SRM-Buffer:

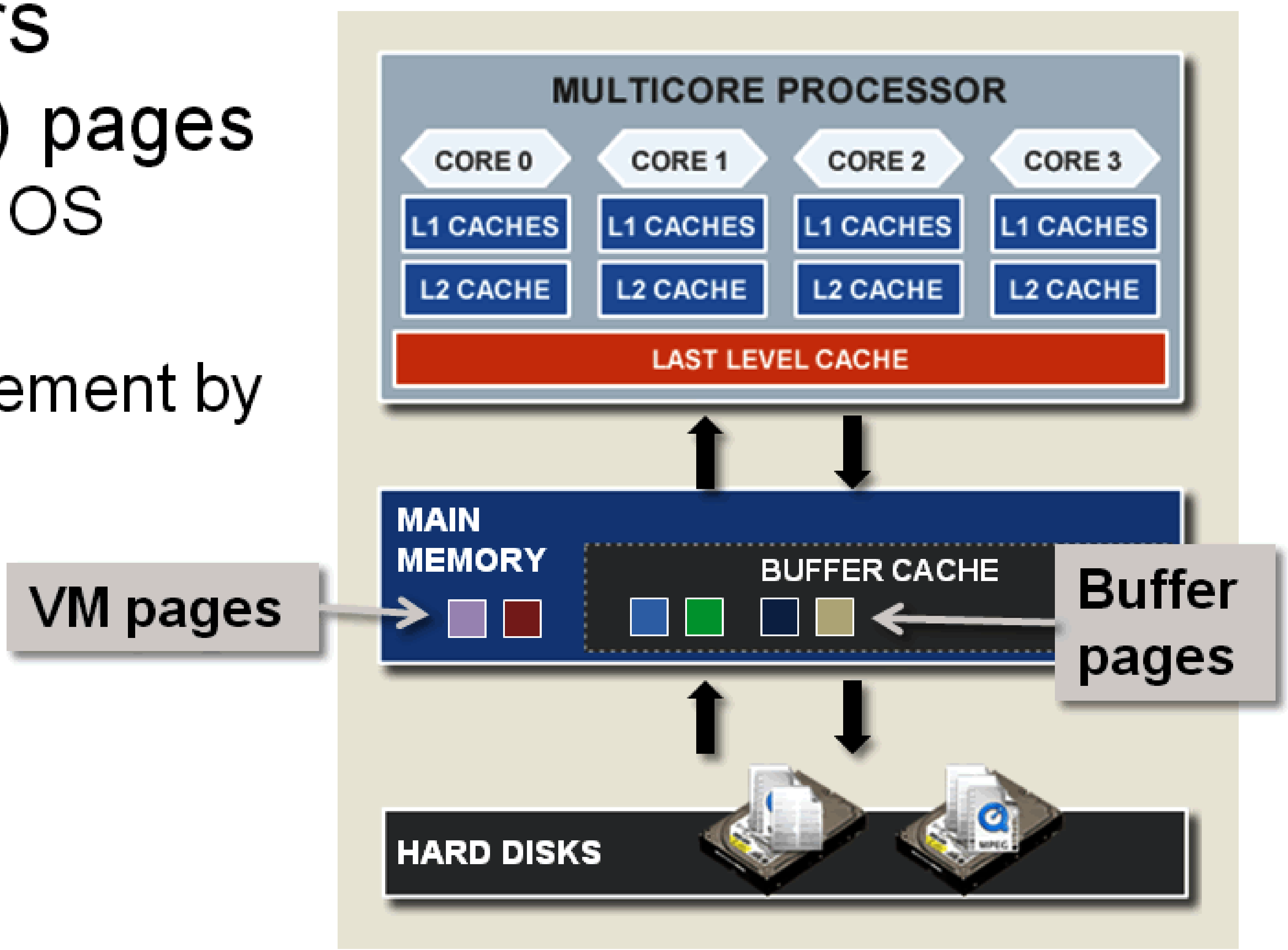
An OS Buffer Management Technique to Prevent Last Level Caches from Thrashing in Multicores

Xiaoning Ding
Intel Labs Pittsburgh

Kaibo Wang , Xiaodong Zhang
The Ohio State University

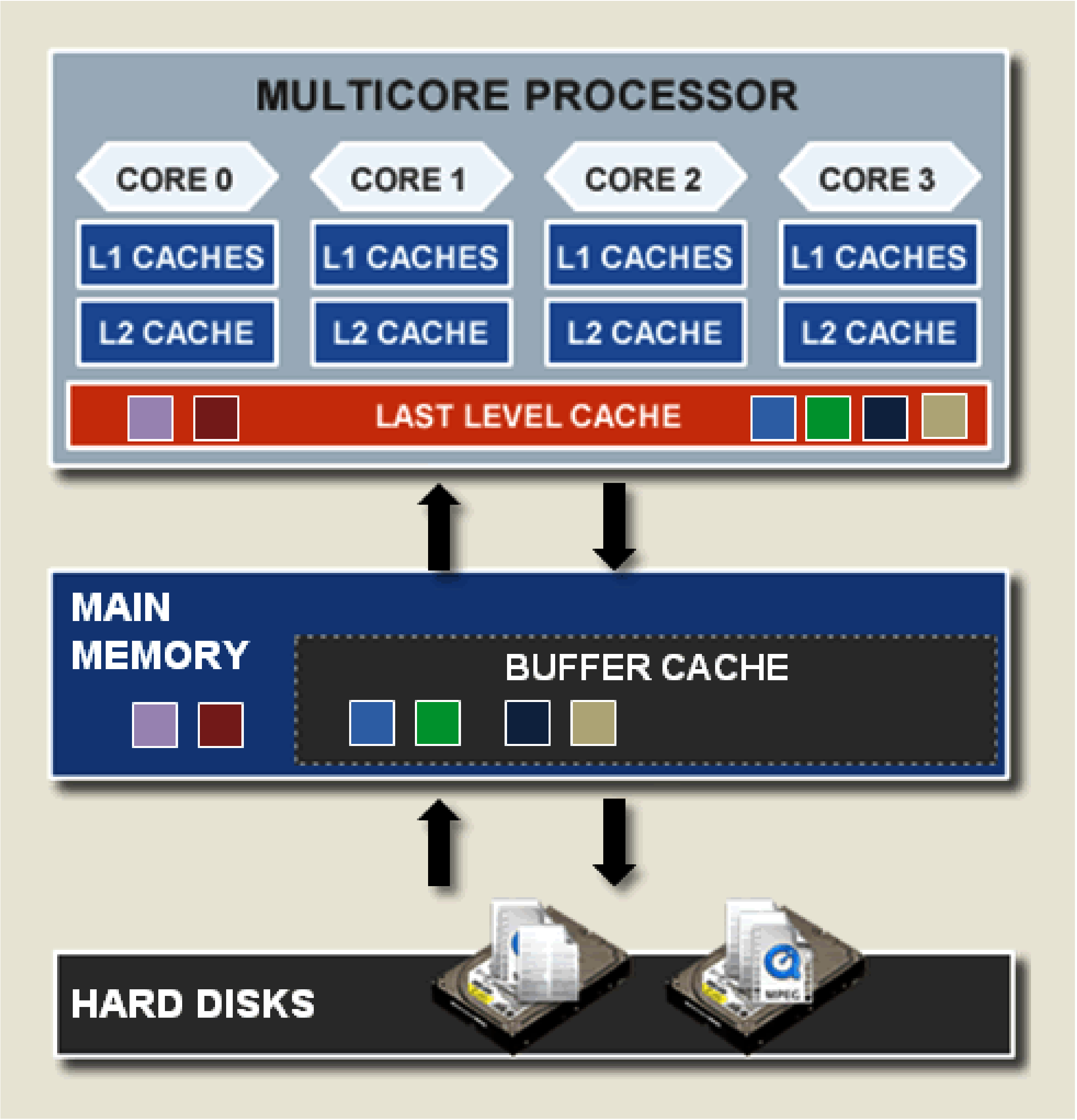
Memory Hierarchy in Multicores

- Main memory buffers
  - Virtual memory (VM) pages
    - VM management by OS
  - File blocks
    - Buffer cache management by OS



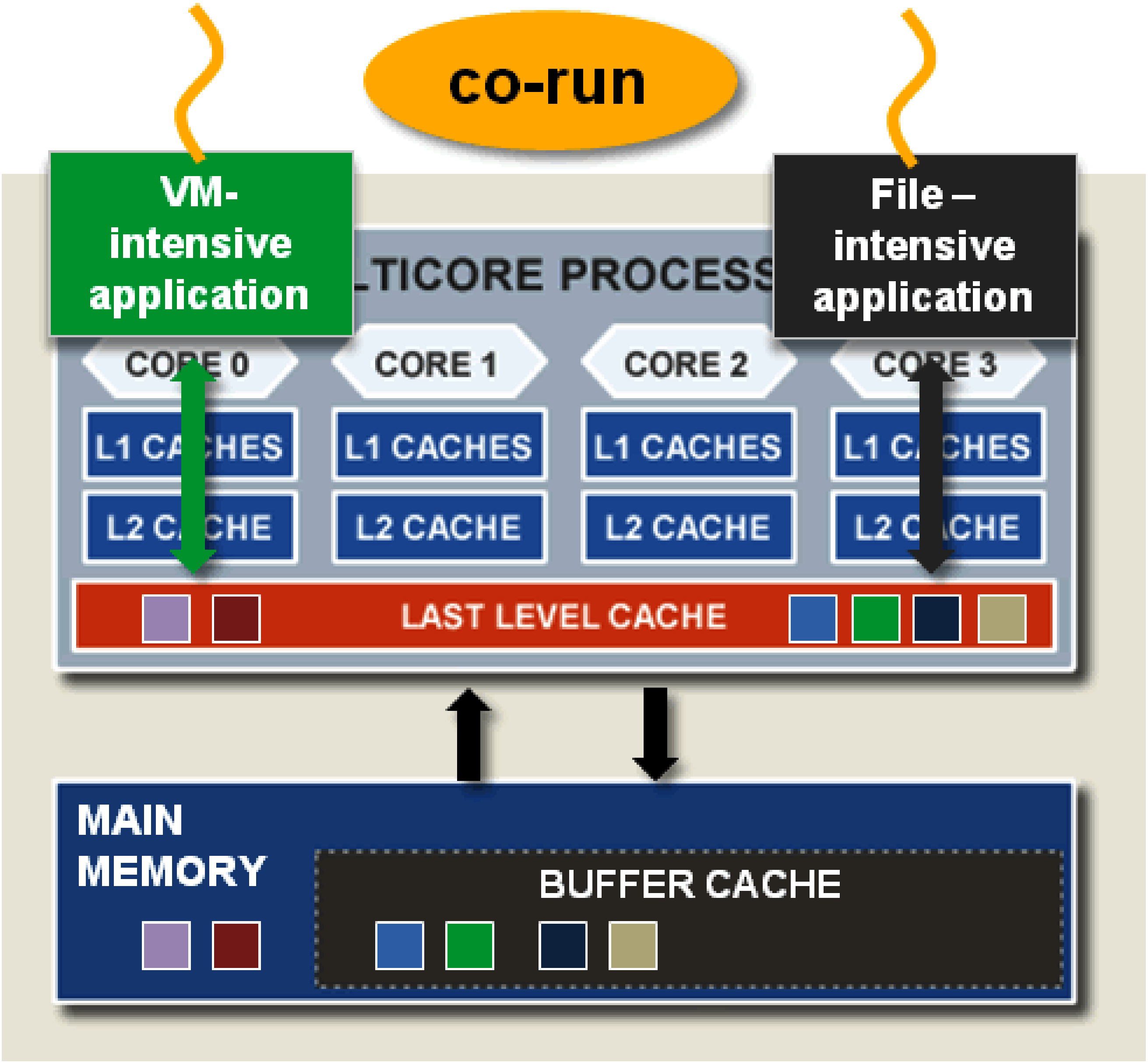
Memory Hierarchy in Multicores

- Main memory buffers
  - Virtual memory (VM) pages
    - VM management by OS
  - File blocks
    - Buffer cache management by OS
- Recently accessed data are stored in CPU caches
- **Last level caches (LLC)** are shared among multiple cores



VM- vs. File-Intensive Applications

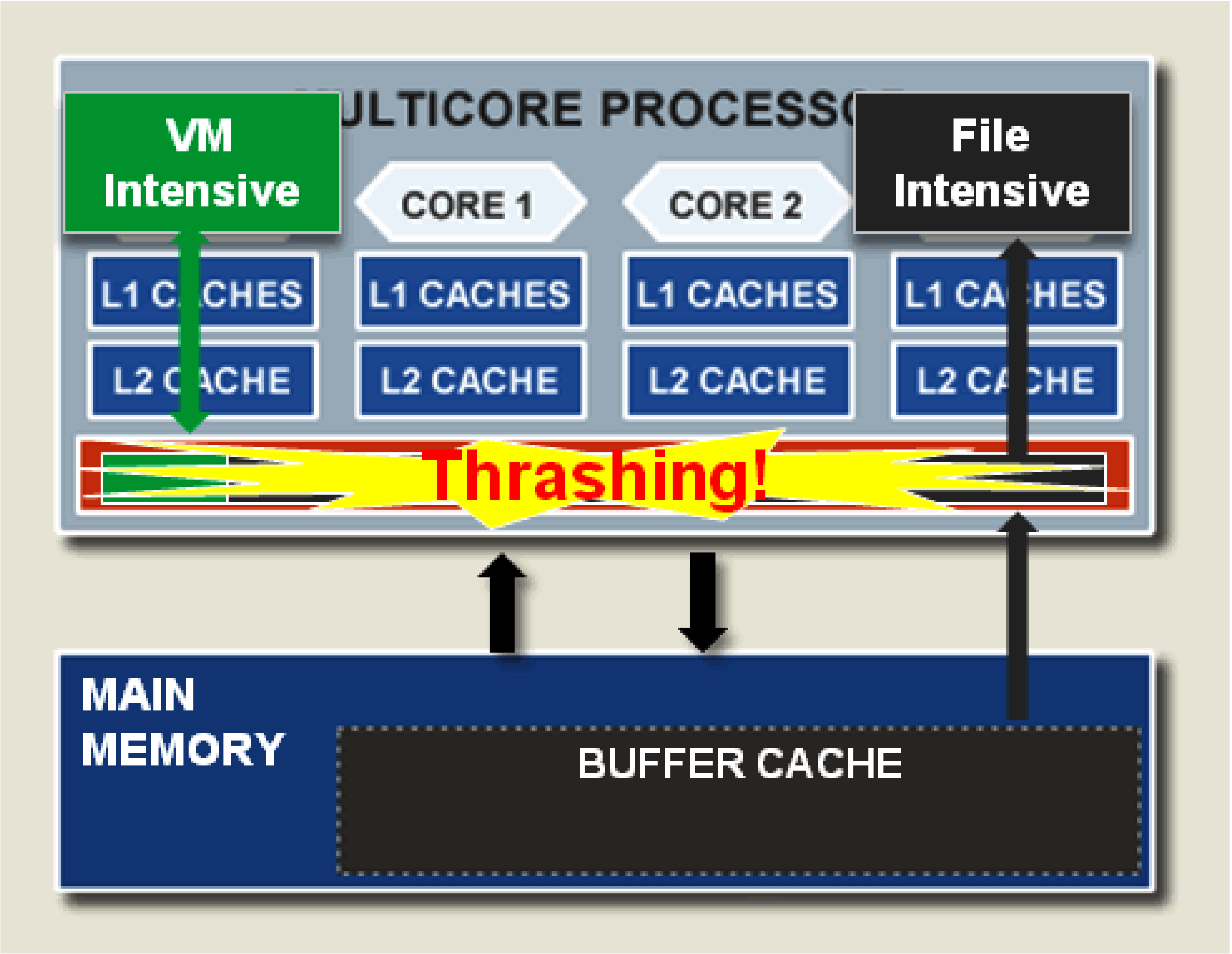
- Applications mainly accessing VM pages
  - VM intensive (scientific apps, app. servers, database queries, etc.)
- Applications mainly accessing buffered file blocks
  - File intensive (web servers, email servers, grep, tar, etc.)



