Hello!

Its warm!!

PERSISTENCE: FSCK, JOURNALING

Shivaram Venkataraman

CS 537, Spring 2023

ADMINISTRIVIA

Project 6 updates → Tests → Tomorrow!!

→ Timing

Midterm 2: Solutions, grades → 4pm

No class on Tuesday! →

AGENDA / LEARNING OUTCOMES

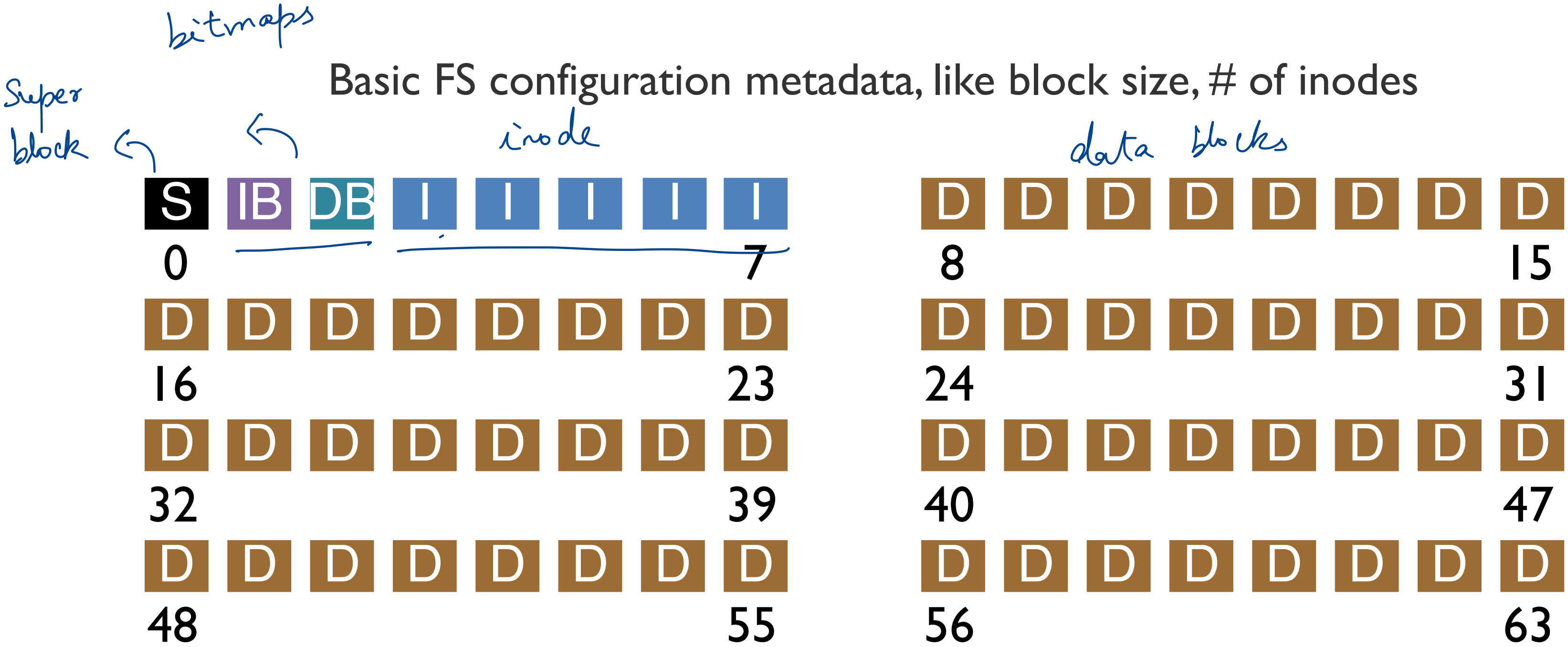
How to check for consistency with power failures / crashes?

How to ensure consistency in filesystem design?

