CS 423
Operating System Design: Adv Storage 2

Ram Alagappan

Acks: Prof. Tianyin Xu and
Prof. Shivaram Venkataraman (Wisconsin) for the slides.

Logistics and Recap

Final exam details

Recap:

Solutions for crash consistency: FSCK and journaling

Today's lecture: LFS and NFS

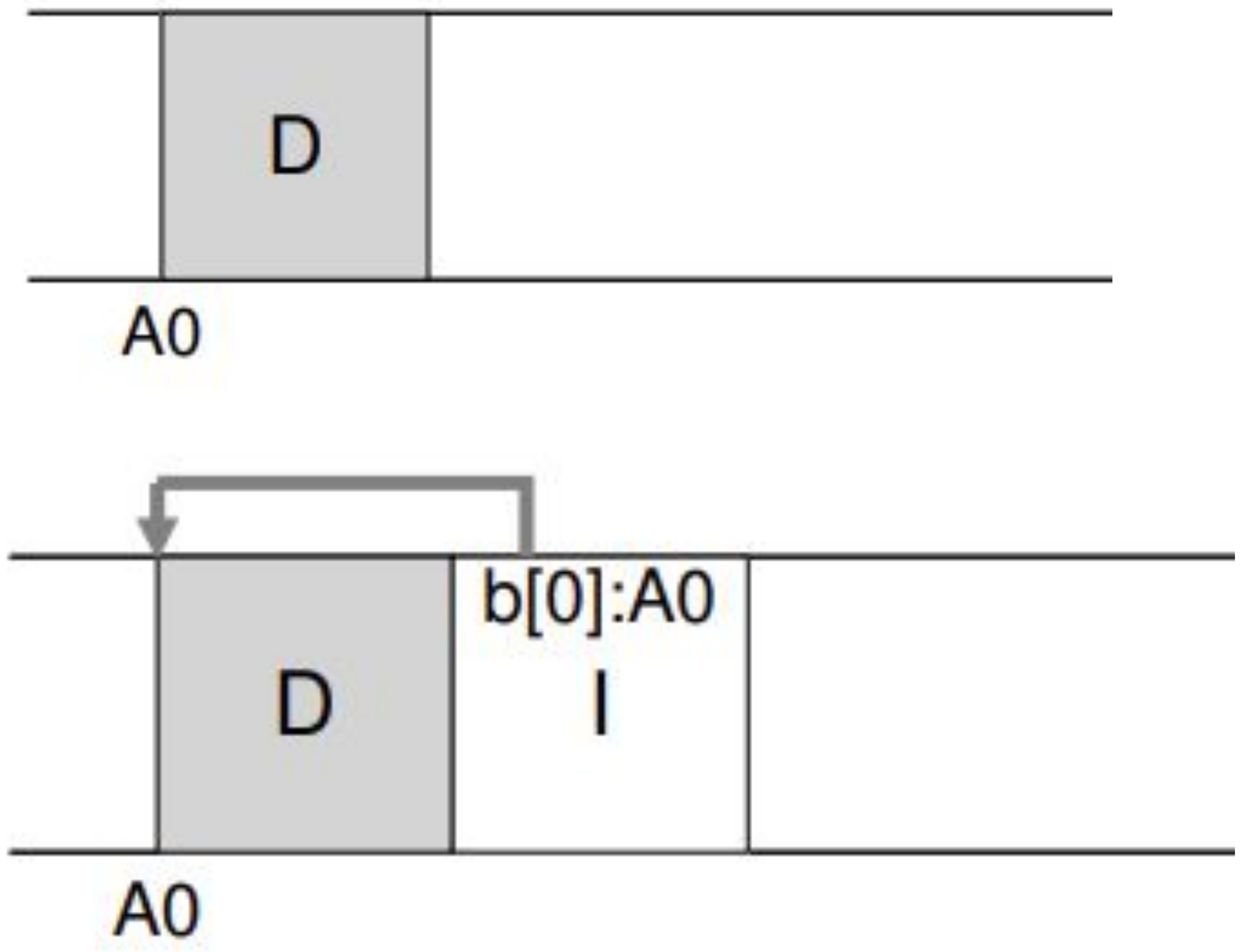
Log Structured FS

Motivation:

1. Sequential speed is much higher than random – all writes must be sequential ideally
2. Memory sizes are growing – write performance matters the most, can also buffer more

Goal

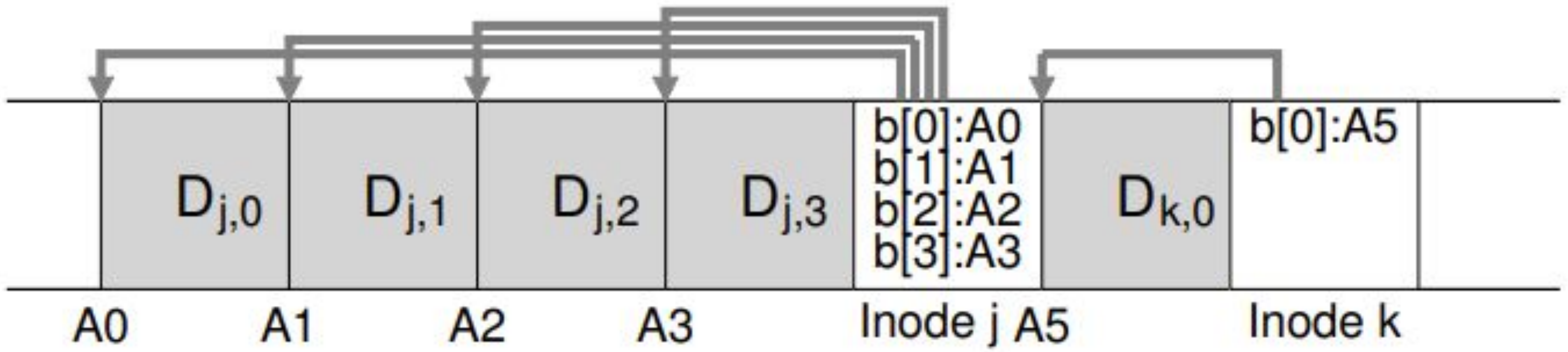
Make *all* writes sequential



Buffering and Segments

Buffer and write in large chunks

Called a segment



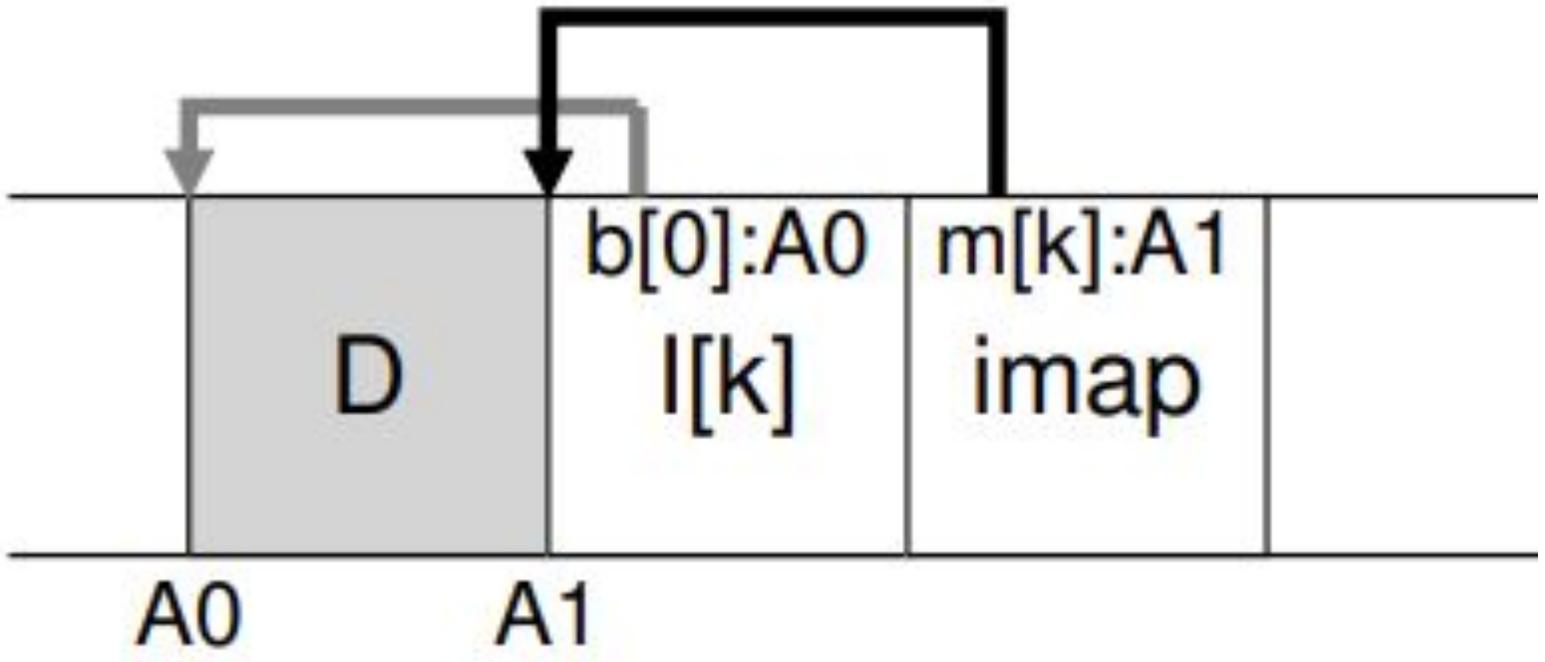
