W4118: Linux file systems

Instructor: Junfeng Yang

References: Modern Operating Systems (3rd edition), Operating Systems Concepts (8th edition), previous W4118, and OS at MIT, Stanford, and UWisc

File systems in Linux

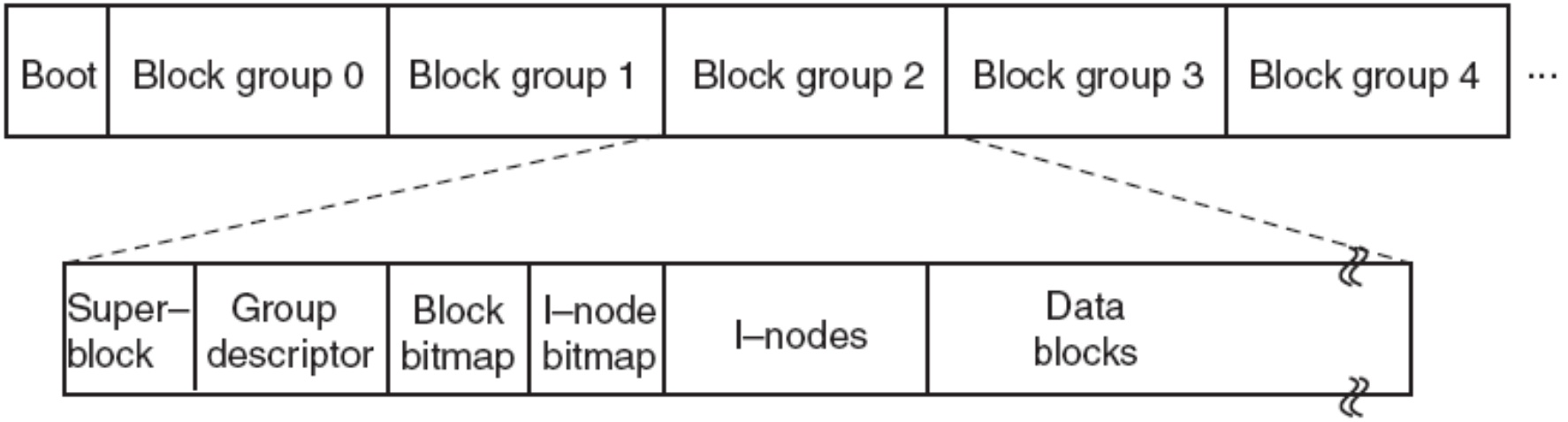
- Linux Second Extended File System (Ext2)
  - What is the EXT2 on-disk layout?
  - What is the EXT2 directory structure?
- Linux Third Extended File System (Ext3)
  - What is the file system consistency problem?
  - How to solve the consistency problem using journaling?
- Virtual File System (VFS)
  - What is VFS?
  - What are the key data structures of Linux VFS?

Ext2

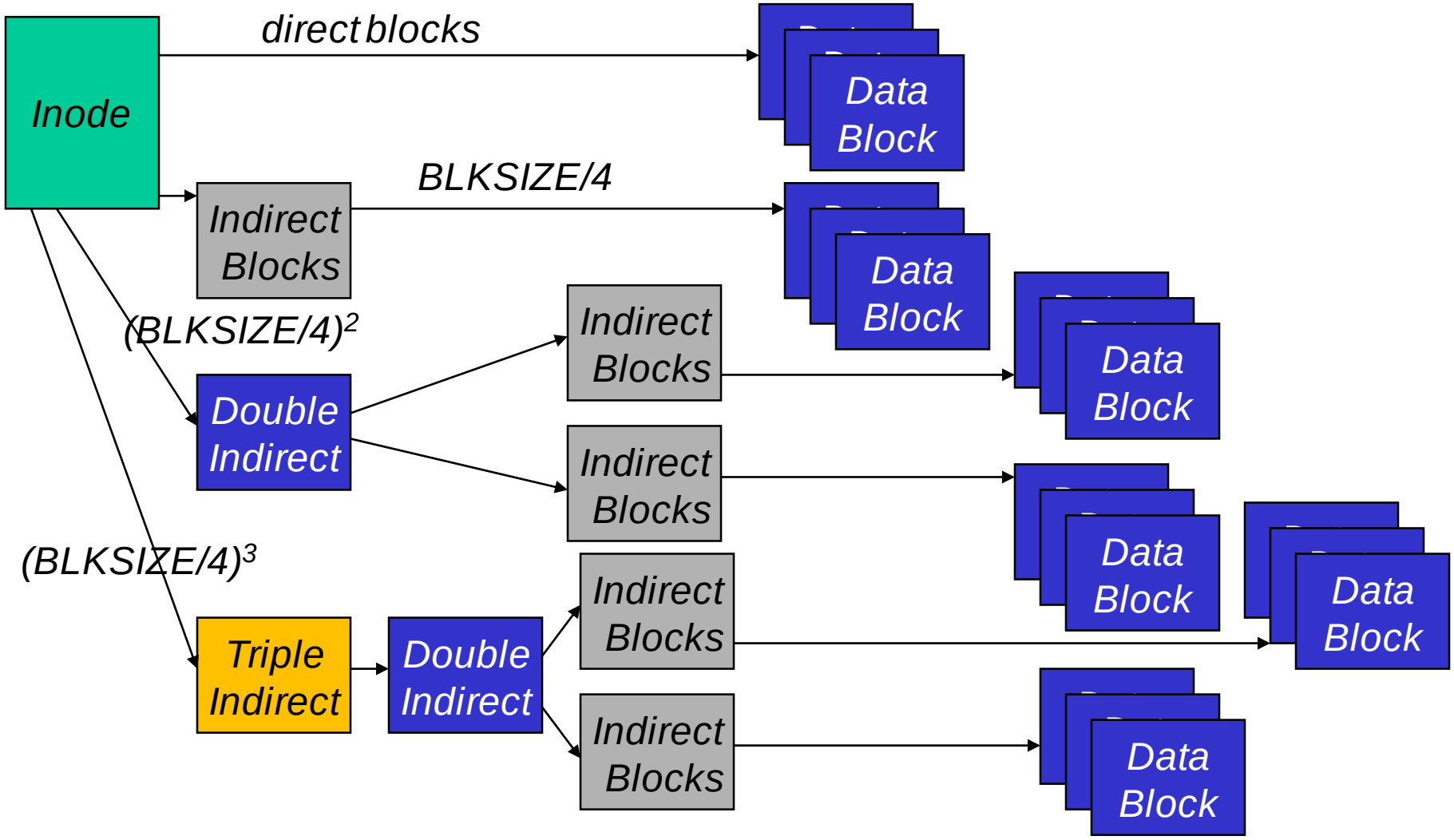
- "Standard" Linux File System
  - Was the most commonly used before ext3 came out
- Uses FFS-like layout
  - Each FS is composed of identical block groups
  - Allocation is designed to improve locality
- inodes contain pointers (32 bits) to blocks
  - Direct, Indirect, Double Indirect, Triple Indirect
  - Maximum file size: 4.1TB (4K Blocks)
  - Maximum file system size: 16TB (4K Blocks)
- On-disk structures defined in include/linux/ext2_fs.h

