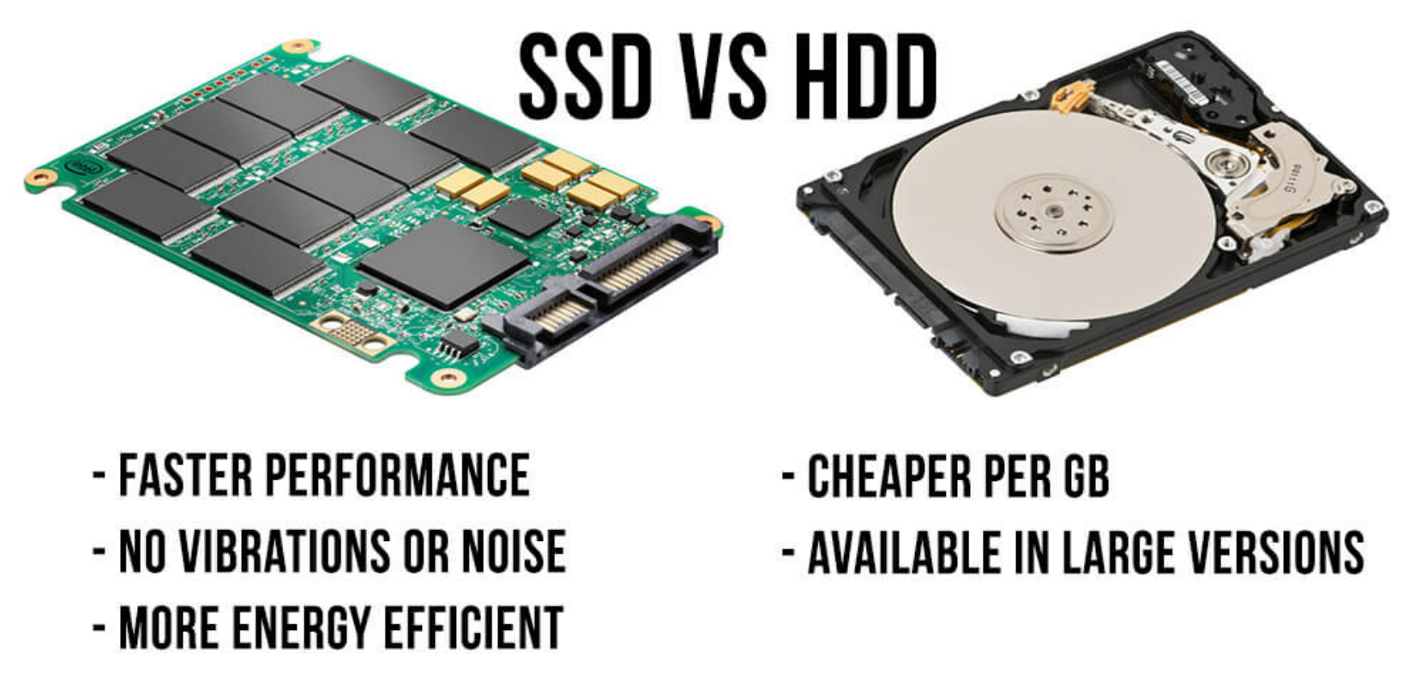
Taming Unaligned Writes in Solid State Disk

Abu Zafar Md Nuruzzaman Abir **Xuechen Zhang**
Washington State University Vancouver

SSD is Widely Used Nowadays



- SSD's exhibit virtually no access time.
- SSD's are at least 15 times faster than HDD in terms of random I/O operations.
- SSD's failure rate is 5 -10% less than that of HDD.

