



CS3281 / CS5281

Exceptional Control Flow

CS3281 / CS5281

Fall 2025

**Some lecture slides borrowed and adapted from CMU's
"Computer Systems: A Programmer's Perspective"*



Tel (615) 343-7472 | Fax (615) 343-7440
1025 16th Avenue South Nashville, TN 37212
www.isis.vanderbilt.edu



This Lecture

- **Exceptional Control Flow**
- Exceptions
- Processes
- Process Control



Control Flow

- Processors do only one thing:
 - From startup to shutdown, a CPU reads and executes (interprets) a sequence of instructions, one at a time
 - This sequence is the CPU's *control flow* (or *flow of control*). The control flow is a sequence of *control transfers* from instruction to another

Time



Physical control flow

<startup>

inst₁

inst₂

inst₃

...

inst_n

<shutdown>

Altering the Control Flow

- You know two mechanisms for changing control flow:
 - Jumps and branches
 - Call and return
- Insufficient for a useful system: difficult to react to events and changing system (processor) state
 - Examples of **changes in system state (event)**:
 - Data arrives from a disk or a network adapter
 - Instruction divides by zero
 - User hits Ctrl-C at the keyboard
 - System timer expires
- System needs mechanisms for “exceptional control flow”

Exceptional Control Flow

- Exists at all levels of a computer system
- Hardware: hardware detects events (e.g., the arrival of disk data) and transfers control to handlers
- Software
 - OS: Hardware timer goes off. Then, OS transfers control from one process to another to use CPU time (virtualization)
 - User application: sends a signal to another process which handles the signal with a handler

