浙江大学
ZHEJIANG UNIVERSITY

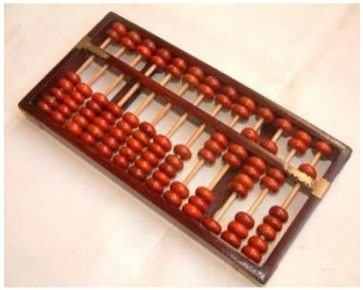
系统结构与网络安全研究所

# 计算机组成与设计
# Computer Organization & Design
## The Hardware/Software Interface

# Preface

林 芃
Lin Peng
penglin@zju.edu.cn

# Which one is a Computer?



# Curriculum System：三位一体、循序递进

立足基础、加强实践、服务专业、进入国际

- **《数字逻辑课程》：计算机组成相关部件的设计** — 基础
  - 组合电路设计、时序电路设计
- **《计算机组成》：设计简单RISC-CPU核** — 核心
  - ALU部件
  - **单周期实现、多周期实现**简单的32位-CPU
    - RISC
    - 写入FPGA，用实验板卡做测试验证
  - 简易计算机系统（微控制系统）
- **《计算机系统结构》：设计流水线RISC-CPU核心** — 提高

# Some Keywords

## You probably know

- Truth table, logic gates
- Combinational circuits, Adder, MUX
- Sequential circuits, Latch

## You will learn in this course

- Instruction Set
- Pipelines
- Cache
- Datapath
- Virtual Memory

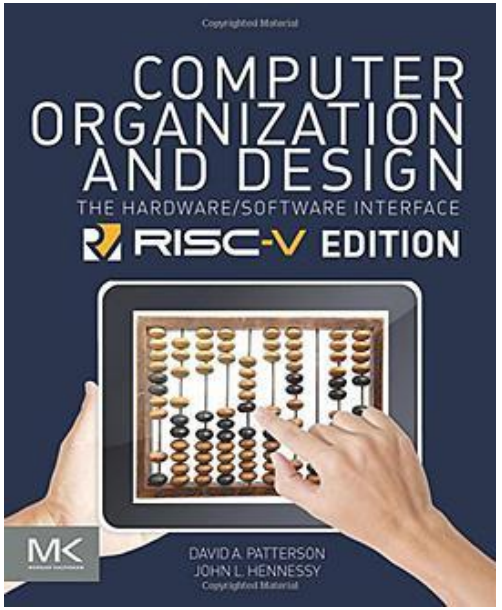
# Text Book

**Computer Organization & Design (MIPS, ARM, RISC-V)**
**——The Hardware/Software Interface**


**John L. Hennessy (Stanford University)**

**David A. Patterson (University of California, Berkeley)**





# Schedule

先讲第三章

| | | |
|---|---|---|
| 1 | Chapter1: Computer abstractions and Technology | ~1 weeks |
| 2 | Chapter 3: Arithmetic for Computer | ~2 weeks |
| 3 | Chapter 2: Language of the Machine | ~3 weeks |
| 4 | Chapter 4: The processor | ~3 weeks |
| 5 | Chapter 5: Memory Hierarchy | ~3 weeks |
| 6 | Appendix: Storage, Networks and Other Peripherals | ~1 week |

# Schedule of Experiments

| | | |
|---|---|---|
| 1 | **Lab00：Vivado介绍和使用** | Vivado介绍和使用，MUX模块封装 |
| 2 | **Lab01：Warmup** | ALU 和Regfile设计 |
| 3 | **Lab02：CPU实验环境搭建** | IP核集成SOC设计，完成CPU测试环境搭建 |
| 4 | **Lab03：复杂操作实现** | 乘除法、浮点加法实现 |
| 5 | **Lab04：单周期CPU设计** | **CPU核集成**<br>1.CPU数据通路<br>2.CPU控制器<br>3.指令扩展<br>4.中断 |
| 6 | **Lab05：流水线CPU设计** | **流水线处理器集成**<br>1.IF-ID设计与集成<br>2.EX-MEM-WB设计与集成<br>3.冒险与stall |

# Course Grading

- **平时占 20%**
  - 作业、课堂、阅读
- **期中闭卷(英文)考试(统一时间段)10%**
- **期末闭卷(英文)考试 40%**
- **实验占 30%**
  - Lab00~03: 基本实验 30%
  - Lab04 :单周期CPU 30%
  - Lab05 : 流水线CPU 40%
  - 附加分奖励：优秀作品 (附加后不超过100分)

**注：卷面成绩不到40分(满分100分)者，总评不及格。**

# One more thing: Attend on time

## 单周

| | 周一 | 周二 | 周三 | 周四 | 周五 |
|---|---|---|---|---|---|
| 第二节 | | | | | |
| 第三节 | | 西2-504<br>10:00-12:25 | | | |
| 第四节 | | | | | |
| 第五节 | | | | | |
| 午休 | | | | | |
| 第六节 | | 东4-509<br>13:25-15:00 | | | |
| 第七节 | | | | | |
| 第八节 | | | | | |

## 双周

| | 周一 | 周二 | 周三 | 周四 | 周五 |
|---|---|---|---|---|---|
| 第二节 | | | | | |
| 第三节 | | 西2-504<br>10:00-11:35 | | | |
| 第四节 | | | | | |
| 第五节 | | | | | |
| 午休 | | | | | |
| 第六节 | | 东4-509<br>13:25-15:00 | | 西2-504<br>13:25-15:00 | |
| 第七节 | | | | | |
| 第八节 | | | | | |

浙江大学 ZHEJIANG UNIVERSITY
系统结构与网络安全研究所

# 计算机组成与设计
# Computer Organization & Design
## The Hardware/Software Interface

# Chapter 1
# Computer abstractions and Technology

林 芃
Lin Peng
penglin@zju.edu.cn

# Outline

- Introduction
- Computer organization
- How to build processors ?
- Computer design: performance and idea
- What you can learn from this course ?

# Outline

- **Introduction**
- Computer organization
- How to build processors ?
- Computer design: performance and idea
- What you can learn from this course ?

# History of Computers

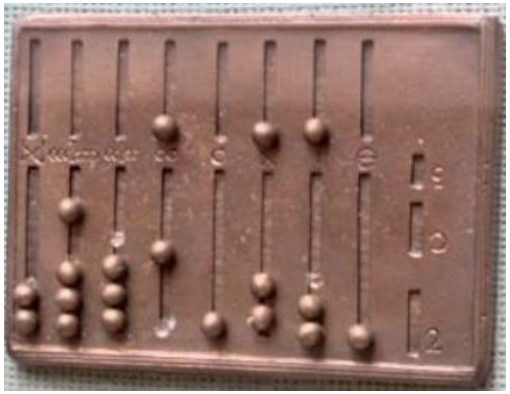
- **Pre-computer – ~ 1946**
  - Non-electrical, non-programmable (非电子，不可编程)
- **First Generation – 1946 ~ 1956**
  - Vacuum tubes, programmable (电子管，可编程，图灵完全)
- **Second Generation – 1956 ~ 1964**
  - Transistor, programming languages (晶体管，编程语言开始应用)
- **Third Generation – 1964 ~ 1971**
  - Integrated Circuit, OS (集成电路，操作系统开始应用)
- **Fourth Generation – 1971 ~ now**
  - Microprocessor, GUI, Personal Computer

  (大规模集成微处理器, 图形用户界面, 个人电脑兴起)
- **Fifth Generation – future**
  - Still in development … …

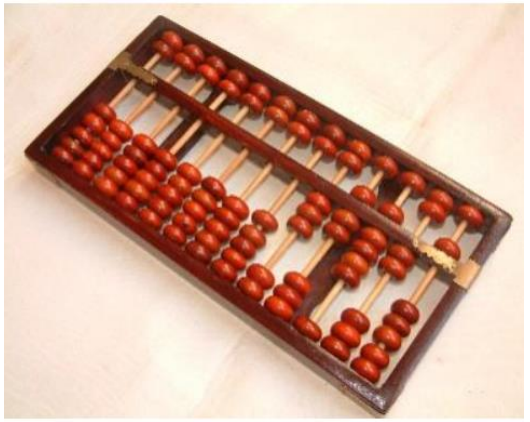
# Pre-computers：计数的辅助工作

Digit: Latin for fingers

Tally sticks



Roman abacus



Asian abacus

# The First Mechanical Calculating Machine

**The 17th century: the beginning of mechanical calculators**

**17世纪：机械式计数器开始出现**

Blaise Pascal (1623-1662)

## Pascaline (Pascal's calculator)

- Invented by Blaise Pascal **in 1642-1645 (19-22岁！)**
- Addition and Subtraction: Two basic operations

  **两个基本操作：加与减**
- Multiplication and division: through repeated addition or subtraction

  **乘与除这两个操作通过重复地加或减间接实现**

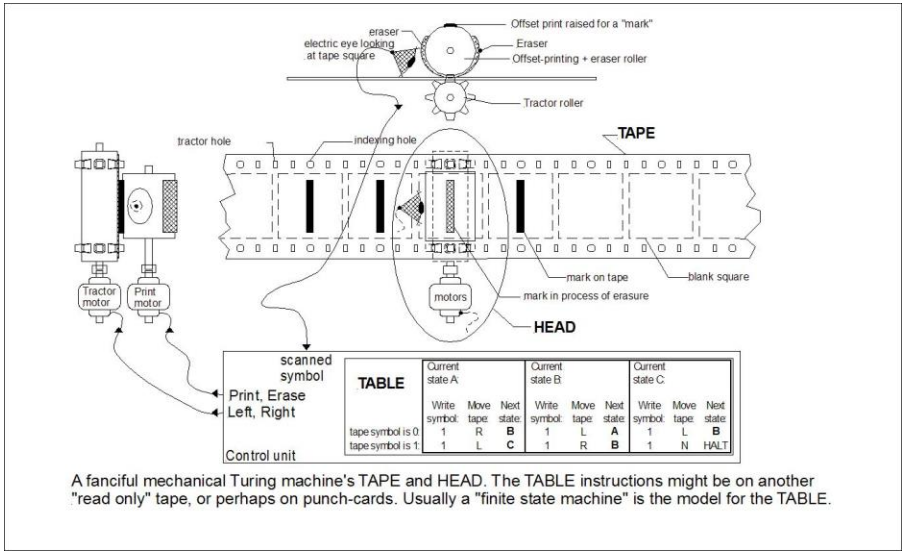
**Pascal编程语言取名就是为了纪念Blaise Pascal**

Pascaline (1645)

# Turing Machine (1936)

**Alan Turing** described a theoretical device called the Turing machine, formalized the concepts of computation and algorithms **in 1936**

A.M.Turing (1912-1954)




A fanciful mechanical Turing machine's TAPE and HEAD. The TABLE instructions might be on another "read only" tape, or perhaps on punch-cards. Usually a "finite state machine" is the model for the TABLE.

Turing machine as a mechanical device

# First Electronic Digital Computing Device

## Atanasoff-Berry Computer(ABC) 1942

Add-subtract module
加减运算模块

- First electronic digit computing device 第一台纯电子的数字计算设备
- Invented by Professor **John Atanasoff** and graduate student Clifford Berry at Iowa State College between **1939 and 1942**
- Special purpose. neither programmable, nor Turing-complete

非常接近现代计算机，但**不可编程、非图灵完全**

**对现代计算机发展的重要贡献**

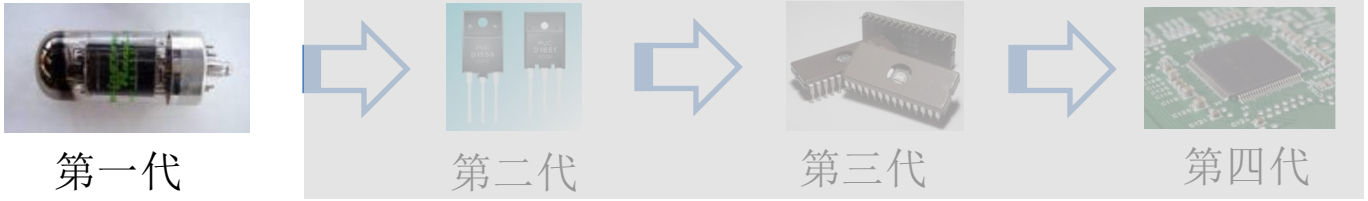
- 使用二进制数表示
- 用纯电子实现算术与逻辑运算
- 计算和存储分离

John Atanasoff (1903-1995)

Clifford Berry (1918-1963)

Atanasoff-Berry Computer (1942)

# The First Generation (1946-1956): Vacuum Tubes




- **基本特征：运算用电子管(即真空管)实现、可编程**

**vacuum tubes** for circuitry and **magnetic drums** for memory, programmable

- Examples:
  - ENIACS
  - EDSAC
  - UNIVACI, UNIVAC II, UNIVAC 1101


**Vacuum tubes**

# First General-Purpose Electronic Computer

**ENIAC (Electronic Numerical Integrator and Computer) in 1946**

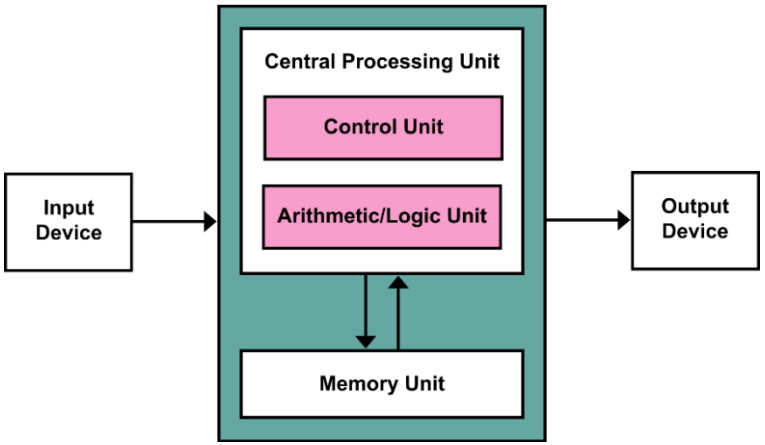
- First general-purpose electronic computer **第一台通用电子计算机**
- Designed by **John Mauchly** and **J. Presper Eckert** of the University of Pennsylvania
- General-purpose instruction set, decimal, programmable & Turing-complete

  **通用指令集、十进制、可编程、图灵完全**
- **But NOT stored program 没有程序存储能力**

# von Neumann Architecture (1945)

- Computation and memory are separated
  **计算与存储分离**
- Memory that stores data and instructions
  **数据与指令保存在同一个存储器**
- Input and output mechanisms
- Instruction set architecture

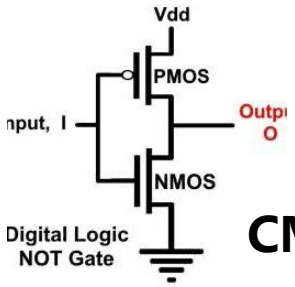
John von Neumann (1903-1957)



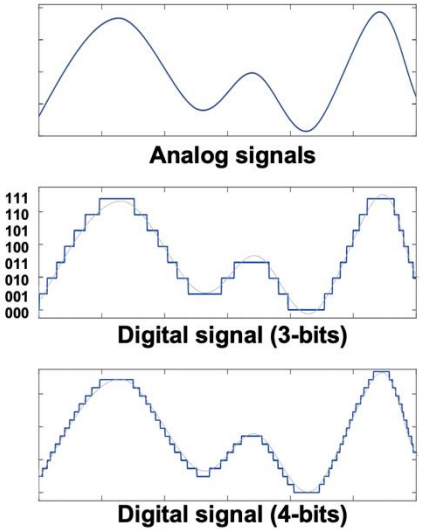

"von Neumann Bottleneck"

# Analog Computing

| Digital computing | Analog computing |
|---|---|
| **Accurate**<br>when encoded in many bits | **Less precision**<br>affected by noise |
| **Scalable**<br>reliable CMOS circuits with billions of transistors | **Difficult to integrate**<br>Noise and error accumulate in cascaded circuits |
| **General purpose** | **Application-specific** |
| **Computational heavy** | **Fast by nature** |
| **Compute-only** | **Sensors and actuators are mostly analog** |



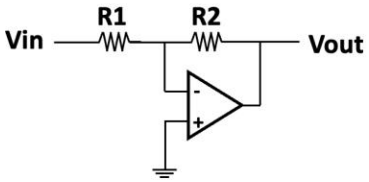

**CMOS gates**




**Resurgence of analog computing in AI applications to overcome von Neumann bottleneck and to emulate brains (neuromorphic)**

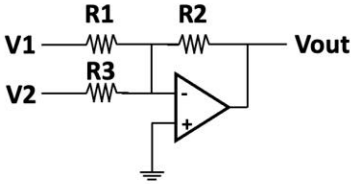
# Analog Computing

**Inverter**



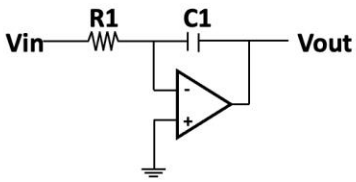

$V_{out} = -\frac{R_2}{R_1} V_{in}$

**(Inverting) Summation**



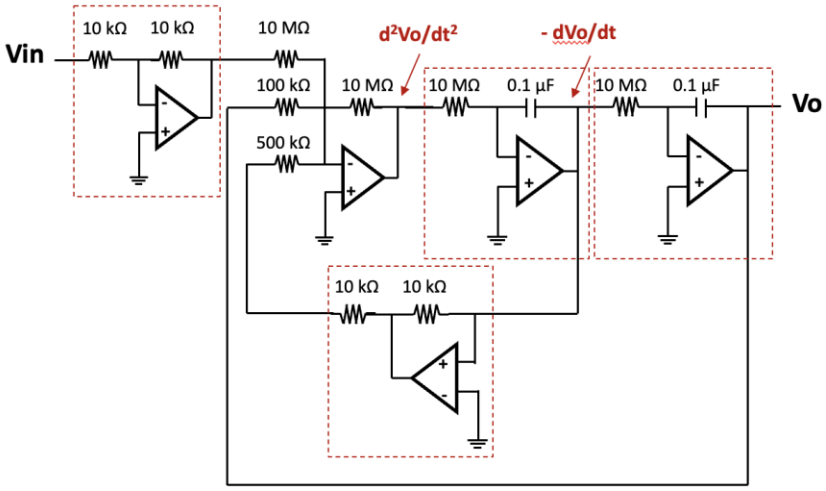

$V_{out} = -\left(\frac{R_2}{R_1} V_1 + \frac{R_2}{R_3} V_2\right)$

**(Inverting) Integration**

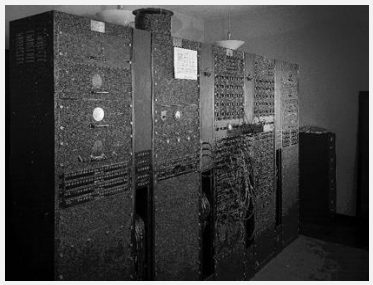



$\frac{dV_{out}}{dt} = -\frac{1}{R_1 C_1} V_{in}$

## Computing Circuits




$$Vin - 100\,Vo - 20\,\frac{dVo}{dt} = \frac{d^2Vo}{dt^2}$$

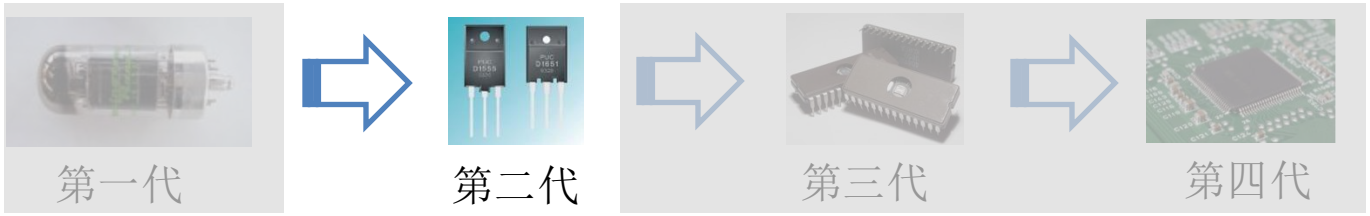


**REAC Analog Computer**

US Navy, Project Cyclone, 1945

"In a typical application, such as the simulation of a guided missile in three dimensions, the average runtime for a single solution on the analog computer facility was approximately one minute. The check solution by numerical methods on an IBM CPC (Card Programmed Calculator) took 75 hours to run; on an Elecom 100 it took from 60 to 130 hours to solve the same problem. "

