S-FTL: An Efficient Address Translation for Flash Memory by Exploiting Spatial Locality

Song Jiang, Lei Zhang, Xinhao Yuan, Hao Hu, and Yu Chen

Department of Electrical and Computer Engineering
Wayne State University

Department of Computer Science and Technology
Tsinghua University

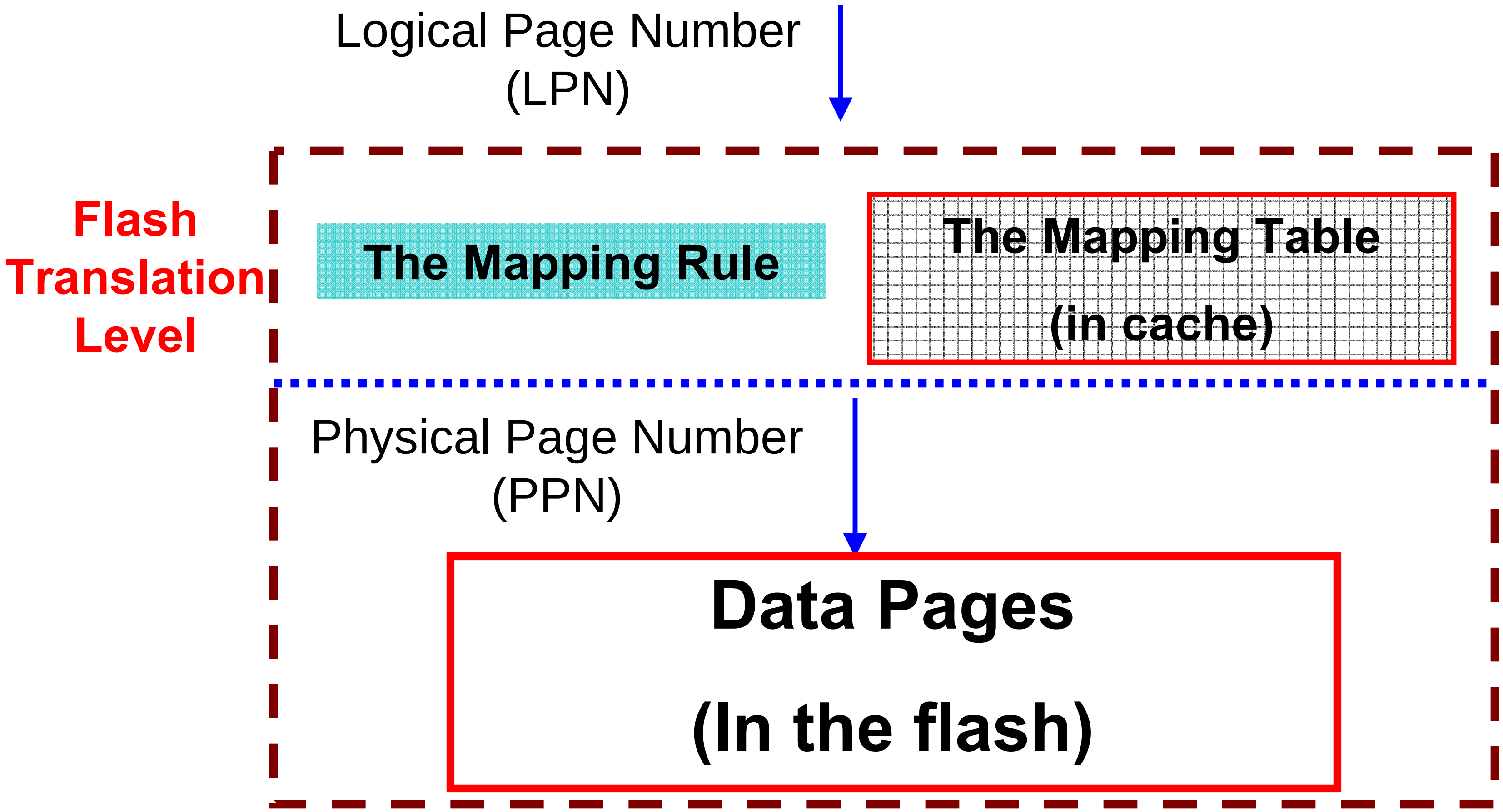
Making Solid-State Disk Cost Competitive

- The solid-state disk (SSD) is becoming increasingly popular.
  - Great performance advantage over HDD.
  - Uncompetitive price as HDD keeps rapidly dropping price.
  - Has to adopt low-end configurations to reduce cost (MLC, small DRAM cache).

- The built-in cache is performance critical.
  - Much fast than the flash, especially for write.
  - Used as buffer for data pages (well studied)
  - Used as a buffer for mapping table of address translation.

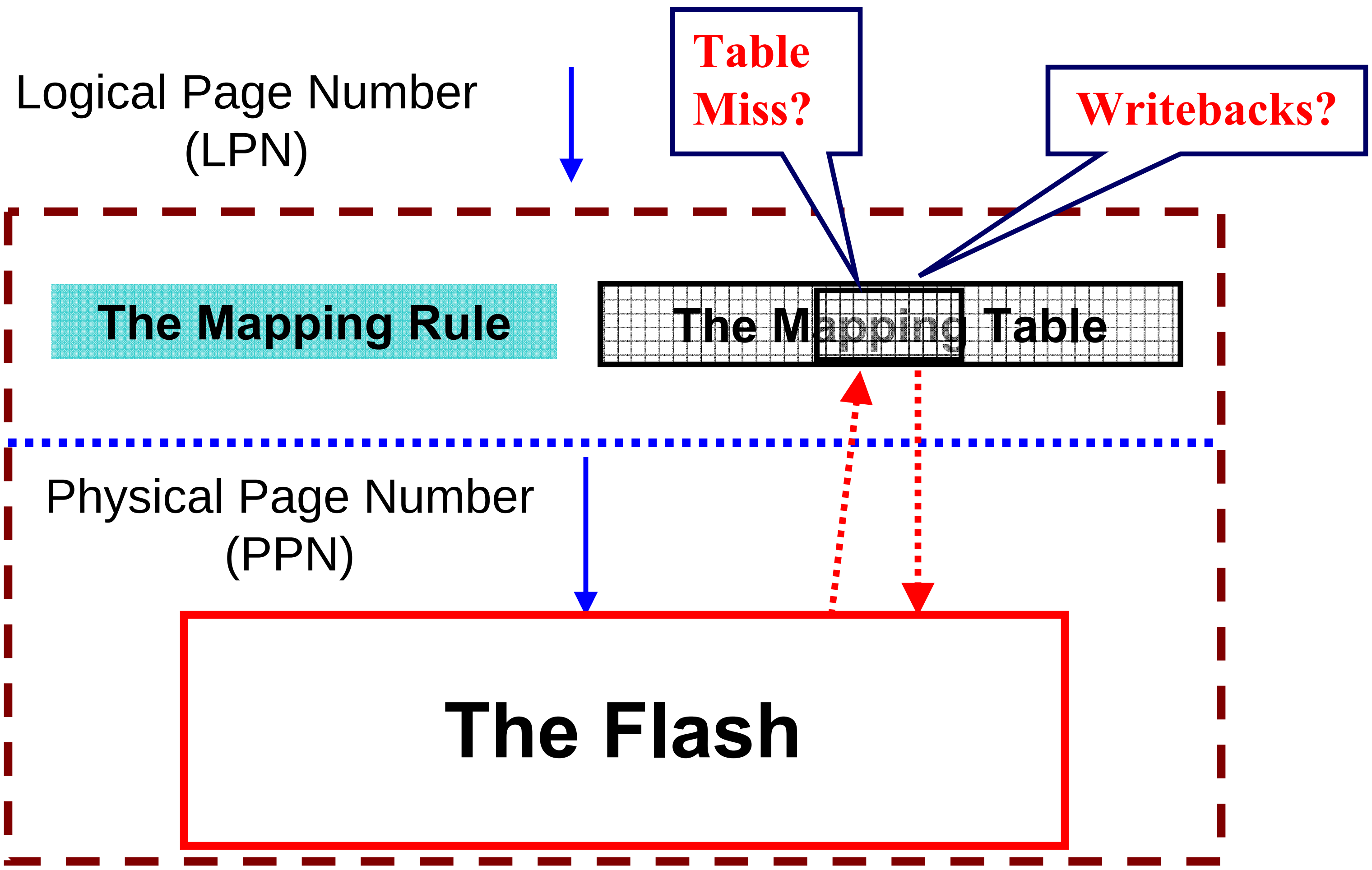
- The challenge: how to maintain efficient SSD operations even with a small buffer cache?

