File Systems: Recovery

- Learning Objectives
  - Identify ways that a file system can be corrupt after a crash.
  - Articulate approaches a file system can take to limit the kinds of failures that can occur.
  - Describe different approaches to recovering a file system after a crash.
  - Evaluate the tradeoffs between the different approaches.
- Topics
  - Identify ways a file system can be corrupt.
  - Figure out some approaches to avoiding corruption.
  - Things you can do in the system while it's running.
  - Things you do after a crash.

What kinds of bad things could happen?

- Individual data structures could be corrupted:
  - The superblock or cylinder group headers
  - Bitmaps could get trashed
  - Individual inodes could be in an invalid state
  - A directory could become corrupted
- Inconsistencies between data structures:
  - Directory entries that don't point to valid files
  - Files without directory entries
  - Files containing disk blocks to which they have not written data.
  - Files not containing disk blocks to which they have written data.
  - Data blocks not attached to any file
  - Disk blocks allocated but still in free list
  - Disk blocks unallocated but NOT in free list

What kinds of bad things could happen?

- Individual data structures could be corrupted:
  - The superblock or cylinder group headers
  - Bitmaps could get trashed
  - Individual inodes could be in an invalid state
  - A directory could become corrupted
- Inconsistencies between data structures:
  - Directory entries that don't point to valid files
  - Files without directory entries
  - Files containing disk blocks to which they have not written data.
  - Files not containing disk blocks to which they have written data.
  - Disk blocks allocated but still in free list
  - Disk blocks unallocated but NOT in free list

Remedies (1)

- Individual data structures could be corrupted:
  - The superblock or cylinder group headers
  - Bitmaps could get trashed
  - Individual inodes could be in an invalid state
  - A directory could become corrupted
- Inconsistencies between data structures
  - Directory entries that don't point to files
  - Files without directory entries
  - Files containing disk blocks to which they have written data.
  - Files not containing disk blocks to which they have written data.
  - Disk blocks allocated but still in free list
  - Disk blocks unallocated but NOT in free list

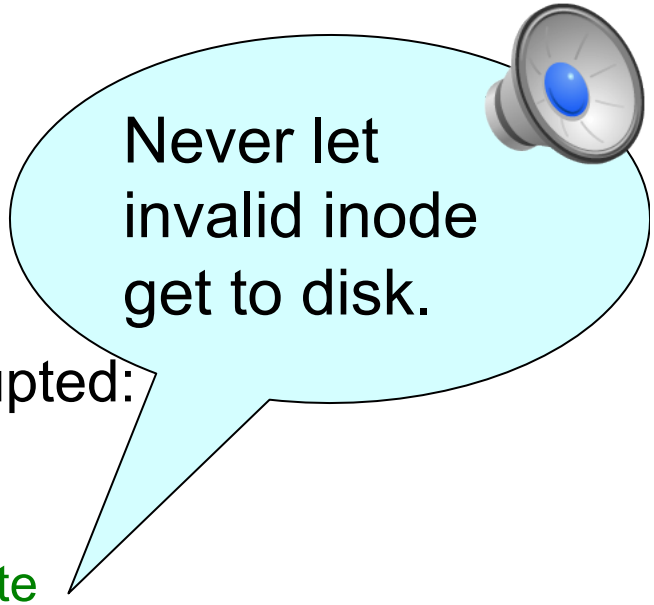
Avoid writing it.
Keep multiple copies of it.

Remedies (2)

- Individual data structures could be corrupted:
  - The superblock or cylinder group headers
  - Bitmaps could get trashed
  - Individual inodes could be in an invalid state
  - A directory could become corrupted
- Inconsistencies between data structures
  - Directory entries that don't point to files
  - Files without directory entries
  - Files containing disk blocks to which they have written data.
  - Files not containing disk blocks to which they have written data.
  - Disk blocks allocated but still in free list
  - Disk blocks unallocated but NOT in free list

