RAID

Redundant Array of Inexpensive* Disks

* In industry, "inexpensive" has been replaced by "independent" :-)

E Pluribus Unum

- Implement the abstraction of a faster, bigger and more reliable disk using a collection of slower, smaller, and more likely to fail disks
  - different configurations offer different tradeoffs
- Key feature: transparency
  - The Power of Abstraction™
  - to the OS looks like a single, large, highly performant and highly reliable single disk (a SLED, hopefully with lower-case "e"!)
    - a linear array of blocks
    - mapping needed to get to actual disk
    - cost: one logical I/O may translate into multiple physical I/Os
- In the box:
  - microcontroller, DRAM (to buffer blocks) [sometimes non-volatile memory, parity logic]

Failure Model

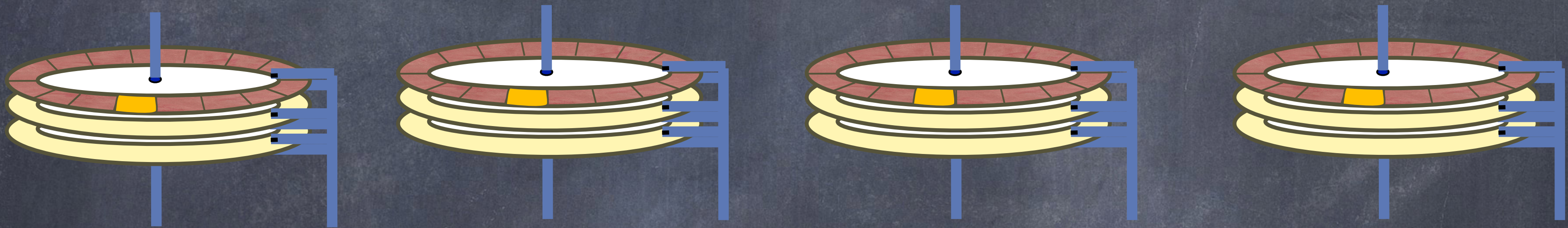
- RAID adopts the strong, somewhat unrealistic Fail-Stop failure model (electronic failure, wear out, head damage)
  - component works correctly until it crashes, permanently
    - disk is either working: all sectors can be read and written
    - or has failed: it is permanently lost
  - failure of the component is immediately detected
    - RAID controller can immediately observe a disk has failed and accesses return error codes
- In reality, disks can also suffer from isolated sector failures
  - Permanent: physical malfunction (magnetic coating, scratches, contaminants)
  - Transient: data is corrupted, but new data can be successfully read from/written to sector

How to Evaluate a RAID

- Capacity
  - what fraction of the sum of the storage of its constituent disks does the RAID make available?
- Reliability
  - How many disk faults can a specific RAID configuration tolerate?
- Performance
  - Workload dependent

RAID-0: Striping

Spread blocks across disks using round robin



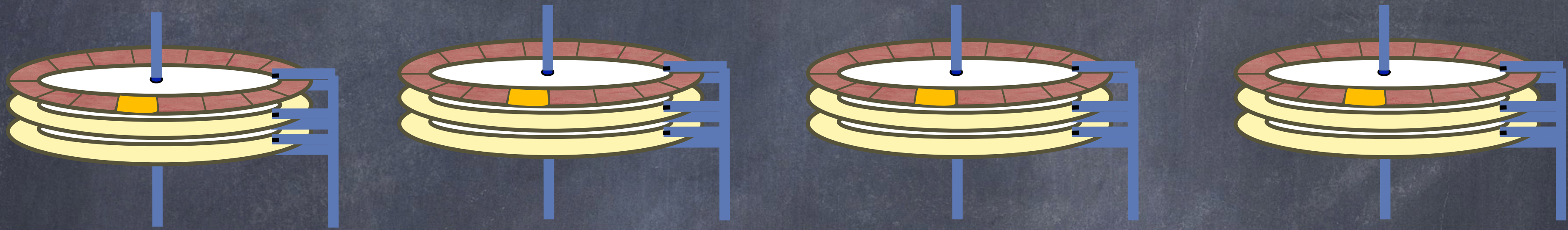
Stripe	0	1	2	3
Stripe	4	5	6	7
Stripe	8	9	10	11
Stripe	12	13	14	15

+ Excellent parallelism
    - can read/write from multiple disks

– Worst-case positioning time
    - wait for largest across all disks

RAID-0: Striping (Big Chunk Edition)

Spread blocks across disks using round robin



Stripe	0	2	4	6
	1	3	5	7
Stripe	8	10	12	14
	9	11	13	15

\+ improve positioning time — decrease parallelism

RAID-0: Evaluation

- Capacity
  - Excellent: N disks, each holding B blocks support the abstraction of a single disk with NxB blocks
- Reliability
  - Poor: Striping reduces reliability
    - Any disk failure causes data loss
- Performance
  - Workload dependent, of course
  - We'll consider two workloads
    - Sequential: single disk transfers S MB/s
    - Random: single disk transfer R MB/s
    - S >> R

RAID-0: Performance

- Single-block read/write throughput
  - about the same as accessing a single disk
- Latency
  - Read: T ms (latency of one I/O op to disk)
  - Write: T ms
- Steady-state read/write throughput
  - Sequential: N x S MB/s
  - Random: N x R MB/s

RAID-1: Mirroring

Each block is replicated twice

0	0	1	1
2	2	3	3
4	4	5	5
6	6	7	7

Read from any

Write to both

RAID-1: Evaluation

- Capacity
  - Poor: N disks of B blocks yield (N x B)/2 blocks
- Reliability
  - Good: Can tolerate the loss (not corruption!) of any one disk
- Performance
  - Fine for reads: can choose any disk
  - Poor for writes: every logical write requires writing to both disks
    - suffers worst seek+rotational delay of the two writes

RAID-1: Performance

- Steady-state throughput
  - Sequential Writes: N/2 x S MB/s
    - Each logical Write involves two physical Writes
  - Sequential Reads: as low as N/2 x S MB/s

0	0	1	1
2	2	3	3
4	4	5	5
6	6	7	7

Suppose we want to read
0, 1, 2, 3, 4, 5, 6, 7

RAID-1: Performance

- Steady-state throughput
  - Sequential Writes: N/2 x S MB/s
    - Each logical Write involves two physical Writes
  - Sequential Reads: as low as N/2 x S MB/s

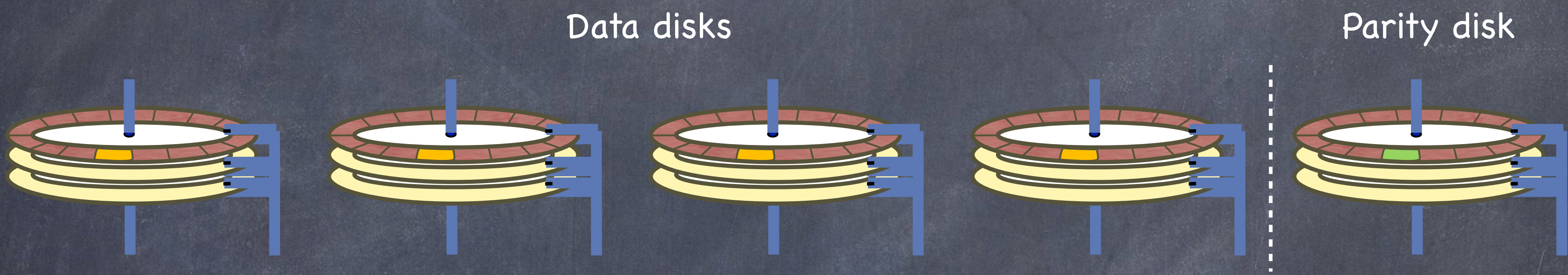
0	0	1	1
2	2	3	3
4	4	5	5
6	6	7	7

Suppose we want to read
0, 1, 2, 3, 4, 5, 6, 7

Each disk only delivers half of his bandwidth:
half of its blocks are skipped!

