File Systems Part 3

Transactions

- "ACID" property:
  - atomic
    - all or nothing
  - consistent
    - take system from one consistent state to another
  - isolated
    - have no effect on other transactions until committed
  - durable
    - persists

How?

- Journaling
  - before updating disk with steps of transaction:
    - record previous contents: *undo journaling*
    - record new contents: *redo journaling*
- Shadow paging
  - steps of transaction written to disk, but old values remain
  - single write switches old state to new

Data vs. Metadata

- Metadata
  - system-maintained data pertaining to the structure of the file system
    - inodes
    - indirect, doubly indirect, triply indirect blocks
    - directories
    - free space description
    - etc.
- Data
  - data written via write system calls

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

Committing vs. Checkpointing

- Checkpointed updates
  - written to file system and are thus permanent
- Committed updates
  - not necessarily written to file system, but guaranteed to be written eventually (checkpointed), even if there is a crash
- Uncommitted updates
  - not necessarily written to file system (yet), may disappear if there is a crash

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
    - does redo journaling

Full Journaling in Ext3

- File-oriented system calls divided into subtransactions
  - – updates go to 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* — written to file system
  - – journal cleared after checkpointing completes

Quiz 1

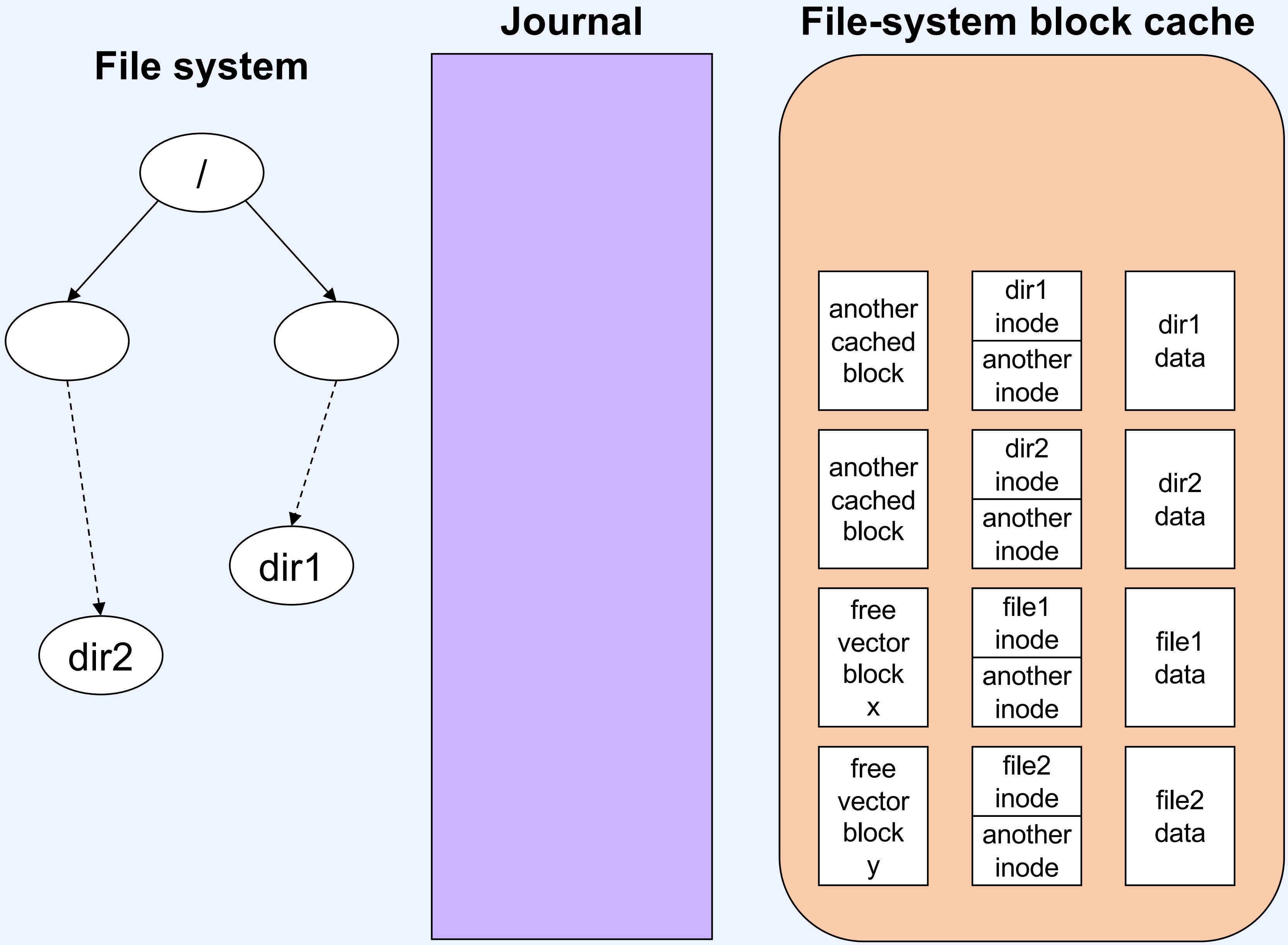
You have a Linux system with an Ext3 file system with full journaling. You run a script that deletes some files, creates some new files and writes data to them, then renames the new files. This takes two seconds. Immediately after the script finishes, there's a power failure and the system crashes. When it comes back up, after crash recovery:

a) some later parts of the script may have completed, even though earlier parts did not

b) it will appear as if the script ran to some point and then terminated (this allows for both none of it and all of it)

c) it will definitely appear as if the script ran to completion

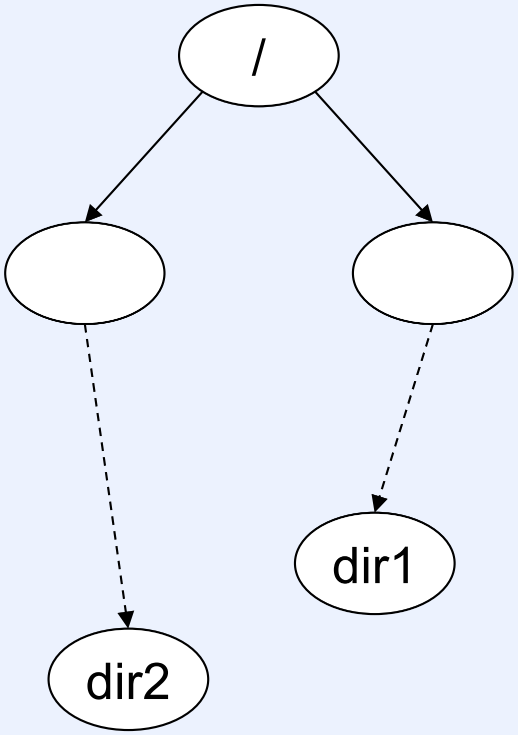
Journaling in Ext3 (part 1)



