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

Instructors:
Sören Schwertfeger & Chundong Wang

https://robotics.shanghaitech.edu.cn/courses/ca/20s/

School of Information Science and Technology SIST

ShanghaiTech University

Slides based on UC Berkley's CS61C

Agenda

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

Agenda

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

Computer Architecture

- CA is your most important course this semester!
  - **6 credit points**
    - 4 credit CA; 2 credit projects => projects in parallel to HW!
  - Your first CS only course
  - Spend a LOT of time on:
    - Textbook reading before class
    - HW and projects
    - Lab preparation
    - Learning for mid-terms and final
  - Understand how computers really work – complicated!
    - Too complicated? => Change major…

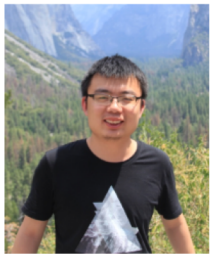
Weekly Schedule

Lecture	Tuesday, 10:15-11:55. 教学中心 (Teaching Center) 201
Lecture	Thursday, 10:15-11:55. 教学中心 (Teaching Center) 201
Lab 1	Monday, 15:55-17:35. SIST 1A-104; TA: Ze Song
Lab 2	Tuesday, 19:35-21:15. SIST 1B-108; TA: Zhongyi Cai
Lab 3	Tuesday, 19:35-21:15. SIST 1B-106; TA: Kaiyuan Xu
Lab 4	Tuesday, 15:55-17:35. SIST 1A-104; TA: Tianyuan Wu
Lab 5	Tuesday, 15:55-17:35. SIST 1A-106; TA: Peifan Li
Lab 6	Tuesday, 15:55-17:35. SIST 1A-108; TA: Mengying Wu
Lab 7	Tuesday, 19:35-21:15. SIST 1B-104; TA: Cheng Yu
Lab 8	Tuesday, 19:35-21:15. SIST 1B-106; TA: Peihao Wang
Lab 9	Tuesday, 19:35-21:15. SIST 1B-108; TA: Zhongyue Lin

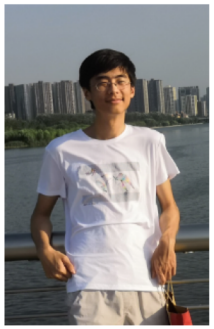
Sören Schwertfeger 师泽仁
<soerensch>

Chundong Wang
<cd_wang AT outlook.com>

Yanjie Song
<songyj>
Head TA

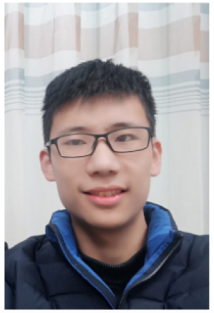


Anqi Pang
<pangaq>



Jinrui Wang
<wangjr>

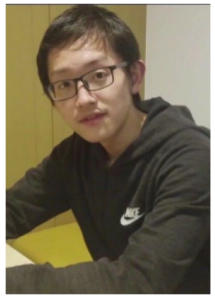
Peifan Li
<lipf>



Ze Song
<songze>

Cheng Yu
<yucheng>

Kaiyuan Xu
<xuky>



Peihao Wang
<wangph>

Zhongyi Cai
<caozhy>

Hang Su
<suhang>

Mengying Wu
<wumyl>

Tianyuan Wu
<wuty>

Zhongyue Lin
<linzhy>

Course Information

- Course Web: https://robotics.shanghaitech.edu.cn/courses/ca/20s/
- Acknowledgement: Instructors of UC Berkeley's CS61C: https://cs61c.org/

- Instructors:
  - Sören Schwertfeger & Chundong Wang
- Teaching Assistants: (see webpage)
- Textbooks: Average 15 pages of reading/week
  - Patterson & Hennessey, *Computer Organization and Design* RISC-V 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

- 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)

Lectures

- Each lecture split into 3-5 videos
- Required to watch videos latest on day of lecture!
- Pay attention!
- Videos available on course website and ShanghaiTech cloud.
- Download the videos, don’t watch online!
- Subtitles are provided. Recommended player: VLC
- Lecture slides available as pdf.
- Read recommended reading BEFORE lecture!

- During online teaching, piazza will have to be used extensively!
- Discuss & ask questions after each class
  - General topics can be discussed outside of class thread
- Participate in the “in-lecture” polls, tasks
- Ask general questions about hw/ projects
- Announcements will be posted on piazza only!
  - Check email & piazza at least once a day!
- Participation is recorded & goes into grade!

Labs

- Labs: Find one partner for your lab-work from you lab class!
  - Labs start next week
  - Projects are done in 2-person teams!
- Need Linux for the labs!
  - Recommendation: install natively (dual boot)!
  - Virtual Machine works fine
  - If in doubt: Ubuntu 18.04
  - Many labs/ homework/ project may work with Mac, but no guarantee, no support. Use Linux!
- Install the zoom app on your laptop – there is a Linux/ Ubuntu version for it!
  - Make an appointment with YOUR lab TA DURING the lab hours.
  - Make a 3 person zoom meeting: You, you lab partner, the TA
  - BOTH students alternate to share their screen to show the code
  - Both explain the TA what they learned, do the lab tasks.

Discussion

- Time (during online phase): During Thursday lecture slot! **10:15-11:55**
- How: Zoom meeting with your lab group! You lab's TA will give you the invitation in the QQ group.
- Content: The TA will give a short presentation. Then you can ask questions.
- Participation is **mandatory!**
  - Write email to head TA Yanjie Song <songyj> if not possible

Office Hours

- During online phase: 1 to 1 chats with the TA.
- First: think if piazza may be better way to answer your question
- If personal tutoring is needed: Ask in the qq group which TA is available.
  - Chat with TA or call.

- For organizational things write to head TA Yanjie Song <songyj> or Prof. Sören Schwertfeger <soerensch>

AUTOLAB

- CA will use Autolab for grading HW & projects
  - Setup maintained by TAs
    - => Please be patient and report any bugs/ problems in piazza or (if sensitive) via email.
  - Your logins will be created soon!
  - http://autolab.sist.shanghaitech.edu.cn/

- HW 1 is available on Autolab. Due March 6!

- Gradescope for text-based HW and exams.

- Gitlab for projects!
- Use for collaboration.
- Autolab will directly pull from your gitlab!
- http://autolab.sist.shanghaitech.edu.cn/gitlab
- Accounts will be created later

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!
- Autolab will take care of this!

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)
- It is NOT acceptable to give anyone other than your project partner access to your gitlab!
- 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**

Recommendation

- Go to autolab, make HW1

- Afterwards: Watch next video:

  Introduction To Computer Architecture

CS 110
Computer Architecture
(a.k.a. Machine Structures)
Lecture 1: *Course Introduction*
Part 2: CA Introduction

Instructors:

Sören Schwertfeger & Chundong Wang

https://robotics.shanghaitech.edu.cn/courses/ca/20s/

School of Information Science and Technology SIST

ShanghaiTech University

Slides based on UC Berkley's CS61C

Agenda

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

Rank	Language	Type	Score
1	Python		100.0

An object-oriented, interpreted language that gains much of its power from a large constellation of libraries, including popular modules for machine learning and scientific computing.

Rank	Language	Type	Score
2	Java		96.3

An object-oriented language that creates code intended to be run on a virtual machine, allowing it to run on different platforms with little or no modification. Java is a popular choice for Web applications.

Rank	Language	Type	Score
3	C		94.4

C is used to write software where speed and flexibility is important, such as in embedded systems or high-performance computing.

Rank	Language	Type	Score
4	C++		87.5

Essentially an object-oriented version of C that proved to be a natural fit for software driven by graphical user interfaces.

