Practice 15

System questions

Review & Discussion

- Why is **bit** the most fundamental unit of information in computers?
- What are the differences between doing a task with **hardware** and **software**?
- What is “**stored program**” concept?
- How many **bits** are needed to identify all memory locations in 32 GB memory?
- If your computer is a **64-bit** machine, what does it actually mean?

Computing Bootcamp

Great Ideas in Computer Architecture, Toward High-level Language

Lecture 23

Hyung-Sin Kim

SNU Graduate School of Data Science

Contents

- Great ideas in computer architecture
  - Abstraction
  - Moore’s Law
  - Parallelism
  - Principle of Locality
  - Dependability via Redundancy
- From machine codes to High-level languages

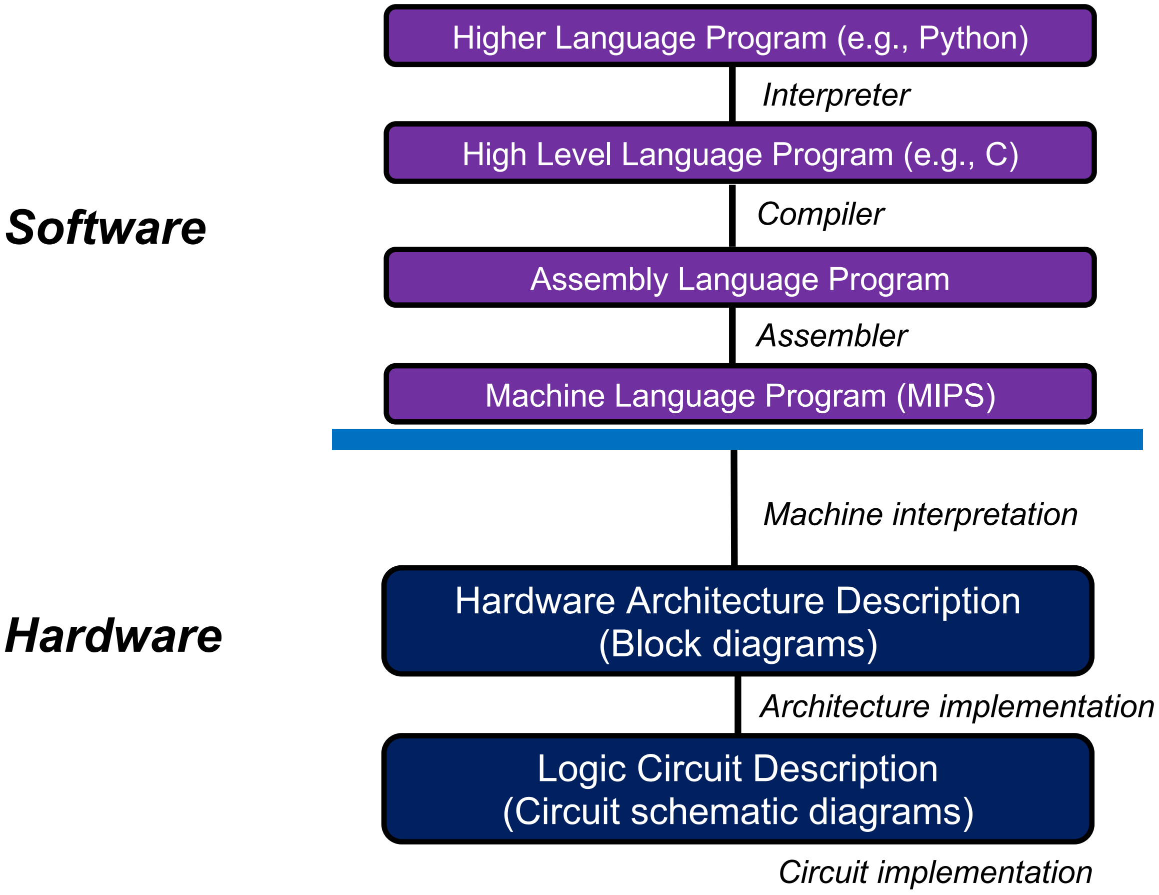
Great ideas in computer architecture

5 Great Ideas in Computer Architecture

- Abstraction: Layers of representation and interpretation
- Moore’s Law
- Parallelism
- Principle of Locality
- Dependability via Redundancy

Abstraction

#1 Abstraction



Software

Higher Language Program (e.g., Python)

