LECTURE

Flash

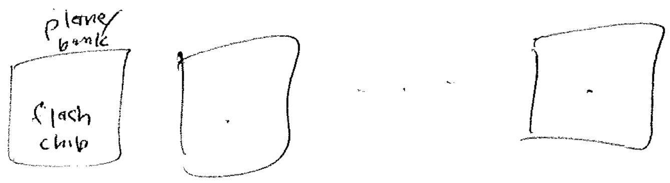
(NAND) Flash

(refs 7, 16, 17)

→ UCSD paper (Micro 2009)
→ DFTL (ASPLOS '09)

data: charge trapped in a gate
each gate can store 1/more bits
(SLC) (MLC)

internally:

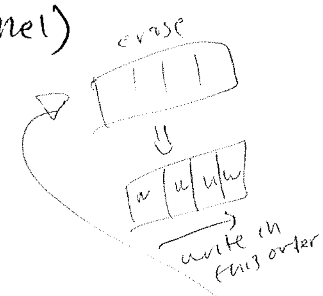


bank:
divided into blocks (size 128k → 256k)

~~Block~~ divided into pages (size ~2k → 8k)
(data + OOB)

three primary operations:
erase (block) → sets all bits to 1
program (page) → can change (some) 1's → 0's
⇒ must erase block before programming page w/in block

read (page) → read whole page (in parallel)



performance: (per page)
read latency ~10 us (10–50)   ~200 MB/s
prog latency ~200 → 500 us   ~
erase ~2 ms

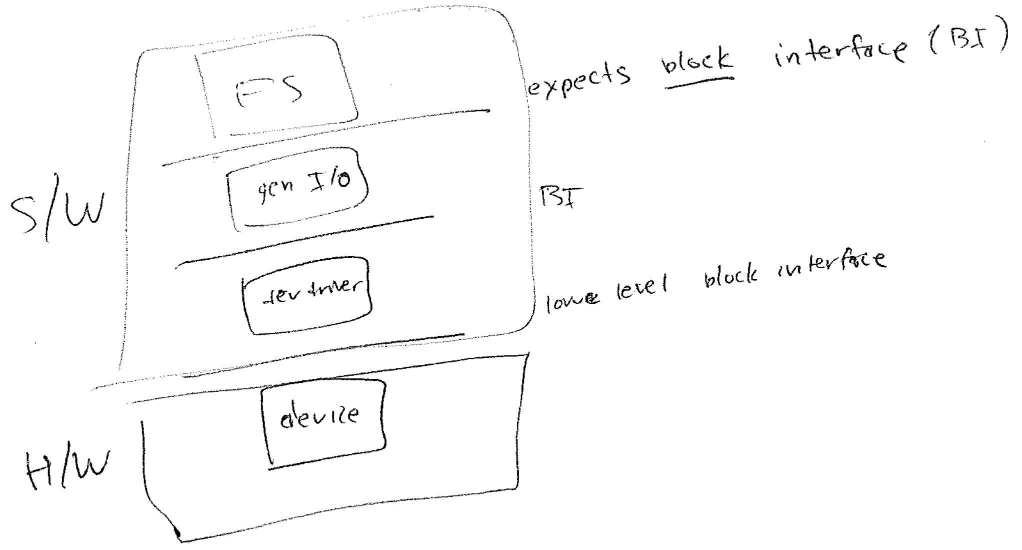
→ no moving parts!
(rare catastophic failure)
→ relatively high bit-error rates (disturb)

reliability:
primary problem: wearout
erase/program cycle → trapped charge → can't tell difference between 1/0
for each block: 10K P/E cycles (MLC)
100K P/E cycles (SLC)

