FFS Recovery: Soft Updates

- Learning Objectives
  - Explain how to enforce write-ordering without synchronous writes.
  - Identify and discuss tradeoffs between synchronous updates and soft updates.
- Topics
  - Recap
  - Deriving soft updates
  - Soft updates challenges and solutions

Recap FFS

- Synchronous writes to ensure that things can be made consistent.
- FSCK to check and repair on-disk state to make things consistent.
- Two fundamental problems:
  - Synchronous updates make FFS slow
  - Running fsck on a large file system, before you can do anything else makes recovery unacceptably slow.
- What can you do?
  - Recall why we performed synchronous writes and see if we can get around them.

Improving FFS

- Why did we perform synchronous writes?

Improving FFS

- Why did we perform synchronous writes?
  - To ensure certain ordering constraints.
- What alternatives are there to these ordering constraints?

Improving FFS

- Why did we perform synchronous writes?
  - To ensure certain ordering constraints.
- What alternatives are there to these ordering constraints?
  - Journaling
  - Enforce the ordering in the kernel (buffer cache).

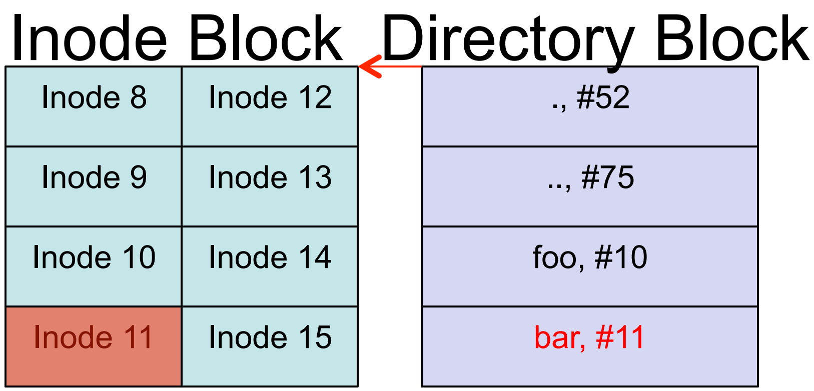
Approach 3: Soft Updates

- What are the synchronous updates really doing?
- Enforcing:
  - Never point to a structure before it has been initialized.
  - Never reuse a resource before invalidating all previous references to it.
  - Never reset the last pointer to a live resource before a new pointer has been set.
- Principles:
  - Prioritize latency over durability: buffer writes and ensure recoverability rather than pushing writes to disk synchronously.
  - Applications should never wait for a disk write unless they explicitly ask to do so.
  - Propagate data to disk using the minimum number of I/Os possible.
  - Minimize memory requirements.
  - Avoid constraining cache write-back and disk ordering (enable intelligent disk scheduling).

Soft Updates, Ganger 1994, Ganger, McKusick et al 2000

The Original Soft Updates

- Maintain dependency information between blocks in the buffer cache.
- Make sure that blocks are flushed to disk in an order that preserves those dependencies.

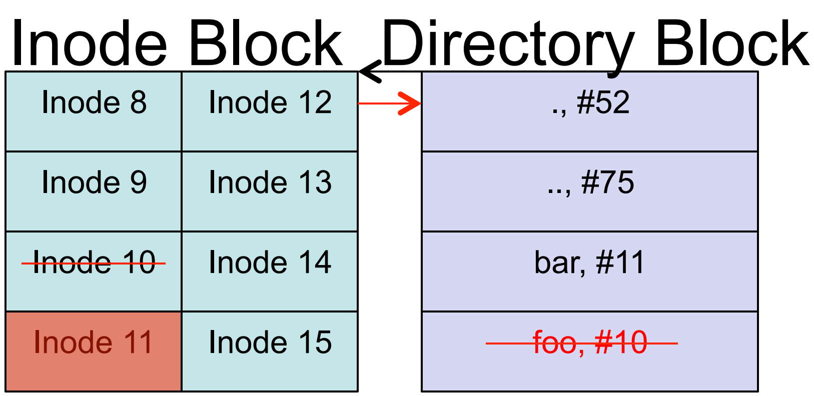


Create file bar

Create ordering dependency

The Problem

- Maintain dependency information between blocks in the buffer cache.
- Make sure that blocks are flushed to disk in an order that preserves those dependencies.



