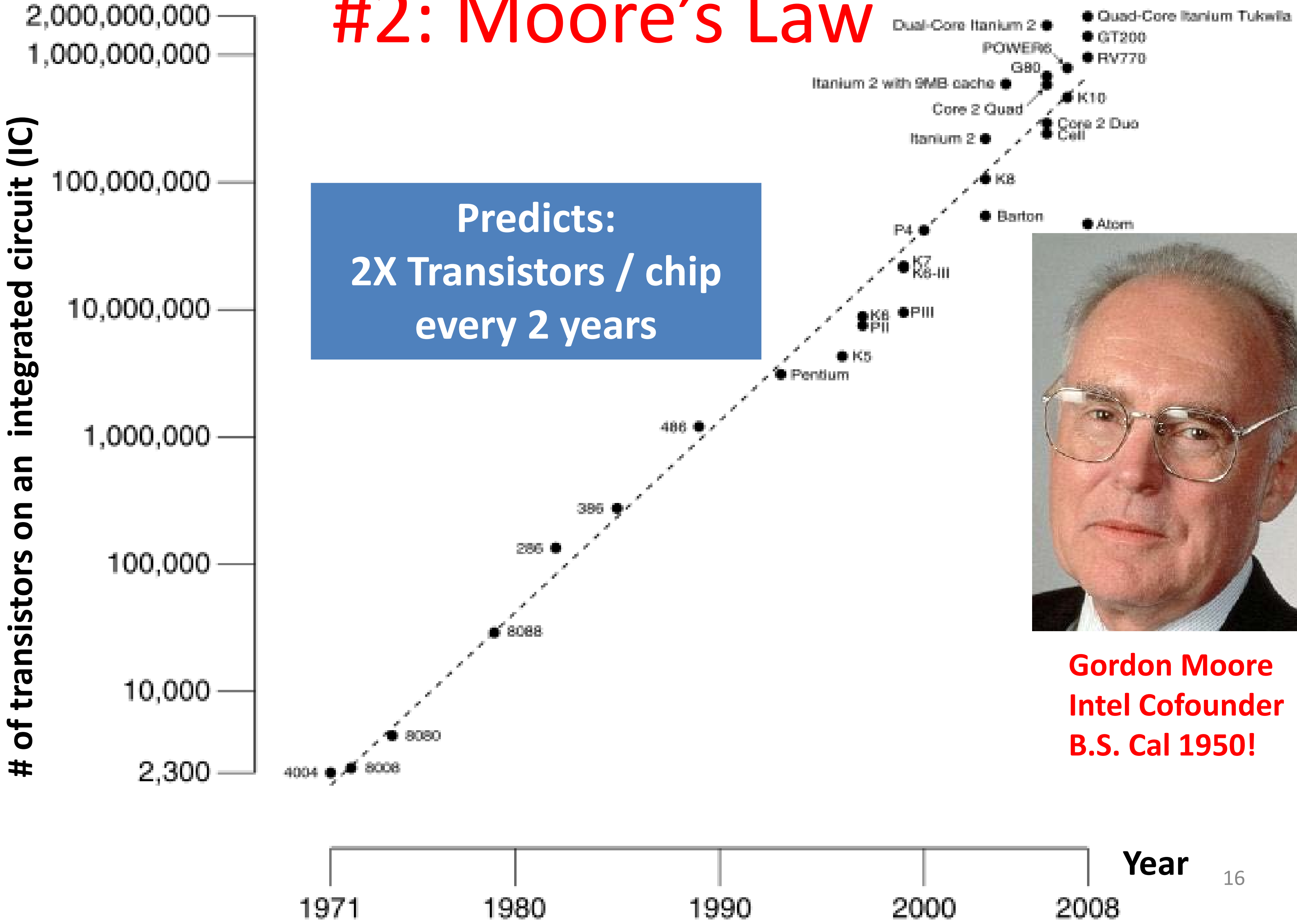
```
lw   $t0, 0($2)
lw   $t1, 4($2)
sw   $t1, 0($2)
sw   $t0, 4($2)
```

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

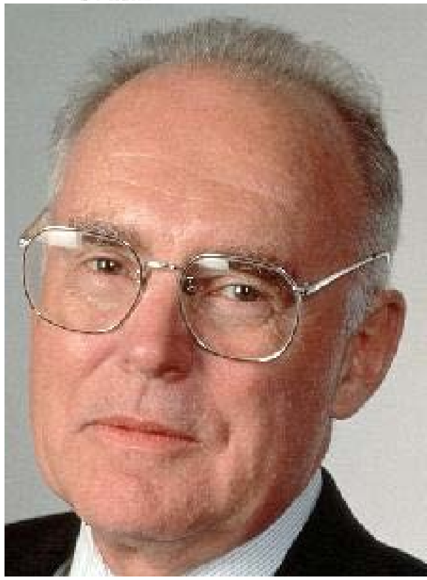
```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1                 1
0101 1000 0000 1001 1100 0                 1
```



#2: Moore's Law



Predicts:
2X Transistors / chip every 2 years



Gordon Moore
Intel Cofounder
B.S. Cal 1950!

Interesting Times

Moore's Law relied on the cost of transistors scaling down as technology scaled to smaller and smaller feature sizes.

BUT 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?

