

计算机系统基础 I

谢旻晖
2025.9

Overview of Computer Systems

Outline

- Understanding of computer systems
- Layers of computer systems

从高级语言到硬件执行

- 例子：最简单的C语言程序

```
#include <stdio.h>
```

```
int main() {  
    printf("hello, world!\n");  
}
```

从高级语言到硬件执行

• 1. 信息的存储和传输格式

- `hello.c`是文本文件，其中的字符在计算机中是以**ASCII**码的形式来表示的，每个字符用一个**0~255**的数字来表示，在计算机中占用一个字节（每个字节固定为**8**个二进制位）

#	i	n	c	l	u	d	e	<sp>	<	s	t	d	i	o	.		
35	105	110	99	108	117	100	101	32	60	115	116	100	105	111	46		
h	>	\n	\n	i	n	t	<sp>	m	a	i	n	(	)	{	\n		
104	62	10	10	105	110	116	32	109	97	105	110	40	41	123	10		
<sp>	<sp>	<sp>	<sp>	p	r	i	n	t	f	(	"	h	e	l	l	o	,
32	32	32	32	112	114	104	110	116	102	40	34	104	101	108	108	111	44
<sp>	w	o	r	l	d	!	\	n	"	)	;	\n	}				
32	119	111	114	108	100	33	92	110	34	41	59	10	125				

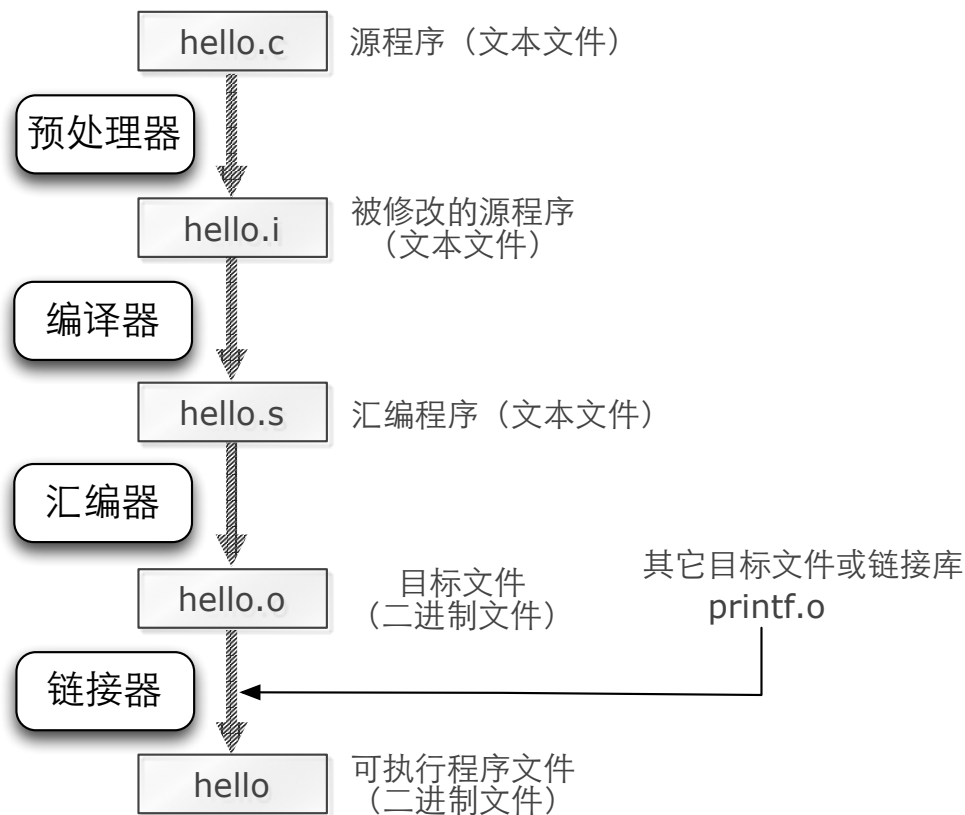
从高级语言到硬件执行

- 计算机中其他的文件基本都是二进制文件，即直接用二进制的**0**、**1**串来表示信息
 - 包括磁盘文件、存储器中的程序、存储器中存放的用户数据，以及网络上传输的数据
- 可执行程序文件中，若干个二进制字节可能对应某一条计算机硬件可以识别的指令，指挥计算机硬件执行某些操作

从高级语言到硬件执行

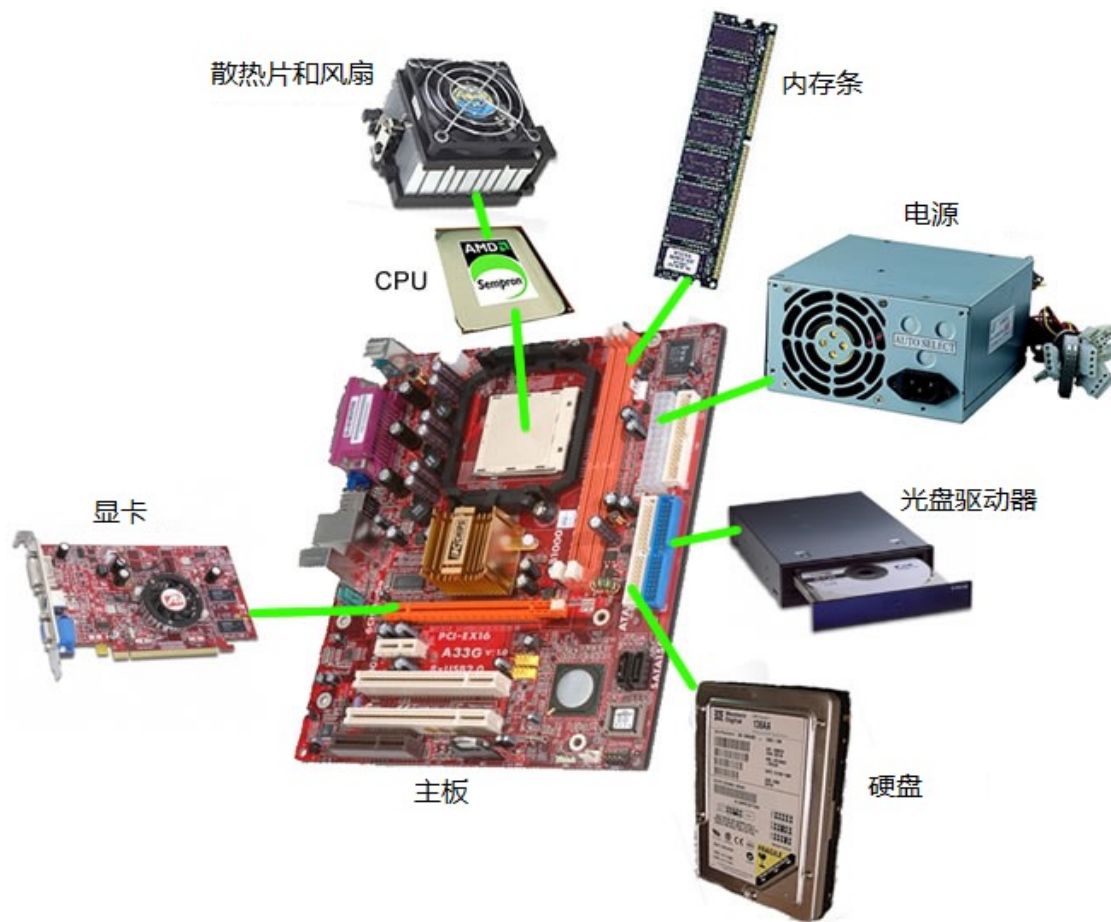
2. 从高级语言源代码到可执行程序文件生成的过程 (Unix)

- 预处理器根据以字符`#`开头的命令
- 编译器将`hello.i`翻译为汇编语句
- `printf`函数存在于一个名为`printf.o`的单独预编译好的目标文件中，这个文件必须以某种方式与我们的`hello.o`合并起来
- `./hello`运行该程序



