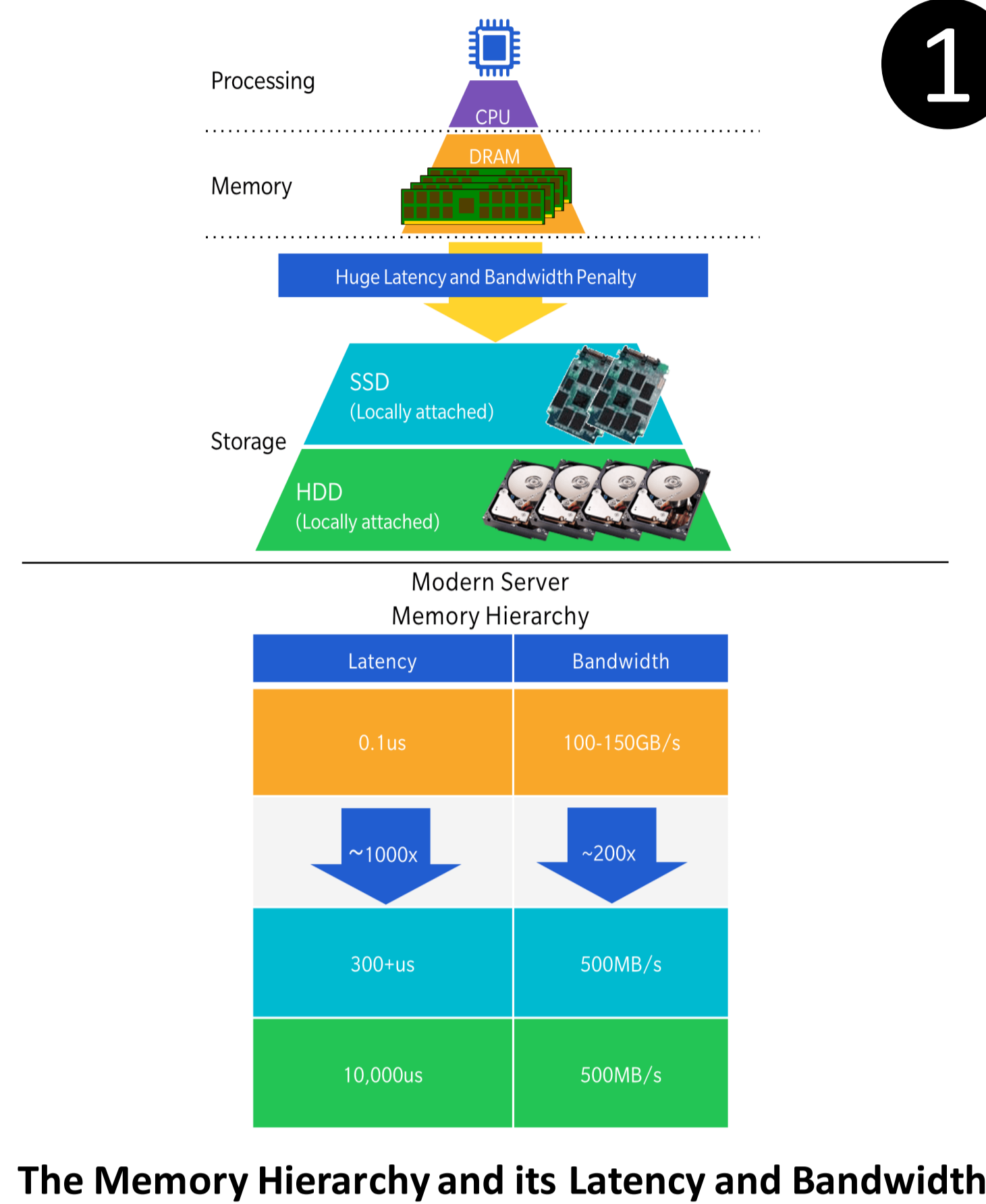# Memory-Aware Management for Multi-Level Main Memory Complex using an Optimization of the Aging Paging Algorithm

