CMPE 220

Class – Operating Systems - part 2

What Does a Modern Operating System Do?

1. Process Management
   - Interprocess Communications
2. Input / Output (I/O) Management
3. Memory Management
4. File System Management
5. System Functions and Kernel Mode
6. User Interaction – (maybe)

(4) File System Management

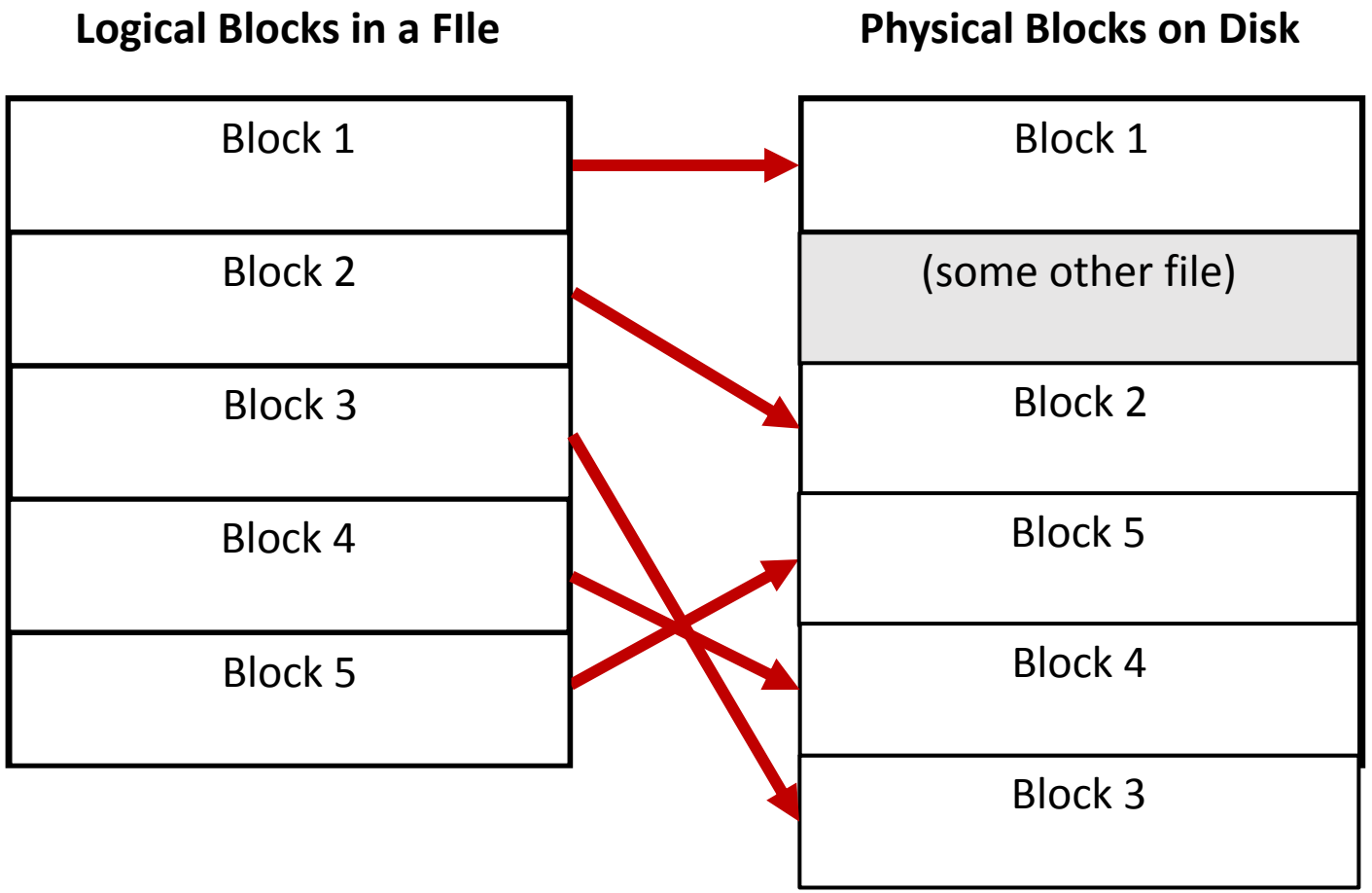
- The operating system is responsible for creating, deleting, opening, closing, reading and writing files on disk
- It manages a “directory” to allow files to be accessed and located by name
- It manages the organization and placement of file data on the disk
- The File System uses the underlying I/O System to perform the physical I/O to and from the disk.

