CSC 252: Computer Organization
Spring 2019: Lecture 19

Instructor: John Criswell

Department of Computer Science
University of Rochester

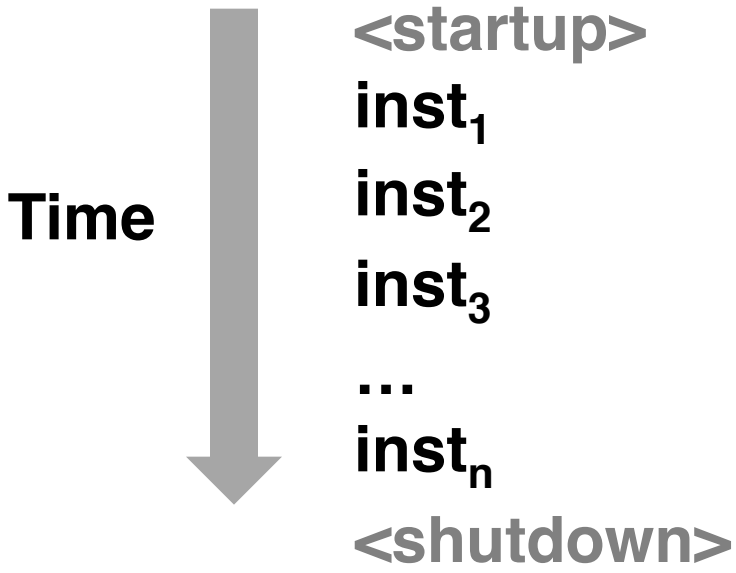
Action Items:

- **Programming Assignment 5 is out**
- **Trivia 5 is out**

So Far in CSC252…

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

Physical control flow



Altering the Control Flow

- Up to now: two mechanisms for changing control flow:
  - Jumps and branches
  - Call and return

  React to changes in ***program state***

Altering the Control Flow

- Up to now: two mechanisms for changing control flow:
  - Jumps and branches
  - Call and return

  React to changes in ***program state***

- Insufficient for a useful system: Difficult to react to changes in *system state*
  - Data arrives from a disk or a network adapter
  - Instruction divides by zero
  - User hits Ctrl-C at the keyboard
  - System timer expires

Altering the Control Flow

- Up to now: two mechanisms for changing control flow:
  - Jumps and branches
  - Call and return

  React to changes in ***program state***

- Insufficient for a useful system: Difficult to react to changes in *system state*
  - Data arrives from a disk or a network adapter
  - Instruction divides by zero
  - User hits Ctrl-C at the keyboard
  - System timer expires

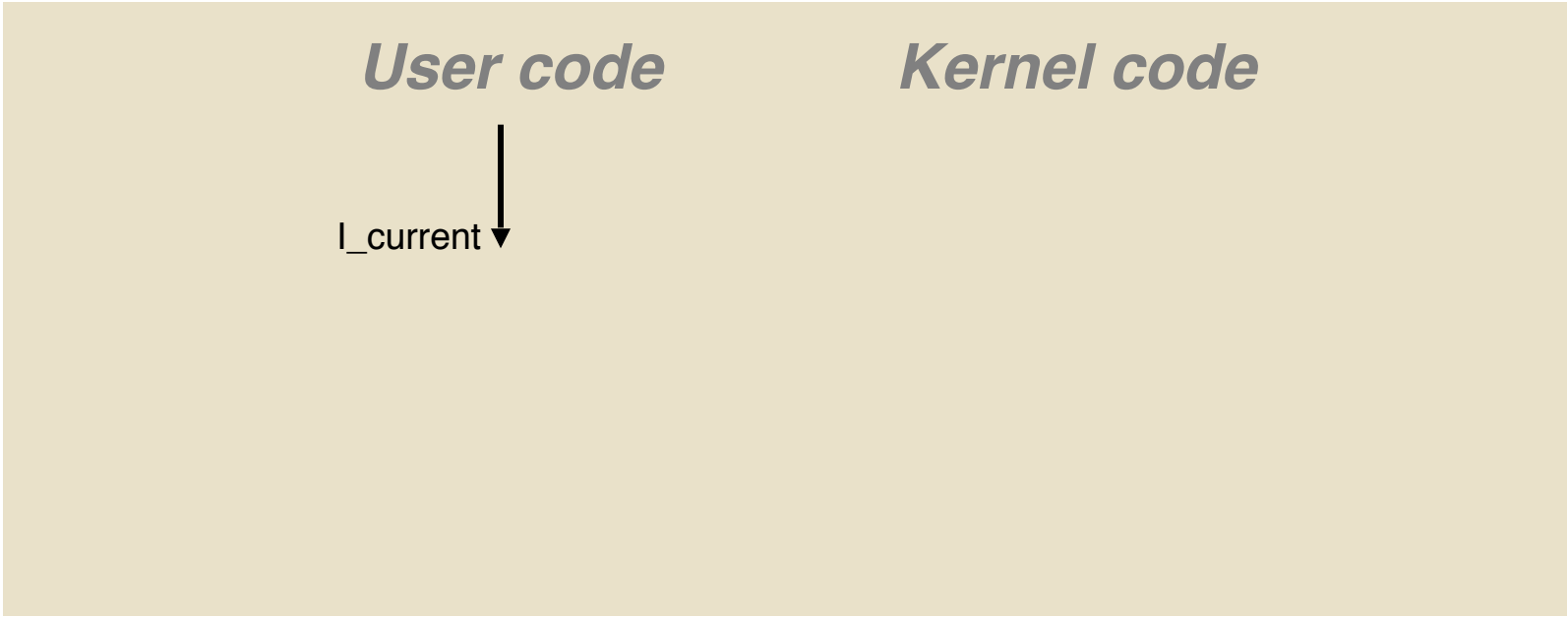
- System needs mechanisms for “exceptional control flow”