Flash Address Translation

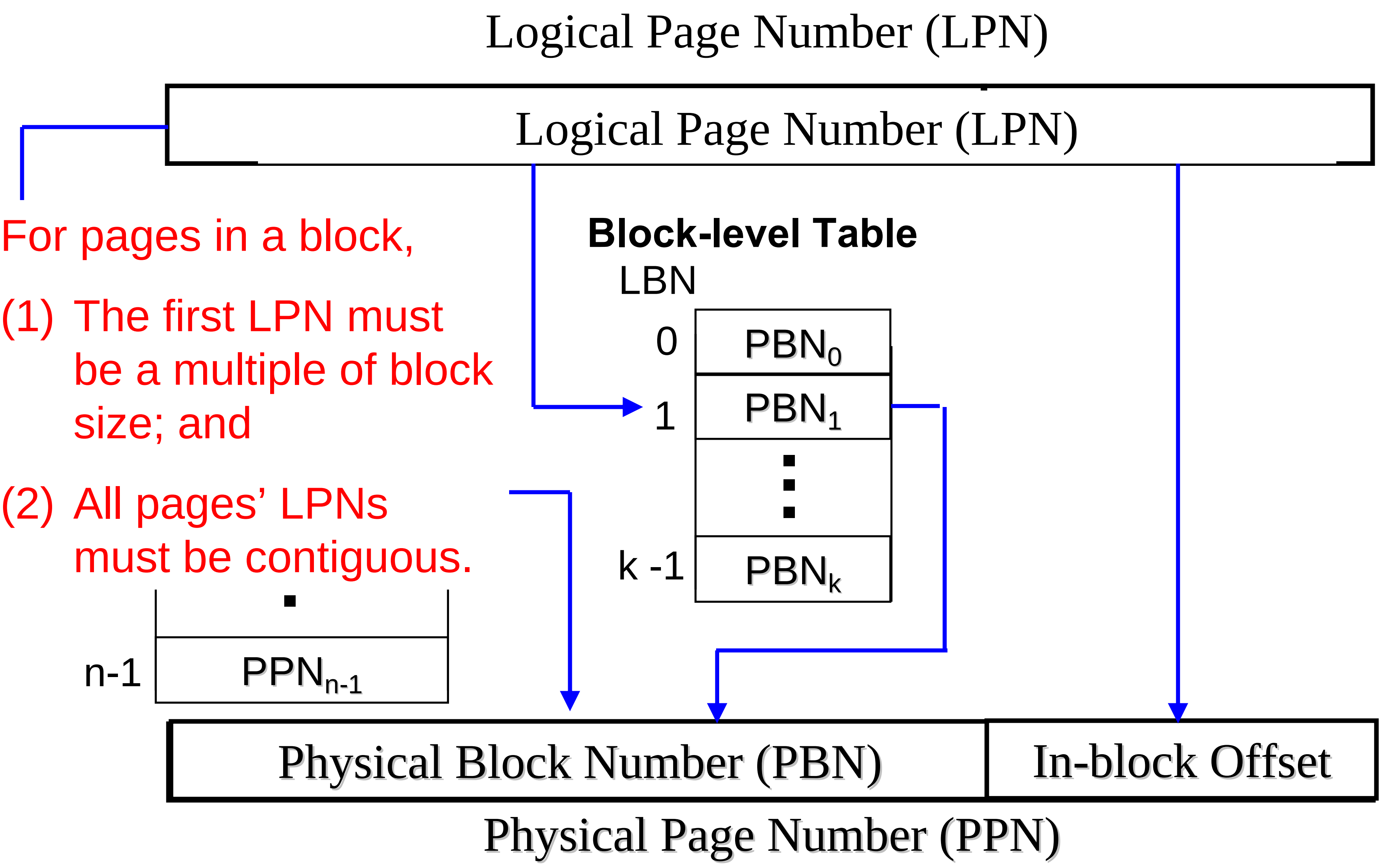


What if the table is too large to be fully held in the cache?

Method 1: Move the Table to the Flash (Page-level FTL)



Method 2: Make the Table Smaller (Block-level FTL)



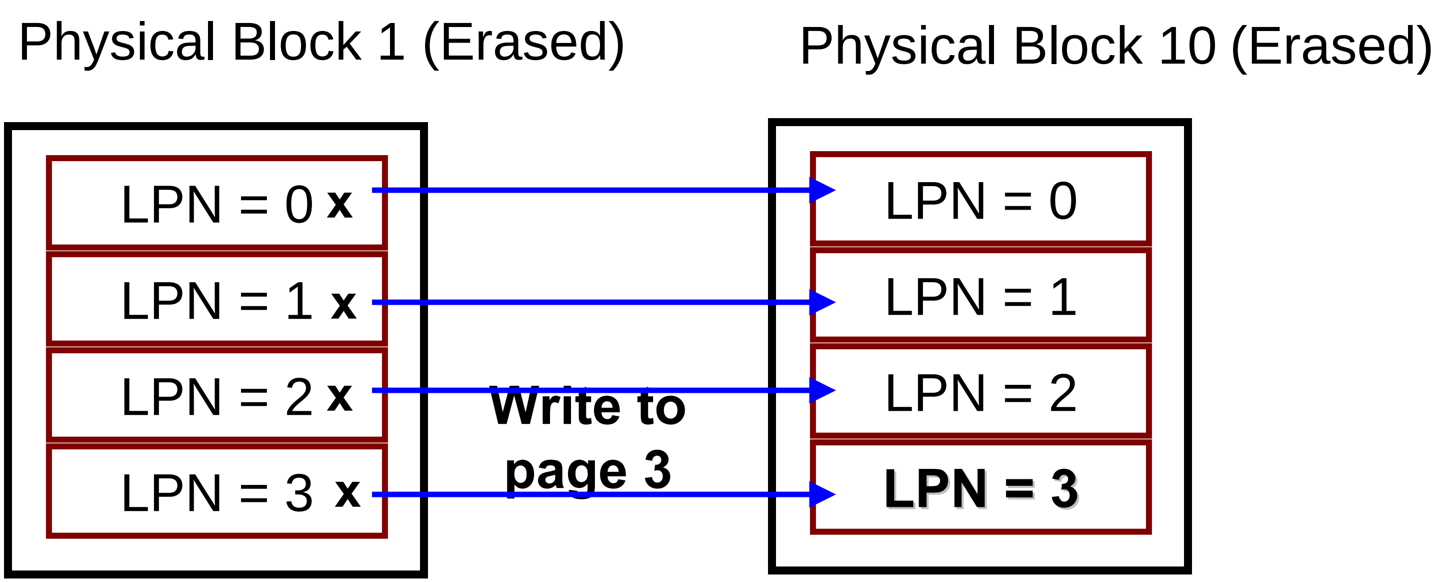
For pages in a block,

(1) The first LPN must be a multiple of block size; and

(2) All pages' LPNs must be contiguous.