# The Second Generation (1956-1964): Transistors




- **基本特征：晶体管替代电子管**

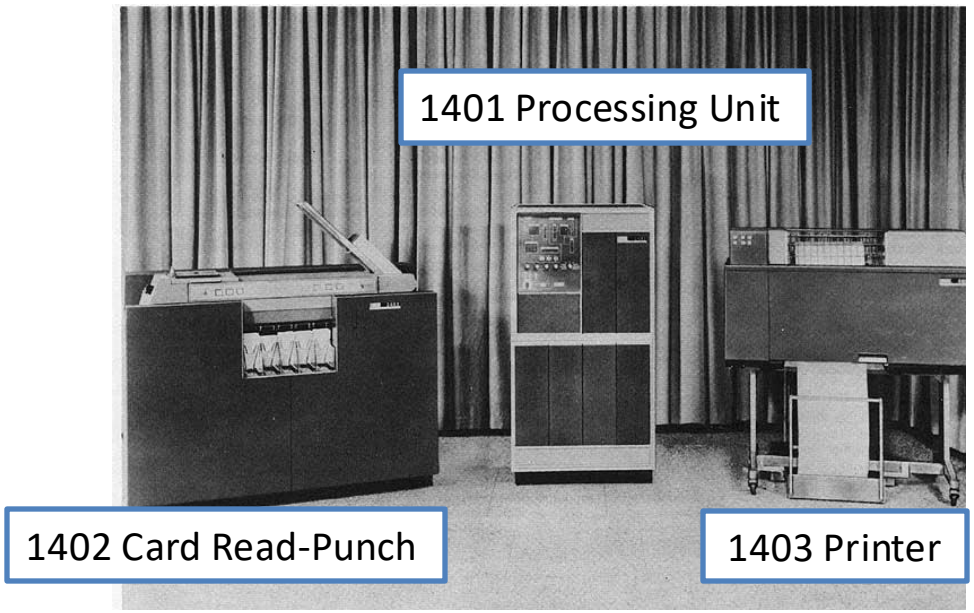
  **Transistor** replace **Vacuum tubes**
  - **Smaller, faster, less power, more reliable**

    **体积减小、速度加快、计算更可靠**
- **Programming**: from binary machine language **to symbolic languages**
- **I/O**: Punched cards for input and printouts for output.
- **Examples**: UNIVAC III, RCA 501, Philco TransactS-2000, NCR 300 series, IBM7030 Stretch


Transistors
晶体管

# IBM 1401 (1959)

- announced by IBM on October 5, 1959.
- a variable-wordlength decimal computer **(可变字长、十进制)**
- installed more than 10,000 units between 1960 and 1964. **商业上大获成功**



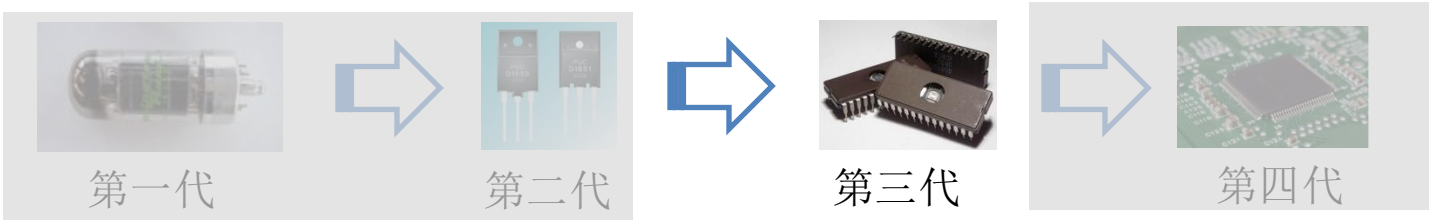

IBM 1401 system

IBM 1401

# UNIVAC III (1962)

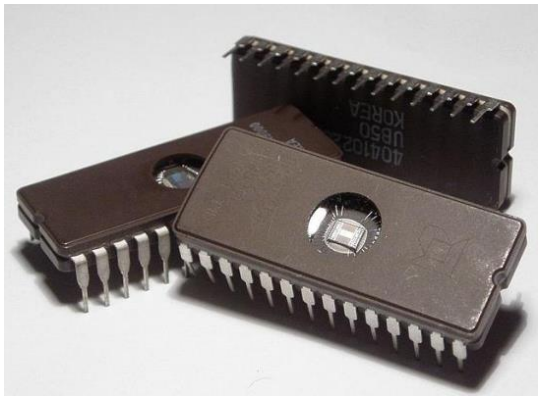
- **UNIV**ersal **A**utomatic **C**omputer **III (1962)**
- Designed by **John Perper Eckert** and **John Mauchly** (ENIAC designers)
- An improved transistorized replacement for UNIVAC I and UNIVAC II

# The Third Generation (1964-1971): Integrated Circuits



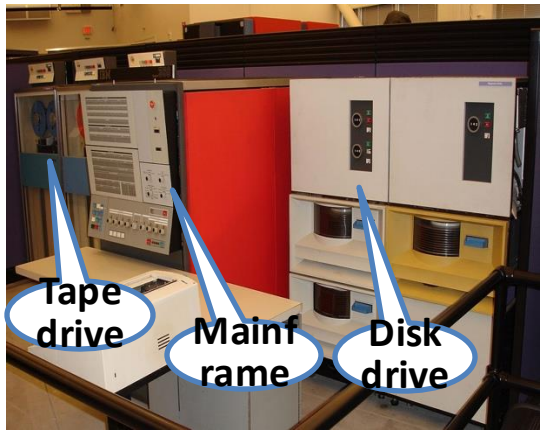

- **基本特征：集成电路替代晶体管**

  **Integrated Circuits** Replaced Transistors
  - smaller and cheaper than their predecessors (**更小、更便宜**)
  - Electronic components on single chips of silicon
- **Operating systems**： run many different applications at one time
- **Examples**: IBM System/360, Honeywell 200



# IBM System/360 Family (1964)

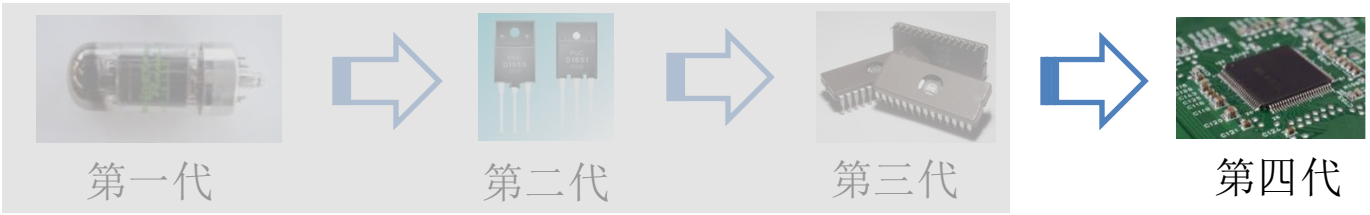
- **Announced in 1964, delivered in 1965**
  - Model 30: up to 34,500 instructions per second, with memory from 8 to 64 KB.
  - Model 91 in 1967: up to 16.6 million instructions per second, with memory from 1M to 4M
  - Extremely successful in the market **商业上极为成功**



IBM System/360 Model 30

- **Backward compatibility 向前兼容**
- **Computer System Architecture Specification**
  - Chief Architect: Gene Amdahl (NAE in 1967)
- **A Single Operating System for All: OS/360**
  - Dynamic address translation (virtual memory) 虚拟内存
  - Time-sharing 分时

# The Fourth Generation (1971-now): Microprocessors




- **基本特征：微处理器替代集成电路**

  Microprocessors replaced Integrated Circuits

  - smaller, cheaper, more powerful

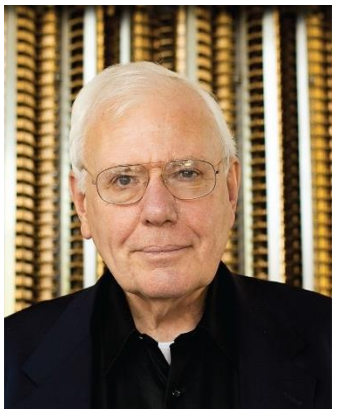

- **Microprocessor:** entire CPU fits on a single chip.
  - Three companies developed the microprocessor independently at the same time: Texas Instruments, Intel, and Garrett AiResearch **in 1971**
- **Emerging of Personal Computers 个人计算机兴起**

# The First Modern Personal Computer

## Xerox Alto in 1973 at Xerox PARC

- **designed mostly by Charles P. Thacker**
- Mouse-driven Graphical User Interface

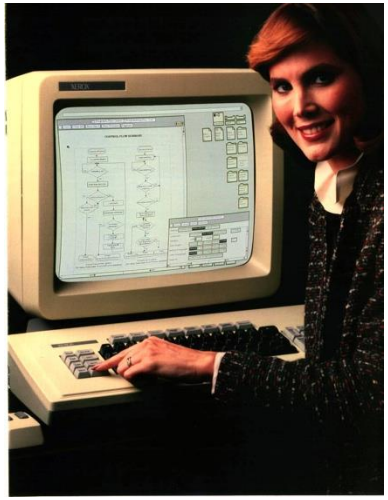
**Turing Award 2009**



Charles Thacker (1943-2017)

Alto Mouse

Xerox Alto (1973)

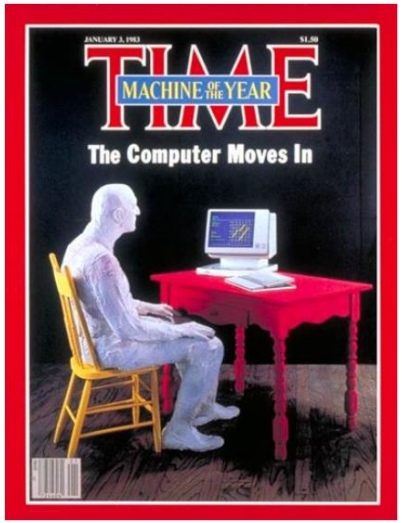


Xerox Star (1981)

就对计算机科学的贡献而言，我想不出有谁能够抵得上 Chuck Thacker，这使得我在听到他得到图灵奖后非常高兴。人们现在往往把个人电脑视作理所应当，而Chuck则是一个真正能够认识到它的潜能的人。——比尔·盖茨

# Commercial Personal Computer

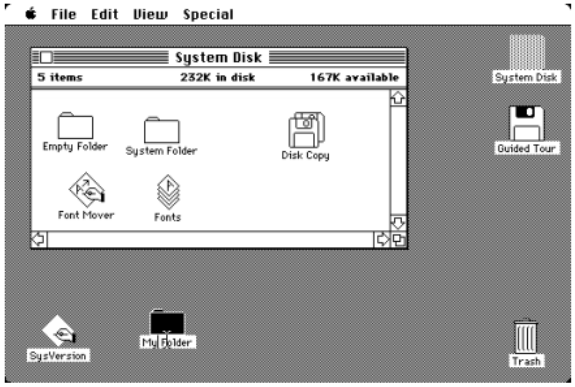
- **IBM PC**: with an **Intel CPU & Microsoft's DOS** in **1981**
  - Price started at $1,565
  - 300,000 sold in 1982
  - **3,274,000 sold in 1982**



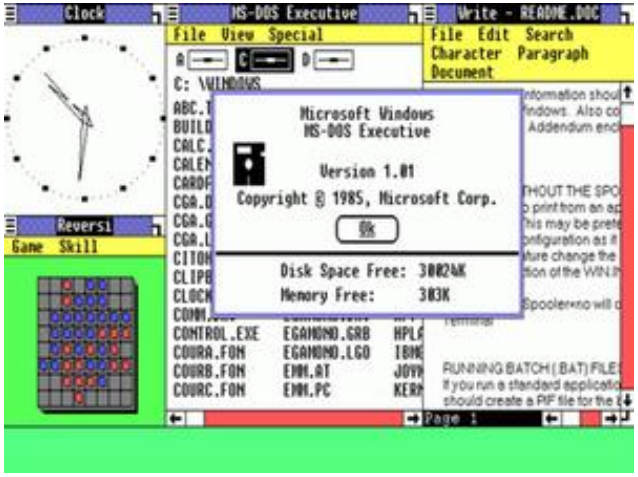

TIME named in 1982

**Machine of the year**

- Apple (Steve Jobs) developed **Macintosh OS** with GUI in **1984**




- Microsoft (Bill Gates) released **Windows 1.0 in 1985**, the first version.



# RISC Architecture (1980s)：更高的性能

- **RISC (Reduced Instruction Set Computer): a computer instruction set**
  - **fewer cycles per instruction (CPI)** than a Complex Instruction Set Computer (CISC).
    指令执行用尽量少的时钟周期、指令编码长度定长等 → 提高CPU与编译效率
- **Now used in 99% of new computer chips, e.g. smartphones**

**现在绝大部分芯片已采用RISC架构，如ARM芯片、智能手机**

**Turing Award 1987**

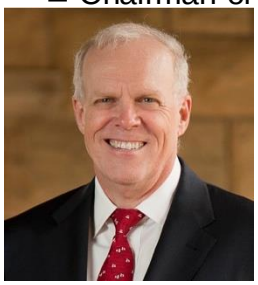
1975-1980: **IBM 801**
**The 1st RISC system**
→ IBM **PowerPC**
■spent his entire career as an industrial researcher for IBM in1956-1992.

John Cocke
(1925-2002)

**Turing Award 2017**

MIPS project of a Stanford graduate course in 1981

→ found **MIPS company** in 1985
■ President of Stanford Univ. (2000-2016)
■ Chairman of Alphabet



John L. Hennessy
(1952-)

**Turing Award 2017**

Berkeley RISC project from 1980

→ commercialized as the **SPARC**.
■ one of the innovators of RAID in 1989-1993

David Patterson
(1947-)

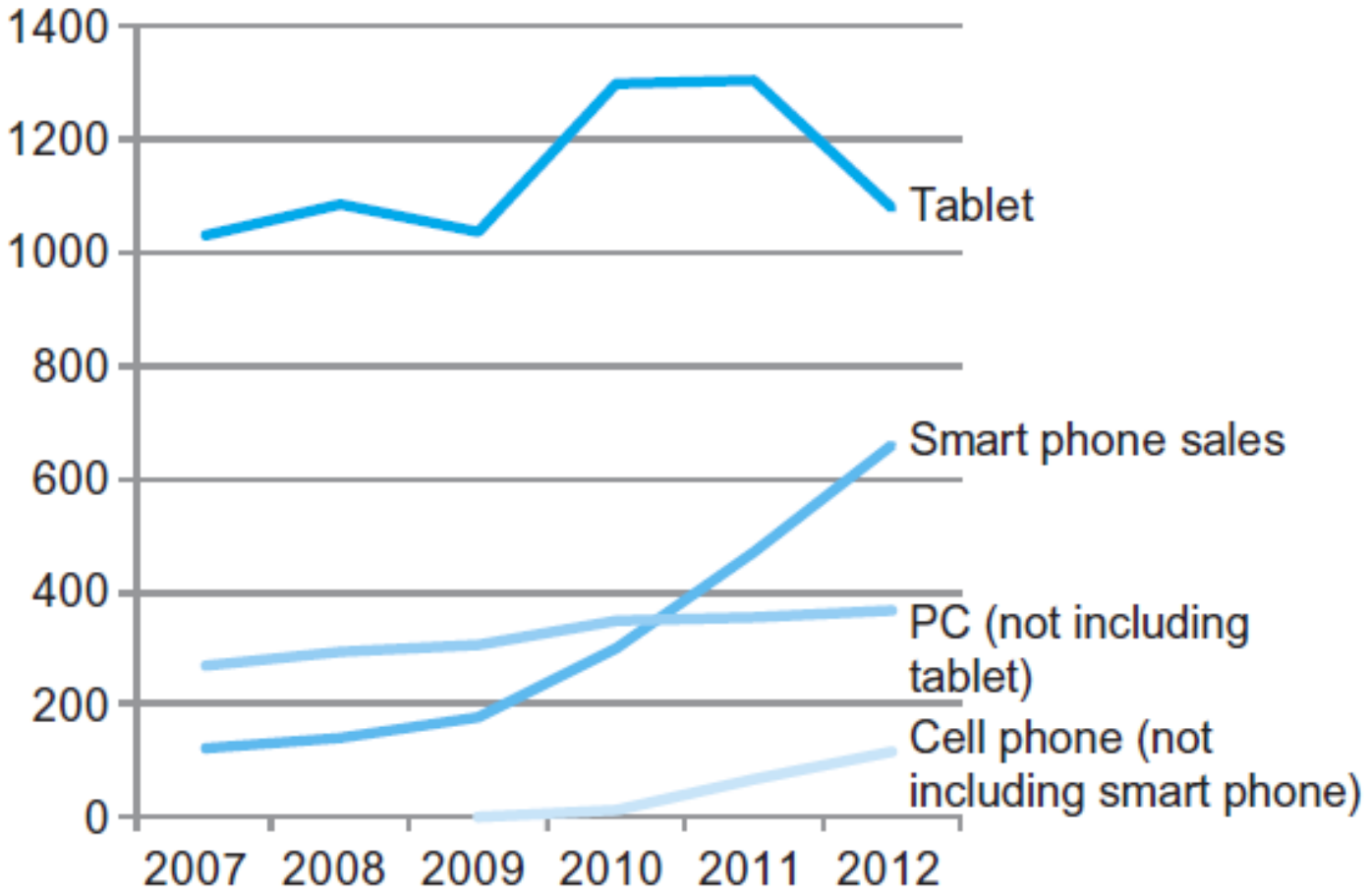
# The Computer Revolution

- **Progress in computer technology**
  - Underpinned by Moore's Law
- **Enable novel applications**
  - Computers in automobiles
  - Cell phones
  - Human genome project
  - World Wide Web
  - Search Engines
- **Computers are pervasive**