The File Catalog

- Disk files are managed by the operating system. The system manages a *catalog* or *directory* of files, as well as file layout information.
- The catalog is simple an index that maps files by name to their location on disk.
- On most modern systems, the catalog is hierarchical, allowing for nested directories or folders.

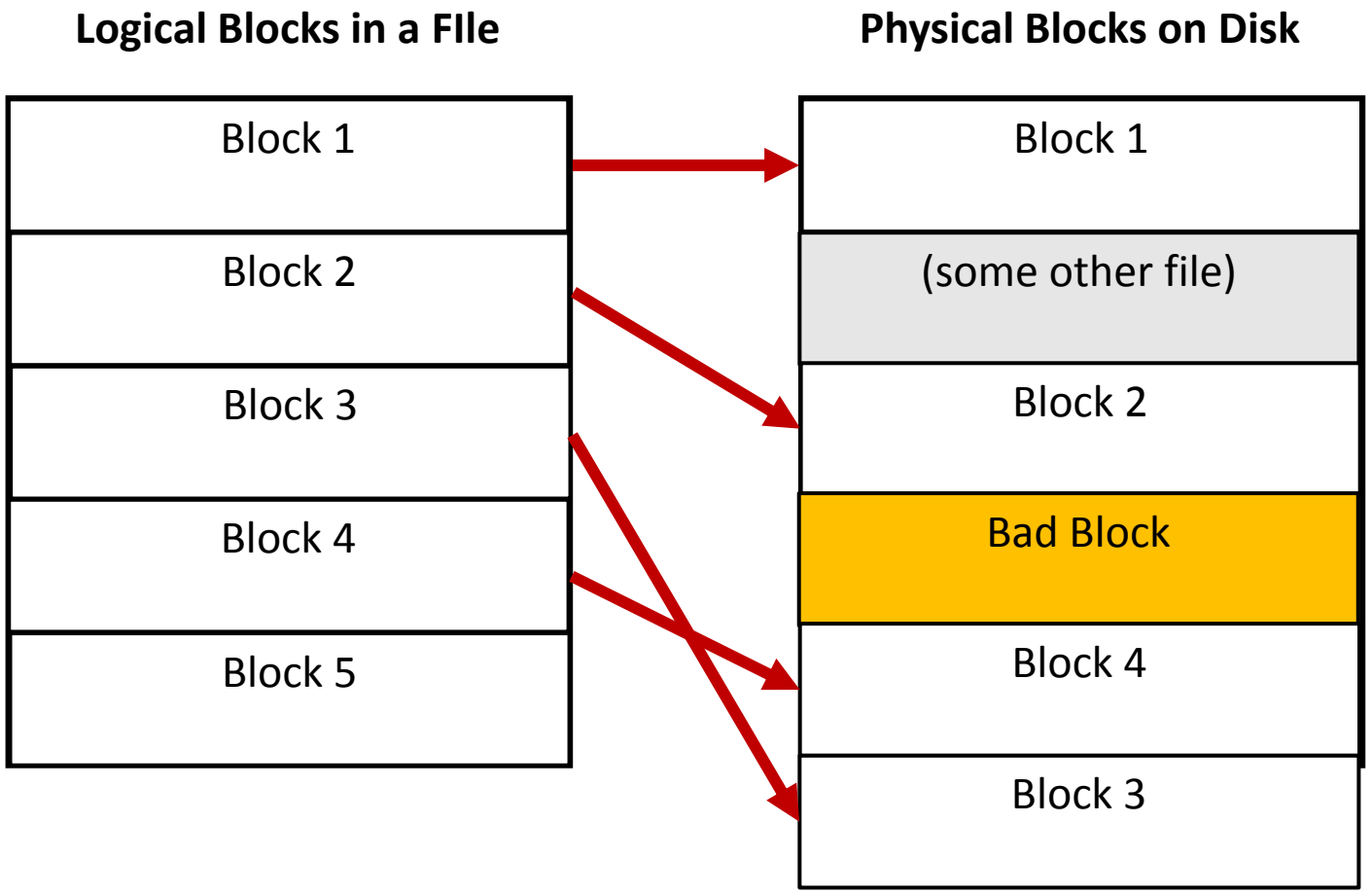
File Layout on Disk

- Files are made up of fixed size blocks. A file is a series of blocks; these blocks are mapped to corresponding physical blocks on the disk.



