Log File System

Operating Systems
Based on: Three Easy Pieces by Arpaci-Dusseaux

Carmi Merimovich

Tel-Aviv Academic College

Motivation

- RAM is getting bigger: Reads served from cache.
- Sequential disk performance is much better than Random disk performance.
- Traditional FSs do rather bad on many workloads.

Hence.

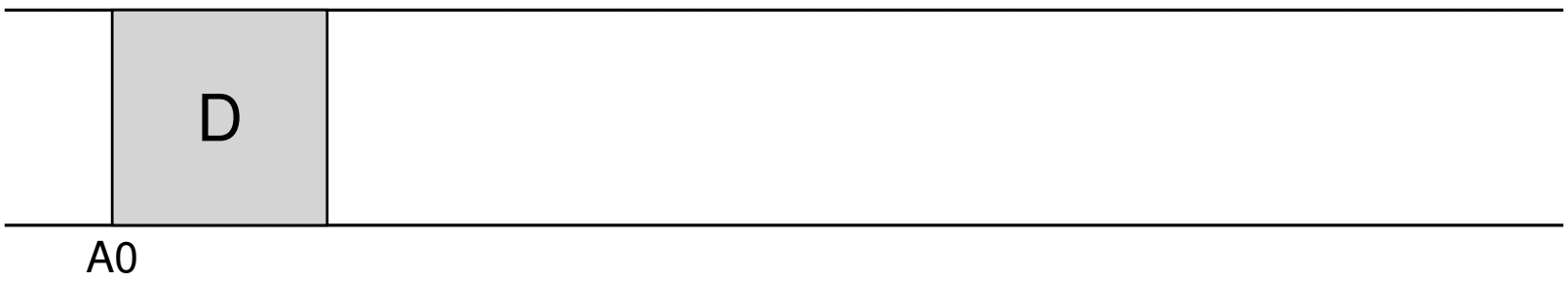
Design FS which utilizes disk sequential performance.

Log-structured File System

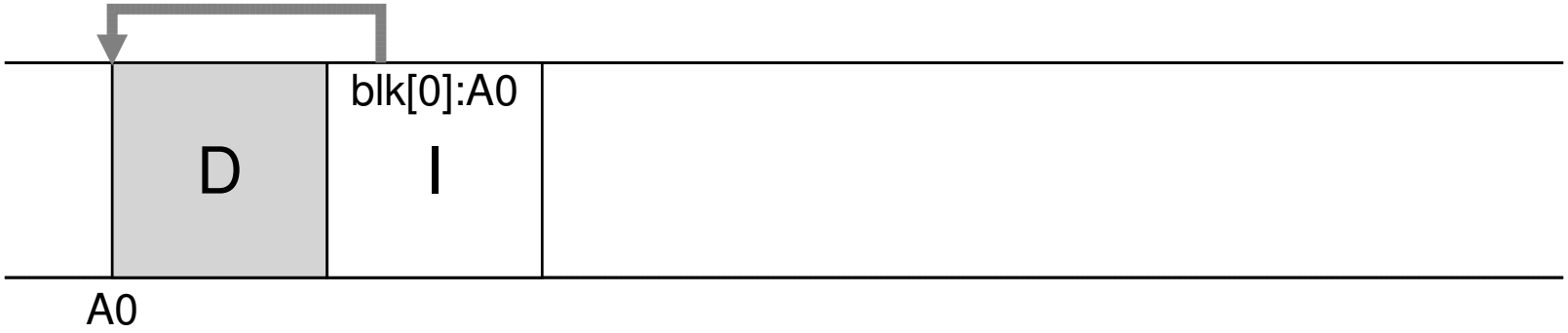
Buffer writes into a long segment, both data and metadata.
Write the segment in one operation to the disk to a free area on disk.