Interpreter

High Level Language Program (e.g., C)

Compiler

Assembly Language Program

Assembler

Machine Language Program (MIPS)

Hardware

Machine interpretation

Hardware Architecture Description
(Block diagrams)

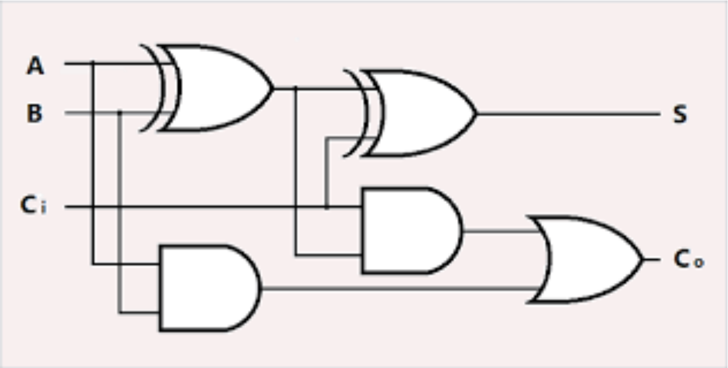
Architecture implementation

Logic Circuit Description
(Circuit schematic diagrams)

Circuit implementation

#1 Abstraction – Logic Gate

- AND, NOT, XOR, NAND, OR ….



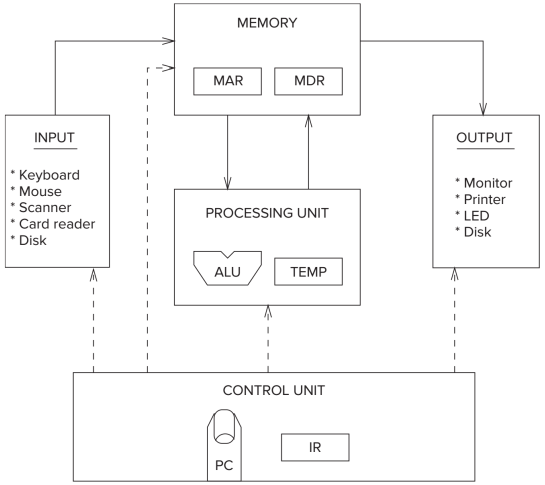
- Gate designers
  - Draw schematics above to provide more complex operations (ADD, Register…) by using logic gates
  - Do not have to consider how each logic gate is implemented as an electronic circuit

Logic Circuit Description
(Circuit schematic diagrams)

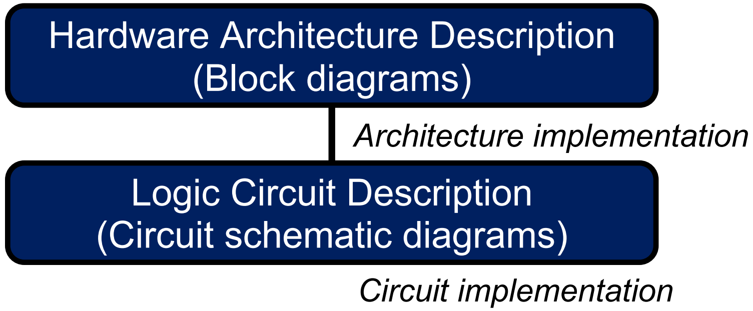
Circuit implementation

#1 Abstraction – Hardware Architecture

- ALU, Register…



- Architecture designers
  - Draw block diagrams of a computer by using many hardware modules
  - Do not have to consider how each module is implemented as logic gates



Hardware Architecture Description (Block diagrams)

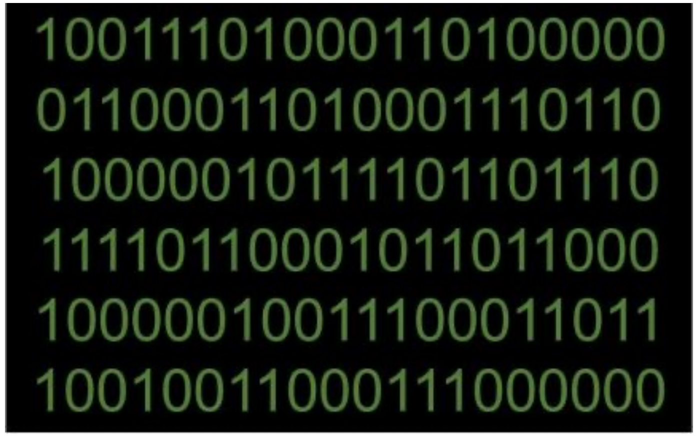
Architecture implementation

Logic Circuit Description (Circuit schematic diagrams)

