Welcome back

PERSISTENCE: FSCK, JOURNALING

Shivaram Venkataraman

CS 537, Spring 2020

ADMINISTRIVIA

Project 5: Are you working on it?! → Extra Credit!

Discussion this week: Practice for the final

AGENDA / LEARNING OUTCOMES

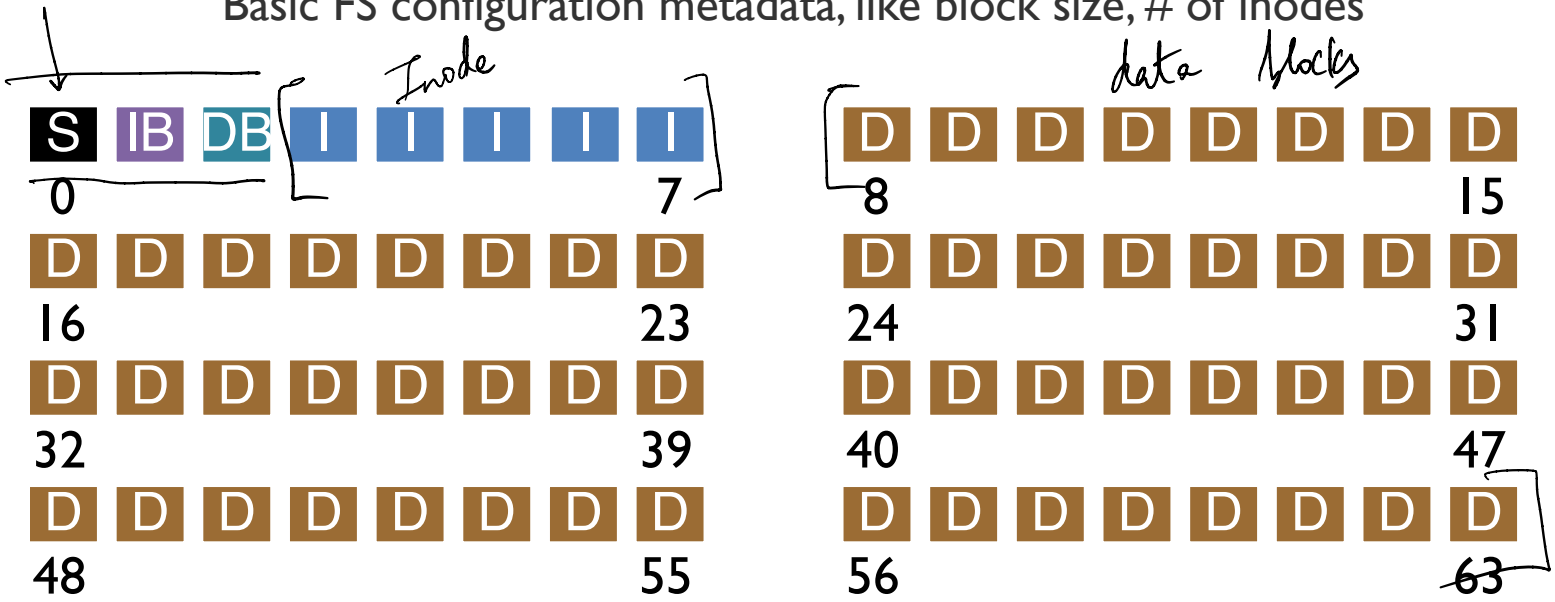
How to check for consistency with power failures / crashes?

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			

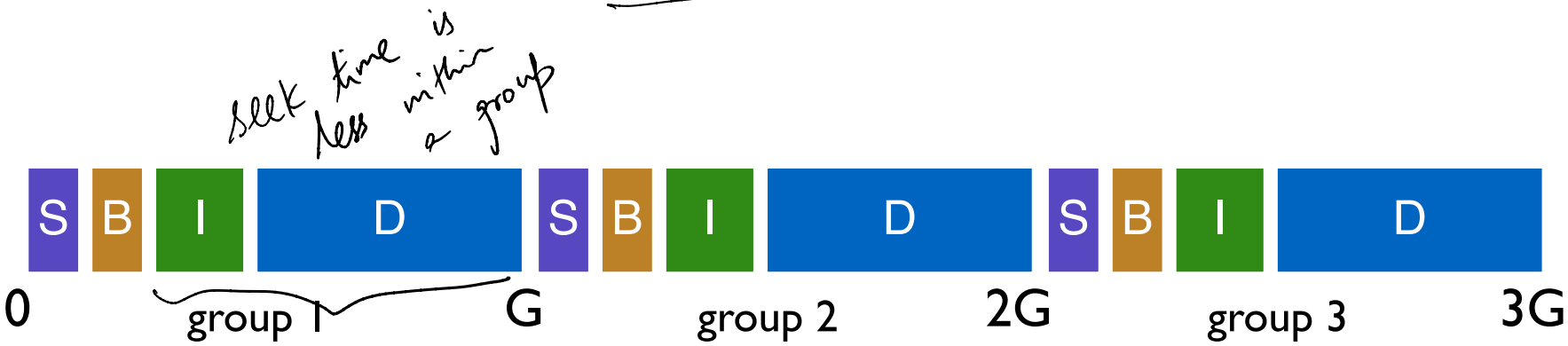
↓ Parent directory for bar

10 steps to create a file

inode → data block etc.

last modified

FFS PLACEMENT GROUPS



Key idea: Keep inode close to data

Use groups across disks;

Strategy: allocate inodes and data blocks in same group.