Example

Writing data block D to physical block A0:



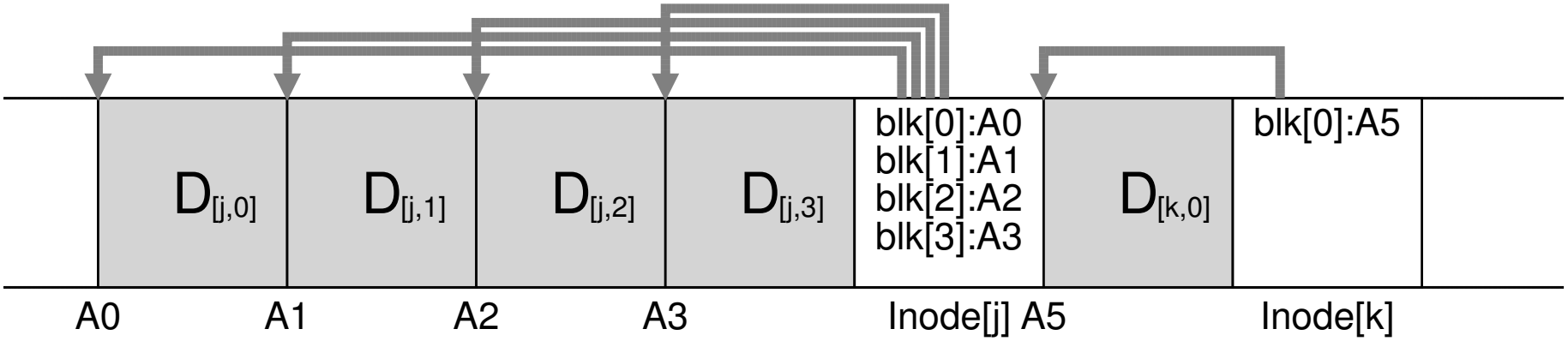
In reality we need also to write an updated inode. Hence.



This is the idea. We never write **in-place**.

Buffering

- Many small sequential writes does not yield peak performane.
- Buffer many writes into long segments.
- Write four blocks to file j and one block to file k.

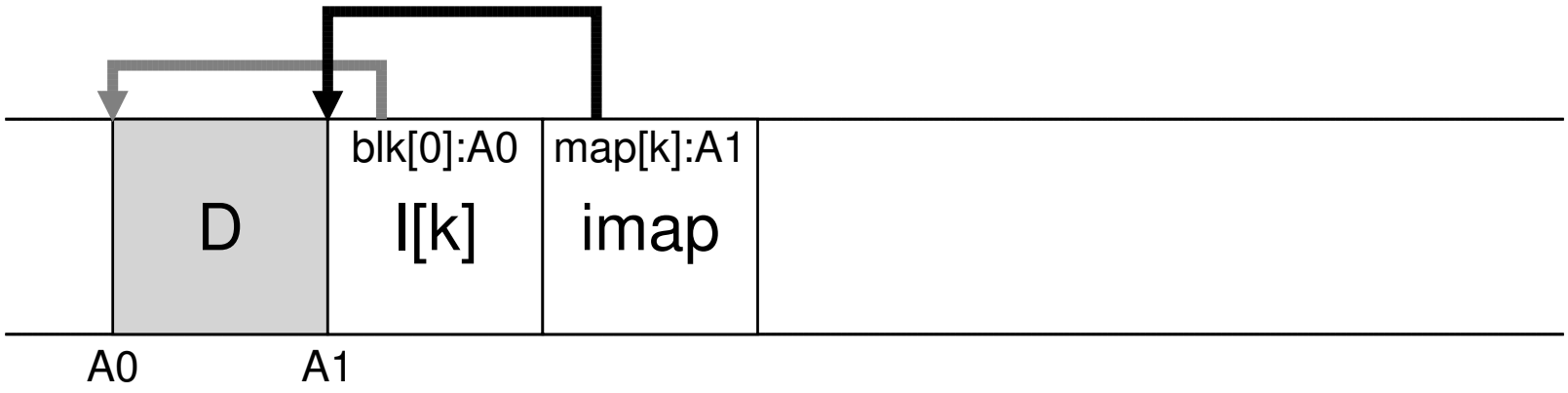


inodes?