Ext2 Disk Layout

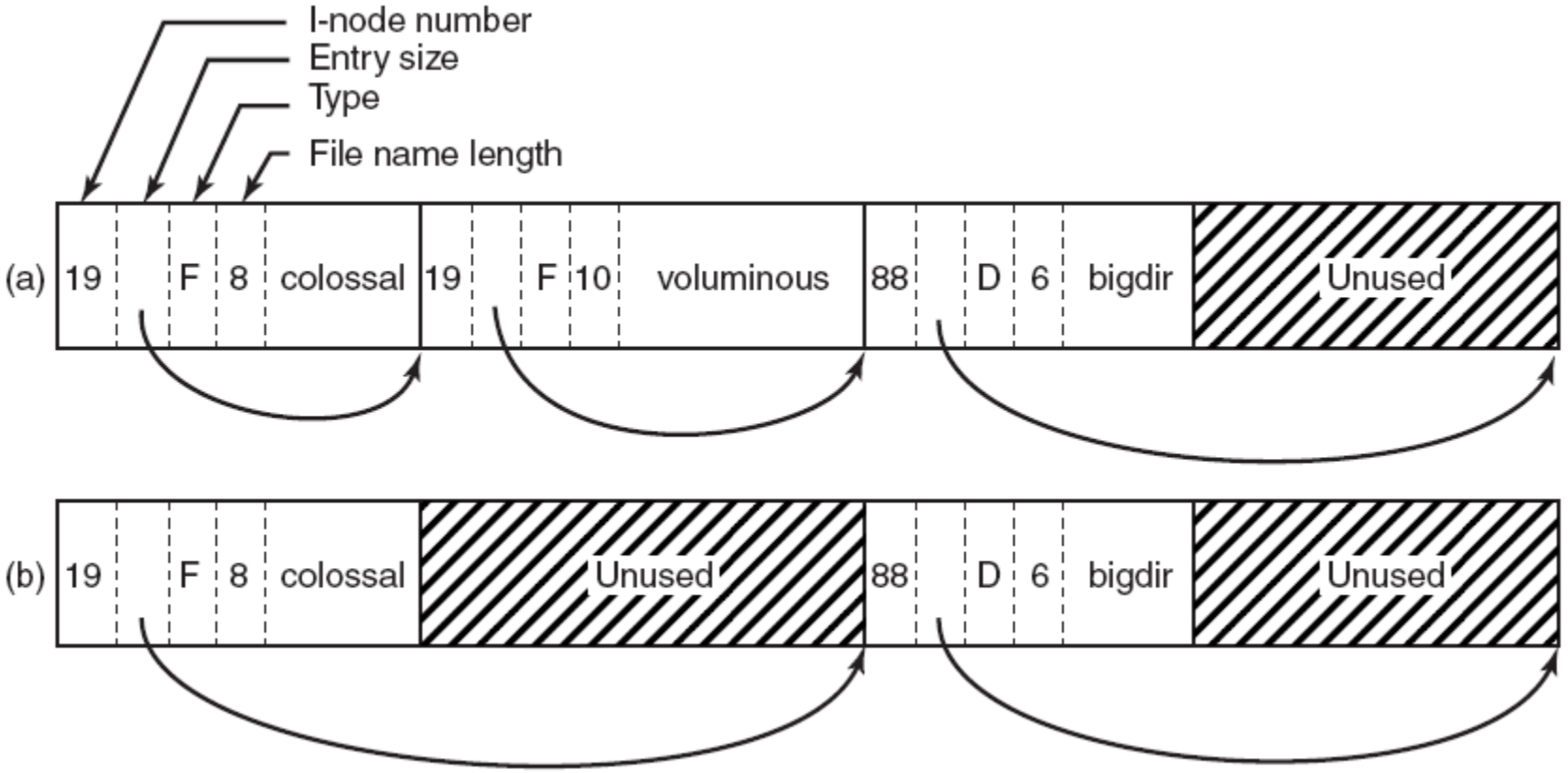
- Files in the same directory are stored in the same block group
- Files in different directories are spread among the block groups



