# RocksDB on Intel Optane PMEM

peifeng.si@intel.com

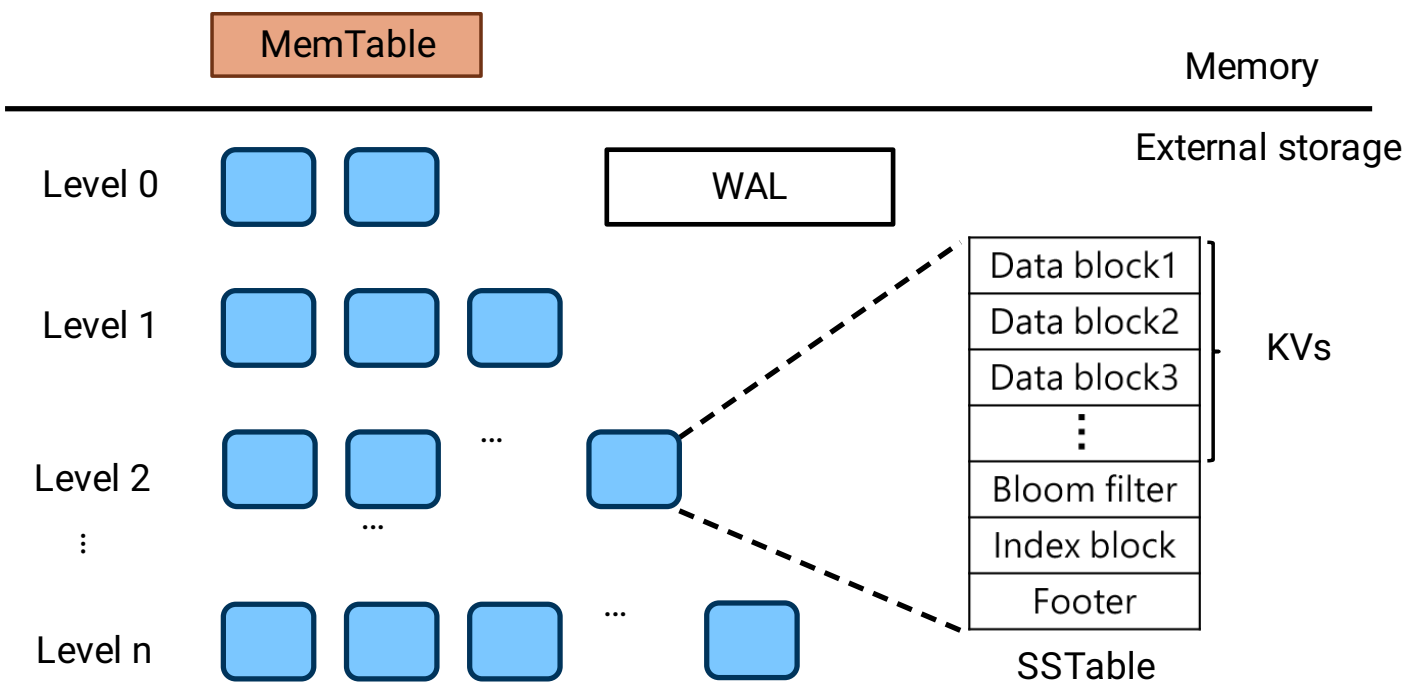
# Agenda

- RocksDB Architecture
- Optane Persistent Memory Overview
- Exploration of RocksDB on PMEM for Write
  - Multiple DB Paths
  - WAL on PMEM
  - Key-value separated store
- Exploration of RocksDB on PMEM for Read
  - Persistent Block-Cache on PMEM
  - Block-Cache on PMEM
  - Dynamic LRU
  - Fine-grained Read Block