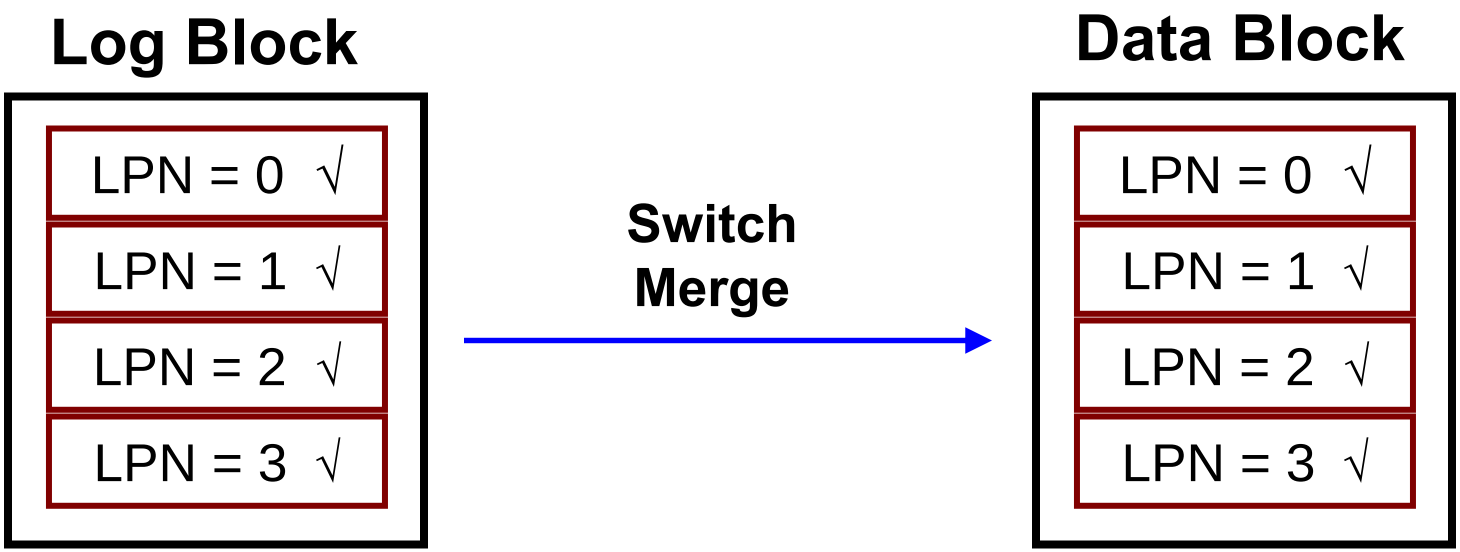
Maintaining Block-based Page Layout is Expensive!

- The flash requires “erase before write”, or has to “out of place” write to erased place.
  - The erase unit is block, which usually contains 32-128 pages.
  - Erase operation is expensive (2ms compared with 0.1ms for read and 0.4ms for write)
- For the block-level FTL, one page write can lead to tens of reads/writes.

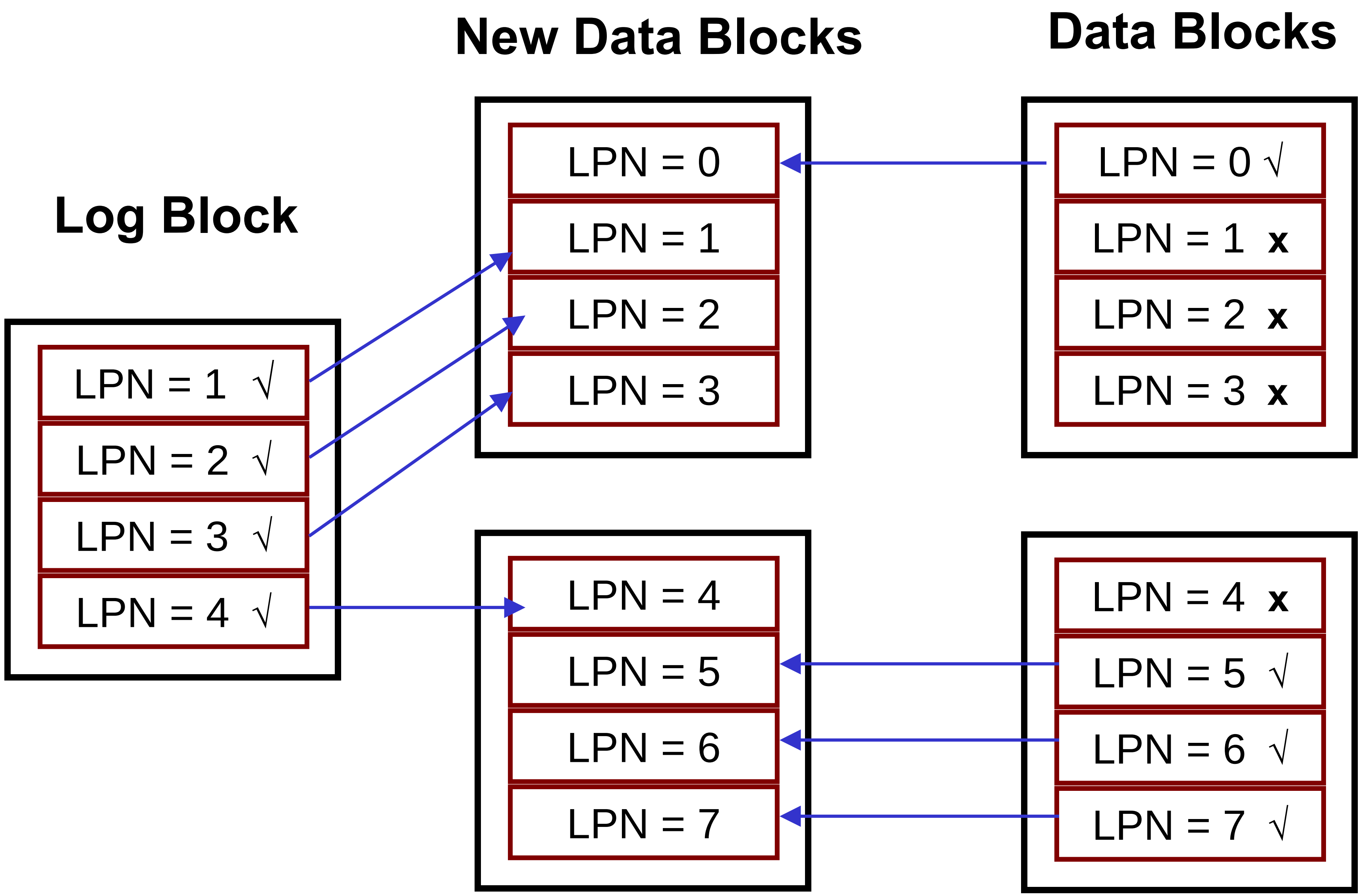


Hybrid FTL attempts to Address the Issue

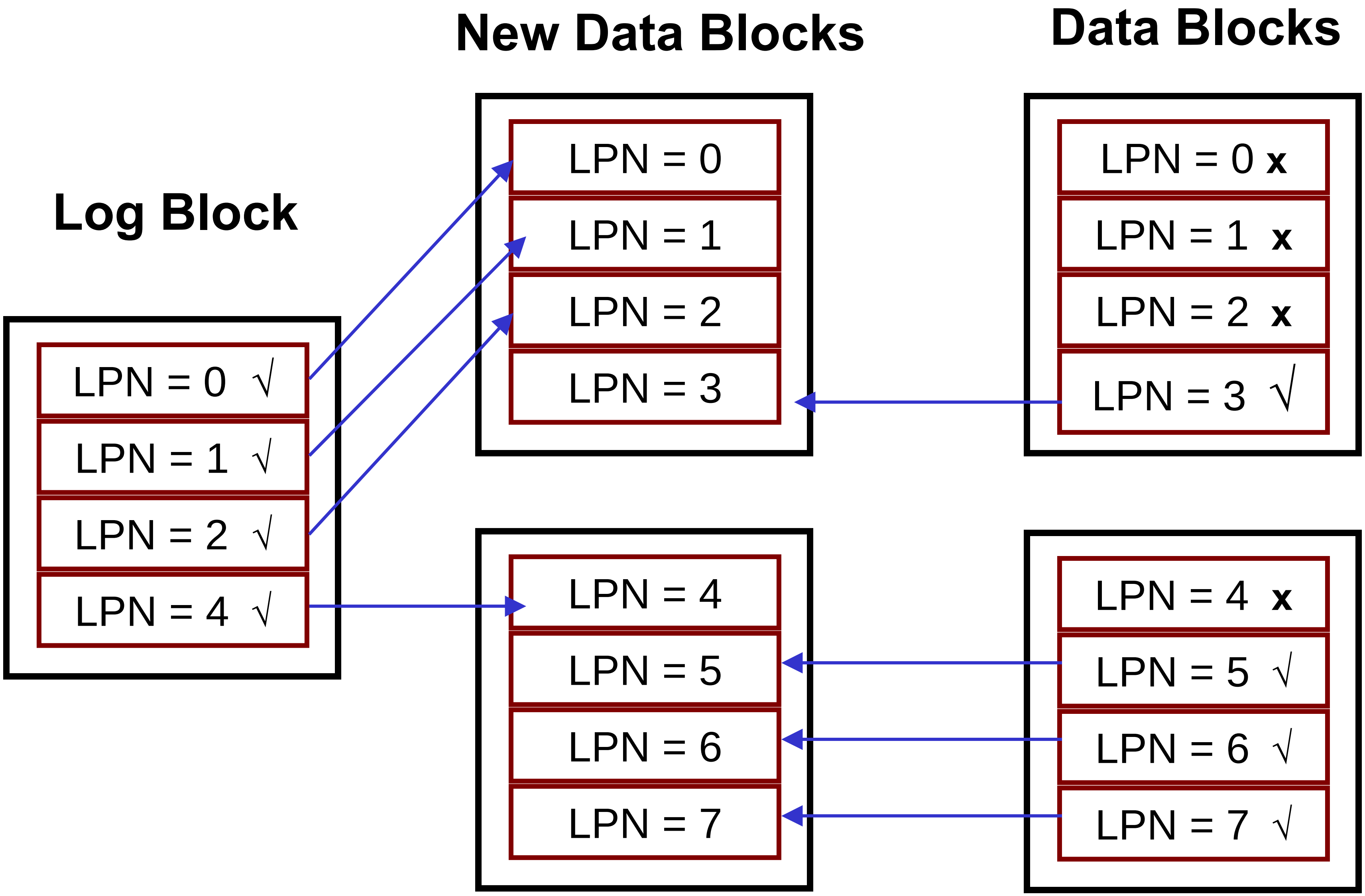
- Set aside a small number of log blocks to hold newly written pages managed by the page-level mapping.
- Majority of pages stay in the data blocks managed by the block-level mapping.
- But garbage collection and block merging can still be expensive!
  - because the rule for the page layout in a block is so strict!



