Today (5/1)

Almost Done!

Happy, or Sad?

Correct Answer: Both

Now: It's warm ...

Today: Virt

Conc

Persistence

→ LFS (log-structured file system)

→ Not Hard drives: Flash-based SSDs

Admin → (logging)

→ End: Words of wisdom ?

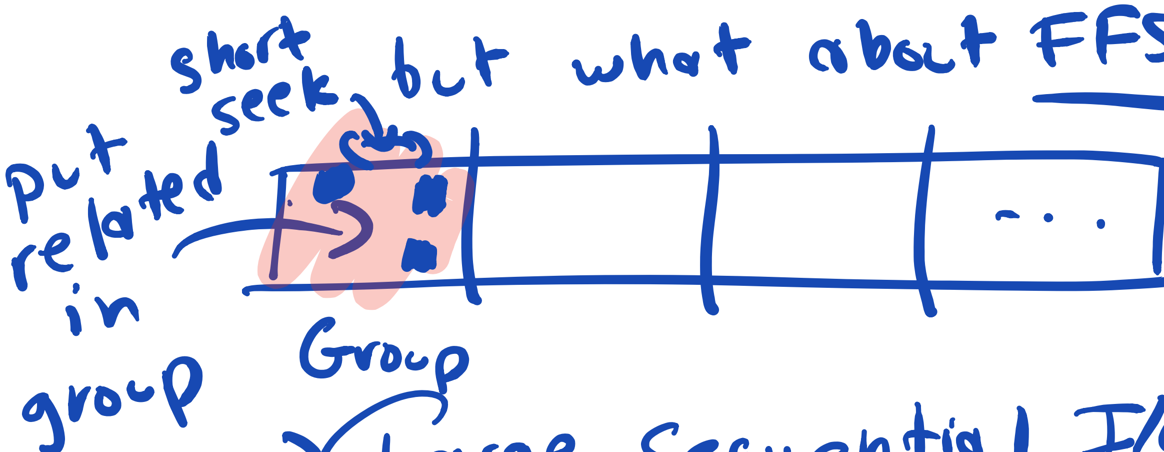
Log-Structured File Systems (LFS)

Motivation:

Hard Drive Performance

⇒ (small) Random I/Os : Poorly (seeks, rotations)

but what about FFS?

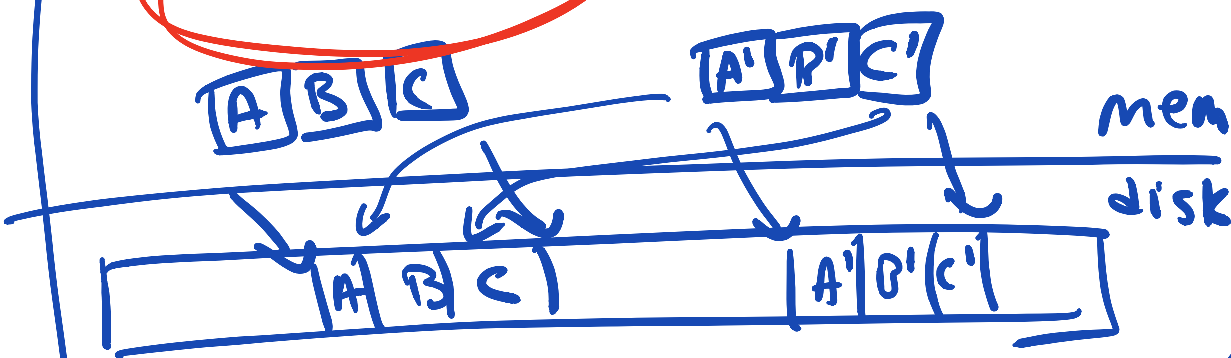


⇒ Large, Sequential I/Os: perform well

Problem:

Reads: hard to ensure large, contiguous

Writes: we have choice:



⇒ insight: can write data anywhere

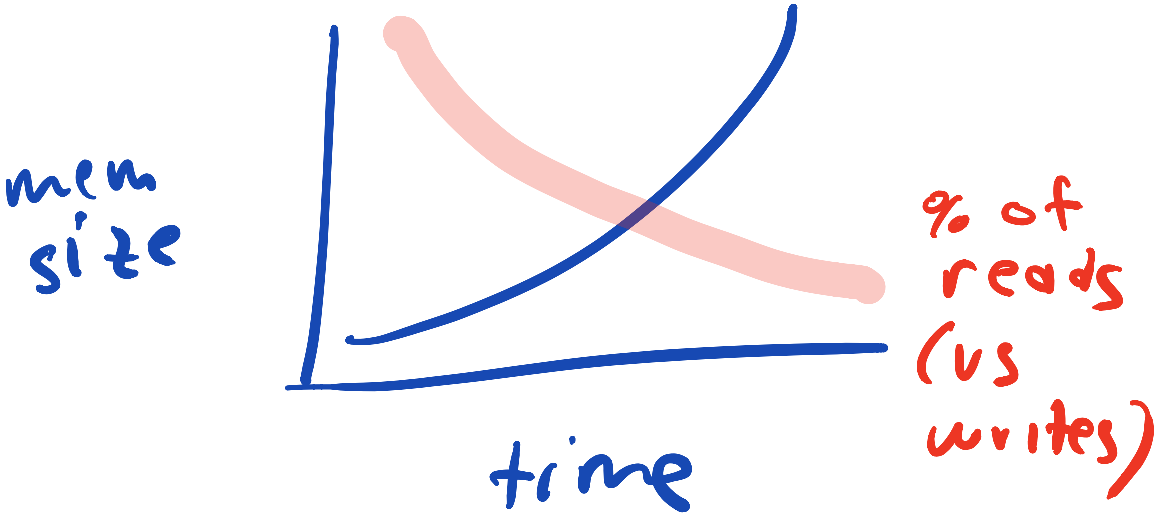
Read Problem: Memory has gotten larger

⇒ 1990 Main memory [1 MB]

