File System Consistency

Jin-Soo Kim
(*jinsoo.kim@snu.ac.kr*)

Systems Software &
Architecture Lab.

Seoul National University

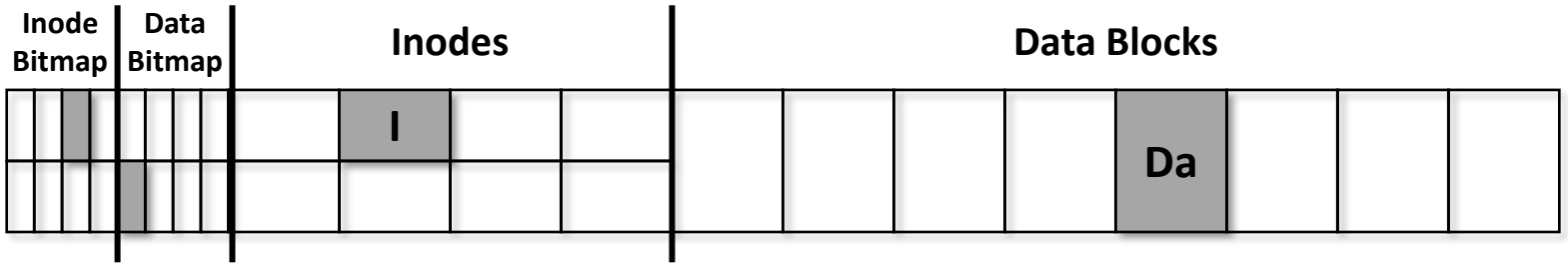
Spring 2020

Crash Consistency

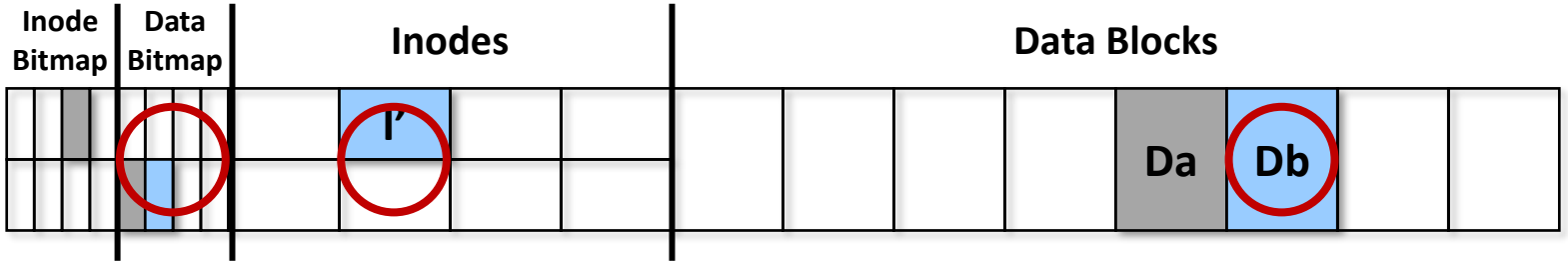
- File system may perform several disk writes to complete a single system call
  - e.g., `creat()`, `write()`, `unlink()`, `rename()`, ...
  - But, disk only guarantees atomicity of a single sector write
- If file system is interrupted between writes, the on-disk structure may be left in an inconsistent state
  - Power loss
  - System crash (kernel panic)
  - Transient hardware malfunctioning
- We want to move file system from one consistent state to another atomically

