Operating Systems

Fast File System

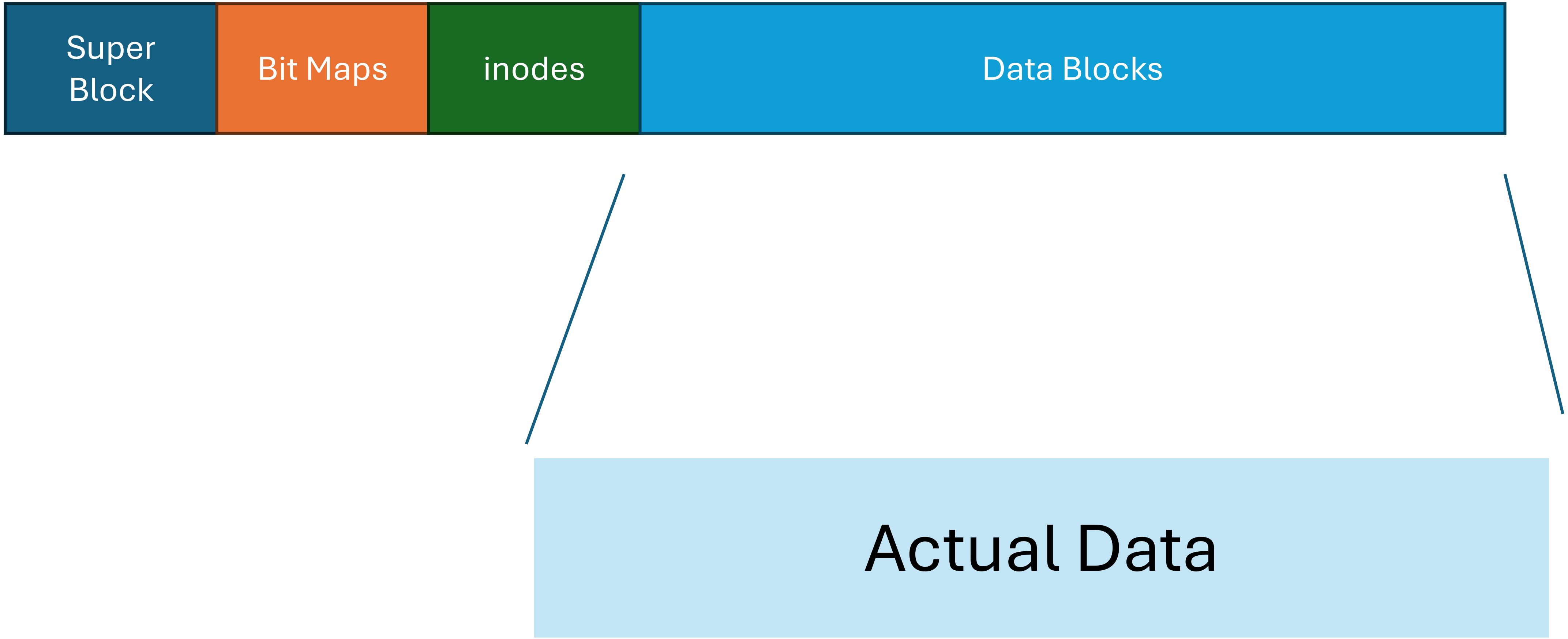
Overview

- FFS: Fast File Systems
- LFS: Log-Structured File Systems

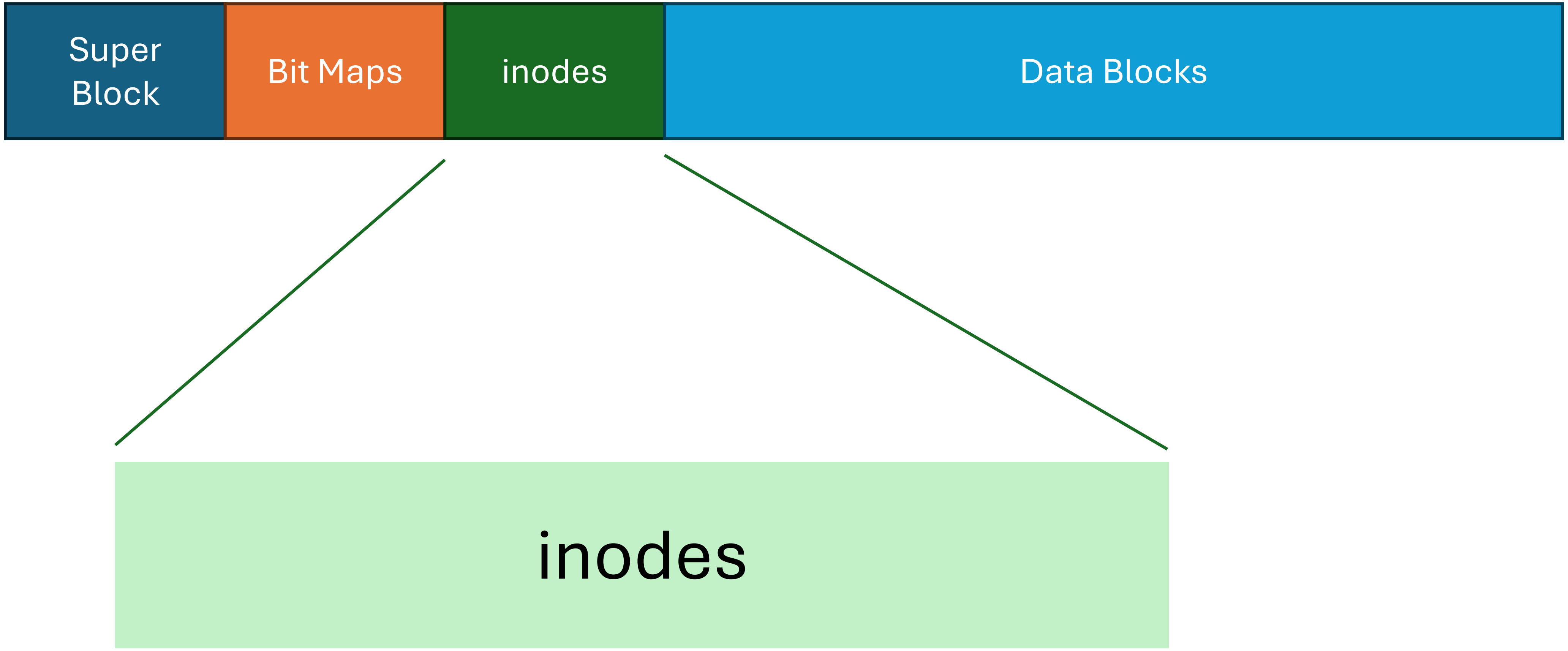
- File System Checking (FSCK)
- Journaling

Review

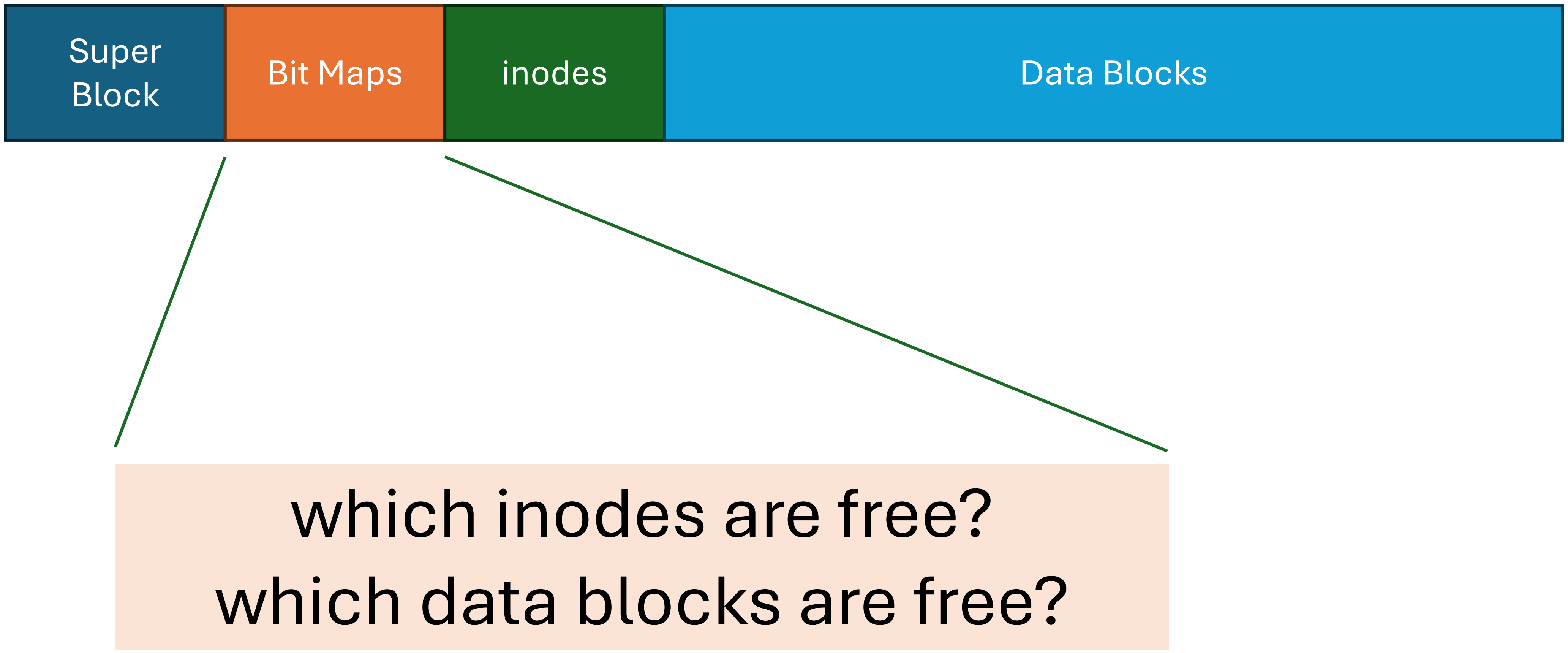
Review



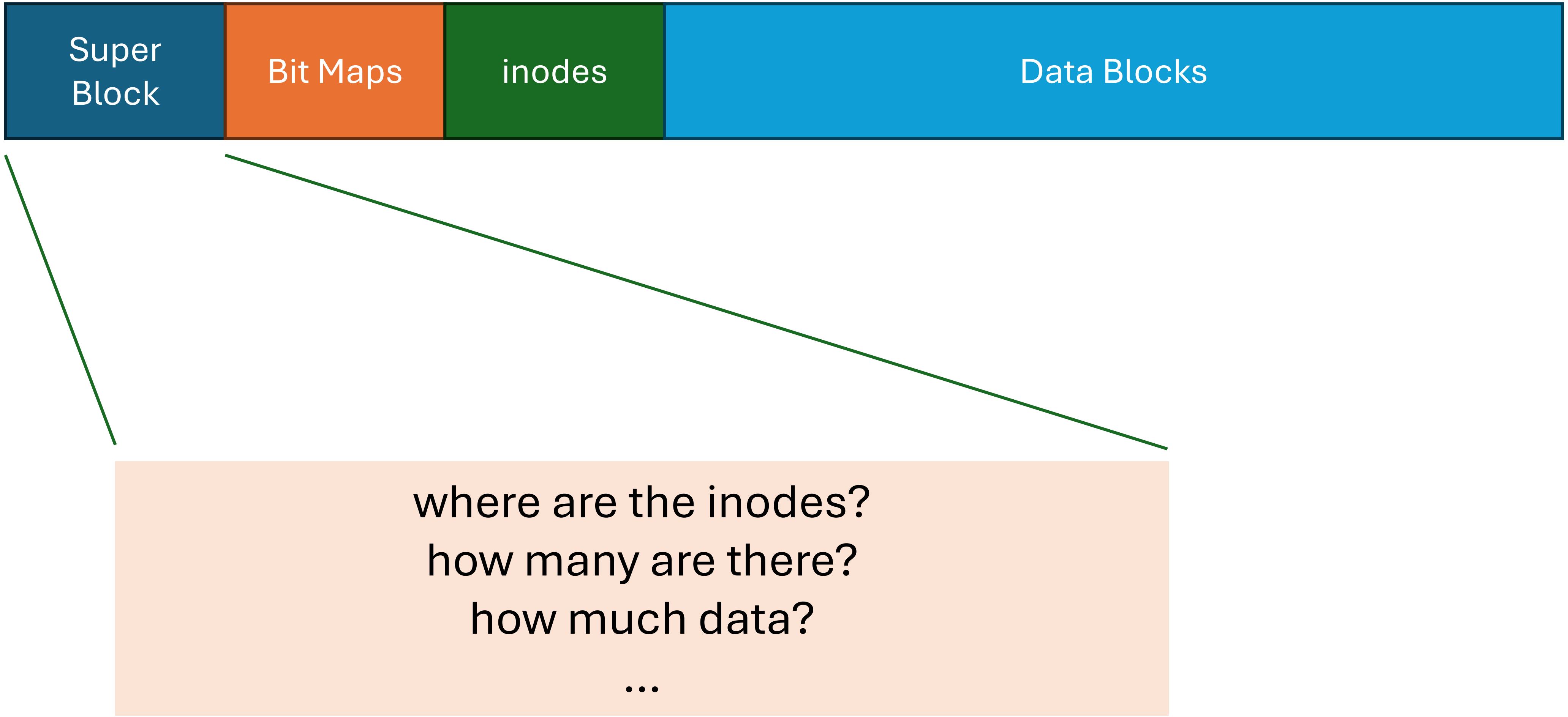
Review



Review


Super Block
Bit Maps
inodes
Data Blocks
which inodes are free?
which data blocks are free?

