Case study: ext3 FS

Some commentary on ext2

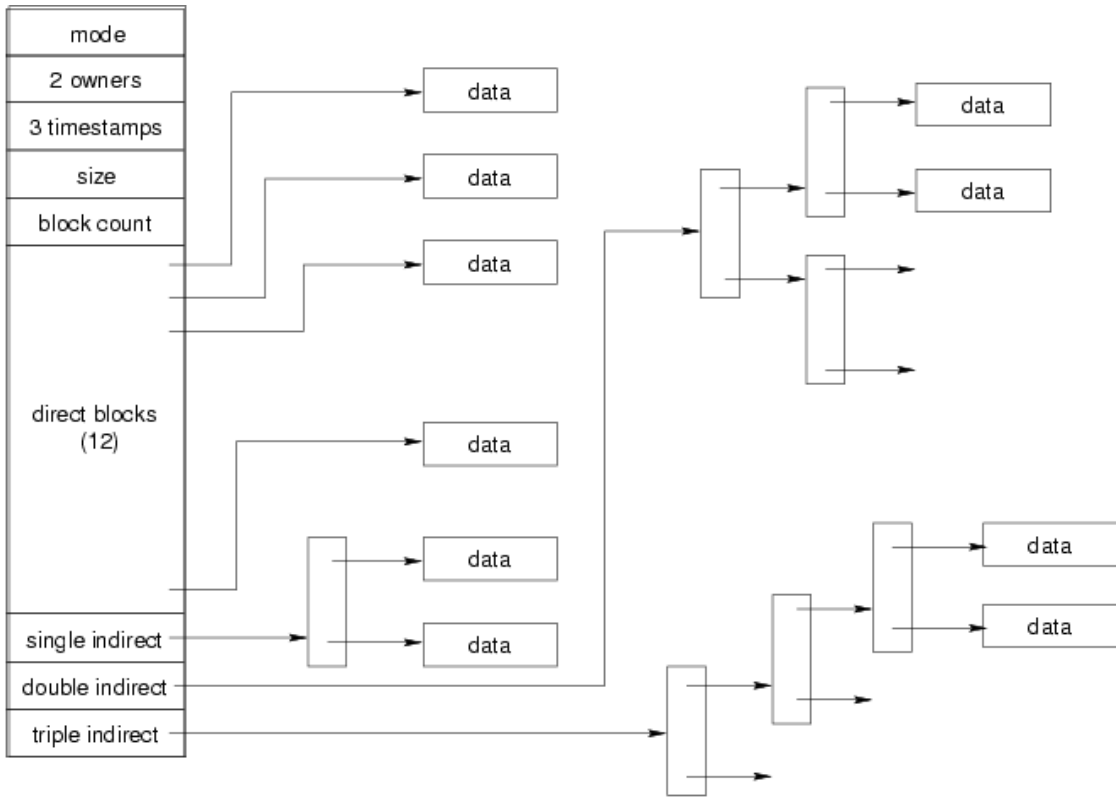
- It is a fairly complicated data structure
  - Custom data formats
  - Non-standard allocation approach
- It is mostly just a data structure
  - Lays out logical data in a memory array
  - Similarities to standard in-memory structures
    - Allocation
    - Depth of lookup trees