Example: Appending Data

- Initial state

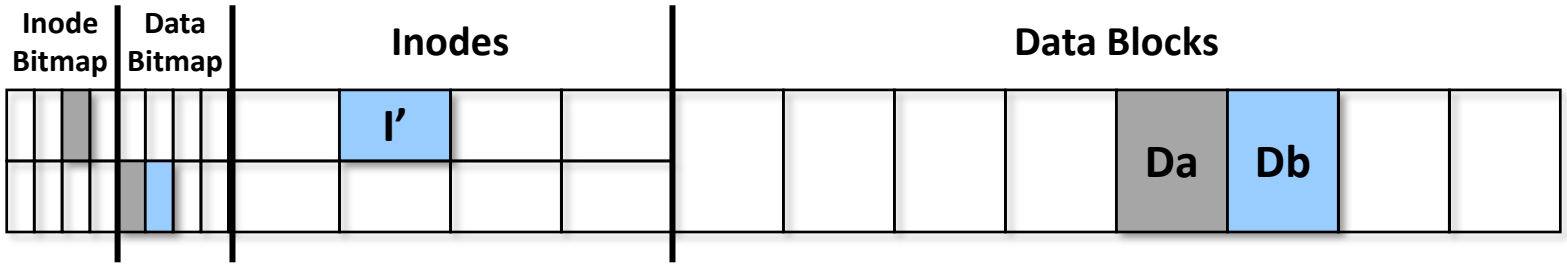


- Appending a data block Db

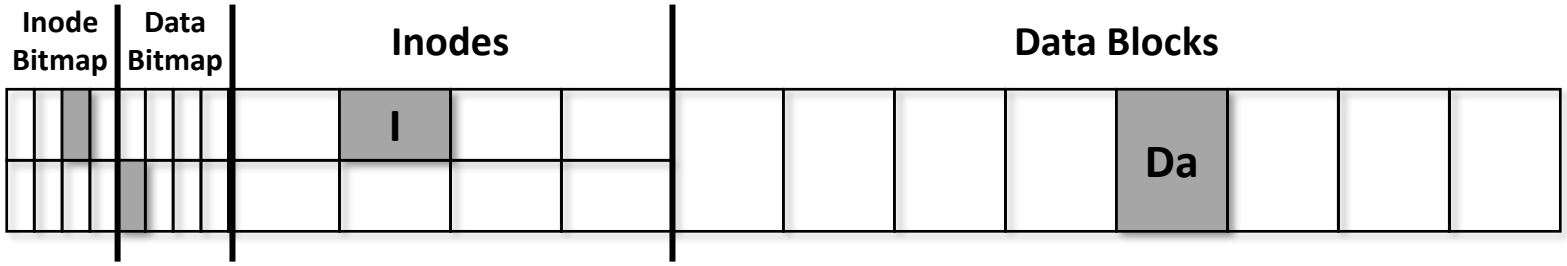


Example: Crash Scenarios (1)

- Everything touched media: No problem

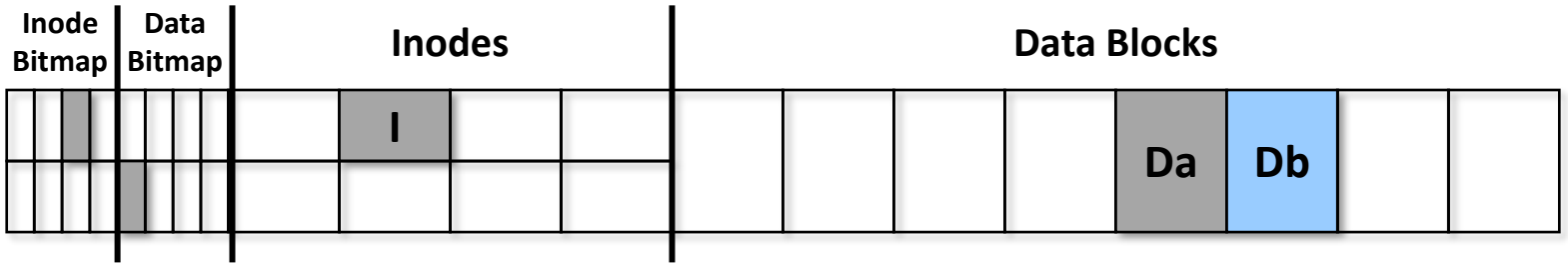


- Nothing touched media: No problem
  - Due to page cache or internal disk write buffer



Example: Crash Scenarios (2)

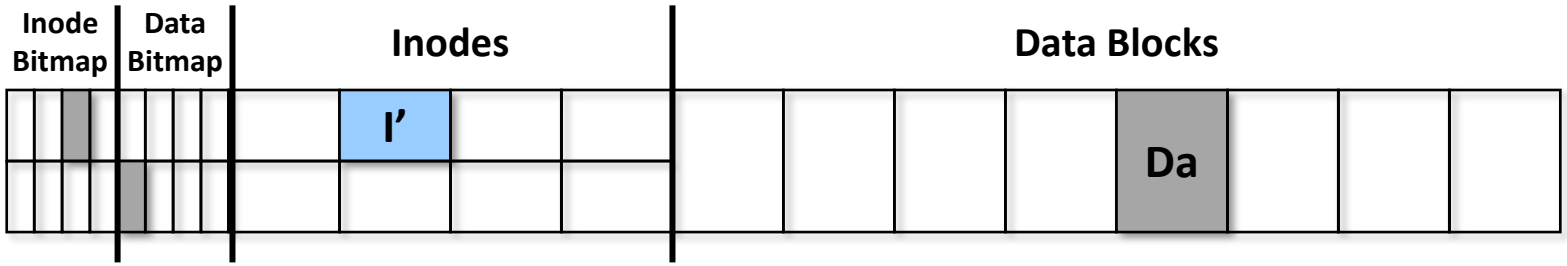
- Only data block (Db) is written: OK



  - No inode points to data block 5 (Db)
  - Data bitmap says data block 5 is free

Example: Crash Scenarios (3)