# RocksDB Architecture

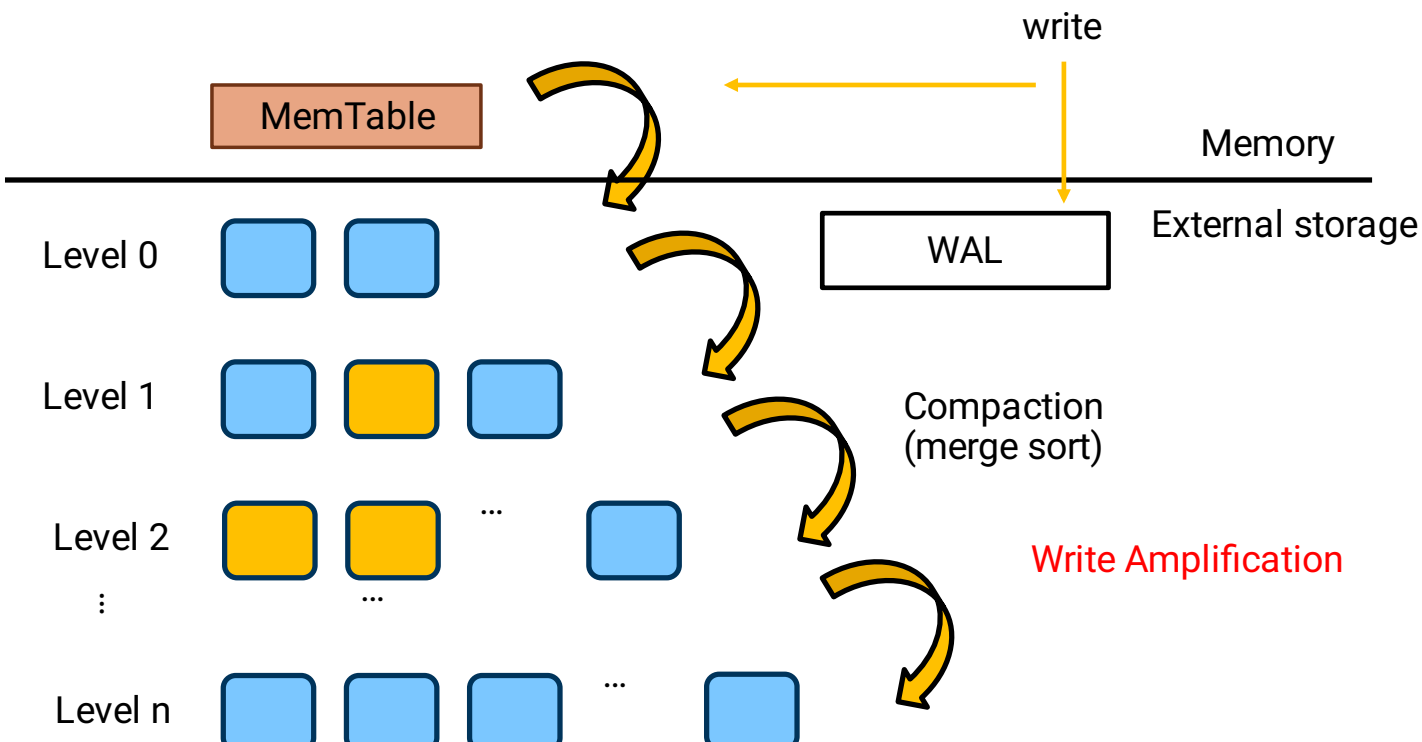
# RocksDB Architecture – Level Structure

- Ordered in each level
- Size increased by a ratio (normally 10)



