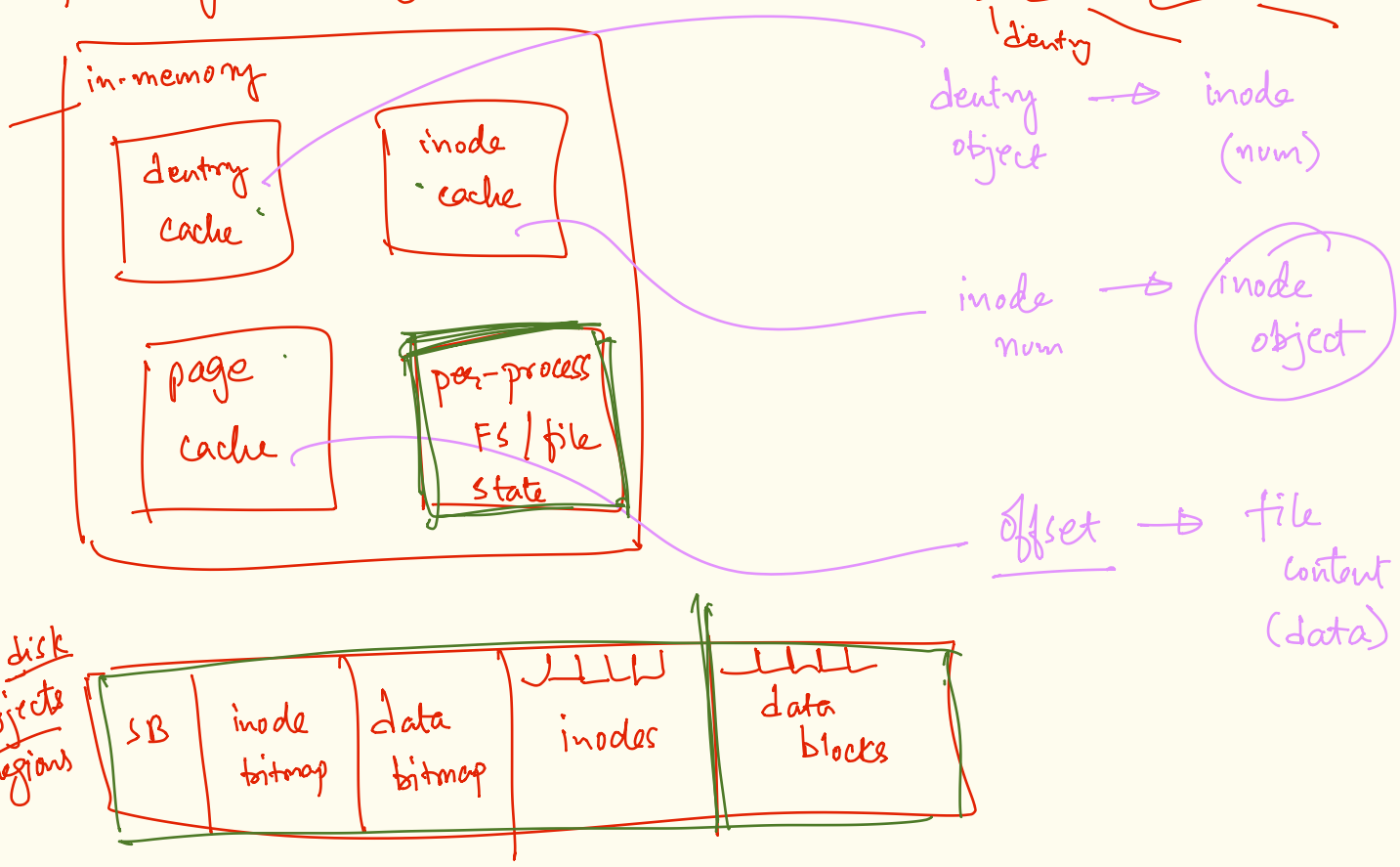
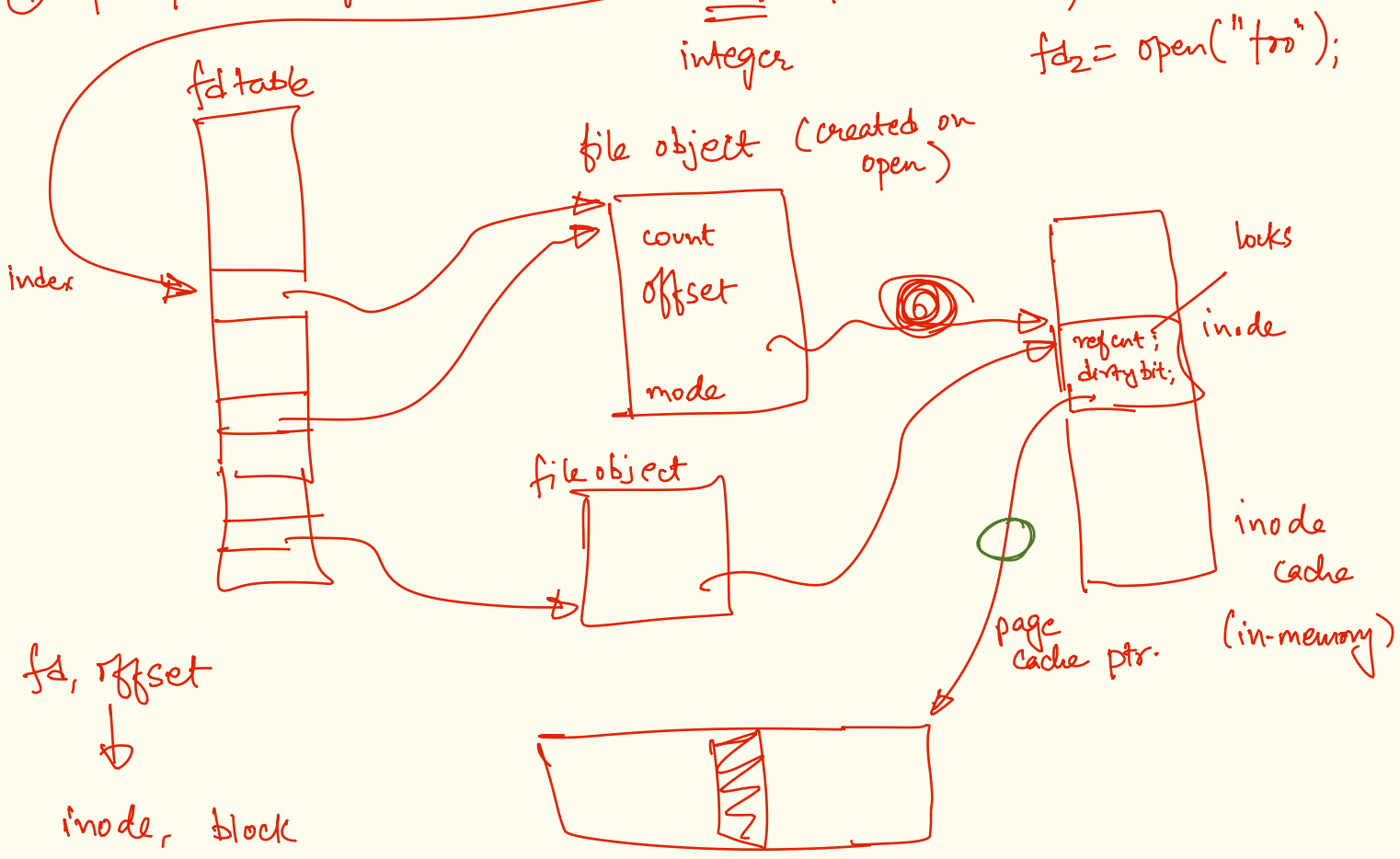
Lecture # 24