POLICY SUMMARY

All files in a dir are in same group

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

→ direct data block

Amortize time to seek to a new group

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

Move to another group (w/ fewer than avg blocks) every subsequent 1MB.

FILE SYSTEM CONSISTENCY

FILE SYSTEM CONSISTENCY EXAMPLE

Superblock: field contains total number of blocks in FS

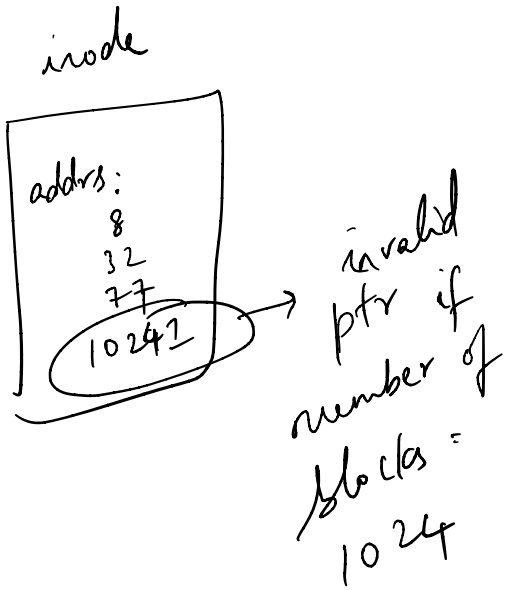
DATA = N ≃ 1024 in xv6

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?

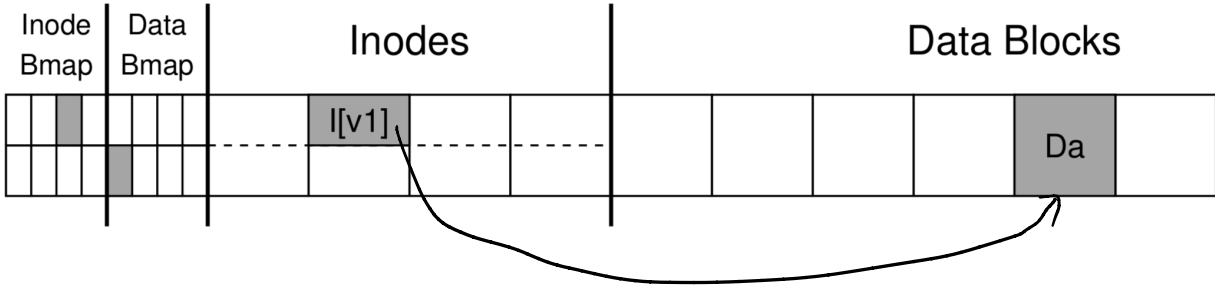
- power loss
- kernel panic
- reboot

→ OS crashed

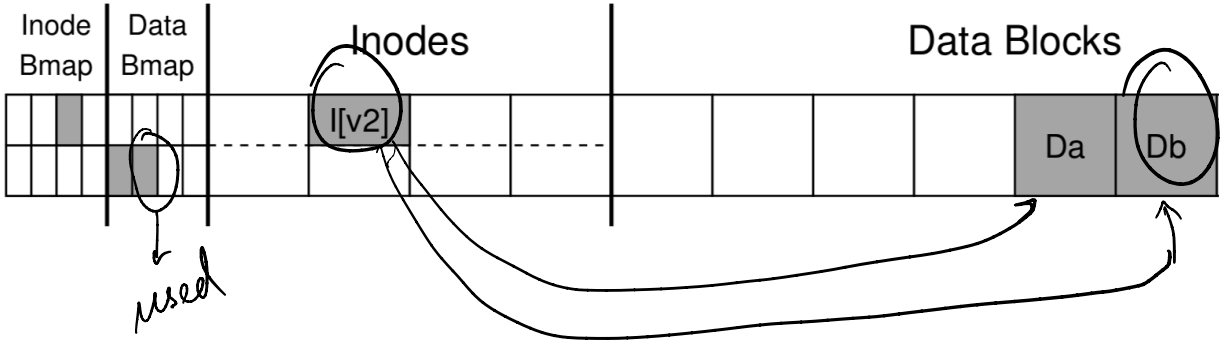
FILE APPEND EXAMPLE

Write 3 blocks to disk

1 data block



3 changes



1. Only data block D_b is on disk
   → Can't reach the data
2. Only inode is updated on disk
   → Pointing to garbage data
   → Bitmap inconsistent
3. Only bitmap is updated
   → Bitmap inconsistent
   But no inode points to it

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

Detect inconsistency

FSCK CHECKS

Do superblocks match?