Circuit implementation

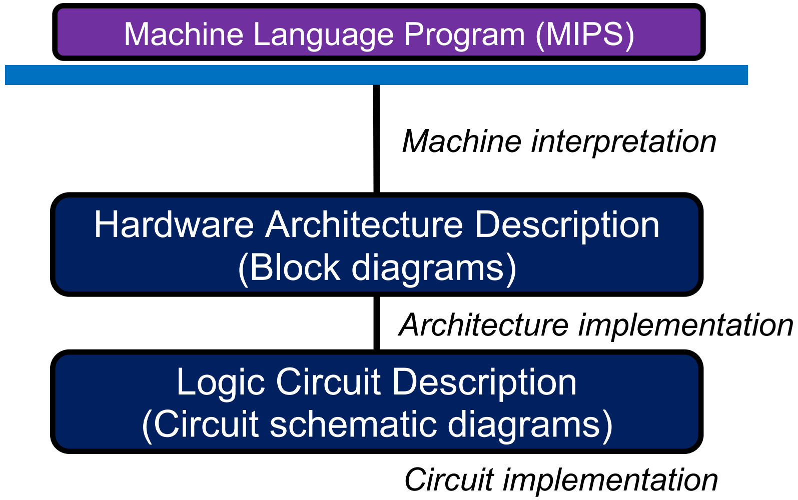
#1 Abstraction – Machine Language Program

- Just bit sequences in memory…



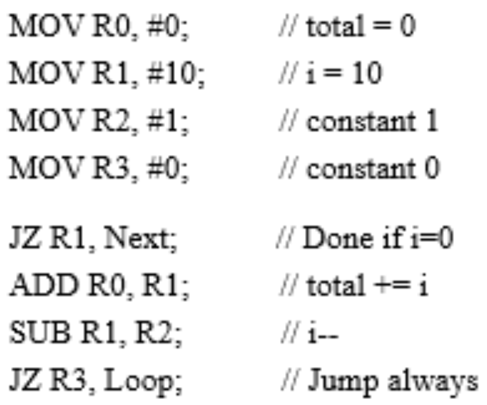
```
10011101000110100000
01100011010001110110
10000010111101101110
11110110001011011000
10000010011100011011
10010011000111000000
```

- Very hard for humans to figure out what’s going on…

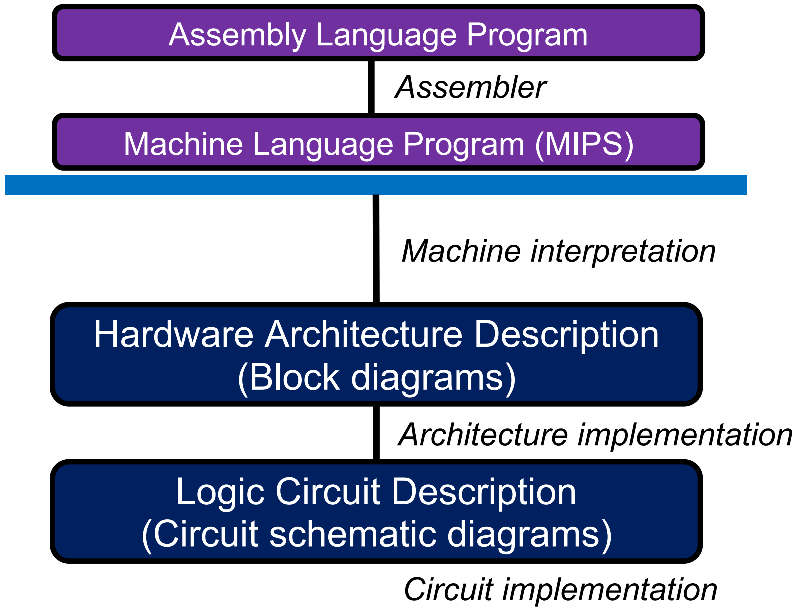

Machine Language Program (MIPS)
Machine interpretation
Hardware Architecture Description
(Block diagrams)
Architecture implementation
Logic Circuit Description
(Circuit schematic diagrams)
Circuit implementation