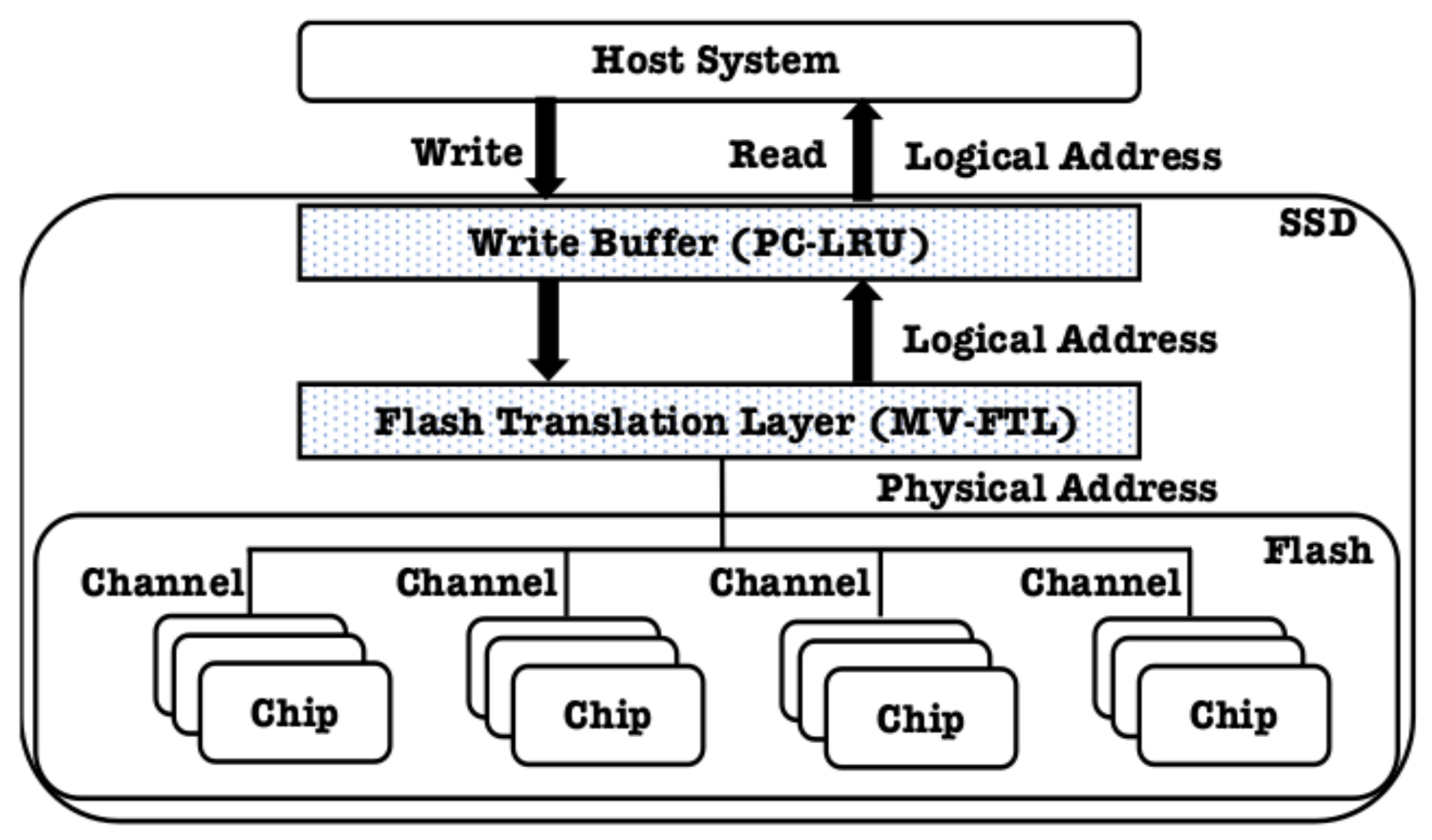
Underlying Properties of SSD

- SSDs use flash memory as a storage medium.
- Flash memory reads and writes data in the unit of a page and erases in the unit of a block.
- SSDs usually perform out-of-place updates.
- SSDs use the FTL (Flash Translation Layer) to support hosts to access flash memory via the block interface.

Challenges of Unaligned Writes in SSD

- Unaligned writes are not aligned with the page boundary - the number of unaligned writes increases proportional to the increase in flash density.
- For architectural nature, performance-sensitive applications may generate many unaligned writes on SSDs.
- Unaligned writes causes many issues like e.g., chip congestion, sub-request blocking, chip load imbalance, and write space amplification.
- Problem: How to reduce the effect of unaligned writes? Solution: Modifying the write buffer with **Partial page congestion aware Least Recently Used (PC-LRU)** algorithm

