CS 110
Computer Architecture
(a.k.a. Machine Structures)
Lecture 1: *Course Introduction*

Instructor:

Sören Schwertfeger

http://shtech.org/courses/ca/

School of Information Science and Technology SIST

ShanghaiTech University

Slides based on UC Berkley's CS61C

Agenda

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

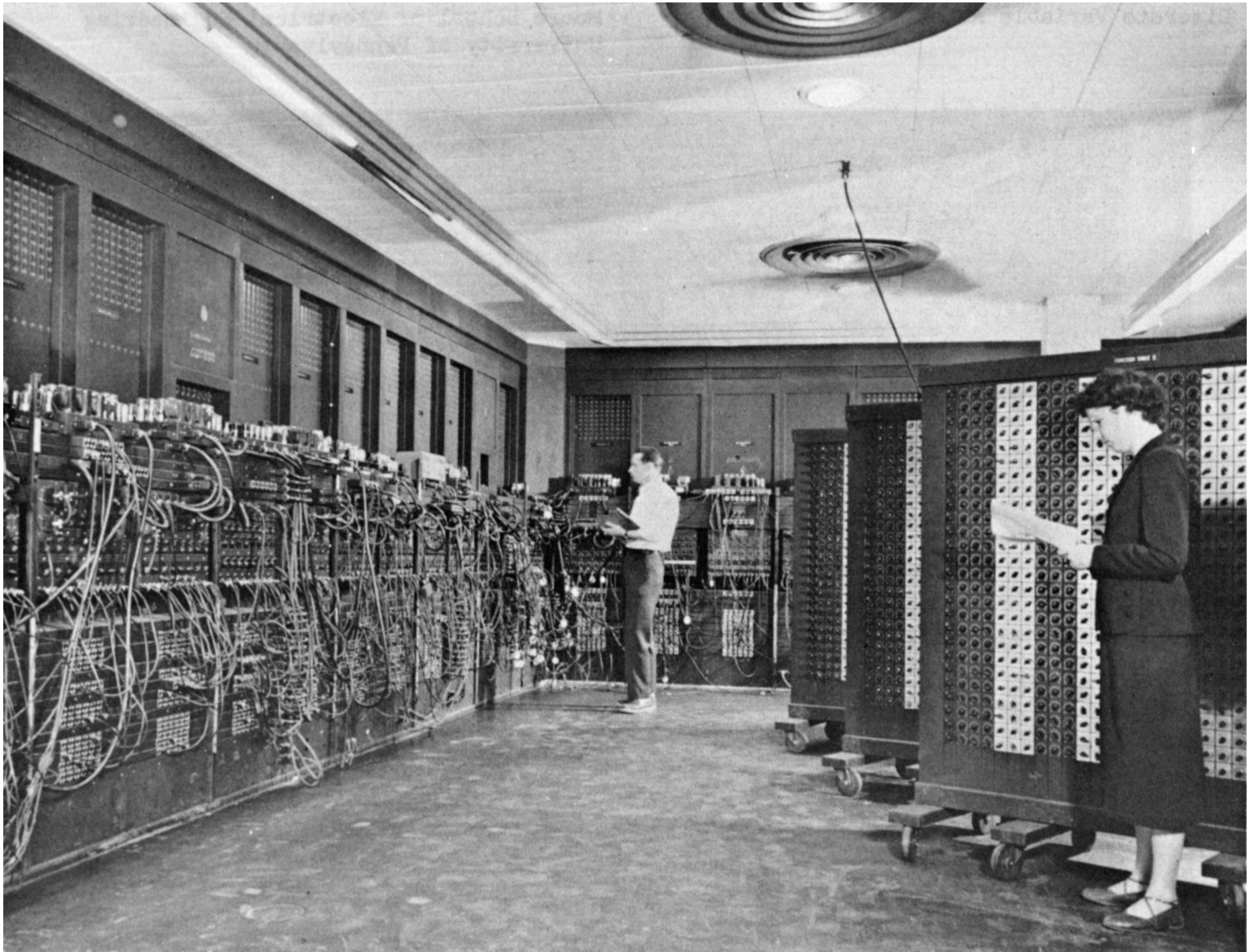
Agenda

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

CS 110 is NOT really about C Programming

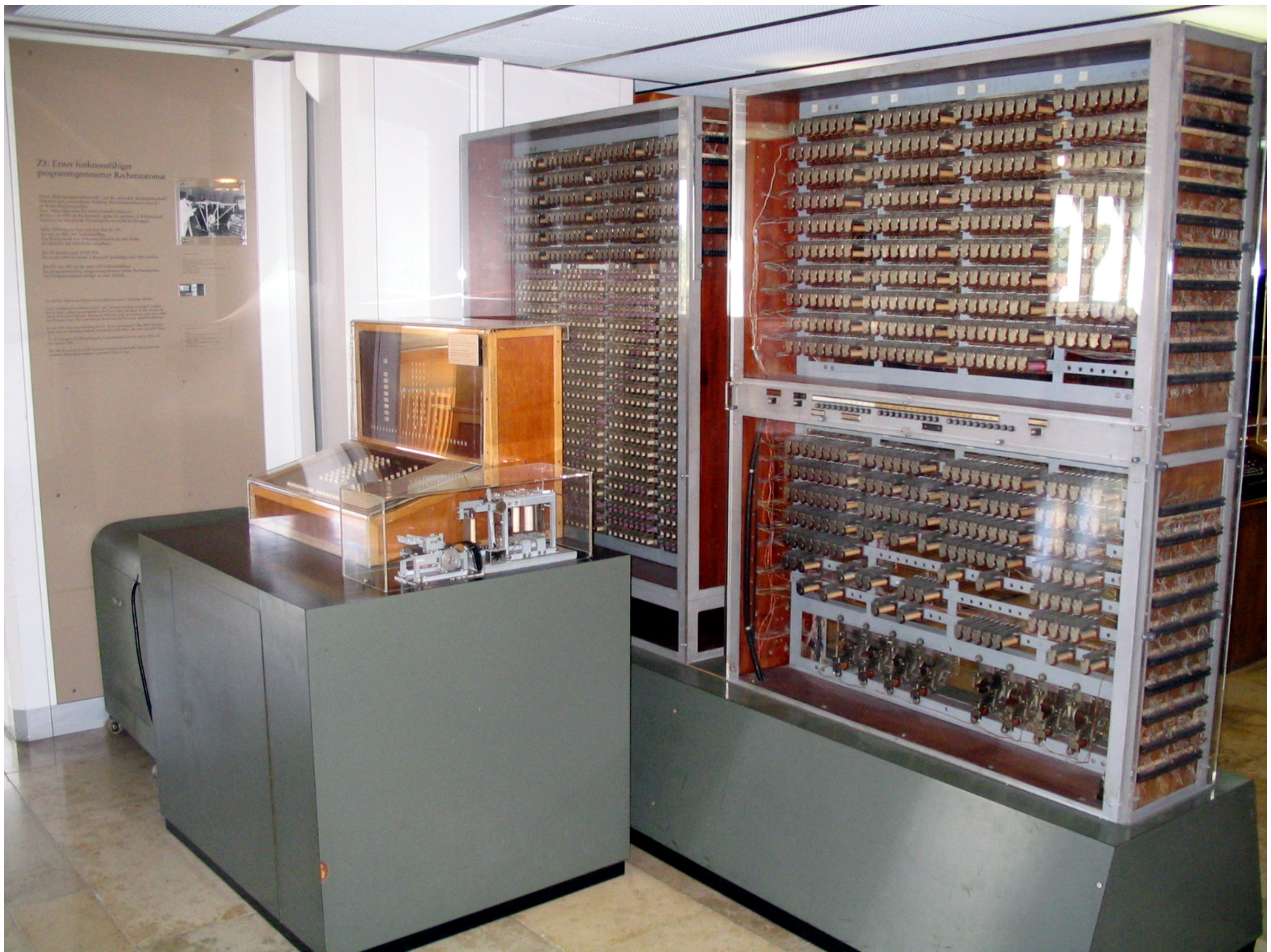
- It is about the *hardware-software interface*
  - What does the programmer need to know to achieve the highest possible performance
- C is close to the underlying hardware, unlike languages like Rust, Python, Java!
  - Allows us to talk about key hardware features in higher level terms
  - Allows programmer to explicitly harness underlying hardware parallelism for high performance