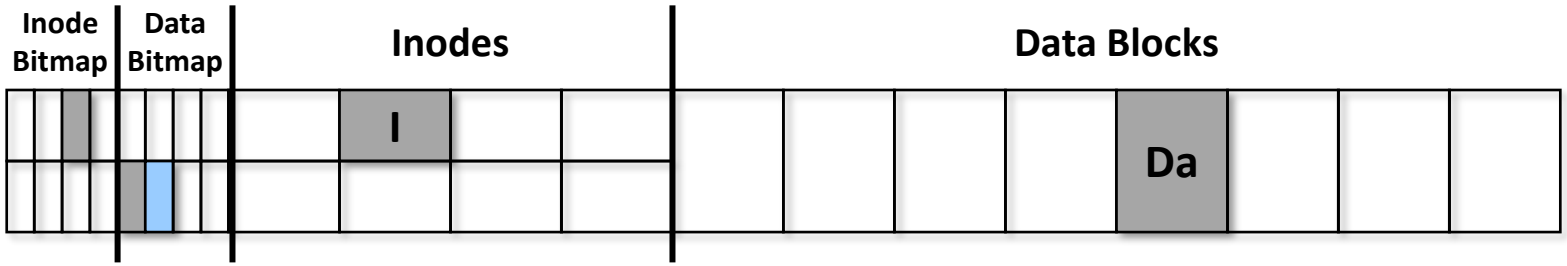
- Only updated inode (I') is written: Inconsistency



- Inode I' points to data block 5, but data bitmap says it's free
- Read will get garbage data (old contents of data block 5)
- If data block 5 is allocated to another file later, the same block will be used by two inodes

Example: Crash Scenarios (4)

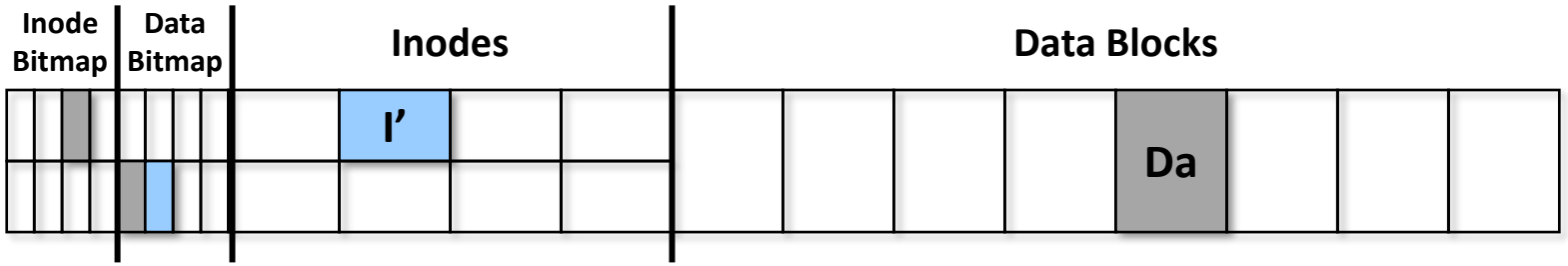
- Only updated data bitmap is written: Inconsistency



- Data bitmap indicates data block 5 is allocated, but no inode points to it
- Data block 5 will never be used by the file system
- Lost data block (space leak)

Example: Crash Scenarios (5)

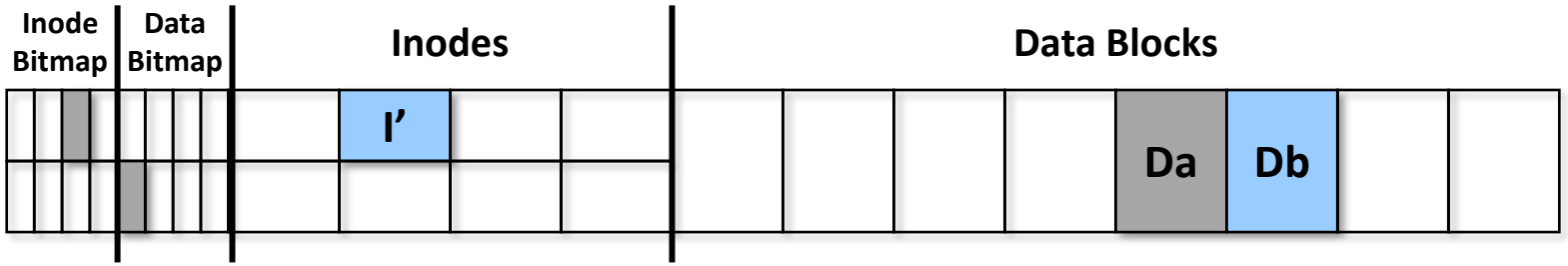
- Only inode and bitmap are written: OK



- File system metadata is completely consistent
- Inode I' has a pointer to data block 5 and data bitmap indicates it is in use
- Read will get garbage data (old contents of data block 5)

