CSE 120
Principles of Operating Systems

Spring 2023

Lecture 16: File Caching and Reliability

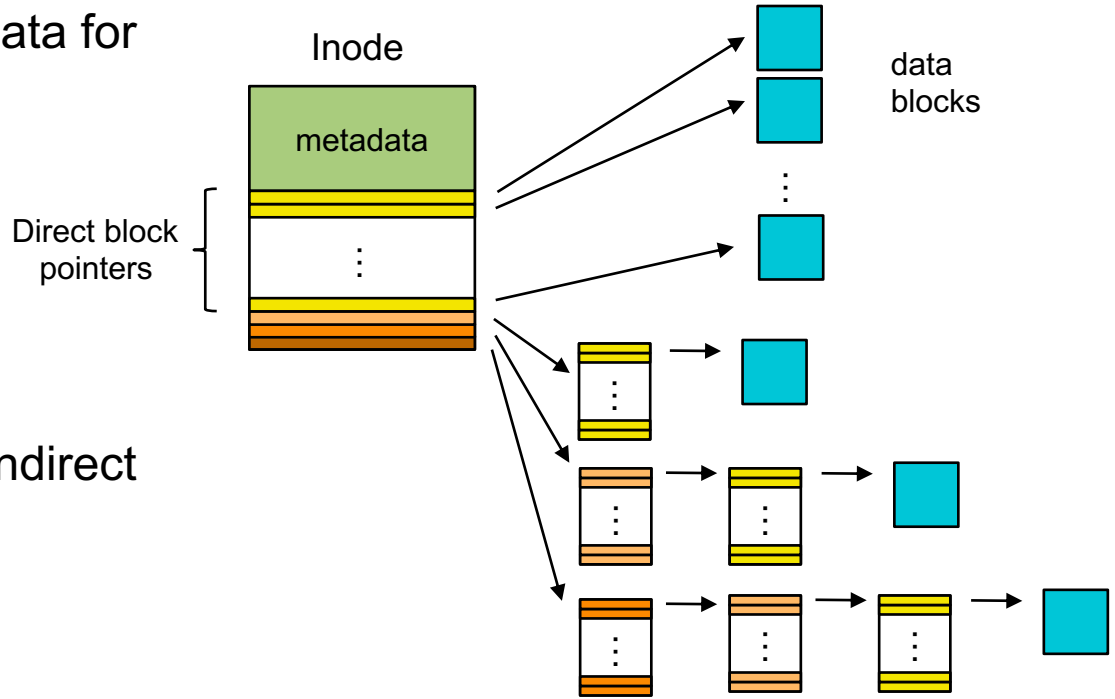
Amy Ousterhout

Administrivia

- Homework #4
  - Due Tuesday June 6th
- Project 3
  - Deadline to change teams: today
  - Due June 10th – there will be no extensions
  - Get started early!

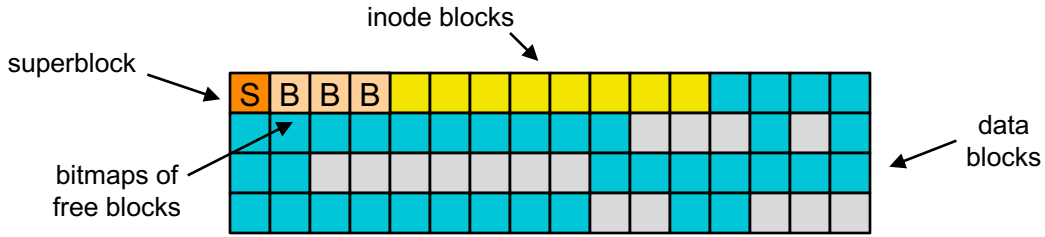
Inodes and Indirect Blocks

- An inode stores all the metadata for a file:
  - Size
  - Owner
  - Protection bits
  - Link count
  - etc.
- Inodes also store direct and indirect pointers to data blocks
- Inodes are small
  - 256 bytes each



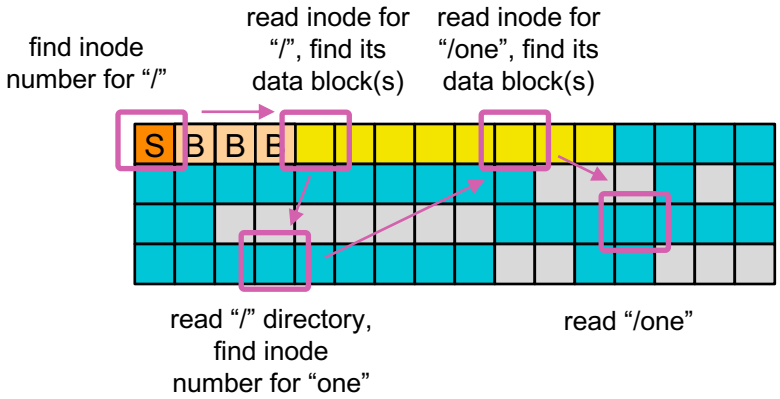
File System Layout

- A superblock defines a file system:
  - Type, number of blocks, number of inodes, etc.
  - Location of inode for the root directory
- Bitmaps indicate whether blocks are free or allocated
  - One for inodes, one for data blocks
- Inodes store metadata about files and the locations of their blocks
- Remaining disk blocks used to store files and directories



