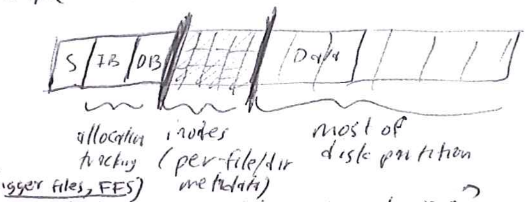
LECTURE

FFS

Review:

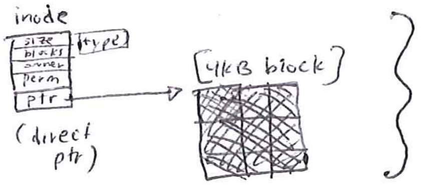
Last time: FS implementation

simple unix fs



today: (bigger files, FFS)

=> support for large files, directories?



how to make bigger?

→ more direct ptrs

→ linked (FAT)

next
0
1
2
3
3
EOF

→ extents (ptr, length)

→ indirect blocks (multi-level index)

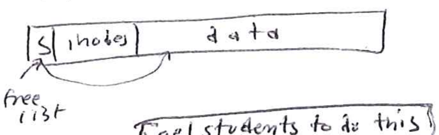
[example: how to calc block # of offset X]

=> FFS fast file system old unix fs: easy to use but...

problem: performance 2% of disk peak

{keep: same interface
change: underlying impl}

old unix fs:

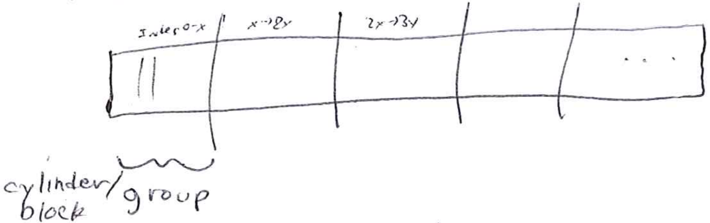


512-byte (small) blocks

problems: [get students to do this]
→ inodes far from data
→ related files: far from each other
→ free list: fragmented (example)

overall: structures are not "disk aware" (treats disk like RAM)

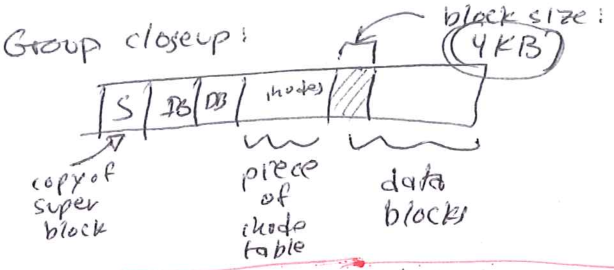
FFS: treat disk like disk!



1) chop disk into groups

2) put "related" things in same group (corrolary: "unrelated" in different groups)

Group closeup:



Q) given inode # X, how to calc. on-disk addr?

=> Bitmaps not free list: why? (can see if nearby blocks are free easily)

need: allocation policies

what is related?

{inode → it's own data
files in same directory (namespace-based locality)}

Q) always true?

mkdir: pick group w/ [high free # of inodes, data blks, low # of directories]

create file: place data blocks in same group as inode

exception: large files

didn't want to place a large file into one group: why?

(fills up group, can't put future "related" stuff there)

FFS rule: once file reaches certain size, place next chunk of file in some other group:

(how big should chunk be?)

{e.g. xfer: 50 MB/s (peak)
avg seek + rot: 10 ms}

to achieve 90% of peak b/w for large files?

other FFS fun:

→ sym links

→ long file names

→ parameterization of layout

→ sub-blocks (for really small files) => avoid internal frag

Summary: treat disk like disk (group related things)