Transactions

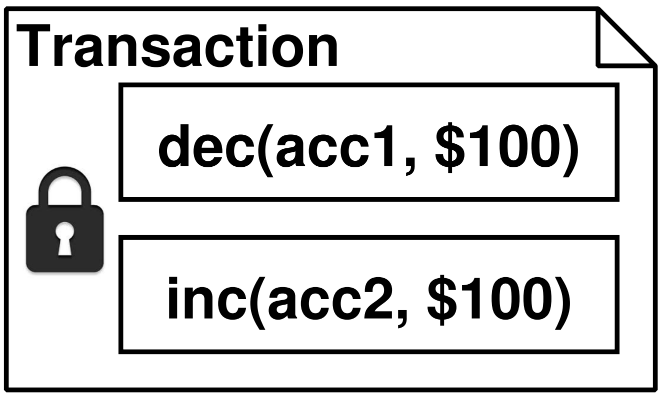
- *Group disk writes* into *transactions*
- Classic example: transfer $100 from account 1 to account 2
  - need to decrease account 1 balance by $100
  - need to increase account 2 balance by $100

dec(acc1, $100)

inc(acc2, $100)

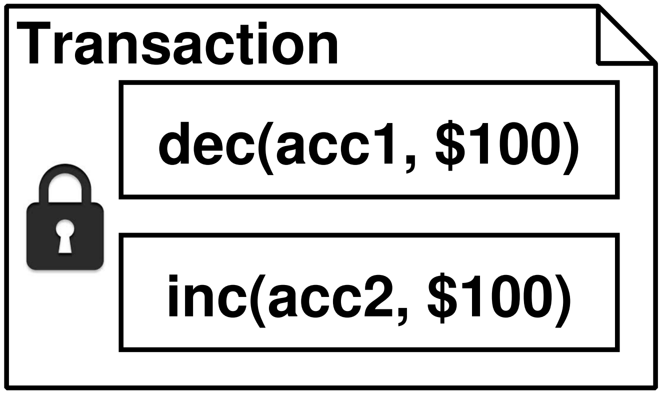
Transactions

- *Group disk writes* into *transactions*
- Classic example: transfer $100 from account 1 to account 2
  - need to decrease account 1 balance by $100
  - need to increase account 2 balance by $100
  - do this while satisfying *ACID property*



Transactions

- A *transaction* has the *"ACID" property:*
  - *atomic*
    - all or nothing
      - *commitment time* determines whether it's going to be "all" or "nothing"
  - *consistent*
    - take the file system from one consistent state to another
  - *isolated*
    - have no effect on other transactions until *committed*
  - *durable*
    - persists



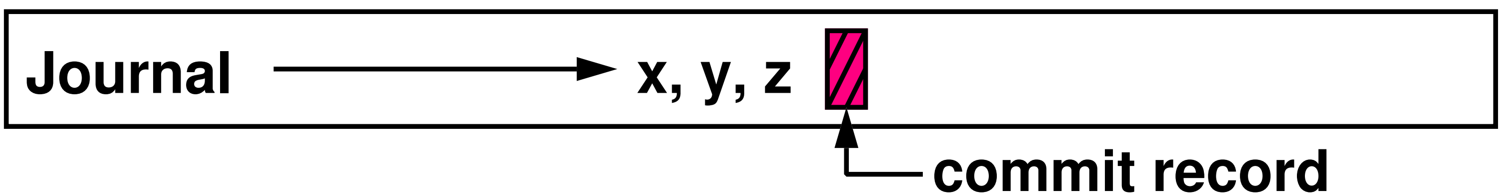
- Once you start running transactions to modify the file system, the *only* way to modify the file system is to run transactions

How?

- Journaling
  - before updating disk with steps of transaction:
    - record previous contents: *undo journaling*
      - "*before images*" of disk blocks are written into the journal
    - record new contents: *redo journaling*
      - "*after images*" of disk blocks are written into the journal
- Shadow paging
  - steps of transaction written to disk, but old values remain
  - single write switches old state to new

Journaling

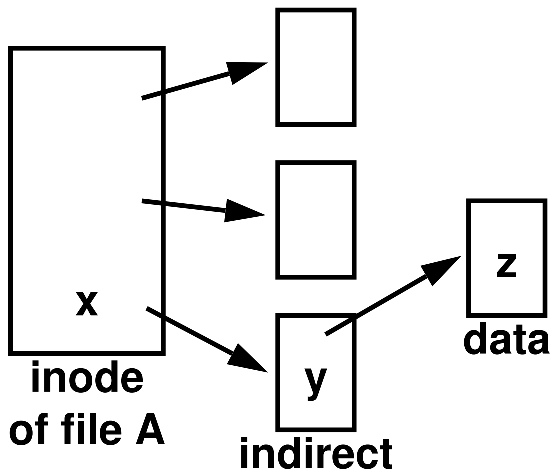
- A *journal* is a *separate part of the disk*
  - can add journaling to *any* file system
- A *journal* is *append-only*, like a log
  - for a *redo journal*, append what you are *going to write* to the main part of the disk (i.e., the file system)
  - append a *commit record*
    - a commit record is *one disk block in size*
    - the disk guarantees that a commit record is either written to the disk or not (nothing in between)

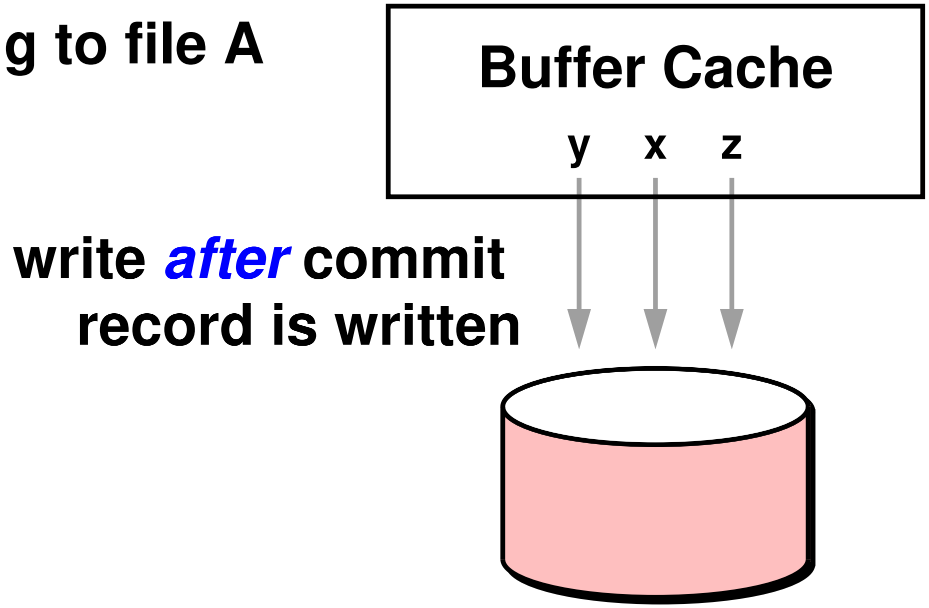


- When it's time to update the file system, *write to journal first*
  - write data to file system *asynchronously* only *after* the commit record has been written to the journal

