# Operating Systems

Log-structured File System

# Overview

- LFS: Log-Structured File System

# Some Observations

- Memory gets bigger => Cache gets bigger => Read less frequent
- Hard Drives (density) gets better
- Hard Drive motors (seek, rotate) are mostly stagnant
- FFS: inode, data, bitmap, bitmap
- RAID is not part of the picture

# Summary

**Write >>> Read**

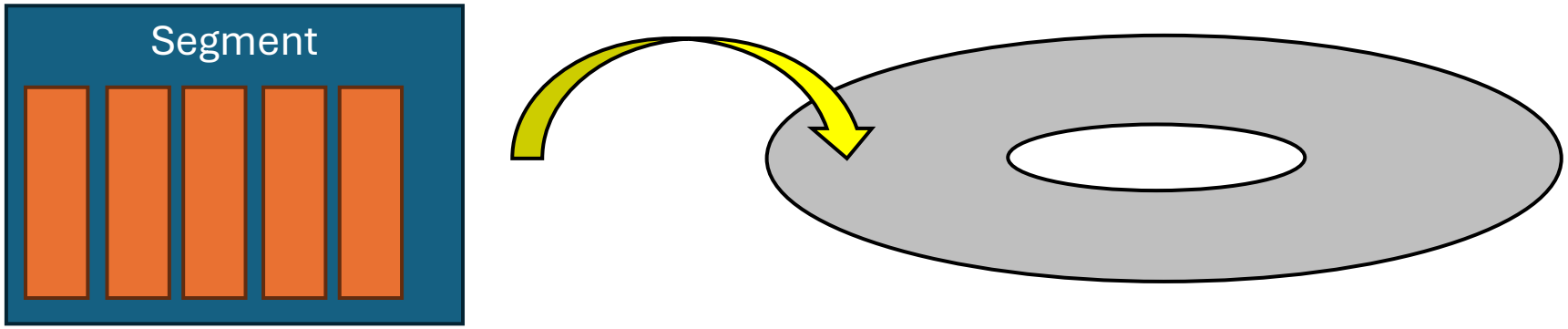
# Ideas

**Principles**

- We want to minimize ‘writes’
- We want to write ‘slabs’ of data (not random access)

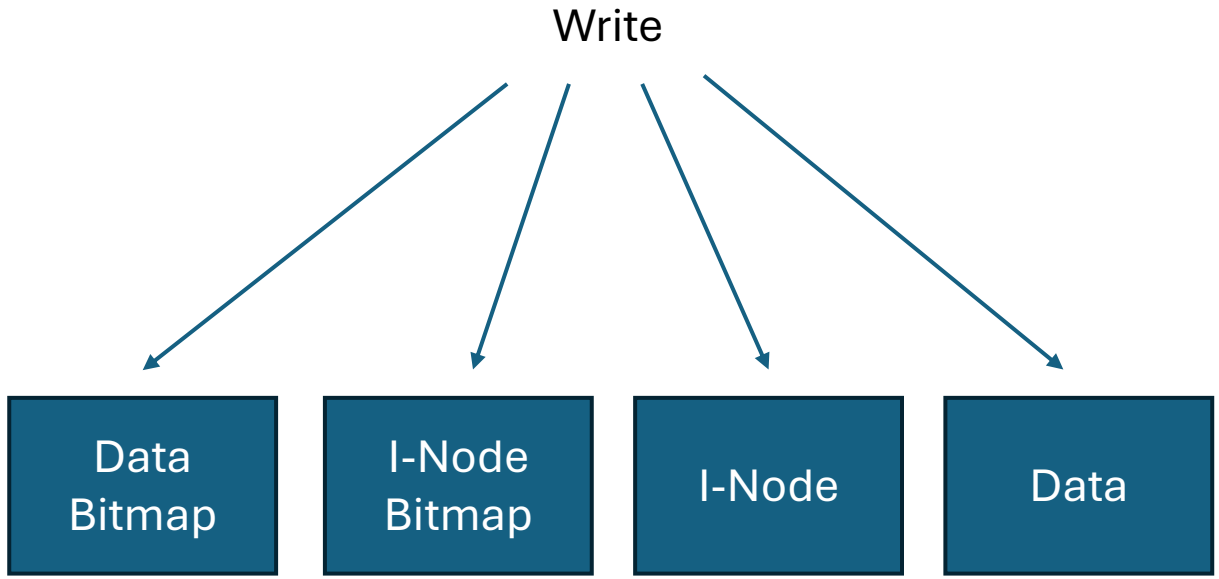
**Theory**

- Buffer all the reads into a segment
- Write the segment
- Write to a new location



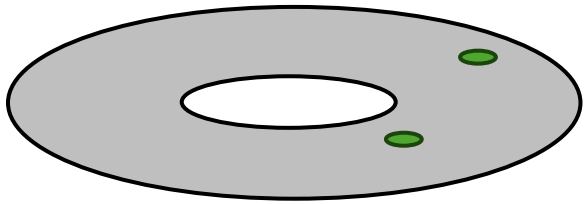
# Problems

**#1:** Writing everything together isn't 'free'. We need to think about this.



# Problems

**#1:** Writing everything together isn't 'free'. We need to think about this.

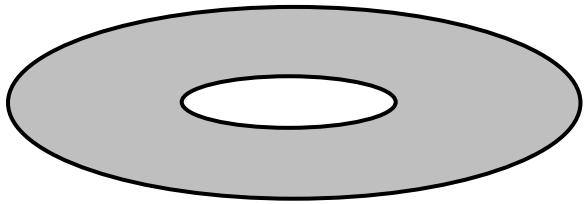


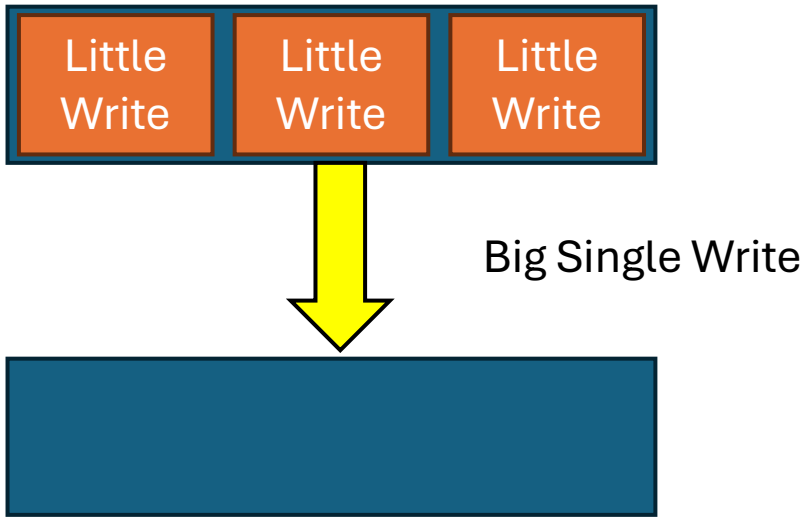
Sequential

Contiguous

# Problems

**#1:** Writing everything together isn't 'free'. We need to think about this.





# Problems

**#2:** How much to buffer?

# Activity

| | Big Buffer | Small Buffer |
|---|---|---|
| **Good** | | |
| **Bad** | | |

# Thoughts

| | Big Buffer | Small Buffer |
|---|---|---|
| **Good** | • Less write operations<br>• Better if seek/rotation time is high | • Less ‘waiting’ for memory writes |
| **Bad** | • Could lose a lot of data if it isn’t written | • More write operations<br>• Worse if seek/rotation time is high |

