Write Amplification and WOM Codes in Flash Memories

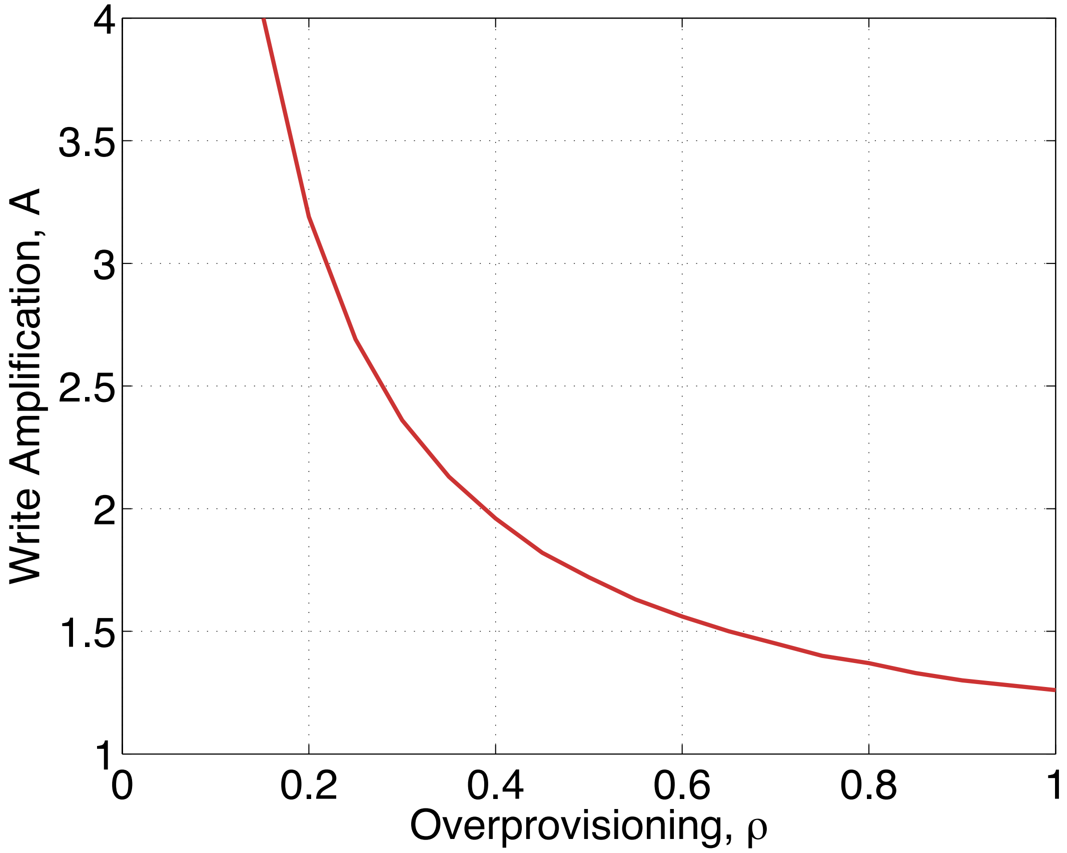
Luojie Xiang
Purdue University
West Lafayette, IN, USA
xiang7@purdue.edu

Brian M. Kurkoski
Japan Advanced Institute of Science and Technology
Nomi, Japan

Eitan Yaakobi
California Institute of Technology
Pasadena, CA, USA

Non-Volatile Memories Workshop, 2013
University of California San Diego

Background



Write amplification

Flash memories unique problem:

Unneeded writes are due to:

- block erase,
- page write

architecture of flash memories.

Mitigated by **overprovisioning ρ**

- allocating more physical memory than logical memory

WOM Codes

WOM codes allow rewriting flash memories without erasing.

- Extend the lifetime of flash memories

We show that WOM codes can also reduce write amplification

Overview

Agarwal & Marrow [Globe2010] gave an analytic expression for write amplification

We give

- an improved-accuracy expression write amplification
- analytic expression for write amplification when using WOM codes
- conditions when WOM codes reduce write amplification

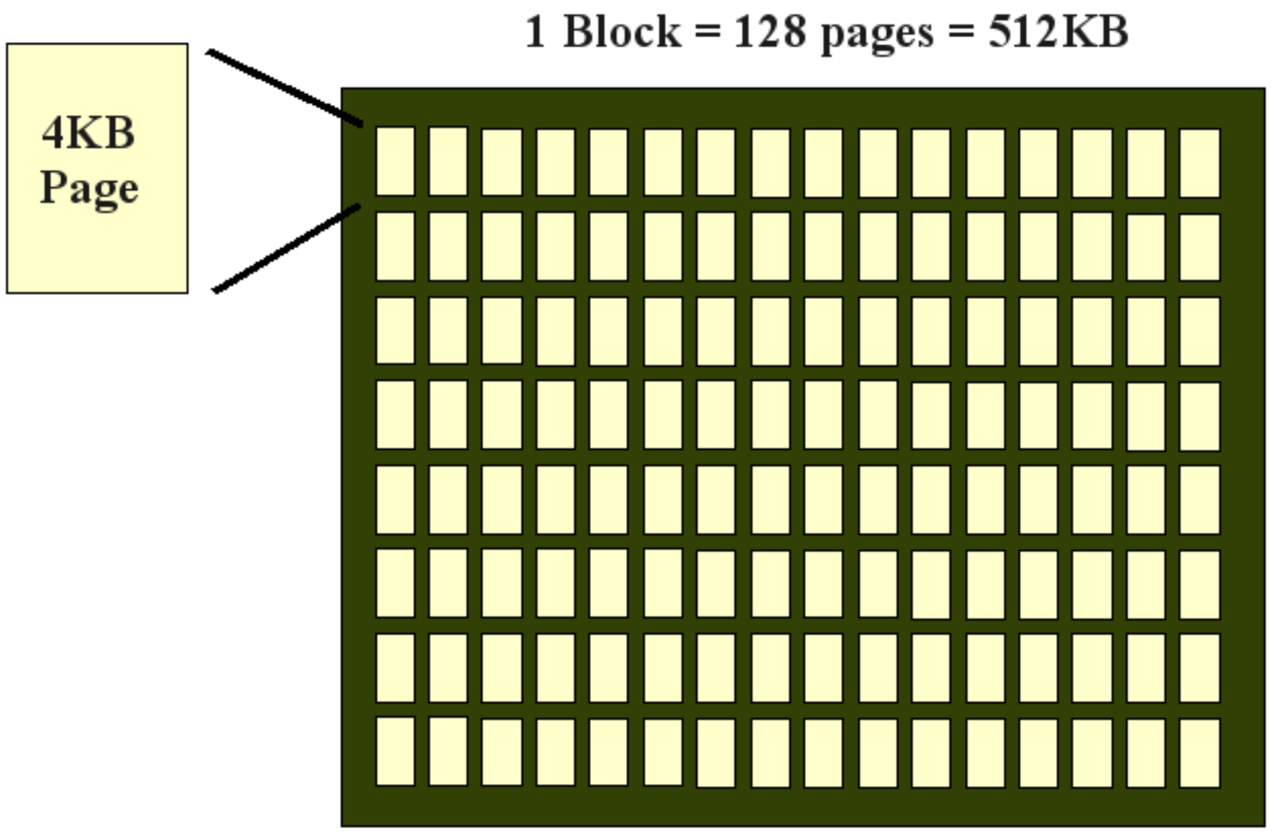
See also Desnoyers et al.

Caveats

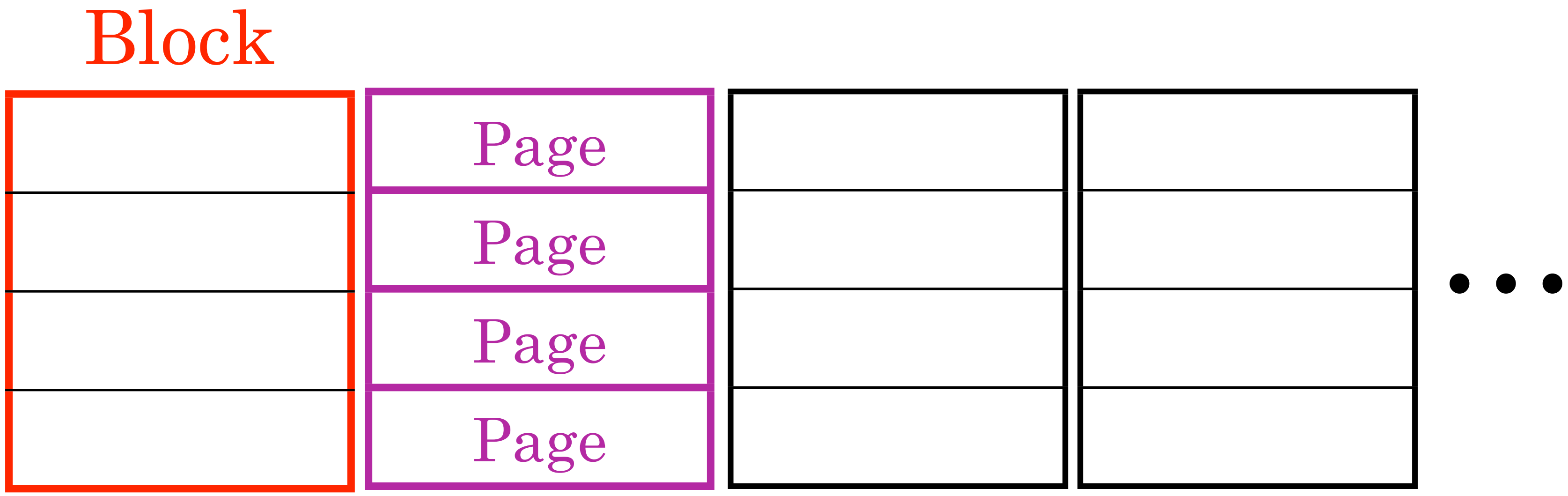
- The memory system model is idealized
  - random writes on the user space
  - logical memory (user memory) is always full
- Explain write amplification as a coding theorist understands it

Organization of flash memory

- Organization of flash memory
  - Contains thousands of blocks
  - A block contains typically 64 pages
  - A page is typically 4 KB, smallest unit
- Operations on flash memory
  - Page-level write operations
  - Can write only to empty blocks
  - Block-level erase operations

