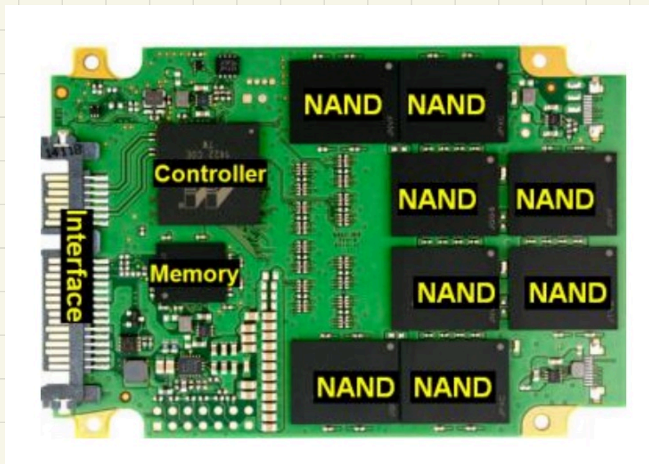
5/8/24

SSD & Filesys Basics

Solid State Drive

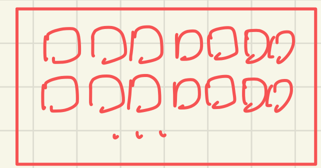
→ persistent, block addressable, large capacity

NAND Flash packages connected by multiple channels, highly parallel architecture



no moving parts

Units: blocks, pages



a block contains hundreds of pages,
page is the unit of read / write.
(4KB)

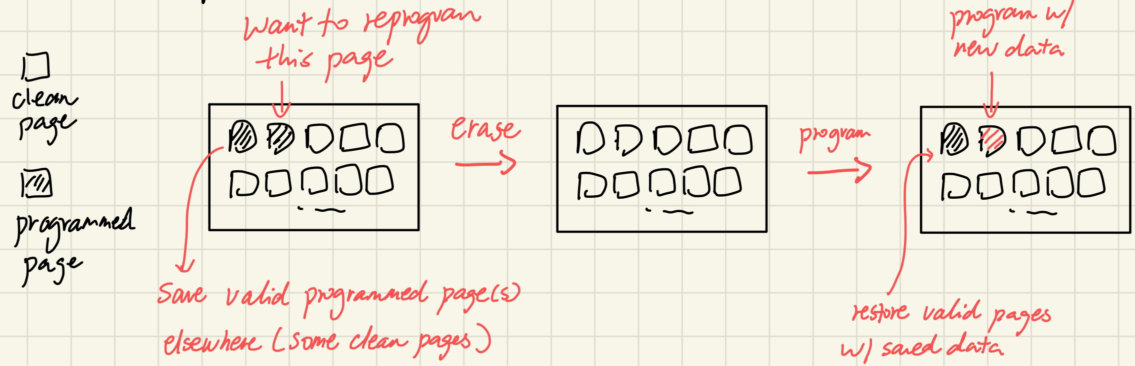
SSD Operations

→ **read a page** (4KB) : fast access ~10 μs

→ **write a page** (4KB) : can only write to a clean page (all bits are set to 1)
(program) program bits to 0s to write data ~ 100 μs

→ **erase a block** (1-8MB) : erase all pages within the block (all 1s)
slow operation ~ 1-3 ms

To do inplace update on SSD

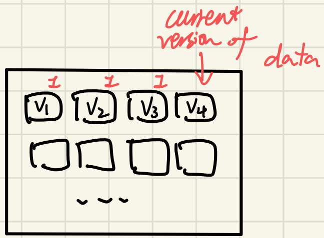


☆ there's a valid bit (metadata) associated w/ each page, easy way to mark garbage

SSD Reliability

- → a page can only reliably endure 10-100k writes
  - → repeated writes to pages cause frequently modified pages to wear out faster (no longer retain data reliably)

- → wear-leveling: spread writes to different pages to wear out pages evenly ⇒ each write moves the data to a new page



moving data around all the time not good for SSD clients, better transparent!

☆ Flash Translation Layer

- → translate logical block address
  ⇓
  physical block address

- → garbage collection
  - → move sparsely valid pages into a new block, frees up a block for erasure

SSD Request Latency

total time = access latency + transfer time + (erasure time)

