Operating Systems

Youjip Won

KAIST

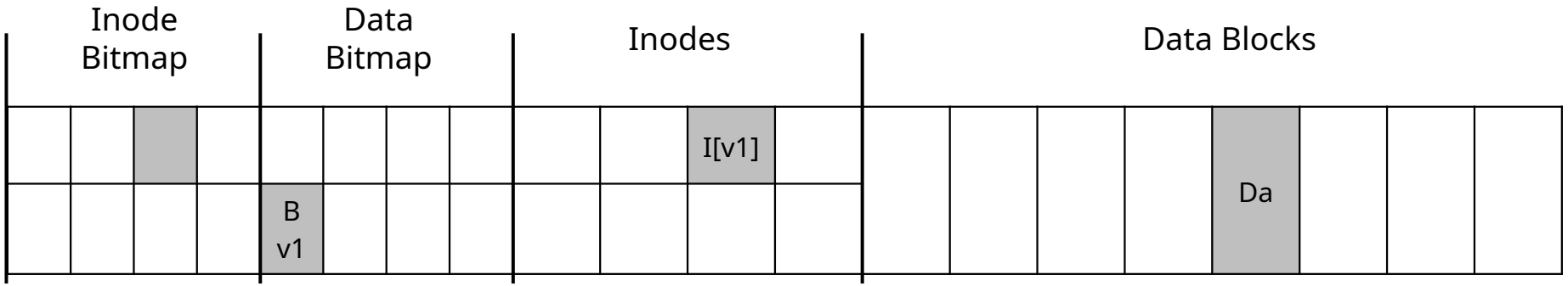
42. Crash Consistency: FSCK and Journaling

Overview

- File system data structures must **persist.**
  - files, directories, all of the other metadata ,etc
- How to update persistent data structure?
  - If the system crashes or loses power, on-disk structure will be in **inconsistent** state.
- In this chapter, we describe how to update file system consistently

An Example of Crash Consistency

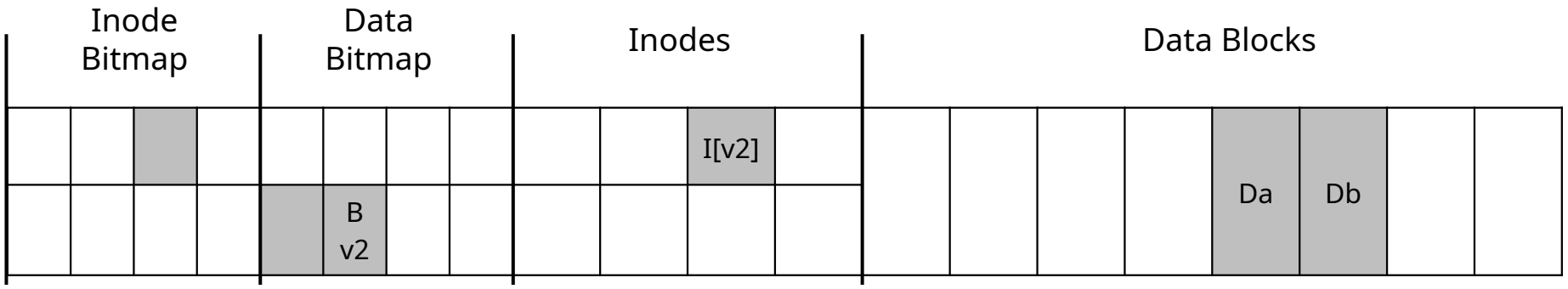
- Scenario
  - Append of a single data block to an existing file.



Before Append a single data block

An Example of Crash Consistency (Cont.)

- File system perform three writes to the disk.
  - inode I[v2]
  - Data bitmap B[v2]
  - Data block (Db)



After Append a single data block

Crash Scenario

- Only one of the below block is written to disk.
  - Data block (Db): lost update
  - Update inode (I[v2]) block: garbage, consistency problem
  - Updated bitmap (B[v2]): space leak

- Two writes succeed and the last one fails.
  - The inode(I[v2]) and bitmap (B[v2]), but not data (Db).: consistent
  - The inode(I[v2]) and data block (Db), but not bitmap(B[v2): inconsistent
  - The bitmap(B[v2]) and data block (Db), but not the inode(I[v2]): inconsistent

Crash-consistency problem (consistent- update problem)

Solution

- The File System Checker (**`fsck`**)
  - `fsck` is a Unix tool for finding inconsistencies and repairing them.
  - super block: if the number of blocks in the filesystem is larger than the filesystem size
  - free blocks: find all the blocks accessible from the root directory and see if it matches the block bitmap.
  - inode state: check if the state of each inode is valid
  - inode link: check if the reference count for each inode is consistent
  - check if a block is shared by the two inodes.
  - check for “bad” block pointer: ”bad” block pointer is the one that points to the location that lies outside the filesystem partition.
  - directory: Check if . and .. are properly set up. Make sure that there are only one hardlink for a directory.

