COS 318: Operating Systems

File Caching and Reliability

Jaswinder Pal Singh
Computer Science Department
Princeton University

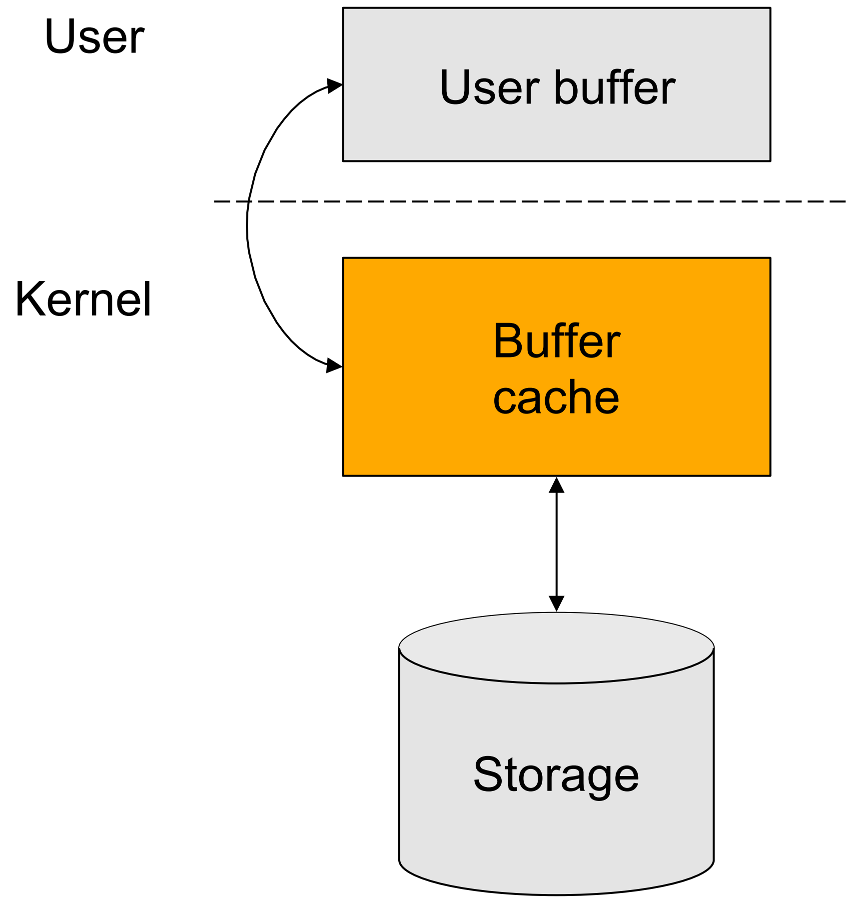
(http://www.cs.princeton.edu/courses/cos318/)

Topics

- File buffer cache
- File system recovery
- Consistent updates
- Transactions

File Buffer Cache

- A large cache in kernel
- Read: check if the block is in
  - Yes: Copy block to user buffer
  - No: Read from storage to buffer cache and copy to user buffer
- Write: check if the block is in
  - Yes: Update it with user buffer
  - No: Copy to buffer cache (may replace a block)
  - Write the block to storage
- Usual questions
  - What to cache?
  - How to size the cache?
  - What to prefetch?
  - How and what to replace?
  - Which write policies?

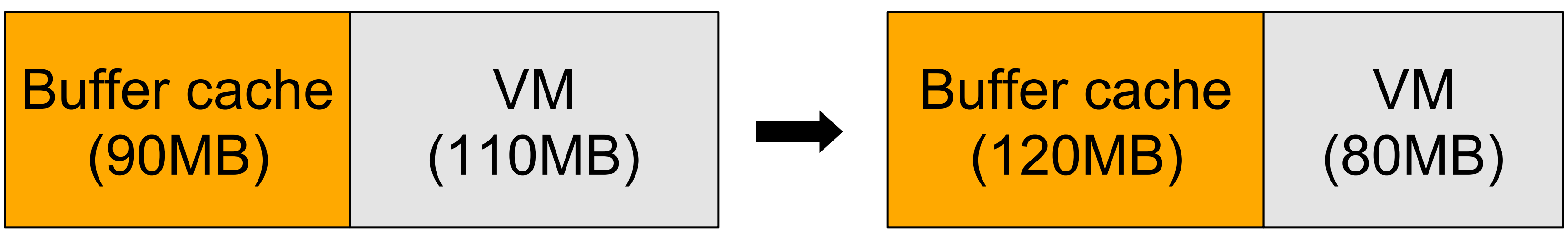


What to Cache?

- For different kinds of blocks
  - i-nodes
  - Indirect blocks
  - Directories
  - Data blocks
- Issues
  - Are all blocks equal?

