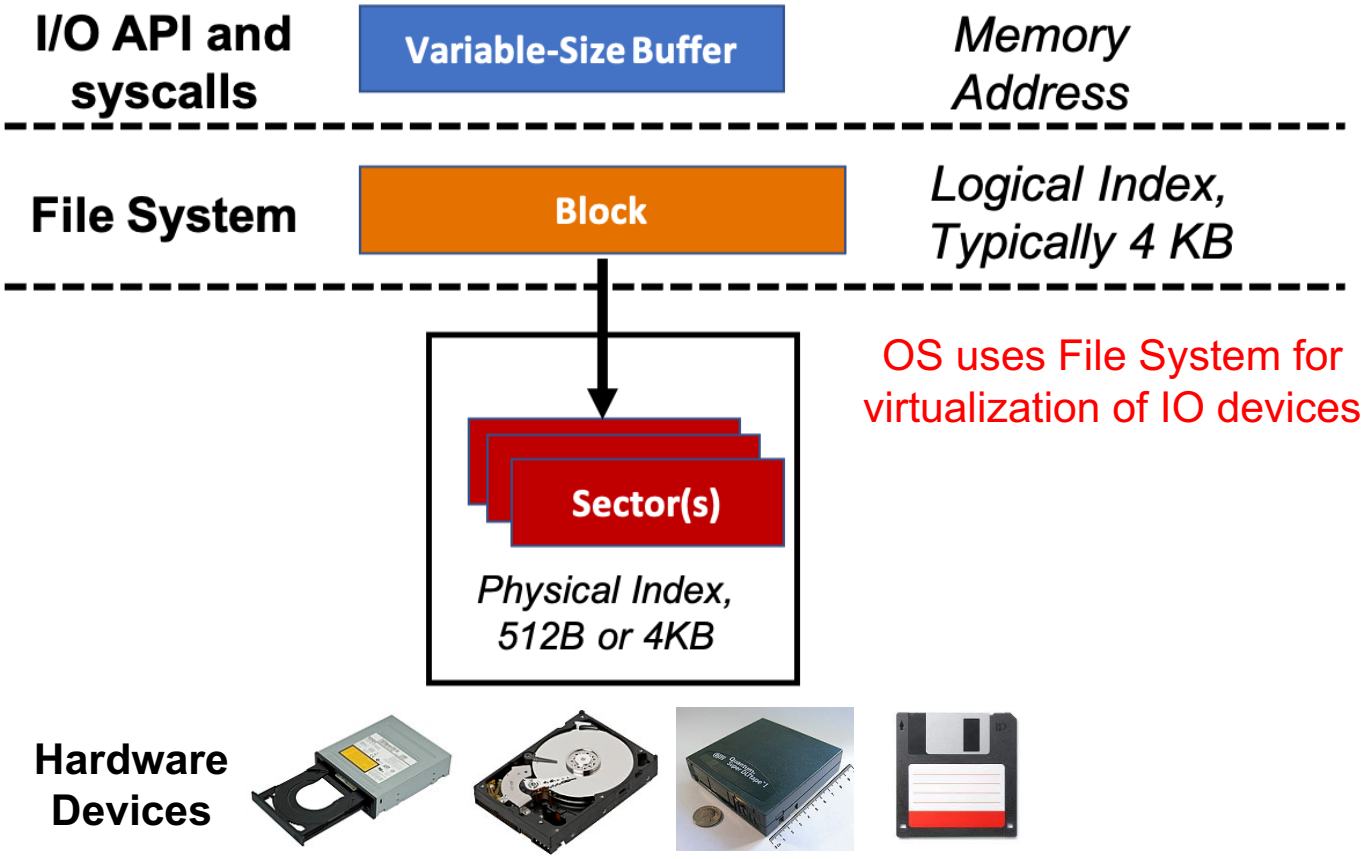
Lecture 24: Introduction to File Systems

Vivek Kumar
Computer Science and Engineering
IIIT Delhi
vivekk@iiitd.ac.in

Today's Class

- Quiz-4 (last remaining quiz)
- Persistent storage (IO) devices
- Virtualization of IO devices

