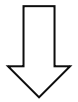
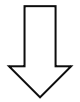
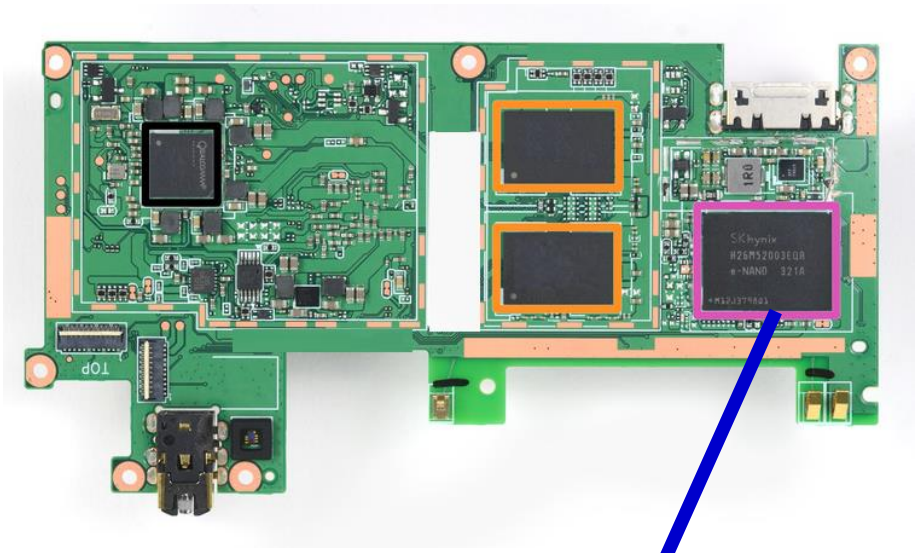
Map Cache Design in Mobile Storage

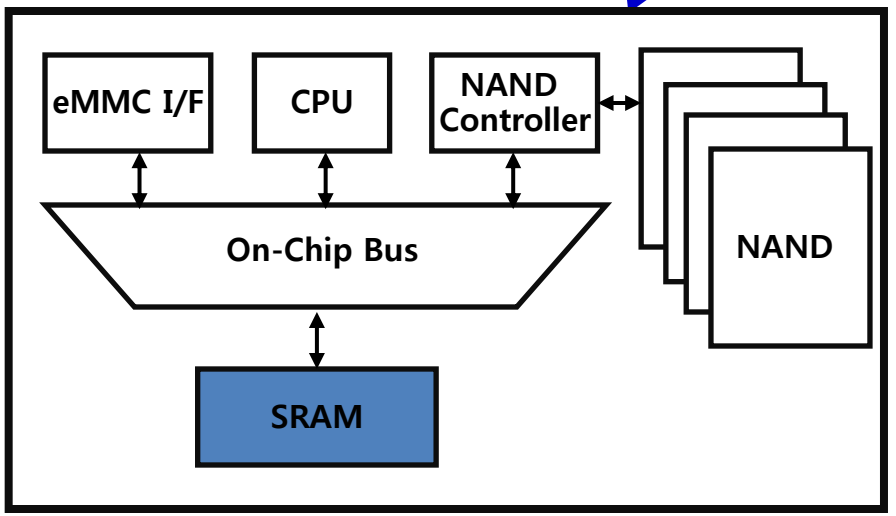
Requirements for Mobile Storage



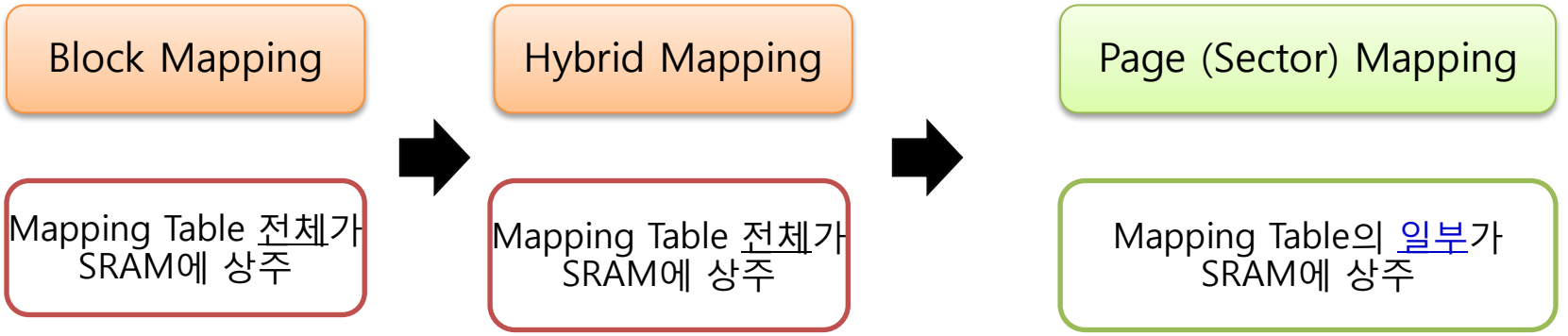


Large SRAM	Small SRAM
Fine grain Address translation	Coarse grain Address translation