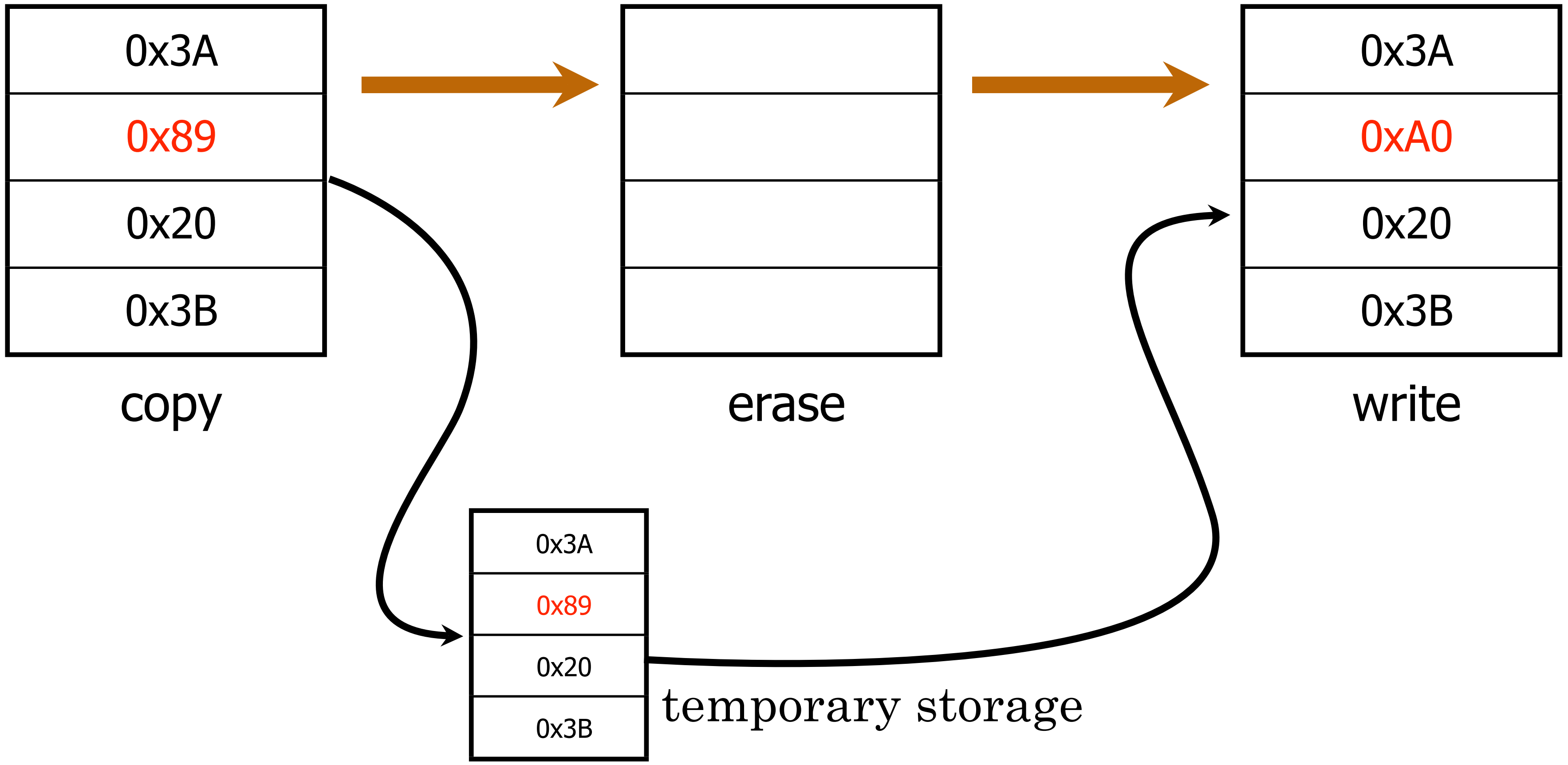


http://www.linux-mag.com/id/7590/



Flash memory: Write Amplification

- Flash memories are page write, block erase
- To change one page, must copy-erase-write
- "Write amplification" Changing one page requires 64 page writes!
- Undesirable, system performance and memory longevity



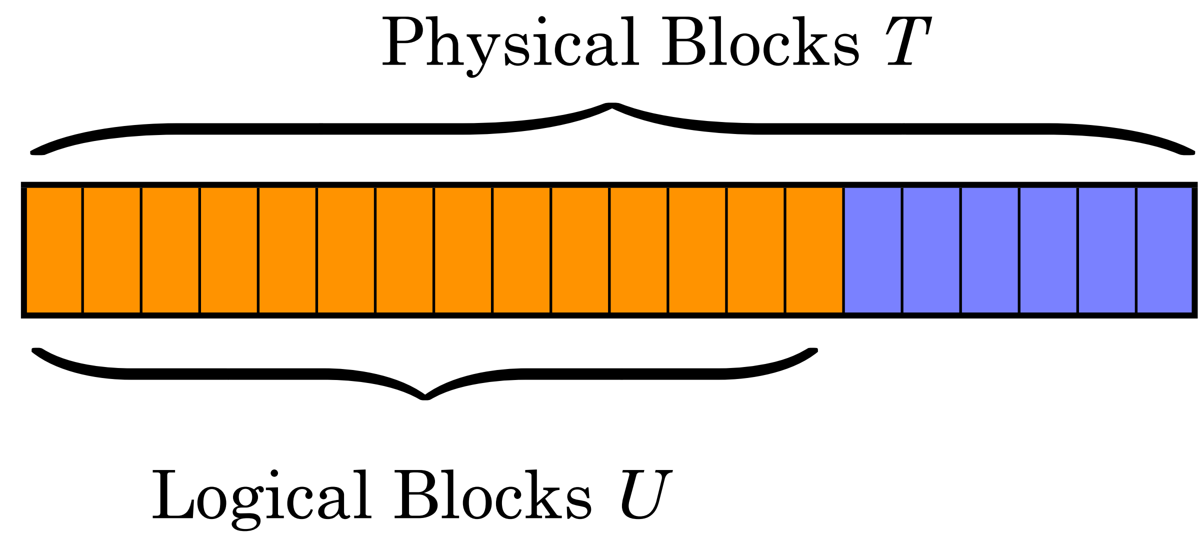
System

Write Amplification and Overprovisioning

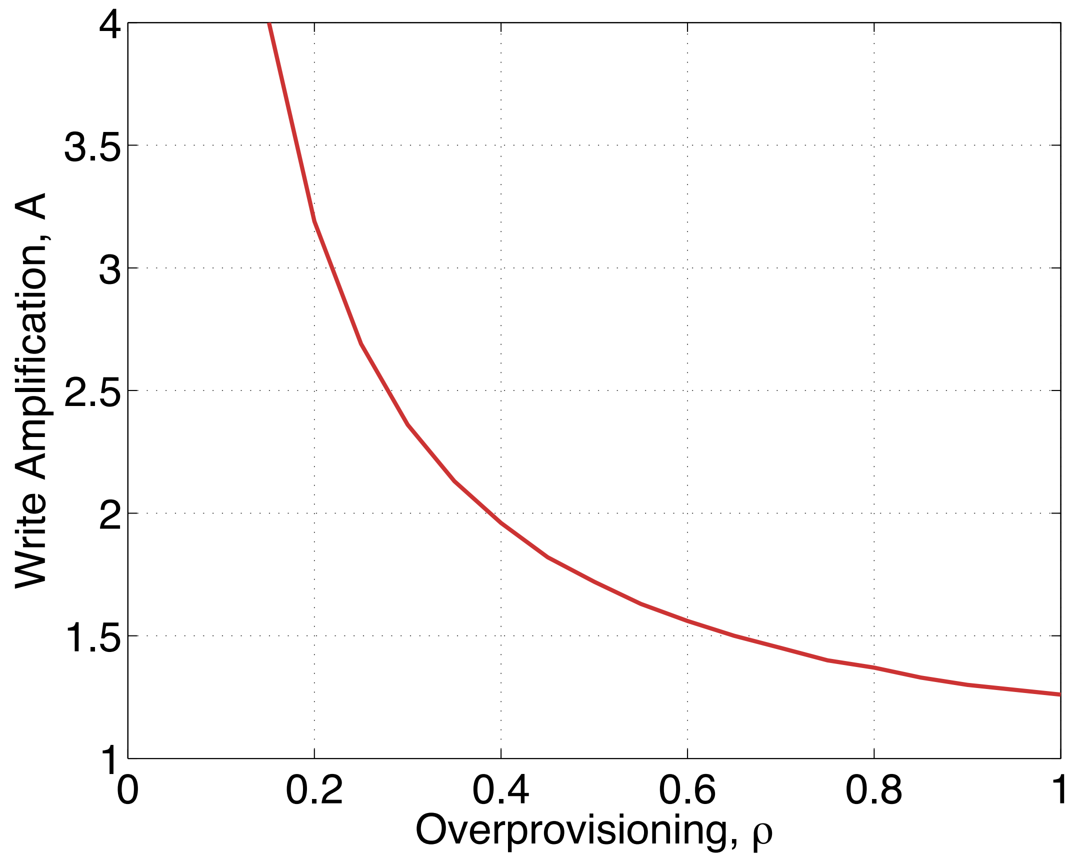
- **Problem:** Write Amplification

$$\text{Write Amplification } A = \frac{\text{Number of Physical Writes}}{\text{Number of Logical Writes}}$$

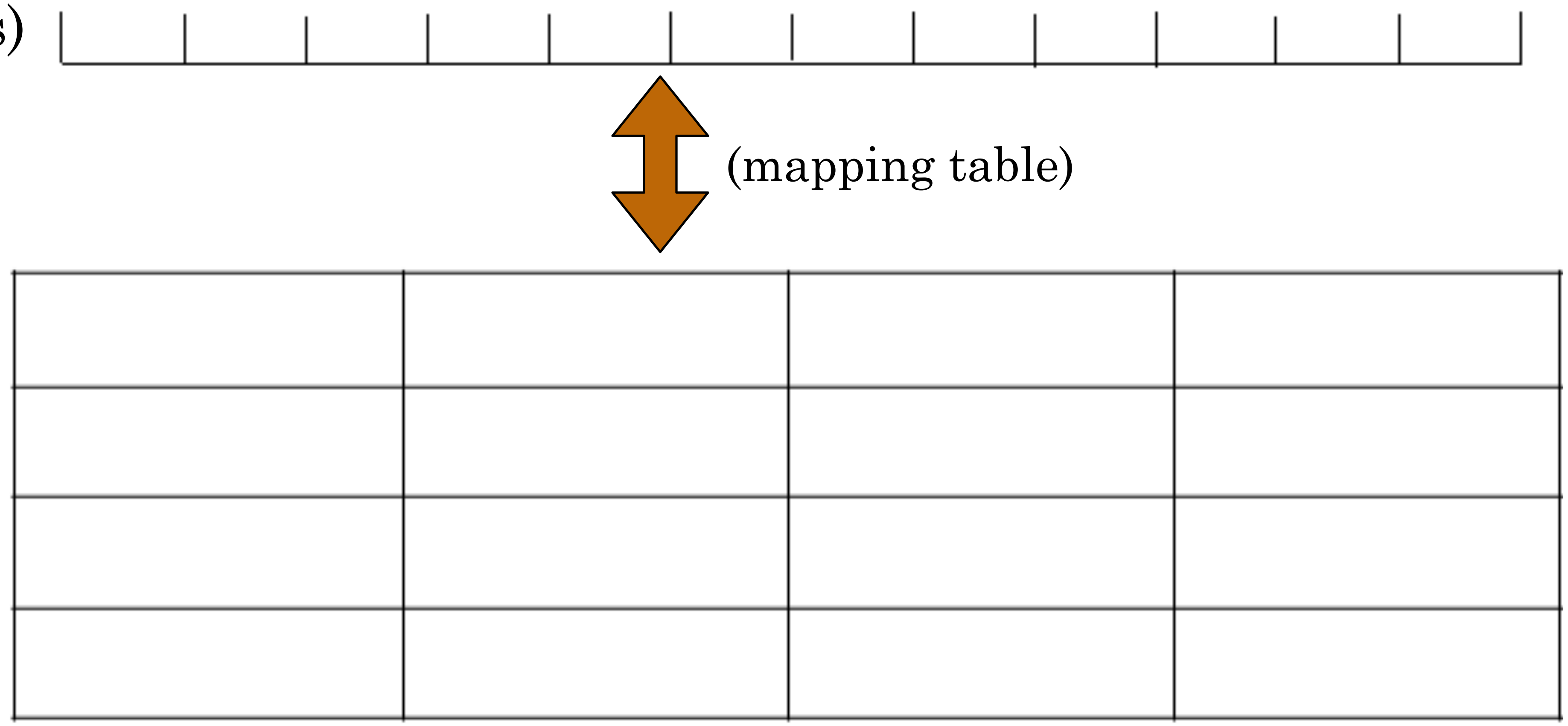
- **Solution:** Overprovisioning
  - More physical memory than logical memory
  - (some physical memory the user cannot see)