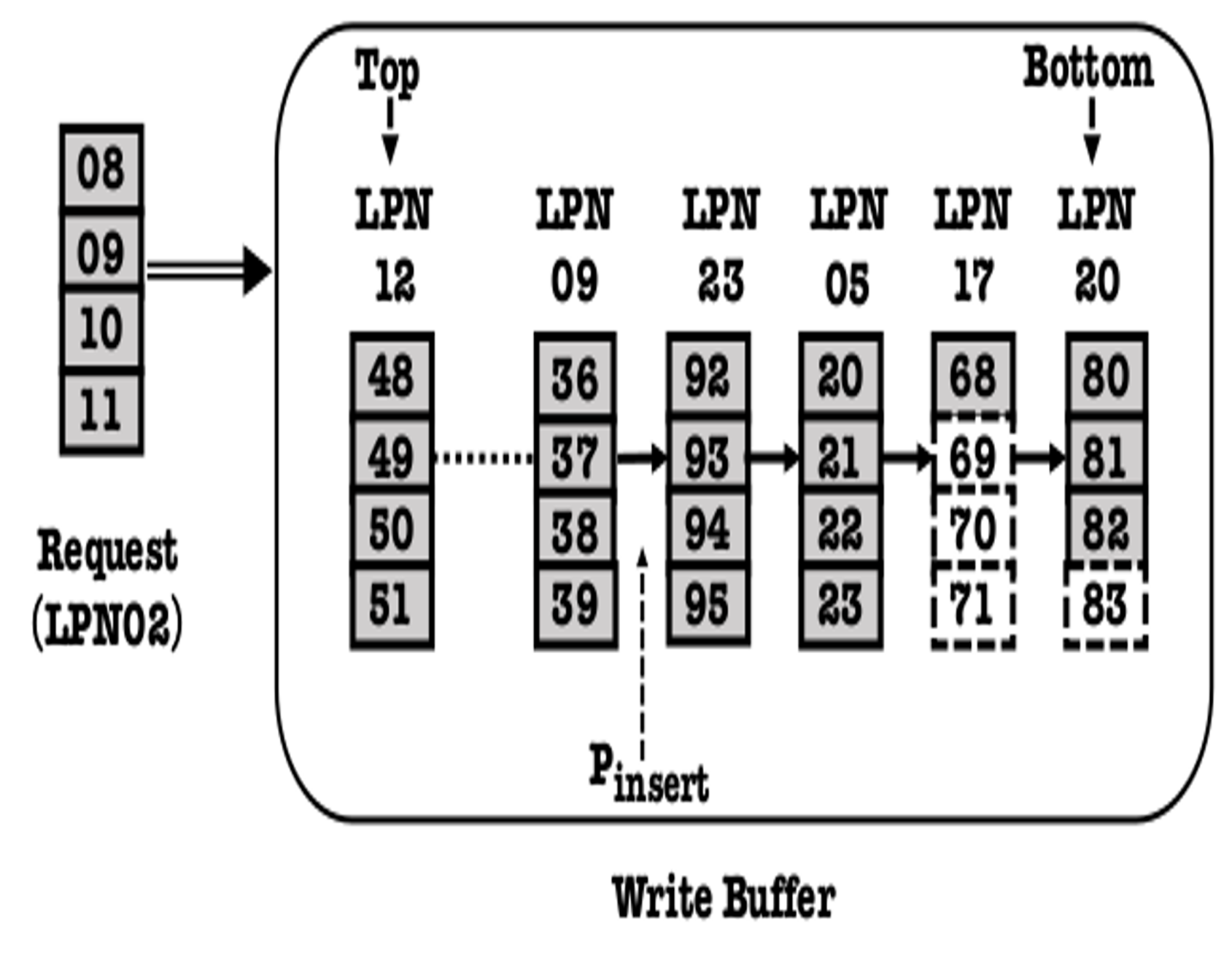
Solid State Disk (SSD) System Architecture



- The host system issues I/O requests to SSDs at the unit of a sector.
- The RAM buffer in SSDs is used only for write requests.
- FTL writes the data to flash memory after determining the physical location of the write requests.
- When FTL receives unaligned overwrite requests, it incurs read-modify-write operations.

Operation Process of PC-LRU

- **Partial pages vs Full page**
  - ❖ Partial pages refer to partially-filled pages where some sectors in the page are not updated and full pages refer to pages that all sectors in the pages are to be updated.
  - ❖ PC-LRU prioritizes the eviction of full pages over partial pages.
- **I/O Request Movement**
  - ❖ I/O requests enter the top of LRU stack L and then pushed to the bottom of L, further dropped out of the write buffer.
  - ❖ If a full page is not at the bottom and there are partial pages between the position of the full page and the bottom of L, each of the partial pages should be re-inserted to a new position P_{insert} in the stack.



Write buffer, the dotted boxes having 4 sectors are partial pages and the filled boxes are full pages

- **Decision regarding P_{insert} adjustment**

  Traditional LRU algorithm is being modified with the idea of implementing P_{insert} by moving the tail page of the buffer to
  - ❖ a fixed position in the buffer
  - ❖ the head of the buffer

Major data structure (write buffer modification) code

```
temp_node = ssd->dram->buffer->buffer_tail;

    while (temp_node->complete_flag == 0 && temp_node->whether_access == 0)

            {
                ssd->dram->buffer->buffer_tail = temp_node->LRU_link_pre;
                ssd->dram->buffer->buffer_tail->LRU_link_next = NULL;
                temp_node->LRU_link_pre = NULL;

                temp = ssd->dram->buffer->buffer_head;
                    int i;
                   for ( i = 1; i < pos -1 ; i++) {
                   temp = temp->LRU_link_next;
                    }
                temp_node->LRU_link_next = temp->LRU_link_next;
                temp_node->LRU_link_pre = temp;
                temp->LRU_link_next = temp_node;

                   if (temp_node->LRU_link_next != NULL)
                       temp_node->LRU_link_next->LRU_link_pre = temp_node;
                temp_node->whether_access = 1;
                temp_node = ssd->dram->buffer->buffer_tail;

            }
```