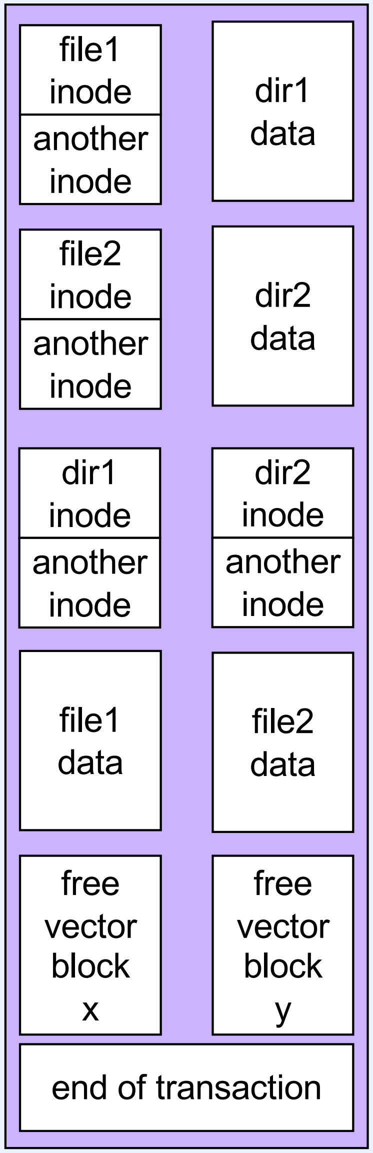
another cached block	dir1 inode another inode	dir1 data
another cached block	dir2 inode another inode	dir2 data
free vector block x	file1 inode another inode	file1 data
free vector block y	file2 inode another inode	file2 data

Journaling in Ext3 (part 2)

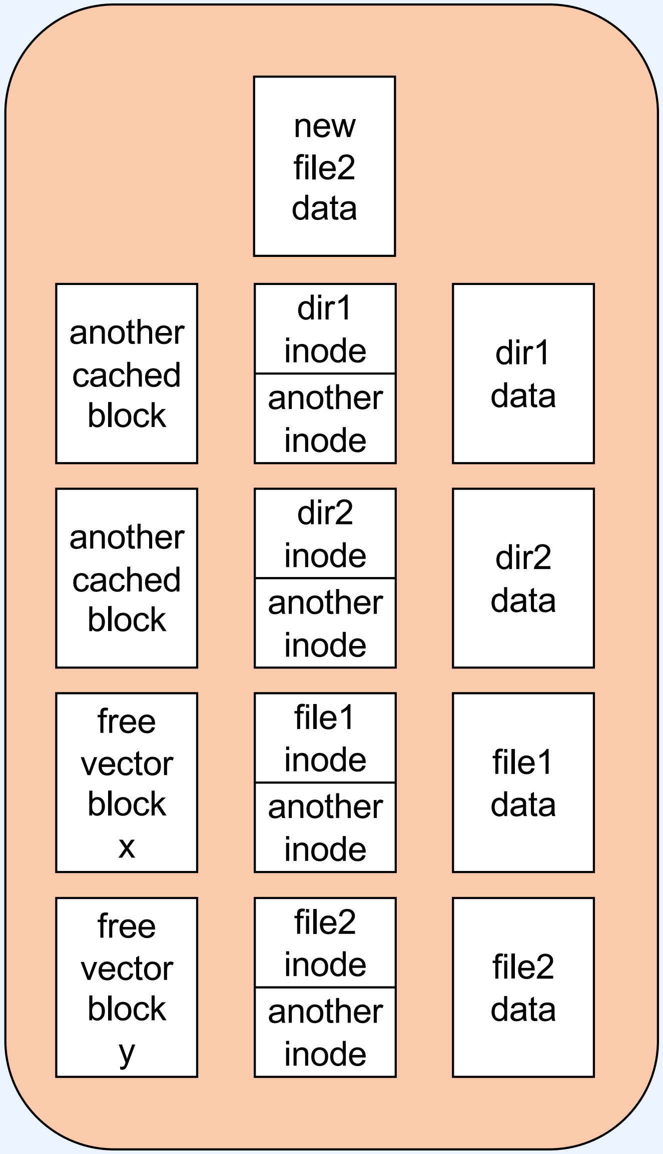
File system



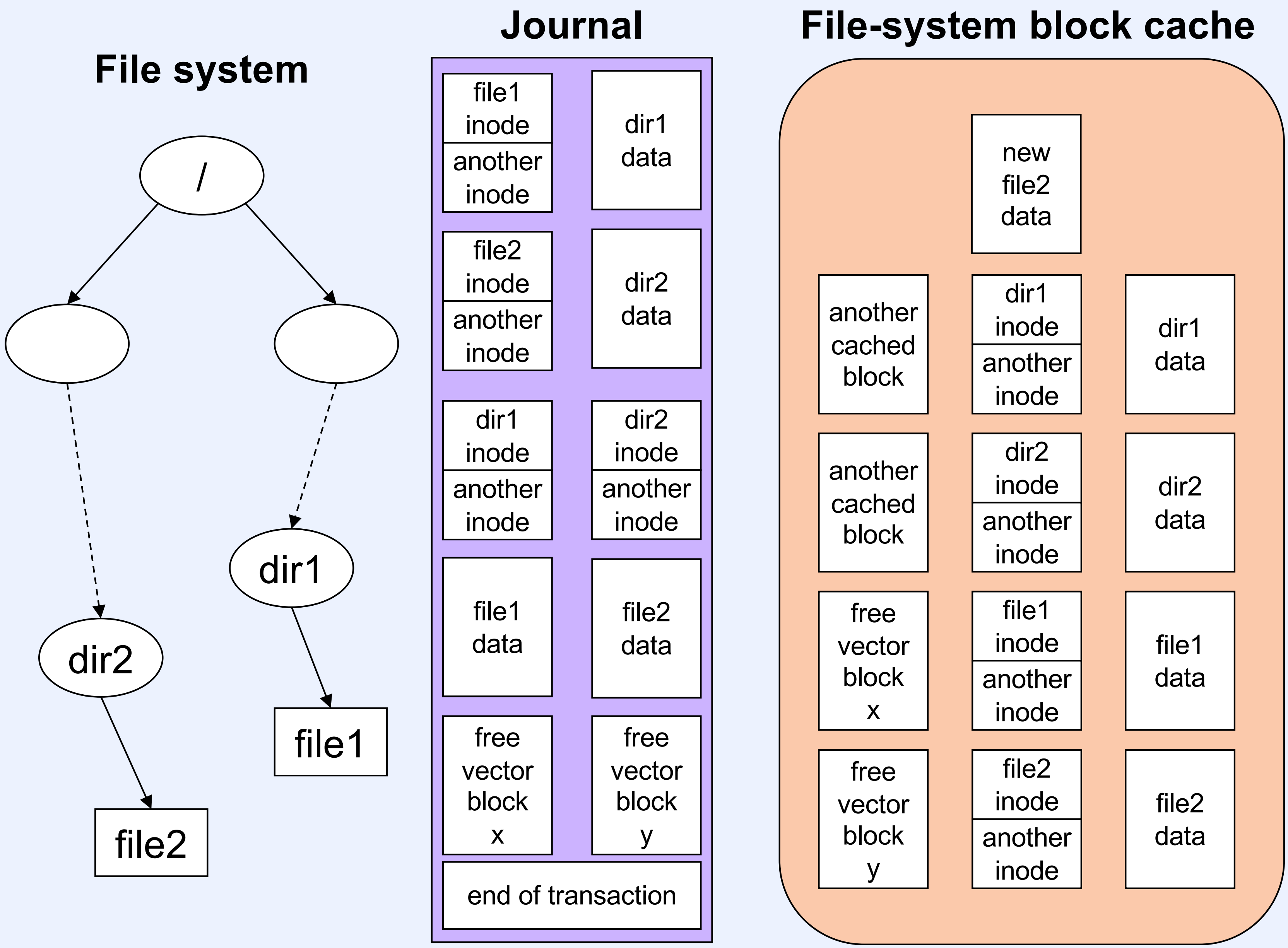
Journal



File-system block cache



Journaling in Ext3 (part 3)