  - Random Writes: N/2 x R MB/s
    - Each logical Write involves two physical Writes
  - Random Reads: N x R MB/s
    - Reads can be distributed across all disks
- Latency for Reads and Writes: T ms

RAID-4: Block Striped, with Parity



Stripe	0	1	2	3	P0
Stripe	4	5	6	7	P1
Stripe	8	9	10	11	P2
Stripe	12	13	14	15	P3

1	1	0
0	1	0
0	0	1

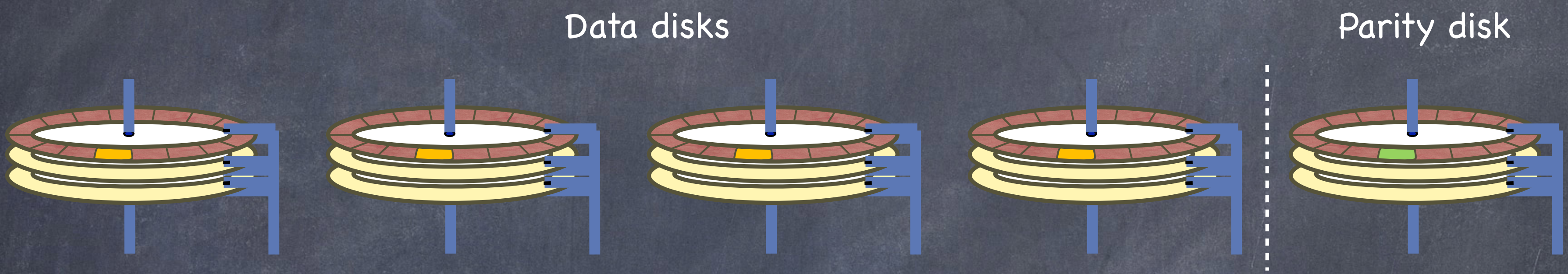
1	0	0
1	1	0
0	1	1

1	0	0
0	1	0
1	0	1

1	1	0
1	1	1
0	0	1

0	0	0
0	0	1
1	1	0

RAID-4: Block Striped, with Parity



Stripe 0	1	2	3	P0
Stripe 4	5	6	7	P1
Stripe 8	9	10	11	P2
Stripe 12	13	14	15	P3

1	1	0
0	1	0
0	0	1

1	0	0
1	1	0
0	1	1

1	0	0
0	1	0
1	0	1

1	1	0
1	1	1
0	0	1

0	0	0
0	0	1
1	1	0

Disk controller can identify faulty disk

- single parity disk can detect and correct errors

RAID-4: Evaluation

- Capacity
  - N disks of B blocks yield (N-1) x B blocks
- Reliability
  - Tolerates the failure of any one disk
- Performance
  - Fine for sequential read/write accesses and random reads
  - Random writes are a problem!

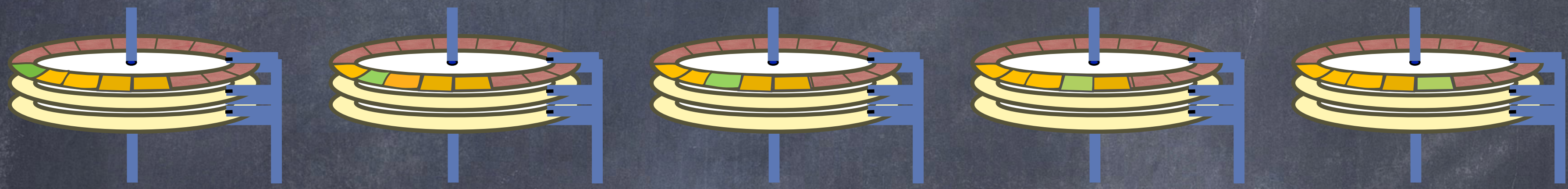
RAID-4: Performance

- Sequential Reads: (N-1) x S MB/s
- Sequential Writes: (N-1) x S MB/s
  - compute & write parity block once for the full stripe
- Random Read: (N-1) x R MB/s
- Random Writes: R/2 MB/s (N is gone! Yikes!)
  - need to read block from disk and parity block
  - Compute $P_{new} = (B_{old} \text{ XOR } B_{new}) \text{ XOR } P_{old}$
  - Write back B_{new} and P_{new}
  - Every write must go through parity disk, eliminating any chance of parallelism
  - Every logical I/O requires two physical I/Os at parity disk: can at most achieve 1/2 of its random transfer rate (i.e. R/2)

- Latency: Reads: T ms; Writes: 2T ms

RAID-5: Rotating Parity (avoids the bottleneck)

Parity and Data distributed across all disks



0	1	2	3	P0
5	6	7	P1	4
10	11	P2	8	9
15	P3	12	13	14
P4	16	17	18	19

RAID-5: Evaluation

- Capacity & Reliability
  - As in Raid-4
- Performance
  - Sequential read/write accesses as in RAID-4
    - (N-1) x S MB/s
  - Random Reads are slightly better
    - N x R MB/s (instead of (N-1) x R MB/s)
  - Random Writes much better than RAID-4: R/2 x N/2
    - as in RAID-4 writes involve two operations at every disk: each disk can achieve at most R/2
    - but, without a bottleneck parity disk, we can issue up to N/2 writes in parallel (each involving 2 disks)

SSDs

Why care?

HDD

- Require seek, rotate, transfer on each I/O
- Not parallel (one active head)
- Brittle (moving parts)
- Slow (mechanical)
- Poor random I/O (10s of ms)

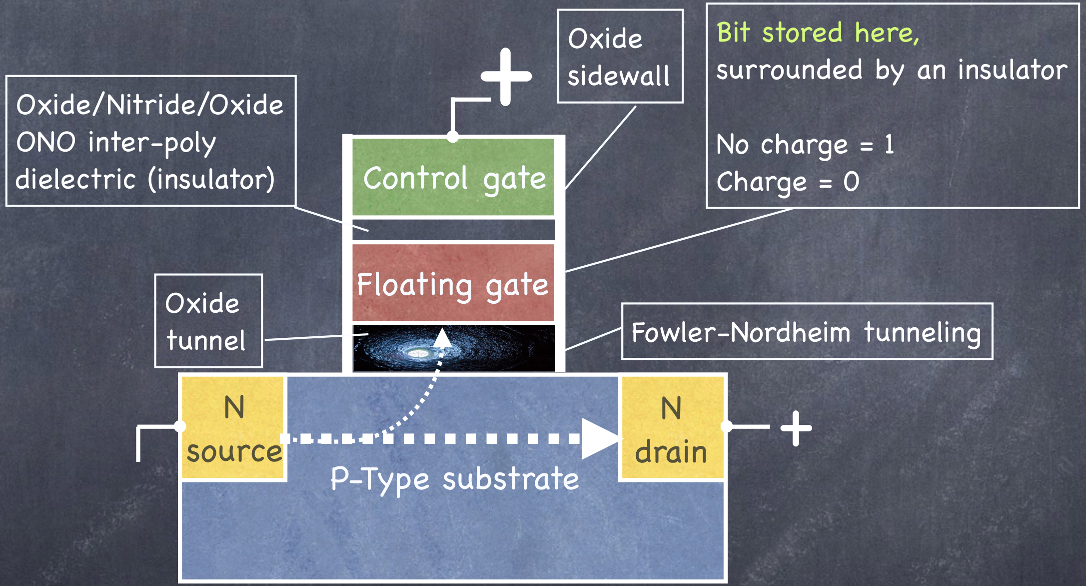
SSD

- No seeks
- Parallel
- No moving parts
- Random reads take 10s of μs
- Wears out!

