Recap

Problem of Crash Consistency

- An access to file system may involve multiple operations
  - e.g., write to a file requires updates to inode, inode bitmap, and data blocks
- Crash (power out) occurring between the operations can cause inconsistency after the system reboots
  - inconsistency scenarios

Solutions

- Checking the entire filesystem: Fsck
- Journaling: Journal the updates before checkpioint (write updates to final locations at storage)

Physical Logging

In **physical logging** the complete contents of the update are first written to a log

Protocol (version 1):

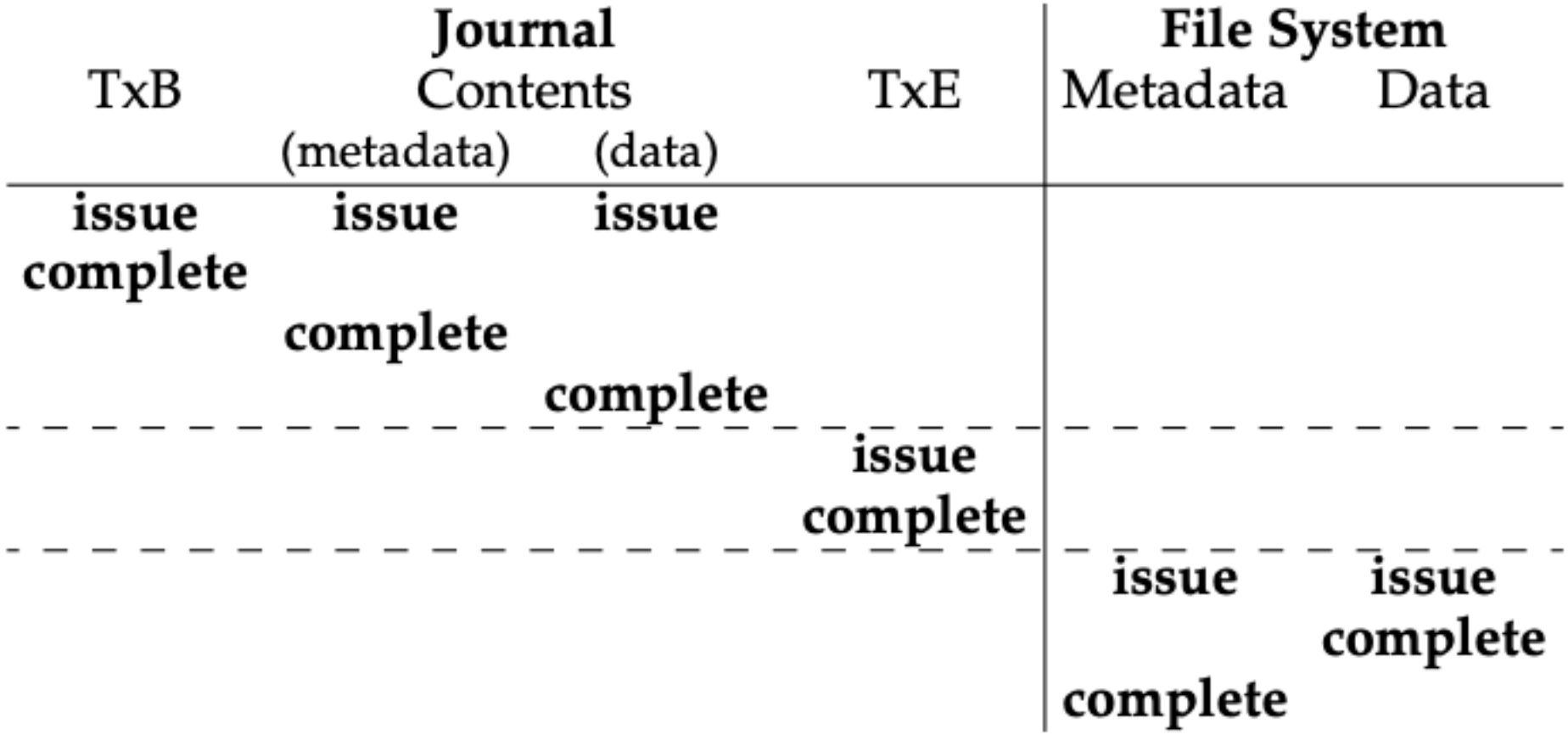
1. **Journal write** – write the full metadata and data to be updated (inode, bitmap and data block) to log
2. **Checkpoint** – write the metadata and data to their final locations in the file system

Journal Protocol

Protocol (version 2):

