计算机导论

中国科学技术大学
University of Science and Technology of China

课程介绍

课程信息

- 计算机导论（20学时/1学分）
- 时间地点：周二 6,7节，3A312

- 教师：张燕咏（yanyongz@ustc.edu.cn）
- 助教（4人）
  - 冷佳明（lengjm@mail.ustc.edu.cn）
  - 孟成真（czmeng@mail.ustc.edu.cn）
  - 肖宇煊（xiaoyx@mail.ustc.edu.cn）
  - 王凯（wkzcml@mail.ustc.edu.cn）

课程信息

- 考核方式：随堂小测验+课程报告
- 评分制：五分制
- 是否计算绩点：**是**

- 参考教材（不是必需）
  - “计算机科学导论”， 徐志伟，孙晓明，清华大学出版社
  - “Foundations of Computer Science”, C Edition, Alfred V. Aho, Jeffrey D. Ullman, W. H. Freeman, 1994, ISBN 0716782847

课程信息

- 课程章节内容（以往）
  - 计算机科学概论：4学时
  - 计算模型与逻辑思维：4学时
  - 算法思维：4学时
  - 网络思维：4学时
  - 系统思维：4学时
- 变化
  - 增加示例和演示
  - 补充课程中没有的内容
  - 学习计算机专业的本地经验

内容纲要

- 计算机是一个交叉学科，仍在不断发展
- 从编程语言的发展看计算机的发展
- 中国科大计算机专业本科教学体系及特色

交叉学科来源

恩格斯说：“社会一旦有技术上的需要，这种需要就会比十所大学更能把科学推向前进。”

——习近平在科学家座谈会上的讲话

（2020年9月11日）

科学上的突破和创新越来越依赖于交叉学科。据统计，从诺贝尔奖结果来看，最近25年交叉研究获得诺贝尔奖的比例已接近一半（49.07%）。

我国将新增交叉学科作为新的学科门类，交叉学科将成为我国第14个学科门类。

交叉学科来源

动力：需求导向 和 问题导向

过程：学科交叉 —> 学科融合 —> 交叉学科

科学经历了综合、分化、再综合的过程。

学科交叉是“学科际”或“跨学科”研究活动，其结果导致的知识体系构成了交叉科学。学科交叉导致众多交叉科学前沿。

交叉科学是综合性、跨学科的产物，科学知识体系具有整体化的本质特征，有利于解决人类面临的重大复杂科学问题、社会问题和全球性问题。

交叉学科类型

- 边缘学科，两门学科发生交叉而形成的学科
- 综合学科，即通过研究一些综合性问题而发展起来的学科
- 横断学科，即研究各种对象或专门学科中的某些共同性问题而发展起来的学科

生物化学  + 

生态经济学  + 

空间科学 

围绕宇宙演化、生命起源、物质结构、暗物质、宇宙航行等综合性科学问题

人工智能

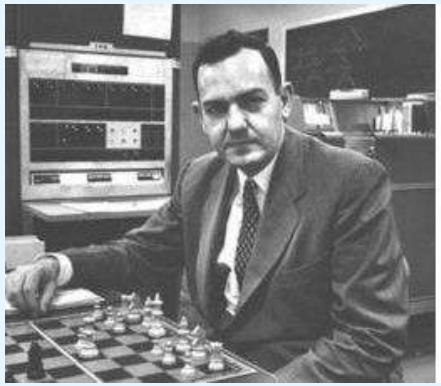
神经科学 | 认知科学 | 计算机科学 | 控制科学 | 脑科学 | 语言学

研究共性理论与方法

计算机科学也是交叉学科

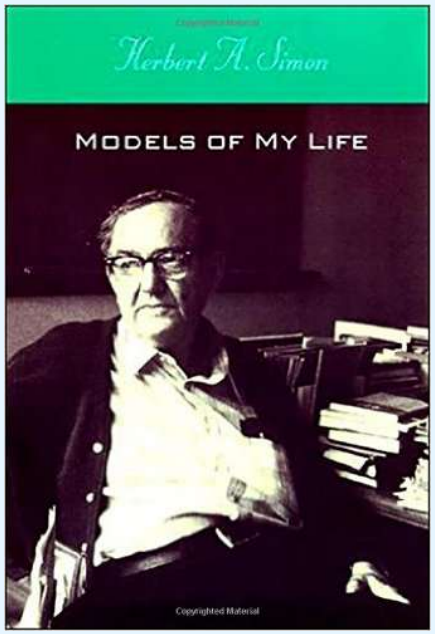
- 以中科大计算机专业为例。1958年创校时成立应用数学和计算技术系，系主任是华罗庚。
  - 1952年华罗庚在中科院数学所创建了我国第一个计算机科研小组(包括闵乃大、夏培肃和王传英)，开展电子计算机研究工作。
  - 1946年华罗庚在美国普林斯顿高等研究院访问时，参观过冯·诺依曼的通用电子计算机实验室，与他一起讨论过计算技术的研究问题。
- “华夏” 英才班名字取自计算机专业的两位创始人：华罗庚（数学）和夏培肃（电机）。

开展交叉研究的典型人物



- Herbert A. Simon (1916 — 2001)
  - 中文名：司马贺
  - 1937年获美国芝加哥大学学士学位
  - 1943年获美国芝加哥大学博士学位 (Political Science)
  - 1946年-1949年，美国伊利诺伊理工学院(IIT) (Political Science)
  - 1949年-2001年，美国卡内基 - 梅隆大学(CMU)工业管理系、计算机系和心理系
- 学术荣誉
  - 美国科学院院士(1967)、中国科学院外籍院士(1994)、ACM图灵奖(1975)、诺贝尔经济学奖(1978)
- 主要研究领域
  - 政治学/管理学/社会学/经济学/计算机/心理学/认知科学/人工智能
- 主要研究问题：
  - “社会环境中人类行为的原子现象” （人如何进行思考和决策？）

- 我生活的种种模式:赫尔伯特·A·西蒙自传/(美)赫伯特·A·西蒙著 曹南燕，秦裕林译，1998年
  - Models of My Life，1996
- 穿越歧路花园:司马贺传，2009年
  - Herbert A. Simon: The Bounds of Reason in Modern America, 2005



从专业目录看计算机专业的发展

- 1998年版教育部《普通高等学校专业目录》
  - 计算机科学与技术专业是电气信息类的一个专业，代码是080605
  - 计算机软件（080619W）、软件工程（080611W）、网络工程（080613W）、信息安全（071205W）列为电子信息类目录外专业
  - 智能科学与技术（080627S）列为在部分高校试点的电子信息类目录外专业
- 2012年版教育部《普通高等学校专业目录》
  - 设立了计算机类专业，代号为0809。包括计算机科学与技术(080901)、软件工程(080902)、网络工程(080903)、信息安全(080904)、物联网工程(080905)、数字媒体技术(080906)6个基本专业，智能科学与技术(080907T)、空间信息与数字技术(080908T)、电子与计算机工程(080909T) 3个特设专业。
  - 交叉：数字媒体技术、空间信息与数字技术等

从专业目录看计算机专业的发展

- 计算机类(0809)近年来的变化
  - 2016年增加两个特设专业：数据科学与大数据技术(080910T)、网络空间安全(080911TK)。
  - 2017年增加三个特设专业：新媒体技术(080912T)、电影制作(080913T)和保密技术(080914TK)。
  - 2019年增加了服务科学与工程(080915T)、虚拟现实技术(080916T)和区块链工程(080917T)。

- 2019年，在电子信息类(0807)下设人工智能专业(080717T)。

从专业目录看与其他学科的交叉

- 数学类
  - 070102信息与计算科学，070104T数据计算及应用
- 地理科学类/地质学类/测绘类
  - 070504地理信息科学，070903T地球信息科学与技术，081205T地理空间信息工程
- 生物科学类
  - 071003生物信息学
- 管理学类
  - 120102 信息管理与信息系统、120108T大数据管理与应用、120110T计算金融、120503 信息资源管理

国内人工智能学科设置情况

- 2017.09.10 中国科学院大学成立人工智能学院
- 2017.11.02 西安电子科技大学成立人工智能学院
- 2018.01.13 湖南工业大学成立人工智能学院
- 2018.02.07 重庆邮电大学成立人工智能学院
- 2018.03.06 南京大学成立人工智能学院
- 2018.05.16 天津大学、南开大学成立人工智能学院
- 2018.05.26 吉林大学成立人工智能学院
- 2018.06.28 清华大学成立人工智能研究院
- 2018.09.27 电子科技大学成立人工智能研究院
- …
- 2019.01.14 东南大学成立人工智能学院
- 2019.01.21 西安交通大学成立人工智能学院
- 2019.01.22 中国人民大学成立高瓴人工智能学院
- 2019.01.26 华中科技大学成立人工智能与自动化学院
- 2019.04.26 北京大学成立人工智能研究院
- 2019.06.01 北京师范大学成立人工智能学院
- 2019.09.30 大连理工大学成立人工智能学院
- 2019.11.16 华北电力大学成立人工智能学院
- …

人工智能交叉学科性质

- 人才培养：多学科交叉，产学研结合

一级学科：
- 计算机科学与技术
- 控制科学与工程
- 信息与通信工程
- 生物医学工程
- 电子科学与技术
- 机械工程

本科专业：
- 电子信息类：人工智能(080717T)
- 计算机类： 智能科学与技术(080907T)

- 国内高校人工智能学科设置：依托学科多样

控制科学与工程

上海交通大学
西安交通大学
西安电子科技大学
南开大学
北京理工大学
华南理工大学
南京邮电大学

计算机科学与技术

浙江大学
南京大学
哈尔滨工业大学
电子科技大学
四川大学
中山大学
湖南大学

信息安全

武汉大学

软件工程

北京航空航天大学

信息通信

华中科技大学
北京师范大学

机械工程

北京邮电大学

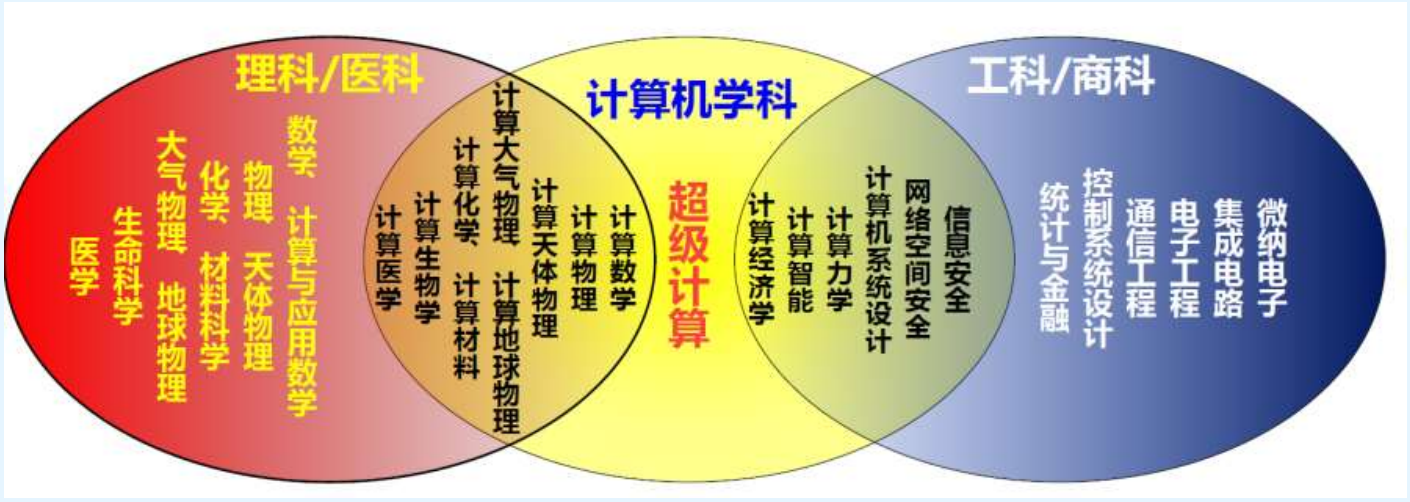
与传统计算机人才培养的异同

- 传统计算机人才培养：硬件系统、算法理论、软件工程等3大基本能力 加 应用知识

- 差异
  - 学科交叉：跨学科交叉是人工智能的典型标志，也是对人才培养的挑战
  - 应用导向：“计算机应用”从以“计算机”为中心到以“应用”为中心

- 相同
  - 强调数理基础
  - 重视实践能力

计算机学科与理工医商的交叉融合

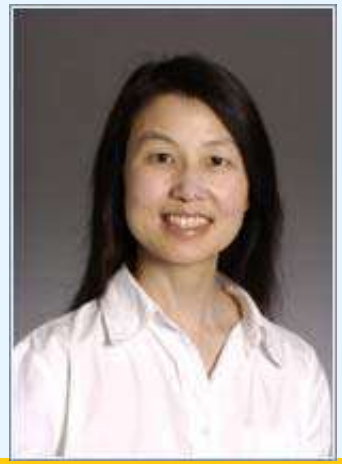


科学与科学思维

“科学思维不仅是一切科学研究和技术发展的起点，而且始终贯穿于科学研究和技术发展的全过程，是创新的灵魂”。

- **科学思维**
  - 理论思维(公理、规则、结论)：推理->数学
  - 实验思维(重现、自洽、预见)：实验-> 物理
  - 计算思维(设计、模拟、仿真、挖掘)->使自动：计算机

计算思维的提出



- Edsger Dijkstra
  - 我们所使用的工具影响着我们的思维方式和思维习惯,从而也将深刻的影响着我们的思维能力.
- Computational thinking will be a fundamental skill used by everyone in the world by the middle of the 21st Century.

计算思维的提出

Computational Thinking

It represents a universally applicable attitude and skill set everyone, not just computer scientists, would be eager to learn and use.

Computational thinking builds on the power and limits of computing processes, whether they are executed by a human or by a machine. Computational methods and models give us the courage to solve problems and design systems that no one of us would be capable of tackling alone. Computational thinking confronts the riddle of machine intelligence: What can humans do better than computers? and What can computers do better than humans? Most fundamentally it addresses the question: What is computable? Today, we know only parts of the answers to such questions.

Computational thinking is a fundamental skill for everyone, not just for computer scientists. To reading, writing, and arithmetic, we should add computational thinking to every child's analytical ability. Just as the printing press facilitated the spread of the three Rs, what is appropriately incestuous about this vision is that computing and computers facilitate the spread of computational thinking.

Computational thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science. Computational thinking includes a range of mental tools that reflect the breadth of the field of computer science.

Having to solve a particular problem, we might ask: How difficult is it to solve? and What's the best way to solve it? Computer science rests on solid theoretical underpinnings to answer such questions precisely. Stating the difficulty of a problem accounts for the underlying power of the machine—the computing device that will run the solution. We must consider the machine's instruction set, its resource constraints, and its operating environment.

In solving a problem efficiently, we might further ask whether an approximate solution is good enough, whether we can use randomization to our

- 计算思维（**Computational Thinking**）：2006年3月，周以真（Jeannette M. Wing）在《Communications of the ACM》上给出。

计算思维概念

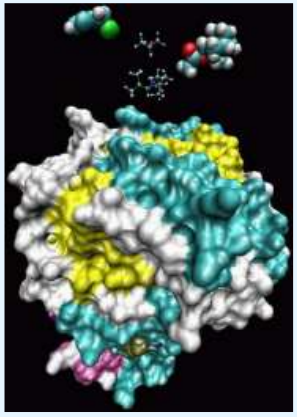
- 计算思维是运用计算机科学的基础概念去**求解问题、设计系统和理解人类行为**
- 其本质是抽象和自动化
  - 计算思维包括数学思维和工程思维，其中抽象能力必不可少（如抽象算法、模型、语言、协议等）；
  - 自动化是计算机的灵魂（如系统、程序、编译等）；
  - 如同所有人都具备“读、写、算”（简称3R）能力一样，都必须具备的思维能力。

中国2050年信息科技发展路线图

由李国杰院士任组长的中国科学院信息领域战略研究组撰写的《中国至2050年信息科技发展路线图》中对“计算思维”给予了足够的重视；**计算思维的培育是克服“狭义工具论”的有效途径，是解决其他信息科技难题的基础。**

计算思维——抽象与模拟

- 实验和理论思维无法解决的问题
- 大量复杂问题求解、宏大系统建立、大型工程组织都可通过计算模拟（融合工程思维）
- 核爆炸、蛋白质生成、大型飞机、舰艇设计…



计算思维——多学科交叉

- 计算思维对各个学科的人才培养都有重要意义

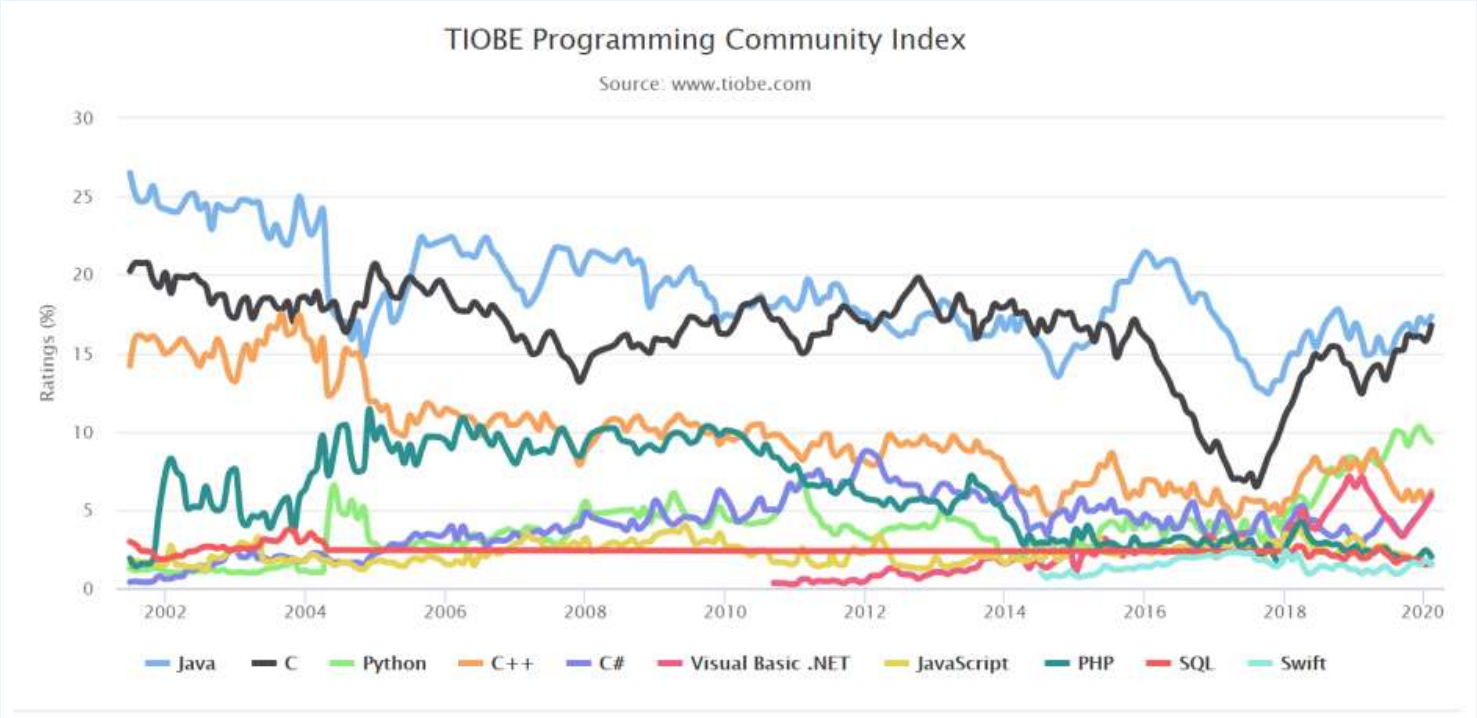
- 1998年的诺贝尔化学奖被授予一个化学领域研究工具Gaussian的开发者：波普(John Pople)
- 作为把计算机应用于化学研究的主要科学家，其建立了可用于化学各个分支的一整套量子化学方法，把量子化学发展成一种工具，并已为一般化学家所使用，以便在计算机里模拟分子赋予它们异种特性的方法，研究分子间如何相互发生作用并如何随环境而改变，从而使化学迈向用实验和理论共同探索分子体系各种性质的新时代

计算思维——多学科交叉

- 2013诺贝尔奖化学奖得主为：马丁·卡普拉斯(Martin Karplus)，迈克尔·莱维特(Michael Levitt)和亚利耶·瓦谢尔(Arieh Warshel)，以奖励他们在“发展复杂化学体系多尺度模型”方面所做的贡献。
- 当科学家在模拟分子反应的过程时，他们会在必要时借助计算机的力量。反应系统核心的计算基于量子物理学，而在远离反应核心区域的地方，模型计算则基于经典物理学；在最外的几层，原子和分子甚至混合在一起，形成同质的物体。通过这些理论简化，我们可以对大型的化学系统进行模拟计算。

计算思维——多学科交叉

- AlphaFold is an artificial intelligence (AI) program developed by Alphabet's/Google's DeepMind which performs predictions of protein structure.The program is designed as a deep learning system.



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

The index can be used to check whether your programming skills are still up to date or to make a strategic decision about what programming language should be adopted when starting to build a new software system.

Feb 2020	Feb 2019	Change	Programming Language	Ratings	Change
1	1		Java	17.358%	+1.48%
2	2		C	16.766%	+4.34%
3	3		Python	9.345%	+1.77%
4	4		C++	6.164%	-1.28%
5	7	↑	C#	5.927%	+3.08%
6	5	↓	Visual Basic .NET	5.862%	-1.23%
7	6	↓	JavaScript	2.060%	-0.79%
8	8		PHP	2.018%	-0.25%
9	9		SQL	1.526%	-0.37%
10	20	⇈	Swift	1.460%	+0.54%
11	18	⇈	Go	1.131%	+0.17%
12	11	↓	Assembly language	1.111%	-0.27%
13	15	↑	R	1.005%	-0.04%
14	23	⇈	D	0.917%	+0.28%
15	16	↑	Ruby	0.844%	-0.19%
16	12	⇊	MATLAB	0.794%	-0.40%
17	21	⇈	PL/SQL	0.764%	-0.05%
18	14	⇊	Delphi/Object Pascal	0.748%	-0.32%
19	13	⇊	Perl	0.697%	-0.40%
20	10	⇊	Objective-C	0.688%	-0.76%

Position	Programming Language	Ratings
21	SAS	0.656%
22	Visual Basic	0.603%
23	Dart	0.553%
24	Scratch	0.532%
25	Scala	0.450%
26	Groovy	0.413%
27	Transact-SQL	0.379%
28	F#	0.352%
29	Rust	0.346%
30	COBOL	0.339%
31	ABAP	0.309%
32	Lisp	0.294%
33	Kotlin	0.267%
34	Logo	0.253%
35	RPG	0.240%
36	Lua	0.235%
37	Fortran	0.229%
38	PowerShell	0.209%
39	Ada	0.206%
40	LabVIEW	0.176%
41	Erlang	0.174%
42	Julia	0.170%
43	ML	0.169%
44	Scheme	0.169%
45	Haskell	0.160%
46	TypeScript	0.155%
47	OpenEdge ABL	0.150%
48	LiveCode	0.146%
49	PostScript	0.144%
50	ActionScript	0.142%

#51 to #100

(Visual) FoxPro, Apex, ATLAS, Awk, Bash, bc, Bourne shell, C shell, cg, CL (OS/400), Clojure, Common Lisp, Crystal, cT, Curl, Elixir, Emacs Lisp, Forth, Hack, Icon, Inform, Io, J, JScript, Korn shell, Ladder Logic, Maple, MEL, Mercury, MQL4, NATURAL, OpenCL, Oz, PL/I, Programming Without Coding Technology, Prolog, Pure Data, Q, Raku, Red, Ring, S, Smalltalk, SPARK, Stata, Tcl, VBScript, Verilog, VHDL, WebAssembly

程序设计语言

- C语言
  - 发明者的母语是英语
  - 诞生于1972年

梅森质数(2^p-1)

- $2^{67}-1$是质数么？

- *In 1903 Frank Nelson Cole famously made a presentation to a meeting of the American Mathematical Society where he identified the factors of the Mersenne number . During Cole's so-called "lecture", he approached the chalkboard and in complete silence proceeded to calculate the value of M_{67}, with the result being 147,573,952,589,676,412,927. Cole then moved to the other side of the board and wrote 193,707,721 × 761,838,257,287, and worked through the tedious calculations by hand. Upon completing the multiplication and demonstrating that the result equaled M_{67}, Cole returned to his seat, not having uttered a word during the hour-long presentation. His audience greeted the presentation with a standing ovation. Cole later admitted that finding the factors had taken "three years of Sundays".*

Frank Nelson Cole

```
a = 2 ** 67 -1
print(a)
flag = 1
i = 3

while flag:
    if a % i == 0:
        print(i)
        print(a/i)
        flag = 0
    i = i + 2
```

```
147573952589676412927
193707721
761838257287.0
>>>
```

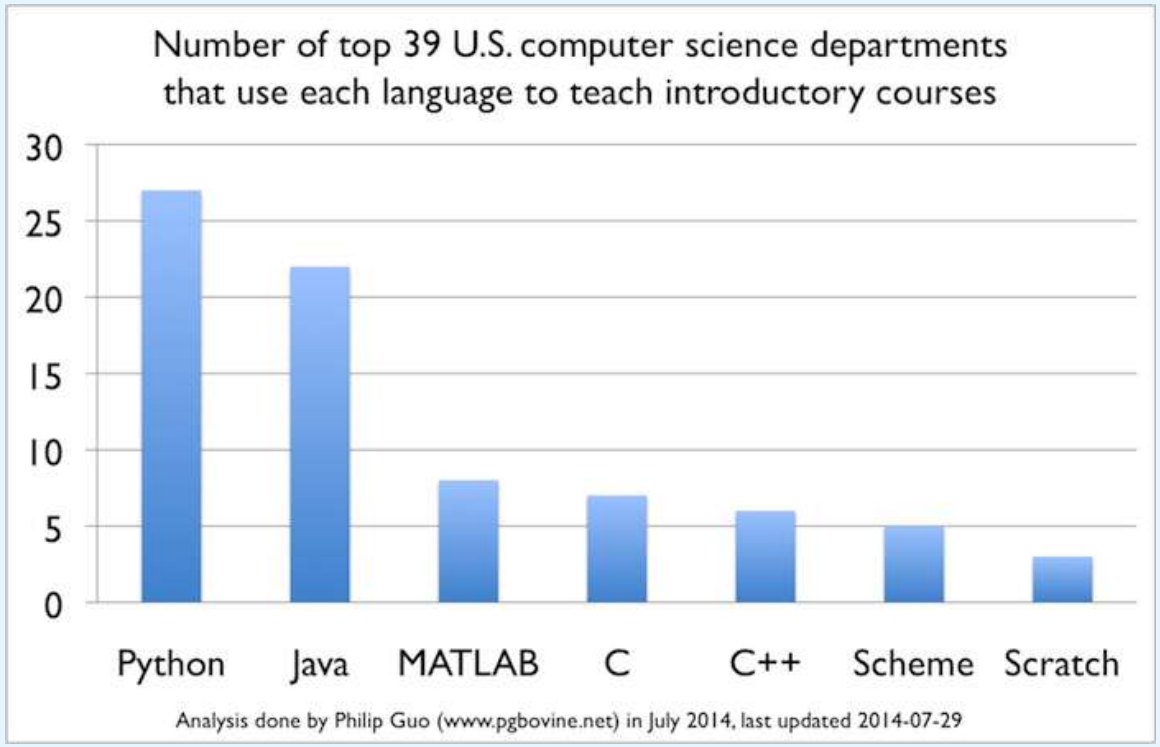
Python

- Python是一种计算机程序设计语言。是一种面向对象的动态类型语言，最初被设计用于编写自动化脚本(shell script)，随着版本的不断更新和语言新功能的添加，越来越多被用于独立的、大型项目的开发。

- 1991年至今

- https://www.python.org/

Guido van Rossum(1956年—)


Number of top 39 U.S. computer science departments that use each language to teach introductory courses
30
25
20
15
10
5
0
Python
Java
MATLAB
C
C++
Scheme
Scratch
Analysis done by Philip Guo (www.pgbovine.net) in July 2014, last updated 2014-07-29

为什么选择Python?

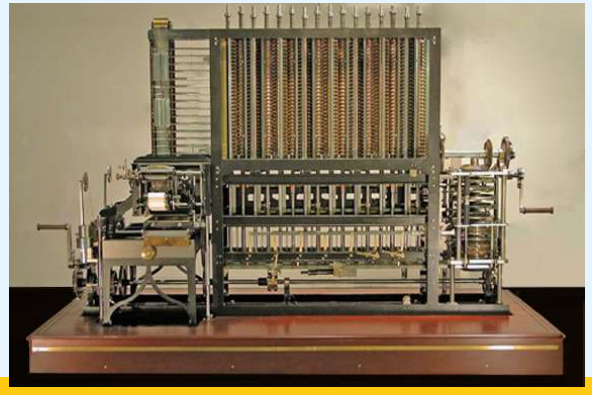
1. 简单，便于学习
2. 免费
3. 开源
4. 不是“玩具”，应用广泛
5. 跨平台（Windows，苹果，Linux等）

当代计算机科学的三个基本问题

- **巴比奇问题**：如何构造有效的计算系统?
  - Charles Babbage，19世纪剑桥大学教授
  - 提出了差分机和分析机概念，是计算机系统先驱
- **布什问题**：如何有效地使用计算系统，人-计算机-信息如何互联?
  - Vannevar Bush，20世纪麻省理工学院教授
  - 发明了微分分析机，提出了Memex个人计算机
- **图灵问题**：如何用计算机解题、最终产生智能?
  - Alan Turing，20世纪英国计算机科学家艾伦·图灵
  - 提出了图灵机与图灵测试，是计算机科学理论的奠基人与人工智能先驱

巴贝奇问题：构造计算机

- 巴贝奇：差分机 （1837）
  - 动机：编制航海、天文用表时，需要大量人工劳动
  - Ada：第一位程序员
- 第一台“图灵完全的”计算机设计
  - 包括40个十进制数字符号、一个算数逻辑单元、一个存储器、输入输出
  - 自动执行、图灵完全



“世界上第一位程序员”——数学奇女子 Ada

人民邮电出版社 已认证账号

14 人赞同了该文章

“我有一个梦想，做飞行员、科学家、数学家……”但长大后，我发现这些梦想似乎都没有实现。跟 100 多年前的巴贝奇比起来，我的梦想也就是嘴上说说罢了。

Charles Babbage
1792—1871

计算机能够求解任意可计算问题

图灵机是通用计算机

- Alonzo Church & Alan Turing:
  Church-Turing Hypothesis:

> *Any reasonable attempt to model mathematically computer algorithms and their performance is bound to end up with a model of computation and associated time cost that is equivalent to Turing machines within a polynomial.*

Alan Turing
1912-1954

图灵奖：国际著名的计算机科学奖

- Turing Awards
  - 获奖者是“大牛”
  - 很多还同时是优秀的教师

斯坦福大学
高德纳教授

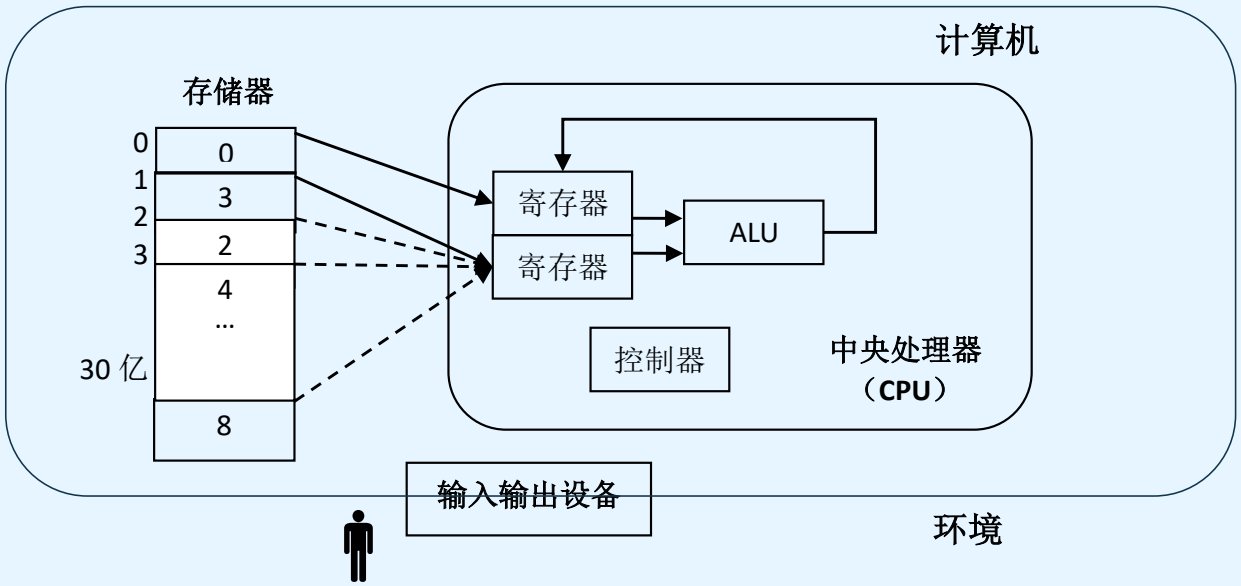
清华大学
姚期智教授

ENIAC：现代真实计算机的代表

- 1943年初，美国陆军出现弹道计算迫切需求
- 宾州大学电子计算机的概念、立项、实施、使用
  - 1941年，毛捷利与艾克特开始合作
    - 1942年，毛捷利备忘录：《高速真空管器件的计算用途》
  - 1943年4月2-9日，正式立项
  - 1943年5月31日，正式启动
  - 1944年6月，开始建造
  - 1945年11月，ENIAC研制成功
  - 随后在BRL 用了10年
  - 每周使用145小时
  - 长期是世界上最快的计算机
  - 对科学家开放
- ENIAC的主要特征
  - 数字计算机：非模拟计算机
  - 电子计算机：所有运算操作用电子速度“自动执行”
  - 足够的可靠性：18000个真空管的大系统

图片来自Charles Nietubicz博士

当代计算机基本组成



今天有三大类计算机

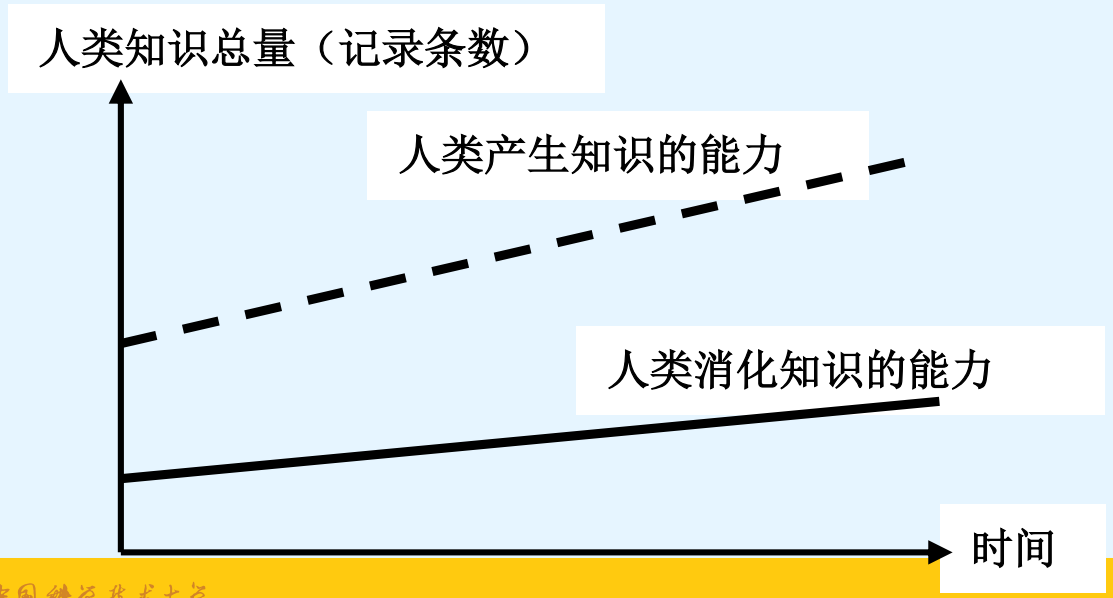
- 客户端计算机（我们最熟悉的）
- 服务端计算机（在机房里边的）
- 嵌入式计算机（我们看不见的）

- 贝尔定律：十年出一小类
  - 超级计算机（supercomputer）
  - 大型机（mainframe）
  - 小型机（minicomputer）
  - 机群（cluster）
  - 工作站（workstation）
  - 微型机（microcomputer）
  - 便携计算机（portable）
  - 个人专用终端（dedicated device）
  - 智能手机（smart phone）
  - 可穿戴计算机（wearable computer）

机群（cluster）、机柜（rack）、1U机箱（节点，node）、互联

布什问题：人-计算机-信息如何互联

- 信息互联问题
- 人机交互问题 = 使用模式问题
- 来源于科学研究中长期存在的一个根本问题
  - “**布什诘难**”或“**孟德尔-布什诘难**”



1865年，孟德尔发表了通过多年豌豆实验发现的遗传规律，后来被称为孟德尔定律。

孟德尔的论文长期无人理睬，甚至无人知道。

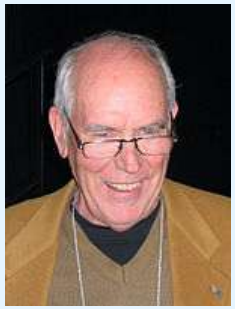
30年以后，才有懂行的人读懂了他的论文，并加以发展，建立了遗传学

布什的思路：Memex个人计算机

- 存储器，永久地存储用户一生所需的所有知识
  - 知识的基本单位是一条研究成果记录，存储器=知识库
  - 知识库的组织方式：索引，关联（超链接）
- 输入输出设备：方便地操作存储器
  - 操作：选择、显示、生产
- Memex是一台交互式个人电脑
  - 所有操作即刻完成
- 美国国防部DARPA Memex研究计划（2014）
  - 比现有搜索引擎更加先进，能够搜索出浅网和深网信息，并组合出知识。第一个目标应用是打击人口贩卖。
  - 2015年破获纽约的绑架案件

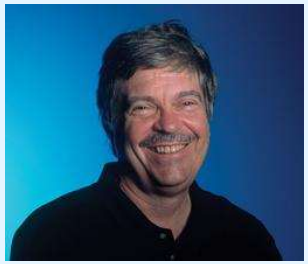
计算机的使用模式

- **使用模式 = 用户群+信息组织方式+人机交互方式**
  - 科学家、工程师（科学计算）
  - 企业（企业计算）
  - 消费者（消费者计算）
- **历史上重要的使用模式**
  - 批处理
  - 交互式计算
  - 个人计算
  - 图形用户界面
  - 多媒体计算
  - 便携式计算
  - 互联网计算
  - 移动互联网计算
  - **万物互联网计算（正在兴起）**

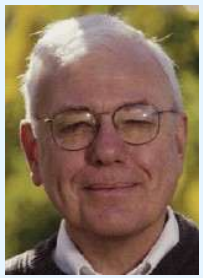


Ivan Sutherland

Butler Lampson



Alan Kay



Charles Thacker

个人计算模式的四位图灵奖得主

倪光南发明联想式汉卡

- 新使用模式需要新的技术支撑
- 1981年8月，IBM PC
  - 4.77 MHz主频、16位8088处理器，16-256KB内存
    - 难以实现简单的汉字交互式计算
  - 为个人电脑在中国的普及造成了巨大障碍
- 倪光南从1968年就开始研究汉字显示器技术
  - 1983年，LX-80汉字图型微型机
  - 1985年，第一型联想式汉卡诞生并走向市场
    - 支持20多种中文字体，基本不占PC资源
    - 联想功能：“记”字 → 自动出现“记者”、“记录”等
    - 是今天全球PC机第一品牌的联想集团的来源

文言文编程语言

- 来自CMU本科生Huang Lingdong。可编译成Javascript，也可编译成Python，亦有线上编辑器 (IDE) 可直接运行。
- GitHub地址：https://github.com/LingDong-/wenyan-lang

EDITOR

```
吾有一術。名之曰「快排」。欲行是術。必先得一列。曰「甲」。乃行是術曰。
    若「甲」之長弗大於一者。
        乃得「甲」
    也。
    吾有三列。名之曰「首」。曰「頷」。曰「尾」。

    夫「甲」之一。名之曰「甲一」。
    充「頷」以「甲一」。
    夫「甲」之其餘。名之曰「甲餘」。
```

COMPILED JAVASCRIPT

```
var KUAI4PAI2 = () => 0;
KUAI4PAI2 = function(JIA3) {
    if (JIA3.length <= 1) {
        return JIA3
    };
    var SHOU3 = [];
    var HAN4 = [];
    var WEI3 = [];
    var _ans49 = JIA3[1 - 1];
    var JIA3YI1 = _ans49;
    HAN4.push(JIA3YI1);
```

> 夫唐、虞之世，結繩而足治，屈指而足算。是時豈料百代之後，計算機械之巧，精於公輸之木鳶，善於武侯之流馬；程式語言之多，繁若《天官》之星宿，奇勝《山經》之走獸。鼠、蟹、鑽、魚，或以速稱。蛇、象、駱、犀，各爭文采。方知鬼之所以夜哭，天之所以雨粟。然以文言編程者，似所未有。此誠非文脈之所以傳，文心之所以保。嗟予小子，遂有斯志。然則數寸之烏絲猶覆於頭，萬卷之素書未破於手；一身長羈于远邦，兩耳久旷于雅言。然夫文章者吾之所宿好，程式者偶承時人之謬譽。故希孟不慚年少，莊生不望無涯。乃作斯言。誠未能嘔瀝長吉之心血，亦庶幾免於義山之流沫。既成之後，復學干將鑄劍而自飼，越王嚐糞而當先。自謂偶追《十書》之筆意，但恨少八家之淋漓。此子山所謂士衡抚掌而甘心，平子見陋而固宜。然則雖實覆甕之質，尚存斧正之望；雖乏呂相之金，易字之渴蓋同。此亦開源之大義，吾輩之所以勉勵也。一笑。