$$\text{Overprovisioning factor } \rho = \frac{T-U}{U}$$



Example of Writing Flash Memory



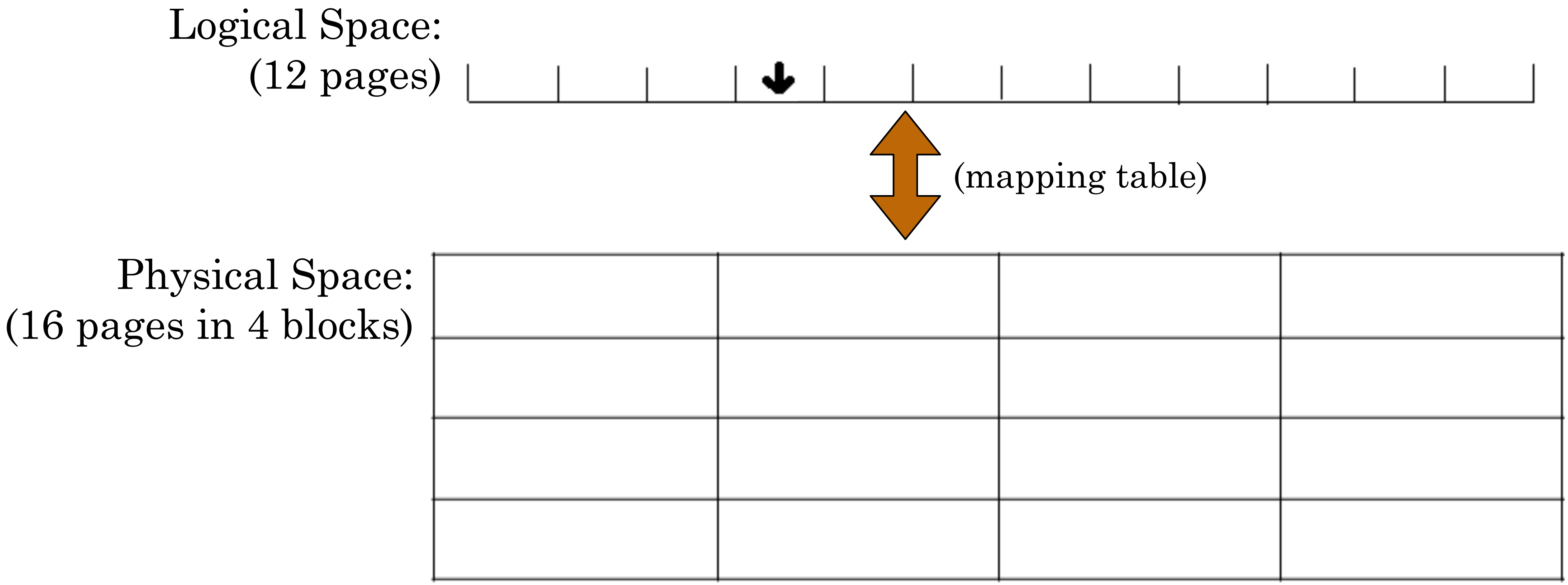
Logical Space:
(12 pages)

(mapping table)

Physical Space:
(16 pages in 4 blocks)

Initial condition: Start with an empty memory
User writes uniformly and randomly distributed on user space
stationary condition: Logical memory is always full (worst case)

Example of Writing Flash Memory



Logical Space:
(12 pages)

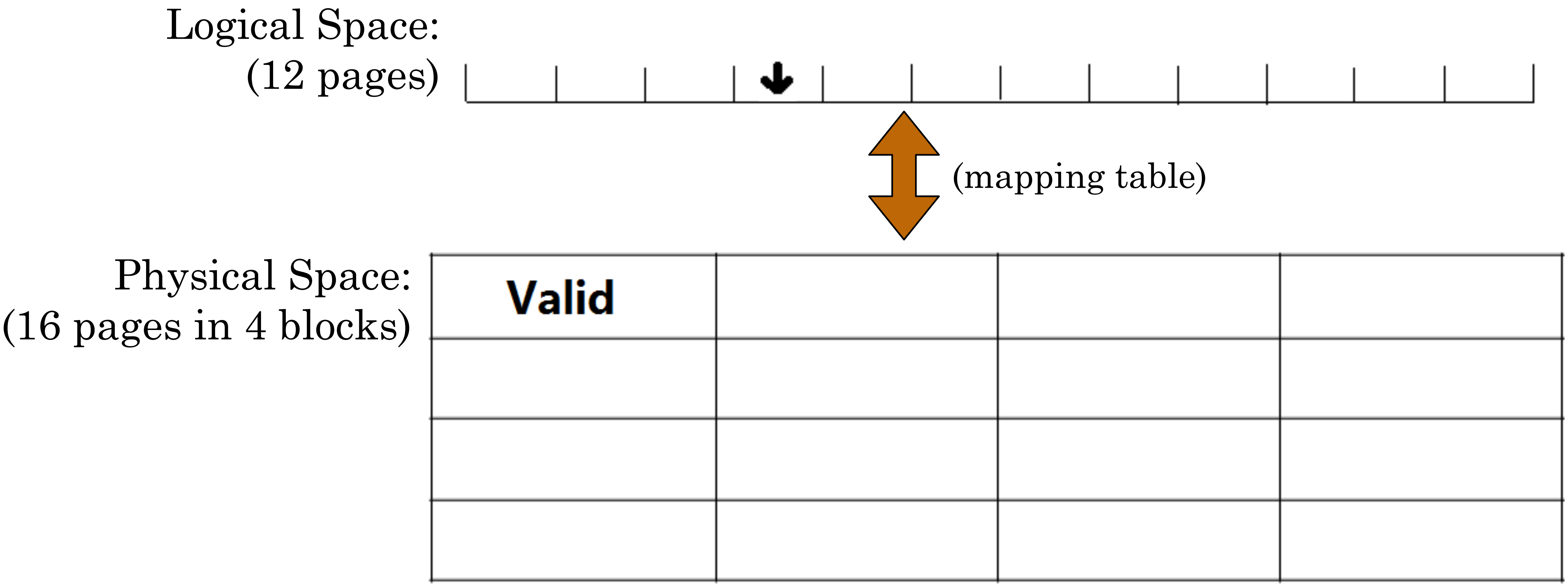
(mapping table)

Physical Space:
(16 pages in 4 blocks)

Initial condition: Start with an empty memory
User writes uniformly and randomly distributed on user space
stationary condition: Logical memory is always full (worst case)

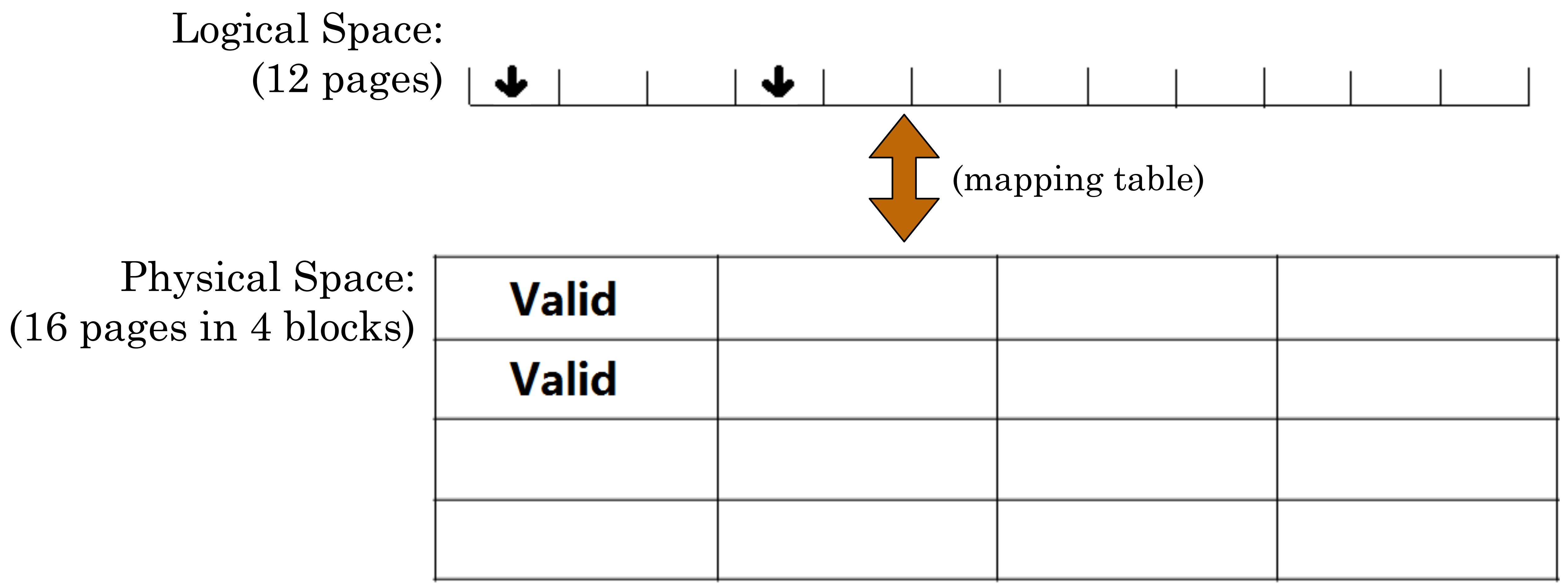
System

Example of Writing Flash Memory



Initial condition: Start with an empty memory
User writes uniformly and randomly distributed on user space
stationary condition: Logical memory is always full (worst case)

Example of Writing Flash Memory



Logical Space:
(12 pages)

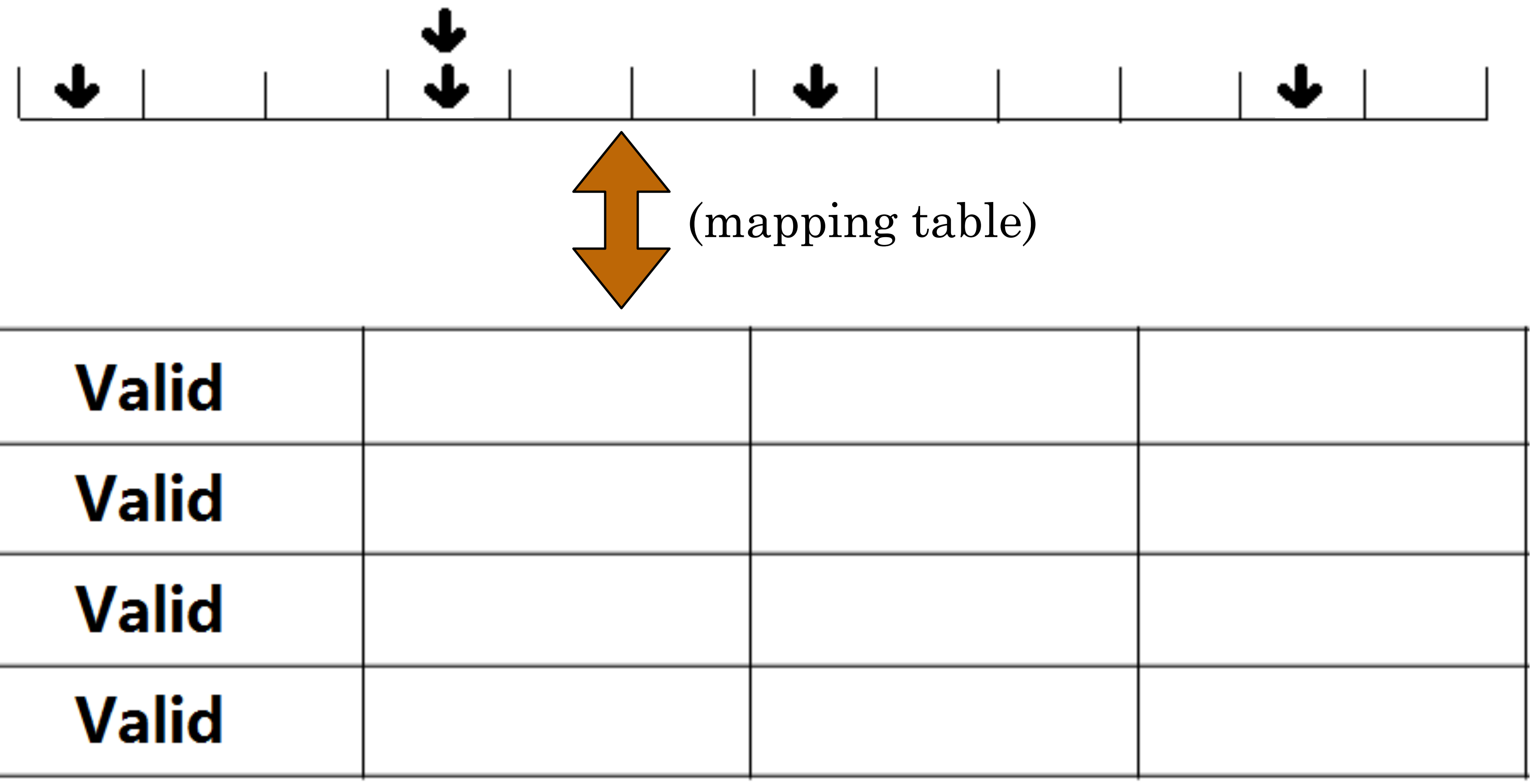
(mapping table)

