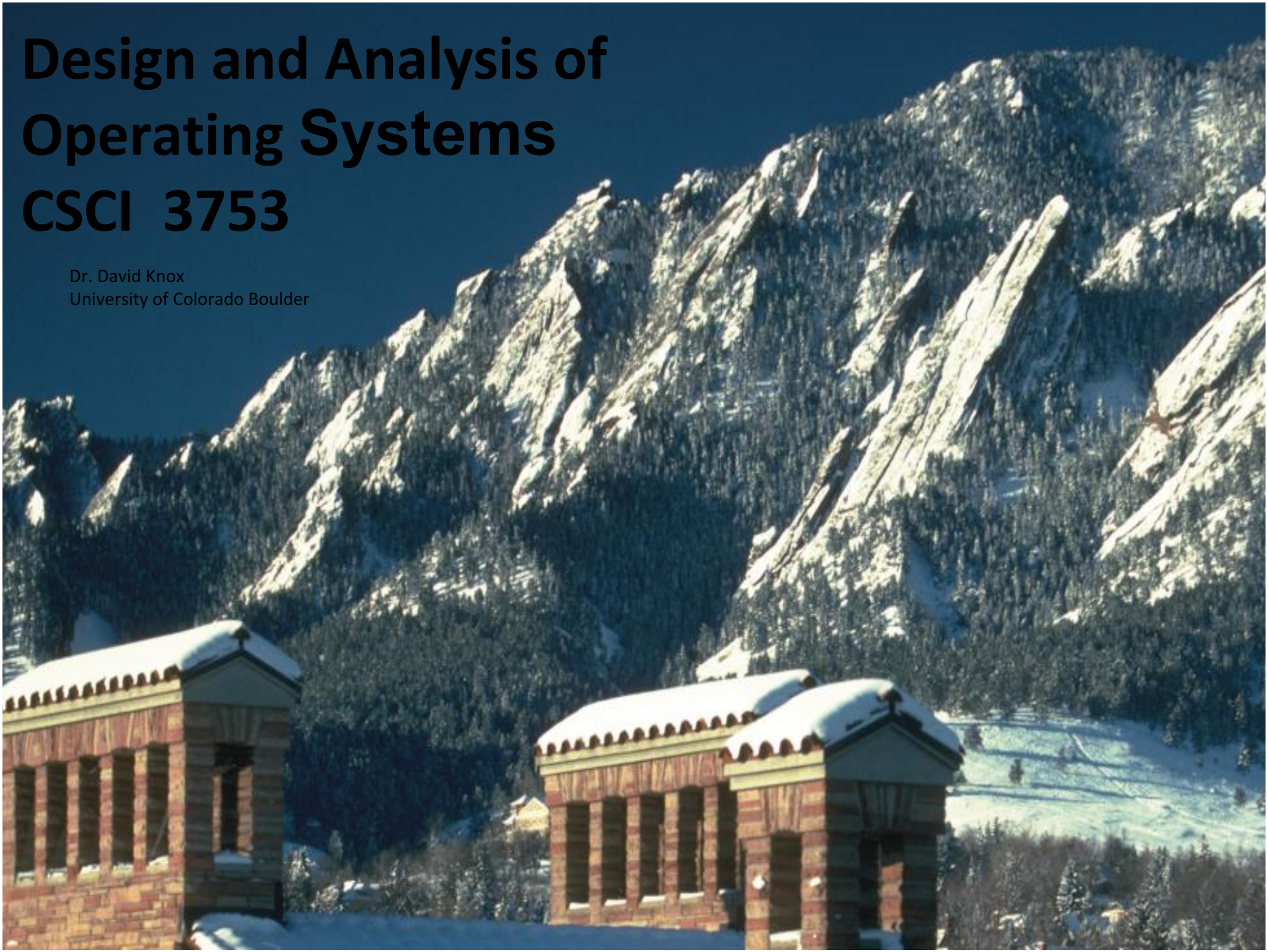
Design and Analysis of Operating Systems CSCI 3753