# Classes of Computers

- **Personal computers**
  - General purpose, variety of software
  - Subject to cost/performance tradeoff
- **Server computers**
  - Network based
  - High capacity, performance, reliability
  - Range from small servers to building sized
- **Supercomputers**
  - High-end scientific and engineering calculations
  - Highest capability but represent a small fraction of the overall computer market
- **Embedded computers**
  - Hidden as components of systems
  - Stringent power/performance/cost constraints

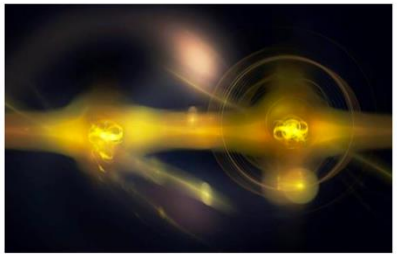
# The PostPC Era



# The PostPC Era

- **Personal Mobile Device (PMD)**
  - Battery operated
  - Connects to the Internet
  - Hundreds of dollars
  - Smart phones, tablets, electronic glasses
- **Cloud computing**
  - Warehouse Scale Computers (WSC)
  - Software as a Service (SaaS)
  - Portion of software run on a PMD and a portion run in the Cloud
  - Amazon ,Google , Alibaba, Huawei

# The Fifth Generation?



后摩尔时代
两大颠覆性计算技术

**量子计算**

**借鉴量子力学原理发展新型计算技术**

**类脑计算**

**借鉴脑科学原理发展新型计算技术**

Finally，subcommittee members are aware of recent announcements from china，Europe and Japan related to Quantum Computing and Neuromorphic Computing that foretell a high level of international competitiveness in the post-Moore Computing era

---美国 Advanced Scientific Computing Advisory Committee， 2019.3.26

重要战略突破口、中美博弈的必争之地

# Neuromorphic Computing

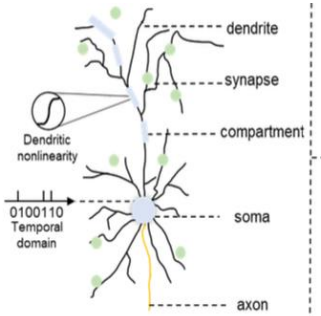
Spiking neural networks (SNN)： 主要的类脑计算算法

计算稀疏　　高能效

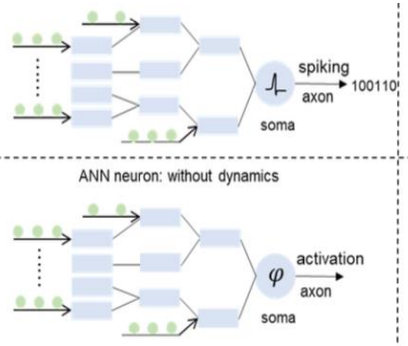
高能效

**神经元网络**
**(脉冲神经网络)**

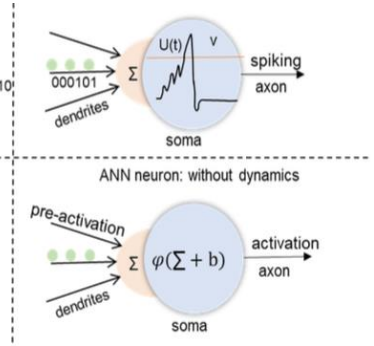
**当前AI网络**
**(人工神经网络)**



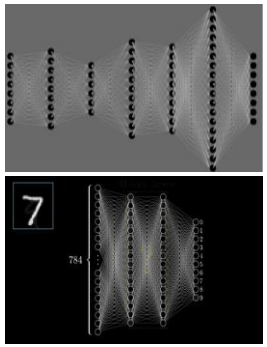

突触及神经元




相同的突触模型

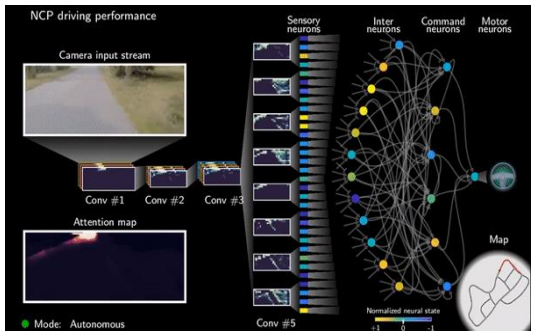



不同的神经元模型



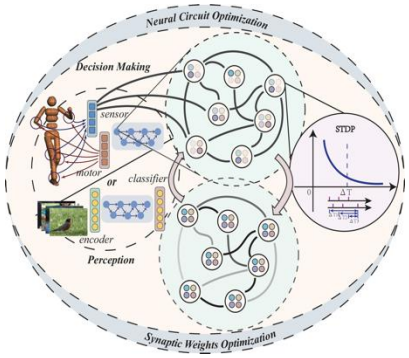

不同的网络表现

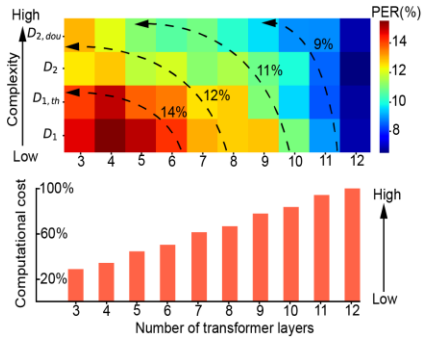
高智能




对线虫的TW神经回路进行建模，利用19个神经元实现自动驾驶中的道路保持功能

Lechner et al, Nat. Mach. Intell. 2020

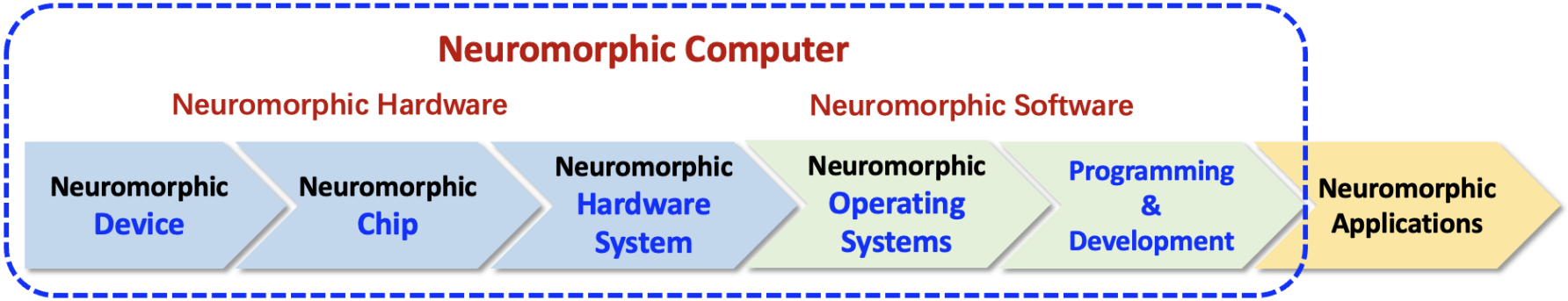



构建包含多种神经环路的类脑循环神经网络，在较低计算开销下实现媲美SOTA准确率

Shen et al. PNAS, 2023, Han et al. arxiv

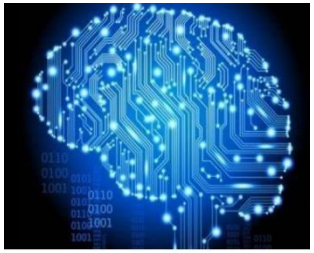
# Neuromorphic Computing

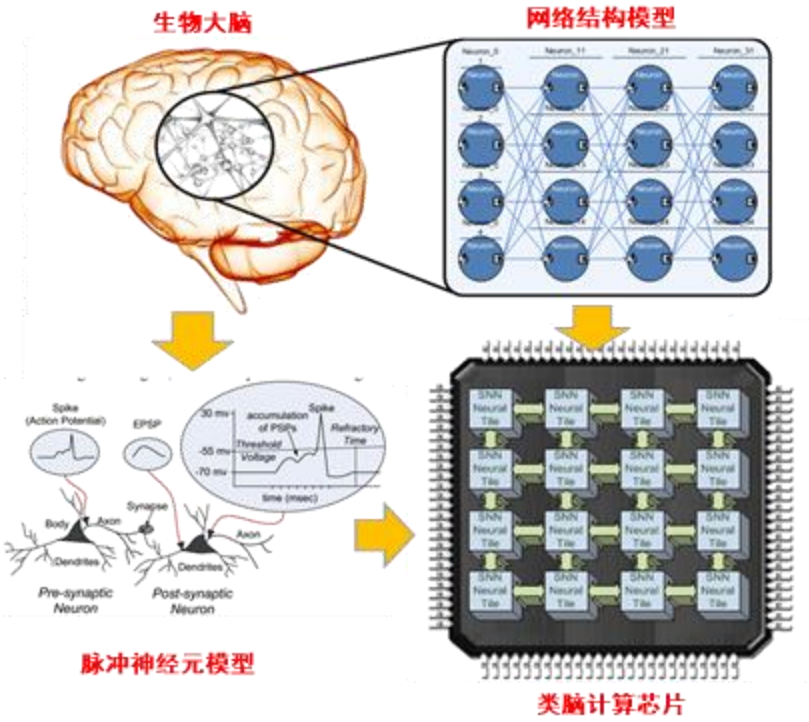



大脑

启发

类脑计算机





# What is a Computer?

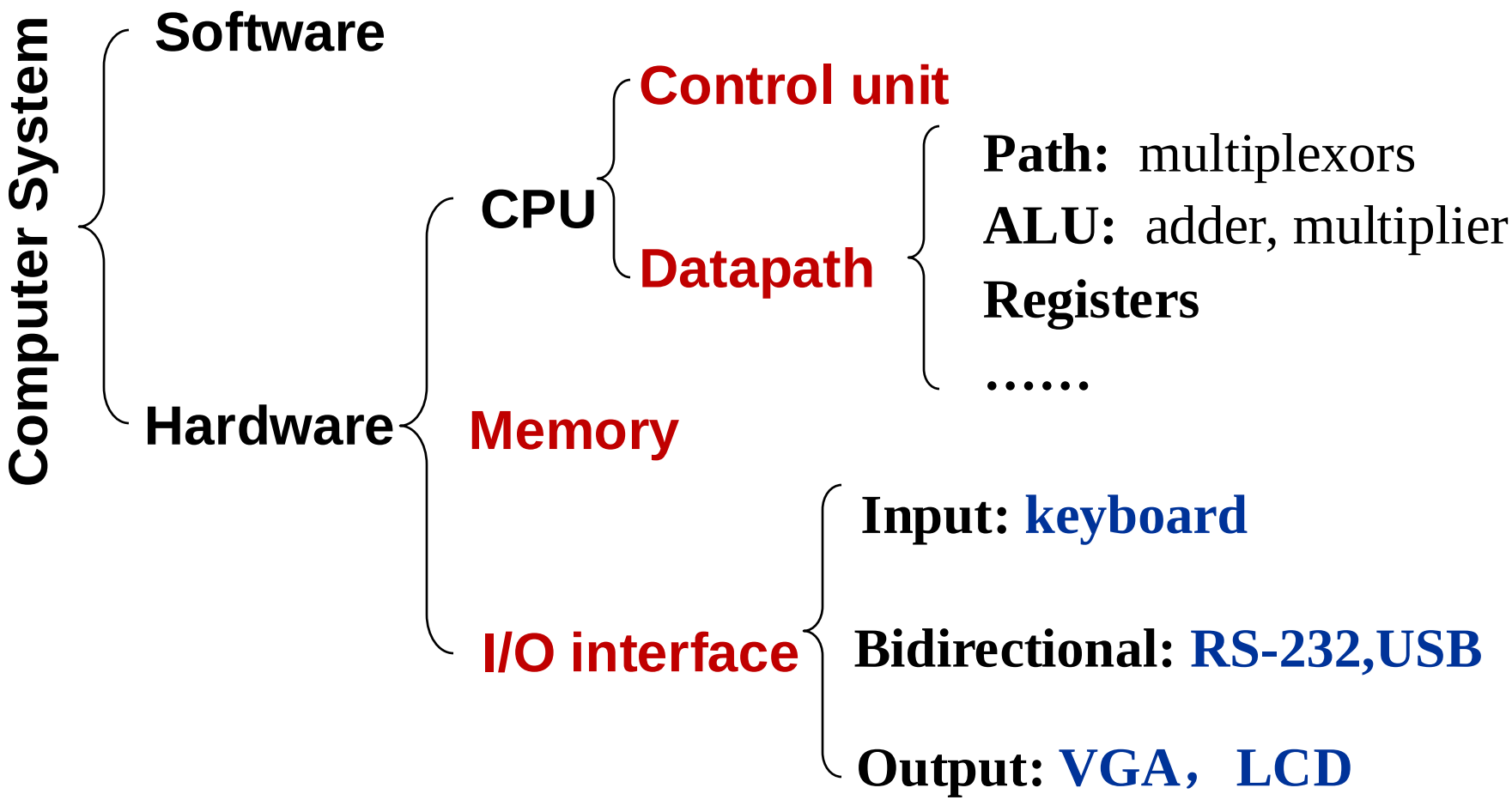
- Computer is an electronic device that manipulates data according to a list of **instructions** (program), with capability of a Turing machine.
  - Electronic realization **电子化的实现方式**
  - A **set of instructions** in a well-defined manner **有指令集** → general-purpose
  - Execution of a **pre-recorded list of instructions** **可执行指令** → program-controlled
  - Memory that can **store** instructions and data **可存储指令与数据** → stored program
  - **Turing-complete** in theory **计算能力上是图灵完全** → equivalent to Turing machine

# Outline

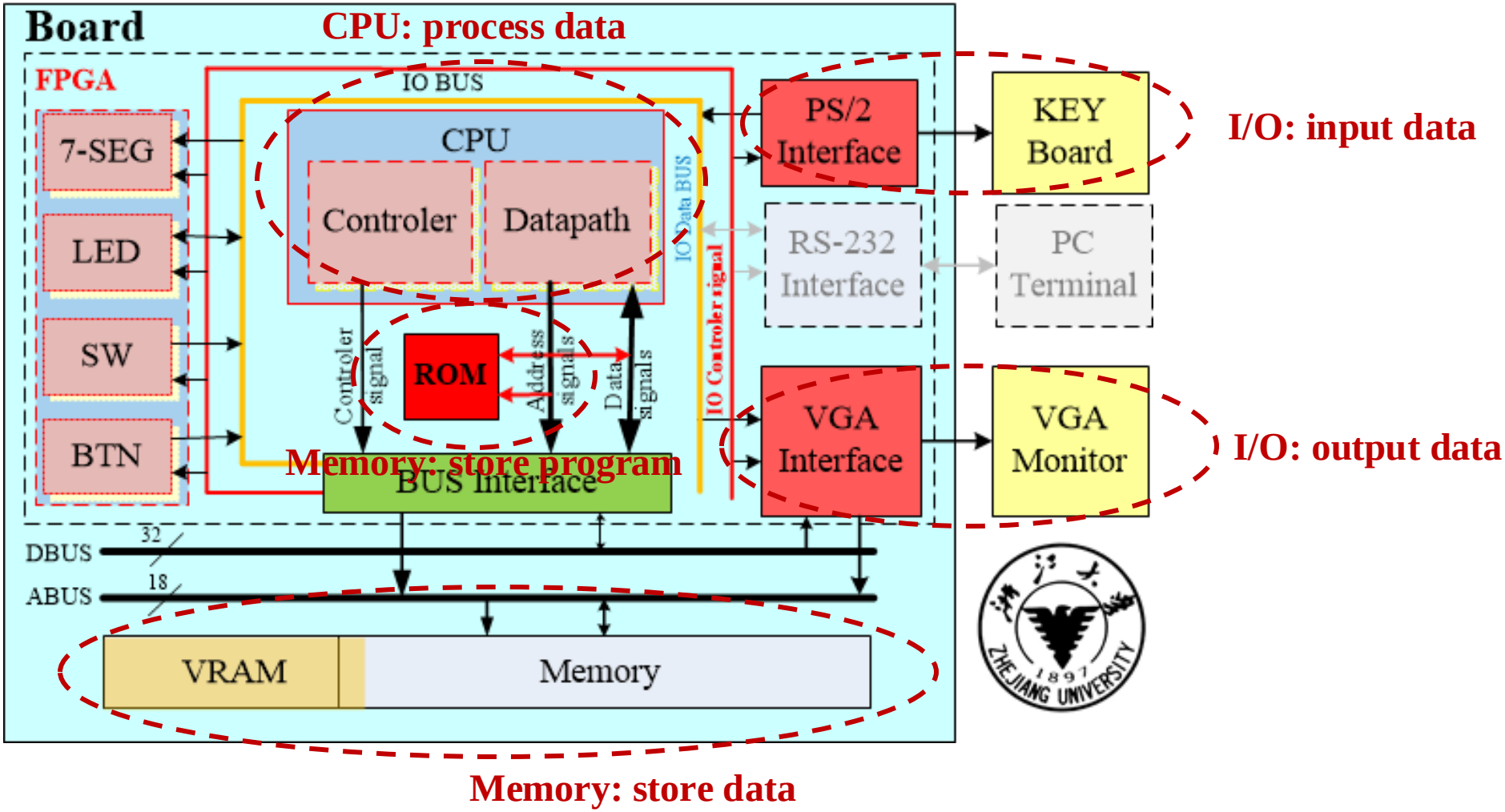
- Introduction
- **Computer organization**
- How to build processors ?
- Computer design: performance and idea
- What you can learn from this course ?

