Case study: ext3 FS

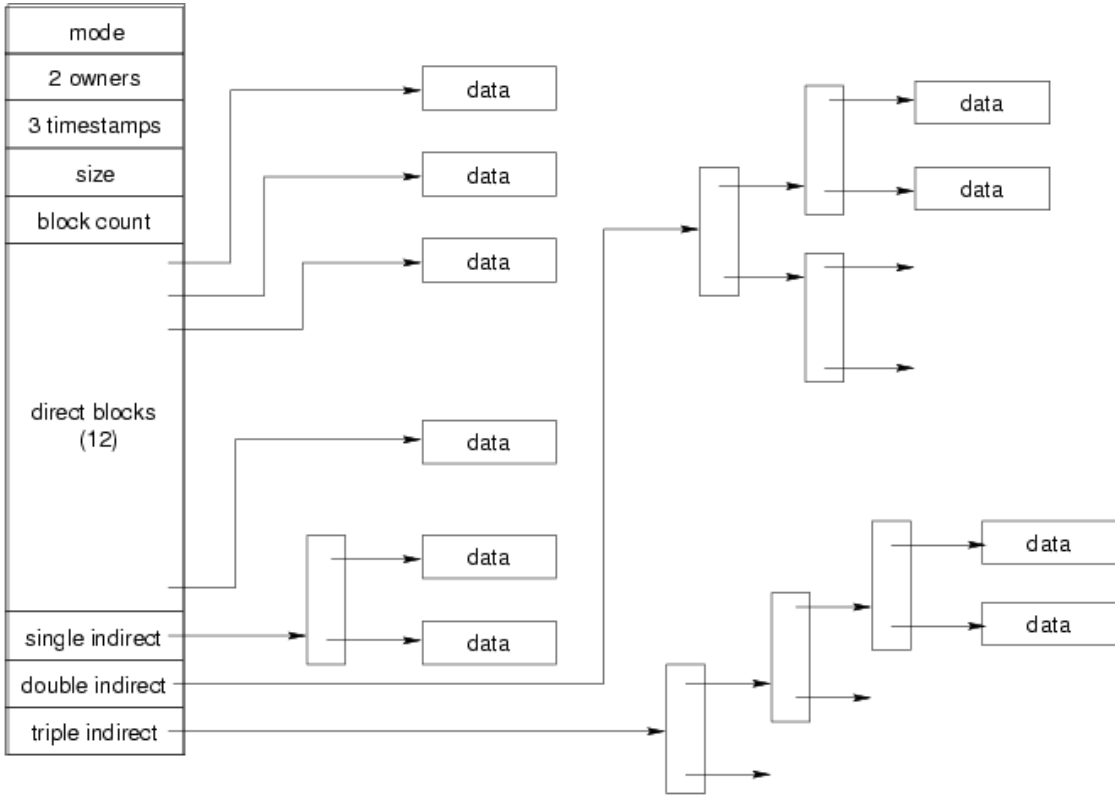
ext2: Some Commentary

- A file system is just a data structure.
  - Lays out logical data in a “memory”.
  - Custom data formats specified for inodes, dirs, etc.
- Design issues look like other structures.
  - Depth of lookup trees.
- Provides its own primitives.
  - Allocation strategy.

ext2: Prefix Trees

- Recall the lookup process for double/triple indirect blocks

- This is a prefix tree
  - Similar to a “trie”
  - Index each node by digits of the key

