

中国科学院软件研究所

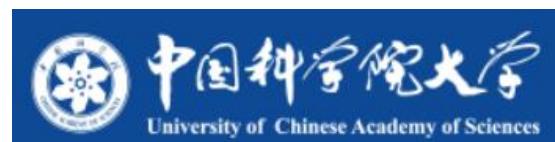
Institute of Software, Chinese Academy
of Sciences

操作系统简介

授课教师：武延军

改编声明

- 本课程教学及PPT内容基于**上海交通大学并行与分布式系统研究所**发布的操作系统课程修改，已获得原作者授权，原课程官网：
 - <https://ipads.se.sjtu.edu.cn/courses/os/index.shtml>
- 本课程由**中国科学院软件研究所**修改，用于国科大操作系统课程教学。



什么是操作系统？

应用与服务

框架：语言运行时，算子库，开发库，功能库

内核：内核和硬件驱动

芯片与硬件

大纲

- 操作系统的历史
- 什么是操作系统
- 为什么要有操作系统
- 开源：操作系统的主流发展模式
- RISC-V：操作系统的未来发展方向
- 为什么要学习操作系统

大纲

- **操作系统的[历史](#)**
- **什么是操作系统**
- **为什么要有操作系统**
- **开源：操作系统的主流发展模式**
- **RISC-V：操作系统的未来发展方向**
- **为什么要学习操作系统**

计算机诞生时的操作员

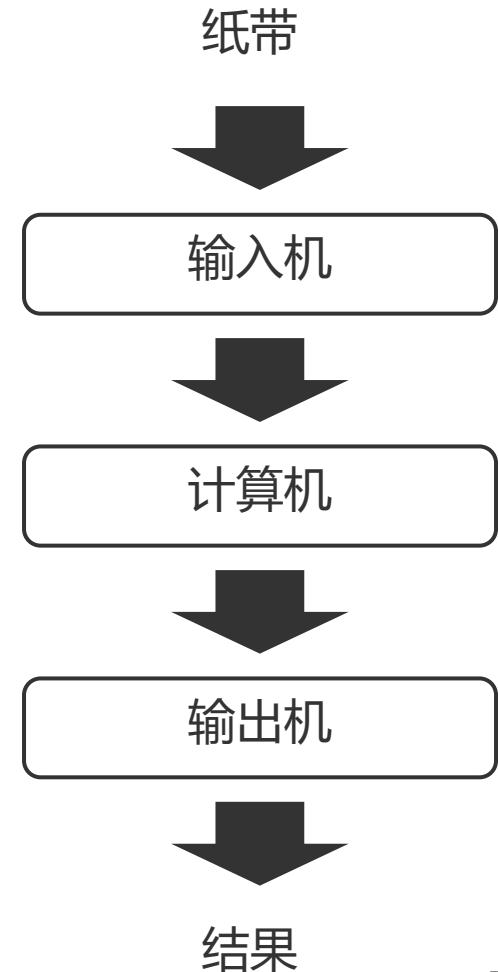
1946年2月14日，世界上第一台通用计算机ENIAC在美国宾夕法尼亚大学诞生，请在这个特殊的节日多陪陪你的电脑