File Layout on Disk

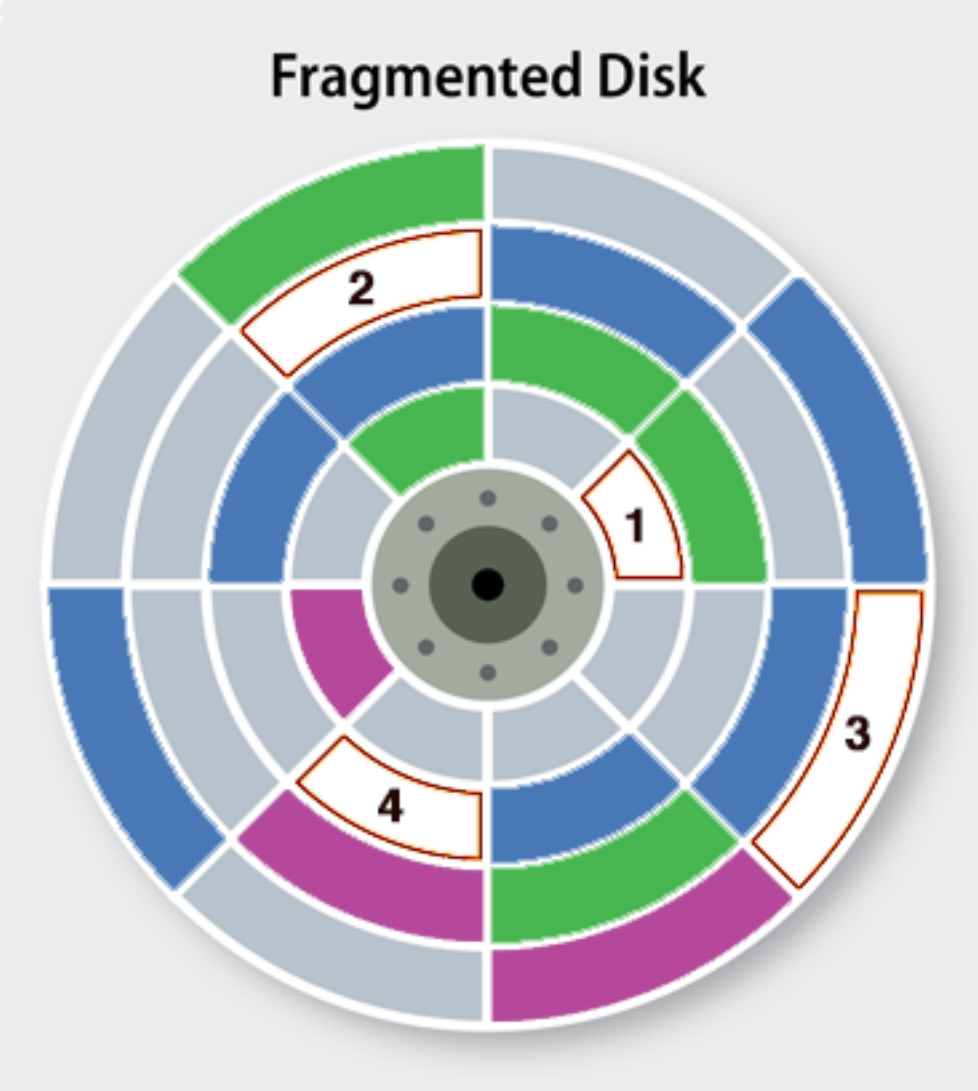
- When disk media develops errors, some blocks may be marked as “bad” and not used.



Disk Fragmentation

- Just like physical memory allocation, the space on disk becomes broken up into small chunks as files are added, deleted, expanded.
- The result dramatically degrades system performance, because reading sequential data from disk becomes very slow.
- Disk transfer rates are pretty fast
- Disk rotation is slower… waiting for the disk to rotate to read the next block is *expensive*
- Moving the disk head from track to track is slower still… so if the file blocks are scattered, reading successive blocks of data can be *very* expensive

