中国科学院大学
University of Chinese Academy of Sciences

CS101

# Systems Thinking

## Abstraction: Objectives and Properties

zxu@ict.ac.cn
zhangjialin@ict.ac.cn

## Outline

- What is systems thinking?
- Three objectives of systems thinking
  - Being thorough
  - Being systematic
  - Coping with complexity
- Abstraction
  - What is abstraction?
  - The COG properties of abstraction
  - One abstraction for many scenarios
  - Data abstractions
  - Control abstractions
- Modularization
- Seamless transition

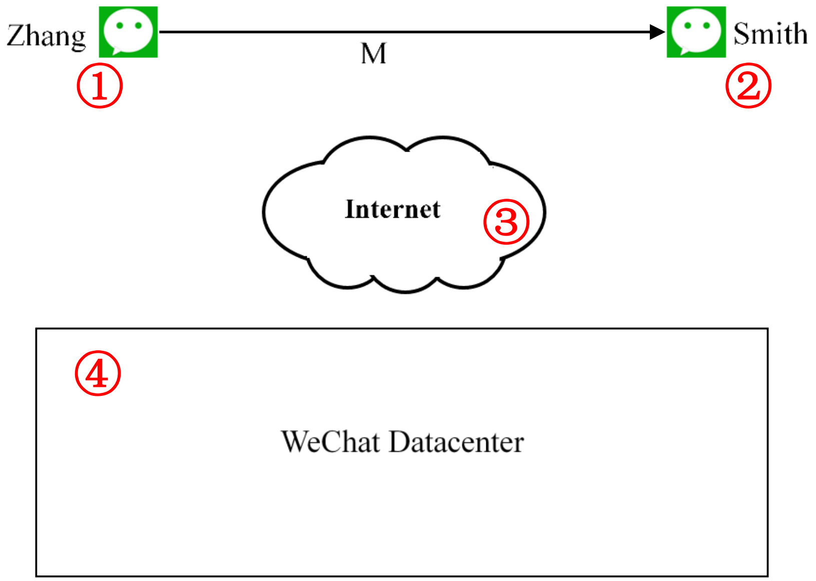
*These slides acknowledge sources for additional data not cited in the textbook*

## 1. What is systems thinking?

- Systems thinking makes computational process practical
- How?
  - Using abstractions to compose modules into a system, to enable seamless execution of computational processes
  - In this process, we need to accomplish the seeming impossible task of achieving three objectives simultaneously
    - Being thorough
    - Being systematic
    - Coping with complexity
- System thinking is a synergy of science, engineering, and art. A system designer is known as an architect

## Example: the WeChat message storage problem

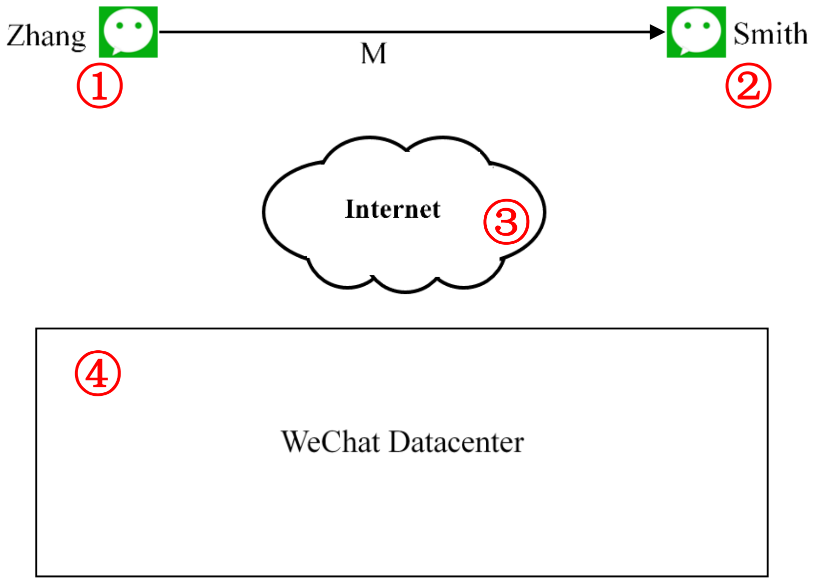
- Where to store a chat message M created by user Zhang?
- Possible choices
  1. At sender Zhang's device (phone)
  2. At receiver Smith's device (laptop)
  3. At WeChat's Datacenter
  4. Somewhere in the Internet



## Example: the WeChat message storage problem

- Where to store a chat message M created by user Zhang?
- Possible choices
  1. At sender Zhang's device (phone)
  2. At receiver Smith's device
  3. At WeChat's Datacenter
  4. Somewhere in the Internet

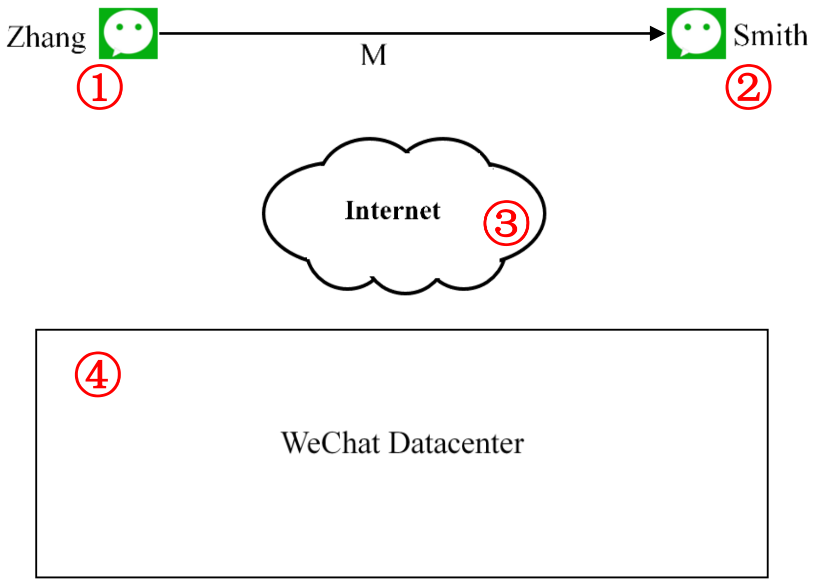
Another choice: Don't store it. M disappears after Smith receives it.