# Maths

$$Rate_{effective} = \frac{Data}{T_{write}} = \frac{Data}{T_{position} + \frac{Data}{Rate_{peak}}}$$

Seek/Rotation Time

# Maths

We want:
$Rate_{effective} = 0.9 \times Rate_{peak}$

$$Rate_{effective} = \frac{Data}{T_{write}} = \frac{Data}{T_{position} + \frac{Data}{Rate_{peak}}}$$

# Maths

We want:
$Rate_{effective} = 0.9 \times Rate_{peak}$

$$Rate_{effective} = \frac{Data}{T_{position} + \frac{Data}{Rate_{peak}}}$$

# Maths

We want:
$Rate_{effective} = 0.9 \times Rate_{peak}$

$$Rate_{effective} = \frac{Data}{T_{position} + \frac{Data}{Rate_{peak}}}$$

$$0.9 * Rate_{peak} = \frac{Data}{T_{position} + \frac{Data}{Rate_{peak}}}$$

# Maths

We want:
$Rate_{effective} = 0.9 \times Rate_{peak}$

$$0.9 * Rate_{peak} = \frac{Data}{T_{position} + \frac{Data}{Rate_{peak}}}$$

$$Data = 0.9 * Rate_{peak} * \left( T_{position} + \frac{Data}{Rate_{peak}} \right)$$

# Maths

We want:
$Rate_{effective} = 0.9 \times Rate_{peak}$

$$Data = 0.9 * Rate_{peak} * \left( T_{position} + \frac{Data}{Rate_{peak}} \right)$$

$$Data = 0.9 * Rate_{peak} * T_{position} + 0.9 * Rate_{peak} * \frac{Data}{Rate_{peak}}$$

# Maths

We want:
$Rate_{effective} = 0.9 \times Rate_{peak}$

$$Data = 0.9 * Rate_{peak} * T_{position} + 0.9 * \cancel{Rate_{peak}} * \frac{Data}{\cancel{Rate_{peak}}}$$

$$Data = 0.9 * Rate_{peak} * T_{position} + 0.9 * Data$$

# Maths

We want:
$Rate_{effective} = 0.9 \times Rate_{peak}$

$$Data = 0.9 * Rate_{peak} * T_{position} + 0.9 * Data$$

$$Data - 0.9 * Data = 0.9 * Rate_{peak} * T_{position}$$

# Maths

We want:
$Rate_{effective} = 0.9 \times Rate_{peak}$

$$Data - 0.9 * Data = 0.9 * Rate_{peak} * T_{position}$$

$$Data = \frac{0.9 * Rate_{peak} * T_{position}}{1 - 0.9}$$

# Maths

We want:
$Rate_{effective} = F \times Rate_{peak}$

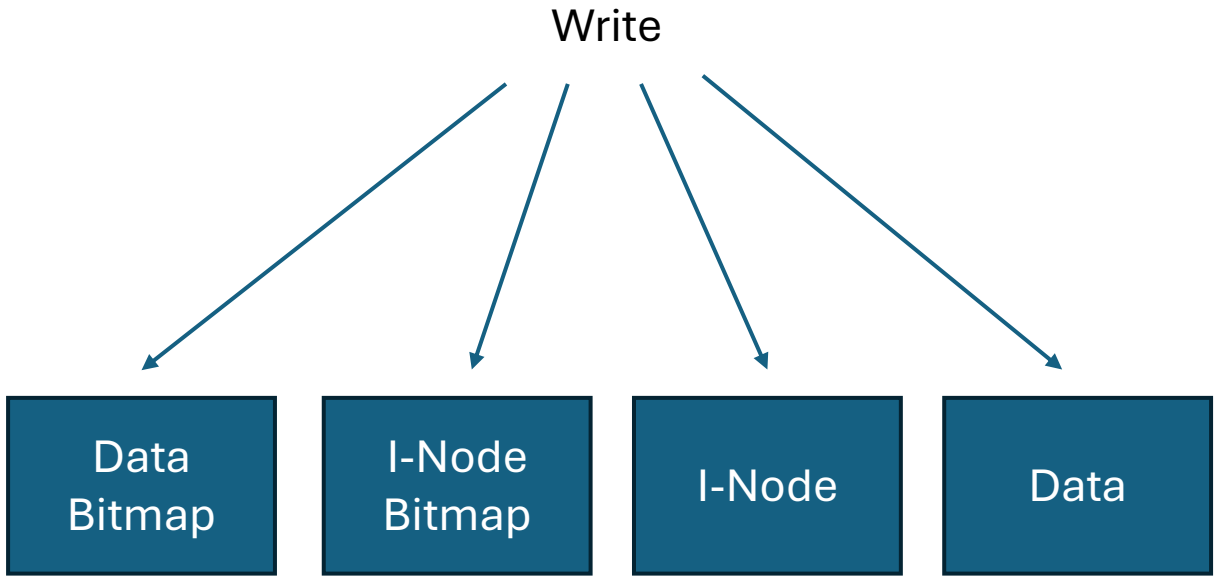
$$Data - 0.9 * Data = 0.9 * Rate_{peak} * T_{position}$$

$$Data = \frac{F}{1 - F} * Rate_{peak} * T_{position}$$

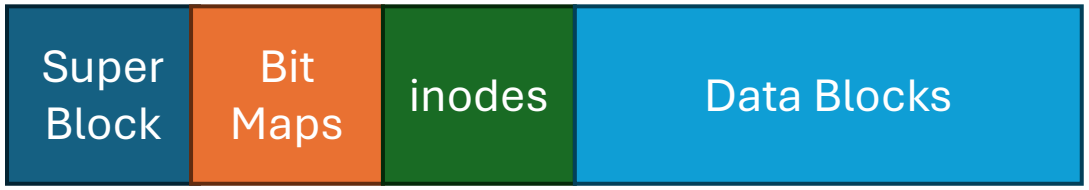
We now know how much data we would need to write to get the required level of throughput