Old School Computer Architecture



Zuse Z3

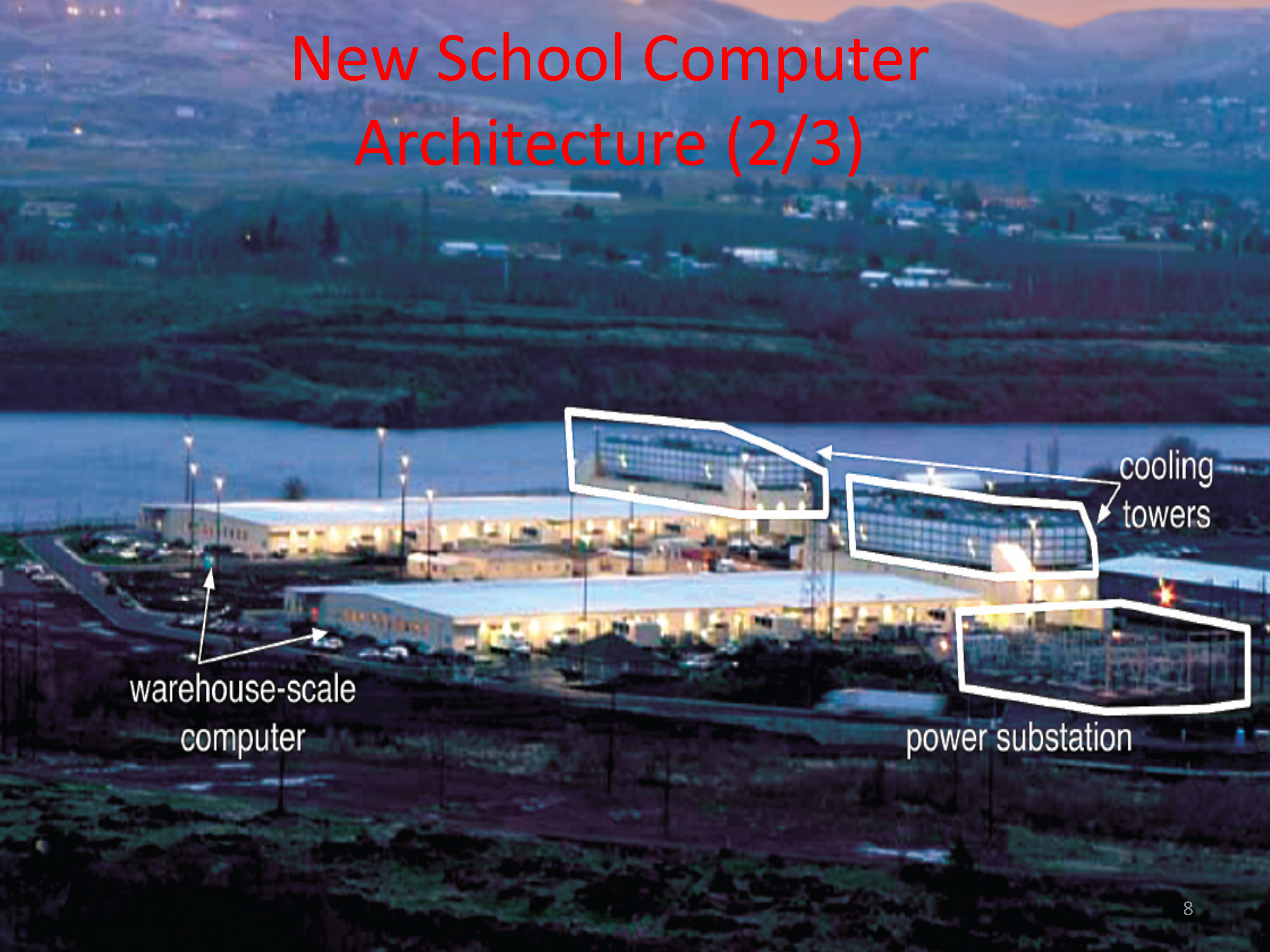
first working programmable, fully automatic digital computer by Konrad Zuse in Berlin, 1941 (Inventor of Computer)



New School Computer Architecture (1/3)