file system design & optimizations.

file system objects

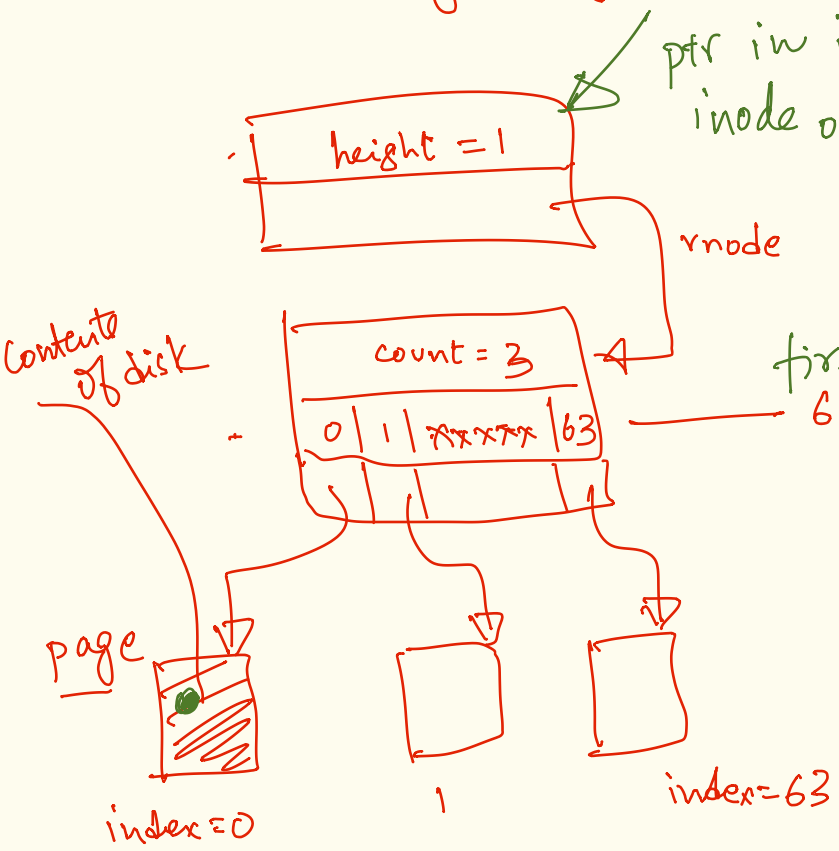


per-process file state.



e.g: Linux page cache.

