Welcome Back!

Abstraction

File System
Design + Implementation
S/W

Abstraction

read/write array of sectors
interface

Where we were:

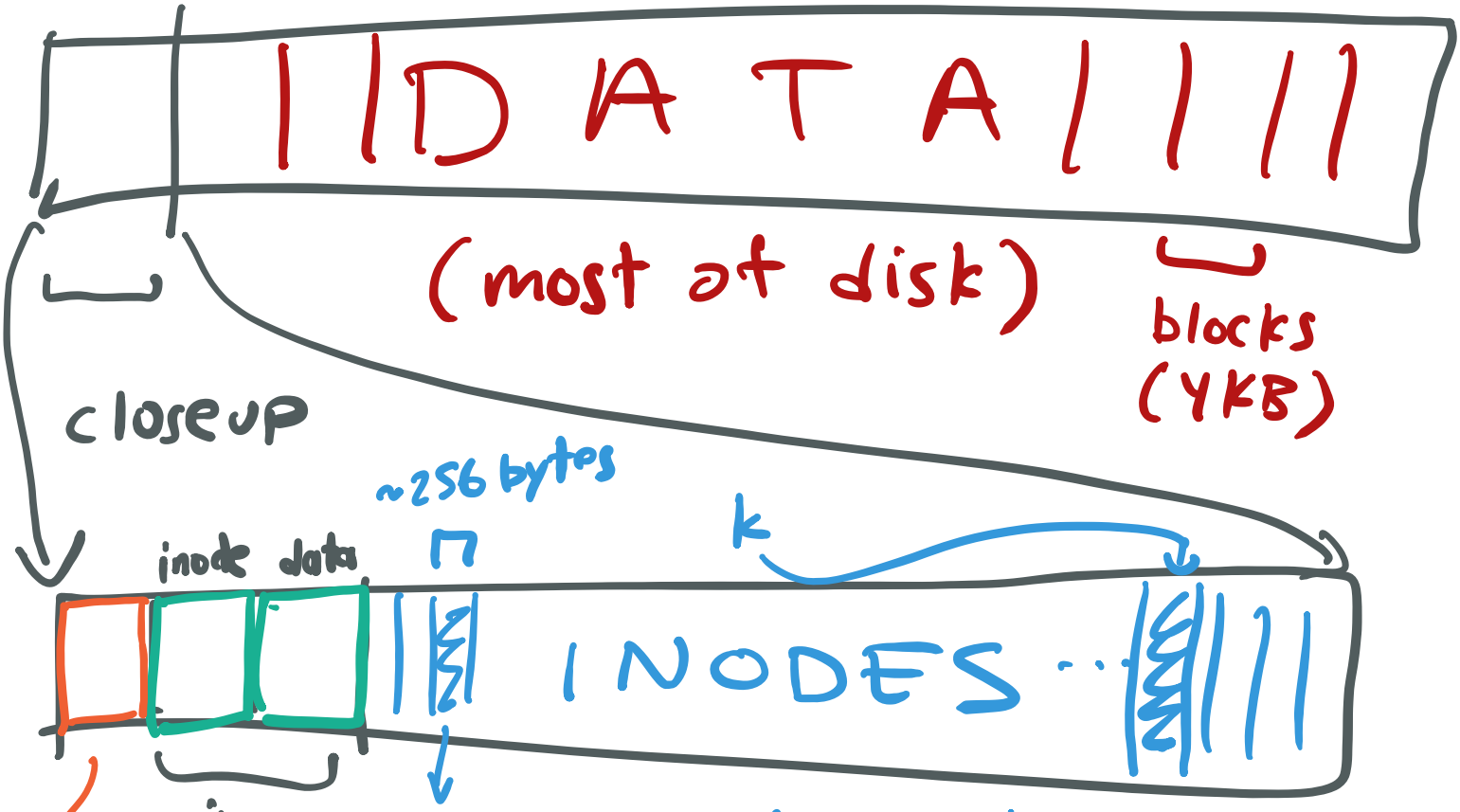
→ FS API open, read/write/close, link, unlink, mkdir, rmdir, etc.