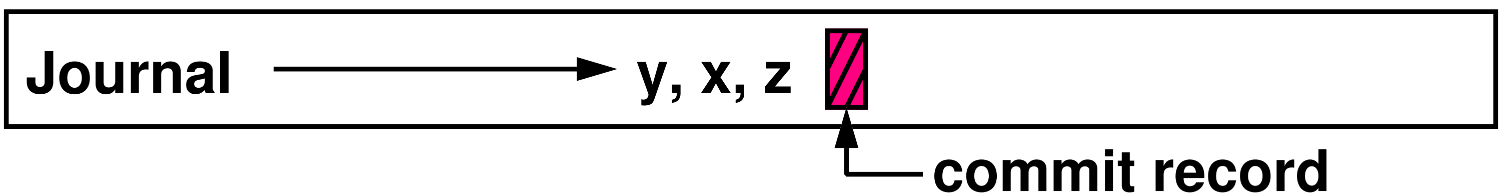
Journaling

- Let's say that you are appending to file A





- The journal is in a separate part of the disk



  - periodically:
    - find all transactions in the buffer cache
    - append to journal, one transaction at a time
    - release dirty blocks to disk update thread after commit

Recovery

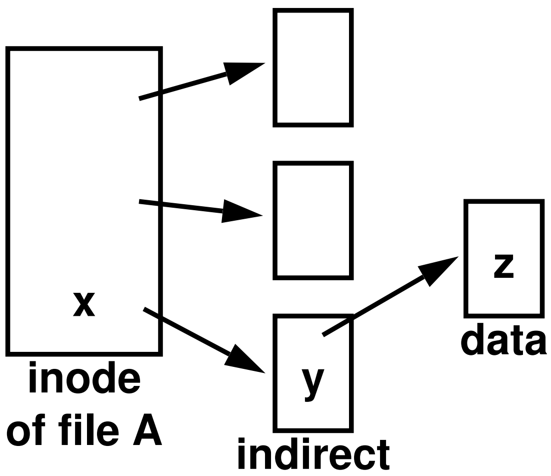
- The system can crash at any time
  - data in the file system may be inconsistent when the system reboots
  - recovery will take the file system into a *consistent state*
    - at a *transaction boundary*

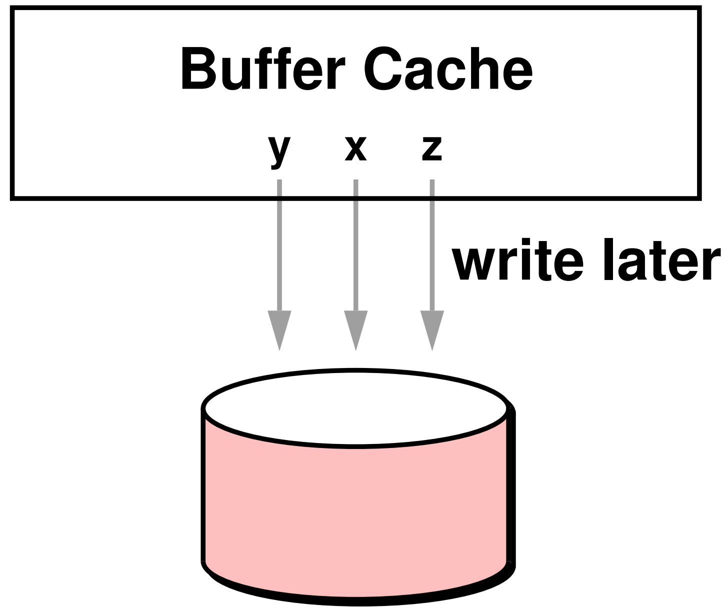
- If a *redo journal* is used, recovery involves
  - finding all committed transactions
  - redo (replay) all these transactions
    - if system crashes in the middle of a recovery, no harm is done
    - can perform recovery again and again
      - copying a disk block to the file system is *idempotent*, i.e., doing it twice has the same effect as doing it once
      - dec(acc1, $100) is *not idempotent*

- After recovery, the state of the file system is what it was at the end of the *last committed transaction*
  - by ACID property, in a consistent state