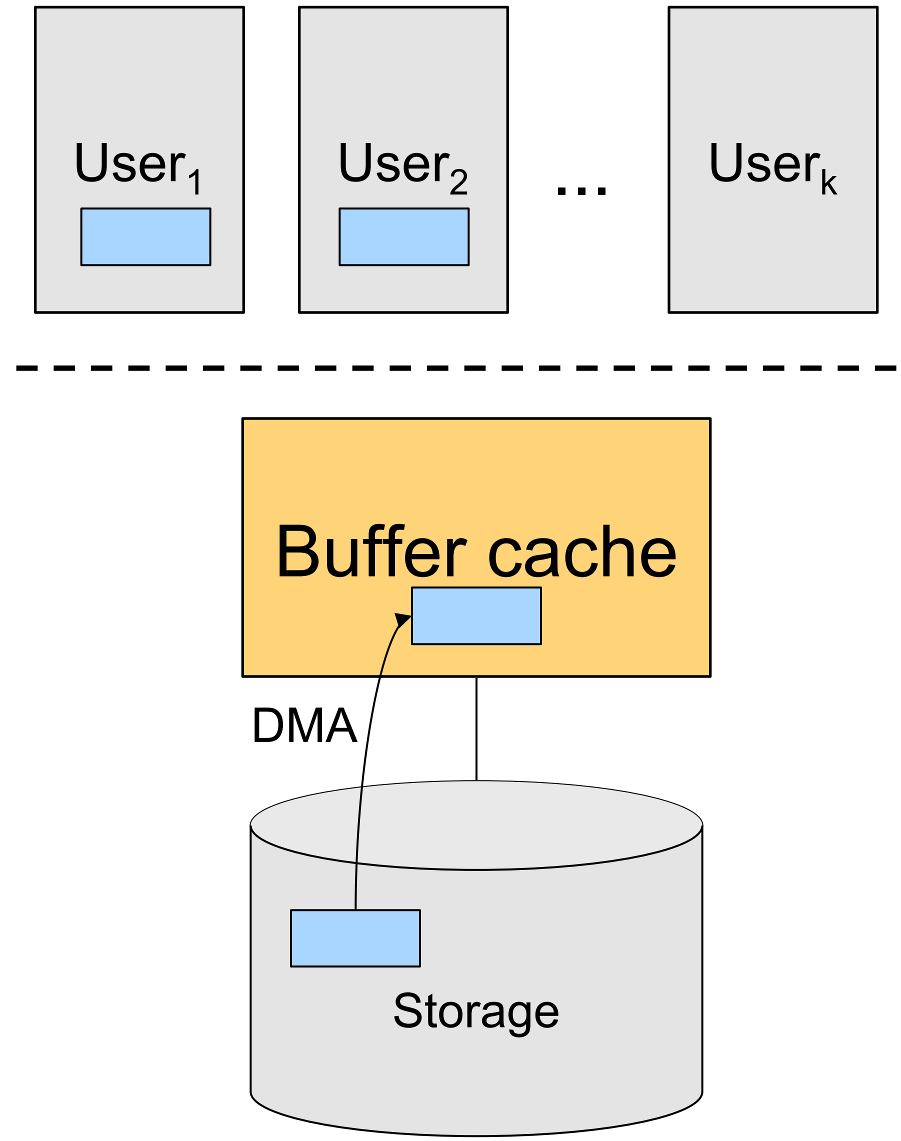
Buffer Cache Size

- Competition
  - Completes with VM and the rest of the system for memory
- Two approaches
  - Fixed size
  - Variable size
- How to adjust buffer cache size?
  - Users make decisions
  - Working set idea with dynamic adjustments within thresholds



Why in the Kernel?

- DMA
  - DMA works with “pinned” physical memory
- Multiple user processes
  - Share the buffer cache
- Typical replacement strategy
  - Global LRU
  - Working set for each process
- Questions
  - Move buffer cache to the user level?

