Lec 22

More Transactional Storage

FFS: Create a file

- allocate data block
- write data block
- allocate inode
- write inode block
- update free block bitmap
- update directory name → number
- update modification time

Recovery: scan inode table
- delete any unlinked inodes
- synchronize free block bitmap
- update missing mod. times

} O(n) in size of disk

Application File Save

- typically cannot update in place (why?)
- instead: write new version to temp file
- atomically rename temp file to original name
  - POSIX rename is guaranteed atomic!
- careful use of fsync
  - fsync(tempfile) before rename (why?)
  - fsync(dir) after rename (why?)
- Recovery: look for any unrenamed temp files
  - can append deltas to end of file

Careful Ordering Summary

Pros:

- "simple" — can be understood by app developer
- "efficient" — minimizes additional data structures

Cons:

- complex reasoning required to ensure crash safety
- slow because fsync must be interleaved with updates
- very slow recovery: O(n) in disk size / file size
- difficult to support concurrent operations

Write-ahead Logging

- instead of updating in place, first write intended changes to separate journal/log
  - log is append-only
- Once intended changes are logged on disk then start performing updates to actual data
- once changes are made, garbage collect log

Recovery: scan log to see what changes were intended

Redo Logging

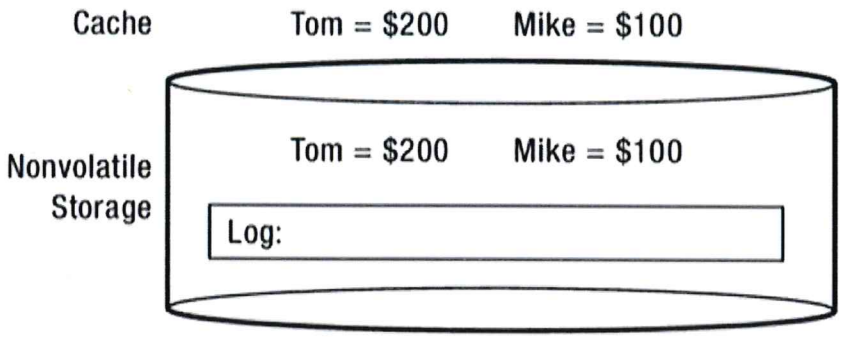
- Prepare: write intended changes to log
  - + wait for these writes to be on disk
- Commit: Write "commit record" to log
  - + wait for this write to be on disk
- Write back: actually do the updates
- once updates are on disk, reclaim that part of log

Recovery: scan log, redoing all updates for committed txns
- ignore uncommitted txns; at end, delete log

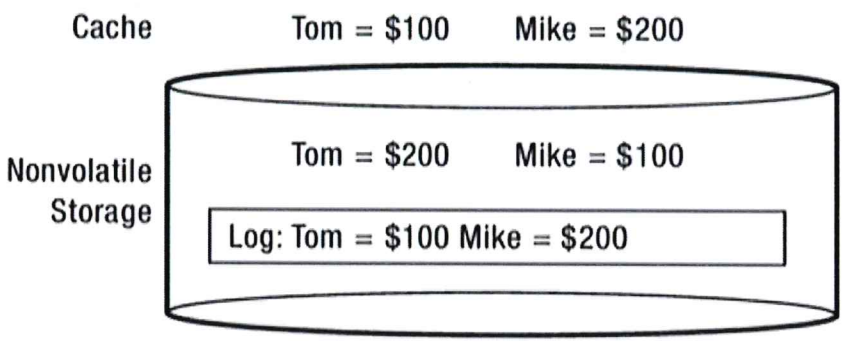
Q1: When is txn committed? Q2: crash during recovery?