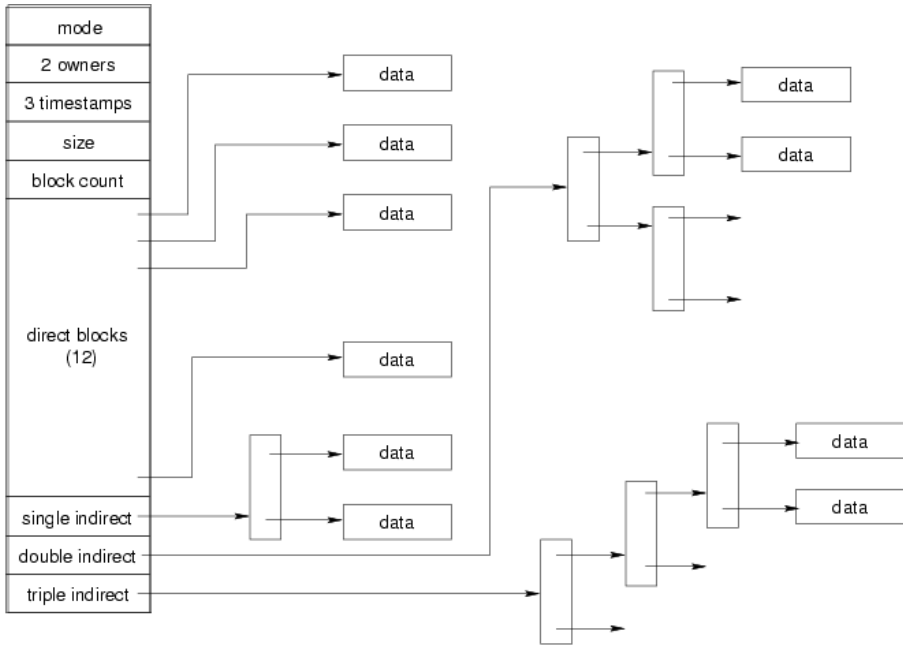


- Also used in virtual memory
- Once used in telephones
  - (02) 94XXXXXX

More Prefix Trees

- File system lookup is itself a prefix tree scheme.

- Look up:
  - “os_3231/lectures/26t2/”
  - “/lib/x86_64-linux-gnu/libc.so.6”



ext2: Prefix Tree Calculation

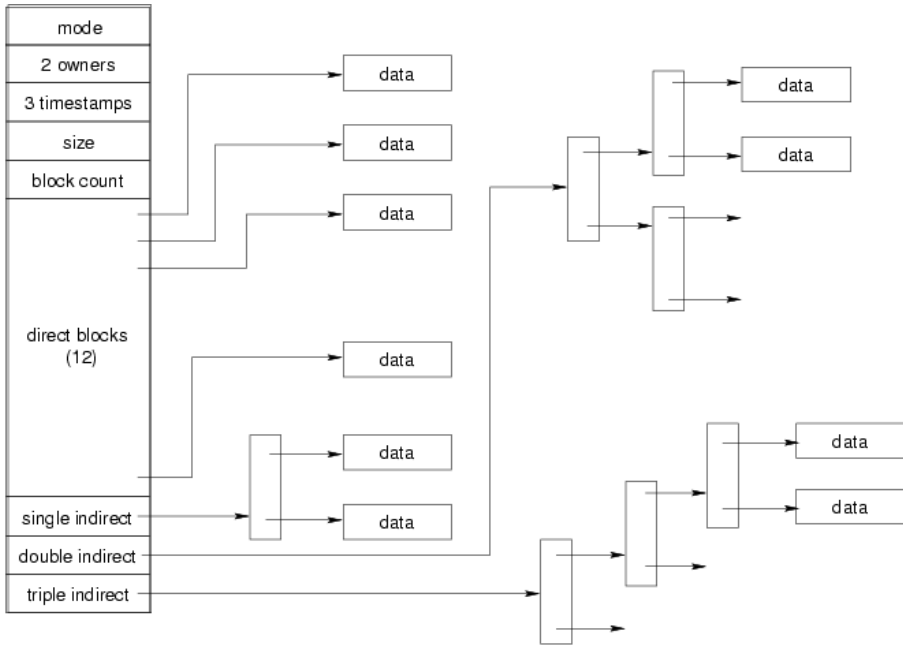
Where is byte number 17000000?

The double-indirect space begins at byte number 4243456.

Its offset in this space is 12756544, 0b110000101010011001000000.

This can be split into:

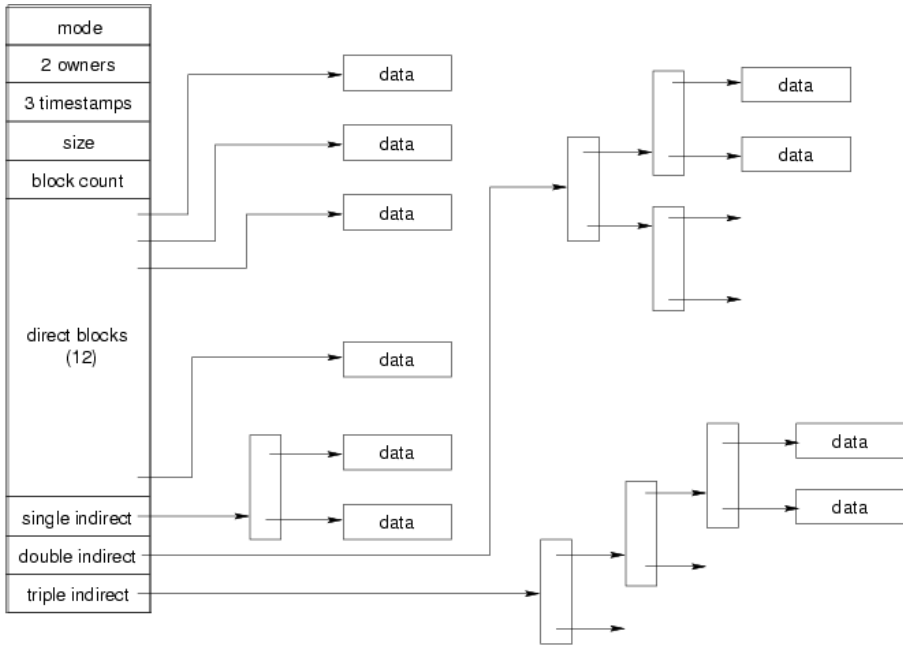
- 0b011: index into DI block
- 0b0000101010: index into SI block
- 0b011001000000: address of byte in data block



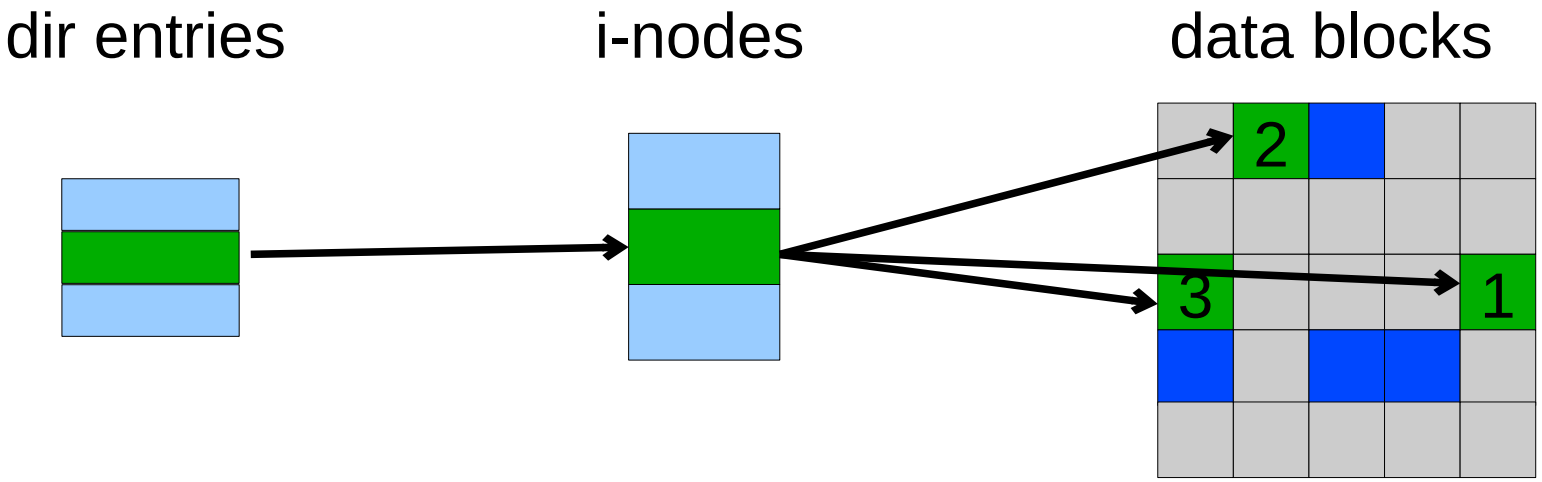
Commentary: Alternative Lookups

Could we use another standard data structure to encode the block number to block ID map?

- A hash table?
- A balanced tree?
  - What would rebalancing look like?
- A more standard prefix tree?