Web

Enterprise

Mobile

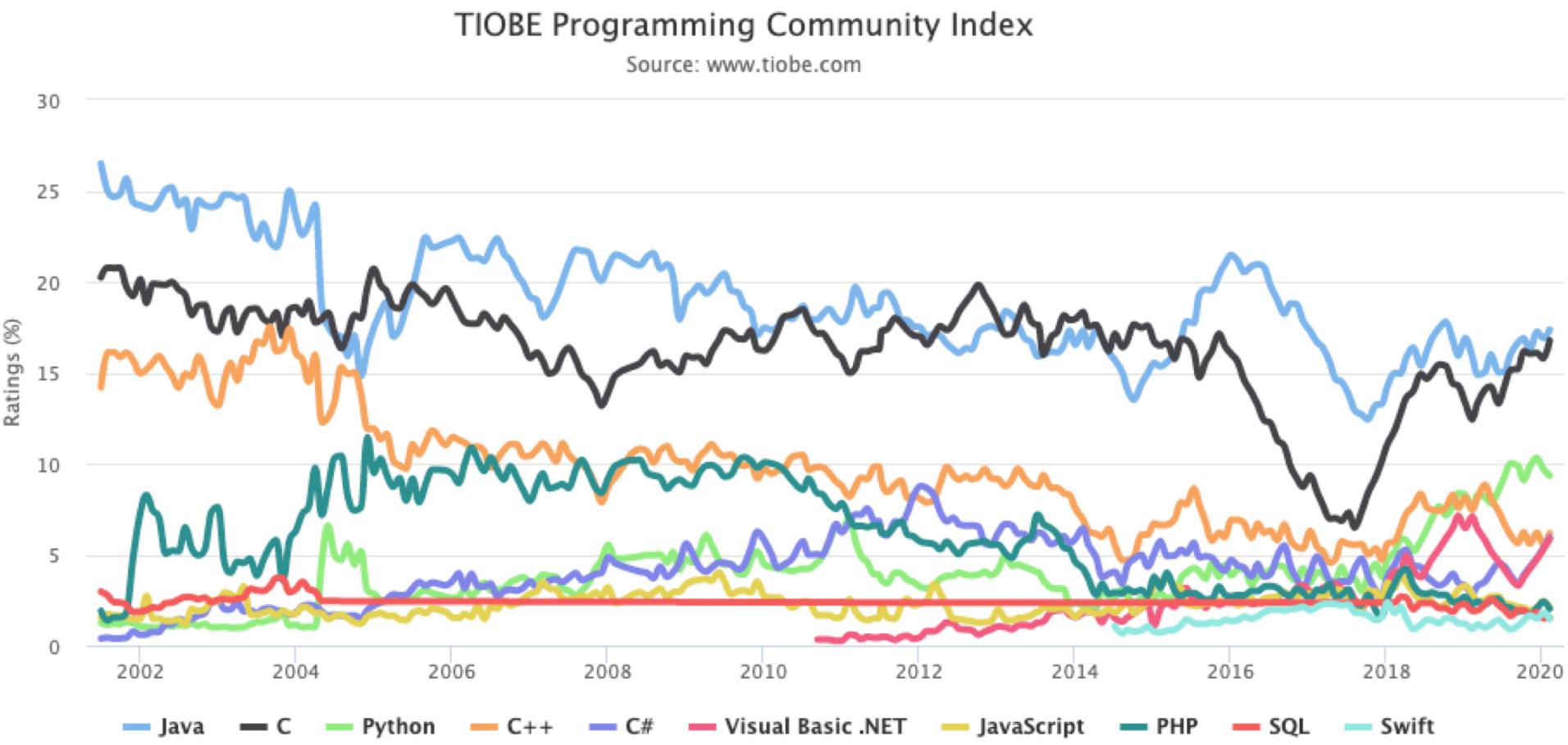
Embedded

Rank	Language	Type	Score
1	Python		100.0
2	Java		96.3
3	C		94.4
4	C++		87.5
5	R		81.5
6	JavaScript		79.4
7	C#		74.5
8	Matlab		70.6
9	Swift		69.1
10	Go		68.0
11	Arduino		67.2
12	HTML,CSS		66.8
13	PHP		65.1
14	Assembly		63.7

A catchall term for a vast family of processor instruction sets, writing assembly code requires considerable expertise, but it allows the creation of high-speed software that can run directly "on the metal."

Rank	Language	Type	Score
15	SQL		63.4

Previous slide: https://spectrum.ieee.org/computing/software/the-top-programming-languages-2019



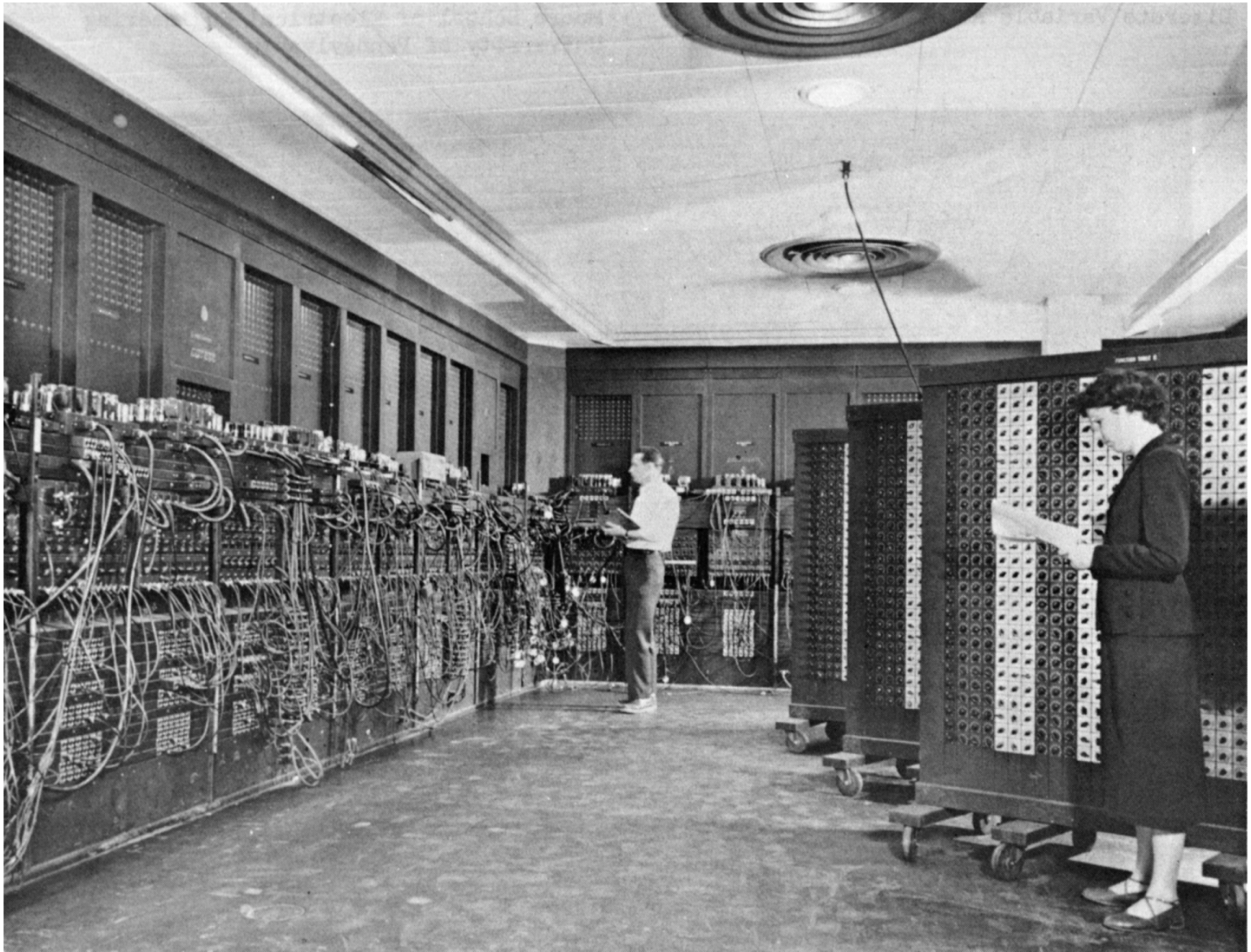
https://tiobe.com/tiobe-index/

Why You Need to Learn C!