Solution

- **Journaling** (or Write-Ahead Logging)
  - Before overwriting the on-disk structures in place, write down a little note on the disk, describing what you are to do.
  - Writing this note is the “write ahead”. The structure that is the destination of the “write ahead” is called log. hence, This is Write-Ahead Logging.

Journaling

- File system reserves some small amount of space within the partition or on another device.

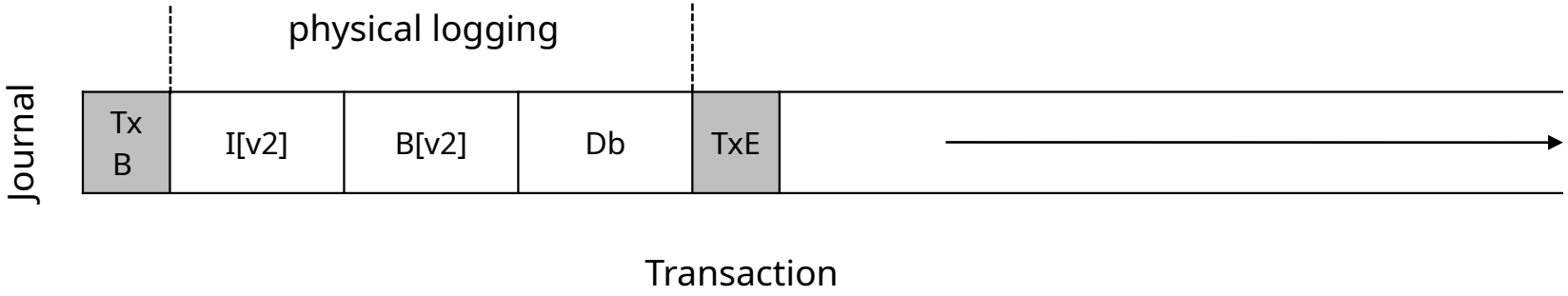
without journaling

with journaling

Data Journaling

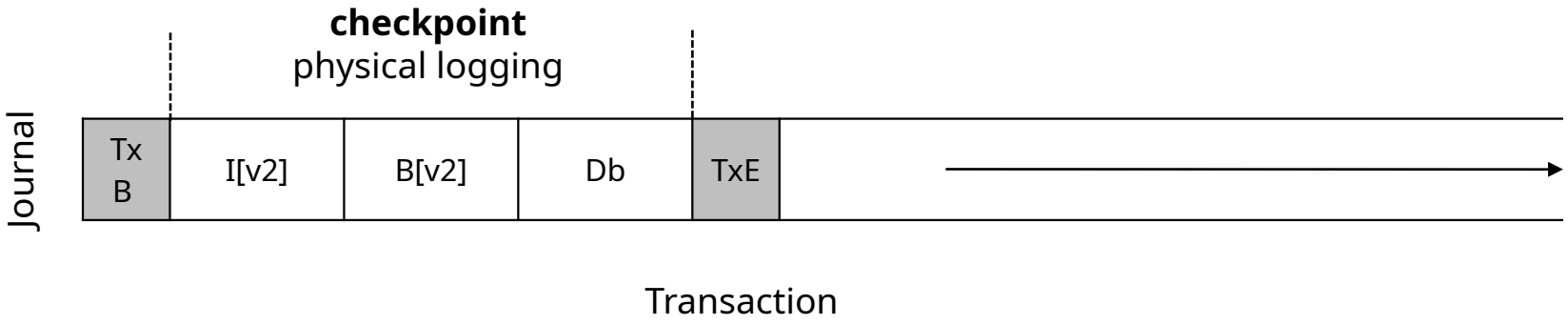
- Lets' update a file (appending a data block to a file). Following structures are updated.
  - inode (I[v2]), bitmap (B[v2]), and data block (Db)

