Operating System: File System Consistency

Sang Ho Choi (shchoi@kw.ac.kr)
School of Computer & Information Engineering
KwangWoon University

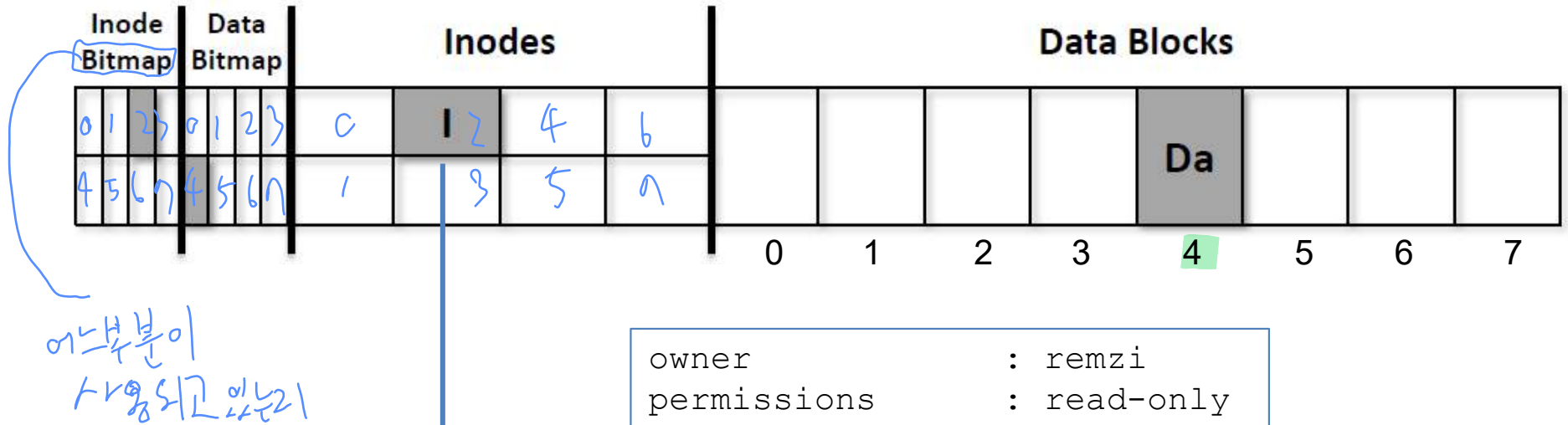
Crash Consistency

- 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 ondisk 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