Co-running VM- and file-intensive programs can cause cache thrashing and degrade performance

Cache Thrashing Happens with Multithreading

- Data in files normally have **weak locality**
  - Files are used as data storage rather than working space
- To-be-reused objects (**strong locality)** in VM are **repeatedly** replaced by objects in buffer cache (**weak locality**)



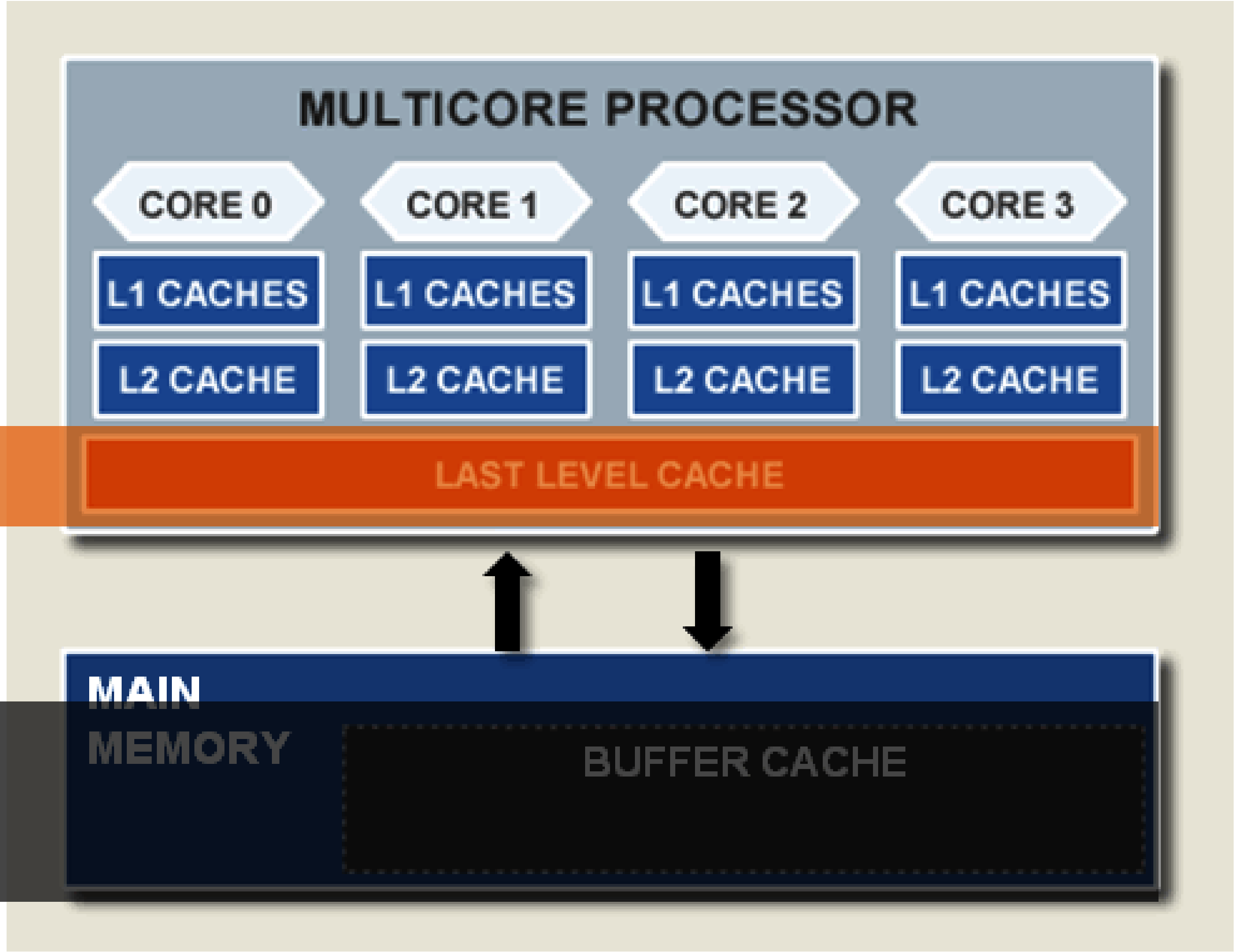
Cache Thrashing Happens with Multithreading

- Data in files normally have **weak locality**
  - Files are used a[...] rather than work[...]
- To-be-reused obj[...] [...]are[...] [...]bjects in buffer [...]ca[...]
- OS does not addr[...] issue

Last level CPU cache managed by the hardware

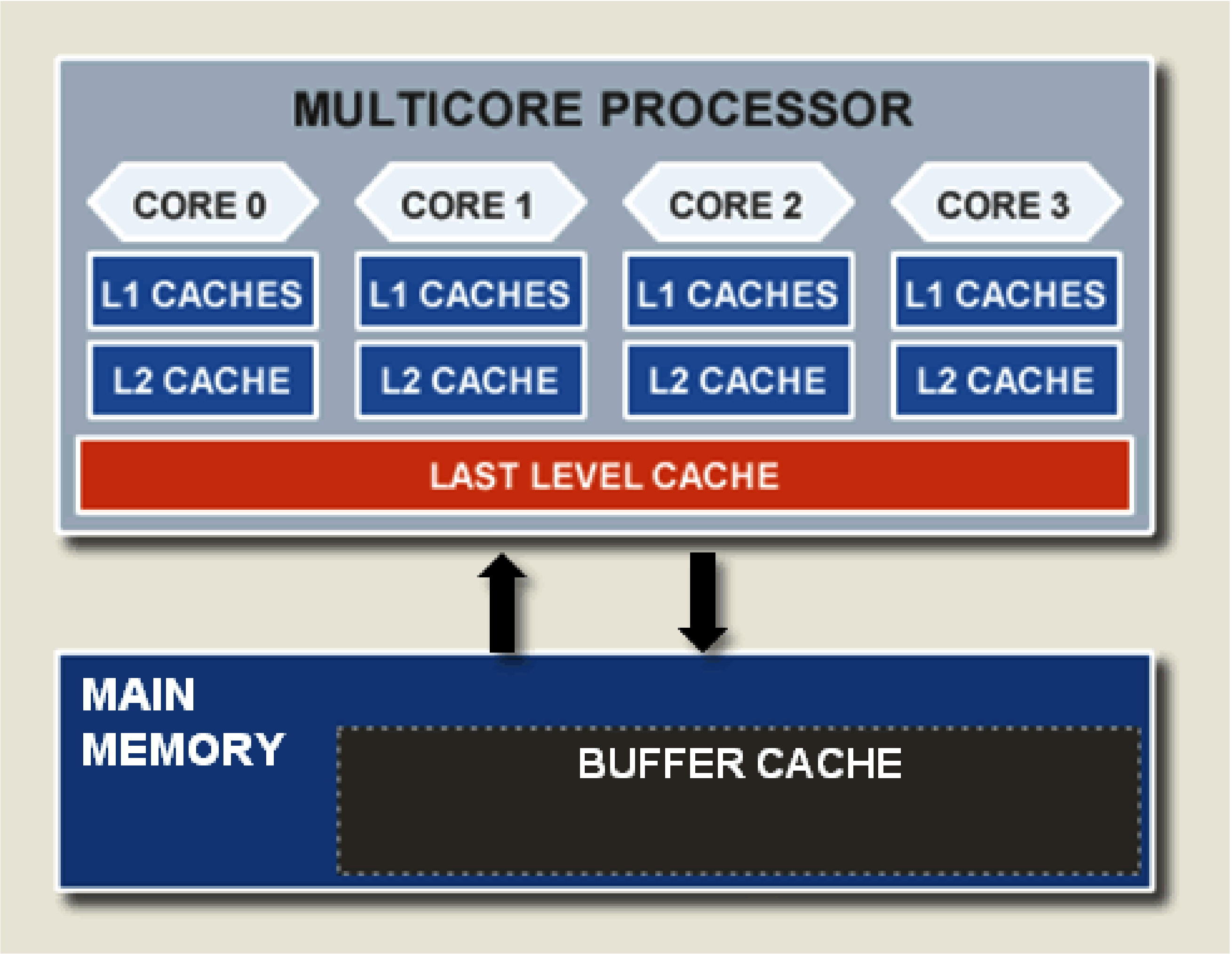
These two layers have been designed independently

Buffer cache managed by the OS



Cache Thrashing Happens with Multithreading

- Data in files normally have weak locality
  - Files are used as data storage rather than working space
- To-be-reused objects (strong locality) in VM are repeatedly replaced by objects in buffer cache (weak locality)
- OS does not address this issue
- The problem becomes worse as # of cores and diverse threads increase

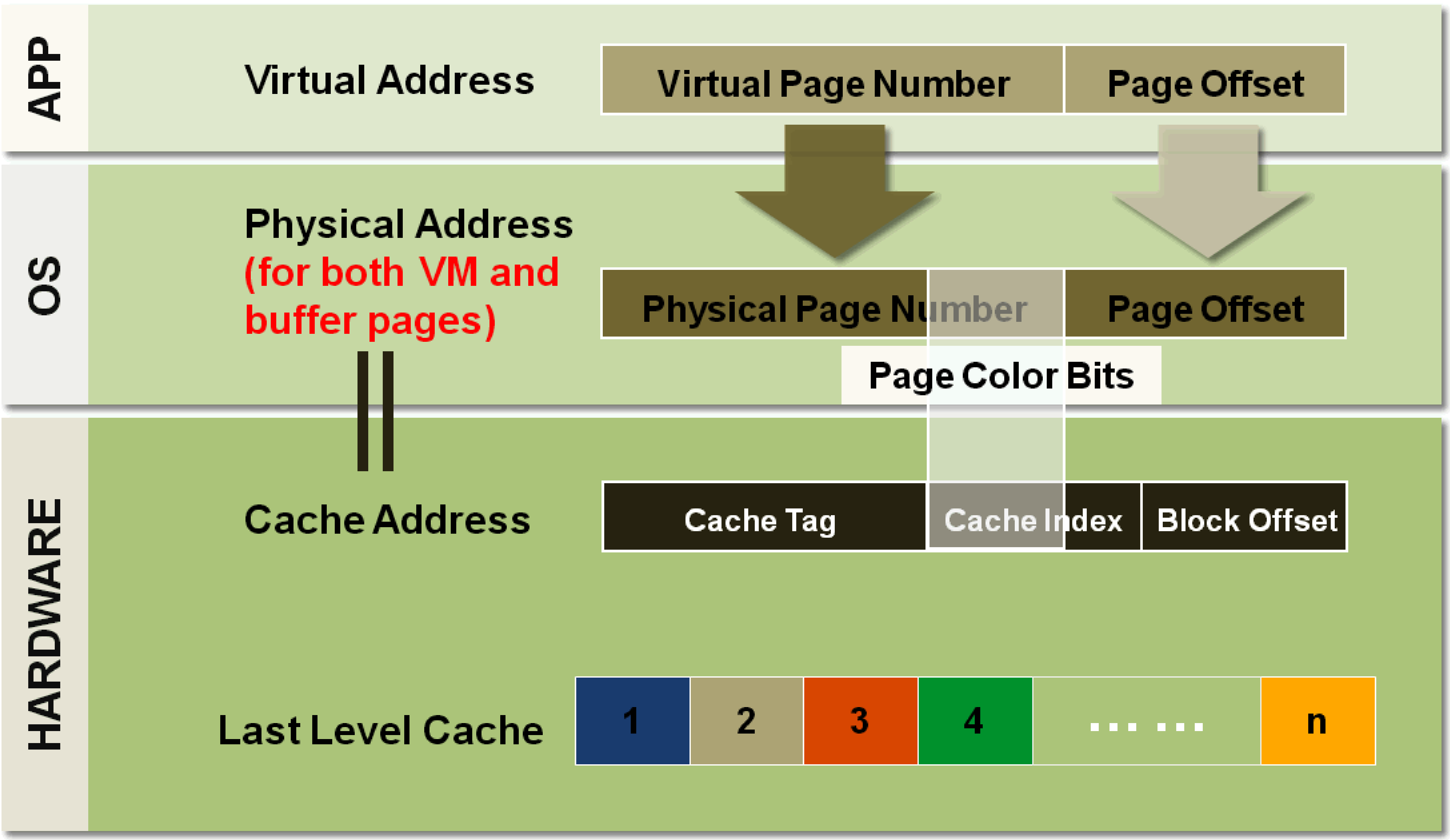


We design and implement an OS buffer management technique to prevent LLC from thrashing

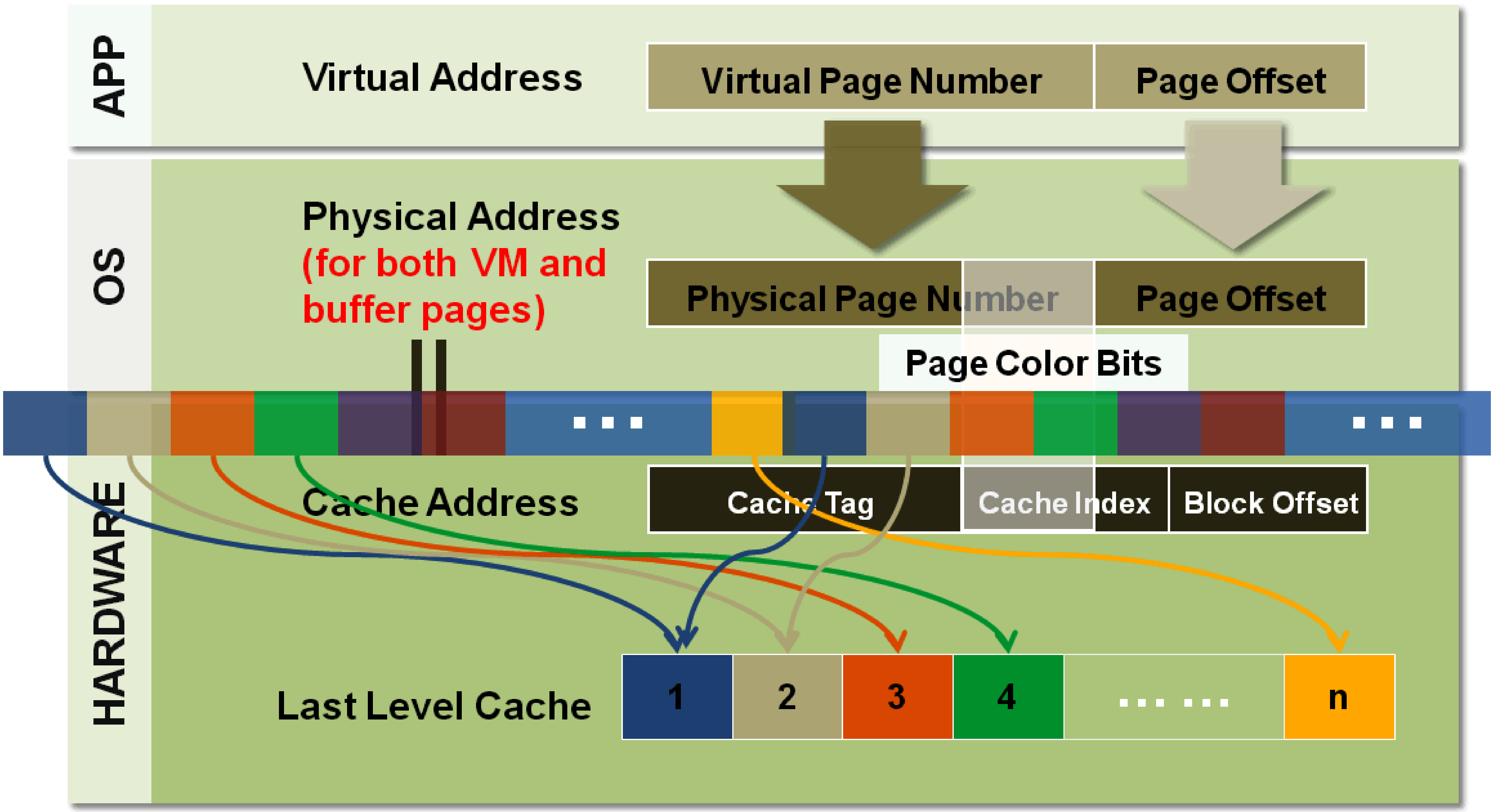
Outline

- Background and Motivation
- SRM-Buffer Design
  - Cache-memory address mapping
  - Conventional OS Buffer
  - The SRM-Buffer Design
  - Technical Challenges
- Performance Evaluation
- Conclusion

