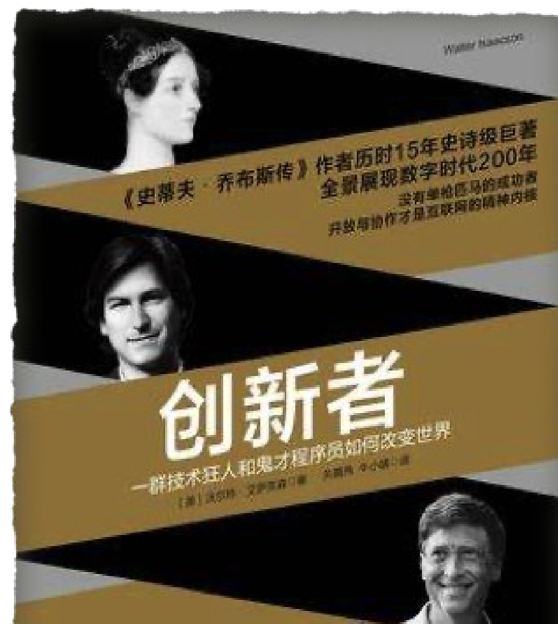
OS Introduction

Outline

- Operating Systems
  - History
  - Three Easy Pieces:
    - Processes （CPU/ 计算资源虚拟化）
    - Virtual Memory （内存虚拟化）
    - Files （外存虚拟化 / 持久化）

OS History

- 起步阶段
  - 硬件的附属和扩展
  - 工具集 =>OS
- 第一个软件
- 最复杂的软件(之一)



平台的演进:

- < 硬件 > - 软件
- 硬件 - < 系统软件 > - 应用软件
- < 算力, 数据 > - < 系统软件 > - < 算法, 模型 > - Agent

Cloud

OS History

- 计算机硬件的发展
  - **机械式计算机**， Charles Babbage (1792-1871) 的“分析机”（设计，具备现代计算机很多特征）
  - **机电式计算机**， Babbage- 二次世界大战
    - 第一台可工作的数字计算机， 300 个真空管
    - Zuse 的 Z3 计算机
    - Howard Aiken 在哈佛建造了 Mark I
  - **电子计算机**， ENIAC， 3000 个真空管
    - 宾夕法尼亚大学的 William Mauchley 和他的学生 J. Presper Eckert
    - 操作方式：
      - 插线板控制计算机（上千根电缆）
      - 机器语言（穿孔卡片）

OS History

- 第二代电子计算机（1955-1964）
  - 晶体管 + 批处理系统
  - 批处理:
    - Motivation 是减少机时的浪费，任务不间断执行
    - 便宜的计算机执行 I/O （IBM 1401 把全部作业从穿孔卡带读出，写入磁带)
    - 收集一小时的批量作业，将磁带连到高端计算机执行计算任务（IBM 7094）
    - 程序员和运维人员分开
  - 这个阶段的操作系统：Just Libaries
    - 例如对处理 I/O 的代码进行封装，提供 API

OS History

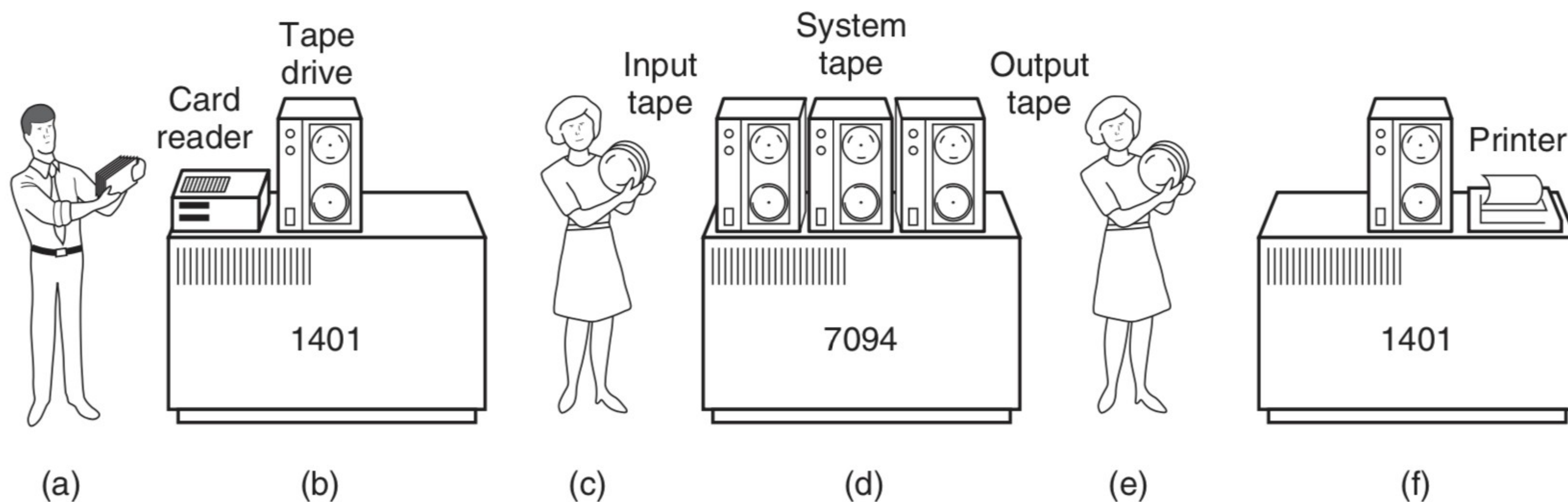


Figure 1-3. An early batch system. (a) Programmers bring cards to 1401. (b) 1401 reads batch of jobs onto tape. (c) Operator carries input tape to 7094. (d) 7094 does computing. (e) Operator carries output tape to 1401. (f) 1401 prints output.

批处理系统在处理复杂任务（例如 :I/O 和计算无规律交替进行）时不够高效

OS History

- 第三代电子计算机（1965-1980）
  - 集成电路和多道程序设计
  - 背景：兼容机（IBM 360 系列）
  - 多道程序设计（multiprogramming）
    - Motivation：尽可能让 CPU 利用率 100%
    - 同时有多个作业运行，每个作业占一部分内存
    - 当某个作业等待 I/O 时，其他作业可以利用 CPU 进行计算
  - 分时系统
    - 本质还是批处理系统，程序提交作业到返回要等很久
    - 程序员希望像早期计算机一样独占 => 分时系统 (time sharing)
    - 每个用户一个通道，按照小时间片轮流使用计算机

OS History

- 第三代电子计算机（1965-1980）
  - 操作系统：在 Library 之外，提供保护
  - System call 概念（Atlas 系统，1962/1978）
    - 与 procedure call 的主要区别是同时提升了硬件的特权级别（CPU 支撑不同级别权限，如 Ring0~3）
    - 有了用户态、内核态的区别，权限不同
    - System call 通过特殊的 trap 指令触发（软中断）