File System

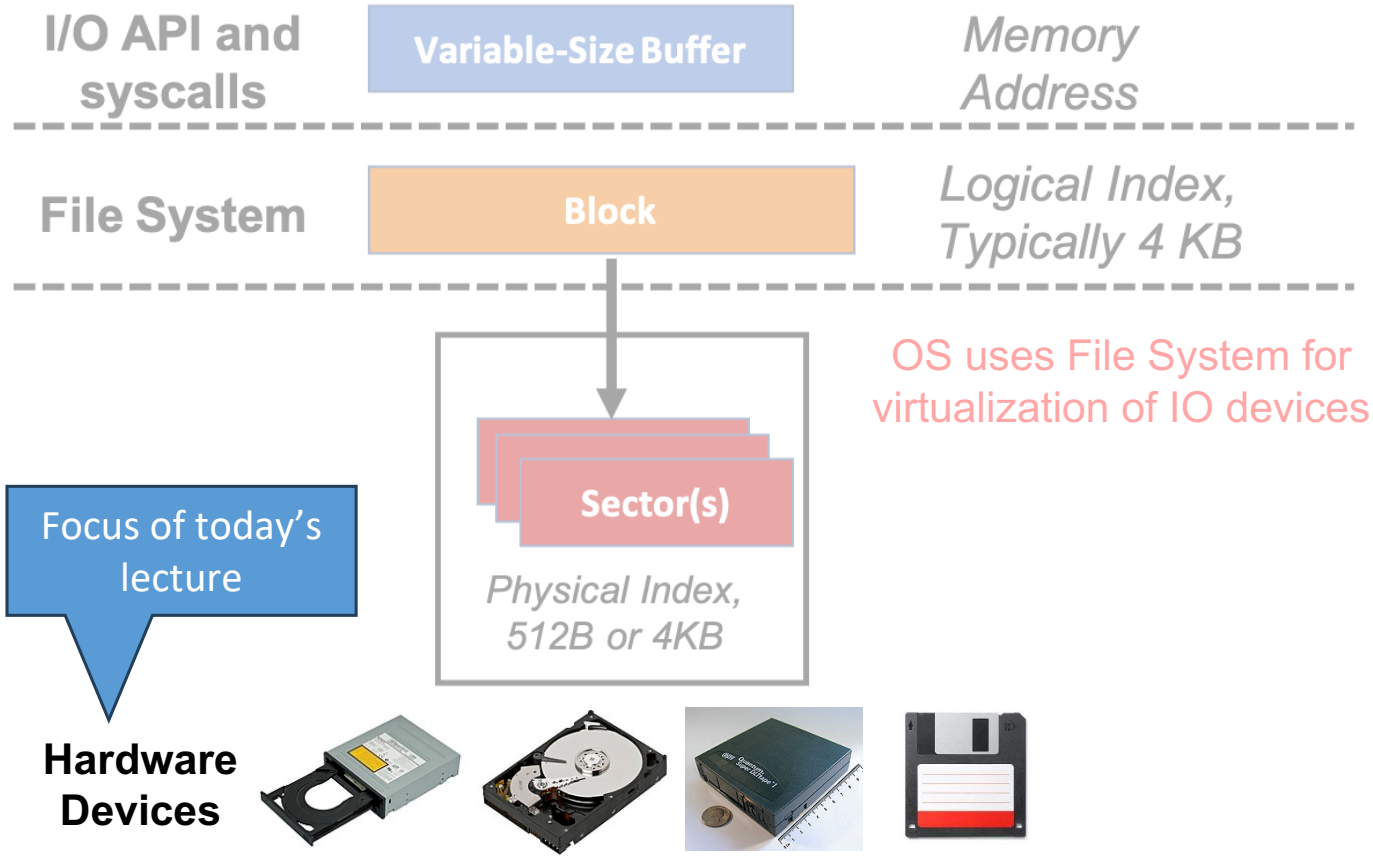


- User's view of File:
  - Durable Data Structures
- System's view of File (system call interface):
  - Collection of Bytes (UNIX)
  - Oblivious to specific data structures user wants to store
- System's view of file (inside OS):
  - File is a collection of blocks

Role of File System

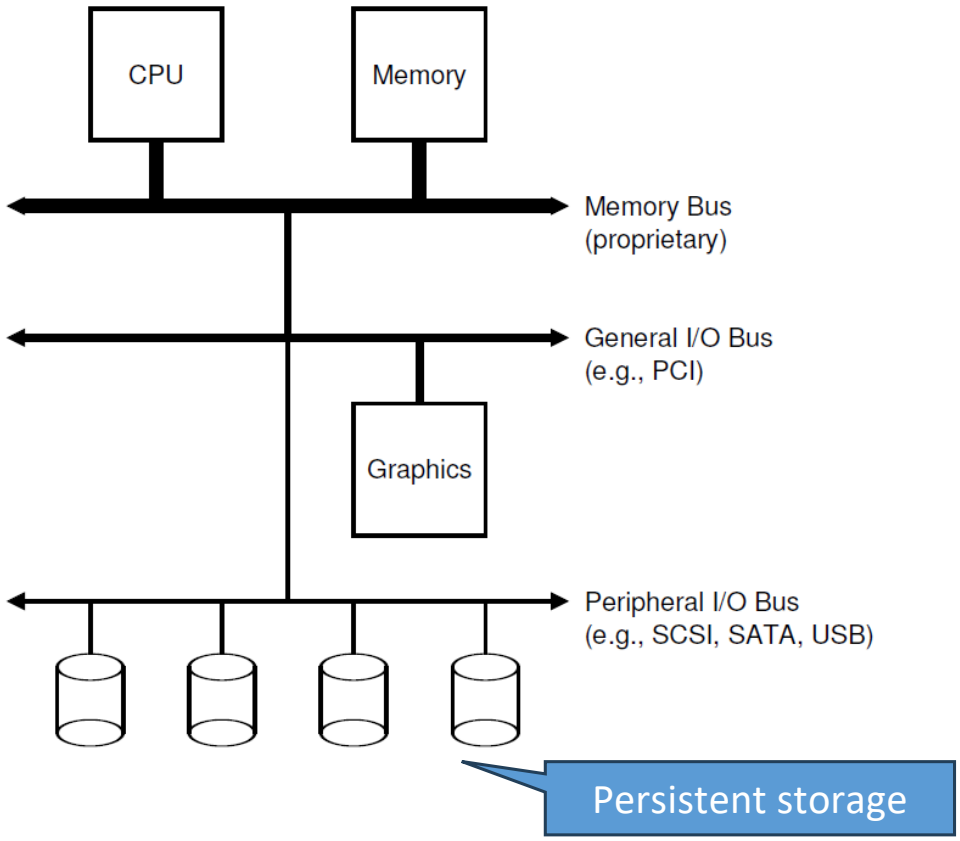
- Virtualization of IO devices
  - Provide abstraction over a variety of IO devices by providing a uniform interface for accessing data
- Provide support for crash recovery
  - The data must be available even after a reboot or accidental power cycle
- Hide the access latency
  - Disk access is several orders of magnitude slower than DRAM access

File System



- User's view of File:
  - Durable Data Structures
- System's view of File (system call interface):
  - Collection of Bytes (UNIX)
  - Oblivious to specific data structures user wants to store
- System's view of file (inside OS):
  - File is a collection of blocks