# Computer Organization

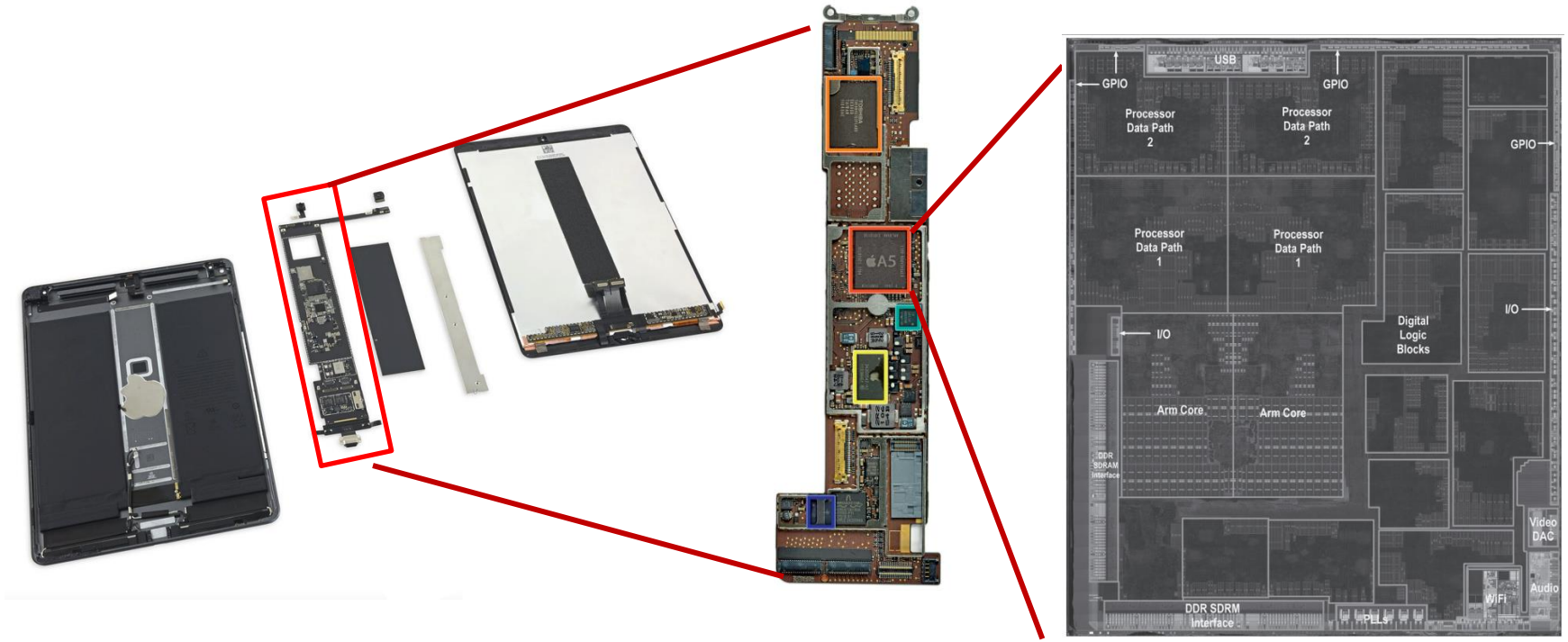
## Decomposability of computer systems



# Five Classic Components of Hardware



# Opening the Box




Components of Apple iPad

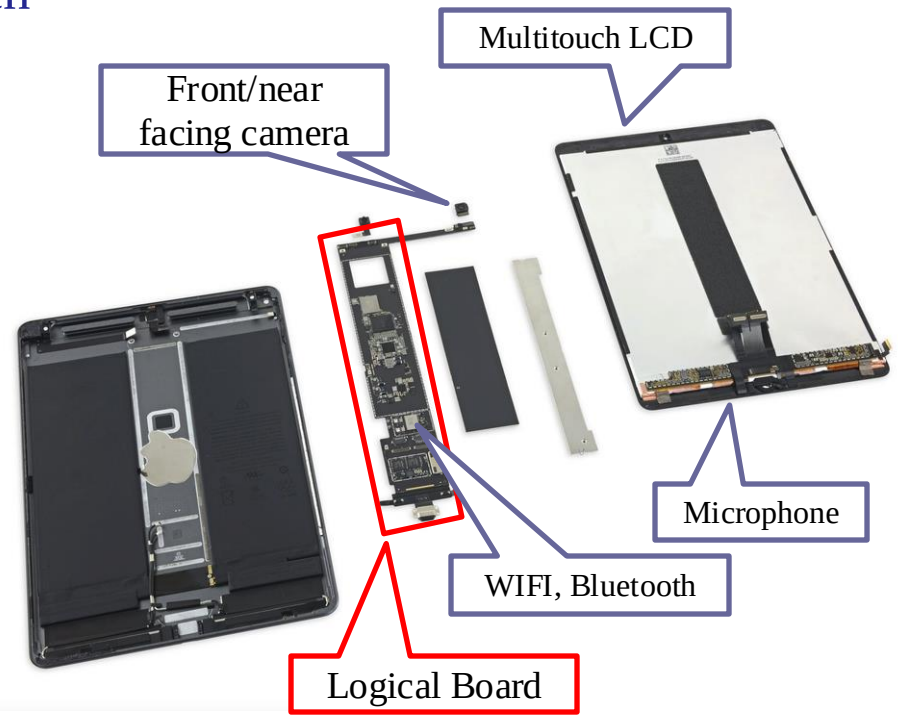
Logical Board (main board:主板)

Integrated Circuit on it

Central Processor Unit (CPU)

# I/O Interfaces

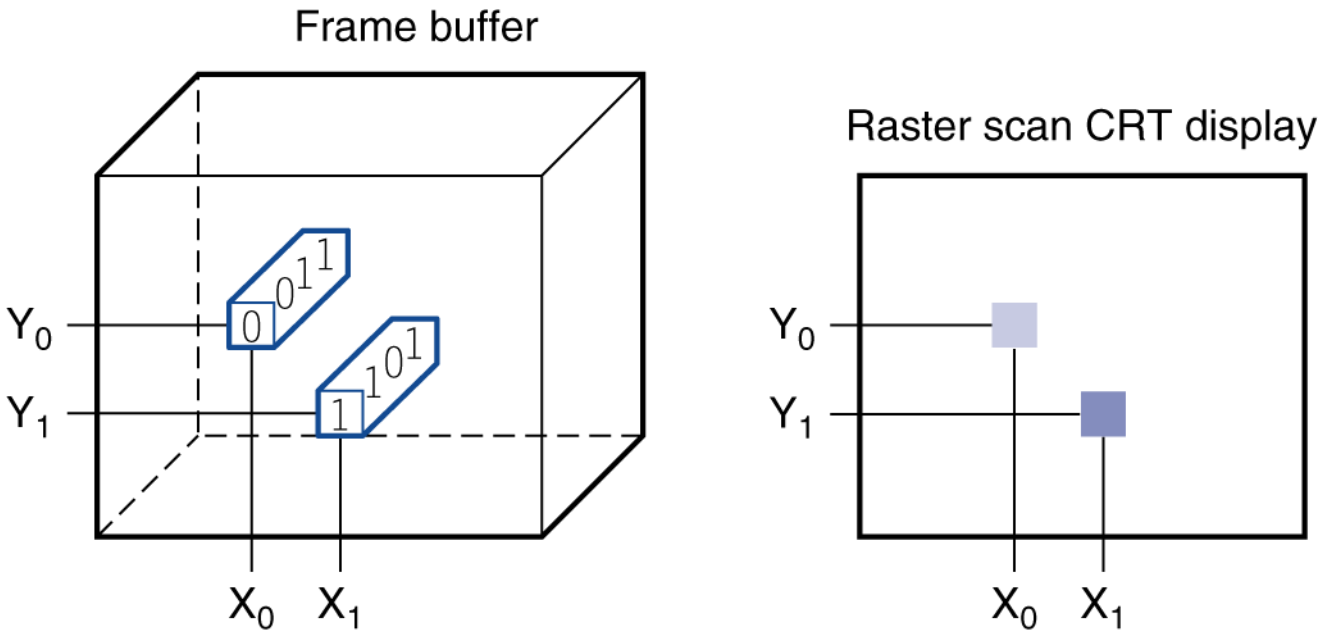
- I/O dominates this device.
- **Input device:** A mechanism through which the computer is fed information
  - Multitouch LCD(触摸屏)
  - Front/near facing camera(前置摄像头)
  - Microphone, headphone(麦克风)
  - Gyroscope(陀螺仪)
  - WIFI, Bluetooth(蓝牙)
- **Output device:** A mechanism that conveys the result of a computer to a user, such as a displayer, or to another computer
  - Speaker(扬声器)
  - Multitouch LCD
  - WIFI, Buluetooth




Components of Apple iPad

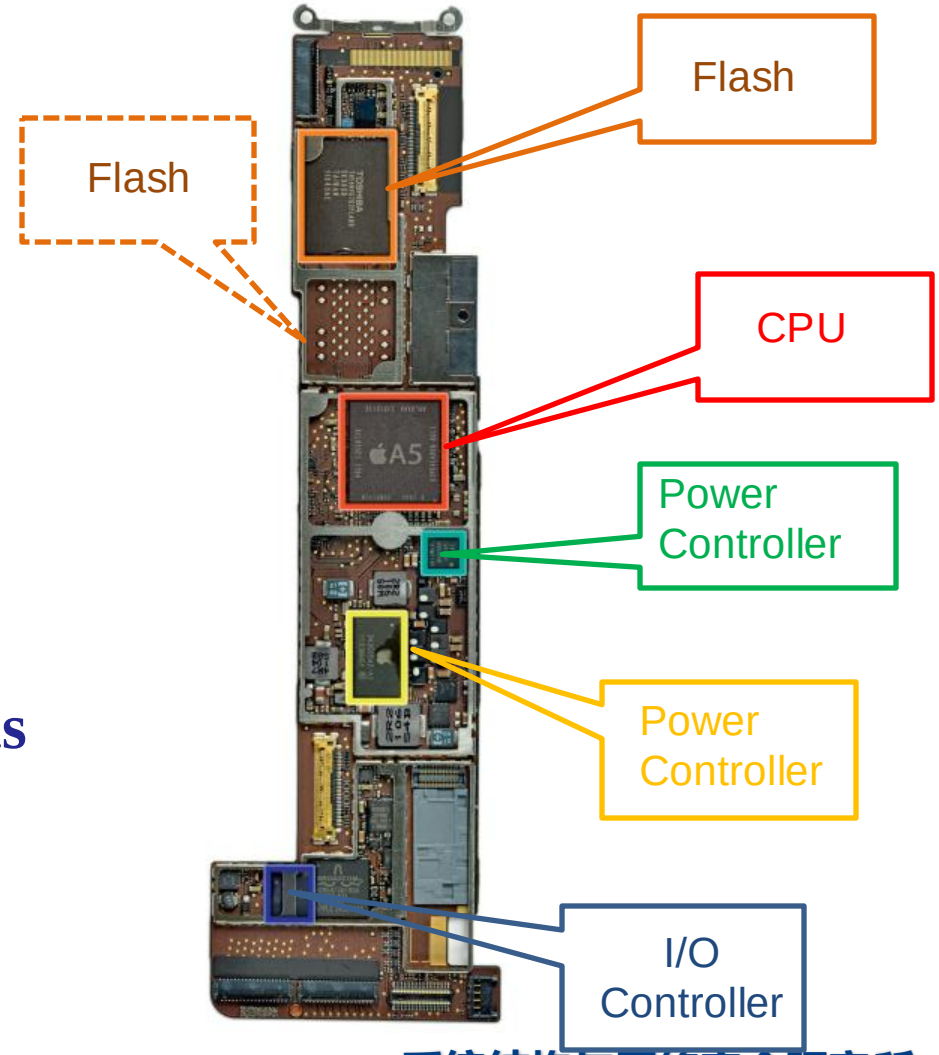
# Through the Looking Glass

- **LCD screen: picture elements (pixels)**
  - Mirrors content of frame buffer memory



# Logical Board

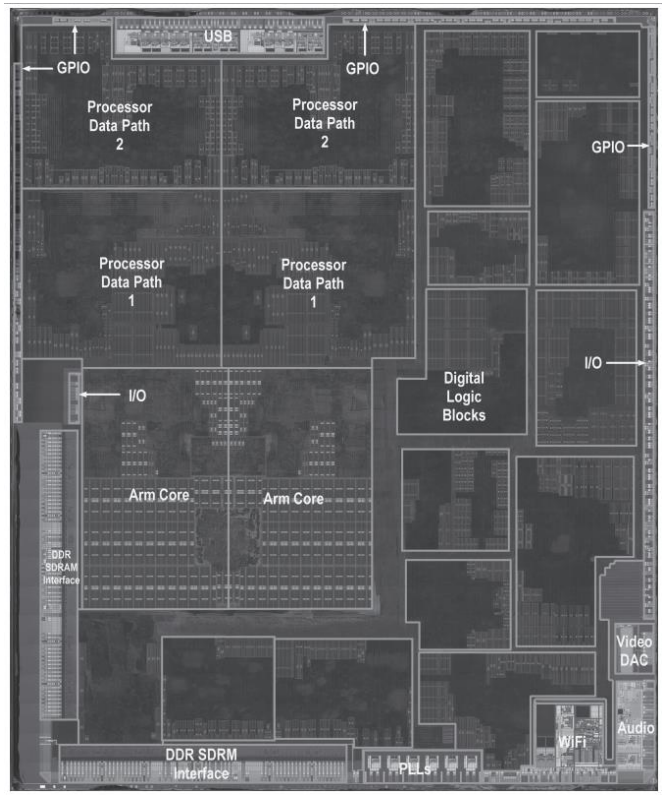
- Composed of **integrated circuits:**
  - CPU(Apple A5): Dual ARM processor, 1GHZ, 512MB DDR inside package.
  - Flash(闪存): 32GB Flash.
  - Power controller: start up(启动), shutdown(关机), standby(待机)
  - I/O controller: e.g. Control Touchscreen , control microphone




- **Composed of integrated circuits (chips：集成电路，芯片):** A device combining dozens of millions of transistors.

# CPU (Processor)

- **CPU(Processor):** active part of the computer, which contains the **datapath** and **control** and which adds numbers, tests numbers, signals I/O devices to activate, and so on.
  - **Datapath (数据通路):** performs arithmetic operation
  - **Control (控制通路):** commands the datapath, memory, and I/O devices according to the instructions of the program



# Memory: A Safe Place for Data

- **Memory(存储):** the storage area programs are kept and that contains the data needed by the running programs
- **Main Memory(主存): volatile**; used to hold programs while they are running.(e.g. DRAM in computers)
- **Second memory**: **nonvolatile**; used to store programs and data between runs. (Flash in PMD, magnetic disks)
- **Volatile (易失性)**
  - DRAM (Dynamic Random-Access Memory):动态随机存储器
  - SRAM (Static Random Access Memory)：静态随机存储器
- **Nonvolatile (非易失性)**
  - Solid state memory (Flash Memory):固态硬盘 or 闪存
  - Magnetic disk (Hard disk)：硬盘

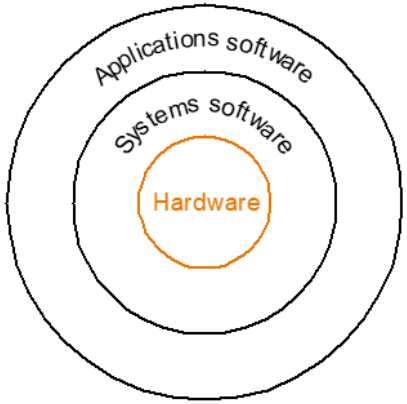
| Memory | | Price | Speed | Capacity | Used for |
|---|---|---|---|---|---|
| SRAM | volatile | / | Fastest | KB~MB | Cache |
| DRAM | volatile | $3~4/GB | Fast | MB~GB | Main memory |
| Hard disk | nonvolatile | $0.05~1/GB | Slow | TB~PB | PC Storage |
| Flash Memory | nonvolatile | $0.75~$1/GB | Medium | GB~TB | PMD Storage |

# Computer Organization

## Decomposability of computer systems

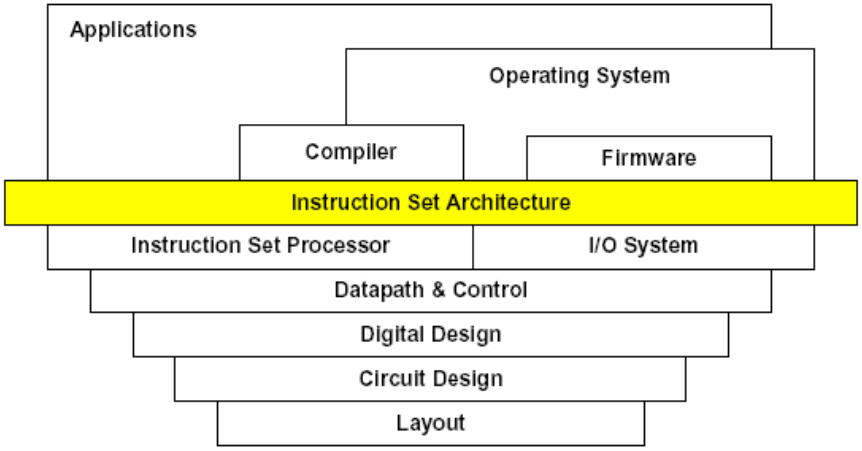
Computer System

- Software
  - Application software(应用软件): word, PPT, office
  - System software
    - Operation system: Linux, MacOS
    - Compiler(编译器): GCC
    - Firmware(Driver software): 例如：网卡驱动
- Hardware



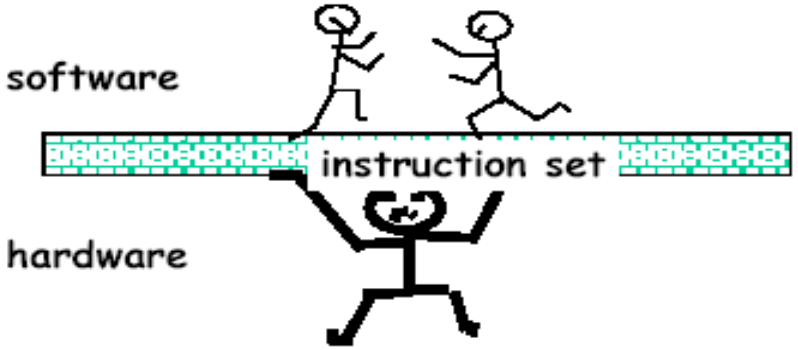
# Software Categorization

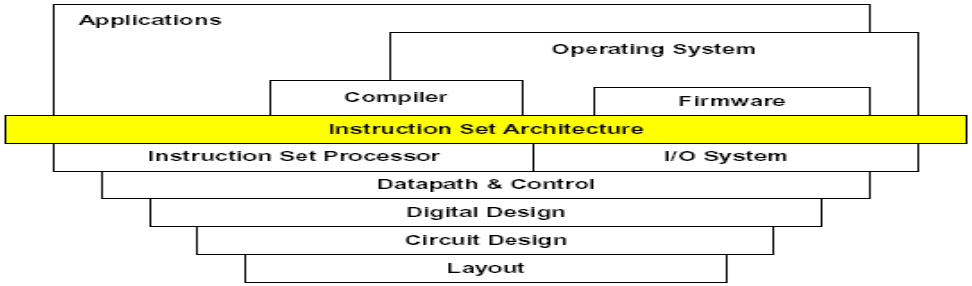
- **Categorize software by its use**
  - Systems software ---- aimed at programmers
  - Applications software ---- aimed at users
- **Operating system**
  - Handing basic input and output operations
  - Allocating storage and memory
  - Sharing the computer among multiple applications using it simultaneously
- **Compiler: Translation of a program written in HLL(高级编程语言)**
- **Firmware(驱动): Software specially designed for a piece of hardware**