Is the list of free blocks correct?

Do number of dir entries equal inode link counts?

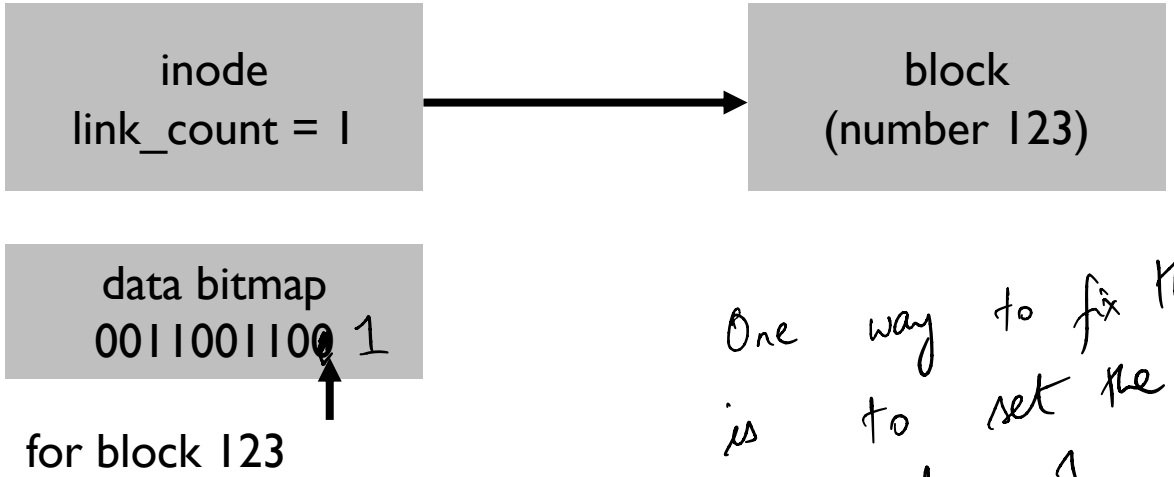
Do different inodes ever point to same block?

Are there any bad block pointers?