Gal Oren, Dr. Leonid Barenboim, Dr. Lior Amar

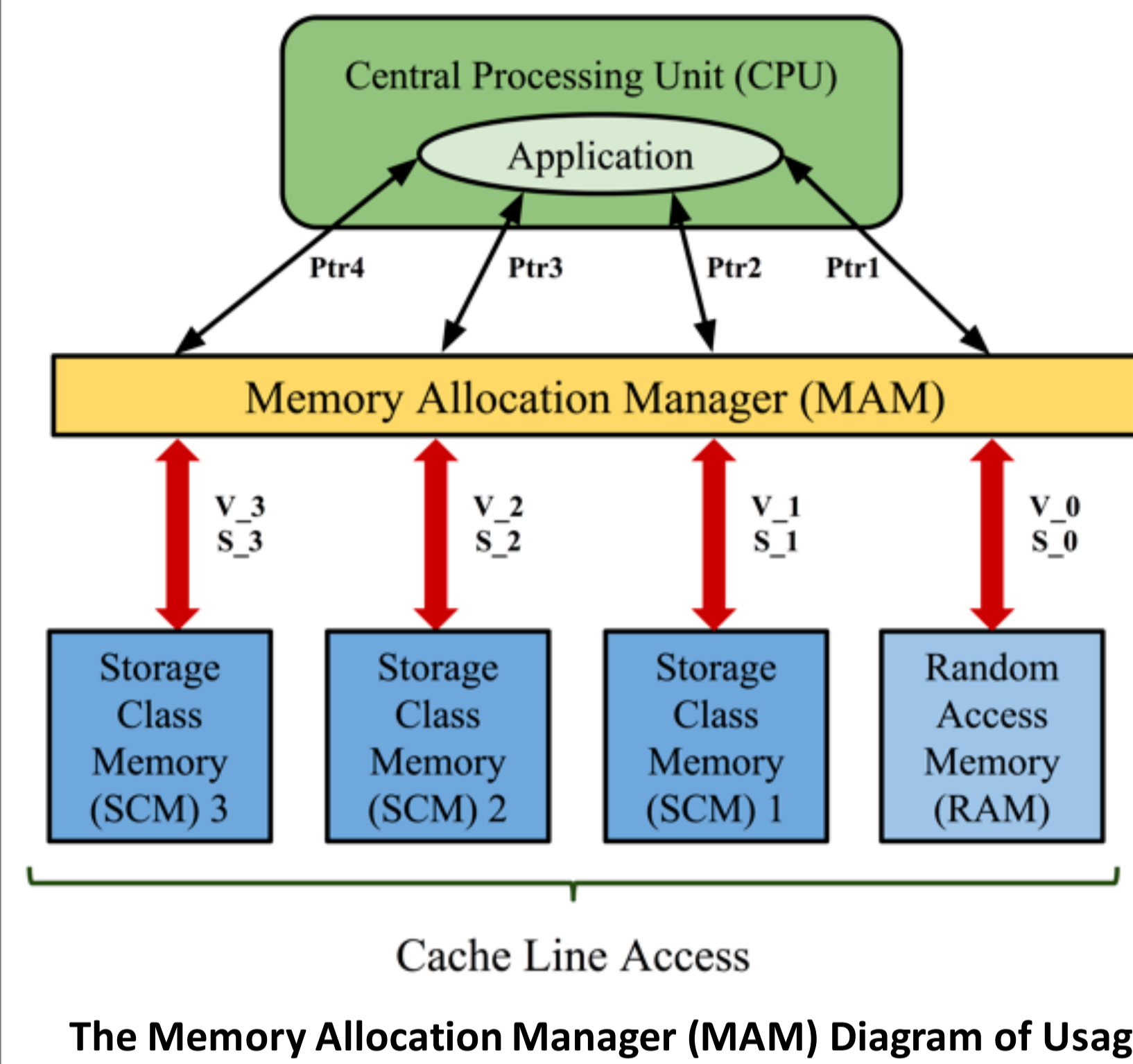orenw@post.bgu.ac.il , leonidb@openu.ac.il, liororama@gmail.com

1 — Introduction: Memory-aware vs. Memory-oblivious Algorithms

- Understanding the memory hierarchy can be useful in order to enhance the performance of an algorithm or a data structure.