## Example: the WeChat message storage problem

- Where to store a chat message M created by user Zhang?
- Possible choices
    1. At sender Zhang's device (phone)
    2. At receiver Smith's device
    3. At WeChat's Datacenter
    4. Somewhere in the Internet

Another choice:
Don't store it. M disappears after Smith receives it.

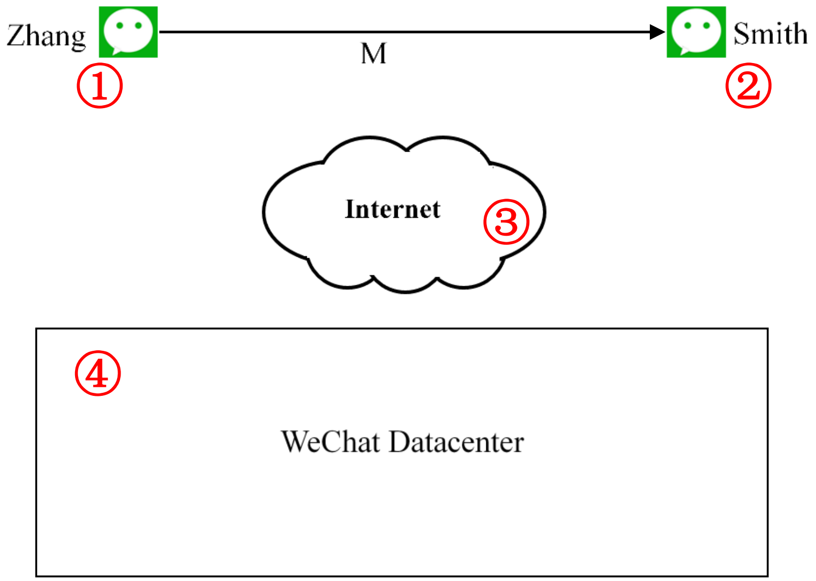


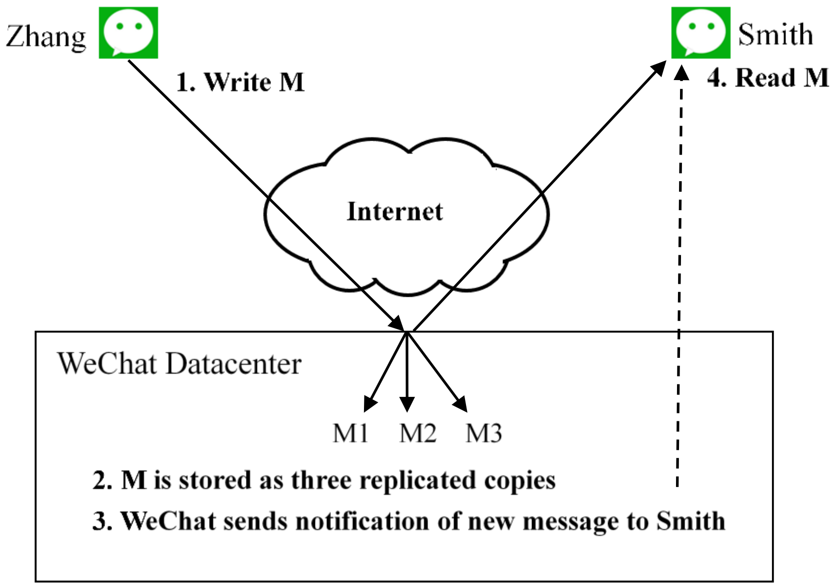
- Each choice leads to further unanswered problems
- **Logic and algorithmic thinking alone cannot solve this problem**
- Need thoughtful considerations and tradeoffs involving many issues of practical systems

## Example: the WeChat message storage problem

- Where to store a chat message M created by user Zhang?
- Possible choices
  1. At sender Zhang's device (phone)
  2. At receiver Smith's device
  3. At WeChat's Datacenter √
  4. Somewhere in the Internet

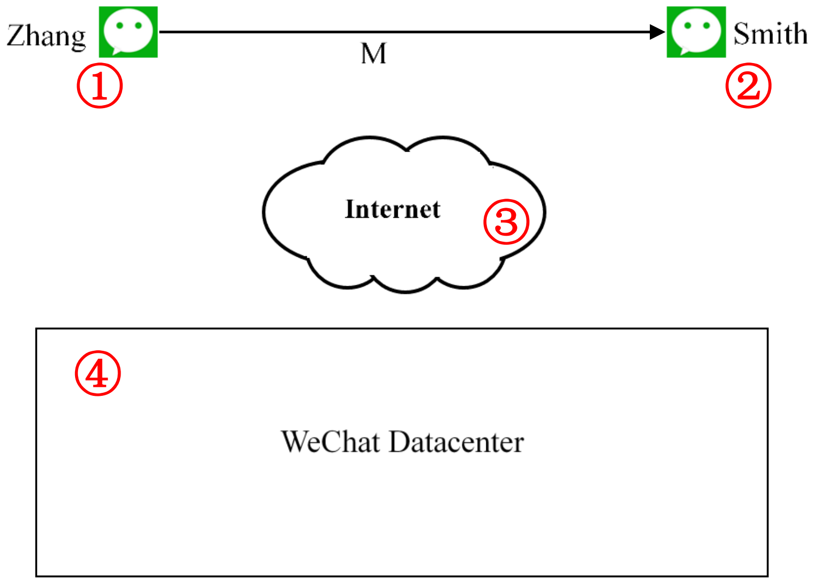
Message M is not directly sent from Zhang's device to Smith's device

