4/24

welcome to CS 537!

should we hold class outside?

□ Yes

☒ No

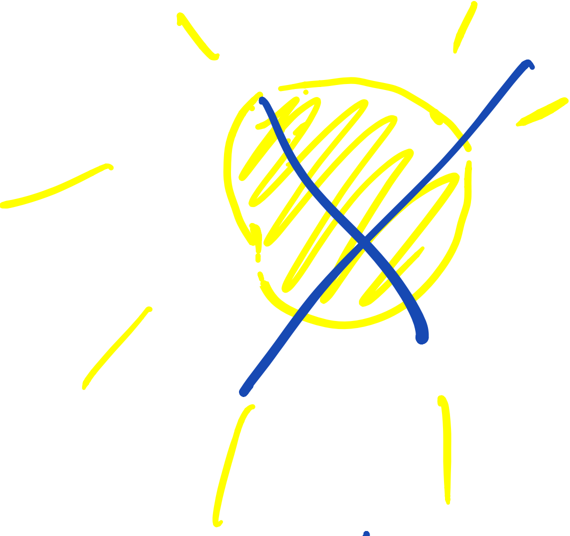
Do you believe that this is a REAL

QUESTION

?

☒ No

Today



Wrong Thing:

Enjoy Life

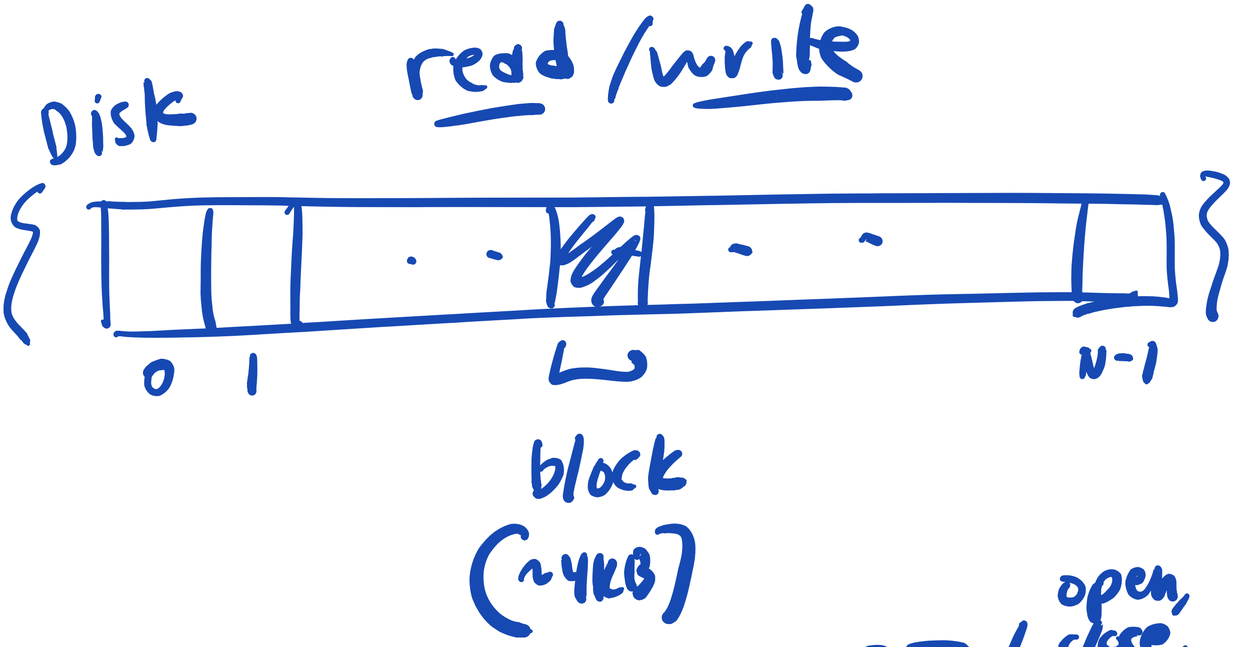
Later

Today: File Systems (implementation)

-> **Locality** } ②

-> [**Crash Consistency**] } ①

Review:

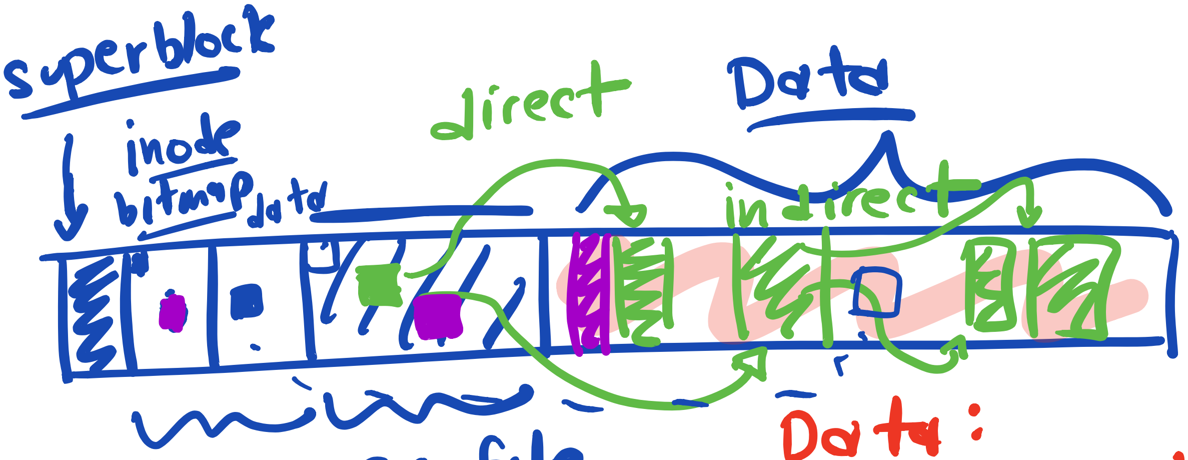


File **System:** **API** ↓ open, close, read etc

uses

main memory cache/write buffer | ops on disk
=> low-level reads/writes

FS:
{ => On-disk structures
=> Access Methods



alloc. structures

per-file metadata
=> inode table

Data:
-> user data
-> dir data
-> indirect blocks

create:
"/foo"
(empty file)