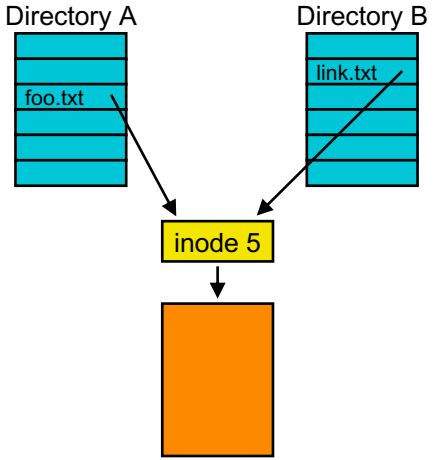
Path Name Translation

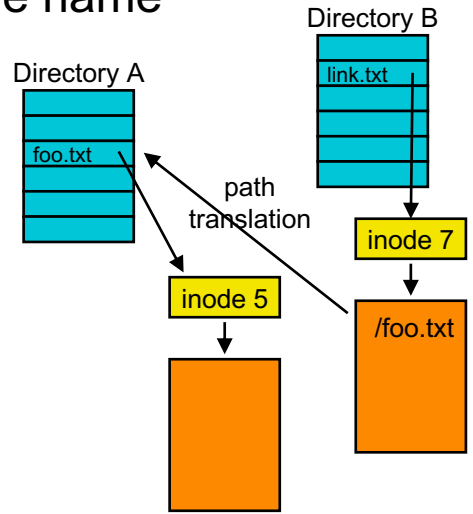
- Suppose you want to open “/one”
- What does the file system do?



Hard Links vs. Soft (Symbolic) Links

- Links enable aliasing, or multiple ways to refer to the same thing
- Hard link: a directory entry that associates a name with a file
- Soft link: a file whose contents are another file name



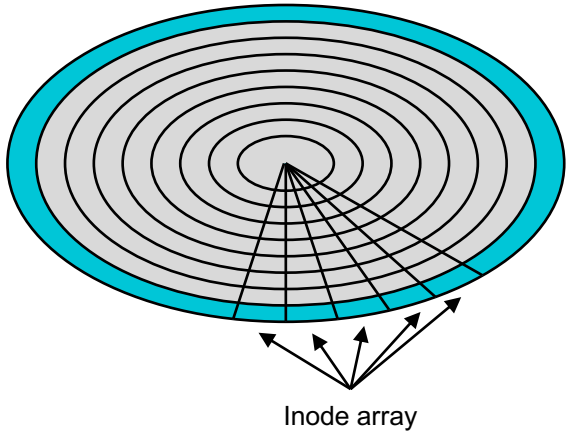


Today's Outline

- Optimized disk layout
  - Fast File System (FFS)
- File buffer cache
- File system reliability
  - `fsck`
  - Ordered writes
  - Journaling

Original Unix File System Disk Layout

- Simple disk layout:
  - Block size is sector size (512 bytes)
  - Inodes are in an array in outermost cylinders
  - Data blocks are on inner cylinders
  - Use linked list for free blocks
- Cons:
  - Inodes must point to many small blocks
  - Linked list of free blocks becomes unordered
  - Fixed max number of files
  - Inodes are far away from data blocks