OS History

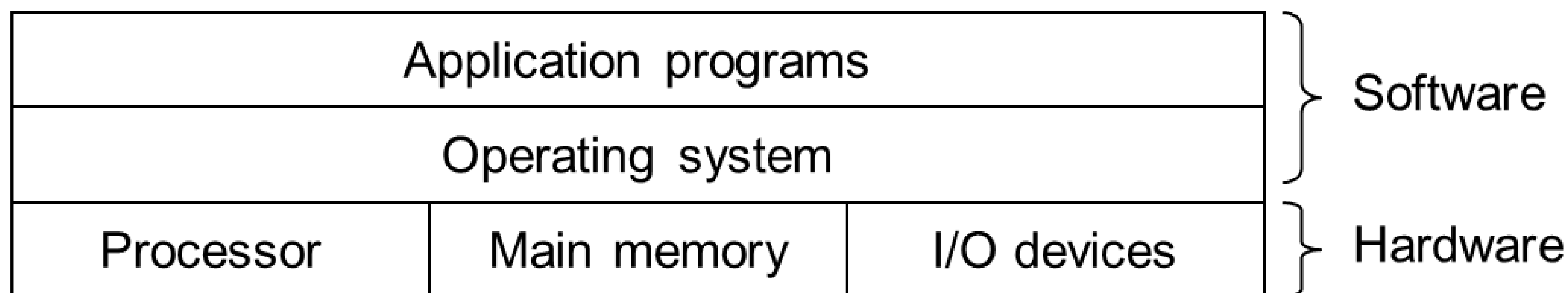
- 第三代电子计算机（1965-1980）
  - MULTICS 操作系统
    - 一台机器满足波士顿地区所有用户需求（野心过大）
    - 贝尔实验室 -> 通用电气公司 ->MIT，在 80 多个公司和大学中部署，用户量小，但使用了 30 年
    - 提出了很多原创概念，包括计算服务 -> 云计算
    - 对 UNIX 及其衍生系统（FreeBSD, Linux, iOS, Andoid）有巨大影响
  - Unix
    - 1969，基于 DEC PDP-7 设计
    - 贝尔实验室， Ken Thompson, Dennis Ritchie 等（1983 年图灵奖）
    - Rewritten in C in 1973, announced in 1974 (开始是用汇编写的)

OS History

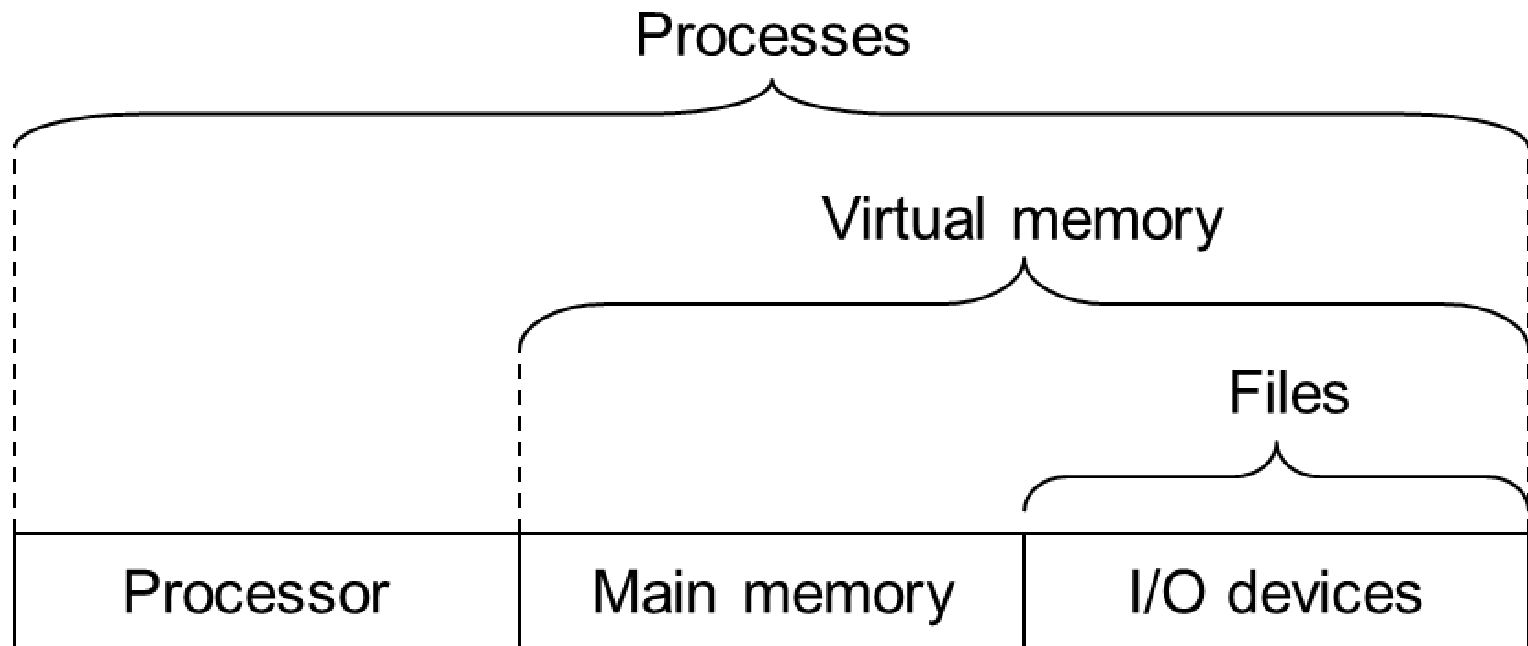
- 第四代电子计算机（1980 至今）
  - 个人计算机
    - Mac, IBM, Wintel, Linux
    - IBM PC (<- MS DOS, Intel CPU)
      - DOS <- Seattle Computer Products (7.5 万美元)
      - MS-DOS <- DOS (雇佣 DOS 作者 Tim Paterson)
    - 图形界面操作系统
      - 施乐 ->Apple->Windows
      - 1985-1995， Windows 运行在 MS-DOS 上（类似 shell）
      - Windows 95, Windows 98 (使用了大量 Intel 汇编)
      - Windows NT (NT 5.0 -> Windows 2000, **Windows XP**)
      - Windows Server 2003, 2008 (服务端)
      - Vista, **Windows 7**, Windows 10 (个人)

Operating Systems

- 应用软件不能直接访问硬件资源
- 它们必须依赖操作系统提供的服务来访问硬件
- 操作系统对硬件的封装就是虚拟化
  - OS 创造了进程、虚存、文件等抽象概念



Abstractions Provided by OS



抽象能力

- 合适的抽象很重要
  - 对于外存硬件，抽象为块设备
    - 设备号、地址、读写、范围大小
  - 更高级抽象：文件
    - 路径 / 文件名、偏移量、读写、大小，元数据操作
  - 硬件是丑陋的（复杂的），程序员需要软件抽象
  - 抽象原则：
    - 方便使用
    - 模型 / 概念简单
    - 共性多，覆盖面广（文件覆盖文本、音视频、指执行文件等多种类型）

