P&S Modern SSDs

Advanced NAND Flash Commands & Address Translation

Dr. Jisung Park
Prof. Onur Mutlu
ETH Zürich
Spring 2022
4 April 2022

Recap: SSD & NAND Flash Memory

- SSD organization
  - SSD controller: Multicore CPU + per-channel flash controllers
  - DRAM: Metadata store, 0.1% of SSD capacity
  - NAND flash chips
    - Channel (Package(s)) – Die (Chip) – Plane – Block – Page

- NAND flash characteristics
  - Erase-before-write, asymmetry in operation units (read/program: page, erase: block), limited endurance, retention loss…

- Basic NAND flash opeartions
  - Read/program/erase

Today's Agenda

- Advanced NAND Flash Commands
- Address Translation & Garbage Collection

SSD Performance

- Latency (or response time)
  - The time delay until the request is returned
  - Average read latency (4 KiB): 67 us
  - Average write latency (4 KiB): 47 us

HDD: 5~8 ms

- Throughput
  - The number of requests that can be serviced per unit time
    - IOPS: Input/output Operations Per Second
  - Random read throughput: up to 500K IOPS
  - Random write throughput: up to 480K IOPS

HDD: > 1K IOPS

- Bandwidth
  - The amount of data that can be accessed per unit time
  - Sequential read bandwidth: up to 3,500 MB/s
  - Sequential write bandwidth: up to 3,000 MB/s

HDD: ~100 MB/s

Source: https://www.anandtech.com/show/16504/the-samsung-ssd-980-500gb-1tb-review

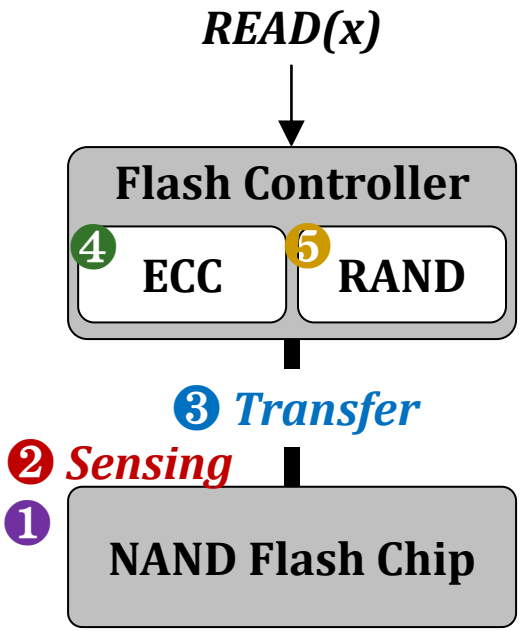
NAND Flash Chip Performance

- Chip operation latency
  - tR: Latency of reading (sensing) data from the cells into the on-chip page buffer
  - tPROG: Latency of programming the cells with data in the page buffer
  - tBERS: Latency of erasing the cells (block)
  - Varies depending on the MLC technology, processing node, and microarchitecture
    - In 3D TLC NAND flash, tR/tPROG/tBERS ≈ 100us/700us/3ms
- I/O rate
  - Number of bits transferred via a single I/O pin per unit time
  - A typical flash chip transfers data in a byte granularity (i.e., via 8 I/O pins)
  - e.g., 1-Gb I/O rate & 16-KiB page size → tDMA = 16 us

NAND Flash Chip Performance (Cont.)

- tR, tPROG, and tBERS
  - Latencies for chip-level read/program/erase operations
  - tR: 50~100 us
  - tPROG: 700us~1000 us
  - tBERS: 3ms~5ms

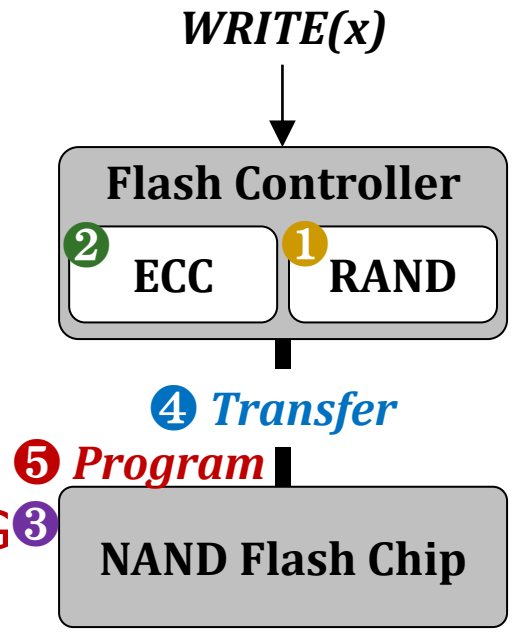
- Flash-controller level latency
  - 1-Gb I/O rate and 16-KiB page size
  - Read
    - (tCMD) + tR + tDMA + $tECC_{DEC}$ + (tRND)
    - e.g., 100 + 16 + 20 = 136 us



NAND Flash Chip Performance (Cont.)

- tR, tPROG, and tBERS
  - Latencies for chip-level read/program/erase operations
  - tR: 50~100 us
  - tPROG: 700us~1000 us
  - tBERS: 3ms~5ms

- Flash-controller level latency
  - 1-Gb I/O rate and 16-KiB page size
  - Read
    - (tCMD) + tR + tDMA + $tECC_{DEC}$ + (tRND)
    - e.g., 100 + 16 + 20 = 136 us
  - Program
    - (tRND) + $tECC_{ENC}$ + (tCMD) + tDMA + tPROG
    - e.g., 20 + 16 + 700 = 736 us



NAND Flash Chip Performance (Cont.)

- How about bandwidth?
  - Read
    - 16 KiB / 136 us ≈ 120 MB/s
  - Write
    - 16 KiB / 736 us ≈ 22 MB/s

WAIT!

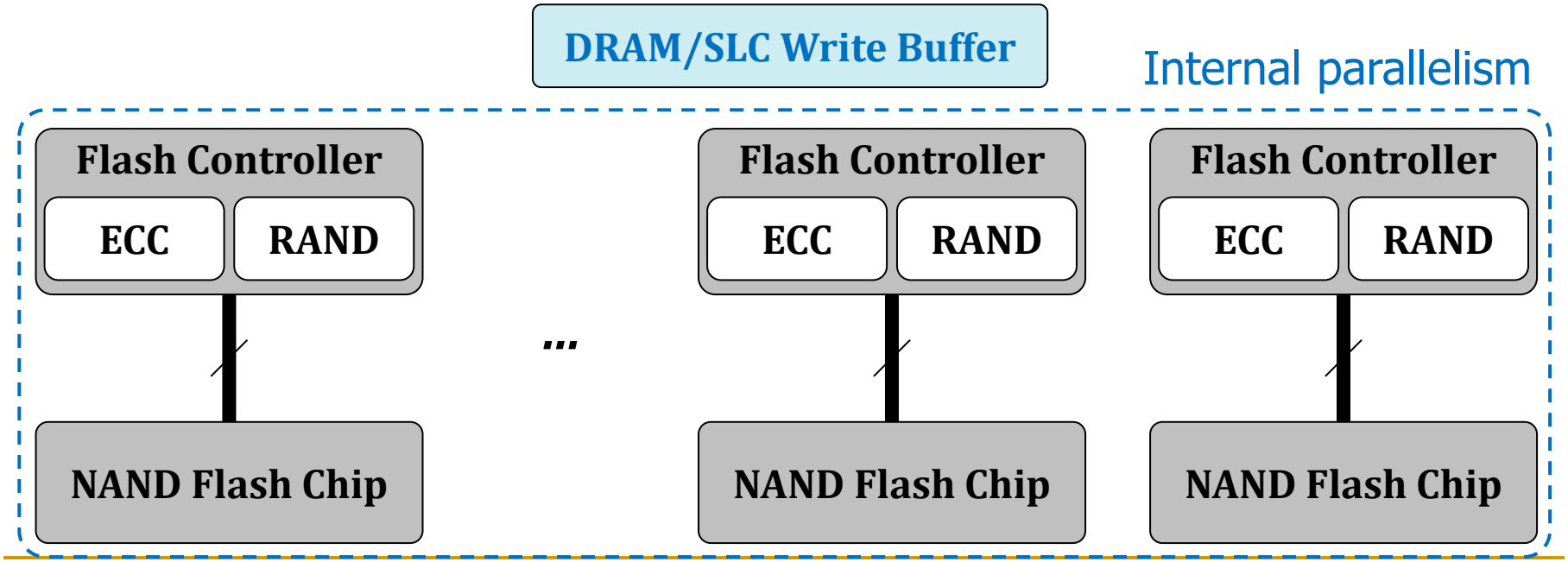
SSD read latency: 67 us

SSD read bandwidth: 3.5 GB/s

SSD write latency: 47 us

SSD write bandwidth: 3 GB/s

Optimizations w/ advanced commands



Advanced Commands for Small Reads

- Minimum I/O units in modern file systems: 4 KiB
  - Latency & bandwidth waste due to I/O-unit mismatch
  - e.g., A page read unnecessarily reads/transfers 12-KiB data

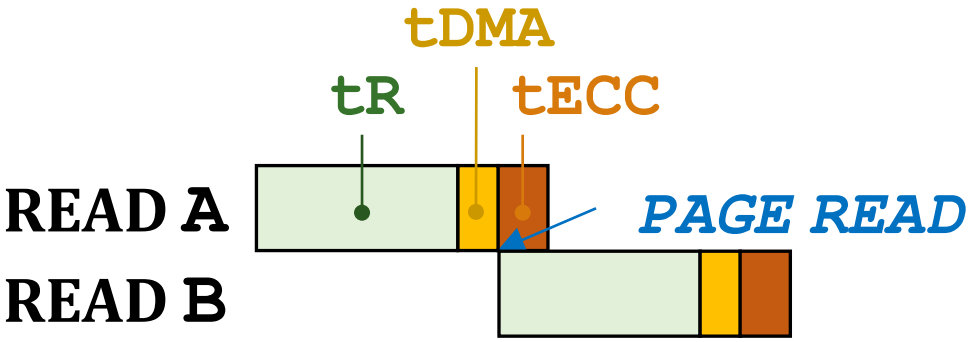
- Optimization 1: Sub-page sensing
  - e.g., Micron SNAP READ operation[1]
  - Microarchitecure-level optimization – directly reduces tR

- Optimization 2: Random Data Out (RDO)
  - Data transfer with an arbitrary offset and size
  - Reduce tDMA and $tECC_{DEC}$

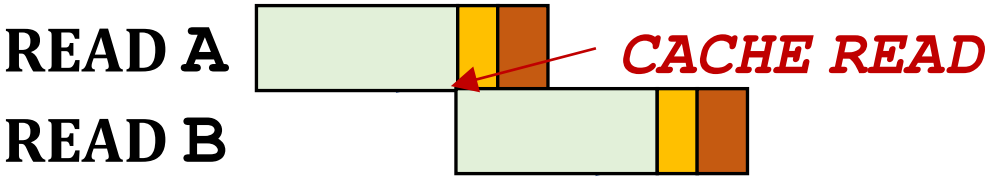
[1]https://media-www.micron.com/-/media/client/global/documents/products/technical-note/nand-flash/tn_2993_snap_read.pdf

CACHE READ Command

- Performs consecutive reads in a pipelined manner



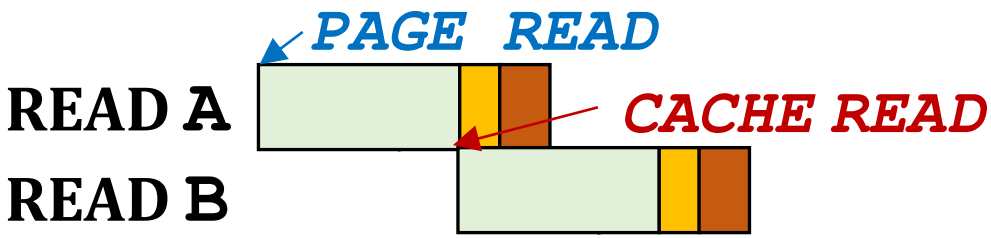
Regular PAGE READ:
Overlaps only tECC with tR

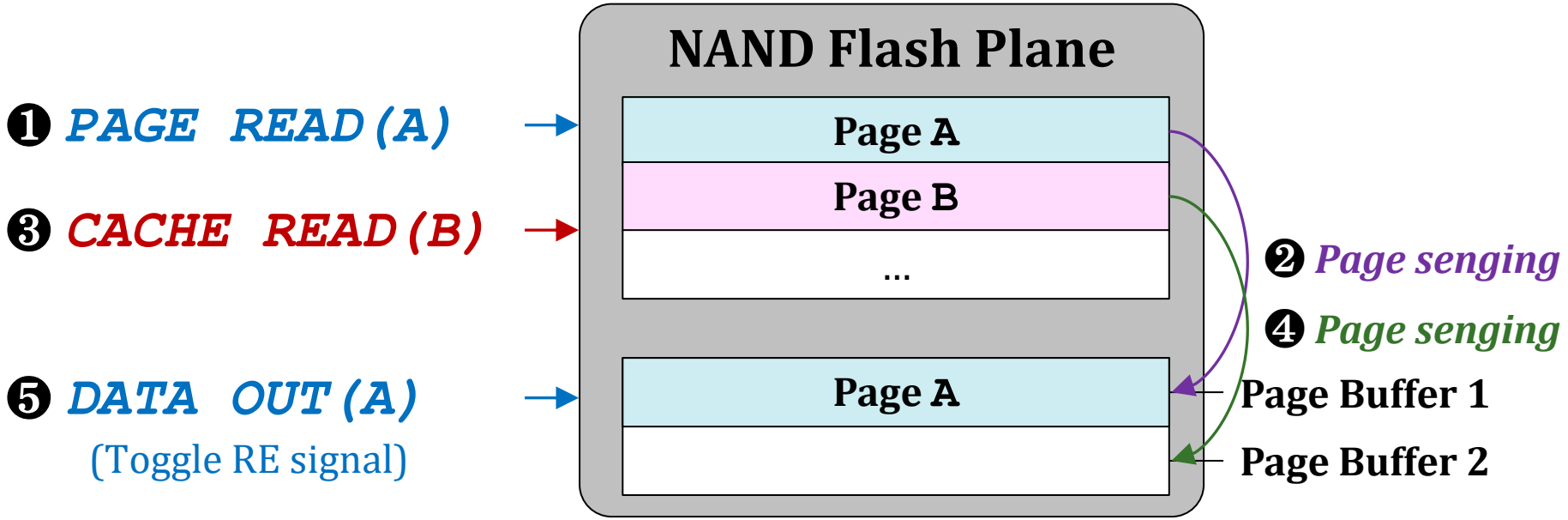


CACHE READ:
Overlaps tDMA & tECC with tR

Enabling the CACHE READ Command

- Needs additional on-chip page buffer



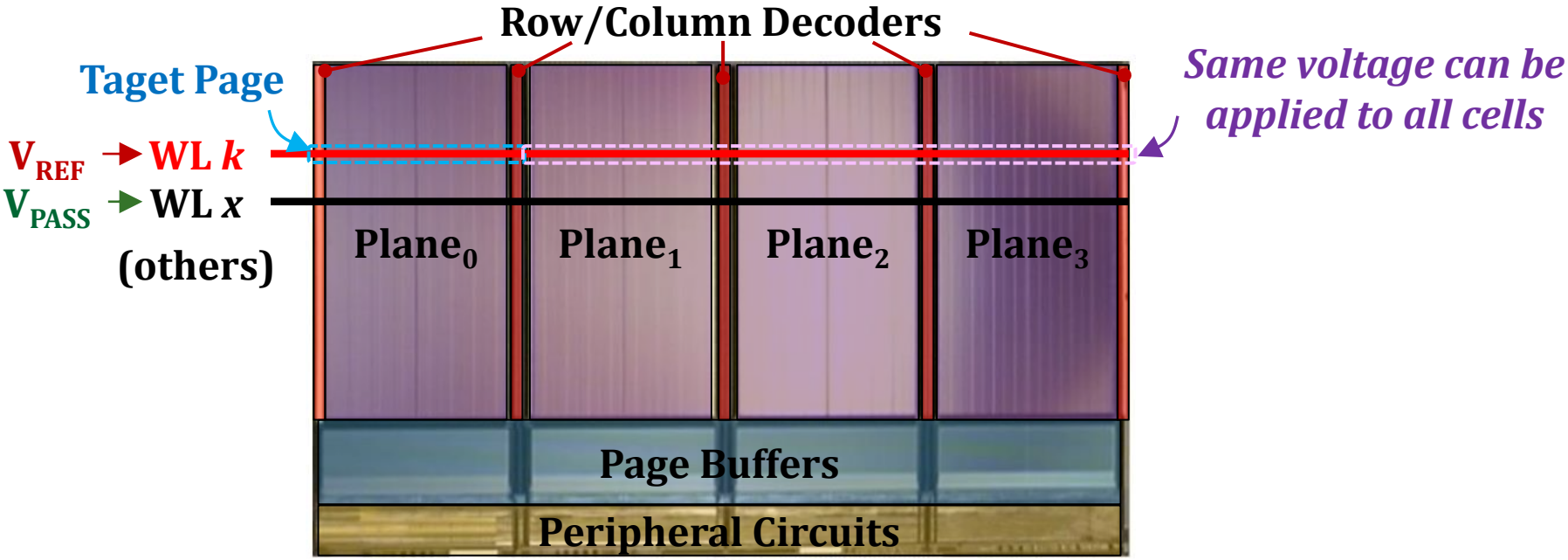


CACHE READ Command: Benefit

- Removes tDMA from the critical path
  - Increases throughput/bandwidth
  - Reduces effective latency
    - By reducing the time delay for a request being blocked by the previous request