Brief Journaling Intro

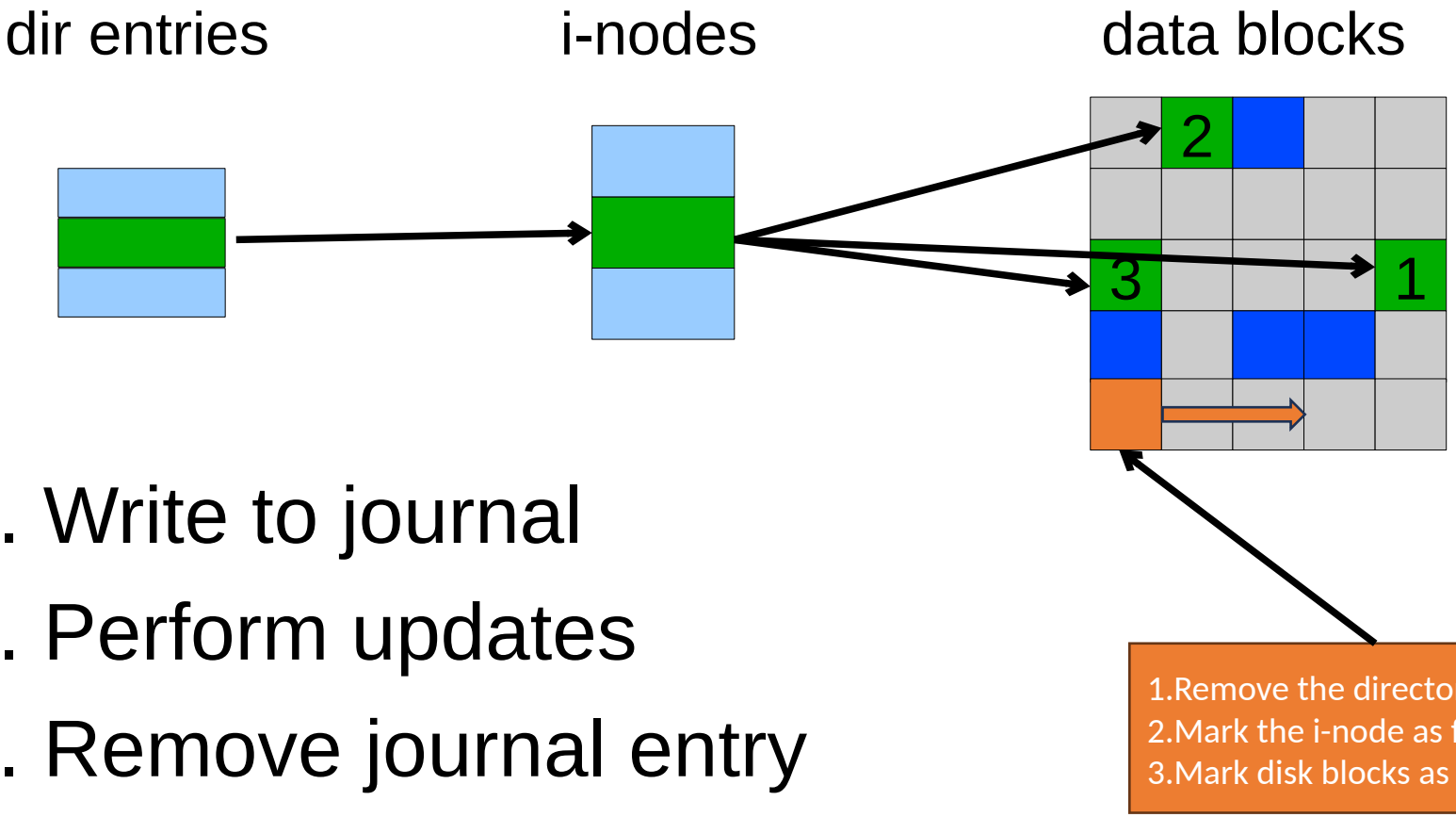


- Example: deleting a file
  1. Remove the directory entry
  2. Mark the i-node as free
  3. Mark disk blocks as free

Concurrency in File Systems

- There is no concurrent access to a file system.
  - No OS would try to share an active file system.
- Consistency issues are a bit like concurrency issues.
  - Two instances of an OS are accessing stored data.
  - The previous instance has shut down, and the new instance is executing in a different “thread”.
  - In this case, there is no way to wait for the previous instance.