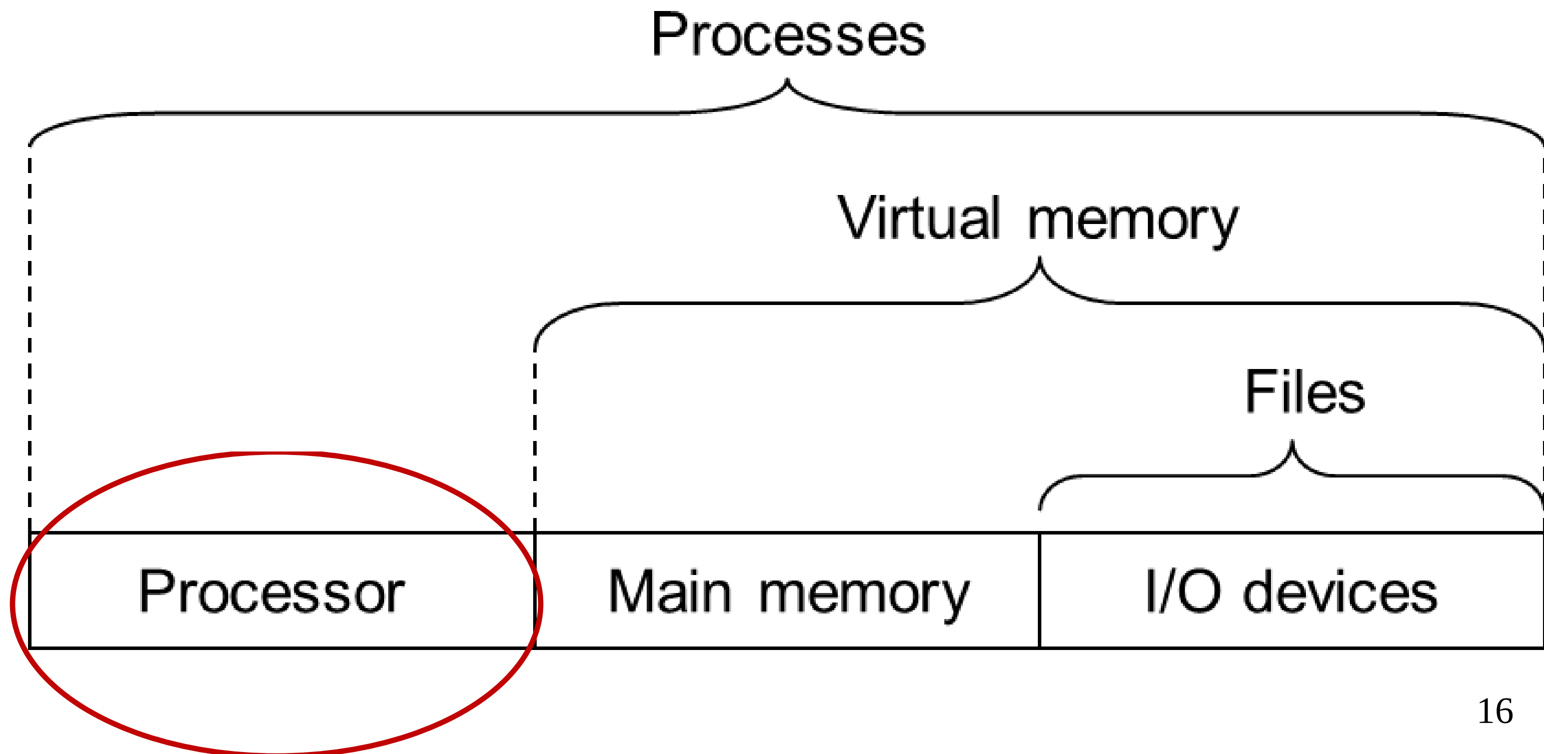
Operating System

- 是一组管理计算机硬件的程序
- 也提供一些基本的支撑给应用程序
- 作为计算机用户和计算机硬件之间的中间件

OS 的两个主要作用

- 保护硬件免受应用程序的错误使用
  - 应用程序太多，程序员水平参差不齐，很容易错误使用硬件；
  - E.g.，定制 open channel SSD，应用层自己写 FTL，每个 cell 最多 7000 次擦除
- 给应用程序提供简单、统一的操作硬件设备的机制

Abstractions Provided by OS



Processes

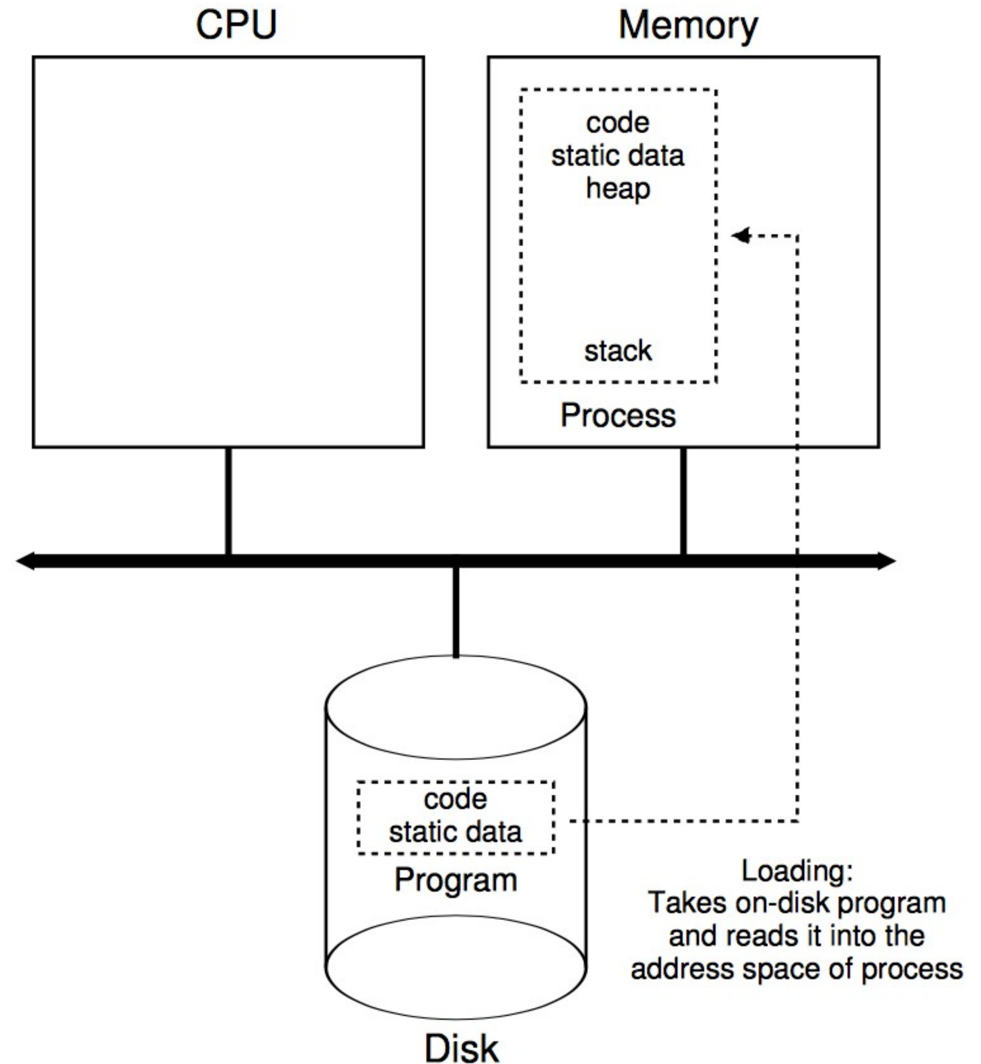
- 定义 : A *process* is an instance of a running program.

- 进程是整个计算机科学领域最成功的、影响深远的概念之一

- 系统中的每个程序都运行在某个进程的上下文中

Processes

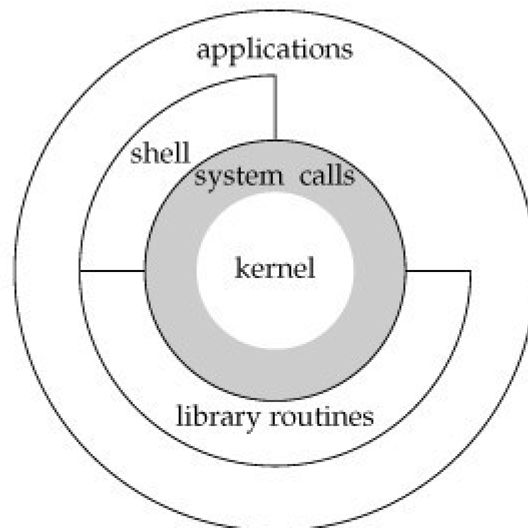
- Running program



When a New Process is Created

- 用户运行一个程序的方法：
  - 在 shell 程序中输入可执行文件的路径和回车
  - Shell 程序会创建一个新的进程
  - 然后在这个新进程的上下文中，运行这个可执行文件

Figure 1.1. Architecture of the UNIX operating system



- **Application**
- **Shell/ library**
- **System calls**

When a New Process is Created

进程也可以通过其他方式启动和运行

- 在图形界面中打开应用程序的进程
- 通过代码创建一个进程
- …

Process

- 当一个程序运行在现代计算机系统上时，
  - 操作系统给程序提供的错觉
    - 这个程序感觉自己是唯一运行在系统中的程序（隔离性）
  - 延伸：
    - 属于“功能隔离”，不是“性能隔离”
      - 进程越多，运算和 IO 越慢
    - 性能隔离：云计算产品必备
      - 达到承诺给用户的 SLO/SLA，例如 10MB/s

