OurRocks: offloading disk scan directly to GPU in write-optimized database system

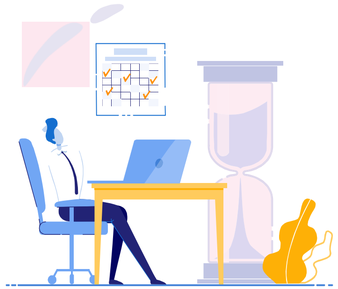
연세대학교 컴퓨터과학과 최원기

과제명: IoT 환경을 위한 고성능 플래시 메모리 스토리지 기반 인메모리 분산 DBMS 연구개발

과제번호: 2017-0-00477

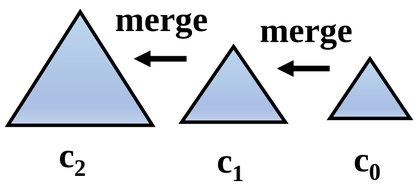
Index

Introduction



Recent Workload

- Hybrid Transactional /Analytical Processing



LSM-tree

- Write-optimized storage engine

Modern Hardware

- NVMe SSD
- GPU

Recent Workload

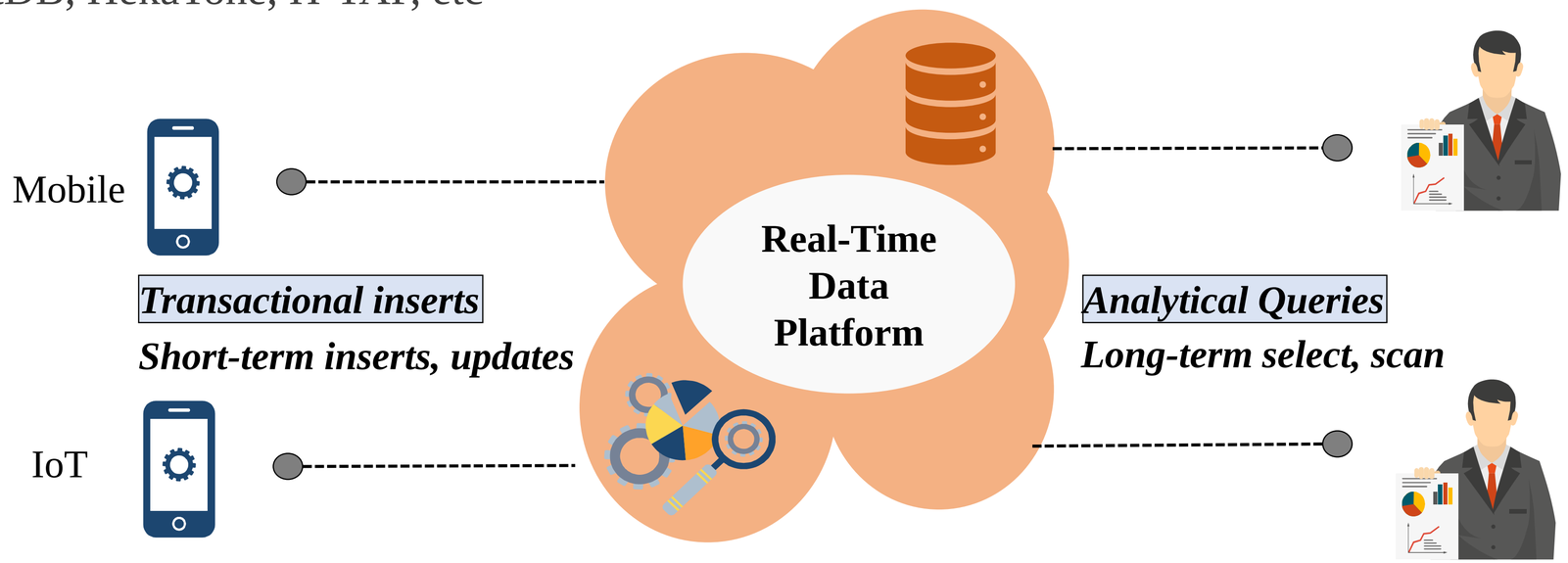
Hybrid Transactional/Analytical processing

■ Big Data analytics application

- Real-time recommendation service, fraud detection service
- Mobile, Internet of Thing

■ Requirement for a new generation of data management systems

- Handling both Transactional and Analytical workload
- E.g. VoltDB, HekaTone, H^{2}TAP, etc



LSM-tree (1)

Log Structured Merge Tree

■ **Widely utilized in various systems (as Backend Storage)**

- RDBMS : MyRocks, MongoDB
- Big Data : Redis on Flash, Lightning DB(SK Telecom)
- Virtual currency : Ethereum

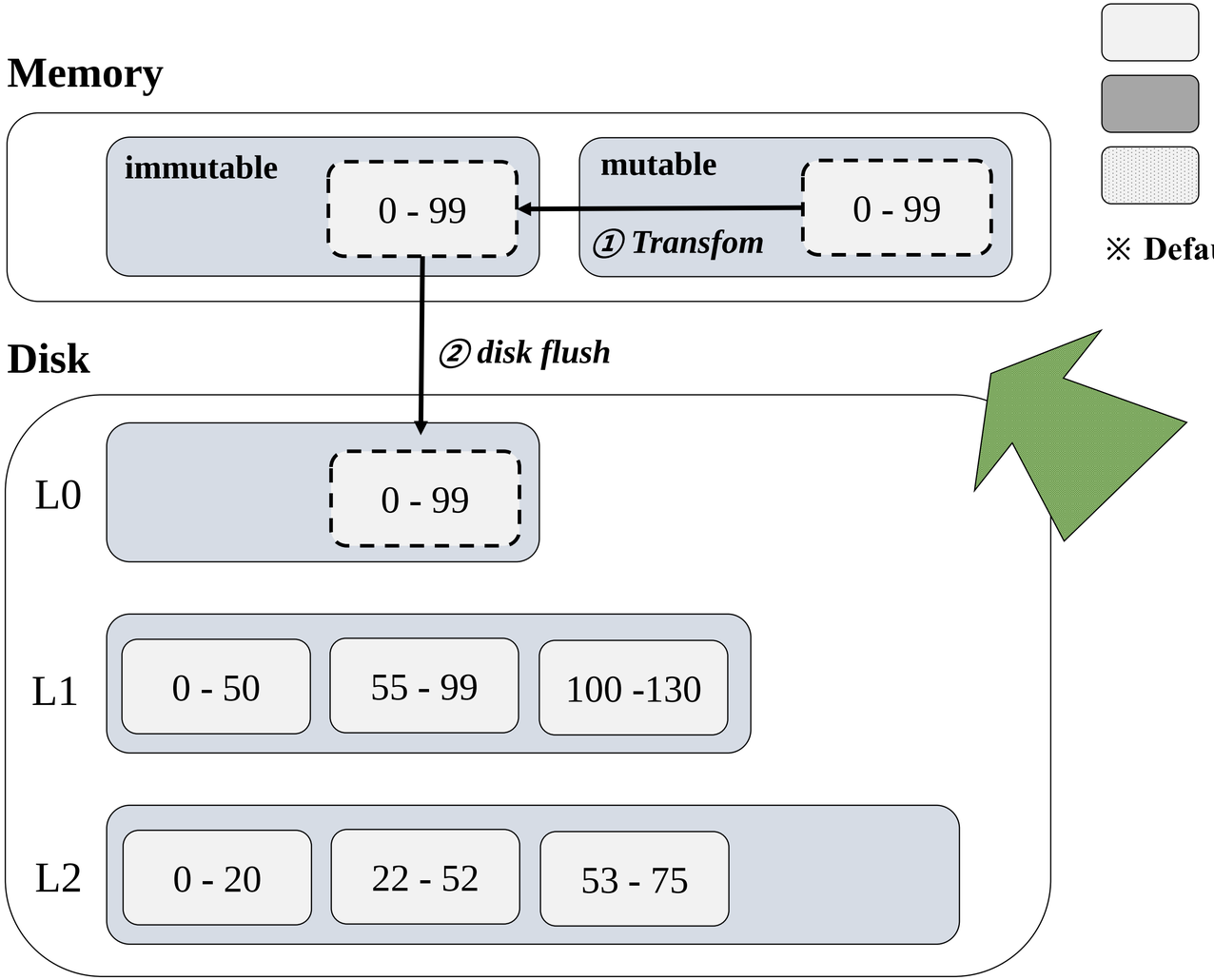
■ **Write and space friendly features**

- Designed for high bandwidth devices (e.g. flash drives)
- Great compression performance
- Less write amplification, which improves endurance of flash storage and overall throughput

**For this reason,
Facebook has migrated
their database to MyRocks**

LSM-tree (2)

Overall architecture of LSM-tree



※ Default size of SST file is 64 MBytes

Data Record Format

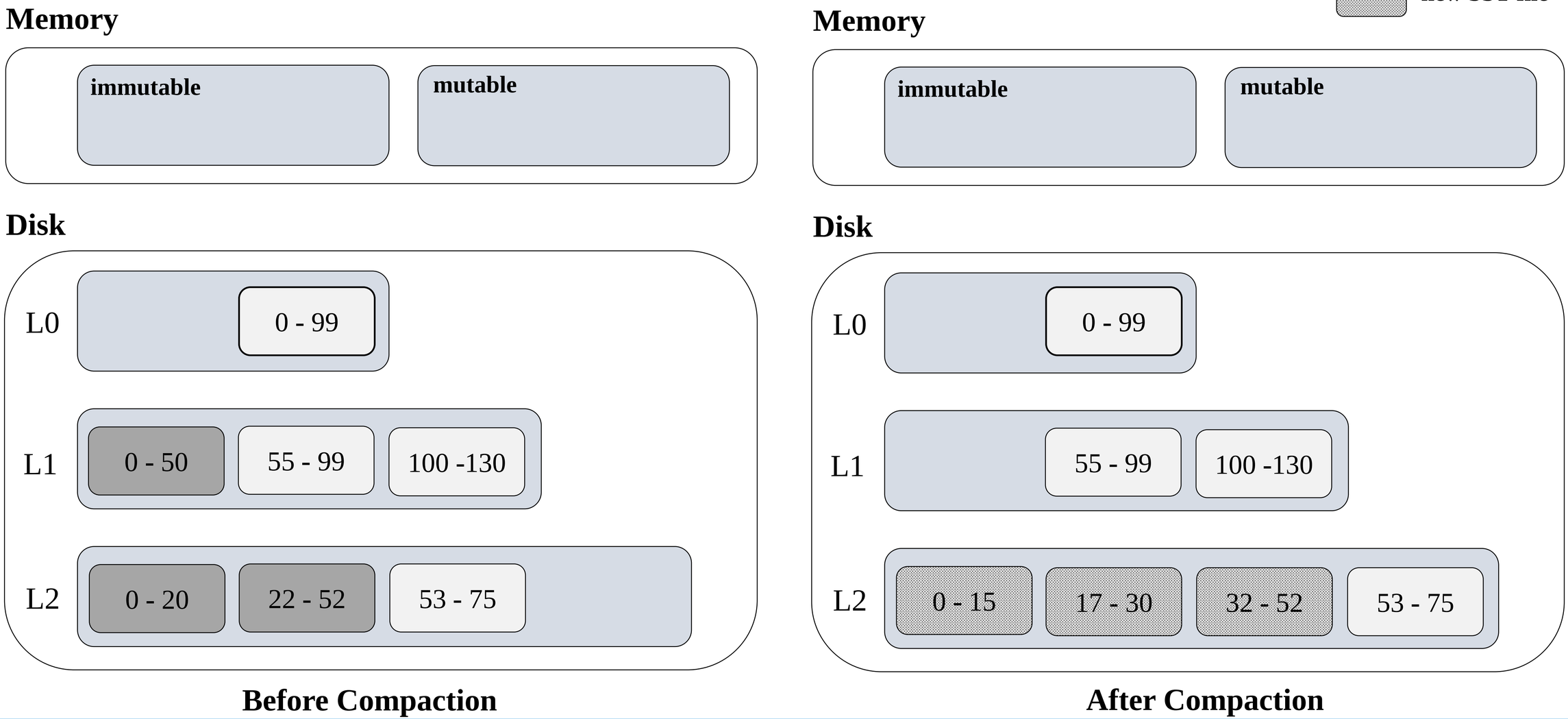
RocksDB key		RocksDB Value		RocksDB Metadata
Internal Index id	Primary Key	Rest Columns	Checksum	SeqID, Flag

Internal Index id : Auto-generated 4 byte(0 ~ 2^{32})

Relational Database record is transformed to the key-value pair of LSM-tree

LSM-tree (3)

