Persistence: Crash Consistency

Sridhar Alagar

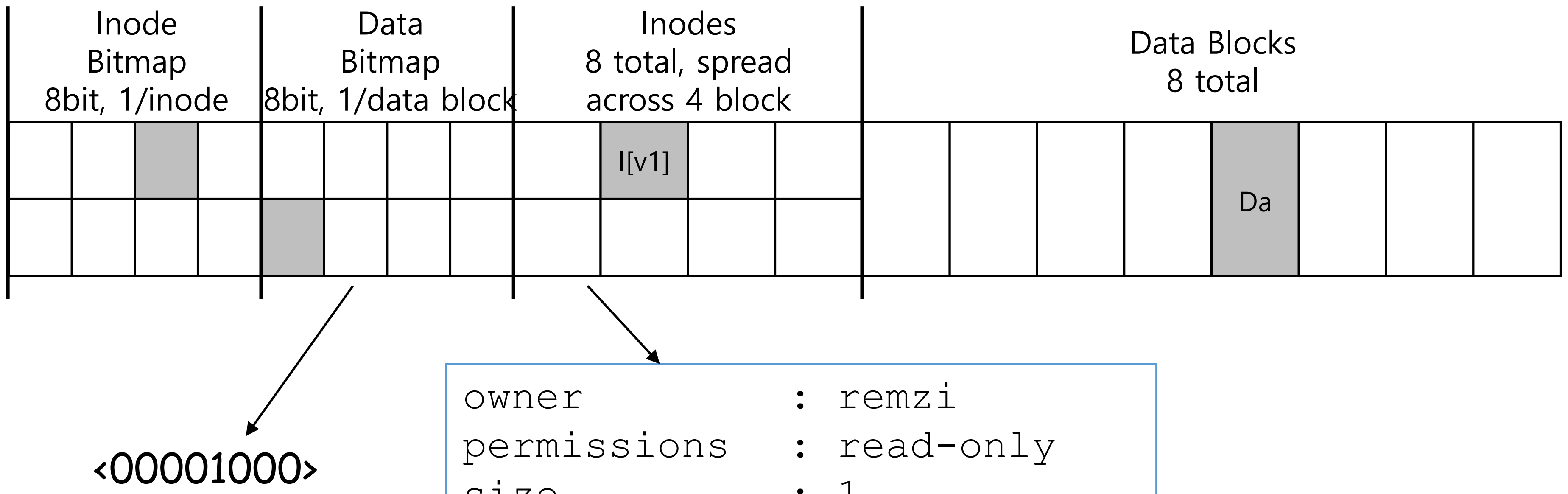
Crash Consistency

- An append to a file involves updating (non atomic)
  - Data blocks
  - Inode
  - Bitmap for data blocks

- What happens if the system crashes in the middle of updating the on-disk structures?
  - Crash due to: power loss or kernel panic or reboot

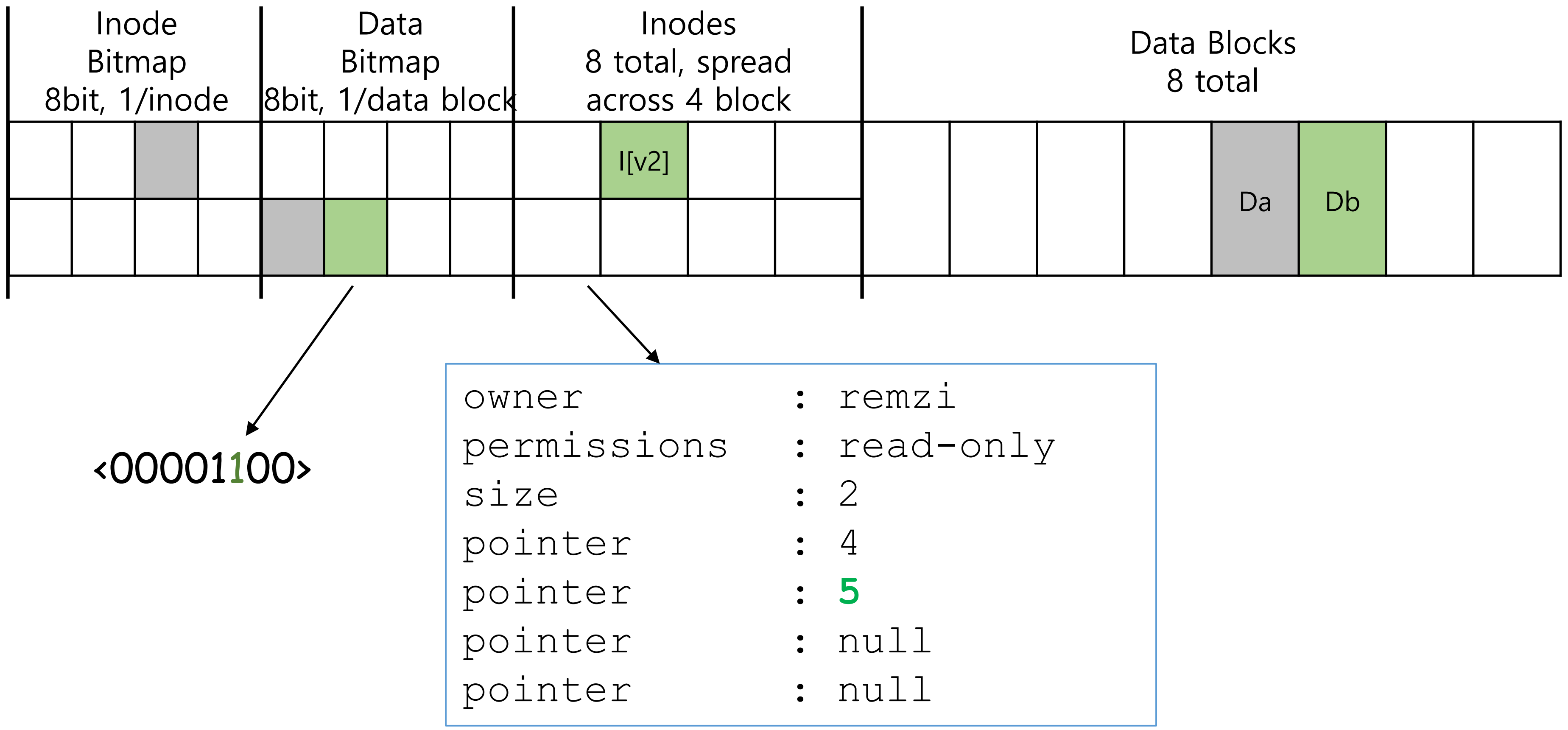
- The file system can be in an inconsistent state

Example



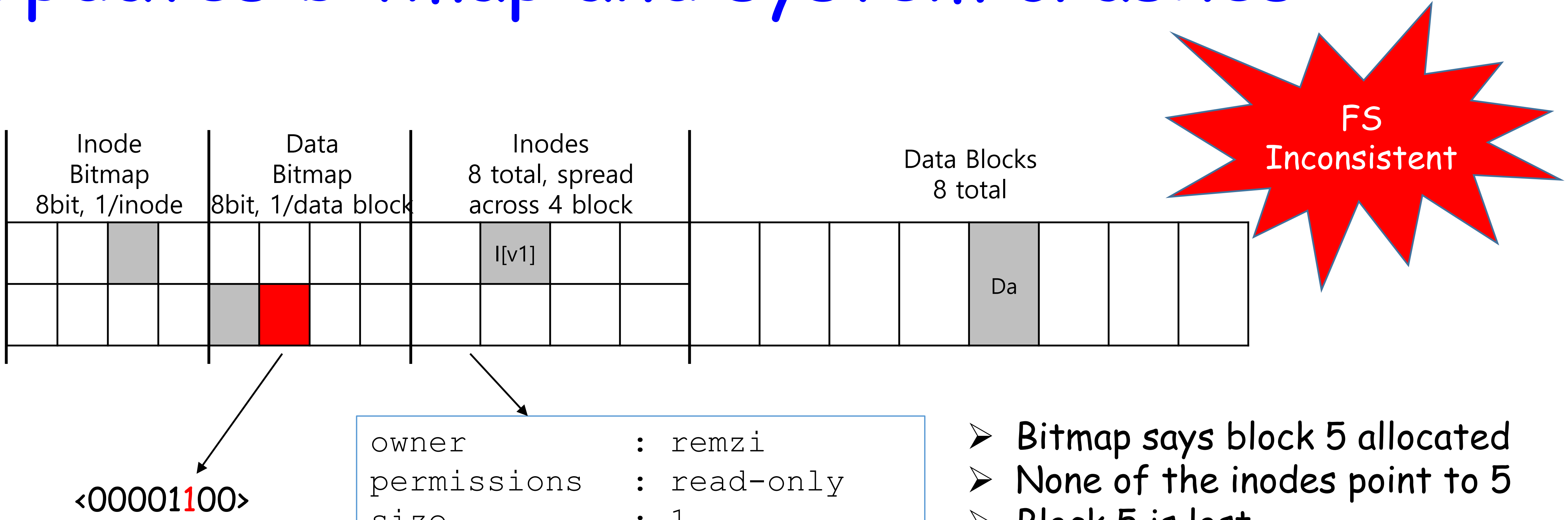
```
owner          : remzi
permissions    : read-only
size           : 1
pointer        : 4
pointer        : null
pointer        : null
pointer        : null
```