- 在尧舜时代，人们使用结绳和数手指来计算。当时怎么能够预料到，几百代人之后计算机的巧妙呢！计算机比鲁班（公输盘）的木鸢更加精巧，比诸葛亮（武侯）的木牛流马更好。此外，编程语言数量众多，如同《天官书》记录的星宿一般多，又比《山海经》中记录的飞禽走兽还要奇特。**Go**（鼠）、**Rust**（蟹）、**Ruby**（鑽）、**Fishshell**（鱼）因速度而出名。**Python**（蛇）、**Php**（象）、**Perl**（骆）和 **JavaScript**（犀）则各有独特之处。我这才理解到，为什么鬼会夜哭，天上会下粟雨。
- 但以往从未有人使用过文言文进行编程。这并不是传承文脉、保护文心的好方法，所以我才产生了用文言文编程的想法。我目前还太年轻，读过的书也没有破万卷。如今身处遥远的国家（美国），也很久没有接触中文了。但是我一直对文学很有兴趣，编写的程序有时候也得到人们的一些肯定。正如王希孟和庄子一般，并不因为年轻或者知识的浩瀚无涯而退缩，于是写下了这些话。
- 我既没有像李贺那样呕心沥血，也没有像李商隐那样口角流沫（形容读书勤奋）。项目完成后，我将继续以**干将铸剑**的精神勉励自己，带着越王**卧薪尝胆**的精神继续向前。我自己虽想效仿《算经十书》的笔法，只是遗憾没有唐宋八大家那样淋漓的文笔。正如庾信在《哀江南赋》所写：「陆机听了心甘情愿地拍掌；张衡见了将轻视它也是理所当然」（意指如果被人嘲笑，也是理所当然，我不会太过介意）。尽管这项目只有覆瓮的价值（一点微小的工作），但是还有完善的空间。虽然没有像吕不韦那样有一字千金的本钱，但是我对交流的渴望是一样的。这也正是开源的精神内核，我们以此互相勉励吧。

```
吾有一數。曰三。名之曰「甲」。
為是「甲」遍。
    吾有一言。曰「「問天地好在。」」。書之。
云云。
```

```
var n = 3;
for (var i = 0; i < n; i++) {
    console.log("問天地好在");
}
```

```
a = 3
for i in range(a):
    print("問天地好在")
```

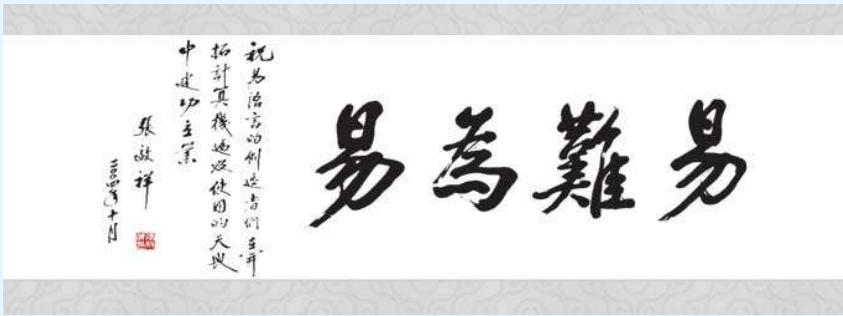
wenyan-lang Javascript Python

圖靈機者抽象計算機之模型也乃西人艾倫圖靈提出通用圖靈機者模擬任意圖靈機之圖靈機也或曰凡世間可算之物藉此皆可得是謂邱奇圖靈論題也

吾有一術名之曰圖靈機欲行是術必先得一列曰諸律二言曰始態曰終態乃行是術曰吾有一列名之曰帶充帶以白有數一名之曰針吾有二數曰一曰一名之曰左疆曰右疆吾有一言曰始態名之曰態吾有一術名之曰圖靈機畫法是術曰吾有一言名之曰畫帶有數左疆名之曰筆恆為是若筆大於右疆者乃止也夫帶之筆名之曰符若針等於筆者加符於〔加其以〕昔之符者今其是矣若非加符於、加其以、昔之符者今其是矣云云加畫帶以符昔之畫帶者今其是矣加一以筆昔之筆者今其是矣云云吾有四言曰態曰《曰畫帶曰》書之是謂圖靈機畫法之術也施圖靈機畫法噫恆為是凡諸律中之律夫律之一名之曰甲態夫律之二名之曰讀符夫律之三名之曰乙態夫律之四名之曰畫符夫律之五名之曰移若態等於甲態者若帶之針等於讀符者昔之帶之針者今畫符是矣昔之態者今乙態是矣若移等於左者減一於針昔之針者今其是矣云云若移等於右者加一於針昔之針者今其是矣云云乃止云云若針小於左疆者昔之帶之針者今白是矣昔之左疆者今針是矣云云若針大於右疆者昔之帶之針者今白是矣昔之右疆者今針是矣云云施圖靈機畫法噫若態等於終態者乃止也云云是謂圖靈機之術也吾有一術名之曰製律欲行是術必先得一列曰諸律五言曰甲態曰讀符曰乙態曰畫符曰移乃行是術曰吾有一列名之曰律充律以甲態以讀符以乙態以畫符以移充諸律以律是謂製律之術也

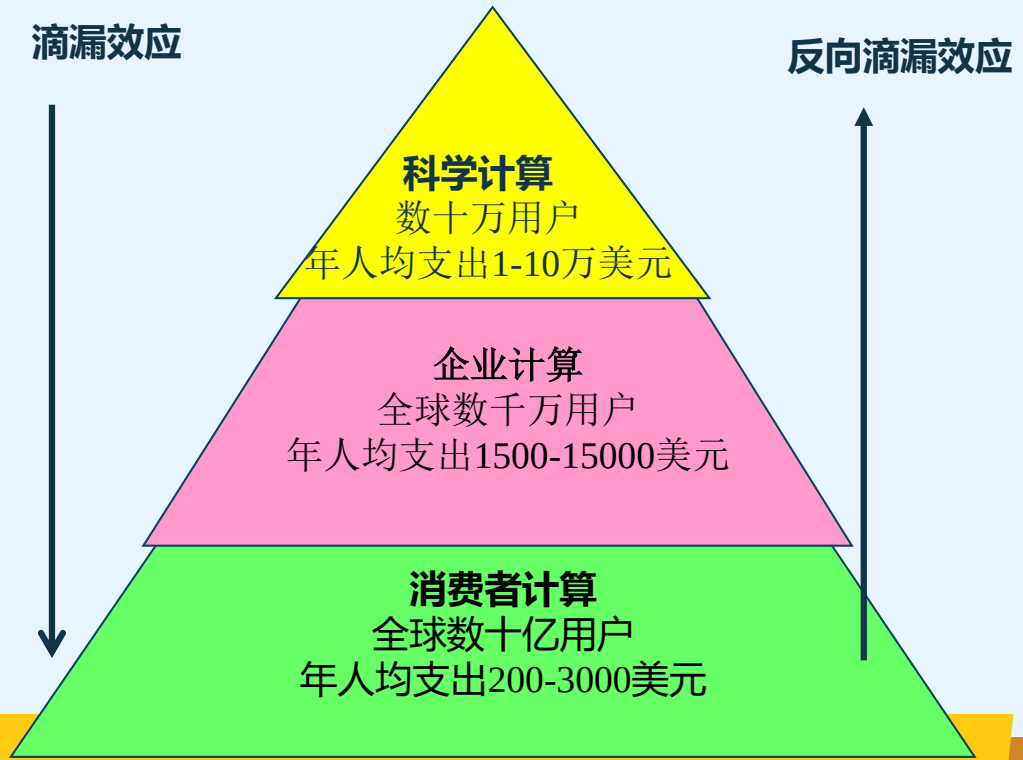
易语言

- 易语言（EPL）是一门以中文作为程序代码编程语言，其以“易”著称。易语言早期版本的名字为E语言。最早的版本的发布可追溯至2000年9月11日。
- 创造易语言的初衷是进行用中文来编写程序的实践，方便中国人以中国人的思维编写程序，并不用再去学习西方思维。
- 最新版本：5.9 (2019/05/24)
- 官网：http://www.dywt.com.cn/



图灵问题：如何开发出智能应用

- 计算机的应用类别
  - 科学计算
  - 企业计算
  - 消费者计算
- 滴漏效应



什么是智能？智能应用示例

- 1950年，图灵发表“计算机与智能”论文
  - 提出“图灵测试”和“模仿游戏”
    - 提问者（C）在一个房间中，向看不见的一个男人（A）和一个女人（B）通过电传设备提问并得到回答
    - 图灵测试：在模仿游戏中用一台计算机取代女人（B）
  - 猜想：2000年计算机进步足够大，能够通过图灵测试
- 对图灵测试的批评
  - 1980年，希尔勒提出**中文屋**（Chinese Room）实验
    - **强人工智能**（strong AI）：计算机真地理解中文
    - **弱人工智能**（weak AI）：计算机只是模仿理解中文
- 智能的（smart，intelligent）三类含义
  - 计算机化，密集计算，感知与认知能力

智能应用示例

- 计算机在智力竞赛中超过人类
  - 国际象棋、围棋
  - “危险边缘”知识问答
- 无人驾驶汽车竞赛
  - 野外无人车赛、城市无人车赛

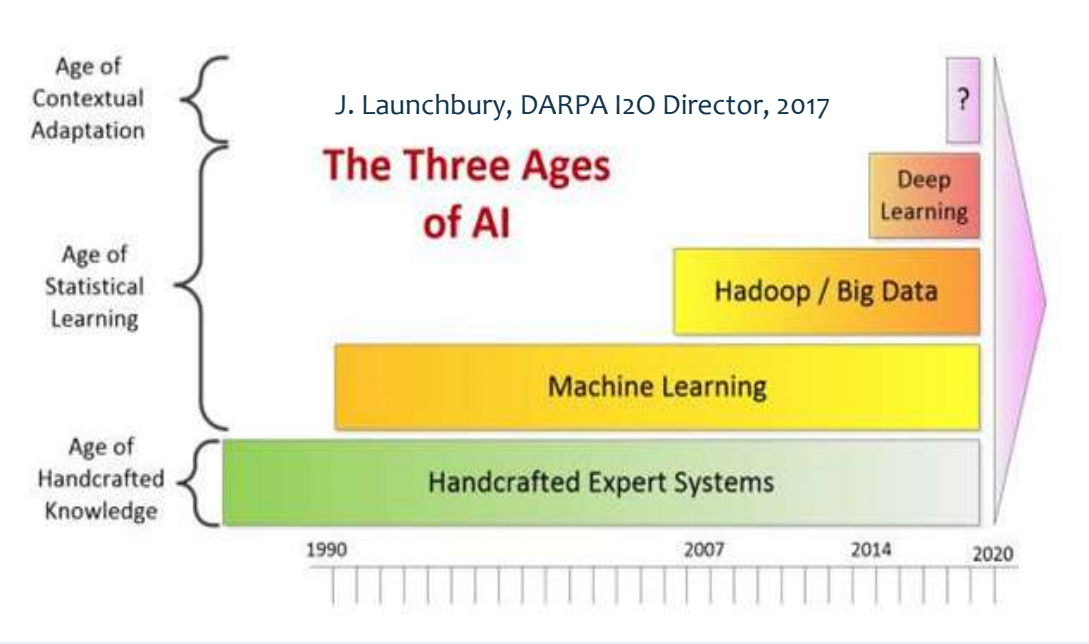
Q：尽管马耳他语从意大利语中引入大量单词，但它是从这个闪米特语族的这个方言发展而来的。
A：“什么是阿拉伯语？”

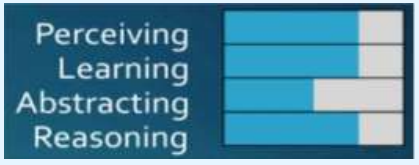
Q：这个语言的方言包括吴语，粤语和客家话。
A：“什么是广东话？”

Bob Davies, Rainer Lienhart, Using Cart to Segment Road Images, Technical Report, Technische Berichte der Fakultät für Angewandte Informatik der Universität Augsburg (2005-18), June 2006.

智能应用发展趋势

- 一种观点：人工智能发展的三次浪潮
  - DARPA：未来发展方向是AI的第三次浪潮
    - 综合感知，学习，抽象，推理等功能，实现可解释智能





实用的智能应用例子

- 近年来“深度学习”发展较大，得到多种应用
  - 图像识别
  - 语音识别
  - 用户画像
  - 语言翻译
  - 机器人

对真善美提出新挑战

写作机器人：越来越多的“报道”是机器人写的

“神翻译”：

对公服务 → To Male Service

美政路 → The United States Government Road

XX词典
美政路 → United States political road

5. 本课程的要点
对计算思维的十种理解

理解1：自动执行。 算得**快**！

理解2：正确性。 算得**对**！

理解3：通用性。 算得**广**！

理解4：构造性。 算得**妙**！

理解5：复杂度。 算得**妙**！

理解6：连接性。 算得**爽**！

理解7：协议栈。 算得**爽**！

理解8：抽象化。 算得**爽**！

理解9：模块化。 算得**爽**！

理解10：无缝衔接。 算得**爽**！

理论计算机

应对真实系统的复杂性

对计算思维的十种理解

理解1：自动执行。计算机能够自动执行由离散步骤组成的计算过程。

理解2：正确性。计算机求解问题的正确性往往可以精确地定义并分析。

理解3：通用性。计算机能够求解任意可计算问题。

理解4：构造性。人们能够构造出聪明的方法让计算机有效地解决问题。

理解5：复杂度。这些聪明的方法（算法）具备时间/空间复杂度。

理解6：连接性。很多问题涉及用户/数据/算法的连接体，而非单体。

理解7：协议栈。连接体的节点之间通过协议栈通信交互。

理解8：抽象化。少数精心构造的计算抽象可产生万千应用系统。

理解9：模块化。多个模块有规律地组合成为计算系统。

理解10：无缝衔接。计算过程在计算系统中流畅地执行。

自动

通用

算法

联网

抽象

计算机科学的重要抽象举例

- 软件
  - **算法**：巧妙的信息变换方法
  - **程序**：算法的代码实现
  - **进程**：运行时的程序
  - **指令**：程序的最小单位，计算机能够直接执行
- 硬件
  - **指令流水线**，所有指令都由这一种机制执行
    - 指令流水线由若干时钟周期组成
  - **时序电路**与**自动机**：说明每一个时钟周期的操作
    - 由组合电路与存储单元组成，理论上等同于**图灵机**
  - **组合电路**：实现二值逻辑，布尔逻辑

随时回到本源：计算过程是通过操作数字符号变换信息的过程

- ENIAC解决问题实例
  - 火炮弹道F(t)计算
  - 建模、调参、操作表

QUADRANT ELEVATION - MILS

Height of Target - Meters

Range Meters	-400	-300	-200	-100	0	100	200	300	400	500	600	700	800	900	1000
10000	318.3	329.9	341.5	353.2	364.9	376.6	388.4	400.1	412.0	423.8	435.7	447.7	459.6	471.7	483.7
10100	323.1	334.6	346.1	357.7	369.4	381.0	392.7	404.5	416.2	428.0	439.9	451.8	463.7	475.7	487.7
10200	327.8	339.3	350.8	362.3	373.9	385.5	397.1	408.8	420.5	432.3	444.1	455.9	467.8	479.7	491.7
10300	332.6	344.0	355.5	366.9	378.4	390.0	401.6	413.2	424.9	436.6	448.4	460.2	472.0	483.9	495.8
10400	337.5	348.8	360.2	371.6	383.1	394.6	406.1	417.7	429.3	441.0	452.7	464.4	476.2	488.1	500.0
10500	342.3	353.6	364.9	376.3	387.7	399.2	410.6	422.2	433.8	445.4	457.1	468.8	480.5	492.4	504.2
10600	347.2	358.4	369.7	381.0	392.4	403.8	415.2	426.7	438.3	449.9	461.5	473.2	484.9	496.7	508.6
10700	352.1	363.3	374.5	385.8	397.1	408.5	419.9	431.4	442.9	454.4	466.0	477.7	489.4	501.2	513.0
10800	357.1	368.2	379.4	390.6	401.9	413.2	424.6	436.0	447.5	459.0	470.6	482.2	493.9	505.7	517.5
10900	362.1	373.2	384.3	395.5	406.7	418.0	429.4	440.8	452.2	463.7	475.3	486.9	498.5	510.3	522.1
11000	367.1	378.1	389.2	400.4	411.6	422.9	434.2	445.5	457.0	468.4	480.0	491.6	503.2	515.0	526.8
11100	372.2	383.2	394.2	405.4	416.5	427.8	439.0	450.4	461.8	473.3	484.8	496.4	508.0	519.8	531.6
11200	377.3	388.2	399.3	410.4	421.5	432.7	444.0	455.3	466.7	478.1	489.7	501.2	512.9	524.6	536.5

$$x'' = -E(x' - w_x) + 2\Omega \cos L \sin a \; y'$$

$$y'' = -E\,y' - g - 2\Omega \cos L \sin a \; x'$$

$$z'' = -E(z' - w_z) + 2\Omega \sin L\, x' + 2\Omega \cos L \cos a \; y'$$

$$\bar{x}'_1 = x'_o + x''_o \Delta t$$

$$\bar{x}_1 = x_o + x'_o \Delta t$$

$$x'_1 = x'_o + (x''_o + \bar{x}''_1)\frac{\Delta t}{2}$$

$$x_1 = x_o + (x'_o + \bar{x}'_1)\frac{\Delta t}{2} + (x''_o - \bar{x}''_1)\frac{(\Delta t)^2}{12}$$

火炮弹道计算的微分方程与算法

符号有窍门
二进制补码例子：-127到127的整数加法

- 直截了当的表示
  - 需要8比特，2^7=128，以及1比特表示正负
  - 63 = 00111111，64 = 01000000
  - (-63) = 10111111，(-64) = 11000000
  - 63 + 64 = 00111111 + 01000000 = 01111111 = 127
  - (-63) + (-64) = 10111111 + 11000000 = 11111111 = (-127)
  - 63 + (-63) = 00111111 + 10111111 = 11111110 = (-126) **错!**
- 补码表示
  - 负数：绝对值的逐位取反，然后加00000001，即从最低一位加1
  - (-63) = 00111111逐位取反 + 00000001 = 11000000 + 00000001 = 11000001
  - 63 + (-63) = 00111111 + 11000001 = 00000000 = 0 **正确!**
- 溢出：127+1=？ (-127)+(-1)=?

如何表示实数：浮点数概念

- 本课程编程练习中不涉及
- 特定的有穷数字符号组合不能精确表达无穷数
- 办法：近似表达
  - 例如，圆周率π=3.14159265…，假设6位十进制精度

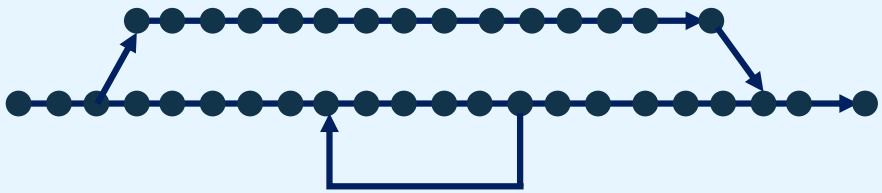
- 一个浮点数可由两个整数（定点数）表示

必须考虑“异常”并巧妙处理

- 是实现“自动执行”的保障
- 异常与正常同等重要，是矛盾统一
- 整数运算：从左到右与从右到左不一样
  - 8比特：127+127+（-127）= 127 或 溢出
- IEEE 754浮点计算标准的异常
  - Invalid operation，例如√(-3)，返回值 qNaN
  - Division by zero，例如1/0或log(0)，返回值±∞
  - Overflow，例如$9.999999\times10^{96} + 9.9\times10^{2}$，返回值±∞
  - Underflow，例如$1.9\times10^{-95} / 9.9\times10^{2}$，返回subnormal值
  - Inexact
  - **没有这些巧妙的设计，很多计算过程会报错终止，不能完成**

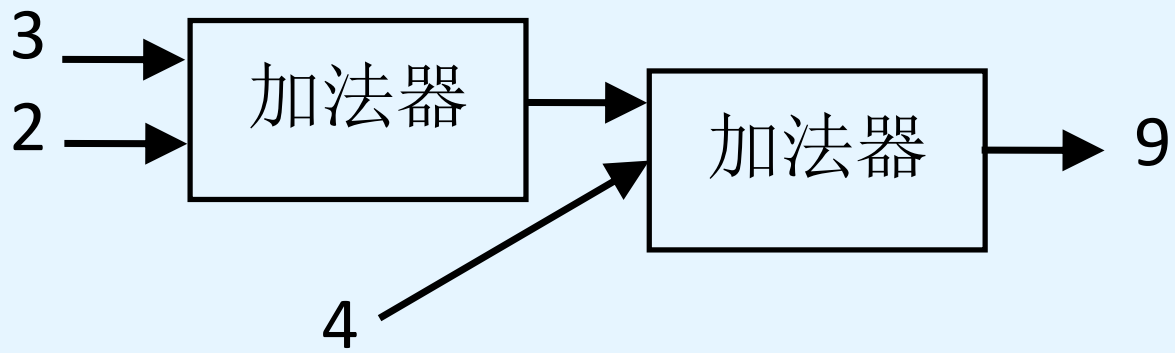
符号、操作、步骤、过程

- 一个过程是一个有限的步骤序列
  - 下图包含几个计算过程?
- 每个步骤包含“本步操作+1个下一步骤”
  - 下一步骤可是顺序或跳转
  - 操作可是:
    - 算术逻辑运算
    - 访存操作
    - 跳转操作
    - 输入输出操作
  - 它们都是数字符号变换操作



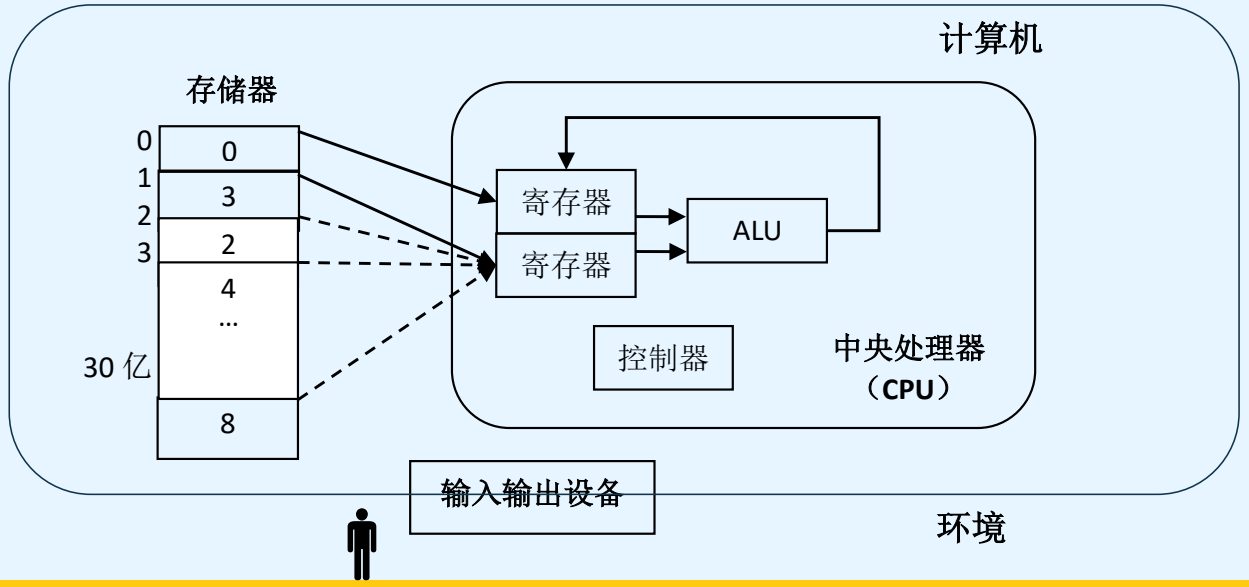
如何求3个数之和?

- 直接用硬件逻辑电路实现
  - 步骤可以是硬件



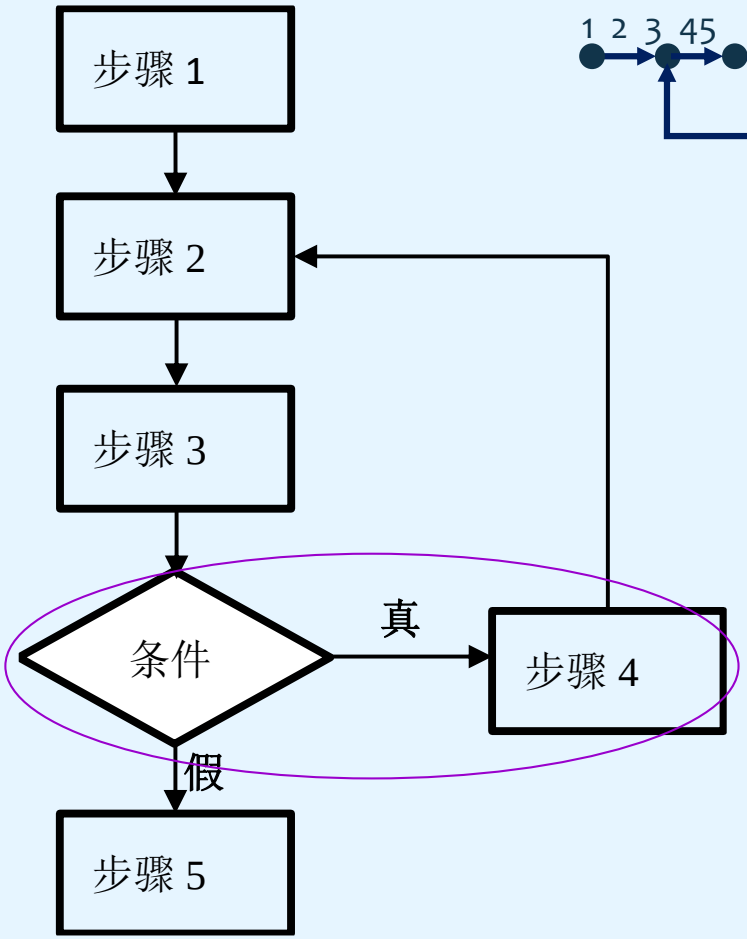
如何求30亿个数之和?

Step 1（第一步骤）：将存储器地址0里的数取到第一个寄存器
Step 2（第二步骤）：将存储器地址1里的数取到第二个寄存器
Step 3（第三步骤）：将两个寄存器的数在ALU相加，结果送到第一个寄存器
Step 4（第四步骤）：如果第二步的地址<30亿，将其地址增1，回到第二步骤
Step 5（第五步骤）：将第一个寄存器的数存入存储器地址0



用流程图描述计算过程

- Step 1（第一步骤）
  - 将存储器地址0里的数取到第一个寄存器
- Step 2（第二步骤）
  - 将存储器地址1里的数取到第二个寄存器
- Step 3（第三步骤）
  - 将两个寄存器的数送到ALU相加，结果送到第一个寄存器
- Step 4（第四步骤）
  - 如果第二步的地址<30亿，将其地址增1，回到第二步骤
- Step 5（第五步骤）
  - 将第一个寄存器的数存入存储器地址0

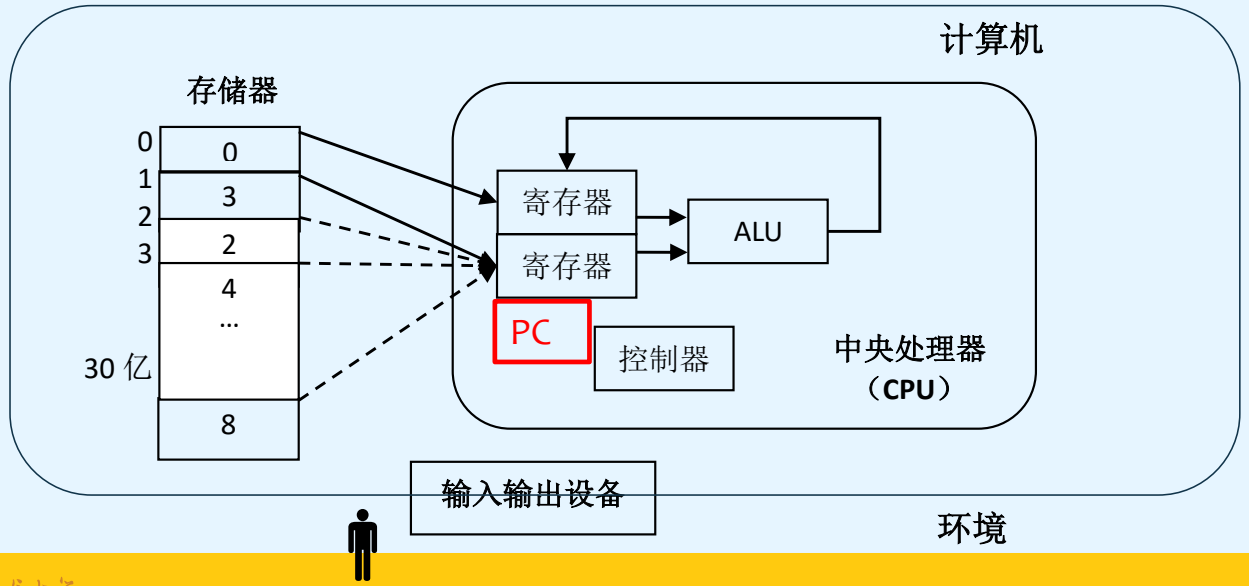


理解1：计算机自动执行由离散步骤组成的计算过程

- **计算过程刻画：**
  - 一个计算过程是解决某个问题的**有限个计算步骤**的执行序列。
  - 计算过程有一个**起始步骤**（第一步骤）。
  - 每个步骤执行一个操作，然后进行到下一步骤。
    - 每个操作可以是一个算术逻辑运算操作，也可以是一个访存操作、一个跳转操作、或是一个输入输出操作。它们都可被视为数字符号变换操作。
    - 下一步骤可以是顺序下一步骤，也可以是跳转操作的目的步骤。
  - 当不存在下一步骤时，计算过程**终止**。
- **计算思维要点**：“**精准地**描述信息变换过程的操作序列，并**有效地**解决问题”是如何体现的？
  - 精准性在于（1）比特精准地定义该问题涉及的数字符号、操作和步骤；（2）每个符号、操作、步骤要足够能行
  - 有效性体现于：除了输入输出操作之外，计算过程的所有步骤和操作在计算机中**自动执行**

计算过程自动执行的要点
PC机制，需要存储程序

Step 1（第一步骤）：将存储器地址0里的数取到第一个寄存器
Step 2（第二步骤）：将存储器地址1里的数取到第二个寄存器
Step 3（第三步骤）：将两个寄存器的数在ALU相加，结果送到第一个寄存器
Step 4（第四步骤）：如果第二步的地址<30亿，将其地址增1，回到第二步骤
Step 5（第五步骤）：将第一个寄存器的数存入存储器地址0



PC =
Step 2
Step 3
Step 4
Step 2
Step 3
Step 4
…
Step 2
Step 3
Step 4
Step 5

自动执行与无缝衔接

- 第五章进一步讲解自动执行：无缝衔接
  - 扬雄周期原理
  - 波斯特尔鲁棒性原理
  - 冯诺依曼穷举原理
  - 阿姆达尔定律
- 自动执行难题尚未完全解决
  - 基本解决了单机自动执行难题
  - 在多台计算机上执行的计算过程如何自动执行?
    - 网络思维（连通性、协议栈）
  - 人机物三元世界

感谢关注~！

中国科学技术大学
University of Science and Technology of China

计算机导论

中国科学技术大学
University of Science and Technology of China

课程信息

- 计算机导论（20学时/1学分）
- 时间地点：周二 6,7节，3A312

- 教师：张燕咏（yanyongz@ustc.edu.cn）
- 助教（4人）
  - 冷佳明（lengjm@mail.ustc.edu.cn）
  - 孟成真（czmeng@mail.ustc.edu.cn）
  - 肖宇煊（xiaoyx@mail.ustc.edu.cn）
  - 王凯（wkzcml@mail.ustc.edu.cn）

计算模型与逻辑思维

中国科学技术大学
University of Science and Technology of China

逻辑思维

- 逻辑——①思维的规律、规则：这个想法似乎不合逻辑。 ②研究思维规律的科学，即逻辑学。 ③客观事物的规律：历史的逻辑。 ④观点，主张。多用于贬义：霸权主义的逻辑。 （新华字典）
- 广义：泛指规律、道理
- 狭义：逻辑学
  - 哲学（古希腊）、数学（19世纪）
  - 计算机科学、语言学、心理学……

逻辑思维主要内容

- 逻辑学基础
  - 布尔逻辑、真值表
  - 合取范式、析取范式
  - 谓词逻辑
  - 公理系统
- 图灵机模型
- 后续课程
  - 离散数学、数理逻辑、理论计算机基础……

图灵机模型

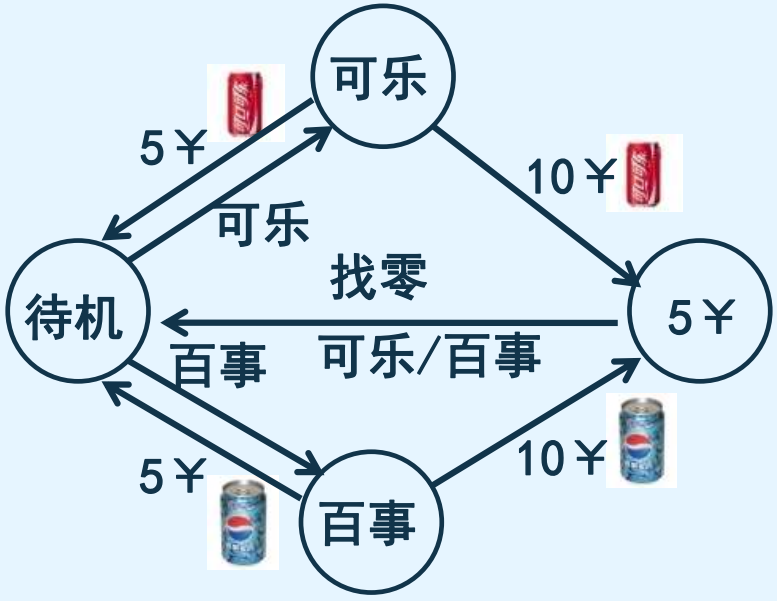
中国科学技术大学
University of Science and Technology of China

自动售卖机

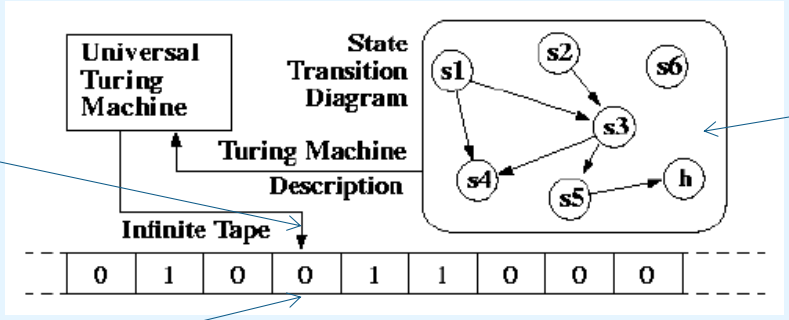
如何模拟售卖机？可乐和百事都是五块钱。只收五块和十块纸币。

自动售卖机

DFA：确定性有穷自动机



图灵机(Turing Machine)



head

state

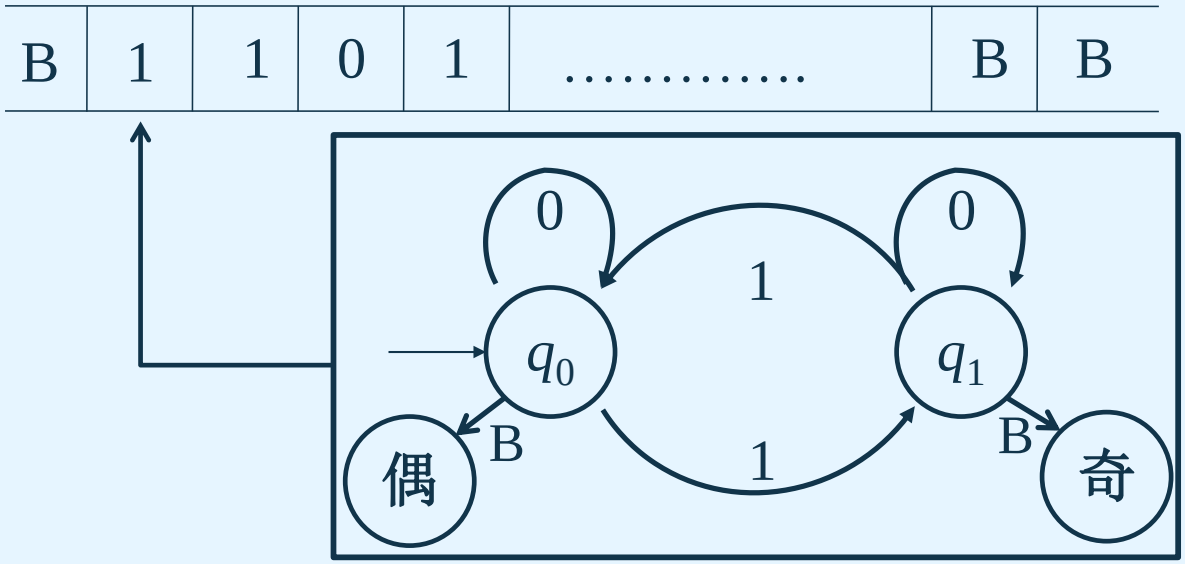
table

Tape

...an unlimited memory capacity obtained in the form of an infinite tape marked out into squares, on each of which a symbol could be printed. At any moment there is one symbol in the machine; it is called the scanned symbol. The machine can alter the scanned symbol and its behavior is in part determined by that symbol, but the symbols on the tape elsewhere do not affect the behavior of the machine. However, the tape can be moved back and forth through the machine, this being one of the elementary operations of the machine. Any symbol on the tape may therefore eventually have an innings. (by Turing 1948)

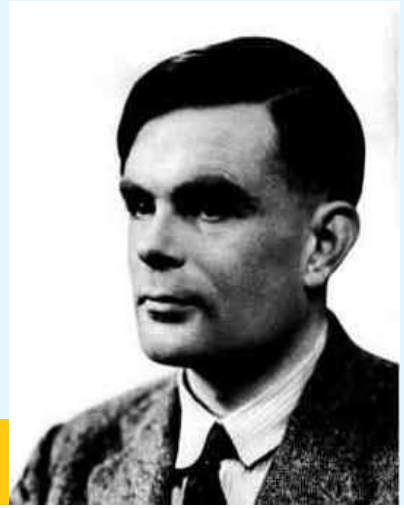
例1

- 输入：11010011…111，判断有奇数还是偶数个1。



图灵机

- 图灵机是一个七元组，{Q，Σ，Γ，δ，$q_0, q_{\text{accept}}, q_{\text{reject}}$}，其中 Q，Σ，Γ 都是有限集合，
  - 状态集合Q；
  - 输入字母表Σ；
  - 带字母表Γ，其中 B∈Γ；
  - 转移函数: δ：$Q\times\Gamma\rightarrow Q\times\Gamma\times\{\rightarrow,\leftarrow\}$
  - 起始状态 q_0∈Q；
  - 接受状态q_{accept}；
  - 拒绝状态q_{reject}。

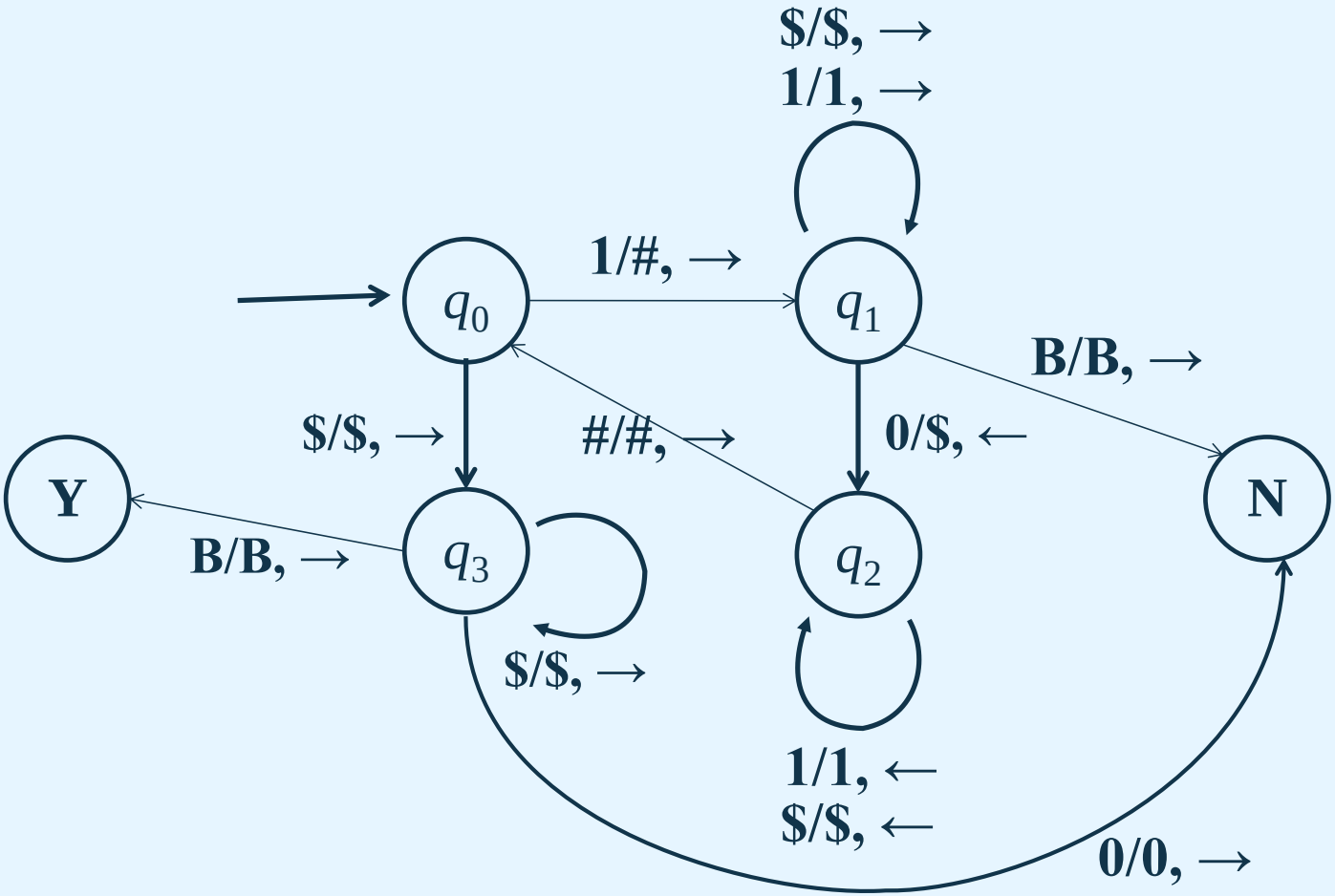


例2

- 输入：111…11000…00，判断1和0的个数是否相等？

例2

- 输入：111…11000…00，判断1和0的个数是否相等？
- （1） 从最左端开始;如果读到1， 置为#; 继续往右，如果读到0，替换为$; 回到左端重复此步骤；如果没读到0，则退出（No）
- (2) 如果纸带上只剩下0/1， 输出No；否则输出Yes