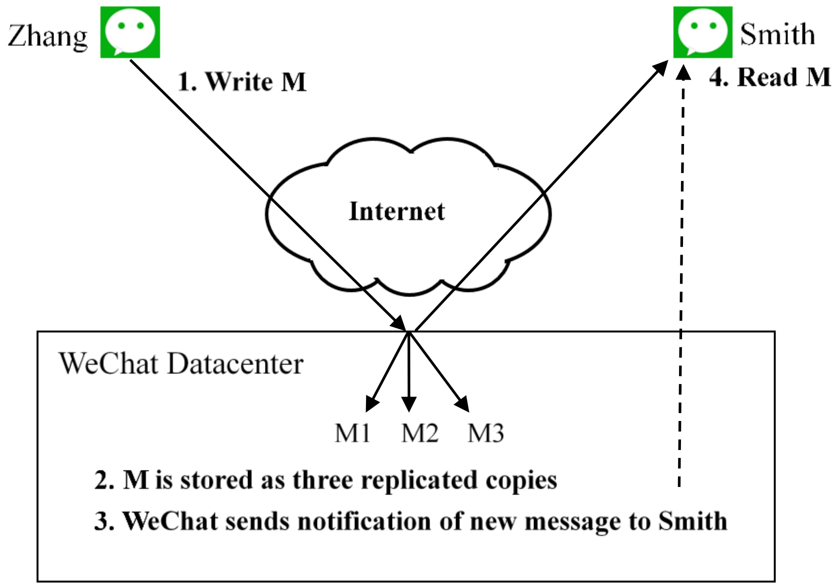



Zhang
Smith
1. Write M
4. Read M
Internet
WeChat Datacenter
M1
M2
M3
2. M is stored as three replicated copies
3. WeChat sends notification of new message to Smith

# 2.1 Being thorough

- Cover all the details of the whole system from end to end, ignoring no necessary details
  - Including functionality, convenience, performance, fault tolerance, privacy, scalability
  - From the sender end to the receiver end
  - Considering the whole stack of systems components
    - From the highest application end to the lowest hardware end

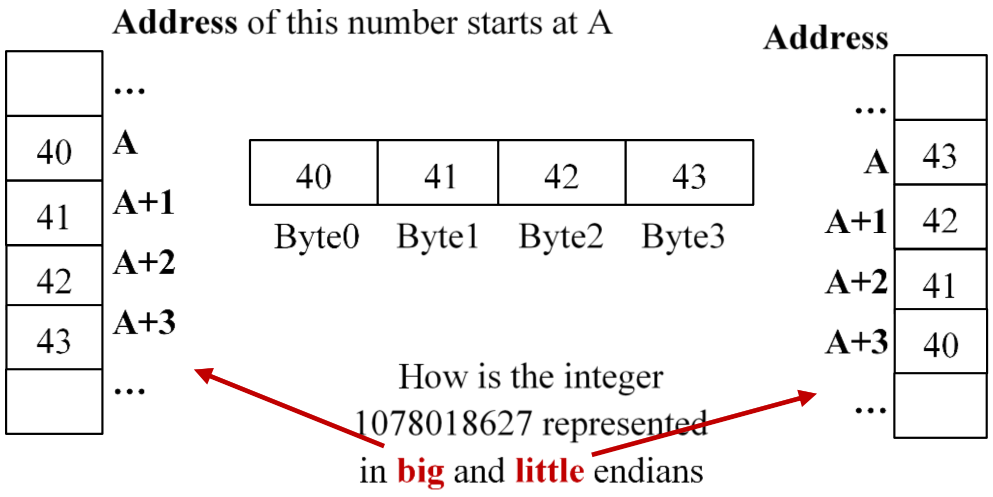



Zhang
Smith
1. Write M
4. Read M
Internet
WeChat Datacenter
M1
M2
M3
2. M is stored as three replicated copies
3. WeChat sends notification of new message to Smith

# A necessary but boring detail
# big endian vs. little endian

- How to order bytes of a multi-byte number?
- Danny Cohen (USC, 1980): “Agreement upon an order is more important than the order agreed upon.”
  - Little endian: least significant byte, i.e., 0x43 of 0x40414243, in the smallest address A
  - Big endian: most significant byte, i.e., 0x40 of 0x40414243, in the smallest address A
- State of the art
  - Big endian is used in TCP/IP networks, MIPS processors
  - Little endian is used in x86 processors, ARM processors, RISC-V processors