CS 110 is NOT really about C Programming

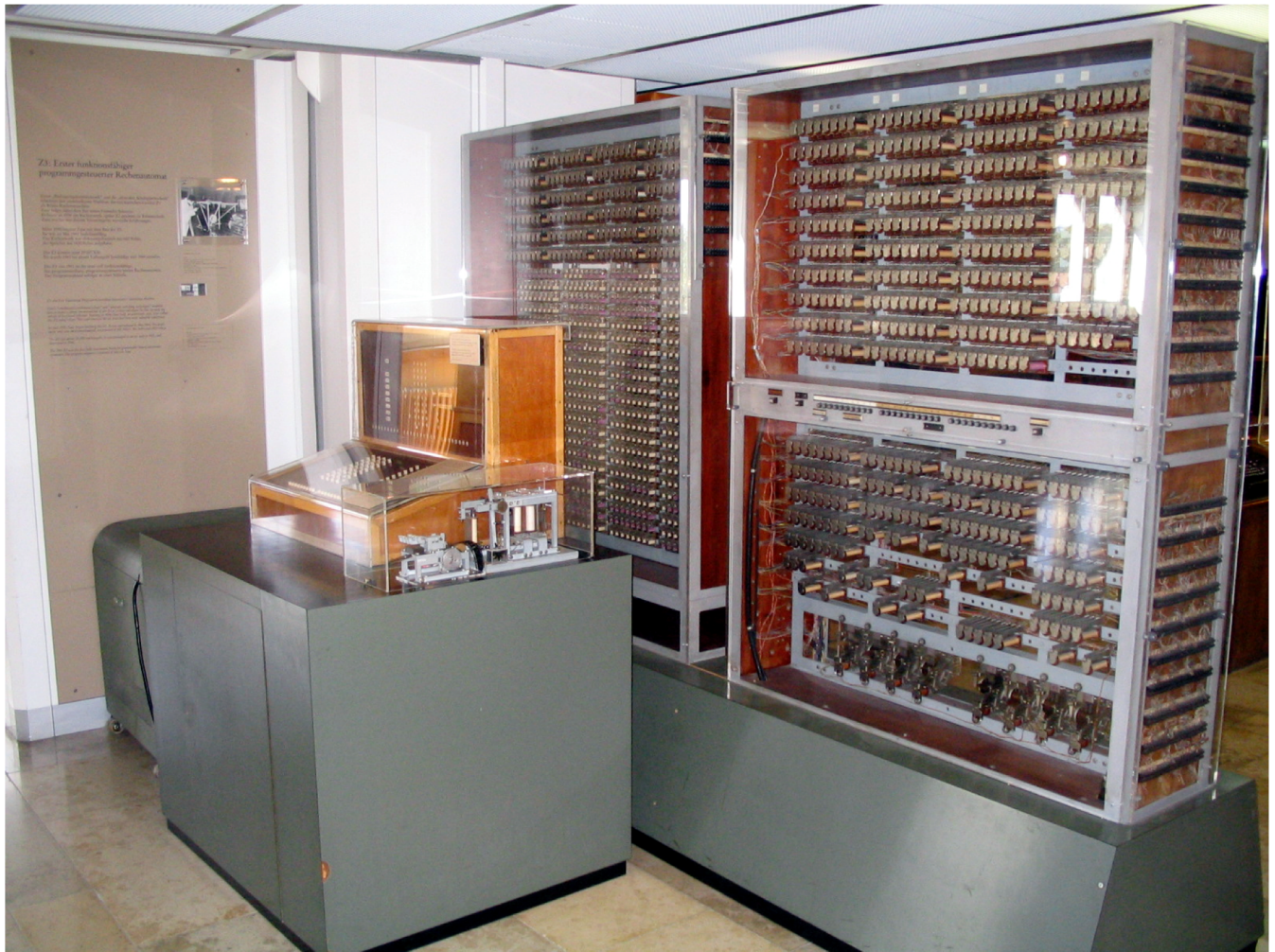
- It is about the *hardware-software interface*
  - What does the programmer need to know to achieve the highest possible performance
- C is close to the underlying hardware, unlike languages like Rust, Python, Java!
  - Allows us to talk about key hardware features in higher level terms
  - Allows programmer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

Old School Computer Architecture



Zuse Z3

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

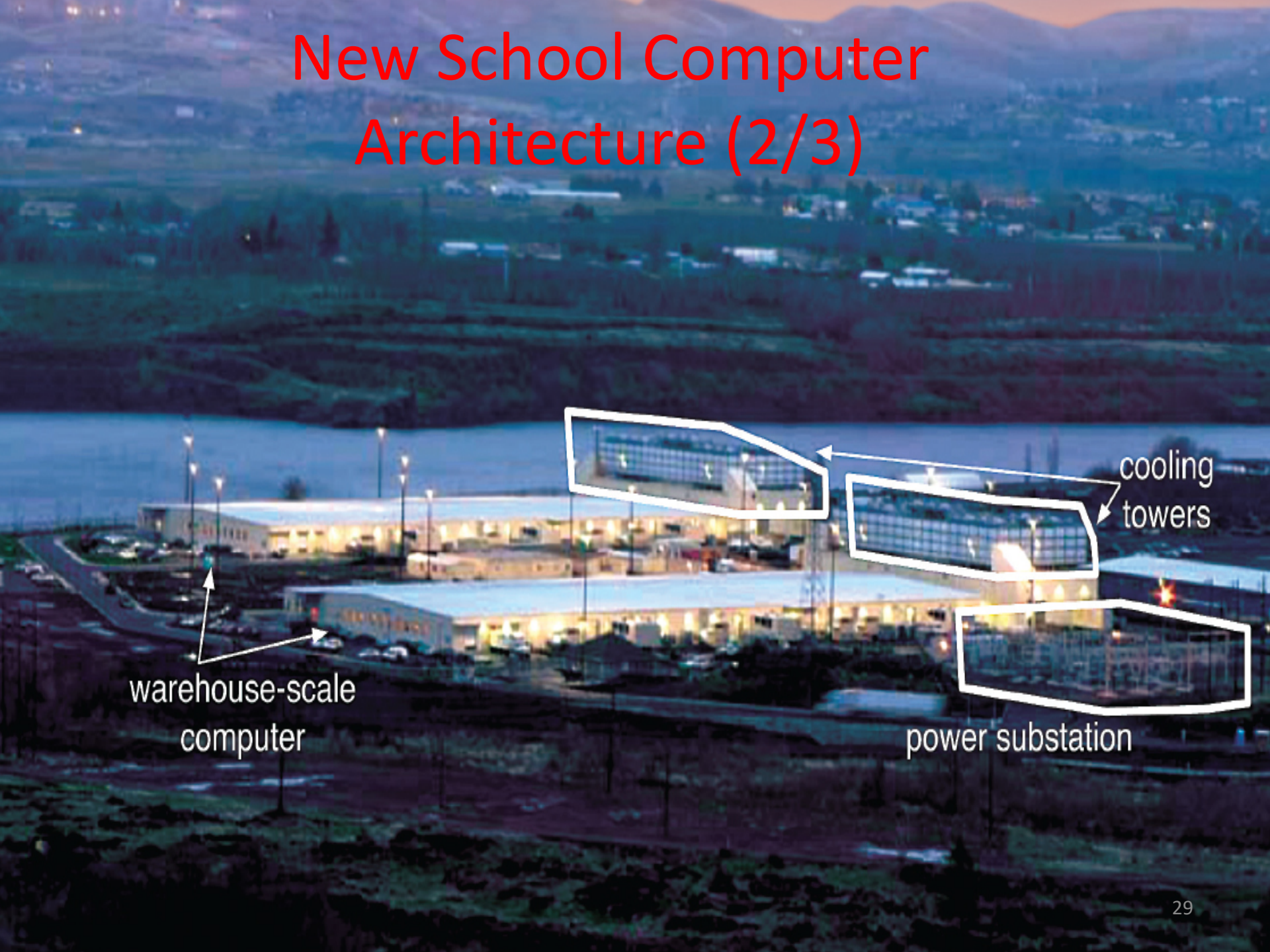


New School Computer Architecture (1/3)

Personal Mobile Devices

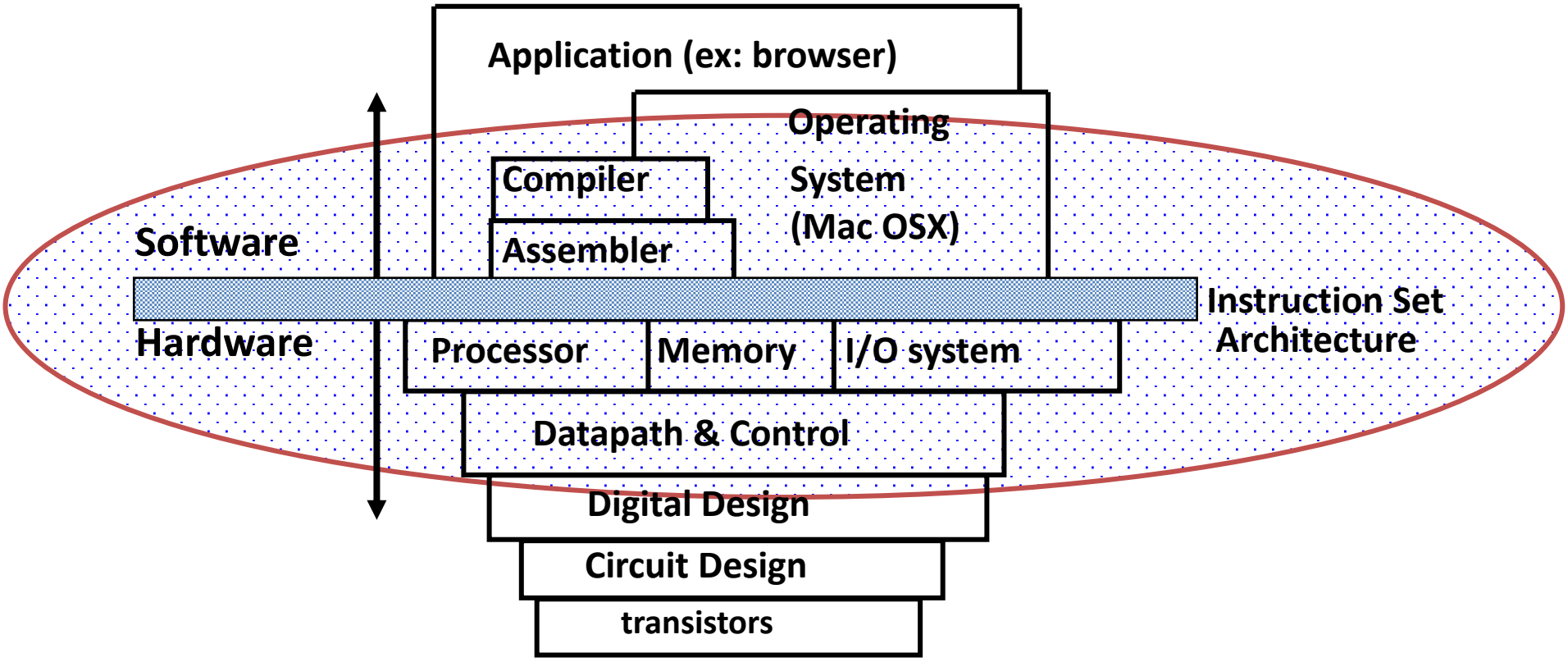
Network Edge Devices

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

Warehouse -Scale Computer

Smart Phone

Early 2018: Meltdown and Spectre

- Hardware vulnerability
- Affecting Intel x86 microprocessors, IBM POWER processors, and some ARM-based microprocessors
- All Operating Systems effected!
- They are considered "**catastrophic**" by security analysts!
- Allow to read all memory (e.g. from other process or other Virtual Machines (e.g. other users data on Amazon cloud service!))
- Towards the end of this CA course you can understand the basics of how Meltdown and Spectre work. Keywords:
  - Virtual Memory; Protection Levels; Instruction Pipelining; Speculative Execution; CPU Caching;

