PERSISTENCE: FSCK, JOURNALING

Shivaram Venkataraman

CS 537, Fall 2024

ADMINISTRIVIA

Project 5 updates

Midterm 2: Solutions, grades

Next week's schedule

AGENDA / LEARNING OUTCOMES

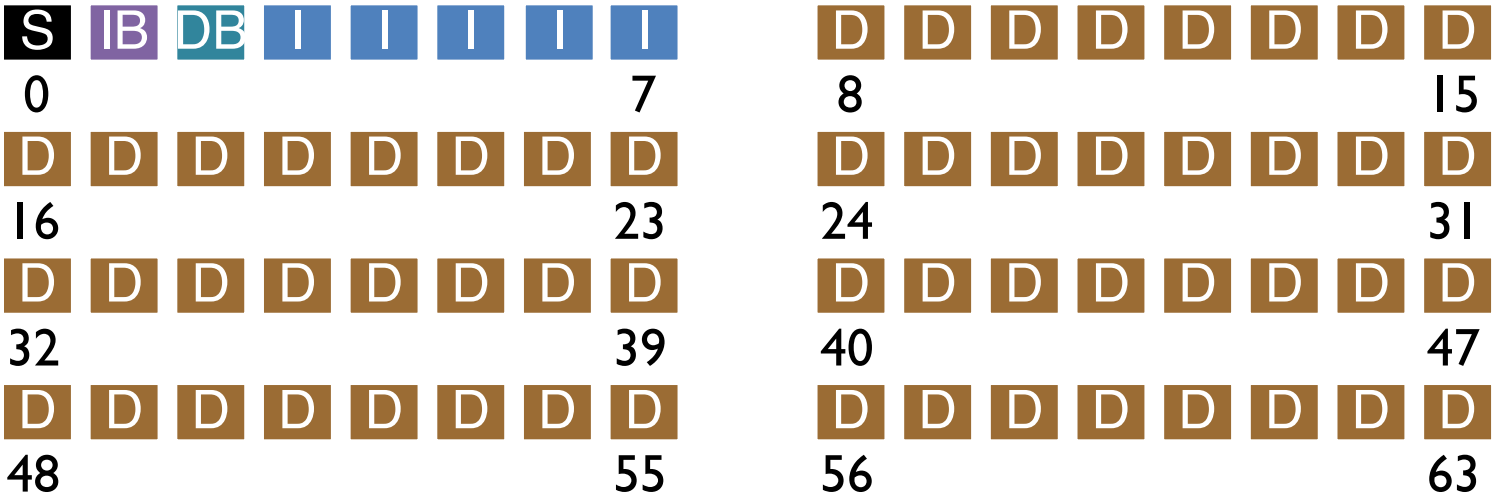
How to check for consistency with power failures / crashes?

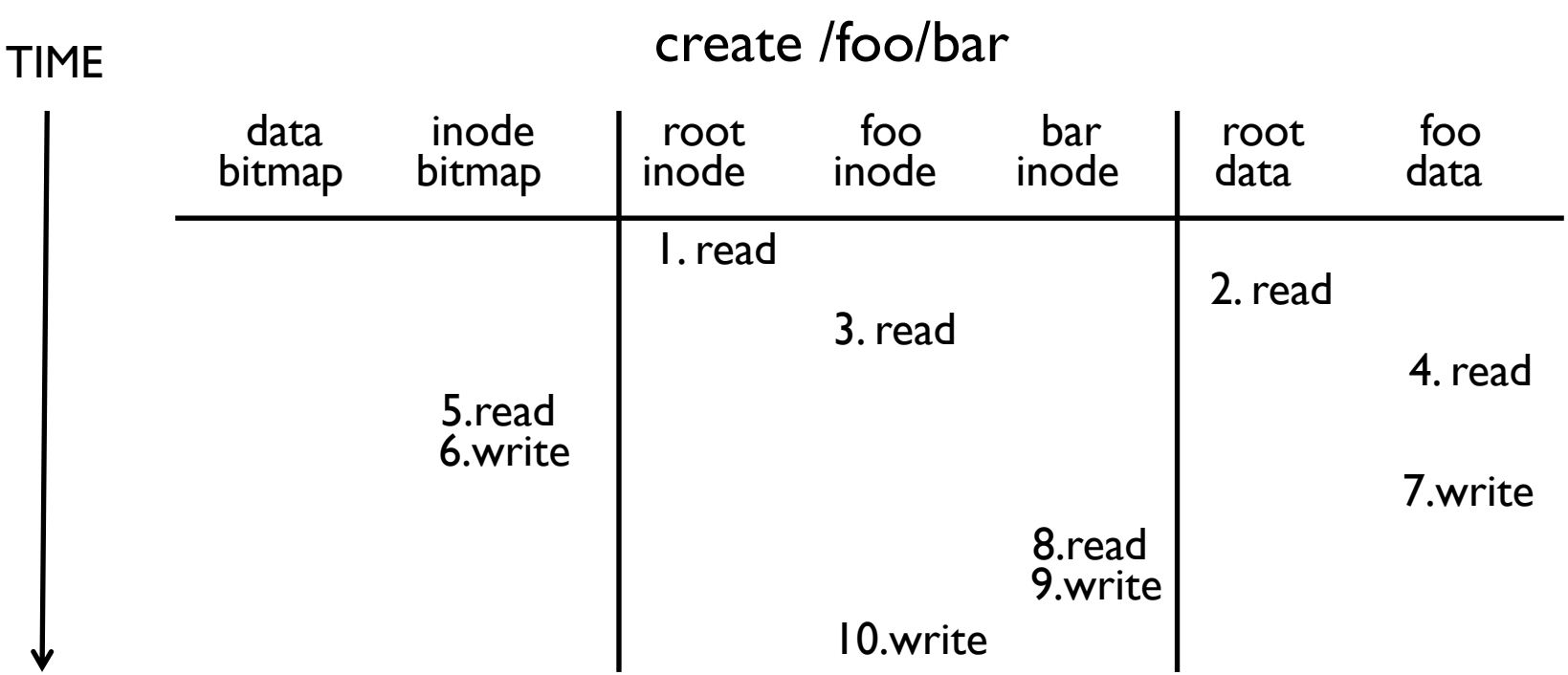
How to ensure consistency in filesystem design?