Physical Space:
(16 pages in 4 blocks)

Valid			
Valid			

Initial condition: Start with an empty memory
User writes uniformly and randomly distributed on user space
stationary condition: Logical memory is always full (worst case)

Example of Writing Flash Memory



Logical Space:
(12 pages)

(mapping table)

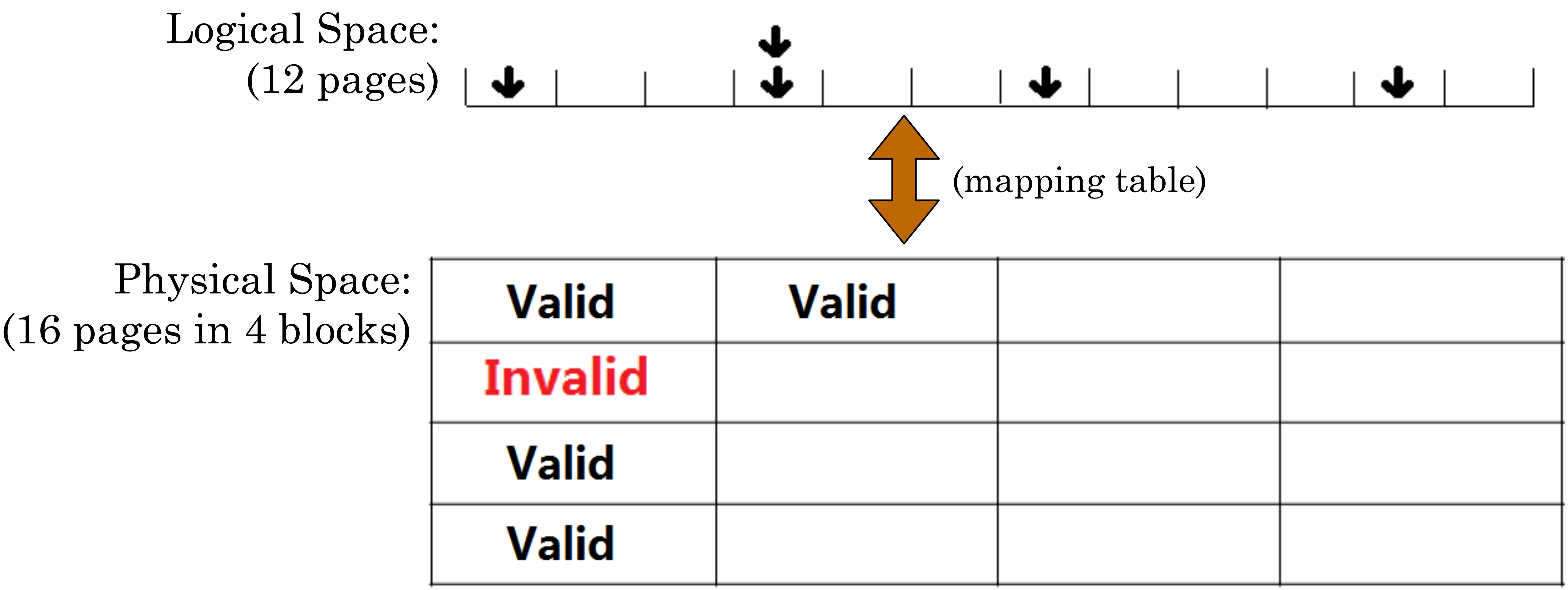
Physical Space:
(16 pages in 4 blocks)

Valid			
Valid			
Valid			
Valid			

Initial condition: Start with an empty memory
User writes uniformly and randomly distributed on user space
stationary condition: Logical memory is always full (worst case)

System

Example of Writing Flash Memory



Logical Space:
(12 pages)

(mapping table)

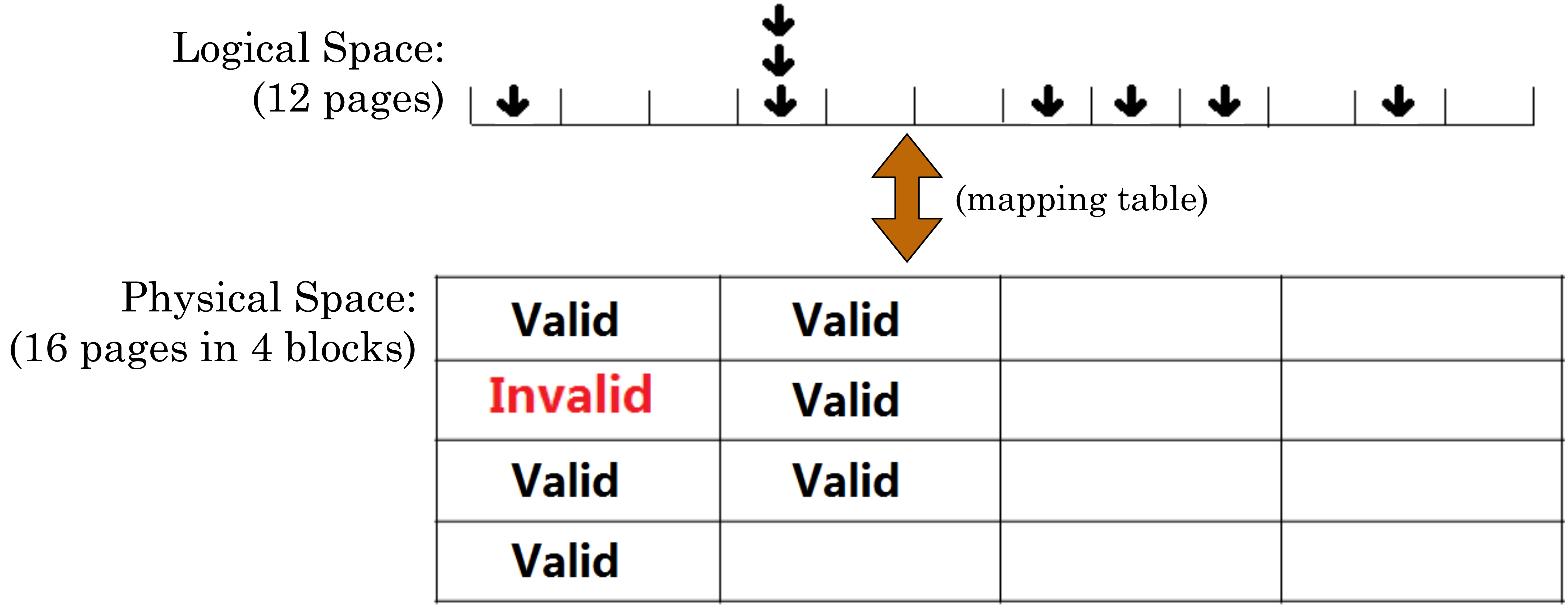
Physical Space:
(16 pages in 4 blocks)

Valid	Valid		
Invalid			
Valid			
Valid			

Initial condition: Start with an empty memory
User writes uniformly and randomly distributed on user space
stationary condition: Logical memory is always full (worst case)

System

Example of Writing Flash Memory



Logical Space:
(12 pages)

(mapping table)

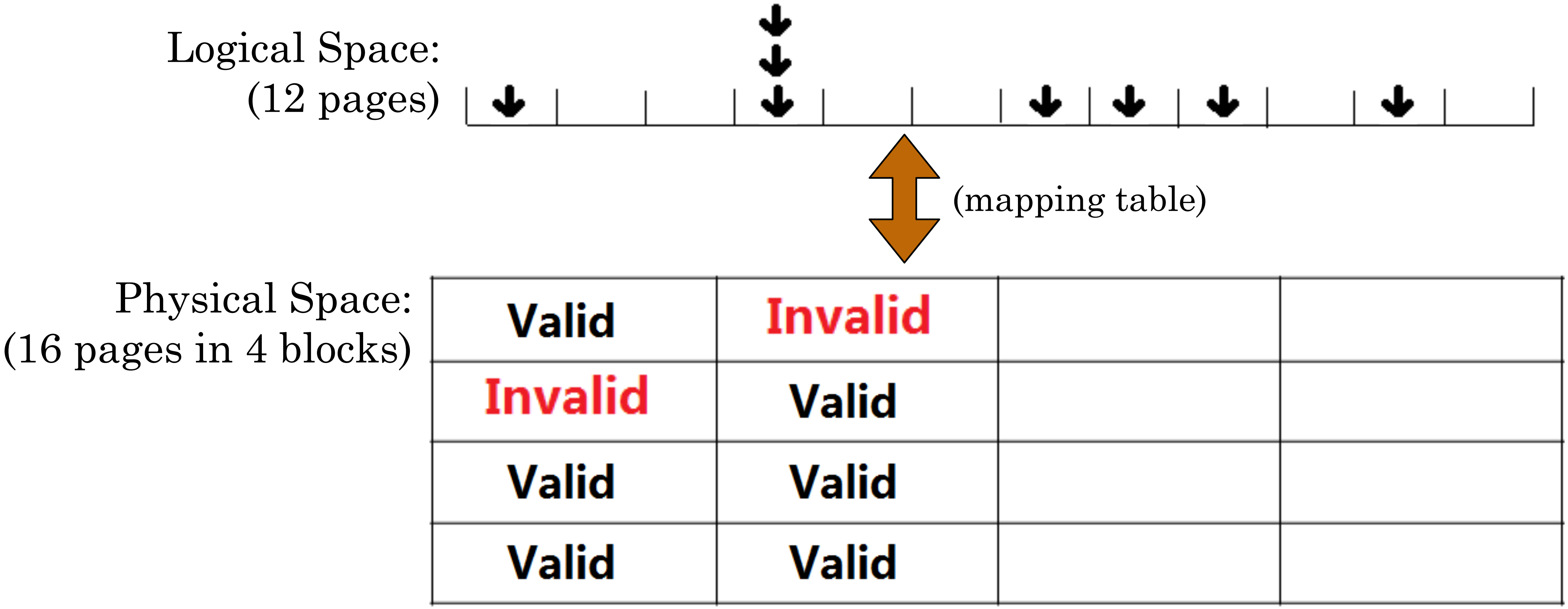
Physical Space:
(16 pages in 4 blocks)

Valid	Valid		
Invalid	Valid		
Valid	Valid		
Valid			

Initial condition: Start with an empty memory
User writes uniformly and randomly distributed on user space
stationary condition: Logical memory is always full (worst case)

System

Example of Writing Flash Memory



Logical Space:
(12 pages)

(mapping table)

Physical Space:
(16 pages in 4 blocks)