Computer System Architecture

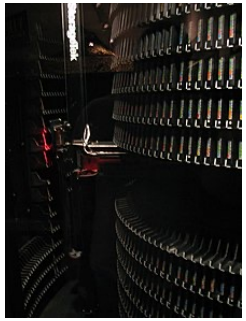


- I/O devices connect to the CPU and memory via a bus
  - High speed bus, e.g., PCI (Peripheral Component Interconnect)
  - Other: SCSI (Small Computer System Interface), USB, SATA
- "Port" is the point of connection to the system

Persistent Storage (IO Devices)

Hard disk

Tape cartridge



Tape library

DVD drive

Floppy drive

USB drive

Image source: Google images

- Persistence refers to the characteristic of state that outlives the process that created it
  - Otherwise, data can only reside in DRAM which will be lost at power cycle
- Persistent storage device is any data storage media that retains the data even after a power cycle

Prototype of a Simple IO Device

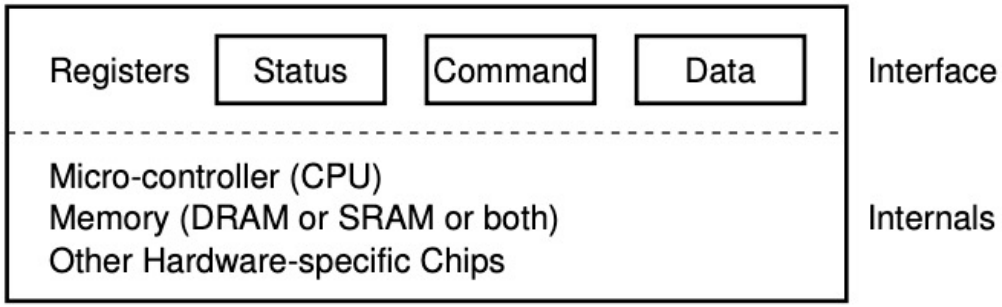
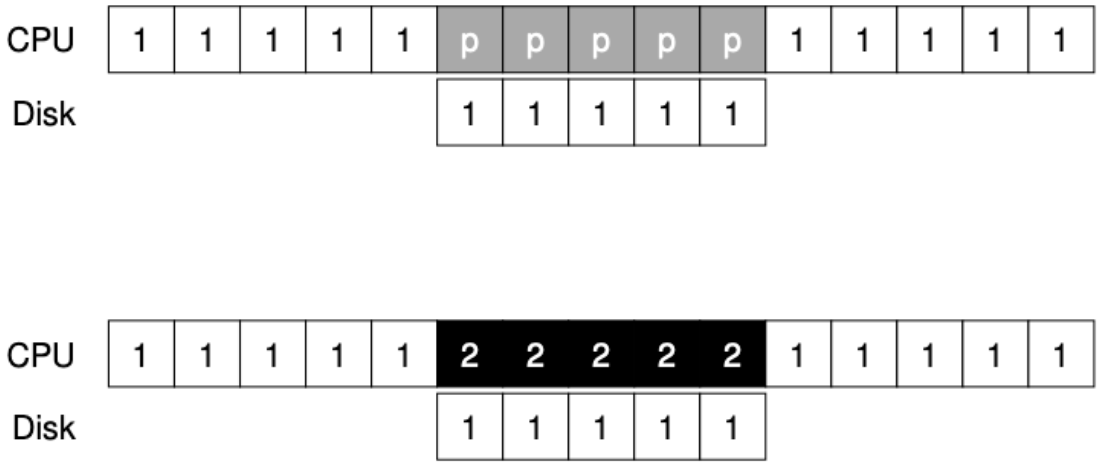


Figure 36.3: **A Canonical Device**

```
While (STATUS == BUSY)
    ; // wait until device is not busy
Write data to DATA register
Write command to COMMAND register
    (starts the device and executes the command)
While (STATUS == BUSY)
    ; // wait until device is done with your request
```

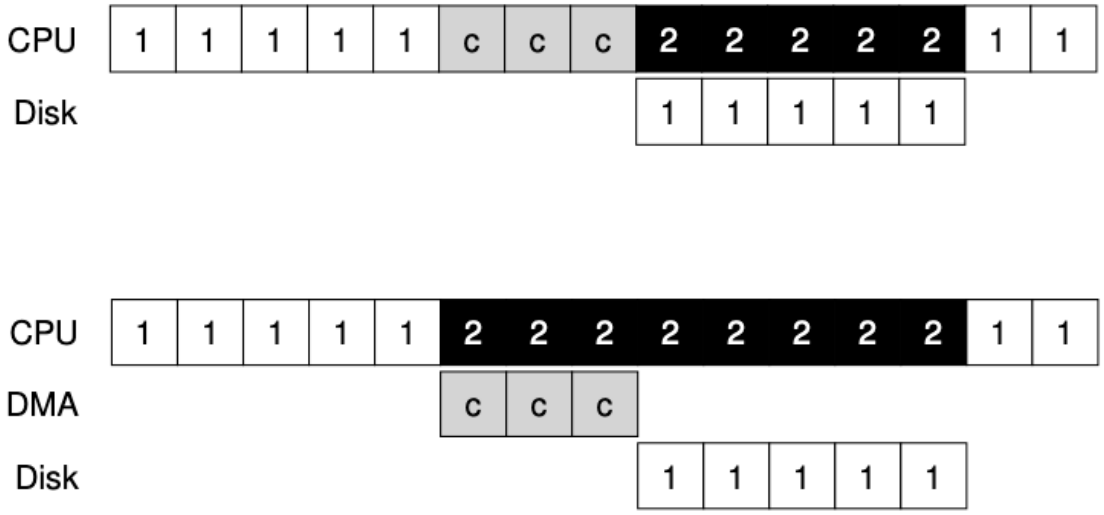
- OS interact with the IO device via a software layer called as device driver
- Devices expose an interface of memory registers
  - Current status of device
  - Command to execute
  - Data to transfer
- The internals of device are usually hidden
- Device interaction in four steps
  - Polling for the device to be ready
  - Sending data to data register
  - Inform the completion of data transfer by writing to command register
  - Polling for device for completion
- Do you see any issues here?
  - Wastage of CPU in polling

