

Using Next Generation NVRAM as a Write Buffer for Flash Memory-based Storage Devices

강 수 용
한양대학교





Motivation

❖ Solid-State Disk 성능

- 낮은 random write 성능
- Random write에 대한 효율적 관리 필요

Device	Sequential	Random 8KB	Price \$	Power	iops/\$	iops/watt
SCSI 15k rpm	75 MBps	200 iops	500\$	15 watt	0.5	13
SATA 10k rpm	60 MBps	100 iops	150\$	8 watt	0.7	12
Flash - read	53 MBps	2,800 iops	?? 400\$	0.9 watt	7.0	3,100
Flash - write	36 MBps	27 iops	?? 400\$	0.9 watt	0.07	30

< Jim Gray, "Flash Disk Opportunity for Server-Application", Microsoft Research, 2007 >



Motivation

❖ 플래시 메모리를 위한 차세대 NVRAM 의 활용

- 메타데이터 저장소로 활용
 - Mapping info. 저장에 따른 부팅시간의 단축
 - 메타데이터에 대한 접근을 흡수
- 쓰기 버퍼로 활용
 - Flash memory에 대한 접근 횟수 감소
 - Random write 성능 향상
- 기타 방안으로 취급되어서는 안될 다른 활용 방안들
- 기타 활용 방안들



NVRAM Write Buffer for Hard Disks

❖ 하드 디스크를 위한 쓰기 버퍼 알고리즘

- 고려 사항
 - 시간 지역성
 - 접근 횟수를 줄임
 - LRW (Least Recently Written page) 등
 - Metric : buffer hit ratio
 - 공간 지역성
 - 접근 비용을 줄임
 - LST(Largest Segment per Track), CSCAN 등
 - Metric : delay
- Stack Model : LRW + LST
- WOW : LRW + CSCAN



Differences between HDD & SSD

❖ H/W의 관점 : well known

❖ S/W의 관점 : Mapping Layer 의 존재

- Mapping Layer의 형태: FTL, Filesystem
- Mapping 방식: 블록 맵핑, 섹터 맵핑
- 잦은 랜덤 쓰기 발생시 merge로 인한 성능 저하
 - Extra operations (erase, valid page copy) 발생



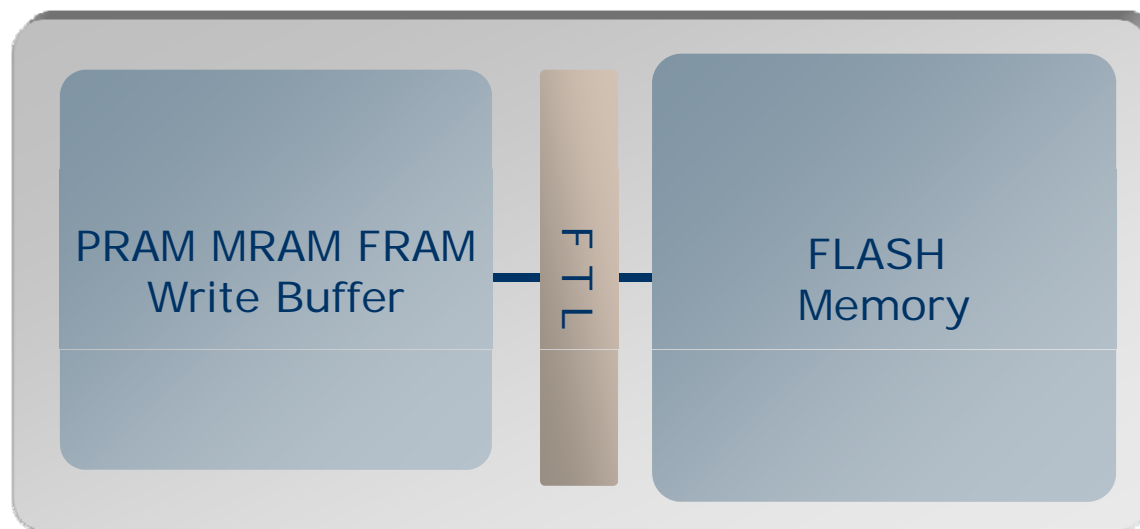
NVRAM Write Buffer for Flash Memory

❖ 시간 지역성 고려

- Try to increase the buffer hit ratio

❖ Mapping 기법의 동작 특성 고려

- Try to reduce the No. of extra operations



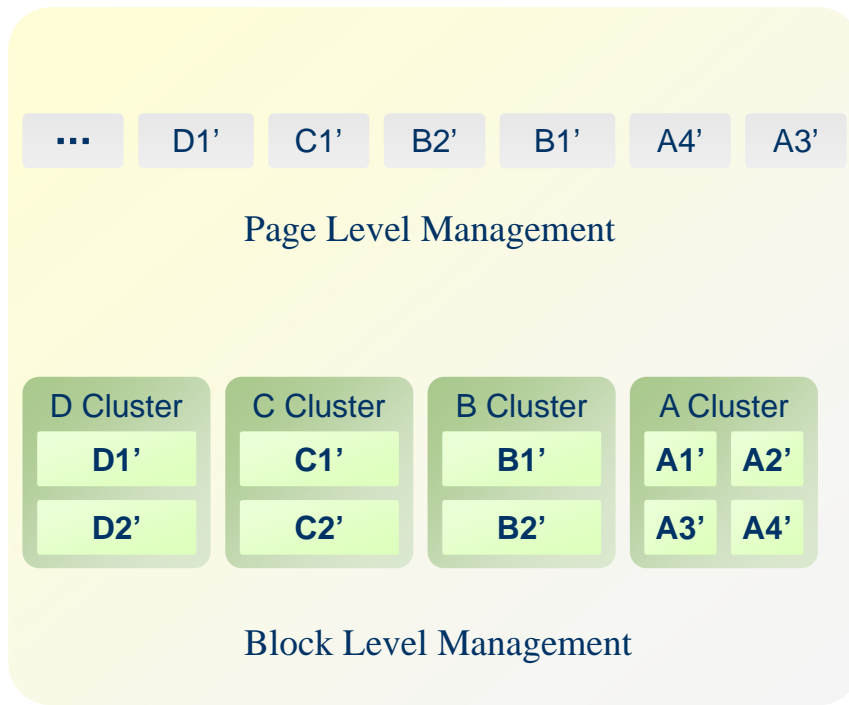


How to reduce the number of extra operations?

Necessity of Page Clustering



NVRAM



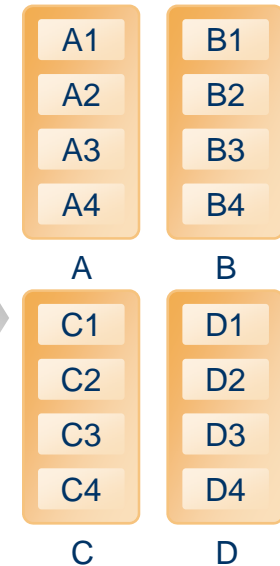
NVRAM Management

write A3' to L1
 write A4' to L1
 write B1' to L2
 write B2' to L2
 Merge A and L1
(smart Copy)
 write C1' to L1

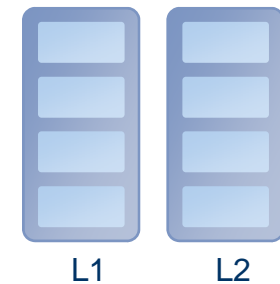
 write A1' to L1
 write A2' to L1
 write A3' to L1
 write A4' to L1
 write B1' to L2
 write B2' to L2
 Merge A and L1
(switch)
 write C1' to L1

BAST FTL
 Operation

Data block



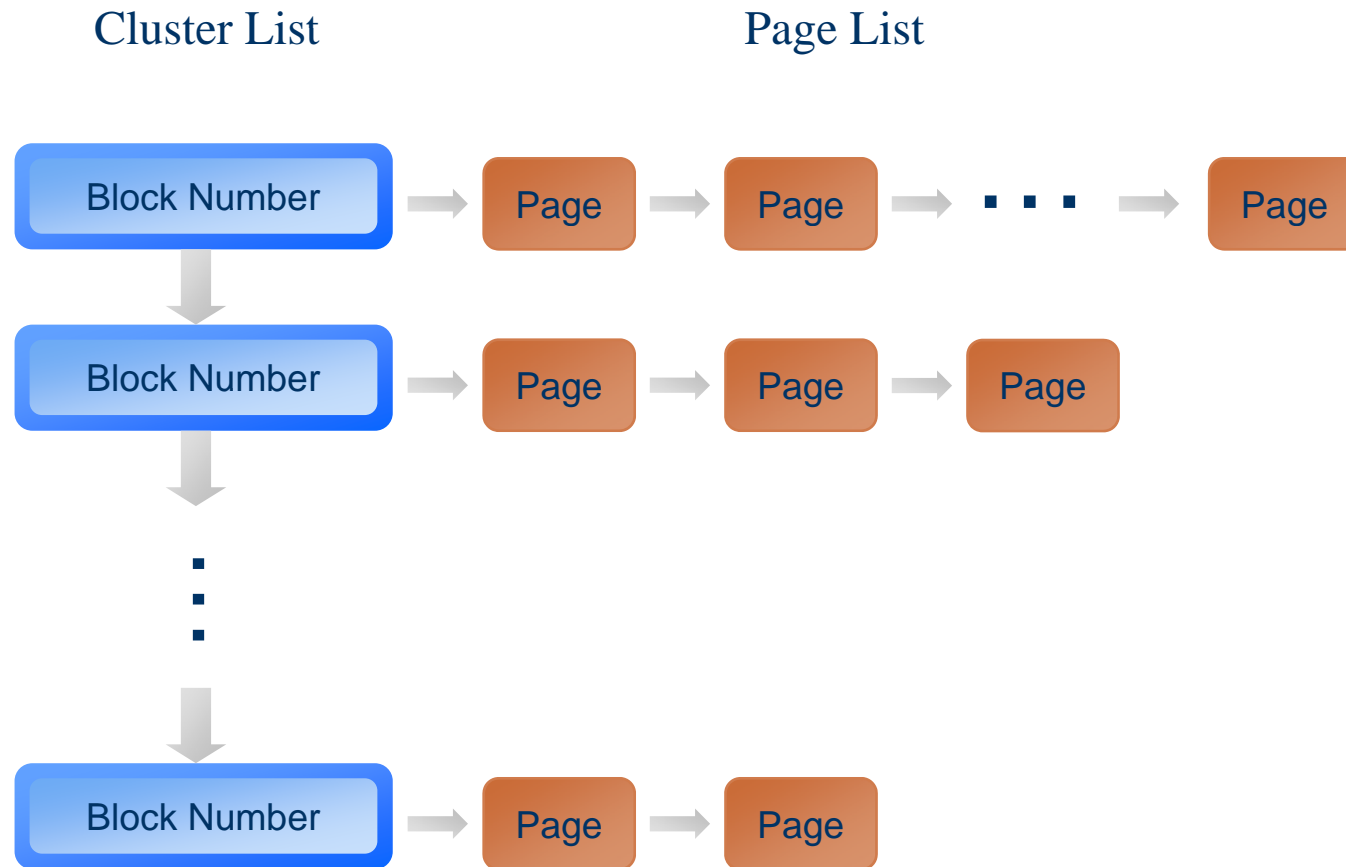
Log block



Flash Memory



Data Structure of Page Cluster

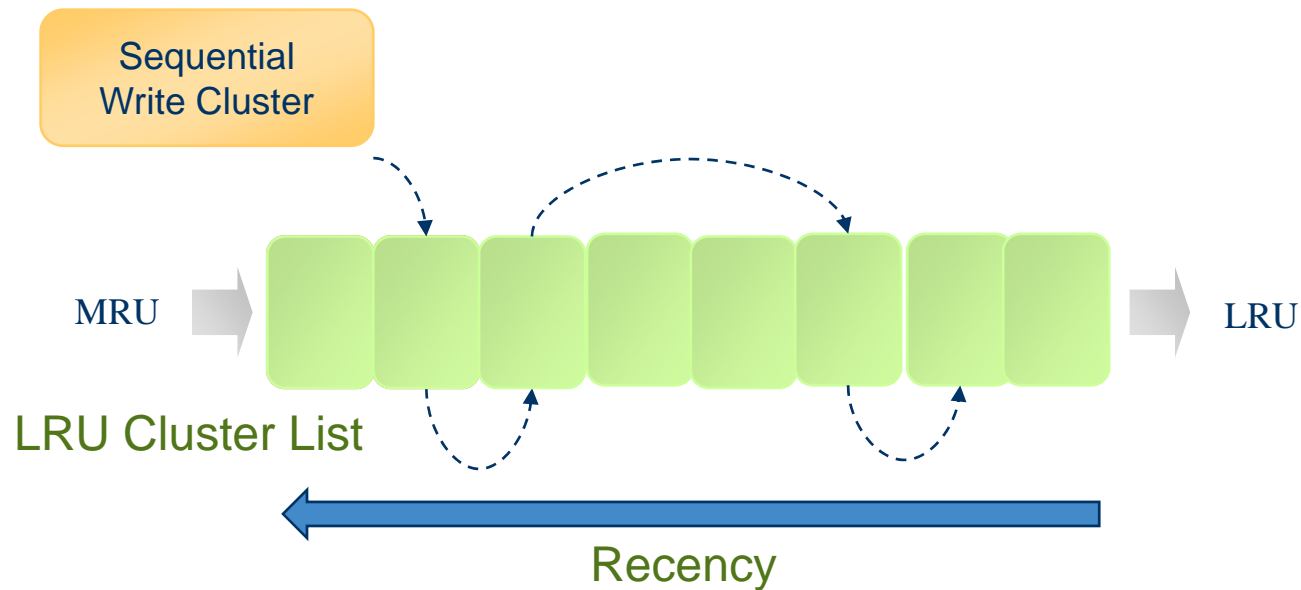




Cluster-based Algorithm 1

❖ LRU-C

- Least Recently Used (Written) Cluster first
- 클러스터의 recency를 고려하여 교체 클러스터를 선정
- Sequential write cluster list를 유지하여 우선적으로 교체대상으로 삼음



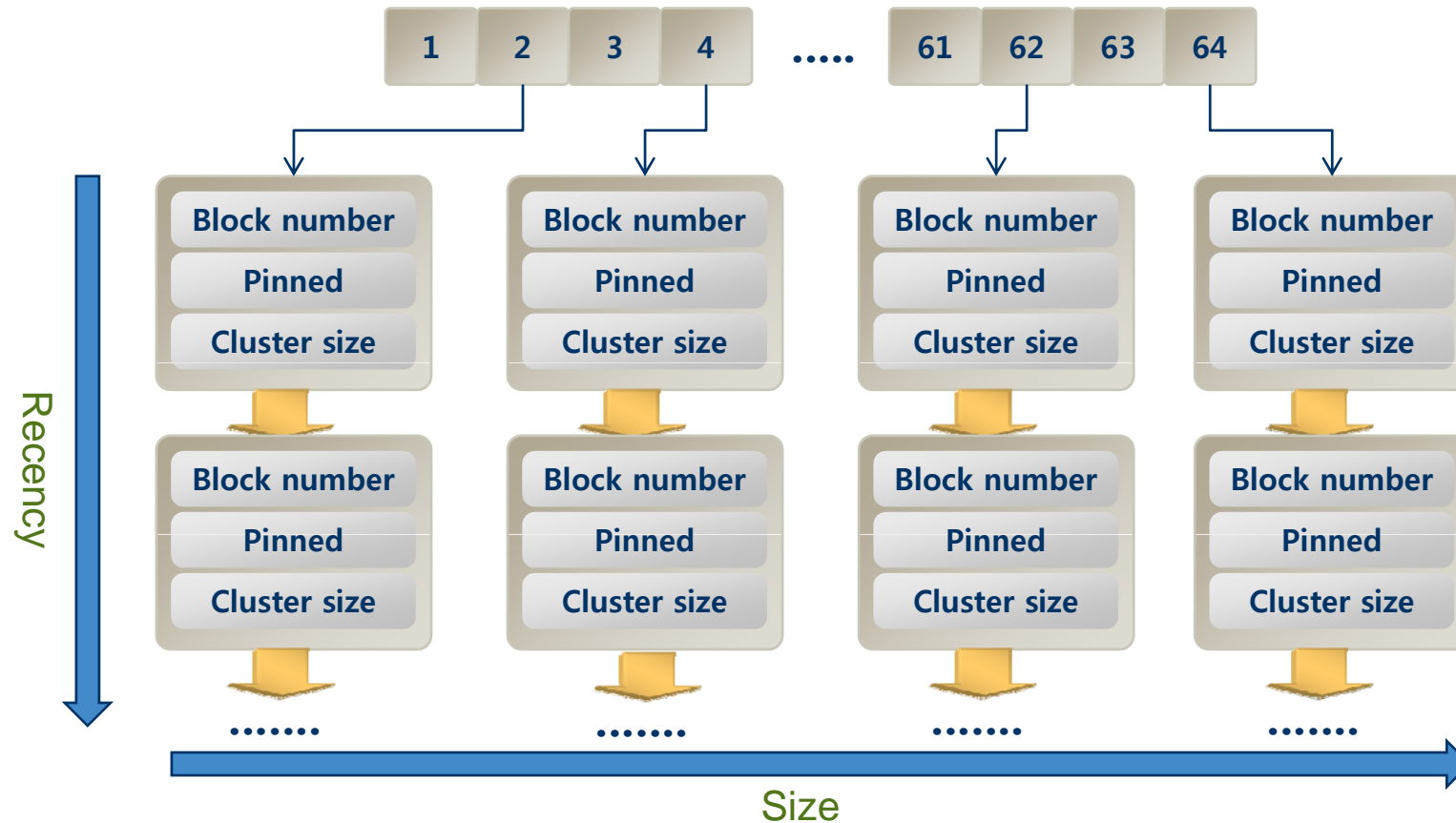
Cluster-based Algorithm 2



❖ LC

- Largest Cluster first
- 클러스터의 크기만을 고려하여 교체 클러스터 선정

LRU Cluster Pointer Array

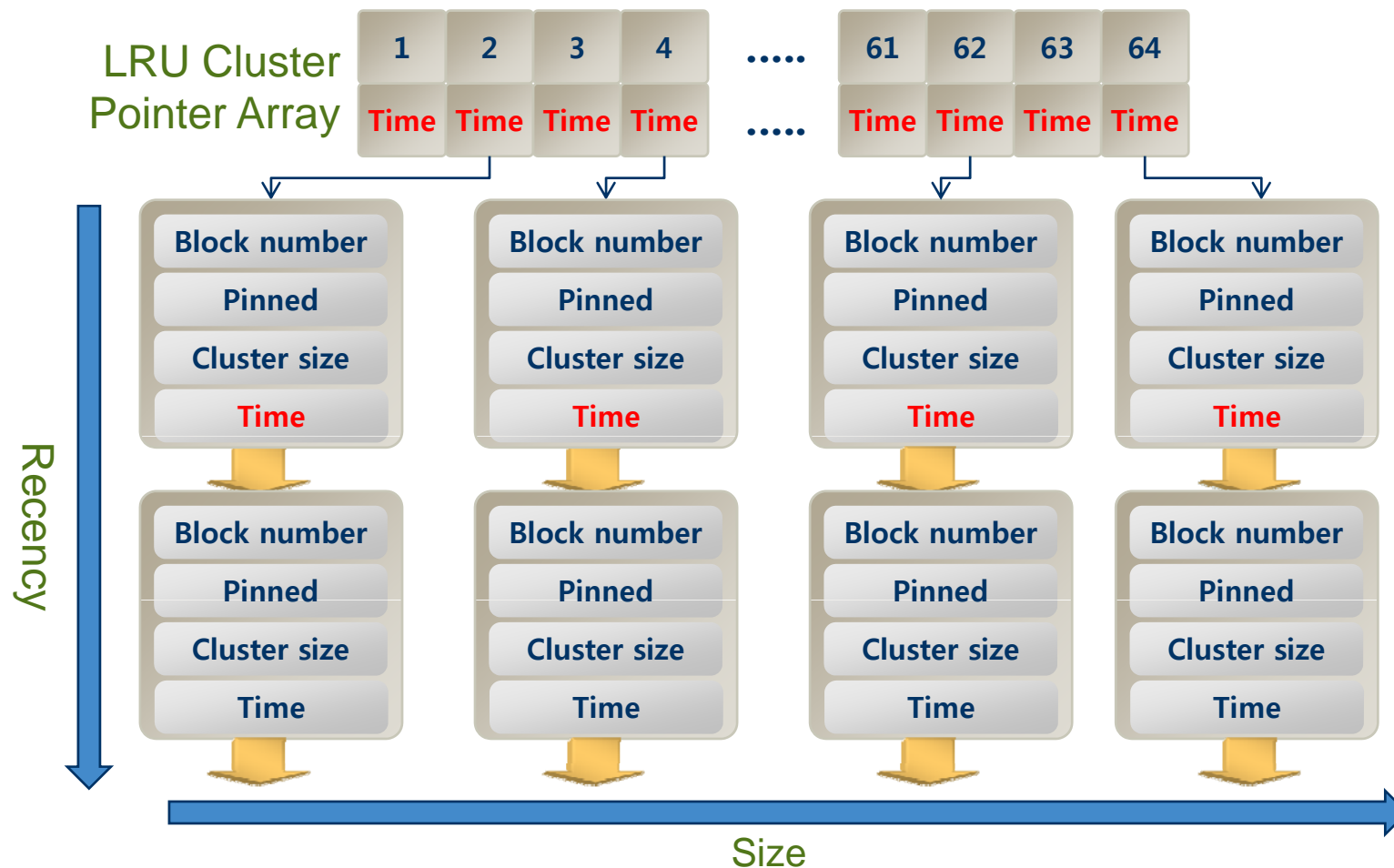


Cluster-based Algorithm 3



❖ Greedy-Dual

- Recency와 cluster size를 동시에 고려
- $\text{Time} * \text{cluster size}$ 의 값이 큰 클러스터를 교체 대상으로 선정





Is there any off-line optimal algorithm?



Off-line Optimal Algorithm

❖ We can know

- When a page (or cluster) will be referenced again in the future

❖ However, it is hard to know

- How many extra operations will occur **in the future** by replacing this page (or cluster) **now**

❖ Possibility of finding an Optimal algorithm

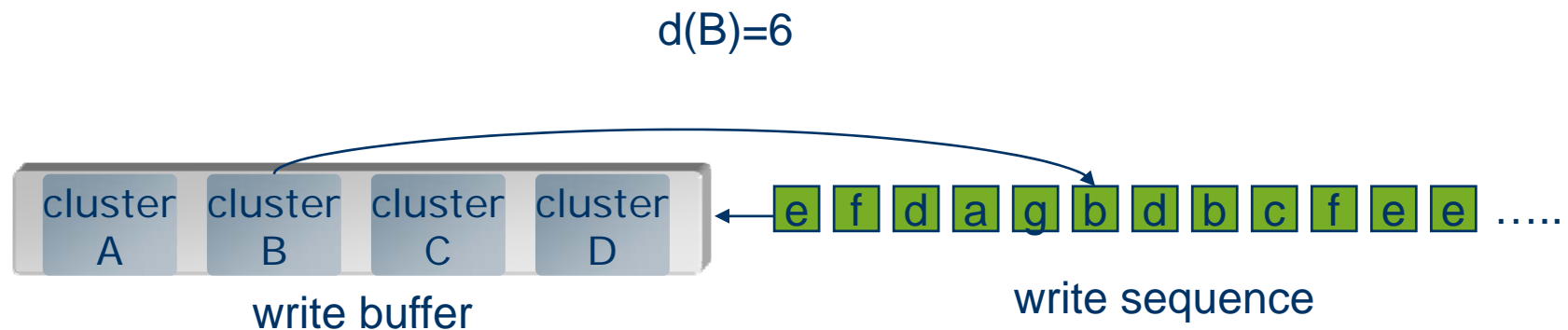
- Very, very hard
- Maybe intractable!



Off-line Pseudo Optimal Algorithms

❖ MIN-C (MIN Cluster)

- 기존의 MIN 알고리즘을 클러스터에 적용
 - 미래에 가장 나중에 참조될 클러스터를 교체대상으로 선정
 - 다음에 참조될 거리 값(d)을 이용



$d(A)=4, d(B)=6, d(C)=9, d(D)=7$

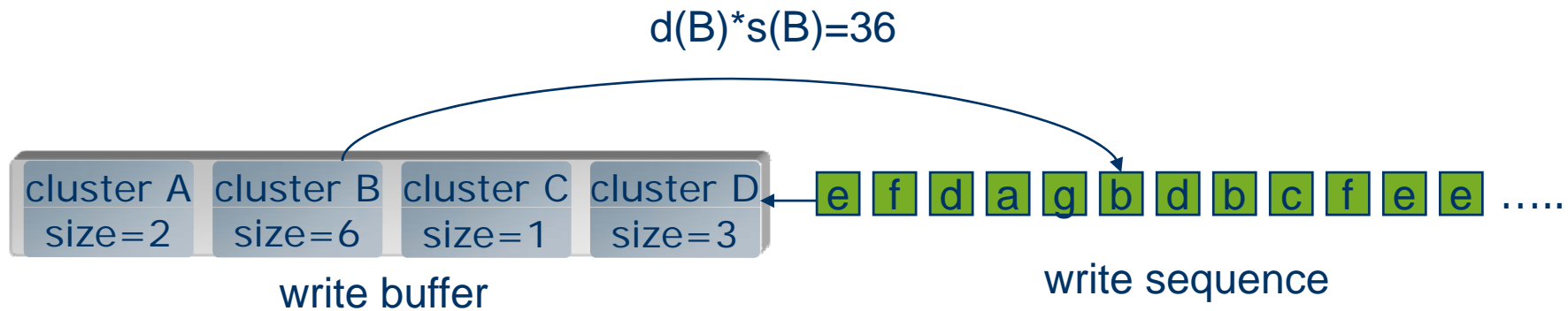
victim cluster = C



Off-line Pseudo Optimal Algorithms

❖ MIN-CS

- MIN-C 정책에 Cluster size도 고려
- $d(x)*s(x)$

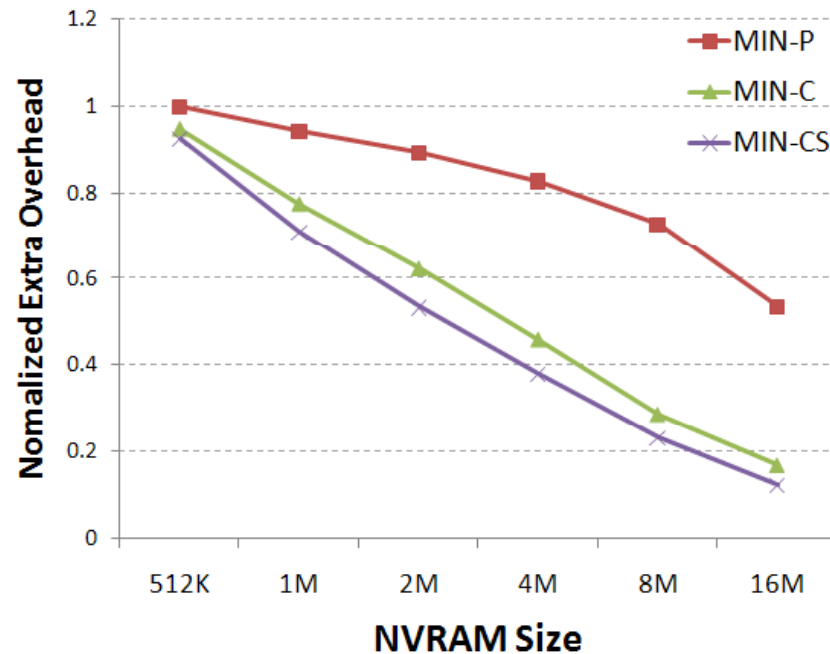


$$d(A)*s(A)=8, d(B)*s(B)=36, d(C)*s(C)=9, d(D)*s(D)=21$$

victim cluster = B



Off-Line Pseudo Optimal Algorithms



❖ Careful conjecture

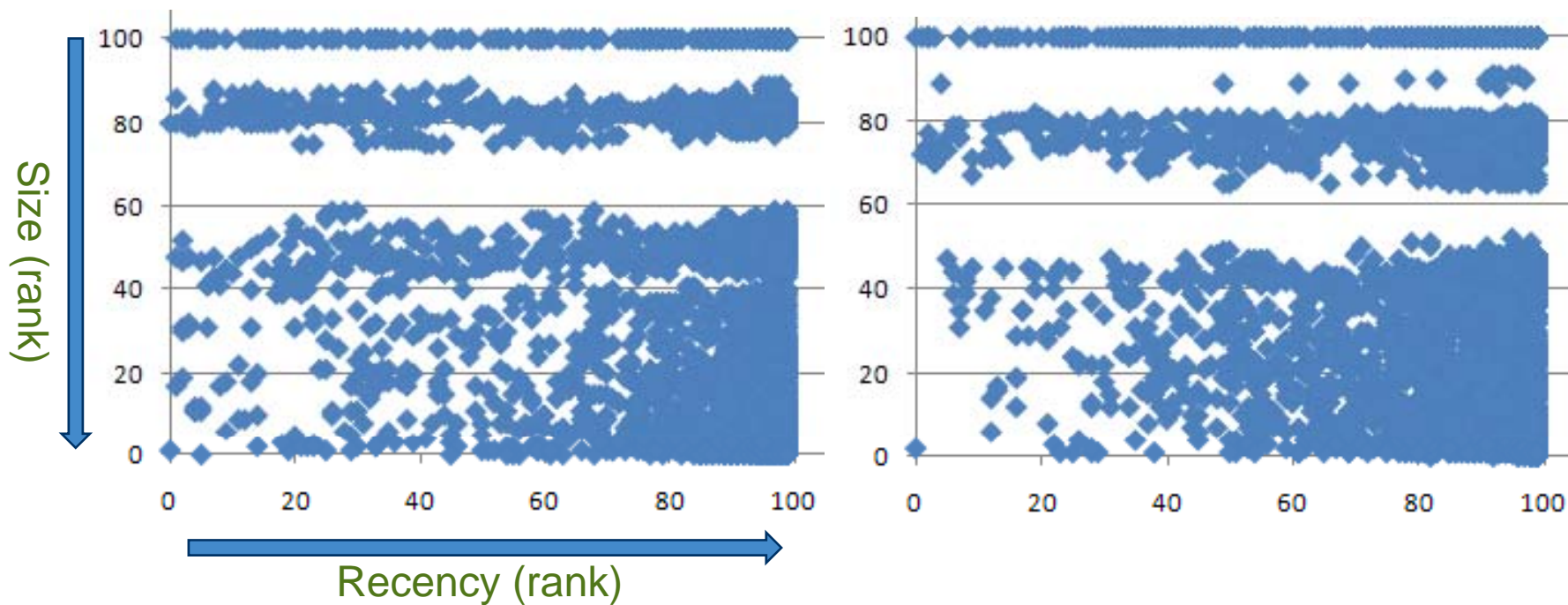
- Optimal algorithm
 - Would be cluster-based
 - Would consider the cluster size

❖ Metric

- Which one is more important metric between 'Recency' and 'Cluster size'?
- Maybe we can get hint from the behavior of MIN-CS



"Recency" versus "Size" in MIN-CS

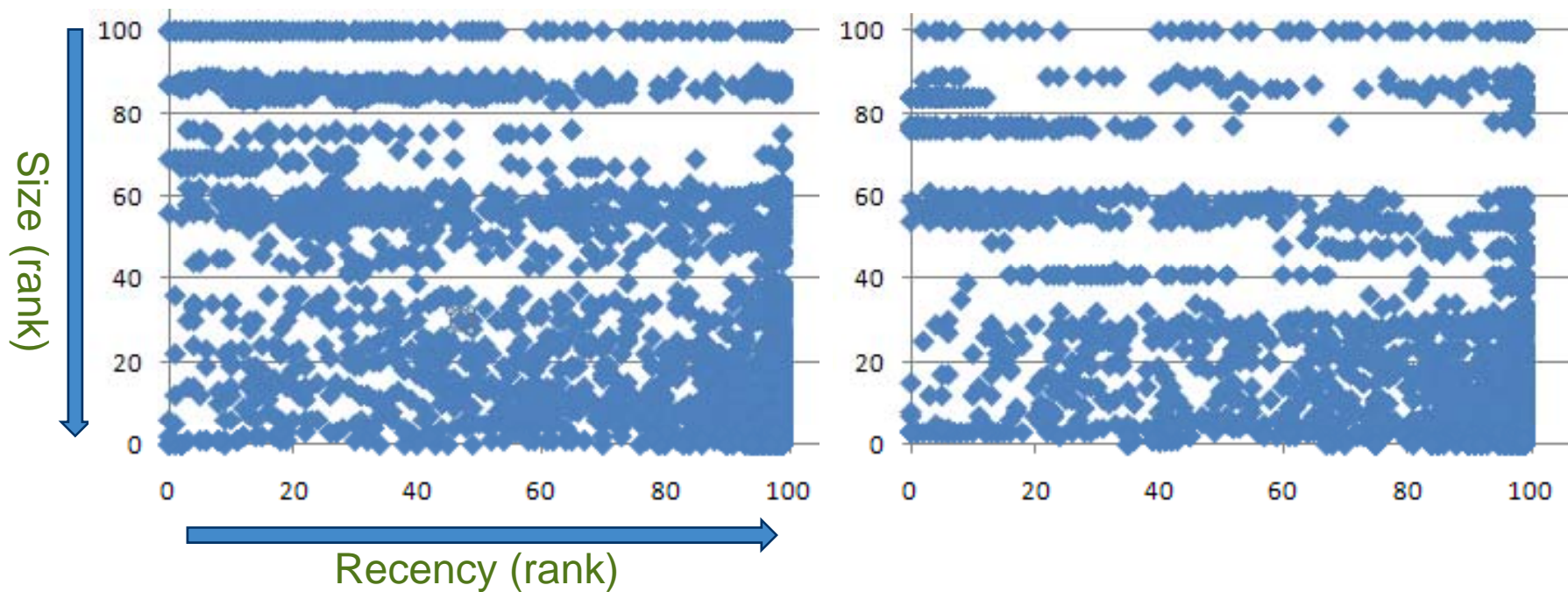


WB size= 0.5M

WB size= 2M



"Recency" versus "Size" in MIN-CS



WB size= 8M

WB size= 16M



Performance ?



Workload

❖ Sequential and cache insensitive

Desktop Trace

Attribute	Value
Filesystem	FAT32
Running Application	Web surfing, emails sending / receiving, document typesetting, and gaming, multimedia file download
Duration	One month
Locality	75 of total requests access 41 of total page's
Total page written	14,147,956

Attribute	Value
Filesystem	NTFS
Running Application	Web surfing, emails sending / receiving, document typesetting, and gaming, multimedia file download
Duration	One month
Locality	75 of total requests access 23 of total page's
Total page written	58,545,851

Workload



❖ Random and cache sensitive

Database Trace

Attribute	Value
Filesystem	EXT2
Running Application	PostgreSQL
Duration	One Week
Locality	75 of total requests access 6 of total page's
Total page written	16,659,675

Simulation



❖ Underlying FTL: BAST

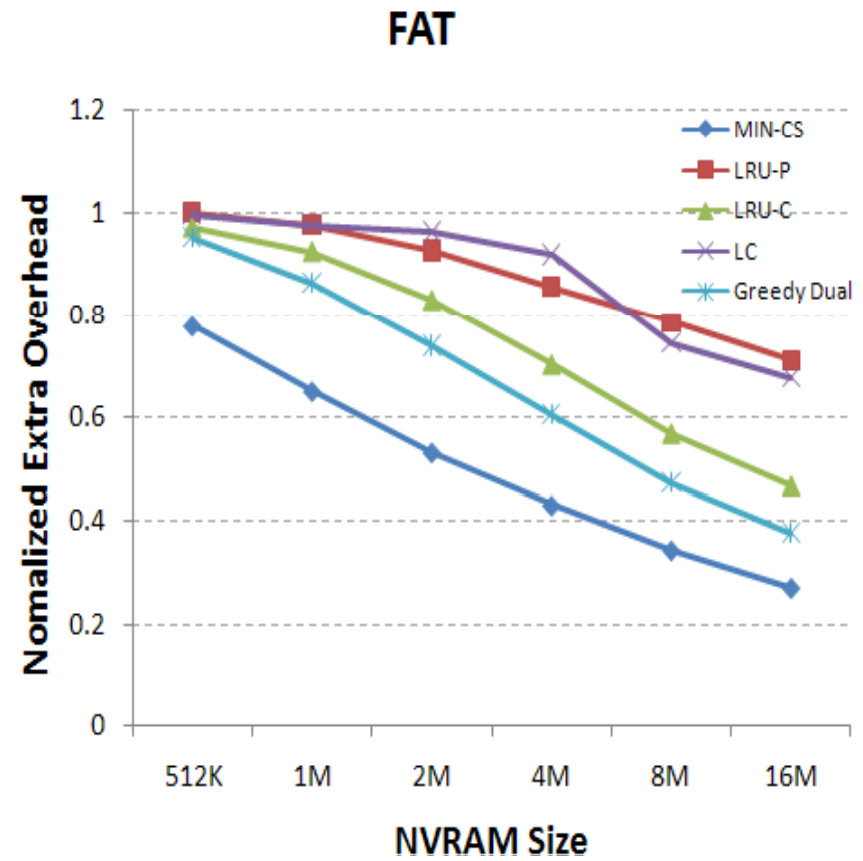
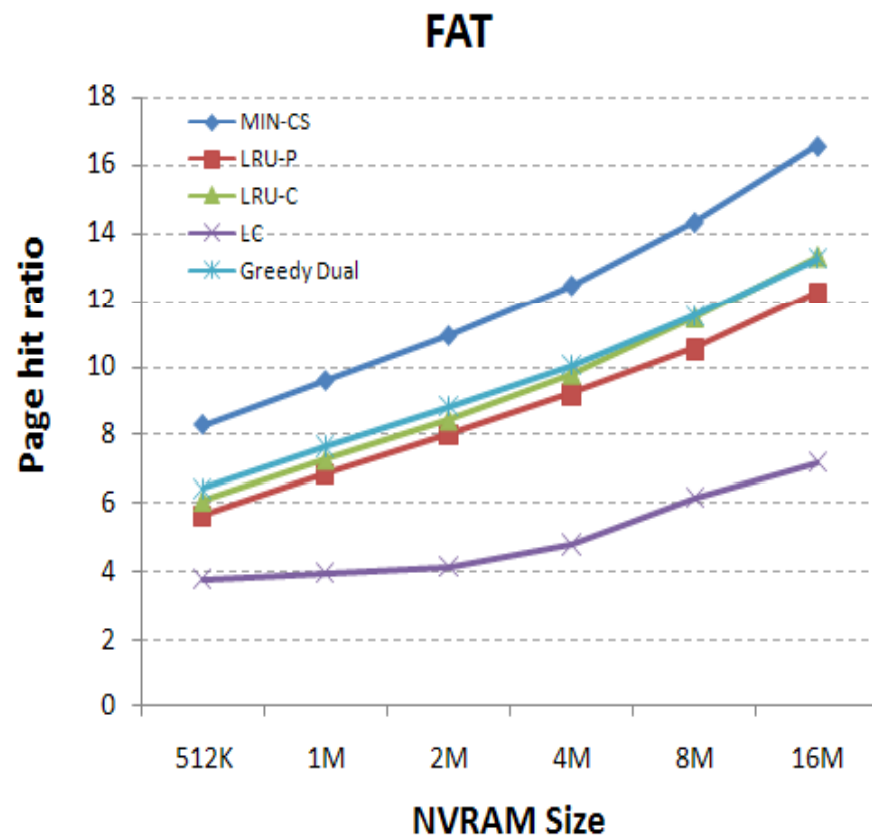
❖ Metrics

- Write Buffer page hit ratio
- Extra Overhead for Extra Operations
 - Erase, valid page copy

Results



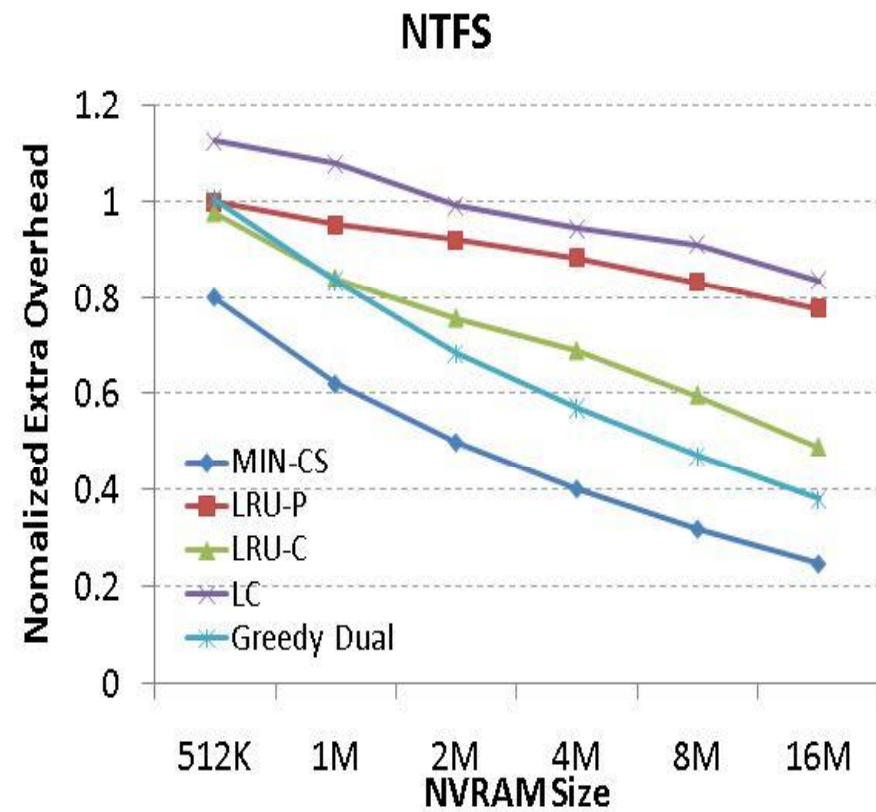
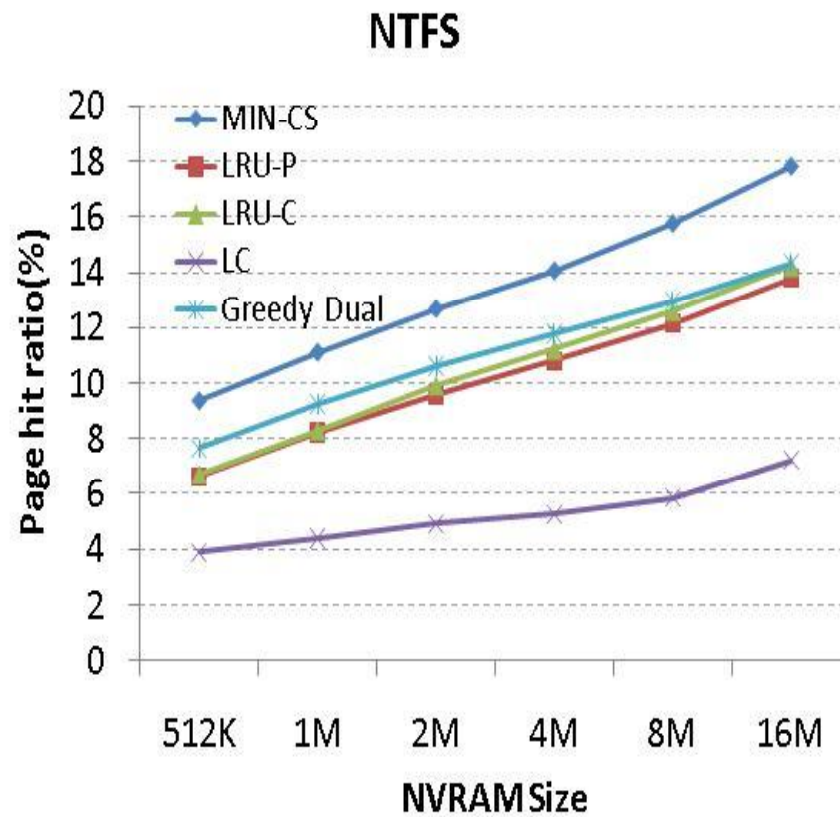
❖ FAT Trace



Results



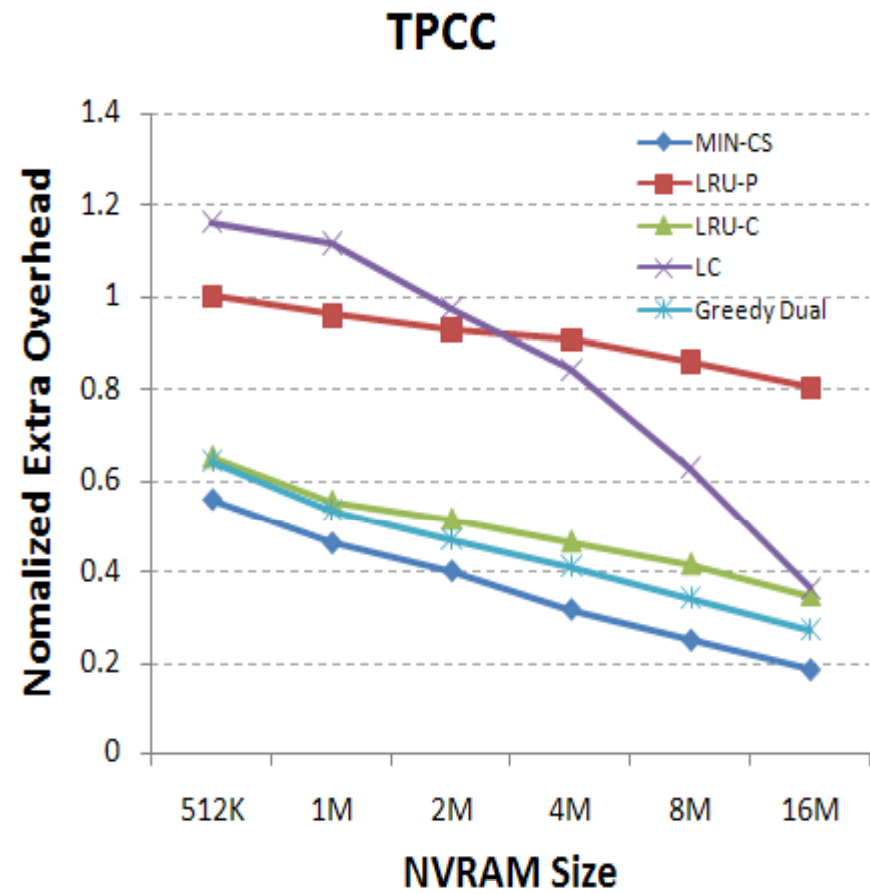
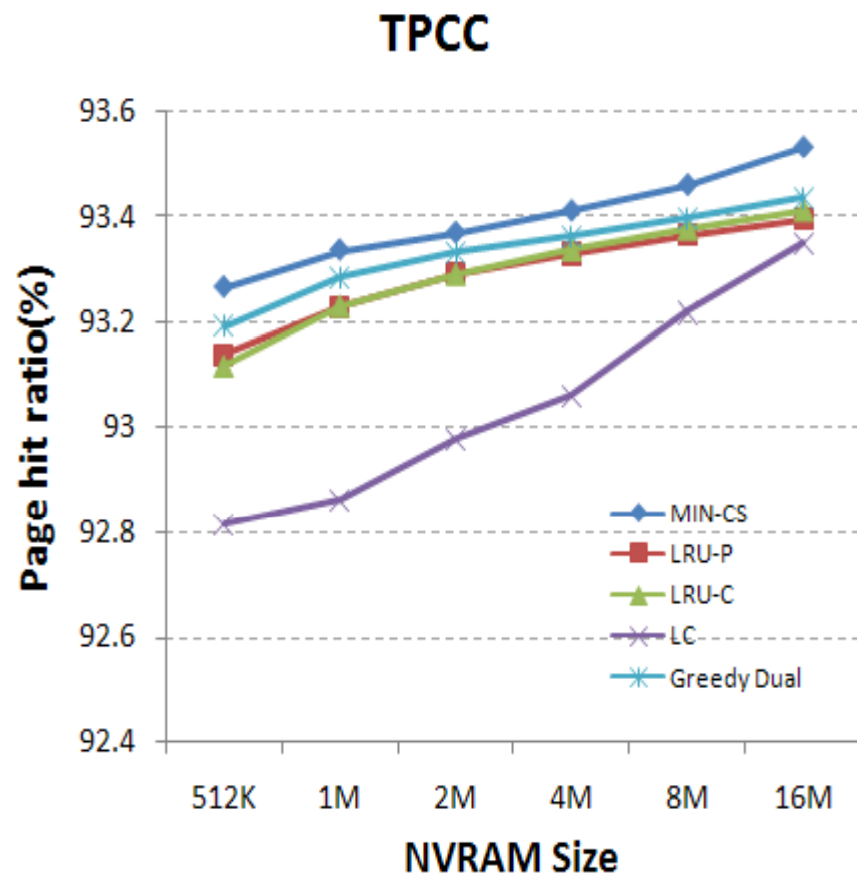
❖ NTFS trace



Results



❖ TPCC trace





Write Buffer-Aware FTL ?



Observations and Insight

❖ Observations

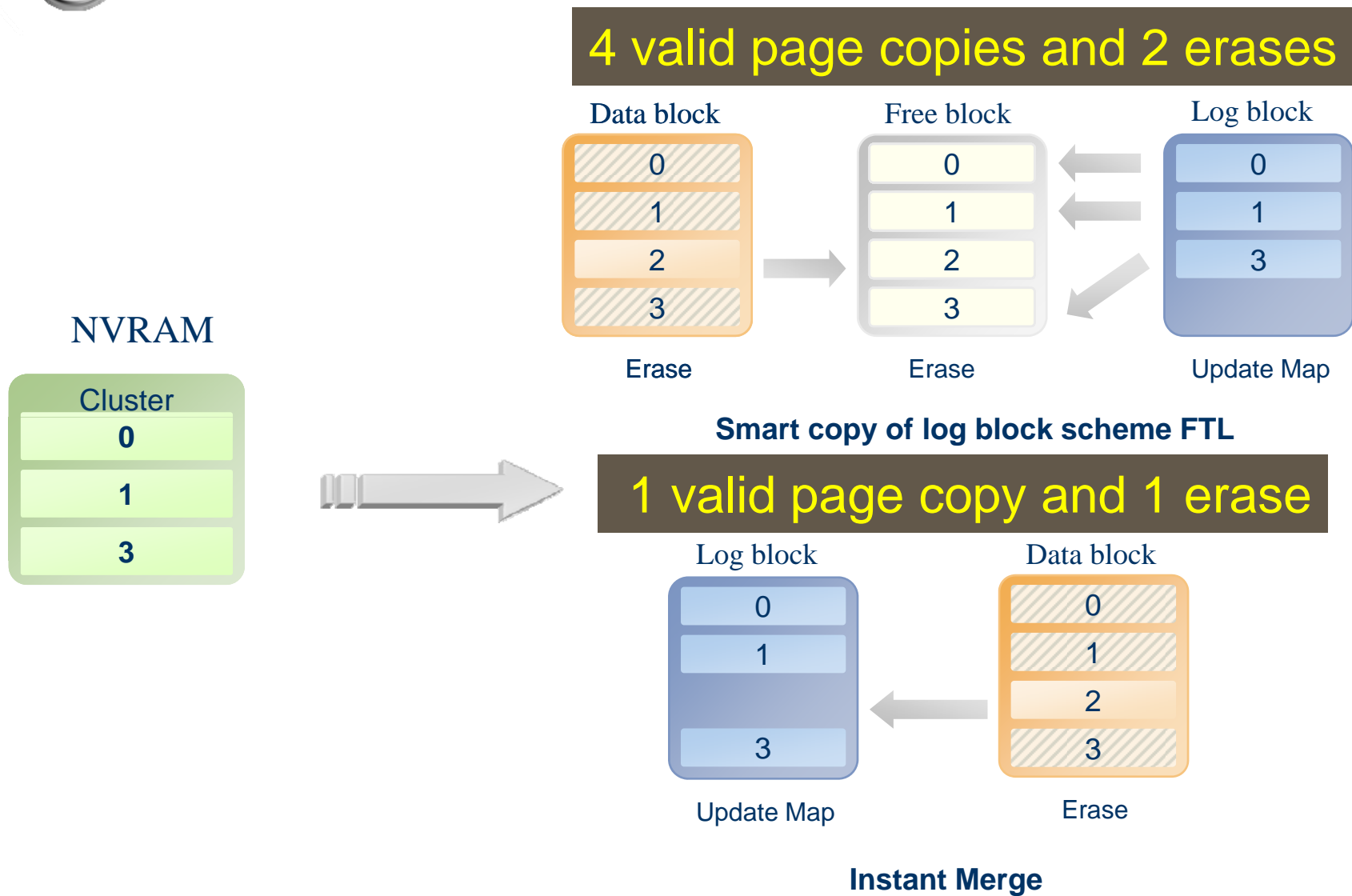
- 클러스터 단위의 Flushing
 - Random write의 성질이 희석됨
 - FTL에서 Log block의 필요성이 낮아짐
- 로그블록 기반 FTL의 문제점
 - 로그 블록에 쓴 이후에 데이터 블록과 병합하는 과정에서 smart copy로 인한 오버헤드가 큼

❖ Insight

- Flush되는 클러스터와 플래시 메모리의 데이터 블록을 즉시 병합 → Instant Merge operation



Smart Copy versus Instant Merge





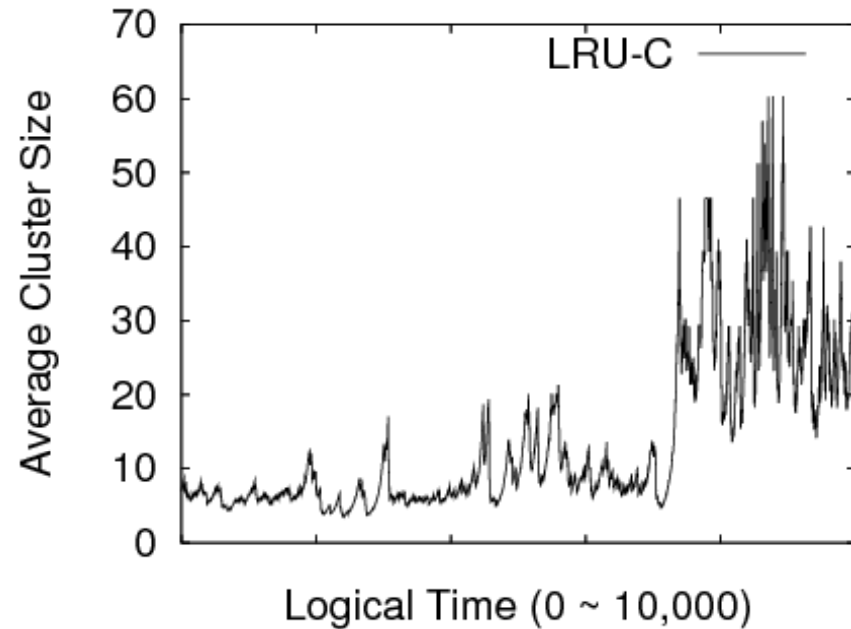
Optimistic FTL

❖ Be OPTIMIST!

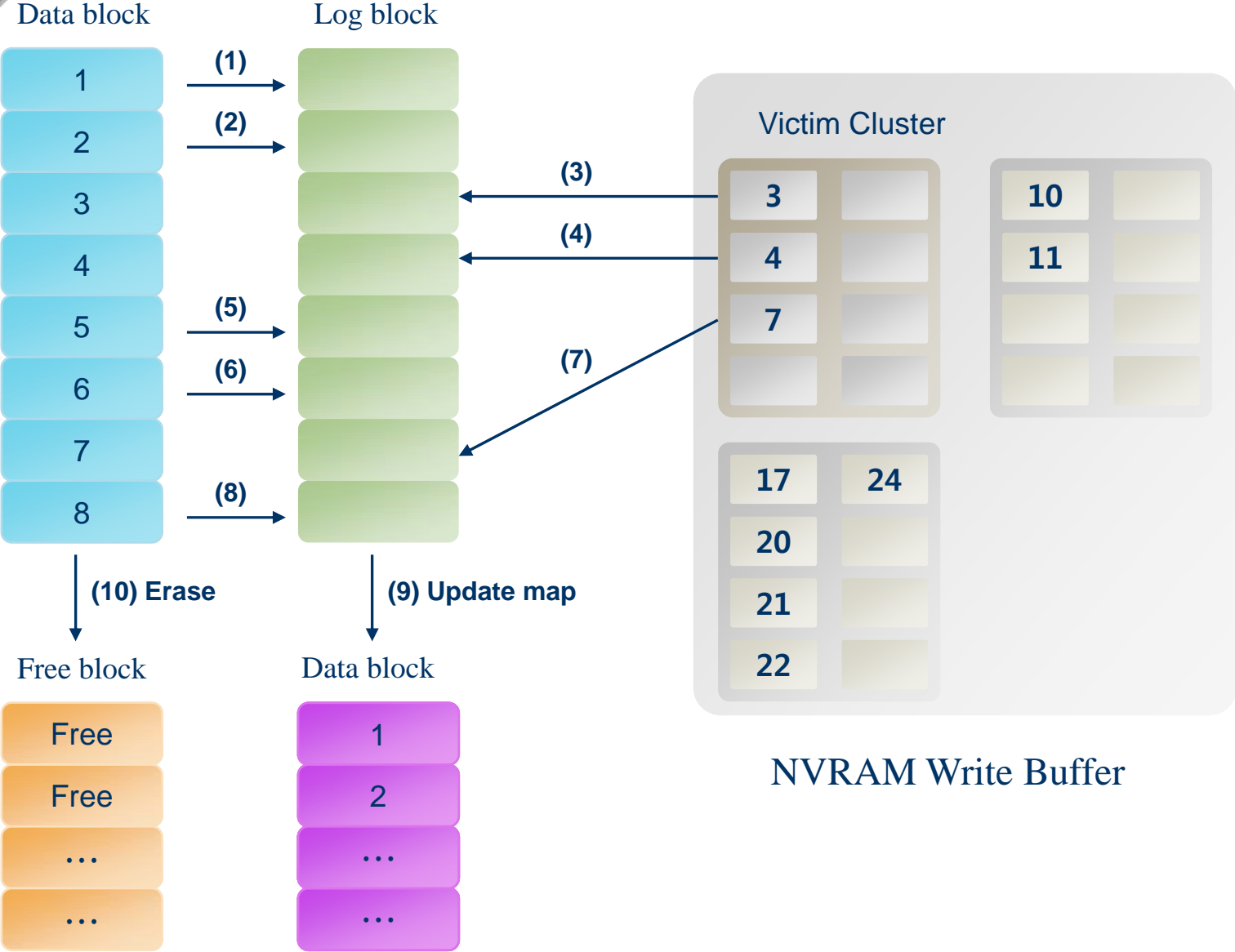
“적절한 크기의 Write Buffer
가 있고, 클러스터 단위의
버퍼교체정책을 사용할
경우, Flush 되는
클러스터는 충분히 많은
페이지를 포함할 것이다”

❖ And

- Use Instant Merge
 - No Log block except one for IM
- Use only block mapping (No page mapping)