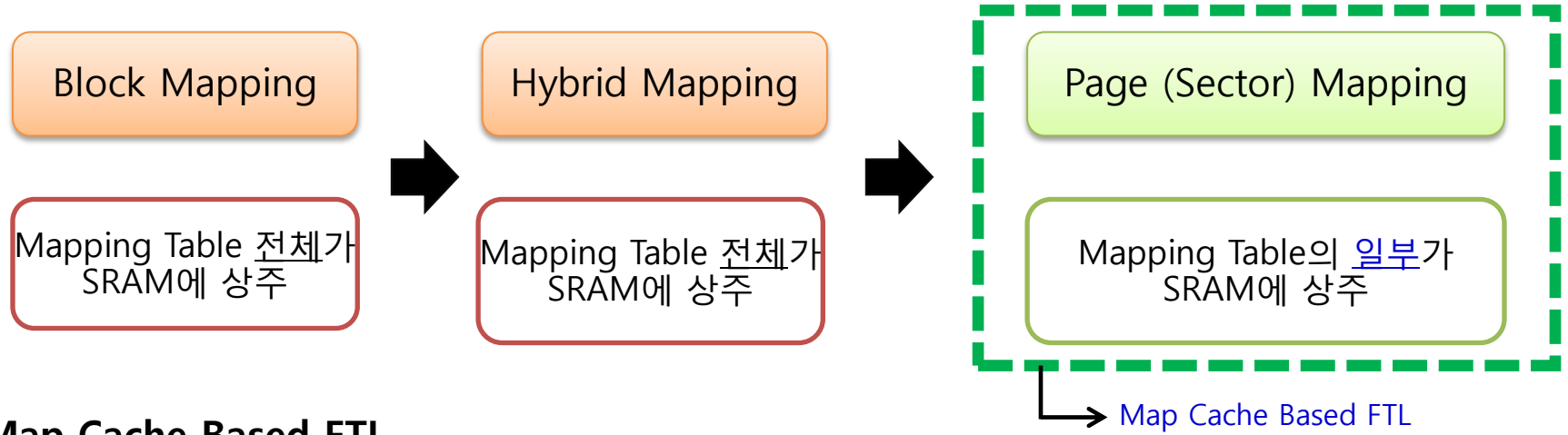


Evolution of FTL for Mobile Storage



- **Block Mapping**
  - Block 단위의 mapping. Logical block 이 Physical block으로 mapping.
  - 장점 : Map table size가 작아서, 작은 크기의 SRAM이 가능.
  - 단점 : Random access에서 성능 저하.
- **Hybrid Mapping**
  - Log block에는 block mapping을 data block에는 page mapping 을 사용.
  - 장점 : Block mapping 보다 우수한 성능. Page mapping보다 더 작은 mapping table size.
  - 단점 : 여전히 random access pattern에서는 큰 garbage collection (merge) overhead 발생.
  - BAST (2002), FAST (2006), SAST (2007), LAST (2009), KAST (2009).

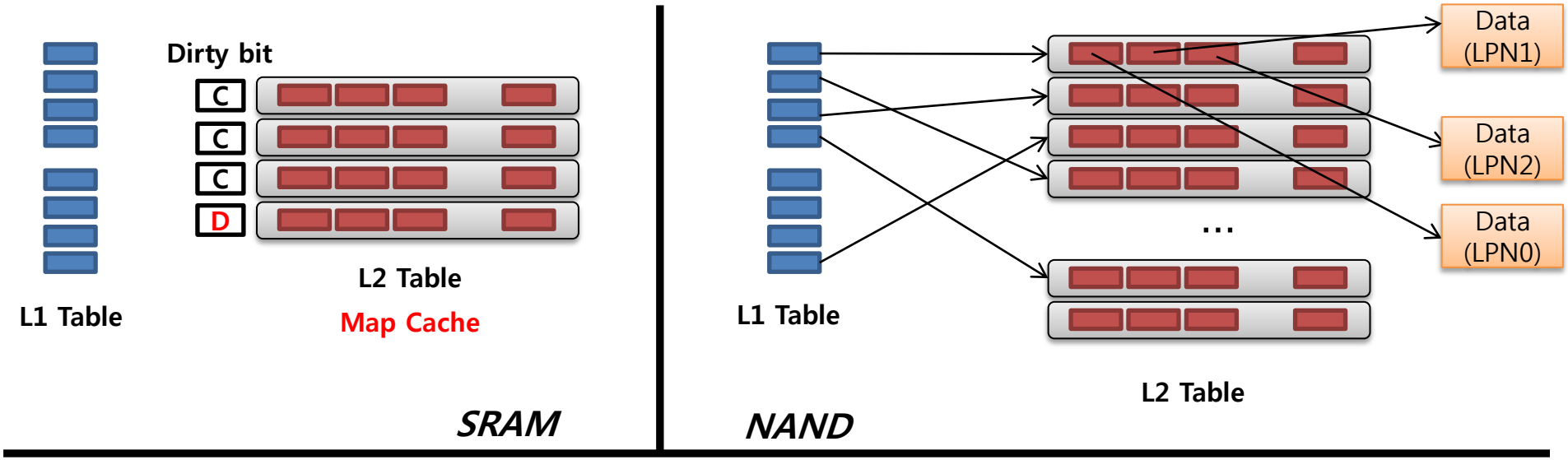
Evolution of FTL for Mobile Storage



- **Map Cache Based FTL**
  - 2009년 DFTL (Demand-based Flash Translation Layer) [1] 에서 처음 소개
  - Mapping 단위는 page mapping. 또는 sector mapping.
  - 기본 개념은 paging system에서의 TLB (Translation Lookaside Buffer)와 유사.
  - SRAM에 mapping table 중 일부만 (자주 access되는 entries) 상주시킴.
  - 일반적인 workload에 존재하는 spatial, temporal locality를 활용.
  - 대체적으로 hybrid FTL들 보다 더 뛰어난 성능.

[1] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: a flash translation layer employing demand-based selective caching of page-level address mapping," in Proc. ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Washington D.C., USA, pp. 229-240, March 2009.

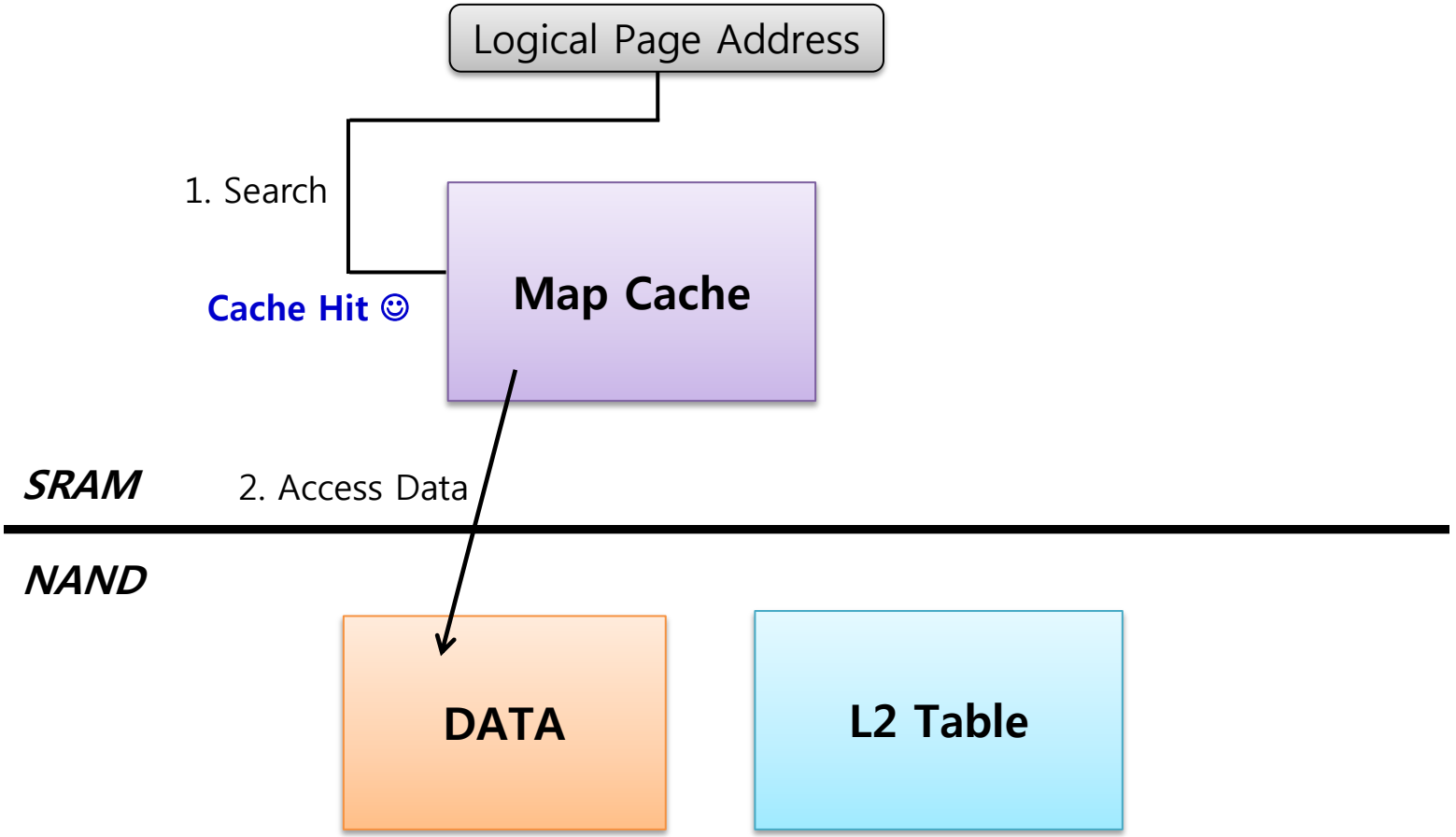
Typical Data Structure of Map Cache Based FTL



