CS202 (-002) Lecture 18: File System

Last time

- Disks

- The point of a file system

  + Data on disks persist reboots, power cycles, etc.
  + Want a convenient way for users (🧍) to track what this data is.

  + So,

    - Want a way to name collections of blocks, etc.

.

File system

+ ⟨File, Offset⟩ ⟶ Location on disk

  - Offset → Byte offset
  - File → Name "/users/apanda/foo.txt" / Previously opened file descriptor — 22 / ....
  - Location on disk → Block (sector), byte location within block

+ Naming, etc. - Directories...

File, Offset → Disk location

- Goals

  ⓐ Mapping persists ← Stored on disk

  ① M...

(b) Minimize # of disk blocks accessed during disk accesses

(c) Minimize space used to store mapping

Observations to help with meeting goals

(a) Often - all blocks in a file are accessed together
↳ Sequentially

(b) Most files are SMALL (a few disk blocks)

(c) Most of the DISK SPACE is used by large files

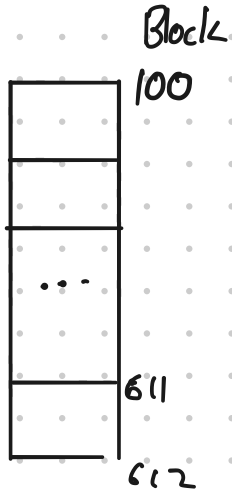
(d) Sequential access to disk faster than random access - Last lecture

Approaches

- Contiguous allocation [Also called extents]

Core Idea: - When creating a file, specify size
- File system reserves sufficient # of blocks
- Mapping - file ⟶ Start block, Length

bop.mp4 ⟶ start: 100
(2mb) len: 512



Pros: - Good for sequential access

- Simple, small metadata

Cons/Problem: - How to grow a file?

bop.mp4 ⟶ <100, 512>

blah.txt ⟶ <612, 1> ← How to add more than a block?

boom.mp4 ⟶ <613, 100>

Aside: Extents core in active use today

- APFS
- Ext 4

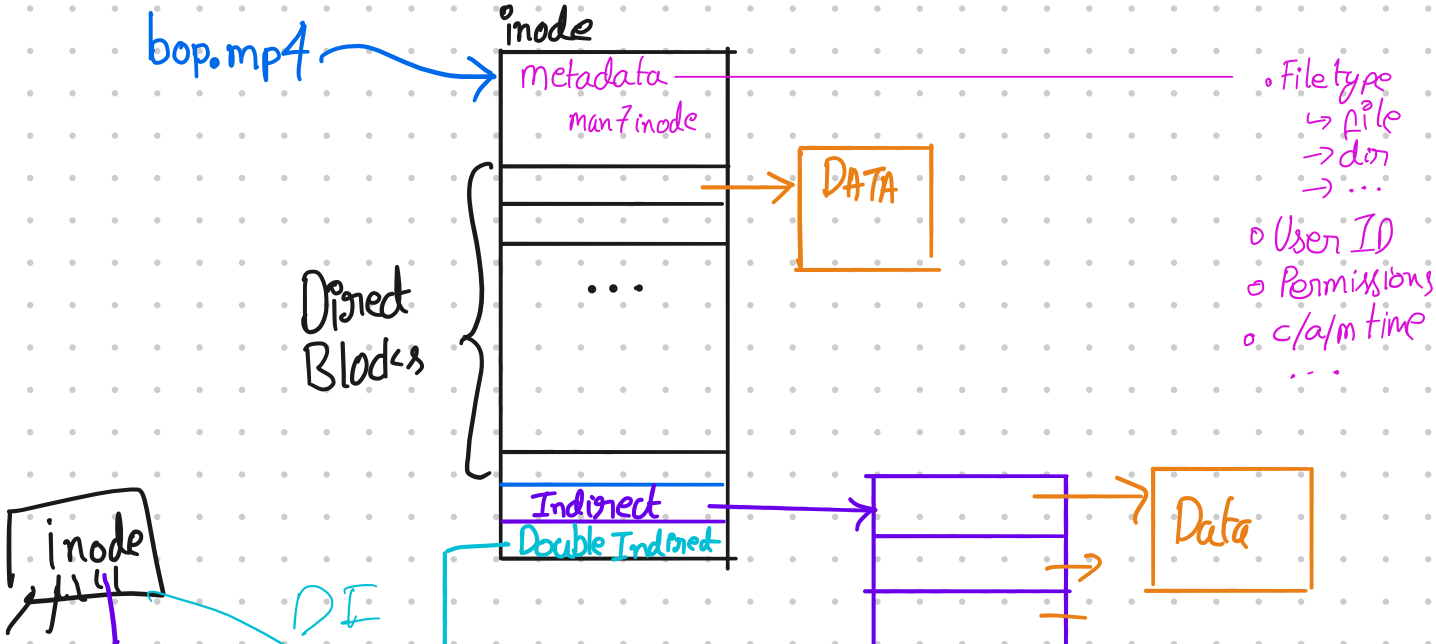
} - Usually in combination with "indexed files" (coming up)

