USENIX ATC '24

July 10-12, 2024
SANTA CLARA, CA, USA

Scalable Billion-point Approximate Nearest Neighbor Search Using SmartSSDs

Bing Tian, Haikun Liu, Zhuohui Duan, Xiaofei Liao, Hai Jin

Huazhong University of Science and Technology

華中科技大學
HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Outline

- Background and Motivation
- SmartANNS Design
- Results
- Conclusion

Background of ANNS

❖ Top-K Nearest Neighbor Search (NNS)

Query → $y_o, y_1, \dots, y_{N-1}$ → **find K closest vector from the database** → $y_1, y_5, \dots, y_m$

Top-K Results

query vector: $[q_o, q_1, \dots, q_{D-1}] \in \mathbb{R}^D$

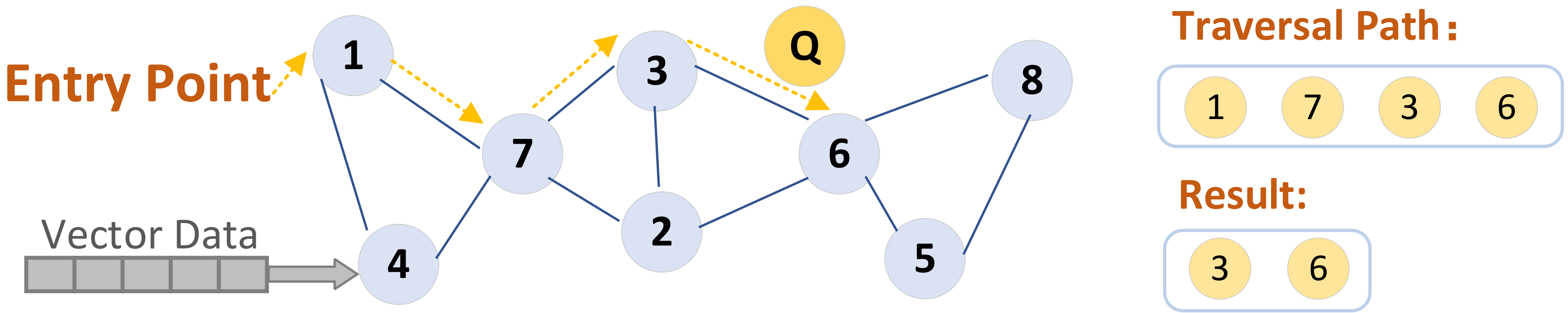
y_n **in database :** $[x_o, x_1, \dots, x_{D-1}] \in \mathbb{R}^D$

distance metric: $\underset{n \epsilon \{0,1,\dots,N-1\}}{\operatorname{argmin}} \|q - [X]_n\|_2^2$

Background of ANNS

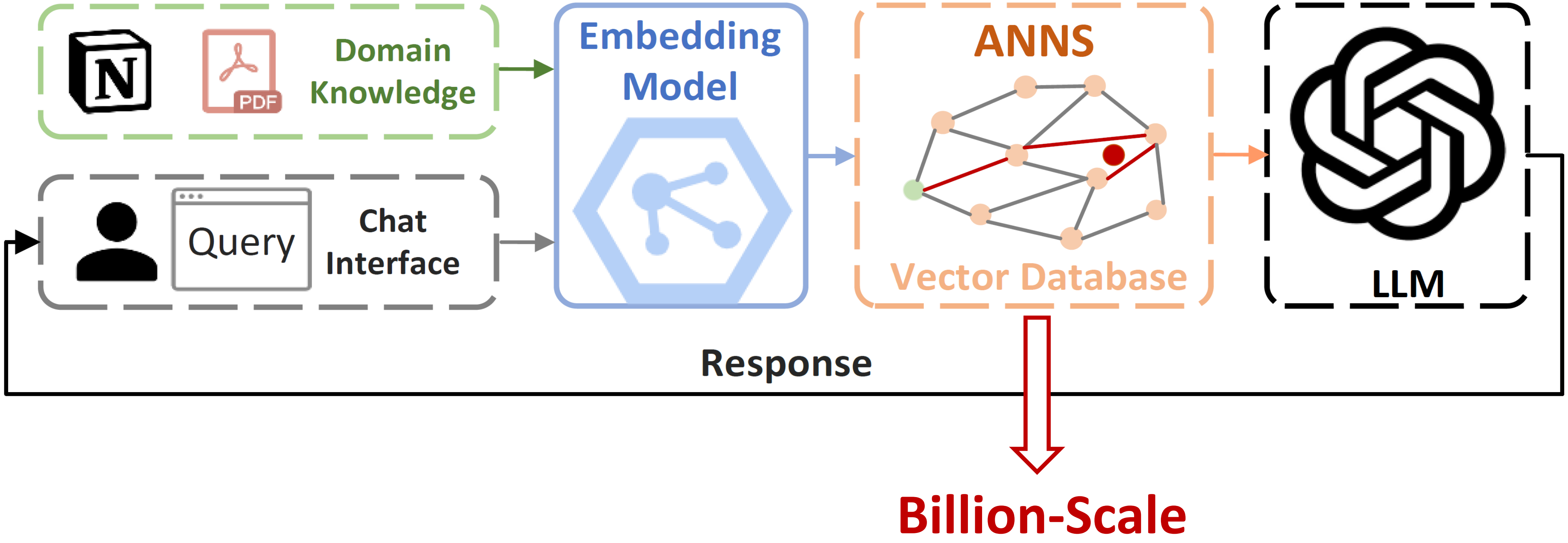
❖ Approximate Nearest Neighbor Search (ANNS)