Example: Crash Scenarios (6)

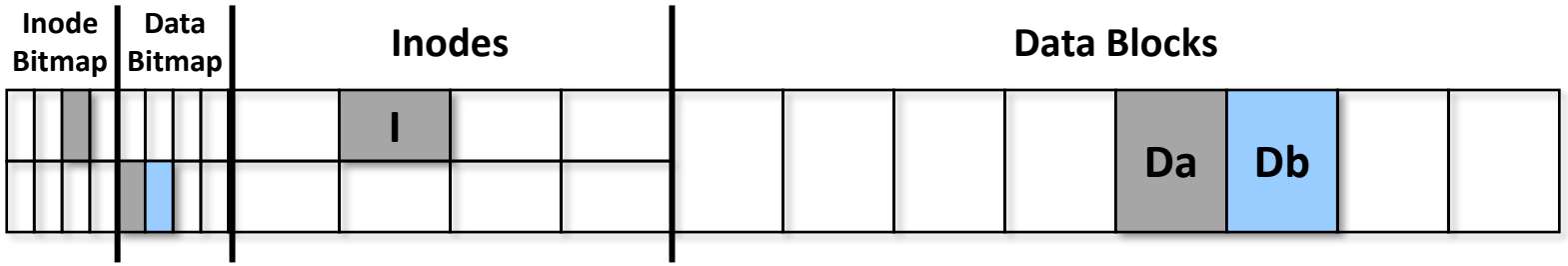
- Only inode and data block are written: Inconsistency



- Inode I' has a pointer to data block 5, but data bitmap indicates it is free
- Data block 5 can be reallocated to another inode (the same block will be used by two inodes)

Example: Crash Scenarios (7)

- Only bitmap and data block are written: Inconsistency



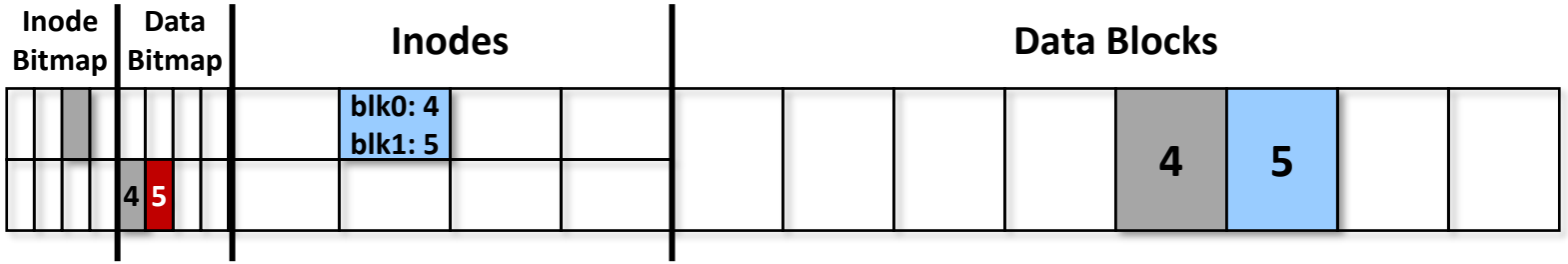
- Data bitmap indicates data block 5 is in use, but no inode points to it
- Data block 5 will never be used by the file system
- Lost data block (space leak)