Finding an Inode

How VSFS (or most UNIX FS) does this?

LFS: what’s the problem?

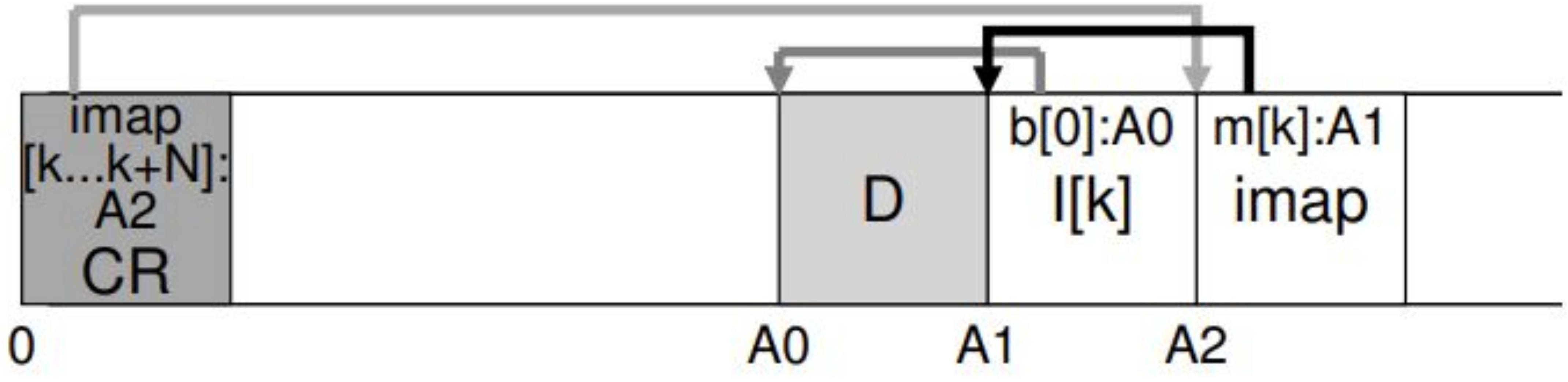
Finding an Inode

LFS uses a new structure called imap
Imap: take inode as input give disk address
Where should imap be?
Memory? Fixed location on disk?



How to Find IMAP!?

If in the log, how to find it?