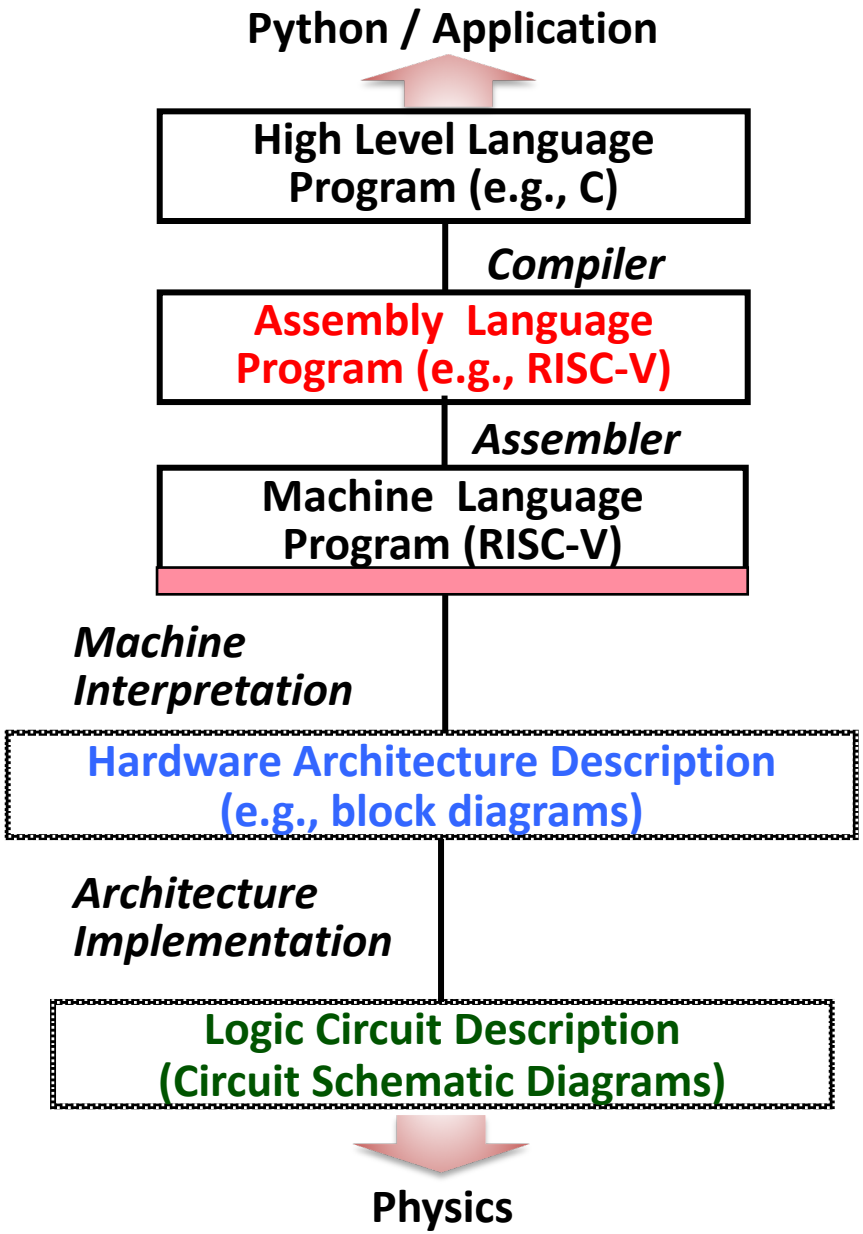
Agenda

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

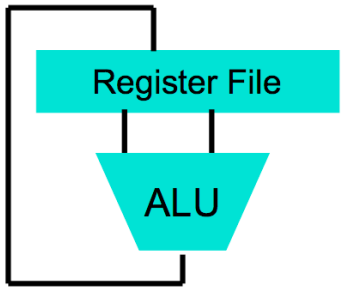


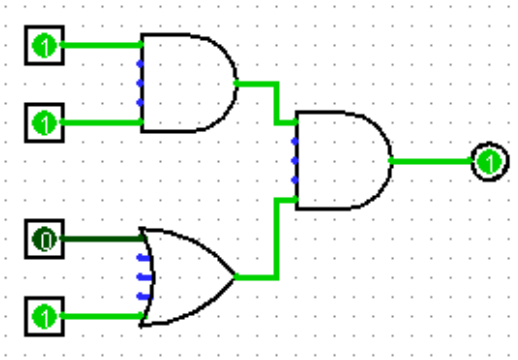

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
lw    t0, 0(s2)
lw    t1, 4(s2)
sw    t1, 0(s2)
sw    t0, 4(s2)
```

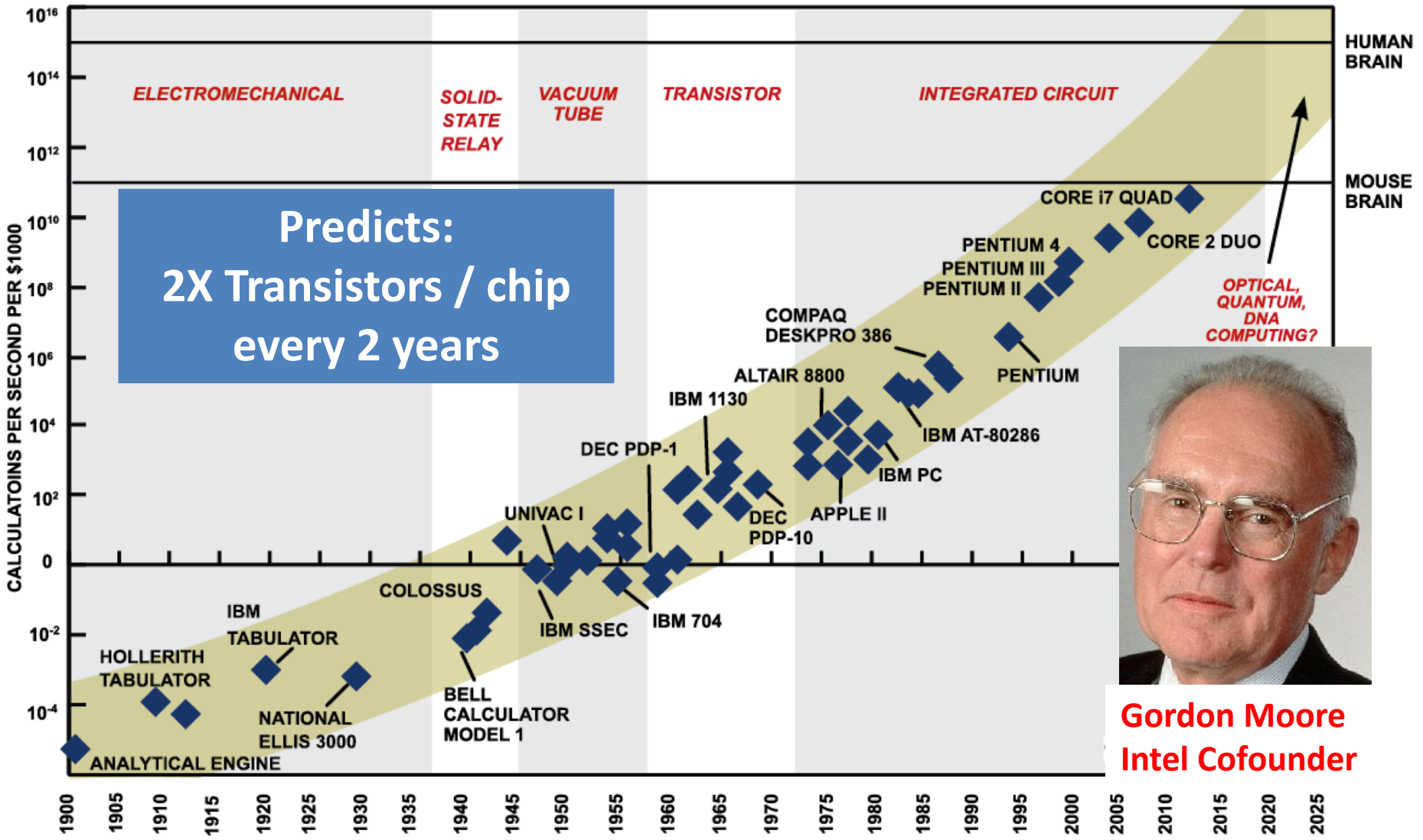
Anything can be represented as a *number*, i.e., data or instructions

```
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



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 <7nm, might have greater cost/transistor !!!!
So, why shrink????

