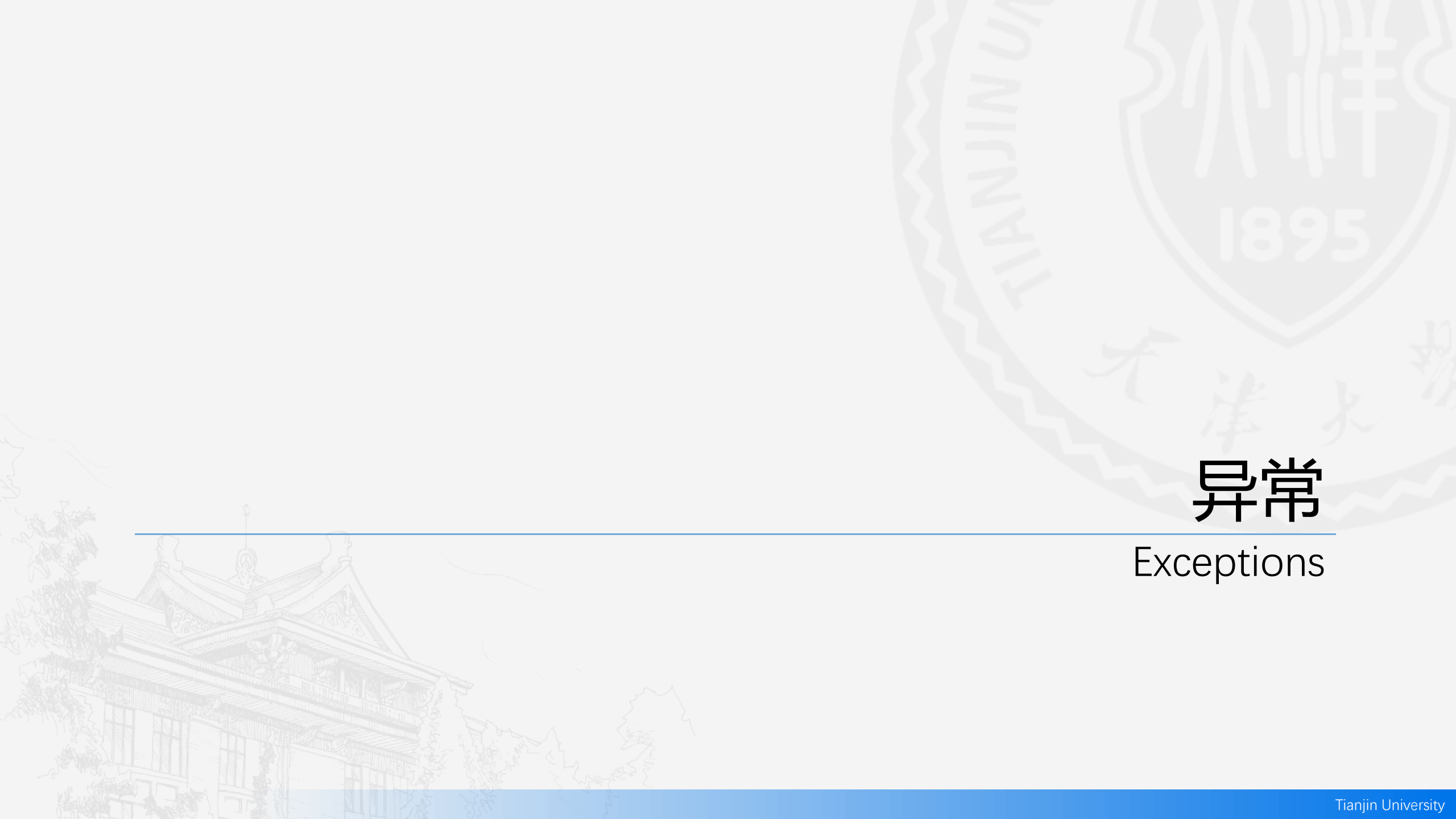


异常

Exceptions

本章内容

Topic

- 异常控制流

  Exceptional Control Flow

- 异常

  Exceptions

- 进程

  Processes

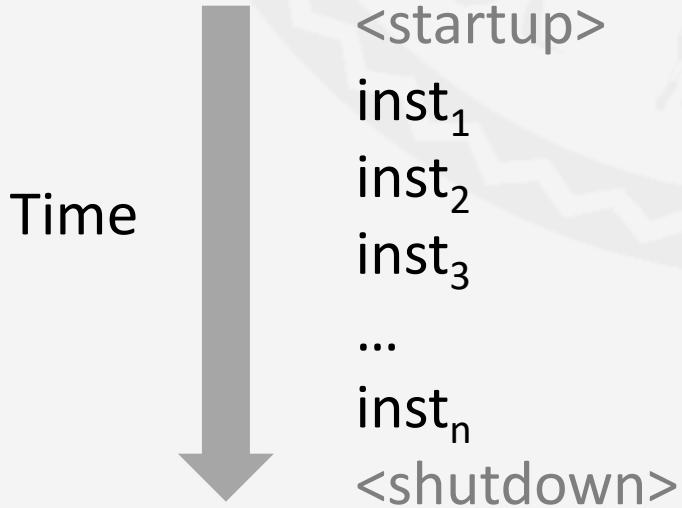
异常控制流

Exceptional Control Flow

控制流
Control Flow

- 处理器其实只做了一件事情：
  Processors do only one thing:
  - 从开机到停机，一个CPU只是单纯地对一系列的指令序列进行加载和执行（解释），一次一条指令
    From startup to shutdown, a CPU simply reads and executes (interprets) a sequence of instructions, one at a time
  - 这个指令序列就是这个处理器的控制流
    This sequence is the CPU's control flow (or flow of control)

Physical control flow



异常控制流

Exceptional Control Flow

改变控制流
Altering the Control Flow

- 截止目前：我们已经知道两种改变控制流的方法
  Up to now: two mechanisms for changing control flow
  - 跳转和分支
    Jumps and branches
  - 过程调用和返回
    Call and return
- 这些都反映了**程序状态**的改变
  React to changes in **program state**

- 这对一个有用的系统来说并不充分：不能反映**系统状态**变化
  Insufficient for a useful system: Difficult to react to changes in **system state**
  - 磁盘或网络适配其中的数据就绪
    Data arrives from a disk or a network adapter
  - 指令除0
    Instruction divides by zero
  - 通过键盘输入Ctrl-C
    User hits Ctrl-C at the keyboard
  - 系统定时器超时
    System timer expires
- 系统需要另一种机制：异常控制流
  System needs mechanisms for “exceptional control flow”

异常控制流
Exceptional Control Flow

- 计算机系统的各层次上都存在异常控制流
  Exists at all levels of a computer system
- 底层机制
  Low level mechanisms
  - 异常
    Exceptions
    - 通过改变控制流以响应系统事件（即系统状态改变）
      Change in control flow in response to a system event (i.e., change in system state)
    - 硬件和操作系统相互配合实现
      Implemented using combination of hardware and OS software
- 更高层次的机制
  Higher level mechanisms
  - 进程上下文切换
    Process context switch
    - 由硬件定时器配合操作系统实现
      Implemented by OS software and hardware timer
  - 信号
    Signals
    - 由操作系统实现
      Implemented by OS software
  - 非本地跳转：`setjmp()` 和 `longjmp()`
    Nonlocal jumps: `setjmp()`and `longjmp()`
    - 由C语言库函数实现
      Implemented by C runtime library