$/$, →
1/1, →
1/#, →
q_0
q_1
B/B, →
$/$, →
#/#, →
0/$, ←
Y
N
B/B, →
q_3
q_2
$/$, →
1/1, ←
$/$, ←
0/0, →

- 转移规则：
- $(q_0,1)\rightarrow(q_1,\#, R)$, $(q_0,\$)\rightarrow(q_3,\$, R)$
- $(q_1,1)\rightarrow(q_1,1, R)$, $(q_1,\$)\rightarrow(q_1,\$, R)$, $(q_1,0)\rightarrow (q_2,\$, L)$, $(q_1,B)\rightarrow (q_{reject},B, R)$
- $(q_2,1)\rightarrow(q_2,1, L)$, $(q_2,\$)\rightarrow(q_2,\$, L)$, $(q_2,\#)\rightarrow (q_0,\#, R)$
- $(q_3,\$)\rightarrow(q_2,\$, R)$, $(q_3,0)\rightarrow(q_{reject},0, R)$, $(q_3,B)\rightarrow(q_{accept},B, R)$

思考题

- 用图灵机判断 1^n0^n，需要大约$2n^2$次状态转移，是否能够做的更快？

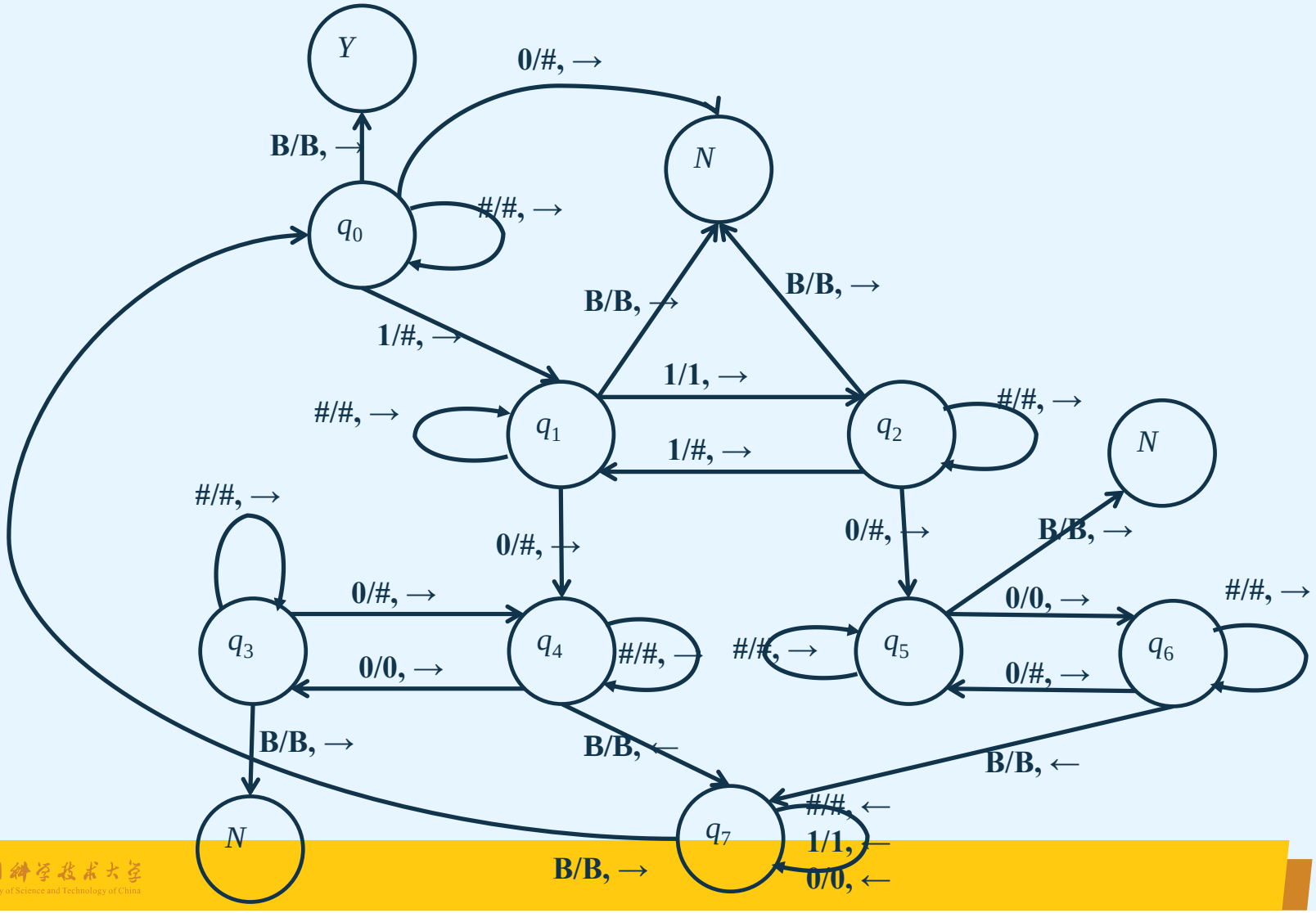
思考题

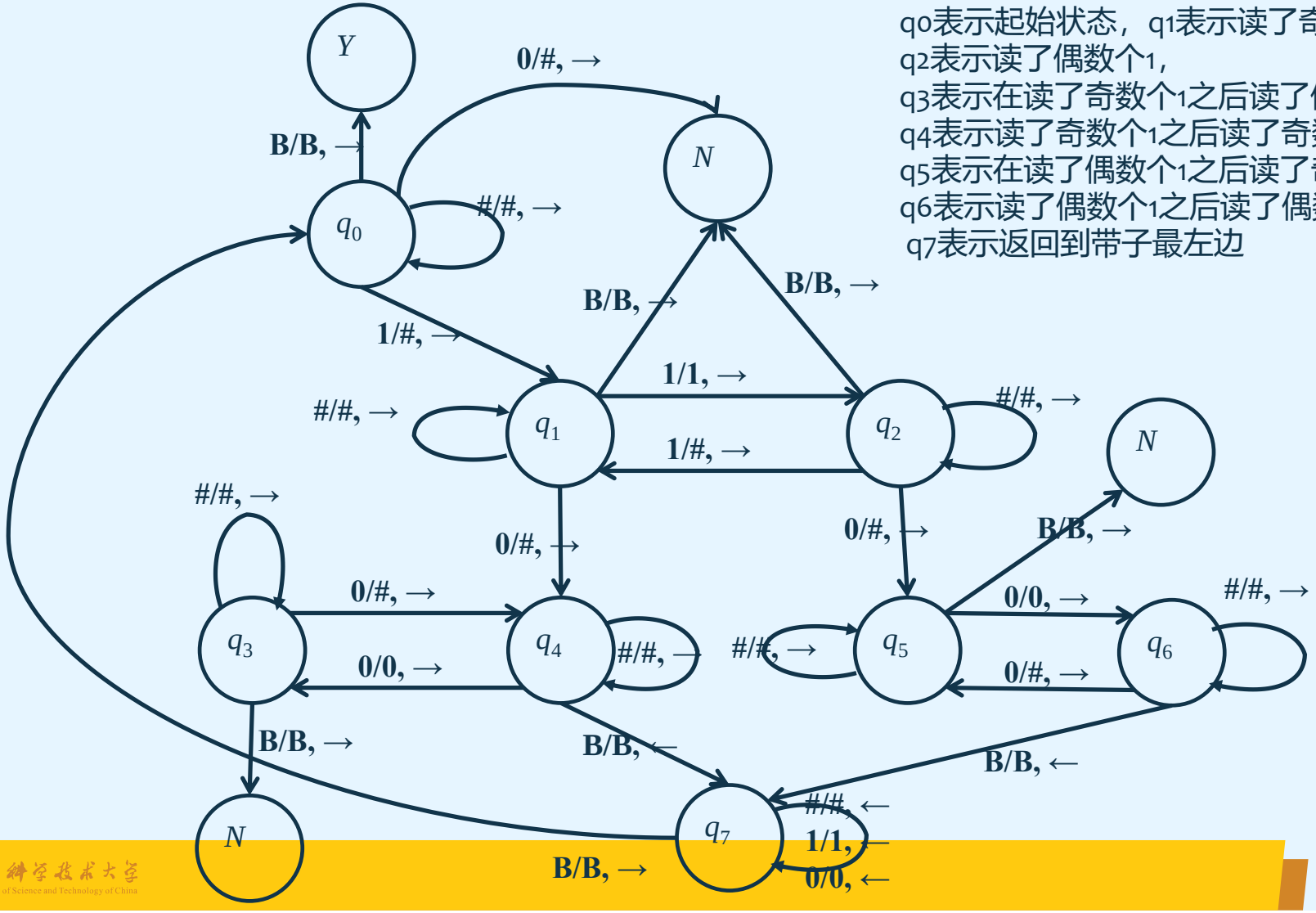
- 方案一：每次匹配最中间的1和0
  - 1+2+3+4+…+(2n-1)≈ $2n^2$
- 方案二：增加纸带字符数目或状态数
  - 111 → $#%
  - q_1, q_2, q_3
  - $\approx cn^2$

思考题

- 能不能用更少的转移次数？

- 每次不是只把一个1改成#，而是把**“一半”**的1改成#！
- $\approx cn \log n$


Y
N
N
N
q_0
q_1
q_2
q_3
q_4
q_5
q_6
q_7
0/#, →
B/B, →
#/#, →
1/#, →
1/1, →
0/0, →
B/B, ←
#/#, ←
1/1, ←
0/0, ←



q0表示起始状态，q1表示读了奇数个1.
q2表示读了偶数个1，
q3表示在读了奇数个1之后读了偶数个0，
q4表示读了奇数个1之后读了奇数个0.
q5表示在读了偶数个1之后读了奇数个0，
q6表示读了偶数个1之后读了偶数个0.
q7表示返回到带子最左边

图灵机模型

- 这是一个用于计算问题的数学模型
- 核心：状态转移函数
- 注意：状态集合是有限的！

- 图灵机模型的计算能力
  - 哪些问题能用图灵机解决？

- Alonzo Church & Alan Turing:
  Church-Turing Hypothesis:

Any reasonable attempt to model mathematically computer algorithms and their performance is bound to end up with a model of computation and associated time cost that is equivalent to Turing machines within a polynomial.

Alan Turing
1912-1954

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. TURING.

[Received 28 May, 1936.—Read 12 November, 1936.]

The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable *numbers*, it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are, however, the same in each case, and I have chosen the computable numbers for explicit treatment as involving the least cumbrous technique. I hope shortly to give an account of the relations of the computable numbers, functions, and so forth to one another. This will include a development of the theory of functions of a real variable expressed in terms of computable numbers. According to my definition, a number is computable if its decimal can be written down by a machine.

In §§ 9, 10 I give some arguments with the intention of showing that the computable numbers include all numbers which could naturally be regarded as computable. In particular, I show that certain large classes of numbers are computable. They include, for instance, the real parts of all algebraic numbers, the real parts of the zeros of the Bessel functions, the numbers π, e, etc. The computable numbers do not, however, include all definable numbers, and an example is given of a definable number which is not computable.

Although the class of computable numbers is so great, and in many ways similar to the class of real numbers, it is nevertheless enumerable. In § 8 I examine certain arguments which would seem to prove the contrary. By the correct application of one of these arguments, conclusions are reached which are superficially similar to those of Gödel†. These results

† Gödel, "Über formal unentscheidbare Sätze der Principia Mathematica und ver-

逻辑学基础

中国科学技术大学
University of Science and Technology of China

布尔逻辑

- 真 T (true)，假 F (false)
  - 今天下雨。
  - $a^2 \geq 0$。
- 合取，与 $\wedge$ (conjunction, and)
  - $x \wedge y = 1$ (T) *iff* $x = y = 1$ (T)
- 析取，或 $\vee$ (disjunction, or)
  - $x \vee y = 0$ (F) *iff* $x = y = 0$ (F)

- 非 ¬ (negation, not)
  - $\neg x = 1$ (T) *iff* $x = 0$ (F)
- 蕴含 $\rightarrow$ (material implication)
  - $(x \rightarrow y) = 1$ (T) *iff* $x = 0$ (F) or $y = 1$ (T)
- 异或 $\oplus$(exclusive or)
  - $x \oplus y = 1$ (T) *iff* $x \neq y$

真值表

x	y	$x \wedge y$	$x \vee y$	$x \rightarrow y$	$x \oplus y$
0	0	0	0	1	0
0	1	0	1	1	1
1	0	0	1	0	1
1	1	1	1	1	0

布尔函数

- 布尔函数 $f:\{0,1\}^n \to \{0,1\}$
- 举例
  - $g(x_1, x_2, \cdots, x_n) = (\vee_{i=1}^{n-1} x_i) \oplus x_n$
- 两个布尔函数相同：有相同的真值表
  - 类似：$f(x,y) = x^2 - y^2$和$g(x,y) = (x+y)(x-y)$是相同的多项式

- 性质：
  - $x \vee 0 = x,\ x \vee 1 = 1,\ x \wedge 0 = 0,\ x \wedge 1 = x$
  - $x \vee \neg x = 1,\ x \wedge \neg x = 0$
  - $x \wedge y = y \wedge x,\ x \vee y = y \vee x,\ x \oplus y = y \oplus x$ （**交换律**）
  - $(x \wedge y) \wedge z = x \wedge (y \wedge z)$
  - $(x \vee y) \vee z = x \vee (y \vee z)$ （**结合律**）
  - $(x \wedge y) \vee z = (x \vee z) \wedge (y \vee z)$
  - $(x \vee y) \wedge z = (x \wedge z) \vee (y \wedge z)$ （**分配律**）
  - more ??

- 性质：
  - $\neg(x \vee y) = \neg x \wedge \neg y$
  - $\neg(x \wedge y) = \neg x \vee \neg y$
  - $x \rightarrow y = \neg x \vee y$
  - $x \rightarrow y = \neg y \rightarrow \neg x$
  - $x \oplus y = (\neg x \wedge y) \vee (x \wedge \neg y)$
  - $x \oplus y = (x \vee y) \wedge (\neg x \vee \neg y)$

$x \wedge y = \neg(\neg x \vee \neg y)$

$x \vee y = \neg(\neg x \wedge \neg y)$

（**De Morgan律**）

（**逆否命题**）

- 合取范式(conjunctive normal form, CNF)
  - $f(x_1,\ldots,x_n)= Q_1 \wedge Q_2 \wedge Q_3 \ldots \wedge Q_m$
  - 其中: $Q_i = l_1 \vee l_2 \vee \ldots \vee l_n$，　$l_j = x_j$ 或 $\neg x_j$

- 析取范式(disjunctive normal form, DNF)
  - $f(x_1,\ldots,x_n)= Q_1 \vee Q_2 \vee Q_3 \ldots \vee Q_m$
  - 其中: $Q_i = l_1 \wedge l_2 \wedge \ldots \wedge l_n$，　$l_j = x_j$ 或 $\neg x_j$

- 例：$x \rightarrow (y \rightarrow z) = ?$

x	y	z	f
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

- 合取范式：$(\neg x \vee \neg y \vee z)$
- $(\neg x \wedge \neg y \wedge \neg z) \vee (\neg x \wedge \neg y \wedge z) \vee (\neg x \wedge y \wedge \neg z) \vee (\neg x \wedge y \wedge z) \vee (x \wedge \neg y \wedge \neg z) \vee (x \wedge \neg y \wedge z) \vee (x \wedge y \wedge z)$

思考题

- 根据De Morgan律和合取范式（或析取范式），使用¬和∧(或∨)可以表示出所有的布尔函数 (YES!）。
- 问题：能否只用∧, ∨表示出所有的布尔函数？
  - **不能！**

思考题

- 问题：能否只用$\oplus$？（允许使用0,1）
- 问题：如果只用$\rightarrow$呢？（允许使用0,1）

思考题

- 问题：能否只用$\oplus$？（允许使用0，1）
  - **不能！**
  - **只能写出** $x_{i1} \oplus \ldots \oplus x_{ik} \oplus c$
- 如果只用$\rightarrow$呢？
  - **可以！**
  - $x \rightarrow 0 = \neg x$
  - $(x \rightarrow 0) \rightarrow y = x \vee y$

谓词逻辑

- **全称量词**$\forall$ 和**存在量词**$\exists$
  - “都不是”和“不都是”
  - $\forall x\,(\neg P(x)),\quad \exists x(\neg P(x))$
  - $\exists x\big(\neg P(x)\big) = \neg(\forall x\,\big(P(x)\big))$

谓词逻辑

- 量词的范围
  - 任何一个自然数，要么它本身为偶数，要么加1后为偶数。
  - $\forall n\ [\mathrm{Even}(n) \vee \mathrm{Even}(n+1)]$，其中断言$\mathrm{Even}(n)$表示 n是偶数
  - $\forall n \in \mathbb{N}\ [\mathrm{Even}(n) \vee \mathrm{Even}(n+1)]$
- 量词的顺序
  - $\forall\, x, \exists\, y\ (y = x + 1)$
  - $\exists\, y, \forall\, x\ (y = x + 1)$

逻辑思维

- 存在无穷多个素数。
- 如何用数学语言描述？

Euclid of Alexandria
Elements

逻辑思维

- 存在无穷多个素数。
- 如何用数学语言描述？

Euclid of Alexandria
Elements

$$\forall n, \exists m, [(m > n) \wedge (\text{Prime}(m))]$$

$$\forall n, \exists m, \forall p, q\,[(m > n) \wedge (p, q > 1 \rightarrow pq \neq m)]$$

- 对$n>2$，丢潘图方程 $x^n + y^n = z^n$ 不存在非平凡解。

$$\forall a,b,c,n[(abc \neq 0) \wedge (n > 2) \rightarrow a^n + b^n \neq c^n]$$

- 四色定理：任何平面图都可以被四着色，使得任何两个相邻的顶点不同色。

$$\forall \text{平面图} G=(V,E), \exists c: V \to \{1,2,3,4\} \ \ \forall (u,v) \in E, [c(u) \neq c(v)]$$

- 可满足性问题(SAT)：给定CNF公式φ，是否存在一种赋值，使得这个CNF为真

$$(\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee \neg x_4 \vee \neg x_6) \wedge (x_2 \vee x_4 \vee \neg x_8) \wedge \ldots$$

$$\exists A: \{x_1, x_2, \ldots, x_n\} \to \{0,1\}, [\varphi(x_1, \ldots, x_n) = 1]$$

思考题

- 一位探险者在奥斯仙境旅行，他想要去翡翠城，但路上必须经过说谎国。说谎国的人永远说谎话，而诚实国的人永远讲真话。他走到了一个岔路口，两条路分别通向诚实国和说谎国。他不知道哪一条路是去说谎国的路。正在他犹豫不决的时候，路上来了两个人，已知其中一个是诚实国的，另一个是说谎国的。探险者需要问这两个人哪一条是去往说谎国的路，请问他应该怎么问？
- 要求：他只能问一次，而且答案只能是“是”或者“否”。

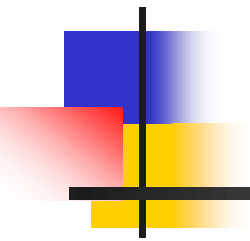
解答

- 两个人分别记做A和B，两条路分别记做S和T。
- 用0, 1表示来自诚实国和说谎国，因此A=1, B=0；或者A=0, B=1
- 用S=1表示走路S可以到说谎国，S=0表示走路S不能够到说谎国（i.e. 走路P可以到说谎国）。
- 注意到：无论问A还是问B“你来自诚实国吗？”回答总是Yes.
- 问A：$A \oplus S=?$
- 如果=0, 则走路S，否则走路P。

你对于“你来自诚实国”和“路S通往说谎国”这两个问题的回答是一样的吗？

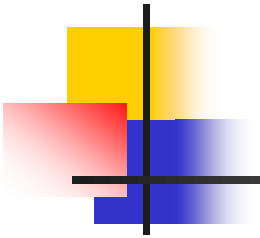
感谢关注~！

中国科学技术大学
University of Science and Technology of China

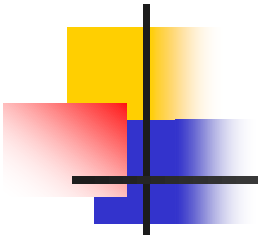


计算机科学导论

算法思维

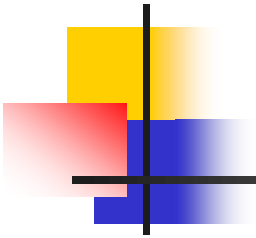


算法思维



引入：排序

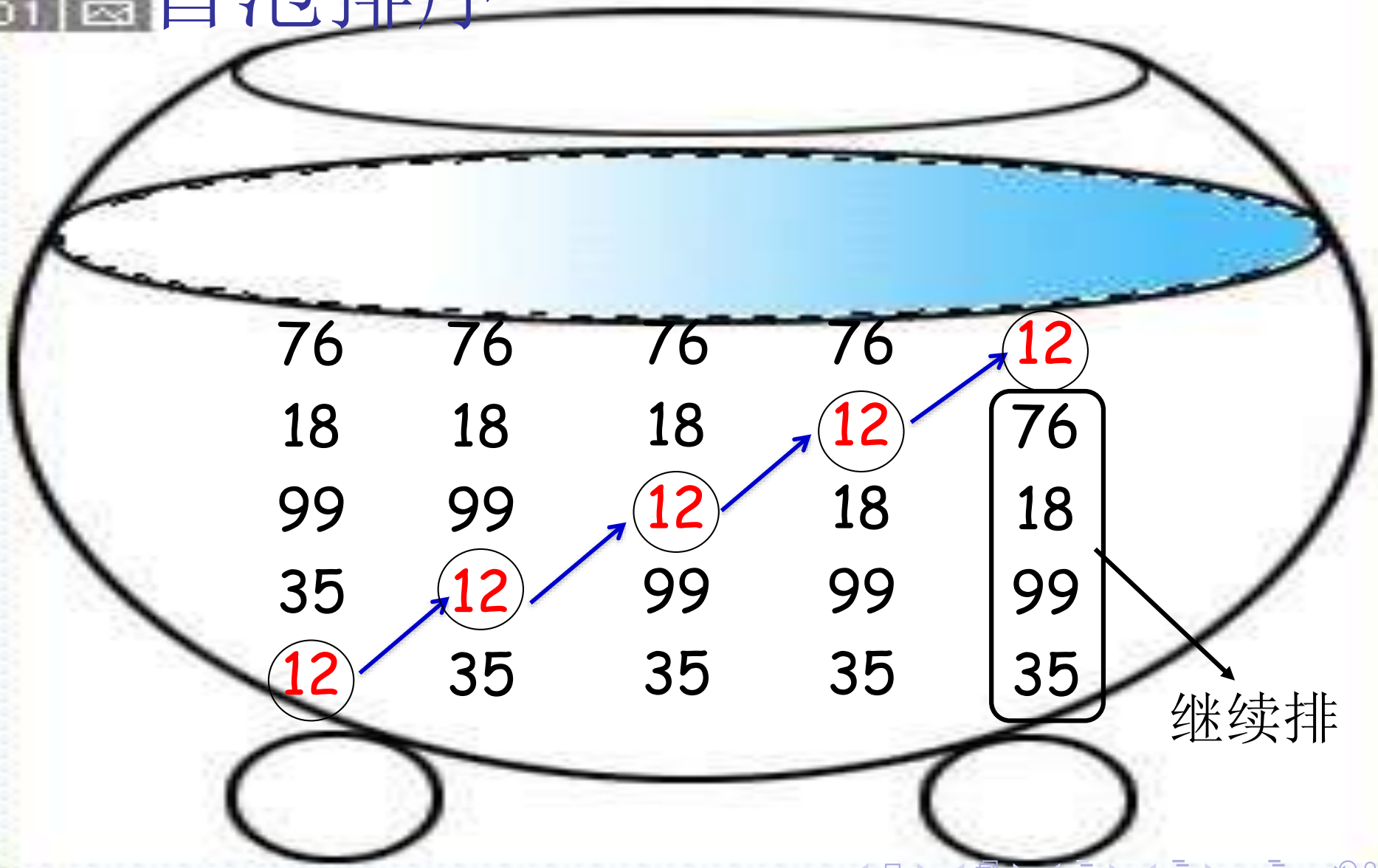
- 任务：给定n个正整数，把他们从小到大排起来
- 请思考在打斗地主、升级、桥牌.....的时候，你是怎么整理牌的？



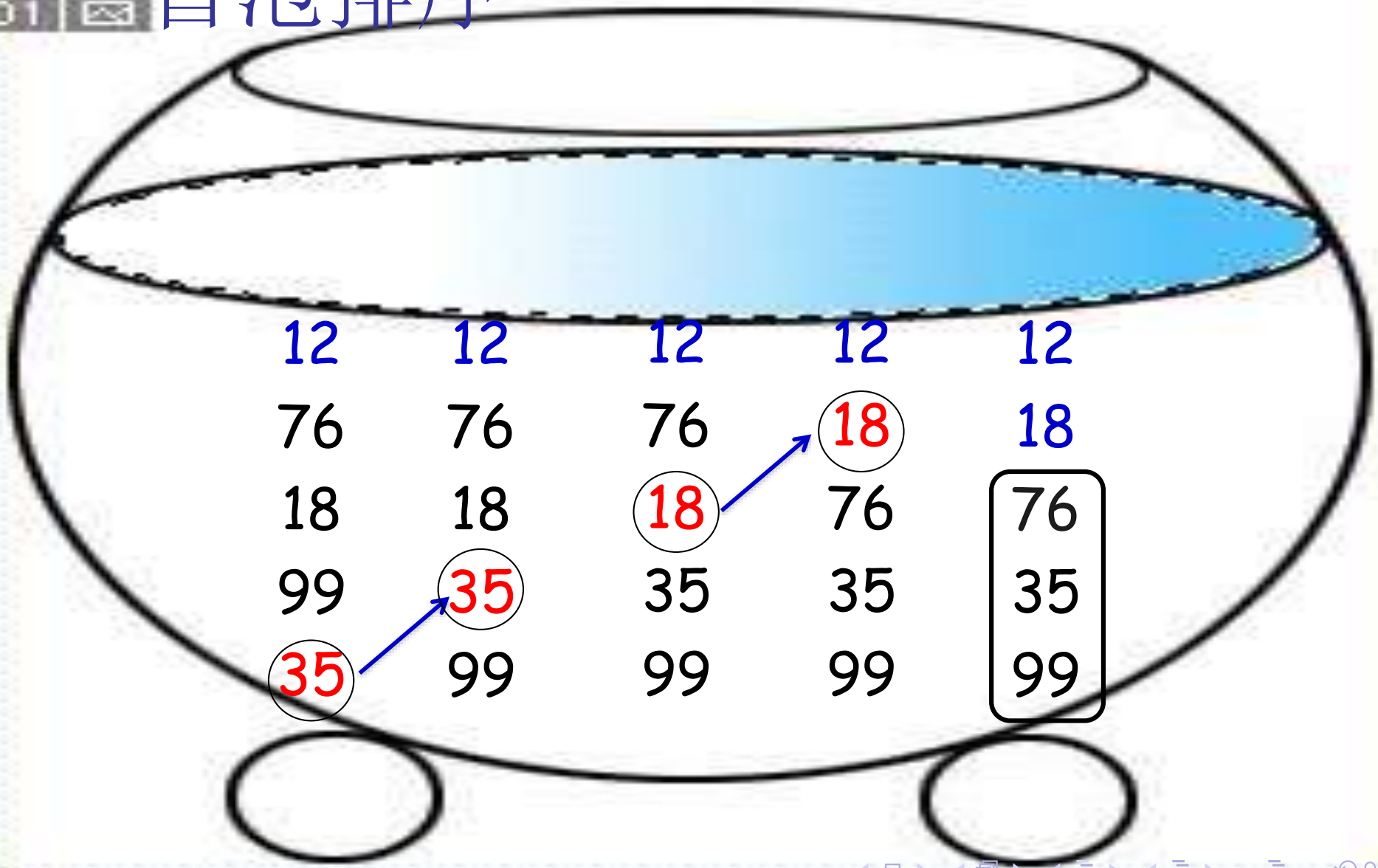
引入：排序

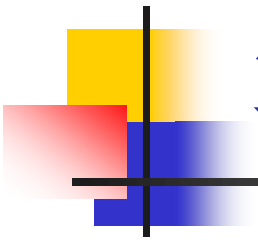
- 任务：给定n个正整数，把他们从小到大排起来
- 请思考在打斗地主、升级、桥牌.....的时候，你是怎么整理牌的？
  - 一张一张摸牌，每次插入已经理好的牌里面（插入排序法）
  - 把最大的牌放到最左边（选择排序法）
  - 先按花色分，再各个花色整理

冒泡排序


76 76 76 76 12
18 18 18 12 76
99 99 12 18 18
35 12 99 99 99
12 35 35 35 35
继续排

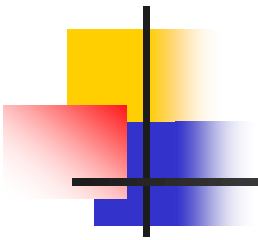
冒泡排序


12 12 12 12 12
76 76 76 18 18
18 18 18 76 76
99 35 35 35 35
35 99 99 99 99



算法

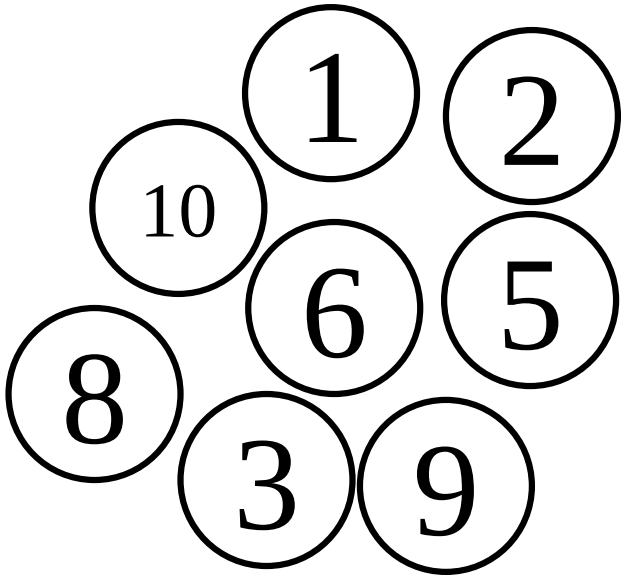
- 有穷性：在有穷步骤内终止执行
- 明确性：执行指令明确
- 输入。
- 输出。
- 可行性。



冒泡排序

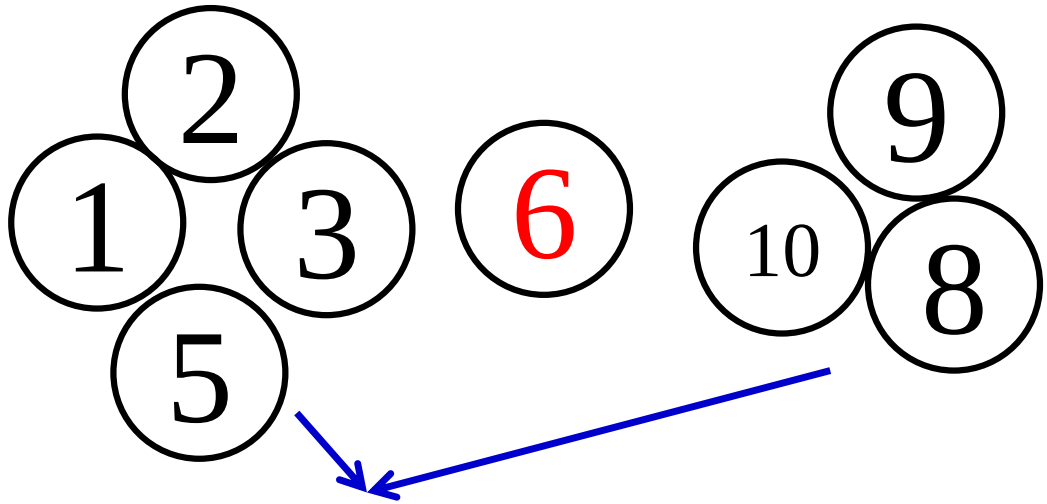
- 需要n-1+n-2+...+1=n*(n-1)/2次比较操作
- 最坏情况下需要n*(n-1)/2次交换操作
  - n, n-1, ..., 2, 1
- 能不能更快？

快速排序



Step1：随机选其中一个数字

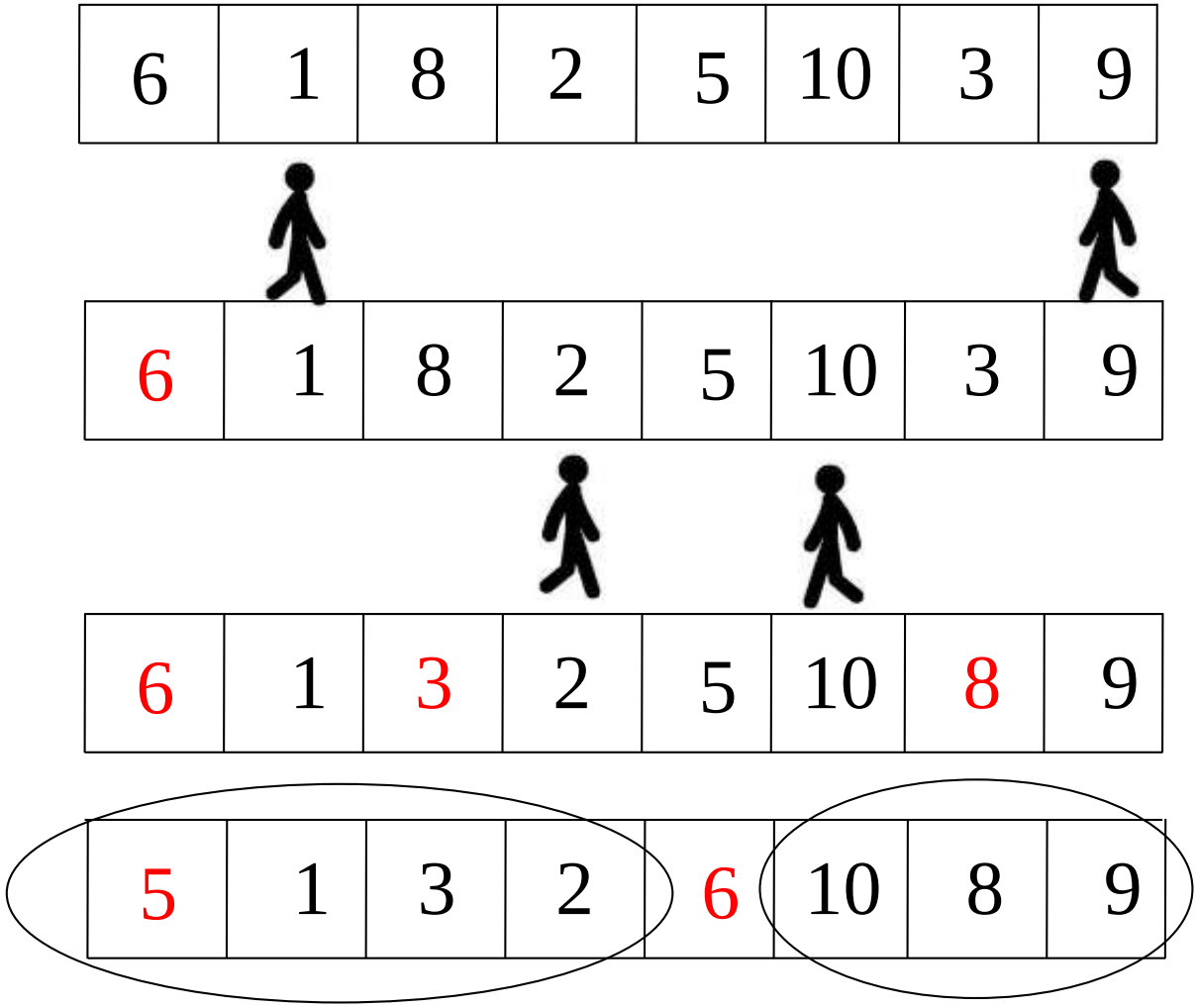
Step2：其他数字按照和基准数的大小关系分成两部分

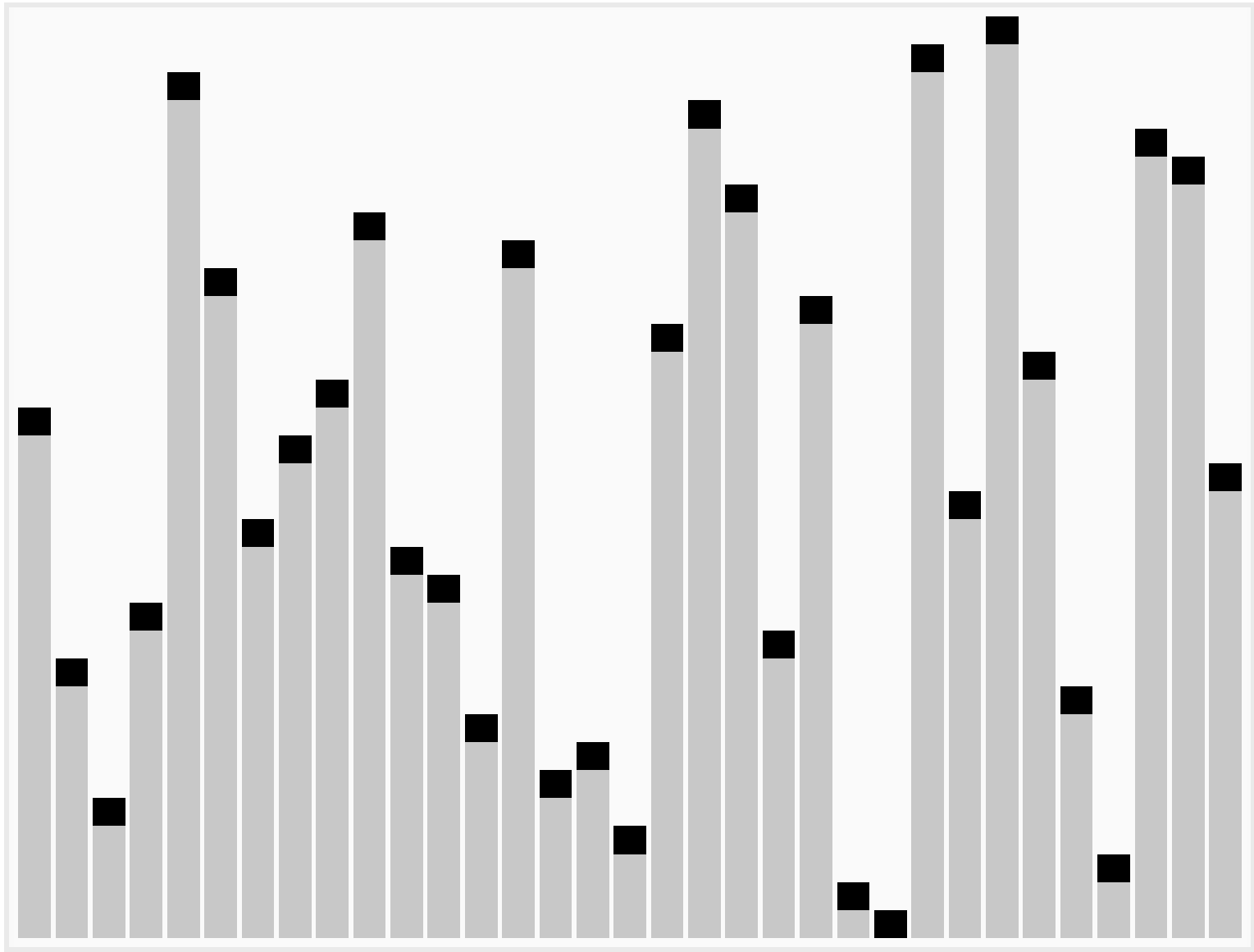


Step3：分别递归

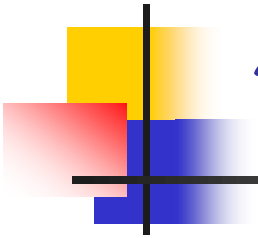
快速排序

基准数6



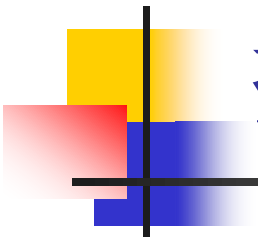


https://en.wikipedia.org/wiki/Quicksort



快速排序

- 需要多少次比较？
  - 最好情况：$n\log_2 n$
  - 最坏情况：$n*(n-1)/2$
  - 平均情况：$O(n\log_2 n)$



排序问题

- 冒泡排序
- 快速排序
- 还有更多的排序算法
  - 选择排序、插入排序、归并排序、堆排序、二叉搜索树、基数排序.....
  - 外部排序

小o，大O记号

- 冒泡排序
  - 运行时间 $O(n^2)$
  - 大O它是用另一个（通常更简单的）函数来描述一个函数数量级的渐近上界
- 快速排序
  - 平均运行时间 O(n logn)
  - 是$o(n^2)$
  - 小o则表示一个函数渐进地小于另一个函数，没有等于

小o，大O记号

- $f(n) = o(g(n))$: $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$
  - $n^{1.58} = o(n^2)$
  - **$n^{1000} = o(2^n)$**
  - $(\log n)^{200} = o(n)$
- $f(n) = O(g(n))$: $\exists$常数$c > 0$, $f(n) \le cg(n)$ 对充分大的n成立
  - $n^{1.58} = O(n^2)$　　$10n^{1.58} = O(n^{1.58})$
  - **$10^{1000}\, n = O(n)$**

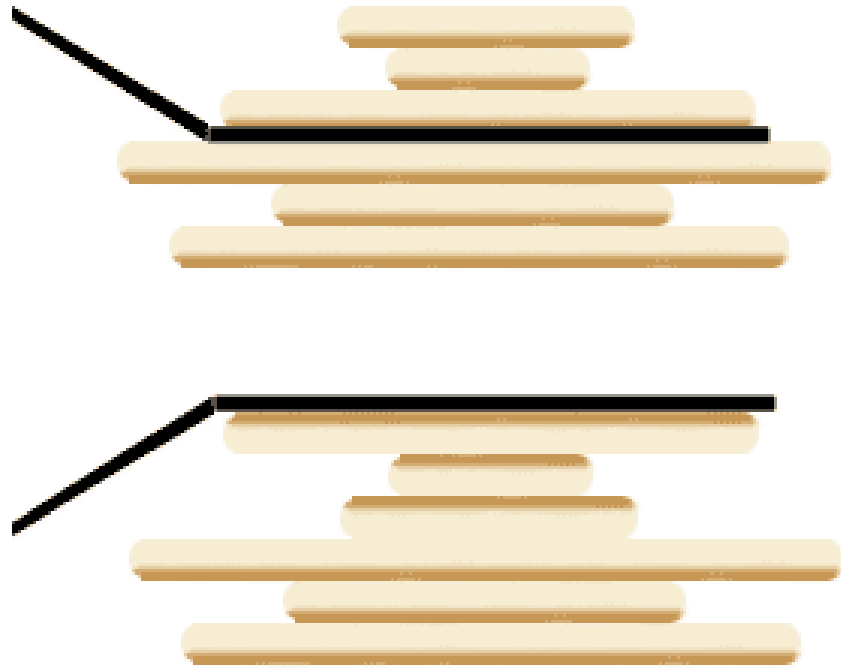
$\Omega(\cdot), \Theta(\cdot)$记号

- $f(n) = \Omega(g(n))$: ∃常数$c > 0, f(n) \geq cg(n)$ 对充分大的n成立
  - $n^2 = \Omega(n^{1.58})$　　$10n^{1.58} = \Omega(n^{1.58})$
- $f(n) = \Theta(g(n))$:
  - $f(n) = O(g(n))$ 并且 $f(n) = \Omega(g(n))$
  - $10n^2 - 20n + 45 = \Theta(n^2)$

思考题

- 翻煎饼问题(Pancake Sorting)

一个厨师做了一叠大小不同的煎饼，他要不断从上面拿起几个煎饼翻到下面。假设有**n**个煎饼，厨师需要翻动多少次，才能把煎饼按从小到大排好？



翻煎饼问题

- 2, 1, 4, 5, 3

- 5, 4, 1, 2, 3

- 3, 2, 1, 4, 5

- 1, 2, 3, 4, 5

The simplest pancake sorting algorithm performs at most $2n - 3$ flips. In this algorithm, a kind of selection sort, we bring the largest pancake not yet sorted to the top with one flip; take it down to its final position with one more flip; and repeat this process for the remaining pancakes.