- First, **Journal write**: write the transaction as below.
  - TxB: Transaction begin block (including transaction identifier)
  - TxE: Transaction end block
  - others: contain the exact contents of the blocks



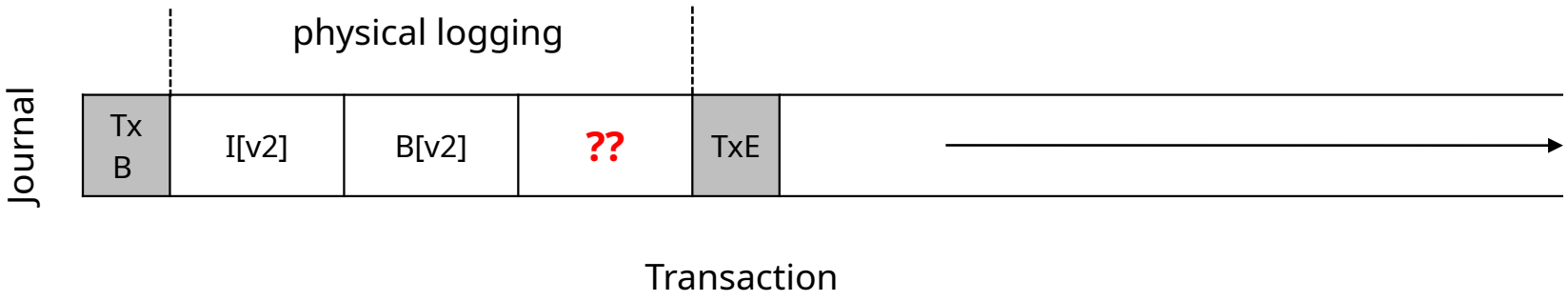
Data Journaling (Cont.)

- Second, **Checkpoint**: Write the physical log to their original disk locations.



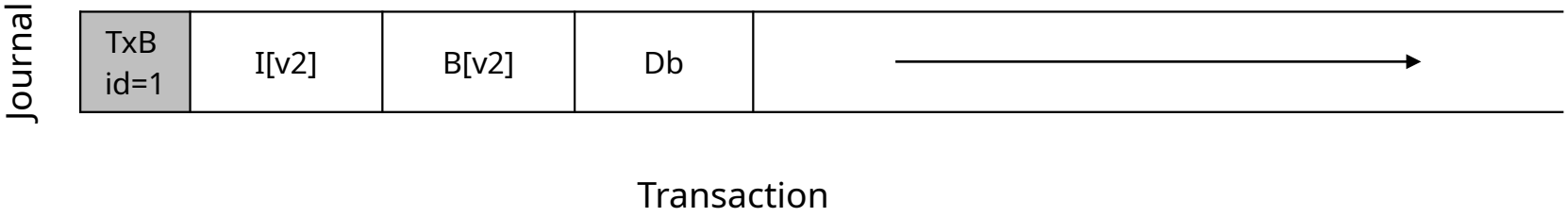
Crash during Data Journaling

- Wat if a *crash occurs* during the writes to the journal?

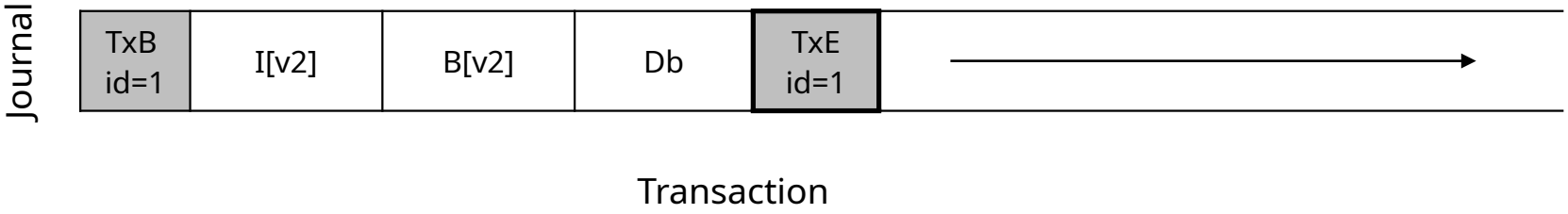


To avoid data being inconsistent

- First, write all blokes **except the TxE block** to journal.