1. **Journal write** – write the contents of the transaction (TxB, metadata and data)
2. **Journal commit** – write the transaction commit block (TxE)
3. **Checkpoint** – write the metadata and data to their final locations in the file system

Data Journaling Timeline

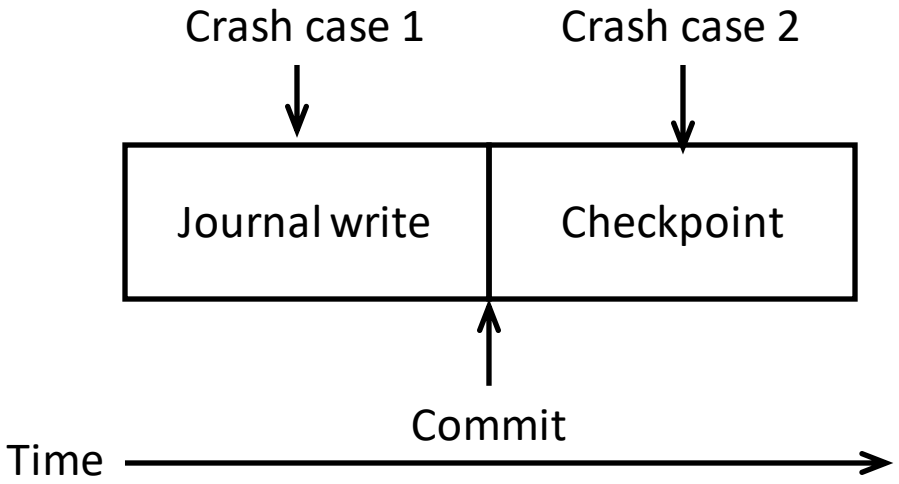


Recovery

Two cases

1. If crash happens before transaction written to log, simply ignore the pending update, the file system is in a consistent state
2. If crash happens after commit to log, but before checkpoint completes, during reboot replay every committed transaction in order

Doesn't matter when during checkpointing crash occurs, some of the replayed transactions will just repeat what is already on the disk, end result is a consistent file system



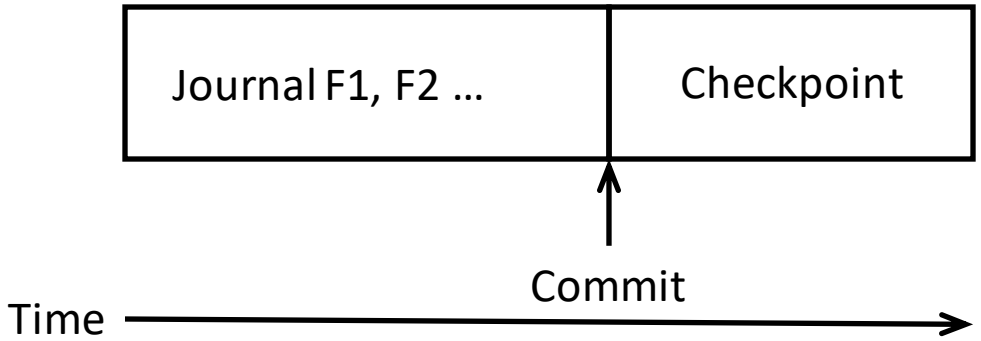
Batching Log Updates for Better Performance

Logging can create a lot of disk traffic

Suppose a directory gets modified several times to write to multiple files, every modification requires logging the directory inode, writing the same block over and over

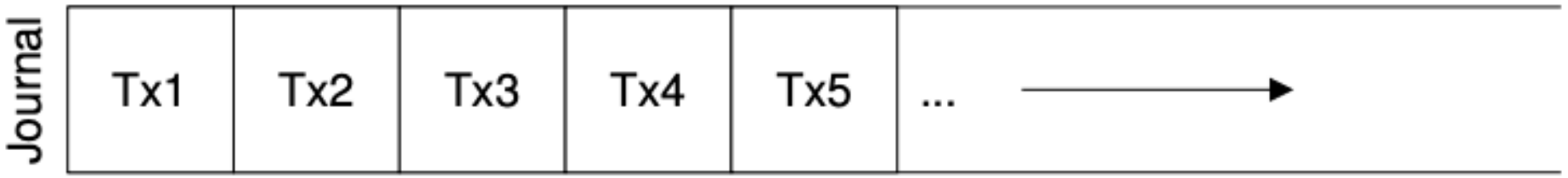
Some file systems use buffering, multiple updates go into one global transaction