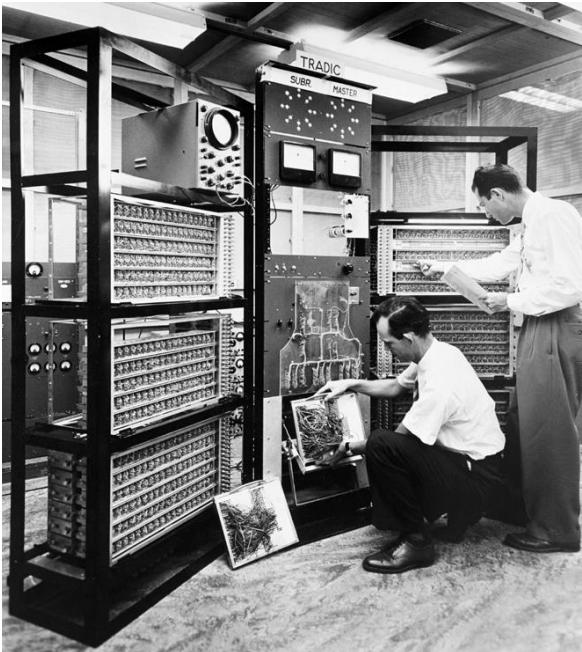
人工操作系统



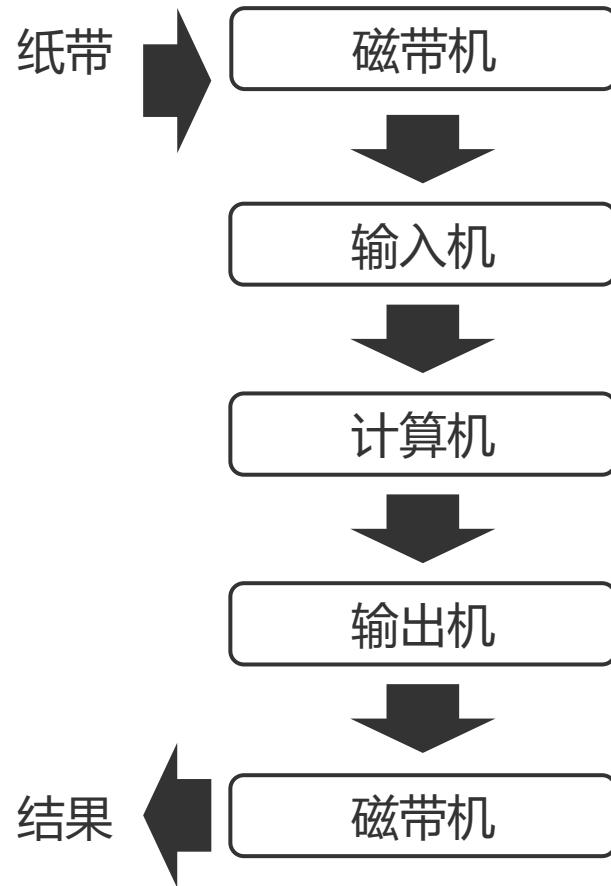
穿孔纸带



批处理操作系统



TRADIC 1954
第一台晶体管计算机

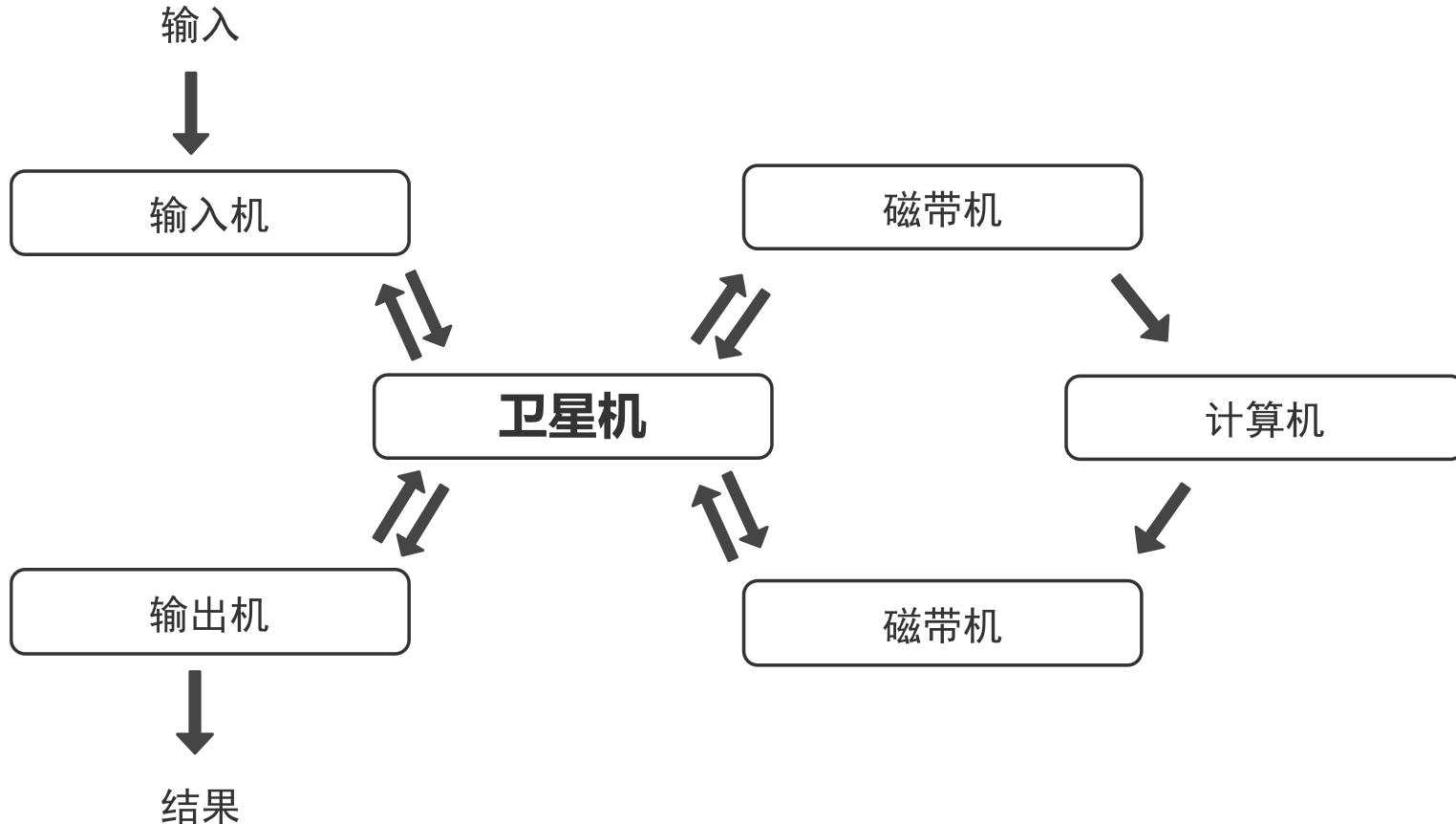


批处理操作系统：GM-NAA I/O

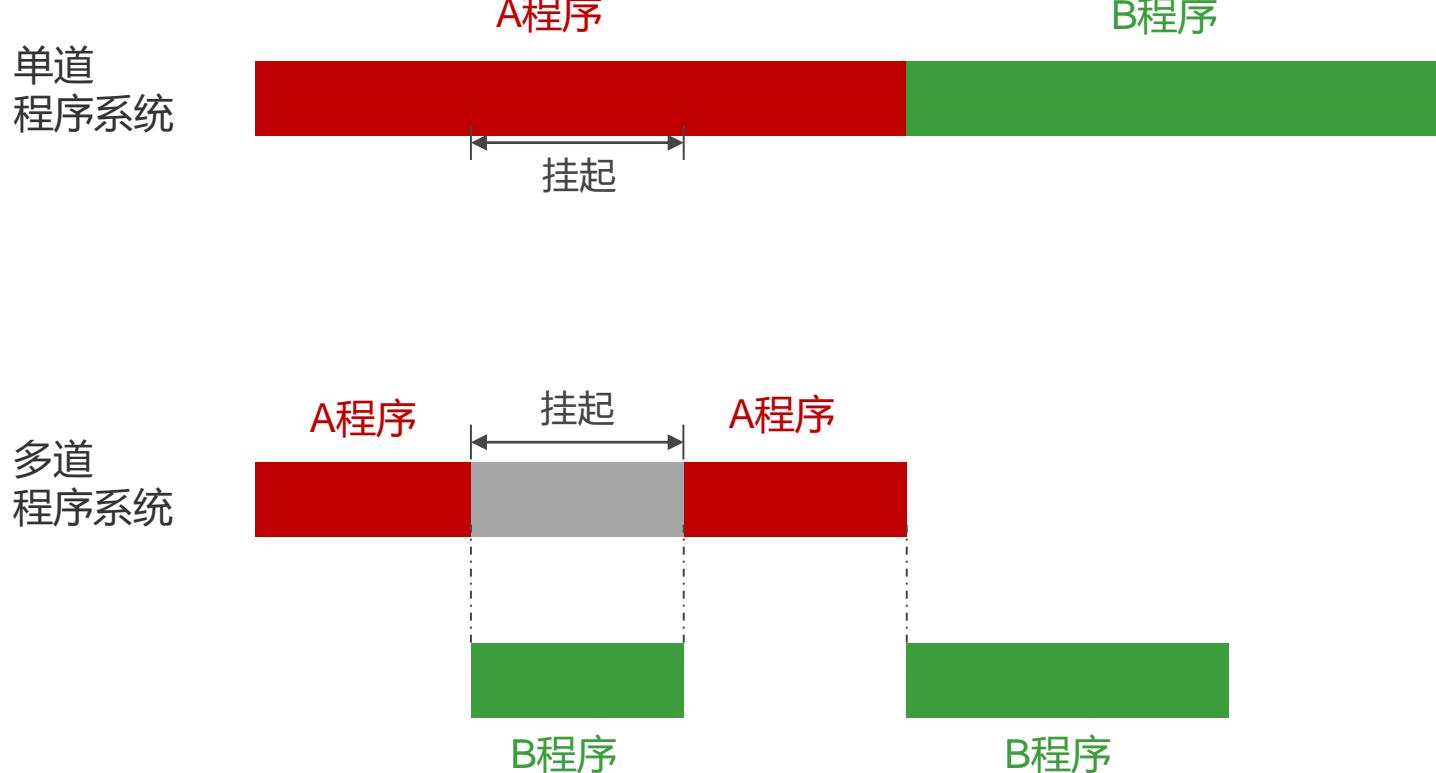


- Robert L. Patrick和Owen Mock于1956年建设
 - 运行在IBM 704上
 - 主要功能：批处理运行任务

脱机批处理系统



多道程序系统

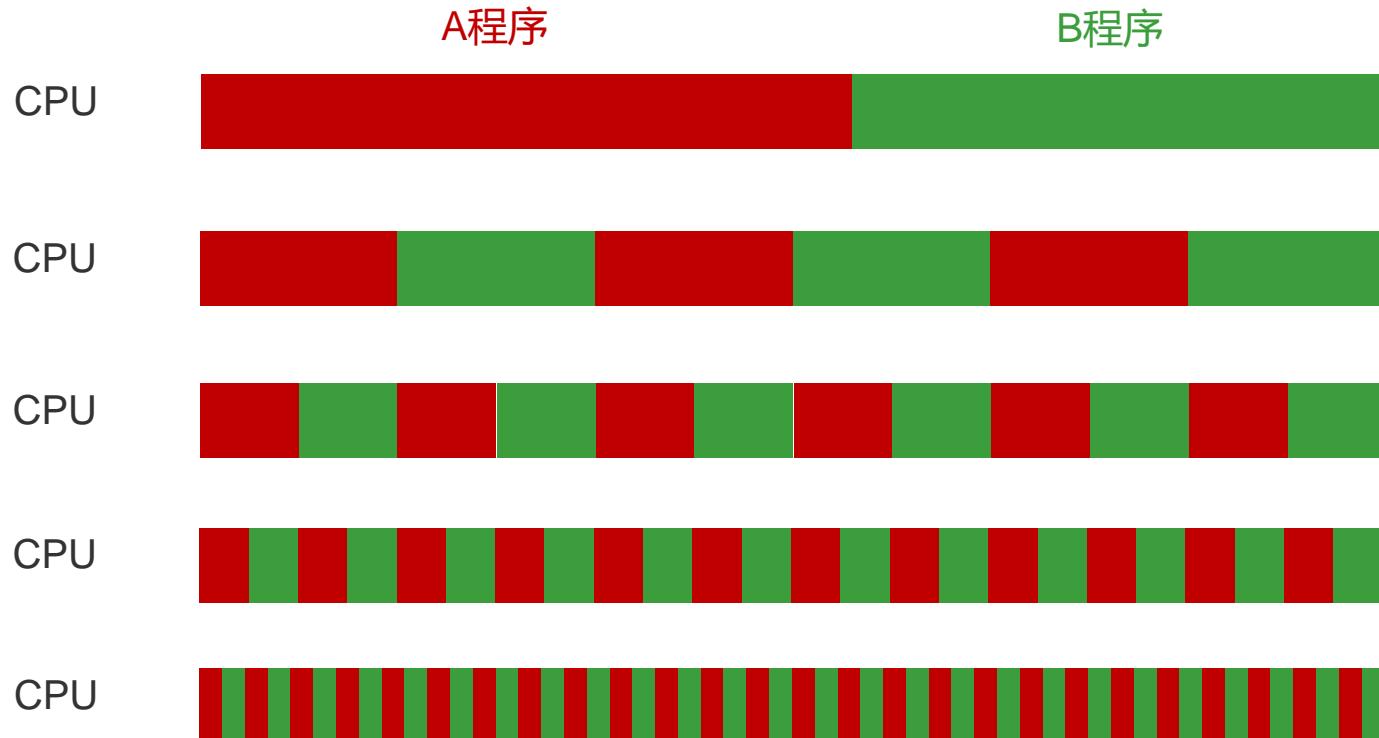


通用操作系统：OS/360

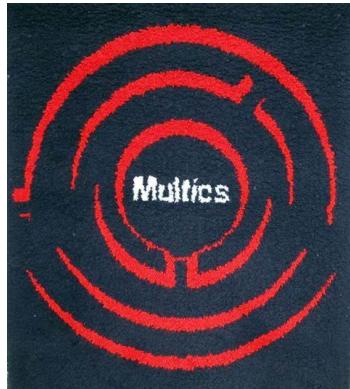


- **IBM System/360 OS, 1964**
 - 首个通用操作系统，首次将操作系统与计算机分离
 - 架构师：Gene Amdahl (Amdahl's Law)
 - 项目经理：Fred Brooks (《人月神话》，1999年图灵奖得主)

分时系统



过于超前的产品



Multics (1965)

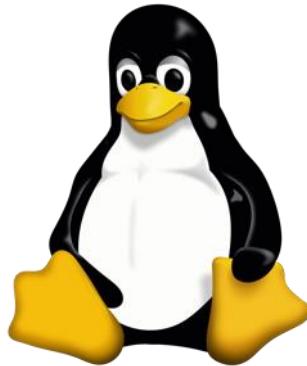
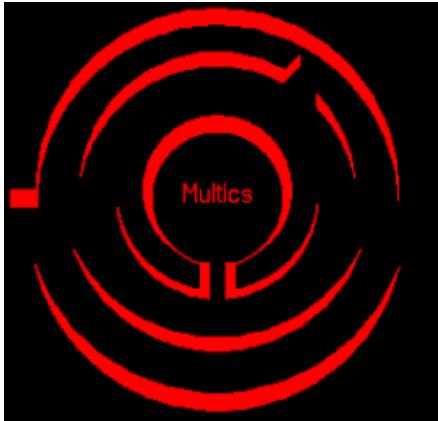


Windows平板 (1991)



Google Glass (2012)

分时与多任务： Multics/Unix/Linux



Multics: Fernando Corbató
(1990年图灵奖) MIT/GE, 1964
分时, 文件系统, 动态链接等

Unix: Ken Thompson, Dennis Ritchie (1983年图灵奖), 1969
Shell, 层次化文件系统

Linux: Linus Torvalds, 1991
最流行的开源操作系统

Unix: Cat Command

```
[~]$ cat
```

© www.SoftwareTestingHelp.com

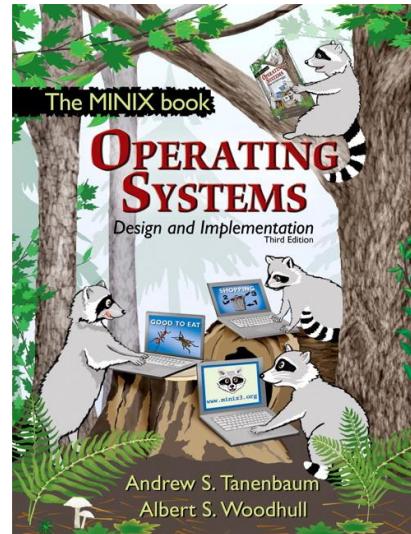
Minix



Andrew Stuart Tanenbaum



Minix



《操作系统：设计与实现》

Linux发明人 – Linus Torvalds



Linus Torvalds

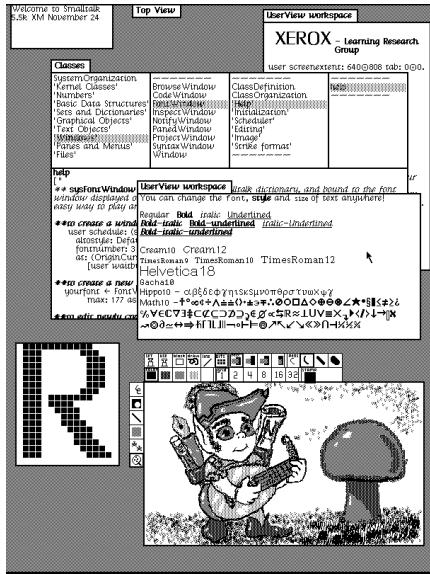
1991.1

1991.9.17

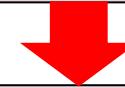
Linux 0.01
(10239行代码)



图形界面： Xerox Alto/MacOS/Windows



Xerox Alto (1973) : 第一个图形化操作系统，首次使用鼠标 (Chuck Thacker, 2009年图灵奖)

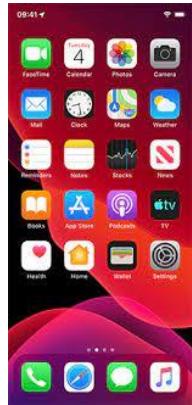


Mac OS (Apple LISA, 1983): 1979年乔布斯访问Xerox PARC，意识到GUI的重要性，买下了GUI进行研究



Windows 1.0 (1985) : 基于图形界面的操作系统

现代操作系统



现代操作系统



苹果，正在准备新操作系统，名为RealityOS

2022-02-11 13:31:53 来源: Innovation科技资讯

举报



AR·VR设备专用OS的开源代码泄露

有消息称，苹果正在开发用于增强现实(AR)和虚拟现实(VR)设备的新操作系统(OS)。IT媒体MacRumors最近报道称，苹果正在开发的AR·VR头戴设备的新OS，“reality OS”的源代码在开发者社区GitHub中流出。



后来其实是 VisionOS.....