ext2: Prefix Trees

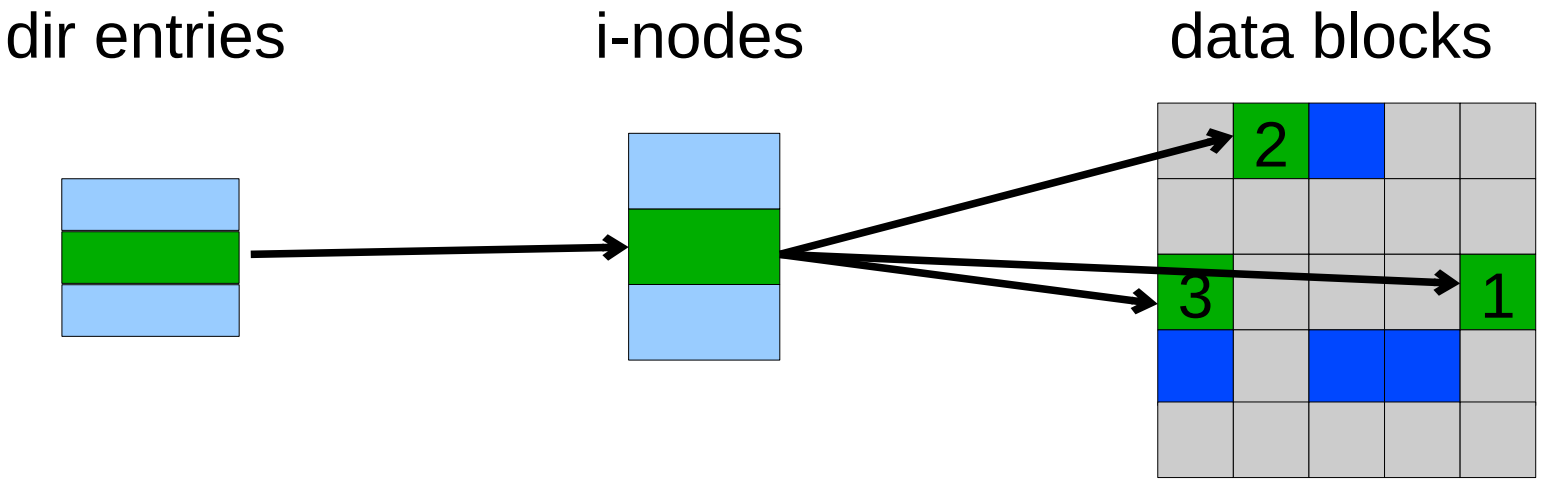
- Recall the lookup process for double/triple indirect blocks

