CS 764: Topics in Database Management Systems
Lecture 27: GPU Databases

Xiangyao Yu
12/7/2022

Announcements

DAWN workshop

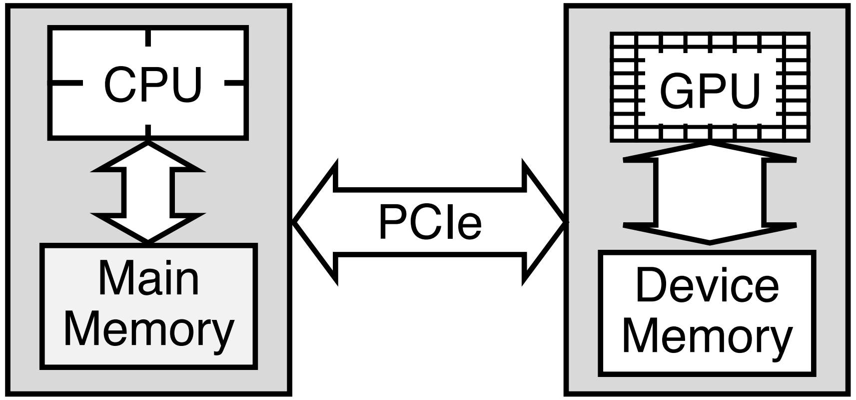
- Reserve a presentation slot using the following google sheet
  https://docs.google.com/spreadsheets/d/1Re1M9FmJwI_YkidhNgeV0iKn-clssFrK_J1PMidaAuw/edit?usp=sharing
- 8-min per group (presentation + QA)

Project report (DDL: Dec. 19)

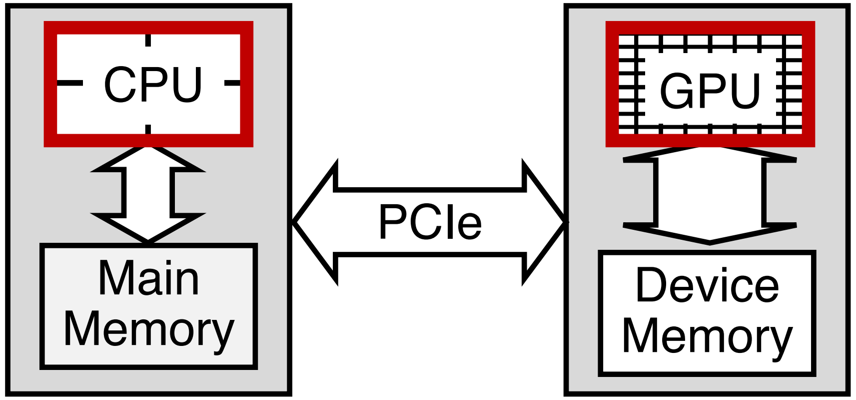
- **Submit to the hotcrp website** (like the proposal)

Submit course evaluation on aefis.wisc.edu

System Architecture



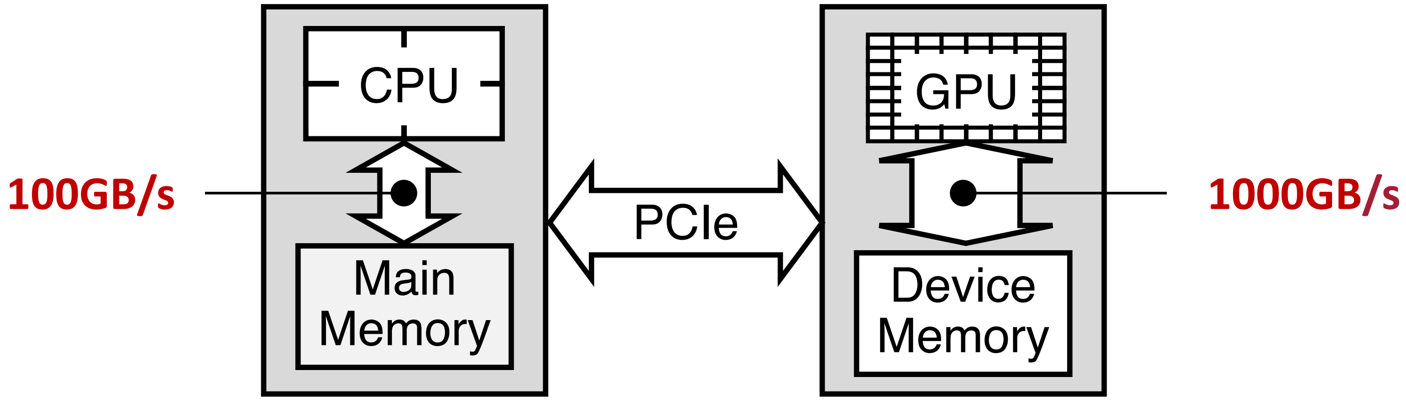
Advantages of GPU for Data Analytics



Advantage 1: **High computational power**

– GPU has massive parallelism using SIMT model

Advantages of GPU for Data Analytics

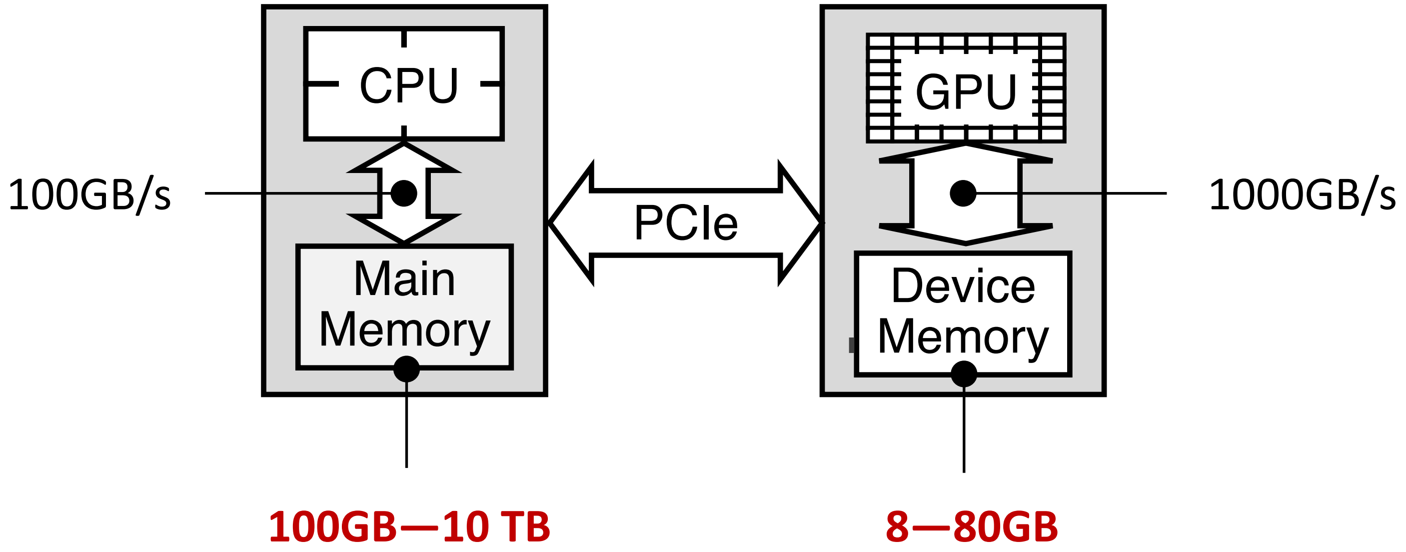


Advantage 1: **High computational power**

Advantage 2: **Higher Memory Bandwidth**

– GPU memory bandwidth is one-order-of-magnitude higher than CPU memory bandwidth