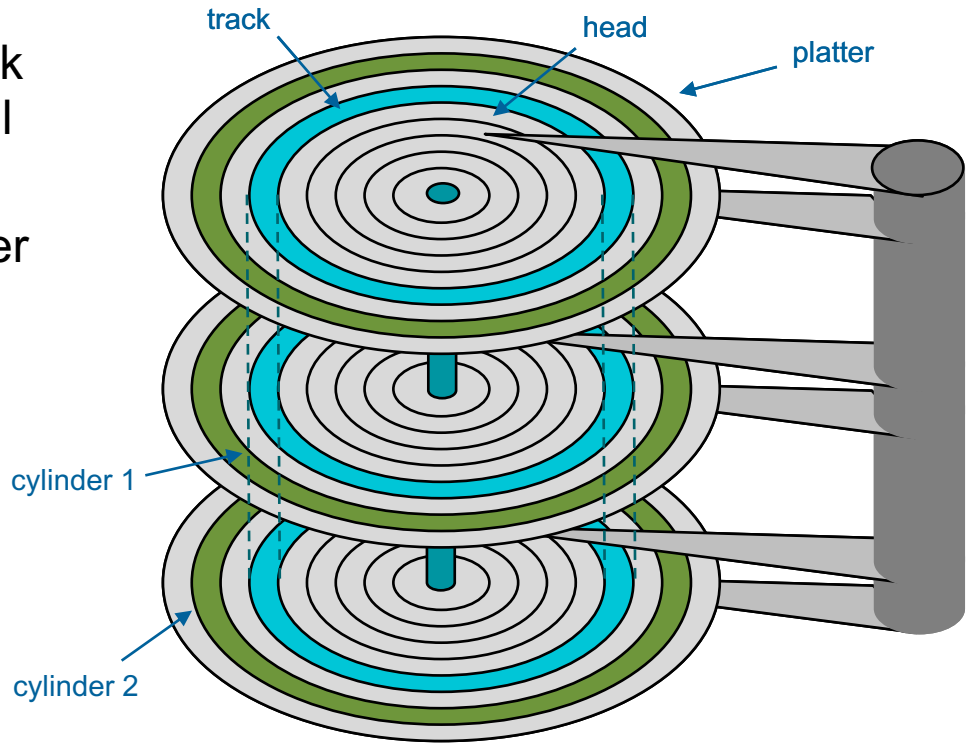


BSD Fast File System (FFS)

- Use a larger block size – 4 KB or 8 KB
- Use a bitmap instead of a free list
- How can we take advantage of spatial locality?

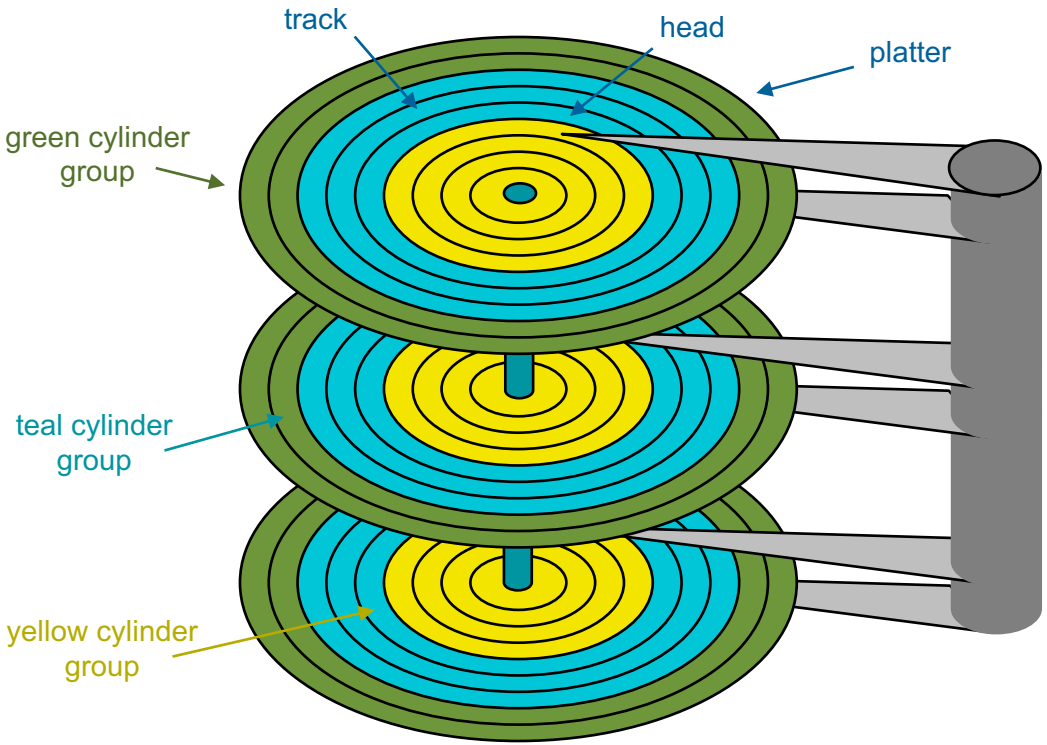
Disk Cylinders

- Cylinder: the data in a given track number across all surfaces on all platters
- Can access any data in a cylinder without performing a seek



