Relay-based Key Management to Support Secure Deletion for Resource-Constrained Flash-Memory Storage Devices

Wei-Lin Wang (Institute of Information Science, Academia Sinica),
Yuan-Hao Chang, Po-Chun Huang, Chia-Heng Tu,
Hsin-Wen Wei, and Wei-Kuan Shih

Outline

- Introduction
- Background
- An Efficient Secure Deletion Scheme
- Performance Evaluation
- Conclusion

Outline

- Introduction
- Background
- An Efficient Secure Deletion Scheme
- Performance Evaluation
- Conclusion

Intro – Flash Memory

- Nice features:
  - Shock-resistance
  - Non-volatility
  - Low energy consumption (Compared to hard drives)
  - Small size
  - Low cost
- Diversified application domains
  - Portable storage devices
  - Consumer electronics
  - Industrial applications

Intro - Observation

- Trends in NAND flash memory
  - About 40% annual price reductions in last few years
  - Storage capacity has grown 100 times in the last ten years
- Formatting of file systems
  - Quick formatting
    - Only virtually delete by resetting the file system's metadata information
    - Malicious users still can retrieve the data by accessing memory address
  - Full formatting
    - Support secure deletion by physically rewriting file contents or storage space to ensure there is no way to get any file content back again

Intro - Motivation

- Secure deletion is an importable issue for portable devices

- Original secure formatting is not suitable any more
  - Because the time to reset all the data will not be acceptable due to the fast-growing storage capacity

- We want to provide an efficient scheme to support secure deletion for resource-constrained huge-capacity flash storage devices

Outline

- Introduction
- Background
  - Flash memory
  - System architecture
  - Flash Translation Layer - DFTL
- An Efficient Secure Deletion Scheme
- Performance Evaluation
- Conclusion

Background – flash memory

Block:

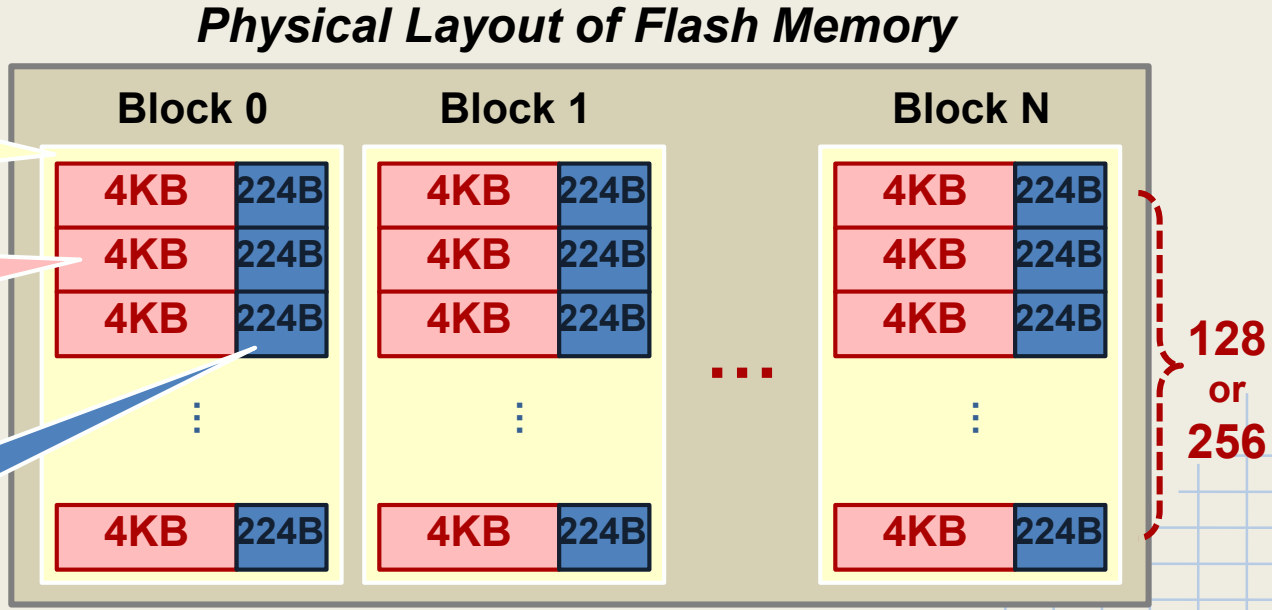
Basic erase unit

Page (Data Area):

Basic W/R unit

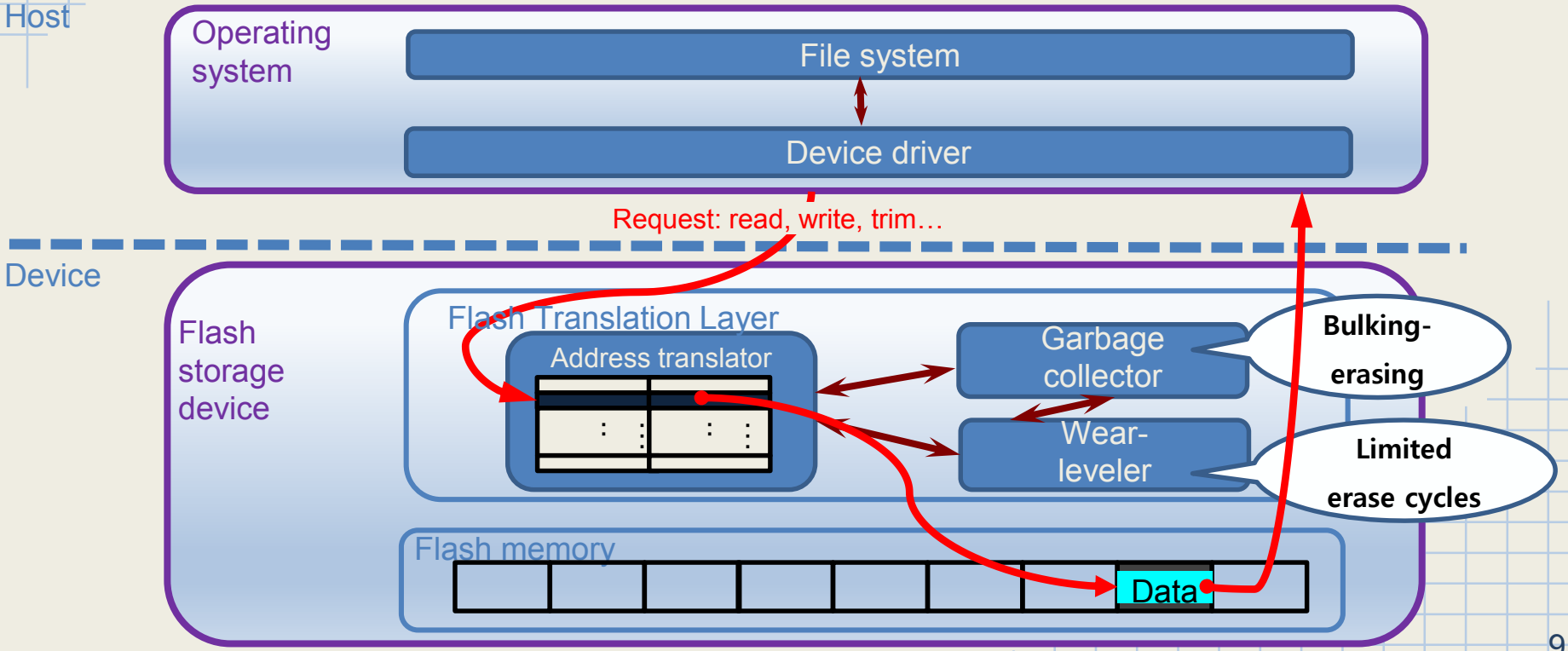
Page (Spare Area):

metadata

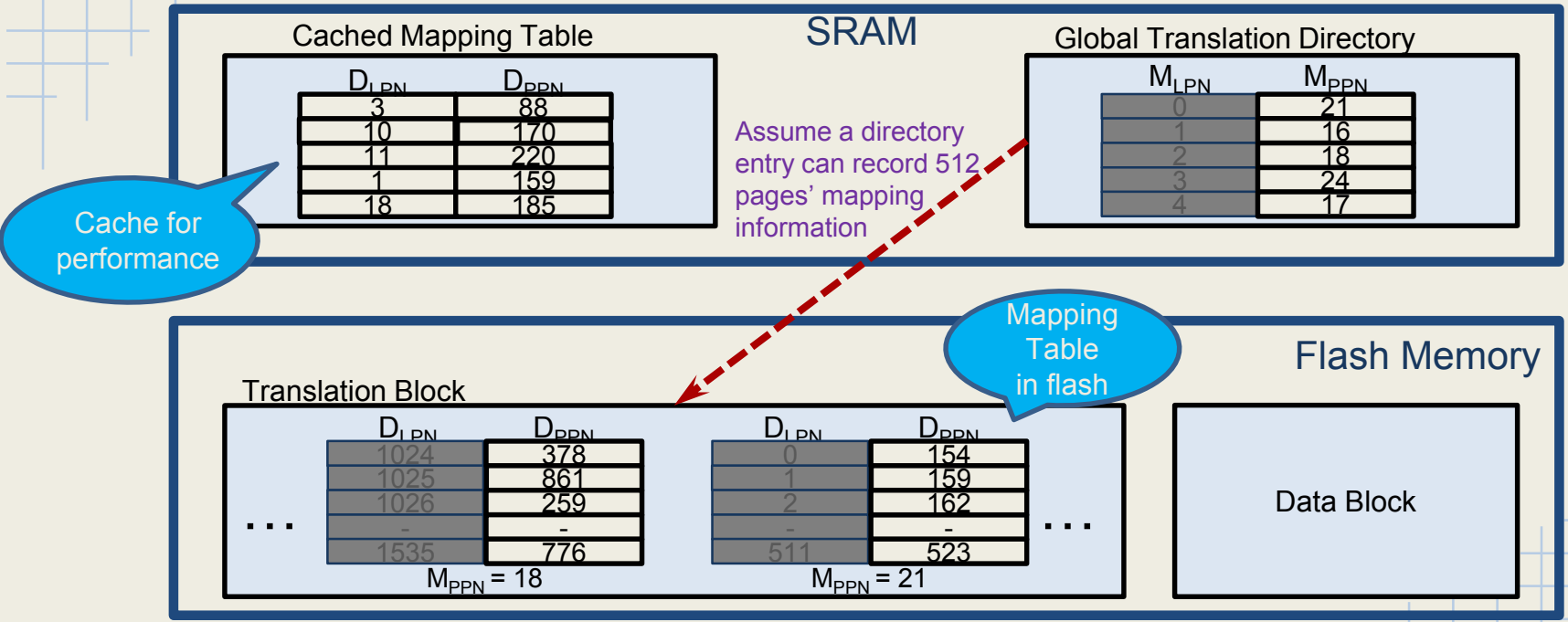


- Bulk-erasing
  - Erase unit (block) is larger than r/w unit(page)
- Write-once property
  - Data can not be overwrite until it is erased
  - Out-place updates