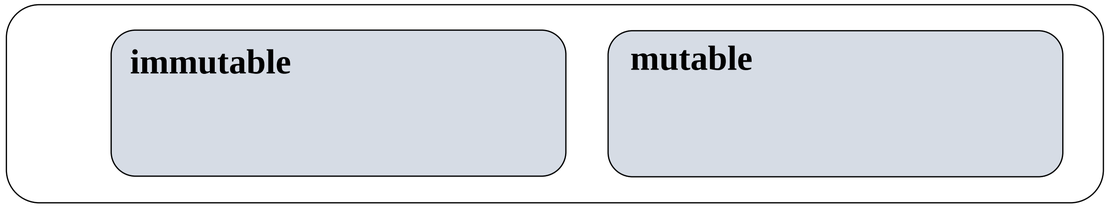
Compaction of LSM-tree



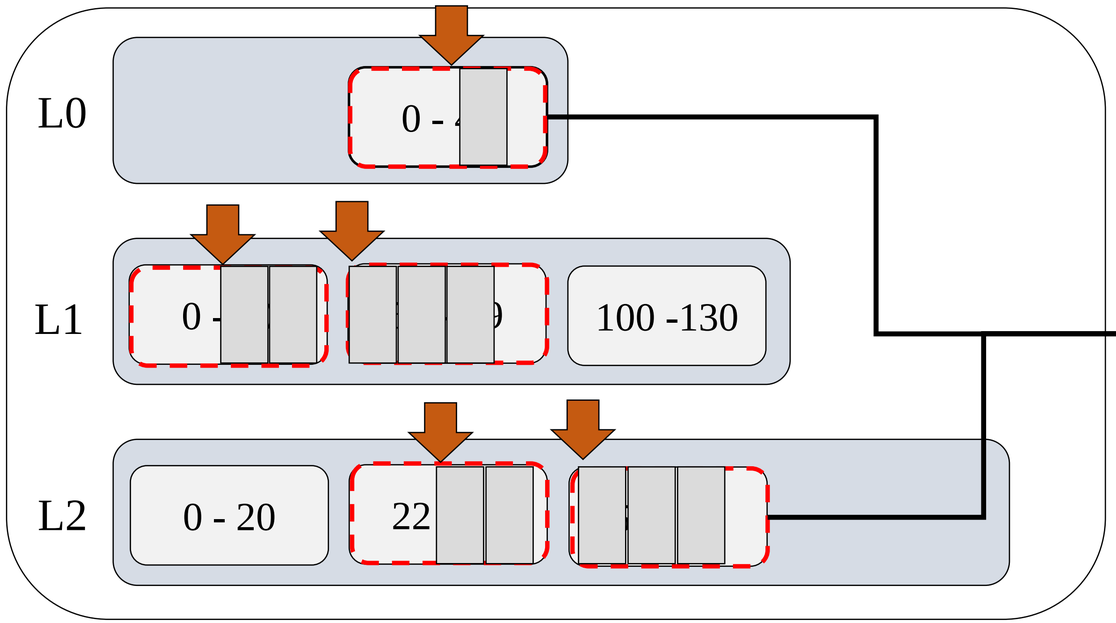
LSM-tree (4)

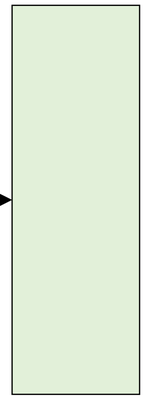
Table scan procedure of LSM-tree

Memory



Disk





Record Buffer

Data Record Format

RocksDB key		RocksDB Value		RocksDB Metadata
Internal Index id	Primary Key	Rest Columns	Checksum	SeqID, Flag

Internal Index id : Auto-generated 4 byte(0 ~ 2^{32})

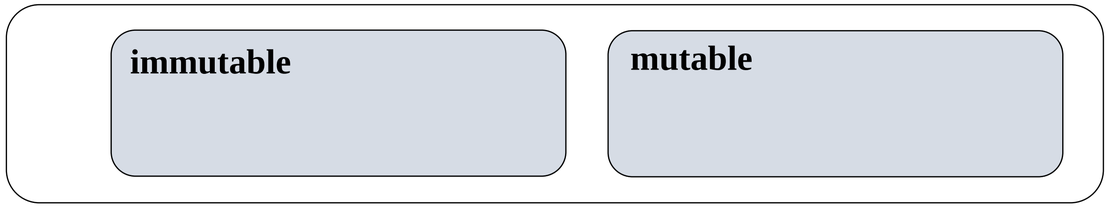
[Query] select from tbl1 where tbl1.columnTwo < 15;

Trigger Range Scan (e.g. scan in key range [30 ~ 70])

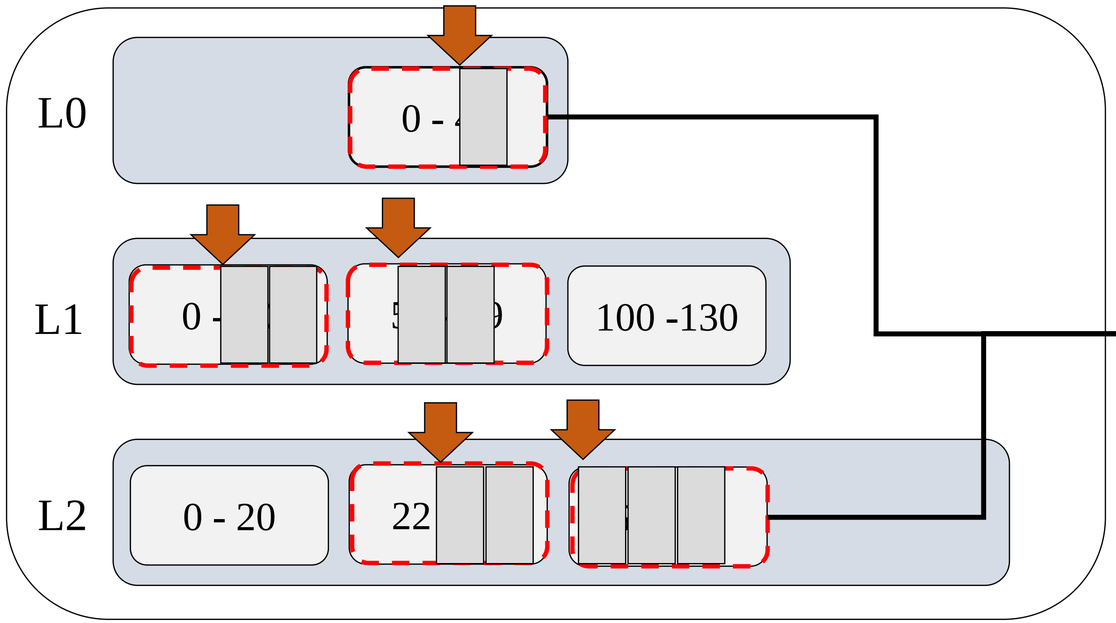
LSM-tree (5)

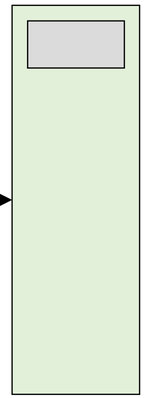
Table scan procedure of LSM-tree

Memory



Disk





Record Buffer

Data Record Format

RocksDB key		RocksDB Value		RocksDB Metadata
Internal Index id	Primary Key	Rest Columns	Checksum	SeqID, Flag

Internal Index id : Auto-generated 4 byte(0 ~ 2^{32})

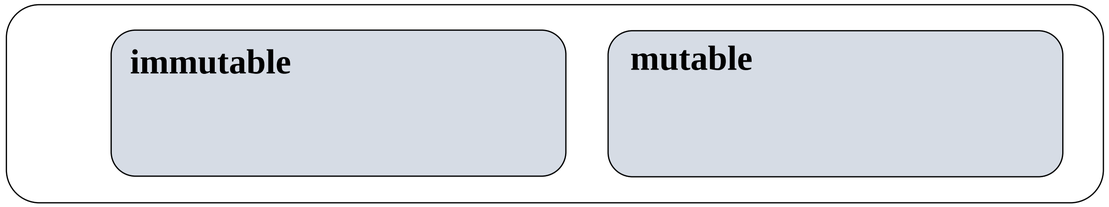
[Query] select from tbl1 where tbl1.columnTwo < 15;

Trigger Range Scan (e.g. scan in key range [30 ~ 70])

LSM-tree (6)

Table scan procedure of LSM-tree

Memory

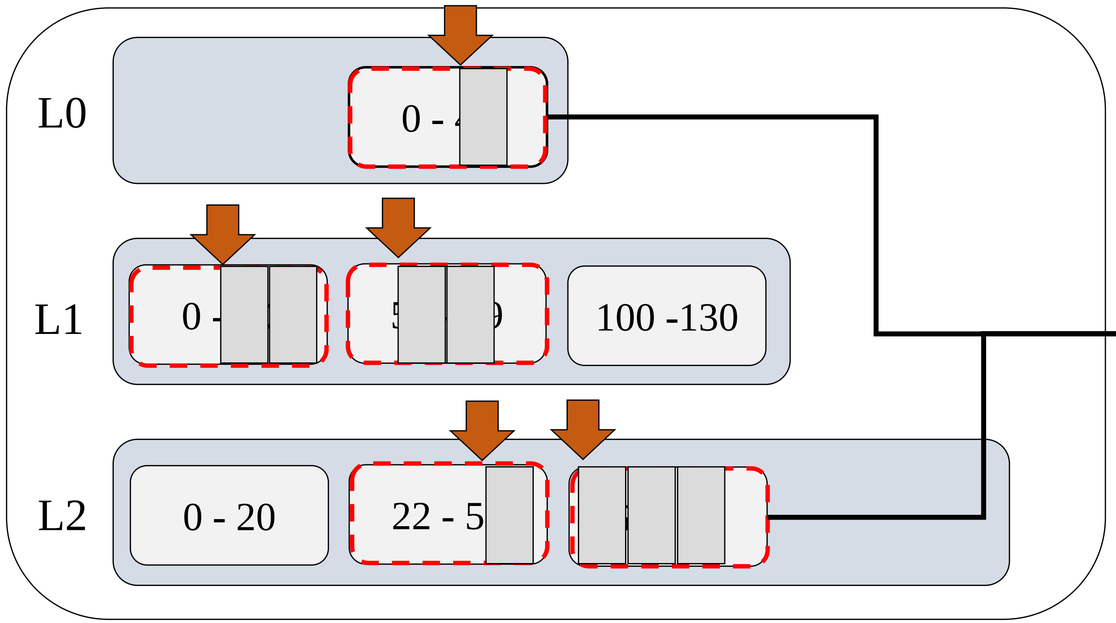


Data Record Format

RocksDB key		RocksDB Value		RocksDB Metadata
Internal Index id	Primary Key	Rest Columns	Checksum	SeqID, Flag

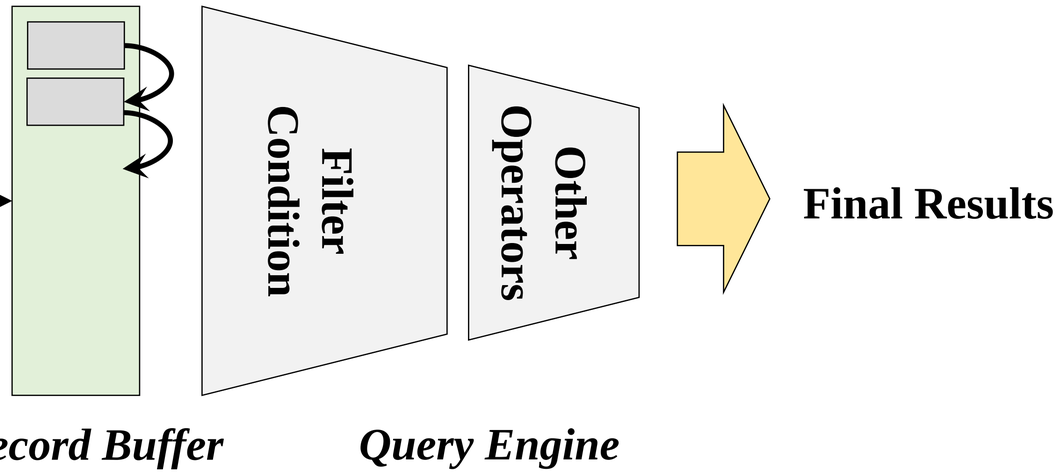
Internal Index id : Auto-generated 4 byte(0 ~ 2^{32})