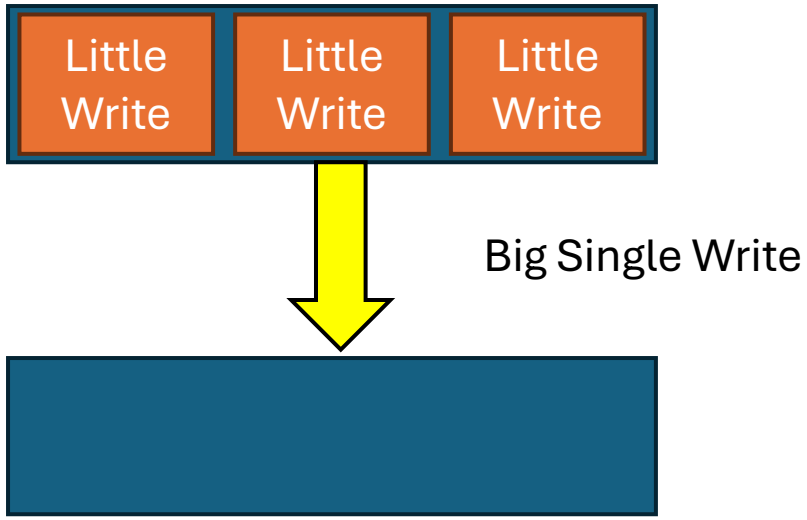
# Problems

**#3:** I-Nodes



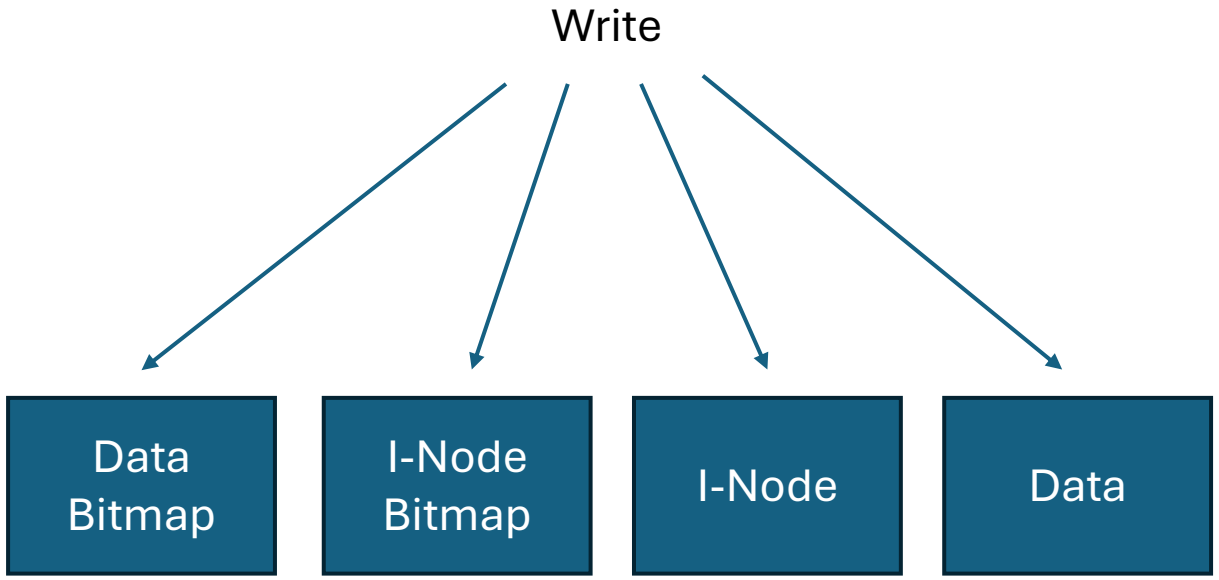

How we used to do it...