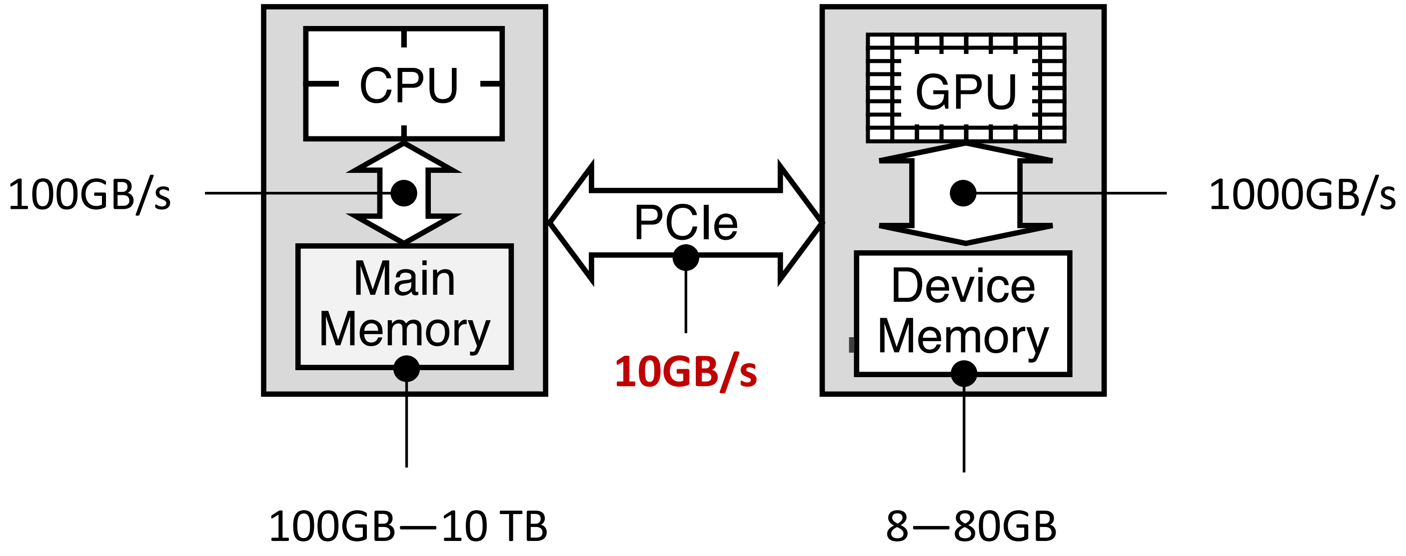
Challenges of GPU for Data Analytics



Challenge 1: **Limited memory capacity**

– Some data sets do not fit in GPU memory

Challenges of GPU for Data Analytics

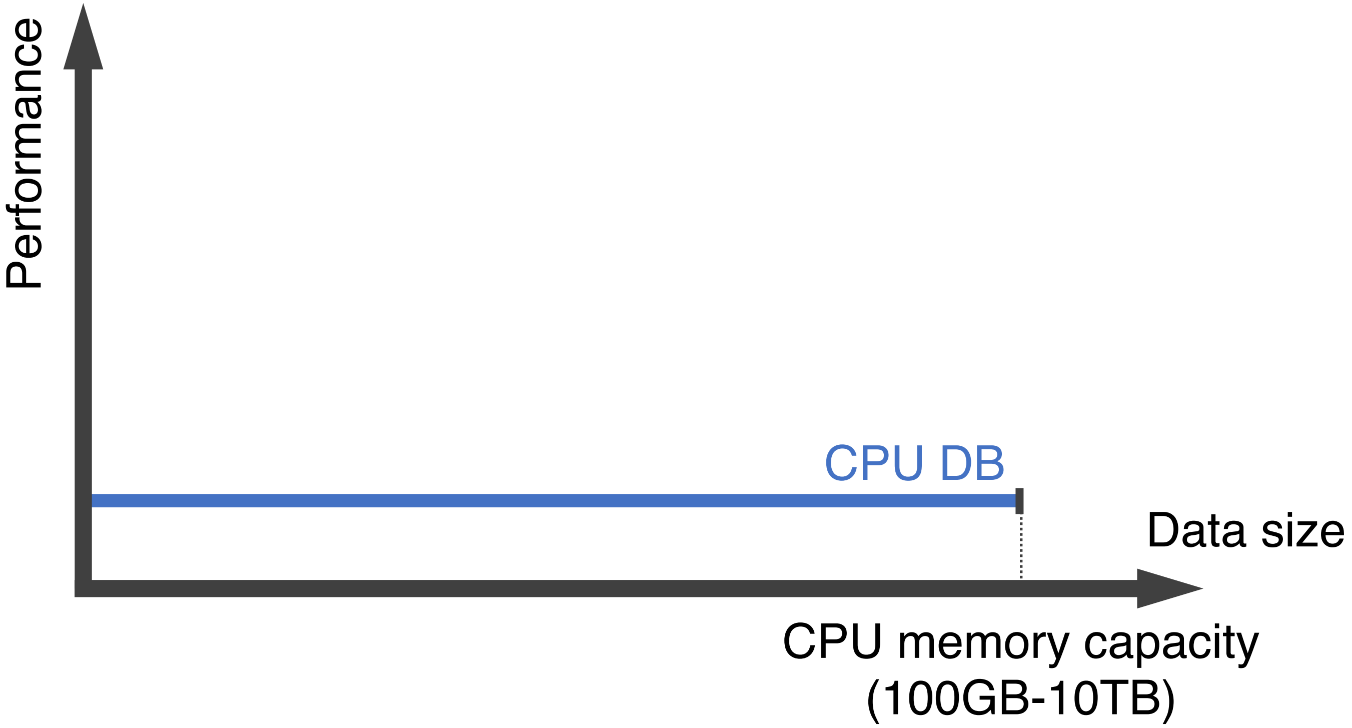


Challenge 1: **Limited memory capacity**

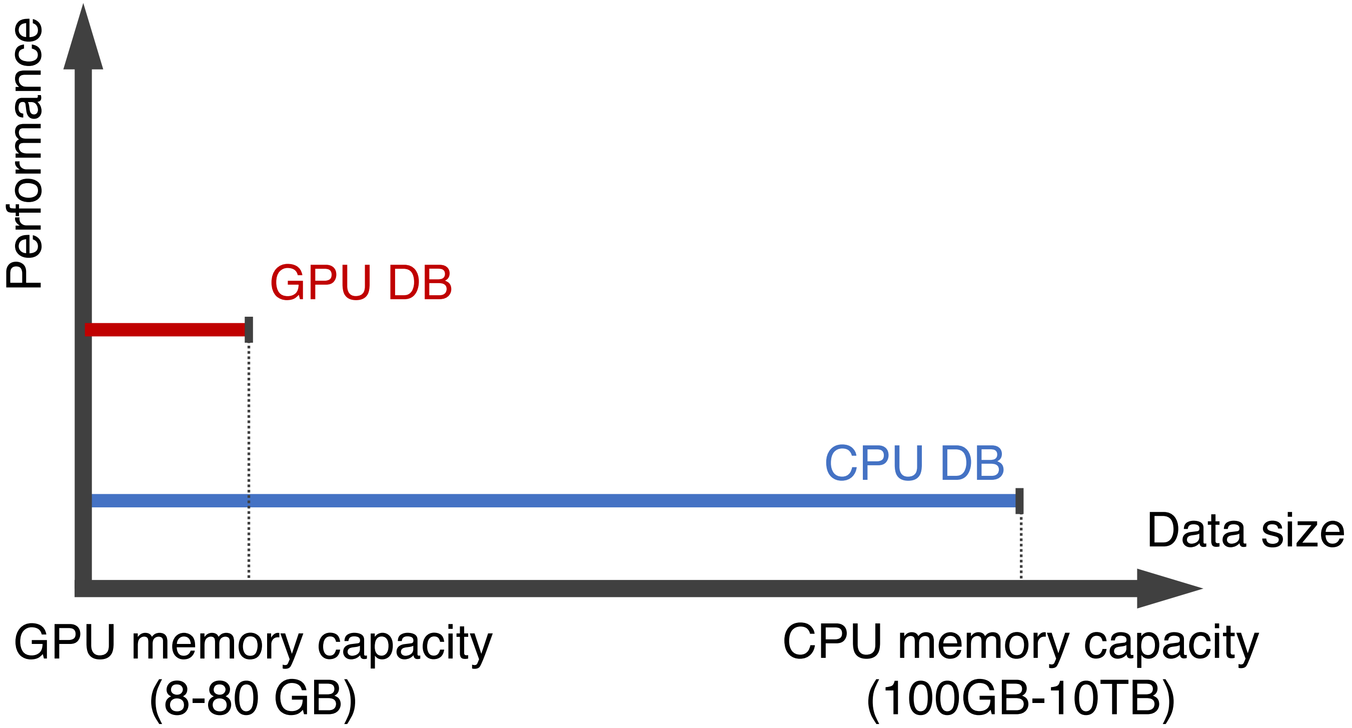
Challenge 2: **Limited interconnect bandwidth**