#1 Abstraction – Assembly Language Program

- Human-readable code, which is translated into machine code by assembler



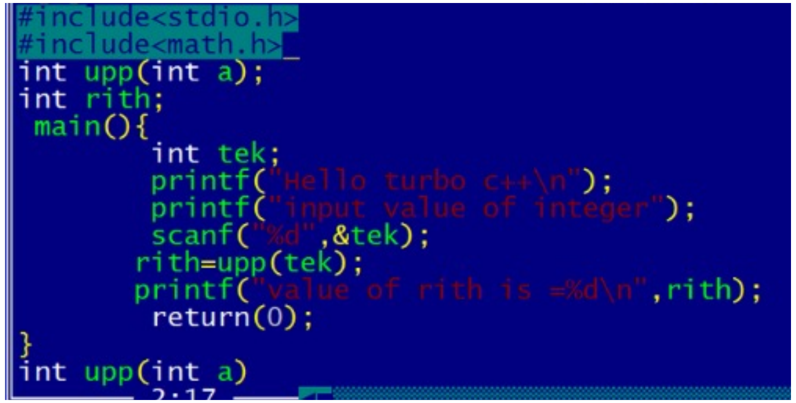
```
MOV R0, #0;      // total = 0
MOV R1, #10;     // i = 10
MOV R2, #1;      // constant 1
MOV R3, #0;      // constant 0

JZ R1, Next;     // Done if i=0
ADD R0, R1;      // total += i
SUB R1, R2;      // i--
JZ R3, Loop;     // Jump always
```


Assembly Language Program
Assembler
Machine Language Program (MIPS)
Machine interpretation
Hardware Architecture Description
(Block diagrams)
Architecture implementation
Logic Circuit Description
(Circuit schematic diagrams)
Circuit implementation

- Still incredibly tedious to write an assembly code …

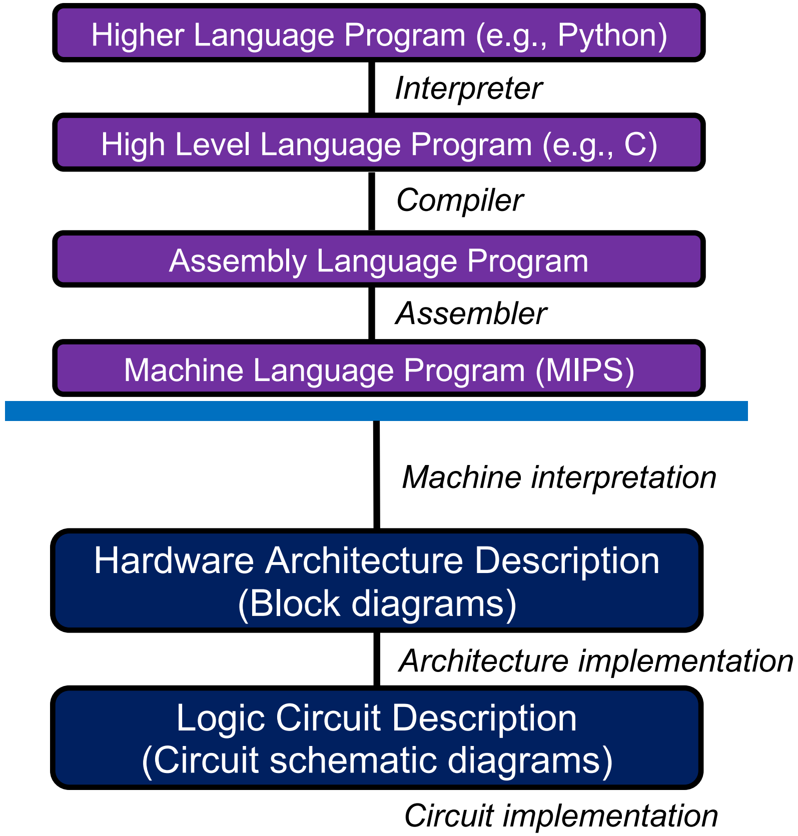
#1 Abstraction – High Language Program

- C language is not only human readable, but also can represent many lines of assembly codes as just a few lines



```
#include<stdio.h>
#include<math.h>
int upp(int a);
int rith;
 main(){
         int tek;
         printf("Hello turbo c++\n");
         printf("input value of integer");
         scanf("%d",&tek);
        rith=upp(tek);
        printf("value of rith is =%d\n",rith);
         return(0);
}
int upp(int a)
```

