File Systems II

COMS W4118

References: Operating Systems Concepts (9e), Linux Kernel Development, previous W4118s

Copyright notice: care has been taken to use only those web images deemed by the instructor to be in the public domain. If you see a copyrighted image on any slide and are the copyright owner, please contact the instructor. It will be removed.

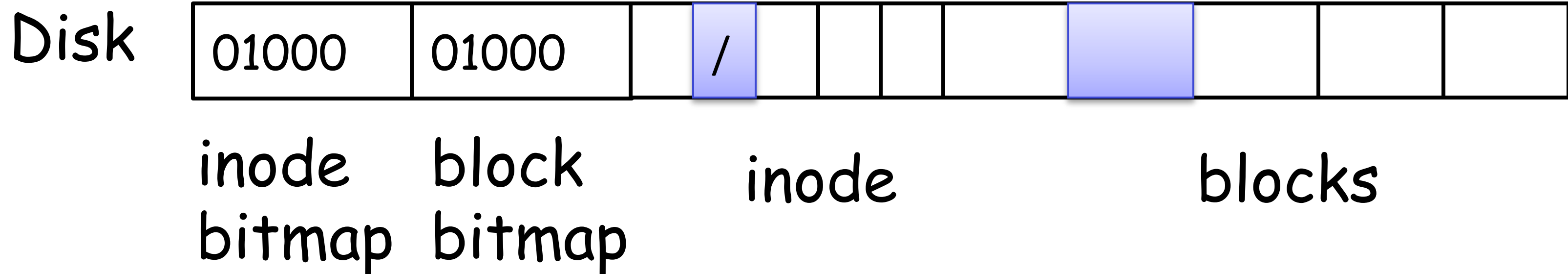
Outline

- The Linux Third Extended File System (Ext3)
  - What is the file system consistency problem?
  - How to solve the consistency problem using journaling?
- Log-Structured File system (LFS)
  - What was the motivation of LFS?
  - How did LFS work?

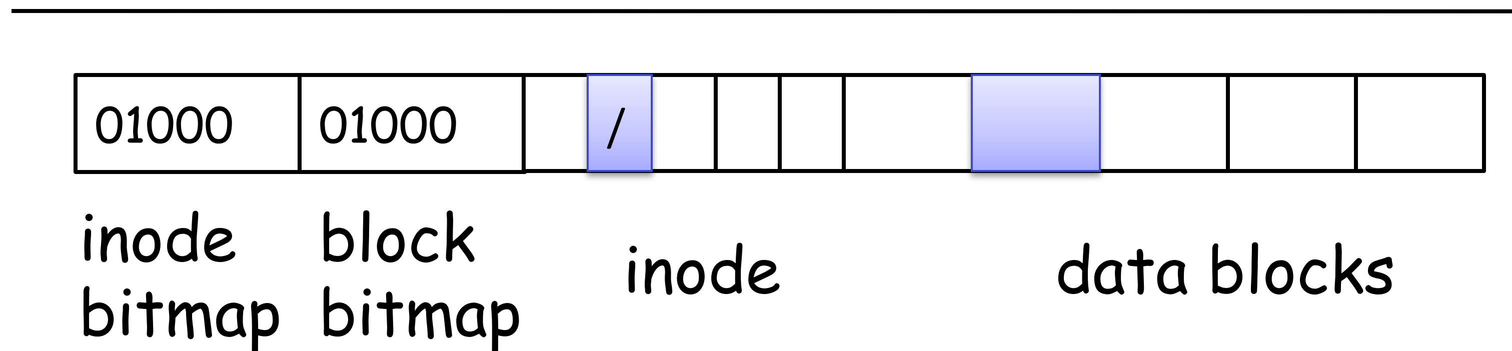
The consistent update problem

- Atomically update file system from one consistent state to another, which may require modifying several sectors, despite that the disk only provides atomic write of one sector at a time