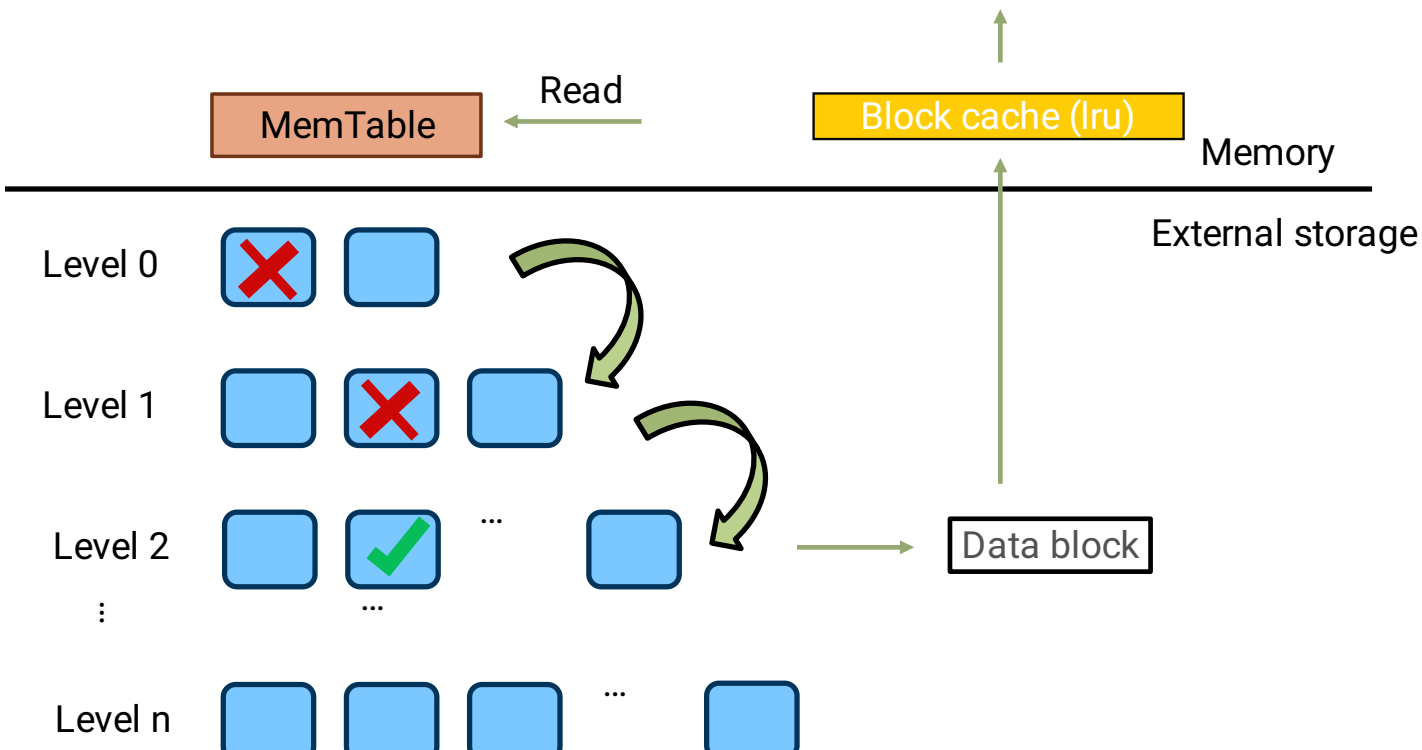
# RocksDB Architecture – Data Write

- Data write to WAL for reliability, then memtable, then compact to L0, L1, …



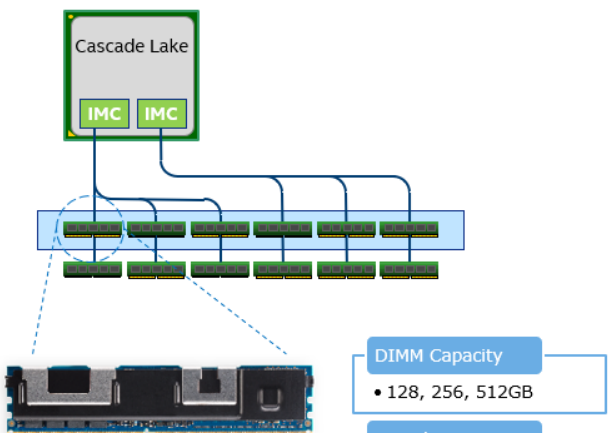
# RocksDB Architecture – Data Read

- Search data level by level
- Cache data in block cache



# Optane Persistent Memory Overview

# Key Features and Performance

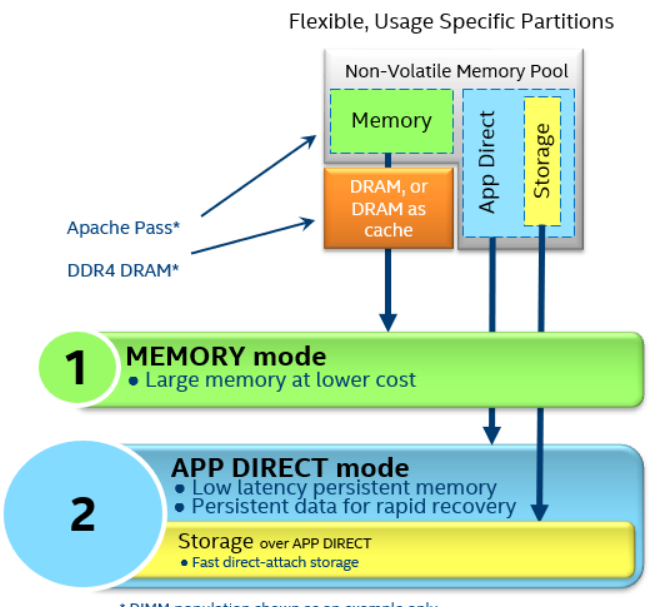



- DDR4 electrical & physical
- Close to DRAM latency
- Cache line size access

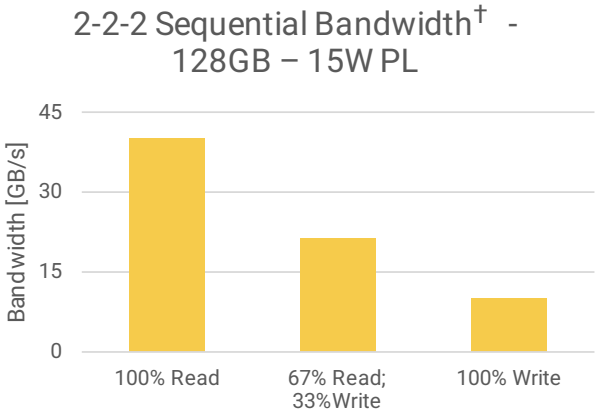
**DIMM Capacity**
- 128, 256, 512GB

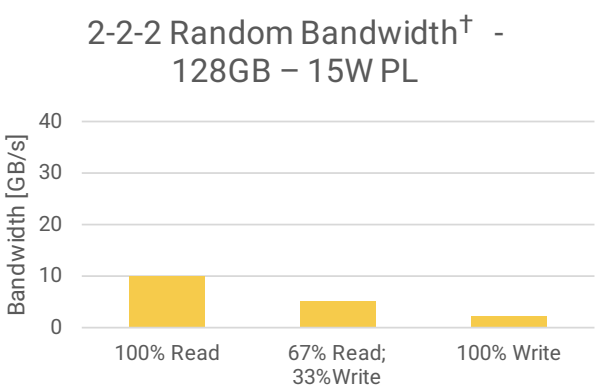
**Speed**
- 2666 MT/sec

**Platform Capacity**
- 6TB (3TB per CPU)




* DIMM population shown as an example only.