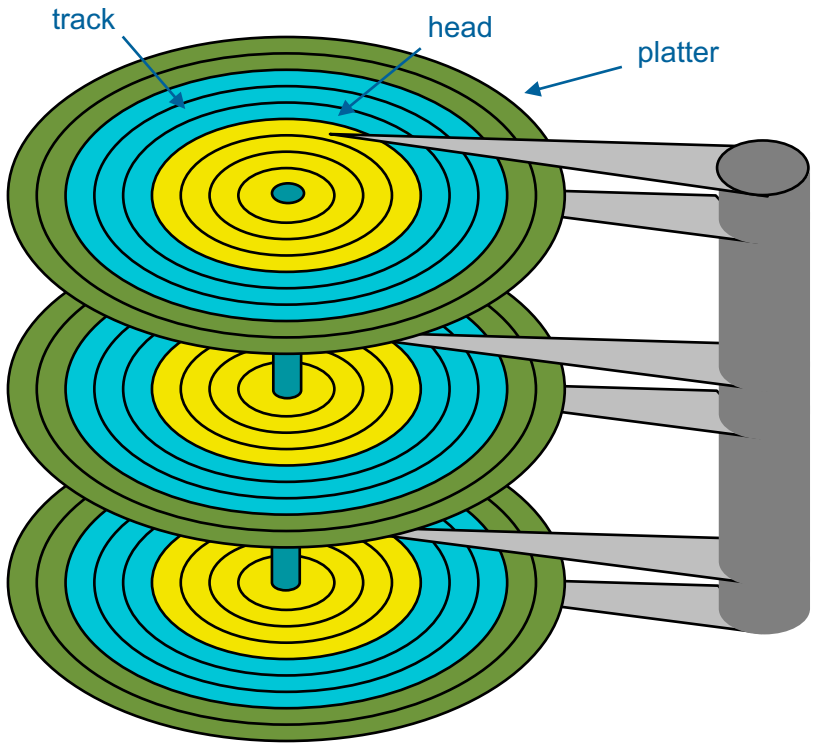
Cylinder Groups

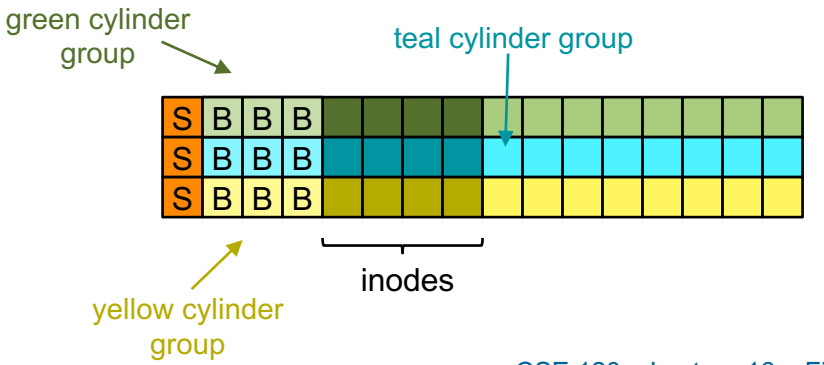
- Cylinder group : a group of consecutive cylinders
- Data in the same cylinder group can be accessed quickly



FFS Disk Layout

- Divide the disk into cylinder groups
- Include all file system data structures within each group
  - Replicate the superblock
  - Include bitmaps, inodes, etc.



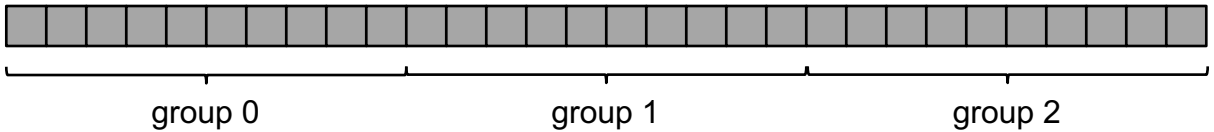


FFS Disk Layout Policy

- Balance directories across cylinder groups
  - Try to find a group with few directories and many free inodes
- Group file data together with its directory
  - Place file inodes in the same group as their directory
  - Place file data blocks in the same group as their inode

Cylinder Groups vs. Block Groups

- Modern disk drives do not export information about cylinder groups
  - Block interface – a logical array of blocks
- Modern file systems use block groups instead of cylinder groups
  - Block group – a consecutive portion of the disk's address space