本章内容

Topic

- 异常控制流

  Exceptional Control Flow

- 异常

  Exceptions

- 进程

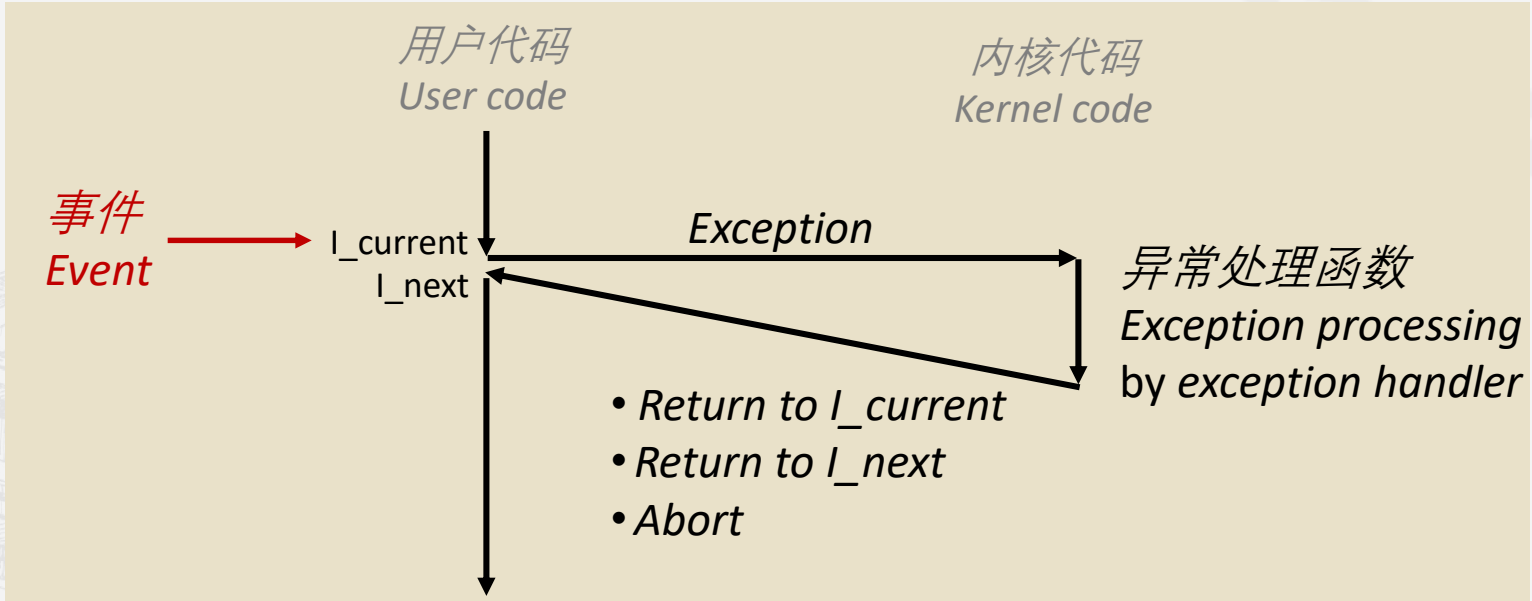
  Processes

异常

Exceptions

异常控制流
Exceptional Control Flow

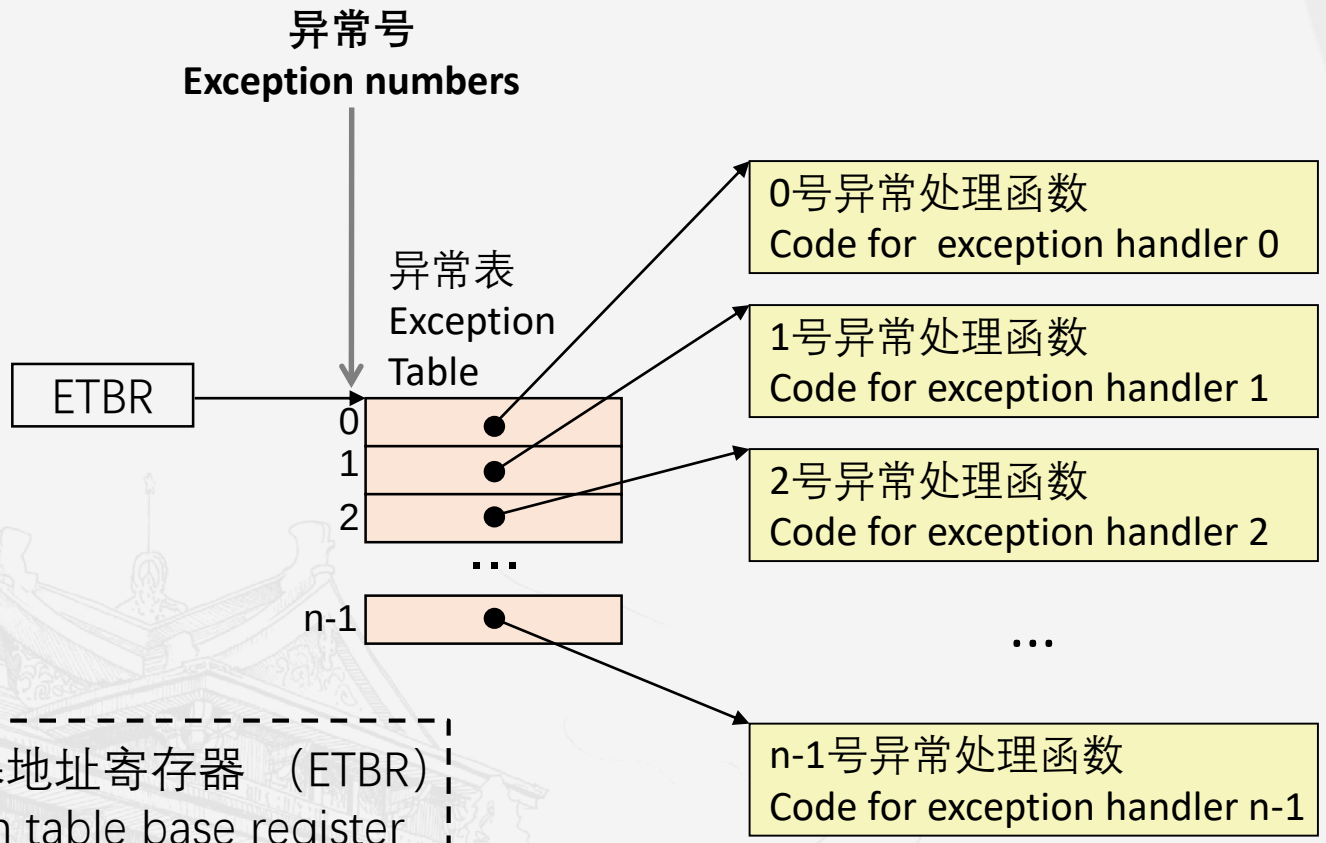
- 一个异常是为了响应某个事件（即处理器状态的改变）而将控制权转移到操作系统内核
  An exception is a transfer of control to the OS kernel in response to some event (i.e., change in processor state)
  - 内核是操作系统的驻留在内存的部分
    Kernel is the memory-resident part of the OS
  - 异常事件示例：除0，算术运算溢出，缺页，I/O请求完成，输入Ctrl-C
    Examples of events: Divide by 0, arithmetic overflow, page fault, I/O request completes, typing Ctrl-C



异常

Exceptions

异常表
Exception Table



异常表基地址寄存器 （ETBR）
Exception table base register

每个类型的事件都有一个唯一的异常号 k
Each type of event has a unique exception number k

K是异常表的索引 （又名：中断向量）
k = index into exception table (a.k.a. interrupt vector)