# From a High-Level Language to the Language of Hardware

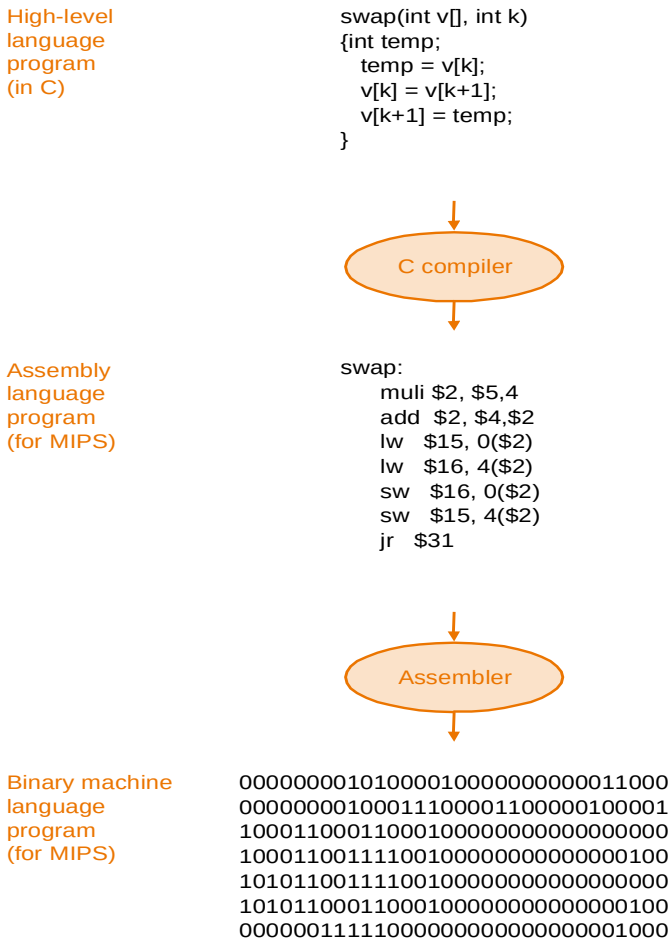
- Lower-level details are hidden to higher levels
- **Instruction set architecture（指令集）** ---- the interface between hardware and lowest-level software
- Many implementations of varying cost and performance can run identical software



# From a High-Level Language to the Language of Hardware

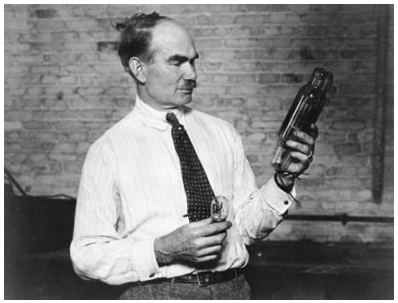
- **High-level programming language(高级编程语言)**
  - Notations more closer to the natural language
    ex. A + B
  - The compiler translates into assembly language
  - Advantages over assembly language
    - Think in a more natural language
    - Programs can be independent of hardware
- **Assembly language(汇编语言)**
  - Symbolic notations ex. add A, B
  - The assembler translates them into machine instruction
- **Machine language(机器语言)**
  - Computers only understands electrical signals on/off
  - Binary numbers express machine instructions
    ex. 1000110010100000 means to add two numbers



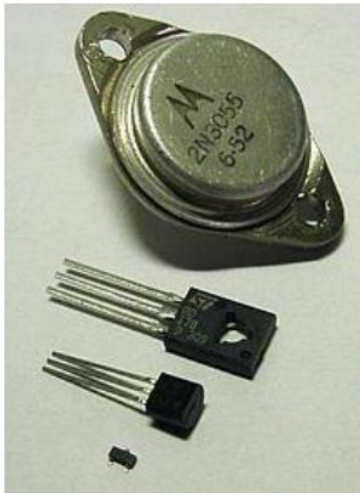
# Outline

- Introduction
- Computer organization
- **How to build processors ?**
- Computer design: performance and idea
- What you can learn from this course ?

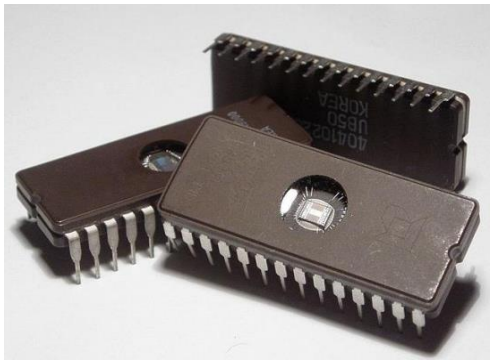
# 尺寸微缩是集成电路发展的原动力之一！



~30 cm




~3 cm



>1K devices/cm$^2$

length = few um
~= 1/3000 cm


>1B devices/cm$^2$

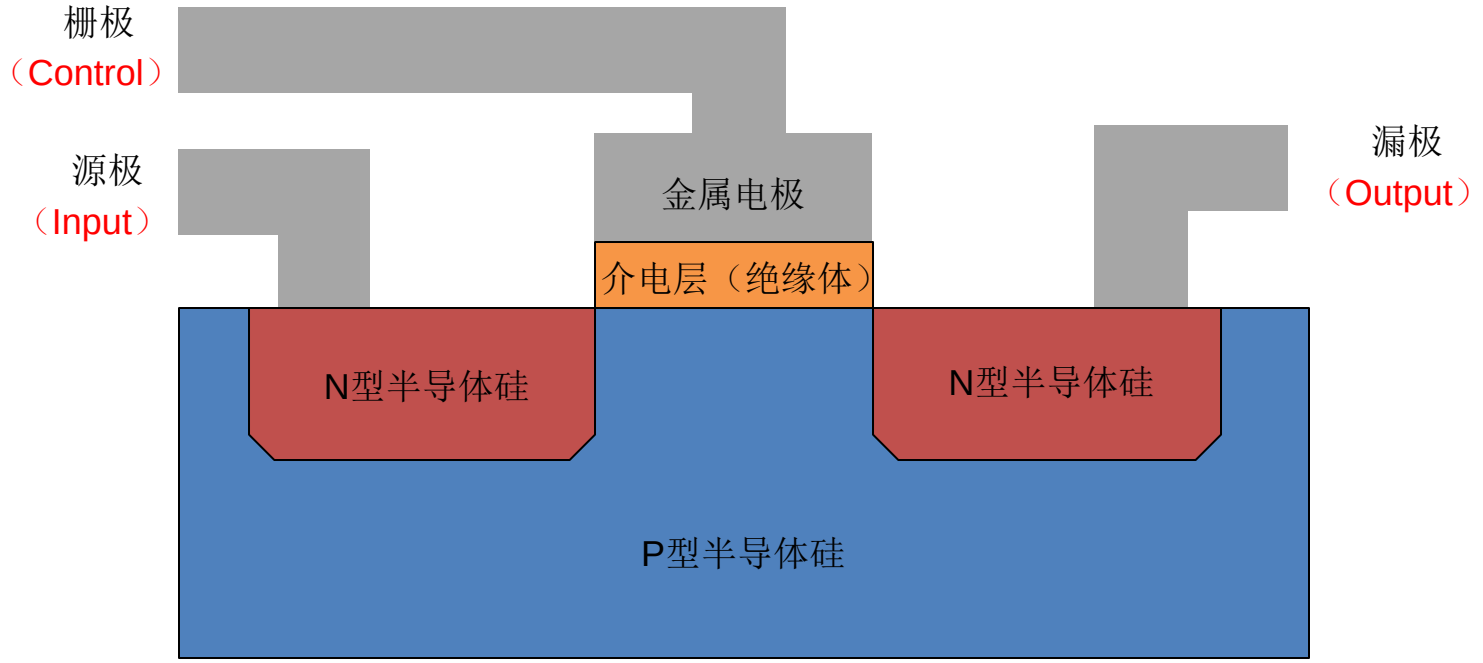
length = few nm
~= 1/3000000 cm

Nanotechnology！

头发丝直径 = 500 um = 1/20 cm

# Transistor 晶体管

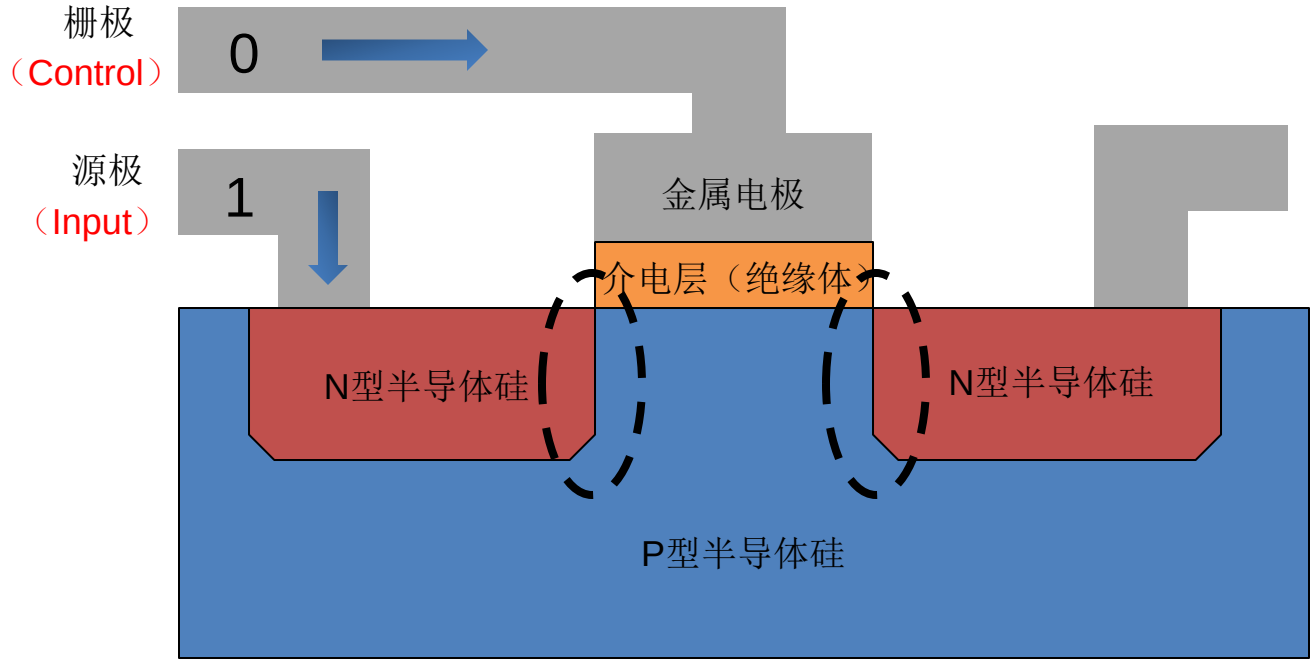
- 半导体：性质介于金属与绝缘体之间，通过加电场可以改变其性质
- P型半导体（电子密度比正常水平小），N型半导体（电子密度大）



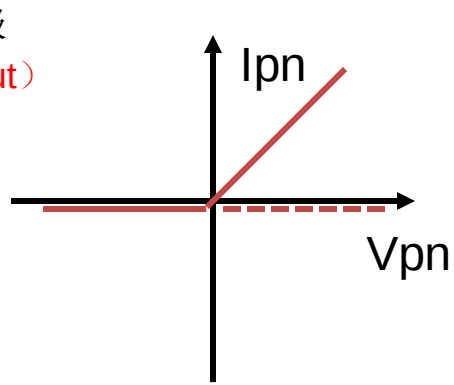

Back-to-back PN, no current! Switch off!

# Transistor 晶体管

- 半导体：性质介于金属与绝缘体之间，通过加电场可以改变其性质
- P型半导体（电子密度比正常水平小），N型半导体（电子密度大）




Back-to-back PN, no current! Switch off!

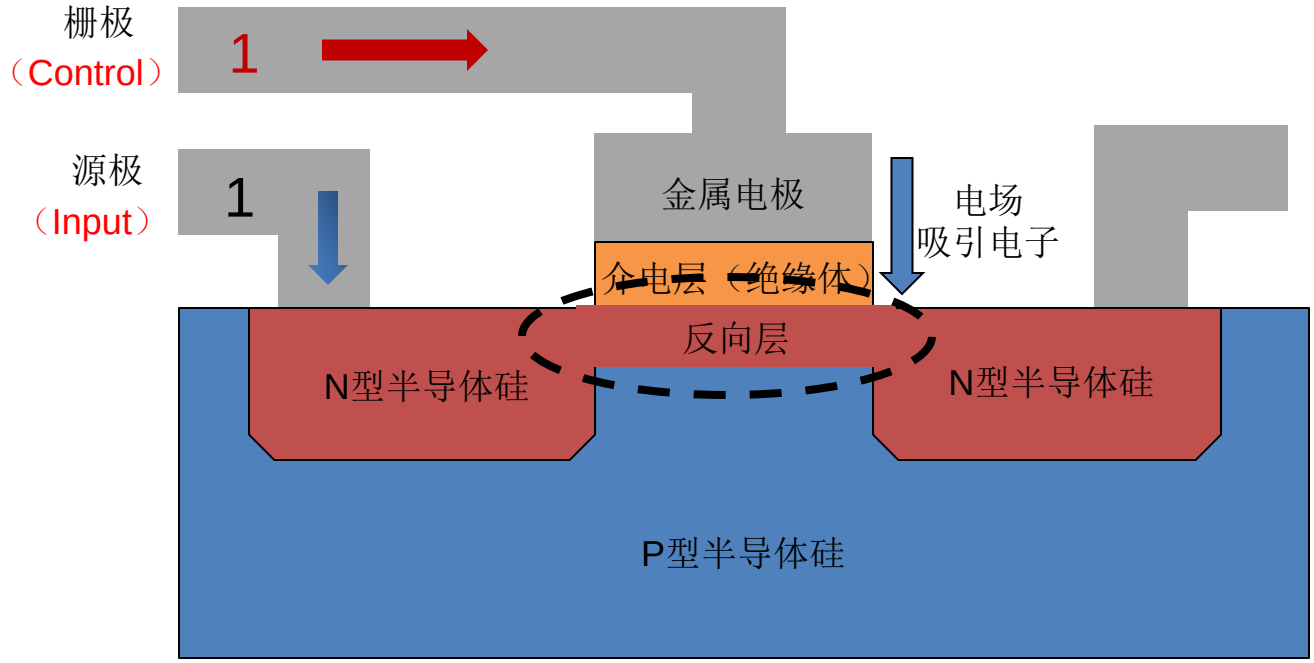


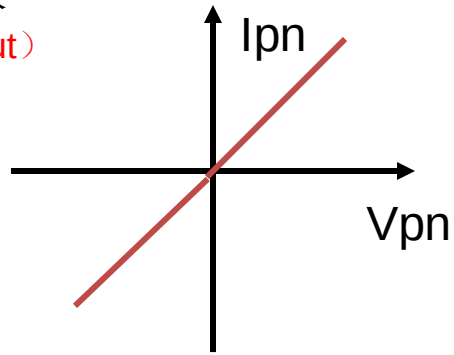

PN结

电流只能从P流向N

# Transistor 晶体管

- 半导体：性质介于金属与绝缘体之间，通过加电场可以改变其性质
- P型半导体（电子密度比正常水平小），N型半导体（电子密度大）



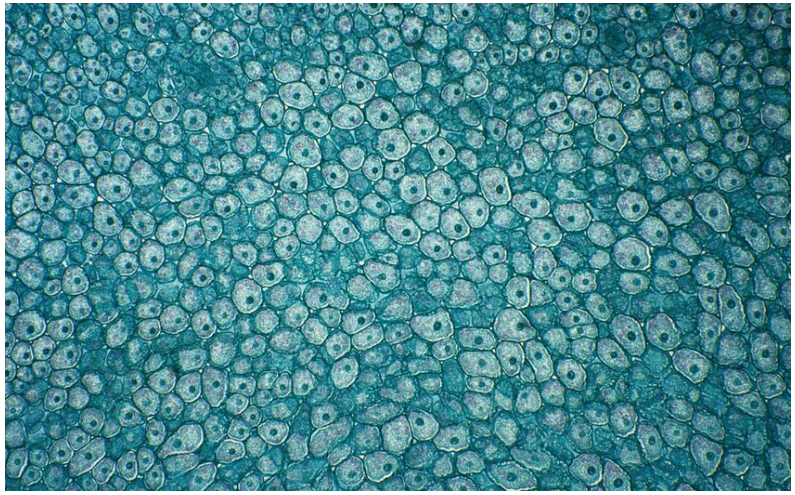

消除PN结

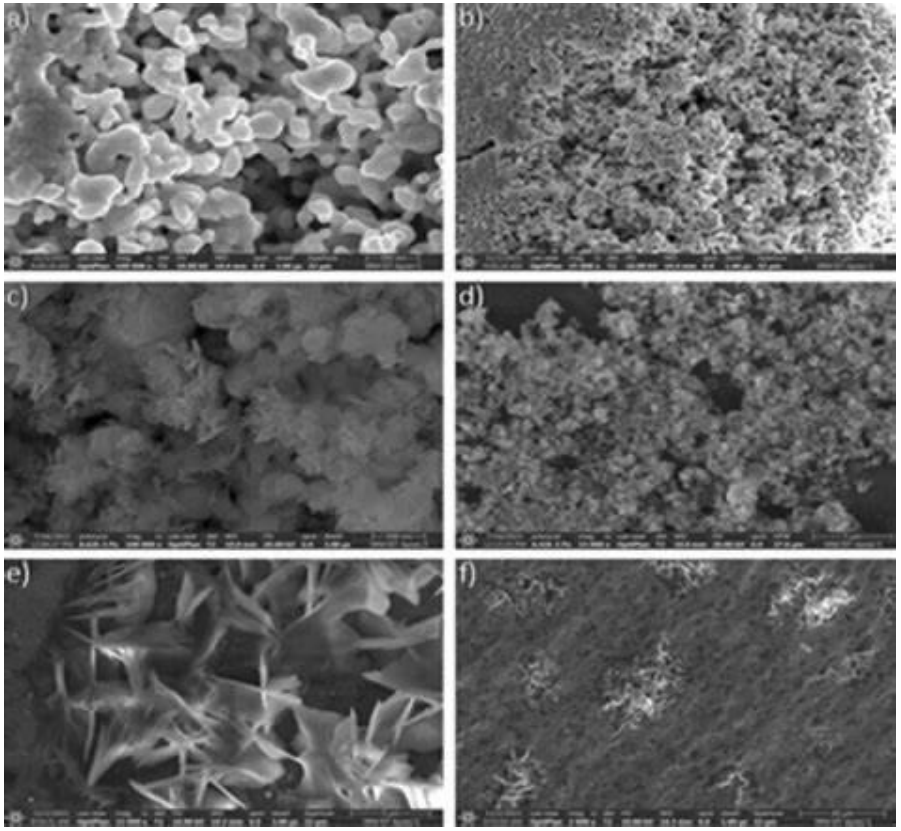
晶体管导通

Locally at interface, P-type becomes N-type, switch ON!