Disk



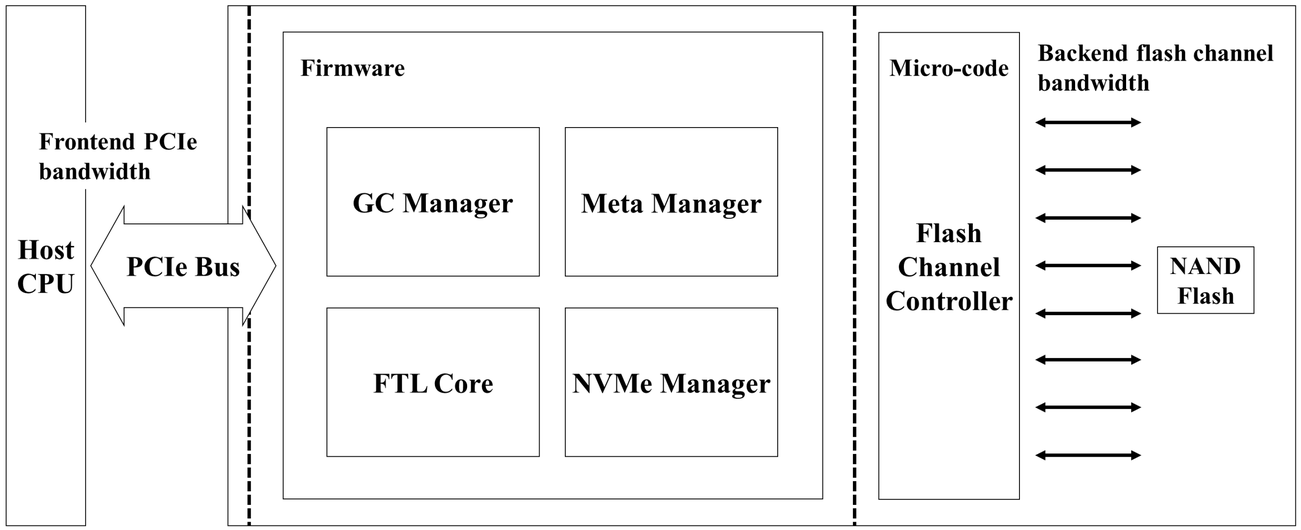
[Query] select from tbl1 where tbl1.columnTwo < 15;

Trigger Range Scan (e.g. scan in key range [30 ~ 70])



Limitation(1)

Non-volatile memory express SSD

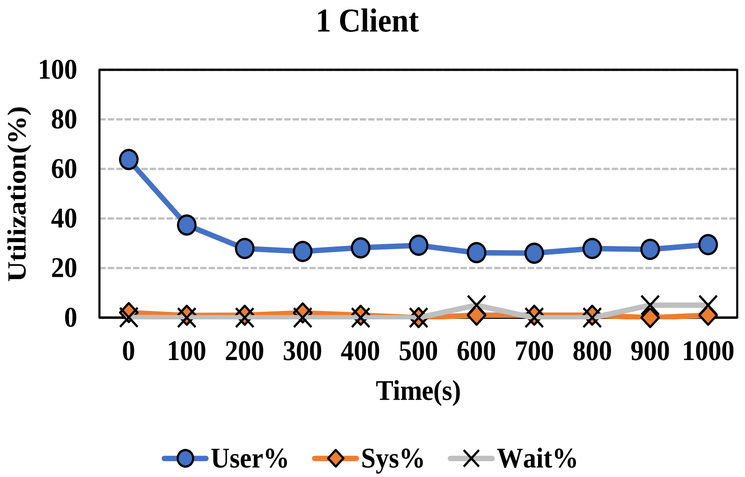


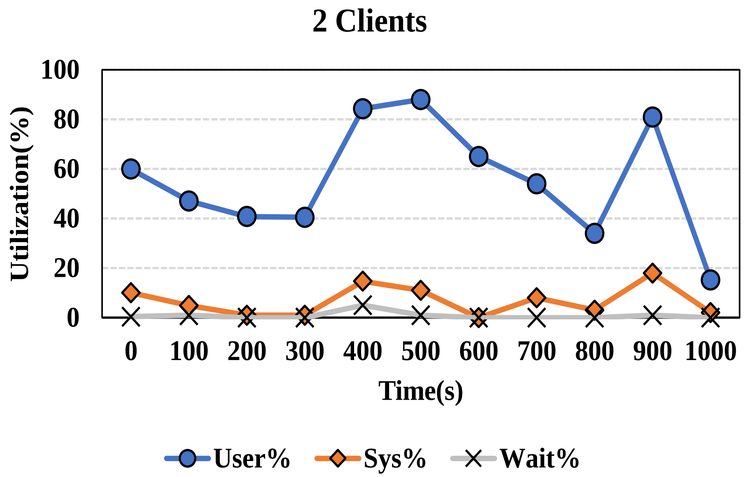
- **NVMe fully exploit the levels of parallelism possible in modern SSDs**
  - NVMe SSD serves multi-GB/s I/O rates
  - NVMe SSD is now being used as primary storage units in the enterprise
  - Nonetheless, **the general CPU processor makes limited use of the throughput and iops of NVMe SSDs**

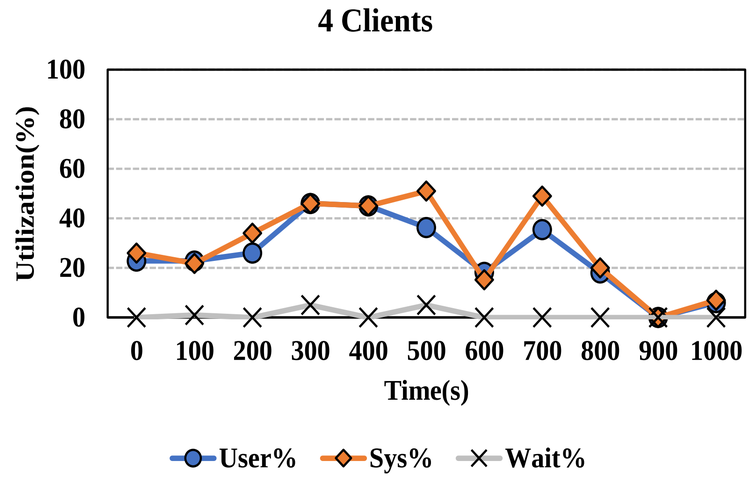
- **How about Scan operation in LSM-tree?**

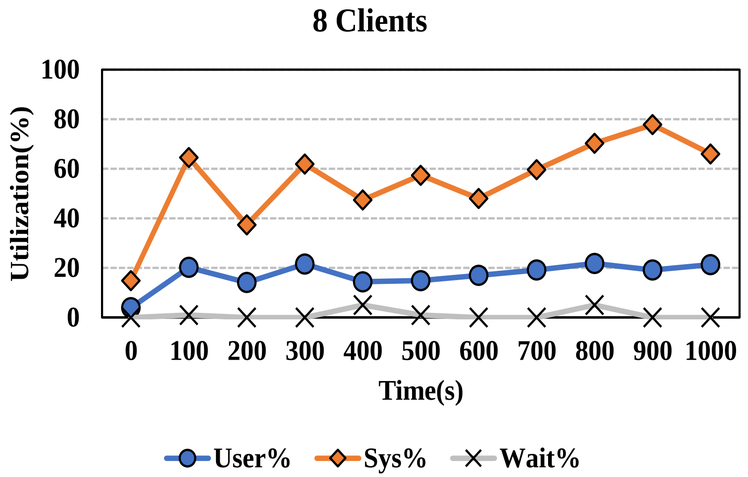
Limitation(2)

CPU intensive characteristic







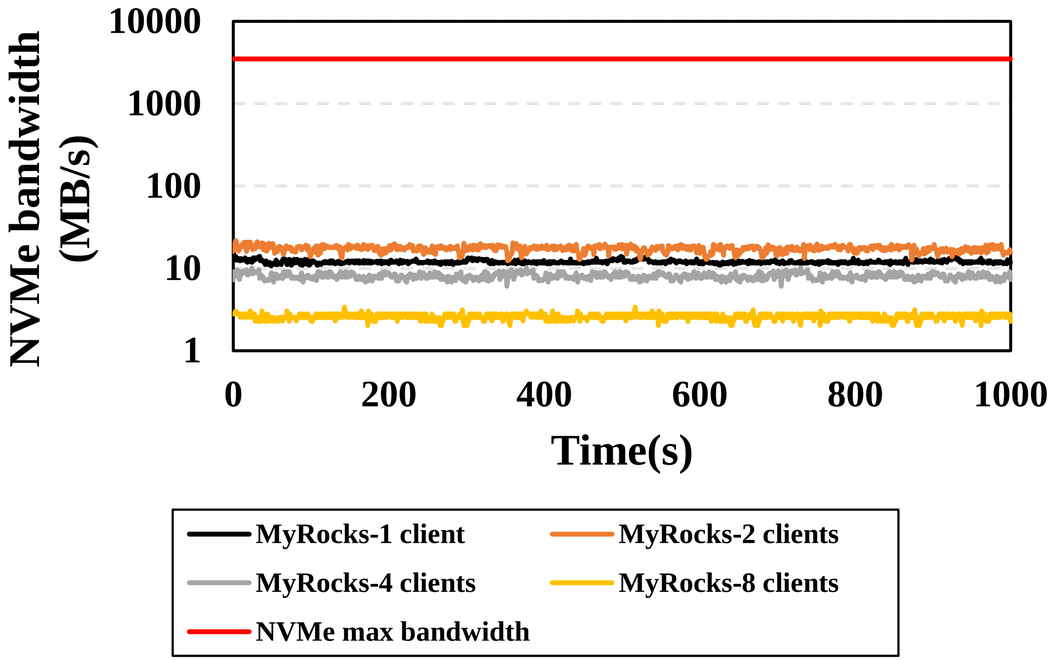


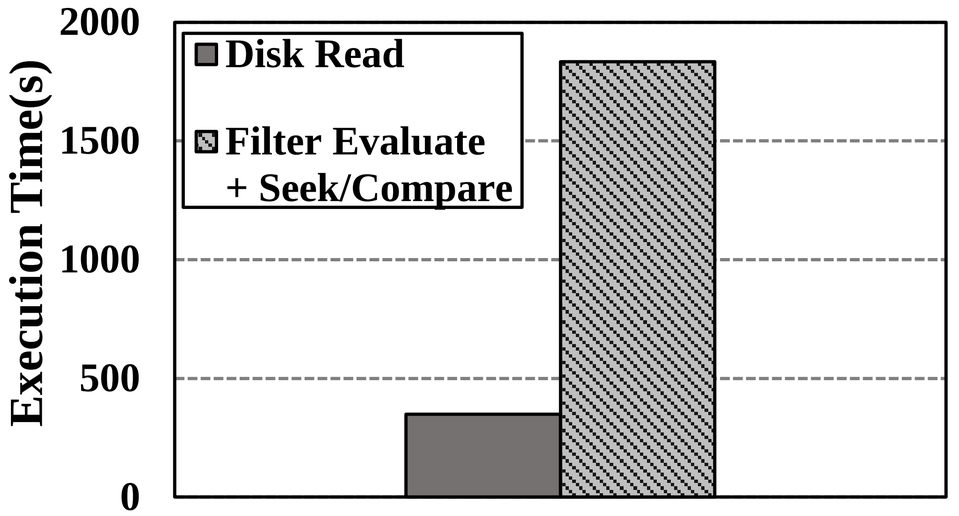
■ CPU utilization per core

- As the number of client is increased, the **CPU overhead from context switching sharply increases**
- CPU usage occupied by the user space and context switching is higher than I/O wait time

Limitation(3)

NVMe SSD utilization





■ NVMe bandwidth

- MyRocks poorly utilize the NVMe SSD's bandwidth (**only 1% of maximum bandwidth**)
- High CPU usage prevents the database from optimally utilizing the NVMe SSD bandwidth

■ Weight of operations

- MyRocks requires a considerable amount of time in **Seek, Compare, Filter evaluation** compared to the disk read
- CPU-driven query processing cannot keep pace with the data loading speed offered by NVMe SSD

Main Approaches

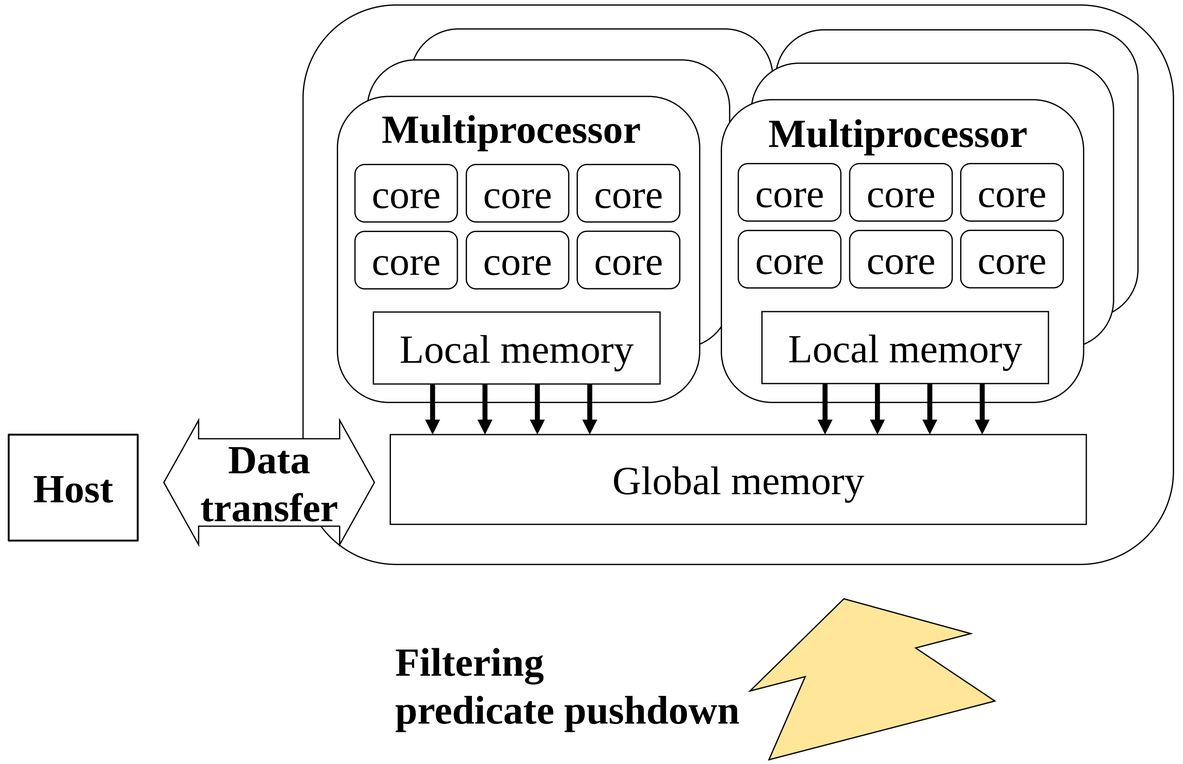
■ **Filtering predicate pushdown w/**

- Graphics Processing Unit (GPU)
  - ✓ General version of GPU acceleration
  - ✓ Optimized version of GPU acceleration
  - ✓ OurRocks that utilizes Direct Memory Access
- Advanced Vector Processing (AVX)