Dr. David Knox
University of Colorado Boulder



Department of Computer Science
UNIVERSITY OF COLORADO **BOULDER**

Design and Analysis of Operating Systems CSCI 3753

Memory Management
Memory Mapped Files

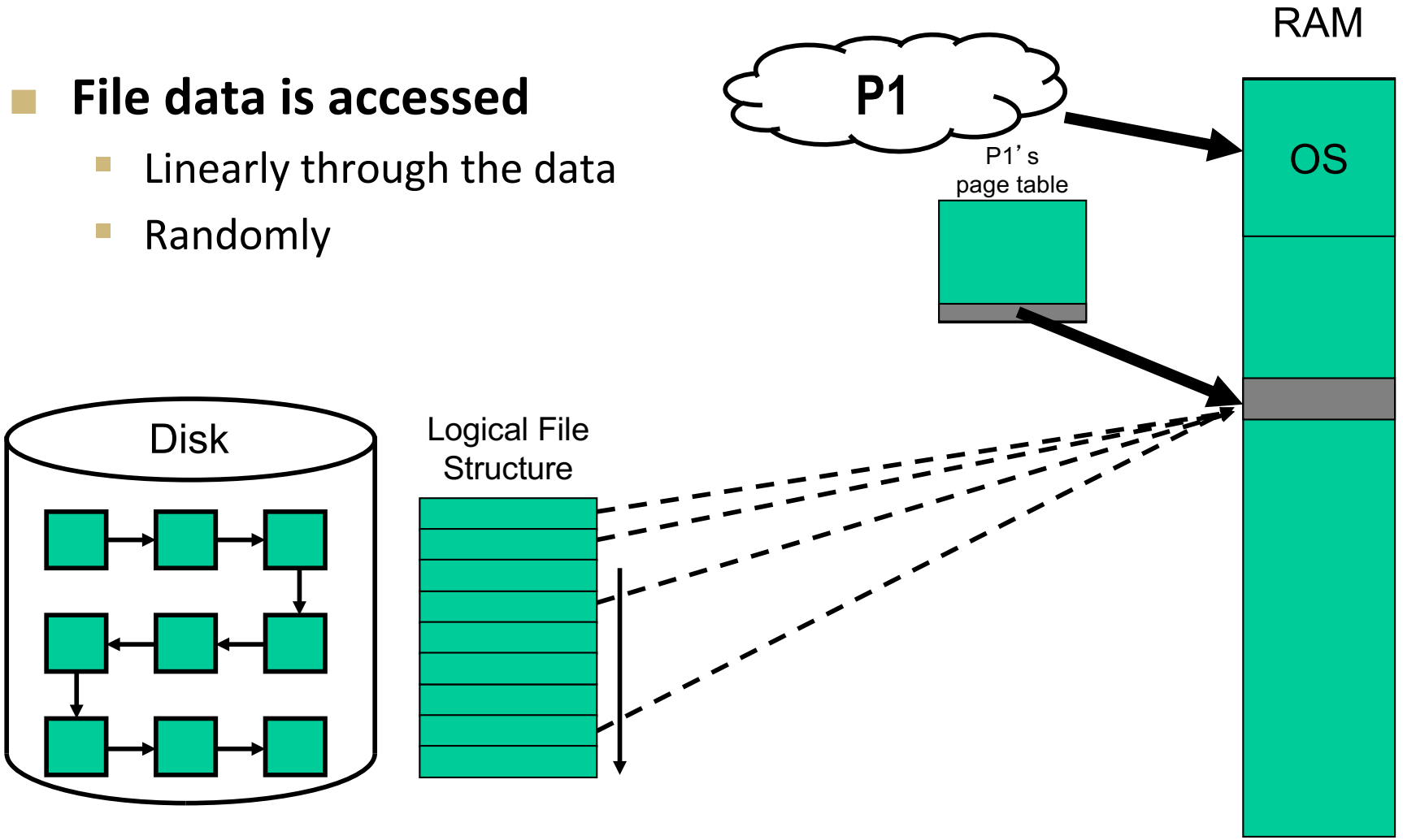
Dr. David Knox

University of
Colorado Boulder

Memory Management
Memory Mapped Files

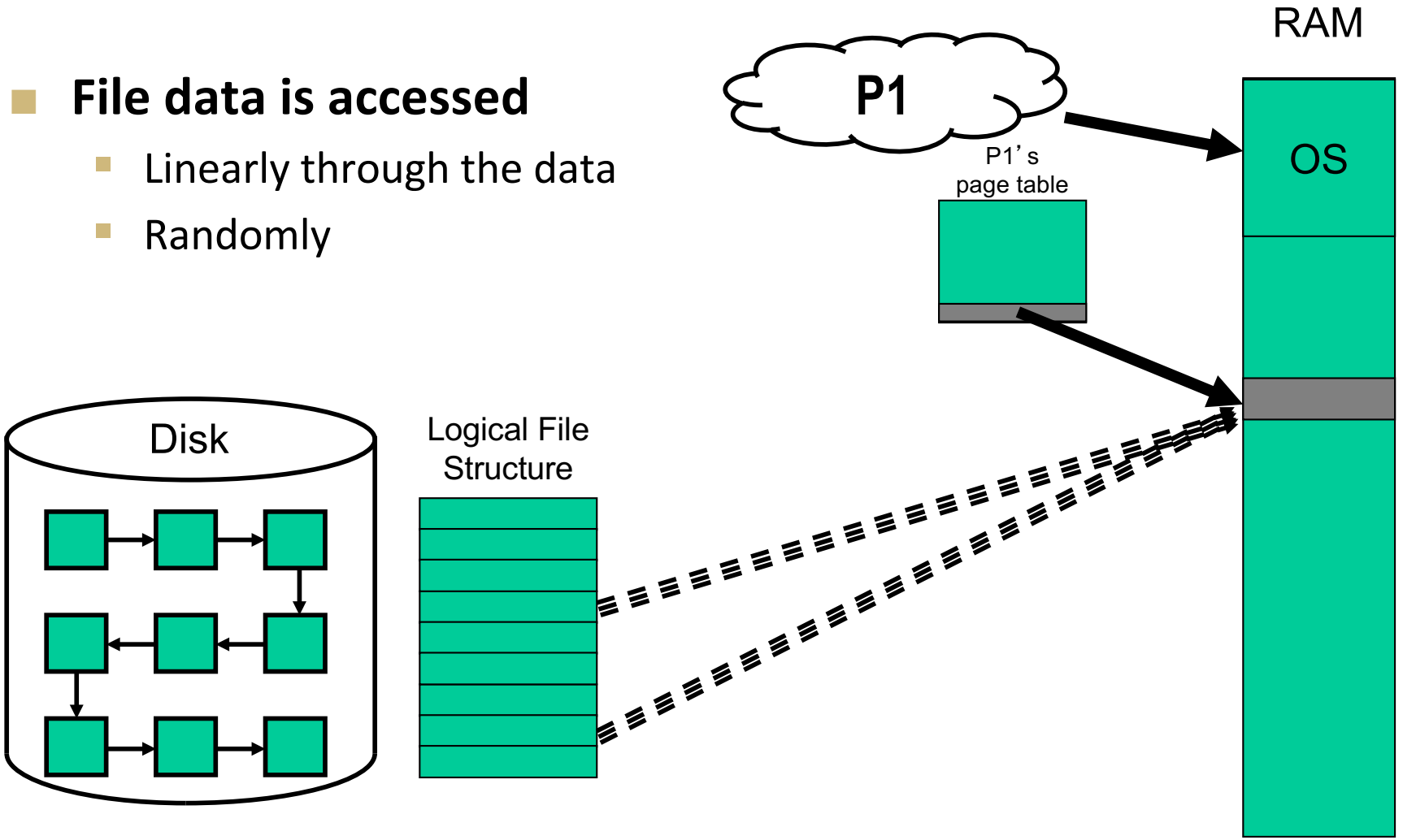
Memory-Mapped Files

- **File data is accessed**
  - Linearly through the data
  - Randomly