Insert to a fixed position, say 60

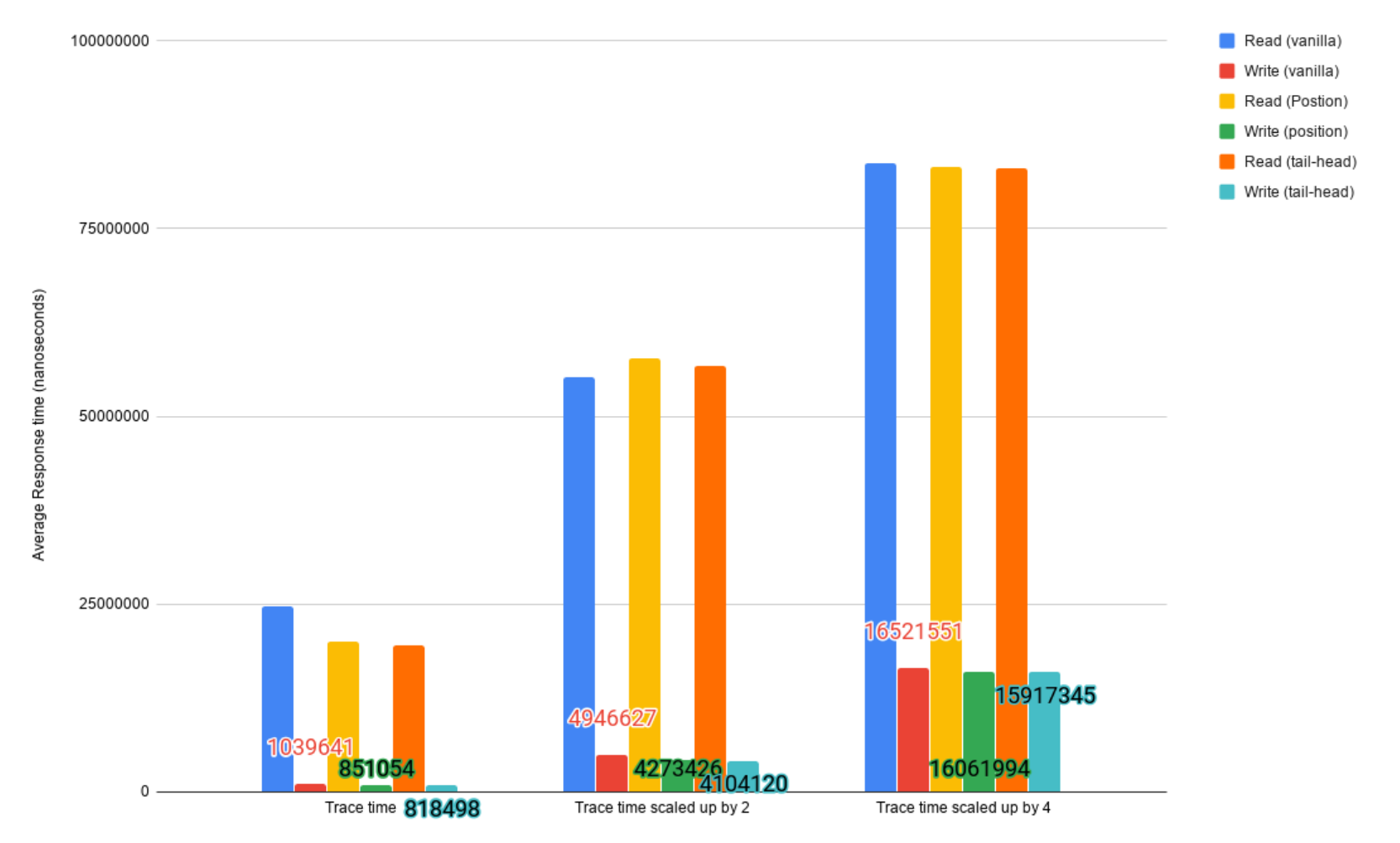
```
temp_node = ssd->dram->buffer->buffer_tail;

    while (temp_node->complete_flag == 0 && temp_node->whether_access == 0)

            {
                ssd->dram->buffer->buffer_tail = temp_node->LRU_link_pre;
                    ssd->dram->buffer->buffer_tail->LRU_link_next = NULL;
                    temp_node->LRU_link_pre = NULL;
                    temp_node->LRU_link_next = ssd->dram->buffer->buffer_head;
                    ssd->dram->buffer->buffer_head->LRU_link_pre = temp_node;
                    ssd->dram->buffer->buffer_head = temp_node;
                    temp_node->whether_access = 1;
                    temp_node = ssd->dram->buffer->buffer_tail;

            }
```