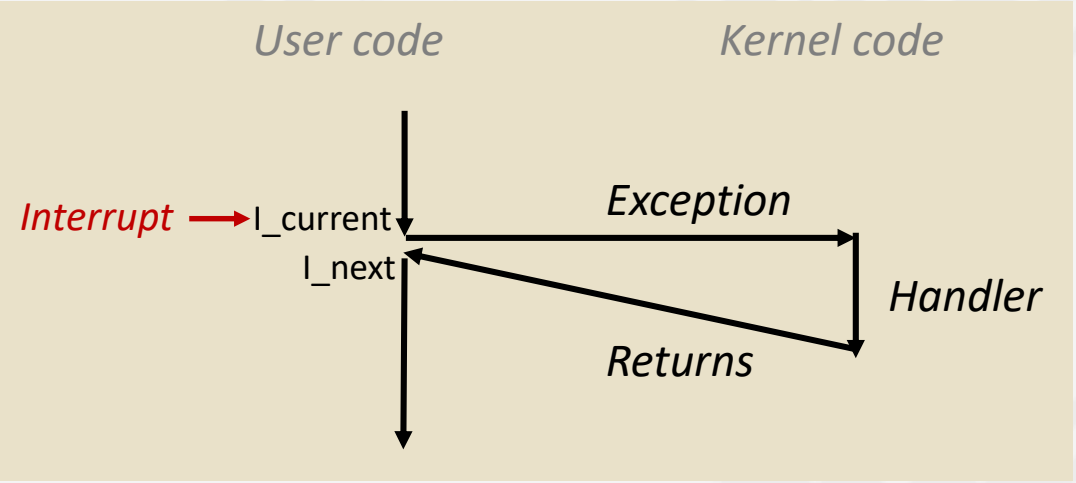
每次发生异常k时，调用异常处理函数k
Handler k is called each time exception k occurs

异常

Exceptions

异步异常（中断）
Asynchronous Exceptions (Interrupts)

- 由处理器外部的事件引起
  Caused by events external to the processor
  - 通过设置好的处理器中断引脚触发
    Indicated by setting the processor's interrupt pin
  - 处理完成后返回“下一个”指令
    Handler returns to “next” instruction
- 时钟中断
  Timer interrupt
  - 每几个毫秒，外部定时器芯片会触发一次中断
    Every few ms, an external timer chip triggers an interrupt
  - 内核处理后返回用户程序
    Used by the kernel to take back control from user programs



- 外部设备的I/O中断
  I/O interrupt from external device
  - 键盘输入`Ctrl-C`
    Hitting Ctrl-C at the keyboard
  - 磁盘/网络的数据包就绪
    Arrival of a packet from a network or a disk

异常

Exceptions

同步异常
Synchronous Exception

- 由执行指令发生的事件引起的：
  Caused by events that occur as a result of executing an instruction:
  - 陷阱
    Traps
    - 有意触发的
      Intentional
    - 例如：系统调用、断点陷阱、特殊指令
      Examples: system calls, breakpoint traps, special instructions
    - 处理完成后返回“下一个”指令
      Handler returns to “next” instruction
  - 故障
    Faults
    - 非有意，但有可能恢复
      Unintentional but possibly recoverable
    - 例如：缺页故障（可恢复）、保护故障（不可恢复）、浮点异常
      Examples: page faults (recoverable), protection faults (unrecoverable), floating point exceptions
    - 重新执行引起故障的（“当前”）指令或者终止
      Either re-executes faulting (“current”) instruction or aborts
  - 终止
    Aborts
    - 非有意，不可恢复
      Unintentional and unrecoverable
    - 例如：非法指令、奇偶校验错、机器检查异常（硬件错误）
      Examples: illegal instruction, parity error, machine check
    - 终止当前程序
      Aborts current program

X86-64/Linux 系统调用
X86-64/Linux System Calls

每个x86-64系统调用都有一个唯一的ID号

Each x86-64 system call has a unique ID number

Number	Name	Description
0	read	Read file
1	write	Write file
2	open	Open file
3	close	Close file
4	stat	Get info about file
57	fork	Create process
59	execve	Execute a program
60	_exit	Terminate process
62	kill	Send signal to process

系统调用举例：打开文件
System Call Example: Opening File

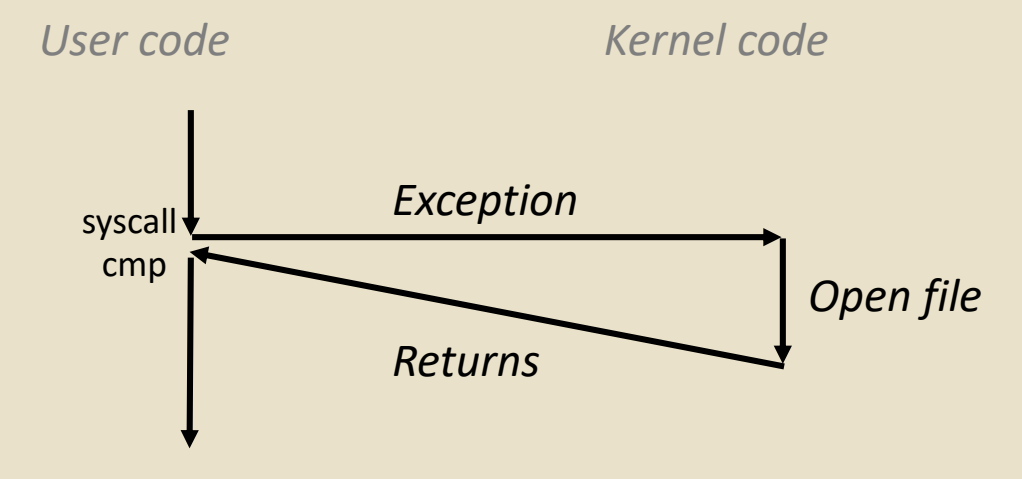
```
00000000000e5d70 <__open>:
...
e5d79:   b8 02 00 00 00      mov  $0x2,%eax            # open is syscall #2
e5d7e:   0f 05               syscall                   # Return value in %rax
e5d80:   48 3d 01 f0 ff ff   cmp  $0xfffffffffffff001,%rax
...
e5dfa:   c3                  retq
```