今天，操作系统空前繁荣



这些设备中，都有操作系统么？

大纲

- 操作系统的历史
- 什么是操作系统
- 为什么要有操作系统
- 开源：操作系统的主流发展模式
- RISC-V：操作系统的未来发展方向
- 为什么要学习操作系统

什么是操作系统？

- 你觉得以下哪些属于操作系统？
 - A. Windows 10所包含的所有软件
 - B. Linux内核以及所有设备的驱动
 - C. 在Macbook上下载安装的第三方NTFS文件系统
 - D. 华为Mate 30出厂时所有的软件
 - E. 大疆无人机出厂时所有的软件
 - F. 火星车上运行的软件
 - G. 运行在用户态的I/O框架
- 你觉得应当如何定义操作系统？

操作系统：在硬件和应用之间的软件层

操作系统和应用的关系

- **服务**应用：如提供硬件操作
- **管理**应用：如加载、调度等



操作系统和硬件的关系

- **管理**硬件：操作硬件以完成功能
- **抽象**硬件：应用不关心硬件差异

“**操作系统是管理硬件资源、控制程序运行、改善人机界面和为应用软件提供支持的一种系统软件。**”

《计算机百科全书(第2版)》

“**操作系统将有限的、离散的资源，高效地抽象为无限的、连续的资源。**”

《现代操作系统：原理与实现》

究竟什么是操作系统？

- “**操作系统**”并没有严格的唯一定义
 - 是一个相对概念：相对“应用”而言
 - 操作系统也可包含运行在用户态的框架（framework）
- **例如：SSL库、Dalvik（Java虚拟机）是否属于操作系统？**
 - 对Android来说，属于操作系统框架层；App属于应用
 - 对Linux来说，不属于操作系统；hello属于应用
- **我们课程中的操作系统：以hello作为典型应用**

从 Hello World 说起

```
#include <stdio.h>

int main()
{
    printf("Hello World!\n");
    return 0;
}
```

运行hello时，操作系统的作用？

```
bash$ gcc hello.c -o hello

# 运行一个hello world程序
bash$ ./hello
Hello World!

# 同时启动两个hello world程序
bash$ ./hello & ./hello
[1] 144
Hello World!
Hello World!
[1]+ Done          ./hello
```

操作系统考虑的一些问题

- hello 这个可执行文件存储在什么位置？是如何存储的？
- hello 这个可执行文件是如何从硬盘加载到内存中运行的？
- hello 这个可执行文件是如何将"Hello World!"这行字输出到屏幕？
- 两个 hello 程序是如何同时运行在一个 CPU 上的？
- 两个 hello 程序之间如果互相通信该怎么做？
- 如果其中一个 hello 出bug了，如何保证另一个能不受影响正常运行？
- 如何同时运行1,000个 hello？10,000个？1,000,000个？
- ...

操作系统的工作：**1、服务应用；2、管理应用**

操作系统为应用提供的服务（部分）

- **为应用提供计算资源的抽象**
 - CPU：进程/线程，数量不受物理CPU的限制
 - 内存：虚拟内存，大小、连续性和隔离性等不受物理内存的限制
 - I/O设备：将各种设备统一抽象为文件，提供统一接口
- **为应用提供线程间的同步**
 - 应用可以实现自己的同步原语（lock）
 - 操作系统提供了更高效的同步原语（与线程切换配合，如futex）
- **为应用提供进程间的通信**
 - 应用可以利用网络进行进程间通信（如loopback设备）
 - 操作系统提供了更高效的本地通信机制（具有更丰富的语义，如pipe）

操作系统对应用的管理（部分）

- **应用生命周期的管理**
 - 应用的加载、迁移、销毁等操作
- **计算资源的分配**
 - CPU：调度机制
 - 内存：内存分配机制
 - I/O设备：设备的复用与分配机制
- **安全与隔离**
 - 应用程序内部：访问控制机制
 - 应用程序之间：隔离机制，包括错误隔离和性能隔离

操作系统的功能：管理

- 问：如何避免一个流氓应用独占CPU资源？
- 方法-1：每10ms发生一个时钟中断（时间片）
 - 调度器决定下一个要运行的任务
- 方法-2：可通过信号等打断当前任务执行
 - 如：kill -9 1951

rogue.c

```
int main () {  
    while (1);  
}
```

操作系统的功能：管理

- 问：如何通过3行代码卡死一个OS?
 - 例：rogue-1.c 可以fork出无数的进程
- 如何解决这个问题?
 - 方法一：万能方法——重启
 - 方法二：将代码运行在虚拟机中，虚拟机外的应用不受影响
 - 方法三：Linux cgroup，预先设置某个应用允许占用的资源

rogue-1.c

```
int main () {  
    while (1)  
        fork();  
}
```

大纲

- 操作系统的历史
- 什么是操作系统
- **为什么要有操作系统**
- 开源：操作系统的主流发展模式
- RISC-V：操作系统的未来发展方向
- 为什么要学习操作系统

应用与操作系统的解耦

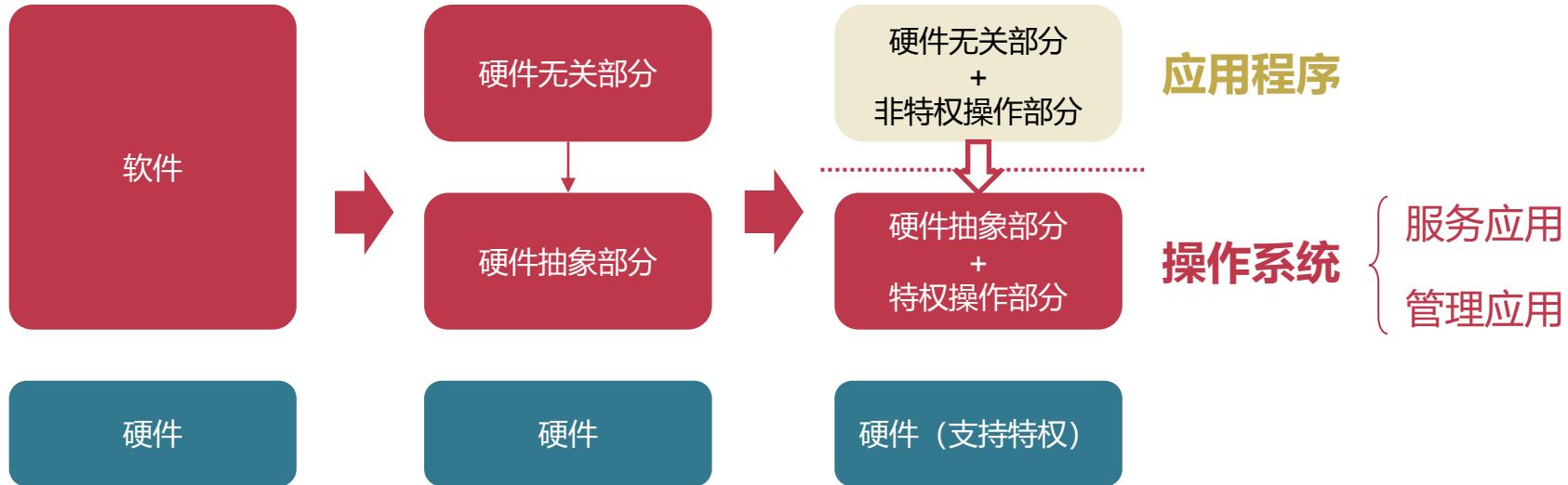
- 在最早的时候，并不区分应用与操作系统
 - 所有功能都在一起，负责开发的也是同一批人
- 随着功能越来越多，开始出现分化和分工
 - 有些底层功能被频繁复用，如对存储设备的操作
 - 这些常用的功能形成了特定的模块，通过接口与其他模块交互
 - 例如：存储模块，可将存储抽象成一个大数组
 - 但这并不一定需要操作系统——比如，可以以库的方式存在
- 那么，为什么不能把操作系统作为一个库呢？

特权级的必要性

- **多道 (Multi-programming) 操作系统的诞生**
 - 通过时分复用计算资源的方式，在一台计算机上同时运行多个应用程序
- **新的问题：如何保证不同应用间的隔离？**
 - 如果所有的应用均能完全控制硬件计算资源，则会导致混乱
 - 例如：某个应用希望关机，某个应用希望格式化硬盘
 - 因此必须先让应用降权，不允许直接改变全局的系统状态
 - 例如：中断是否打开
- **方案：必须要有不同的权限级——至少两种权限**
 - 低权限：不允许改变全局系统状态，用来运行应用
 - 高权限：集中运行能改变全局系统状态的操作，分化出操作系统（内核）

特权操作

操作系统的分化



对硬件 (CPU) 的要求

- CPU对软件提供的接口称为ISA
 - ISA: Instruction Set Architecture
 - 包含指令、寄存器等软件可见、可操作的接口
- CPU相应分化出两个模式：非特权模式和特权模式
 - 非特权模式ISA：应用可使用的指令和寄存器
 - 包括各种运算指令、通用寄存器等
 - 特权模式ISA：只有操作系统才可使用的指令和寄存器
 - 包括各种特权指令、系统寄存器等
 - 从上到下的切换过程通常称为陷入（trap）

非特权部分与特权部分的交互

- **系统调用 (System Call)**
 - 应用调用操作系统的机制，实现应用不能实现的功能
 - 应用通过CPU的陷入机制进行模式切换（非特权→特权）
 - 有多种陷入方式，包括特定指令、异常、硬件中断等
 - 操作系统内核通过特定的硬件指令返回应用（特权→非特权）
 - 通常只有一种返回方式：如ARM的eret
- **应用调用操作系统的功能，就像调用普通函数一样**
 - 例如：库函数 printf() → 系统调用 write() → 内核实现 sys_write()
 - `write(1, "Hello World!\n", 13)`

>Hello运行中的系统调用 (strace)

```
/* 运行hello程序 */
execve("./hello", ["./hello"], 0x7ffed5a79e80 /* 64 vars */) = 0
...
/* 将``Hello World!\n''写到标准输出中，在这里，1代表标准输出，13代表一共写了13个字符。 */
write(1, "Hello World!\n", 13Hello World!) = 13
/* 执行结束后，hello程序退出*/
exit_group(0)
```