Create file bar

Remove file foo

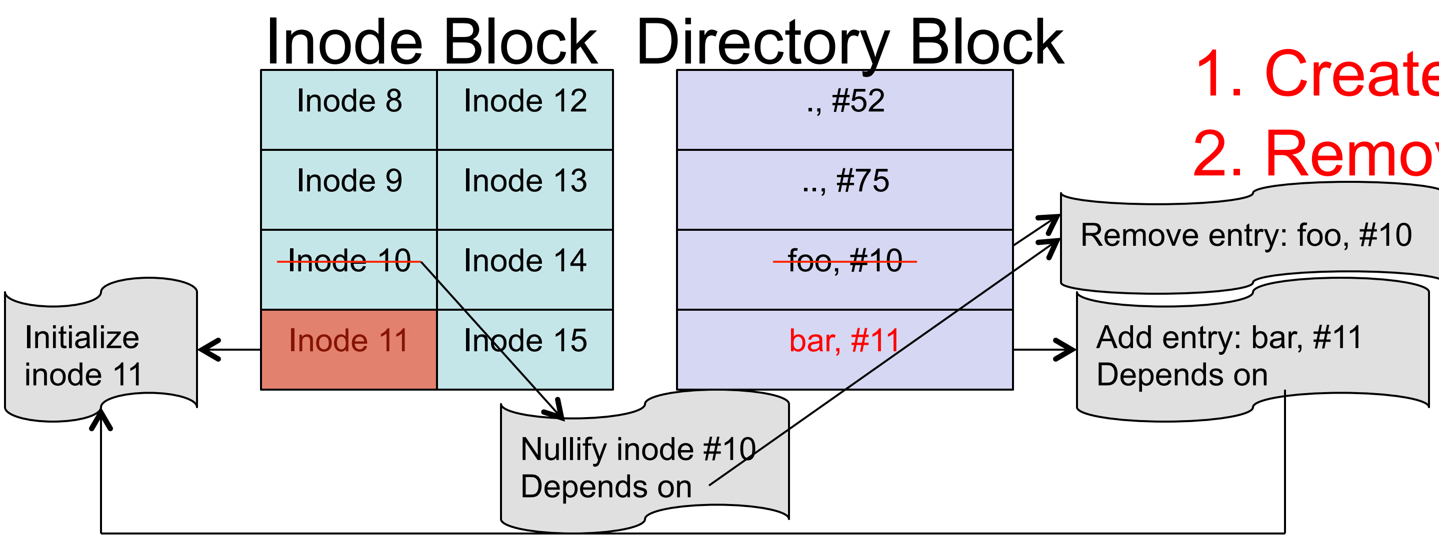
Create ordering dependency

Create ordering dependency

Which block do you write first?

The Solution

- Maintain fine-grain dependency information
  - Maintain dependencies on a per pointer or per-field basis.
  - In addition, keep “before” and “after” versions, so that you can undo an update before writing a block and then redo the update to preserve it in-memory.