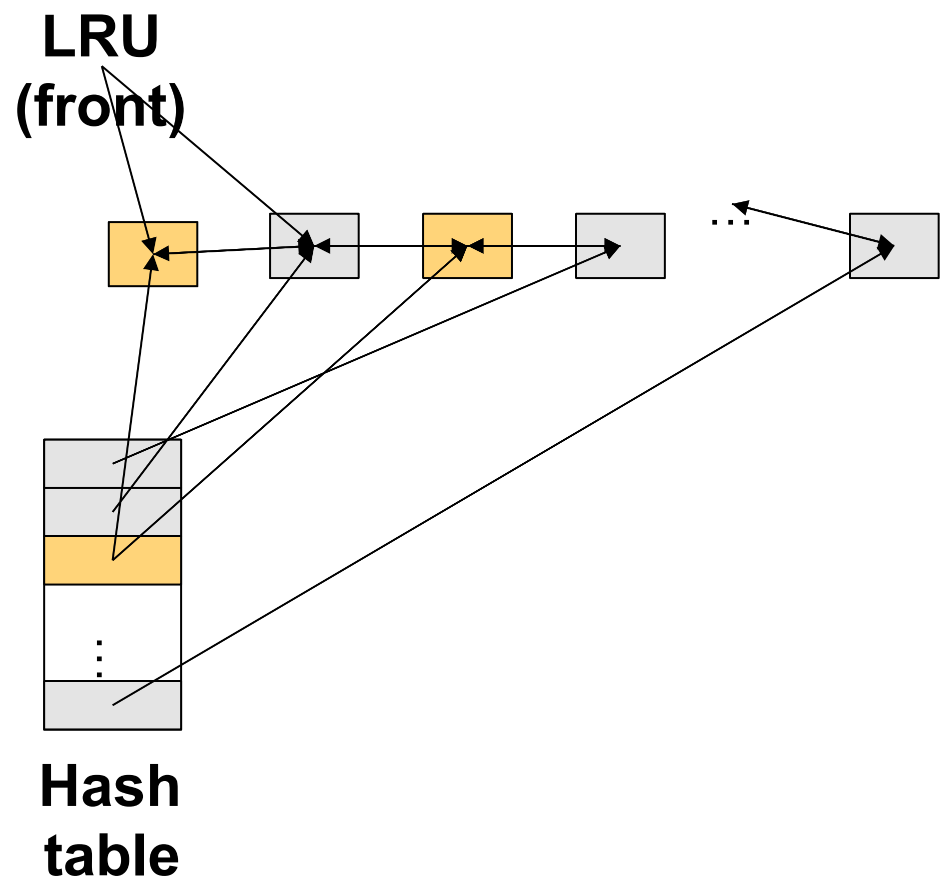


What to Prefetch?

- Optimal
  - Prefetch in just enough time to use them
- Good news: file accesses have locality
  - Temporal locality
  - Spatial locality
- Common strategies
  - Prefetch next k blocks together
  - Discard unreferenced blocks
  - Layout consecutive blocks to the same cylinder group
  - Fetch directory and i-nodes together
- Advanced strategy
  - Prefetch all small files of a directory

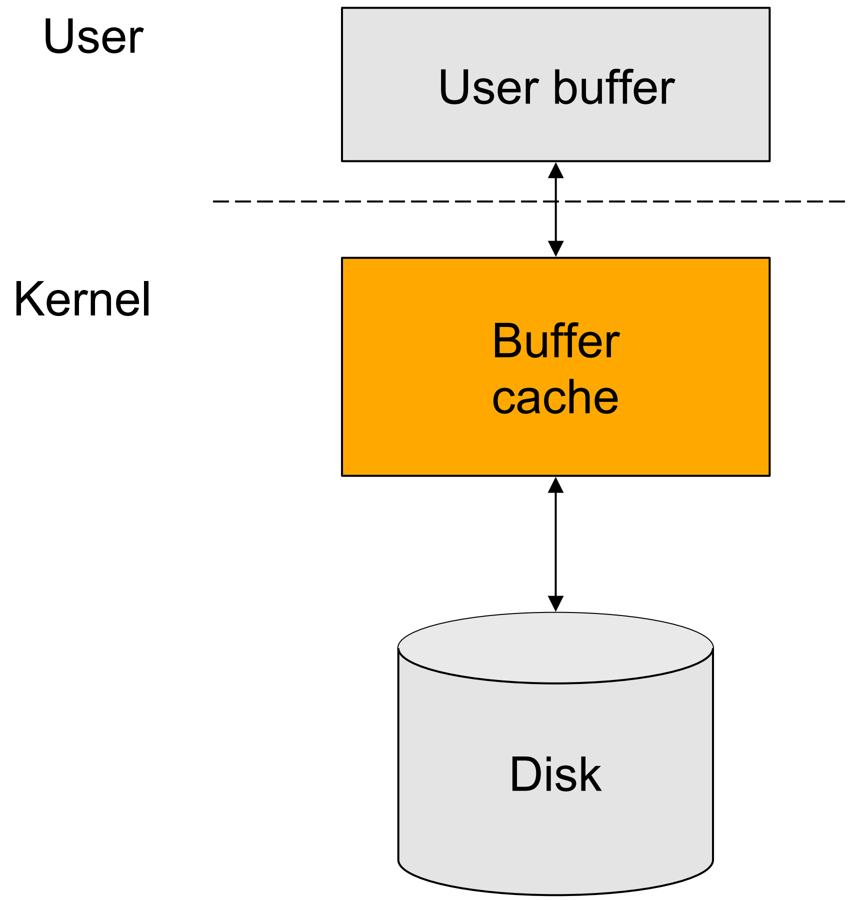
How and What to Replace?