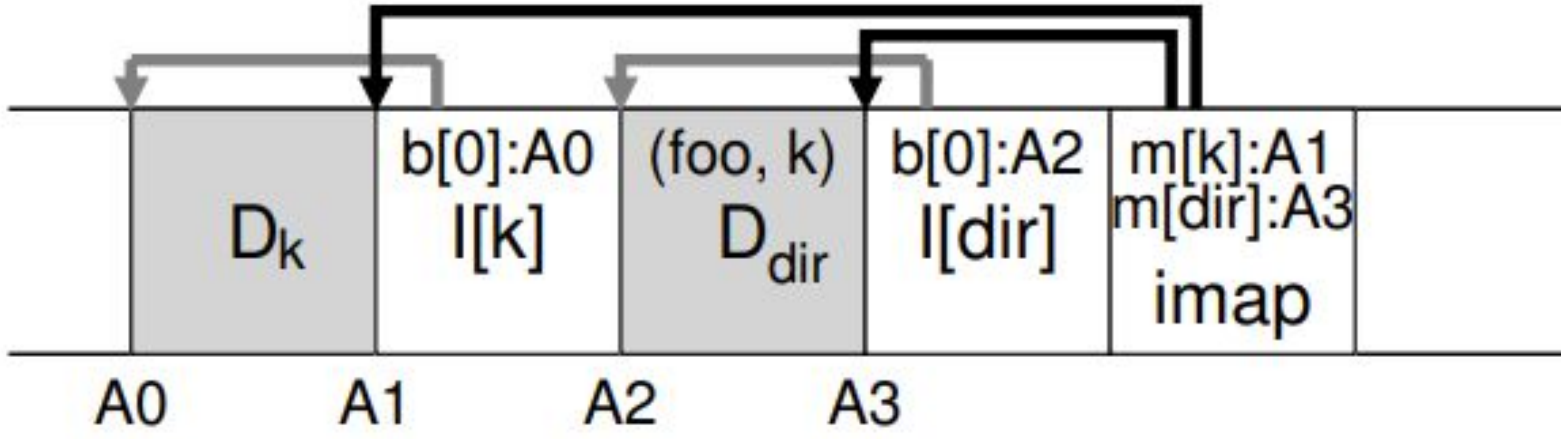
Reading a File

Assume nothing in memory

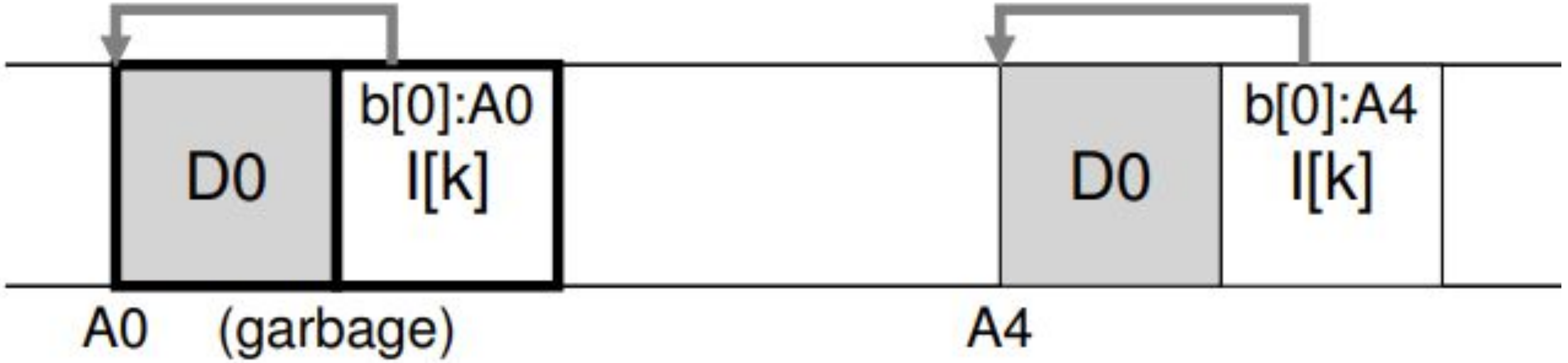
What are the steps?

Directories

Creating a file foo in a directory and appending a block to it



Garbage Collection



The problem?