Background - System Architecture



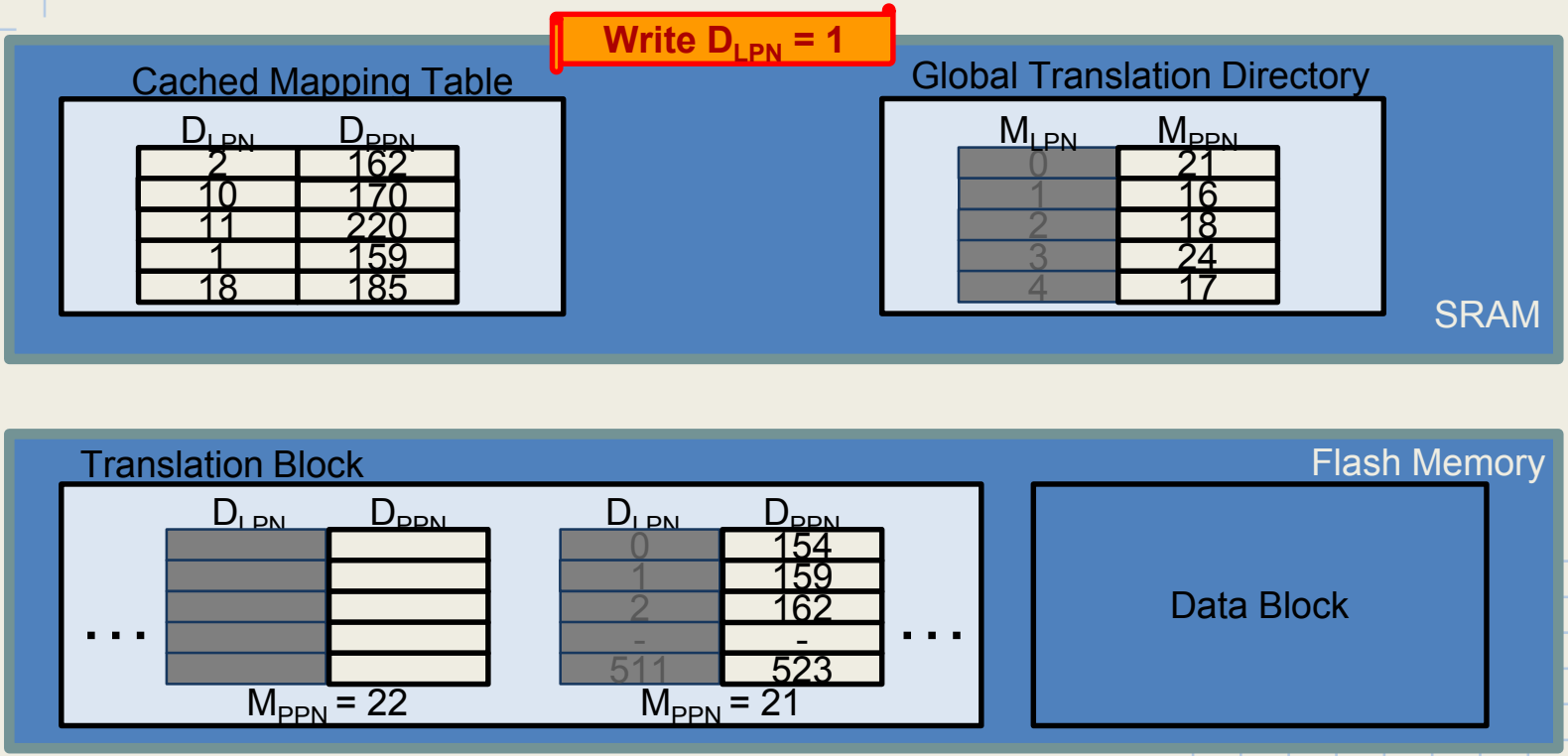
Background-DFTL



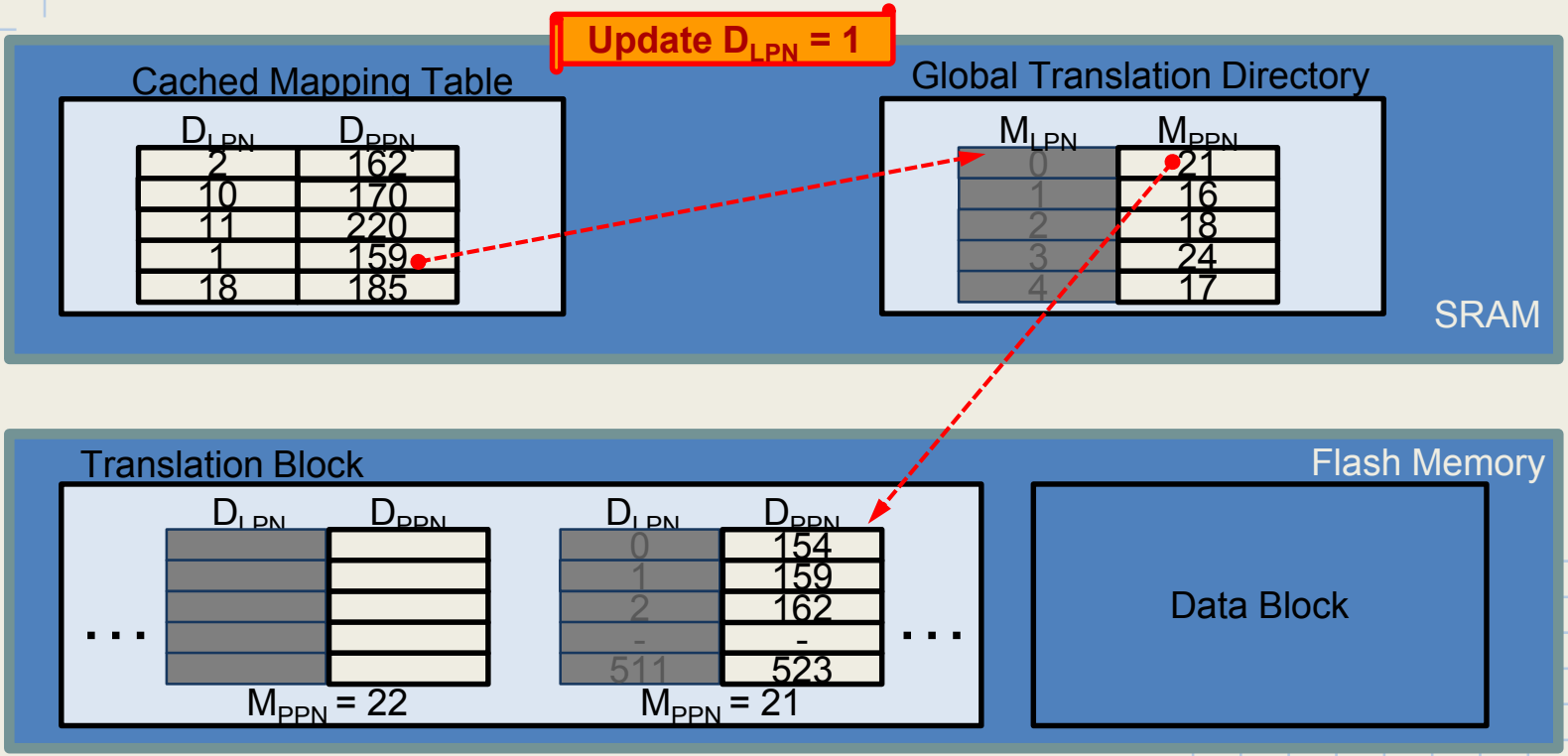
D_{LPN} : Data logical page number
D_{PPN} : Data physical page number

M_{LPN} : Mapping table logical page number
M_{PPN} : Mapping table physical page number