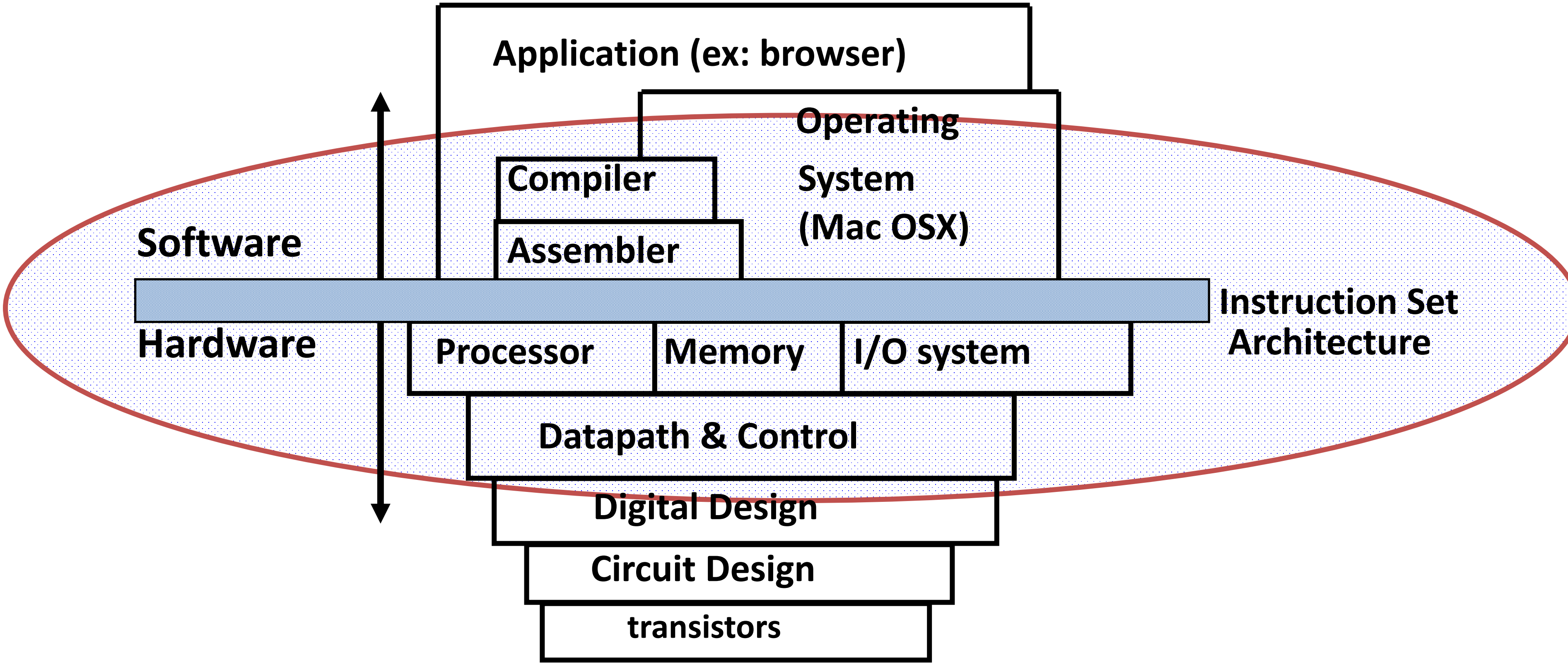
Personal Mobile Devices

New School Computer Architecture (2/3)



New School Computer Architecture (3/3)

Old School Machine Structures

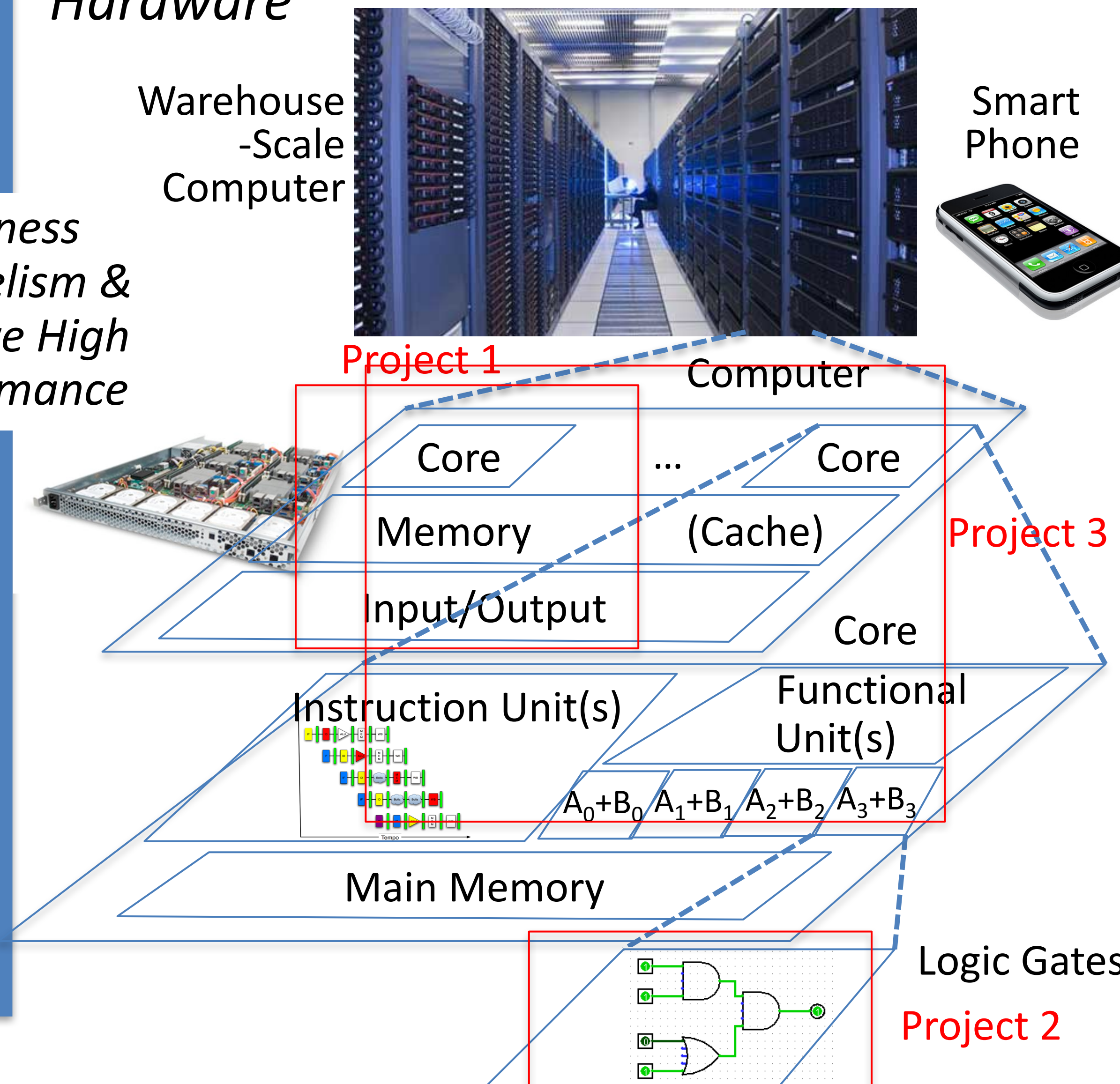


New-School Machine Structures (It’s a bit more complicated!)