- Theory
  - Use past to predict future
  - LRU is good
- LRU replacement
  - double linked list with a hash table
  - If b is in buffer cache, move it to front and return b
  - Otherwise, replace the tail block, get b from storage, insert b to the front
- Questions
  - Why a hash table?



Write Policies

- Write through
  - Write to storage immediately
  - Cache is consistent
  - Simple, but cause more I/Os
- Write back
  - Update a block in buffer cache and mark it as dirty write to storage later
  - Fast writes, absorbs writes, and enables batching
  - So, what's the problem?

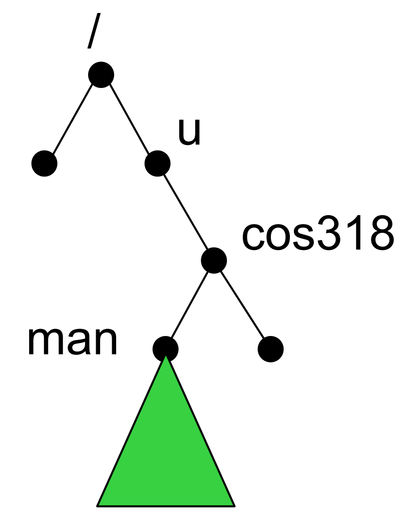


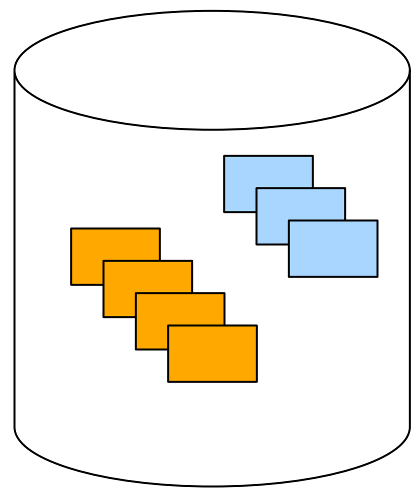
Write Back Complications

- Tension
  - On crash, all modified data in cache is lost.
  - Postpone writes ⇒ better performance but more damage
- When to write back
  - When a block is evicted
  - When a file is closed
  - On an explicit flush
  - When a time interval elapses (30 seconds in Unix)
- Issues
  - These options have no guarantees

File Recovery Tools

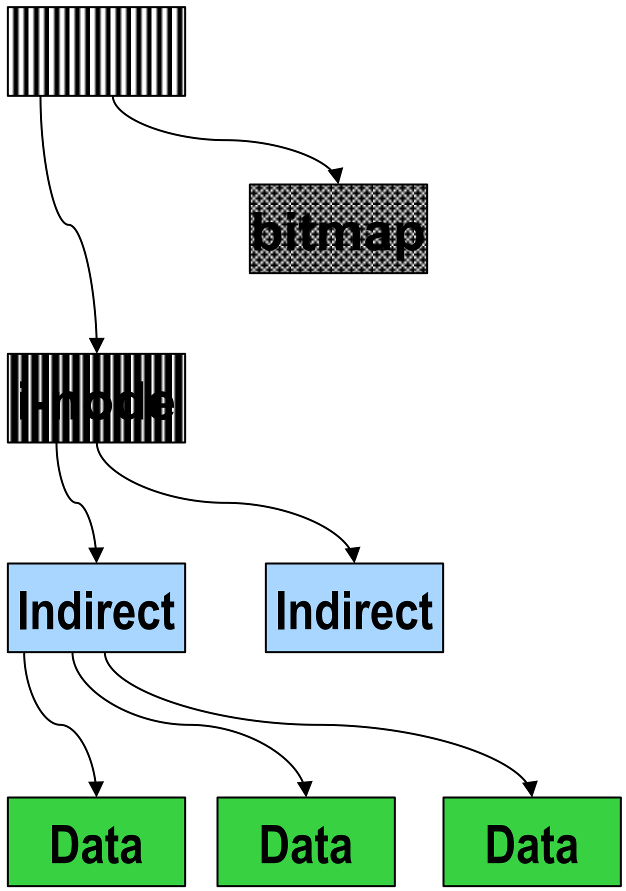
- Physical backup (dump) and recovery
  - Dump disk blocks by blocks to a backup system
  - Backup only changed blocks since the last backup as an incremental
  - Recovery tool is made accordingly
- Logical backup (dump) and recovery
  - Traverse the logical structure from the root
  - Selectively dump what you want to backup
  - Verify logical structures as you backup
  - Recovery tool selectively move files back
- Consistency check (e.g. fsck)
  - Start from the root i-node
  - Traverse the whole tree and mark reachable files
  - Verify the logical structure
  - Unreachable blocks are free