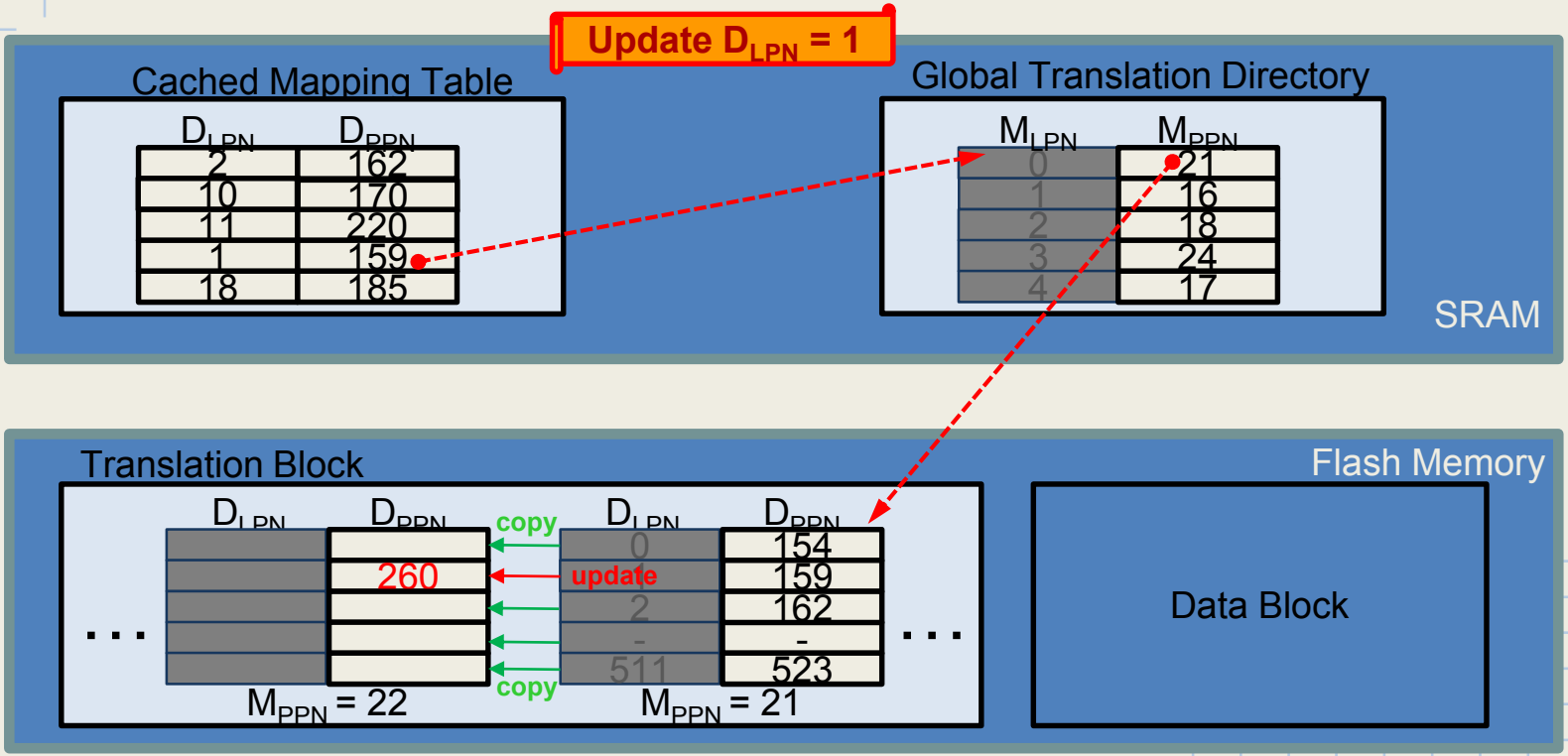
Background-DFTL



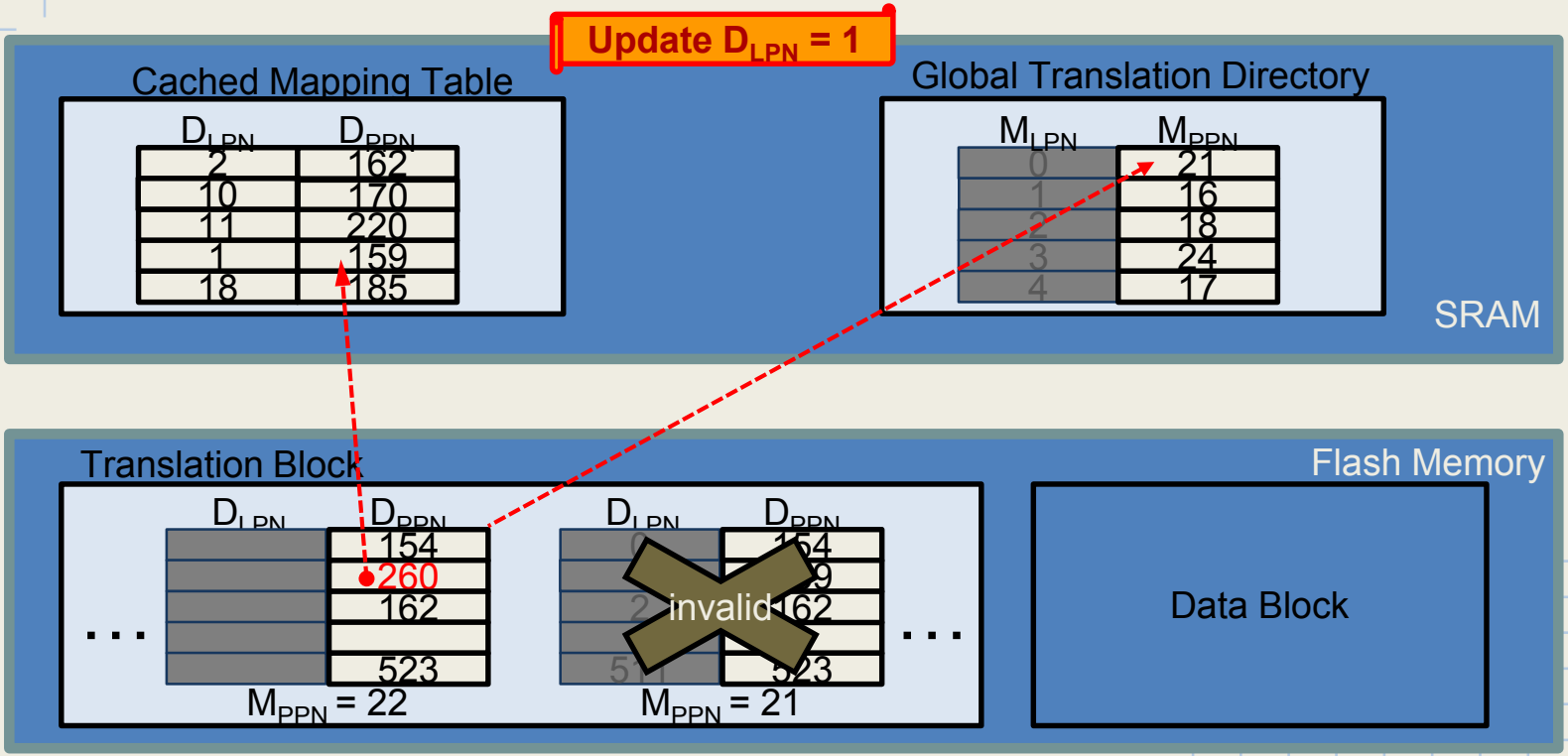
Background-DFTL



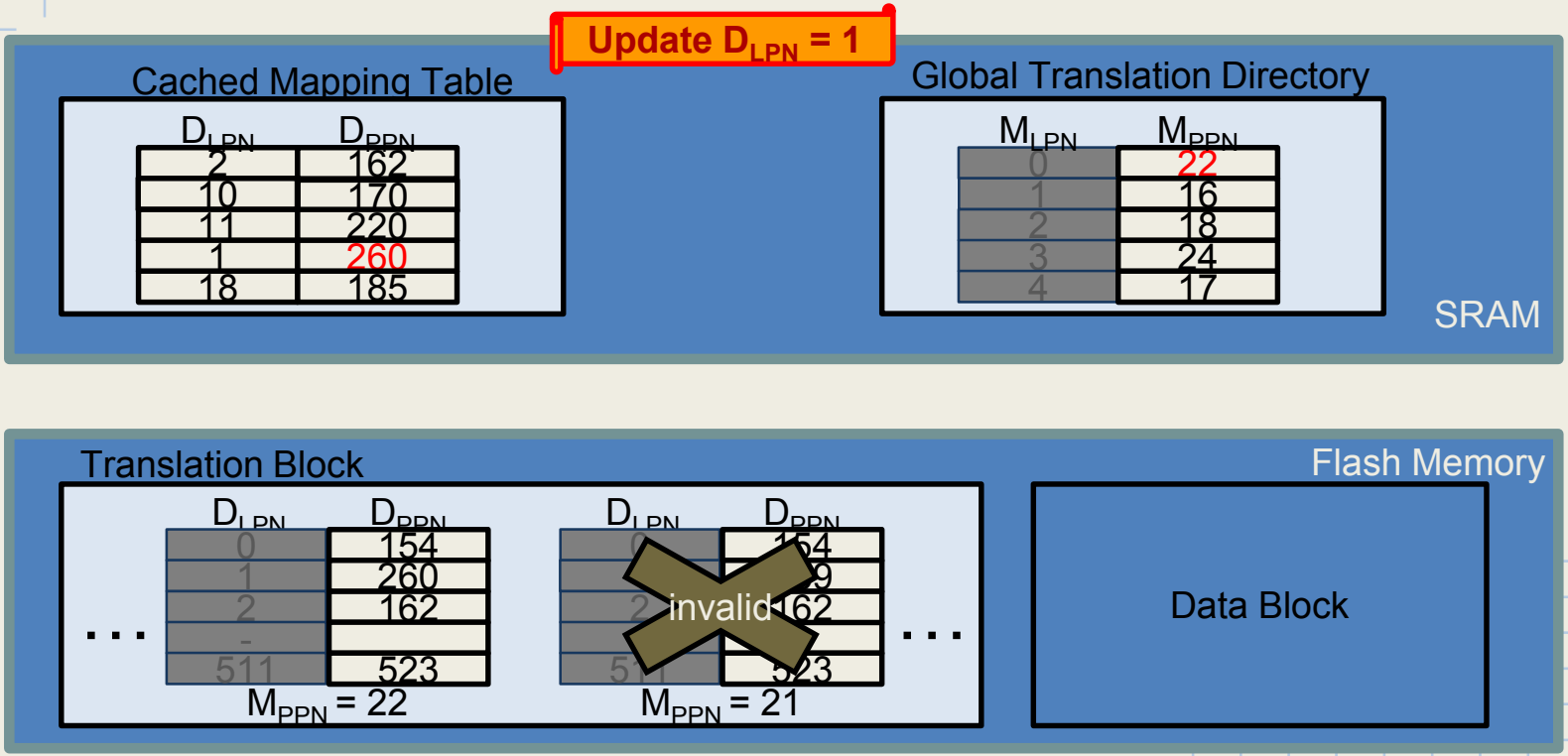
Background-DFTL



Background-DFTL



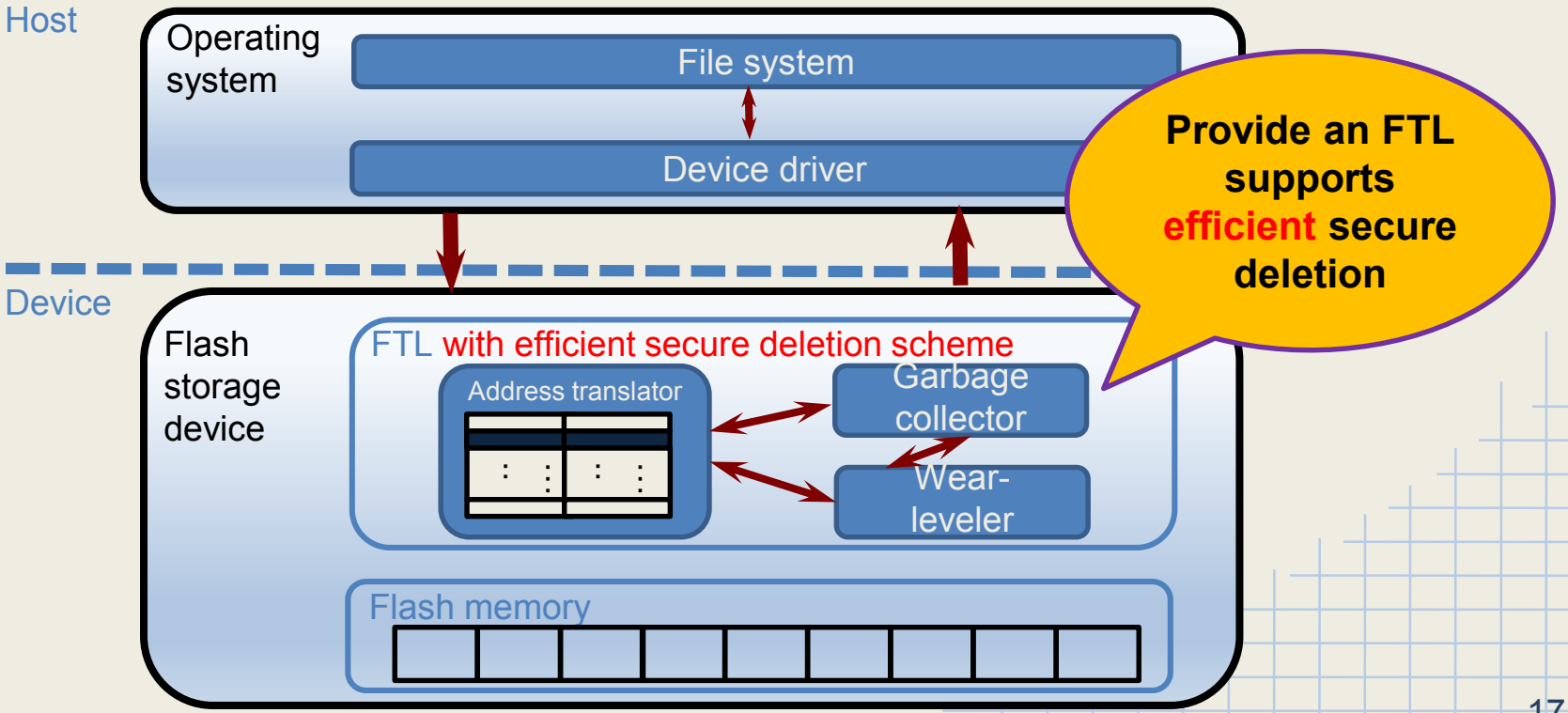
Background-DFTL



Outline

- Introduction
- Background
- An Efficient Secure Deletion Scheme
  - Overview
  - Secure Deletion Scheme (SDS)
  - Advanced Secure Deletion Scheme (SDS^+)
- Performance Evaluation
- Conclusion

Overview

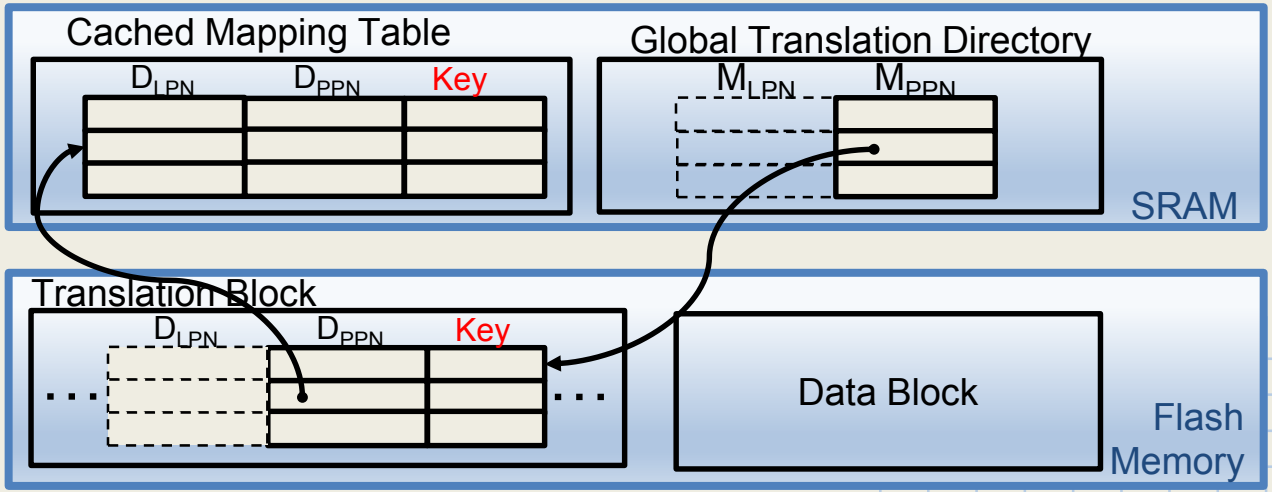

Host
Operating system
File system
Device driver
Device
Flash storage device
FTL with efficient secure deletion scheme
Address translator
Garbage collector
Wear-leveler
Flash memory
Provide an FTL supports efficient secure deletion

Overview - Design Issues

- Implement on flash translation layer
  - Performance of secure deletion is independent of the device capacity and file systems
- Encrypt the data when writing and only need to destroy the key when formatting.
  - High performance
  - Support secure deletion

Secure Deletion Scheme (SDS)

- Based on DFTL
- Each logical page has an individual key
  - Each entry in either cache or flash needs an extra field for its key.

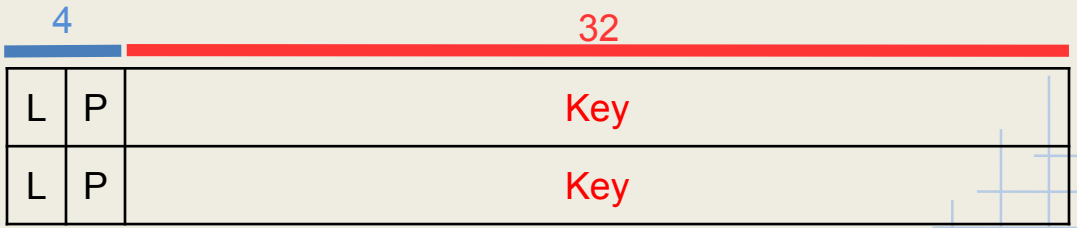


SDS - Weak Point

- The number of entries will decrease dramatically due to the key field

D_{LPN}	D_{PPN}
-	-
-	-
-	-
-	-
-	-
-	-
-	-
-	-

D_{LPN}	D_{PPN}
-	-
-	-
-	-
-	-
-	-
-	-
-	-
-	-

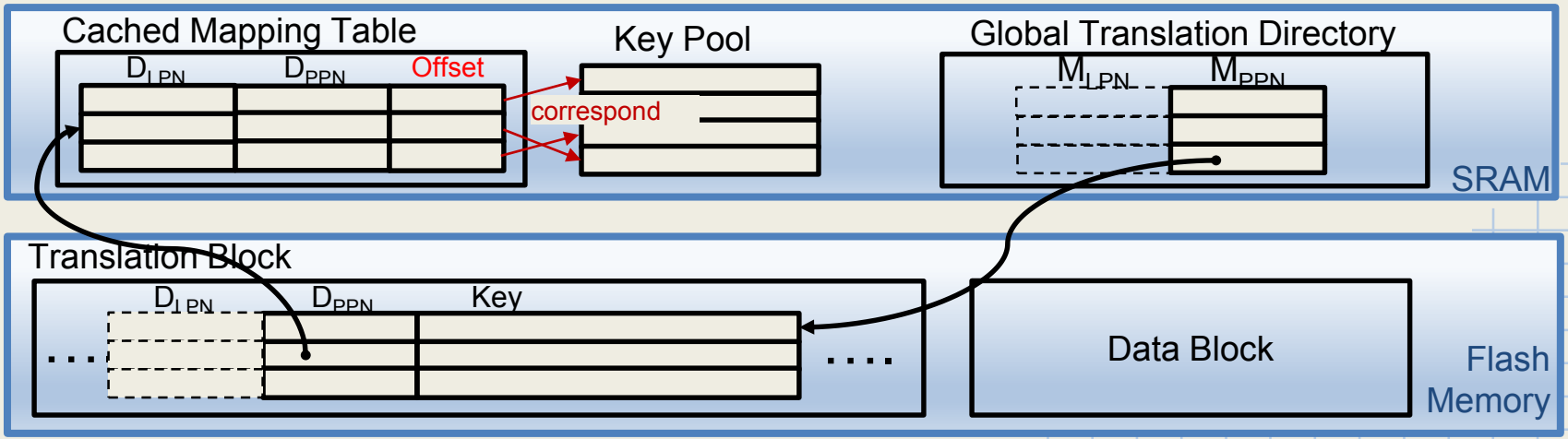


Suppose the size of page information is 4Bytes

and the key length is 32 Bytes

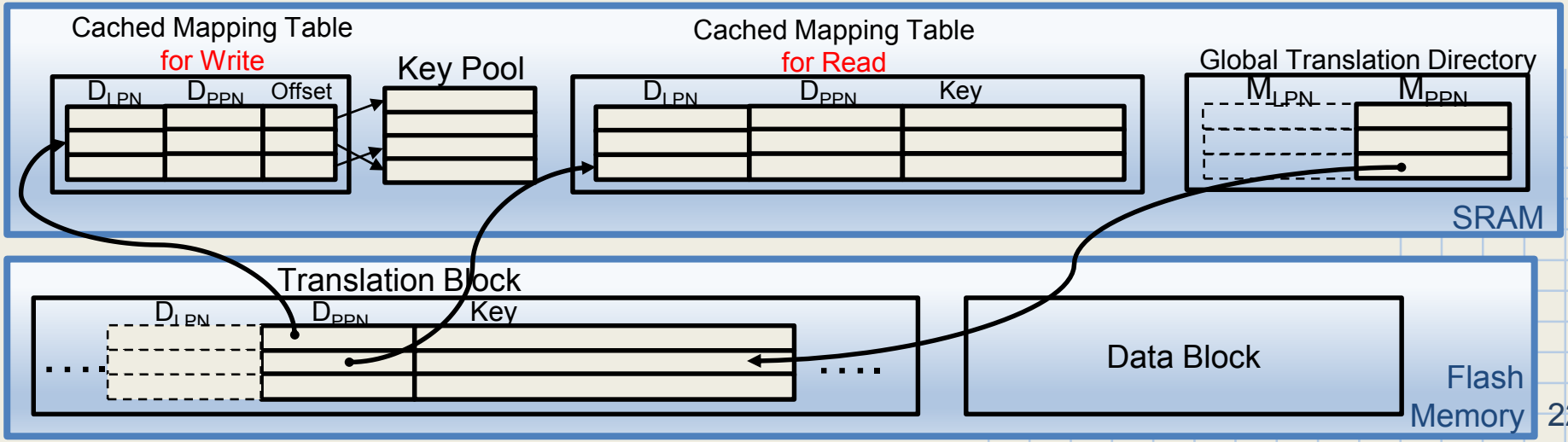
Advanced Secure Deletion Scheme (SDS+)

- Fixed-sized key pool
  - When a page is written, use the key in key pool to encrypt
  - Only need to record key pool's offset instead of the key
    - If there are 256 keys in the key pool, key field size is 8bits instead of 256 bits



Advanced Secure Deletion Scheme (SDS+) (Cont.)