- Algorithms and data structures that adjust to a specific memory organization are known as memory-aware or memory-conscious.

- Algorithms and data structures that do not take into consideration memory parameters and hierarchy are called memory-oblivious.

- Design of memory-aware algorithms requires awareness of the memory hierarchy, as the latency and bandwidth penalty between the different levels of the memory is significant.

The Memory Hierarchy and its Latency and Bandwidth

2 — Previous Work: The Parallel Disk Model (PDM) & The STXXL Library

- Several simple models have been introduced for designing I/O-efficient algorithms and data structures.

- The most realistic model is the Parallel Disk Model (PDM) of Vitter and Shriver. In this model, I/Os are handled explicitly by the application.

- The most common implementation of the PDM model can be found at the STXXL project. The core of STXXL is an implementation of the C++ standard template library STL for external memory (out-of-core) computations, i.e., STXXL implements containers and algorithms that can process huge volumes of data that only fit on disks.

- The performance features of STXXL include transparent support of multiple disks, variable block length, overlapping of I/O and computation, and prevention of OS file buffering overhead.

Serial Disk Model (SDM) vs. Parallel Disk Model (PDM)

3 — Future Memory: Storage Class Memory (SCM)

- Storage Class Memory (SCM) proposes to minimize or even close the widening gap between CPU processing speeds, the need to rapidly transfer big data blocks, and the read-write speeds suggested by HDD reliant systems.

- The SCM is a technology which represents a new hybrid form of storage and memory with unique characteristics, meaning a memory which is non-volatile, cheap in a per bit cost, has fast access times for both read and writes using cache line access, and is solid state.

- Also, the SCM is supposed to have different versions with different access speeds and different volumes, meaning that it may be possible to add different SCM devices to the memory hierarchy as an extension of the RAM, and manage this enlarged main memory complex using special algorithms, as the PDM model manages the disk complex using STXXL library.

The SCM in Context of the Available Memories

4 — Suggested Solution: Memory Allocation Manager (MAM)

- A Memory Allocation Manager (MAM), based on the idea of C language *malloc* and STXXL allocation modules, should be able to take the power of the memory management from the virtual memory mechanism and to deliver it to the developer.

- In this scenario there is no common 'flat' view of the main memory, but a plurality of memories, which the new MAM, by the instructions from the application, will take charge of them.

- The developer will have the option to control those memories – serially or in parallel – in order to achieve the best performance possible.

The Memory Allocation Manager (MAM) Diagram of Usage

5 — The Elephant in the room: A Re-Write of All Codes?

- Although an explicit usage of the MAM seems to be the best approach for multi-level main memory complex management, this technique is still very far from being realistic, and the chances that it would be implemented in current codes using High Performance Computing (HPC) platforms – the primary target group which need this massive enlargement of the main memory – is quite low.

- The reason for this problem is that the MAM require not only to re-modify the memory platform and the access to it, but obviously also to re-write the memory-oblivious codes that is currently based on transparent memory access; a re-write that in many important codes is very complicated.

6 — The Solution: Paging Algorithms as a Practical Resort

- Introducing the MAM concepts and the multiple main memory levels awareness into the classic paging algorithms can be a good solution, which will not be a big performance compromise, but a modest one.

- Thus, creating a simulator that will be able to run on any computer or sever, and that will be able to simulate a situation in which frames of memory are managed and mapped to specific levels of memory in a memory-aware fashion, is beneficial for the development of this technology.

- To this end, we built the **DeMemory simulator** framework for testing of memory-aware and memory-oblivious paging algorithm as well as for other algorithms that will be built in the future.