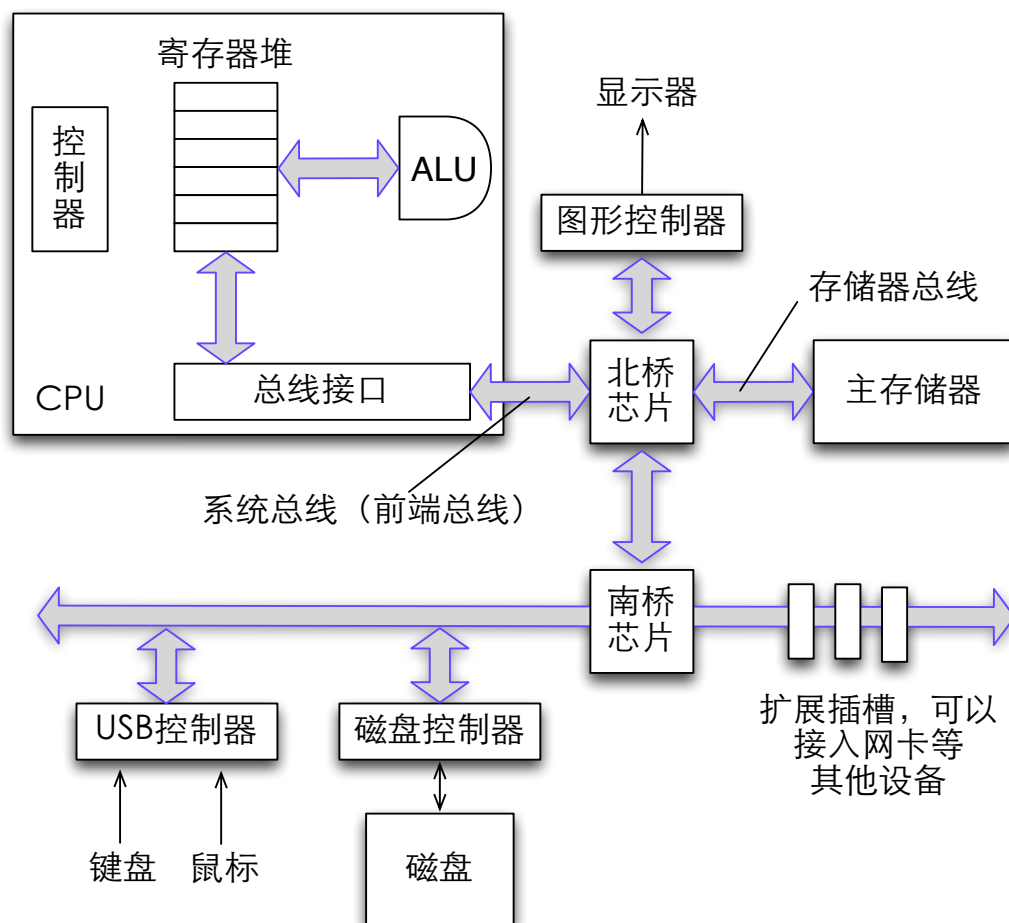
从高级语言到硬件执行

- 3. 计算机的主要硬件结构



从高级语言到硬件执行

• 3. 计算机的主要硬件结构

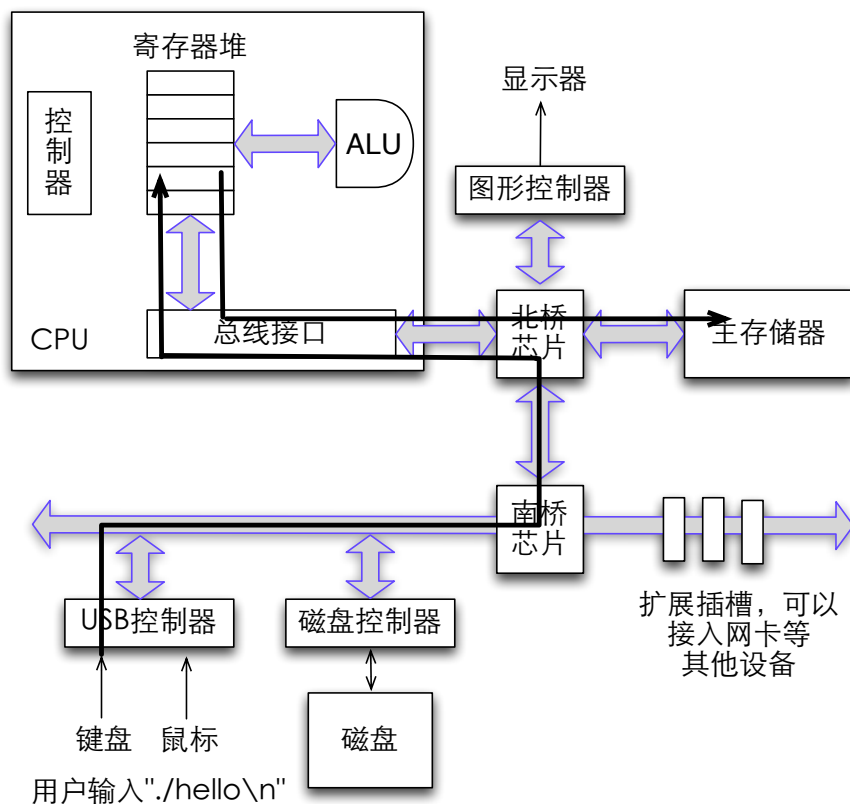


从高级语言到硬件执行

- 1) **总线**是连接各个设备的通道，负责在各个设备之间传输数据
 - CPU与北桥芯片的之间的总线被称为**系统总线或前端总线**
 - 主存储器与北桥芯片之前的总线被称为**存储器总线**
- 2) **主存储器**是一个临时存储设备，也常被称为内存，存储数据和指令
- 3) **CPU**是计算机的核心部件，是解释或执行存储在主存储器中指令的引擎
- 4) **磁盘**是一种相对慢速的存储设备，但是容量大，价格便宜，而且断电之后所存储内容不会消失

