File Systems Part 3

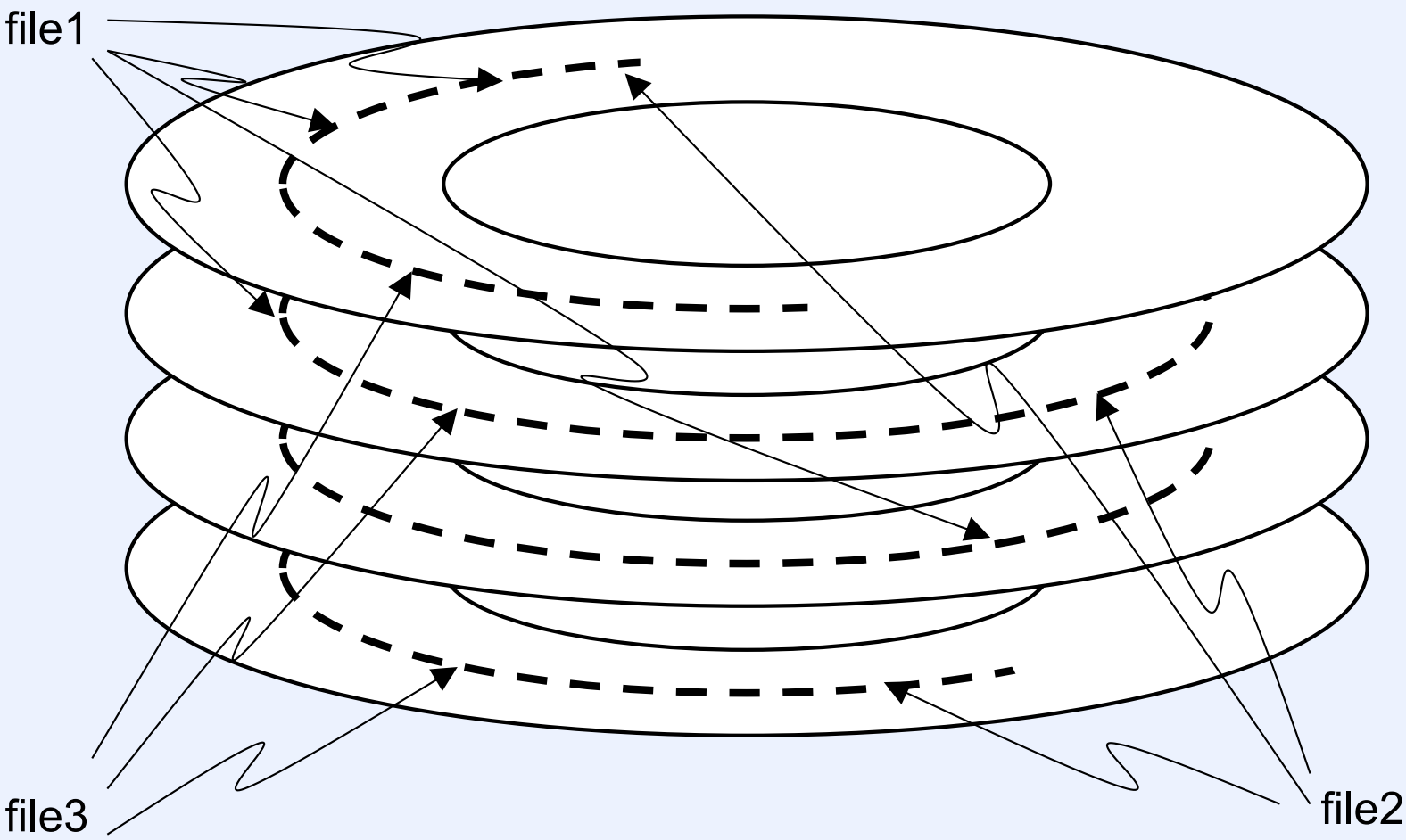
File Servers

- Servicing many unrelated clients
  - successive requests to file system come from different clients
  - how can we optimize the use of the disk?

A Different Approach

- We have lots of primary memory
  - enough to cache all commonly used files
- Read time from disk doesn't matter
- Time for writes does matter

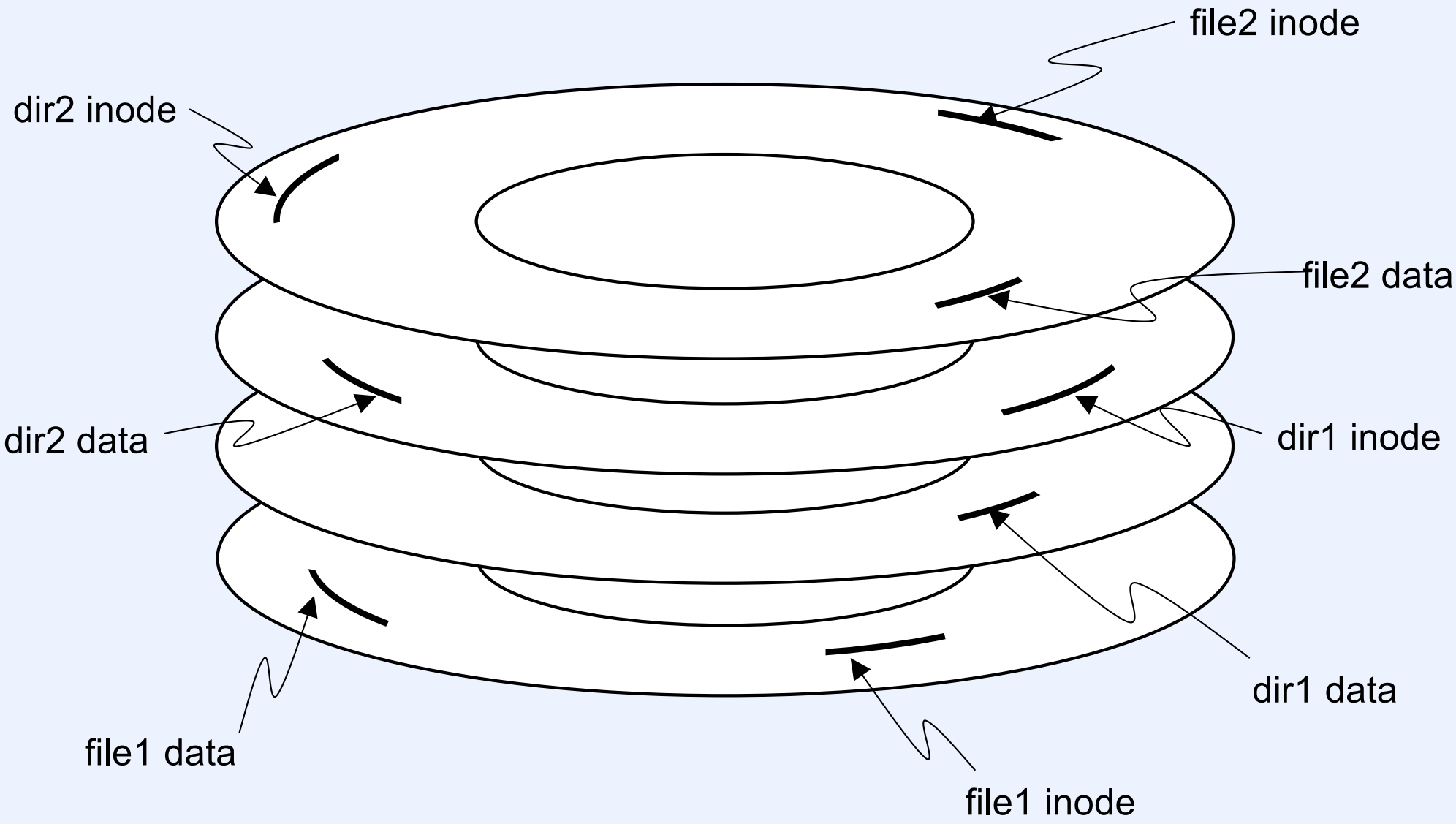
Log-Structured File Systems



Example

- We create two single-block files
  - *dir1/file1*
  - *dir2/file2*
- FFS
  - allocate and initialize inode for *file1* and write it to disk
  - update *dir1* to refer to it (and update *dir1* inode)
  - write data to *file1*
    - allocate disk block
    - fill it with data and write to disk
    - update inode
  - six writes, plus six more for the other file
    - seek and rotational delays