Multi-Plane Operations

- Concurrent operations on different planes
  - Recall: Planes share WLs and row/column decoders



- Opportunity: Planes can concurrently operate
- Constraints: Only for the same operations on the same page offset

Multi-Plane Operations: Benefit

- Increase the throughput/bandwidth almost linearly with # of planes that concurrently operate
  - Bandwidth with regular page programs:
    16 KiB / 736 us ≈ 22 MB/s
  - Bandwidth with multi-plane page programs (2 plane):
    32 KiB / 736 + 16 (tDMA) + 20 (tECC) us ≈ 41.5 MB/s

- Per-operation latency increases
  - Regular page program: $tECC_{ENC}$ + tDMA + tPROG
  - Multi-plane page program: $N_{Plane} \times (tECC_{ENC} + tDMA)$ + tPROG

- The benefits highly depend on the access pattern and FTL's data placement
  - Random-read-dominant vs. Random-write-dominant

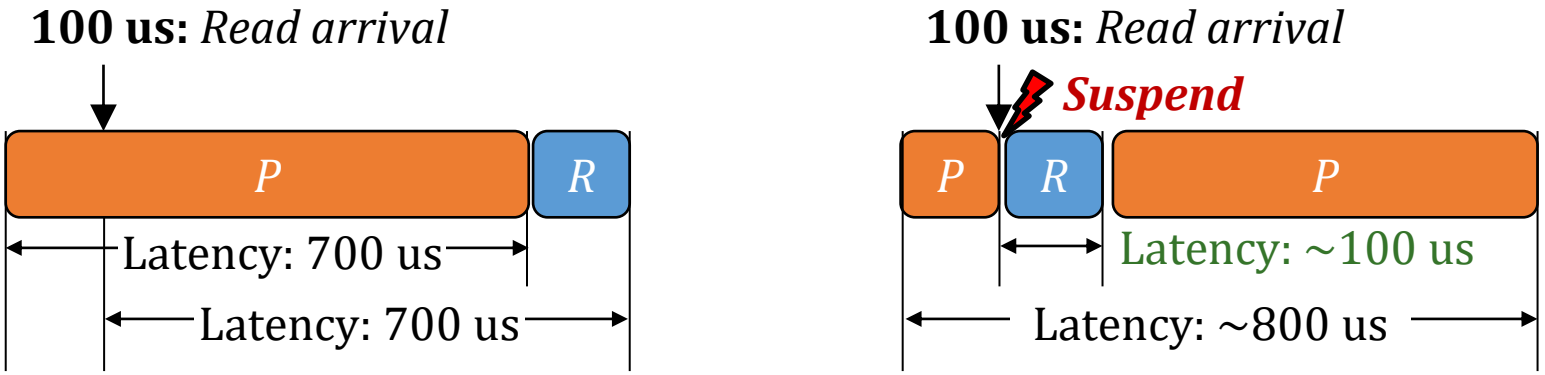
Program & Erase Suspensions

- Read performance is often more important
  - Writes can be done in an asynchronous manner using buffers
    - e.g., return a write request immediately after receiving the data (and storing it to the write buffer)
  - A read request can be returned only when the requested data is ready (after reading the data from the chip)

- Significant latency asymmetry
  - tR: 100 us, tPROG: 700 us, tBERS: 5 ms (TLC NAND flash)
    - If the chip is designed to program all the pages in the same WL at once, the actual program latency is 2,100 us
  - The worst-case chip-level read latency can be 50x longer than the best-case latency

Program & Erase Suspensions (Cont.)

- Suspends an on-going program (erase) operation once a read arrives



  - Pros: Significantly decreases the read latency
  - Cons
    - Additional page buffer (for data to program)
    - Complicated I/O scheduling (Until when can we suspend on-going program requests?)
    - Negative impact on the endurance

Summary

- **Subpage Sensing & Random Data Out (RDO)**
  - For I/O-unit mismatch b/w OS and NAND flash memory
- **Cache Read Command**
  - For improving a chip's read throughput
  - By overlapping data transfer and page sensing
- **Multi-Plane Operations**
  - For improving a chip's throughput
  - By enabling concurrently operation of multiple planes
- **Program & Erase Suspensions**
  - For improving the read latency (operation latency asymmetry)
  - By prioritizing latency-sensitive reads over writes/erases

Today's Agenda

- Advanced NAND Flash Commands
- Address Translation & Garbage Collection

Flash Translation Layer: Overview

- SSD firmware (often referred to as SSD controller)
  - Provides backward compatibility with traditional HDDs
  - By hiding unique characteristics of NAND flash memory

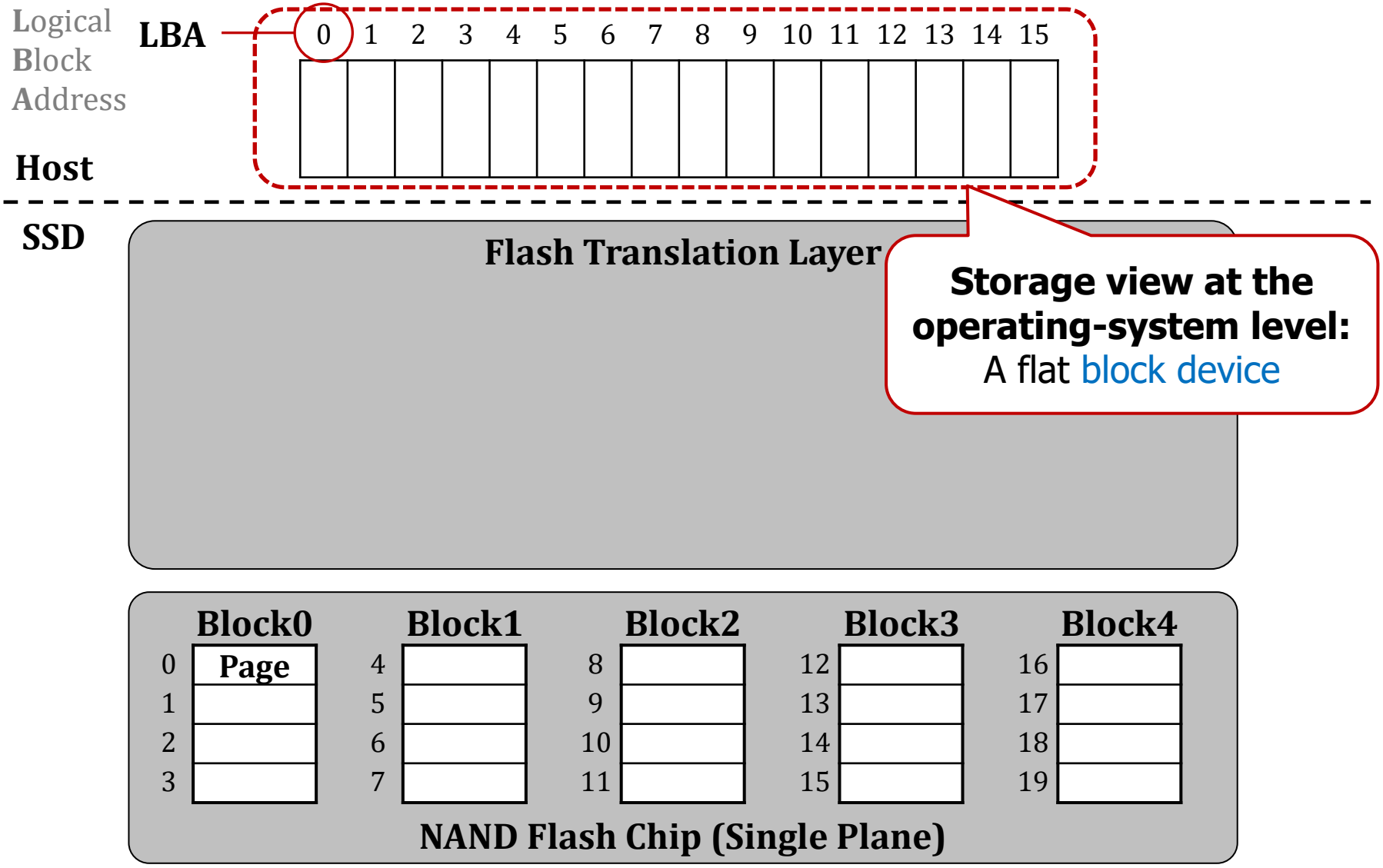
- Responsible for many important SSD-management tasks
  - Address translation + garbage collection
    - Performs out-of-place writes due to erase-before-write property
  - Wear leveling
    - To prolong SSD lifetime by evenly distributing P/E cycles
  - Data refresh
    - Resets transient errors by copying data to a new page(s)
  - I/O scheduling
    - To take full advantage of SSD internal parallelism

Flash Translation Layer: Overview

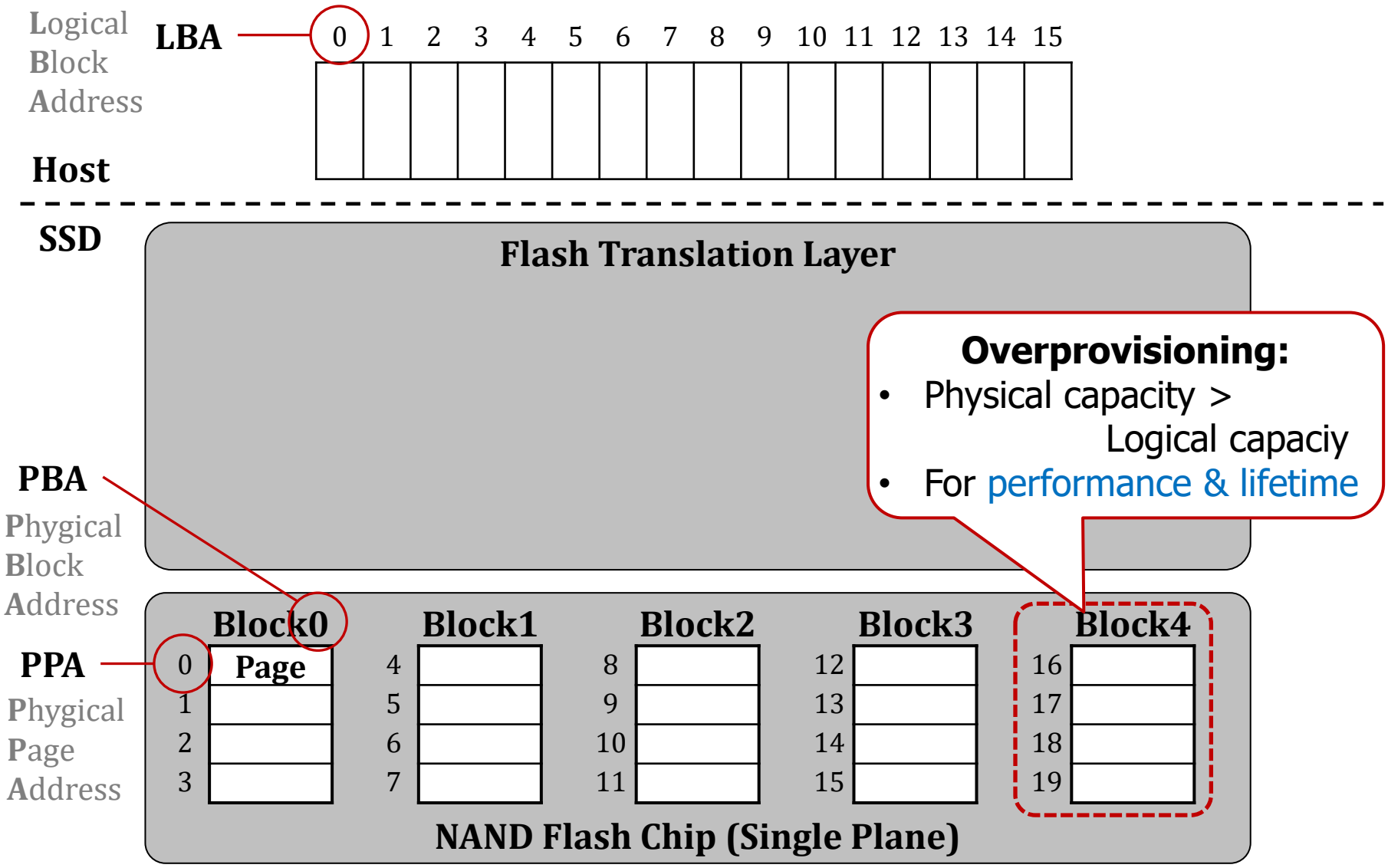
- SSD firmware (often referred to as SSD controller)
  - Provides backward compatibility with traditional HDDs
  - By hiding unique characteristics of NAND flash memory

- Responsible for many important SSD-management tasks
  - Address translation + garbage collection
    - Performs out-of-place writes due to erase-before-write property
  - Wear leveling
    - To prolong SSD lifetime by evenly distributing P/E cycles
  - Data refresh
    - Resets transient errors by copying data to a new page(s)
  - I/O scheduling
    - To take full advantage of SSD internal parallelism