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

…

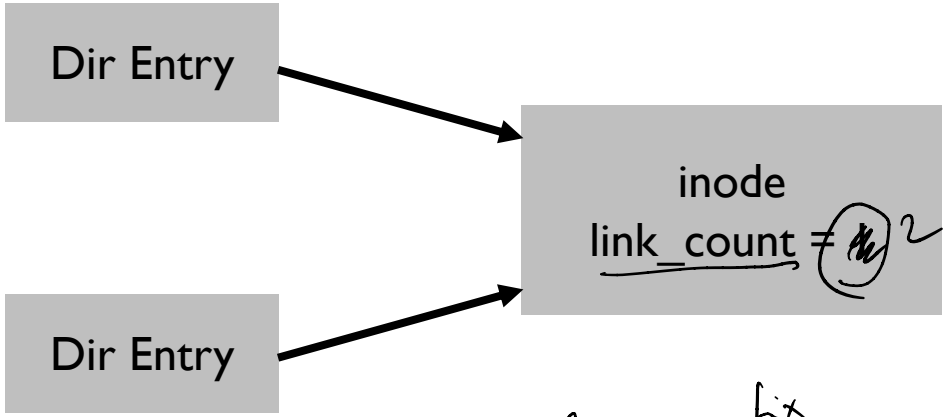
FREE BLOCKS EXAMPLE



One way to fix this is to set the bitmap bit to 1

LINK COUNT EXAMPLE

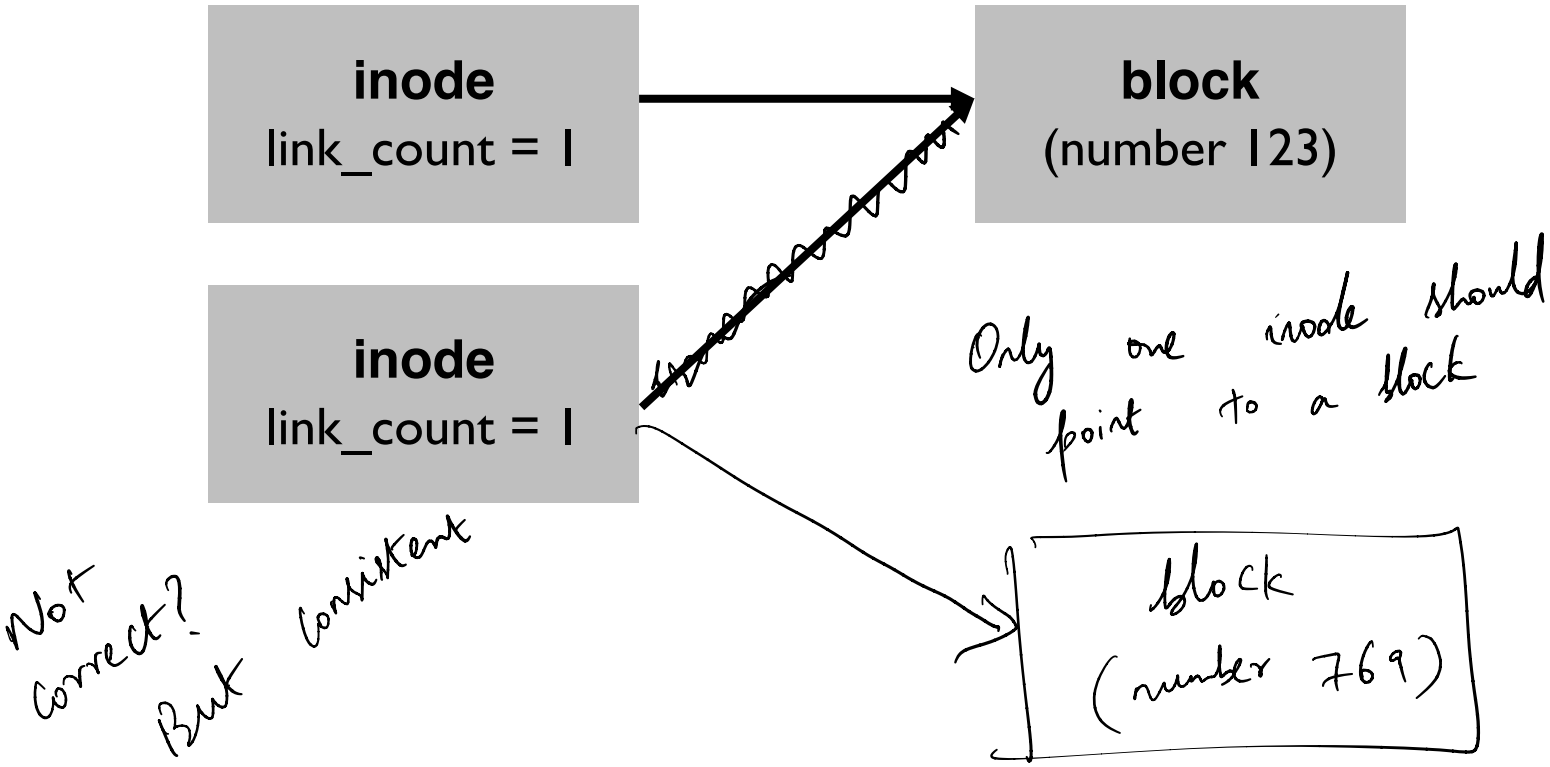
(hard links)



But 2 dir entries point to this inode

common fix is to make the count = 2

DUPLICATE POINTERS



BAD POINTER

inode
link_count = 1

999

9999

→ Can't access block 9999

super block
tot-blocks=8000

Common fix
→ Remove the addr which is > 8000

QUIZ 27

https://tinyurl.com/cs537-sp20-quiz27

(a) FILE SYSTEM STATE: Consistent or inconsistent? If inconsistent, how to fix

addr of data block

```
Inode Bitmap : 11111111
Inode Table  : [size=1,ptr=0,type=d] [] [] [] [] [] [] []
Data Bitmap  : 10000000
Data         : [("." 0),("..." 0)] [] [] [] [] [] [] []
```

Inconsistent.

Inode bitmap should be

1 0 0 0 0 0 0 0

```
Inode Bitmap : 11000000
Inode Table  : [size=1,ptr=0,type=d] [size=1,ptr=1,type=d] [] [] [] [] [] []
Data Bitmap  : 11000000
Data         : [("." 0),("..." 0),("a" 1)] [("." 1),("..." 0)] [] [] [] [] [] []
```

number of blocks?

/ /a

Consistent

```
Inode Bitmap : 11100000
Inode Table  : [size=1,ptr=0,type=d] [size=1,ptr=1,type=r] [size=1,ptr=2,type=r] [] [] [] [] []
Data Bitmap  : 11100000
Data         : [("." 0),("..." 0)] [DATA] [DATA] [] [] [] [] []
```

dir file

("a",1) ("b"2)