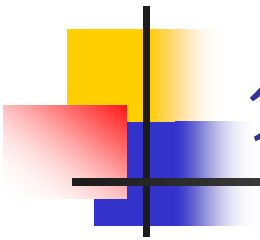
更关注如何减少翻的次数，不在意数字大小的比较

汉诺塔

有三根杆子A，B，C。A杆上有 N 个 (N>1) 穿孔圆盘，盘的尺寸由下到上依次变小。要求按下列规则将所有圆盘移至 C 杆：

1. 每次只能移动一个圆盘；
2. 大盘不能叠在小盘上面。

提示：可将圆盘临时置于 B 杆，也可将从 A 杆移出的圆盘重新移回 A 杆，但都必须遵循上述两条规则。

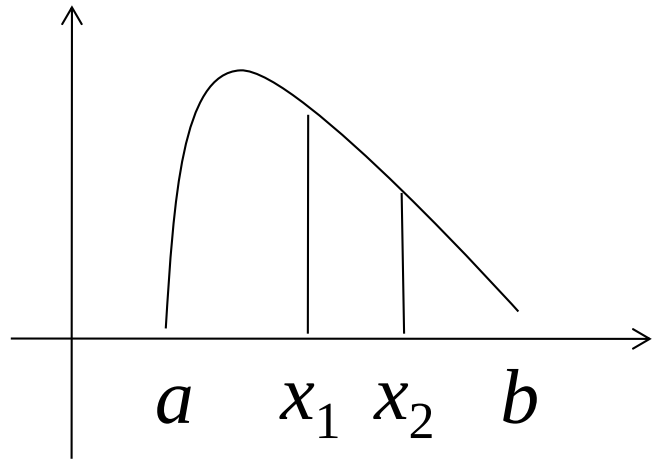


分治算法

- 单因素优选法: $f(x)$在$[a, b]$上先递增后递减，找出$\max f(x)$.

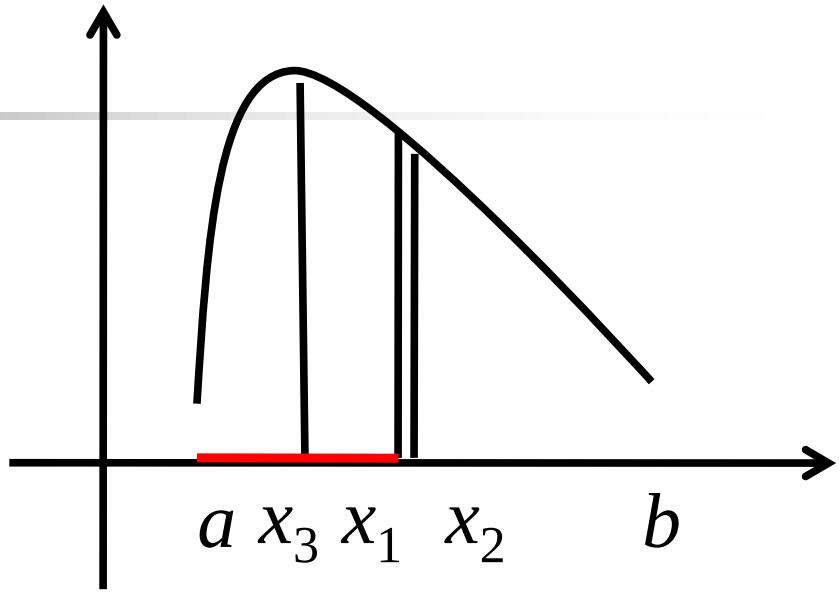
分治算法

- 单因素优选法: $f(x)$在$[a, b]$上先递增后递减，找出$\max f(x)$.
- idea:
- 选取点x_1, x_2
- If $f(x_1) > f(x_2)$, 舍去$[x_2, b]$
- If $f(x_1) < f(x_2)$, 舍去$[a, x_1]$
- If $f(x_1) = f(x_2)$, 舍去$[a, x_1]$和$[x_2, b]$



- 方案一：

$x_1 \approx x_2 \approx (a + b)/2$



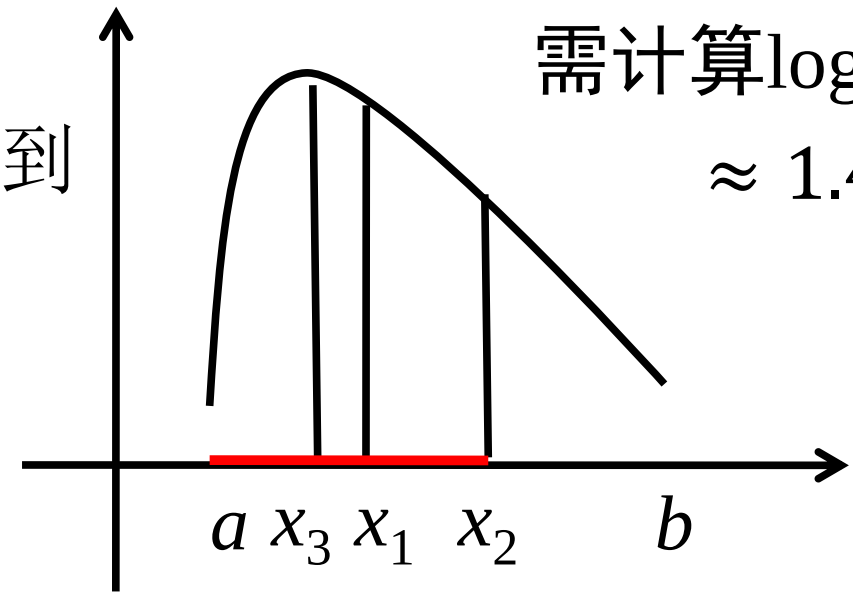
假定将$[a, b]$离散化成n个点

- $T(n) = T(n/2) + 2$
- $T(2) = 2$

需计算 $2\log_2 n$次函数值

- 方案二： $x_1 = ta+(1-t)b$, $x_2 = (1-t)a+tb$，其中 $t = \frac{\sqrt{5}-1}{2} \approx 0.618$

利用黄金分割找到最大值



需计算$\log_{1.618} n$次函数值

$\approx 1.44 \log_2 n$

乘法(1)

n^2次乘法运算

- 输入：$X=x_n x_{n-1}\ldots x_1$, $Y=y_n y_{n-1}\ldots y_1$
- 输出：$Z = XY$

- idea:

  $X = X_1 \times 10^{n/2} + X_2$,

  $Y = Y_1 \times 10^{n/2} + Y_2$

  $Z = XY =$

  $X_1Y_1 \times 10^n + (X_1Y_2 + X_2Y_1) \times 10^{n/2} + X_2Y_2$

$$\begin{array}{r} 123 \\ \times 321 \\ \hline 123 \\ 246 \\ 369 \\ \hline 39483 \end{array}$$

乘法(2)

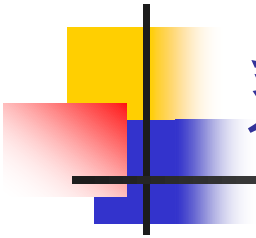
- 分别计算： $X_1Y_1, X_1Y_2, X_2Y_1, X_2Y_2$
- 乘法次数：$T(n) = 4\,T(n/2), T(1) = 1$
- $T(n) = O(n^2)$ **性能没有实质性提高**

乘法(2)

- $X_1Y_1 \times 10^n + (X_1Y_2 + X_2Y_1) \times 10^{n/2} + X_2Y_2$
- **计算：** X_1Y_1, X_2Y_2

$$X_1Y_2 + X_2Y_1 = (X_1 + X_2)(Y_1 + Y_2) - X_1Y_1 - X_2Y_2$$

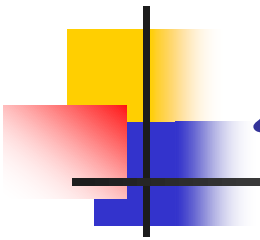
- 乘法次数：$T(n) = 3\,T(n/2), T(1) = 1$
- $T(n) = n^{\log_2 3} \approx n^{1.59}$



乘法(3)

- 乘法次数：$T(n) = 3\,T(n/2),\ T(1) = 1$
- $T(n) = n^{\log_2 3} \approx n^{1.59}$

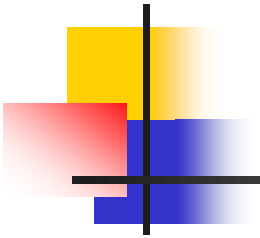
, $\mathbf{(X_1+X_2)(Y_1+Y_2)}$



思考题

- 能不能更快？

- 可以！快速傅里叶变换（FFT）
- 通过把X, Y各等分成3段，来演示FFT的思想



$$X = X_2 \times 10^{2n/3} + X_1 \times 10^{n/3} + X_0,$$

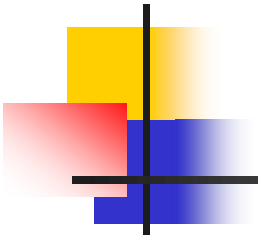
$$Y = Y_2 \times 10^{2n/3} + Y_1 \times 10^{n/3} + Y_0,$$

$$Z = XY =$$

$$X_2Y_2 \times 10^{4n/3} + (X_1Y_2 + X_2Y_1) \times 10^{n}$$

$$+ (X_0Y_2 + X_1Y_1 + X_2Y_0) \times 10^{2n/3}$$

$$+ (X_1Y_0 + X_0Y_1) \times 10^{n/3} + X_0Y_0$$

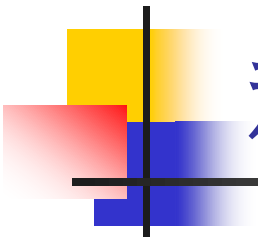


- 尝试一
  - 分别计算X_0Y_0, X_1Y_1, X_2Y_2, $(X_0+X_1)(Y_0+Y_1)$ $(X_0+X_2)(Y_0+Y_2)$, $(X_1+X_2)(Y_1+Y_2)$
  - $T(n) = 6T(n/3), T(1) = 1$
  - $T(n) = n^{\log_3 6} \approx n^{1.631}$
- 能否用更少的乘法次数实现

问题的“难”与“易”

- **算法复杂度(complexity)**：算法运行的总“步数”（时间）
  - 通常考虑在最坏的输入情况下
  - 例如：冒泡排序

- **问题的复杂度**：最优算法解决此问题的算法复杂度
  - 例如：基于比较的排序 **$O(n\log n)$**
  - 两个数相乘 $O(n^2)$, $O(n^{1.59})$, **$O(n\log n)$**



规约

- 假设A和B是两个计算问题，称可以从问题A**规约**到问题B (记做**$A \leq_P B$**)：

  如果任给一个求解B问题的算法，都可以“使用”此算法求解问题A

  A is “**easier**” than B

思考题

- 问题A：判定一个整系数多项式方程是否有整数解？
- 问题B：判定一个整系数多项式方程是否有非负整数解？

  例如：$x^3 + y^3 = z^3$，$x^3 + y^3 = z^3 + u^3$

- **证明：** $\mathbf{A} \leq_P \mathbf{B}$, $\mathbf{B} \leq_P \mathbf{A}$

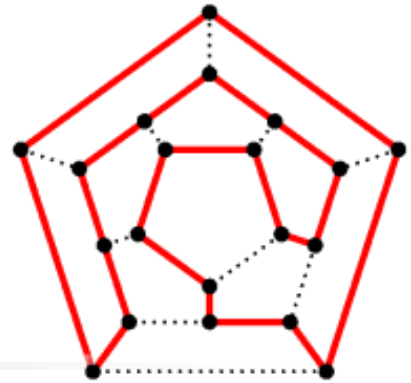
P vs NP 之 P

- 多项式时间（可求解）问题(**P**olynomial time)：存在某个能解决该问题的算法A，它的复杂度是O(n^c)，其中c是某常数
  - n是输入的规模
  - O(n), O(n^2), O(n^3), O(n^{10000}), O($n^{2^{100}}$)都是多项式时间
  - 多项式时间问题被认为是计算机能够有效解决的问题

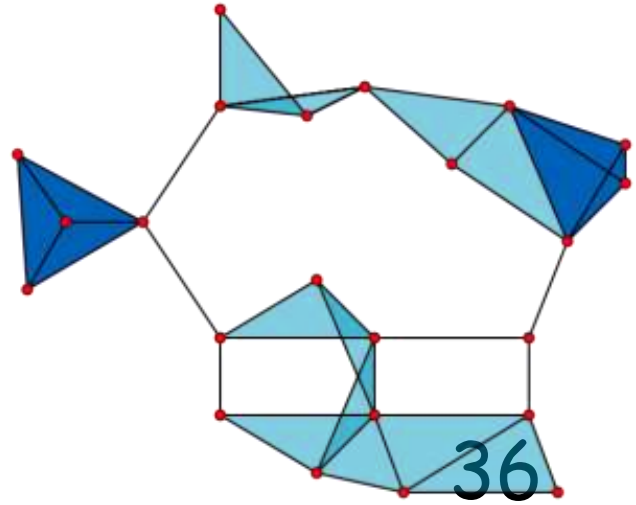
P vs NP 之 NP

- 多项式时间可验证问题(**NP,** Non-deterministic Polynomial time)：问题的“答案”可以在多项式时间内验证
  - 存在多项式时间的验证算法，对任何输入，如果答案是“正确”(接受)，那么存在证据使得验证算法能证明这一点；反之，一切证据都不能证明
  - 例如：一张地图是否可以进行3染色？
  - P的问题都属于NP, i.e. $P \subseteq NP$

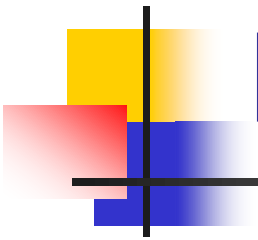
更多NP中的例子



- 给定布尔表达式ϕ，判定是否有一组赋值使得这个布尔表达式的取值为真？SAT
- 一张图是否存在Hamiltonian回路？
- 给定一张图及参数k，判断图里是否有k个点构成clique？



- 目前为止，不知道这些问题是否属于P！

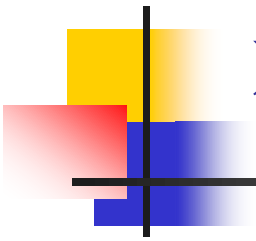


NP-完全

- SAT, Hamilton, k-clique都是NP-完全的
- NP-完全：NP中最“难”的问题
  - 所有其他的NP问题都可以规约到它
  - 如果找到了一个NP-完全问题的多项式时间算法，则所有NP问题都有多项式时间算法，即 P=NP

素数判定

- 给定正整数n，判定n是否是素数。
- 这个问题属于NP吗？
  - 是！
  - 存在g，$g^{n-1} \equiv 1 \ (mod\ n)$ 且对n-1的任何素因子p，$g^{(n-1)/p} \neq 1 \ (mod\ n)$
  - 证据：g，n-1的所有素因子*
    - 如何判断$g^{n-1} \equiv 1 \ (mod\ n)$？
    - 如何判断给出n-1的素因子都是素数？
  - Agrawal–Kayal–Saxena primality test，2002



有没有不在NP的问题？

- 给定n*n的棋盘，两个人下广义的围棋，先手是否必胜？
  - 目前为止，不知道在不在NP中
- 给定一个图灵机M和一个输入字符串x，判定M在x这个输入上是否能停机？
  - 不在NP中
  - 事实上，没有图灵机能判定这个问题。

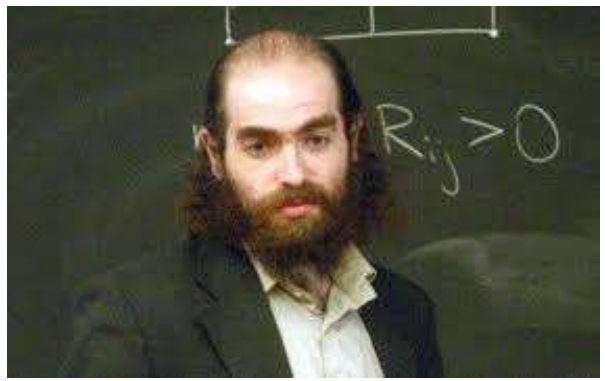
停机问题：不可判定

	1	2	3	4	5	6	7	8	9	…
1	0	0	0	0	0	0	0	0	0	…
2	1	1	1	1	1	1	1	1	1	…
3	0	1	0	0	0	0	0	0	0	…
4	0	0	1	1	1	1	1	1	0	…
5	1	0	0	0	0	0	0	0	0	…
6	1	1	1	1	1	1	1	1	1	…
7	1	1	0	0	1	1	1	1	1	…
…	.	.	.	.	.	.	.	.	.	.

- 假如存在图灵机H能判定
- 定义图灵机G
  - 对任何输入i,
  - 如果H(i,i) =0, 则停机.
  - 否则无限循环

Millennium Prize

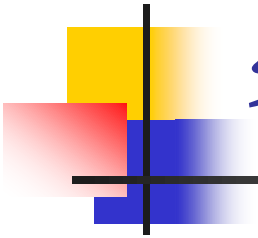
- Birch and Swinnerton-Dyer Conjecture
- Hodge Conjecture
- Navier-Stokes Equations
- **P vs NP**
- Poincaré Conjecture (solved)
- Riemann Hypothesis
- Yang-Mills Theory



First Clay Mathematics Institute Millennium Prize Announced: Prize for Resolution of the Poincaré Conjecture Awarded to Dr. Grigoriy Perelman

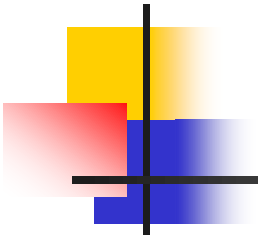
- 如果 P = NP

- 如果 P ≠ NP
  - **密码学！**



密码学

- 基于大整数分解的RSA算法
  - $n = p*q$ is HARD
  - Rivest, Shamir, Adleman (1979)
  - $\Phi(n) = (p-1)(q-1)$, $d \times e \equiv 1 \pmod{\Phi(n)}$
  - $E(M) = M^e$, $D(C) = C^d \pmod{n}$
- 基于离散对数的公钥加密算法
  - Discrete Logarithm is HARD
- 加密算法能用来做什么
  - 别人发给我的文件只有我能看
  - 别人不能仿造我来发文件
  - ……



谢谢！

计算机科学导论

计算系统思维

程序：计算过程的体现、载体

- 计算过程：**通过操作数字符号变换信息的过程**
- **数字符号**
  - 有名字
  - 有类型
  - 有值
  - 表示数据（名词）
  - 表示操作（动词）
    - 函数、方法
  - 可以组合
  - 可以分解

```
1  package main
2  import "fmt"
3  func main() {
4          var name string = “Yanyong Zhang"
5          var sum int = 0
6          var i int
7          for i = 0; i < len(name); i++ {
8                  sum = sum + int(name[i])
9          }
10         var sum_bytes [5]byte
11         var j int
12         for j = len(sum_bytes) - 1; sum != 0; j-- {
13                 sum_bytes[j] = byte(sum % 10) + '0'
14                 sum = sum / 10
15         }
16         var k int
17         for k = j + 1; k < len(sum_bytes); k++ {
18                 fmt.Printf("%c", sum_bytes[k])
19         }
20         fmt.Println()
21 }
```

系统思维将模块整合成为系统

解决“中间有一段很大的空白”难题

- **计算思维是交响乐**
- 同学们如何操作
  - 一定要看一遍教科书
  - 一定要自己做一遍编程练习

计算系统方法

- **构造计算系统有条理、有门道，不是随意**
  - 系统地（systematic）、而不是随意地（ad hoc）
    - 计算系统往往没有物理世界强加的限制，系统性更加重要
    - 产出一个系统整体，而不是一堆罗列
  - 系统性不是教条，不是用于阻碍创新
    - 既有系统性，又鼓励创新和多样性
    - 为什么计算系统设计者叫architect?
      - 为什么要有“计算机体系结构”（computer architecture）
- 近70年计算系统的发展
  - 做得好：支持数十亿用户，近百亿设备
  - 做的不好：多样性欠缺，一个病毒可毁坏百万设备

什么是计算系统思维？

- 计算过程必须在计算系统中运行
  - 计算系统包括**抽象计算系统**或**真实计算系统**
- 设计与理解计算系统的最大挑战：复杂性
- 系统思维方法
  - **通过抽象，将模块组合成系统，无缝执行计算过程**
    同义表达：**将系统分解成模块的组合（decomposition）**
    - 巧妙地定义抽象，将部件组合成系统，流畅地执行计算过程，提供应用价值
    - 模块就是精心设计的部件
  - 三个利器，即抽象化、模块化、无缝衔接
    - 抽象是最本质的考虑，模块化与无缝衔接是对抽象的补充
    - 系统思维方法本质上是抽象化方法（**abstraction**）

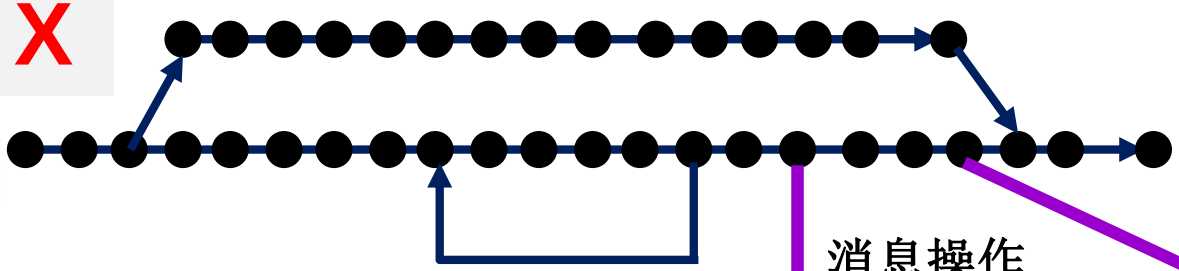
提纲

1. 计算过程与系统思维
    1. 应对复杂性挑战
    2. 抽象、模块、无缝衔接
2. 系统思维例子
3. 系统思维要点
    1. 抽象化
    2. 模块化
    3. 无缝衔接
4. 系统思维的创新实例

1. 计算过程与系统思维

- 计算过程在计算系统中运行
  - 计算系统执行计算过程
  - 通用系统：一个计算系统可执行万千计算过程
  - 专用系统：一个计算系统可执行一个或几个计算过程

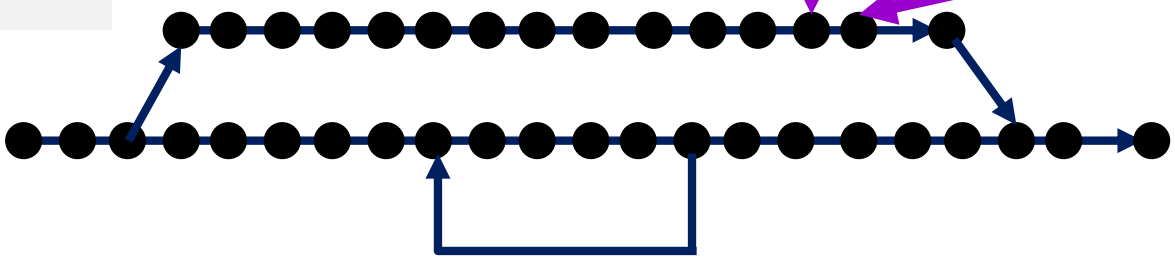
X

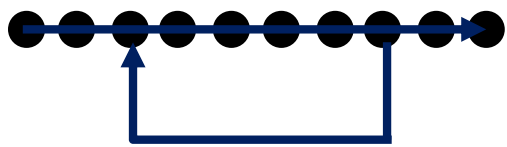


消息操作

Z

Y





从计算系统思维角度看到的一些计算机科学基本抽象

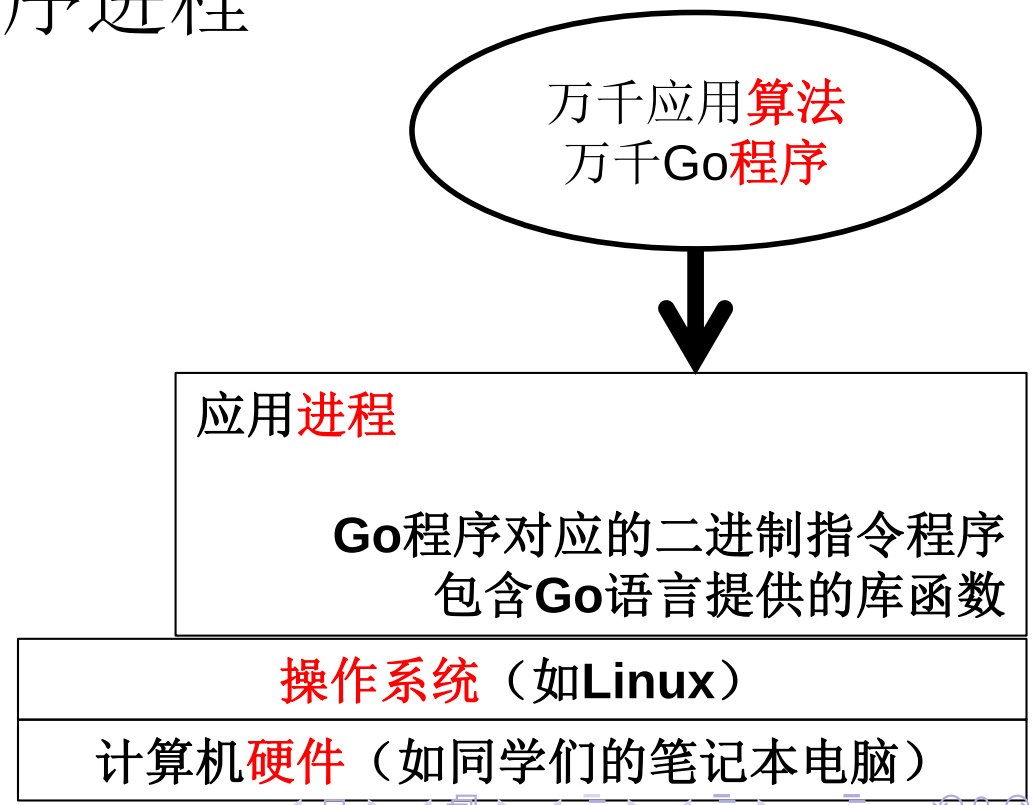
- 抽象使得计算机科学成为一门优美的领域
  - 本质：采用一种方法解决该层次的所有问题，应对系统的复杂性，**以不变的抽象应系统的万变**

数字符号	**比特、字节、四进制数、十六进制数、整数、数组、BMP 图像。**	
软件	**算法**	巧妙的信息变换方法。例如信息隐藏算法。
	程序	算法的代码实现。例如实现信息隐藏算法的 hide.go 程序。
	进程	运行时的程序。例如在 Linux 环境中的 hide 进程。
	指令	程序的最小单位，计算机能够直接执行。
硬件	**指令流水线**	每条指令都通过“取指-译码-执行-写回”四个操作组成的指令流水线得以自动执行。所有指令都由这一种机制执行，指令流水线由若干时钟周期组成。
	时序电路	等同于**自动机**，说明每一个时钟周期的操作。时序电路由组合电路与存储单元组成，理论上等同于**图灵机**。
	组合电路	实现二值逻辑表达式（**布尔逻辑表达式**）。

一套方法支持万千应用

- 程序员：应用算法→Go语言程序
- 编译器： Go语言程序→ 机器语言程序
- 计算机：执行二进制指令程序
- 操作系统：控制应用程序进程

- 万千应用，**一套方法**

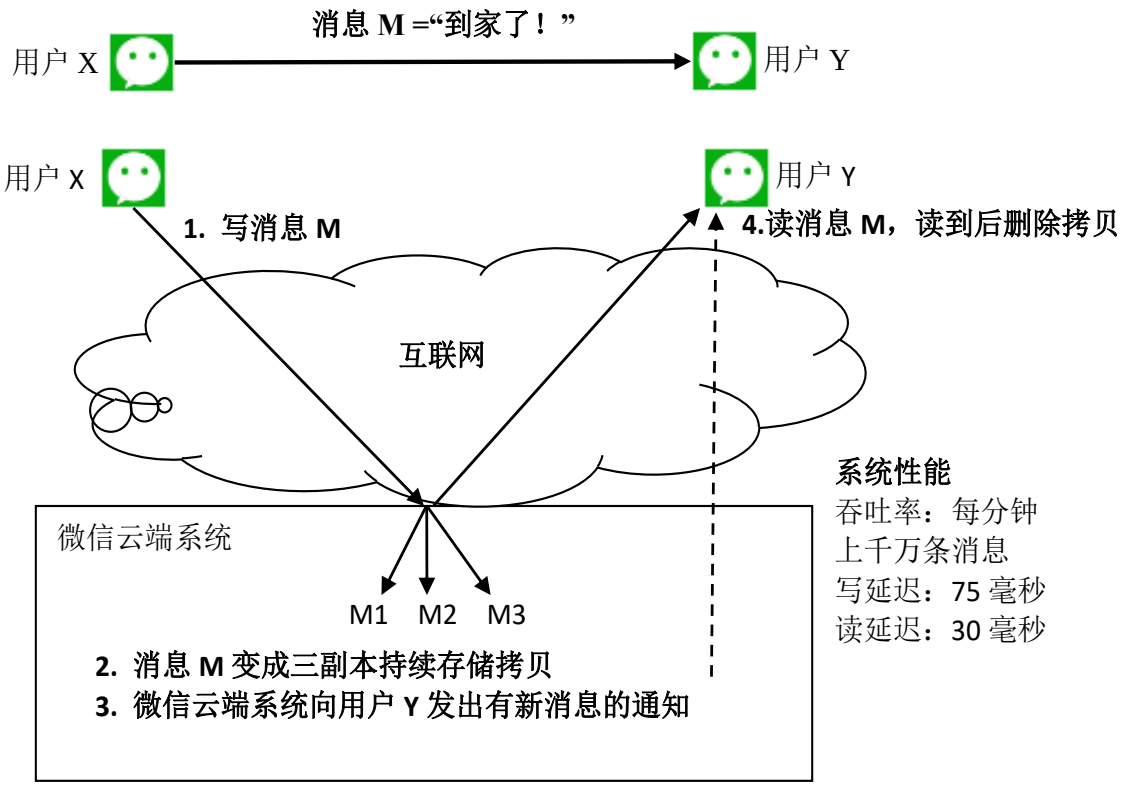


应对复杂性挑战

- 系统思维的主要目的是应对复杂性挑战
  - 微信系统比全世界的米级地图还要复杂的多
    - 多：微信系统涉及20亿亿个晶体管
    - 杂：多种应用需求、多种计算过程
    - 乱：不能靠随意堆砌20亿亿个晶体管解决问题
- **主要手段：抽象、模块、无缝衔接**
  - 系统思维的主要手段是**抽象化**（abstraction）
  - 强调针对信息的抽象：**数据抽象**与**控制抽象**
    - **强调“比特精准、可自动执行的抽象”**
  - 任何计算系统包含：**处理**、**存储**、**互连**三大子系统

2. 一个系统思维实例

- 不能靠简单地堆砌20亿亿个晶体管
  - 从多个层次（角度）理解一个系统
  - 每个层次仅考虑该层次特有问题，忽略其他问题
  - 用一套抽象概念和方法统一地处理该层次所有的计算过程，解决这些特有问题

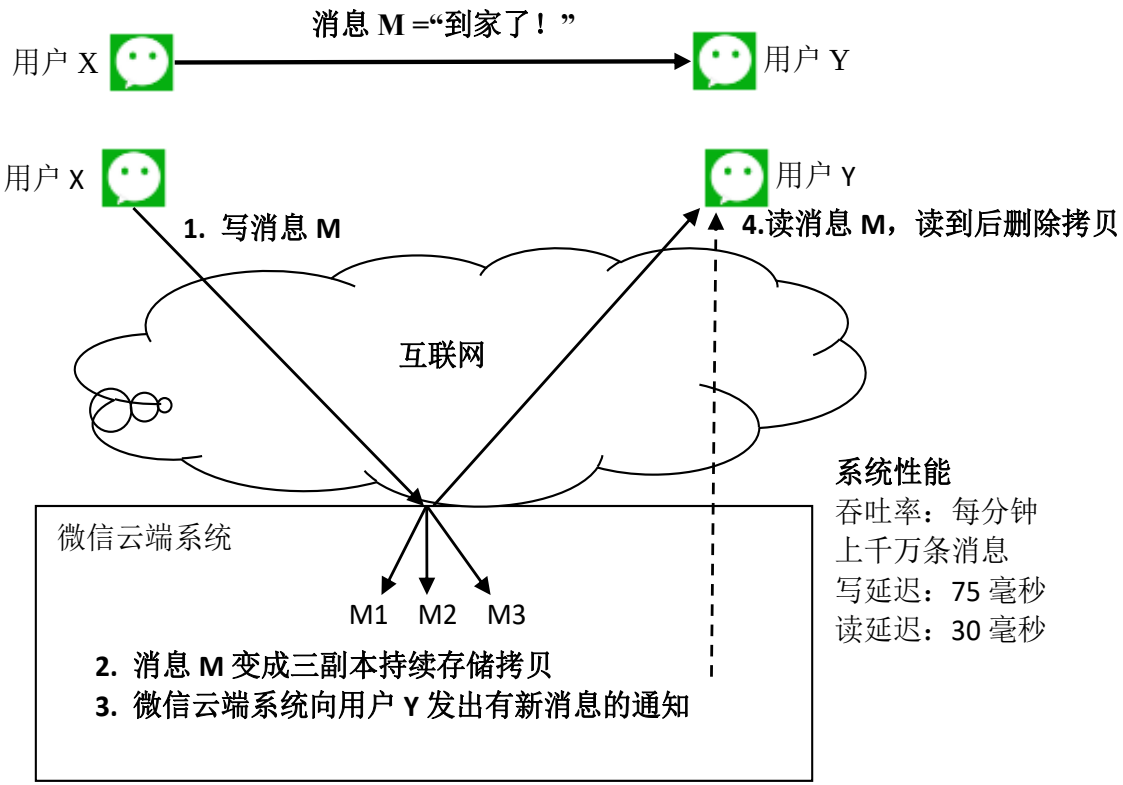


用户层抽象

抽象概念	用户 客户端设备 消息
计算过程	浏览消息 收消息 发消息
关心问题	如何浏览消息 如何收消息 如何发消息 消息可以含自拍视频吗 如何加好友 多久好友才能收到 什么样的终端设备 什么样的网络
忽略问题	消息不丢失、不发错

一个系统思维实例

- 不能靠简单地堆砌20亿亿个晶体管
  - 从多个层次（角度）理解一个系统
  - 每个层次仅考虑该层次特有问题，忽略其他问题
  - 用一套抽象概念和方法统一地处理该层次所有的计算过程，解决这些特有问题

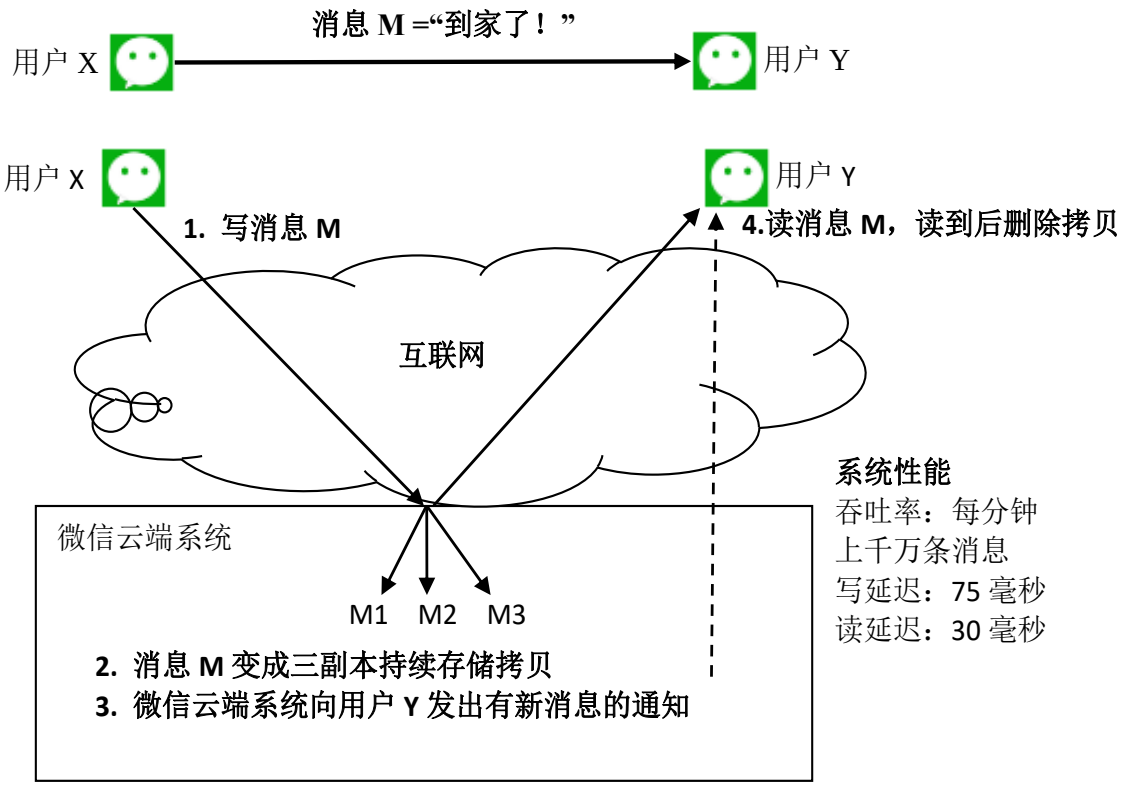


应用层抽象

抽象概念	用户、客户端设备、客户端应用软件、消息、微信云端系统
计算过程	编写并发送消息 向微信云端系统写消息 复制三份写入持续存储 发送“有新消息”通知 读取消息
关心问题	如何实现消息的处理和传递、如何保证消息不丢失、不发错，微信系统能够支持多少用户同时在线
忽略问题	计算过程如何由系统软件和系统硬件实现