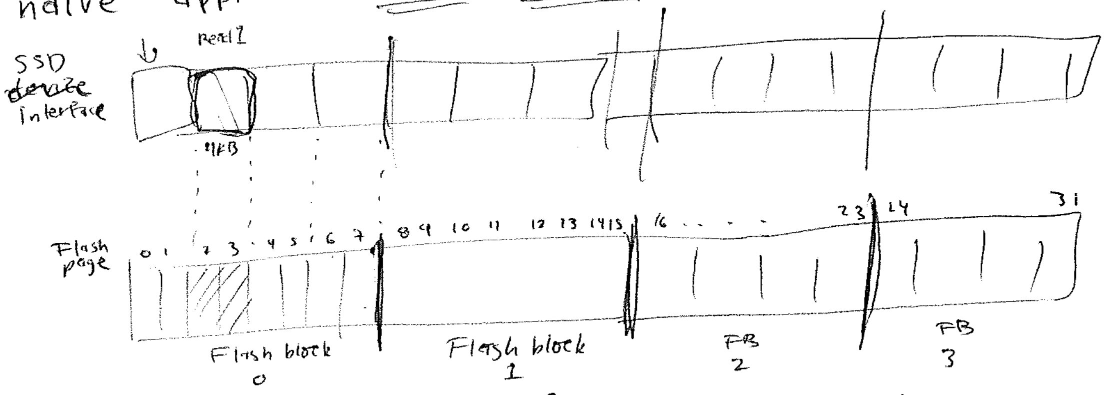
also, program disturb ⇒ when prog one page, neighbor may be affected
⇒ program in order (once written, only one strong disturb will happen)

ugly partial-page prog
→ data rentention @ high temp

how to use in system?



naive approach: direct mapping



read $\overset{d}{\underline{B1}}$ ⇒ read $\overset{fp}{(2,3)}$ ⇒ works!

write $\underline{d1}$ ⇒ erase (b0), program (2,3)

bad why? ⇒ performance (each erase ~ 2 ms)

⇒ reliability (wear out!)

(prog. disturb)

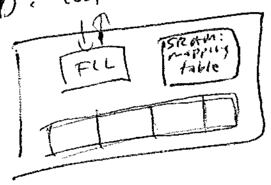
more sophisticated/real: flash translation layer (FTL)

SSD: looks like disk

takes read/write blocks

⇓

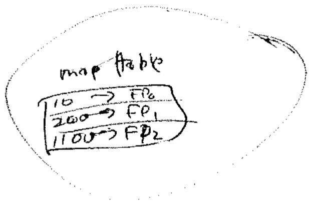
e, p, r of blocks/page

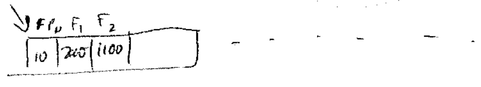


simple page mapping

write 10
write 200
write 1100

⇓

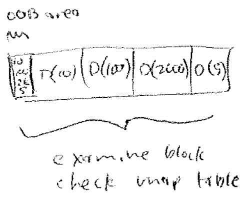




page-mapped
Implications of FTL:
→ can erase blocks in background — performance ⇒ good! } perf
→ fill in log-structured style ⇒ helps w/ wear leveling (reliability)

⇒ generates garbage
→ device must find old blocks, clean, make available again
→ device must also make sure to <u>wear level</u> erase/program blocks } reliability

device tracks:
live/dead : how? ⇒
erasure count / block : how? ⇒



(must periodically clean block for wear too!)

Problem: mapping table is too big! (costly, energy)
compute size : 200 GB device w/ 2K pages
100 m entries * 4 MB ⇒ $\boxed{400\ MB}$ in device!

solutions:
zoning (low-end) ⇒ swap table per zone in/out mem
(keep most ~~tables~~ tables in flash)

Block size = 4

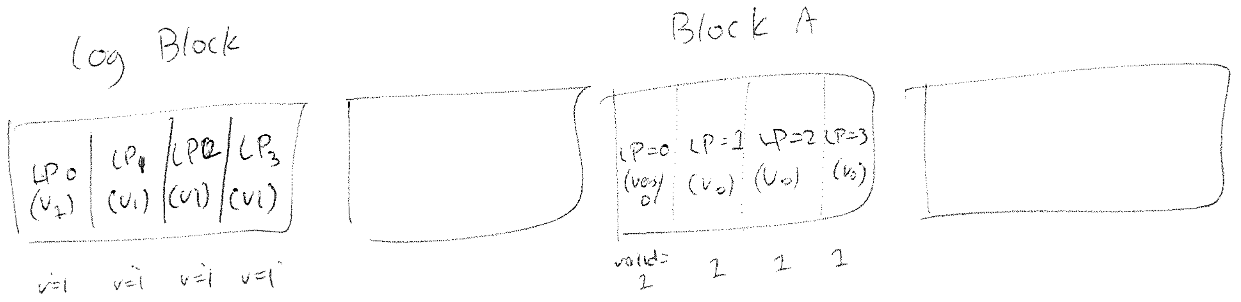
Page Num
0 1 2 3 | 4 5 6 7
BN=0 BN=1
BN = PN / block size,
Boff = PN % Block size,