Today

- Exceptions/Interrupts

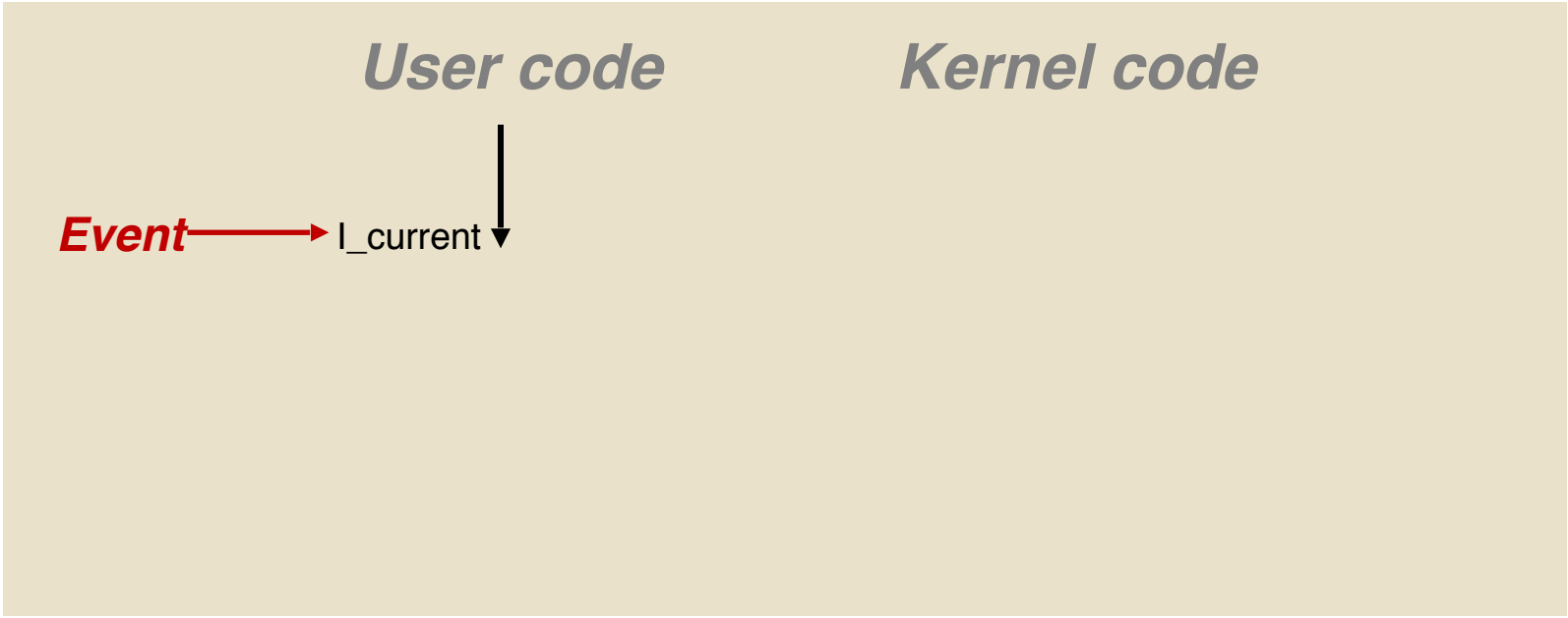
Exceptions

- An *exception* is a transfer of control to the OS *kernel* in response to some *event* (i.e., change in processor state)
  - Kernel is the memory-resident part of the OS
  - Examples of events: Divide by 0, arithmetic overflow, page fault, I/O request completes, typing Ctrl-C



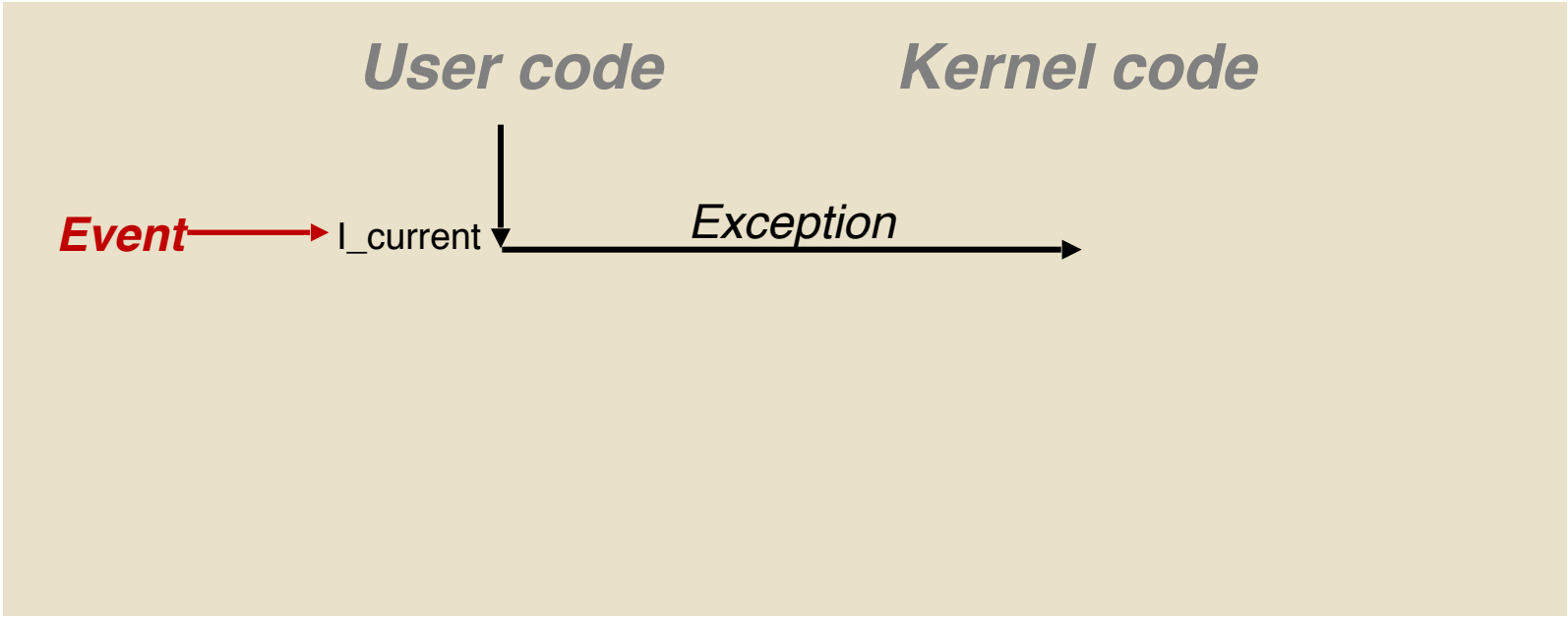
Exceptions

- An *exception* is a transfer of control to the OS *kernel* in response to some *event* (i.e., change in processor state)
  - Kernel is the memory-resident part of the OS
  - Examples of events: Divide by 0, arithmetic overflow, page fault, I/O request completes, typing Ctrl-C



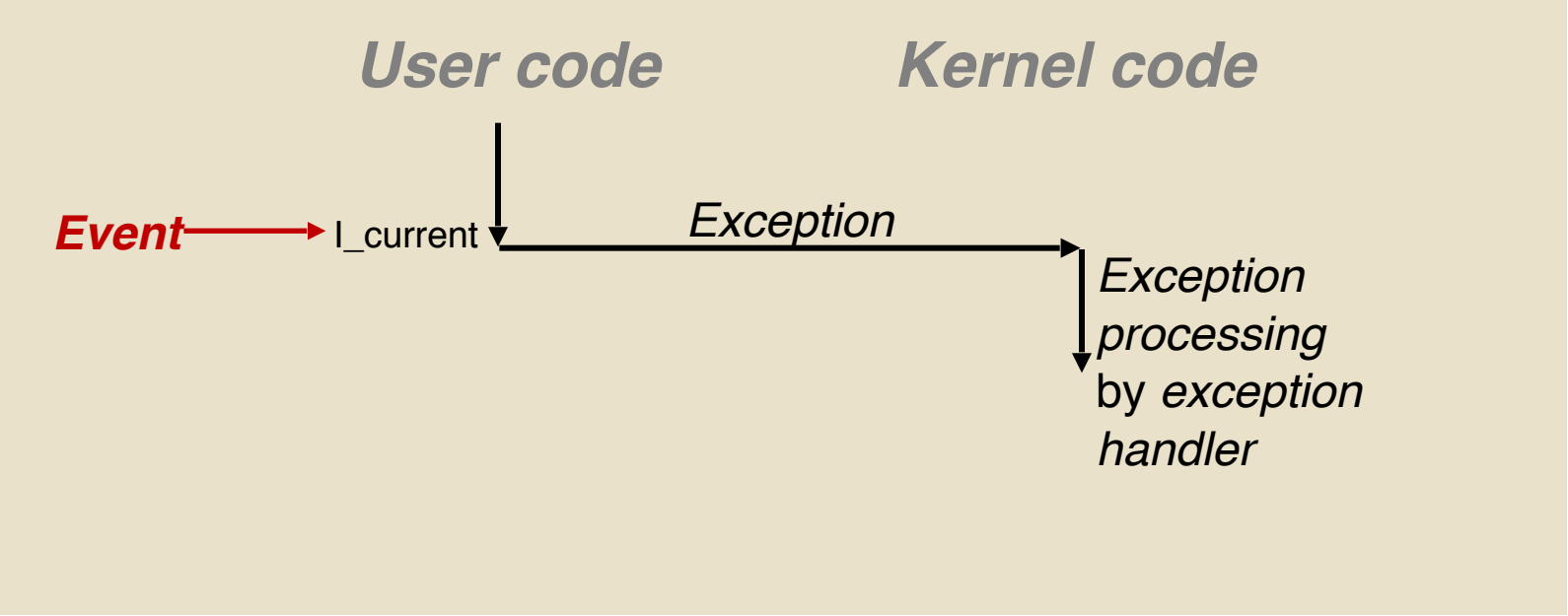
Exceptions

- An *exception* is a transfer of control to the OS *kernel* in response to some *event* (i.e., change in processor state)
  - Kernel is the memory-resident part of the OS
  - Examples of events: Divide by 0, arithmetic overflow, page fault, I/O request completes, typing Ctrl-C