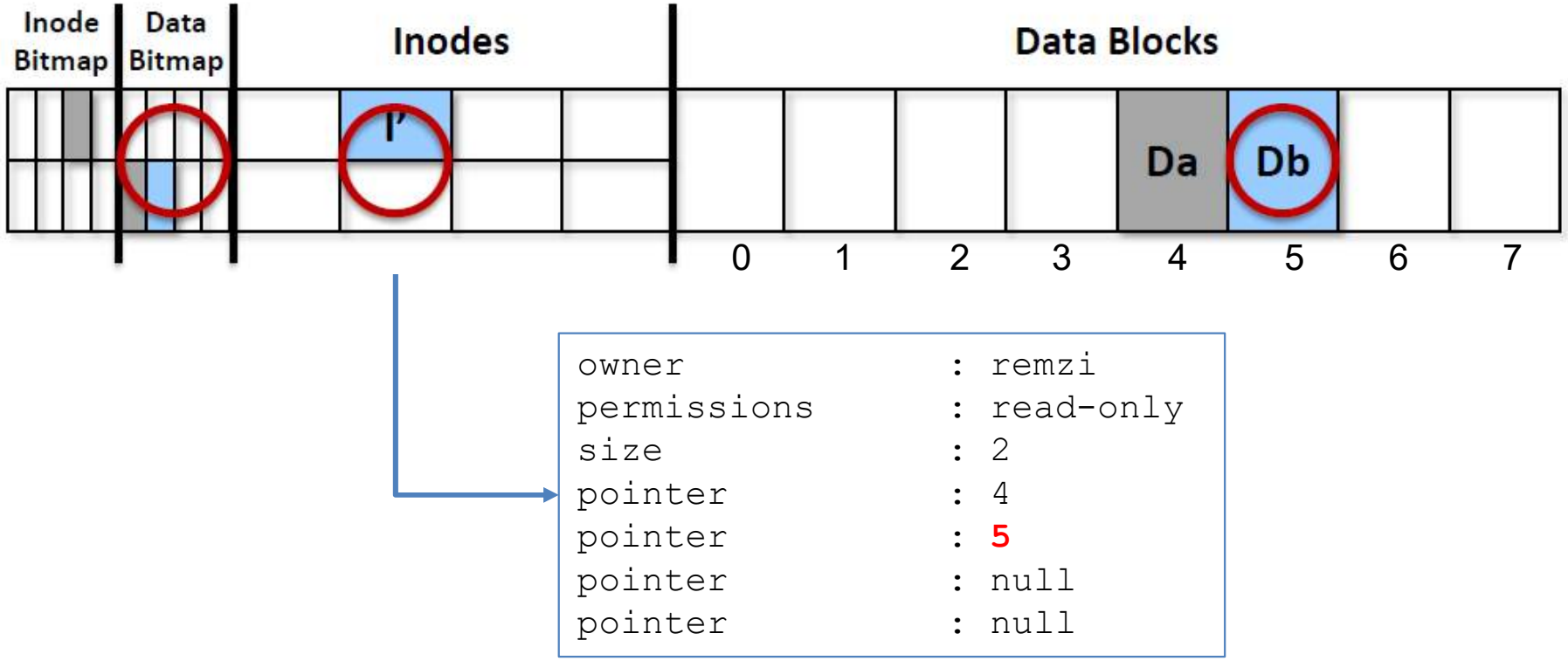
8bit, 1/inode	8bit, 1/data block	8 total, spread across 4 block	8 total




```
owner           : remzi
permissions     : read-only
size            : 1
pointer         : 4
pointer         : null
pointer         : null
pointer         : null
```

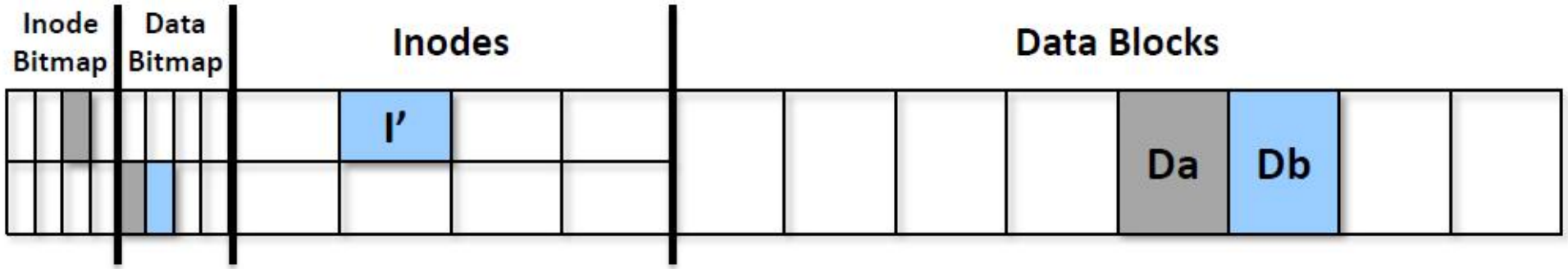
Example: Appending Data

- Appending a data block Db
  - Data bitmap is updated
  - Inode is updated (I')
  - New data block is allocated (Db)