Block Addressing in Ext2



Ext2 Directory Structure



(a) A Linux directory with three files

(b) After the file voluminous has been removed

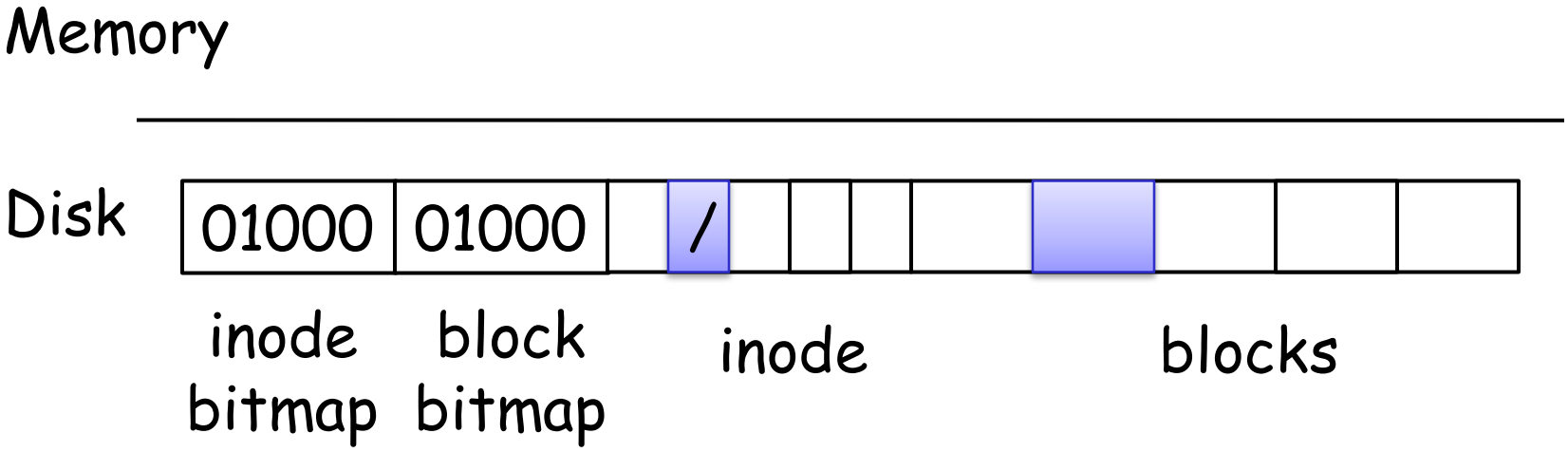
File systems in Linux

- Linux Second Extended File System (Ext2)
  - What is the EXT2 on-disk layout?
  - What is the EXT2 directory structure?
- Linux Third Extended File System (Ext3)
  - What is the file system consistency problem?
  - How to solve the consistency problem using journaling?
- Virtual File System (VFS)
  - What is VFS?
  - What are the key data structures of Linux VFS?