Example: Append a new block of data


Inode Bitmap 8bit, 1/inode
Data Bitmap 8bit, 1/data block
Inodes 8 total, spread across 4 block
Data Blocks 8 total
I[v2]
Da
Db
<00001100>

```
owner          : remzi
permissions    : read-only
size           : 2
pointer        : 4
pointer        : 5
pointer        : null
pointer        : null
```

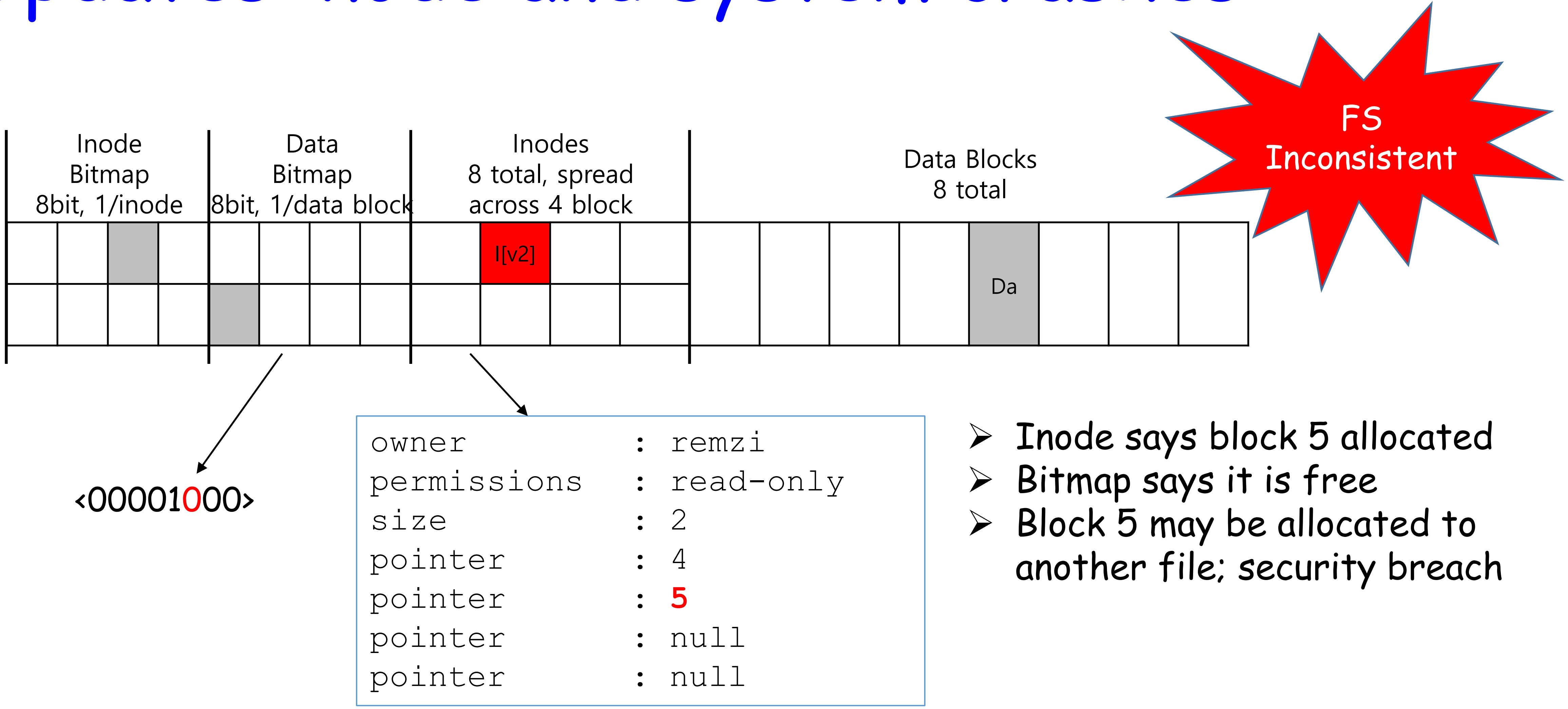
Updates bitmap and system crashes


FS Inconsistent
Inode Bitmap 8bit, 1/inode
Data Bitmap 8bit, 1/data block
Inodes 8 total, spread across 4 block
Data Blocks 8 total
I[v1]
Da
<00001100>

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

- Bitmap says block 5 allocated
- None of the inodes point to 5
- Block 5 is lost

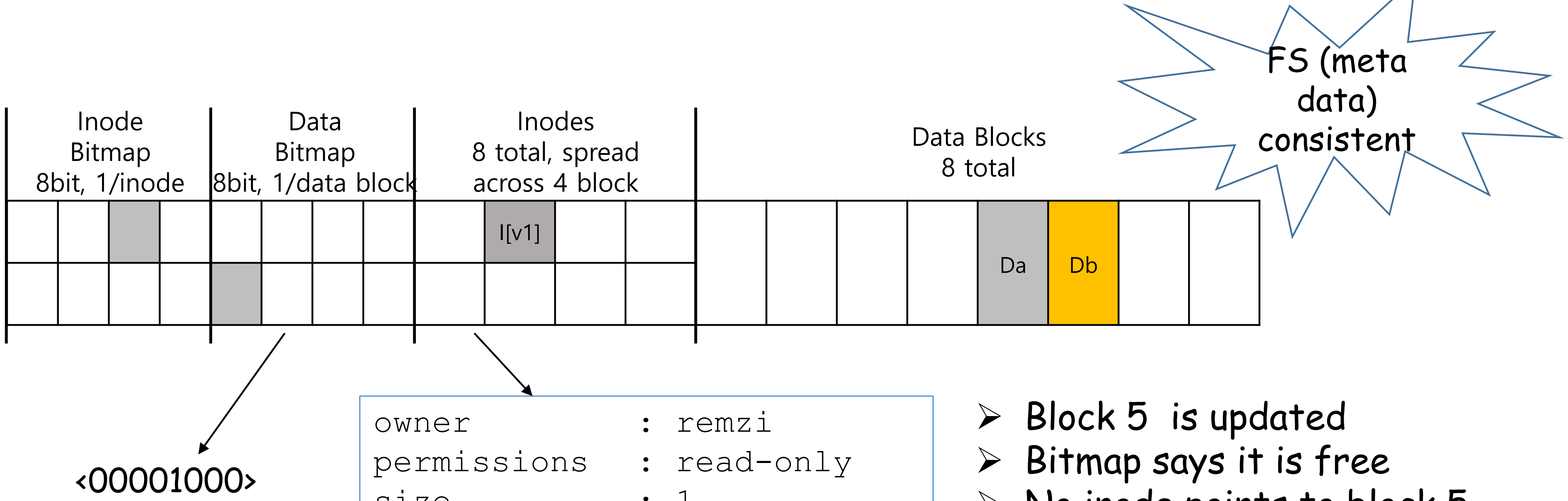
Updates inode and system crashes


FS Inconsistent
Inode Bitmap 8bit, 1/inode
Data Bitmap 8bit, 1/data block
Inodes 8 total, spread across 4 block
Data Blocks 8 total
I[v2]
Da
<00001000>

```
owner          : remzi
permissions    : read-only
size           : 2
pointer        : 4
pointer        : 5
pointer        : null
pointer        : null
```

- Inode says block 5 allocated
- Bitmap says it is free
- Block 5 may be allocated to another file; security breach

Updates data block and system crashes


FS (meta data) consistent
Inode Bitmap 8bit, 1/inode
Data Bitmap 8bit, 1/data block
Inodes 8 total, spread across 4 block
I[v1]
Data Blocks 8 total
Da
Db
<00001000>

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

- Block 5 is updated
- Bitmap says it is free
- No inode points to block 5
- Data is lost

Three more crash scenarios

- Inode and bitmap updated, but not data block
  - ➢Meta data consistent
  - ➢But stale data in data block
- Inode and data block updated but not bitmap
  - ➢FS inconsistent
  - ➢Data block may be allocated to another file
  - ➢Similar to update inode only
- Bitmap and data block updated not inode
  - ➢FS inconsistent
  - ➢Data block lost
  - ➢Similar to update bitmap only

Crash Consistency Problem

- Transition the FS from one state to another state atomically

- Not possible
  - Disk commits one write at a time
  - Crash in the middle of update