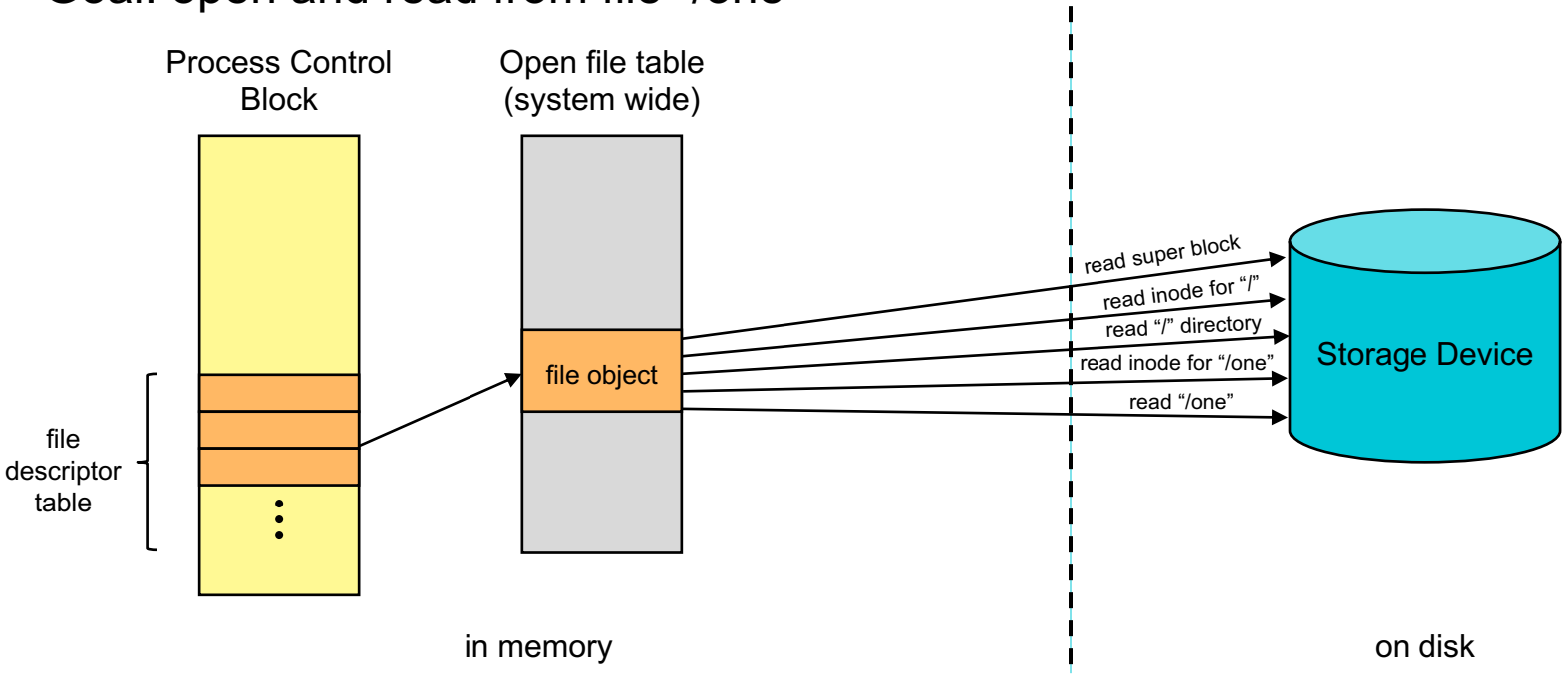


Today's Outline

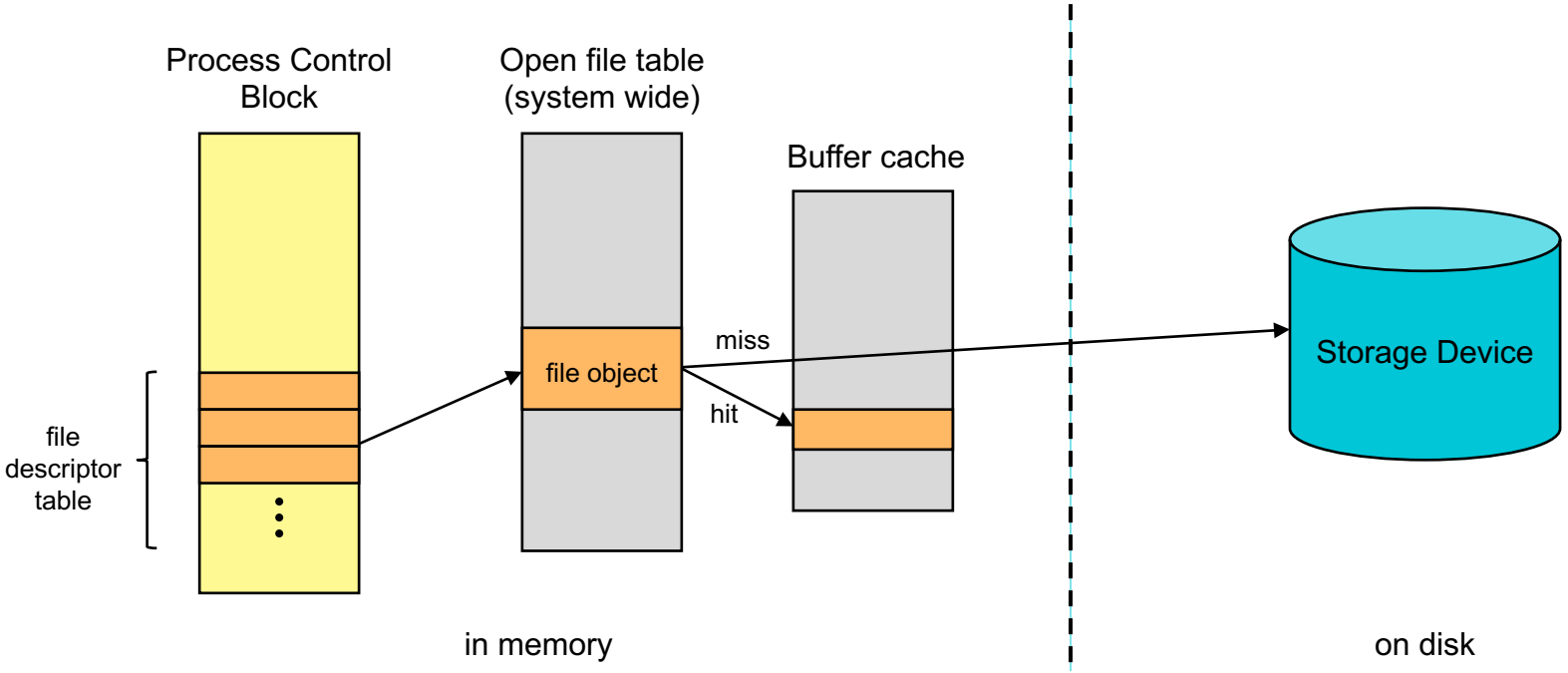
- Optimized disk layout
  - Fast File System (FFS)
- File buffer cache
- File system reliability
  - `fsck`
  - Ordered writes
  - Journaling

Path Name Translation

- Goal: open and read from file “/one”



File Buffer Cache

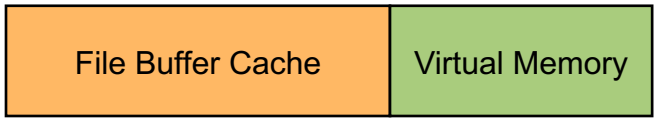


File Buffer Cache

- Applications exhibit significant locality for reading and writing files
- The file buffer cache caches file blocks in memory to capture locality
  - Caches all kinds of file blocks (super blocks, inode blocks, data blocks, etc.)
  - Reading from cache makes the disk behave like memory
  - System-wide, used and shared by all processes
- Challenges:
  - The file buffer cache competes with virtual memory
  - Need a replacement algorithm
  - Handling writes

Physical Memory Split

- Virtual memory and buffer cache dynamically compete for physical memory
- Originally: fixed size buffer cache
- Now: unified page cache for file blocks and virtual memory pages
  - Any page can be used for either
  - Shared replacement policy



File Buffer Cache - Reads

- Read: check if the block is in the buffer cache
  - Yes:
    - » Copy from the buffer cache to the user buffer
  - No:
    - » Replacement if necessary
    - » Read the file block into the buffer cache
    - » Copy from the buffer cache to the user buffer

File Buffer Cache - Writes

- Write: check if the block is in the buffer cache
  - Yes:
    - » Write data from the user buffer to the buffer cache
  - No:
    - » Replacement if necessary
    - » Read the file block into the buffer cache
    - » Write data from the user buffer to the buffer cache

Write Policies

- On a write, applications may assume that data has been written to the disk
  - Not necessarily true with caching!
- Write-through caching
  - Write to storage immediately
  - Pros: simple, cache is consistent
  - Con: more writes
- Write-back caching
  - Gather (buffer) writes in memory and then write all buffered data back to the storage device (e.g., every 30 seconds in Unix)
  - Pros: fast writes, batching
  - Con: may lose data

Today's Outline

- Optimized disk layout
  - Fast File System (FFS)
- File buffer cache
- File system reliability
  - `fsck`
  - Ordered writes
  - Journaling