RECAP

FS STRUCTS: SUPERBLOCK

Basic FS configuration metadata, like block size, # of inodes




create /foo/bar
TIME
data bitmap
inode bitmap
root inode
foo inode
bar inode
root data
foo data
1. read
2. read
3. read
4. read
5.read
6.write
7.write
8.read
9.write
10.write

FFS PLACEMENT GROUPS

Key idea: Keep inode close to data

Use groups across disks;

Strategy: allocate inodes and data blocks in same group.

FFS STRATEGY

Put more-related pieces of data near each other

Put less-related pieces of data **far**

```
/a/b
/a/c
/a/d
/b/f
```

```
group inodes     data
    0 /--------- /---------
    1 acde------ accddee---
    2 bf-------- bff-------
    3 ---------- ----------
    4 ---------- ----------
    5 ---------- ----------
    6 ---------- ----------
    7 ---------- ----------
    ...
```

PROBLEM: LARGE FILES

Single large file can fill nearly all of a group

Displaces data for many small files

```
group inodes     data
    0 /a-------- /aaaaaaaaa aaaaaaaaaa aaaaaaaaaa a---------
    1 ---------- ---------- ---------- ---------- ----------
    2 ---------- ---------- ---------- ---------- ----------
    ...
```

Most files are small!
Better to do one seek for large file than
one seek for each of many small files

SPLITTING LARGE FILES

```
group inodes     data
    0 /a-------- /aaaaa---- ---------- ---------- ----------
    1 ---------- aaaaa----- ---------- ---------- ----------
    2 ---------- aaaaa----- ---------- ---------- ----------
    3 ---------- aaaaa----- ---------- ---------- ----------
    4 ---------- aaaaa----- ---------- ---------- ----------
    5 ---------- aaaaa----- ---------- ---------- ----------
    6 ---------- ---------- ---------- ---------- ----------
    ...
```

Define "large" as requiring an indirect block

Starting at indirect (e.g., after 48 KB) put blocks in a new block group.

Each chunk corresponds to one indirect block

Block size 4KB, 4 byte per address => 1024 address per indirect

1024*4KB = 4MB contiguous "chunk"

POLICY SUMMARY

File inodes: allocate in same group with dir

Dir inodes: allocate in new group with fewer used inodes than average group

First data block: allocate near inode

Other data blocks: allocate near previous block

Large file data blocks: after 48KB, go to new group.

Move to another group (w/ fewer than avg blocks) every subsequent 4MB.

OTHER FFS FEATURES

FFS also introduced several new features:

- large blocks (with libc buffering / fragments)
- long file names
- atomic rename
- symbolic links

Inspired modern files systems, including ext2 and ext3

FILE SYSTEM CONSISTENCY

FILE SYSTEM CONSISTENCY EXAMPLE

Superblock: field contains total number of blocks in FS

DATA = N

Inode: field contains pointer to data block; possible DATA?