Analysis of the DeMemory Workflow

7 — Memory-Aware Aging Paging Algorithm for Multi-Level Main Memory Complex
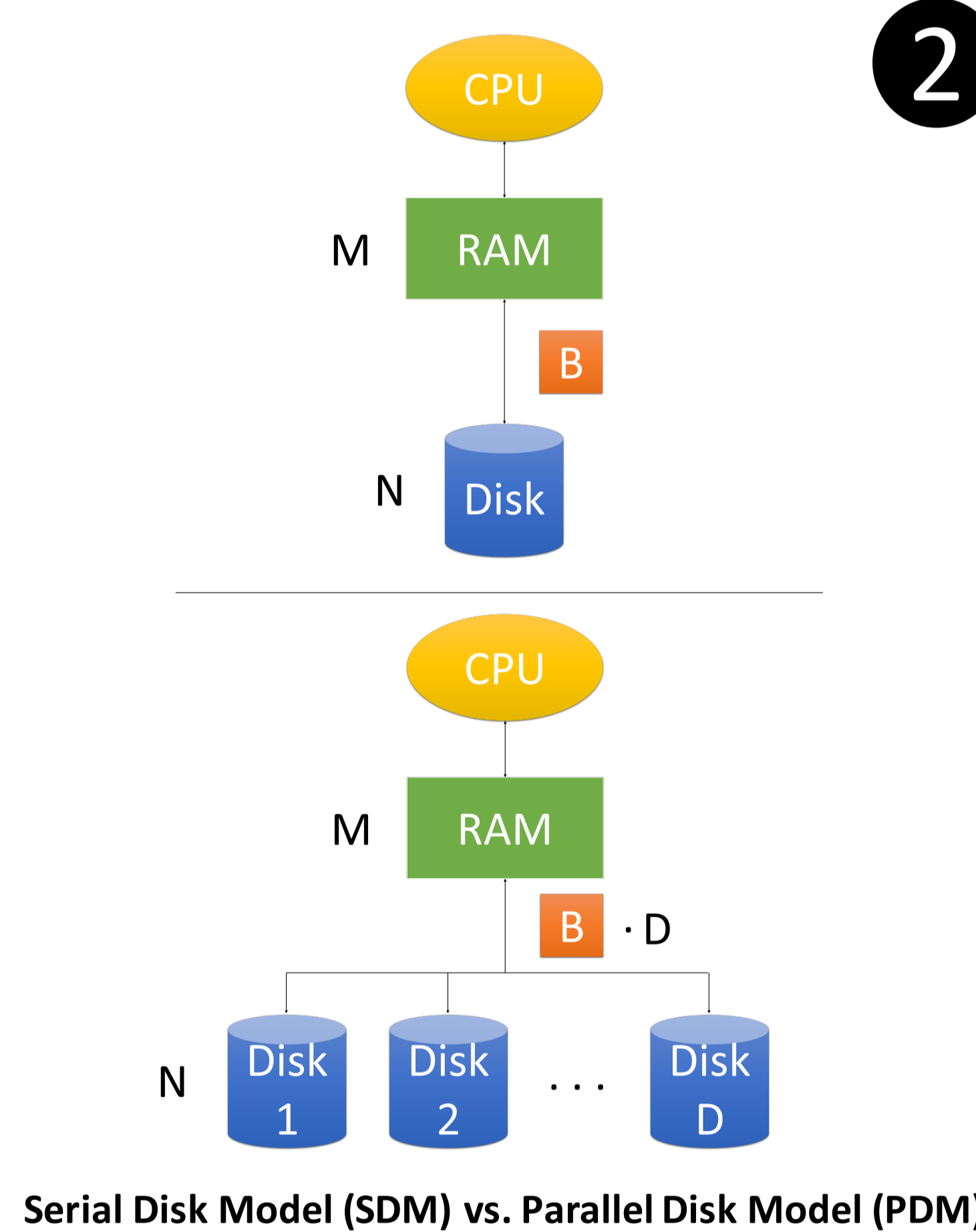