Samsung's V1 fab opened Feb 2020

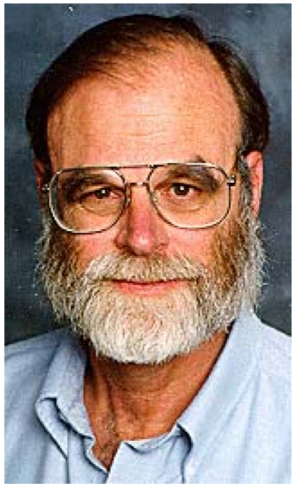
Now: 7 & 6 nm process

Future: up to 3 nm

Investment:
6,000,000,000 USD
(4,2 million 万元)

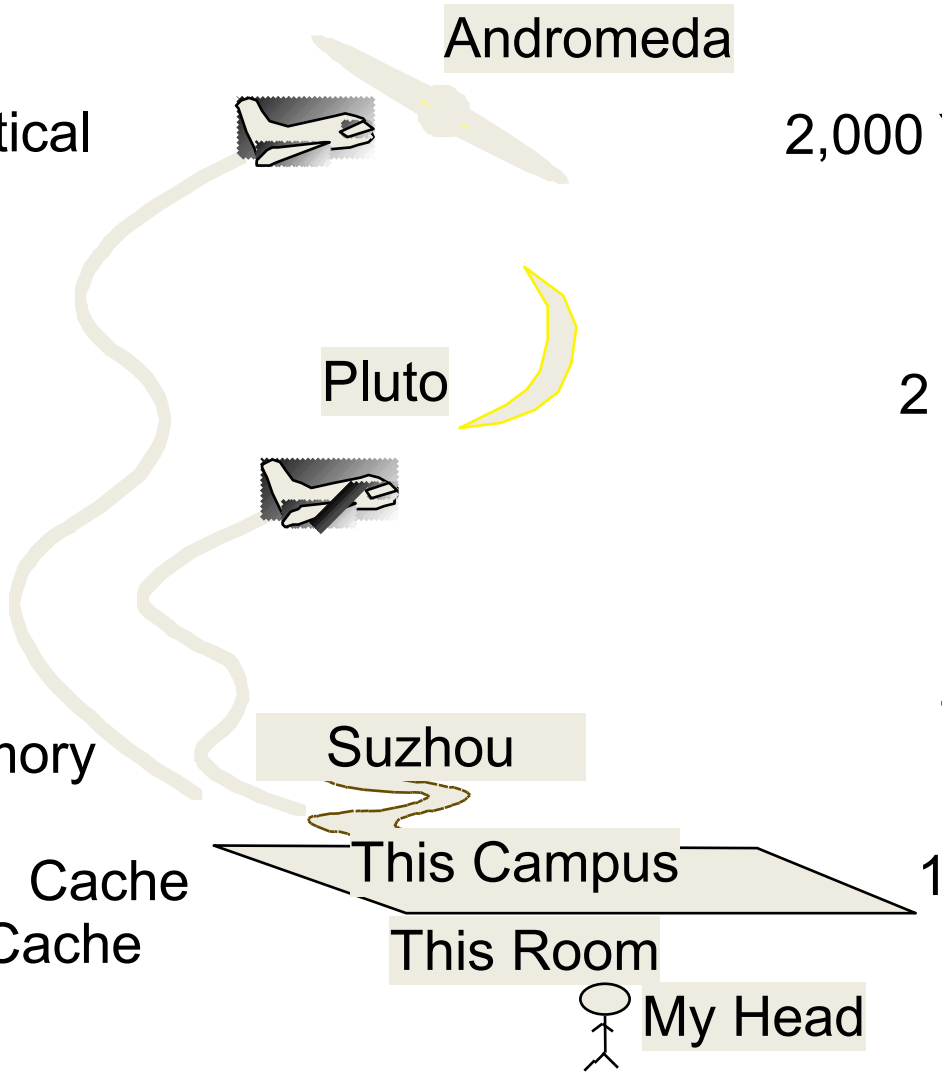
Other notable players: Intel, TSMC, GlobalFoundries (AMD), IBM
China: SMIC: 14nm production, developing 10 and 7nm