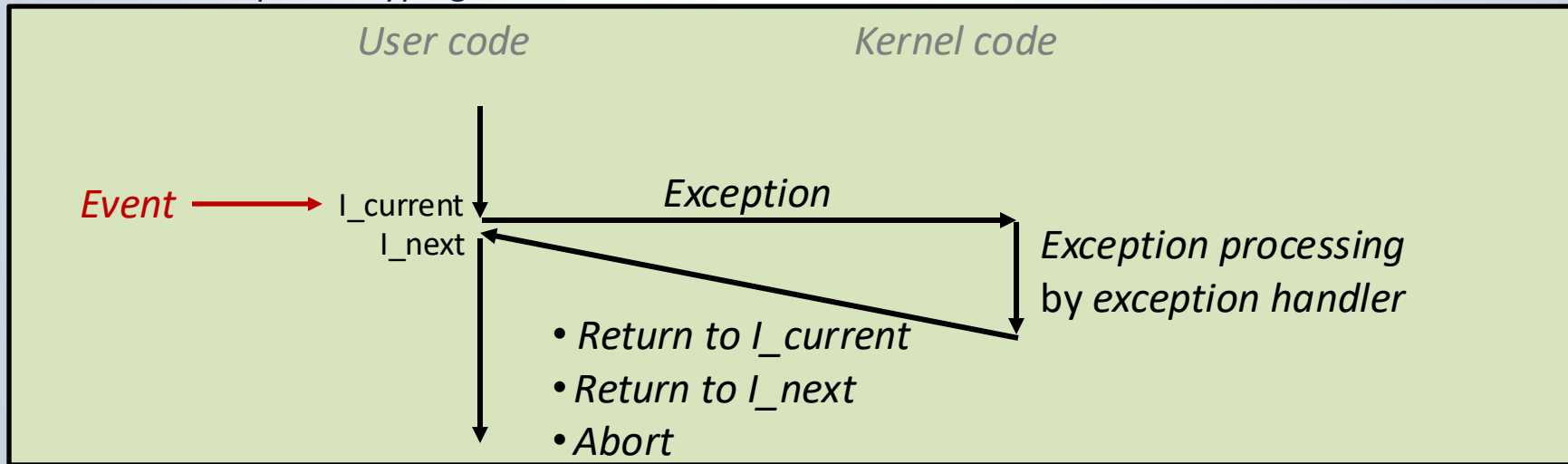
This Lecture

- Exceptional Control Flow
- **Exceptions**
- Processes
- Process Control

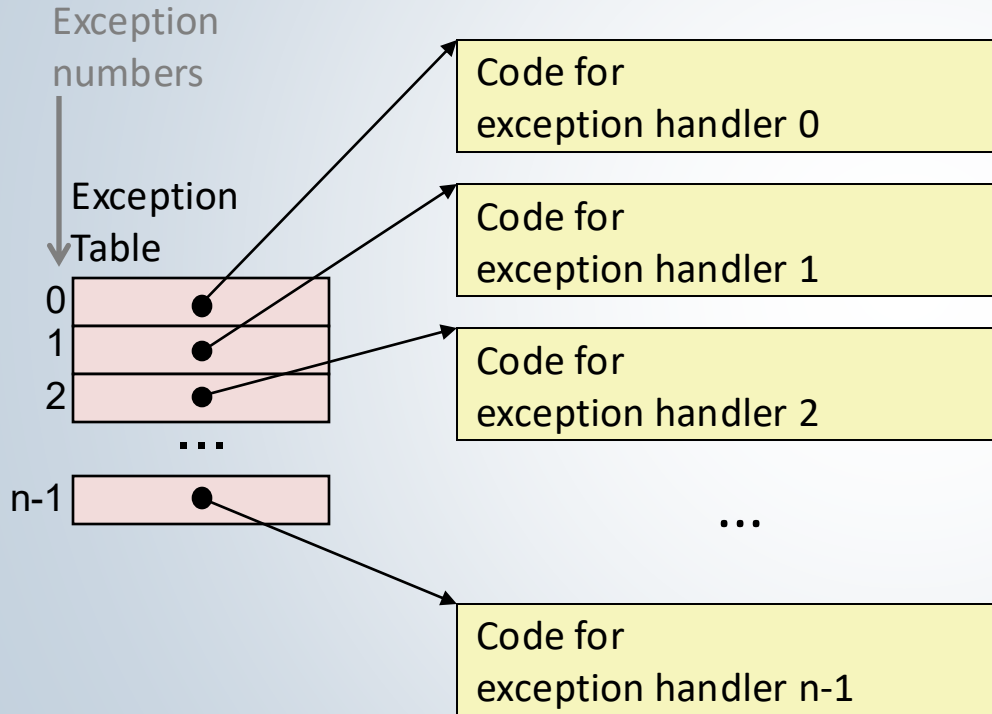


Exceptions

- An exception is an abrupt transfer of control from the user mode to the kernel mode in response to some event (i.e., change in processor state: program counter, registers, memory)
 - Examples of events: Divide by 0, arithmetic overflow, page fault, I/O request completes, typing Ctrl-C



Exception Tables



- Each type of event has a unique exception number k
- k = index into exception table (a.k.a. interrupt vector)
- Handler k is called each time exception k occurs

Asynchronous Exceptions

- Asynchronous exceptions are called interrupts
- Caused by events external to the processor
 - Indicated by setting the processor's interrupt pin
 - Handler returns to “next” instruction
- Examples:
 - Timer interrupt
 - Every few ms, a timer chip trigger an interrupt
 - Used by kernel to take back control from user programs
 - I/O interrupt from external device
 - Arrival of network packet
 - Arrival of data from disk

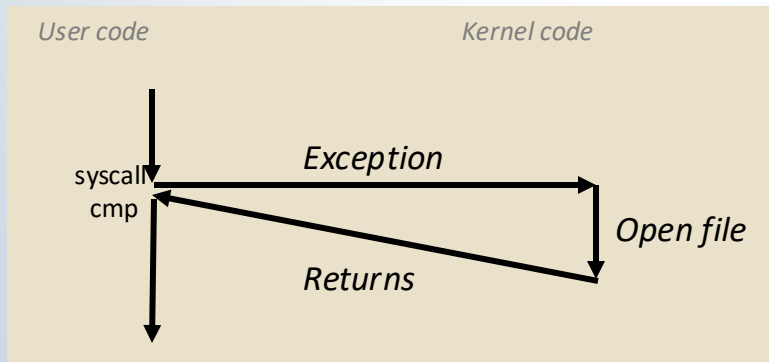
Synchronous Exceptions

- Caused by events that occur as a result of executing an instruction:
 - Traps
 - Intentional (explicit)
 - Examples: system calls
 - Returns control to “next” instruction in the user program
 - Faults
 - Unintentional but possibly recoverable
 - Examples: page faults (recoverable), protection faults (unrecoverable), floating-point exceptions (unrecoverable)
 - Either re-execute faulting instruction or abort
 - Aborts
 - Unintentional and unrecoverable
 - Examples: illegal instruction, faulty hardware
 - Aborts current program

System Call Example: Opening File

- User calls: `open(filename, options)`
- Calls `__open` function, which invokes system call instruction `syscall` (trap instruction)

```
0000000000e5d70<__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 $0xfffffffffff001,%rax  
...  
e5dfa: c3               retq
```