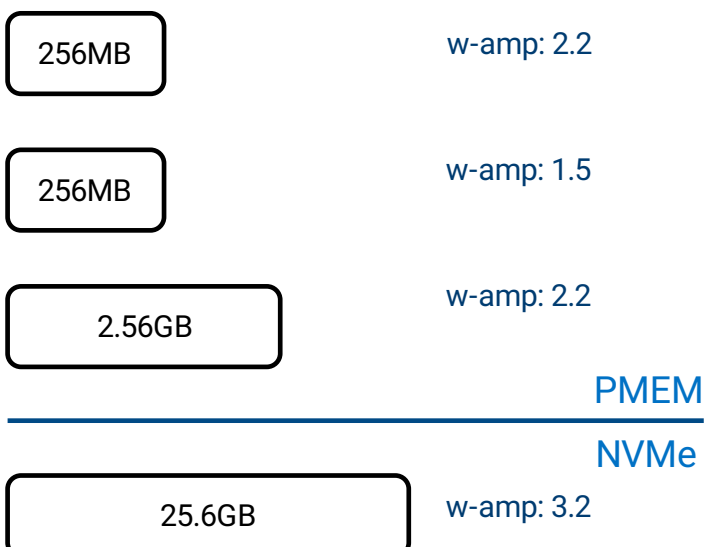



# RocksDB on PMEM for Write

# Multiple DB Paths

- Multiple paths is friendly to PMEM
  - Low levels go through most write amplification of total. (fully utilize PMEM's write BW)
  - Low levels contains the newest data. ( fully utilize PMEM's read BW)
  - PMEM capacity is large enough for low levels.
- Configuration example
  - db_paths=[{"/pmem_path", 3GB}, {"/hard_drive", 100GB}]
  - Lower numbered levels will be placed earlier in the db_paths vector
- Benchmark
  - data stored in both pmem and nvme, the size is 3G, 27G respectively.

| Case | Write ops/ sec | Write p99 latency (us) | Read ops/ sec | Read p99 latency (us) |
|---|---|---|---|---|
| One path | 76,132 | 4,135 | 340,825 | 1,765 |
| Multi-path | 125,714 | 2,656 | 451,221 | 1,218 |