Only Block-based Page Layout can Help



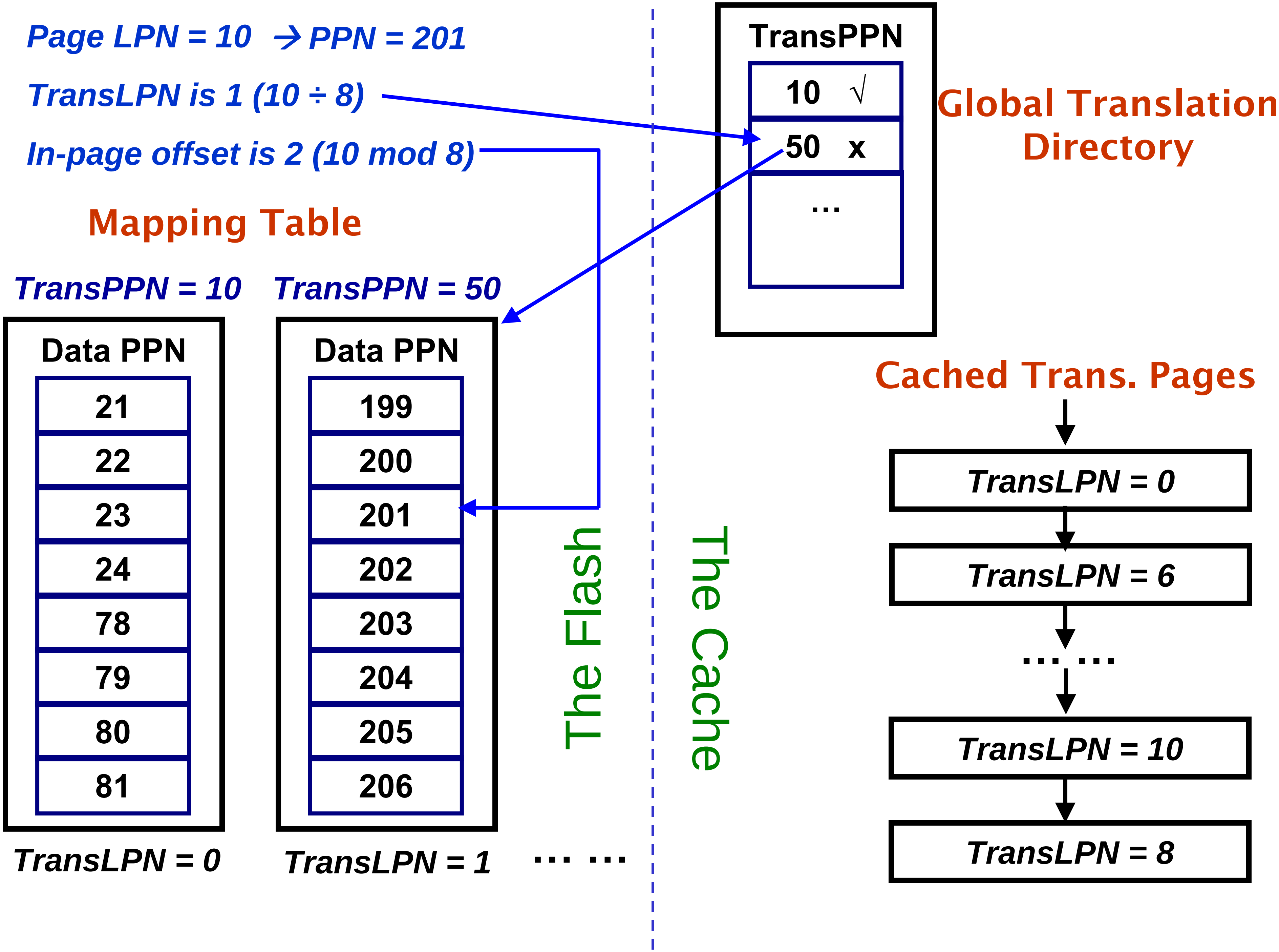
Only Block-based Page Layout can Help (cont'd)



S-FTL: Spatial-locality-aware FTL

- S-FTL does not impose mapping regularity to reduce mapping table.
  - It uses page-level mapping.
  - The table is stored in the flash and cached in the memory.

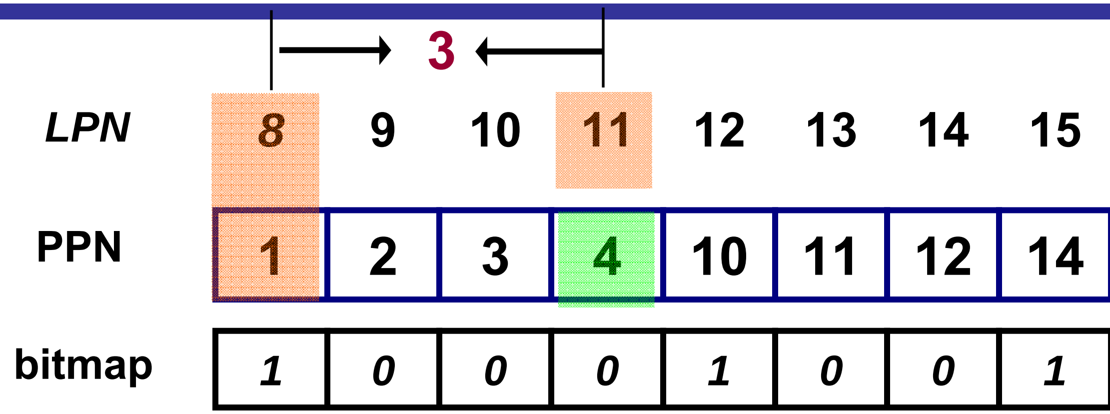
- S-FTL can reduce the cached table by exploiting readily available locality in the access streams.
  - Any access sequentiality exhibited in the request stream can be used to reduce the table.
  - The more sequential, the more reduction.