- Need solutions to resolve inconsistencies after system recovers from crash

File System Checker (FSCK)

- After crash, scan the entire disk for inconsistencies and resolve them

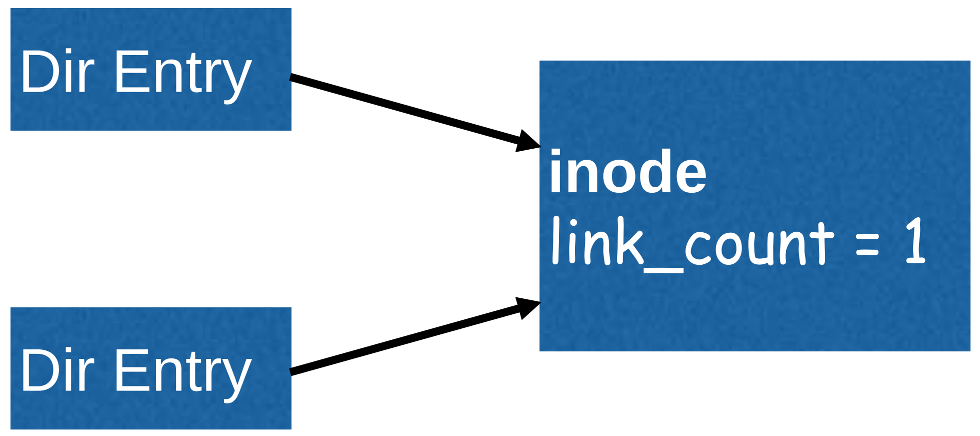
- How to check if bitmap data is consistent?
  - Read every valid inode's direct + indirect block pointers
  - If pointer to data block, the corresponding bit should be 1; else bit is 0

FSCK Checks

- Hundreds of types of checks over different fields…
- Do superblocks match?
- Do directories contain "." and ".."?
- Do number of dir entries equal inode link counts?
- Do different inodes ever point to same block?
- …

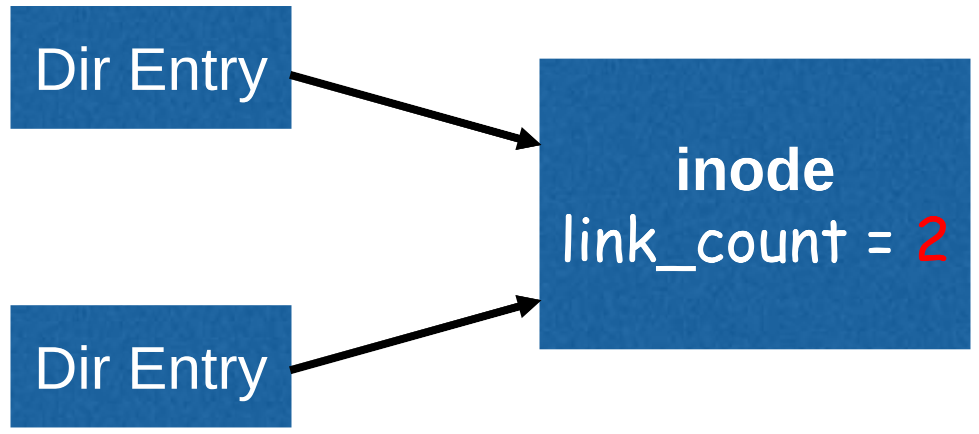
- How to fix inconsistencies?

Link Count (example 1)



How to fix to have consistent file system?

Link Count (example 1)



How to fix to have consistent file system?
Simple fix. Change link_count = 2

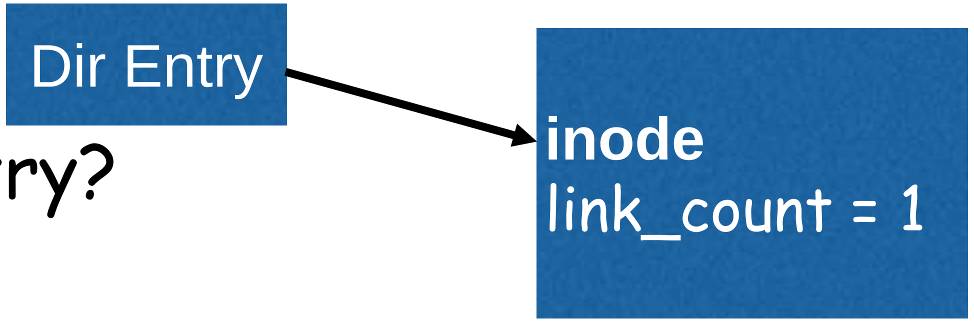
Link Count (example 2)

inode
link_count = 1

No dir entry. How to fix?

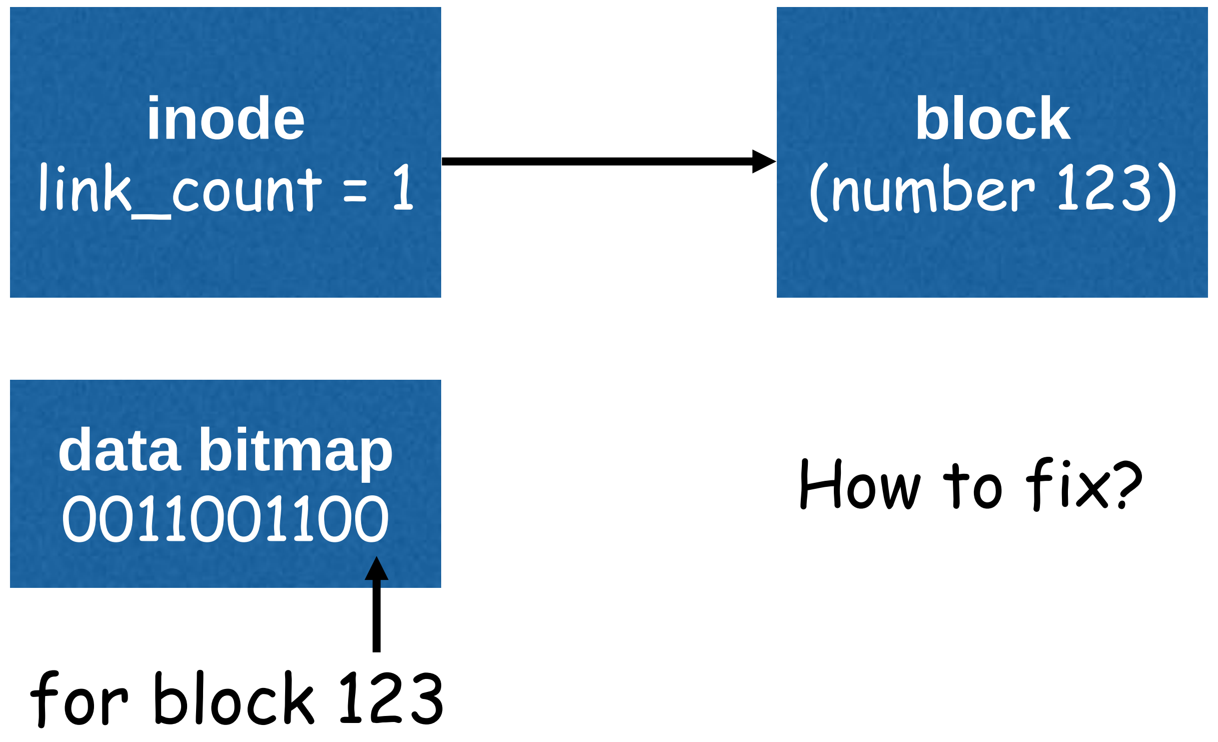
Link Count (example 2)

Create dir entry.
Where to put the entry?