- Second, The file system issues the write of the TxE.



To avoid data being inconsistent (Cont.)

- **Journal write**: write the contents of the transaction to the log
- **Journal commit**: write the transaction commit block
- **Checkpoint**: write the contents of the update to their locations.

Recovery

- If the crash happens, before the transactions is written to the log
  - The pending update is skipped.

- If the crash happens, after the transactions is written to the log, but before the checkpoint.
  - **Recover** the update as follow:
    - Scan the log and lock for transactions that have committed to the disk.
    - Transactions are replayed.

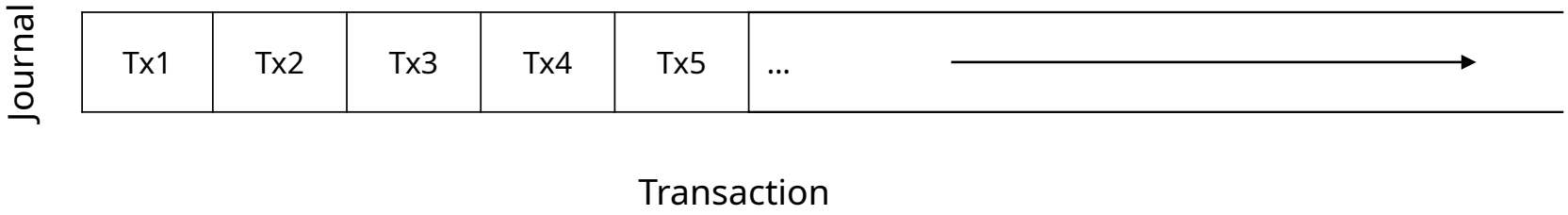
Batching Log Updates

- If we create two files in same directory, the same inode and the directory entry block is to the log and committed twice.

- To reduce excessive write traffic to disk, journaling manage the **global transaction**.
  - Write the content of the global transaction forced by synchronous request.
  - Write the content of the global transaction after timeout of 5 seconds.

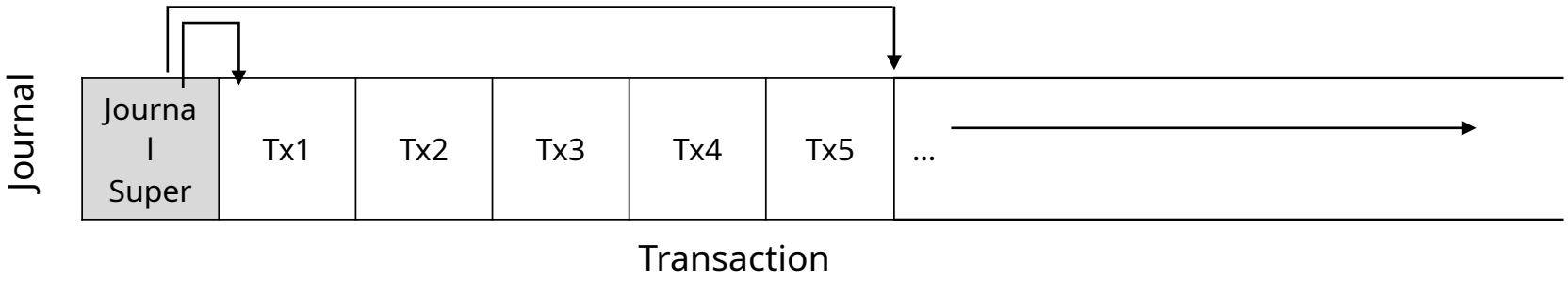
Making the log finite

- The log is of a finite size (**circular log**).
  - To re-using it over and over



Making The log Finite (Cont.)