No dir entry pointing to inode 1 or 2

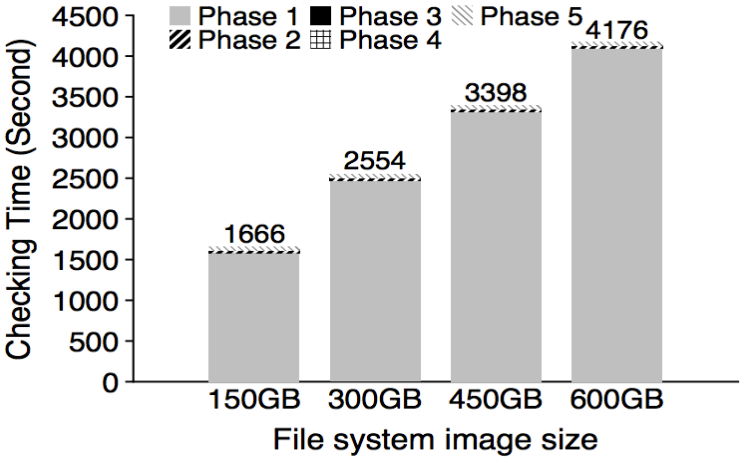
Inconsistent

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: FSCK IS VERY SLOW



2 TB disks

4 TB

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

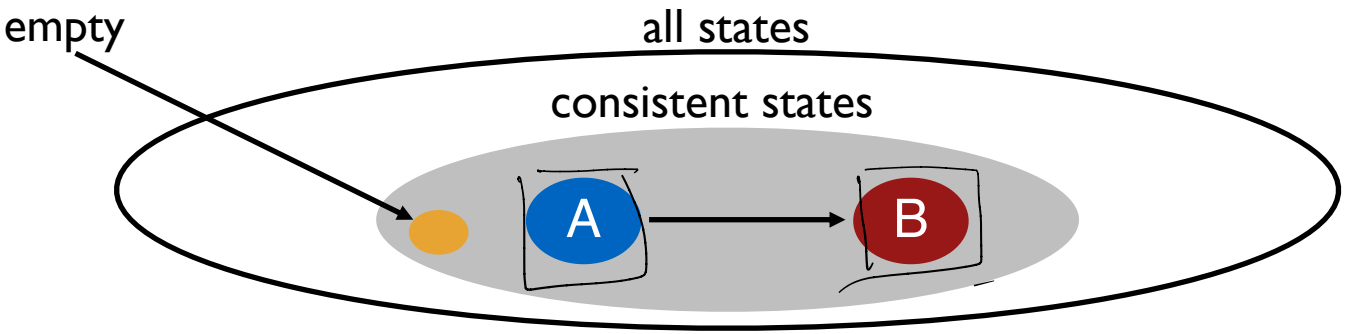
"Atomicity"

- 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

write (fd, buf, 512)
↳ Data block
↳ Inode
↳ Bitmaps

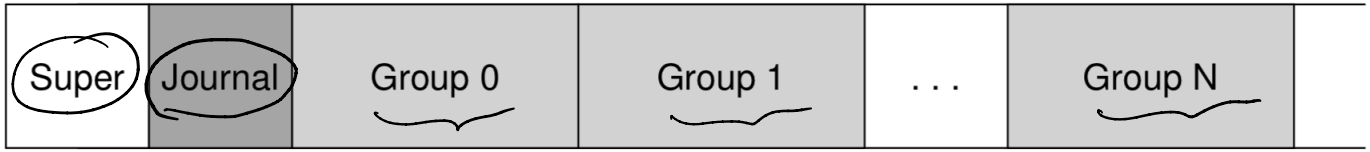
CONSISTENCY VS ATOMICITY

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

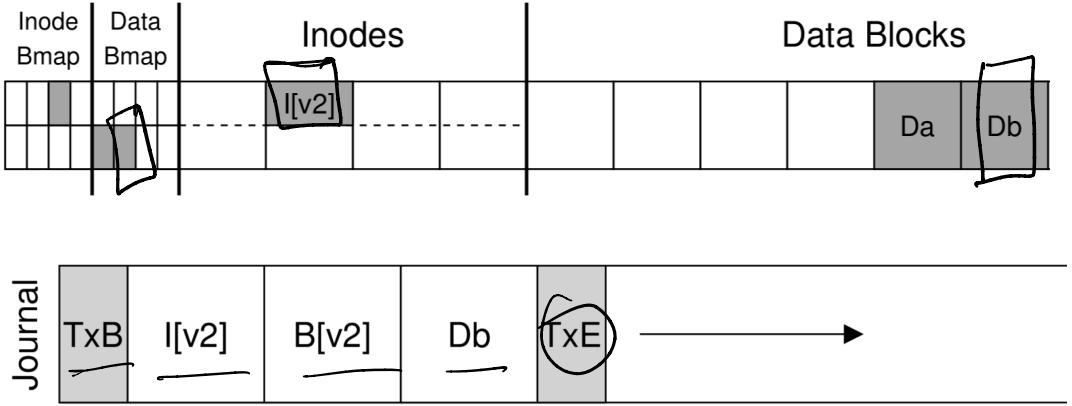


fsck gives consistency
Atomicity gives A or B.