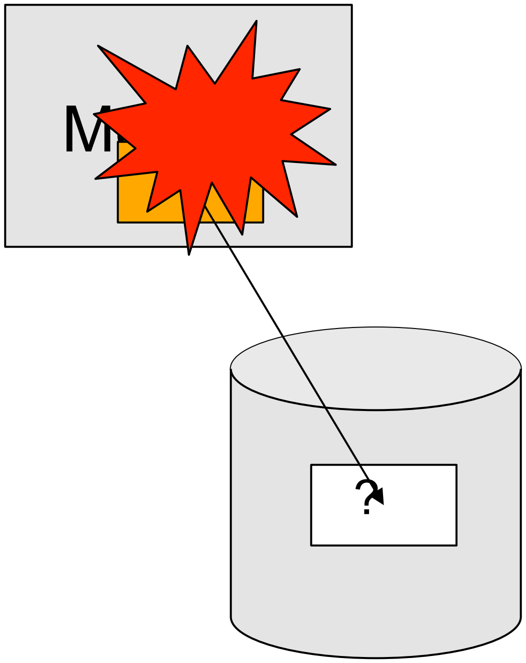
Recovery from Disk Block Failures

- Boot block
  - Create a utility to replace the boot block
  - Use a flash memory to duplicate the boot block and kernel
- Super block
  - If there is a duplicate, remake the file system
  - Otherwise, what would you do?
- Free block data structure
  - Search all reachable files from the root
  - Unreachable blocks are free
- i-node blocks
  - How to recover?
- Indirect or data blocks
  - How to recover?



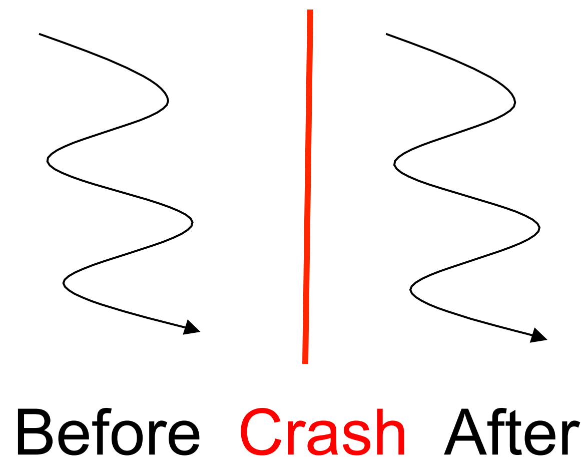
Persistency and Crashes

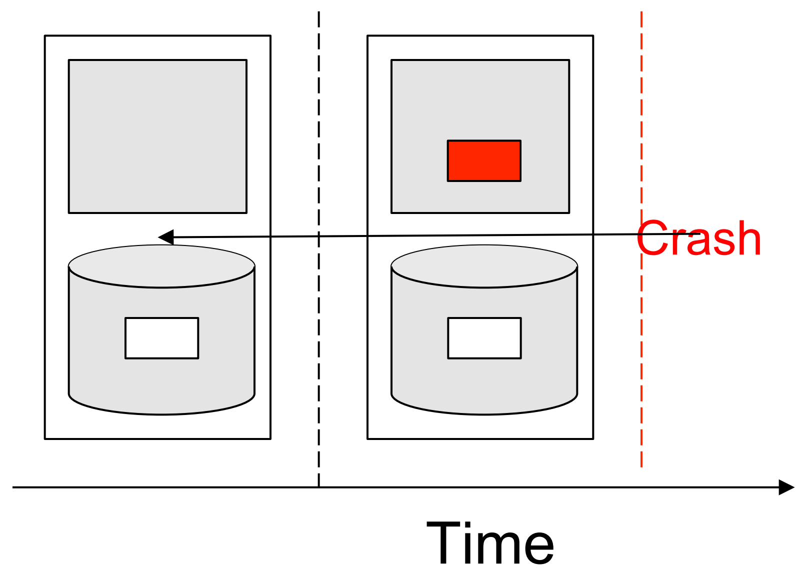
- File system promise: Persistency
  - Store files until explicitly deleted
  - Backups can recover your file beyond the deletion point
- Why is this hard?
  - Systems can crash anytime
  - A crash will destroy memory content
  - Cache more ⇒ better performance
  - Cache more ⇒ lose more on a crash
  - A write may modify multiple blocks but the system can only atomically modify one at a time



What Is A Crash?