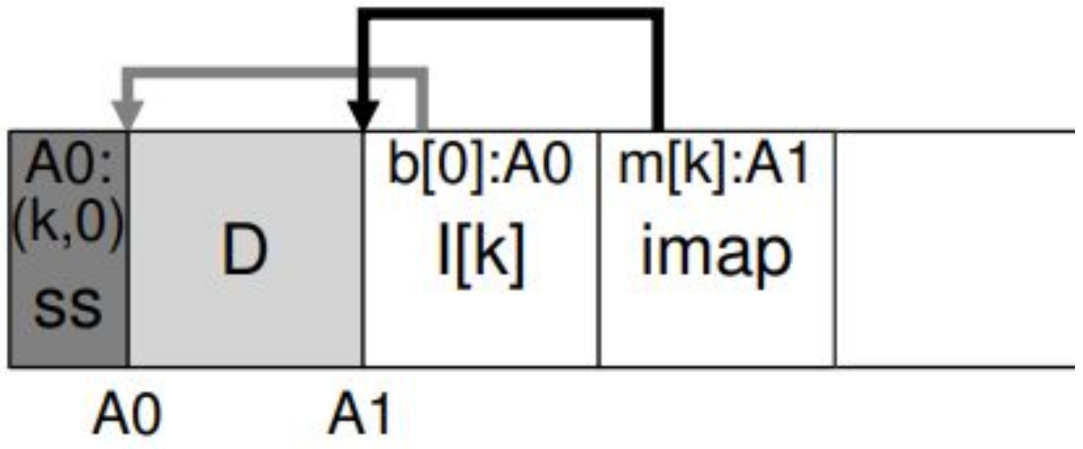
The opportunity?

Cleaning

See which blocks are live within a segment

Write live blocks into a new segment, reuse the old segment

Segment summary block (for every data block, store its inode number and block offset)



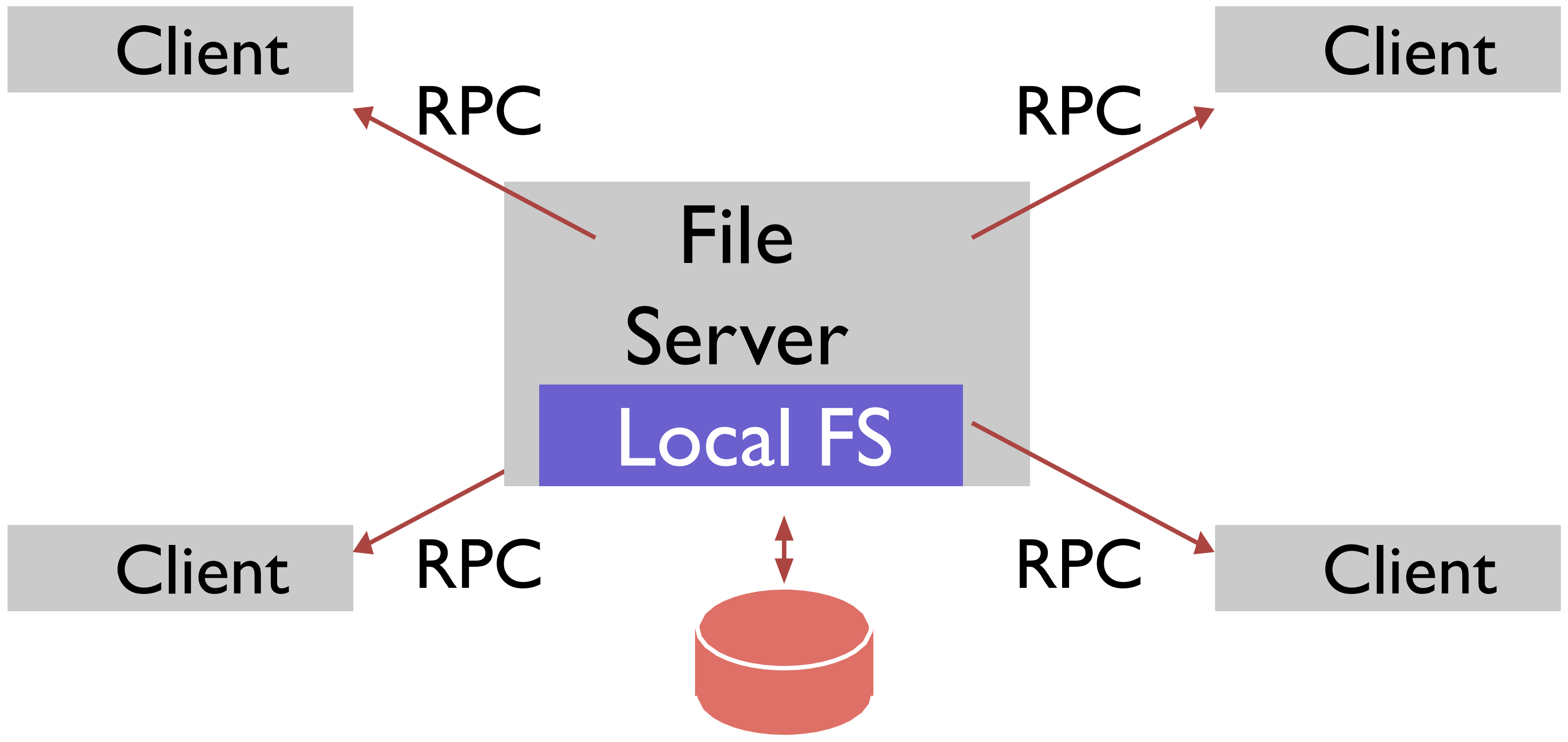
```
(N, T) = SegmentSummary[A];
inode  = Read(imap[N]);
if (inode[T] == A)
    // block D is alive
else
    // block D is garbage
```