JOURNAL LAYOUT



FFS



Start transaction

Transaction

1. Start transaction entry
2. Write blocks that belong to Tx to journal
3. End Tx.

JOURNAL WRITE AND CHECKPOINTS

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

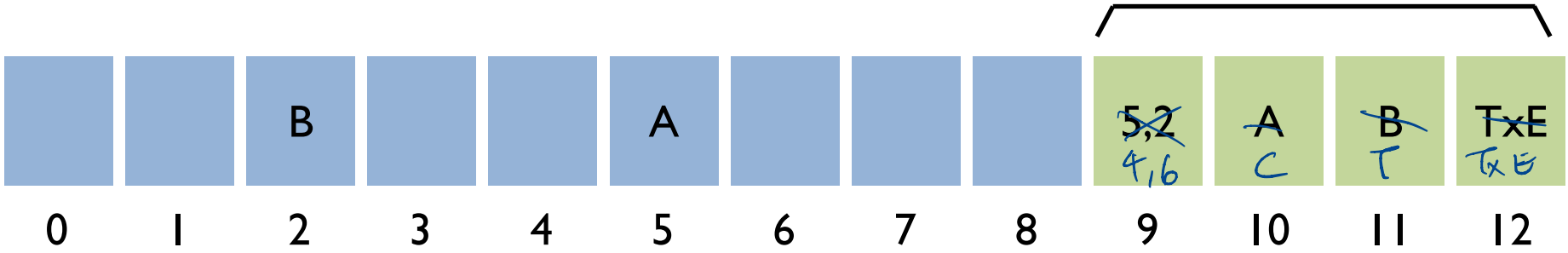
Checkpoint: Writing new data to in-place locations

Transaction

Checkpoint

Free journal for next Txn

JOURNAL REUSE AND CHECKPOINTS



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

Checkpoint: Writing new data to in-place locations

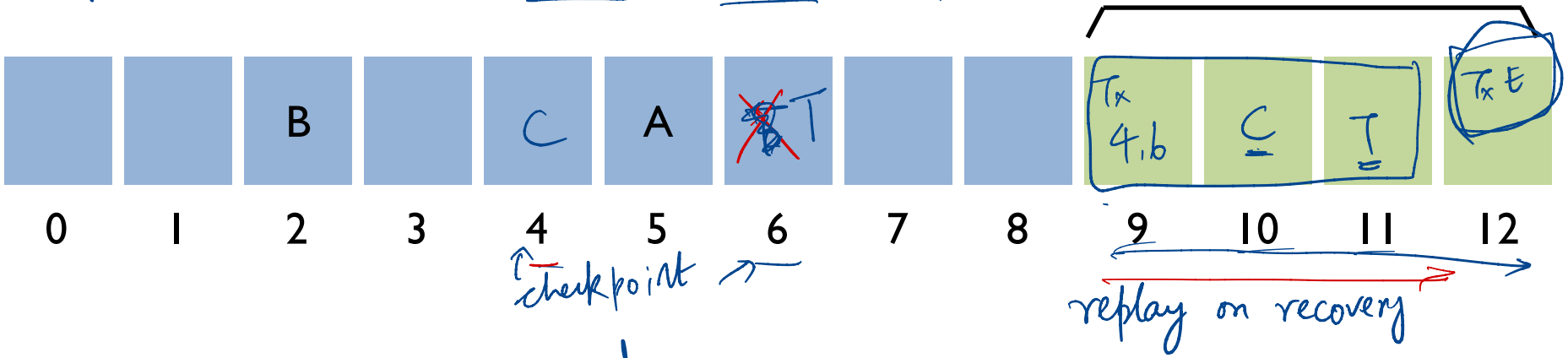
transaction: write C to block 4; write T to block 6

ORDERING FOR CONSISTENCY

Replay or redo logging

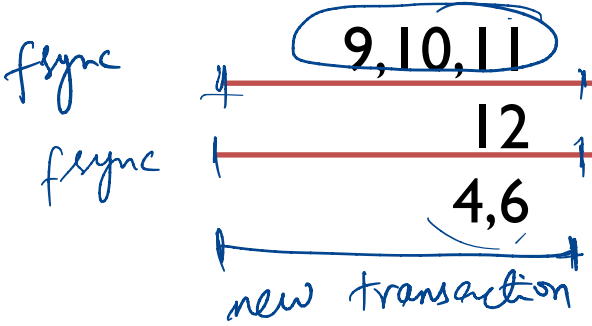
Journal on DISK!

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



Transaction End appears on disk

write order



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

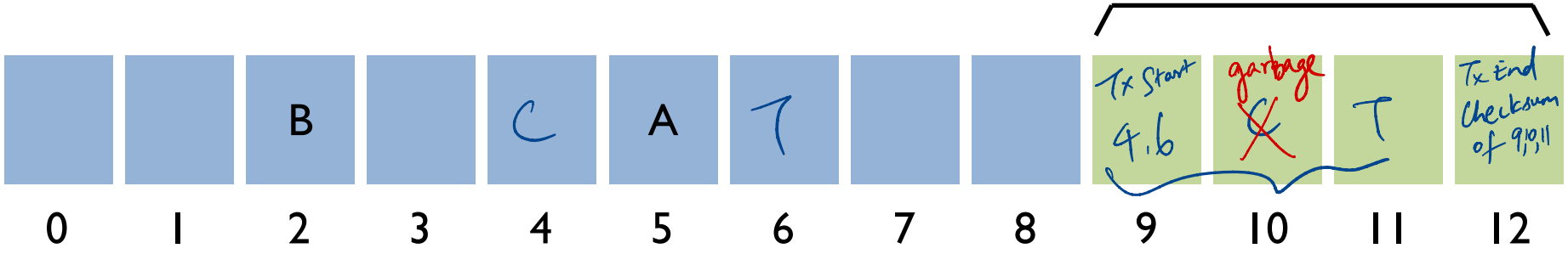
CHECKSUM OPTIMIZATION

fsync to impl barrier

checksum = hash

Chksum (9,10,11) ?