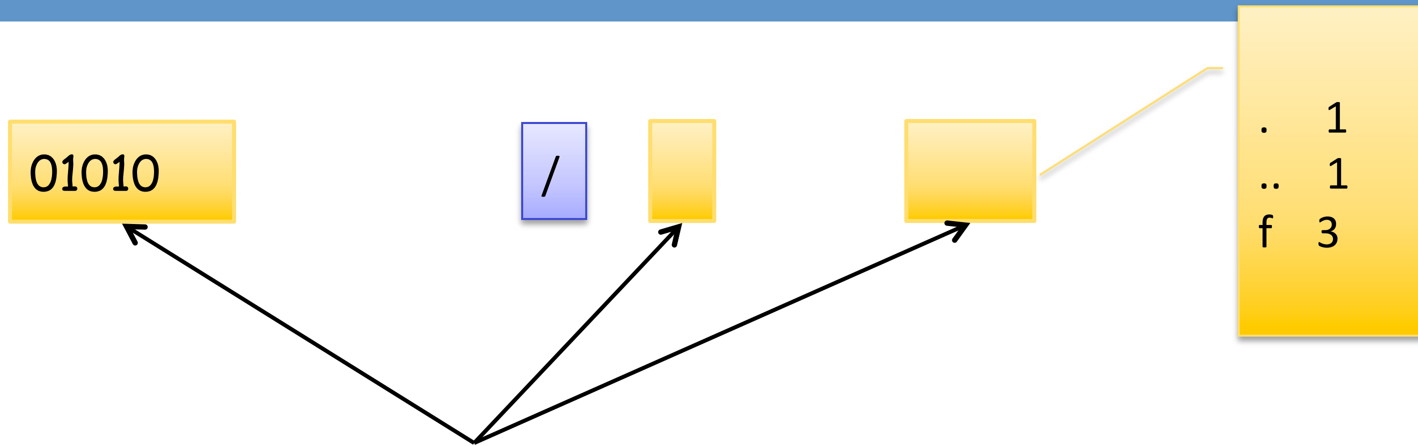
Example: Ext2 File Creation

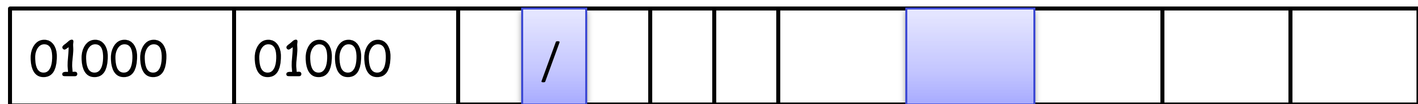


Read to In-memory Cache



Modify blocks





Crash?

- Disk: atomically write one sector
  - Atomic: if crash, a sector is either completely written, or none of this sector is written
- An FS operation may modify multiple sectors
  - Crash ➔ FS partially updated
- Like race conditions in concurrent programs
  - But, can't lock out a failure using a lock!

Possible Crash Scenarios

- File creation dirties three blocks
  - inode bitmap (B), inode for new file (I), parent directory data block (D)
- Old and new contents of the blocks
  - B = 01000 B' = 01010
  - I = free I' = allocated, initialized
  - D = {} D' = {<f, 3>}
- Crash scenarios: any subset can be written

B I D	Consistent (new data lost)
B' I D	Inconsistency. Bitmap says I allocated, but no file/dir using I
B I' D	As if nothing occurred
B I D'	Serious problem. Trust D' and follow pointer? Garbage! Trust B that I not allocated? Inconsistency!
B' I' D	Inconsistency. Bitmaps says I allocated, but no one uses I.
B' I D'	Most serious problem. FS completely consistent if we just look at the pointers and bitmap. But I hasn't been initialized and contains garbage.
B I' D'	Inconsistency
B' I' D'	Consistent (new data preserved)

One solution: fsck

- Upon reboot, scan entire disk to make FS consistent

- Advantages
  - Simplify FS code
  - Can repair more than just crashed FS (e.g., bad sector)

- Disadvantages
  - Slow to scan large disk
  - Cannot correctly fix all crashed disks (e.g., B’ ı D’)
  - Not well-defined consistency