Brute Force Search	Hash based Search	Tree based Search	Graph based Search ☺

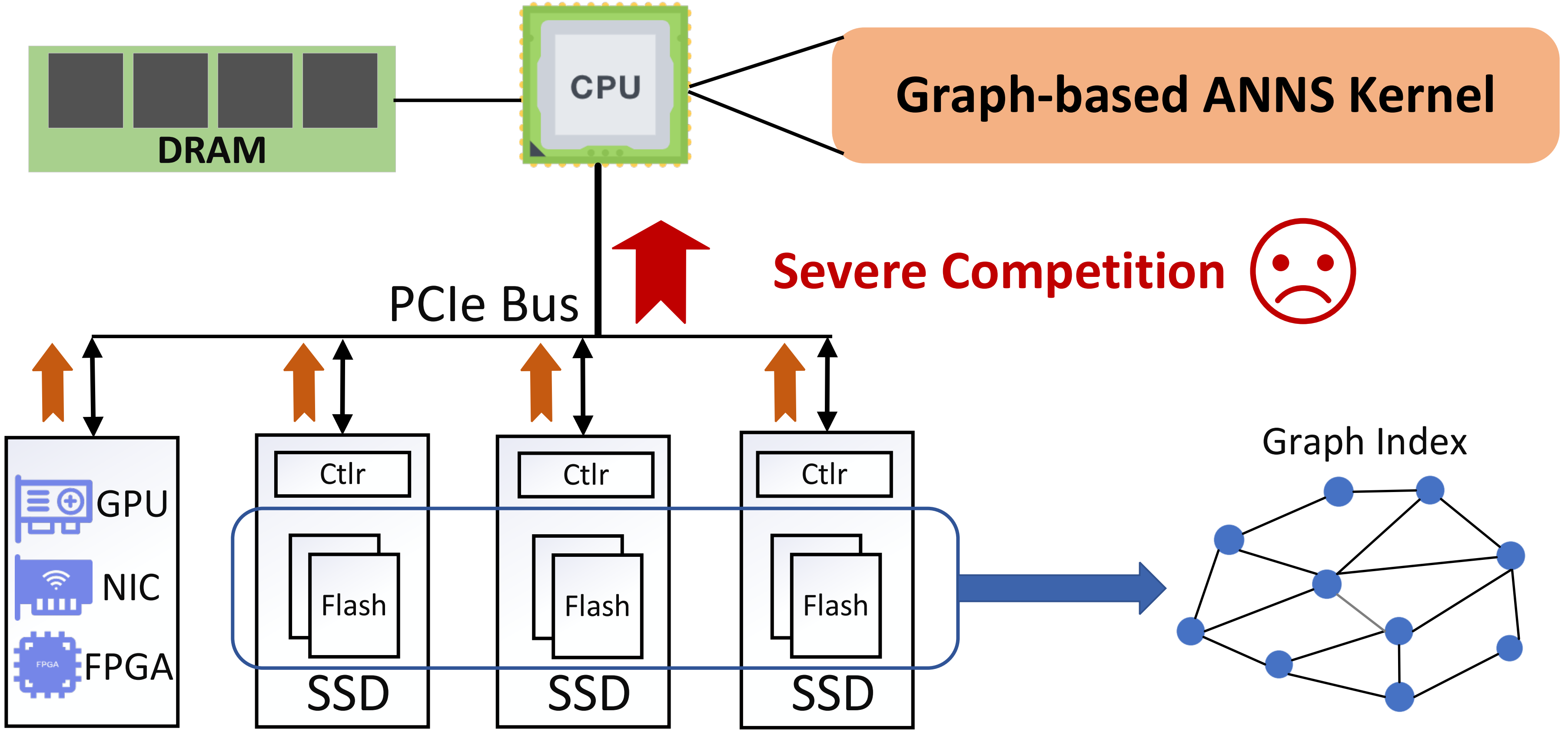


Background of ANNS

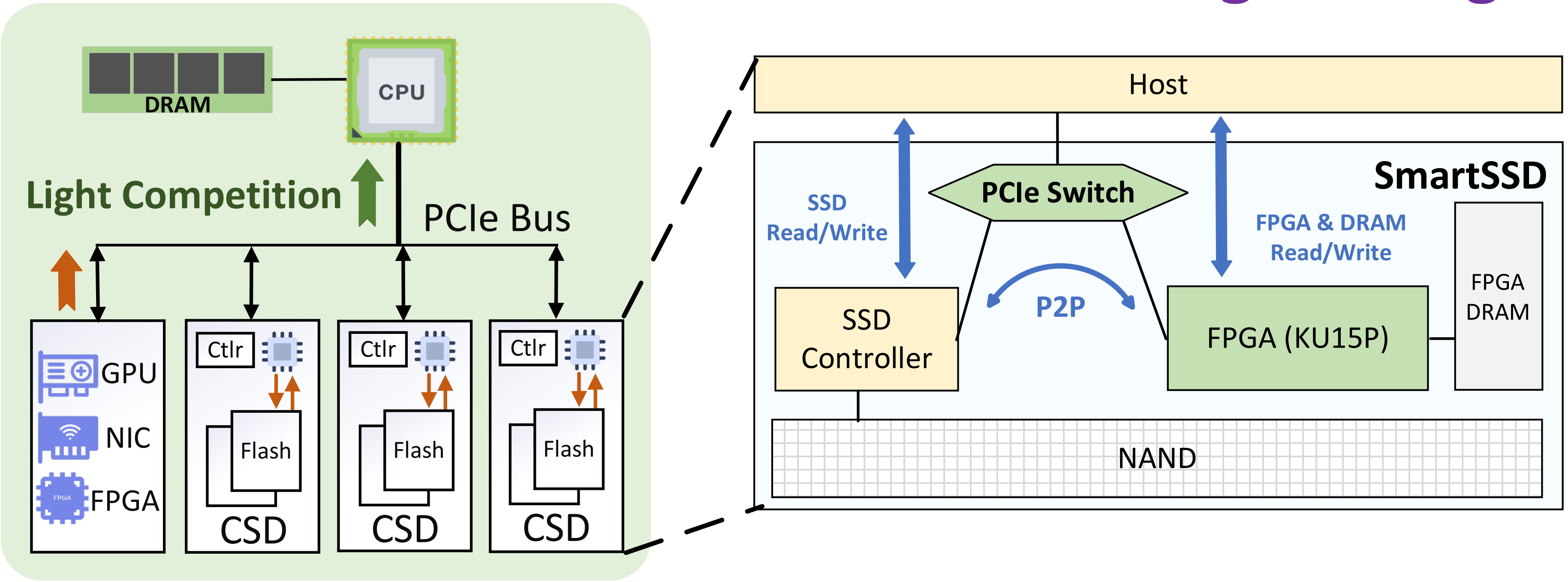
❖ Retrieval-Augmented Generation



Traditional Computing Architecture for ANNS



CSD-empowered NDP Architecture



Large Storage Capacity

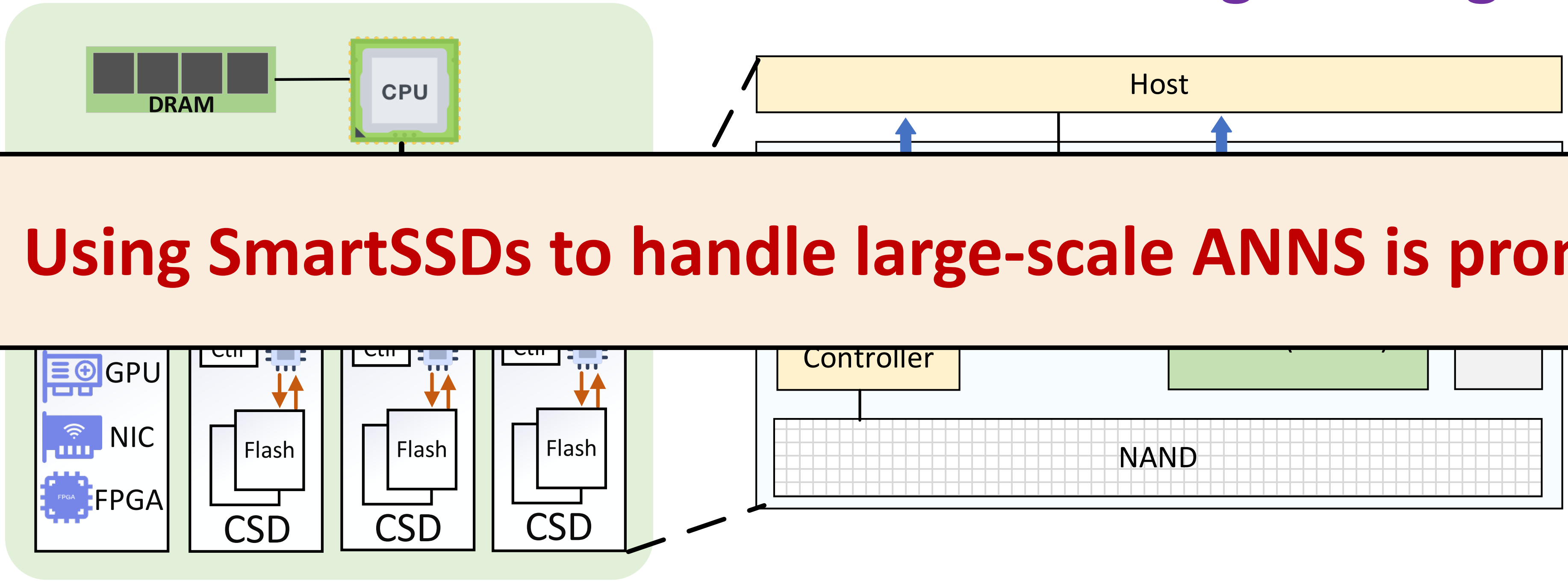
Less Data Movement

Performance scalability

Mitigate Host Resource Contention

CSD-empowered NDP Architecture

Large Storage Capacity



Using SmartSSDs to handle large-scale ANNS is promising …

Movement

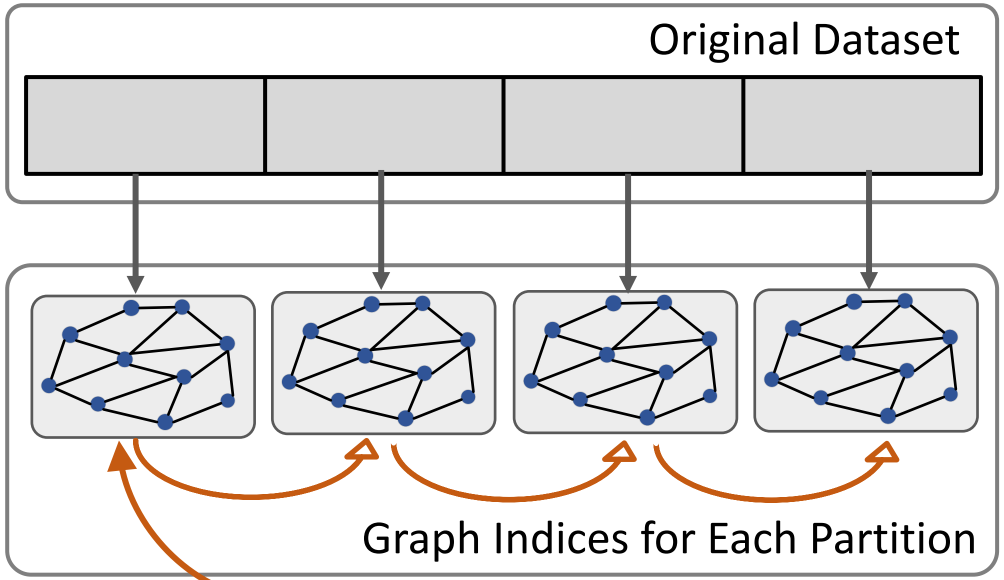
Mitigate Host Resource Contention

Performance scalability

CSD-based ANNS Solution

- Offline Index Construction
  - Split dataset
  - Construct graph for each partition
- Online Search
  - Traverse all the graph indices
  - Merge all intermediate results
  - Return top-k result