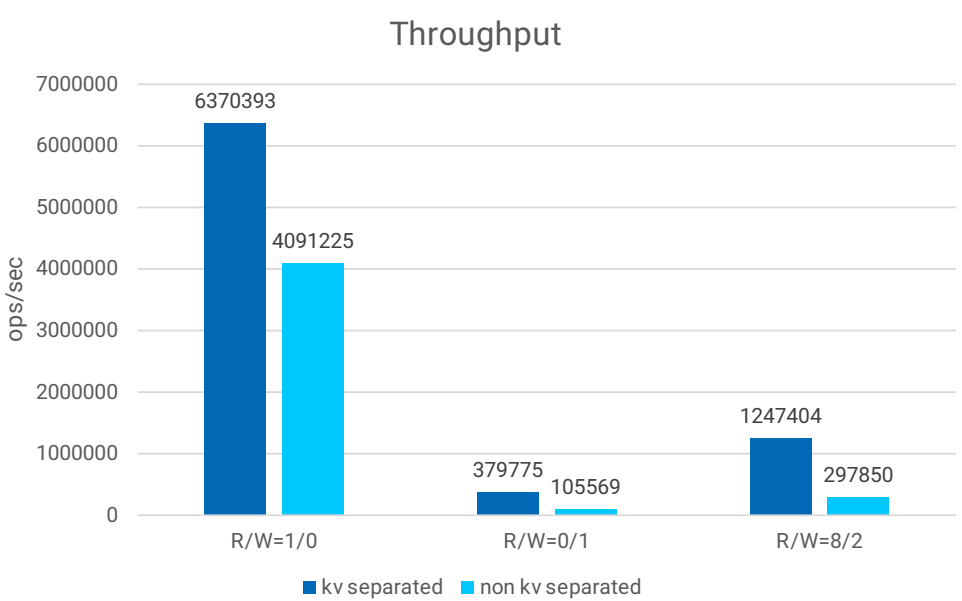


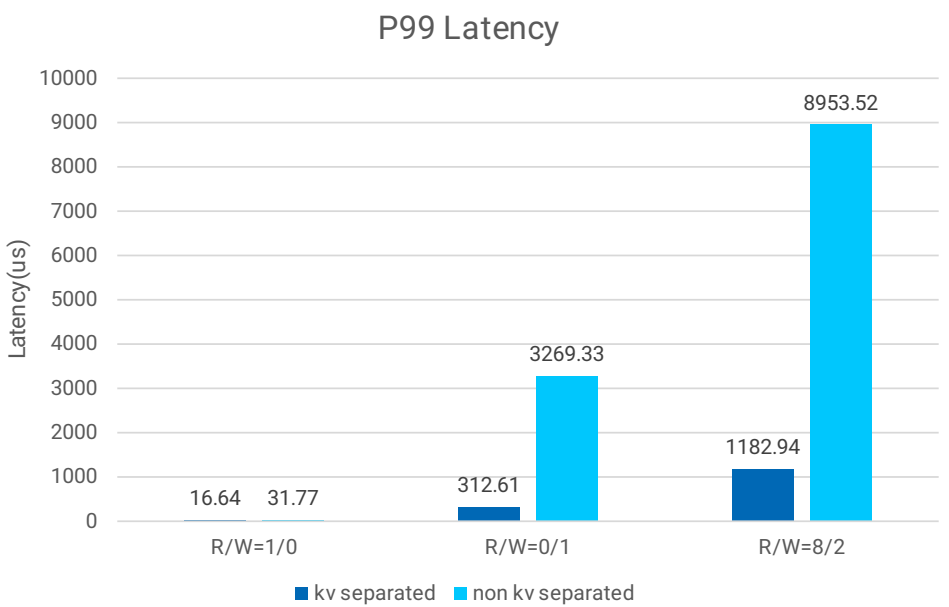
# WAL on PMEM

| Benchmark of fillrandom | | | | |
|---|---|---|---|---|
| Batch size | Key/Value size (bytes) | WAL on NVMe | WAL on PMEM | Ops improved |
| 1 | 16/128 | 470,083 | 478,766 | 1.0 X |
| 10 | 16/128 | 2,399,145 | 3,756,818 | 1.6 X |
| 20 | 16/128 | 2,351,397 | 4,815,041 | 2.1 X |
| 1 | 128/1024 | 205,852 | 463,737 | 2.3 X |
| 10 | 128/1024 | 202,675 | 771,872 | 3.8 X |
| 20 | 128/1024 | 206,220 | 730,095 | 3.5 X |

- WAL on PMEM can get better performance, especially for the workloads with large value size and high batch size.

# Key-Value Separated Store

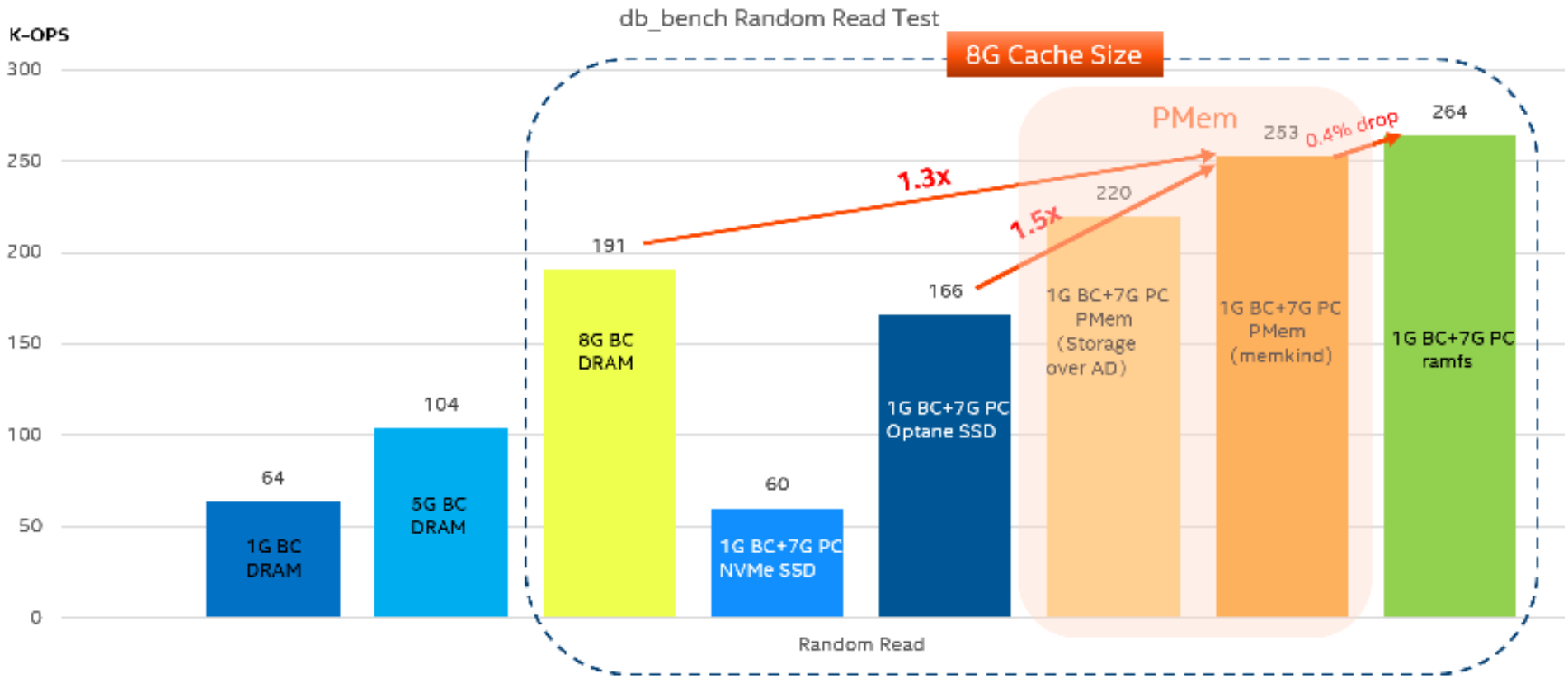



- Key-value separated RocksDB can get higher throughput and lower latency.
- Test Configuration: 15 GB dataset; 20% block-cache; 1 KB value size; 64 threads