⇒ now many GB:

Cache: main memory is cache for

file data



LFS: Structure of LFS

=) Data Structures

=) mostly, keep "same"
(inodes, data blocks, directories)

+ few new structures

=) Access Methods

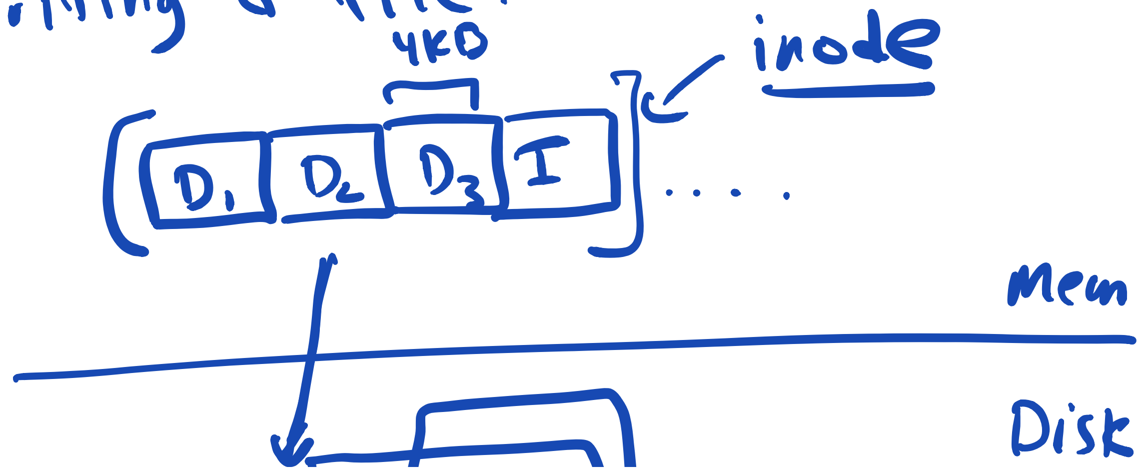
⇒ writing, reading
what's <u>different?</u>

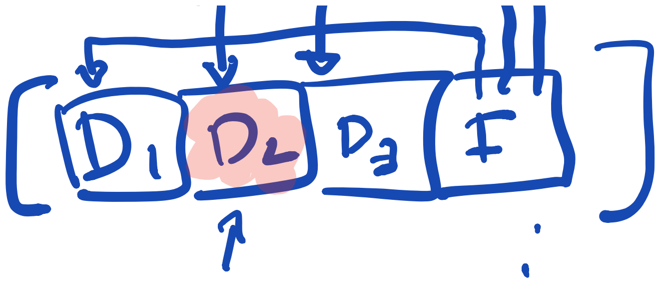
Basic idea: focus on <u>writes</u>

1) buffer many writes in memory
(⇒ in-memory "<u>segment</u>")
<u>~ a few MB in size</u>

2) write to disk
key: what to write to enable later reads?

<u>Example</u>
writing a file:

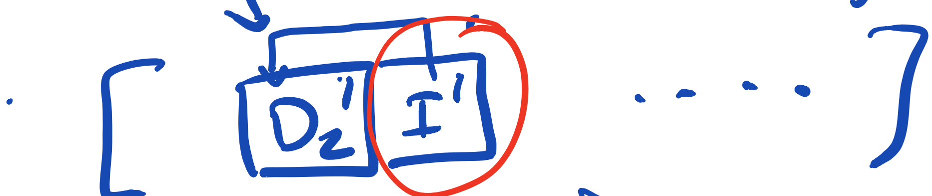




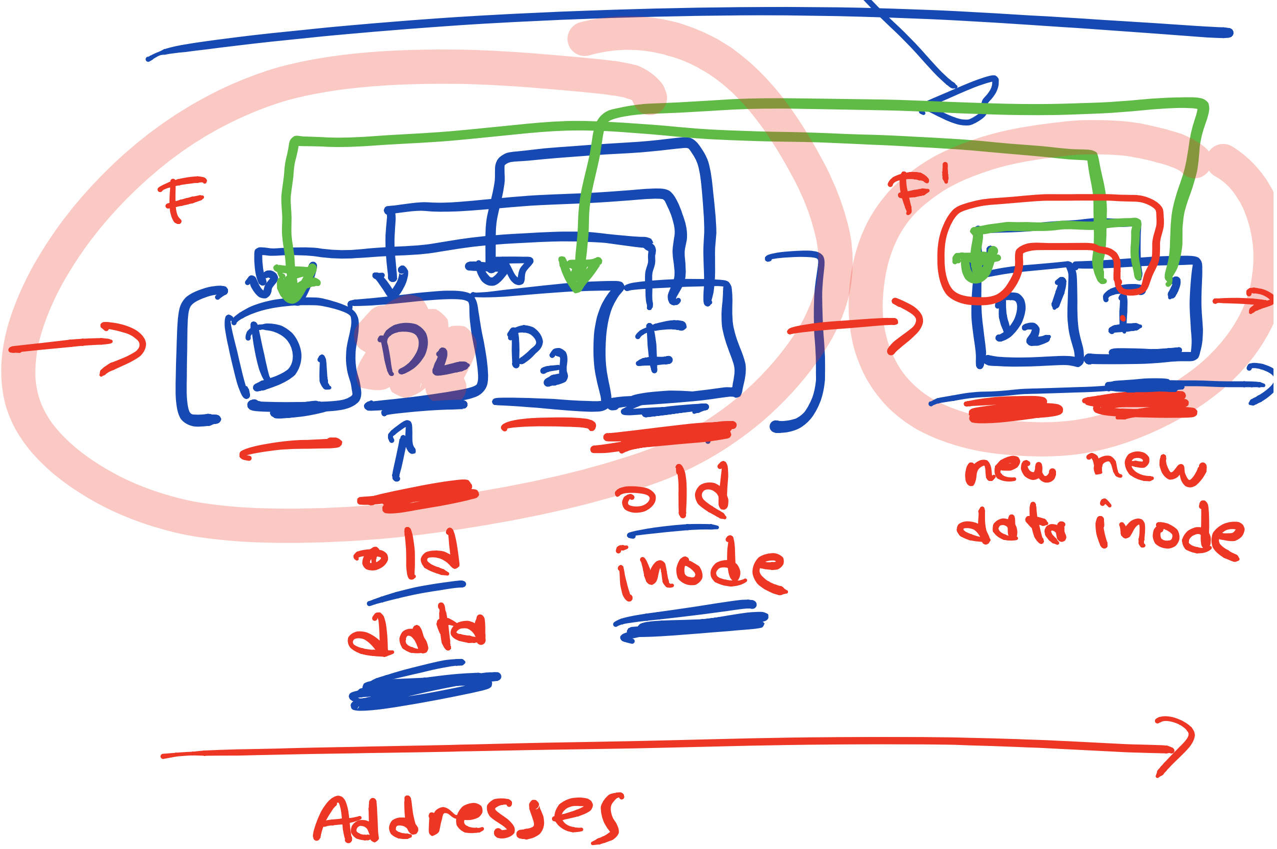
Later: user overwrites D_2

[write (D_2')]

in-memory segment



mem



Problems:

⇒ how to find latest version of inode? (I' not I)

⇒ garbage: old versions of inodes, data lying around: what to do?

how to find inode (latest?)

⇒ in-memory "inode map"

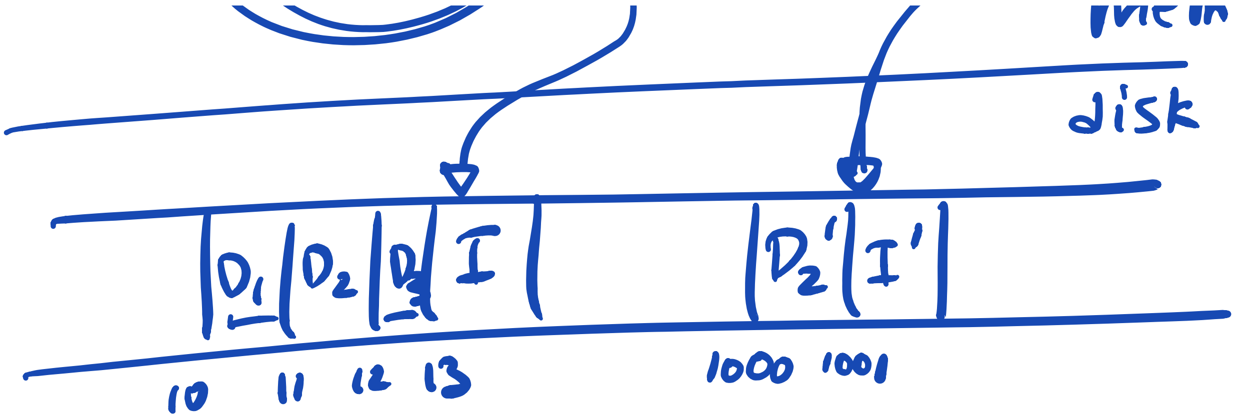
array: for each inode #,

⇒ location of latest version on disk

inode #100

"F": imap: 100 ⇒ 1~~13~~ 1002

mem



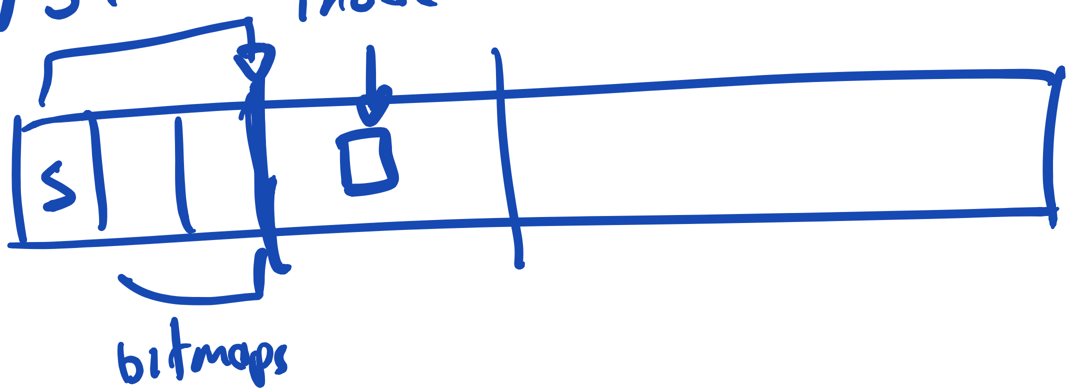
Reads:

directory contents:
same as before

inode #

"F"	100

VSFS: inode table



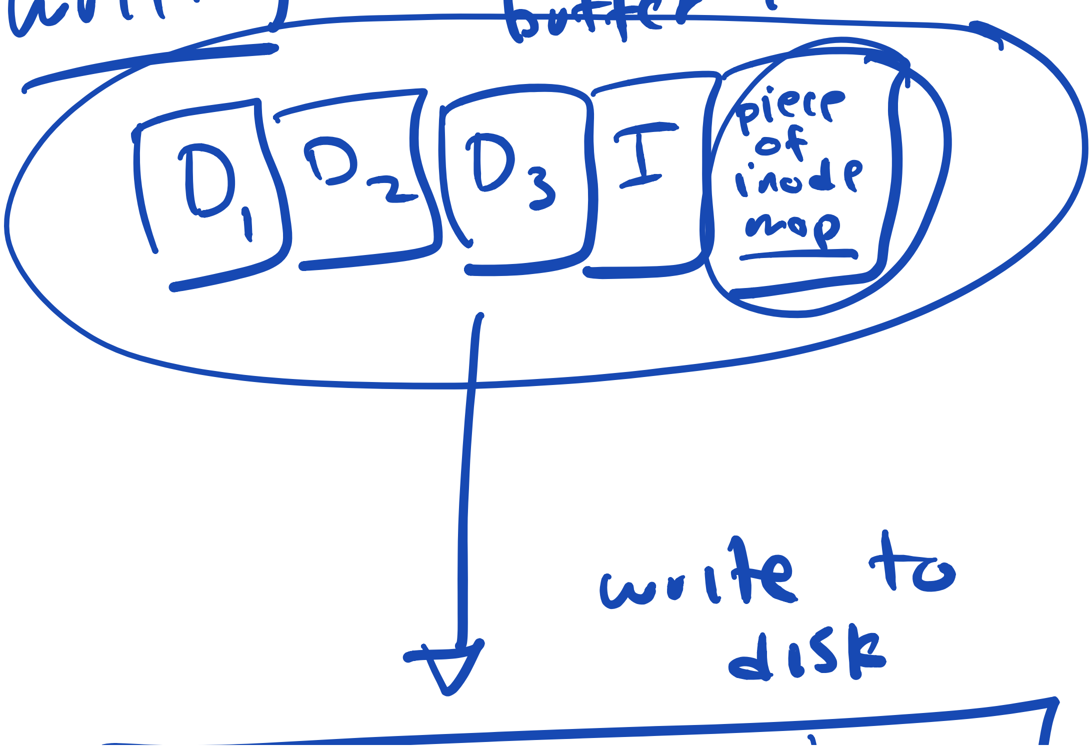
But: crash! (lose inode map)

options:

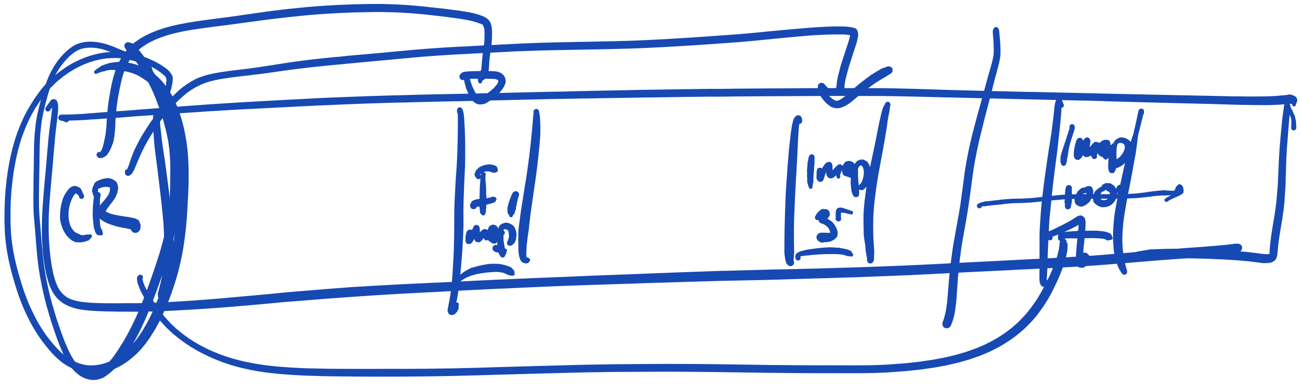
1) write out pieces of inode map

2) reboot: scan entire disk, find "latest" of each inode
$\Rightarrow$ rebuild inode map

writing: buffer in mem



periodically: write out one more data structure: Checkpoint Region



Old: "overwrite in place"

=> fixed location for inodes

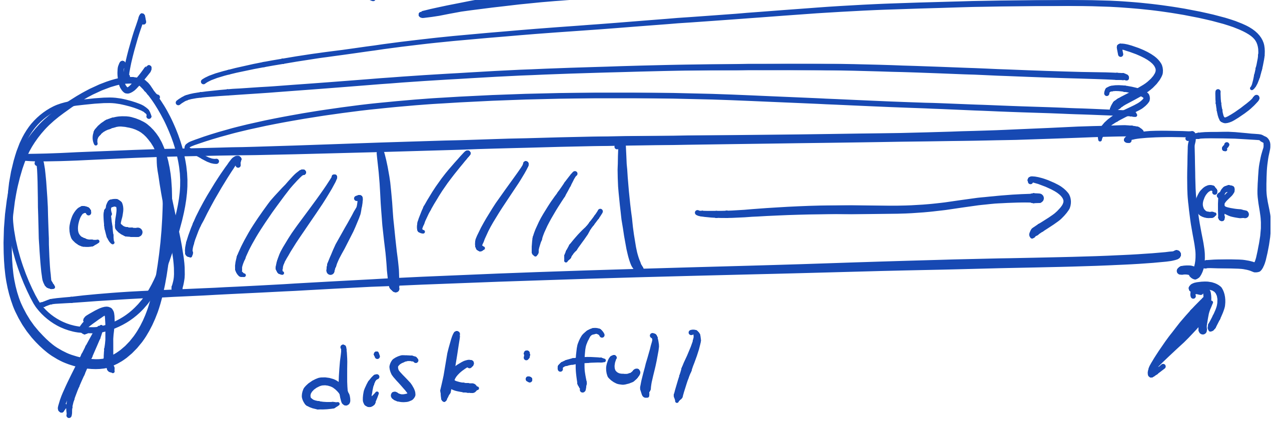
=> once written, data stays in place

New: "copy on write" (cow)

LFS: "cow"-based file system

Problem: lots of old copies

⇒ [Garbage] ("dead")



"Background Process":