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

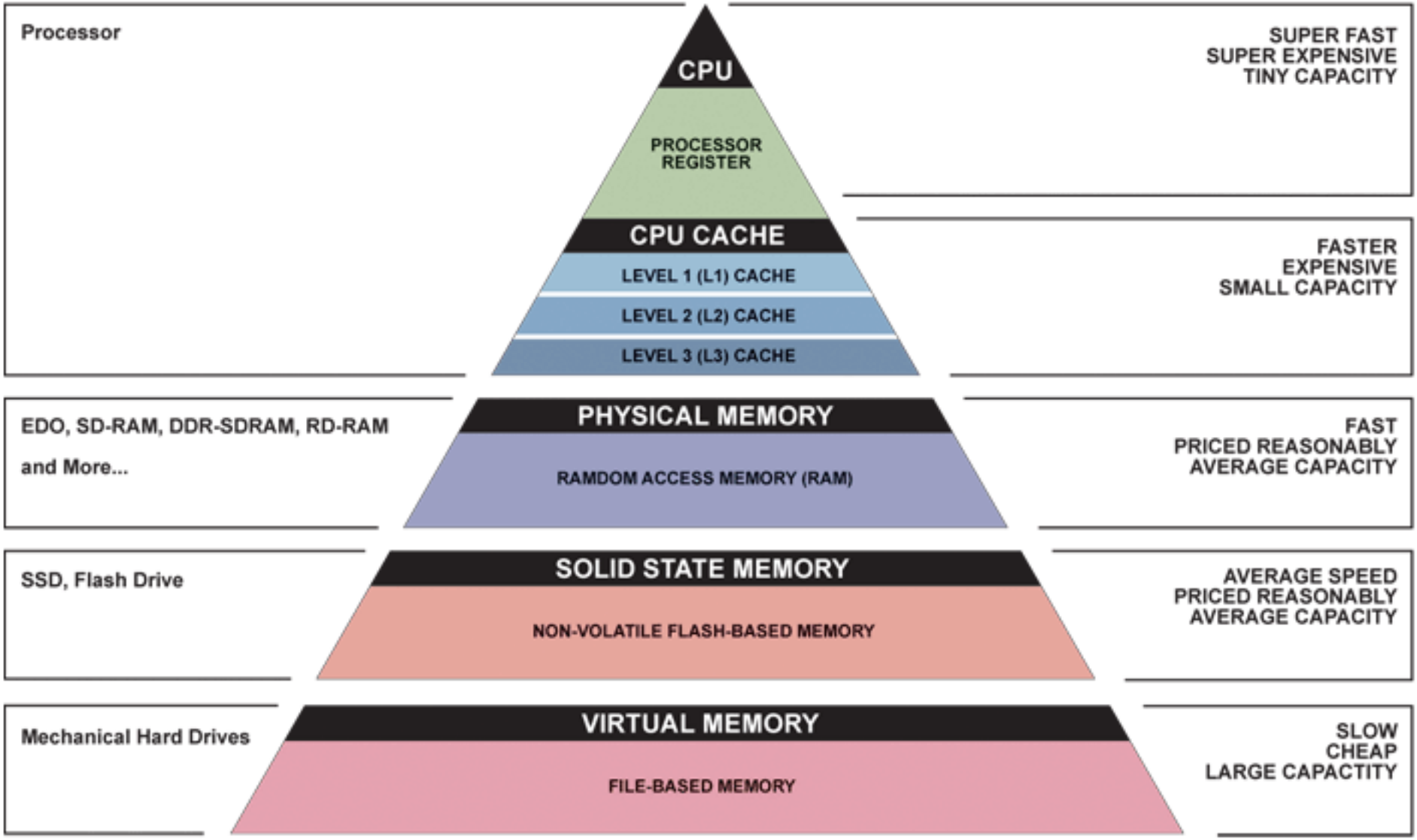


Jim Gray
Turing Award

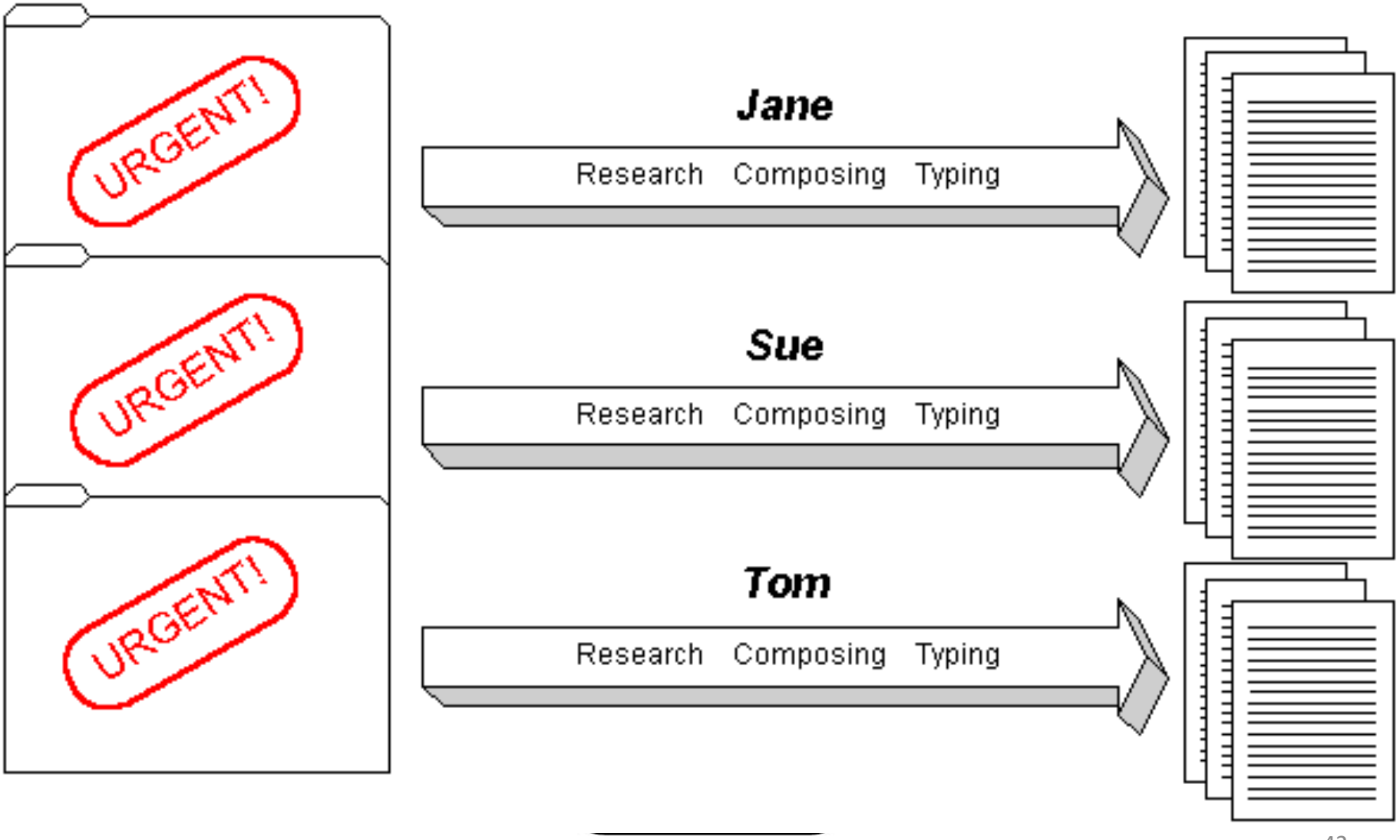
10^9	Tape /Optical Robot	2,000 Years
10^6	HardDisk	2 Years
100	Main Memory	1.5 hr
10	On Board Cache	10 min
2	On Chip Cache	
1	Registers	1 min
(ns)		



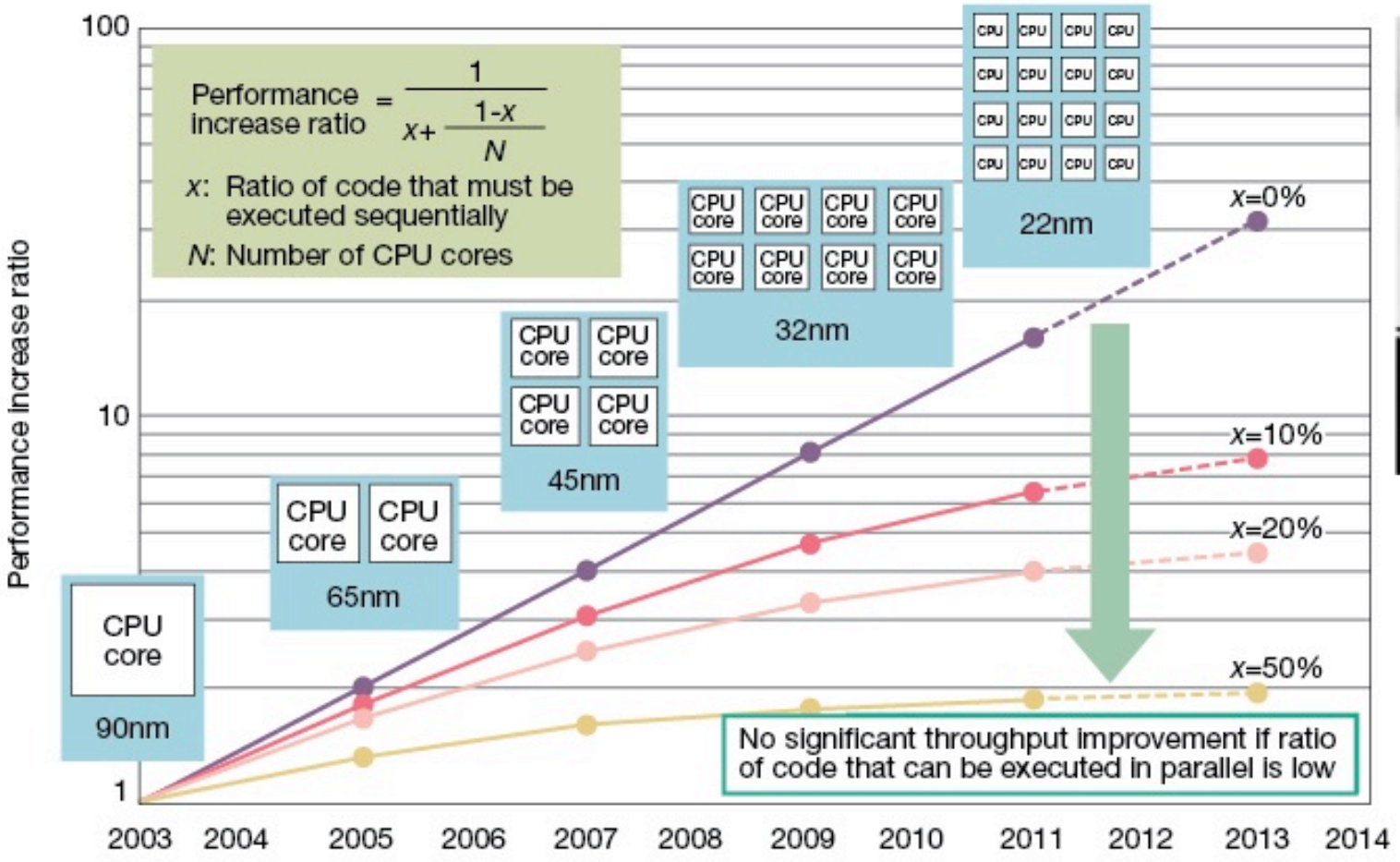
Great Idea #3: Principle of Locality/ Memory Hierarchy



Great Idea #4: Parallelism



Caveat: Amdahl's Law



Gene Amdahl
Computer Pioneer

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.