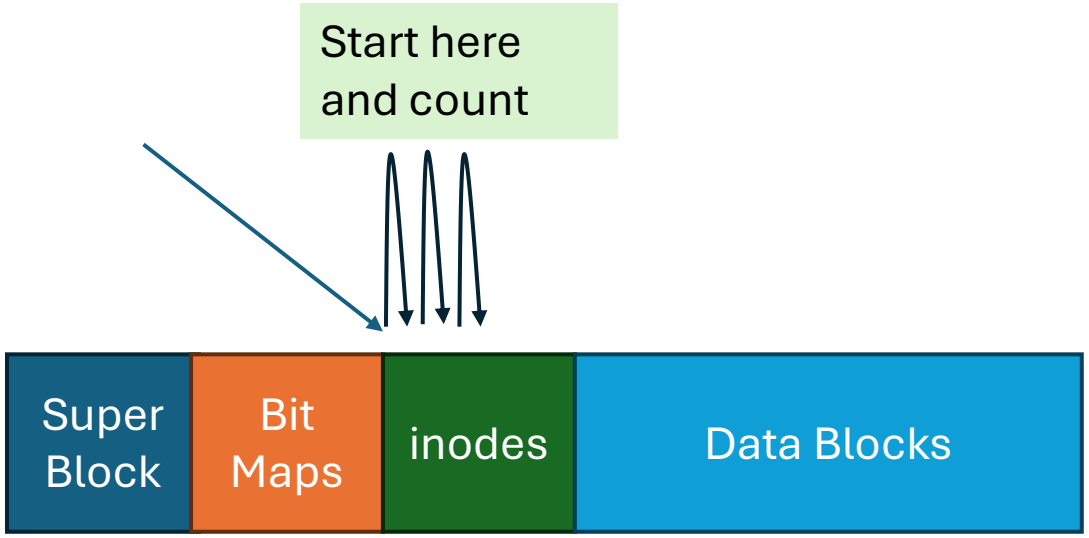


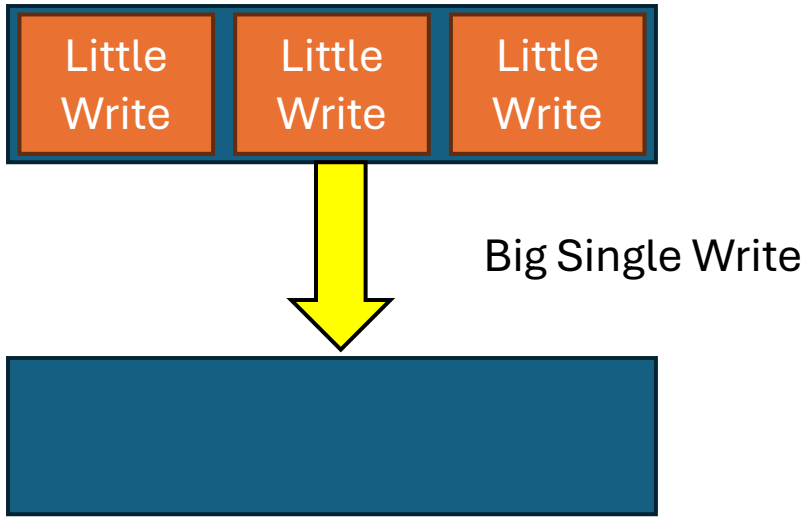



# Problems

**#3:** I-Nodes

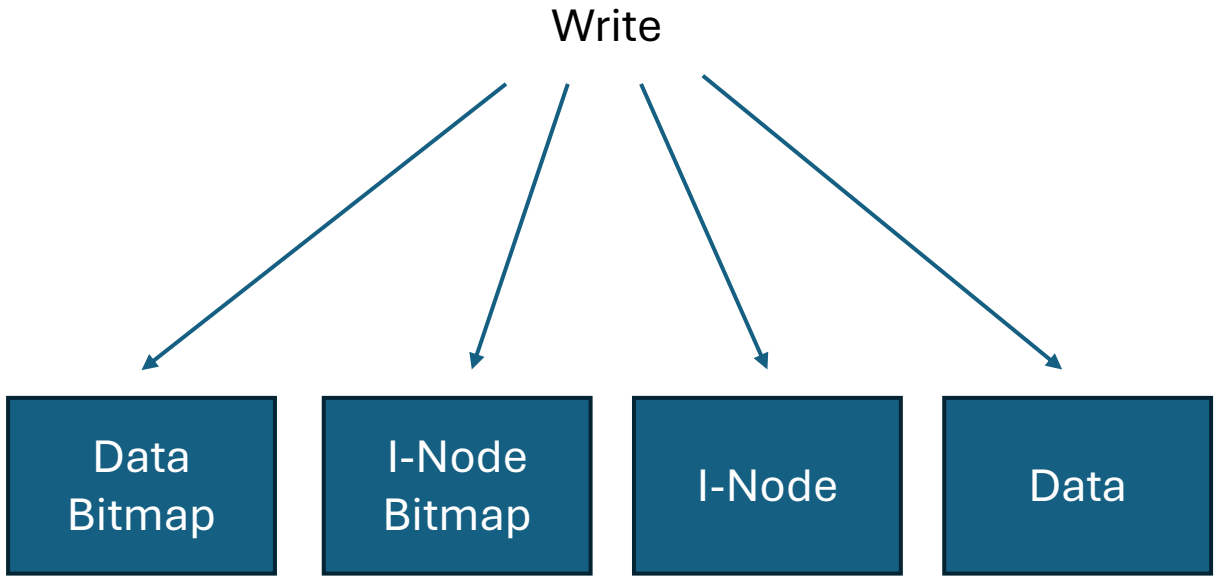


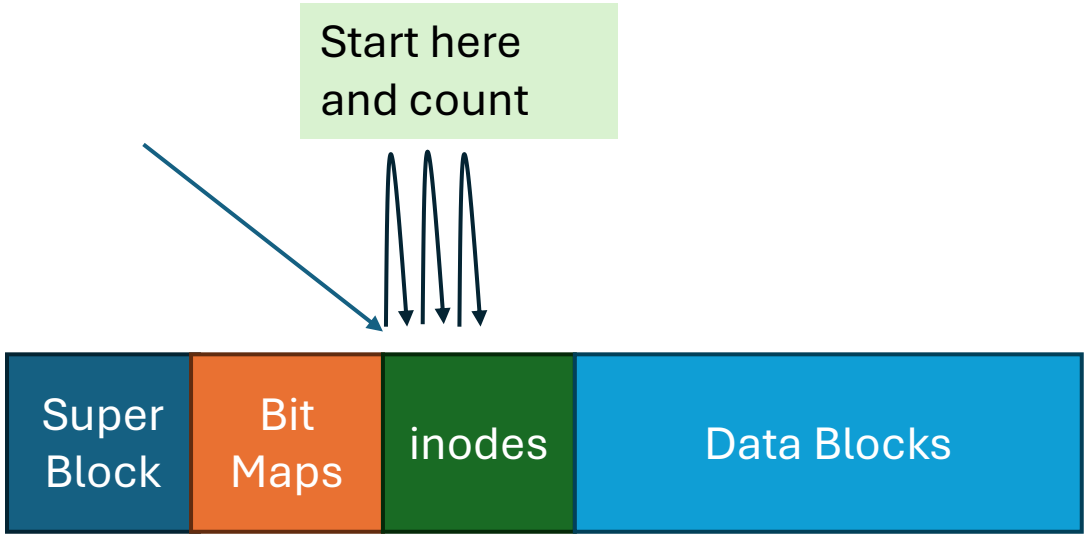



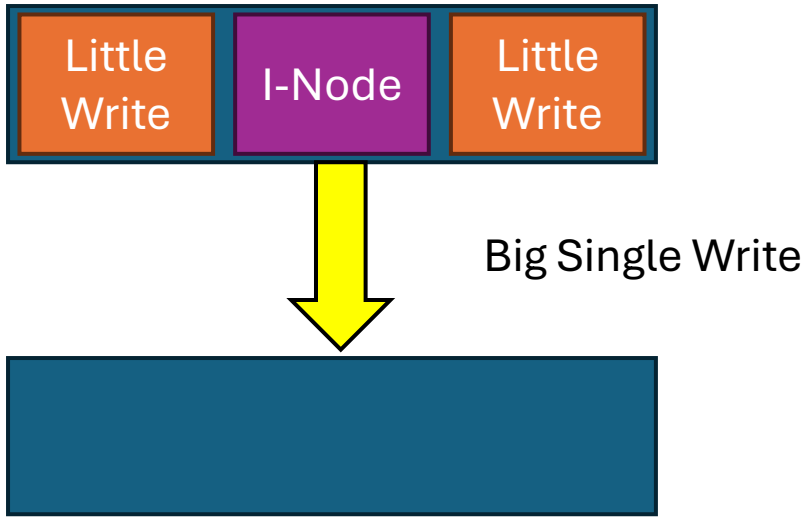



# Problems

**#3:** I-Nodes



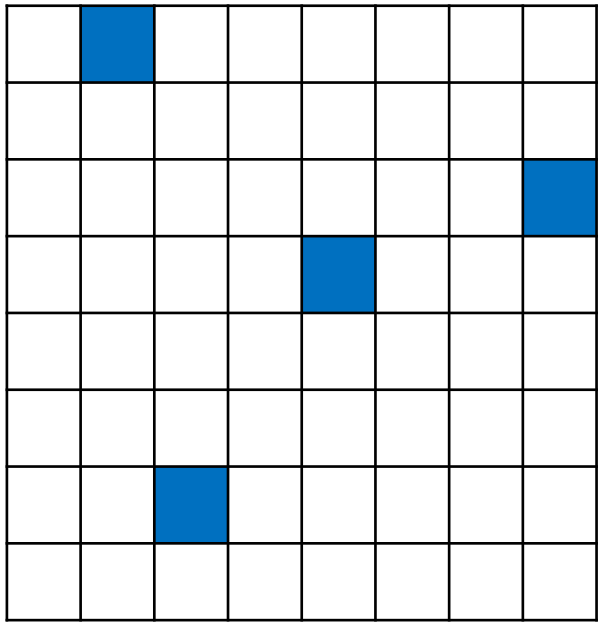



# Problems

**#3:** Where are the I-Nodes?

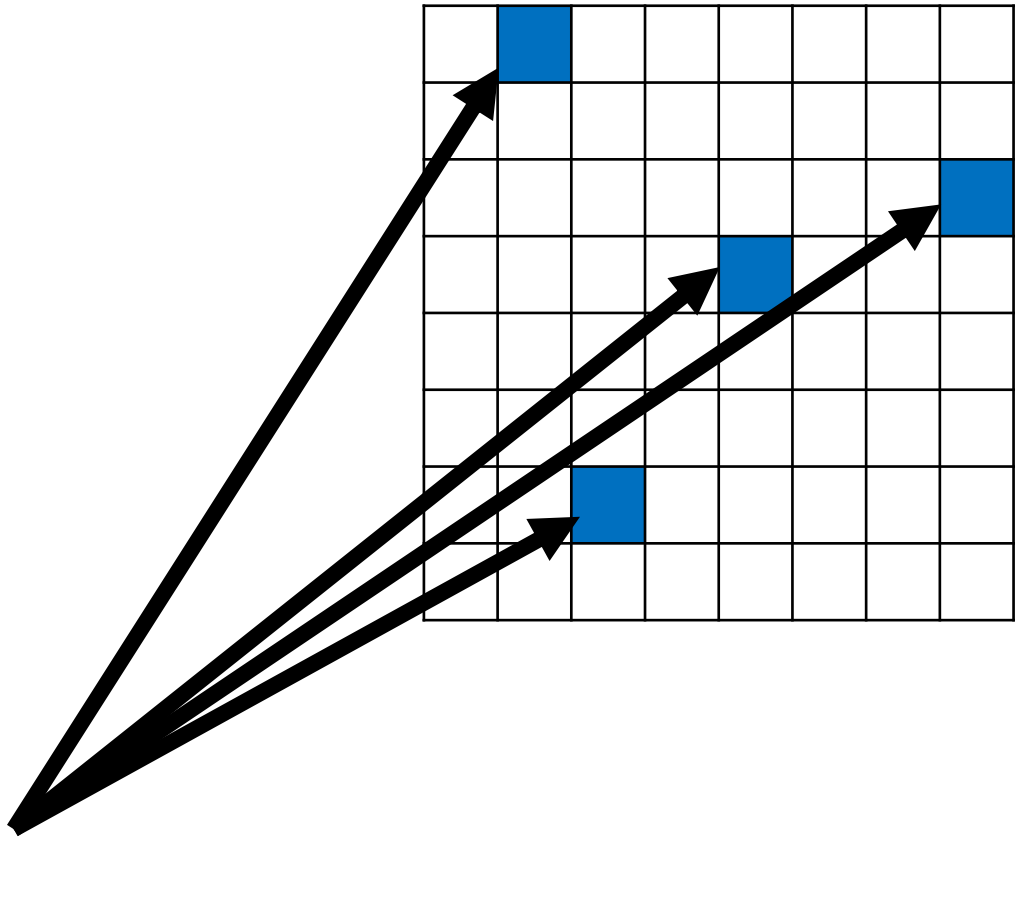
- They are scattered
- The ‘latest version’ keeps moving