10^9 Tape /Optical Robot

10^6 Disk

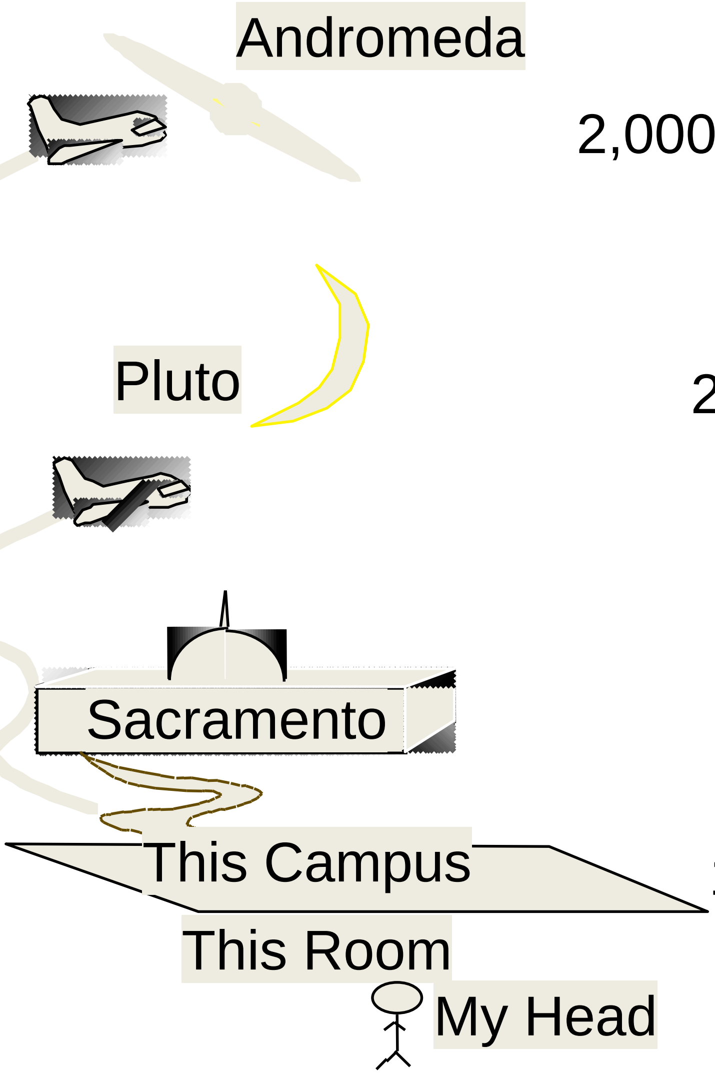
100 Main Memory

10 On Board Cache

2 On Chip Cache

1 Registers

(ns)