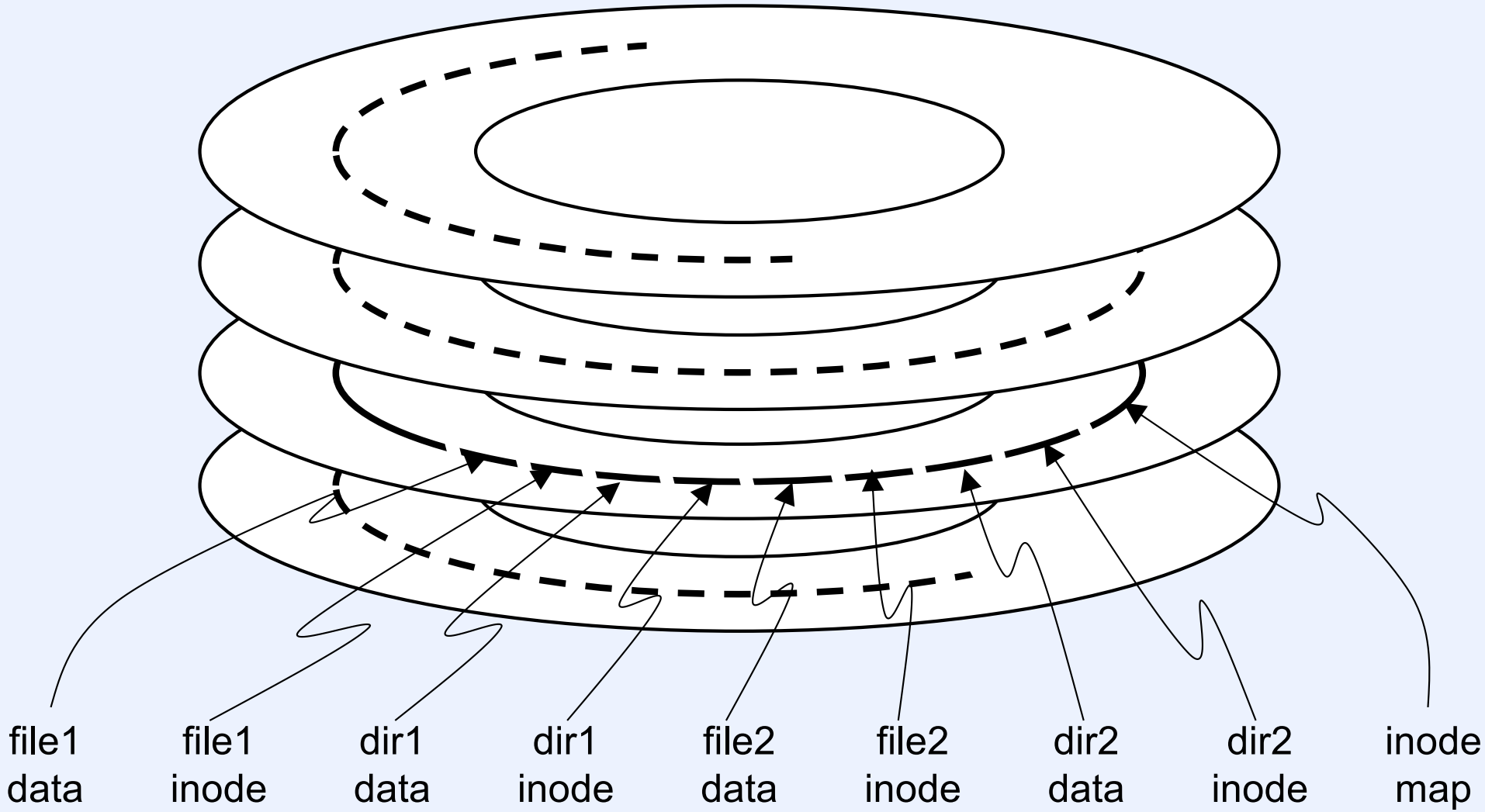
FFS Picture



Example (Continued)

- Sprite (a log-structured file system)
  - one single, long write does everything

Sprite Picture

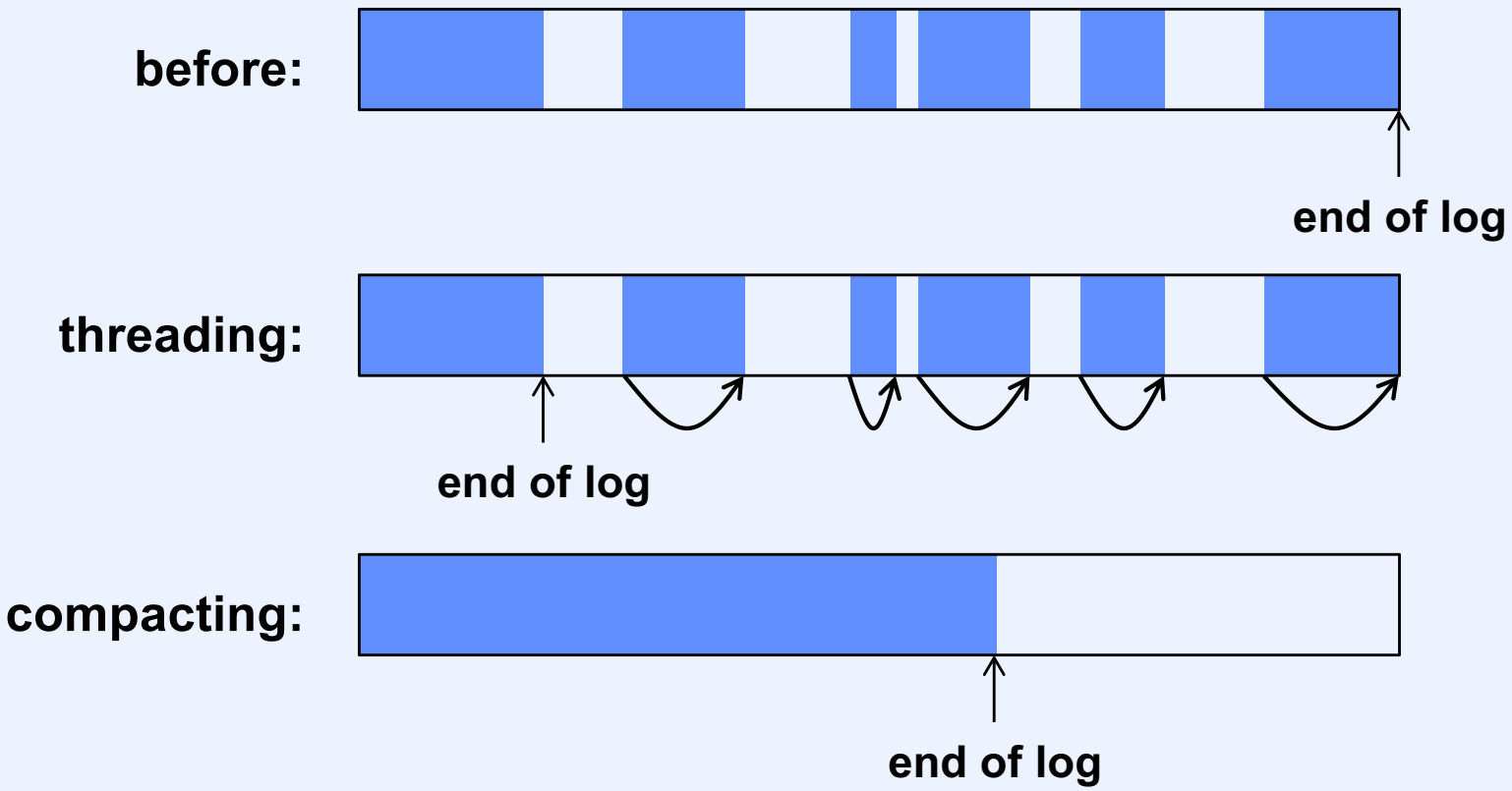


Some Details

- Inode map cached in primary memory
  - indexed by inode number
  - points to inode on disk
  - written out to disk in pieces as updated
  - checkpoint file contains locations of pieces
    - written to disk occasionally
    - two copies: current and previous
- Commonly/recently used inodes and other disk blocks cached in primary memory