- For security, the times of a key used to encrypt need to be controlled
  - Need to update the key in key pool
- When a page is read in cache, its key might be already evicted
  - Our method: Separate read / write cache, only write cache can use offset.



SDS+ - Key Management

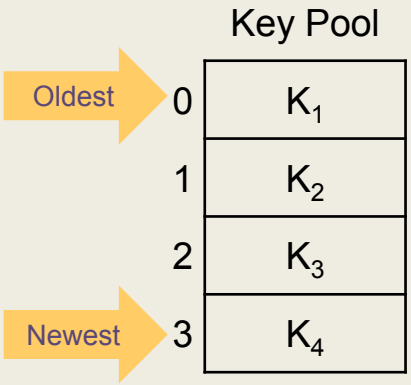
- Ideal goal
  - Each entry in the cache uses a key once, so that each key will not be used anymore.

- Design feature
  - Each entry gradually uses the key in key pool
    - When a cache write to an entry occurs, use the next key of the key that is currently used by the cache entry (Relay-based!!!)
  - Key pool is a circular buffer
    - When a new key is generated, it replaces the oldest key and all entries using the oldest key need to be saved back to flash and be invalidated from cache

SDS+ - Example of Key Usage

Request

Write (4,140) (D_{LPN}, D_{PPN})



D_{LPN}	D_{PPN}	Key
3	150	1
10	170	0
11	220	3
1	260	0
18	185	1
26	56	0
7	178	0

Cached Mapping Table

SDS+ - Example of Key Usage

Request

Write (4,140)

Key Pool

Oldest	0	K_1
	1	K_2
	2	K_3
Newest	3	K_4

D_{LPN}	D_{PPN}	Key
3	15	
[illegible]	evict	0
11	220	3
1	260	0
18	185	1
26	56	0
7	178	0

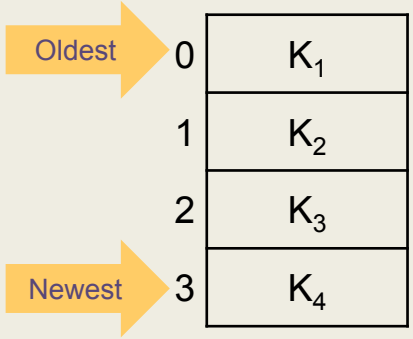
Choose by LRU

Cached Mapping Table

SDS+ - Example of Key Usage

REQUEST

Write (4,140)



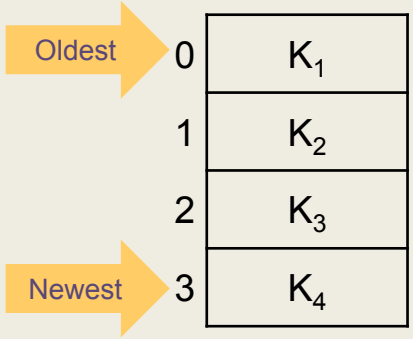
D_{LPN}	D_{PPN}	Key
3	150	1
4	140	0
11	220	3
1	260	0
18	185	1
26	56	0
7	178	0

Cached Mapping Table

SDS+ - Example of Key Usage

REQUEST

Write (4,140)



D_{LPN}	D_{PPN}	Key
3	150	1
4	140	1
11	220	3
1	260	0
18	185	1
26	56	0
7	178	0

Cached Mapping Table

SDS+ - Example of Key Usage

Request

Write (3,171)

Key Pool

Oldest	0	K_1
	1	K_2
	2	K_3
Newest	3	K_4

D_{LPN}	D_{PPN}	Key
3	150	1
4	140	1
11	220	3
1	260	0
18	185	1
26	56	0
7	178	0

Cached Mapping Table

SDS+ - Example of Key Usage

REQUEST

Write (3,171)

D_{LPN}	D_{PPN}	Key
3	171	1
4	140	1
11	220	3
1	260	0
18	185	1
26	56	0
7	178	0

Cached Mapping Table

Key Pool

Oldest	0	K_1
	1	K_2
	2	K_3
Newest	3	K_4

SDS+ - Example of Key Usage

REQUEST

Write (3,171)

D_{LPN}	D_{PPN}	Key
3	171	2
4	140	1
11	220	3
1	260	0
18	185	1
26	56	0
7	178	0

Cached Mapping Table

Key Pool

Oldest	0	K_1
	1	K_2
	2	K_3
Newest	3	K_4

SDS+ - New Key Generation

- If a cache write occurs and the written entry is using the newest key, generate a new key to replace the oldest key.
  - If a hot page is always hit, it will generate many keys and flush other pages back to flash memory
  - We have an inverse mode for this situation
  - When a cache miss occurs and the corresponding entry is using the newest key, generate a new key

SDS+ - Inverse Mode

- If cache hit occurs and the corresponding entry is using the newest key, this enters the inverse mode
  - Inverse mode is to use the key in an inverse direction from newest to oldest
- A hot page will use keys from the oldest to the newest, and then use back to the oldest one, followed by using back to the newest again
- To guarantee the security: Limit the number of times to use a key
  - When the hot page refers to the newest key for the i_{th} time (i >1), generate a new key

SDS+ - Example of *Inverse*

REQUEST

Write (18,186)

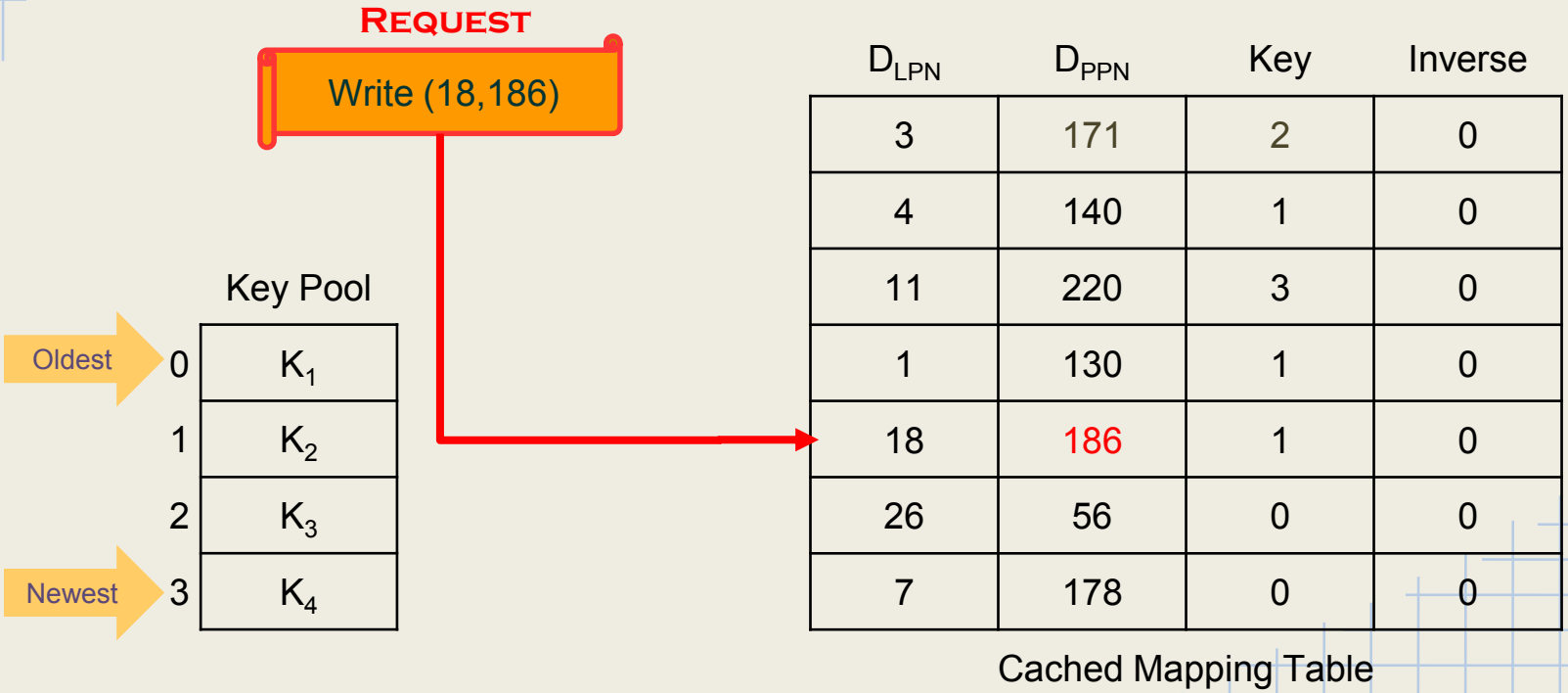
Key Pool

Oldest	0	K_1
	1	K_2
	2	K_3
Newest	3	K_4

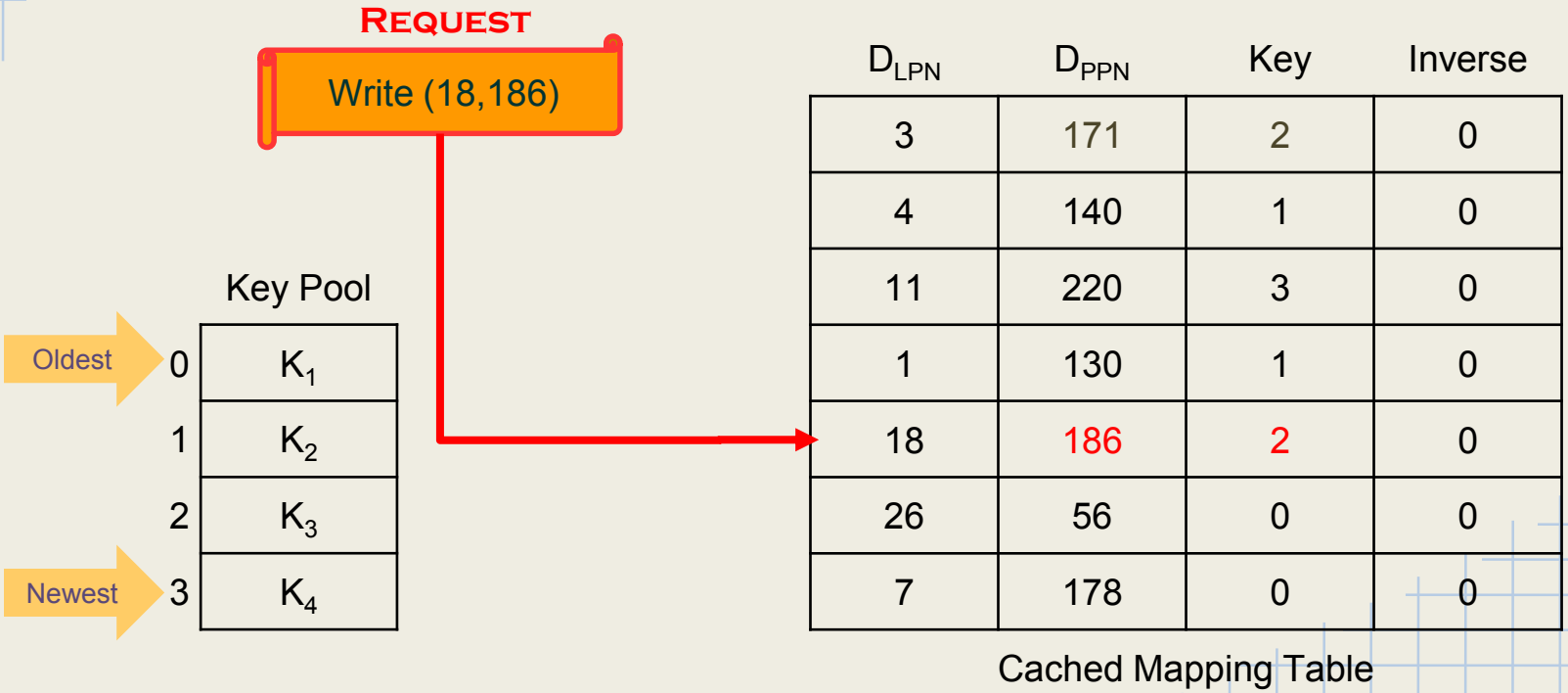
D_{LPN}	D_{PPN}	Key	Inverse
3	171	2	0
4	140	1	0
11	220	3	0
1	130	1	0
18	185	1	0
26	56	0	0
7	178	0	0

Cached Mapping Table

SDS+ - Example of *Inverse*



SDS+ - Example of *Inverse*



SDS+ - Example of *Inverse*

REQUEST

Write (18,187)