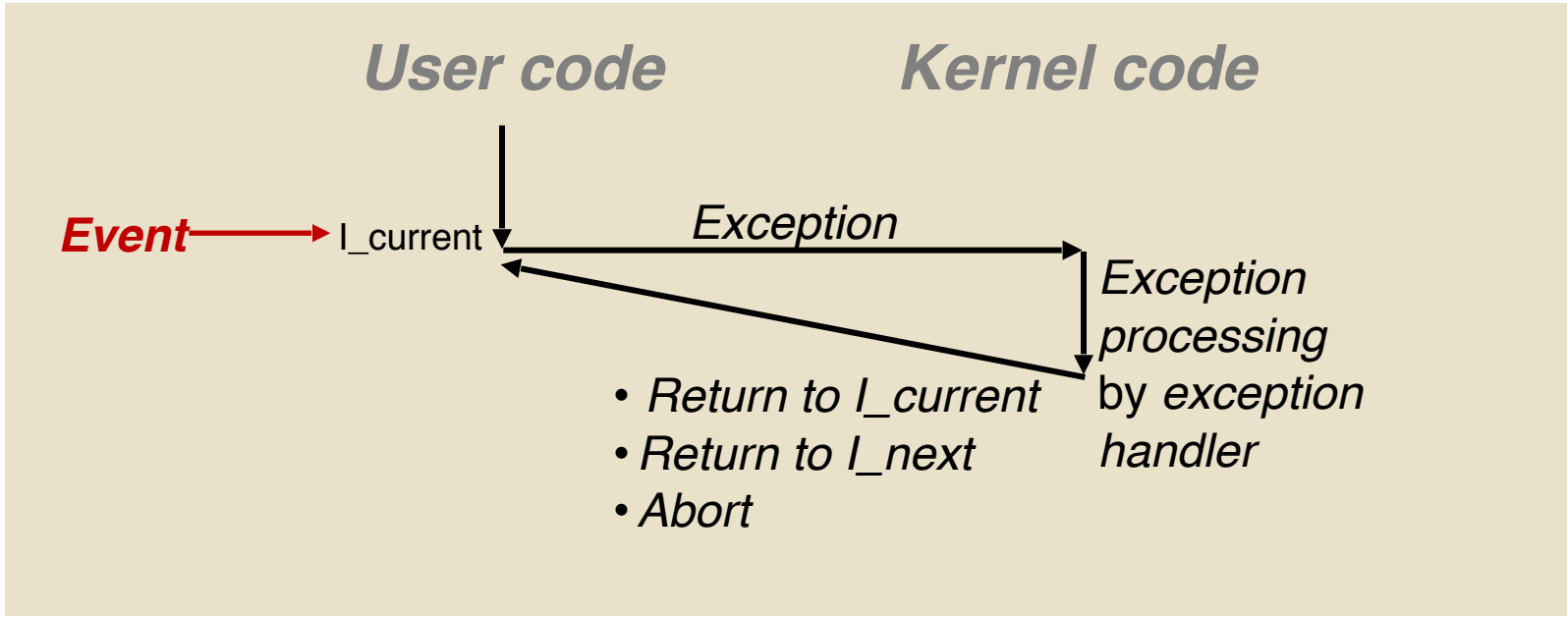
Exceptions

- An *exception* is a transfer of control to the OS *kernel* in response to some *event* (i.e., change in processor state)
  - Kernel is the memory-resident part of the OS
  - Examples of events: Divide by 0, arithmetic overflow, page fault, I/O request completes, typing Ctrl-C



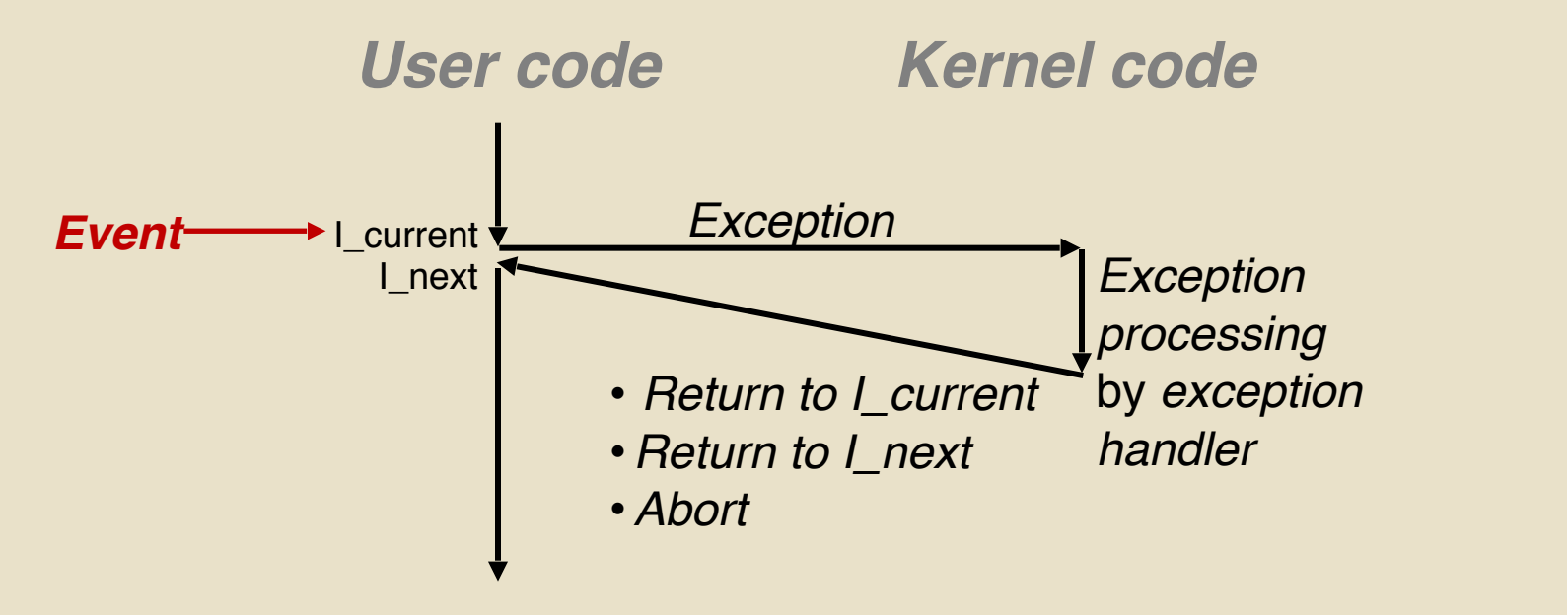
Exceptions

- An *exception* is a transfer of control to the OS *kernel* in response to some *event* (i.e., change in processor state)
  - Kernel is the memory-resident part of the OS
  - Examples of events: Divide by 0, arithmetic overflow, page fault, I/O request completes, typing Ctrl-C