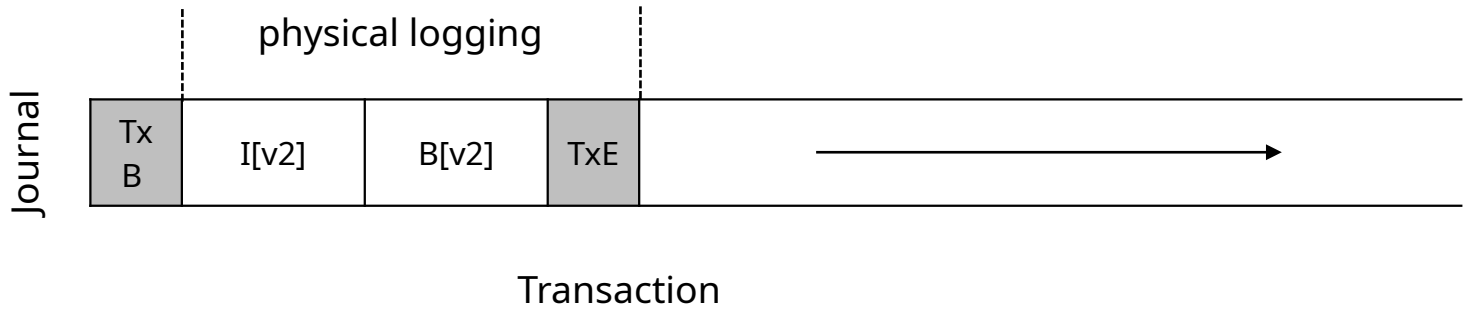
- journal super block
  - Mark the oldest and newest transactions in the log.
  - The journaling system records which transactions have not been check pointed.



Metadata Journaling

- Because of the high cost of writing every data block to disk twice
  - commit to log (journal)
  - checkpoint to on-disk location.

- Filesystem uses ordered journaling (metadata journaling).

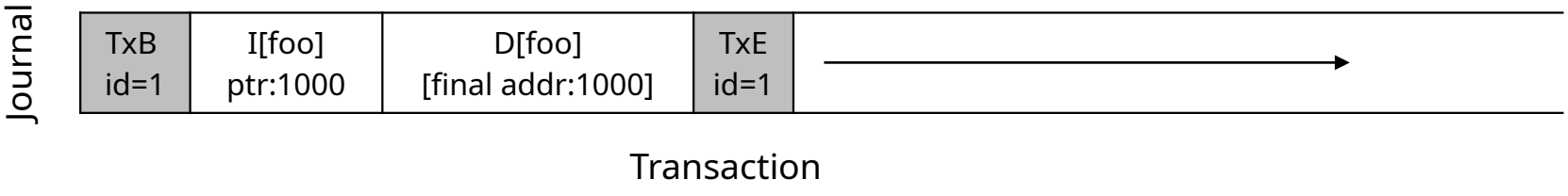


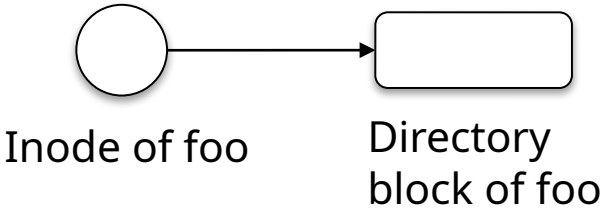
Metadata Journaling (Cont.)

- **Data Write**: Write data to final location
- **Journal metadata write**: Write the begin and metadata to the log
- **Journal commit**: Write the transaction commit block to the log
- **Checkpoint metadata**: Write the contents of the metadata to the disk
- **Free**: Later, mark the transaction free in journal super block

Tricky case: Block Reuse

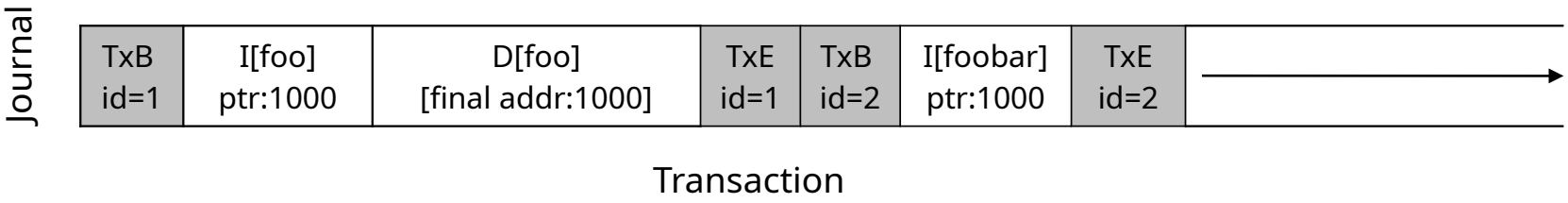
- Revoke record: some metadata should not be replayed.
- Scenario.
  1. Directory “foo” is updated.