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

6 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. Performance Measurement & Improvement
6. Dependability via Redundancy

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

Python / Application

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)

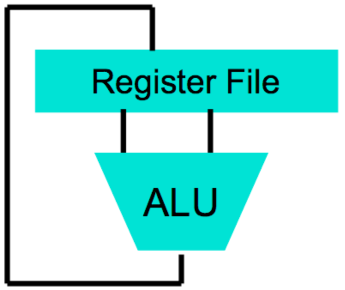
Physics

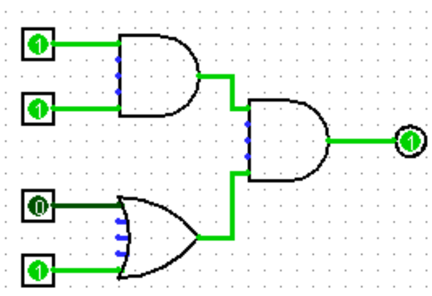
```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

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

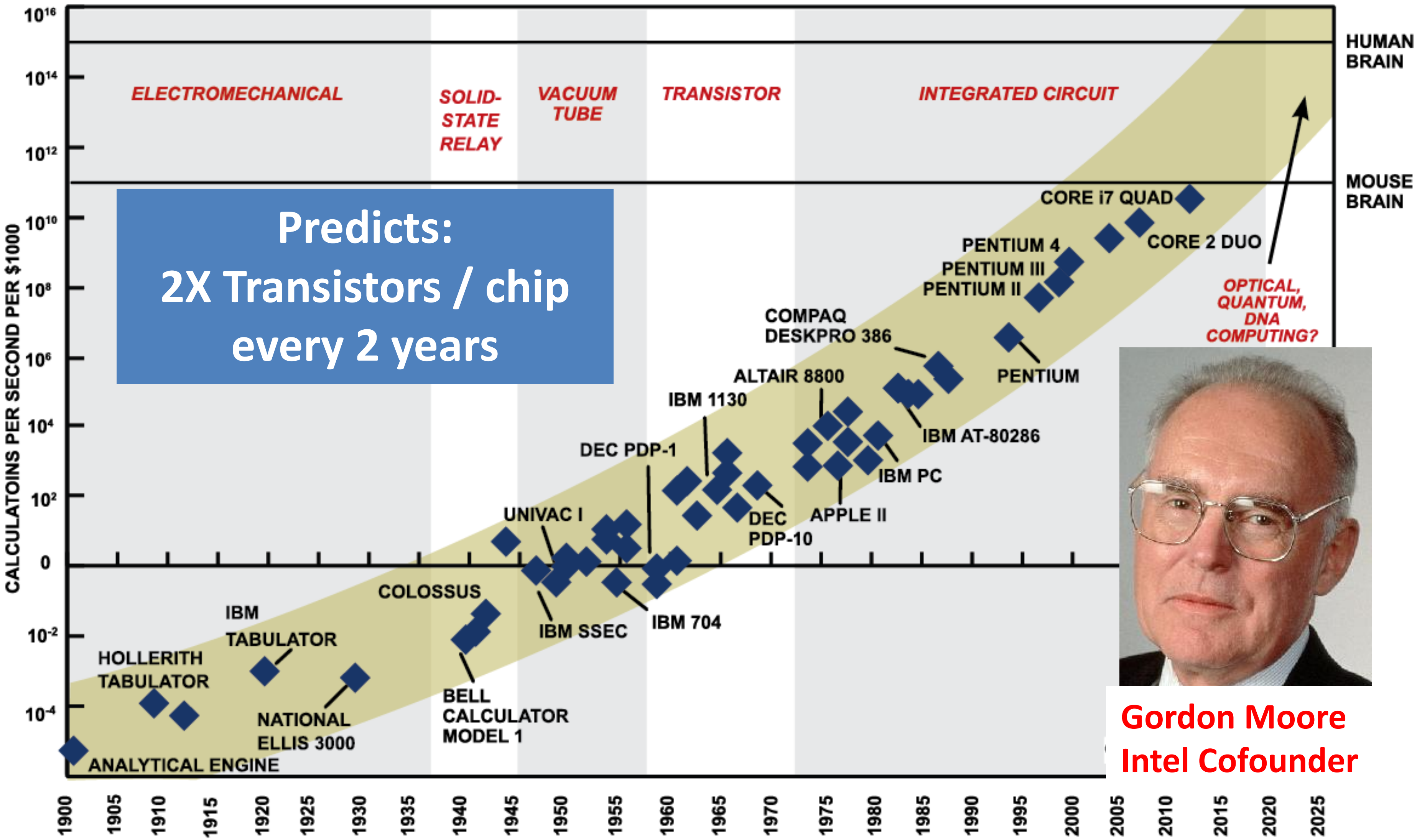
Anything can be represented as a *number*, i.e., data or instructions

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





#2: Moore’s Law



Predicts:
2X Transistors / chip every 2 years

Gordon Moore
Intel Cofounder

SOURCE: RAY KURZWEIL, "THE SINGULARITY IS NEAR: WHEN HUMANS TRANSCEND BIOLOGY", P.67, *THE VIKING PRESS*, 2006. DATAPOINTS BETWEEN 2000 AND 2012 REPRESENT BCA ESTIMATES.

Interesting Times

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

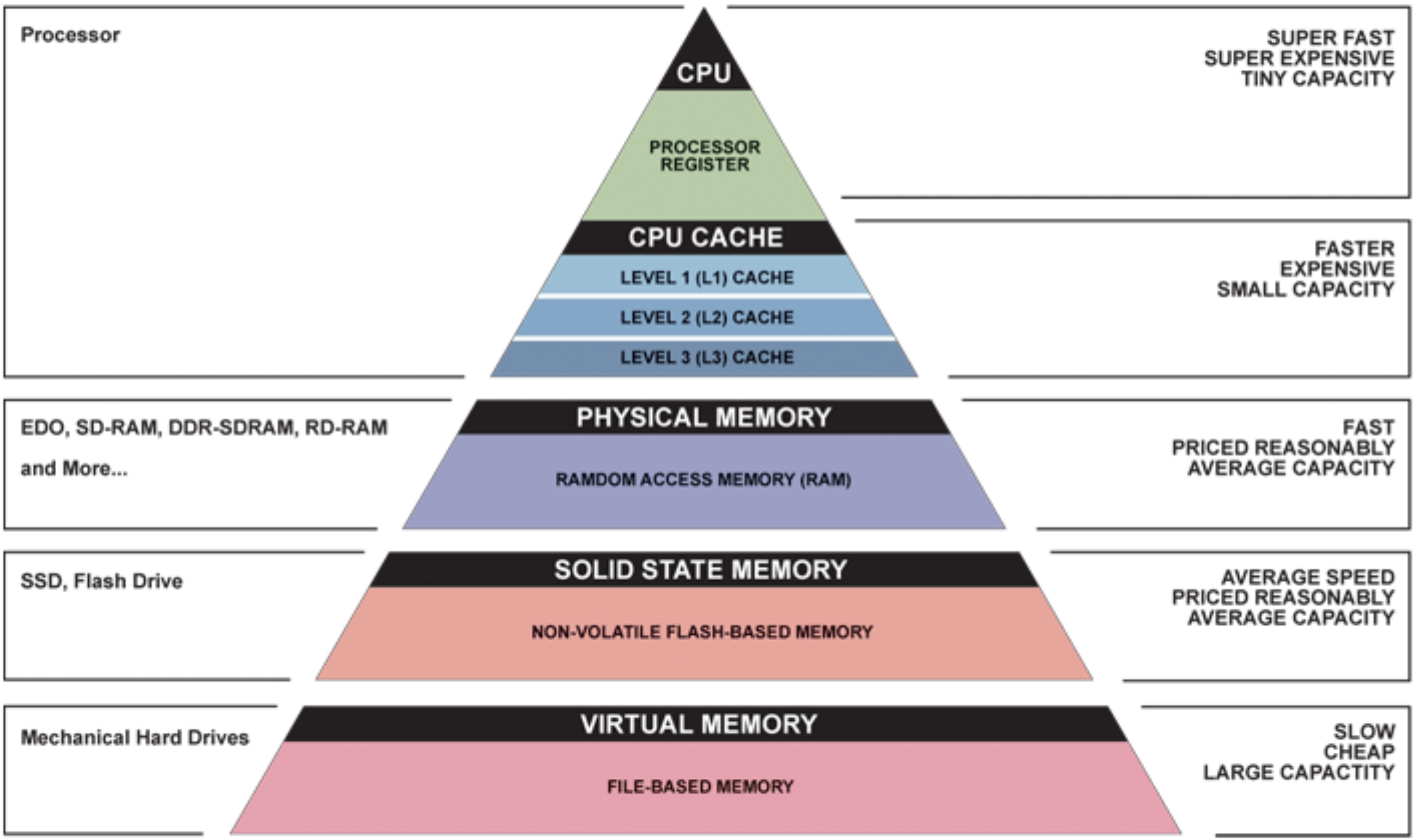
BUT newest, smallest fabrication processes <10nm, might have greater cost/transistor !!!!
So, why shrink????

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

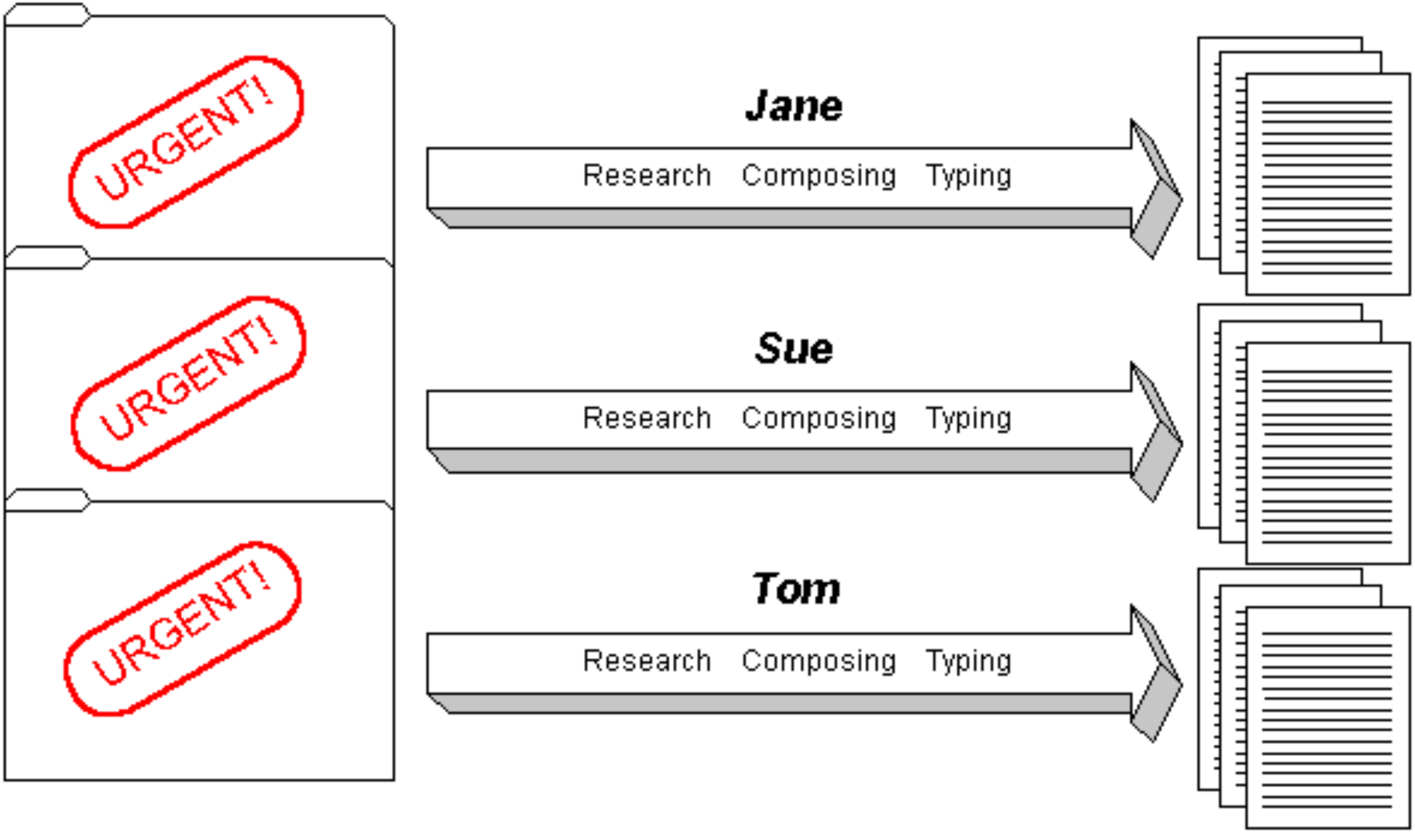
Jim Gray
Turing Award

(ns)			
10^9	Tape /Optical Robot	Andromeda	2,000 Years
10^6	HardDisk	Pluto	2 Years
100	Main Memory	Suzhou	1.5 hr
10	On Board Cache	This Campus	10 min
2	On Chip Cache	This Room	
1	Registers	My Head	1 min

