CSE 451

Module 11
Berkeley Log-Structured File System

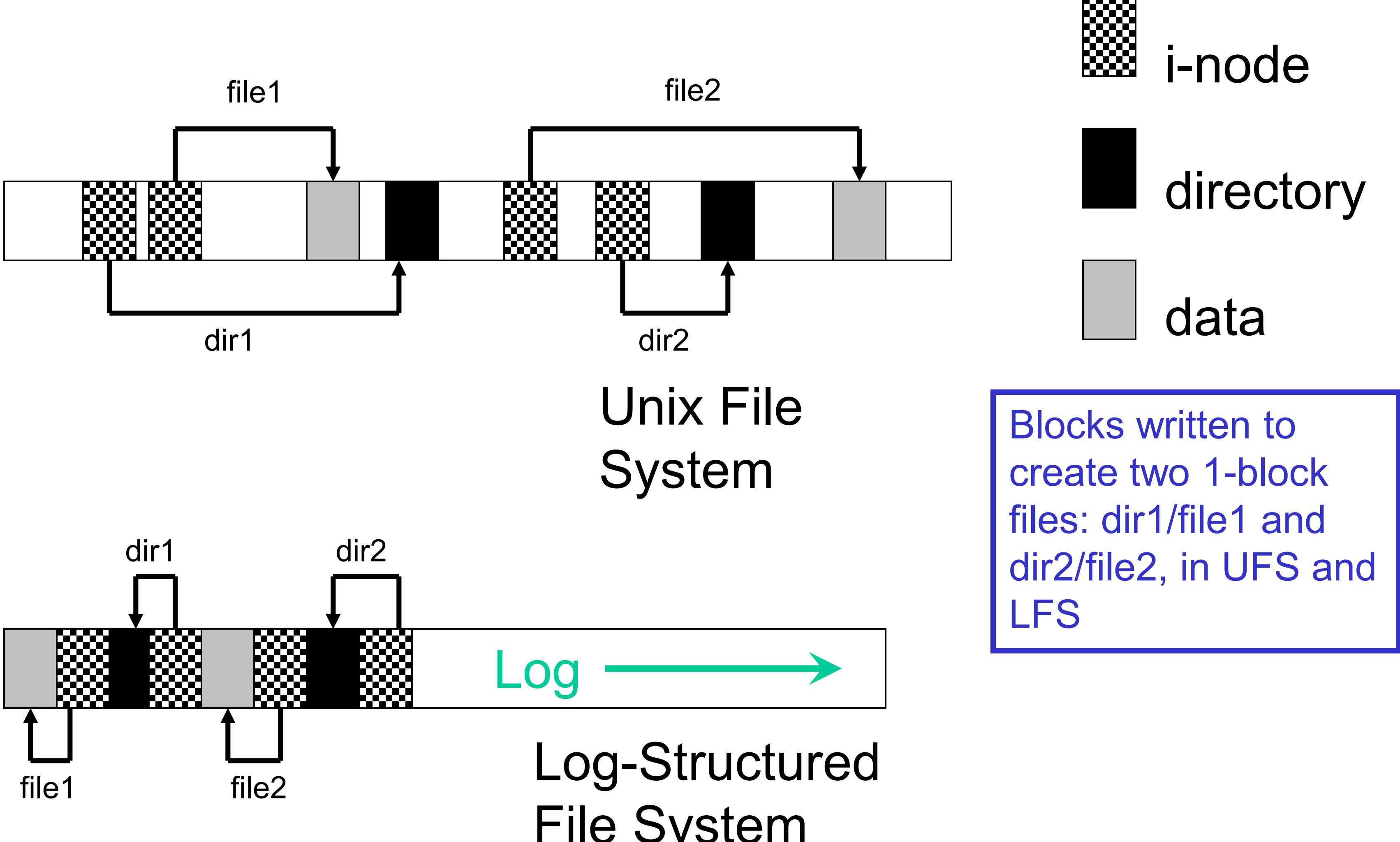
LFS inspiration

- Memory caching is generally effective
  - Result is that most physical disk operations are writes
    - Can delay writing only so long
    - Writes to journal (redo log) are writes
- Suppose all writes to disk were written as a log (i.e., appended)
  - log includes modified data blocks and modified metadata blocks
  - buffer a huge block (“segment”) in memory
  - when full, write it to disk in one efficient contiguous transfer
- So the disk contains a single long log of changes, consisting of threaded segments
- Reminds you of journaling?
  - Yes, except that there is no “home location” for data or metadata
  - The log is all there is