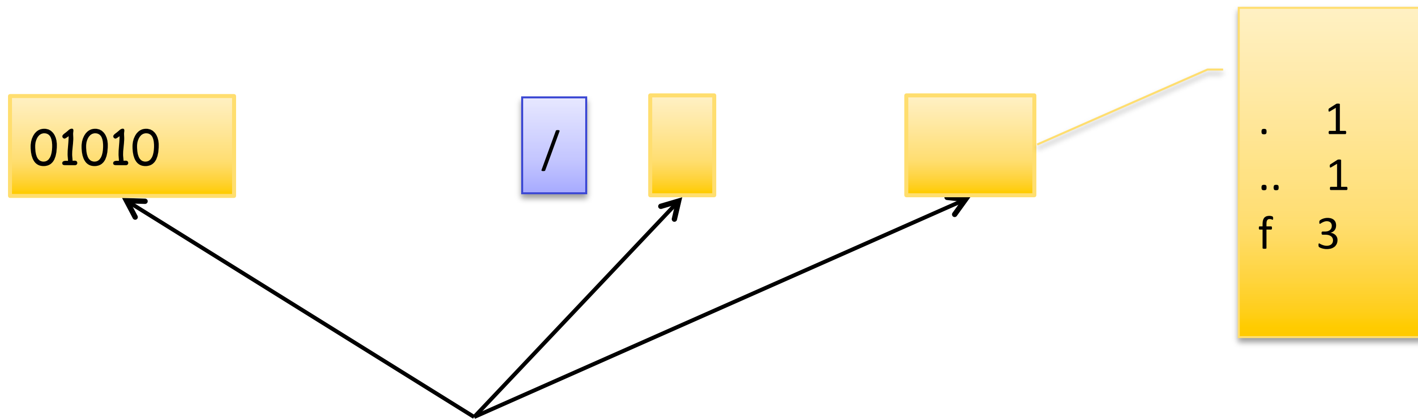
Another solution: Journaling

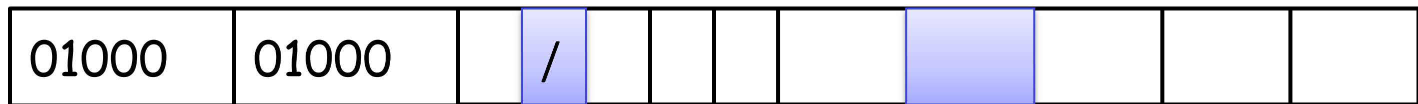
- Write-ahead logging from database community
- Persistently write intent to log (or journal), then update file system
  - Crash before intent is written == no-op
  - Crash after intent is written == redo op
- Advantages
  - no need to scan entire disk
  - Well-defined consistency

Ext3 Journaling

- Physical journaling: write real block contents of the update to log
  - Four totally ordered steps
    - Commit dirty blocks to journal as one transaction
    - Write commit record
    - Write dirty blocks to real file system
    - Reclaim the journal space for the transaction
- Logical journaling: write logical record of the operation to log
  - "Add entry F to directory data block D"
  - Complex to implement
  - May be faster and save disk space