CSDANN [TC'22], SmartSSD-based ANNS



Significant Computation + **Limited Resource**

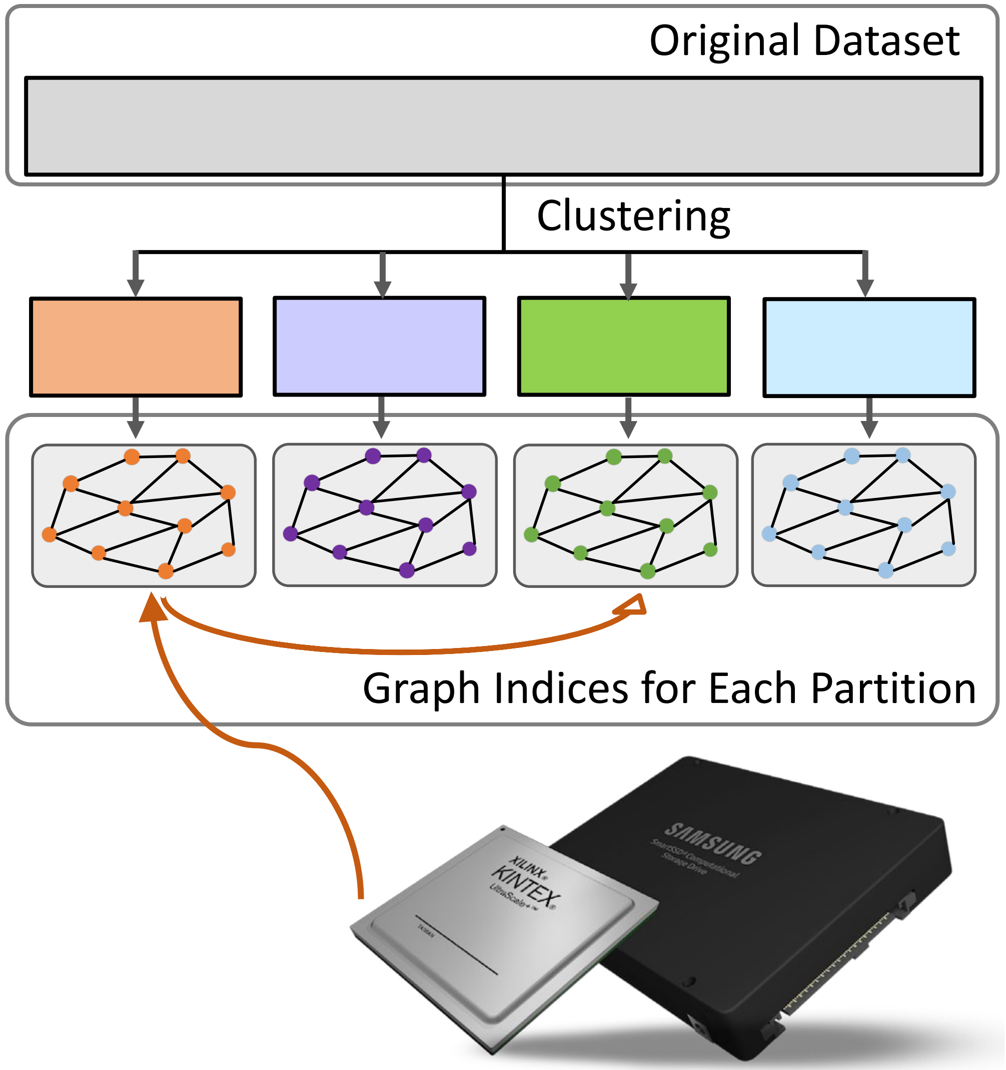
⇩

Sub-optimal Performance

Opportunities of Hierarchical Indexing

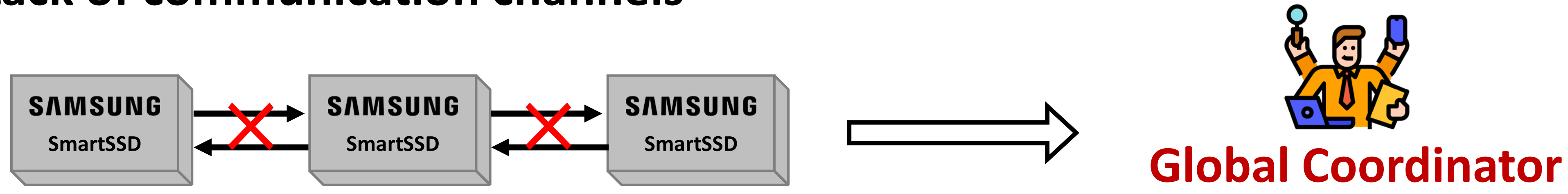
- Offline Index Construction
  - Partition dataset using clustering
  - Construct graph for each shard
- Online Search
  - Prune irrelevant shards
  - Traverse closest graph of shards
  - Merge and return top-k result