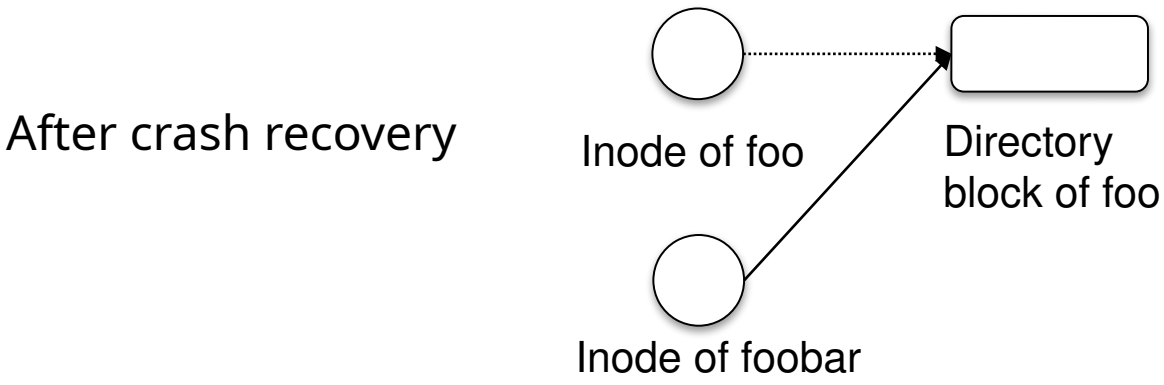




Tricky case: Block Reuse

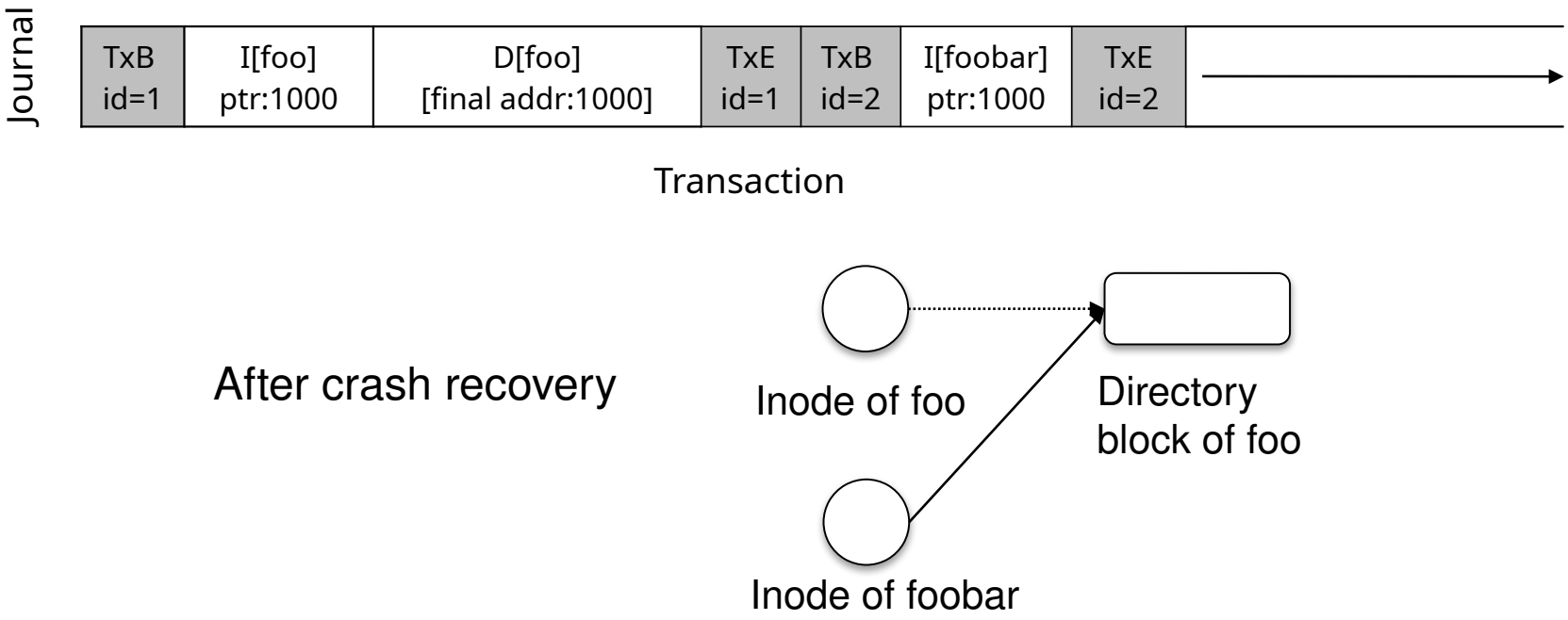
- Scenario.
  1. Delete “foo” directory, freeing up block 1000 for reuse
  2. Create a file “foobar”, reusing block 1000 for data