- Crash is like a context switch
  - Think about a file system as a thread before the context switch and another after the context switch
  - Two threads read or write same shared state?
- Crash is like time travel
  - Current volatile state lost; suddenly go back to old state
  - Example: move a file
    - Place it in a directory
    - Delete it from old
    - Crash happens and both directories have problems





Approaches

- Throw everything away and start over
  - Done for most things (e.g., make again)
- Reconstruction
  - Try to fix things after a crash (“fsck”)

- **Make consistent updates**
  - Either new data or old data, but not garbage data
- **Make multiple updates appear atomic**
  - Build large atomic units from smaller atomic ones

Write Metadata First

- Modify /u/cos318/foo
  - Traverse to /u/cos318/

Crash — **Consistent**

  - Allocate data block

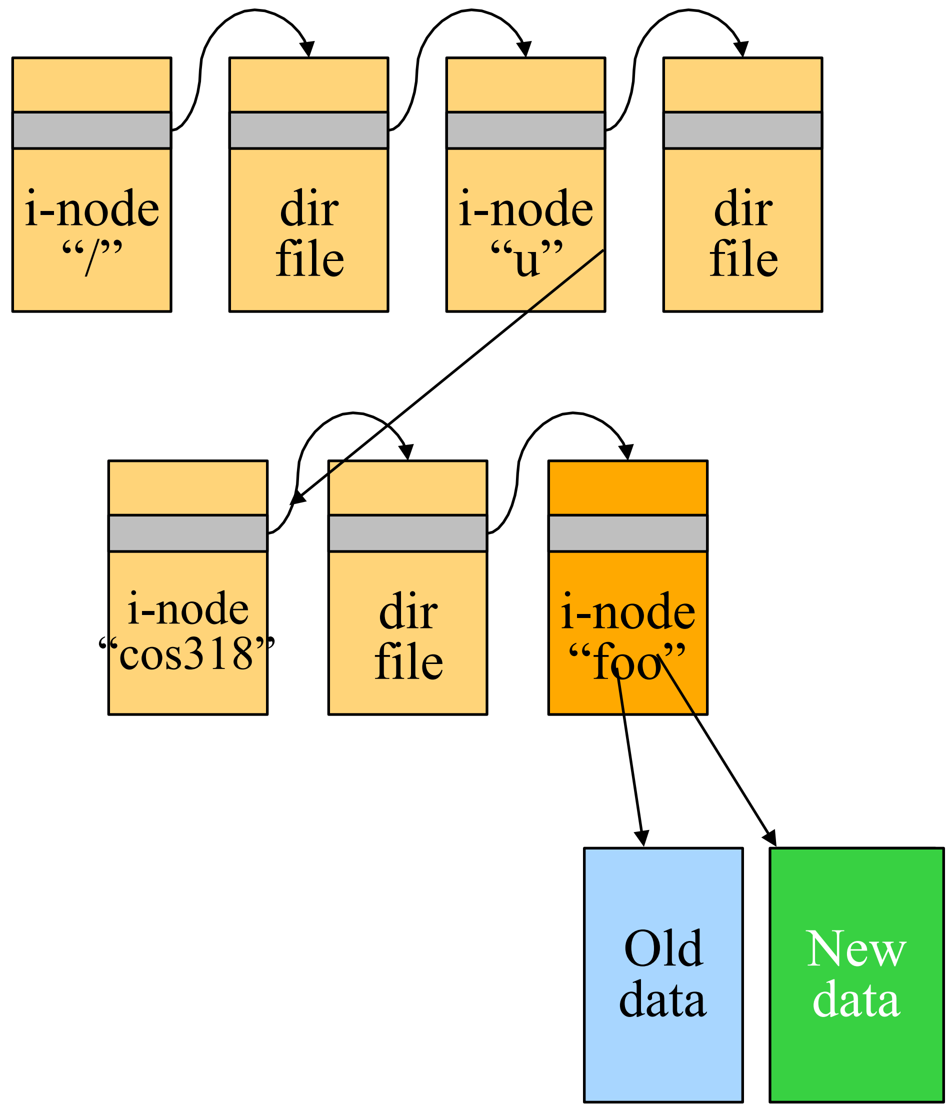
Crash — **Consistent**

  - Write pointer into i-node

Crash — **Inconsistent**

  - Write new data to foo

Crash — **Consistent**



Writing metadata first can cause inconsistency

Write Data First

- Modify /u/cos318/foo
  - Traverse to /u/cos318/

Crash → Consistent

  - Allocate data block

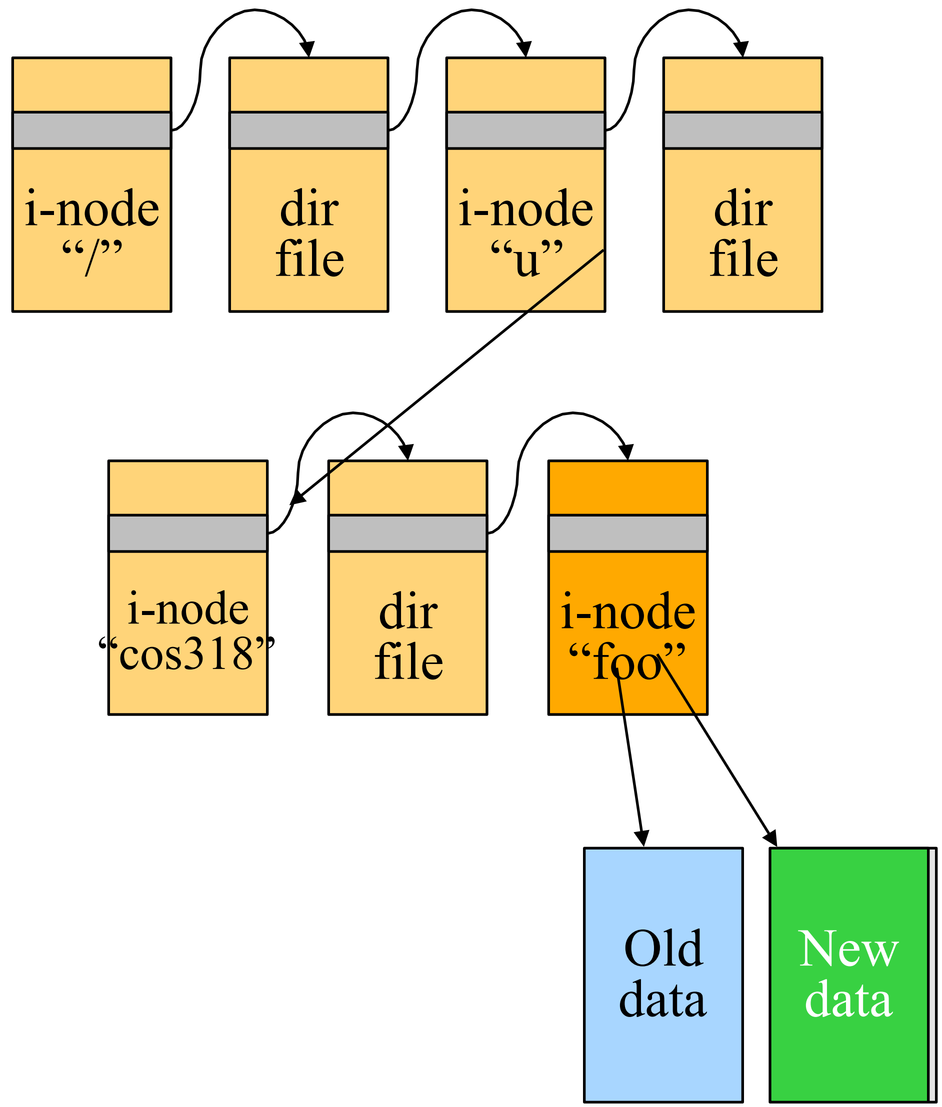
Crash → Consistent

  - Write new data to foo

Crash → Consistent

  - Write pointer into i-node

Crash → Consistent



Consistent Updates: Bottom-Up Order

- The general approach is to use a “bottom up” order
  - File data blocks, file i-node, directory file, directory i-node, …
- What about file buffer cache
  - Write back all data blocks
  - Update file i-node and write it to disk