Less computing overhead

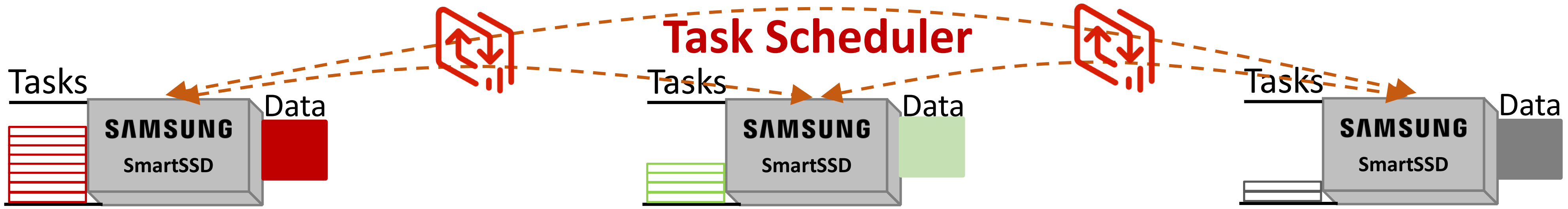


Challenges

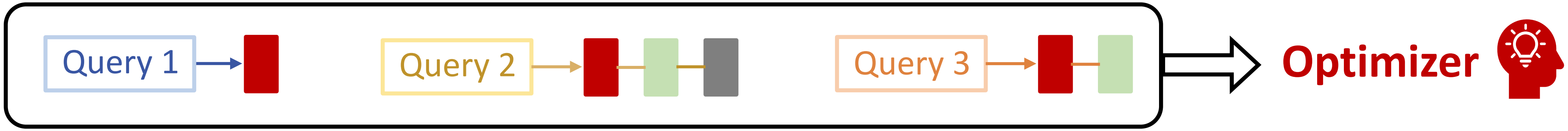
1. Lack of communication channels



2. Load imbalance across SmartSSDs



3. Differences between queries



Outline

- Background and Motivation
- SmartANNS Design
- Results
- Conclusion

Key Designs

Lack of communication channels

Hierarchical indices in host and SmartSSDs

Load imbalance across SmartSSDs

Task scheduling based on the optimized data layout

Differences between queries

Learning-based shard pruning algorithm

Hierarchical indices

Construction Step

- ❖ Hierarchical Balanced Clustering
- ❖ Construct HNSW graph indices

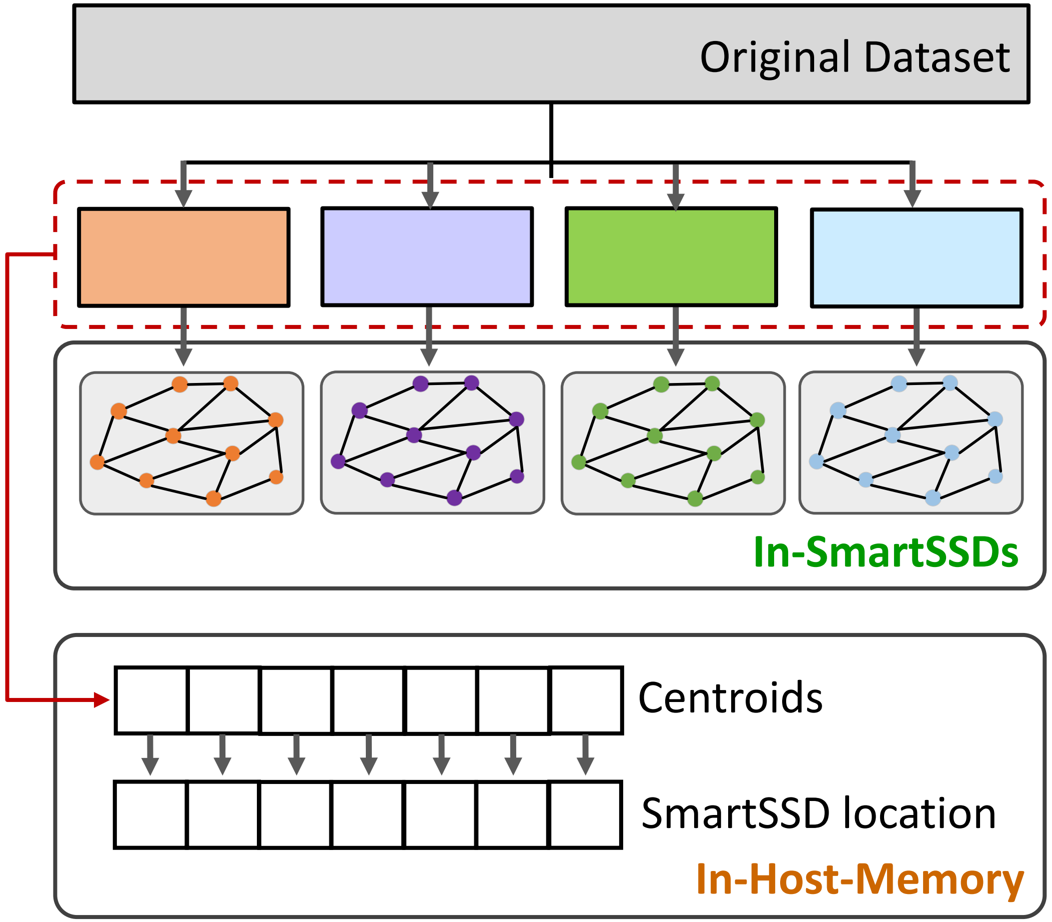
For SmartSSDs

- ❖ Store graph indices in SmartSSDs

For Host CPU

- ❖ Extract centroids for each shard
- ❖ Build shard-SmartSSD mapping table

"Host CPU + SmartSSDs" cooperation



Shard Pruning

Gradient Boosting Decision Trees (GBDT)

- ❖ A strong predictive model combing multiple decision trees

Training Stage

- ❖ Iteratively predicting the mean, computing residuals, and fitting weak decision trees to these negative gradients

Inference Stage

- ❖ Assimilating weighted contributions of all individual weak models

Training Setup

Training size	One million
Learning rate	0.05
Iteration	500

Input Features

- ❖ The query vector
- ❖ Distance between the query and the top-k nearest shards (D_k) / distance between the query and the top-1 nearest shard (D_1)
- ❖ The total number of all shards

Lightweight and high performance

Task Scheduling

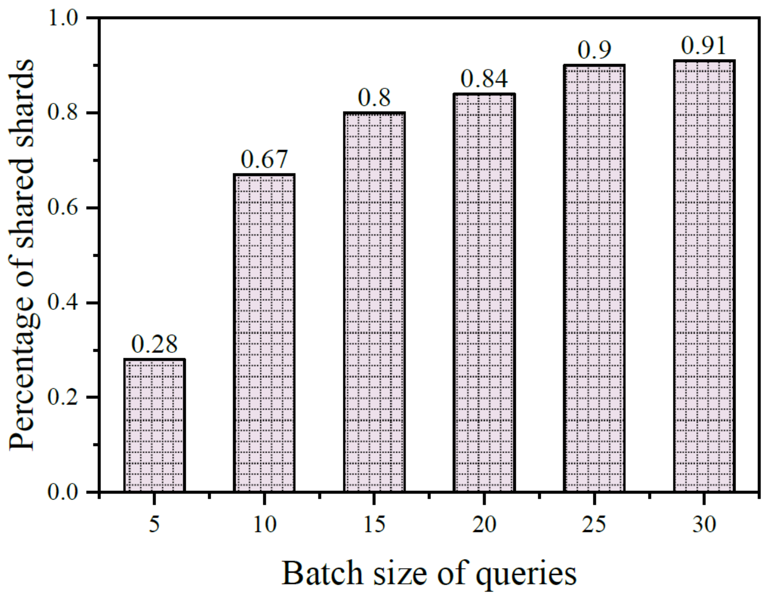
Data Access Pattern of Hierarchical Indices

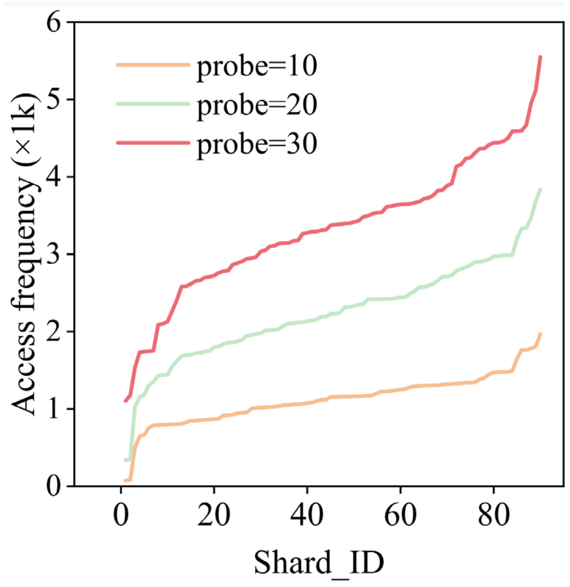
Observation 1

❖ A large portion of shards are accessed by different queries over a period of time, implying a good data locality

Observation 2

❖ The access distribution of different shards are highly skewed





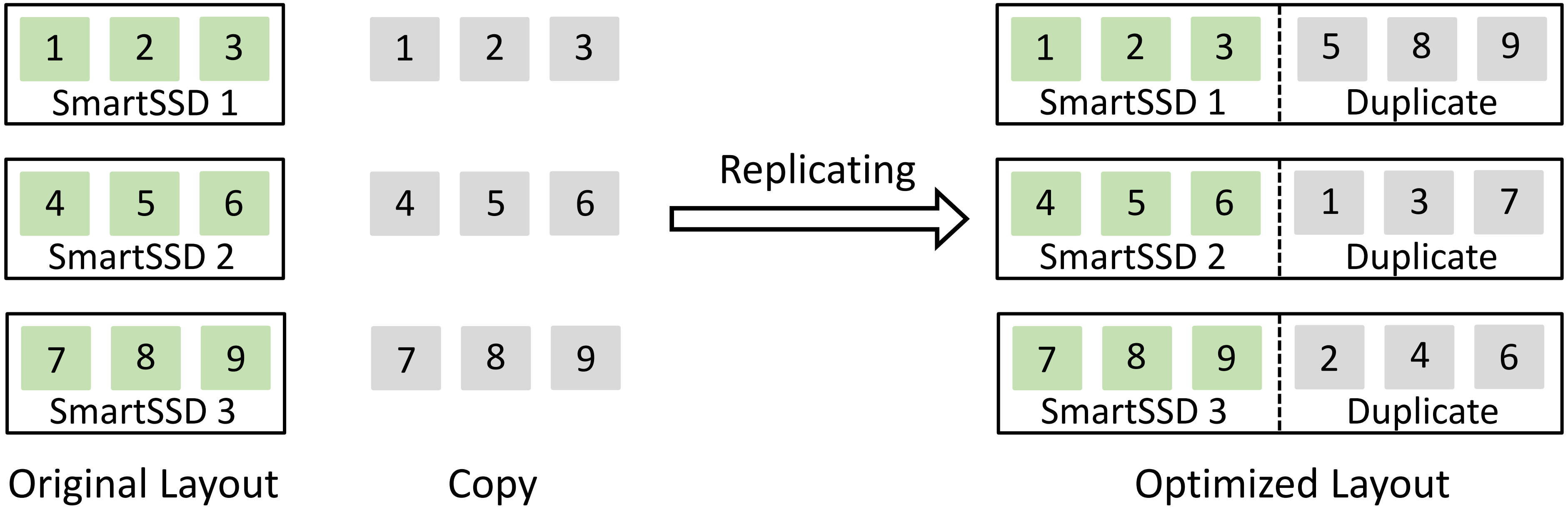
1. Exploiting the data locality among queries
2. Placing hot shards on different SmartSSDs