DATA in {0, 1, 2, …, N - 1}

Pointers to block N or after are invalid!

Total-blocks field has redundancy with inode pointers

WHY IS CONSISTENCY CHALLENGING?

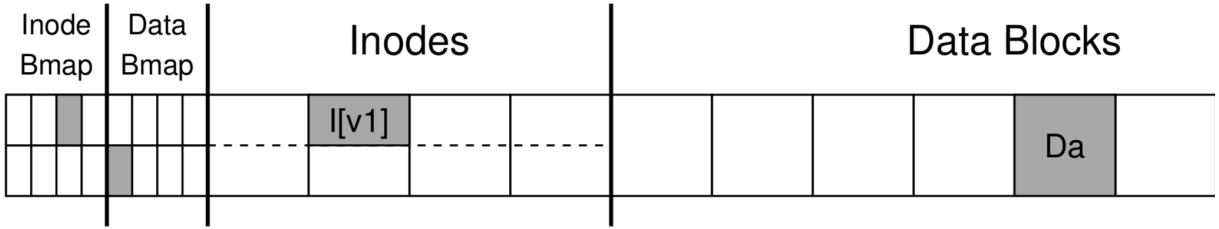
File system may perform several disk writes to redundant blocks

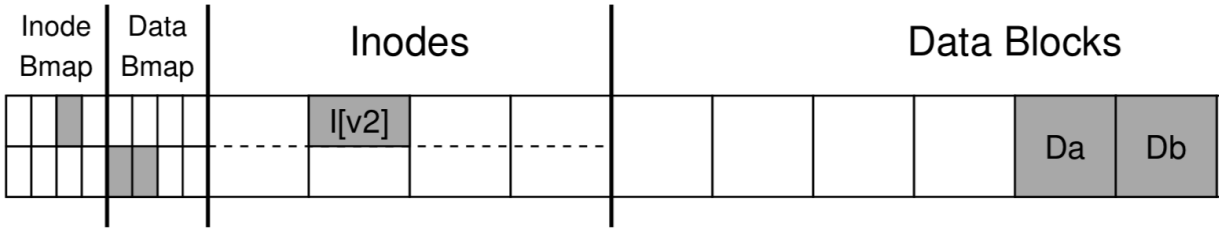
If file system is interrupted between writes, may leave data in inconsistent state

What can interrupt write operations?

- power loss
- kernel panic
- reboot

FILE APPEND EXAMPLE





HOW CAN FILE SYSTEM FIX INCONSISTENCIES?

Solution #1:

FSCK = file system checker

Strategy:

After crash, scan whole disk for contradictions and "fix" if needed

Keep file system off-line until FSCK completes

For example, how to tell if data bitmap block is consistent?

Read every valid inode+indirect block

If pointer to data block, the corresponding bit should be 1; else bit is 0

FSCK CHECKS

Do superblocks match?

Is the list of free blocks correct?

Do number of dir entries equal inode link counts?

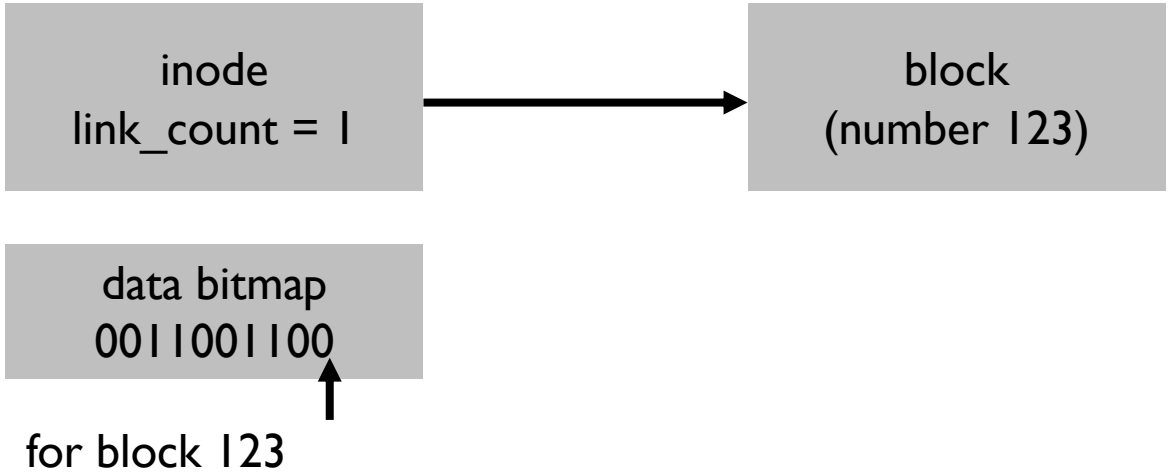
Do different inodes ever point to same block?