用户调用：

User calls: `open(filename, options)`

通过调用 `__open` 函数，实现系统调用 `syscall`

Calls `__open` function, which invokes system call instruction `syscall`



`%rax` 包含了系统调用号

`%rax` contains syscall number

其他参数通过`%rdi, %rsi, %rdx, %r10, %r8, %r9`传递

`Other arguments in %rdi, %rsi, %rdx, %r10, %r8, %r9`

返回值在`%rax`中

Return value in `%rax`

负数返回值表示错误，与负`errno`对应

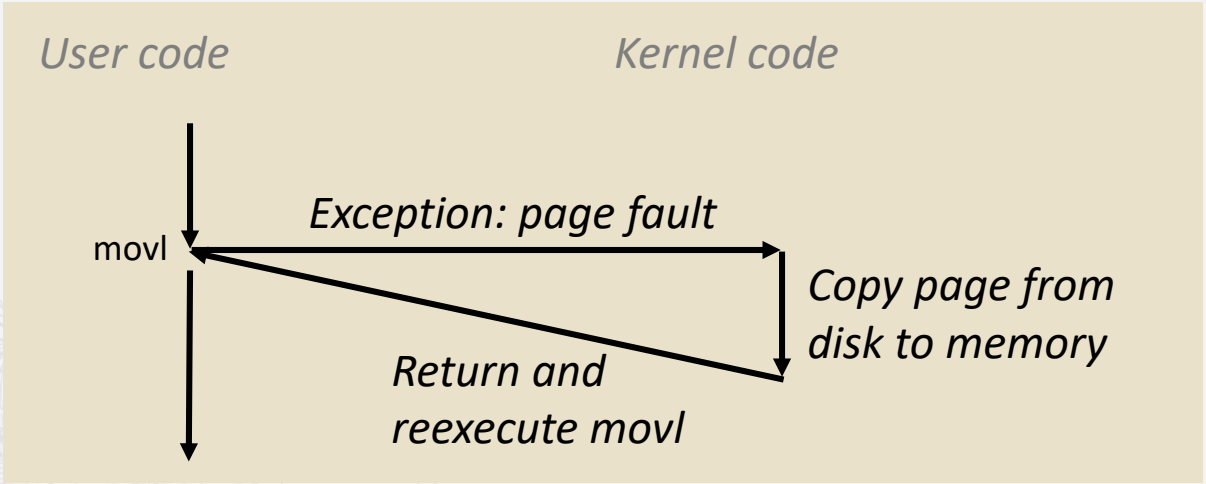
Negative value is an error corresponding to negative `errno`

故障异常举例：缺页故障
Fault Example: Page Fault

用户写存储器的某个位置
User writes to memory location
写的那个存储器位置当前在磁盘上
That portion (page) of user's memory is currently on disk

```
int a[1000];
main ()
{
    a[500] = 13;
}
```

```
80483b7:        c7 05 10 9d 04 08 0d     movl   $0xd,0x8049d10
```

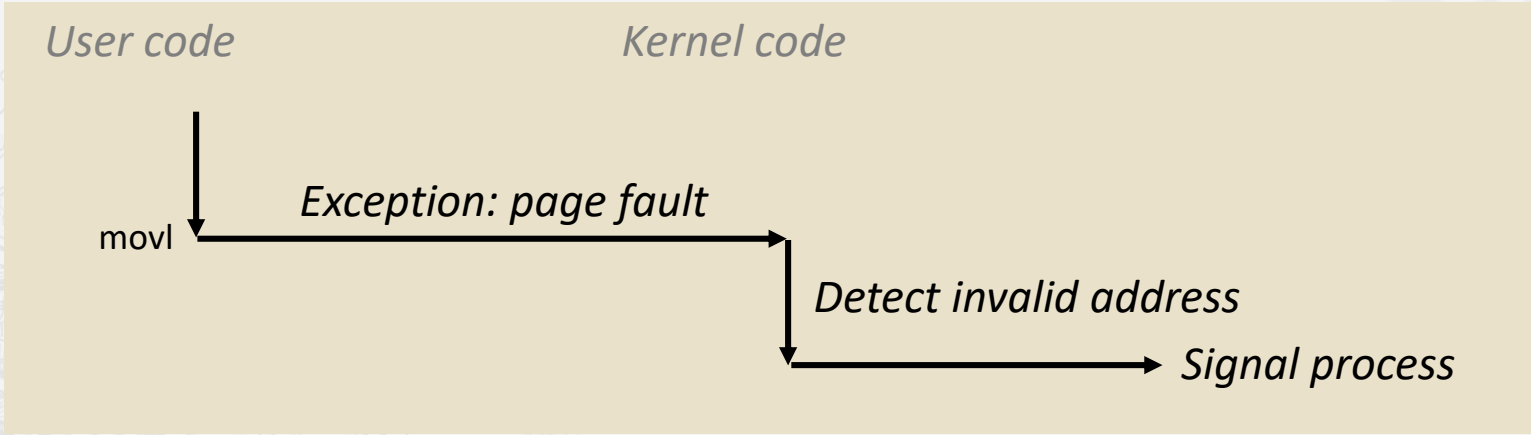


故障异常举例：无效内存引用
Fault Example: Page Fault

```
int a[1000];
main ()
{
    a[5000] = 13;
}
```

向用户进程发送SIGSEGV信号
Sends SIGSEGV signal to user process
用户进程退出，并打印“段错误”
User process exits with “segmentation fault”

```
80483b7:        c7 05 60 e3 04 08 0d     movl   $0xd,0x804e360
```



本章内容

Topic

- 异常控制流

  Exceptional Control Flow

- 异常

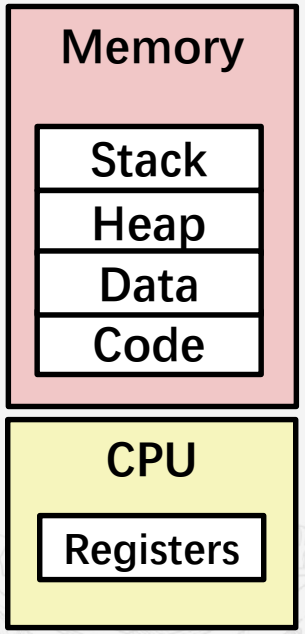
  Exceptions

- 进程

  Processes

进程

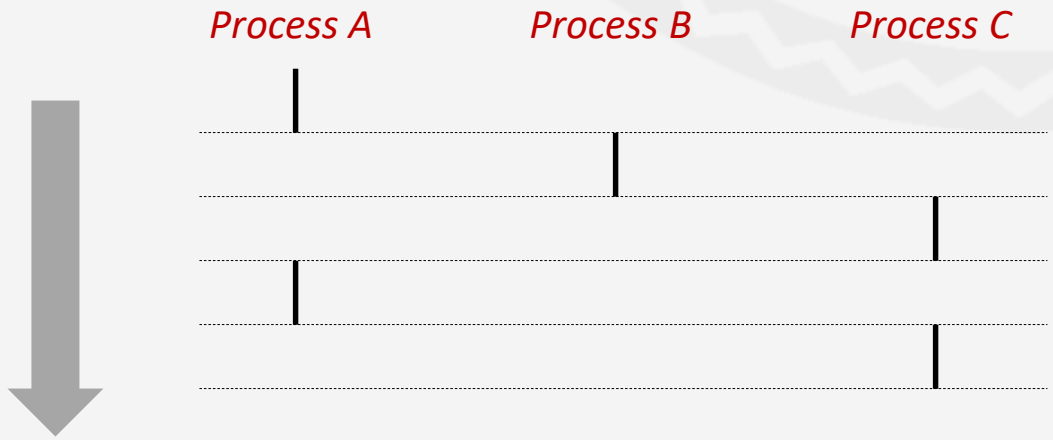
Processes



- 定义：一个进程是一个正在运行的程序的实例 （动态的概念）
  Definition: A **process** is an instance of a running program
  - 这是计算机科学中最深刻的思想之一
    One of the most profound ideas in computer science
  - 不等同于“程序”或“处理器”
    Not the same as “program” or “processor”
- 进程为每一个程序提供了两个关键的抽象
  Process provides each program with two key abstractions:
  - 逻辑控制流
    Logical control flow
    - 看上去每一个程序都是独占CPU的
      Each program seems to have exclusive use of the CPU
    - 这主要是由内核的上下文切换机制实现的
      Provided by kernel mechanism called context switching
  - 私有地址空间
    Private address space
    - 看上去每个程序都独占主存空间
      Each program seems to have exclusive use of main memory
    - 由内核的虚拟内存机制实现的
      Provided by kernel mechanism called virtual memory