Exceptions

- An *exception* is a transfer of control to the OS *kernel* in response to some *event* (i.e., change in processor state)
  - Kernel is the memory-resident part of the OS
  - Examples of events: Divide by 0, arithmetic overflow, page fault, I/O request completes, typing Ctrl-C



Asynchronous Exceptions (Interrupts)

- Caused by events external to the processor
  - Events that can happen at any time. Computers have little control.
  - Indicated by setting the processor's interrupt pin
  - Handler returns to "next" instruction

Asynchronous Exceptions (Interrupts)

- Caused by events external to the processor
  - Events that can happen at any time. Computers have little control.
  - Indicated by setting the processor's interrupt pin
  - Handler returns to "next" instruction

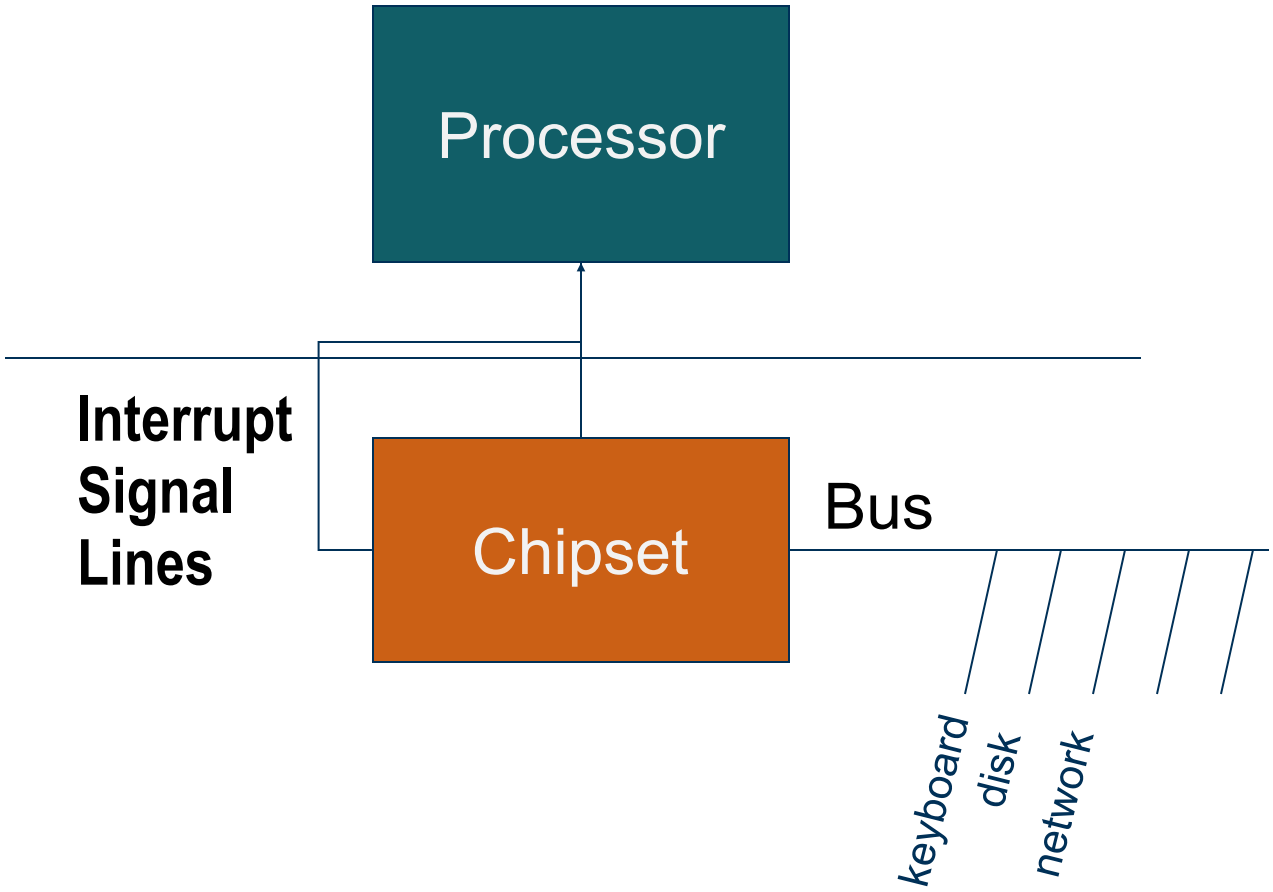
- Examples:
  - Timer interrupt
    - Every few ms, an external timer chip triggers an interrupt

Asynchronous Exceptions (Interrupts)

- Caused by events external to the processor
  - Events that can happen at any time. Computers have little control.
  - Indicated by setting the processor's interrupt pin
  - Handler returns to "next" instruction

- Examples:
  - Timer interrupt
    - Every few ms, an external timer chip triggers an interrupt
    - Used by the kernel to take back control from user programs
  - I/O interrupt from external device
    - Hitting Ctrl-C at the keyboard
    - Arrival of a packet from a network
    - Arrival of data from a disk

Interrupts in a Processor



Synchronous Exceptions

- Caused by events that occur as a result of executing an instruction:

Synchronous Exceptions

- Caused by events that occur as a result of executing an instruction:
  - *Traps*
    - Intentional
    - Examples: *system calls*, breakpoint traps, special instructions
    - Returns control to “next” instruction

Synchronous Exceptions

- Caused by events that occur as a result of executing an instruction:
  - *Traps*
    - Intentional
    - Examples: *system calls*, breakpoint traps, special instructions
    - Returns control to “next” instruction
  - *Faults*
    - Unintentional but possibly recoverable
    - Examples: page faults (recoverable), ***protection faults (the infamous Segmentation Fault!)*** (unrecoverable in Linux), floating point exceptions (unrecoverable in Linux)
    - Either re-executes faulting (“current”) instruction or aborts

Synchronous Exceptions

- Caused by events that occur as a result of executing an instruction:
  - *Traps*
    - Intentional
    - Examples: *system calls*, breakpoint traps, special instructions
    - Returns control to “next” instruction
  - *Faults*
    - Unintentional but possibly recoverable
    - Examples: page faults (recoverable), ***protection faults (the infamous Segmentation Fault!)*** (unrecoverable in Linux), floating point exceptions (unrecoverable in Linux)
    - Either re-executes faulting (“current”) instruction or aborts
  - *Aborts*
    - Unintentional and unrecoverable
    - Examples: parity error, machine check
    - Aborts current program

Fault Example: Page Fault

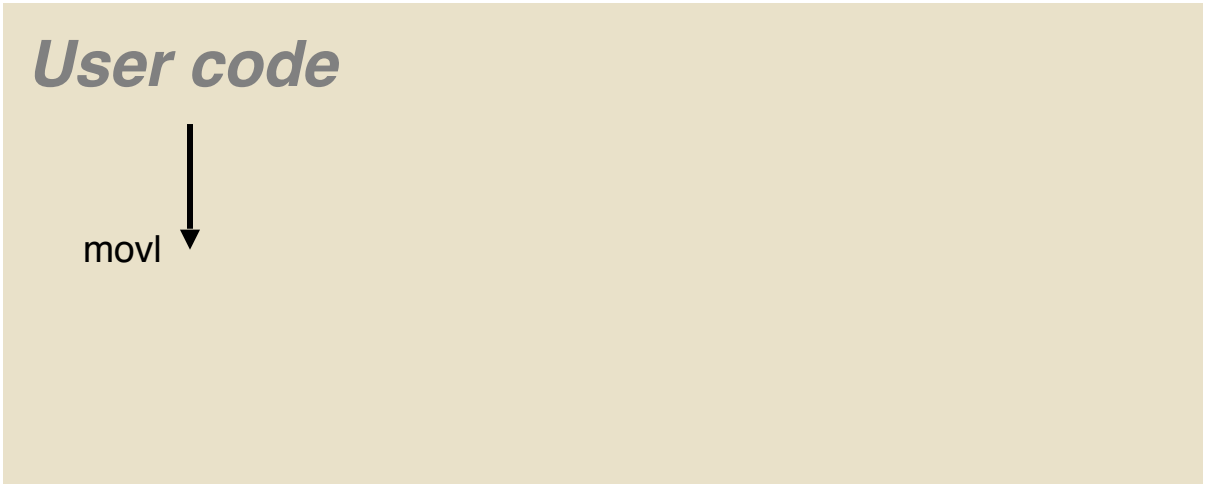
- User writes to memory location
- That memory location is not found in memory because it is currently on disk
- Trigger a Page Fault (recoverable), the exception handler loads the data from disk to memory (will discuss in detail later in the class)