Example

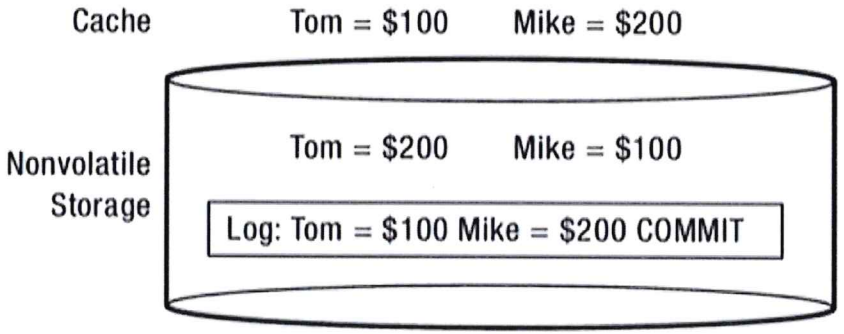
a) Original state



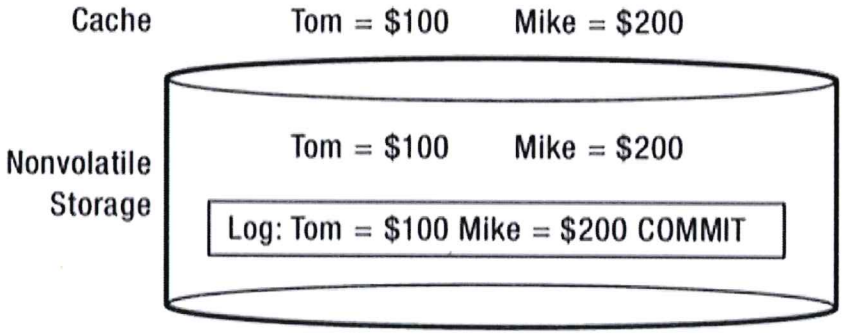
b) Updates appended to log



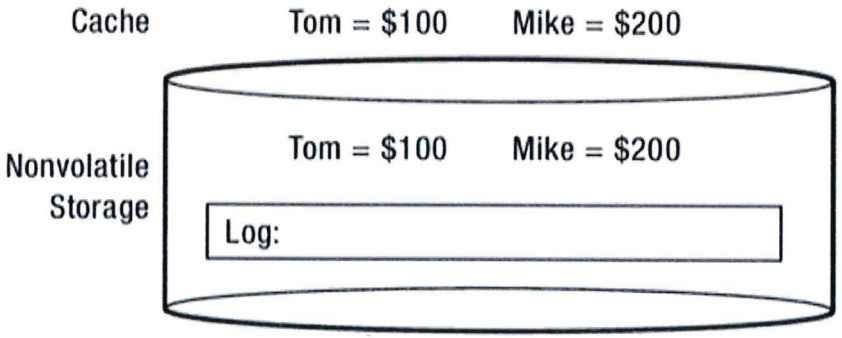
c) Commit appended to log



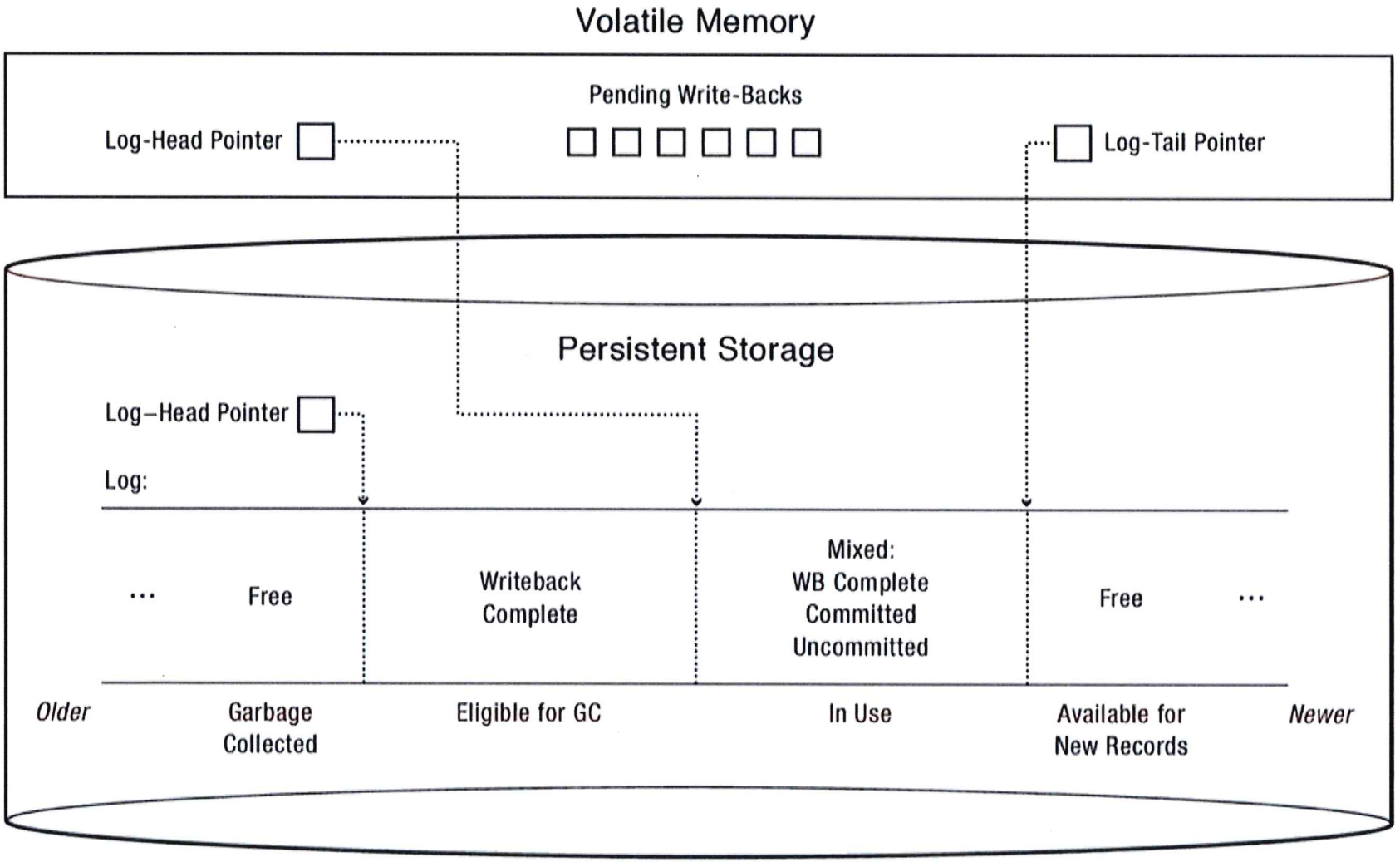
d) Updates applied



e) Garbage collect completed transactions from log



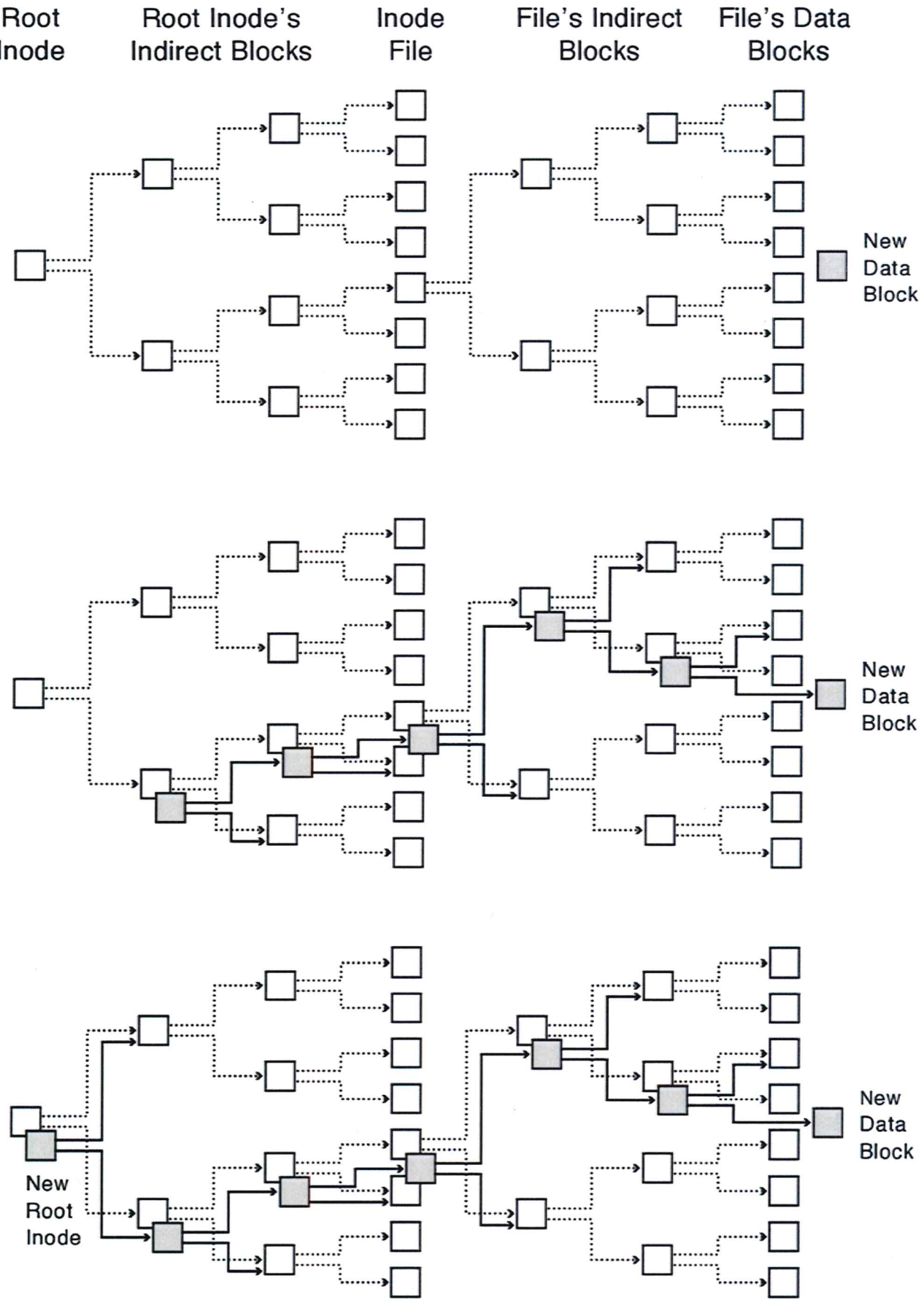
Redo Logging Implementation



Copy-on-write Filesystems

- to update a file, clone the entire filesystem(!)
  - nothing updated in place except superblock
  - made efficient by sharing most of old FS
- Pros: - updates proceed in parallel + can batch commit
  - trivial to support snapshot operation
- widely used: ZFS, XFS, apple FS, btrfs

Copy-on-write write



Log-structured File Systems

- all data is stored in log
  - no write back!
- log is append only → all writes are sequential
- challenge: efficient reads
  - can keep track of which log segments touch which files
  - can do periodic checkpointing
- usually not used in its pure form
  - though possible for flash