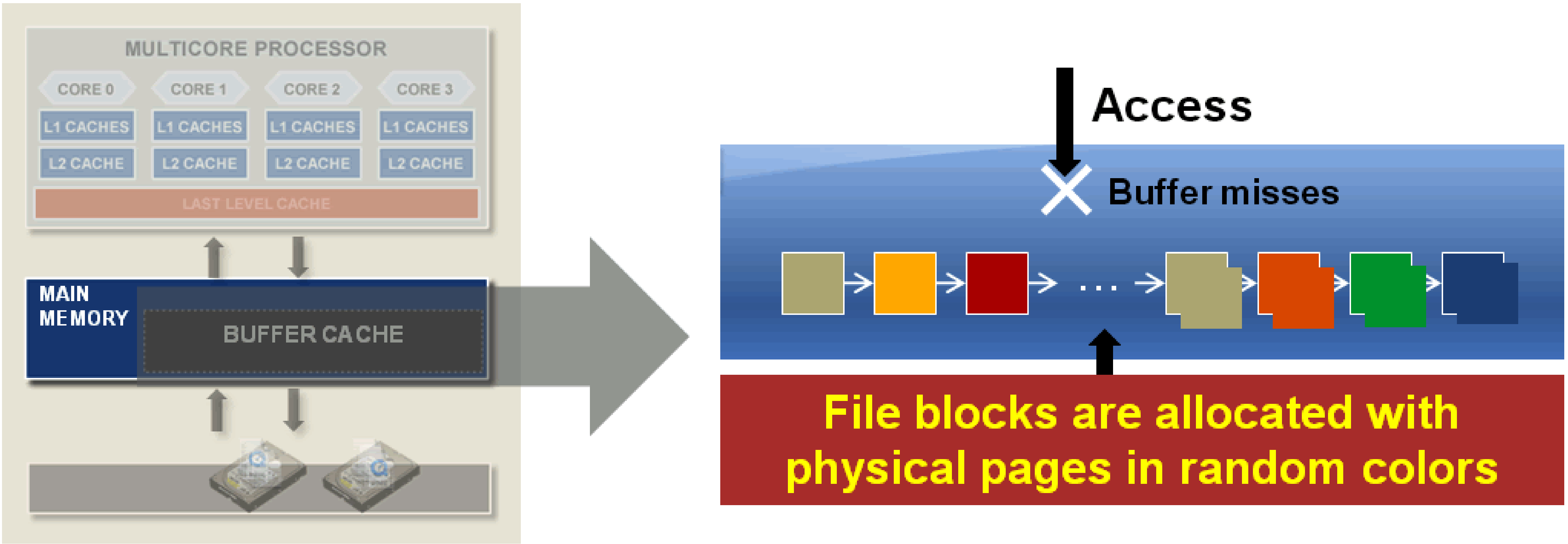
Cache-Memory Address Mapping



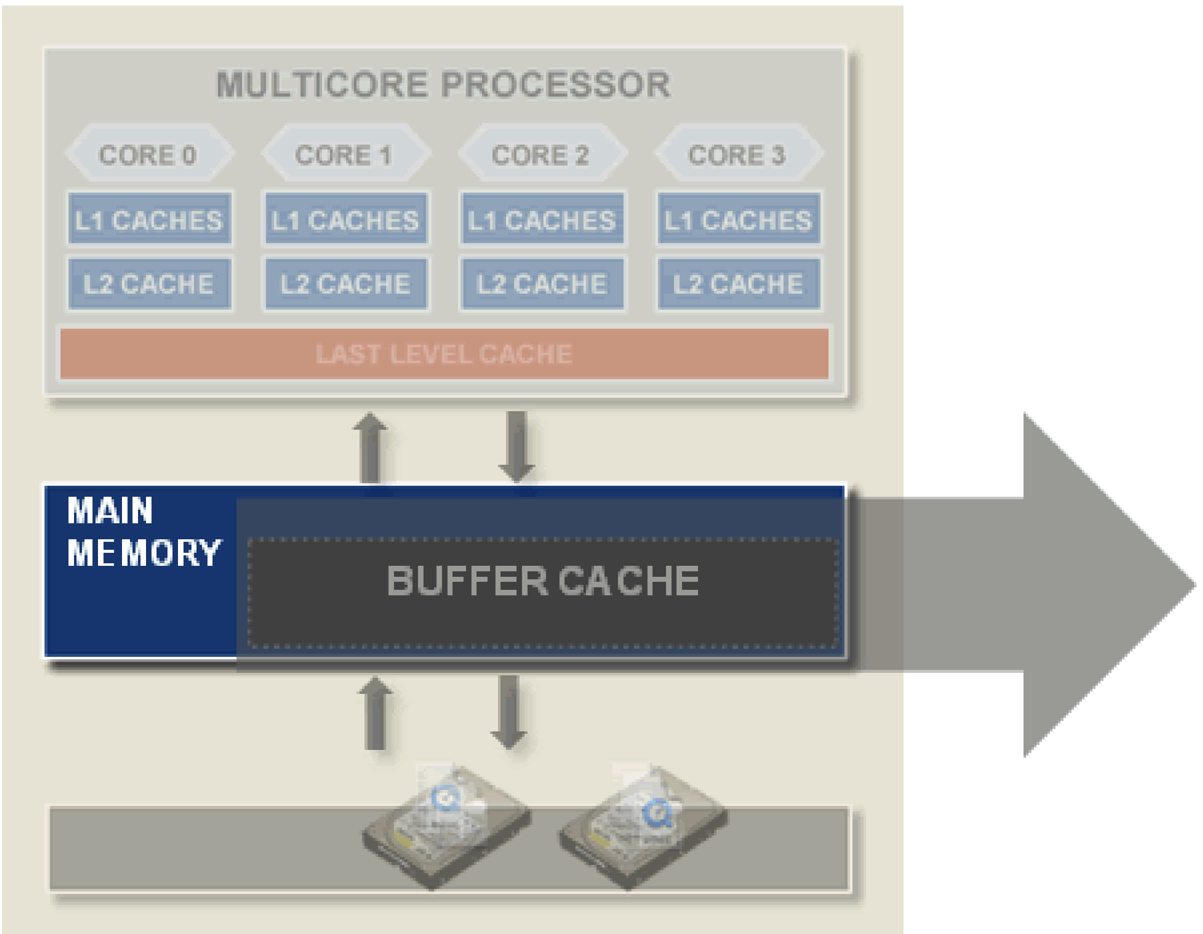
Cache-Memory Address Mapping

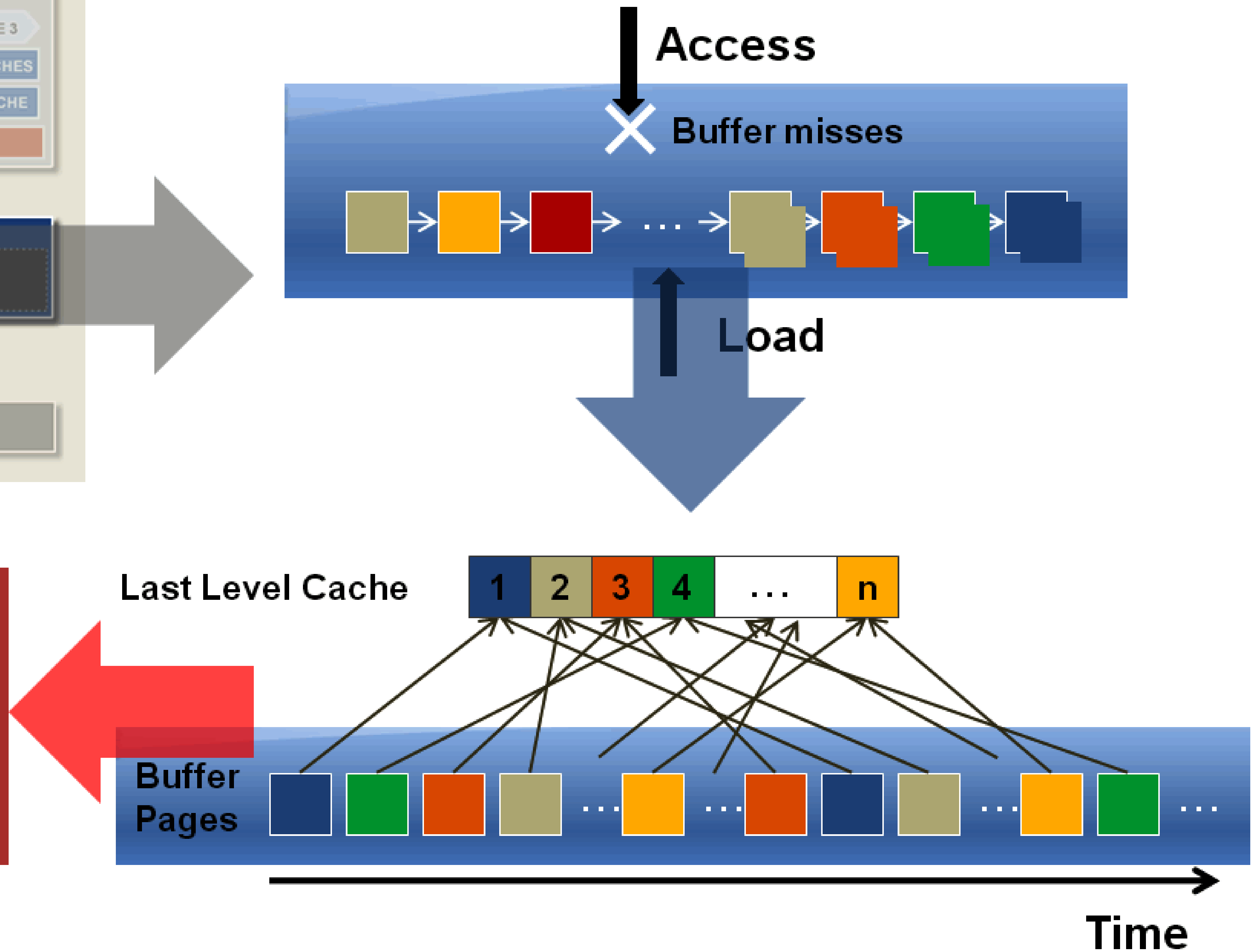


Conventional OS Buffer Cache



Conventional OS Buffer Cache



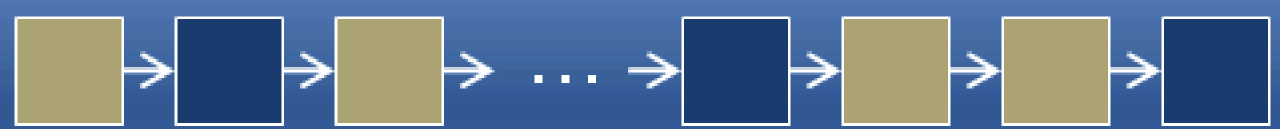


Accessing OS buffer cache pollutes LLC very quickly, causing severe cache thrashing.

Inability of "Fixed Cache Region"

Give a small number of dedicated page colors to buffer cache

Buffer Pages (e.g., using only two colors)



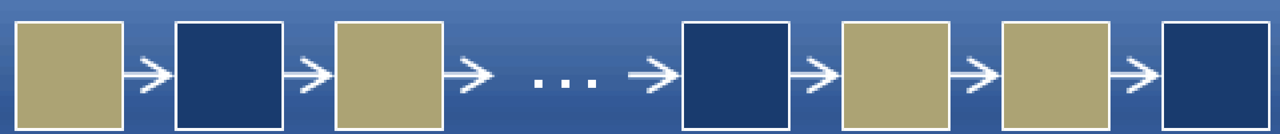
4GB / 64 colors = 64MB per color

Max buffer cache size: 128MB
v.s.
Desired buffer cache size: >2GB

Inability of "Fixed Cache Region"

Give a small number of dedicated page colors to buffer cache

Buffer Pages (e.g., using only two colors)



4GB / 64 color

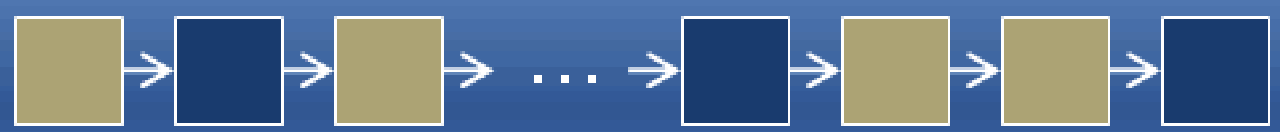
Max buffer ca
v.s.
Desired buffer

High page miss ratio for buffer data due to limited buffer size

Inability of "Fixed Cache Region"

Give a small number of dedicated page colors to buffer cache

Buffer Pages (e.g., using only two colors)



4GB / 64 color

Max buffer ca
v.s.
Desired buffer

High page miss ratio for buffer data due to limited buffer size

Give enough dedicated page colors to buffer cache

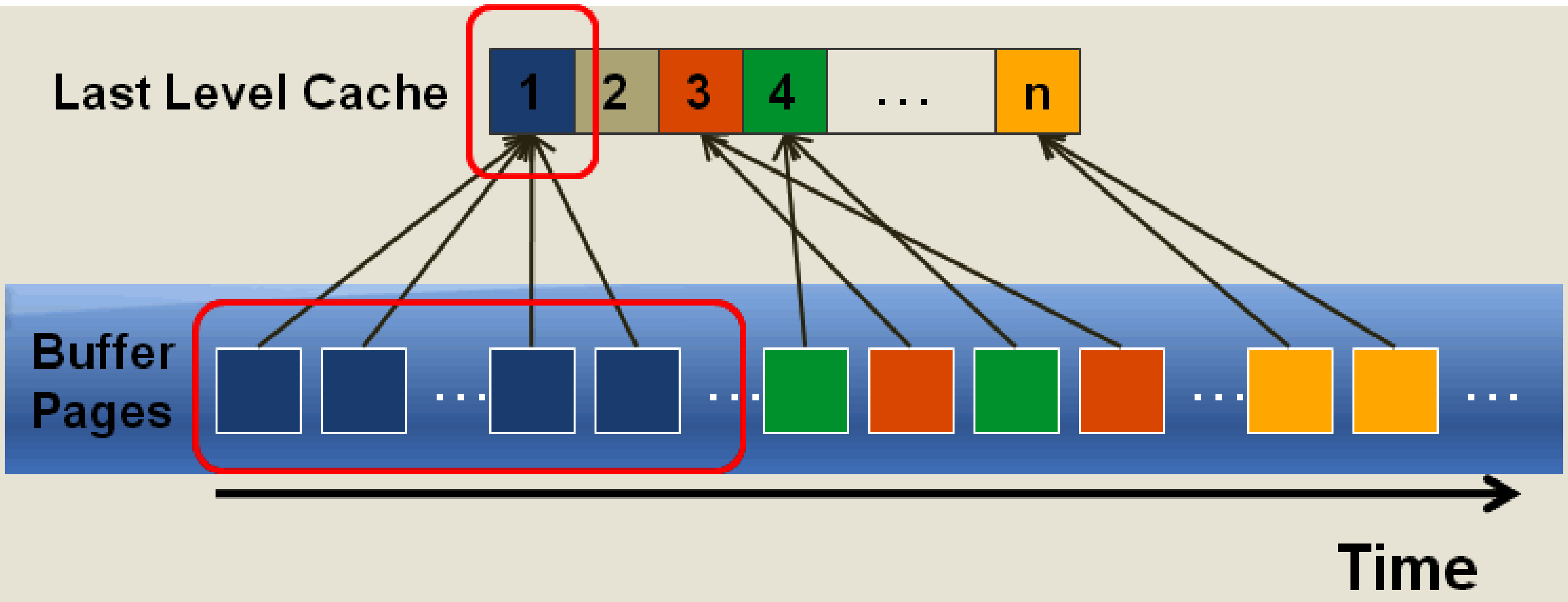
A large portion of LLC reserved for buffer cache

The available cache space for other purposes, especially VM data, will become seriously limited

High LLC miss ratio for VM data due to limited cache space

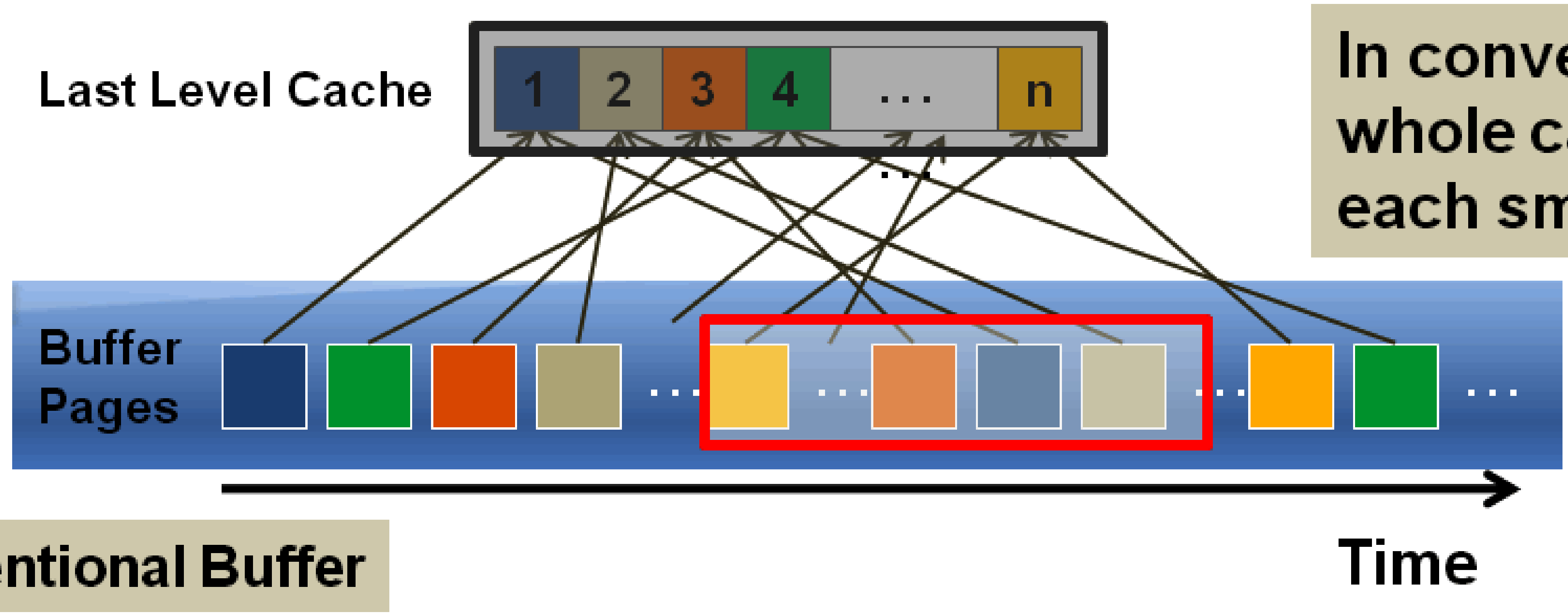
Our Objective: to coordinate VM and buffer cache demands and limit pollution in LLC

Selected Region Mapping Buffer



- Identify **sequences** (streams of file blocks)
- Blocks in the same sequence are mapped to the same cache region **(same color**) when they are loaded into OS buffer
- Change colors **dynamically** for different sequences of blocks