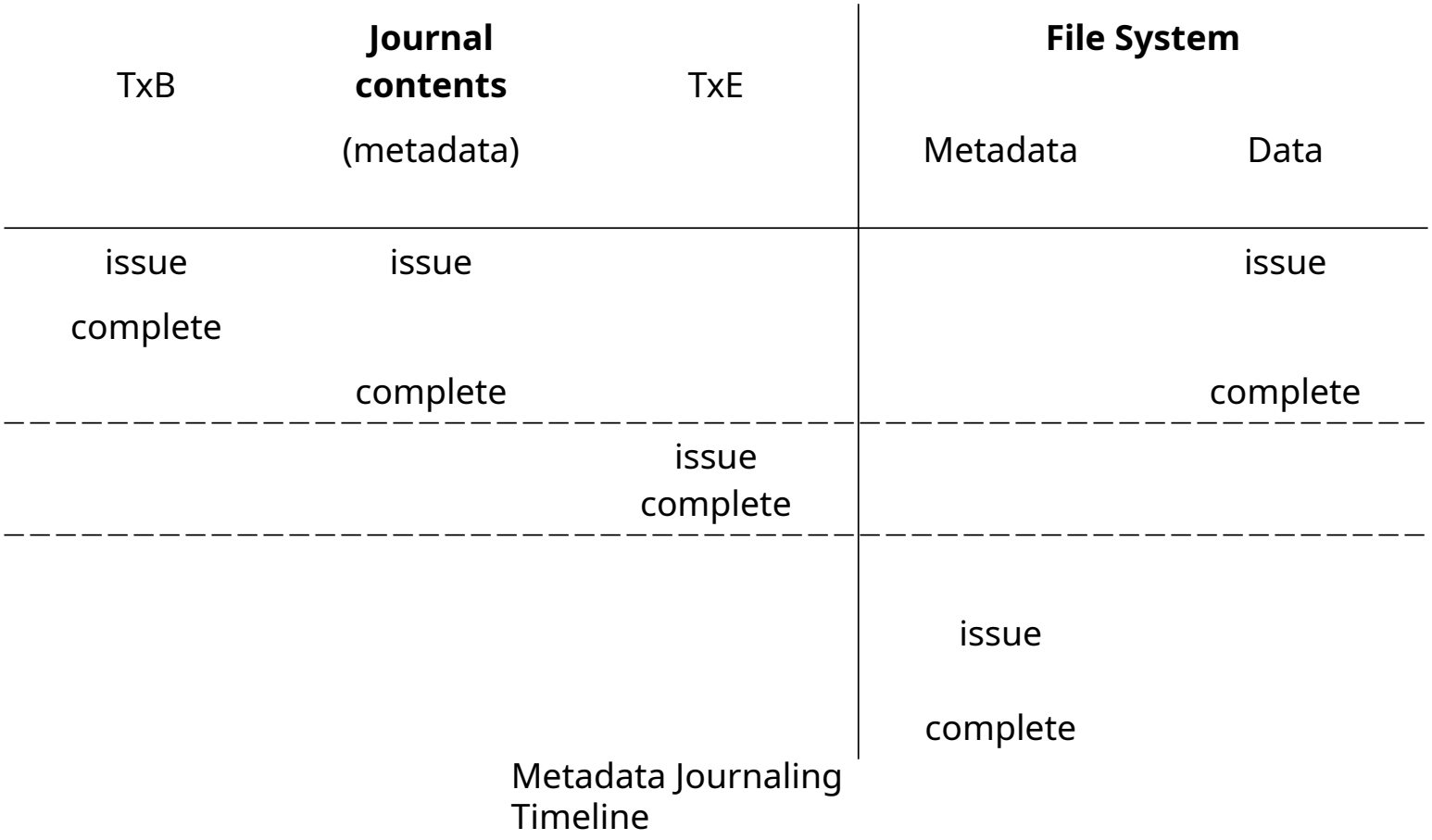
Tricky case: Block Reuse

- Problem: allocate the block that was deleted but was not checkpointed.