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*

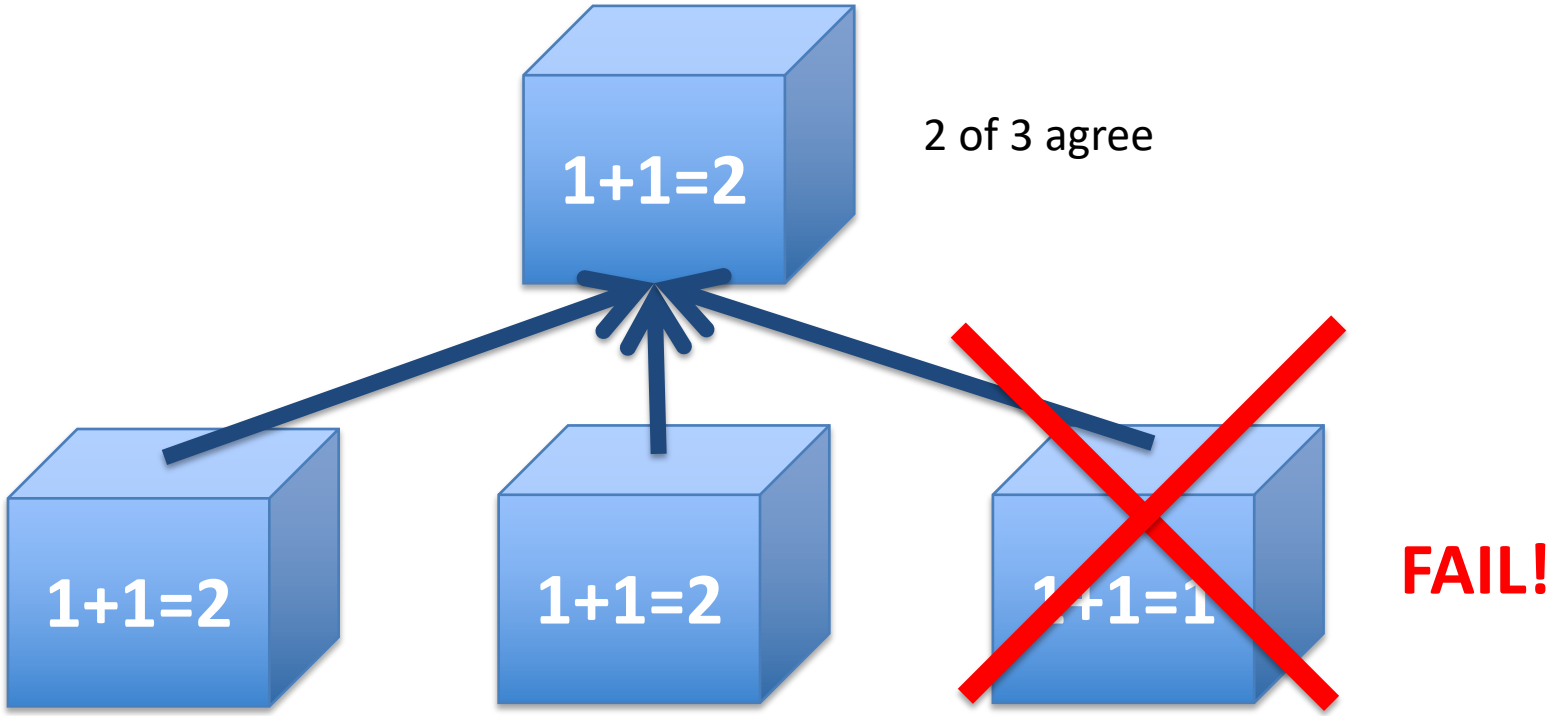
Coping with Failures

- 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

50,0 disks
200,00 sks fail
365 days 60 hours

Great Idea #6: Dependability via Redundancy

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

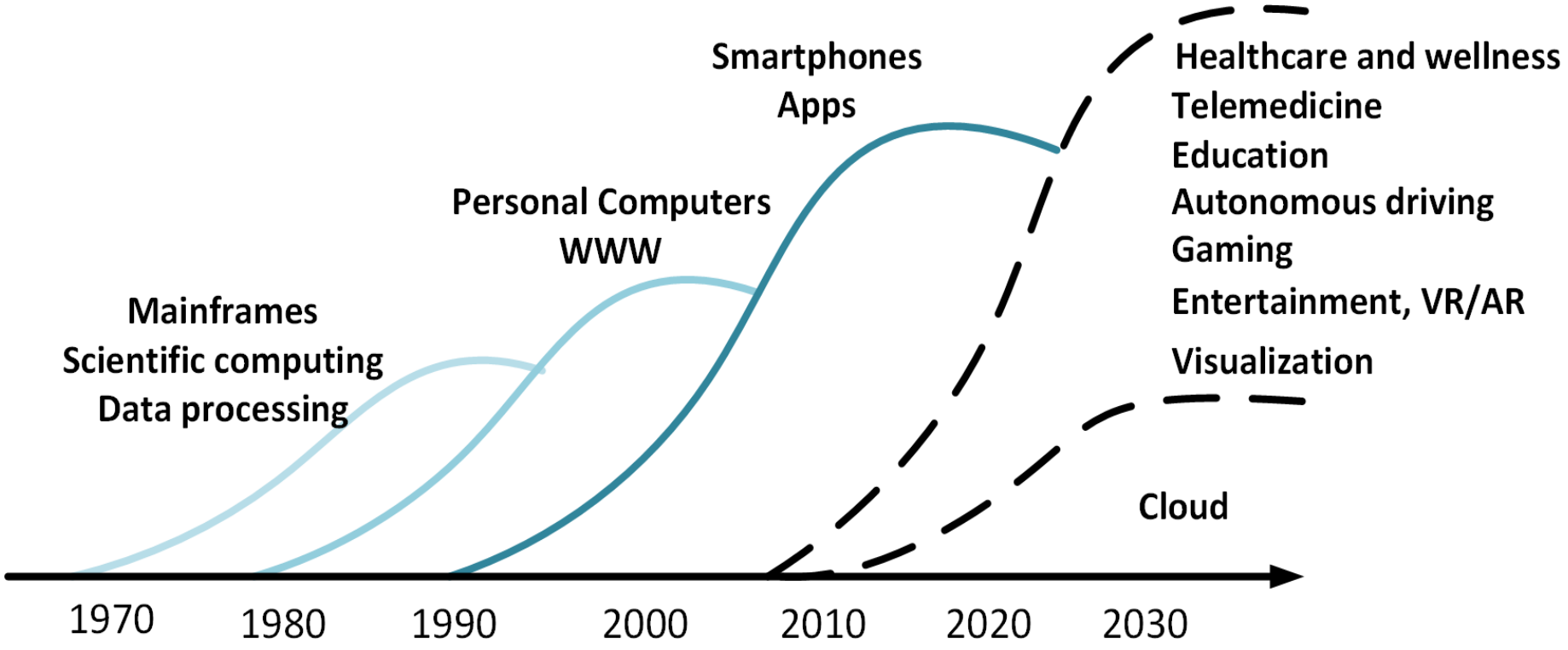


Increasing transistor density reduces the cost of redundancy

Great Idea #6: Dependability via Redundancy

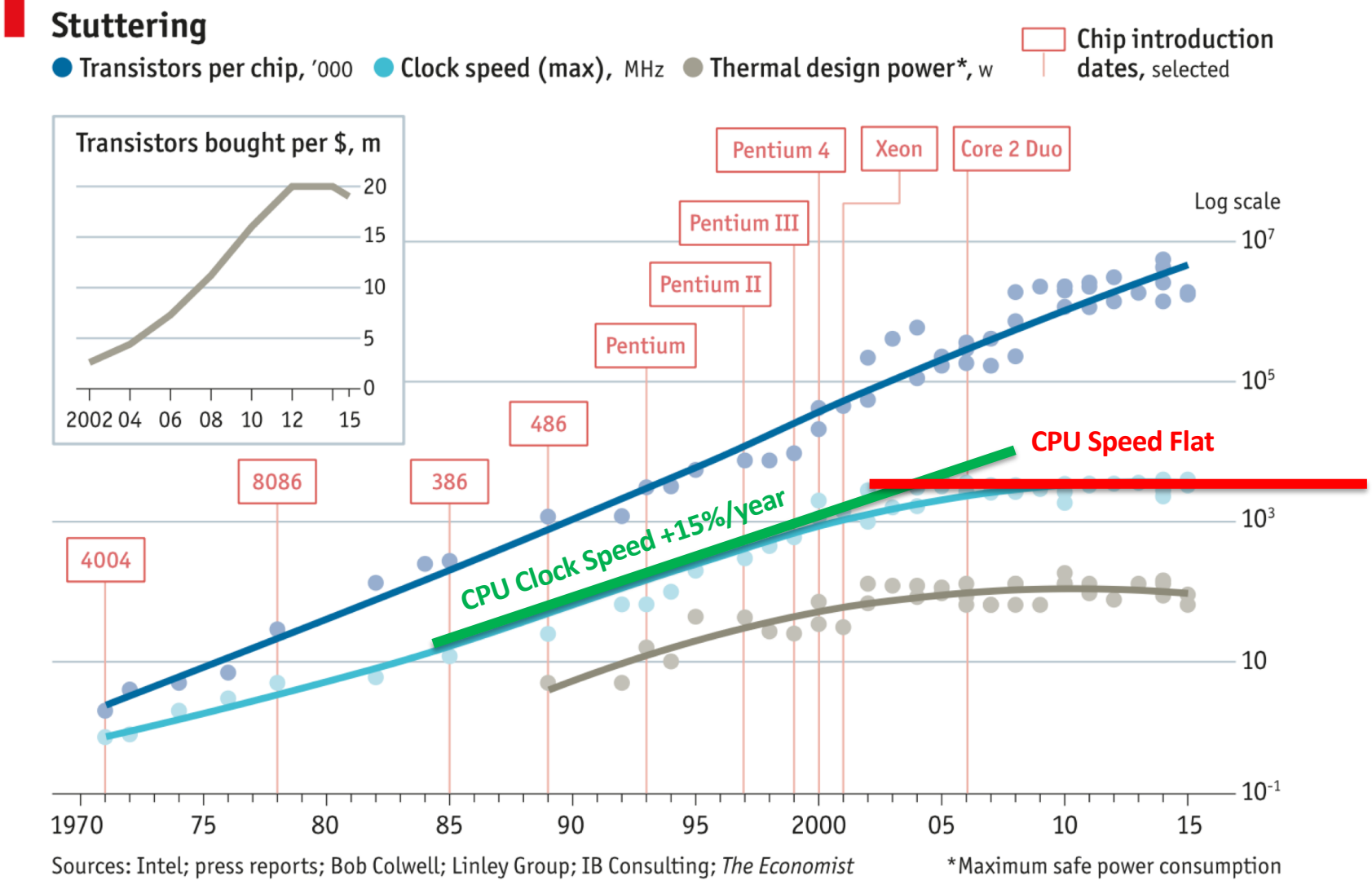
- Applies to everything from datacenters to storage to memory to instructors
  - Redundant datacenters so that can lose 1 datacenter but Internet service stays online
  - Redundant disks so that can lose 1 disk but not lose data (Redundant Arrays of Independent Disks/RAID)
  - Redundant memory bits of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