All writes within a time period (e.g., 5 seconds) are combined into a single transaction



Limiting Log Size

Log continues to grow with every transaction, can fill disk



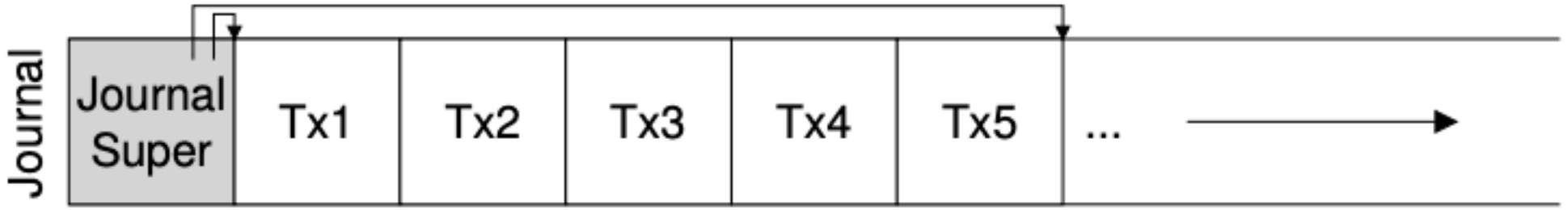
Idea: transactions only need to replay when crash happens during checkpointing, after checkpoint is complete the transaction can be removed

Journal Superblock

Write transaction into a circular log (new transactions written over old freed transactions)

Journal superblock points to the first and last valid transaction in the circular log

When checkpoint completes, first transaction is freed by changing pointer to the next transaction



Adding Free Step

Protocol (version 3):

1. **Journal write** – write the contents of the transaction (TxB, metadata and data)
2. **Journal commit** – write the transaction commit block (TxE)
3. **Checkpoint** – write the metadata and data to their final locations in the file system
4. **Free** – mark the transaction free in the journal by updating the journal superblock

Metadata Journaling

We have now reduced time to recover (replay) and size of log (free), but normal operations of the file system are still slow

Need to write everything to the journal first, doubling write traffic

Idea: reduce writes to log by only logging metadata (bitmaps and inodes) and not data blocks, the data blocks are written directly to the file system

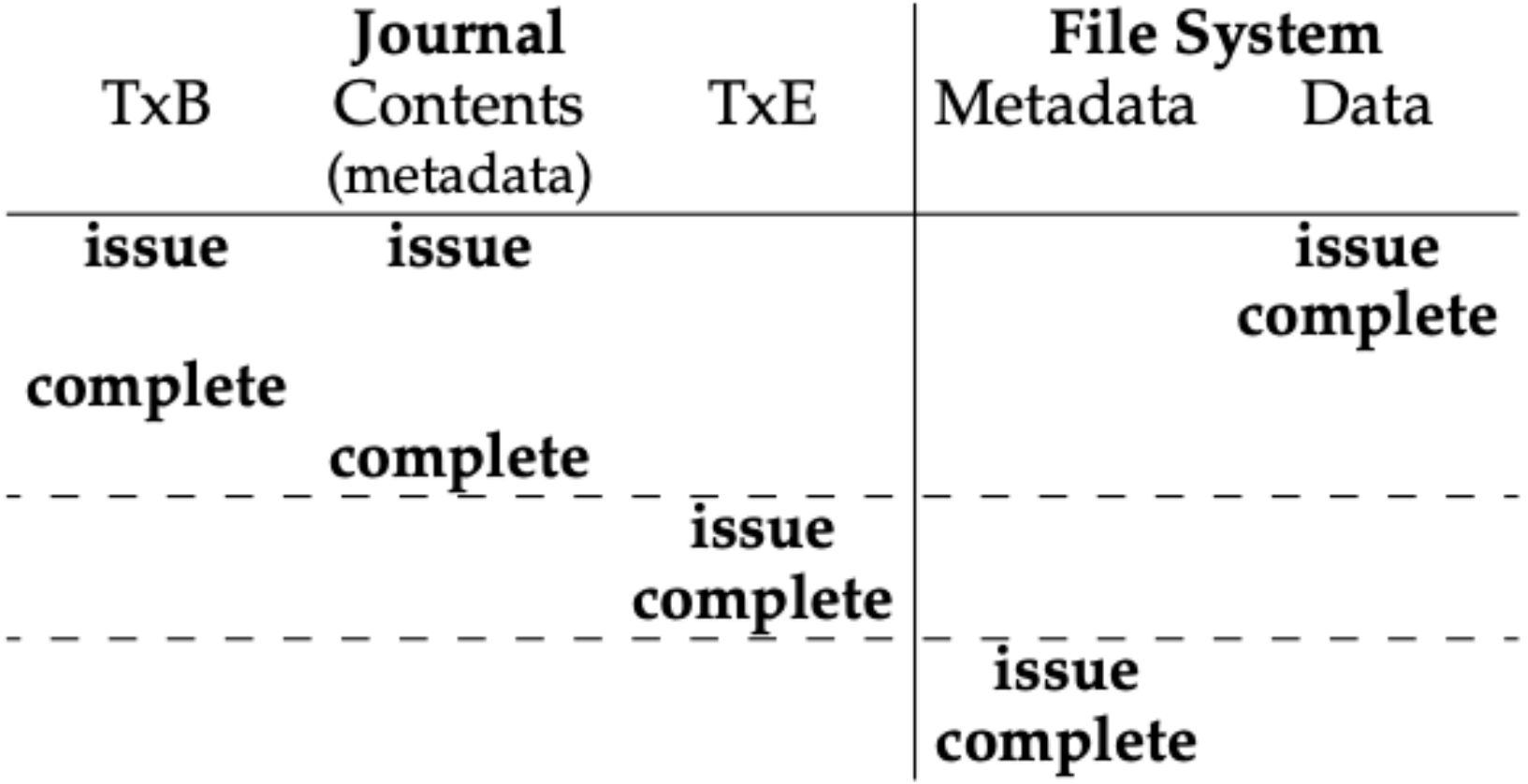
Question: when to write the data blocks to make the file system recoverable?