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

Fault Example: Page Fault

- User writes to memory location
- That memory location is not found in memory because it is currently on disk
- Trigger a Page Fault (recoverable), the exception handler loads the data from disk to memory (will discuss in detail later in the class)

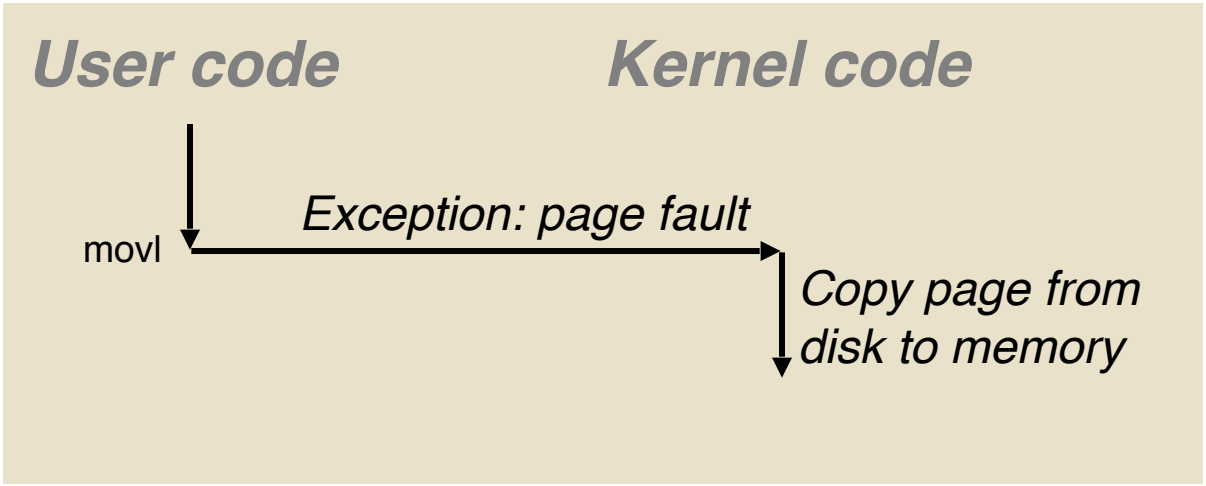
```
80483b7:      c7 05 10 9d 04 08 0d   movl   $0xd,0x8049d10
```



Fault Example: Page Fault

- User writes to memory location
- That memory location is not found in memory because it is currently on disk
- Trigger a Page Fault (recoverable), the exception handler loads the data from disk to memory (will discuss in detail later in the class)

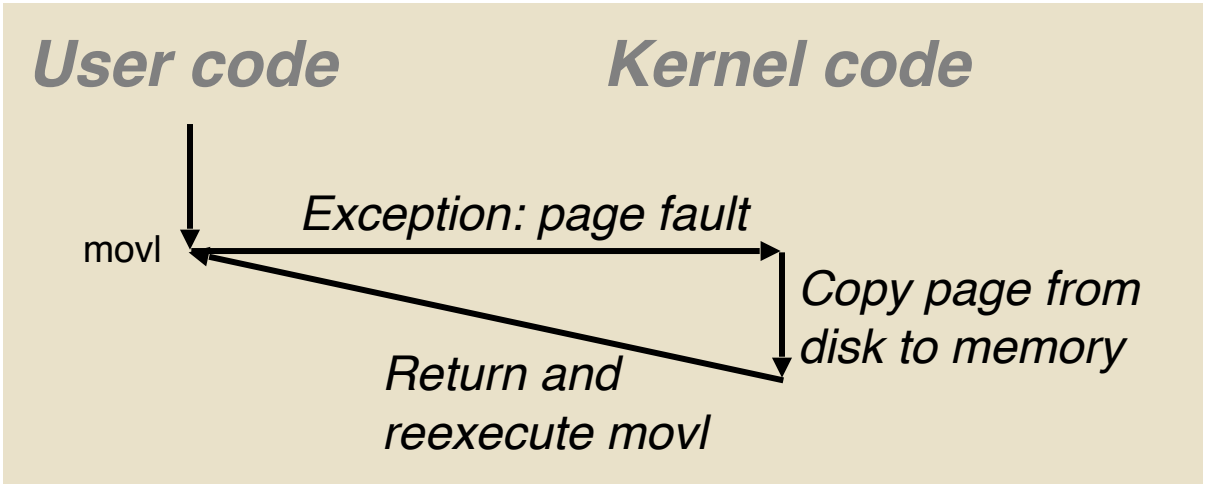
```
80483b7:      c7 05 10 9d 04 08 0d   movl   $0xd,0x8049d10
```



Fault Example: Page Fault

- User writes to memory location
- That memory location is not found in memory because it is currently on disk
- Trigger a Page Fault (recoverable), the exception handler loads the data from disk to memory (will discuss in detail later in the class)

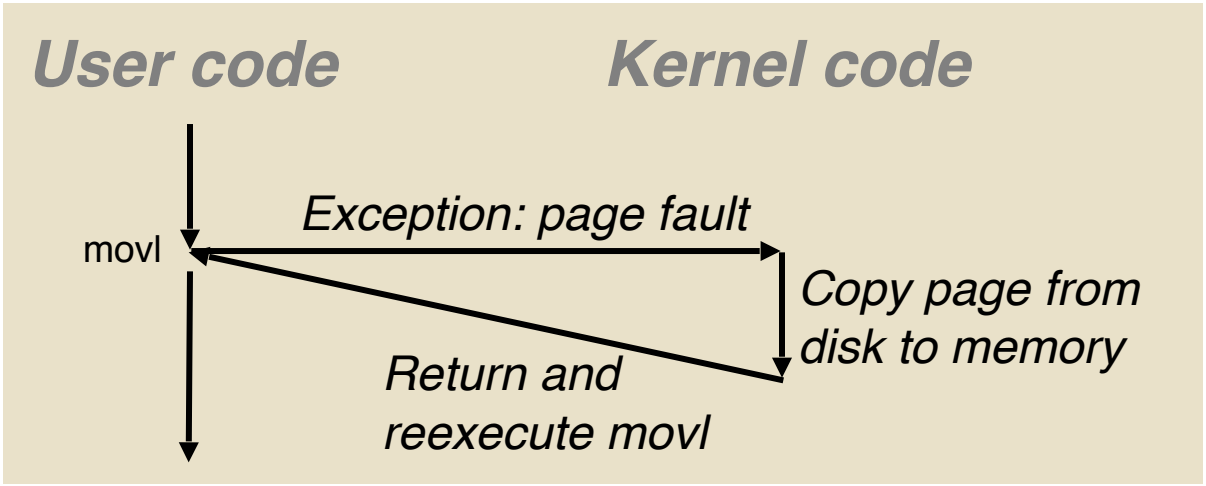
```
80483b7:      c7 05 10 9d 04 08 0d   movl   $0xd,0x8049d10
```