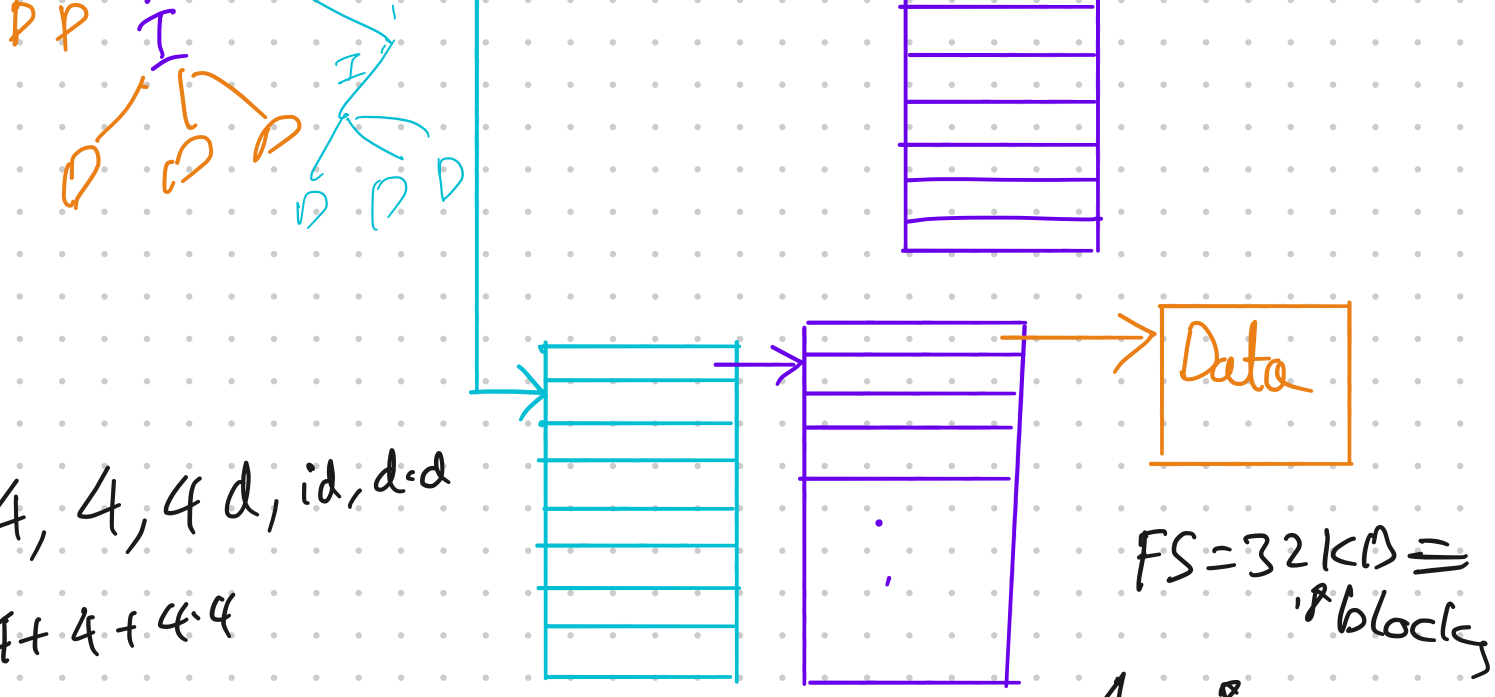
- Usually so that a file can have more than one extent