RECAP

FS STRUCTS: SUPERBLOCK

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



create /foo/bar

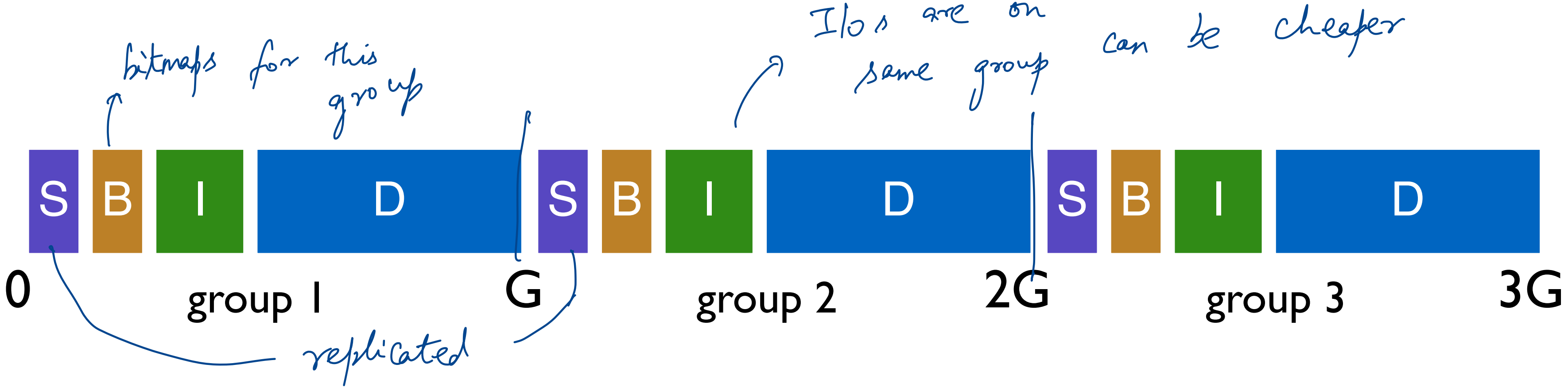
TIME

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
	allocate					
				8.read		
				9.write		
			10.write			

10 different I/o operations

for 1 call to create

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

/a.c } same group as /
/b. c

/src → new 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

inode data block

group 1 D D D D D D < >

group 4 D D D D D D

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

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

OTHER FFS FEATURES

FFS also introduced several new features:

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

Inspired modern files systems, including ext2 and ext3 , ext4

FILE SYSTEM CONSISTENCY

FILE SYSTEM CONSISTENCY EXAMPLE

Superblock: field contains total number of blocks in FS

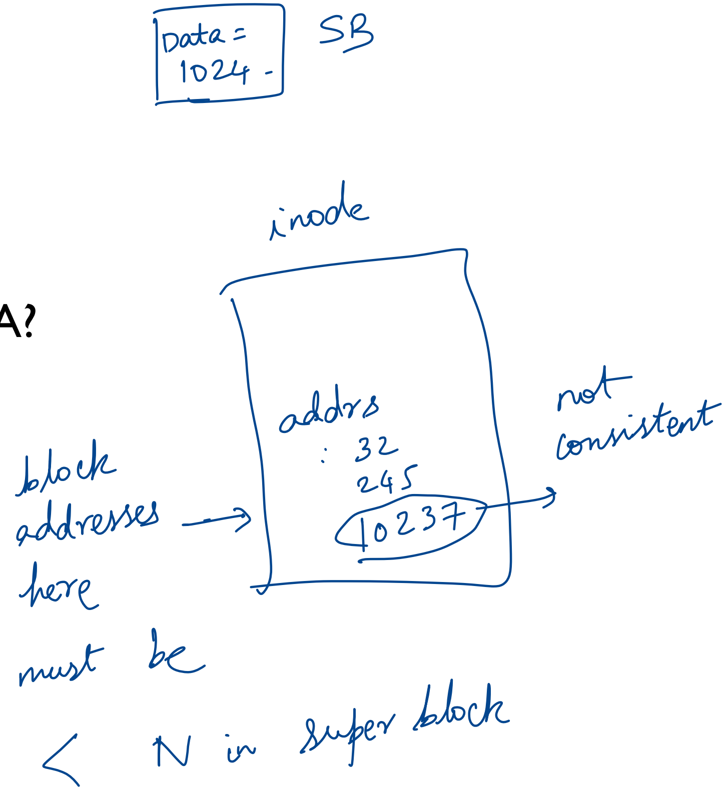
DATA = N   1024

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

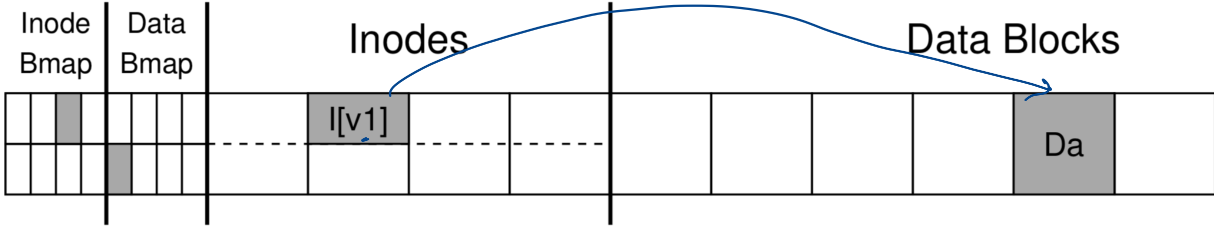
If file system is interrupted between writes, may leave data in inconsistent state

What can interrupt write operations?