Avoiding Polling Overheads



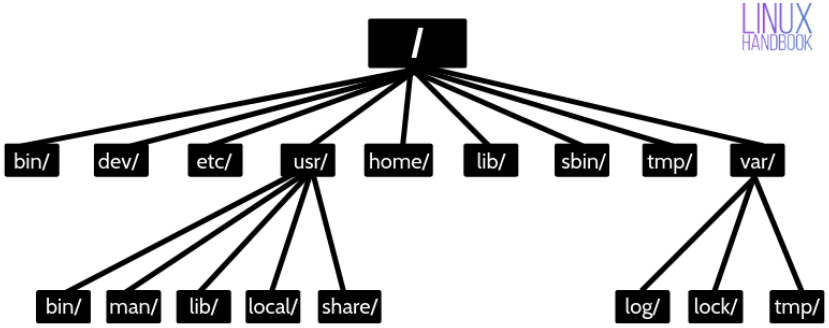
- Polling
  - CPU cycles wasted as CPU simply spins without doing any other job
- Interrupt
  - OS can put the polling process in wait queue and bring another process from ready queue
  - Interrupt sent to blocked process by the IO device upon completion of IO
- Hybrid
  - Start with a poll and if IO did not complete quickly then wait for interrupt

Direct Memory Access (DMA)



- CPU cycles wasted in copying data to/from device
- Instead, a special piece of hardware (DMA engine) copies from main memory to device
  - CPU gives DMA engine the memory location of data
  - In case of read, interrupt raised after DMA completes
  - In case of write, disk starts writing after DMA completes

Virtualization of IO Devices



Linux directory structure

- OS provides abstraction over different types of IO devices by having a uniform interface of files and directories
- **File** is the smallest unit of storage device as seen by the user
  - User cannot write data to storage device unless data is within a file
  - File is an abstract data type with basic set of operations
    - Create, Write, Read, Delete, Reposition (seek), Truncate, etc.
  - OS refers to file name as its "**inode**" number (Unix based OS)
- **Directory** is also a file but with the information on how to find other files
  - It also has its **inode** number

File Descriptor and File Table (Unix)