一个系统思维实例

- 不能靠简单地堆砌20亿亿个晶体管
  - 从多个层次（角度）理解一个系统
  - 每个层次仅考虑该层次特有问题，忽略其他问题
  - 用一套抽象概念和方法统一地处理该层次所有的计算过程，解决这些特有问题



系统层抽象

抽象概念	客户端和云端系统所涉及的硬件、操作系统软件、数据库软件、应用框架软件
计算过程	具体的计算过程
关心问题	让微信系统有效地及时地服务上亿用户，每分钟传递上千万条消息、同时能够保证读/写一条消息的延迟分别控制在30毫秒和75毫秒，如何控制系统成本
忽略问题	更底层的系统软件与硬件的组成

3. 计算系统思维的要点

- 通过**抽象**，将**模块**组合成为系统，**无缝**执行计算过程
  - **抽象化**：一个通用抽象代表多个具体需求
  - **模块化**：系统由多个模块组合而成
    - 计算机 = 硬件 + 系统软件 + 应用软件
    - 全系统一致性；信息隐藏原理，接口概念
  - **无缝衔接**
    - 避免缝隙：
      - 扬雄周期原理、波斯特尔鲁棒性原理、冯诺依曼穷举原理
    - 重视瓶颈：阿姆达尔定律
      - 系统性能改进受限于系统瓶颈（针对某任务）
        - 加速比 = $1 / ((1-f)/p + f) \rightarrow 1/f$ 当 $p \rightarrow \infty$；f为系统瓶颈部分

3.1 计算系统抽象

- **抽象化**：一个通用抽象代表多个具体需求
  - 计算抽象（abstraction）：既是计算机科学最重要的方法，也是最重要的产物
    - 一个计算抽象是一个语义精确、格式规范的计算概念
- 硬件实例：一个处理器中的运算器应该如何设计，才能有一个通用处理器？
  - 科学计算、事务处理、文字处理 → 定点部件、浮点部件
  - 新的应用出现怎么办？
    - 图形图像与音视频等多媒体→新指令、GPU加速器
    - 人工智能应用兴起→寒武纪神经网络处理器、张量处理器（TPU）
- 软件实例：万千应用，万千算法，如何用统一方式执行？

抽象化的要点

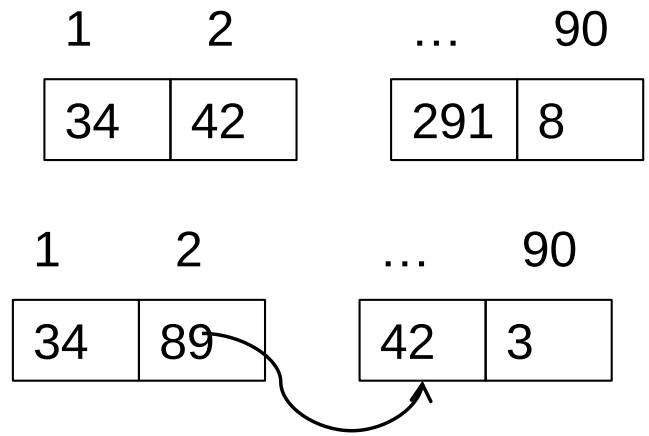
- 一个系统可从多个层次（或多个视野）理解
- 每个层次仅仅考虑有限的、该层次特有的问题
  - 并用一套精确规定的抽象概念和方法，统一处理该层次所有的计算过程，解决这些特有问题
- **抽象三性质：**
  - **有限性**：每个抽象仅仅考虑一个层次的有限的特有问题，忽略其他层次，忽略同一层次的其他问题
  - **精确性**：抽象化的产物是一个计算抽象，它是一个语义精确、格式规范的计算概念
  - **泛化性**：一个通用抽象代表多个具体需求
    - 可以触类旁通、用于其他实例（generalization）
    - 只对某个实例有效的抽象，不是好的抽象

计算机科学的抽象化特色

- 强调信息，即数据以及对数据的操作
  - 包括**存储操作、运算操作、通信操作（传、算、存）**
    - **存储操作：**将数据存放在某个地方（比如内存），或者从某个地方取出来放在寄存器中。该“地方”的名字称为存放该数据的**地址**。
    - **运算操作：**加、减、乘、除、与、或、非、移位等
    - **通信操作：**将数据从一个地方（比如硬盘）传递到另一个地方（比如内存）。数据传递到显示器时，相应的信息就被显示出来了。
  - **数据抽象：**一类数据及其操作
  - **控制抽象：**控制多个步骤组合实现计算过程
- **强调“比特精准、可自动执行的抽象”**

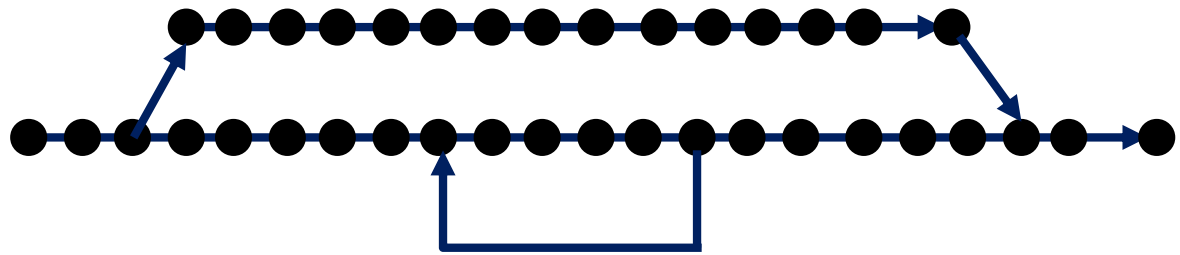
数据抽象（数据类型）

- 比特（bit）：基本二进制数位
- 4比特（hex）：16进制数位
  - 0:0000, 1:0001, 2:0010, …, 9:1001, A:1010,…, F:1111
- 字节（byte）：8位字节
  - 应用实例：ASCII码、Unicode码
- 字（word）：32/64位的整数、浮点数、字符串
- 指针（pointer）、数组（array）
- ……
- 文档、文件
- App



符号、操作、步骤、过程

- 一个过程是一个有限的步骤序列
  - 下图包含几个计算过程？
- 每个步骤包含“1个操作+1个下一步骤”
  - 下一步骤可是顺序或跳转
  - 操作可是：
    - 算术逻辑运算
    - 访存操作
    - 跳转操作
    - 输入输出操作
  - 它们都是数字符号变换操作

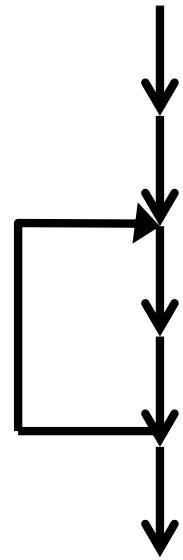


计算系统的环节

- 任何计算系统都有处理、存储、互连通信模块
- 任何模块抽象都要考虑数据、操作、控制三方面
  - 例子：如何应对输入输出（I/O）设备的多样性？
    - 鼠标键盘显示器,音箱麦克风,硬盘,打印机,网络WiFi
  - 回答：发明系统抽象
    - 操作系统
      - 驱动程序：字符设备、块设备
      - 文件概念：open, close, read, write；ioctl
    - 互连通信硬件接口
      - 串口、并口
      - PCI总线
      - USB总线

操作与控制

- 算术逻辑运算：+,-,×,÷； ∧,∨,¬； 移位
- 控制操作：跳转、条件跳转、同步、异常处理



- 存储操作：读、写、打开、关闭
- 元数据操作：创建、删除、访问控制

- 连接通信操作：socket

数据的
地址、内容、格式、语义

- 假设一段内存内容如右所示
  - 常见地址：byte address，word address
  - 内容：黄框里的值（十进制）
  - 格式与语义
    - 64位整数
      - 8028048
        6908526
        6383171
        7212509

控制抽象

- 三种常见的控制抽象（**顺序、条件跳转、调用**）
  - **顺序**（sequential order，或sequence）是最基本、最常用的、缺省的控制抽象
  - **跳转**（条件跳转）
  - **调用**一个子程序

步骤0：开始计算过程
步骤1：Sum=0
步骤2：i=0
步骤3：Sum = A[i] + B[i] + Sum
步骤4：i = i +1
步骤5：如果i<30亿，跳转到步骤3
步骤6：打印出结果Sum
步骤7：终止计算过程

- 这三种抽象可组合成更高级的控制抽象
  - 步骤2、3、4、5组合成新的步骤2（**循环**抽象）

步骤0：开始计算过程
步骤1：Sum=0
步骤2：for (i=0; i<30亿; i++) Sum = A[i] + B[i] + Sum;
步骤3：打印出结果Sum
步骤4：终止计算过程

3.2 模块化：计算系统与计算模块

- **模块化**：系统由多个模块组合而成
  - 计算机 = 硬件 + 系统软件 + 应用软件
    - 计算机硬件 = 处理器 + 存储器 + 输入输出（I/O）设备
      - 处理器 = 运算器 + 控制器 + 寄存器 + 数据通路 + …
        - ……
        - 最终到达计算机的基本操作
- 全系统一致性，不能相互矛盾
  - 不同的表示需要自动的翻译、解析
- 信息隐藏原理
  - 规范（specification）与实现（implementation）分离
  - 接口概念
    - 只能看到和使用模块接口，看不到模块内部

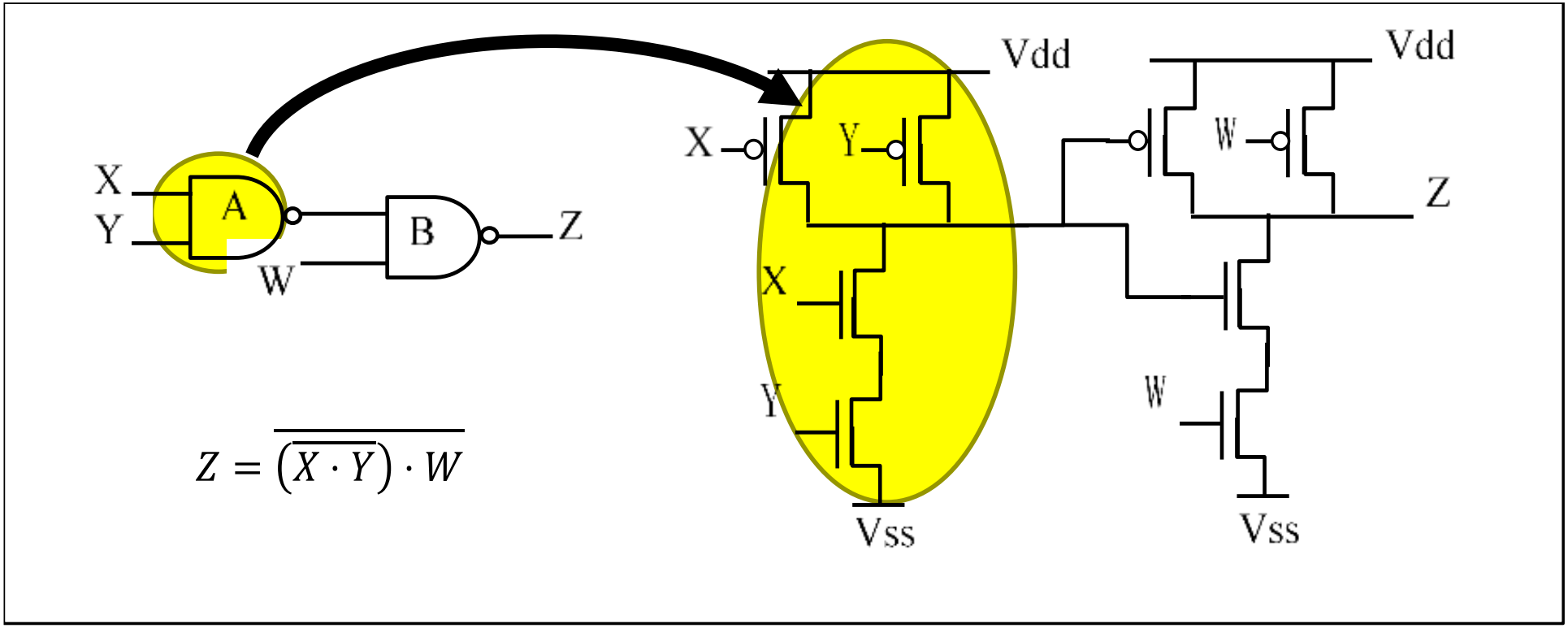
信息隐藏原理

- 精心定义模块的接口，将外界调用模块所需要的信息放在模块接口处，将外界调用模块不需要的信息放在模块内部隐藏起来
  - **隐藏内部信息（information hiding）**
    - 每个模块仅暴露其接口（interface）以及通过接口可见的外部行为，隐藏所有内部细节行为和内部信息
  - **区分规范与实现（separation of specification and implementation）**
    - 精确给出每个模块的规范（specification），即其接口和外部行为规定，独立于内部实现（implementation）
  - **抽象并重用模块**
    - 给出每种模块的抽象化描述，给予每种模块抽象特定的命名指称，并尽量重用模块的抽象

信息隐藏原理实例

W	X	Y	Z
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

- 图中三者表达同样的逻辑电路
- 但抽象不同，暴露的信息不同

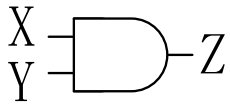


接口：逻辑值+逻辑操作　　电压值+晶体管操作

用真值表定义基本逻辑运算门

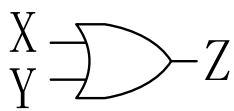
- 基本逻辑运算
  - 与：

X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1



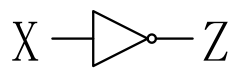
  - 或：

X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1



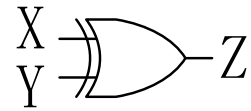
  - 非：

X	Z
0	1
1	0



  - 异或：

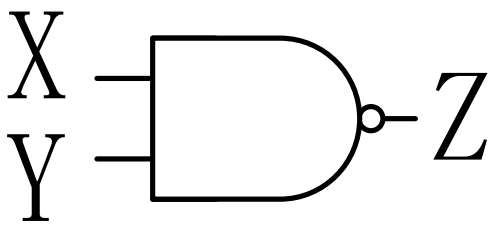
X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

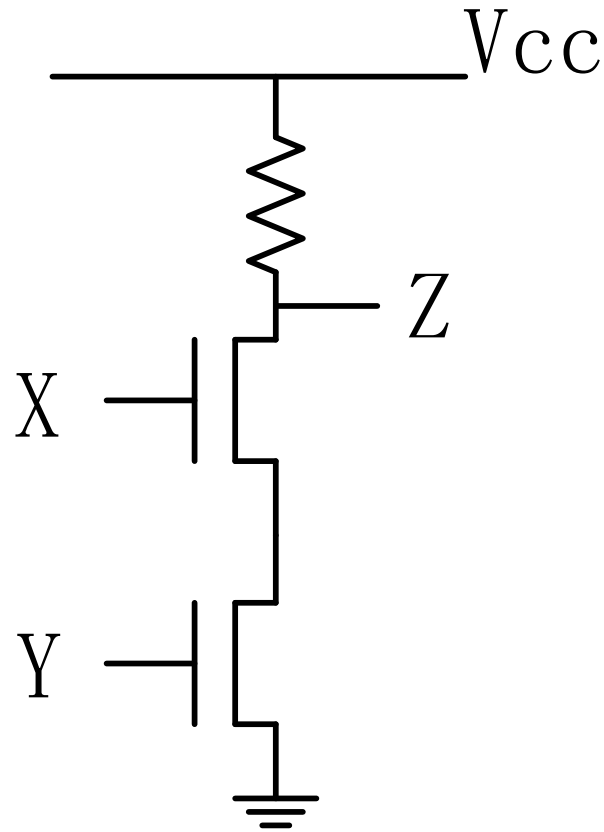


- 基本逻辑运算的组合可产生任意**组合逻辑**表达式
- 真值表可定义任意**组合逻辑**表达式

用晶体管电路实现基本逻辑运算
与非(NAND)门例子

X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0

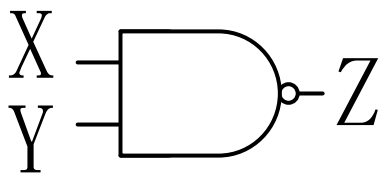




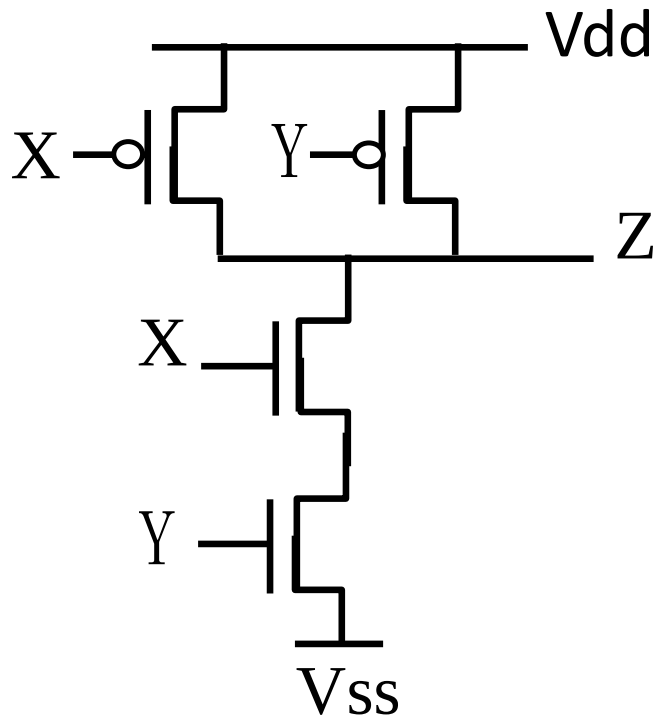
用晶体管电路实现基本逻辑运算
与非门例子

与非门

X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0



CMOS 电路



任意组合电路
都可以通过组合基本门电路实现

- 理论依据
  - 第二章的析取范式、合取范式定理

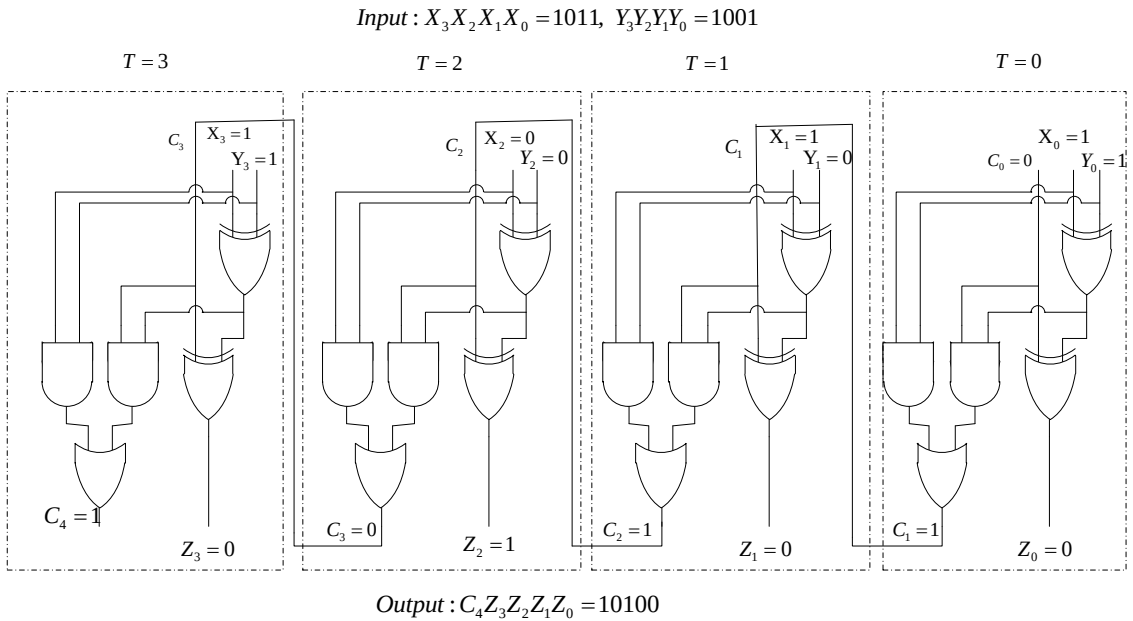
逻辑门名称	两个输入变量表达式	n 个输入变量表达式
与门	$Z = X \cdot Y$	$y = x_1 \cdot x_2 \cdots x_n$
或门	$Z = X + Y$	$y = x_1 + x_2 + \cdots + x_n$
异或门	$Z = X \oplus Y$	$y = x_1 \oplus x_2 \oplus \cdots \oplus x_n$
与非门	$Z = \overline{X \cdot Y}$	$y = \overline{x_1 \cdot x_2 \cdots x_n}$

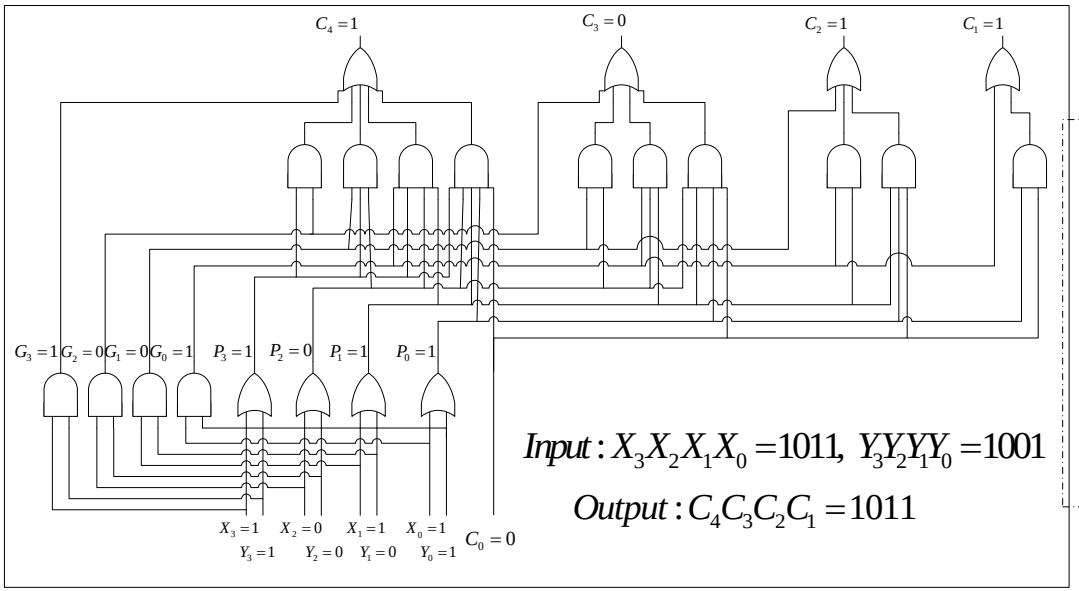
二进制加法例子：11+9=20 → 1011+1001=10100

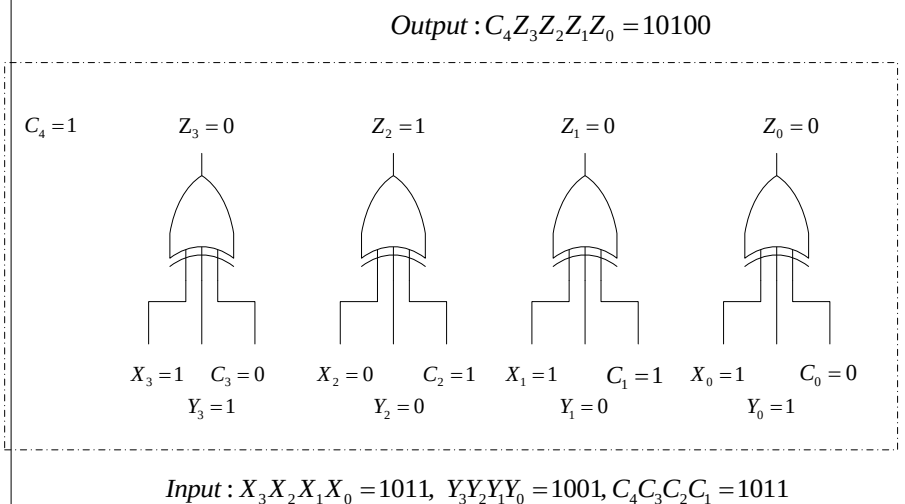
$X_3X_2X_1X_0$		1	0	1	1
$Y_3Y_2Y_1Y_0$	+	1	0	0	1
C_1				1	0
C_2			1	0	
C_3		0	1		
C_4	1	0			
$Z_3Z_2Z_1Z_0$	1	0	1	0	0

二进制加法实现：11+9=20 → 1011+1001=10100

- 全加器+波纹进位
  - 3*4 = 12门延迟
- 优化加法器
  - 3+1=4门延迟

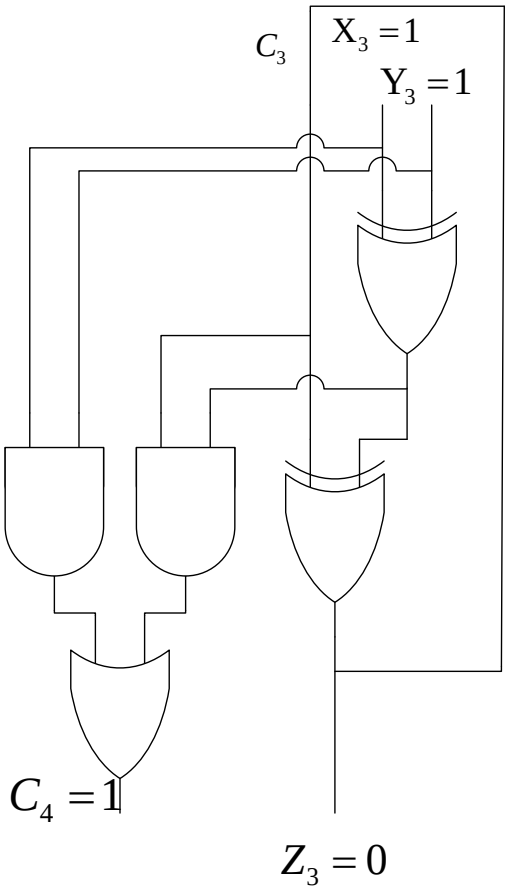




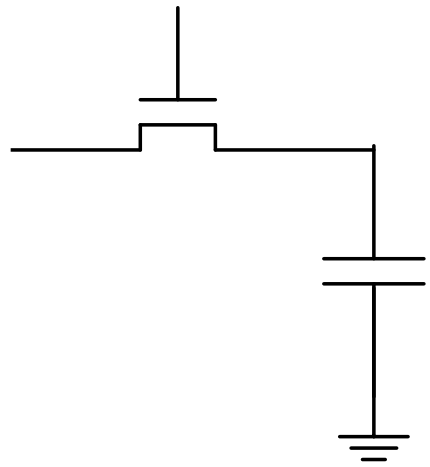


包含状态的电路称为时序逻辑电路（如何表示状态？）

- 有回路的电路

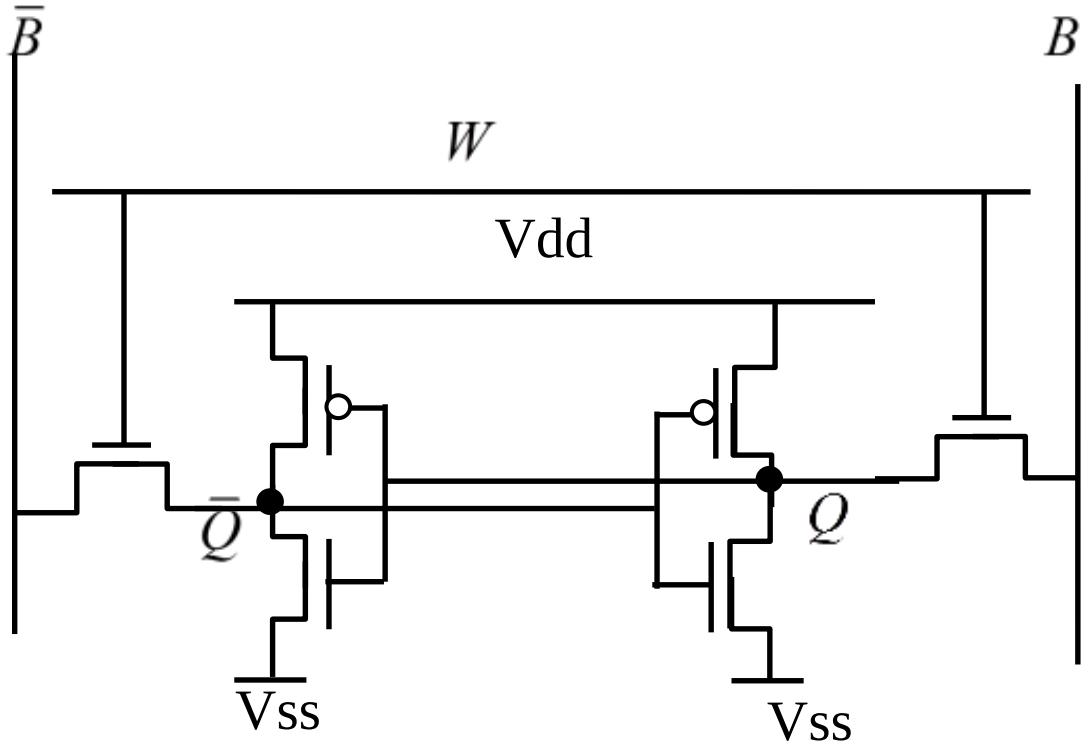


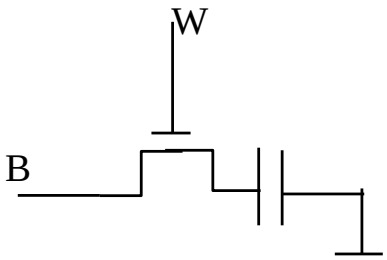
- 具有存储信息功能物理器件（如电容）
  - DRAM 单元



静态存储器单元
SRAM单元

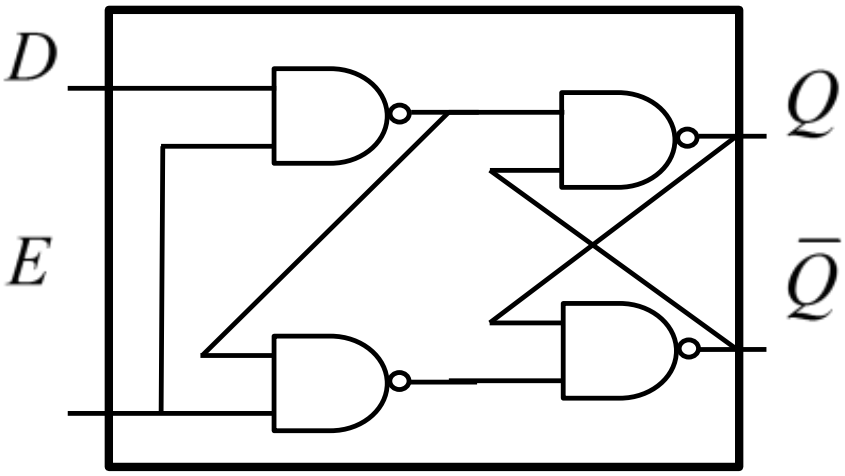
- 用六个晶体管组成1比特存储单元
- 为什么叫静态
- 三种存储器
  - DRAM
  - SRAM
  - NVM





D-触发器

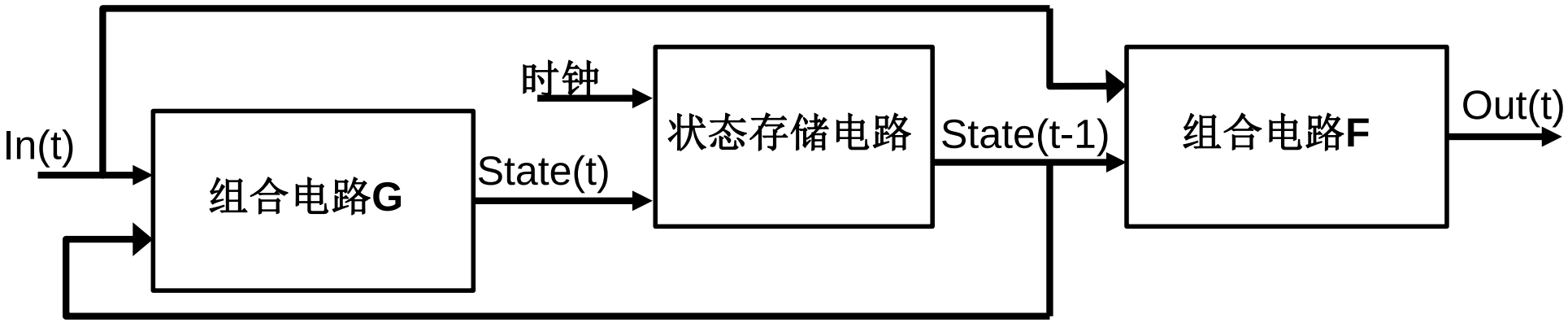
- Delay Flip-Flop
- Enable往往用时钟信号
- D: set/reset



E	D	Q	Q_{next}	
0	0	0	0	
0	0	1	1	
0	1	0	0	No change
0	1	1	1	
1	0	0	0	
1	0	1	0	Reset
1	1	0	1	
1	1	1	1	Set

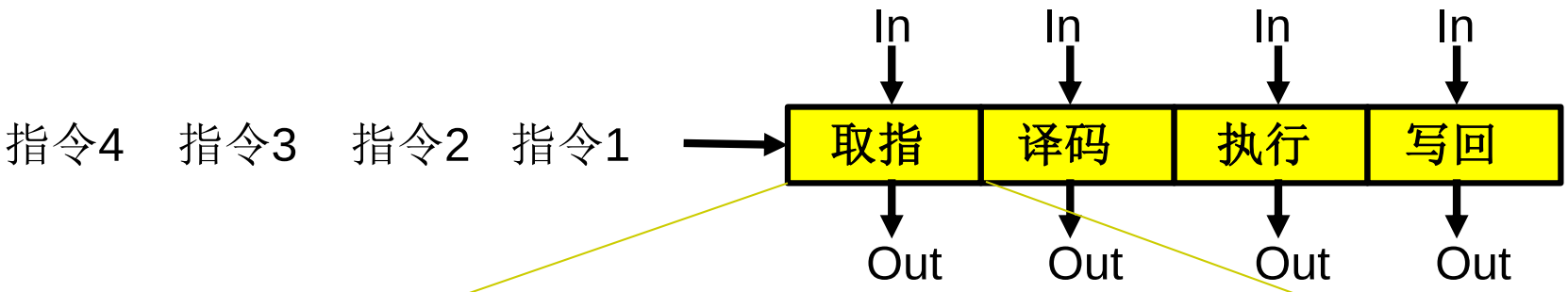
组合电路与状态电路产生自动机
即时钟同步的时序电路

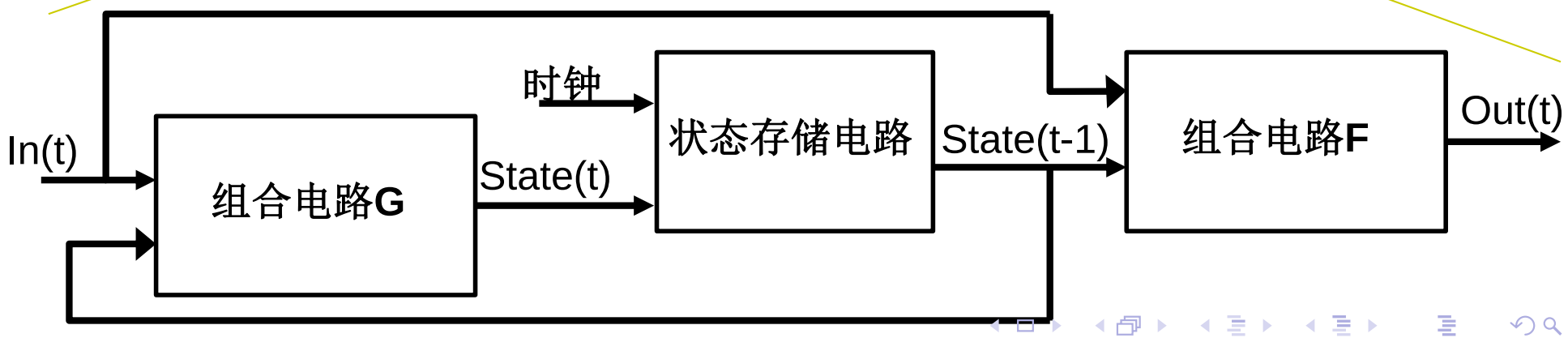
- 第t时刻的输出是t时刻输入与t-1时刻状态的函数
- 第t时刻的状态是t时刻输入与t-1时刻状态的函数
  - Out(t) = F(In(t), State(t-1))
  - State(t) = G(In(t), State(t-1))



自动机、时序电路是系统基本概念几乎无处不在

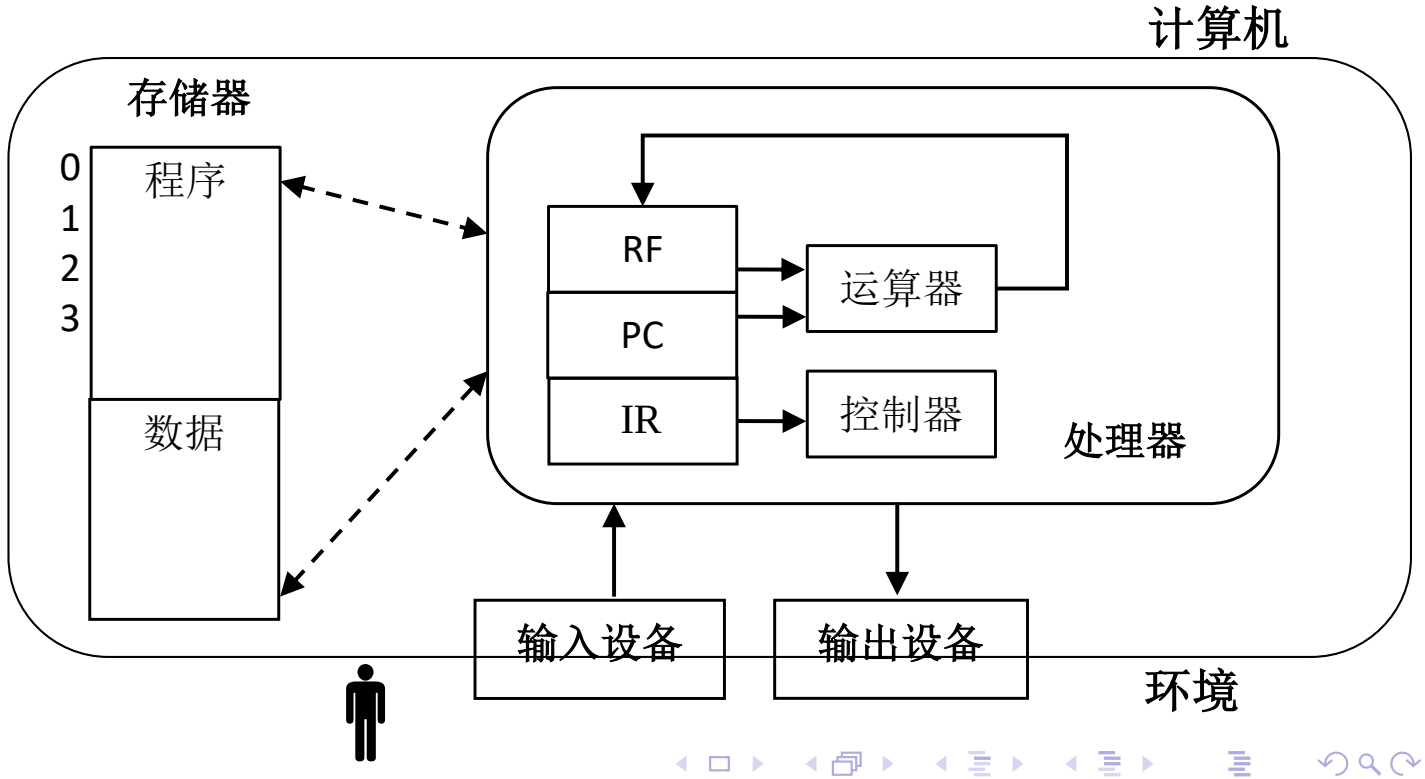
- 例如，任何程序、所有指令都是由指令流水线执行
  - 指令流水线是时钟同步的时序电路



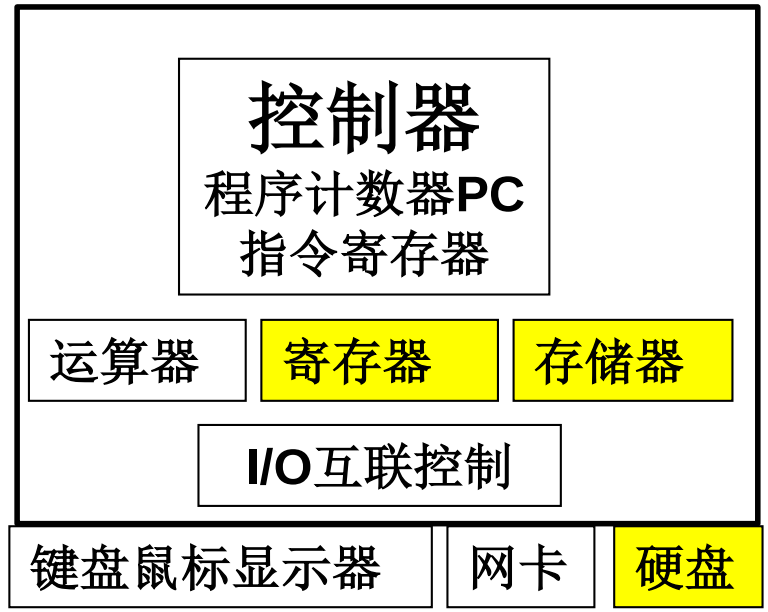


存储程序计算机（冯诺依曼模型）

- 二进制数据及其算术逻辑操作
- 计算机 = 处理器 + 存储器 + 输入输出设备
  - 处理器 = 运算器+控制器+寄存器
  - 冯诺依曼瓶颈（von Neumann bottleneck）
- 存储程序计算机（stored program computer）
- 串行执行