– Inter-device data transfer is a performance bottleneck

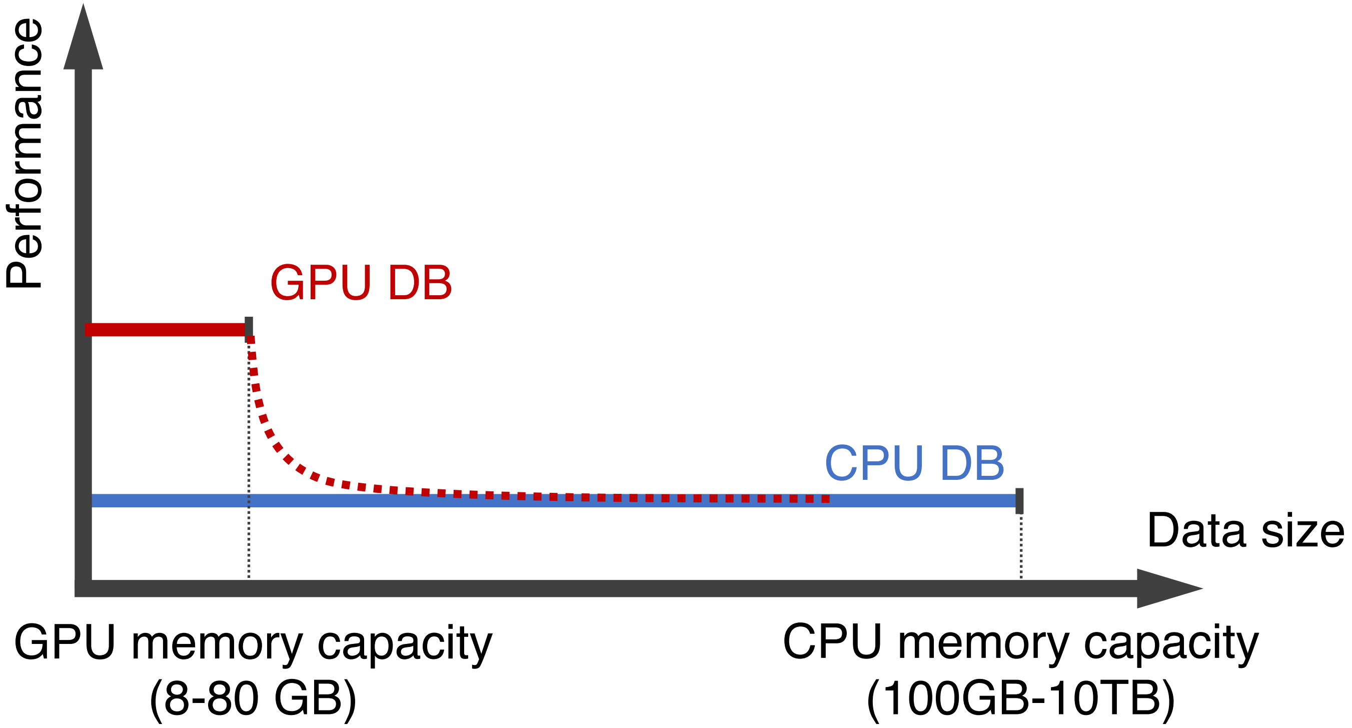
GPU Database Roadmap



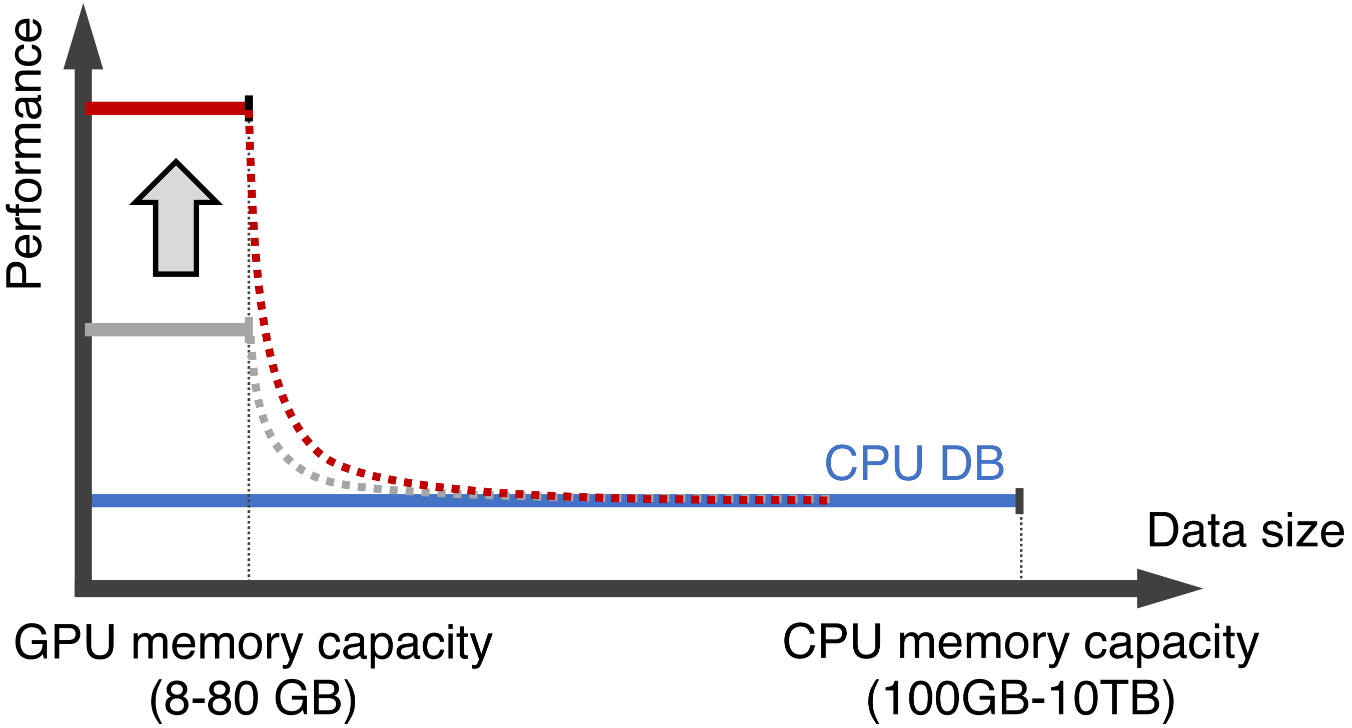
GPU Database Roadmap



GPU Database Roadmap



GPU Database Roadmap

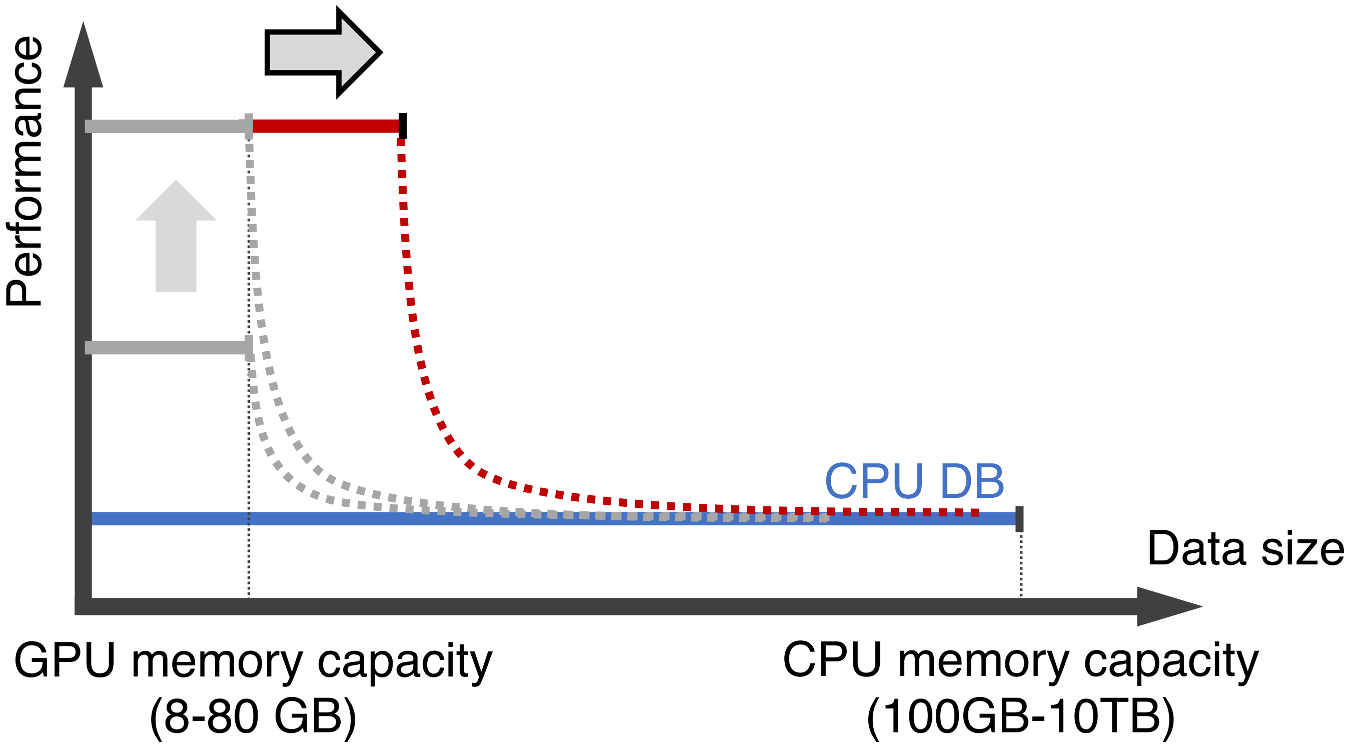


Project 1: **Saturate GPU memory** [1]

- Saturate GPU memory bandwidth when data fits in GPU memory

[1] Anil Shanbhag, Samuel Madden, Xiangyao Yu, *A Study of the Fundamental Performance Characteristics of GPUs and CPUs for Database Analytics*, **SIGMOD 2020**

GPU Database Roadmap



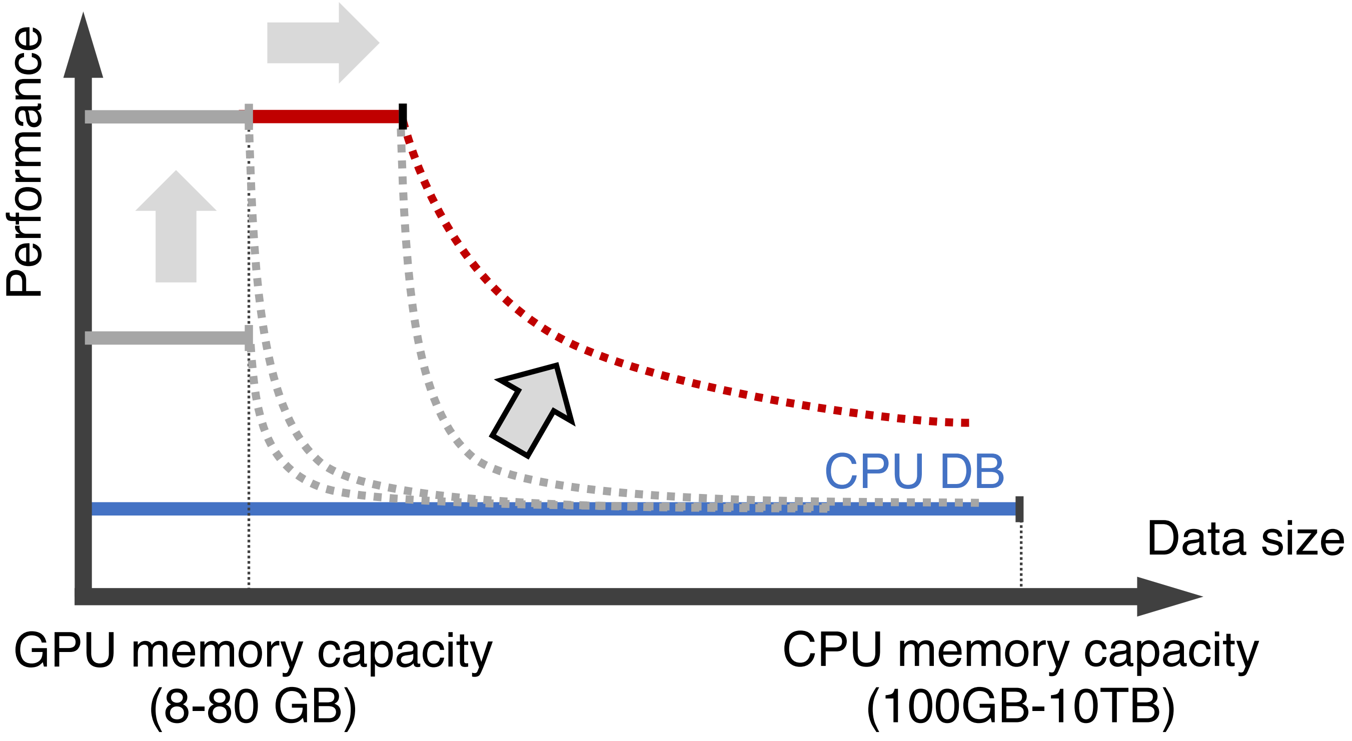
Project 1: **Saturate GPU memory** [1]

Project 2: **Data compression** [2]

- More data can fit in GPU memory

[1] Anil Shanbhag, Samuel Madden, Xiangyao Yu, *A Study of the Fundamental Performance Characteristics of GPUs and CPUs for Database Analytics*, **SIGMOD 2020**
[2] Anil Shanbhag*, Bobbi Yogatama*, Xiangyao Yu, Samuel Madden, *Tile-based Lightweight Integer Compression in GPU*, **SIGMOD 2022**

GPU Database Roadmap



Project 1: **Saturate GPU memory** [1]

Project 2: **Data compression** [2]

Project 3: **Hybrid CPU-GPU DB** [3]