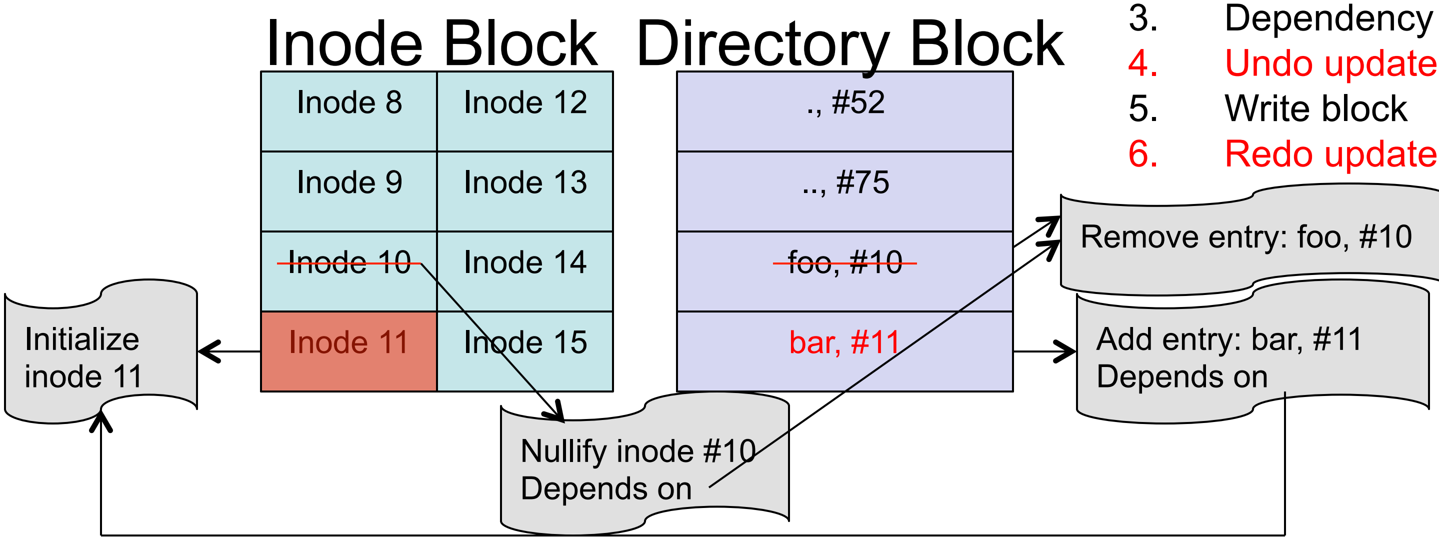


1. Create file bar
2. Remove file foo

The Solution: Writing the Inode Block

- Maintain fine-grain dependency information
  - Maintain dependencies on a per pointer or per-field basis.
  - In addition, keep "before" and "after" versions, so that you can undo an update before writing a block and then redo the update to preserve it in-memory.

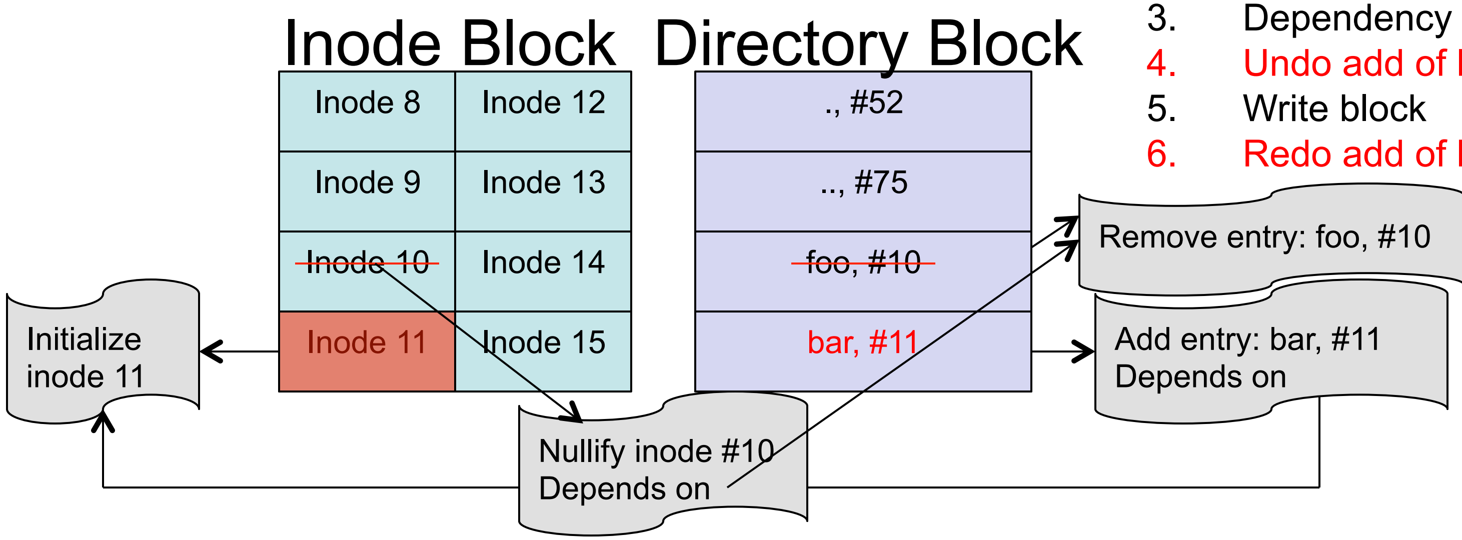
1. Check block for dependencies.
2. No dependency on inode #11
3. Dependency for inode #10
4. Undo update on inode #10
5. Write block
6. Redo update on inode #10



The Solution: Writing Directory Block

- Maintain fine-grain dependency information
  - Maintain dependencies on a per pointer or per-field basis.
  - In addition, keep "before" and "after" versions, so that you can undo an update before writing a block and then redo the update to preserve it in-memory.

1. Check block for dependencies.
2. No dependency on remove
3. Dependency on add
4. Undo add of bar, #11
5. Write block
6. Redo add of bar, #11



Soft Updates: Summary

- Fine-grain dependency tracking allows any block to be written at any time.
- Blocks involved in cycles may be written multiple times.
- Dirty blocks not in cycles written once.
- Post-crash on-disk state is always consistent, except possibly for bitmaps.
  - Order bitmap operations so bitmaps are always conservative: may think a block is allocated that isn't, but never thinks a block is free when it isn't.
  - Bitmaps can be reclaimed in the background.
  - No long fsck before mounting file system.