# Problems

**#3:** Where are the I-Nodes?

- They are scattered
- The ‘latest version’ keeps moving



# imap

**Inode Map (imap)**

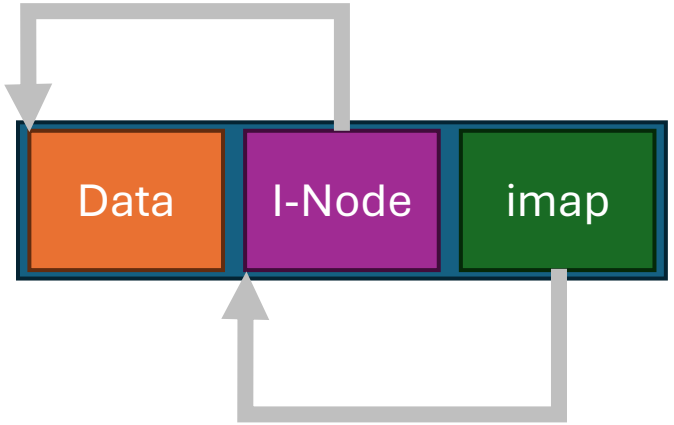
- Maps inode # to physical address

| # | |
|---|---|
| 1 | x0000ffff |
| 2 | x012e23b2 |
| 3 | x351f2a21 |
| 4 | x9eeebac4 |
| … | |

But… where must we store said map?

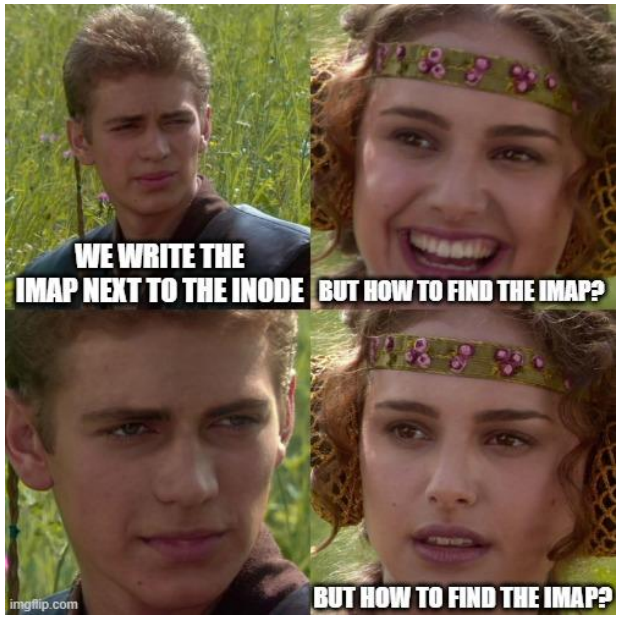
**In Memory**

# Writing a Segment




**The Cunning Plan**

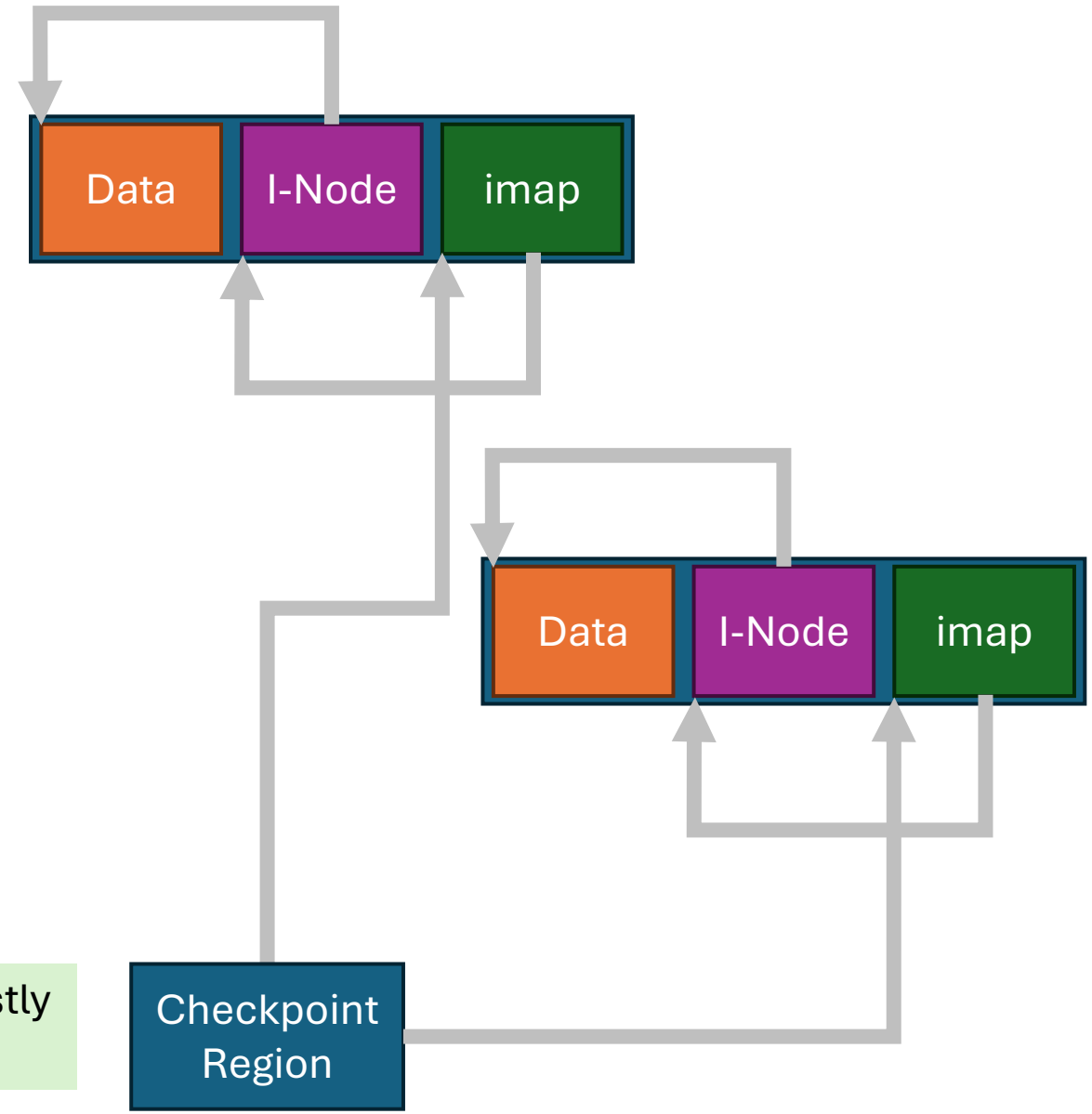
- We write the data
  - But where is the data?
- We write the inode next to the data!
  - But where is the inode?
- We write the imap (or part of it) next to the inode