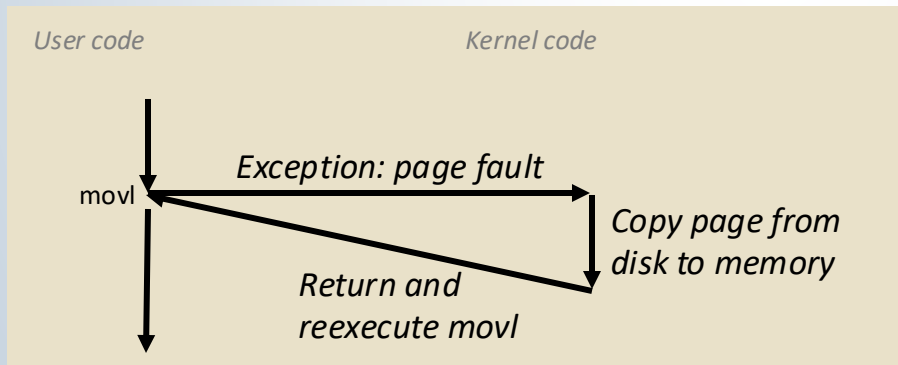
- `%eax` contains syscall number
- Other arguments in `%rdi`, `%rsi`, `%rdx`, `%r10`, `%r8`, `%r9`
- Return value in `%rax`
- Negative value is an error corresponding to negative `errno`

Fault Example: Page Fault

- User writes to memory location
- That portion (page) of user's memory is currently on disk

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

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

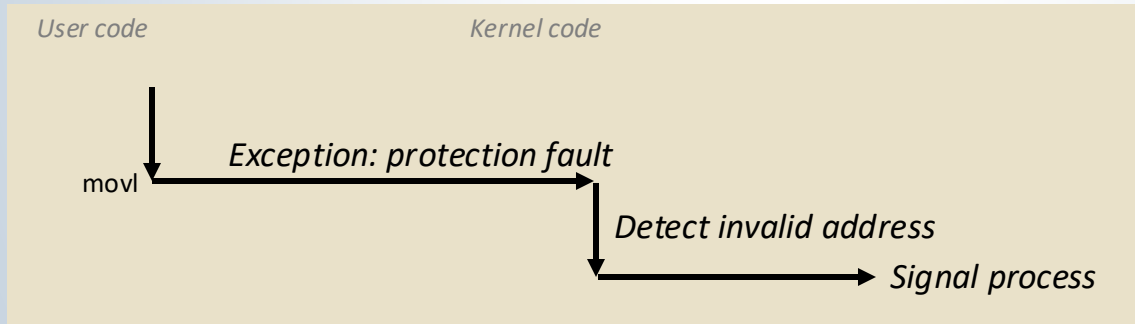


Fault Example: Invalid Memory Reference

- Buffer overflow
- Sends `SIGSEGV` signal to user process
- User process exits with “segmentation fault”

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

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



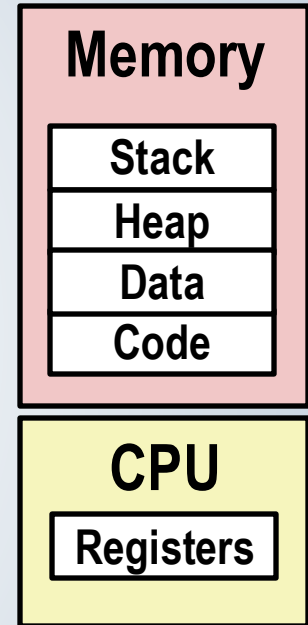
This Lecture

- Exceptional Control Flow
- Exceptions
- **Processes**
- Process Control



Processes

- Definition: A process is an instance of a running program
 - One of the most profound ideas in computer science
 - Not the same as “program” or “processor”
- OS keeps track of a process via a data structure called Process Control Block (PCB). PCB stores information (known as CPU state) such as stack, instruction address (program counter), and other registers
- Process provides each program with two key abstractions:
 - Logical control flow
 - Each program seems to have exclusive use of the CPU
 - Provided by kernel mechanism called context switching
 - Private address space
 - Each program seems to have exclusive use of main memory
 - Provided by kernel mechanism called virtual memory