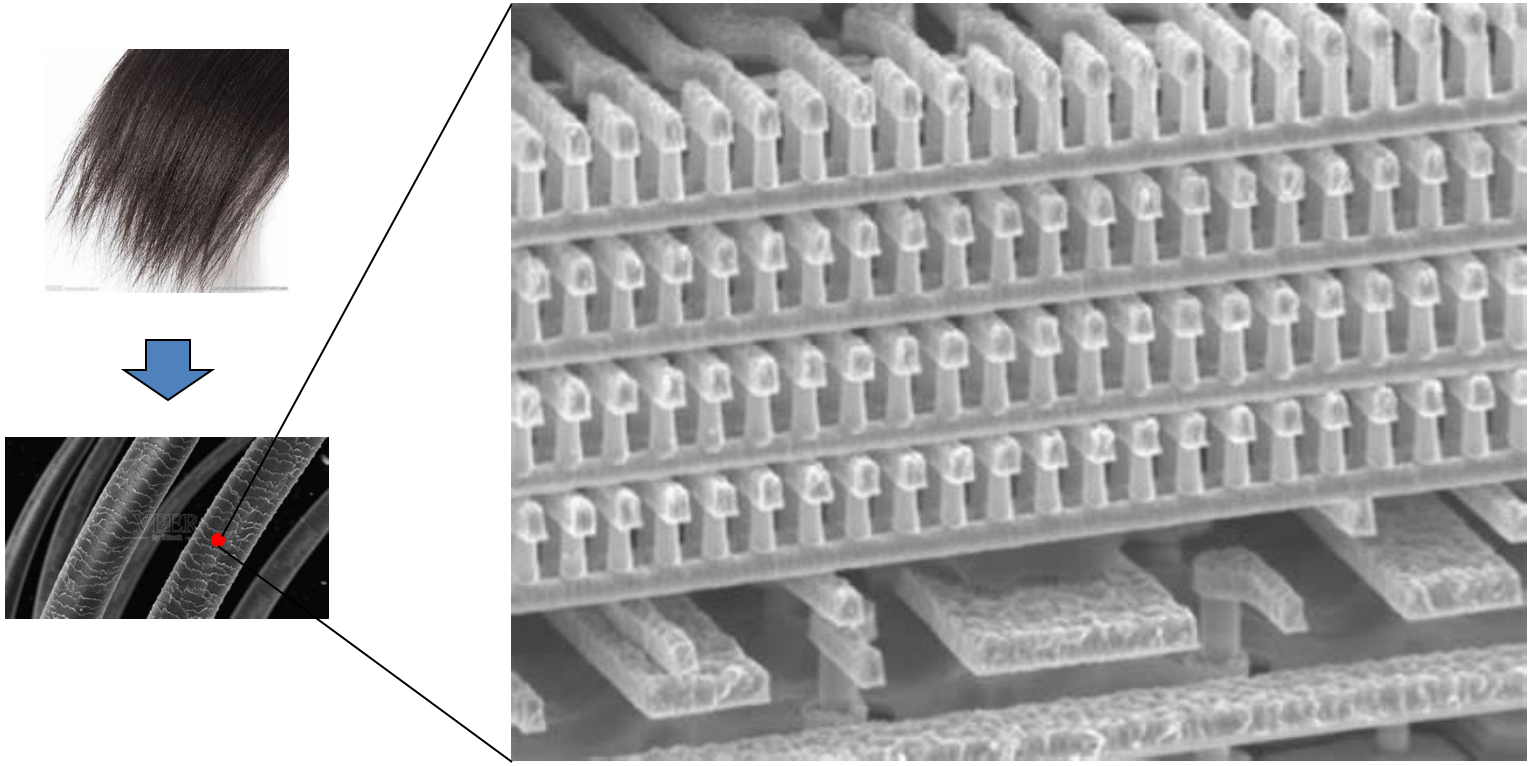
# How to make a chip?

How does transistor look like in nanoscale?



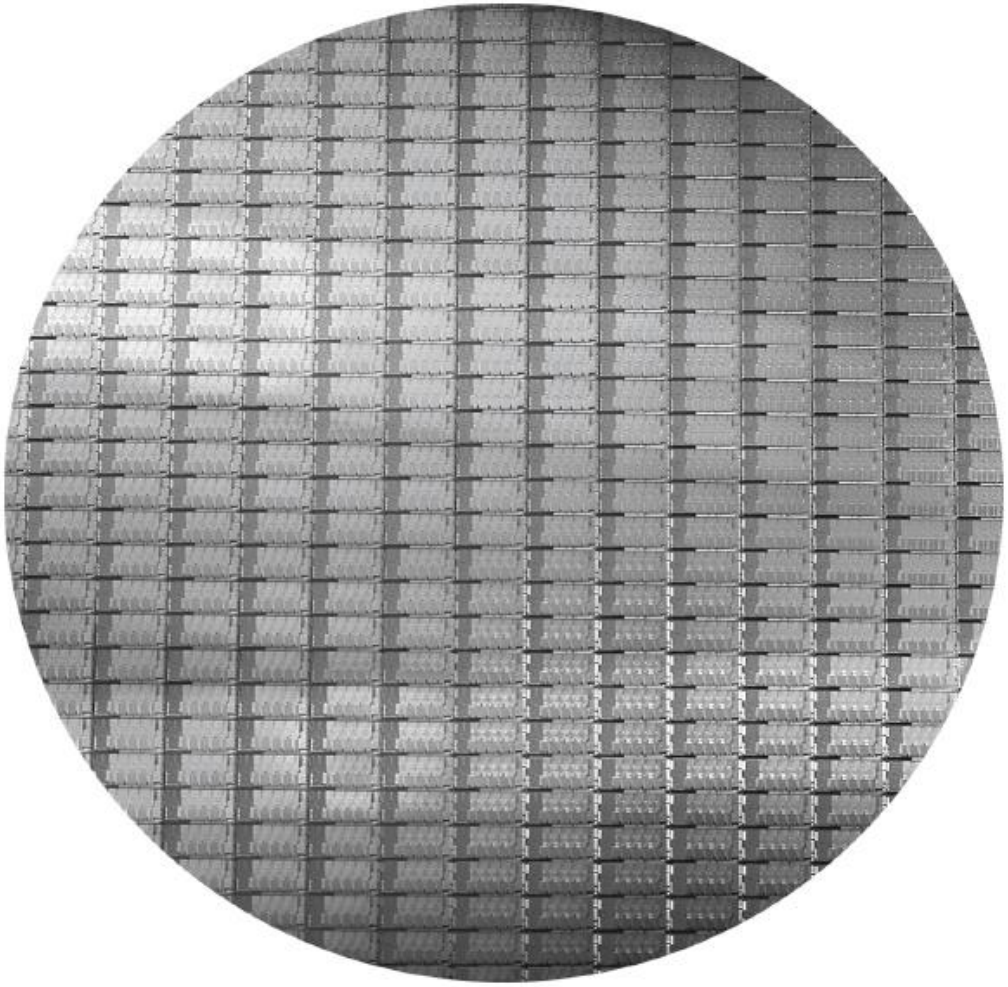


# Skyscraper in nanoscale



# Intel Core i7 Wafer

- 300mm wafer,
  280 chips,
  32nm technology
- Each chip is:
  20.7 × 10.5 mm



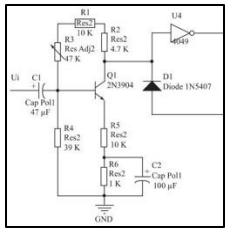
# The Chip Manufacturing Process

芯片设计
Chip design

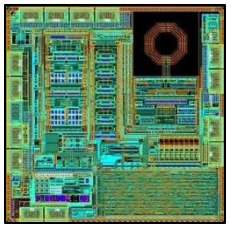
晶圆生产
Wafer production

芯片封装
Chip package

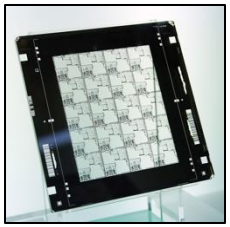
芯片测试
Chip test



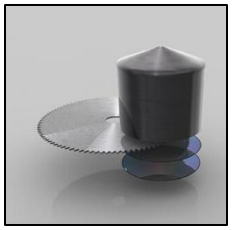
Circuit design(电路设计)



Layout design(版图设计)



Mask(掩膜)



Ingot cutting(铸锭切割)

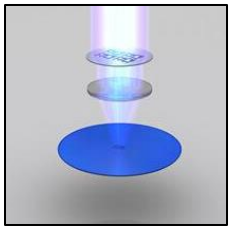
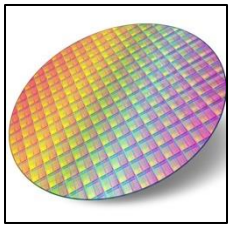
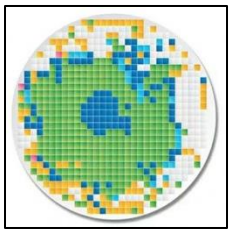


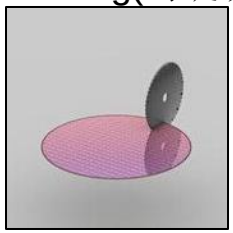
Photo resist(光刻)



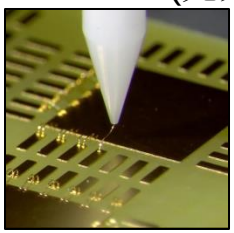
Wafer(晶圆)



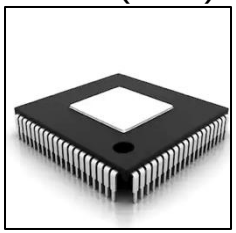
Wafer test(晶圆测试)



Slicing(晶圆切片)



Wire Bond(焊线)



Package(封装)

Final test(成品测试)

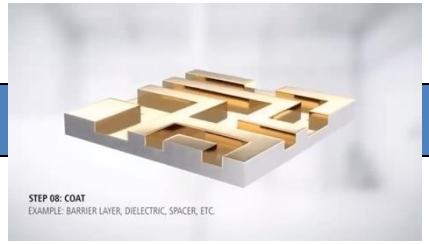
Finished chip(芯片成品)

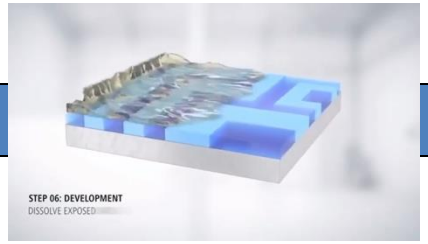
# The fabrication process of a chip

沙子/sand/silicon

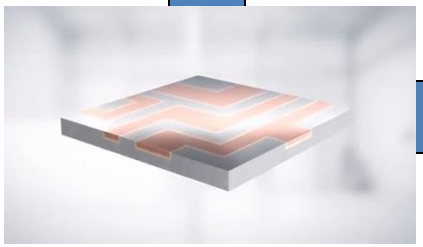


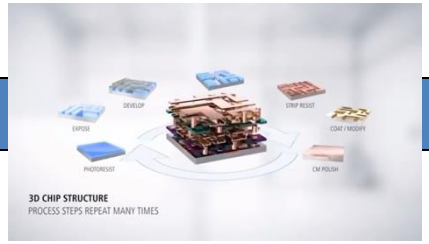


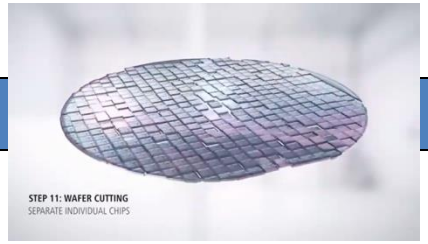


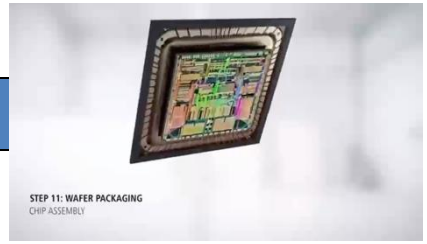









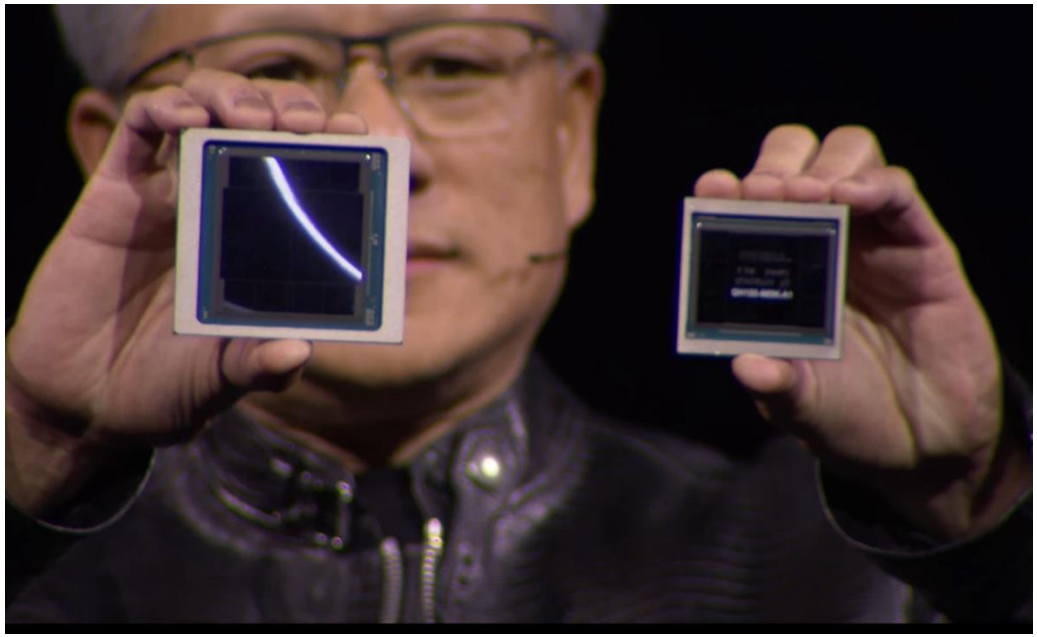



# Intro video on how a chip is made

# Intro video on how a chip is made

# Hardware advancement push computing forward

## Nvidia Blackwell架构

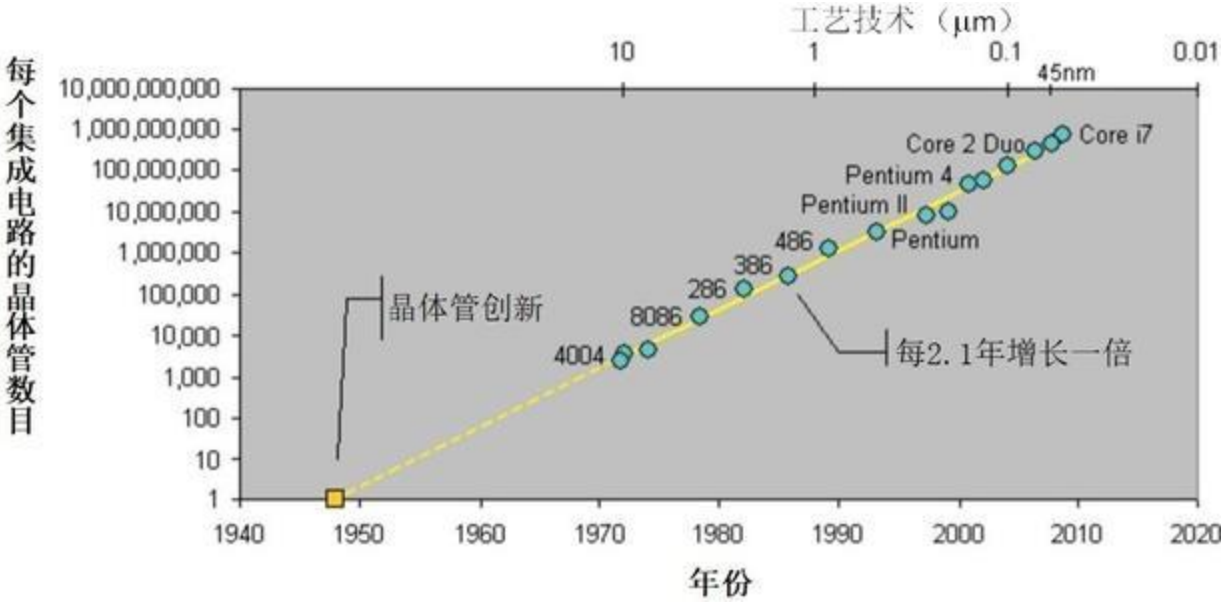


- 台积电 4nm工艺
- 超200亿个晶体管
- 支持192GB HBM3e

# Development of integrated circuits

Relative performance per unit cost of technologies used in computers over time

| Year | Technology used in computers | Relative performance/unit cost |
|---|---|---|
| 1951 | Vacuum tube | 1 |
| 1965 | Transistor | 35 |
| 1975 | Integrated circuit | 900 |
| 1995 | Very large-scale integrated circuit | 2,400,000 |
| 2013 | Ultra large-scale integrated circuit | 250,000,000,000 |



# Challenges of Integrated Circuits Manufacturing

- The integration of SoC increases rapidly(1billions transistor on single chip), and complexity augments increasingly.
  (集成度越来越高，复杂度快速增加)

- The size of the IC process reaches physical limitation(5nm now). More and more precision requirements and more and more difficult to manufacture.
  (工艺尺寸达到物理极限，加工精度越来越高， 制造难度越来越大)

- Involved many technologies: Electronics, optics, Mechanics, computer science… Each technology is hard to break through.
  (多个技术的集成：电子，光学，机械，计算机，每一个技术都能难突破)

# Integrated Circuit Cost

- **Yield: proportion of working dies per wafer**

$$\text{Cost per die} = \frac{\text{Cost per wafer}}{\text{Dies per wafer} \times \text{Yield}}$$

$$\text{Dies per wafer} \approx \text{Wafer area} / \text{Die area}$$

$$\text{Yield} = \frac{1}{(1 + (\text{Defects per area} \times \text{Die area}/2))^2}$$

- **Nonlinear relation to area and defect rate**
  - Wafer cost and area are fixed
  - Defect rate determined by manufacturing process
  - Die area determined by architecture and circuit design

# Outline

- Introduction
- Computer organization
- How to build processors ?
- **Computer design: performance and idea**
- What you can learn from this course ?