- This is a prefix tree
  - Index each node by digits of the key

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



Brief Journaling Intro

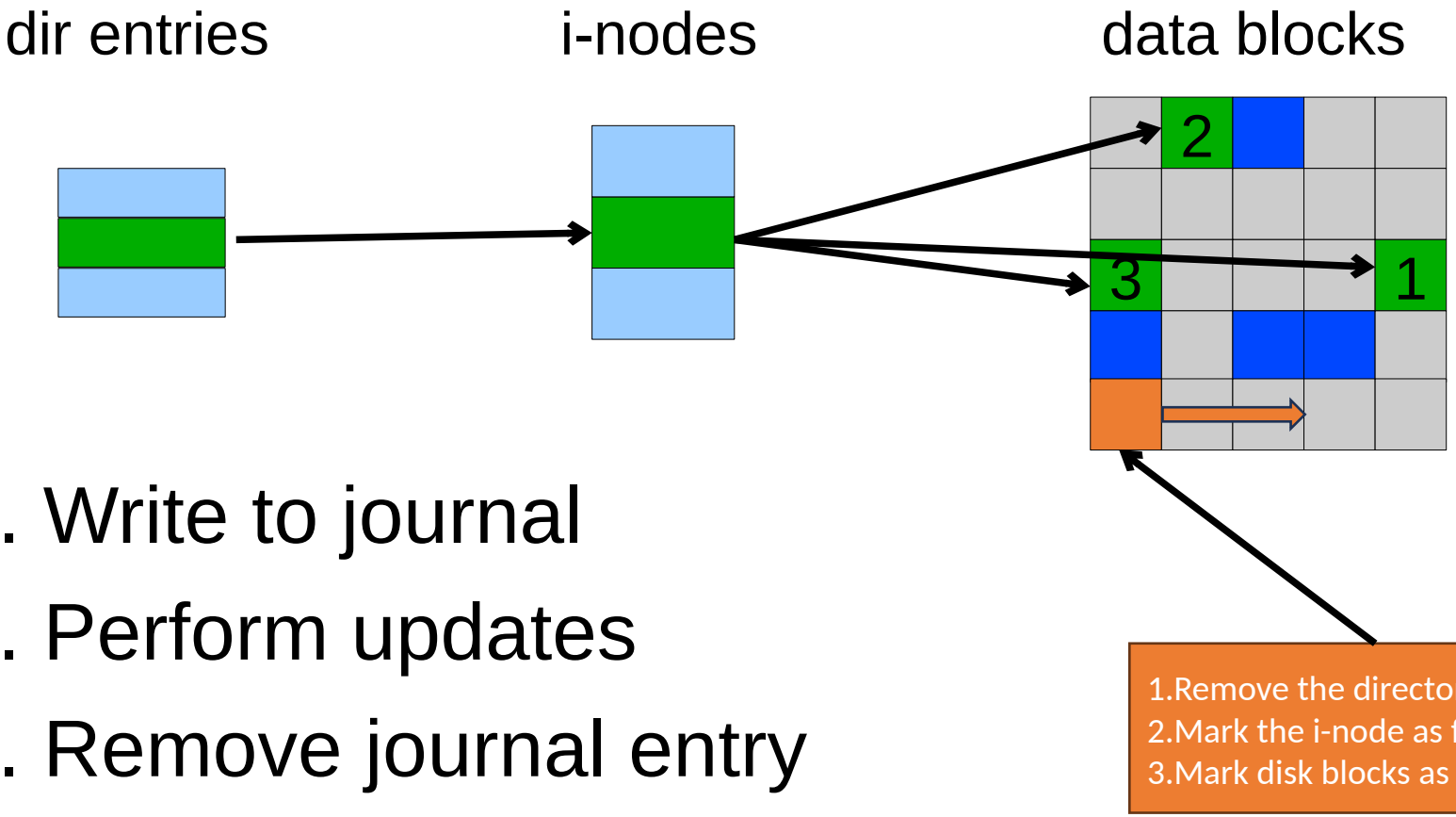


- Example: deleting a file
  1. Remove the directory entry
  2. Mark the i-node as free
  3. Mark disk blocks as free

Concurrency in File Systems

- No OS permits concurrent access to a file system
- Consistency issues are a bit like concurrency issues
  - Two instances of OS X accessing stored data
  - In this case, there is no way to go back to instance 1