The Benefits of SRM-Buffer

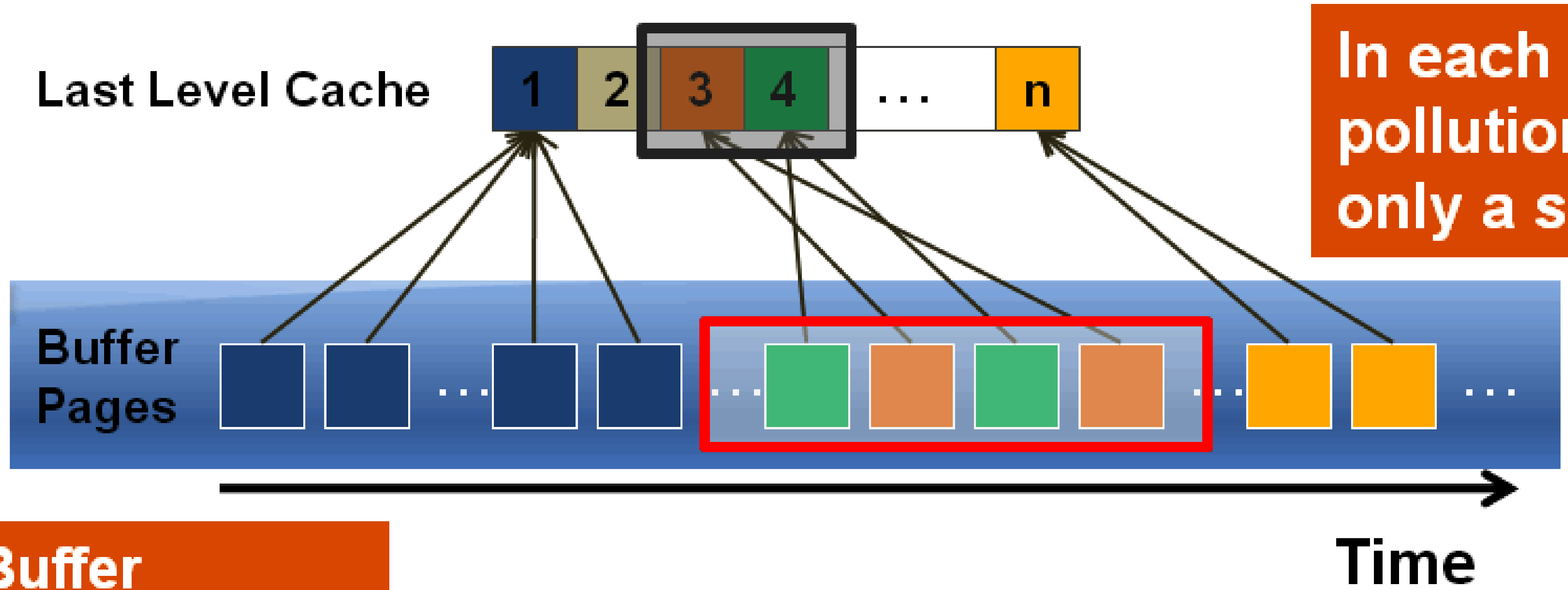


In conventional buffer, the whole cache is polluted in each small period of time

Conventional Buffer

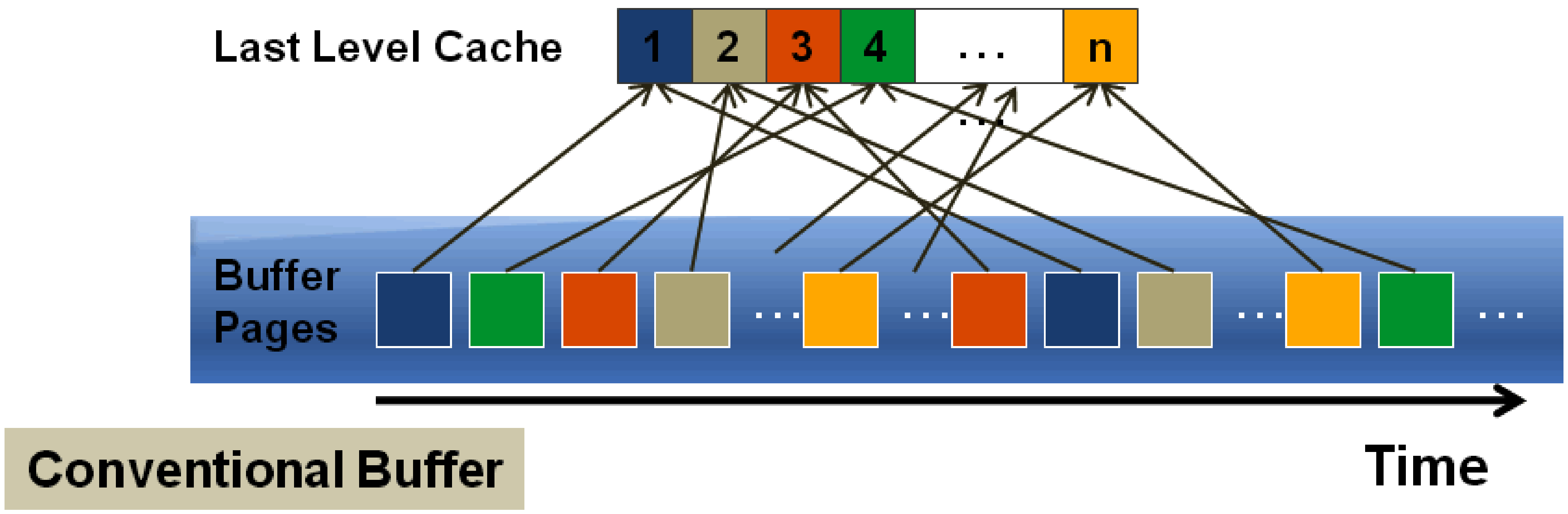
Benefit 1

In each small period, pollution is constrained to only a small cache region



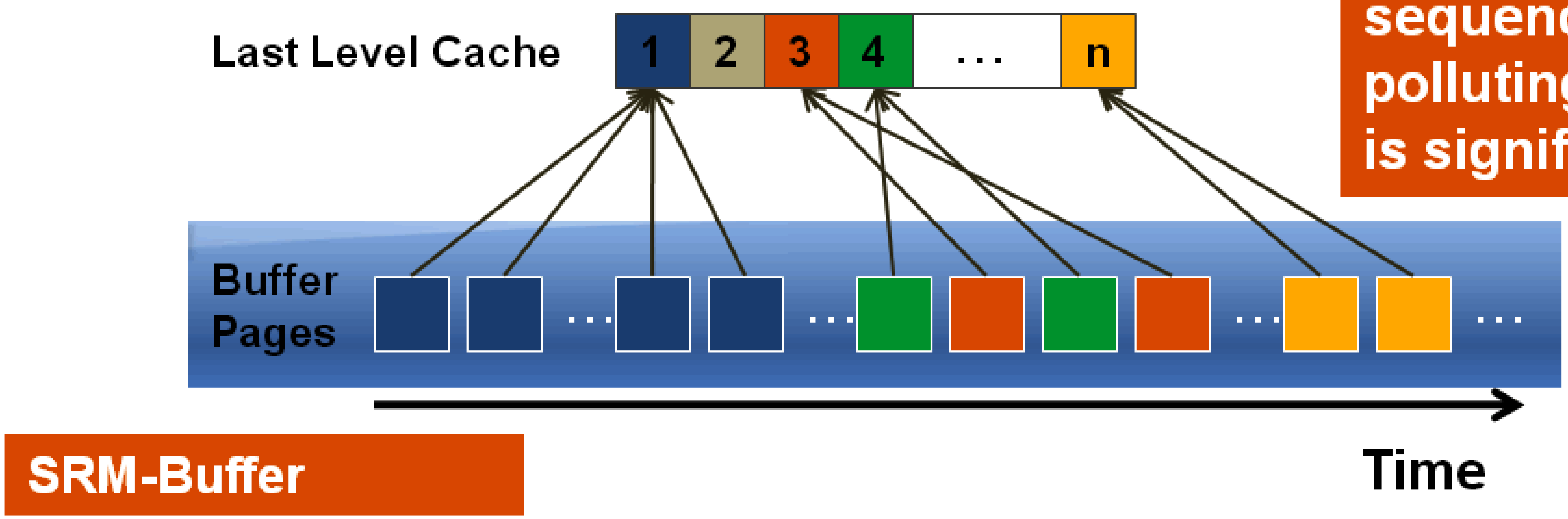
SRM-Buffer

The Benefits of SRM-Buffer

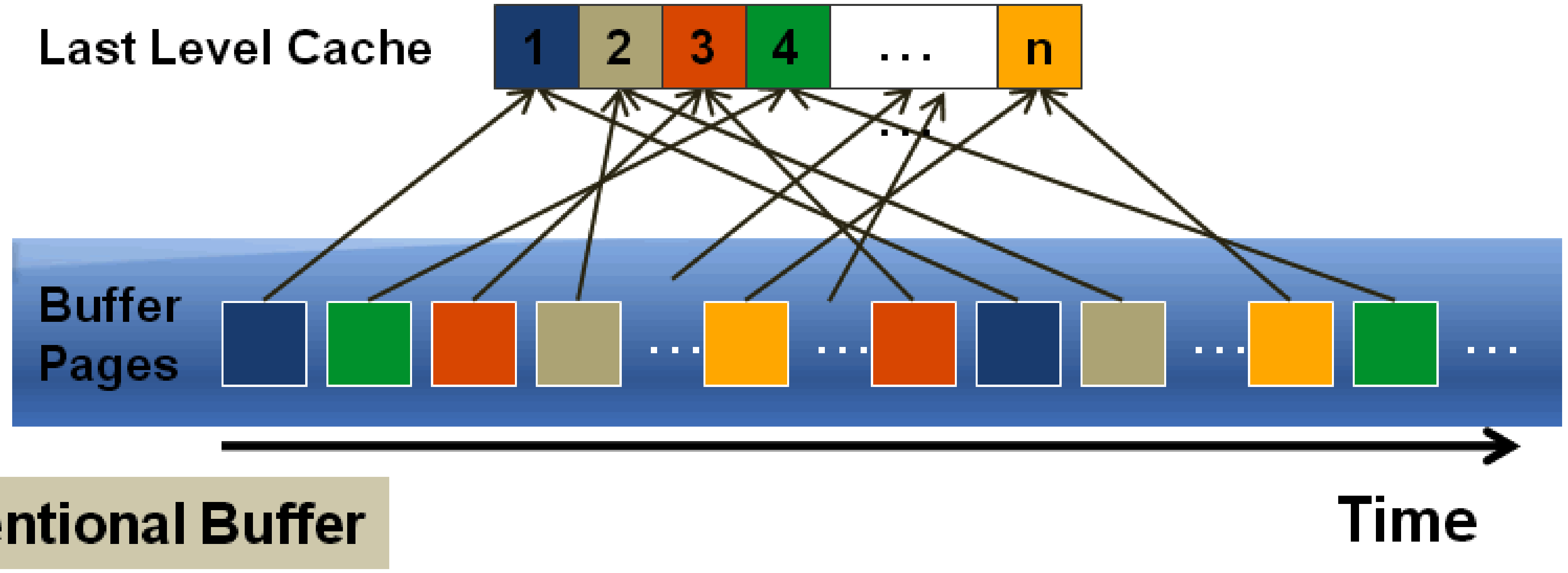


Benefit 2

For a long access sequence, the speed of polluting the whole cache is significantly lowered



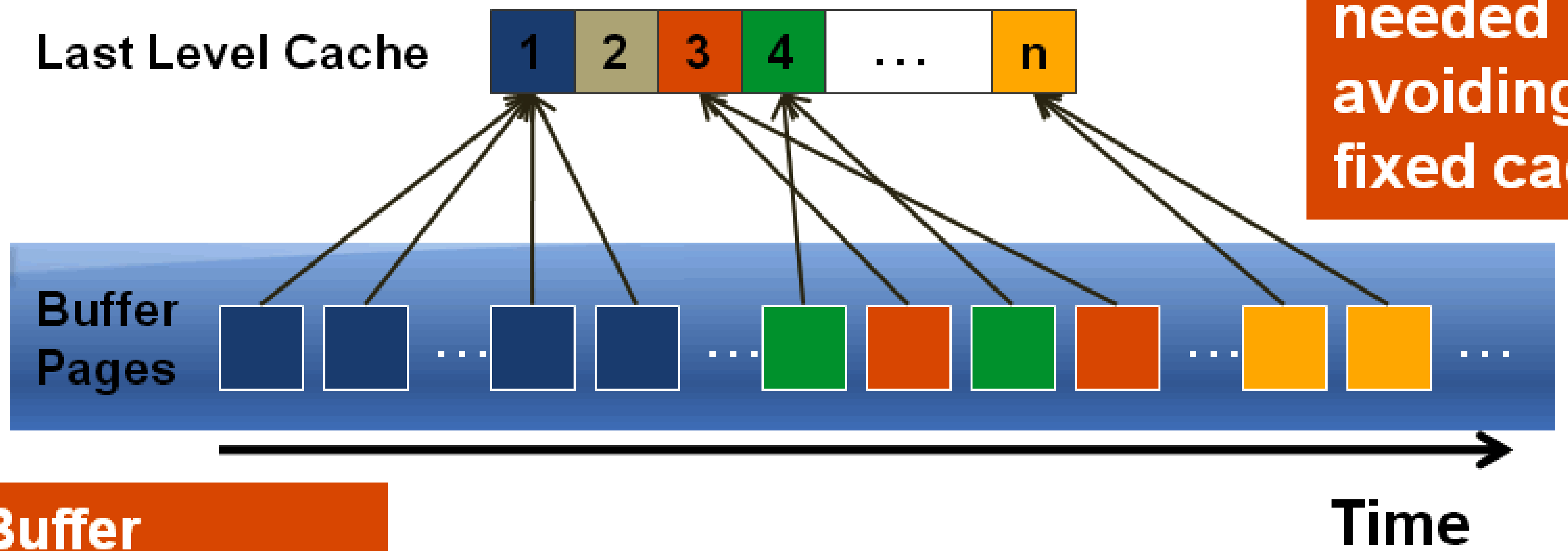
The Benefits of SRM-Buffer



Conventional Buffer

Benefit 3

No dedicated colors are needed in SRM-buffer, thus avoiding all the limits of fixed cache region



SRM-Buffer

Issue 1: Determine Block Sequences

- **How to decide which blocks are to be accessed together (in the same sequence)?**
  - **Same-File Heuristic**: blocks in the same file are usually accessed together
  - **Same-Application Heuristic**: blocks consecutively loaded by the same process are usually accessed together

Issue 2: A Constraint Optimization

To minimize last level cache thrashing

subject to retaining

VM Management Performance

– Uniform cache regions for VM pages

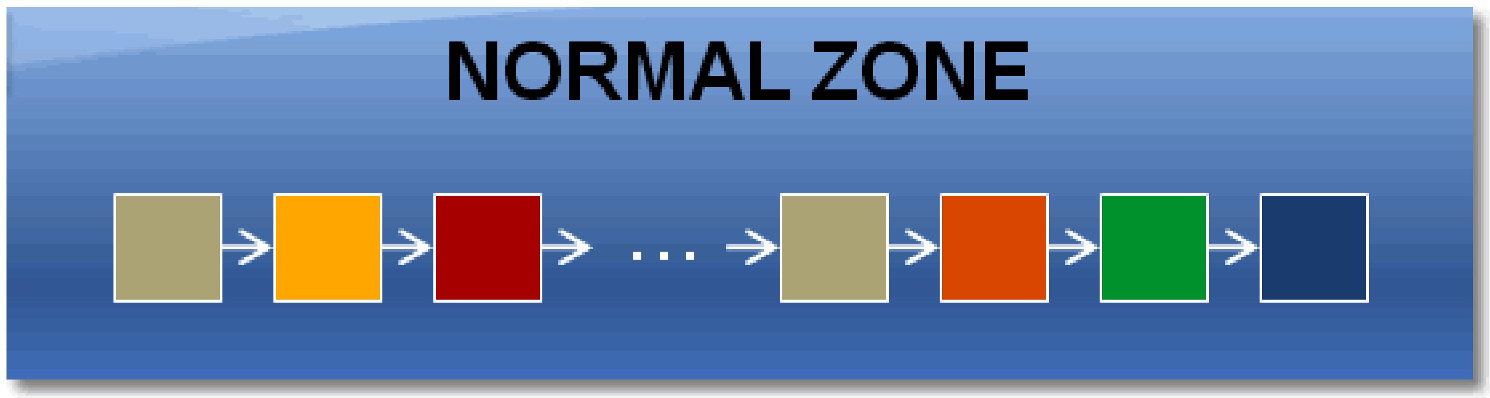
and

Page Cache Performance

– High page hit ratio

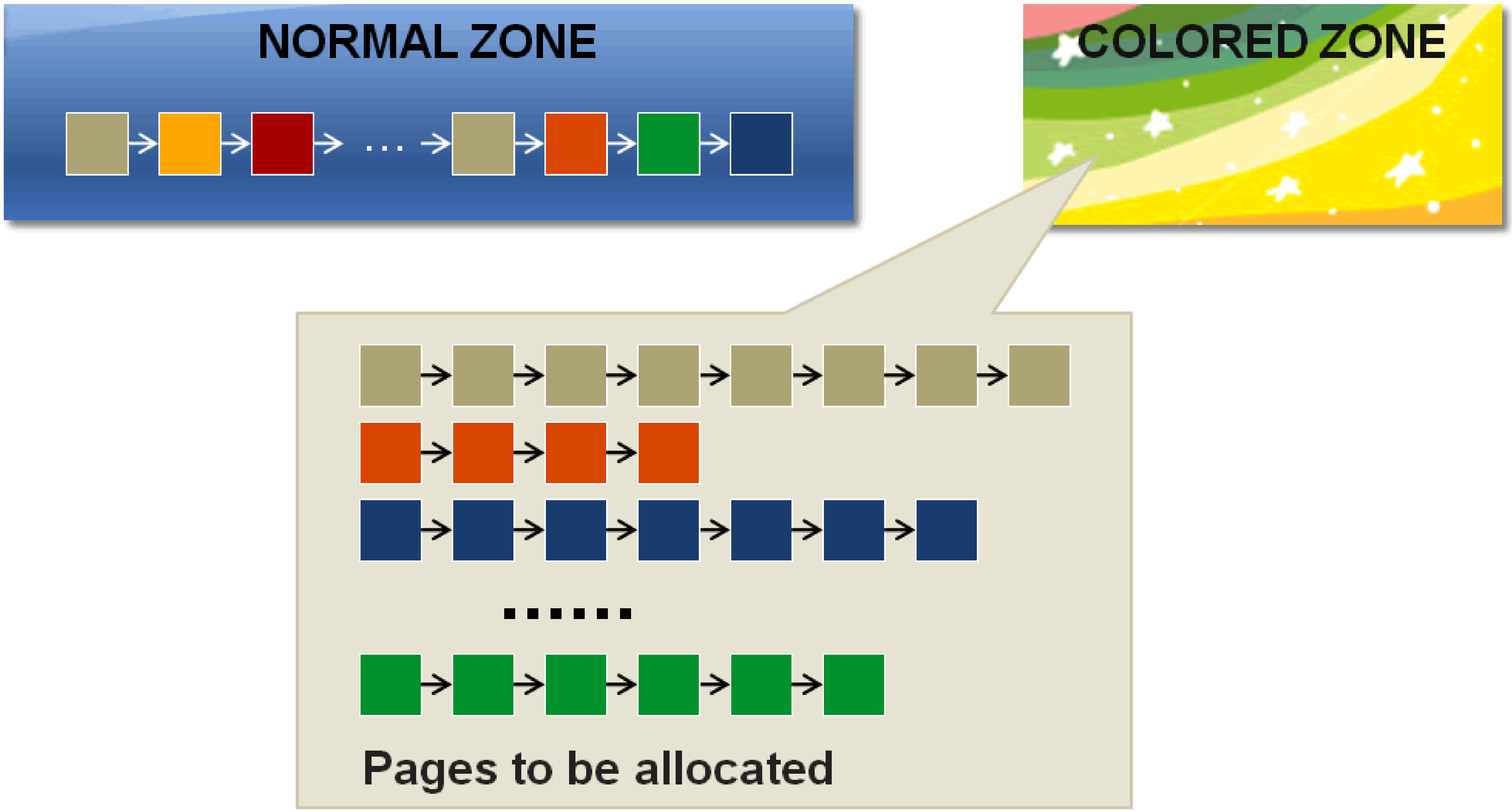
How to achieve the objective with the constraints?