问题

- **问题一：**如果一台机器有且只有一个应用程序，开机后自动运行且不会退出，是否还需要操作系统？

问题

- **问题二：**如果一个应用希望自己完全控制硬件而不是使用操作系统提供的抽象，是否还需要操作系统？

操作系统的两种演化

- **外部演化**
 - 接口的演化：更好的应对新场景
 - POSIX接口：定义了一组系统调用的接口，以为应用提供兼容性
 - Linux：系统调用不断有新的加入、旧的退出
 - 鸿蒙：分布式软总线等
- **内部演化**
 - 架构的演化：更好地应对复杂性
 - 如：宏内核架构、微内核架构、外核架构、多内核架构等
 - 更好的扩展性、容错性、安全性、兼容性、灵活性、性能等

例：图形界面是在用户态还是内核态？

- **Windows的GUI**
 - 早期放在内核态，但代码量过于庞大导致内核臃肿
 - 后来放到用户态（Windows NT内核初期）
 - 再后来又部分回到内核态，从而获得更高的性能（低时延）
- **Linux的GUI**
 - 用户态的X Window，将图形显示设计成网络协议
 - 任意应用的界面均可在任意显示器显示（类比远程桌面）
 - 缺点是性能差，尤其是时延过高

现代意义上的操作系统

移动服务：应用商店，本地服务，云服务

授权

移动应用



目的：控制生态

应用框架

开源

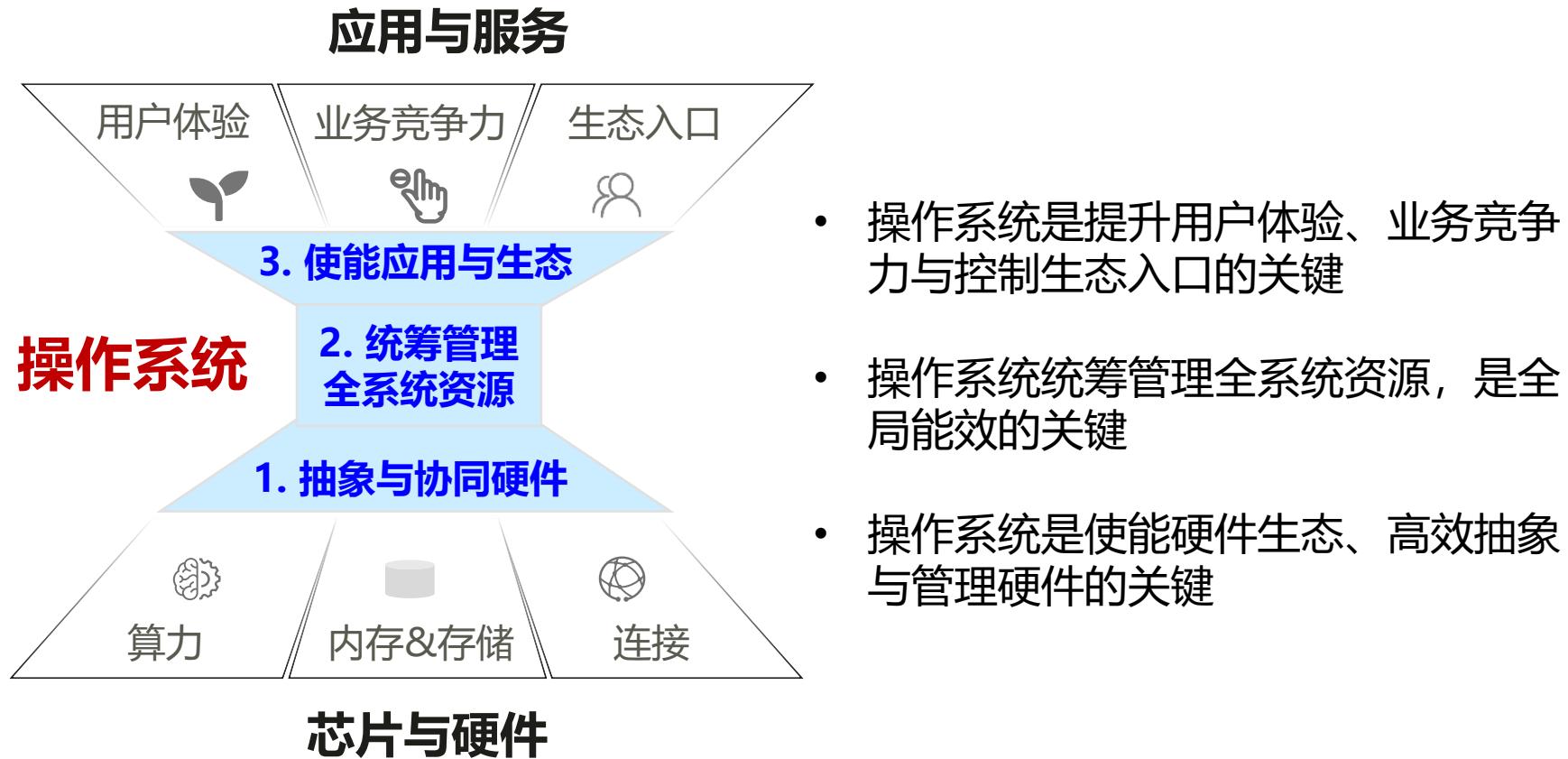
系统服务：语言运行时，算子库，开发库，功能库

目的：控制技术发展

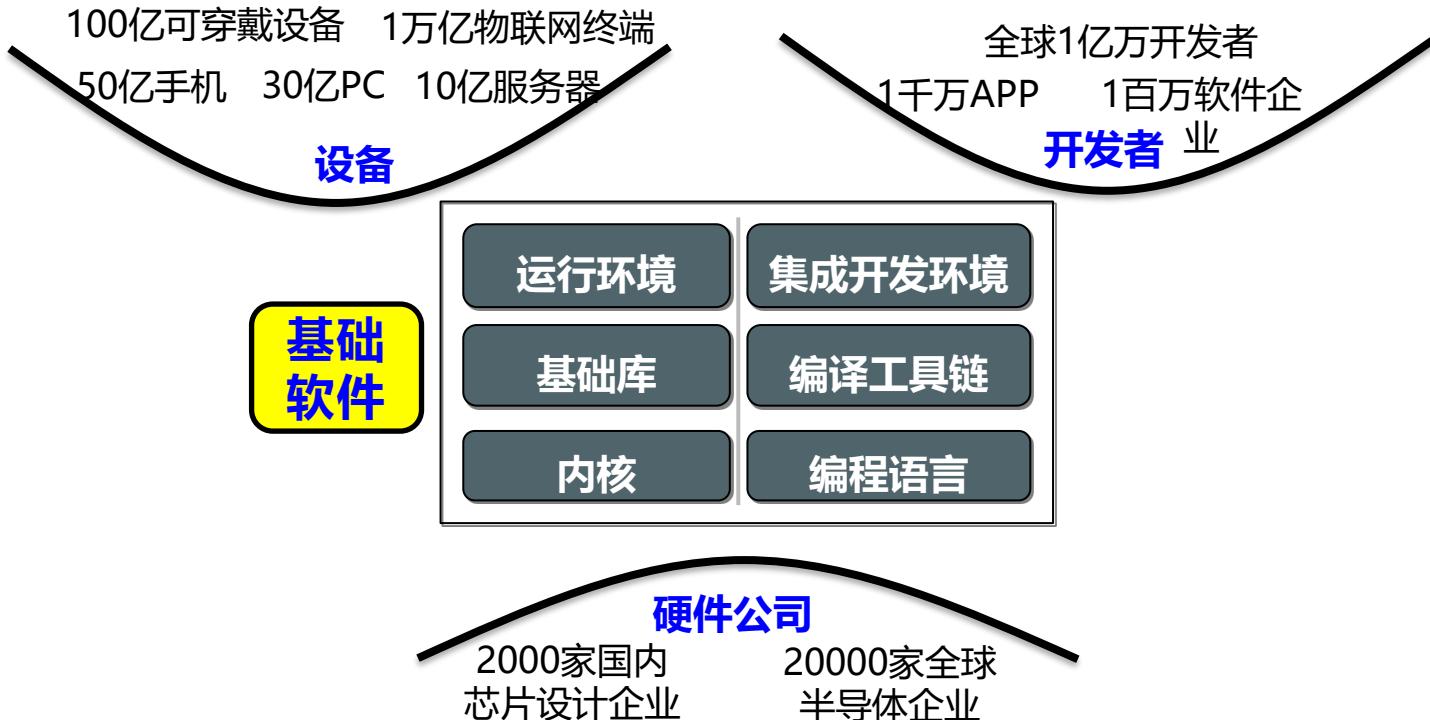
内核子系统：内核和驱动框架

芯片与硬件

操作系统在计算系统中发挥的关键作用



操作系统是IT生态的“粘合剂”和“催化剂”



操作系统是IT产业的核心竞争力和商业价值高地



当前 IT 巨头的核心竞争力
都有基础软件（操作系统）

大纲

- 操作系统的历史
- 什么是操作系统
- 为什么要有操作系统
- **开源：操作系统的主流发展模式**
- RISC-V：操作系统的未来发展方向
- 为什么要学习操作系统

开源理念从何而来

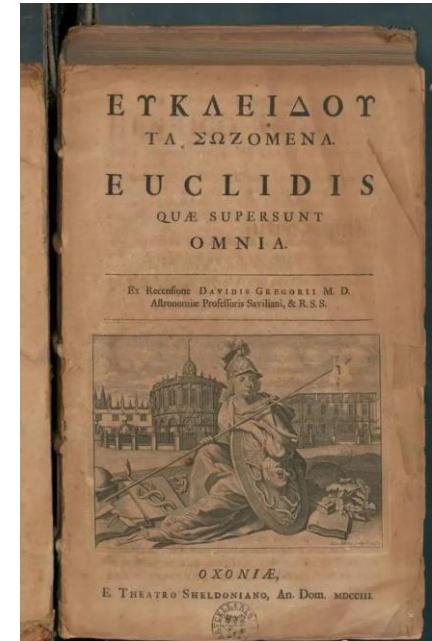
人类科技文明并不总是以阶梯式前进，有时候是波浪式.....