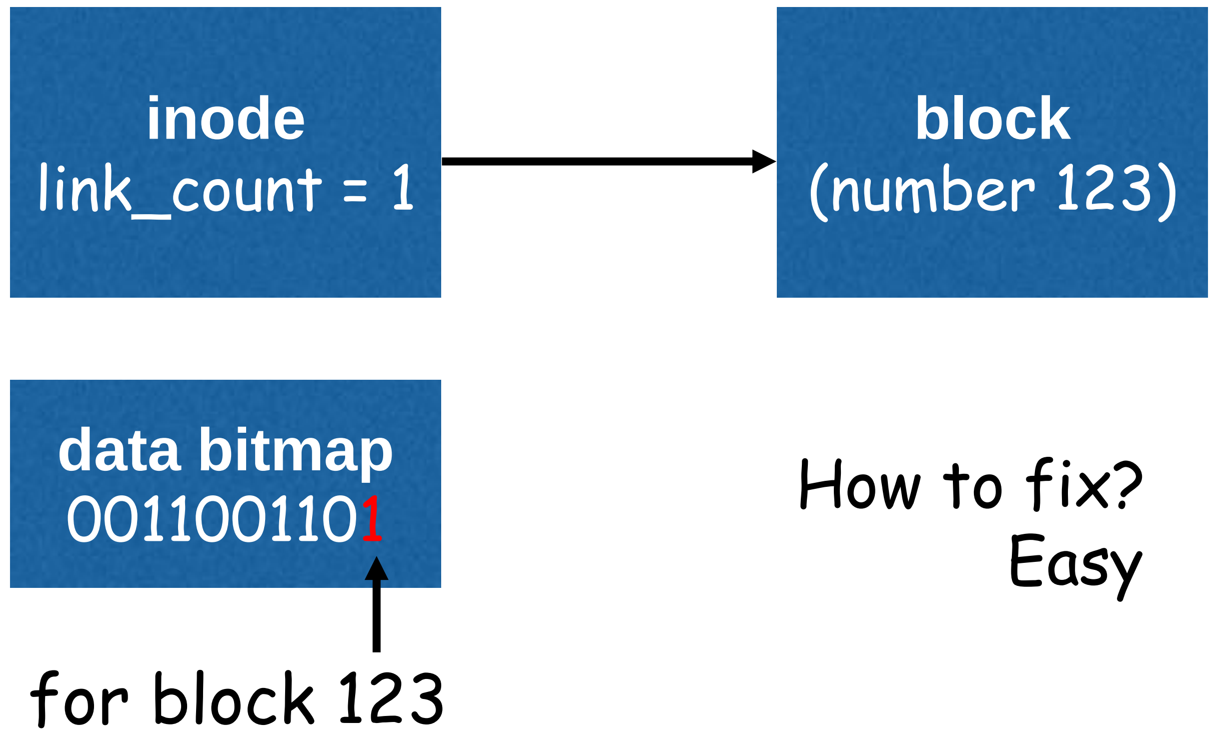
```
ls -l /
total 150
drwxr-xr-x  401 18432 Dec 31  1969 afs/
drwxr-xr-x.   2 4096  Nov  3 09:42 bin/
drwxr-xr-x.   5 4096  Aug  1 14:21 boot/
dr-xr-xr-x.  13 4096  Nov  3 09:41 lib/
dr-xr-xr-x.  10 12288 Nov  3 09:41 lib64/
drwx------.   2 16384 Aug  1 10:57 lost+found/
...
```

Data Bitmap



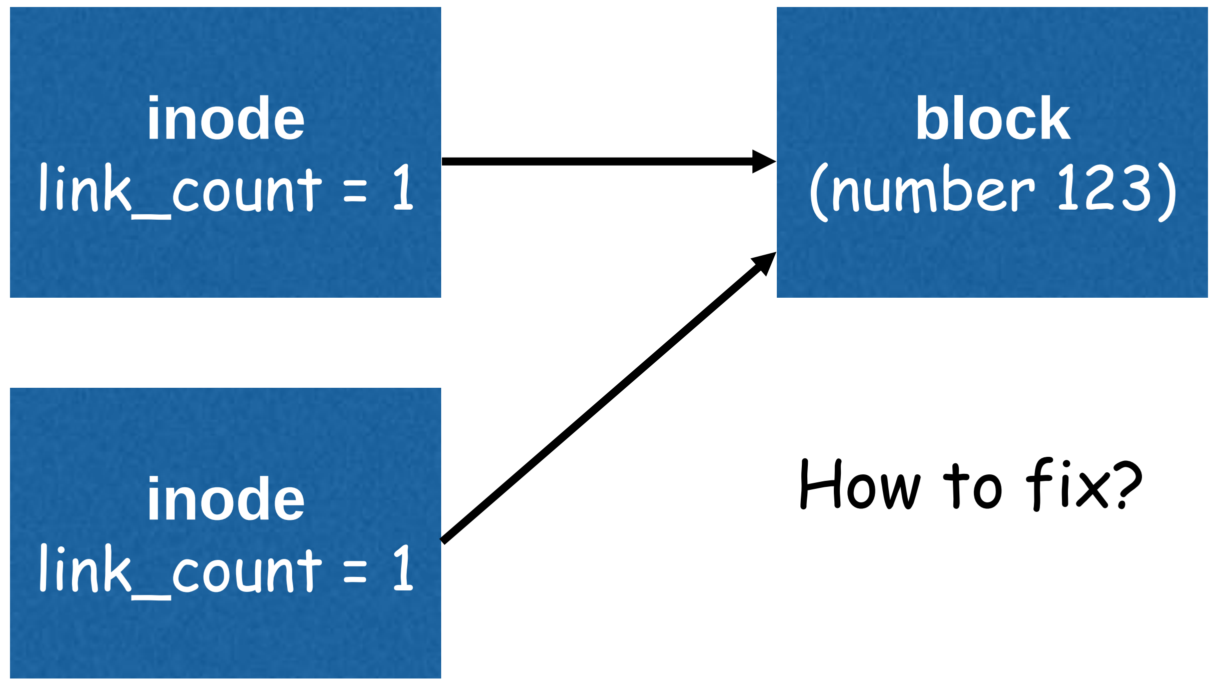
How to fix?

