CSC 252: Computer Organization
Spring 2018: Lecture 18

Instructor: Yuhao Zhu

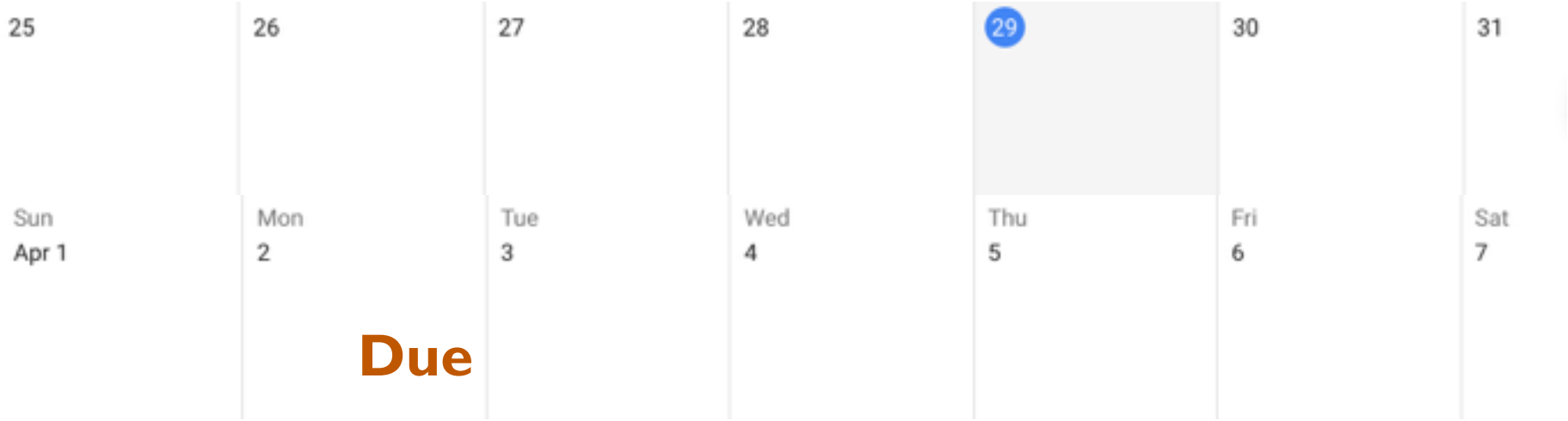
Department of Computer Science
University of Rochester

Action Items:

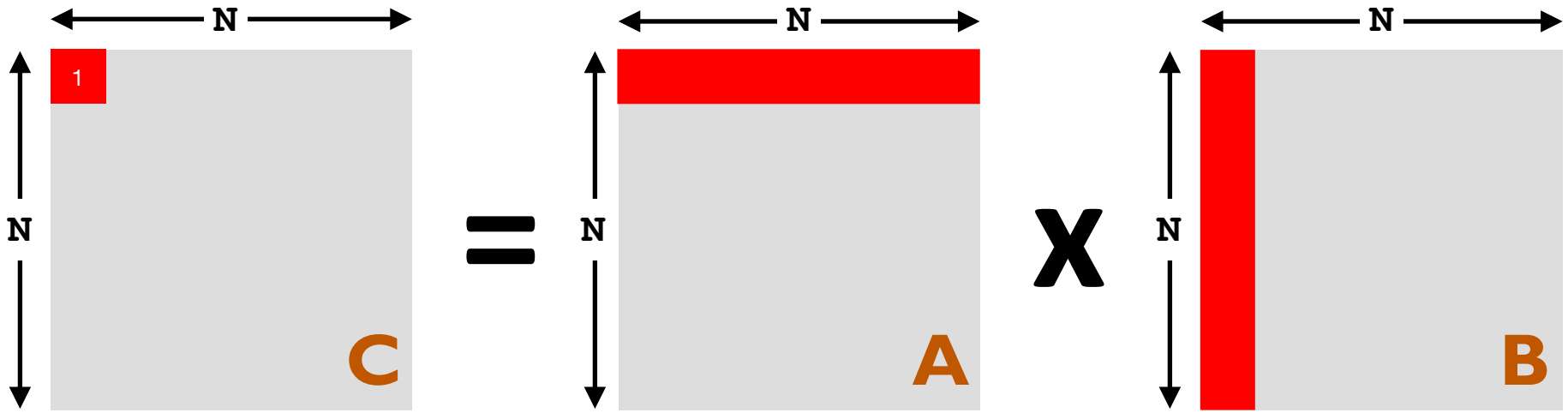
- **Programming Assignment 4 is due soon**
- **Get your exam after the class**
- **Take a look at the Cache Problem Set**

Announcement

- Programming Assignment 4 is due soon
  - 11:59pm, **Monday, April 2**.
- Cache Problem Set
  - No turn-in required. Solutions are released also
  - Get a preview of final exam problems!!!