Task Scheduling

Optimized Data Layout

Offering more flexibility for task scheduling

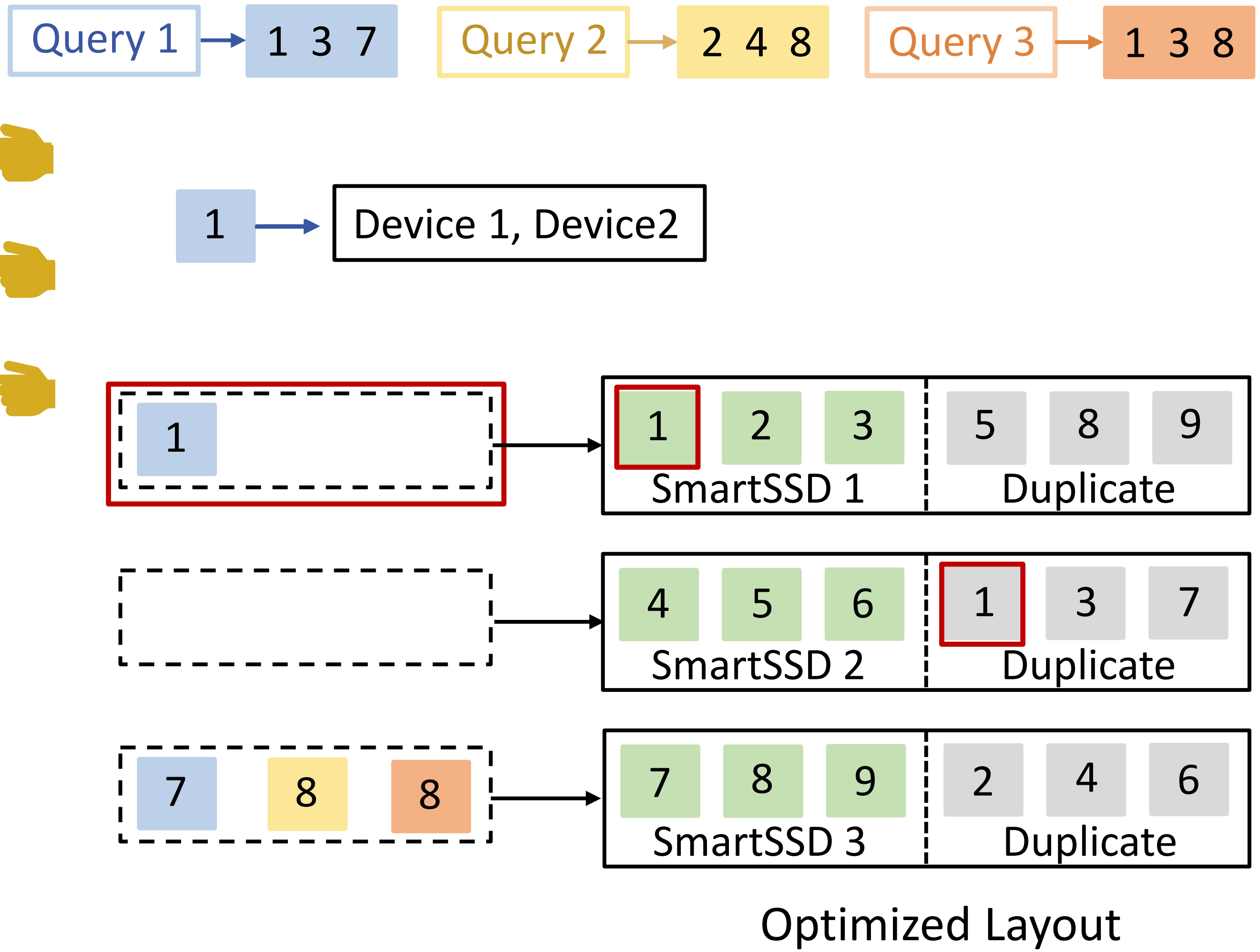
- Iteratively placing shard with highest hotness to the SmartSSD with lowest cumulative hotness
- Replicating shards from one SmartSSD to another SmartSSD once



Task Scheduling

Scheduling Steps

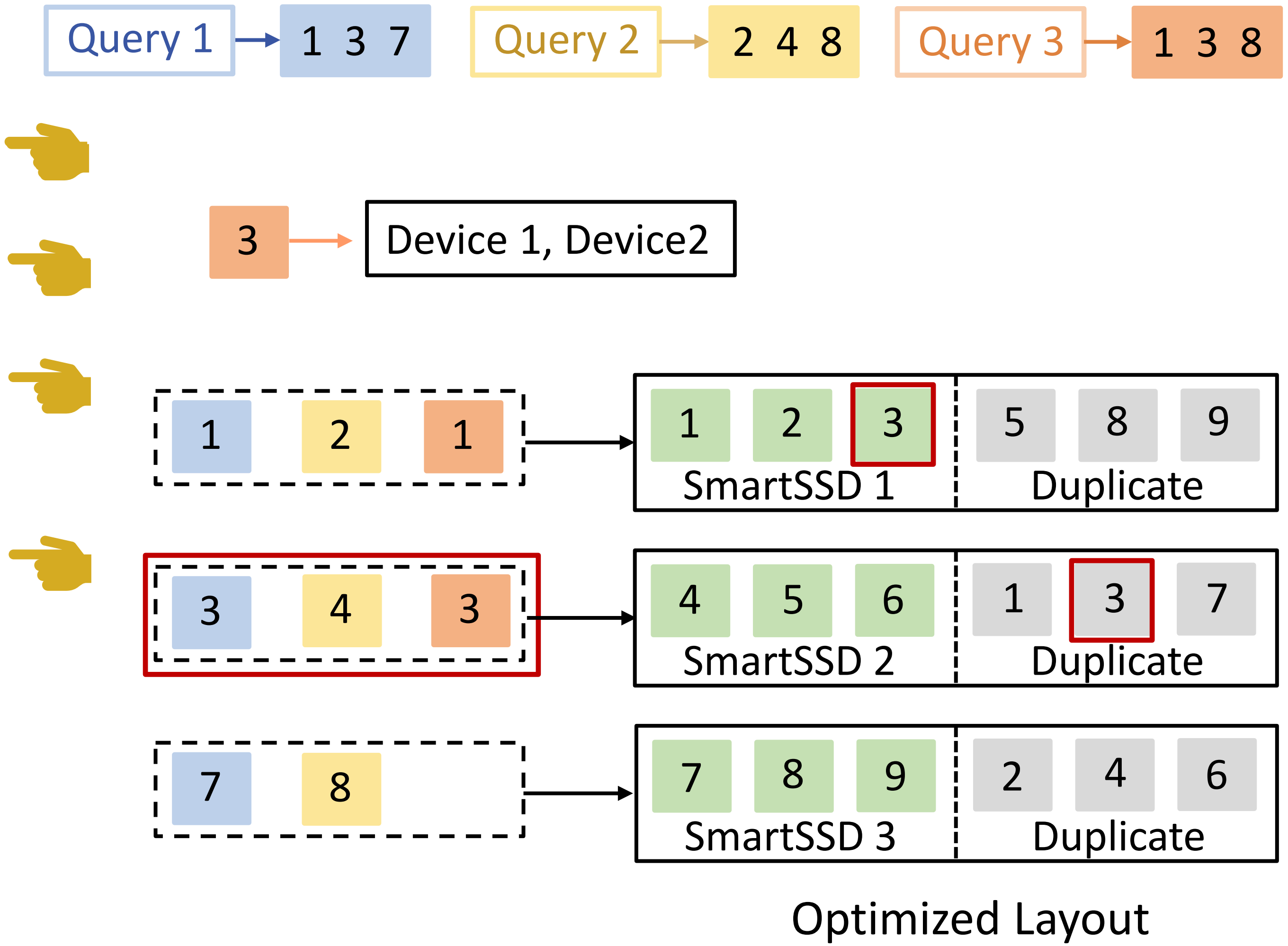
```
for i = 0; i < Tasks.size(); i++ do
    /* Find devices that store task-relevant shards.   */
    Base[] ← FindBase(Devices, Tasks[i]);
    /* Check whether there is data reuse on device for
       this task.                                        */
    Assign[] ← Check(Device_to_Tasks, Tasks[i]);
    if (Assign.size = 0) ∨ (Assign.size = Base.size)
     then
        Target ← MinWork(DeviceLoad, Base);
        Device_to_Tasks[Target].insert(Tasks[i]);
        DeviceLoad.update(Target);
    else
        TimeInfo[] ← NULL;
        for j = 0; j < Base.size(); j++ do
            Temp[] ← Device_to_Tasks[Base[j]];
            Temp.insert(Tasks[i]);
            TimeInfo[j] ← Estimate(Temp);
        end
        Target ← Min(TimeInfo);
        Device_to_Tasks[Target].insert(tasks[i]);
        DeviceLoad.update(Target);
    end
end
```