软件与硬件的配合



- 开机序列干什么？
- 如何求10亿对实数的内积？
  - $S=\sum_{k=0}^{n-1}(x_k\times y_k), n=10^9$
- 算法与程序
  - 假设程序与输入数据已在存储器中

$$S=0; k=0$$

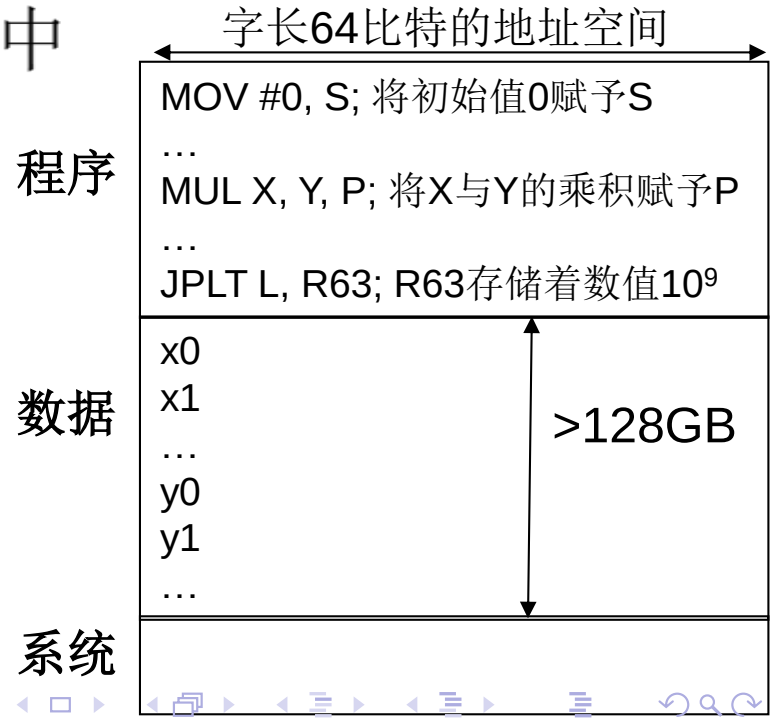
L: $P=x[k]\times y[k]$

$$S=S+P$$

$$k=k+1$$

$$Jump\ L\ if\ k<10^9$$

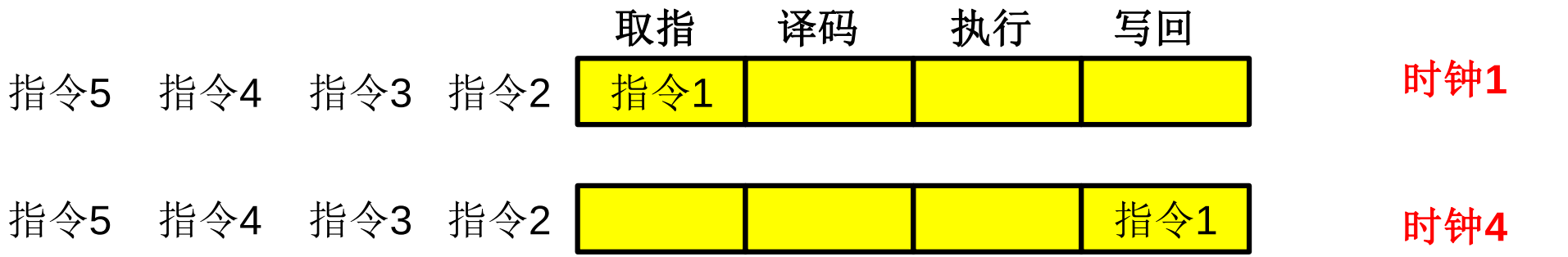
Print S



指令流水线

- 每条指令的执行都包含四个操作阶段（stage）
  - 取指操作：IR ← M(PC)
  - 译码操作：Signals = Decode(IR)
  - 执行操作：Result ← Op, 或 Address ← Op
  - 写回操作：
    - RF(i) ← Result,
    - RF(i) ← M(Address) 或 M(Address) ← RF(i)
- 执行的同时，PC←PC+1；周而复始

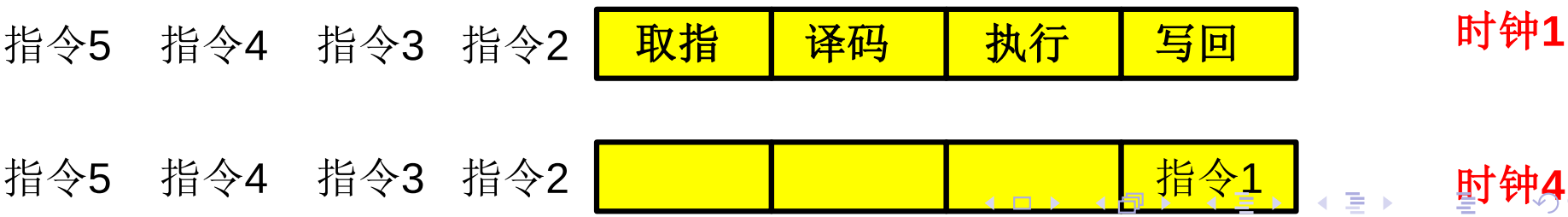
流水线延时
= 1纳秒
处理器主频
= 1 GHz



扬雄周期原理实例：指令流水线

- 每条指令的执行都包含四个操作阶段（stage）
  - 取指操作：IR ← M(PC)
  - 译码操作：Signals = Decode(IR)
  - 执行操作：Result ← Op, 或 Address ← Op
  - 写回操作：
    - RF(i) ← Result,
    - RF(i) ← M(Address) 或 M(Address) ← RF(i)
- 执行的同时，PC←PC+1；周而复始

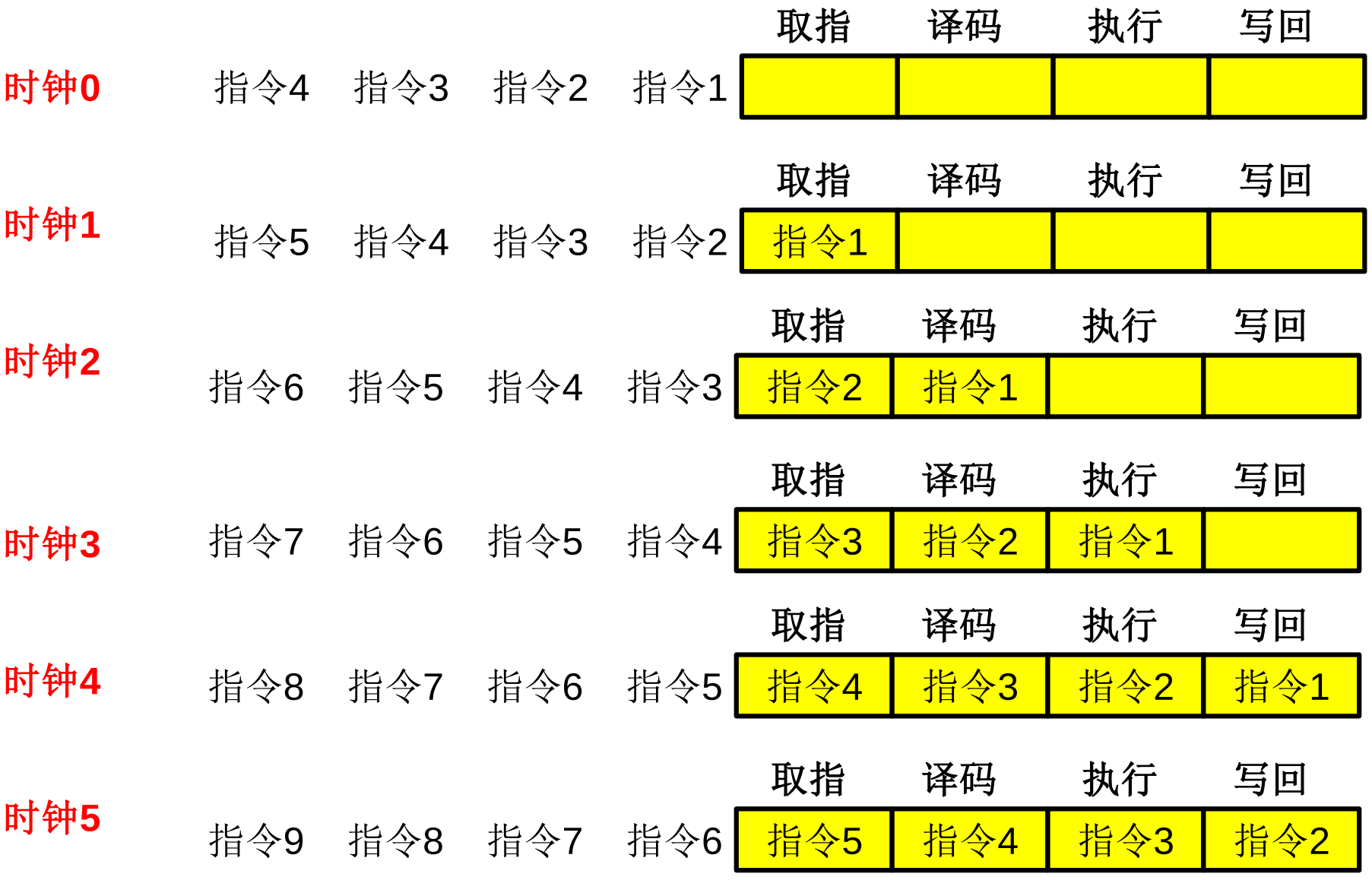
流水线延时
= 1纳秒
处理器主频
= 1 GHz



指令流水线

流水线延时= 1纳秒，处理器主频= 4 GHz

假设数据和指令都在缓存中，速度是4 GOPS（或4 GFLOPS）



冯诺依曼穷举原理

来源： First Draft of a Report on the EDVAC

- 人们必须事先给计算机全面的指示，绝对穷举所有细节（“in absolutely exhaustive detail”），使得计算机能够自动处理所有情况
  - 这些细节包括：
    - 计算机需要执行的程序的指令、程序的输入数据、程序需要的函数等
  - 其他实现细节包括：
    - 计算机开机后执行的第一条指令
    - 计算机正常执行程序时，下一条指令是什么
    - 执行程序出现异常时，有哪些异常，每种异常如何处理

开机后执行的第一条指令

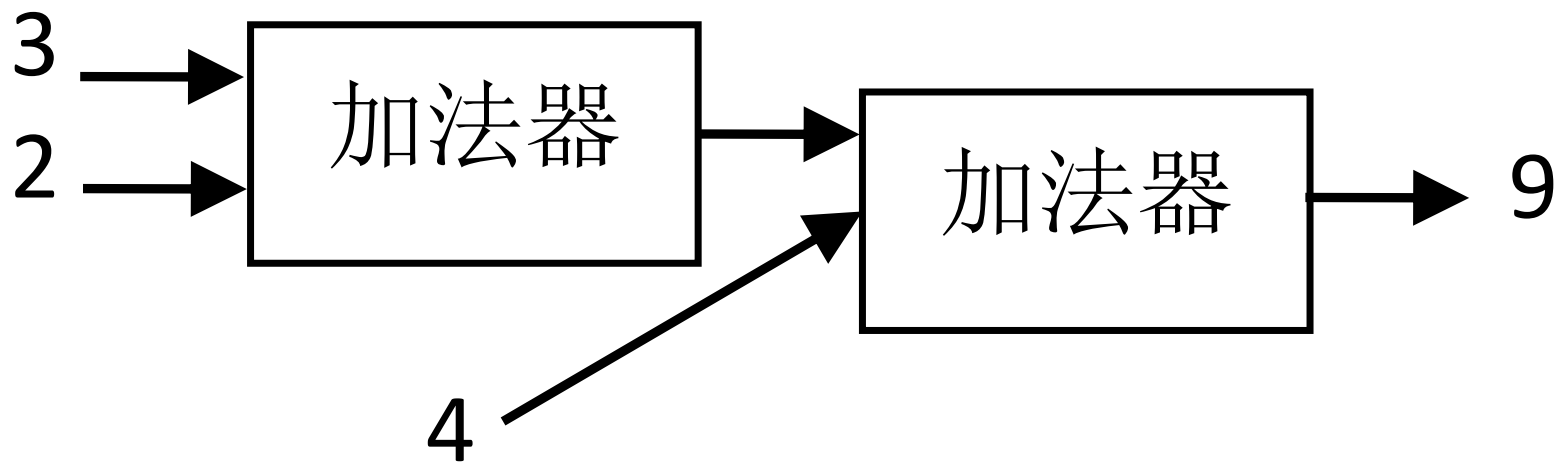
- x86计算机开机后执行的第一条指令
  - 位于地址FFFFFFF0（0xFFFFFFF0）
  - 其内容是一条跳转指令JUMP 000F0000
    - 最底层的一个系统软件（称为BIOS，即Basic Input-Output System）的第一条指令，称为BIOS入口地址
- 龙芯计算机开机后执行的第一条指令
  - 位于地址FFFFFFFFBFC00000
  - 其内容是一条特殊的赋值指令，将处理器的状态寄存器复位（置为零，清零），这也称为**初始化**处理器的状态寄存器

下一条指令是什么
下一条指令在哪里

- 三种方式
  - 哈佛马克一号（Harvard Mark I）
    - 学名是“Automatic Sequence Controlled Calculator”（自动顺序控制计算机）
    - “直线程序”， 没有跳转指令，不支持循环控制
  - ENIAC改造后
    - 在每一条指令的内容中，明确地列出下一条指令的地址
  - 今天的计算机系统
    - 采用“程序计数器”（program counter，即PC）方式

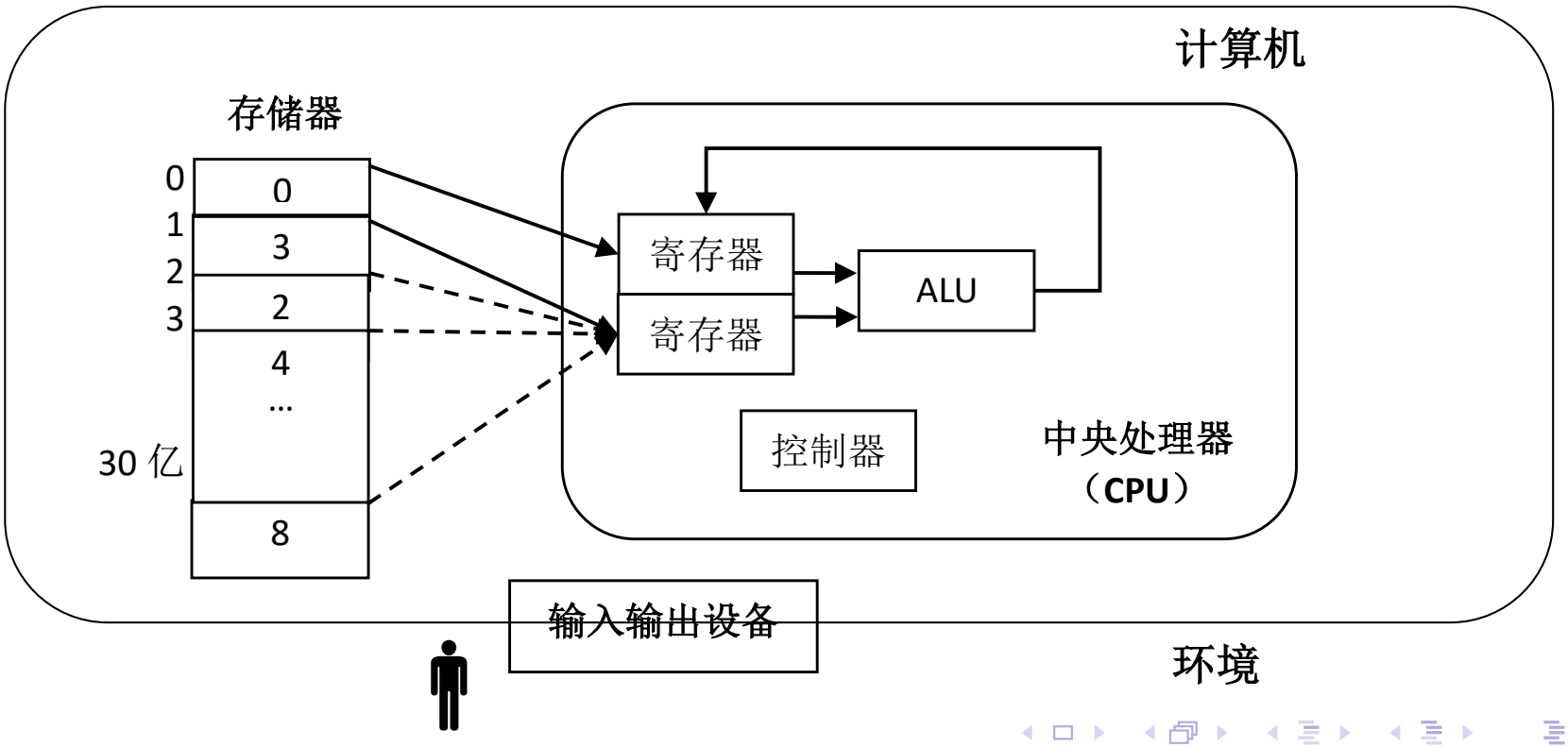
如何求3个数之和？

- 直接用硬件逻辑电路实现
- 步骤可以是硬件

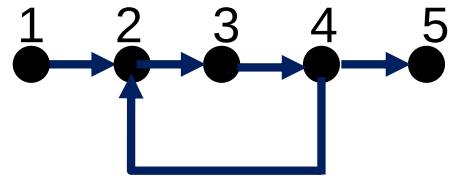


如何求30亿个数之和？

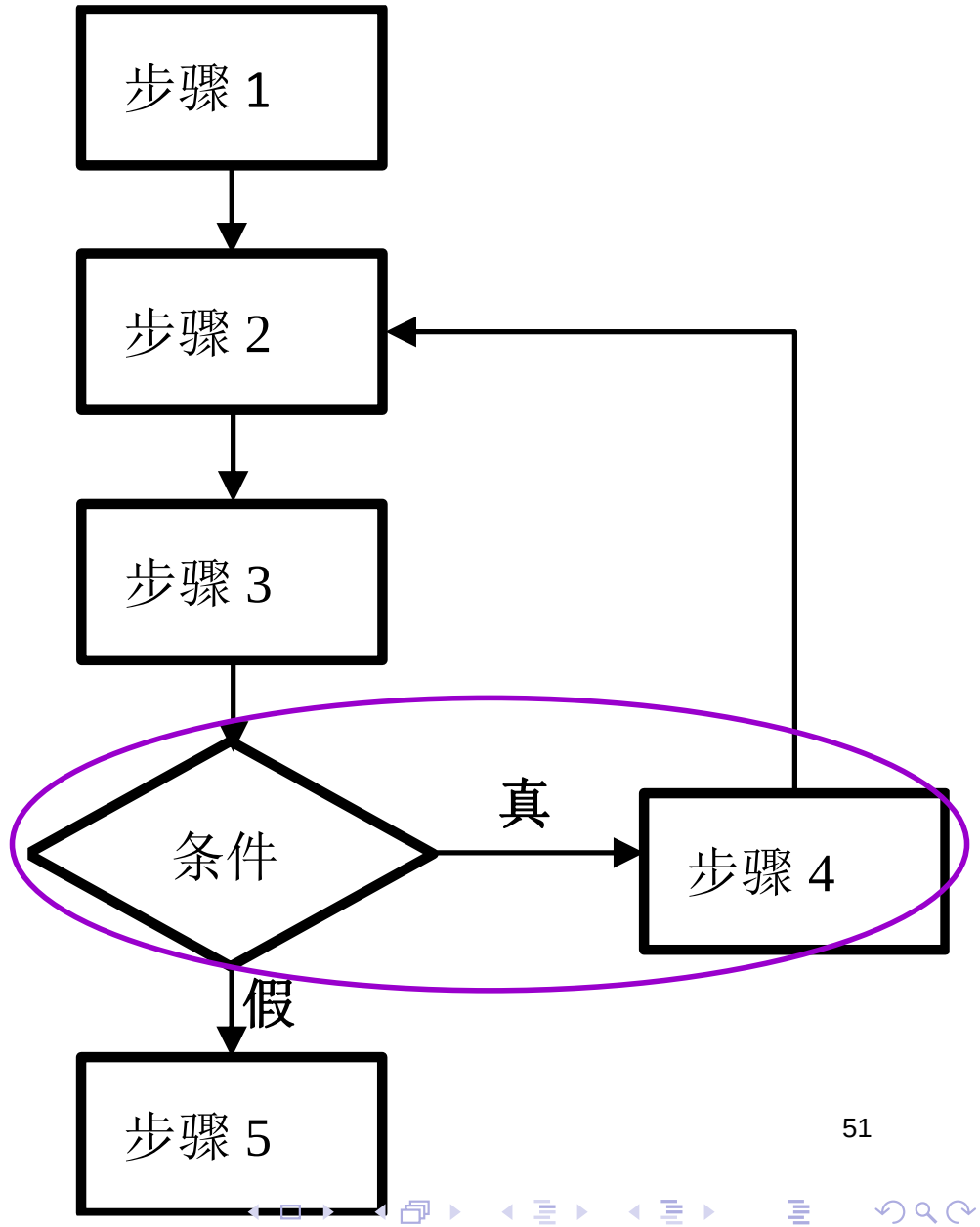
Step 1（第一步骤）：将存储器地址0里的数取到第一个寄存器
Step 2（第二步骤）：将存储器地址1里的数取到第二个寄存器
Step 3（第三步骤）：将两个寄存器的数在ALU相加，结果送到第一个寄存器
Step 4（第四步骤）：如果第二步的地址<30亿，将其地址增1，回到第二步骤
Step 5（第五步骤）：将第一个寄存器的数存入存储器地址0



用流程图描述计算过程

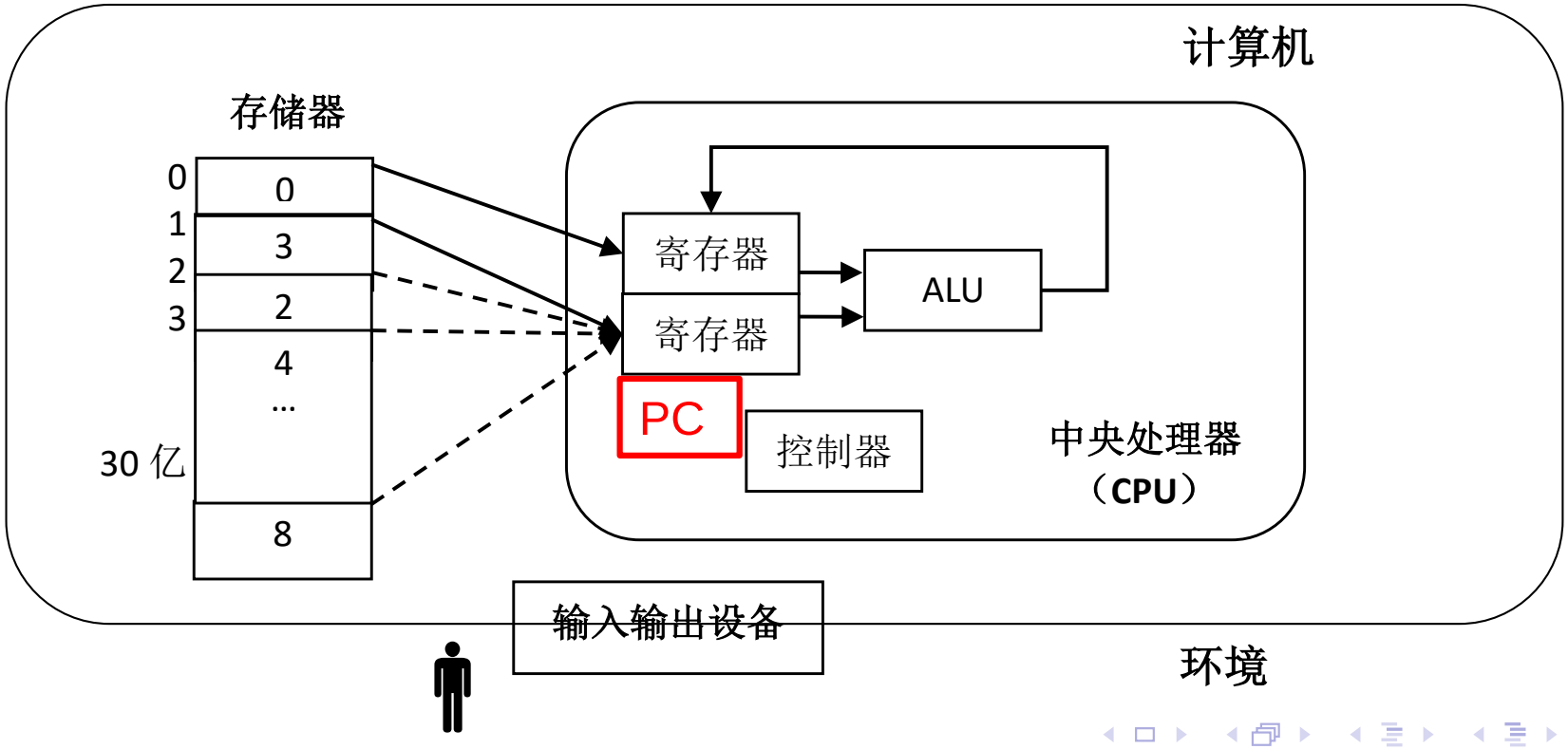


- Step 1（第一步骤）
  - 将存储器地址0里的数取到第一个寄存器
- Step 2（第二步骤）
  - 将存储器地址1里的数取到第二个寄存器
- Step 3（第三步骤）
  - 将两个寄存器的数送到ALU相加，结果送到第一个寄存器
- Step 4（第四步骤）
  - 如果第二步的地址<30亿，将其地址增1，回到第二步骤
- Step 5（第五步骤）
  - 将第一个寄存器的数存入存储器地址0



计算过程自动执行的要点：PC机制

Step 1（第一步骤）：将存储器地址0里的数取到第一个寄存器
Step 2（第二步骤）：将存储器地址1里的数取到第二个寄存器
Step 3（第三步骤）：将两个寄存器的数在ALU相加，结果送到第一个寄存器
Step 4（第四步骤）：如果第二步的地址<30亿，将其地址增1，回到第二步骤
Step 5（第五步骤）：将第一个寄存器的数存入存储器地址0



PC =
Step 2
Step 3
Step 4
Step 2
Step 3
Step 4
…
Step 2
Step 3
Step 4
Step 5

正常执行与异常处理

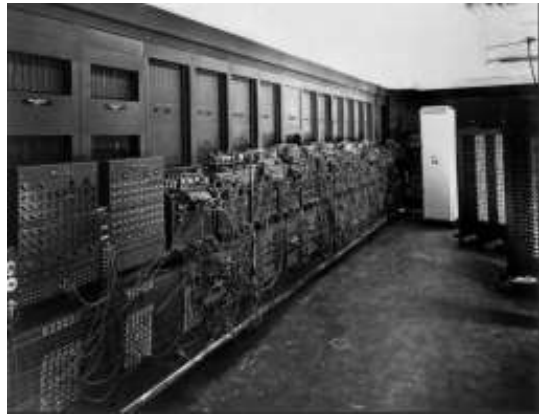
- 正常执行
  - 第一条指令在哪里？
  - 当前指令如何执行？
  - 下一条指令在哪里？
- 异常处理
  - 中断
    - 执行完毕当前指令，然后执行异常处理程序
  - 硬件出错
    - 立即执行一个事先设计好的异常处理程序
  - 保底异常
    - 为了做到穷举，保底异常（通常被称为machine check）覆盖其他异常没有覆盖的情况

重视瓶颈：阿姆达尔定律

- 系统性能改进受限于系统瓶颈
  - 加速比 = 1 / ((1-f)/p + f) →1/f 当 p→∞
    - “假如一个系统可以分成两部分X和Y, X+Y=1, 0≤X≤1, 0≤Y≤1。Y能够改善（即缩小它的数值），X不能被改善（即X是瓶颈）。那么，系统最多能被改善到1/X。”
  - 一台电脑使用了500 MHz主频的处理器芯片，假设X＝Y＝0.5，即程序代码只有一半可以随处理器速度的增加而改善
    - 那么，即使我们将处理器的速度提高1000倍（提到500 GHz）、1万倍、甚至无穷大，整个电脑的速度也最多只能变到1/0.5=2，即提高一倍

ENIAC

- 1943年初，美国陆军出现弹道计算迫切需求
- 宾州大学“电子计算机”概念、立项、实施
  - 1941年，毛捷利与艾克特开始合作
    - 1942年，毛捷利备忘录：《高速真空管器件的计算用途》
  - 1943年4月2-9日，正式立项
  - 1943年5月31日，ENIAC项目正式启动
  - 1945年11月，ENIAC研制成功
- ENIAC的主要特征
  - 数字计算机：而不是模拟计算机
  - 电子计算机：所有运算操作用电子速度执行
  - 足够的可靠性（用了10年）：18000个真空管的大系统



IBM 360

- 1960年，IBM总裁华森启动战略性业务调整
  - 全球计算机使用量数千台
    - 8+6款晶体管电脑+X款真空管电脑，各不相干
- 1960-1962.12，IBM 360总体方案出炉
  - 通用计算机 + 计算机家族 + 兼容性
  - 计算机体系结构 + 系统实现，两者的分离
  - 1963.1，IBM总部批准实施
- 应对普遍而剧烈的反对意见
  - 市场竞争力：单一系统太冒险、易被复制
  - 改软件、改外设：成本太大、时间太长
    - 解决问题的技术：自动翻译、软件模拟、硬件仿真
- 1964.4，IBM发布S/360，1965年发货

IBM 360经验（如何鼓励创新）

- 三种创新
  - 新问题：兼容性问题
    - 设计一代具有竞争力的电脑，改变多个互不兼容的产品线带来的软件移植、市场、研究开发、生产、管理的杂乱和低效率状况。
  - 新概念：通用性（360）和电脑家族
    - 从根本上解决兼容性问题
      - 效果（What）：从汇编语言和外部设备的角度看，家族成员都是一样的（相互兼容）
      - 措施（How）：家族成员有同样标准的指令系统、地址格式、数据格式和与外部设备的接口
  - 新指标：“大内存”（提高100倍）等，而不是“核能”
- 应对致命弱点（knockoff）：动态寻址

个人电脑

- 1973, Xerox PARC Alto
- 1974, MITS Altair 8800
  - 1975, MS BASIC解析器
- 1976, Apple-I, (1979, VisiCalc)
- 1981, IBM PC
  - Intel CPU, MS DOS,主板总线, BIOS
- 1996, 联想SpeedEasy无跳线主板

Worldwide PC Vendor Unit Shipment Estimates (Wiki)

Source	Date	Lenovo	HP	Dell	Acer Inc.	Asus	Others
IDC	Q2,2014	19.6%	18.3%	14.0%	8.2%	6.2%	33.6%
Gartner	Q2,2014	19.2%	17.7%	13.3%	7.9%	6.9%	35.0%

操作系统创新实例

- 1965，MIT Project MAC, Multics
  - Time sharing, file system, process, shell
- 1972，Unix
  - 可移植性：Unix及其软件工具支持很多厂家的计算机
    - 用C语言实现Unix及其软件工具
  - 灵活性：允许用户和编程人员添加新的软件工具
    - pipe, shell scripts; Keep It Simple, Stupid (KISS)
- 1991，Linux
  - 开放源码的“集中-分散-集中”开发模式
  - 优良的设计原理
    - 实用的原则、有限目标的原则（实用、速度快、可移植）
    - 简洁设计的原则（归纳共性、极小接口、模块）

计算机科学是研究计算过程的科学

十种理解是一个整体：如，抽象是能够构造性自动执行的抽象

理解1：自动执行。计算机能够自动执行由离散步骤组成的计算过程。

理解2：正确性。计算机求解问题的正确性往往可以精确地定义并分析。

理解3：通用性。计算机能够求解任意可计算问题。

理解4：构造性。人们能够构造出聪明的方法让计算机有效地解决问题。

理解5：复杂度。这些聪明的方法（算法）具备时间/空间复杂度。

理解6：连接性。很多问题涉及用户/数据/算法的连接体，而非单体。

理解7：协议栈。连接体的节点之间通过协议栈通信交互。

理解8：抽象化。少数精心构造的计算抽象可产生万千应用系统。

理解9：模块化。多个模块有规律地组合成为计算系统。

理解10：无缝衔接。计算过程在计算系统中流畅地执行。

自动

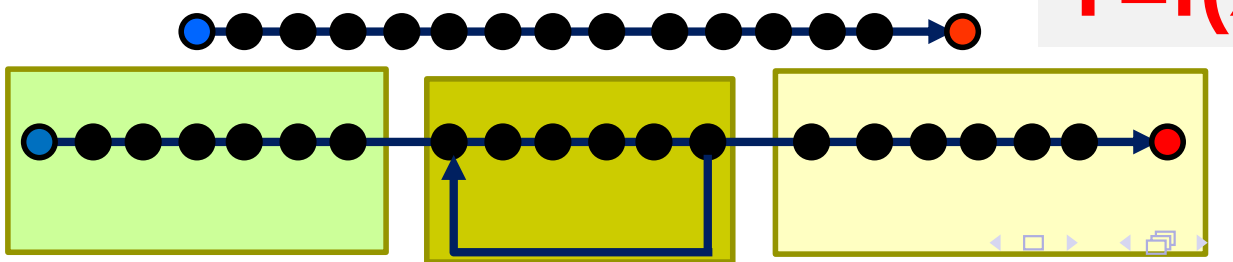
通用

算法

联网

抽象

$$Y=f(X)$$



从计算系统思维角度看到的一些计算机科学基本抽象

- 抽象使得计算机科学成为一门优美的领域
  - 本质：采用一种方法解决该层次的所有问题，应对系统的复杂性，**以不变的抽象应系统的万变**

数字符号	**比特、字节、四进制数、十六进制数、整数、数组、BMP 图像。**	
软件	**算法**	巧妙的信息变换方法。例如信息隐藏算法。
	程序	算法的代码实现。例如实现信息隐藏算法的 hide.go 程序。
	进程	运行时的程序。例如在 Linux 环境中的 hide 进程。
	指令	程序的最小单位，计算机能够直接执行。
硬件	**指令流水线**	每条指令都通过“取指-译码-执行-写回”四个操作组成的指令流水线得以自动执行。所有指令都由这一种机制执行，指令流水线由若干时钟周期组成。
	时序电路	等同于**自动机**，说明每一个时钟周期的操作。时序电路由组合电路与存储单元组成，理论上等同于**图灵机**。
	组合电路	实现二值逻辑表达式（**布尔逻辑表达式**）。

计算机体系结构

张燕咏

yanyongz@ustc.edu.cn

Welcome to Computer Architecture

- **主讲：张燕咏 (yanyongz@ustc.edu.cn)**
  科技楼106

- **辅讲：闫宇博 (yuboyan@ustc.edu.cn)**
  图书馆1408

- **课程主页:**
  http://staff.ustc.edu.cn/~comparch

Eight Great Ideas in Computer Architecture

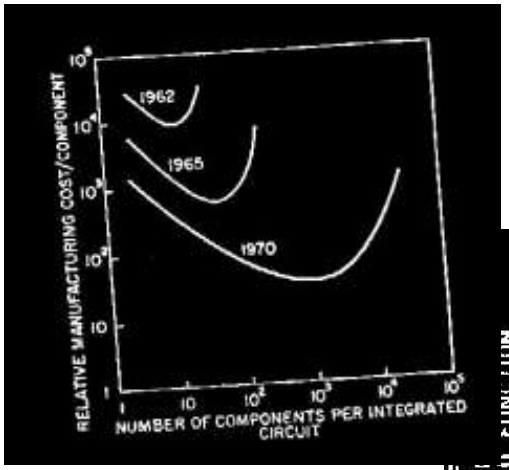
These are eight great ideas that computer architects have invented in the last 60 years of computer design. They are so powerful they have lasted long after the first computer that used them, with newer architects demonstrating their admiration by imitating their predecessors -- Patterson

https://www.elsevier.com/connect/8-great-ideas-in-computer-architecture

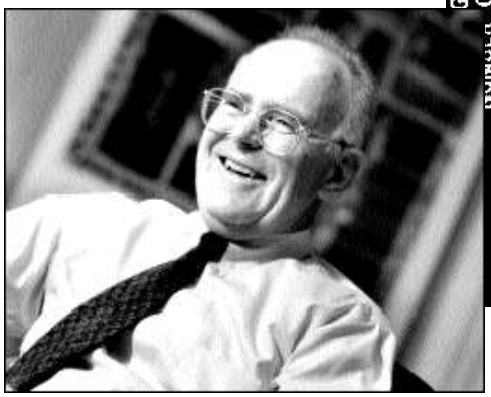
Eight Great ideas in Computer Architecture

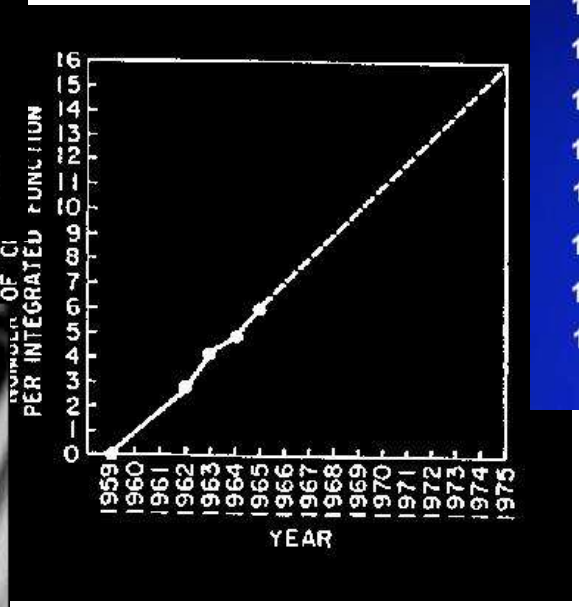
1. **Design for Moore’s Law**
2. **Abstraction to Simplify Design**
3. **Make the Common Case Fast**
4. **Dependability via Redundancy**
5. **Memory Hierarchy**
6. **Performance via Parallelism**
7. **Performance via Pipelining**
8. **Performance via Prediction**

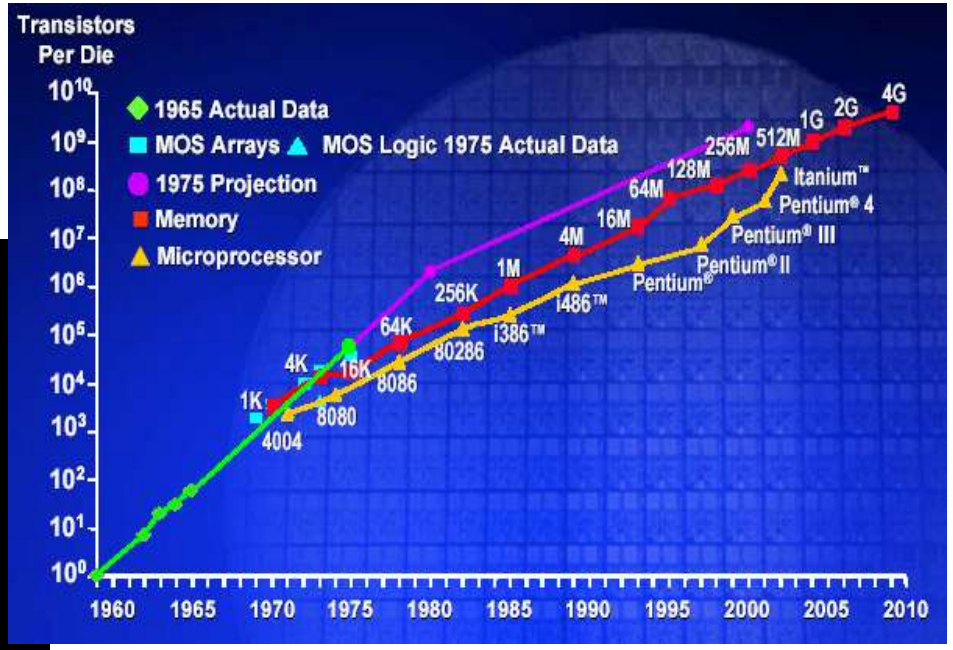
1. Moore's Law（摩尔定律）



Who is Moore?