File system

Journal

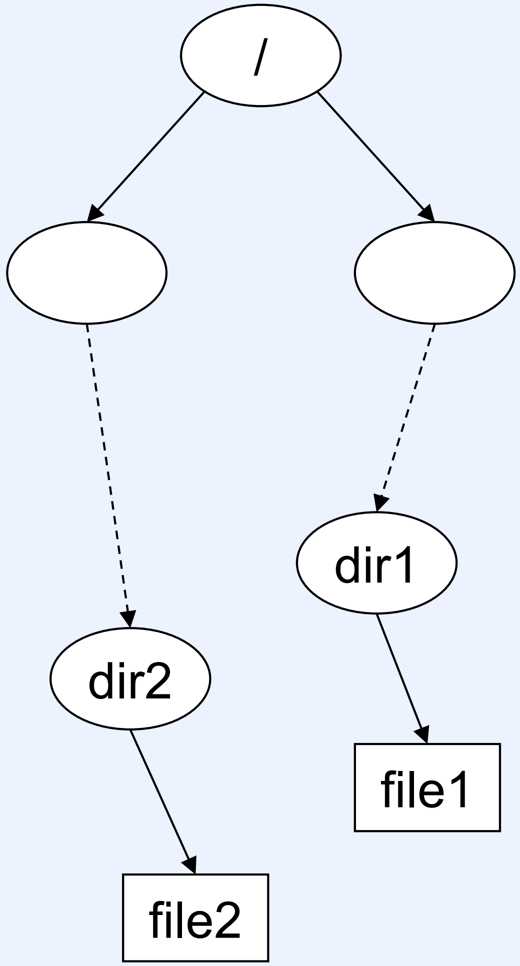
- file1 inode / another inode
- dir1 data
- file2 inode / another inode
- dir2 data
- dir1 inode / another inode
- dir2 inode / another inode
- file1 data
- file2 data
- free vector block x
- free vector block y
- end of transaction

File-system block cache

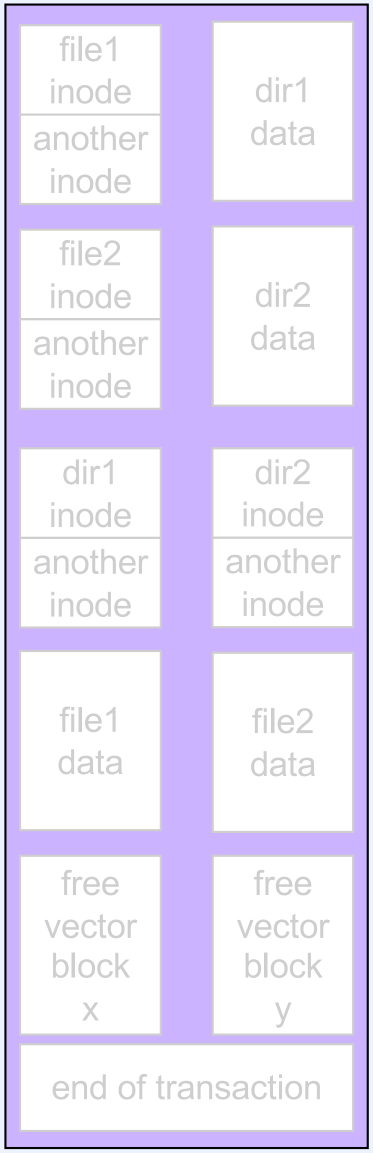
- new file2 data
- another cached block
- dir1 inode / another inode
- dir1 data
- another cached block
- dir2 inode / another inode
- dir2 data
- free vector block x
- file1 inode / another inode
- file1 data
- free vector block y
- file2 inode / another inode
- file2 data

Journaling in Ext3 (part 4)

File system



Journal



file1 inode

another inode

dir1 data

file2 inode

another inode

dir2 data

dir1 inode

another inode

dir2 inode

another inode

file1 data

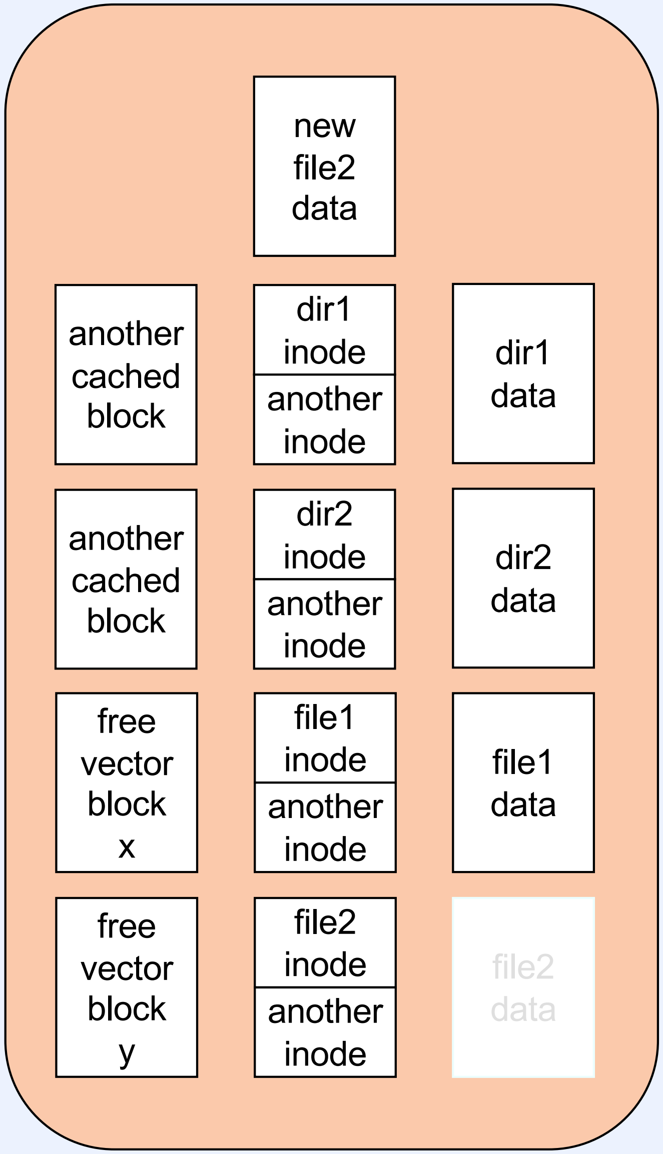
file2 data

free vector block x

free vector block y

end of transaction

File-system block cache



new file2 data

another cached block

dir1 inode

another inode

dir1 data

another cached block

dir2 inode

another inode

dir2 data

free vector block x

file1 inode

another inode

file1 data

free vector block y

file2 inode

another inode

file2 data

Quiz 2

You have a Linux system with an Ext3 file system with metadata-only journaling. You run a script that deletes some files, creates some new files and writes data to them, then renames the new files. This takes two seconds. Immediately after the script finishes, there's a power failure and the system crashes. When it comes back up, after crash recovery:

a) there may be data written to the new files, but no files were deleted

b) it will definitely appear as if the script ran to some point and then terminated (this allows for both none of it and all of it)

c) the script may appear to have completed, though there's no data written to the new files