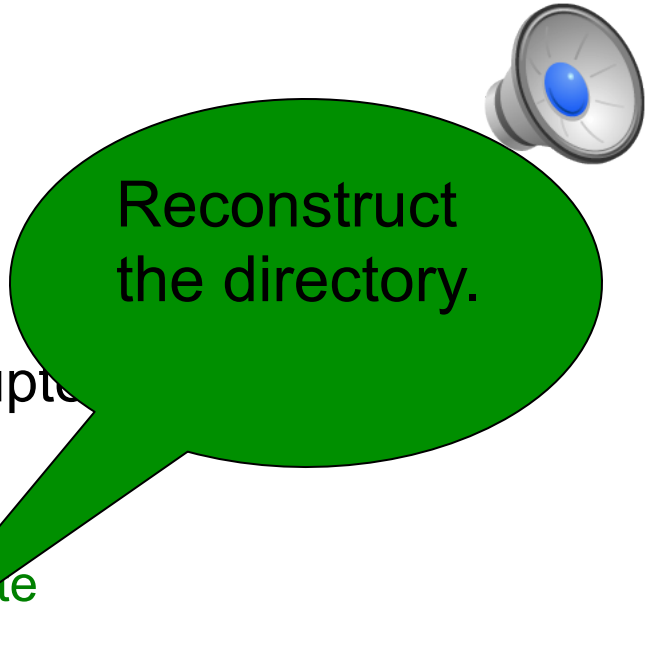
Rebuild them from inodes after a crash.

Remedies (3)



- Individual data structures could be corrupted:
  - The superblock or cylinder group headers
  - Bitmaps could get trashed
  - Individual inodes could be in an invalid state
  - A directory could become corrupted
- Inconsistencies between data structures:
  - Directory entries that don't point to valid files
  - Files without directory entries
  - Files containing disk blocks to which they have not written data.
  - Files not containing disk blocks to which they have written data.
  - Disk blocks allocated but still in free list
  - Disk blocks unallocated but NOT in free list

Remedies (4)



- Individual data structures could be corrupt
  - The superblock or cylinder group headers
  - Bitmaps could get trashed
  - Individual inodes could be in an invalid state
  - A directory could become corrupted
- Inconsistencies between data structures:
  - Directory entries that don't point to valid files
  - Files without directory entries
  - Files containing disk blocks to which they have not written data.
  - Files not containing disk blocks to which they have written data.
  - Disk blocks allocated but still in free list
  - Disk blocks unallocated but NOT in free list

Remedies (5)

- Individual data structures could be corru
  - The superblock or cylinder group headers
  - Bitmaps could get trashed
  - Individual inodes could be in an invalid state
  - A directory could become corrupted
- Inconsistencies between data structures:
  - Directory entries that don't point to valid files
  - Files without directory entries
  - Files containing disk blocks to which they have not written data.
  - Files not containing disk blocks to which they have written data.
  - Disk blocks allocated but still in free list
  - Disk blocks unallocated but NOT in free list

Remedies (6)

- Individual data structures could be corrupted:
  - The superblock or cylinder group headers
  - Bitmaps could get trashed
  - Individual inodes could be in an invalid state
  - A directory could become corrupted
- Inconsistencies between data structures:
  - Directory entries that don't point to valid files
  - Files without directory entries
  - Files containing disk blocks to w[illegible] they have not written data.
  - Files not containing disk blocks to[illegible]
  - Disk blocks allocated but still in[illegible]
  - Disk blocks unallocated but [illegible]

Read all the inodes; traverse directory tree; find all disconnected files.

Remedies (7)

- Individual data structures c
  - The superblock or cylinder
  - Bitmaps could get trashed
  - Individual inodes could be in an invalid st
  - A directory could become corrupted
- Inconsistencies between data structur
  - Directory entries that don't point to valid es
  - Files without directory entries
  - Files containing disk blocks to which they have not written data.
  - Files not containing disk blocks to which they have written data.
  - Disk blocks allocated but still in free list
  - Disk blocks unallocated but NOT in free list

Write new disk blocks before letting updated inodes get to disk.

Remedies (8)

- Individual data structures c
  - The superblock or cylinder
  - Bitmaps could get trashed
  - Individual inodes could be in an invalid st
  - A directory could become corrupted
- Inconsistencies between data structure
  - Directory entries that don't point to valid f
  - Files without directory entries
  - Files containing disk blocks to which the have not written data.
  - Files not containing disk blocks to which they have written data.
  - Disk blocks allocated but still in free list
  - Disk blocks unallocated but NOT in free list

Maybe this is OK: we allow some recent writes to be lost?

Remedies (9)

- Individual data structures could be corrupted:
  - The superblock or cylinder group headers
  - Bitmaps could get trashed
  - Individual inodes could be in an inva
  - A directory could become corrupted
- Inconsistencies between data struc
  - Directory entries that don't point to valid fi
  - Files without directory entries
  - Files containing disk blocks to which they e not written data.
  - Files not containing disk blocks to which ey have written data.
  - Disk blocks allocated but still in free list
  - Disk blocks unallocated but NOT in free list

You'll find these when you rebuild the bitmaps after a crash.