Great Idea #3: Principle of Locality/ Memory Hierarchy



Great Idea #4: Parallelism



Caveat: Amdahl's Law

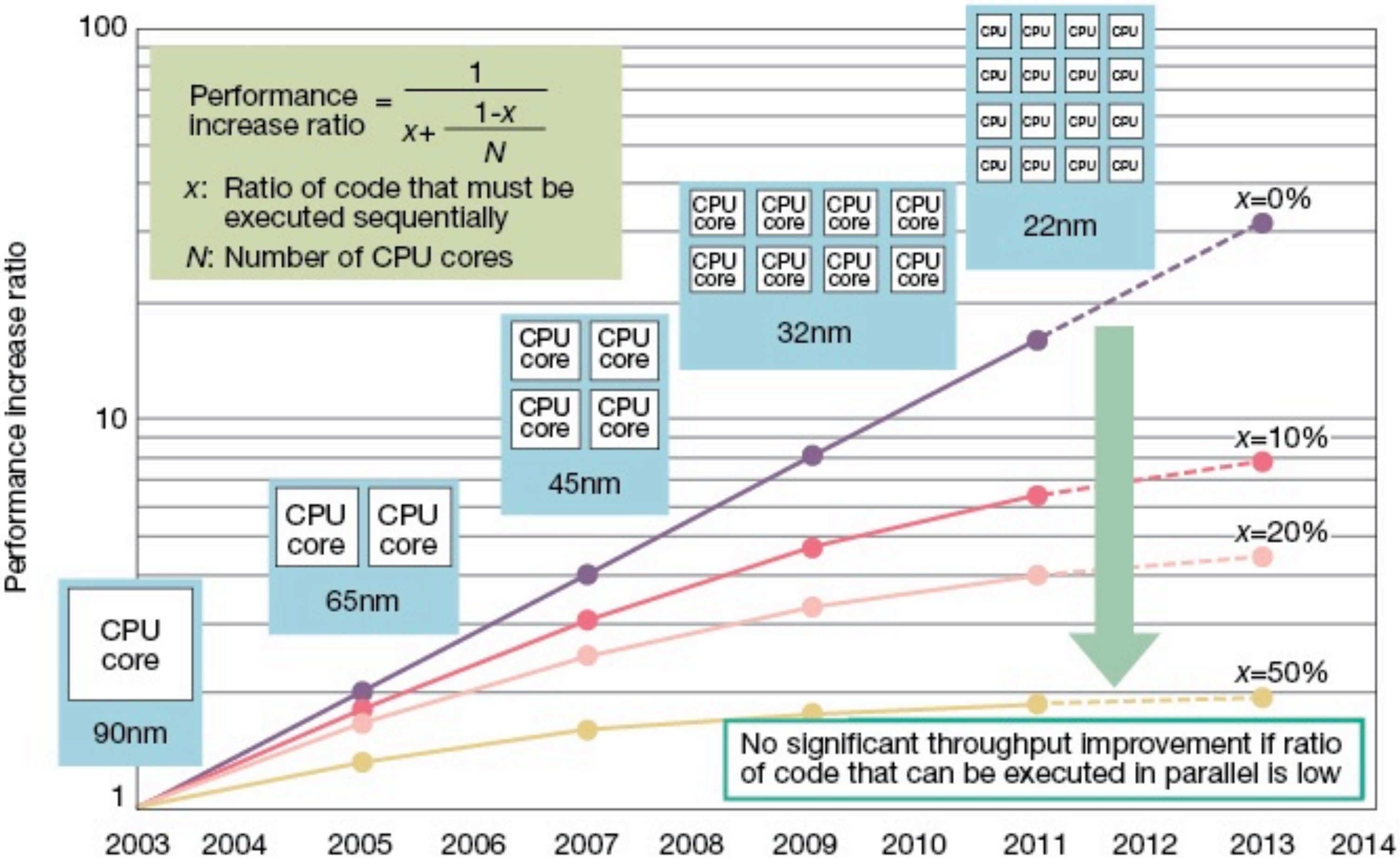


Fig 3 Amdahl's Law an Obstacle to Improved Performance Performance will not rise in the same proportion as the increase in CPU cores. Performance gains are limited by the ratio of software processing that must be executed sequentially. Amdahl's Law is a major obstacle in boosting multicore microprocessor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

Great Idea #5: Performance Measurement and Improvement

- Tuning application to underlying hardware to exploit:
  - Locality
  - Parallelism
  - Special hardware features, like specialized instructions (e.g., matrix manipulation)
- Latency
  - How long to set the problem up
  - How much faster does it execute once it gets going
  - It is all about *time to finish*

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

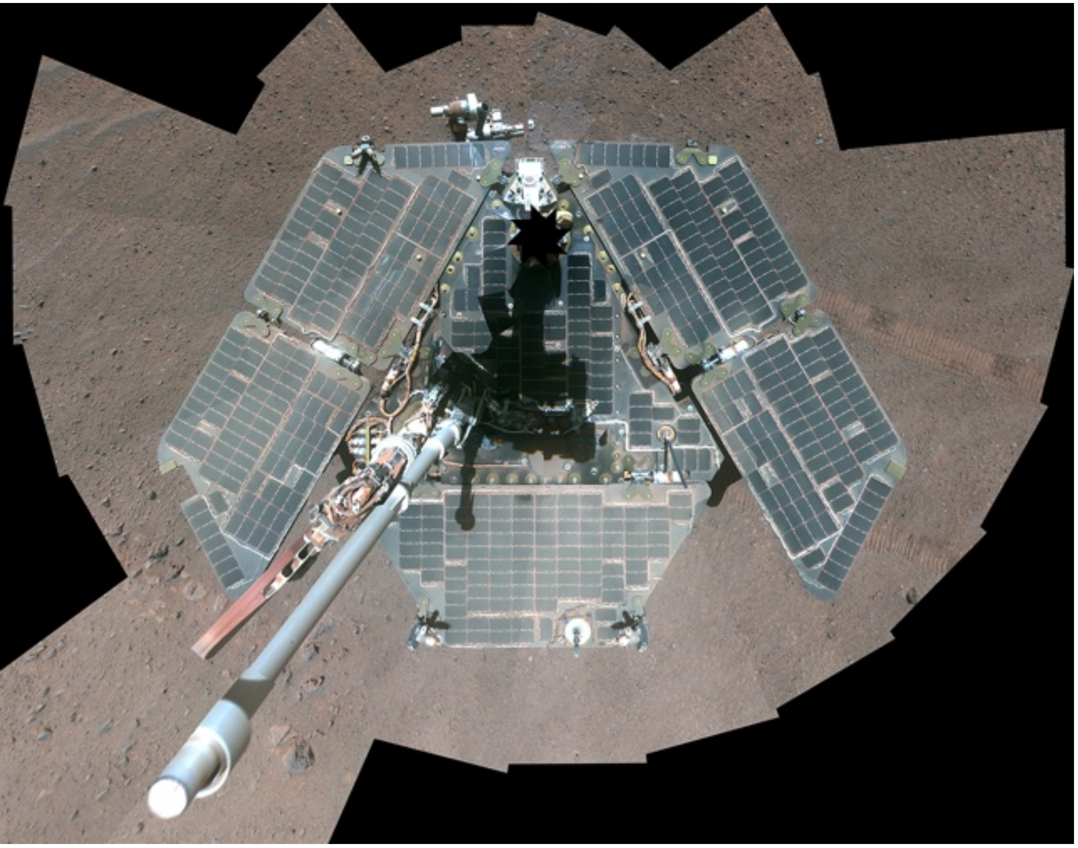
- 4 disks/server, 50,000 servers
- Failure rate of disks: 2% to 10% / year
  - Assume 4% annual failure rate
- On average, how often does a disk fail?
  a) 1 / month
  b) 1 / week
  c) 1 / day
  d) 1 / hour

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

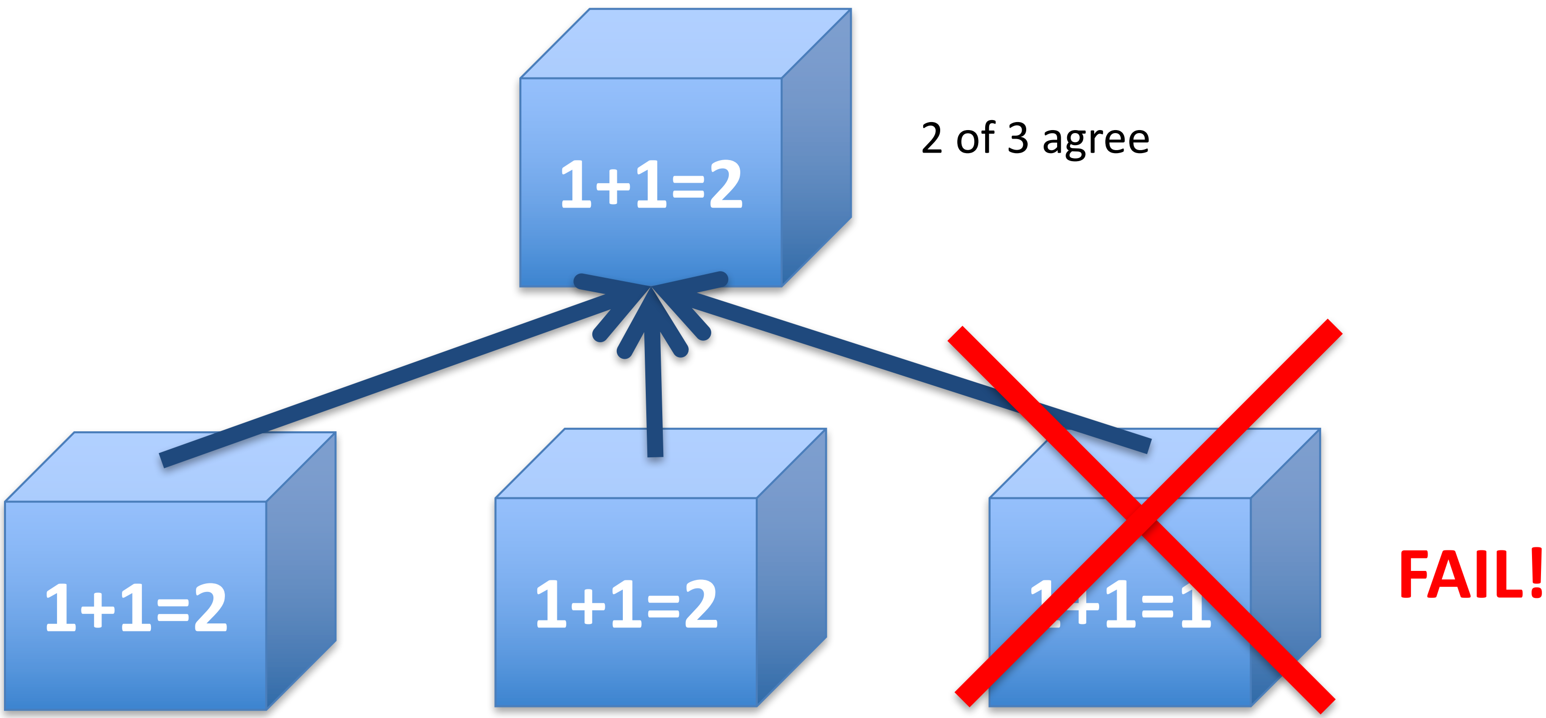


- Opportunity still active on Mars after >10 years
- But flash memory worn out
- Software update to avoid using worn out memory banks

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