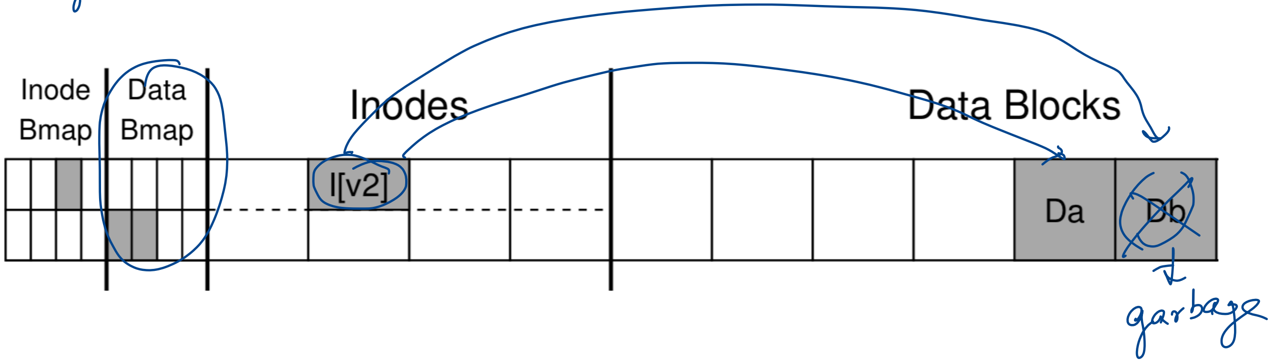
- power loss
- kernel panic
- reboot

FILE APPEND EXAMPLE

Old state



Target state



① New data block

→ No way to read this block

② Inode needs new ptr

→ Pointer to garbage!

→

③ Data bitmap mark block as used

→ space waste as block is not really used!

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? → replicated

Is the list of free blocks correct?

Do number of dir entries equal inode link counts?

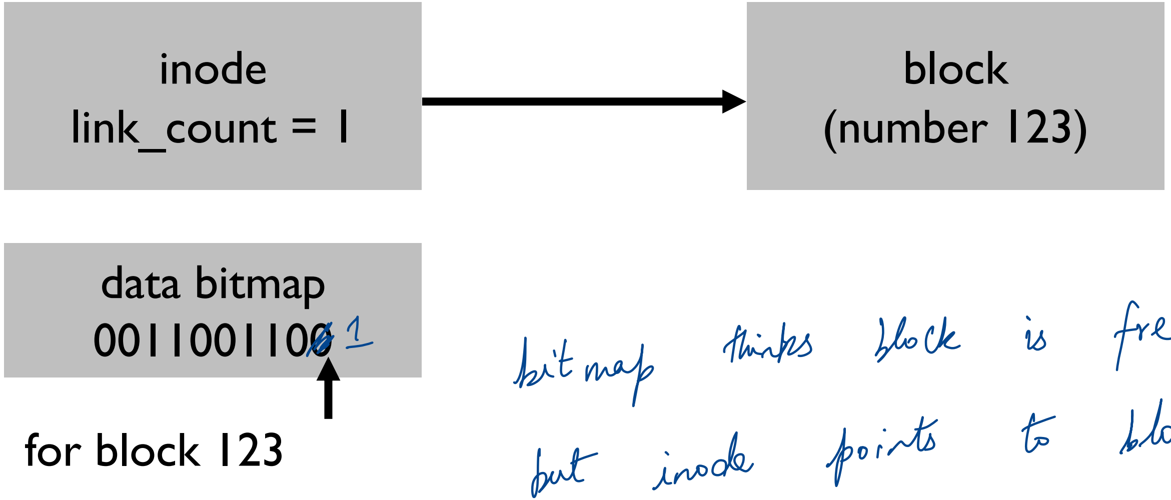
Do different inodes ever point to same block?

Are there any bad block pointers?

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

…

FREE BLOCKS EXAMPLE



inode
link_count = 1

block
(number 123)

data bitmap
001100110~~0~~1

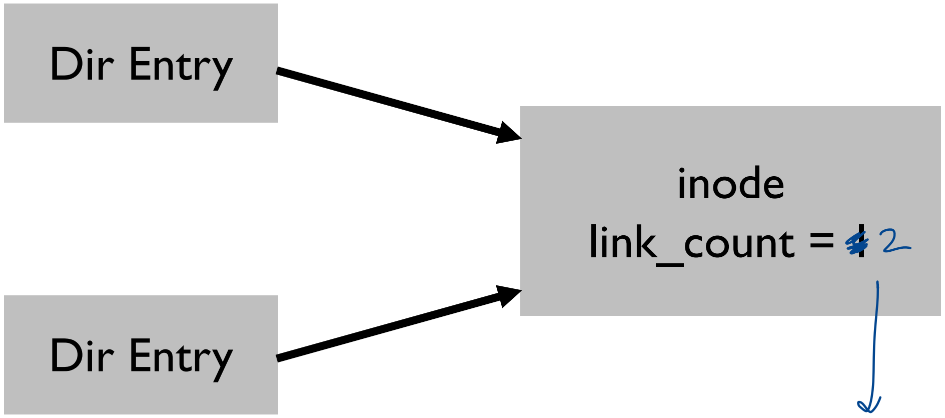
for block 123

bit map thinks block is free

but inode points to block

Fix: change the bit 123 to be used
in bitmap!