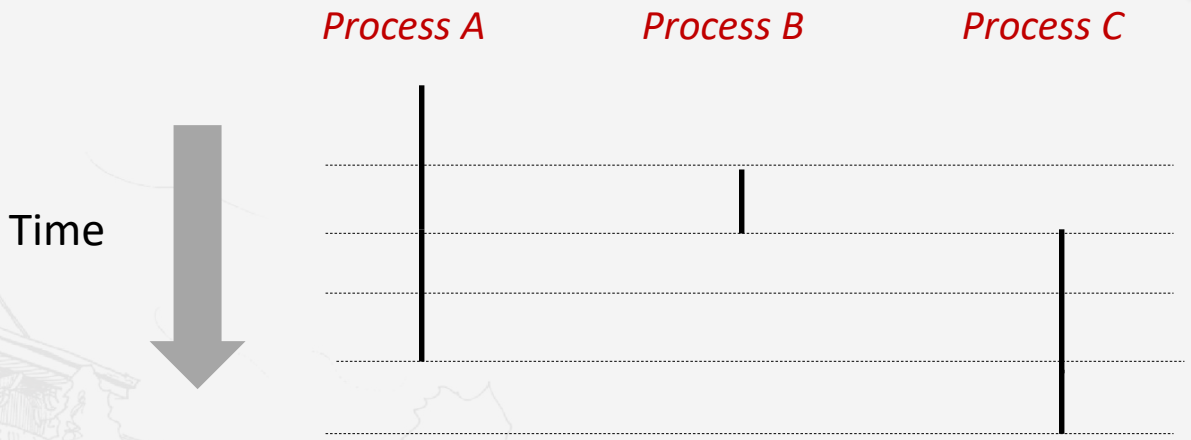
进程的并发
Concurrent Processes

- 每个进程都是一个逻辑控制流
  Each process is a logical control flow.
  - 如果两个进程的控制流在一个时间段内重合，则说明他们正在**并发**执行
    Two processes run **concurrently** (are concurrent) if their flows overlap in time
  - 如果不是，则说明他们是**顺序**执行的
    Otherwise, they are sequential
- 示例（在单处理器上运行）：
  Examples (running on single core):
  - 并发：A 与 B，A 与 C
    Concurrent: A & B, A & C
  - 顺序：B 与 C
    Sequential: B & C



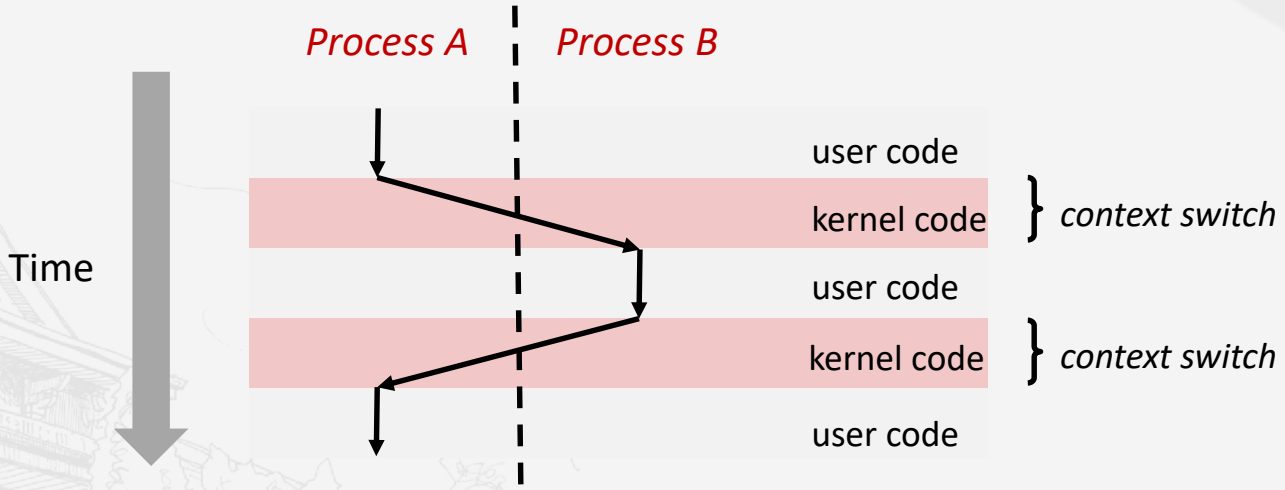
用户视角的进程并发
User View of Concurrent Processes

- 实际上，并发进程的控制流在时间上是不相交的
  Control flows for concurrent processes are physically disjoint in time
- 但是，我们可以把并发进程看做是彼此并行运行
  However, we can think of concurrent processes as running in parallel with each other



上下文切换
Context Switching

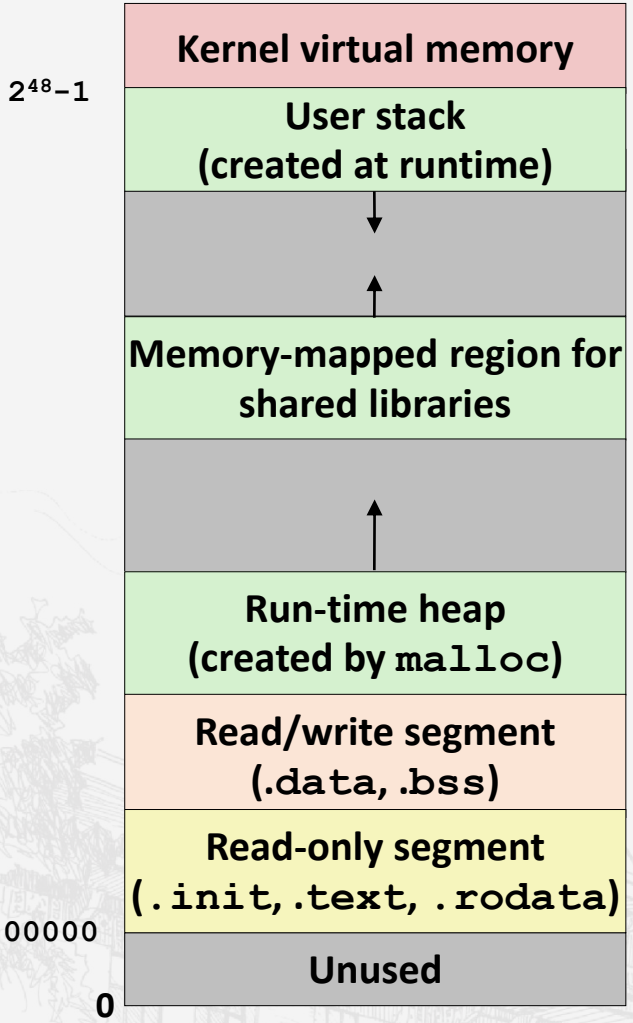
- 进程由一个内核（进程间共享的驻留在内存中的操作系统程序）管理
  Processes are managed by a shared chunk of memory-resident OS code called the kernel
  - 重要：内核不是一个独立的进程，而是作为现有进程的一部分运行
    Important: the kernel is not a separate process, but rather runs as part of some existing process.
  - 控制流通过上下文切换从一个进程传递到另一个进程
    Control flow passes from one process to another via a context switch