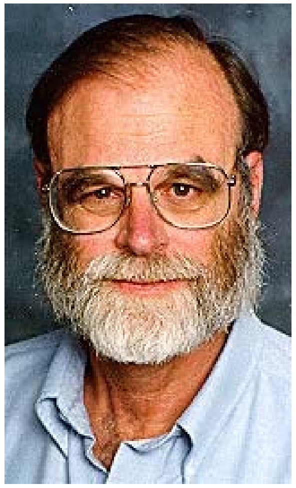
2,000 Years

2 Years

1.5 hr

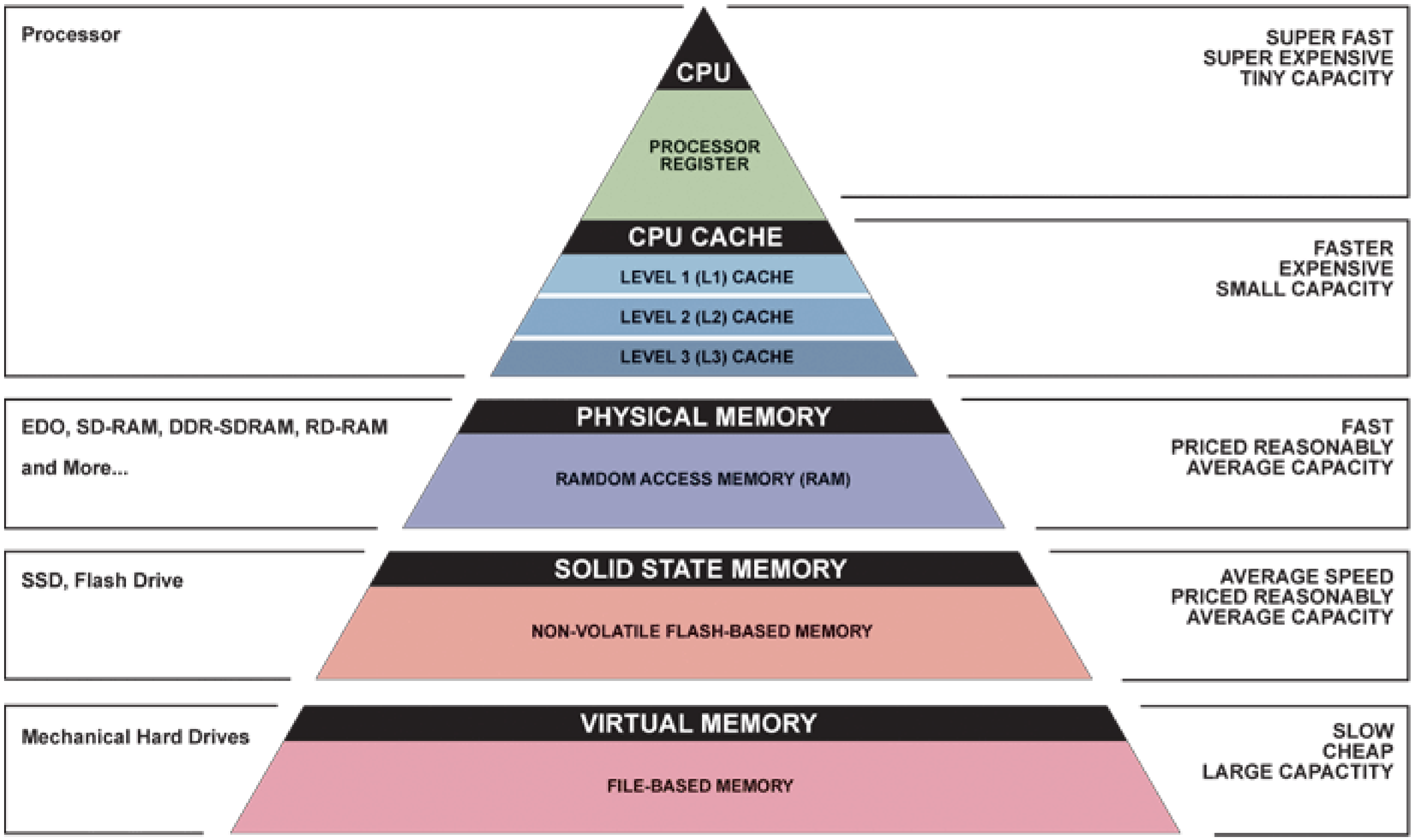
10 min

1 min

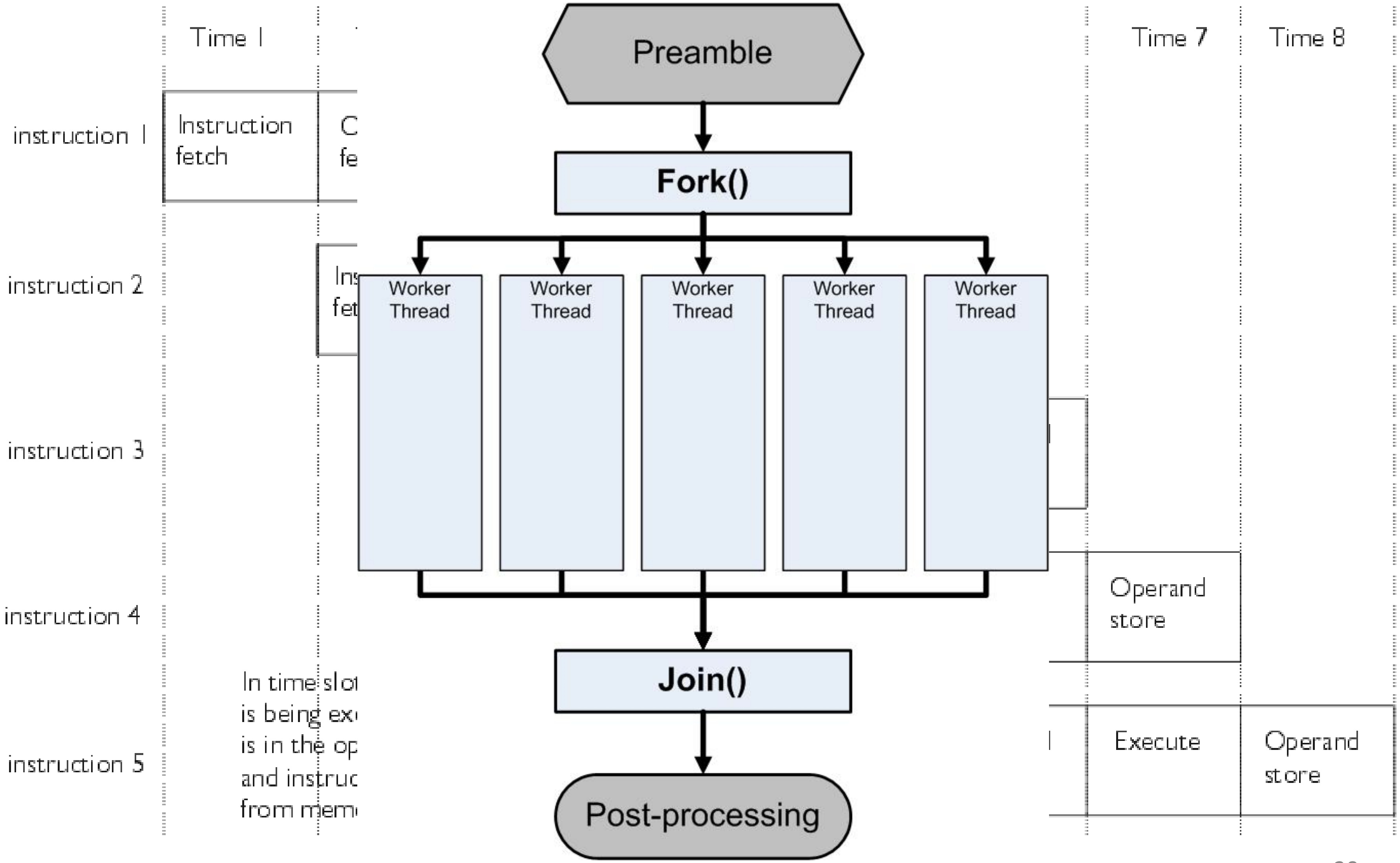


Jim Gray
Turing Award
B.S. Cal 1966
Ph.D. Cal 1969!