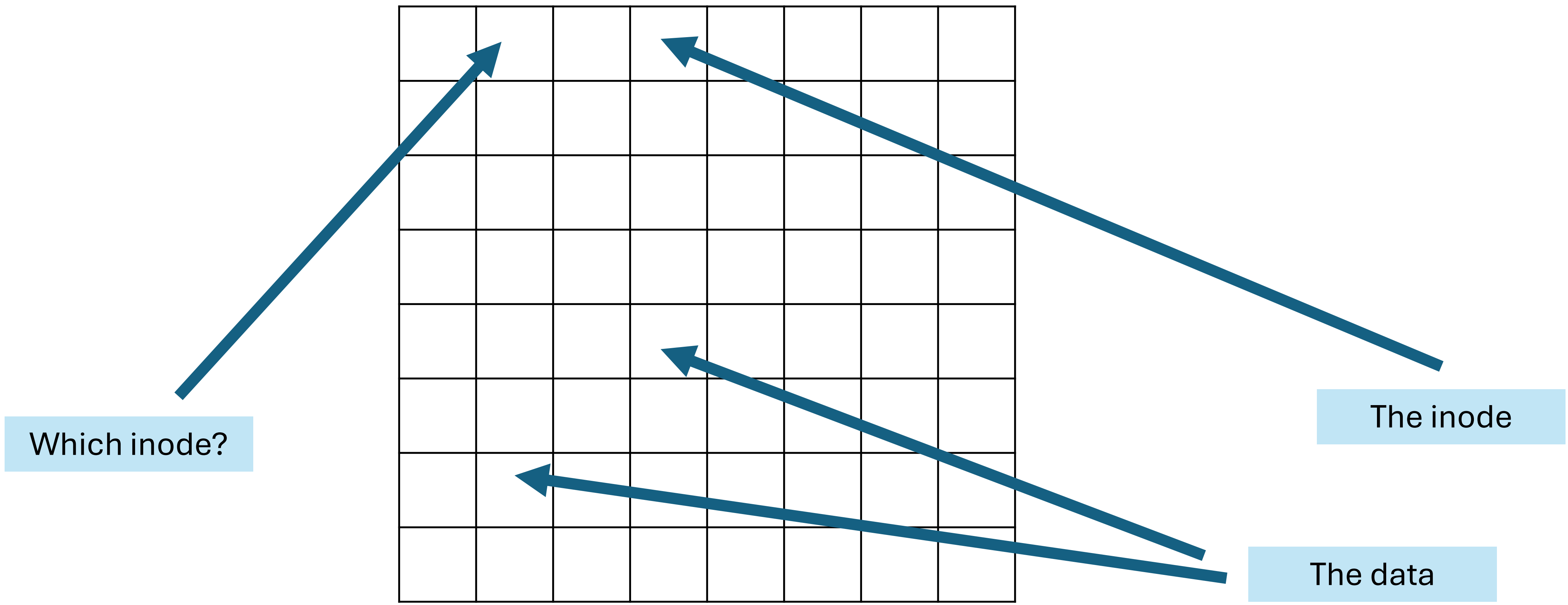
Review


Super Block
Bit Maps
inodes
Data Blocks
where are the inodes?
how many are there?
how much data?
…

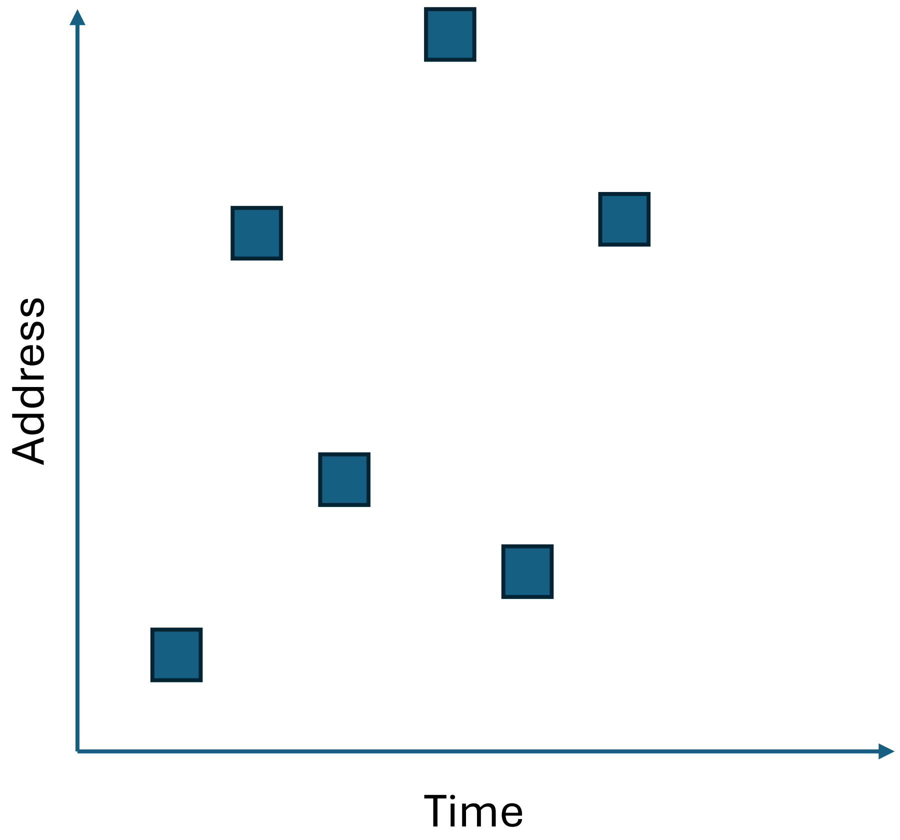
What went wrong?

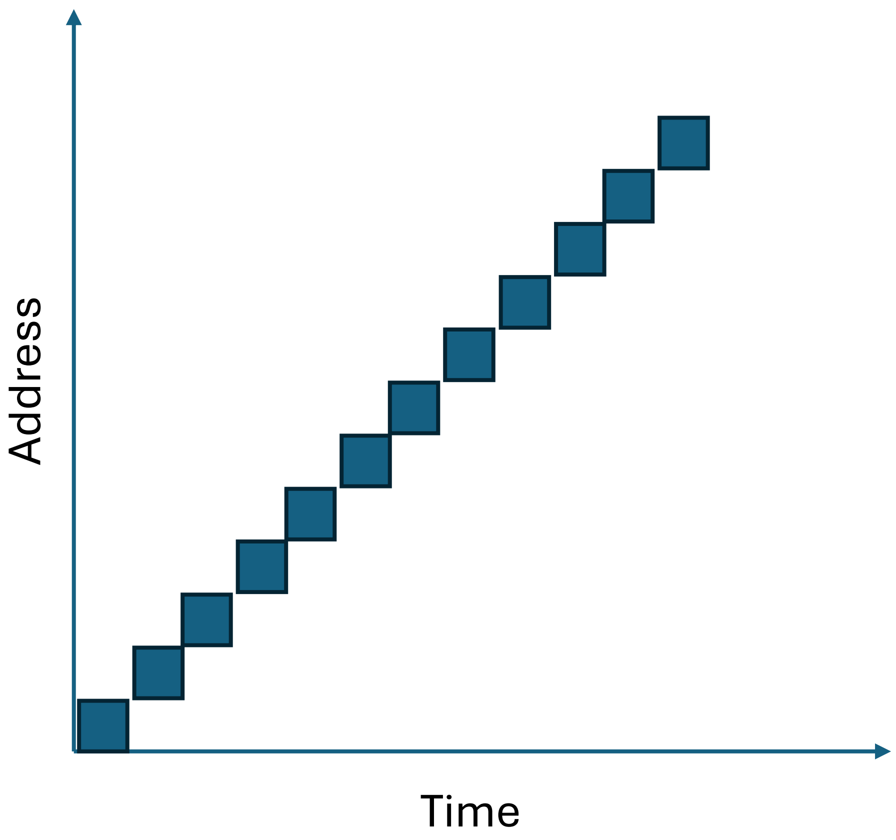
PERFORMANCE!

inode -> data



Locality

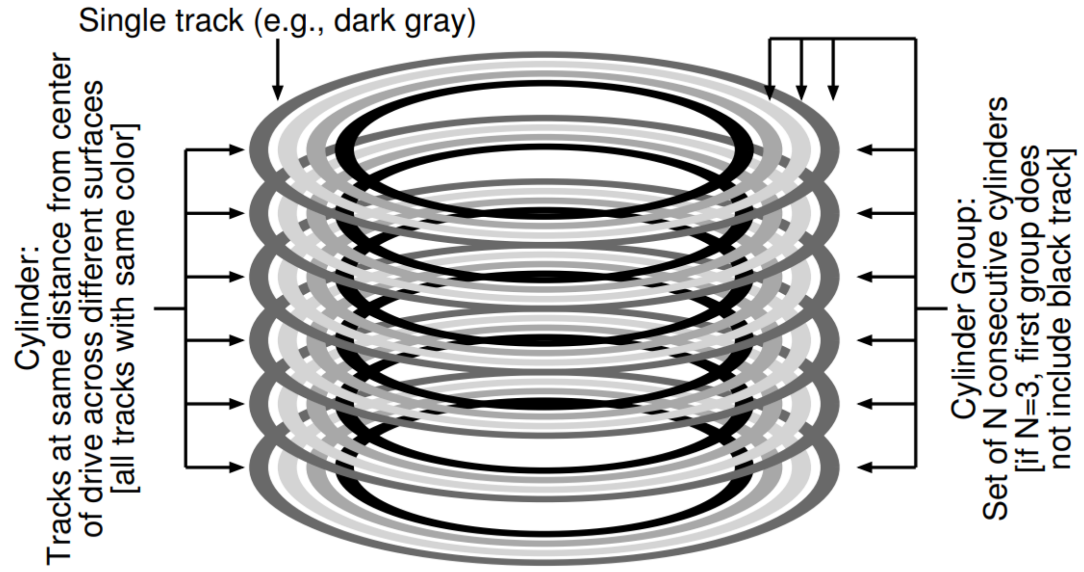




Cylinder Groups

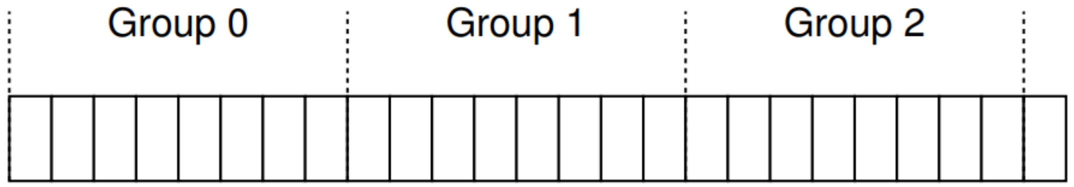
Cylinder groups are separated by ‘seek time’ dimension (i.e., radius)

Here 3 cylinders are grouped into a single ‘cylinder group’



Block Groups

Consecutive portions of the disk's address space

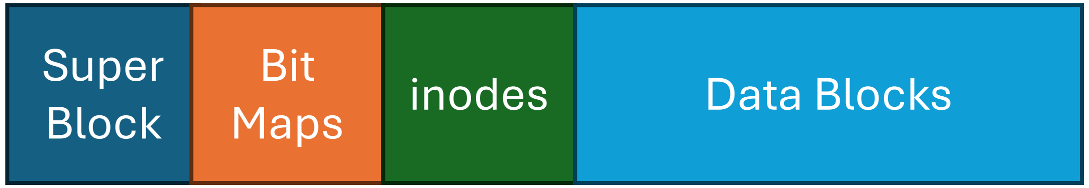


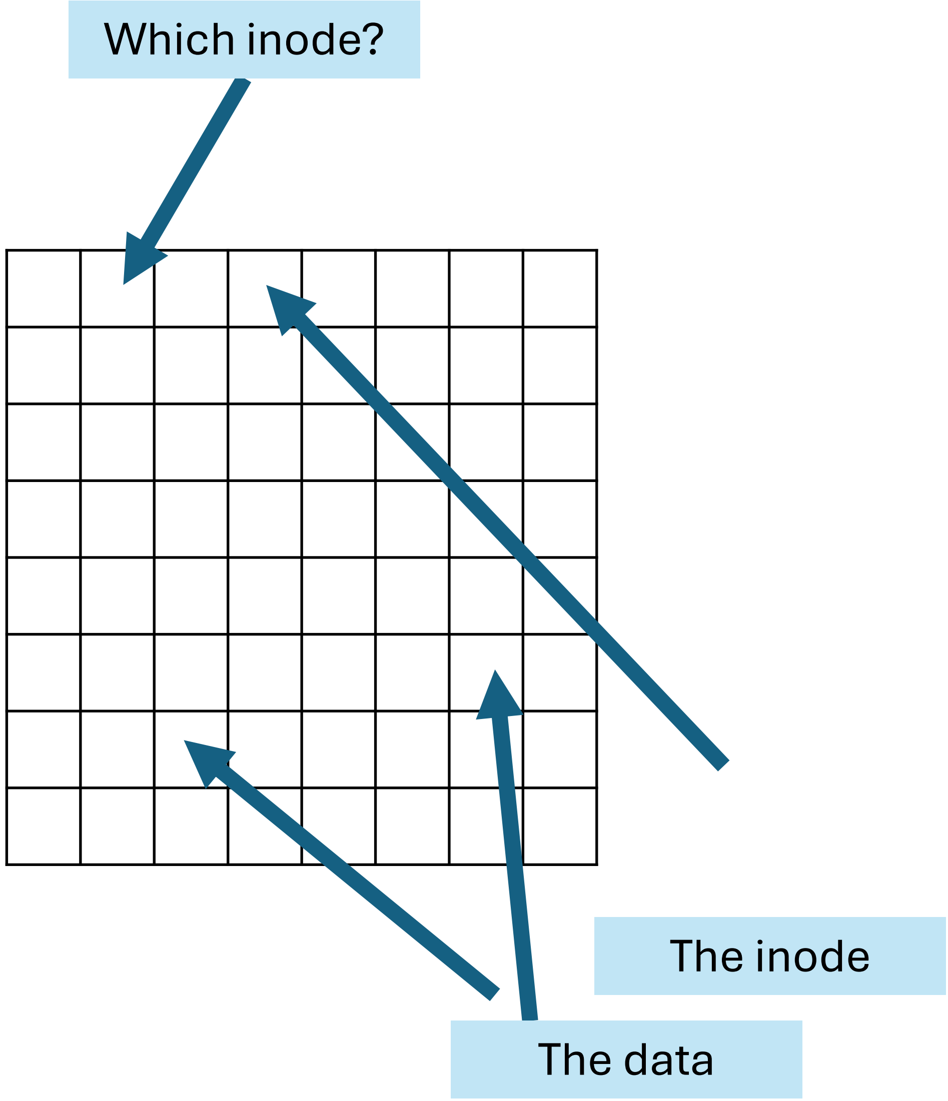
The Cunning Plan

Groups

- If two files are in the same group, the time to access both files ‘together’ will be shorter

Cylinder Group Structure





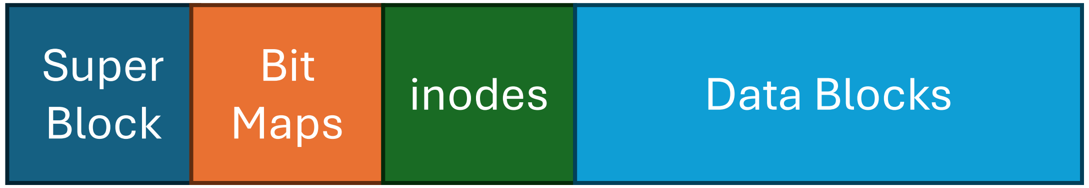
The Cunning Plan

- Groups
- If two files are in the same group, the time to access both files ‘together’ will be shorter

Super Block

- Redundant duplication
- Reliable!!

Cylinder Group Structure



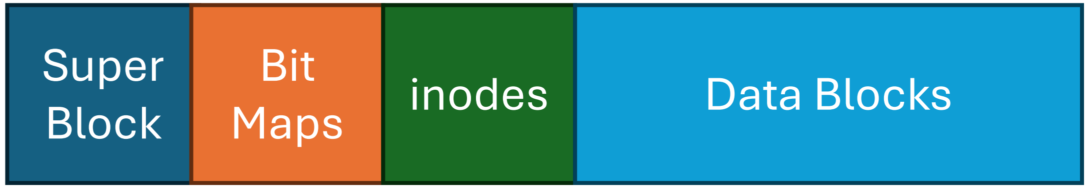
The Cunning Plan

- Groups
- If two files are in the same group, the time to access both files ‘together’ will be shorter

Bit Maps

- As per a normal FS
- Per-group maps

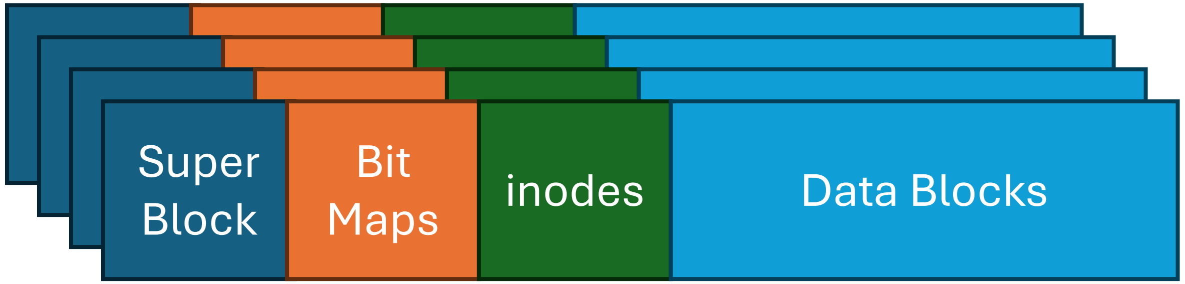
Cylinder Group Structure



The Cunning Plan

- Groups
- If two files are in the same group, the time to access both files ‘together’ will be shorter

Cylinder Group Structure



inodes

- Nothing new

Data Blocks

- Nothing new

Problems?