- These days, it is very rare to write an assembly code directly



Higher Language Program (e.g., Python)
Interpreter
High Level Language Program (e.g., C)
Compiler
Assembly Language Program
Assembler
Machine Language Program (MIPS)
Machine interpretation
Hardware Architecture Description (Block diagrams)
Architecture implementation
Logic Circuit Description (Circuit schematic diagrams)
Circuit implementation

Moore’s law

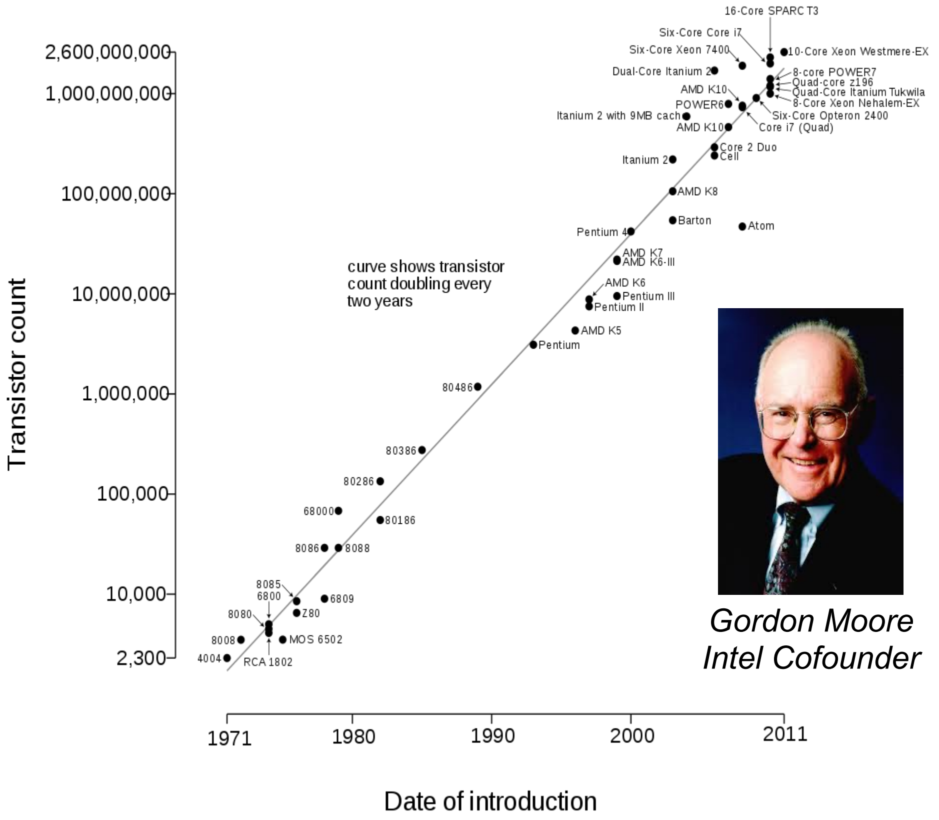
Computing Bootcamp

#2 Moore's Law [1965]

- Prediction
  - 2x transistors per chip, every 2 years
  - Shrink, shrink, and shrink…

- Until a decade ago, computer architecture people focused on designing a better single processor
  - Improved performance 50~60% every year (1000x in 10 years)
  - If you delay opening your proposal one week, everybody else gets 1% faster meanwhile