# Checkpoint Region (CR)

**Checkpoint Region**

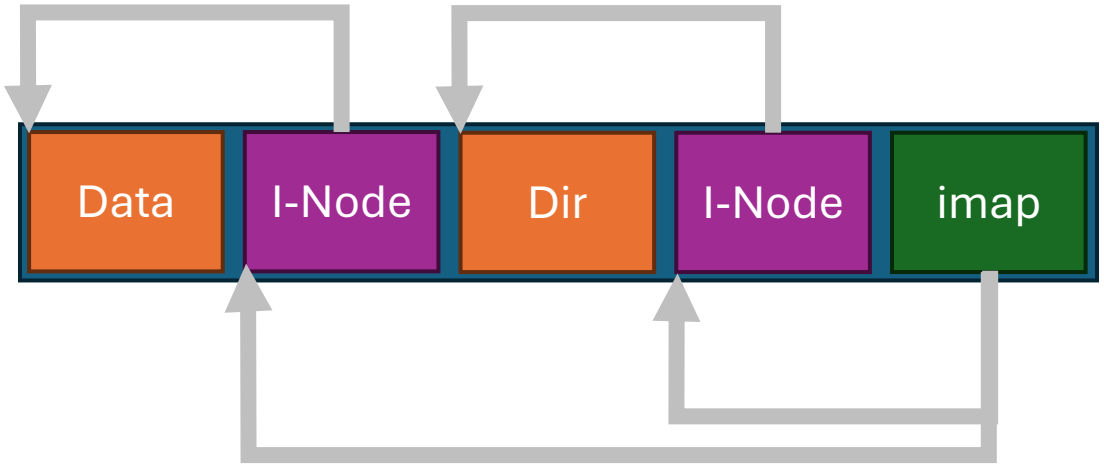
- Contains pointers to the latest pieces of the **inode** map
- Updated in ‘slow time’ (~30s)



# Updating Files

**Directory Updates**

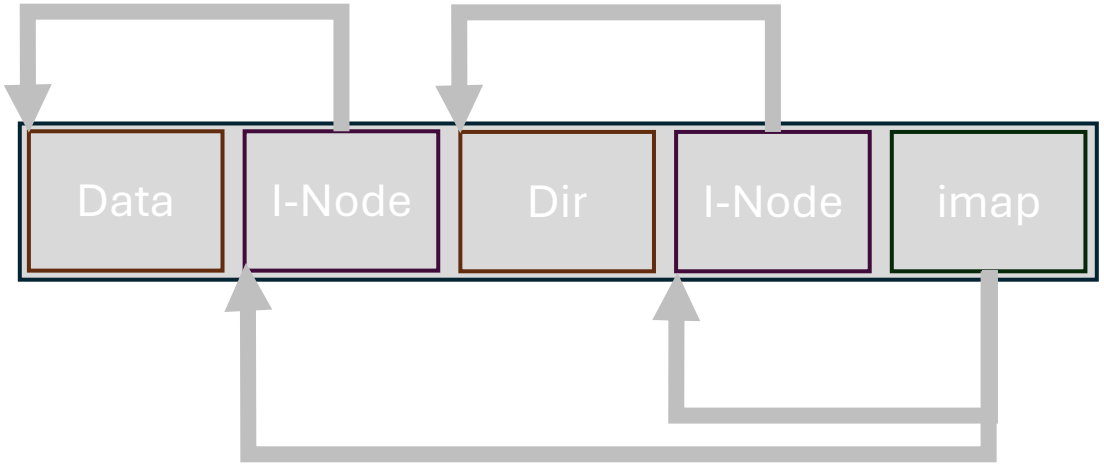
- Creating/modifying a file updates the directory, so…
- This also prevents the **recursive update problem** (only the inode map is updated with inode locations)



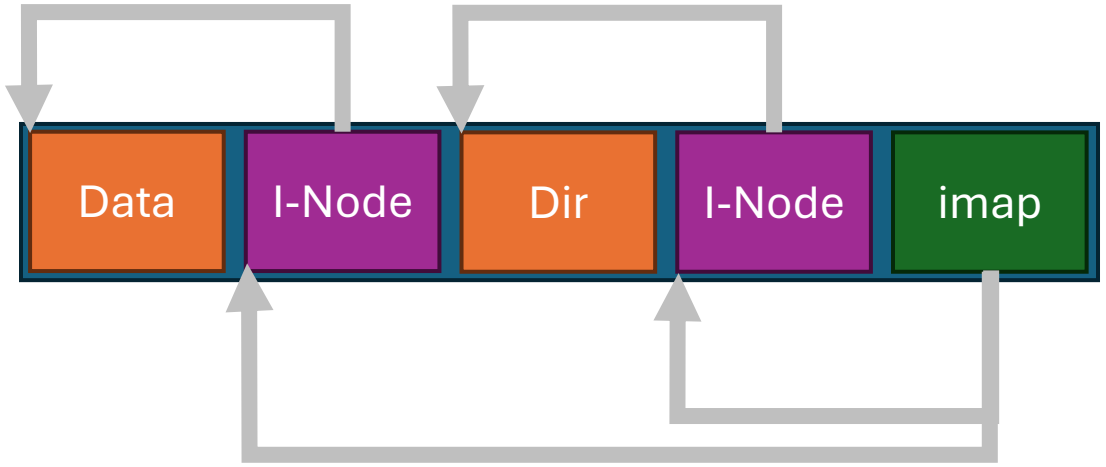
# Problems

**#4:** New ‘writes’ leave old writes…

Old version?




New version?