Flash Storage

To write 0

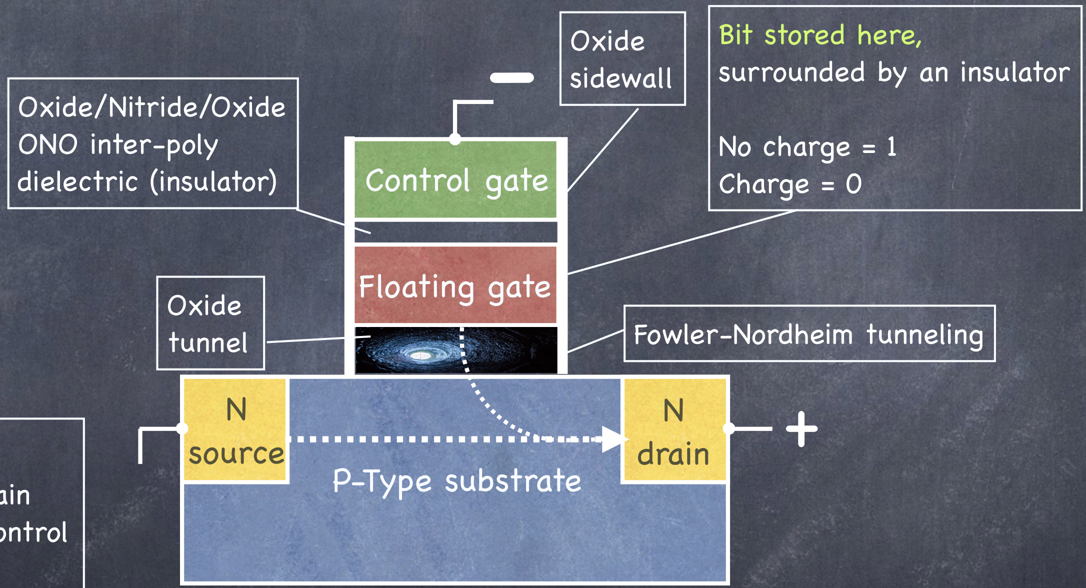
- apply positive voltage to drain
- apply even stronger positive voltage to control gate
- some electrons are tunneled into floating gate



Flash Storage

To write 0

- apply positive voltage to drain
- apply even stronger positive voltage to control gate
- some electrons are tunneled into floating gate



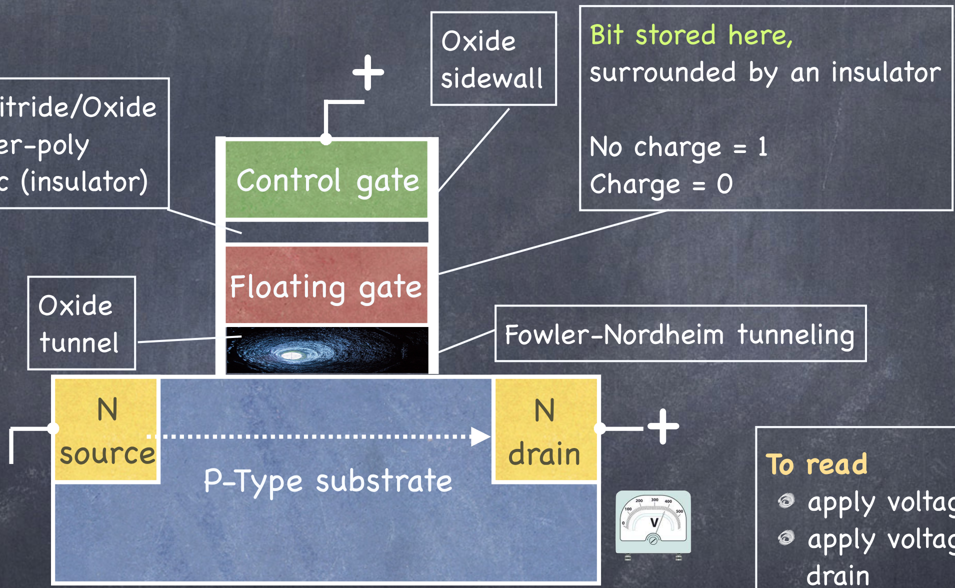
To write 1

- apply positive voltage to drain
- apply negative voltage to control gate
- electrons are forced out of floating gate into source

Flash Storage

To write 0

- apply positive voltage to drain
- apply even stronger positive voltage to control gate
- some electrons are tunneled into floating gate



To write 1

- apply positive voltage to drain
- apply negative voltage to control gate
- electrons are forced out of floating gate into source

To read

- apply voltage to control gate
- apply voltage across source and drain
- measure current between source and drain to determine whether electrons in gate
  - if electrons in floating gate, must apply higher voltage ato control gate to have current
  - measured current can encode more than a single bit

The Cell

- Single-level cells
  - faster, more lasting (50K to 100K program/erase cycles), more stable
  - 0 means charge; 1 means no charge
- Multi-level cells
  - can store 2, 3, even 4 bits
  - cheaper to manufacture
  - wear out faster (1k to 10K program/erase cycles)
  - more fragile (stored value can be disturbed by accesses to nearby cells)

The SSD Storage Hierarchy

Cell
1 to 4 bits

Page
2 to 8 KB
not to be confused with a VM page

Block
64 to 256 pages
not to be confused with a disk block

Plane/Bank
Many blocks (Several Ks)

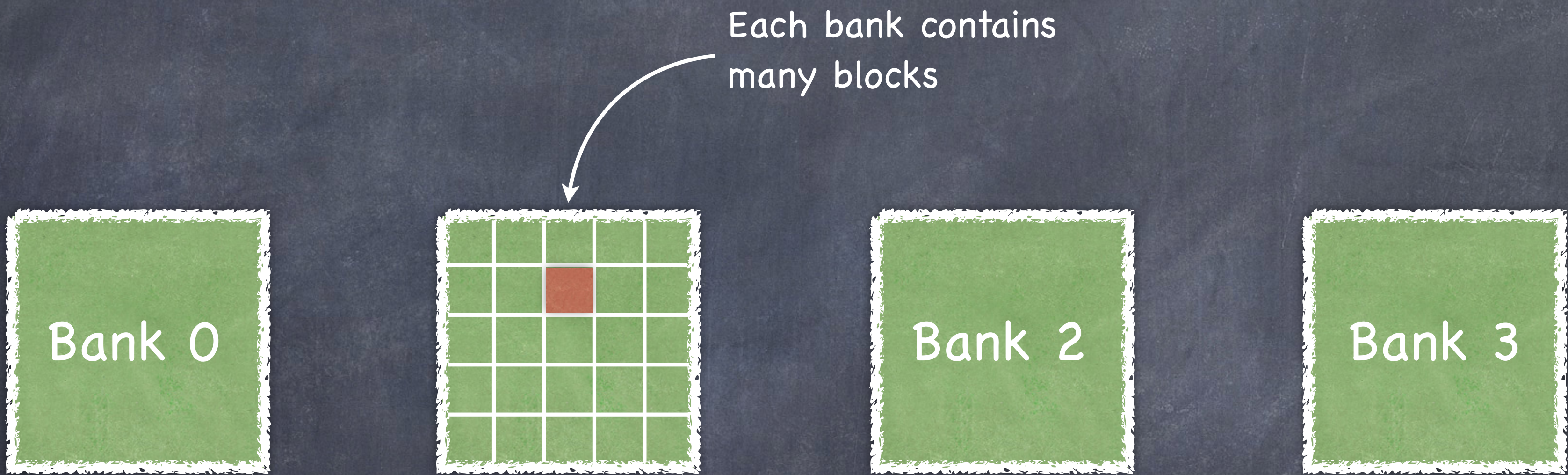
Flash Chip
Several banks that can be accessed in parallel