– Leverage both CPU and GPU computation

[1] Anil Shanbhag, Samuel Madden, Xiangyao Yu, *A Study of the Fundamental Performance Characteristics of GPUs and CPUs for Database Analytics*, **SIGMOD 2020**
[2] Anil Shanbhag*, Bobbi Yogatama*, Xiangyao Yu, Samuel Madden, *Tile-based Lightweight Integer Compression in GPU*, **SIGMOD 2022**
[3] Bobbi Yogatama, Weiwei Gong, Xiangyao Yu, *Orchestrating Data Placement and Query Execution in Heterogeneous CPU-GPU DBMS*, **VLDB 2022**

Outline

Project 1: Crystal library for in-GPU data analytics

Project 2: Data compression

Project 3: Hybrid CPU-GPU DB

Outline

Project 1: Crystal library for in-GPU data analytics

Project 2: Data compression

Project 3: Hybrid CPU-GPU DB

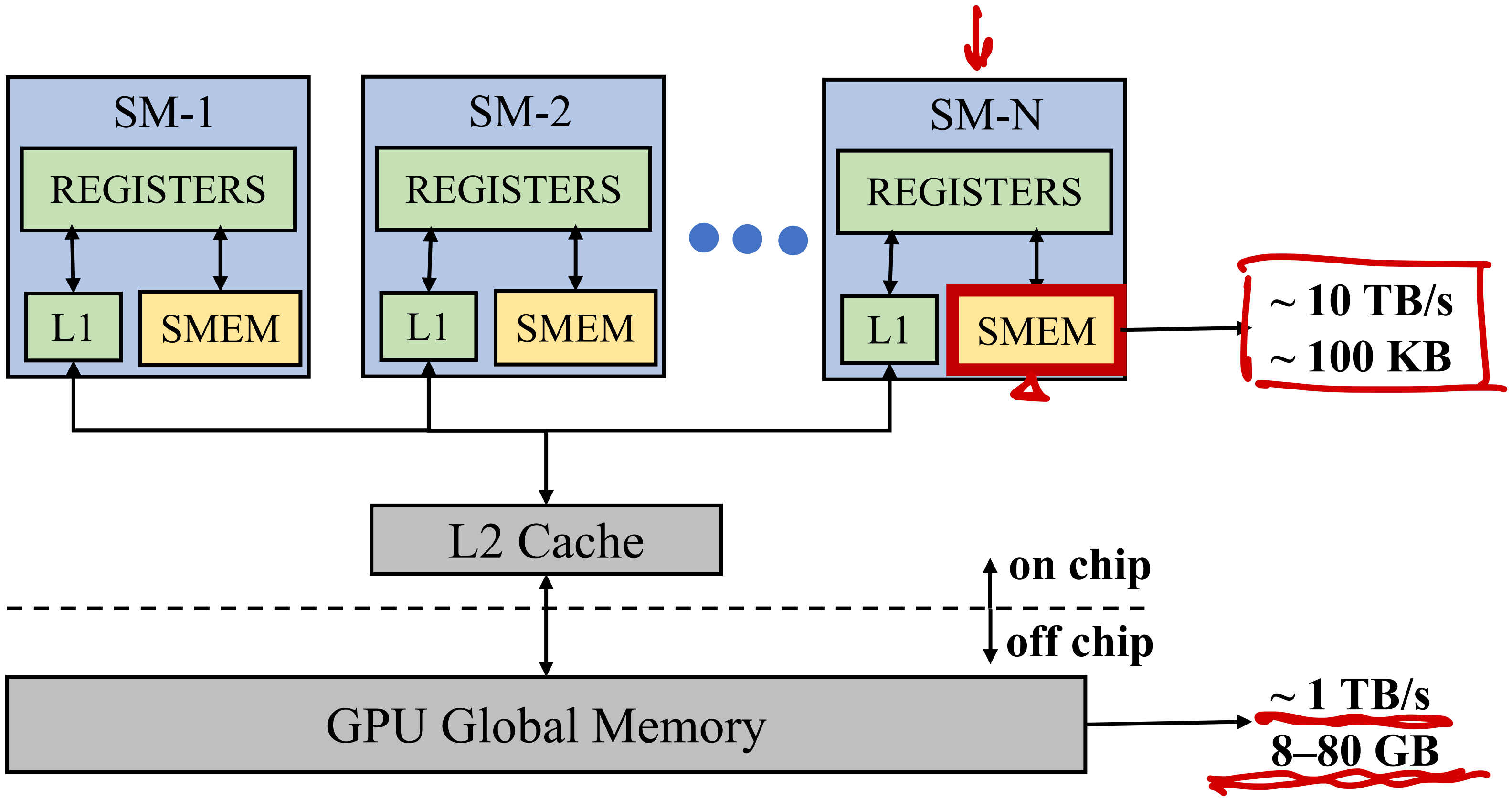
Issues with Prior Work on GPU Database

Past work reported wide variety of gains from **2× to 1000×**

One would expect the maximum gain to be roughly equal to the **ratio of the memory bandwidth** of GPU to that of CPU

Key contribution: developed Crystal library that allows GPU data analytics to saturate GPU memory bandwidth (V100, 880GB/s)

GPU Architecture

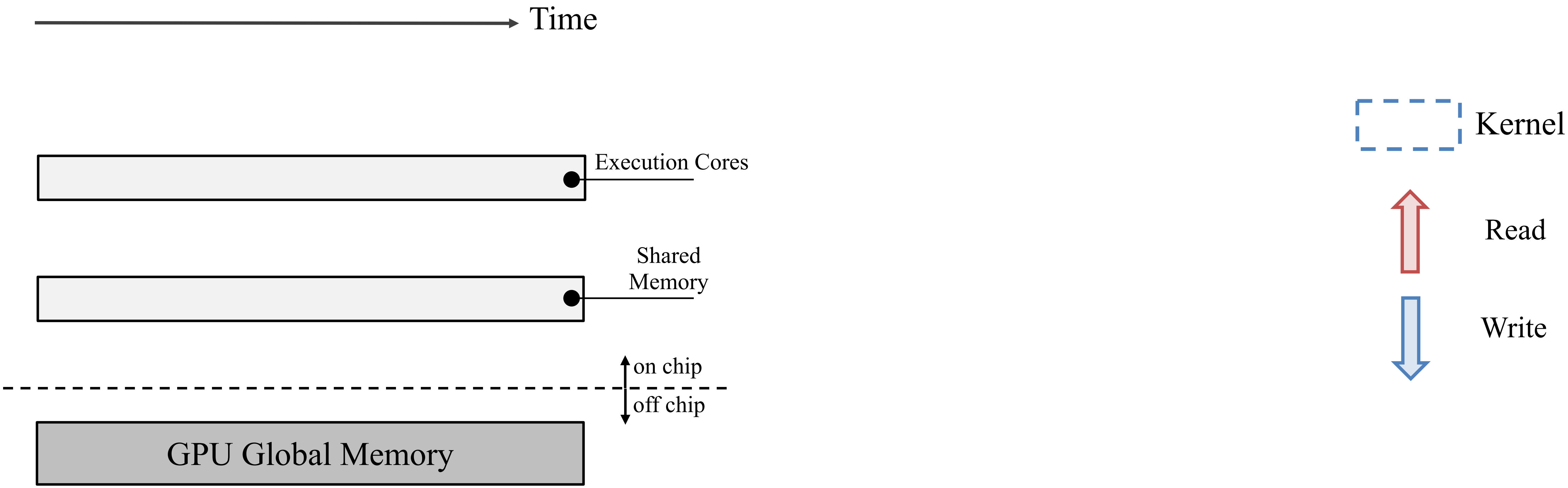


Shared memory (~100 KB) is local to every Streaming Multiprocessors (SM)

Shared memory access is **10x** faster than global memory.

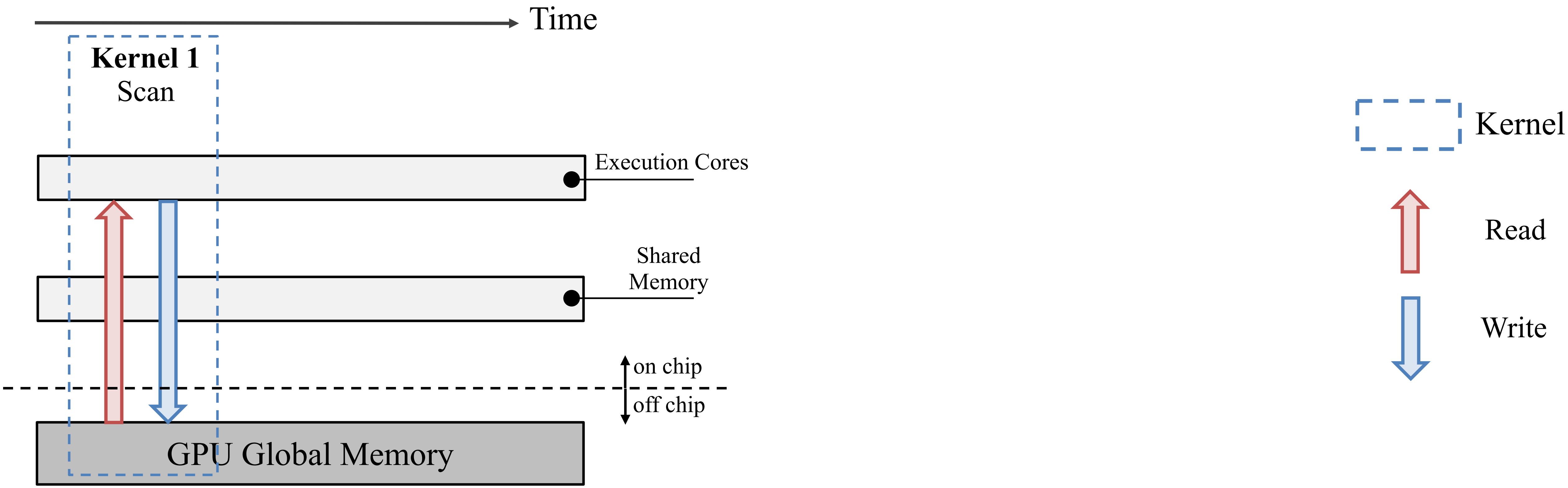
Key Idea: Tile-Based Execution Model

Store intermediate result between operators in **the shared memory (~10x faster)**.