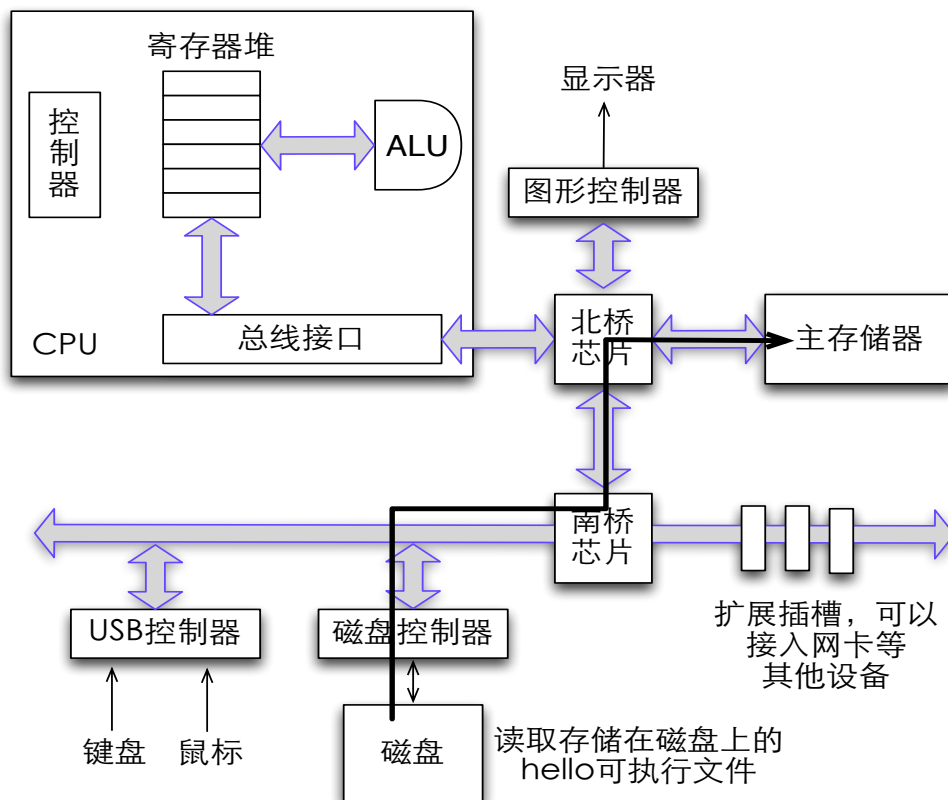
从高级语言到硬件执行

- 4. hello world程序在计算机硬件上的执行
 - 从键盘读入这些信息，并把它存放到主存储器中



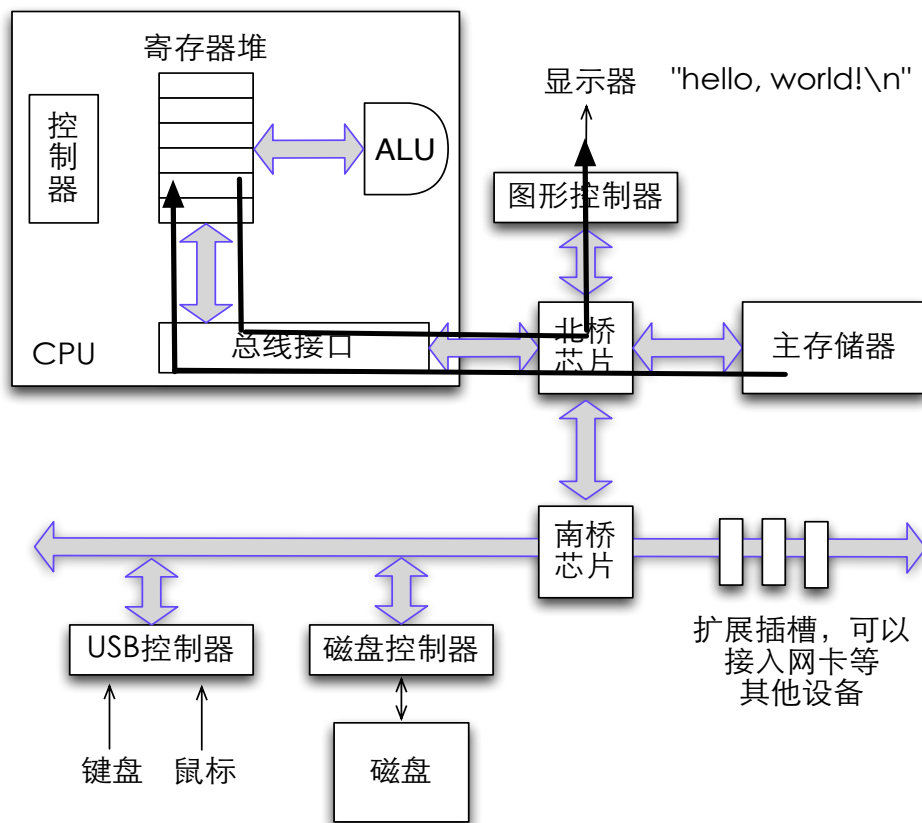
从高级语言到硬件执行

- 4. hello world程序在计算机硬件上的执行
 - 从磁盘加载可执行程序文件hello到主存储器



从高级语言到硬件执行

- 4. **hello world**程序在计算机硬件上的执行
 - 将输出字符串从内存写到显示器



计算机层次结构概述