Matrix Multiplication Example

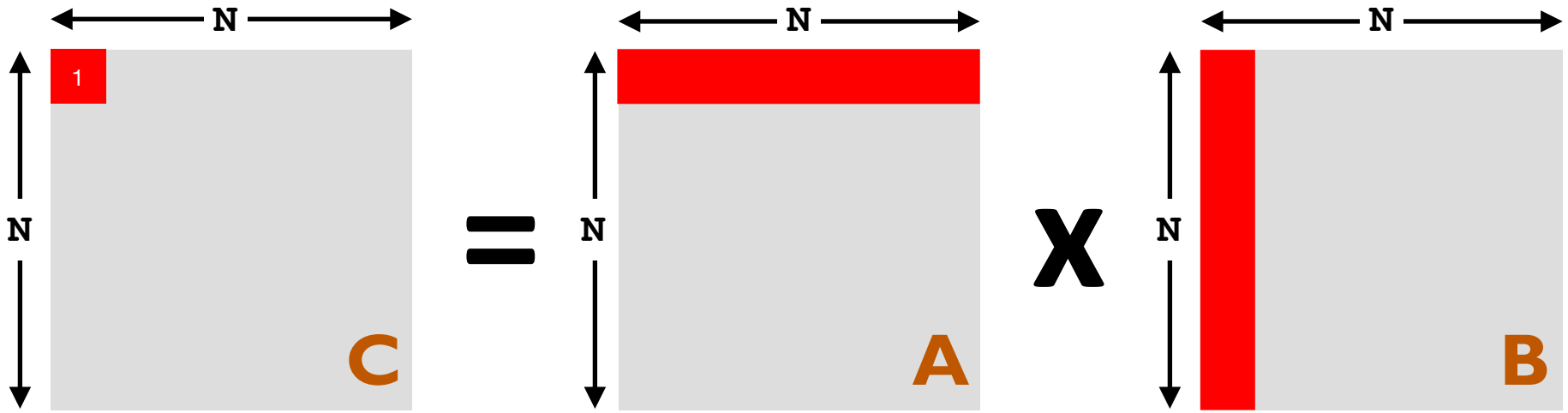


Assume each matrix element is 8 bytes, cache line is 16 bytes, N = 1024

Cache Misses

Iteration	A	B	C
1			
2			
3			
4			
Total			

Matrix Multiplication Example

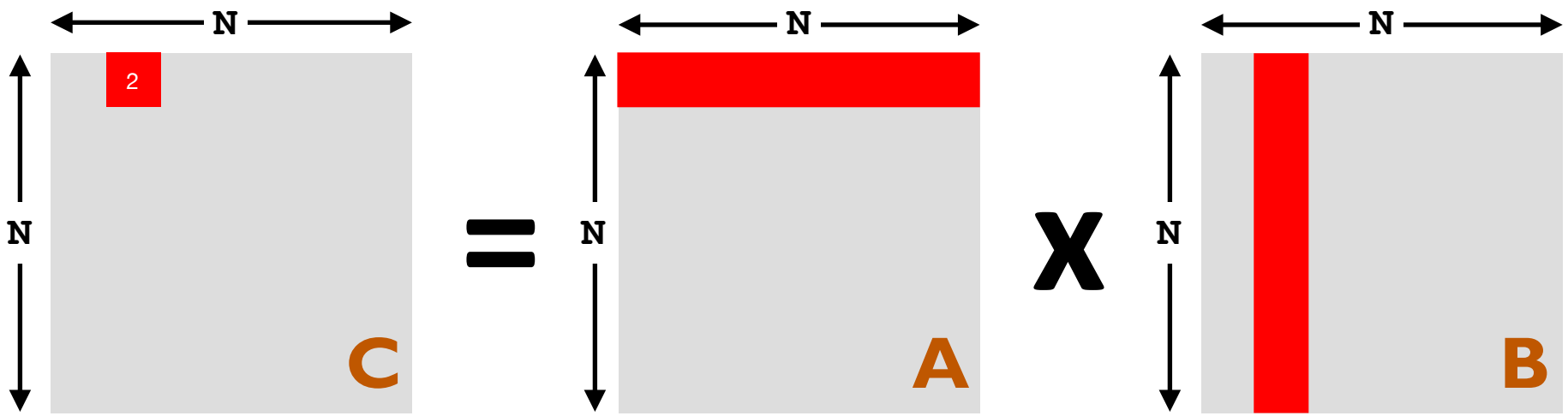


Assume each matrix element is 8 bytes, cache line is 16 bytes, N = 1024

Cache Misses

Iteration	A	B	C
1	512	1024	1
2			
3			
4			
Total			

Matrix Multiplication Example

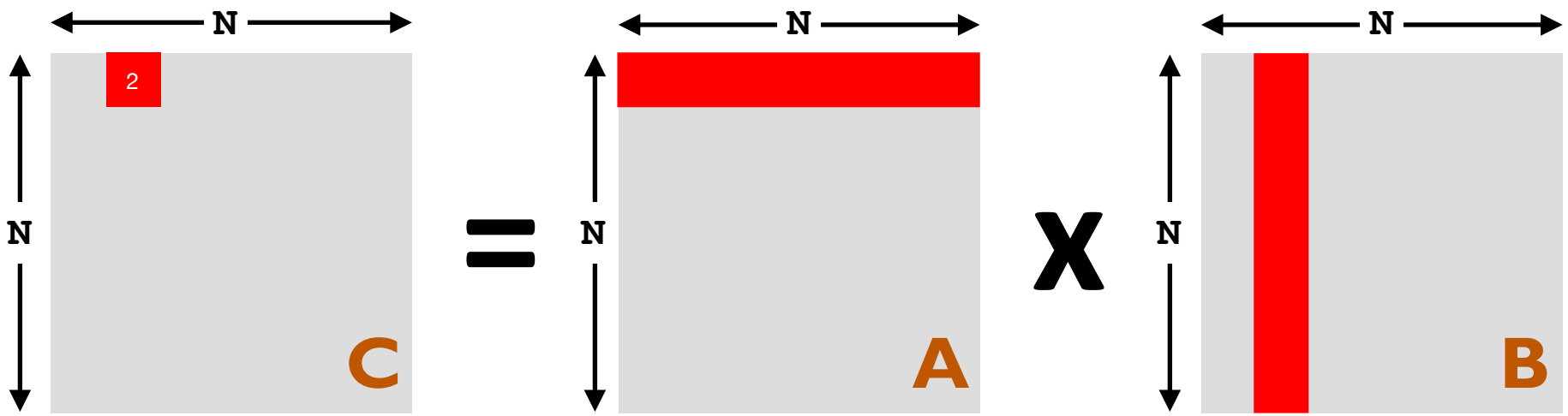


Assume each matrix element is 8 bytes, cache line is 16 bytes, N = 1024

Cache Misses

Iteration	A	B	C
1	512	1024	1
2			
3			
4			
Total			

Matrix Multiplication Example

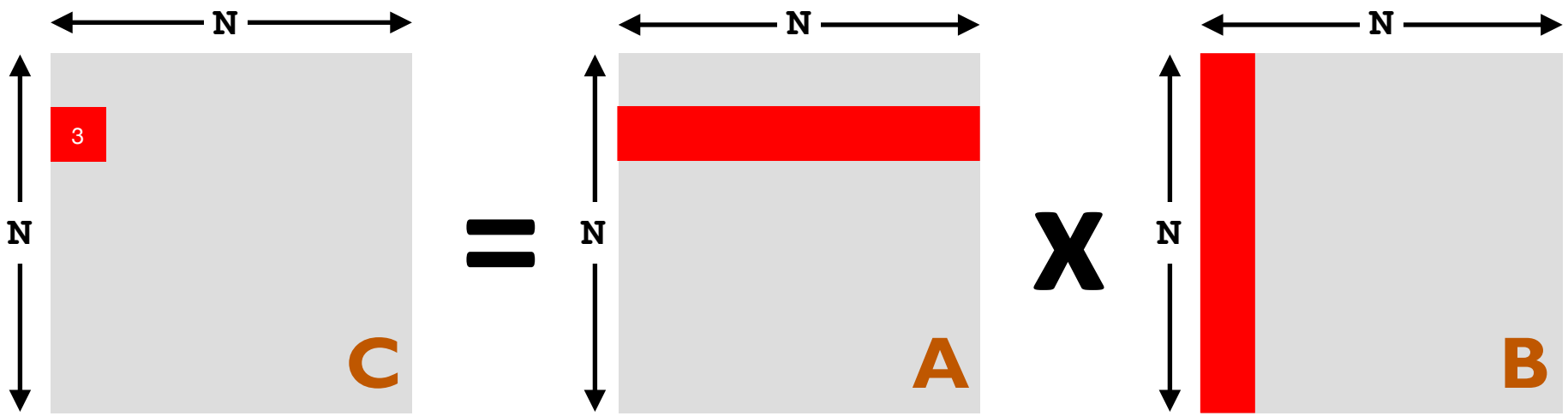


Assume each matrix element is 8 bytes, cache line is 16 bytes, N = 1024

Cache Misses

Iteration	A	B	C
1	512	1024	1
2	512	1024	0
3			
4			
Total			

Matrix Multiplication Example

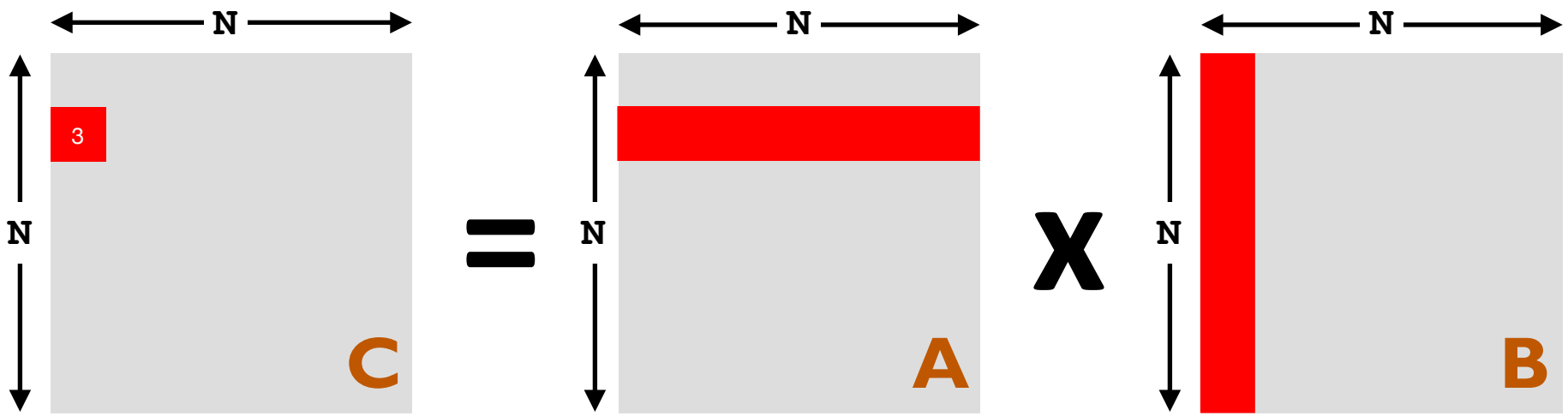


Assume each matrix element is 8 bytes, cache line is 16 bytes, N = 1024

Cache Misses

Iteration	A	B	C
1	512	1024	1
2	512	1024	0
3			
4			
Total			

Matrix Multiplication Example

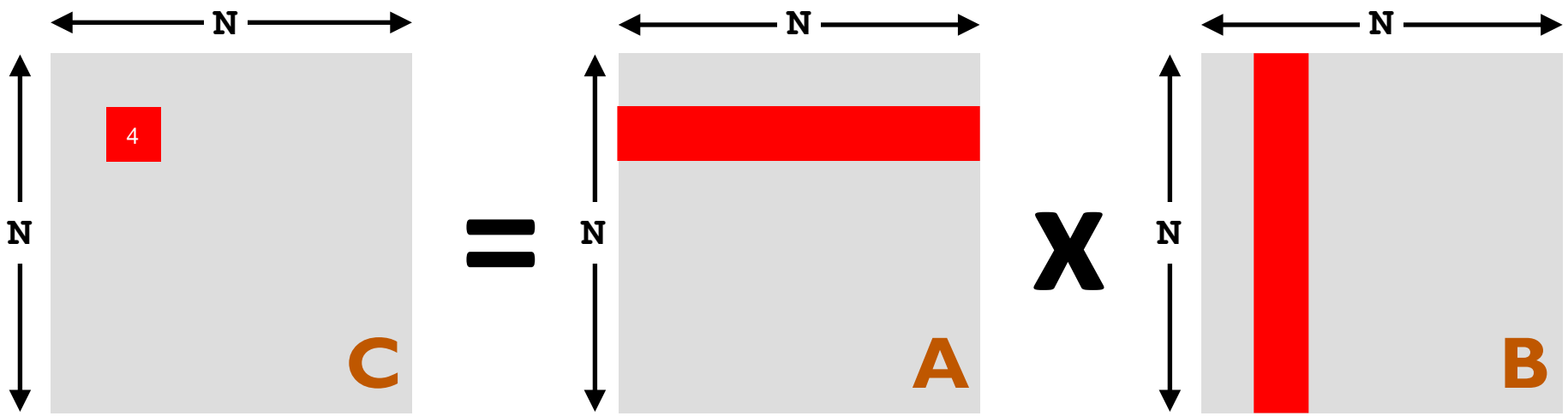


Assume each matrix element is 8 bytes, cache line is 16 bytes, N = 1024

Cache Misses

Iteration	A	B	C
1	512	1024	1
2	512	1024	0
3	512	1024	1
4			
Total			

Matrix Multiplication Example

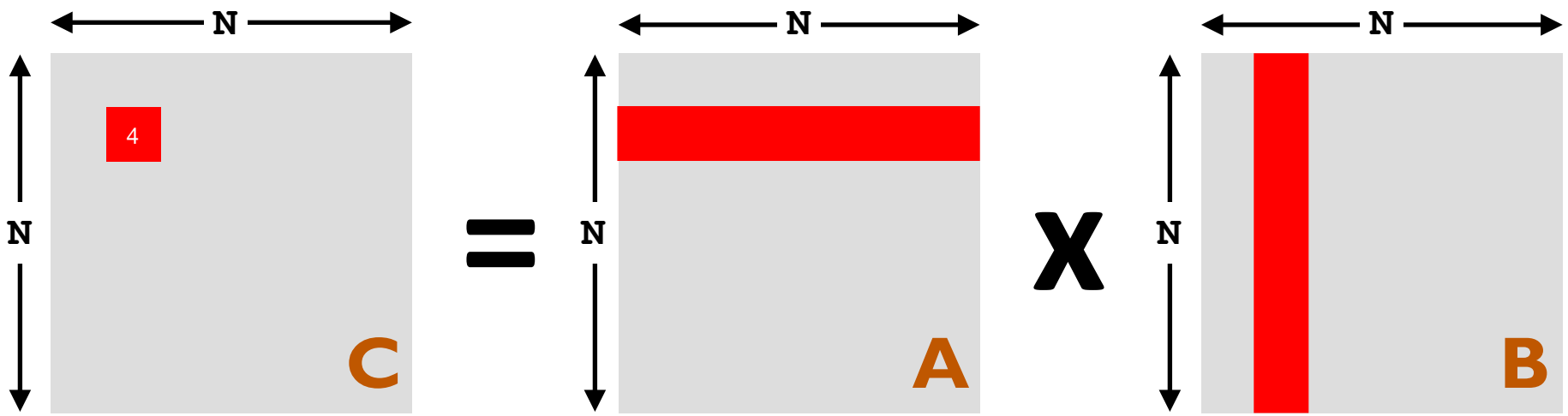


Assume each matrix element is 8 bytes, cache line is 16 bytes, N = 1024

Cache Misses

Iteration	A	B	C
1	512	1024	1
2	512	1024	0
3	512	1024	1
4			
Total			

Matrix Multiplication Example

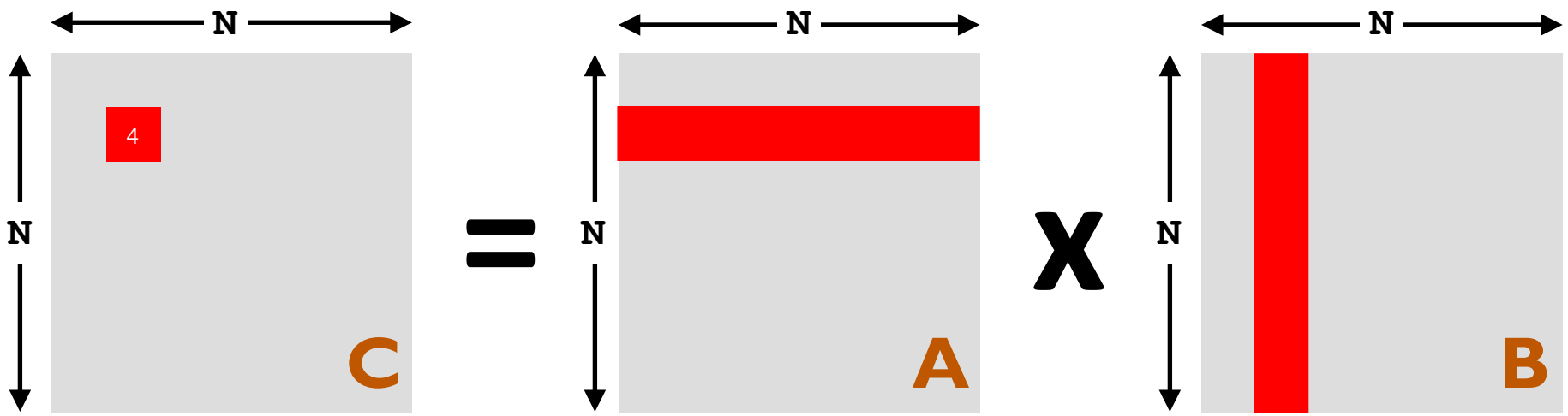


Assume each matrix element is 8 bytes, cache line is 16 bytes, N = 1024

Cache Misses

Iteration	A	B	C
1	512	1024	1
2	512	1024	0
3	512	1024	1
4	512	1024	0
Total			

Matrix Multiplication Example

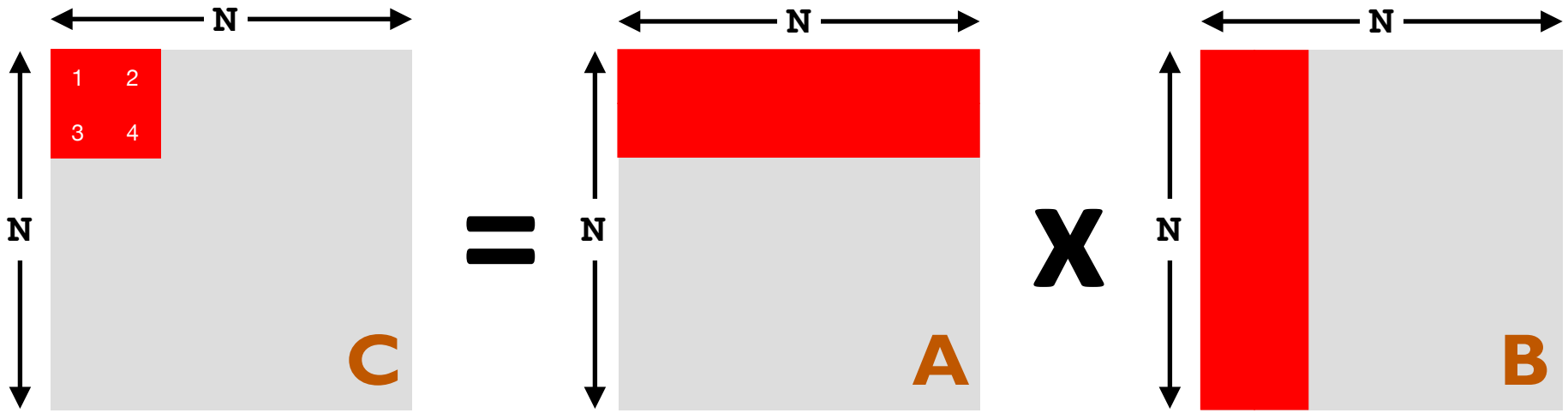


Assume each matrix element is 8 bytes, cache line is 16 bytes, N = 1024

Cache Misses

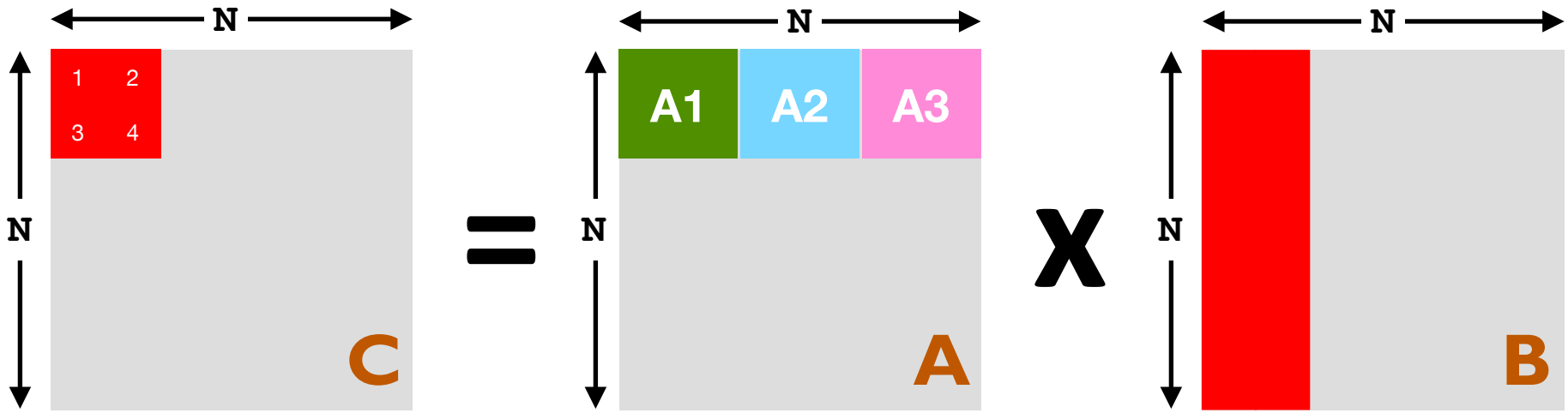
Iteration	A	B	C
1	512	1024	1
2	512	1024	0
3	512	1024	1
4	512	1024	0
Total	2048	4096	2