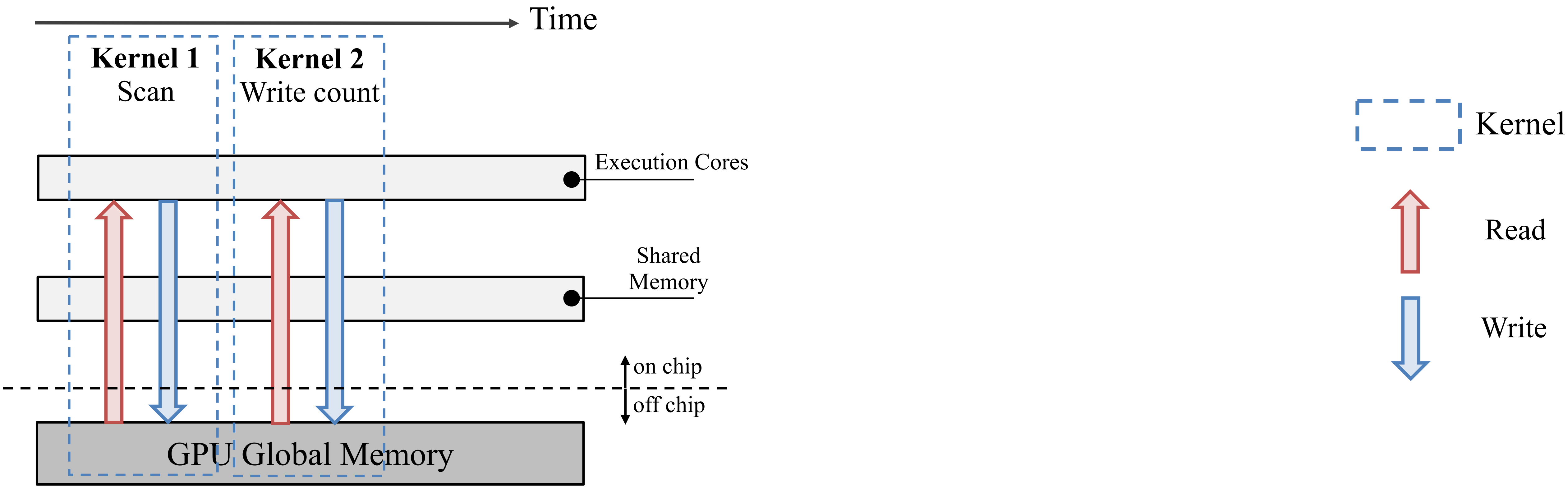
(a) Conventional execution model

Key Idea: Tile-Based Execution Model



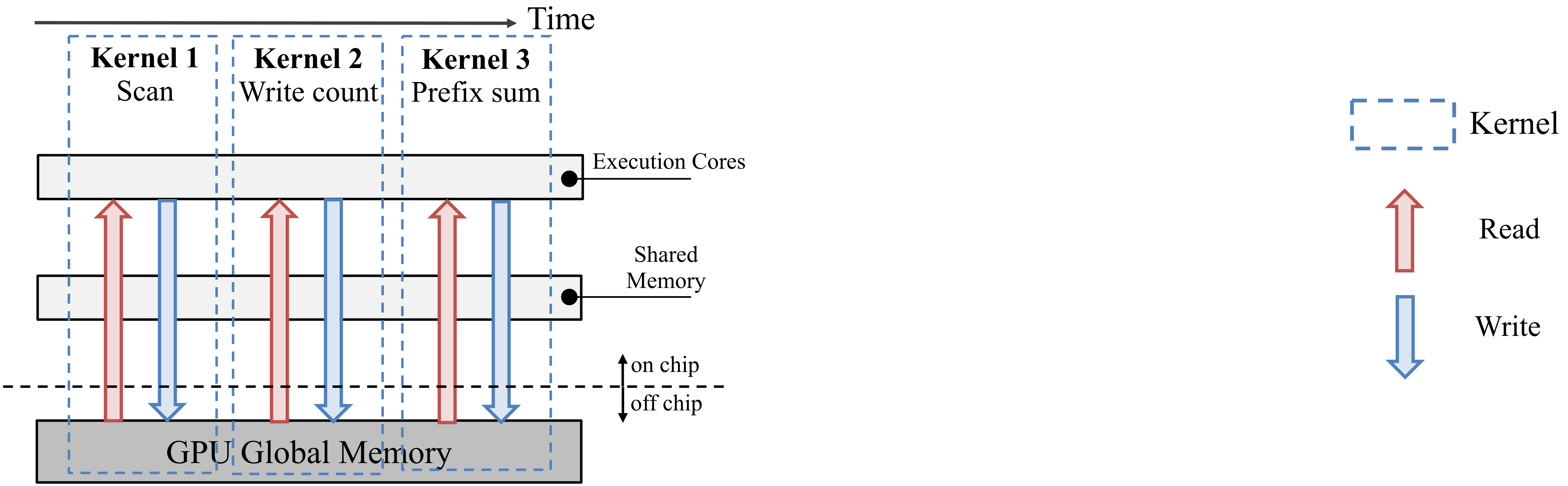
(a) Conventional execution model

Key Idea: Tile-Based Execution Model



(a) Conventional execution model

Key Idea: Tile-Based Execution Model

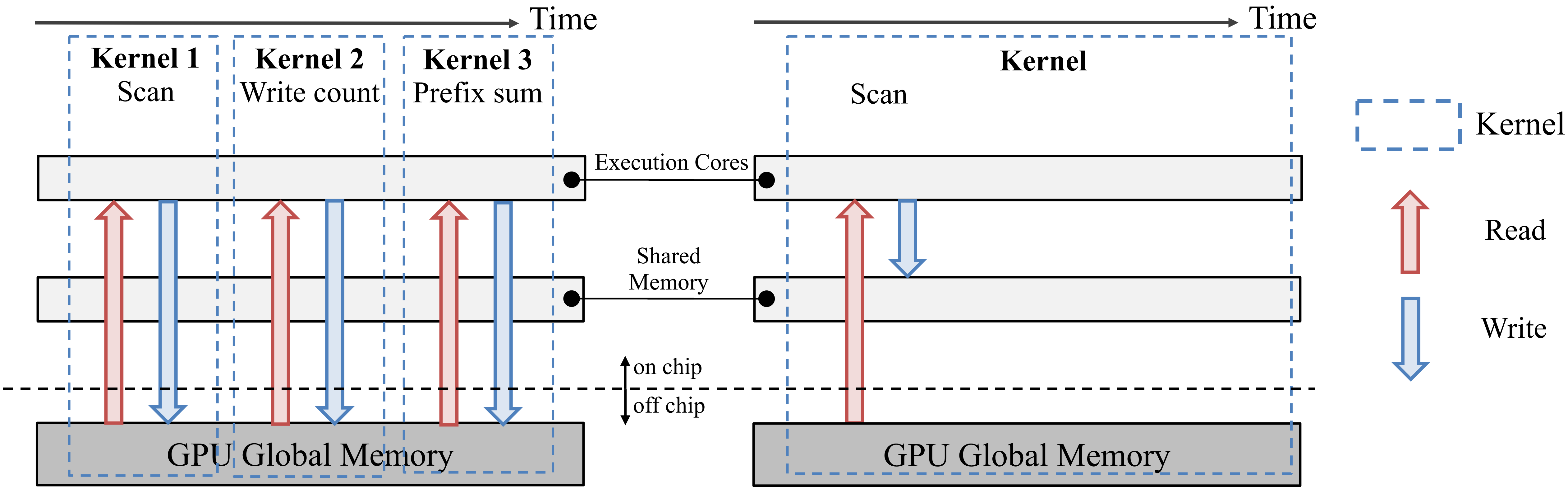


(a) Conventional execution model

Key Idea: Tile-Based Execution Model

Store intermediate result between operators in **the shared memory (~10x faster)**

Data is partitioned and processed in tiles **(each tile must fit in shared memory)**



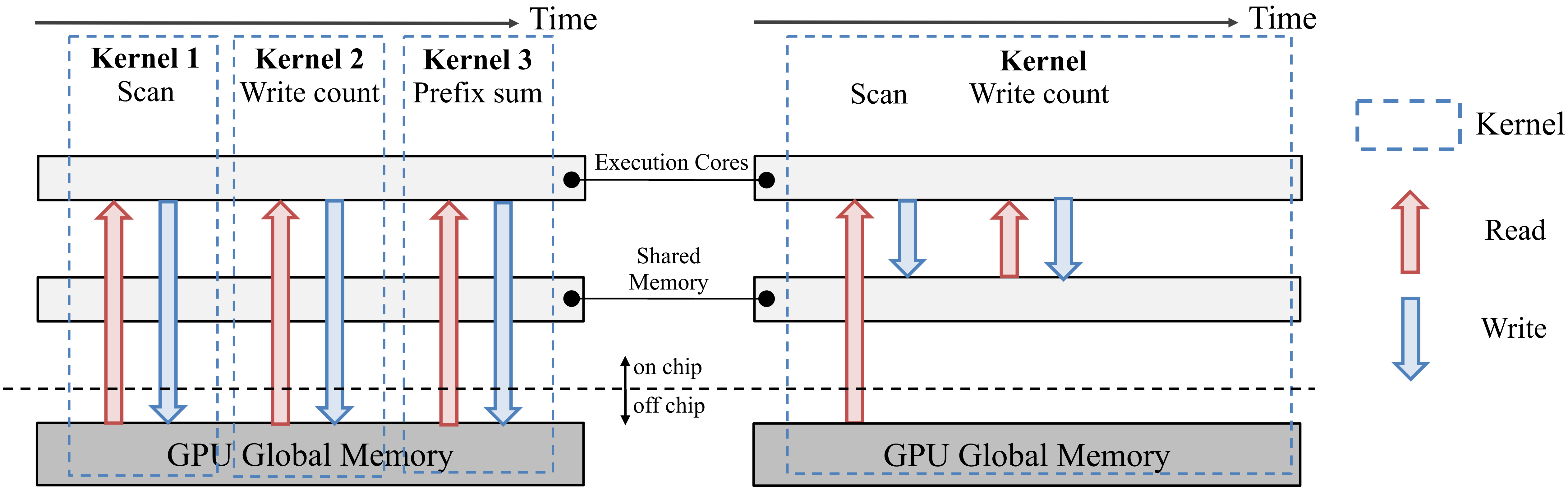
(a) Conventional execution model

(b) Tile-based execution model

Key Idea: Tile-Based Execution Model

Store intermediate result between operators in **the shared memory (~10x faster)**