→ Implementation

simple implementation:

→ On-disk data structures

{ → Access methods)



Questions:

Access?

Large Files?

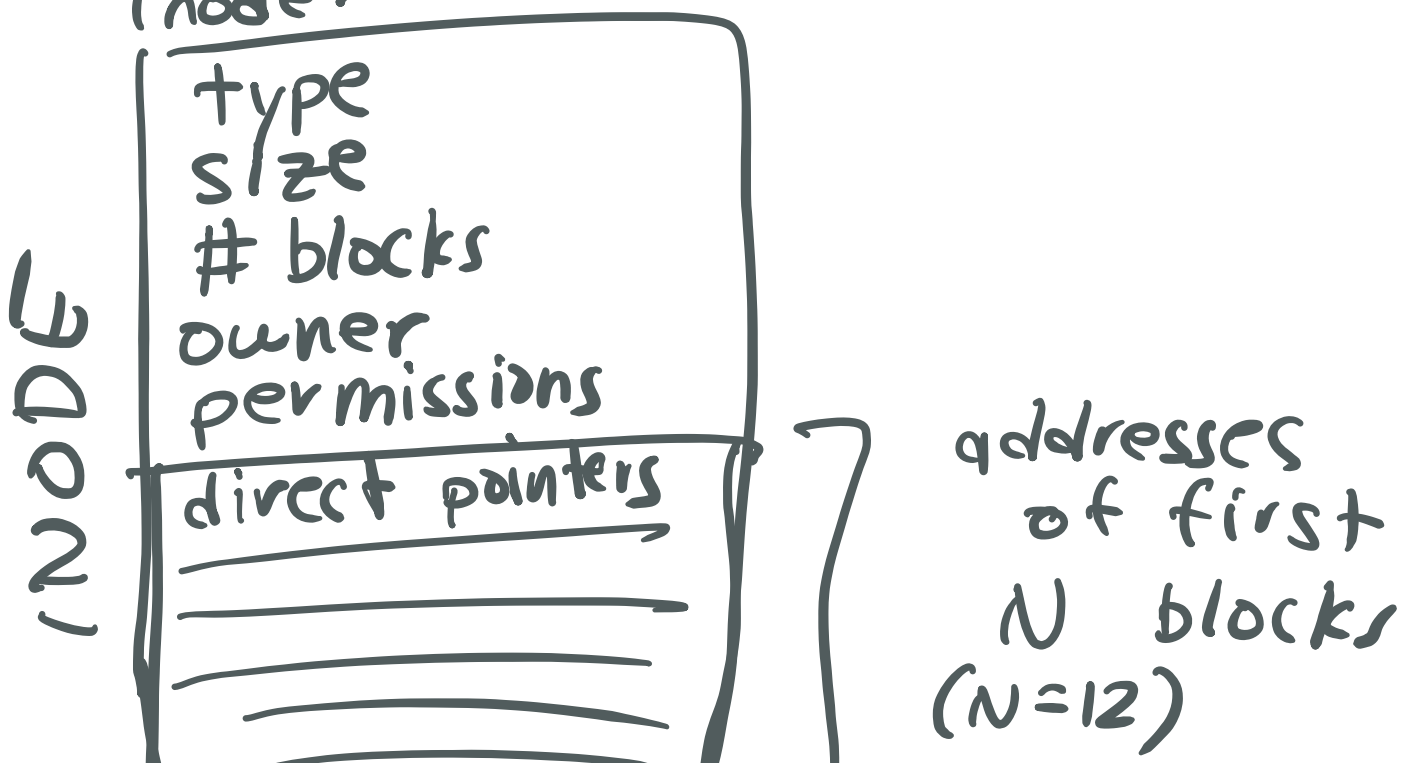
Performance: Use Memory (DRAM) as a cache