SRM-Buffer Structure & Operations



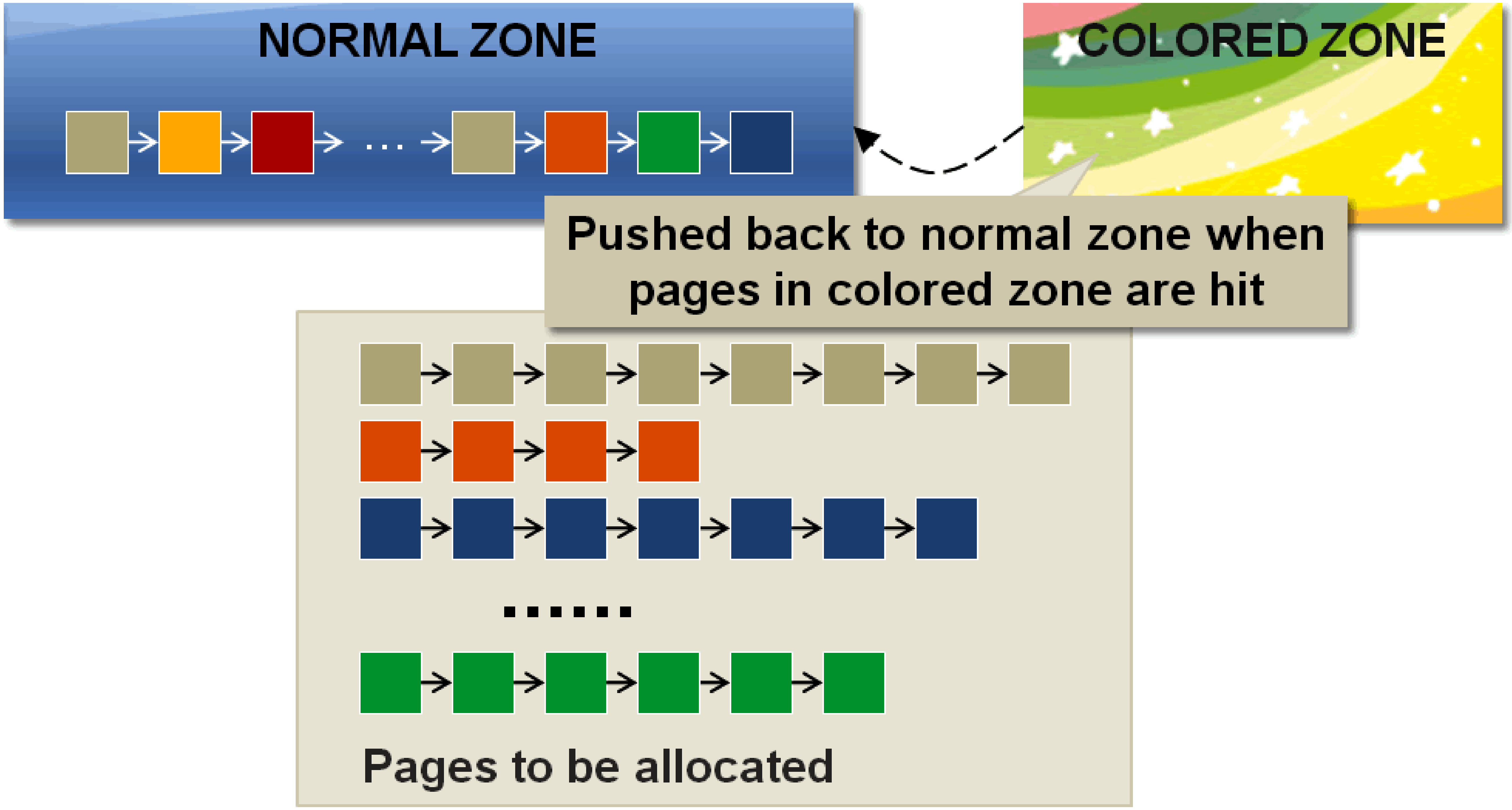
Normal Zone: managed by conventional OS replacement; containing LRU page list

SRM-Buffer Structure & Operations

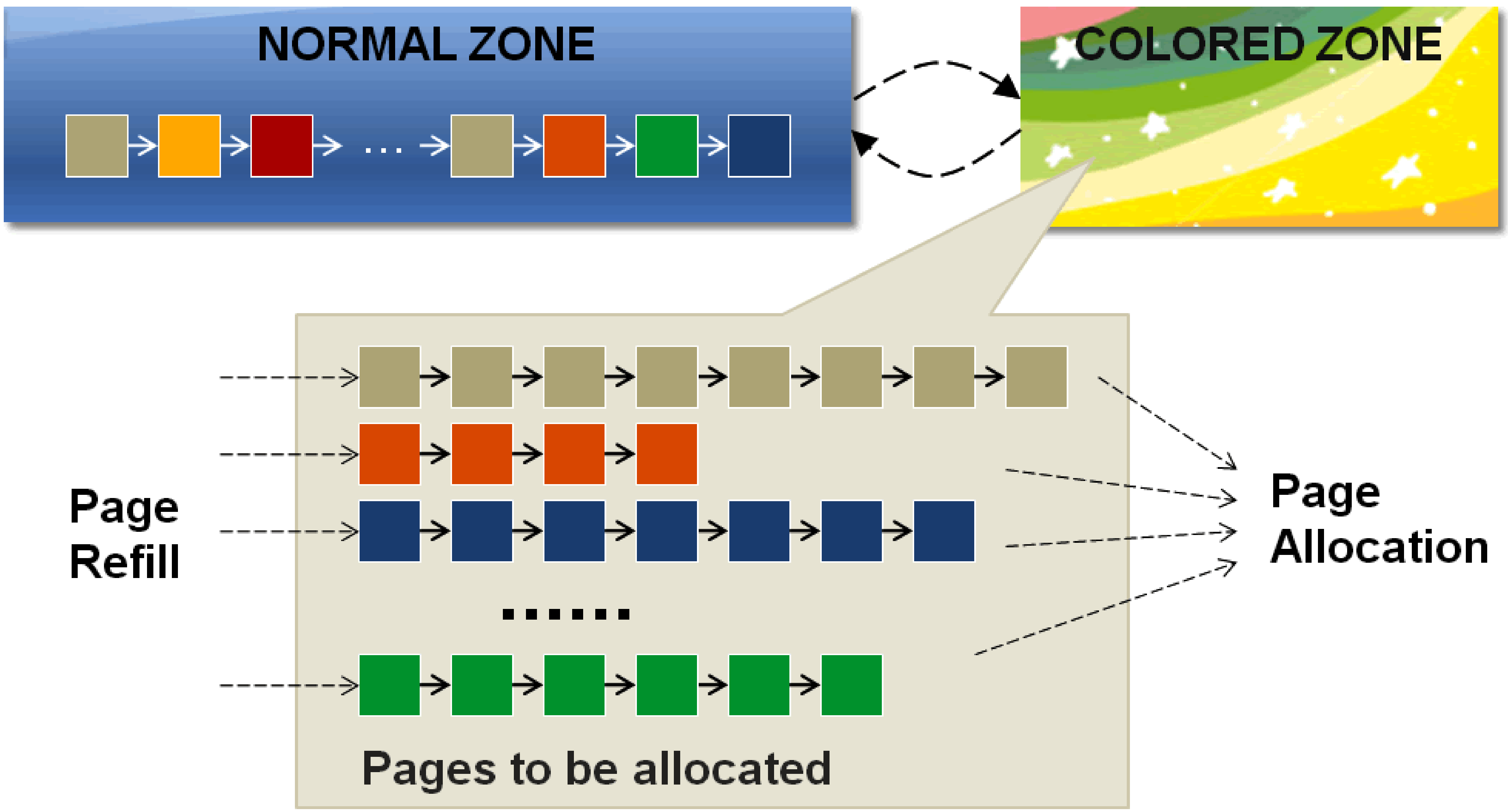


Colored Zone: Free pages and small amount of inactive pages (e.g. less than 1% of non-free pages) when system is short of free pages

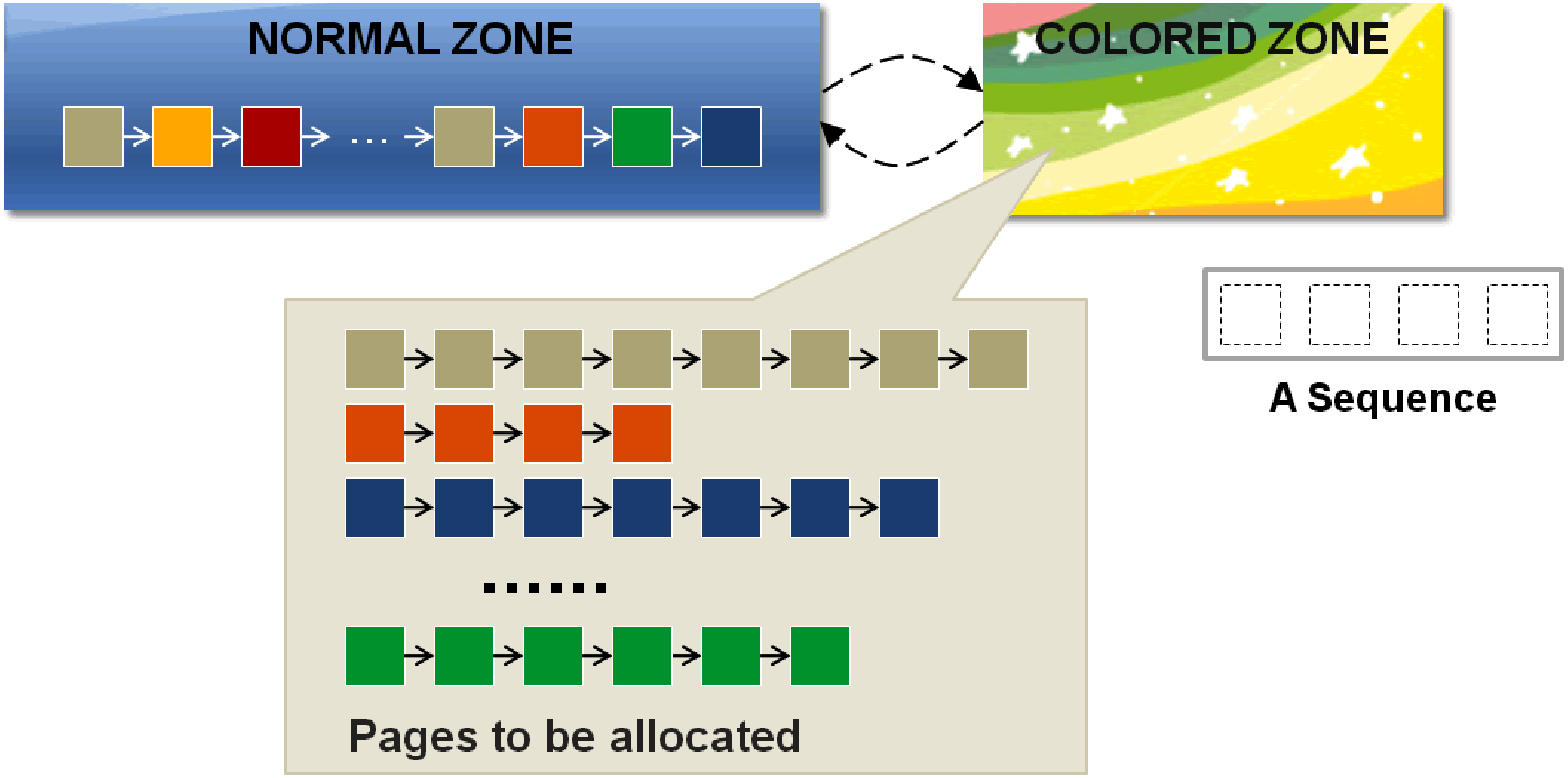
SRM-Buffer Structure & Operations



SRM-Buffer Structure & Operations

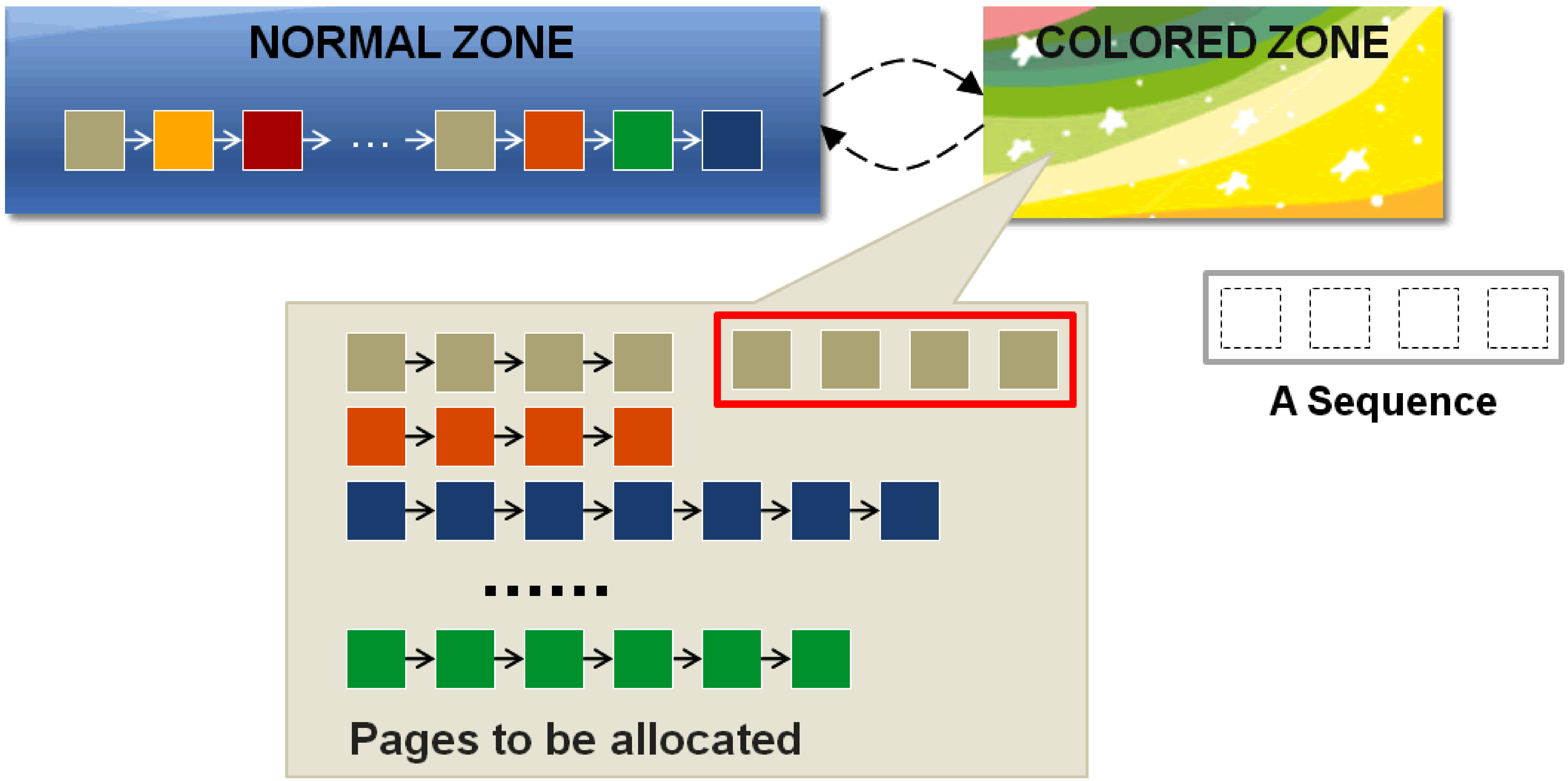


SRM-Buffer Structure & Operations



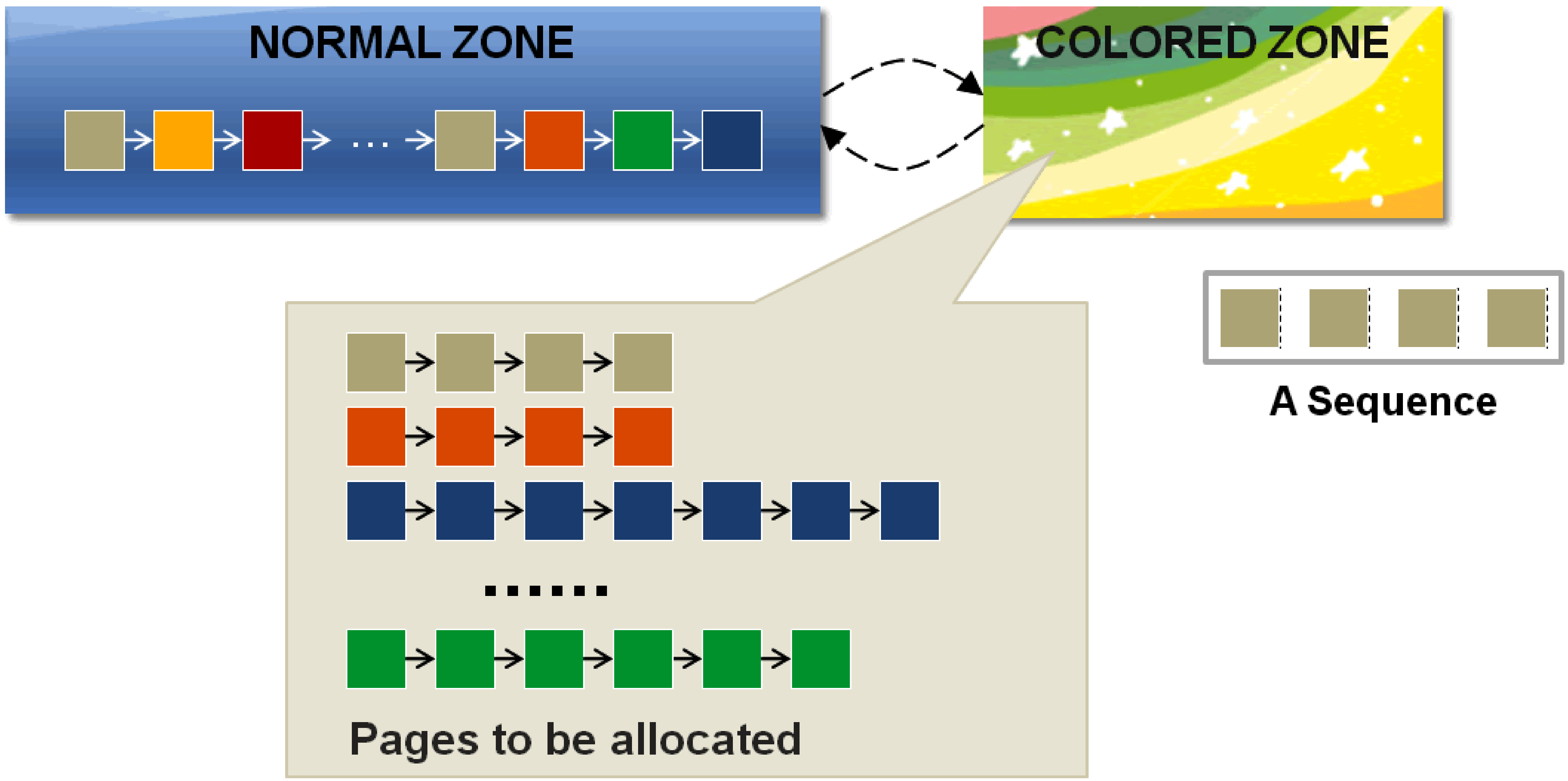
1. On buffer misses, allocate physical pages in a single color to file blocks loaded in a sequence