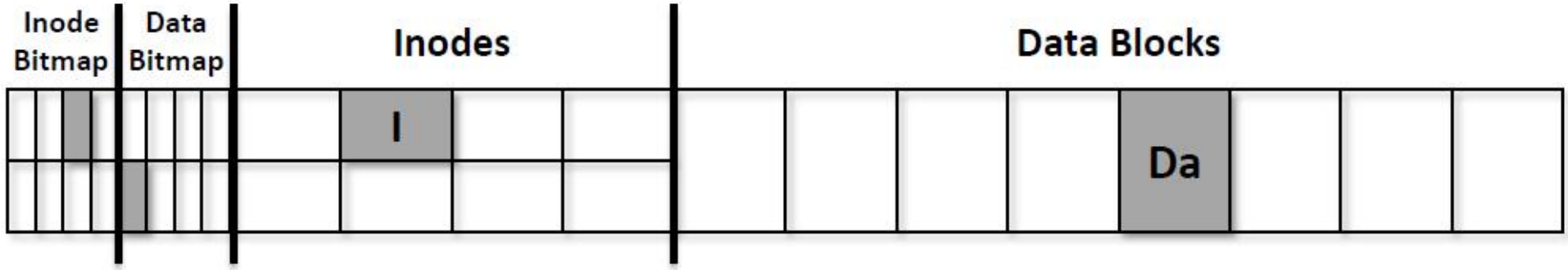


Example: Crash Scenarios (1)

- Everything touched media: No problem

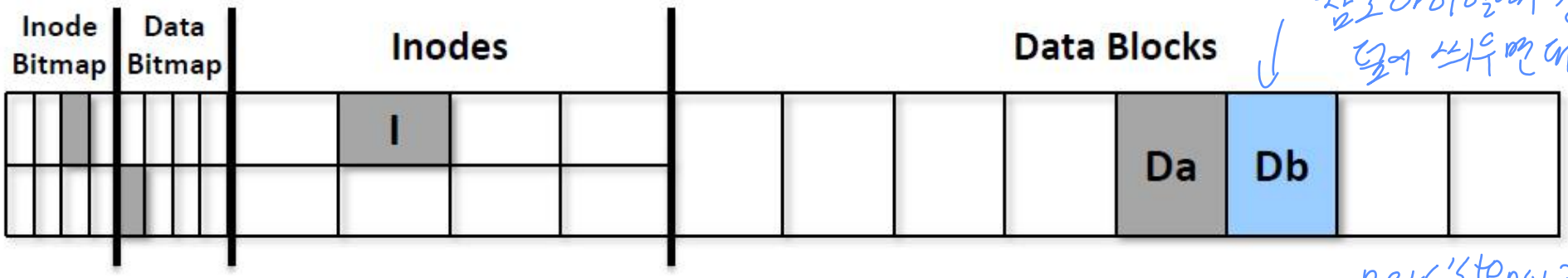


- Nothing touched media: No problem
  - Due to page cache or internal disk write buffer



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

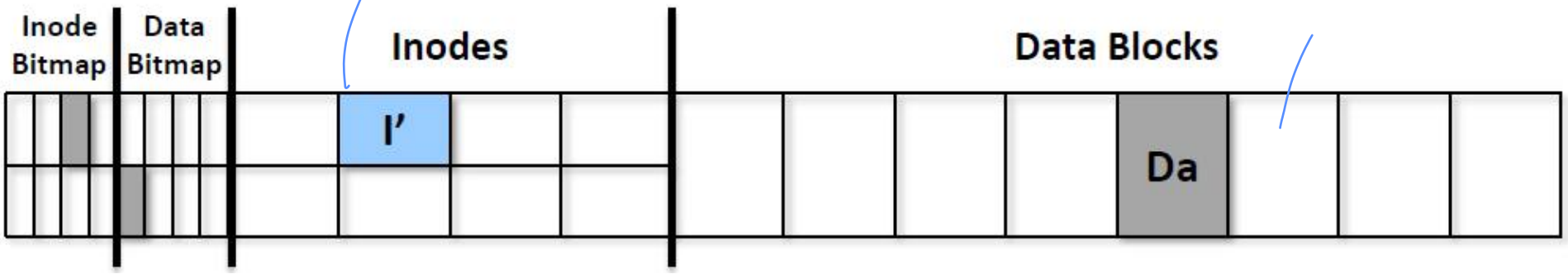
이전 가리키는 Inode가 없으니 어차피 이 블록을 참조하려할때 걍 덮어 쓰우면 되서

persistency 관점에서는 옳기 X.