LFS basic approach

- Use the disk as a *log*
  - A log is a data structure that is written only at one end
- If the disk were managed as a log
  - [spinning] there would be effectively no seeks (for writes)
  - you'd be updating an entire erasure block every write
  - you'd (hope to) be spreading updates across the device
- New data and metadata (i-nodes, directories) are accumulated in the buffer cache, then written all at once in large blocks
- If you write enough data at once, you can achieve close to the transfer rate of the device
  - both spinning and SSD
- Sounds simple – but complicated under the covers

LFS vs. FFS



Blocks written to create two 1-block files: dir1/file1 and dir2/file2, in UFS and LFS

LFS Challenges

- Locating data on the disk
  - FFS place inodes in a well-known location
  - LFS writes data “at the end of the log”
- Managing free space on the disk
  - Disk is finite, and therefore log must be finite
  - So cannot just keep appending to log, ad infinitum!
    - need to recover space used by deleted blocks in log
    - need to fill holes in segments created by recovered blocks
      - why?
    - “cleaning”
- Note:
  - in-memory caching is the same as before
  - reads that go to disk are the same as FFS, once you find the i-node

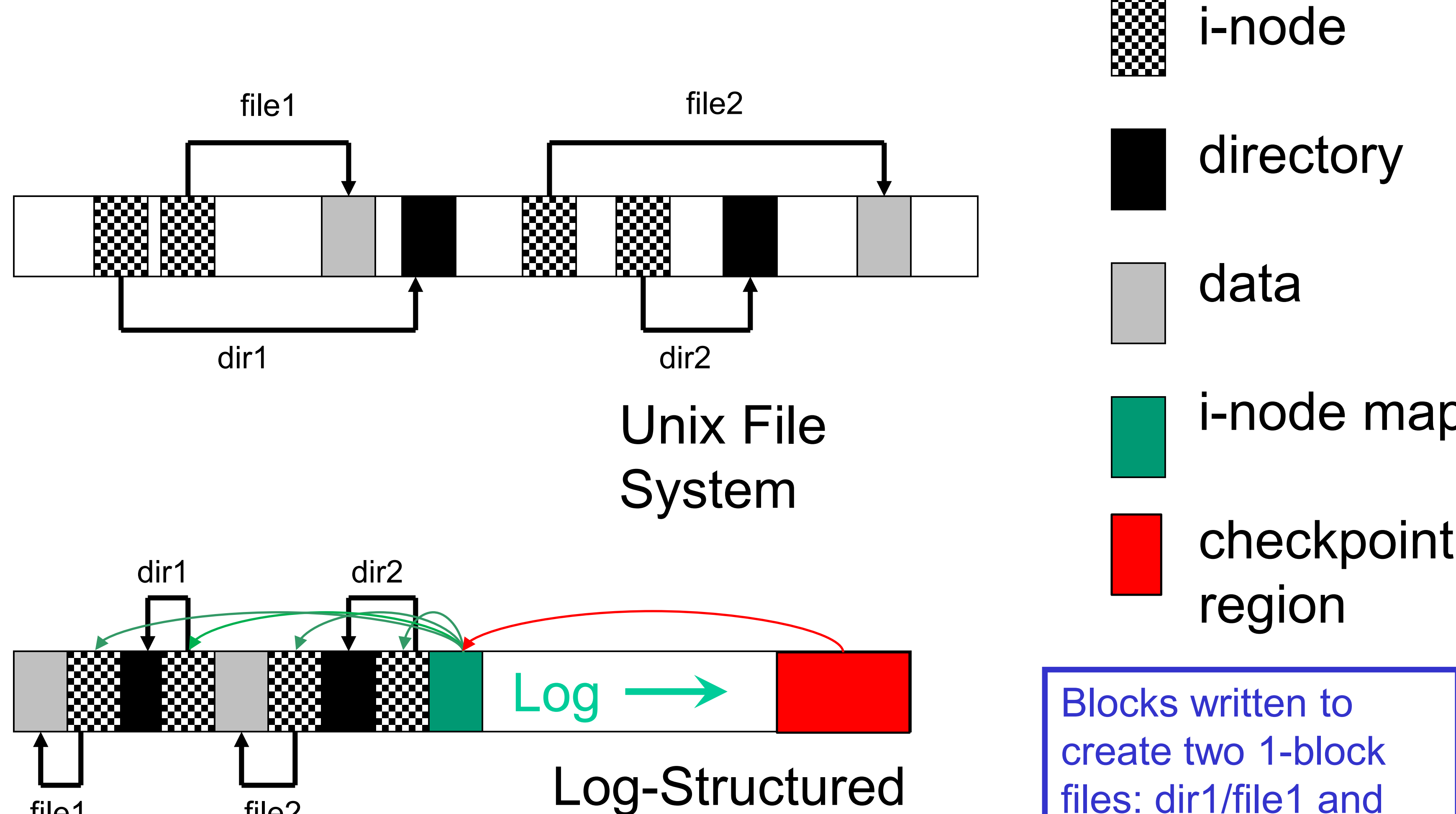
LFS: Locating data and i-nodes

- Data
  - LFS uses i-nodes to locate data blocks, just like FFS
- i-nodes
  - i-nodes are appended to end of log, not at all like FFS
  - i-node number is no longer an index in an array, it's just a name
- How to locate i-node on disk?
  - Use another level of indirection
    - **i-node maps**
    - i-node # → i-node location

LFS: Locating the i-node map

- Use another level of indirection
  - **i-node maps:** i-node # → i-node location
  - the map is indexed by the i-node #
  - i-node map is a logical structure, kept on disk
  - it's updated often →
    - don't store as an array in a fixed location, instead
    - write changes to it to the log (!)
- How do you find the i-node map?
  - location of i-node map blocks are kept in a **checkpoint region**