- Obviously the inodes are moving...
- How do we find them?
  - UFS: Trivial.
  - FFS: almost trivial.
  - LFS: indirection.

The imap

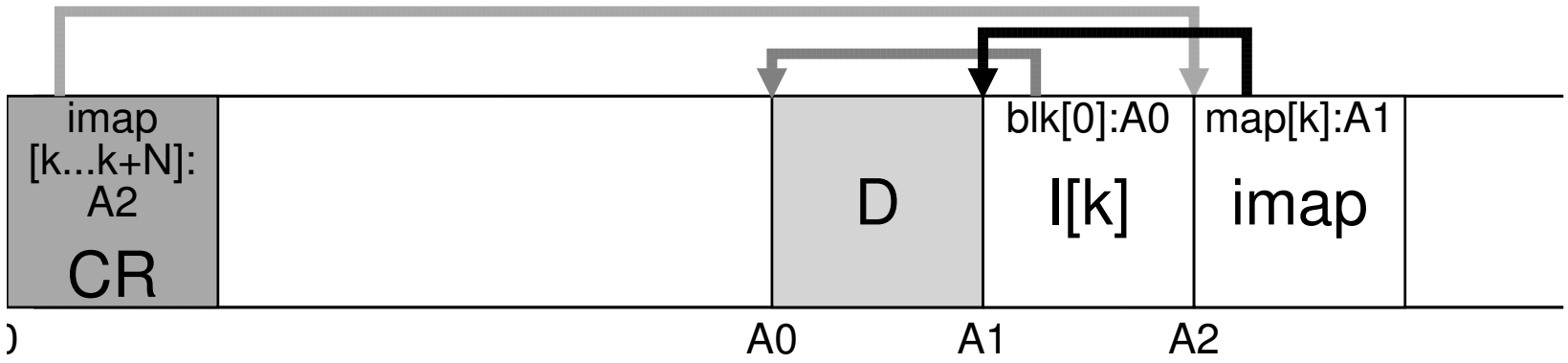
- A structure mapping inode number to its 'last' position on disk.
  - Can this be an array in memory?
  - Can it be in a fixed place on disk?



- A piece of the imap is added to the segment.
- What is the problem now??????

The Checkpoint Region

- No escape from a fixed place on disk...



- In the CR there are pointers to the imap chunks written to the Disk.