Metadata Journaling Protocol

Protocol (version 4):

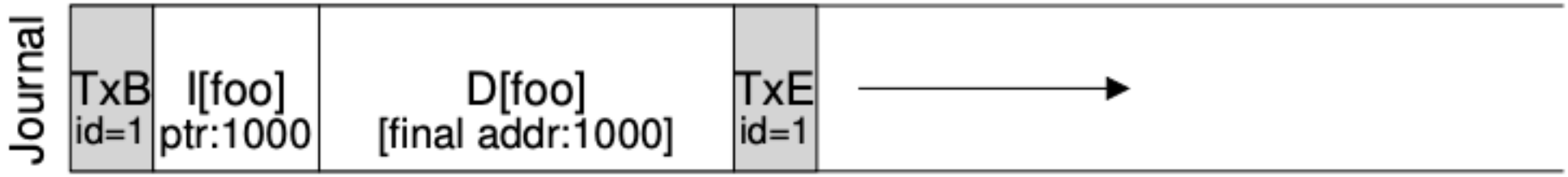
1. **Data write** – write the data to final location and wait for completion
2. **Journal metadata write** – write the begin block and metadata to the log, wait for writes to complete
3. **Journal commit** – write the transaction commit block (TxE) to the log, wait for write to complete
4. **Checkpoint metadata** – write the contents of the metadata update to their final locations within the file system
5. **Free** – mark the transaction free in the journal by updating the journal superblock

Metadata Journaling

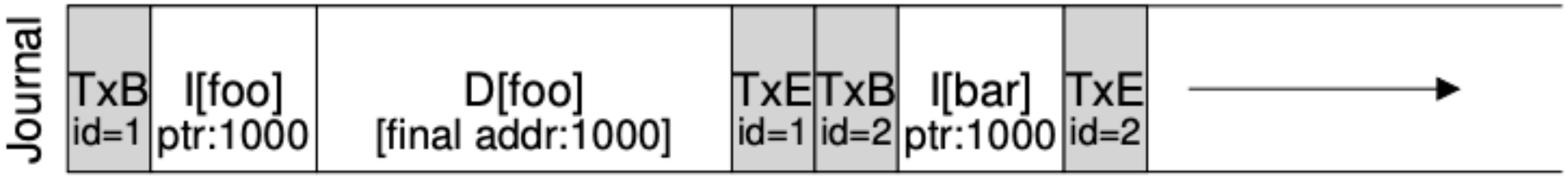


Tricky corner cases when deleting files/directories

User creates directory (note: D[foo] is journaled because it is directory metadata)



User deletes directory and creates new file that reuses data block (note: data block for bar is not journaled because it stores "pure" data not metadata)



System crashes before checkpointing the above journal --> data block of new file bar is overwritten!

Solutions: never reuse data block before the delete transaction is checkpointed; revoke the transactions of the deleted file/directory.