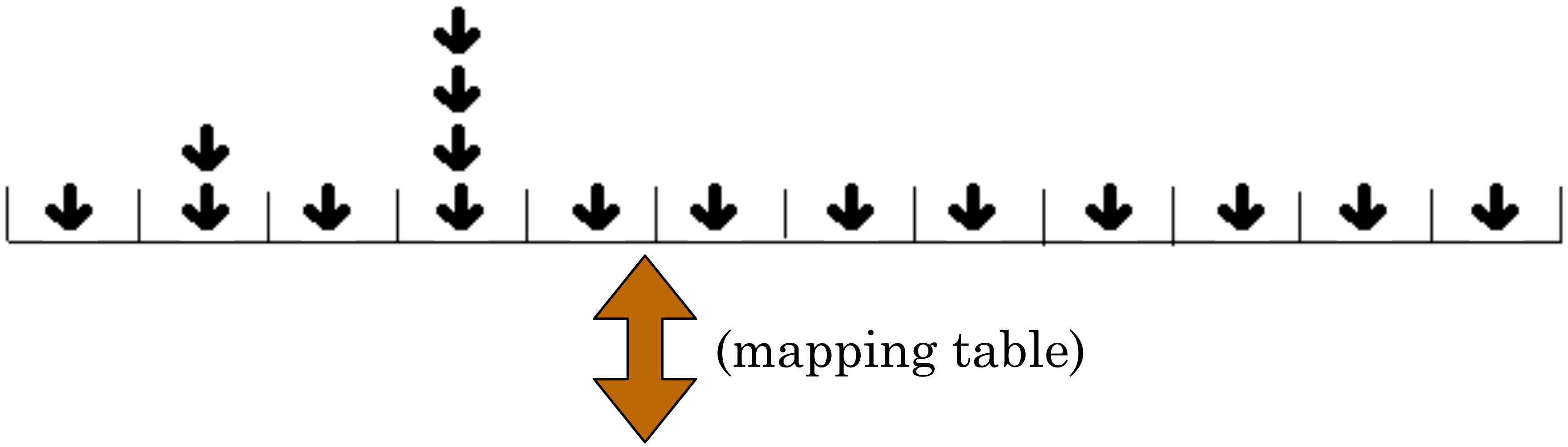
Valid	Invalid		
Invalid	Valid		
Valid	Valid		
Valid	Valid		

Initial condition: Start with an empty memory
User writes uniformly and randomly distributed on user space
stationary condition: Logical memory is always full (worst case)

System

Example of Writing Flash Memory

Logical Space:
(12 pages)



Physical Space:
(16 pages in 4 blocks)

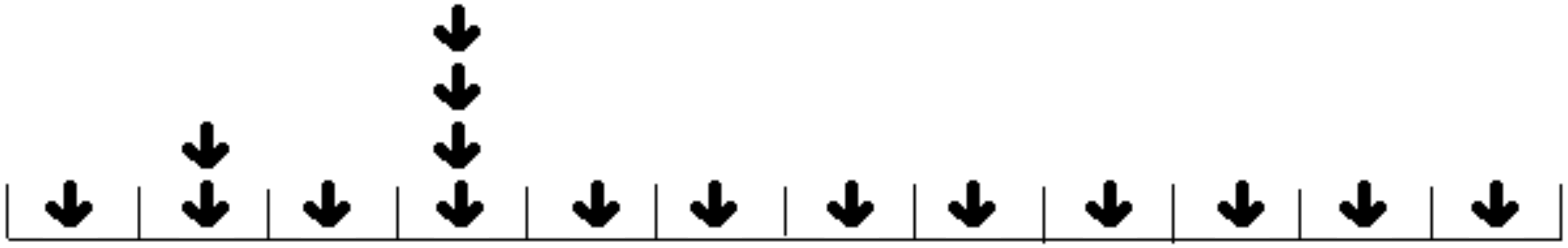
Valid	Invalid	Valid	Valid
Invalid	Valid	Valid	Valid
Valid	Valid	Invalid	Valid
Valid	Invalid	Valid	Valid

Initial condition: Start with an empty memory
User writes uniformly and randomly distributed on user space
stationary condition: Logical memory is always full (worst case)

System

Garbage Collection

Logical Space:
(12 pages)



Physical Space:
(16 pages in 4 blocks)

Valid	Invalid	Valid	Valid
Invalid	Valid	Valid	Valid
Valid	Valid	Invalid	Valid
Valid	Invalid	Valid	Valid

Time to erase

Greedy Garbage collection:

- Block with most invalid pages

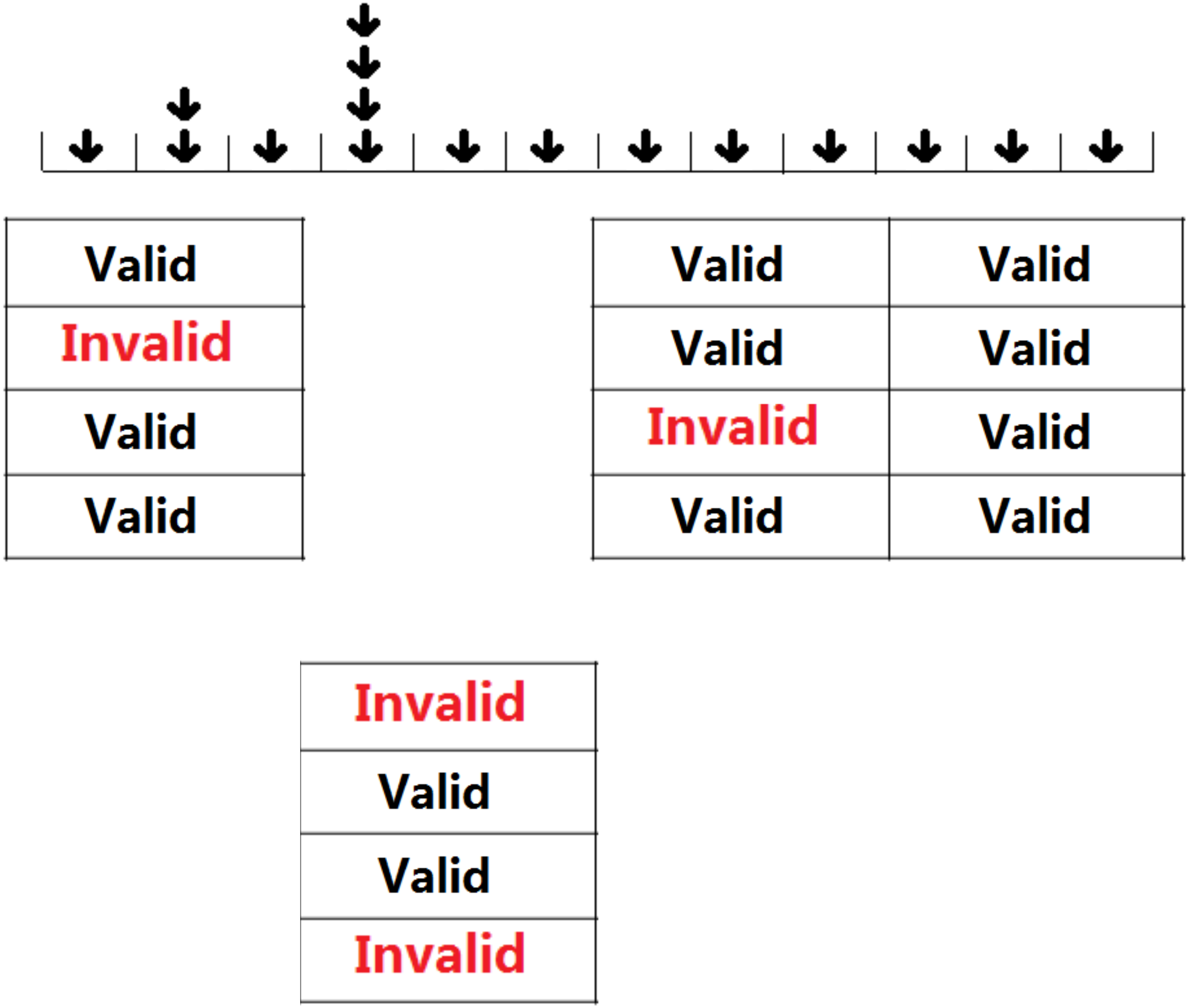
Only two writes needed

System

Garbage Collection

Logical Space:
(12 pages)

Physical Space:
(16 pages in 4 blocks)



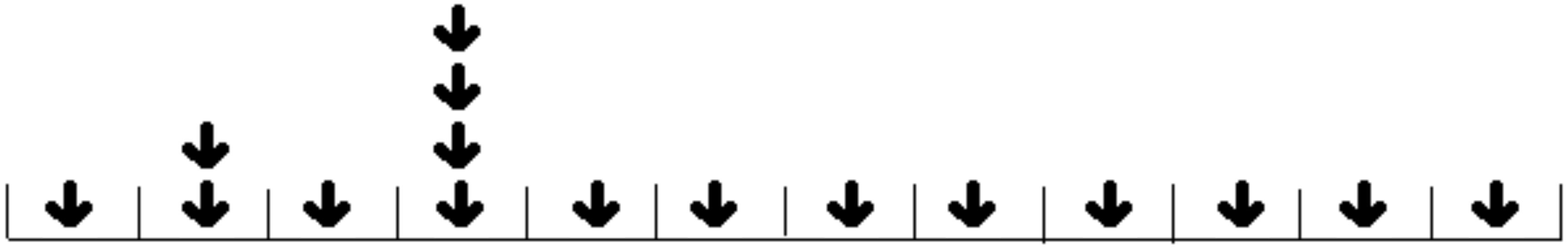
Valid
Invalid
Valid
Valid

Valid	Valid
Valid	Valid
Invalid	Valid
Valid	Valid

Invalid
Valid
Valid
Invalid

Garbage Collection

Logical Space:
(12 pages)



Physical Space:
(16 pages in 4 blocks)

Valid	Valid	Valid
Invalid	Valid	Valid
Valid	Invalid	Valid
Valid	Valid	Valid

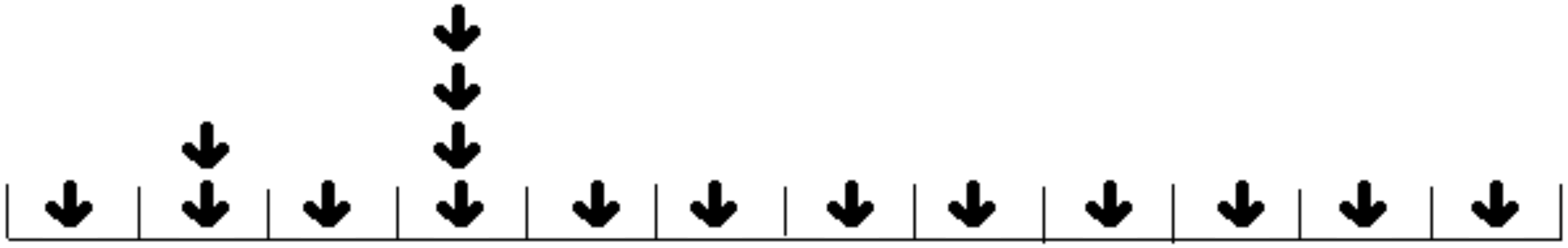
← "Block queue": Older blocks/more invalid pages

Invalid
Valid
Valid
Invalid

System

Garbage Collection

Logical Space:
(12 pages)



Physical Space:
(16 pages in 4 blocks)