Microprocessor transistor counts 1971-2011 & Moore's law



Gordon Moore
Intel Cofounder

#2 Moore’s Law [1965] – Interesting Times

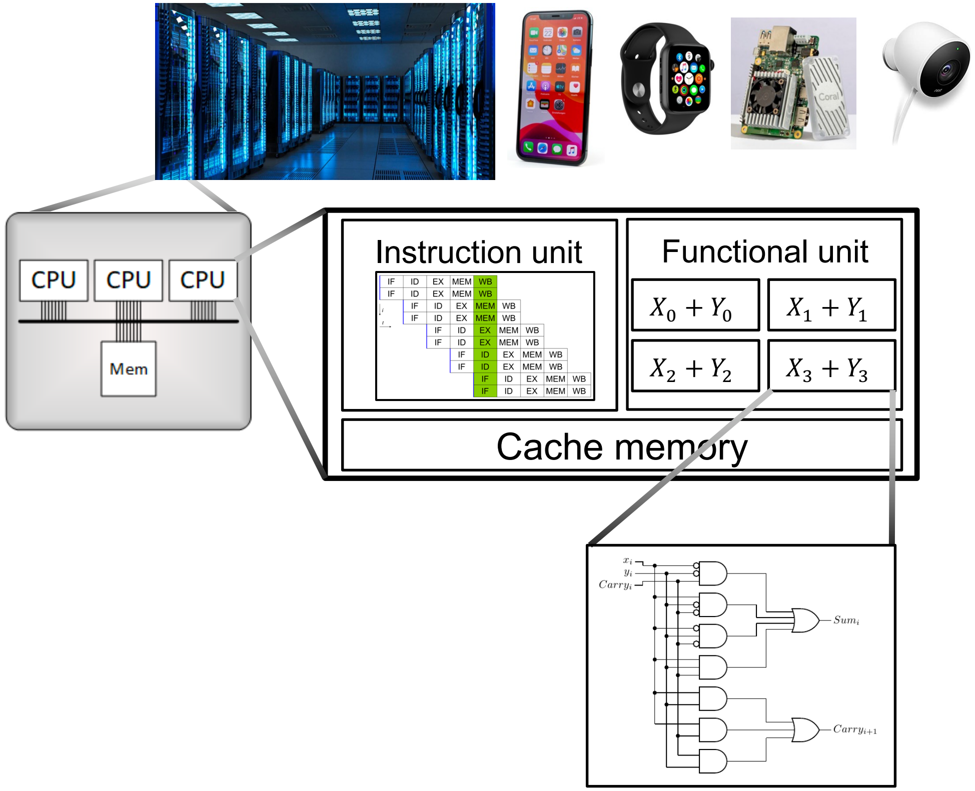
- Moore’s Law was based on
  - How many transistors can be in a single chip (making smaller transistors)
  - How cheap a transistor is (without increasing the cost)

- But in 2000s, this strategy hit the borderline (due to the law of physics)
  - Now a transistor (nm scale) becomes too small to contain many atoms
  - It is still possible but takes longer and is more expensive to do this (cost increase!)
  - Industry does not want a smaller transistor if it is expensive

Parallelism

#3 Parallelism

- Do multiple jobs at once by using multiple levels of parallelization!
  - Parallel requests
    - Assigned to computer
  - Parallel Threads
    - Assigned to core
  - Parallel Instructions
    - > 1 instruction at a time
  - Parallel Data
    - > 1 data item at a time (same operation)
  - Hardware descriptions
    - All gates functioning in parallel

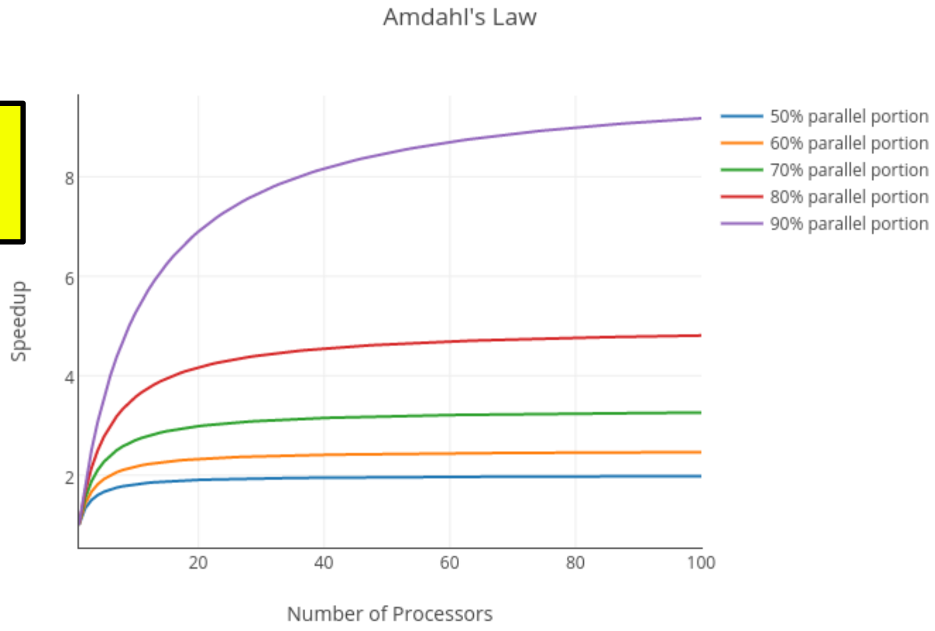


#3 Parallelism – Amdahl's Law

- Performance improvement by parallelism
  - If **X%** of your program has to be run sequentially (cannot be parallelized)
  - If you have **N** CPU cores

