EECS 482
Introduction to Operating Systems

Winter 2018

Baris Kasikci

Slides by: Harsha V. Madhyastha

Multiple updates and reliability

- File system must ensure reliability/durability
  - Okay to lose data in address space
  - Data in file system must survive system crashes and power outages
- Challenge: Crashes in midst of multi-step updates

- Example: Transfer $100 from Baris's account to Tia's account
  1. Deduct $100 from Baris
  2. Add $100 to Tia
- What happens on crash between steps 1 and 2?
  - Inconsistency

Multiple updates and reliability

- Example: Move file to new directory
  1. Delete file from old directory
  2. Add file to new directory

- Example: Create new (empty) file
  1. Update directory to point to new file header
  2. Write new file header to disk

How to fix these problems?

Maintaining free disk blocks

- Option 1:
  1. Write new file header to disk
  2. Update directory to point to new file header
  3. Write the new free map
- Option 2:
  1. Write new file header to disk
  2. Write the new free map
  3. Update directory to point to new file header
- What about bank account example?

The bank transfer problem

- Transfer $100 from Baris's account to Tia's account
- Option 1:
  1. Deduct $100 from Baris
  2. Add $100 to Tia
- Option 2:
  1. Add $100 to Tia
  2. Deduct $100 from Baris

Does this sound familiar?

- Similar to preempting a thread in the middle of a critical section
  - Both allow other events to see shared variables in an inconsistent state

- Can I just acquire a lock?

Atomicity

- Threads: need atomic unit of **execution**
- Storage: need atomic unit of **storage update**
- Is this even possible?

Transactions

- Commonly used in databases: ACID property
- Main aspect for file systems: atomicity and durability (all or nothing)

```
begin
    write disk
    write disk
    write disk
end (this "commits" the transaction)
```

- Atomic operation provided by hardware: write a single sector to disk
- How to make a sequence of updates atomic?

Recap: Multiple updates and reliability

- Many file system operations need multiple disk I/Os
  - Example: Move file to new directory
    1. Delete file from old directory
    2. Add file to new directory
  - Must protect consistency from failures in between
- Careful ordering of I/Os can help in simple cases
- Transactions: Atomic sequence of updates
  - Build upon atomic hardware operation of reading from or writing to a sector

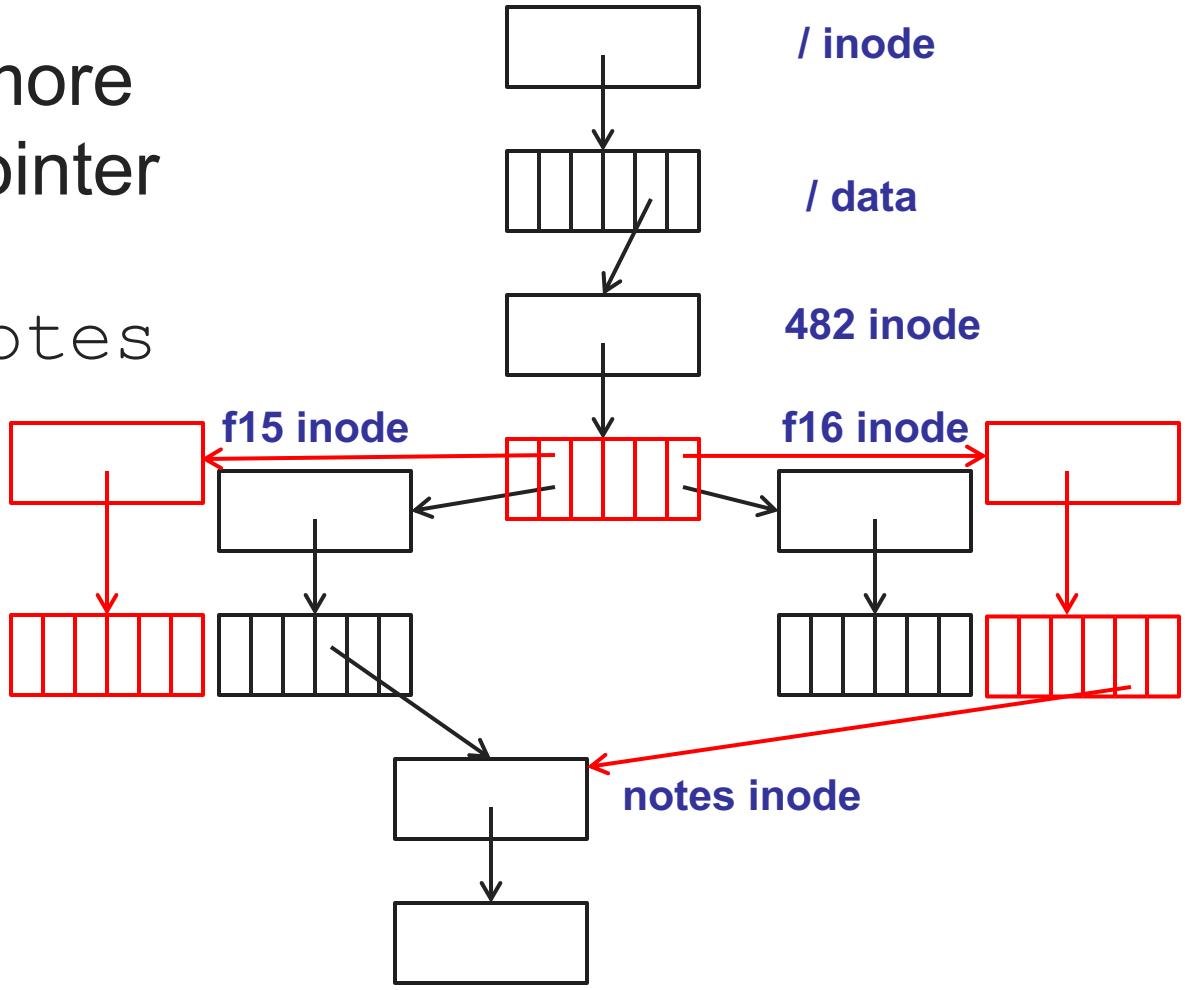
Implementing transactions with shadowing