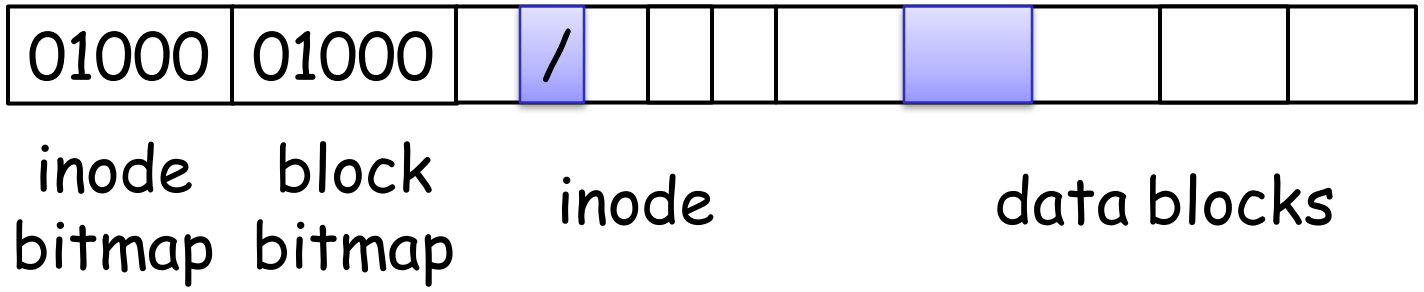
The consistent update problem

- 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

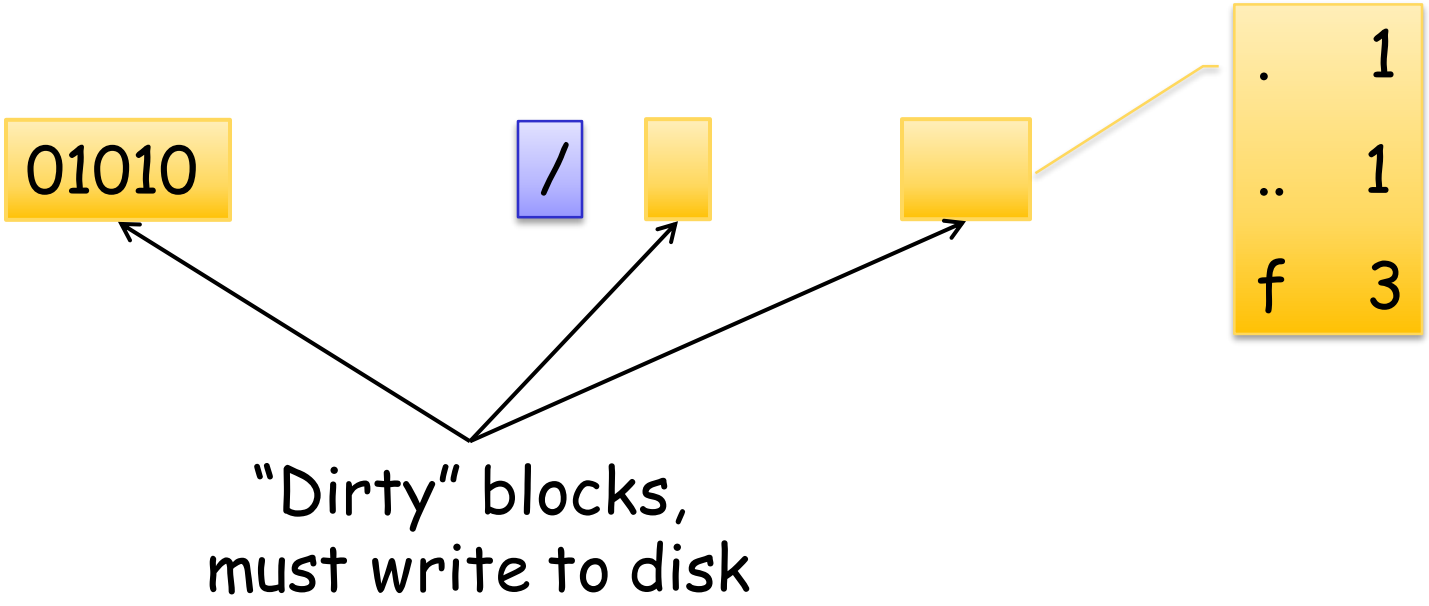
Example: Ext2 File Creation

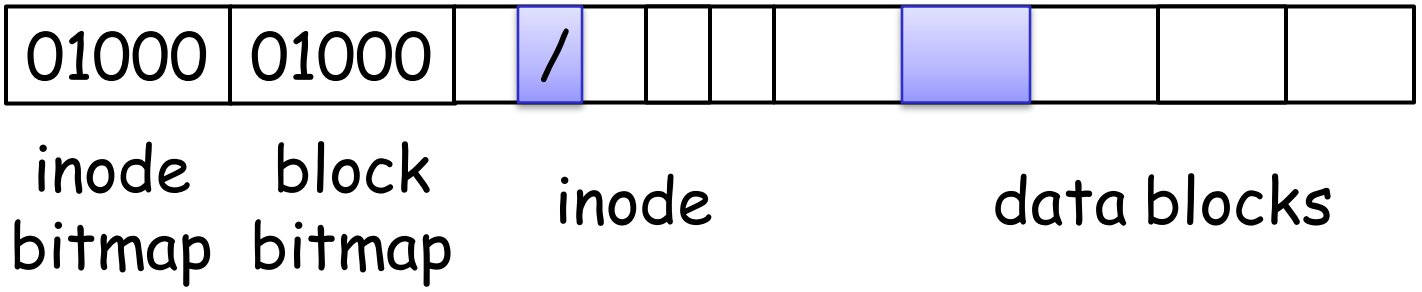


Read to In-memory Cache



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

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
  - B′ I D
  - B I′ D
  - B I D′
  - B′ I′ D
  - B′ I D′
  - B I′ D′
  - B′ I′ D′

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' I D')
  - Not well-defined consistency