- 计算机系统是一种典型的**层次化结构**
 - 原因主要是由于计算机系统实在太过**复杂**，一起把所有功能都实现非常困难，维护和升级也不容易。
 - 层次化可以把任务分开，每个层次只负责自己的工作，各个层次之间有标准的**接口**，这样每个层次可以独立发展，甚至有多个可用的选择
 - 例如操作系统可以选择**Windows、Linux或Mac OS**等，竞争可以促进发展。

Layers of Computer Systems

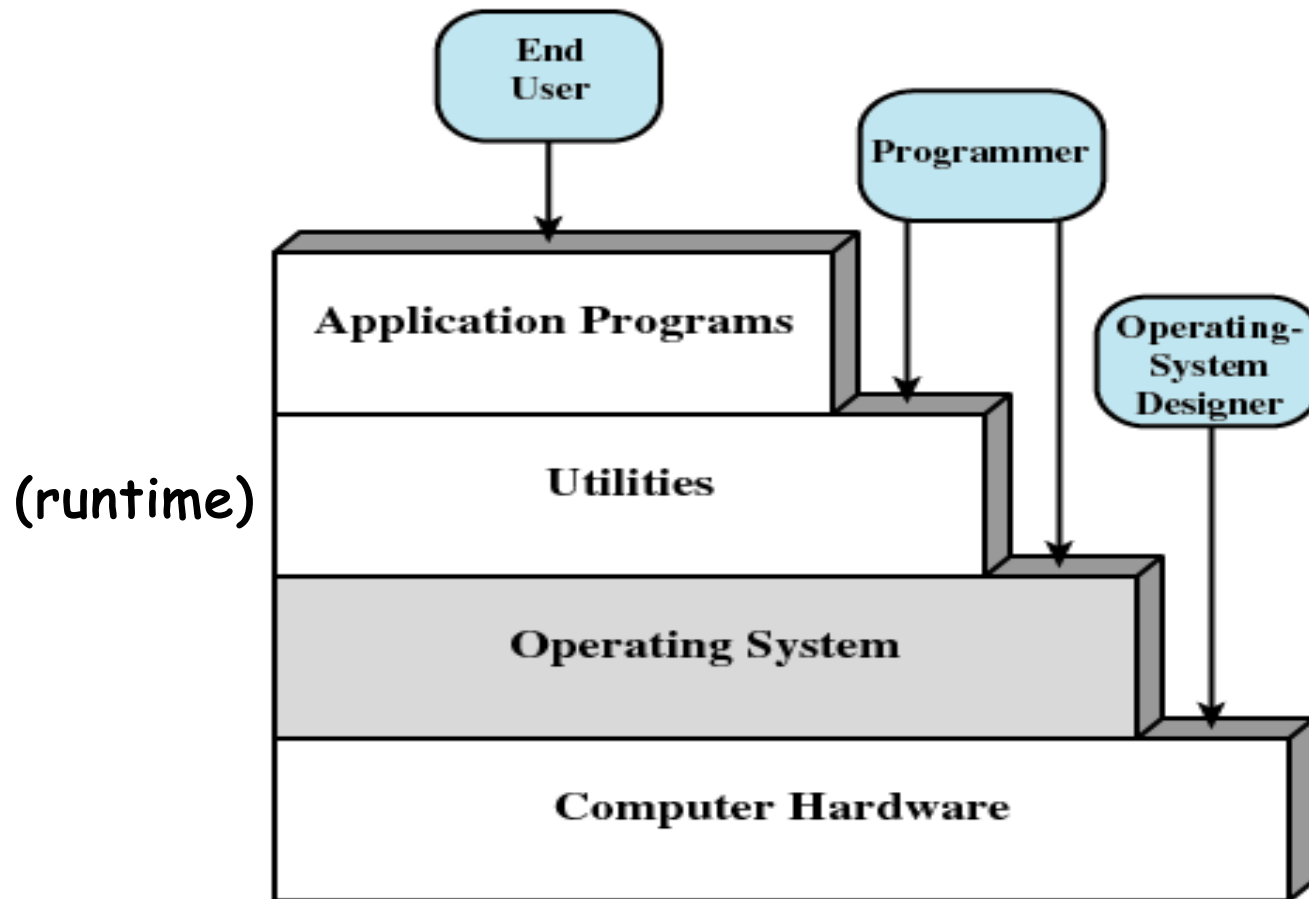
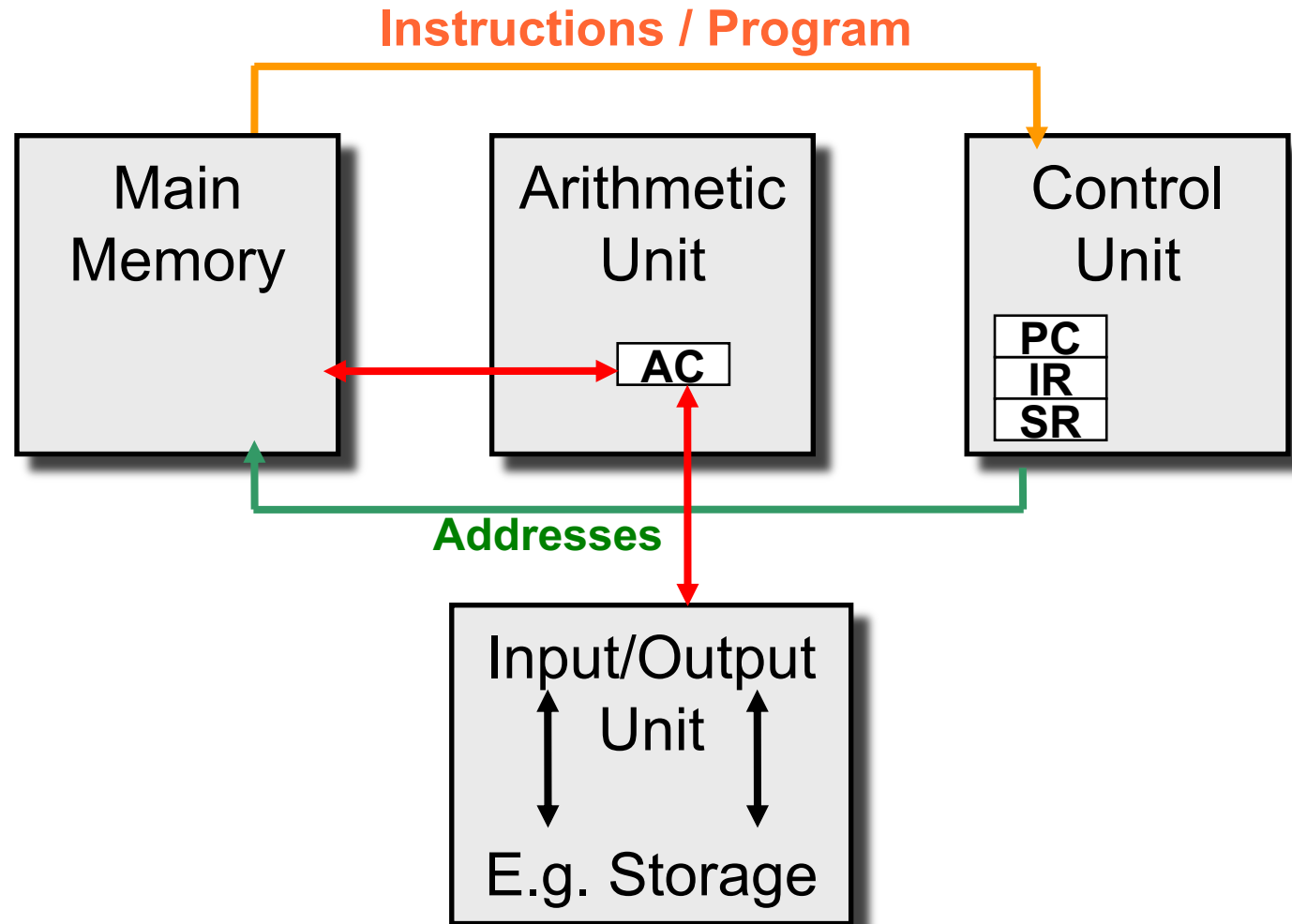