SRM-Buffer Structure & Operations



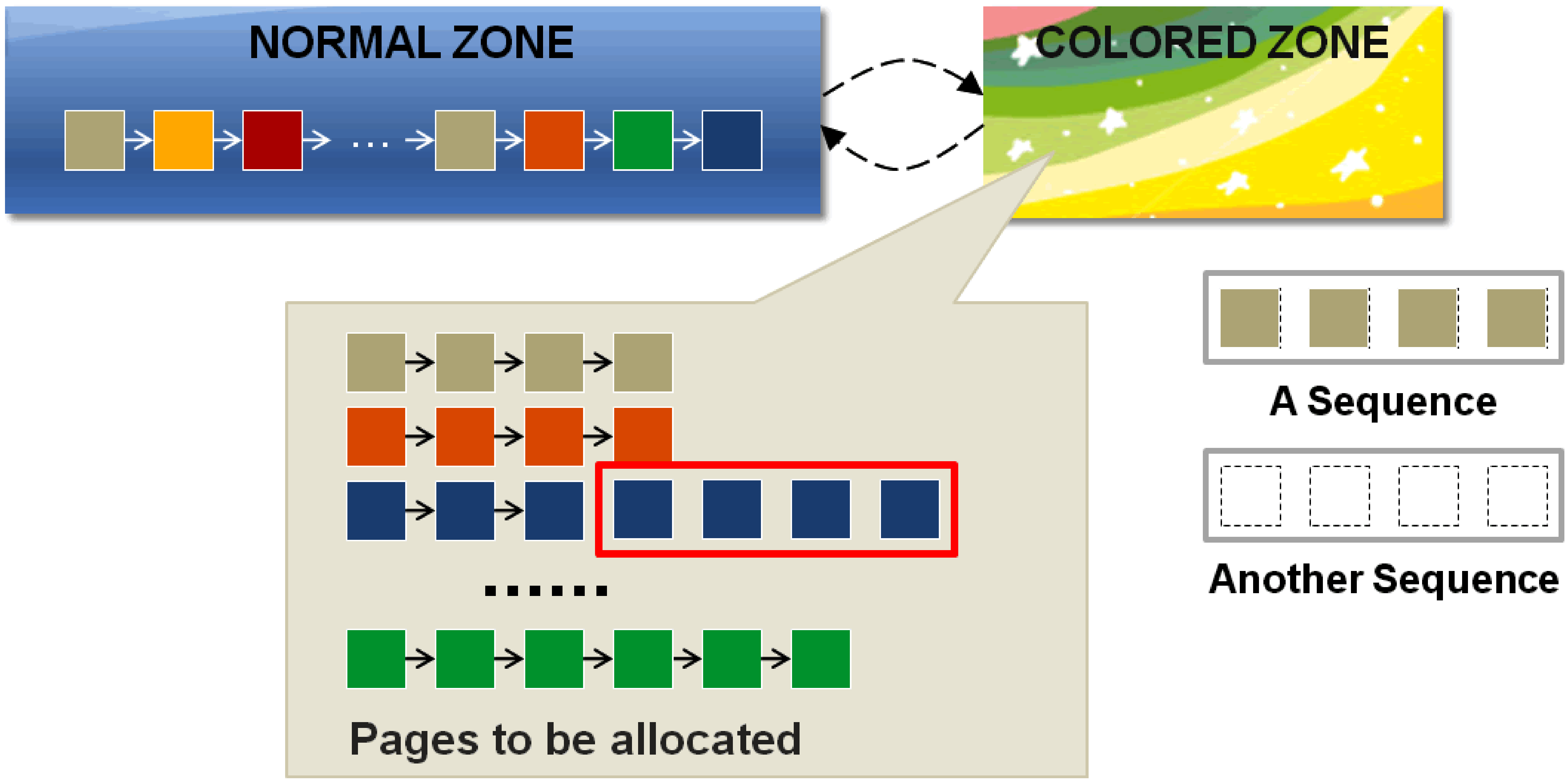
1. On **buffer misses**, allocate physical pages in a single color to file blocks loaded in a **sequence**

SRM-Buffer Structure & Operations



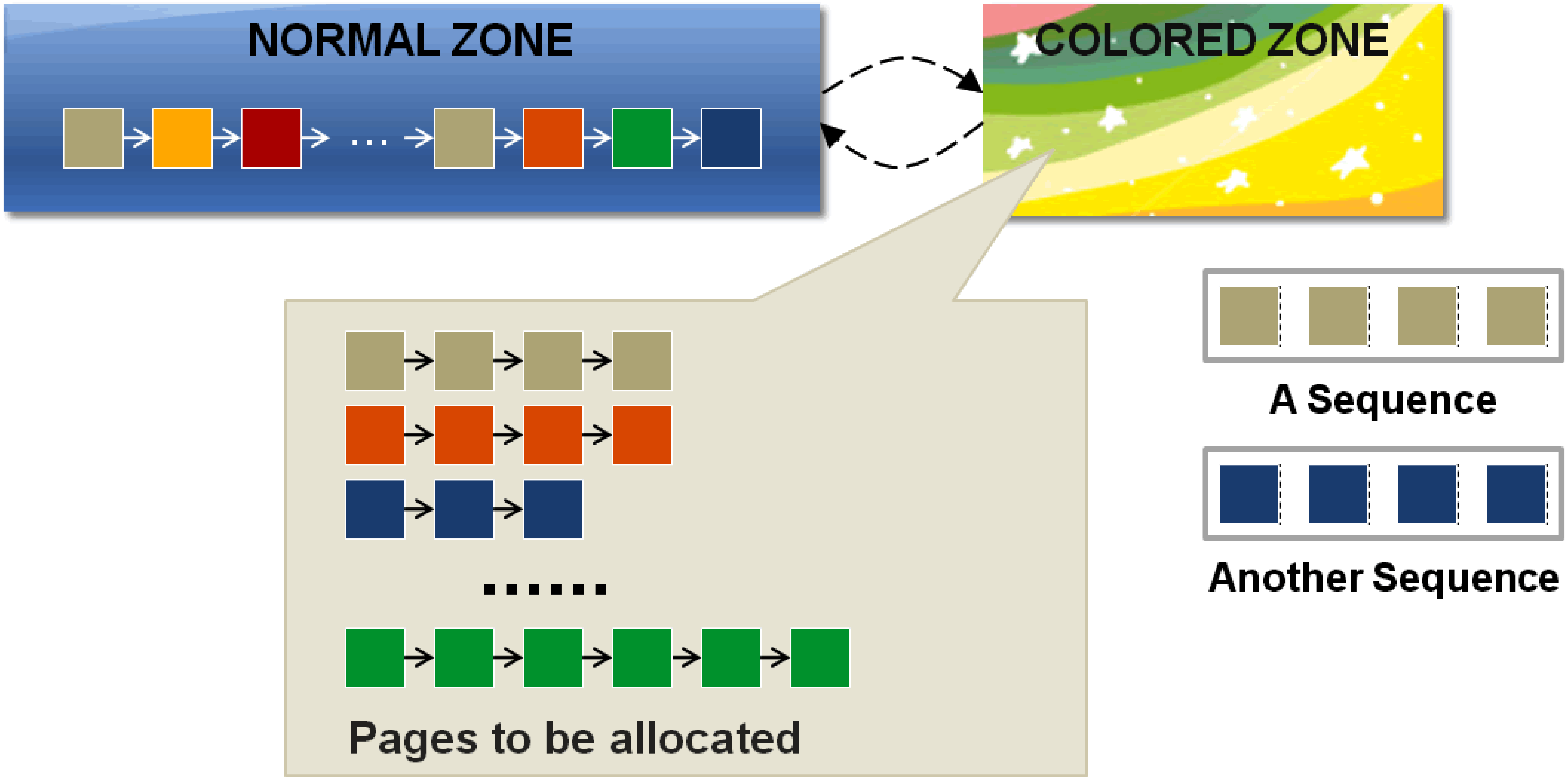
1. On buffer misses, allocate physical pages in a single color to file blocks loaded in a sequence

SRM-Buffer Structure & Operations



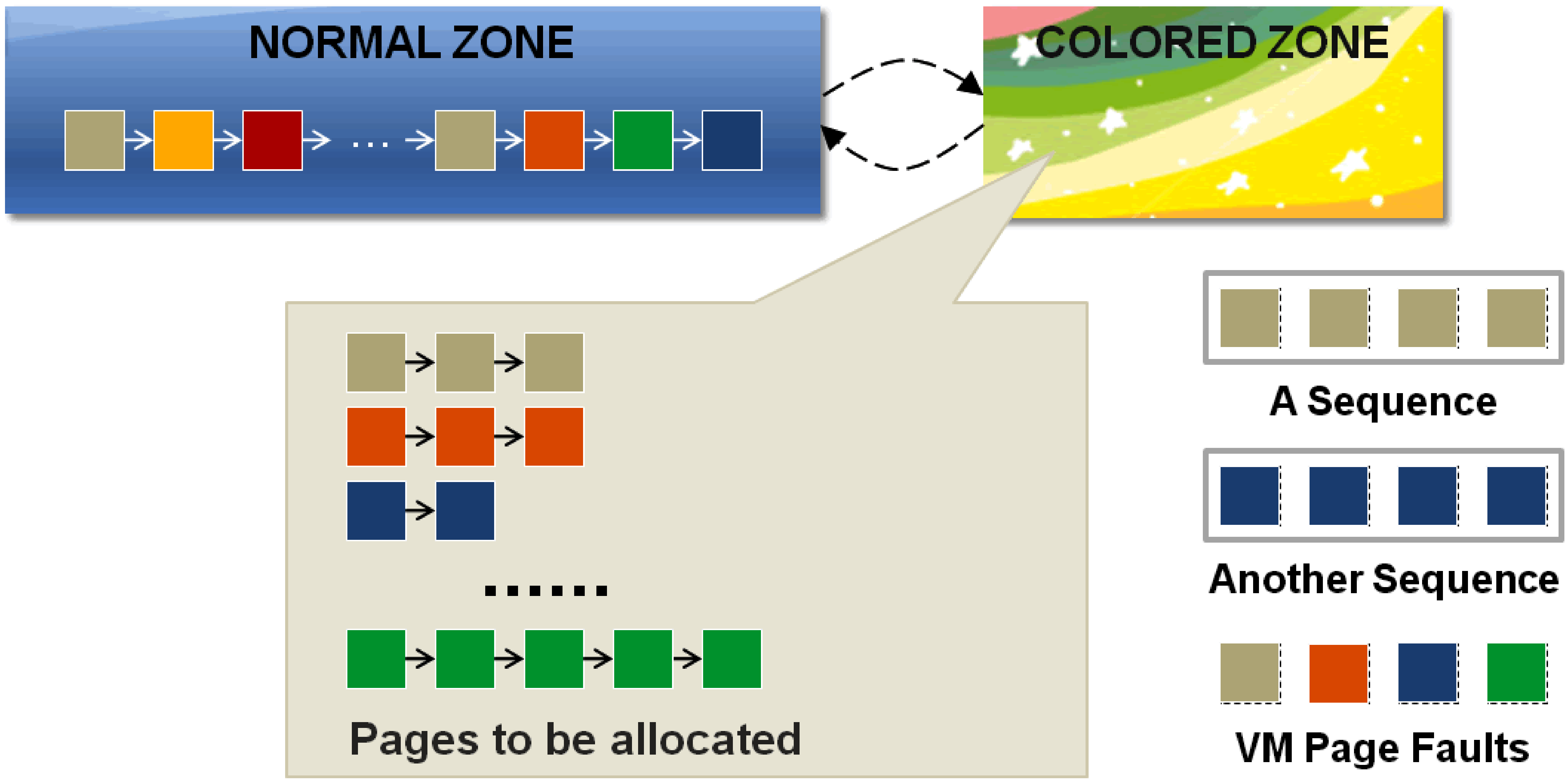
2. Change colors dynamically after the number of pages allocated in a given color reaches a threshold

SRM-Buffer Structure & Operations



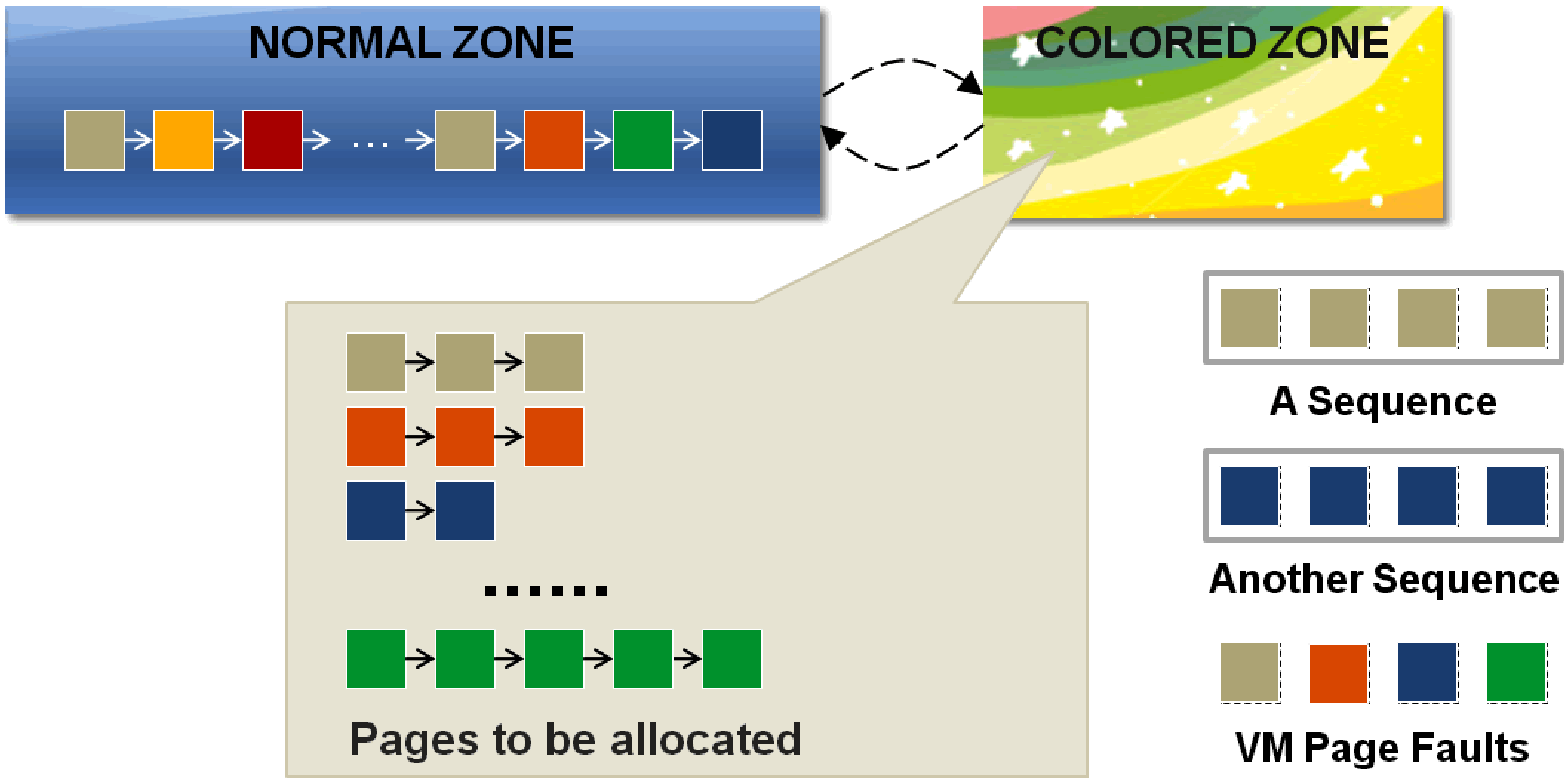
2. Change colors dynamically after the number of pages allocated in a given color reaches a threshold