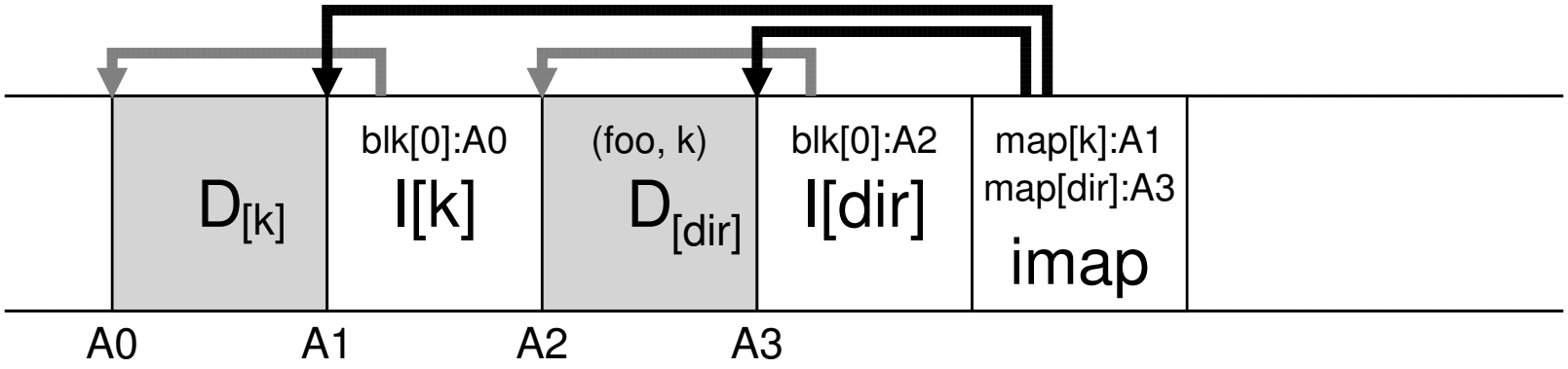
Reading Block of File

- Build the inode map from the Critical Region.
- From inode number and the imap find the block hosting the inode.
- Read the inode and proceed as usual.

Since the imap is in RAM we have the same number of reads as a traditional FS.

Folders?

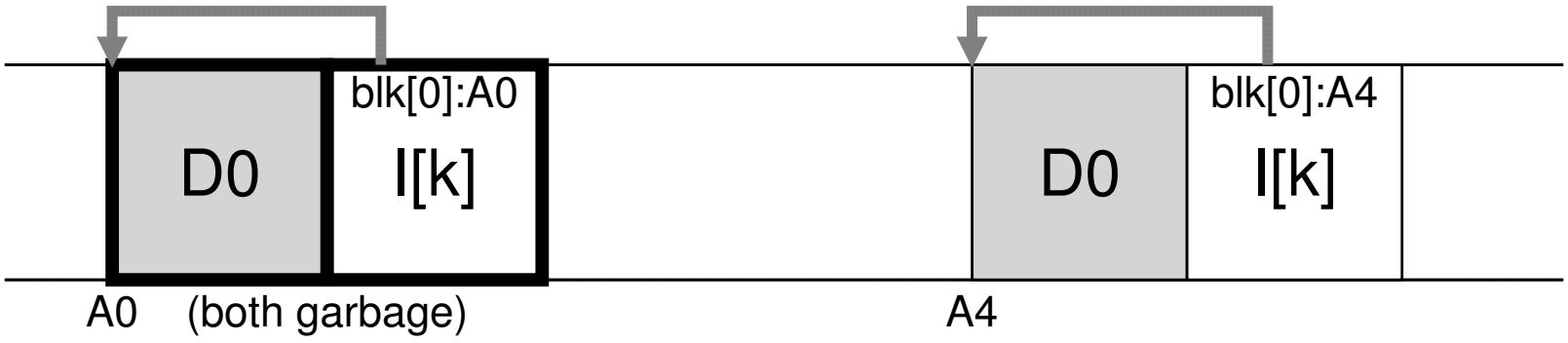
- Since folders are just file with OS interest in, it is the same.



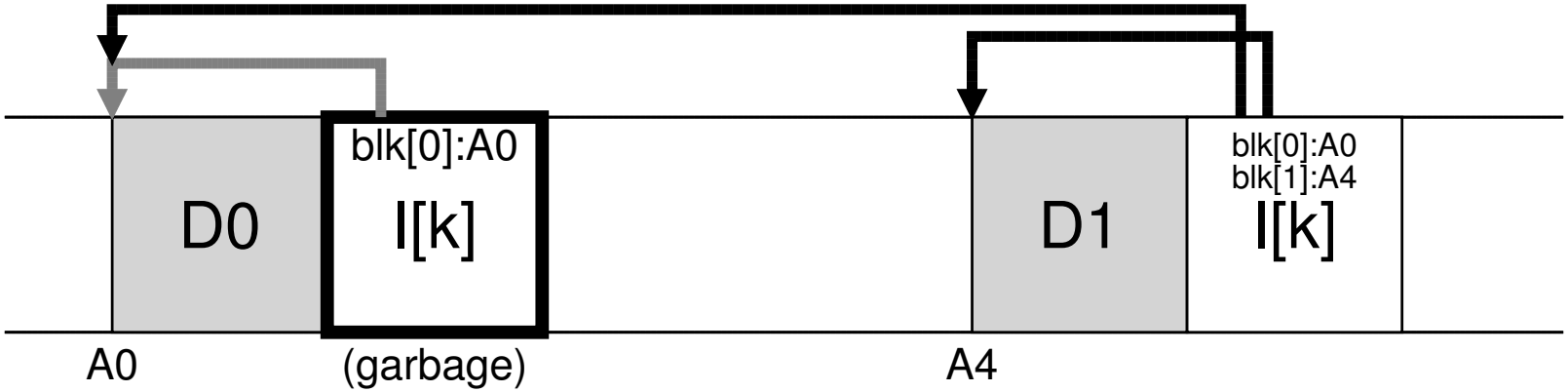
- Creating a file in a folder and writing to it.
- Note inode location may change, but its number is fixed.
  - So no problem with folder structure.