Inode가 가리키는 데이터와 실제 데이터간의 일치도를 보는게 핵심.

Example: Crash Scenarios (3)

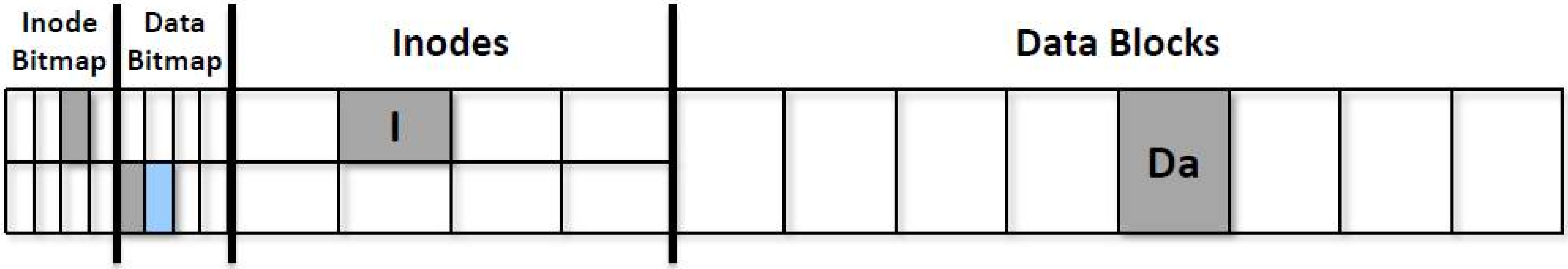
- Only updated inode (I') is written: Inconsistency



  - 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

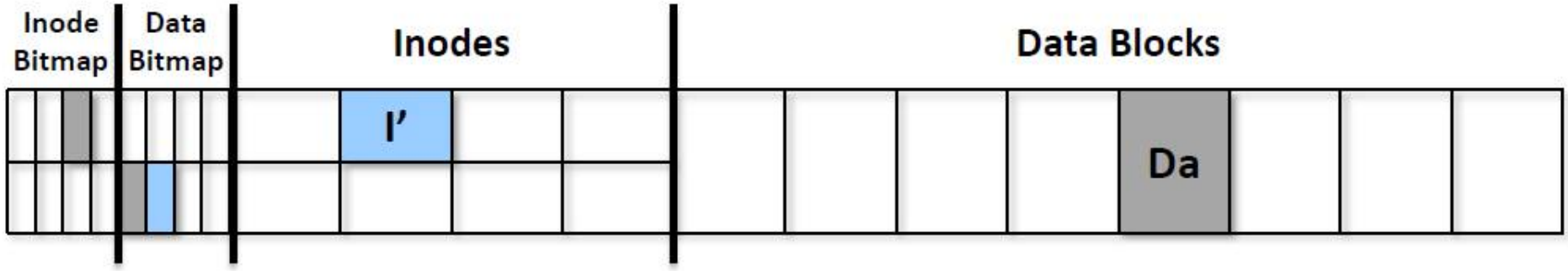


  - Data bitmap indicates data block 5 is allocated, but no inode points to it
  - Data block 5 will never be used by the file system
  - Lost data block (space leak)

Bitmap에서는 사용되고 있다는데
포인터가 없으니까
계속 메모리 누수

Example: Crash Scenarios (5)