NFS Distributed File System

NFS: more of a protocol than a particular file system

Many companies have implemented NFS: Oracle/Sun, NetApp, EMC, IBM

NFS Arch


Client
RPC
RPC
Client
File
Server
Local FS
Client
RPC
RPC
Client

Benefits?

Sharing across machines

Central admin

Goal: Simple Server Crash Recovery

Why do servers crash?

Goal: Simple Crash Recovery

Strategy-1: server returns fd upon open, client passes fd on each call

```
int fd= open(“foo”, O_RDONLY);
read(fd,  buf,  MAX);
read(fd,  buf,  MAX);
…
read(fd,  buf,  MAX);
```

Goal: Simple Crash Recovery

Strategy-1: server returns fd upon open, client passes fd on each call

```
int fd= open(“foo”, O_RDONLY);
read(fd, buf, MAX);
read(fd, buf, MAX);
...
read(fd, buf, MAX);
```

$\leftarrow$ Server crash!

Problems

Complicates crash recovery. Why?

Server crash – what happens? What must client do?

General idea: Statelessness

Server keeps no state: no fd □ file map, no file position pointer

Server doesn't keep any state about a client

Client passes all info needed in each call to server

Advantage:

- no special crash recovery - the server just starts running again
- client might have to retry a request

Pass all info – option-1

Stateless protocol: server maintains no state about clients

Need API change. One possibility:

read(char *path, buf, size, offset)

Specify path and offset each time

Pros? Cons?

Pass all info – option-2

Stateless protocol: server maintains no state about clients

Use file handles

```
fh  = open(char *path);
pread(fh, buf,size, offset);
pwrite(fh,buf,size, offset);
```

File Handle = <volume ID, inode #, generation#>

Opaque to client, purpose of generation#? when incremented?

Some NFS calls

Lookup – notice no open (open == series of lookups)