玛雅文明消失



地动仪内部构造已经失传



欧式几何原本得以保留

开源理念从何而来

数字世界出现，人类文明薪火相传有了更便捷的载体



Ken Thompson & Dennis Ritchie
(图灵奖获得者)

UNIX®
00011110 00011110 00011110 00011110 00011110 00011110 00011110

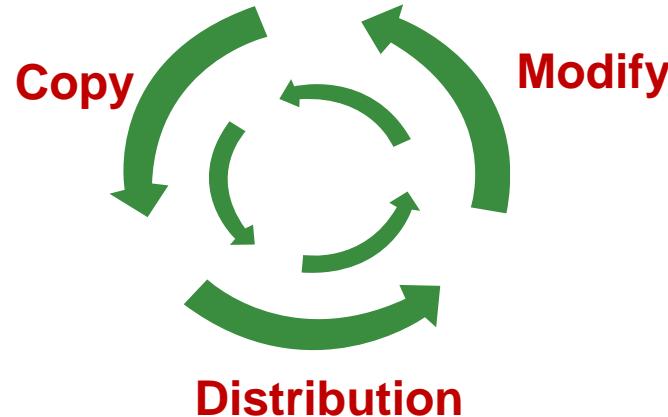
免费分发给学校，开启了开源的规模化实践（1970s）

开源的伟大理念

自由软件和“GNU运动”的出现，让开源成为一种哲学理念和思想体系



Richard M. Stallman



任何人在遵守GPL的前提下，应当具有复制、修改、分发软件源代码的自由

开源的伟大实践

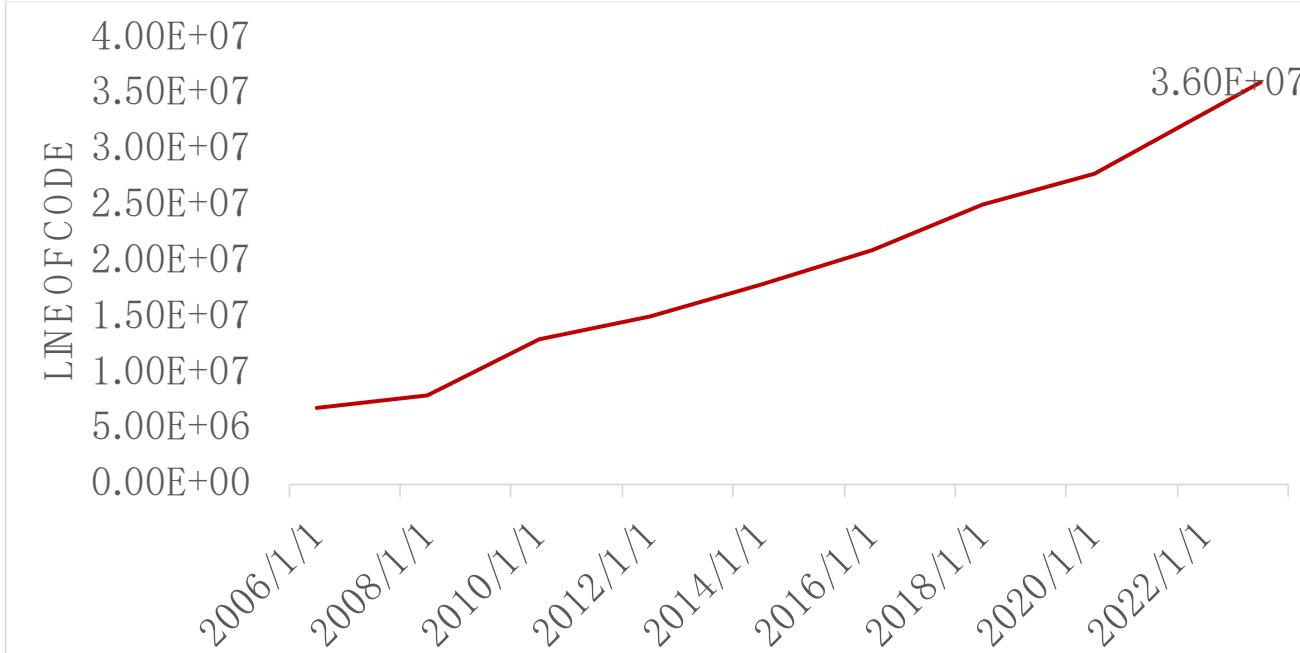
“伟大复兴绝不是轻轻松松、敲锣打鼓就能实现的”，开源的理想也一样



“Talk is cheap, show me the code”