Fault Example: Page Fault

- User writes to memory location
- That memory location is not found in memory because it is currently on disk
- Trigger a Page Fault (recoverable), the exception handler loads the data from disk to memory (will discuss in detail later in the class)

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

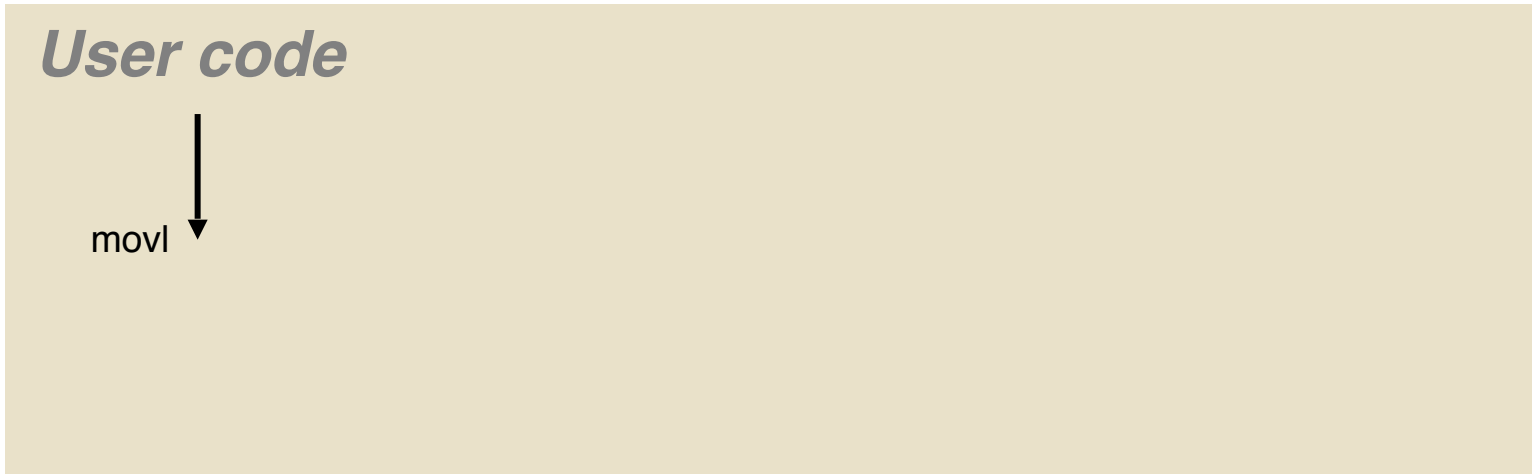


Fault Example: Protection Fault

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

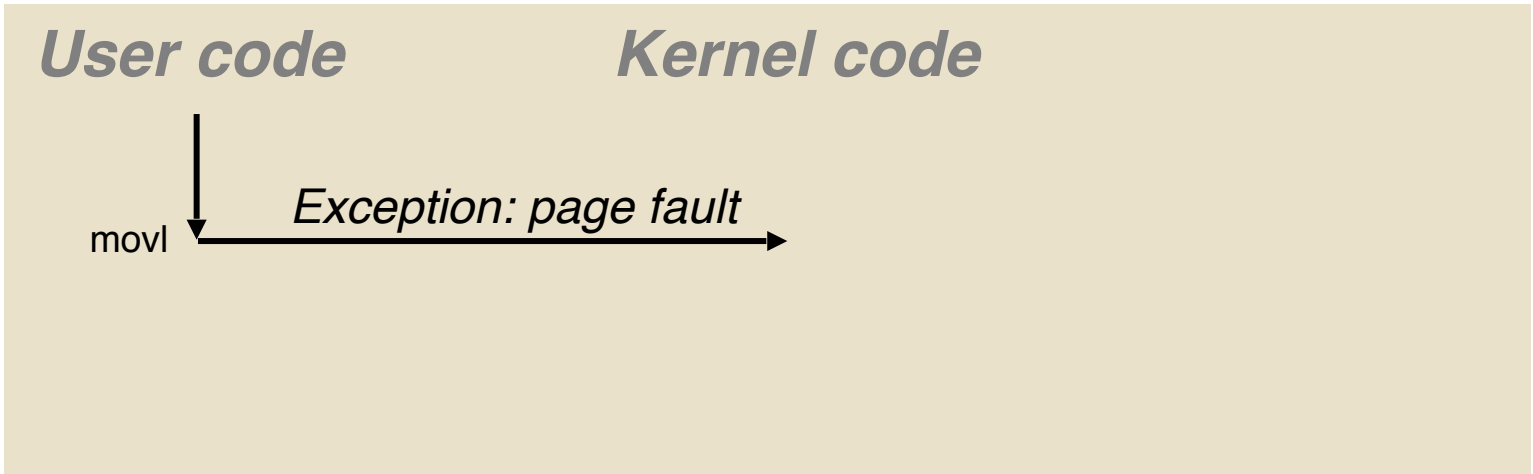
Fault Example: Protection Fault

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



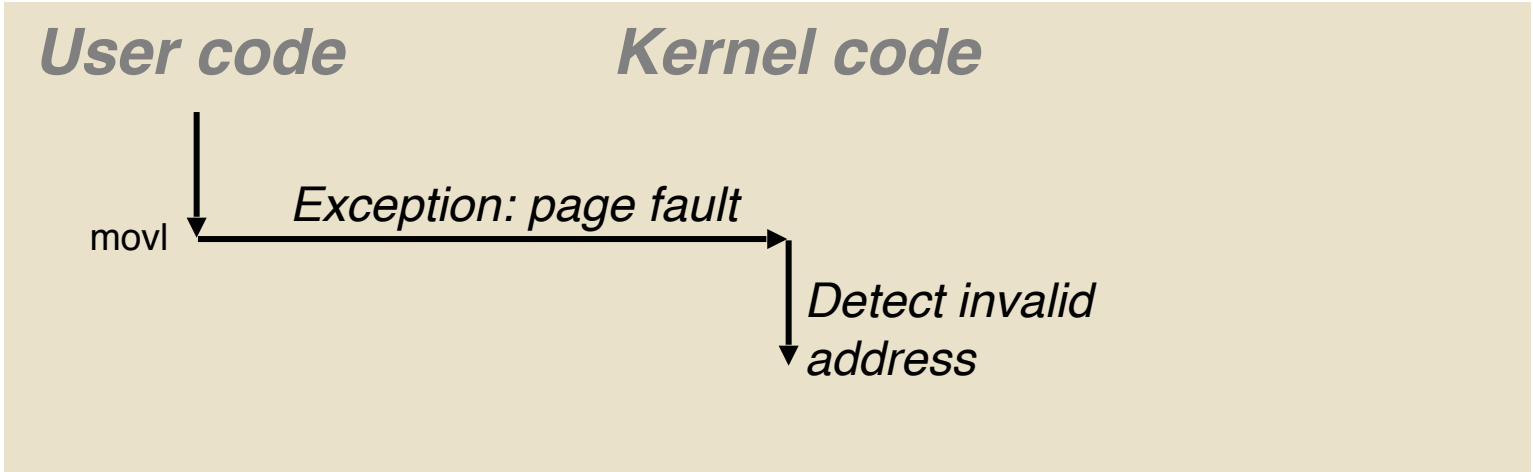
Fault Example: Protection Fault

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