- Indexed Files

- We sort of talked about this last class

· INode ≡ Index Node





4, 4, 4 d, id, d·d

4+4+4·4

FS = 32KB ≡ 8 blocks

1 + 8

Why this design?

Indirect Block — 128 ent

Direct Block — 128 ent - # of extra blocks accessed for small files?

File: 8 block = 32KB

Seq read

How many blocks read

- Max file size?

- # of extra accesses for large files?

132 blocks

258 block.

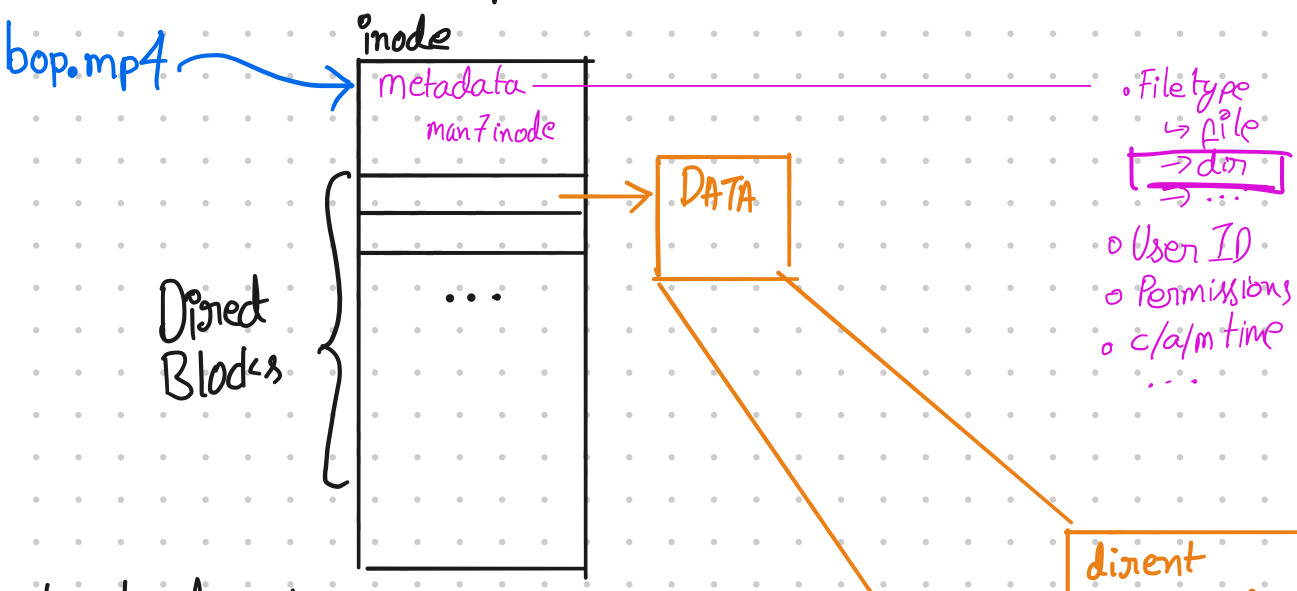
- Beyond double indirect?

So far - all about a file, nothing about

- Naming files [Remember, inode metadata does not contain name]
- Finding files.