What is the issue with this?
What have we not done?

Problems?

How do we know??

What rules could we make??

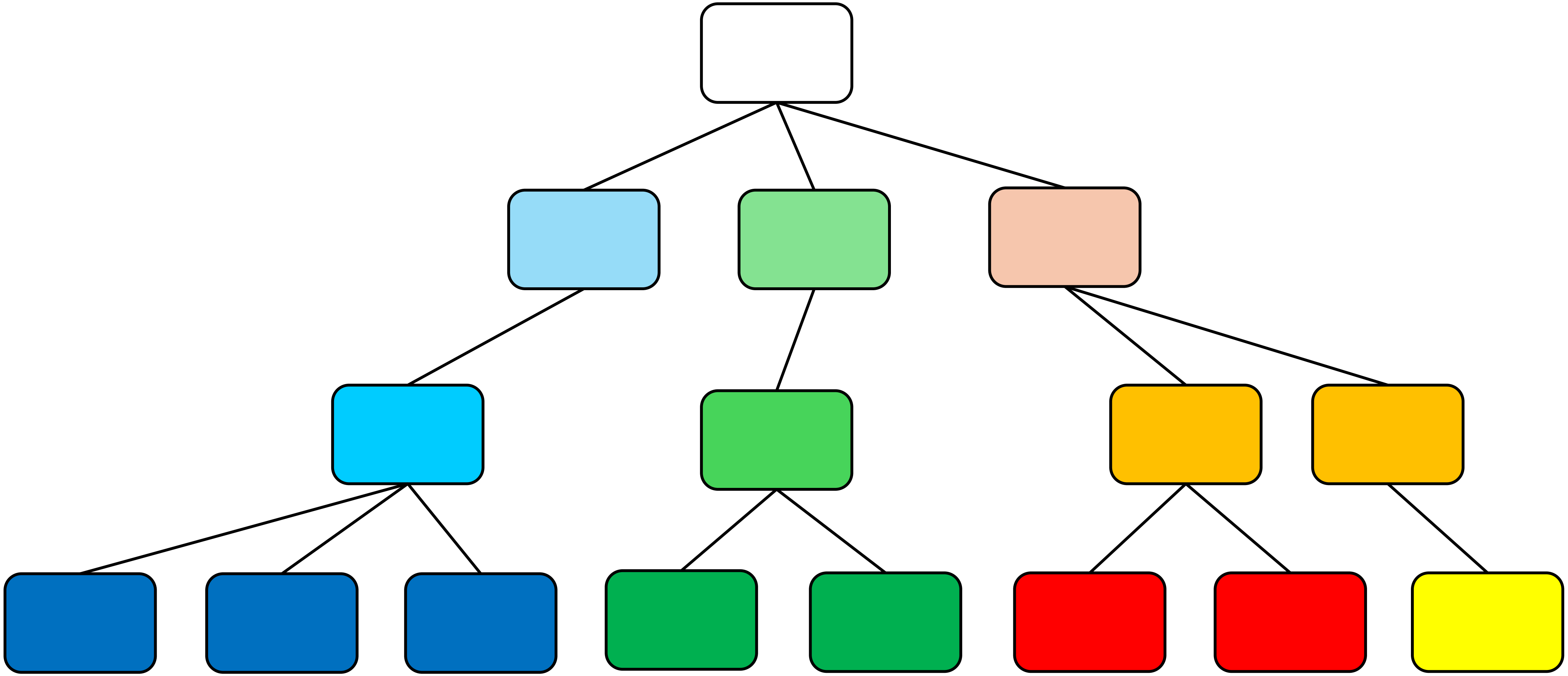
Easy Rules

#1: A **block** should be in the same **group** as its **inode**.

#2: A **file** should be **grouped** with all files in the same **directory**

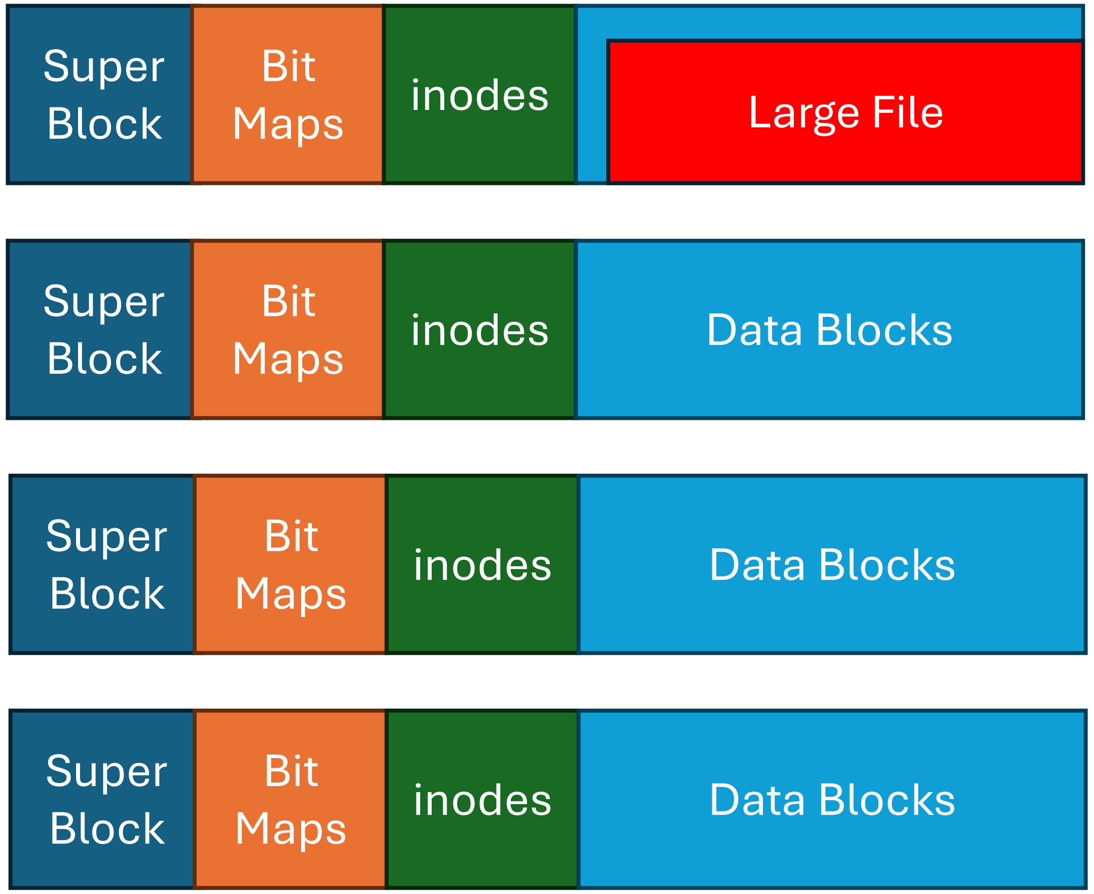
#3: A **file** should be **grouped** differently from files it doesn’t share a parent/grandparent with

Visually



Large Files

Large Files will fill blocks and thus ‘defeat’ locality ☹

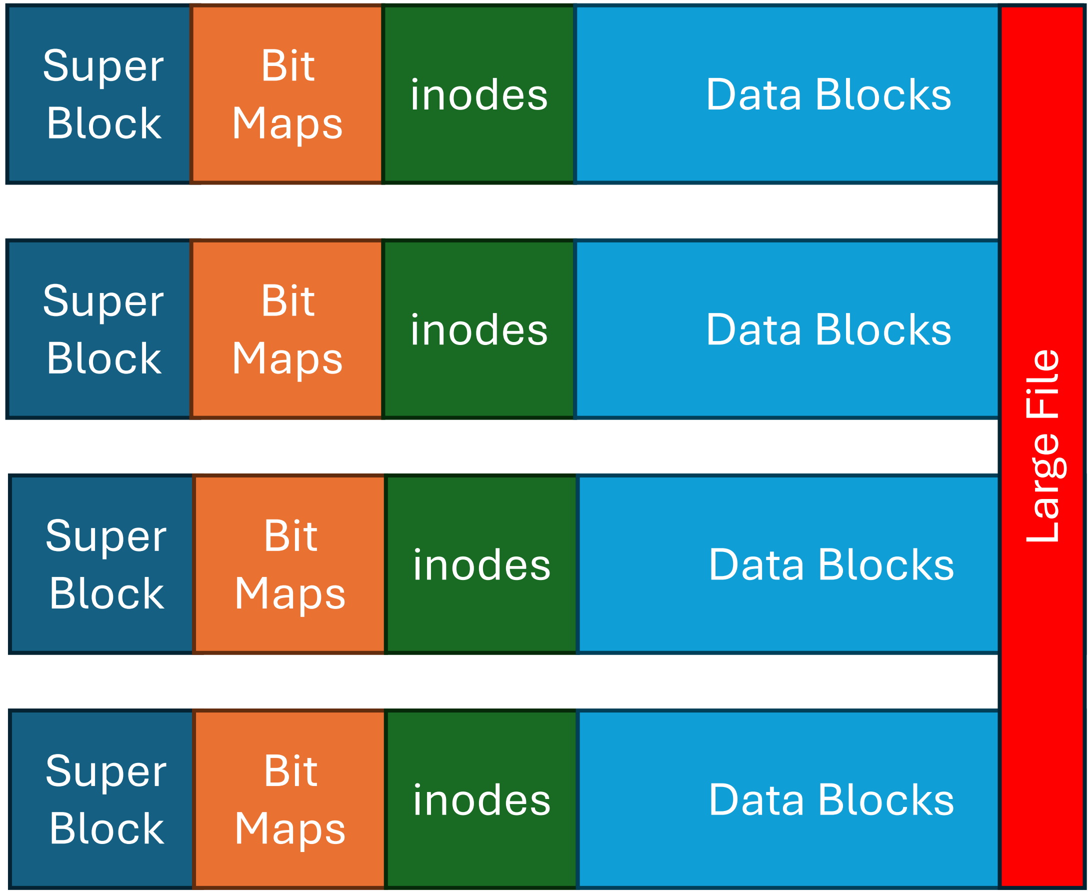


Large Files

Distributed Large Files is good

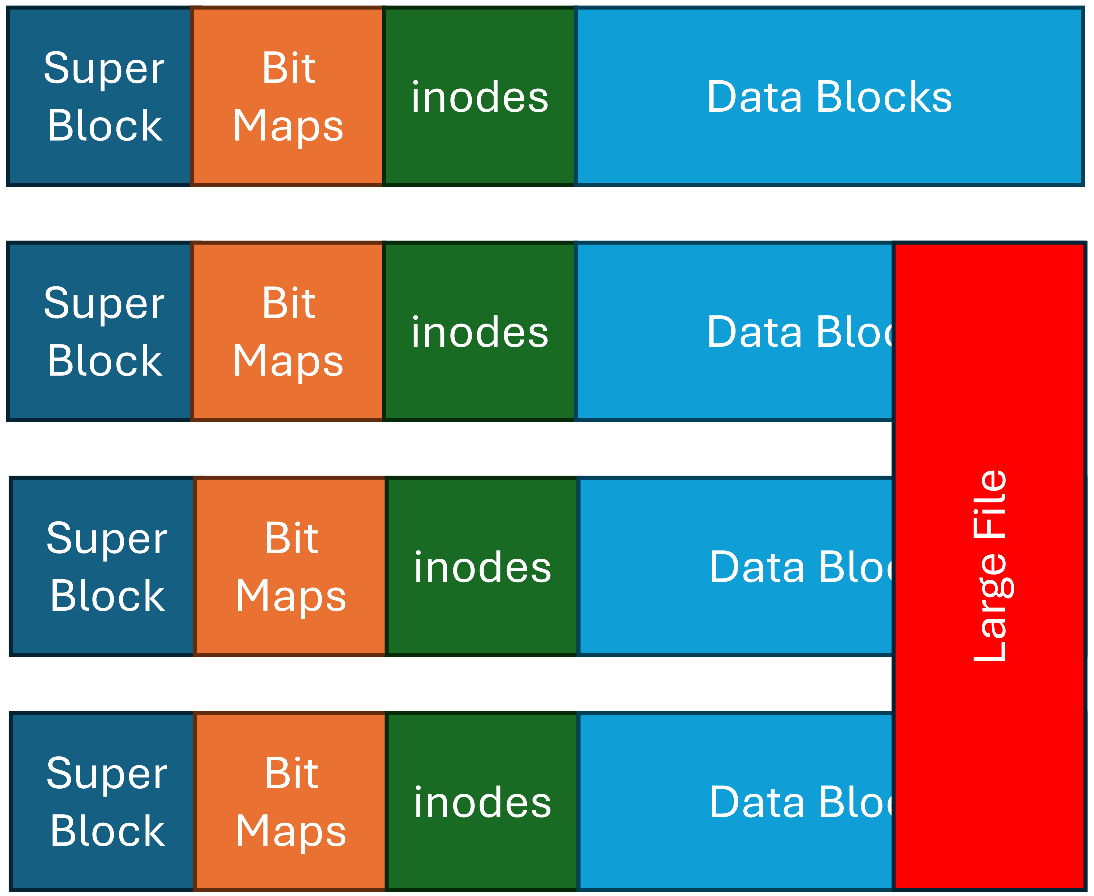
Except...

Why not?

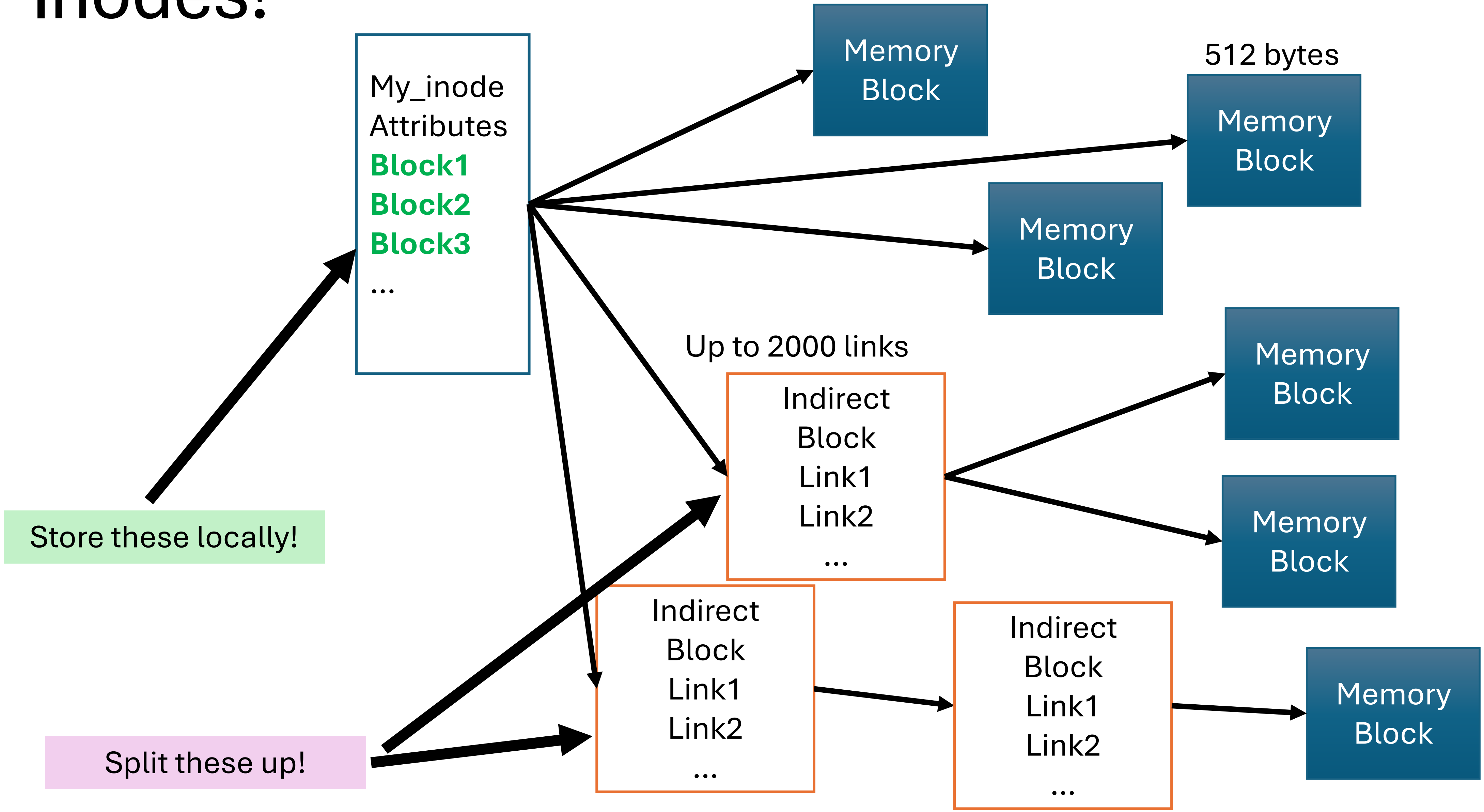


Large Files

Fix it so the data transfer and data seek are ~ the same



Inodes!