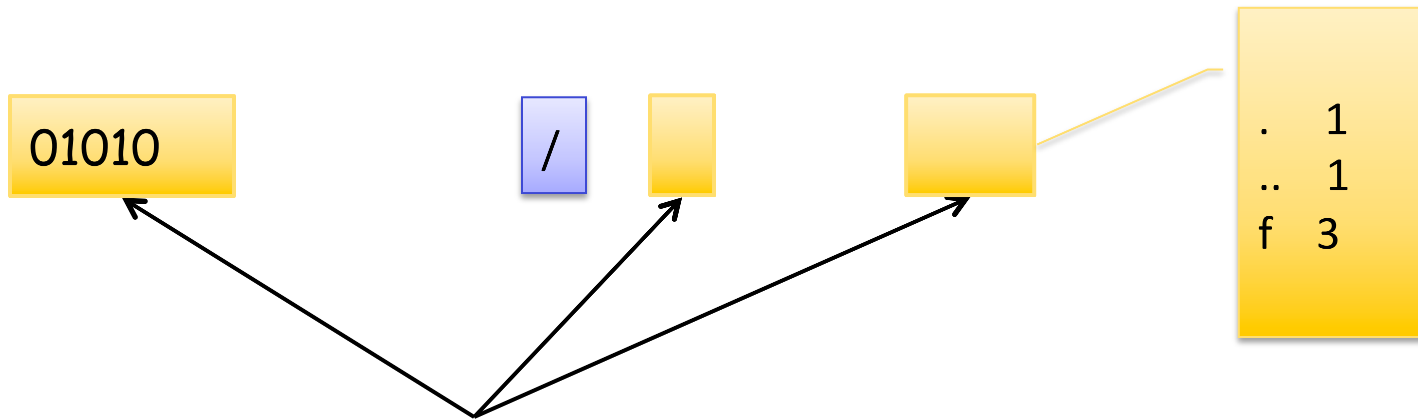
Step 1: write blocks to journal

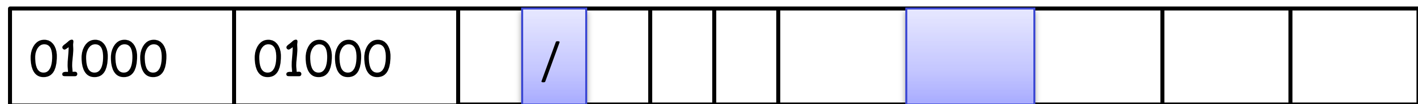




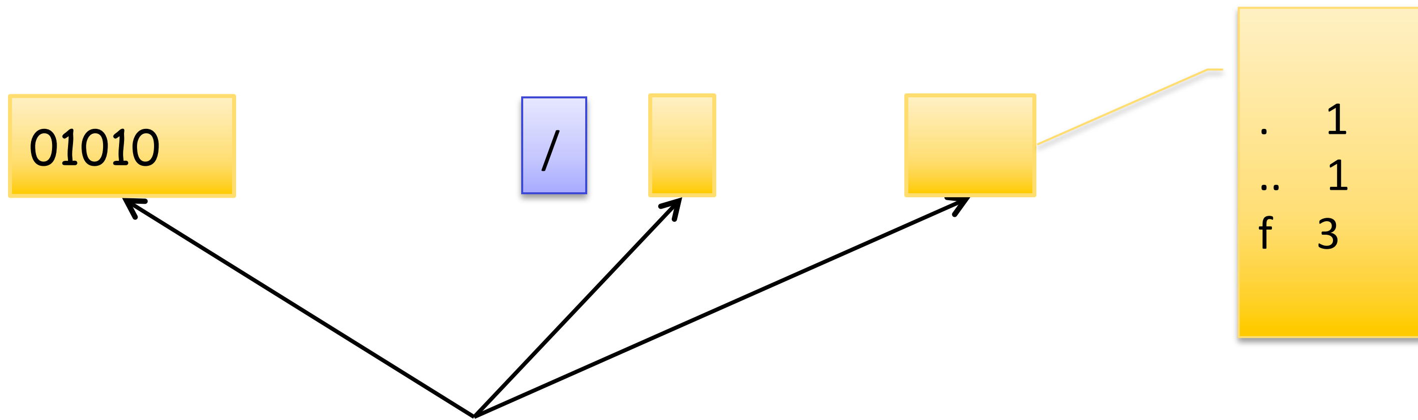
journal

Step 2: write commit record

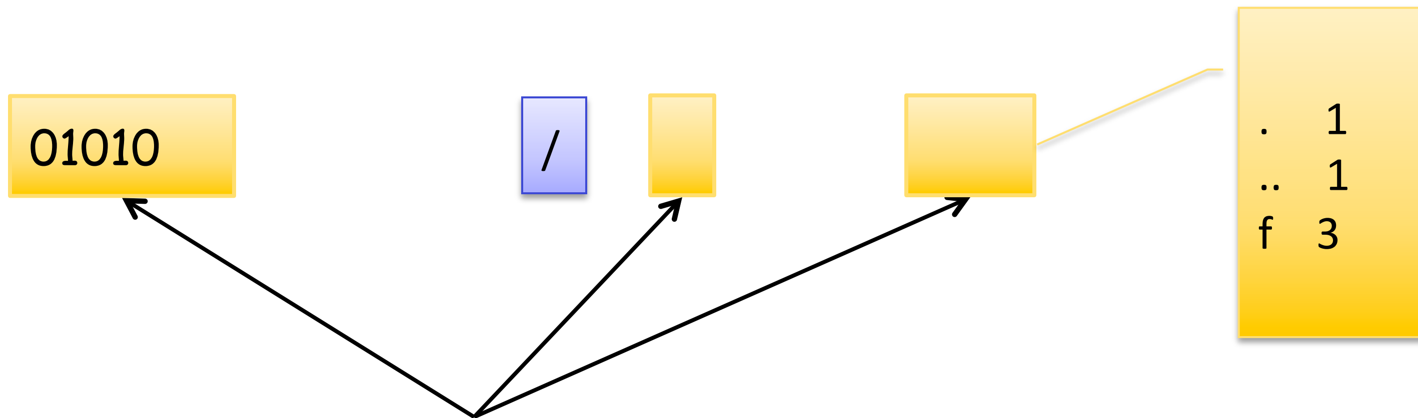




Step 3: write dirty blocks to real FS



Step 4: reclaim journal space



"Dirty" blocks,
must write to disk

Summary of Journaling write orders

- Journal writes < FS writes
  - Otherwise, crash ➔ FS broken, but no record in journal to patch it up
- FS writes < Journal clear
  - Otherwise, crash ➔ FS broken, but record in journal is already cleared
- Journal writes < commit block < FS writes
  - Otherwise, crash ➔ record appears committed, but contains garbage

Ext3 Journaling Modes

- Journaling is expensive
  - one write = two disk writes, two seeks
  - Several journaling modes balance consistency and performance
- Data journaling: journal all writes, including file data
  - Problem: expensive to journal data
- Metadata journaling: journal only metadata
  - Used by most FS (IBM JFS, SGI XFS, NTFS)
  - Problem: file may contain garbage data
- Ordered mode: write file data to real FS first, then journal metadata
  - Default mode for ext3
  - Problem: old file may contain new data

Outline

- The Linux Third Extended File System (Ext3)
  - What is the file system consistency problem?
  - How to solve the consistency problem using journaling?
- Log-Structured File system (LFS)
  - What was the motivation of LFS?
  - How did LFS work?

Log-structured file system

- Motivation