■ **An intuitive manner is …**

- Offloading the task of checking whether the data satisfies the condition of the queries to the available computing resources
- Parallel processing with Bulk loading

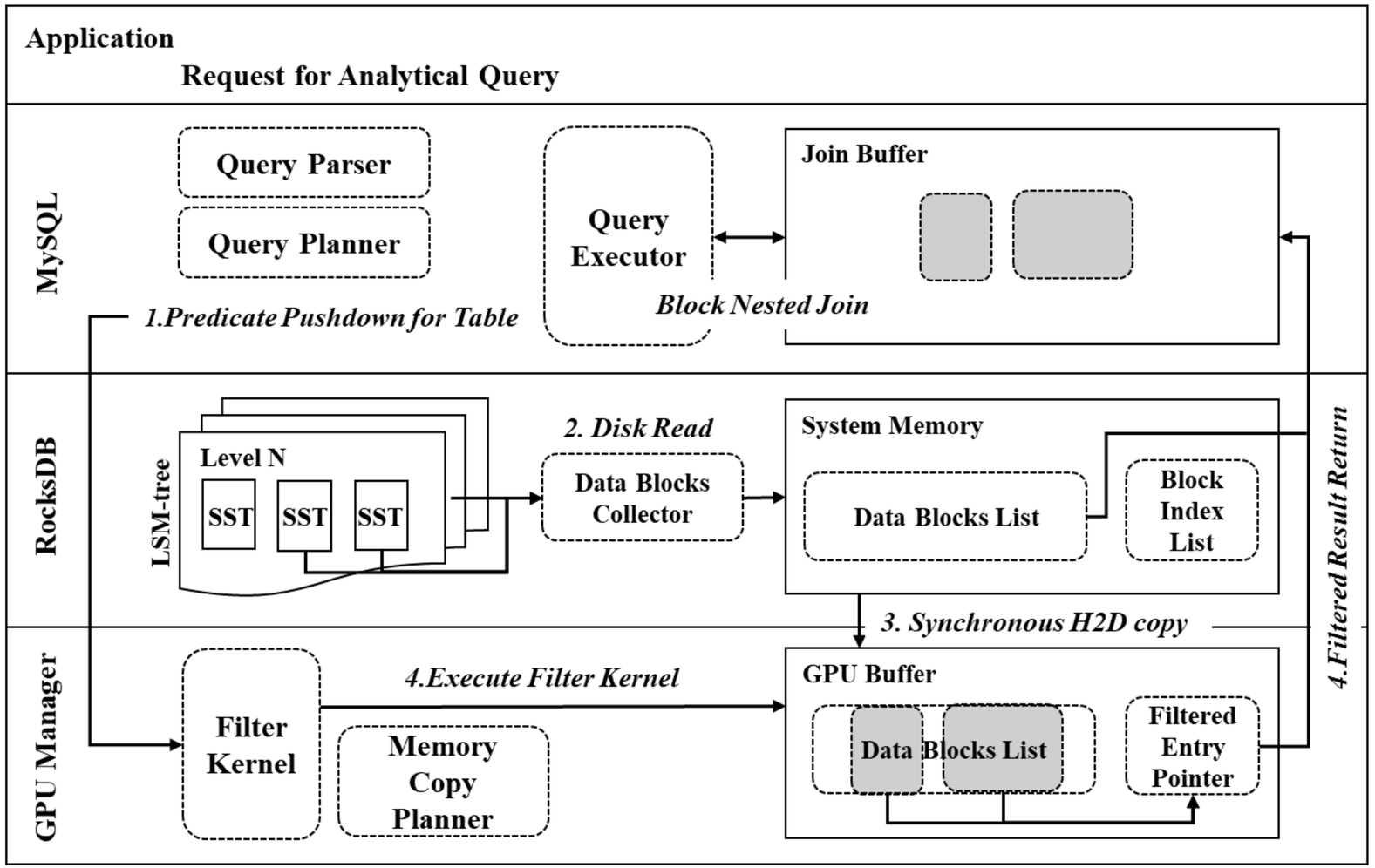
GPU Architecture



[Query] select from tbl1 where tbl1.columnTwo < 15;

GPU Acceleration (1)

General version

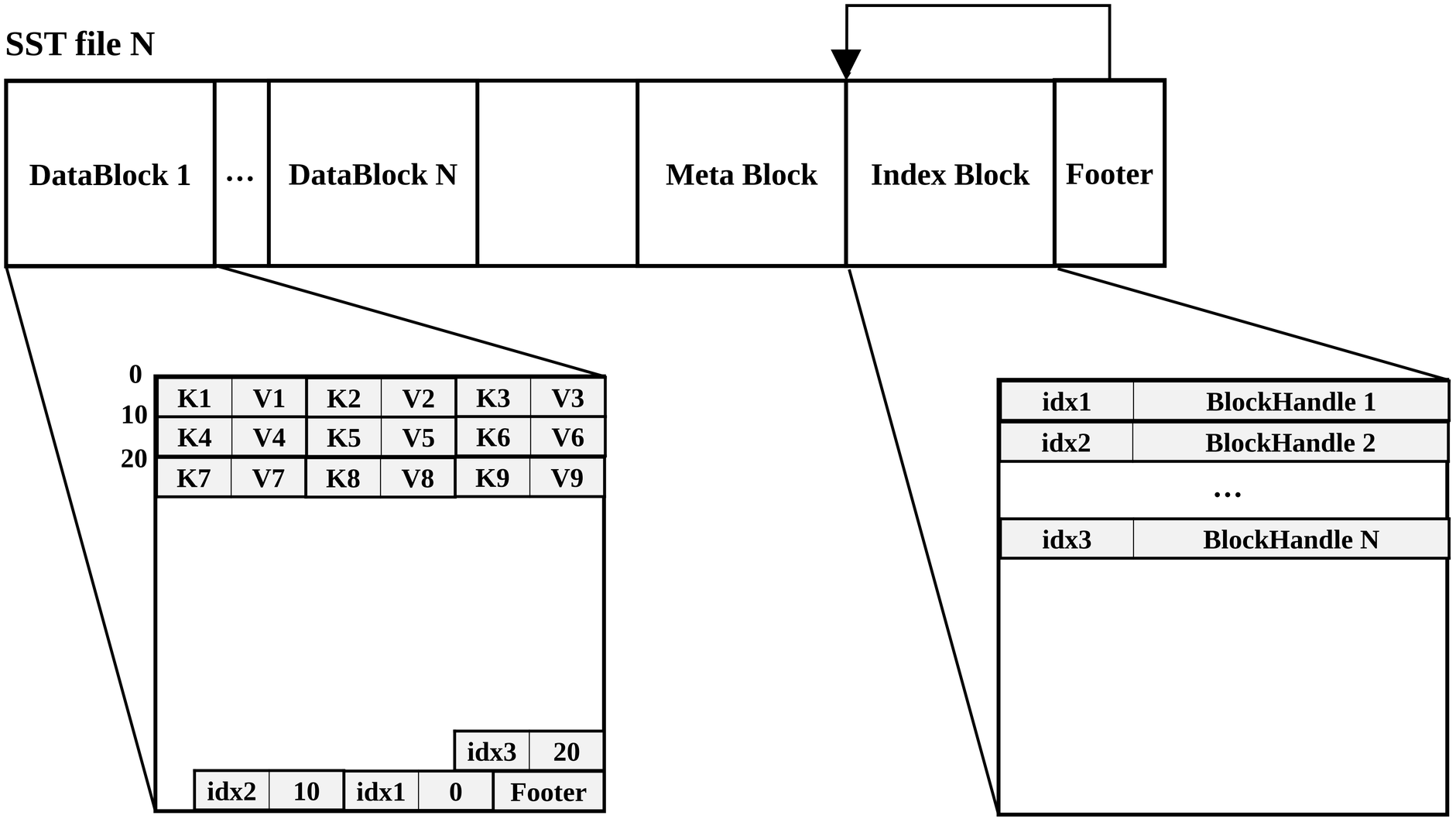


■ GPU accelerated Scan flow

1. Select target table to accelerate
2. Parsing the predicate
3. Collect data blocks
4. Synchronous host to device copy
5. Execute filter kernel
6. Return filtered results
7. Query execution with filtered results

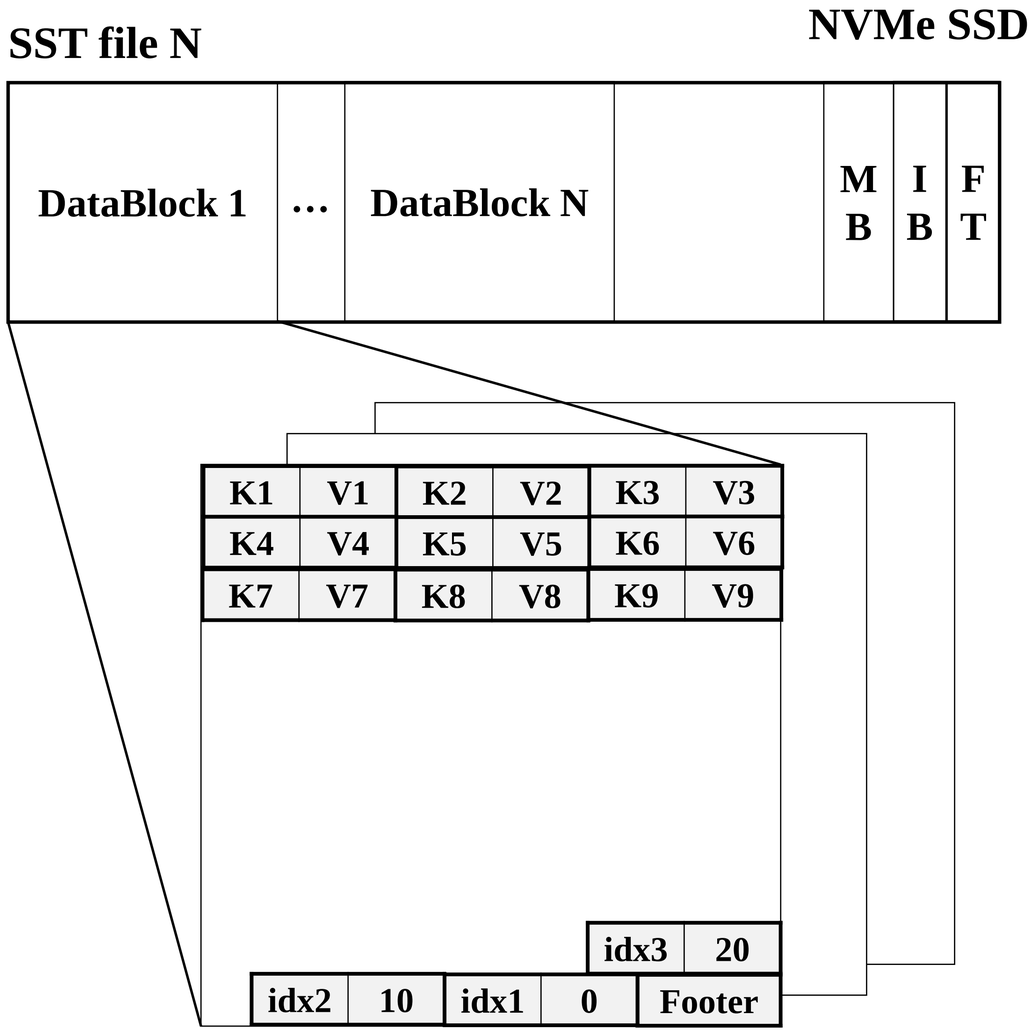
GPU Acceleration (2)