Great Idea #3: Principle of Locality/ Memory Hierarchy



Great Idea #4: Parallelism



Caveat: Amdahl's Law

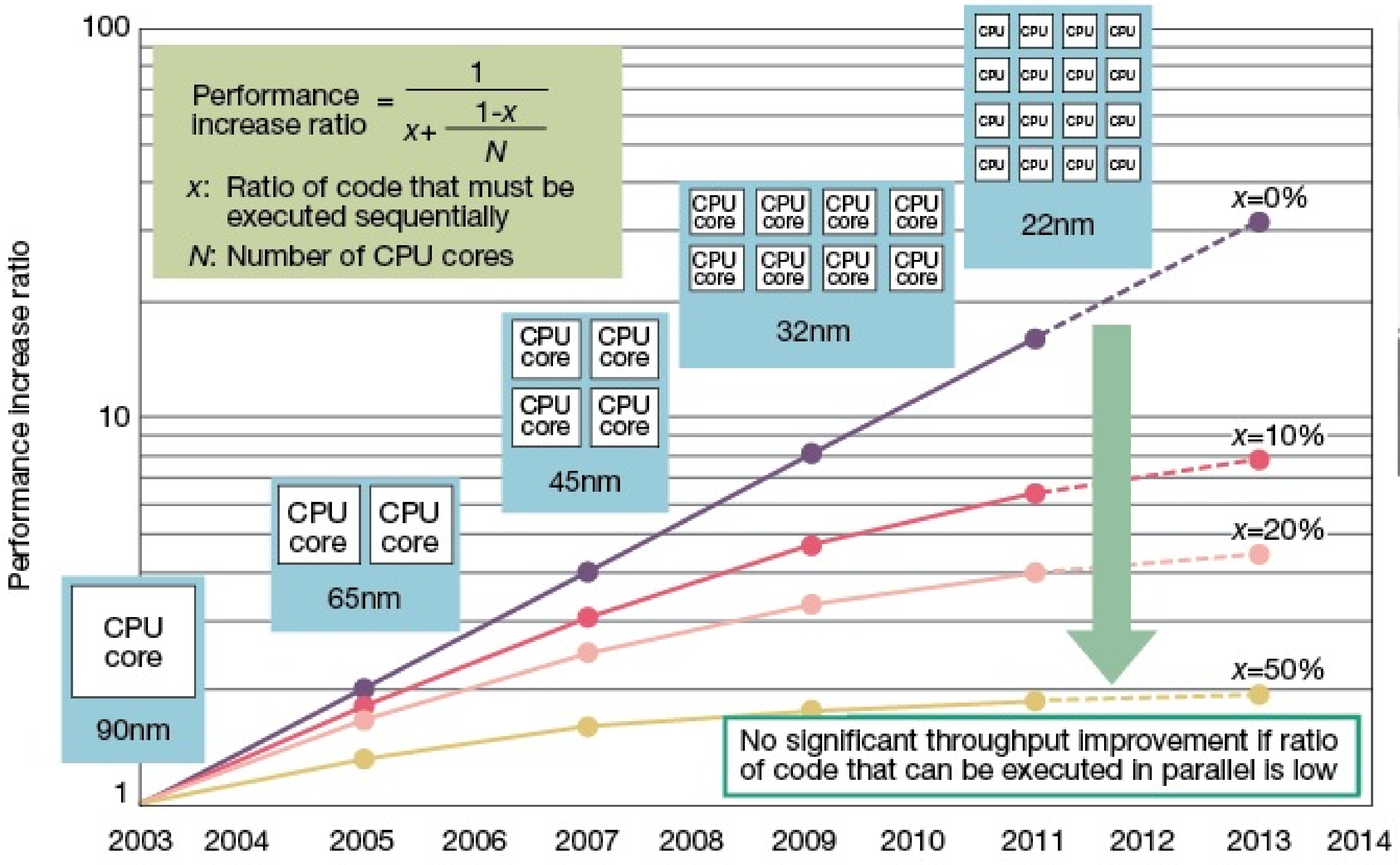


Fig 3 Amdahl's Law an Obstacle to Improved Performance Performance will not rise in the same proportion as the increase in CPU cores. Performance gains are limited by the ratio of software processing that must be executed sequentially. Amdahl's Law is a major obstacle in boosting multicore microprocessor 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