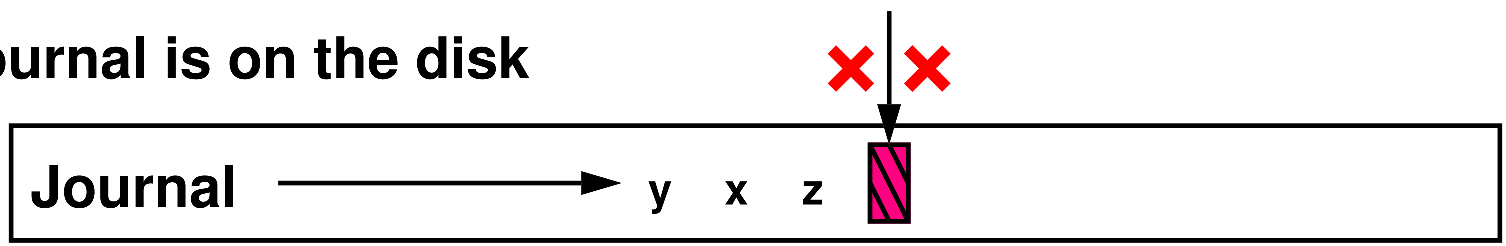
Back To The Example

- Let's say that you are appending to file A





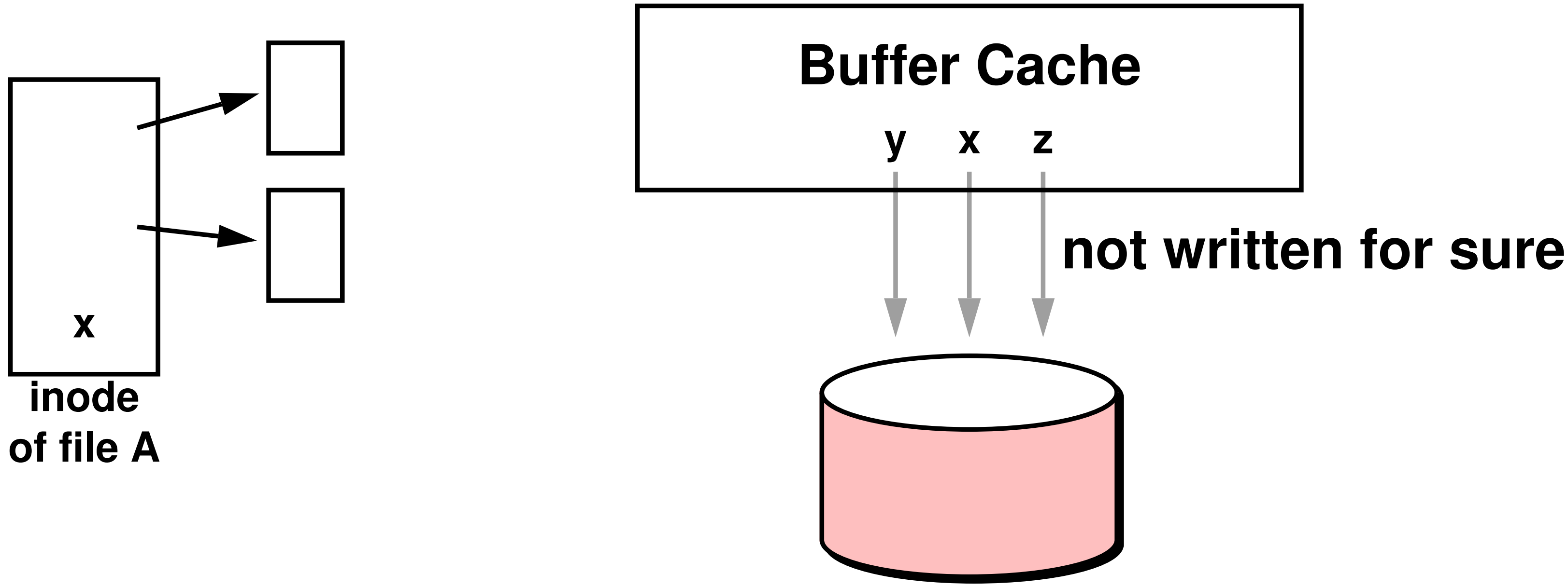
- The journal is on the disk



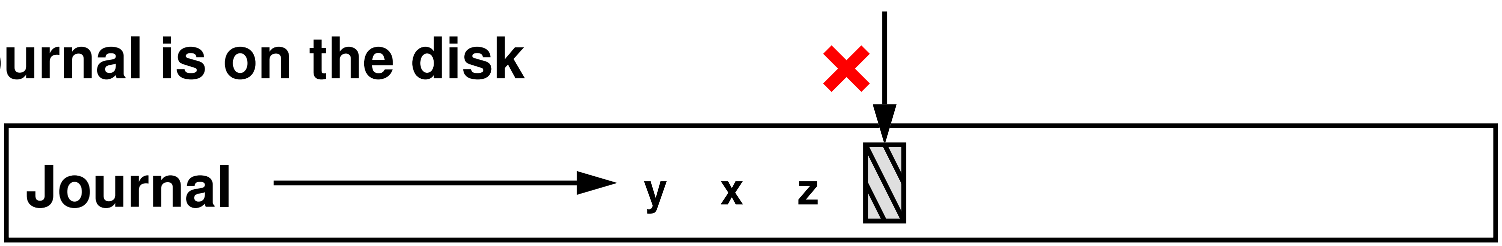
  - question is, did failure happen *before* or *after* the *commit*

Back To The Example

- Let's say that you are appending to file A



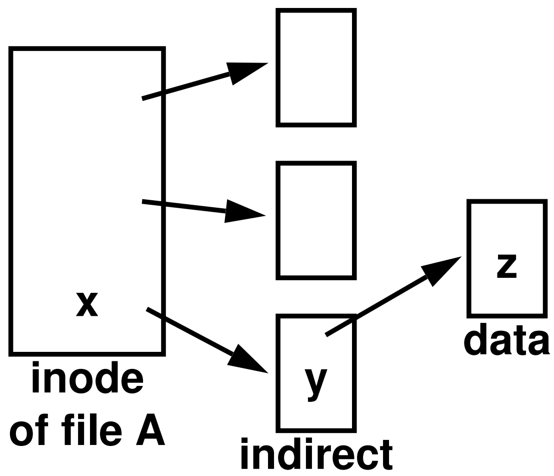
- The journal is on the disk

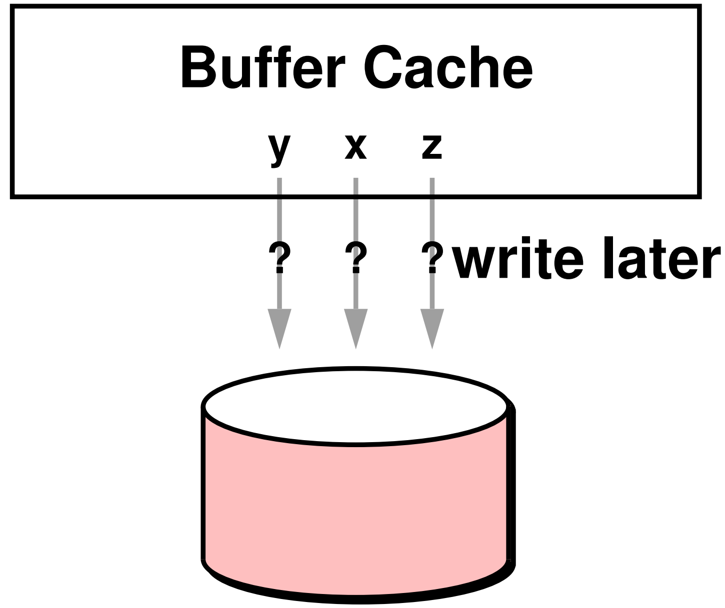


  - question is, did failure happen *before* or *after* the *commit*
  - is this bad?
    - no