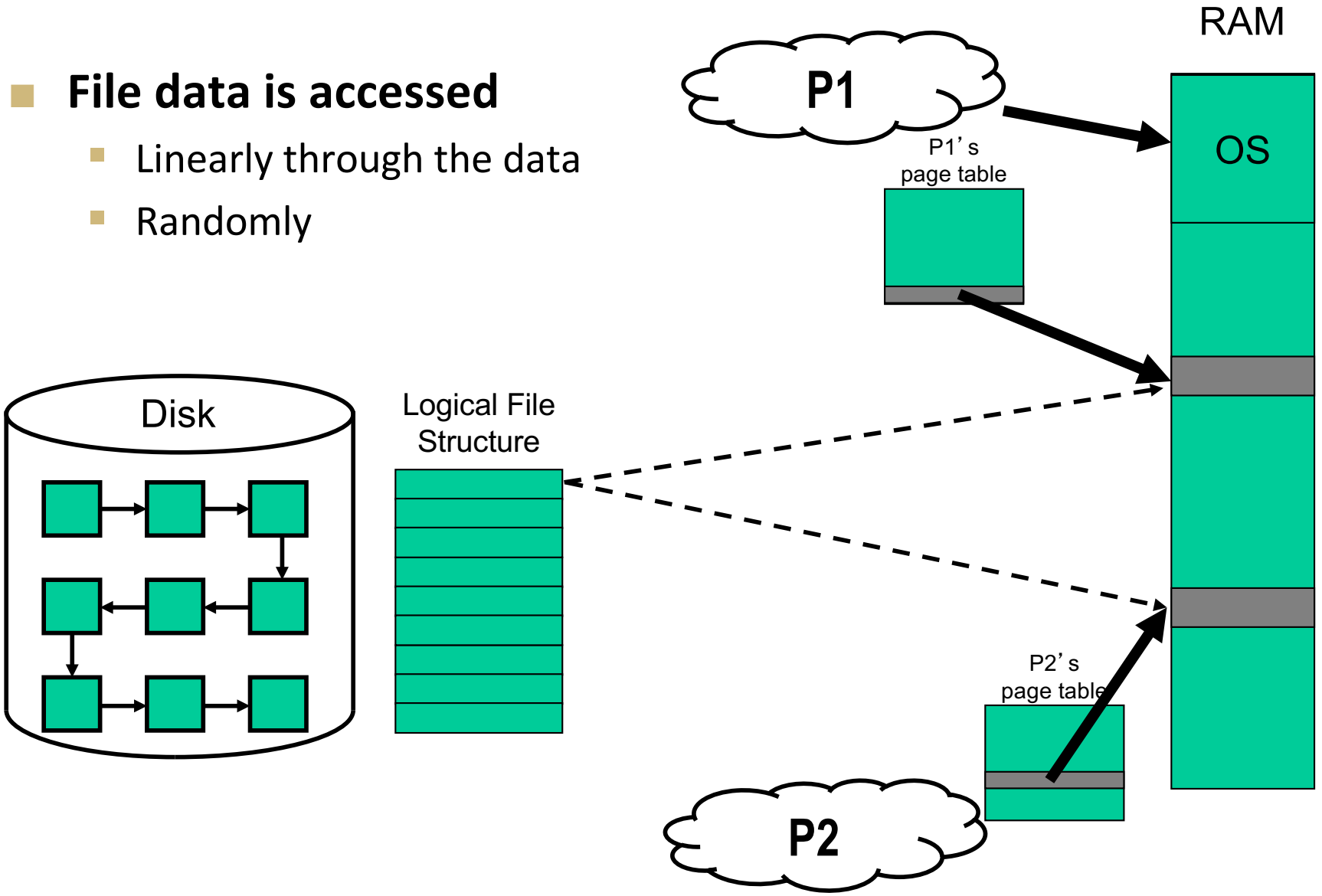
Memory-Mapped Files

- **File data is accessed**
  - Linearly through the data
  - Randomly



Memory-Mapped Files

- **File data is accessed**
  - Linearly through the data
  - Randomly



Memory-Mapped Files

- **Map some parts of a file on disk to pages of virtual memory**
  - Each part is in a frame of memory mapped to the process
  - Normally, each read/write from/to a file requires a system call plus file manager involvement plus reading/writing from/to disk
    - Programmer can improve performance by copying part of or entire file into a local buffer, manipulating it, then writing it back to disk
      - this requires manual action on the part of the programmer
  - Instead, it would be faster and simpler if the file was loaded into memory
    - (almost) transparently so that reads/writes take place from/to RAM

- **Use the virtual memory mechanism to map (parts of) files on disk to pages in the logical address space**