Valid	Valid	Valid
Invalid	Valid	Valid
Valid	Invalid	Valid
Valid	Valid	Valid

← “Block queue”: Older blocks/more invalid pages

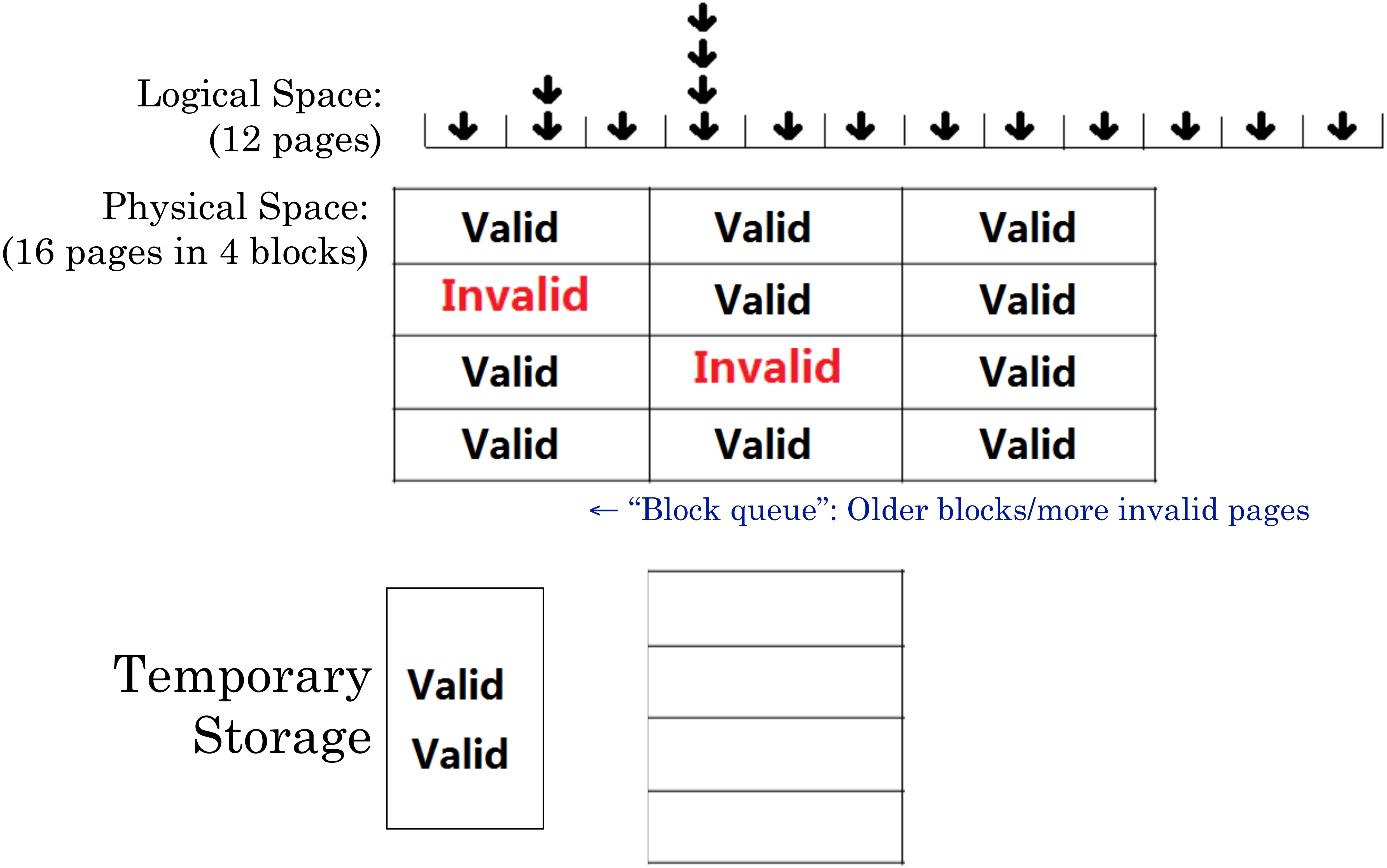
Temporary Storage

Valid
Valid

Invalid
Invalid

System

Garbage Collection



Logical Space:
(12 pages)

Physical Space:
(16 pages in 4 blocks)

Valid	Valid	Valid
Invalid	Valid	Valid
Valid	Invalid	Valid
Valid	Valid	Valid

← “Block queue”: Older blocks/more invalid pages

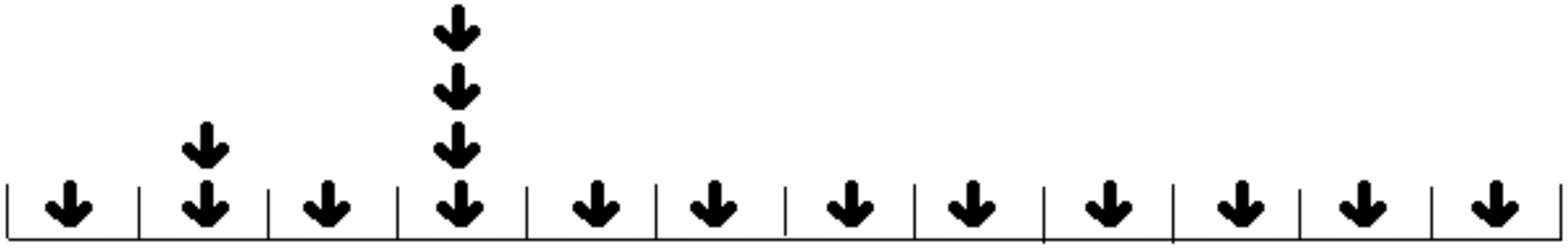
Temporary Storage

Valid
Valid

System

Garbage Collection

Logical Space:
(12 pages)



Physical Space:
(16 pages in 4 blocks)

Valid	Valid	Valid	
Invalid	Valid	Valid	
Valid	Invalid	Valid	
Valid	Valid	Valid	

← “Block queue”: Older blocks/more invalid pages

Temporary Storage

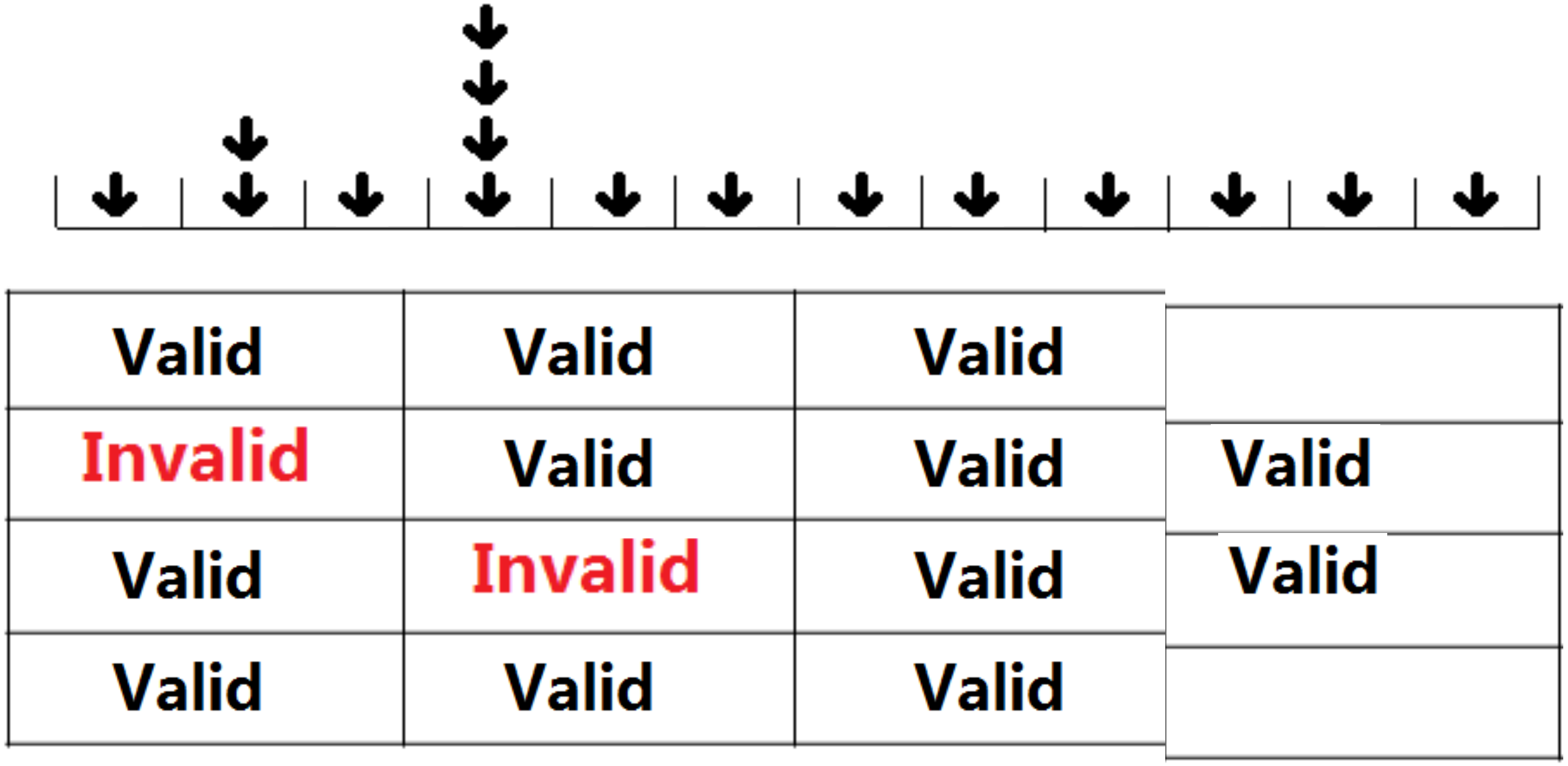
Valid
Valid

System

Garbage Collection

Logical Space:
(12 pages)

Physical Space:
(16 pages in 4 blocks)



Valid	Valid	Valid	
Invalid	Valid	Valid	Valid
Valid	Invalid	Valid	Valid
Valid	Valid	Valid	

Temporary
Storage

Time to erase

Greedy Garbage collection:

- Block with most invalid pages

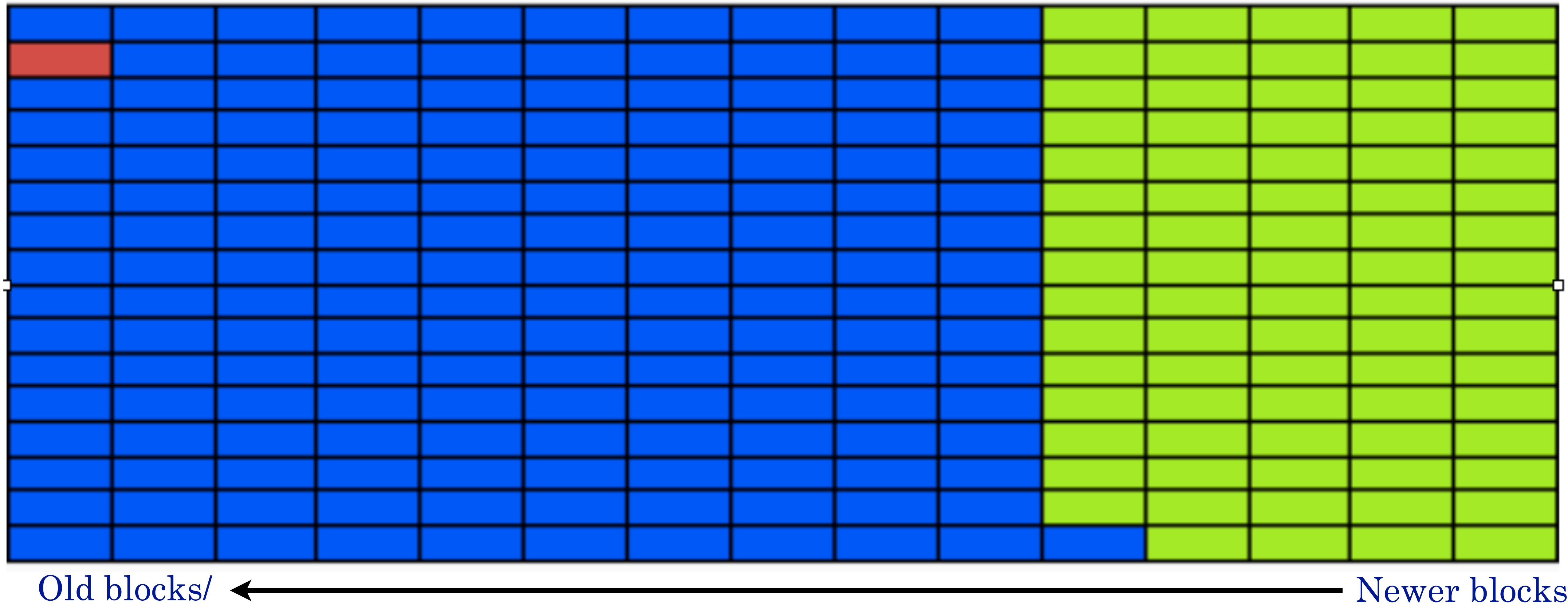
Only two writes needed

Block Queue Model

🟦 Valid page

🟥 Invalid page

🟩 Free page



Old blocks/
Many invalid
pages

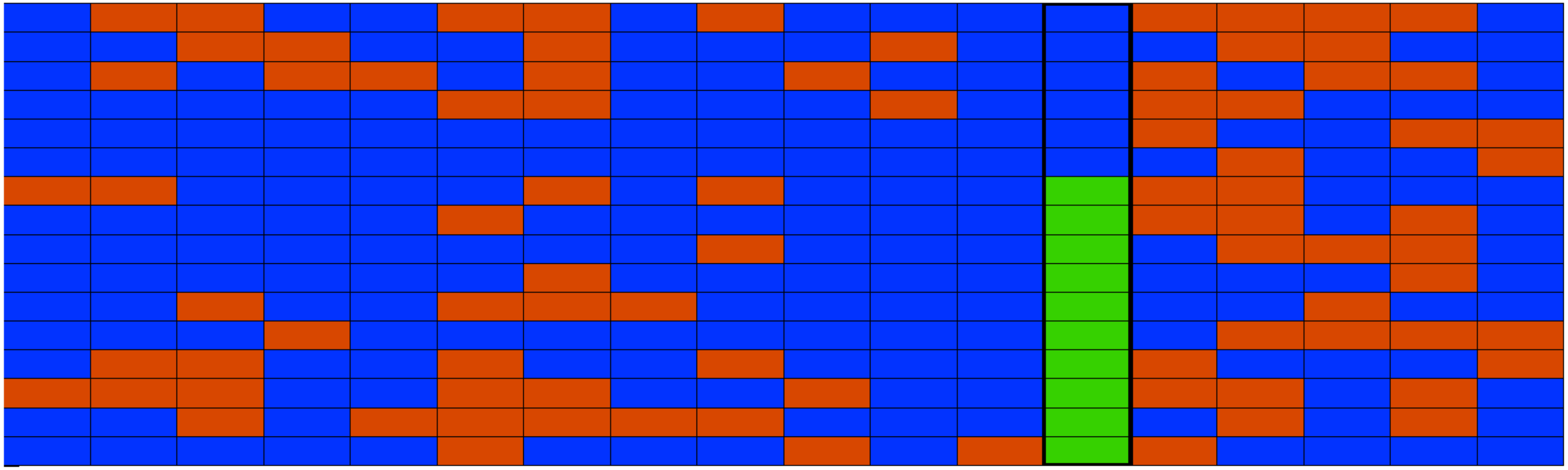
Newer blocks/
Fewer invalid
pages

Garbage Collection Animation
No WOM Codes

🟩 Free page

🟦 Valid page

🟥 Invalid page



Animation of garbage collection:
http://bit.ly/ZPdMn0

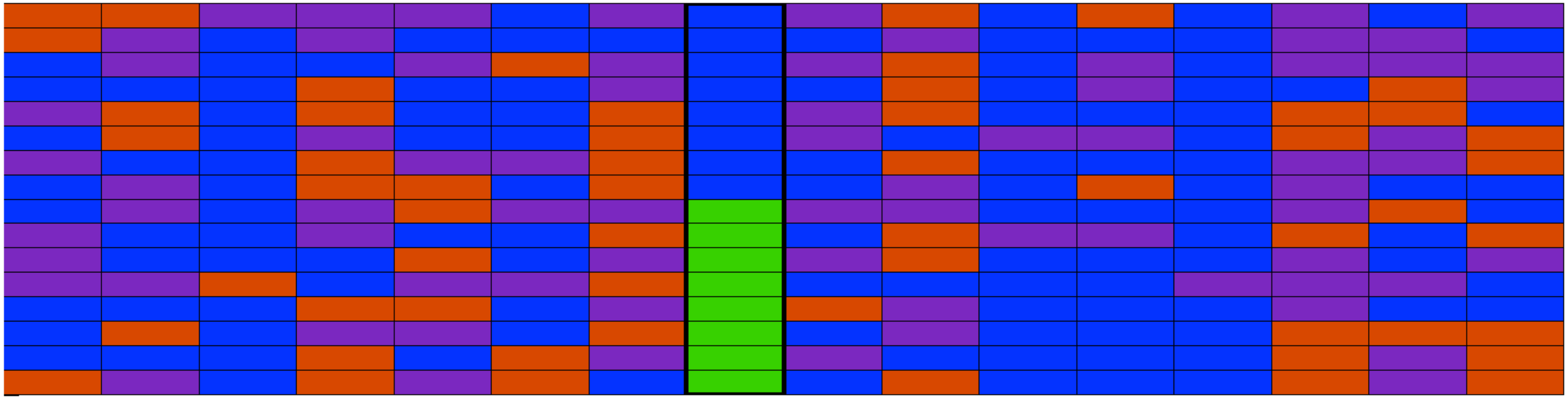
Garbage Collection Animation With WOM Codes

🟩 Free page (2 writes remain)

🟦 Valid page (1 write remains)

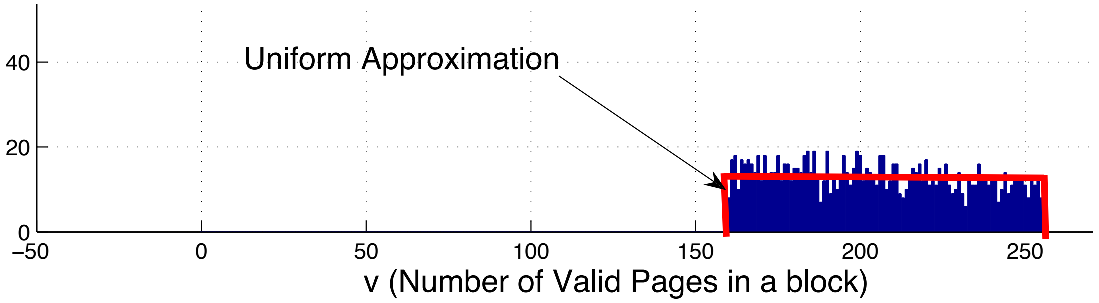
🟪 Valid page (0 writes remain)

🟥 Invalid page



Animation of garbage collection:
http://bit.ly/ZPdMn0

Analysis "Technique A" [Agarwal & Marrow]



The number of valid pages **in a block** (over all blocks)

- Assumed uniform distribution
- Easy to compute the expected value

The number of valid pages **per block** (over one block)

- random distribution of writes gives binomial distribution
- Easy to compute the expected value

Equate two ways to find the **expected number of valid blocks**

- Simple analytic expression for write amplification A:

$$A = \frac{1+\rho}{2\rho} \qquad \text{Overprovisioning factor } \rho$$

The uniform distribution assumption valid under some conditions.

Analysis: Technique B

Our Approach

Technique A works with number invalid pages over the entire memory

Technique B works with the number invalid pages in the block selected for garbages collection

- Each garage collection, x invalid blocks are freed
- # of invalid pages = # blocks per page × probability of being invalid

$$x = N \times \Big(1 - (1-p)\Big)^{Tx}$$

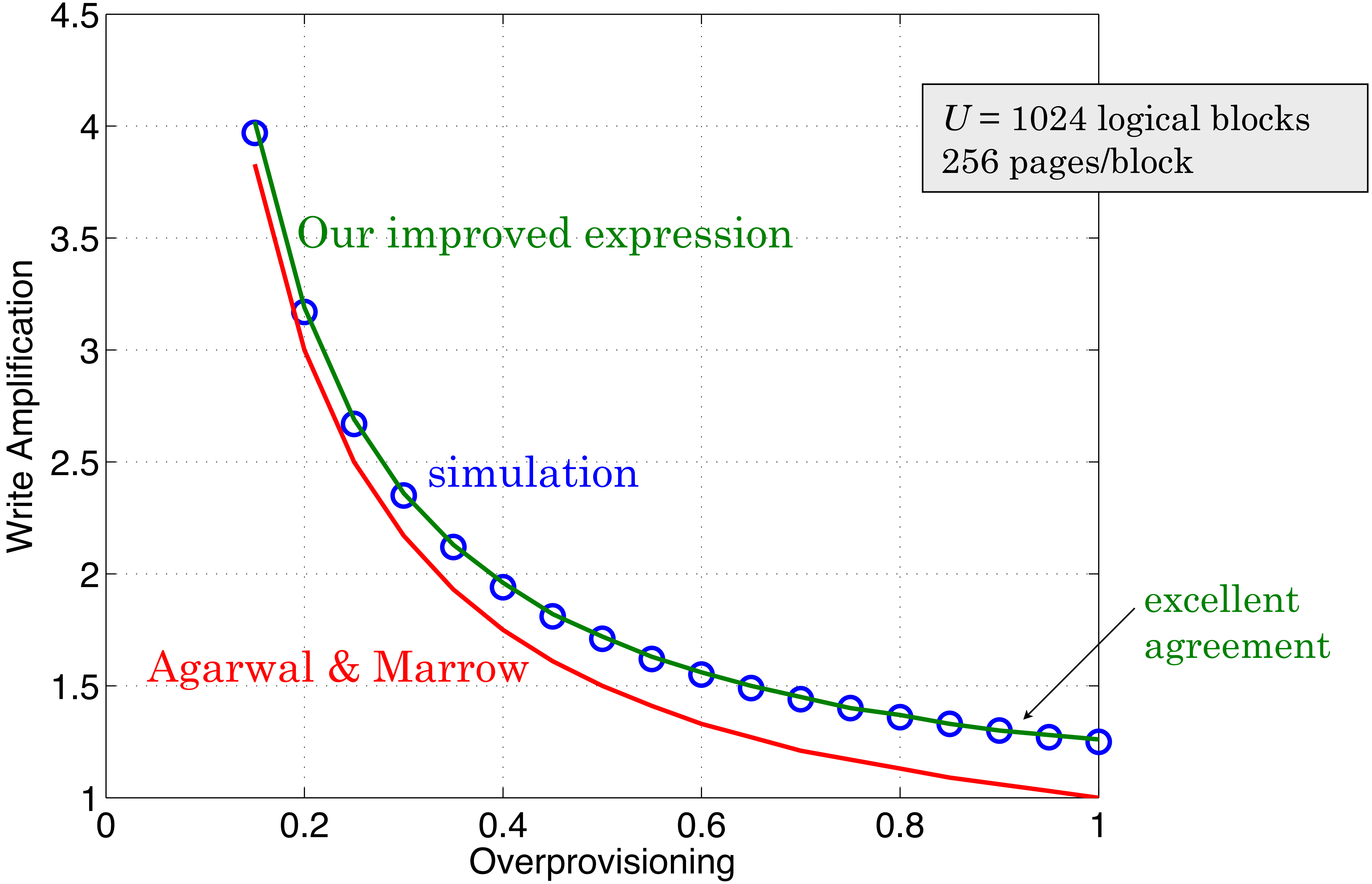
... Contribution — Obtain write amplification:

$$A = \frac{1+\rho}{1+\rho+W\Big(-(1+\rho)e^{-(1+\rho)}\Big)}$$

$W(.)$ is the Lambert W function. The solution to $c = xe^{x}$ is $W(c)$.

Let the number of pages →∞. Reasonable, since flash memories are huge.