Directory: A special file that is a list of other

files



struct dirent

dirent.h(0P) POSIX Programmer's Manual dirent.h(0P)

PROLOG
This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME
dirent.h - format of directory entries

SYNOPSIS
#include <dirent.h>

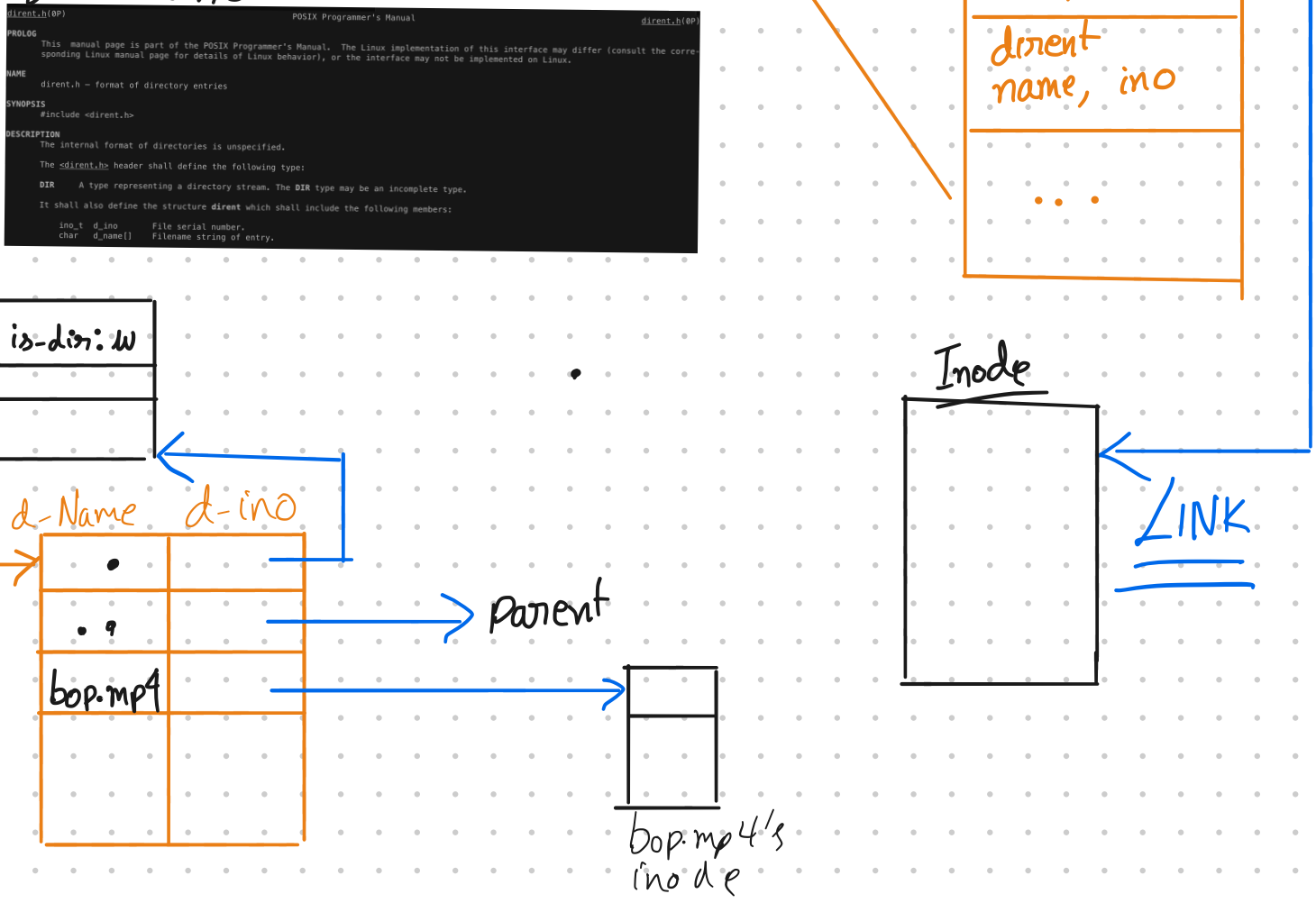
DESCRIPTION
The internal format of directories is unspecified.