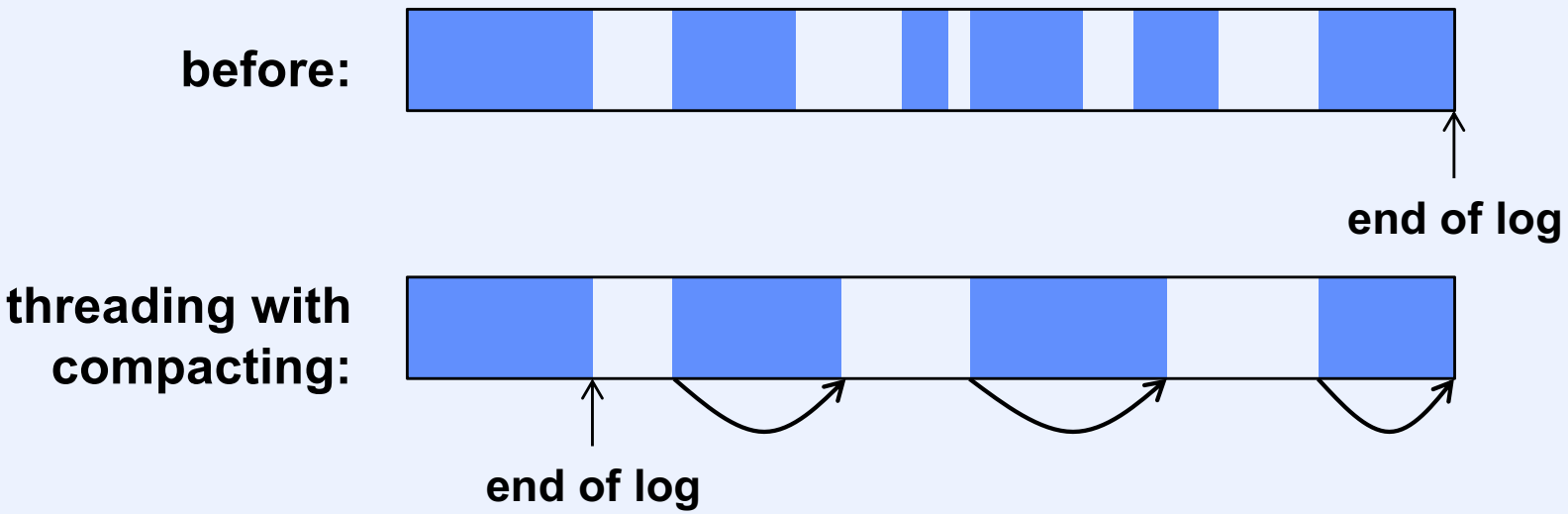
More Details

- What happens when end of log reaches end of disk?