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

Coping with Failures

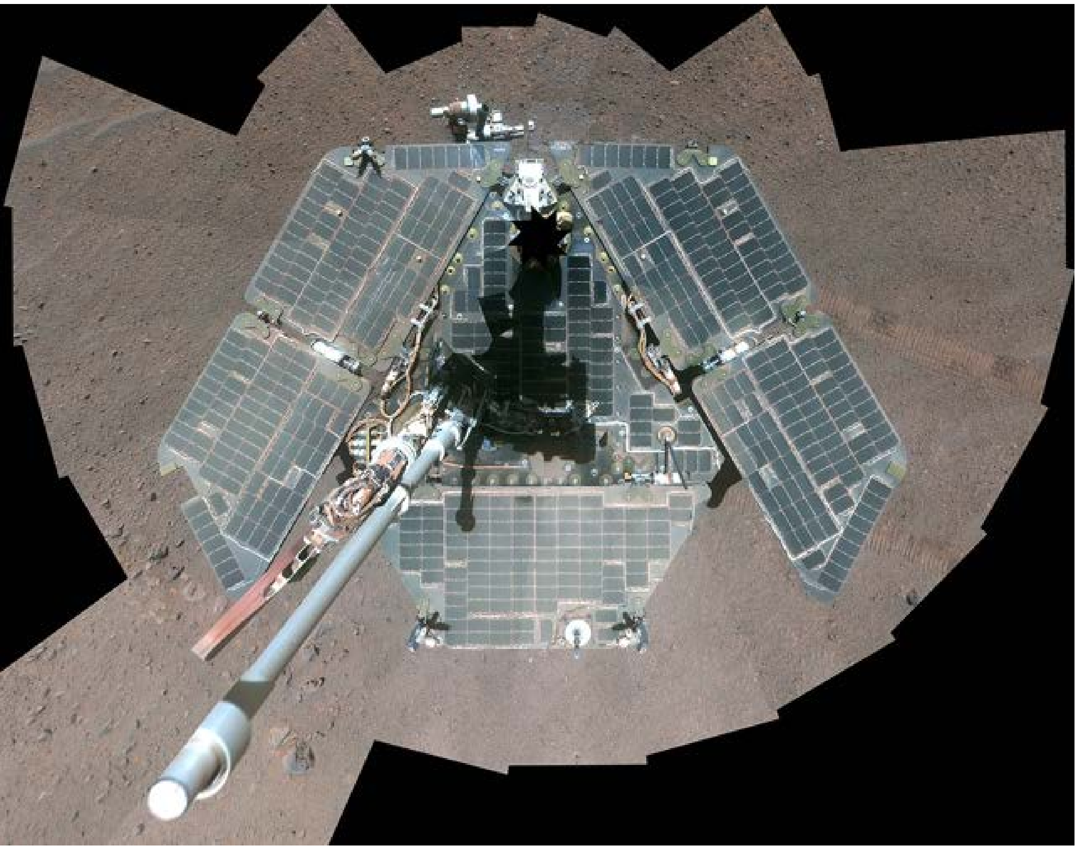
- 4 disks/server, 50,000 servers
- Failure rate of disks: 2% to 10% / year
  - Assume 4% annual failure rate
- On average, how often does a disk fail?
  a) 1 / month
  b) 1 / week
  c) 1 / day
  [✓ d) 1 / hour]

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 still active on Mars after >10 years

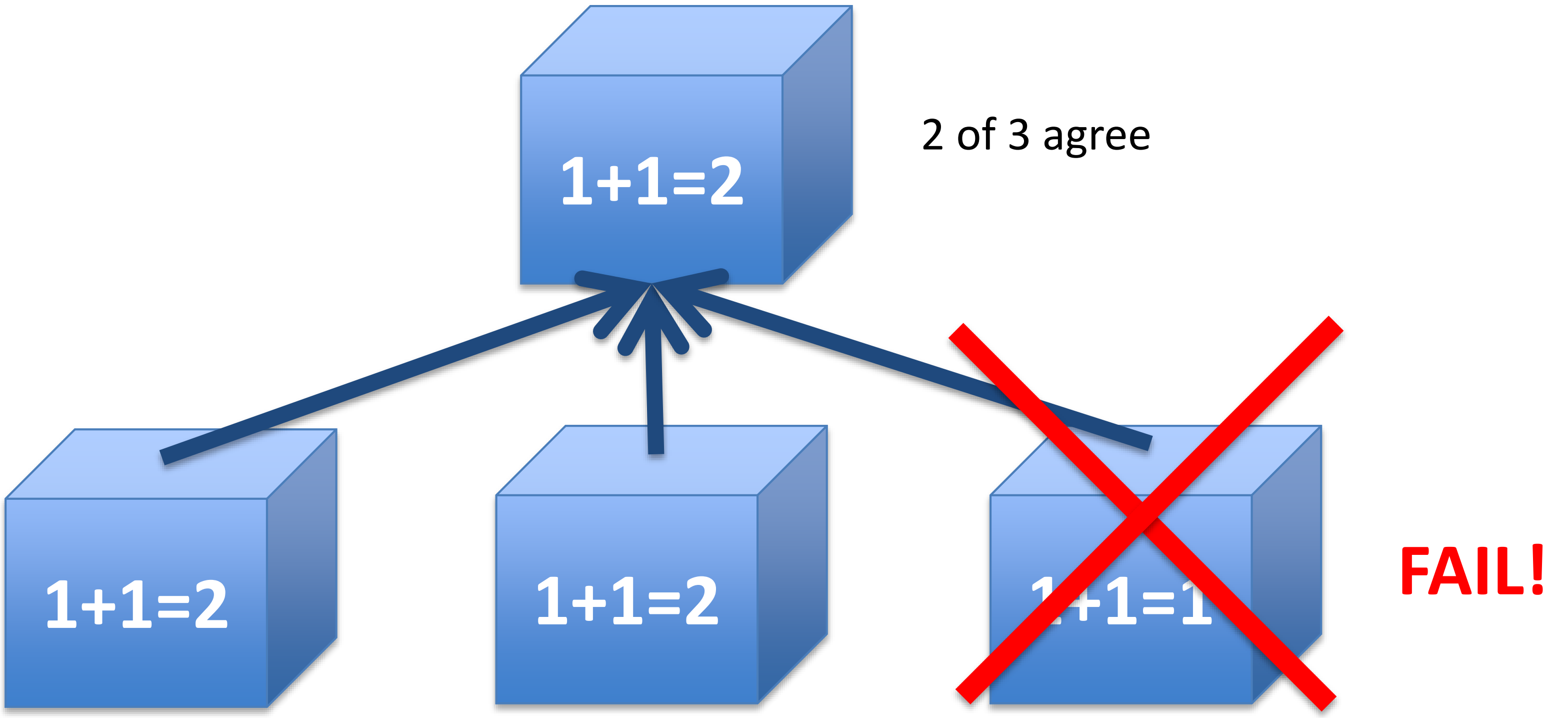
But flash memory worn out

New software update to avoid using worn out memory banks

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

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 <u>datacenters</u> so that can lose 1 datacenter but Internet service stays online
  - Redundant <u>disks</u> so that can lose 1 disk but not lose data (Redundant Arrays of Independent Disks/RAID)
  - Redundant <u>memory bits</u> of so that can lose 1 bit but no data (Error Correcting Code/ECC Memory)