- **L2 Table : 각 logical address들의 physical address들을 담고 있는 table.**
  - 전체 L2 Table은 NAND에 저장되어 있고, 이중 일부가 SRAM의 Map Cache에 load됨.
- **L1 Table : L2 Table을 가리키는 Table.**
  - L2 Table의 update 역시 out-of-place update를 통해 이루이지기 때문에 valid한 L2 Table들의 위치를 가리키는 L1 Table이 필요함.
  - 일반적으로 L1 Table 은 크기가 작기 때문에, 전체가 SRAM에 상주.

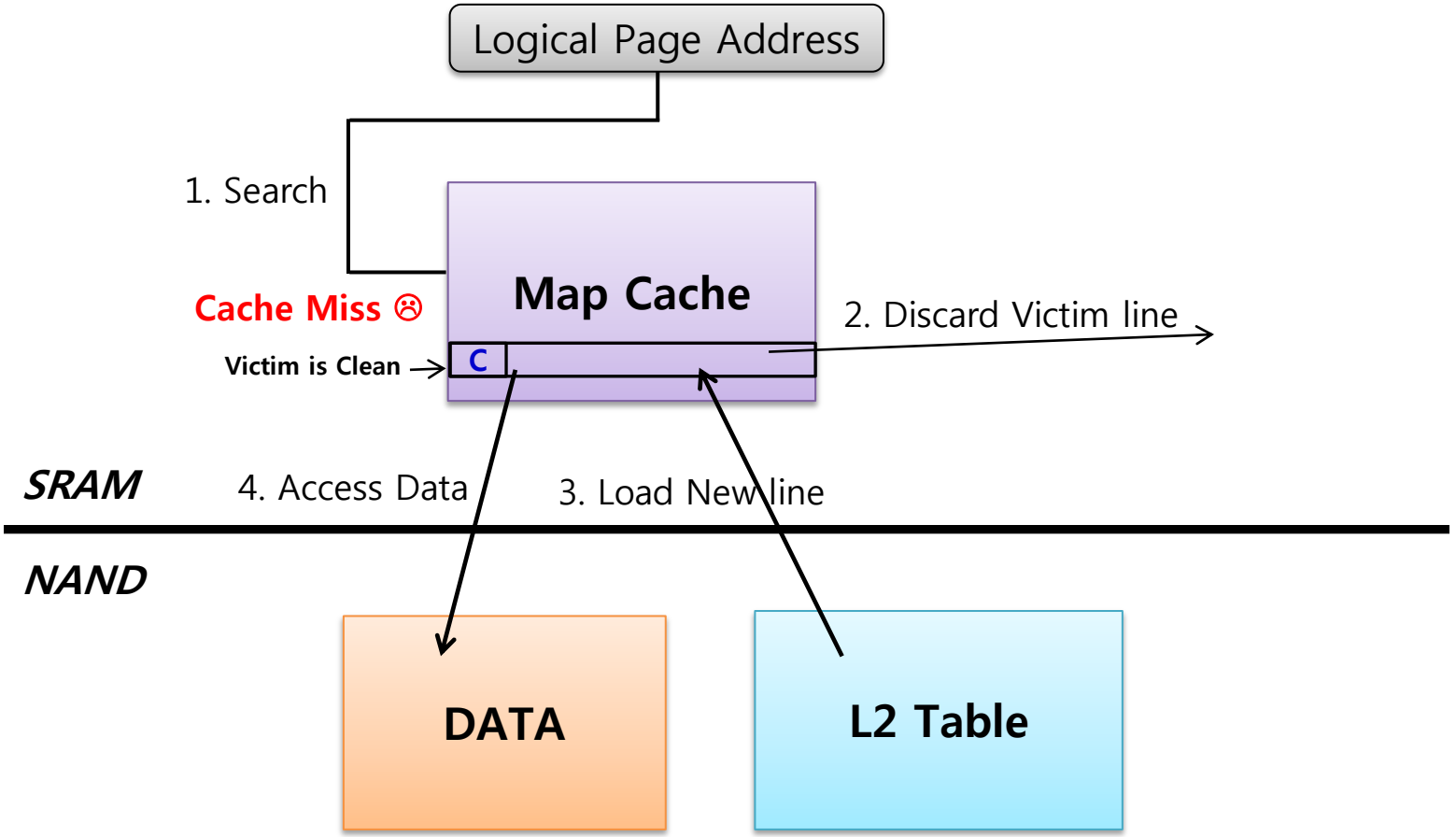
Typical Algorithm of Map Cache Based FTL (1/3)