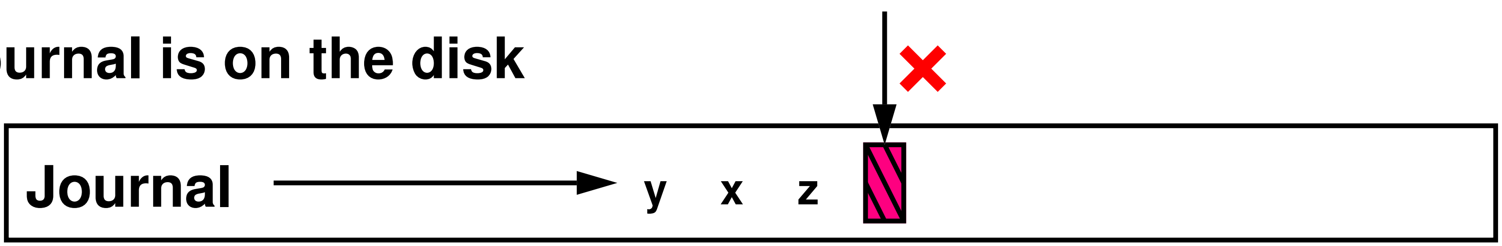
Back To The Example

- **Let's say that you are appending to file A**





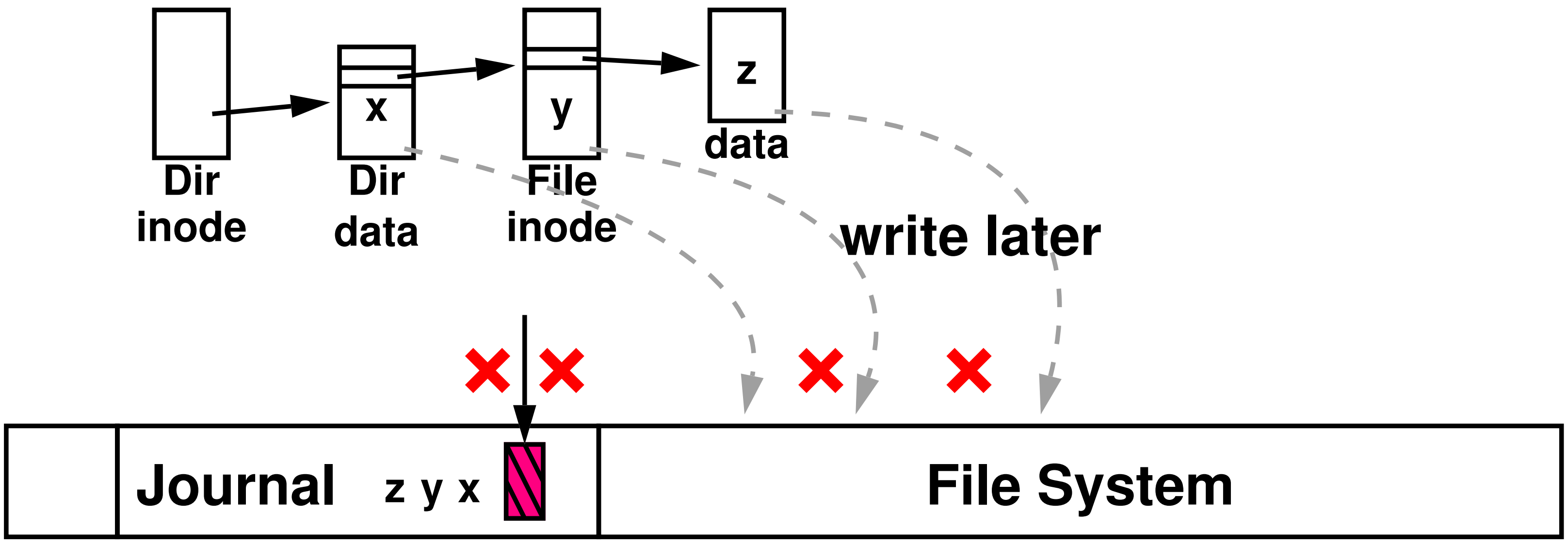
- **The journal is on the disk**



  - question is, did failure happen *before* or *after* the *commit*
  - is this bad?
    - no

Back To Example 2

- Create a new file with one data block



Journaling

- Journaling options
  - journal everything
    - everything on disk made consistent after crash
    - last few updates possibly lost
    - expensive
  - journal metadata only
    - metadata made consistent after a crash
      - *user data not*
    - last few updates possibly lost
    - relatively cheap

- In general, it's extremely costly if you want to make sure that data is *never lost*
  - if you lose power at the same time you save a document, did you click on "save" before or after you lost power?

Ext3

- A journaled file system used in Linux
  - same on-disk format as Ext2 (except for the journal)
    - (Ext2 is an FFS clone)
  - supports both full journaling and metadata only journaling

- File-oriented system calls divided into *subtransactions*
  - updates go to file system cache only
  - subtransactions grouped together

- When sufficient quantity collected or 5 seconds elapsed, *commit* processing starts
  - updates (new values) written to journal
  - once entire batch is journaled, end-of-transaction record is written
  - cached updates are then *checkpointed*, i.e., written to file system
  - *journal cleared* after checkpointing completes