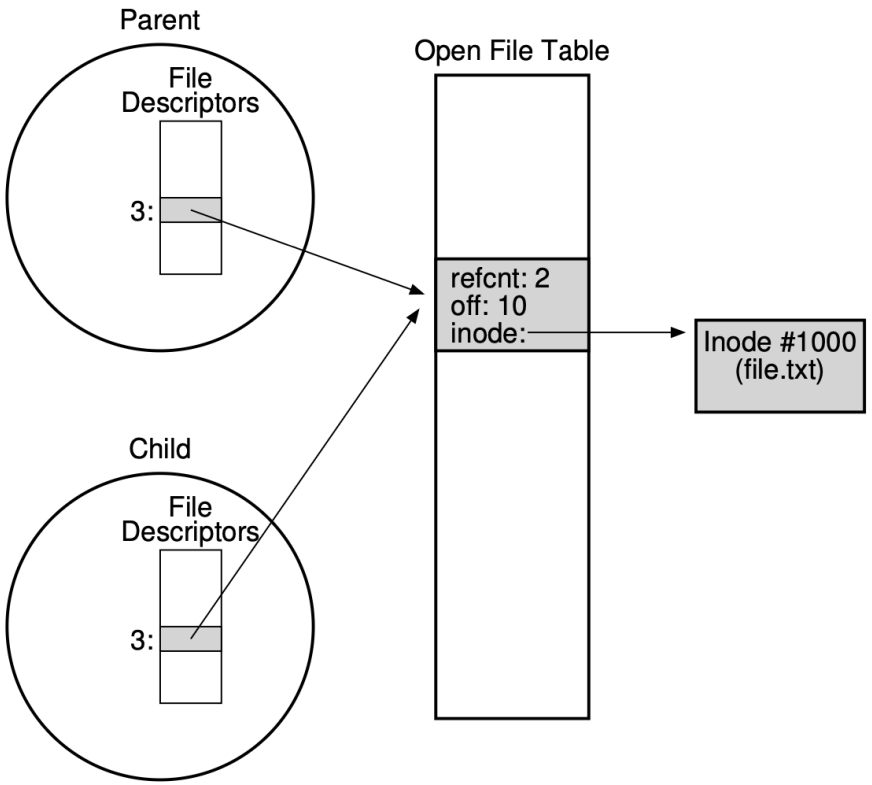


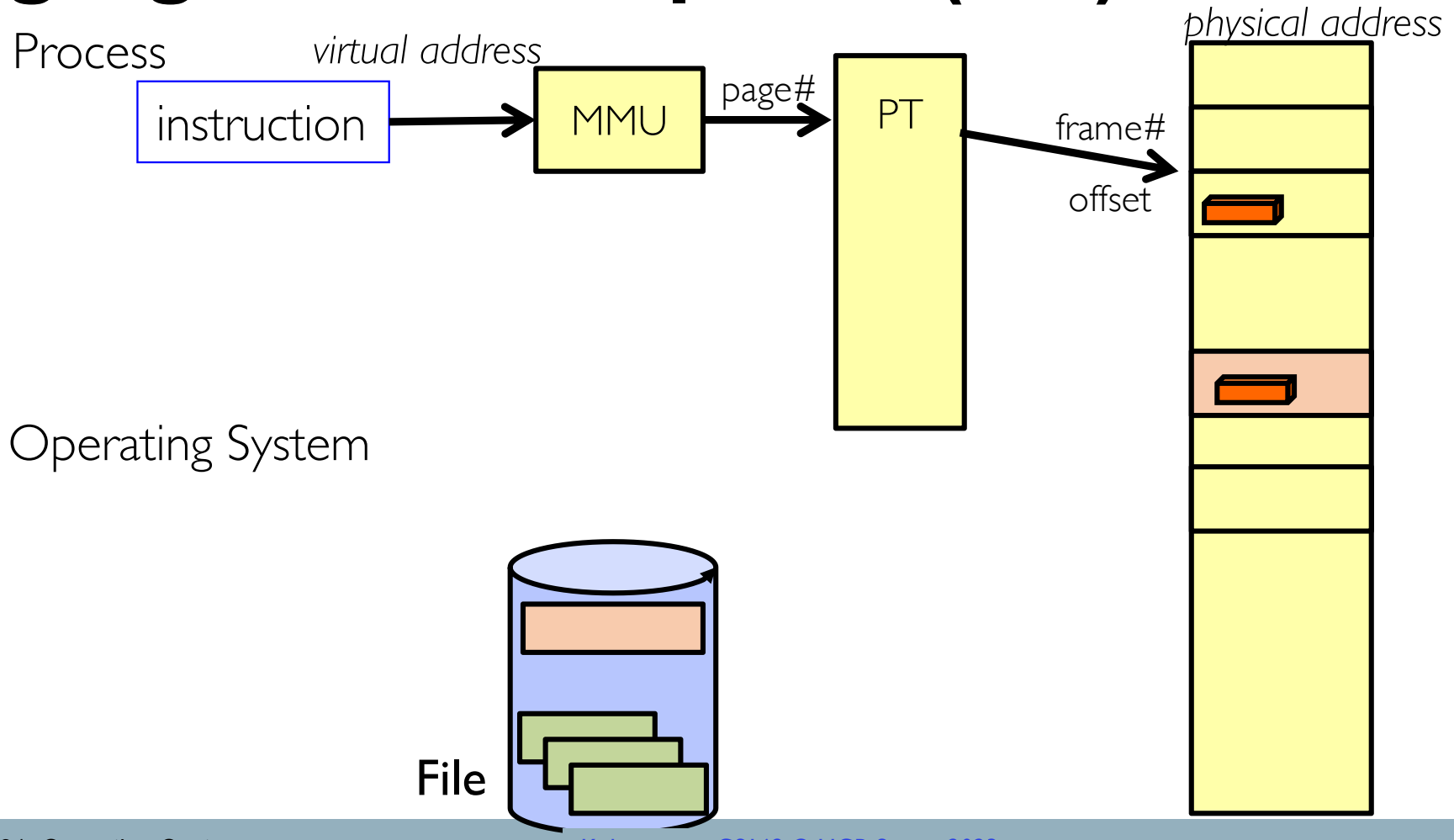
Figure 39.3: **Processes Sharing An Open File Table Entry**

- Each process has an array of open **file descriptors**
- Each file descriptor points to a entry inside system wide open **file table**
- "**open**" call will create a new entry inside file table if no entry found, otherwise a copy of the previous entry is added, but with its file offset reset to zero
- Each entry inside file table keeps track of total processes **sharing** this file descriptor (e.g., sharing due to parent-child relation), current offset inside file, inode pointer to the file, file permissions etc.)
- File table entry is removed once the reference count reaches zero