Data is partitioned and processed in tiles **(each tile must fit in shared memory)**



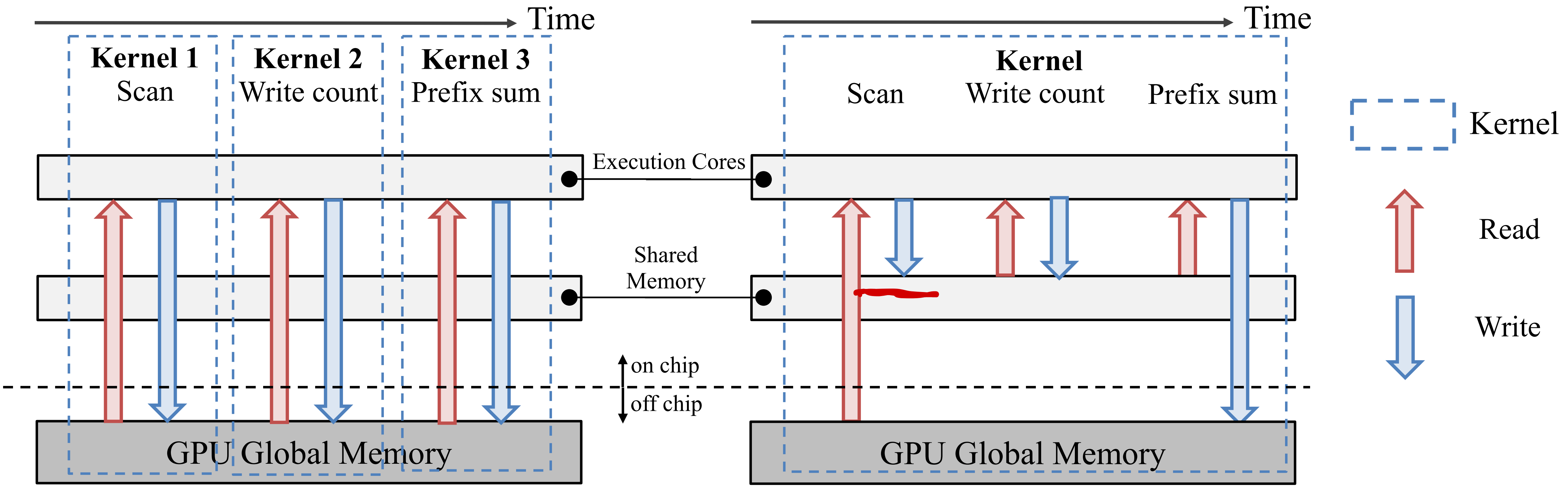
(a) Conventional execution model

(b) Tile-based execution model

Key Idea: Tile-Based Execution Model

Store intermediate result between operators in **the shared memory (~10x faster)**

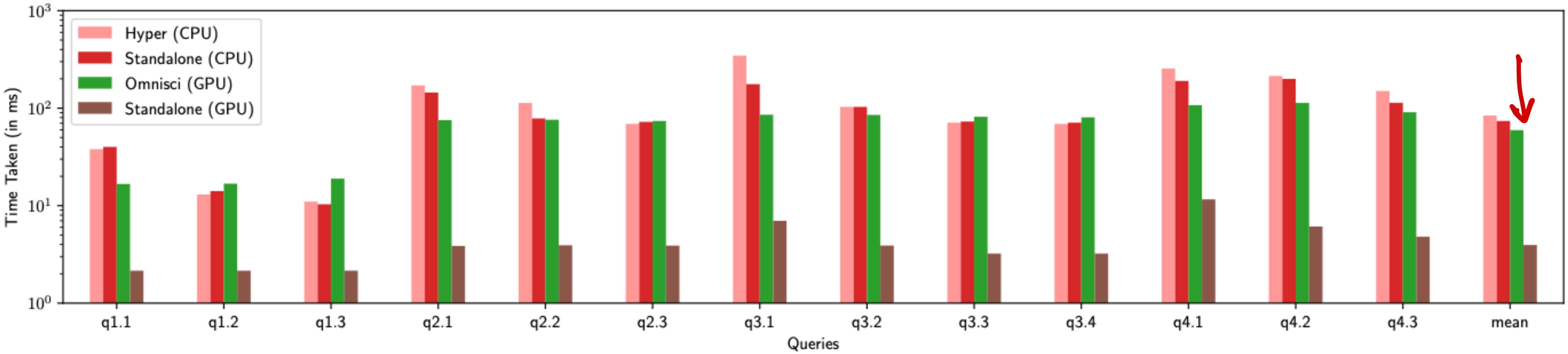
Data is partitioned and processed in tiles **(each tile must fit in shared memory)**



(a) Conventional execution model

(b) Tile-based execution model

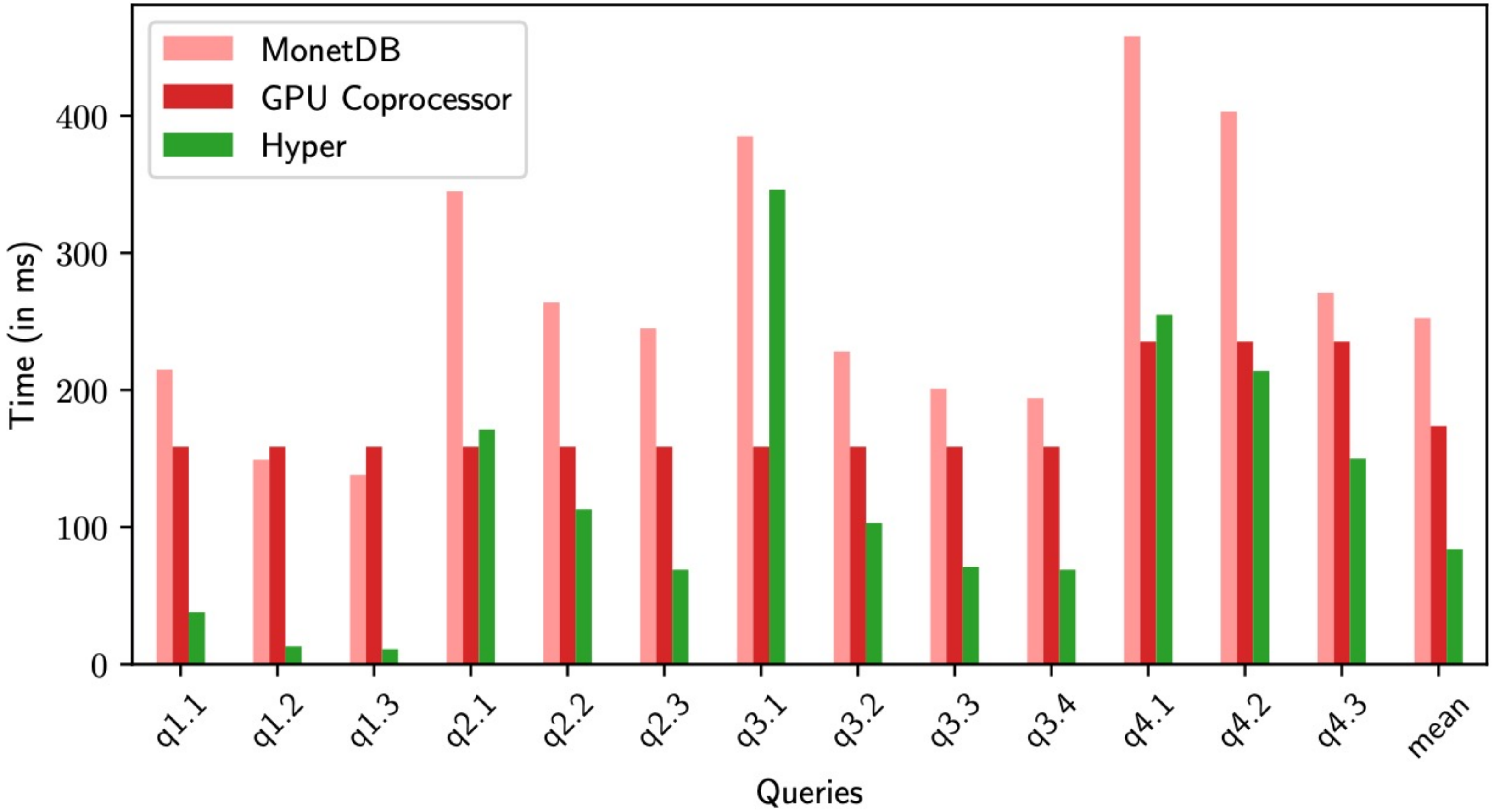
Experimental Results



With Crystal, GPU is on average **25X** faster than CPU running Star-Schema Benchmark (SSB)

[1] Anil Shanbhag, Samuel Madden, Xiangyao Yu, *A Study of the Fundamental Performance Characteristics of GPUs and CPUs for Database Analytics*, SIGMOD 2020

Experimental Results



GPU database with frequent PCIe data transfer can underperform a highly optimized CPU database

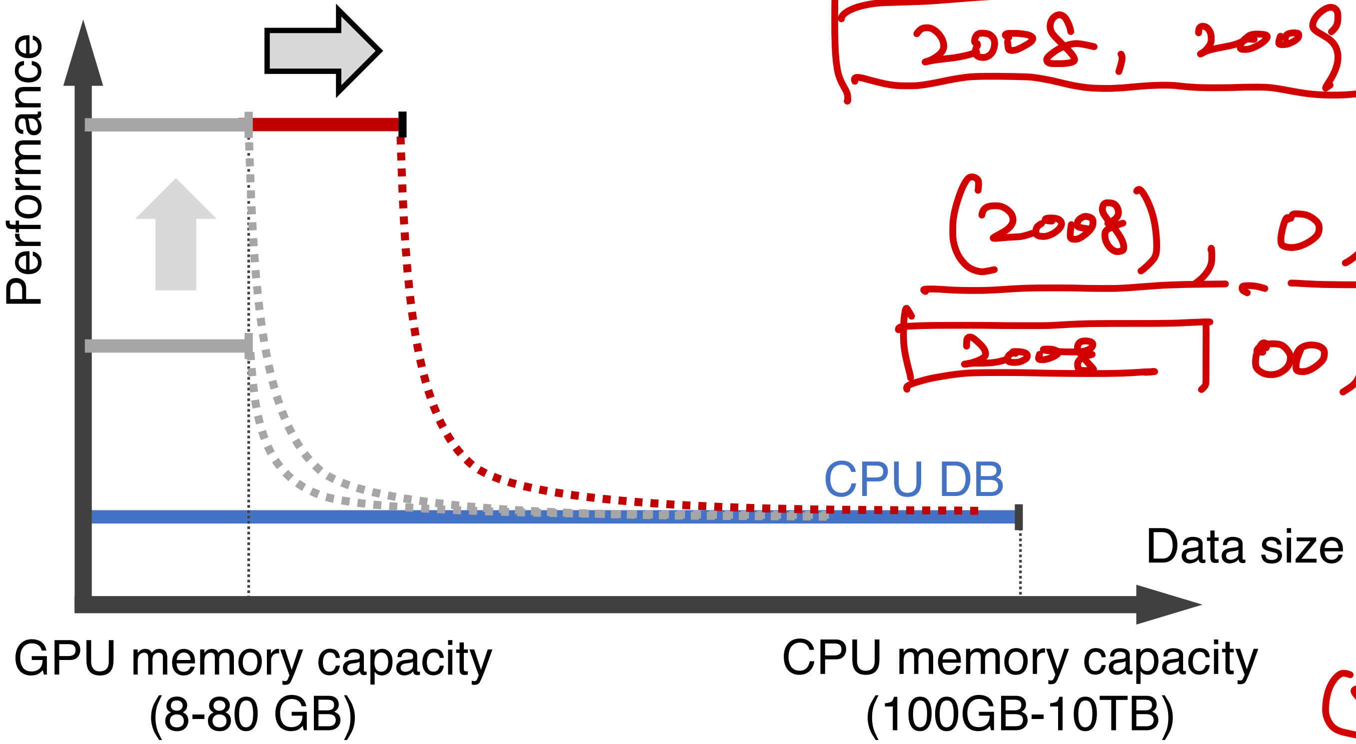
Outline

Project 1: Crystal library for in-GPU data analytics

Project 2: Data compression

Project 3: Hybrid CPU-GPU DB

GPU Data Compression



2008, 2009, 2010, 2011

(2008), 0, 1, 2, 3

2008, 1, 1, 1

2008 | 00, 01, 10, 11

2008, 2008, 2008, 2009, 2009, 2010

(2008, 3), (2009, 2), (2010, 1)

Supported compression schemes: (1) frame-of-reference + bit-packing, (2) delta encoding, (3) run-length encoding

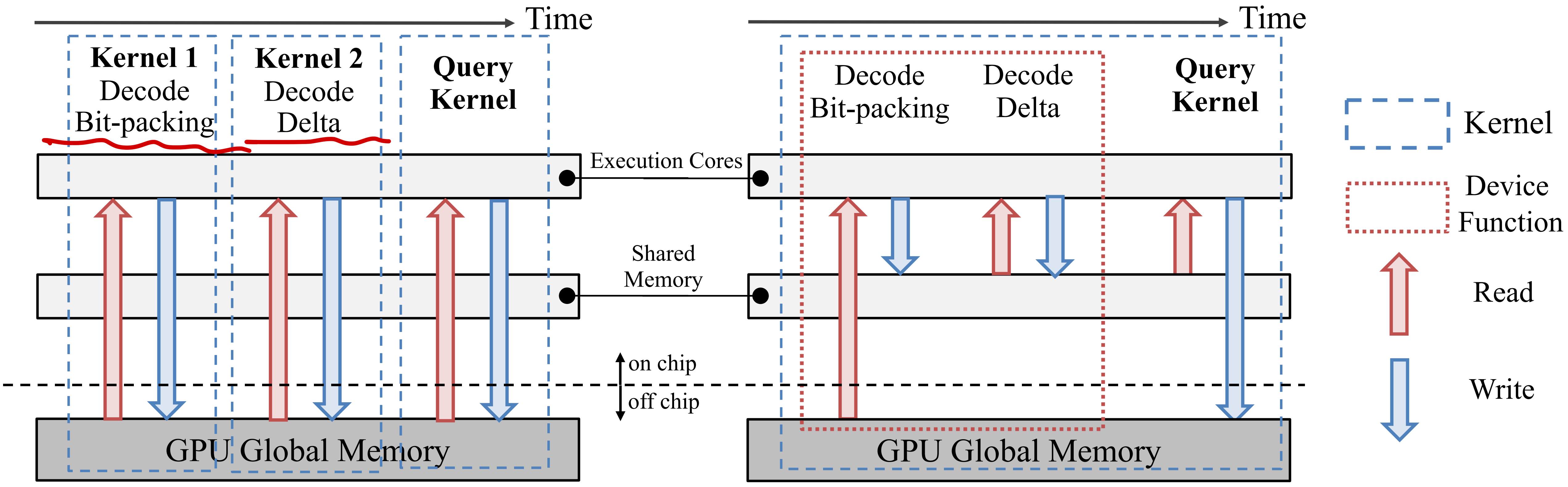
Key challenge: decompression should not be a performance bottleneck

(2008, 2009, 2010) (3, 2, 1)

GPU Data Compression — Key Ideas

Idea 1: **Tile-based decompression**

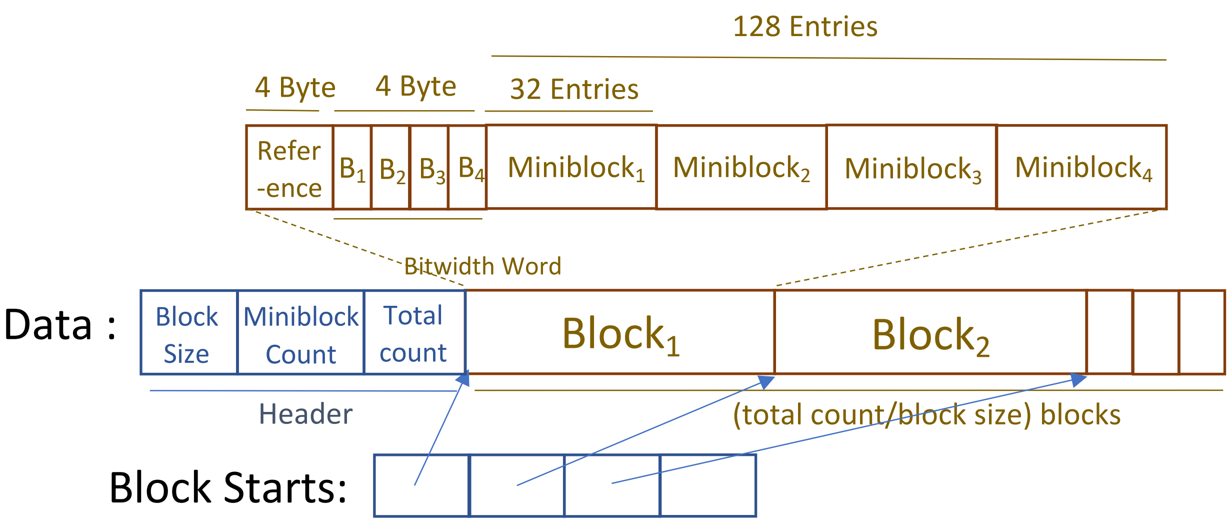
- Multiple decompression steps can be encapsulated into a single device function
- Decompression can be done inline with query execution