  - checkpoint region has a fixed location
    - two copies, actually
      - why?
  - cache these structures in memory for performance

LFS vs. FFS



Blocks written to create two 1-block files: dir1/file1 and dir2/file2, in UFS and LFS

LFS: File reads and writes

- Reads are no different than in FS/FFS, once we find the i-node for the file
  - The i-node map, which is cached in memory, gets you to the i-node, which gets you to the blocks
- Every write causes new blocks to be added to the tail end of the current "segment buffer" in memory
  - When the segment is full, it's written to disk

LFS: Free space management

- Writing segments to the log eats up disk space
- Over time, segments in the log become fragmented as we replace old blocks of files with new blocks
  - live i-nodes no longer point to blocks, but those blocks still occupy their space in the log
    - "dead i-nodes" provide opportunity to save versions of file system
- Garbage-collect segments
  - coalesce "live" data from sparsely populated segments into fully populated segments
  - results in a "clean segment" that can be fully written / reused

LFS: Segment cleaning

- Cleaning is an issue
  - costly overhead, when do you do it?
- A cleaner daemon cleans old segments, based on
  - utilization: how much is to be gained by cleaning?
  - age: how likely is the segment to change soon?

LFS summary

- As caches get big, most reads will be satisfied from the cache
  - No matter how you cache write operations, though, they are eventually going to have to get back to disk
  - Thus, most disk traffic will be write traffic
- If you eventually put blocks (i-nodes, file content blocks) back where they came from, then even if you schedule disk writes cleverly, you're effectively doing random writes (bad)
- Instead, do all writes as appends to disk log
  - A modest amount of data is located in a fixed location, so that you can find the i-nodes, and is updated only occasionally

- What happens when a crash occurs?
- Suppose you have to read a file?
- How do you prevent overflowing the disk?