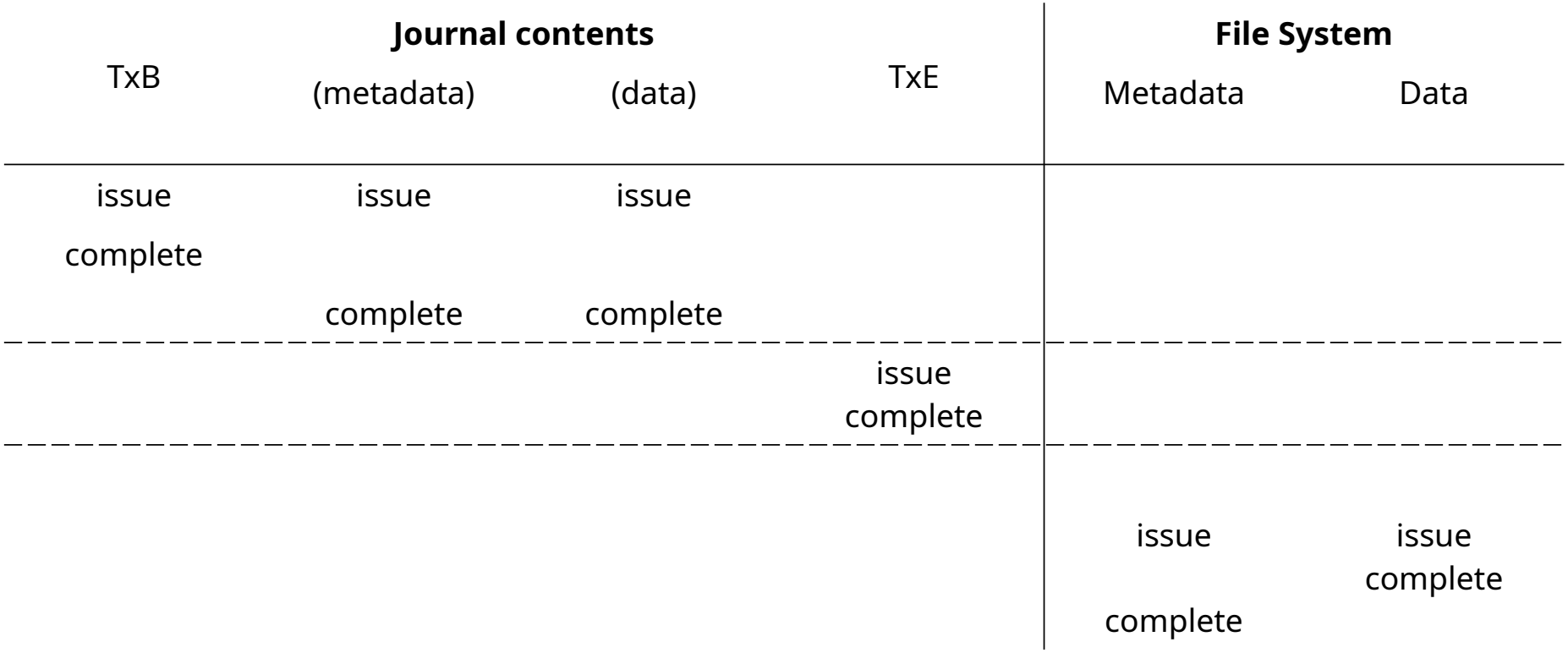


- Solution1: Do not use the deleted block until it is checkpointed.
- Solution 2: When deleting a directory, record “revoke” at the journal.

Metadata Journaling Timeline



Data Journaling Timeline



Data Journaling Timeline

Summary

- The problem of crash consistency
  - Filesystem becomes inconsistent so users cannot use it
- Approaches to attacking this problem
  - **fsck**: it works but is likely too slow to recover on modern systems.
  - **Journaling**: It reduces recovery time from O(size-of-the-disk-volume) to O(size-of-the-log)
- Two modes of journaling
  - Metadata journaling(Ordered journaling): Log only metadata only after data written
  - Data journaling: Log both of metadata and data