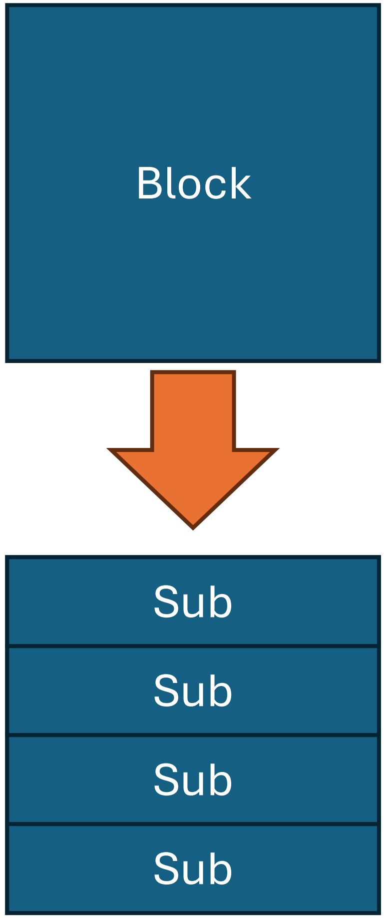
My_inode
Attributes
Block1
Block2
Block3
...
Memory Block
512 bytes
Memory Block
Memory Block
Up to 2000 links
Indirect Block
Link1
Link2
...
Memory Block
Memory Block
Store these locally!
Indirect Block
Link1
Link2
...
Indirect Block
Link1
Link2
...
Memory Block
Split these up!

Fragmentation

If blocks are 4KB…

How big are files… actually?

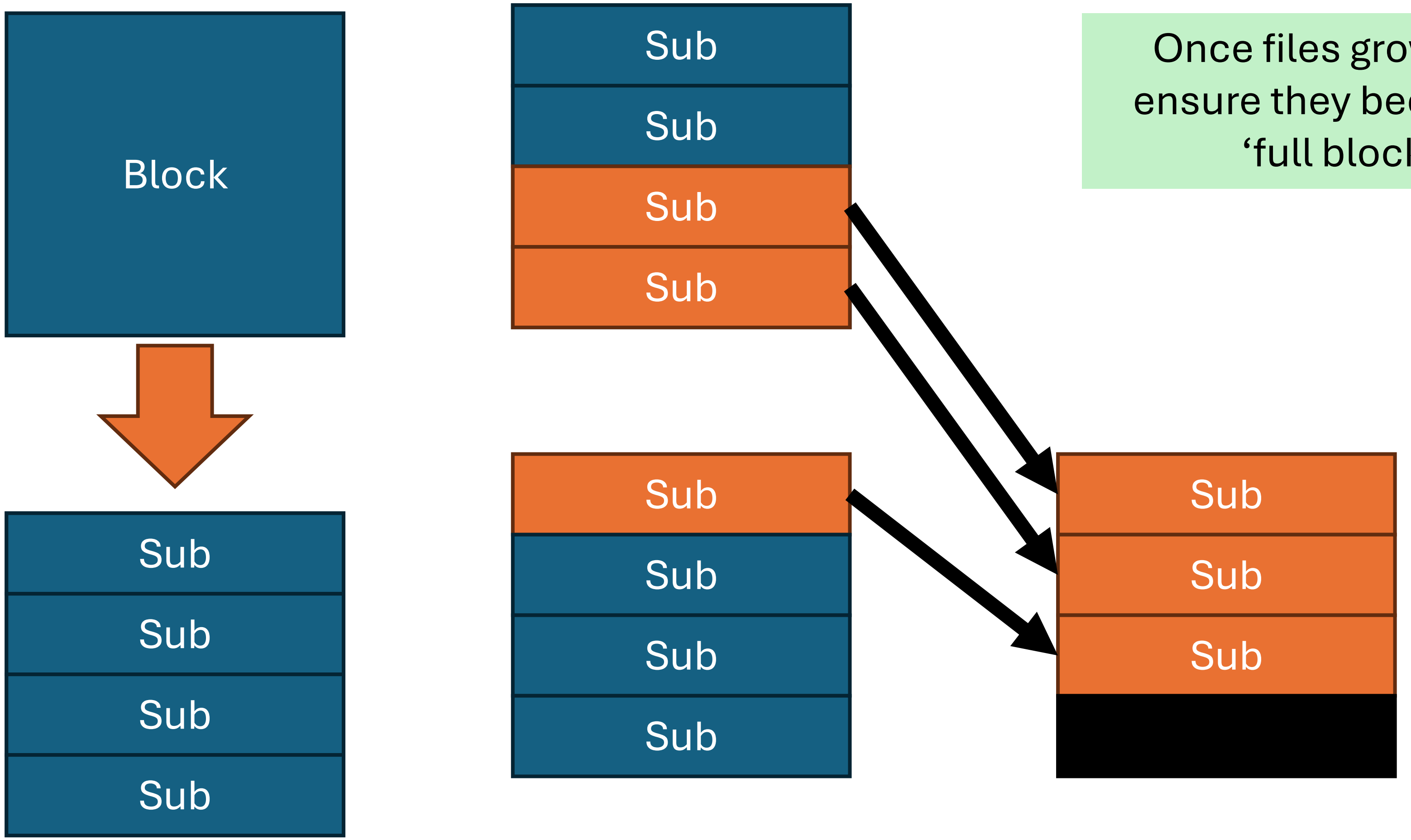
Sub-blocks (Fragments)



Problems

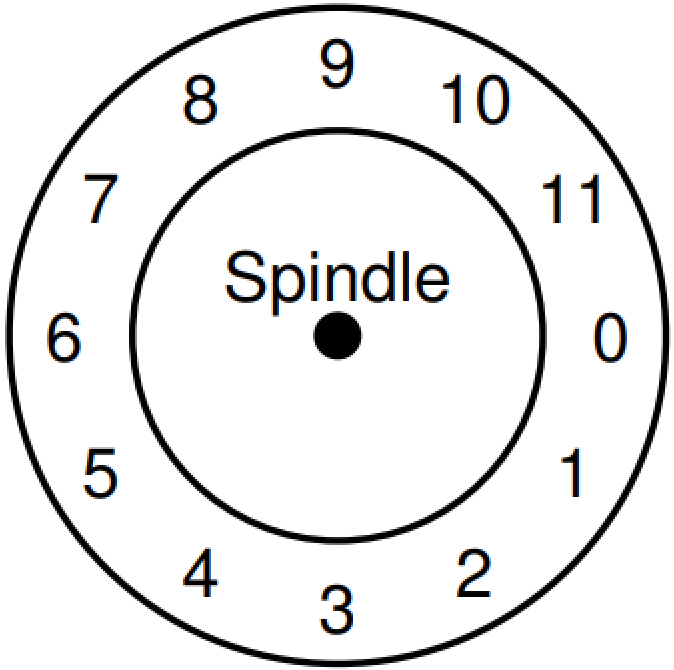
- Feels inefficient
  - Lots of ‘little bits’
  - Solution is buffering writes
- Growing Files?

Sub-blocks (Fragments)

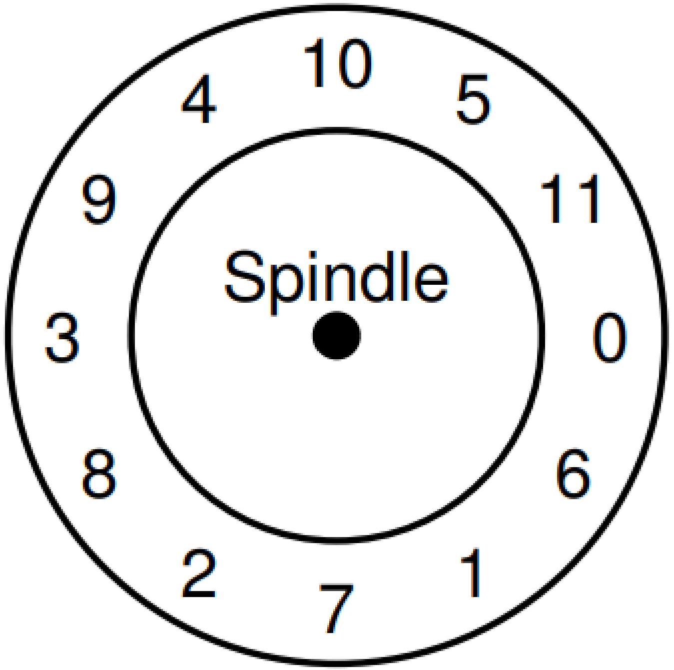


Once files grow, you ensure they become a 'full block'

Memory Layout



This is bad



This is better?

Memory Layout

Layout Optimisation

- Parameterization!
  - Determine layout based on disk performance

Buffering

- Avoids missing data

File Names

Fixed Length

- Early OS's had a fixed length file name (a bit unsurprising)

FFS:

- Added a length field
- Directory block steals bytes from short names to store long names

Atomic Renaming

- Process
  - Lock the parents directories (new & old)
  - Check constraints
  - Update directory entries
  - Commit the change (both changes hit simultaneously)

FFS Summary

Summary:

- Divide into groups
- Bunch common files (same directory)
- Split up big files (via inodes)
- Allows long filenames
- Added symbolic links

History:

- Basis for modern files systems (ext2, ext 3)

Crash Consistency

Oh dear

Crash Consistency

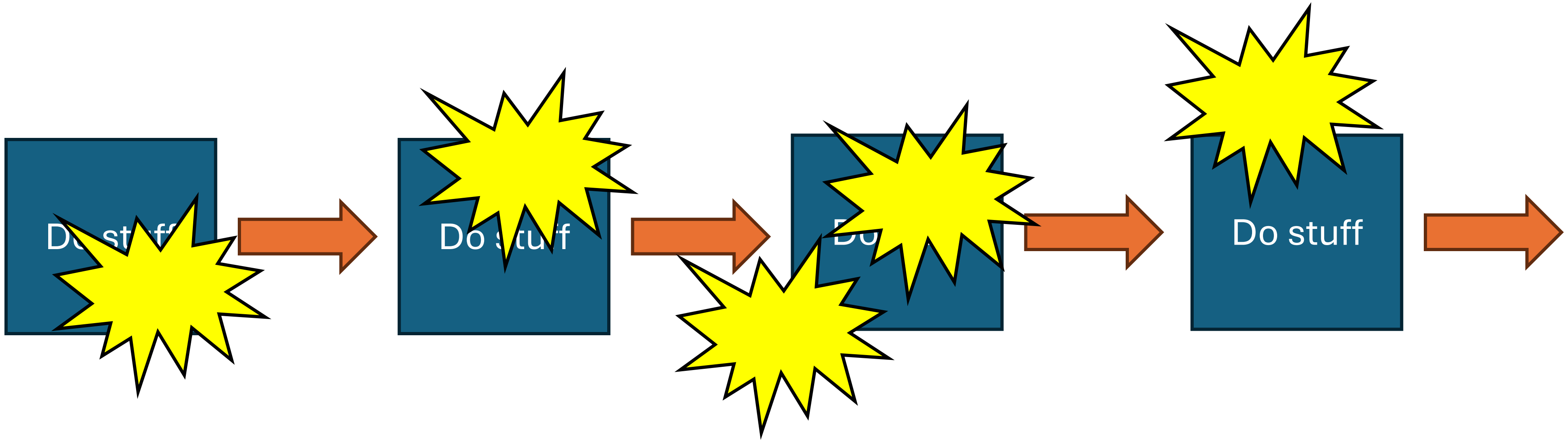
The Theory:

- Data stored on a disk, must be persistent

The Reality:

- Systems that run on electricity can stop working randomly

The Problem



Our methodology must account for sudden stops

Old Methods

Methods

- **fsck** (File system checking)
- **Journaling**

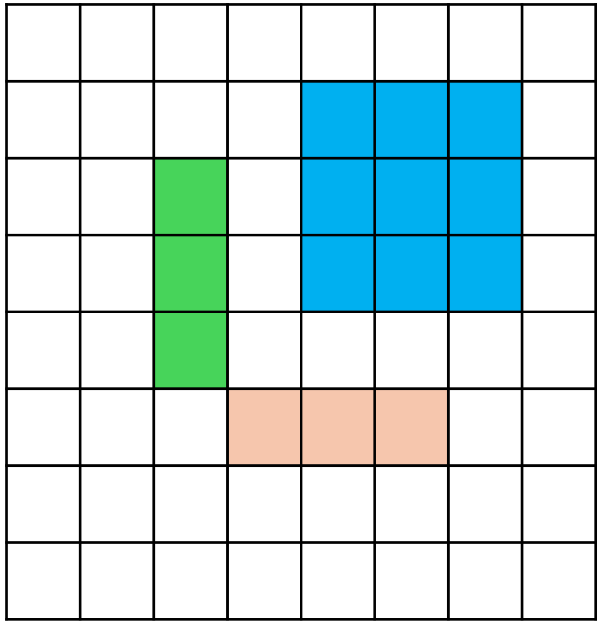
RAID

Backup

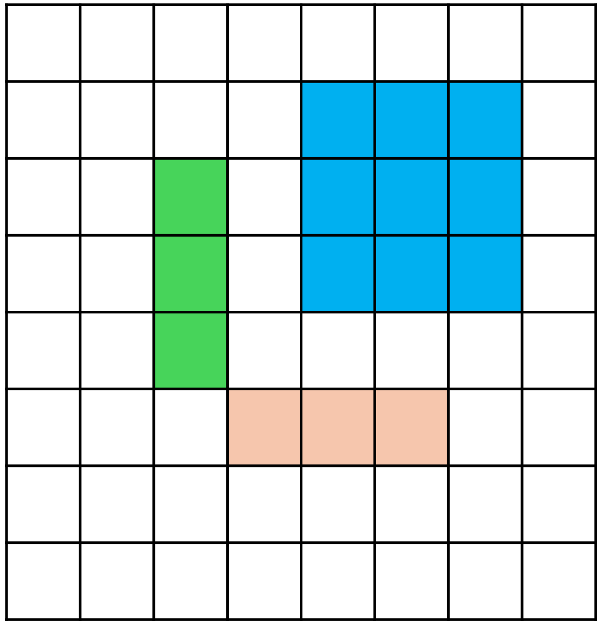
- Mirror, Parity (RAID 1, RAID 4/5)