Data Bitmap



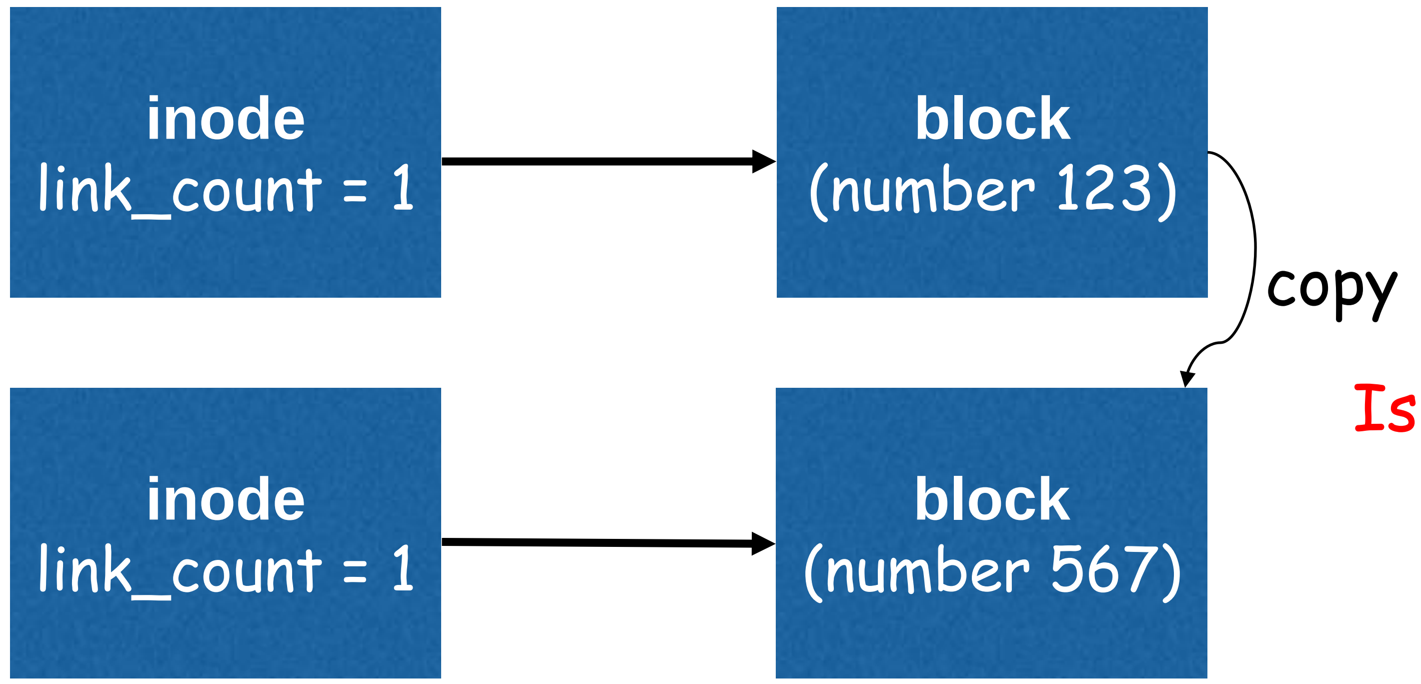
How to fix?
Easy

Duplicate Pointers



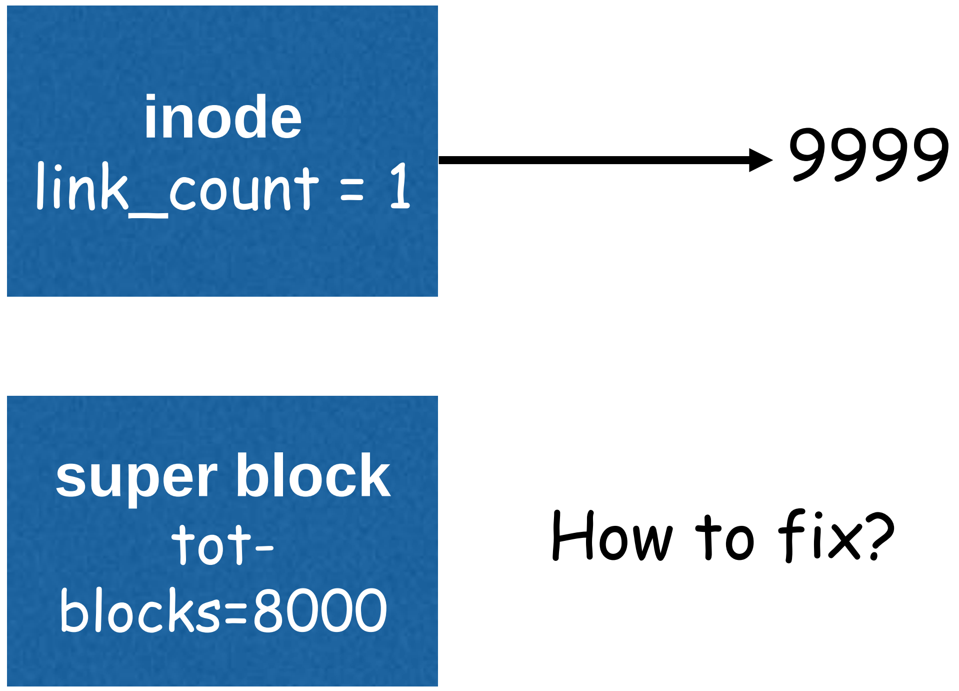
How to fix?

Duplicate Pointers



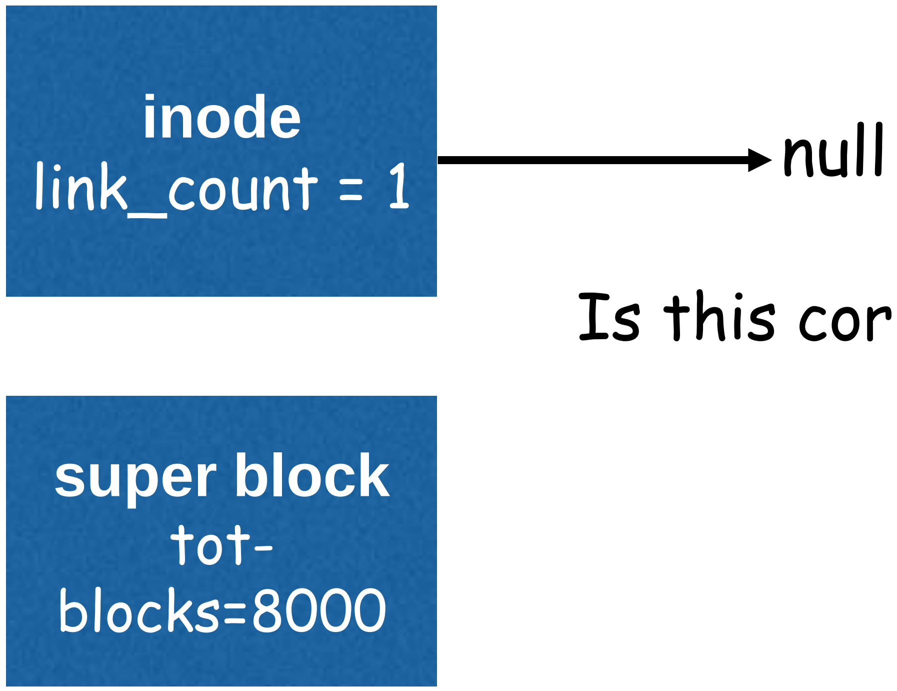
Is this correct?

Bad Pointer



How to fix?

Bad Pointer

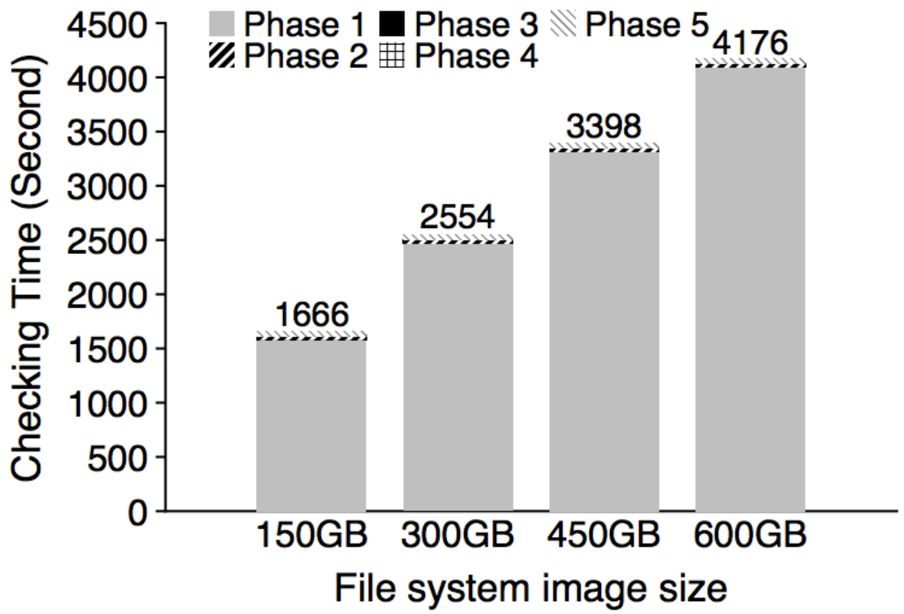


Is this correct?

Problems with fsck

- Not always obvious how to fix file system image
- Don't know "correct" state, just consistent one
- Easy way to get consistency: reformat disk!

Bigger Problem: fsck is very slow



Checking a 600GB disk takes ~70 minutes

ffsck: The Fast File System Checker

Ao Ma, EMC Corporation and University of Wisconsin—Madison; Chris Dragga,
Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau, University of Wisconsin—Madison

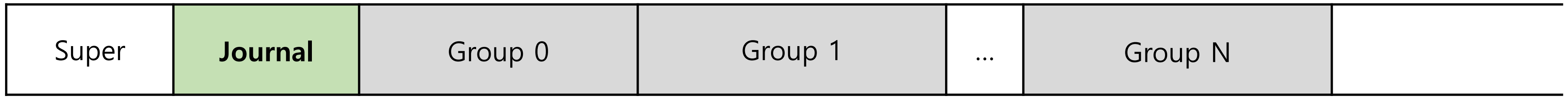
Better Solution Goals

- Ok to do some recovery work after crash, but not to read entire disk
- Don't move file system to just any consistent state, get to correct state
- Journaling meet the goals

Journaling

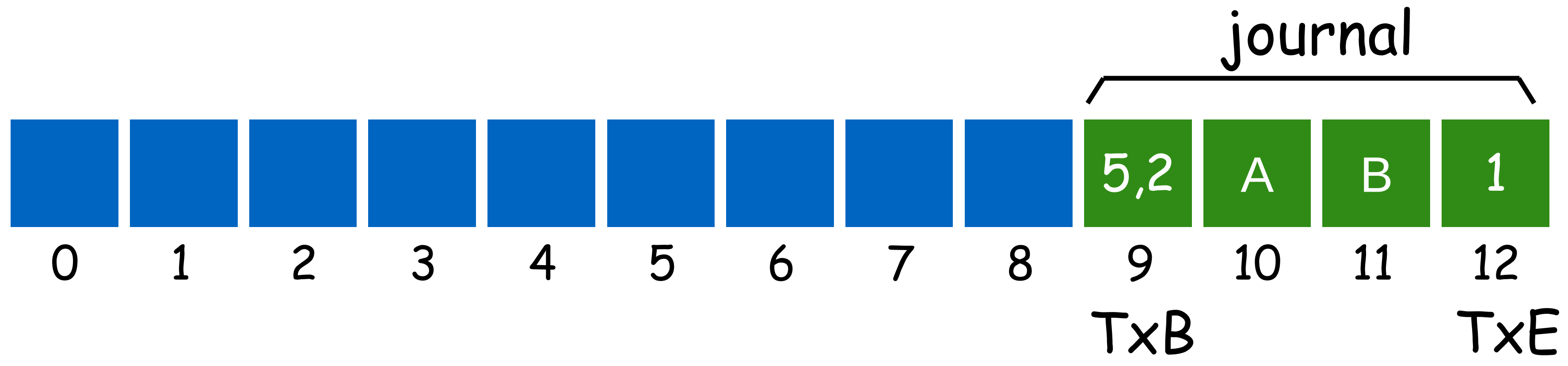
- Prepare a log with details of the update operations
- Write the log before the update operations on disk
  - Write-ahead-logging (on disk)
- If there is a crash during updates, on recovery, read the log and perform the updates
- What happens if system crashes while logging?