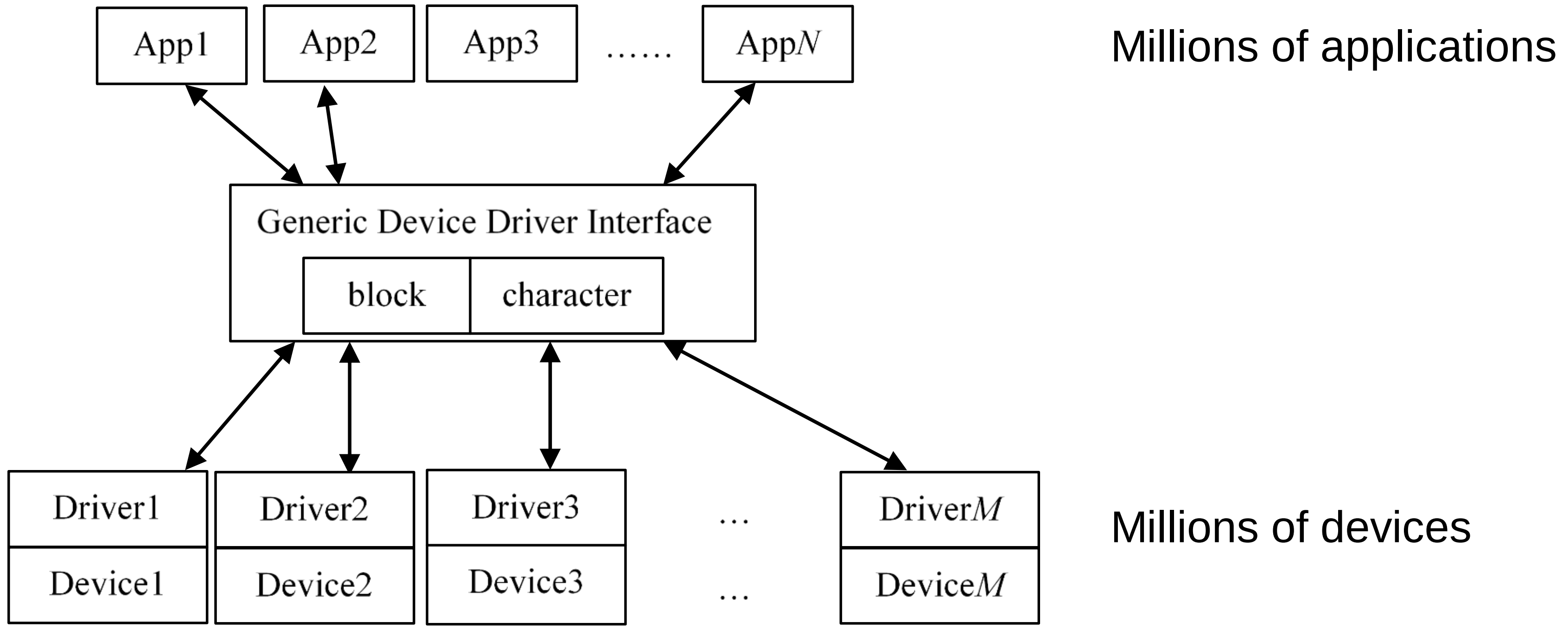
A 32-bit integer $1078018627_{10}$,
or 0x40414243 in hex,
consists of four bytes:
Byte0 is 01000000=0x40,
Byte1 is 01000001=0x41,
Byte2 is 01000010=0x42,
Byte3 is 01000011=0x43.
In what order to place 0x40, 0x41, 0x42, 0x43 in memory cells A, A+1, A+2, A+3?



Address of this number starts at A
Address
...
40
A
41
A+1
42
A+2
43
A+3
...
40
41
42
43
Byte0
Byte1
Byte2
Byte3
...
A
43
A+1
42
A+2
41
A+3
40
...
How is the integer 1078018627 represented in big and little endians

# Clever abstraction: device driver

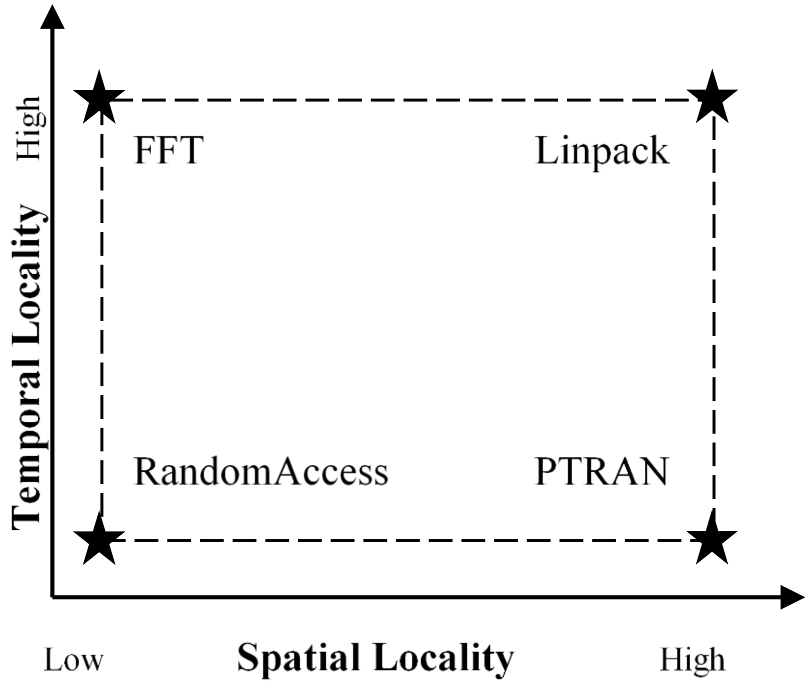
- Worst case: each device matches millions of applications
- With device driver: each device matches one interface
- Innovation: device driver
  - Only need to write one device driver for each device
  - Every device is either a block device (disk) or character device (keyboard)



# Clever abstraction: smartly designed benchmarks

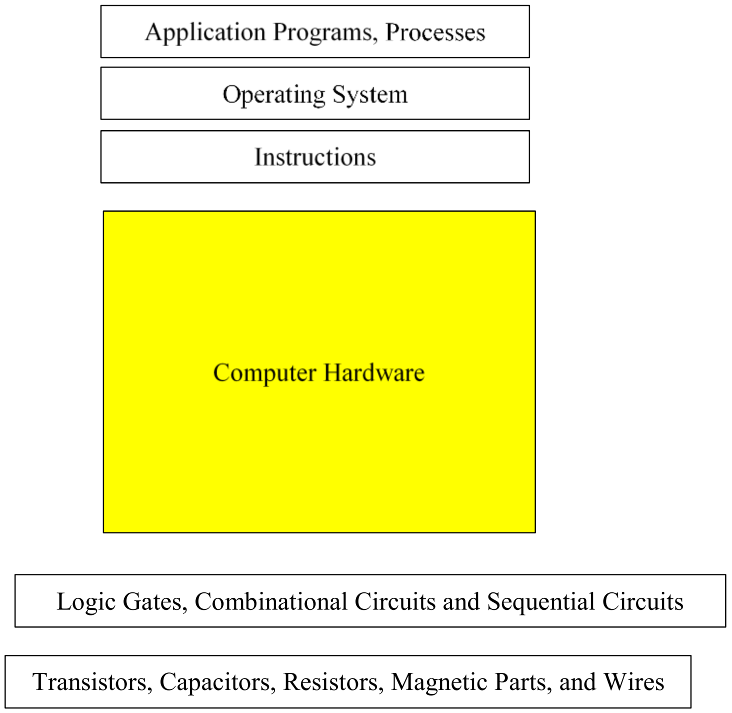
- How to design a small set of benchmark programs to measure supercomputer performance?
- Make sure the benchmarks cover the four extreme points
  - Low-low, low-high, high-low, high-high
  - Other applications are in the area enclosed by the extreme points
- A lot can be learned by measuring extreme points