进程

Processes

小知识：用户模式和内核模式
Tips: User & Kernel Mode



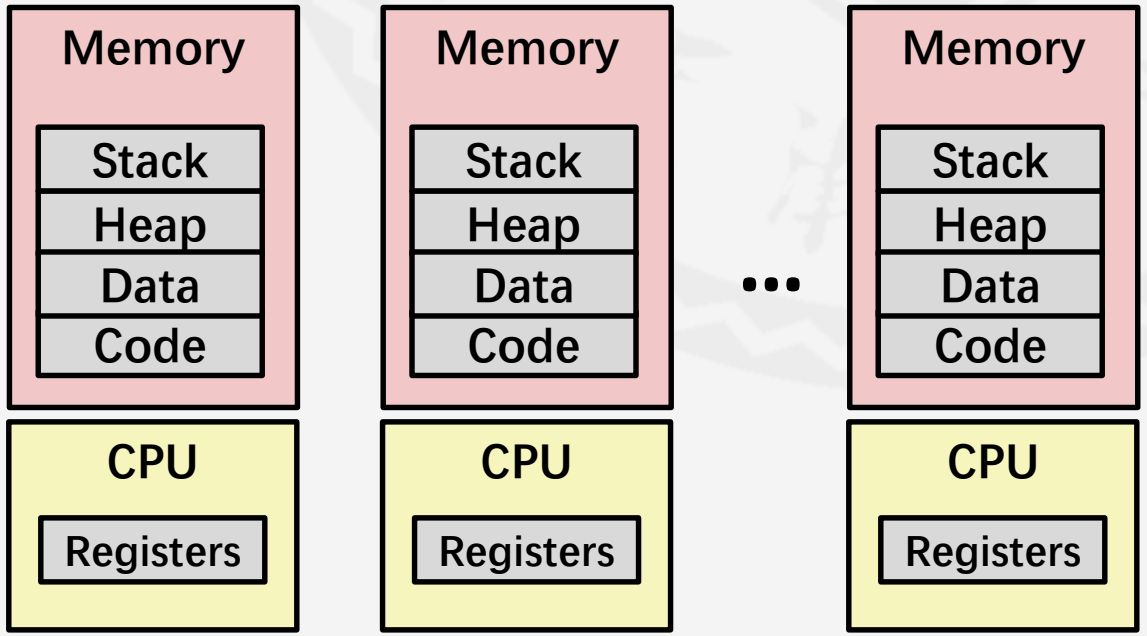
- 为了保证操作系统内核在每个进程中安全封闭，处理器提供了一种机制
  In order for the kernel to provide an airtight process abstraction, the processor provides a mechanism
  - 限制可以执行的命令和可访问的地址空间
    restricts the instructions an application can execute，and the portions of the address space that it can access
  - 通过模式位设置
    Setting by mode bit
- 用户模式
  User Mode
  - 不能够运行特权指令 （例如：停止处理器、改变模式位、发起I/O操作）
    Not allowed to execute privileged instructions
    （e.g.: halt the processor, change the mode bit, initiate an I/O operation）
  - 不能直接引用地址空间中内核去区的代码和数据
    Not allowed to directly reference code or data in the kernel area of the address space
  - 任何以上操作都会引起保护故障异常
    Any such attempt results in a fatal protection fault
  - 运行应用程序代码的进程初始时是处在用户模式中的
    A process running application code is initially in user mode
- 内核模式 （超级用户模式）
  Kernel Mode (Supervisor Mode)
  - 没有限制
    No restrictions
  - 通过异常（例如：中断、故障、陷阱），用户模式可转换为内核模式
    Change from user mode to kernel mode is via an exception such as an interrupt, a fault, or a trapping

进程

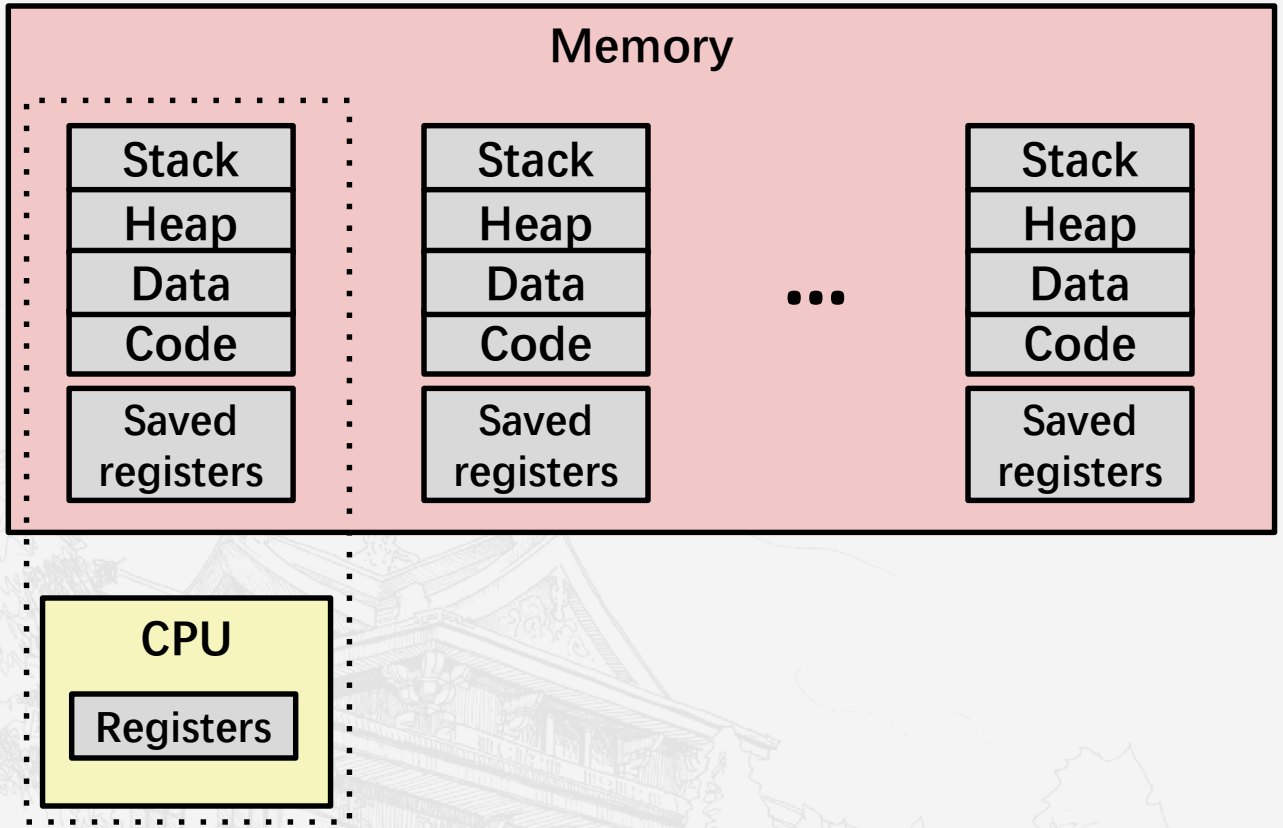
Processes

多进程：错觉
Multiprocessing: The Illusion

- 计算机同时运行着许多进程
  Computer runs many processes simultaneously
  - 一个或多个用户的应用程序
    Applications for one or more users
    - 浏览器、邮件客户端、编辑器等
      Web browsers, email clients, editors, …
  - 后台任务
    Background tasks
    - 网络和I/O设备的监控服务
      Monitoring network & I/O devices