SRM-Buffer Structure & Operations



3. On VM page faults, uniformly allocate pages in different lists to hold the virtual pages

SRM-Buffer Structure & Operations



4. **Page hit ratio is retained** automatically by the Normal Zone

Performance Evaluation

- Prototype implementation in Linux kernel 2.6.30
- Sequence length threshold: 256 pages
- Experiment setup
  - **Dell PowerEdge 1900 workstation**
    - **Two 2.66GHz quad-core Xeon X5355 processors**
    - **Each pair of two cores sharing a 4MiB L2 cache**
    - **16GiB memory**
  - **Dell Precision T1500 workstation**
    - Intel Core i7 860 processor
    - Four cores sharing an **8MiB L3 cache**
    - 8GiB memory

We will mainly present the results on PowerEdge 1900

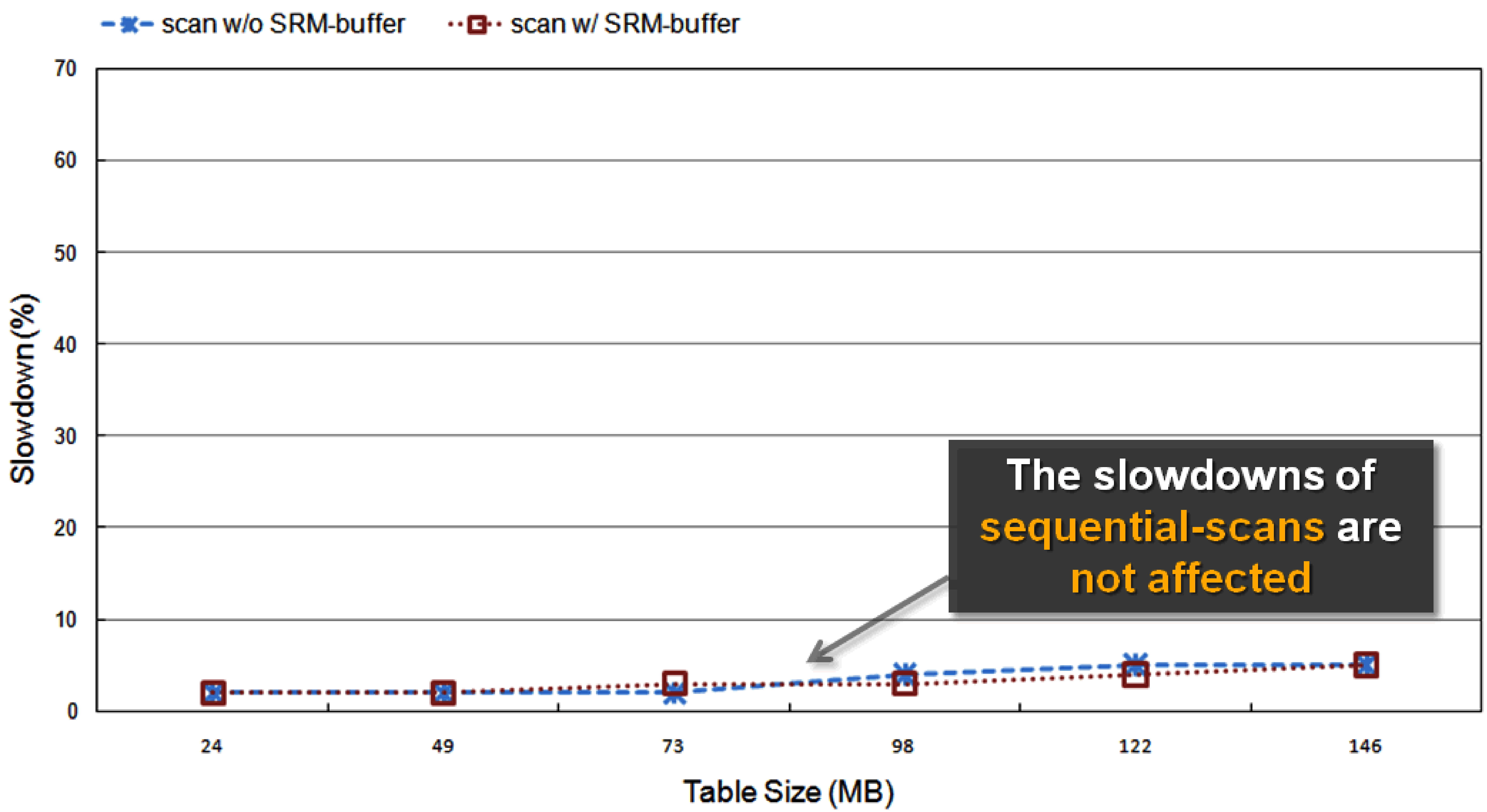
Experiments – Database 1

- PostgreSQL DB server supporting data warehouse workloads (star schema)
- One large fact table (≈4GB)
- Several small dimension tables (≈ 24MB – 146 MiB each)
- Hash-join-based queries co-running with sequential-scan-based queries

VM-intensive

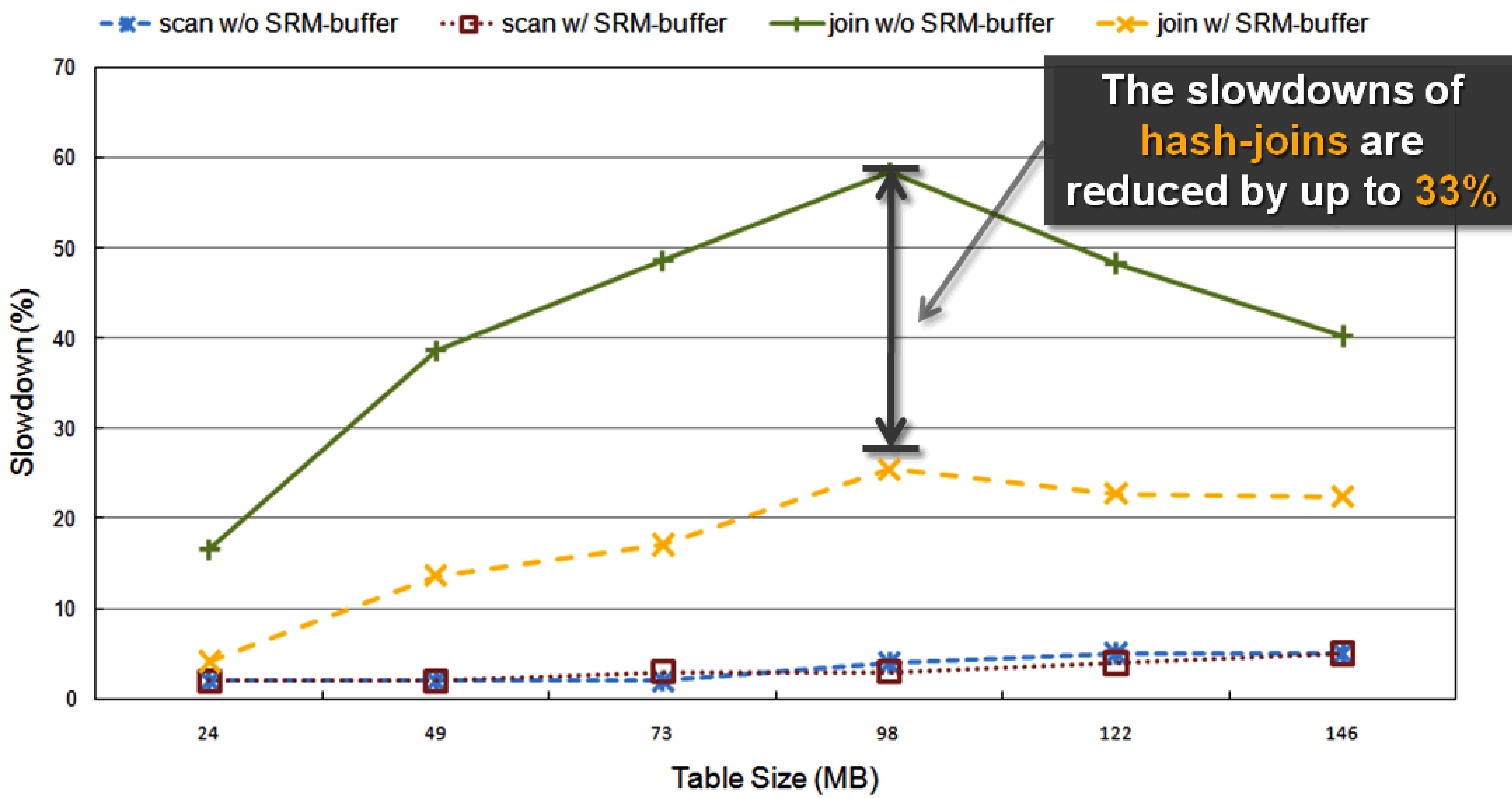
File-intensive

Experiments – Database 1



Slowdown of hash-join and sequential-scan compared to their solo runs

Experiments – Database 1

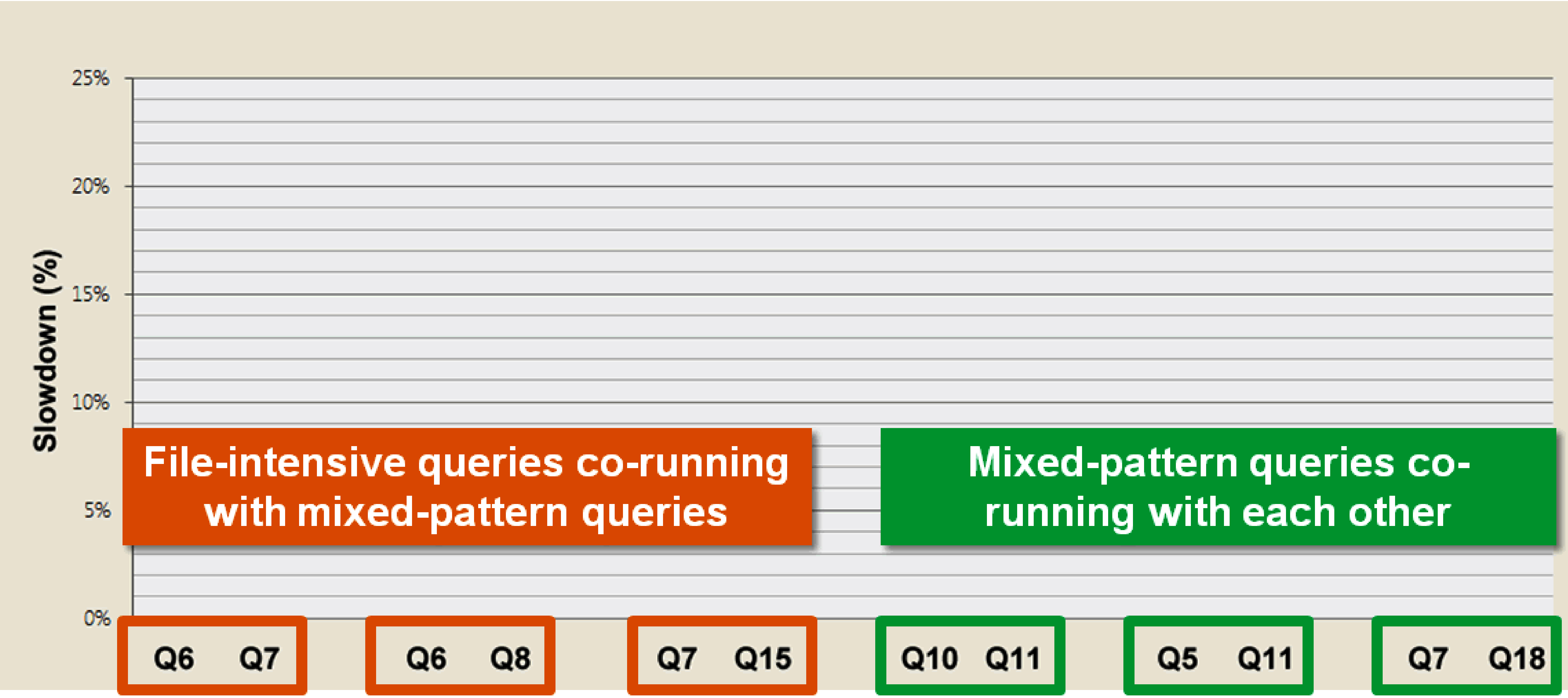


Slowdown of hash-join and sequential-scan compared to their solo runs

Experiments – Database 2

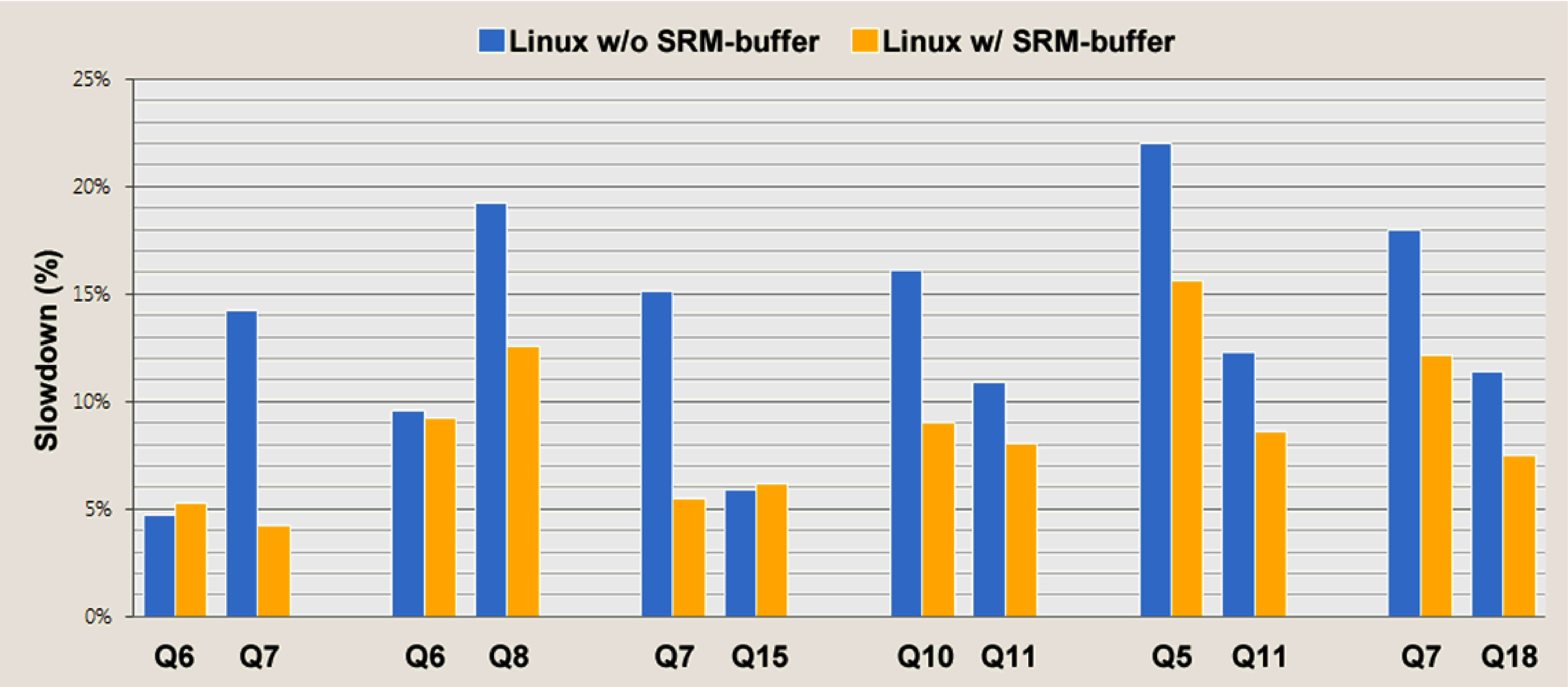
- Standard TPC-H Benchmarks (scale factor 2) on PostgreSQL
- Two groups of queries
  - Q6, Q15: spend most time sequentially scanning fact table, *lineitem*, in buffer cache
  - Q5, Q7, Q8, Q10, Q11, Q18: mixed with VM and file-intensive operations

Experiments – Database 2



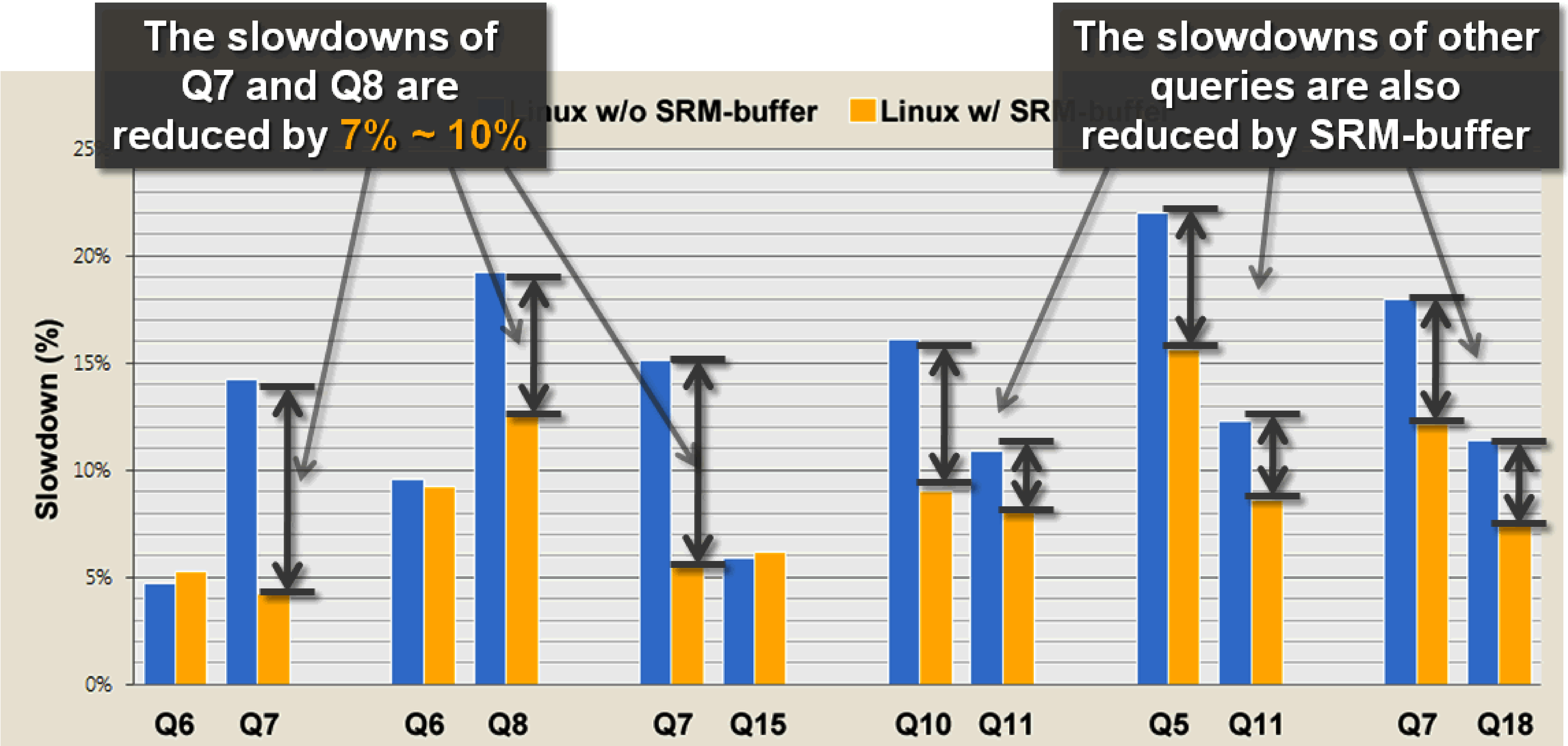
Slowdown of TPC-H queries compared to their solo runs