Process

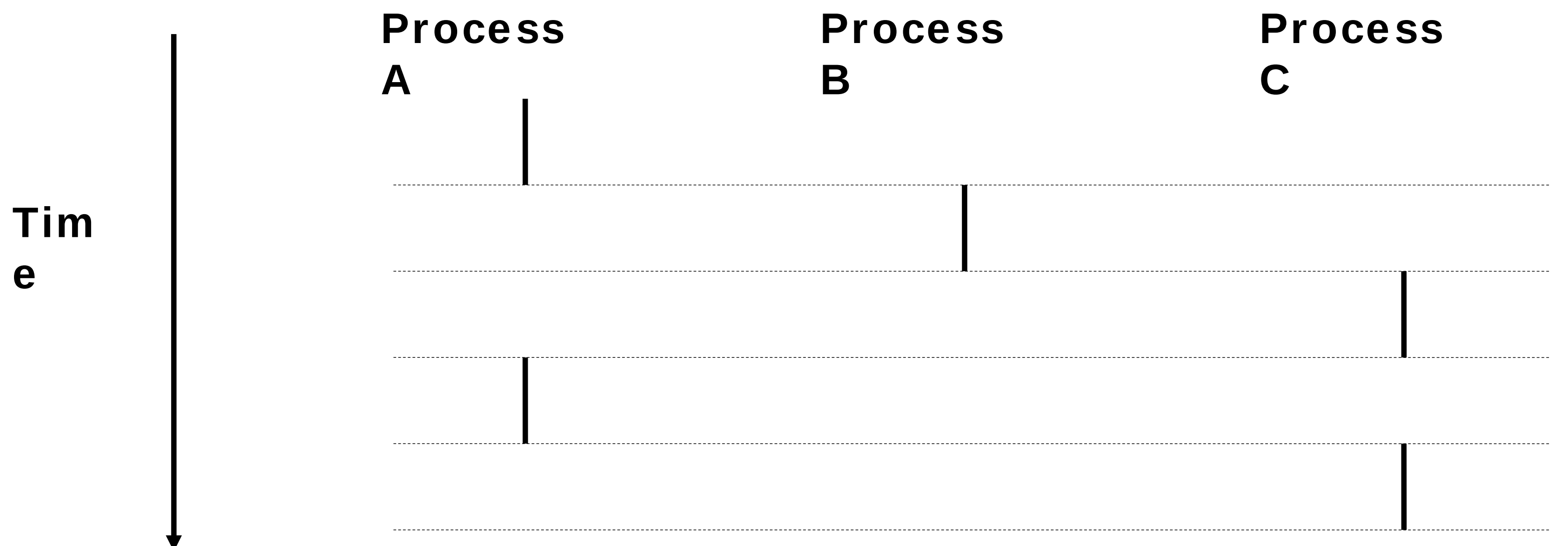
- 进程给每个程序提供两个关键的抽象：
  - 逻辑控制流
    - 让每个进程感到自己是独自使用 CPU 的
  - 私有的地址空间
    - 让每个进程感到自己是独自使用整个内存的

Process

- 如何维护进程独享系统资源的错觉?
  - 进程的指令交替执行（多任务）
  - 由虚存系统管理地址空间，每个进程有自己独立的地址空间

Logical control flows

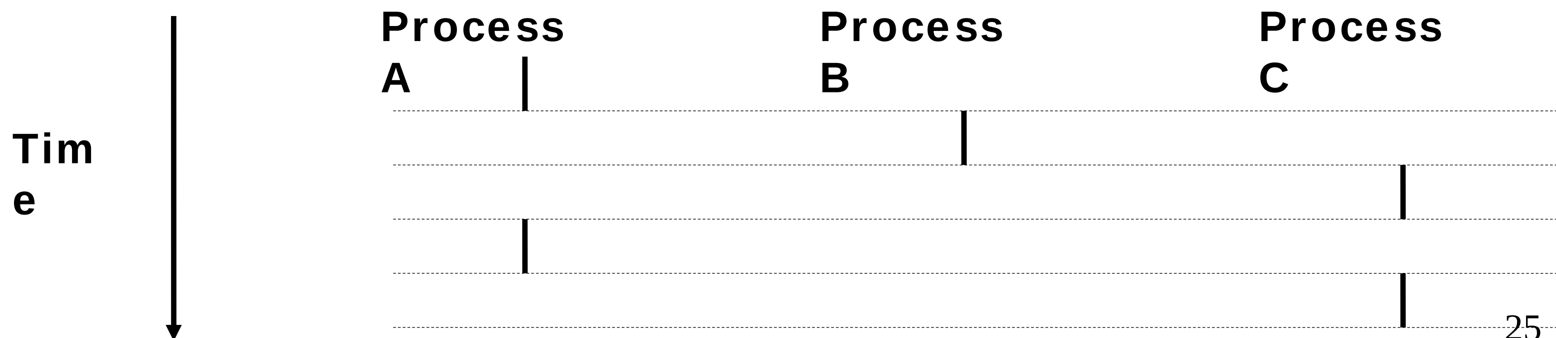
- 每个进程有自己的逻辑控制流
  - 不会影响其他任何进程的状态
- 抢占
- 时间片



Concurrent processes

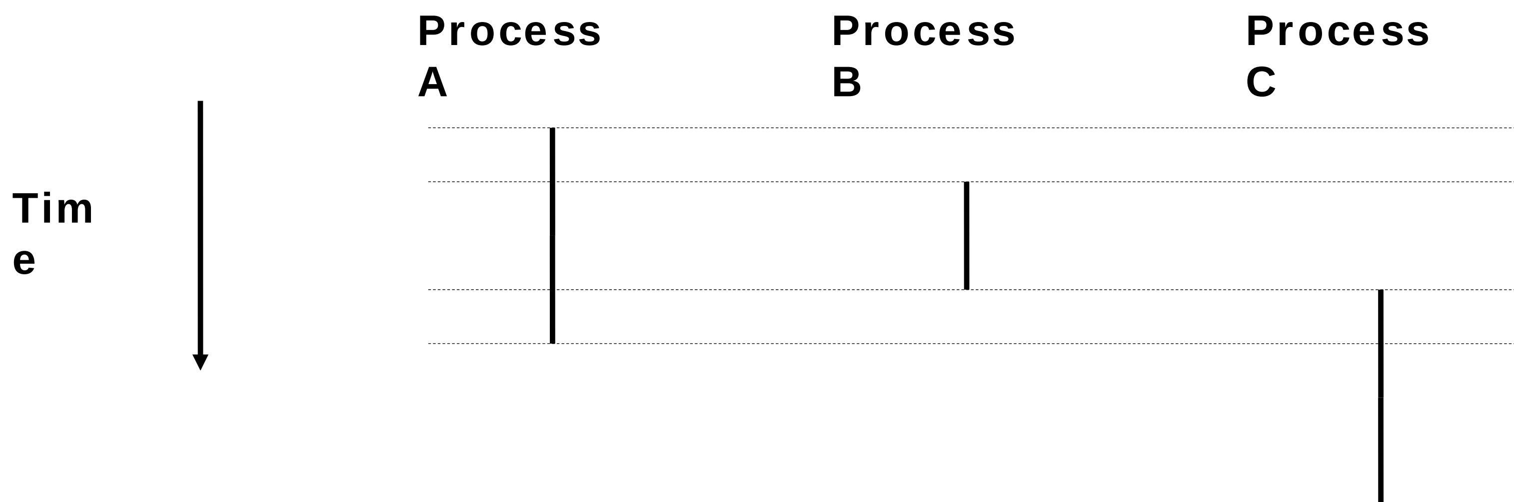
- 两个进程并发运行
  - 指他们在运行时间上有重叠
- 否则，进程间是顺序执行的
- Examples:
  - Concurrent: A & B, A&C
  - Sequential: B & C