Figure 2.1 Layers and Views of a Computer System

Computer Hardware - Von Neumann Architecture



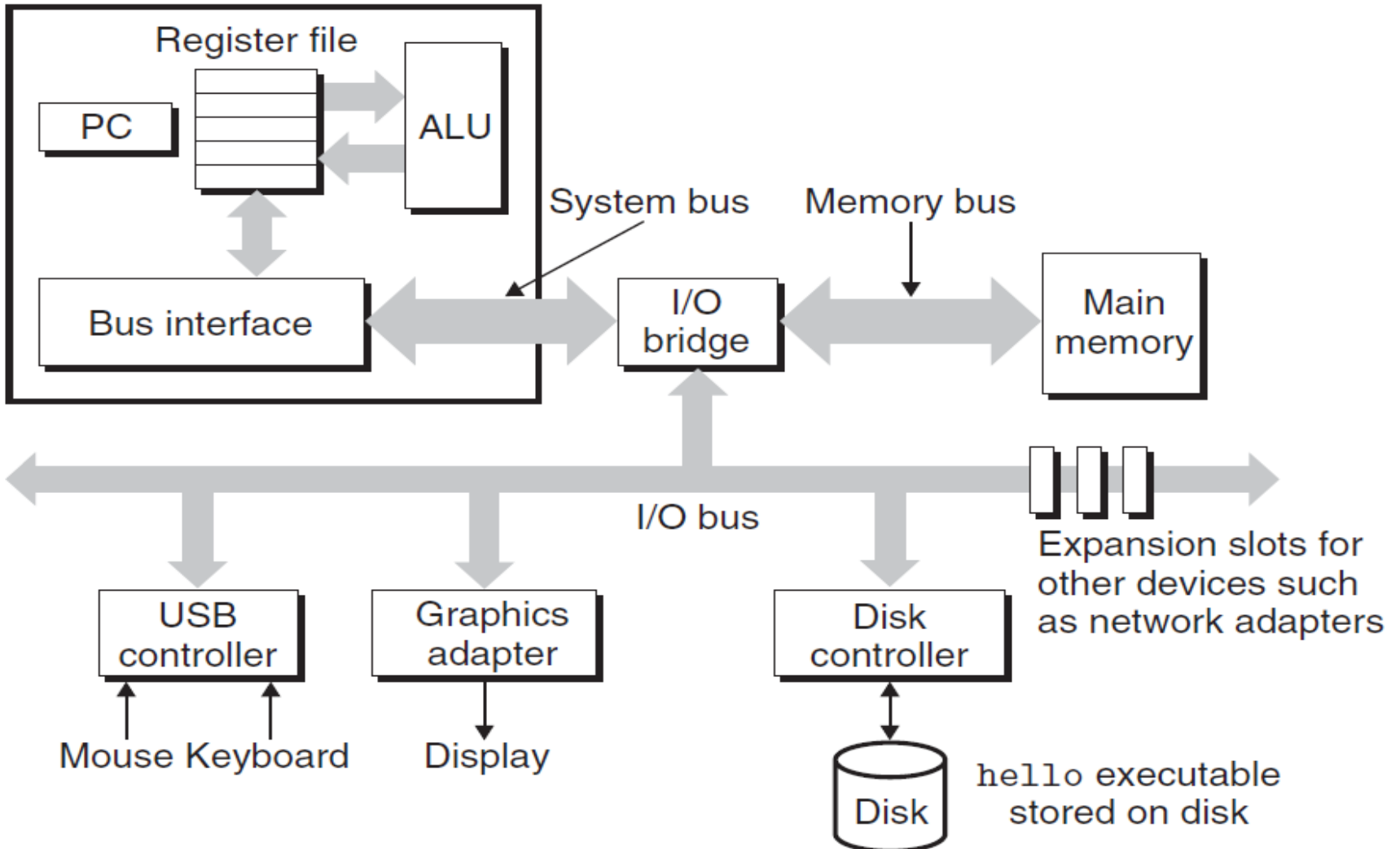
EDVAC: 冯·诺依曼架构计算机

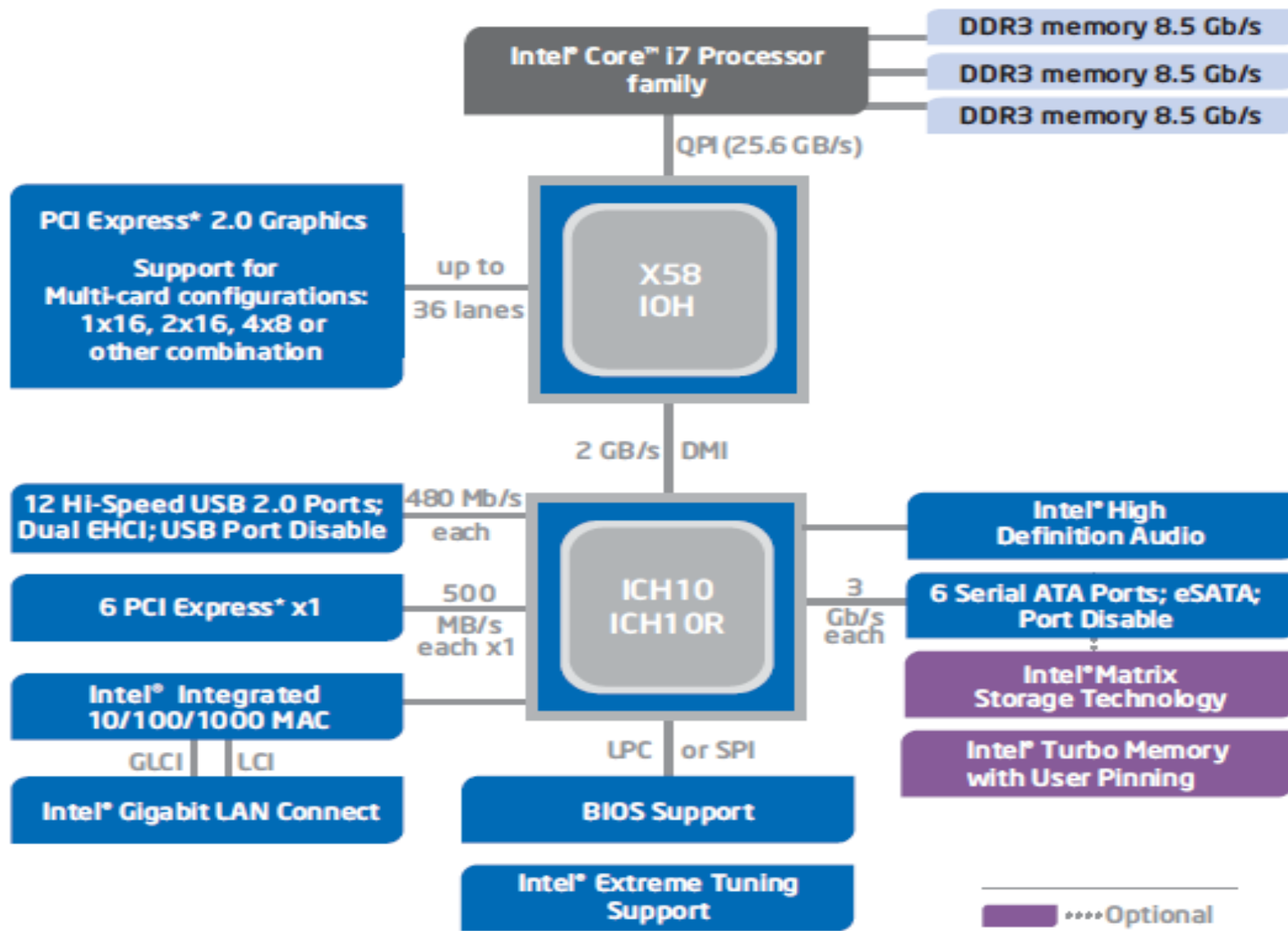


ENIAC: 世界上第一台计算机

IA-32

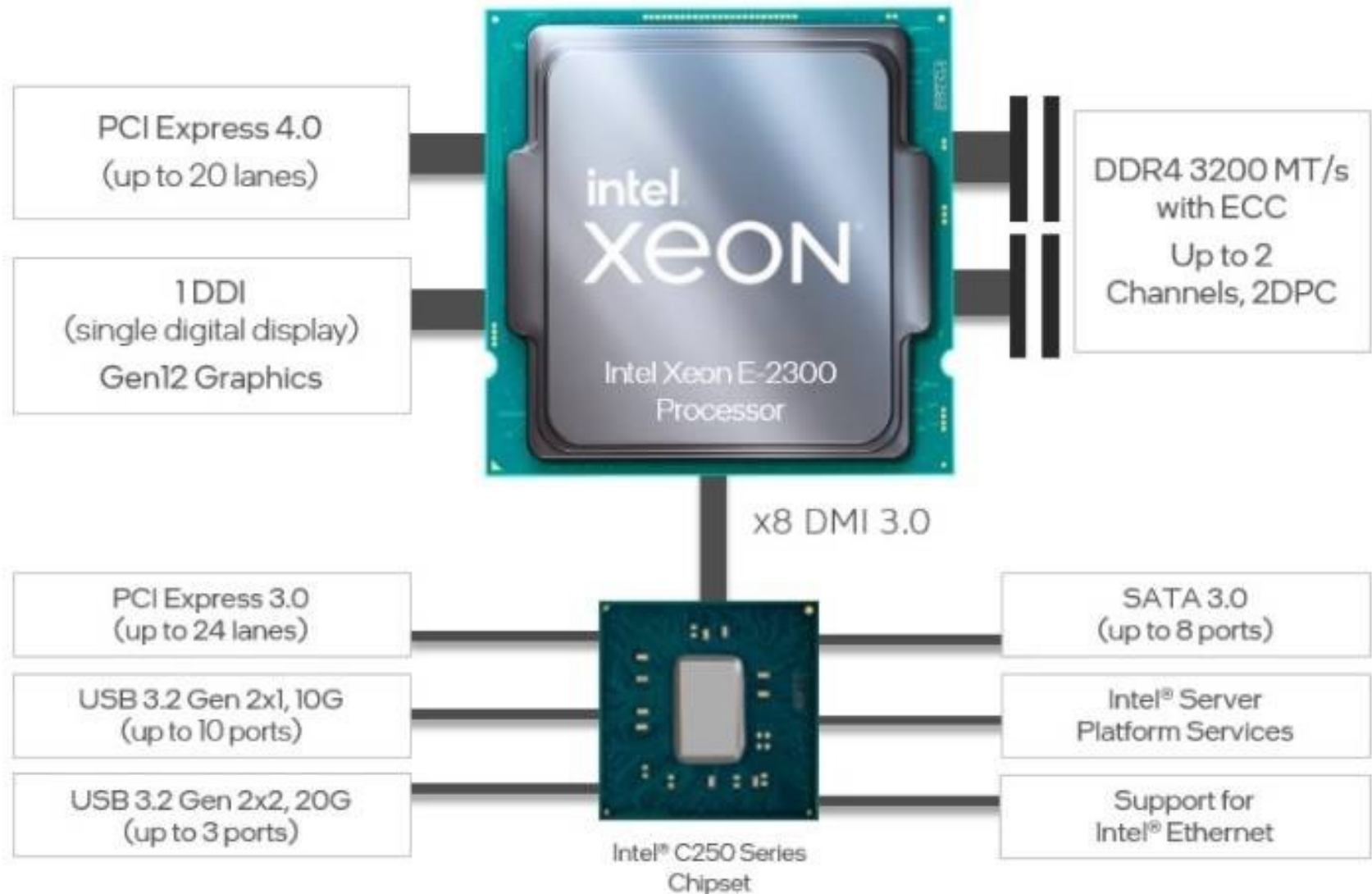
CPU





Intel® X58 Express Chipset Block Diagram

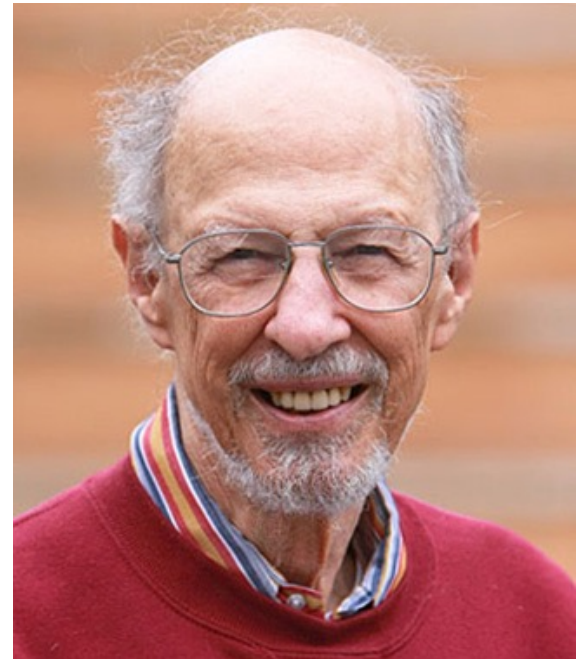
New Intel Xeon E-2300 Processors for Entry-Level Servers