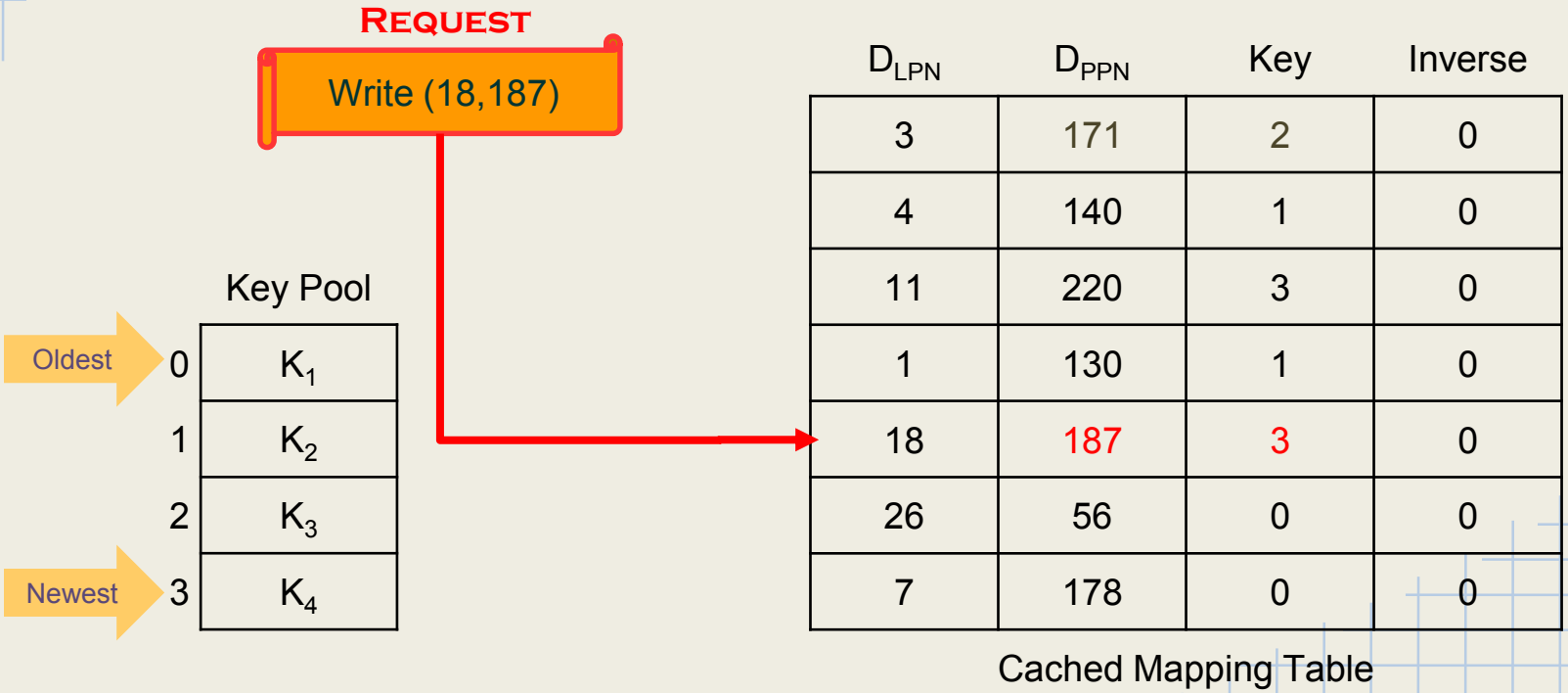
Key Pool

Oldest	0	K_1
	1	K_2
	2	K_3
Newest	3	K_4

D_{LPN}	D_{PPN}	Key	Inverse
3	171	2	0
4	140	1	0
11	220	3	0
1	130	1	0
18	186	2	0
26	56	0	0
7	178	0	0

Cached Mapping Table

SDS+ - Example of *Inverse*



SDS+ - Example of *Inverse*

REQUEST

Write (18,188)

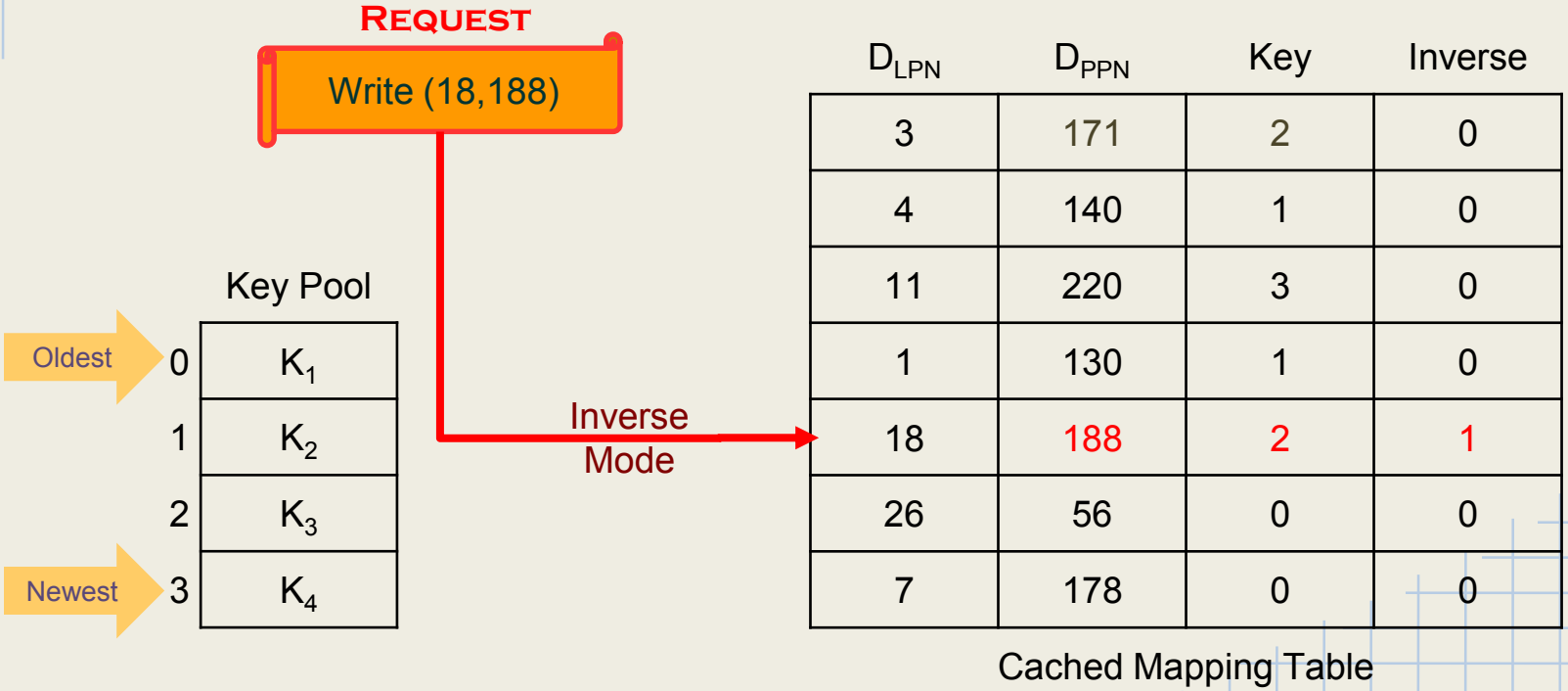
Key Pool

Oldest	0	K_1
	1	K_2
	2	K_3
Newest	3	K_4

D_{LPN}	D_{PPN}	Key	Inverse
3	171	2	0
4	140	1	0
11	220	3	0
1	130	1	0
18	187	3	0
26	56	0	0
7	178	0	0

Cached Mapping Table

SDS+ - Example of *Inverse*



SDS+ - Example of *Inverse*

REQUEST

Write (18,189)

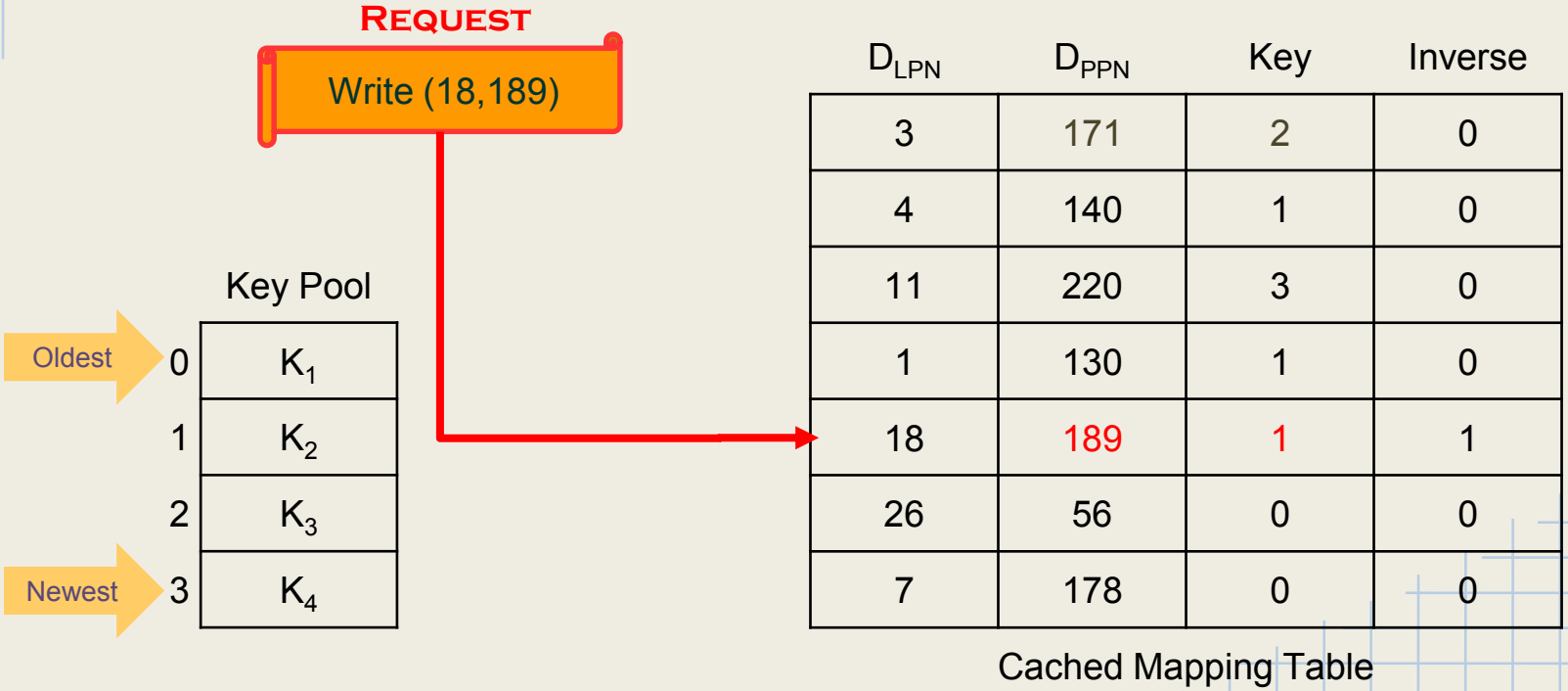
Key Pool

Oldest	0	K_1
	1	K_2
	2	K_3
Newest	3	K_4

D_{LPN}	D_{PPN}	Key	Inverse
3	171	2	0
4	140	1	0
11	220	3	0
1	130	1	0
18	188	2	1
26	56	0	0
7	178	0	0

Cached Mapping Table

SDS+ - Example of *Inverse*



SDS+ - Example of *Inverse*

REQUEST

Write (18,190)

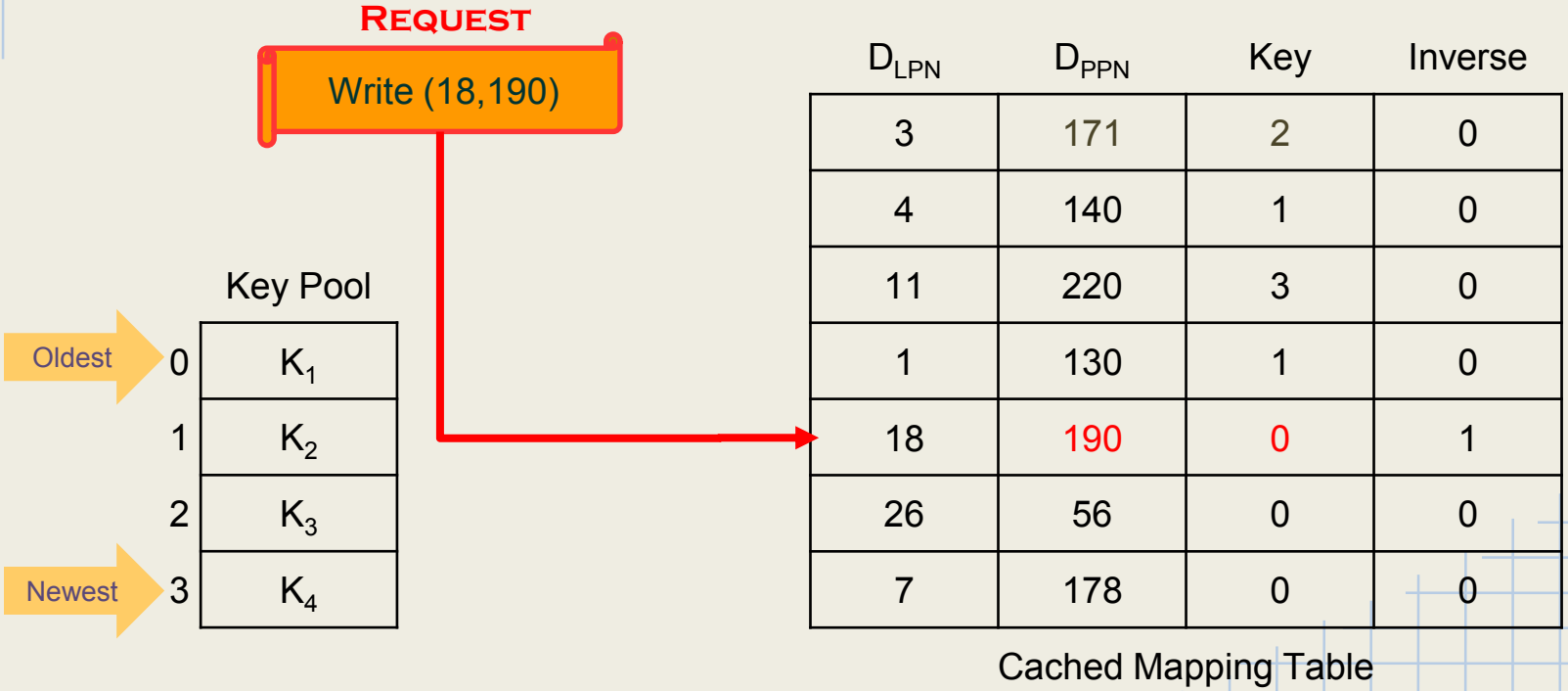
Key Pool

Oldest	0	K_1
	1	K_2
	2	K_3
Newest	3	K_4

D_{LPN}	D_{PPN}	Key	Inverse
3	171	2	0
4	140	1	0
11	220	3	0
1	130	1	0
18	189	1	1
26	56	0	0
7	178	0	0

Cached Mapping Table

SDS+ - Example of *Inverse*



SDS+ - Example of *Inverse*

REQUEST

Write (18,191)