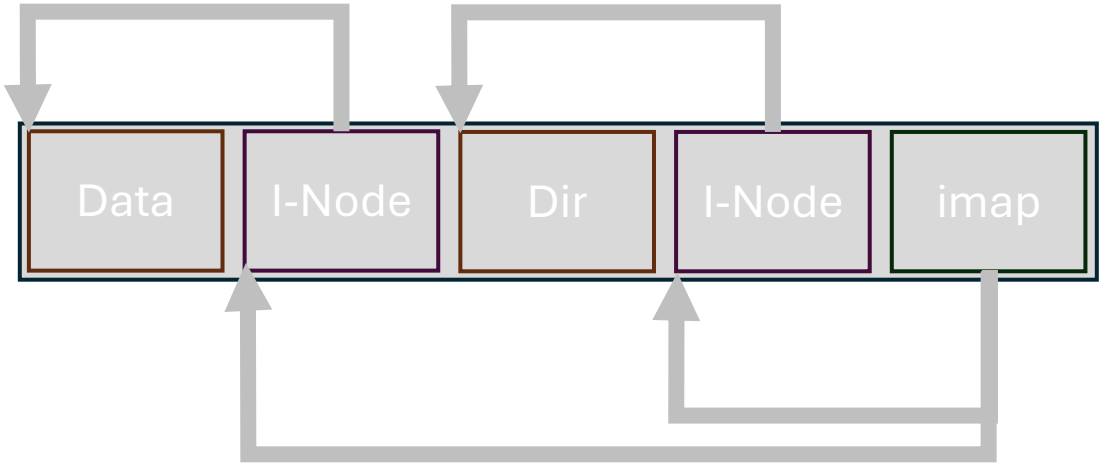
# Problems

**#4:** New ‘writes’ leave old writes…

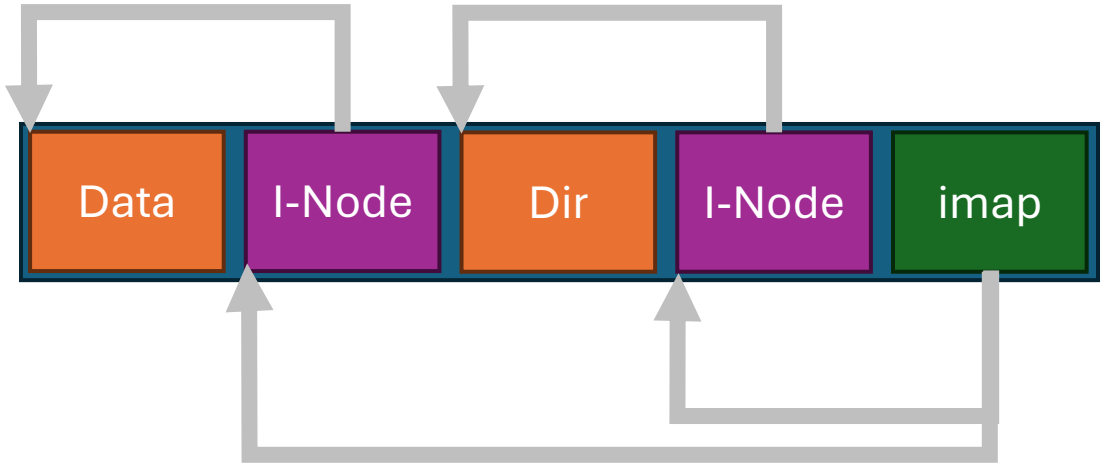
**Case 1:** Everything new

**Case 2:** Inode now points to a new block

Old version?




New version?



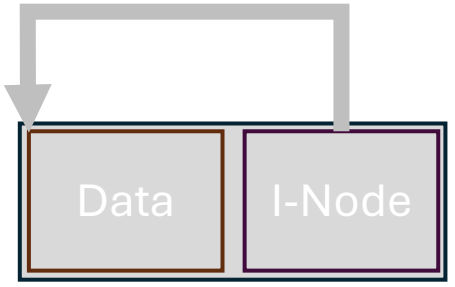
# Problems

**#4:** New ‘writes’ leave old writes…

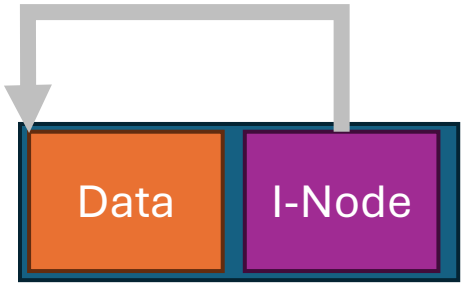
**Case 1:** Everything new

**Case 2:** Inode now points to a new block

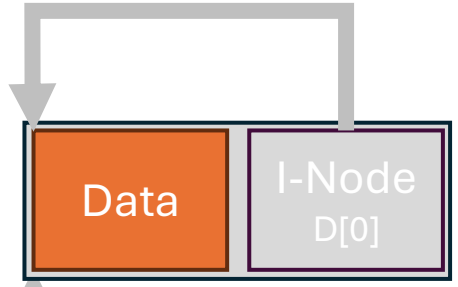
Old version?




New version?




Old version?

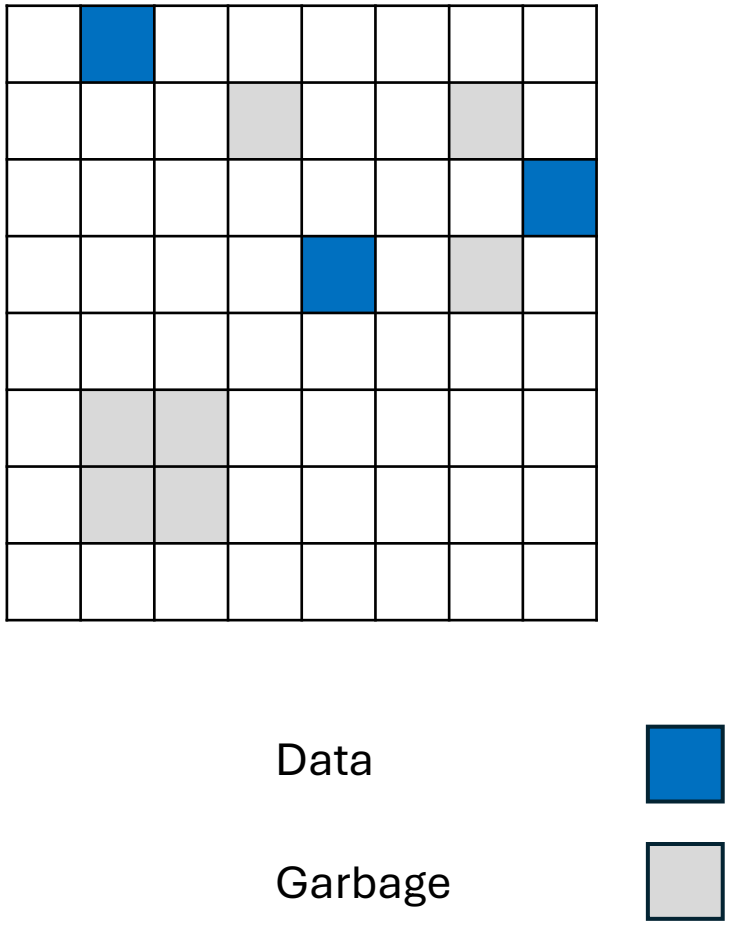



New version?