Basic Flash Operations

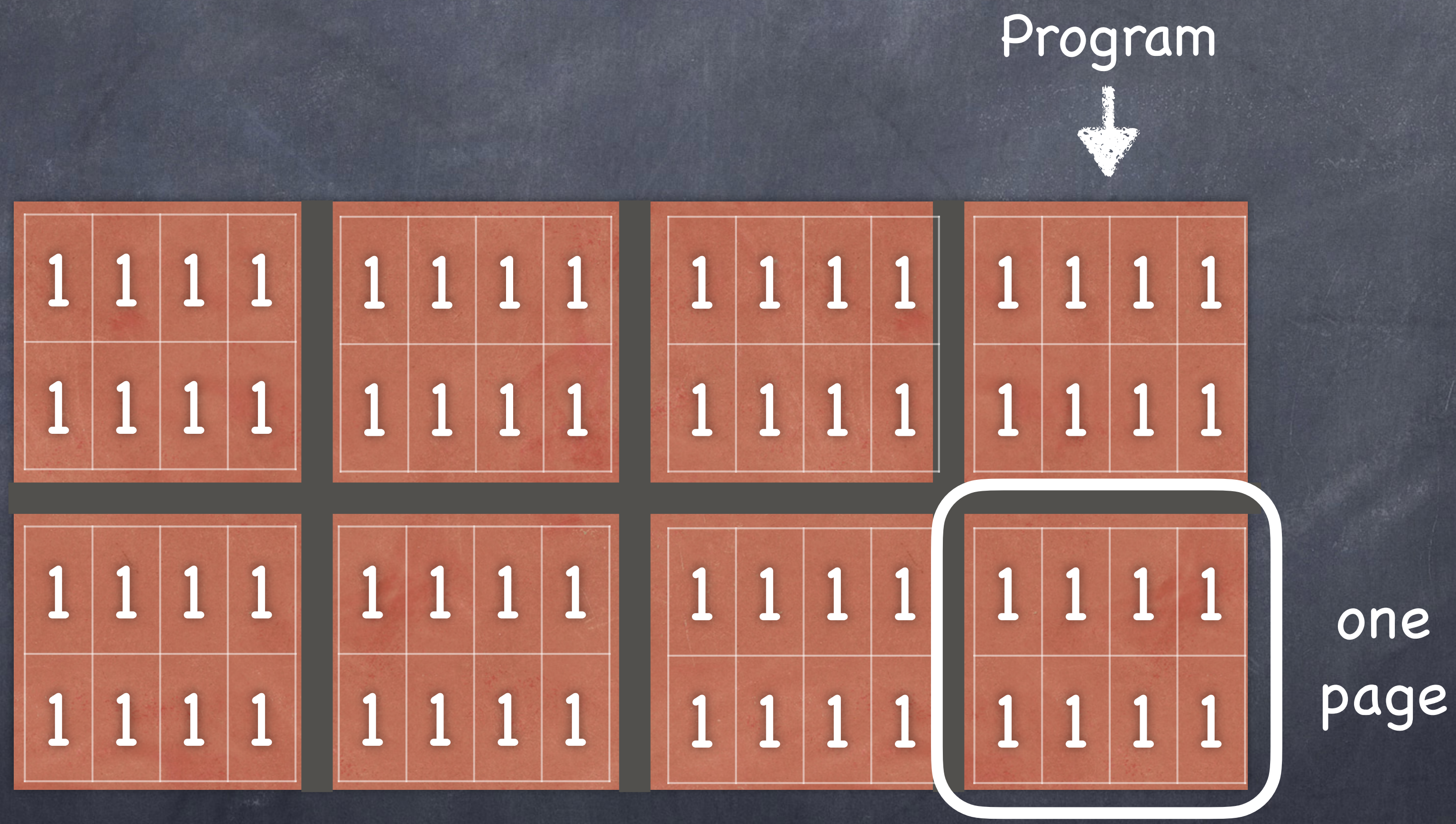
- Read (a page)
  - 10s of μs, independent of the previously read page
    - great for random access!
- Erase (a block)
  - sets the entire block (with all its pages) to 1 (!)
  - very coarse way to write 1s...
  - 1.5 to 2 ms (on a fast SLC)
- Program (a page)
  - can change some bits in a page of an erased block to 0
  - 100s of μs
  - changing a 0 bit back to 1 requires erasing the entire block!