⇒ cleaner

→ read old segments

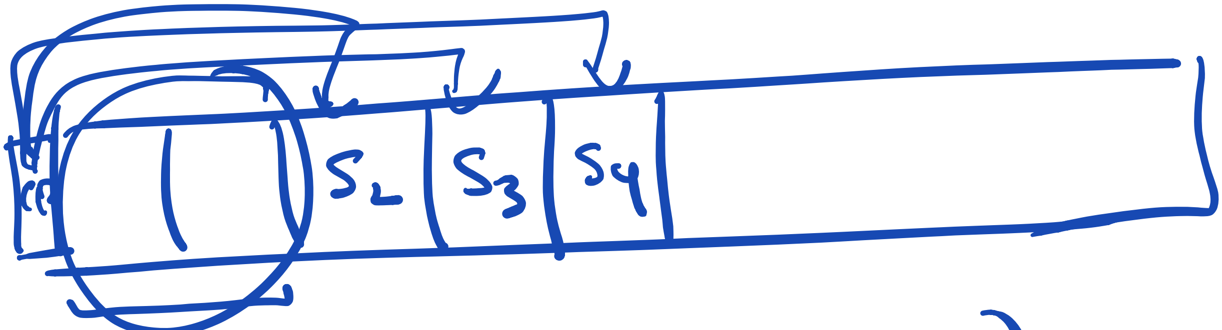
→ determines what is live / not

→ write out live data

→ free old segments

Before

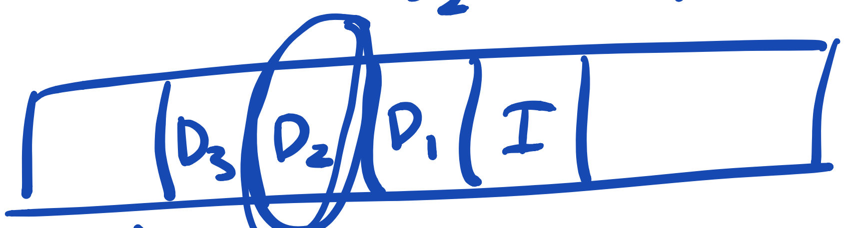
cleaner:
reads S_0, S_1
determine liveness
write S_4 (w/ live info)
free S_0, S_1



segment (up close):
how to tell if something is "live"?

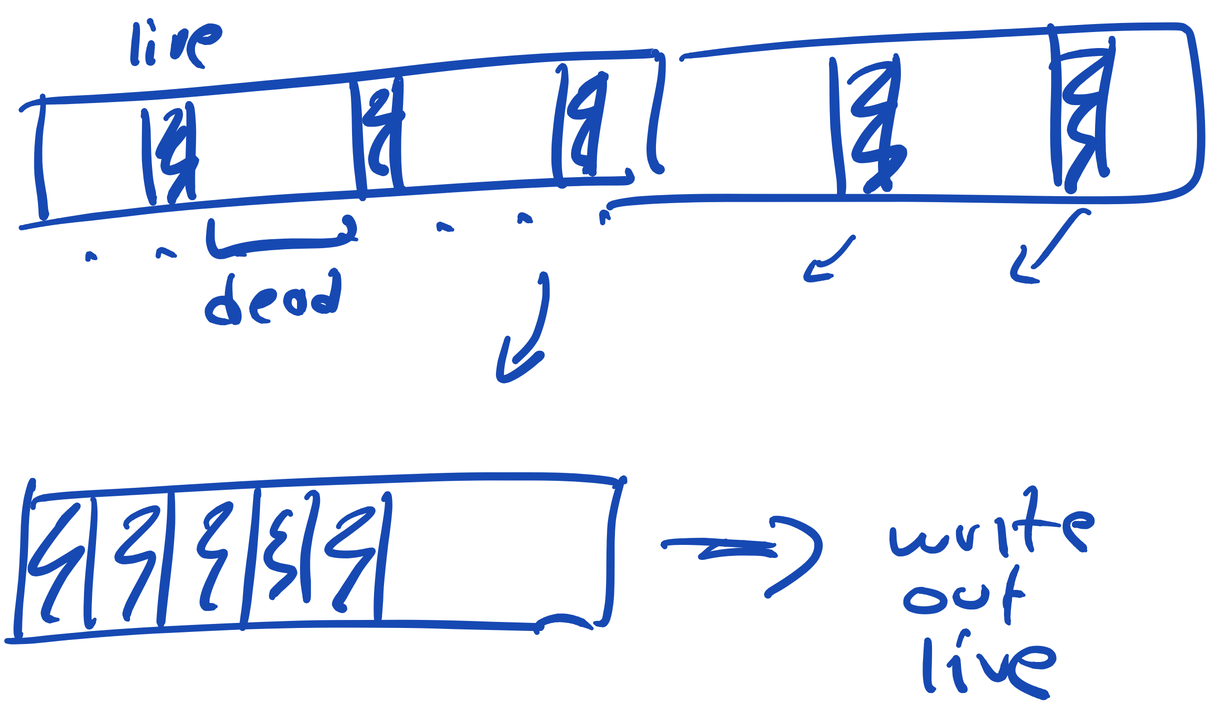
know: inode #, offset

D_2: live? dead?