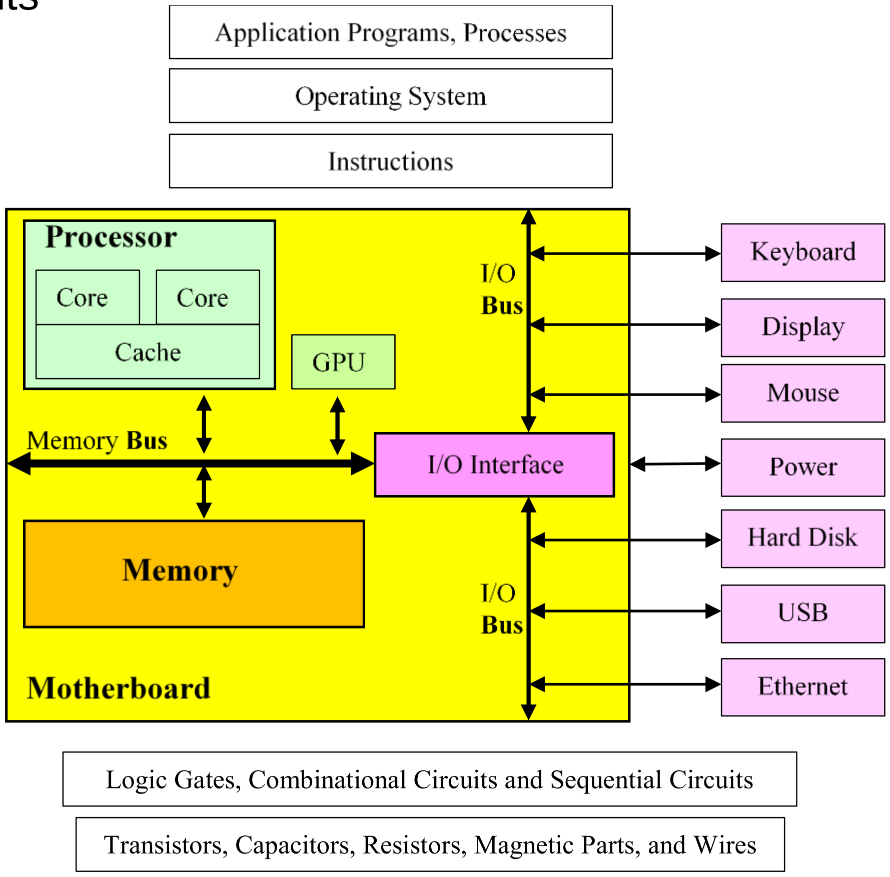
Computer performance is critically influenced by locality.
Temporal locality: data and instructions currently used tend to be used again in near future.
Spatial locality: locations nearby a reference item will also likely be referenced.



## 2.2 Being systematic

- The technical stack approach
  - Use one set of layers of abstractions to support many applications
    - Instead of one stack for an application, in an *ad hoc* way
  - Upper layer provides higher abstraction than lower layers
    - Processor is more abstract than circuits


Application Programs, Processes
Operating System
Instructions
Computer Hardware
Logic Gates, Combinational Circuits and Sequential Circuits
Transistors, Capacitors, Resistors, Magnetic Parts, and Wires



## 1-minute quiz

Which of the following list correctly orders abstractions from high-level to low-level?

(a) Computer, transistor, logic gate, processor

(b) Computer, logic gate, processor, transistor

(c) Computer, processor, logic gate, transistor

(d) Transistor, processor, logic gate, computer

Which of the following list orders abstractions from high-level to low-level?

(a) Computer, transistor, combinational circuit, sequential circuit

(b) Computer, sequential circuit, combinational circuit, transistor

(c) Combinational circuit, computer, transistor, sequential circuit

(d) Transistor, sequential circuit, combinational circuit, computer

## 1-minute quiz

Which of the following list correctly orders abstractions from high-level to low-level?

(a) Computer, transistor, logic gate, processor
(b) Computer, logic gate, processor, transistor
(c) Computer, processor, logic gate, transistor
(d) Transistor, processor, logic gate, computer

Which of the following list orders abstractions from high-level to low-level?

(a) Computer, transistor, combinational circuit, sequential circuit
(b) Computer, sequential circuit, combinational circuit, transistor
(c) Combinational circuit, computer, transistor, sequential circuit
(d) Transistor, sequential circuit, combinational circuit, computer

## Technical stack method is widely used

- Software stack
  - Application software
  - Middleware
  - Operating system
  - Firmware
- Cloud computing stack
  - BaaS, Business as a Service
  - SaaS, Software as a Service
  - PaaS, Platform as a Service
  - IaaS, Infrastructure as a Service
- Web over Internet protocol stack
  - Application layer
  - HTTP layer
  - TCP/UDP layer
  - IP layer
  - Data link layer
  - Physical layer

## Another systematic method: cycle

- Computational processes are executed as a sequence of different types of cycles
  - Program cycle
  - Instruction cycle
  - Machine cycle
  - Clock cycle
- Cycles are layered to form a stack
  - A program cycle consists of a number of instruction cycles
  - A instruction cycle consists of a number of machine cycles
  - A machine cycle consists of a number of clock cycles