- **Cache Hit Case**



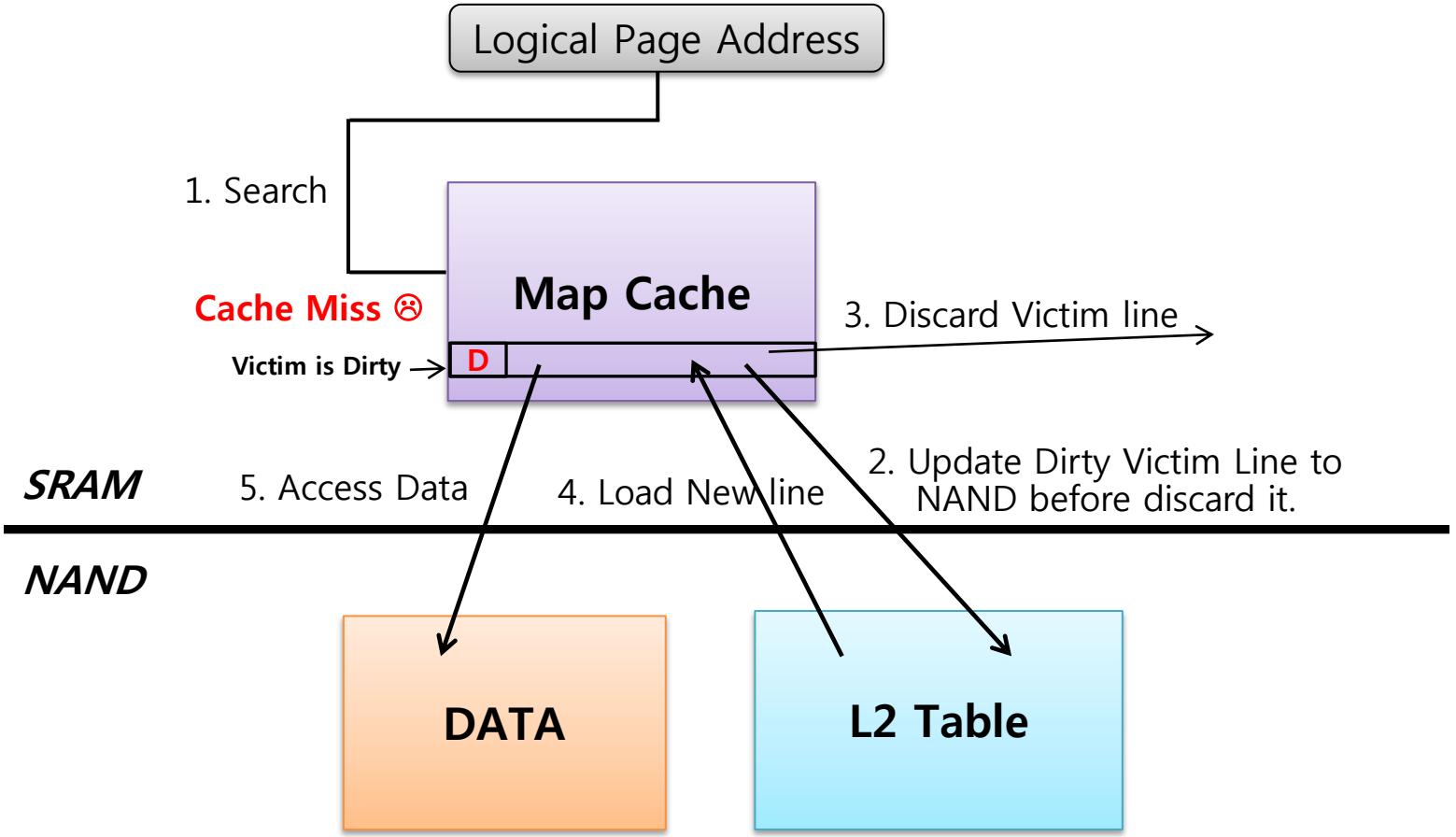
Typical Algorithm of Map Cache Based FTL (2/3)

- **Cache Miss Case (when victim is clean)**



Typical Algorithm of Map Cache Based FTL (3/3)

- **Cache Miss Case (when victim is dirty)**



Performance Parameters of Map Cache

- **Average Map Access Time 은 다음과 같은 식으로 표현 가능.**

  hit time + (1 - hit ratio) x (miss penalty)

  - hit ratio : Hit 횟수 / Access 횟수.
  - hit time : Map Cache를 access 하는데 걸리는 시간.

    (hit 인지, miss인지 판별하는데 소요되는 시간 포함)
  - miss penalty : Miss시에 해당 Map을 읽어오는데 걸리는 시간. *(NAND Read 발생)*

    ,Miss시 victim line이 dirty일 경우, victim을 NAND에 update하는데 걸리는 시간.

    (NAND Program 발생)

- **Average Map Access Time 을 최소화 하기 위해서는,**
  - Hit ratio를 올리고,
  - Hit time을 줄이고,
  - Miss penalty를 줄여야 함.

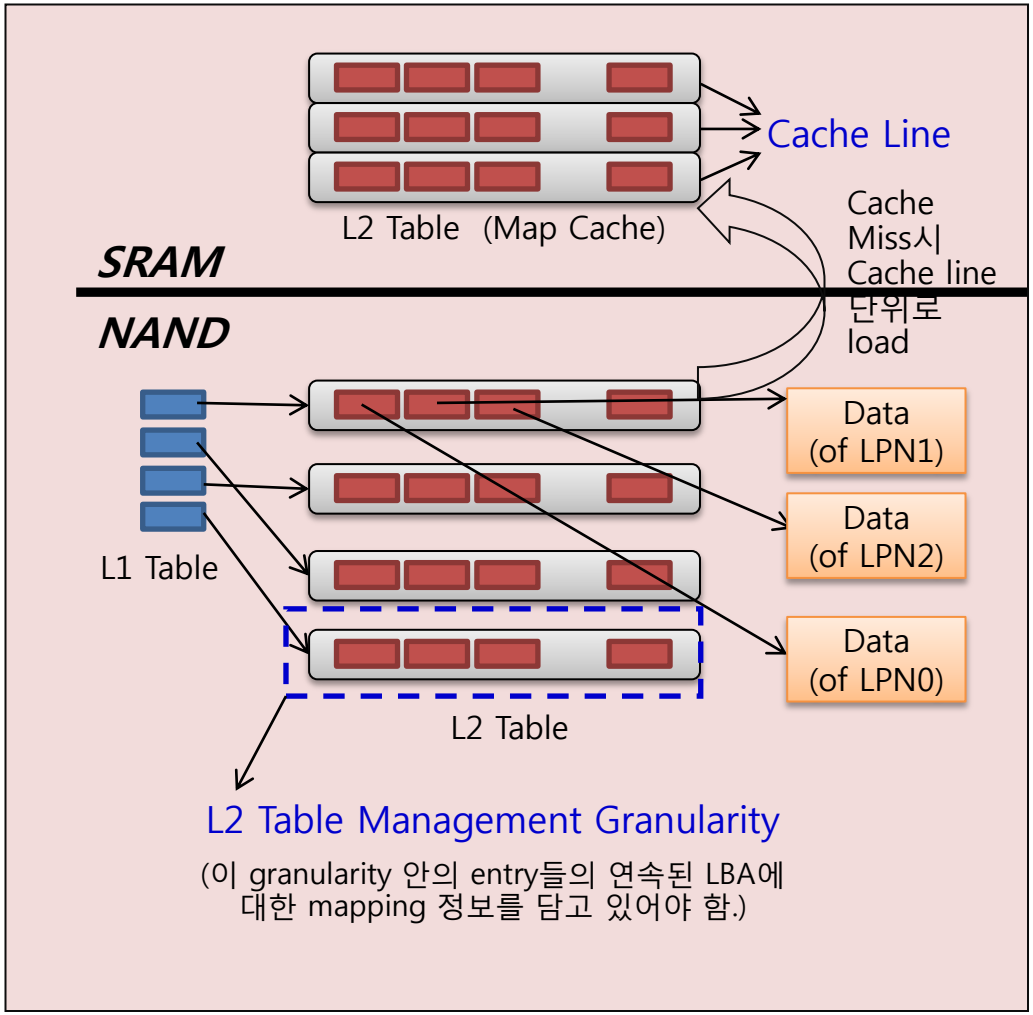
    -. NAND Access time.