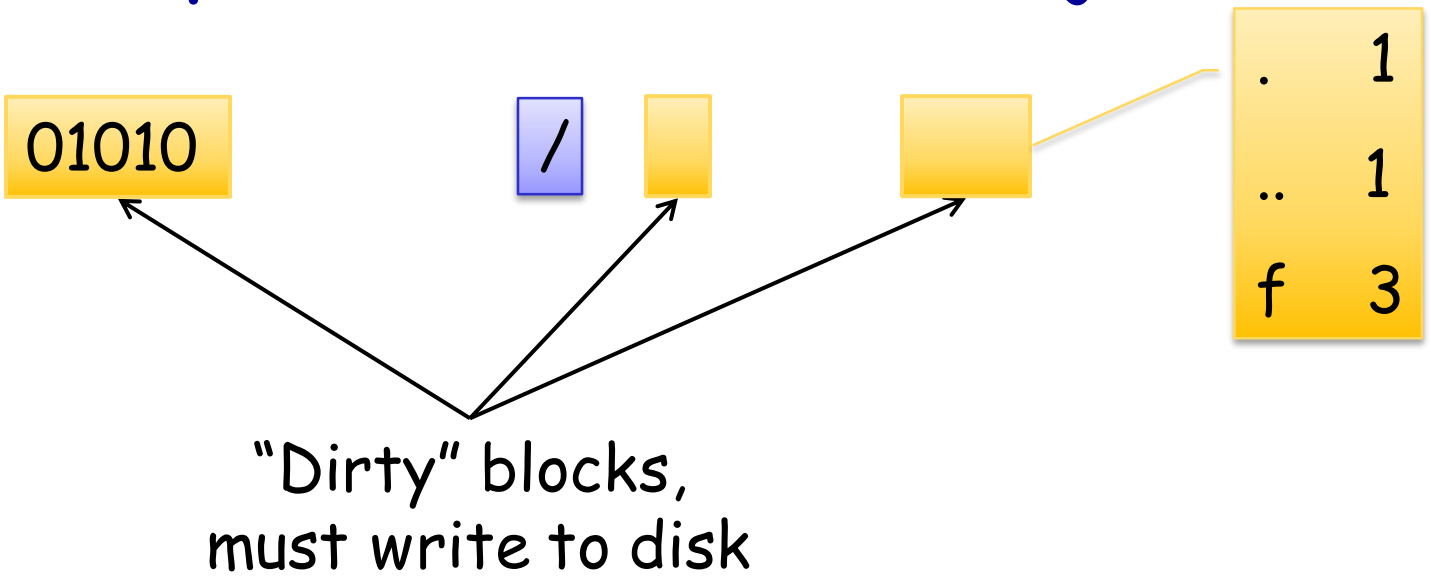
Another solution: Journaling

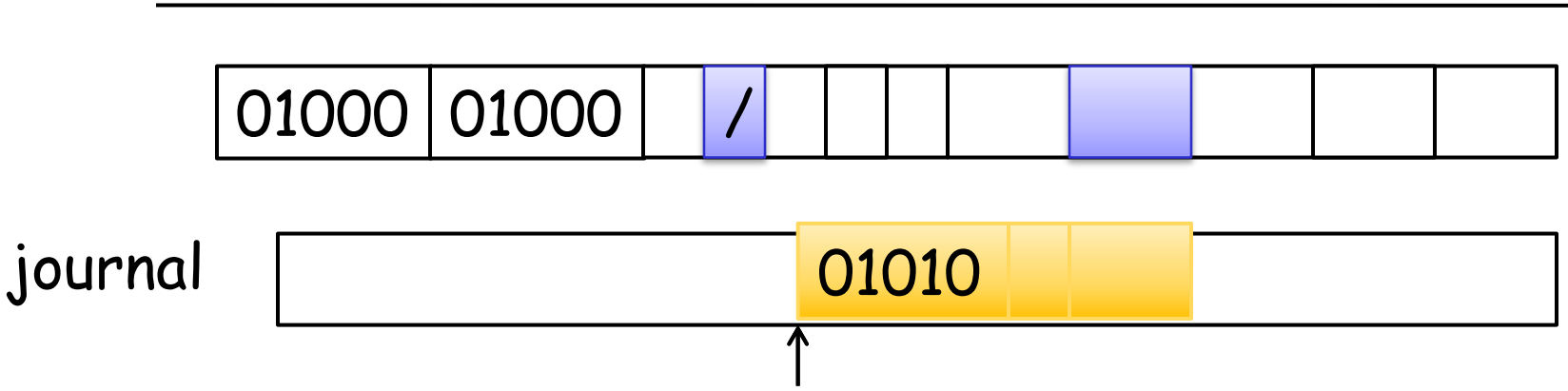
- Write-ahead logging from database community
- Persistently write intent to log (or journal), then update file system
  - Crash before intent is written == no-op
  - Crash after intent is written == redo op
- Advantages
  - no need to scan entire disk
  - Well-defined consistency

Ext3 Journaling

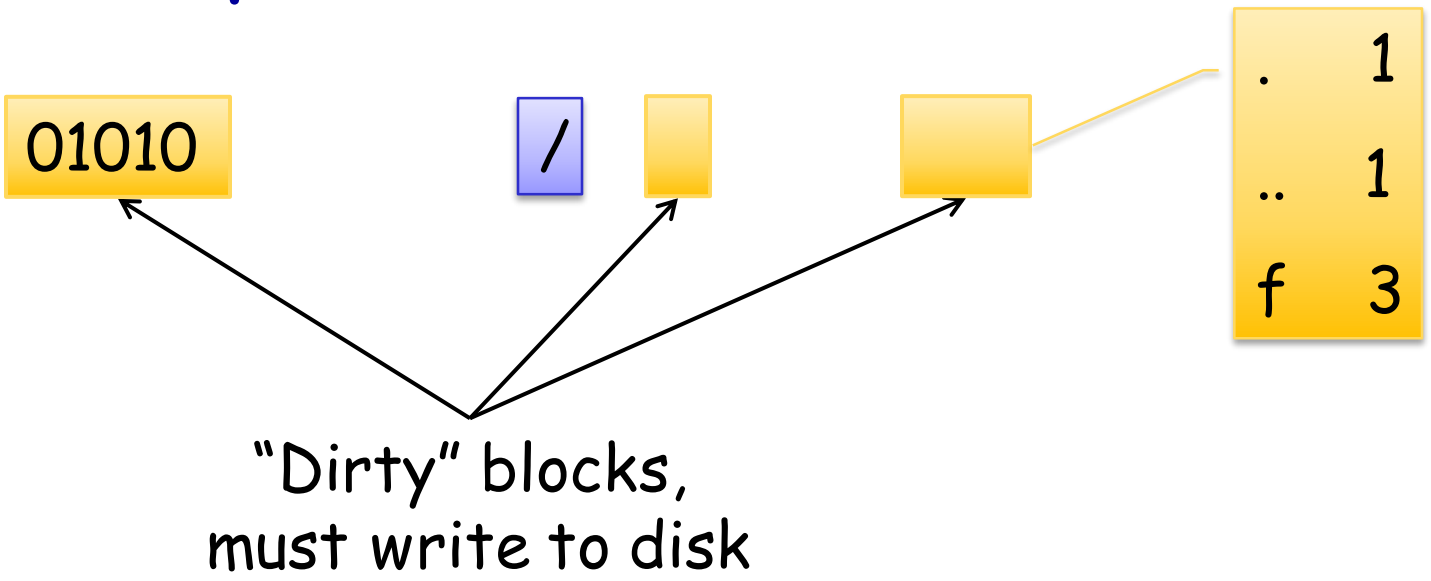
- Physical journaling: write real block contents of the update to log
  - Four totally ordered steps
    - Commit dirty blocks to journal as one transaction
    - Write commit record
    - Write dirty blocks to real file system
    - Reclaim the journal space for the transaction
- Logical journaling: 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