Better Approach

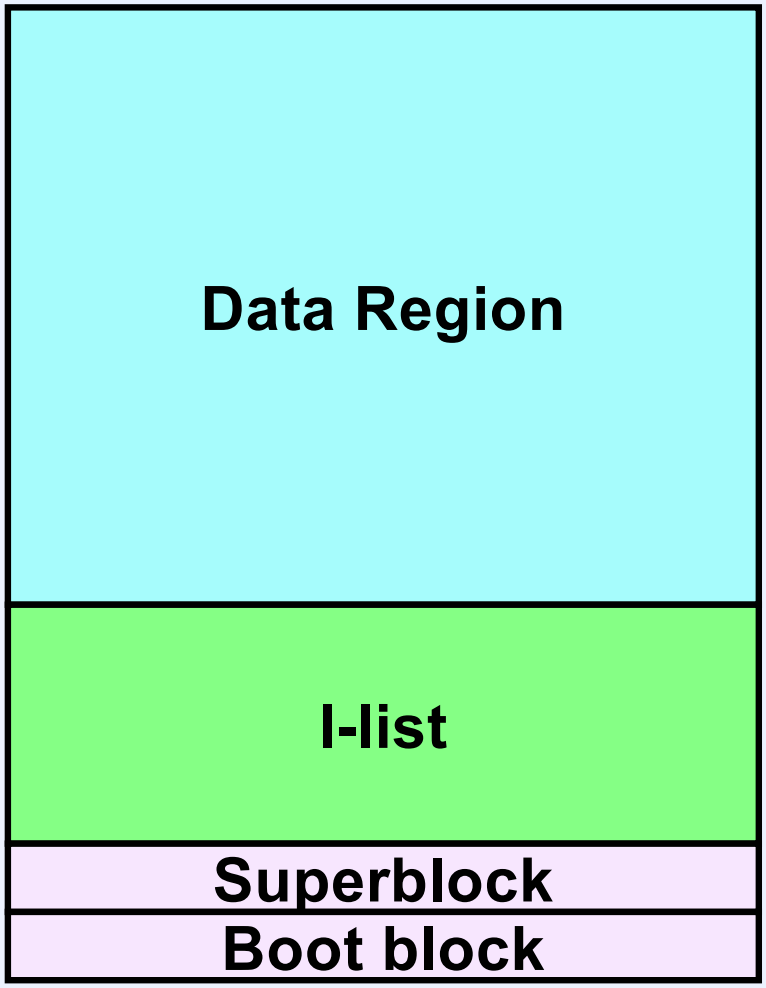
- Compact into largish segments



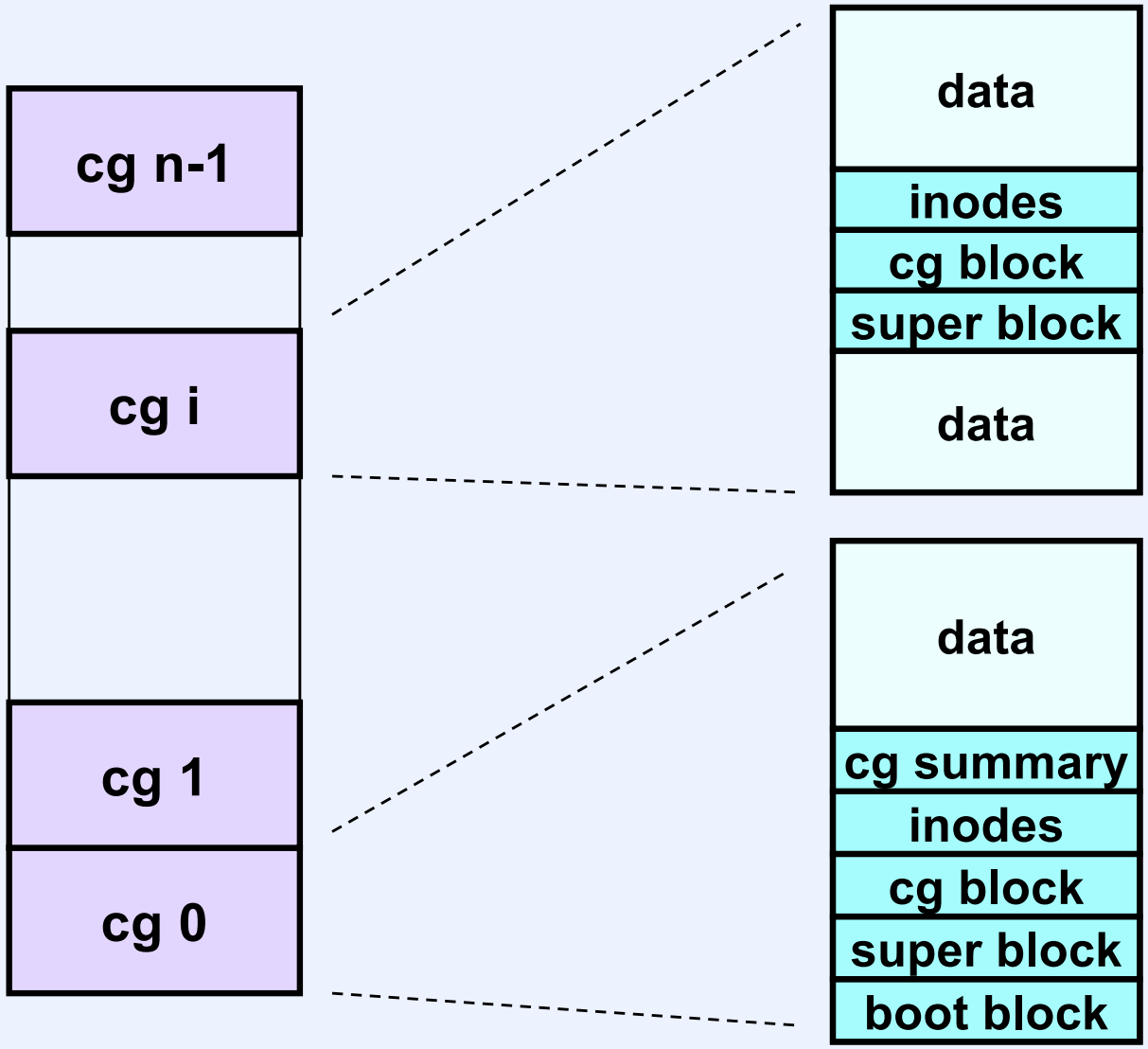
LFS

- Sprite was a proof of concept
  - developed by Mendel Rosenblum and John Ousterhout
- LFS (log-structured file system) extended Sprite
  - developed by Margo Seltzer, Keith Bostic, Kirk McKusick, and Carl Staelin
  - added extents
  - better integrated into Unix

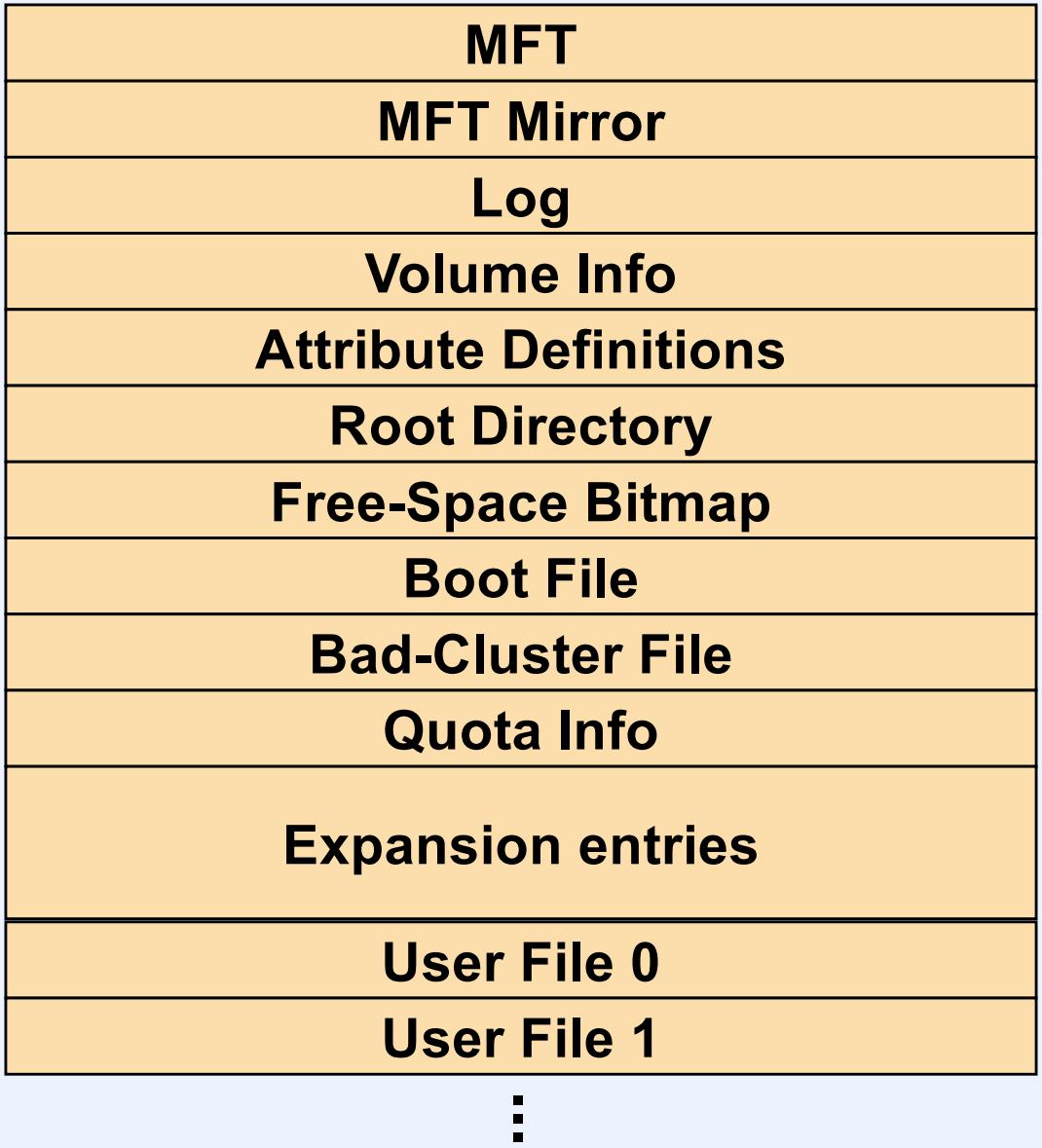
S5FS Layouts



FFS Layout



NTFS Master File Table



Quiz 1

Your disk drive is nearing capacity. So you buy a new, larger drive to replace it. You copy your old disk drive to the new one. What else has to be done to take advantage of the larger disk? Assume you're using NTFS.

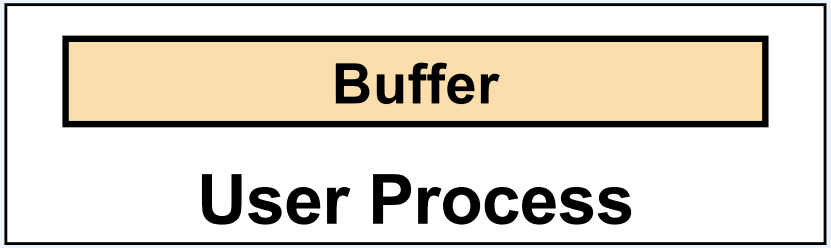
a) Nothing

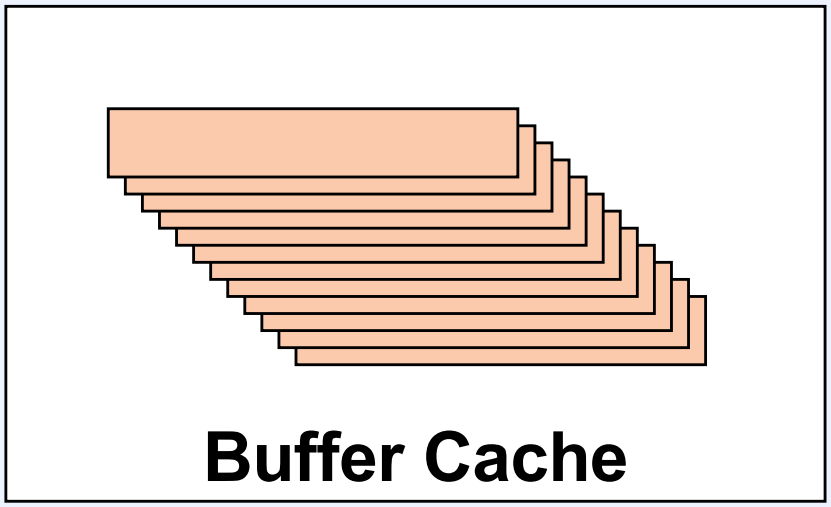
b) Modify the free-space bitmap

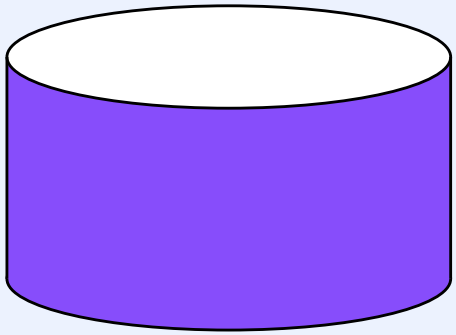
c) Modify the free-space bitmap and adjust the extent lists for all files