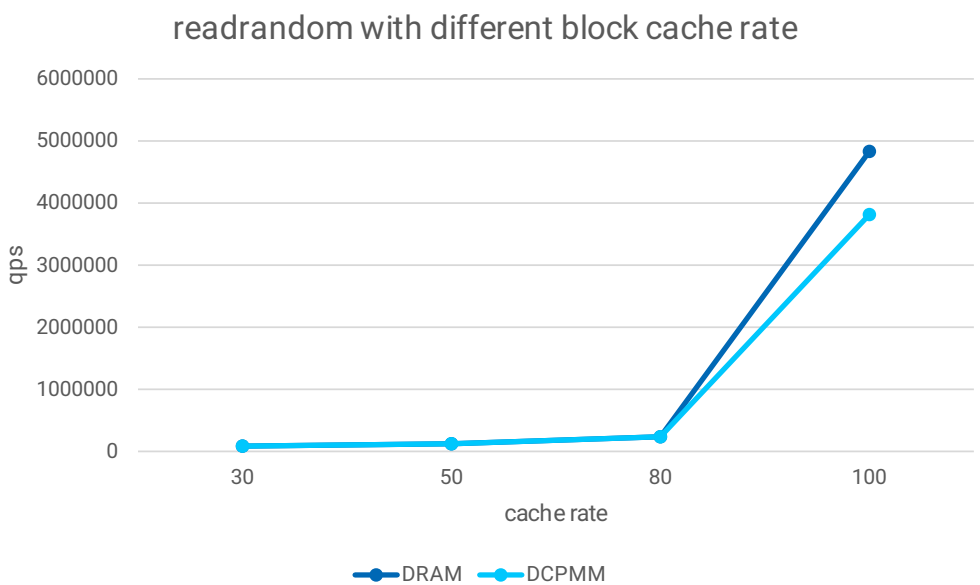
# RocksDB on PMEM for Read

# Persistent Block-Cache on PMEM




❖ Test Configuration: 10 GB dataset on S3700;  NVMe is P3520; 32 read threads

# Block-Cache on PMEM

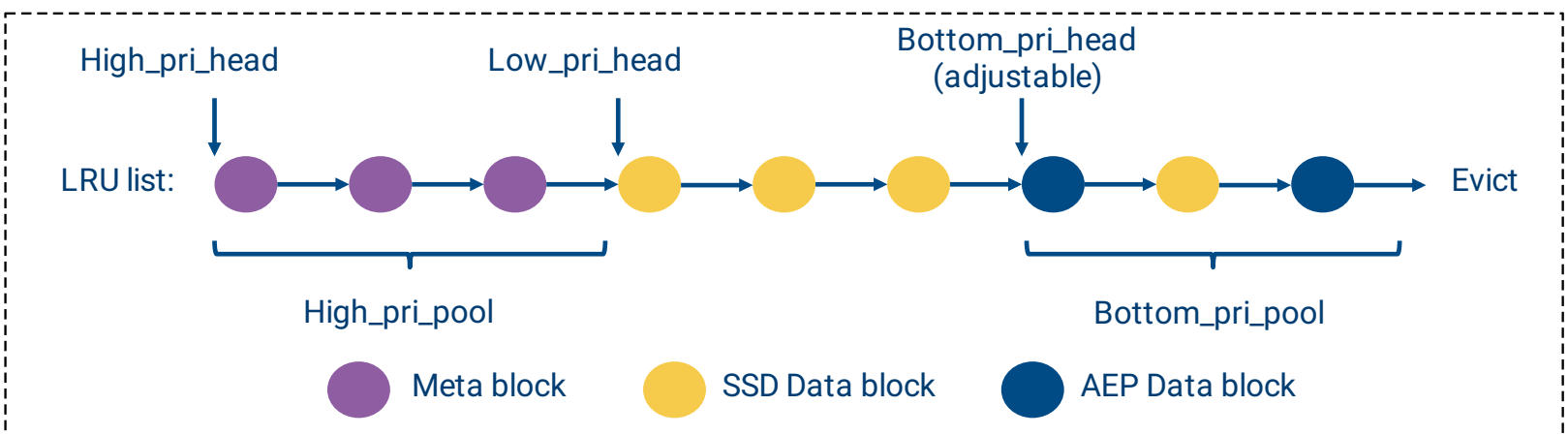



- Block cache is significant to read performance.
- Replace DRAM by PMEM for block cache almost has no performance degradation.
- PMEM can provide large block cache.