FSCK

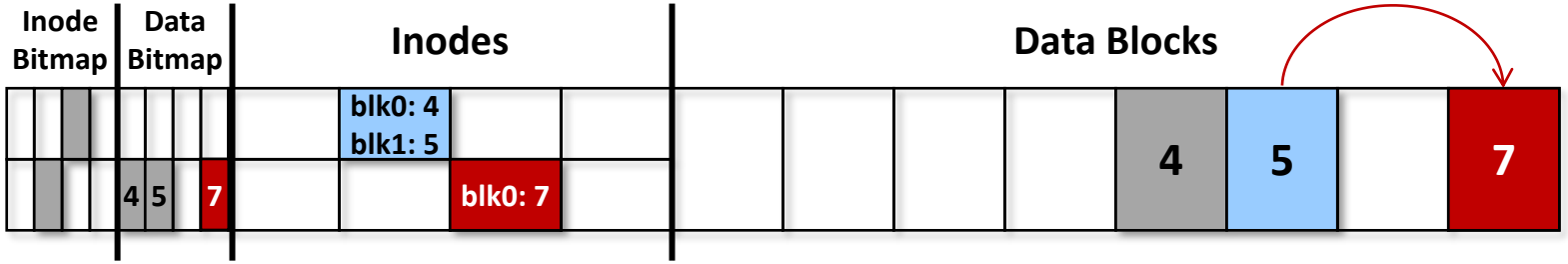
- File System Checker
  - A Unix tool for finding inconsistencies in a file system and repairing them (cf. Scandisk in Windows)
  - Run before the file system is mounted and made available
- After crash, scan whole file system for contradictions and "fix" it if needed
  - Inode bitmap consistency
  - Data bitmap consistency
  - Inode link count
  - Duplicated/invalid data block pointers
  - Other integrity checks for superblock, inode, and directories