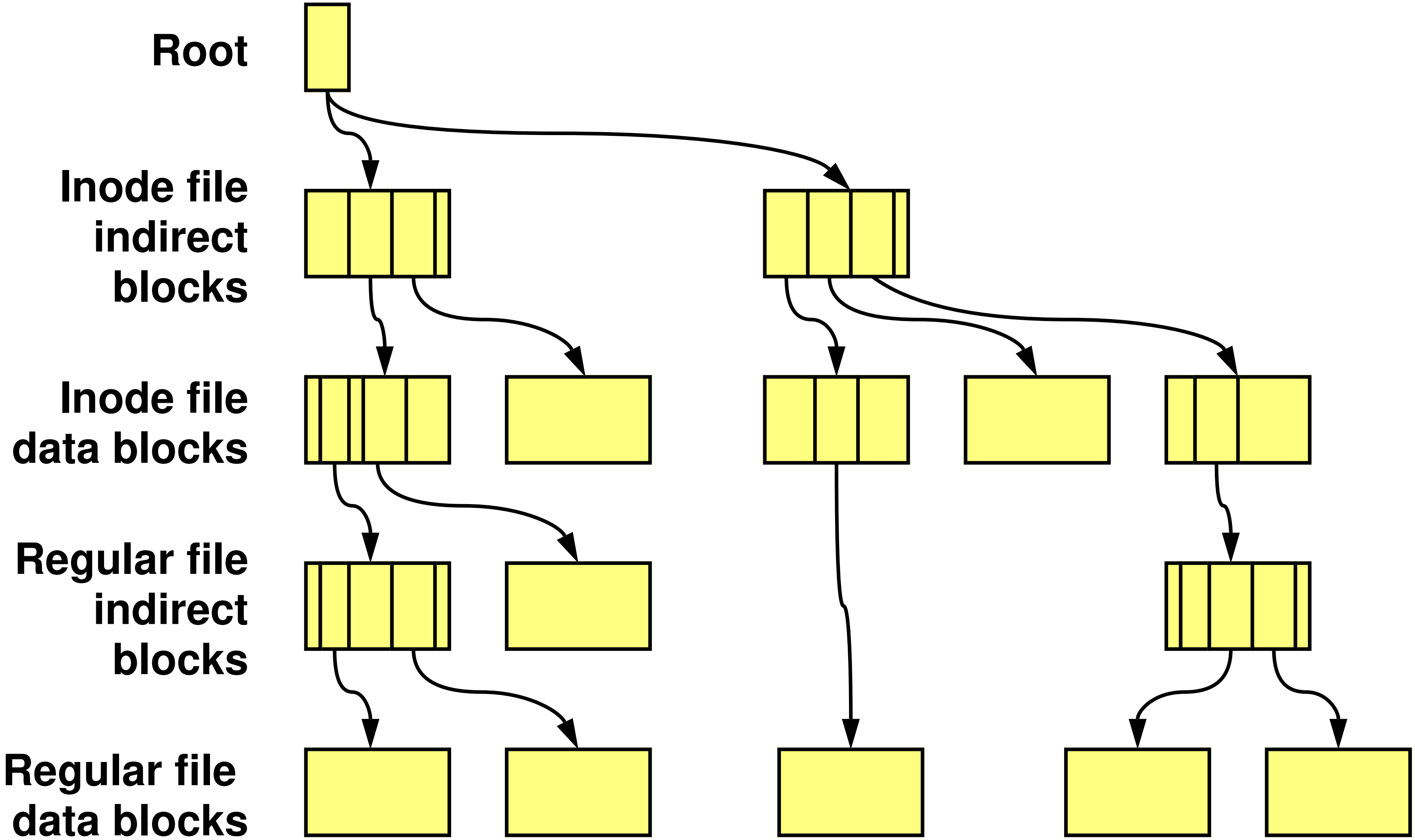
Journaling vs. Log-structured file system

- Some people confuse *journaling* with *log-structured file system*
  - log-structured file system: good *write performance*
    - coarse-grained recovery using *checkpoint file*
    - it’s a file system
  - journaling: *crash resiliency*
    - can be *added* to *any existing file system*
    - use *checkpointing* to perform write-back
      - then clear the journal

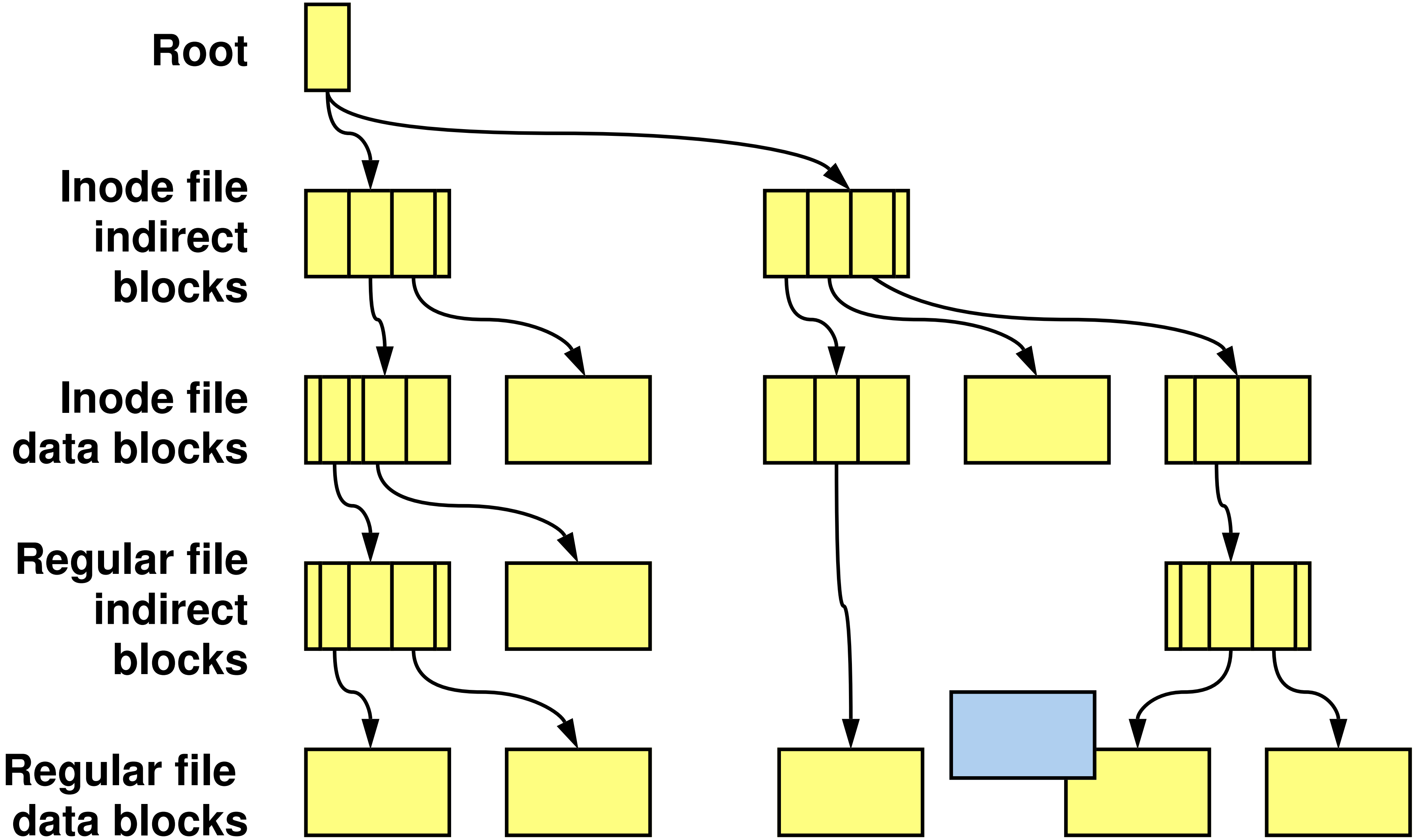
Shadow Paging

- Refreshingly simple
- Based on *copy-on-write* ideas
- Examples
  - WAFL (Network Appliance)
  - ZFS (Sun)