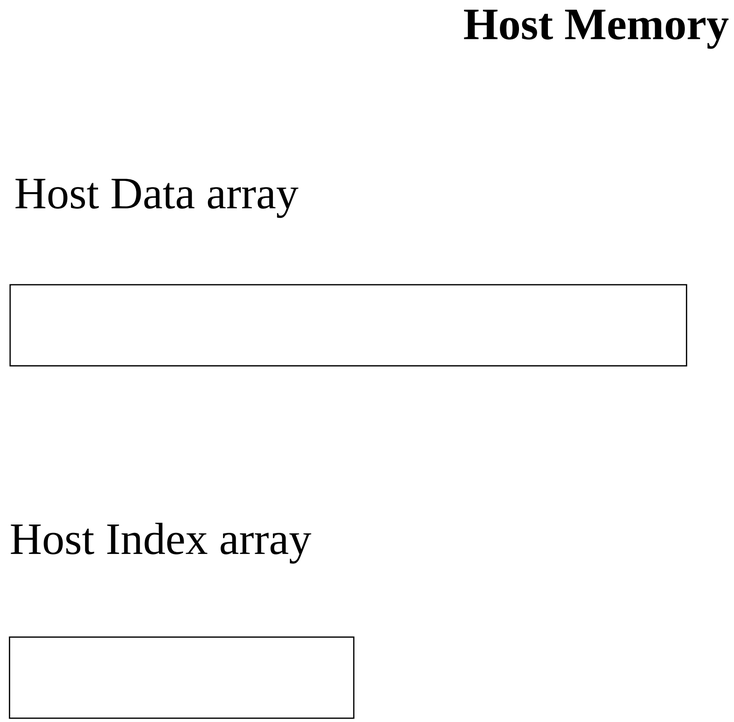
General version



GPU Acceleration (3)

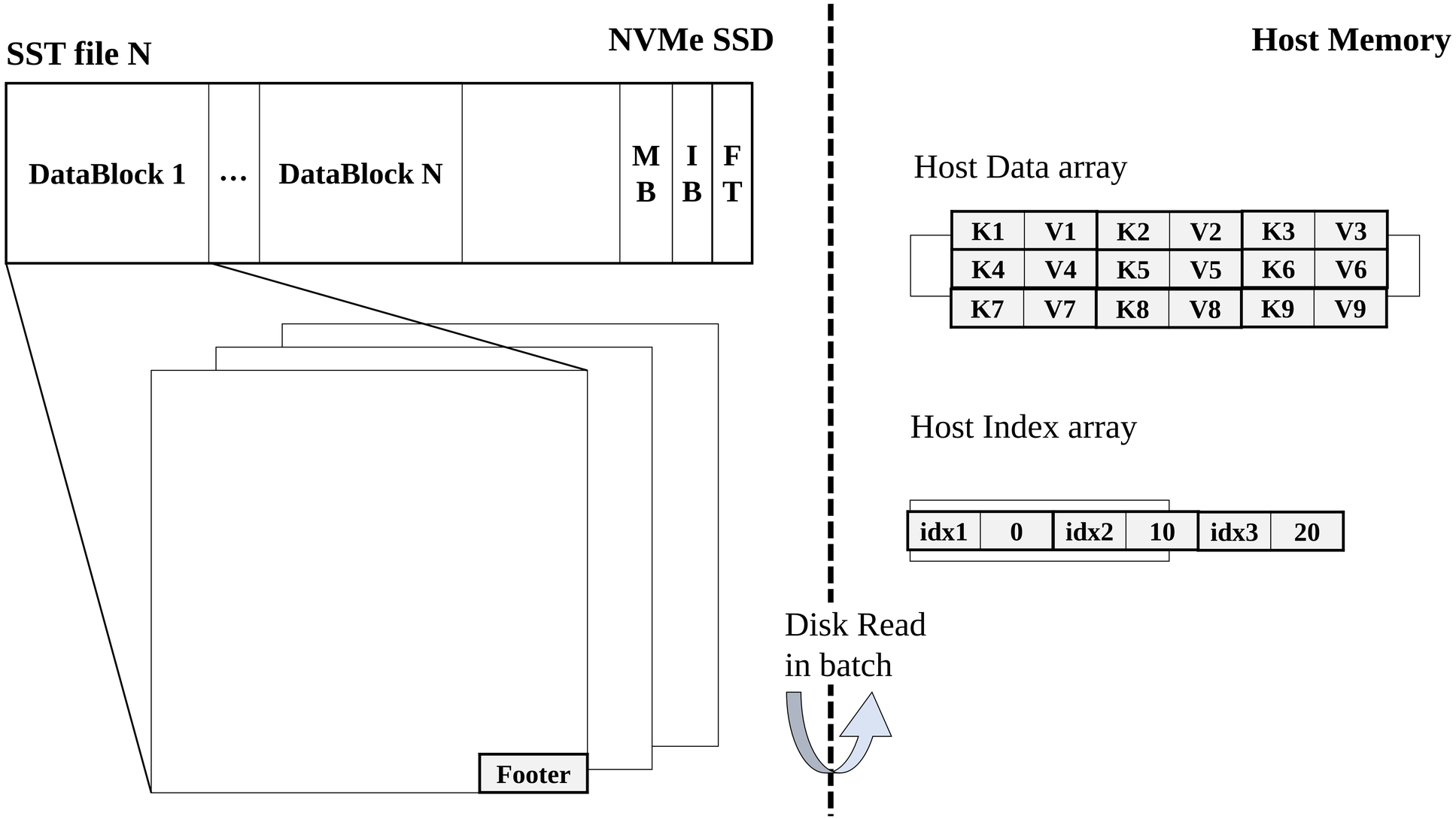
General version





GPU Acceleration (4)

General version

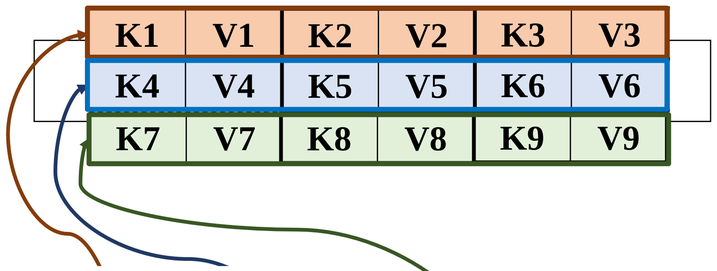


GPU Acceleration (5)

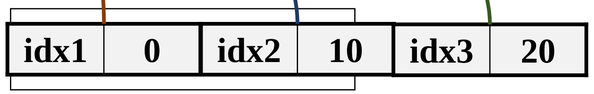
General version

Host Memory

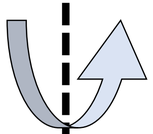
Host Data array



Host Index array

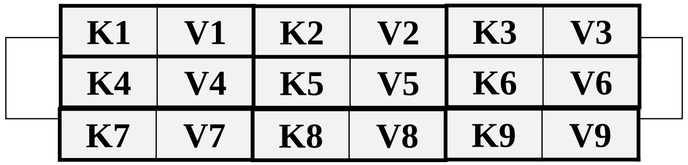


Memory Copy H2D

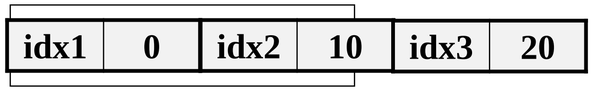


GPU
Device memory

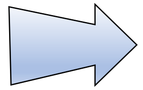
Device Data array



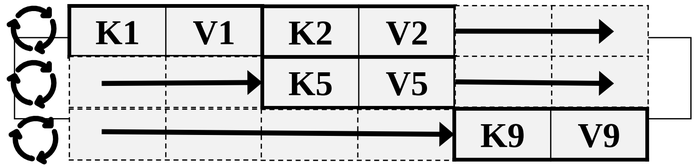
Device Index array



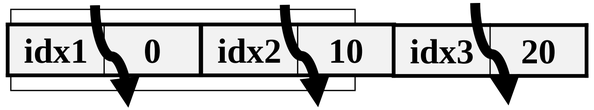
Kernel Execution



Device Result Data array



Device Index array



Thread 1 **Thread 2** **Thread 3**

GPU Acceleration (6)

General version

GPU
Device memory

Query Engine

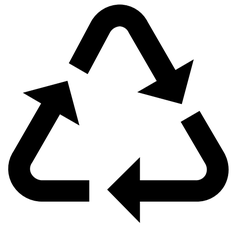
Device Result Data array

K1	V1	K2	V2	K5	V5	K9	V9

Temporary buffer

K1	V1	K2	V2	K5	V5	K9	V9

Row by Row

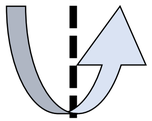


Join

Query Execution

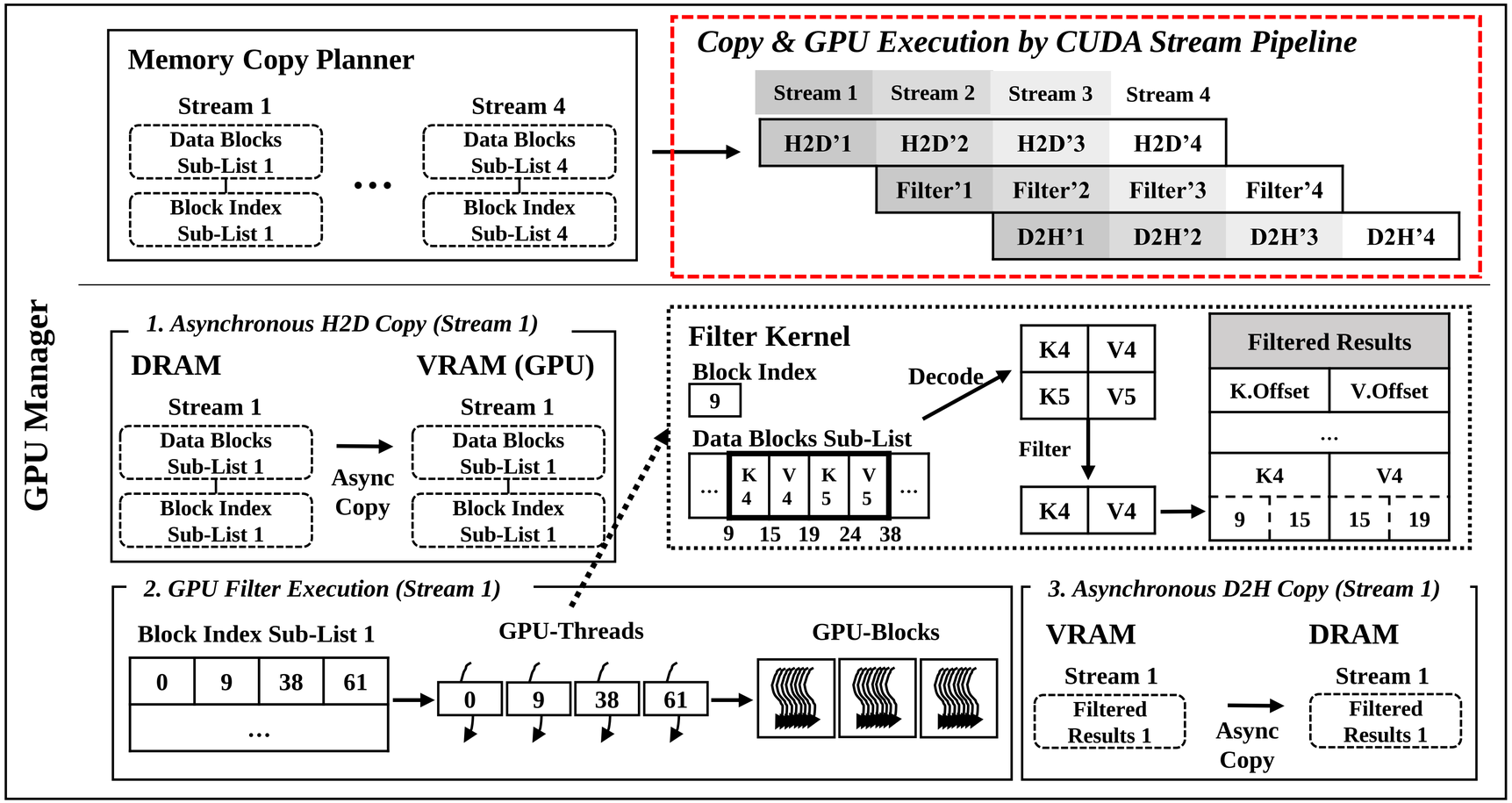
Aggregation

Memory Copy D2H



GPU Acceleration (7)