Linus Torvalds 等实干家为开源提供了最核心的基础设施软件

Linux规模



36 million



180亿美元

CAGR: 20%

Linux 代码量

开源协议与开发模式



Linux采用GPL协议

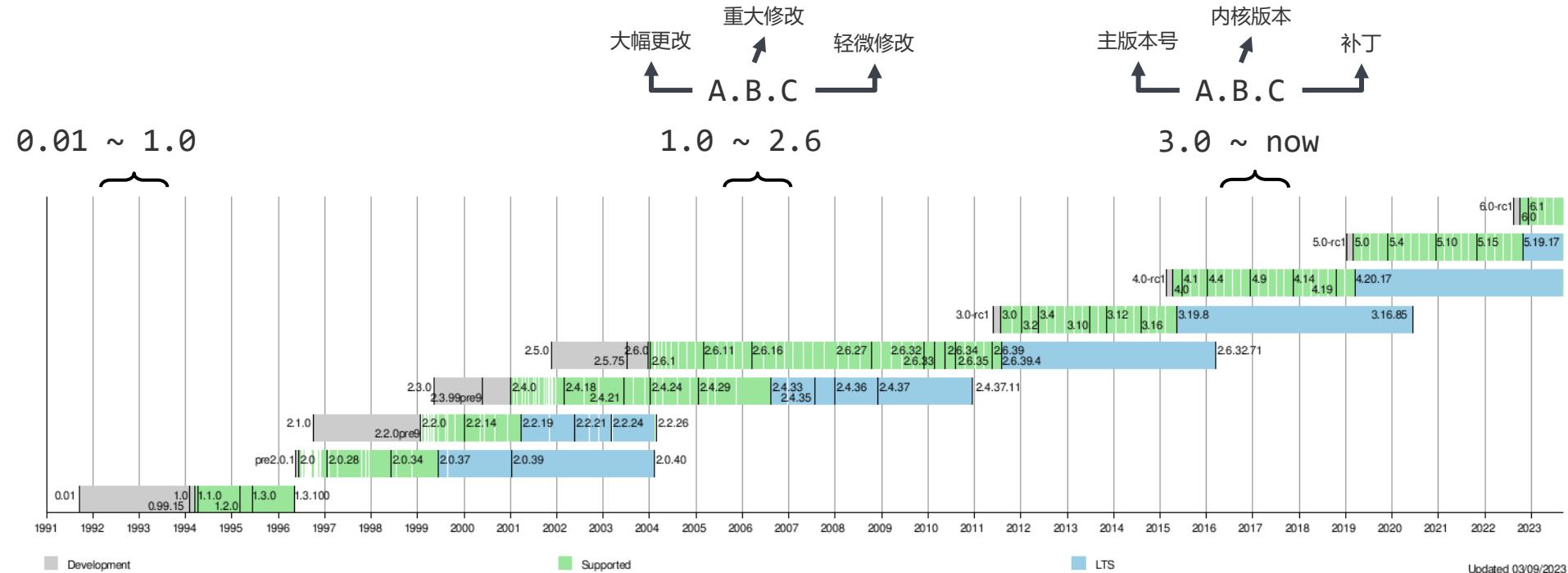
特点：使用开源必须开源

具有极强的传染性，是Linux大规模发展的基础

版本管理



Linux历代版本



Linux无处不在



公共镜像

Aliyun Linux

Aliyun Linux

Windows Server

CentOS

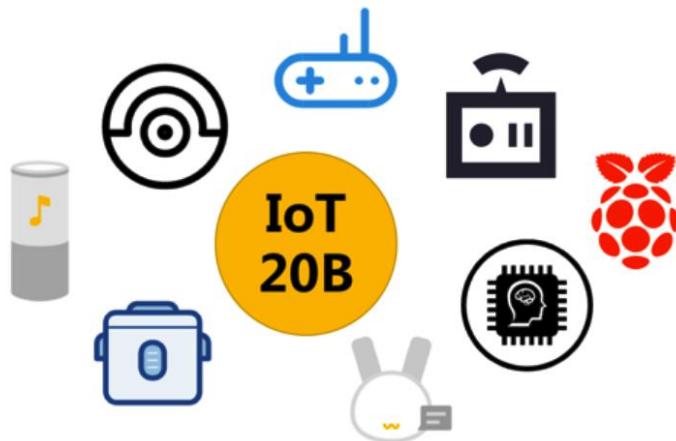
Ubuntu

Red Hat

Debian

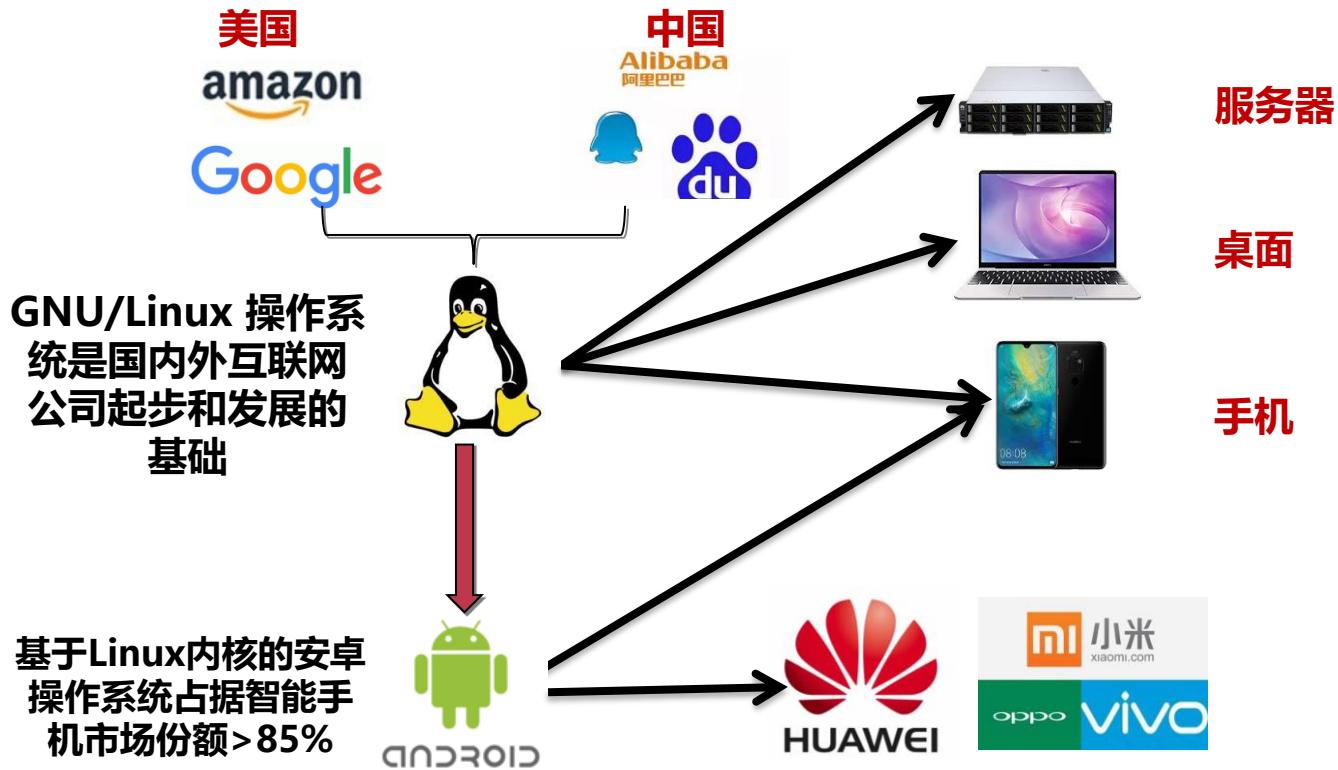
SUSE Linux

OpenSUSE



海量嵌入式设备

开源Linux成就了互联网和智能终端产业



Linux内核开发组成员合照

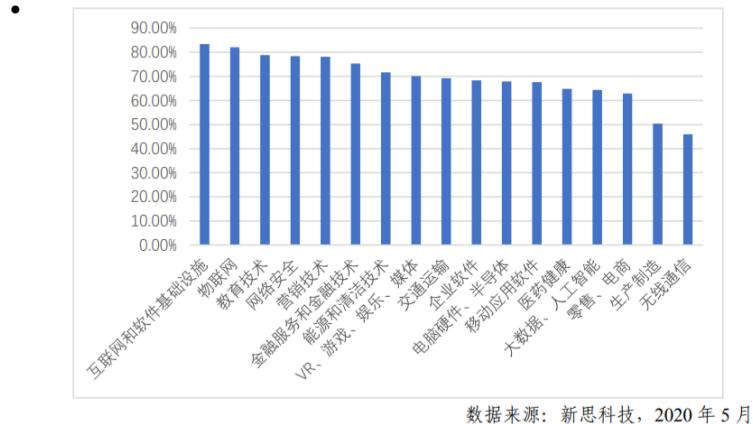


当前，开源软件已经被普遍使用

- 「**97%**的软件包含开源代码，其中计算机硬件和半导体、网络安全、能源和清洁技术、“物联网”设备以及互联网和移动应用软件相关系统中，**100%**都发现了开源代码。」

——新思科技2022年《开源安全与风险分析报告》

- 「**2021年我国使用开源技术的企业占比为88.2%。**」



开源代码应用在各领域，应用比例可超过80%

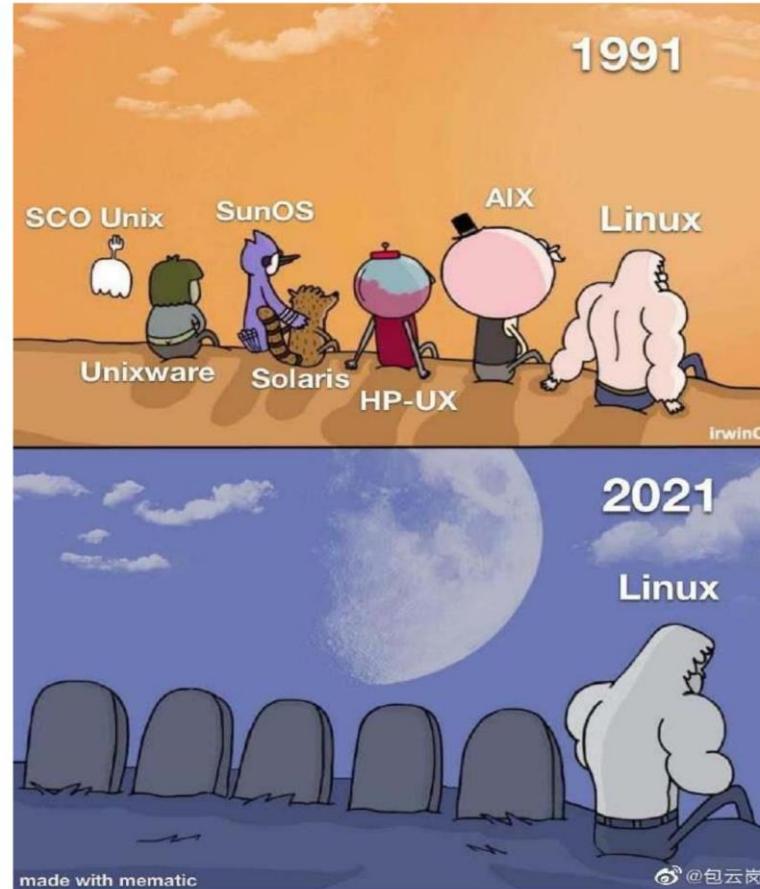
——中国信息通信研究院调查数据



数据来源：中国信息通信研究院，2020年5月

我国企业正在大量应用开源软件

为什么开源不可阻挡？



背后的原因：

- (1) 梅特卡夫定律
- (2) 贝尔定律

未来，绝大多数软件都将来源于开源软件



数据源：“源图”开源软件重大基础设施

开源模式下，操作系统构建的本质是什么？

开源模式下，构建操作系统的本质是对开源软件供应链（涉及上万软件包）的整合优化过程，即选取好用的、优化可用的、补齐缺失的，然后按照依赖关系逐个编译构建。

大纲

- 操作系统的历史
- 什么是操作系统
- 为什么要有操作系统
- 开源：操作系统的主流发展模式
- RISC-V：操作系统的未来发展方向
- 为什么要学习操作系统

什么是指令集？

指令集架构 (Instruction Set Architecture, ISA), 简称指令集，是计算机系统中硬件与软件之间**分界线和交互规范标准**，也是软硬件**生态的起始原点**



指令集的诞生是为了机器兼容和软硬解耦

IBM System/360的四个型号

Model	M30	M40	M50	M65
Datapath width	8 bits	16 bits	32 bits	64 bits
Control store size	4k x 50	4k x 52	2.75k x 85	2.75k x 87
Clock rate (ROM cycle time)	1.3 MHz (750 ns)	1.6 MHz (625 ns)	2 MHz (500 ns)	5 MHz (200 ns)
Memory capacity	8–64 KiB	16–256 KiB	64–512 KiB	128–1,024 KiB
Performance (commercial)	29,000 IPS	75,000 IPS	169,000 IPS	567,000 IPS
Performance (scientific)	10,200 IPS	40,000 IPS	133,000 IPS	563,000 IPS
Price (1964 \$)	\$192,000	\$216,000	\$460,000	\$1,080,000
Price (2018 \$)	\$1,560,000	\$1,760,000	\$3,720,000	\$8,720,000



Wilkes
指令和微程序
(图灵奖获得者)



Brooks
IBM/360 操作系统

One ISA to Rule Them All



指令集是基础软硬件生态的核心规范

通过指令集形成的软硬件联盟，国际巨头们占据了IT生态的价值高地

大型机时代



IBM大型机55年内创造了
5000+亿美元收入

IBM私有指令集
下的封闭体系



PC时代



市值约20000亿美元



X86私有指令集
下的Wintel联
盟



移动终端时代



市值约15000亿美元



ARM私有指令
集下的AA联盟



指令集领域一直缺乏开放的国际标准

开放标准

Field	Standard	Free, Open Impl.	Proprietary Impl.
Networking	Ethernet, TCP/IP	Many	Many
OS	Posix	Linux, FreeBSD	M/S Windows
Compilers	C	gcc, LLVM	Intel icc, ARMcc
Databases	SQL	MySQL, PostgresSQL	Oracle 12C, M/S DB2
Graphics	OpenGL	Mesa3D	M/S DirectX
ISA	???????	-----	x86, ARM, IBM360

RISC-V是什么：使命和愿景

RISC-V is a **free and open** ISA enabling a new era of processor innovation through open standard collaboration.

The RISC-V ISA delivers a new level of free, extensible software and hardware freedom on architecture, paving the way for the **next 50 years** of computing design and innovation.

UC Berkeley: Instruction Sets Want to Be Free

RISC-V 的技术内涵

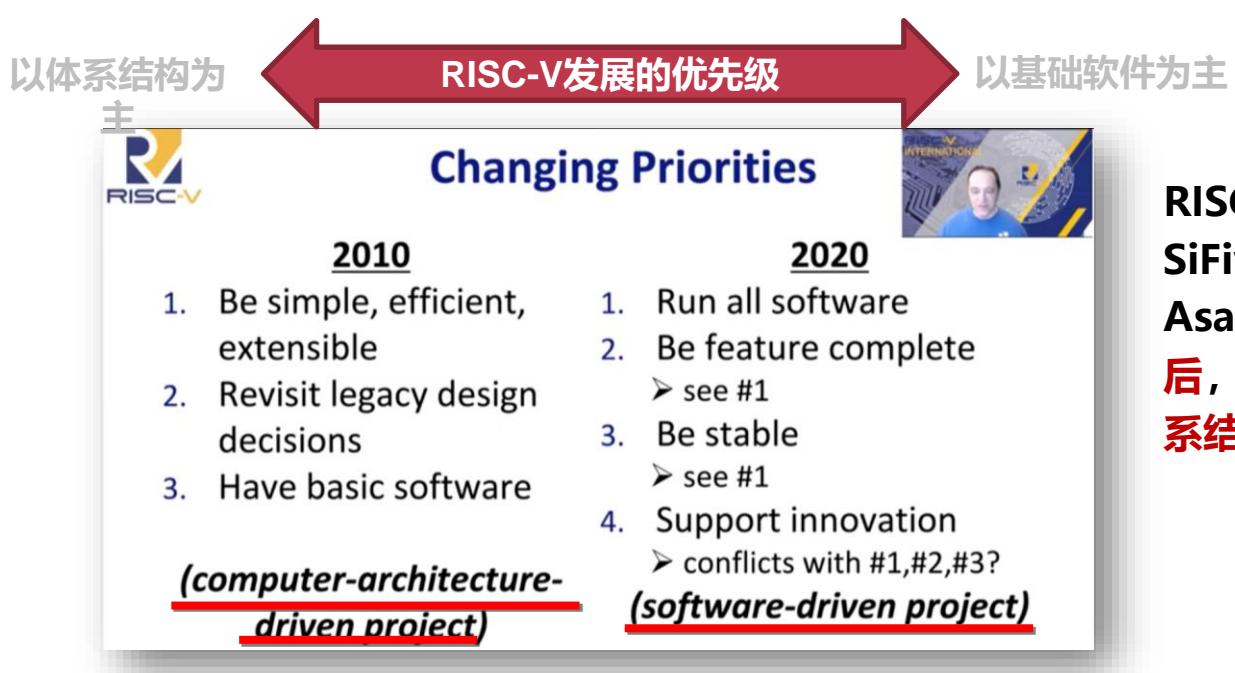
RISC-V指令集 = 基础指令集 + 标准扩展指令集 + 用户自定义扩展指令集