- Will discuss in detail in Seamless Transition

## 2.3 Coping with complexity

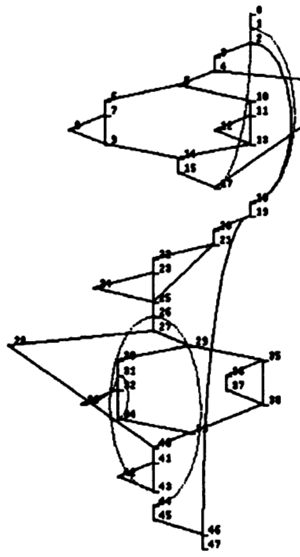
- Systems complexity refers to the degree of how complex a system is when we try to design, build, or understand it.
  - It is a cognitive complexity as well as a physical complexity
  - There is no general definition of systems complexity
  - But, sub-areas of computer science has measurements on systems complexity
    - In software engineering, we can measure the complexity of a software system
      - LoC (lines of code)
      - Cyclomatic complexity (~number of conditionals, or decision statements, in the software)
- Systems complexity is related to, but different from algorithmic complexity such as
  - O(NlogN) time complexity
  - O(N) space complexity

```
sum = sum + int(name[0])
sum = sum + int(name[1])
sum = sum + int(name[2])
sum = sum + int(name[3])
sum = sum + int(name[4])
sum = sum + int(name[5])
sum = sum + int(name[6])
sum = sum + int(name[7])
sum = sum + int(name[8])
sum = sum + int(name[9])
sum = sum + int(name[10])
```

A **straight line program**
LoC=11, CC=1

```
for i := 0; i < 11; i++ {
    sum = sum + int(name[i])
}
```

A **loop**: LoC=3; CC=2



CC=17

Watson, A. H., Wallace, D. R., & McCabe, T. J. (1996). Structured testing: A testing methodology using the cyclomatic complexity metric. US National Institute of Standards and Technology.

## Systems complexity factors

- Scale (size)
  - Number of components of a system
  - N*C, where N = LoC = 11, C is the number of components in a line, C~5
  - What if N=1024? That is, the last statement becomes
    - sum = sum + int(name[1023])
- Heterogeneity
  - Types of components
  - Homogeneous, all components are the same except for the index value
- Organization (structure)
  - How the components organized
  - Very simple: a straight line
- Variability (variation)
  - Times, rate, scale, and nature of change
  - No variation

```
sum = sum + int(name[0])
sum = sum + int(name[1])
sum = sum + int(name[2])
sum = sum + int(name[3])
sum = sum + int(name[4])
sum = sum + int(name[5])
sum = sum + int(name[6])
sum = sum + int(name[7])
sum = sum + int(name[8])
sum = sum + int(name[9])
sum = sum + int(name[10])
```

A **straight line program**
LoC=11, CC=1

Overall, this system has low complexity, except that the program size is proportional to input size.

In general, this is a bad design, except for small input sizes.

# Systems complexity factors

- Scale (size)
  - Number of components of a system
  - A constant (~10) in a single loop construct
- Heterogeneity
  - Types of components
  - Highly homogeneous
- Organization (structure)
  - How the components organized
  - Well structured loop
- Variability (variation)
  - Time, rate, scale, and nature of change
  - Low variability with regular and small variations
    - Time: changes happen at two time moments: initially (i:=0) and at each iteration (i++)
    - Rate: once per iteration
    - Scale: constant (increment by one)
    - Nature: change the index of array name

```
for i := 0; i < 11; i++ {
      sum = sum + int(name[i])
}
```

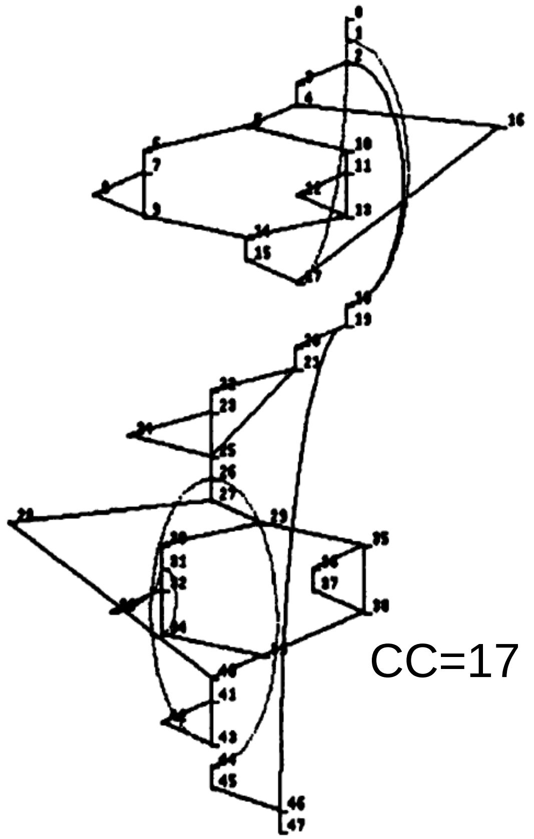
A **loop**: LoC=3; CC=2

This system has low complexity. The program size is constant, not proportional to input size.

Con: magic number 11

# Systems complexity factors

- Scale (size)
  - Number of components of a system
  - 50 code blocks
- Heterogeneity
  - Types of components
  - Very heterogeneous, dozens of types
- Organization (structure)
  - Badly structured
    - Too many decision points; CC=17 too high
    - McCabe's suggestion: CC<10
- Variability (variation)
  - Times, rate, scale, and nature of change
  - Highly variable. The control flow (program logic) jumps all over the place.



A highly complex system

Watson, A. H., Wallace, D. R., & McCabe, T. J. (1996). Structured testing: A testing methodology using the cyclomatic complexity metric. US National Institute of Standards and Technology.