AVAILABLE IN SINGLE-SOCKET CONFIGURATION ONLY

Operating Systems

- 1960's
 - IBM OS/360, Honeywell Multics
 - 贝尔实验室、MIT、通用电气公司
 - 分时操作系统，动态链接，分层文件系统
 - 失败，求全，先驱（先烈）
- Fernando Jose Corbató
 - MIT
 - IEEE Computer Pioneer Award, 1982
 - ACM Turing Award, 1990
 - his pioneering work on timesharing and the Multics operating system

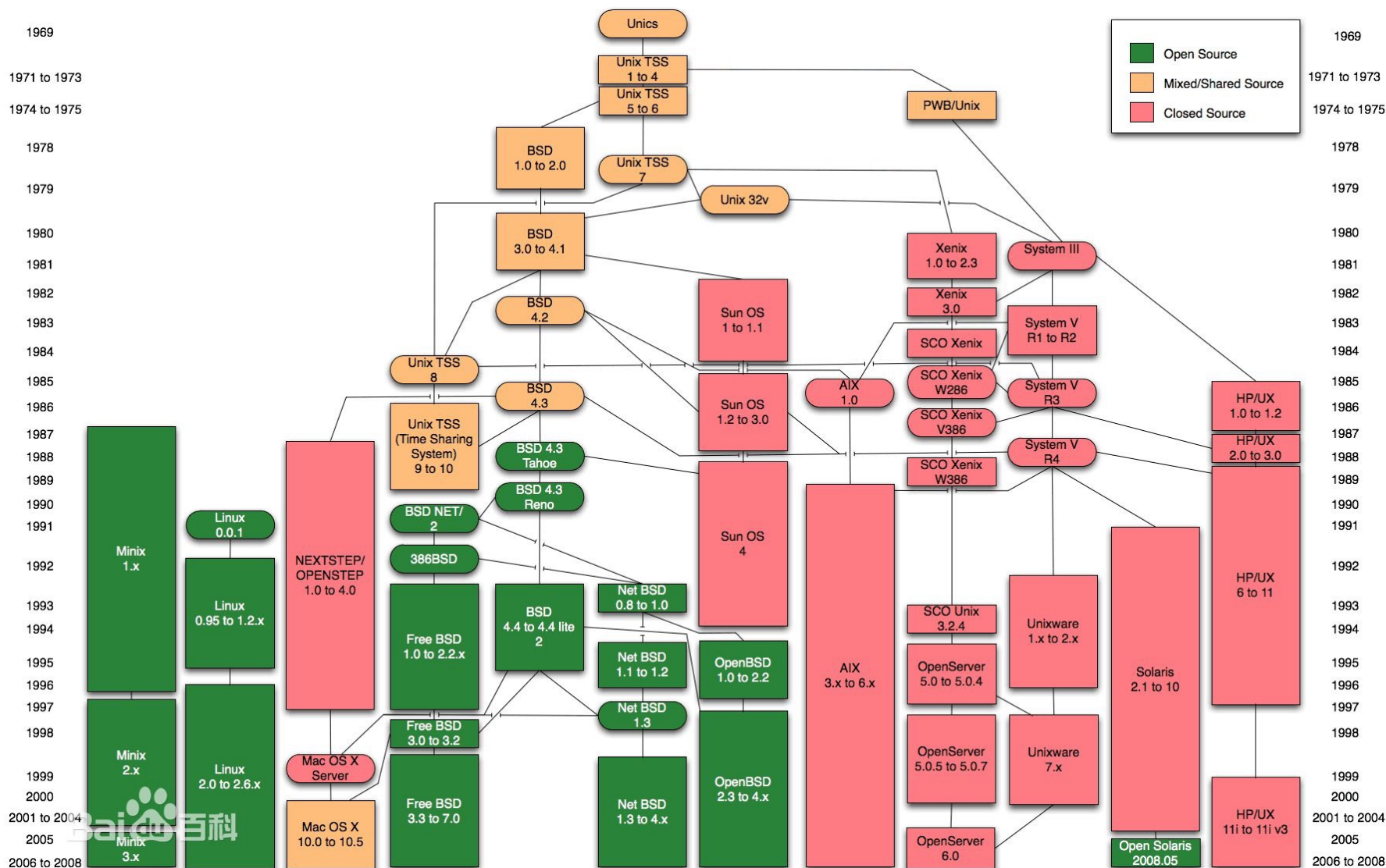


Operating Systems

- Unix
 - Bell Lab, DEC PDP-7, 1969
 - Ken Thompson, Dennis Ritchie, Doug McIlroy, Joe Ossana
 - 1970 Brian Kernighan dubbed the system "Unix"
 - Rewritten in C in 1973, announced in 1974 (开始是用汇编写的)
 - BSD (UC, Berkeley), System V(Bell lab)
 - Solaris (Sun Microsystem)
- Posix standard (OS可移植性)
- Ken Thompson, Dennis Ritchie
 - ACM Turing Award, 1983

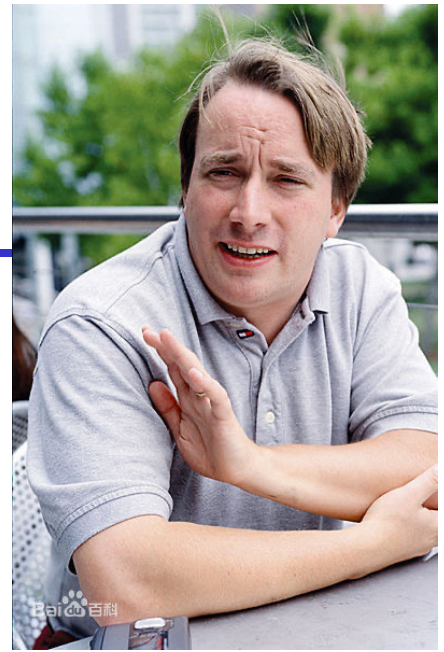


Unix家族



Linux

- 1991, Linus Torvalds
- Unix-like operating systems
- 386(486)AT, bash(1.08), gcc(1.40)
- Posix complaint version of Unix operating system
- Available on a wide array of computers
 - From handheld devices to mainframe computers
 - wristwatch



We have seen a bunch of Operating Systems



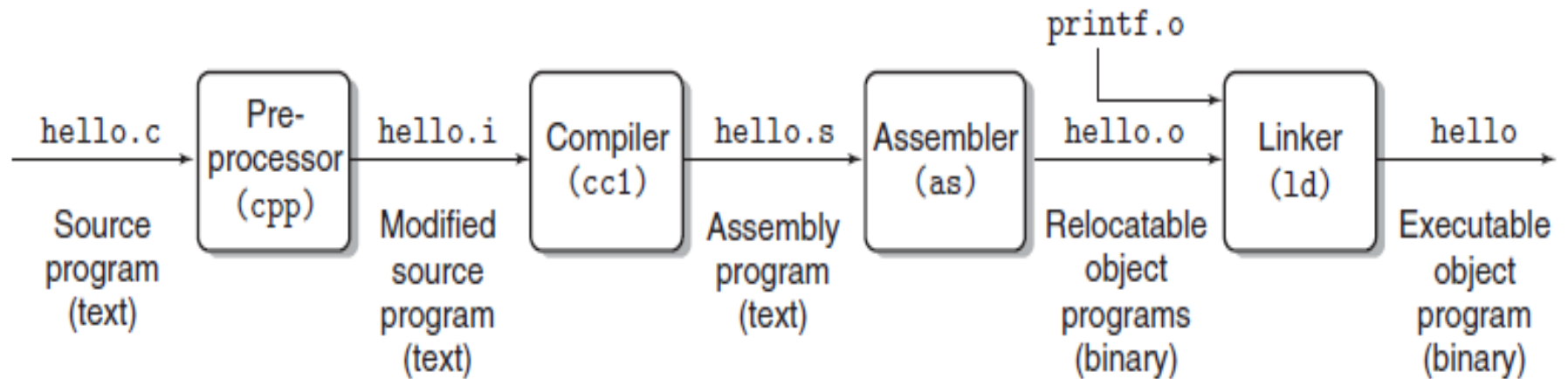
We have seen a bunch of Operating Systems