LINK COUNT EXAMPLE



Dir Entry

Dir Entry

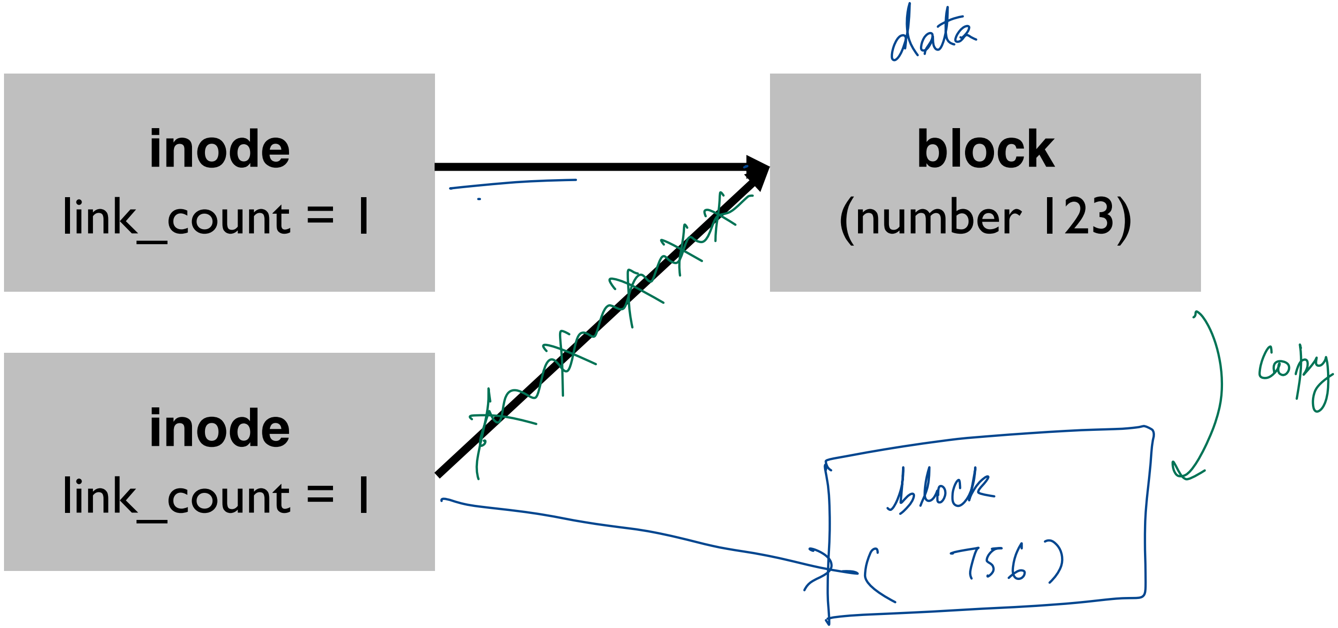
inode

link_count = ~~1~~ 2

update link count

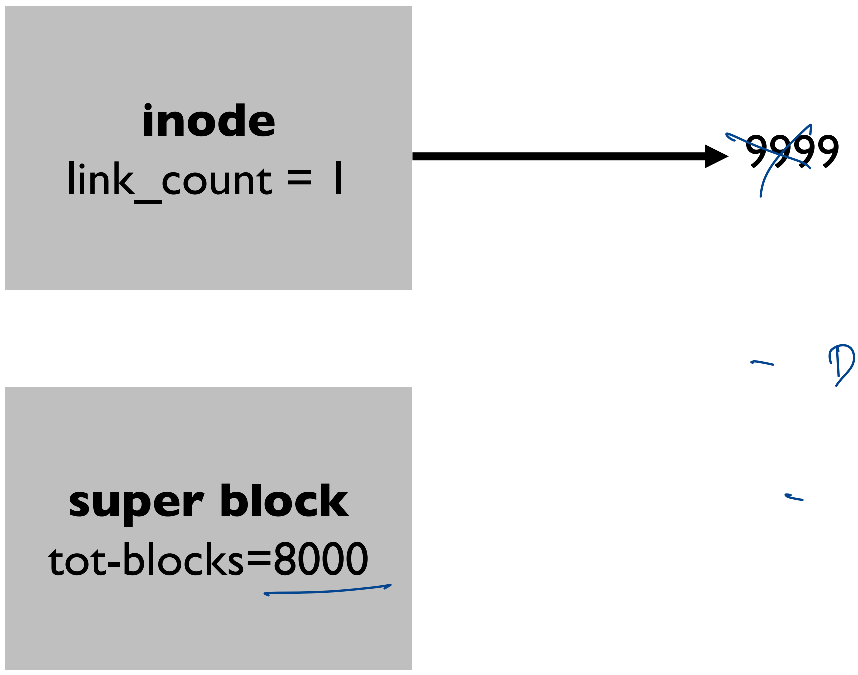
link_ count tracks how many dir entries point to this inode