  - Update directory file and write it to disk
  - Update directory i-node and write it to disk (if necessary)
  - Continue until no directory update exists
- Solve the write back problem?
  - Updates are consistent but leave garbage blocks around
  - May need to run fsck to clean up once a while
- Ideal approach: consistent update without leaving garbage

Transaction Properties

- Group multiple operations to have “ACID” property
  - Atomicity
    - It either happens or doesn’t (no partial operations)
  - Consistency
    - A transaction is a correct transformation of the state
  - Isolation (serializability)
    - Transactions appear to happen one after the other
  - Durability (persistency)
    - Once it happens, stays happened
- Question
  - Do critical sections have ACID property?

Transactions

- Bundle many operations into a transaction
- Primitives
  - BeginTransaction
    - Mark the beginning of the transaction
  - Commit (End transaction)
    - When transaction is done
  - Rollback (Abort transaction)
    - Undo all the actions since “Begin transaction.”
- Rules
  - Transactions can run concurrently
  - Rollback can execute anytime
  - Sophisticated transaction systems allow nested transactions

Implementation

- BeginTransaction
  - Start using a “write-ahead” log on disk
  - Log all updates
- Commit
  - Write “commit” at the end of the log
  - Then “write-behind” to disk by writing updates to disk
  - Clear the log
- Rollback
  - Clear the log
- Crash recovery
  - If there is no “commit” in the log, do nothing
  - If there is “commit,” replay the log and clear the log