d) All of the above, plus more

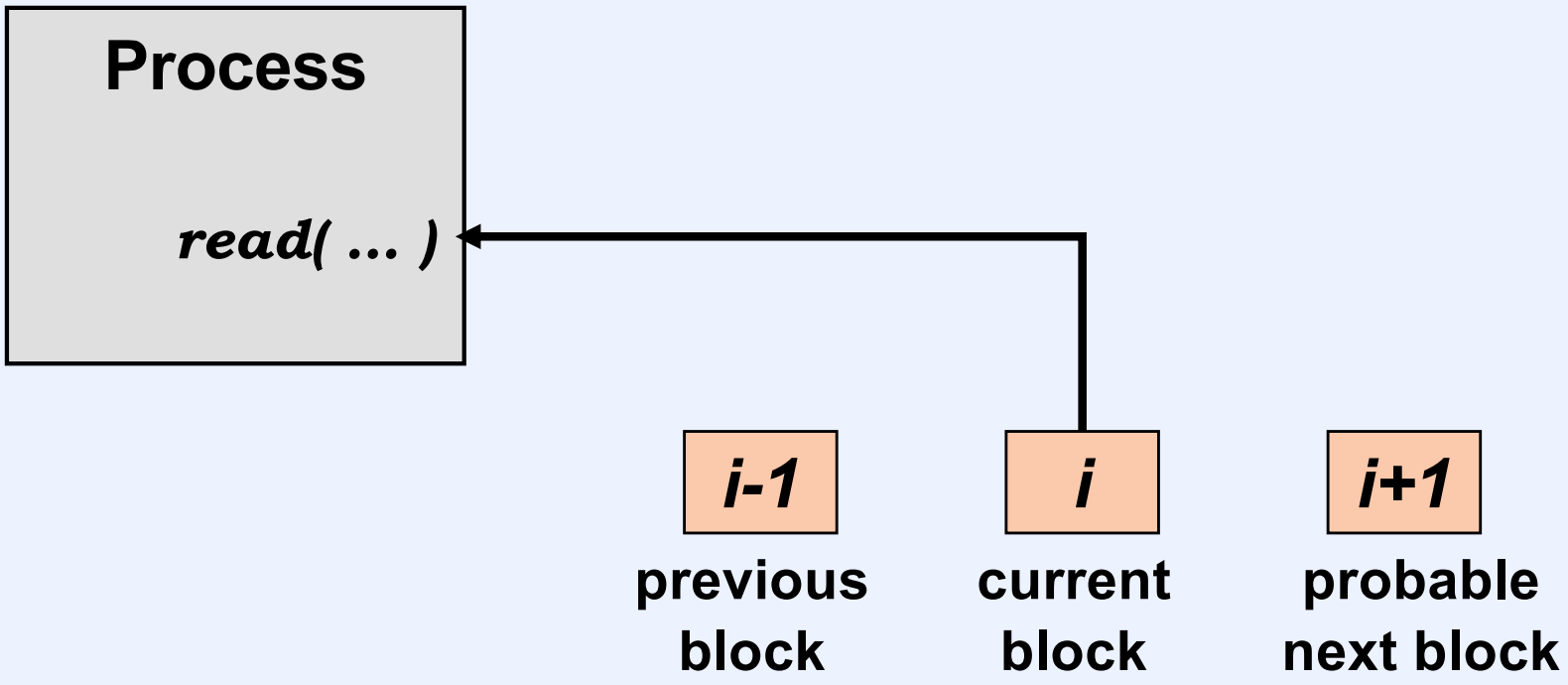
The Buffer Cache



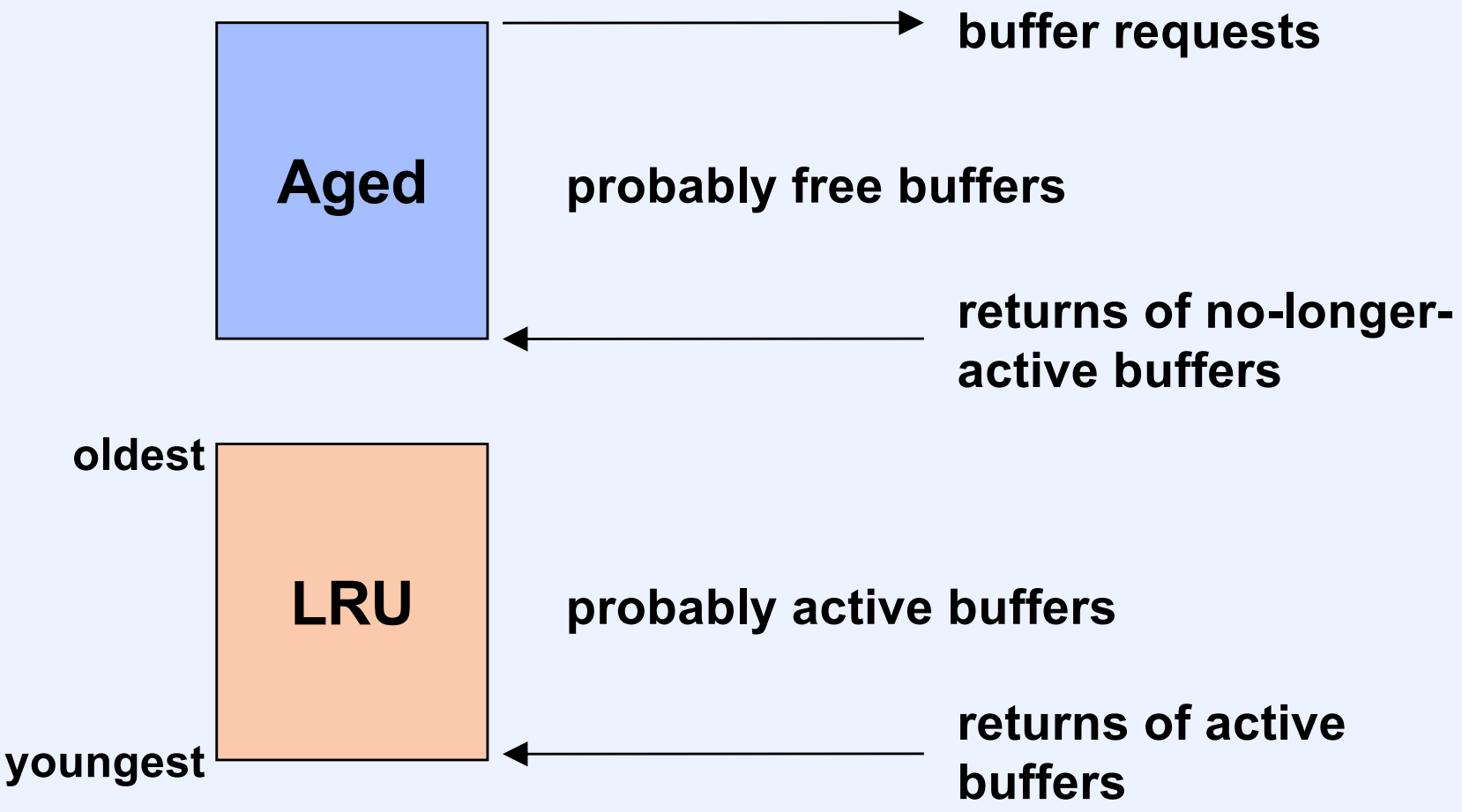




Multi-Buffered I/O



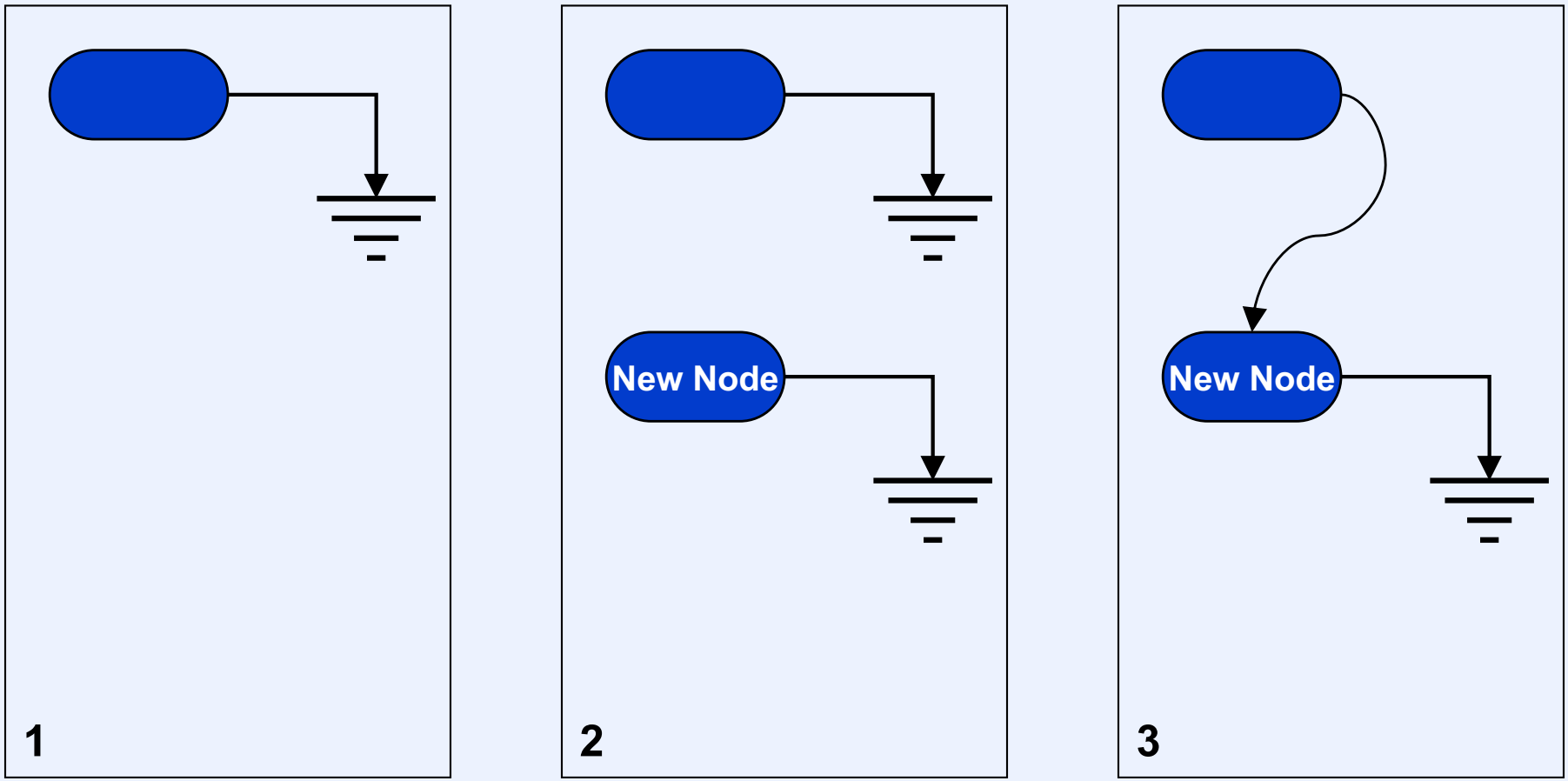
Maintaining the Cache



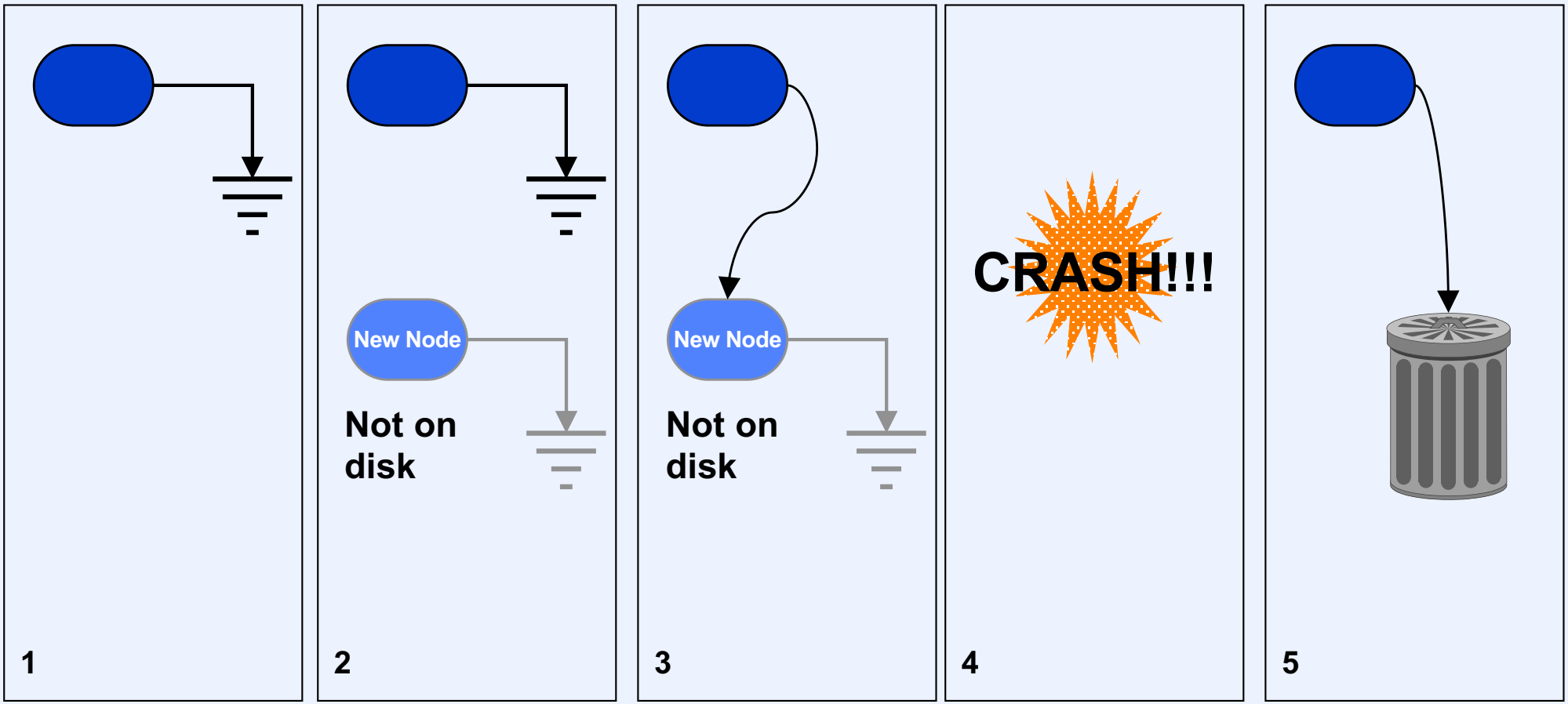
In the Event of a Crash …

- Most recent updates did not make it to disk
  - is this a big problem?
    - equivalent to crash happening slightly earlier
      - but you may have received (and believed) a message:
        - "file successfully updated"
        - "homework successfully handed in"
        - "stock successfully purchased"
    - there's worse …

File-System Consistency (1)