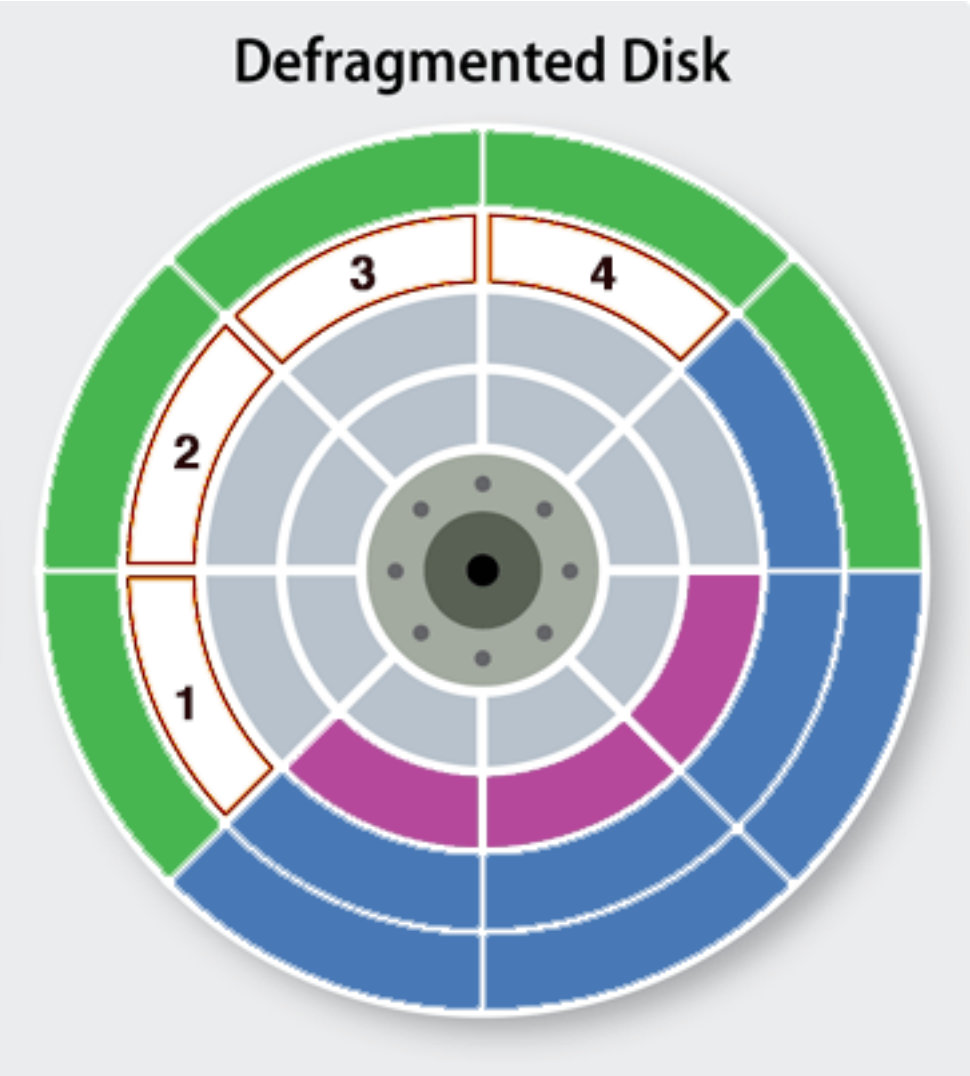
Disk Fragmentation



Defragmentation

- To reduce the performance impact of disk fragmentation, utility programs were developed to “defrag” the disk. This involved rewriting each file as a contiguous series of blocks, and eliminating unused space between files.
- Defragmenting a large disk could take several hours, and the disk could not be used while the utility was running.
- Modern operating systems dynamically defragment the disk, so it is in most cases no longer necessary to run a separate utility.