  - Faster CPUs: I/O becomes more and more of a bottleneck
  - More memory: file cache is effective for reads
  - Implication: writes compose most of disk traffic
- Problems with previous FS
  - Perform many small writes
    - Good performance on large, sequential writes, but many writes are still small, random
  - Synchronous operation to avoid data loss
  - Depends upon knowledge of disk geometry

LFS idea

- Insight: treat disk like a tape-drive
  – Disk performs best for sequential access
  – Essentially, extreme journaling
  – Get rid of FS snapshot, everything in the journal
- Write data to disk in a sequential log
  – Delay all write operations
  – Write metadata and data for all files intermixed in one operation
  – Do not overwrite old data on disk

Pros and cons

- Pros
  - Always Large sequential writes → good performance
  - No knowledge of disk geometry
    - Assume sequential better than random
- Potential problems
  - How do you find data to read?
  - What happens when you fill up the disk?

Read in LFS

- Same basic structures as Unix
  - Directories, inodes, indirect blocks, data blocks
  - Reading data block implies finding the file's inode
    - Unix: inodes kept in array
    - LFS: inodes move around on disk

- Solution: inode map indicates where each inode is stored
  - Small enough to keep in memory
  - inode map written to log with everything else
  - Periodically written to known checkpoint location on disk for crash recovery

Efficient Reads: Indexing the Log

UNIX FFS (or Ext2)