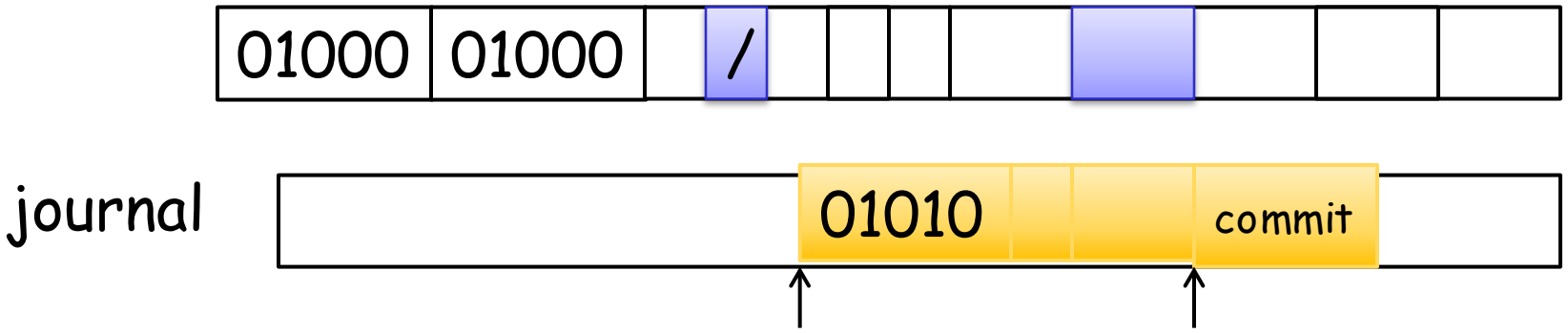
Step 1: write blocks to journal



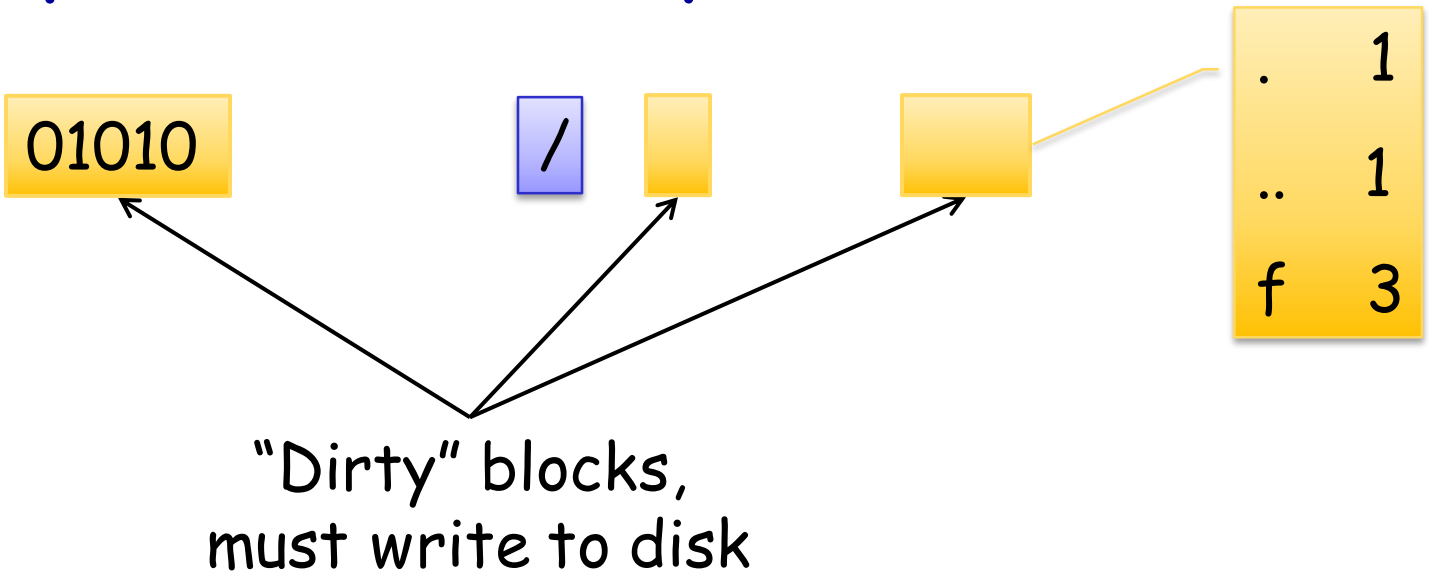


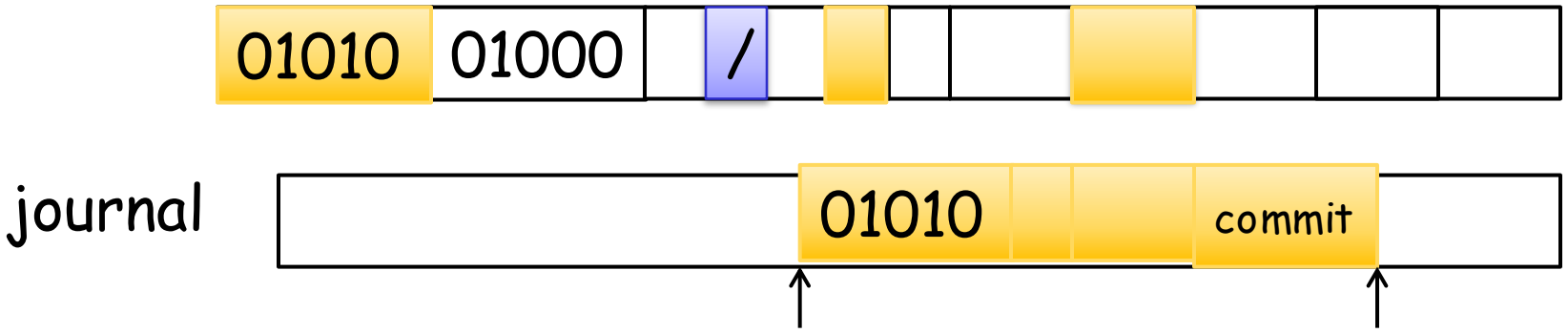
Step 2: write commit record



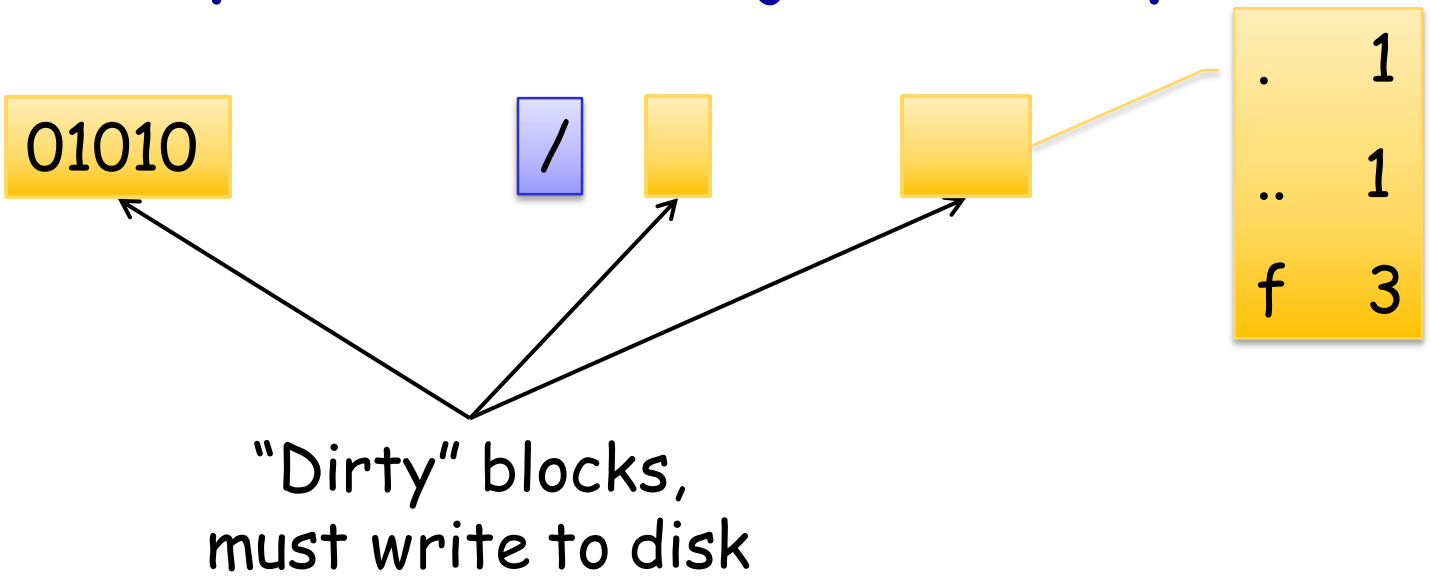