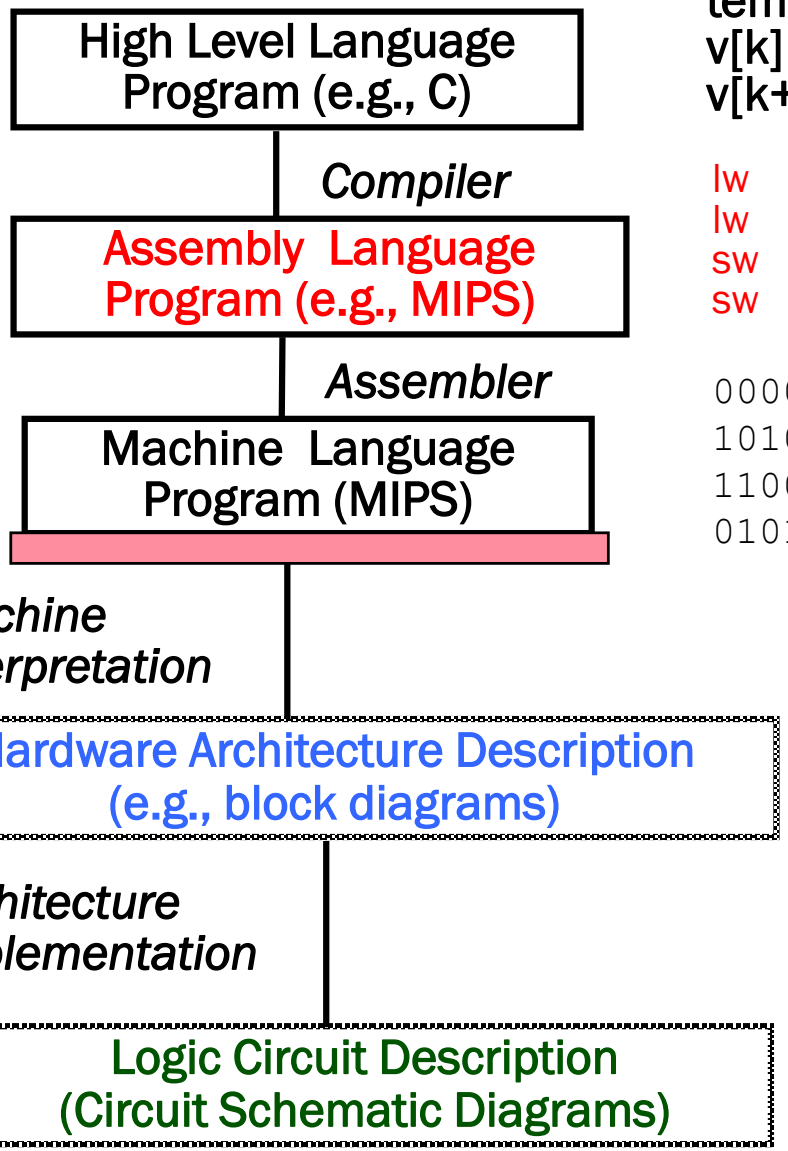
1．集成电路芯片上所集成的电路的数目，每隔18个月就翻一番。
2．微处理器的性能每隔18个月提高一倍，或价格下降一半。
3．用一个美元所能买到的电脑性能，每隔18个月翻两番。

- "Cramming More Components onto Integrated Circuits"
  - Gordon Moore, Electronics, 1965
- # on transistors on cost-effective integrated circuit double every 18 months

1. Design for Moore’s Law

The one constant for computer designers is rapid change, which is driven largely by Moore's Law. It states that integrated circuit resources double every 18–24 months. Moore's Law resulted from a 1965 prediction of such growth in IC capacity made by Gordon Moore, one of the founders of Intel. As computer designs can take years, the resources available per chip can easily double or quadruple between the start and finish of the project. Like a skeet shooter, computer architects must anticipate where the technology will be when the design finishes rather than design for where it starts.

2. Abstraction via Layers of Representation




```
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)
```

```
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
```

Abstraction uses multiple levels with each level hiding the details of levels below it. For example:

The instruction set of a processor hides the details of the activities involved in executing an instruction.

High-level languages hide the details of the sequence of instructions needed to accomplish a task.

Operating systems hide the details involved in handling input and output devices.

2. Use Abstraction to Simplify Design

Both computer architects and programmers had to invent techniques to make themselves more productive, for otherwise design time would lengthen as dramatically as resources grew by Moore's Law. A major productivity technique for hardware and software is to use abstractions to represent the design at different levels of representation; lower-level details are hidden to offer a simpler model at higher levels. We'll use the abstract painting icon to represent this second great idea.

GIVE ME AN EXAMPLE?

3. Make the Common Case Fast

- 在进行设计选择时，注重优化经常发生的事件
- 优化不经常执行的代码意义不大
- 选择一种（性能）度量方式来确定经常性事件（Common case)

你的任务是把人以最快速度从A点送到B点，绝大多数情况下只有一个人。你选哪辆车？

在设计ISA时该如何利用这个idea?

如何知道哪个指令最常运行？

4. Dependability via Redundancy

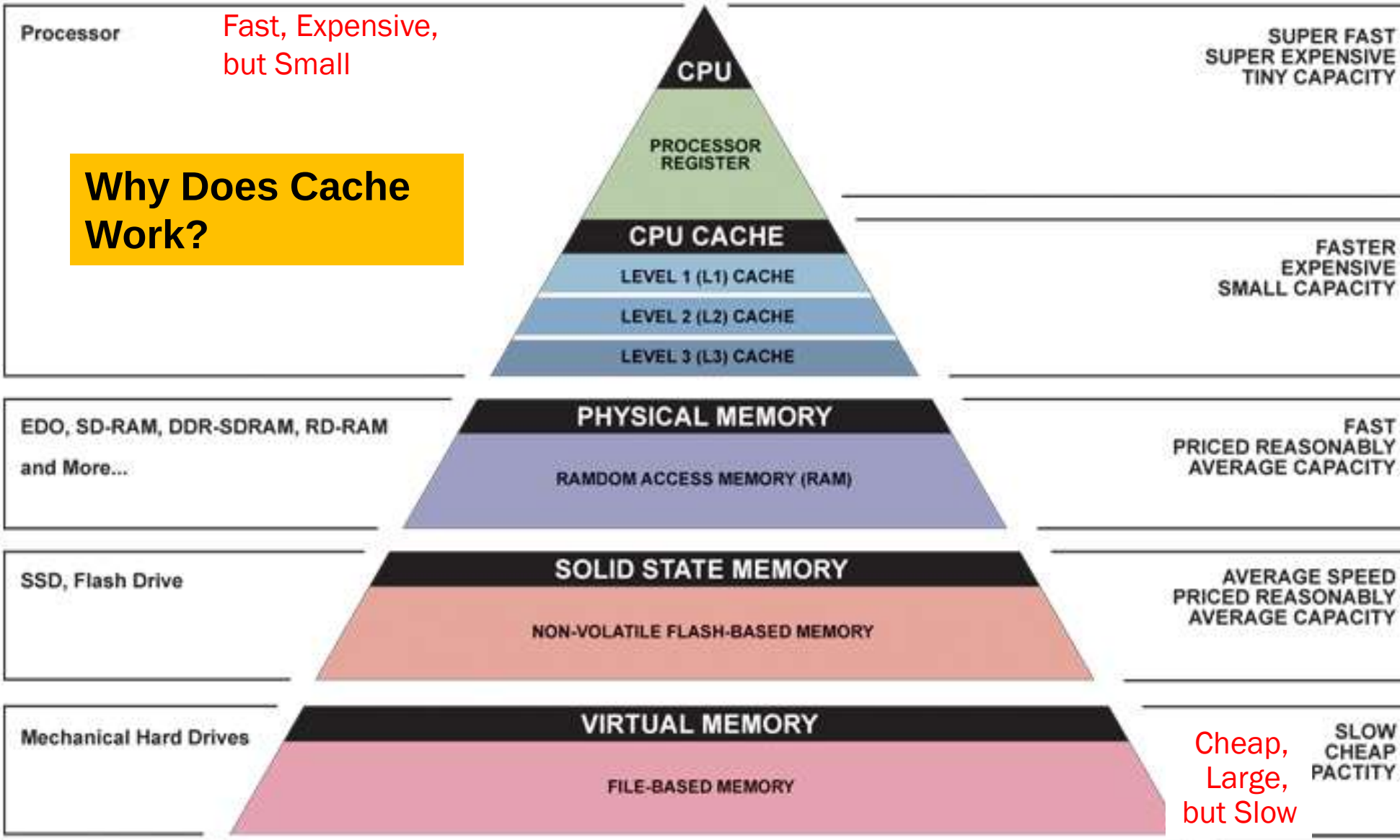
- **通过冗余使得部分部件失效不影响整个系统的运行**

假设你的预算为T，是买一台单价为T的大容量磁盘，还是买100台单价为T/100的廉价磁盘？

Storage servers with 24 hard disk drives and built-in hardware RAID controllers supporting various RAID levels

One of the most important ideas in data storage is the Redundant Array of Inexpensive Disks (RAID) concept. In most versions of RAID, data is stored redundantly on multiple disks. The redundancy insures that if one disk fails the data can be recovered from other disks.

5. Memory Hierarchy



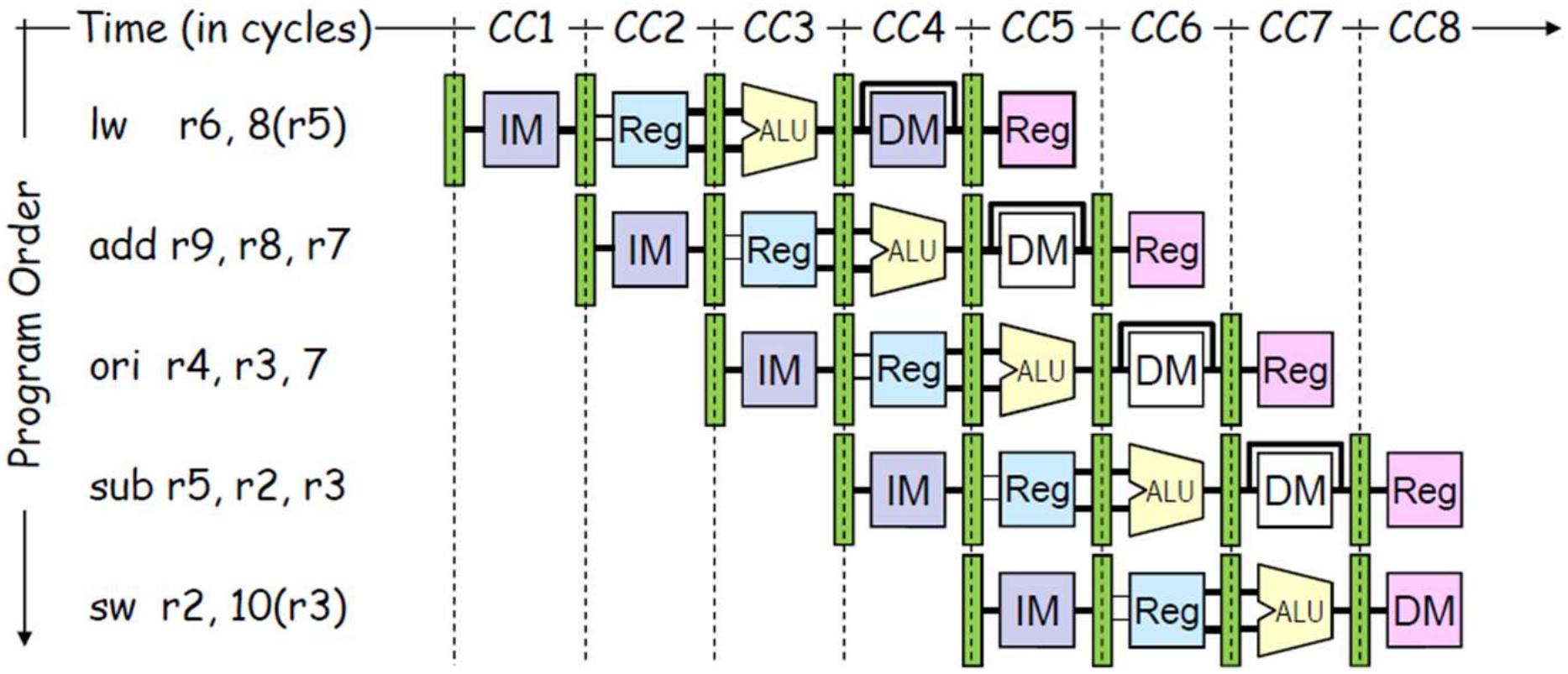
6. Performance Through Parallelism

Doing different parts of a task in parallel accomplishes the task in less time than doing them sequentially. A processor engages in several activities in the execution of an instruction. It runs faster if it can do these activities in parallel.

- Parallel Requests
  Assigned to computer, e.g., Search “Katz”
- 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, GPU is SIMD (Single Instruction Multiple Word)
- Hardware Descriptions
  All gates functioning in parallel at same time

Performance Through Pipelining

This idea is an extension of the idea of parallelism. It is essentially handling the activities involved in instruction execution as an assembly line. As soon as the first activity of an instruction is done you move it to the second activity and start the first activity of a new instruction. This results in executing more instructions per unit time compared to waiting for all activities of the first instruction to complete before starting the second instruction.

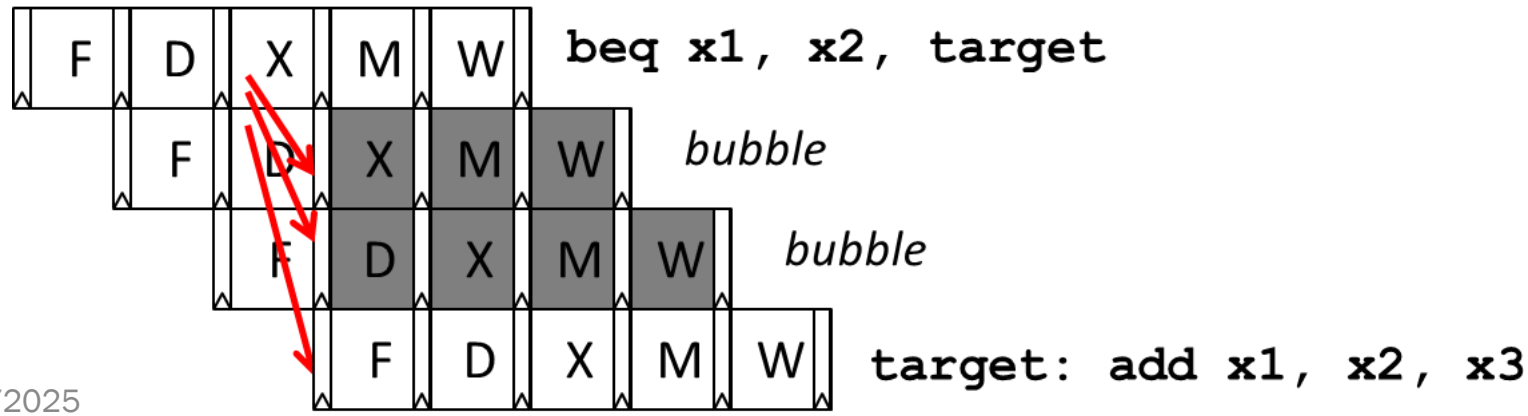


8. Performance Through Prediction

A conditional branch is a type of instruction that determines the next instruction to be executed based on a condition test. Conditional branches are essential for implementing high-level language if statements and loops.

Unfortunately, conditional branches interfere with the smooth operation of a pipeline — the processor does not know where to fetch the next instruction until after the condition has been tested.

Many modern processors reduce the impact of branches with speculative execution: make an informed guess about the outcome of the condition test and start executing the indicated instruction. Performance is improved if the guesses are reasonably accurate and the penalty of wrong guesses is not too severe.



Acknowledgements

- **These slides contain material developed and copyright by:**
  - John Kubiatowicz (UCB)
  - Krste Asanovic (UCB)
  - John Hennessy (Standford)and David Patterson (UCB)
  - Chenxi Zhang (Tongji)
  - Muhamed Mudawar (KFUPM)
- **UCB material derived from course CS152、CS252、CS61C**
- **KFUPM material derived from course COE501、COE502**

计算机科学导论

网络思维

提纲

1. 计算过程与网络思维
    1. 连通性、消息传递协议
2. 网络思维例子
3. 网络思维要点
    1. 名字空间
    2. 拓扑：网络是图（节点+连接）
    3. 协议栈
4. 服务质量（即使出现错误或故障）
5. 网络的规律

网络思维与算法思维有很大的不同

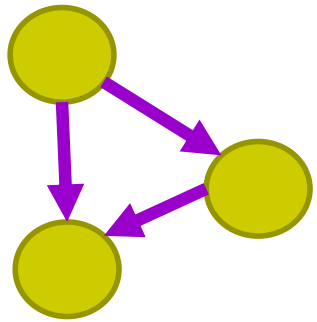
- 算法有统一的定义，网络有很多定义
  - 算法的定义【**Donald Knuth**】. 一个算法是一组有穷的规则，给出**求解特定类型问题**的运算序列，并具备下列五个特征：
    - **有限性**：一个算法在有限步骤之后必然要终止。
    - **确定性**：一个算法的每个步骤都必须精确地（严格地和无歧义地）定义。
    - **输入**：一个算法有零个或多个输入。
    - **输出**：一个算法有一个或多个输出。
    - **能行性**：一个算法的所有运算必须是充分基本的，原则上人们用笔和纸可以在有限时间内精确地完成它们。

功能

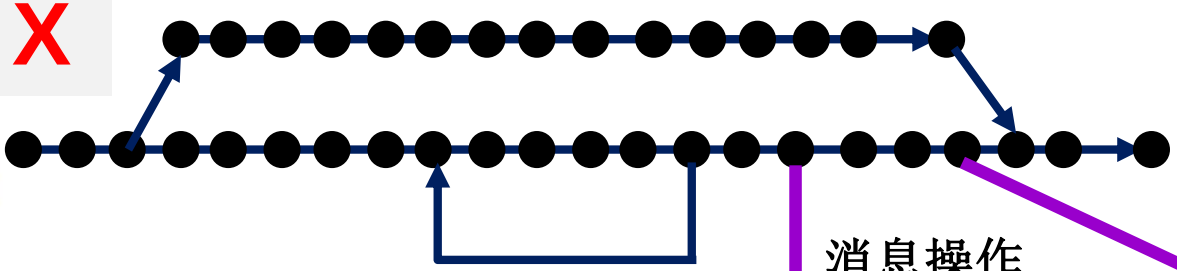
- 算法有时间复杂度等少数几个统一指标；网络有很多指标
- 算法由基本运算与步骤组合而成，时间复杂度具有组合性；网络的指标组合性较弱

1. 网络思维的主要知识点

- 计算过程涉及（由多个**节点**连接而成的）**网络**
  - 网络成为计算过程的对象、执行系统
- 核心概念：连通性、消息传递、协议
  - **名字空间、拓扑、协议栈**



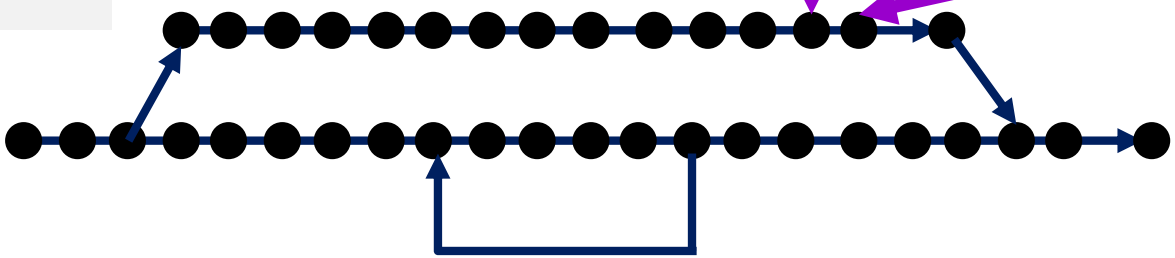
X



消息操作

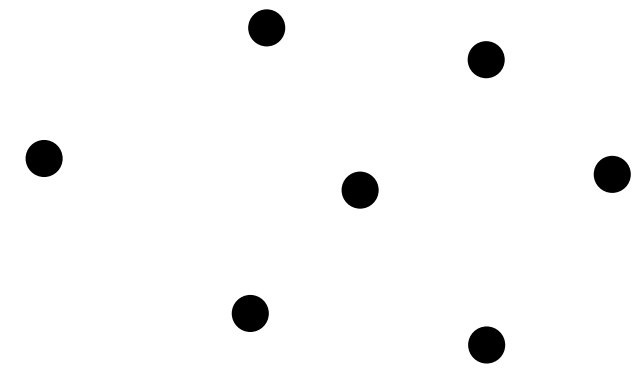
Z

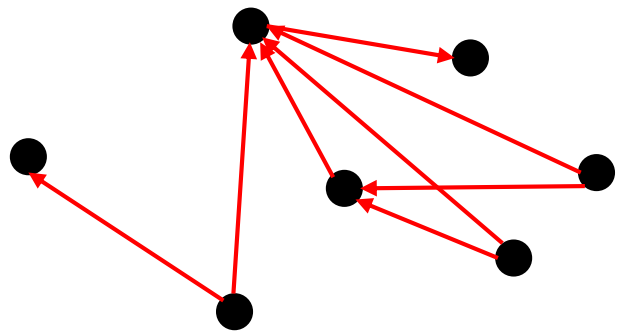
Y



连通性与消息传递是松耦合关系

- 网络思维并不必涉及消息传递（或通信协议）
- 此时，重点是**连通性**（connectivity）
  - 即：有什么节点？节点之间如何连接？
  - 拓扑本身就有价值
- 搜索引擎实例
  - 第一代：无网络思维
    - 只关心节点的内容
  - 第二代：有网络思维
    - Page、 Kleinberg、李彦宏
    - 关心节点内容
    - **还关心网络拓扑（pagerank）**



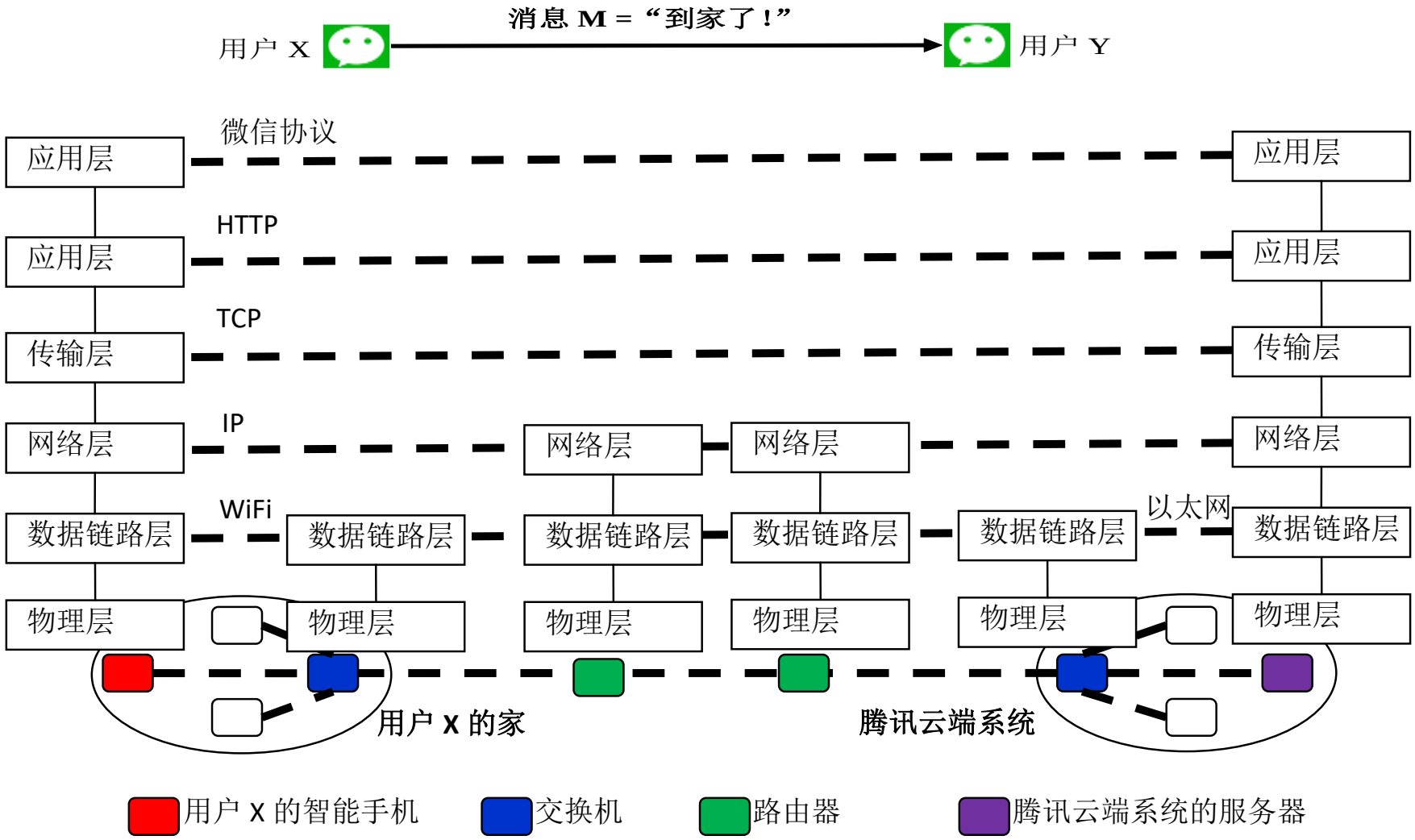


用网络思维解决问题的另一个实例

- 传统的电话通信系统
  - 采用专用的通信信令协议栈
    - 保证语音通信质量
    - 成本高
- 今天很多通信系统采用通用互联网协议栈实现
  - 挑战：保证质量
  - 优点：利用全球互联网
  - 构建在互联网之上 → Over The Top（简称OTT）
  - “OTT服务”是网络思维实例，成功例子很多
    - 微信
    - 视频
    - 语音

2. 从微信例子看协议

- 网络：多个（算法+通信操作）组合而成



微信网络协议栈（实线表示层间接口，虚线表示对等接口）

3. 网络思维要点

- 网络思维：是连通性和协议的整体
  - 是名字空间、网络拓扑、协议栈的整体
- 三个核心概念
  - **名字空间**：精确的说明一个网络有哪些节点
    - 通信“对方”是谁、连接“伙伴”是谁
  - **拓扑**：哪些节点间需要连接和通信？
  - **协议栈**：节点间如何连接与通信，甚至做更高级的操作？
    - 传递消息：互联网
    - 传递信任？区块链
    - 传递情感？社交网络部分提供
    - 传递智能？

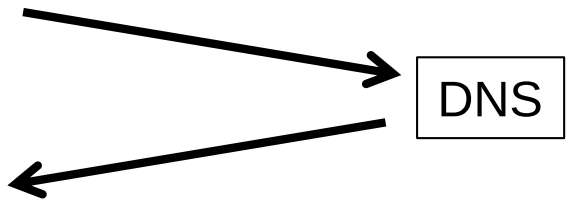
名字空间

- Name space；naming
- 主要用于指称网络中的节点

名字空间实例	节点的名字举例	名字空间解释
微信名字	中关村民	腾讯公司规定的任意“合法的”字符串
电子邮箱地址	zxu@ict.ac.cn	用户名@因特网域名
手机号码	189-8888-9999	通信公司规定的11位10进制数字串
本机文件路径（本地路径）	/我的文件/教材.pdf	本机操作系统规定的文件名
本机网卡地址（MAC地址）	00-1E-C9-43-24-42	全球统一规定的12位16进制数字串
网站域名	www.ict.ac.cn	互联网协议栈规定的域名
网站IP地址	159.226.97.84	IP协议规定的合法地址

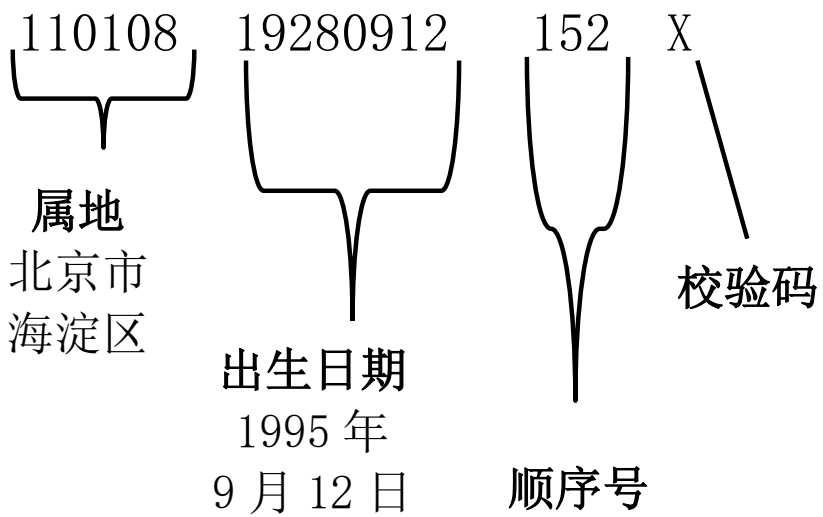
名字空间设计的两个问题

- 设计与理解名字空间的基本考虑
  - **唯一性**：zxu@ict.ac.cn　vs.　中关村民
    - 重用性：手机号码 vs. 万维网资源的URI
  - **动态性**：身份证 vs ？
    - 固定IP地址 vs. 动态生成IP地址
  - **友好性**：中关村民 vs. 以太网MAC地址
- 不同层次间的名字如何解析
  - 本地解析 vs. 全网解析（远程解析）
    - http://www.ict.ac.cn/本地路径...
    - http://159.226.97.84/本地路径...



第二代身份证18位数字名字空间

- 名字空间的性质
  - 能否保证名字唯一性？
  - 名字的自主性如何？
  - 友好性如何？
  - 11010819560921141的解释和验证码



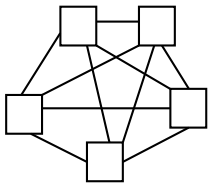
余数	校验码
0	1
1	0
2	X
3	9
4	8
5	7
6	6
7	5
8	4
9	3
10	2

网络拓扑

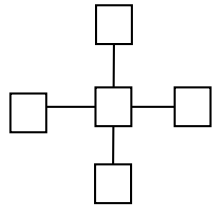
- 网络往往可以看成是一个由节点和连线（也称为边）构成的图
  - 网络不是节点的集合
- 设计与理解网络拓扑的基本考虑
  - 可扩展性
    - 规模可扩展（10个、10亿个节点）
    - 地域可扩展（局域网、广域网等）
  - 连通性与容错
  - 节点的连接度、节点间距离、网络的直径
    - 消息通信的延迟、带宽、成本、功耗
  - 动态性：静态网络、动态网络、演化网络

按动态性划分的三类网络拓扑

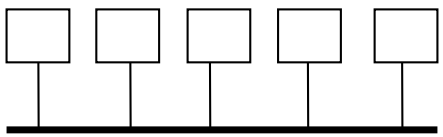
- 静态网络：节点完全确定、连接完全确定
- 动态网络：节点完全确定、连接部分确定
- 演化网络：节点部分确定、连接部分确定
- **你的微信朋友圈是什么网络？**



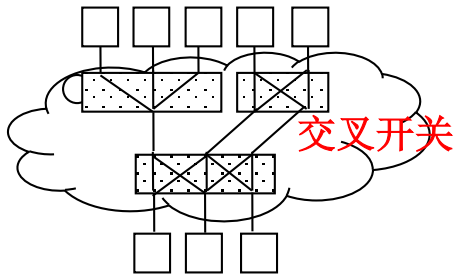
(a) 全连通图



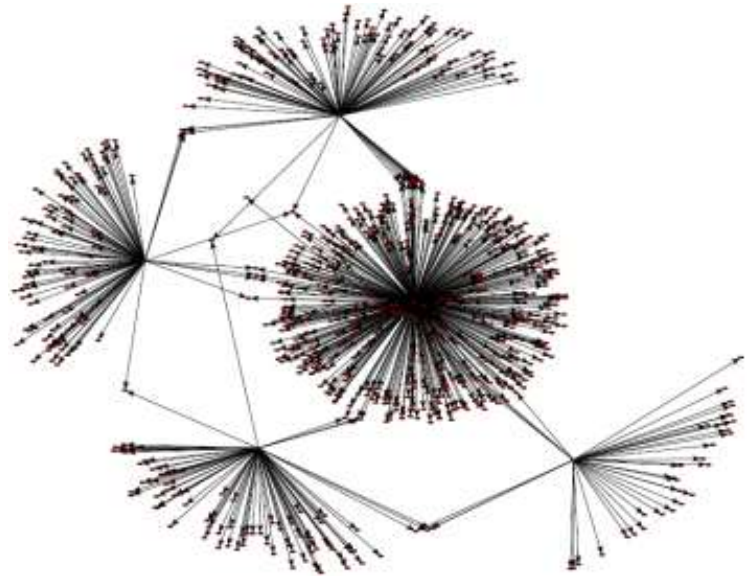
(b) 星型网络



(c) 总线



(d) 交换网络

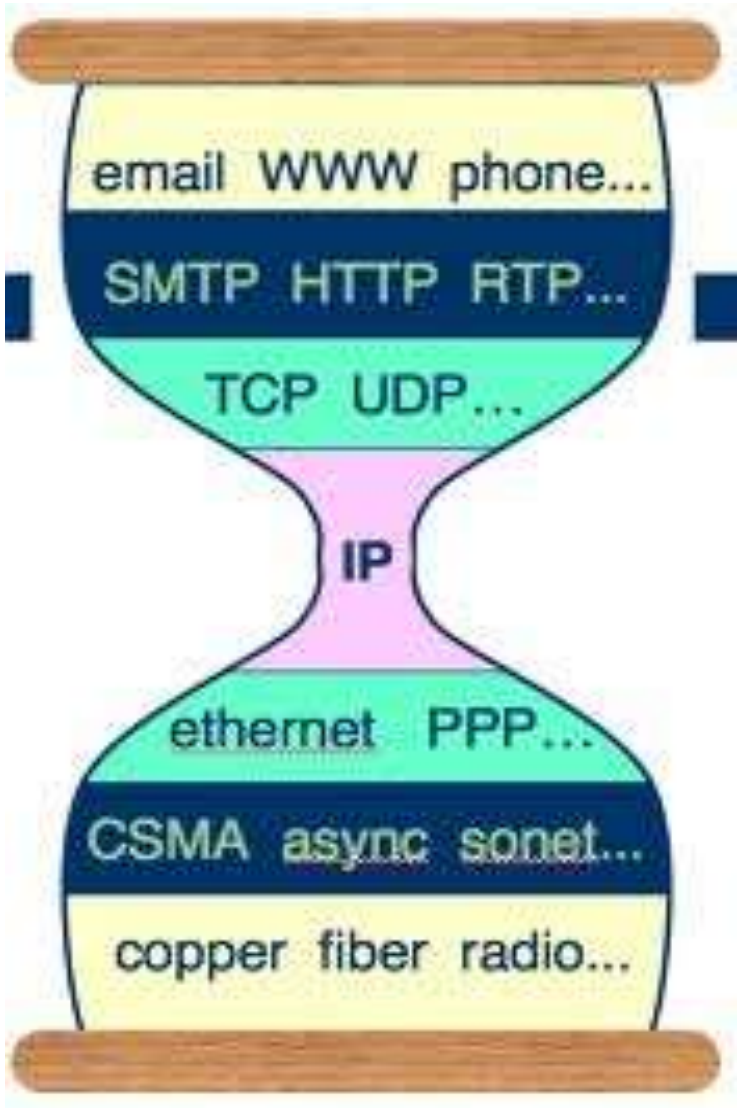


(e) 演化网络

协议栈的生态系统思想
互联网协议的沙漏模型

- IP是细腰
- 2000年以来，
  HTTP也是细腰
- OTT (over the top)服务

- 支持多种上层应用
- 支持多种下层技术



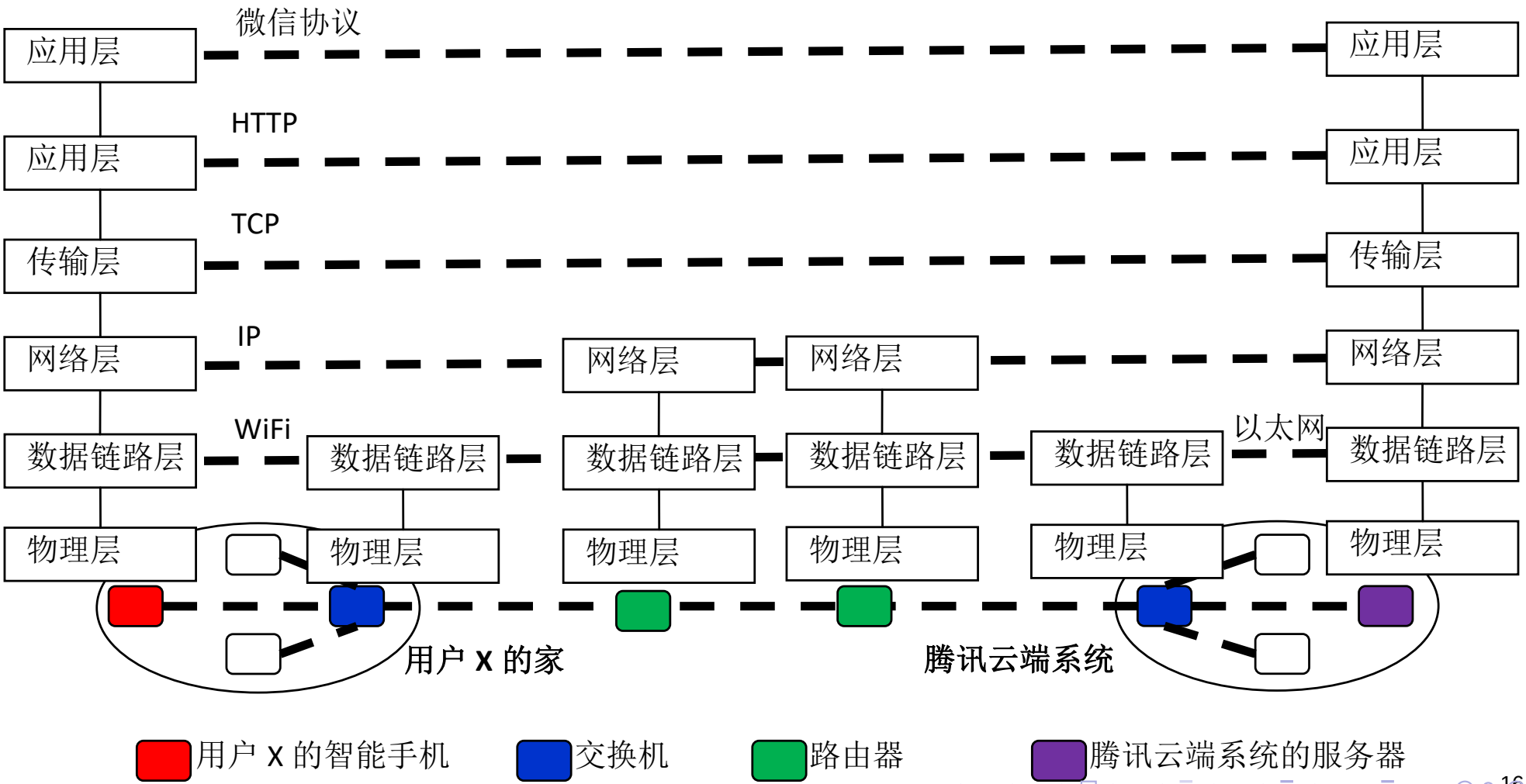
图片来源： Famous Internet Quotes，
https://www.ofcourseimright.com/?page_id=1504

从应用角度理解协议栈的四个问题

- 通信过程涉及互联网协议栈的哪些接口？
- 通信过程涉及互联网协议栈的哪些层次？
- 通信过程涉及哪些硬件？
- 一条应用层的微信消息“到家了！”如何解析成底层的消息包？