Inode area

File data

File inode

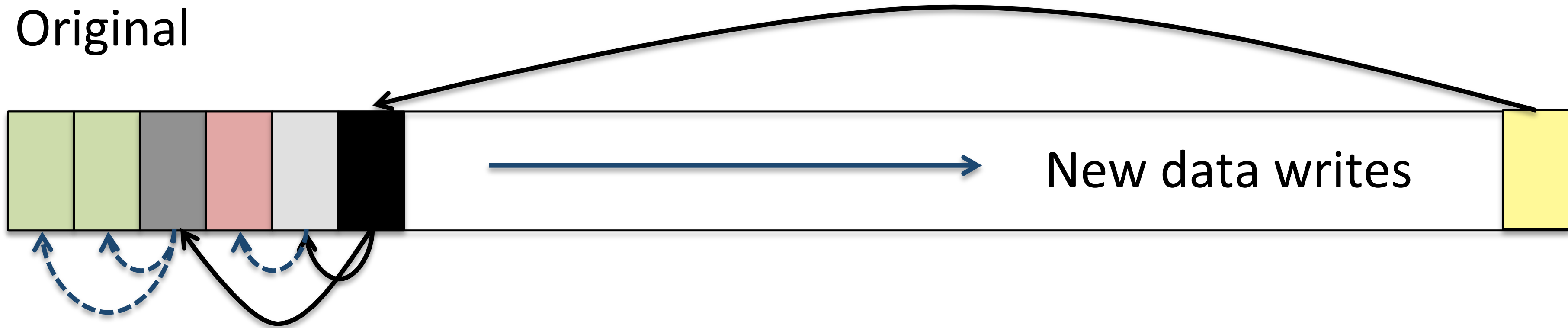
Dir data

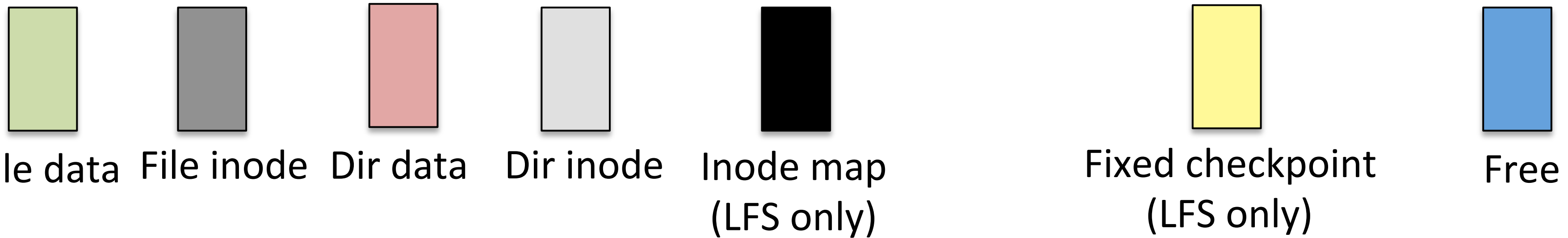
Dir inode

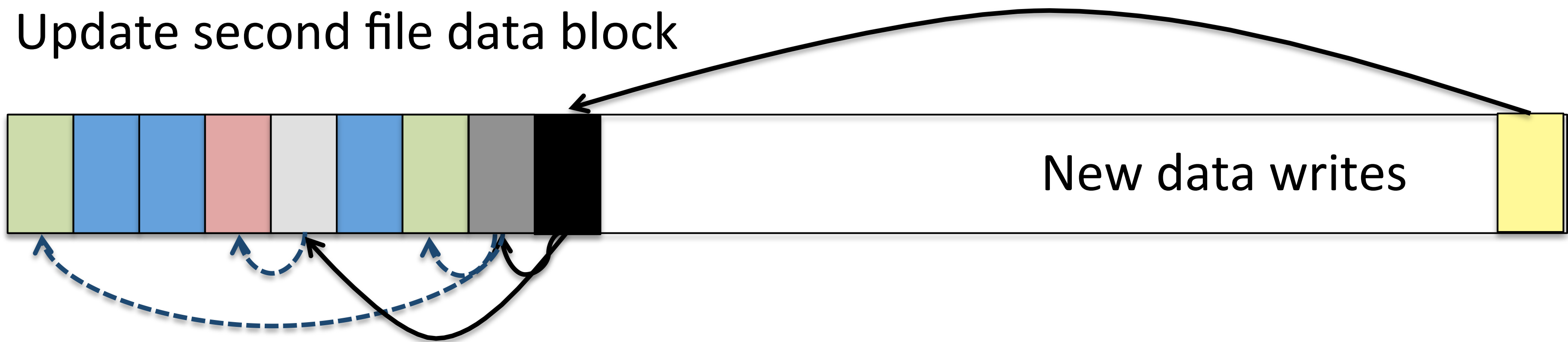
Inode map (LFS only)