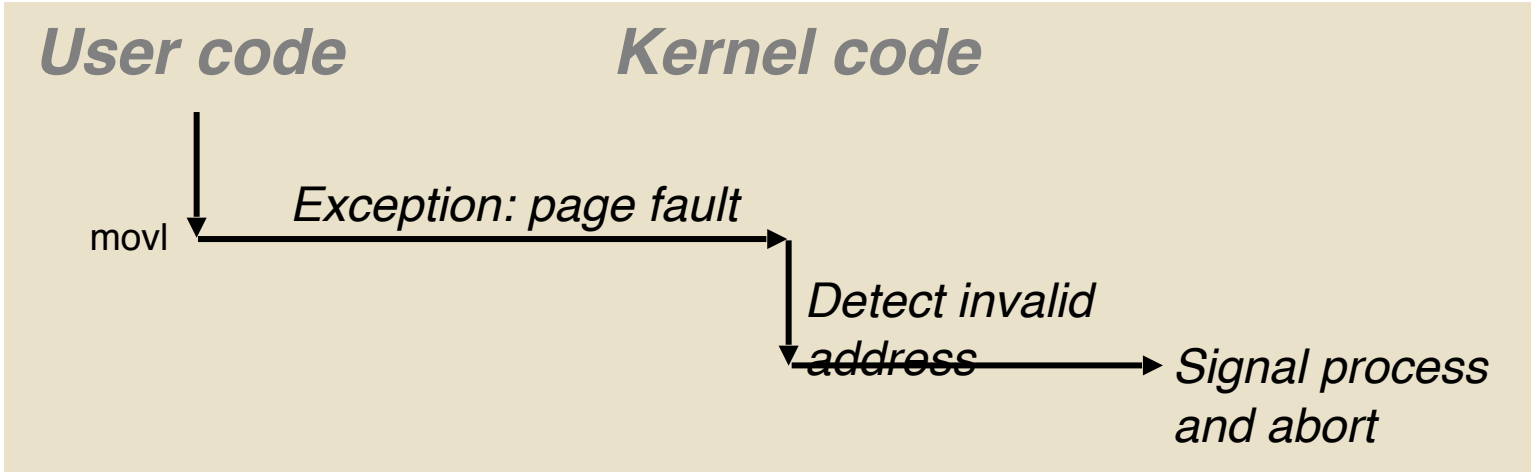
Fault Example: Protection Fault

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



Fault Example: Protection Fault

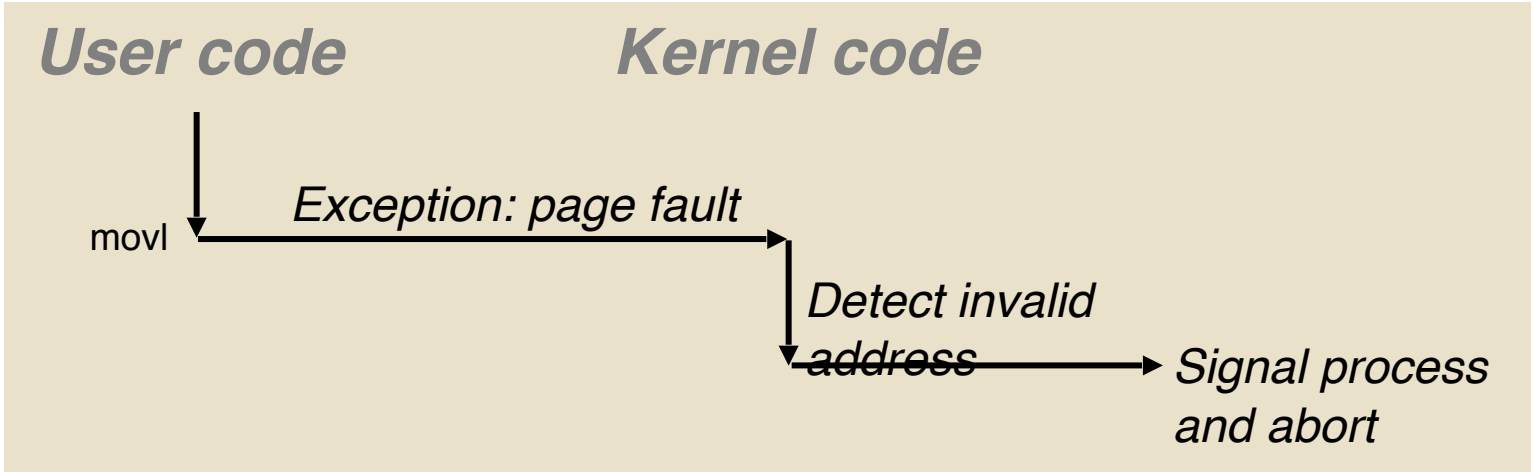
```
80483b7:      c7 05 60 e3 04 08 0d   movl   $0xd,0x804e360
```



Fault Example: Protection Fault

- Access illegal memory location (e.g., dereferencing a null pointer)

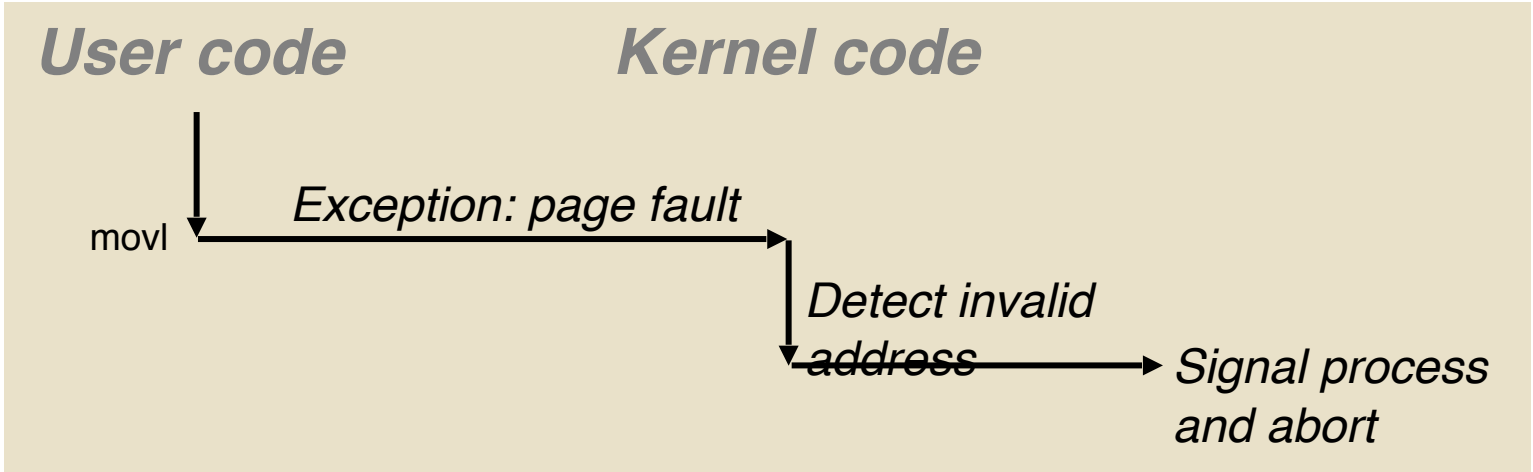
```
80483b7:      c7 05 60 e3 04 08 0d   movl   $0xd,0x804e360
```



Fault Example: Protection Fault

- Access illegal memory location (e.g., dereferencing a null pointer)
- First trigger a Page Fault, the exception handler decides that this is unrecoverable, so simply aborts