The <dirent.h> header shall define the following type:

DIR A type representing a directory stream. The DIR type may be an incomplete type.

It shall also define the structure **dirent** which shall include the following members:

```
ino_t  d_ino      File serial number.
char   d_name[]   Filename string of entry.
```



Tells us how to name, not how to "find"

Many ideas

① One directory

→ List of all files

→ Look through list to find

Issues
- → Name collision
- → Directory size
- → ...

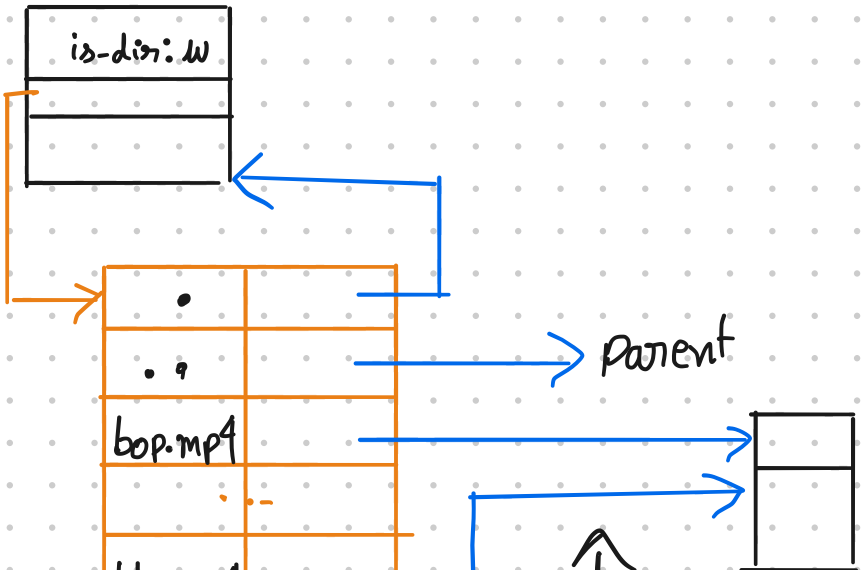
(1) [Multi-User] One directory per user
- ↳ No name collision between users
- → Per user isolation

Issues

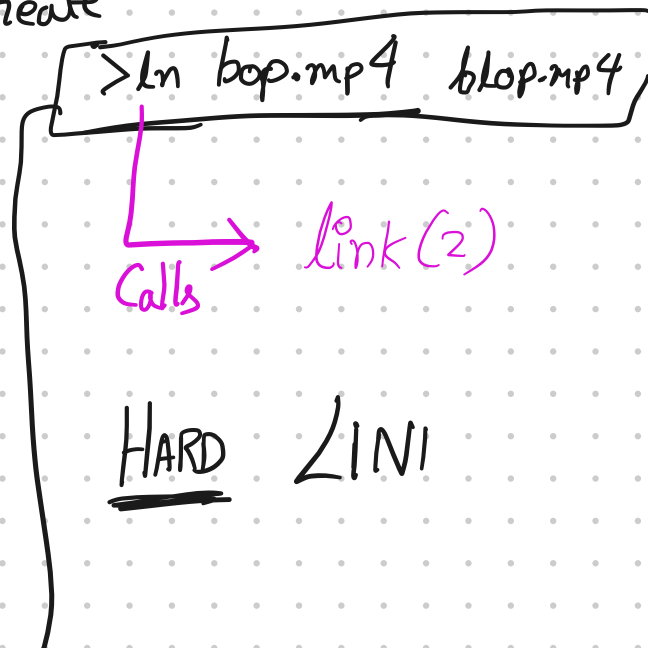
(2) What you are used to — deeply nested trees