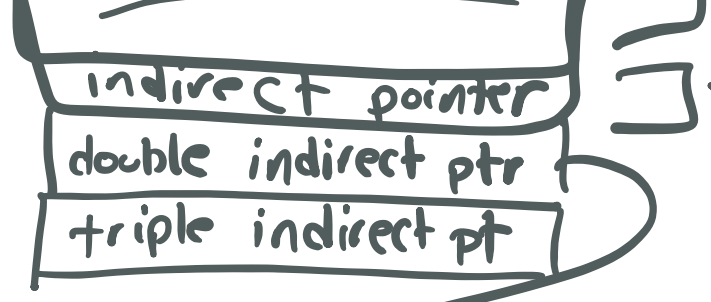
pathname: /a/b/c/d/main.c

e.g. (a, 10)

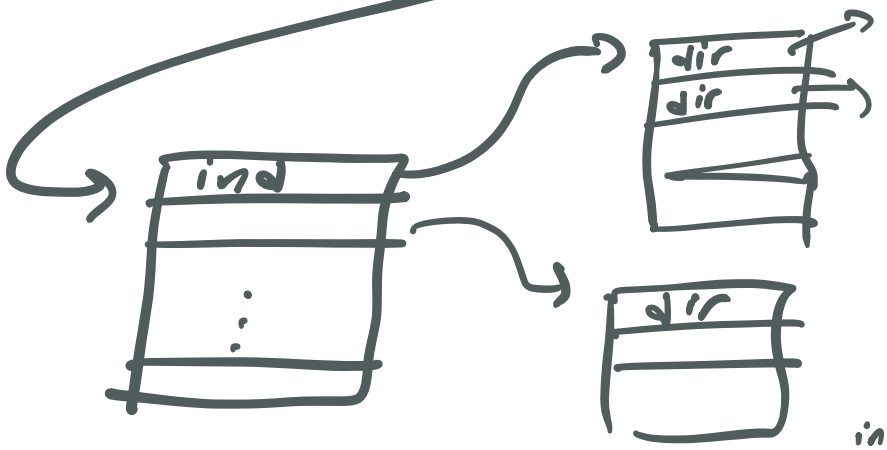
root inode
root data → a's inode #
a's inode
a's data → b's inode #
b's inode
b's data
c's inode
c's data
d's inode
d's data → main.c inode
main.c inode

Large Files:

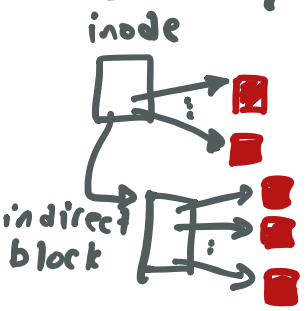




→ data block (direct pointers)



~1024 (4KB / 4 bytes)



=) Some file systems use extents: [ptr, length]

[not just pointers]

=) [Last Project] coming soon

=) Caching / Write Buffering

Caching → frequently accessed → in memory

→ delayed write benefits
+ batch
+ latency
+ avoid I/O (overwrite, delete)

write() → fast → buffers in memory + returns (not immediately written to disk)

(use LRU-like replacement)

⇒ crash ⇒ (lost data)

to ensure no data loss:

→ write()

→ fsync() (slow)

forces writes to disk

Locality: case study of early file system

(Berkeley Fast FileSystem) [FFS]

FFS: main idea

"treat disk like a disk"

(technology aware)