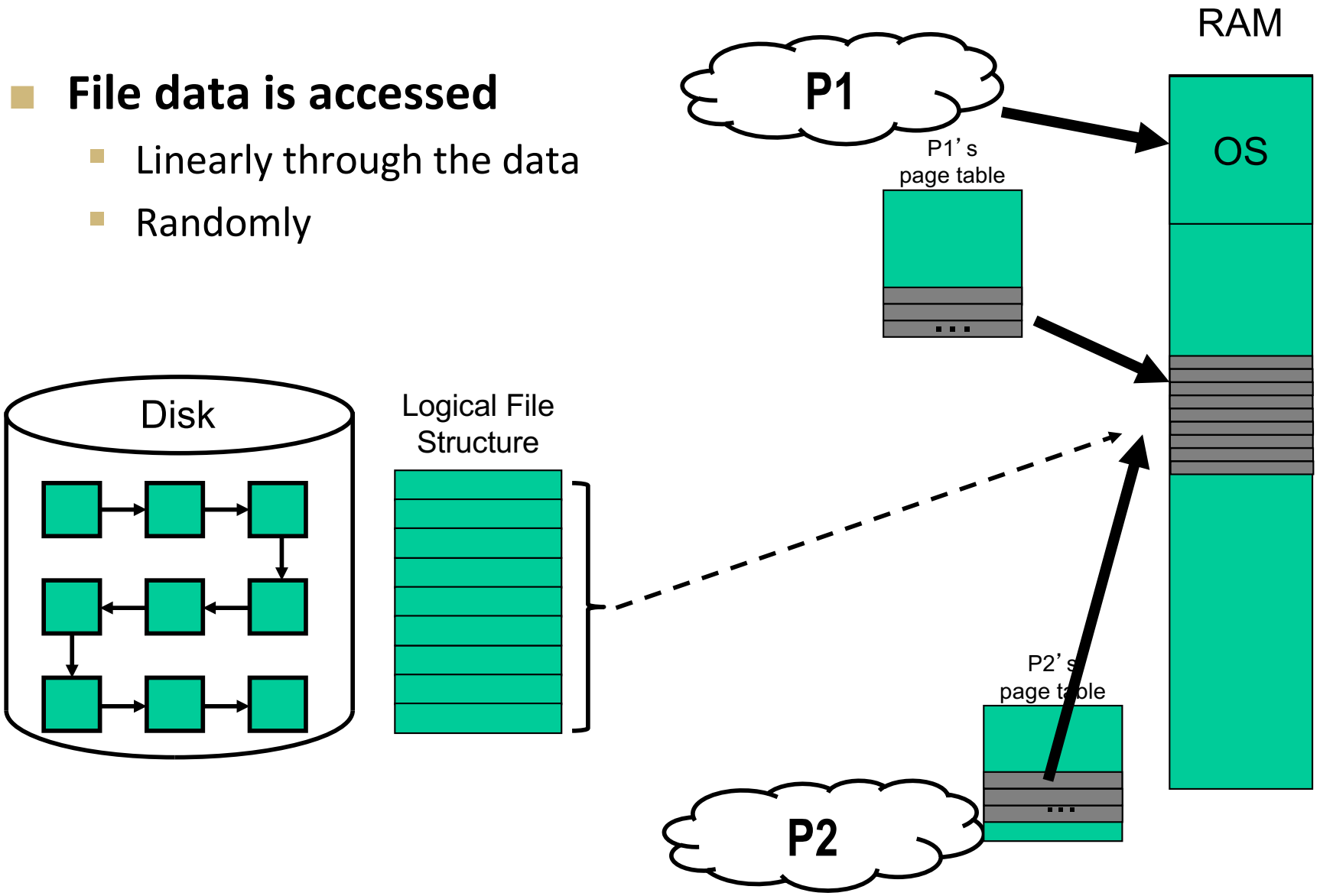
Memory-Mapped Files

Steps for memory-mapping a file:

1. **Obtain a handle to a file by creating or opening it**
2. **Reserve virtual addresses for the file in your logical address space**
3. **Declare a (portion of a) file to be memory mapped by establishing a mapping between the file and your virtual address space**

Memory-Mapped Files

- **File data is accessed**
  - Linearly through the data
  - Randomly



Memory-Mapped Files

Steps for memory-mapping a file: (cont.)

4. **When file is first accessed, it's demand paged into physical memory**
5. **Subsequent read/write accesses to (that portion of) the file are served by physical memory**

Memory-Mapped Files

- **Advantages of memory-mapping files:**
  - **All subsequent reads/writes from/to a memory file are fast**
    - After the first accesses
    - No longer use file system to read/write. Instead use MMU
  - **Multiple processes can map the same file concurrently**
    - Shares file efficiently
    - In Windows, this mapping mechanism is also used to create shared memory between processes and is the preferred memory for sharing information among address spaces
    - on Linux, there are separate mmap() and shared memory calls
      - e.g. shmget() and shmat()

Memory-Mapped Files

- **Remember that the entire file need not be mapped into memory, just a portion or "view" of that file**
  - in previous figure, the in memory data could represent just a portion of some larger file
  - File data can also be subdivided into pages, each of which is mapped to a separate page in memory by the page table
- **File System has to be consulted**
  - Step 4 needs to determine where on disk a view of a file is located
  - File Open is an expensive operation