Optimized version of GPU acceleration



■ CUDA Stream

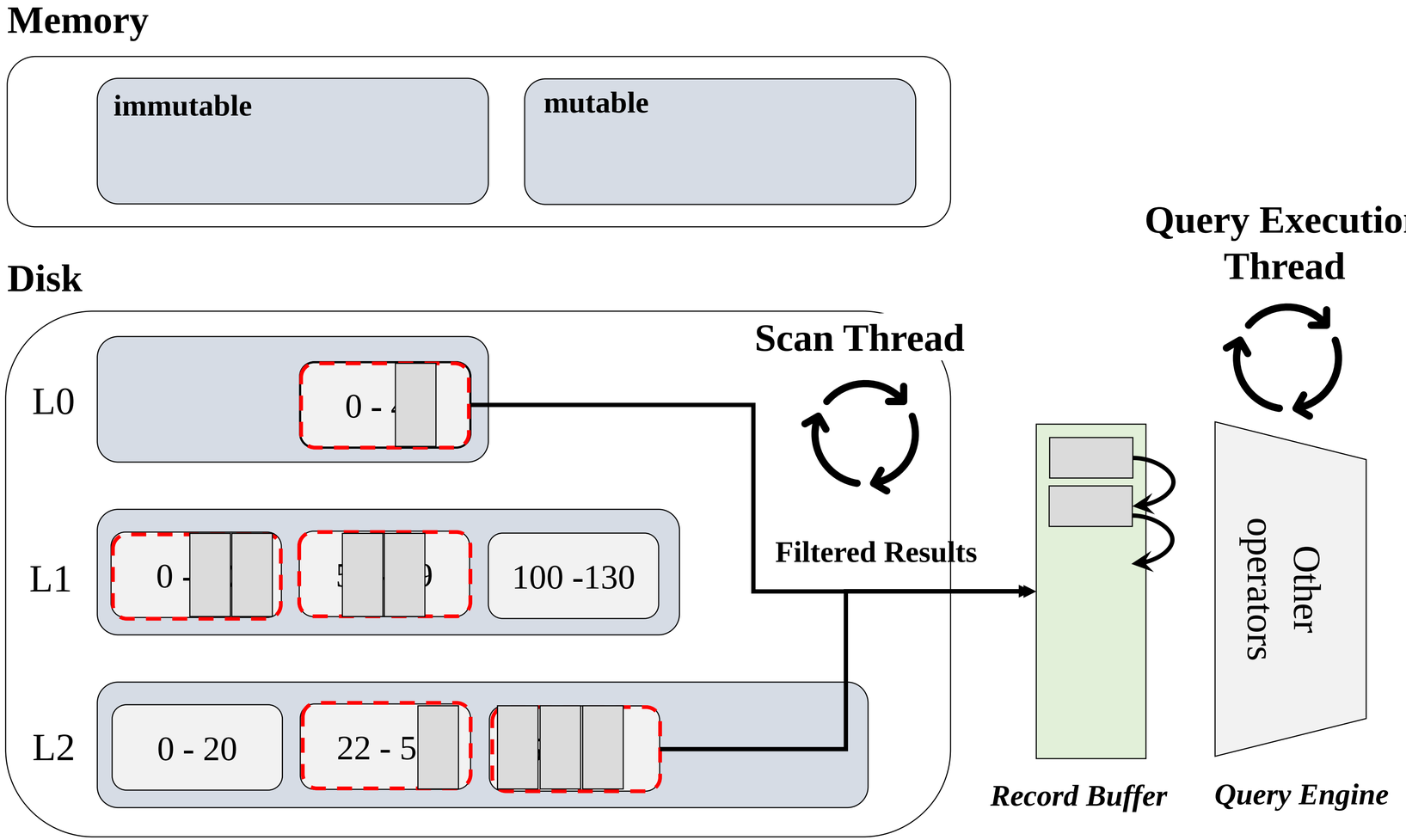
- Data transfer cost is heavy
- Overlap data transfer and kernel execution

■ Flow

1. Split the kv-pairs and indices in unit of stream
2. Transfer streams to GPU engine asynchronously
3. Interleave data transfer and kernel execution
4. Results are also returned asynchronously

GPU Acceleration (8)

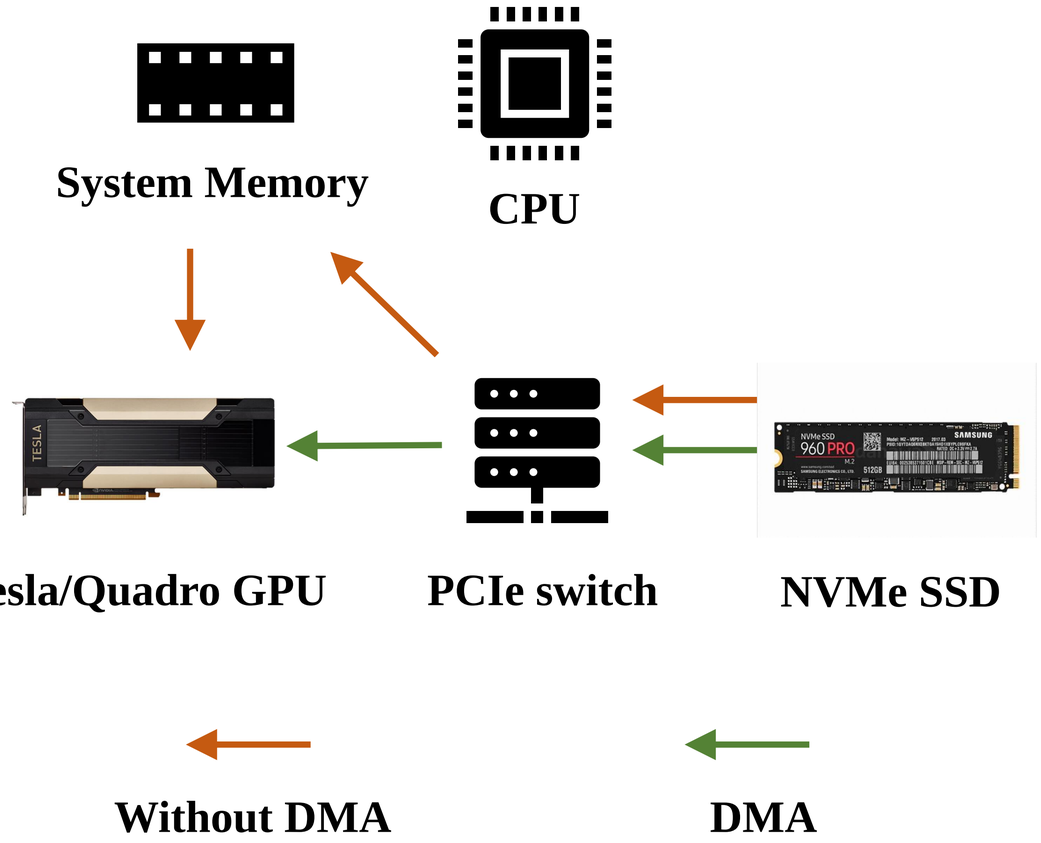
Scan Pipelining



- **Scan pipelining**
  - Delay from table scan affects overall query execution
  - Overlap the query execution and GPU accelerated scan

GPU Acceleration (9)

Direct Memory Access

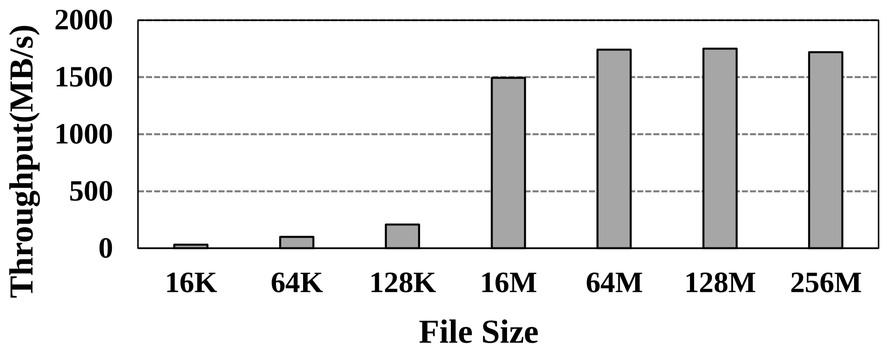


■ Peer-to-Peer, Direct Memory Access

- Modern GPU characteristic (Tesla, Quadro)
- Project Donard, NVMMU, SPIN, GPUDirect Storage

■ Adaptable with LSM-tree

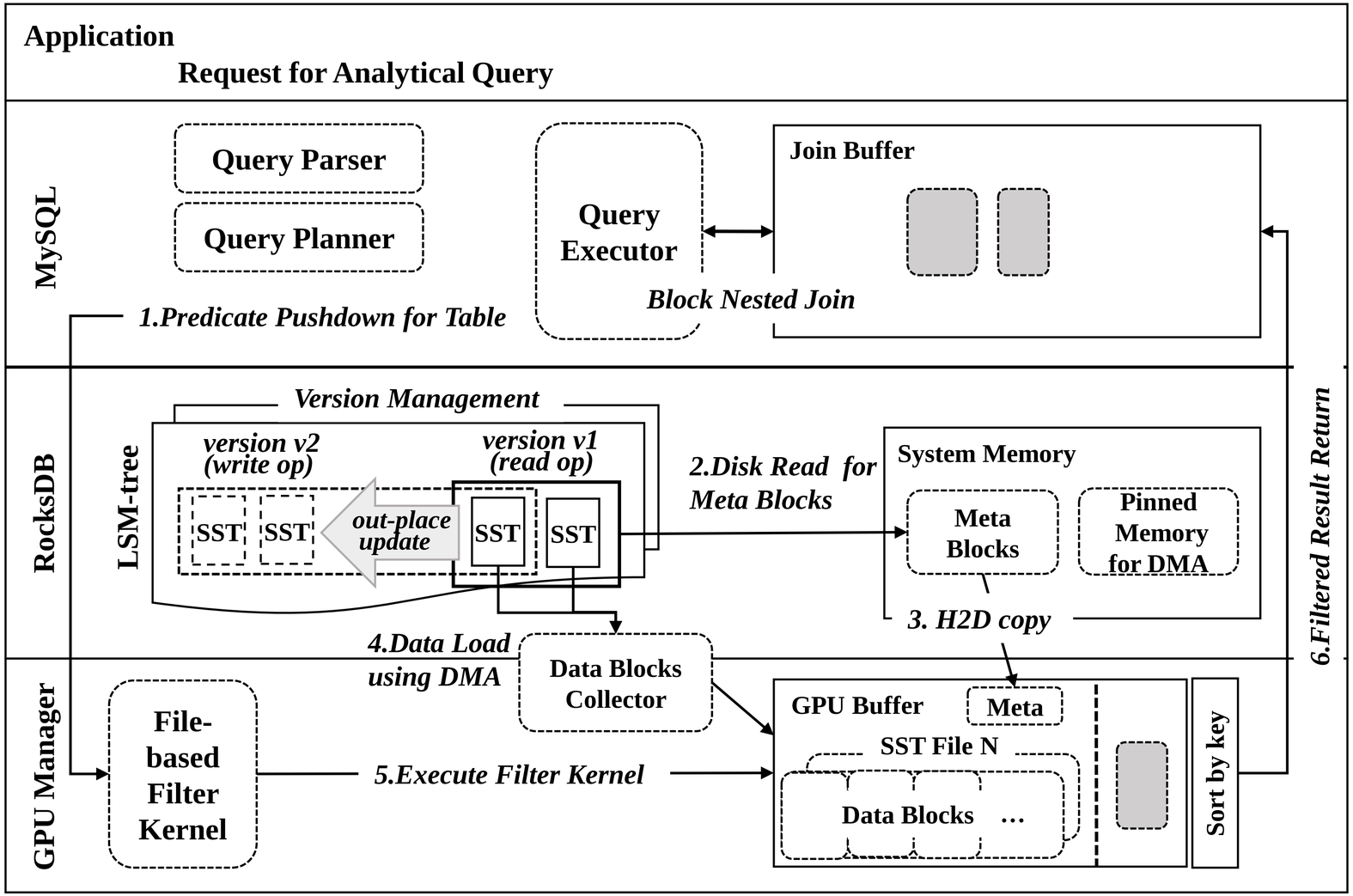
- SST file size is a few megabytes



NVMe-GPU DMA throughput

GPU Acceleration (10)

Direct Memory Access



■ with Direct Memory Access

1. Disk Read for Meta Blocks related to SST files (Trivial burden)
2. Synchronous transfer Meta Data
3. Data load using DMA
4. Filtering kernel directly accesses SST files mapped in GPU memory

■ File Version Management

- Meta data **can be corrupted** because of mixed inserts-scans
- Version Management is necessary
  - ✓ Insert generates **new version of files**
  - ✓ Invalid files are garbage collected after scan operation is completed

Evaluation

■ Hardware/Software Configuration

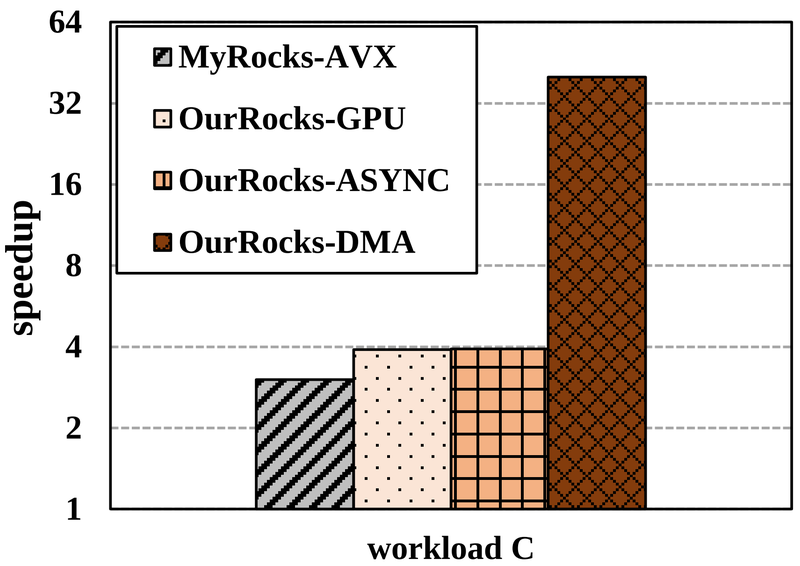
SPECIFICATION	
CPU	Xeon Processor (8-core) 4110
RAM	32GB DDR4 PC4 2666
Storage	1TB Samsung NVMe SSD 960 Pro
GPU	NVDIA Tesla V100 NVDIA Tesla K80
OS	Ubuntu 14.04
File System	ext4
GPU Driver	384.183 version
CUDA	9.0 version