FS Structure with Journaling



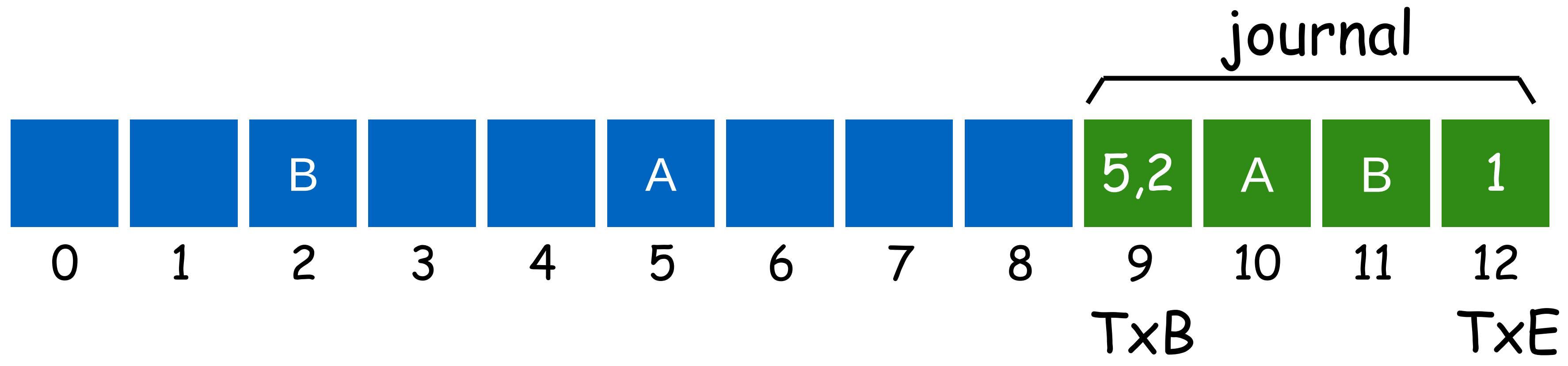
EXT3 FS structure

Journaling Example: Logging



transaction: write A to block 5; write B to block 2

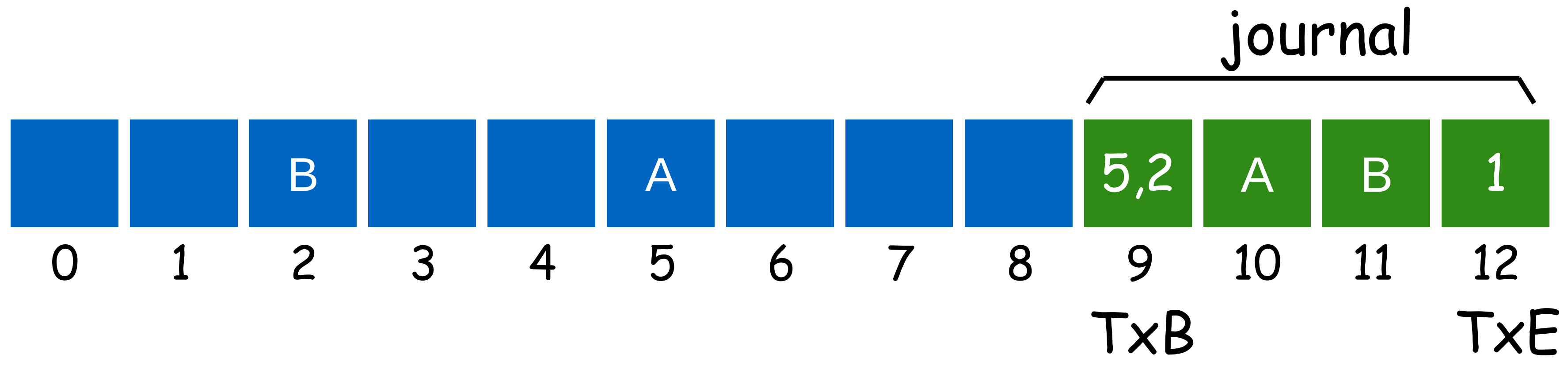
Journaling Example: Checkpointing



transaction: write A to block 5; write B to block 2

Checkpoint: Writing new data to in-place locations

Total Ordering



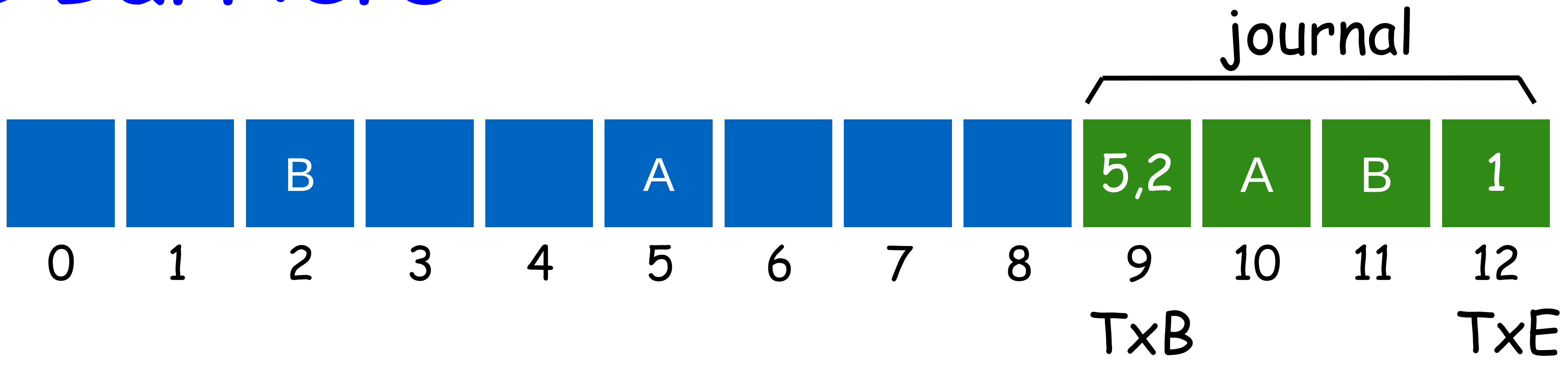
transaction: write A to block 5; write B to block 2

write order: 9, 10, 11, 12, 5, 2

Enforcing total ordering is inefficient. Why?

Random writes

Use Barriers



transaction: write A to block 5; write B to block 2

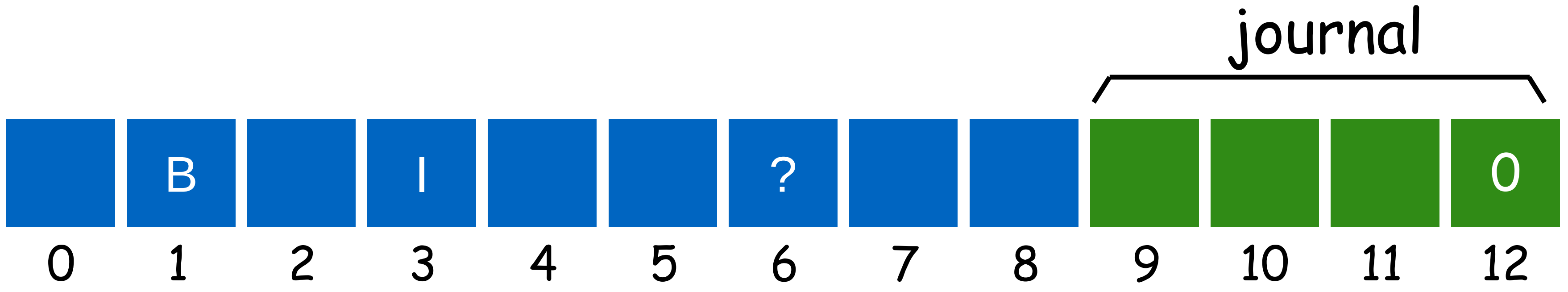
write order: 9, 10, 11 | 12 | 5, 2 |

Use barriers at key points in time:
1) Before journal commit, ensure journal transaction entries complete
2) Before checkpoint, ensure journal commit complete
3) Before free journal, ensure in-place updates complete

How to avoid writing all blocks twice?