Are there any bad block pointers?

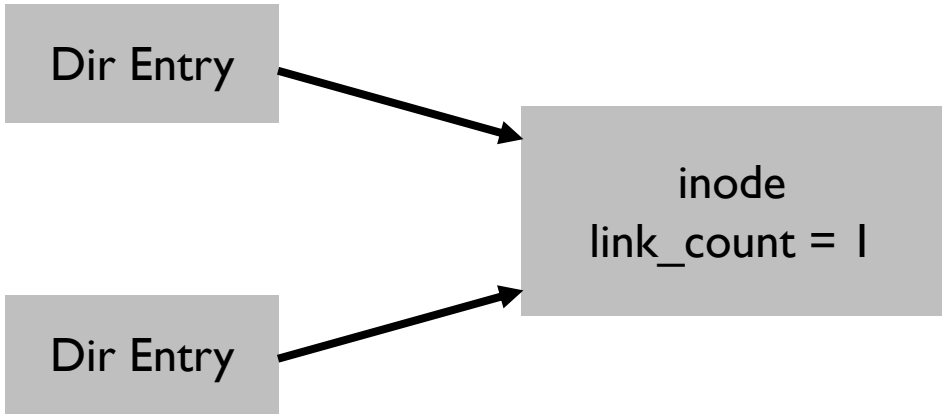
Do directories contain "." and ".."?