Most systems prescribe a structure → later

Links



Can create

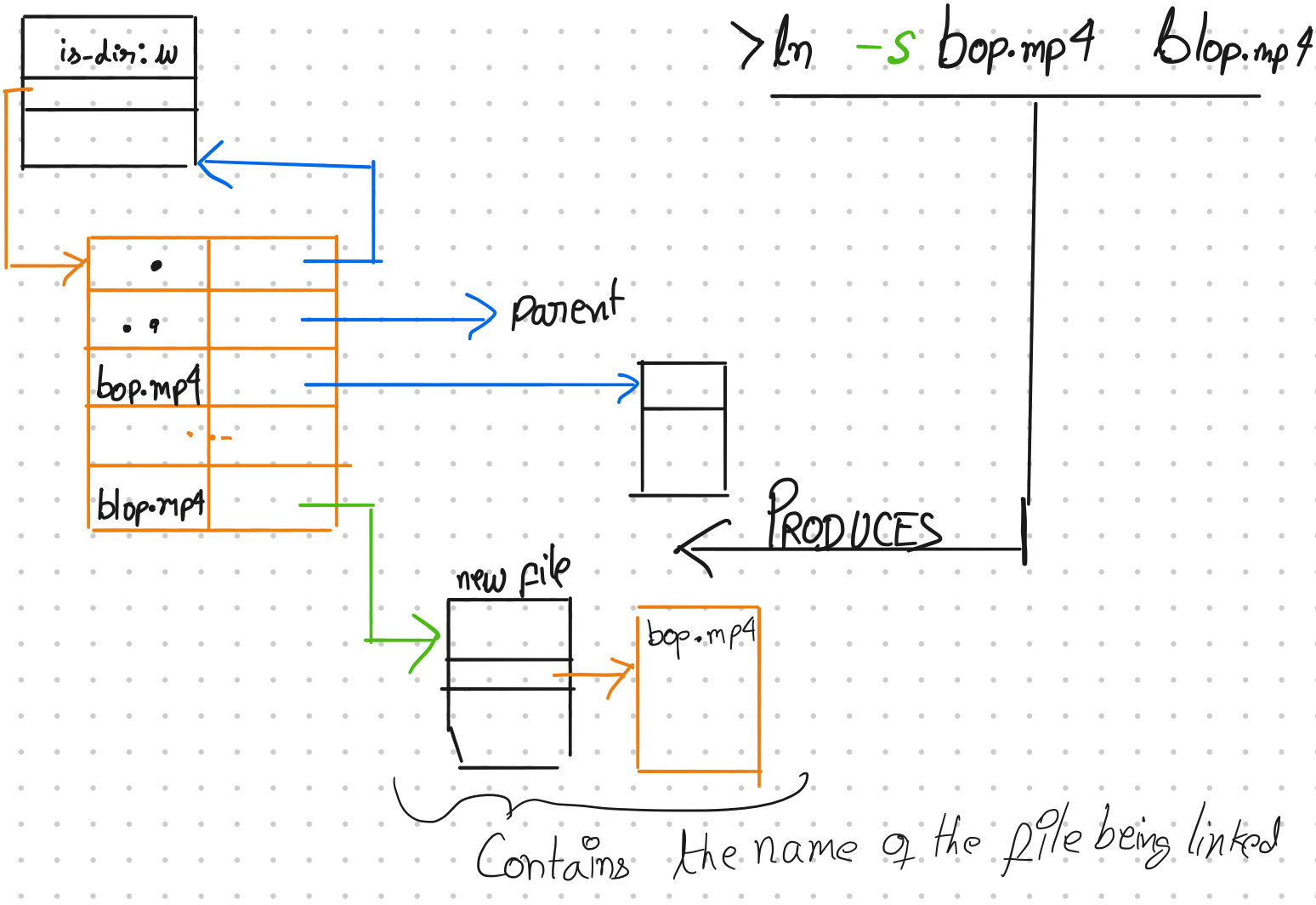