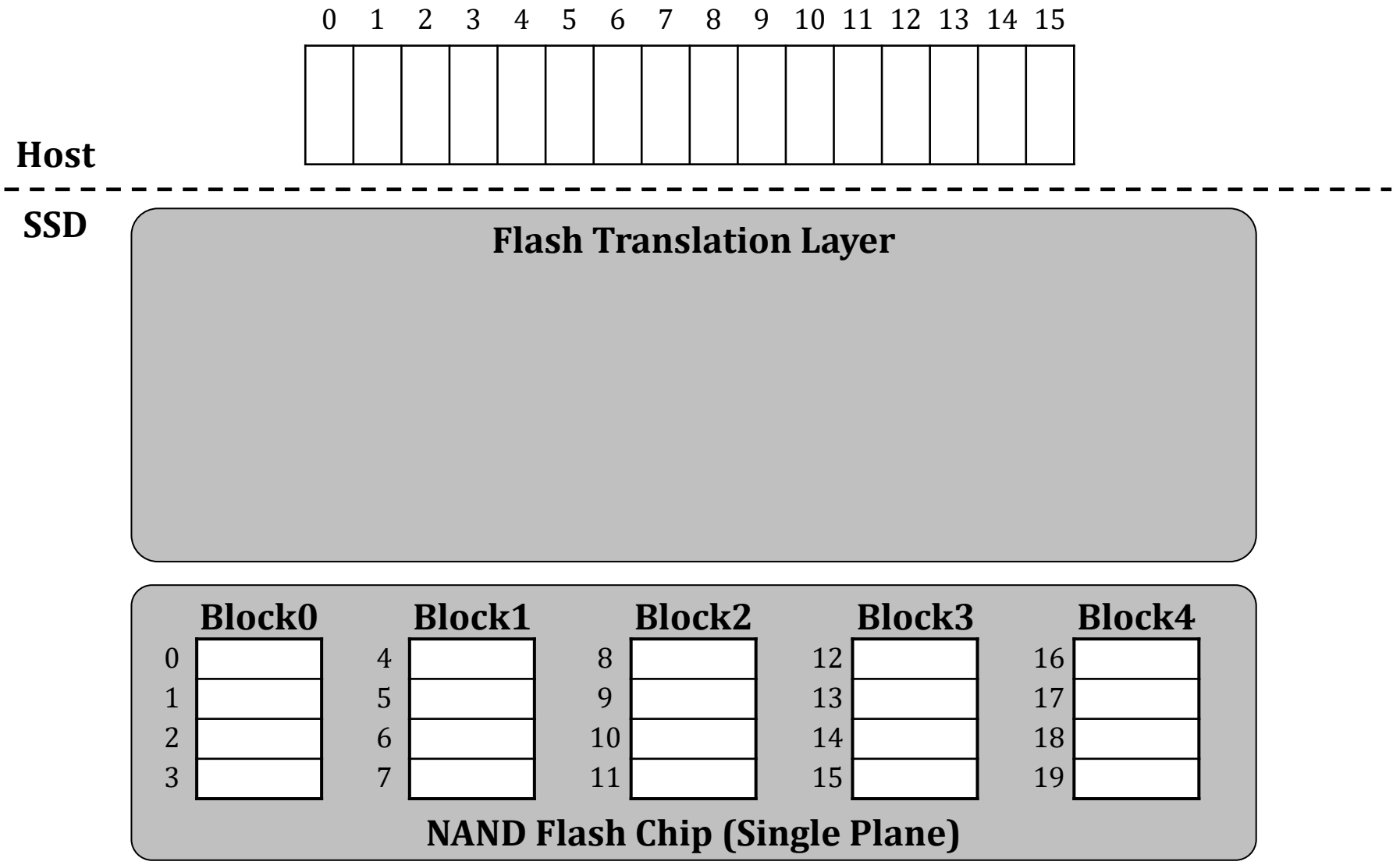
Simple SSD Architecture



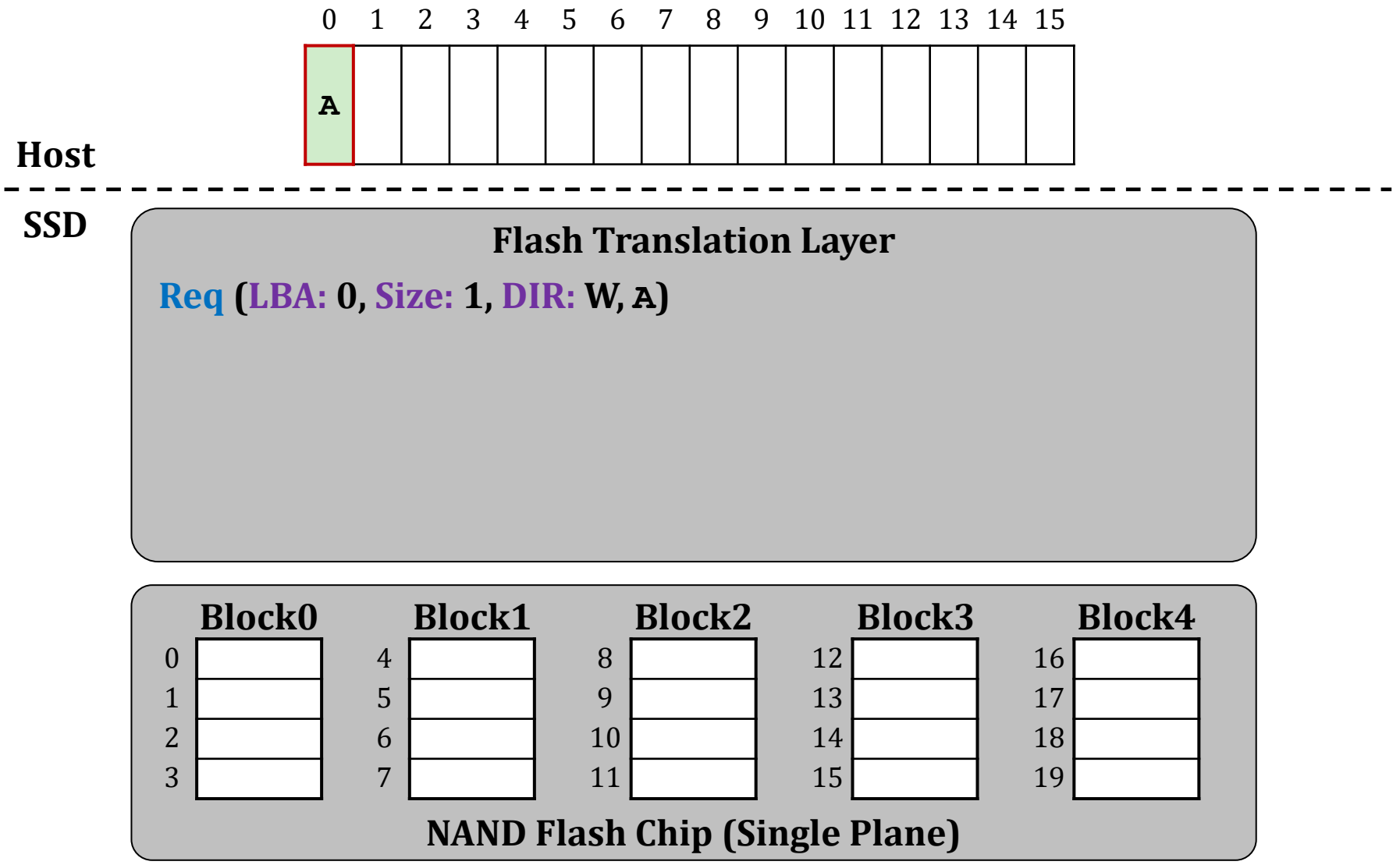
Simple SSD Architecture



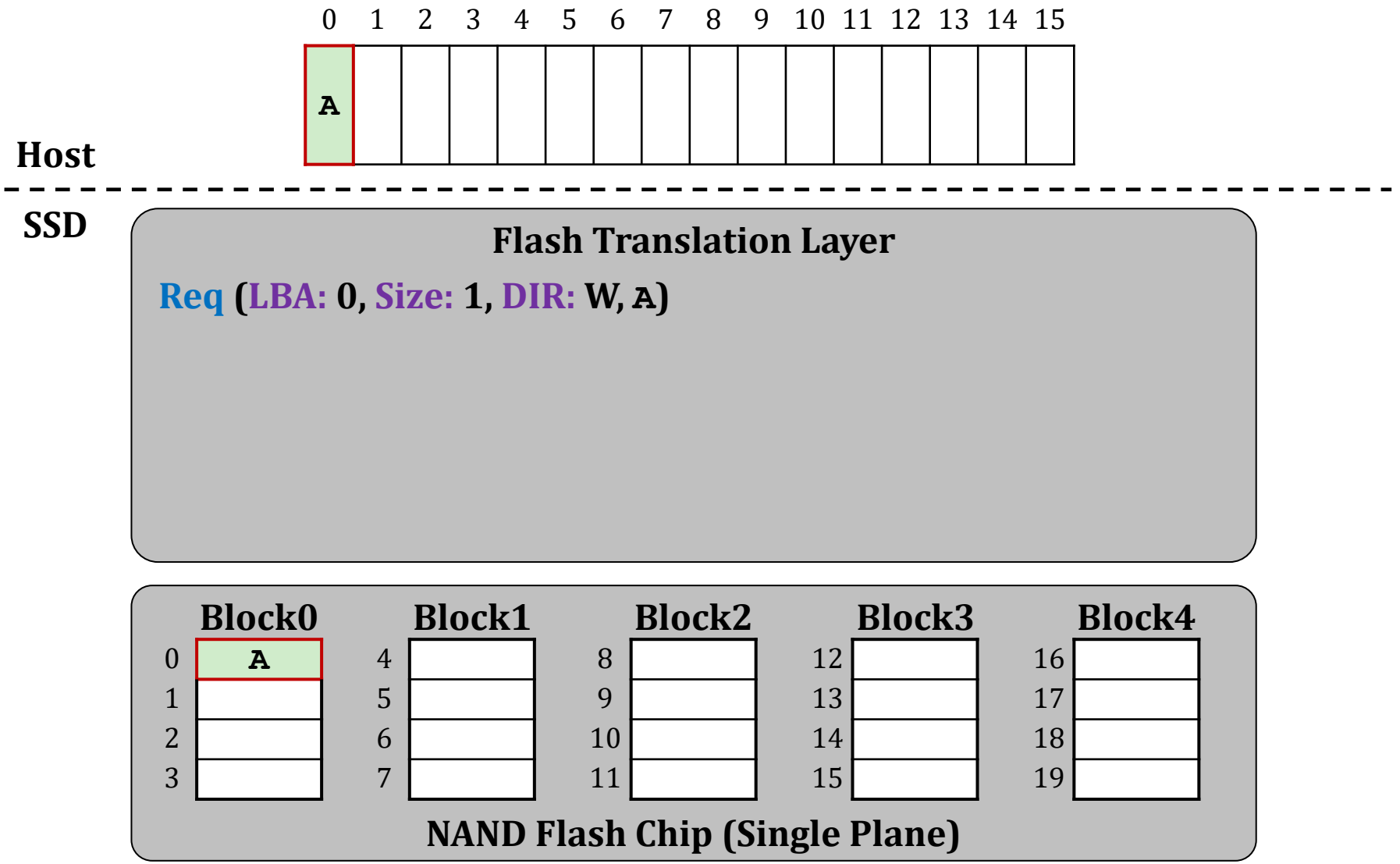
Write Request Handling: Page Write



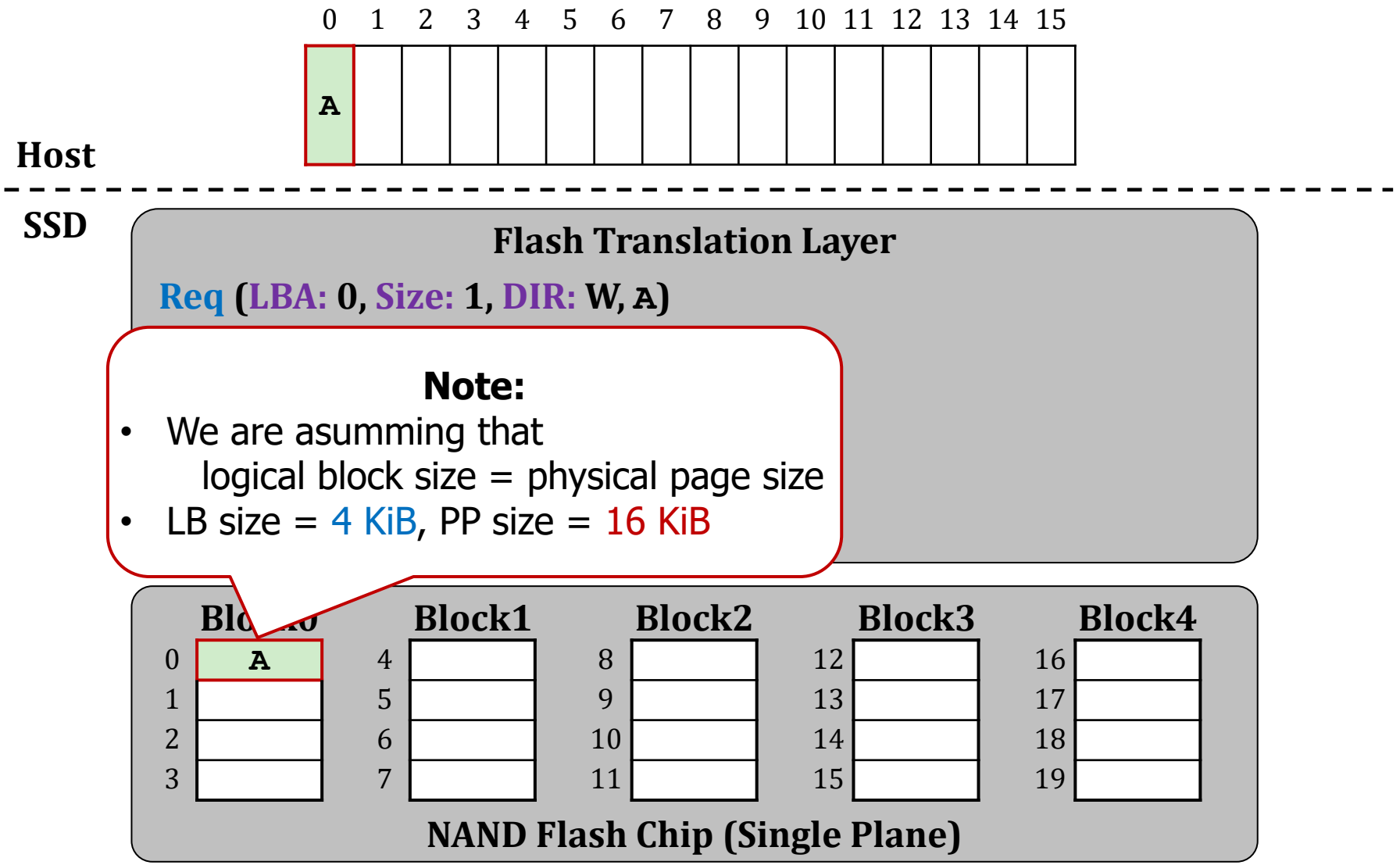
Write Request Handling: Page Write



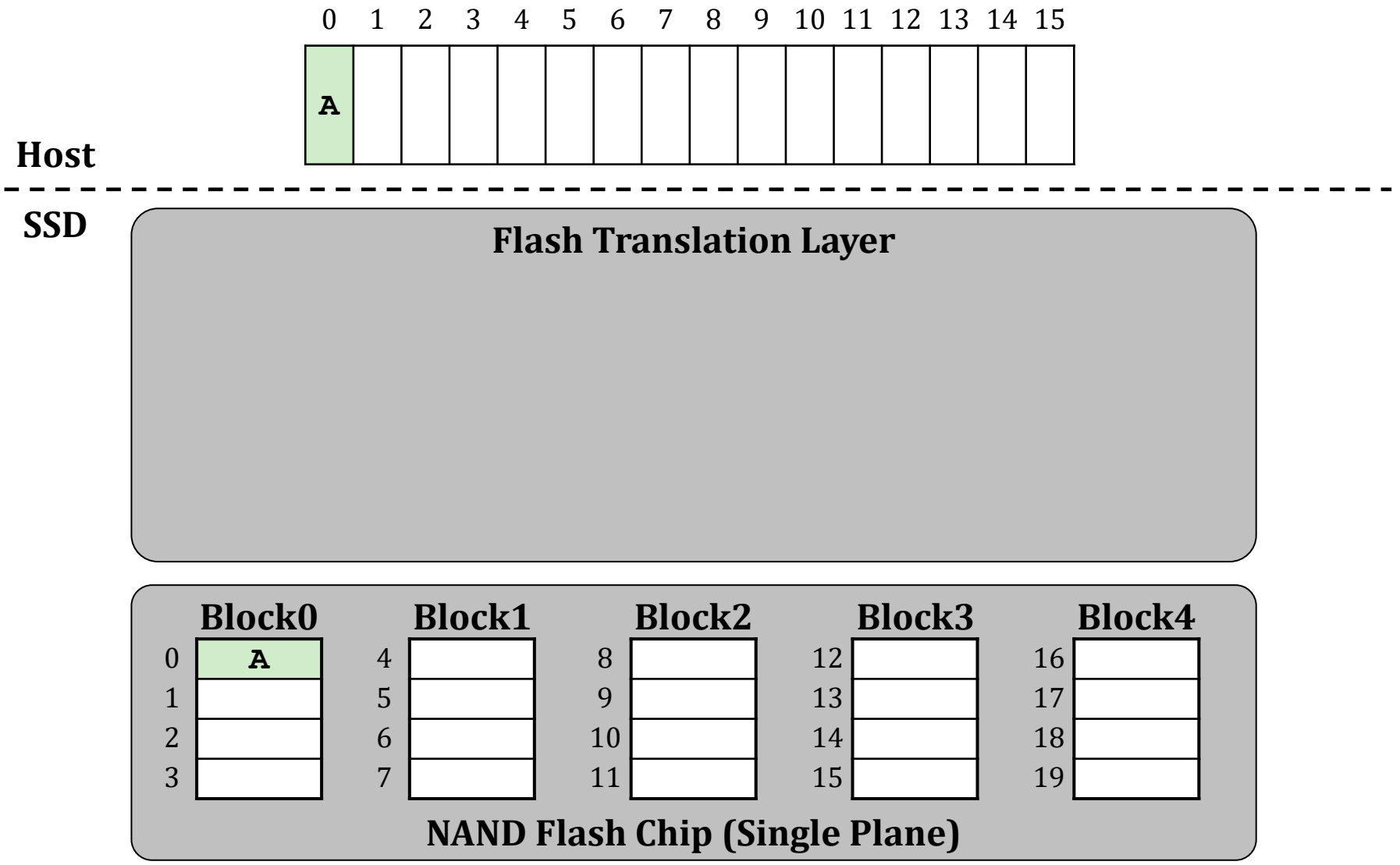
Write Request Handling: Page Write



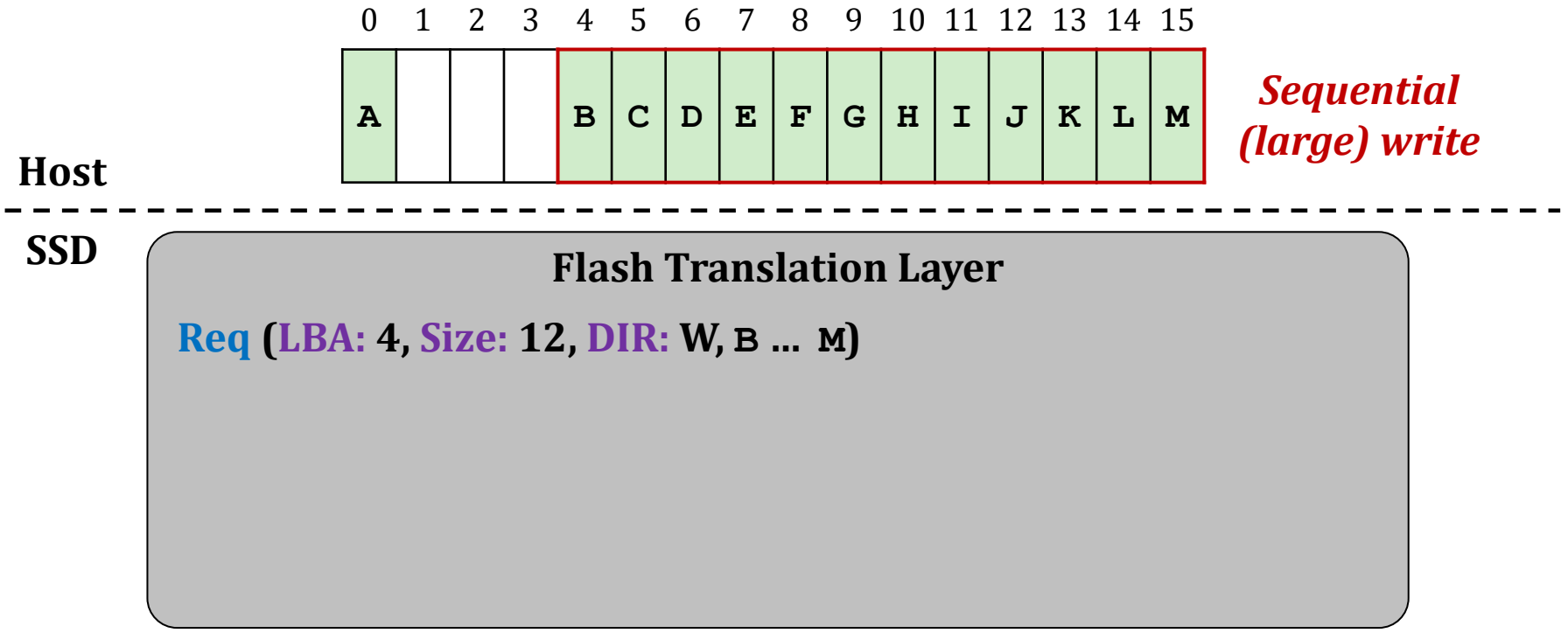