Banks

Banks

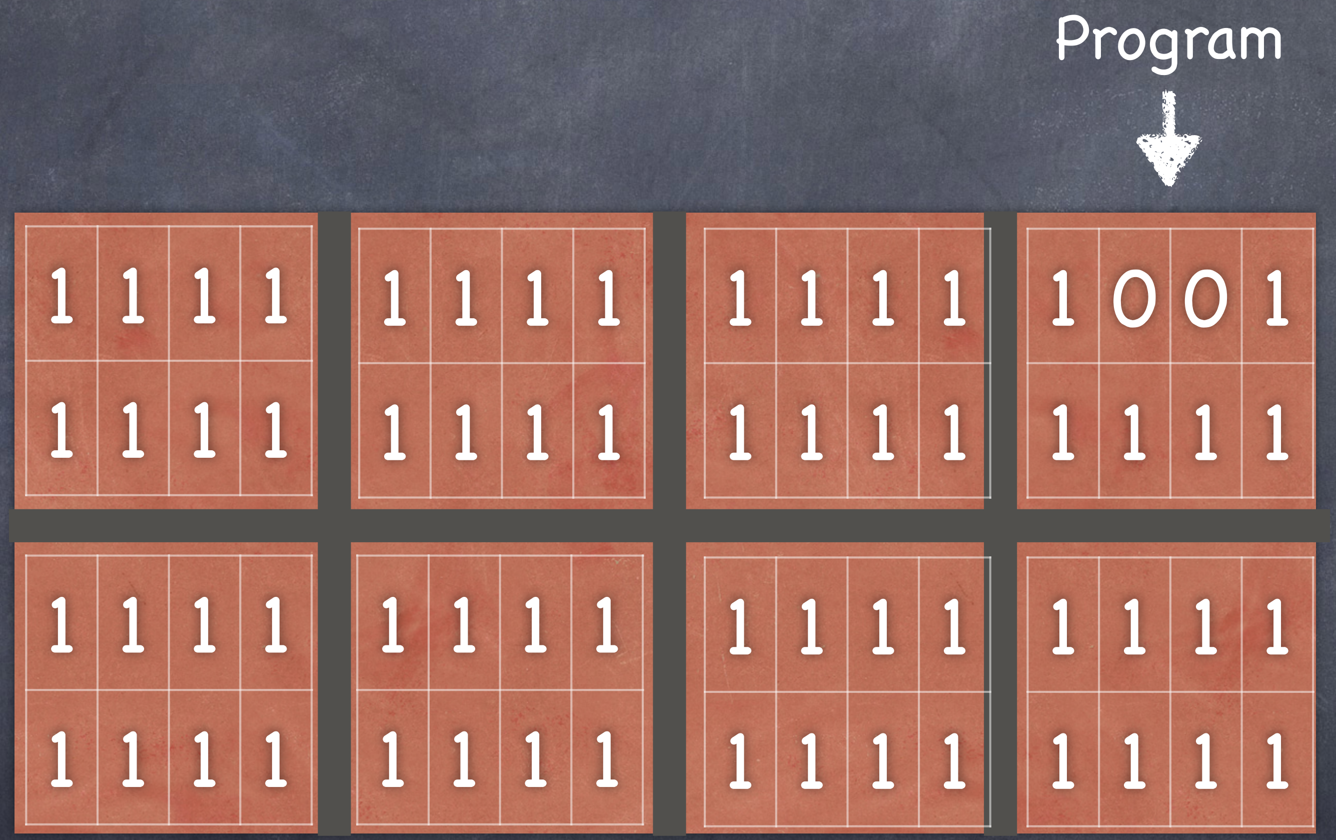


Block

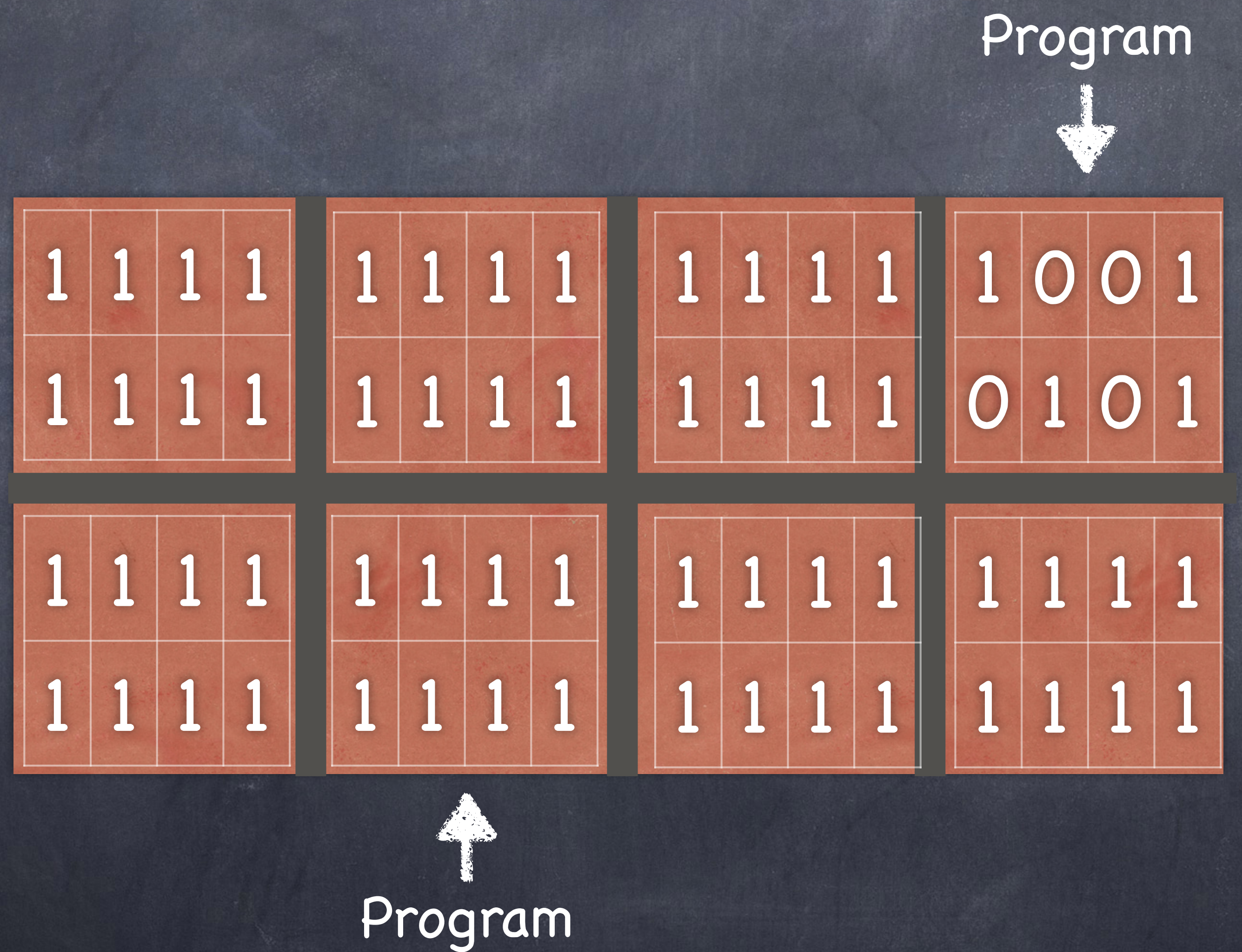


After an Erase, all cells are discharged (i.e., store 1s)

Block

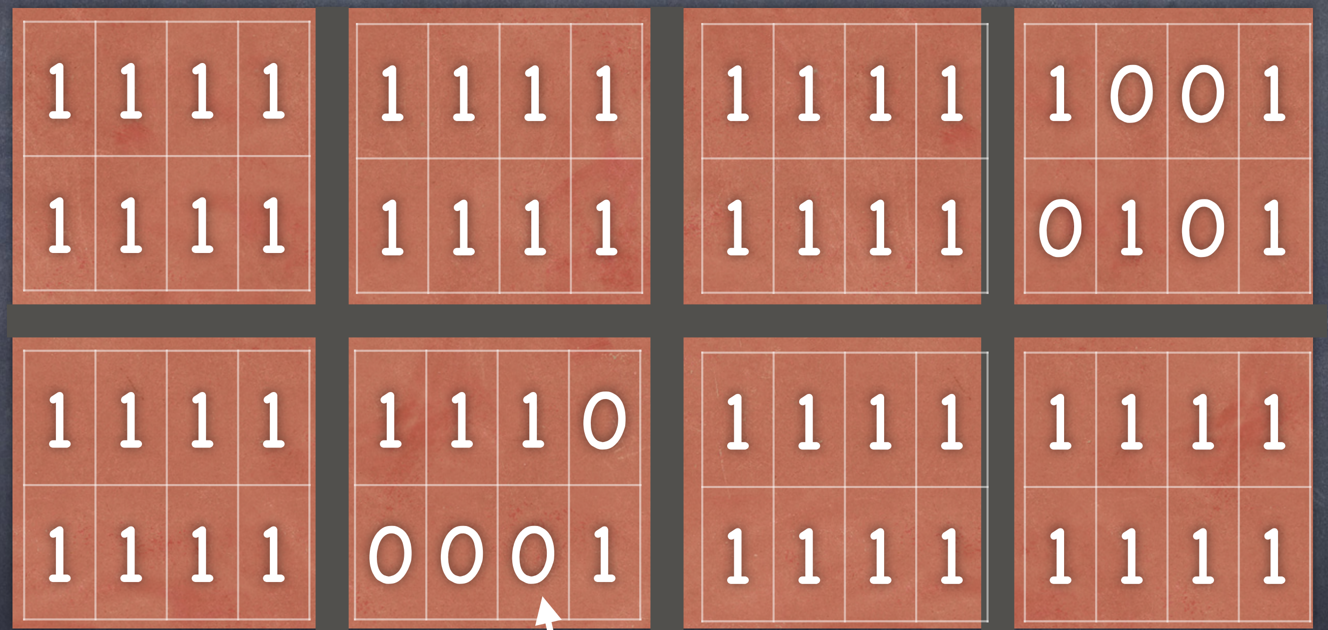

Program
1 1 1 1
1 1 1 1
1 1 1 1
1 0 0 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1

Block


Program
1 1 1 1
1 1 1 1
1 1 1 1
1 0 0 1
1 1 1 1
1 1 1 1
1 1 1 1
0 1 0 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
Program

Block

Erase (!)