Do

Sett on Thurs:
5:30

not
erase

3:30
~~Drinks on him~~!

Crash: what is a "crash"?

=> power loss ← UPS

=> BSOD: kernel panic/ bug

=> user restarts

why important for FS?

=> in the middle of an update

Examples: Updates

⇒ **File creation**:

{what data structures are updated?}

/ ⇐ root ⇒ creat("/foo");

e.g. ⇒ inode bitmap (find free, mark it used)

10 ⇐ ⇒ inode: all its various contents

⇒ root directory: data

.	
..	
foo	10

name ⇒ low-level name (inode #)

⇒ root inode: update times

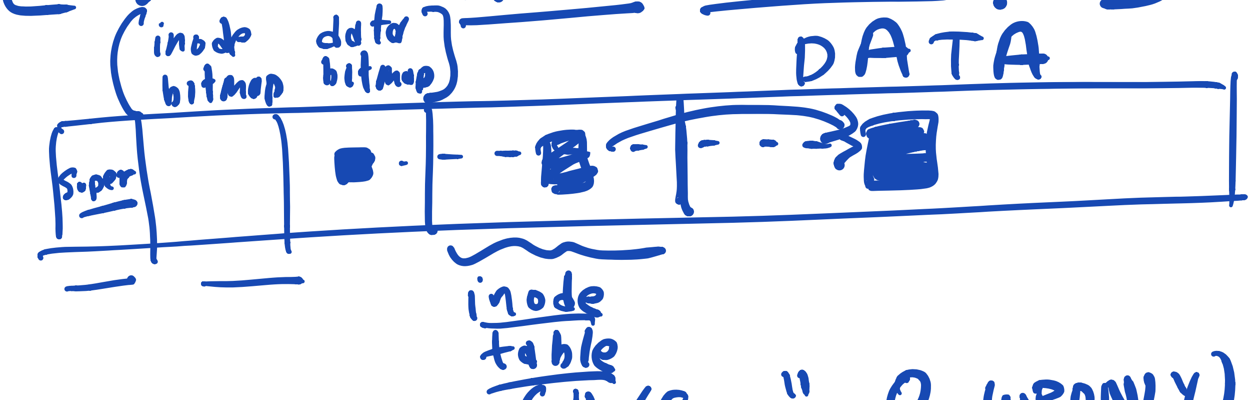
[could also (if dir grows)
⇒ data bitmap,

new data block

⇒existing file, goal ⇒ append new data block

⇒ how API? (system calls)

⇒ which blocks written?