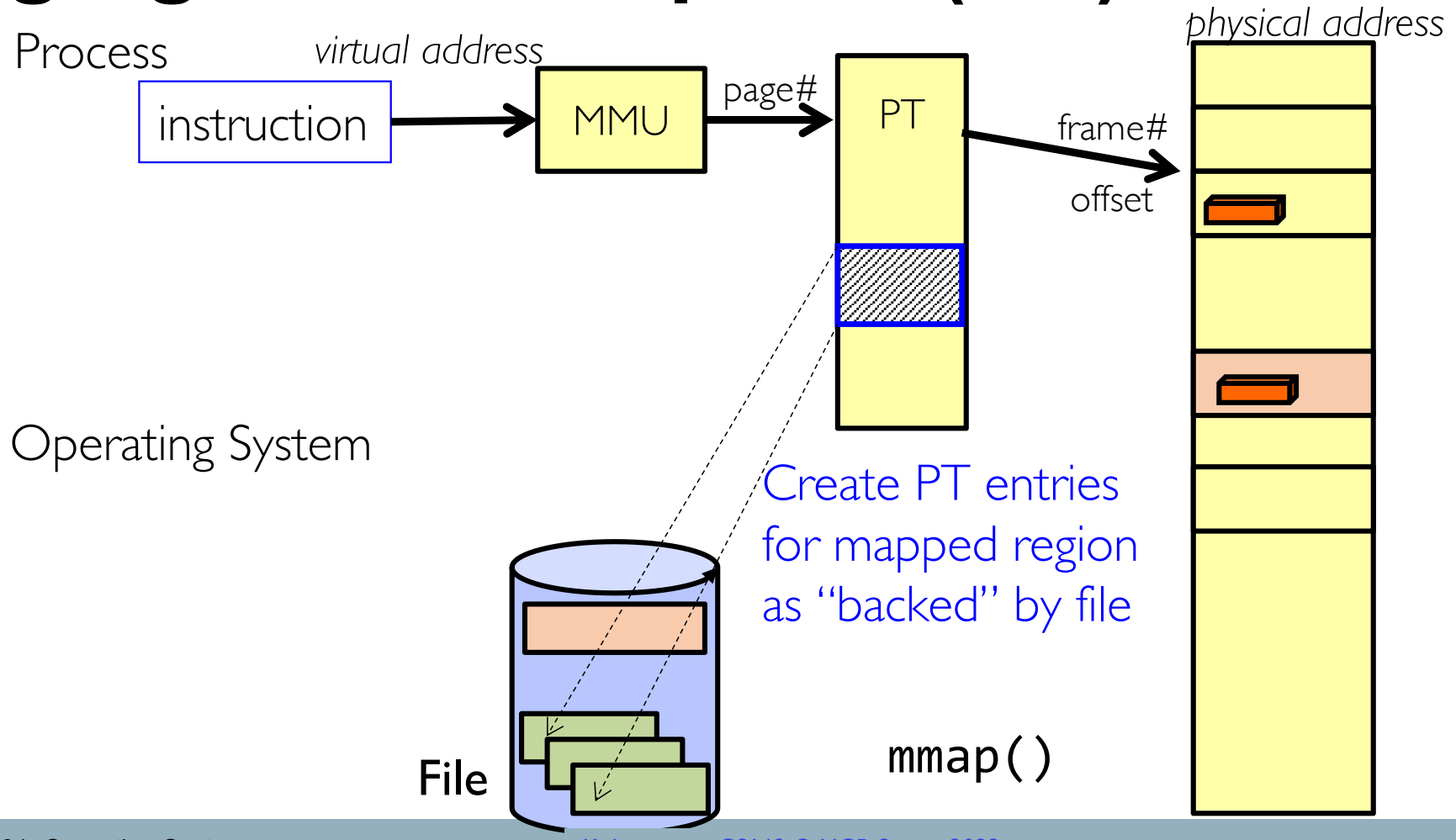
Types of IO

- File IO
  - Explicit transfer between buffers in process address space to regions of a file
  - Overhead due to multiple copies in memory and system calls (read, write, etc.)
- Memory mapped IO
  - **mmap** file directly into an empty region of a process address space
    - No need to issue system calls for IO (read, write, etc.)
  - ELF file is treated this way when we execute a process (recall your assignment-4)

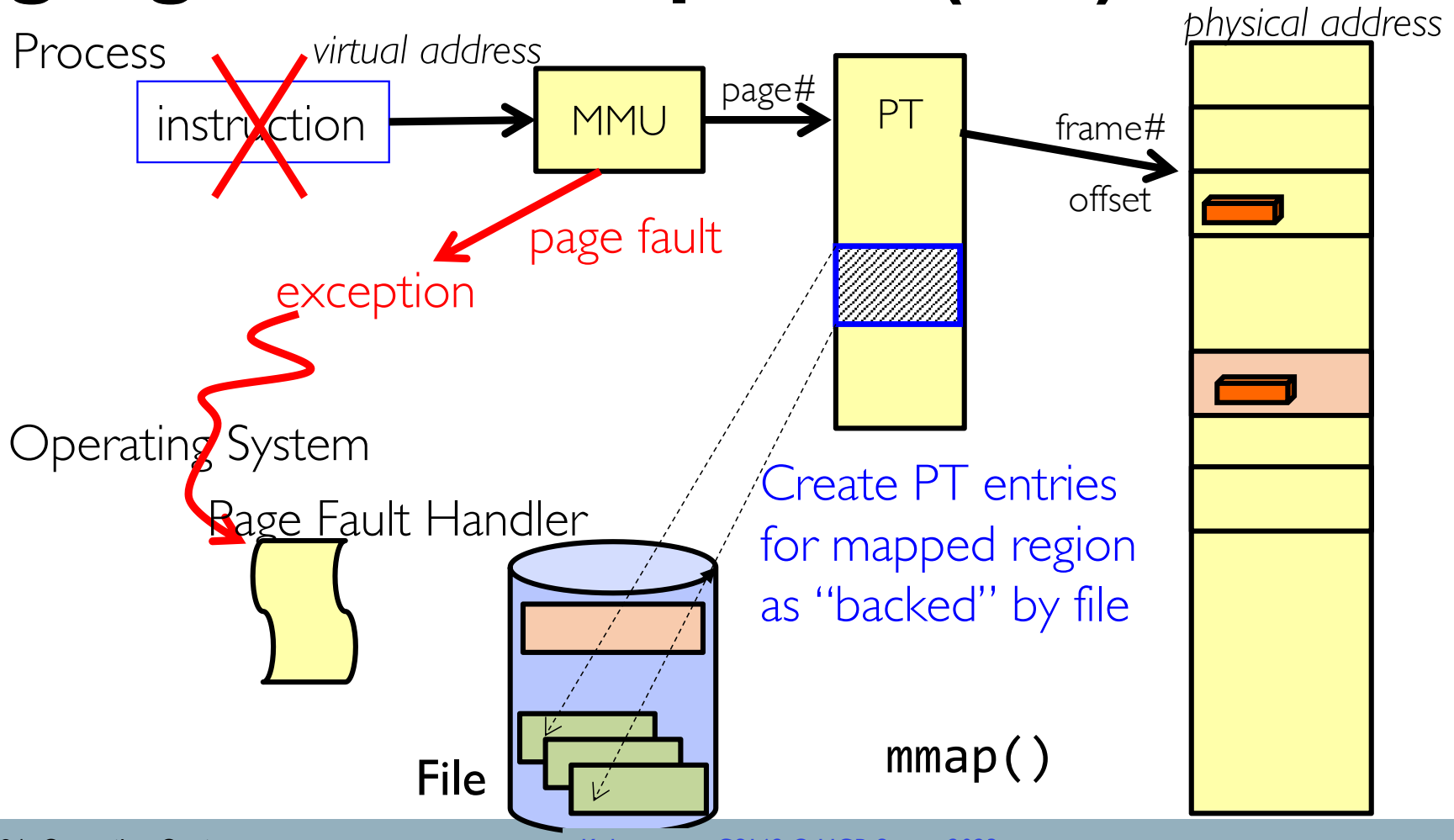
Paging With mmap File (1/6)