Page LPN = 10 → PPN = 201
TransLPN is 1 (10 ÷ 8)
In-page offset is 2 (10 mod 8)
Mapping Table
TransPPN = 10
TransPPN = 50
Data PPN
21
22
23
24
78
79
80
81
Data PPN
199
200
201
202
203
204
205
206
TransLPN = 0
TransLPN = 1
The Flash
The Cache
TransPPN
10 √
50 x
Global Translation Directory
Cached Trans. Pages
TransLPN = 0
TransLPN = 6
TransLPN = 10
TransLPN = 8

Make the Translation Page Smaller

Translation Page



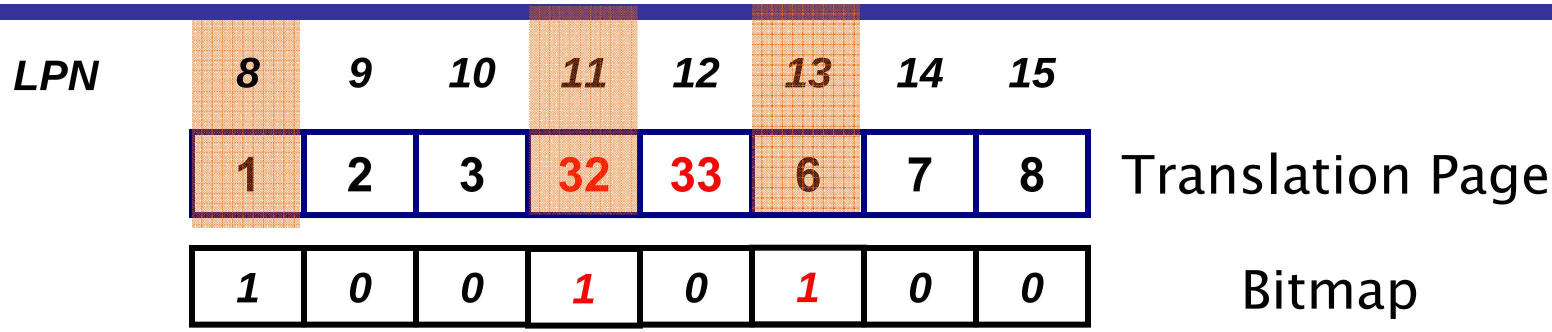
- If we know some PPNs are contiguous, all PPNs can be computed from the first mapping entry (head entry).
  - An example:

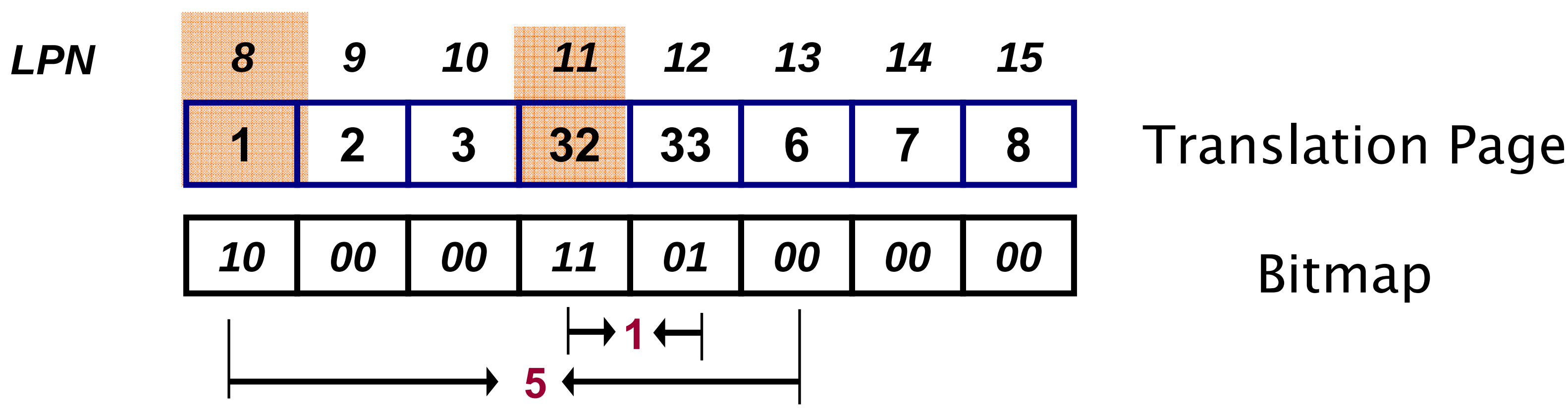
    PPN of LPN 11 is 4 [1 + (11 - 8)]

  - Therefore, only the head entry needs to be stored.

- Use bitmap to record the contiguousness of PPNs.
- How much space can be saved?
  - Assume 512 entries / page, 4B entries, and fully contiguous PPNs in the page ➔ reduced by 96.6%.
  - Assume average contiguous sequence length is 2 ➔ reduced by nearly 50%.

Use Multi-bit Bitmap to Retain Long Sequences





- The first bit: is it a head entry?
- The second bit: is it my head entry?

Efficient Caching and Batched Writeback

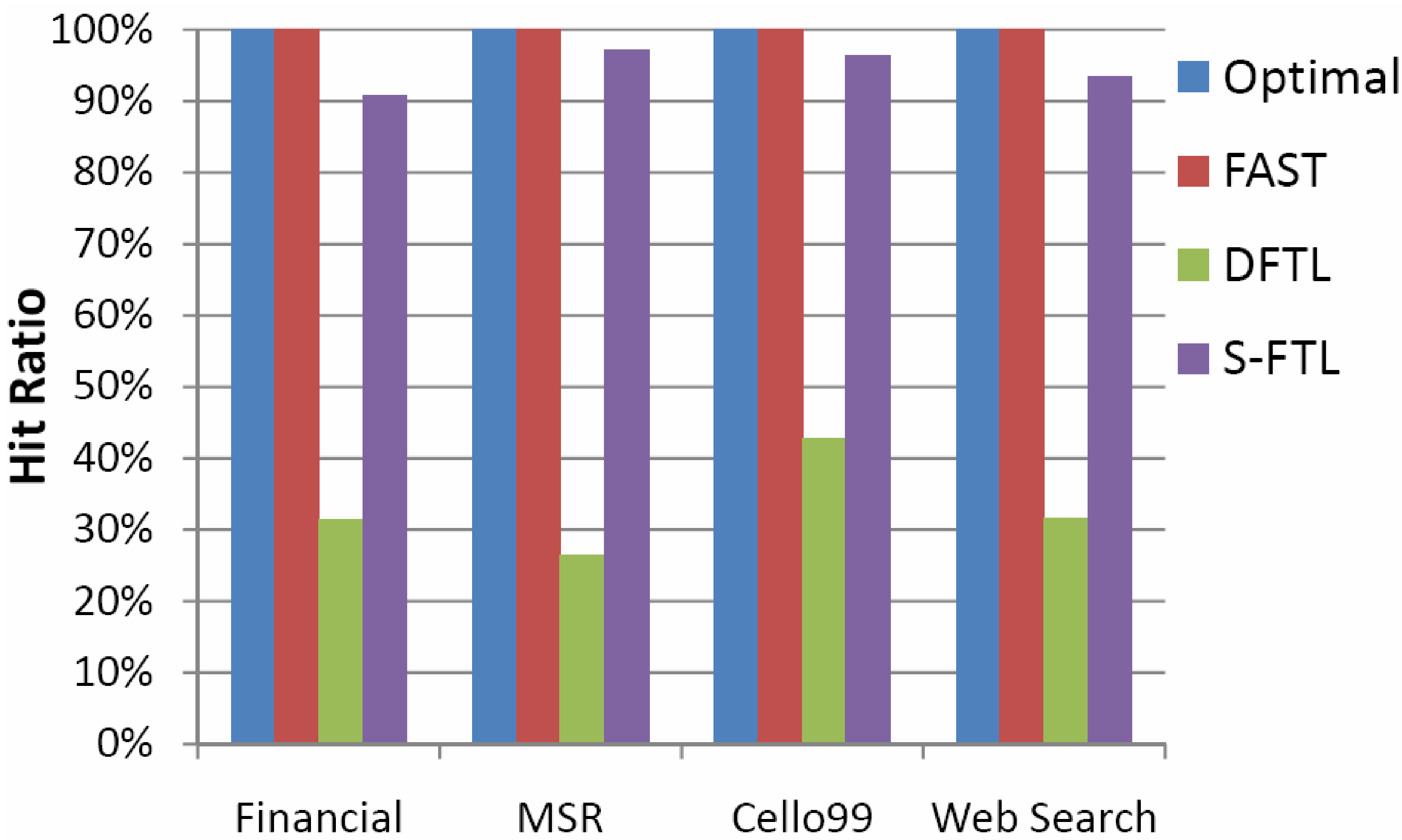
- Exploit sequential access for efficient caching.
  - Two convertible representations of a translation page: In-flash form and bitmap form.
  - The size of translation page in the bitmap form changes.
  - Use translation page as the caching unit.
  - More space-efficient form is used for caching.
- Use batched writeback to reduce overhead.
  - Replaced dirty pages need to be written back.
  - If there are only few dirty entries in a page, keep them in the cache for longer time.
  - Batch the dirty entries for cost-effective writeback.

Experiment Setup

- The SSD to be simulated:
  - 32GB large-block NAND SSD
  - 2KB pages, 128KB blocks, and 64KB cache by default.
  - 0.12ms page read, 0.41ms page write, and 2.0ms block erasure.
  - Using the enhanced FlashSim simulator.
- The FTLs to be compared:
  - DFTL: use page-level mapping and cache recently used mapping entries. [Kim, et al. ASPLOS'09]
  - FAST: use hybrid mapping. [Sang, et al, ACM TECS 2007]
  - Optimal FTL: use page-level mapping with an infinitely large cache
- The traces

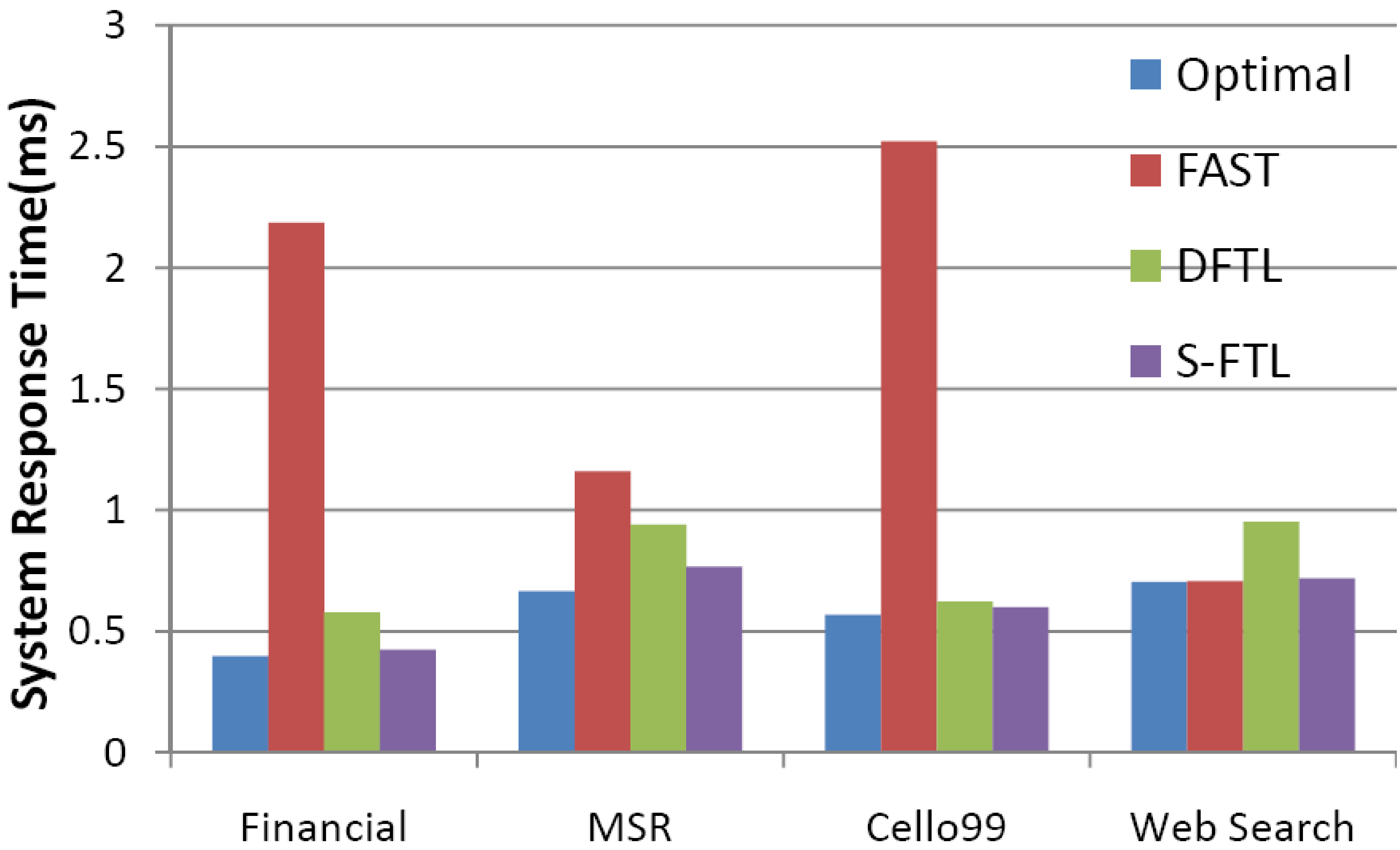
Workloads	Request Size (KB)	Read (%)	Seq. Read (%)	Seq. Write (%)
Financial	9.85	23.16	0.71	0.40
MSR	9.31	18.45	13.86	2.60
Cello99	9.87	57.34	2.6	0.92
Websearch	32.29	99.98	6.47	0.94

Hit Ratios

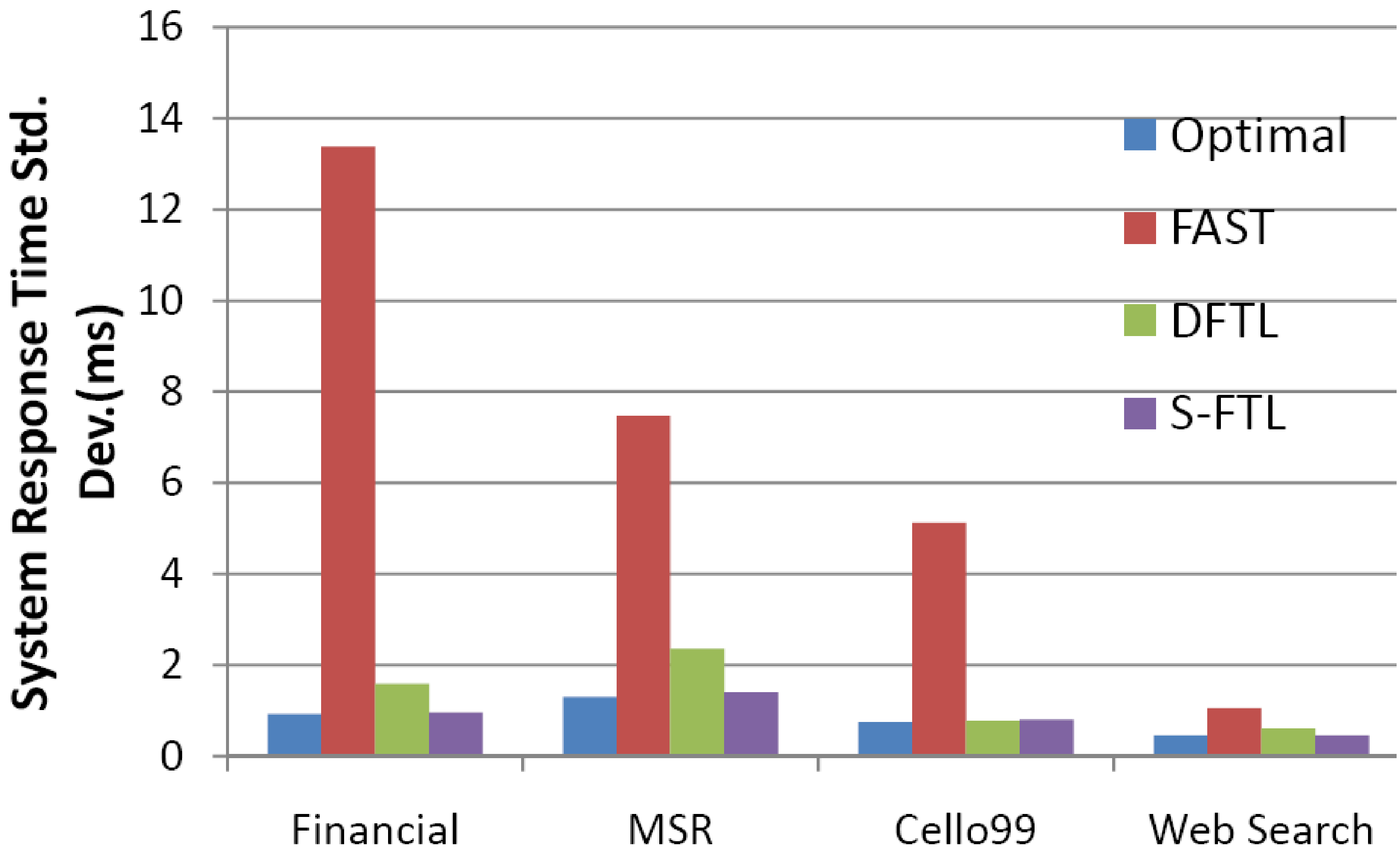


Hit Ratio
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
Financial
MSR
Cello99
Web Search
Optimal
FAST
DFTL
S-FTL