## Systems complexity factors

- Scale (size)
  - Number of components of a system
- Heterogeneity
  - Types of components
- Organization (structure)
  - How the components organized
- Variability (variation)
  - Times, rate, scale, and nature of change

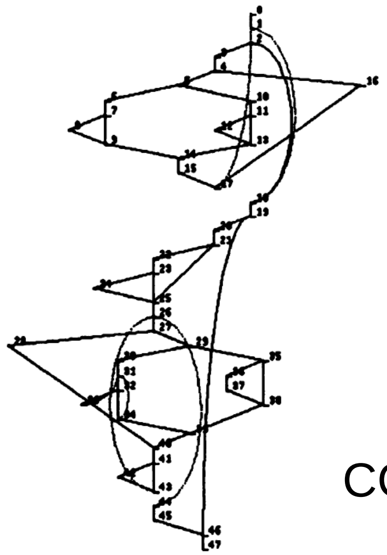
- Contrast the three cases

```
sum = sum + int(name[0])
sum = sum + int(name[1])
sum = sum + int(name[2])
sum = sum + int(name[3])
sum = sum + int(name[4])
sum = sum + int(name[5])
sum = sum + int(name[6])
sum = sum + int(name[7])
sum = sum + int(name[8])
sum = sum + int(name[9])
sum = sum + int(name[10])
```

A **straight line program**
LoC=11, CC=1

```
for i := 0; i < 11; i++ {
    sum = sum + int(name[i])
}
```

A **loop**: LoC=3; CC=2



CC=17

Watson, A. H., Wallace, D. R., & McCabe, T. J. (1996). Structured testing: A testing methodology using the cyclomatic complexity metric. US National Institute of Standards and Technology.

## 3.1 What is abstraction?

- The creative process of abstracting a high-level entity from low-level instances by focusing on the essential
- Also the outcome of the creative process of abstracting

- Examples of 4-level of abstractions
  - Data types
  - Software
  - Architecture
  - Hardware

| | | |
|---|---|---|
| Data Type | bit (1 bit), hexadecimal number (4 bits), byte (8 bits), uint8 (8-bit unsigned integer), integer (64 bits); array (n elements of the same type), slice (a descriptor pointing to an array); text file, BMP image file; hypertext and hyperlink | |
| Software | Algorithm | Smart method of information transformation, such as quicksort, hiding text in a BMP file, etc. |
| | Program | Code realizing algorithms in computer language, such as hide.go in the Text Hider project |
| | Process | Program in execution, such as the "hide" process running in a Linux environment |
| | Instruction | The smallest unit of software, directly executable by computer hardware |
| von Neumann Architecture: a computer model bridging software and hardware | | |
| Hardware | Instruction Pipeline | The basic hardware mechanism to automatically execute any instruction |
| | Sequential Circuit | More precisely, only consider Synchronous Sequential Circuit comprised of combinational circuits and state circuits and driven by a clock signal; equivalent to the automata concept |
| | Combinational Circuit | Aka Boolean circuit, realizing a Boolean function |

## 3.2 The COG properties

- A computing abstraction is **C**onstrained, **O**bjective, and **G**eneralizable
  - Constrained: a high-level concept constrained by hiding details
    - Focuses on the essential, while hiding (or ignoring) details
    - The ability to hide and ignore, namely, to constrain, is why abstraction can cope with complexity
  - Objective: a named, objective entity, no vagueness or ambiguity
    - Syntactically and semantically precisely defined concept
    - Objectivity enables bit-accurate and automatically executable abstraction
  - Generalizable: to future instances and scenarios
    - Able to handle existing instances already seen, as well as unseen instances and unexpected scenarios
    - Generalization is why we can use one set of abstractions to solve all present and future problems, instead of treating each problem instance individually

# Example: Unicode

- Process of abstracting: Encoding the world's writing systems
- Outcome of abstracting: Unicode (with COG properties)
  - Constrained: focus on the essential; ignore details such as a symbol’s meanings
    - U+5174 represents character 兴; ignore meanings such as happy, agitated, a name
  - Objective: a named, objective entity, no vagueness or ambiguity
    - Syntactically and semantically precisely defined by Unicode standards
    - Enables bit-accurate and automatic execution in systems worldwide
  - Generalizable: to unseen instances or unexpected scenarios
    - Able to handle existing instances already seen, as well as unseen instances and unexpected scenarios
    - Generalization is why we can use one set of abstractions to solve all present and future problems, instead of treating each problem instance individually

| Symbol | Description | Unicode |
|---|---|---|
| T | English capital letter T | U+0054 |
| Ω | Greek letter Omega | U+03A9 |
| € | The Euro sign | U+20AC |
| 志 | A Chinese character | U+5FD7 |
| 𐍈 | A Gothic letter | U+10348 |

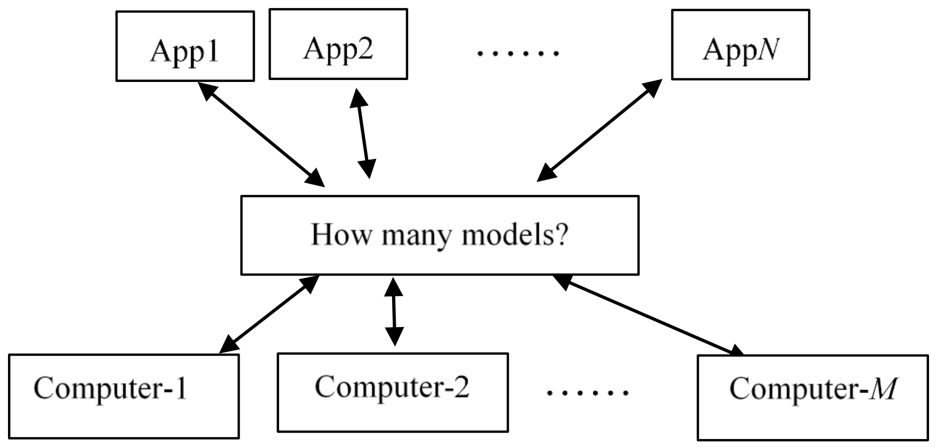
## Example: World Wide Web

- Abstracting: Accessing the world’s information resources by hypertext
- Outcome: Tim Berners-Lee’s WWW abstraction
- Constrained
  - Focusing on accessing Web resources via URL, HTTP, and HTML; ignore details of storing, processing, etc.
- Objective
  - Precise specification by IETF & W3C standards
- Generalizable
  - Originally, resource = document
  - Generalized to multimedia, programs, data, things

https://www.w3.org/comm/assets/logos/Web@30_logo/Logo_web/PNG/Logo_Logo_horizontal.png

## 3.3 One abstraction for thousands of scenarios 以一耦万

- There are millions of computer applications and thousand types of computers worldwide
  - Q: How many models does a user see when writing an application, such as when writing a Go program fib.dp.go to compute Fibonacci numbers?
  - 1? 10? 100? 1000?
  - A related question
    - Our class has hundreds of students, each using having a laptop computer. These hundreds of computers come in different types, with different processors, memory configurations, and operating systems. When writing their fib.dp.go programs, do the students see the same model or different models of computers?