File System Reliability

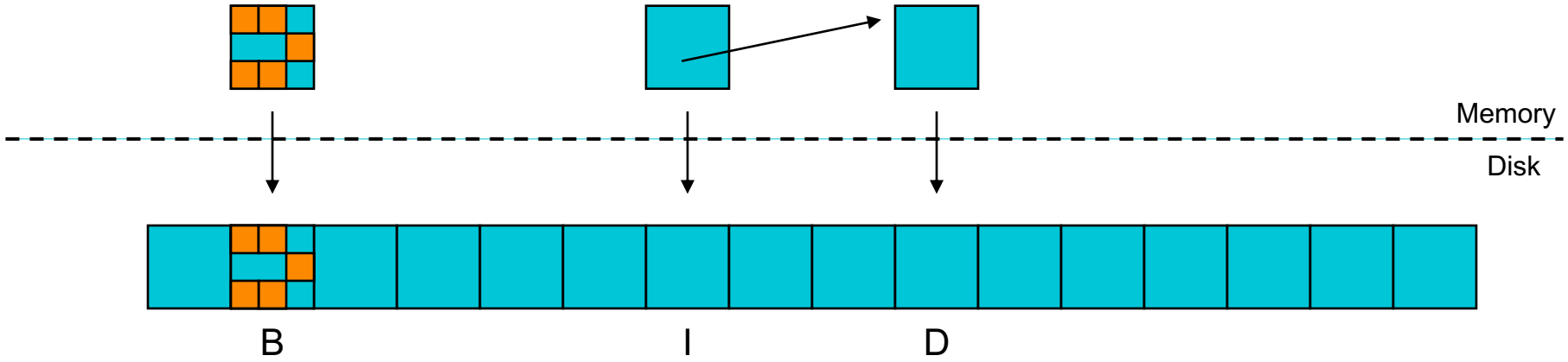
- Loss of data in a file system can have catastrophic effects
- Three threats:
  - Accidental or malicious deletion of data
    - » Back up entire file system periodically
  - Disk failure
    - » Replicate data (e.g., RAID)
  - System crash or loss of power
    - » Consistency

Crash Consistency Problem

- Crashes can happen at any time, even in the middle of critical sections
- Some file system operations involve writing multiple blocks:
  - Moving a file between directories
    - » Delete file from old directory
    - » Add file to new directory
  - Create a new file
    - » Allocate space on disk for inode
    - » Write new inode to disk
    - » Add the new file to the directory
- If a crash occurs in the middle of a file system operation, the filesystem may be left in an inconsistent state

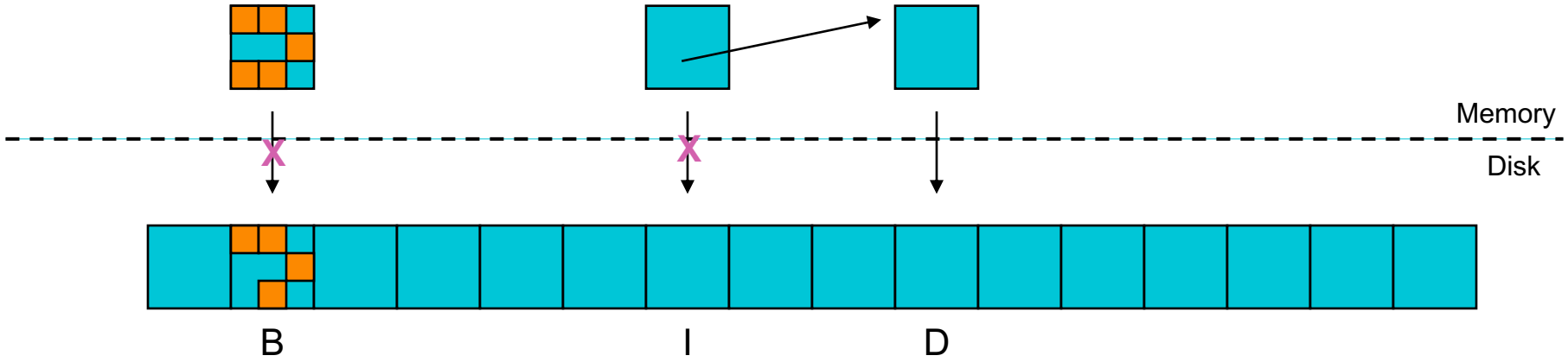
Crash Example

- Goal: append a new block to an existing file
  - Write data bitmap B
  - Write new data block D
  - Write inode I of file