Write Request Handling: Page Write



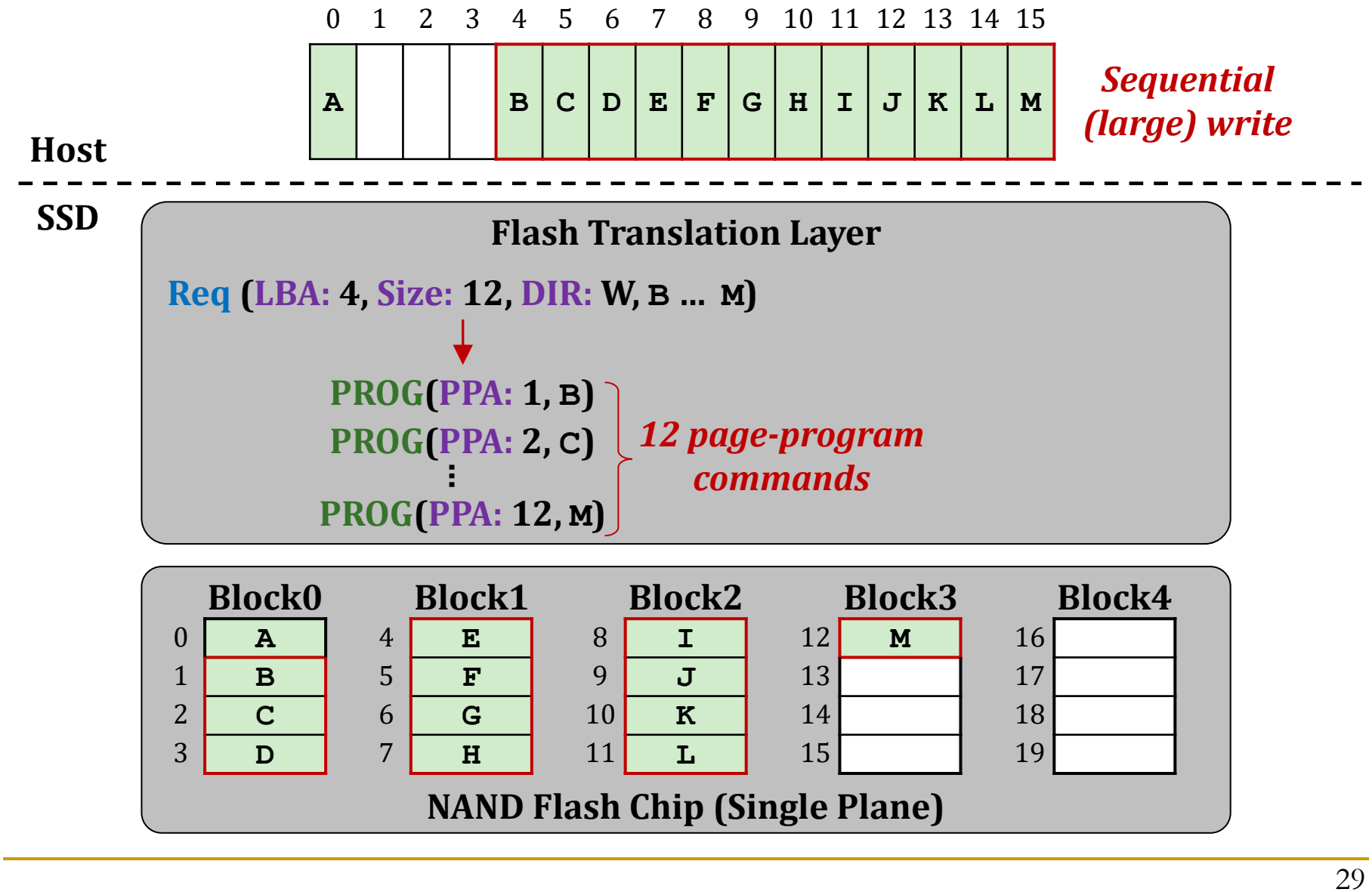
Write Request Handling: Sequential Write



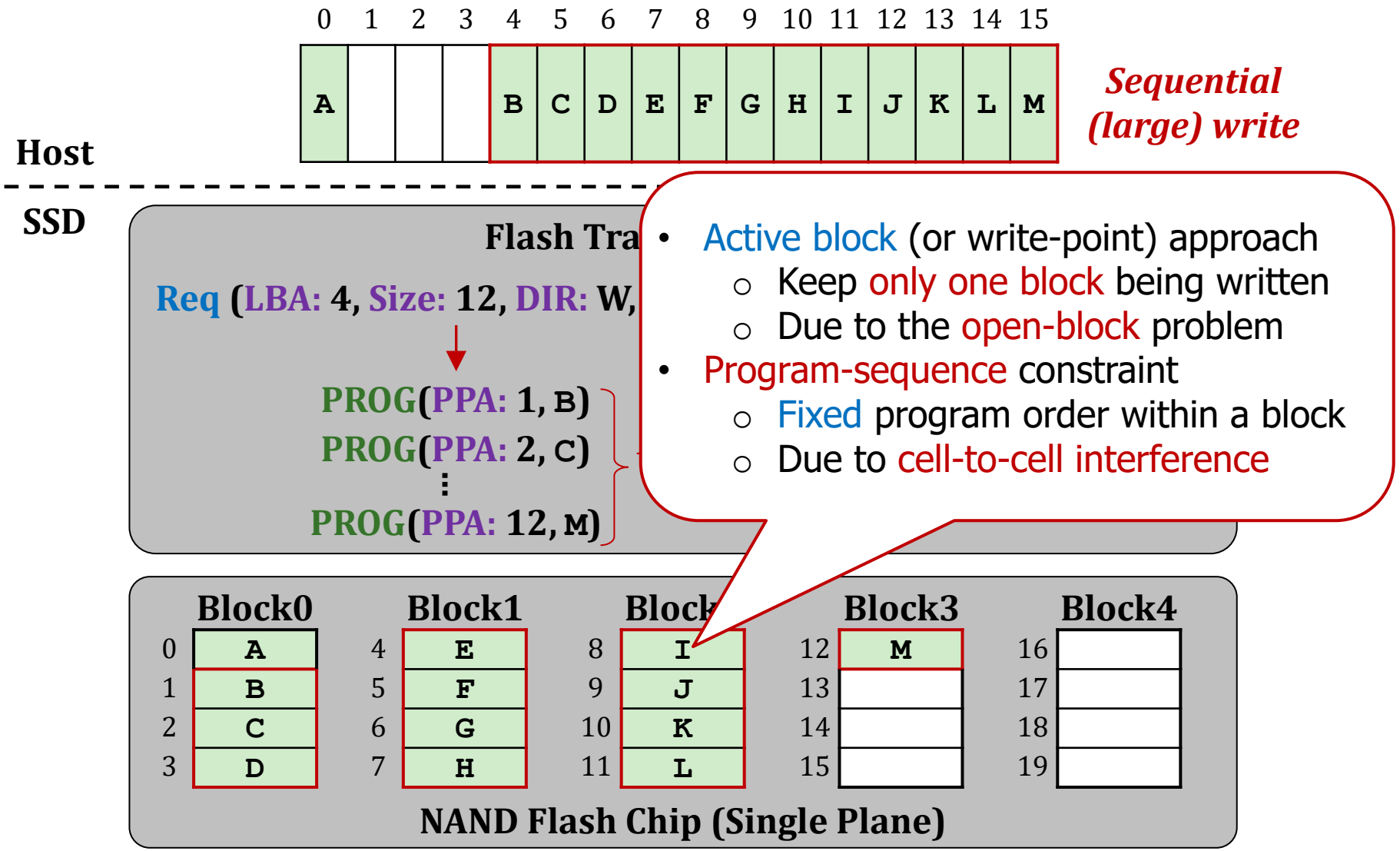
Write Request Handling: Sequential Write



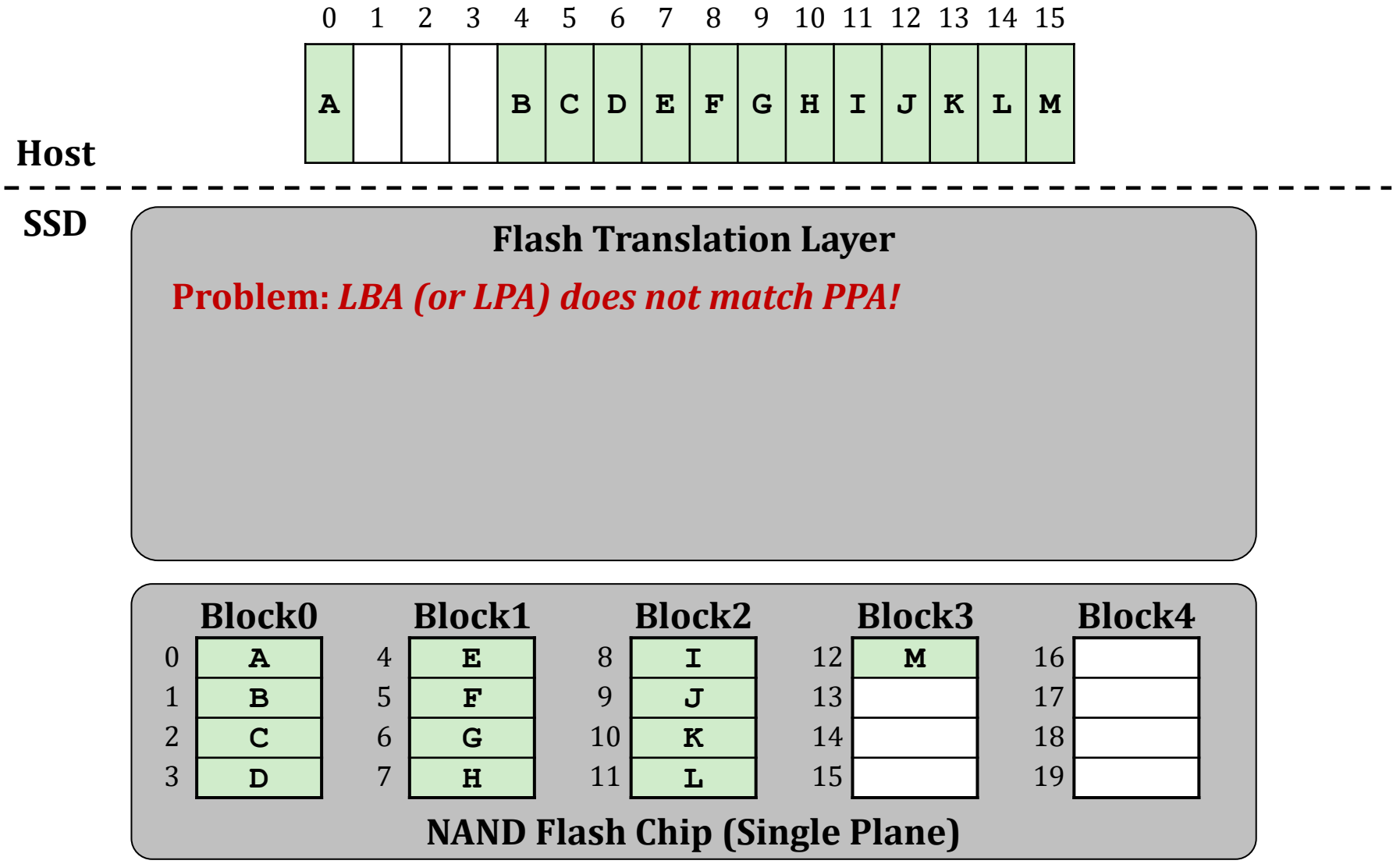
Write Request Handling: Sequential Write



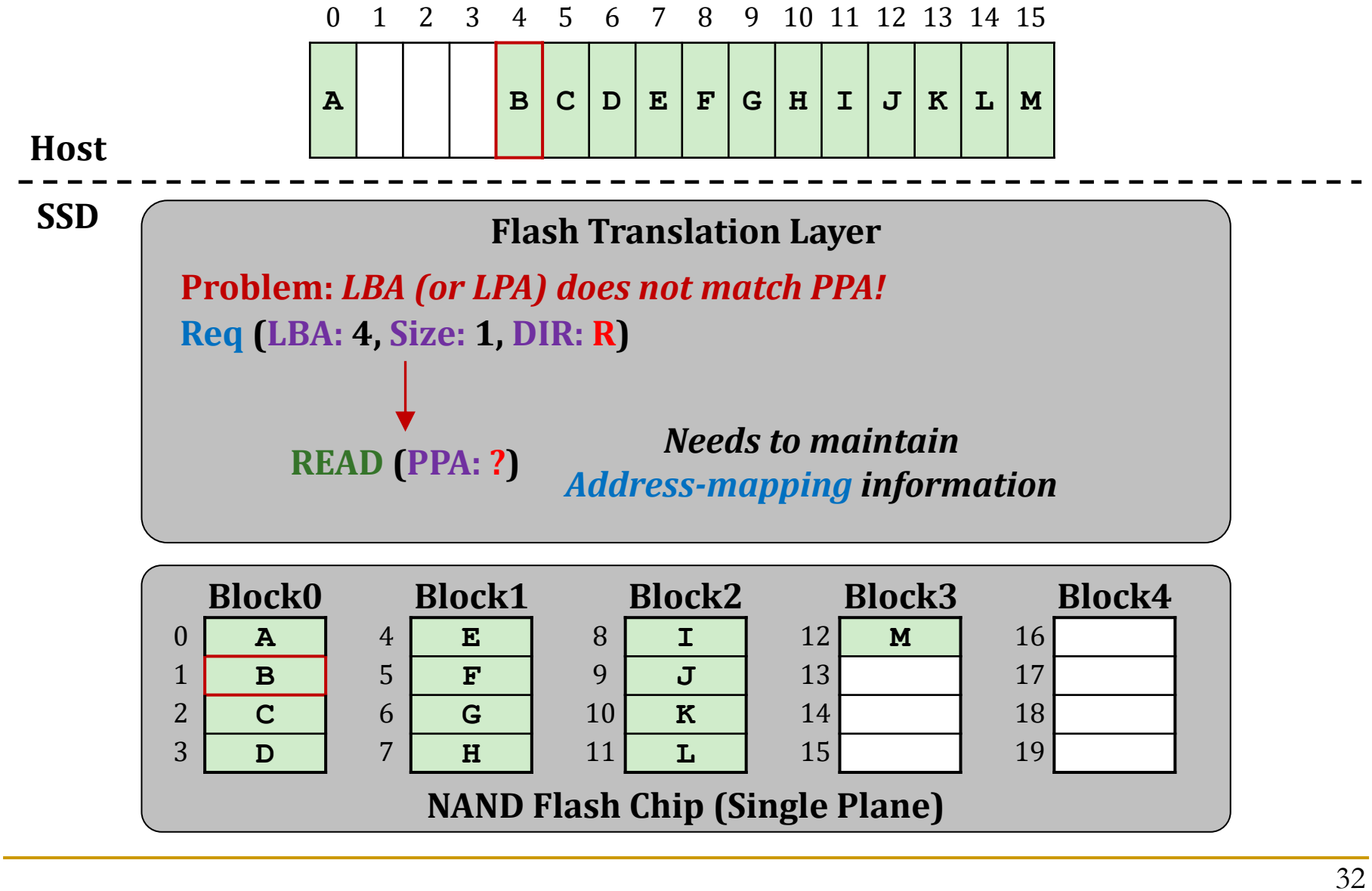
Write Request Handling: Sequential Write