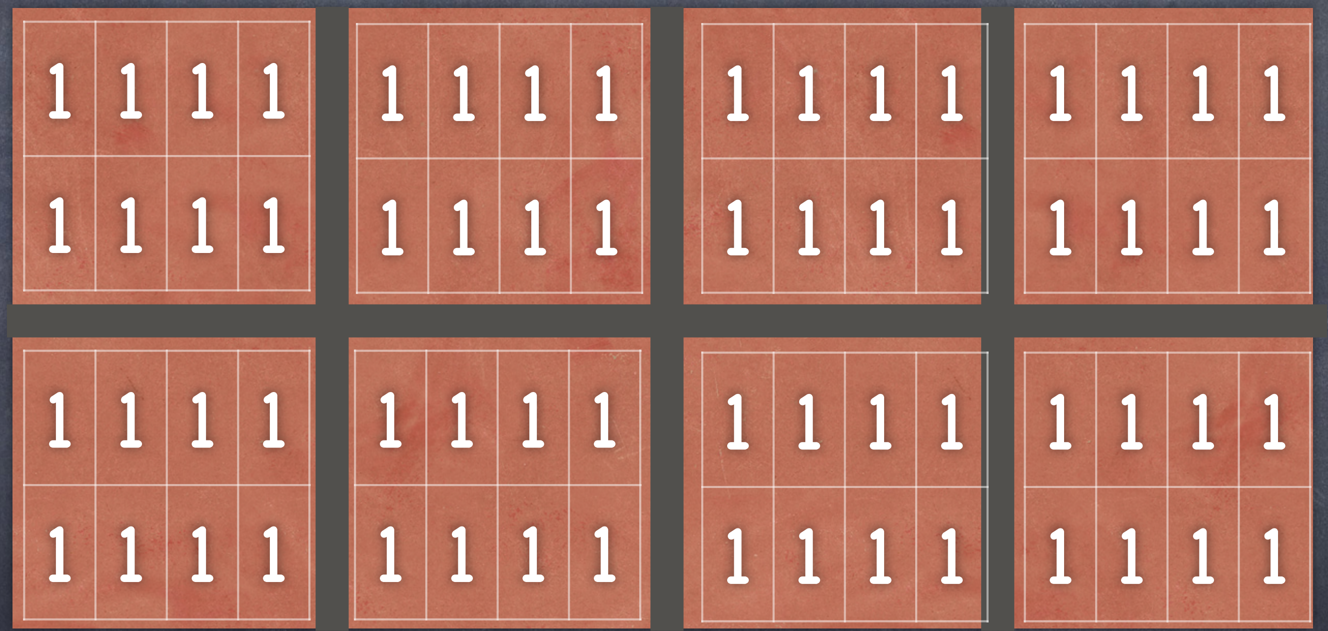


If now we want to set this bit to 1, we need to erase the entire block!

Modified pages must be copied elsewhere, or lost!

Block

Erase



Wear Out

Every erase/program cycle adds some charge to a block; over time, hard to distinguish 1 from 0!

APIs

	HDD	Flash
read	read sector	read page
write	write sector	program page (0's) erase block (1's)

Performance

	HDD	Flash
Throughput	≈ 130MB/s (sequential)	≈200MB/s (random or sequential)
Latency	≈ 10ms	read 25μs program 200-300μs erase 1.5-2 ms

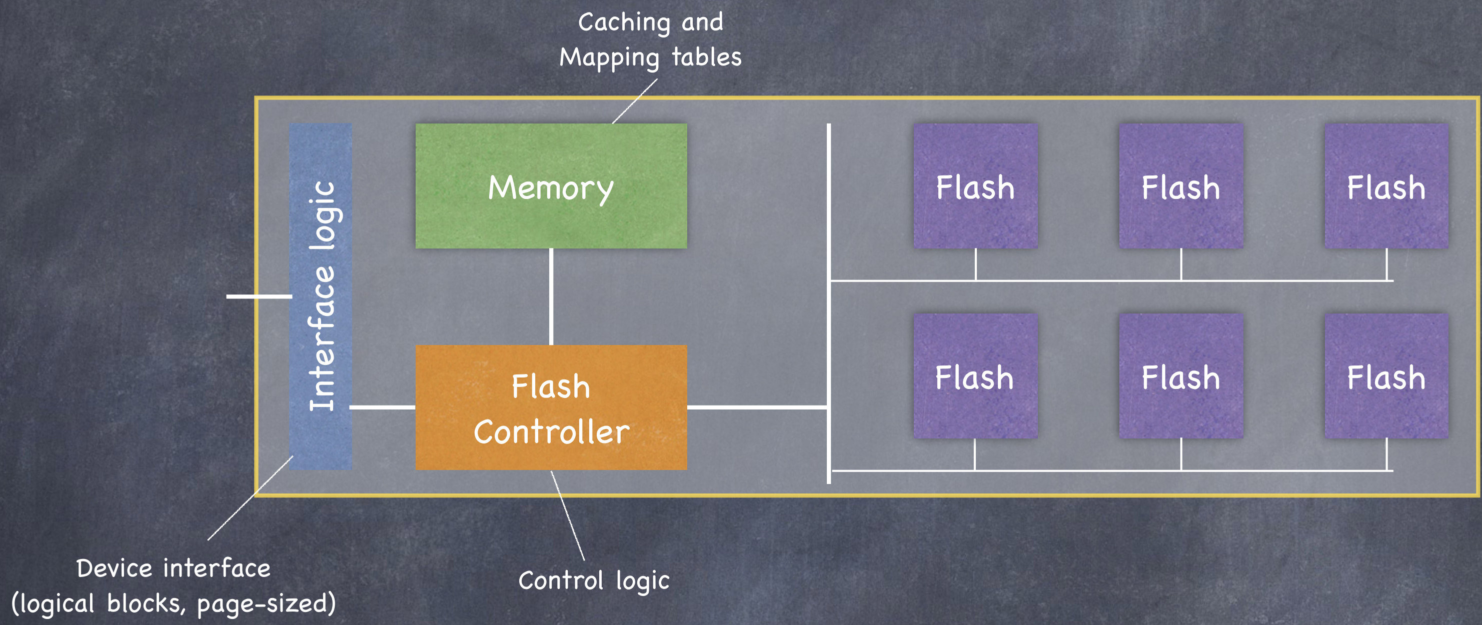
Using Flash Memory

- Need to map reads and writes to logical blocks to read, program, and erase operations on flash

↓

Flash Translation Layer (FTL)

From Flash to SSD



- Flash Translation Layer
  - tries to minimize
    - write amplification: $\left[\frac{\text{write traffic (bytes) to flash chips}}{\text{write traffic (bytes) from client to SSD}}\right]$
    - wear out: practices wear leveling
    - disturbance: writes pages in a block in order, low to high

FTL through Direct Mapping

- Just map logical disk block i to physical page i
  - reads are fine (yahoo!)
  - write to logical block i, however, involves
    - reading the (physical) block where physical page i lives
    - erasing the block
    - (re)programming old pages as well as new page i
- Severe write amplification
  - writes are slow!
- Poor wear leveling
  - pages corresponding to "hot" logical block experience disproportionate number of erase/program cycles

FT… ect Mappi…

- …nap… …isk block i to physical pa…
  - …eads are
  - write to logica… …volves
    - reading the (phy… …here physical page i li…
    - erasing the block
    - programming old pages a… …page i
- …re write amplification
  - …s are slow!
- Po… …eveling
  - pag… …to "hot" … …periences disprop… …am cycles