Remedies (10)

- Individual data structures could be corrupted:
  - The superblock or cylinder group headers
  - Bitmaps could get trashed
  - Individual inodes could be in an inva
  - A directory could become corrupted
- Inconsistencies between data struc
  - Directory entries that don't point to valid fi
  - Files without directory entries
  - Files containing disk blocks to which they ha t written data.
  - Files not containing disk blocks to which the ave written data.
  - Disk blocks allocated but still in free list
  - Disk blocks unallocated but NOT in free list

You'll find these when you rebuild the bitmaps after a crash.

Recovery Principles

- Do what is necessary on the live file system to ensure that after a failure, you can fix any inconsistencies that could happen.
- Have a recovery process that can fix up any remaining problems in the file system upon startup.
- Two key things we do:
  - Enforce ordering on when we write things to disk.
  - Use what we know about those orderings to fix/rebuild things at startup.

Which are ordering constraints?

- Individual data structures could be corrupted:
  - The superblock or cylinder group headers
  - Bitmaps could get trashed
  - Individual inodes could be in an invalid state
- Inconsistencies between data structures:
  - Directory entries that don't point to valid files
  - Files without directory entries
  - Files containing disk blocks to which they have not written data.
  - Files not containing disk blocks to which they have written data.
  - Data blocks not attached to any file
  - Disk blocks allocated but still in free list
  - Disk blocks unallocated but NOT in free list

Ordering constraint

Three Approaches to Enforcing Ordering Constraints

- Synchronously write things in order.
- Maintain dependencies in-memory and when it's necessary to write things, make sure they get written in order (called *soft updates*).
- Keep a *log* of all the things you do so that after a crash you can read through the log and figure out precisely what you have to do.