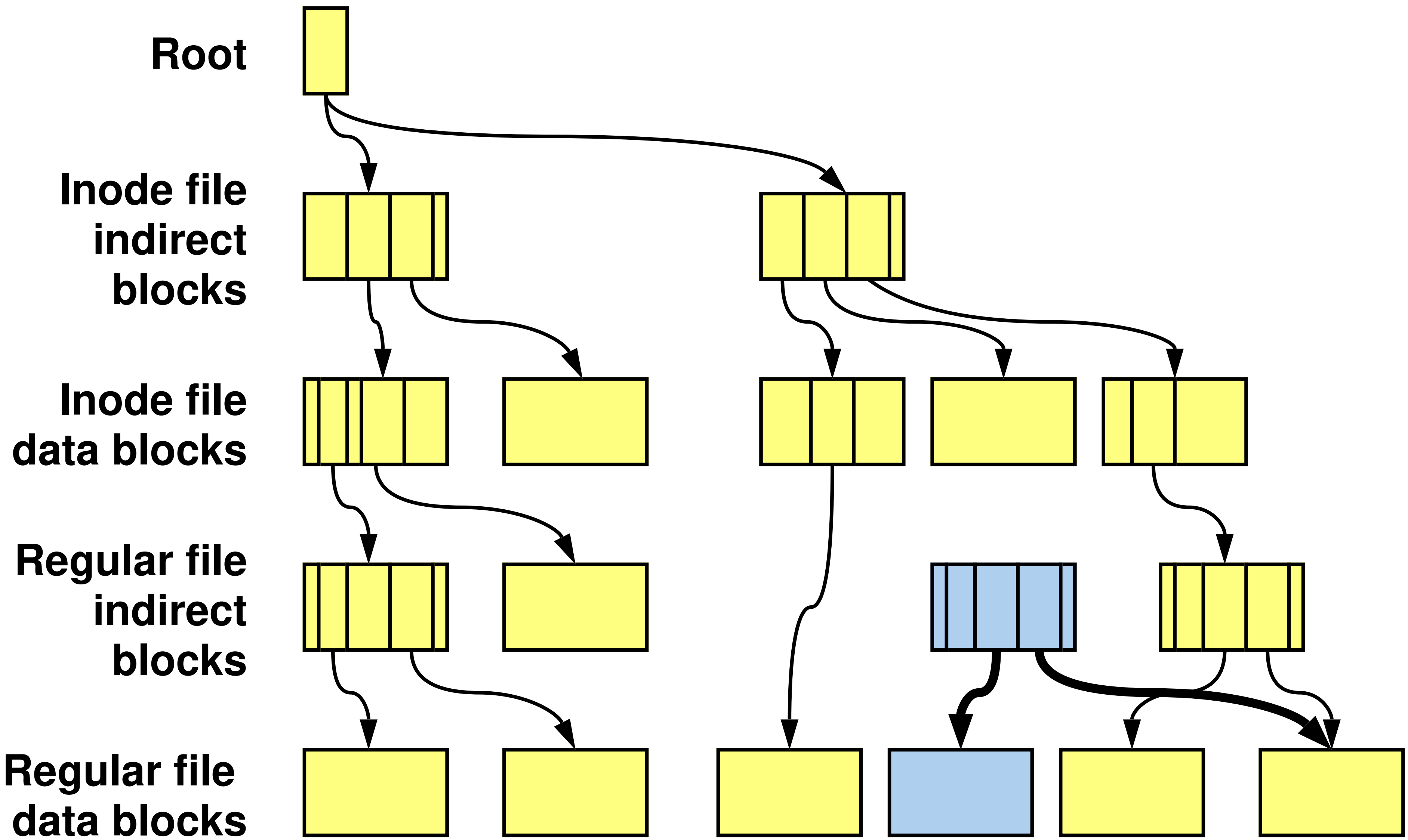
Shadow-Page Tree



Shadow-Page Tree: Modifying a Node



Shadow-Page Tree: Propagating Changes



Shadow-Page Tree: Propagating Changes

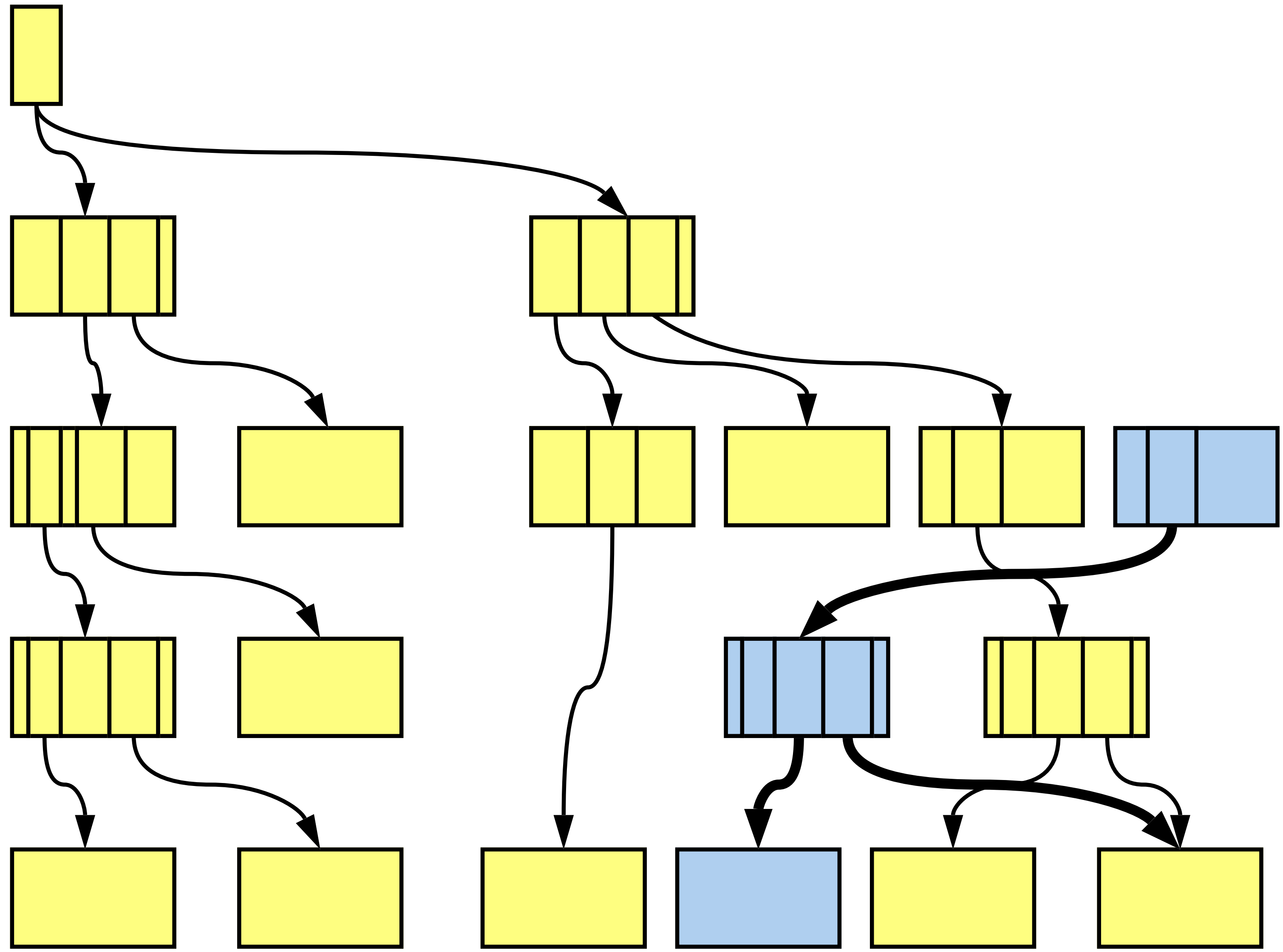