Defragmentation



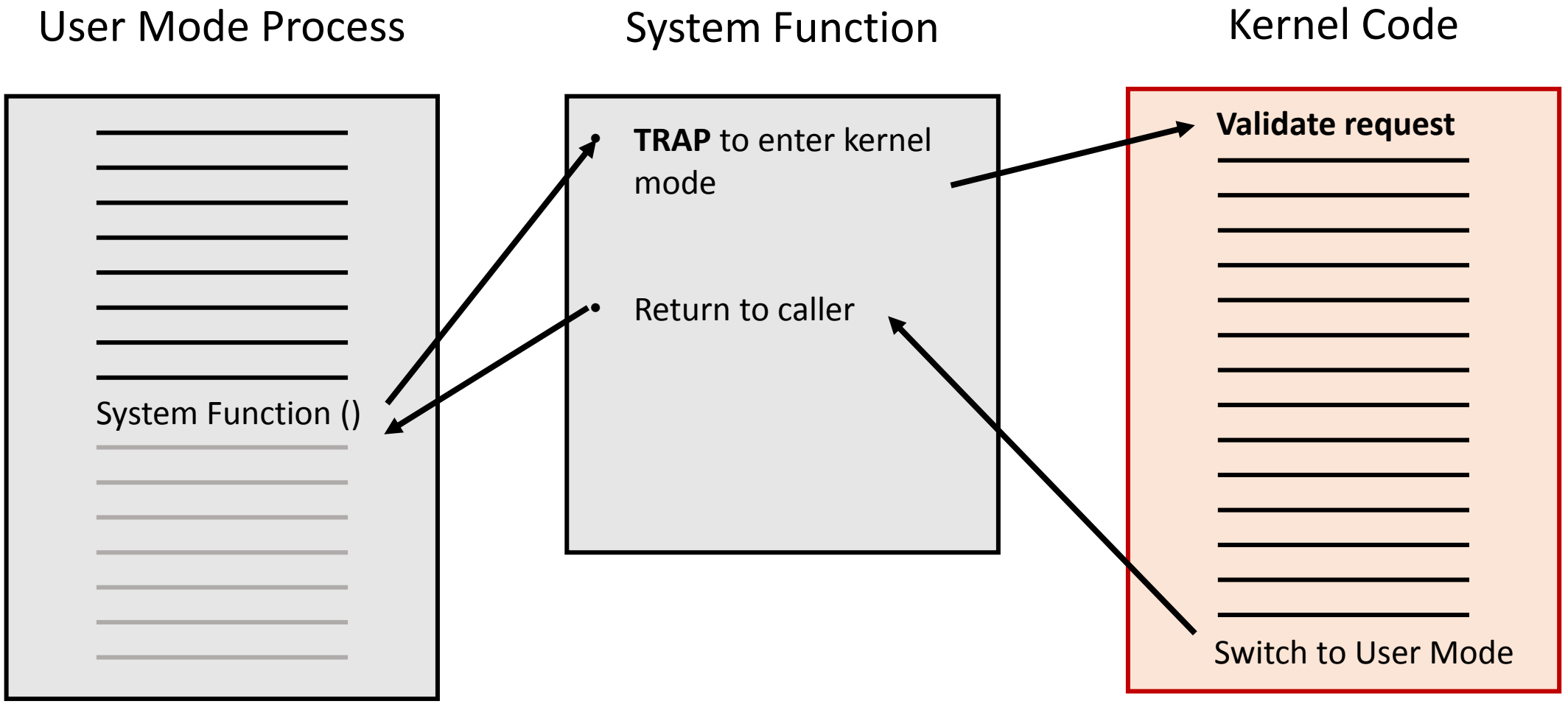
(5) System Functions and Kernel Mode

- Modern CPUs support two modes of instruction execution: *user mode* and *kernel mode*
- When a process is running in *user mode*, its capabilities are restricted:
  - It cannot execute certain instructions, such as I/O instructions
  - It cannot trigger an interrupt
  - It cannot access a different process's memory partition
- The operating system needs to do all of those things, so system processes may run in *kernel mode*

Calling System Functions

- User programs may call system functions that have been statically or dynamically linked. But since the entire process is running in user mode, those system functions still cannot perform restricted operations, such as initiating I/O, or even updating system control structures.
- System functions have a mechanism for invoking code in *kernel mode*.
- This mode switch is initiated with a special software interrupt, sometimes called a *trap*.

Switching Execution Modes



Kernel Mode is Dangerous

- Modern operating systems are well-protected. It's very difficult for a program running in *user mode* to crash or penetrate the OS
- *Kernel mode* allows essentially free reign to system structures and hardware
- Code that is executed in kernel mode must be very well written and tested
- Kernel code that is invoked via a *trap* must carefully validate the request

Traps Are Expensive

- On most systems, switching into and out of kernel mode is a relatively expensive operation
- Calls to kernel mode code should be kept to a minimum
- A great deal of system optimization work goes into dividing the operations that can be performed in user mode from the operations that must be performed in kernel mode

Resource Conflicts

- System processes – or system calls from user processes – may often access the same resource (for example, multiple processes may be updating PCBs or IOCBs)
- This introduces the possibility of errors
  - Process 1 is in the midst of making some updates to PCBs
  - In the midst of the changes, process 2 gets control – but the changes process 2 tries to make may conflict with the changes process 1 is making. Or, process 2 may run into errors accessing the PCBs, because they are in an inconsistent state.
- There are two ways to resolve this problem