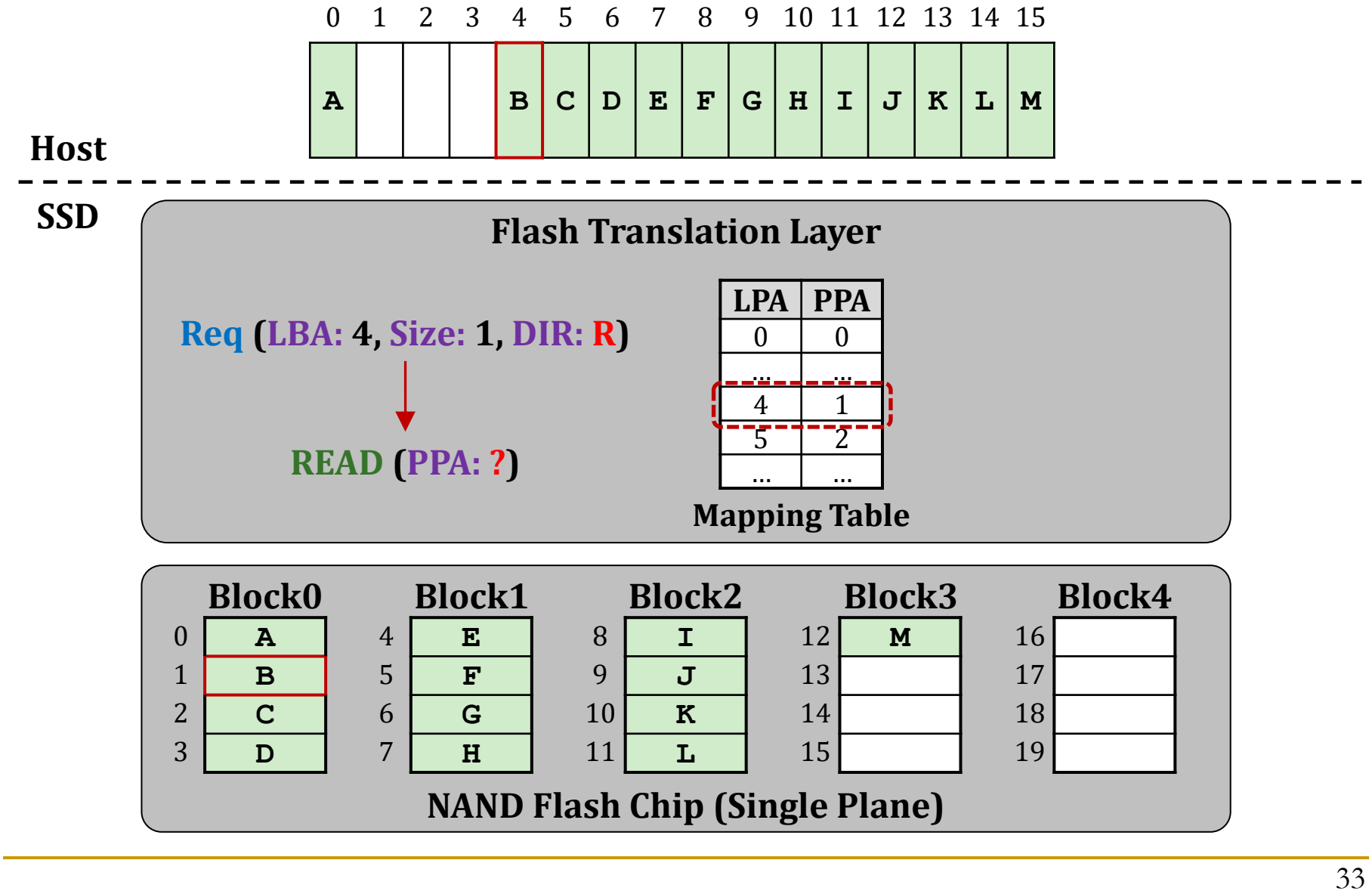
Write Request Handling: Address Mapping



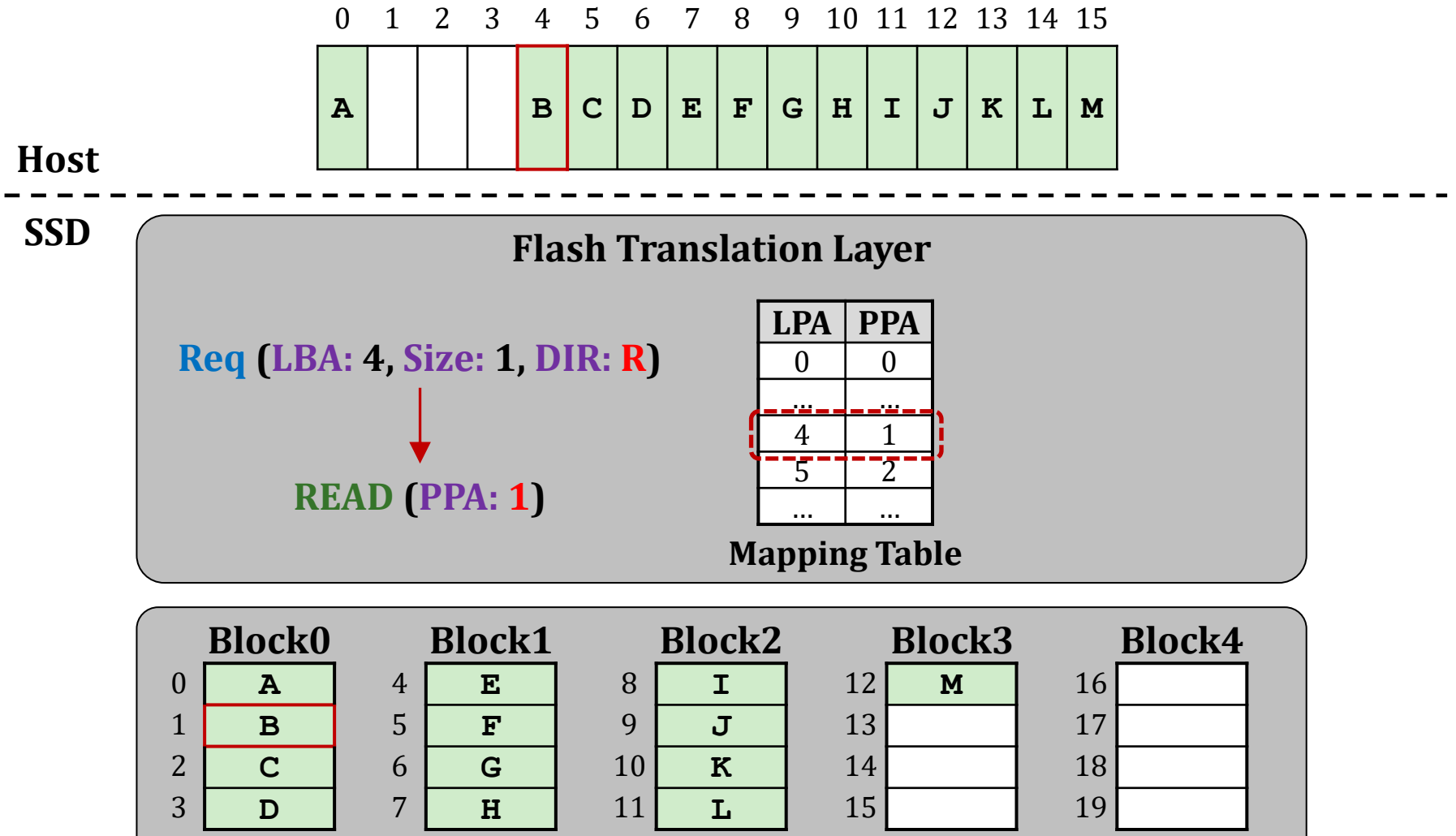
Write Request Handling: Address Mapping



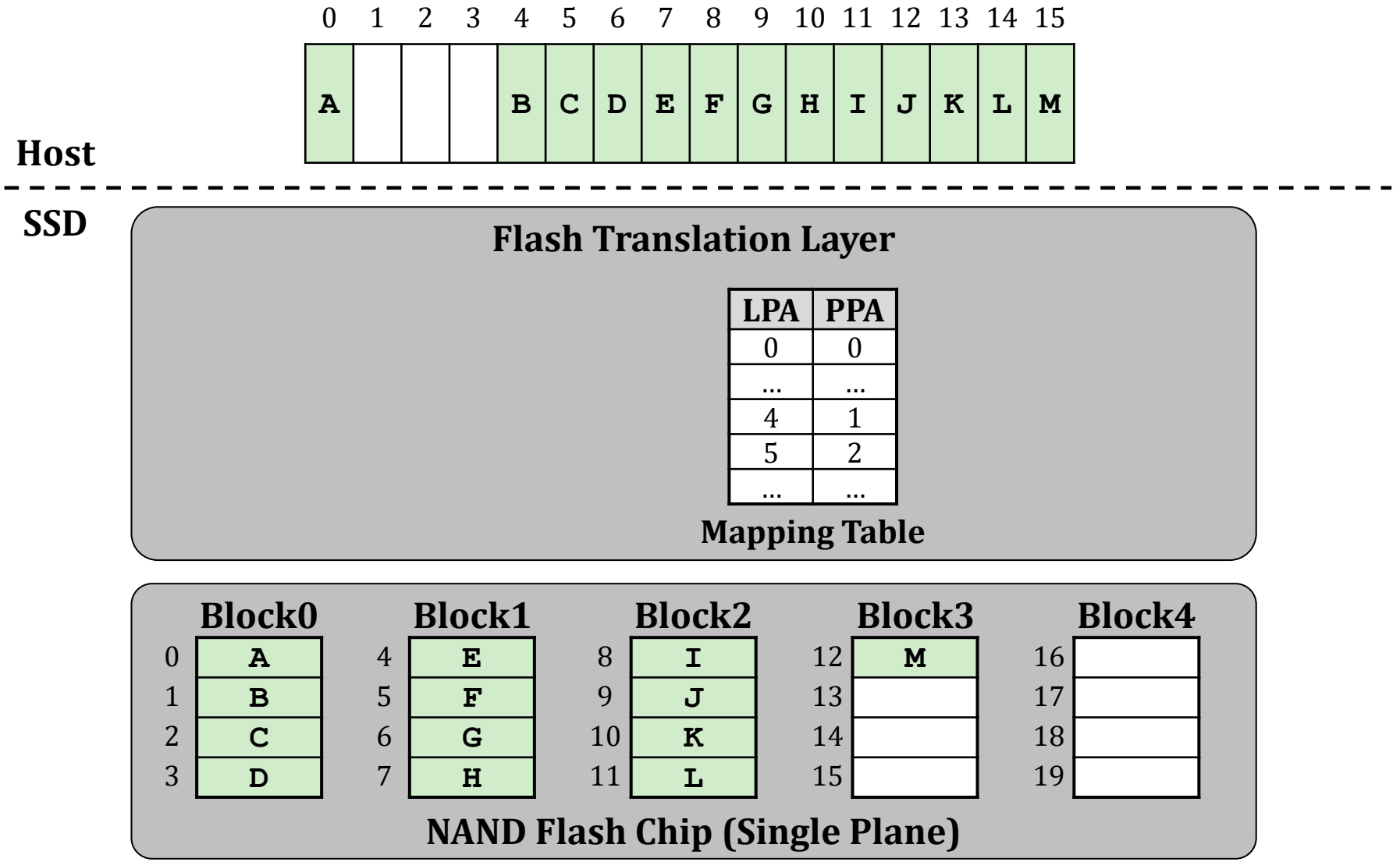
Write Request Handling: Address Mapping



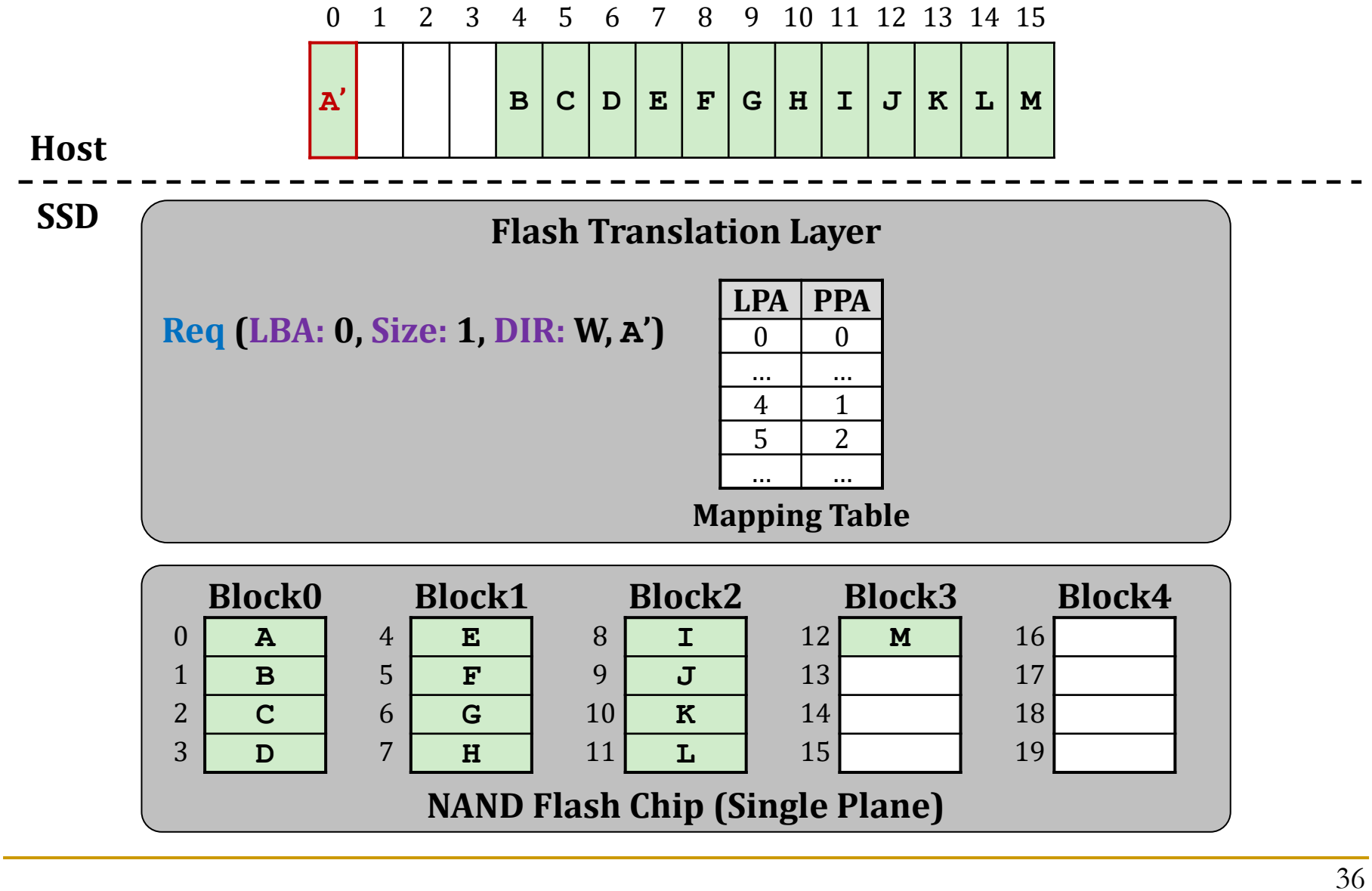
Write Request Handling: Address Mapping



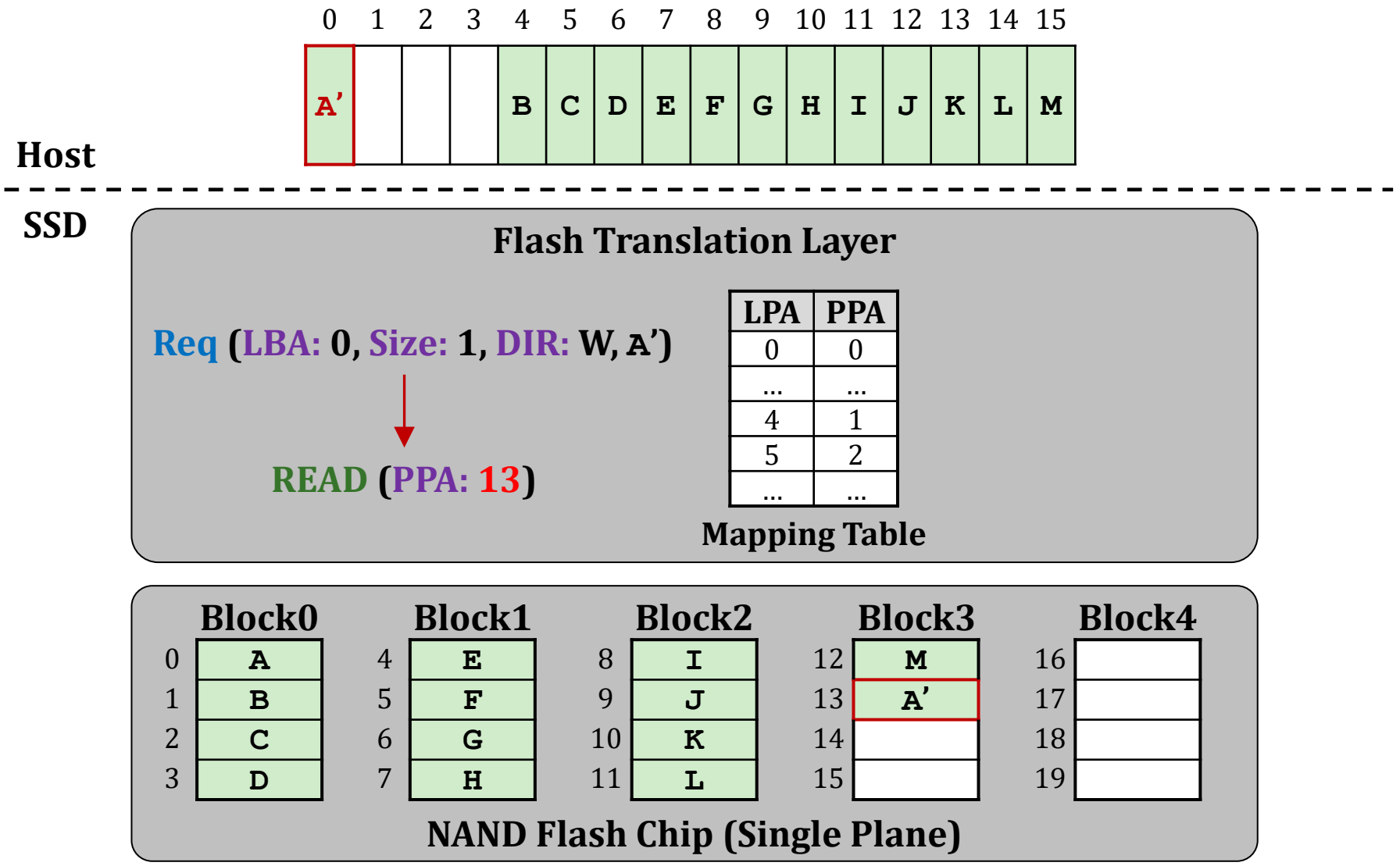
Write Request Handling: Update



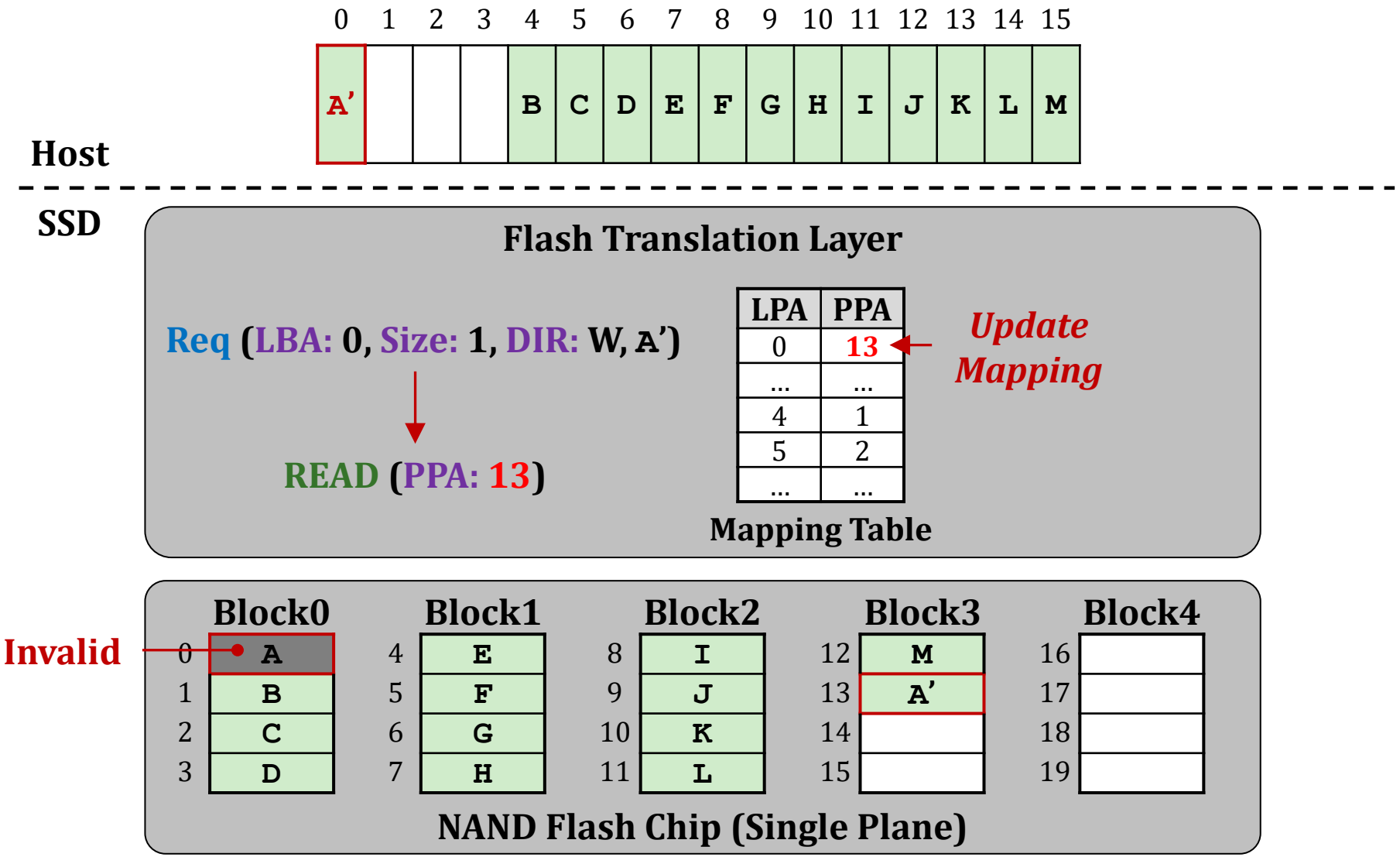
Write Request Handling: Update