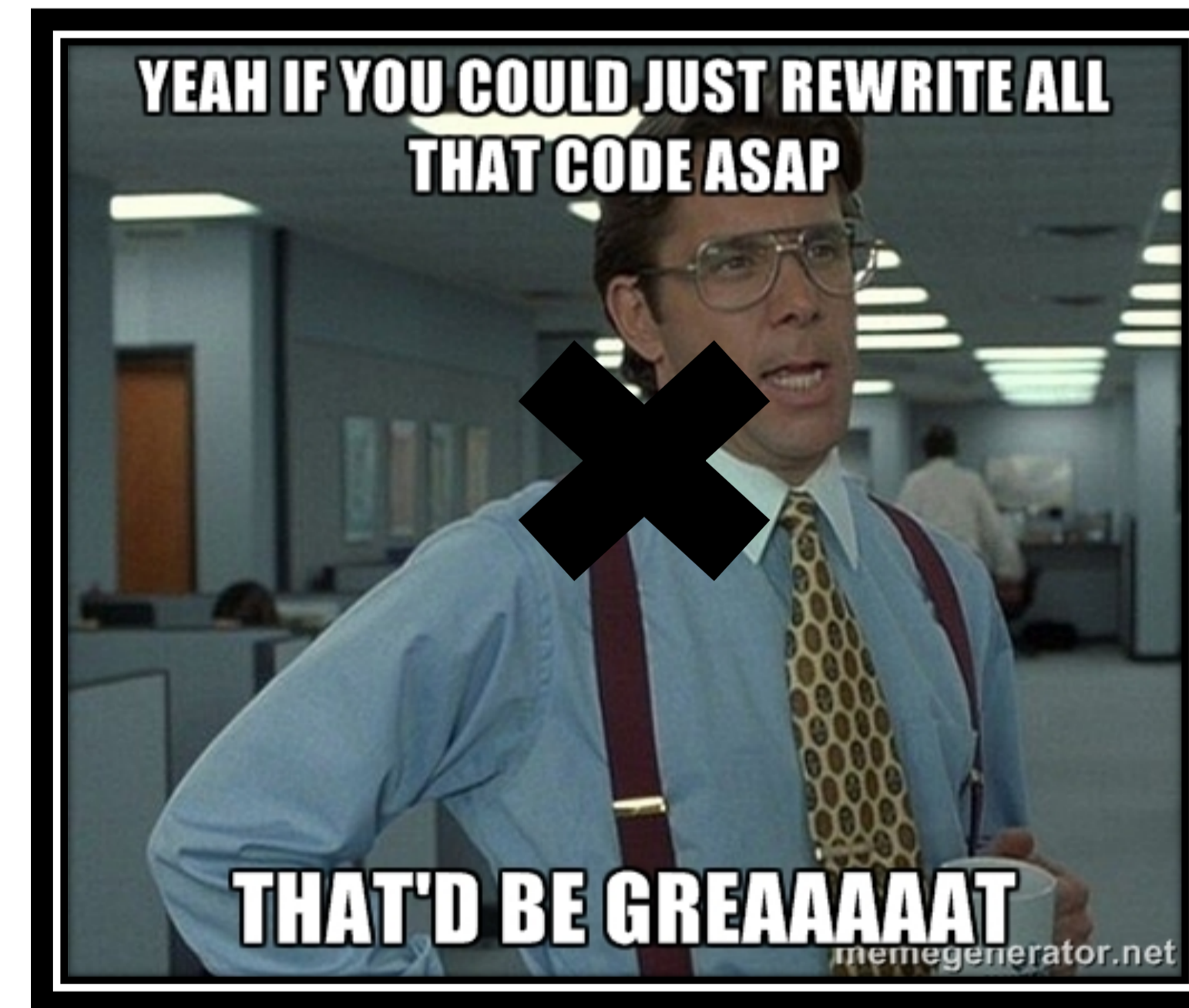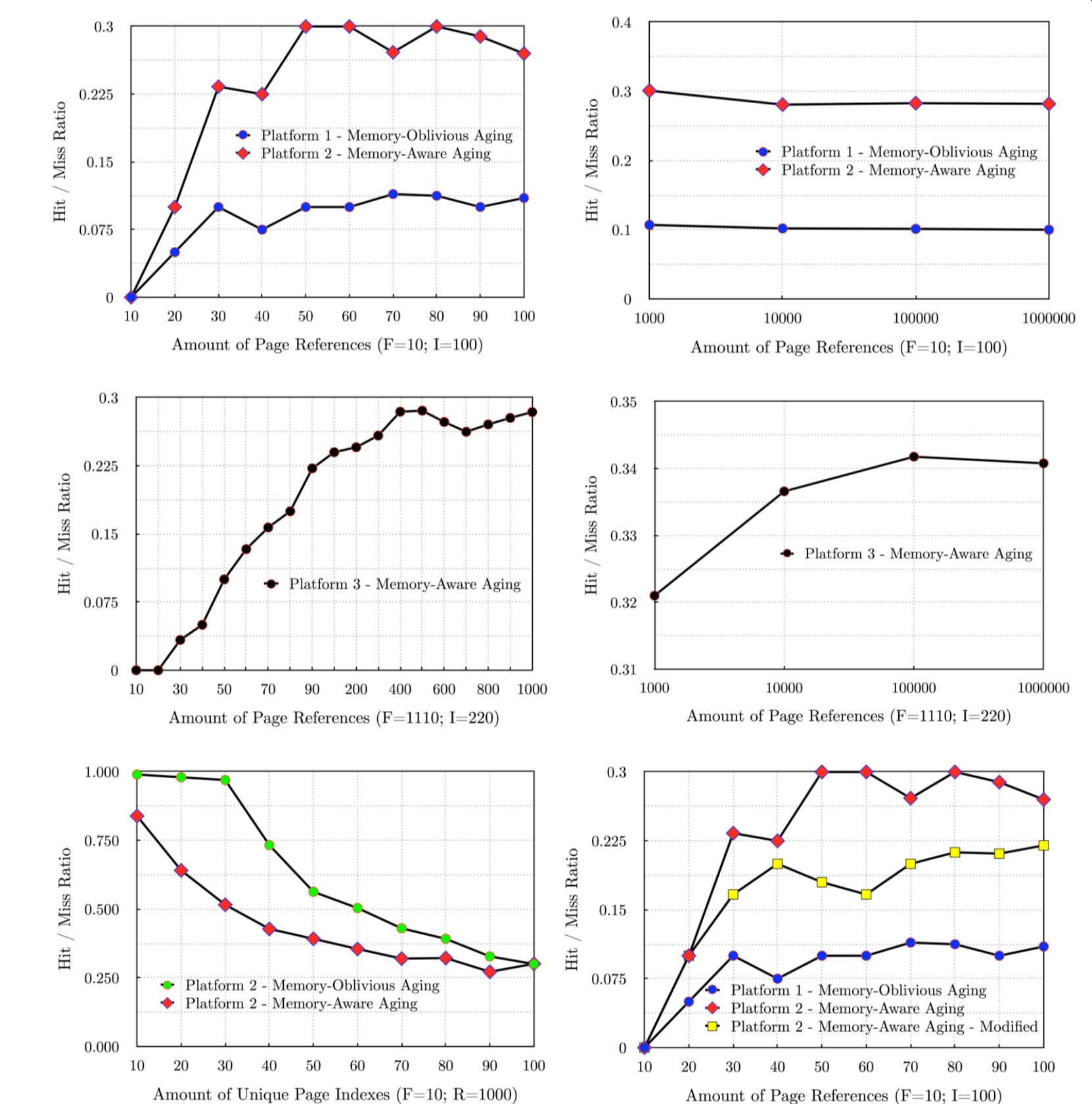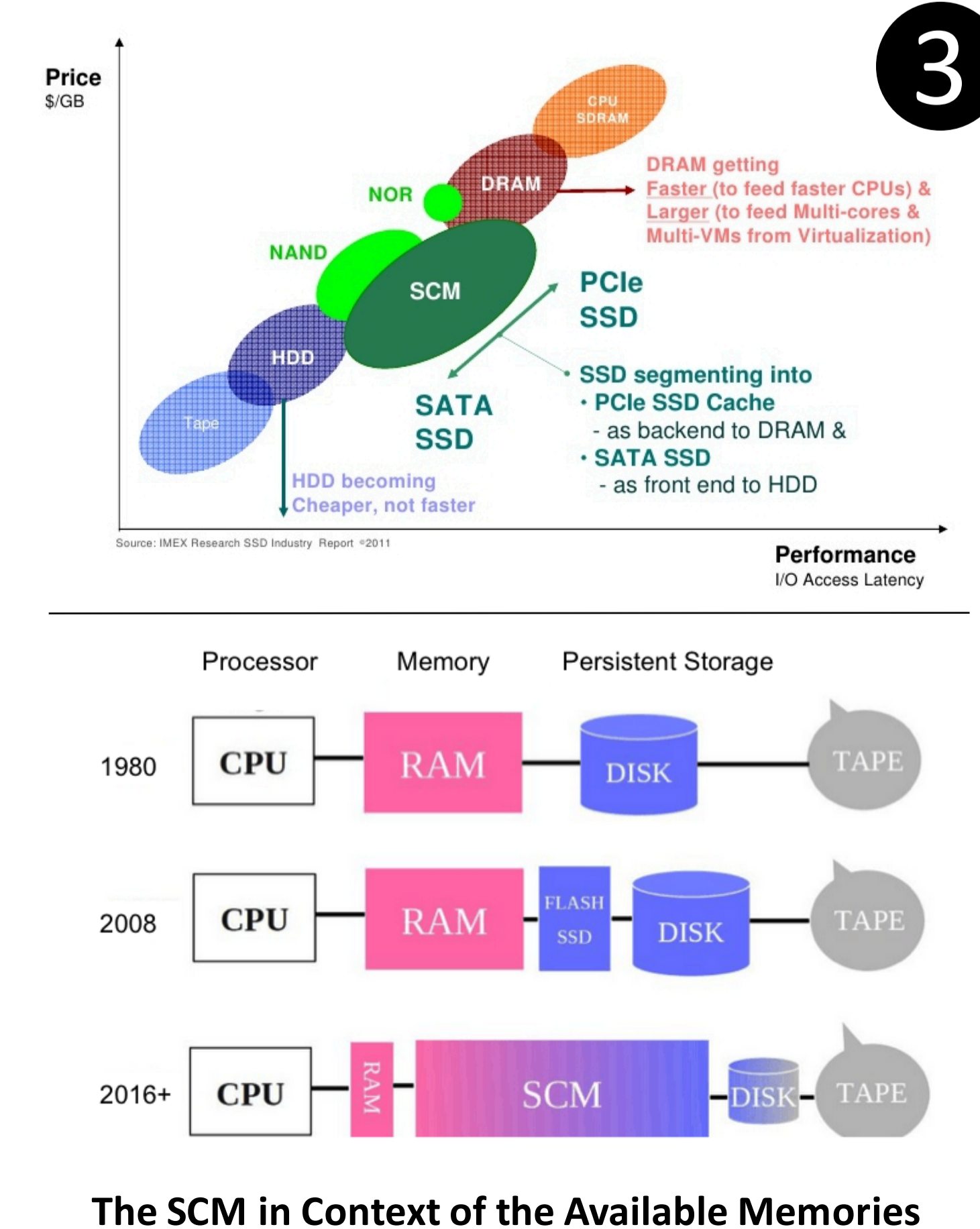
- A transition of the memory-oblivious Aging algorithm to be a memory-aware algorithm discovered to be the best paging algorithm for our problem, especially because of the existence of a linear proportion between the degradation of the referenced bits and the amount of time that a specific page has not been in use.