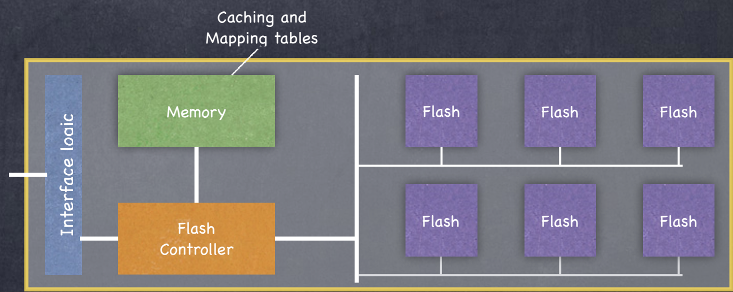
Log Structured FTL

- Think of flash storage as implementing a log

Log Structured FTL

- Think of flash storage as implementing a log

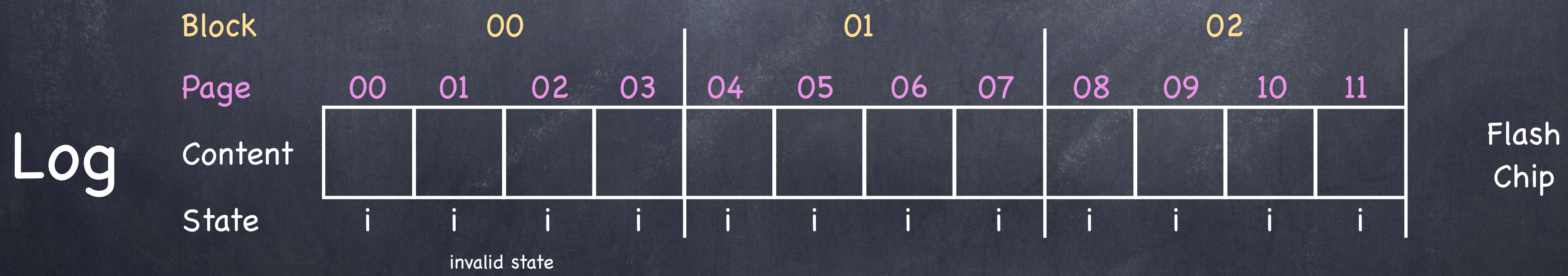
- On a write, program next available page of physical block being currently written
  - i.e., "append" the write to your log
- On a read, find in the log the page storing the logical block
  - don't want to scan the whole log...
  - keep an in-memory map from logical blocks to pages!



Example

- SSD's clients read/write 4KB logical blocks
- Many physical SSD blocks; each holds 4 pages, each 4KB

A logical block maps to a physical page



- Client operations { Write (a1, 100)

1) Erase(00)

Example

- SSD's clients read/write 4KB logical blocks
- Many physical SSD blocks; each holds 4 pages, each 4KB

A logical block maps to a physical page

Block	00				01				02			
Page	00	01	02	03	04	05	06	07	08	09	10	11
Content												
State	E	E	E	E	i	i	i	i	i	i	i	i

Flash Chip

- Client operations { Write (a1, 100)

2) Program(00)

Example

- SSD's clients read/write 4KB logical blocks
- Many physical SSD blocks; each holds 4 pages, each 4KB

A logical block maps to a physical page

Table 100 → 00 — Memory

Block	00				01				02			
Page	00	01	02	03	04	05	06	07	08	09	10	11
Content	a1											
State	V	E	E	E	i	i	i	i	i	i	i	i

Flash Chip

- Client operations: Write (a1, 100)

Example

- SSD's clients read/write 4KB logical blocks
- Many physical SSD blocks; each holds 4 pages, each 4KB

A logical block maps to a physical page

Table 100 → 00 — Memory

Block	00				01				02			
Page	00	01	02	03	04	05	06	07	08	09	10	11
Content	a1											
State	V	E	E	E	i	i	i	i	i	i	i	i

Flash Chip

- Client operations
  - Write (a1, 100)
  - Write (a2, 101)

3) Program(01)

Example

- SSD's clients read/write 4KB logical blocks
- Many physical SSD blocks; each holds 4 pages, each 4KB

A logical block maps to a physical page

Table 100 → 00 101 → 01 Memory

Block	00				01				02				
Page	00	01	02	03	04	05	06	07	08	09	10	11	
Content	a1	a2											Flash Chip
State	V	V	E	E	i	i	i	i	i	i	i	i	

- Client operations
  - Write (a1, 100)
  - Write (a2, 101)

Example

- SSD's clients read/write 4KB logical blocks
- Many physical SSD blocks; each holds 4 pages, each 4KB

A logical block maps to a physical page

Table 100 → 00 101 → 01 2000 → 02 2001 → 03 Memory

Block	00				01				02				
Page	00	01	02	03	04	05	06	07	08	09	10	11	
Content	a1	a2	b1	b2									Flash Chip
State	V	V	V	V	i	i	i	i	i	i	i	i	

- Client operations
  - Write (a1, 100)
  - Write (a2, 101)
  - Write (b1, 2000)
  - Write (b2, 2001)

Example

- SSD's clients read/write 4KB logical blocks
- Many physical SSD blocks; each holds 4 pages, each 4KB

A logical block maps to a physical page

Table 100 → 00 101 → 01 2000 → 02 2001 → 03 — Memory

Block	00				01				02			
Page	00	01	02	03	04	05	06	07	08	09	10	11
Content	a1	a2	b1	b2								
State	V	V	V	V	i	i	i	i	i	i	i	i

Flash Chip

- Client operations { Write (c1, 100)

Erase(01)

Example

- SSD's clients read/write 4KB logical blocks
- Many physical SSD blocks; each holds 4 pages, each 4KB

A logical block maps to a physical page

Table 100 → 00 101 → 01 2000 → 02 2001 → 03 Memory

Block	00				01				02			
Page	00	01	02	03	04	05	06	07	08	09	10	11
Content	a1	a2	b1	b2								
State	V	V	V	V	E	E	E	E	i	i	i	i

Flash Chip

- Client operations { Write (c1, 100)

Program(04)

Example

- SSD's clients read/write 4KB logical blocks
- Many physical SSD blocks; each holds 4 pages, each 4KB

A logical block maps to a physical page

Table	100 → 00	101 → 01	2000 → 02	2001 → 03	Memory

Block	00				01				02				
Page	00	01	02	03	04	05	06	07	08	09	10	11	
Content	a1	a2	b1	b2	c1								Flash Chip
State	V	V	V	V	V	E	E	E	i	i	i	i	

- Client operations { Write (c1, 100)

Example

- SSD's clients read/write 4KB logical blocks
- Many physical SSD blocks; each holds 4 pages, each 4KB

A logical block maps to a physical page

Table 100 → 04 101 → 01 2000 → 02 2001 → 03 Memory

Block	00				01				02				
Page	00	01	02	03	04	05	06	07	08	09	10	11	
Content	a1	a2	b1	b2	c1								Flash Chip
State	V	V	V	V	V	E	E	E	i	i	i	i	

- Client operations { Write (c1, 100)

Example

- SSD's clients read/write 4KB logical blocks
- Many physical SSD blocks; each holds 4 pages, each 4KB

A logical block maps to a physical page

Table 100 → 04 101 → 05 2000 → 02 2001 → 03 Memory

Block	00				01				02				
Page	00	01	02	03	04	05	06	07	08	09	10	11	
Content	a1	a2	b1	b2	c1	c2							Flash Chip
State	V	V	V	V	V	V	E	E	i	i	i	i	

- Client operations
  - Write (c1, 100)
  - Write (c2, 101)

Example

- SSD's clients read/write 4KB logical blocks
- Many physical SSD blocks; each holds 4 pages, each 4KB