Utilities

- Programming language
 - ANSI C
- Compiler
 - GNU-gcc
- Tools
 - GNU tool chain

Compilation



GNU

- Free software
- Richard Mathhew Stallman (RMS), 1984
 - gcc, gdb, emacs
- A complete Unix-like system with source code
- An environment
 - All major components of a Unix operating system
 - Except for kernel (Linux替代了GNU的Hurd内核)

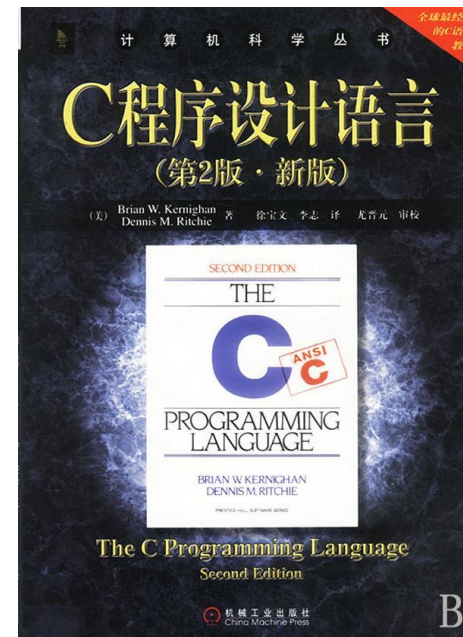


The C Programming Language

- C was developed
 - in 1969 to 1973
 - by Dennis Ritchie of Bell Laboratories.
- The American National Standards Institute (ANSI)
 - ratified the ANSI C standard in 1989.
- The standard defines
 - the C language
 - and a set of library functions known as the *C standard library*.

The C Programming Language

- Kernighan and Ritchie describe ANSI C in their classic book
 - which is known affectionately as "K&R" .
- In Ritchie's words, C is
 - quirky,
 - flawed,
 - and an enormous success.
- Why the success?



The C Programming Language

- *C was closely tied with the Unix operating system*
 - C在开始是作为Unix的编程语言而开发的
 - 绝大多数Unix内核，以及它的支撑工具和库，是用C来开发的

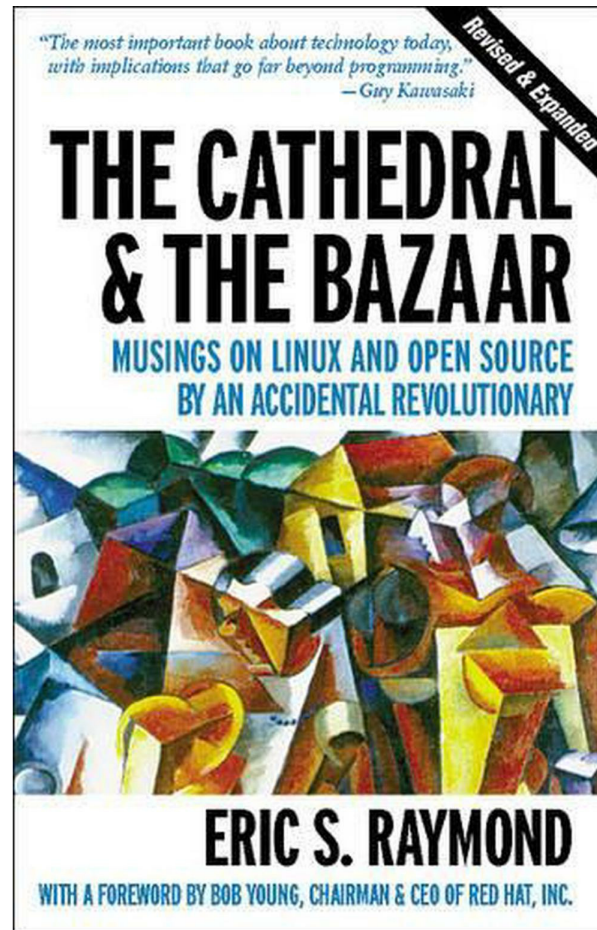
The C Programming Language

- *C was closely tied with the Unix operating system*
 - 在1970年代末和1980年代初，Unix在大学里面开始流行，这时很多人了解到C语言，并且喜欢上它
 - 由于Unix基本都是用C开发的，因此比较容易被移植到其他计算机。这种广泛的移植无形中进一步扩大了C和Unix的用户范围

The C Programming Language

- *C is a small, simple language.*
 - C语言的设计是由一个人来做的，而不是一个委员会，结果是C呈现一种干净、连贯、轻量级的设计
 - K&R这本书用261页的篇幅就描述了完整的语言 and 标准库，还包括了大量的例子和练习
 - C的简单性使他相对容易学习，而且容易移植到其他计算机
 - Assembly, Pascal, Fortran, Lisp, Prolog
 - Java, Python
 - Go, Scala

补：大教堂和市集 *The Cathedral and the Bazaar*



Worse is Better, 完美 vs. 简单

- MIT Approach: LISP 系统 (下一页补充)
- New Jersey Approach: Unix/C 系统



The Rise of ``Worse is Better''

By Richard Gabriel

I and just about every designer of Common Lisp and CLOS has had extreme exposure to the MIT/Stanford style of design. The essence of this style can be captured by the phrase ``the right thing." To such a designer it is important to get all of the following characteristics right:

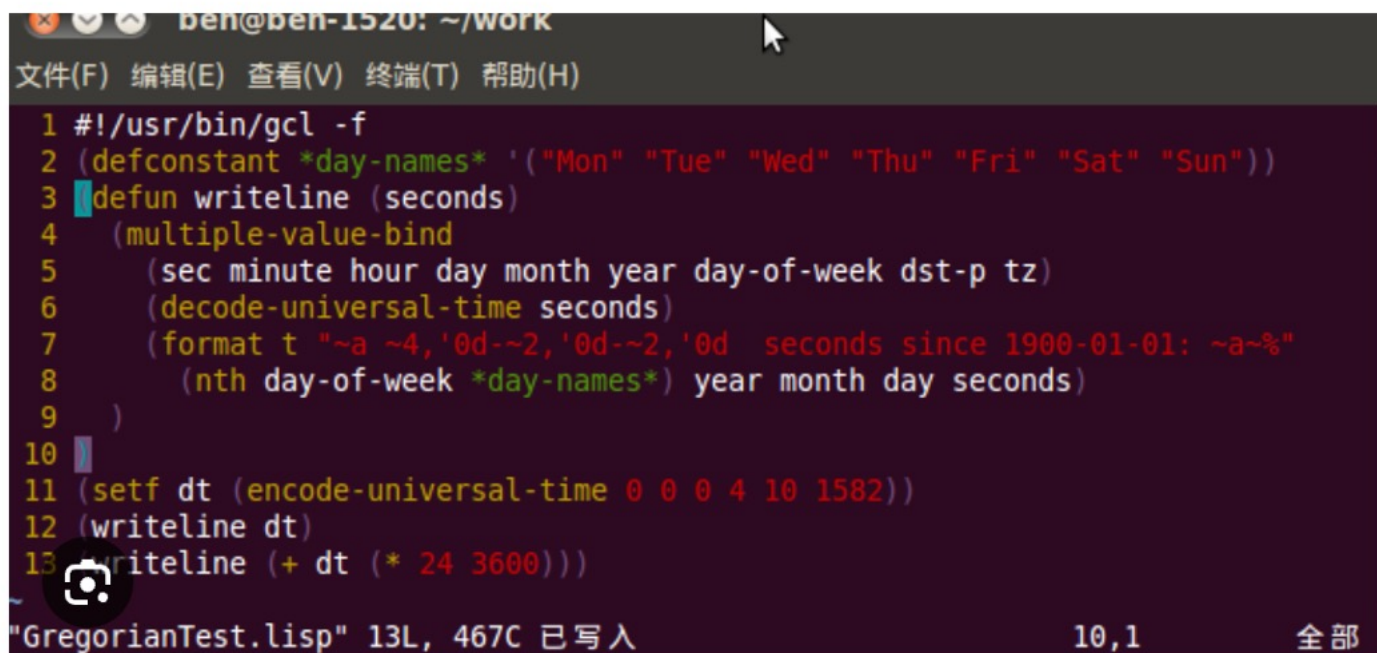
- Simplicity-the design must be simple, both in implementation and interface. It is more important for the interface to be simple than the implementation.
- Correctness-the design must be correct in all observable aspects. Incorrectness is simply not allowed.
- Consistency-the design must not be inconsistent. A design is allowed to be slightly less simple and less complete to avoid inconsistency. Consistency is as important as correctness.
- Completeness-the design must cover as many important situations as is practical. All reasonably expected cases must be covered. Simplicity is not allowed to overly reduce completeness.