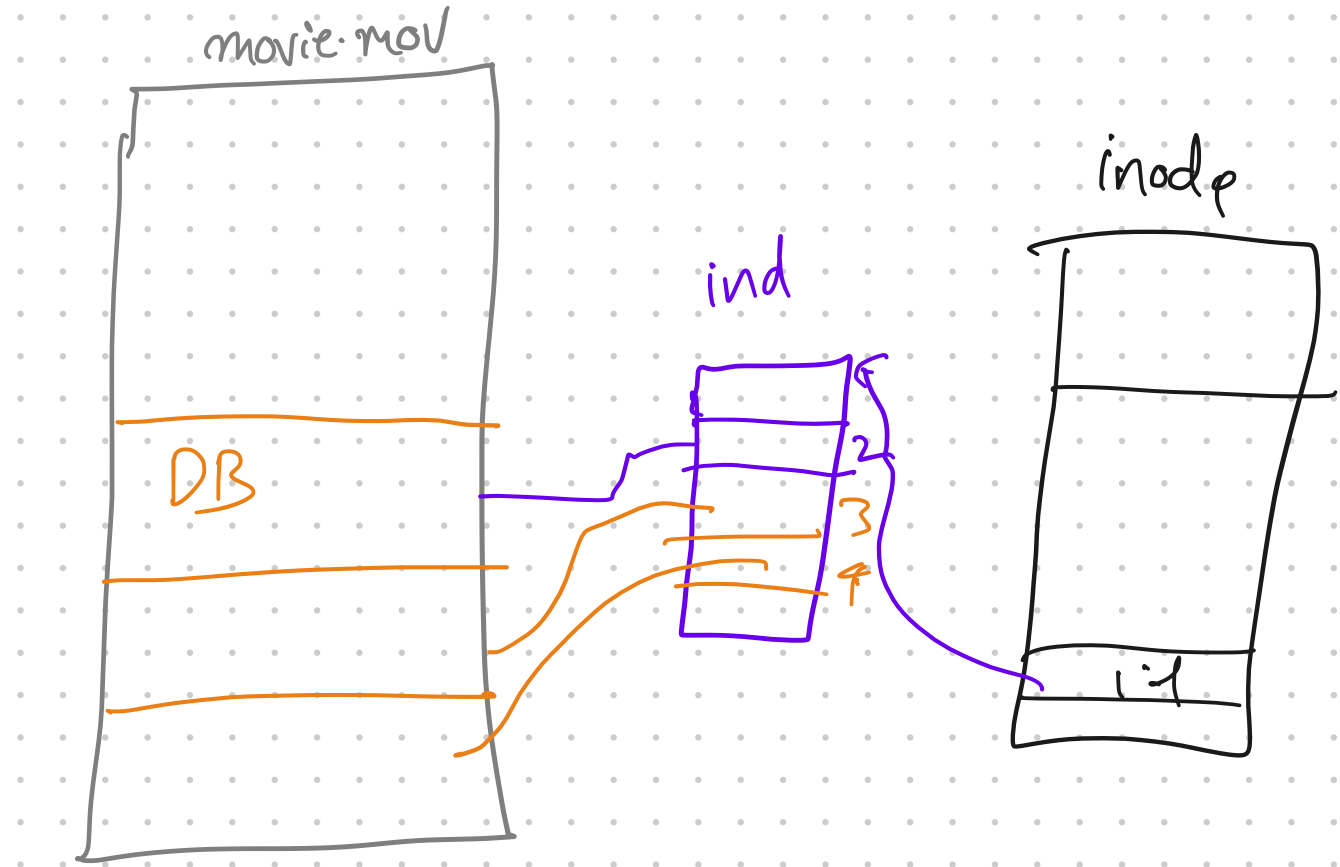
HARD LINK

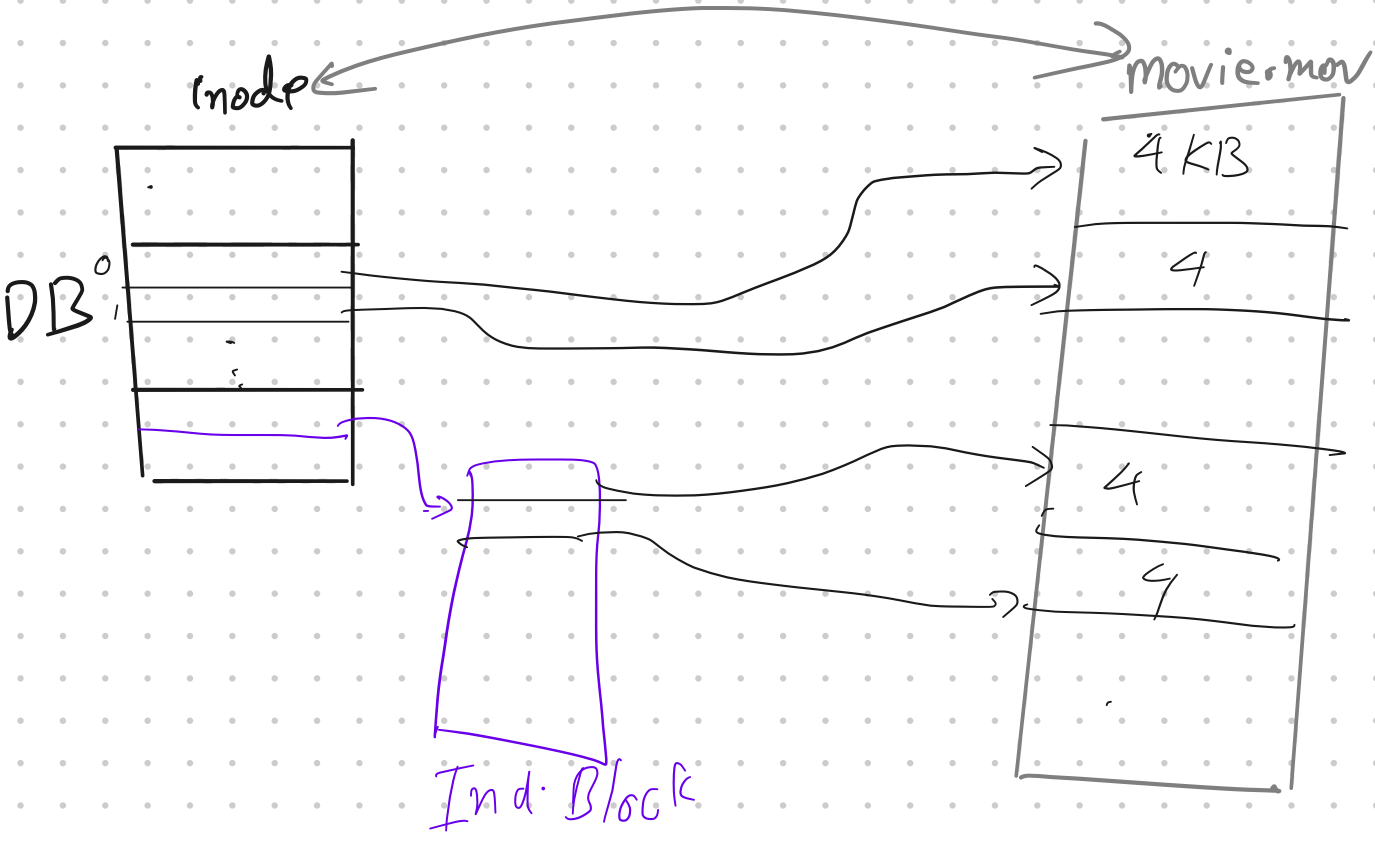
blop.mp4

PRODUCES

Alternative: Symbolic link




movie.mov
DB
ind
2
3
4
inode
i-1


inode
movie.mov
DB
0
1
4 KB
4
4
4
Ind. Block