Fixed checkpoint (LFS only)

Writes: Copy on Write

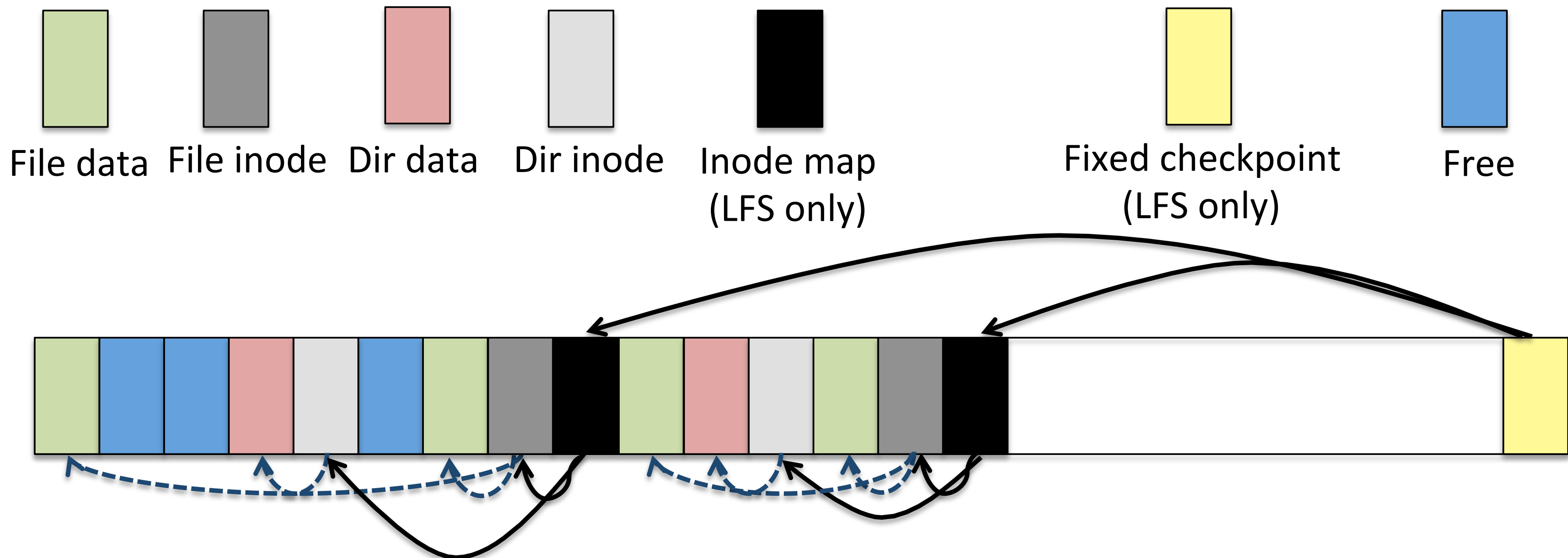






Disk cleaning

- When disk runs low on free space
  - Run a disk cleaning process
  - Compacts live information to contiguous blocks of disk



In reality, too expensive to clean contiguously.
FS is split into moderately large segments (e.g., 1MB or more).
Segments close to being full untouched. So, segment sized holes are allowed.

Disk cleaning

- When disk runs low on free space
  - Run a disk cleaning process
  - Compacts live information to contiguous blocks of disk
- Problem: long-lived data repeatedly copied over time
  - Solution: Group older files into same segment
  - Old segments won't have many changes. Skip.
  - But when old segment does have space, prioritize it. Why?
- Try to run cleaner when disk is not being used
- LFS: neat idea, influential
  - Paper on LFS one of the most widely cited OS paper
  - Many real file systems based on the idea