FSCK: Fixing Data Blocks

- Inconsistent data bitmap

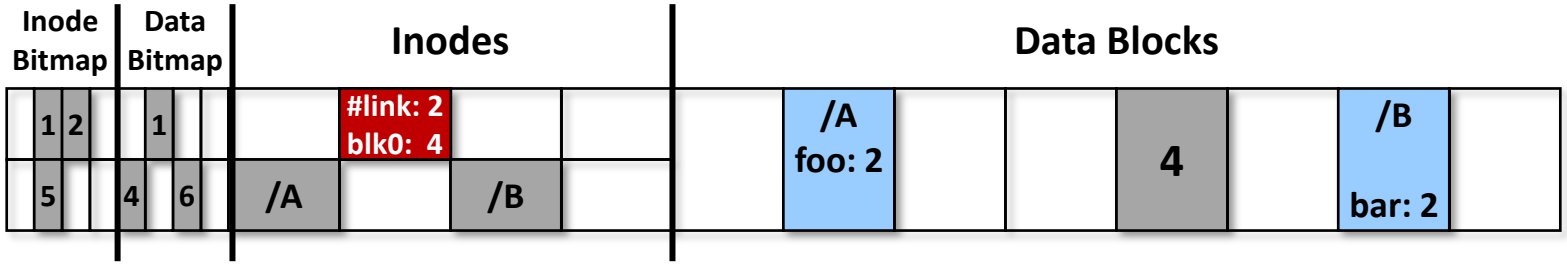


- Duplicated data block pointers

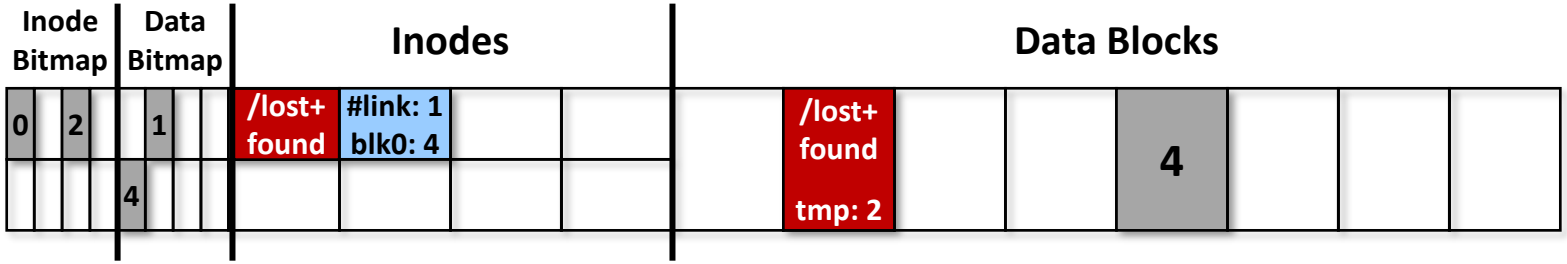


FSCK: Fixing Inode Link Count

- Inconsistent link count

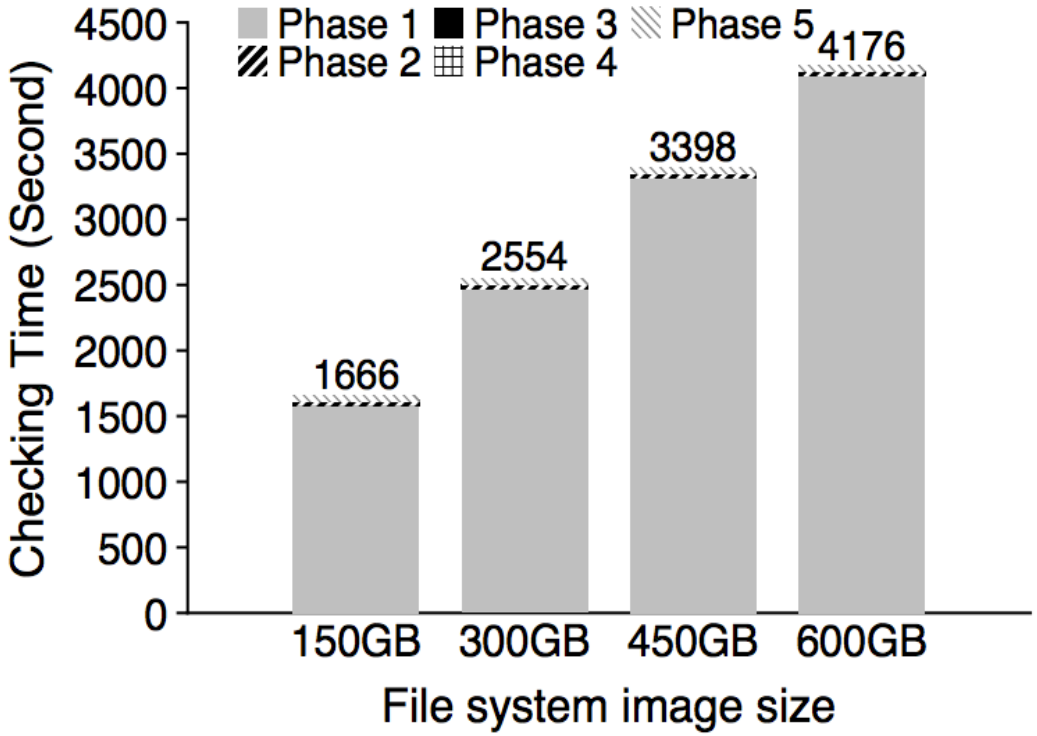


- Lost file: no corresponding directory entry
  - Create a temporary file in `/lost+found`



FSCK Problems

- Too slow!
  - 5 phases
  - Need to scan the entire directory tree and block pointers
  - Fsck'ing a 600GB disk takes ~ 70 min.



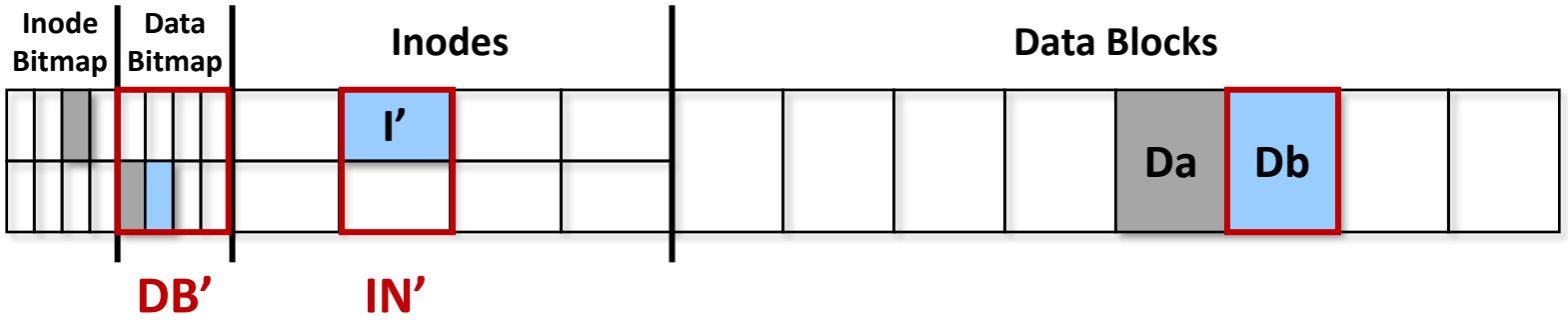
- Requires intricate knowledge of the file system
- Not always obvious how to fix file system image
- Don't know "correct" state, just consistent one