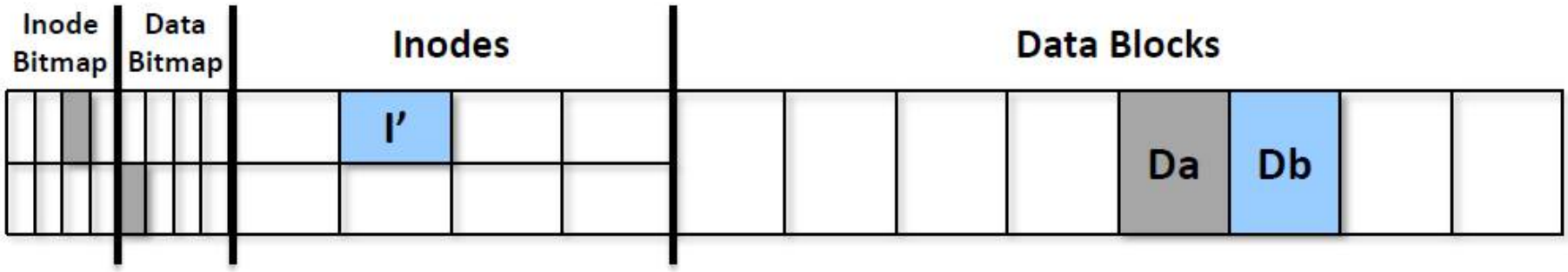
- Only inode and bitmap are written: OK



  - 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)

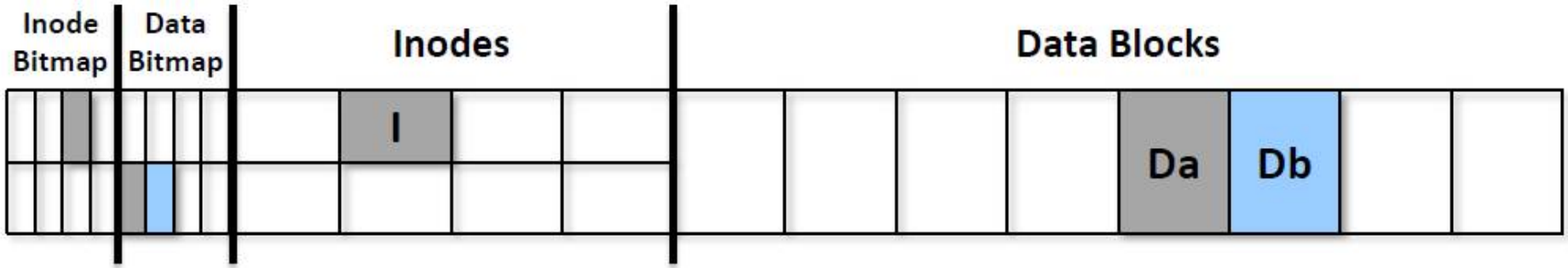
- Only inode and data block are written: Inconsistency



  - 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

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)

결과적으로 Data bitmap과 inode간 불일치가 없다면 시스템 관점에서의 consistency는 유지가 되는거다.