- **Concurrent：宏观上并行，微观上可能还是串行**
- **Parallel：微观上是并行的**



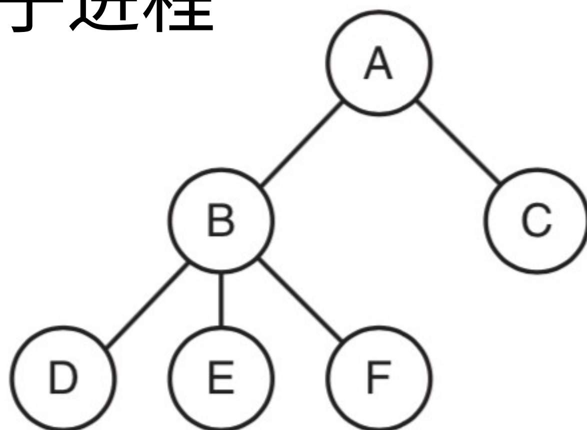
User view of concurrent processes

- 同一个 CPU 核心上的并发进程的控制流（指令执行）在物理上实际是不相交的
- 然而，在用户看起来他们是并行执行的

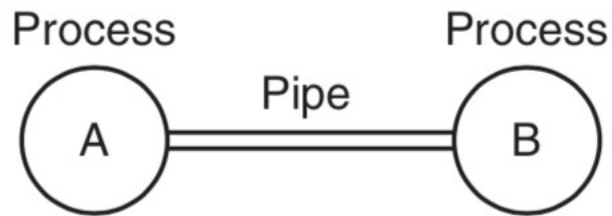


Process 间关系

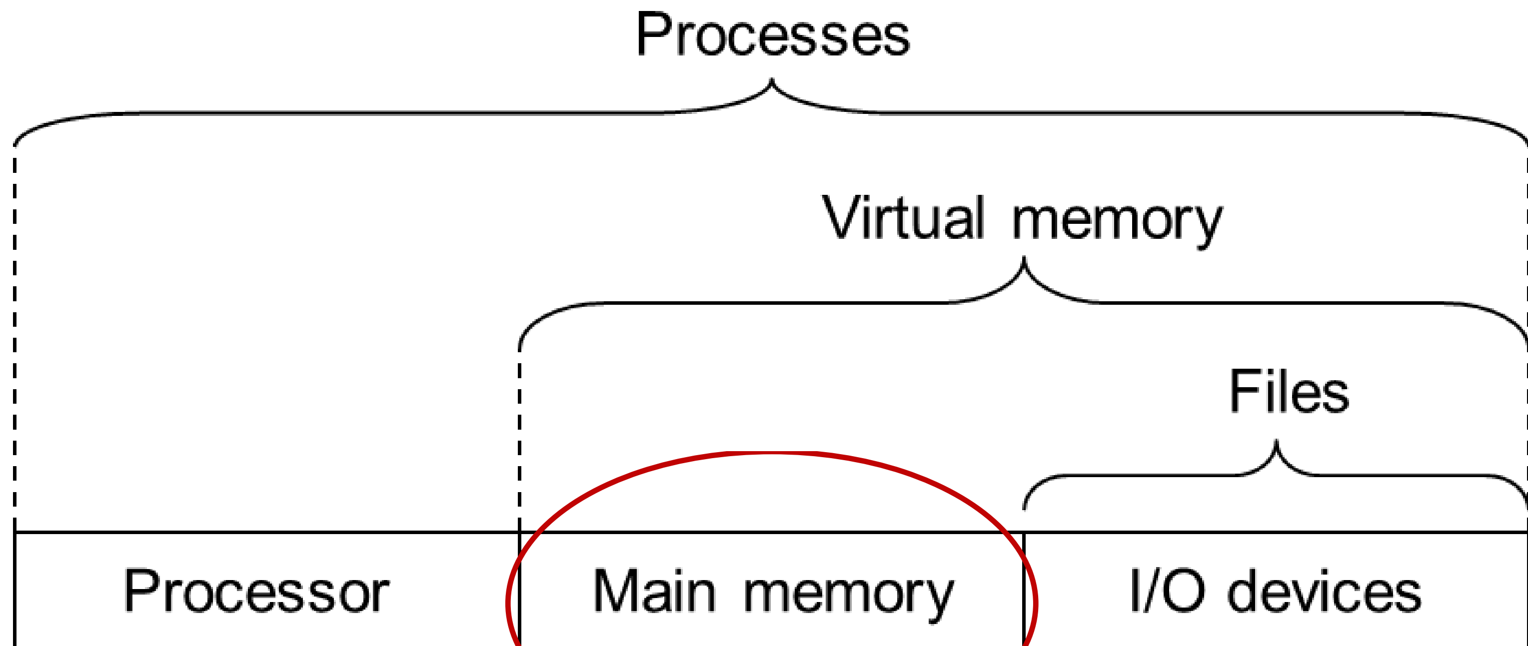
- 父进程 / 子进程



- 进程间通信（管道）

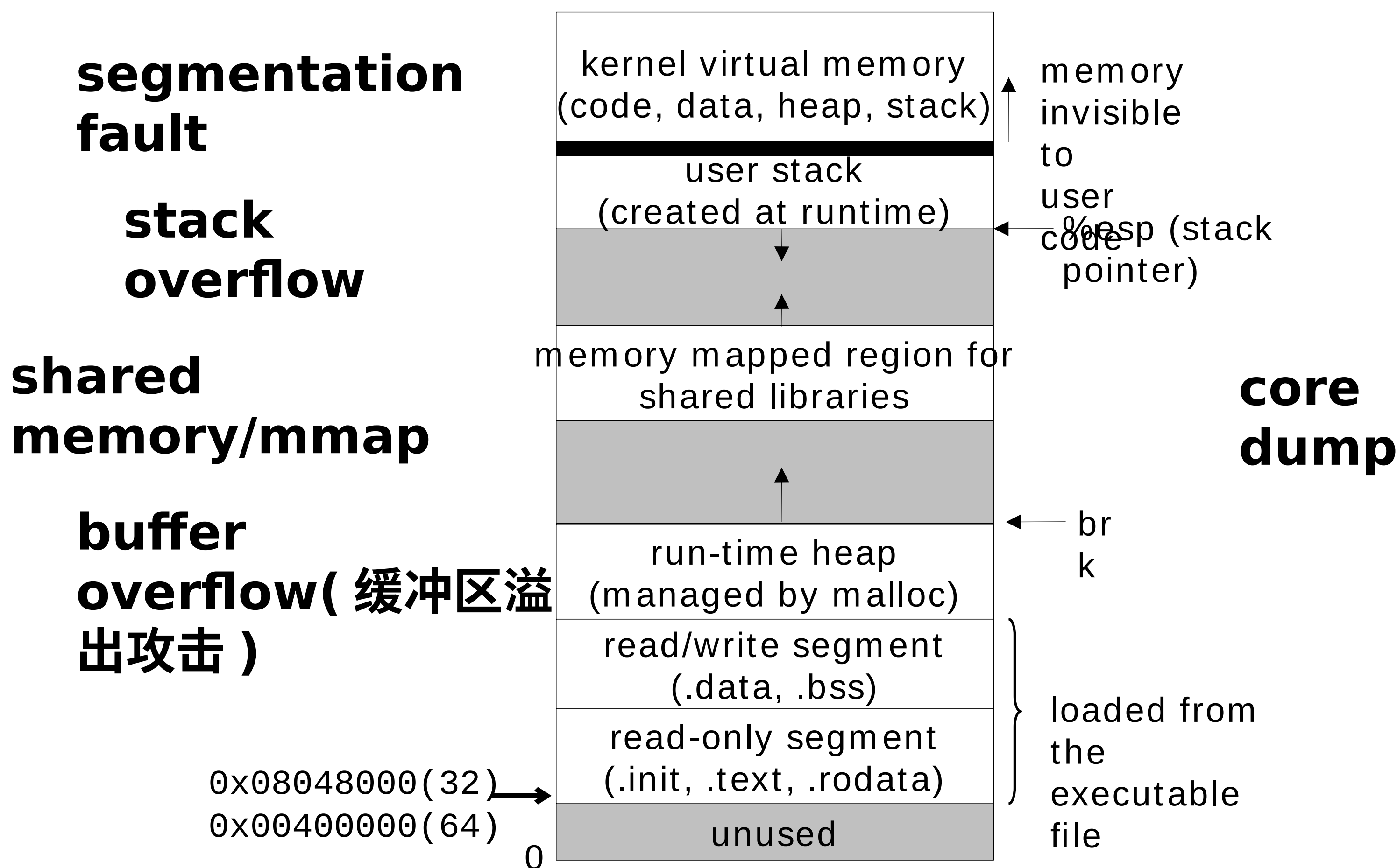