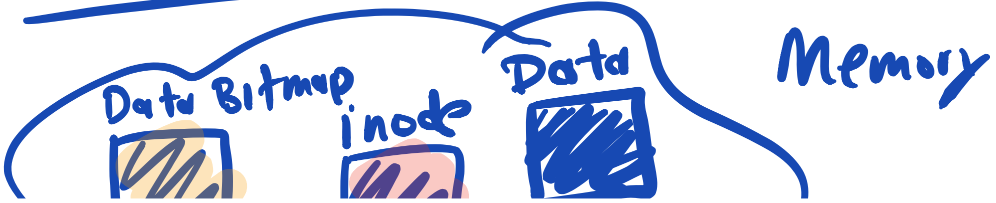


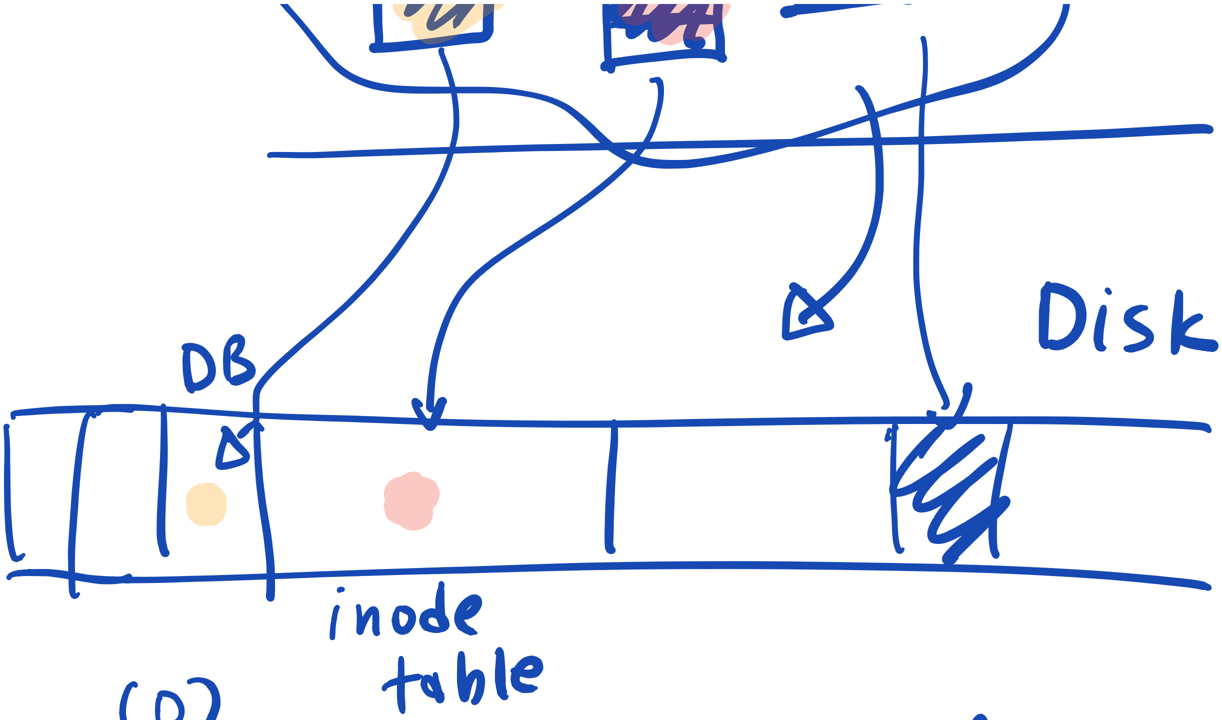
```
int fd = open("/foo", O_WRONLY);
lseek(fd, 0, SEEK_END);
write(fd, buffer, size);
```

4KB

```
close(fd);
```

Data/Metadata:





(D)

Data, Inode, Data Bitmap:

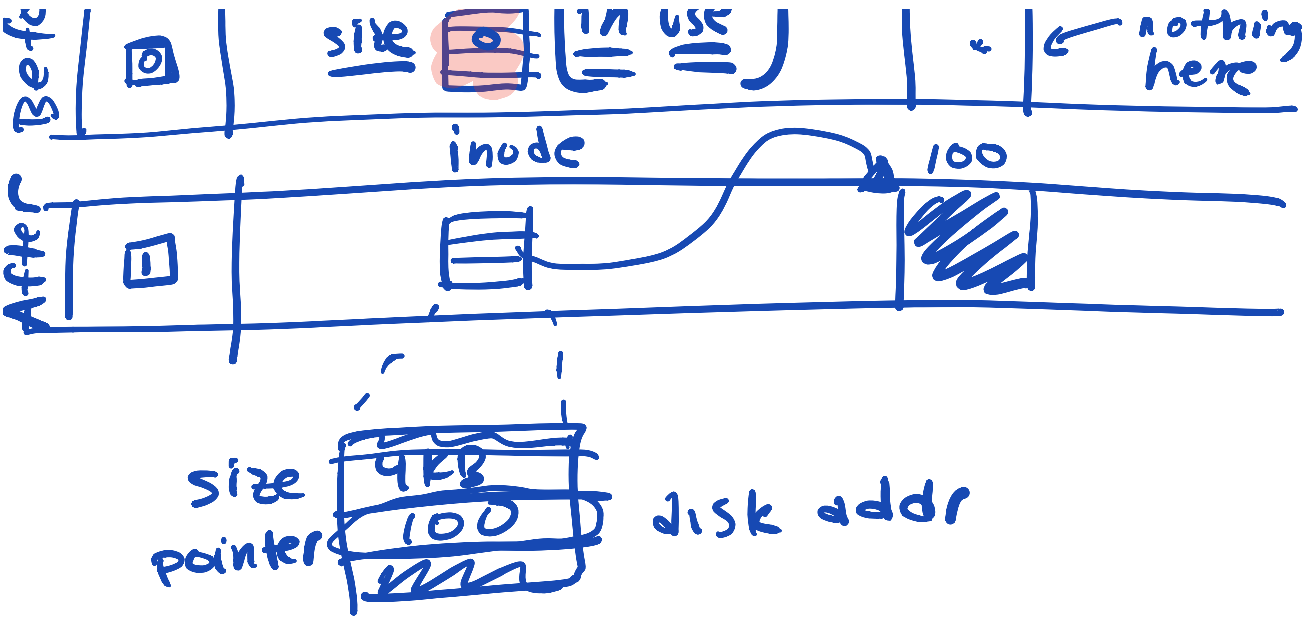
(D) (I) (DB)

orders of writing:

D	D	I	I	DB	DB
I	DB	D	DB	I	D
DB	I	DB	D	D	I

crash here
or
here

[what happens?]



(D) Data (D), Inode (I), Data Bitmap (DB):

orders of writing:

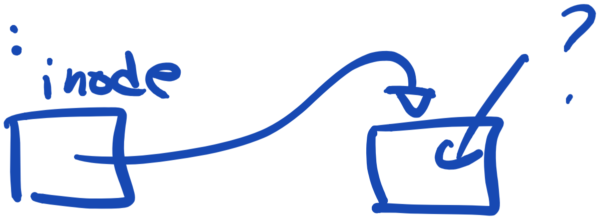
D ①	D ③	I ⑤	I	DB	DB
I ②	DB ④	D ⑥	DB	I	D
DB	I	DB	D	D	I

① ③ looks good! file system metadata "consistent"

② inode <-> data bitmap don't agree! => inconsistent

④ inconsistent : space leak

⑤ inconsistent : could read garbage (or zeroes) (or privacy problem)



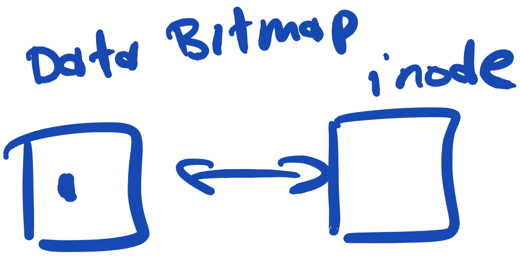
Solutions ?