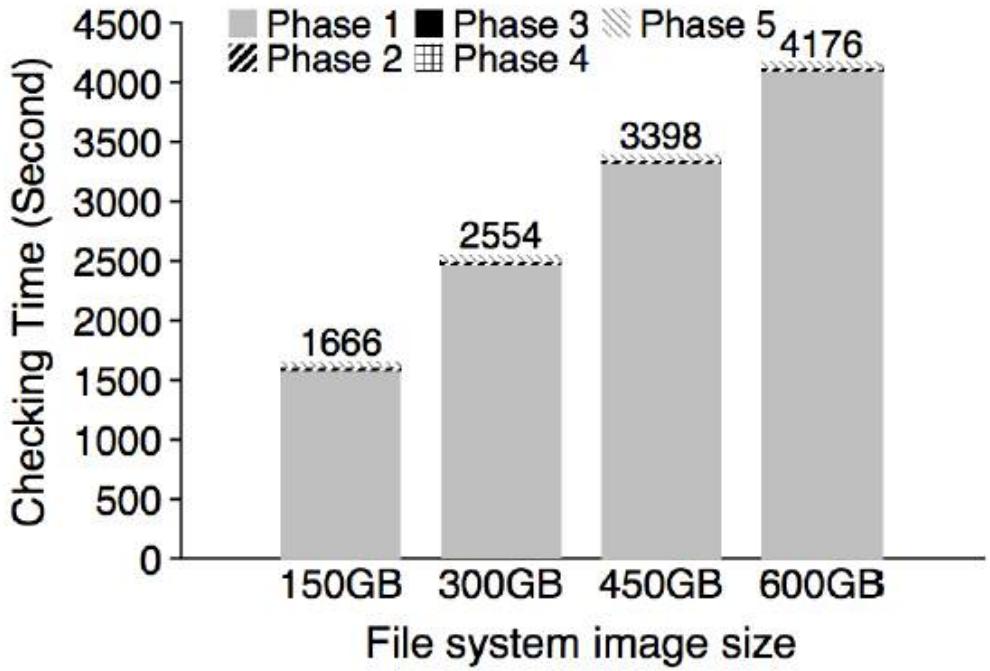
FSCK

- File System Checker
  - A Unix tool for finding inconsistencies in a file system and repairing them (cf. Scandisk in Windows) 불일치를 찾아서 고침
  - 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 Problems

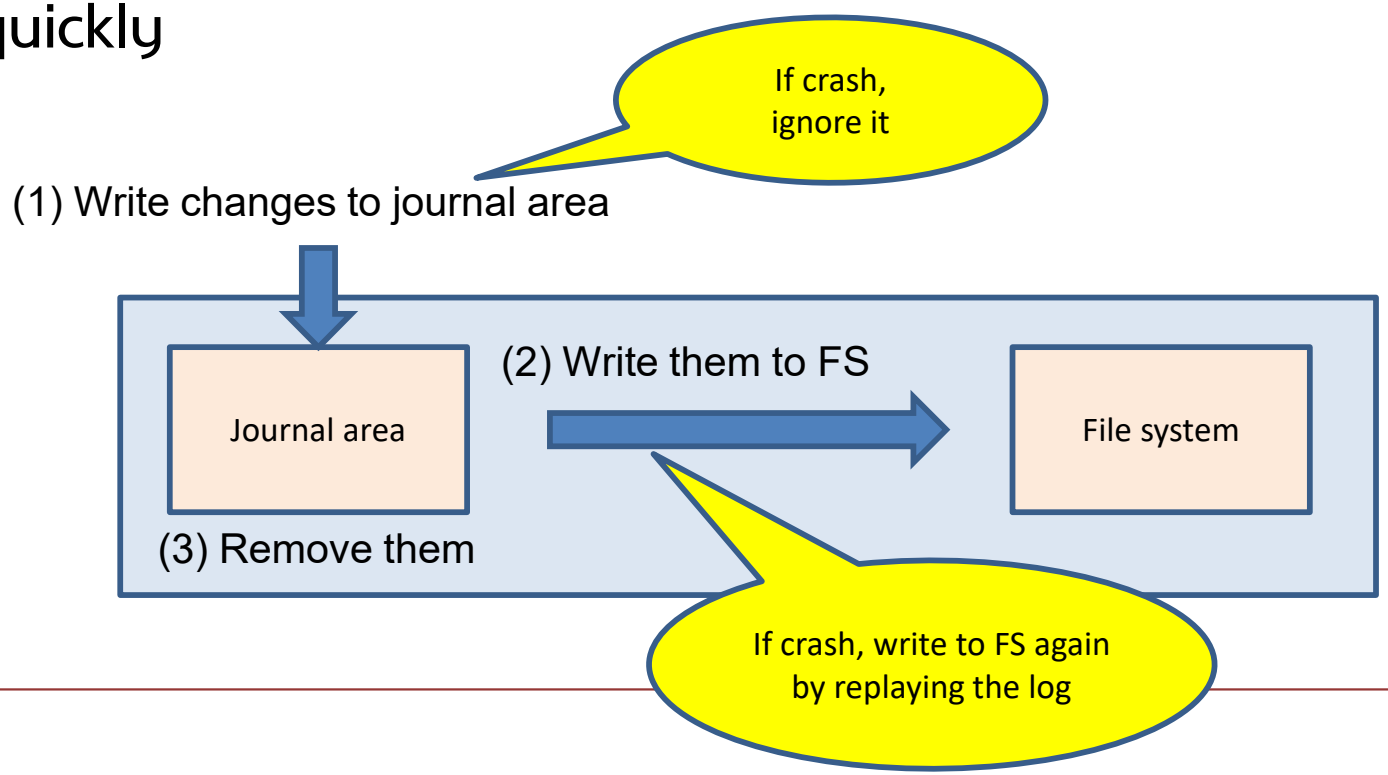
- **Too slow!**
  - Need to scan the entire directory tree and block pointers
  - Fsck'ing a 600GB disk takes ~ 70 min