Why is Architecture Exciting Today?



- Number of deployed devices continues growing, but no single killer app
  - Diversification of needs, architectures

Why is Architecture Exciting Today?

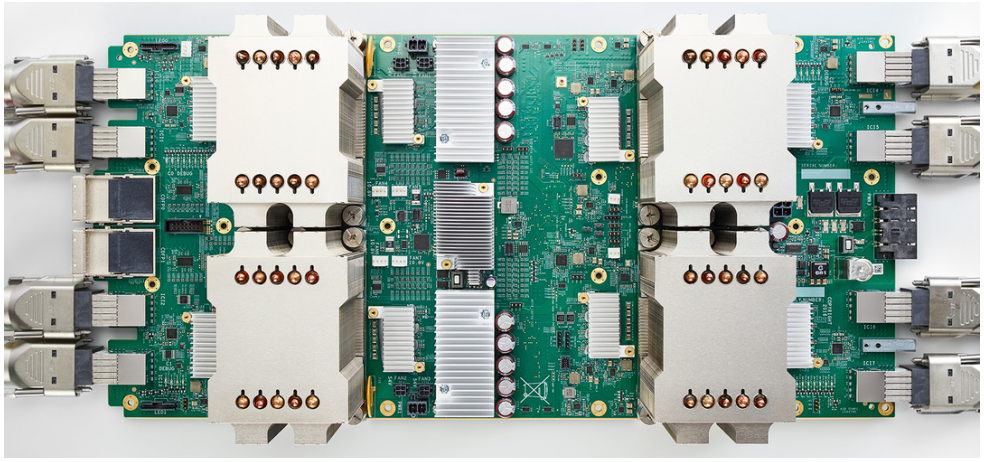


Old Conventional Wisdom

- Moore's Law + Dennard Scaling = faster, cheaper, lower-power general-purpose computers each year
- In glory days, 1%/week performance improvement!

- Dumb to compete by designing parallel or specialized computers
- By time you've finished design, next generation of general-purpose will beat you

New Conventional Wisdom



Google TPU2
Specialized Engine for NN training
Deployed in cloud
45 TFLOPS/chip

Serious heatsinks!

Next

- Go to piazza, write *some* reply in the "Lecture 1 chatter" post – keep it civil!

- Read textbook P&H: 2.4 "Signed and Unsigned Numbers"

- Afterwards: Watch last video of lecture 1

CS 110
Computer Architecture
(a.k.a. Machine Structures)
Lecture 1: *Course Introduction*
Part 3: Number Repesentation

Instructors:
Sören Schwertfeger & Chundong Wang

https://robotics.shanghaitech.edu.cn/courses/ca/20s/

School of Information Science and Technology SIST

ShanghaiTech University

Slides based on UC Berkley's CS61C

Agenda

- What you need to know about this class
- Thinking about Machine Structures
- Great Ideas in Computer Architecture
- 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:
- $123_{10} = 1_{10} \times 10_{10}^{2} + 2_{10} \times 10_{10}^{1} + 3_{10} \times 10_{10}^{0}$

  $= 1x100_{10} + 2x10_{10} + 3x1_{10}$

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

  $= 123_{10}$
- 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
- $FFF_{hex} = 15_{ten} \times 16_{ten}^{2} + 15_{ten} \times 16_{ten}^{1} + 15_{ten} \times 16_{ten}^{0}$
  $= 3840_{ten} + 240_{ten} + 15_{ten}$
  $= 4095_{ten}$
- $1111\ 1111\ 1111_{two} = FFF_{hex} = 4095_{ten}$
- 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}

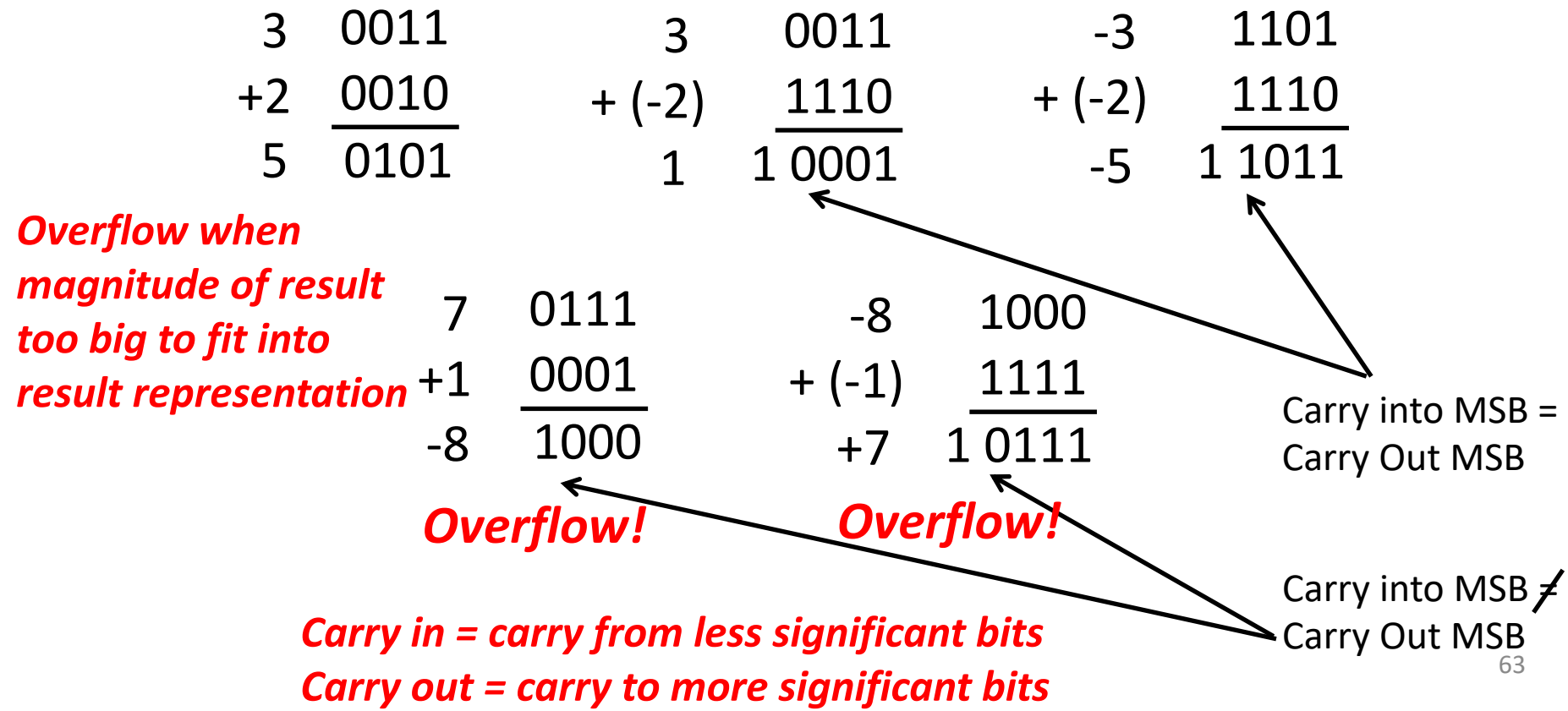
Bitwise complement 1100_{two}

$+ \quad 1_{two}$

-3_{ten} 1101_{two}

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?

A -32 to +31

B 0 to +31

C -16 to +15

D -15 to +16

Go to piazza, participate in the poll "Lecture 1 5-bit poll"

You may discuss number representation in piazza, but do not give the answer (A, B, C or D) away!

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!