Root

Inode file indirect blocks

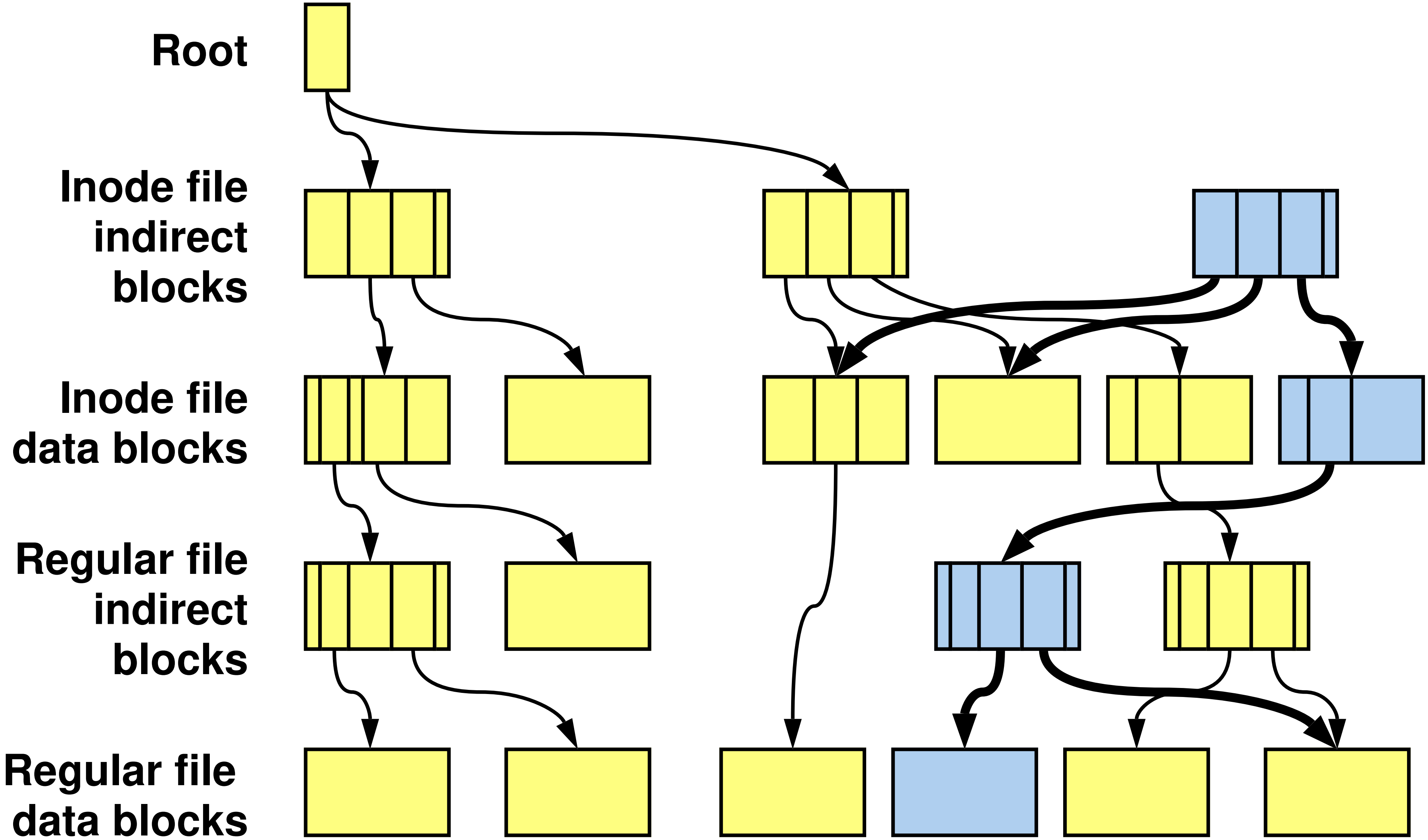
Inode file data blocks

Regular file indirect blocks

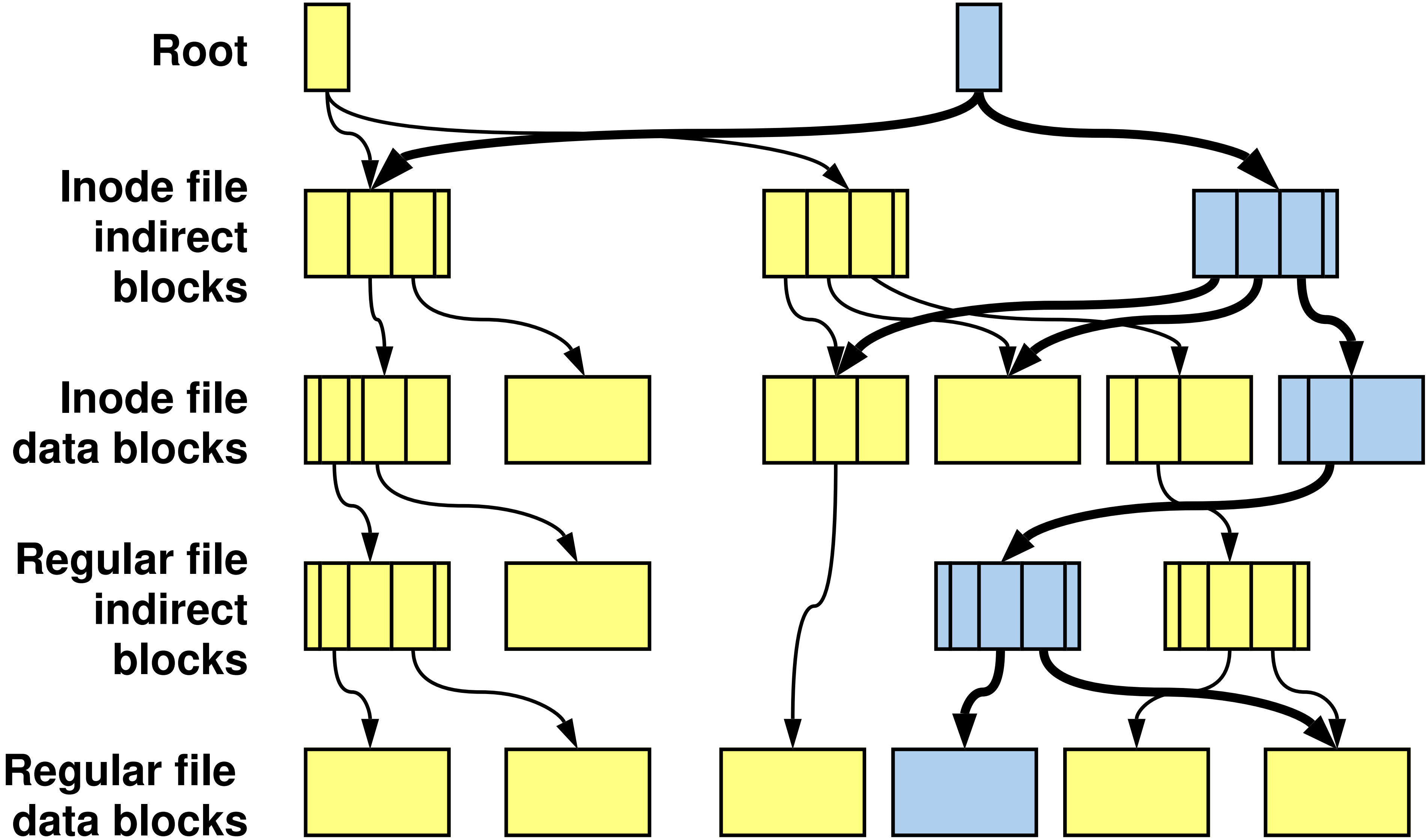
Regular file data blocks