Uninterruptible Code Segments

- Some systems allow small sections of code to be protected from interrupts
- Process 1 can safely update the PCBs, knowing that it cannot lose control until the changes are complete
- Risk: too many sections of uninterruptible code can effectively “lock out” other processes

Resource Locks

- Critical resources can be *locked* so that other processes cannot access the resource until the current process unlocks it
- To accomplish this, each critical shared resource is associated with a *semaphore* or *mutex* (mutual exclusion flag). Testing and setting is an atomic operation that cannot be interrupted.
- Once a process has the semaphore, it can safely use the resource until it releases the semaphore

Resource Locking Can Lead to “Deadlocks”

- Assume two processes, 1 and 2, each require the use of resources A and B – but request them in the opposite order!
- Process 1 requests the resource A semaphore – and gets it.
- At that point, Process 1 loses control. Process 2 then gets control, and requests the resource B semaphore – which it gets.
- This leads to a deadlock – neither Process 1 nor Process 2 can continue.

Process 1
- Has the Resource A semaphore
- Is waiting for the Resource B semaphore

Process 2
- Has the Resource B semaphore
- Is waiting for the Resource A semaphore

Deadlock Detection and Prevention

- Very hard to debug
- Because there may be only tiny “windows” that can result in a deadlock, they may occur rarely, and be very hard to locate and debug
- Deadlock detection and prevention is a major research area in computer science
- Current detection algorithms tend to be very resource intensive and have a large impact on system performance

(6) User Interaction (maybe)

- The early “proto operating systems” – Job Control Programs – accepted a limited set of *commands* directly from the input device (typically a card reader)
- By the late 1950s, most computers had a *console* where commands could be entered and messages could be output
- Today, user interactions are not part of the operating system
  - A *shell* accepts user command, and runs programs or calls system functions to execute them
  - A *windowing system* interacts with the user through a Graphic User Interface (GUI) and runs programs or calls system functions as required

What Does a Modern Operating System Do?

1. Process Management
   - Interprocess Communications
2. Input / Output (I/O) Management
3. Memory Management
4. File System Management
5. System Functions and Kernel Mode
6. User Interaction – (maybe)

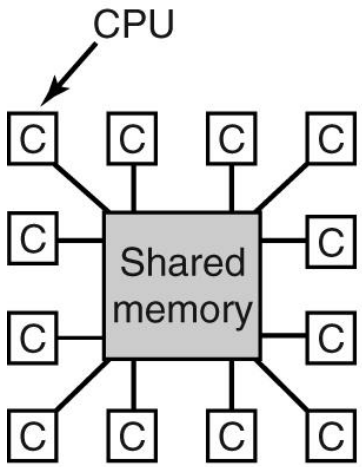
Types of Operating Systems

Embedded Systems (Generalizations)

- May not support a hardware MMU
  - Single address space
  - Limited physical memory
- Might not support a disk or mass storage device
  - Code is loaded from non-volatile memory
  - Swapping is not possible – limited to physical memory
- May still support a simple multi-process software architecture
- Little or no ability for user to add/modify software
  - Fewer security issues

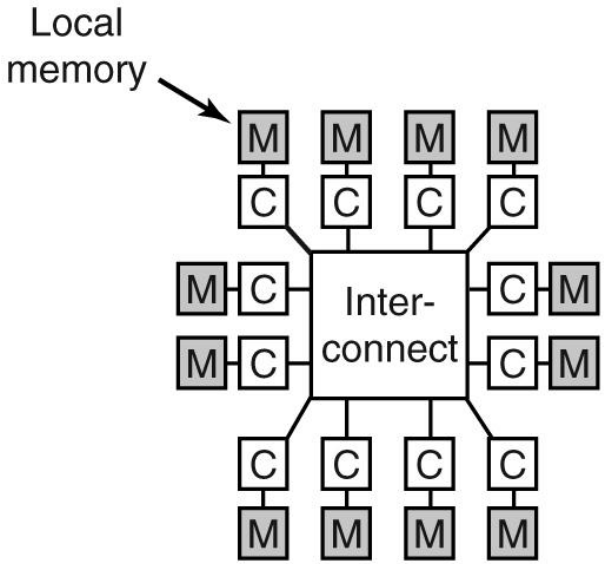
Multiprocessor Architectures

- The goal of Multiprocessor Architectures is to provide greater system throughput through parallelism