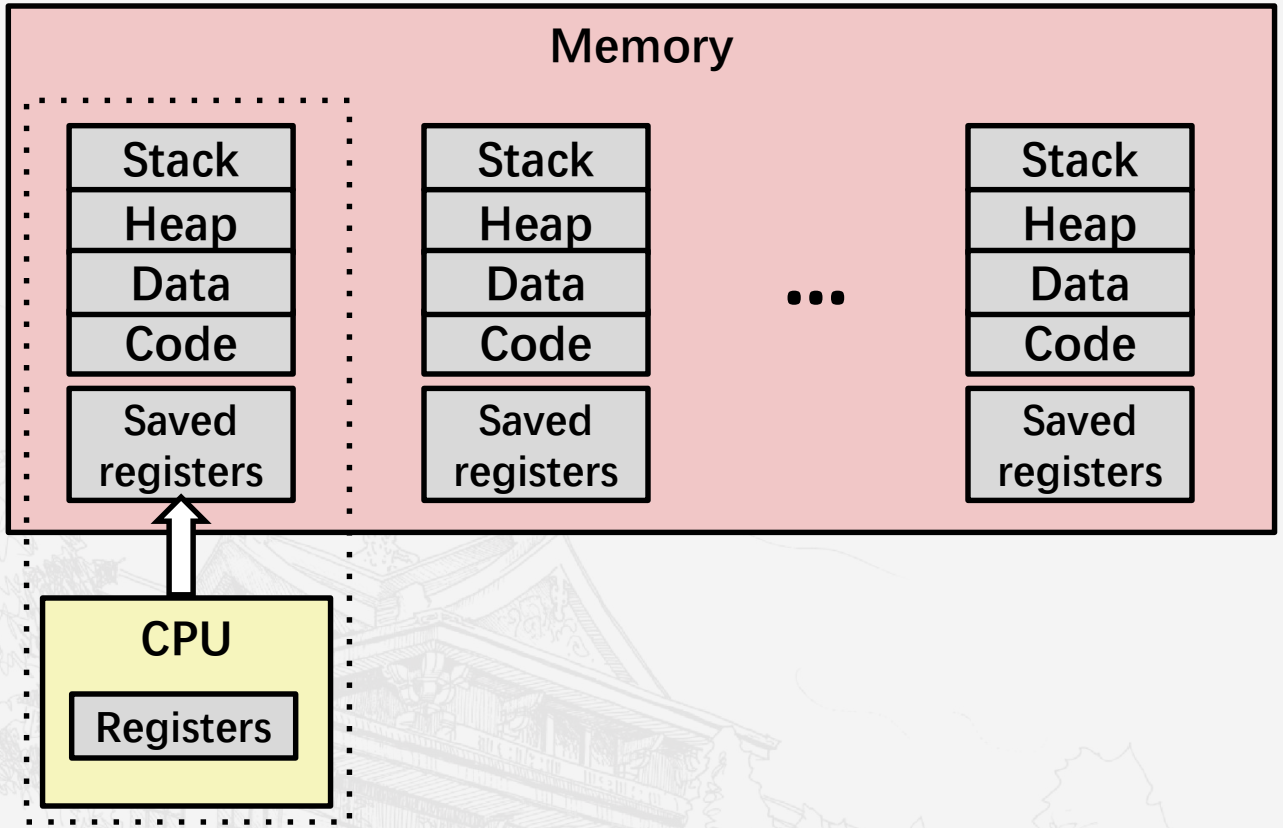


多进程：现实（传统的单处理器）
Multiprocessing: The (Traditional) Reality



- 单处理器并发地执行了多个进程
  Single processor executes multiple processes concurrently
  - 进程交错的执行（多任务）
    Process executions interleaved (multitasking)
  - 虚拟内存系统管理（各进程的）地址空间
    Address spaces managed by virtual memory system
  - 不处于运行状态进程的寄存器值存储在内存中
    Register values for nonexecuting processes saved in memory

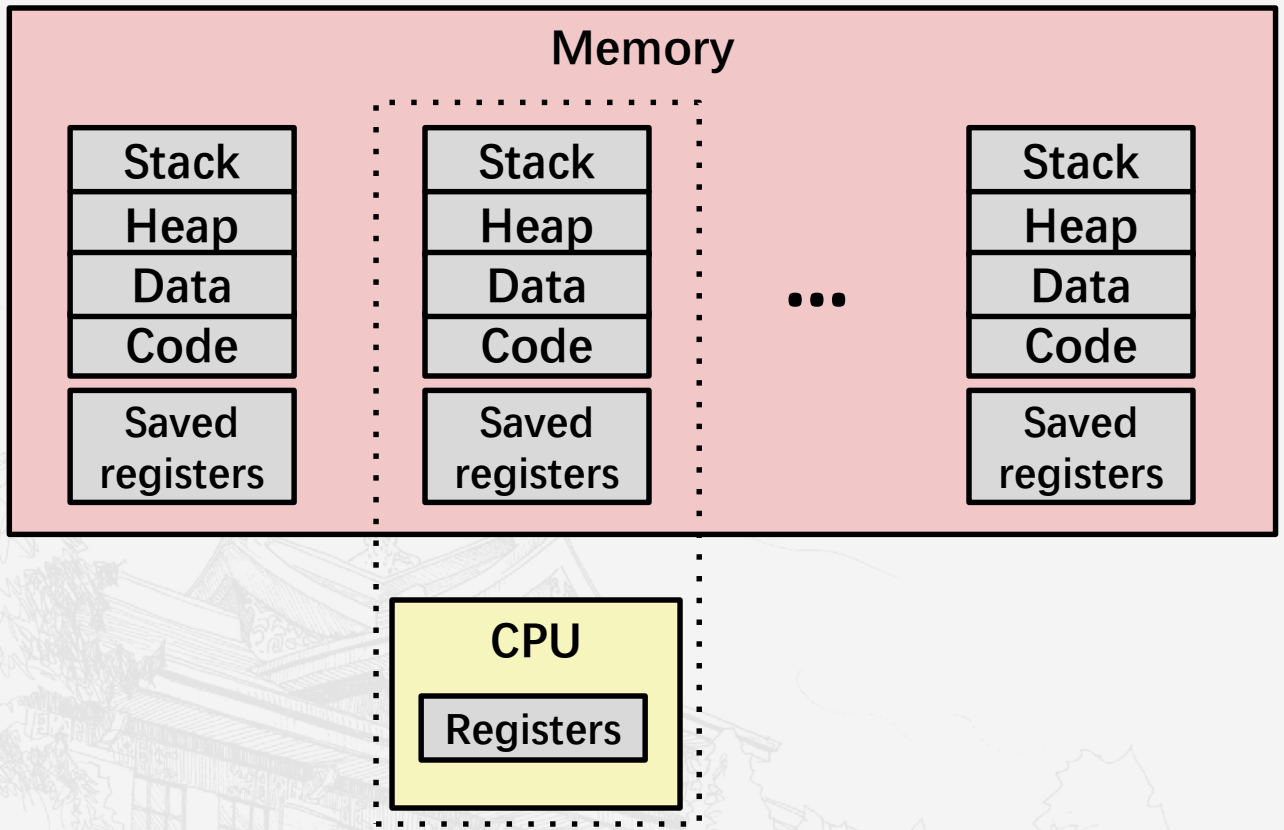
多进程：现实（传统的单处理器）
Multiprocessing: The (Traditional) Reality