⇒ simple ideas

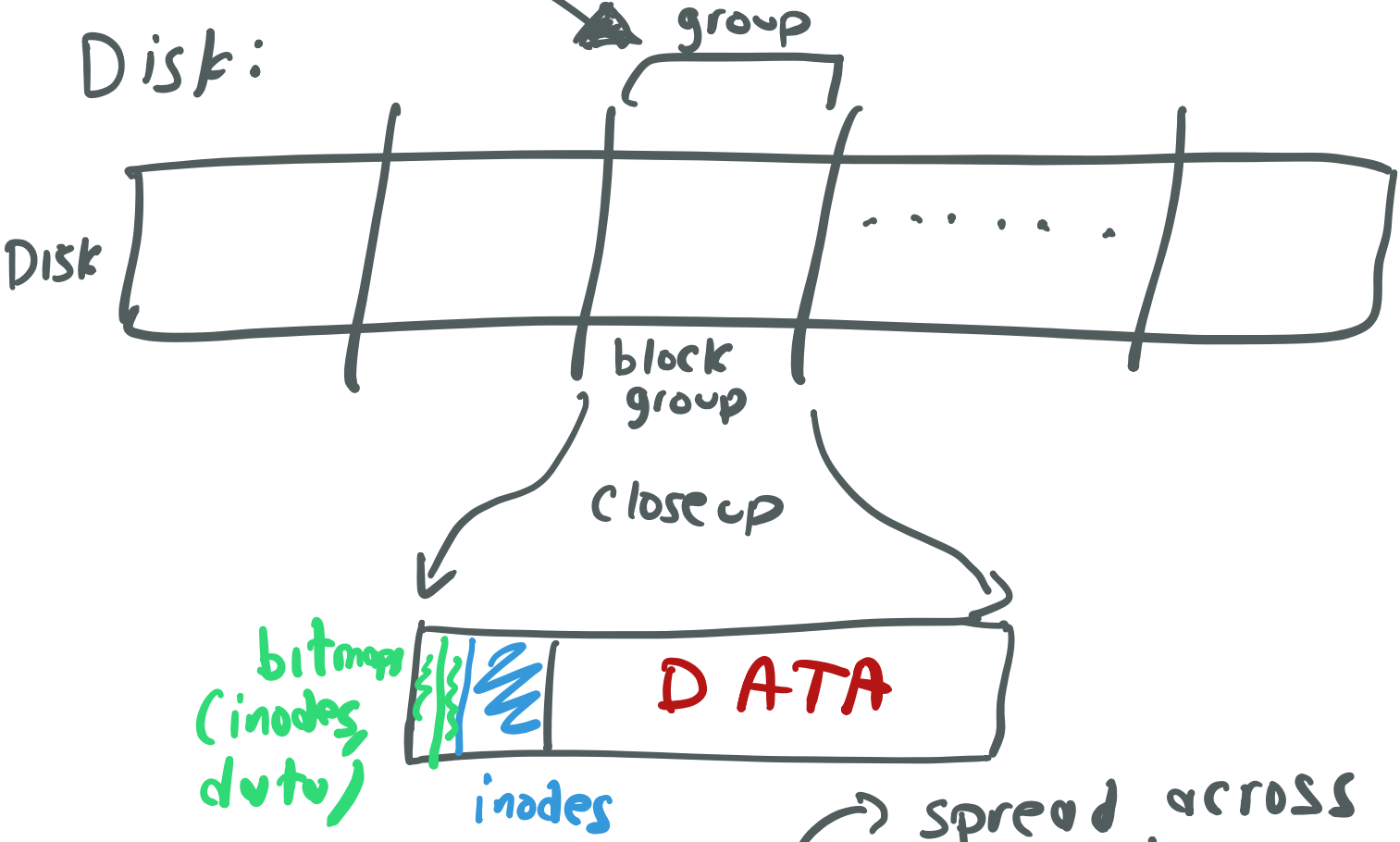
"block group":

→ contiguous part of disk

→ allocate "related" items in a group

(spread out "unrelated" items into different groups)

block



Allocation Policies:

- → mkdir() Directory → spread across disk (e.g. pick group w/ low # of directories)
- → creat() File → put files in same group as parent directory

Large File exception: