A Reconfigurable FTL (Flash Translation Layer) Architecture for NAND Flash-Based Applications

CHANIK PARK, WONMOON CHEON, JEONGUK KANG, KANGHO ROH, and WONHEE CHO Samsung Electronics and
JIN-SOO KIM Korea Advanced Institute of Science and Technology

2024. 07. 10

Presentation by Seonju Koo & Suji Park

pigeon99@dankook.ac.kr

Sujipark@dankook.ac.kr

DANKOOK UNIVERSITY

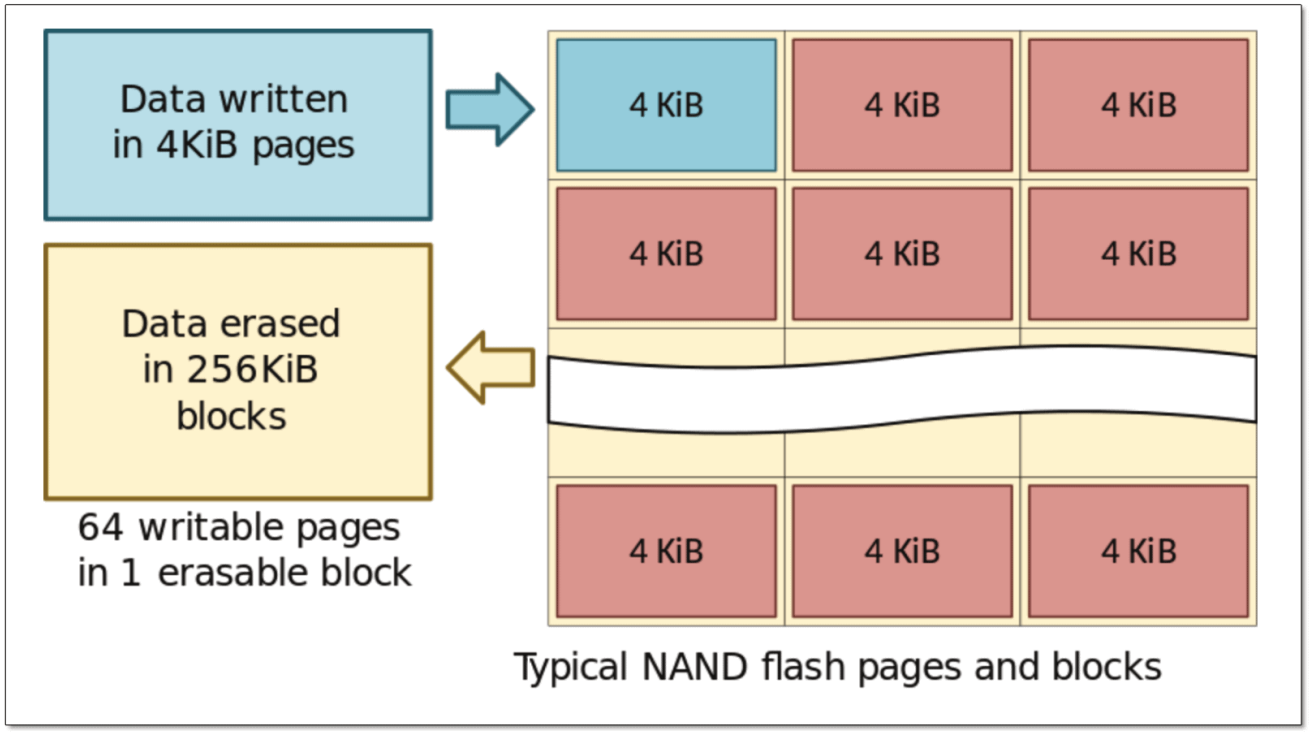
Dankook University
System Software Laboratory

Contents

1. Introuduction

1. Introduction

- NAND Flash의 특징
  - Erase-before-write

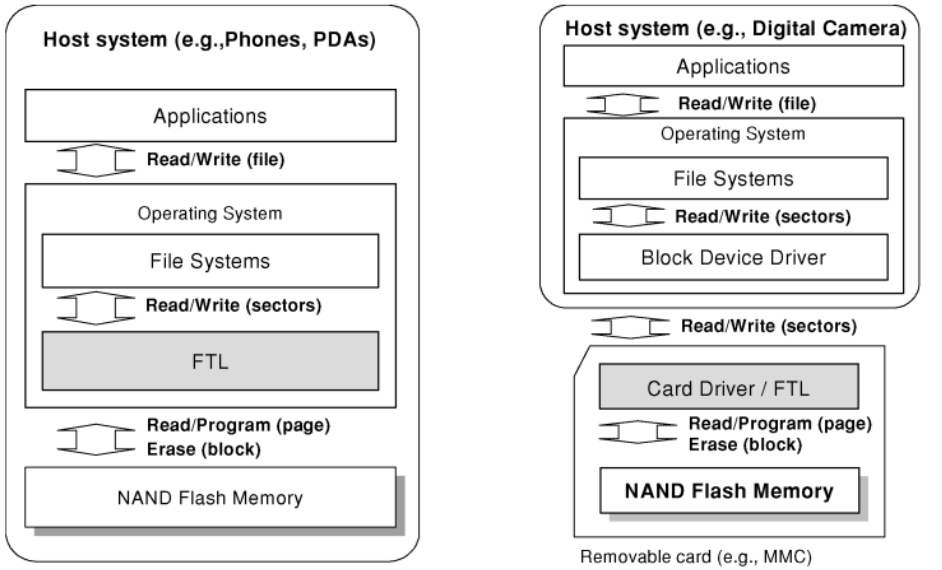


NAND flash에서 읽기, 쓰기, 삭제, 주소 매핑 등의 역할을 담당하는 펌웨어 FTL 필요

1. Introduction

- FTL 설계 시 고려해야 할 요소

  - 응용 프로그램의 제약 조건



(a) Embedded storage architecture (b) Removable storage architecture

Fig. 1. Software architectures for NAND flash-based applications.

  - 응용 프로그램의 접근 패턴

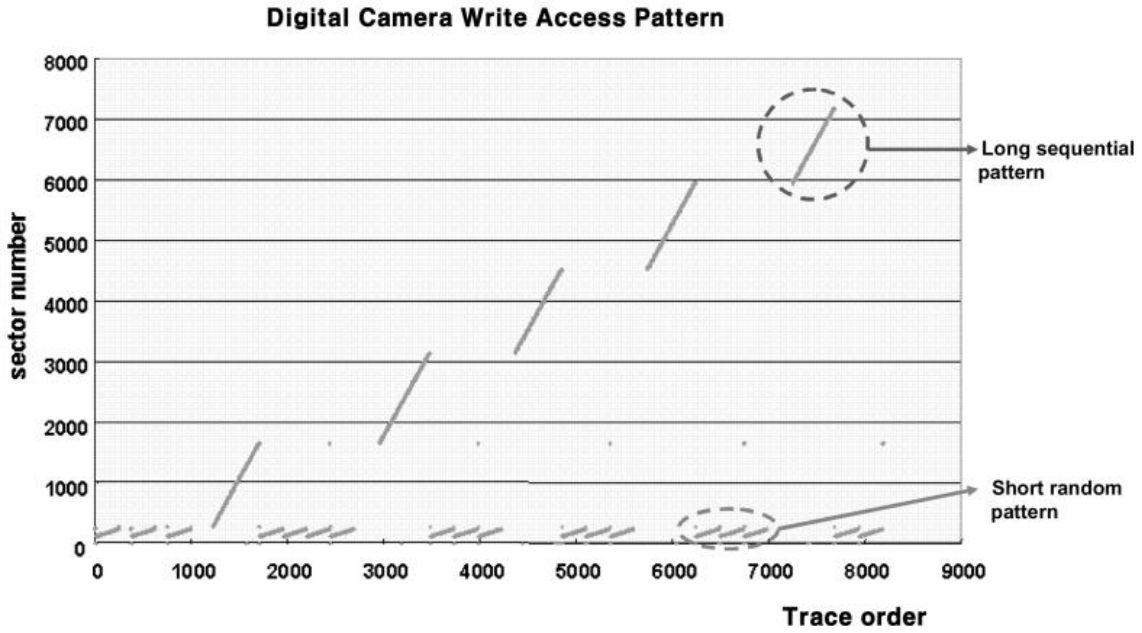


Fig. 2. An example of a workload trace from a digital camera.

2. Background

2. Background

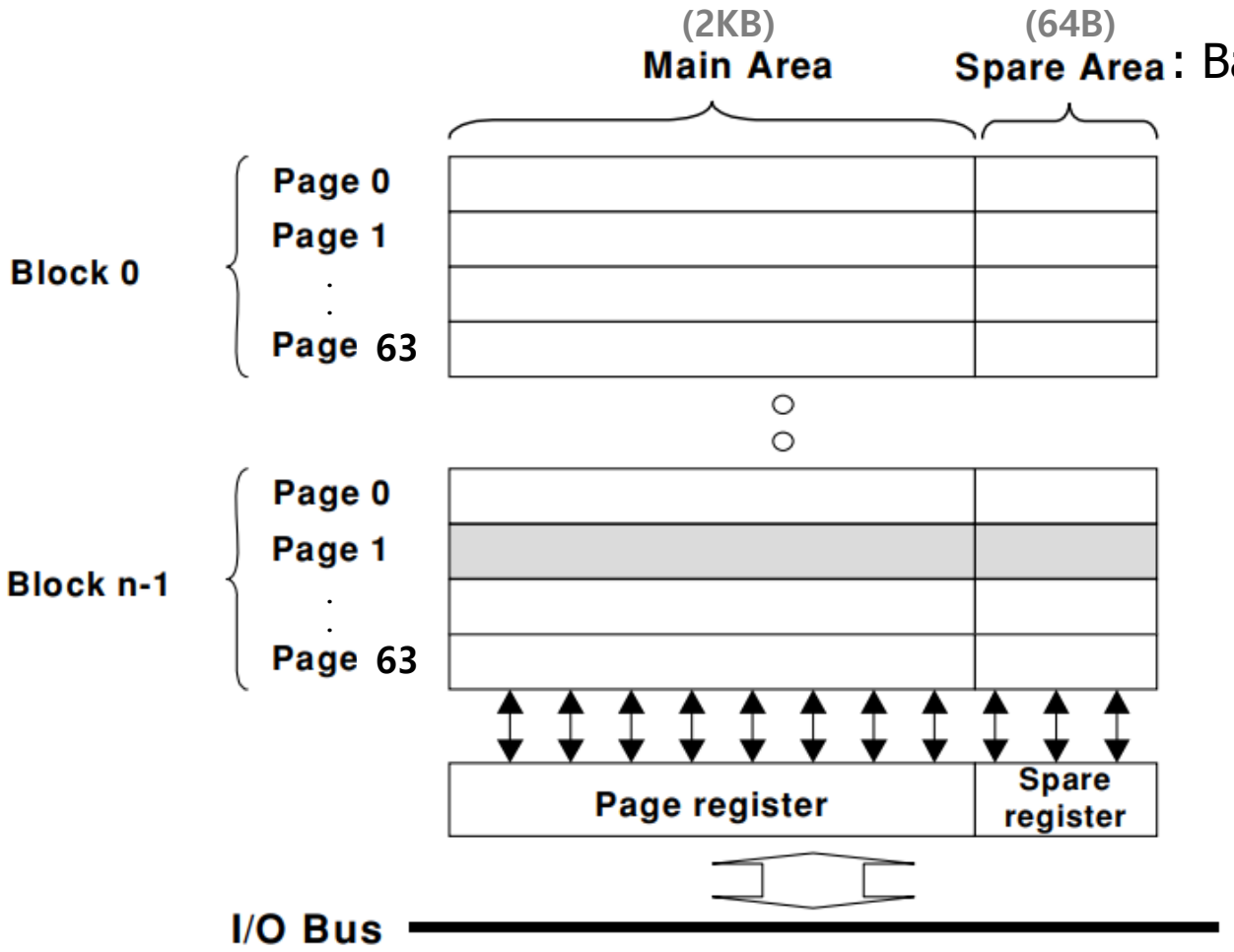


Fig. 3. NAND flash structure.

: Badblock 식별 등의 보조 정보 저장

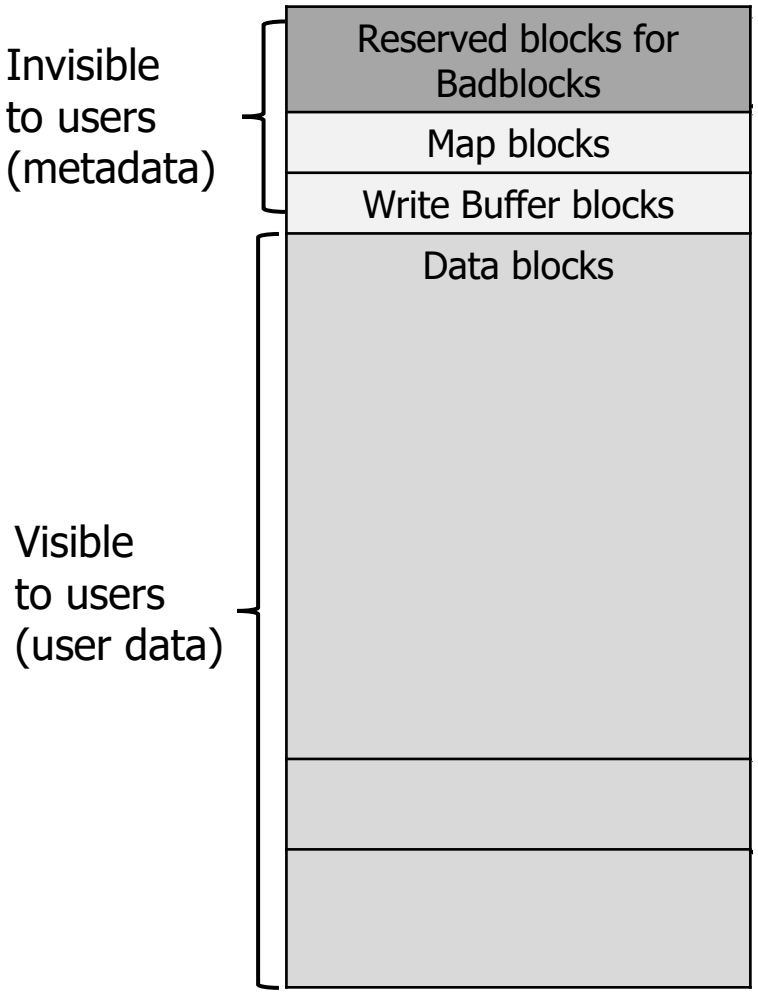
Table I. Operation Latency of NAND Flash[a]

Operation	Latency
Page read	20 us
Page program	200 us
Block erase	1.5 ms

[a]Samsung Electronics [2005].

Erase에서 많은 시간 소요 -> 횟수 줄여야 함

2. Background



FTL의 구조

Merge 과정

- ✓ 데이터를 버퍼에 기록
- ✓ 중복 값 invalid 처리
- ✓ 유효 값 copy & merge하여 새 데이터 블록 생성
- ✓ 새 데이터 블록으로 매핑 수정
- ✓ 기존 데이터 블록, 로그 블록 free

Workload {3, 1, 1, 3}

->merge에 많은 시간 소요.

3. Mapping Scheme

3. Mapping Schemes: page

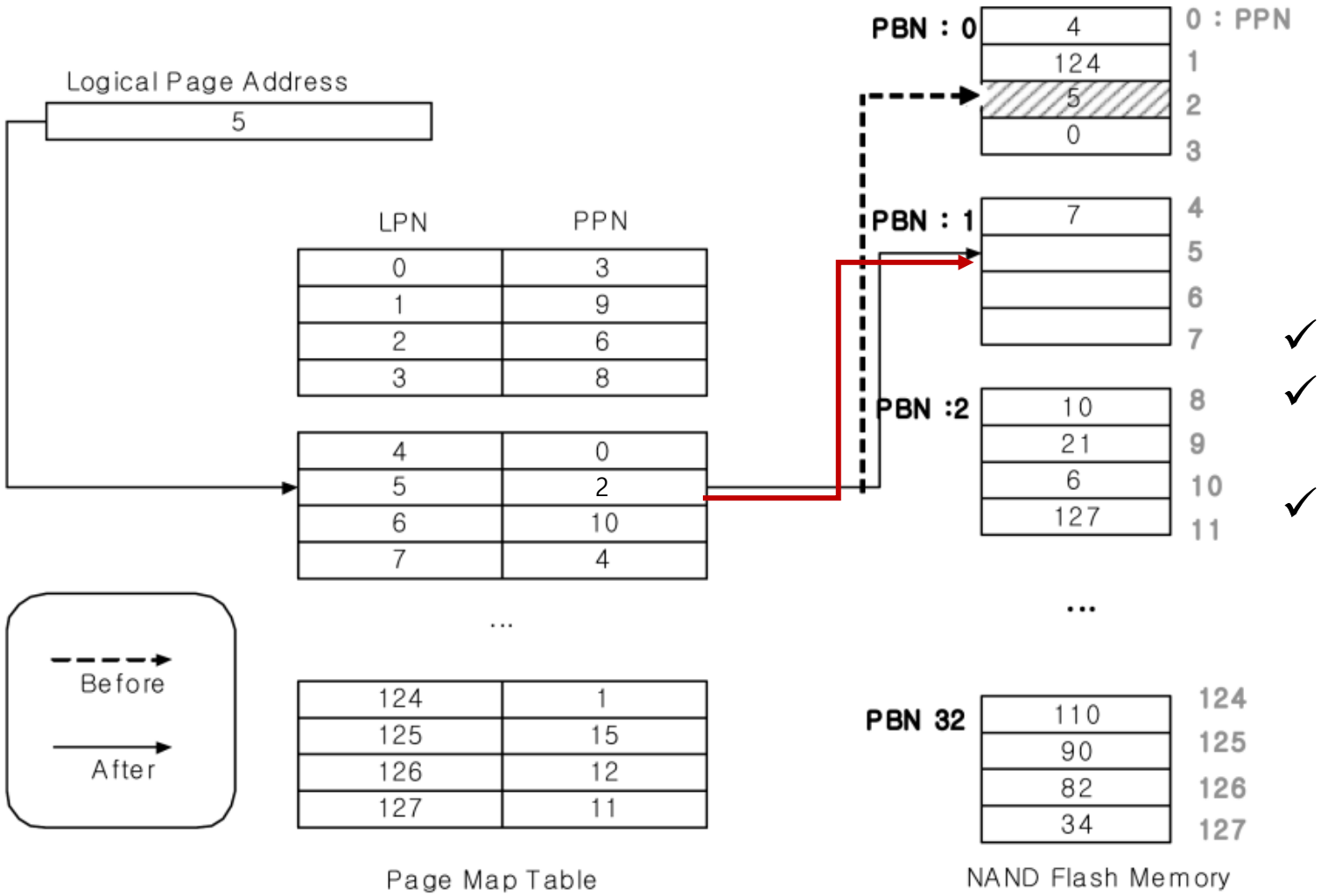


Fig. 5. Page-mapping scheme.

Page-mapping 과정

- ✓ 맵 테이블에서 주소 변환
- ✓ 기존 데이터 있으면 빈 페이지 찾아서 기록
- ✓ 테이블 매핑 변경사항 수정

3. Mapping Schemes: page

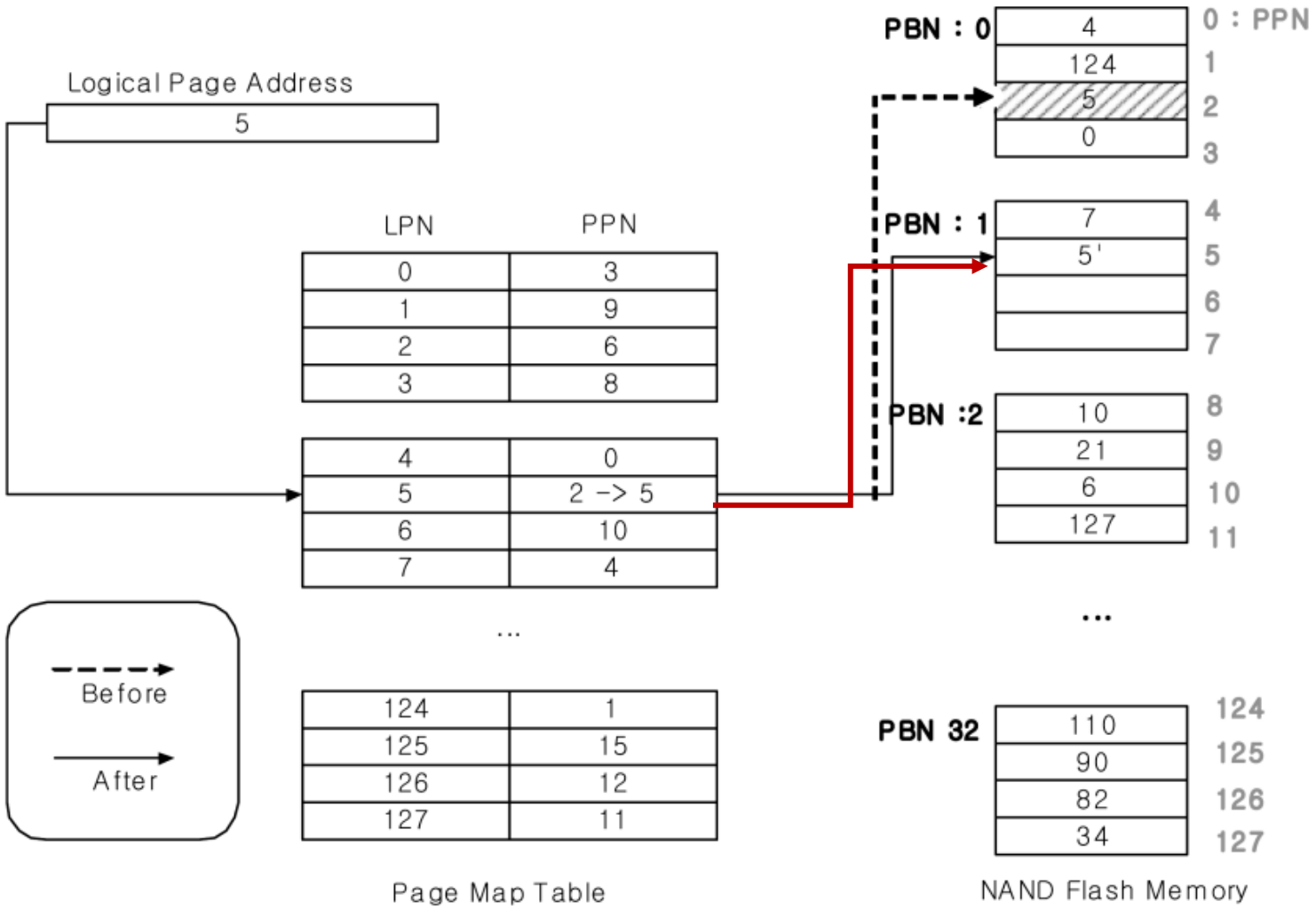


Fig. 5. Page-mapping scheme.

장점: 빈 페이지 어디에나 데이터 기록 가능

단점: **invalid** 페이지 수 많으면 회수할 때 성능 저하

맵 테이블 크기가 큼(메모리 차지)

3. Mapping Schemes: block

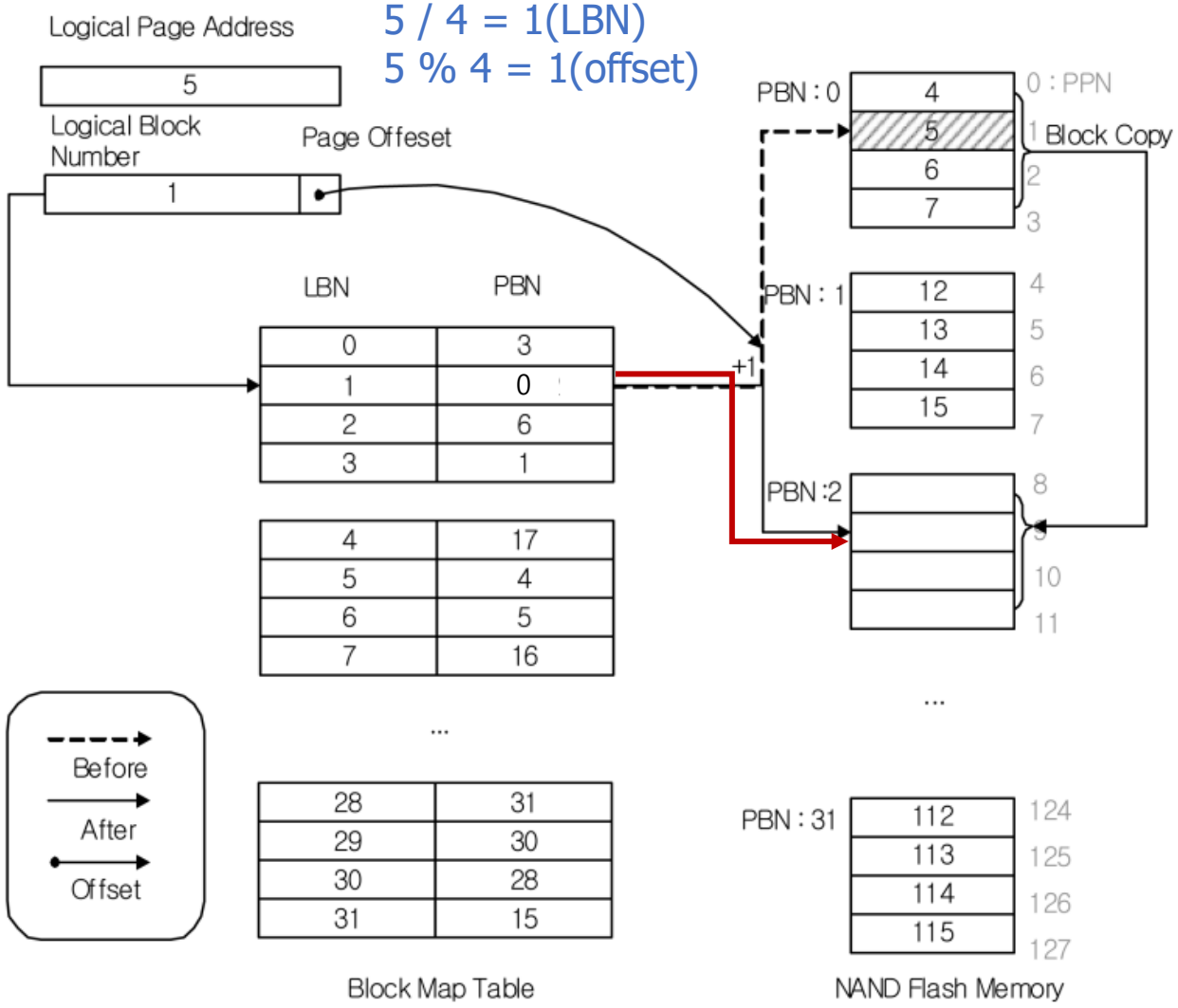


Fig. 6. Block-mapping scheme.

Block-mapping 과정

- ✓ LPN -> LBN과 오프셋으로 변환
- ✓ 맵 테이블에서 주소 변환
- ✓ 기존 데이터 있으면 빈 블록 찾아서 같은 오프셋 위치에 기록
- ✓ 테이블 매핑 변경사항 수정

3. Mapping Schemes: block

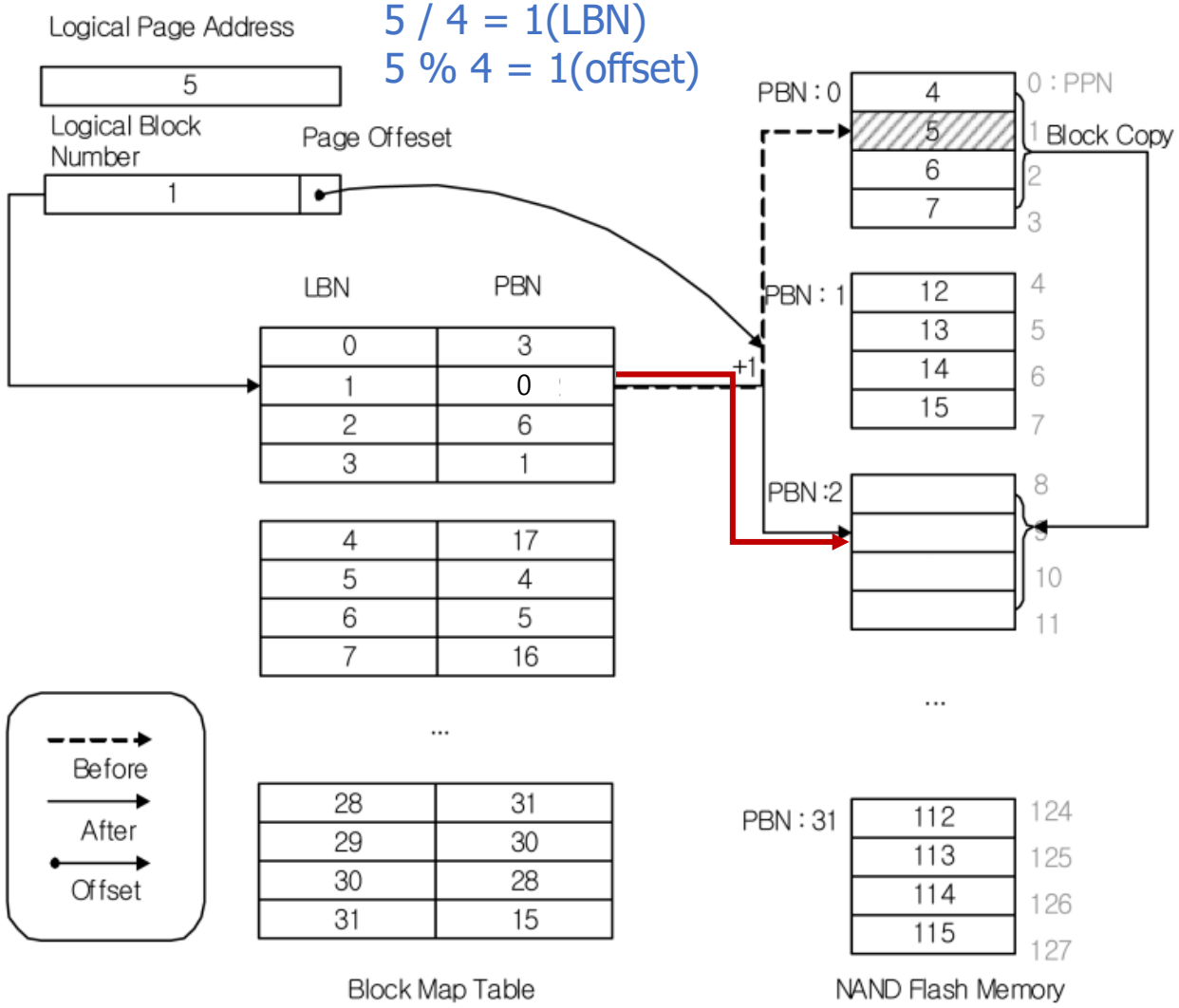


Fig. 6. Block-mapping scheme.

장점: 맵 테이블 크기 축소

단점: 빈번한 블록 수준의 복사 작업

3. Mapping Schemes: hybrid

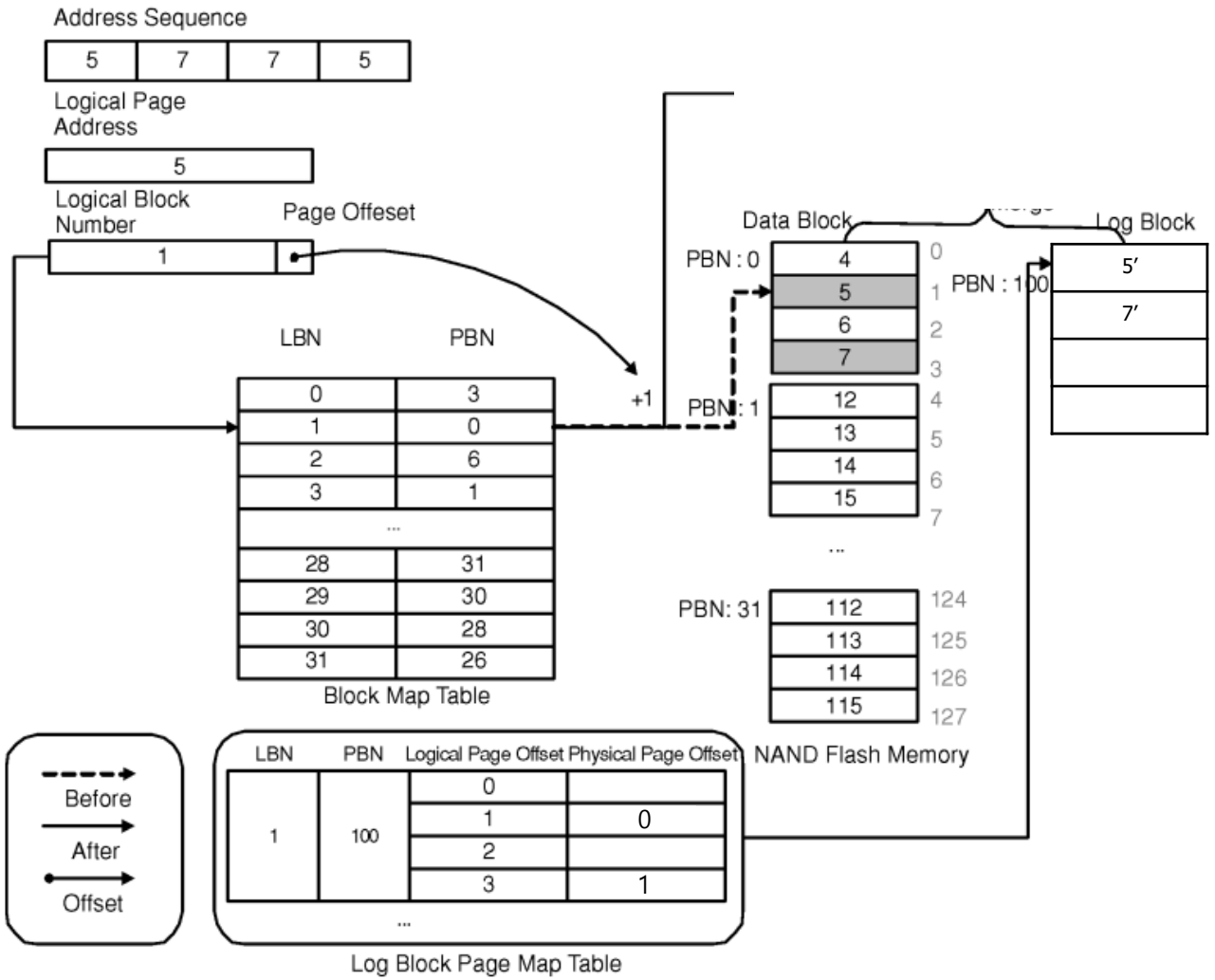


Fig. 7. Hybrid mapping scheme.

Hybrid-mapping 과정

- ✓ LPN -> LBN과 오프셋으로 변환
- ✓ 맵 테이블에서 주소 변환
- ✓ 기존 데이터 있으면 로그 블록에 기록
- ✓ 로그 블록 페이지 테이블 변경사항 수정
- ✓ 꽉 차면 새 데이터 블록에 merge
- ✓ 데이터 블록 매핑 수정

3. Mapping Schemes: hybrid

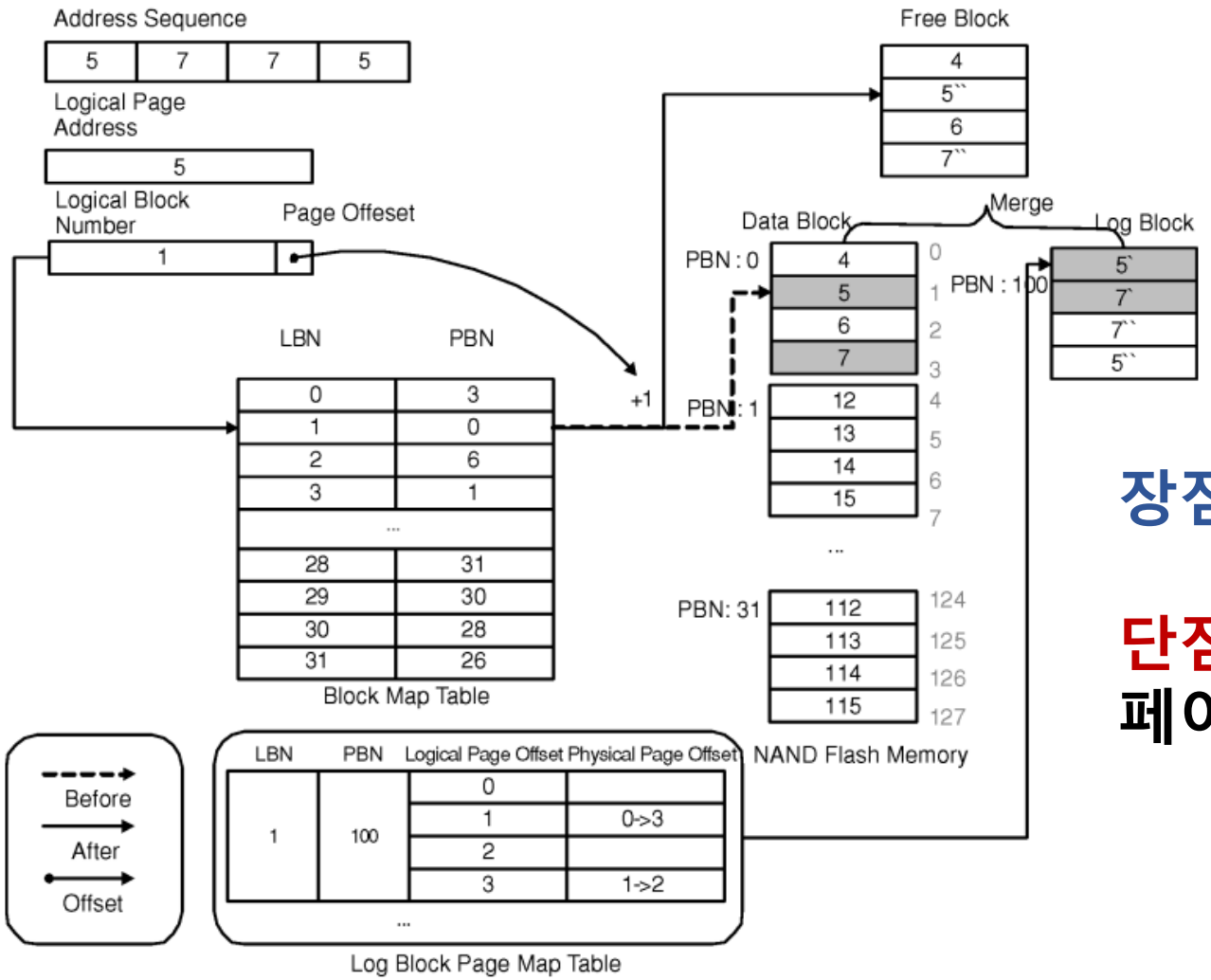


Fig. 7. Hybrid mapping scheme.

장점: 순차/랜덤 쓰기 효율적으로 처리

단점: 낮은 로그 블록 활용도로 인한 미사용 페이지 수 증가 -> **merge** 횟수 많음

3. Mapping Schemes

- **FAST(fully associative sector translation) scheme [Lee et al. 2006]**

  - 로그 블록을 모든 블록 그룹이 공유

  - 순차적 로그 블록 사용

- **Flexible management scheme [Chang and Kuo 2004]**

  - 가변 크기의 매핑 단위를 사용

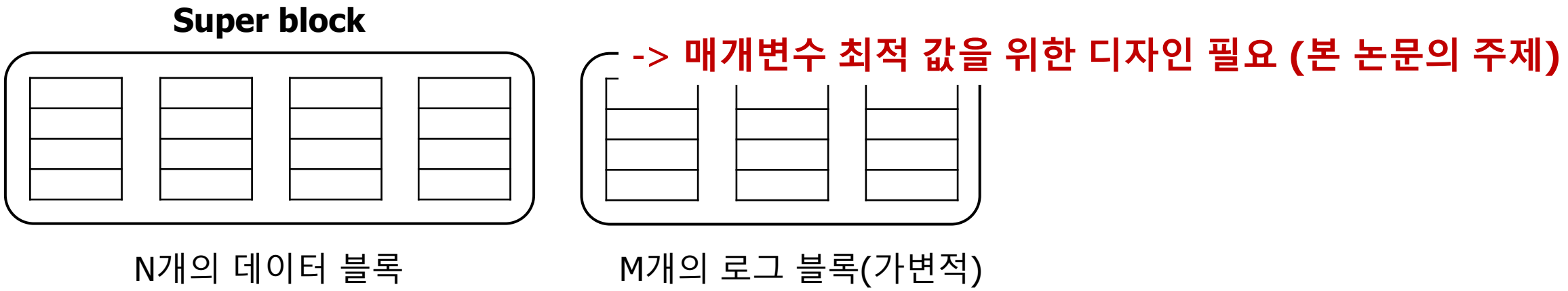
3. Mapping Schemes

- **Super Block mapping [Kang et al. 2006]**

  super block = n개의 블록으로 이루어진 데이터 그룹

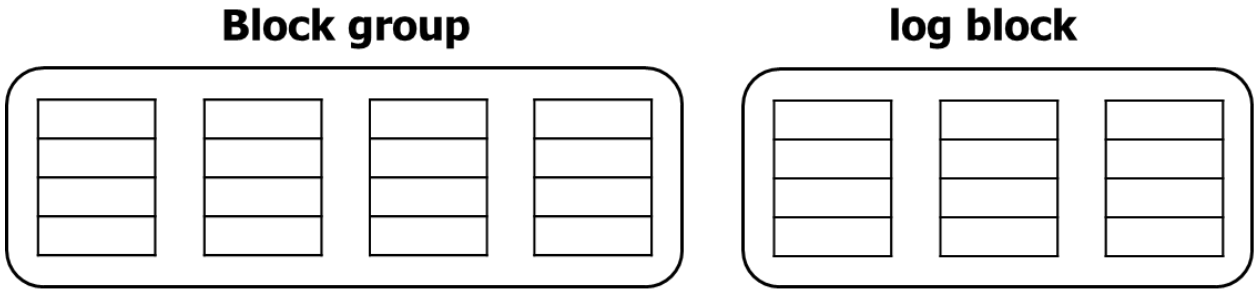
  페이지 수준에서 N + M개의 물리 블록 그룹으로 매핑 됨

  장점: 가비지 컬렉션 오버헤드 감소 단점: 매개변수 N과 M은 특정 요구에 맞게 조정 불가능



4. Flexible group mapping

4. Design



매개변수

N: 한 블록 그룹에 들어있는 데이터 블록 개수

K: 한 블록 그룹에 할당할 수 있는 로그 블록의 최대 개수

기본 구조 (super block scheme과 유사)

: N개의 데이터 블록을 가진 데이터 블록 그룹과 최대 K개의 로그 블록의 쌍

(*로그 블록은 free space에서 할당해서 가져옴)

- DBMT(데이터 블록 매핑 테이블), LBMT(로그 블록 매핑 테이블), LPMT(로그 페이지 매핑 테이블)

4. Design

```
Host requests: write (1,…),  write (1,…),  write (14,…), write (15,…),
               write (2,…),  write (2,…),  write (3,…),  write (3,…),
               write (24,…), write (25,…), write (26,…), write (27,…),
```

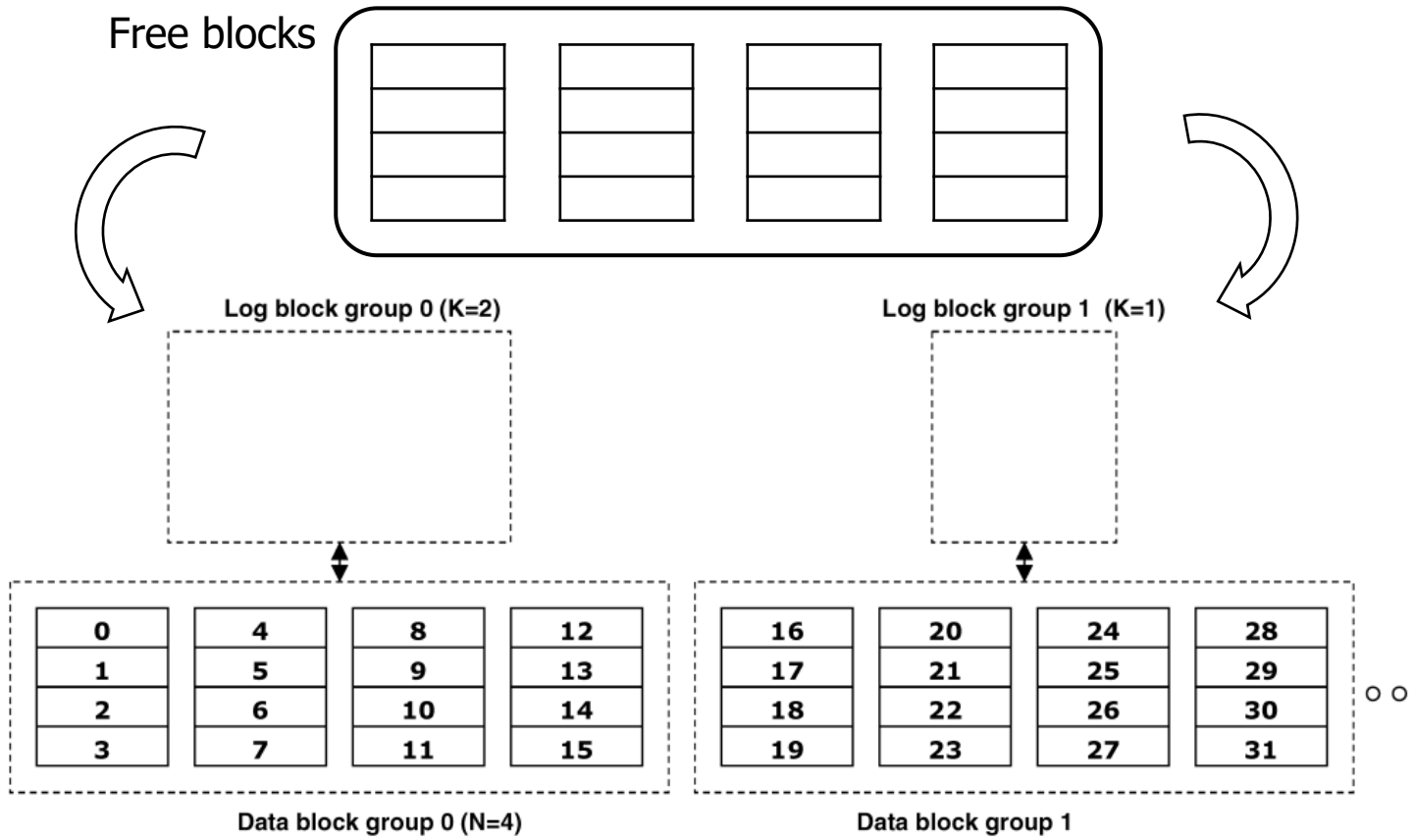


Fig. 8. Flexible group-mapping scheme.

기존 그룹에 관련된 로그가 없거나 차 있으면 Free blocks에서 새로 할당

K값 크면 : 경쟁 발생으로 merge 빈도 증가

K값 작으면: hot 페이지 처리 비효율적

N값 크면: merge 비용 증가

N값 작으면: 로그 블록 이용률 하락

->최적의 N,K값 찾아야 함

4. Design

Free Block Pool에서 로그 블록 할당 -> LBMT 업데이트 -> 데이터 기록 -> LPMT 업데이트

1) WRITE : LPN = 3, Num Of Pages = 2
2) WRITE : LPN = 11, Num Of Pages = 4
3) WRITE : LPN = 17, Num Of Pages = 4

LBN	PBN
0	100
1	101
2	102
3	103
4	104
5	105
6	206
7	303

Data Block MappingTable

DGN	PBN
0	
1	
2	

Log Block Mapping Table

DGN	LPN	PPN
0		
1		

Log Page Mapping Table

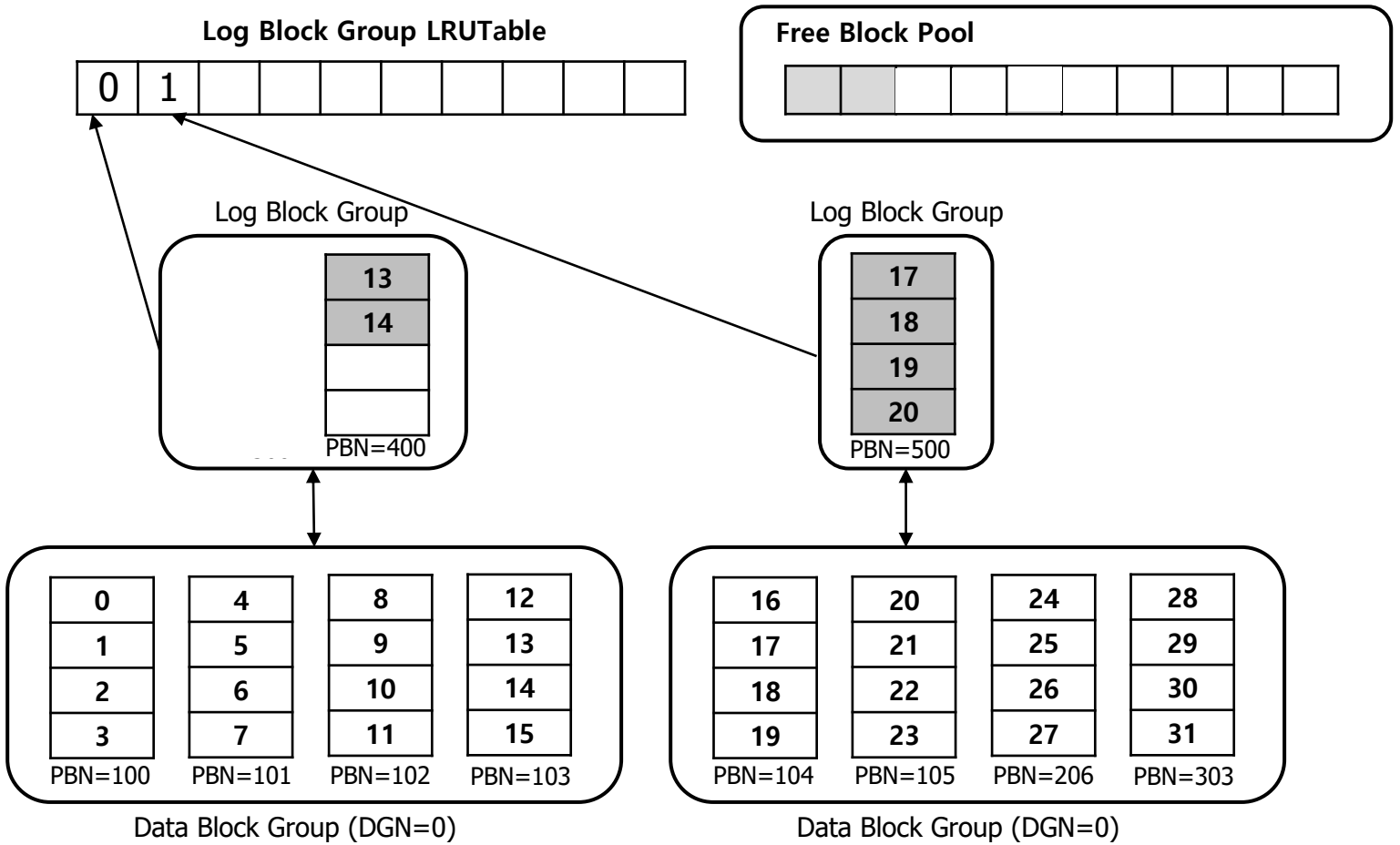


Fig. 9. Write operation in Flexible Group Mapping (N = 4, K = 2).

4. Design

각각 복사해서 새 데이터 블록에 할당하고 free 처리

LBN	PBN
0	100
1	101
2	102
3	103
4	104
5	105
6	206
7	303

Data Block Mapping Table

DGN	PBN
0	300
	400
1	500
2	

Log Block Mapping Table

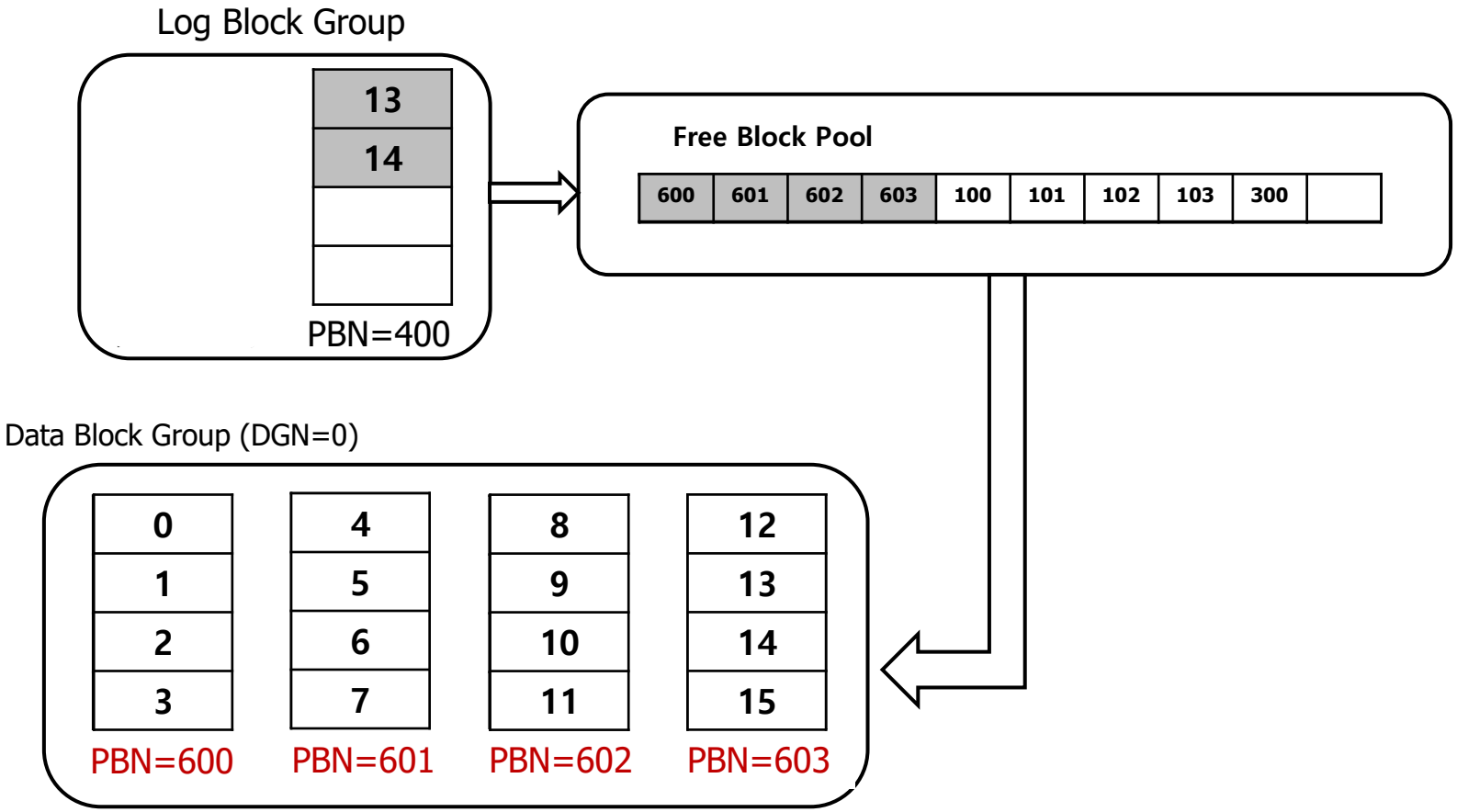


Fig. 10. An example of a simple merge operation (N = 4, K = 2).

4. Design

Log block 그대로 -> New data

기존 Data block -> Free

LBN	PBN
0	100
1	101
2	102
3	103
4	104
5	105
6	206
7	303

Data Block Mapping Table

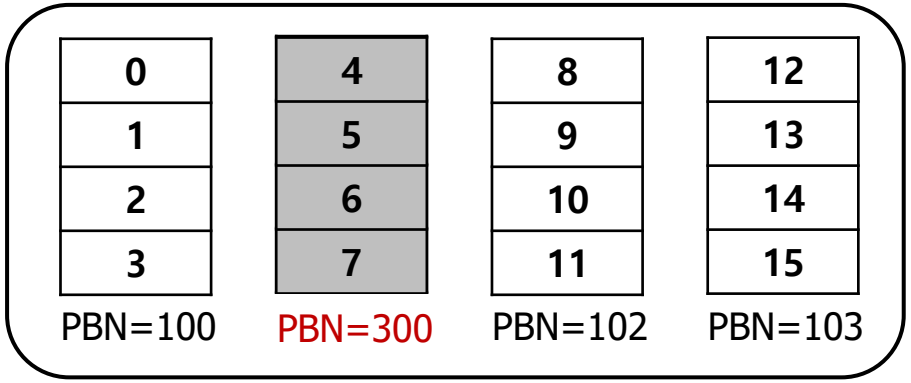


Fig. 11. An example of a swap merge operation (N = 4, K = 2).

4. Design

Log block에 데이터 복사 -> New data

기존 Data block -> Free

LBN	PBN
0	100
1	101
2	102
3	103
4	104
5	105
6	206
7	303

Data Block Mapping Table

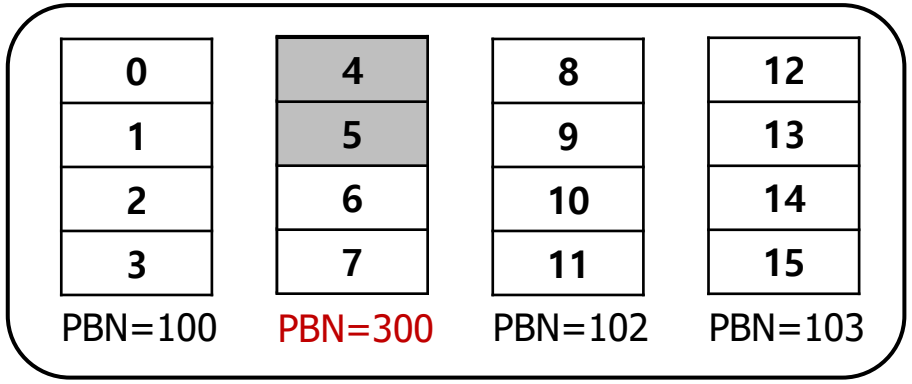


Fig. 12. An example of a copy merge operation (N = 4, K = 2).

4. Design

```
Host requests: write (1,…),  write (1,…),  write (14,…), write (15,…),
               write (2,…),  write (2,…),  write (3,…),  write (3,…),
               write (24,…), write (25,…), write (26,…), write (27,…),
```

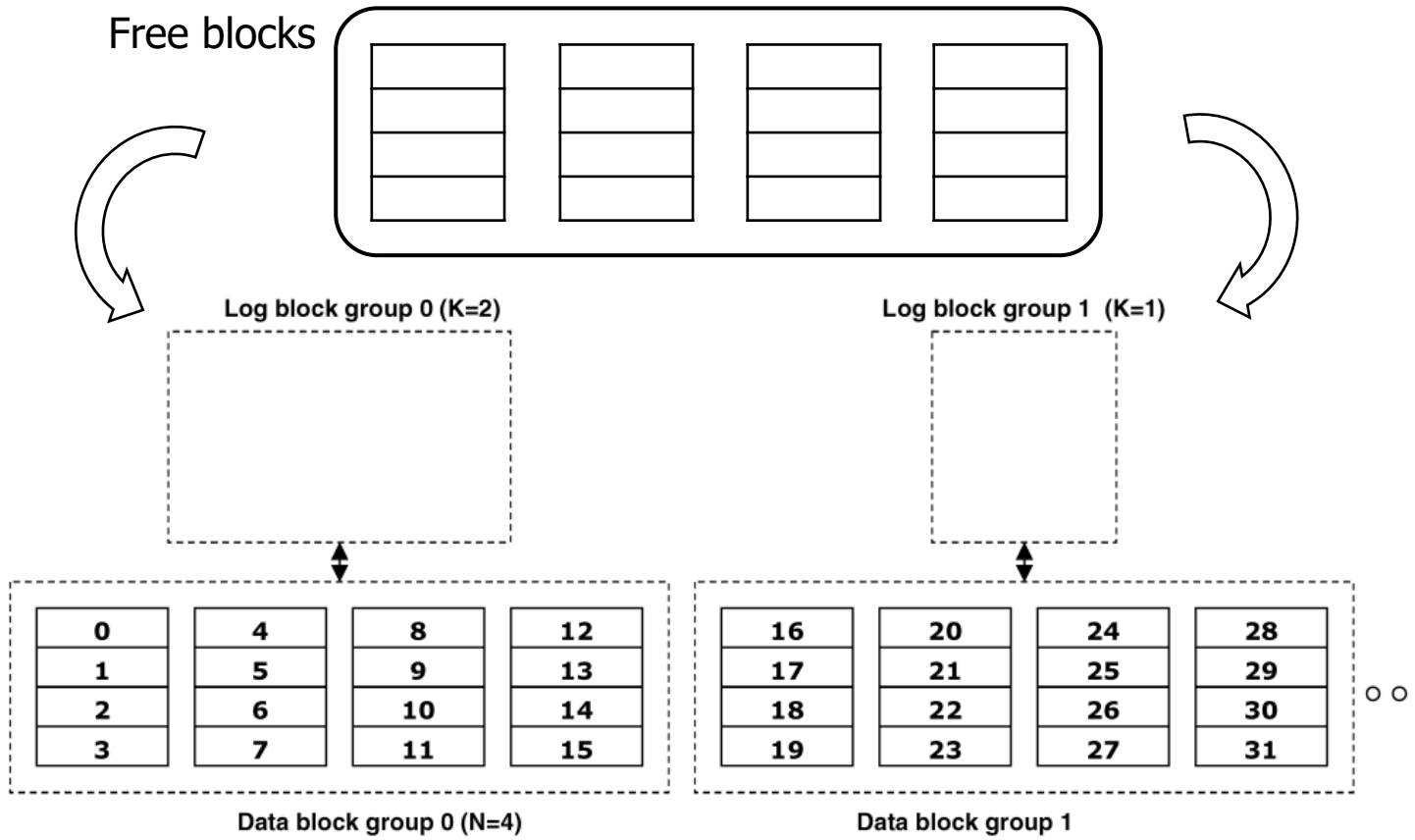


Fig. 8. Flexible group-mapping scheme.

기존 그룹에 관련된 로그가 없거나 차 있으면 Free blocks에서 새로 할당

K값 크면 : 경쟁 발생으로 merge 빈도 증가

K값 작으면: hot 페이지 처리 비효율적

N값 크면: merge 비용 증가

N값 작으면: 로그 블록 이용률 하락

->최적의 N,K값 찾아야 함

5. Evaluation

5. Evaluation

- **최적의 {N,K}를 탐색하는 효율적 방법**

- $R = < R_0, R_1, \ldots, R_{M-1} >$: sequence of write request
- W_j : <R...> 요청 포함한 각 요청창 ,
- |w|: W_j 의 크기(jth 창의 총 요청 개수)

- $C_{i,j}$: the number of requests accessed in the ith logical block

Requests (LPN) : 0, 1, 2, 3, 4, 6, 4, 6, 8, 9, 13, 15, 13, 15, 10, 12, 3, 3, 10, 13

		W_1	W_2	W_3	W_4	W_5
LBN0	Page 0	R_0				
	Page 1	R_1				
	Page 2	R_2				
	Page 3	R_3				R_{16}, R_{17}
LBN1	Page 4		R_4, R_6			
	Page 5					
	Page 6		R_5, R_7			
	Page 7					
LBN2	Page 8			R_8		
	Page 9			R_9		
	Page 10				R_{14}	R_{18}
	Page 11					
LBN3	Page 12				R_{15}	
	Page 13			R_{10}	R_{12}	R_{19}
	Page 14					
	Page 15			R_{11}	R_{13}	

(a) An example request distribution table

	W_1	W_2	W_3	W_4	W_5
LBN0	1.00	0.00	0.00	0.00	0.50
LBN1	0.00	1.00	0.00	0.00	0.00
LBN2	0.00	0.00	0.50	0.25	0.25
LBN3	0.00	0.00	0.50	0.75	0.25
N_j	1	1	2	4	4

(b) Estimating N_j

	W_1	W_2	W_3	W_4	W_5	K
LBN0	0	0	0	0	1	1
LBN1	0	1	1	1	1	1
LBN2	0	0	0	0	1	1
LBN3	0	0	0	1	2	2

(c) Estimating K_i

Fig. 13. Estimating N_j and K_i from an example trace.

5. Evaluation

- Request Density

$$\sum_{i \in \text{all LBNs}} RD_{i,j} = \frac{1}{|W|} \sum_{i \in \text{all LBNs}} C_{i,j} = 1 \quad \text{for any request window} W_j .$$

- Minimum value for a given window Wj

$$N_j = [(1/min(RD_{i,j}))] \quad \text{for } i \in \text{all LBNs.}$$

- K value predicted by temporal locality

$$K_{i,0} = 0$$
$$K_{i,j} = K_{i,j-1} + d_{i,j} \quad \text{for } j > 0$$
$$\text{where } d_{i,j} = \begin{cases} 1 \text{ if } |PAGE_{i,j}| < C_{i,j} \vee \left(\sum_{k=0..j-1} PAGE_{i,k} \cap PAGE_{i,j} \right) \neq \Phi. \\ 0 \text{ otherwise.} \end{cases}$$

Requests (LPN) : 0, 1, 2, 3, 4, 6, 4, 6, 8, 9, 13, 15, 13, 15, 10, 12, 3, 3, 10, 13

		$C_{0,0}$ W_1	W_0	W_2	W_3	W_4
LBN0	Page 0	R_0				
	Page 1	R_1				
	Page 2	R_2				
	Page 3	R_3				R_{16}, R_{17}
LBN1	Page 4	$C_{1,0}$	R_4, R_6			
	Page 5					
	Page 6		R_5, R_7			
	Page 7					
LBN2	Page 8	$C_{2,0}$		R_8		
	Page 9			R_9		
	Page 10				R_{14}	R_{18}
	Page 11					
LBN3	Page 12	$C_{3,0}$			R_{15}	
	Page 13			R_{10}	R_{12}	R_{19}
	Page 14					
	Page 15			R_{11}	R_{13}	

(a) An example request distribution table

	W_1	W_0	W_2	W_3	W_4
LBN0	1.00	0.00	0.00	0.00	0.50
LBN1	0.00	1.00	0.00	0.00	0.00
LBN2	0.00	0.00	0.50	0.25	0.25
LBN3	0.00	0.00	0.50	0.75	0.25
N_j	1	1	2	4	4

(b) Estimating N_j

	W_1	W_0	W_2	W_3	W_4	K
LBN0	0	0	0	0	1	1
LBN1	0	1	1	1	1	1
LBN2	0	0	0	0	1	1
LBN3	0	0	0	1	2	2

(c) Estimating K_i

Fig. 13. Estimating N_j and K_i from an example trace.

5. Evaluation

- Request Density

$$\sum_{i \in \text{all LBNs}} RD_{i,j} = \frac{1}{|W|} \sum_{i \in \text{all LBNs}} C_{i,j} = 1 \quad \text{for any request window} W_j .$$

- Minimum value for a given window Wj

$$N_j = [(1/min(RD_{i,j}))] \quad \text{for } i \in \text{all LBNs.}$$

- K value predicted by temporal locality

$$K_{i,0} = 0$$
$$K_{i,j} = K_{i,j-1} + d_{i,j} \quad \text{for } j > 0$$
$$\text{where } d_{i,j} = \begin{cases} 1 \text{ if } |PAGE_{i,j}| < C_{i,j} \vee \left(\sum_{k=0..j-1} PAGE_{i,k} \cap PAGE_{i,j} \right) \neq \Phi. \\ 0 \text{ otherwise.} \end{cases}$$

Requests (LPN) : 0, 1, 2, 3, 4, 6, 4, 6, 8, 9, 13, 15, 13, 15, 10, 12, 3, 3, 10, 13

		W_1	W_2	W_3	W_4	W_5
LBN0	Page 0	R_0				
	Page 1	R_1				
	Page 2	R_2				
	Page 3	R_3				R_{16}, R_{17}
LBN1	Page 4		R_4, R_6			
	Page 5					
	Page 6		R_5, R_7			
	Page 7					
LBN2	Page 8			R_8		
	Page 9			R_9		
	Page 10				R_{14}	R_{18}
	Page 11					
LBN3	Page 12				R_{15}	
	Page 13			R_{10}	R_{12}	R_{19}
	Page 14					
	Page 15			R_{11}	R_{13}	

1) Wj동안 LBN i 의 하나 이상의 페이지가 2번 이상 update

2) 이전 윈도우의 같은 페이지에 update

(a) An example request distribution table

	W_1	W_2	W_3	W_4	W_5
LBN0	1.00	0.00	0.00	0.00	0.50
LBN1	0.00	1.00	0.00	0.00	0.00
LBN2	0.00	0.00	0.50	0.25	0.25
LBN3	0.00	0.00	0.50	0.75	0.25
N_j	1	1	2	4	4

(b) Estimating N_j

	W_1	W_2	W_3	W_4	W_5	K
LBN0	0	0	0	0	1	1
LBN1	0	1	1	1	1	1
LBN2	0	0	0	0	1	1
LBN3	0	0	0	1	2	2

(c) Estimating K_i

Fig. 13. Estimating N_j and K_i from an example trace.

5. Evaluation

- **성능 평가 모델**

$$\sum_{AG_k \in SAG(W_j)} L(AG_k) > LB$$

병합 연산은 시스템에서 사용 가능한 최대 로그 블록 수 LB를 초과할 경우 발생한다.

$$|SAG(W_j)| \times K^{\partial} > LB \quad \text{for } \partial (0 \le \partial \le 1)$$

로그 블록 수는 K보다 작다.

$$C \times \frac{K^{\partial}}{N^{\varepsilon}} > LB \quad \text{for a constant } C.$$

- ∂
  0: 활성그룹 증가
  1: 병합빈도 증가
- ε
  0: 활성그룹 증가
  1: 병합빈도 증가

- **Memory requirement**

$$Memory\ requirement = |SAG(W_j)| \times (c_0 + c_1 N + c_2 K)$$

- LB : 시스템에서 사용 가능한 최대 로그 블록 수
- Ak : Rk에 의해 접근된 활성 데이터 블록
- AGk : Rk에 의해 접근된 활성 데이터 블록 그룹, 항상 Ak를 포함한다.
- SA(Wj) 요청 Wj 에서 접근된 활성 블록의 집합
- SAG(Wj): 요청 Wj 에서 접근된 활성 블록 그룹의 집합
- L(AGk): AGk 와 연관된 로그 블록의 수
- C0: log group 내부 변수
- c1N: Map table 크기
- c2K :로그 블록 내부 유효 최대 페이지

6. Experiment

6.1. Experiment Setup

- CPU: Intel Pentium-4
- Memory: 512 MB of RAM
- Disk: 80 GB
- OS: Windows XP
- File system: NTFS
- Tracing tool : In-house monitoring tool

6. Experiment

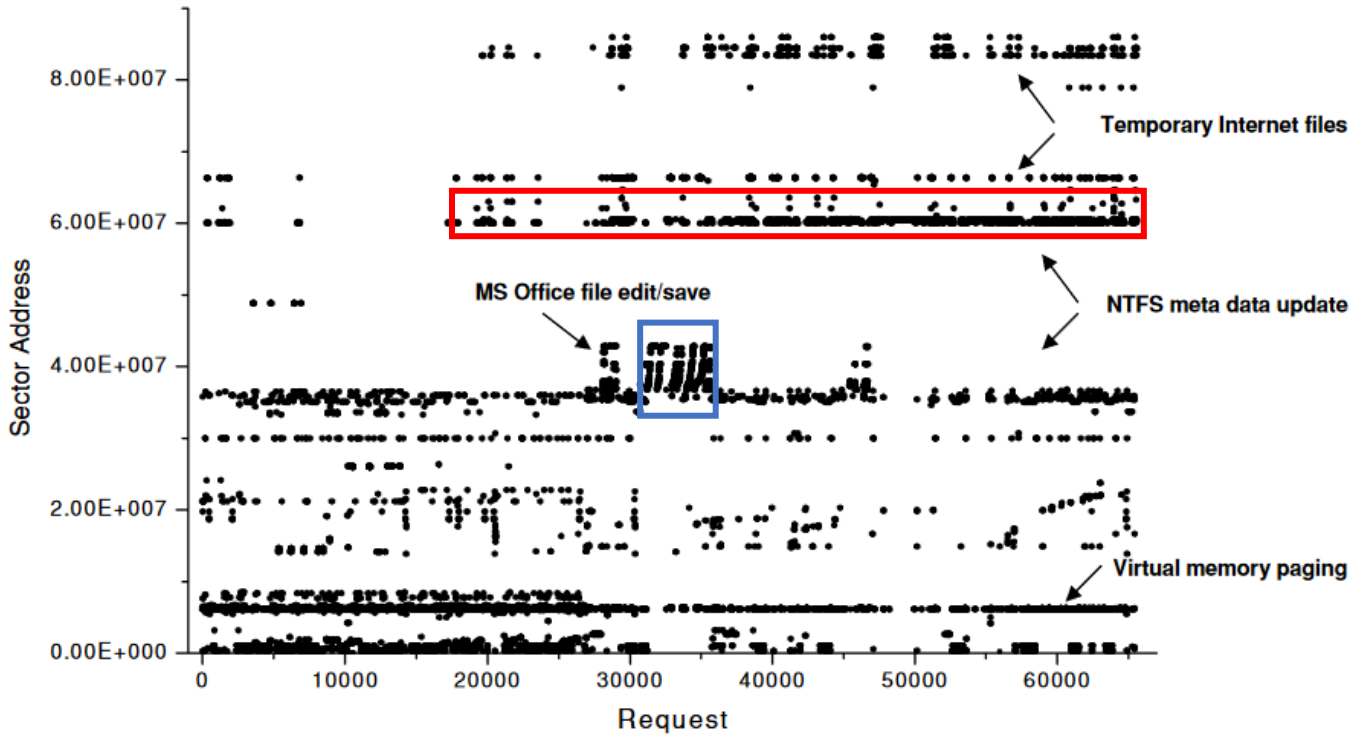


Fig. 14. Trace distribution from PC applications.

Locality

: Temporal

: Spatial

- PC Application에서는 동시 실행되는 응용 프로그램들로 인해 순차보다 랜덤 접근이 많이 발생한다.

6. Experiment

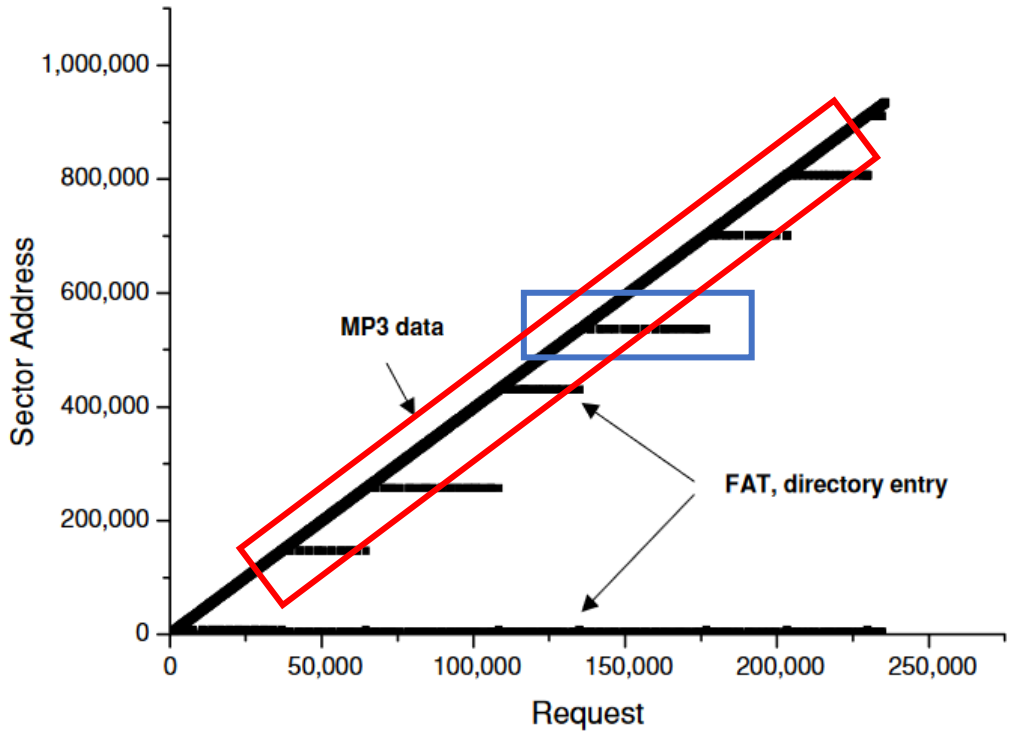


Fig. 15. Trace distribution from an MP3 download.

Access Pattern

: Sequential

: Random

- 반면, MP3 파일 다운로드 사용 사례는 파일 시스템 메타데이터(ex: FAT, directory 항목) 업데이트로 인해 소규모 무작위 요청이 있음에도 불구하고 대부분 순차적 액세스 패턴을 나타낸다.

6. Experiment

- PC 응용 프로그램은 MP3 응용 프로그램보다 더 높은 연관성을 가지며, 2에서 8 사이의 N 값과 4에서 8 사이의 K 값이 최적의 성능을 보인다.

- MP3 응용 프로그램은 주로 순차적 쓰기 패턴을 보이며, N 값은 1에서 2, K 값은 10에서 30 사이가 최적의 성능을 보인다.

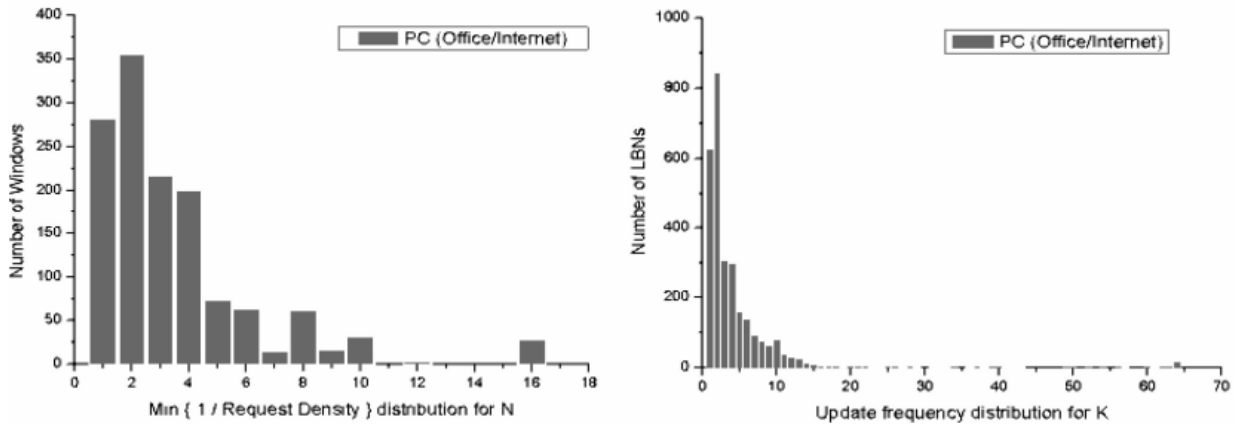


Fig. 16. Distribution of N's and K's for PC applications.

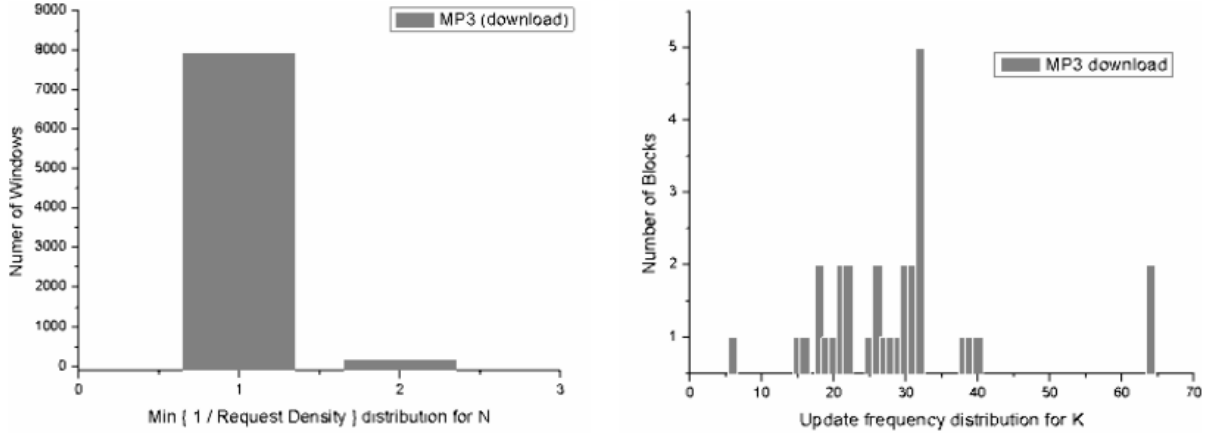


Fig. 17. Distribution of N and K values for the MP3 application.

6. Experiment

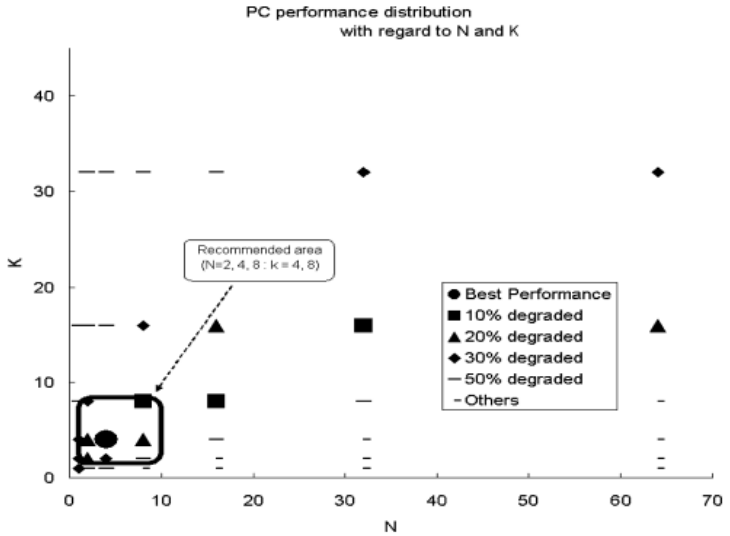


Fig. 18. PC application performance variation with the change of N and K.

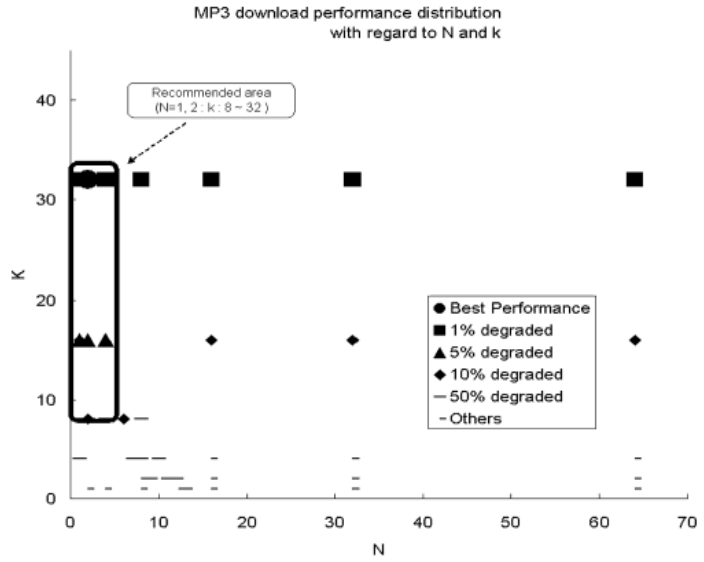


Fig. 19. MP3 download performance variation with the change of N and K.

- PC는 N의 영향이 크지 않고, **K가 작을 때** 최상의 성능을 보인다.
  - 다양한 접근 패턴에서는 데이터 블록 수보다 로그 블록의 효율적인 관리가 더 중요하다.
  - 작은 K값에서는 로그블록이 더 적절하게 유지관리 될 수 있다.
- MP3는 N의 영향을 거의 받지 않고, **K가 클 때** 최상의 성능을 보인다.
  - 연속적인 데이터 접근에는 데이터 블록 수보다 연속된 작업을 처리하는 로그 블록의 수가 더 중요하다.
  - 블록의 수가 적으면 병목현상이 발생하고, 많으면 불필요한 오버헤드가 발생하여 성능 저하를 가져온다.

7. Conclusion

7. Conclusion

- 본 논문에서는 NAND 플래시 기반 응용 프로그램의 성능과 수명을 향상시키기 위해 새로운 FTL 아키텍처를 제안한다.

- 다양한 워크로드 분석을 통해 FTL 매핑 매개변수(N, K)의 최적 구성을 결정하는 재구성 가능한 아키텍처를 설계하였으며, 데이터 블록과 로그 블록의 연관성을 기반으로 설계 공간을 효율적으로 탐색한다.

- 실험 결과, 제안된 아키텍처가 MP3부터 PC 응용 프로그램까지 다양한 워크로드에 대해 최적의 성능을 제공하며, 제안된 분석 방법이 주어진 시간 내에 최적의 N 및 K 값을 효율적으로 찾을 수 있음을 보여준다.

Thank you