Multiprocessing Example

- Running program “top” on Mac
 - System has 123 processes, 5 of which are active
 - Identified by Process ID (PID)

```

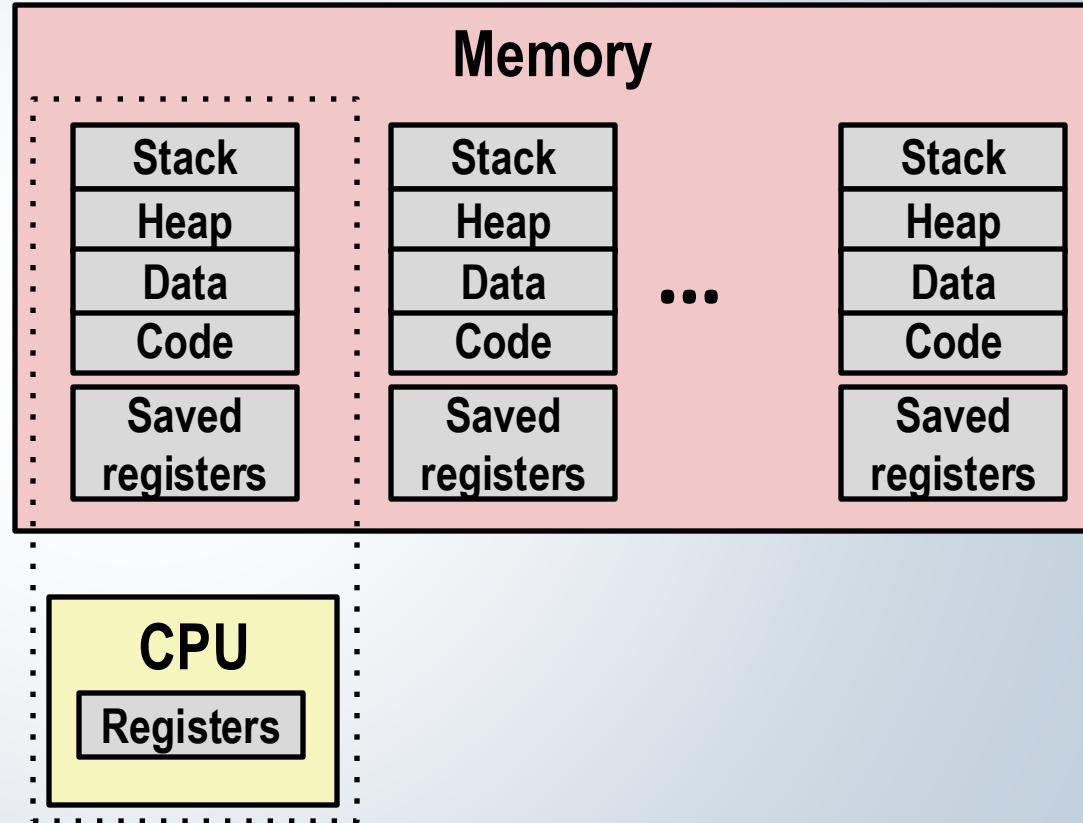
xterm
Processes: 123 total, 5 running, 9 stuck, 109 sleeping, 611 threads
Load Avg: 1.03, 1.13, 1.14 CPU usage: 3.27% user, 5.15% sys, 91.56% idle
SharedLibs: 576K resident, 0B data, 0B linkedit.
MemRegions: 27958 total, 1127M resident, 35M private, 494M shared.
PhysMem: 1039M wired, 1974M active, 1062M inactive, 4076M used, 18M free.
VM: 280G vsize, 1091M framework vsize, 23075213(1) pageins, 5843367(0) pageouts.
Networks: packets: 41046228/11G in, 66083096/77G out.
Disks: 17874391/349G read, 12847373/594G written.

PID  COMMAND    %CPU  TIME    #TH   #WQ   #PORT  #HREG  RPRVT  RSHRD  RSIZE  VPRVT  VSIZE
99217- Microsoft Of 0.0  02:28.34 4    1    202    418    21M    24M    21M    66M    763M
99051  usbmuxd    0.0  00:04.10 3    1    47     66    436K    216K    480K    60M    2422M
99006  iTunesHelper 0.0  00:01.23 2    1    55     78    728K    3124K   1124K   43M    2429M
84286  bash       0.0  00:00.11 1    0    20     24    224K    732K    484K    17M    2378M