…

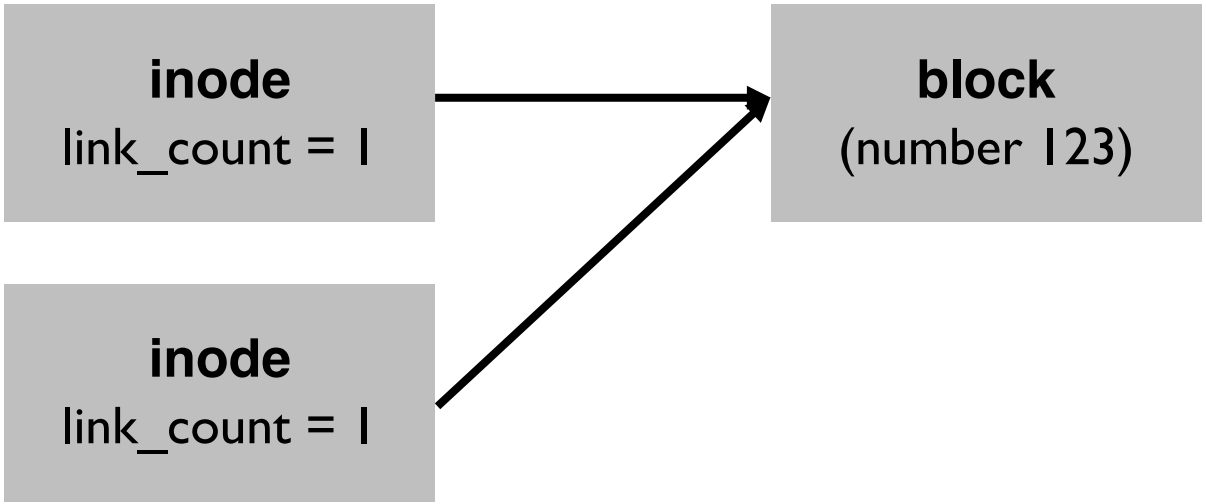
FREE BLOCKS EXAMPLE



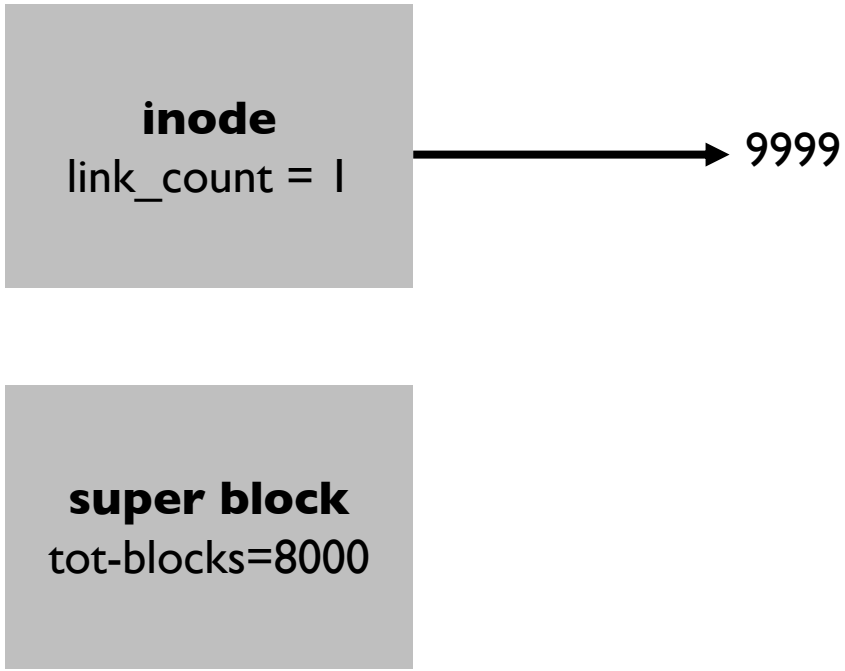
LINK COUNT EXAMPLE



DUPLICATE POINTERS



BAD POINTER



QUIZ 17

Offset for inode with number 0 (in kB)?

Offset for inode with number 4 (in kB)?

```
inode bitmap  10000000
inodes        [d a:0 r:2] [] [] [] [] [] [] []
data bitmap   10000000
data          [(.,0) (..,0)] [] [] [] [] [] [] []
```

```
inode bitmap 11100000
inodes       [d a:0 r:4] [d a:1 r:2] [d a:2 r:2] [] [] [] [] []
data bitmap  11100000
data         [(.,0) (..,0) (d,1) (w,2)] [????] [(.,2) (..,0)] [] [] [] [] []
```

```
inode bitmap  10000000
inodes        [d a:0 r:2] [] [] [] [] [] [] []
data bitmap   10000000
data          [(.,0) (..,0)] [] [] [] [] [] [] []
```

```
inode bitmap  11000000
inodes        [d a:0 r:2] [f a:1 r:2] [] [] [] [] [] []
data bitmap   ????????
data          [(.,0) (..,0) (c,1) (m,1)] [foofoofoo] [] [] [] [] [] []
```

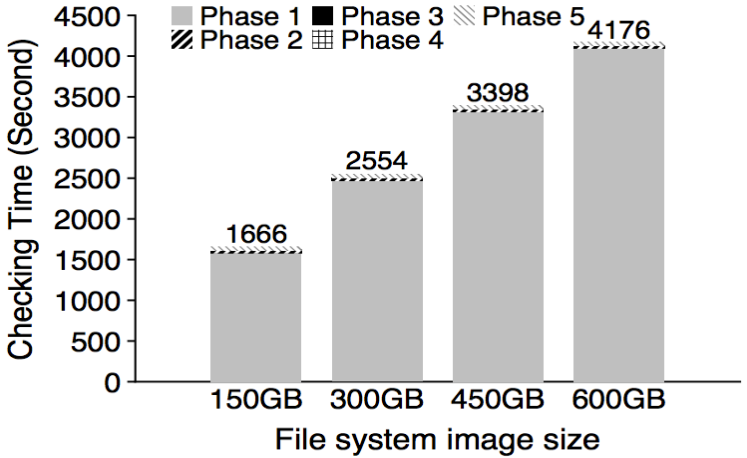
PROBLEMS WITH FSCK

Problem 1:

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

Problem 2:

Checking a 600GB disk takes ~70 minutes



ffsck: The Fast File System Checker
Ao Ma, Chris Dragga, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau

CONSISTENCY SOLUTION #2: JOURNALING

Goals

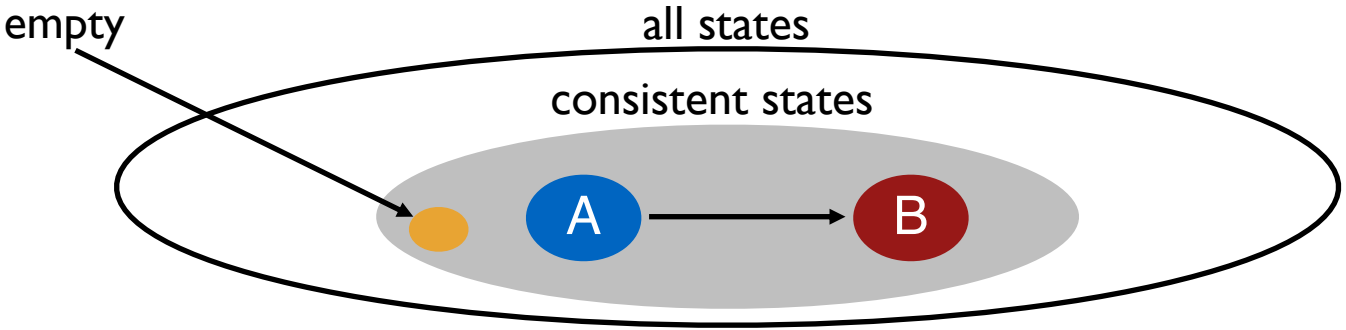
- Ok to do some **recovery work** after crash, but not to read entire disk
- Don't move file system to just any consistent state, get **correct** state