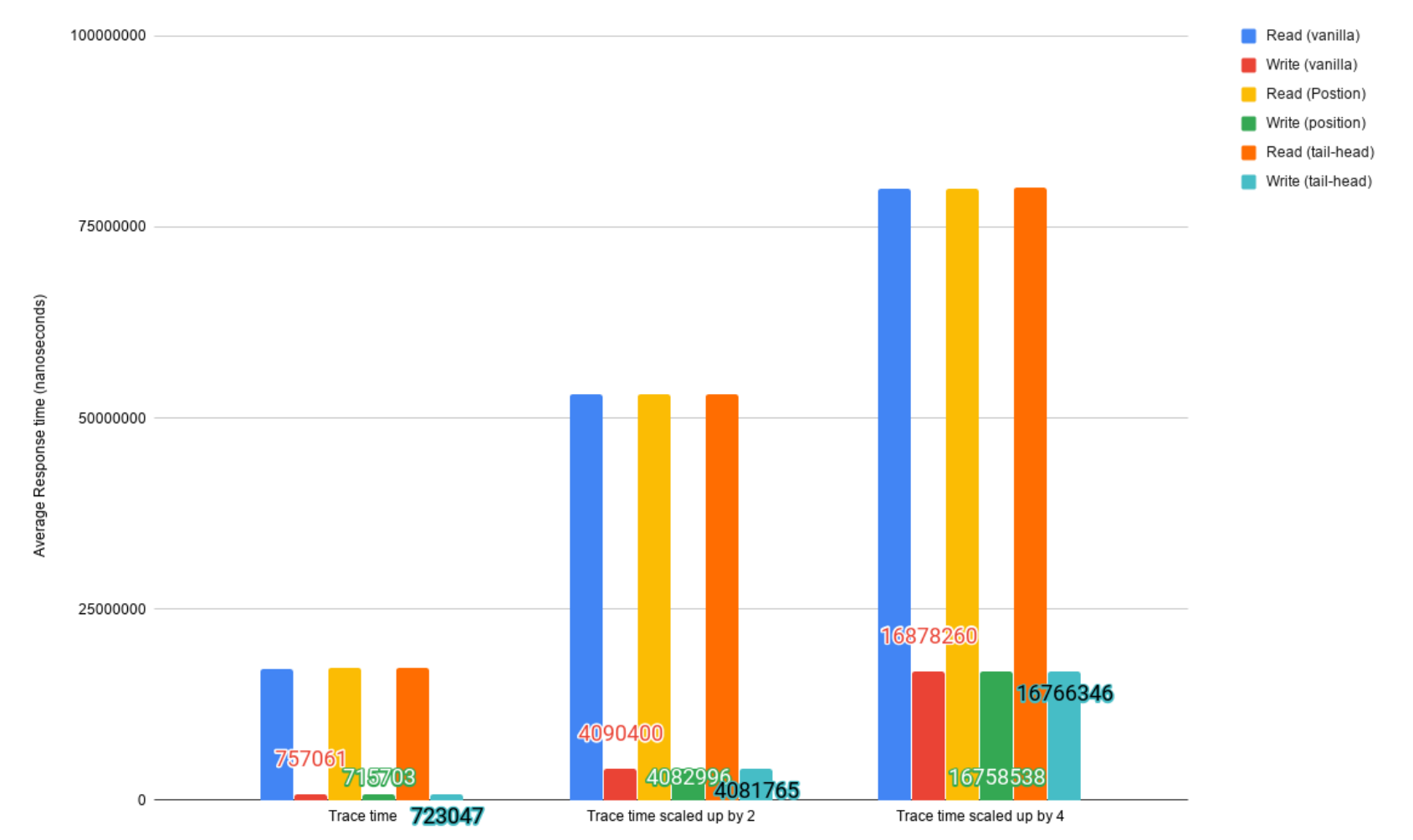
Insert from tail to all the way to head

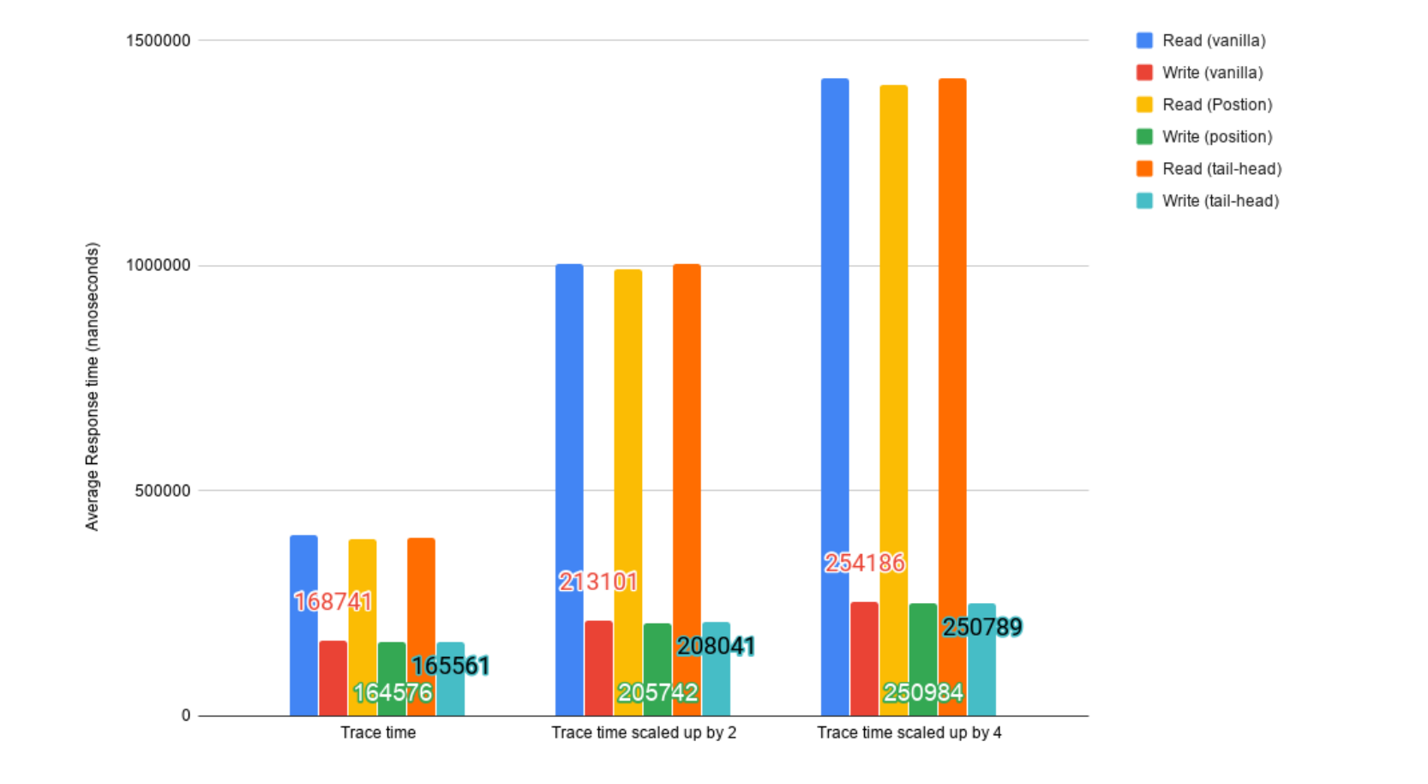
Experimental Setup

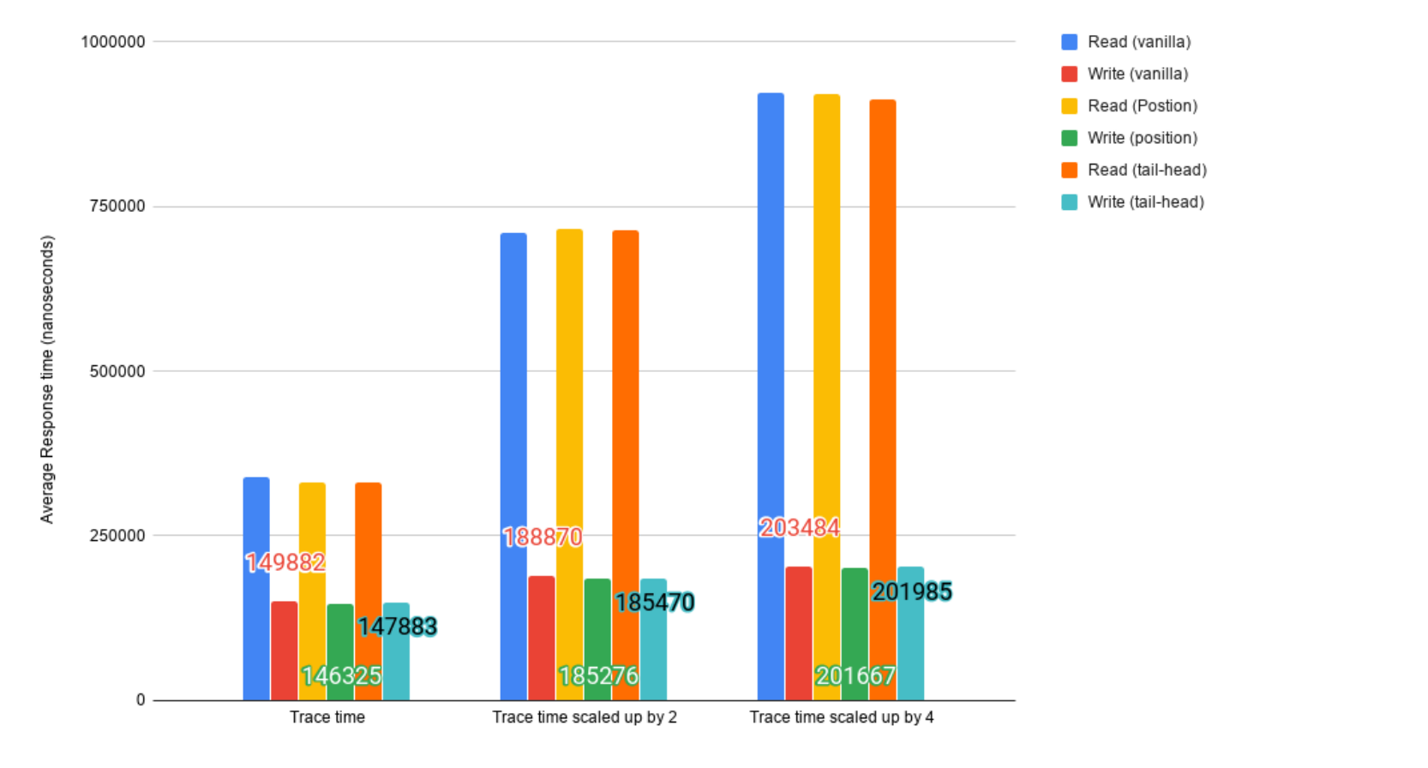
SSDSim Simulator	Description
Traces used: Block I/O traces	1. Microsoft Network (MSN) File Server trace (MSNFS) 2. MSN Storage metadata server trace (MSCFS) 3. MSN Development tool release trace (MSDTR).
variable dram (buffer) size	128KB, 8MB, 16 MB, 32MB
variable page size	2KB, 4KB, 8KB
Aging Condition	40%, 70%
Timestamp Scaling	2x, 4x, 5x