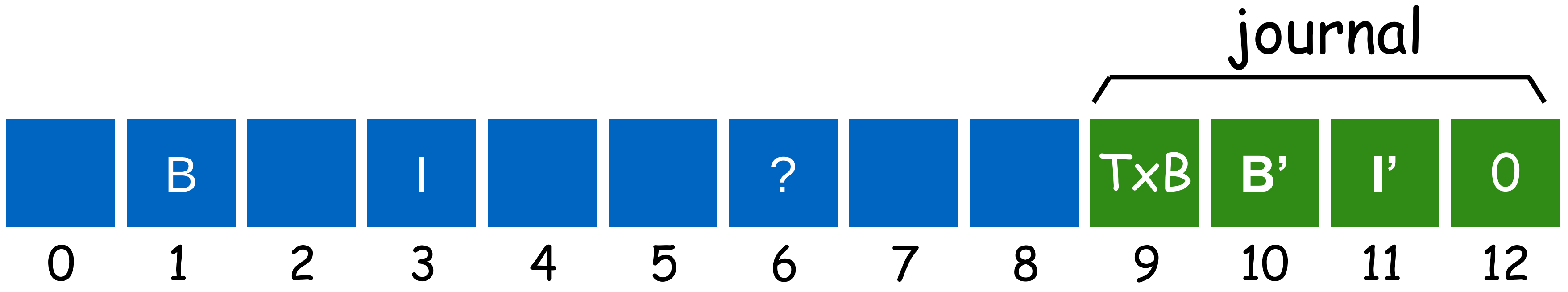
- Observation: some blocks (e.g., user data) are less important
- Strategy: journal all metadata, including:
  superblock, bitmaps, inodes, indirects, directories
- For regular data, write it back whenever convenient.
  Of course, files may contain garbage.

Writeback Journal



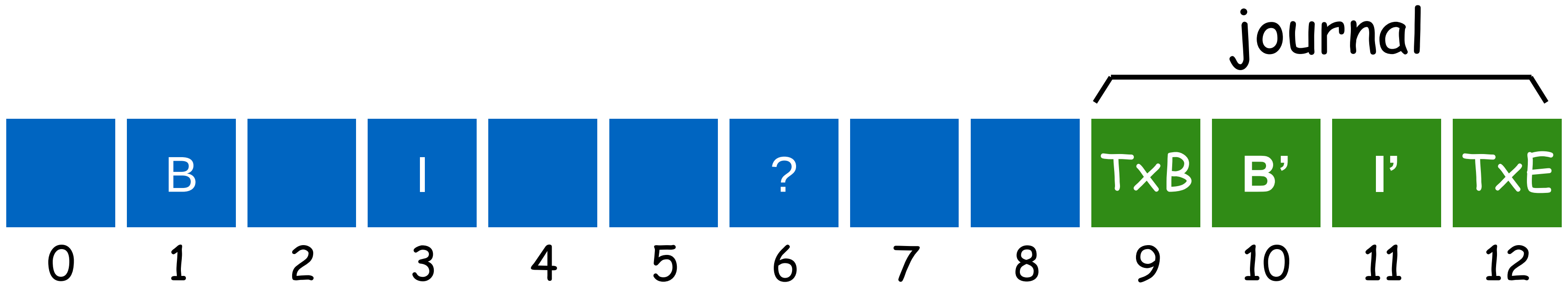
transaction: append to inode I

Writeback Journal



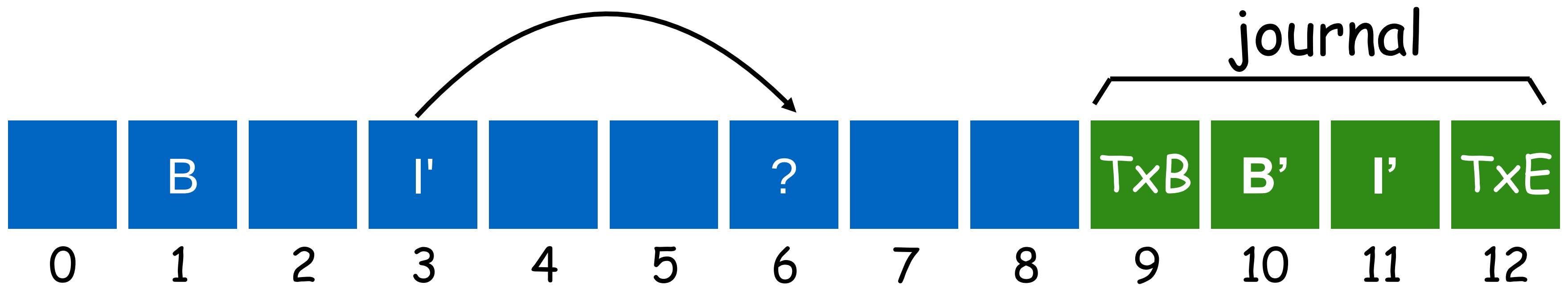
transaction: append to inode I

Writeback Journal



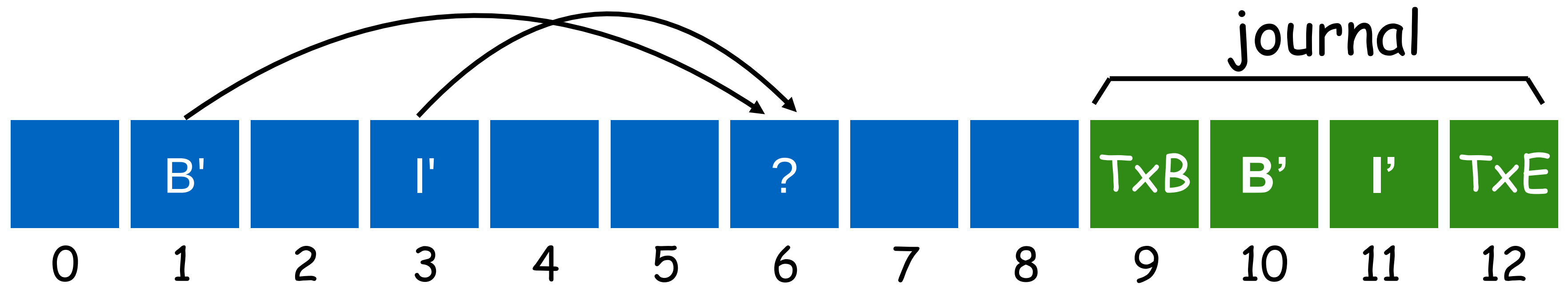
transaction: append to inode I

Writeback Journal



transaction: append to inode I

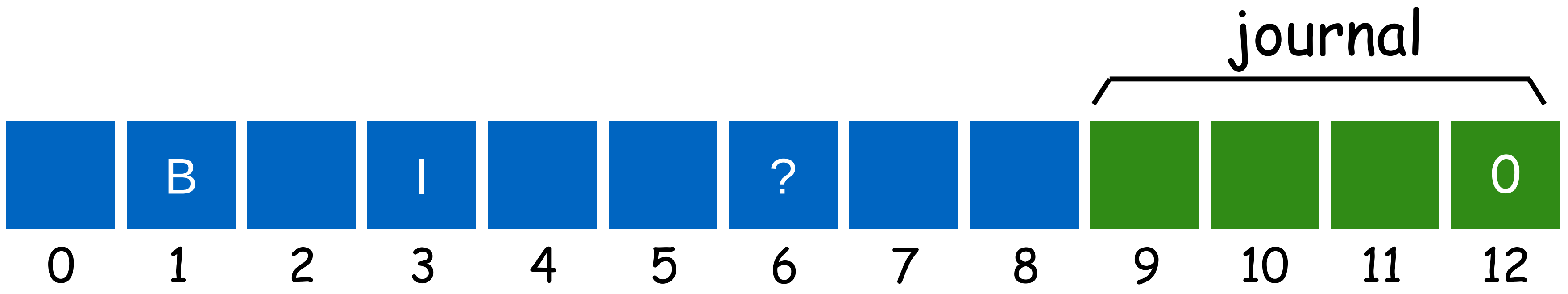
Writeback Journal



transaction: append to inode I

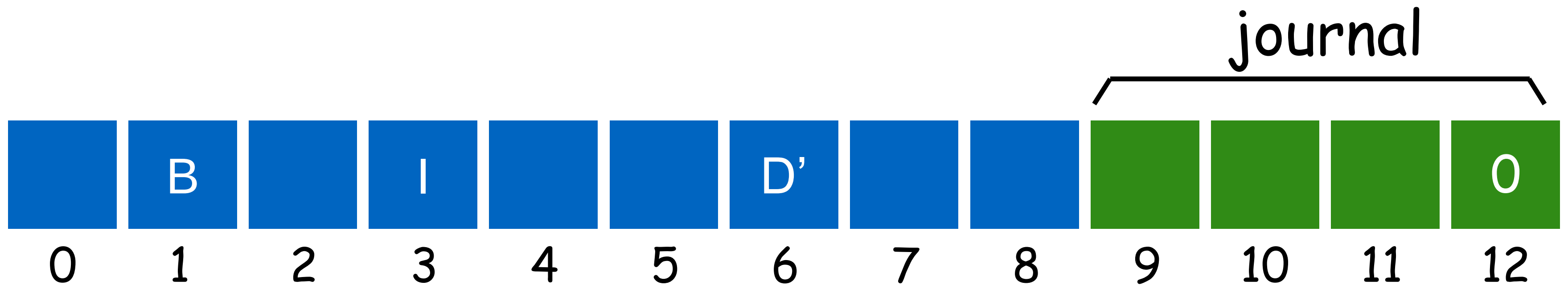
what if we crash now? Solutions?