Check : is the FS consistent ?

=> before usage (before mount)

=> file system check (fsck)

=> [focus of PS (good news)]



But: Disks got large
(RAID too)

=> too slow?

Something else: Journaling
(OR write-ahead logging)

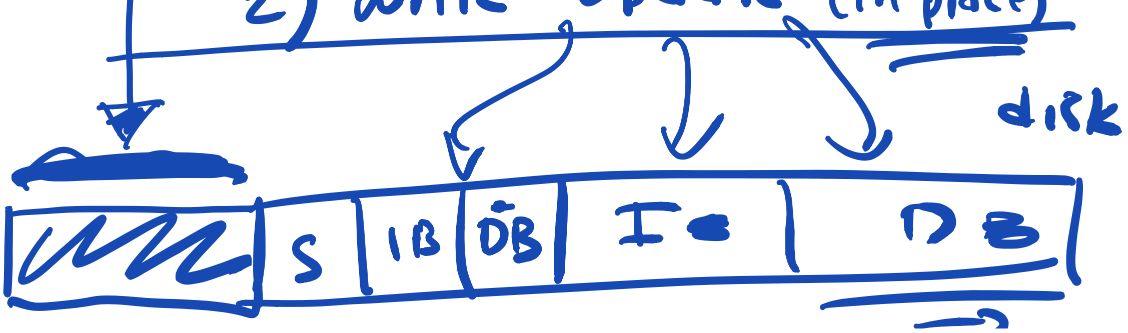
=> Eager:
-> some work w/ every update

-> recovery (after crash): fast

1) write note to log/journal about pending update

2) write update (in place)

memory



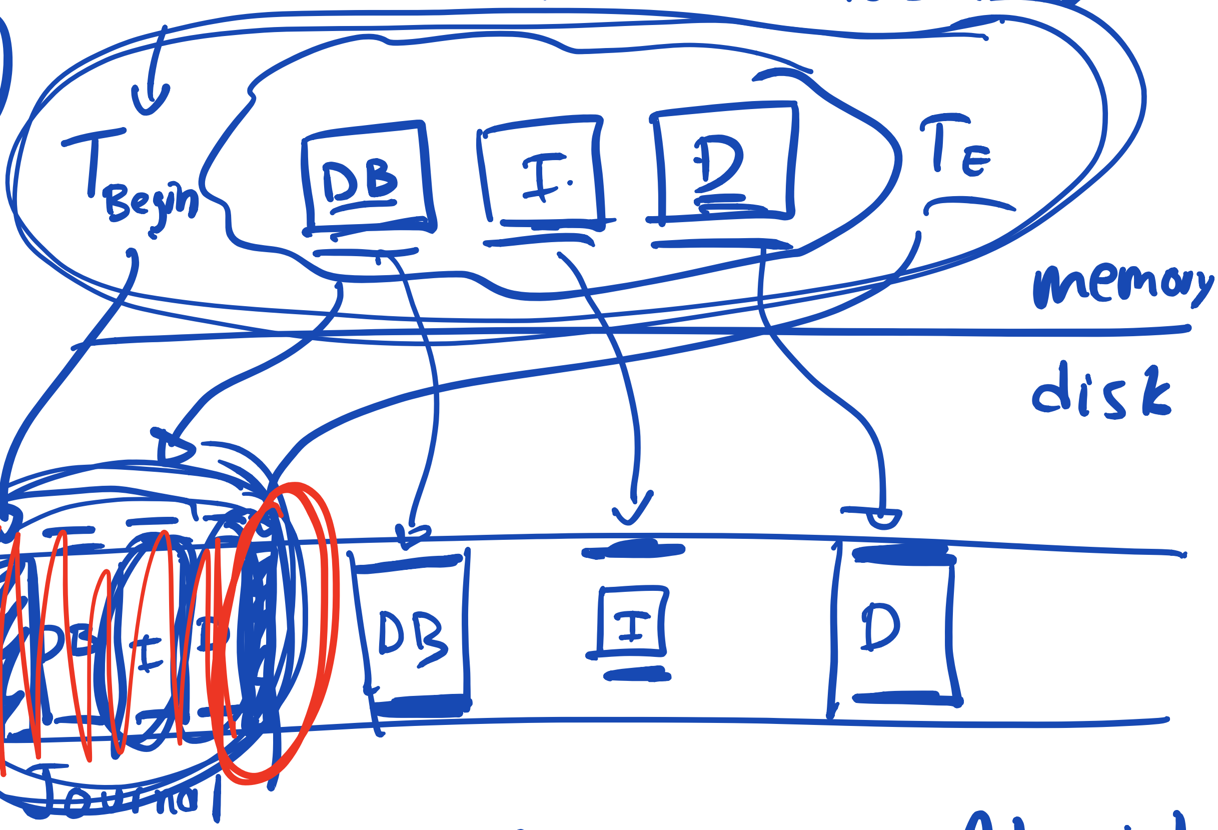
Journal
(or
Log)

Details :

example
-> append to existing file

info about update => location

①



All or none : Atomicity

② writes "in place"
=> DB, I, D

Crash:

[during step 2 (after step 1)]
recover from journal

during step 1 :
or (issue all writes)
to journal

Aside : disk guarantees
that any 512 byte write
is atomic

but larger write may
partially complete

Step1 => 2 parts

1a) write Trans. Begin +
contents (not TEND)
to journal

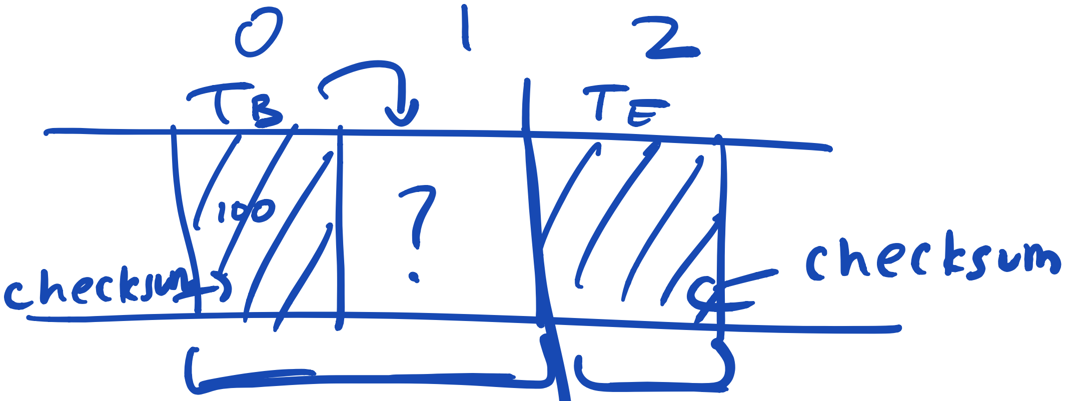
=> wait

1b) write TEND

=> transaction commit

wait

2) in place updates



2005: transaction checksum => {Linux ext4}

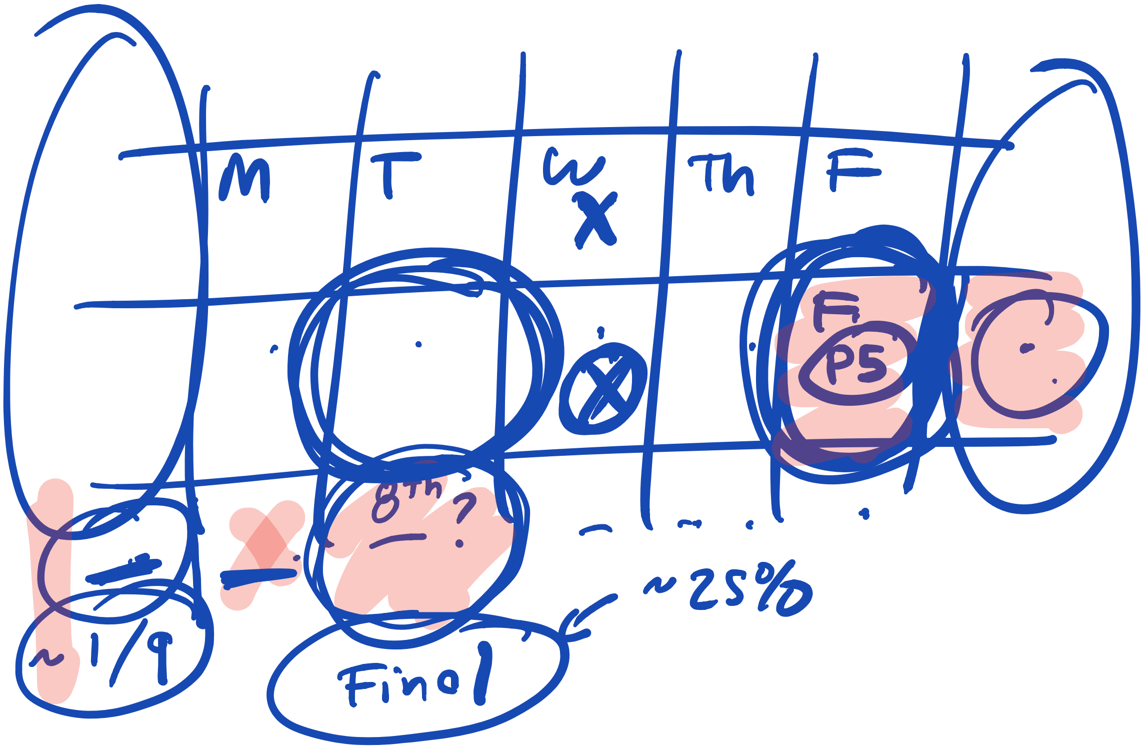
Crash Consistency:

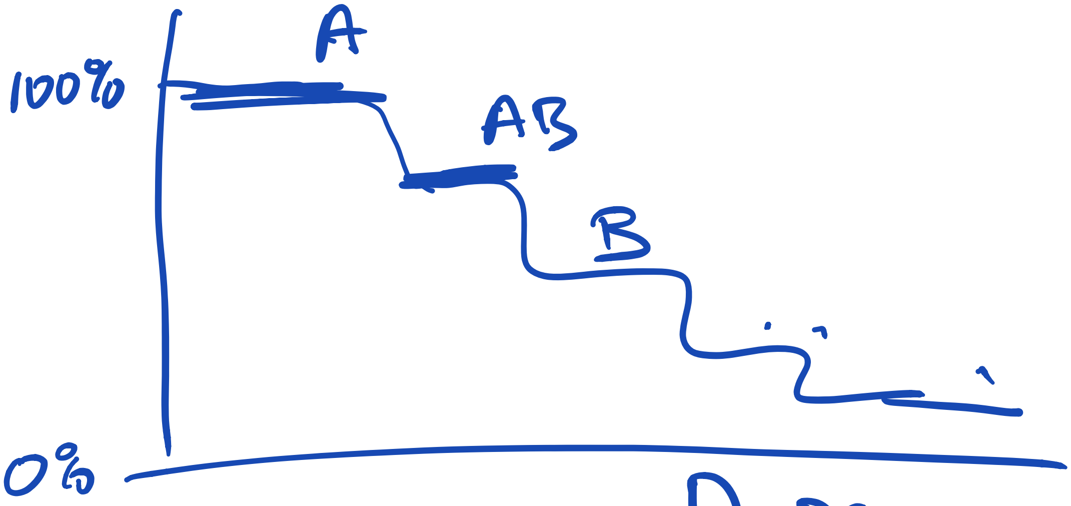
=> [FS] ∈ initial deadline

xv6 file system image

C/Linux

look for inconsistencies





D og

Ba D

Locality

Hard Disks: long seeks costly

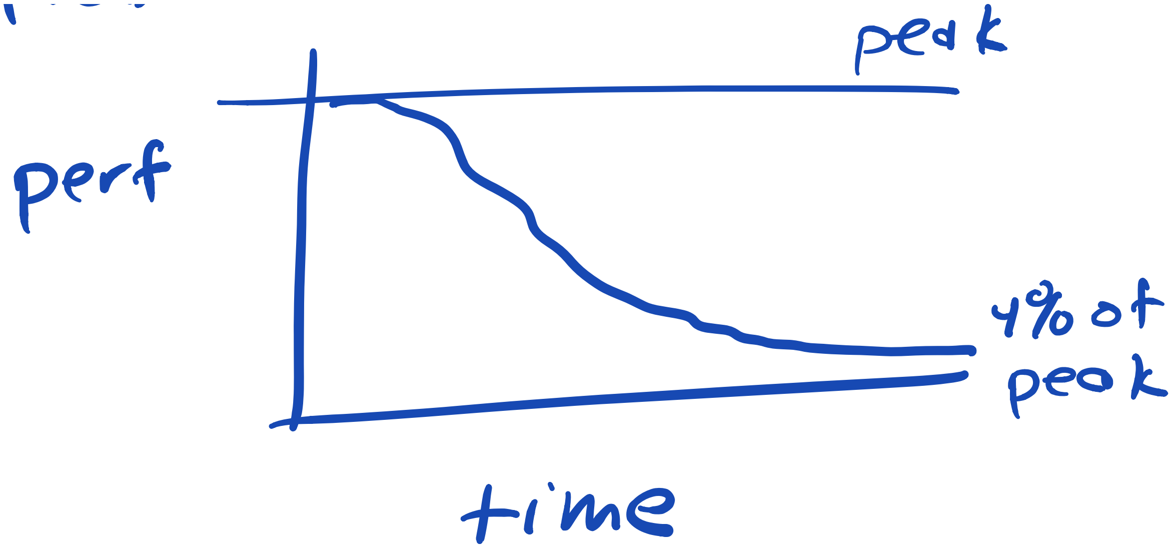
SSDs: locality important too

goal: put files/etc. on disk ⇒ accessed together should be near one another

Fast File System (FFS)