Agenda

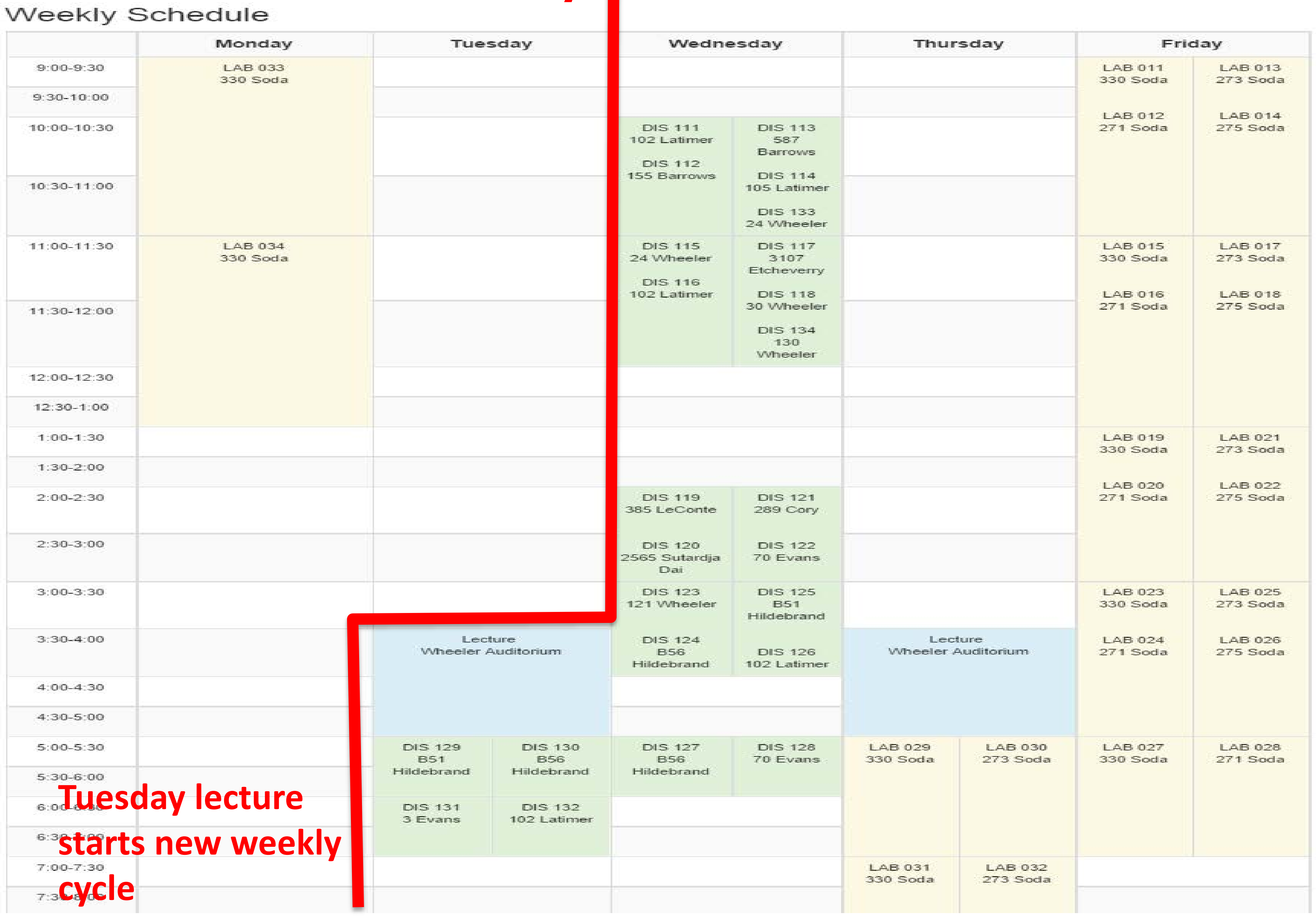
- Thinking about Machine Structures
- Great Ideas in Computer Architecture
- What you need to know about this class
- Everything is a Number

Yoda says…

"Always in motion, the future is…"

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

Weekly Schedule



Weekly Schedule

	Monday	Tuesday	Wednesday	Thursday	Friday
9:00-9:30	LAB 033 330 Soda				LAB 011 330 Soda LAB 013 273 Soda
9:30-10:00					
10:00-10:30			DIS 111 102 Latimer DIS 112 155 Barrows DIS 113 587 Barrows DIS 114 105 Latimer DIS 133 24 Wheeler		LAB 012 271 Soda LAB 014 275 Soda
10:30-11:00					
11:00-11:30	LAB 034 330 Soda		DIS 115 24 Wheeler DIS 116 102 Latimer DIS 117 3107 Etcheverry DIS 118 30 Wheeler DIS 134 130 Wheeler		LAB 015 330 Soda LAB 017 273 Soda
11:30-12:00					LAB 016 271 Soda LAB 018 275 Soda
12:00-12:30					
12:30-1:00					
1:00-1:30					LAB 019 330 Soda LAB 021 273 Soda
1:30-2:00					
2:00-2:30			DIS 119 385 LeConte DIS 121 289 Cory		LAB 020 271 Soda LAB 022 275 Soda
2:30-3:00			DIS 120 2565 Sutardja Dai DIS 122 70 Evans		
3:00-3:30			DIS 123 121 Wheeler DIS 125 B51 Hildebrand		LAB 023 330 Soda LAB 025 273 Soda
3:30-4:00		Lecture Wheeler Auditorium	DIS 124 B56 Hildebrand DIS 126 102 Latimer	Lecture Wheeler Auditorium	LAB 024 271 Soda LAB 026 275 Soda
4:00-4:30					
4:30-5:00					
5:00-5:30		DIS 129 B51 Hildebrand DIS 130 B56 Hildebrand	DIS 127 B56 Hildebrand DIS 128 70 Evans	LAB 029 330 Soda LAB 030 273 Soda	LAB 027 330 Soda LAB 028 271 Soda
5:30-6:00					
6:00-6:30		DIS 131 3 Evans DIS 132 102 Latimer			
6:30-7:00					
7:00-7:30				LAB 031 330 Soda LAB 032 273 Soda	
7:30-8:00					

Tuesday lecture starts new weekly cycle

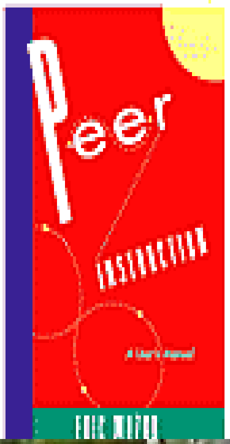
Course Information

- Course Web: http://inst.eecs.Berkeley.edu/~cs61c/
- Instructors:
  - John Wawrzynek & Vladimir Stojanovic
- Teaching Assistants: (see webpage)
- Textbooks: Average 15 pages of reading/week (can rent!)
  - Patterson & Hennessey, *Computer Organization and Design*, 5/e (we'll try to provide 4th Ed pages, not Asian version 4th edition)
  - Kernighan & Ritchie, *The C Programming Language*, 2nd Edition
  - Barroso & Holzle, *The Datacenter as a Computer, 2nd Edition*
- Piazza:
  - Every announcement, discussion, clarification happens there

Course Grading

- EPA: Effort, Participation and Altruism (5%)
- Homework (10%)
- Labs (5%)
- Projects (20%)
  1. Build your own Git repo (C)
  2. Non-Parallel Application (MIPS & C)
  3. Computer Processor Design (Logisim)
  4. Parallelize for Performance, SIMD, MIMD
  5. Massive Data Parallelism (Spark on Amazon EC2)
- Two midterms (15% each): 6th & 12th week in class, can be clobbered!
- Final (30%): 2015/5/15 @ 7-10pm
- Performance Competition for honor (and EPA)

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
  - We'll start this next week
  - No web-based clickers, sorry!

EECS Grading Policy

- http://www.eecs.berkeley.edu/Policies/ugrad.grading.shtml

  "A typical GPA for courses in the lower division is 2.7. This GPA would result, for example, from 17% A's, 50% B's, 20% C's, 10% D's, and 3% F's. A class whose GPA falls outside the range 2.5 - 2.9 should be considered atypical."
- Fall 2010: GPA 2.81
  26% A's, 47% B's, 17% C's,
  3% D's, 6% F's
- Job/Intern Interviews: They grill you with technical questions, so it's what you say, not your GPA

  (New 61C gives good stuff to say)

	Fall	Spring
2010	2.81	2.81
2009	2.71	2.81
2008	2.95	2.74
2007	2.67	2.76

Our goal as instructors

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

EPA!

- Effort
  - Attending prof and TA office hours, completing all assignments, turning in HW, doing reading quizzes
- Participation
  - Attending lecture and voting using the clickers
  - Asking great questions in discussion and lecture and making it more interactive
- Altruism
  - Helping others in lab or on Piazza
- EPA! points have the potential to bump students up to the next grade level! (but actual EPA! scores are internal)