| Cache rate | 100 | 80 | 50 | 30 |
|---|---|---|---|---|
| DRAM | 4805312 | 221216 | 95049 | 68948 |
| DCPMM | 3800702 | 217703 | 94490 | 68143 |

# Dynamic LRU priority for PMEM blocks

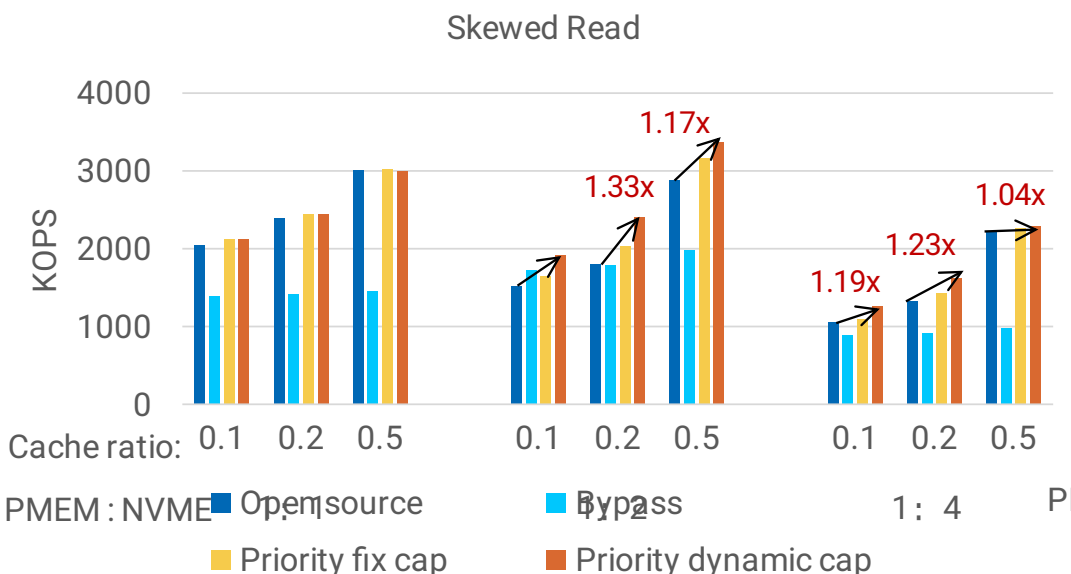
- Idea one: bypass block-cache for data blocks in PMEM.
  - Only cache data blocks that stored in slow devices.
  - When block-cache is not close to full, disable bypass feature.
  - Cons: Inefficient while blocks in PMEM are frequently accessed.
- Idea two: block from PMEM SST files has low priority in block-cache.

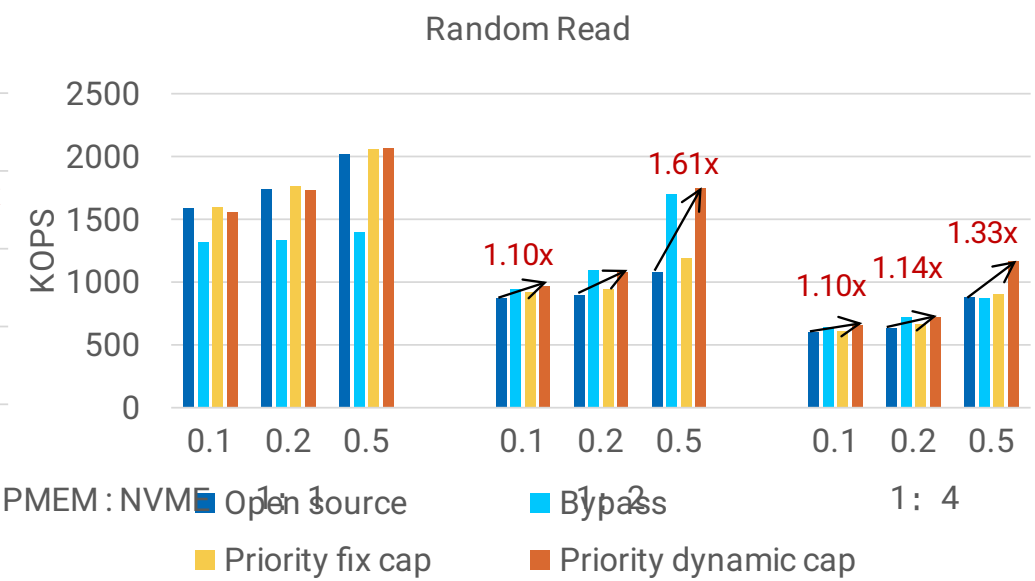



  - Blocks from PMEM SST files are inserted into block-cache with lower priority, so they are more likely to be evicted.
  - Dynamically adjust bottom priority cache capacity according to workloads pattern.