# Response Time and Throughput

- **Response time/execution time**

  响应时间/执行时间

  - How long it takes to do a task
- **Throughput (bandwidth)**

  吞吐率

  - Total work done per unit time
    - e.g., tasks/transactions/… per hour
- **How are response time and throughput affected by**
  - Replacing the processor with a faster version?
  - Adding more processors?
- **We'll focus on response time for now…**

# Relative Performance

- Define Performance = 1/Execution Time
- "X is *n* time faster than Y"

$$\text{Performance}_X / \text{Performance}_Y = \text{Execution time}_Y / \text{Execution time}_X = n$$

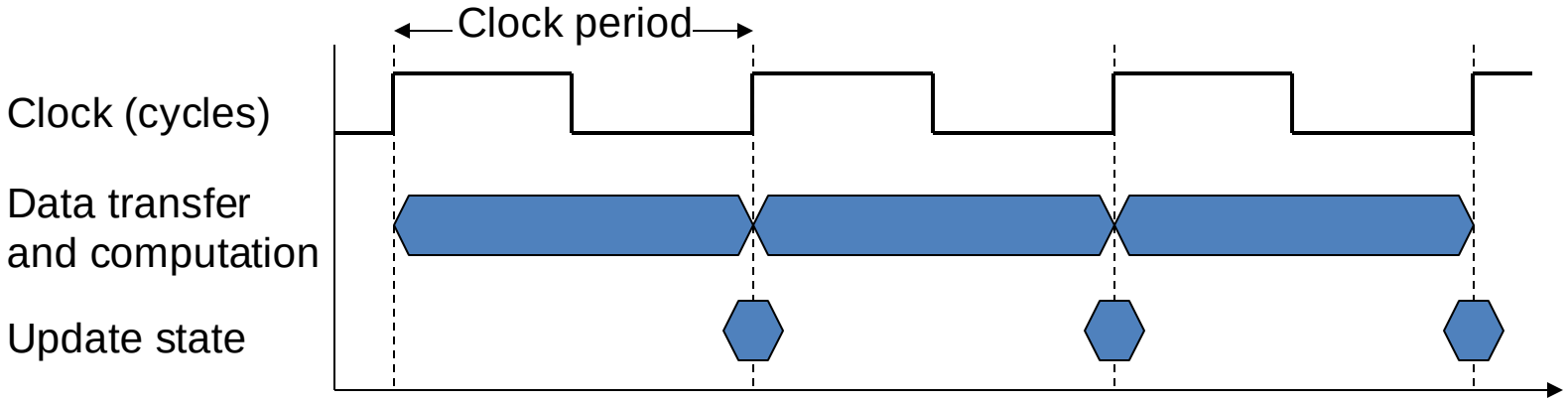
- Example: time taken to run a program
  - 10s on A, 15s on B
  - $\text{Execution Time}_B$ / $\text{Execution Time}_A$ = 15s / 10s = 1.5
  - So A is 1.5 times faster than B

# Measuring Execution Time

- **Elapsed time**
  - Total response time, including all aspects
    - Processing, I/O, OS overhead, idle time
  - Determines system performance
- **CPU time**
  - Time spent processing a given job
    - Discounts I/O time, other jobs' shares
  - Comprises user CPU time and system CPU time
  - Different programs are affected differently by CPU and system performance

# Measuring Execution Time

- Operation of digital hardware governed by a constant-rate clock




- Clock period: duration of a clock cycle
  - e.g., 250ps = 0.25ns = $250\times10^{-12}$s
- Clock frequency (rate): cycles per second
  - e.g., 4.0GHz = 4000MHz = $4.0\times10^{9}$Hz

# CPU Time

$$\text{CPU Time} = \text{CPU Clock Cycles} \times \text{Clock Cycle Time}$$

$$= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}$$

- **Performance improved by**
  - Reducing number of clock cycles
  - Increasing clock rate
  - Hardware designer must often trade off clock rate against cycle count

# CPU Time Example

- **Computer A: 2GHz clock, 10s CPU time**
- **Designing Computer B**
  - Aim for 6s CPU time
  - Can do faster clock, but causes 1.2 × clock cycles
- **How fast must Computer B clock be?**

$$\text{Clock Rate}_B = \frac{\text{Clock Cycles}_B}{\text{CPU Time}_B} = \frac{1.2 \times \text{Clock Cycles}_A}{6\text{s}}$$

$$\text{Clock Cycles}_A = \text{CPU Time}_A \times \text{Clock Rate}_A$$

$$= 10\text{s} \times 2\text{GHz} = 20 \times 10^9$$

$$\text{Clock Rate}_B = \frac{1.2 \times 20 \times 10^9}{6\text{s}} = \frac{24 \times 10^9}{6\text{s}} = 4\text{GHz}$$

# Instruction Count and CPI

$$\text{Clock Cycles} = \text{Instruction Count} \times \text{Cycles per Instruction}$$

$$\text{CPU Time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time}$$

$$= \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}$$

- **Instruction Count for a program**
  - Determined by program, ISA and compiler
- **Average cycles per instruction**
  - Determined by CPU hardware
  - If different instructions have different CPI
    - Average CPI affected by instruction mix

# CPI Example

- Computer A: Cycle Time = 250ps, CPI = 2.0
- Computer B: Cycle Time = 500ps, CPI = 1.2
- Same ISA
- Which is faster, and by how much?

$$\text{CPU Time}_A = \text{Instruction Count} \times \text{CPI}_A \times \text{Cycle Time}_A$$

$$= I \times 2.0 \times 250\text{ps} = I \times 500\text{ps}$$

A is faster…

$$\text{CPU Time}_B = \text{Instruction Count} \times \text{CPI}_B \times \text{Cycle Time}_B$$

$$= I \times 1.2 \times 500\text{ps} = I \times 600\text{ps}$$

$$\frac{\text{CPU Time}_B}{\text{CPU Time}_A} = \frac{I \times 600\text{ps}}{I \times 500\text{ps}} = 1.2$$

…by this much

# CPI in More Detail

- If different instruction classes take different numbers of cycles

$$\text{Clock Cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{Instruction Count}_i)$$

- Weighted average CPI

$$\text{CPI} = \frac{\text{Clock Cycles}}{\text{Instruction Count}} = \sum_{i=1}^{n} \left( \text{CPI}_i \times \frac{\text{Instruction Count}_i}{\text{Instruction Count}} \right)$$

Relative frequency

# CPI Example

- **Alternative compiled code sequences using instructions in classes A, B, C**

| Class | A | B | C |
|---|---|---|---|
| CPI for class | 1 | 2 | 3 |
| IC in sequence 1 | 2 | 1 | 2 |
| IC in sequence 2 | 4 | 1 | 1 |

- **Sequence 1: IC = 5**
  - Clock Cycles
    = 2×1 + 1×2 + 2×3
    = 10
  - Avg. CPI = 10/5 = 2.0
- **Sequence 2: IC = 6**
  - Clock Cycles
    = 4×1 + 1×2 + 1×3
    = 9
  - Avg. CPI = 9/6 = 1.5

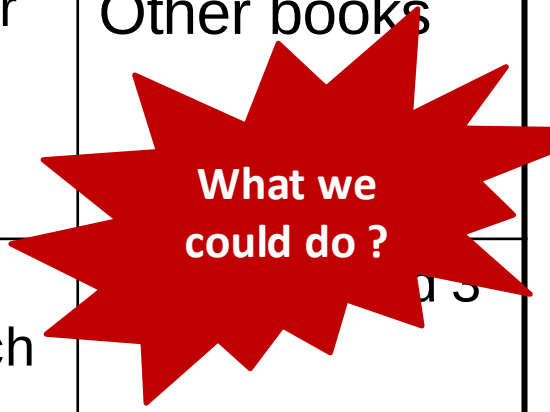
# Performance Summary

$$\text{CPU Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

- **Performance depends on**
  - Algorithm: affects IC, possibly CPI
  - Programming language: affects IC, CPI
  - Compiler: affects IC, CPI
  - Instruction set architecture: affects IC, CPI, $T_c$

# Where is the Performance Bottleneck?

| Hardware or software component | How this component affects performance | Where is this topic covered? |
|---|---|---|
| Algorithm | Determines both the number of source-level statements and the number of I/O operations executed | Other books |
| Programming language, compiler, and architecture | Determines the number of machine instructions for each source-level statements | |
| Processor and memory system | Determines how fast instructions can be executed | Chapter 4，5 |
| I/O system(hardware and operating system) | Determines how fast I/O operations may be executed | Other books |



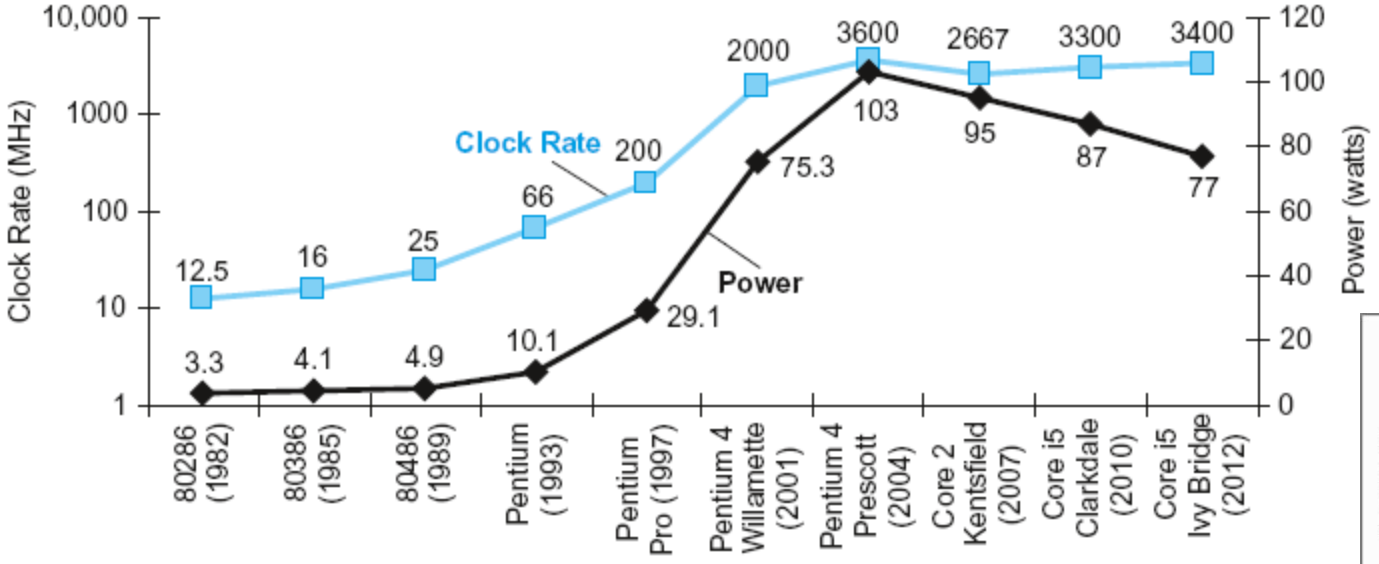
# CPI in More Details

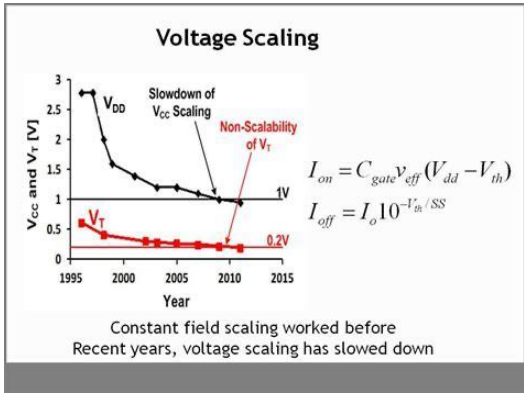
- **Suppose a new CPU has**
  - 85% of capacitive load of old CPU
  - 15% voltage and 15% frequency reduction

$$\frac{P_{new}}{P_{old}} = \frac{C_{old} \times 0.85 \times (V_{old} \times 0.85)^2 \times F_{old} \times 0.85}{C_{old} \times V_{old}^{\ 2} \times F_{old}} = 0.85^4 = 0.52$$

- **The power wall**
  - We can't reduce voltage further
  - We can't remove more heat

# Power Trends

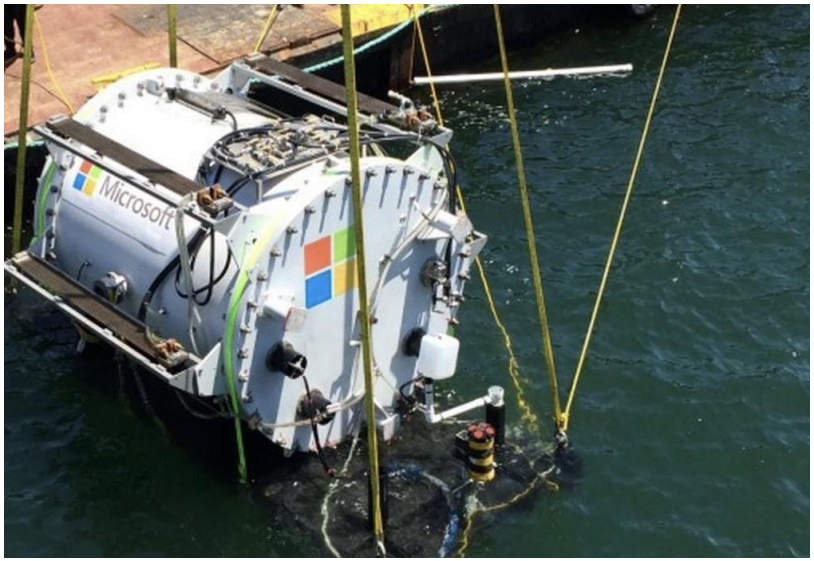


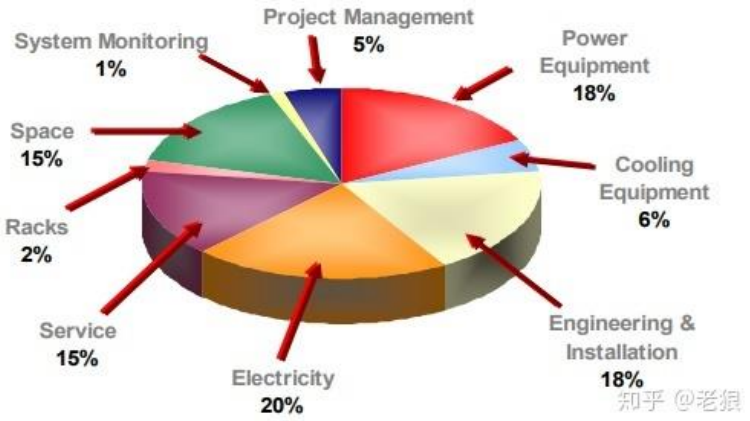

- **In CMOS IC technology**

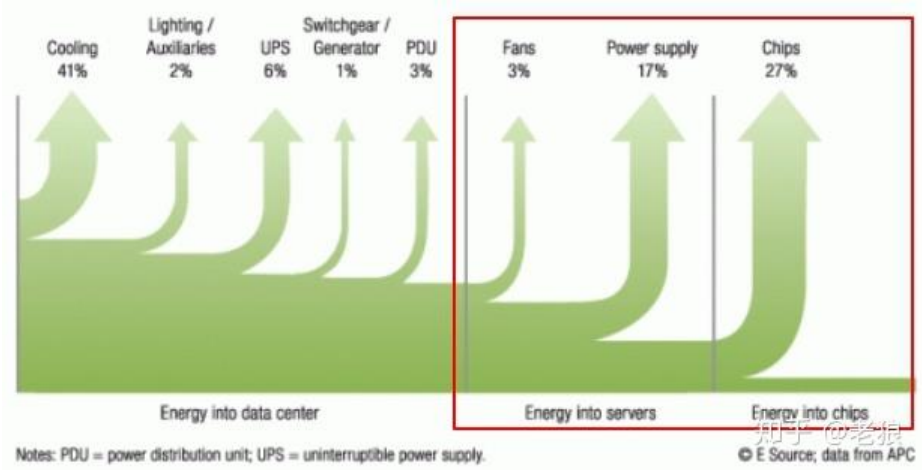
$$\text{Power} = \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency}$$

×30 | 5V → 1V | ×1000

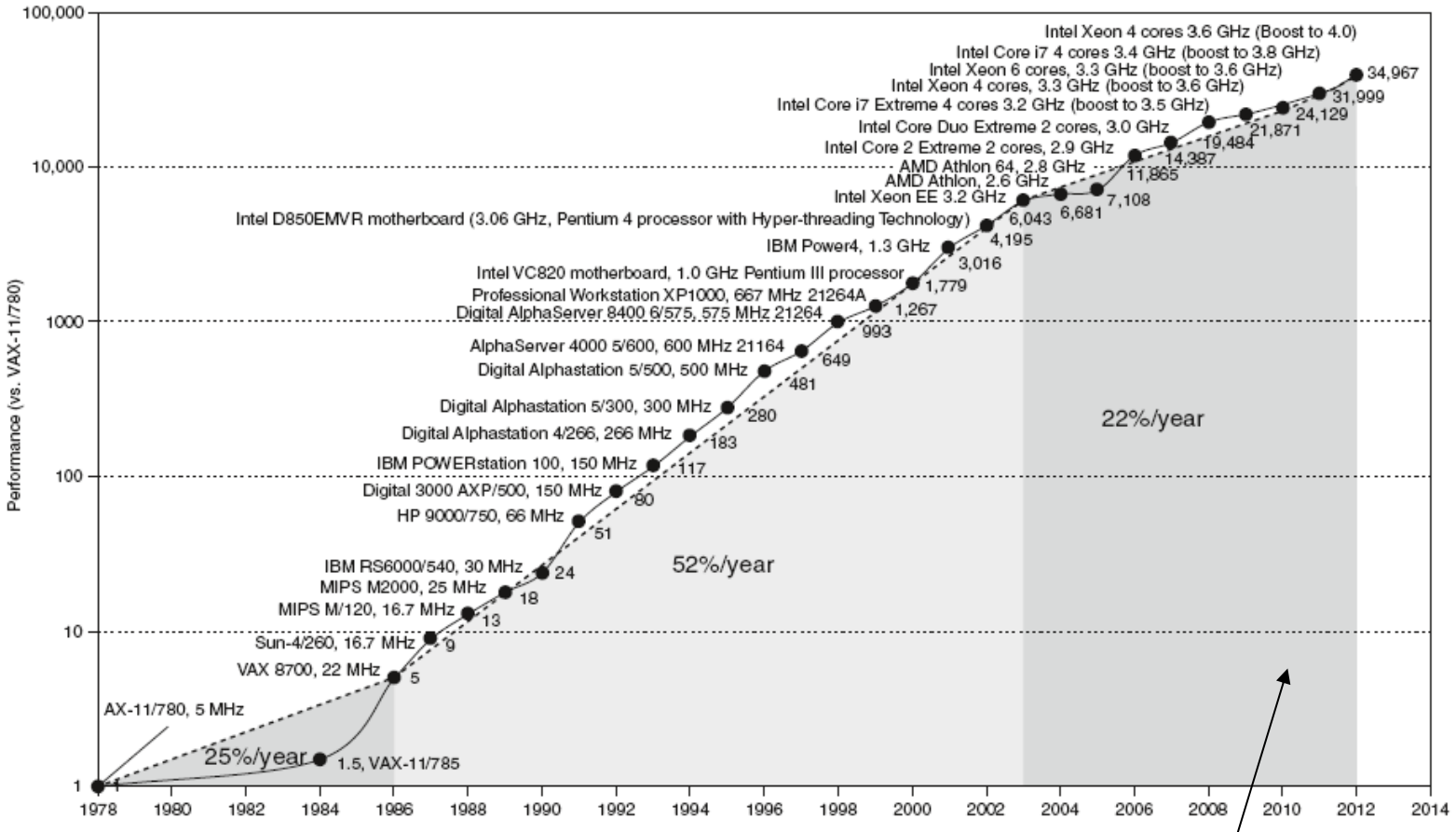

Microsoft


知乎 @老狼


System Monitoring
1%
Project Management
5%
Power
Equipment
18%
Space
15%
Cooling
Equipment
6%
Racks
2%
Service
15%
Electricity
20%
Engineering &
Installation
18%
知乎 @老狼


Cooling
41%
Lighting /
Auxiliaries
2%
UPS
6%
Switchgear /
Generator
1%
PDU
3%
Fans
3%
Power supply
17%
Chips
27%
Energy into data center
Energy into servers
Energy into chips
Notes: PDU = power distribution unit; UPS = uninterruptible power supply.
© E Source; data from APC
知乎 @老狼