    -. Frequency of Map Update.

Design Parameters of Map Cache

- **Cache Size**
- **Cache Line Size (Cache Segment Size)**
- **L2 Table Management Granularity**
- **L1 Table Caching**

- **Replacement Algorithm**
  - LRU, FIFO, Random,…
- **Cache Organization**
  - Two-Level Cache, Zone-based Cache,…

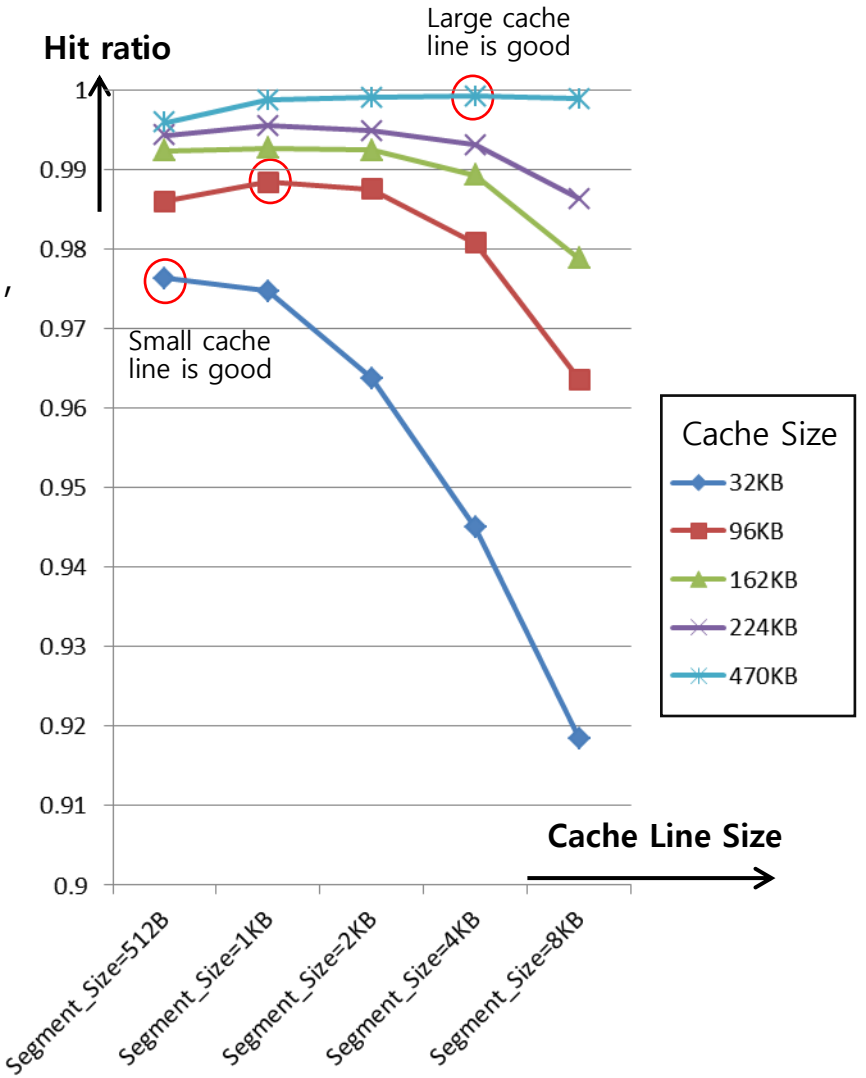


Cache Size

- Cache Size가 크면 hit ratio가 올라감.
- Cache Size가 크면, cost of chip 와 power consumption 이 올라감.
- 각각의 requirement (performance, cost) 를 동시에 만족시키는 최적의 cache size를 찾는 것이 중요.

Cache Line Size

- **Cache line size 는 hit ratio 에 영향을 미침.**
  - Cache line size가 크면, spatial locality에 의해 hit ratio가 올라갈 수 있음.
  - 그러나 line size가 전체 cache size에 비해 너무 크면, cache pollution에 의해 오히려 hit ratio가 떨어짐.
  - 일반적으로 작은 Cache size에서는 작은 Cache line size가 좋음.

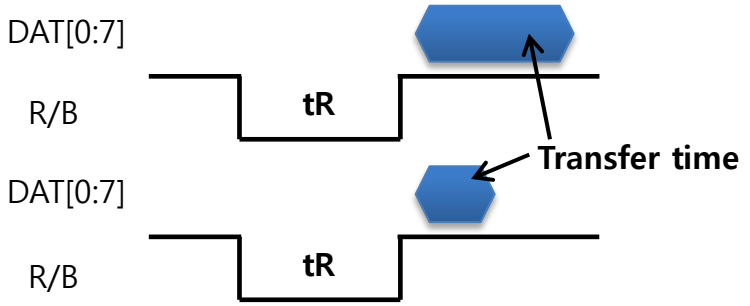


※ We used the workload obtained in mobile system for this simulation

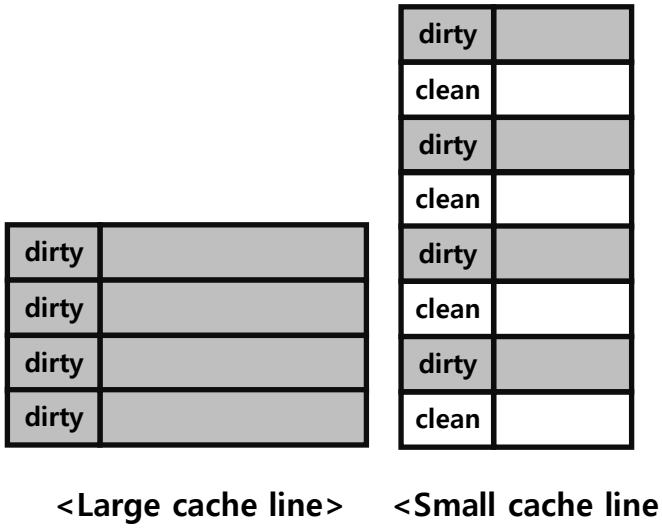
Cache Line Size

Cache line size 는 miss penalty 에 영향을 미침.

- NAND transfer time은 cache line size에 비례.
  - -. NAND I/F 가 DDR400일 경우, line size가 1KB일 때와 8KB일 때의 transfer time의 차이는 18us.
  - -. ECC unit도 고려 해야함.



- Map update 빈도는 cache line size에 비례.
  - -. Line size가 클 경우, 동일한 write operation에 의해 dirty로 mark되는 영역의 크기가 line size가 작을 때보다 더 큼.