Metadata-Only Journaling in Ext3

- File-oriented system calls divided into subtransactions
  - updates to metadata go to cache only
  - updates to data go to cache; may be written to disk at any time
- When sufficient quantity collected or 5 seconds elapsed, *commit* processing starts
  - metadata updates written to journal
  - once entire batch is journaled, end-of-transaction record is written
- Cached metadata updates are then *checkpointed* — written to file system

Metadata-Only Issues

- Scenario (one of many):
  - – you create a new file and write data to it
  - – transaction is committed
    - metadata is in journal
    - user data still in cache
  - – system crashes
  - – system reboots; journal is recovered
    - new file’s metadata are in file system
    - user data are not
    - metadata refer to disk blocks containing other users’ data

Coping

- Zero all disk blocks as they are freed
  - done in “secure” operating systems
  - expensive
- Ext3 approach
  - write newly allocated data blocks to file system before committing metadata to journal
  - fixed?

Yes, but …

- **Mary deletes file A**
  - **A's data block *x* added to free vector**
- **Ted creates file B**
- **Ted writes to file B**
  - **block *x* allocated from free vector**
  - **new data goes into *x***
  - **system writes newly allocated *x* to file system in preparation for committing metadata, but …**
- **System crashes**
  - **metadata did not get journaled**
    - **A still exists; B does not**
    - **B's data is in A**

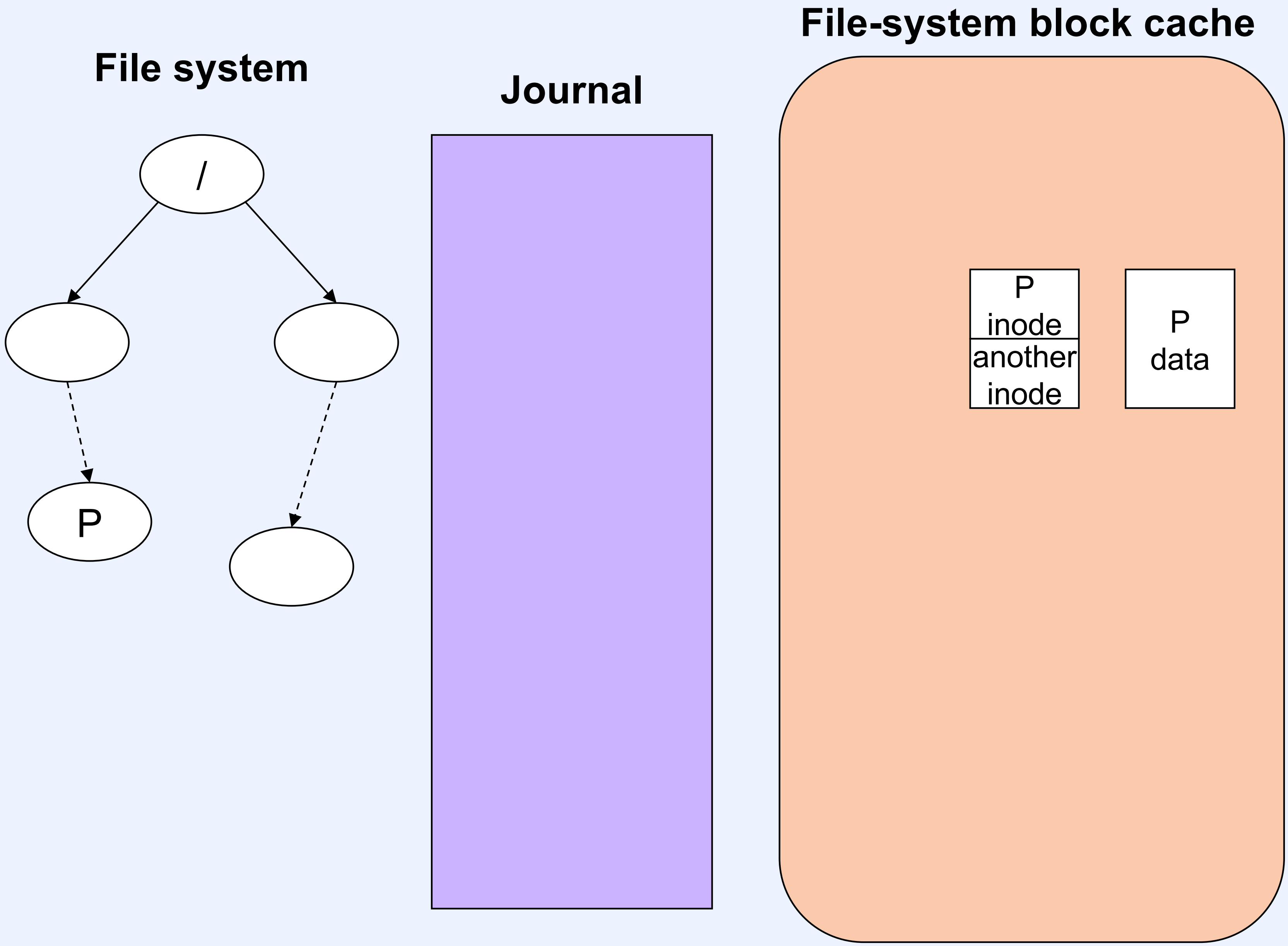
Fixing the Fix

- Don’t reuse a block until transaction freeing it has been committed
  - keep track of most recently committed free vector
  - allocate from it

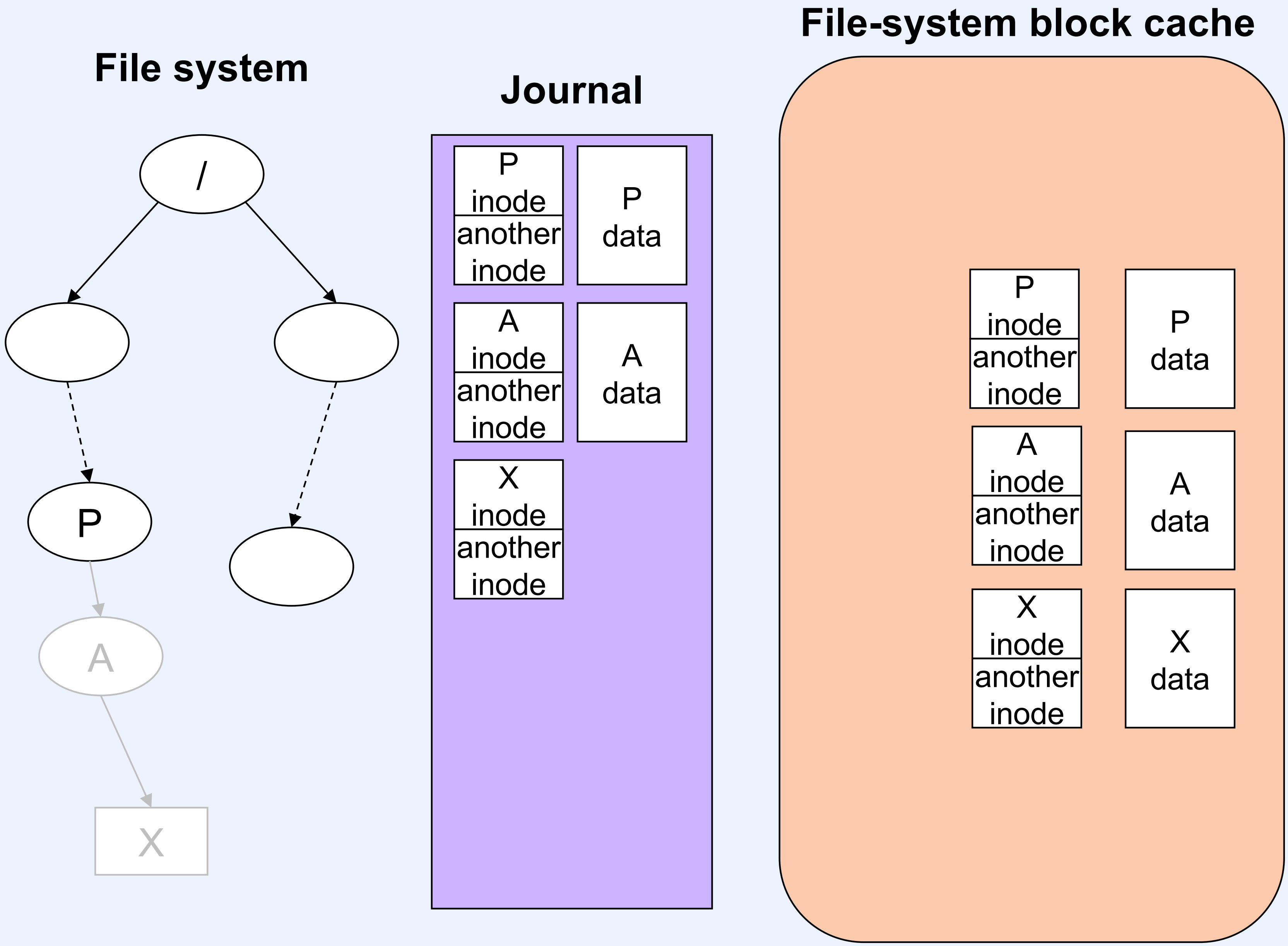
Fixed Now?

- No …

Yet Another Problem (part 1)



Yet Another Problem (part 2)

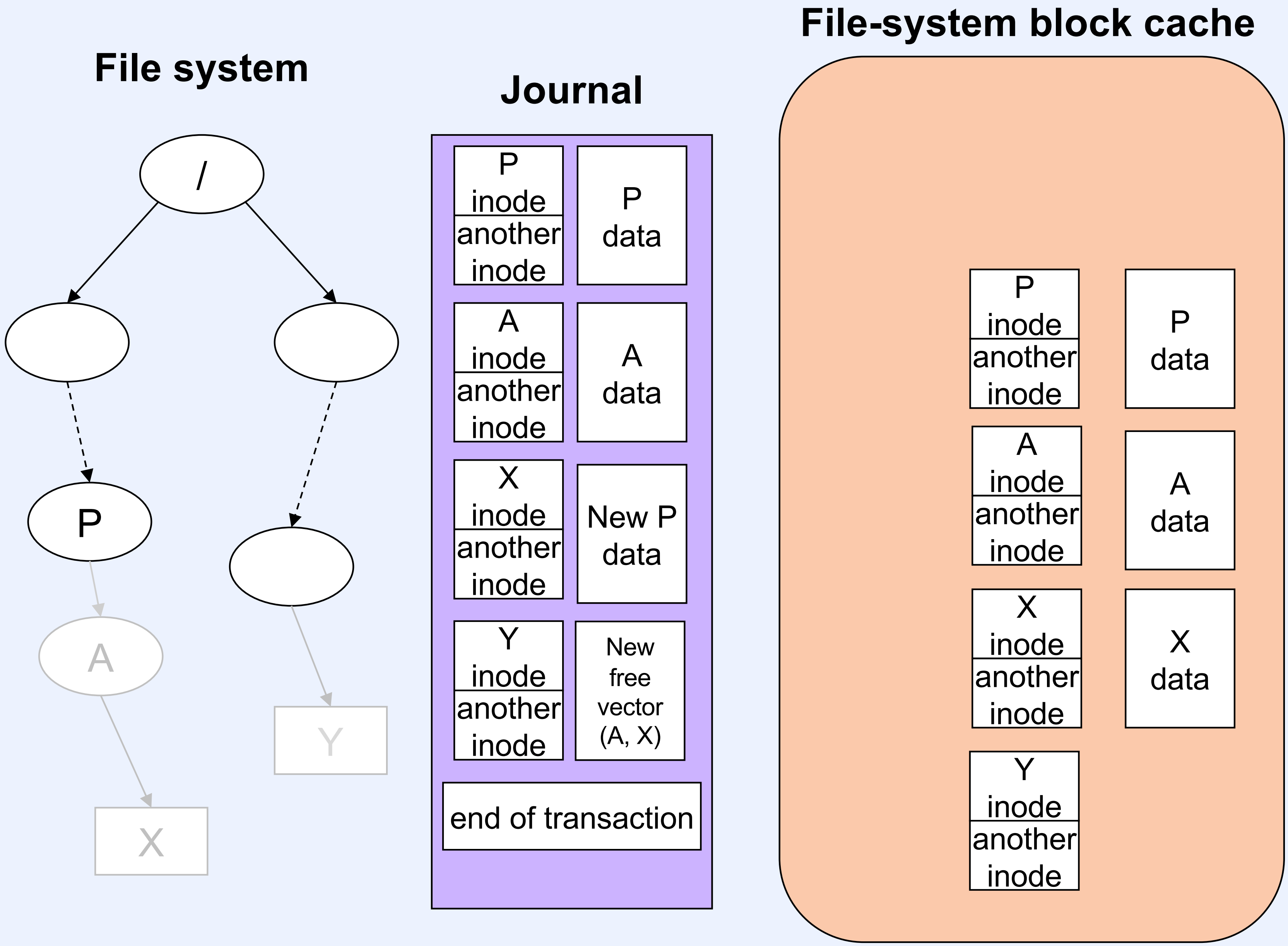


File system

Journal

File-system block cache

Yet Another Problem (part 3)



File system

/

P

A

Y

X

Journal

P inode
another inode

P data

A inode
another inode

A data

X inode
another inode

New P data

Y inode
another inode

New free vector (A, X)

end of transaction

File-system block cache

P inode
another inode

P data

A inode
another inode

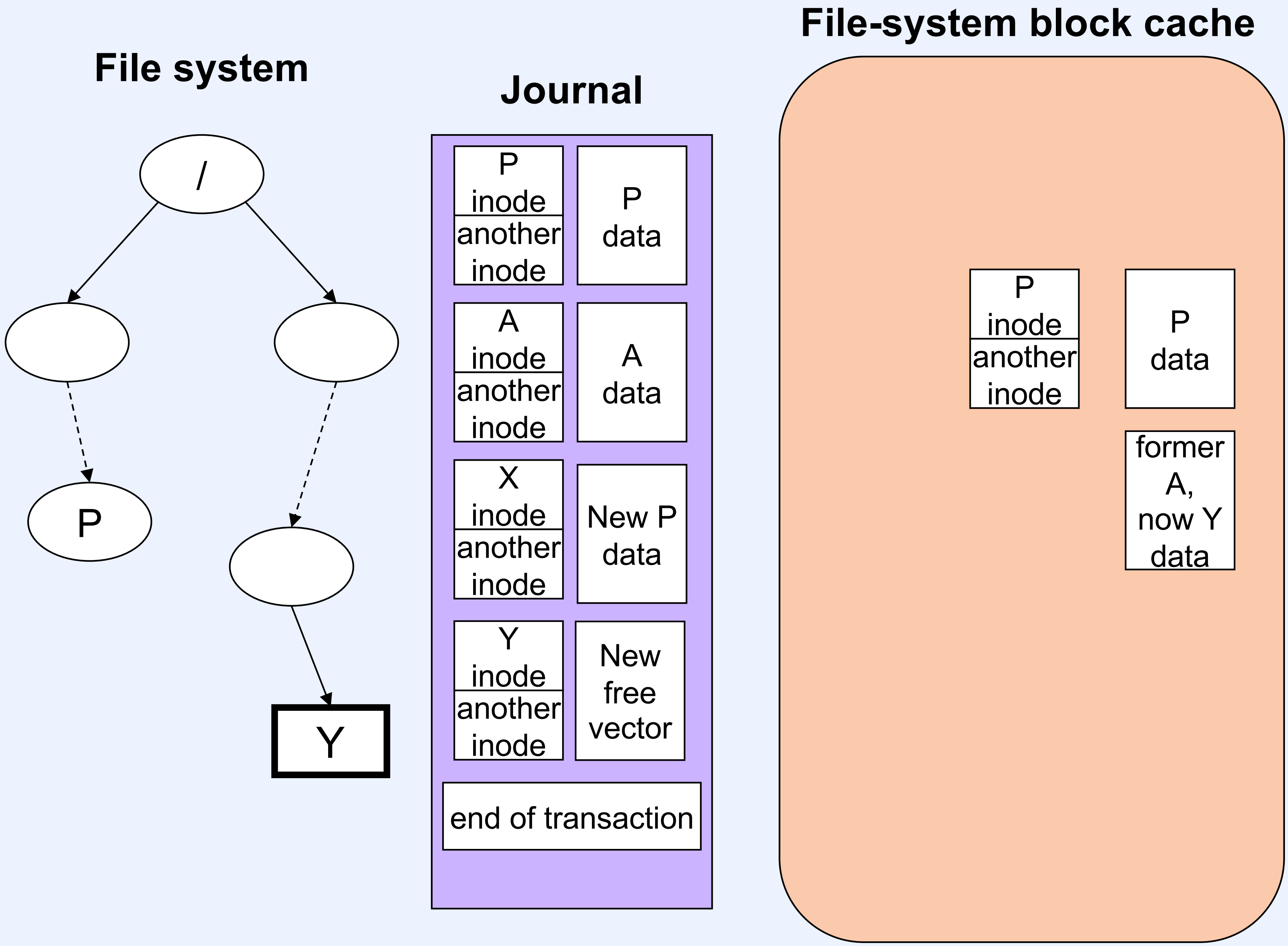
A data

X inode
another inode

X data

Y inode
another inode

Yet Another Problem (part 4)



File system

/

P

Y

Journal

P inode
another inode | P data

A inode
another inode | A data

X inode
another inode | New P data

Y inode
another inode | New free vector

end of transaction

File-system block cache

P inode
another inode | P data

former A, now Y data

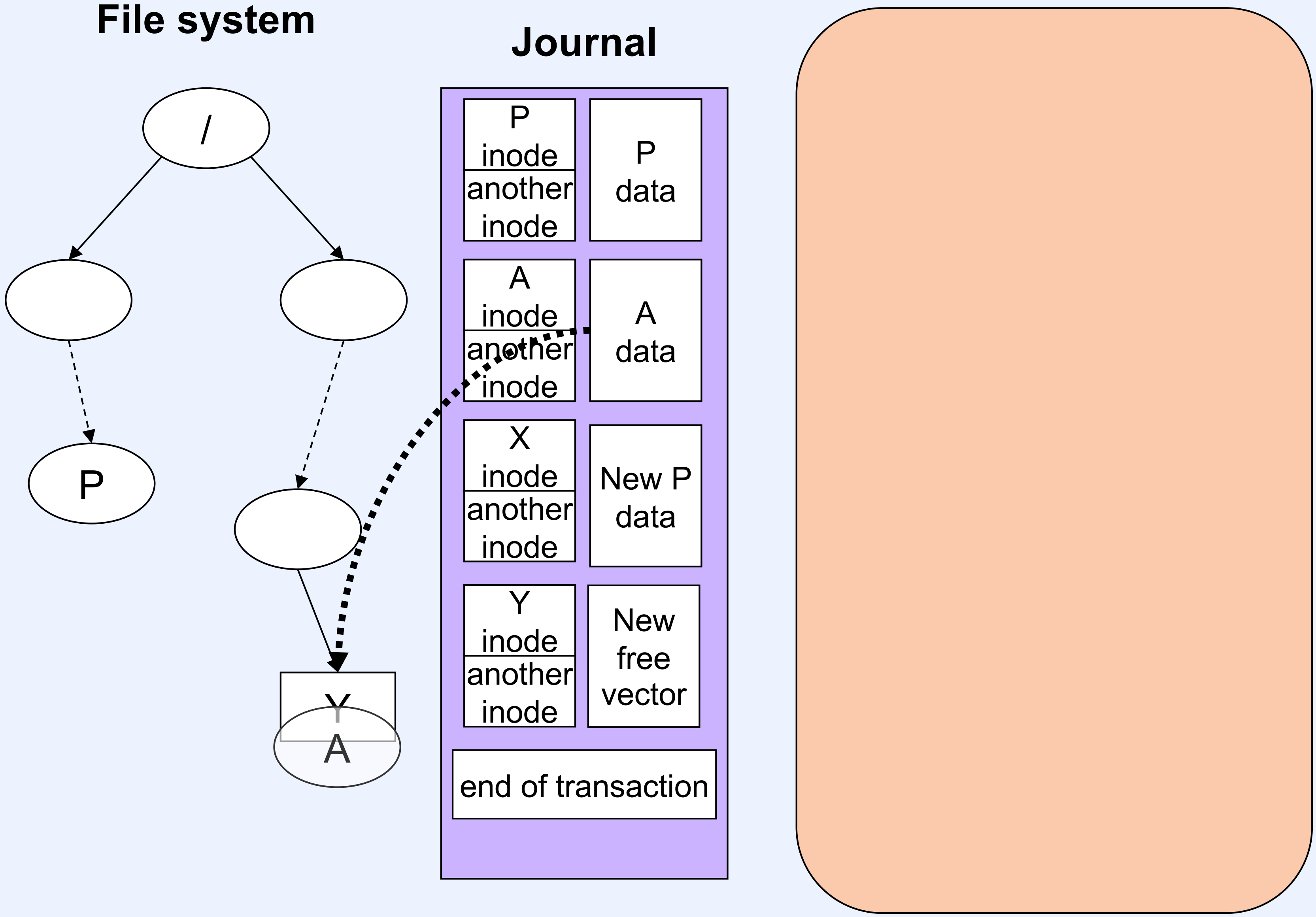
CRASH!!!

Yet Another Problem (part 5)

File-system block cache

File system

Journal



The Fix

- The problem occurs because metadata is modified, then deleted.
- Don’t blindly do both operations as part of crash recovery
  - no need to modify the metadata!
  - Ext3 puts a “revoke” record in the journal, which means “never mind …”

Fixed Now?

- Yes!
  - (or, at least, it seems to work …)

Ext4

- Latest Linux file system
  - used at Brown CS as local FS on Linux systems
- Retains much of Ext3
  - journaling
    - meta-data only used at Brown CS
  - inodes
- Adds extents
  - four extents in inode
  - if more needed, B-tree is used

Undo Journaling

- Old data written to journal (from cache)
- New data written to cache; written to file system when convenient
- Committed after checkpointing
  - new data is on disk
  - (undo) journal entries removed
- Crash recovery
  - if transaction not committed, all changes to file system are undone by playing back the journal

Undo vs. Redo

- Redo
  - data written to cache
  - data copied to journal
    - committed when entire transaction is in journal
  - data stays in cache until it is checkpointed
- Undo
  - old data written to journal (from cache)
  - new data written to cache
  - new data written to disk
    - committed when all data on disk
  - data may be removed from cache prior to committing

Quiz 3

If undo journaling is being used, when can a transaction be committed?

a) when the new contents of all blocks involved in the transaction have been written to the cache

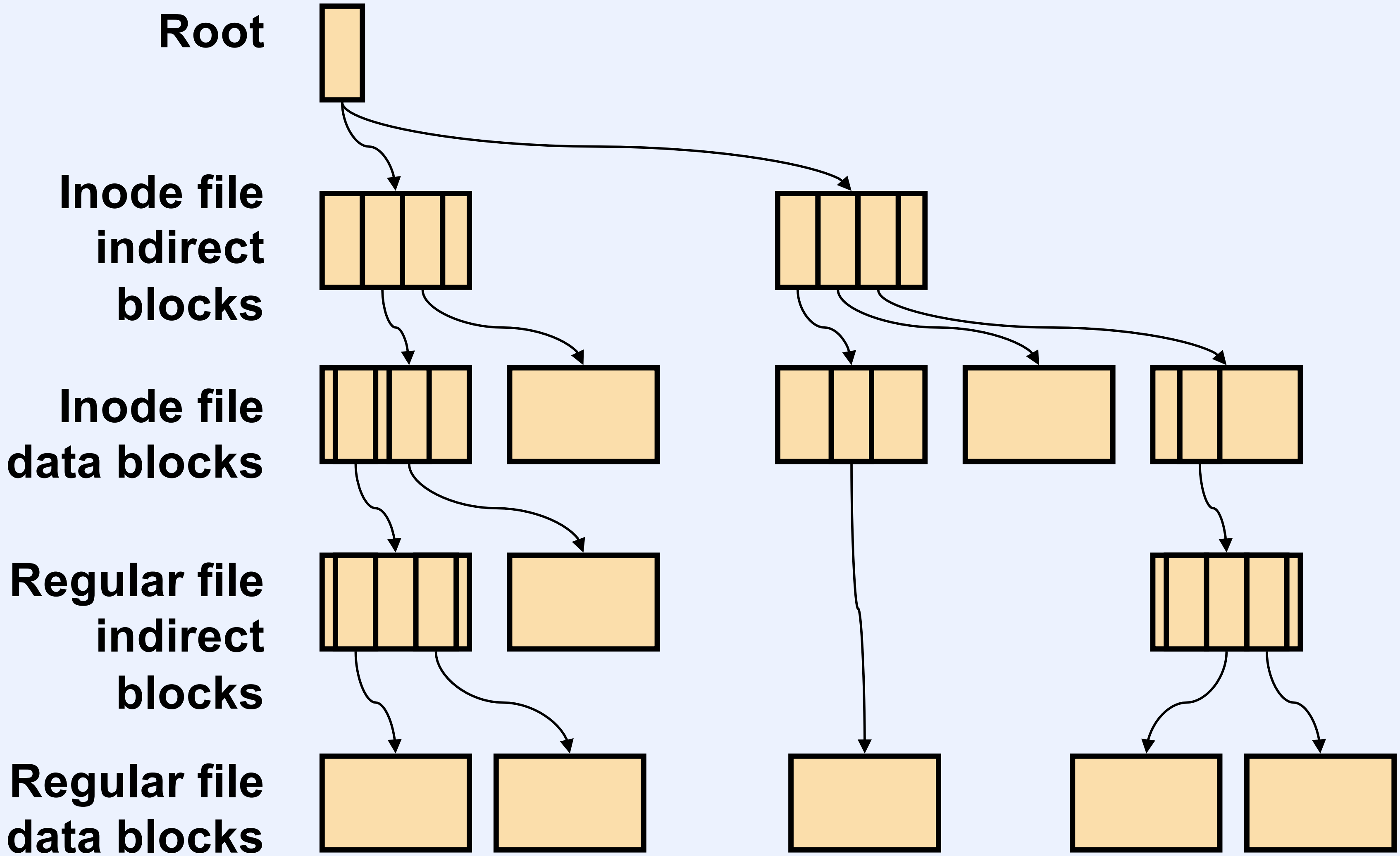
b) when the new contents of all blocks involved in the transaction have been checkpointed to the file system on disk

c) when the old contents of all blocks involved in the transaction have been written to the journal

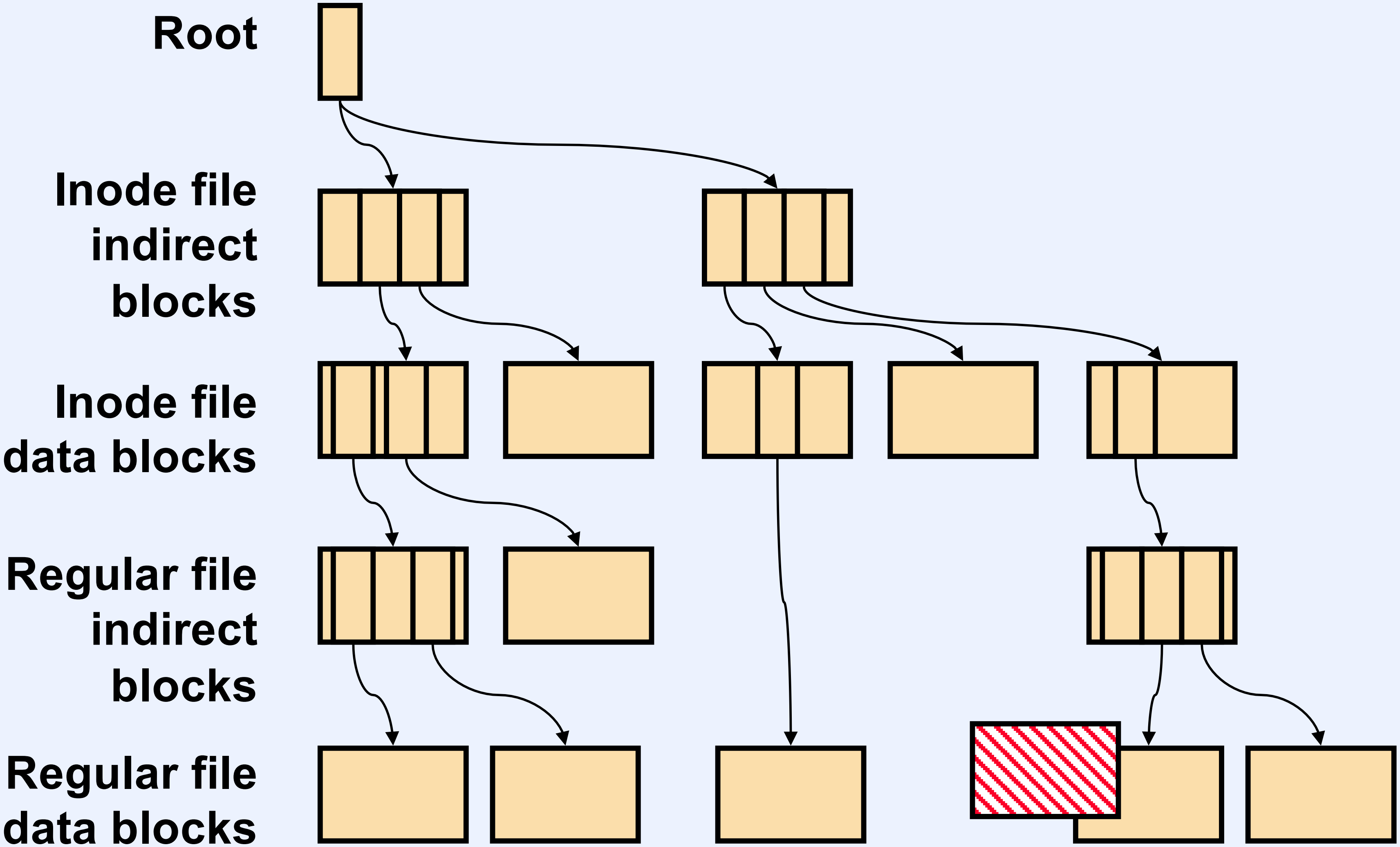
Shadow Paging

- Refreshingly simple
- Provides historical snapshots
- Examples
  - WAFL (Network Appliance)
  - ZFS (Sun)

Shadow-Page Tree



Shadow-Page Tree: Modifying a Node



Root

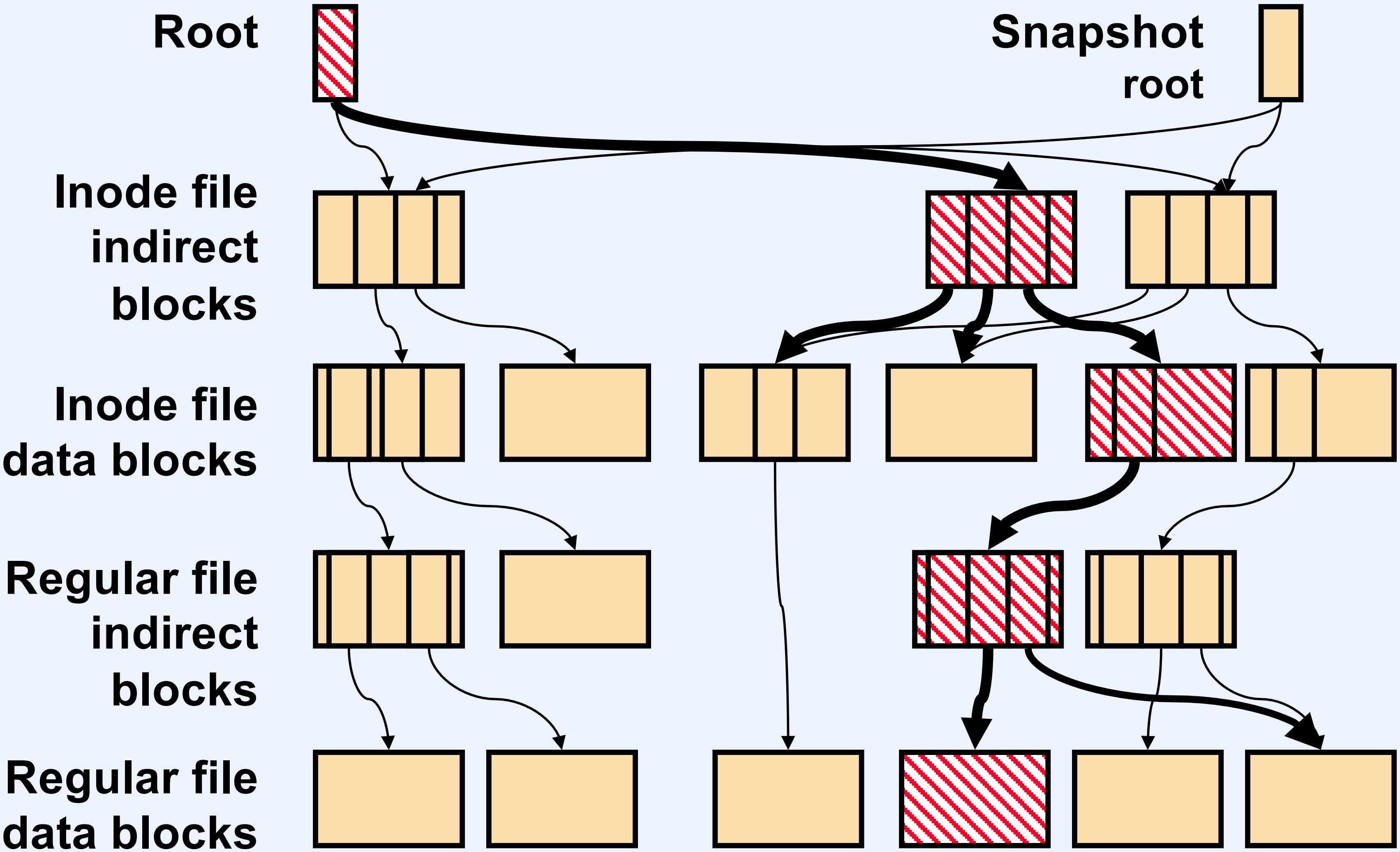
Inode file indirect blocks

Inode file data blocks

Regular file indirect blocks

Regular file data blocks

Shadow-Page Tree: Propagating Changes



Benefits

- Each update results in a new shadow-page tree (having much in common with the previous one)
- The current root identifies the current tree
- If the system crashes after an update has been made, but before changes are propagated to the new root, the update is lost
  - a single write (to the root) effectively serves as a commit
- Older roots refer to previous states of the file system – snapshots

Quiz 4

When the shadow-page tree is updated:

a) file-system data may be cached (and written asynchronously) as long as the root is written last

b) file-system data may be cached (and written asynchronously) only if all lower parts of the tree are written to disk before upper parts

c) all file-system data must be written to disk synchronously: writes may be cached for the sake of reads, but write system calls may not return until the data is on disk