① Base Integer Instructions (32/64/128)				② RV Privileged Instructions (32/64/128)				③ Optional FP Extensions: RV32(F/D/Q) (HP/SP, DP, QP)				④ RISC-V Reference Card						
Category		Name	Fmt	Category		Name	Fmt/Rv mnemonic	Category		Name	Fmt	Category		Name	Fmt			
Loads	Load Byte	I	LB rd,r1,imm	CSR Access	Atomic R/W	R	C3RRW rd,car,r1	Load	I	FLW,D,Q1 rd,r1,imm	Loads	Load Word	CL	C.LW rd',r1',imm				
	Load Halfword	I	LH rd,r1,imm		Atomic Read & Set Bit	R	C3RRS rd,car,r1	Store	S	FLW,D,Q1 rd,r1,imm		Load Word SP	CL	C.LWSP rd',r1',imm				
	Load Byte Unsigned	I	LWU rd,r1,imm		Atomic Read & Clear Bit	R	C3RRC rd,car,r1	Arithmetic	ADD	R FADD,(S D Q) rd,r1,r2		Load Double	CL	C.LD rd',r1',imm				
	Load Half Unsigned	I	LWD rd,r1,imm		Atomic R/W Imm	R	C3RRWI rd,car,r1	Subtract	R FSUB,(S D Q) rd,r1,r2		Load Double SP	CL	C.LWSP rd',r1',imm					
Stores	Store Byte	S	SB rs1,r2,imm		Atomic Read & Set Bit Imm	R	C3RRSI rd,car,r1	Multiply	R FMUL,(S D Q) rd,r1,r2		Load Quad	CL	C.LQ rd',r1',imm					
	Store Halfword	S	SH rs1,r2,imm		Atomic Read & Clear Bit Imm	R	C3RRCI rd,car,r1	Divide	R FDIV,(S D Q) rd,r1,r2		Load Quad SP	CL	C.LQSP rd',r1',imm					
Shifts	Shift Left Immediate	R	SLT,(W D) rd,r1,r2	Change Level	Env. Call	R ECALL	Square Root	R FSQRT,(S D Q) rd,r1		Load Byte Unsigned	R FLD rd',r1',imm		Load Byte	CL	C.LB rd',r1',imm			
	Shift Left Immediate	I	SLLI,(W D) rd,r1,shamt	Environment Breakpoint	R EBREAK							Load Word SP	CL	C.LWSP rd',r1',imm				
	Shift Right Immediate	R	SRG,(W D) rd,r1,shamt	Trap Redirect to Supervisor	R HTSTS							Load Double	CL	C.LD rd',r1',imm				
	Shift Right Immediate	I	SLR,(W D) rd,r1,shamt	Redirect Trap to Hypervisor	R NTH							Load Double SP	CL	C.LWSP rd',r1',imm				
	Shift Right Arithmetic	R	SGR,(W D) rd,r1,r2	Hypervisor Trap to Supervisor	R HTST							Load Quad	CL	C.LQ rd',r1',imm				
	Shift Right Imm 3	R	SRAT,(W D) rd,r1,shamt	Interrupt Wait for Interrupt	R MFENCE							Load Quad SP	CL	C.LQSP rd',r1',imm				
Arithmetic	ADD	R ADD,(W D) rd,r1,r2	MMU	R SFENCE,VN r1								Load Word Unsigned	R FLD rd',r1',imm		Load Word	CL	C.LW rd',r1',imm	
ADD Immediate	I ADDI,(W D) rd,r1,imm											Float Load Double	R FLDL rd',r1',imm		Float Load	CL	C.FLM rd',r1',imm	
Subtract	R SUB,(W D) rd,r1,r2											Float Load Double SP	CL	C.FLDL rd',r1',imm		Float Load	CL	C.FLMSP rd',r1',imm
Load Upper Imm	I LUI rd,imm											Float Load Double	R FLDLSP rd',r1',imm		Float Load	CL	C.FLMSP rd',r1',imm	
Add Upper Imm to PC	R ADDPC rd,imm											Float Load Double SP	CL	C.FLDLSP rd',r1',imm		Float Load	CL	C.FLMSP rd',r1',imm
Logical	XOR	R XOR rd,r1,r2										Float Load Double	R FLDLSP rd',r1',imm		Float Load	CL	C.FLMSP rd',r1',imm	
XOR Immediate	I XORI rd,r1,imm											Float Load Double SP	CL	C.FLDLSP rd',r1',imm		Float Load	CL	C.FLMSP rd',r1',imm
OR	R OR rd,r1,r2											Float Store Word	R FSW rd',r1',imm		Float Store	CL	C.FSW rd',r1',imm	
OR Immediate	I ORI rd,r1,imm											Float Store Word SP	CL	C.FSWSP rd',r1',imm		Float Store	CL	C.FSWSP rd',r1',imm
AND	R AND rd,r1,r2											Float Store Word	R FSWL rd',r1',imm		Float Store	CL	C.FSWL rd',r1',imm	
AND Immediate	I ANDI rd,r1,imm											Float Store Word SP	CL	C.FSWLSP rd',r1',imm		Float Store	CL	C.FSWLSP rd',r1',imm
Compare	Set <	R SLT rd,r1,r2										Convert	R FCVT,(S D Q) rd,r1,r2		Convert	CL	C.FCVT rd',r1',imm	
	Set < Immediate	I SLLI rd,r1,imm										Convert	R FCVT,(S D Q) rd,r1,r2		Convert	CL	C.FCVT rd',r1',imm	
	Set < Unsigned	R STLT rd,r1,r2										Convert	R FCVT,(S D Q) rd,r1,r2		Convert	CL	C.FCVT rd',r1',imm	
	Set < Imm Unsigned	I SLLUI rd,r1,imm										Convert	R FCVT,(S D Q) rd,r1,r2		Convert	CL	C.FCVT rd',r1',imm	
Branches	Branch =	SB BEQ rd,r1,imm										Convert	R FCVT,(S D Q) rd,r1,r2		Convert	CL	C.FCVT rd',r1',imm	
	Branch =	SBEQ rd,r1,imm										Convert	R FCVT,(S D Q) rd,r1,r2		Convert	CL	C.FCVT rd',r1',imm	
	Branch <	SB BNE rd,r1,imm										Convert	R FCVT,(S D Q) rd,r1,r2		Convert	CL	C.FCVT rd',r1',imm	
	Branch <	SBNNE rd,r1,imm										Convert	R FCVT,(S D Q) rd,r1,r2		Convert	CL	C.FCVT rd',r1',imm	
	Branch <	SB BLT rd,r1,imm										Convert	R FCVT,(S D Q) rd,r1,r2		Convert	CL	C.FCVT rd',r1',imm	
	Branch >	SB BGT rd,r1,imm										Convert	R FCVT,(S D Q) rd,r1,r2		Convert	CL	C.FCVT rd',r1',imm	
	Branch > Unsigned	SB BGEU rd,r1,imm										Convert	R FCVT,(S D Q) rd,r1,r2		Convert	CL	C.FCVT rd',r1',imm	
	Branch > Unsigned	SB BGEQU rd,r1,imm										Convert	R FCVT,(S D Q) rd,r1,r2		Convert	CL	C.FCVT rd',r1',imm	
Jump & Link	JAL	I JAL rd,imm										Swap	R FSW rd',r1',imm		Swap	CL	C.FSW rd',r1',imm	
	Jump & Link Register	I JALR rd,c1,imm										Swap	R FSWL rd',r1',imm		Swap	CL	C.FSWL rd',r1',imm	
Synch	Synch thread	I FENCE rd,imm										Swap Status Reg	R FCSR rd,r1		Swap Status Reg	CL	C.FCSR rd,r1	
	Synch Instr & Data	I FENCE..1										Swap Roundings	R FSW rd',r1',imm		Swap Roundings	CL	C.FSW rd',r1',imm	
System	System CALL	I SCALL rd,imm										Swap	R FSWL rd',r1',imm		Swap	CL	C.FSWL rd',r1',imm	
	System BREAK	I SBREAK rd,imm										Swap	R FSW rd',r1',imm		Swap	CL	C.FSW rd',r1',imm	
Counters	Read CYCLE	I RDUCYCLE rd										Swap	R FSWL rd',r1',imm		Swap	CL	C.FSWL rd',r1',imm	
	Read CYCLE upper Half	I RDUCYCLER rd										Swap	R FSW rd',r1',imm		Swap	CL	C.FSW rd',r1',imm	
	Read TIME	I RDUTIME rd										Swap	R FSWL rd',r1',imm		Swap	CL	C.FSWL rd',r1',imm	
	Read INSTR	I RDINST rd										Swap	R FSW rd',r1',imm		Swap	CL	C.FSW rd',r1',imm	
	Read INSTR upper Half	I RDINSTRET rd										Swap	R FSWL rd',r1',imm		Swap	CL	C.FSWL rd',r1',imm	
	Read INSTR upper Half	I RDINSTRETH rd										Swap	R FSW rd',r1',imm		Swap	CL	C.FSW rd',r1',imm	
16-bit (RVC) and 32-bit Instruction Formats																		

指令集规范、商标等权利由
RISC-V国际协会（RVI）维护

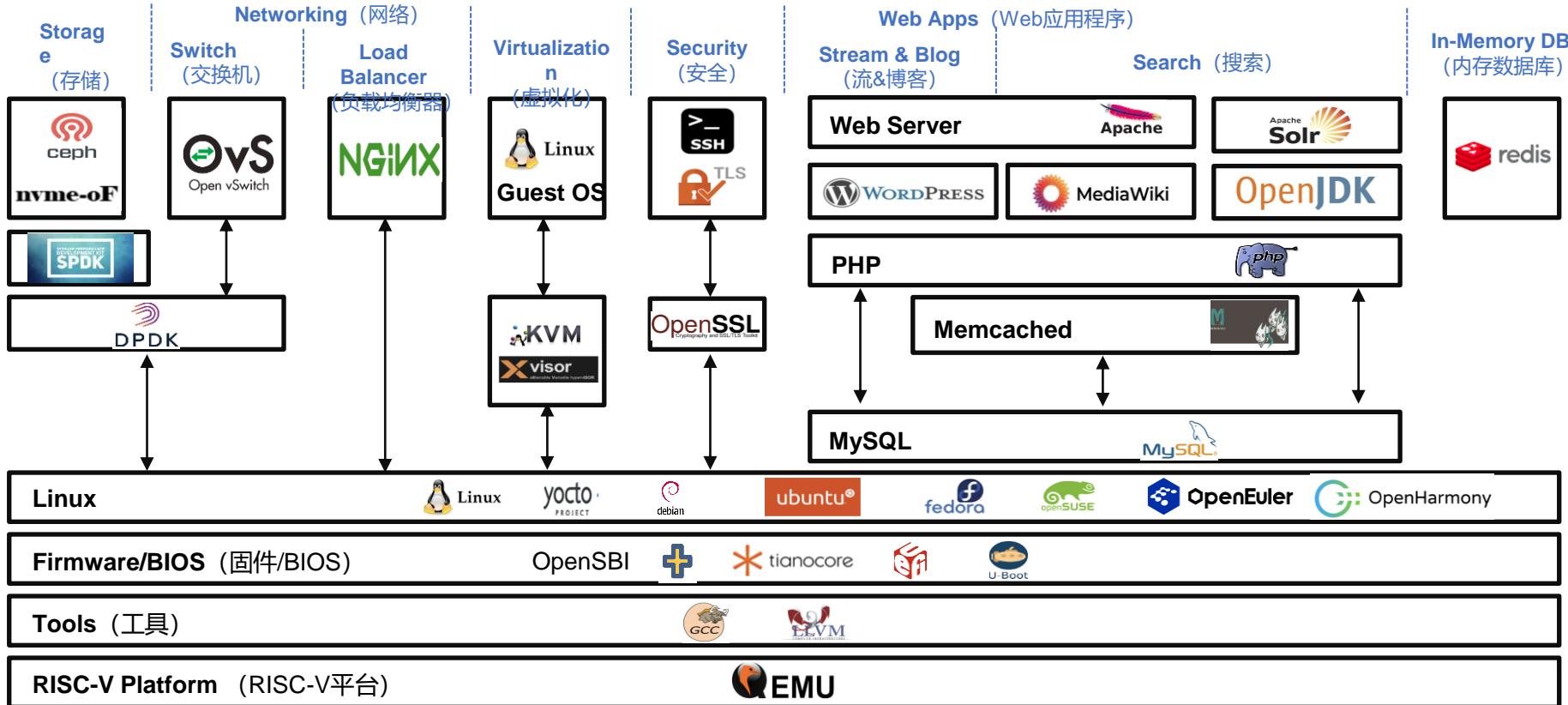


软件是当前 RISC-V 生态的更高优先级



RISC-V的第一发明者，美国 SiFive公司联合创始人 Krste Asanovic 认为，从2020年之后，RISC-V发展的优先级从体系结构驱动切换为软件驱动

RISC-V 生态的软件栈



来自: <https://open-src-soc.org/2022-05/media/slides/RISC-V-International-Day-2022-05-05-11h05-Calista-Redmond.pdf>, 略有修改

RISC-V是众多软件学科方向的一次“复兴”机遇



*来自RVI官方网站
站

RISC-V是众多软件学科方向的一次“复兴”机 遇 cont.

- 基础软件领域已经很久没有这么多轮子可以重新发明了

- ❖ 各种编程语言的编译器 $\times N$
- ❖ 语言虚拟机和运行时 $\times N$
- ❖ 系统虚拟机、指令集模拟器 $\times N$
- ❖ 全系统性能分析和优化、软硬件协同演进的定制和优化
- ❖ 二进制工具 $\times N$
- ❖ 全新的操作系统视角和可能性

对于我国基础软件领域，
RISC-V带来的“造轮子”机
会就是“补短板”的机遇

大纲

- 操作系统的历史
- 什么是操作系统
- 为什么要有操作系统
- 开源：操作系统的主流发展模式
- RISC-V：操作系统的未来发展方向
- 为什么要学习操作系统

为什么学习操作系统？

- **操作系统是一个成熟的领域**
 - Windows曾经一统桌面数十年
 - Linux在各种设备都能运行
 - 让用户改变操作系统是很困难的
 - 我们是否需要新的操作系统？
- **“新的”操作系统依然不断出现**
 - Linux、Mac OS、Android、iOS、ROS...
 - 大疆用什么操作系统？谷歌数据中心用什么操作系统？
火星车用什么操作系统？...

为什么学习操作系统？

- **操作系统是系统领域的基石**
- **系统领域有大量的公司**
 - 微软、谷歌、IBM、EMC、VMware、华为、阿里...
 - 谷歌的核心技术
 - 集群、GFS、MapReduce、BigTable...
 - 都是系统领域的优秀工作
- **学好操作系统，for fun and profit**
 - 优秀的公司，优秀的大学，更多的机会

图灵奖与操作系统的演变



Maurice Wilkes
1967年图灵奖



Frederick Brooks
1999年图灵奖



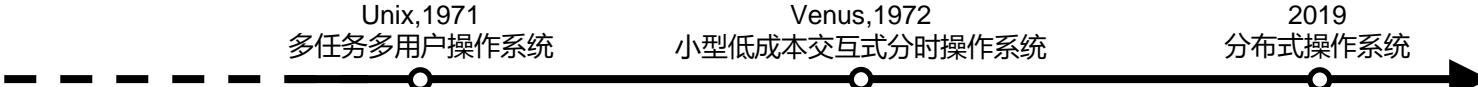
Fernando J. Corbató
1990年图灵奖



EDSAC, 1949
Multi-programming
第一台存储程序式电子计算机

IBM System/360, 1964

CTSS, 1961 & Multics, 1969
分时操作系统



Ken Thompson & Dennis Ritchie
1983年图灵奖

Venus, 1972
小型低成本交互式分时操作系统



Barbara Liskov
2008年图灵奖

2019
分布式操作系统



这门课程希望带给大家的能力

- 成为一个更高效的程序员
 - 更快找到并消灭bug的能力
 - 理解并调试程序性能的能力
- 理解复杂系统设计与实现的能力
- Linux系统开发能力

课程的特点：抽象与具体

- **许多课程强调抽象**
 - 如：抽象数据类型、渐进分析等
- **这些抽象通常会带来不可避免的限制**
 - 需要理解底层的实现细节，才能突破限制

“What I cannot create, I do not understand”

—— Richard Feynman (CIT)

为什么操作系统比较难/有意思？

- 深入事情本质：直接管理硬件细节
 - 好处：实现资源的高效利用，从根本上解决问题，做一个高效程序员（降维）
 - 挑战：需要理解与处理硬件细节，硬件甚至可能出错
 - “**把简单带给用户、把复杂留给自己**
 - 对比：用户态写一个Hello World和在操作系统内核中输出一个Hello World
- 锻炼系统架构能力
 - 将复杂问题进行抽象与化简
 - 计算机科学中**30%**的原则是从操作系统中来的 (13/41, <http://greatprinciples.org>)

操作系统研究

操作系统研究三大驱动力

- **应用模式**: 人工智能、智能驾驶、云计算等
- **硬件演进**: 持久性内存、GPU、智能硬件、AI芯片、硬件Enclave等
- **技术驱动**: 新指令集 (RISC-V) 、新架构 (多内核、分布式内核等) 、新方法 (AI、形式化等)

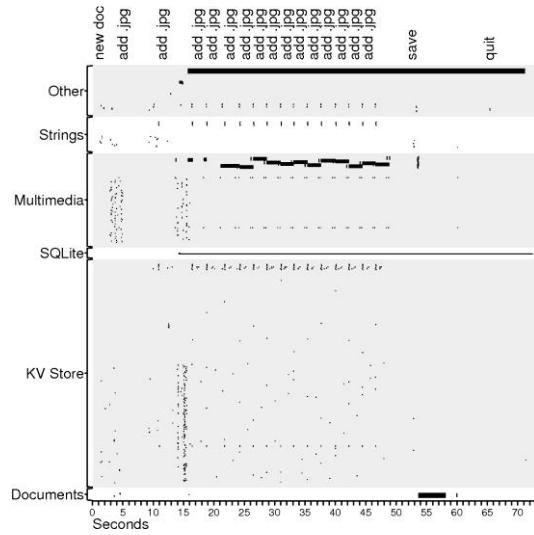
三个核心问题

- 如何为上层应用提供更快、更安全、更易用的接口?
- 如何统筹全局资源提供最优性能/体验?
- 如何为底层硬件建立高效、安全、高利用率的抽象?

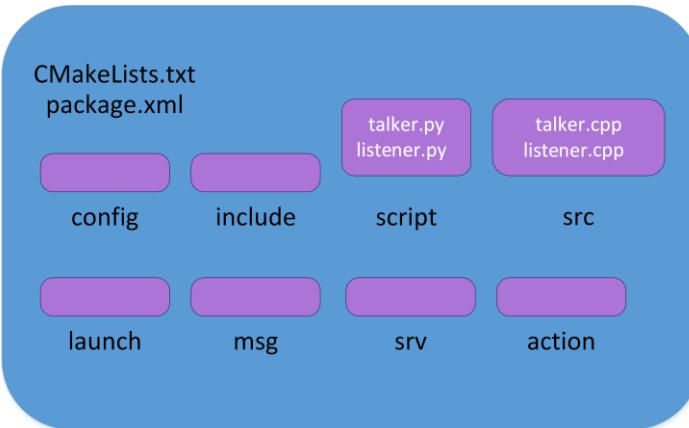
新应用模式的变化

应用模式变化的例子

文件抽象



应用容器



云原生

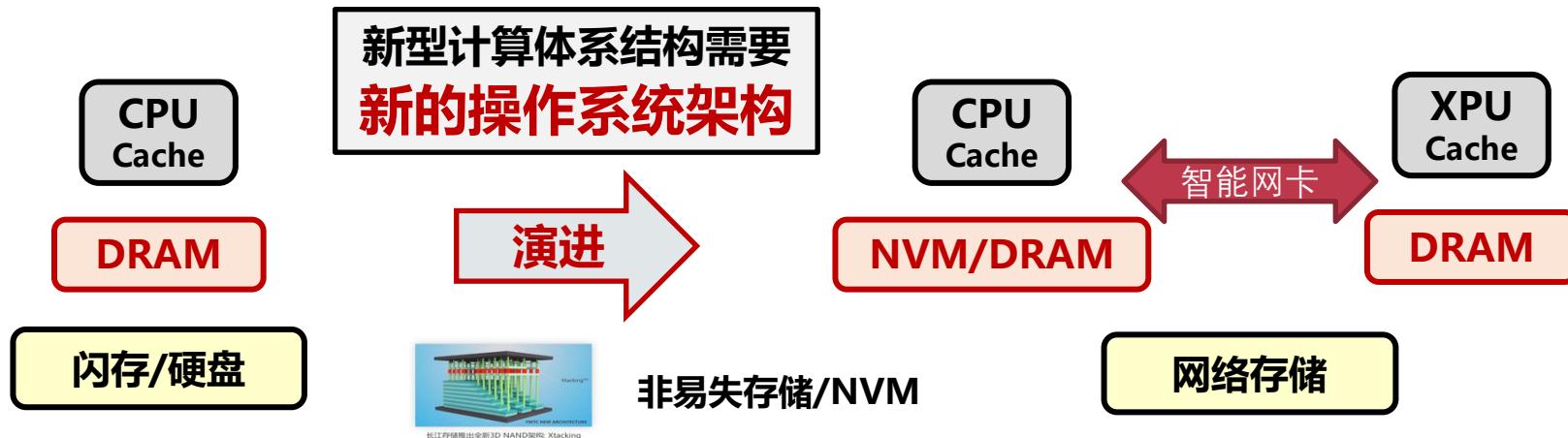


计算体系统结构的演进

新型大容量、非易失内存的出现将替代传统硬盘

从**三层**（硬盘、内存、CPU）变为主要**两层**（内存、XPU）

异构加速硬件繁荣使得CPU-centric计算到分布式协作计算



新技术的驱动

新架构，例如



分布式OS架构

新方法，例如



AI



形式化验证