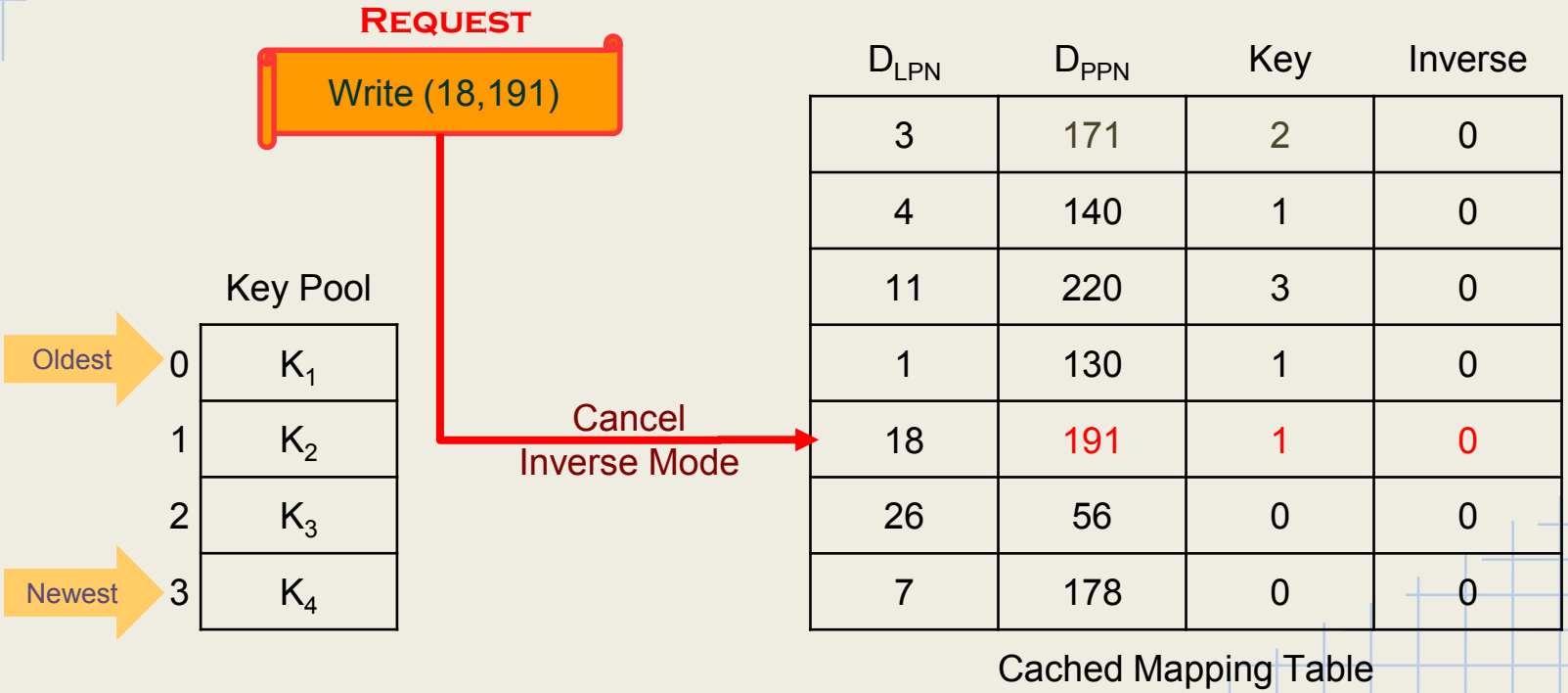
Key Pool

Oldest	0	K_1
	1	K_2
	2	K_3
Newest	3	K_4

D_{LPN}	D_{PPN}	Key	Inverse
3	171	2	0
4	140	1	0
11	220	3	0
1	130	1	0
18	190	0	1
26	56	0	0
7	178	0	0

Cached Mapping Table

SDS+ - Example of *Inverse*



SDS+ - Example of *Inverse*

REQUEST

Write (18,192)

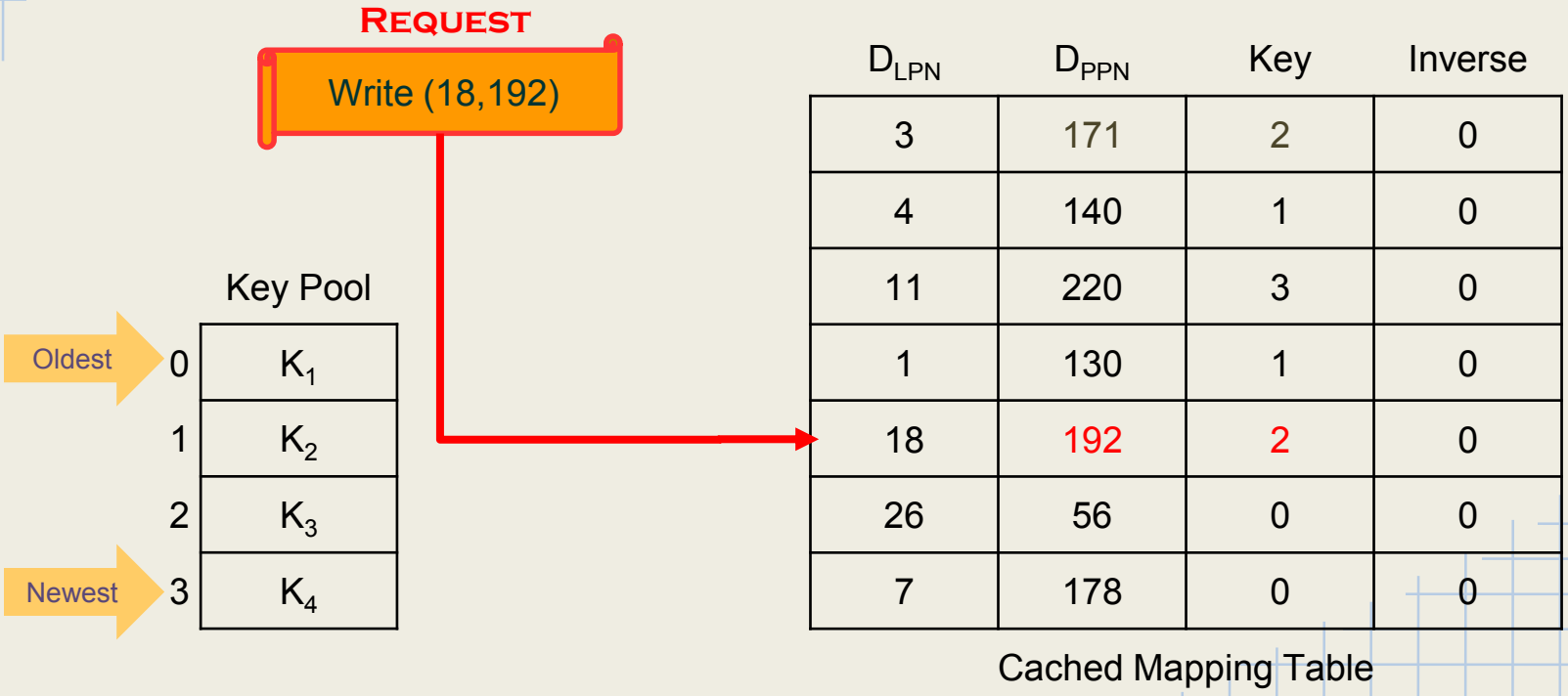
Key Pool

Oldest	0	K_1
	1	K_2
	2	K_3
Newest	3	K_4

D_{LPN}	D_{PPN}	Key	Inverse
3	171	2	0
4	140	1	0
11	220	3	0
1	130	1	0
18	191	1	0
26	56	0	0
7	178	0	0

Cached Mapping Table

SDS+ - Example of *Inverse*



SDS+ - Example of *Inverse*

REQUEST

Write (18,193)

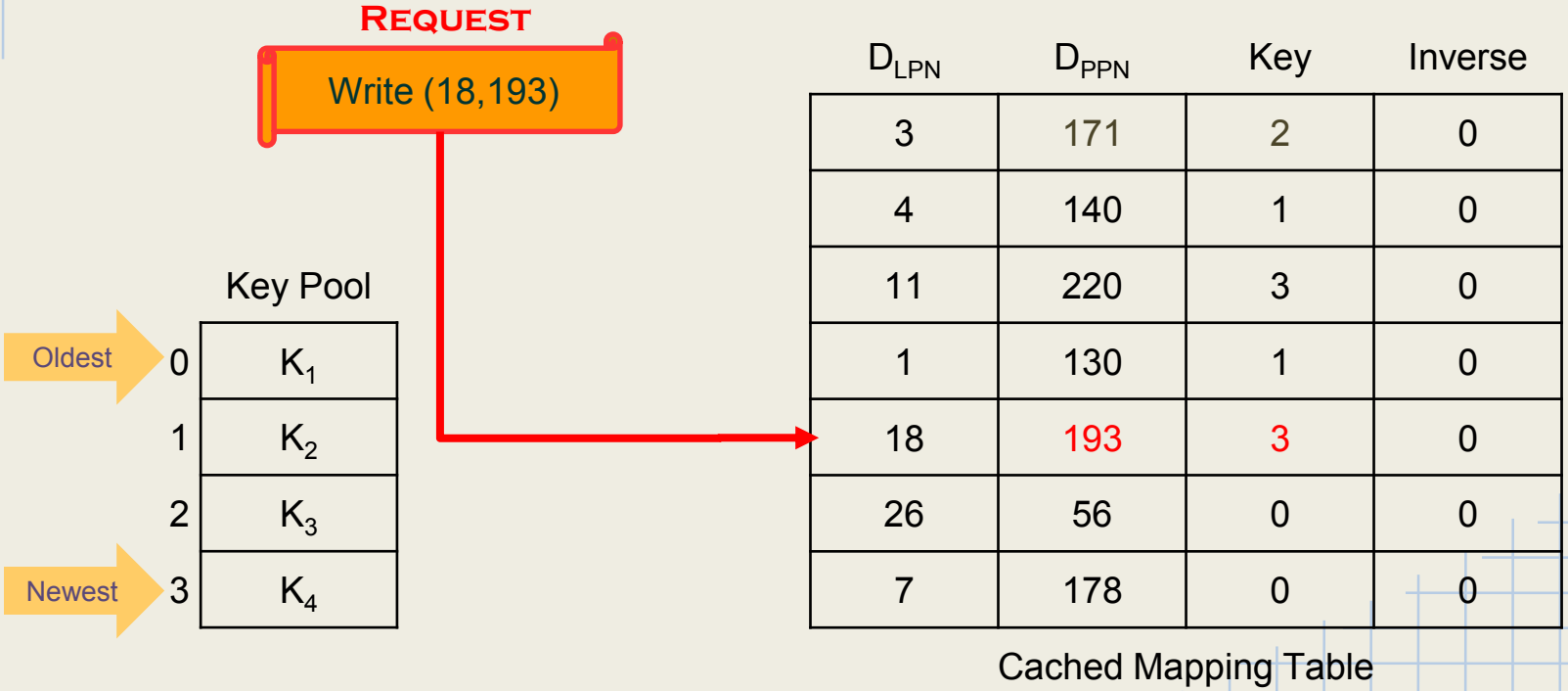
Key Pool

Oldest	0	K_1
	1	K_2
	2	K_3
Newest	3	K_4

D_{LPN}	D_{PPN}	Key	Inverse
3	171	2	0
4	140	1	0
11	220	3	0
1	130	1	0
18	192	2	0
26	56	0	0
7	178	0	0

Cached Mapping Table

SDS+ - Example of *Inverse*



SDS+ - Example of *Inverse*

REQUEST

Write (18,194)