DUPLICATE POINTERS



FS Consistent but may not be correct

BAD POINTER



inode
link_count = 1

~~9999~~

super block
tot-blocks=8000

(should not be greater than 8000)

- Delete the ptr

- Make it point to an empty block

QUIZ 28

https://tinyurl.com/cs537-sp23-quiz28

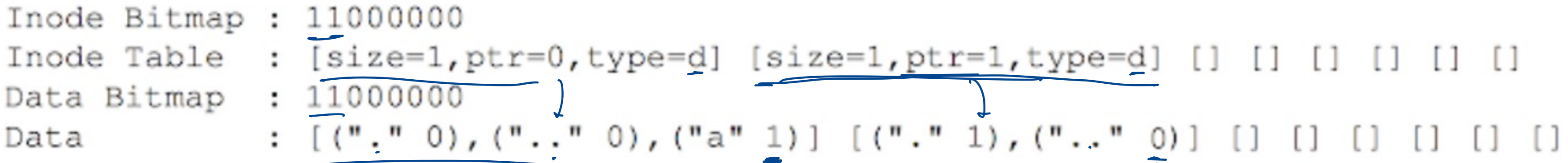
(a) FILE SYSTEM STATE: Consistent or inconsistent? If inconsistent, how to fix?

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

Inconsistent.

Fix: Update Inode Bitmap to be

10000000



```
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)] [] [] [] [] [] []
```

"/"

"/a"

parent is /

```
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] [] [] [] [] []
```

Inconsistent.

Create entries

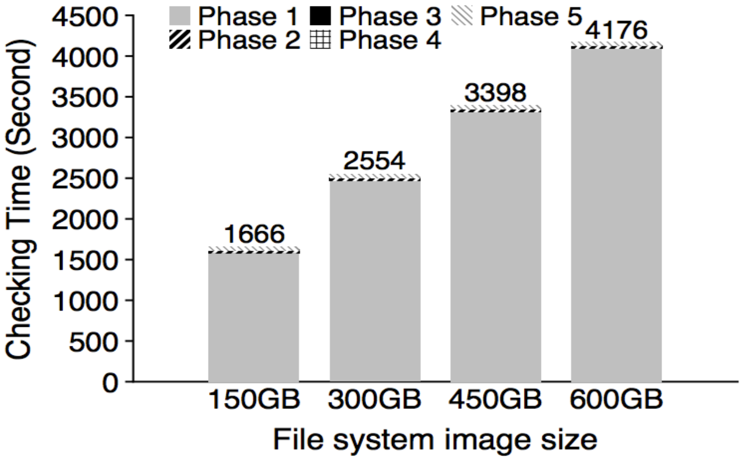
"."	0
".."	0
"file1"	1
"file2"	2

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



1 TB | 2 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

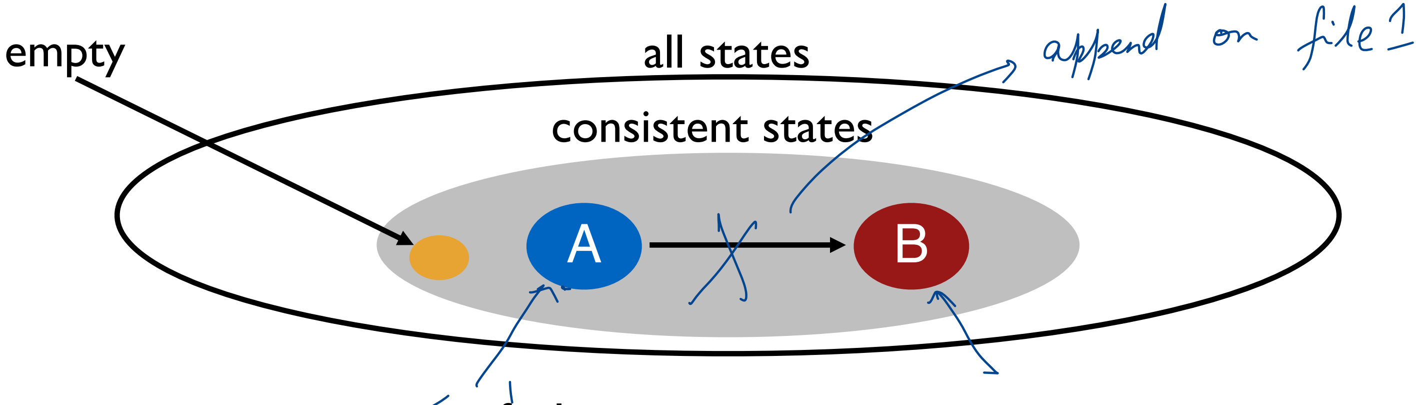
- Ok to do some **recovery work** after crash, but not to read entire disk
- Don't move file system to just any consistent state, get **correct** state