84285  xterm      0.0  00:00.83 1    0    32     73    656K    872K    692K    9728K  2382M
55939- Microsoft Ex 0.3  21:58.97 10   3    360    954    16M    65M    46M    114M   1057M
54751  sleep      0.0  00:00.00 1    0    17     20    92K     212K    360K    9632K  2370M
54739  launchdadd 0.0  00:00.00 2    1    33     50    488K    220K    1736K   48M    2409M
54737  top        6.5  00:02.53 1/1  0    30     29    1416K   216K    2124K   17M    2378M
54719  automountd 0.0  00:00.02 7    1    53     64    860K    216K    2184K   53M    2413M
54701  ocspd      0.0  00:00.05 4    1    61     54    1268K   2644K   3132K   50M    2426M
54661  Grab       0.6  00:02.75 6    3    222+   389+   15M+    26M+    40M+    75M+   2556M+
54659  cookied    0.0  00:00.15 2    1    40     61    3316K   224K    4088K   42M    2411M
53818  mdworker   0.0  00:01.67 4    1    52     91    7628K   7412K   16M     48M    2438M
50878  mdworker   0.0  00:11.17 3    1    53     91    2464K   6148K   9976K   44M    2434M
50410  xterm      0.0  00:00.13 1    0    32     73    280K    872K    532K    9700K  2382M
50078  emacs      0.0  00:06.70 1    0    20     35    52K     216K    88K     18M    2392M

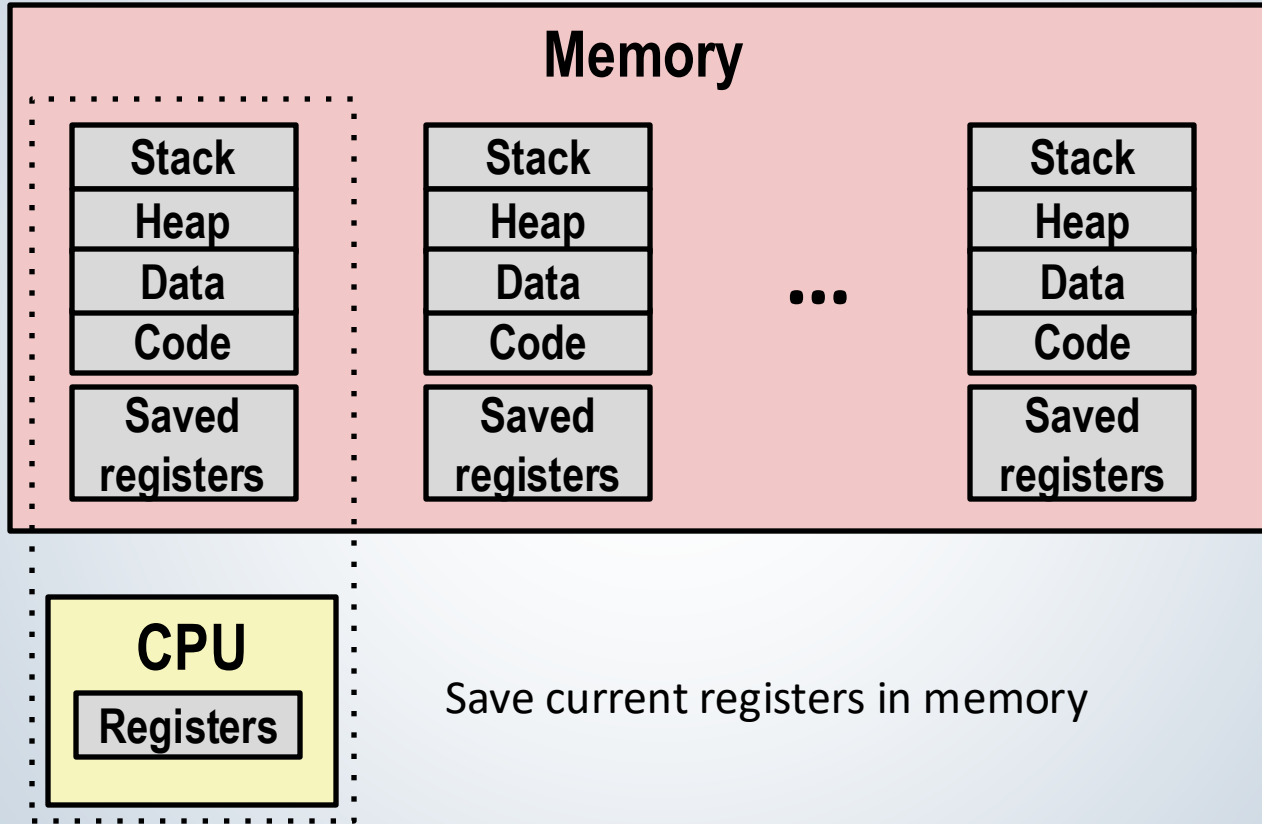
```