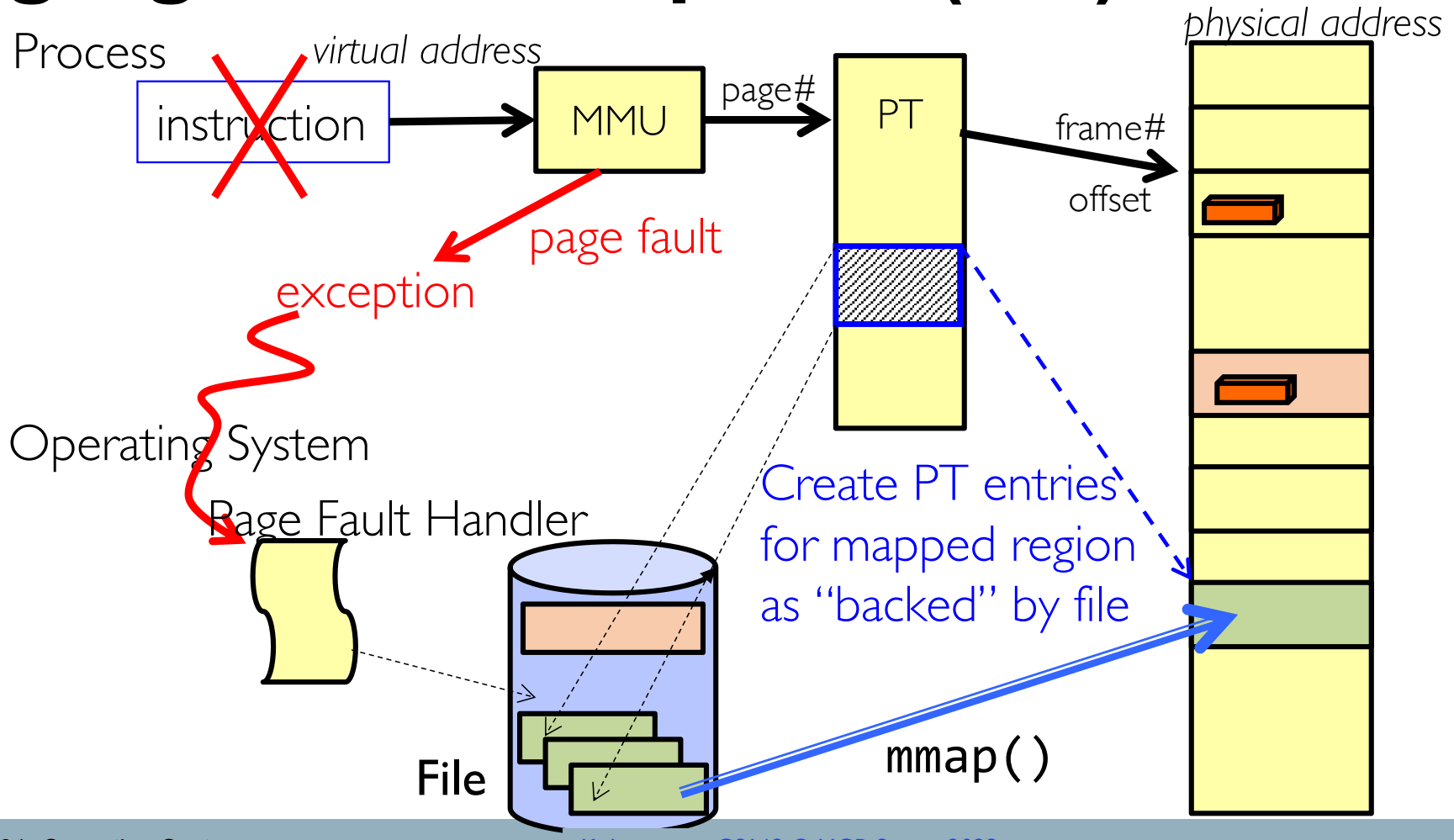
Paging With mmap File (2/6)



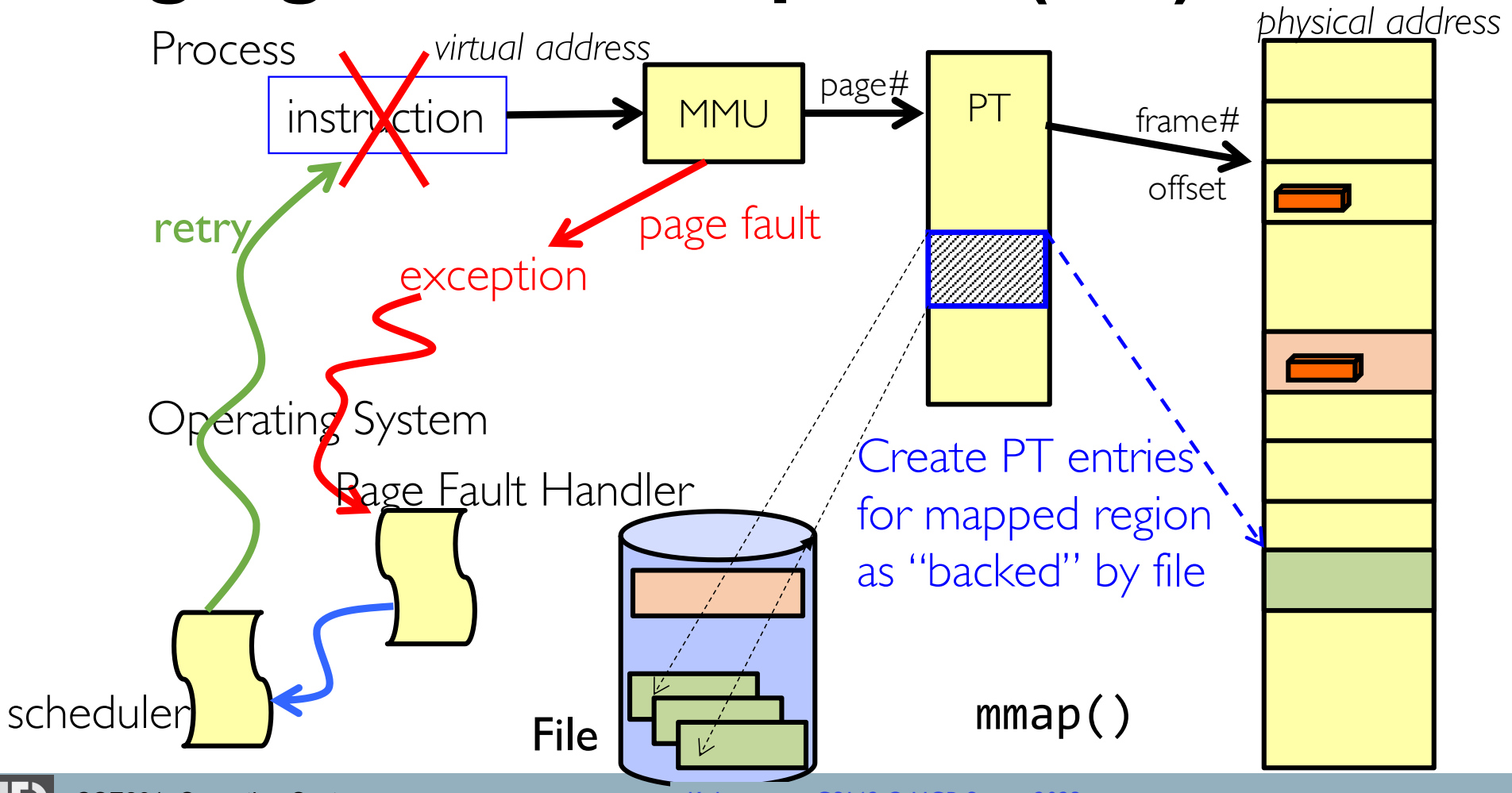
Paging With mmap File (3/6)