# Dynamic LRU priority for PMEM blocks

- Benchmark
  - Read performance (PMEM + NVME)

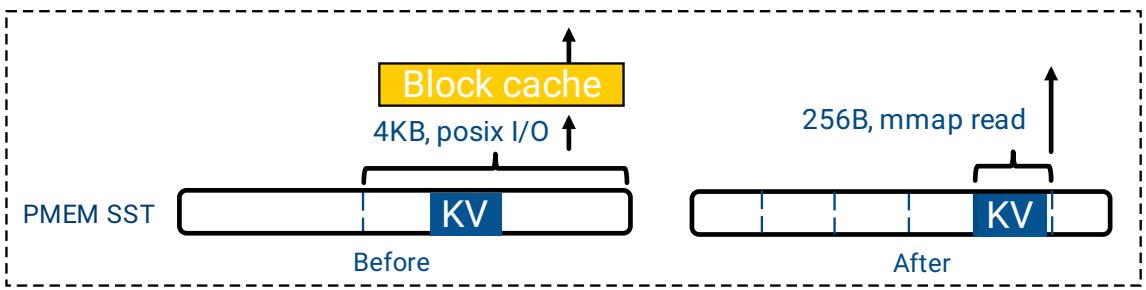


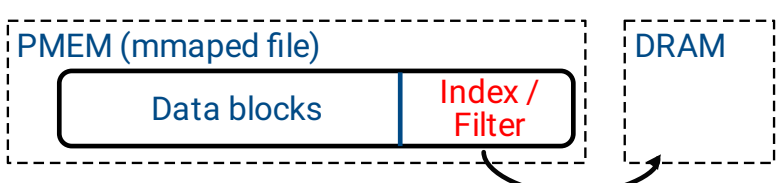

  - Result
    - Achieves 10%~60% performance improvement on both random and skewed reads.
    - Compatible with existing data.

# Fine-grained read for PMEM

- Reduce I/O granularity
  - Organize PMEM SST files with small block size, and read blocks with mmap(), to reduce Read Amplification.
  - Bypass block cache as I/O latency is no longer bottleneck

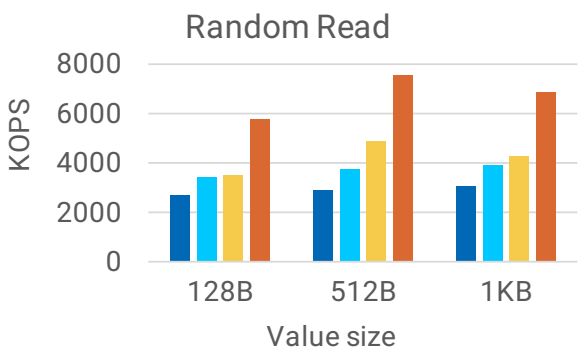


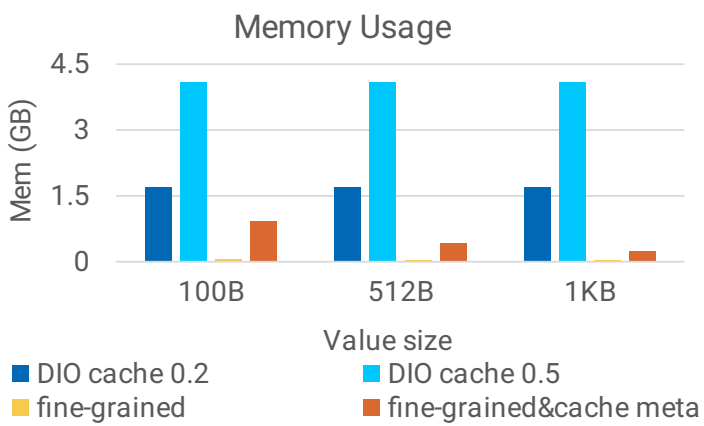

- Cache frequently accessed metadata
  - With mmap, the metadata of data block will be accessed multiple times from PMEM for a single read.

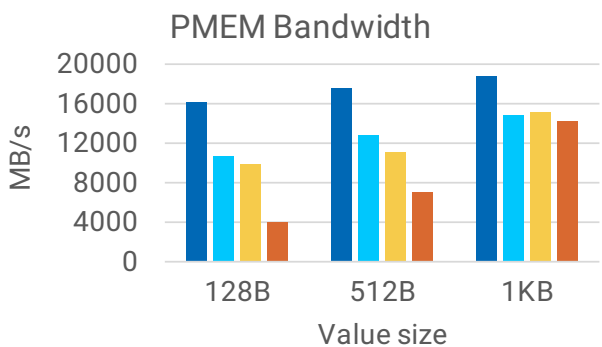


# Fine-grained read for PMEM

- Benchmark (PMEM Only)
  - Load 20GB DB on PMEM, with different value sizes, then issue reads.
  - Read performance / memory usage / bandwidth usage




- Result
  - Increase read throughput > 2x
  - Save DRAM usage > 10 x
  - Decrease the read bandwidth usage of persistent memory > 2.5x

# Thank you