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



checksum (before) vs after

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

mark journal blocks as free

write order after

9, 10, 11, 12

4, 6

new txn here

OTHER OPTIMIZATIONS

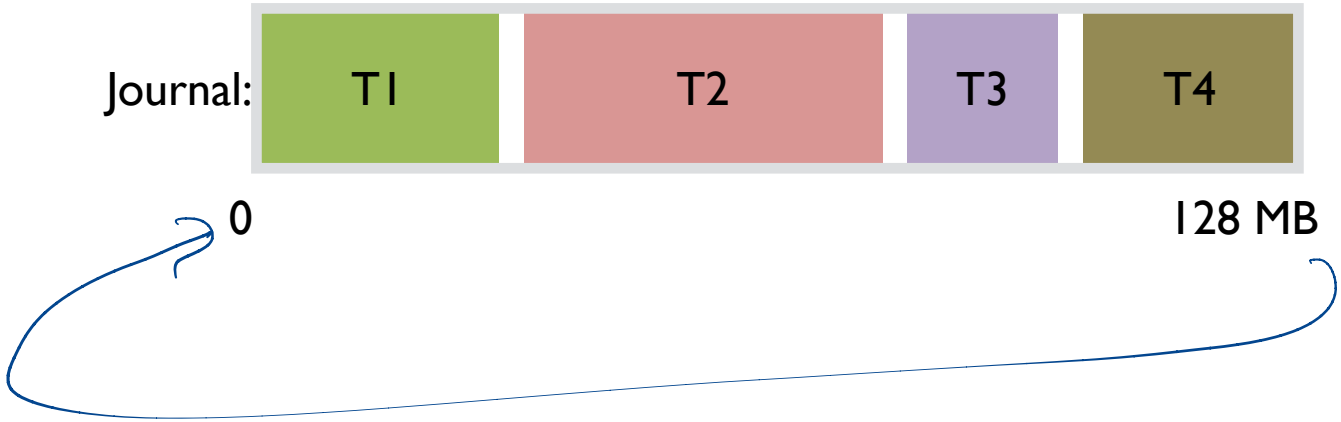
Batched updates

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

Tx create /a
create /b

update inode block 0
update data bitmap

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.

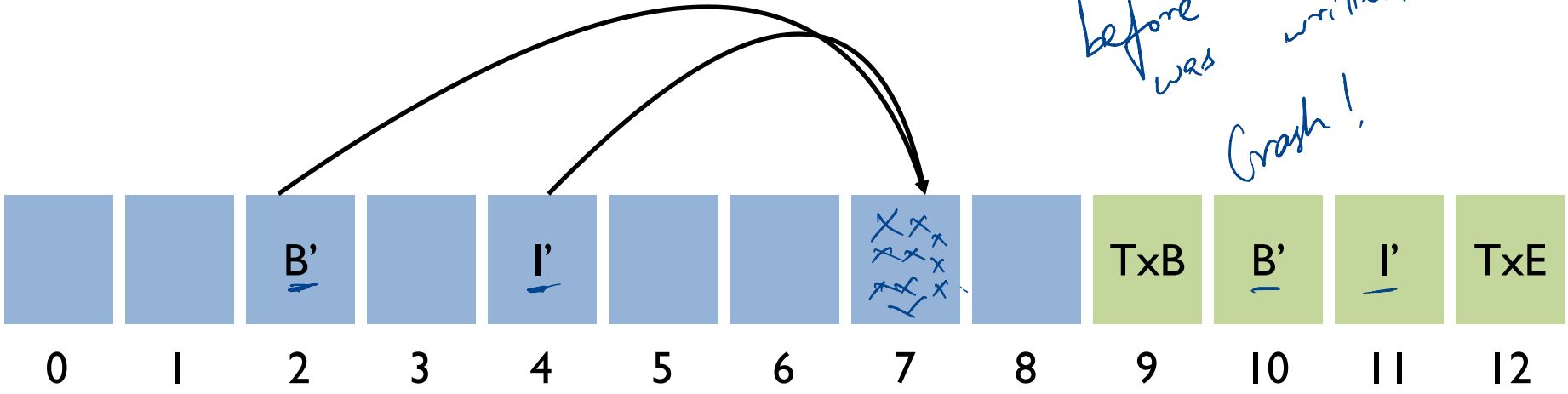
Prev: Data journaling

This: Metadata journaling

METADATA JOURNALING

before data block was written

Crash!



garbage data

transaction: append to inode I

Crash !?!