- radix-tree style organization



first 64 entries.
each storing 1 page worth of disk content.

page size = 4KB

total cache size = 64 × 4KB = 256KB

terms in books.

page cache vs. buffer cache (blocks)

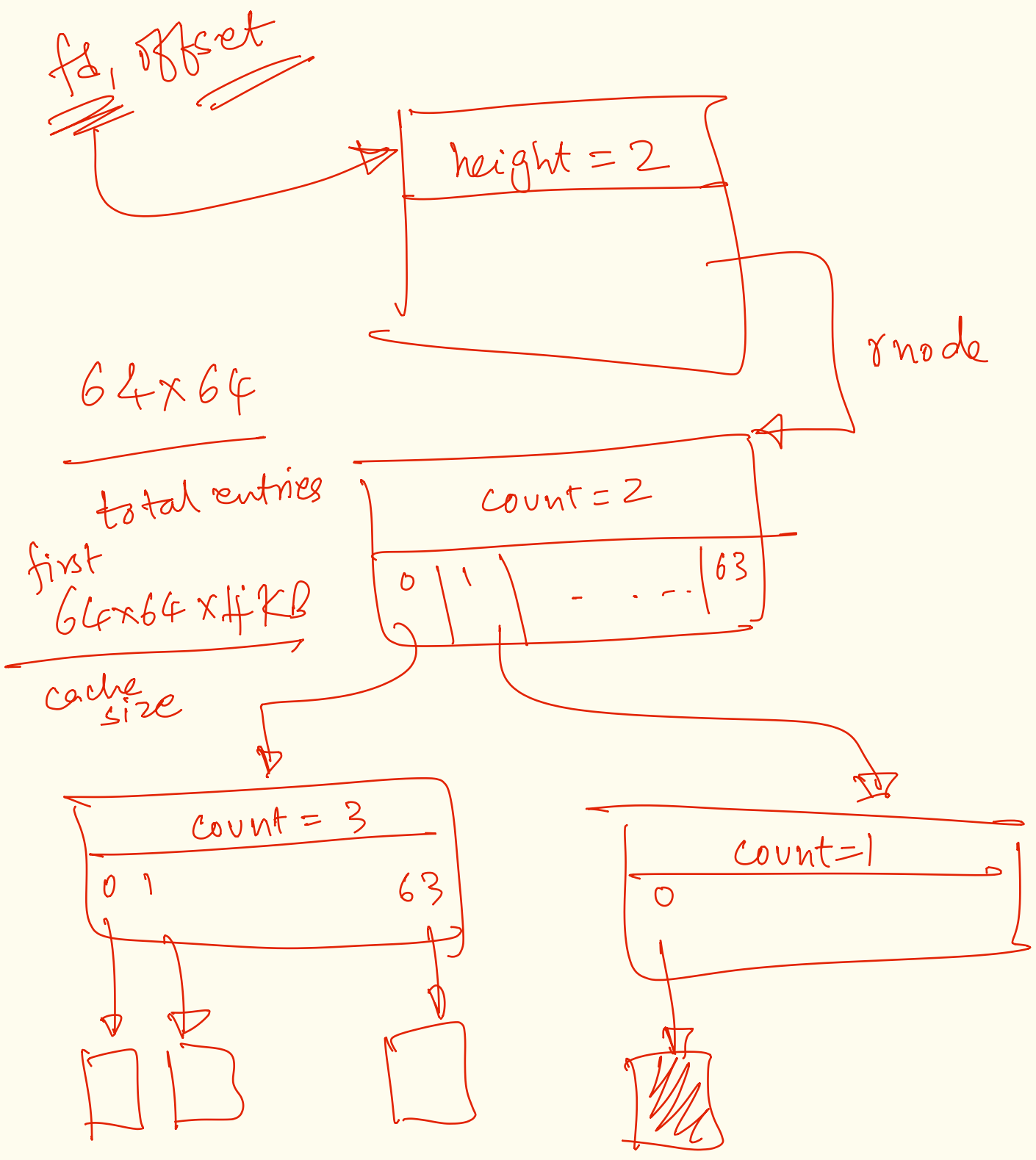
blocks ⟷ sectors

(os step) as a unit of physical RW

offset → page index

read on offset 10
5000
~~25000~~
256KB - 23

Suppose access is on offset 256KB + 7


fd, offset
height = 2
rnode
64x64
total entries
first
64x64x4KB
cache size
count = 2
0
1
63
count = 3
0
1
63
count=1
0

caches are great for reads

(*) what about writes?

- two copies of the same item/data.
- updates to a single copy ⇒ inconsistency.
- writes to memory are faster.
- large writes are economical/efficient than smaller writes.

[kflushd]

O_DIRECT

write-back caching policy

- optimistic outlook
- accumulate writes and flush to disk "later".
- write speeds are faster!
- risky!
  - inconsistency chaos
  - data loss.

write-through caching policy

- each write is through the cache and goes to disk synchronously.
- upfront cost to maintain consistency.
- writes are slower!
- reads still great!