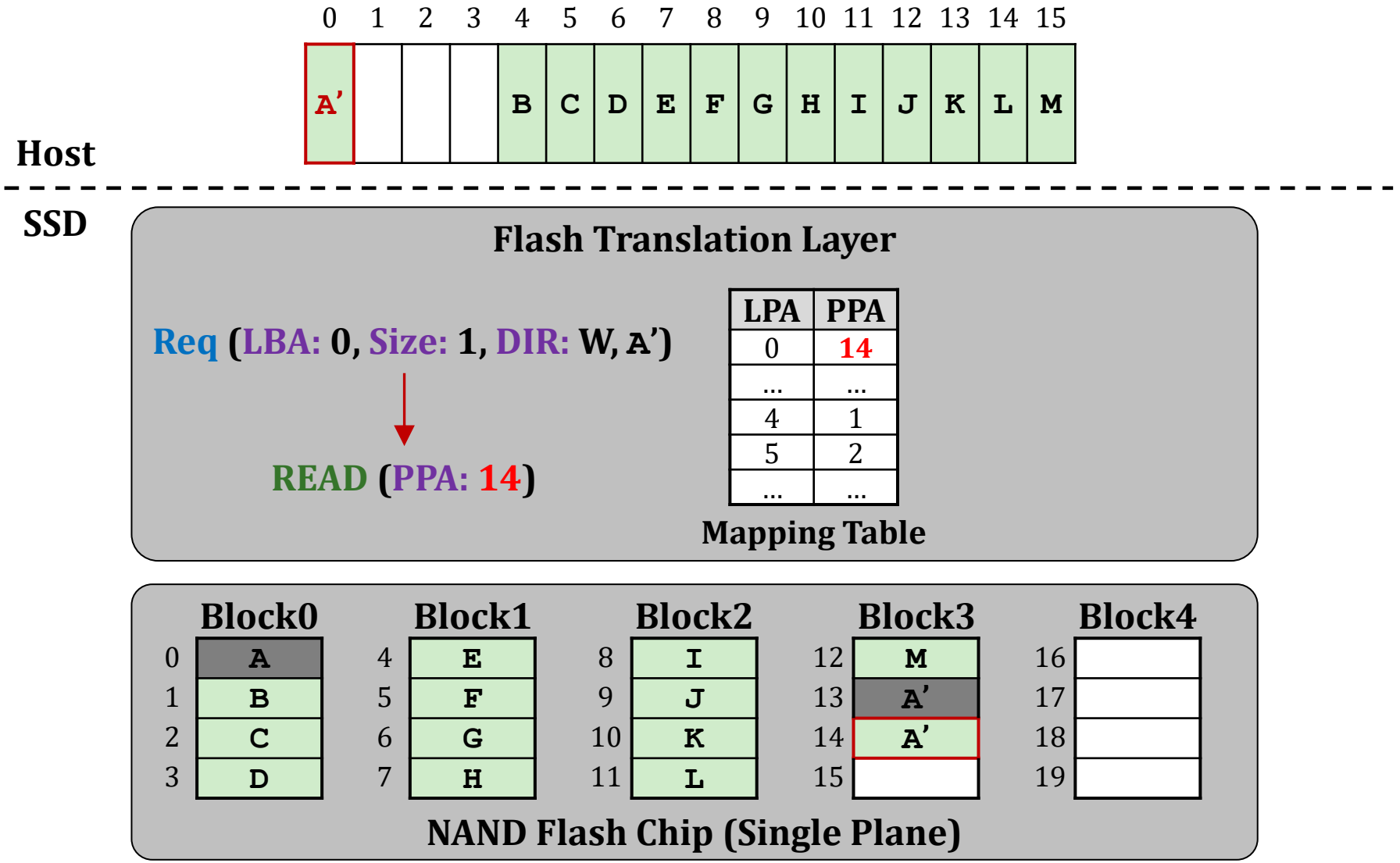
Write Request Handling: Update



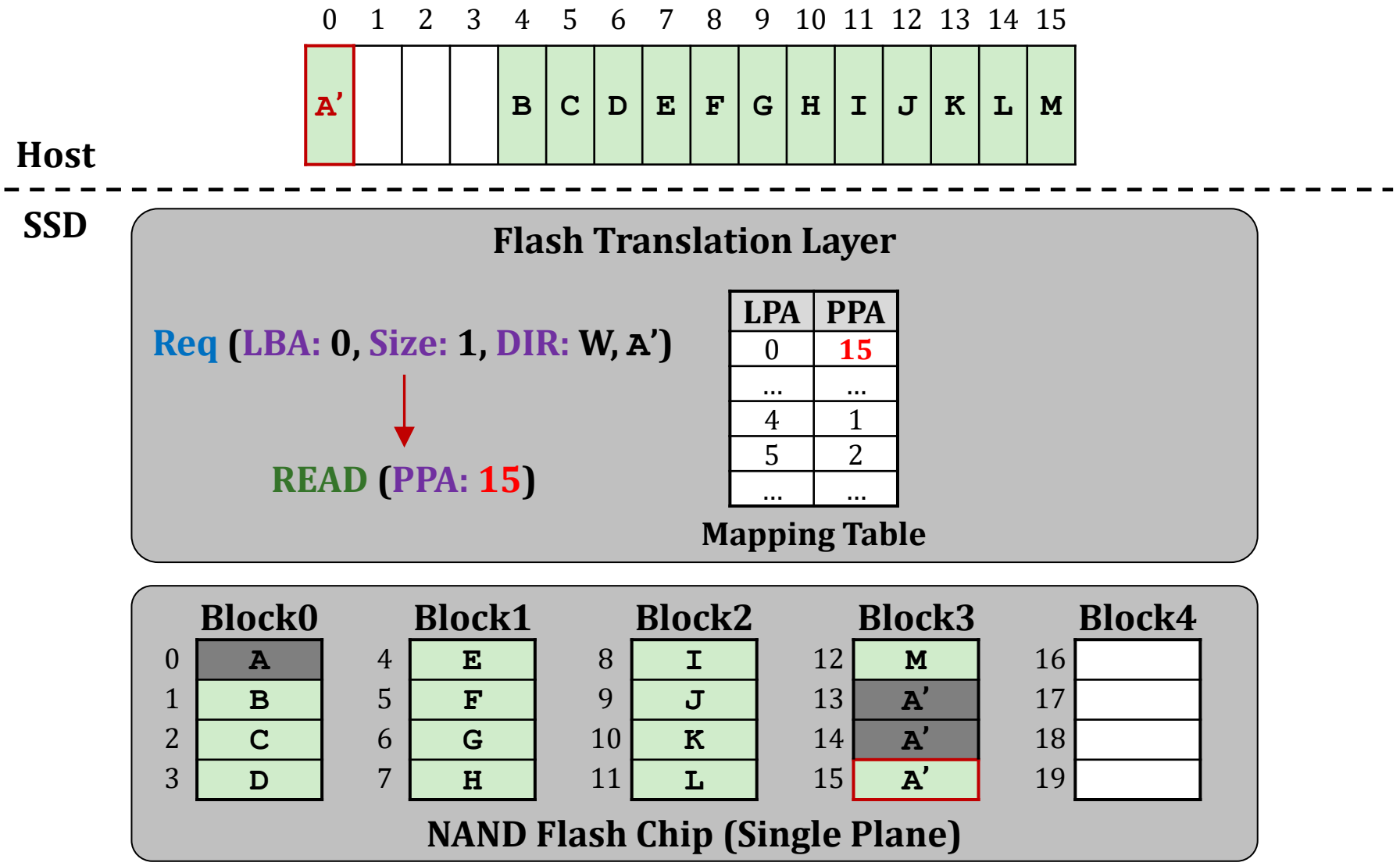
Write Request Handling: Update



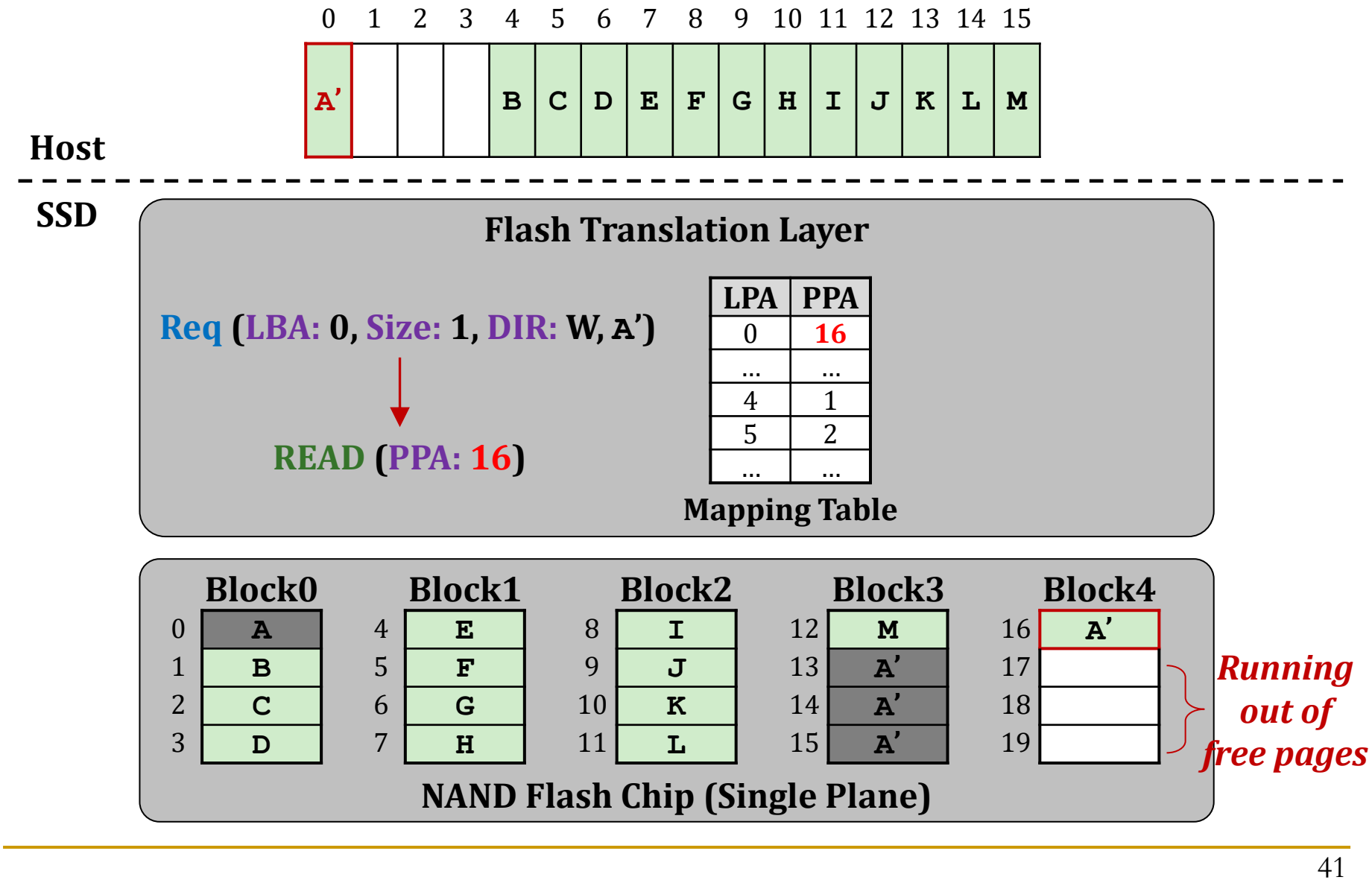
Write Request Handling: Update



Write Request Handling: Update



Write Request Handling: Update

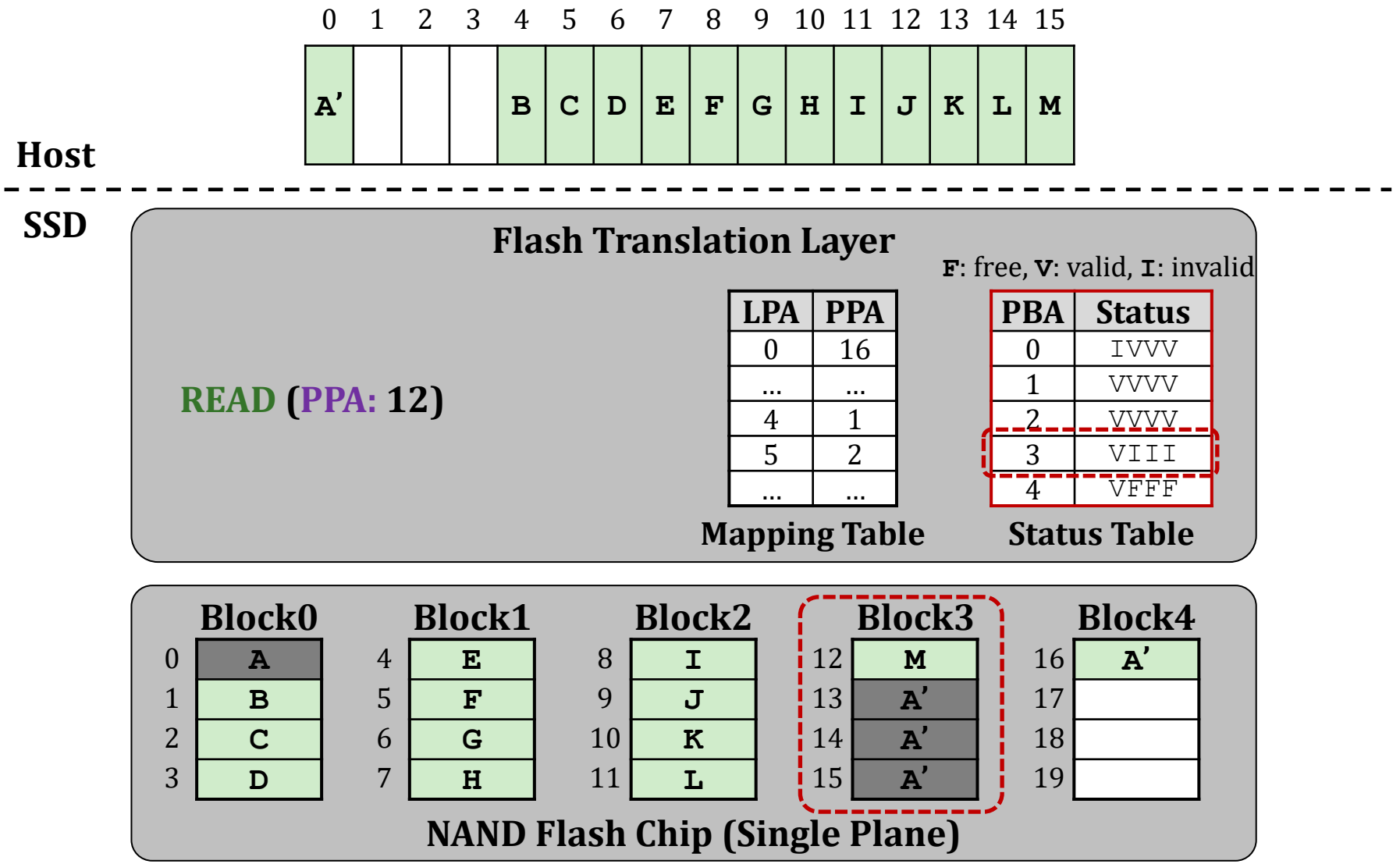


Garbage Collection

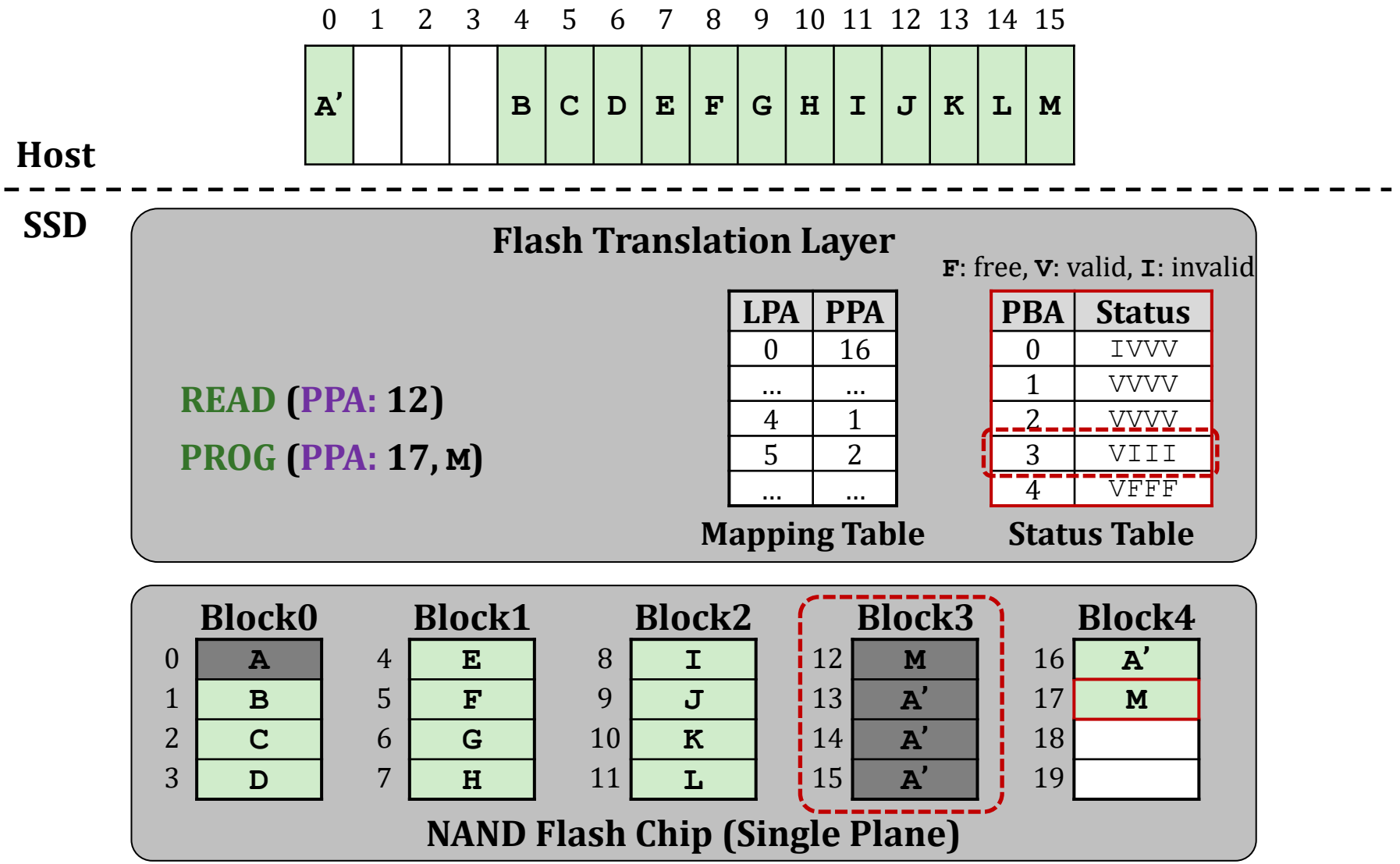
- Reclaims free pages by erasing invalid pages
  - Erase unit: block