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

The ext3 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

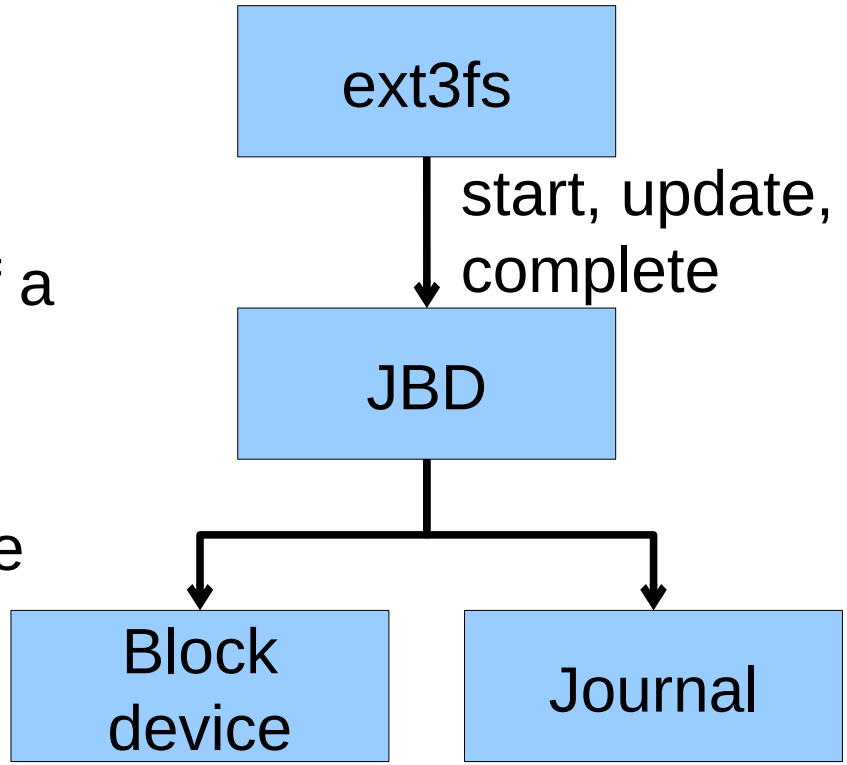
Option2: Journal disk block updates

- ✗ 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

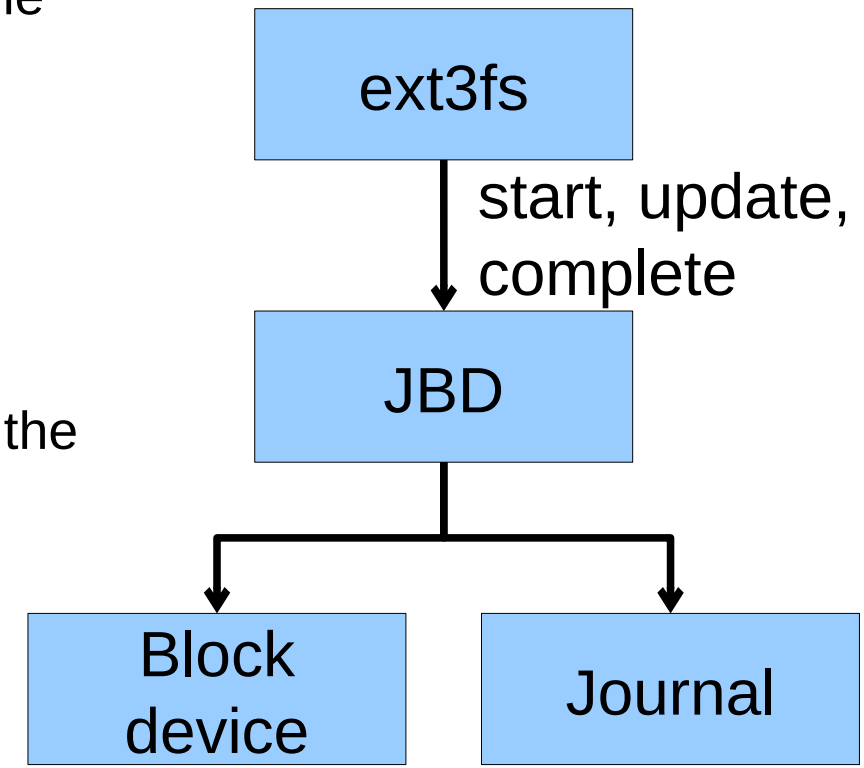
Journaling Block Device (JBD)

- 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
  - Complete a transaction
    - Completed transactions are buffered in RAM

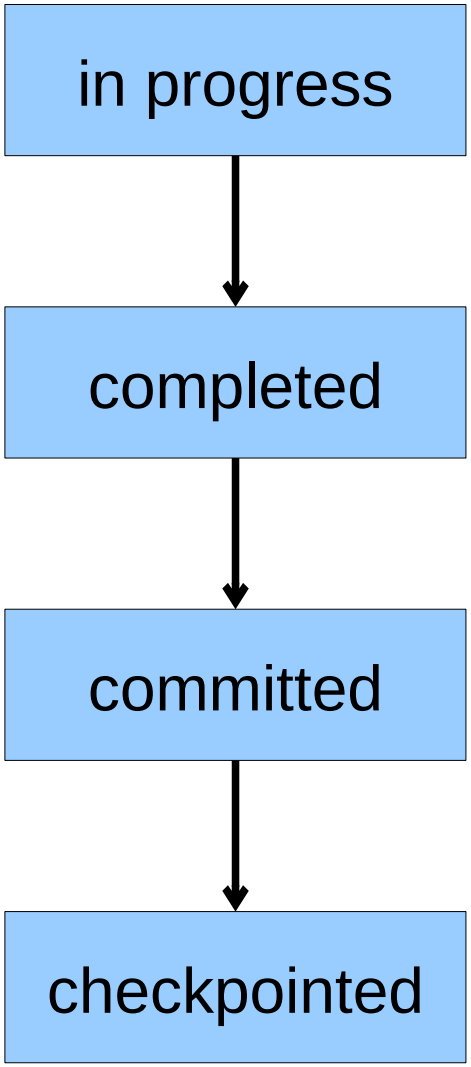


Journaling Block Device (JBD)

- JBD interface (continued)
  - Commit: write transaction data to the journal (persistent storage)
    - Multiple FS transactions are committed in one go
  - Checkpoint: flush the journal to the disk
    - Used when the journal is full or the FS is being unmounted



Transaction lifecycle



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

File Systems

- We’re 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