Cache Line Size

Simulation Result

- Workload characteristics

#	Workload	R:W Ratio	Amount of Data (Read + Write)	Comment
1	Mobile System #1에서 얻은 Workload	78:22	7.67GB	Read intensive
2	Mobile System #2에서 얻은 Workload	89:11	12.11GB	Read intensive
3	Mobile System #3에서 얻은 Workload	26:74	3.36GB	Write intensive
4	Random Write (address range : full area)	0:100	1.25GB	Chunk size : 4KB
5	Sequential Write (address range : full area)	0:100	32GB	Chunk size : 1MB

- Output Parameters of Simulation

-. **Total Map Read Penalty** : Cache Miss시에 Map Read하는데 소용되는 penalty (tR + transfer time)

-. **Total Map Write (update) Penalty** :

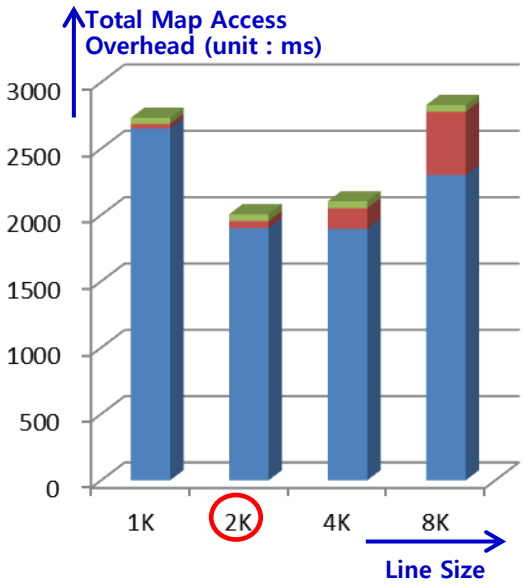
Cache miss시 map update하는데 소요되는 penalty (tPROG + transfer time),

Cache와 NAND간의 consistency를 위해 주기적으로 하는 map update에 소요되는 시간. (tPROG + transfer time)

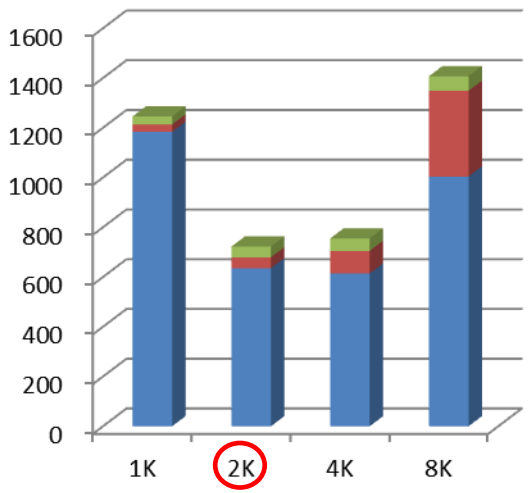
Cache Line Size

Simulation Result

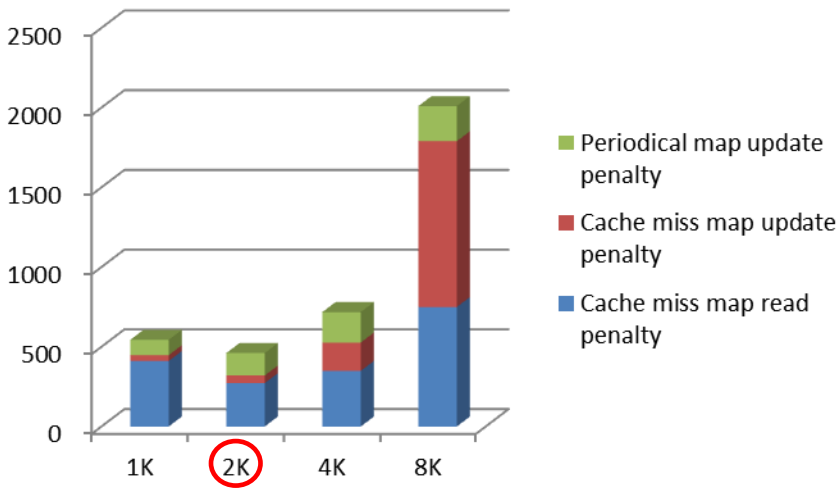
1. System Workload # 1



2. System Workload # 2



3. System Workload # 3



-. 모든 경우에서, map update penalty (빨간색) 은 line size가 클수록 증가함.

이는 line size가 클수록 map update를 자주하기 때문.

-. Map read penalty (파란색) is 2KB 또는 4KB line size에서 가장 작음.

-. System Workload #3는 write operation의 비중이 높음.

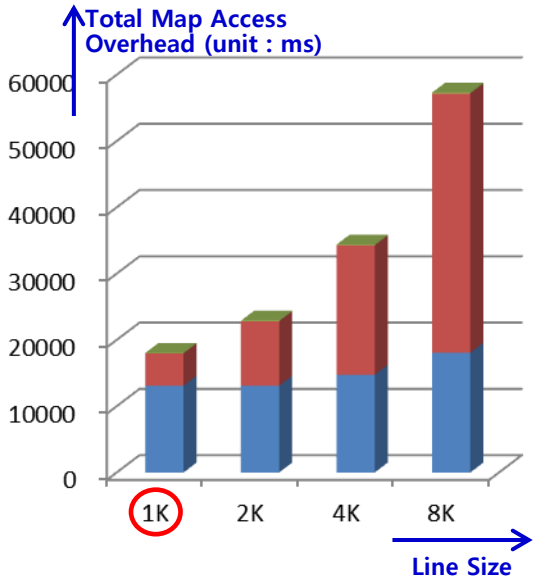
따라서 #1, #2에 비해 map update penalty (빨간색, 녹색)의 비중이 높음.

-. Simulation 결과, System workload에서는 **2KB**의 line size에서 Total Map Access Overhead가 가장 작음.

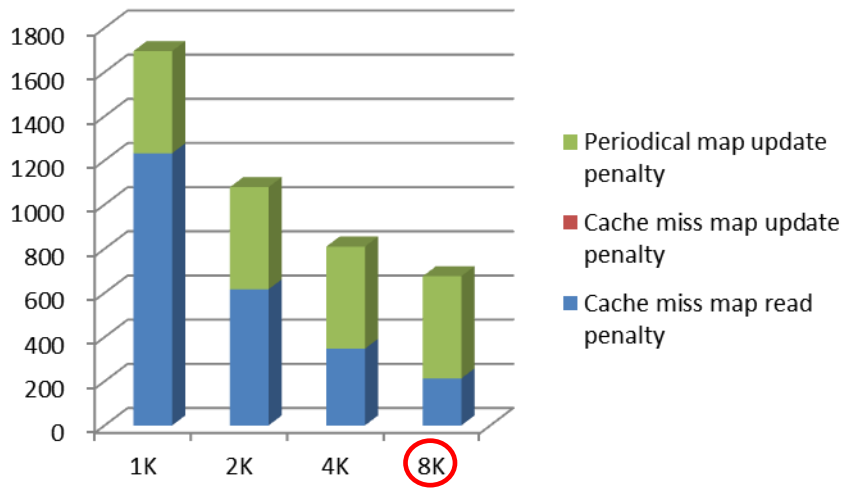
Cache Line Size

Simulation Result

4. Random Write



5. Sequential Write



-. Random write에서 Total map access penalty는 map update penalty (빨간색)에 의해 결정됨.
Map update penalty는 line size가 클수록 증가함. (map update 빈도 때문에)

-. Sequential write에서 Total map access penalty는 map read penalty (파란색)에 의해 결정됨.
Map read penalty는 line size가 클 수록 작은데, 이는 spatial locality를 이용할 수 있기 때문.