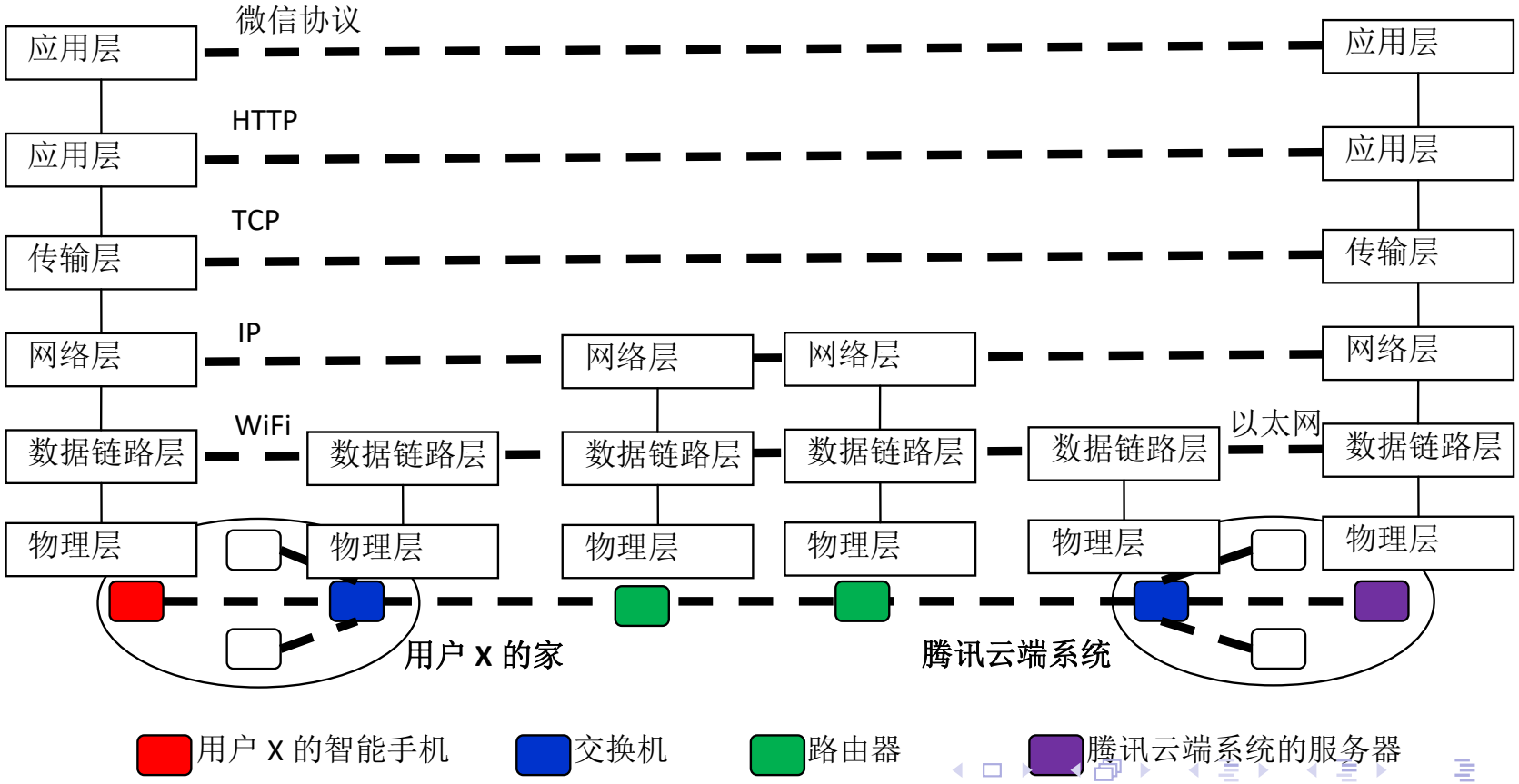
通信过程涉及互联网协议栈的哪些接口？

- 对等接口、层间接口



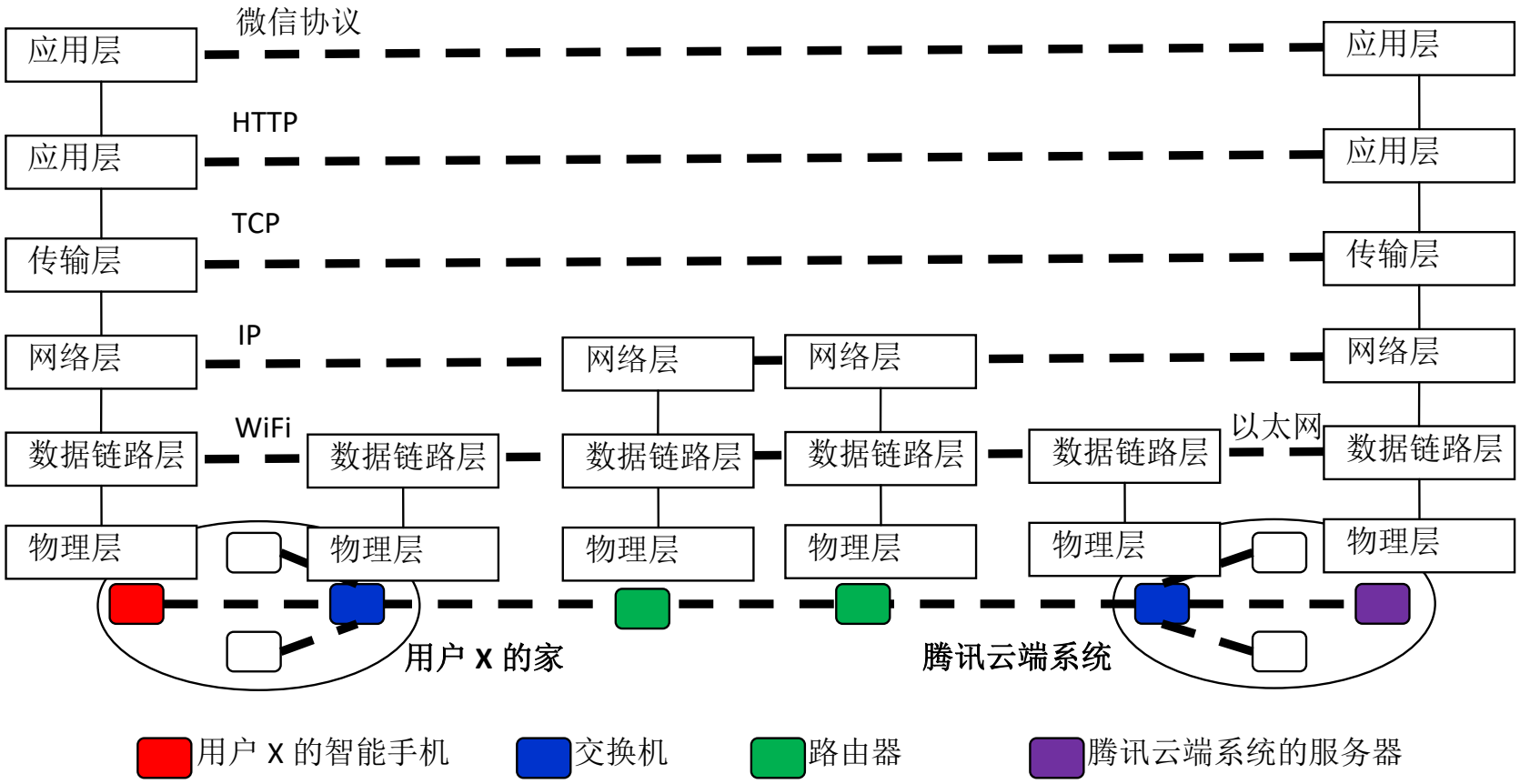
通信过程涉及互联网协议栈的哪些层次？

- 应用层：　　到应用
- 传输层：　　到进程
- 网络层：　　到跨网设备
- 数据链路层、物理层：　　到网内设备



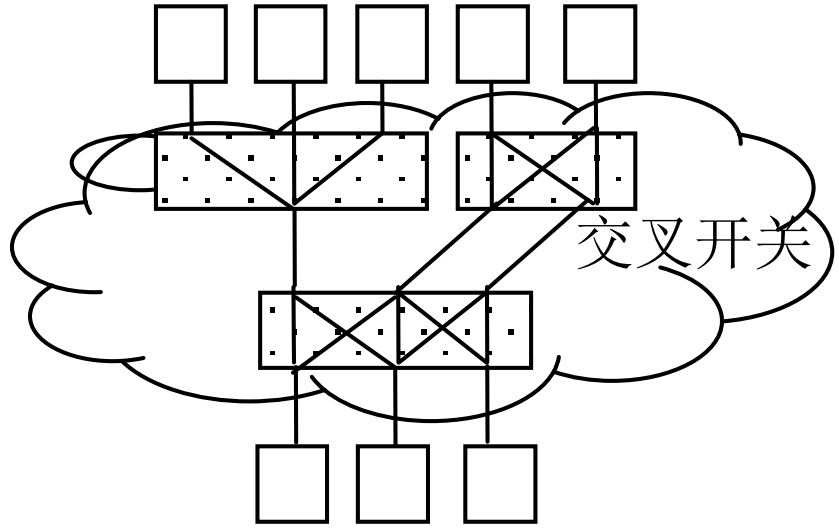
通信过程涉及哪些硬件？

- 用户的智能手机、家庭无线网（WiFi）、WiFi交换机、广域网、两个IP路由器、腾讯云端系统交换机、腾讯云端系统数据中心局域网、腾讯云端系统服务器



“到家了！”如何解析成底层的消息包？

- 关键技术点
  - 线路虚拟化：分组交换
  - 包：首部+数据
  - 逐层解析并传输
- **规范实践与黑客实践**

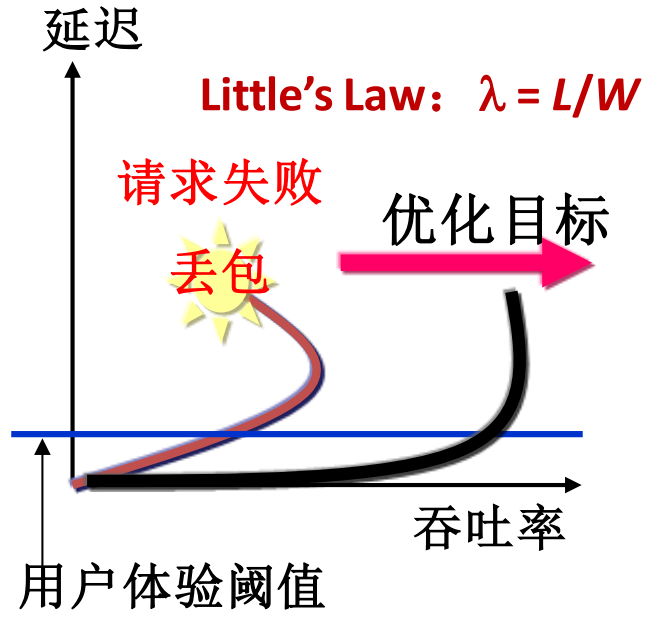


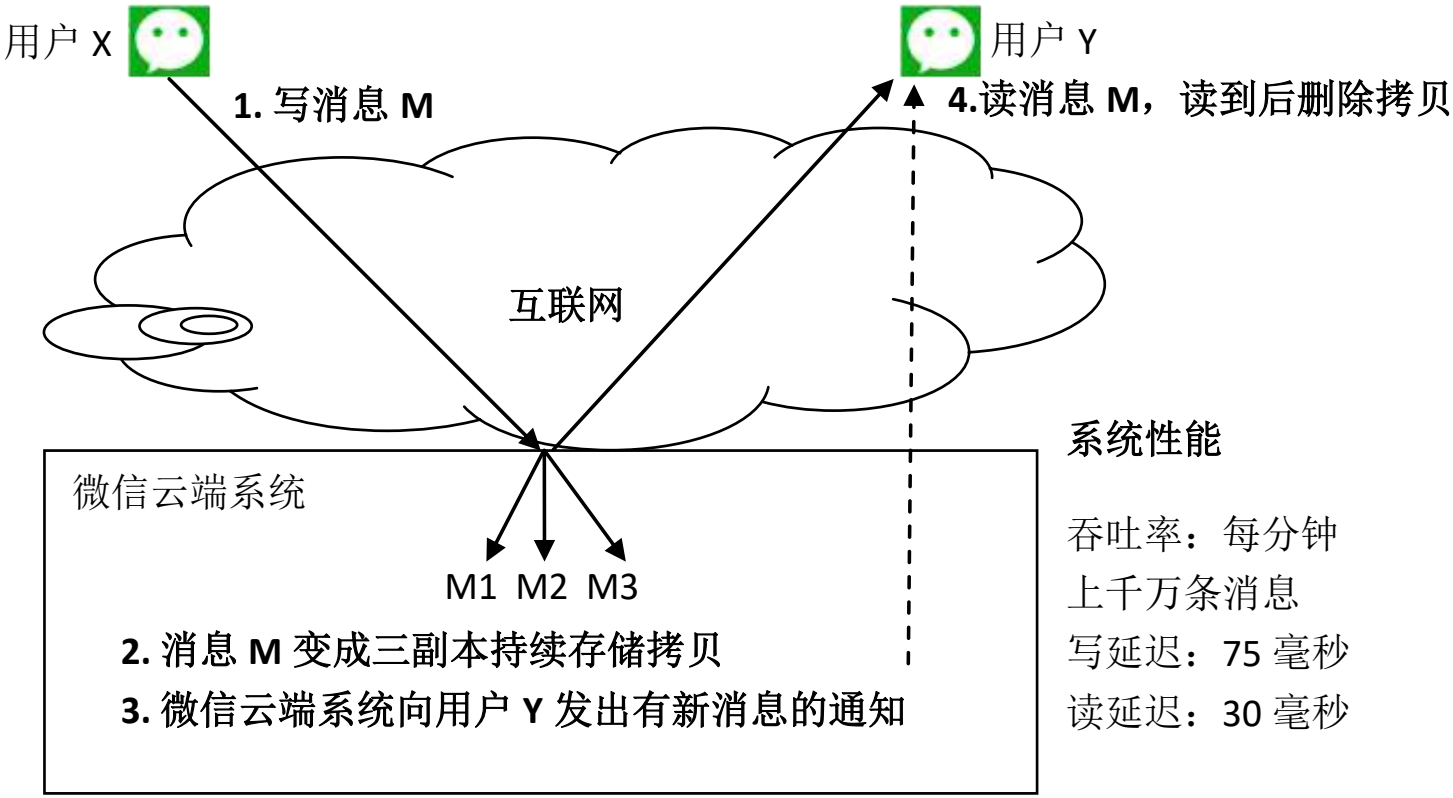
微信协议：中关村民送消息“回家了！”给好友 Y

应用层		
应用层	HTTP	GET http://mp.weixin.qq.com/...回家了！ ...
传输层	TCP	TCP 首部 [**80 端口**] | **GET :// /...回家...**
网络层	IP	IP 首部 **[140.206.160.199]** | TCP 首部 | **GET :// /...回家...**
WiFi 层	802.11	WiFi 首部 | IP 首部 | TCP 首部 | **GET :// /...回家...**

4. 服务质量与用户体验

- 传输单个消息的性能
- 整个网络计算系统的性能
  - 请求规模、延迟、吞吐率





系统性能

吞吐率：每分钟上千万条消息

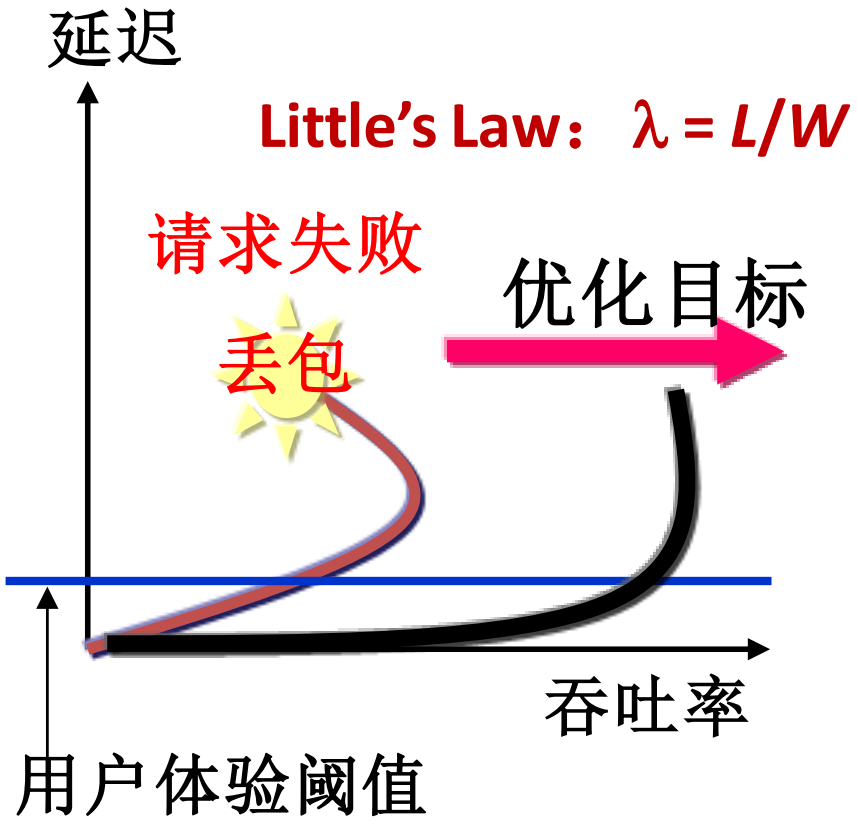
写延迟：75 毫秒

读延迟：30 毫秒

2013.11.11 “光棍节”微信系统实现的性能

服务质量的多面观

- 服务质量=用户体验
- 用户体验差的功能，是缺失的功能
- 体验差的例子
  - 延迟太长
  - 马赛克
  - 视频卡断
  - 陌生人混入朋友圈
  - 机器翻译质量太差



E2E原理实例

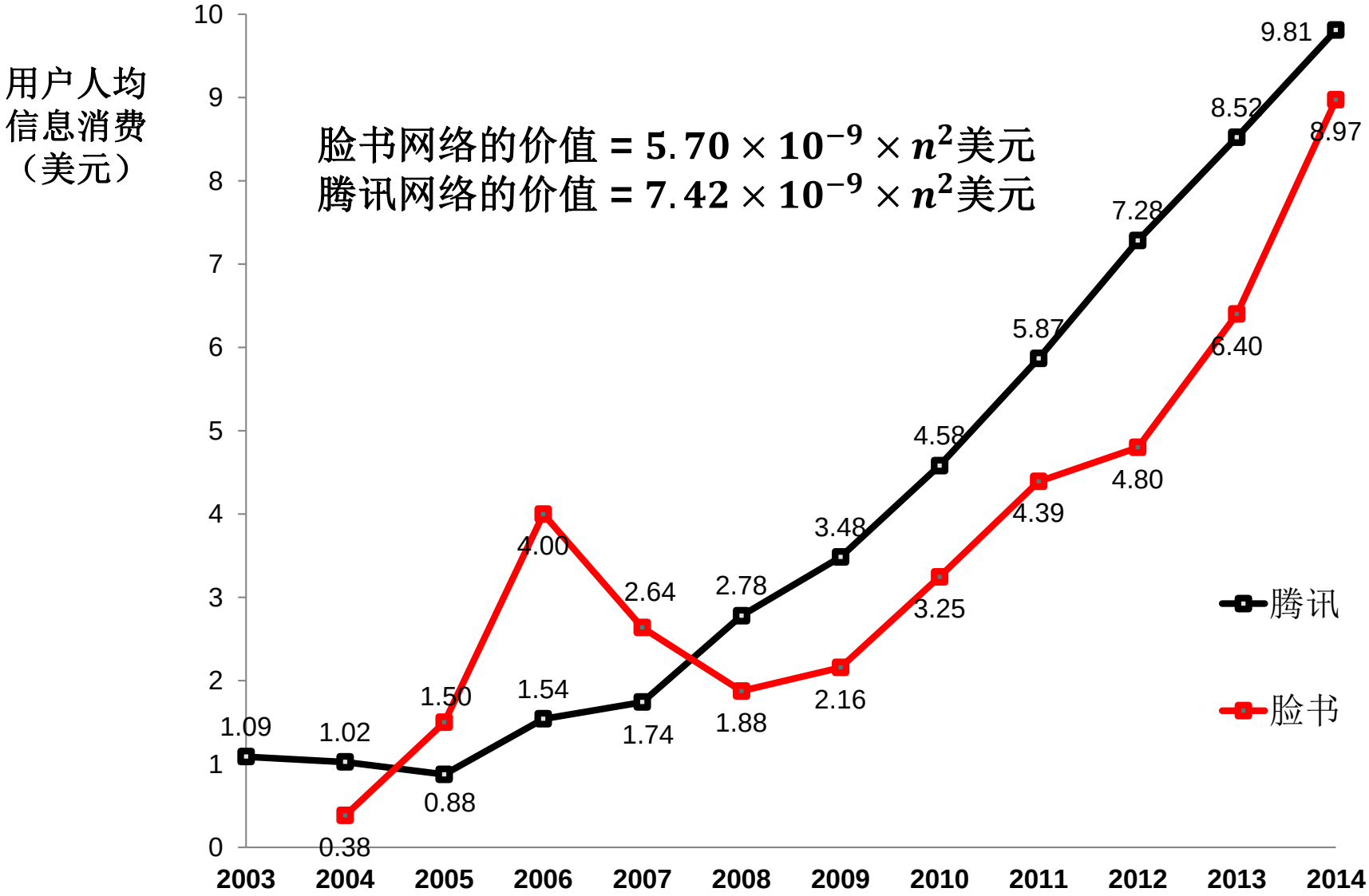
- 为什么网络作业需要这么多查错？

```
 8  func main() {
 9      if httpresp, err := http.Get("http://csintro.ucas.ac.cn/static/code_project
        /Richard_Karp.txt"); err != nil || httpresp.StatusCode != http.StatusOK {
10              if err != nil {
11                      fmt.Fprintln(os.Stderr, err.Error())
12              } else {
13                      fmt.Fprintln(os.Stderr, httpresp.Status)
14              }
15              return
16      } else {
17              if data, err := ioutil.ReadAll(httpresp.Body); err != nil {
18                      fmt.Fprintln(os.Stderr, err.Error())
19              } else {
20                      fmt.Println(string(data))
21              }
22      }
23 }
```

网络的规律

- 无为而治原理（the end-to-end argument）
  - 某些功能必须有边缘节点（应用）参与才能正确实现
    - 除非能够完全而正确地做到，网络不应该实现该功能
      - Keep it simple, stupid；我们不需要自作聪明的网络
- 网络效应：网络的价值随节点数超线性增长
  - 麦特考夫定律：网络的价值与节点数的平方成正比
  - 里德定律：网络价值＝2^{C-1}, C是“社区”个数
- 病毒性市场现象（viral marketing）
  - 低价格（购买成本为零）
  - 好使用（使用成本为零）
  - 易传播（传播成本为零）
  - 强黏糊（sticky）

脸书与腾讯数据验证了梅特卡夫定律



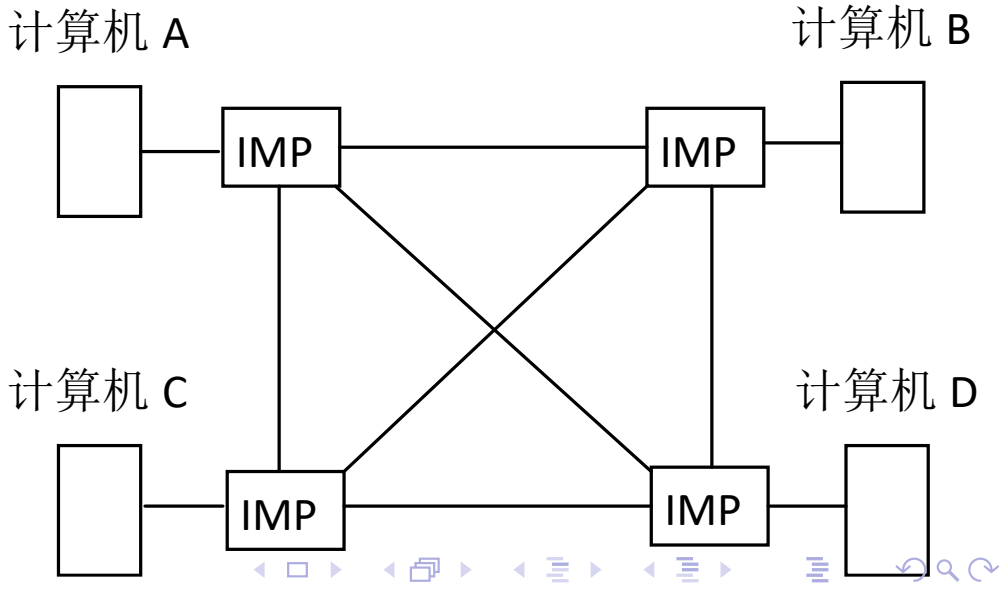
Zhang X, Liu J, Xu Z, Tencent and Facebook Data Validate Metcalfe's Law, JCST, 2015 Vol. 30 (2): 246-251.

电脑网络创新故事：ARPANET

- 条件与思想
  - 1957年10月4日，苏联成功地发射了人造卫星
  - 1958年，美国成立ARPA，IPTO
  - 1960年，利克莱德提出电脑网络思想
- 项目与实施
  - 1966年，鲍伯•泰勒启动ARPANET（为什么？）
  - 1966年秋，罗伯兹开始设计ARPANET
    - 1968年8月，罗伯兹完成了ARPANET的技术规范，并向全国140家公司发出了招标书
    - 1968年12月，BBN公司中标
    - 1969年9月，第一个ARPANET节点安装在加州大学洛杉矶分校

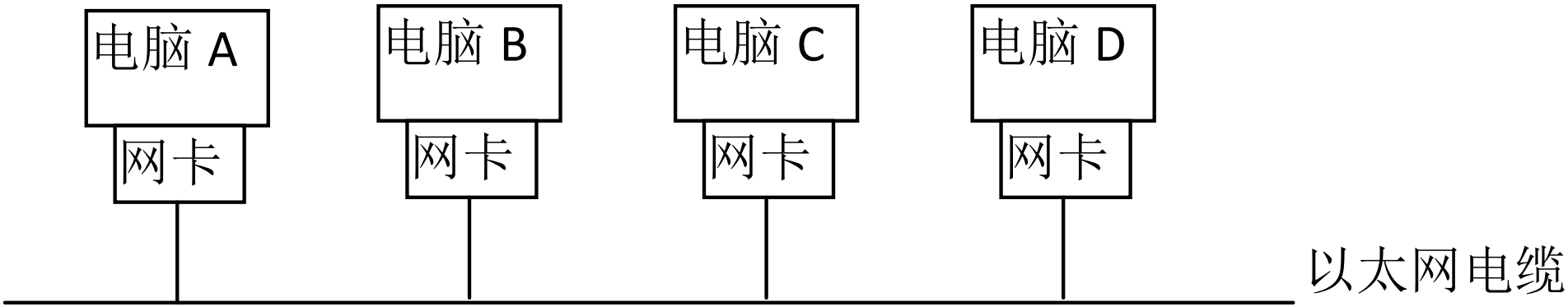
ARPANET成果

- 1969年10月29日，ARPANET进行了第一次试验，传输了“LO”GIN
- 第一个电脑网络
- 第一个计算机网络技术标准（RFC）
- 第一个交换机
- 重传容错机制
- 验证了分组交换
- **引发了三个新问题**
  - 局域网、路由器、因特网



局域网与以太网

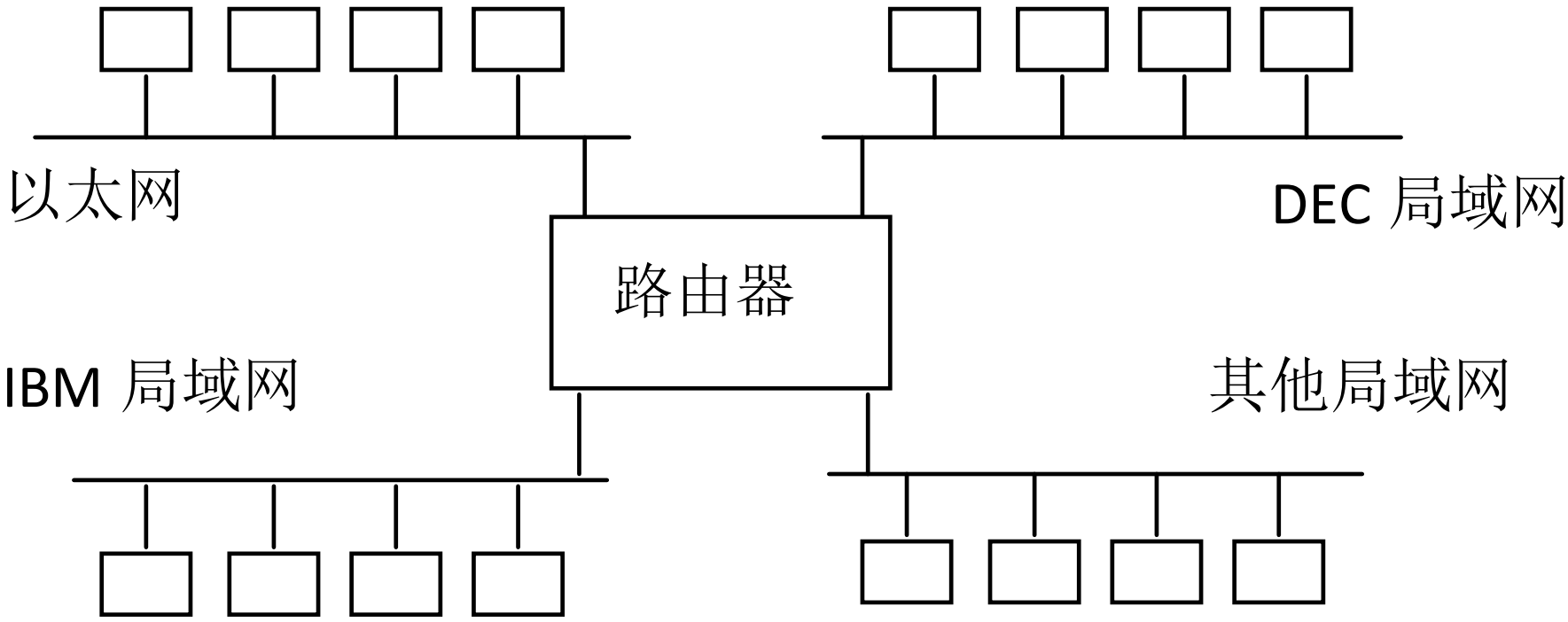
- 1973年，麦特考夫发明以太网
  - 网卡
  - 解决冲突的指数退避方法
    - 第一次传输试图失败后，等候[0, T]中间的一个随机值
    - 第二次重试失败后，等候[0, 2T]中间的一个随机值
    - 第三次重试失败后，等候[0, 4T]中间的一个随机值



以太网和四台电脑构成的局域网

校园网、城域网与路由器

- 1980-1983年，波沙克和勒纳尔实现路由技术
- 1983-1984年，波沙克和勒纳尔发明路由器
  - 专用设备



TCP/IP协议栈

- 1974年5月，康恩和舍夫在IEEE Transactions on Communication发表TCP/IP协议栈论文
  - 特点：协议栈
  - 因特网（Internet）技术正式诞生，它可以把很多计算机网络互联起来组成一个大的网络之网
- 1983年，ARPANET与美国国防部的另一个网络“国防数据网”开始使用TCP/IP协议
  - 有人把这个时间认为是因特网的真正诞生年代，因为我们今天所说的因特网，是指使用IP的网络之网

早期应用：主要并不是计算资源共享

- FTP、Telnet、BBS、电子邮件
- 1972年，BBN工程师汤姆林生在快完成FTP的编程工作时突发奇想：为什么不能用FTP来自动地传送网络电子邮件呢?
  - 本机电子邮件技术 + 远程文件传输技术FTP
  - mailto: zxu@ict.ac.cn
- 1973年，ARPANET上的四分之三的通信是电子邮件
- 电脑网络就只是一个更快的邮局系统吗？

万维网

- 1980-1990年，提姆•伯纳尔斯-李发明万维网
  - 核心思想和价值：将超文本、超链接技术从单机内拓展到全球计算机网络，实现全球电脑网络中的文档资源互连
  - 四项关键技术
    - 文档资源的命名：URL
    - 文档资源的表示：HTML
    - 文档资源的访问：HTTP
    - 核心软件：Web服务器与浏览器
- 1993年，安德雷生发明马赛克Web浏览器
  - 1994年成为网景浏览器（Netscape）

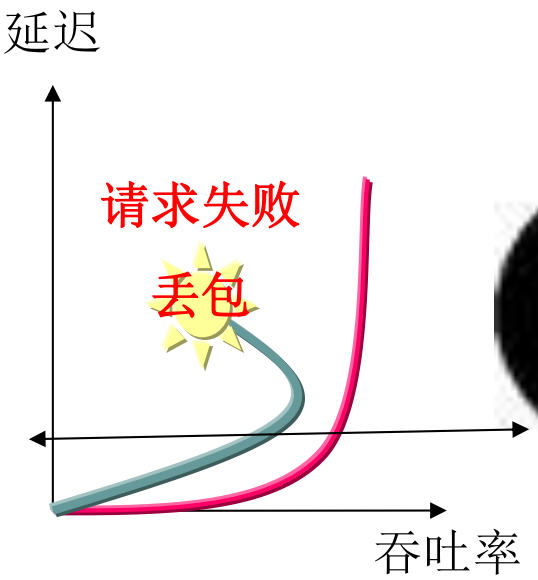
社交网络

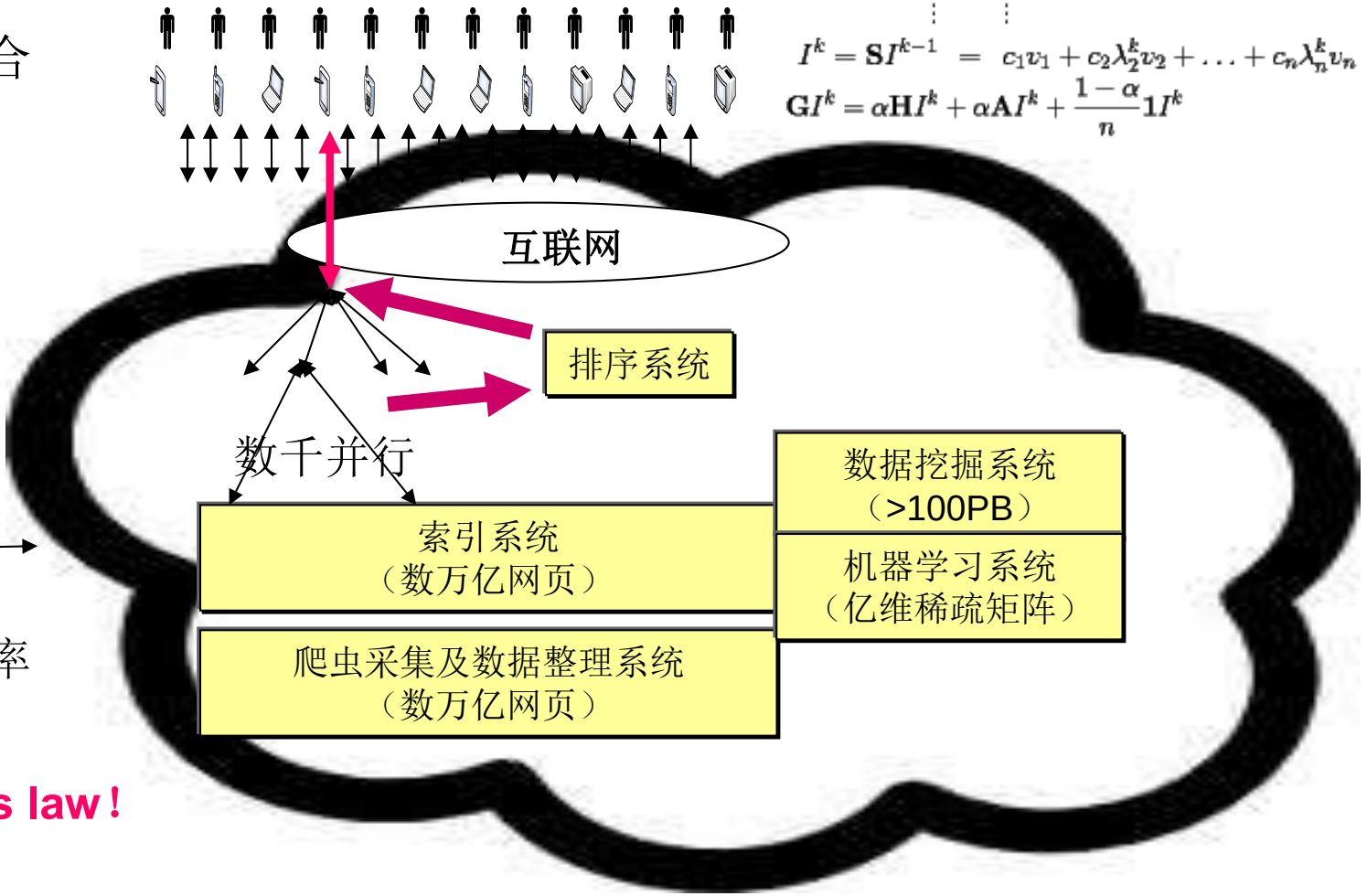
- 2003年11月19日，Harvard Crimson
  - 大学管委会决定放二年级学生马克•扎克伯格（Mark Zuckerberg）一马，不予他离校处分
  - Facemash选“辣的”同学，未经允许使用同学照片，涉嫌破坏计算机网络安全、侵犯版权和侵犯个人隐私
- 2004年2月4日，脸书网（也就是今天的Facebook社交网络服务）正式上线
  - 扎克伯格可能已经满足了“**1万小时定律**”
- 2012年，脸书公司在纳斯达克股票市场上市，同年用户数增长到9亿人；2014年14亿人
- 与MySpace不同，密切参加开源社区（如Hive）

互联网搜索业务系统：在线+离线计算

- 在线性能指标：Amazon三元组
  (最大请求数,质量百分位,响应时间)
  =(百万, 99.9%, 10微秒）
- 离线指标：一天
- 两种模式相互配合

$$I^0 = c_1 v_1 + c_2 v_2 + \ldots + c_n v_n$$
$$I^1 = \mathbf{S} I^0 = c_1 v_1 + c_2 \lambda_2 v_2 + \ldots + c_n \lambda_n v_n$$
$$I^2 = \mathbf{S} I^1 = c_1 v_1 + c_2 \lambda_2^2 v_2 + \ldots + c_n \lambda_n^2 v_n$$
$$\vdots \quad \vdots$$
$$I^k = \mathbf{S} I^{k-1} = c_1 v_1 + c_2 \lambda_2^k v_2 + \ldots + c_n \lambda_n^k v_n$$
$$GI^k = \alpha \mathbf{H} I^k + \alpha \mathbf{A} I^k + \frac{1-\alpha}{n} \mathbf{1} I^k$$



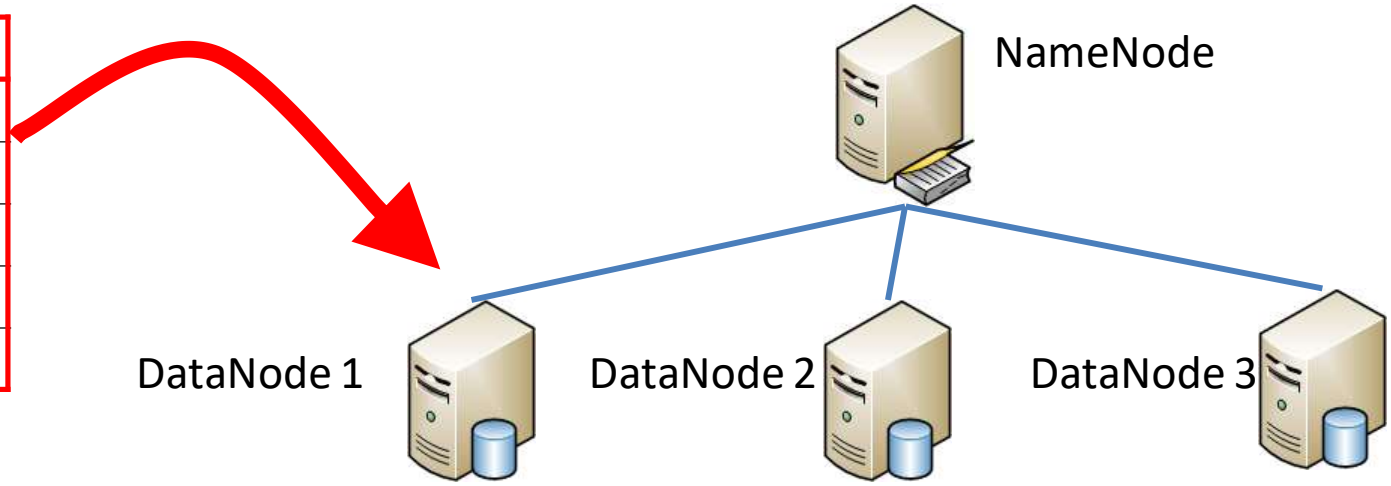


实际情况差于Little's law！

网络计算系统的一个实例：RCFile

- 大数据计算的数据放置问题
  - 如何将100-1000PB的 数据放置到数千个计算结点中，使得离线数据挖掘计算所需存储空间最小、速度最快？

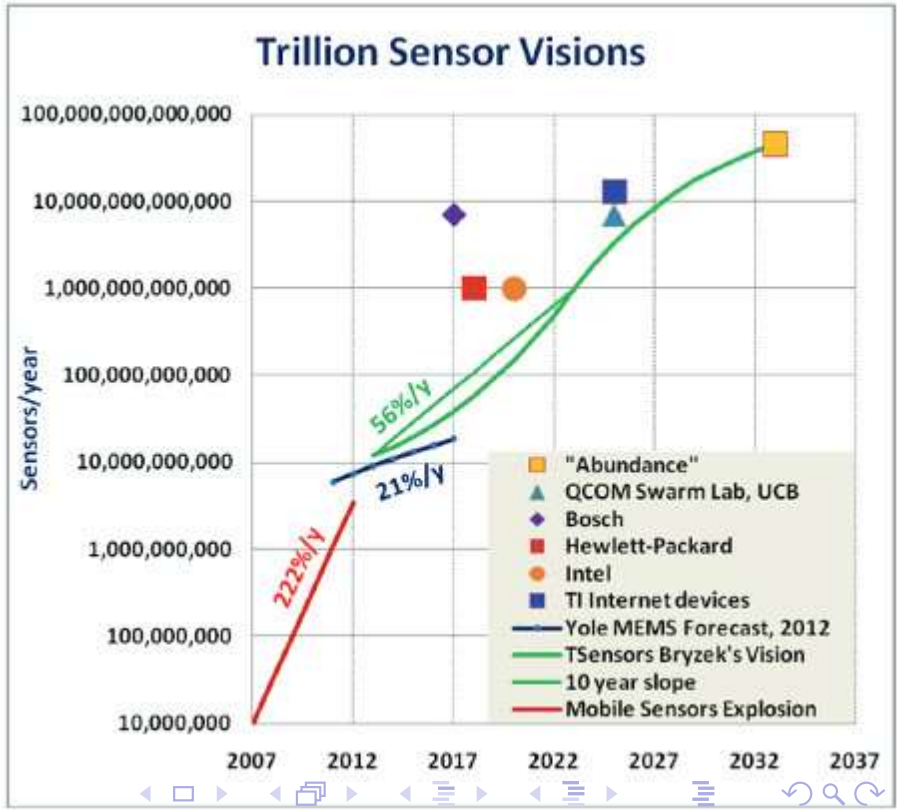
A	B	C	D
101	201	301	401
102	202	302	402
103	203	303	403
104	204	304	404
105	205	305	405



- 2010年，中科院计算所何永强同学
  - 发明行列混合存储技术RCFile
  - 开源到Apache Hive社区，全球使用
  - **http://en.wikipedia.org/wiki/RCFile**

未来十五年信息技术发展趋势研判

- 继桌面互联网、移动互联网之后，信息技术正在进入第三个宏观阶段，万物互联网成为重要方向
  - 互联网→移动互联网→万物互联网（IoE）
  - **IoE: Internet of Everything**
    - **Everything = People + Data + Processes + Things**
- 多个预测：
  - 2030年，全球将有千亿~万亿传感器，数百亿物端设备；每个设备都有处理器、OS、开发环境、使用模式



物端产业生态尚未变成红海

- 产业界：很多小批量产品与方案；无主流生态
- 昆虫纲悖论