# Problems

**#4:** New ‘writes’ leave old writes…

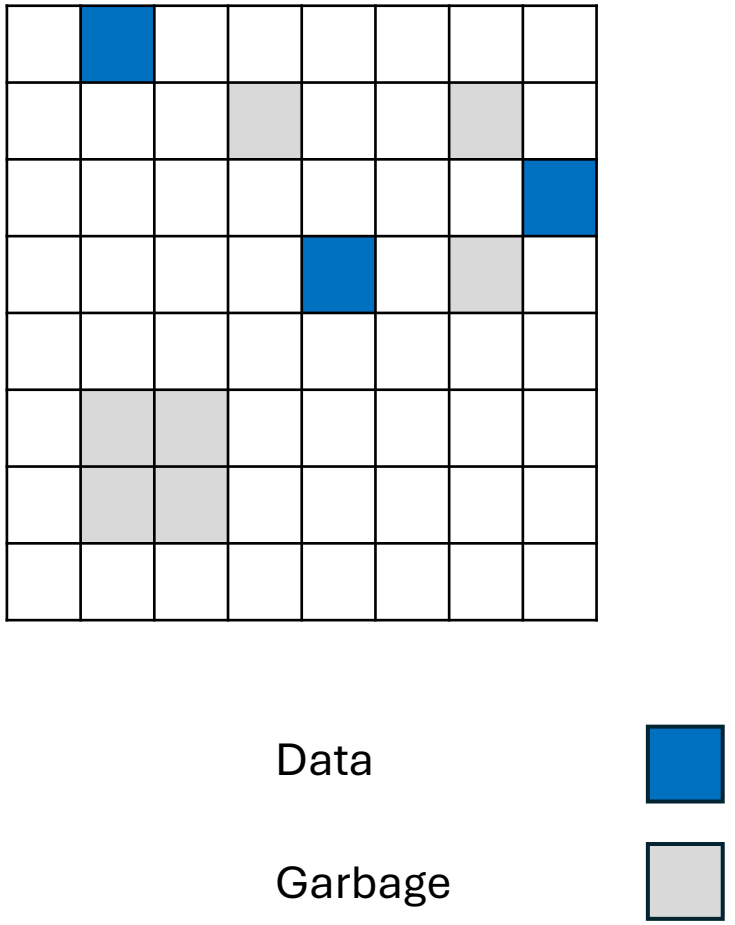
**Garbage Collection!**



# Garbage Collection

**Problems:**

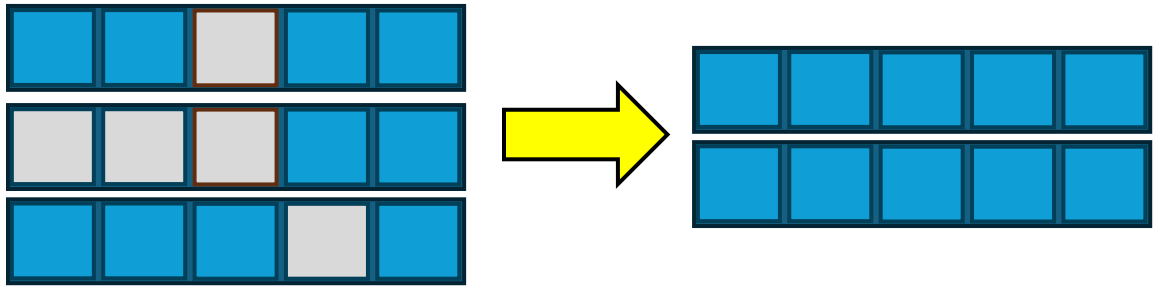
- Need to be careful for holes

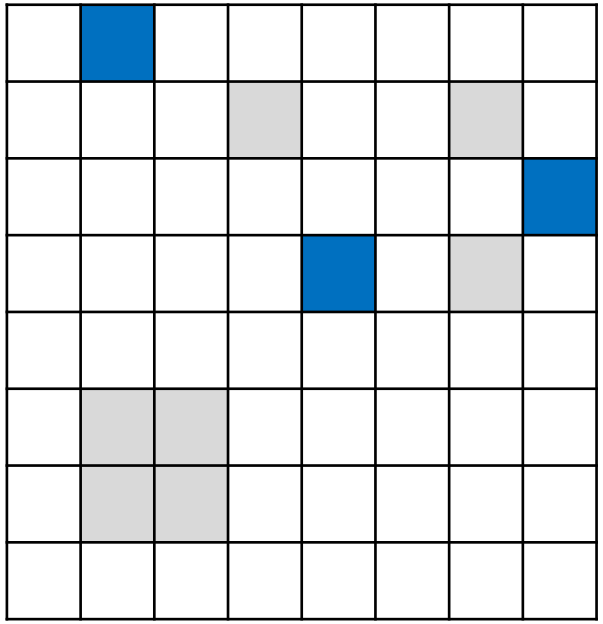



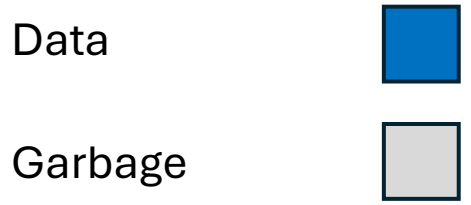
# Garbage Collection

**Problems:**

- Need to be careful for holes



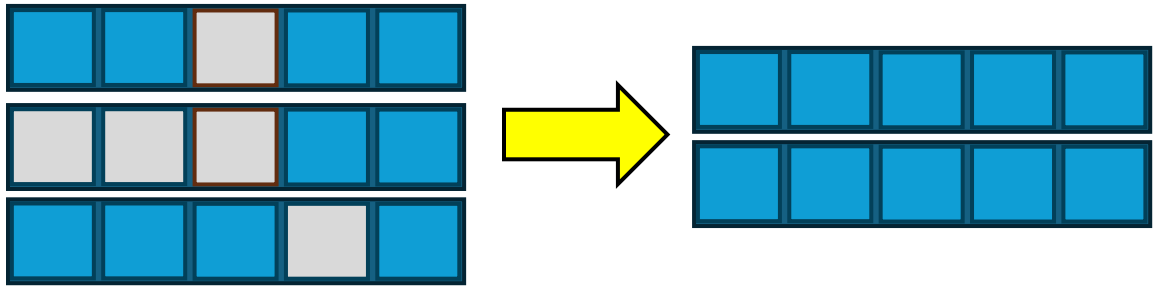


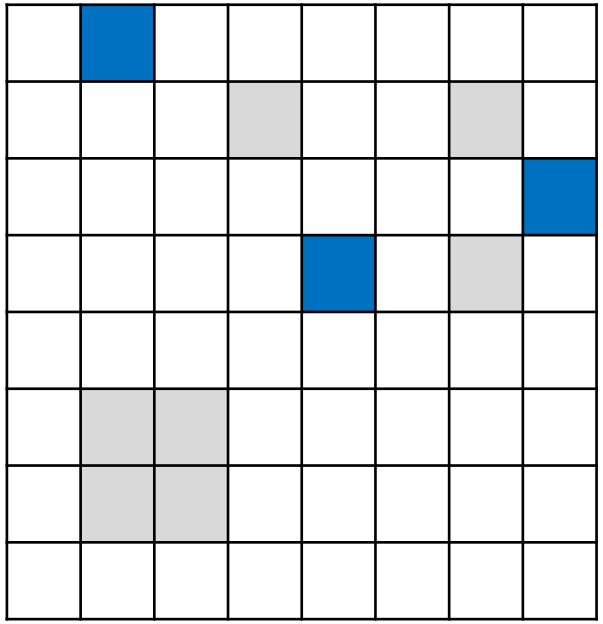


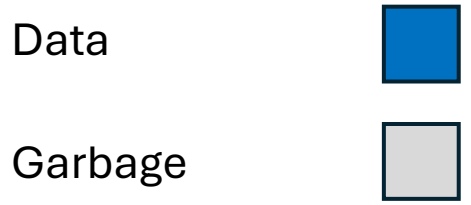
# Garbage Collection

**Questions:**

- How to track ‘dead blocks’?
- How often to collect garbage?







# Block Liveness

- Each segment has a **segment summary block**

## Process

- Read **segment summary block**
- Find **inode number**
- Check **imap** (follow to the data)
- Is it where it says it should be?

## Segment Summary Block

- Data Block:
  - inode number
  - offset (which block am I?)

# What and When?

**When?**

- Periodically
- When you’re bored (idle time)

**What?**

- Subject to research…

**Thoughts:**

- Frequently changing files are collected less frequently?
- Infrequently changing files are collected more frequently?

# Summary

- LFS: Log-Structured File System

# Problems

**#5:** Crash…

- **While writing a segment**
- While writing the checkpoint region

We also do some **logging**…

# Problems

**#5:** Crash…

- While writing a segment
- **While writing the checkpoint region**

We have two… one at each end with alternate writing…

We can use timestamps and consistency to work out which one is correct (basically versioning)

Reminder, the **checkpoint region** is where we store the references to the imap chunks

# Problems

**#5:** Crash…

- While writing a segment
- **While writing the checkpoint region**

We have two… one at each end with alternate writing…

We can use timestamps and consistency to work out which one is correct (basically versioning)

Reminder, the **checkpoint region** is where we store the references to the imap chunks

# Problems

**#5:** Crash…

- While writing a segment
- **While writing the checkpoint region**

We lose ~30s of requests

Can fix through **roll forward**:

- Read through the log finding valid updates…

# Questions?