Evaluation

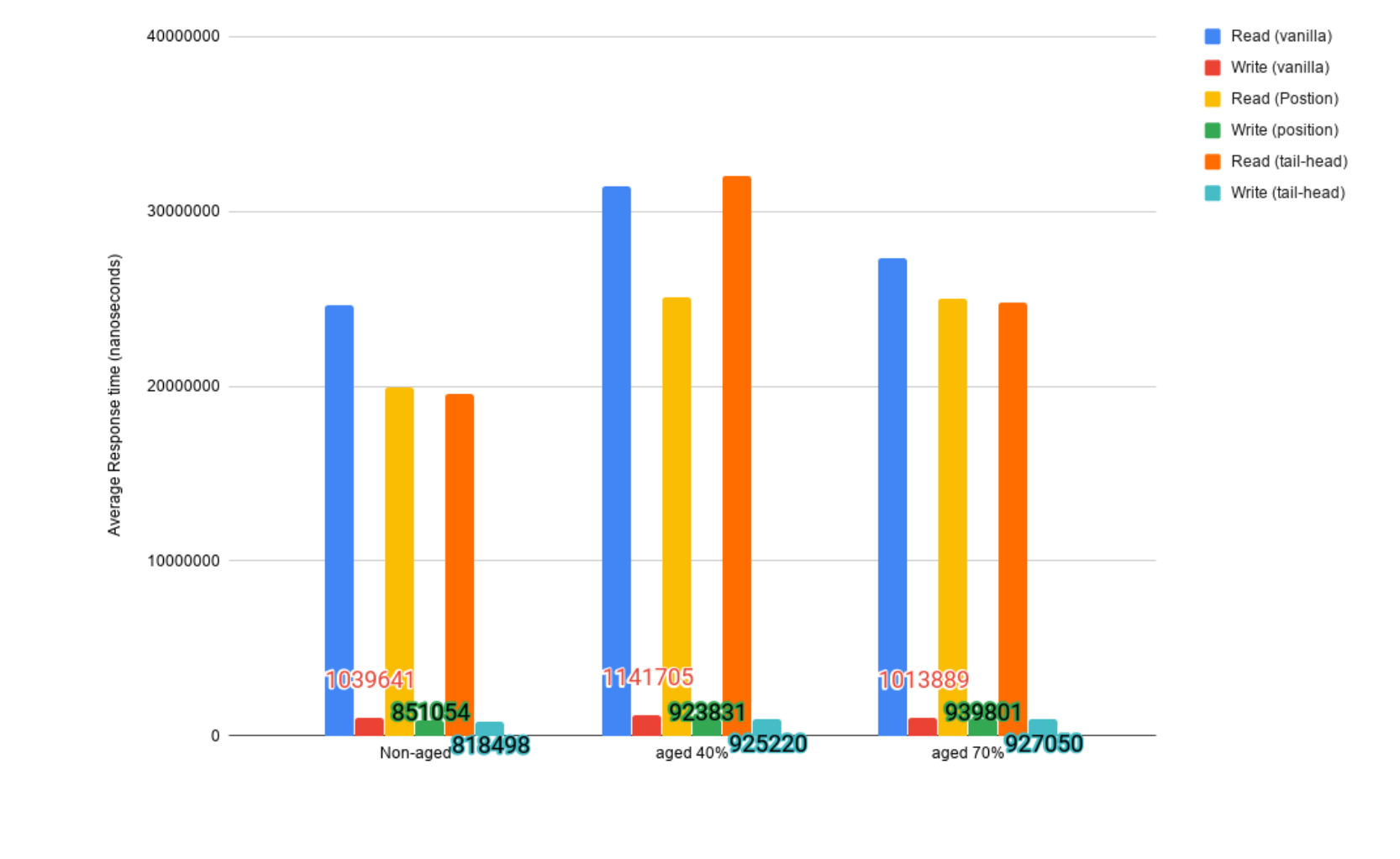

DRAM 16MB, Trace: MSNFS, Page Size: 8KB

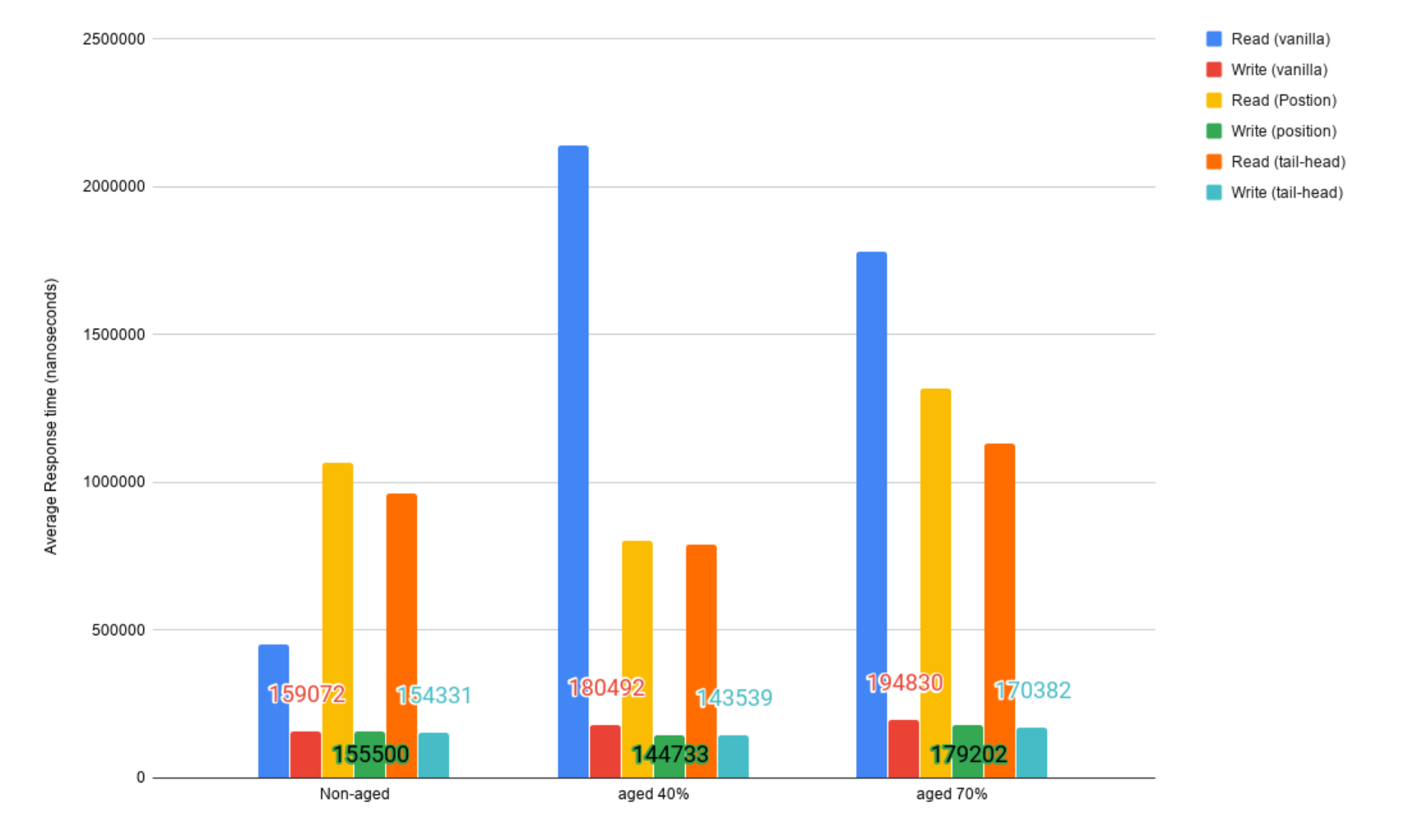

DRAM 16MB, Trace: MSNFS, Page Size: 4KB


DRAM 16MB, Trace: MSDTR, Page Size: 8KB


DRAM 16MB, Trace: MSDTR, Page Size: 4KB

The timestamp is scaled up for traces by 2x,4x, the result shows that the algorithm also works in those cases too. Writing time is reduced by ~10% for each trace, significant decrease found when both the page size (more partial page can be found) and DRAM size is increasing. Only the write response time is shown as that is our interest of reduction.


DRAM 32MB, Trace: MSNFS, Page Size: 8KB


DRAM 16MB, Trace: MSNFS, Page Size: 8KB

PC-LRU algorithm works well in aged condition, here SSD was aged by 40%, 70% and result shows huge improvement by ~20%

Existing Work

- PC-LRU has on average 9-19.5% performance improvements, MCA only reduces request latencies by 1.2-2.7%.
- PC-LRU is more aggressive in replacing full pages to reduce the total number of requests than MCA.
- MCA only shifts one position in the LRU list to delay the eviction of the partial page, whereas PC-LRU can select the P_{insert} in two positions.

Conclusion and Future Work

PC-LRU tries to mitigate unaligned write effects by keeping partial pages in buffer for longer time so that those can convert into full pages.
Future works:
Implementing in VSSIM emulator and incorporating with Multi-Version FTL to reduce extra read-modify-write operations when PC-LRU cannot convert partial pages to full pages.

POSTER ID: 2185