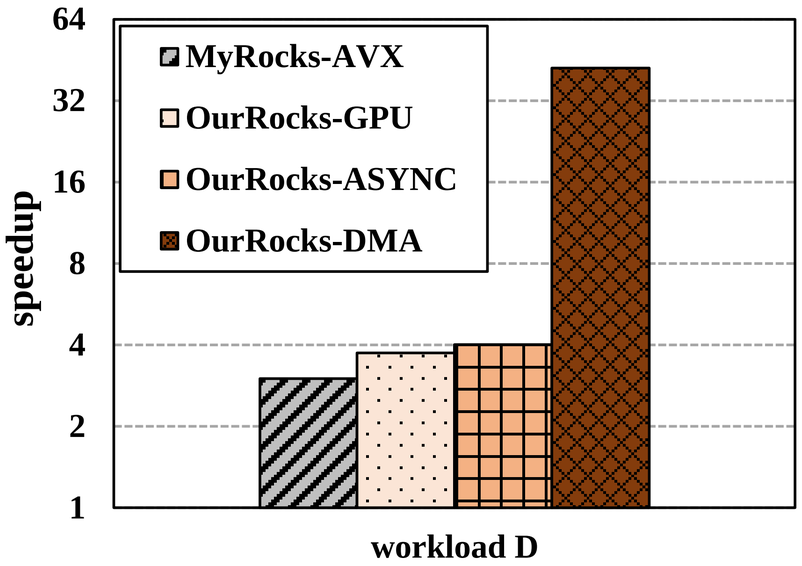
■ Five types database

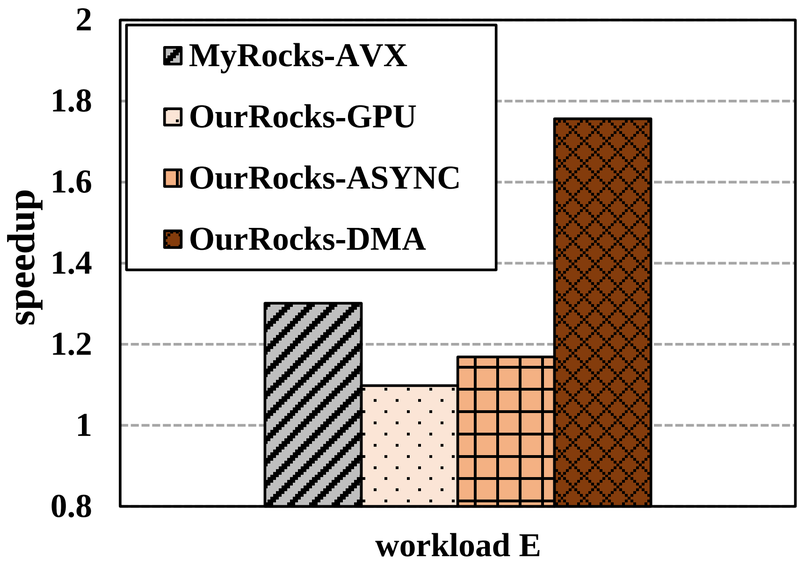
- MyRocks (baseline)
- MyRocks-AVX
- OurRocks-GPU
  – General version + CUDA stream
- OurRocks-GPU async
  – General version + CUDA stream + Scan pipelining
- OurRocks-DMA
  – General version + Direct Memory Access+ Scan pipelining

YCSB

Yahoo Cloud Service Benchmark







■ Most widely used standard key-value store benchmark

- Workload C : read-only, Workload D : point queries and insert queries, Workload E : range queries and insert queries
- six field table (one 64bit integer, five 32 bytes random strings)
- 50 million row entries

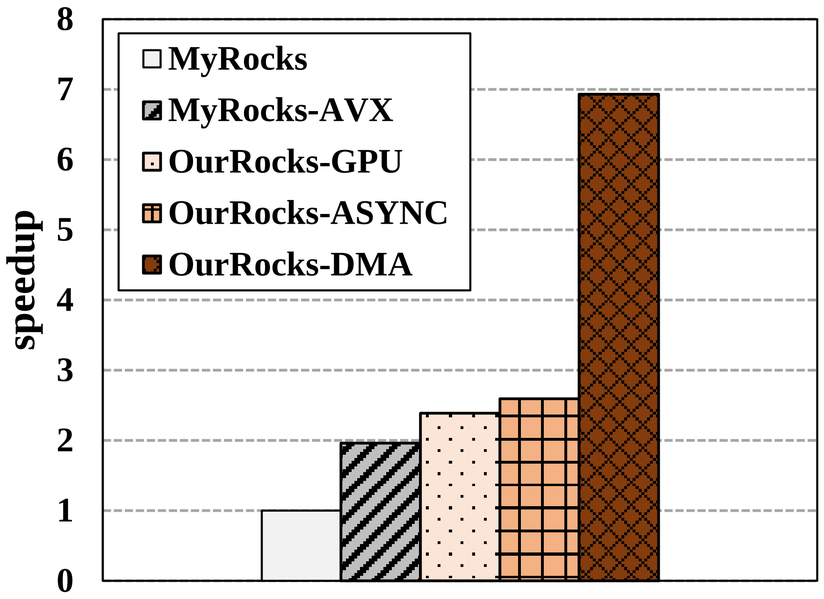
■ OurRocks-DMA significantly reduces the execution time

- **40 x** improvement in Workload C, D and **1.7 x** improvement in Workload E

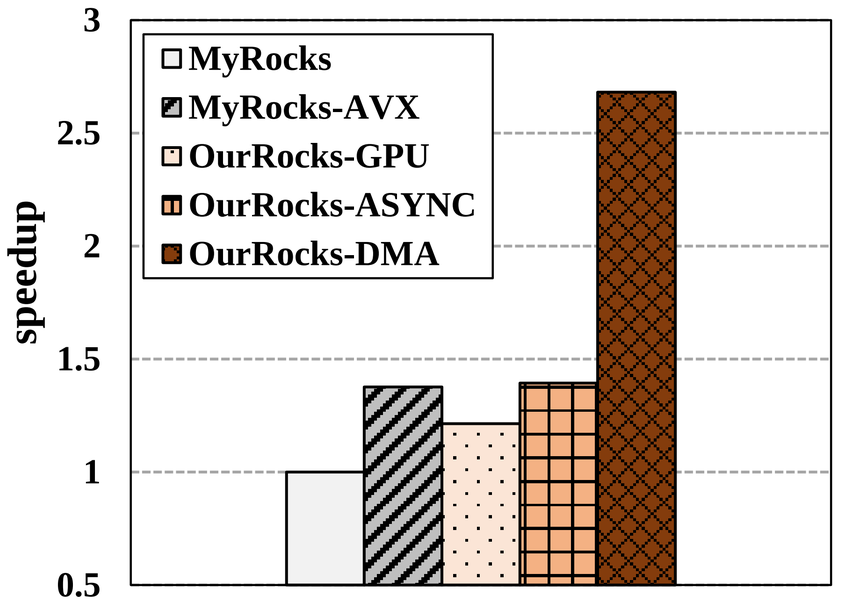
TPC-H (1)

Pre-defined simple query

SELECT count(p_partkey) **FROM** part
WHERE p_size < 15;



SELECT count(p_partkey) **FROM** part
WHERE p_type="SMALL PLATED BRASS";

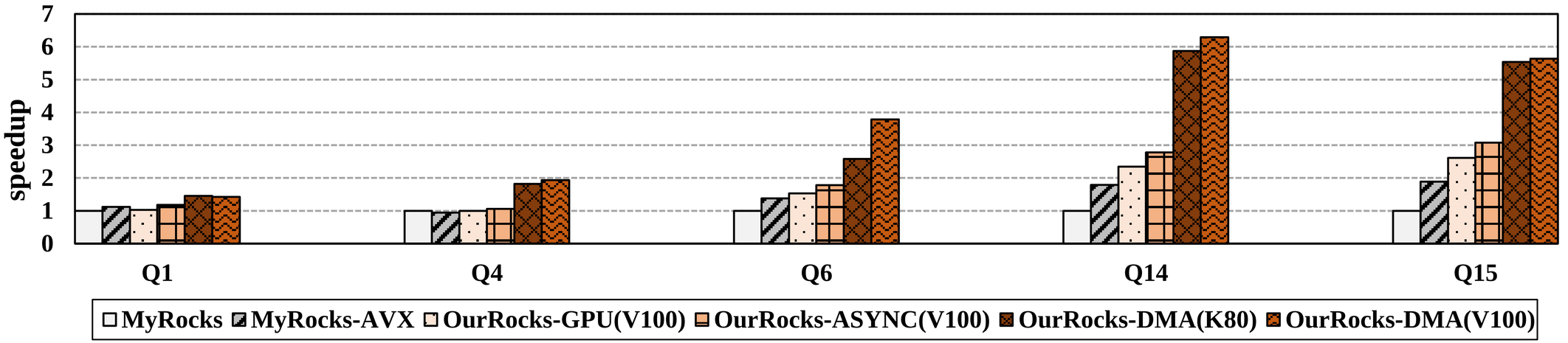


■ **Execution of simple queries with TPC-H dataset**

- 2 GB 'part' table with nine columns
- filter on an integer type column and a variable-sized column
- OurRocks-DMA has up to **6.9x** (integer type column), **2.6x** (variable-sized column) performance improvement

TPC-H (2)

Complex query



■ **Execution of complex queries with TPC-H dataset**

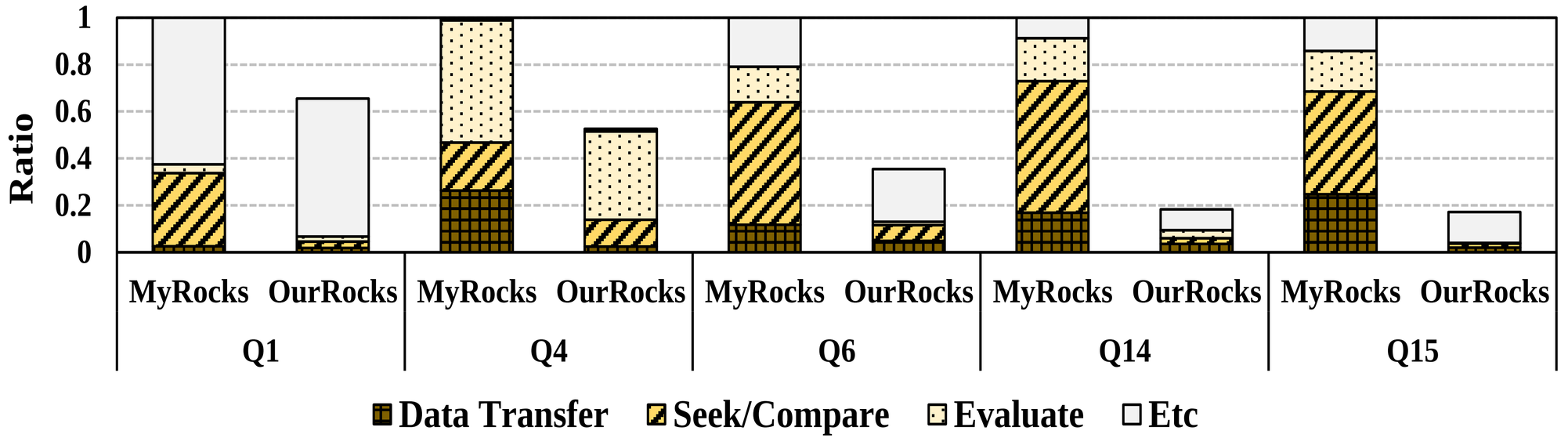
- select representative queries that induce the full scan operation on a table sufficiently large
- Q1, Q4 – OurRocks-DMA improves the execution **by 1.4 to 1.9**
- Q6, Q14, Q15 – OurRocks-DMA improves the execution **by 3.7 to 6.2**

Q6	**select** sum(l_extendedprice * l_discount) as revenue **from** lineitem **where** l_shipdate >= date '1994-01-01' and l_ship date < date '1994-01-01' + interval '1' year and l_discount between 0.06 - 0.01 and 0.06 + 0.01 and l_quantity < 24;

Example of queries in TPC-H

TPC-H (3)