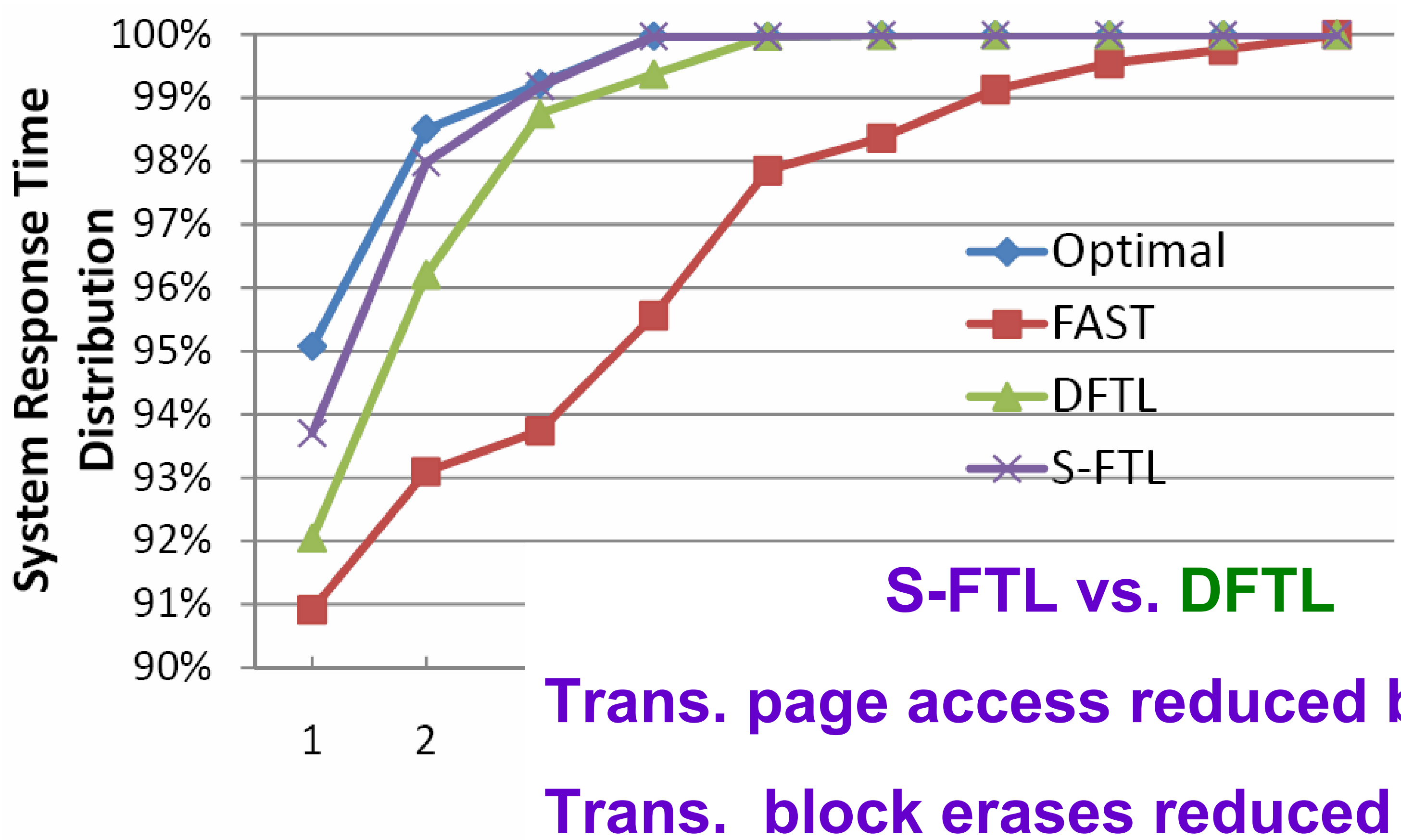
Response times



Standard Deviation of Respond Time



Distribution of System Response Time (*Financial*)