Matrix Multiplication Example



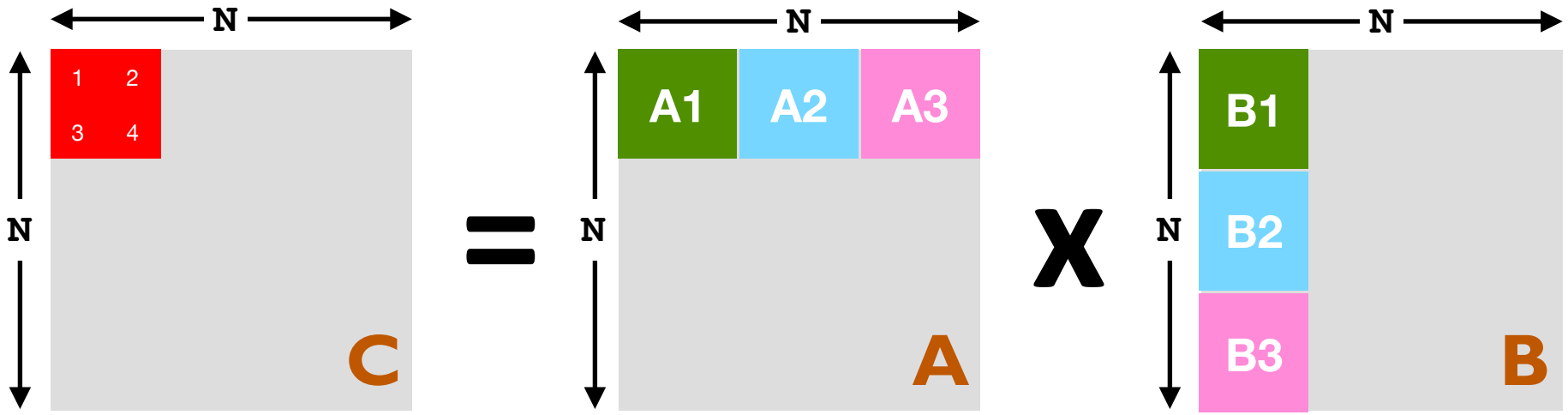
Assume each matrix element is 8 bytes, cache line is 16 bytes, N = 1024

Matrix Multiplication Example



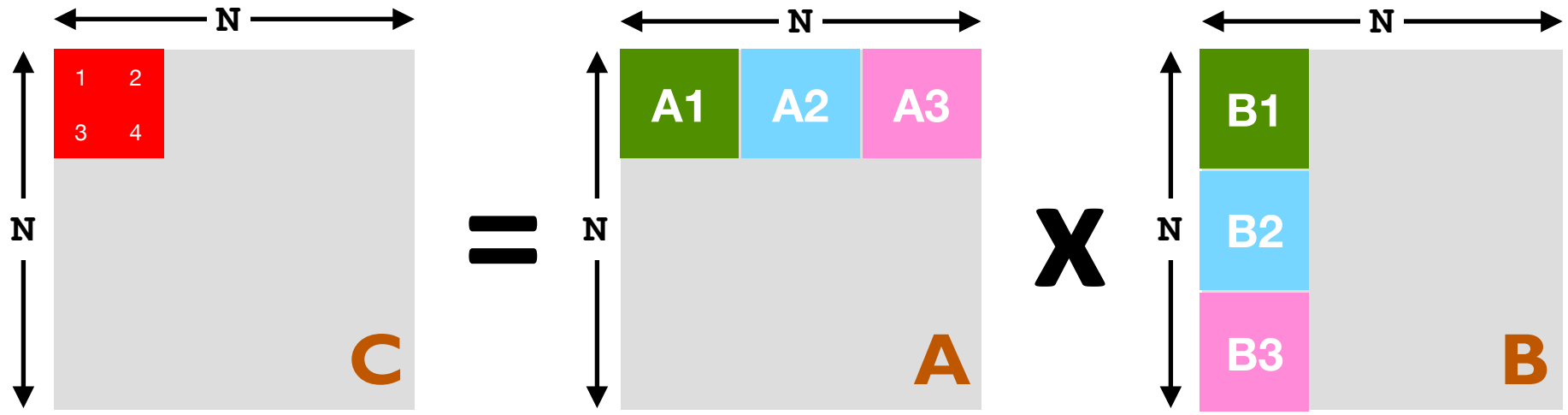
Assume each matrix element is 8 bytes, cache line is 16 bytes, N = 1024

Matrix Multiplication Example



Assume each matrix element is 8 bytes, cache line is 16 bytes, N = 1024

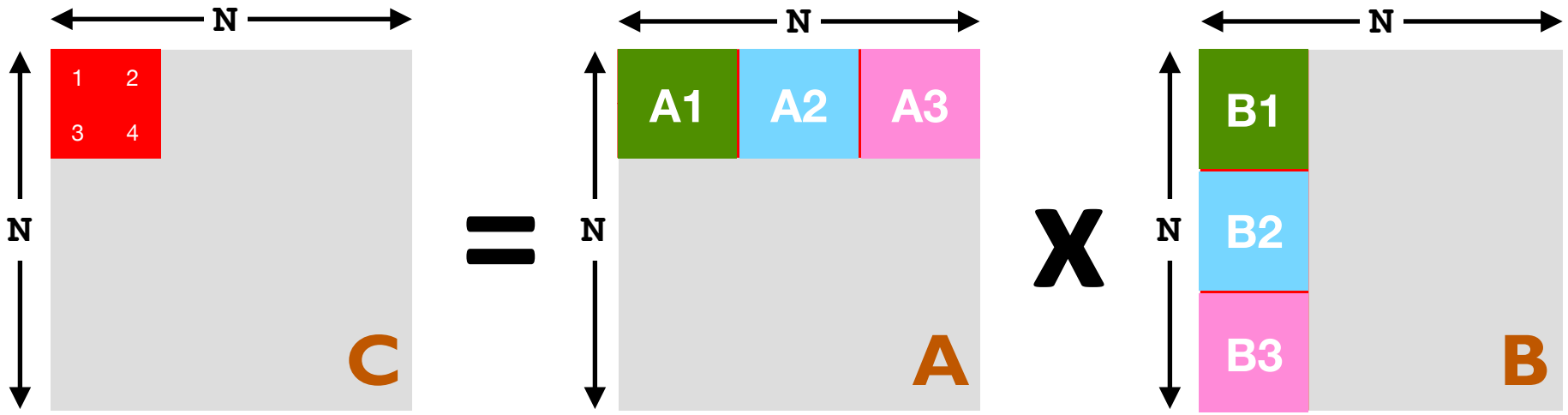
Matrix Multiplication Example



Assume each matrix element is 8 bytes, cache line is 16 bytes, N = 1024

+ … (512 times)

Matrix Multiplication Example



Assume each matrix element is 8 bytes, cache line is 16 bytes, N = 1024

Cache Misses

Block	A	B	C
1	2	2	2
2	2	2	0
3	2	2	0
512	2	2	0
Total	1024	1024	2

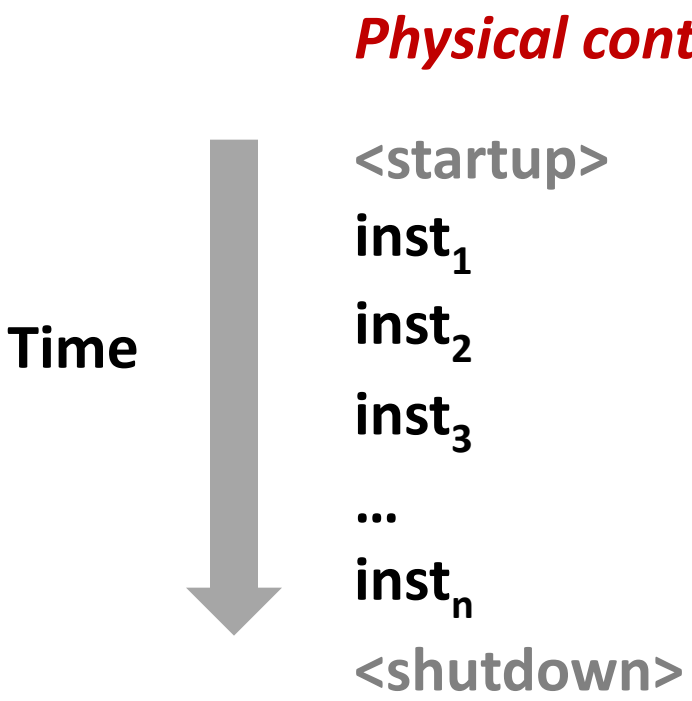
Cache Summary

- Cache memories can have significant performance impact

- You can write your programs to exploit this!
  - Focus on the inner loops, where bulk of computations and memory accesses occur.
  - Try to maximize spatial locality by reading data objects with sequentially with stride 1.
  - Try to maximize temporal locality by using a data object as often as possible once it’s read from memory.

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)



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”

Exceptional Control Flow

- Exists at all levels of a computer system

Exceptional Control Flow

- Exists at all levels of a computer system
- Low level mechanisms
  - 1. **Exceptions**
    - Change in control flow in response to a system event (i.e., change in system state)
    - Implemented using combination of hardware and OS software

Exceptional Control Flow

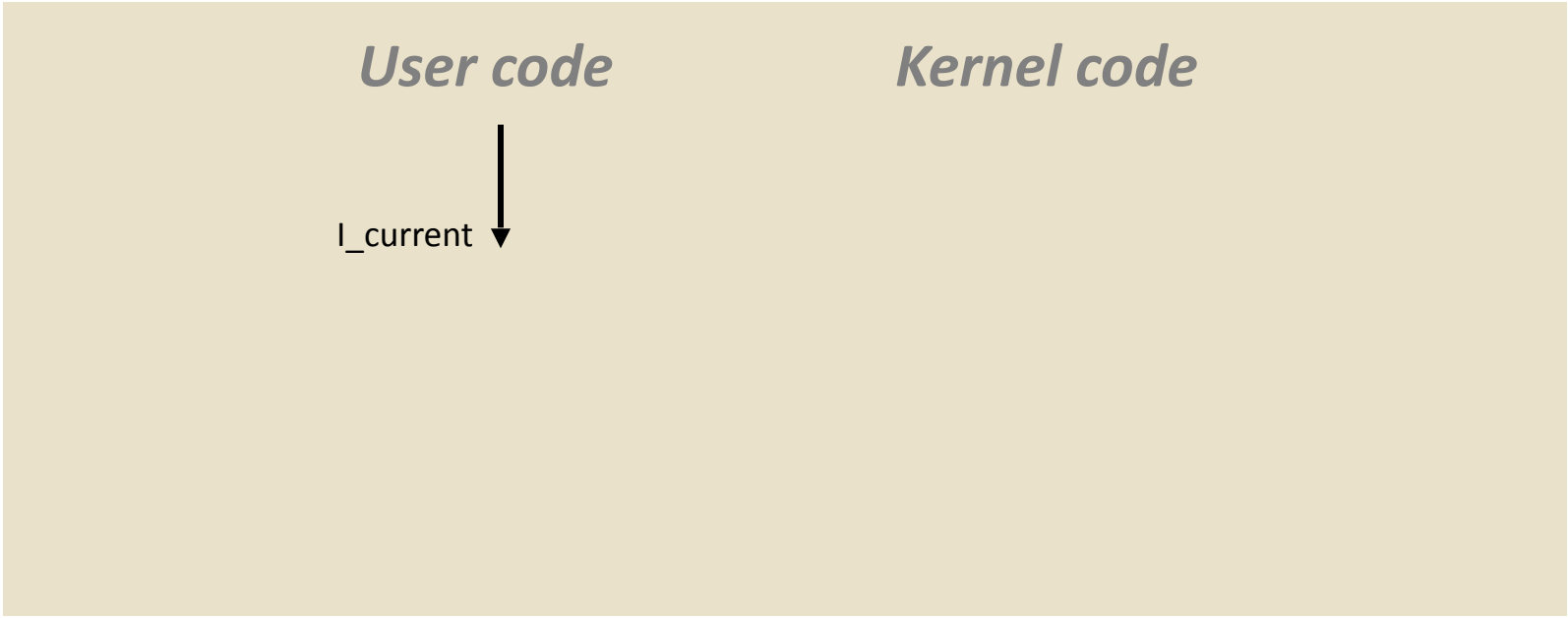
- Exists at all levels of a computer system
- Low level mechanisms
  - 1. **Exceptions**
    - Change in control flow in response to a system event (i.e., change in system state)
    - Implemented using combination of hardware and OS software
- Higher level mechanisms
  - 2. **Process context switch**
    - Implemented by OS software and hardware timer
  - 3. **Signals**
    - Implemented by OS software
  - 4. **Nonlocal jumps**: `setjmp()` and `longjmp()`
    - Implemented by C runtime library