Time breakdown analysis in TPC-H



■ Best Cases

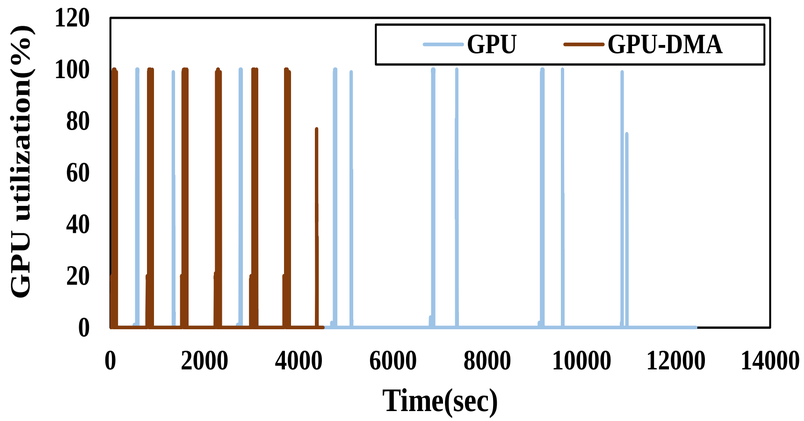
- OurRocks exhibits similar trends on Q6, Q14, Q15
- Weight of operations for selected table (GPU acceleration) is relatively high
- Iteration cost is **significantly reduced**

■ Trivial Cases

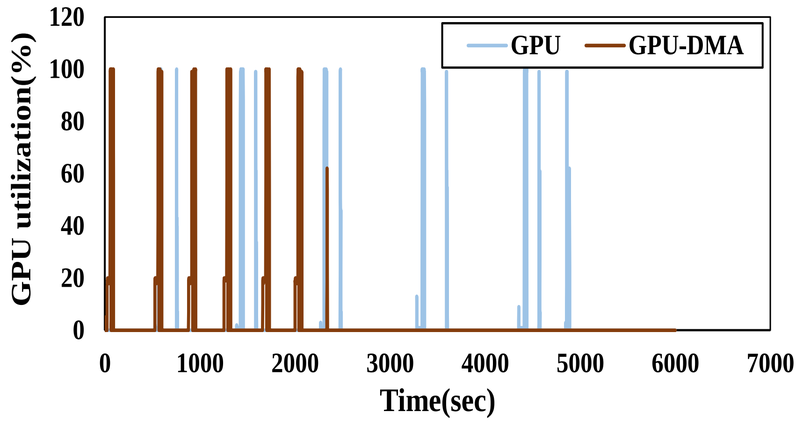
- Q1, Q4 – The ratio for operations independent of table scan is **dominantly high**

Resource Utilization (1)

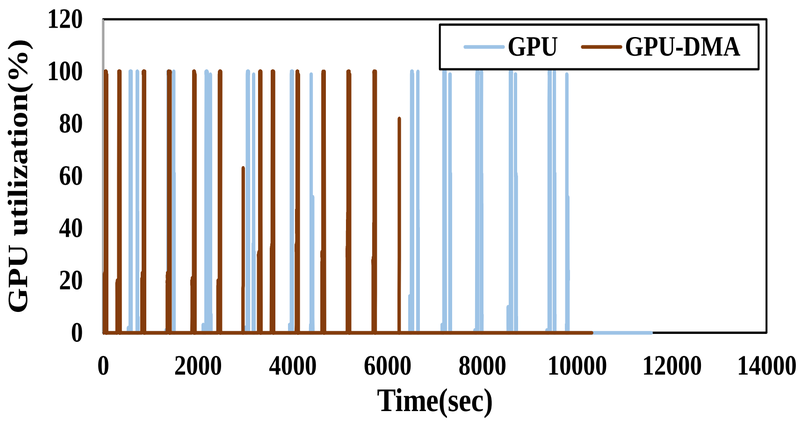
Graphics Processing Unit



(a) Query 6



(b) Query 14



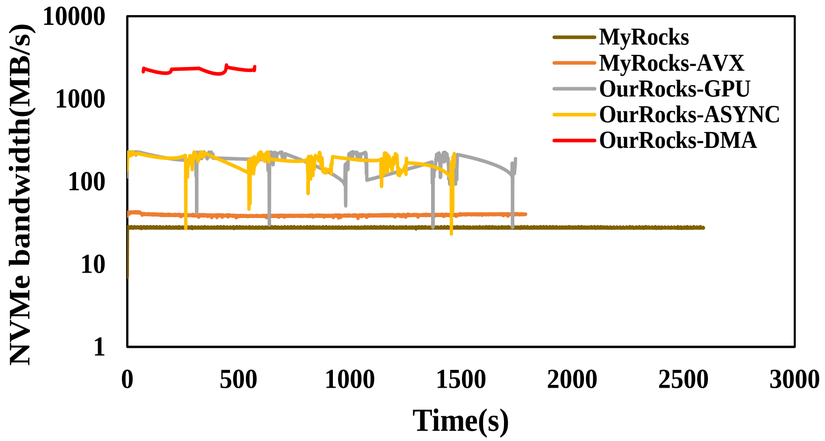
(c) Query 15

■ GPU utilization

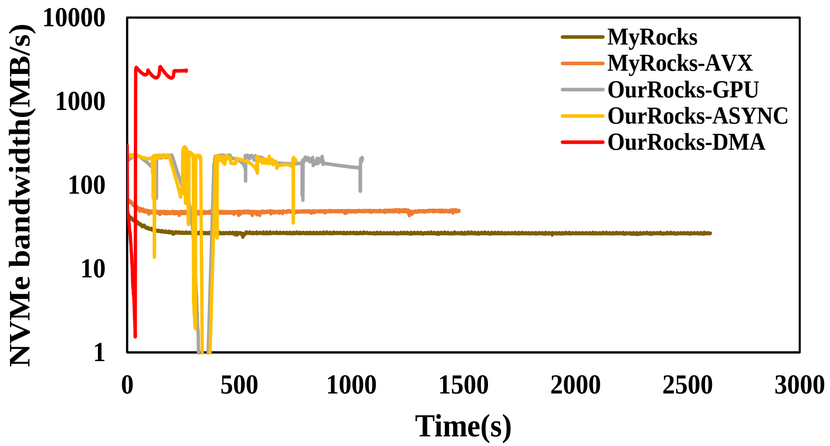
- measuring the percentage of time over the past second during which the kernel function is executed on the GPU
- The total time of kernel function processing occupies the small portion
- **The preparation time for GPU processing** is a dominant factor
- The delay from the preparation **prevents the GPU from being continuously utilized**
- OurRocks-DMA eliminates this bottleneck such that the database fully utilizes the high throughput of GPU per unit time.

Resource Utilization (2)

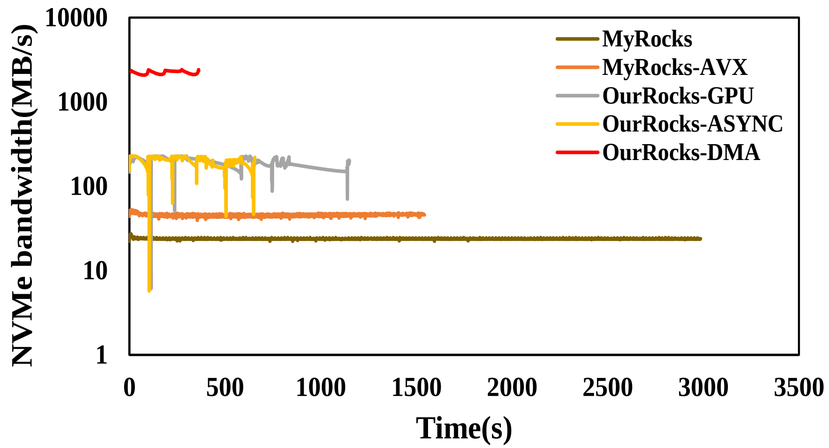
NVMe SSD bandwidth



(a) Query 6



(b) Query 14



(c) Query 15

■ CPU-driven LSM-tree database (MyRocks, MyRocks-AVX)

- Constantly request I/O to NVMe SSD
- However, lower utilization compared to the maximum bandwidth

■ GPU-driven LSM-tree database

- OurRocks-GPU, OurRocks-ASYNC show a higher level of bandwidth due to reading data files in batch
- OurRocks-DMA **fully utilizes the NVMe SSD bandwidth**

Other Factors (1)

Performance-per-Price

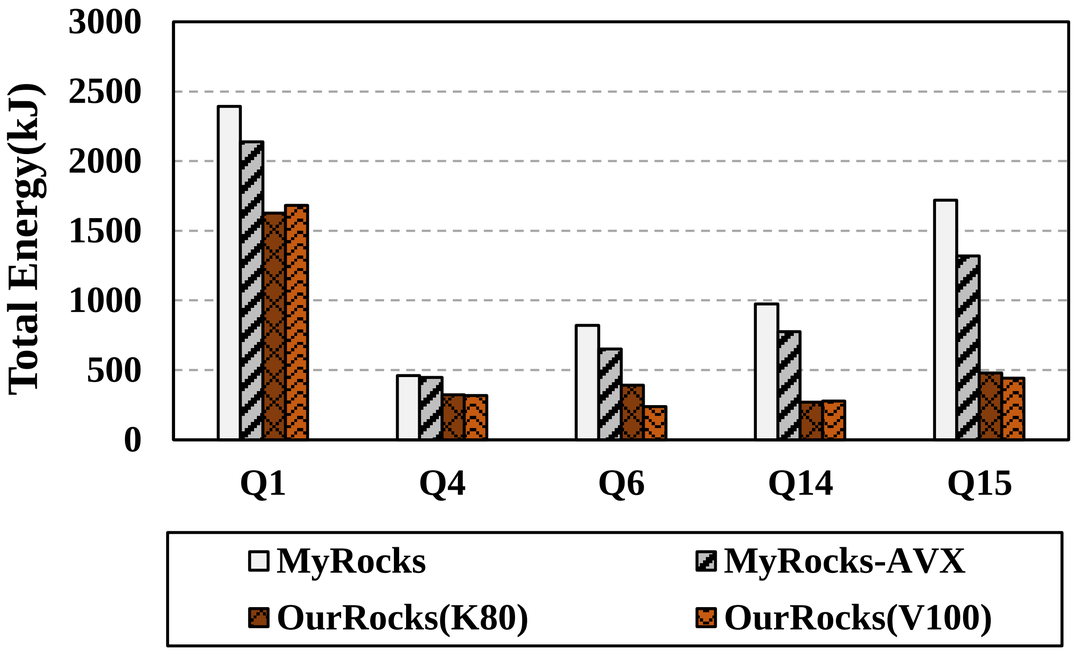
	COST (US$)	PERFORMANCE PER COST (TOTAL AVERAGE)	PERFORMANCE PER COST (TOP 3)
CPU	500 $	1x baseline	1x baseline
CPU(AVX)	500 $	1.36x baseline	1.72x baseline
GPU(Tesla K80)	540 $	1.14x baseline	2.49x baseline

■ Performance-per-Price metric

- Average performance (TPC-H) / server deployment cost
- **Cost :** NVIDIA Tesla K80 ≒ Xeon Processor 4110

Other Factors (2)

Power efficiency



SJPM-C16

■ Energy consumption

- External power measurement device, which is capable of measuring power up to 3520 Watts
- OurRocks consumes less energy by a factor between 1.4 and 3.8 compared to MyRocks
- OurRocks **would be advantageous** in energy saving

Comparison with Previous Studies (1)

■ Research for LSM-tree

- Wu et al. (LSM-trie), Pan et al. (dCompaction), Yao et al. (LWC-tree), Zhang et al. (pipeline compaction)
- Mainly target for improving the performance of write and space
- OurRocks primarily **targets for improving the scan performance** in correlation with the relational database and LSM-tree

■ Database techniques for GPU

- Zhang et al. (Mega-KV), Ashkiani et al. (GPU LSM)
- Paul et al. (GPU hash join), Wang et al. (concurrent GPU query processor), Paul et al. (GPL)
- typically focused on **maximizing the utilization of GPU resources** during the processing of **data loaded onto the main memory** by taking advantage of **the characteristics of modern GPU.**
- OurRocks focuses on optimizing the procedure of the table scan in LSM-tree which stores **most data on the disk**, by leveraging the **characteristics of modern GPU**, thereby maximizing the **utilization of the resources of NVMe device.**

Comparison with Previous Studies (2)

■ Big data platform with GPU

- Yuan et al. (Spark-GPU), Chen et al. (Gflink), Asai et al (GPU enabled Spark)
- Focus on effective data communication schemes between Java virtual memory and GPU memory

■ Storage device with near data processing

- SmartSSDs, Intelligent SSDs (e.g. YourSQL)
- FPGA (e.g. IBM Ibex)
- OurRocks utilizes GPU

Conclusion

■ First academic project that

- Leverages the LSM-tree engine and supports hybrid transactional/analytical processing
- Overcomes the limitation of core operation of LSM-tree (Limited use of high bandwidth storage)

■ Simple approach, but Can be applied in Big Data analytics application

- Real-time recommendation service, fraud detection service with Mobile, Internet of Thing data
- LSM-tree based big data platform, Redis on Flash

■ Noticeable performance improvement in various aspects

- Yahoo! Cloud Serving Benchmark
- TPC-H benchmark
- Performance-per-Price
- Energy Efficiency