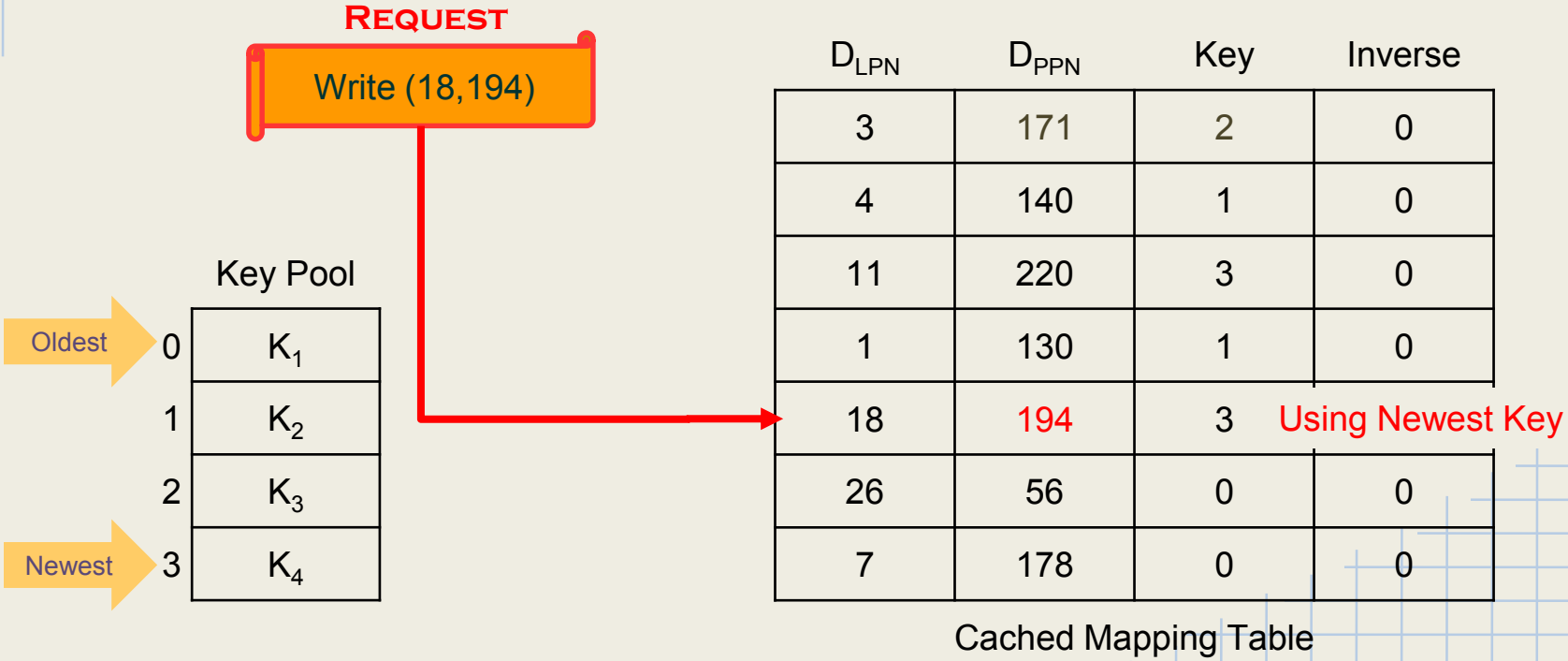
Key Pool

Oldest	0	K_1
	1	K_2
	2	K_3
Newest	3	K_4

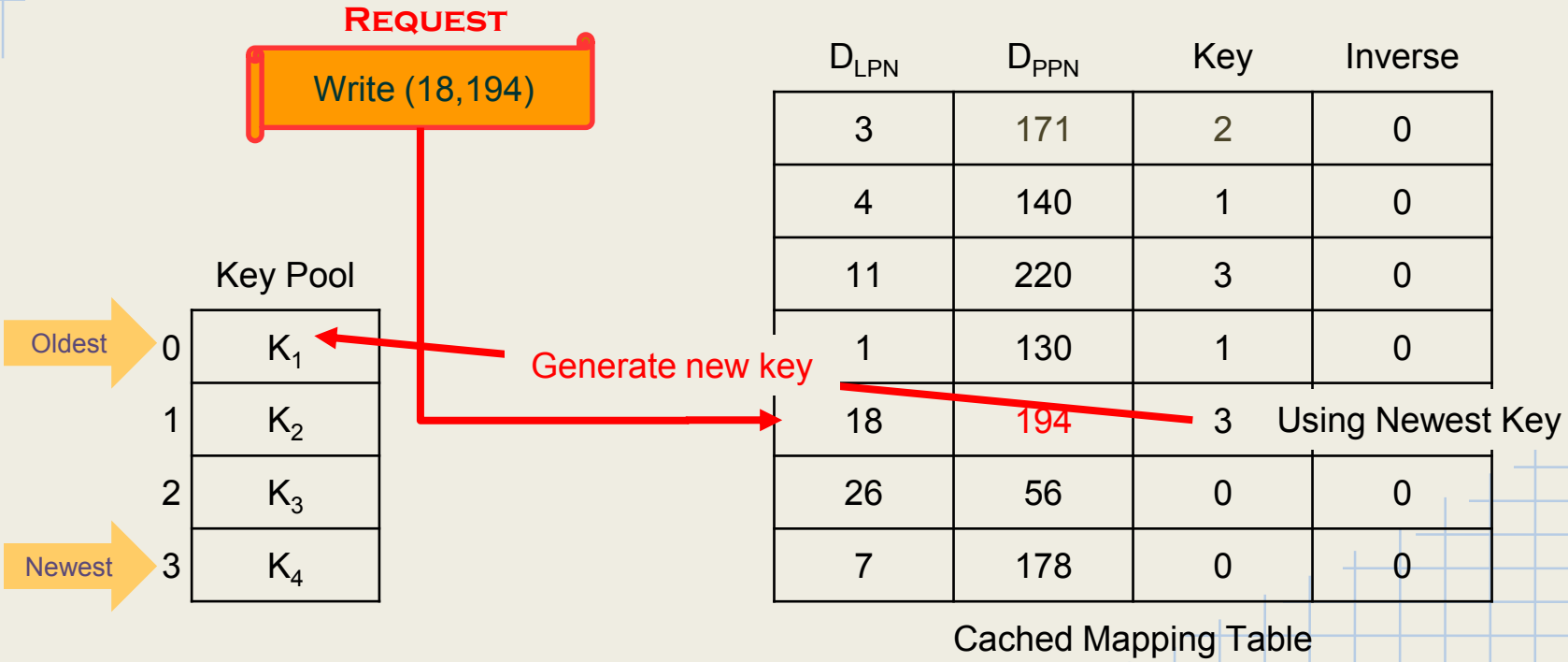
D_{LPN}	D_{PPN}	Key	Inverse
3	171	2	0
4	140	1	0
11	220	3	0
1	130	1	0
18	193	3	0
26	56	0	0
7	178	0	0

Cached Mapping Table

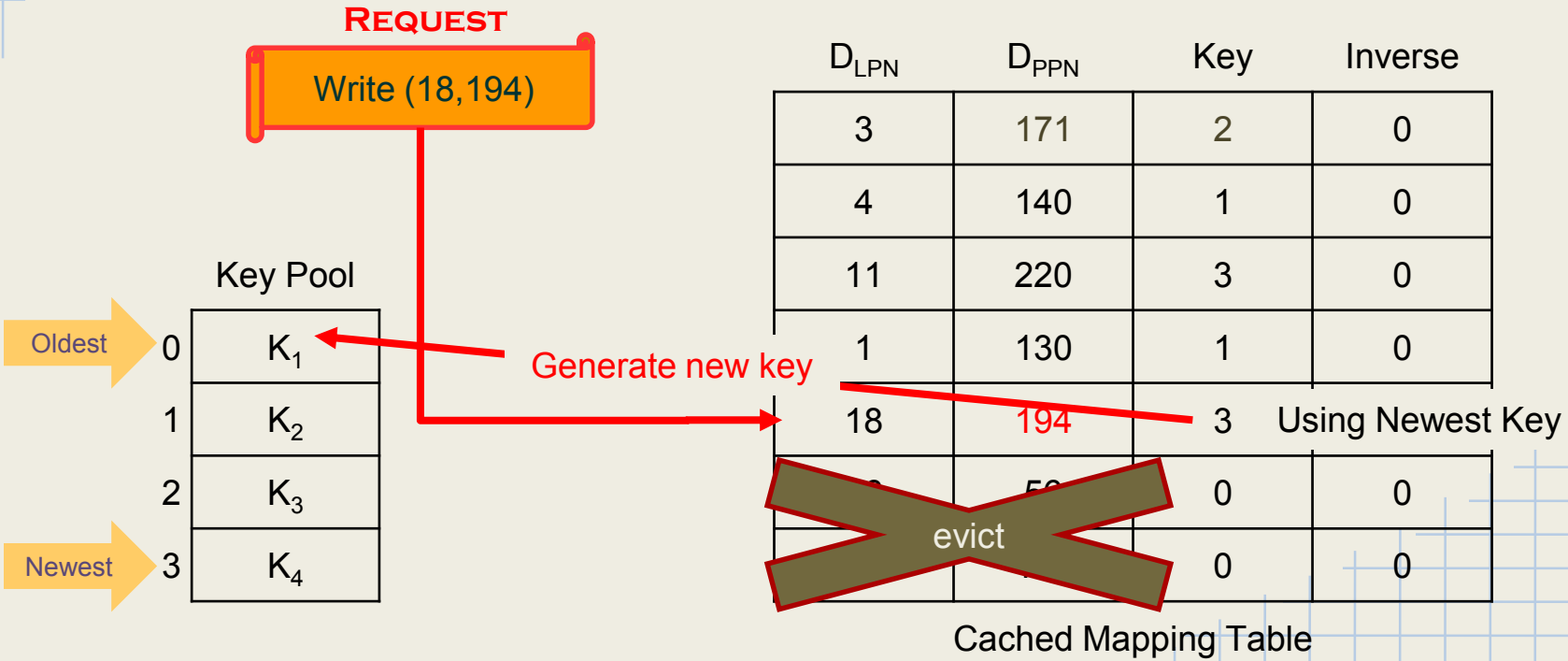
SDS+ - Example of *Inverse*



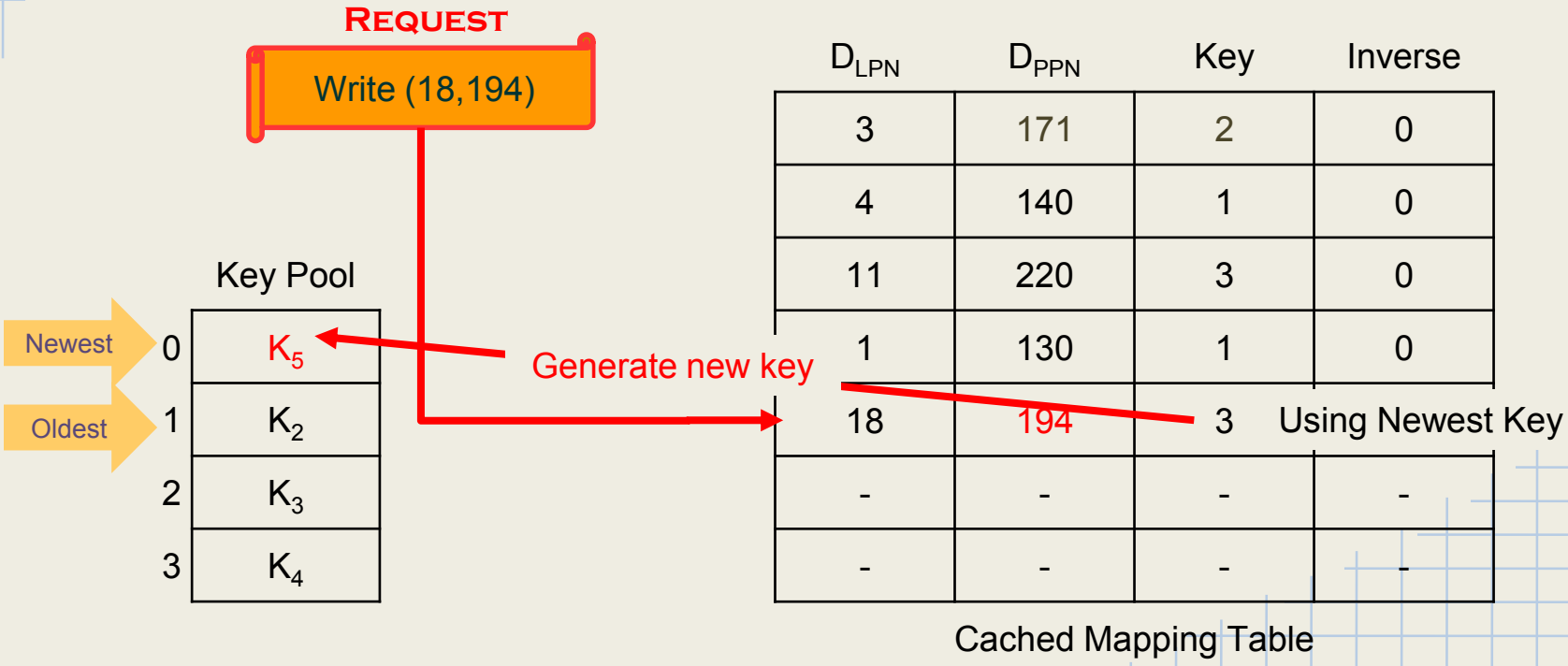
SDS+ - Example of *Inverse*



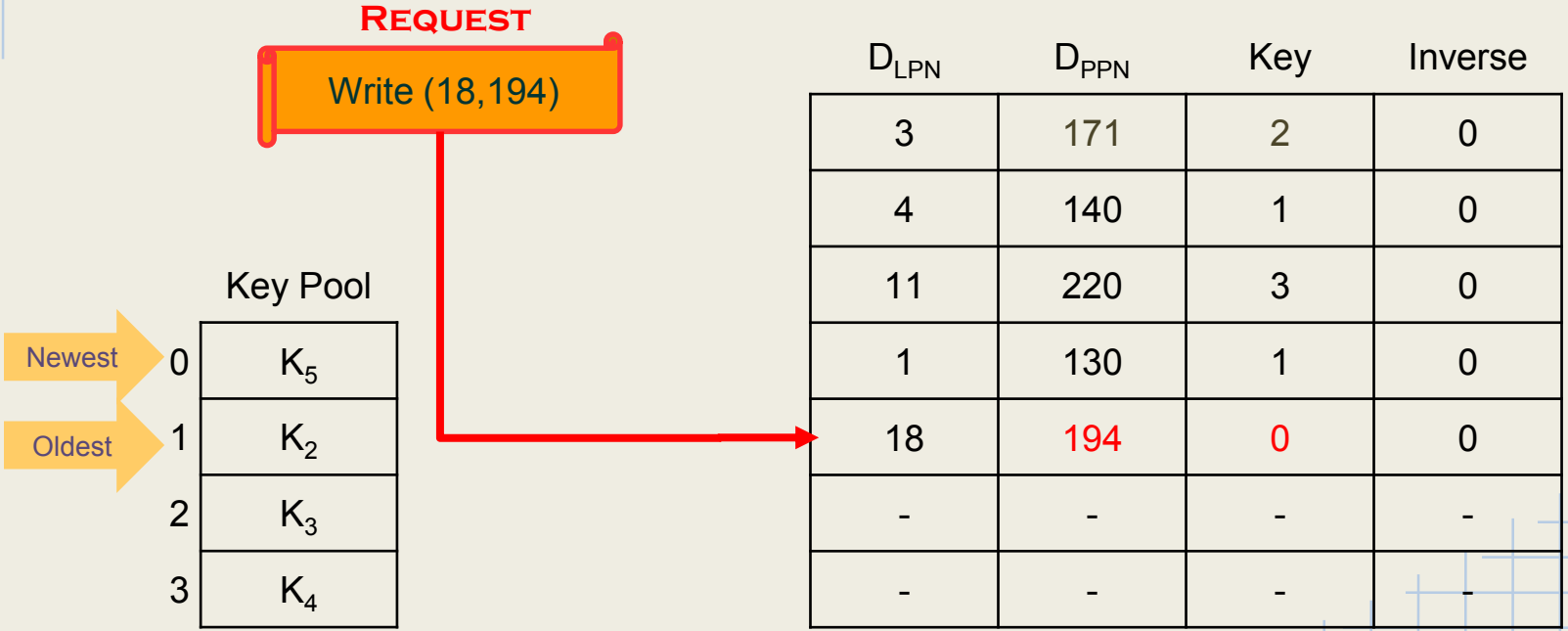
SDS+ - Example of *Inverse*



SDS+ - Example of *Inverse*



SDS+ - Example of *Inverse*



D_{LPN}	D_{PPN}	Key	Inverse
3	171	2	0
4	140	1	0
11	220	3	0
1	130	1	0
18	194	0	0
-	-	-	-
-	-	-	-

Cached Mapping Table

SDS+ - Security Analysis

- Assume the key length is x bits, the average number of unsuccessful brute force trials to crack a key is $2^{(x-1)}$ times
- Assume that the cracked key has been used to encrypt 2^y pages
- The attacker needs $2^{(x-y-1)}$ trials to access 2^y pages on average

Key length (x)	Pages use same key (y)	Time to try one key
256 bits	2^8 pages	1 ns.