Journaling

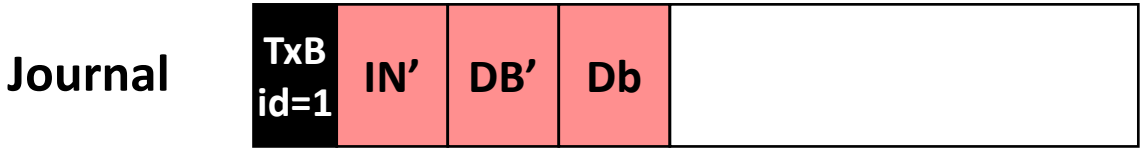
- Write-ahead logging
  - A well-known technique for database transactions
  - Record a log, or journal, of changes made to on-disk data structures to a separate location ("journaling area")
  - Write updates to their final locations ("_____") only after the journal is safely written to disk
  - If a crash occurs:
    - Discard the journal if the journal write is not committed
    - Otherwise, redo the updates based on the journal data
  - Fast as it requires to scan only the journaling area
  - Used in modern file systems:
    Linux Ext3/4, ReiserFS, IBM JFS, SGI XFS, Windows NTFS, …

Example: Appending Data (1)

- Appending a data block Db

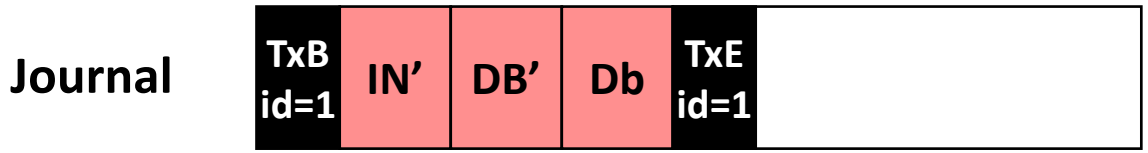


- Step 1: Journal write
  - Write journal header block (TxB), inode block (IN'), data bitmap block (DB') and data block (Db)

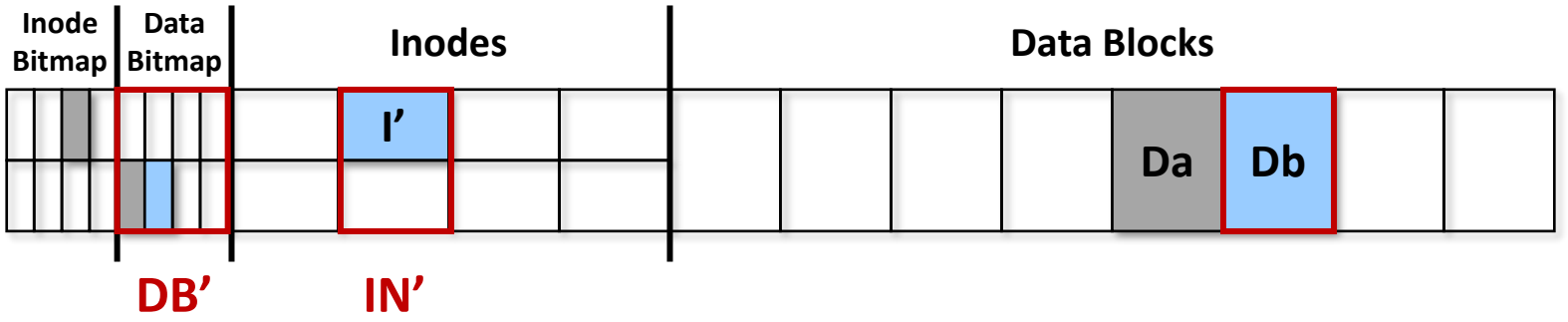


Example: Appending Data (2)

- Step 2: Journal commit
  - Write journal commit block (TxE)

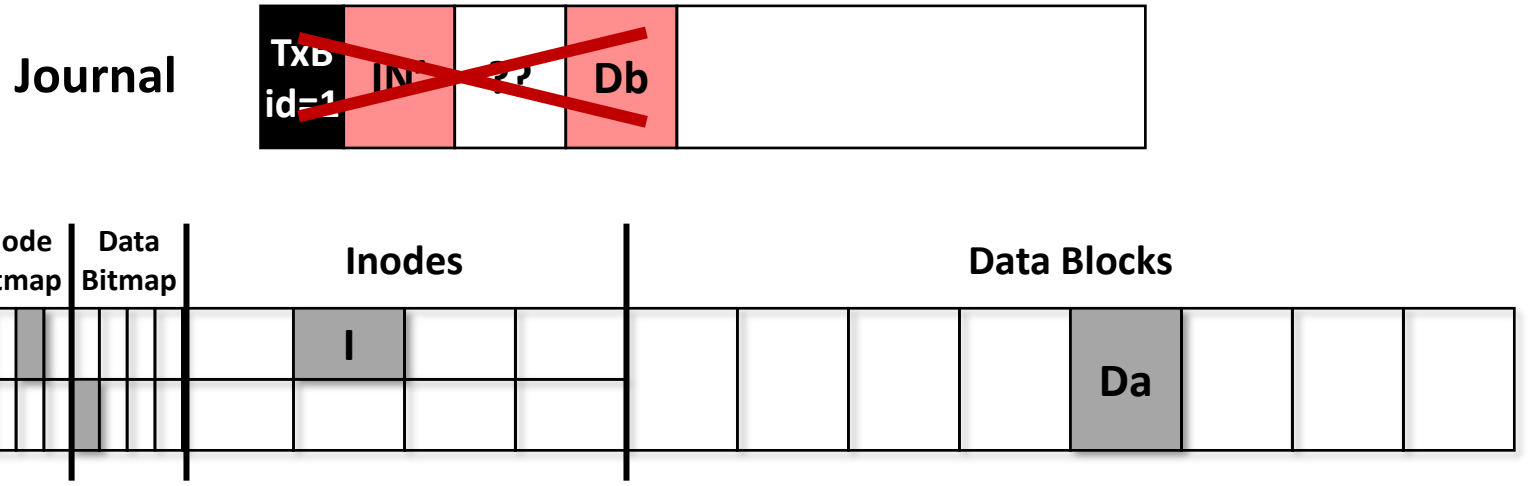


- Step 3: Checkpoint
  - Write updates to their final on-disk locations (IN', DB', Db)



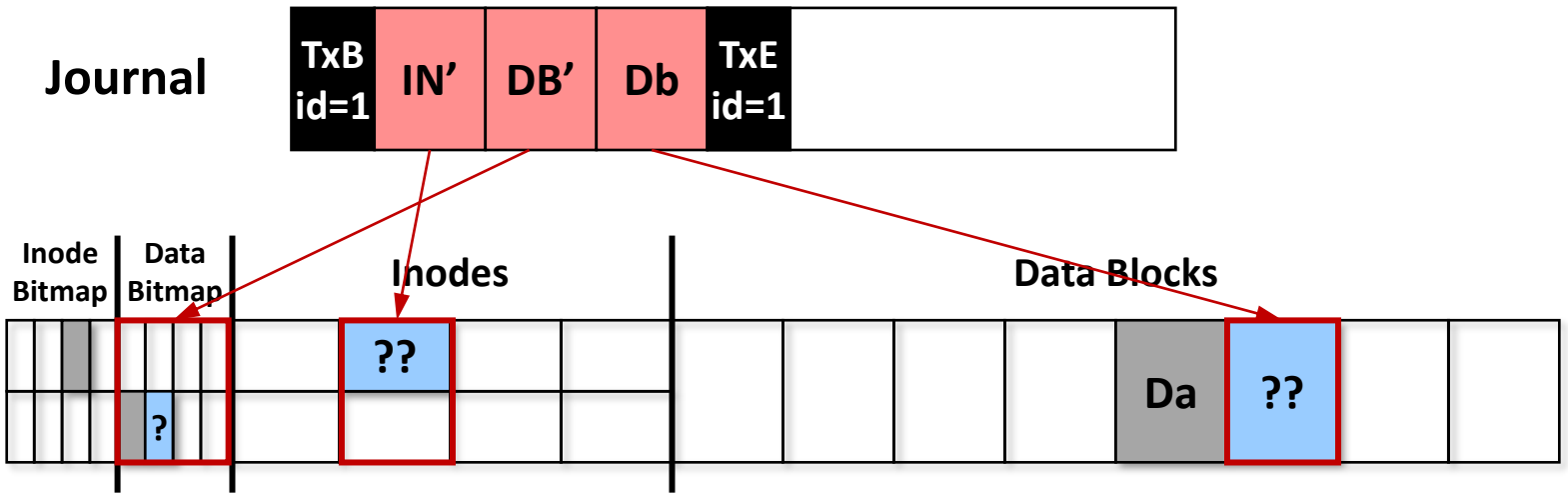
Example: Recovery (1)

- Crash between step 1 & 2
  - Journal write has not been committed
  - Simply discard the journal
  - File system is rolled back to the state before data block Db is appended



Example: Recovery (2)

- Crash between step 2 & 3
  - Doesn't matter which metadata/data blocks were actually updated
  - Roll-forward recovery (redo logging): overwrite their final on-disk locations using the journal data



Optimizing Journaling

- Circular log
  - Mark the transaction free and reuse the journal space
- Batching log updates
  - Buffer all updates into a global transaction
  - e.g., 5 seconds in Ext3/4
- Journal checksums
  - Eliminate write barrier between journal write & commit
- _____ journaling
  - Only guarantees metadata consistency
  - Ordered journaling in Ext3/4: force the data write before the journal is committed so as not to point to garbage