  - If a victim block (to erase) has valid pages,
    all the valid pages need to be copied to other free pages
    - Performance overhead: (tREAD + tPROG) × # of valid pages
    - Lifetime overhead: additional writes → P/E-cycle increase

- Greedy victim-selection policy:
  Erases the block with the largest number of invalid pages
  - Needs to maintain # of invalid (or valid) pages for each block

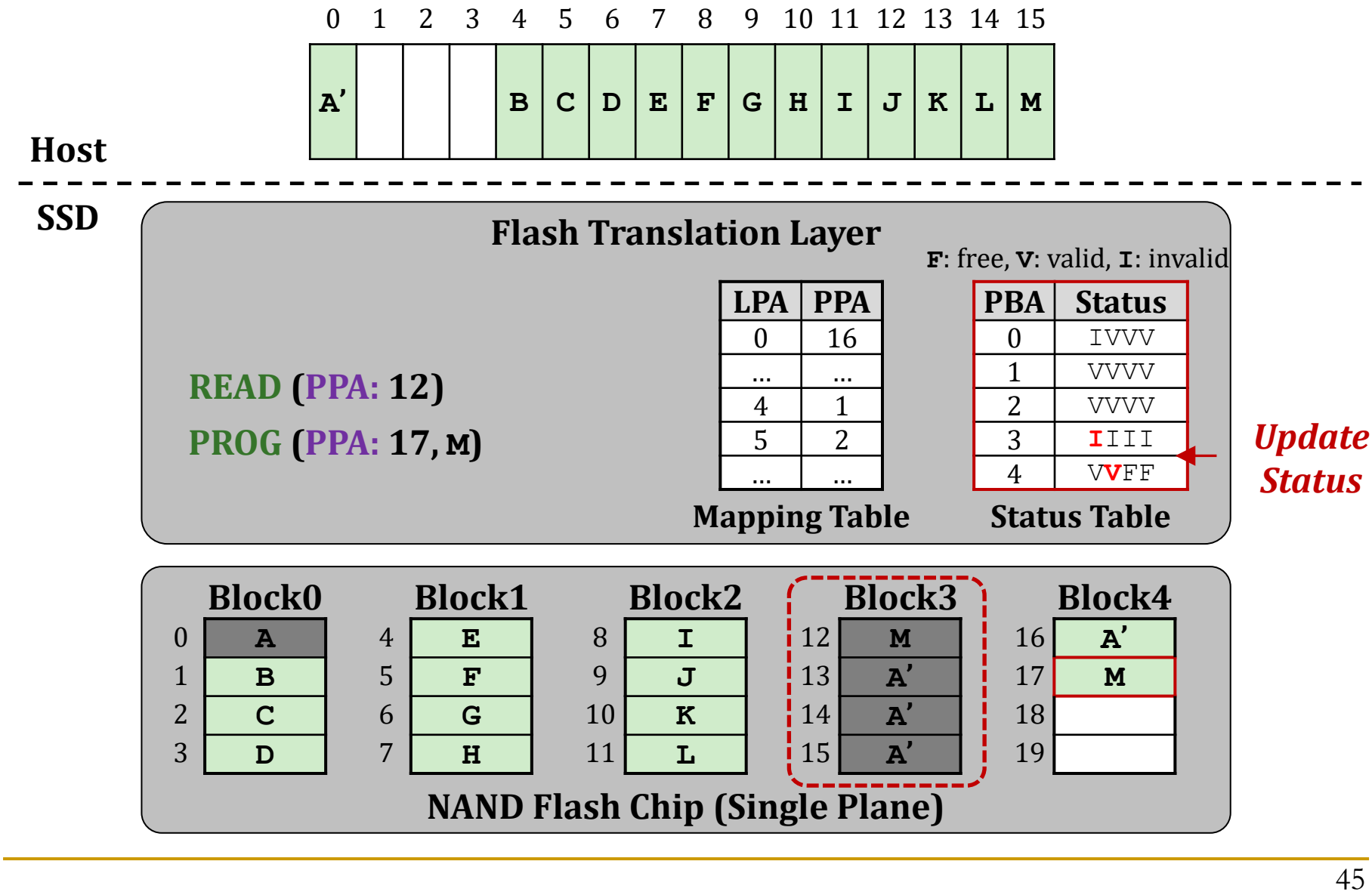
Write Request Handling: Garbage Collection