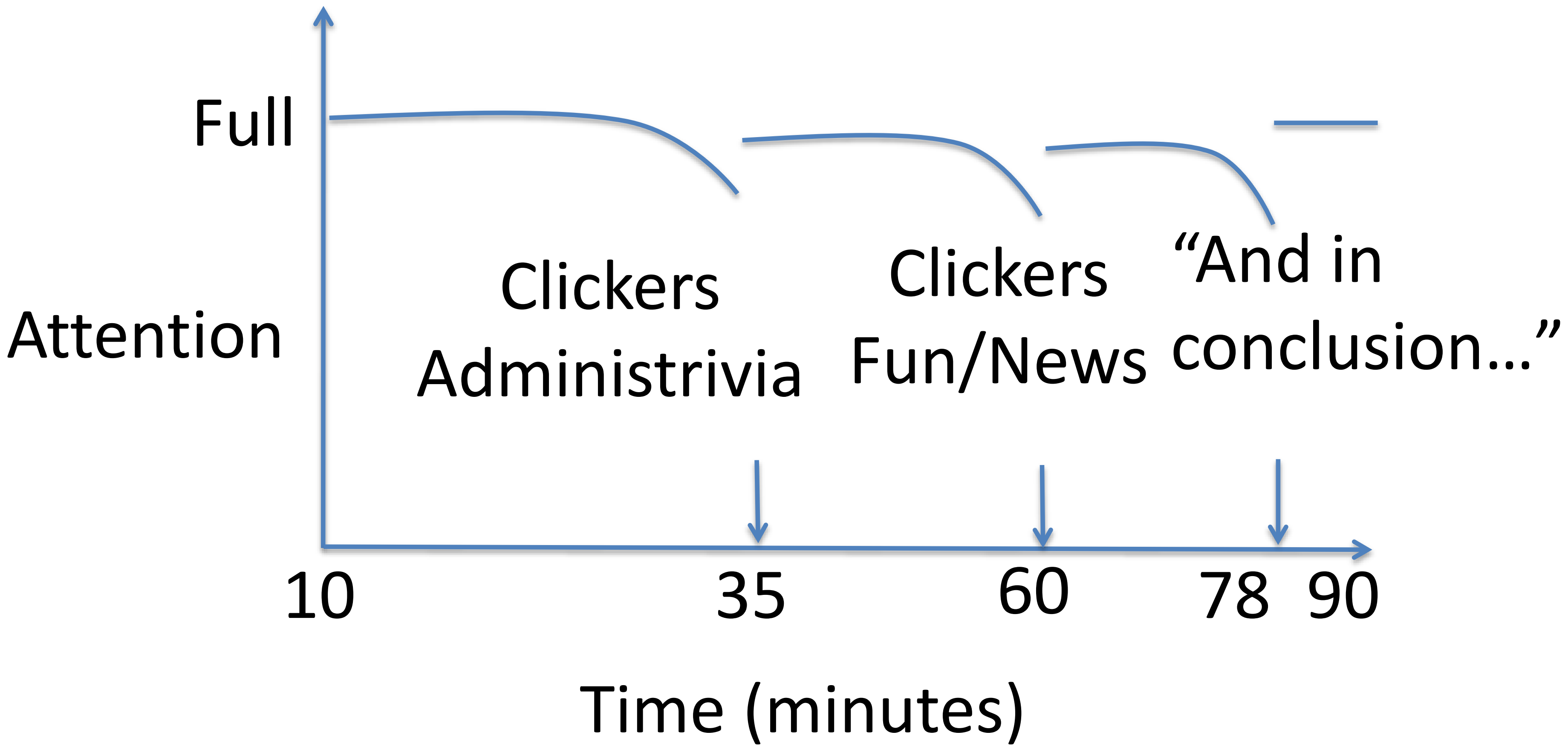
Late Policy … Slip Days!

- Assignments due at 11:59:59 PM
- You have <u>3</u> slip day tokens (NOT hour or min)
- Every day your project or homework is late (even by a minute) we deduct a token
- After you’ve used up all tokens, it’s 33% deducted per day.
  - No credit if more than 3 days late
  - Save your tokens for projects, worth more!!
- No need for sob stories, just use a slip day!

Policy on Assignments and Independent Work

- **ALL PROJECTS WILL BE DONE WITH A PARTNER**
- 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.
- PARTNER TEAMS MAY NOT WORK WITH OTHER PARTNER TEAMS
- 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 solutions from other students.
- It is NOT acceptable to copy (or start your) solutions from 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.
- **At the minimum F in the course**, and a letter to your university record documenting the incidence of cheating.
- (We've caught people in recent semesters!)
- **Both Giver and Receiver are equally culpable and suffer equal penalties**

Architecture of a typical Lecture



Agenda

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

Key Concepts

- Inside computers, 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, …
- 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:
- $123_{10} = 1_{10} \times 10_{10}{}^2 + 2_{10} \times 10_{10}{}^1 + 3_{10} \times 10_{10}{}^0$

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

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

  $= 123_{10}$
- Binary (Base 2), Hexadecimal (Base 16), Decimal (Base 10) different ways to represent an integer
  - We 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
- $\text{FFF}_{\text{hex}} = 15_{\text{ten}} \times 16_{\text{ten}}{}^{2} + 15_{\text{ten}} \times 16_{\text{ten}}{}^{1} + 15_{\text{ten}} \times 16_{\text{ten}}{}^{0}$
  $= 3840_{\text{ten}} + 240_{\text{ten}} + 15_{\text{ten}}$
  $= 4095_{\text{ten}}$
- $1111\ 1111\ 1111_{\text{two}} = \text{FFF}_{\text{hex}} = 4095_{\text{ten}}$
- May put blanks every group of binary, octal, 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

- For N-bit word, complement to 2_{ten}^{N}
  - For 4 bit number 3_{ten}=0011_{two}, two's complement (i.e. -3_{ten}) would be 16_{ten}-3_{ten}=13_{ten} or 10000_{two} – 0011_{two} = 1101_{two}
- Here is an easier way:
  - Invert all bits and add 1
  - Computers actually do it like this, too

$$\begin{array}{lr}
3_{ten} & 0011_{two} \\
\text{Bitwise complement} & 1100_{two} \\
+ & 1_{two} \\
\hline
-3_{ten} & 1101_{two}
\end{array}$$

Two's-Complement Examples

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

$$\begin{array}{rr} 3 & 0011 \\ +2 & \underline{0010} \\ 5 & 0101 \end{array} \qquad \begin{array}{rr} 3 & 0011 \\ +\ (-2) & \underline{1110} \\ 1 & 1\ 0001 \end{array} \qquad \begin{array}{rr} -3 & 1101 \\ +\ (-2) & \underline{1110} \\ -5 & 1\ 1011 \end{array}$$

Carry into MSB = Carry Out MSB

Overflow when magnitude of result too big to fit into result representation

$$\begin{array}{rr} 7 & 0111 \\ +1 & \underline{0001} \\ -8 & 1000 \end{array} \qquad \begin{array}{rr} -8 & 1000 \\ +\ (-1) & \underline{1111} \\ +7 & 1\ 0111 \end{array}$$

Overflow! ***Overflow!***

Carry into MSB $\neq$ Carry Out MSB

Carry in = carry from less significant bits

Carry out = carry to more significant bits

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

- CS61C: Learn 6 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. Performance Measurement and Improvement
  6. Dependability via Redundancy
- Everything is a Number!