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

CONSISTENCY VS ATOMICITY

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



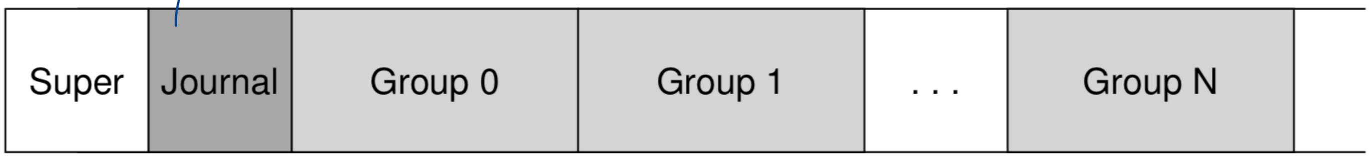
fsck gives consistency

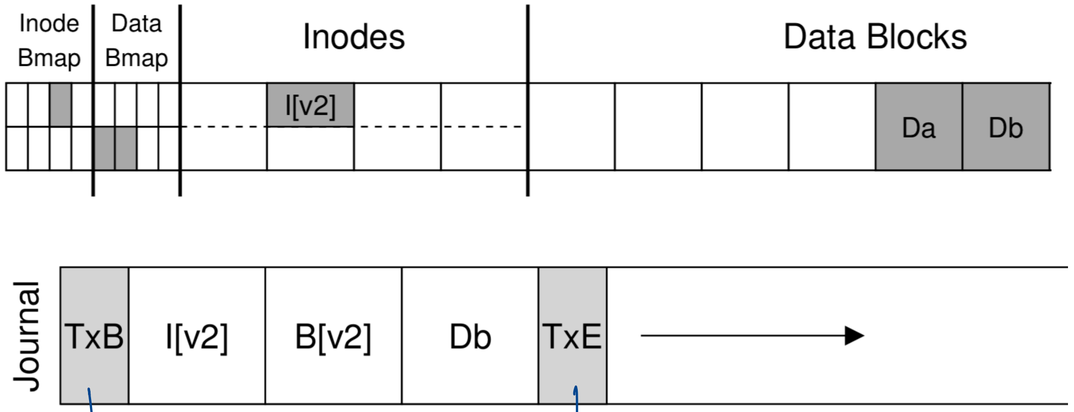
Atomicity gives A or B.

JOURNAL LAYOUT

reserve some blocks

replay of journal during recovery





Transaction begin

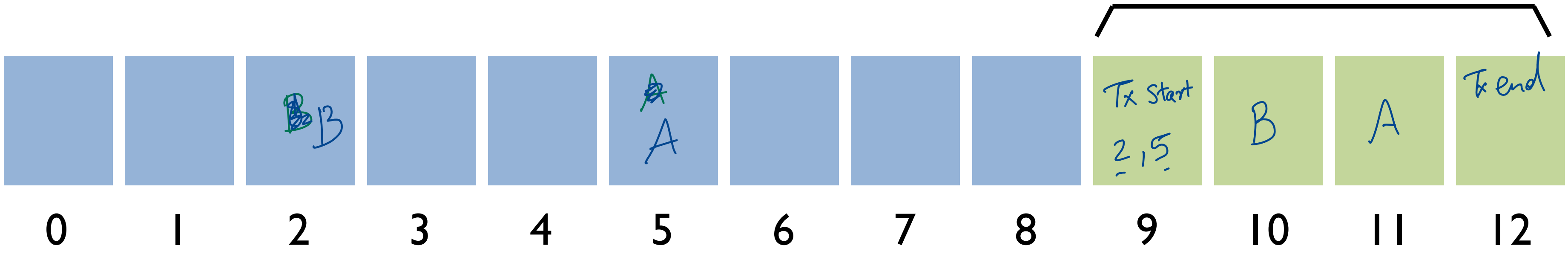
end

Transaction

1. Start Tx entry
2. Write blocks that belong to Tx
3. Tx end entry

Flush the journal

JOURNAL WRITE AND CHECKPOINTS



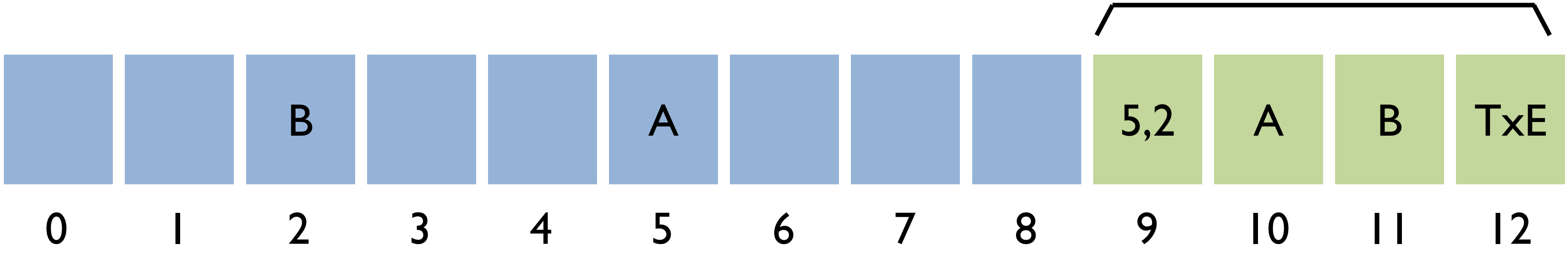
transaction: write A to block 5; write B to block 2