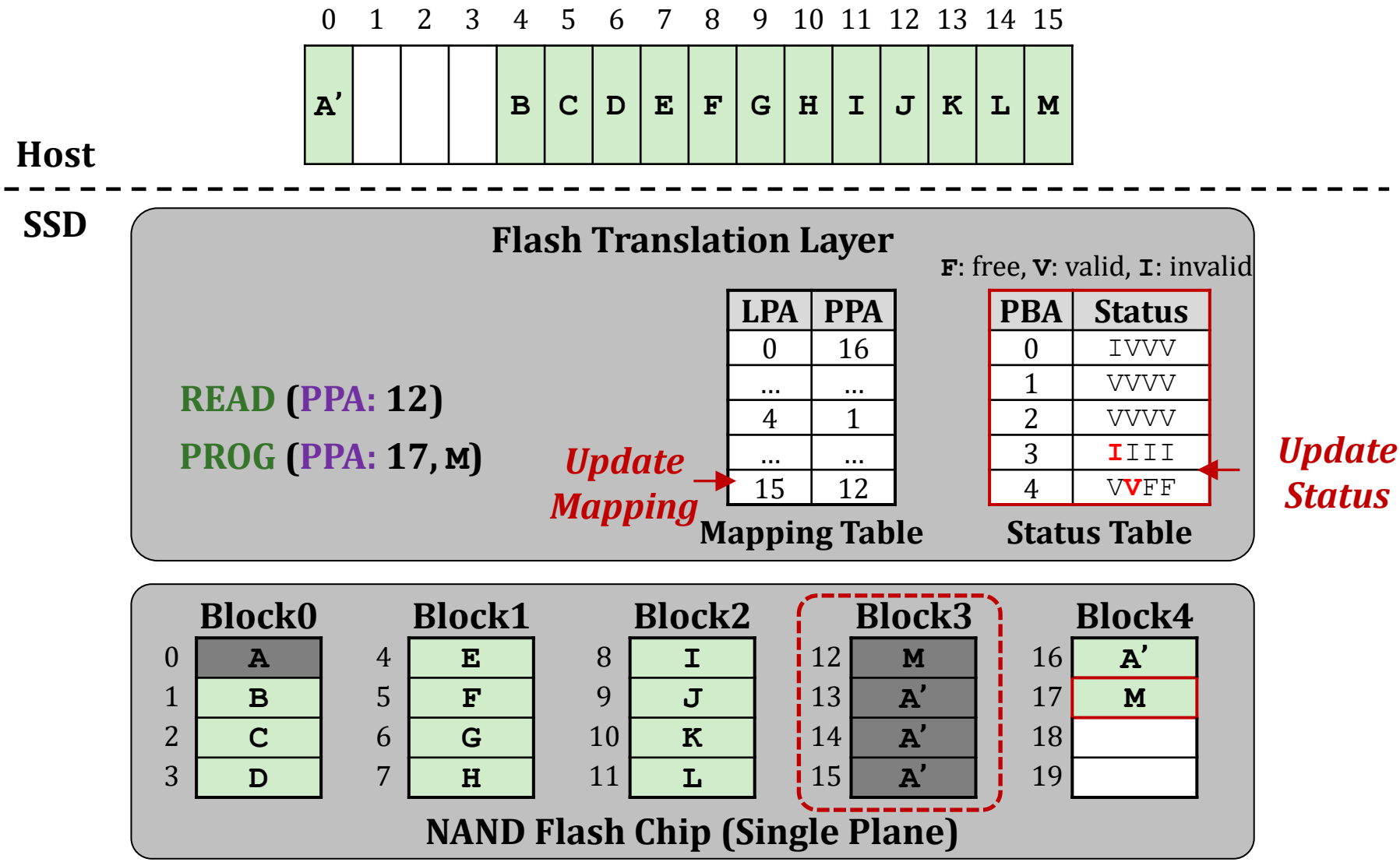
Write Request Handling: Garbage Collection



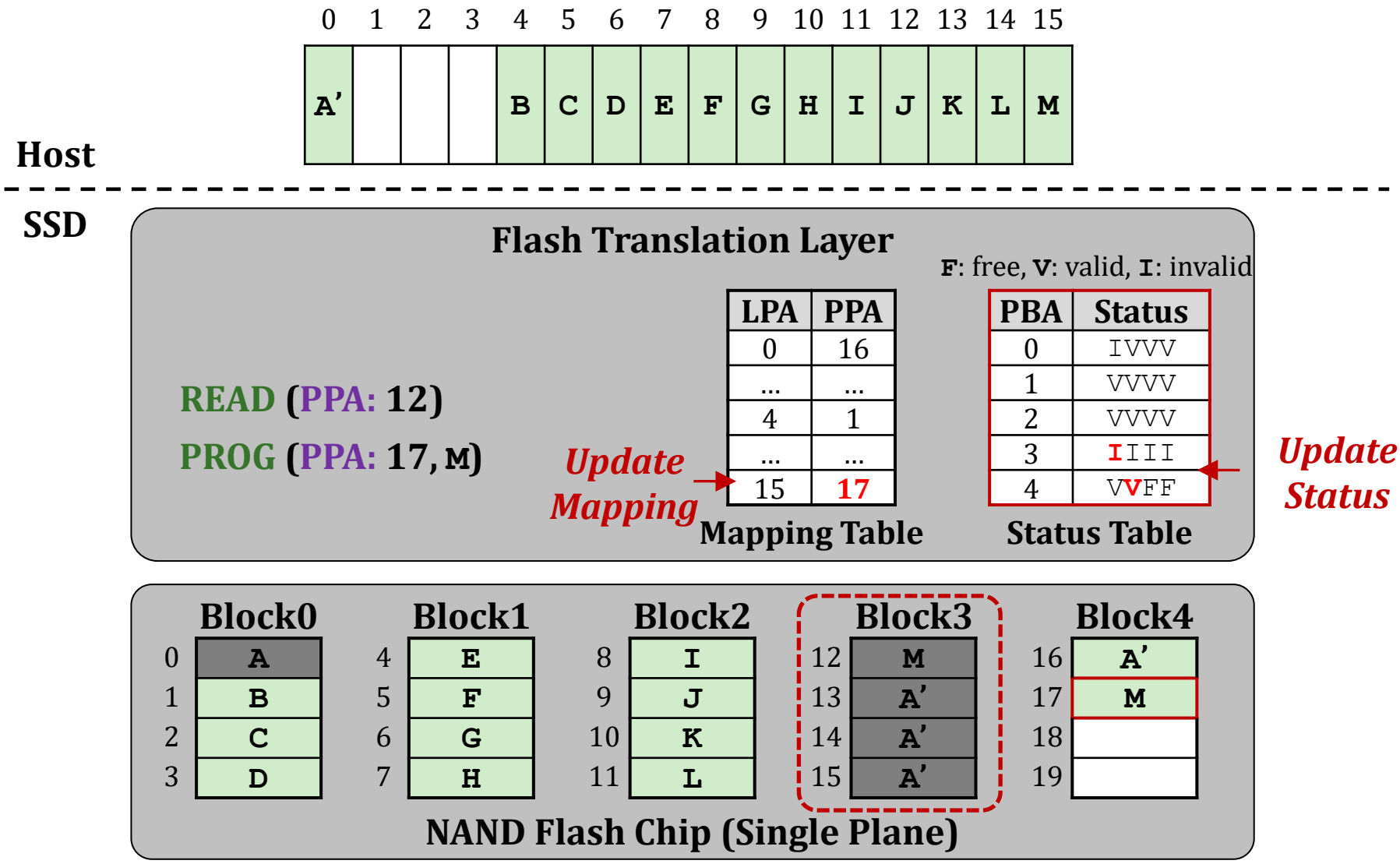
Write Request Handling: Garbage Collection



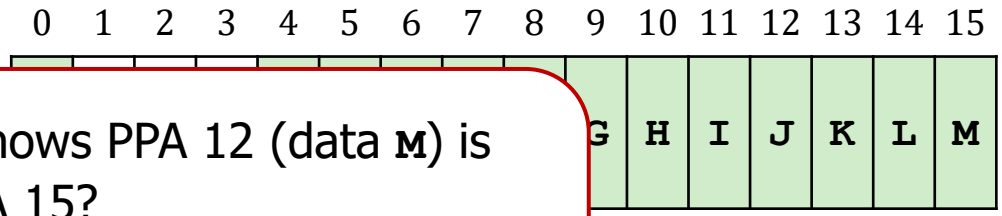
Write Request Handling: Garbage Collection



Write Request Handling: Garbage Collection



Write Request Handling: Garbage Collection



- **Q:** How FTL knows PPA 12 (data **M**) is mapped to LPA 15?
  - Unless it maintains P2L mappings?
- **A:** P2L mapping is stored in each physical page's OOB (Out-of-Band) area

READ (PPA: 12)

PROG (PPA: 17, M)

Update Mapping

on Layer

F: free, **V**: valid, **I**: invalid

LPA	PPA
0	16
...	...
4	1
...	...
15	17

Mapping Table

PBA	Status
0	IVVV
1	VVVV
2	VVVV
3	IIII
4	VVFF

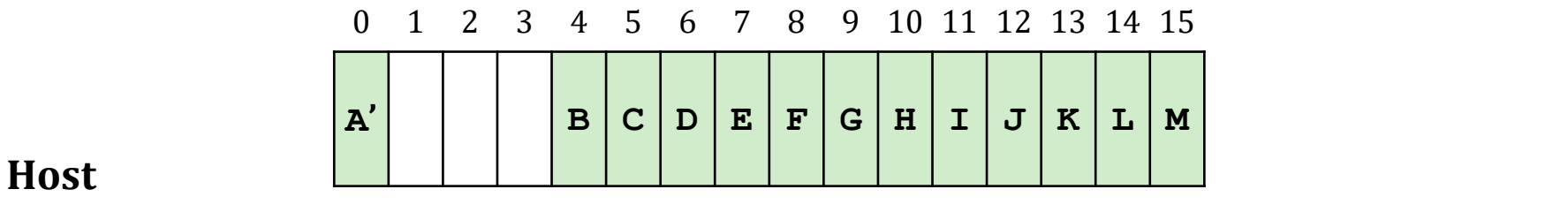
Status Table

Update Status

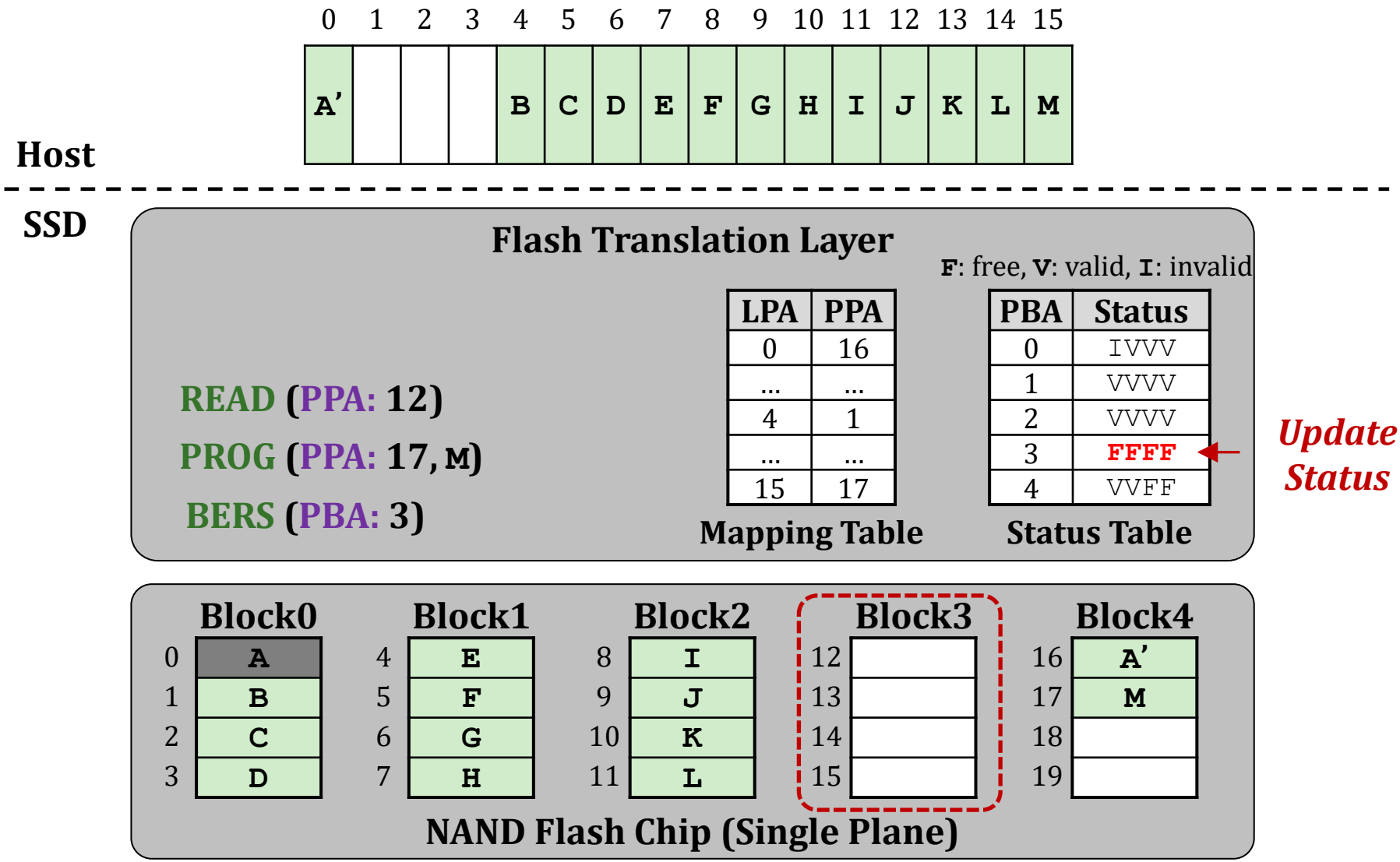
Block0		Block1		Block2		Block3		Block4	
0	A	4	E	8	I	12	M	16	A'
1	B	5	F	9	J	13	A'	17	M
2	C	6	G	10	K	14	A'	18	
3	D	7	H	11	L	15	A'	19	

NAND Flash Chip (Single Plane)

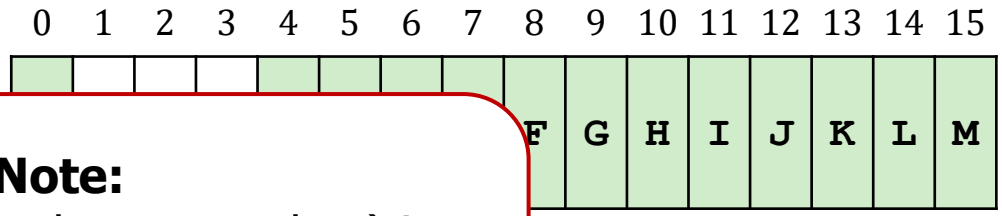
Write Request Handling: Garbage Collection



Write Request Handling: Garbage Collection



Write Request Handling: Garbage Collection



Note:

- Block erasure (and status update) is done just before programming a new page to the block (i.e., lazy erase)
  - Due to the open-block problem

ation Layer

F: free, **V**: valid, **I**: invalid

LPA	PPA
0	16
...	...
4	1
...	...
15	17

Mapping Table

PBA	Status
0	IVVV
1	VVVV
2	VVVV
3	FFFF
4	VVFF

Status Table

Update Status

(PPA: 12)

G (PPA: 17, M)

BERS (PBA: 3)

Block0		Block1		Block2		Block3		Block4	
0	A	4	E	8	I	12		16	A'
1	B	5	F	9	J	13		17	M
2	C	6	G	10	K	14		18	
3	D	7	H	11	L	15		19	

NAND Flash Chip (Single Plane)

Performance Issues

- Garbage collection significantly affects SSD performance
  - High latency: Large block size of modern NAND flash memory
    - Assume 1) a block contains 576 pages,
      2) only 5% of the pages in the victim block are valid
      3) tR = 100 us, tPROG = 700 us, tBERS = 5 ms
      - # of pages to copy = $576 \times 0.05 = 28.8 \rightarrow 28$ pages
      - GC latency > $28 \times (tR + tPROG) + tBERS = 27{,}400$ us
    - Order(s) of magnitude larger latency than tR and tPROG
    - Copy operations are the major contributor (rather than tBERS)
  - If FTL performs GC in an atomic manner, it delays user requests for a significantly long time
    - Long tail latency (performance fluctuation)
    - Noisy neighbor: a read-dominant workload's performance would be significantly affected when running with a write-intensive workload (+ performance fairness problem)

Performance Issues: Mitigation

- TRIM (UNMAP or discard) command
  - Informs FTL of deletion/deallocation of a logical block
  - Allows FTL to skip copy of obsolete (i.e., invalid) data

- Background GC: Exploits SSD idle time
  - Challenge: how to accurately predict SSD idle time
  - Premature GC: copied pages could have been invalidated by the host system

- Progressive GC: Divide GC process into subtasks
  - e.g., copying 28 pages → (copying 1 page + servicing user request) × 28
  - Effective at decreasing tail latency

Required Materials

- Address Mapping
  - Aayush Gupta, Yongjae Kim, and Bhuvan Urgaonkar, "DFTL: A Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings," In ASPLOS 2009.

Recommend Materials

- Cache read & Read-retry
  - Jisung Park, Myungsuk Kim, Lois Orosa, Jihong Kim, and Onur Mutlu, "Reducing Solid-State Drive Read Latency by Optimizing Read-Retry," In ASPLOS 2021.

- Program & Erase Suspension
  - Guanying Wu and Xunbin He, "Reducing SSD Read Latency via NAND Flash Program and Erase Suspension," In USENIX FAST 2012.
  - Shine Kim, Jonghyun Bae, Hakbeom Jand, Wenjing Jin, Jeonghun Gong, Seungyeon Lee, Tae Jun Ham, and Jae W. Lee, "Practical Erase Suspension for Modern Low-latency SSDs," In USENIX ATC 2019.

P&S Modern SSDs

Advanced NAND Flash Commands & Address Translation

Dr. Jisung Park
Prof. Onur Mutlu
ETH Zürich
Spring 2022
4 April 2022