Great Idea #6: 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 #6: 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 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)

Agenda

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

Weekly Schedule

Lecture	Tuesday, 10:15-11:55. Room: 教学中心 203
Lecture	Thursday, 10:15-11:55. Room: 教学中心 203
Discussions	Thursday 19:35. Room: 教学中心 203
Lab 1	Tuesday, 13:00-14:40. SIST 1D-106
Lab 2	Tuesday, 15:00-16:40. SIST 1D-106
Lab 3	Thursday, 13:00-14:40. SIST 1D-106
Lab 4	Thursday, 15:00-16:40. SIST 1D-106
Lab 5	Friday, 13:00-14:40. SIST 1D-106

Course Information

- Course Web: http://shtech.org/course/ca/
- Acknowledgement: Instructors of UC Berkeley's CS61C: http://www-inst.eecs.berkeley.edu/~cs61c/
- Instructor:
  - Sören Schwertfeger
- Teaching Assistants: (see webpage)
- Textbooks: Average 15 pages of reading/week
  - Patterson & Hennessey, *Computer Organization and Design*, 5th Edition (Chinese version is 4th edition – significant differences!)
  - 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

- Projects: 33%
- Homework: 17%
- Lab: 5%
- Exams: 40%
  - Midterm 1: 10%
  - Midterm 2: 10%
  - Final: 20%
- Participation: 5%
  - (in class, in piazza, non credit parts of HW/ projects, help other during labs)

Late Policy … Slip Days!

- Assignments due at 11:59:59 PM
- You have 3 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 25% 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!
- Gradebot is open till 3 days after due date!