- Does not provide protection from power outages, only hard drive failure

Consistency

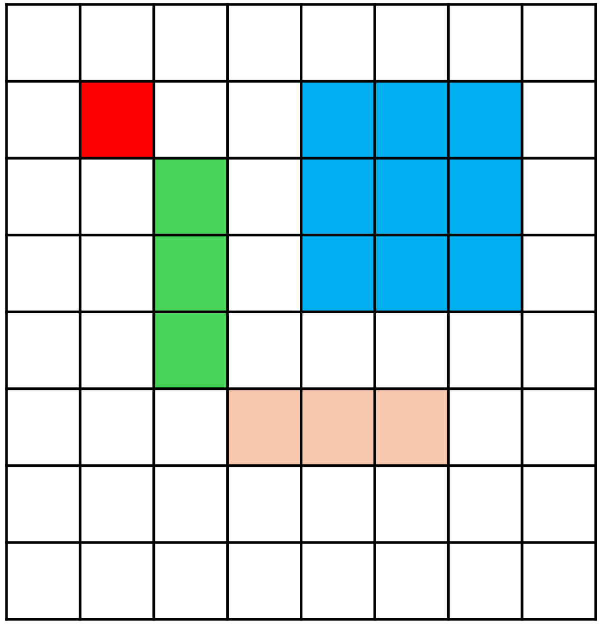


Bitmap

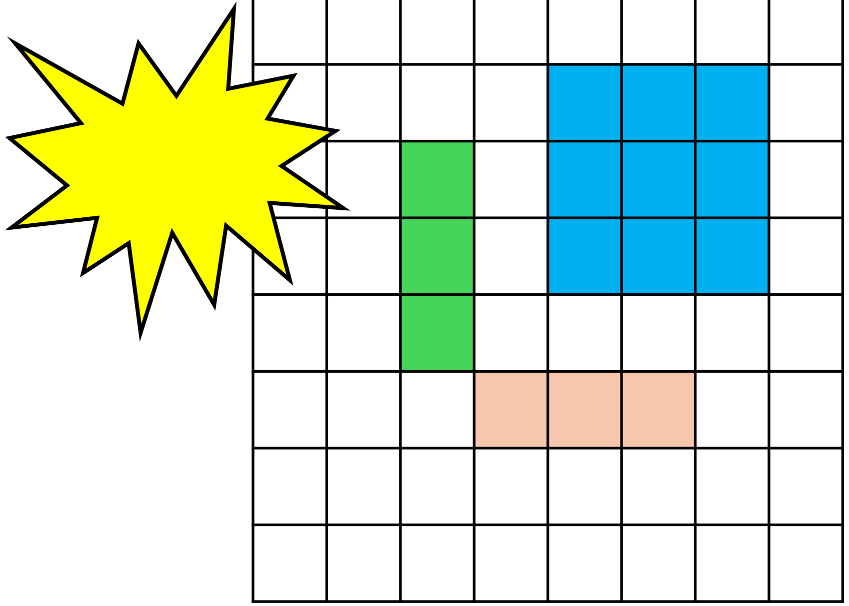


Actual Data

Consistency

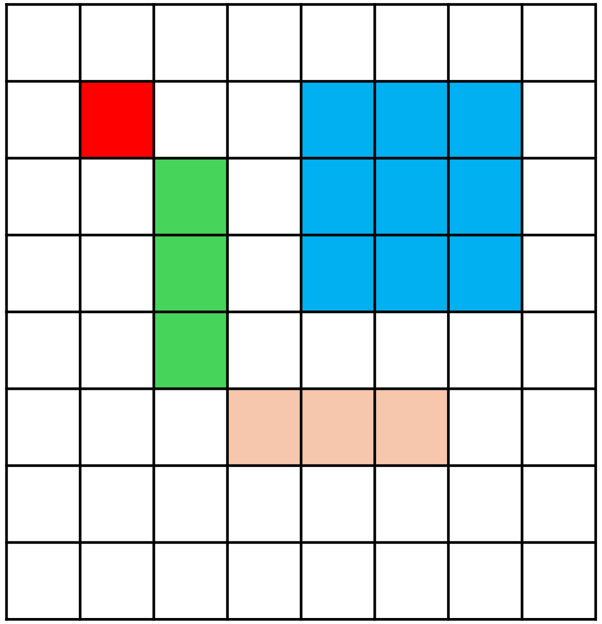


Bitmap

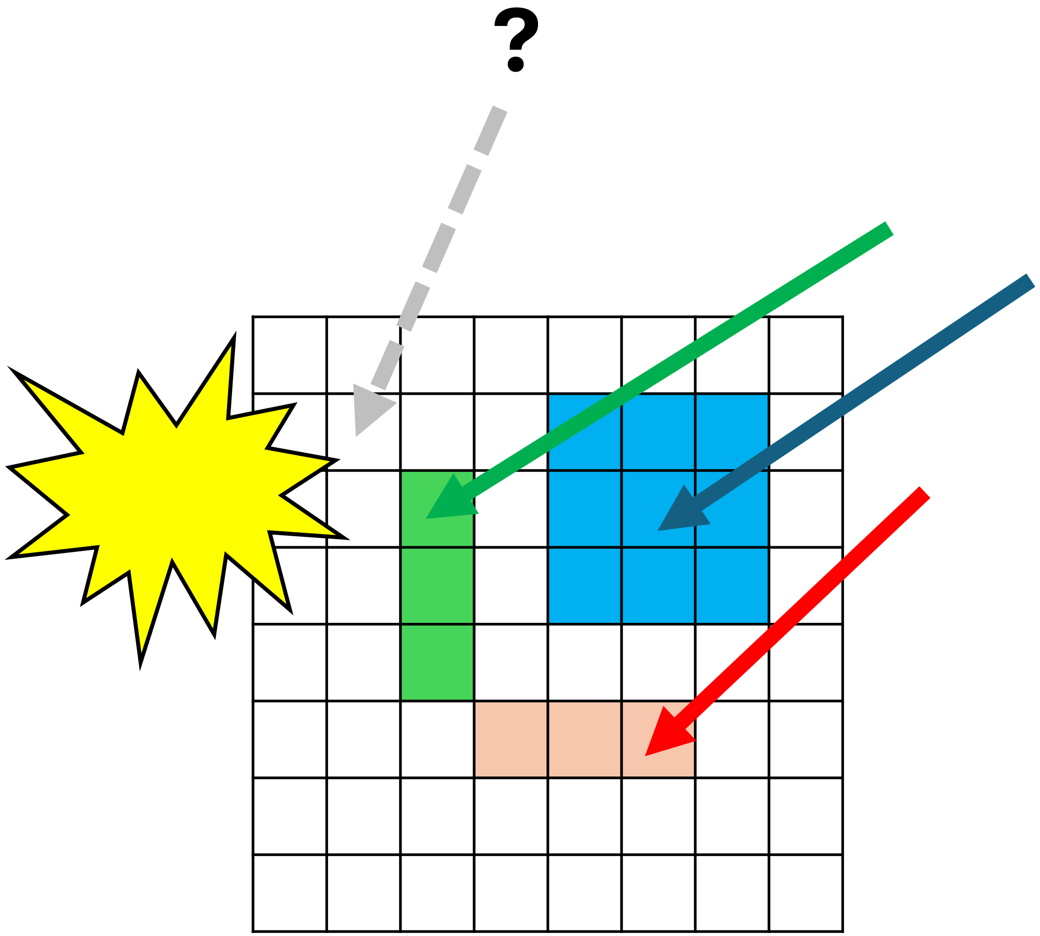


Actual Data

Consistency

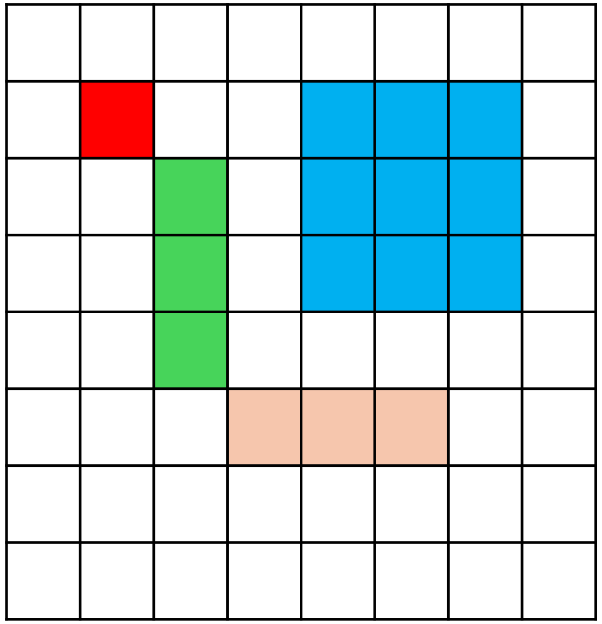


Bitmap

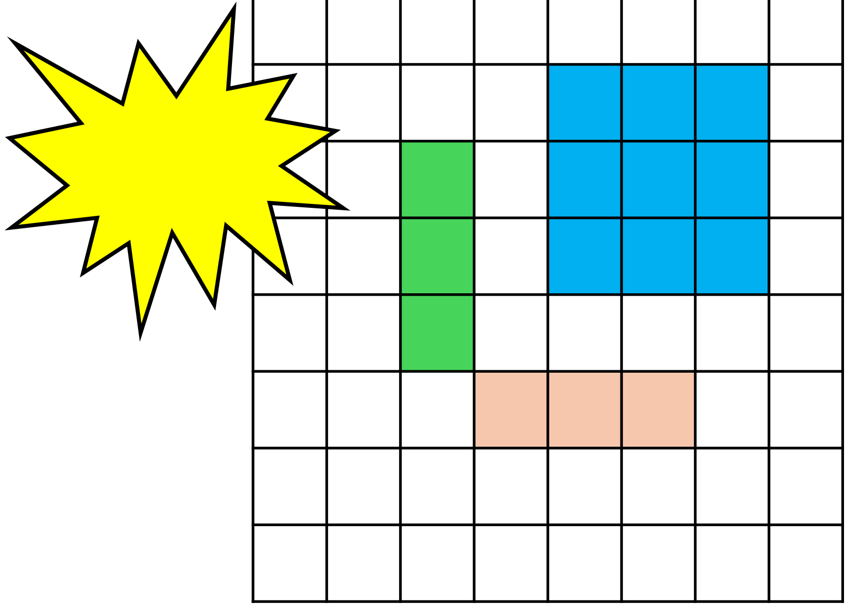


Inodes

Consistency



RAID Drive 1



RAID DRIVE 5

RAID can create new inconsistency risks

Data Update

What must be done

- Update the bitmap
- Update the inodes
- Update the data

What could happen

- **Only** bitmap
- **Only** inodes
- **Only** data
- bitmap & inodes
- bitmap & data
- inodes & data

Data Update

What could happen

• **Only** bitmap	‘lost block on disk’
• **Only** inodes	“nothing bad”
• **Only** data	point to garbage, another file may overwrite
• bitmap & inodes	‘lost block’
• bitmap & data	point to garbage
• inodes & data	another file may overwrite

File System Checker (FSCK)

Strategy:

- After crash, scan whole disk for contradictions
- Fix
- Keep file system “off-line” until FSCK completes

To do this you need to flag “not a crash”

File System Checker (FSCK)

Strategy:

- After crash, scan whole disk for contradictions
- **Fix?**
- Keep file system “off-line” until FSCK completes

To do this you need to flag “not a crash”

What can we detect?

What can we check/fix?

We can check:

- Superblock
  - Free space?
- Inodes
  - Can I get to here from root?
  - Is this block in the bitmap?
  - st_nlink (we can count this)
- Directories
  - Do these point to sensible places?

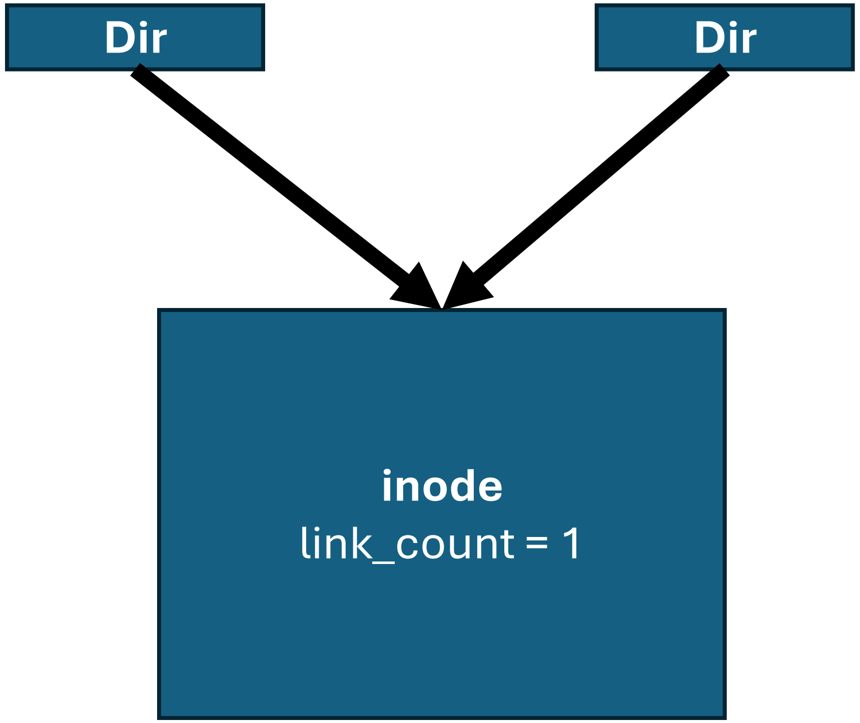
We can also check:

- ./..

What does fix mean?

What is wrong?

- link_count is wrong



Fix:

- Update link count to 2!

What does fix mean?

What is wrong?

- Should it be in a directory?
- Should we just delete it?

inode
link_count = 1

But no actual links

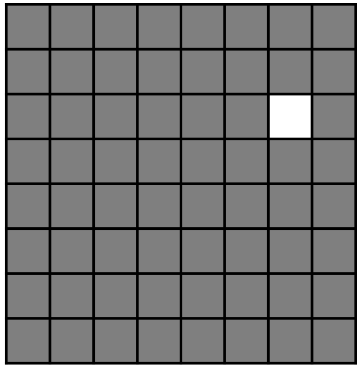
Fix:

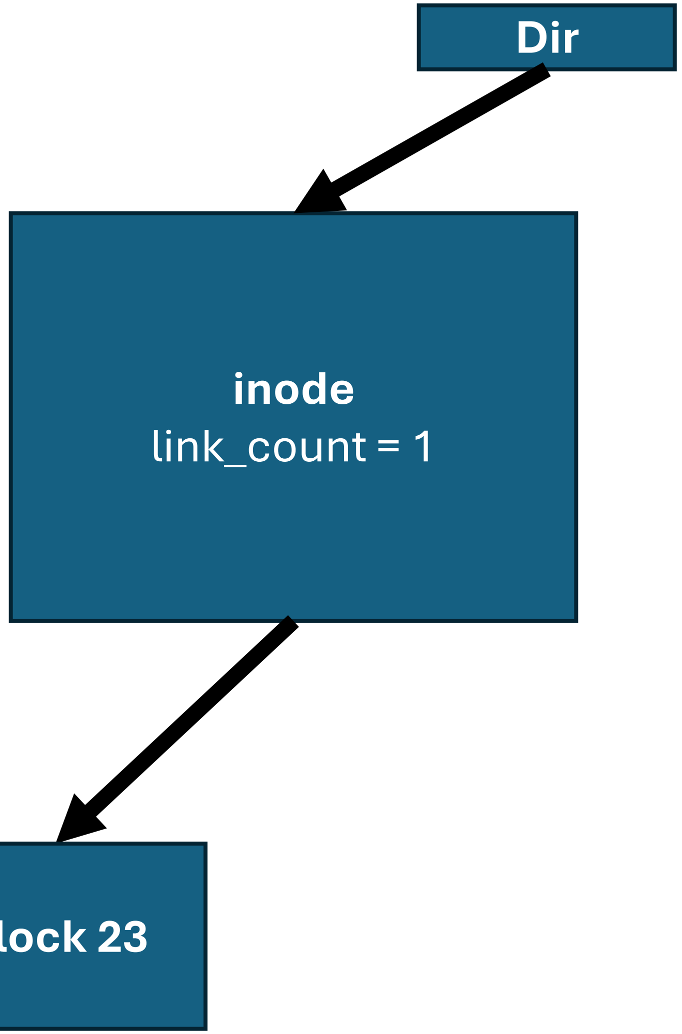
- Put in lost and found directory

What does fix mean?