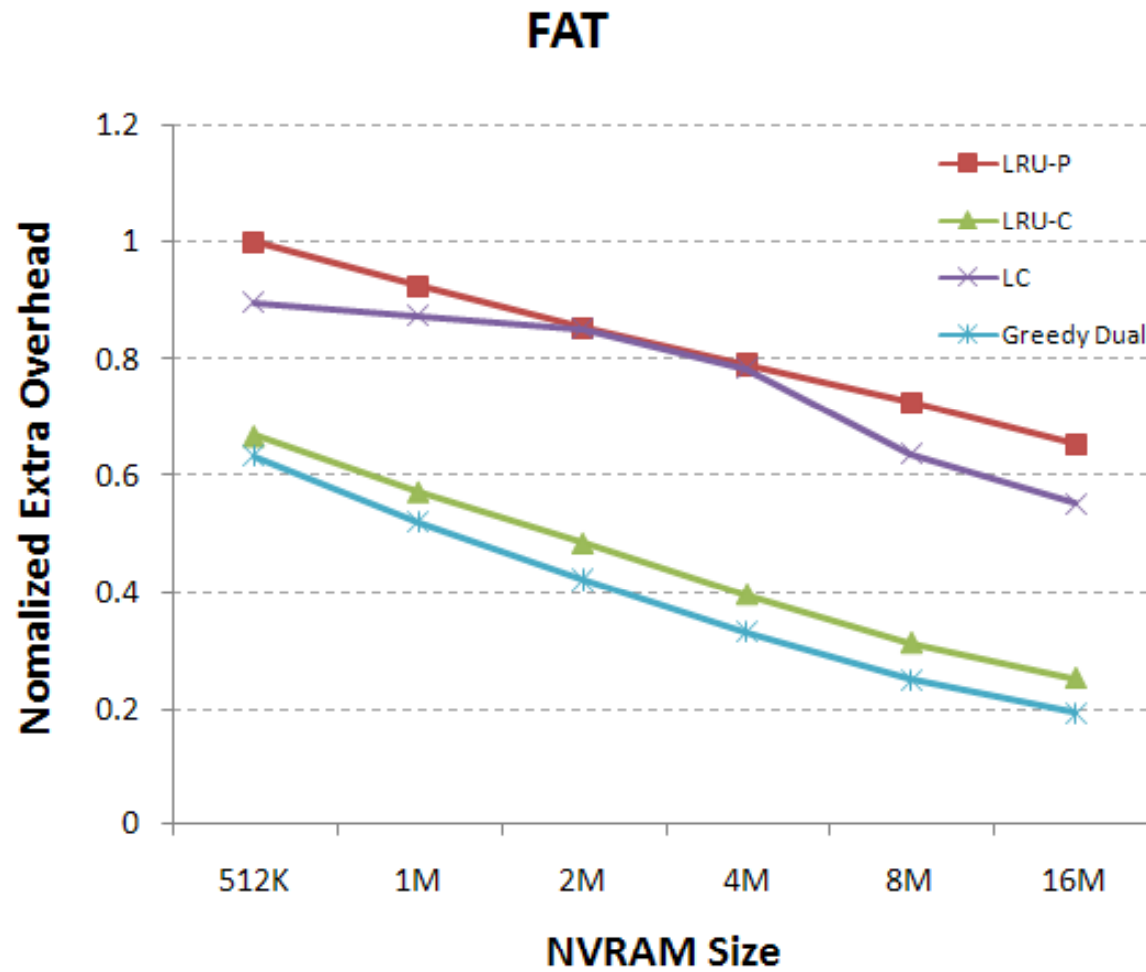
Optimistic FTL



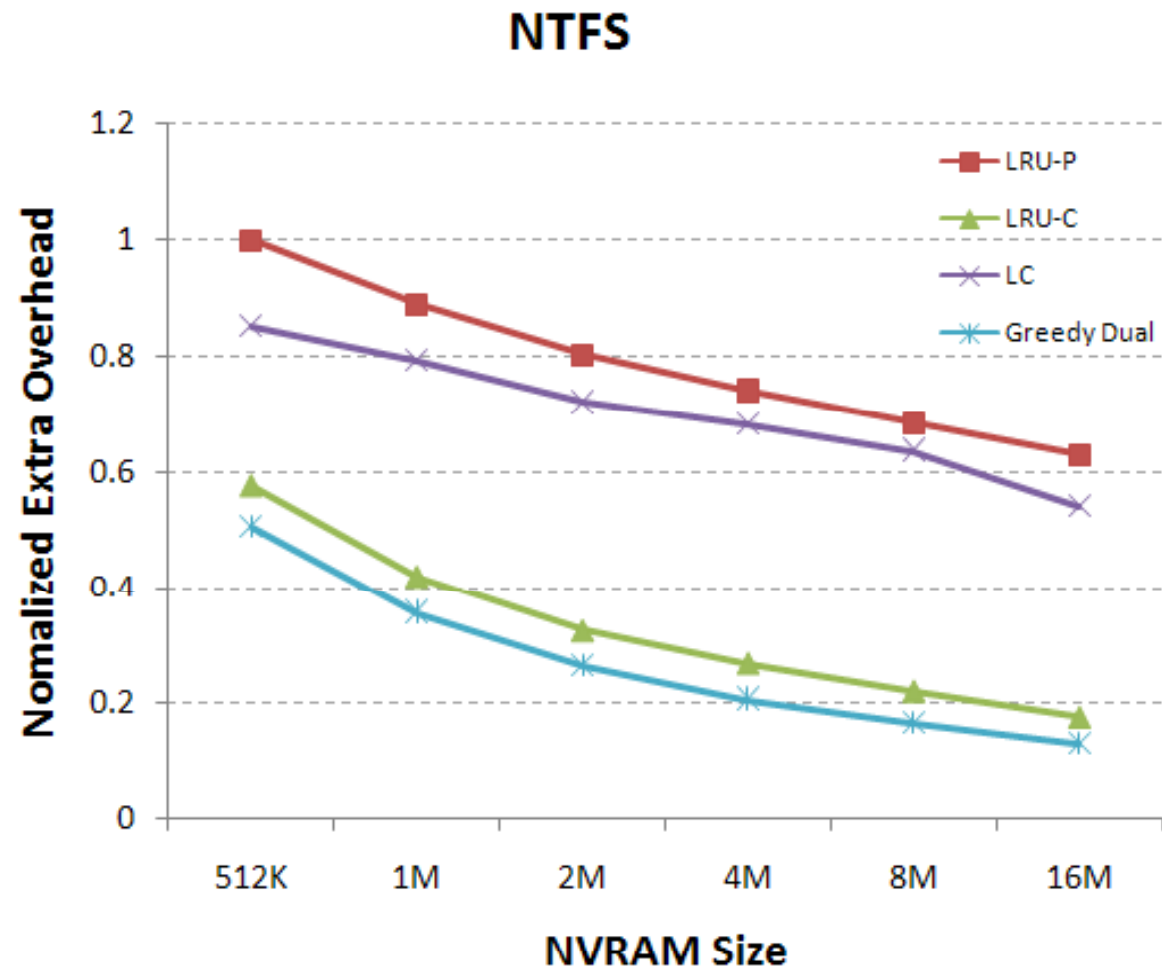


Is it OK to be an Optimist ?

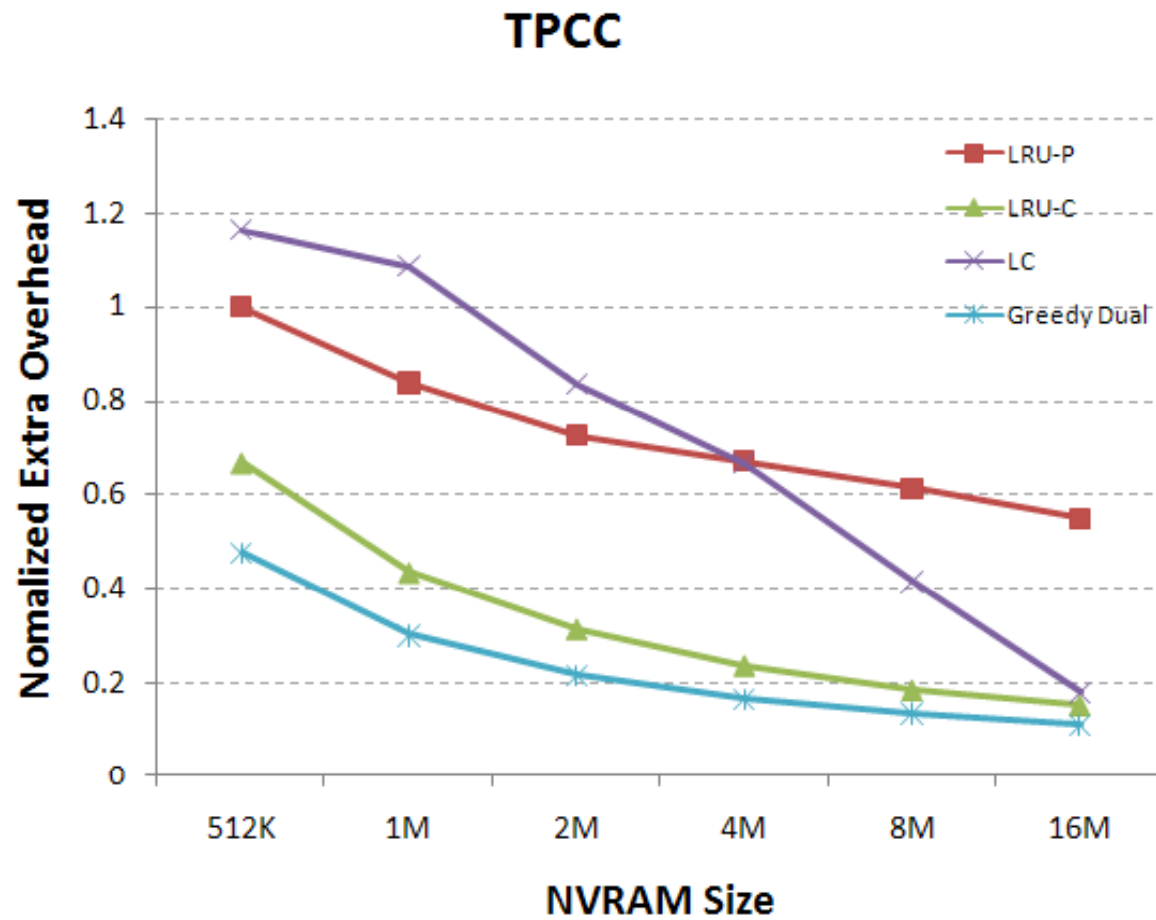
Simulation Result



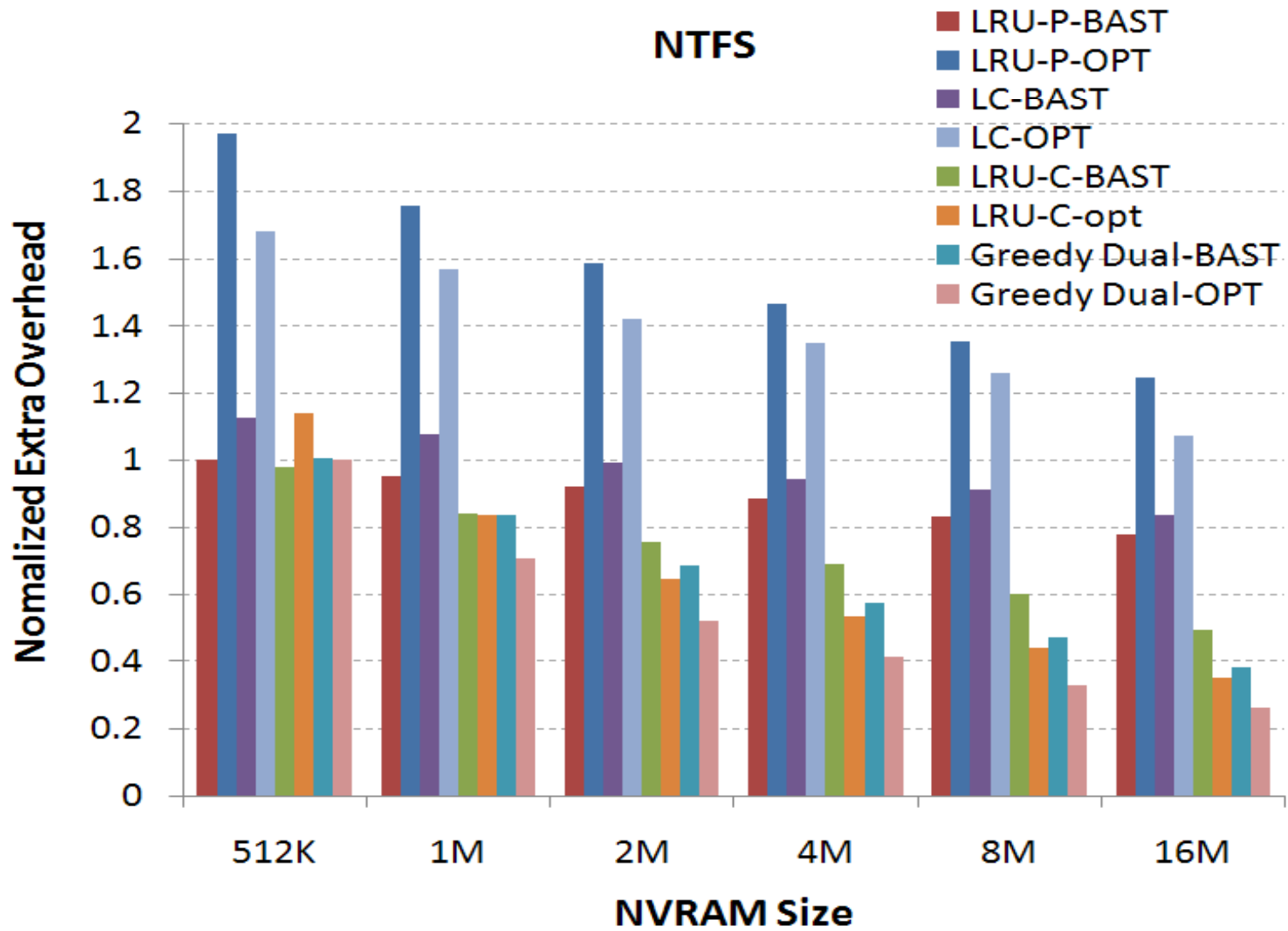
Simulation Result



Simulation Result



BAST vs Optimistic FTL





Conclusion

❖ SSD를 위한 쓰기버퍼 관리

- Page Clustering 필요
- 클러스터 기반 교체정책 필요
 - Hit ratio와 Extra overhead 동시 고려
 - Ex. Greedy-Dual
 - 최적 알고리즘에 대한 고찰 필요
- WB-aware FTL 개발 필요
 - Ex. Optimistic FTL