- 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

- Journaling file system
  - If write a "/a/b"?
    - ➔ b's data write, b's inode write, a's data write, a's inode write, bitmap write(if needed), etc.
  - keeps track of changes in a "journal", which is usually a circular log
  - When a system crash, file systems can be recovered more quickly



Journaling (Cont.)

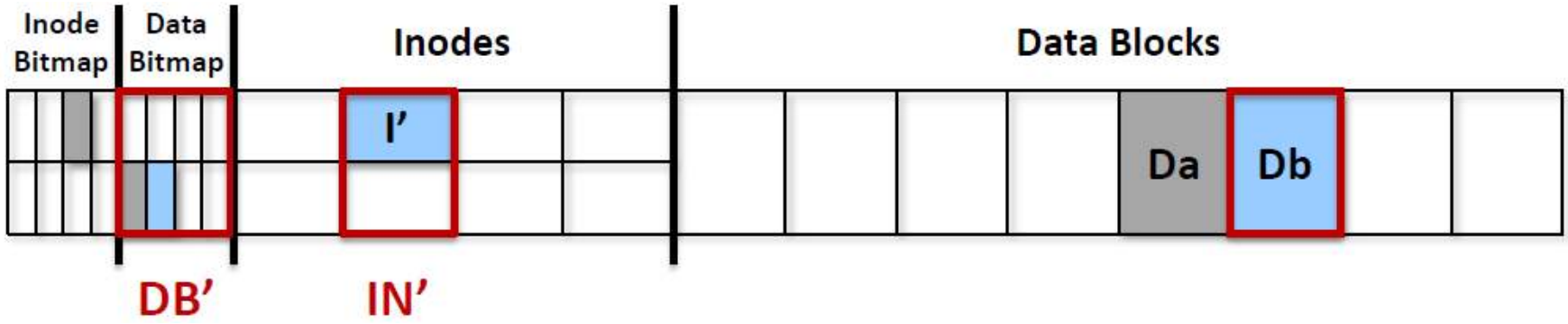
- Write-ahead logging
  - A well-known technique for database transactions
  - Record a log, or journal, of changes made to on-disk data structures to a separate location ("journaling area")
  - Write updates to their final locations ("checkpointing") only after the journal is safely written to disk
  - If a crash occurs:
    - Discard the journal if the journal write is not committed
    - Otherwise, redo the updates based on the journal data