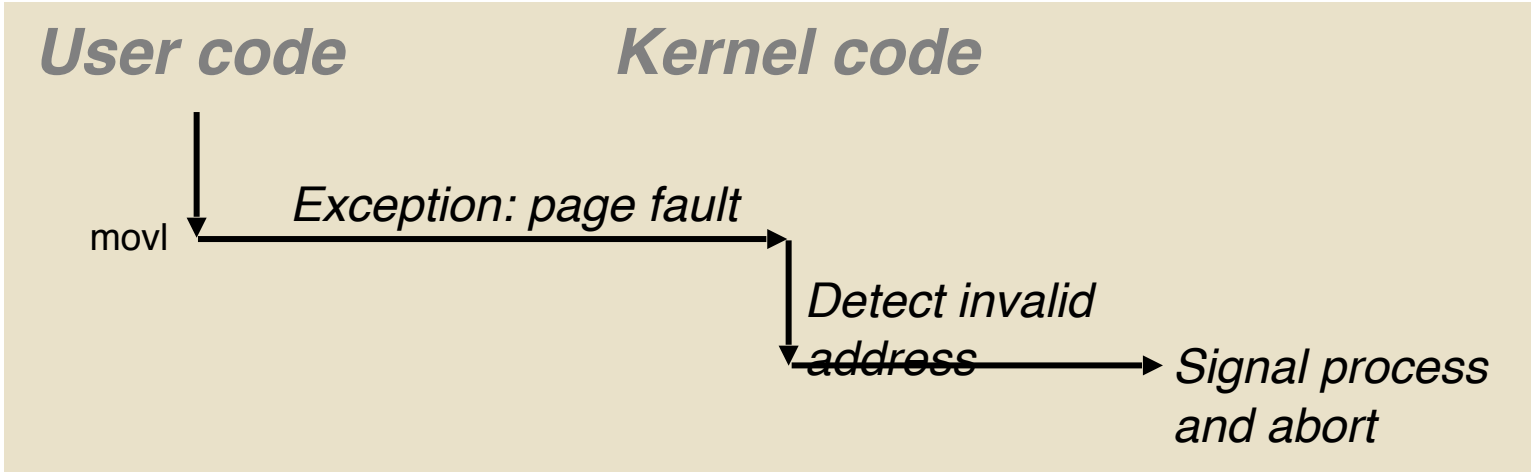
```
80483b7:      c7 05 60 e3 04 08 0d   movl   $0xd,0x804e360
```



Fault Example: Protection Fault

- Access illegal memory location (e.g., dereferencing a null pointer)
- First trigger a Page Fault, the exception handler decides that this is unrecoverable, so simply aborts
- User process exits with “segmentation fault”

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



Others' Definitions

- The textbook's definitions are not universally accepted
- Intel (http://www.intel.com/cd/ids/developer/asmo-na/eng/microprocessors/ia32/xeon/19250.htm?page=2)
  - **Interrupt**: An exception that comes from outside of the processor. There are two kinds of exceptions: local and external. A local exception is generated from a program. External exceptions are usually generated by external I/O devices and received at exception pins.
- PowerPC Architecture
  - Interrupts "allow the processor to change state as a result of external signals, errors, or unusual conditions arising in the execution of instructions"
- PowerPC 604
  - Everything is an exception
- Motorola 68K
  - Everything is an exception
- VAX
  - Interrupts: device, software, urgent
  - Exceptions: faults, traps, aborts

When Do You Call the Handler?

- Interrupts: when convenient. Typically wait until the current instructions in the pipeline are finished
- Exceptions: typically immediately as programs can't continue without resolving the exception (think of page fault)
- Maskable verses Unmaskable
  - Interrupts can be individually masked (i.e., ignored by CPU)
  - Synchronous exceptions are usually unmaskable
- Some interrupts are intentionally unmaskable
  - Called non-maskable interrupts (NMI)
  - Indicating a critical error has occurred, and that the system is probably about to crash

Where Do You Restart?

- Interrupts/Traps

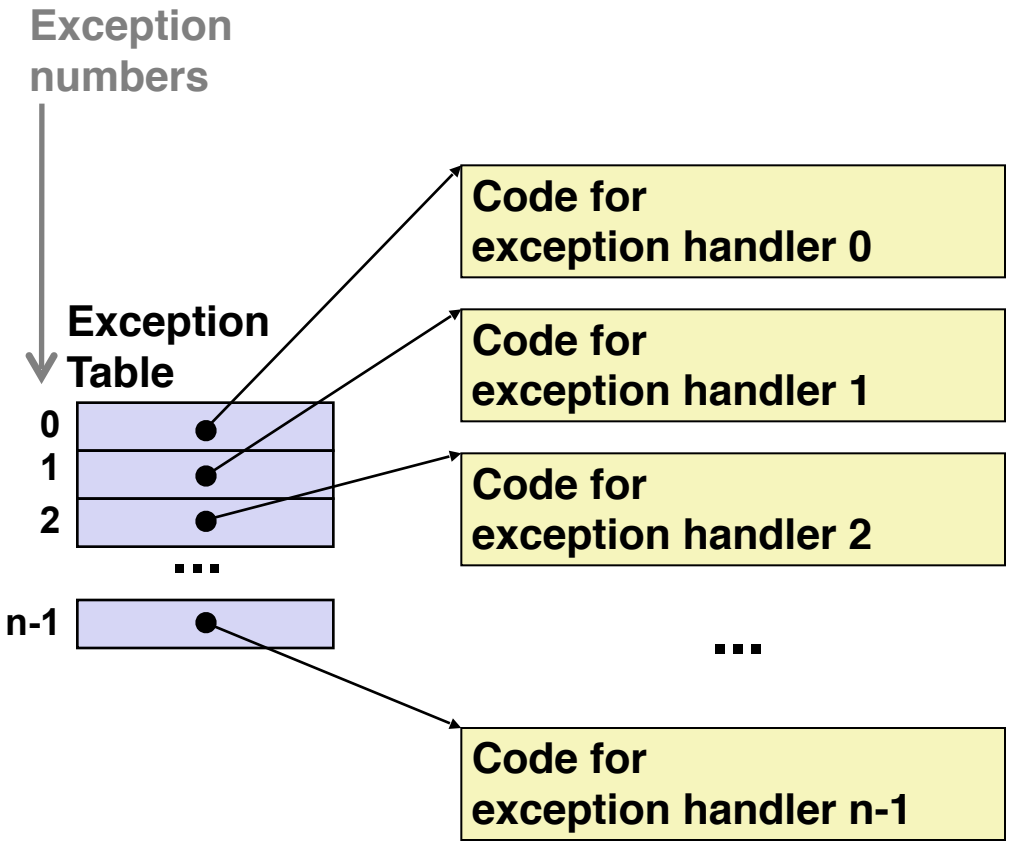
Where Do You Restart?

- Interrupts/Traps
  - Handler returns to the ***following*** instruction

Where Do You Restart?

- Interrupts/Traps
  - Handler returns to the ***following*** instruction
- Faults
  - Exception handler returns to the instruction that caused the exception, i.e., ***re-execute*** it!
- Aborts
  - Never returns to the program

Where to Find Exception Handlers?



- Each type of event has a unique exception number k
- k = index into exception table
- Exception table lives in memory. Its start address is stored in a special register
- Handler k is called each time exception k occurs

Nested Exceptions

- One interrupt/exception occurs when another is already active
- Priority maintained

- Can fundamentally do it
  - Subroutine calls within subroutine calls
  - Handlers need to save appropriate state

Concurrent Interrupts

- More than one interrupts happen at the same time
- Pre-defined priority
- The chipset arbitrates which one to respond to first