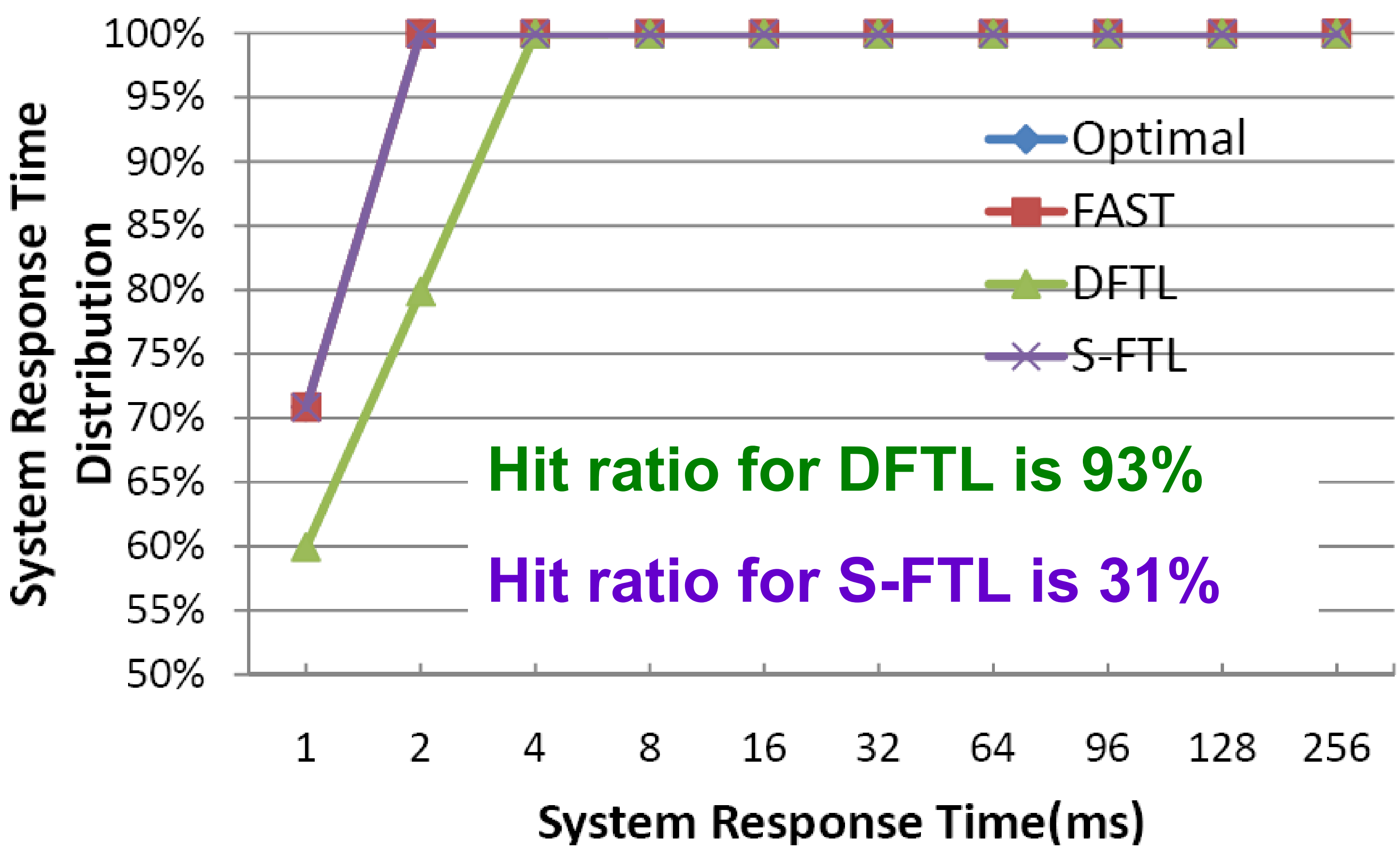


S-FTL vs. DFTL

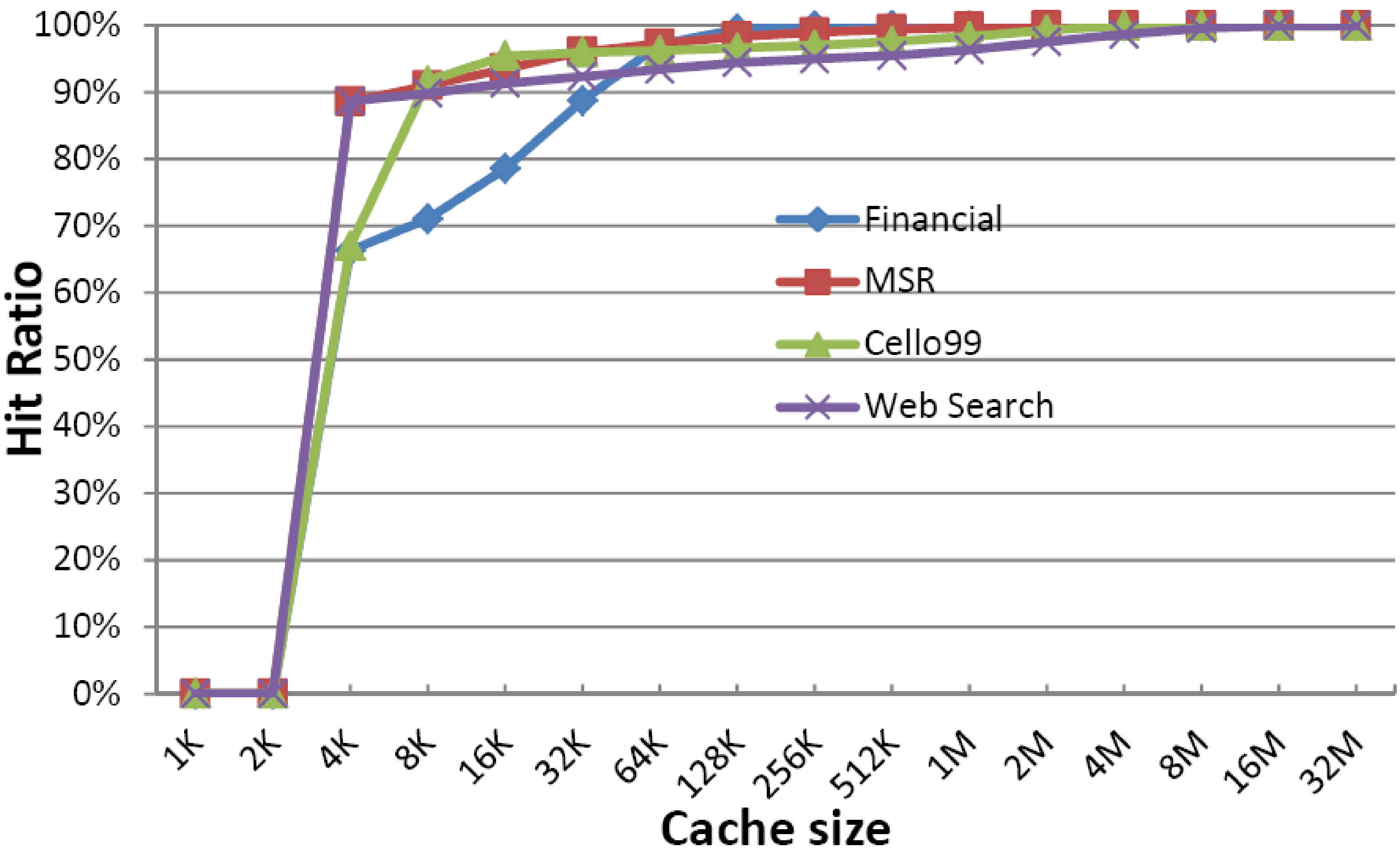
Trans. page access reduced by 93%

Trans. block erases reduced by 28%

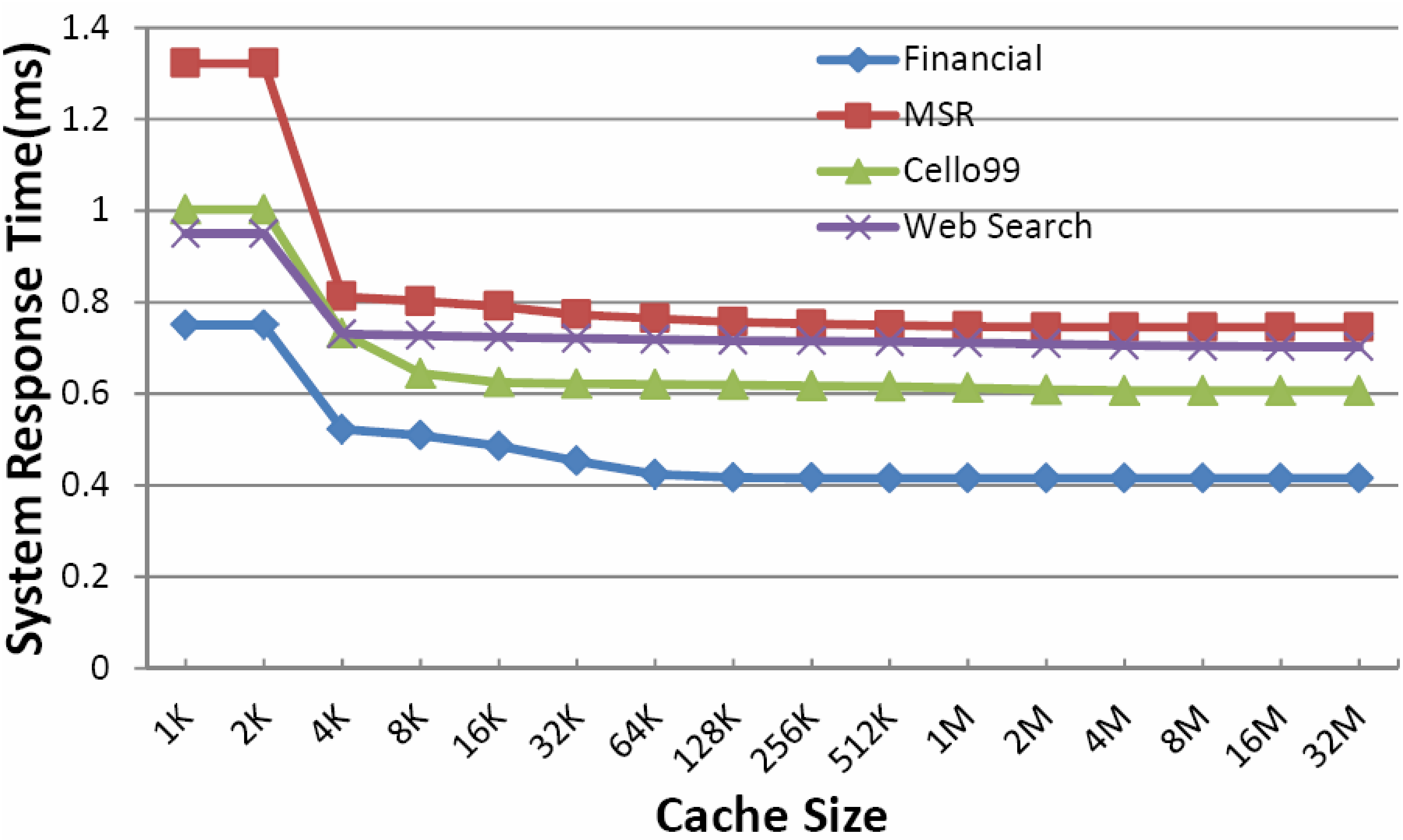
Distribution of System Response Time (*WebSearch*)



Impact of Cache Size on Hit Ratio



Impact of Cache Size on Response Time



Conclusions

- S-FTL reduces space demand for caching mapping table
- S-FTL exploits only readily available spatial locality
- S-FTL does not impose strict mapping rule and does not introduce additional garbage collection cost.
- Experiments demonstrate that it can consistently improve response times for workloads of diverse access behaviors.