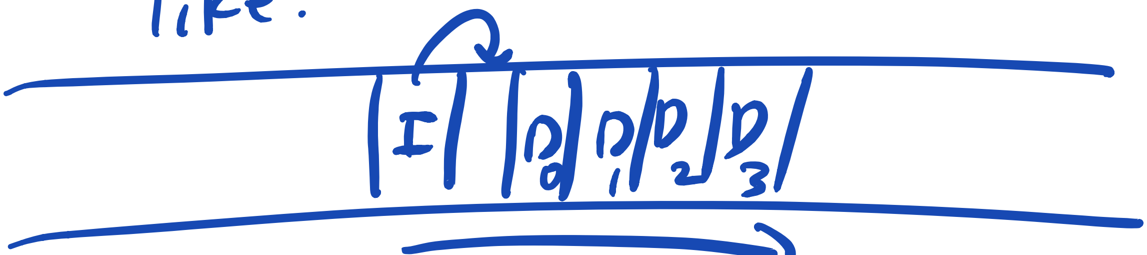
⇒ Performance

Problem: old unix was slow

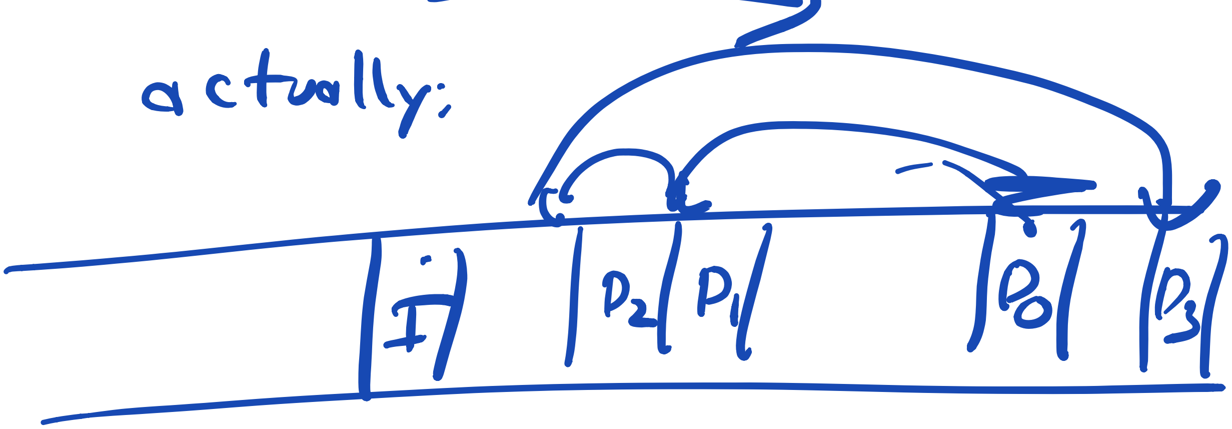


⇒ data blocks, inodes scattered through disk

like:



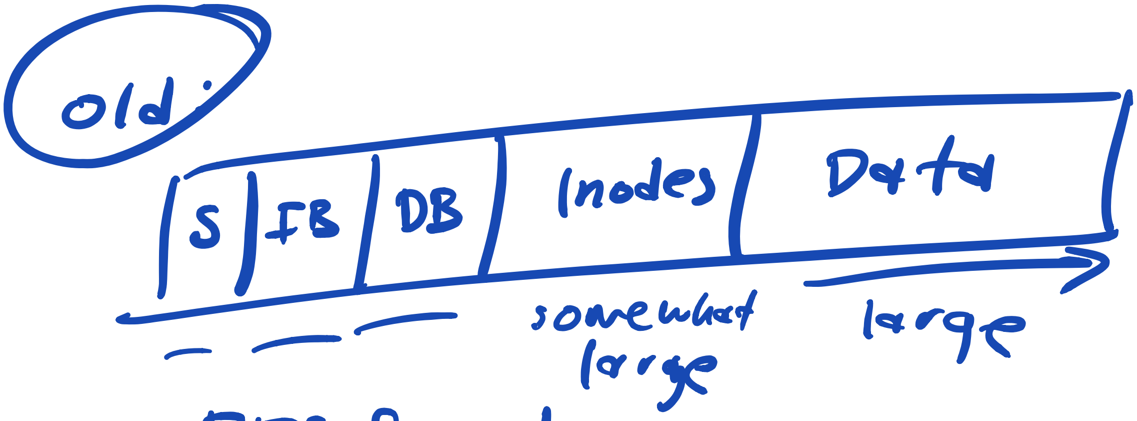
actually:



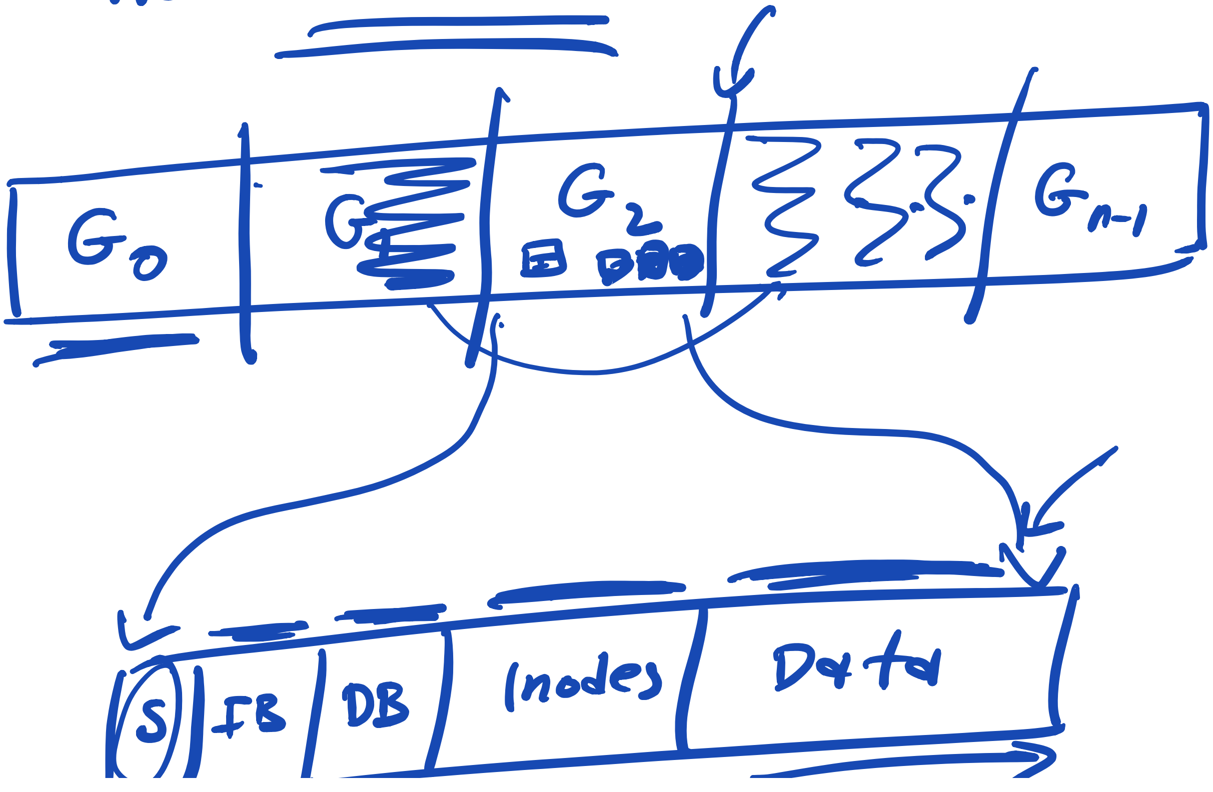
⇒

FFS: Treat Disk like a Disk!