Multiprocessing: The (Traditional) Reality

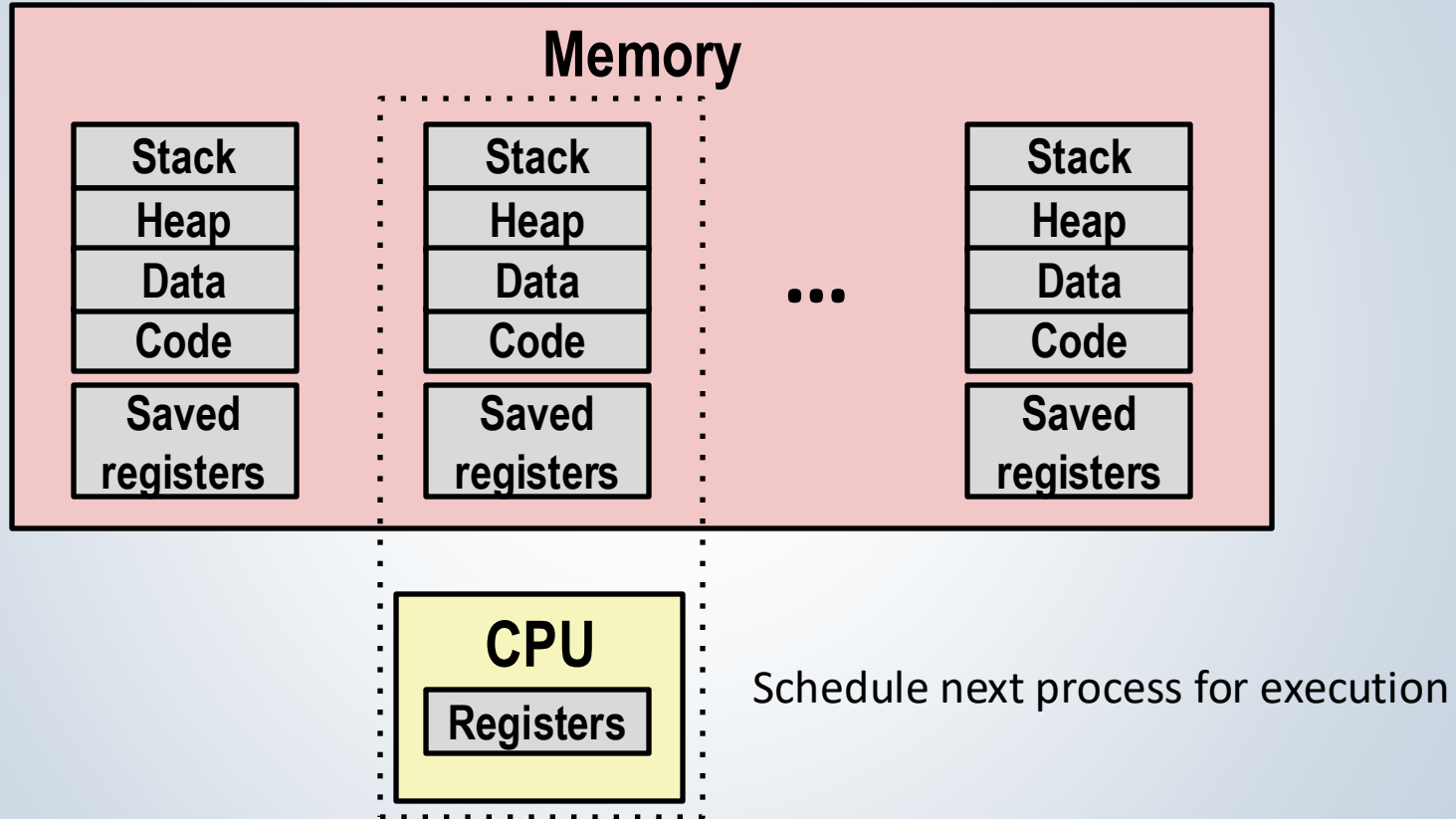
- Single processor executes multiple processes concurrently
 - Process executions interleaved (multitasking)
 - Address spaces managed by virtual memory system (later in course)
 - Register values for nonexecuting processes saved in memory



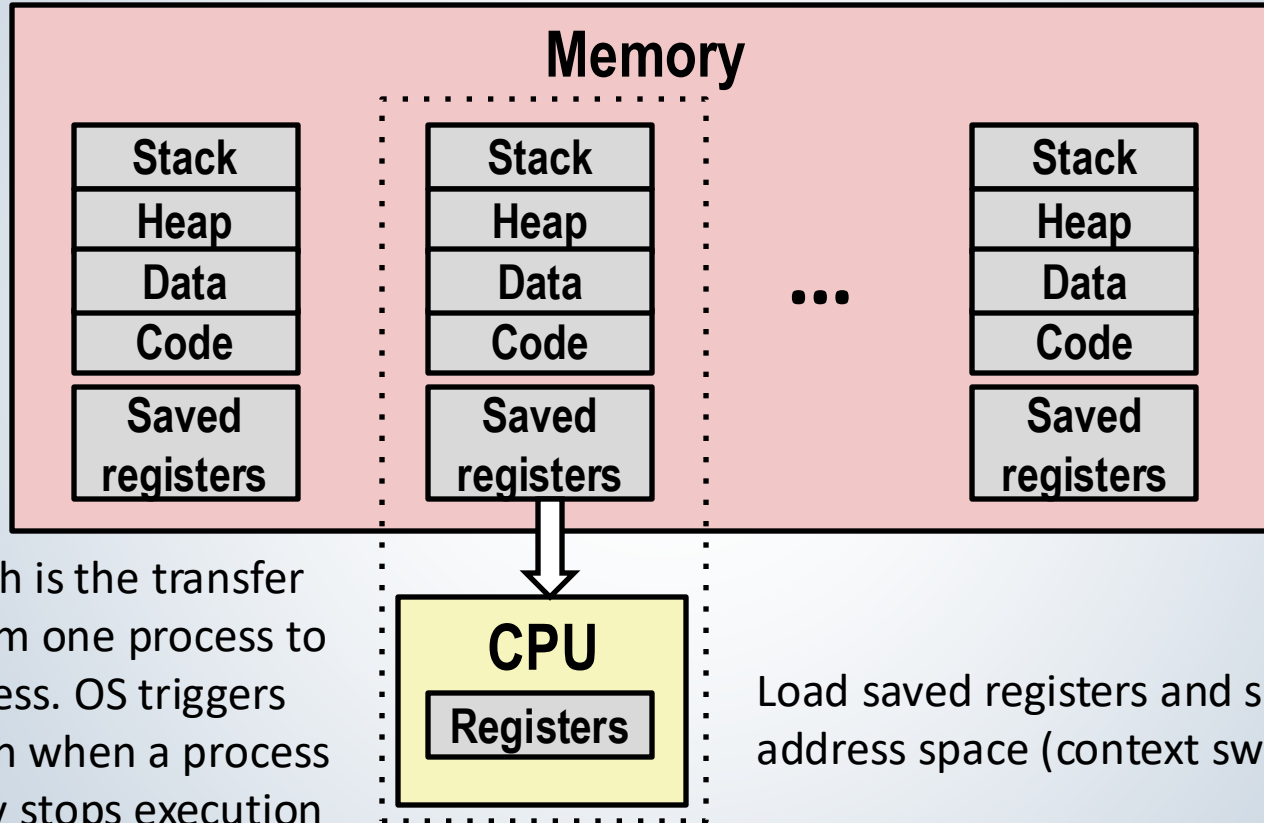
Multiprocessing: The (Traditional) Reality



Multiprocessing: The (Traditional) Reality



Multiprocessing: The (Traditional) Reality



Concurrent Processes

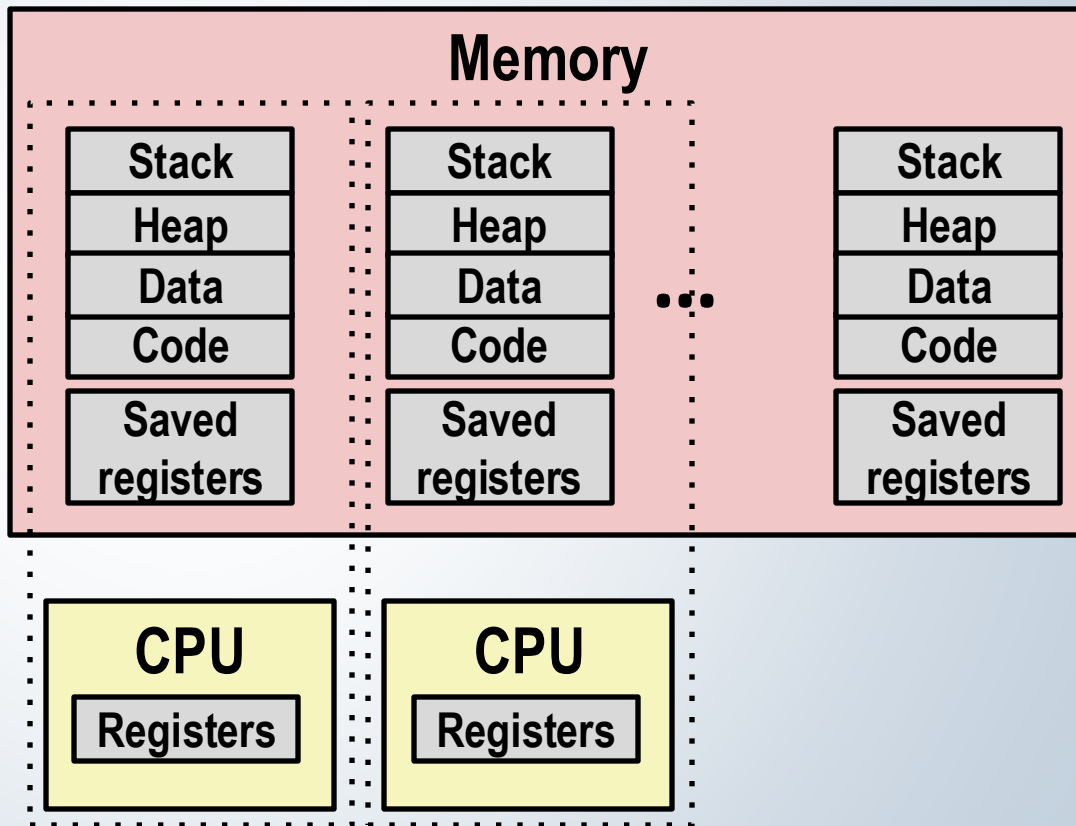
- Each process is a logical control flow
- Two processes *run concurrently* (are *concurrent*) if their flows overlap in time
- Otherwise, they are *sequential*
- Examples (running on single core):
 - Concurrent: A & B, A & C
 - Sequential: B & C



Multiprocessing: The (Modern) Reality

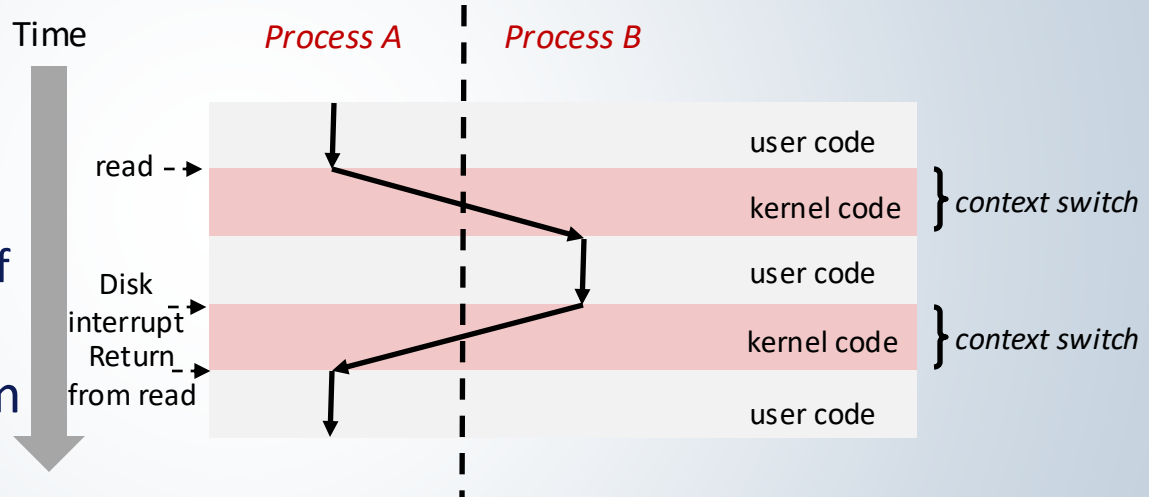
- Multicore processors

- Multiple CPUs on single chip
- Each can execute a separate process.
- Run multiple processes *simultaneously*
- Scheduling of processors onto cores done by kernel



Context Switching

- Processes are managed by a shared chunk of memory-resident OS code called the *kernel*
 - Important:** the kernel is not a separate process, but rather runs as part of some existing process
- Control flow passes from one process to another via a *context switch*



Summary

- Exceptions
 - Events that require nonstandard control flow
 - Generated externally (interrupts) or internally (traps and faults)
- Processes
 - At any given time, system has multiple active processes
 - Only one can execute at a time on a single core, though
 - Each process appears to have total control of processor + private memory space