What is wrong?

- Bitmap doesn’t match data





Fix:

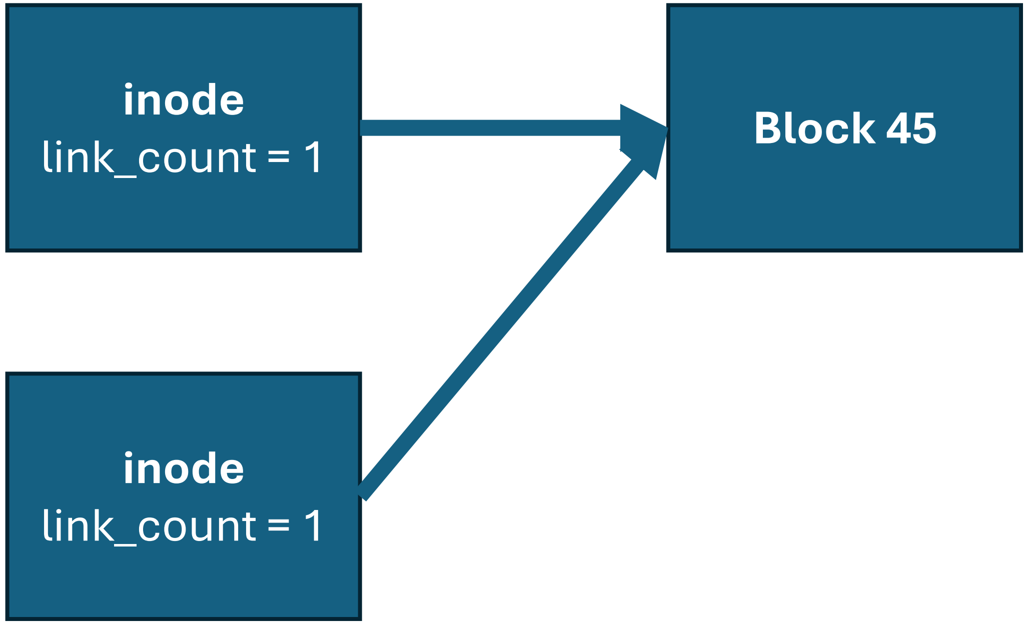
- Update bitmap to ‘1’ from ‘0’

What does fix mean?

What is wrong?

- To inodes pointing to one piece of data

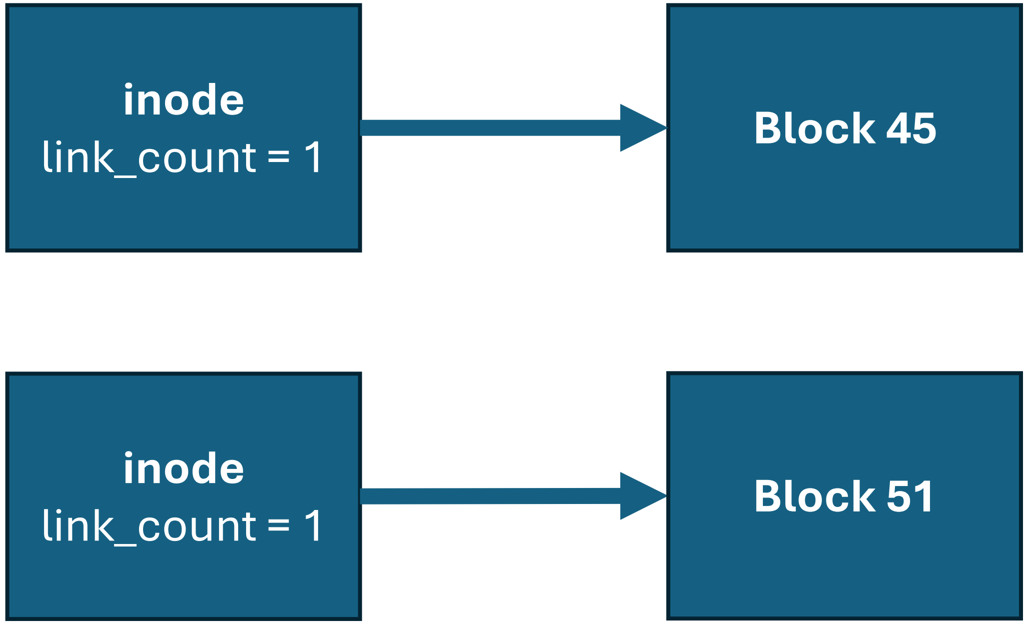
Fix:



What does fix mean?

What is wrong?

- To inodes pointing to one piece of data



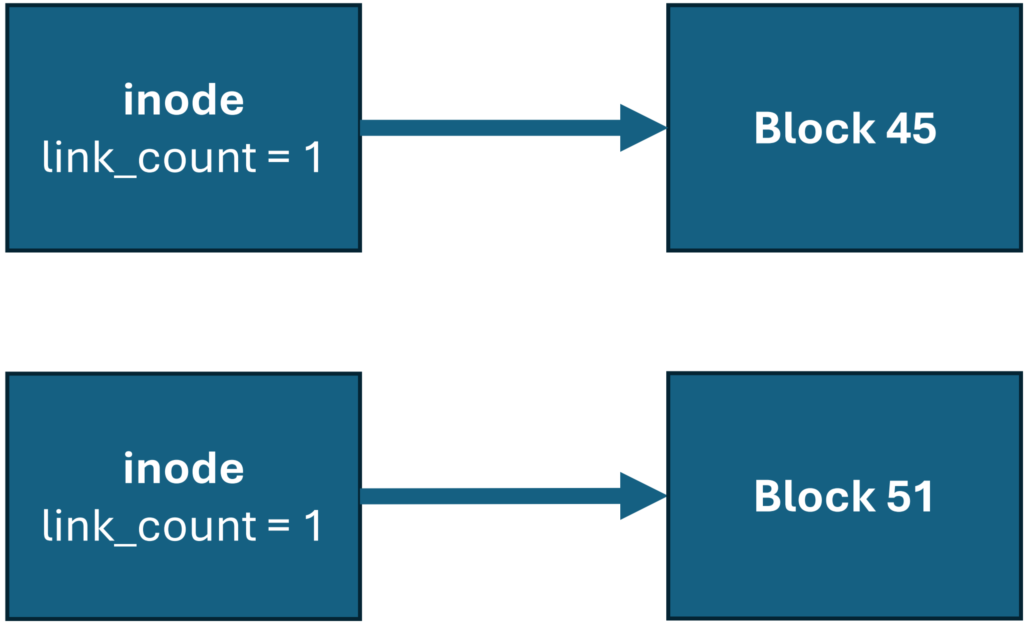
Fix:

- Point inode at its own new block

What does fix mean?

What is wrong?

- To inodes pointing to one piece of data



Fix:

- Point inode at its own new block

Wait a minute…

Is that good enough? Is it just pointing somewhere stupid now?

What does fix mean?

What is wrong?

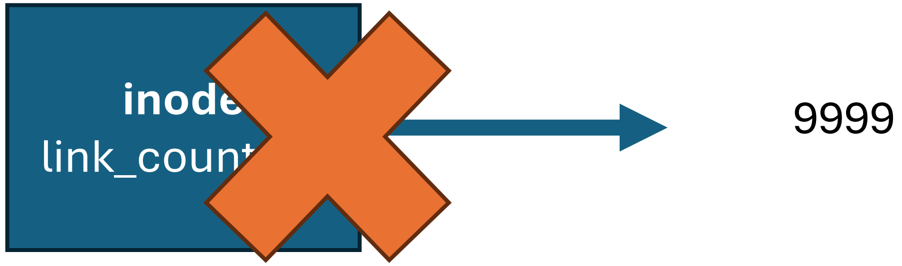
- Bad Pointer

Fix:

What does fix mean?

What is wrong?

- Bad Pointer



Fix:

FSCK Issues

Consistency is easy

... just format the drive

Problems:

- Not always obvious what the 'fix' should be
- Can only achieve **consistency**, not **correctness**
- It is **slow**

Journalling

How to do recovery

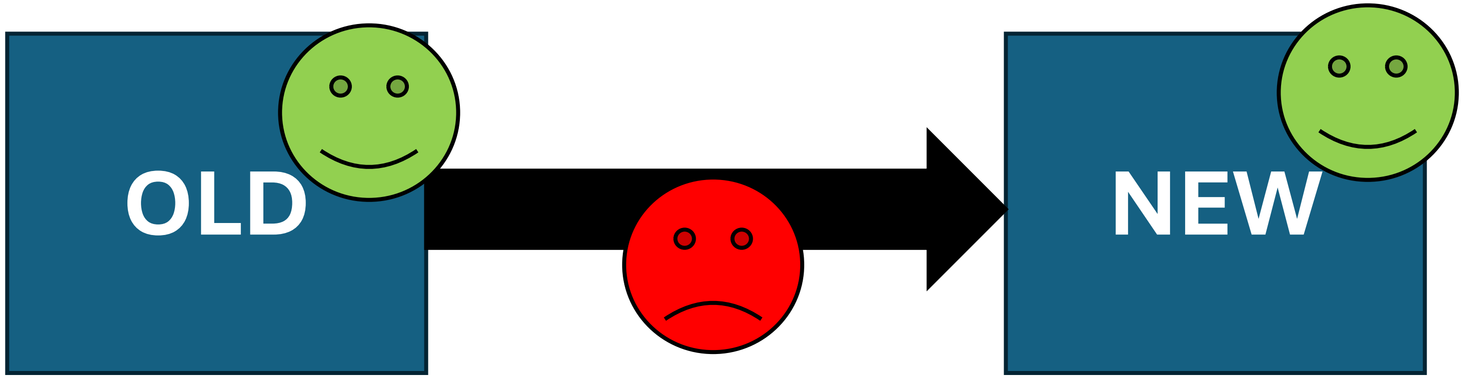
Journalling

The Goal

- Perform **recovery** but without reading a whole disk
- Attain a **correct** state

The Strategy

- Atomicity
- Definition of atomicity for **concurrency**



Journaling Strategy

What are some general rules we might employ?