On-disk structure: change



new FFS format:



Goal: Put related stuff into same group (unrelated → different)

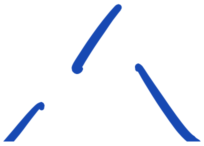
what is "related"?

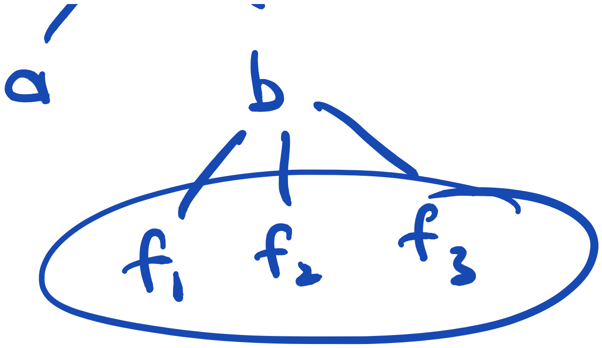
- → data same file
- → files in same dir.
- → inodes/data

make new dir:

pick some group X (how?) → (free space, inodes)

all files in dir (inodes/data) ⇒ group X

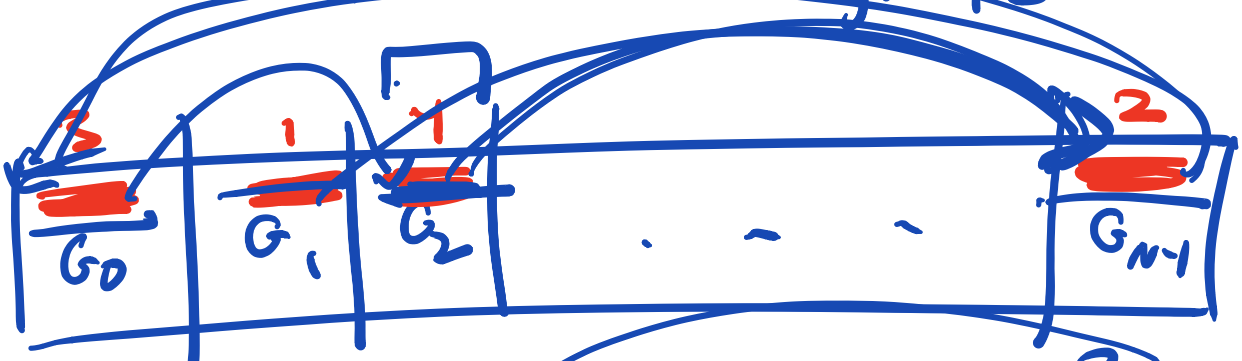




Large Files : exception

only put first N blocks of file => desired group

put each next chunk of file in other groups



Problem: Performance?

how big is each chunk?

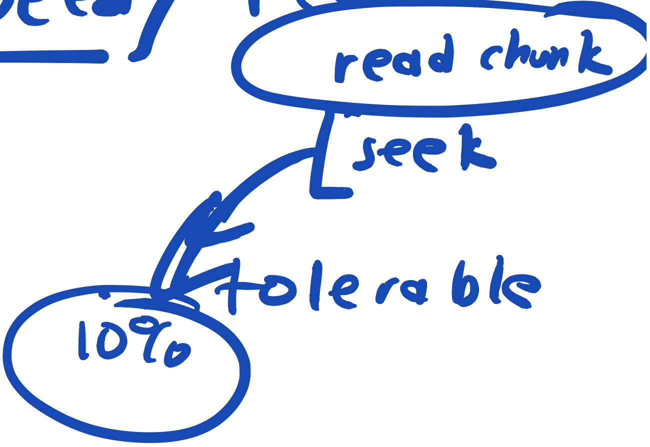
=> need to know:

- => how big file is ✓
- => group size ✓
- => memory ✓

=> cost of seek

=> (transfer speed)

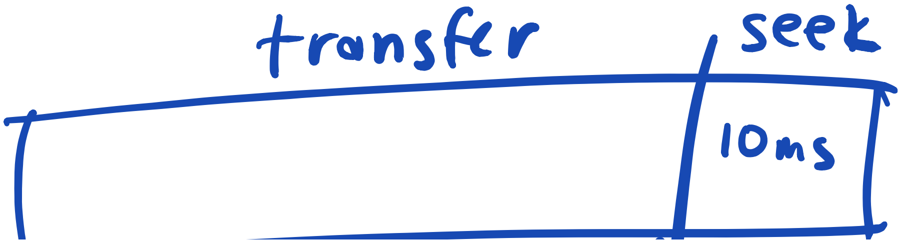
read:



most transfer :

transfer : 100 MB/s

seek (avg) : 10 ms



$\underbrace{\qquad\qquad\qquad\qquad}_{100\text{ms}} \Rightarrow \boxed{.1 \text{ sec}}$

$\boxed{10 \times \text{MB}} \cdot \frac{\text{sec}}{100 \text{ MB}} =$

⇒ [**Final**] ⇐ Study this

FFS : { first time: same API, new implementation }

⇒ symbolic link

⇒ long file names

⇒ other perf. opts (not as relevant)

Treat Disk like Disk

⇒ (peak, sustained

perf.