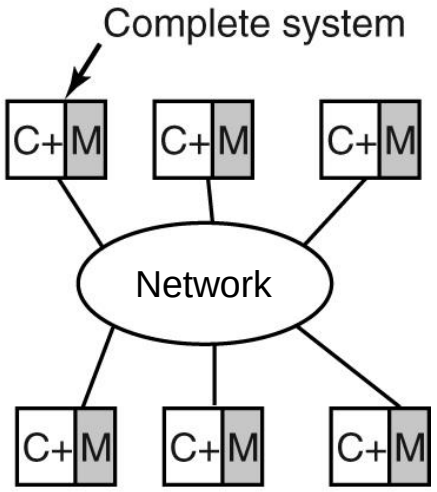


Tightly Coupled
(Shared Memory)

Multicore Architecture



Loosely Coupled
(Message Passing)



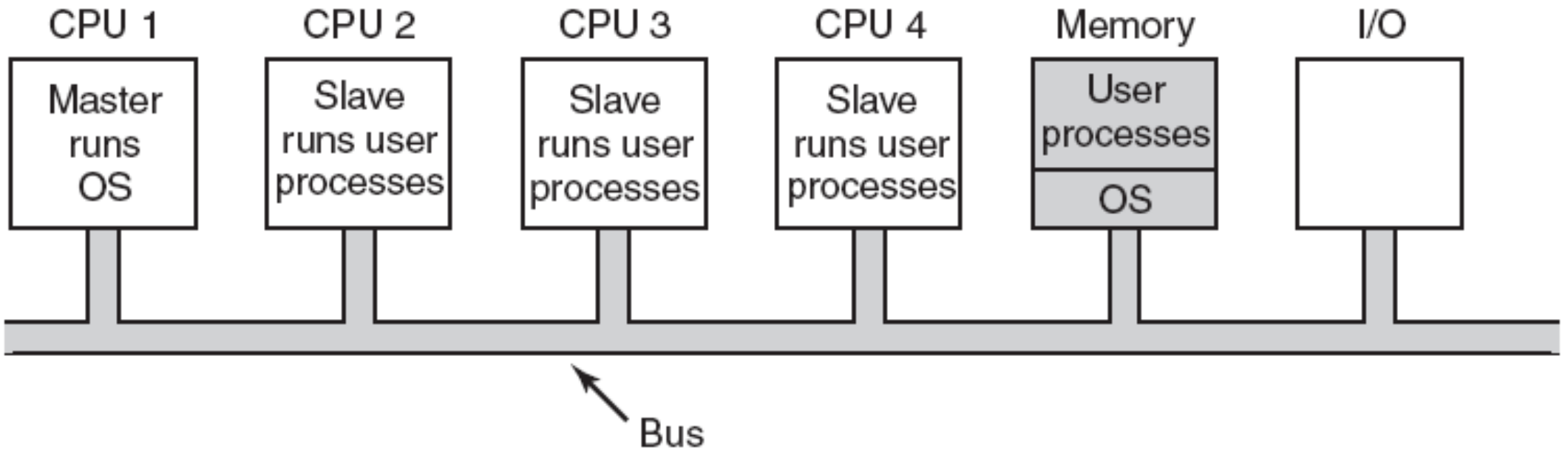
Networked
(Distributed)

Multiprocessor Operating Systems

- The core functions of the OS remain unchanged:
  - Process management
  - Scheduler, dispatcher
  - Memory Management
  - I/O Management
  - File Management
- The *resources* that they manage are expanded. For example, the **scheduler** can assign processors to multiple CPUs.