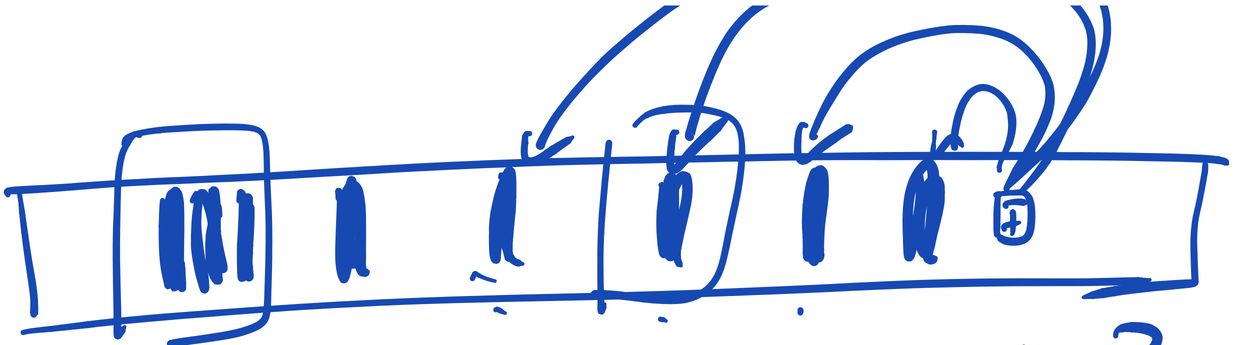


(data) live: is this pointed to by an inode that's in inode map?

extra info w/ segment:
segment summary block



Performance: LFS
→ sum of foreground
+ background
I/O traffic



fragmentation: how to handle?

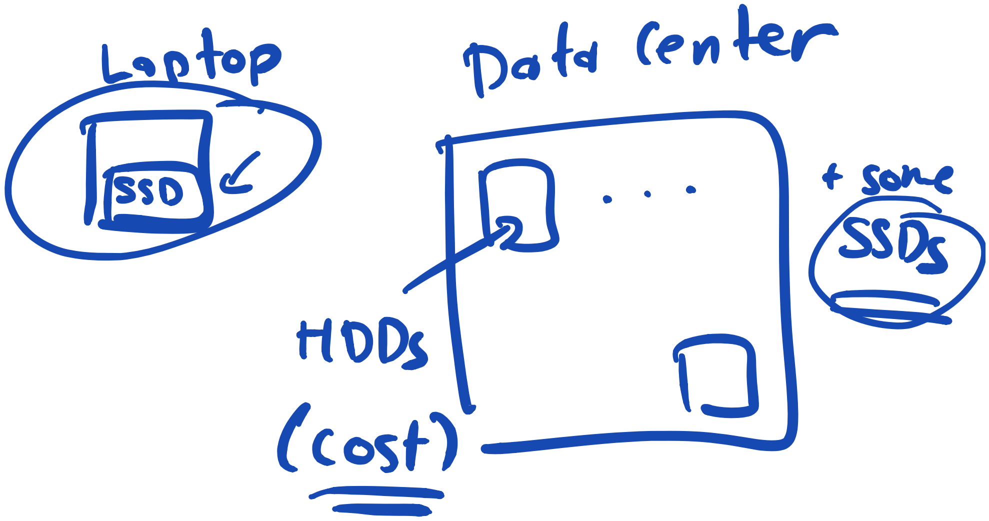
⇒ fragmented file:

read, write it out

(whole thing)

(user program)

SSDs:



Flash-based

⇒ Solid-State Devices

⇒ (computer chips) → non-volatile

⇒ serves as persistent memory

(can survive power loss)

Internals:

⇒ Flash chips

Interface:

just same as hard drive

⇒ blocks (sectors)

⇒ read/write

SSD: has many chips

chip: organized as follows

pages: ~2KB, 4KB

block (~256KB) ⇒ "erase" block

store bit:
"trapping charge" ⇒ 1, 0

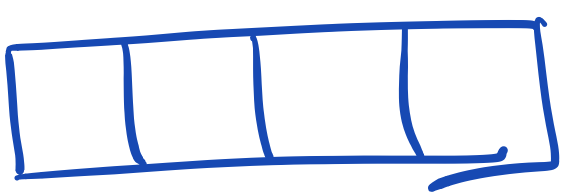
operations: different than just read/write

read (page) ⇒ gives data

update: 2 steps

erase (block ~256KB) ⇒ makes block writeable

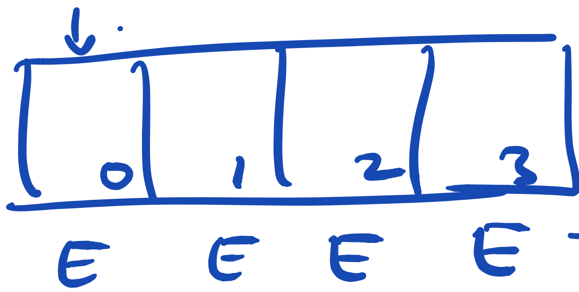
program (page) (w/ in block) ⇒ set its contents



invalid (not writeable)

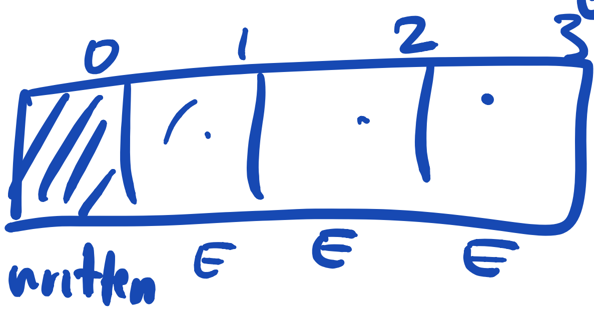
I I I I

erase (block)



erased (writeable)

program(0) ⇒ data



(v)

valid

once programmed ⇒ to overwrite have to erase **entire block**

Performance / **Reliability**

micro

perf:

reads: <u>fast</u> (<u>10s of μs</u>)

erase (block): <u>slow</u> (~ a few ms)

program (page): somewhat fast (<u>100 μs</u>)