Abstractions Provided by OS



Private address spaces

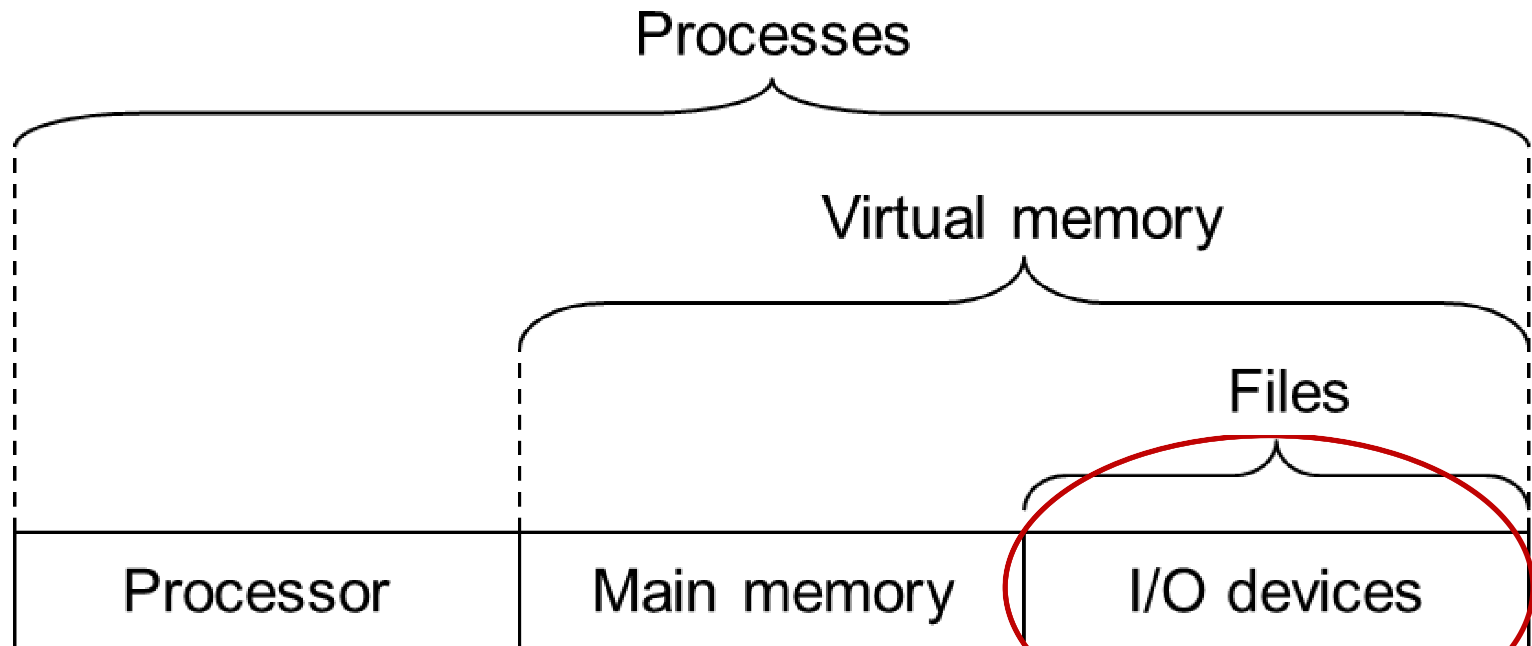
- 每个进程都有自己的私有地址空间



Virtual Memory

- 虚存是一种抽象
  - 每个进程看来都独占整个主存
  - 虚存地址空间包括：
    - Program code and data (global variables)
    - Heap
    - Shared libraries (standard and math libraries)
    - Stack (function calls)
    - Kernel (operating system)
  - 硬件协助进行虚拟地址到物理地址的转换 / 翻译（速度快）

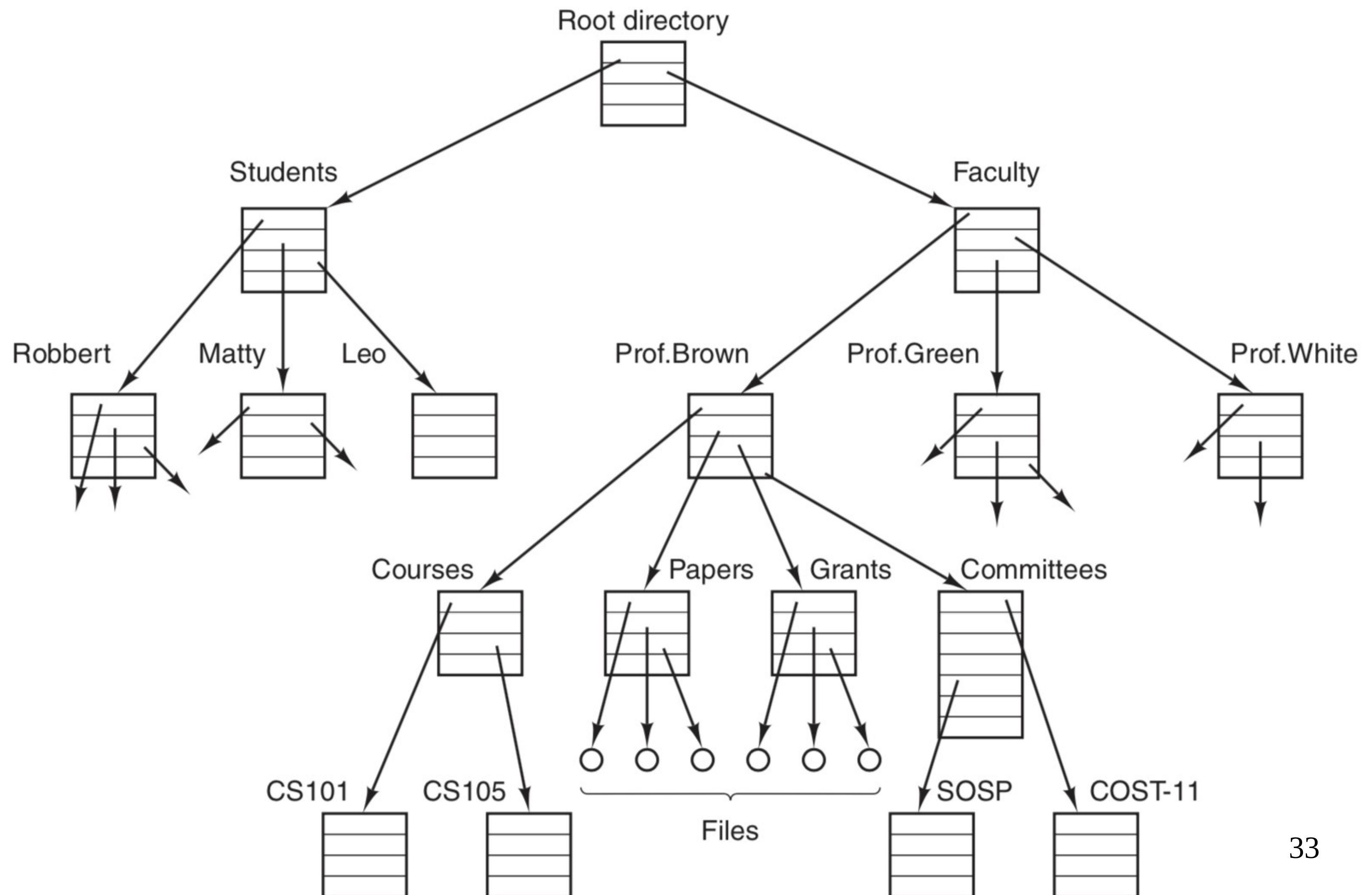
Abstractions Provided by OS



Files

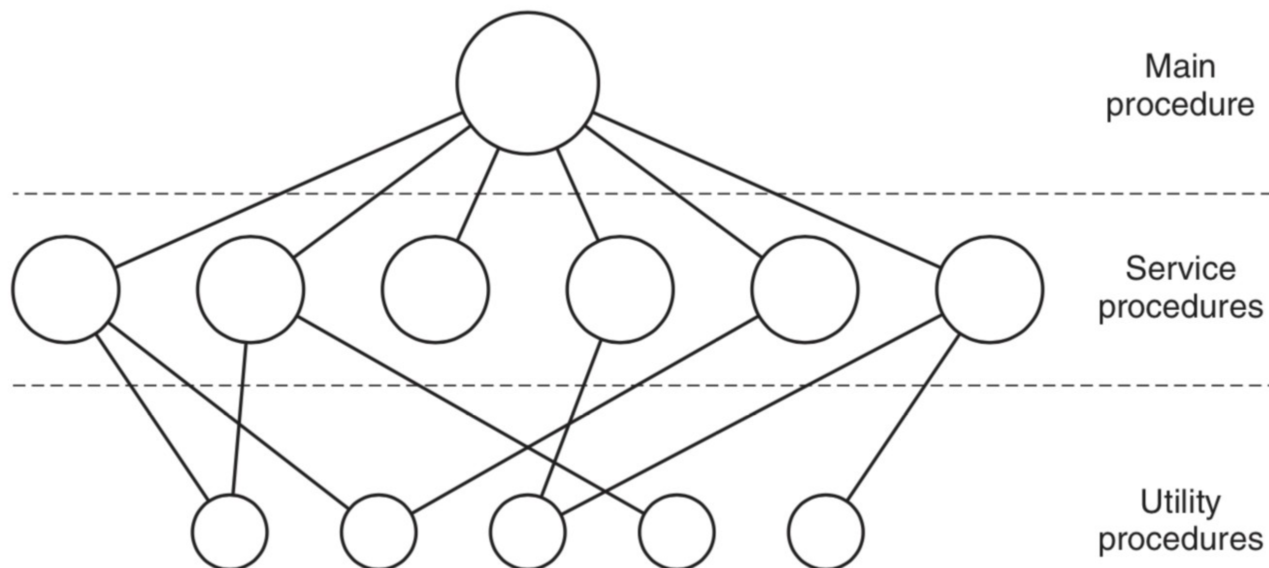
- 文件是对 I/O （持久化存储）资源的一种抽象
  - 是一个 bytes 的序列，大小可变，通过名字（路径）可以访问到
- Linux/Unix 中每个 I/O 设备也被看成一个文件
- Unix I/O
  - 使用一组简单的 system calls 来读、写文件
  - 系统中所有的输入输出工作，都由 Unix I/O 来完成
  - 各种语言和 library 中的文件读写都是对 Unix I/O 的封装

Files



OS Architecture

- **单内核（单体系统，** monolithic systems ）
  - 整个 OS 内核是一个单一程序，复杂，任何一个过程的崩溃都导致整个 OS 崩溃
  - 主过程 -> 服务过程 (对应系统调用)-> 工具过程
  - 例如 Linux （架构虽然落后，但实用）



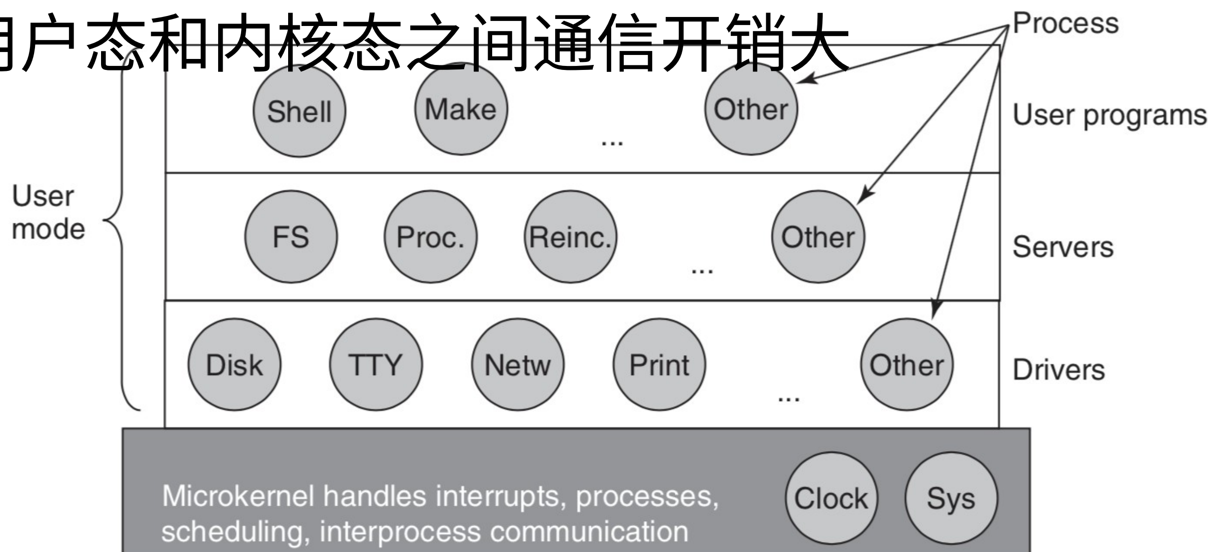
OS Architecture

- **层次式系统**
  - 上层软件是在下层软件基础上构建的
    - 例如 Layer1 以上部分不用关心数据是在内存还是在磁鼓上，完全对上屏蔽
  - 如荷兰的 THE 系统（1968 年，系统很小）、 MULTICS

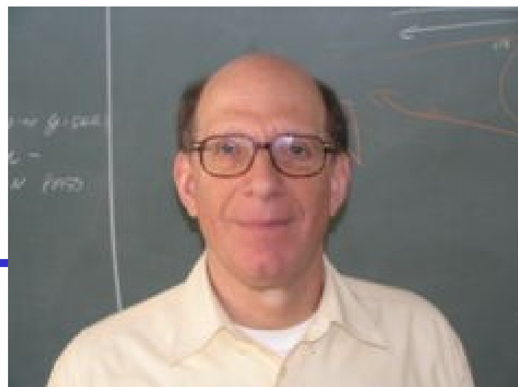
Layer	Function
5	The operator
4	User programs
3	Input/output management
2	Operator-process communication
1	Memory and drum management
0	Processor allocation and multiprogramming

OS Architecture

- 微内核（microkernels）
  - 内核很小，把 OS 划分为小的模块，运行在用户态。这样出现故障时不会导致整个系统崩溃
  - 常用于对可靠性要求高的 realtime、航空、军事领域，例如 Symbian， MINIX 3 （12K C， 1.4 Asm）
  - 但用户态和内核态之间通信开销大