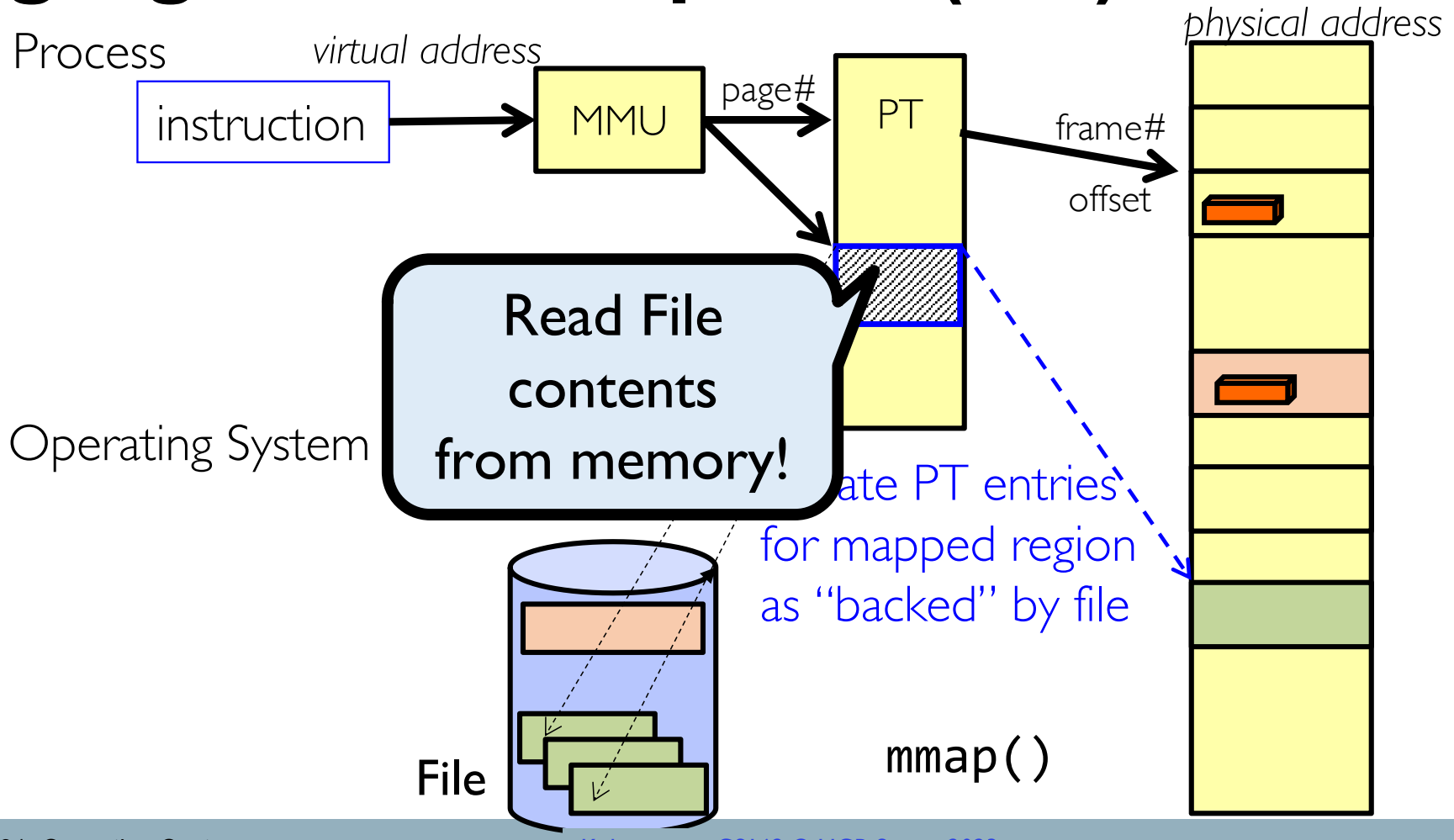
Paging With mmap File (4/6)



Paging With mmap File (5/6)



Paging With mmap File (6/6)



Next Class

- File system implementation