Improved Prediction of Write Amplification



$U = 1024$ logical blocks
256 pages/block

WOM Codes: Codes for Write-Once Memories

WOM codes:

- re-write flash memory without erasing
- Write flash memory t times
- Decrease the code rate R for increasing t
- For a q-level flash [Fu and Han Vinck 1999]:

$$R_1 + R_2 + \cdots + R_t \quad \leq \quad \log_2 \binom{q+t-1}{t}$$

Flash memories have $\log_2 q$ bits/cell

- SLC (1 bit), MLC (2 bits), TLC (3 bits), QLC (4 bits)

We assume the existence of WOM codes that achieve capacity

WOM Memory Controller

Controller tracks the state of the WOM code. For each page:

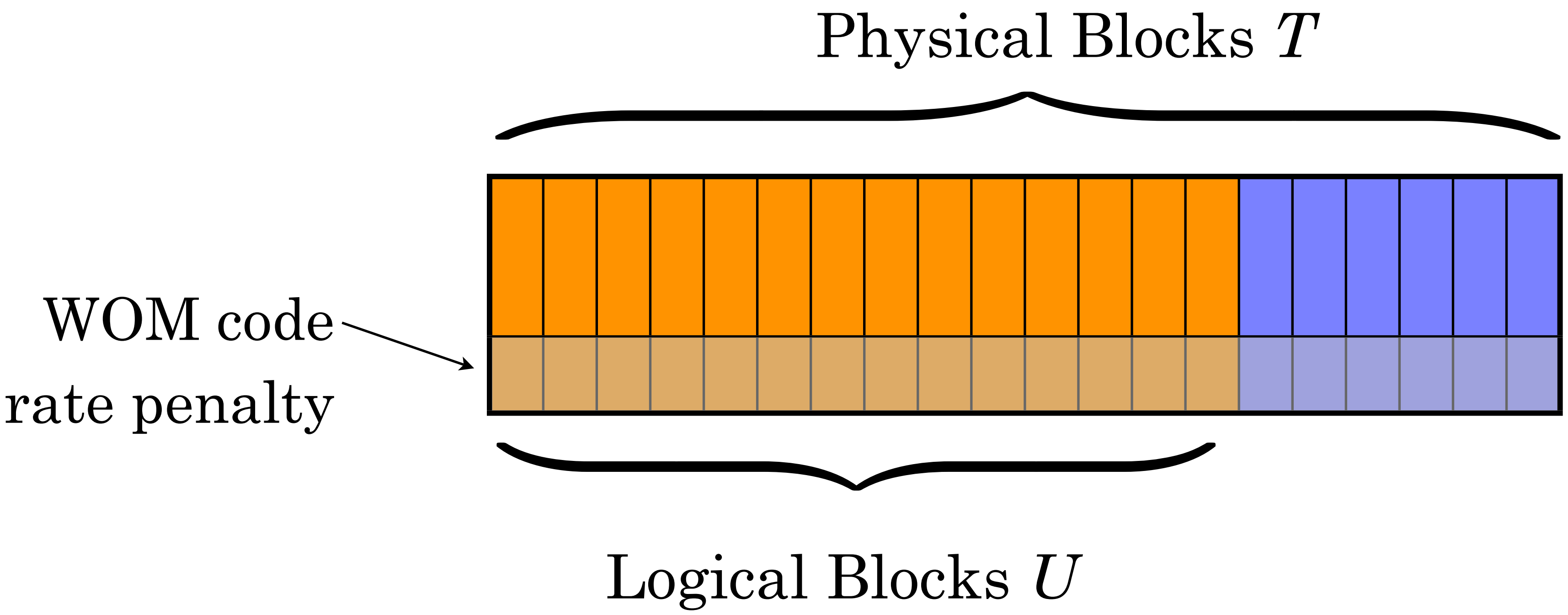
Free → t – 1 writes remain → … → 1 writes → 0 writes → Invalid

“Valid” states

Consider the “total provisioning” ρ_{total}

- Traditional overprovisioning ρ
- WOM code rate/write R, and t writes

$$\rho_{\text{total}} = (\rho + 1)\frac{\log_2 q}{tR} - 1$$



Analysis: Write Amplification with WOM Codes

Technique B does not allow a closed-form solution

Use Technique A. Obtain an expression for write amplification:

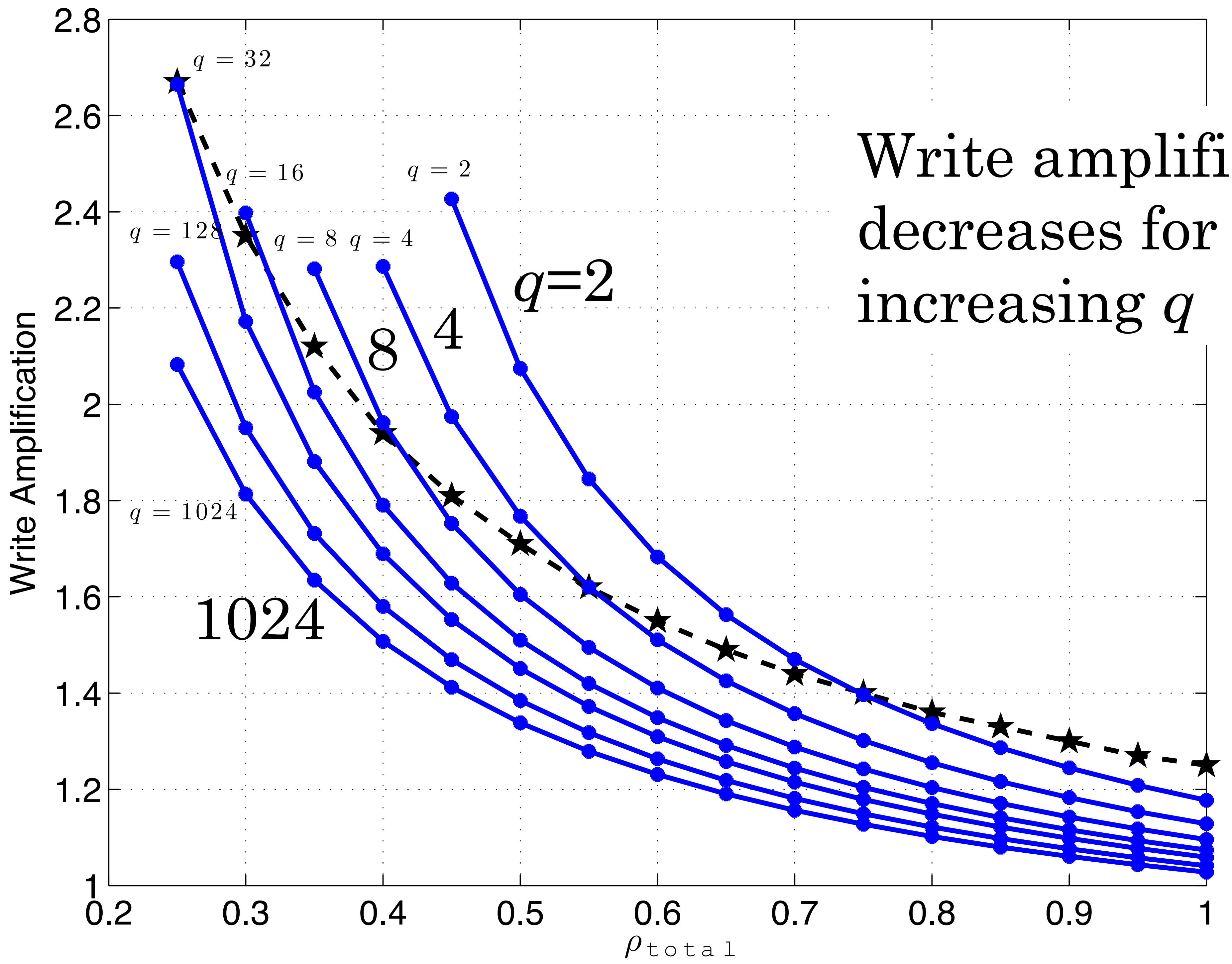
$$1 - \frac{1}{2t} + \frac{1}{2}\frac{1}{(\rho_{total} + 1)\log_q \binom{q+t-1}{t} - t}$$

WOM code shifts the distribution of valid pages in a block,
□ Uniform distribution appears accurate

Comparison of conditions used

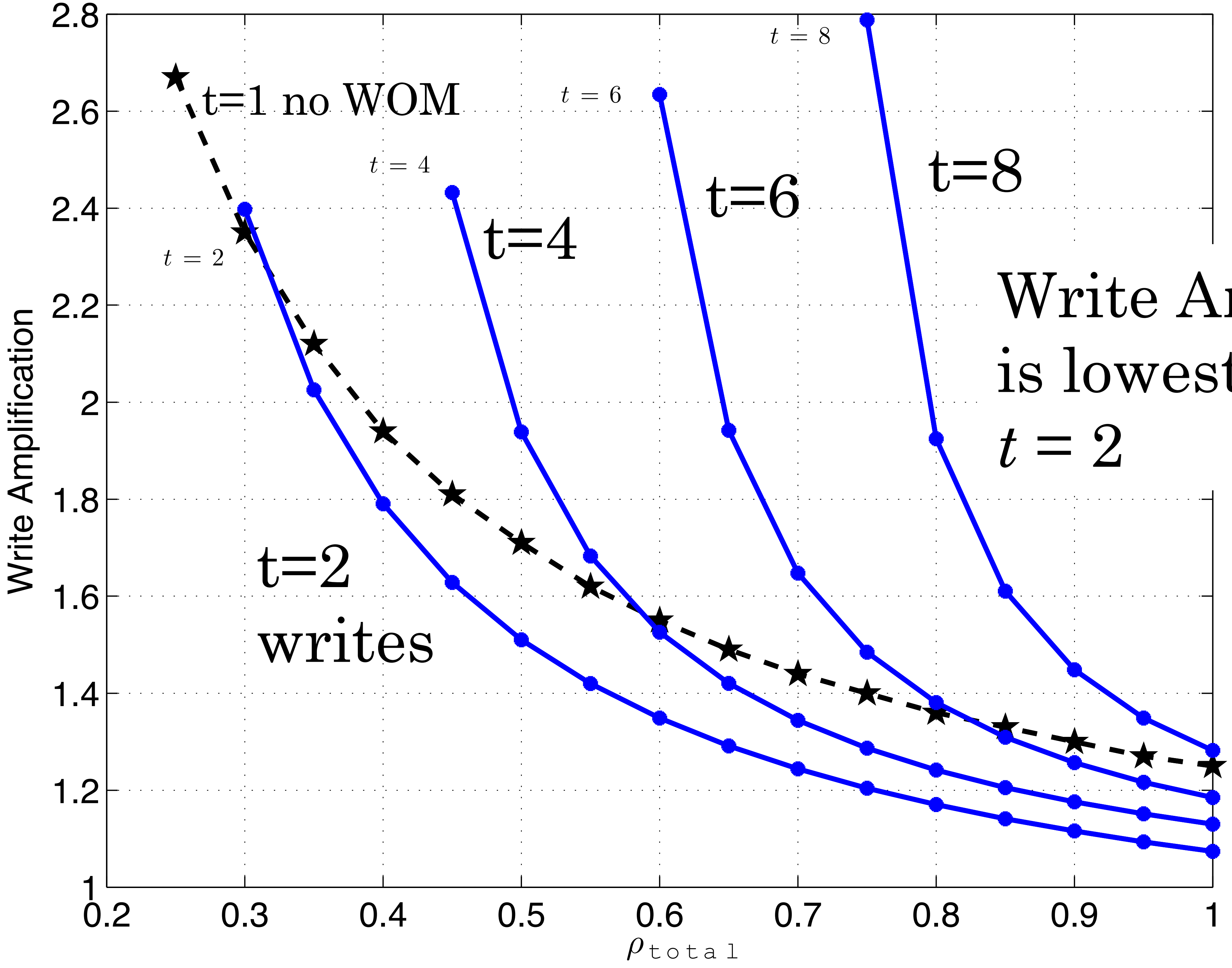
	no WOM t=1	WOM t > 1
Technique A	some agreement Agarwal & Marrow	accurate
Technique B	accurate	not possible

Write Amplification for t=2 WOM Codes



Write amplification decreases for increasing q

Write Amplification for q=16 (QLC) flash



Write Amplification is lowest for $t = 2$

Discussion & Conclusion

Write amplification is are excess writes in flash memory systems:

- conventionally mitigated by overprovisioning

WOM Codes: promise to extend the life of flash memories

- hot topic among coding theorists

Contribution: WOM Codes *can* also reduce write amplification

- as q increases, WOM codes are more effective at reducing WA
- $t = 2$ write WOM codes have lower WA than no WOM/$t \geq 3$ WOM ($q = 16$)