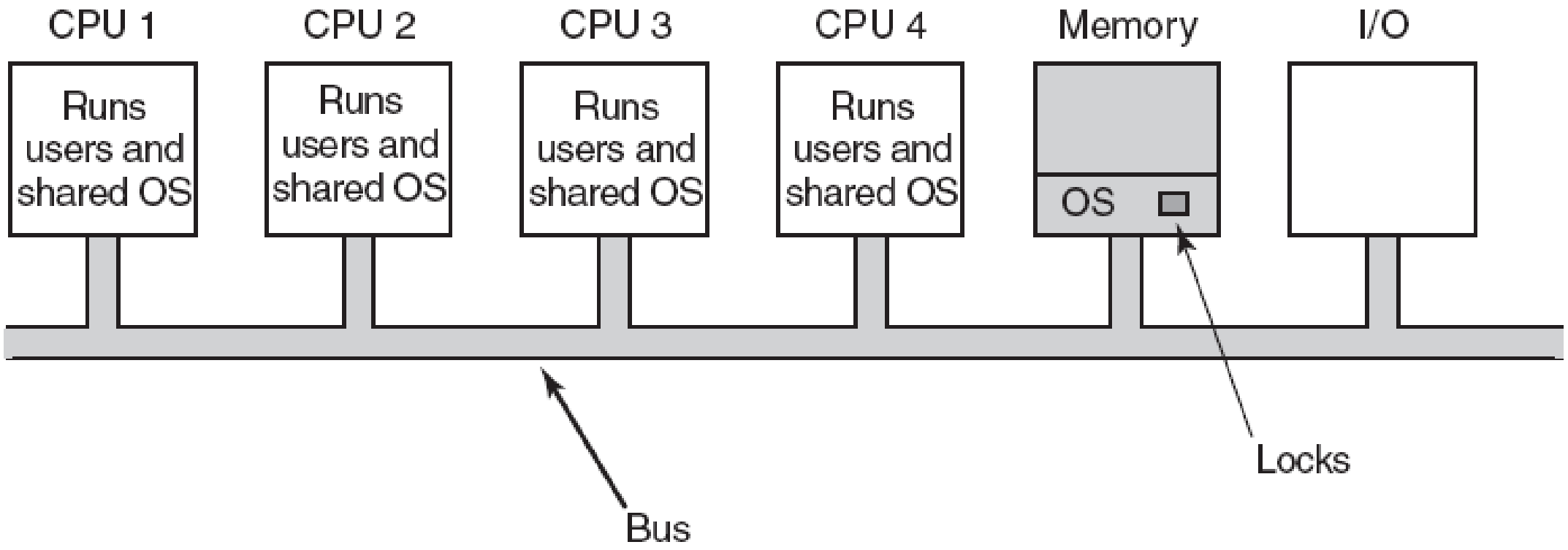
Multiprocessor OS: Master-Slave

- One copy of the OS runs on the Master CPU
- Slave processors make all OS calls to the Master CPU



Multiprocessor OS: Symmetric

- OS Processes can run on any CPU
- Requires memory “lock” on shared OS data structures



Breaking Out of the Box

Operating Systems Across “Computer” Boundaries

Network vs Distributed Operating Systems

- Both provide a means of sharing resources across a communications network
- **Network Operating System:** access to remote resources is explicit
- **Distributed Operating System:** access to remote resources is implicit – programs may not know about locality of references
- **Cloud Operating system:** manages the operation, execution and processes of *virtual machines*, virtual servers and virtual infrastructure, as well as the back-end hardware and software resources.

Network Operating Systems

- Almost all modern operating systems are ”networked”
  - POSIX, Mac OS, Windows, IOS, Android: YES
  - Embedded systems: MAYBE
- Remote resources are accessed via explicit communication protocols
  - FTP, SFTP, RPC, telnet, SSH
- Client-Server applications are supported through standard communication protocols (i.e. TCP/IP)
- The basic architecture of the Internet
- We’ll talk more about this architecture in the next class when we discuss “server” software

Distributed Operating Systems

- Users are not aware that they are on a network
  - Access to remote resources is similar to access to local resources
- Transparent access to remote files

Distributed OS: Access to Remote Files

- **Remote File Access:** Each file access traverses the network
- **File Migration / Data Migration:** File is copied to local machine, accessed, then copied back (requires a file locking mechanism)
- **Program Migration:** Program is executed on remote machine, where is has direct access to the file

Distributed OS: Process Migration

- Execute an entire process, or parts of it, on a remote system
- Allows:
  - Load balancing
  - Access to special purpose hardware or software

Cloud Operating System

- A cloud OS virtualizes system resoruces.
- A cloud OS is a distributed OS that allows multiple operating systems to coexist on a single “system,” or a single OS to span multiple “systems.”
- A web OS is a specialized type of cloud OS. All resources are available through a web browser
- Similar to a concept pioneered at Sun Microsystems ca. 2005
  - “The network is the computer” – Scott McNealy

Object Oriented Operating Systems

- Extend the object oriented programming paradigm into the OS… the operating system manages resources as objects
- An active research area from the mid-1980s through the mid-1990s
- There are no mainstream examples today

For Next Week

- Log in to Canvas and complete Assignment 6