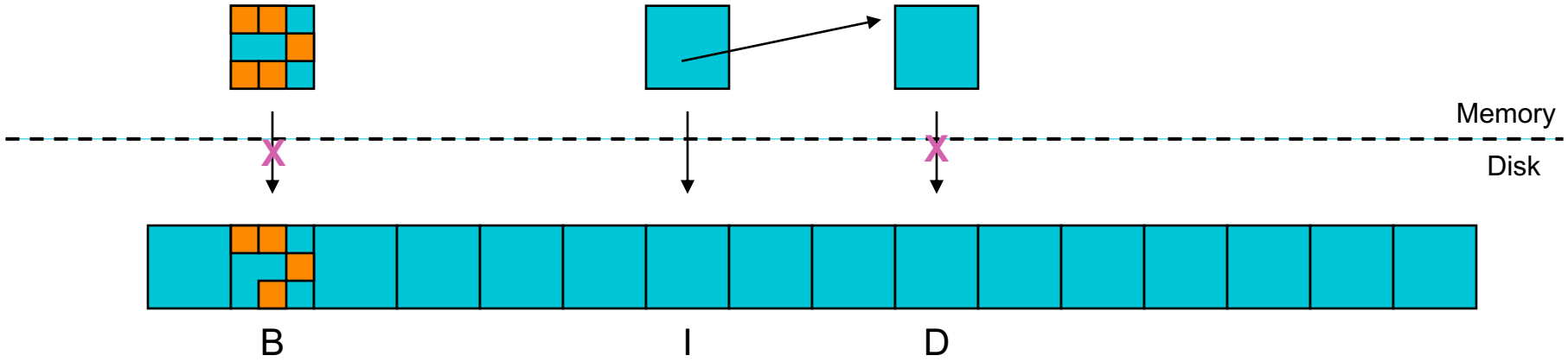
Crash Example: Only Write the Data

- Goal: append a new block to an existing file
  - ~~Write data bitmap B~~
  - Write new data block D
  - ~~Write inode I of file~~
- Outcome:
  - It is as if we never tried to append the block



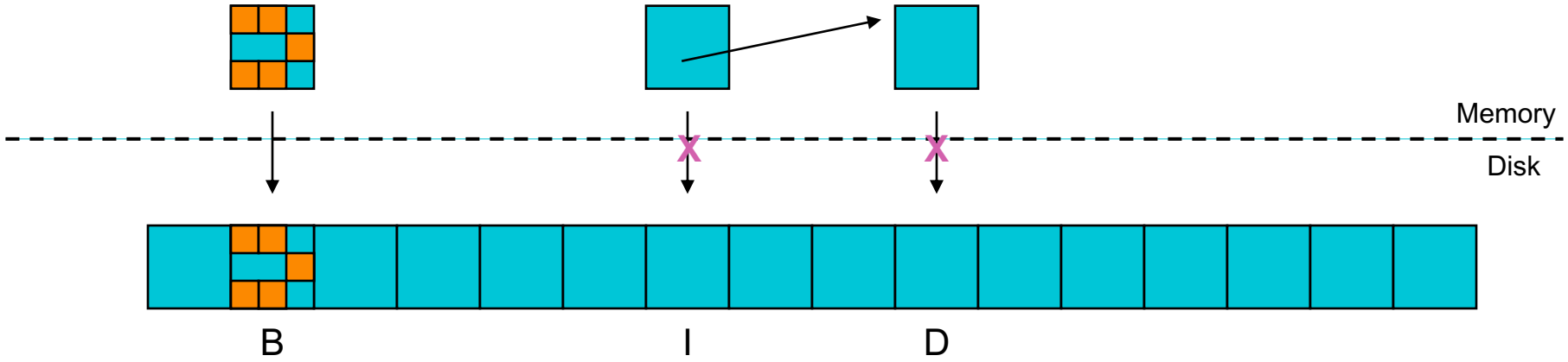
Crash Example: Only Write the Inode

- Goal: append a new block to an existing file
  - ~~Write data bitmap B~~
  - ~~Write new data block D~~
  - Write inode I of file
- Outcome:
  - Could read garbage data
  - File system is inconsistent (bitmap and inode disagree)



Crash Example: Only Write the Bitmap

- Goal: append a new block to an existing file
  - Write data bitmap B
  - ~~Write new data block D~~
  - ~~Write inode I of file~~
- Outcome:
  - Space leak



Crash Consistency Problem

- File system operations may involve writing multiple blocks
  - Inode block, data block, bitmap, indirect block, etc.
- Crashes can happen at any time, even in the middle of critical sections
- If a crash occurs in the middle of a file system operation, the file system may be left in an inconsistent state
- Goal: ensure that updates to the file system occur atomically
- Assume that writing to a single disk block is atomic

Check and Repair with `fsck`

- `fsck`: the file system checker in Unix
- When system boots:
  - Scan the file system, find inconsistencies, and repair them
  - Example inconsistencies:
    - » An inode points to a file that is marked free in the bitmap
    - » Reference count for an inode doesn't match the number of links to it from directories
    - » One block pointed to by multiple inodes
  - Repair inconsistencies or notify an admin
- Cons:
  - Very slow! Can take hours to run on large disk volumes
  - May pose security issues

To be continued next lecture…

For next class…

- Read Appendix B