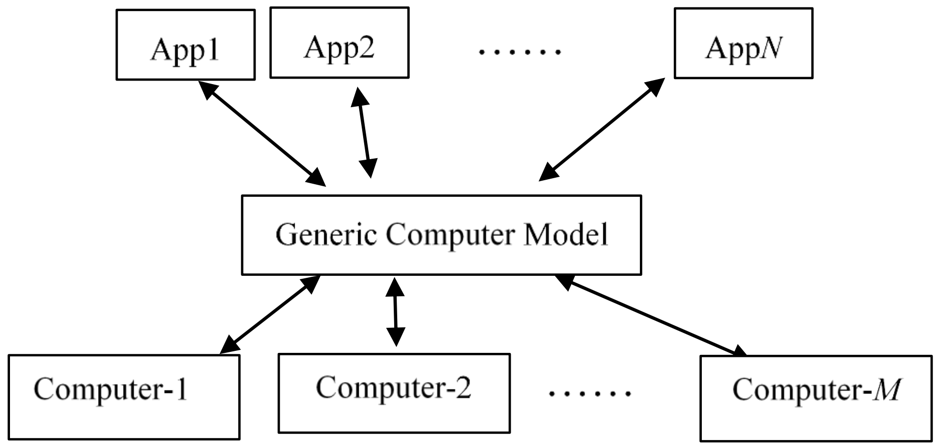


Millions of applications

Thousand types of computers

## One abstraction for thousands of scenarios 以一耦万

- There are millions of computer applications and thousand types of computers worldwide
  - Q: How many models does a user see when writing an application, such as when writing a Go program fib.dp.go to compute Fibonacci numbers?
  - A: One. An application sees all these computers as having the same model. We can ignore the detailed hardware and software particularities.



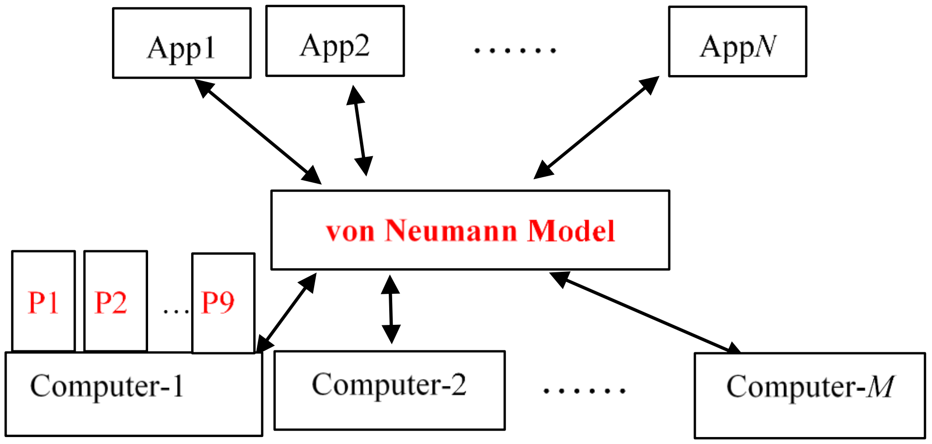
Millions of applications

One abstraction for modeling

Thousand types of computers

# One abstraction for thousands of scenarios 以一耦万

- There are millions of computer applications and thousand types of computers worldwide
  - Operating system uses one abstraction to manage programs in execution, no matter what the programs are
  - Q: What is this abstraction called?
  - A:



Millions of applications

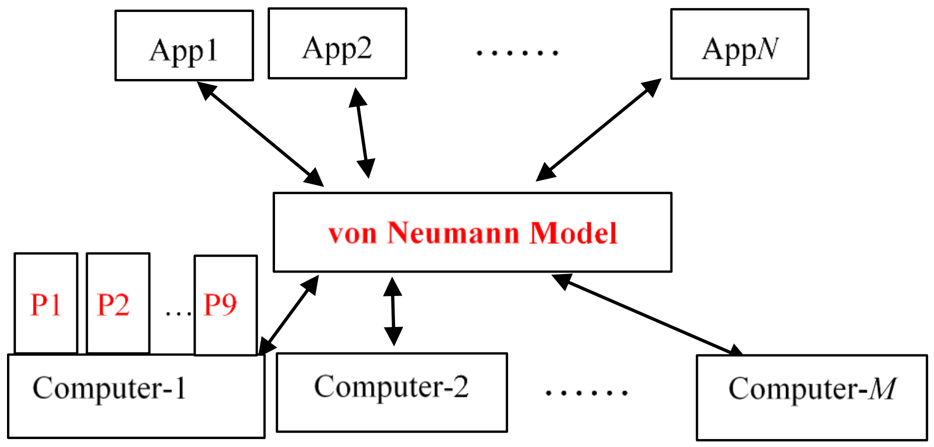
One abstraction for modeling

Thousand types of computers

P1, …, P9 are 9 processes executing on Computer-1
P1 is program fib.go in execution, P2 is program fib.dp.go in execution, P9 is VS Code executing, etc.

## One abstraction for thousands of scenarios 以一耦万

- There are millions of computer applications and thousand types of computers worldwide
  - Operating system uses one abstraction to manage programs in execution, no matter what the programs are
  - Q: What is this abstraction called?
  - A: Process



P1, …, P9 are 9 processes executing on Computer-1
P1 is program fib.go in execution, P2 is program fib.dp.go in execution, P9 is VS Code executing, etc.

Millions of applications

One abstraction for modeling
One abstraction for managing

Thousand types of computers
Billions of computers