OS Architecture



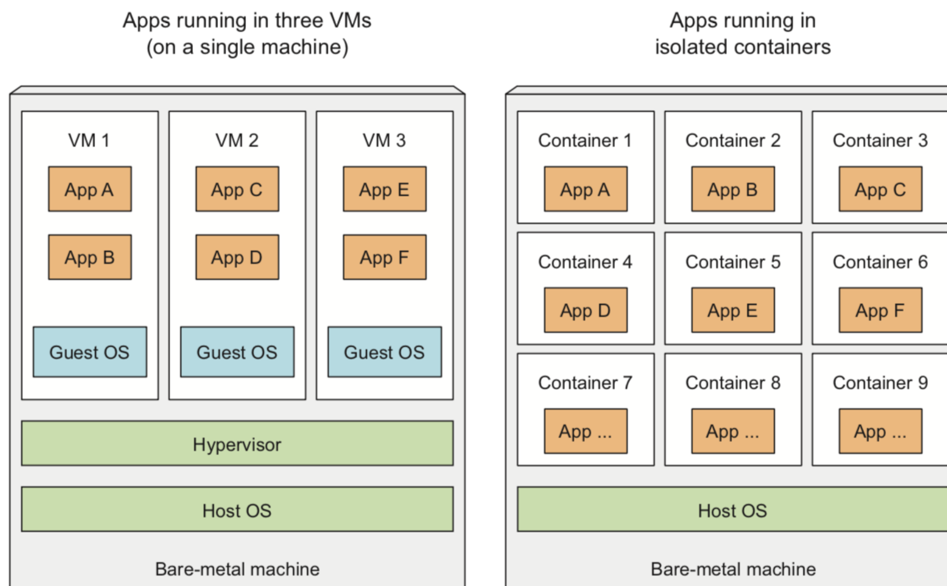
- 单内核和微内核之争
  - Tanenbaum 在 1992 年就此事同 Linus 展开了一场论述（坦能鲍姆《现代操作系统》、 Minix 作者）
  - Tanenbaum： Linux 单内核的设计主要存在几个问题
    - Linux 设计之初的单内核思想使得它同 80x86 的处理器架构紧紧绑定在了一起，这导致 Linux 的移植性低于 Minix
    - 代码的错误取决于代码大小。一般而言工业代码每千行会出现 10 个错误。将越多的代码放到内核，内核的出错可能性也越大
    - 操作系统的设计不应该因为某些代码中偶然出现的错误影响到整个内核的运行
  - Linus： 每个人的代码都不是第一次就正确，除了我

OS Architecture

- 单内核和微内核之争
  - Windows Nt 内核和 Mac OS X 的 Mach 内核是微内核典范示例，但二者最新版本中不让任何微内核服务运行在用户空间，这违背了微内核设计的初衷（为了性能）
  - Linux 是单内核，但也吸收了微内核的精华，采用模块化设计、抢占式内核、支持内核线程，以及动态加载内核模块的能力
  - 实用主义占了上风

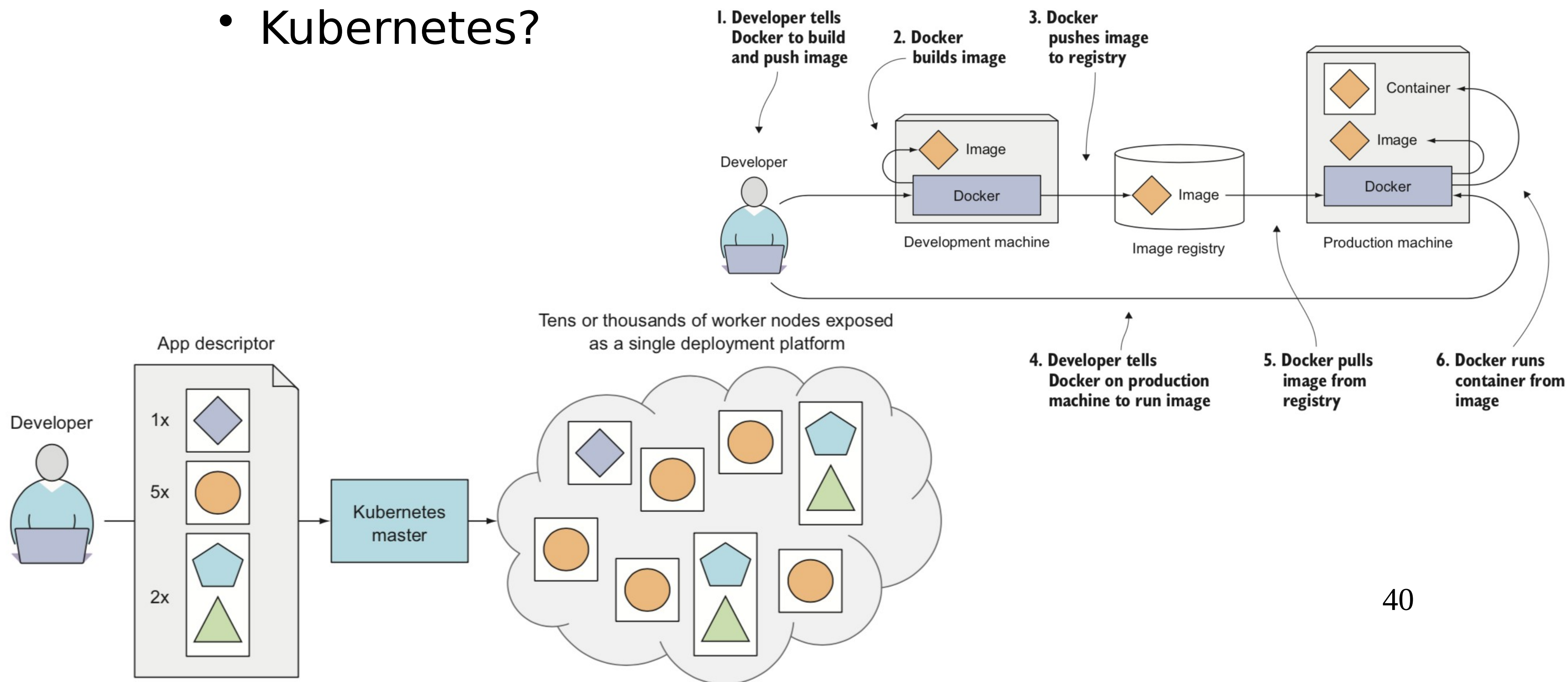
OS 迭代

- 虚拟机
  - Java 虚拟机，...
    - => 软件平台，如 Spark, Flink （上面执行计算作业 / 某种语言），也可以多人共享机器资源
  - 虚拟机（VMWare）
    - 用户态模拟器 + 内核态虚拟设备
  - 容器（Docker）
    - 隔离
    - 叠加



OS 迭代

- Cloud
  - 管理单机资源：传统 OS
  - 管理分布式集群资源： Cloud OS
    - Kubernetes?



备忘录

Exp.	Explicit	Prefix	Exp.	Explicit	Prefix
10^{-3}	0.001	milli	10^{3}	1,000	Kilo
10^{-6}	0.000001	micro	10^{6}	1,000,000	Mega
10^{-9}	0.000000001	nano	10^{9}	1,000,000,000	Giga
10^{-12}	0.000000000001	pico	10^{12}	1,000,000,000,000	Tera
10^{-15}	0.000000000000001	femto	10^{15}	1,000,000,000,000,000	Peta
10^{-18}	0.000000000000000001	atto	10^{18}	1,000,000,000,000,000,000	Exa
10^{-21}	0.000000000000000000001	zepto	10^{21}	1,000,000,000,000,000,000,000	Zetta
10^{-24}	0.000000000000000000000001	yocto	10^{24}	1,000,000,000,000,000,000,000,000	Yotta