Checkpoint: Writing new data to in-place locations

flush journal to disk

name for this step

JOURNAL REUSE AND CHECKPOINTS



0	1	2	3	4	5	6	7	8	9	10	11	12
		B			A				5,2	A	B	TxE

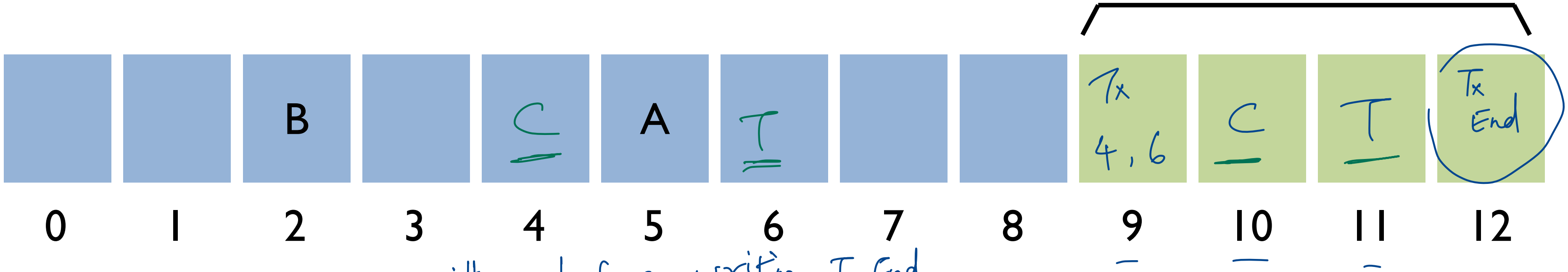
transaction: write A to block 5; write B to block 2

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



① Ensure 9, 10, 11 are written before writing Tx End

② Ensure Tx is committed. Any failure after will replay

flushing writes to disk ← fsync()

write order

9,10,11

12

4,6

barrier if you want reuse journal

Barriers

1) Before journal commit, ensure journal entries complete

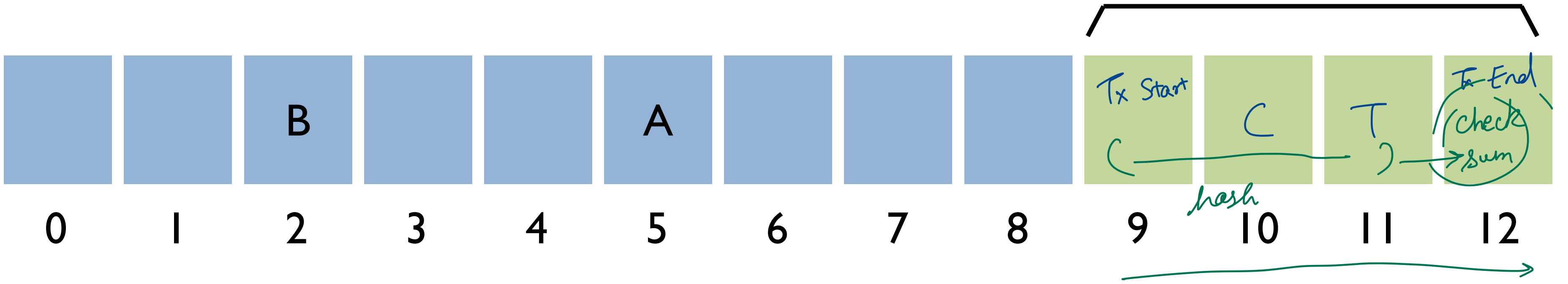
2) Before checkpoint, ensure journal commit complete