Gene Amdahl
(1922-2015)

$$\boxed{\frac{1}{X + \frac{1-X}{N}} \le \frac{1}{X}}$$



Principle of locality

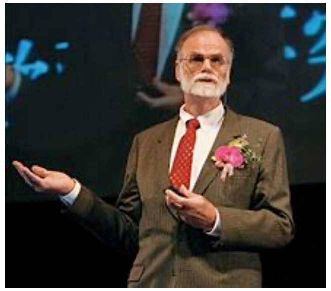
#4 Principle of Locality

- Principle of Locality
  - If a piece of data is used by a program, it is very likely to be used again **much sooner** than other data
- Memory type, size, location, and speed
  - **Type**: There are many different kinds of memory that provide different speed
    - Of course, a faster memory is more expensive
  - **Size**: If the technology is the same, a **bigger** memory is **slower** to access (law of physics, speed of light…)
  - **Location**: A memory closer to CPU is faster to access

Computer scientists figured out how to store data in **different types of memory**, considering **the principle of locality**!

#4 Principle of Locality – Analogy

- Jim Gray’s storage latency analogy: How far away is the data?



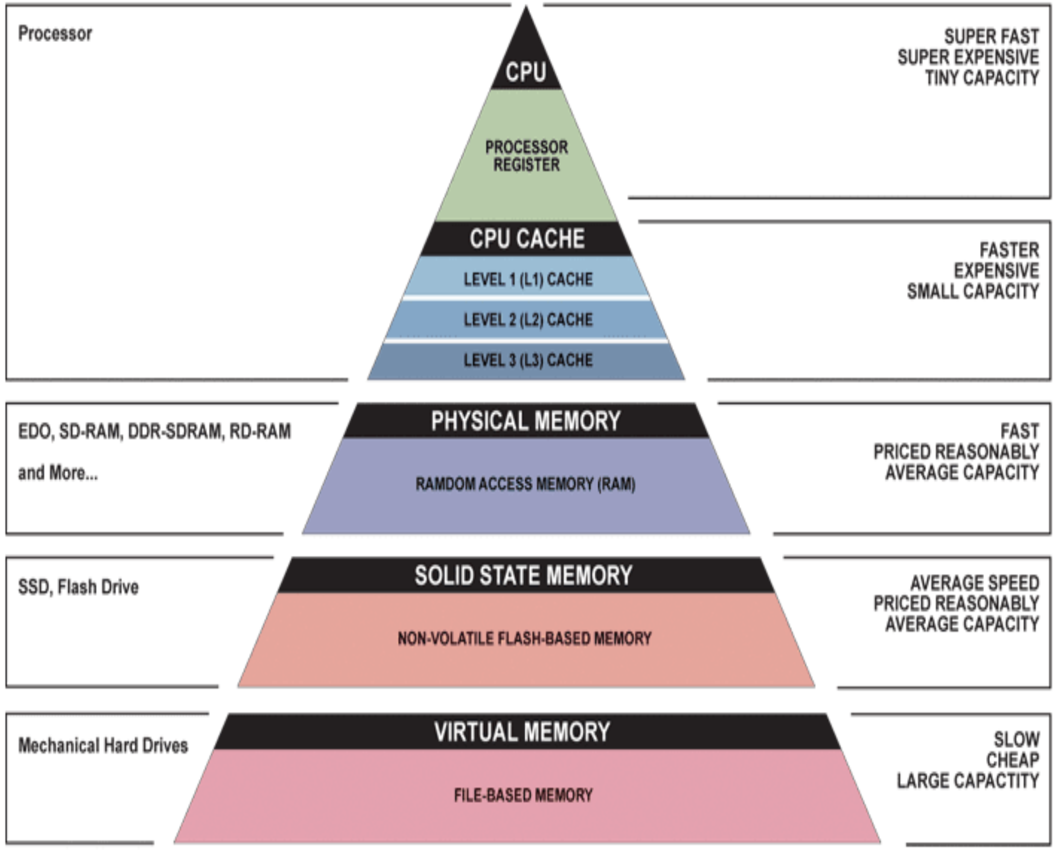
Jim Gray
(1944-2012)