milli

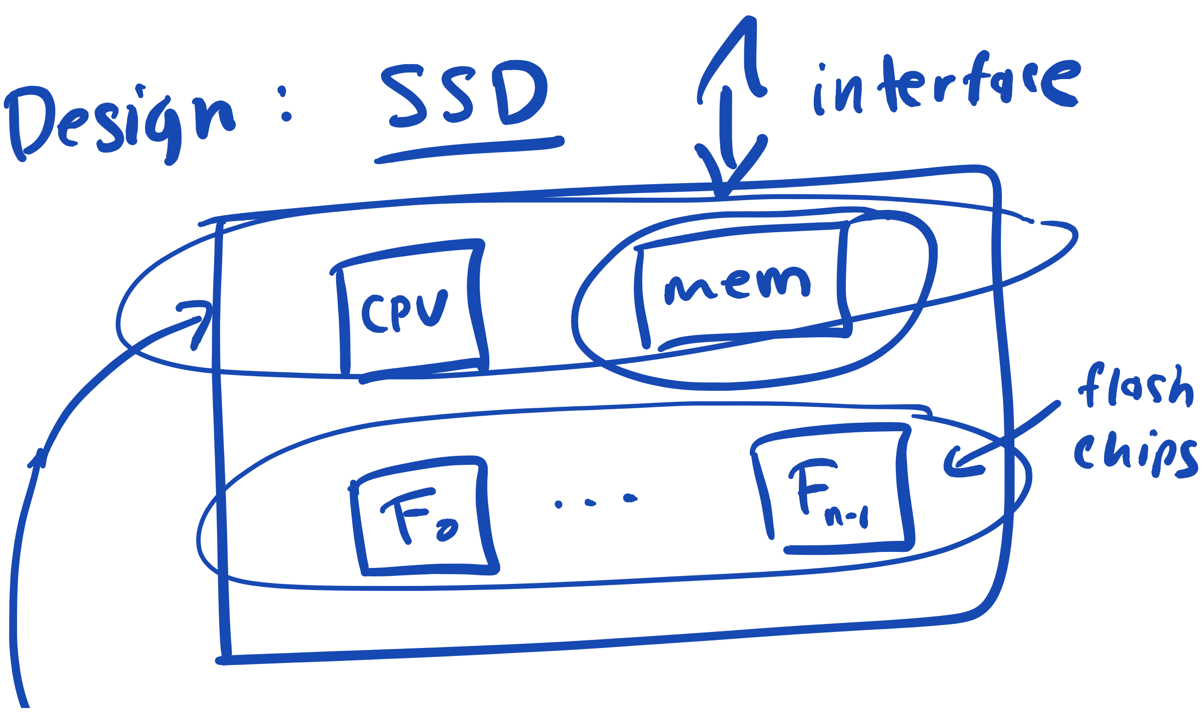
<u>Reliability:</u>

wear out:

<u>erase/program</u> <u>block</u> "too many" times

⇒ unusable

Design: <u>SSD</u>

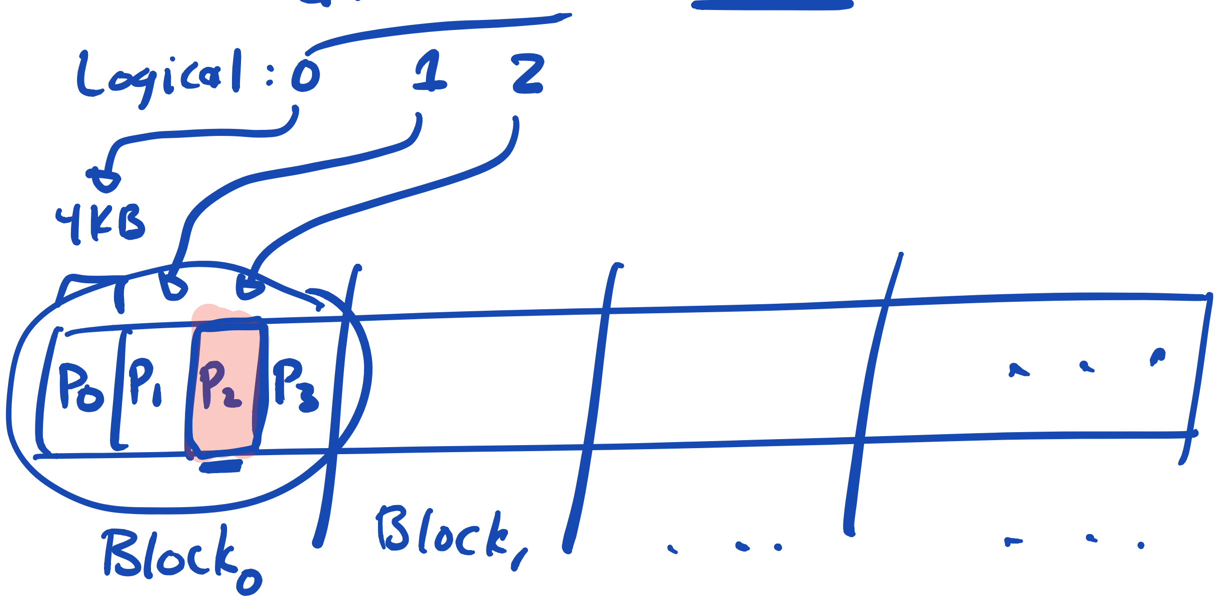


Flash Translation Layer
(FTL)

=> takes reads/writes
to interface

=> map them into
low-level flash
ops (reads, erases, programs)

Bad way to build FTL:
"direct mapped"



reads: easy   read(address)

writes: e.g. write(addr: 2)

read P_0, P_1, P_3

(put it somewhere: where?)

erase block(0)

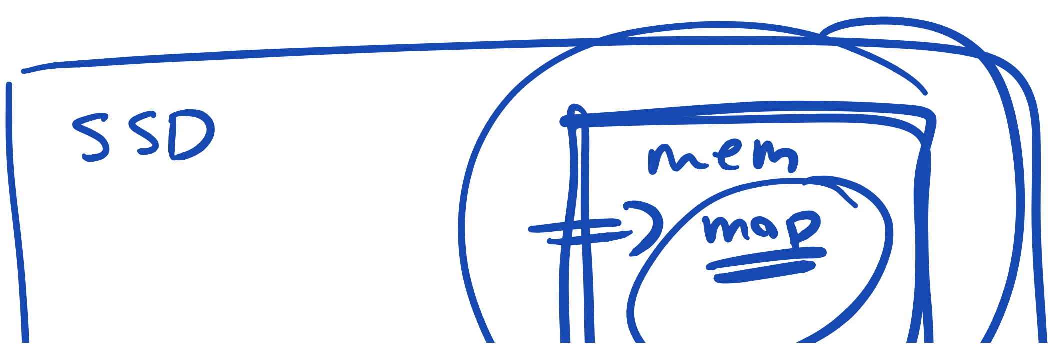
program P_0, P_1, P_2', P_3

⇒ [slow]   ⇒ wearout

what to do?