저널에 쓰다 크래쉬면 무시 저널에 있는걸로 쓸라고 할땐 다시쓰기

  - 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

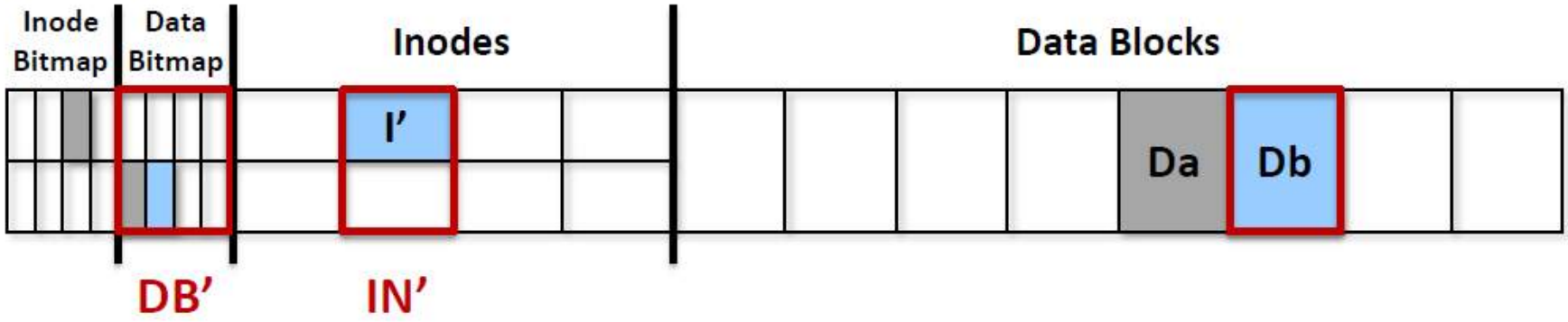


- Step 1: Journal write
  - Write journal header block (TxB), inode block (IN'), data bitmap block (DB') and data block (Db)

Example: Appending Data (2)

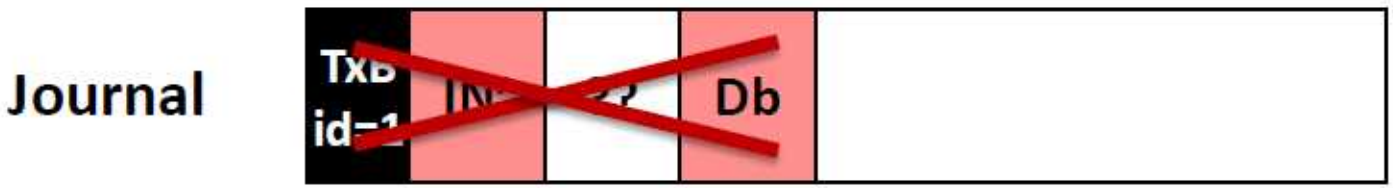
- Step 2: Journal commit
  - Write journal commit block (TxE)

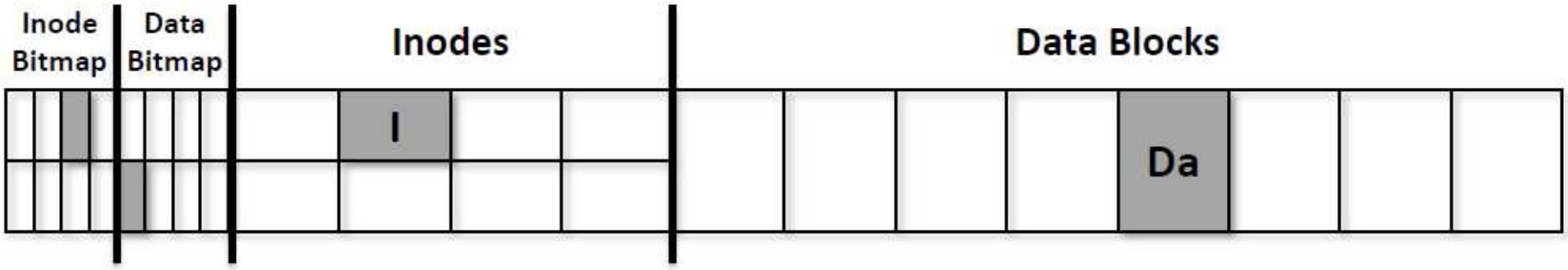
- Step 3: Checkpoint
  - Write updates to their final on-disk locations (IN', DB', Db)



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