10^9	Tape	*Andromeda*	*2,000 years*
10^6	Disk	*Pluto*	*2 years*
100	Memory	*Incheon airport*	*1.5 hours*
10	On-board cache	*SNU*	*10 minutes*
2	On-chip cache	*My office*	*2 minutes*
$1[ns]$	Registers	*My head*	*1 minute*

#4 Principle of Locality – Memory Hierarchy

- Store frequently used data in a faster and smaller storage!

- If there is no principle of locality, having this memory hierarchy will be meaningless
  - A computer with memory hierarchy will be just as slow as one having only the biggest memory

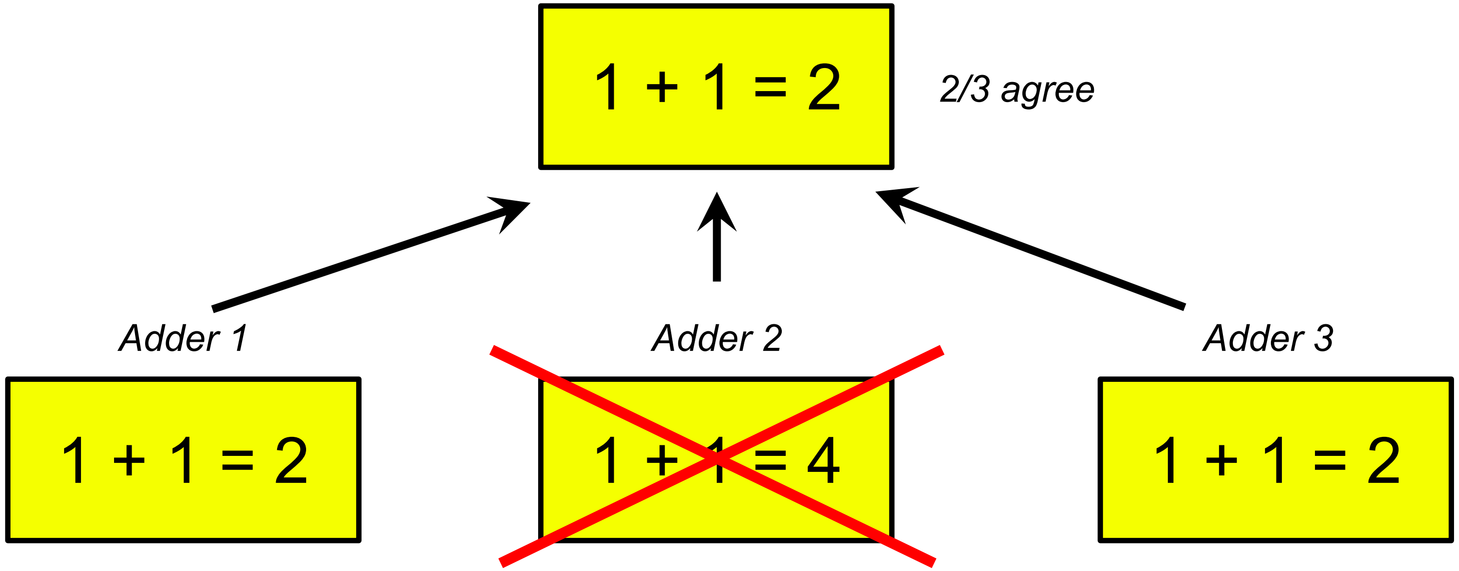


▲ Simplified Computer Memory Hierarchy
Illustration: Ryan J. Leng

Dependability via redundancy

#5 Dependability via Redundancy

- There are errors and failures everywhere. Computer is not an exception.
  - Computer systems achieve dependability via **redundancy**
  - The cost of redundancy is reduced as transistor density increases



#5 Dependability via Redundancy

- Redundant **datacenters** so that Internet service stays online even though 1 datacenter is lost
- Redundant **disks** so that data is preserved even though one disk is lost
  - Redundant arrays of independent disks/RAID
- Redundant **memory bits** so that data is preserved even though 1 bit is lost
  - Error correcting code / ECC memory

For More Details…

- Computer architecture course offered by ECE and CS