FTL: better w/ logging

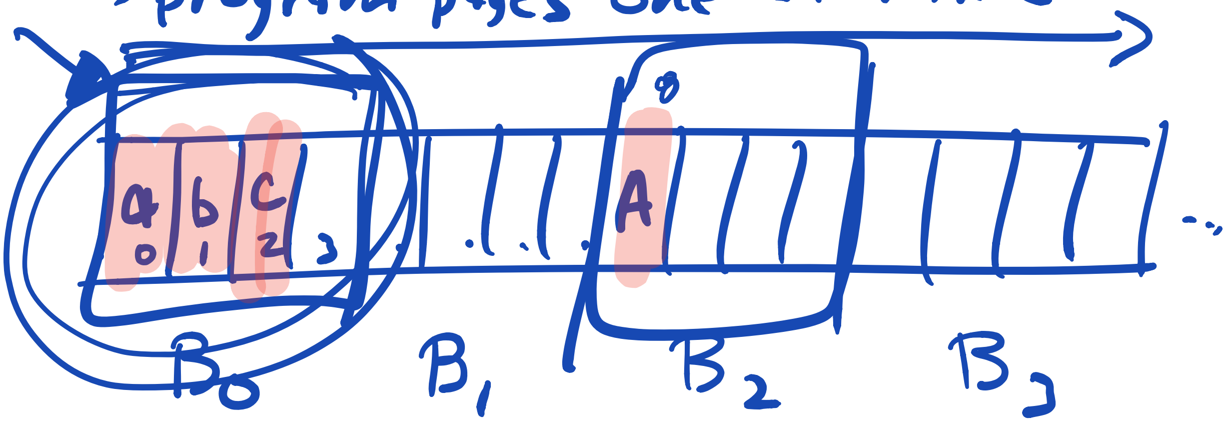
(copy-on-write)



logical writes (4KB in size) : log-style write

→ Erase block once

→ program pages one at a time



→ record (in memory)

→ in FTL

logical block → physical page mapping

→ write(100, "a")

write(101, "b")

:

write(200, "c")

FTL: 100 → 8   200 → 2
101 → 1

write (100 "A")

Problems:

⇒ old: garbage

⇒ old: how to recover FTL translations after crash

⇒ new: wear leveling (in FTL)

⇒ new: how big is FTL?

⇒ 1 TB SSD
per page mappings
page: 1 KB
how many mappings?

⇒ goal: not need

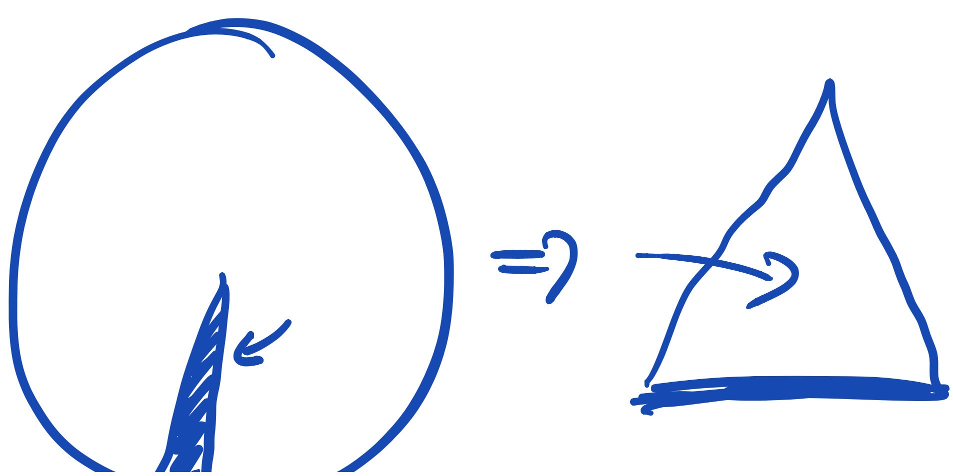
all those mappings
in device
memory

=) better structure

=) only keep "active"
pieces in memory
(caching, swapping)

=) block mappings:
(refer to entire
blocks not pages)

Topics not covered:
Most Stuff



→

→

Maintenance:

⇒ P5 (Friday)
→ send mail

⇒ Final Tuesday (5/8)
evening ~~5/6~~

Review Session:
Monday 4-6 room TBA

"Final" ⇒ emphasis on
"since midterm"

Cheat Sheet: reclaim
old: ok

⇒ 2 sep. pages (4 sides)

=) ~~Recommended Courses~~

=) ~~Intel etc.~~ Meltdown ~~Spectre~~

=) ~~Favorite Architecture~~ :

=) inode, imap ~~=) ipods, iphones~~

=) which OS book ? did I use

=) (Tanenbaum) =) not an A

=) wisdom → Experience != wisdom

\+ thinking → wisdom

=) Comm. :

=) what to work on

Robotics

=) People → who to learn from ?

X

=) Luck ← 12


Prof
Grad Student
1st/2nd
5
5th
9th
1 mph
read_sonars()
CS
OS
H/W