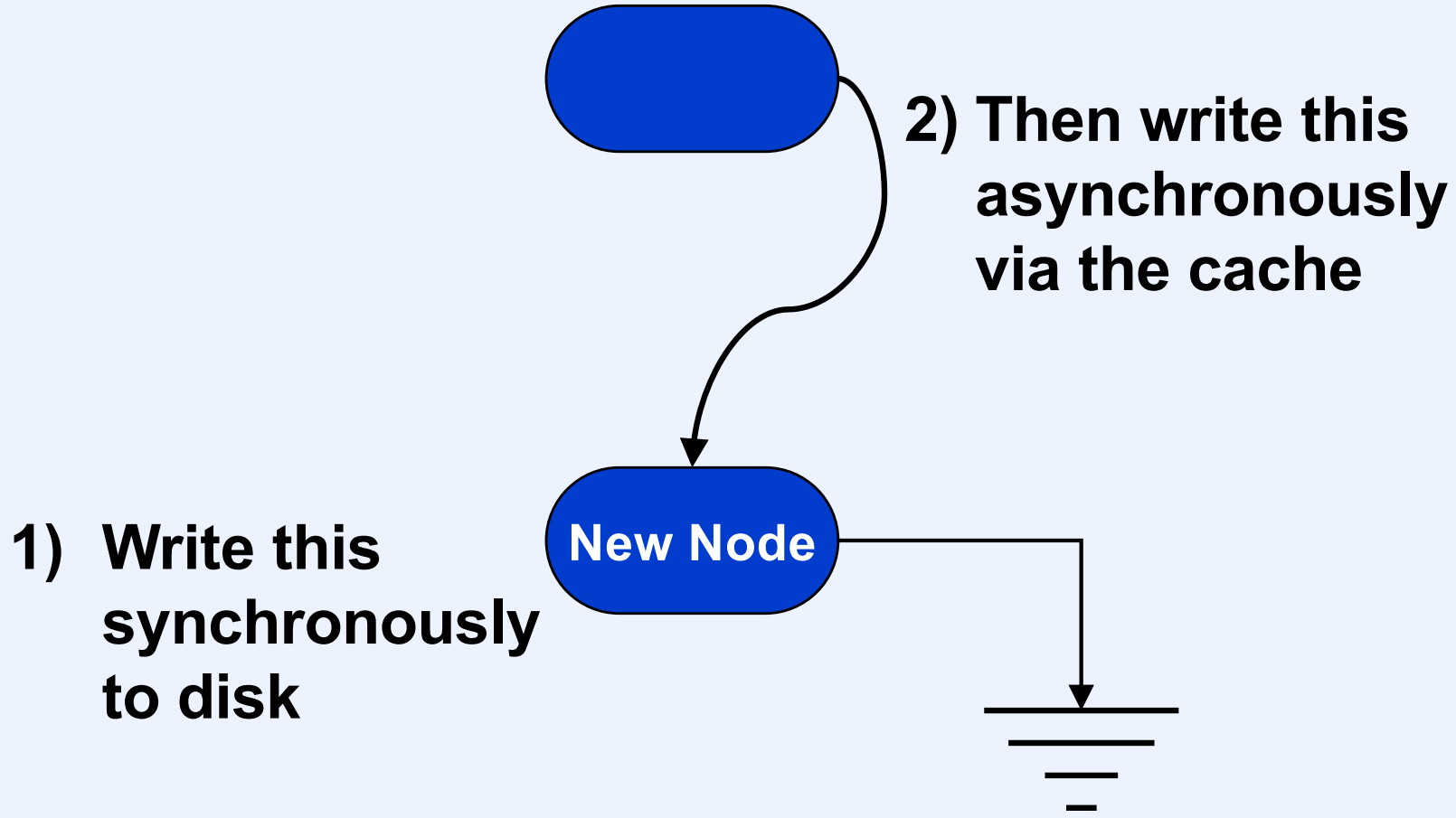
File-System Consistency (2)



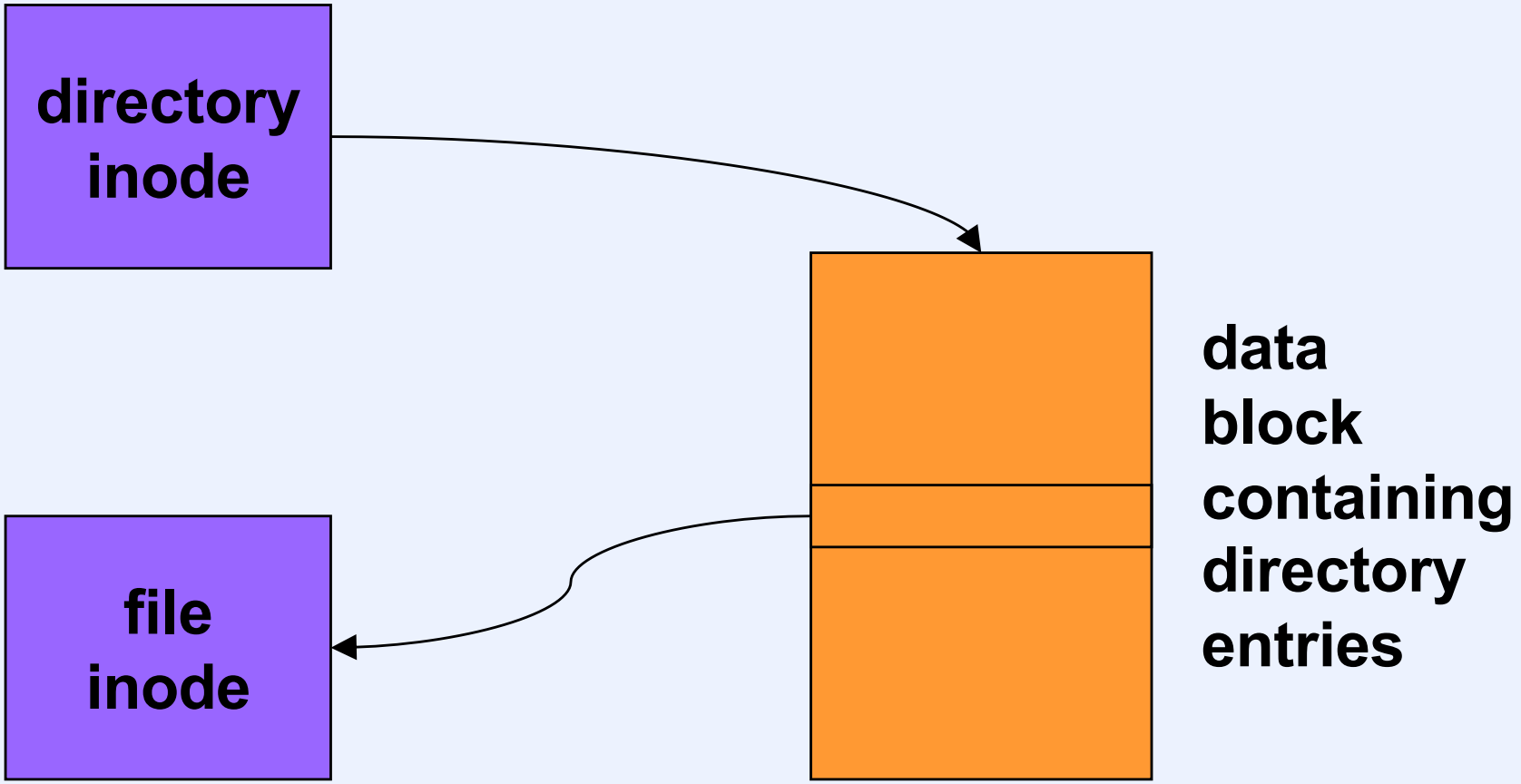
How to Cope …

- Don’t crash
- Perform multi-step disk updates in an order such that disk is always consistent — the *consistency-preserving approach*
- Perform multi-step disk updates as *transactions* — implemented so that either all steps take effect or none do

Maintaining Consistency



Which Order?