  - Needs $2^{256-8-1}$ ns. = 2^{238} sec ≅ $8716 * 2^{200}$ years to decrypt 256 pages

Outline

- Introduction
- Background
- An Efficient Secure Deletion Scheme
- Performance Evaluation
  - Experimental Setup
  - Max used times of key
  - Performance
- Conclusion

Experimental Setup

- Simulate the cached mapping table for writes
  - Only page writes will cause the change of keys
- The following is the statistics of the used workloads

Workload	Detail of workload	# write request	# written LBAs	Avg. req. leng.
proj	project directories	575744	3314497	5.76
prxy	proxy server	12131406	33493485	2.76
src1	source control repository	1323340	12584875	9.51
Financial1	financial computation	4095237	10812392	2.64
WebSearch1	web server	233	1100	4.72

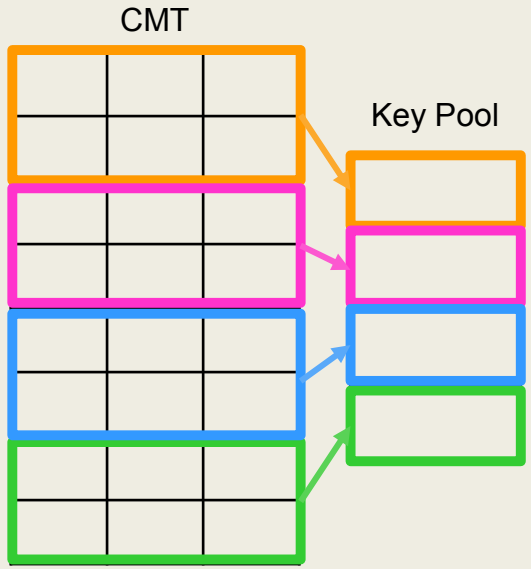
proj, prxy, src1 : D. Narayanan, A. Donnelly, and A. Rowstron. Write off-loading: Practical power management for enterprise storage. *Trans. Storage*,

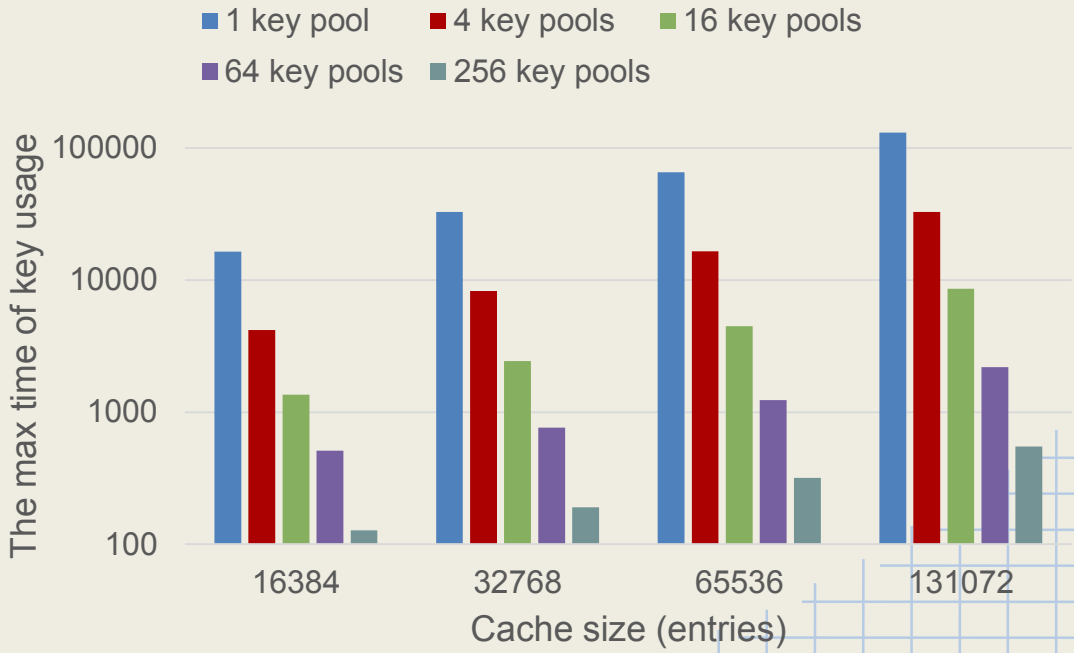
Financial1, WebSearch1 :SNIA IOTTA Repository. *MSR Cambridge Traces*, 2008

Max Used Times of Keys

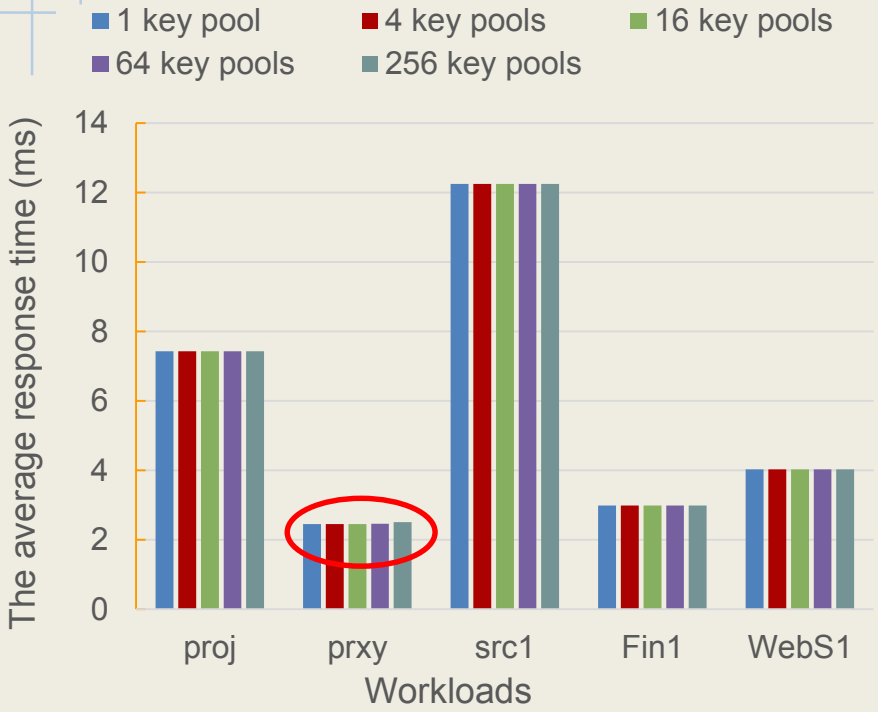
- The data security will be harmed if too many pages are encrypted by the same key
- Our goal is to let each entry use each key once then the used key key is evicted
  - That is, the ideal number of times to use a key is equal the number of cached mapping table entries
  - The larger the cache size is , the lower the security is
- We adopt multiple equal-sized small key pools, each corresponds to a part of the cache, instead of a large key pool

Max Used Times of Keys (Cont.)

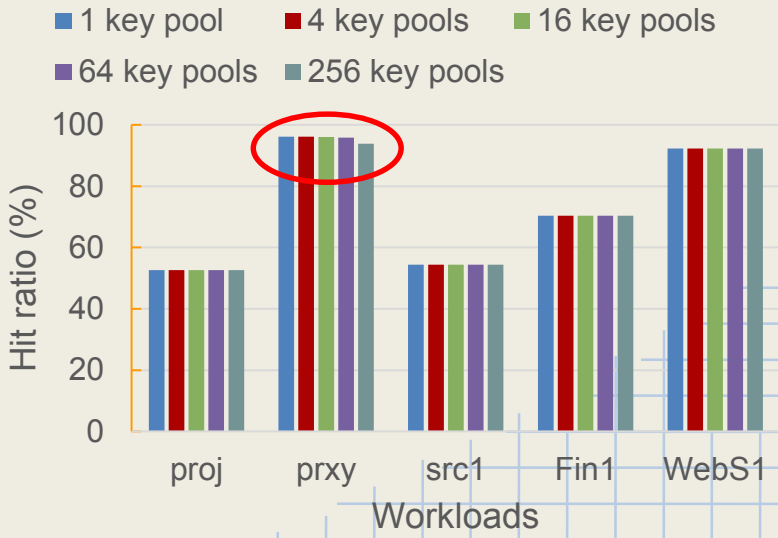




Max Used Times of Keys (Cont.)



Partition the key pool barely decrease the performance



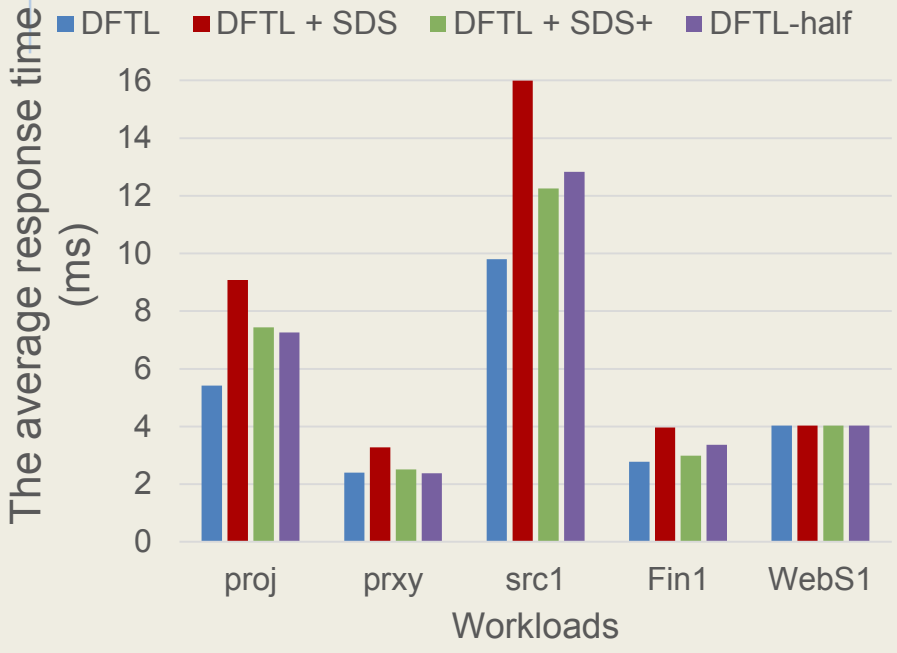
Performance

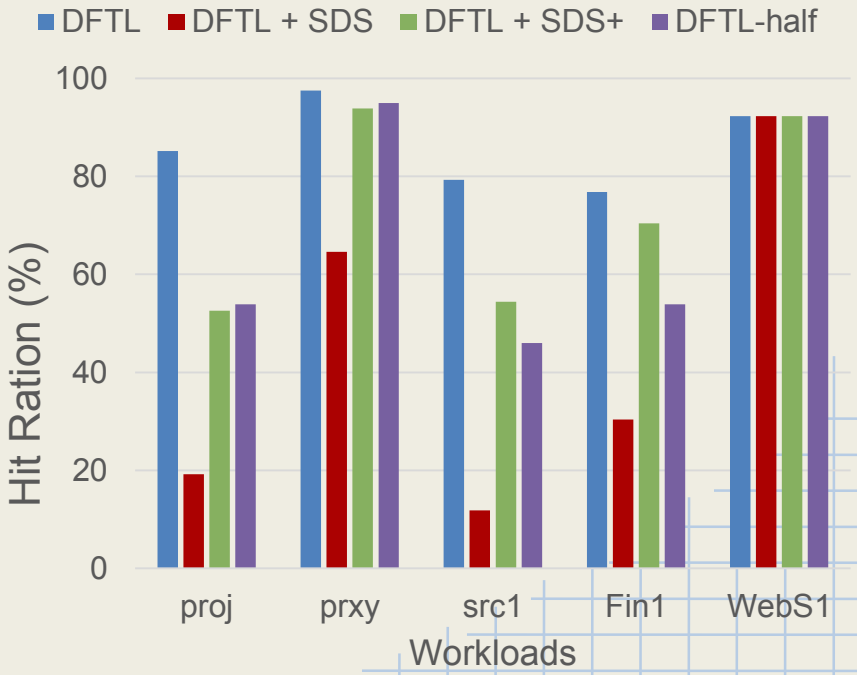
- Investigated methods:

	DFTL	DFTL + SDS	DFTL + SDS⁺	DFTL-half
Secure Deletion	No	Yes	Almost	No
Cache size	131,072 (100%)	14,563 (11.11%)	65,536 (50%)	65,536 (50%)

- SDS⁺ takes half of the cache space to maintain the key pool
- We also conduct the experiments with a DFTL having ½ cache size to show the factors affected by our key management
  - Extra number of victim pages caused new key generations

Performance





Outline

- Introduction
- Background
- An Efficient Secure Deletion Scheme
- Performance Evaluation
- Conclusion

Conclusion

- We propose SDS to enable ***fast*** secure deletion for resource-constrained flash memory storage devices.
  - With SDS, the time to delete data of a storage securely is independent of the storage capacity.
- The SDS$^+$ is proposed to enhance the read/write performance of the proposed SDS
  - SDS$^+$ effectively improve the read/write performance and still guarantees the data security
- Future work
  - Explore the possibility to extend the proposed design to data centers.

Q & A