- In this hypothesis, unlike in the other multi-level main memory complex paging algorithms adaptations and adjustments of ours (like NRU, FIFO etc.), there is an interesting phenomenon.

- Specifically, there is a possibility to create a direct link between the amount of zeroes in the beginning of the page referenced bits to the level of memory that that page should be evicted to according to its usage proportion (L) using a calculation which should take only few floating-point operations, and which is based on information the operating system already holds.

Formulation of the Aging algorithm for multi-level main memory complex:

- Set memory levels to N (ML = N).
- Set current memory level pointer to the highest (L = 1).

1. **Insertion** of a new page:
   1.1. *If* the current memory level pointer is higher than the lowest memory level (L > ML):
       1.1.1. *Return* False. /* Recursion Termination */
   1.2. Call to the page.
   1.3. *If* the page exists in the memory / storage:
       1.3.1. *Check* if placing in the L-level of memory is possible.
       1.3.2. *If* placement possible:
           1.3.2.1. *Place* page at the L-level of memory.
           1.3.2.2. *Return* True. /* Recursion Termination */
       1.3.3. *Else If* placement impossible:
           1.3.3.1. *Find* the page with the lowest referenced counter.
               1.3.3.1.1. *Remove* the page with the lowest referenced counter.
               1.3.3.1.2. *Place* the page instead of the removed page.
               1.3.3.1.3. *Do Insertion* of the page with the lowest referenced counter to the proportionate level of memory based on the amount of the zeroes in the beginning of the page reference bits. /* Recursion Invocation */
   1.4. *Else If* page does not exist in the memory / storage:
       1.4.1. *Return* False. /* Recursion Termination *

$$L = \left\lceil \frac{Amount\ of\ Initial\ Zero\ Bits}{Amount\ of\ Reference\ Bits} \right\rceil \cdot ML$$

1. **Update** of an existing page (by the OS):
   1.1. *If* Read / Write action performed on the page:
       1.1.1. *Set* R bit to 1 (R = 1).
   1.2. *If* clock interrupt:
       1.2.1. *Right Shift* one bit to all of the pages counters.
       1.2.2. *Add* the R bit to the leftmost bit of all of the pages counters

8 — Analysis of the Aging Paging Algorithm - Benchmarking Methods

- In order to verify our hypothesis, especially when the memory complex is a complex of standard RAM and different types of Storage Class Memory (SCM), we need to verify two main hypotheses:

- First, that the memory-aware algorithm is resulting in an equally efficient Hit / Miss ratio as the memory-oblivious Aging algorithm when it implemented on multi-level main memory complex using the explicit cache mechanism.

- Second, that the memory-aware algorithm is resulting in a significantly better access speed than the memory-oblivious Aging algorithm when it is implemented on multi-level main memory complex using the explicit cache mechanism. This parameter should show that the new algorithm actually manages to transfer the different pages to their designated memory levels in the memory, and that this redirection of pages actually manages to move the more needed pages to a better access-time levels in the memory complex, and by that to achieve better total performances.

- Therefore, we tested and compared the two types of algorithms – the memory-aware Aging and the memory-oblivious Aging – on different platforms.

9 — Analysis of the Aging Paging Algorithm - Results and Conclusions

- We discovered that the Hit / Miss ratio as function of the amount of unique page indexes and the amount of page references was almost the same, meaning that despite the fact that the memory-aware Aging algorithm is transferring data to other levels even before the memory level is full, there is no negative impact.

- Furthermore, and most importantly, we examined the access speed to the memory complex as function of the amount of page references and discovered that **the memory-aware Aging algorithm is yielding about 75% improvement in the access speed over the memory-oblivious Aging algorithm**, as evident from the lower average access levels in comparison to the memory-oblivious algorithm (Recall that the memory levels are ordered according to their speed, and lower levels are faster).

- This means that although the Hit / Miss ratio in both algorithms is almost the same in most cases, **there is a clear advantage to the memory-aware Aging algorithm**, as it resulting in much better performances than the memory-oblivious Aging algorithm using multi-level main memory complex.