larger mappings (not used) ⇒ makes table smaller: @ what cost?
[program of small page ⇒ read block, erase some new block, write block (+ new page)
<u>(RMW cycle)</u> ⇒ costly

common : <u>hybrid</u> (both page-level + block-level)
most of SSD : Data blocks (block mapped) ≥ 95%
small, active write area : Log blocks (page mapped) < 5%

tricky part : <u>GC</u>
switch merge : after seq write ⇒ log, switch log + dest block

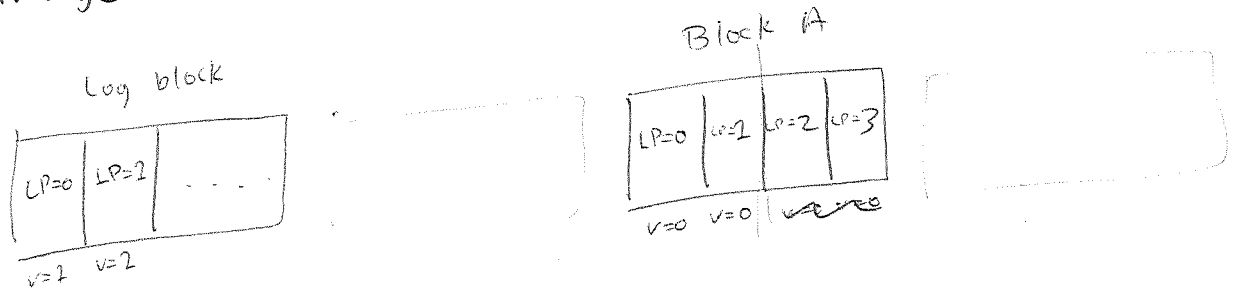
switch merge



make A ⇒ log block
(erase it)

make L ⇒ data block
(update mapping)

partial merge



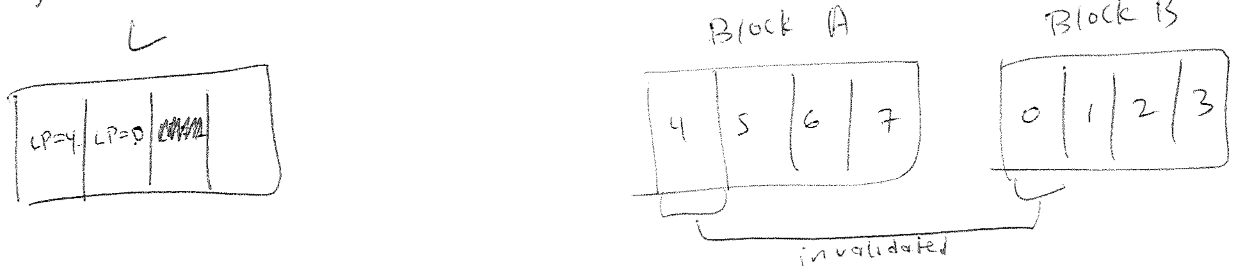
read 2,3 from A

write (prog.) to L

call L → A

call A → L

full merge



⇒ find new erased C

⇒ merge 0(L) + (1,2,3)B
⇒ C
(erase B) ⇒ L

⇒ merge $(5,6,7)_A$ ⇒ L
(erase A)

end result:

SSDs:

- seq r/w ⇒ good (100s of MB/s)
- rand read ⇒ good (nearly seq)
- rand write ⇒ bad (lots of GC + merging)

(FS implications?)