- If you need more time (slip days plus deduction) send the TA and Prof. an email!

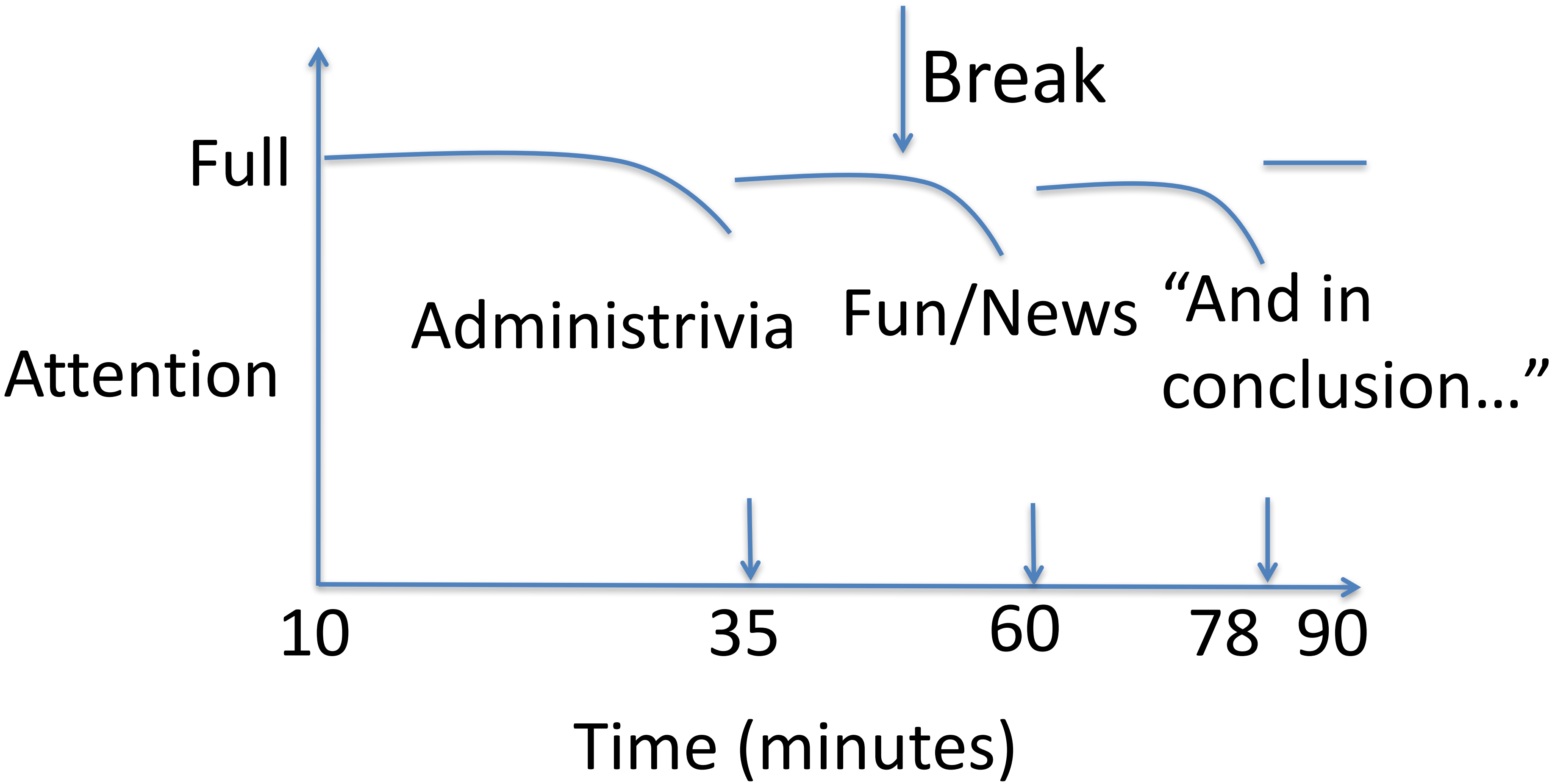
Policy on Assignments and Independent Work

- **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 can discuss your assignments with other students, and credit will be assigned to students who help others by answering questions on Piazza (participation), but we expect that what you hand in is yours.
- Level of detail allowed to discuss with other students: Concepts (Material taught in the class/ in the text book)! Pseudocode is NOT allowed!
- Use the Office Hours of the TA and the Prof. if you need help with your homework/ project!
- Rather submit an incomplete homework with maybe 0 points than risking an F!
- It is NOT acceptable to copy solutions from other students.
- You can never look at homework/ project code not by you/ your team!
- You cannot give your code to anybody else –> secure your computer when not around it
- 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.
- **Both Giver and Receiver are equally culpable and suffer equal penalties**

Discussion & Labs & HW1

- First discussion this week!
- Labs: Find one partner for your lab-work and the projects – from you lab class!
  - Send an email to Xu Qingwen (xuqw)
    - Subject [CA] !!!!!!!!!
  - Labs start next week
  - Your lab partner is also your project partner!
- HW1 will be posted this week.

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:
- $123_{10} = 1_{10} \text{ x } 10_{10}{}^2 + 2_{10} \text{ x } 10_{10}{}^1 + 3_{10} \text{ x } 10_{10}{}^0$

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

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

  $= 123_{10}$
- Binary (Base 2), Hexadecimal (Base 16), Decimal (Base 10) different ways to represent an integer
  - We 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
- $FFF_{hex} = 15_{ten} x\ 16_{ten}{}^{2} + 15_{ten} x\ 16_{ten}{}^{1} + 15_{ten} x\ 16_{ten}{}^{0}$
  $= 3840_{ten} + 240_{ten} + 15_{ten}$
  $= 4095_{ten}$
- $1111\ 1111\ 1111_{two} = FFF_{hex} = 4095_{ten}$
- 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*, e.g. 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) (4 Gig)

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

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

Two's-Complement Examples

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

```
     3    0011
    +2    0010
    ------------
     5    0101
```

```
     3      0011
+ (-2)      1110
    ------------
   1  1     0001
```

```
    -3      1101
+ (-2)      1110
    ------------
   -5  1    1011
```

Carry into MSB = Carry Out MSB

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

```
     7    0111
    +1    0001
    ------------
    -8    1000
```

Overflow!

```
    -8      1000
+ (-1)      1111
    ------------
   +7  1    0111
```

Overflow!

Carry into MSB ≠ 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

- Computer Architecture: 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!