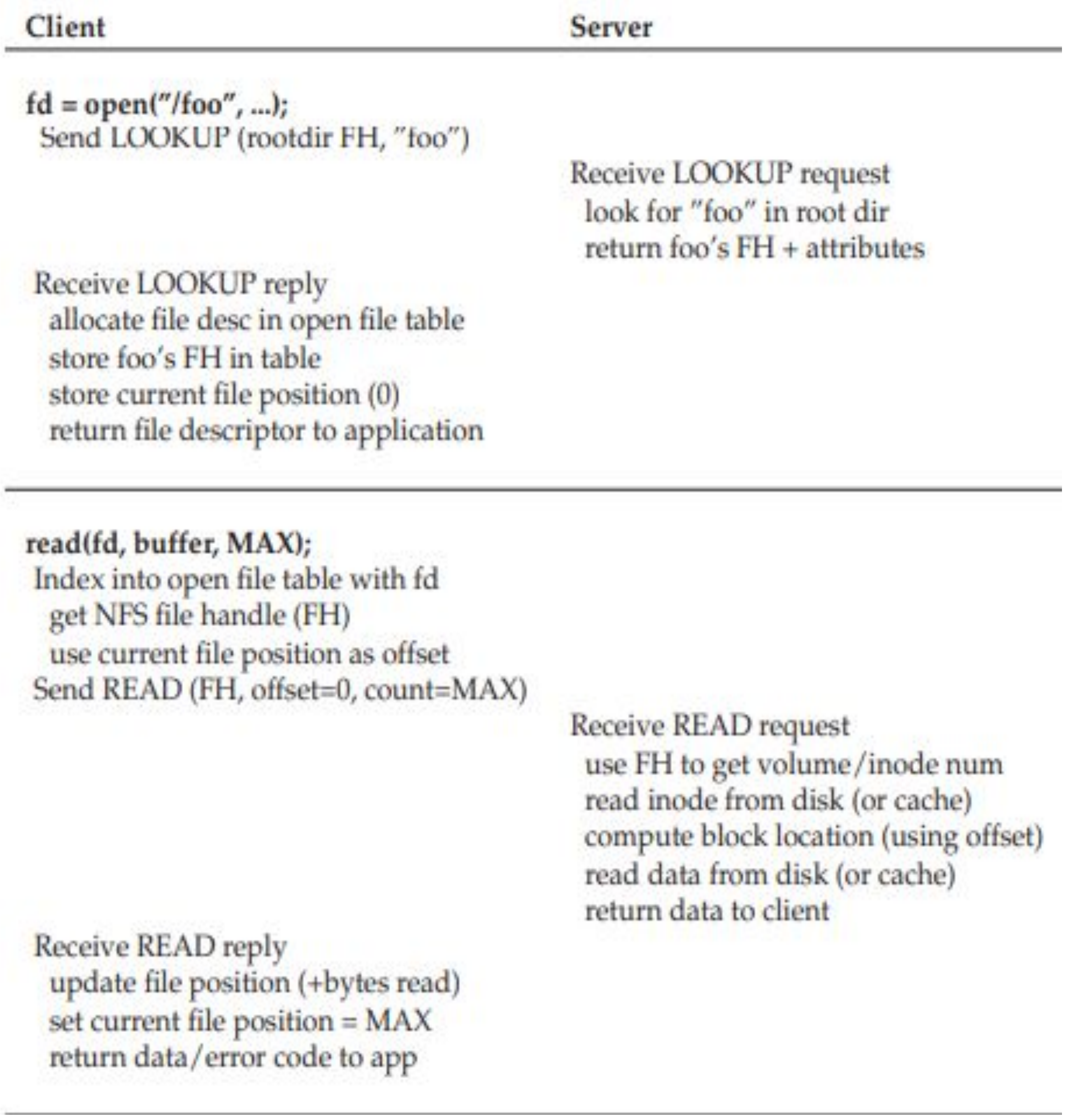
GetAttr

Read

Write

Who keeps the fd to fh mapping?

Reading a File on NFS



Client	Server
fd = open("/foo", ...); Send LOOKUP (rootdir FH, "foo")	
	Receive LOOKUP request look for "foo" in root dir return foo's FH + attributes
Receive LOOKUP reply allocate file desc in open file table store foo's FH in table store current file position (0) return file descriptor to application	
read(fd, buffer, MAX); Index into open file table with fd get NFS file handle (FH) use current file position as offset Send READ (FH, offset=0, count=MAX)	
	Receive READ request use FH to get volume/inode num read inode from disk (or cache) compute block location (using offset) read data from disk (or cache) return data to client
Receive READ reply update file position (+bytes read) set current file position = MAX return data/error code to app	

Close() a file?

What happens?

No server communication

Failures

What do clients do when they don’t get a response?

Request lost

Server down

Reply lost

Simplifying Recovery with Idempotency

All cases are handled uniformly

read

write

mkdir

creat

Client-side caching

Cache data for performance

What are the problems?

P1: update visibility

Scenario: edit a file and move on to a different workstation

Solution: flush-on-close

Drawbacks?

P2: stale cache

Cached content could be old

Solution: getattr

What problems will this introduce?

Write buffering on server

Can server buffer writes?

NetApp bbram…