Ordered Journaling



transaction: append to inode I

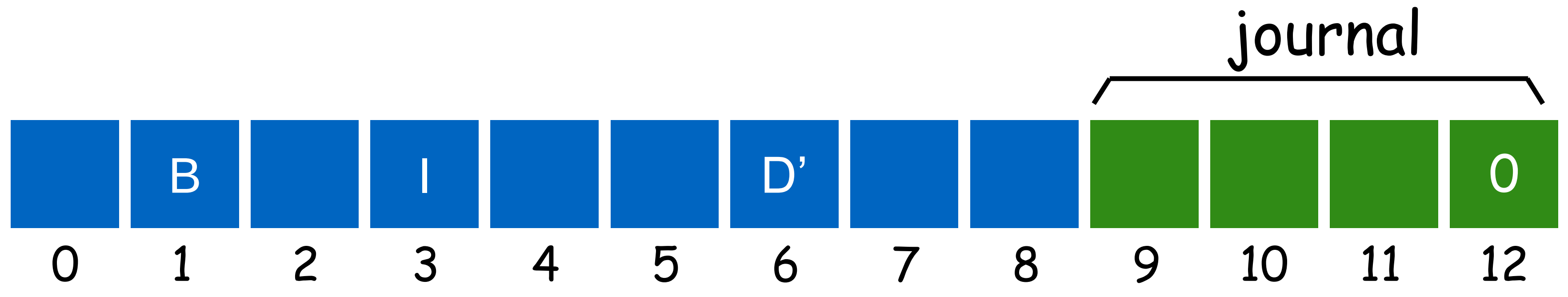
Ordered Journaling



transaction: append to inode I

Write data before journaling

Ordered Journaling

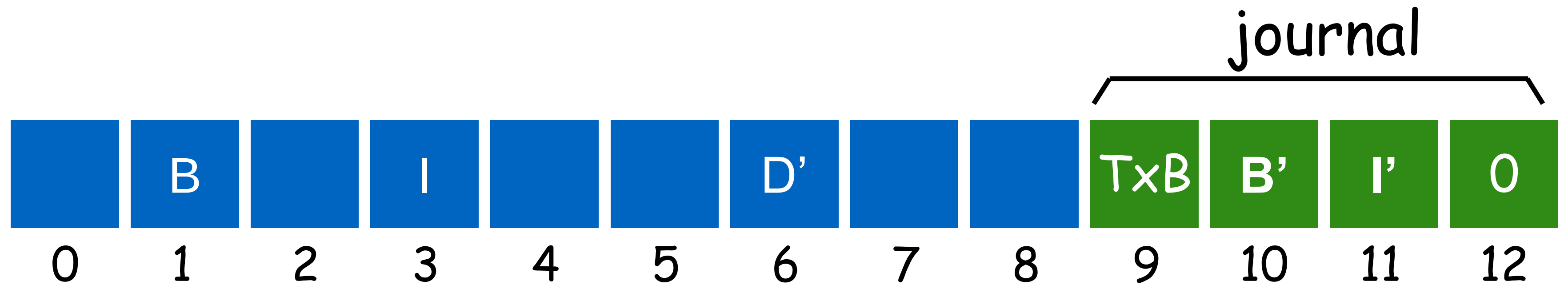


transaction: append to inode I

Write data before journaling

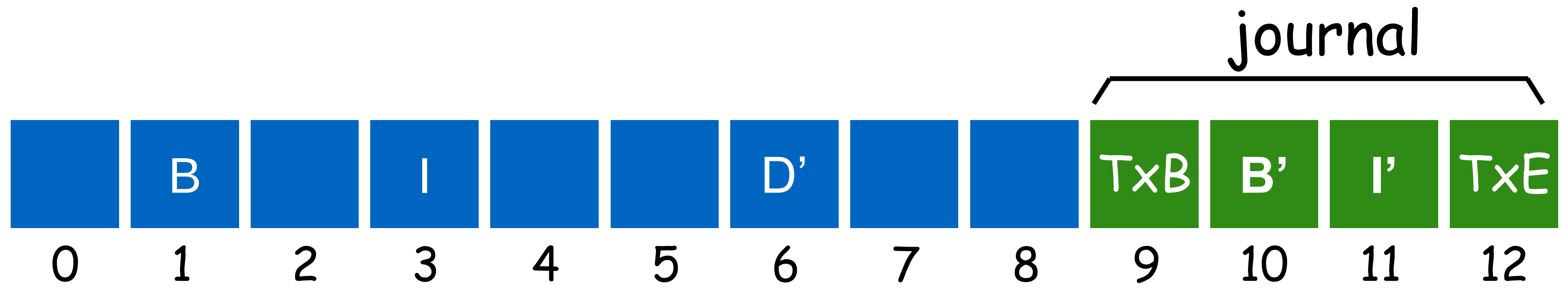
What happens if crash now?
B indicates D' currently free, I does not point to D';
Lose D', but that might be acceptable

Ordered Journal



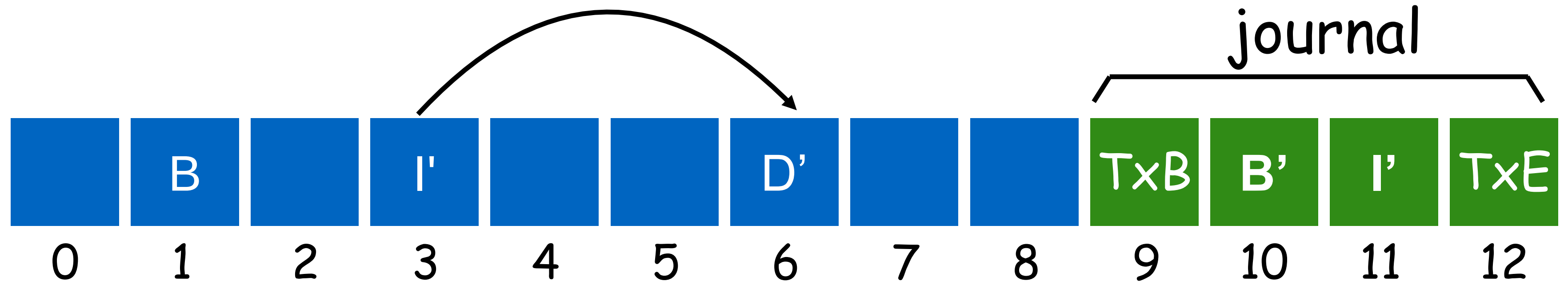
transaction: append to inode I

Ordered Journal



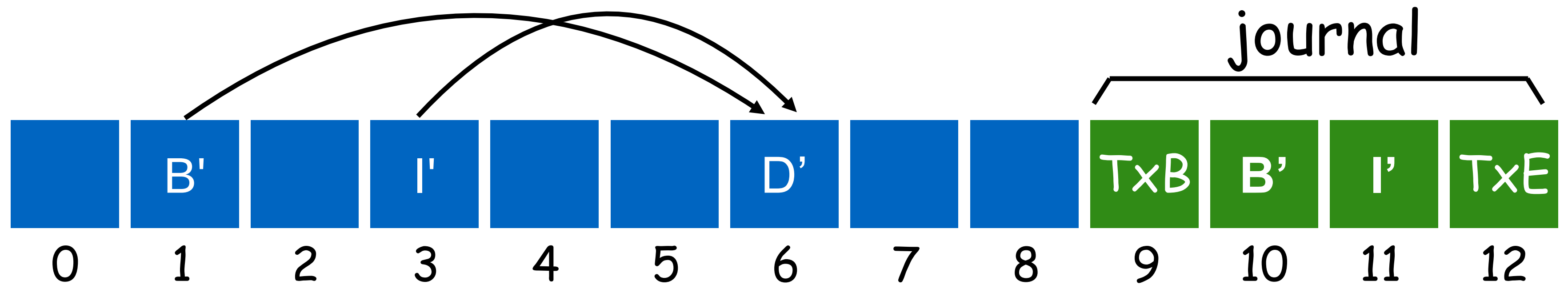
transaction: append to inode I

Ordered Journal



transaction: append to inode I

Ordered Journal



transaction: append to inode I

Which one to choose?

- No need to choose one
- Provide all option to the end users.
  - Let them pick it
- Most modern file systems use journals
  - ordered-mode for meta-data is popular

Disclaimer

- Some of the materials in this lecture slides are from the lecture slides by Prof. Arpaci, Prof. Youjip, and other educators. Thanks to all of them.