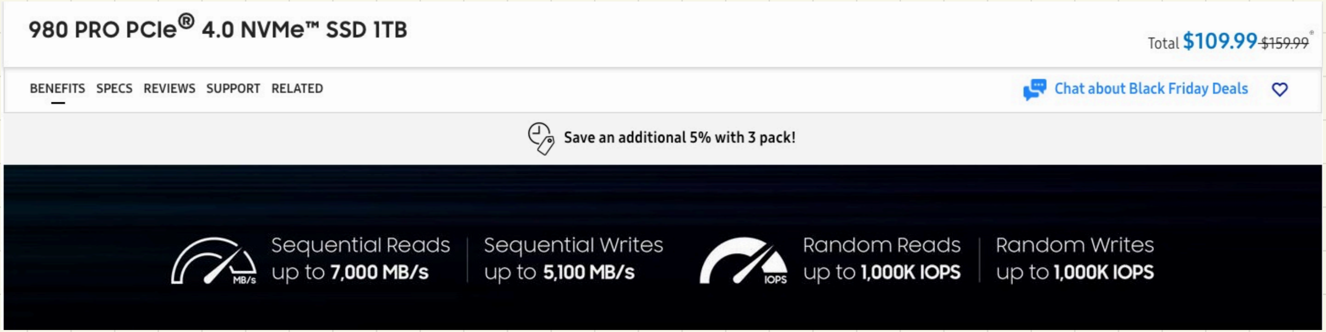
→ latency of a read page request given 10 μs read latency & 500 MiB/s

17.8 μs ⇓ 7.8 μs

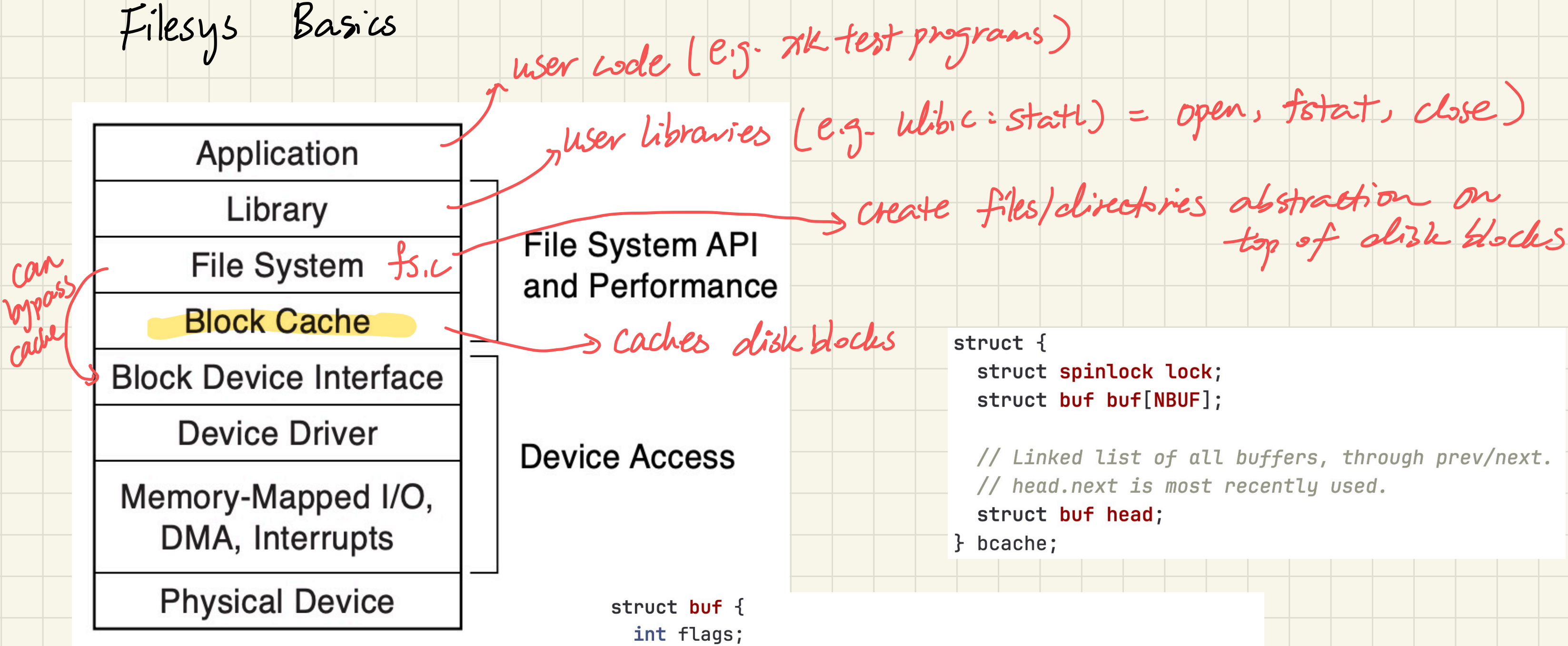
→ way less sensitive to access patterns

→ much closer performance for sequential & random accesses

may be parallelized.



Filesys Basics




```c
struct {
  struct spinlock lock;
  struct buf buf[NBUF];

  // Linked list of all buffers, through prev/next.
  // head.next is most recently used.
  struct buf head;
} bcache;
```

```c
struct buf {
  int flags;
  uint dev;
  uint blockno;
  struct sleeplock lock;
  uint refcnt;
  struct buf *prev; // LRU cache list
  struct buf *next;
  struct buf *qnext; // disk queue
  uchar data[BSIZE];
};
#define B_VALID 0x2 // buffer has been read from disk
#define B_DIRTY 0x4 // buffer needs to be written to disk
```