3) Before free journal, ensure in-place updates complete

CHECKSUM OPTIMIZATION

→ research project

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

dont need this

write order before

9,10,11

12

4,6

12

write order after

9, 10, 11, 12

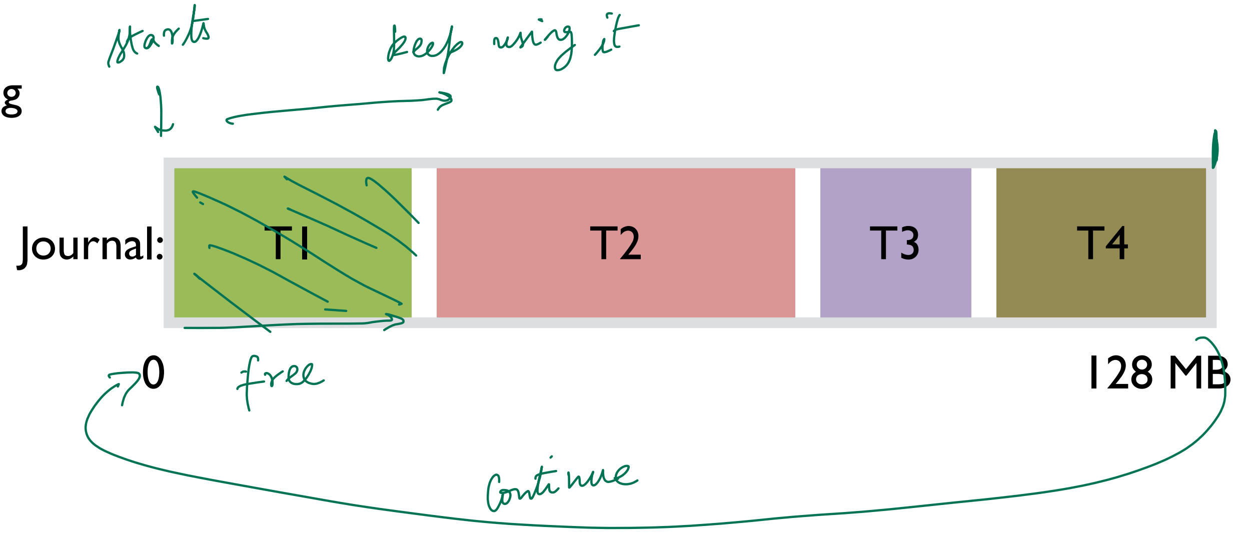
4, 6

OTHER OPTIMIZATIONS

Batched updates → write → Put both inside same transaction
write

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

Circular log



Journal: T1 T2 T3 T4

0 128 MB

HOW TO AVOID WRITING ALL DISK BLOCKS TWICE?

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

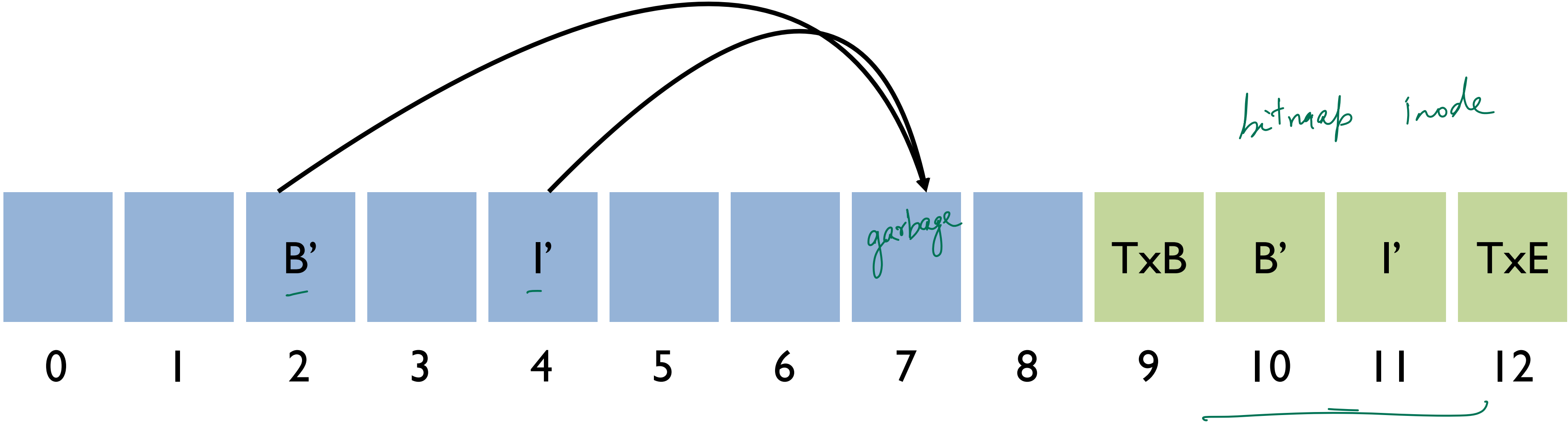
Strategy: journal all metadata, including superblock, bitmaps, inodes, indirects, directories

→ still go to the journal

For regular data, write it back whenever convenient.

↓ data in files not be in the journal

METADATA JOURNALING



0	1	2	3	4	5	6	7	8	9	10	11	12
		B’		I’			garbage		TxB	B’	I’	TxE

replayed after crash

transaction: append to inode I

Crash !?!

ORDERED JOURNALING → default ext3

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

		B		I			D		Tx Start 2, 4	B'	I'	Tx End
0	1	2	3	4	5	6	7	8	9	10	11	12

What happens if crash in between?

Crash

write order

First data → 7

Txn contents → 9, 10, 11

barrier ←

12

replay the Txn and data has been written to disk ←

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

No class on Tuesday!

Next time we meet: How to create a file system optimized for writes