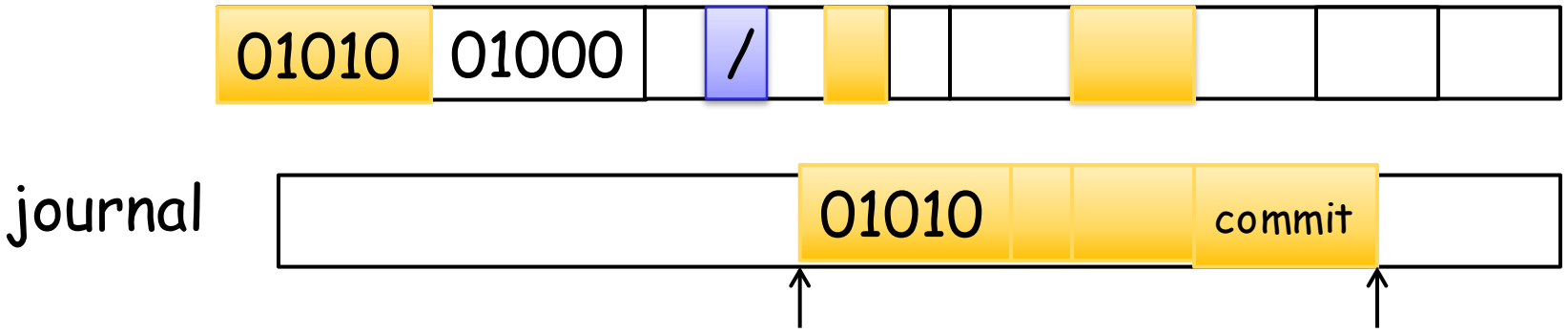
Step 3: write dirty blocks to real FS





Step 4: reclaim journal space





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 has cost
  - 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