Journaling General Strategy

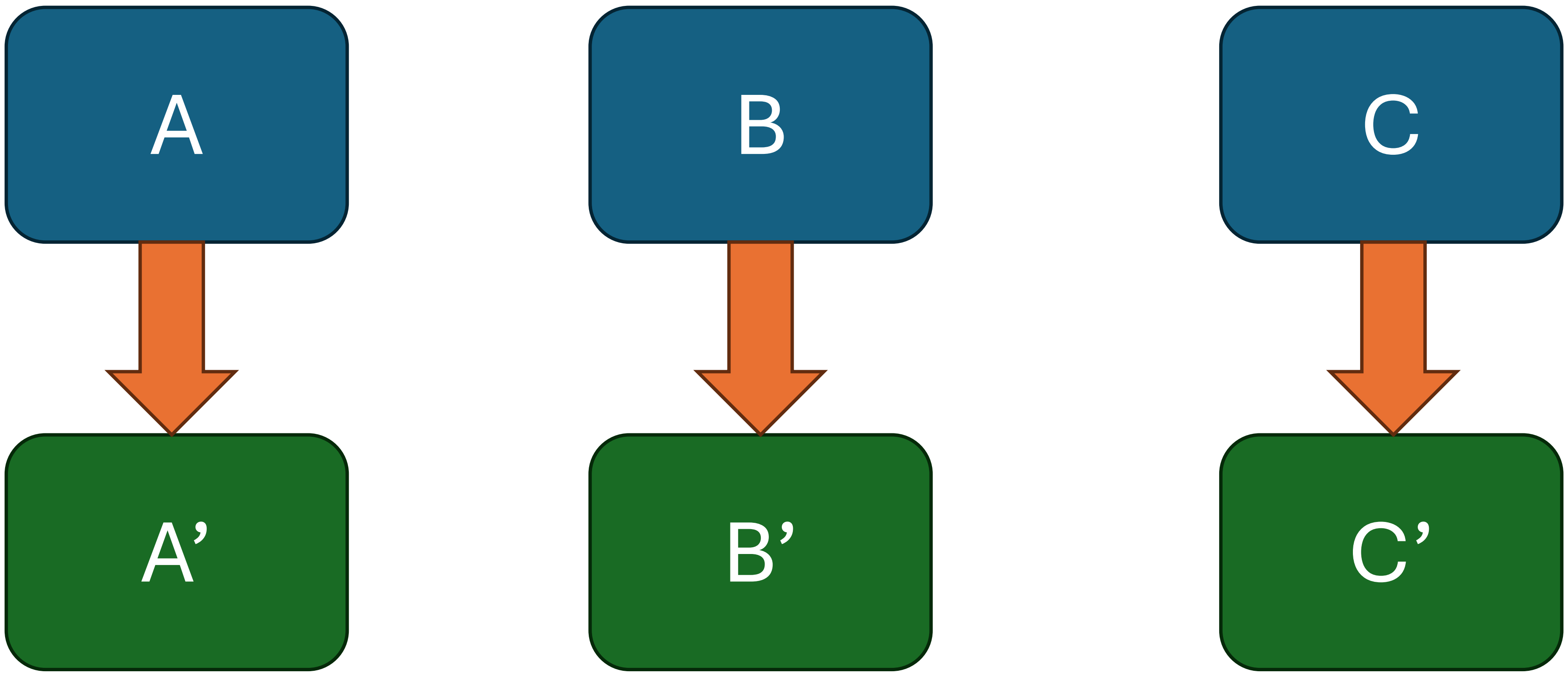
Rule #1

- Never delete ANY old data, until ALL new data is safely on disk

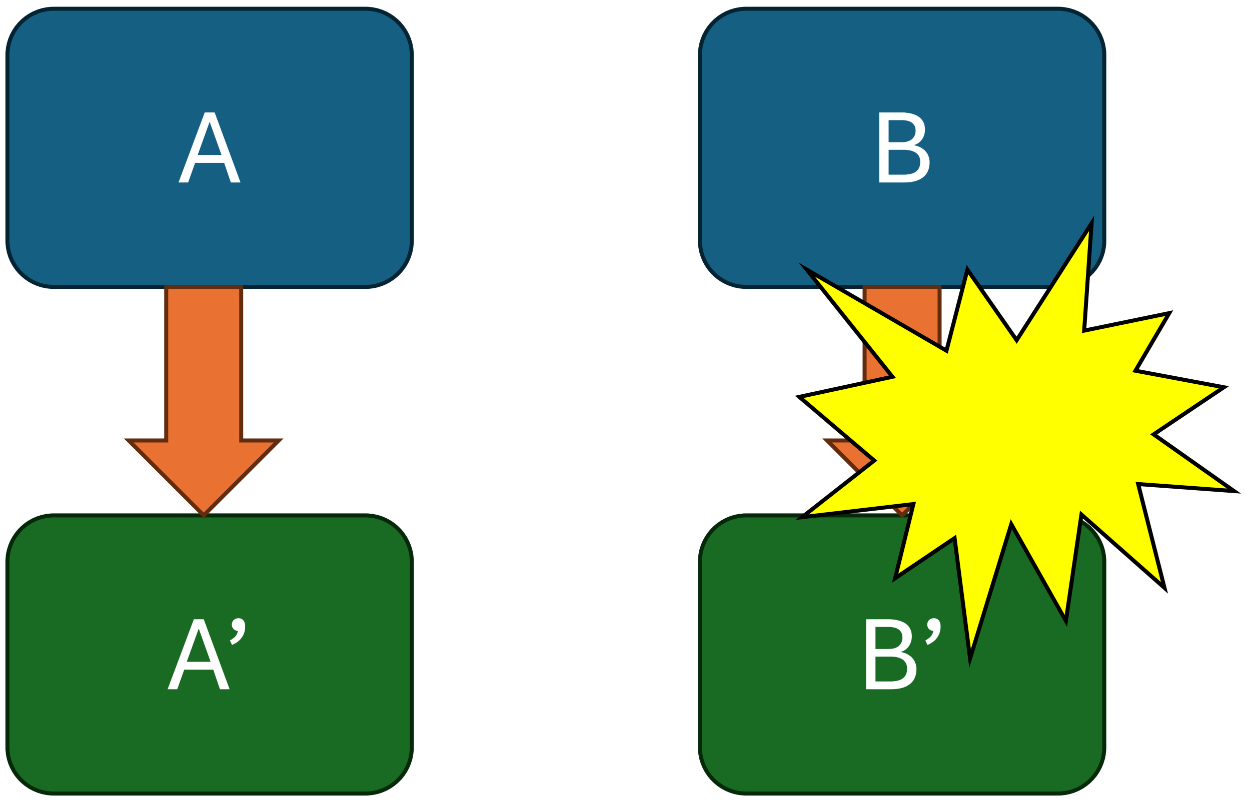
Rule #2

- List what you want to do before you do it

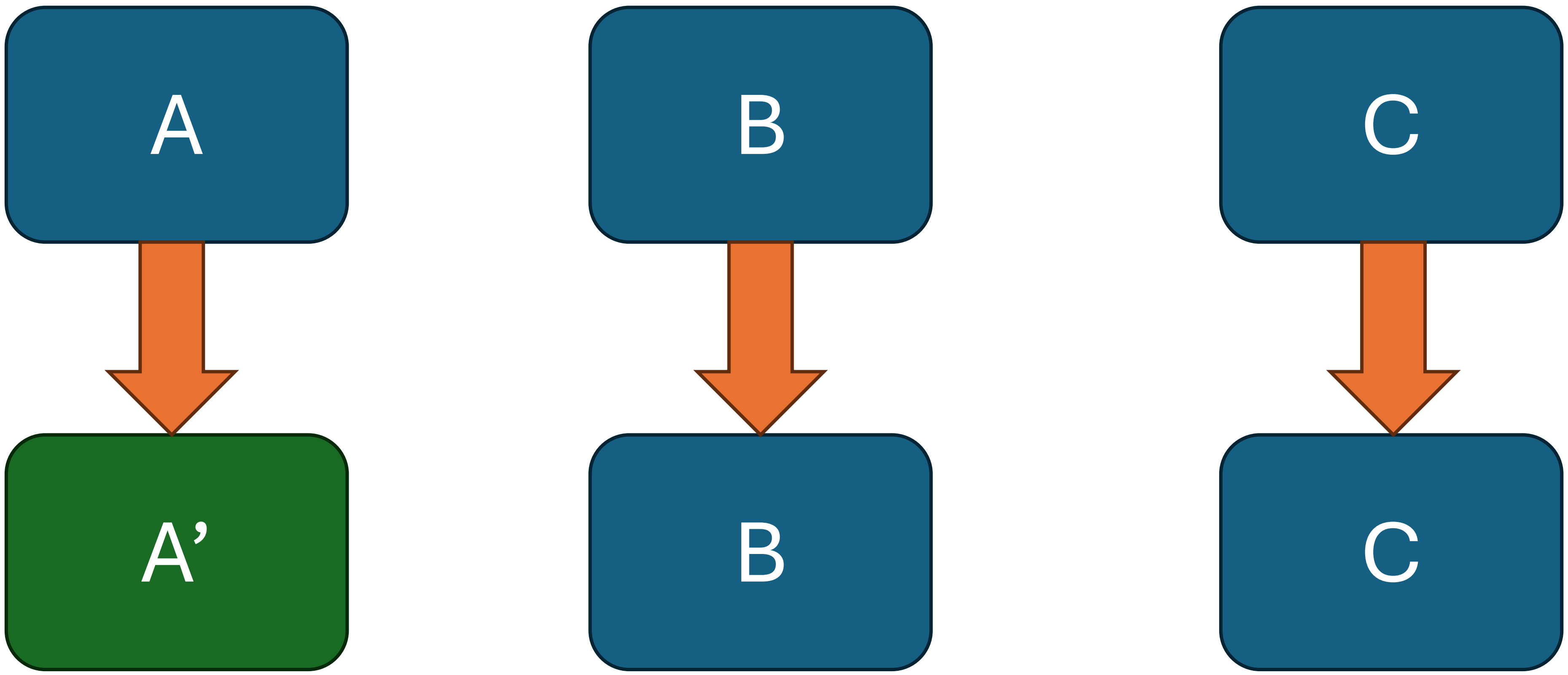
Example



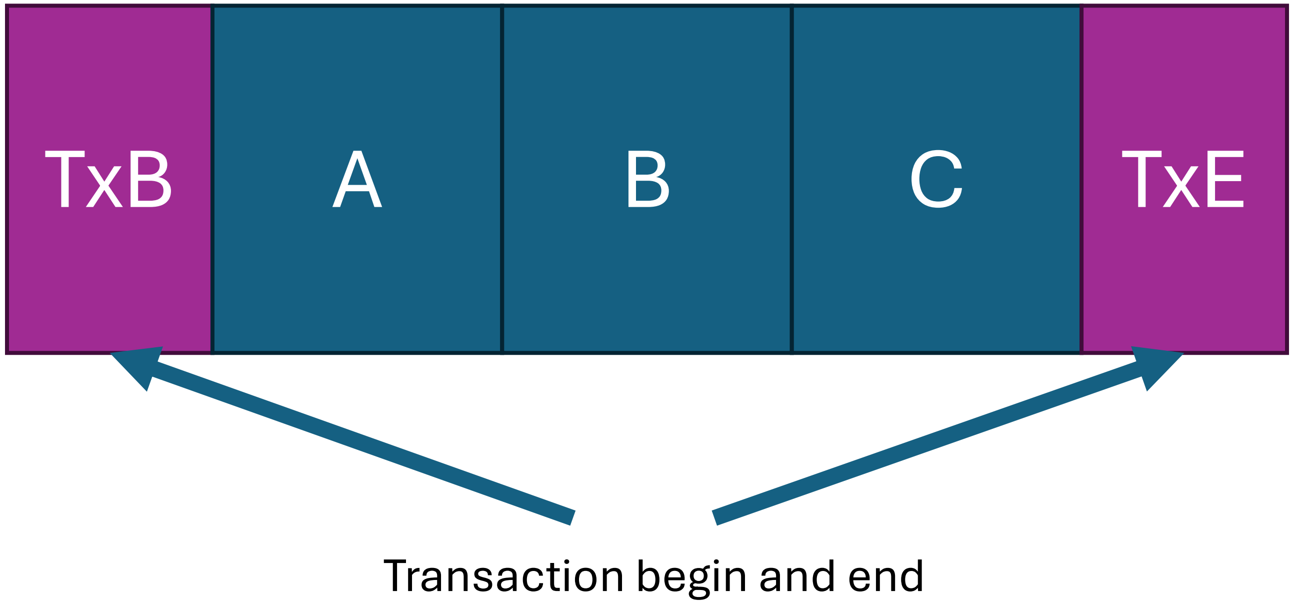
Example



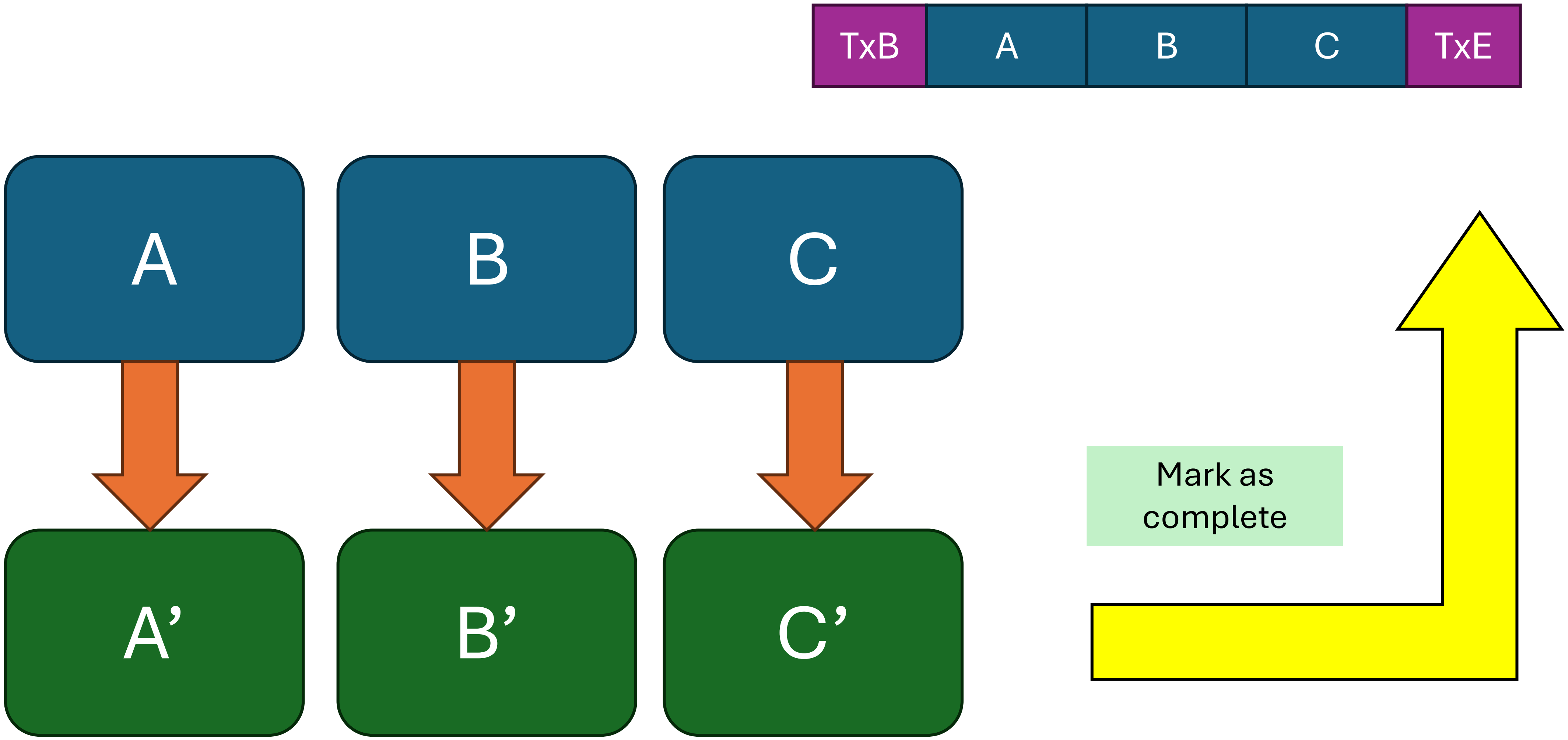
Example



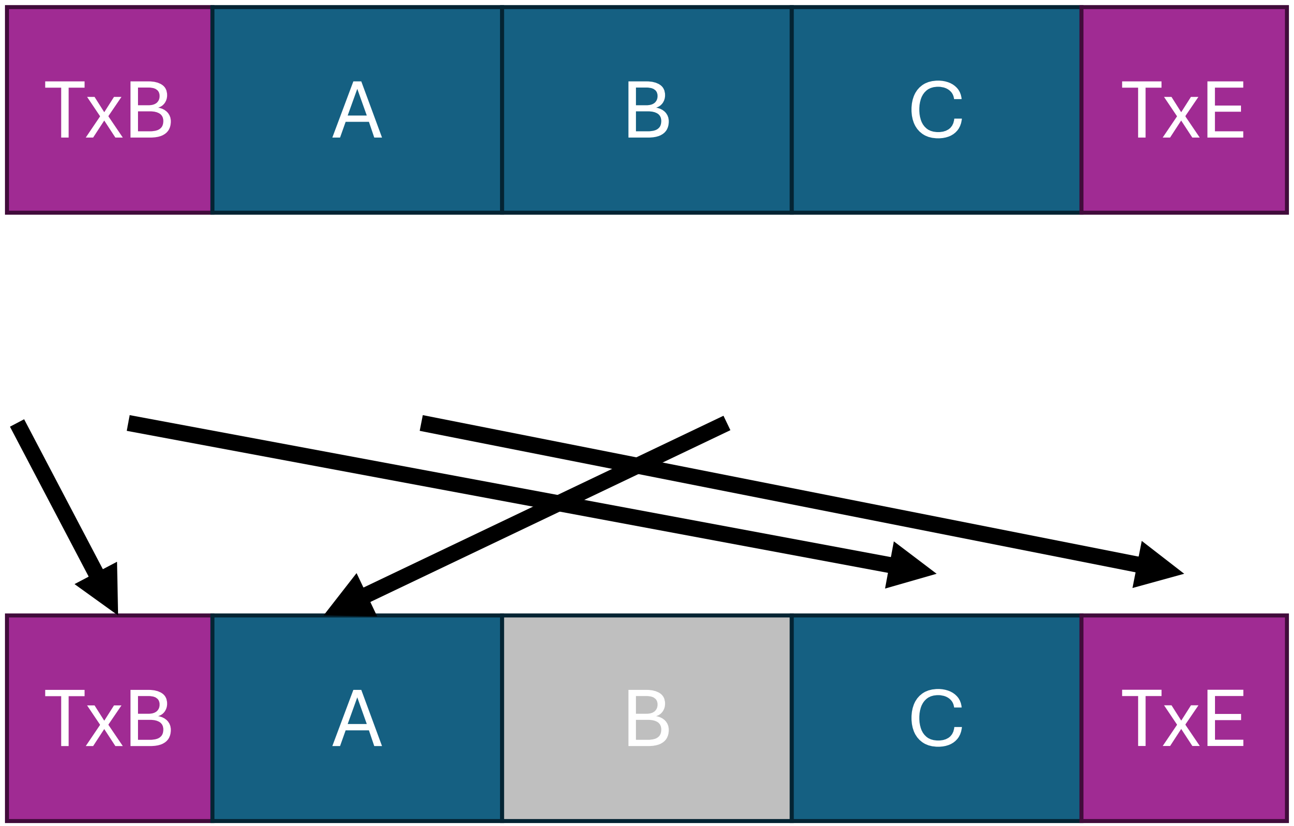
Transaction



Checkpointing

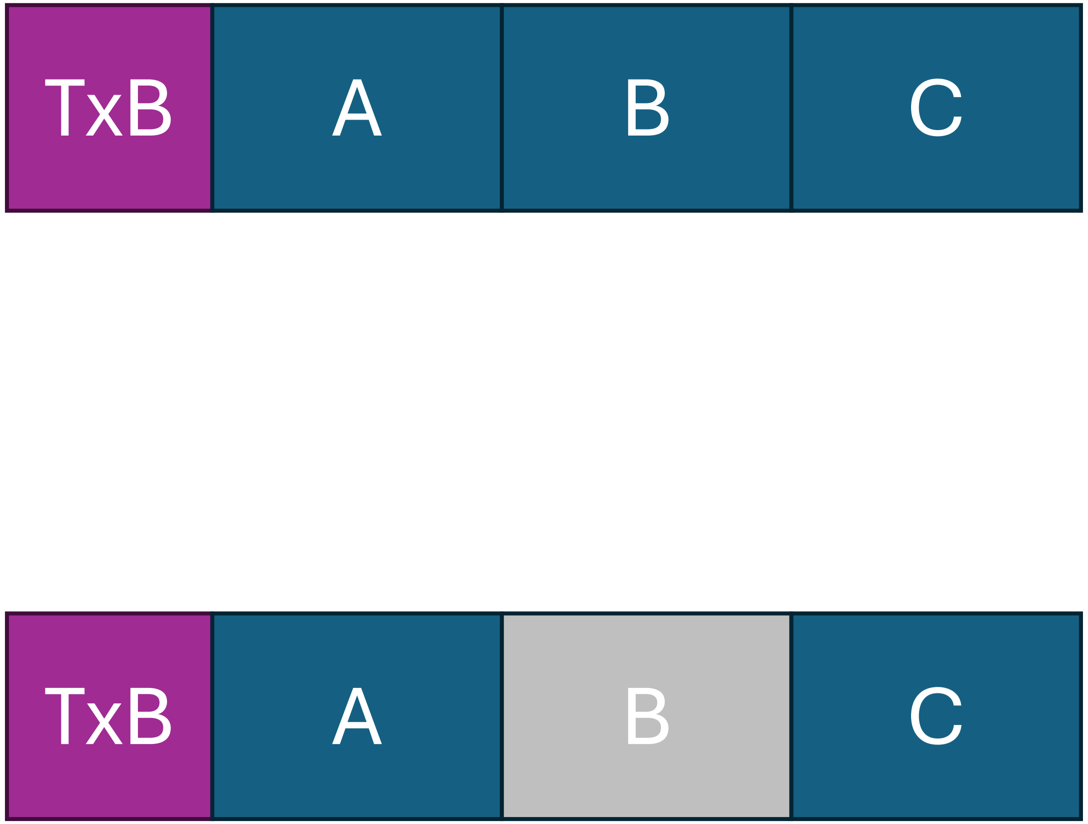


Writing Transactions



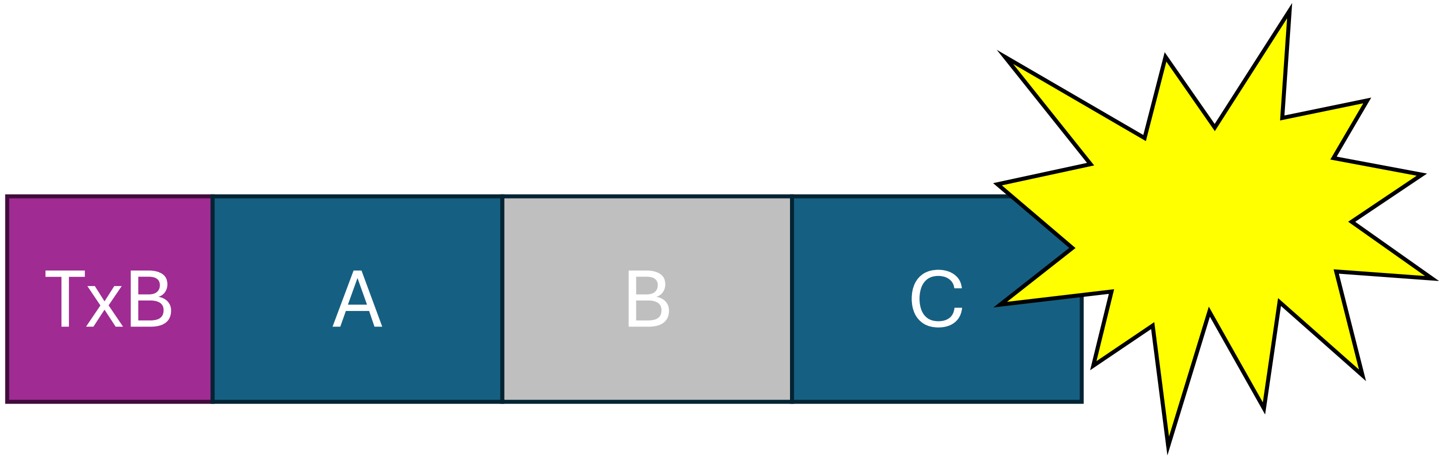
Hard drives are annoying and can re-order writing things...

Writing Transactions



Hard drives are annoying and can re-order writing things...

Writing Transactions



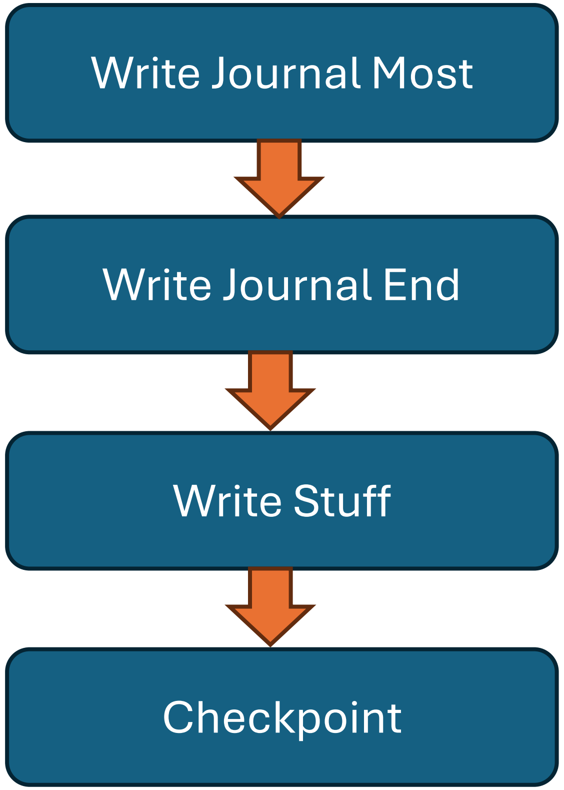
We separate it into 'most' + tail

Journal Location

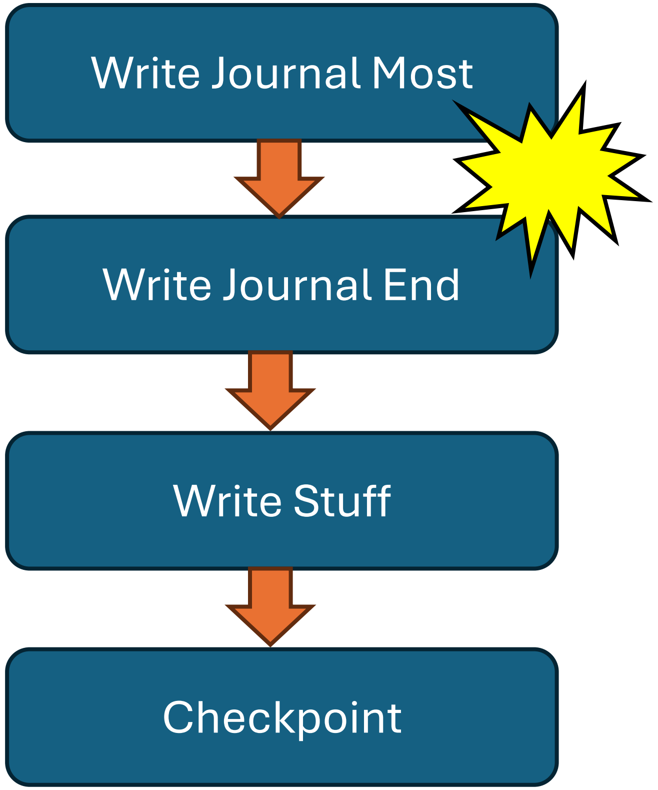
Recovery

What to do in event of a crash?

Recovery

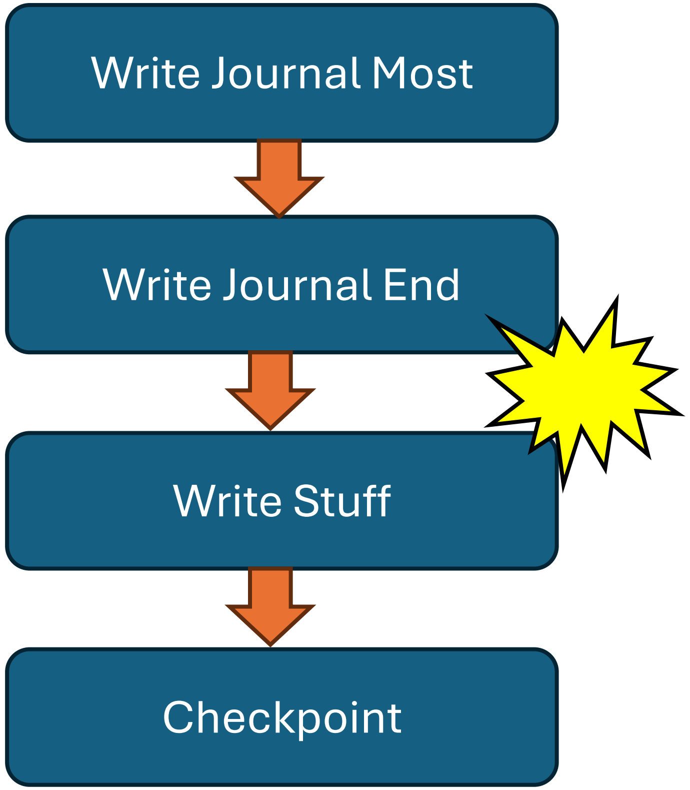

Write Journal Most
Write Journal End
Write Stuff
Checkpoint

Recovery



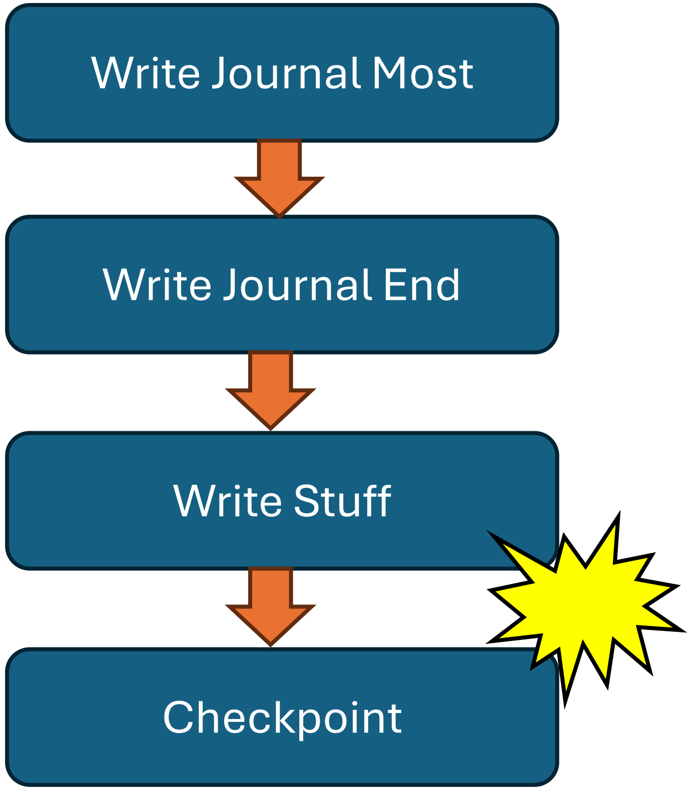
We can just discard the journal entry

Recovery



We can replay the updates (on boot)

Recovery



We can replay the updates (on boot)

Even though we've done them already

Problems with Journaling

What is the problem with journaling?

Data Journaling

Description

- Everything (including data) is written to the journal

Advantages

- Easy Recovery, Maximum Consistency

Disadvantages

- Big Overhead, Slow

Ordered Journaling

Description

- Only metadata is journaled, data is written first