-. Sequential write에서 cache miss map update penalty (빨간색)의 값이 0인데,
이는 sequential write에서는 cache가 가득 차기 전에 periodical map update가 먼저 발생해서
dirty line들을 update해버리기 때문. (Sequential write는 cache 의 많은 영역을 dirty로 만들지 않음)

L2 Table Management Granularity

- **L2 Table Management Granularity는 L1 Table Size에 영향을 미침.**
  - **Number of L1 Table Entry 는 다음과 같은 식으로 표현 가능.**

$$\frac{Total\ Density}{FTL\ Mapping\ Granularity} \Big/ \frac{L2\ Table\ Management\ Granularity}{Size\ of\ one\ entry}$$

Ex) when the condition is

-. Device density : 64GB,

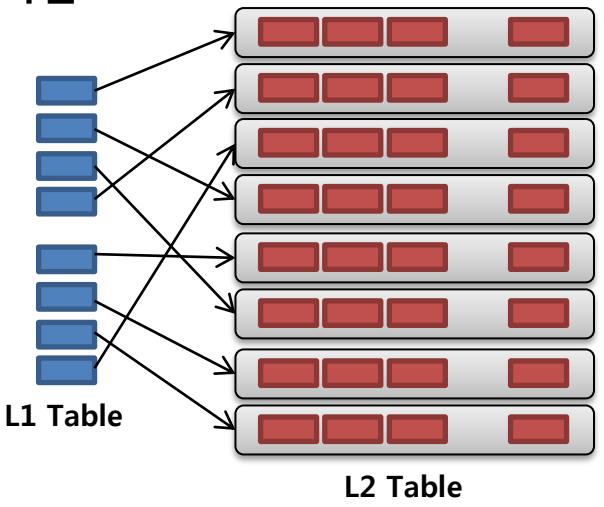
-. FTL mapping : 8KB Logical Page Mapping,

-. L2 Table Granularity : 4KB,

-. size of one entry : 4Byte,

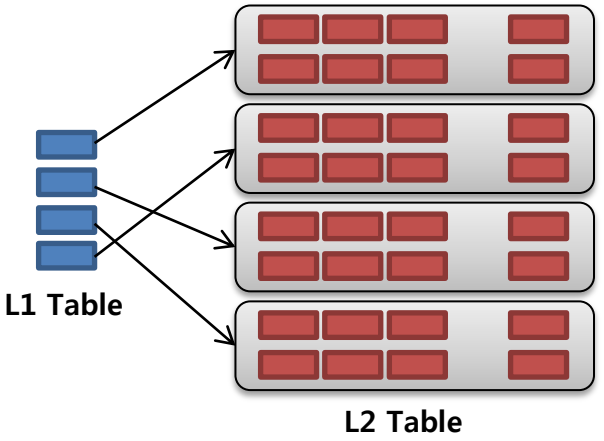
Number of L1 Table entry : **8KB**

Total L1 Table Size : **32KB**



<Small L2 Table Management Granularity>

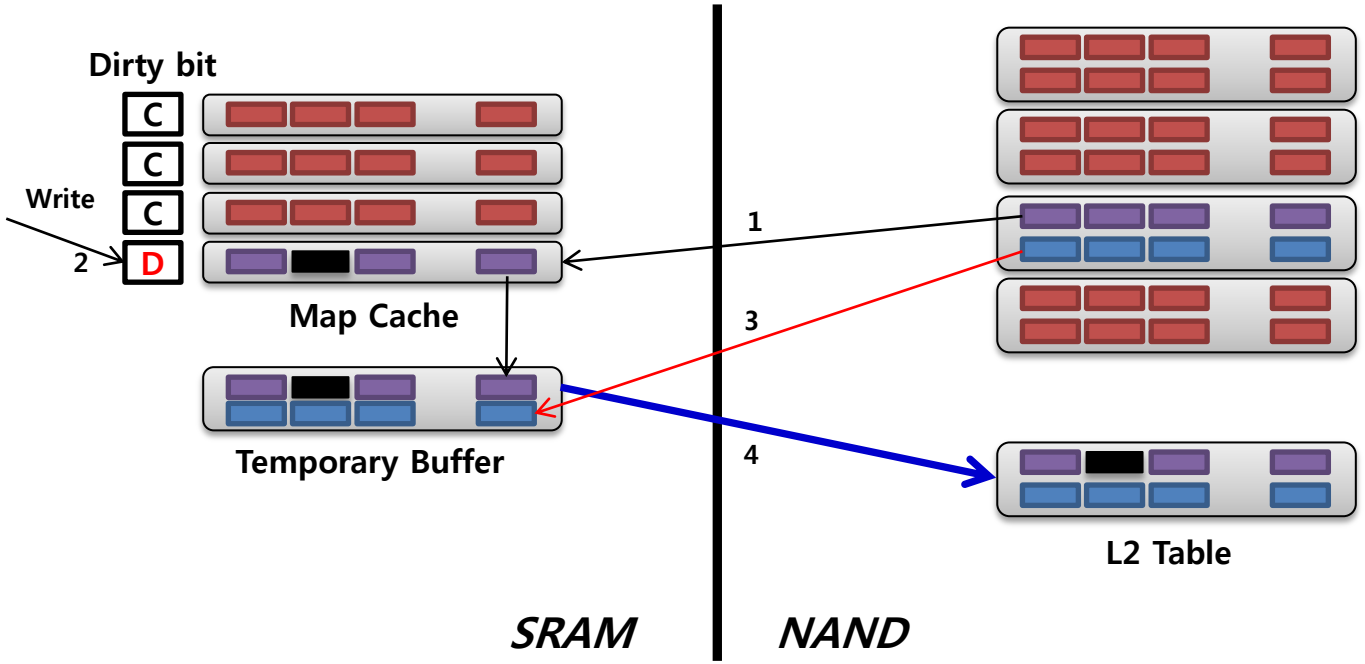
- 일반적으로 L1 table 전체가 SRAM에 올라가기 때문에, SRAM size관점에서는 L2 table management granularity가 클 수록 유리함.