Memory-Mapped Files

- **Writes to a file in memory can result in momentary inconsistency between the file cached in memory and the file on disk**
  - Could write changes immediately (synchronously) to disk
  - Or delay and cache writes (asynchronous policy)
    - wait until OS periodically checks if dirty/modify bit has been set
    - wait until activity is less, then write altogether so that individual writes don't delay execution of process
    - This also allows the OS to group writes to the same part of disk to optimize performance. (we'll see this later)

Memory-Mapped Files

- **Example: Save/modify data in a running program to a file**
  - Map this file to virtual memory
  - Page fault in pages of file as you need them – like demand paging for code/data but for storing data
  - Can avoid slow system calls to read/write
  - Write out modified file pages in a lazy manner
  - Also, since file pages are cached by kernel, so there's just one copy in RAM of a given file page, no need to create a separate copy of file in user space, as for read/write approach

Memory-Mapped I/O (vs. Files)

- **Similar behavior to memory-mapped files**
- **Recall from previous lecture slides that memory-mapped I/O maps device registers (instead of file pages) to memory locations**
  - reads/writes from/to these memory addresses are easy and are automatically caught by the MMU (just as for mem-mapped files), causing the data to be automatically sent from/to the I/O devices
  - e.g. writing to a display's memory-mapped frame buffer, or reading from a memory-mapped serial port
  - Devices use an API to a virtual file

Department of Computer Science
UNIVERSITY OF COLORADO **BOULDER**

Design and Analysis of Operating Systems CSCI 3753

Dr. David Knox
University of
Colorado Boulder