Garbage!

- A file data block of file is updated twice.



- Or we just write two blocks of a file with time difference.

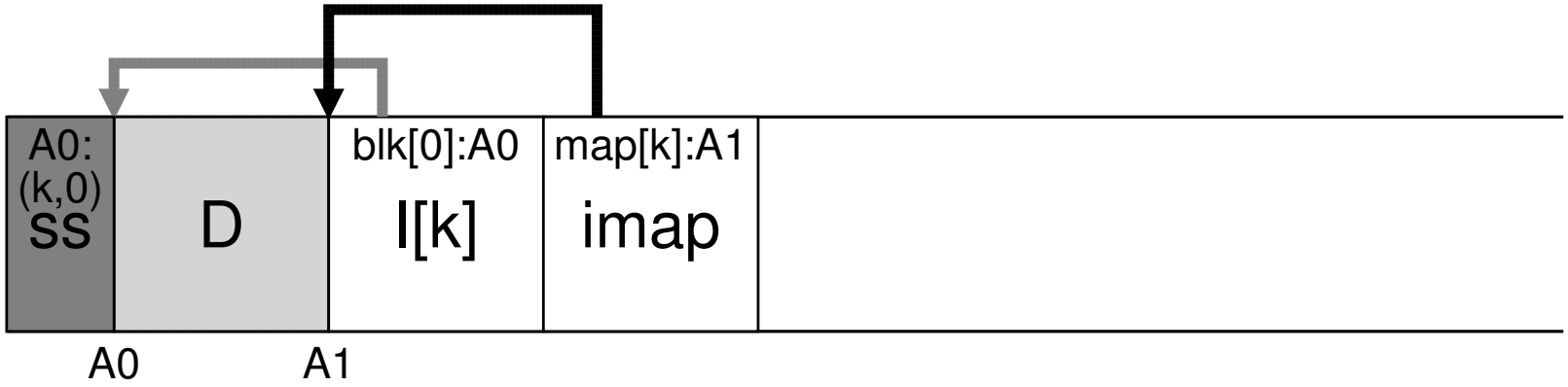


Garbage Collection

- Work periodically.
- Find ‘dead’ blocks.
- Free segments (not blocks).
  - Why? (Recall paging)
- Thus:
  - Read segment.
  - Create new segment with only live blocks.
  - Write new segment (or wait for it to be full)
  - Mark the original segment to be free.

Segment Summary Block.

- For each data block in a segment add:
  - The inode it belongs to.
  - Its offset in the file (i.e., block number)
- Easy to check if a block is aive.



Liveliness

Easy:

```
inode = readInode(imap(N));
if (inode(O) == A)
    A is alive
else
  A is garbage
```

Policies

- When to clean: Disk is full, periodically, idle time
- What to clean: Heuristics
  - Clear cold blocks, delay clearning of hot blocks

Log structure

- Segment are linked list on disk.
- CR contains pointers to head and tail
- Crash?

Crash Recovery

CR atomicity

- Two CRs on both ends of the disk.
- Both with two timestamps. At the head and the commit.
- On recovery only CRs with two identical timstamps are considered.

Segment:

- Roll forward.