Not Correct!

wrt Atomicity

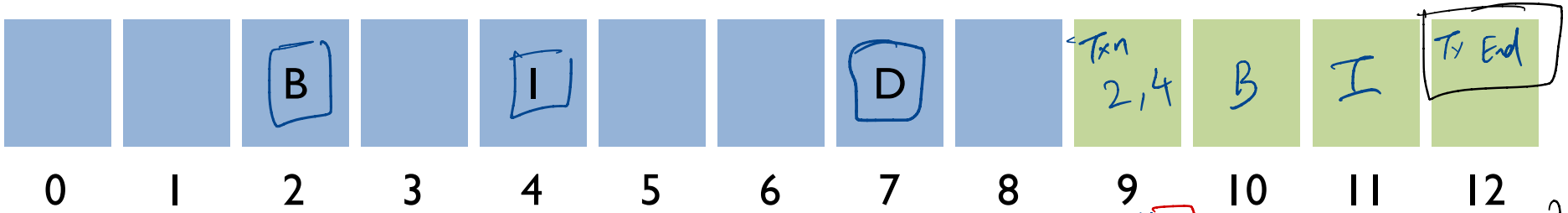
Txn abort [7, 9, 10, 11]

Txn commit replayed 12

ORDERED JOURNALING

Transaction happens only if this is on disk

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



Barrier?

1. First data blocks written in-place
2. Barrier remove
3. Start Txn metadata
4. Txn End
5. In-place updata for meta data

new

What happens if crash in between?

If data blocks are written and the crash as if nothing happened

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

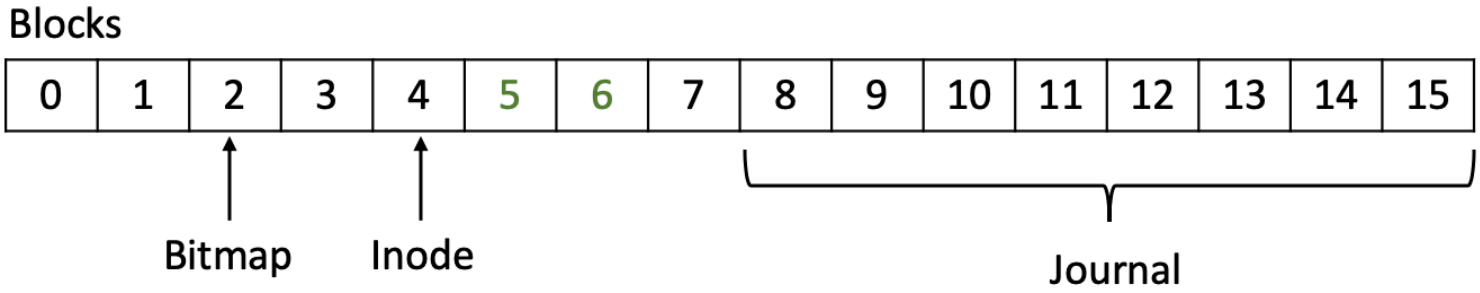
Data

Metadata → ordered

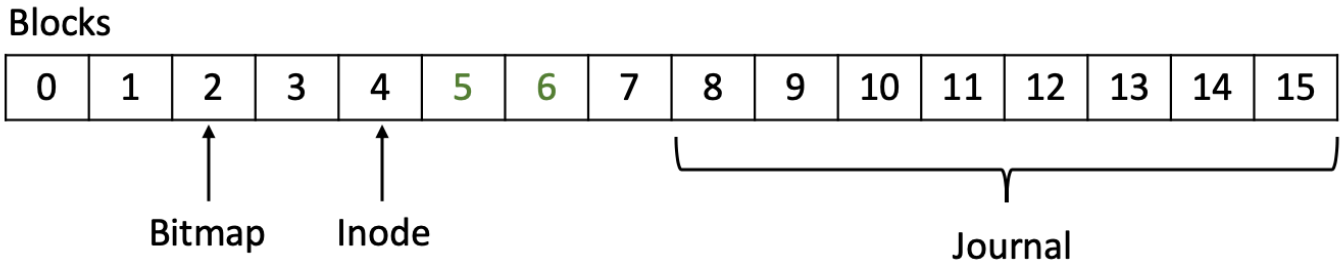
Ext 3 , ext 4

QUIZ 28

https://tinyurl.com/cs537-sp20-quiz28



Write 5,6
Write 8, 9, 10
Barrier
Write 11
Barrier
Write 4, 2



Write 8, 9, 10,11,12
Barrier
Write 13
Barrier
Write 2,4,5,6

Write 8, 9, 10,11,12, 13
Barrier
Write 2,4,5,6

NEXT STEPS

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

Project 5: Are you working on it?!