Today

- Exceptions/Interrupts
- How Different Computer Components Communicate: Bus

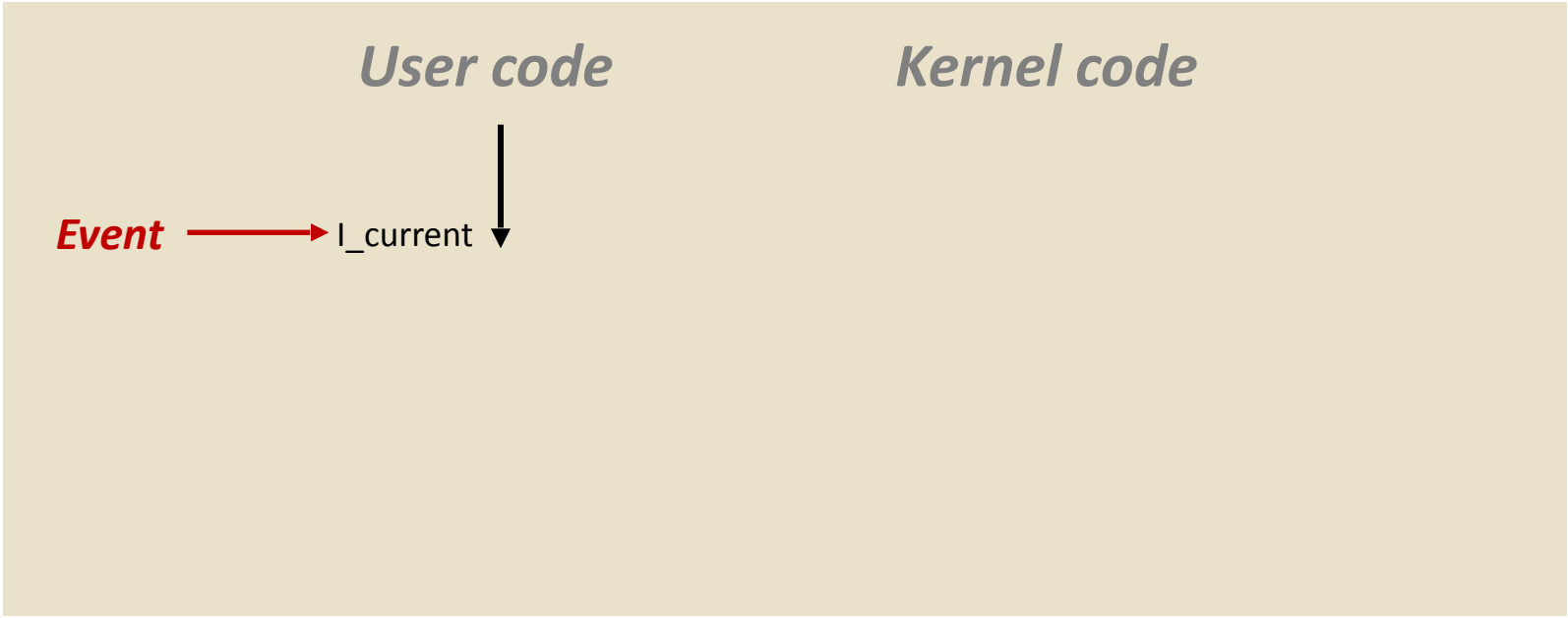
Exceptions

- 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



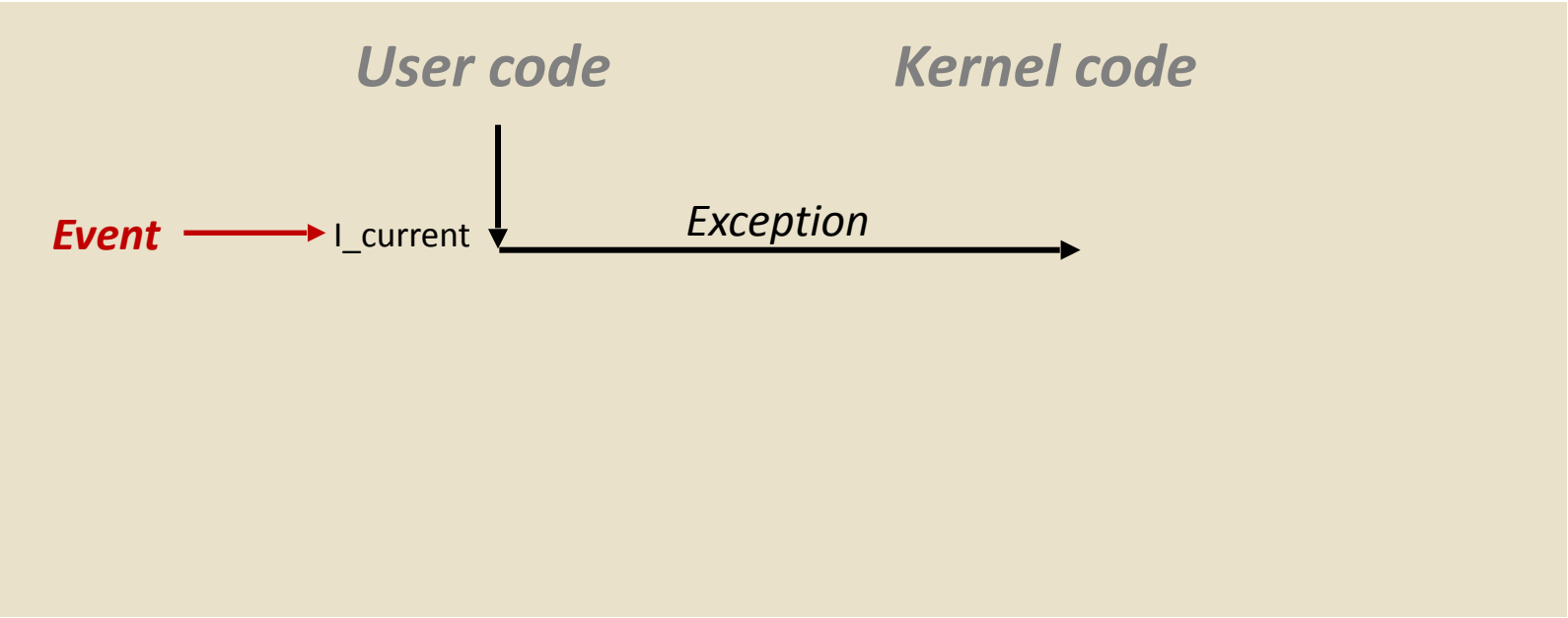
Exceptions

- 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



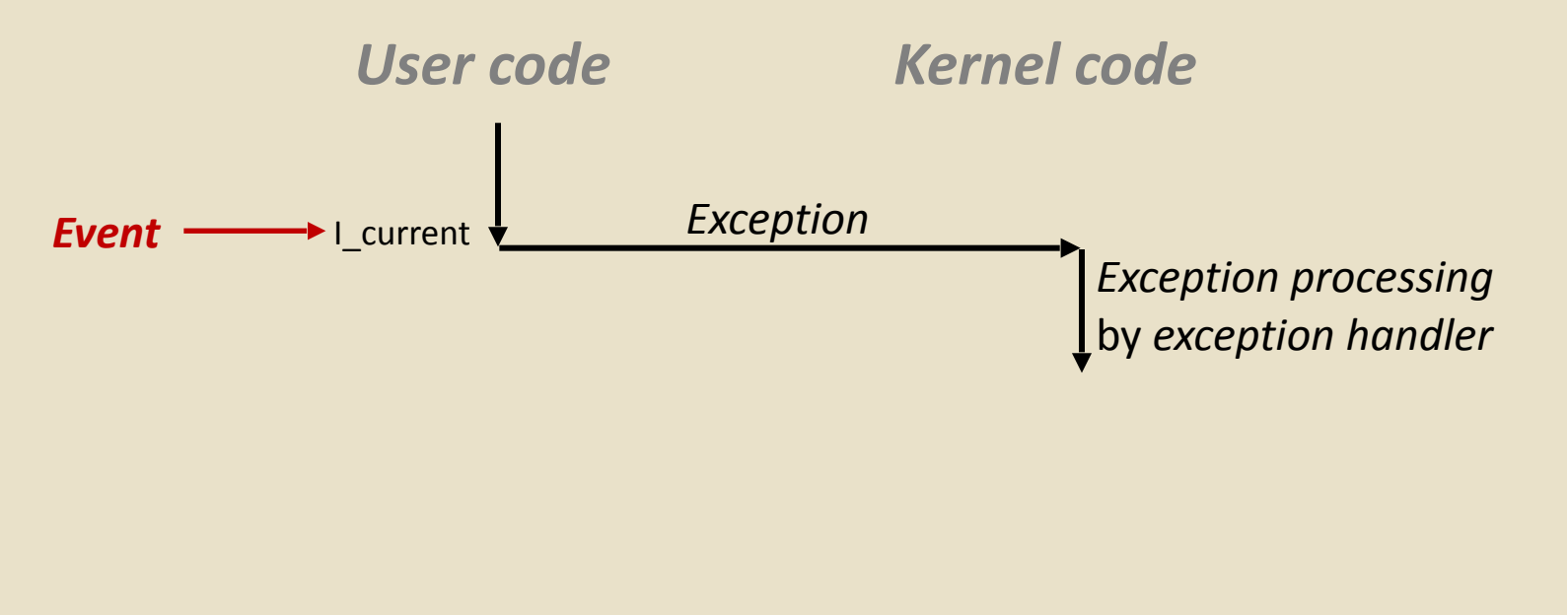
Exceptions

- 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



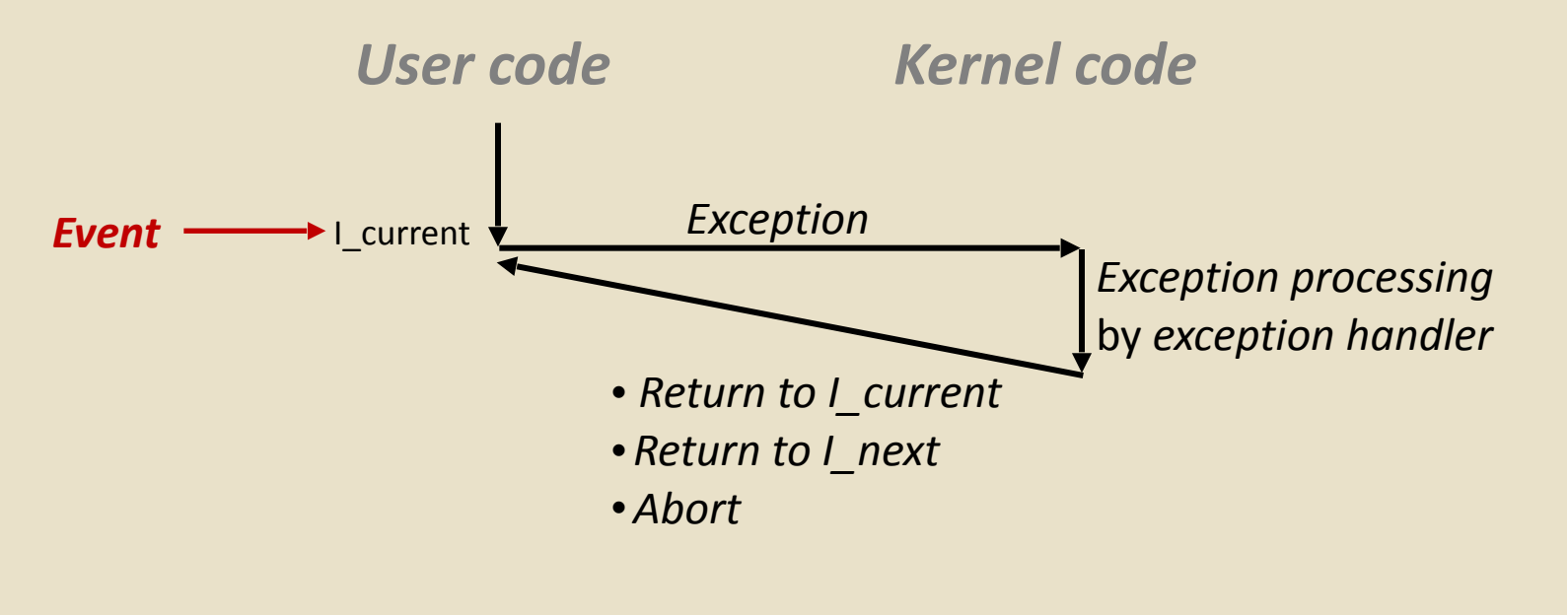
Exceptions

- 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



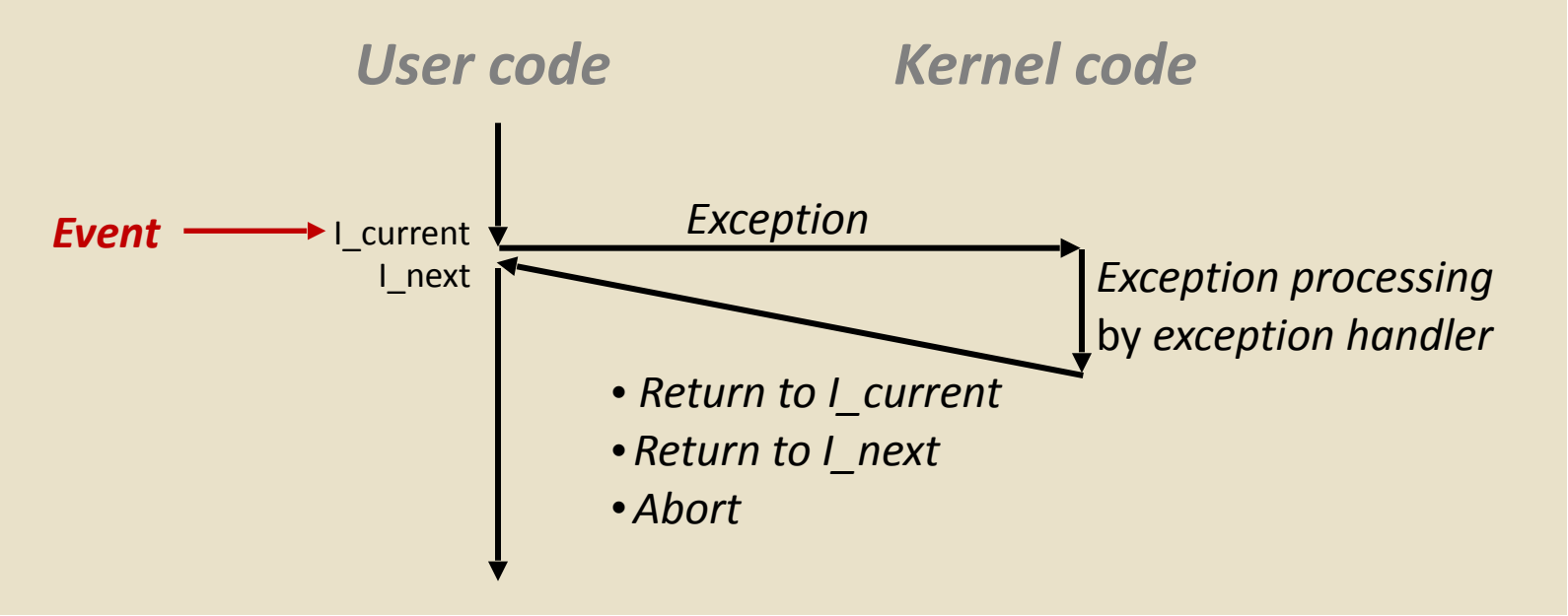
Exceptions

- 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

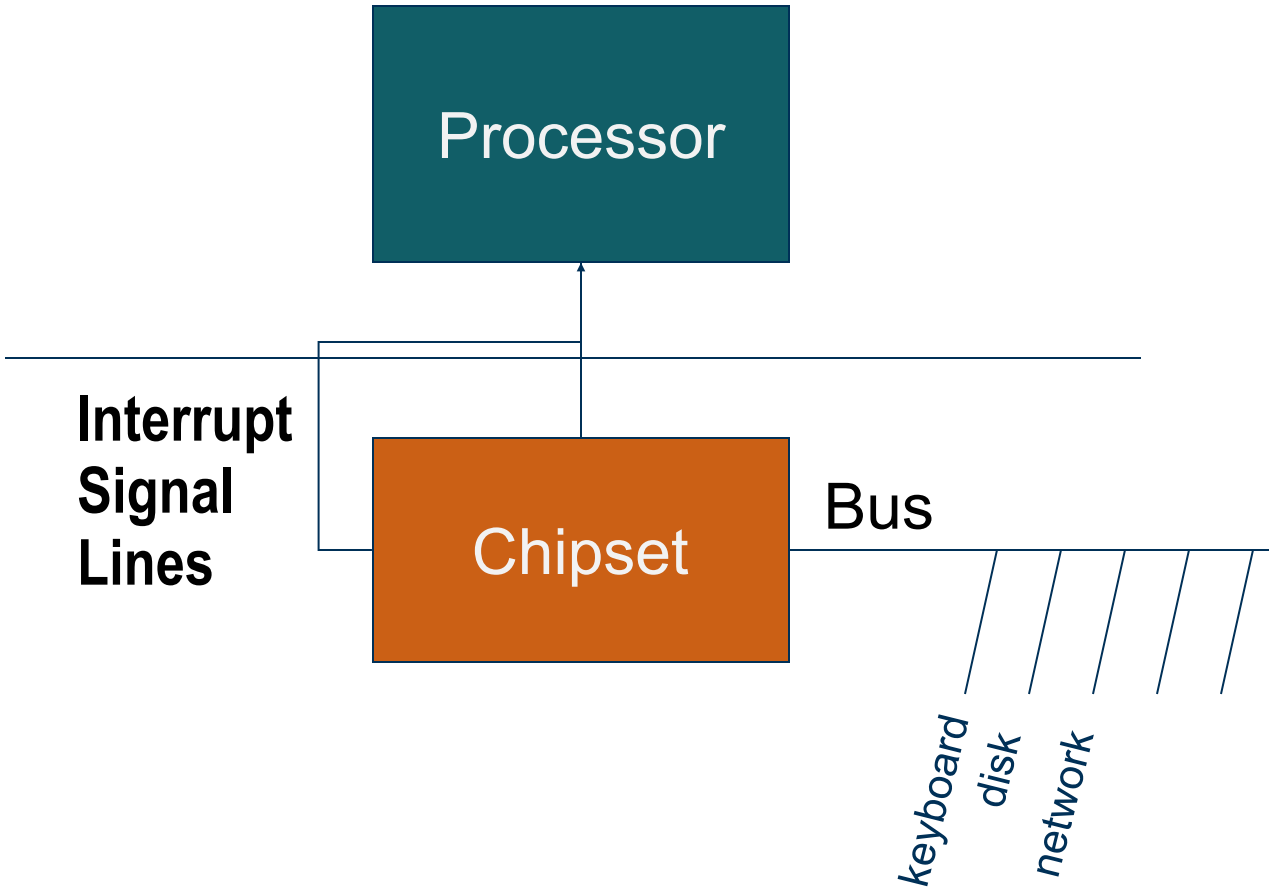
- Examples:
  - Timer interrupt
    - Every few ms, an external timer chip triggers an interrupt

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: illegal instruction, parity error, machine check
    - Aborts current program

Fault Example: Page Fault

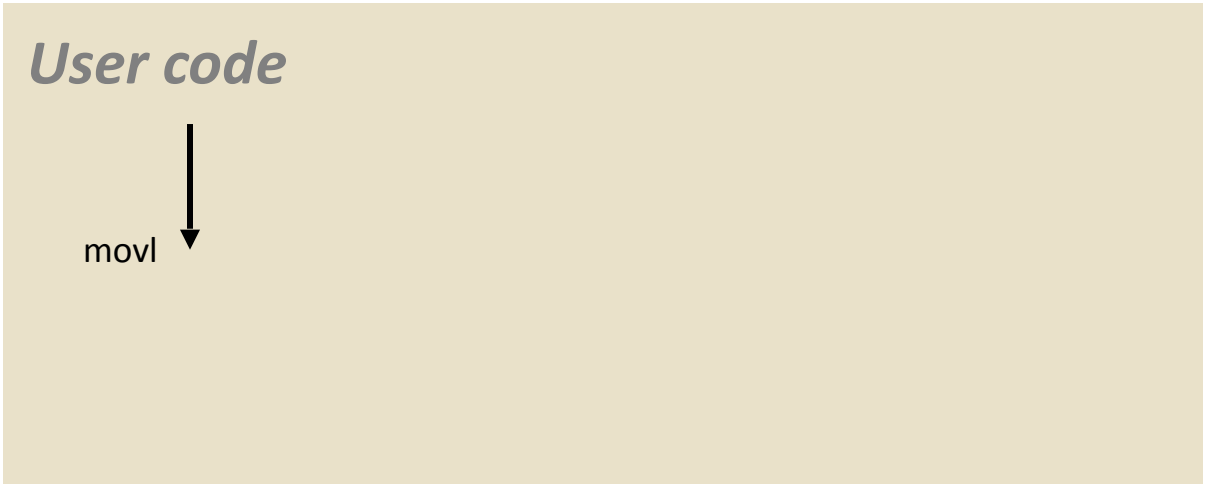
- User writes to memory location
- That memory location is not found in memory because it is currently on disk
- Trigger a Page Fault (recoverable), the exception handler loads the data from disk to memory (will discuss in detail later in the class)

```
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)

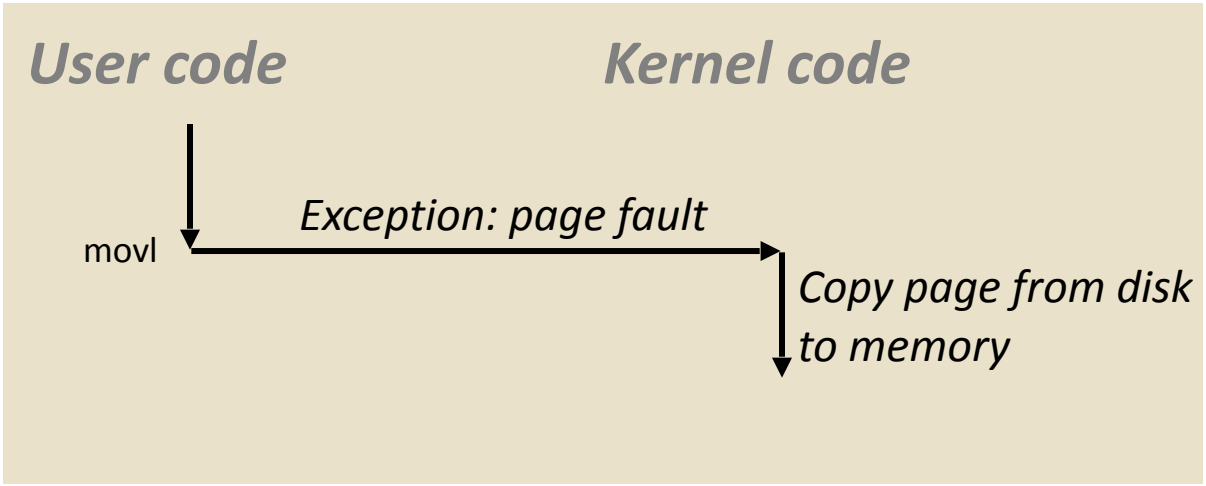
```
80483b7:      c7 05 10 9d 04 08 0d   movl   $0xd,0x8049d10
```



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)

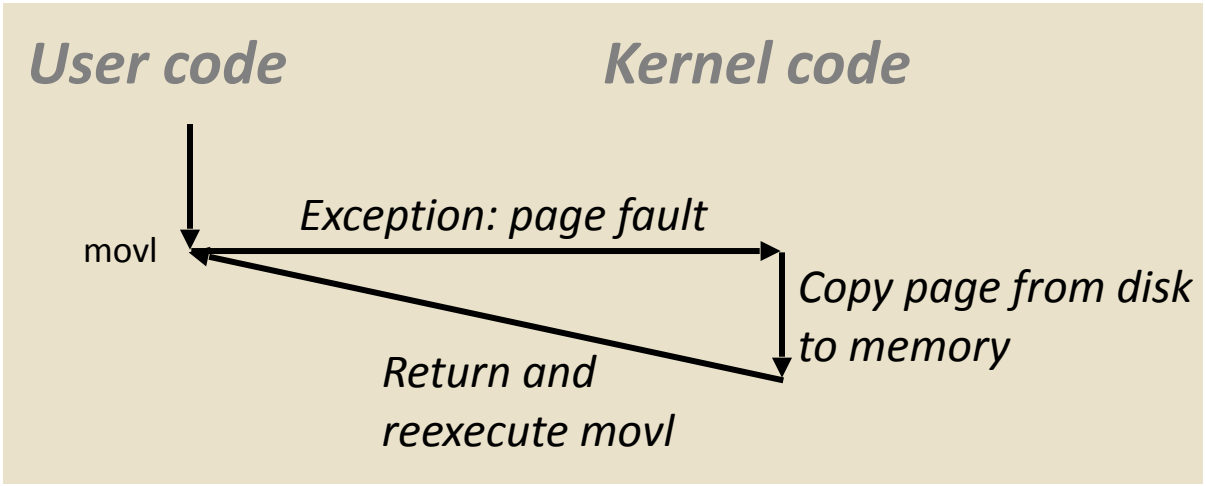
```
80483b7:      c7 05 10 9d 04 08 0d   movl   $0xd,0x8049d10
```



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)

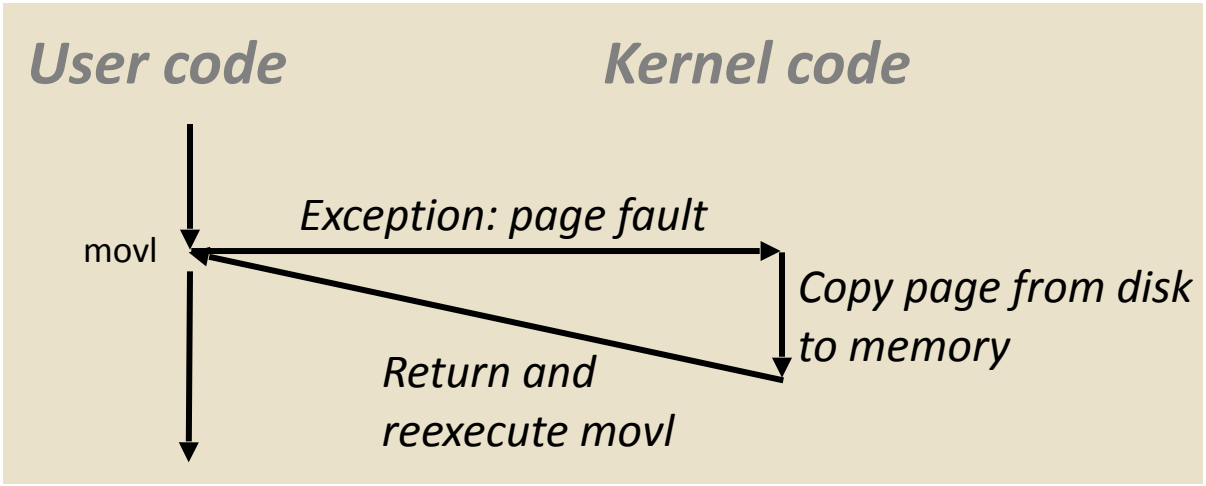
```
80483b7:      c7 05 10 9d 04 08 0d   movl   $0xd,0x8049d10
```



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
```

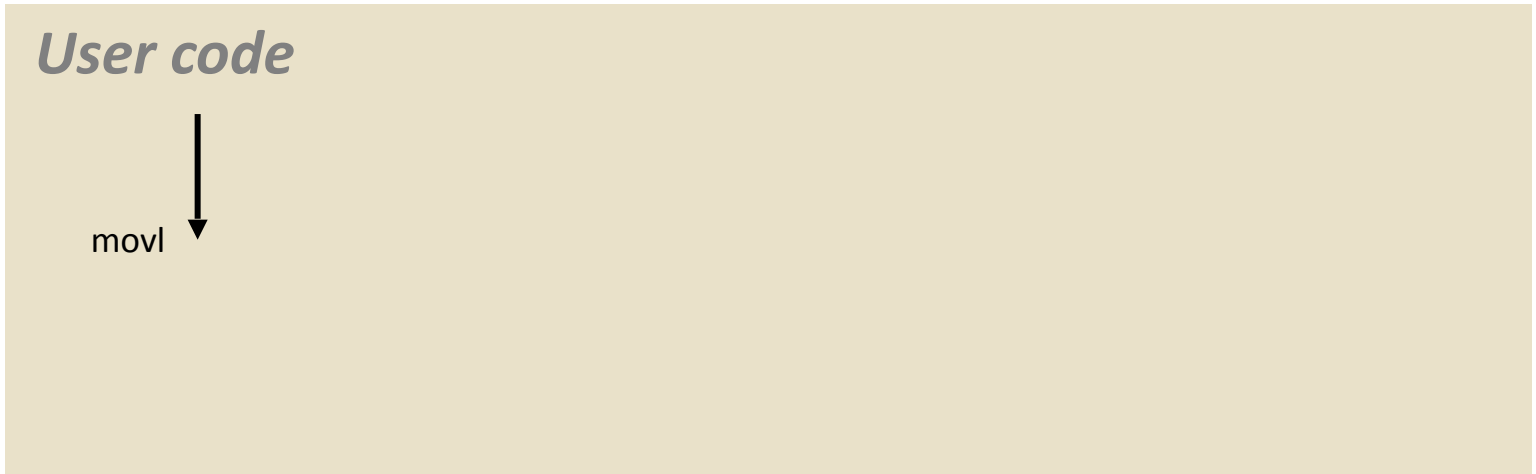


Fault Example: Protection Fault

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

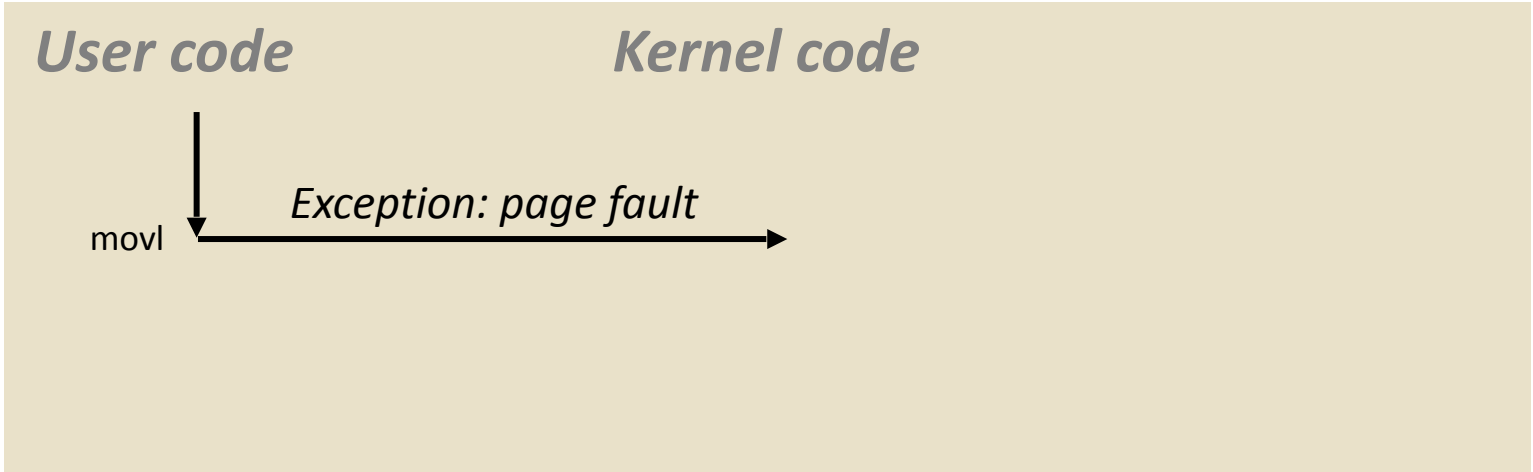
Fault Example: Protection Fault

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



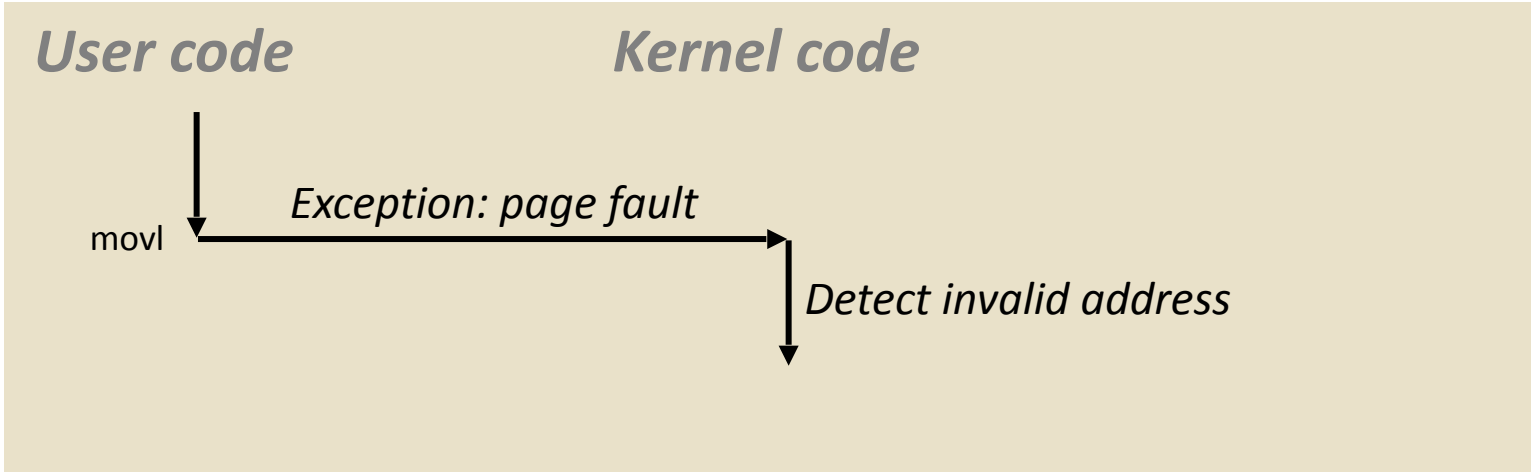
Fault Example: Protection Fault

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



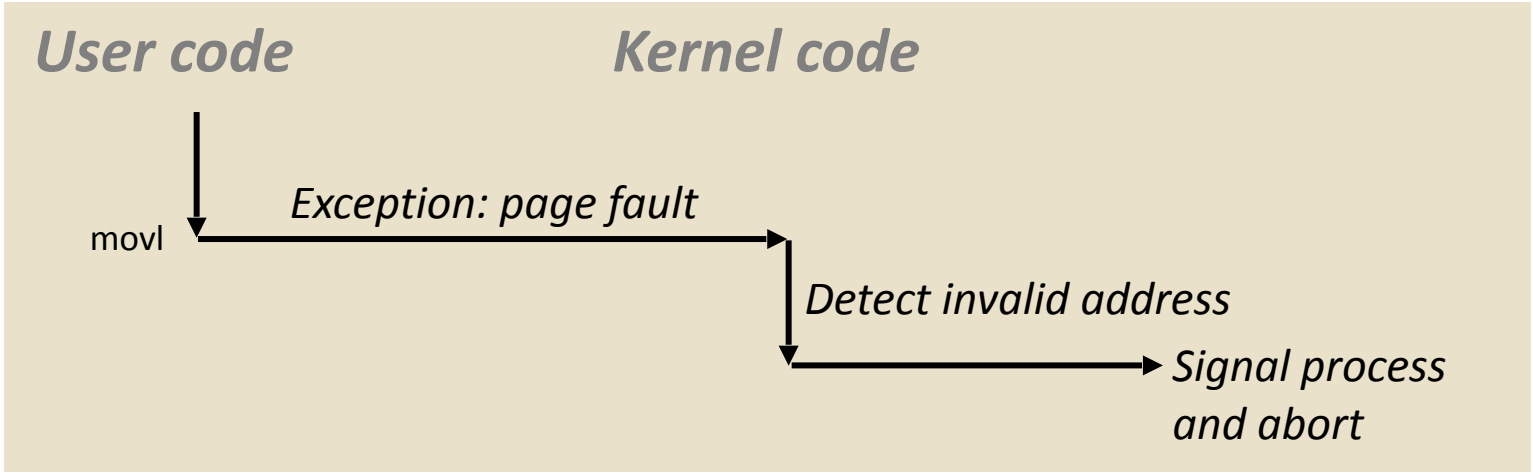
Fault Example: Protection Fault

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



Fault Example: Protection Fault

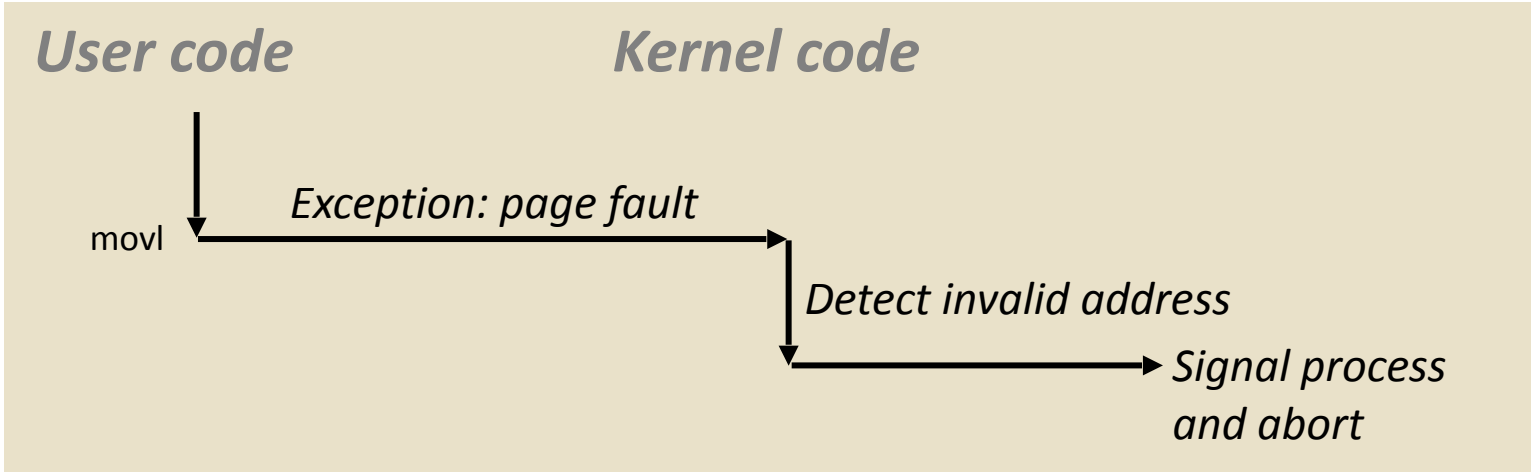
```
80483b7:      c7 05 60 e3 04 08 0d   movl   $0xd,0x804e360
```



Fault Example: Protection Fault

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

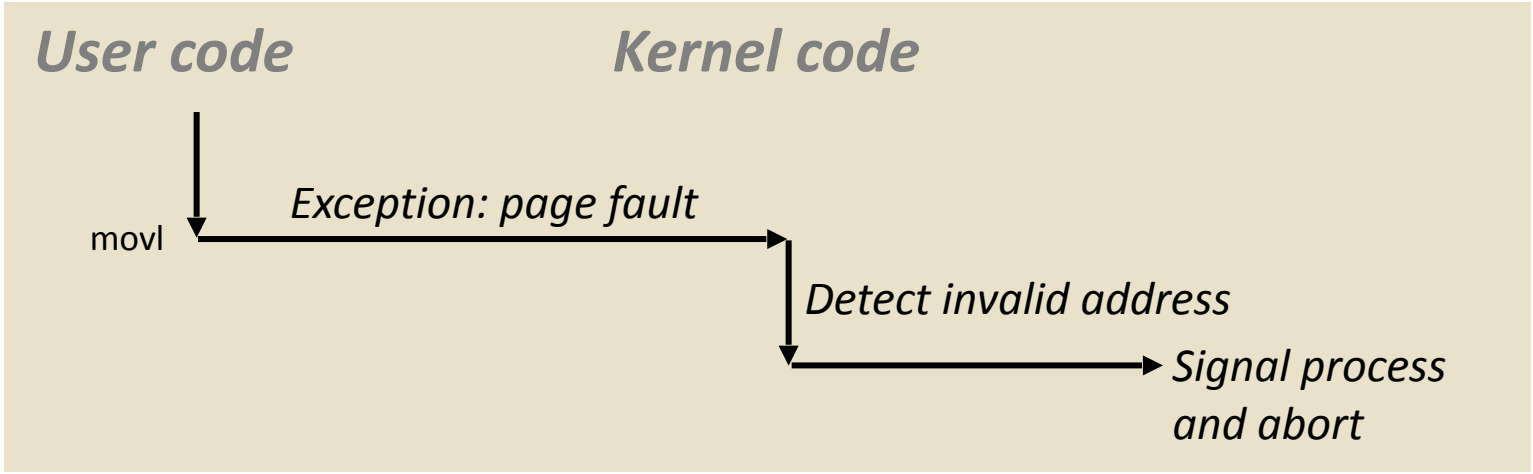
```
80483b7:      c7 05 60 e3 04 08 0d   movl   $0xd,0x804e360
```



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

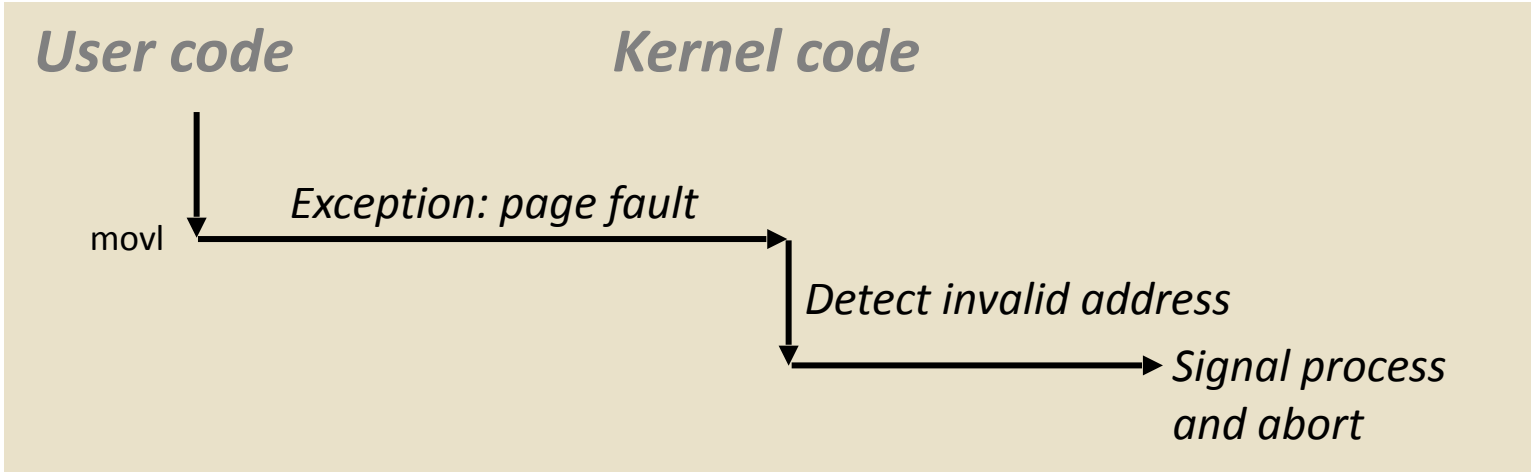
```
80483b7:      c7 05 60 e3 04 08 0d   movl   $0xd,0x804e360
```



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

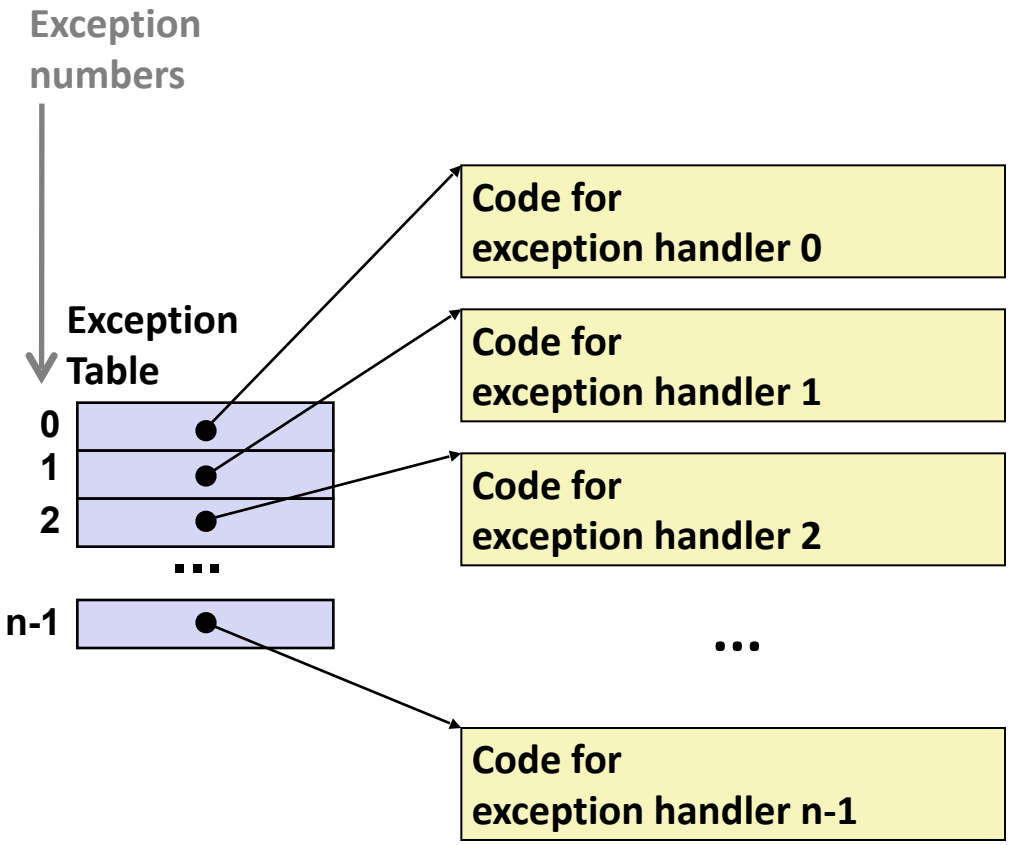
Where Do You Restart?

- 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

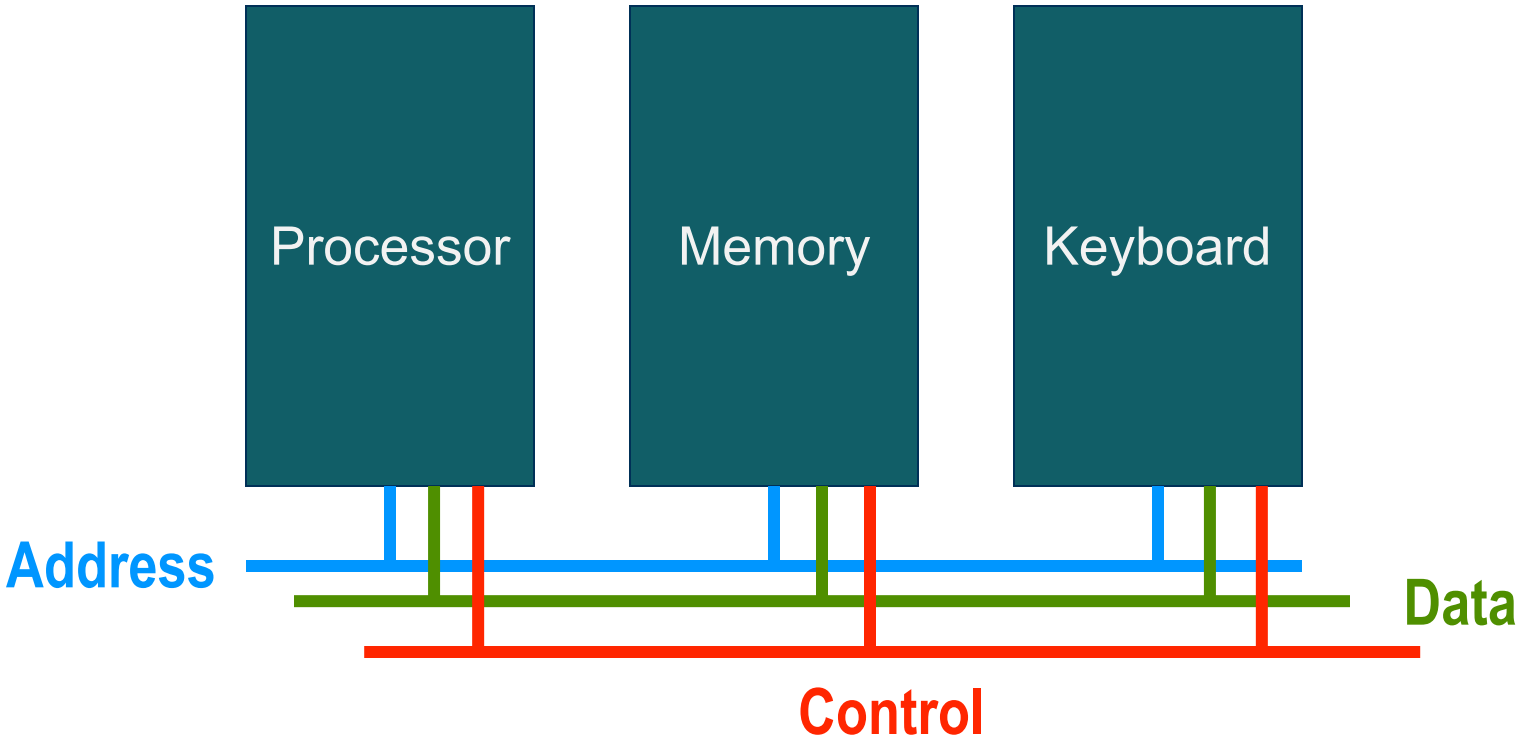
Today

- Exceptions/Interrupts
- How Different Computer Components Communicate: Bus

What Is A Bus?

- Shared data transport
  - Allows different devices in a computer to share data
  - E.g., CPU reads data from memory; keyboard sends keystrokes to the CPU, etc.
- Requires
  - Address
  - Data
  - Control
- Traditionally parallel (multiple wires) but lately some have become serial
  - USB: Universal Serial Bus

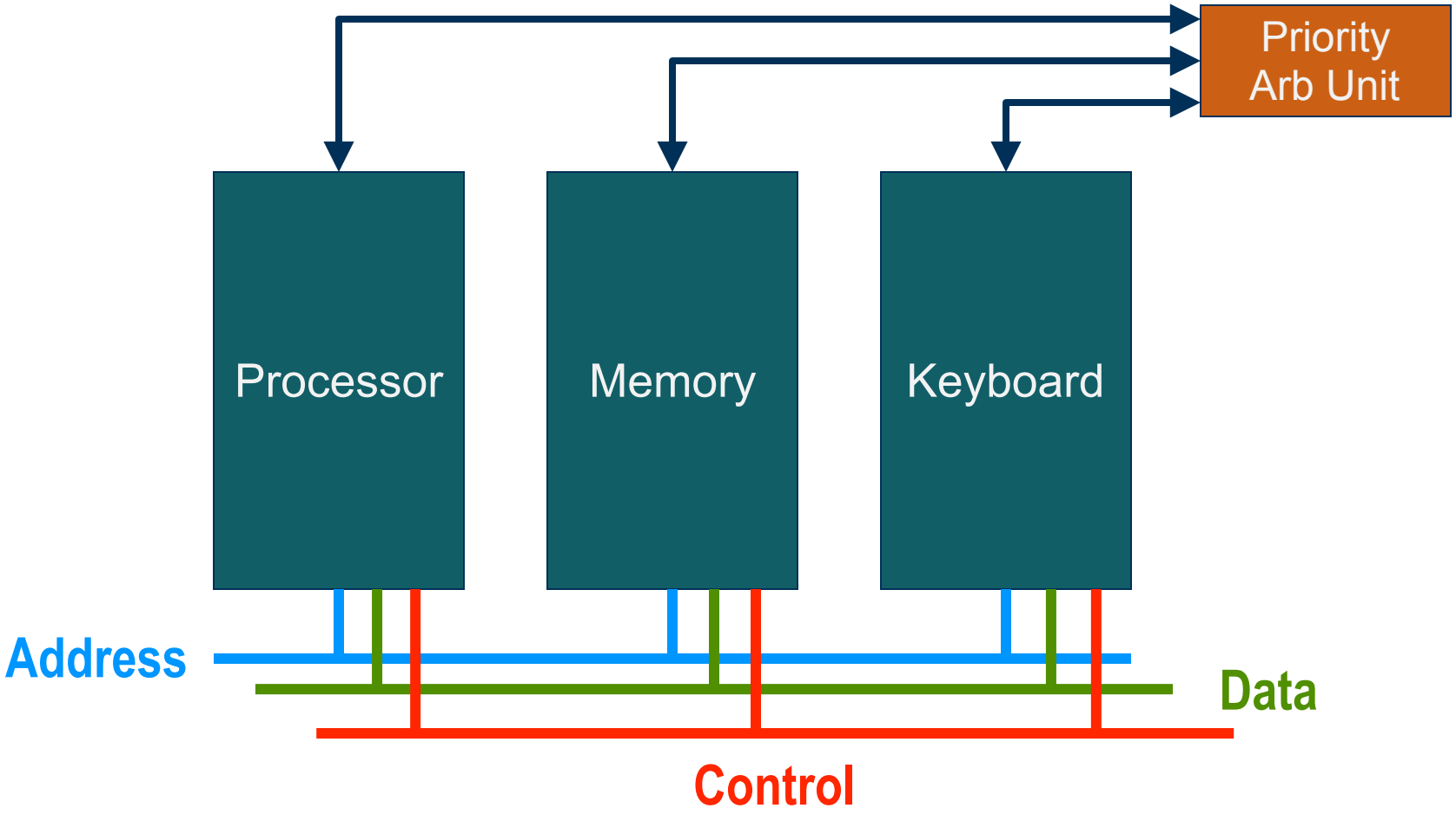
A Simple Bus



Bus Characteristics

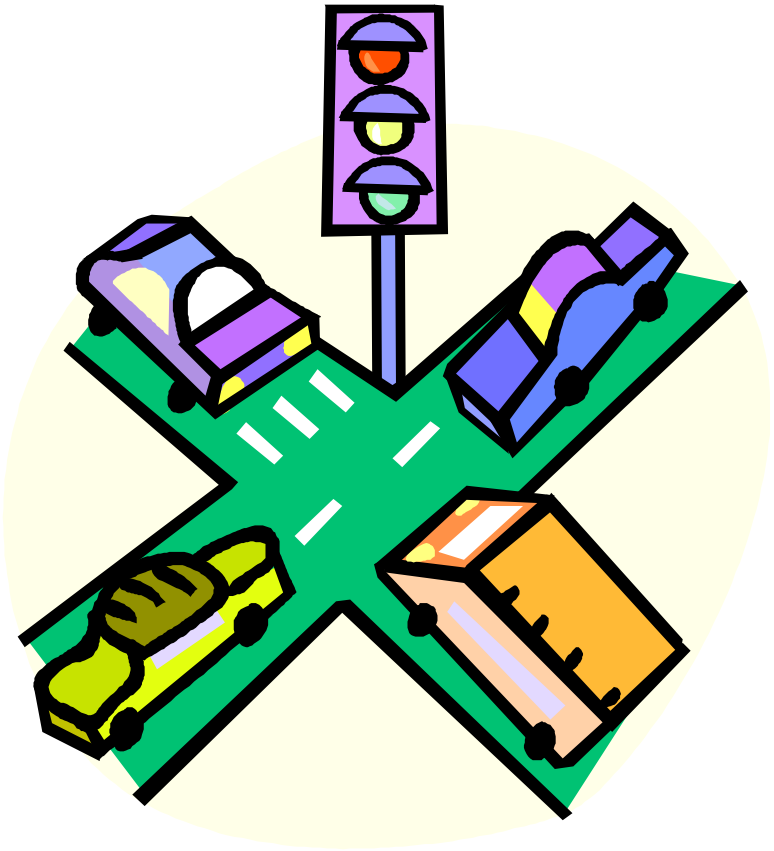
- Arbitration: decides who gets bus
  - Central
  - Distributed
- Transfer of data on bus
  - Asynchronous: notify when you are done
  - Synchronous: every transfer takes the same amount of time
- Bus tenure
  - Pending
  - Split transaction

Centrally Arbitrated



Central Arbitration Example

- Downsides (when you have many devices)
  - Time to collect info
  - Decision time
  - Transmit decision



Distributed Arbitration

- Each device decides on its own whether its allowed to use the resource
  - May have information from other devices

Distributed Arbitration

- Each device decides on its own whether its allowed to use the resource
  - May have information from other devices
- Potential problems?
  - Collisions: Must have some way to detect and resolve collisions

Distributed Arbitration

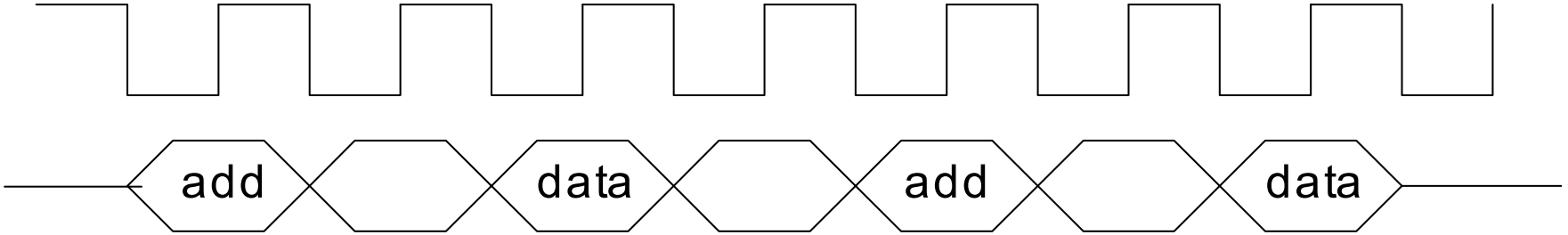
- Each device decides on its own whether its allowed to use the resource
  - May have information from other devices
- Potential problems?
  - Collisions: Must have some way to detect and resolve collisions
- Examples
  - Dinner table
  - Ethernet: Collision Sense Multiple Access (CSMA)

Synchronous Bus

- All transfers take the same amount of time
  - If less time required, waste the time
- Bus cycle fixed
- Tend to be short buses (minimize clock time)
- Tend to be similar speed devices

Synchronous Bus

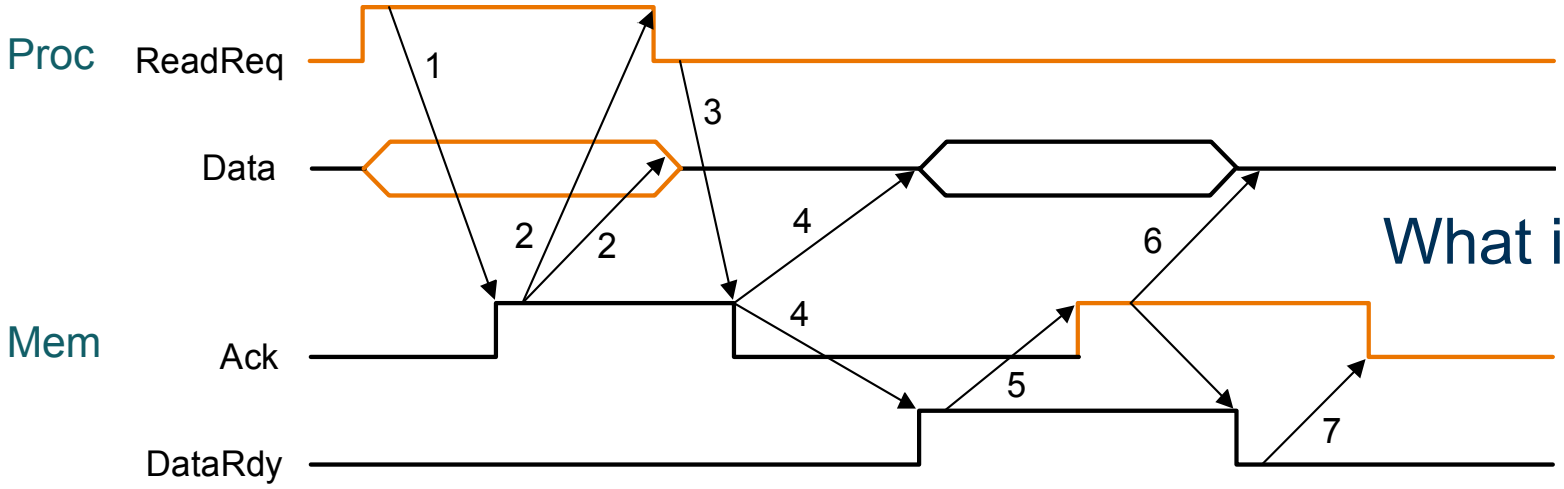
- Example: Processor read data from memory
  - Cycle 1: submit address
  - Cycle 3: return data



Asynchronous

- Handshaking protocol
  - Identify start and end of each phase of the bus protocol
- Not tied to a clock
  - Can start/finish transactions at any time

The Handshake Protocol



What is missing?

Processor reading memory

Processor asserts ReadReq signal and sends the address over the shared bus:

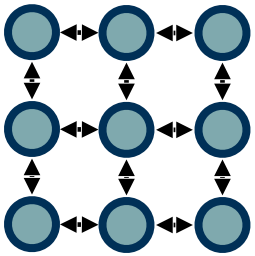
1. Mem sees the Readreq, reads the address and sets Ack.
2. Processor sees the Ack line is set and releases the ReadReq and data lines.
3. Mem sees that ReadReq is low and drops Ack to acknowledge that.
4. Mem places the data on the data lines and raises DataRdy.
5. Processor sees DataRdy, reads the data from the bus and raises Ack.
6. Mem sees the Ack signal, drops DataRdy and releases the data lines.
7. Processor sees DataRdy go low, drops Ack which indicates that transmission is over.

Bus Tenure: Pending vs. Split

- Pending:
  - Hold onto resource until you're done with sending both address and data
  - Other devices can't use the bus during this time
  - Wasteful, but simple
- Split:
  - Allow other transfers to go when you're not ready
  - Send address, release the bus let others use, then send data

From buses to networks

- Buses are great, but:
  - Signal integrity difficult with many “bus drops”
  - Lowers speed
- Point-to-point connections offer highest speed, but:
  - Too many end-points for everything to connect to everything
- Interconnection networks
  - Take multiple hops to get from place to place (node to node)
  - See next slide



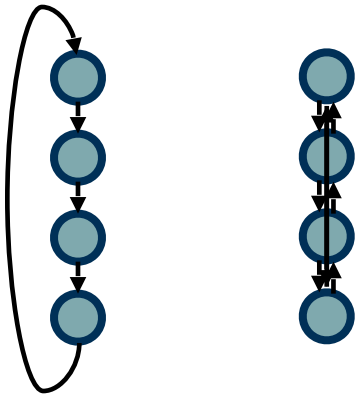
- Can combine buses and various networks in a hierarchy

Interconnection network topologies

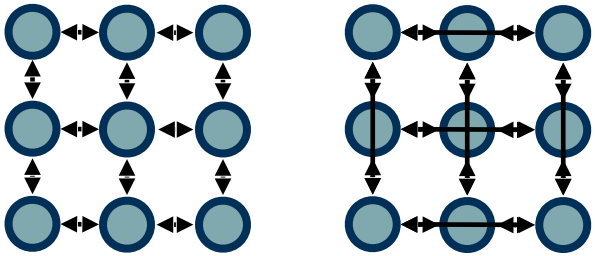
- Network topology = how nodes are connected to one another
- Topology considerations:
  - Diameter (maximum hops between any two nodes)
  - Bi-section bandwidth (minimum BW between any two “halves”)
  - “Embedding” / packaging
  - Cost of links (some more expensive than others)
  - Routability

Direct

Indirect

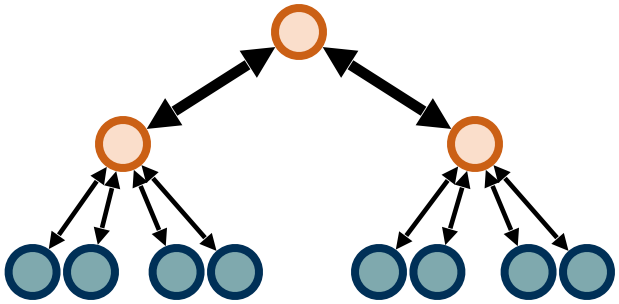


Uni- and bi-directional rings



2D mesh

2D torus



“Folded” Clos
a.k.a. fat tree