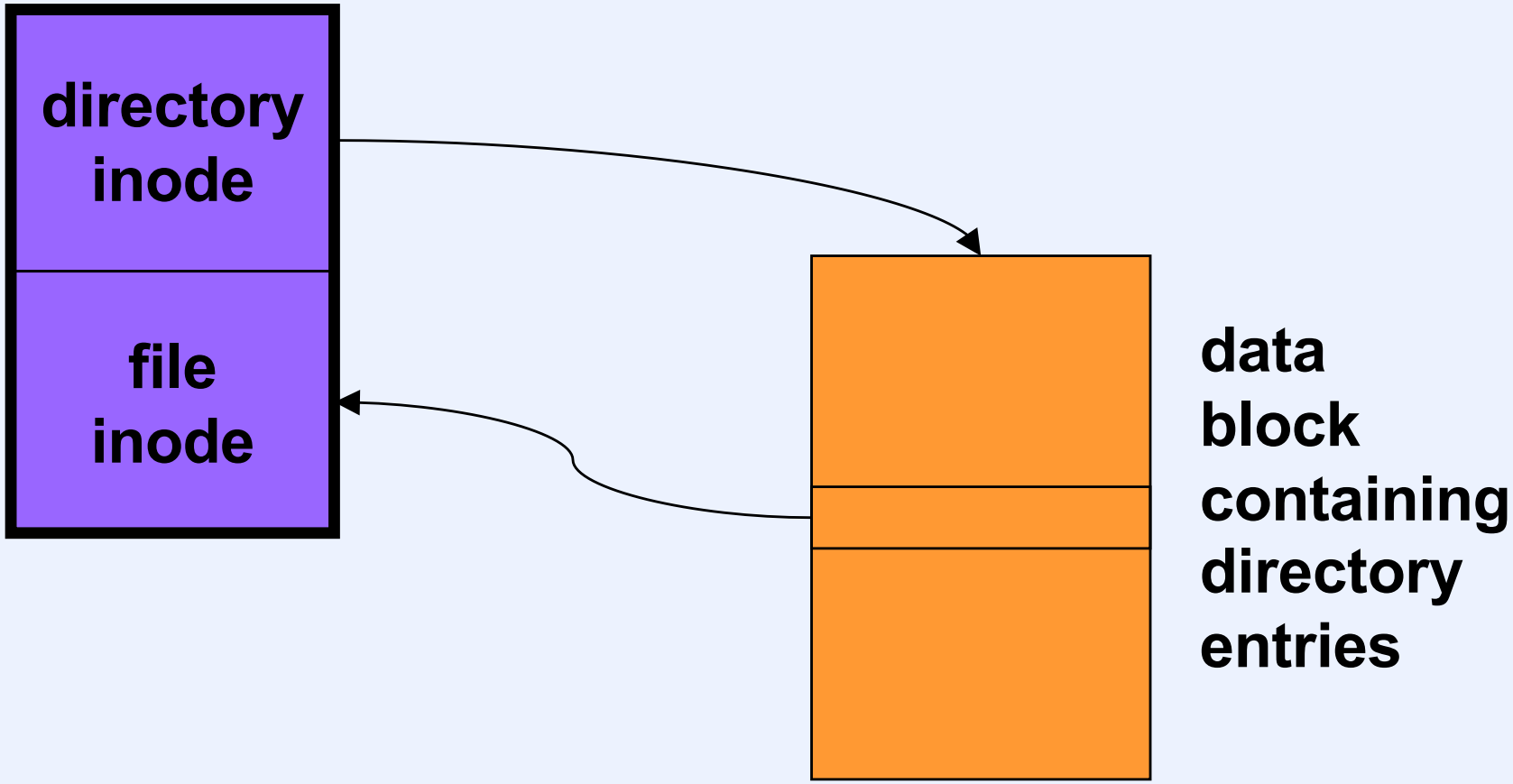
Synchronous FFS

- Updates to system data structures performed in cache in correct order to maintain consistency
- Each update is written synchronously to disk before the cache may be modified with the next update
- Result
  - no loss of consistency after a crash
  - system is very slow

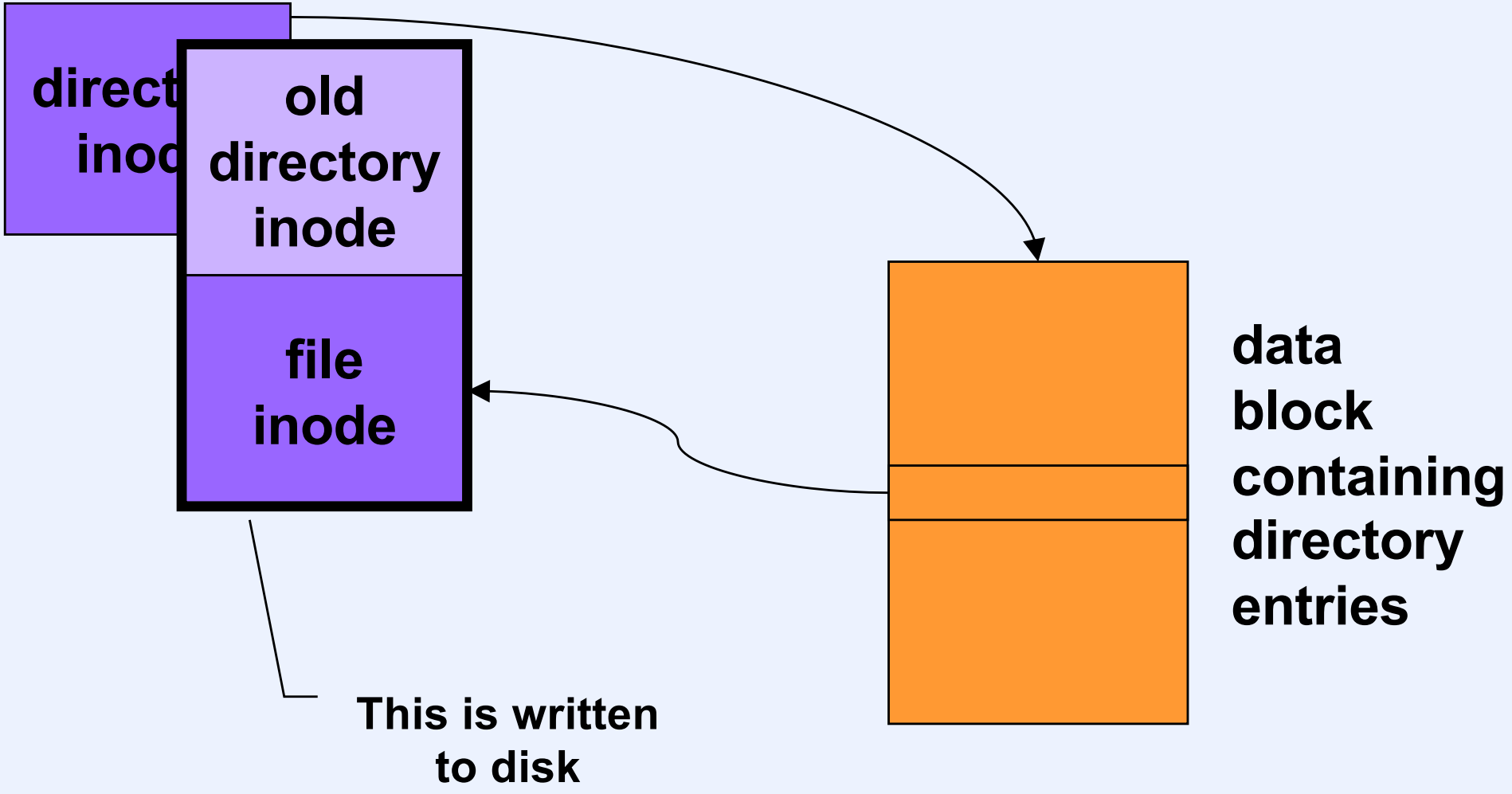
Soft Updates

- An implementation of the consistency-preserving approach
  - should be simple:
    - update cache in an order that maintains consistency
    - write cache contents to disk in same order in which cache was updated, but without requiring synchronous writes
  - isn't …

Problem …



Soft Updates



Quiz 2

Continuing with our example, suppose the system crashes just after the data block containing the new directory entries and the data block containing the new file inode are written to disk. When the system comes back up:

a) no recovery is required and nothing is lost

b) it writes the updated directory inode to disk

c) the update to the directory might be lost and space may need to be reclaimed

Quiz 3

When a disk block is allocated, three things happen: the free vector is modified, some data structure (perhaps an inode) is modified to refer to the new block, and the new block is written to. What is the correct order for updating the disk, so as to preserve consistency?

a) inode, free vector, disk block
b) inode, disk block, free vector
c) disk block, inode, free vector
d) free vector, inode, disk block
e) free vector, disk block, inode

Soft Updates in Practice

- Implemented for FFS in 1994
- Used in FreeBSD's FFS
  - improved performance (over FFS with synchronous writes)
  - disk updates may be many seconds behind cache updates

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

Example Transactions (1)

- Create file
  - create inode
    - modify free vector/list
    - initialize inode
  - update directory
    - modify contents
      - possibly modify free vector
    - update directory inode

Example Transactions (2)

- Rename file
  - update new directory
    - update new directory inode
    - modify new directory
      - update free vector
  - update old directory
    - update old directory inode
    - modify old directory
      - update free vector

Example Transactions (3)

- Write to a file
  - for each block
    - update free vector
    - copy data to block
  - update inode

Example Transactions (4)

- Delete a file
  - for each block
    - update free vector
  - update inode free vector/list
  - update directory
    - update directory inode
    - modify directory
      - update free vector

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