- Keep two versions of file system (old and new)
- Store persistent pointer to the current version
- Write updates to new version
- Switch pointer to new version to commit
  - Atomically

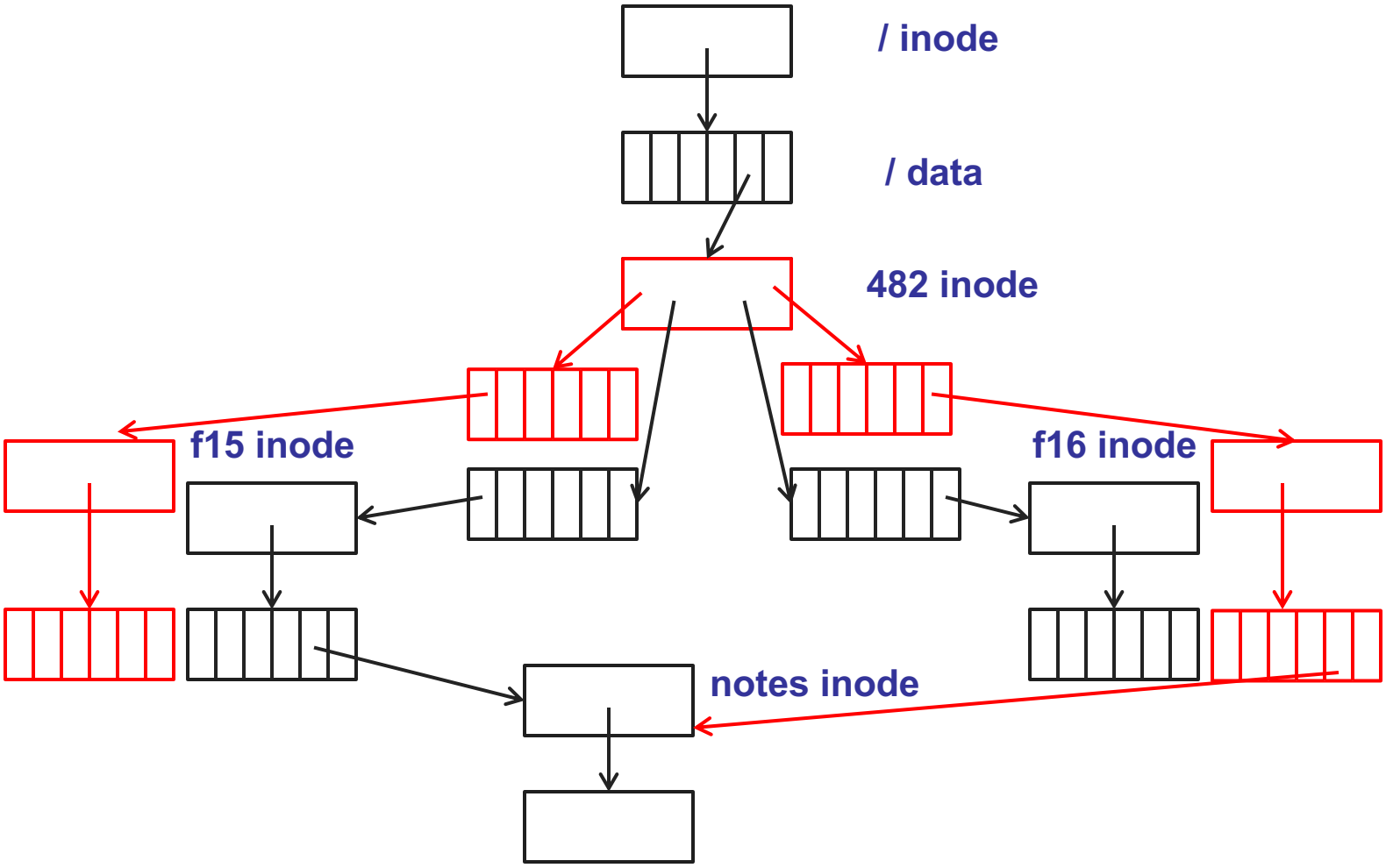
- Principle: Series of changes can be committed with a single-sector write
  - Indirection shrinks the size of the write

Optimizing shadowing

- Sector can store more than just a 1-bit pointer
- Example: move `notes` from `/482/f15/` to `/482/f16/`

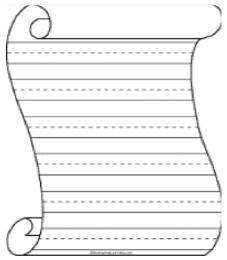


Optimizing shadowing



Implementing transactions with logging

- Write-ahead logging
  - Write updates to append-only log ***before*** applying updates to file system
  - Write commit sector to end of log to commit the set of changes



- Eventually, copy new data from log to the in-place version of the file system

- Again, update committed by single sector write

Implementing transactions with logging

- System crash before writing commit record?
- System crash after writing commit record, but before copying changes to in-place version?
- System crash while replaying log?
- Most recent file systems use logging for atomic updates to file system metadata
  - Called “journaling”
  - Why not atomic updates to data?