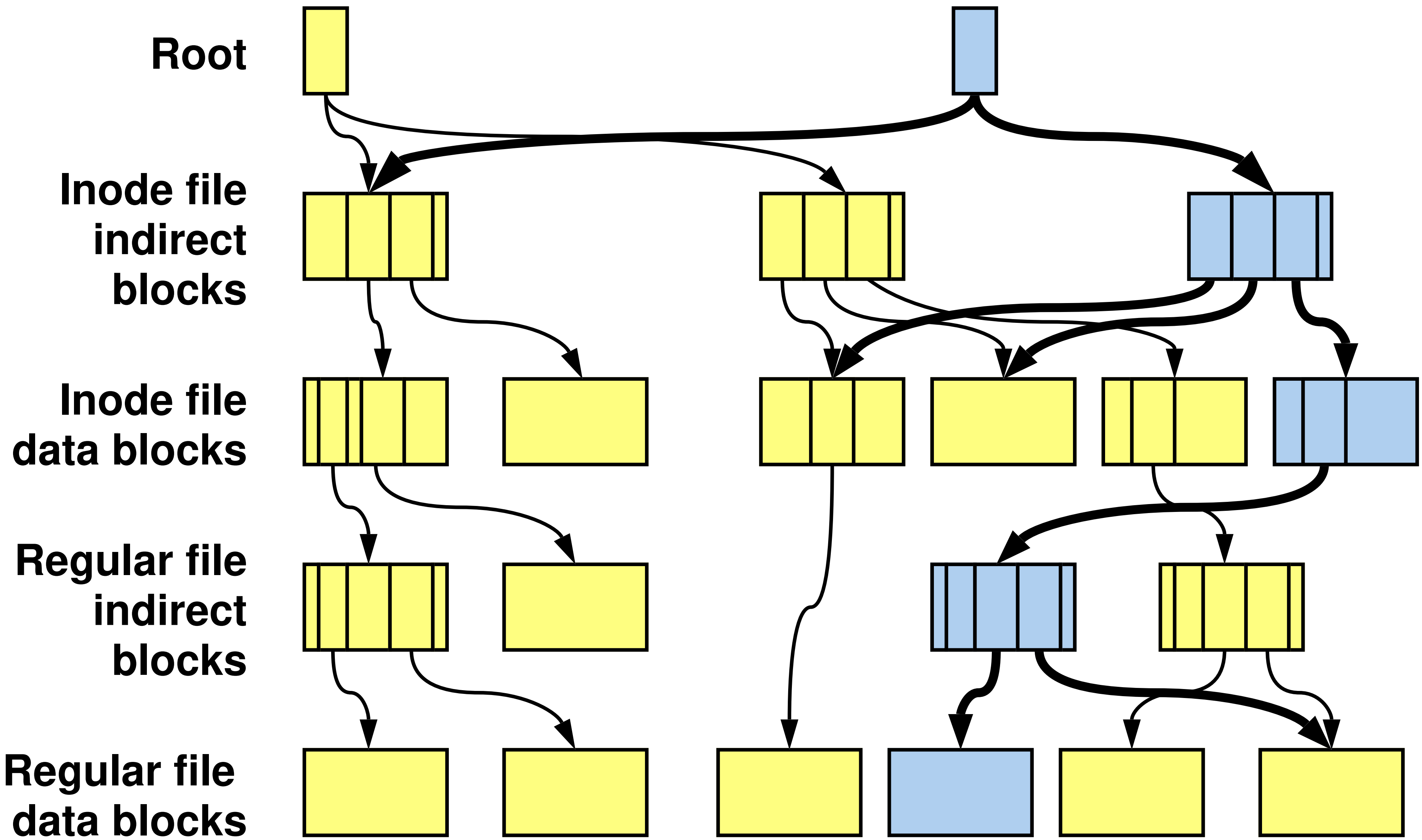
Shadow-Page Tree: Propagating Changes



Shadow-Page Tree: Propagating Changes



Shadow-Page Tree: Propagating Changes



- When root location is written to disk, it's like a *commit* record!