Brief Journaling Intro



1. Write to journal
2. Perform updates
3. Remove journal entry

The ext3 file system

- Design goals
  - Add journaling capability to the ext2 FS
  - Backward and forward compatibility with ext2
    - Existing ext2 partitions can be mounted as ext3
  - Leverage the proven ext2 performance
  - Reuse most of the ext2 code base
  - Reuse ext2 tools, including e2fsck

Adding a journal

Option1: Journal FS data structure updates

- Example:
  - **Start transaction**
  - Delete dir entry
  - Delete i-node
  - Release blocks 32, 17, 60
  - **End transaction**

Option2: Journal disk block updates

- Example:
  - **Start transaction**
  - Update block #n1 *(contains the dir entry)*
  - Update block #n2 *(i-node allocation bitmap)*
  - Update block #n3 *(data block allocation bitmap)*
  - **Add transaction**

Question: which approach is better?

The ext3 journal

Option1: Journal FS data structure updates

- ✔ Efficient use of journal space; hence faster journaling
- ✘ Individual updates are applied separately
- ✘ The journaling layer must understand FS semantics

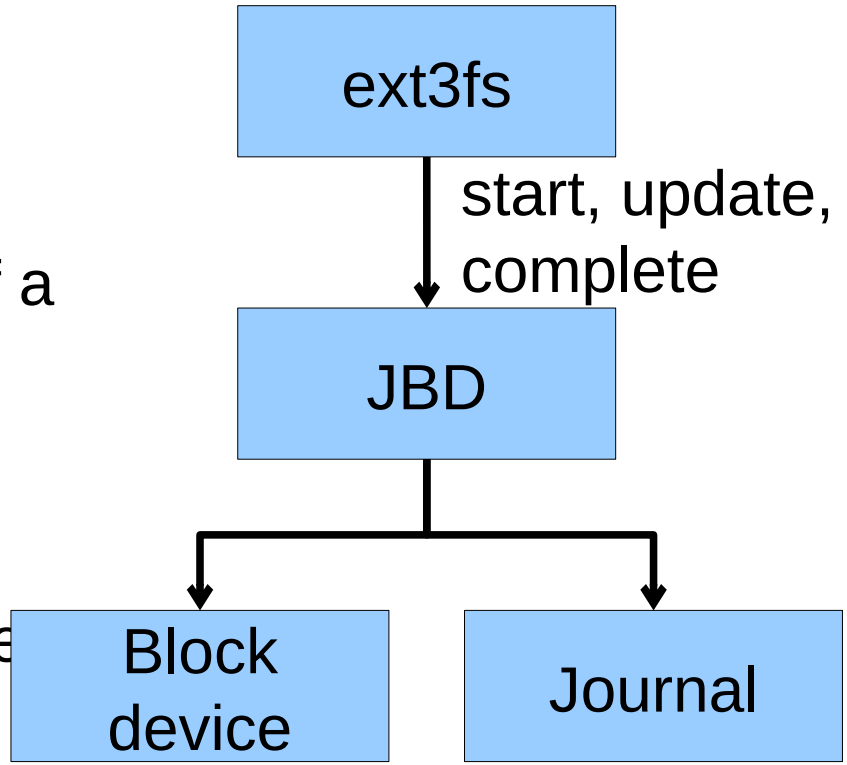
Option2: Journal disk block updates

- ✗ Even a small update adds a whole block to the journal
- ✔ Multiple updates to the same block can be aggregated into a single update
- ✔ The journaling layer is FS-independent (easier to implement)