Experiments – Database 2



Slowdown of TPC-H queries compared to their solo runs

Experiments – Database 2



Slowdown of TPC-H queries compared to their solo runs

Experiments – Other Workloads

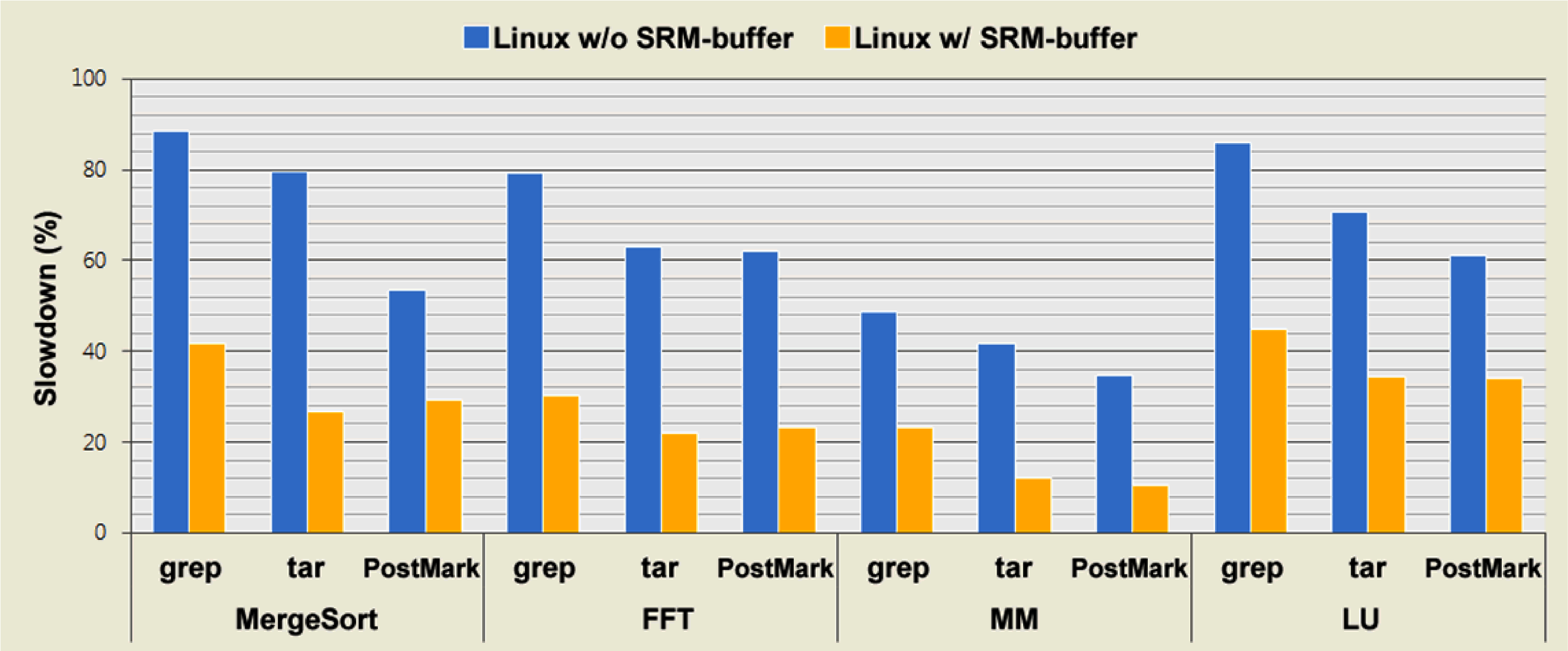
- **grep**: search PostgreSQL source tree
- **tar**: archive PostgreSQL source directory
- **PostMark**: read/append and create/delete small files

File-intensive Applications

- **MergeSort**: sort arrays using merge sort algorithm
- **SciMark2**: scientific computing benchmarks
  - FFT, MM, LU

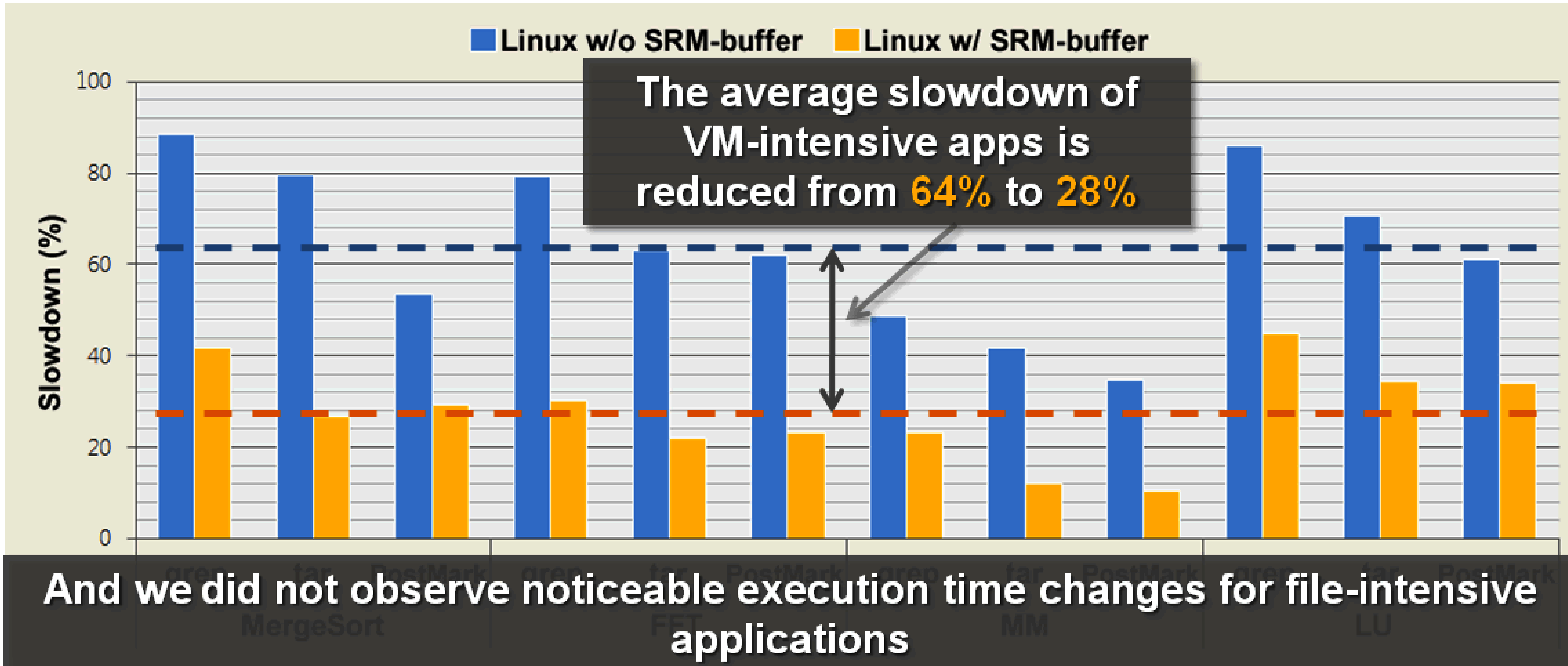
VM-intensive Applications

Experiments – Other Workloads



Slowdown of VM-intensive applications compared to their solo runs

Experiments – Other Workloads



And we did not observe noticeable execution time changes for file-intensive applications

Slowdown of VM-intensive applications compared to their solo runs

Experiments – Changing Access Patterns

The order in which file blocks are first loaded into buffer cache

VS.

The order in which file blocks are accessed by the applications

How is SRM-Buffer performance is affected when these two orders are different?

- **In this experiment, we test the effectiveness of SRM-buffer when access patterns change**
  - **grep: scan files in the order of their layout in the file system**
  - **diff: visit files in the alphabetic order of directory and file names**

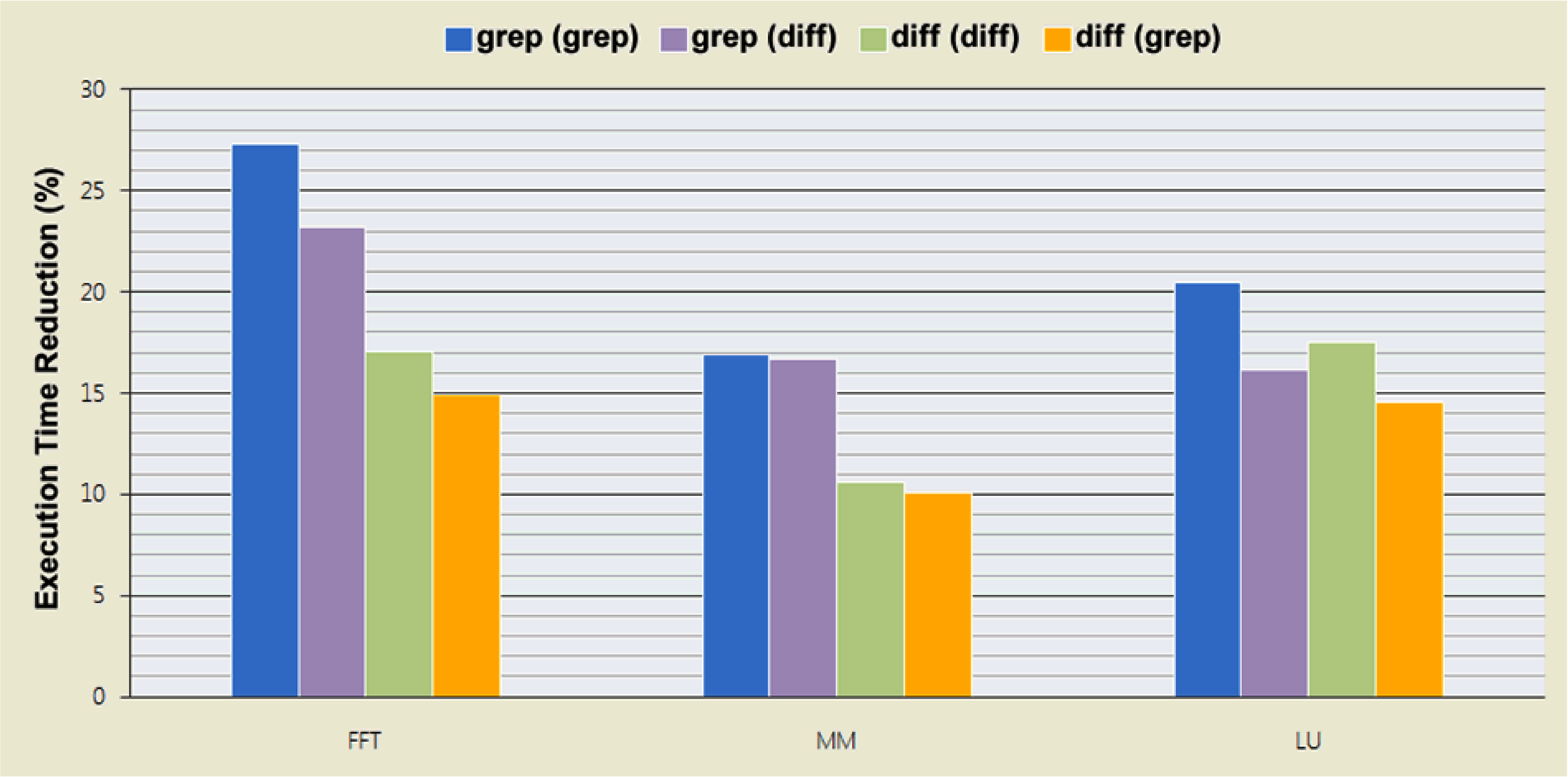
Experiments – Changing Access Patterns

- Each of grep/diff co-runs with each of FFT/LU/MM
- One loads file blocks and the other accesses them in buffer cache

Execution time reductions achieved by SRM-buffer as the access patterns change

Experiments – Changing Access Patterns

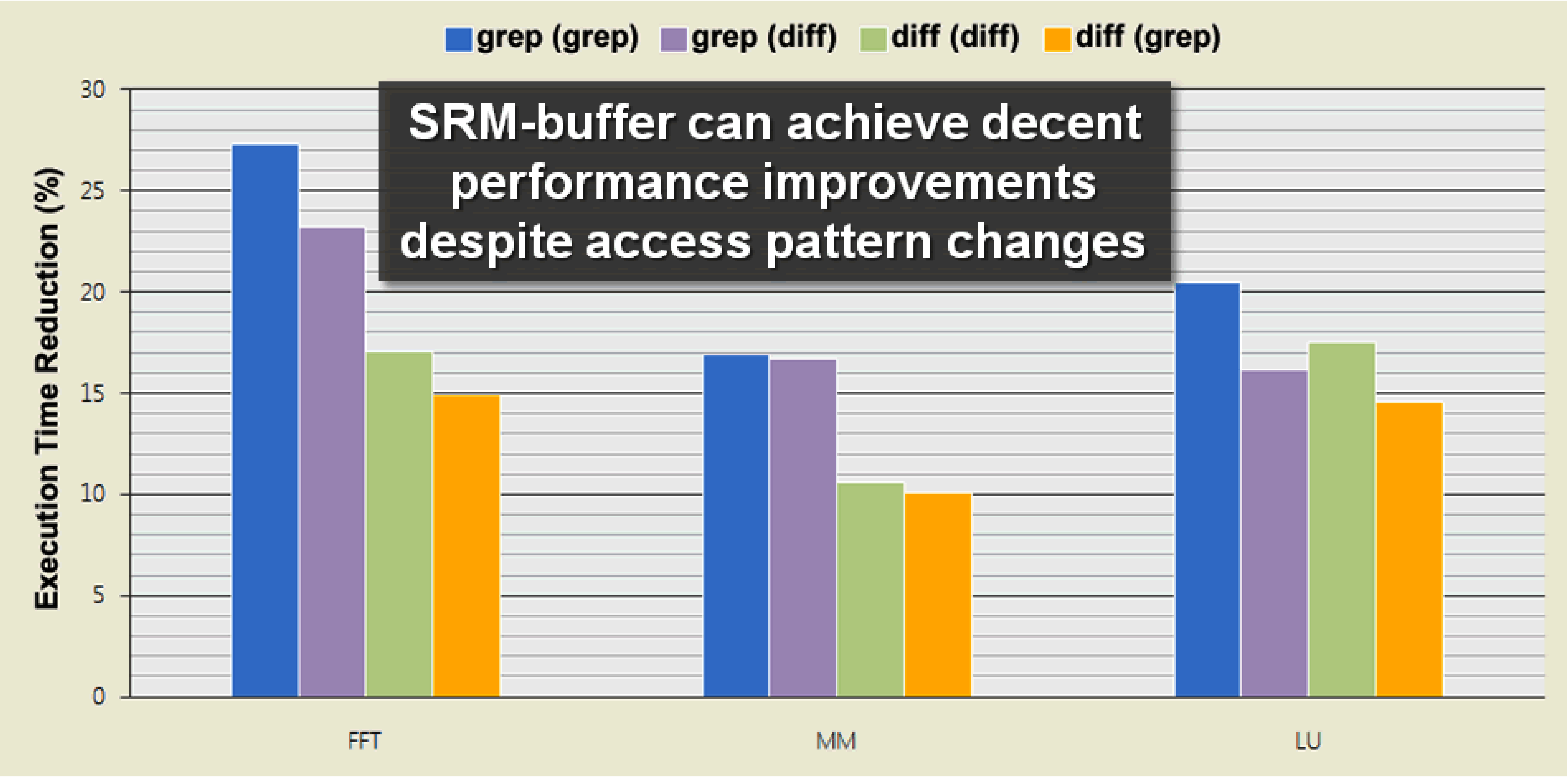
B (A): A loads file blocks; B accesses them in buffer cache



Execution time reductions achieved by SRM-buffer as the access patterns change

Experiments – Changing Access Patterns

B (A): A loads file blocks; B accesses them in buffer cache



Execution time reductions achieved by SRM-buffer as the access patterns change

Conclusion

- Accessing OS buffer cache can pollute shared last level CPU caches on multicores
- SRM-buffer prevents LLC from thrashing by carefully selecting memory-cache mapping
- We showed the effectiveness of SRM-buffer with several co-running workloads
- SRM-buffer is light-weight
  - Effective to mixed VM- and file-intensive workloads
  - Dominant VM-intensive: colored zone evenly allocates pages
  - Dominant file-intensive: colored zone does selected cache region mapping

THANK YOU

- Q & A