Advantages

- Consistent, Faster

Disadvantages

- Can lose ‘new data’ (sometimes in lost & found)

Writeback Journaling

Description

- Only metadata is journaled, no ordering (optimised for speed)

Advantages

- Fastest, Consistent

Disadvantages

- Can result in ‘corrupted/nonsense’ data

Journal Considerations

HDD

- Journal needs to be fast

SSD

- Wear leveling?? Journal will get a big workout.

Journal Considerations

Write Buffering Optimisations

Use a circular buffer

Problem

- Many small journal updates

Solution

- Collect in a buffer and write in batches

What happens if you crash before flushing the buffer?

Journal Considerations

Checksum Optimisation

Problem

- What if the journal itself got corrupted?

Solution

- Add a checksum (computation based on contents) to check for consistency (sometimes to both ends – to ensure consistency)

Logical Journal

Problem

- Journalling all the required changes to files can be ‘big’ in terms of space

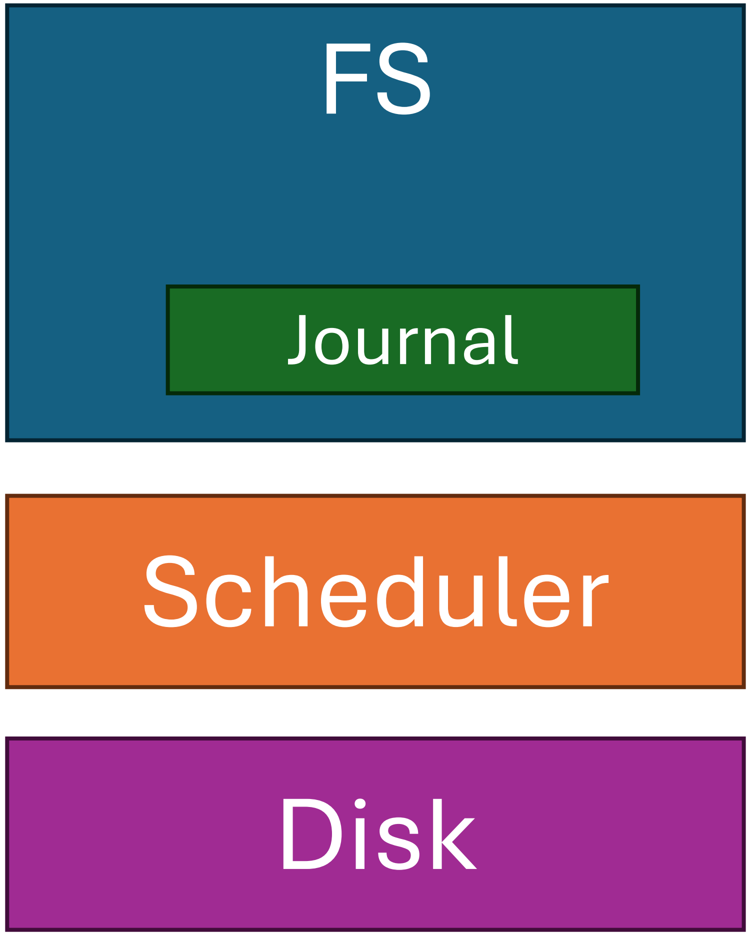
Solution

- Describe high-level changes in journal and interpret them, rather than verbose changes

Example

- “Set file size to 1024”

Integration



Journaling System

- Aware of the file system logical view
  - Knows about Inodes, directory structures, allocation tables
- Entries can contain
  - Transaction IDs
  - Block numbers
  - Operation types

A few final notes

What Journaling Fixes

- Data inconsistency

What Journaling Doesn't Fix

- Bit Flip

Summary

- Fast File System
- Crashes and Recovery

Questions?