Ext3 implements Option 2

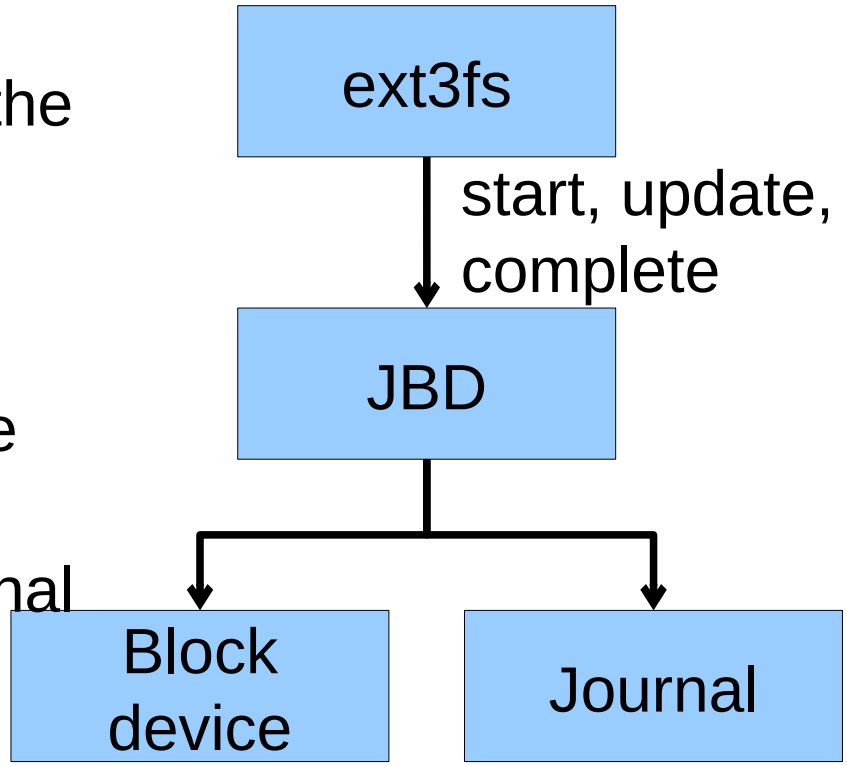
The Journaling Block Device

- The ext3 journaling layer is called Journaling Block Device (JBD)
- JBD interface
  - Start a new transaction
  - Update a disk block as part of a transaction
    - Repeat if necessary
  - Complete a transaction
    - Completed transactions are buffered in RAM

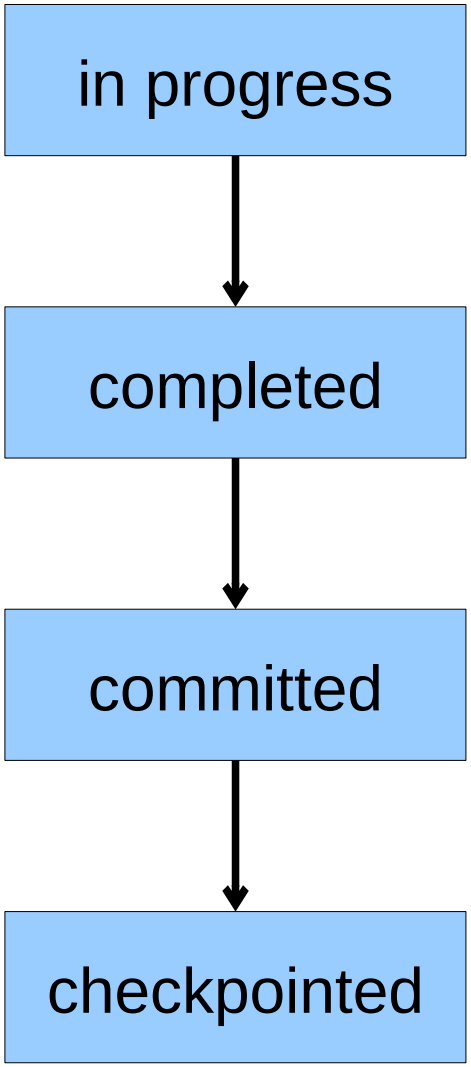


The Journaling Block Device

- JBD interface (continued):
  - Commit
    - Write transaction data to the journal
  - Checkpoint
    - Perform the journalled transactions on the device
    - Finished transactions can be removed from the journal
    - Used on unmount



Transaction Life-Cycle



Updates are buffered in RAM

Updates are buffered in RAM; no additional updates are allowed in the same transaction

Updates are written to the journal and marked as committed. Transaction can be replayed after an unclean unmount

Updates are written to the file system; the transaction is removed from the journal

Journaling Modes

- Ext3 supports two journaling modes
  - Metadata + data
    - Enforces atomicity of all FS operations
  - Metadata journaling
    - Metadata is journalled
    - Data blocks are written directly to the disk
    - Improves performance
    - Enforces file system integrity
    - Does not enforce atomicity of `write`'s
      - New file content can be stale blocks

JBD

- JBD can keep the journal on a block device or in a file
  - Enables compatibility with ext2 (the journal is just a normal file)
- JBD is independent of ext3-specific data structures
  - Separation of concerns
    - The FS maintains on-disk data and metadata
    - JBD takes care of journaling
  - Code reuse
    - JBD can be used by any other FS that requires journaling

This Lecture: Done with File Systems

- We’re now done with the file-system content of the course
- The user-facing file API
- The system layers:
  - File tables
  - VFS
  - File Systems
  - Buffer Cache
- Locality and performance
- Consistency issues and solutions