File systems in Linux

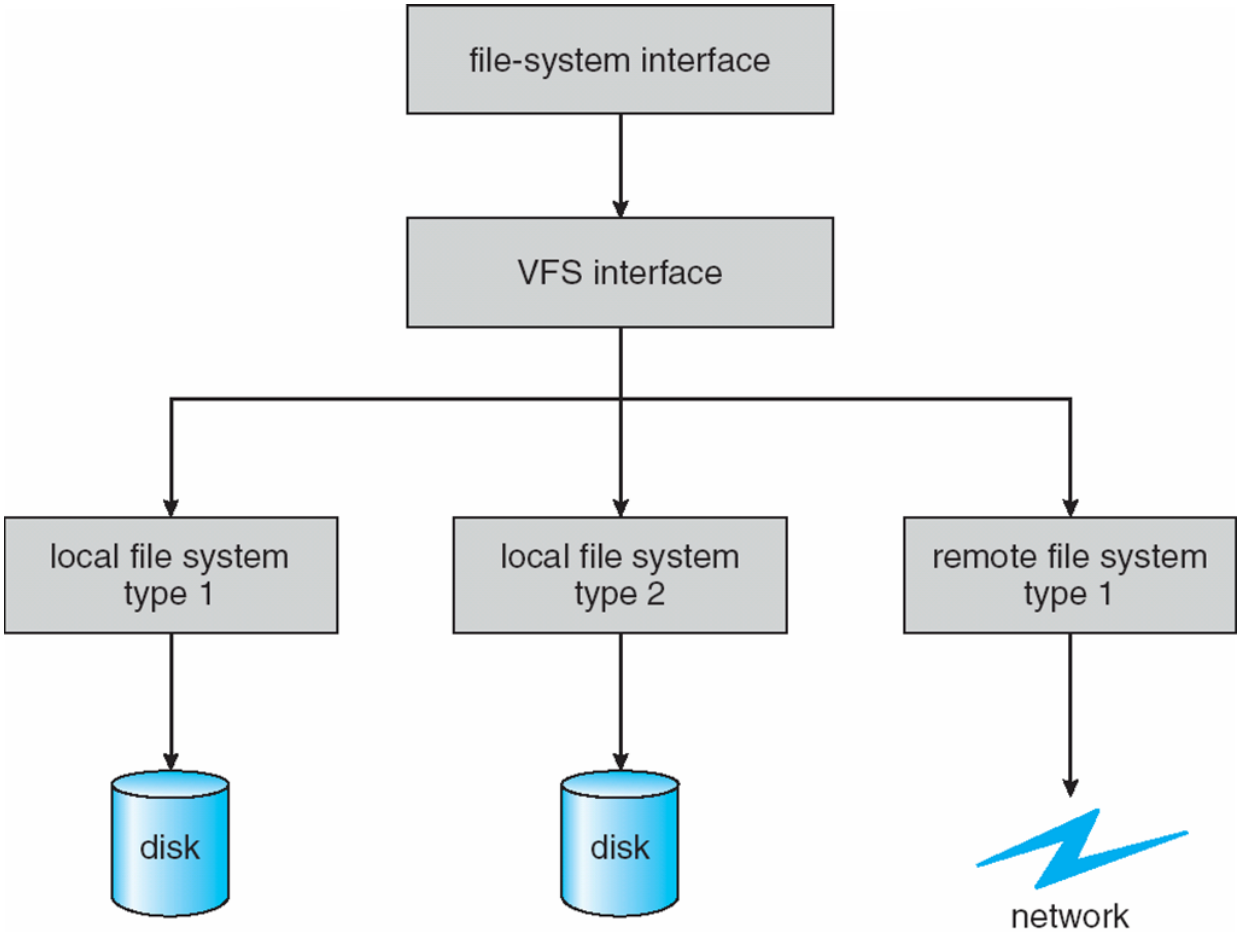
- Linux Second Extended File System (Ext2)
  - What is the EXT2 on-disk layout?
  - What is the EXT2 directory structure?
- Linux Third Extended File System (Ext3)
  - What is the file system consistency problem?
  - How to solve the consistency problem using journaling?
- Virtual File System (VFS)
  - What is VFS?
  - What are the key data structures of Linux VFS?

VFS

- Old days: "the" file system
- Nowadays: many file system types and instances co-exist

- VFS: an FS abstraction layer that transparently and uniformly supports multiple file systems
  - A VFS specifies an interface
  - A specific FS implements this interface
    - Often a struct of function pointers
  - VFS dispatches FS operations through this interface
    - E.g., dir->inode_op->mkdir();

Schematic View of Virtual File System



Key Linux VFS Data Structures

- *struct file*
  - information about an open file
  - includes current position (file pointer)
- *struct dentry*
  - information about a directory entry
  - includes name + inode#
- *struct inode*
  - unique descriptor of a file or directory
  - contains permissions, timestamps, block map (data)
  - inode#: integer (unique per mounted filesystem)
  - Pointer to FS-specific inode structure
    - e.g. *struct ext2_inode_info*
- *struct superblock*
  - descriptor of a mounted filesystem