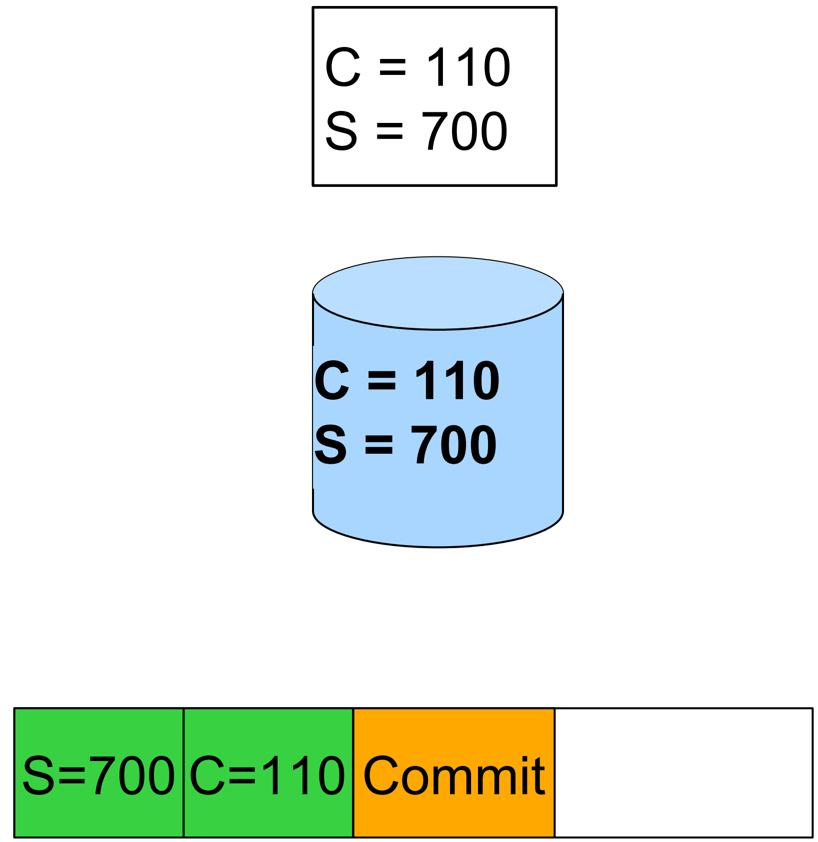
- Assumptions
  - Writing to disk is correct (recall the error detection and correction)
  - Disk is in a good state before we start

An Example: Atomic Money Transfer

- Move $100 from account S to C (1 thread):

```
BeginTransaction
    S = S - $100;
    C = C + $100;
Commit
```

- Steps:
  - 1: Write new value of S to log
  - 2: Write new value of C to log
  - 3: Write commit
  - 4: Write S to disk
  - 5: Write C to disk
  - 6: Clear the log
- Possible crashes
  - After 1
  - After 2
  - After 3 before 4 and 5
- Questions
  - Can we swap 3 with 4?
  - Can we swap 4 and 5?



Revisit The Implementation

- BeginTransaction
  - Start using a “write-ahead” log on disk
  - Log all updates
- Commit
  - Write “commit” at the end of the log
  - Then “write-behind” to disk by writing updates to disk
  - Clear the log
- Rollback
  - Clear the log
- Crash recovery
  - If there is no “commit” in the log, do nothing
  - If there is “commit,” replay the log and clear the log

- Questions
  - What if there is a crash during the recovery?

Two-Phase Locking for Transactions

- First phase
  - Acquire all locks
- Second phase
  - Commit operation release all locks (no individual release operations)
  - Rollback operation always undo the changes first and then release all locks

Use Transactions in File Systems

- Make a file operation a transaction
  - Create a file
  - Move a file
  - Write a chunk of data
  - …
  - Would this eliminate any need to run fsck after a crash?
- Make arbitrary number of file operations a transaction
  - Make sure logging are idempotent
  - Recovery by replaying the log
  - Called "logging file system" or "journaling file system"

Performance Issue with Logging

- For every disk write, we now have two disk writes
  - They are on different parts of the disk!
- Performance tricks
  - Changes made in memory and then logged to disk
  - Log writes are sequential
  - Merge multiple writes to the log with one write
  - Use NVRAM (Non-Volatile RAM) to keep the log

Log Management

- How big is the log?
- Observation
  - Log what's needed for crash recovery
- Method
  - Checkpoint operation: flush the buffer cache to disk
  - After a checkpoint, we can truncate log and start again
  - Log needs to be big enough to hold changes in memory
- Question
  - If you only log metadata (file descriptors and directories) and not data blocks, are there any problems?

Summary

- File buffer cache
  - True LRU is possible
  - Simple write back is vulnerable to crashes
- Disk block failures and file system recovery tools
  - Individual recovery tools
  - Top down traversal tools
- Consistent updates
  - Transactions and ACID properties
  - Logging or Journaling file systems