<Large L2 Table Management Granularity>

L2 Table Management Granularity

- **L2 Table Management Granularity는 miss penalty 에 영향을 미침.**
  - L2 Table Management Granularity 가 Cache line size 보다 더 클 경우,
    Map update시에 **READ-MODIFY-WRITE** 이 발생하여 miss penalty를 증가시킬 수 있음.

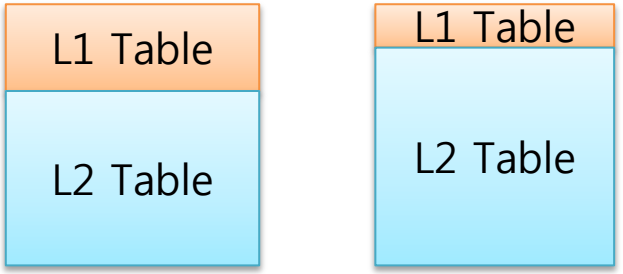


1. Cache miss시 upper half만 cache 에 load.
2. Map cache 에 table을 update 하고, Dirty line으로 mark
3. 후에 이 line이 victim으로 선정 되었을 때, lower half를 NAND 로부터 load 해서 L2 table management granularity 로 merge.
4. Merge가 완료되면, 이를 새로운 map page에 update.

<L2 table management granularity 가 line size보다 2배 더 클 때>

L1 Table Caching

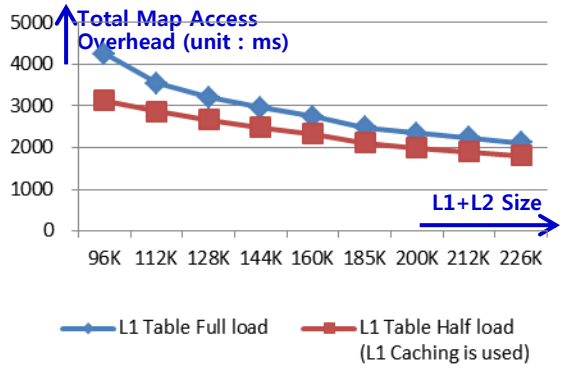
- **일반적으로 L1 Table은 전체가 SRAM에 상주함. 하지만 L1 Table의 size가 너무 클 경우 L1 Table caching을 고려해 볼 수도 있음.**
- **L1 table caching을 위해서는, L1 table을 가리키는 또 하나의 table이 필요. (ex:L0 table).**
  - L0 Table -> L1 Table -> L2 Table -> Data Block
- **단점**
  - 구현이 복잡.
  - cache에서 L1 Table miss가 발생하면, L1 table을 load하고, 그 후에 L2 Table을 load해야 하기 때문에, miss penalty가 증가.
- **장점**
  - 작은 SRAM의 사용.
  - L1 Table Caching에 의해서 절약된 SRAM 공간이 L2 table로 사용 된다면 L2 Table hit ratio가 증가. 만약 L1 Table miss가 자주 발생하지 않는 상황이라면 (workload의 locality가 높을 때) L2 Table hit ratio 증가로 인해 전체 performance가 좋아질 수 있음.



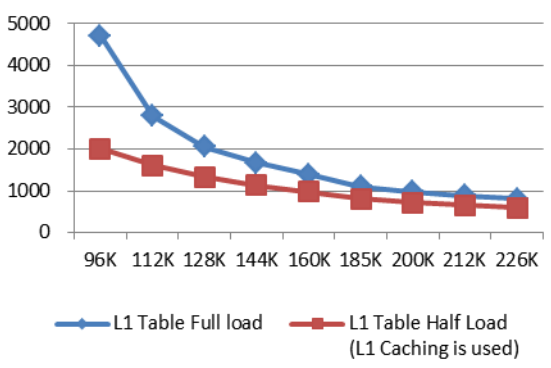
L1 Table Caching

Simulation Result

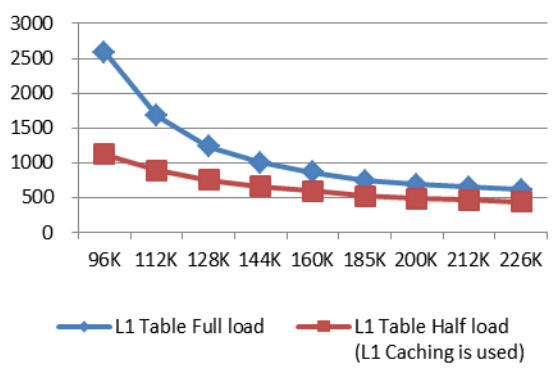
1. System Workload #1



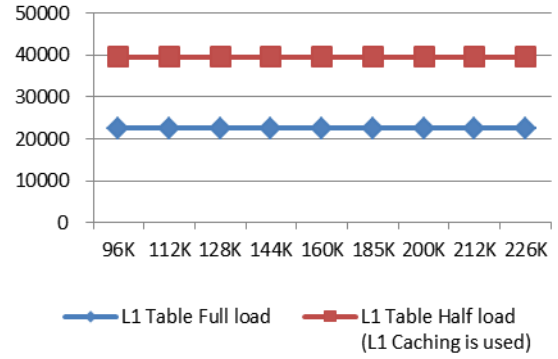
2. System Workload #2



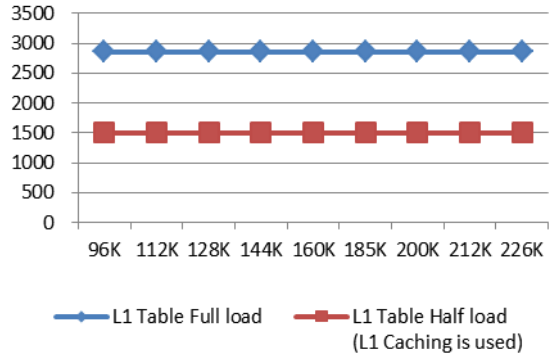
3. System Workload #3



4. Random Write (full area range)



5. Sequential Write (full area range)



- 4. Random Write에서는 L1 Table Miss가 빈번하게 발생하기 때문에, L1 caching (빨간색) 의 성능이 좋지 않음.
- 나머지 case에서는, L1 miss 가 가끔 발생하기 때문에 L1 caching의 성능이 좋음. (L2 hit ratio의 증가로 인해)

Conclusion

- **Map Cache based FTL은 mobile storage 에서 널리 사용.**
- **Map Cache based FTL에서 Map cache design은 performance와 cost 측면에서 매우 중요.**
- **Map Cache design에는 여러 parameters 와 algorithms 이 존재.**
  - Cache size 는 performance and cost requirement를 동시에 만족해야 함.
  - Cache line size 는 performance에 영향을 미침.
  - L2 table management granularity 는 SRAM size 와 performance 에 영향을 미침.
  - L1 table size 가 너무 클 경우, L1 table caching을 고려해 볼 수 있음.
- **Map Cache Design 단계에서 최적의 parameter와 algorithm을 찾기 위해서는 Sophisticated Simulation 이 필요.**

Thank You