A logical block maps to a physical page

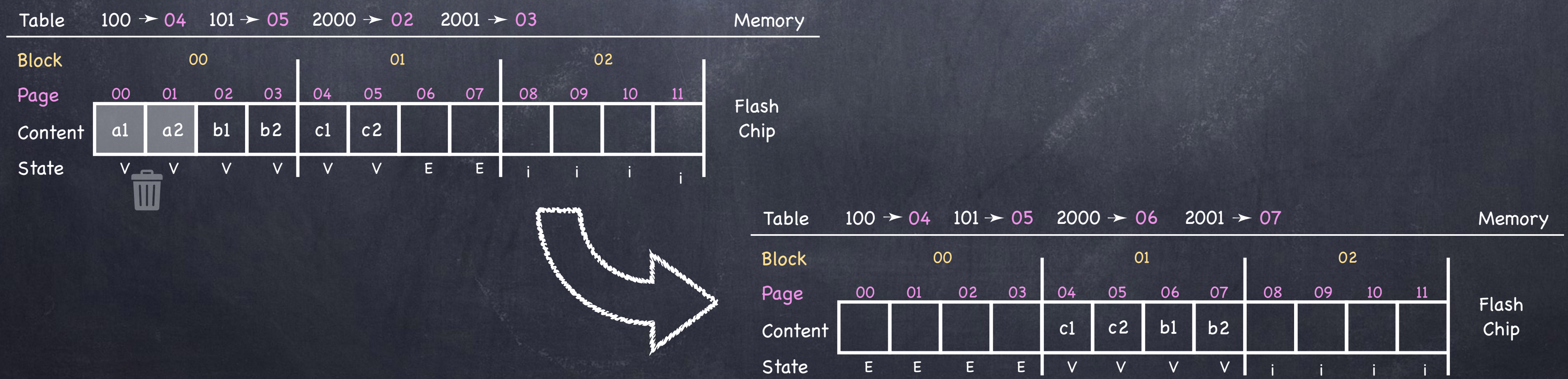
Table 100 → 04 101 → 05 2000 → 02 2001 → 03 Memory

Block	00				01				02				
Page	00	01	02	03	04	05	06	07	08	09	10	11	
Content	a1	a2	b1	b2	c1	c2							Flash Chip
State	V	V	V	V	V	V	E	E	i	i	i	i	

- Client operations
  - Write (c1, 100)
  - Write (c2, 101)

Garbage Collection

- Reclaim dead blocks
  - find a block with garbage pages
  - copy elsewhere the block's live pages
    - use Mapping Table to distinguish live pages from dead
  - make block available for writing again



Shrinking the Mapping Table

- Per-page mapping is memory hungry
  - 1TB SSD, 4KB pages, 4B/MTE: 1GB Mapping Table!

    Mapping Table Entries

Shrinking the Mapping Table

- Per-page mapping is memory hungry
  - 1TB SSD, 4KB pages, 4B/MTE (Mapping Table Entries): 1GB Mapping Table!
- Per-block (physical) mapping? Decreases MT size by $\frac{\text{block size}}{\text{page size}}$
  - The Idea: Divide logical block address space in **chunks** of the size of a physical block

chunk 0 | chunk 1

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

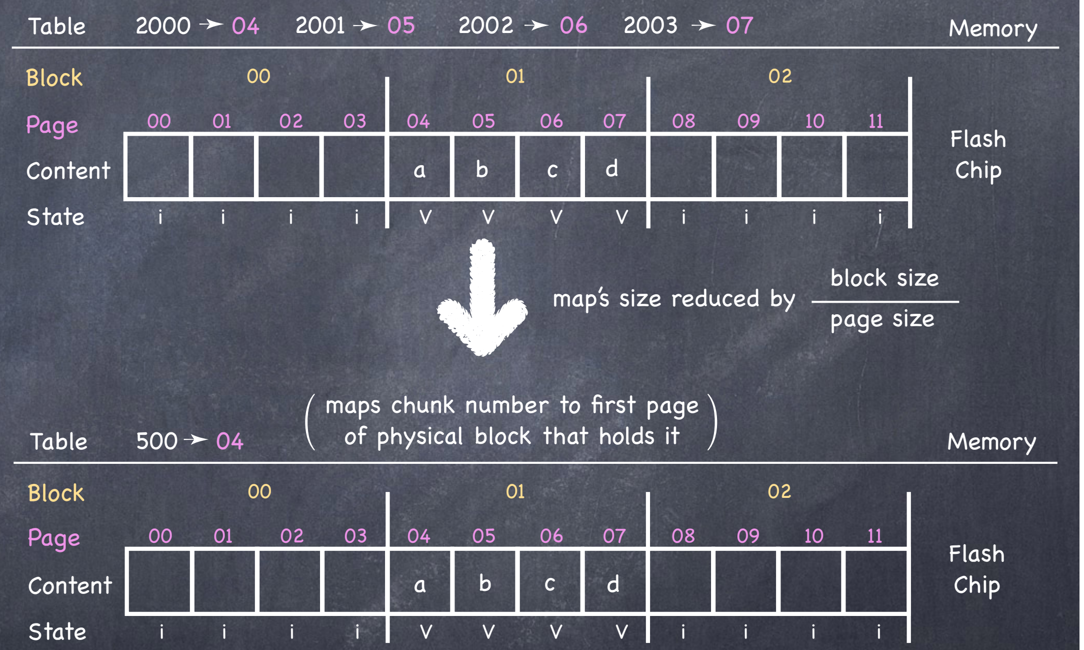
chunk 11 (blocks 40–43)

Logical blocks

- think of logical block address as [chunk #][logical block within chunk]
- E.g., logical block 41: 1 0 1 1 | 0 1 (chunk 11, log. block 1)
  - Map all logical blocks within a chunk C to the same physical block B
    - unlike direct mapping, C can over time map to different Bs (better wear leveling!)

Shrinking the Mapping Table

- Assume every chunk is 4 logical blocks, mapped to some physical block
- Then, to find the location of a logical block L
  - use the high order bits of L' s to determine the chunk C that L belongs to
  - find the physical block B that chunk C is mapped to
  - use least significant bits in L's address to identify the page within B that stores L



To find logical block 2001:

- 2001 div 4 identifies the chunk that holds logical block 2001
- 2001 mod 4 identifies the page within that chunk that holds logical block 2001

Per-block Mapping

- Reading is easy…

Table 500 → 04 | Memory

Block	00				01				02				
Page	00	01	02	03	04	05	06	07	08	09	10	11	
Content					a	b	c	d					Flash Chip
State	i	i	i	i	V	V	V	V	i	i	i	i	

- … but writing a page c′ requires reading in the whole block and writing it elsewhere

Table 500 → 08 | Memory

Block	00				01				02				
Page	00	01	02	03	04	05	06	07	08	09	10	11	
Content									a	b	c′	d	Flash Chip
State	i	i	i	i	E	E	E	E	V	V	V	V	

Hybrid Mapping

- Set aside a few physical blocks to implement log
  - mapped per-page
- Use per-block mapping for the other blocks
- On read
  - search for logical block in Log Table; then go to Data Table (which keeps per-block mapping)
- Periodically, pay the price to copy out content from the log blocks so it can be mapped per block
  - storing contiguous logical blocks in the same physical block may cause write amplification
- For wear leveling, rotate the blocks used for logging

Performance (Throughput)

- Huge difference between SSD and HDD for random I/O
- Not so much for sequential I/O
- On SSDs
  - sequential still better than random
    - FS design tradeoffs for HDD still apply
  - sequential reads perform better than writes
    - sometimes you have to erase
  - random writes perform much better than random reads
    - log transform random accesses into sequential accesses!

Device	Random		Sequential	
	Reads (MB/s)	Writes (MB/s)	Reads (MB/s)	Writes (MB/s)
Samsung 840Pro SSD	103	287	421	384
Seagate 600 SSD	84	252	424	374
Intel SSD 335 SSD	39	222	344	354
Seagate Savvio 15K.3 HDD	2	2	223	223