将当前的寄存器保存至内存中

Single processor executes multiple processes concurrently

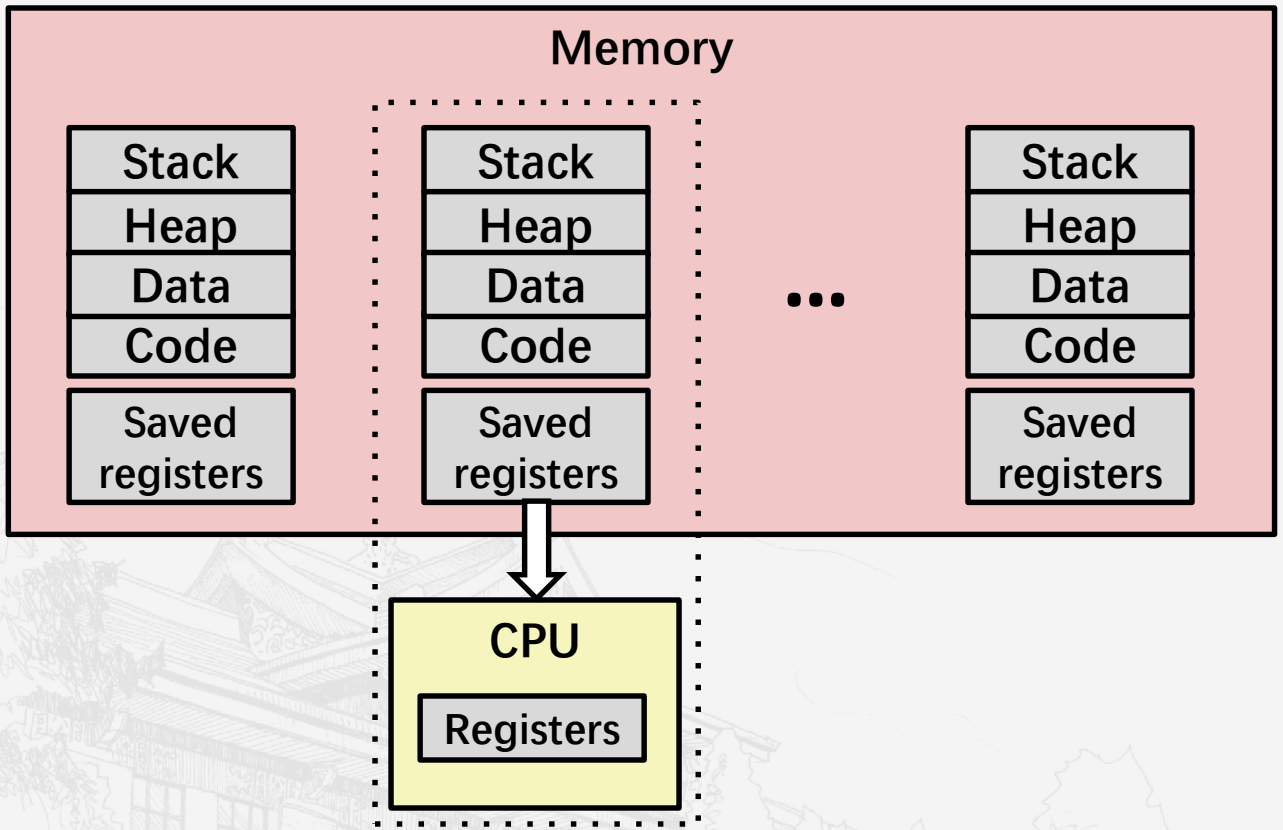
多进程：现实（传统的单处理器）
Multiprocessing: The (Traditional) Reality



调度到下一个要执行的进程

Schedule next process for execution

多进程：现实（传统的单处理器）
Multiprocessing: The (Traditional) Reality

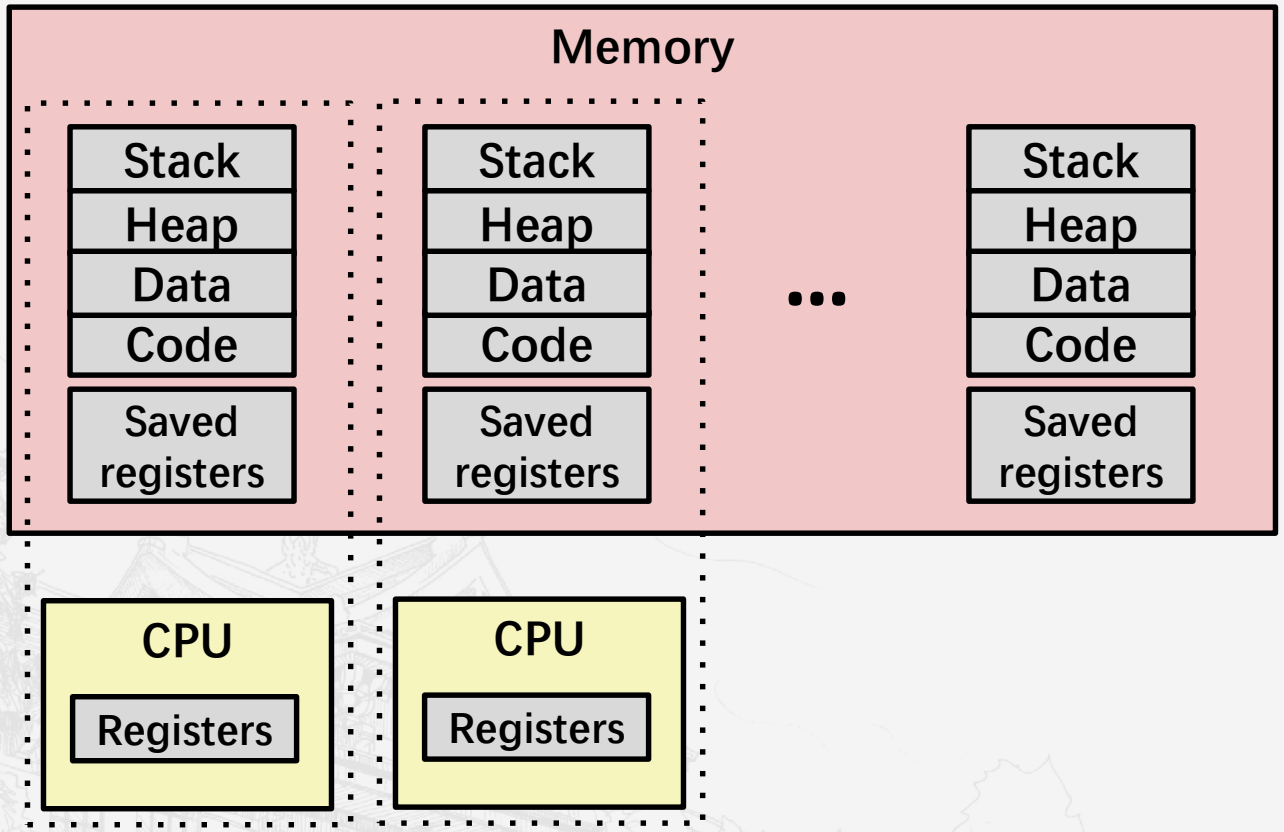


- 上下文切换
  context switch
  - 加载保存的寄存器值（从内存中），并切换地址空间
    Load saved registers and switch address space

进程

Processes

多进程：现实（现代多处理器）
Multiprocessing: The (Modern) Reality



- 多处理器
  Multicore processors
  - 多个CPU在一个芯片中
    Multiple CPUs on single chip
  - 主存共享（部分高速缓存共享）
    Share main memory (and some of the caches)
  - 每个处理器可以执行一个单独的进程
    Each can execute a separate process
    - 内核将把这些进程在多核上进行调度
      Scheduling of processors onto cores done by kernel

总结
Summary

- 异常
  Exceptions
  - 事件需要非标准的控制流（来处理）
    Events that require nonstandard control flow
  - 从外部（中断）或内部（陷阱和故障）产生
    Generated externally (interrupts) or internally (traps and faults)
- 进程
  Processes
  - 在任何时间，系统具有多个活跃的进程
    At any given time, system has multiple active processes
  - 虽然，在某个时刻，单处理器系统智能执行一个进程
    Only one can execute at a time on a single core, though
  - 每个进程都看上去完全拥有处理器和私有内存空间
    Each process appears to have total control of processor and private memory space