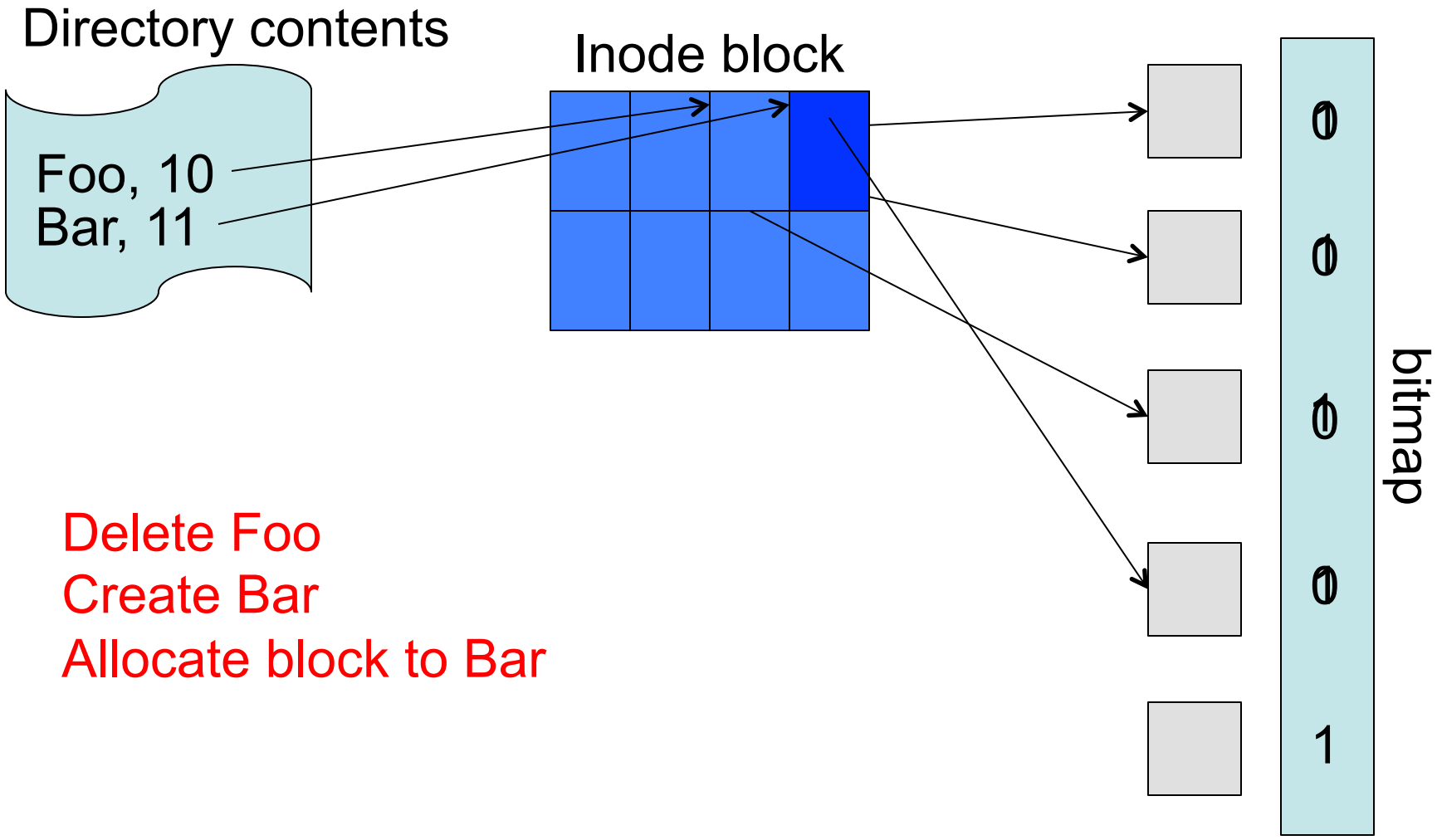
Approach 1: Synchronous Writes

- Goal is to ensure that you never write a pointer to something that has not been properly written/initialized:
  - Entries in directories reference valid inodes.
  - A block cannot belong to multiple files
  - Inodes are valid

Approach 1: Synchronous Writes

- Goal is to ensure that you never write a pointer to something that has not been properly written/initialized:
  - Entries in directories reference valid inodes.
  - A block cannot belong to multiple files
  - Inodes are valid
- Entries in directories reference valid inodes:
  - On create: synchronously write inode to disk before updating directory entry.
  - On delete: synchronously write the directory with the name removed before deallocating the inode.
- A block cannot belong to multiple files:
  - On unlink/truncate: synchronously write the deallocated (or truncated) inode to disk before its blocks are freed.
- Inodes are valid:
  - Fill in all inode fields before synchronous write (above).

Directories, Files, and Inodes (oh my)



Delete Foo
Create Bar
Allocate block to Bar

Now, what happens after a crash?

- The behavior of the file system ensures that it's possible to get to a consistent state after a crash, but it does not ensure that you are always in a consistent state.
- So, how do we make sure that the file system is in a consistent state after a crash?
- Well, first we have to define what consistent means…

FFS Consistency

- What does consistent mean in FFS?

FFS Consistency

- What does consistent mean in FFS?
  - Superblocks and cylinder groups have accurate counts.
  - Every directory entry references a valid inode.
  - Every valid inode appears in a number of directory entries equal to its link count.
  - Blocks allocated to valid inodes are marked inuse in bitmaps.
  - A block belongs to only one file/directory.
  - The first two entries in every directory are . and ..
  - What makes an inode valid?
    - Its length and number of blocks are consistent with the blocks allocated to it.
    - All block pointers are valid in the given file system.
    - If an inode references a directory, its size is a multiple of DIRBLKSIZ.
    - Its inode number is correct.

FSCK: The File System Checker

- The FFS fsck program fixes the following errors:
  - Unreferenced inodes
  - Improper link counts
  - Missing blocks in free map
  - Incorrect superblock counts
  - First two entries in a directory are not . and ..

FSCK Detail (1)

- FSCK analyzes the file system, reports inconsistencies and optionally fixes them:
  - Read superblock (indicates number of cylinder groups, file system block size, etc).
  1. Read cylinder group summary and every inode.
     a. Verify type (directory, file, etc).
     b. Verify size (does not exceed maximum file/directory size).
     c. Verify that blocks in file are set correctly in bit maps.
     d. Verify link count non zero.
     e. Verify size and block count.
     f. Verify fragment summary in cylinder.
  2. Verify directory hierarchy (BFS entire directory tree)
     a. Verify directory link counts.
     b. Verify directories contain . and .. and have valid references.
     c. Verify directory is appropriately sized.

FSCK Detail (2)

3. Iterate over all inodes (checking for proper connectivity)
   a. Verify that every directory we ever found has a valid parent.
4. Check block allocations & reference counts
   a. Verify proper block and fragment accounting and consistency with bitmaps.
   b. Verify that all the link counts are correct.
5. Check cylinder group meta-data
6. Check quotas