# Single Core Performance




Constrained by power, instruction-level parallelism, memory latency

# Multiprocessors

- **Multicore microprocessors**
  - More than one processor per chip
- **Requires explicitly parallel programming**
  - Compare with instruction level parallelism
    - Hardware executes multiple instructions at once
    - Hidden from the programmer
  - Hard to do
    - Programming for performance
    - Load balancing
    - Optimizing communication and synchronization

# SPEC CPU Benchmark

- **Programs used to measure performance**
  - Supposedly typical of actual workload
- **Standard Performance Evaluation Corp (SPEC)**
  - Develops benchmarks for CPU, I/O, Web, …
- **SPEC CPU2006**
  - Elapsed time to execute a selection of programs
    - Negligible I/O, so focuses on CPU performance
  - Normalize relative to reference machine
  - Summarize as geometric mean of performance ratios
    - CINT2006 (integer) and CFP2006 (floating-point)

$$\sqrt[n]{\prod_{i=1}^{n} \text{Execution time ratio}_i}$$

# CINT2006 for Intel Core i7 920

| Description | Name | Instruction Count x $10^9$ | CPI | Clock cycle time (seconds x $10^{-9}$) | Execution Time (seconds) | Reference Time (seconds) | SPECratio |
|---|---|---|---|---|---|---|---|
| Interpreted string processing | perl | 2252 | 0.60 | 0.376 | 508 | 9770 | 19.2 |
| Block-sorting compression | bzip2 | 2390 | 0.70 | 0.376 | 629 | 9650 | 15.4 |
| GNU C compiler | gcc | 794 | 1.20 | 0.376 | 358 | 8050 | 22.5 |
| Combinatorial optimization | mcf | 221 | 2.66 | 0.376 | 221 | 9120 | 41.2 |
| Go game (AI) | go | 1274 | 1.10 | 0.376 | 527 | 10490 | 19.9 |
| Search gene sequence | hmmer | 2616 | 0.60 | 0.376 | 590 | 9330 | 15.8 |
| Chess game (AI) | sjeng | 1948 | 0.80 | 0.376 | 586 | 12100 | 20.7 |
| Quantum computer simulation | libquantum | 659 | 0.44 | 0.376 | 109 | 20720 | 190.0 |
| Video compression | h264avc | 3793 | 0.50 | 0.376 | 713 | 22130 | 31.0 |
| Discrete event simulation library | omnetpp | 367 | 2.10 | 0.376 | 290 | 6250 | 21.5 |
| Games/path finding | astar | 1250 | 1.00 | 0.376 | 470 | 7020 | 14.9 |
| XML parsing | xalancbmk | 1045 | 0.70 | 0.376 | 275 | 6900 | 25.1 |
| Geometric mean | – | – | – | – | – | – | 25.7 |

# SPEC Power Benchmark

- **Power consumption of server at different workload levels**
  - Performance: ssj_ops/sec
  - Power: Watts (Joules/sec)

$$\text{Overall ssj\_ops per Watt} = \left(\sum_{i=0}^{10} \text{ssj\_ops}_i\right) \Big/ \left(\sum_{i=0}^{10} \text{power}_i\right)$$

# SPECpower_ssj2008 for Xeon X5650

| Target Load % | Performance (ssj_ops) | Average Power (Watts) |
|---|---|---|
| 100% | 865,618 | 258 |
| 90% | 786,688 | 242 |
| 80% | 698,051 | 224 |
| 70% | 607,826 | 204 |
| 60% | 521,391 | 185 |
| 50% | 436,757 | 170 |
| 40% | 345,919 | 157 |
| 30% | 262,071 | 146 |
| 20% | 176,061 | 135 |
| 10% | 86,784 | 121 |
| 0% | 0 | 80 |
| Overall Sum | 4,787,166 | 1,922 |
| Σssj_ops/Σpower = | | 2,490 |

# Pitfall: Amdahl's Law

- **Improving an aspect of a computer and expecting a proportional improvement in overall performance**

$$T_{improved} = \frac{T_{affected}}{improvement\ factor} + T_{unaffected}$$

- **Example: multiply accounts for 80s/100s**
  - How much improvement in multiply performance to get 5× overall?

$$20 = \frac{80}{n} + 20$$

  - Can't be done!
- **Corollary: make the common case fast**

# Fallacy: Low Power at Idle

- **Look back at i7 power benchmark**
  - At 100% load: 258W
  - At 50% load: 170W (66%)
  - At 10% load: 121W (47%)
- **Google data center**
  - Mostly operates at 10% – 50% load
  - At 100% load less than 1% of the time
- **Consider designing processors to make power proportional to load**

# Pitfall: MIPS as a Performance Metric

- **MIPS: Millions of Instructions Per Second**
  - Doesn't account for
    - Differences in ISAs between computers
    - Differences in complexity between instructions

$$\text{MIPS} = \frac{\text{Instruction count}}{\text{Execution time} \times 10^6}$$

$$= \frac{\text{Instruction count}}{\frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}} \times 10^6} = \frac{\text{Clock rate}}{\text{CPI} \times 10^6}$$

  - CPI varies between programs on a given CPU

# Concluding Remarks

- **Cost/performance is improving**
  - Due to underlying technology development
- **Hierarchical layers of abstraction**
  - In both hardware and software
- **Instruction set architecture**
  - The hardware/software interface
- **Execution time: the best performance measure**
- **Power is a limiting factor**
  - Use parallelism to improve performance

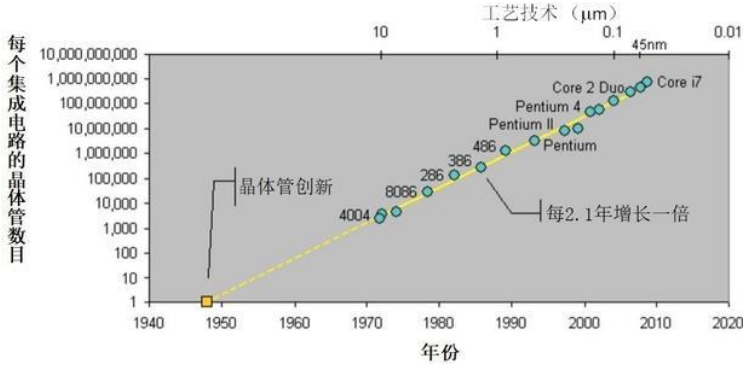
# Eight Great Ideas

- Design for Moore's Law （设计紧跟摩尔定律）
- Use Abstraction to Simplify Design (采用抽象简化设计)
- Make the Common Case Fast (加速大概率事件)
- Performance via Parallelism (通过并行提高性能)
- Performance via Pipelining (通过流水线提高性能)
- Performance via Prediction (通过预测提高性能)
- Hierarchy of Memories (存储器层次)
- Dependability via Redundancy (通过冗余提高可靠性)

# Idea1： Design for Moore's Law

- One **constant** for computer designer is **rapid change**.
- Design for **where it will be** when finishes rather than design **for where it starts**.

**Gordon Moore**, a cofounder of Intel, predicts that the number of transistors which can be placed on a single chip will double every 2 year. It has held steady and was dubbed "**Moore's Law**" around 1970

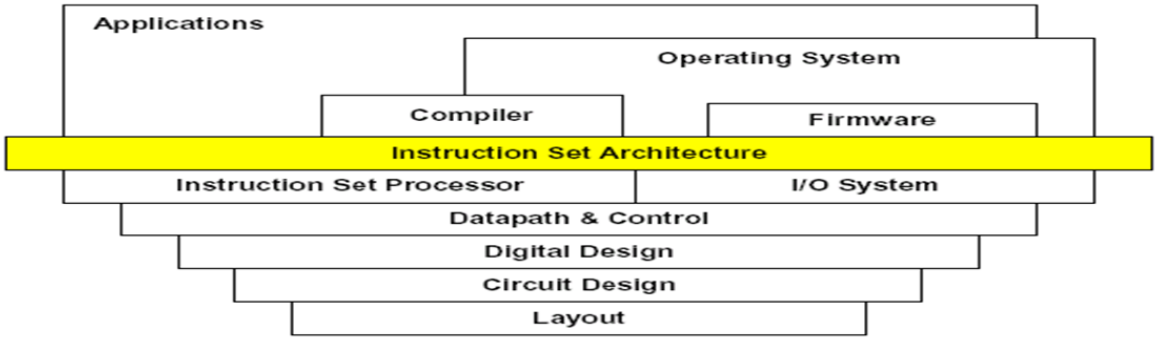


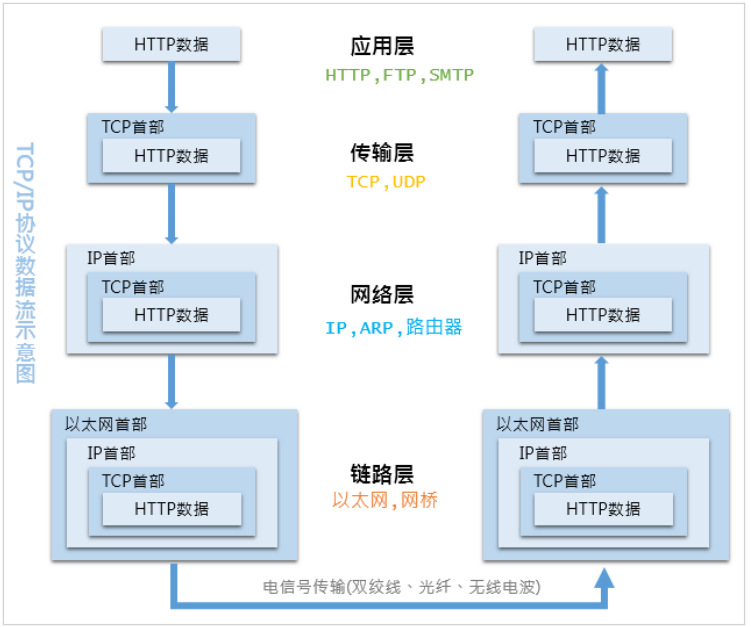

*Why Moore's law?*

- Smaller Transistor, **more function** on same chip area;
- Smaller Transistor, **low power**
- Smaller Transistor, smaller area for same function, **low cost**;

# Idea2:Use Abstraction to Simplify Design

- Lower-level details are hidden to offer a simple model at higher level



# Idea3：Make the Common Case Fast

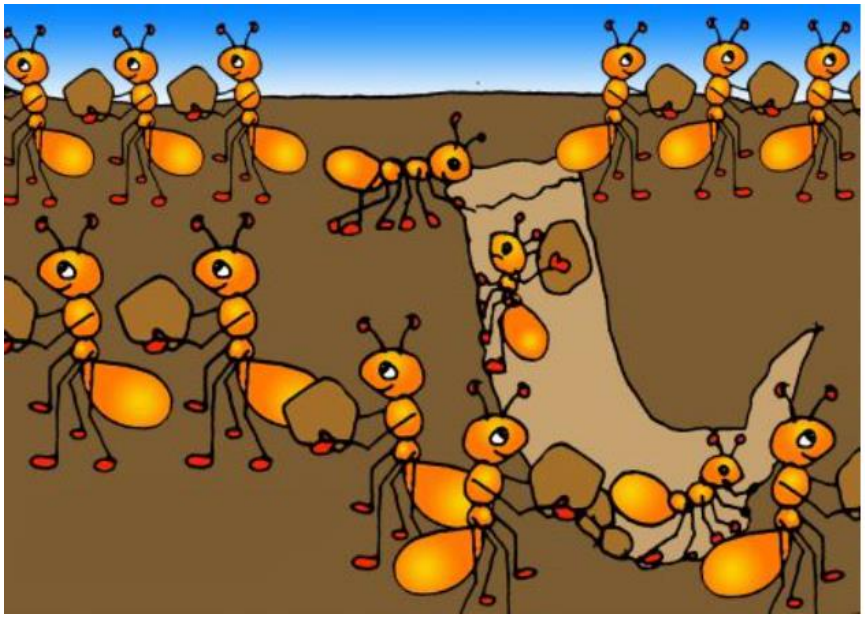
- **Making the common case fast will tend to enhance performance better than optimizing the rare case**

```
for(i=0;i<2;i++){
   洗衣半小时
}
for(i=0;i<1000;i++){
   上班走路1小时
}
for(i=0;i<2;i++){
   做饭半小时
}
```

I think I should buy a car first.

# Idea4:Performance via Parallelism

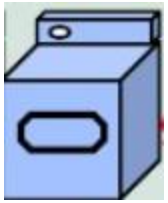
- Get more performance by performing operation in parallel



# Idea5: Performance via Pipelining

- A, B, C, D all have some clothes to wash, dryer and fold

Wash: 30minuts



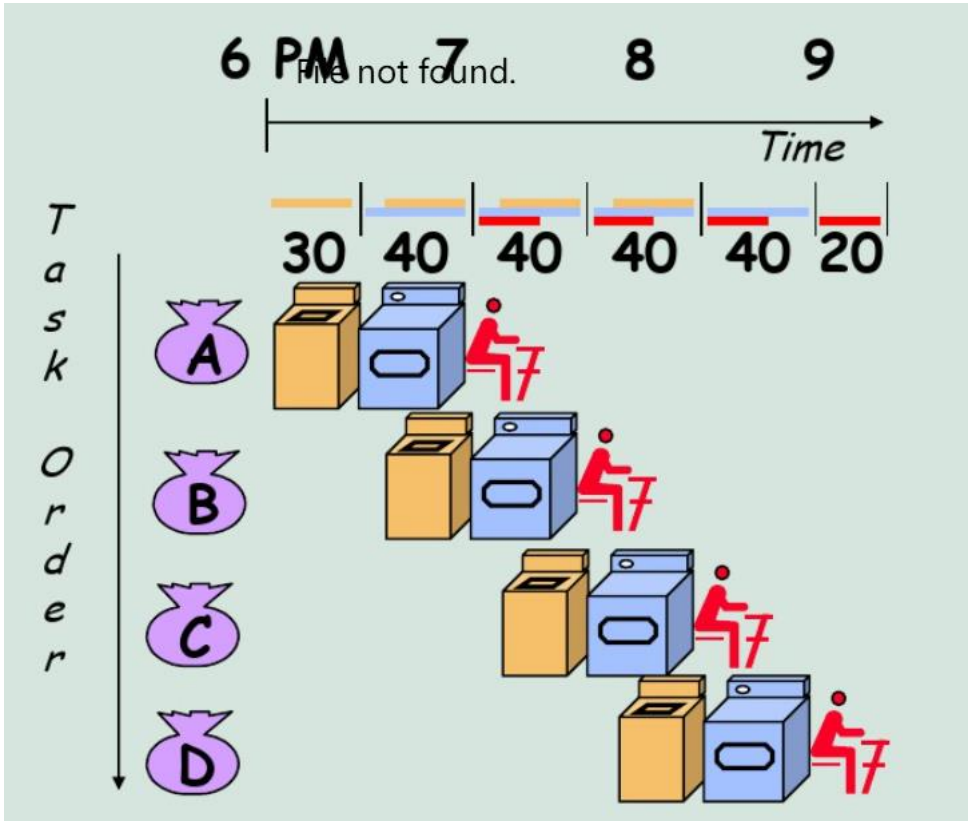
Dryer: 40minuts

Fold: 20minuts

**Total time ?**

**(30+40+20)x4 = 360**

# Idea5 :Performance via Pipelining




Extra Cost?

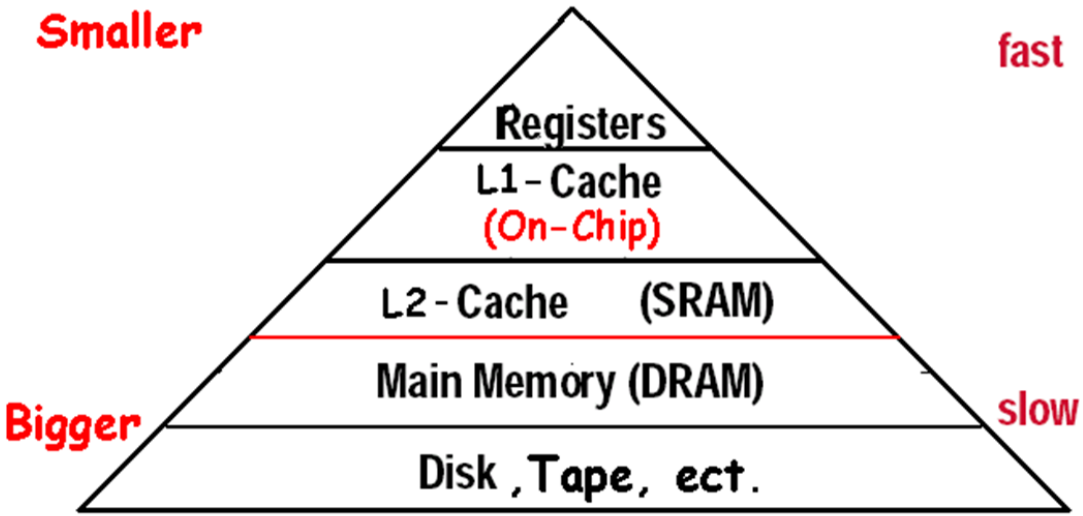
30+40*4+20=210

# Idea6 :Performance via Prediction

- To guess and start working rather than waiting until you know for sure
- The mechanism to recover from a misprediction is not too expensive and prediction is relatively accurate

# Idea7 :Hierarchy of Memories

- With the fastest, smallest, and most expensive memory per bit at the top of the hierarchy and the slowest, largest, and cheapest per bit at the bottom



# Idea8: Dependability via Redundancy

- Not only be fast, also to **be dependable**.
- Any physical device can fail.
- Make systems **dependable** by including redundant components that can take over when a failure occurs *and* to help detect failures

# Outline

- Introduction
- Computer organization
- How to build processors ?
- Computer design: performance and idea
- **What you can learn from this course ?**

# Contents

- **Chapter1: Computer Abstraction and Technology**
- **Chapter2: Instructions: Language of the Computer**
- **Chapter3: Arithmetic for Computers**
- **Chapter4: The Processor: Datapath and Control**
- **Chapter5: Large and Fast: Exploiting Memory Hierarchy**

核心内容

- Chapter 6: Parallel processor from client to Cloud (选讲，非考试内容)
- Appendix: Storage, Networks, and Other Peripherals (Ch8 of Version 3, 了解概念)

# After the course, you should know

- **The internal organization of computers and its influence on the performance of programs (处理器内部组织结构及其性能影响)**
- **The hierarchy of software and hardware**
  - How are programs written in high-level language translated into the language of the hardware, and how does it run?

    (高级语言编写的程序如何变成硬件的语言， 它是如何工作的？)
  - What is the interface between the software and the hardware, and how does software instruct the hardware to perform?

    (软硬件之间的接口是什么， 软件如何指导硬件工作)
  - What determines the performance of a program, and a programmer improve the performance?(什么决定了一个程序的性能)
  - What techniques can used to improve performance?

    (什么技术我们可以用来提高性能)

# Homework

- 1.2, 1.5, 1.6, 1.7, 1.13


END