Task Scheduling

Scheduling Steps

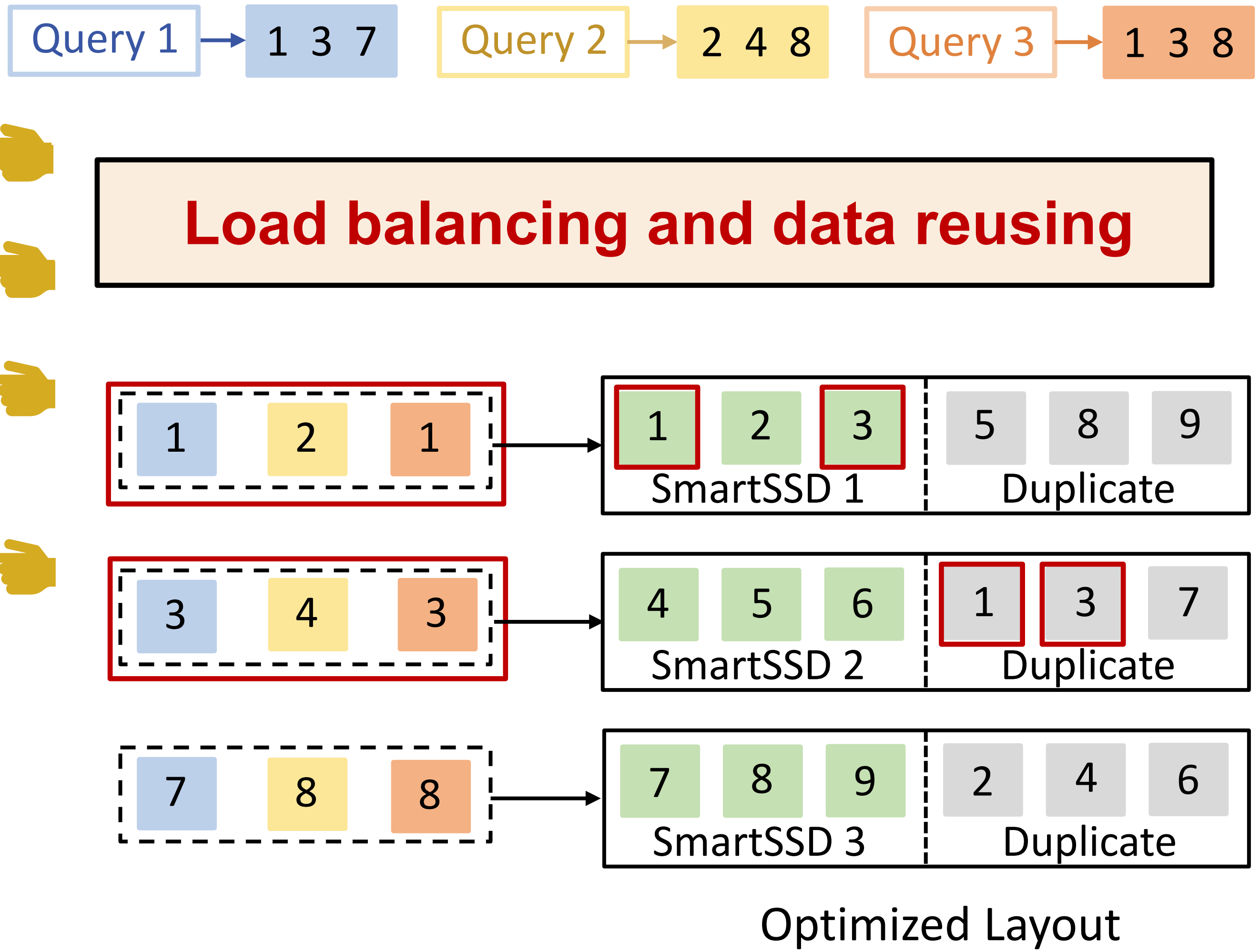
```
for i = 0; i < Tasks.size(); i++ do
    /* Find devices that store task-relevant shards.   */
    Base[ ] ← FindBase(Devices, Tasks[i]);
    /* Check whether there is data reuse on device for
       this task.                                     */
    Assign[ ] ← Check(Device_to_Tasks, Tasks[i]);
    if (Assign.size = 0) ∨ (Assign.size = Base.size)
     then
        Target ← MinWork(DeviceLoad, Base);
        Device_to_Tasks[Target].insert(Tasks[i]);
        DeviceLoad.update(Target);
    else
        TimeInfo[ ] ← NULL;
        for j = 0; j < Base.size(); j++ do
            Temp[ ] ← Device_to_Tasks[Base[j]];
            Temp.insert(Tasks[i]);
            TimeInfo[j] ← Estimate(Temp);
        end
        Target ← Min(TimeInfo);
        Device_to_Tasks[Target].insert(tasks[i]);
        DeviceLoad.update(Target);
    end
end
```



Task Scheduling

Scheduling Steps

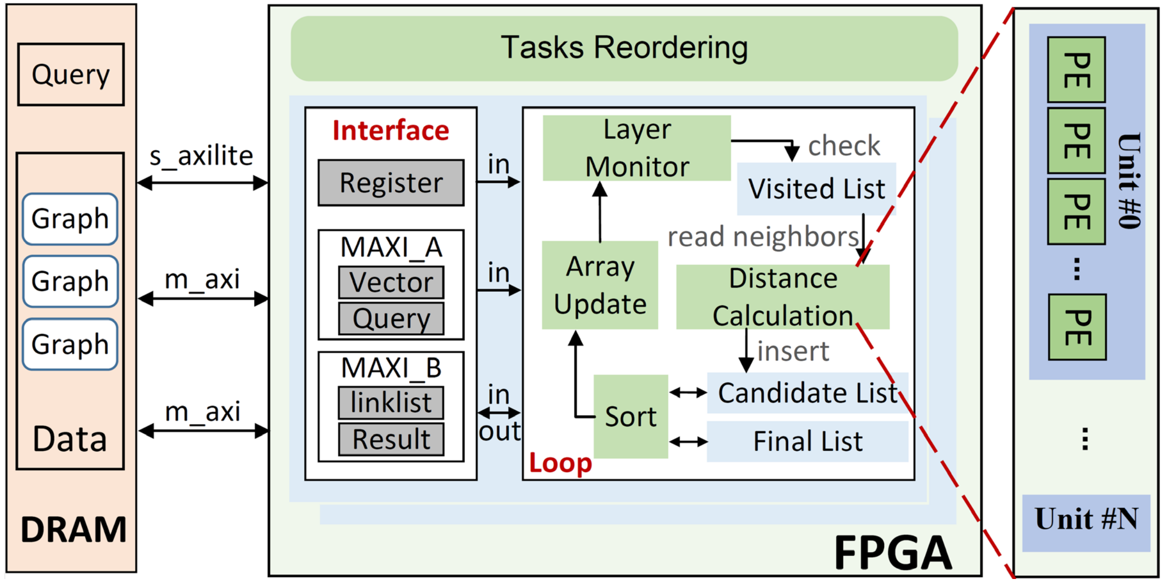
```
for i = 0; i < Tasks.size(); i++ do
    /* Find devices that store task-relevant shards.   */
    Base[ ] ← FindBase(Devices, Tasks[i]);
    /* Check whether there is data reuse on device for
       this task.                                      */
    Assign[ ] ← Check(Device_to_Tasks, Tasks[i]);
    if (Assign.size = 0) ∨ (Assign.size = Base.size)
     then
        Target ← MinWork(DeviceLoad, Base);
        Device_to_Tasks[Target].insert(Tasks[i]);
        DeviceLoad.update(Target);
    else
        TimeInfo[ ] ← NULL;
        for j = 0; j < Base.size(); j++ do
            Temp[ ] ← Device_to_Tasks[Base[j]];
            Temp.insert(Tasks[i]);
            TimeInfo[j] ← Estimate(Temp);
        end
        Target ← Min(TimeInfo);
        Device_to_Tasks[Target].insert(tasks[i]);
        DeviceLoad.update(Target);
    end
end
```



Implementation

Vector Search Engine

- ❖ Separate interface for parallel reading
- ❖ Boolean array as the visited list
- ❖ Bitonic sort algorithm for lists updating
- ❖ Loop unrolling and pipelining
- ❖ Data and kernel pooling

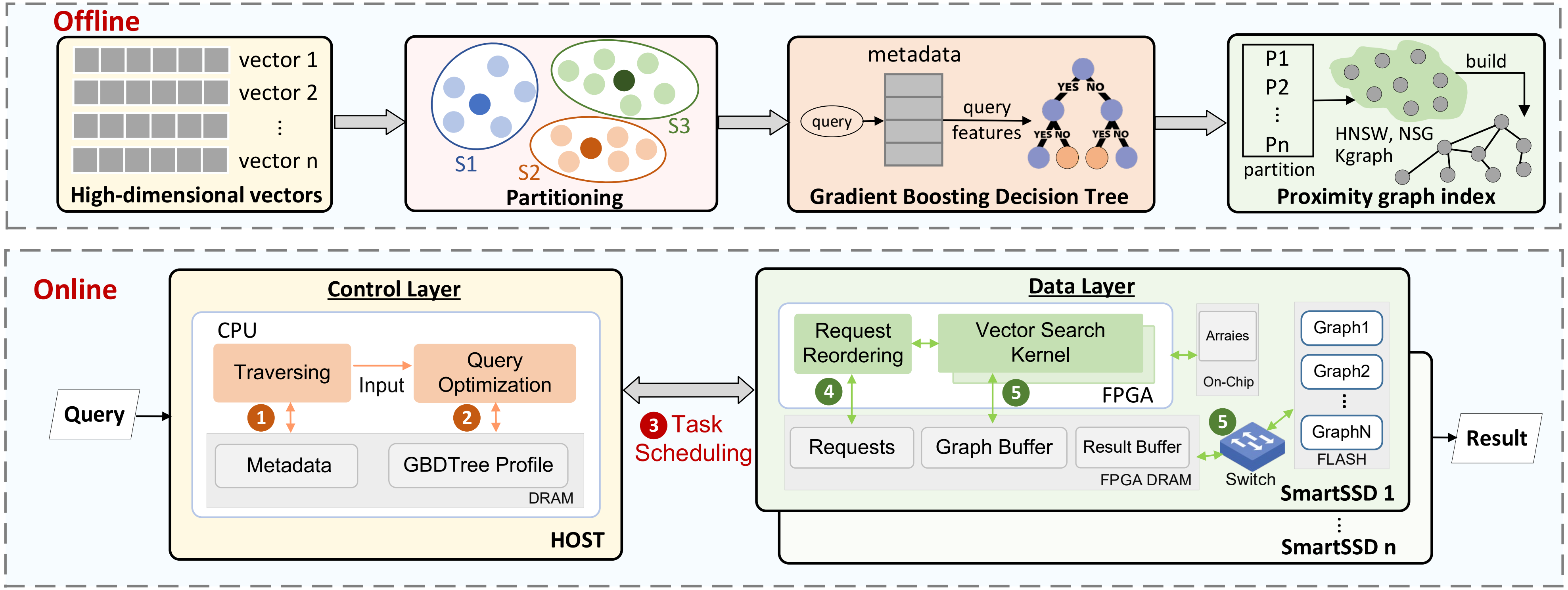


More details: checkout our paper

- ❖ FPGA kernel details
- ❖ GBDT implementation (LAET SIGMOD'20)
- ❖ HNSW search process

SmartANNS System

❖ “host CPU + SmartSSDs” cooperative processing architecture



Outline

- Background and Motivation
- SmartANNS Design
- **Results**
- Conclusion

Experimental Setup

Hardware Platform

Server

CPU	2 * Intel Xeon Gold 5220 CPUs
GPU	Nvidia Tesla V100 (32GB HBM)
DRAM	128 GB DDR4
OS	Ubuntu 20.04.4 LTS

Computational Storage Device

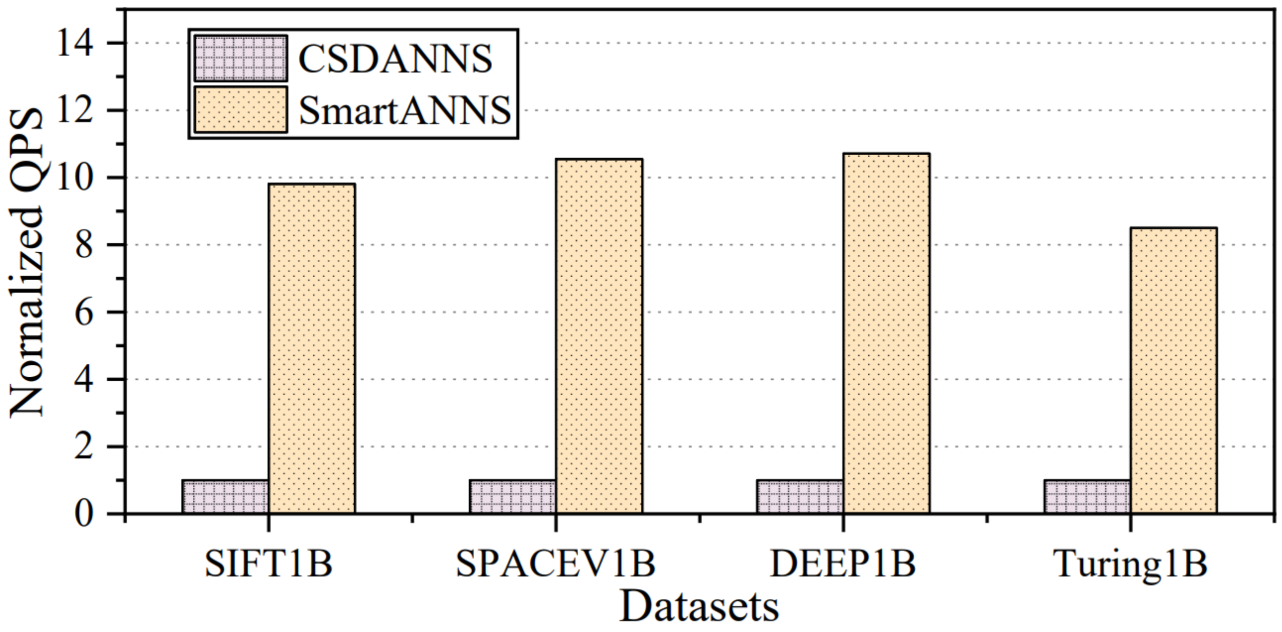
Type	Samsung SmartSSD
FPGA	Xilinx Kintex UltraScale+KU15P
DRAM	4 GB DDR4
Flash	4 TB, 4 GB/s

Datasets

Dataset	Dimension	Data Type	Base Size	Source
SIFT1B	128	Uint8	119 GB	Image
SPACEV1B	100	Int8	93 GB	Web Search
DEEP1B	96	Float32	358 GB	Image
Turing1B	100	Float32	373 GB	Web Search

Comparison with Baselines

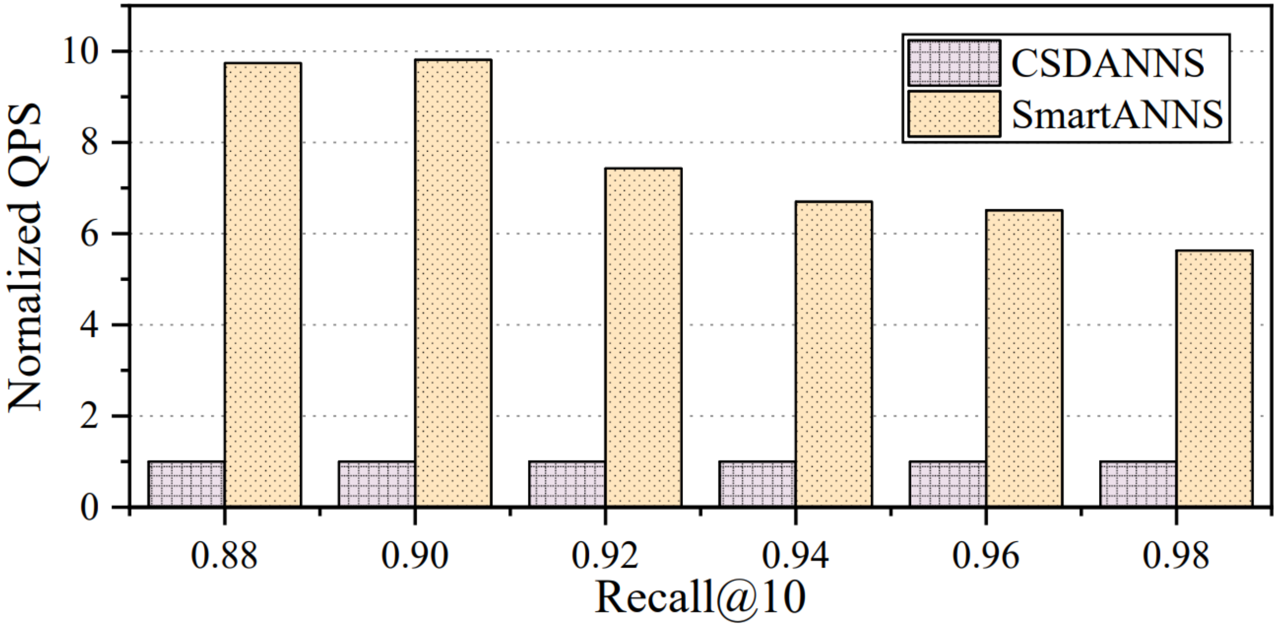
❖ Under different dataset



8.5-10.7X higher QPS compared with the state-of-the-art SmartSSD-based ANNS—CSDANNS

Comparison with Baselines

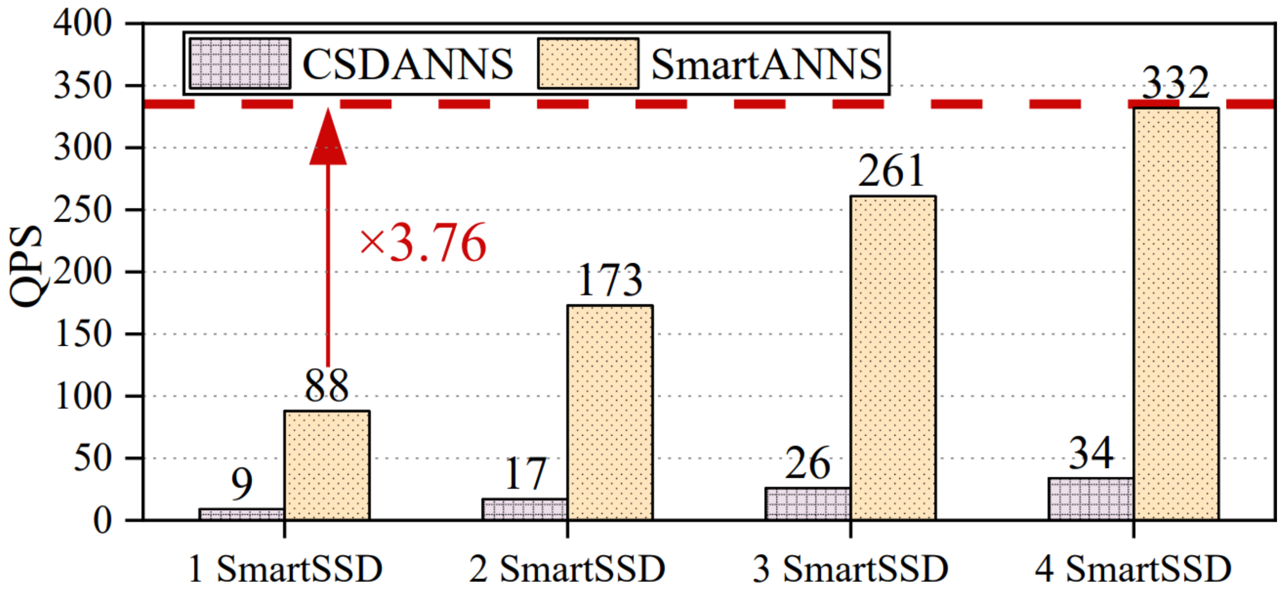
❖ Under different accuracy



With SIFT1B dataset, SmartANNS achieves 5.6-9.8X higher QPS compared with CSDANNS

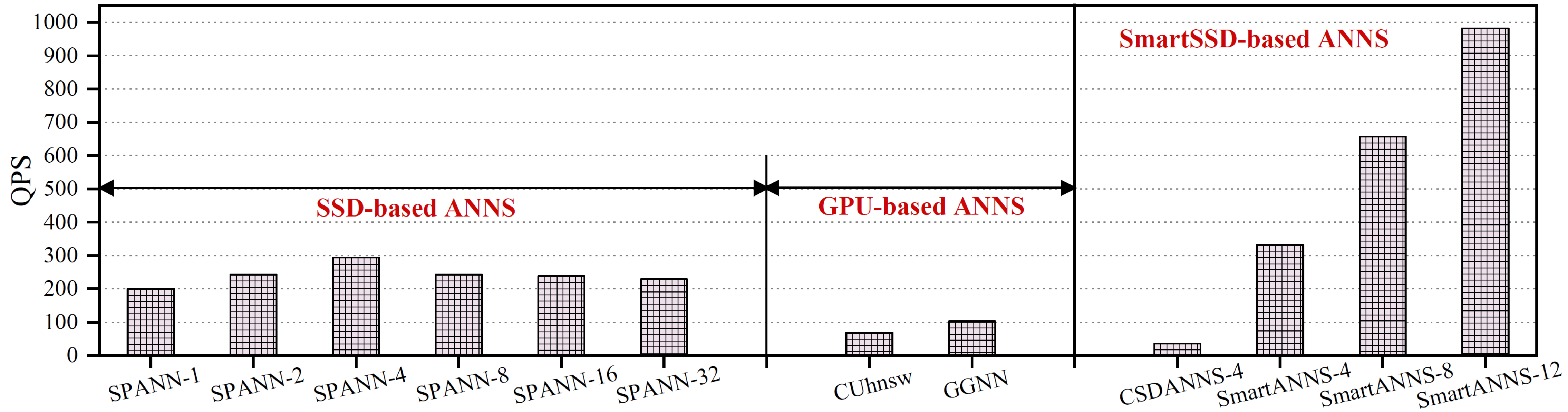
Comparison with Baselines

❖ Scalability



SmartANNS achieve near-linear performance scalability with the increase of SmartSSDS

Comparison with SSD/GPU-based ANNS



SmartANNS is more efficient than SSD-based solution and GPU-based solutions

Outline

- ❖Background and Motivation
- ❖SmartANNS Design
- ❖Results
- ❖Conclusion

❖ SmartANNS:

A hardware/software co-design architecture using SmartSSDs to support the large-scale and scalable ANNS service

Thanks & QA

Contact Information: tbing@hust.edu.cn