Atomicity

- Definition of atomicity for **concurrency:** operations in critical sections are not interrupted by operations on related critical sections
- Definition of atomicity for **persistence:** collections of writes are not interrupted by crashes; either (all new) or (all old) data is visible

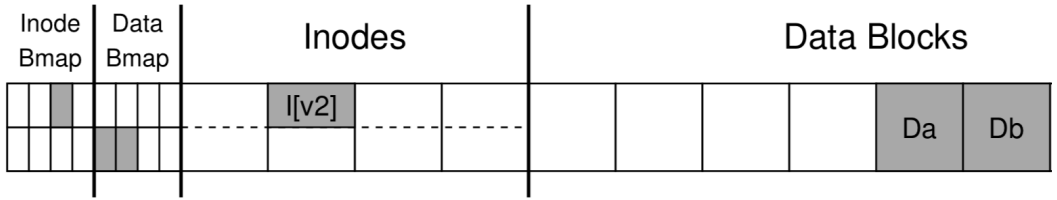
CONSISTENCY VS ATOMICITY

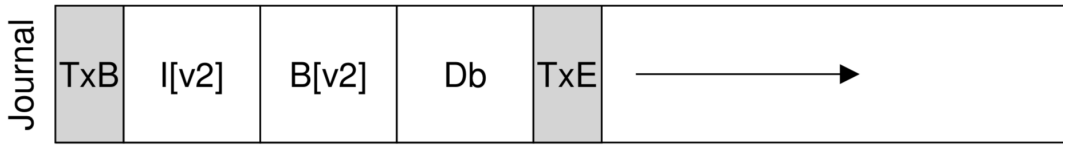
Say a set of writes moves the disk from state A to B



fsck gives consistency
Atomicity gives A or B.

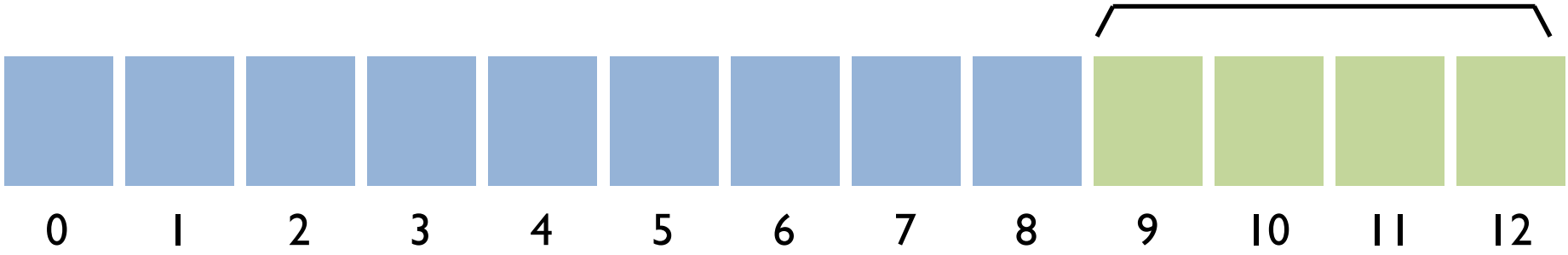
JOURNAL LAYOUT





Transaction

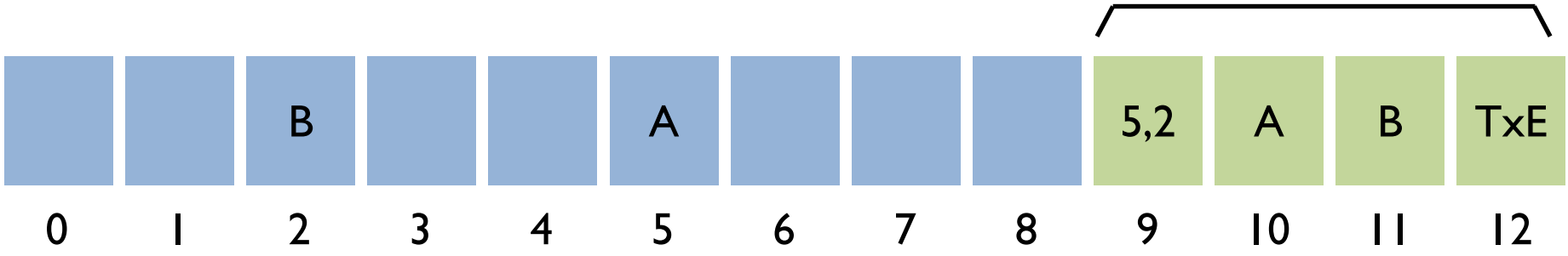
JOURNAL WRITE AND CHECKPOINTS



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

Checkpoint: Writing new data to in-place locations

JOURNAL REUSE AND CHECKPOINTS



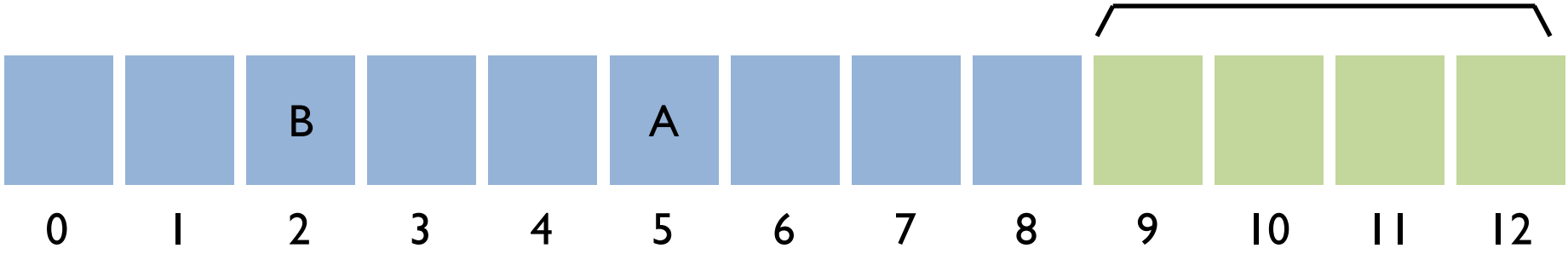
transaction: write A to block 5; write B to block 2

Checkpoint: Writing new data to in-place locations

transaction: write C to block 4; write T to block 6

ORDERING FOR CONSISTENCY

transaction: write C to block 4; write T to block 6



write order

9,10,11

12

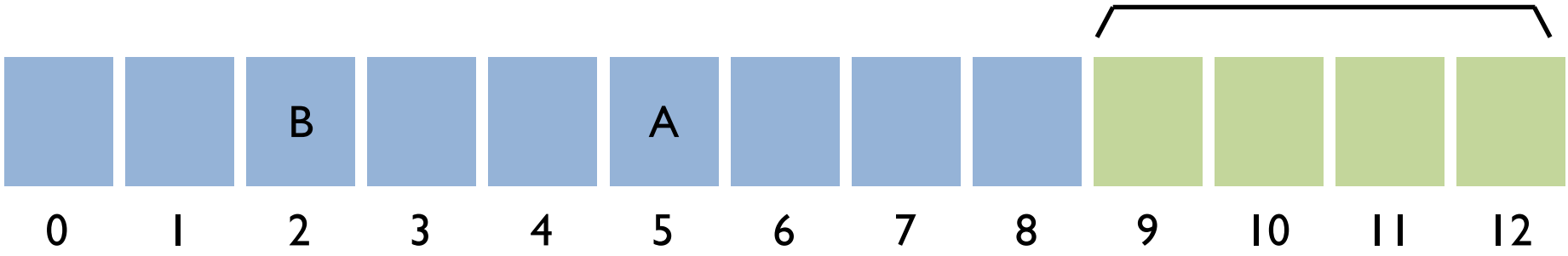
4,6

Barriers

1) Before journal commit, ensure journal entries complete
2) Before checkpoint, ensure journal commit complete
3) Before free journal, ensure in-place updates complete

CHECKSUM OPTIMIZATION

Can we get rid of barrier between (9, 10, 11) and 12 ?



In last transaction block, store checksum of rest of transaction

During recovery: If checksum does not match, treat as not valid

write order before

9,10,11

12

4,6

12

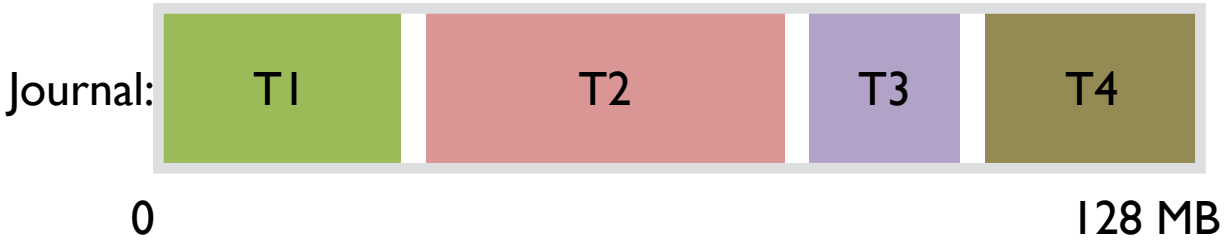
write order after

OTHER OPTIMIZATIONS

Batched updates

- If two files are created, inode bitmap, inode etc. get written twice
- Mark as dirty in-memory and batch updates

Circular log



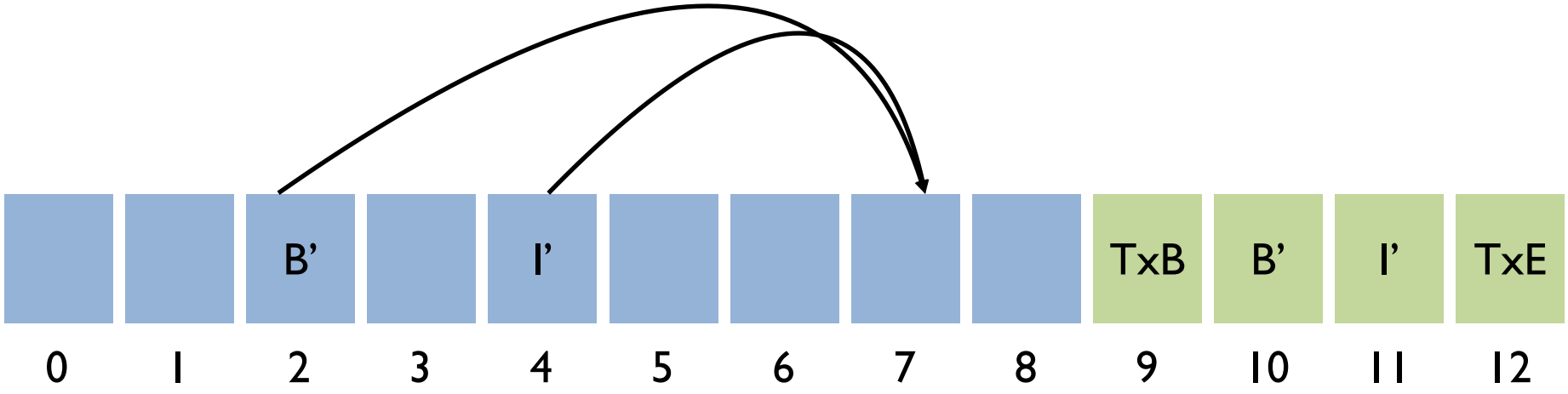
HOW TO AVOID WRITING ALL DISK BLOCKS TWICE?

Observation: Most of writes are user data (esp sequential writes)

Strategy: journal all metadata, including
superblock, bitmaps, inodes, indirects, directories

For regular data, write it back whenever convenient.

METADATA JOURNALING

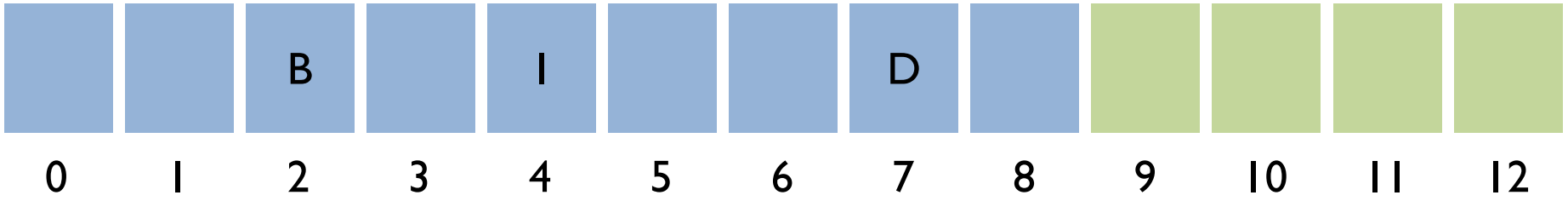


transaction: append to inode I

Crash !?!

ORDERED JOURNALING

Still only journal metadata. But write data **before** the transaction!



What happens if crash in between?

write order

7

9, 10, 11

12

2, 4

SUMMARY

Crash consistency: Important problem in filesystem design!

Two main approaches

FSCK:

- Fix file system image after crash happens
- Too slow and only ensures consistency

Journaling

- Write a transaction before in-place updates
- Checksum, batching, ordered journal optimizations

NEXT STEPS

Next class: How to create a file system optimized for writes