I believe most people would agree that these are good characteristics. I will call the use of this philosophy of design the ``MIT approach." Common Lisp (with CLOS) and Scheme represent the MIT approach to design and implementation.

The worse-is-better philosophy is only slightly different:

- Simplicity-the design must be simple, both in implementation and interface. It is more important for the implementation to be simple than the interface. Simplicity is the most important consideration in a design.

补: LISP的复杂性



```
ben@ben-1520: ~/work
文件(F) 编辑(E) 查看(V) 终端(T) 帮助(H)
1 #!/usr/bin/gcl -f
2 (defconstant *day-names* '("Mon" "Tue" "Wed" "Thu" "Fri" "Sat" "Sun"))
3 (defun writeline (seconds)
4   (multiple-value-bind
5     (sec minute hour day month year day-of-week dst-p tz)
6     (decode-universal-time seconds)
7     (format t "~a ~4,'0d~2,'0d~2,'0d  seconds since 1900-01-01: ~a~%"
8       (nth day-of-week *day-names*) year month day seconds)
9   )
10 )
11 (setf dt (encode-universal-time 0 0 0 4 10 1582))
12 (writeline dt)
13 (writeline (+ dt (* 24 3600)))
"GregorianTest.lisp" 13L, 467C 已写入 10,1 全部
```

汇编语言、操作系统、高级语言、编译器的关系

- 疑问
 - 层次关系一般自底向上是汇编->操作系统->编译器->高级语言
 - 但为什么能用高级语言（如C）开发操作系统？
- 实际情况
 - 操作系统的来历
 - 不是铁板一块，是很多工具的集合发展而来
 - 一部分最基本的工具用汇编开发
 - 编译器能运行后，就可以用C开发其他工具了

The C Programming Language

- *C was designed for a practical purpose.*
 - 设计C语言的目的就是为了实现Unix操作系统
 - 后来人们发现C可以写他们想要的程序，就流行起来
 - 但C的本意并非如此
 - C是系统级别编程的不错选择

The C Programming Language

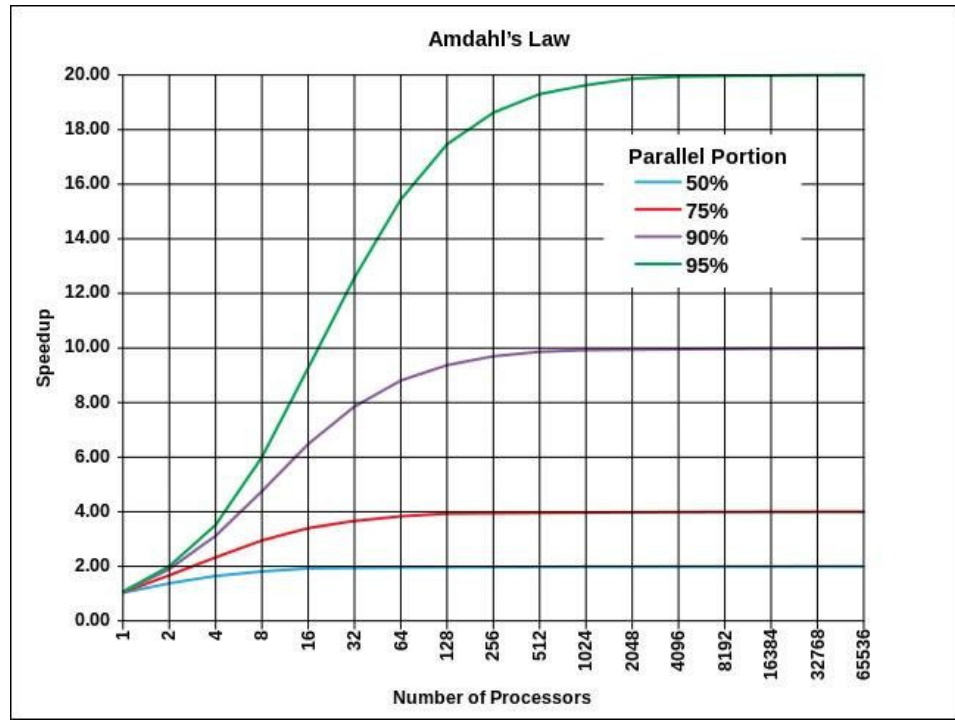
- 然而，对很多程序员和具体情况来说，**C**并不完美
 - **C**的指针经常引起迷惑和编程错误
 - **C**缺少一些抽象的显式支持（可以隐式实现，**struct** + 函数指针）
 - 新的语言如**C++**和**Java**在应用层程序层面解决了一些这样的问题

Platforms

- Hardware platform
 - Intel IA-32/x86-64
- Operating system
 - Linux
- Programming language
 - ANSI C
- Utility
 - GNU
- Networking
 - TCP/IP, Sockets

阿姆达尔定律

- 量化研究方法
 - 软硬件优化、加速
 - 并行计算、并行编程
- 阿姆达尔定律 (**Amdahl**)



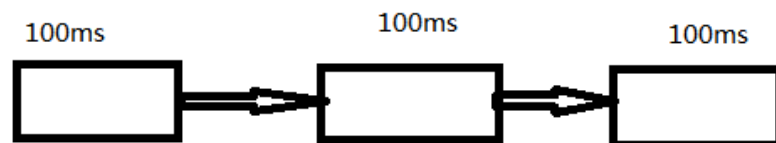
阿姆达尔:

- 改变世界的**25人**”之一;
- 主持**IBM360**的设计工作, 他提出: **360**应是一条向下兼容的生产线, 可使软件和设备用在这一家族的每个成员身上;
- 阿姆达尔定律
- <https://xueqiu.com/3993902801/83059486>⁵⁰

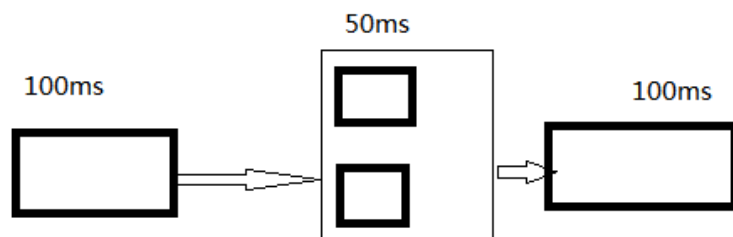
阿姆达尔定律

- 加速比 = 优化之前系统耗时 / 优化后系统耗时

$$\frac{1}{(1 - P) + \frac{P}{S}}$$



串行耗时300ms



部分并行化后耗时250ms

图形处理器中经常需要的一种转换是求平方根。浮点（FP）平方根的实现在性能方面有很大差异，特别是在为图形设计的处理器中，尤为明显。假设FP平方根（FPSQR）占用一项关键图形基准测试中20%的执行时间。

有一项提议：升级FPSQR硬件，使这一运算速度提高到原来的10倍。

另一项提议是让图形处理器中所有FP指令的运行速度提高到原来的1.6倍，FP指令占用该应用程序一半的执行时间。

请问两种方法哪个更好？

A

第一种好

B

第二种好

提交

课堂练习

- 图形处理器中经常需要的一种转换是求平方根。浮点平方根的实现在性能方面有很大差异，特别是在为图形设计的处理器中，尤为明显。假设**FP平方根（FPSQR）**占用一项关键图形基准测试中**20%**的执行时间。
 - 有一项提议：升级**FPSQR**硬件，使这一运算速度提高到原来的**10**倍。
 - 另一项提议是让图形处理器中所有**FP**指令的运行速度提高到原来的**1.6**倍，**FP**指令占用该应用程序一半的执行时间。
 - 请问两种方法哪个更好？

课后练习

- 阅读教材第1章
- 练习题1.1, 1.2