(a) Conventional decompression model

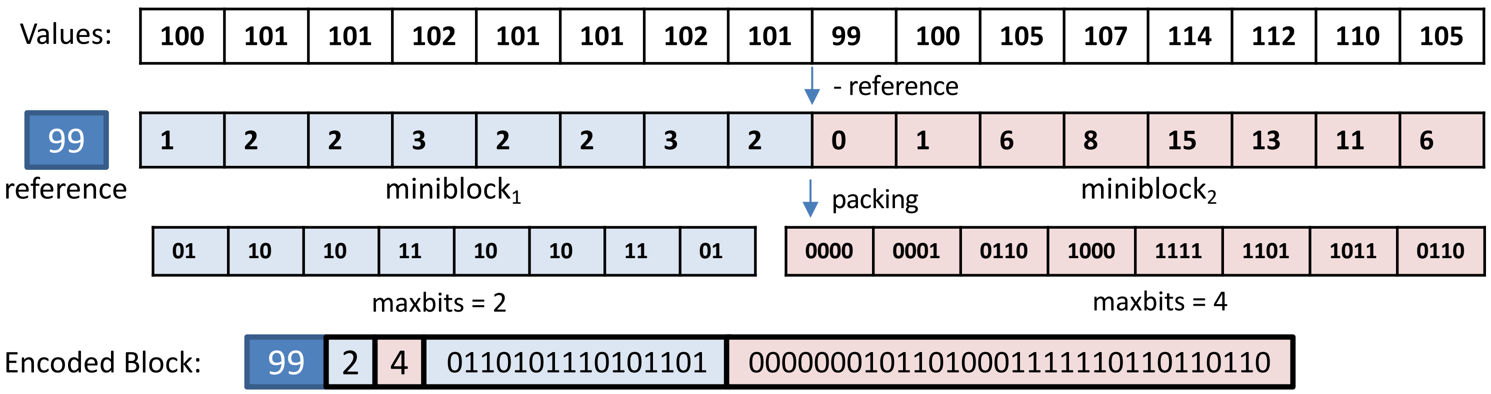
(b) Tile-based decompression model

GPU Data Compression — Key Ideas

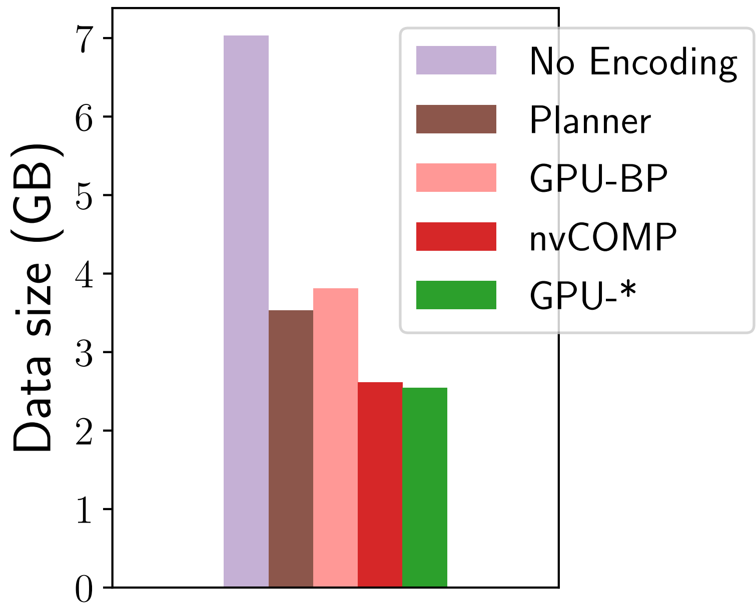


Idea 2: **Efficient bit-packing compression**

- Compact data format
- Low-level performance optimizations



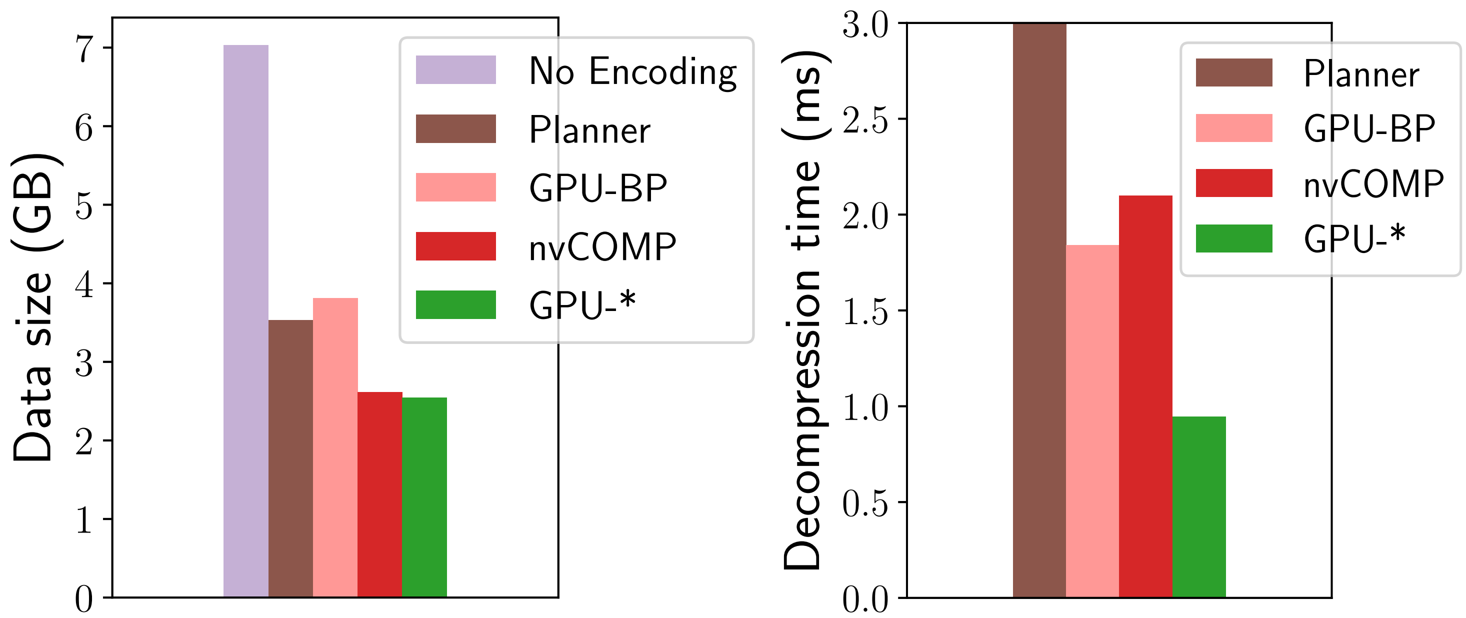
Evaluation – Star Schema Benchmark



(a) Compressed data size

- Our compression rate (**GPU-∗**) is comparable to the best-previous scheme (i.e. nvCOMP).

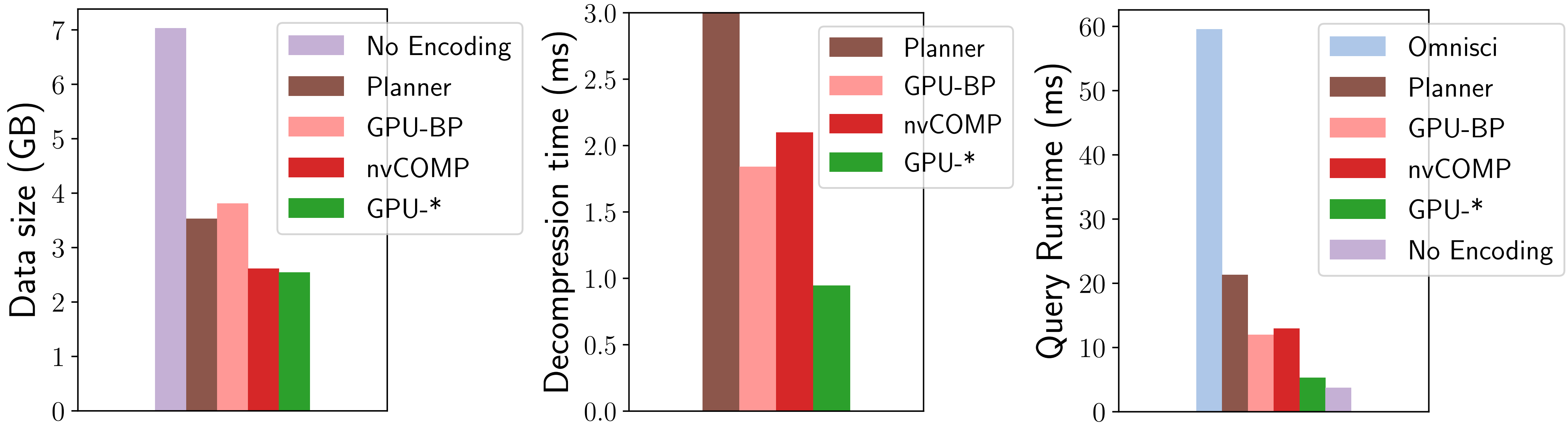
Evaluation – Star Schema Benchmark



(a) Compressed data size **(b) Decompression time**

- Our compression rate (**GPU-∗**) is comparable to the best-previous scheme (i.e. nvCOMP).
- **GPU-∗** is **2.2x** faster in decompression time than the best-previous scheme.

Evaluation – Star Schema Benchmark



(a) Compressed data size **(b) Decompression time** **(c) Query runtime**

- Our compression rate (**GPU-∗**) is comparable to the best-previous scheme (i.e. nvCOMP).
- **GPU-∗** is **2.2x** faster in decompression time than the best-previous scheme.
- **GPU-∗** is **2.6x** faster in query running time than the best-previous scheme.
- **GPU-∗** executes queries with minimal performance degradation compared to no encoding.

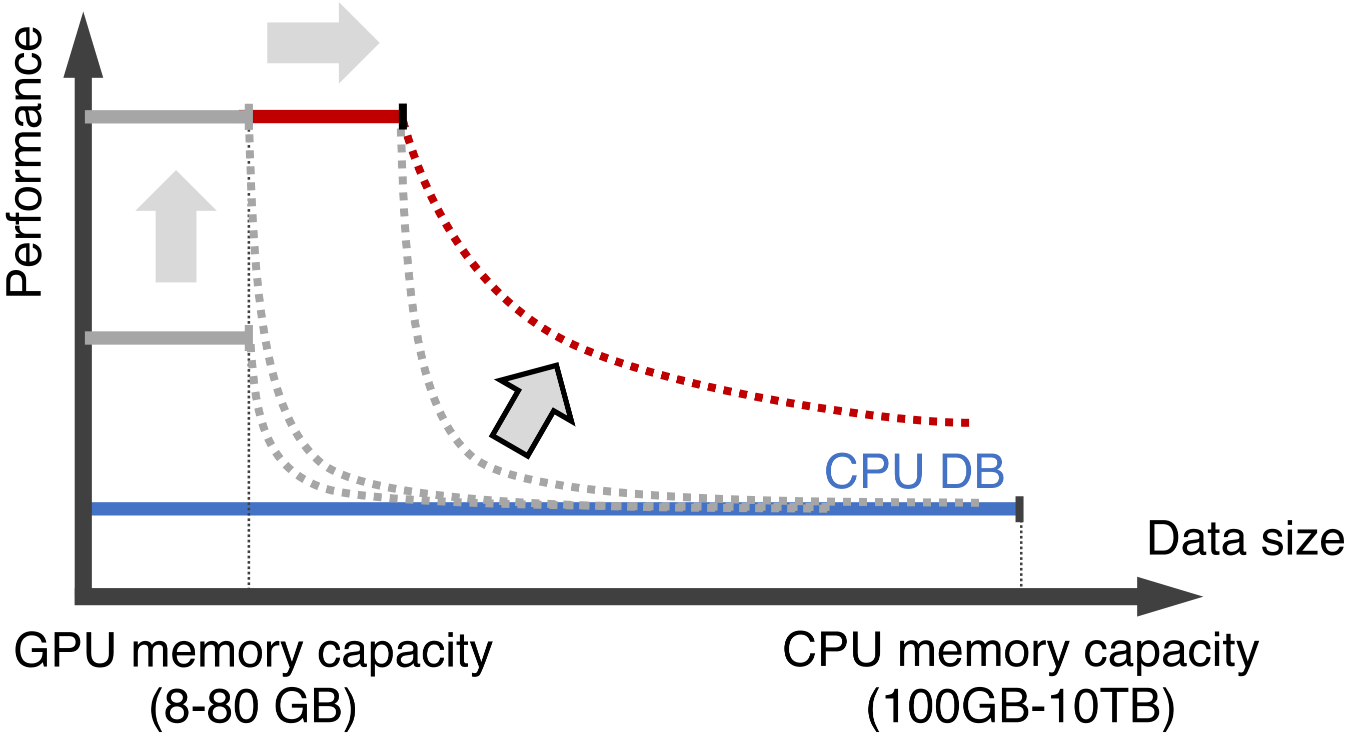
Outline

Project 1: Crystal library for in-GPU data analytics

Project 2: Data compression

Project 3: Hybrid CPU-GPU DB

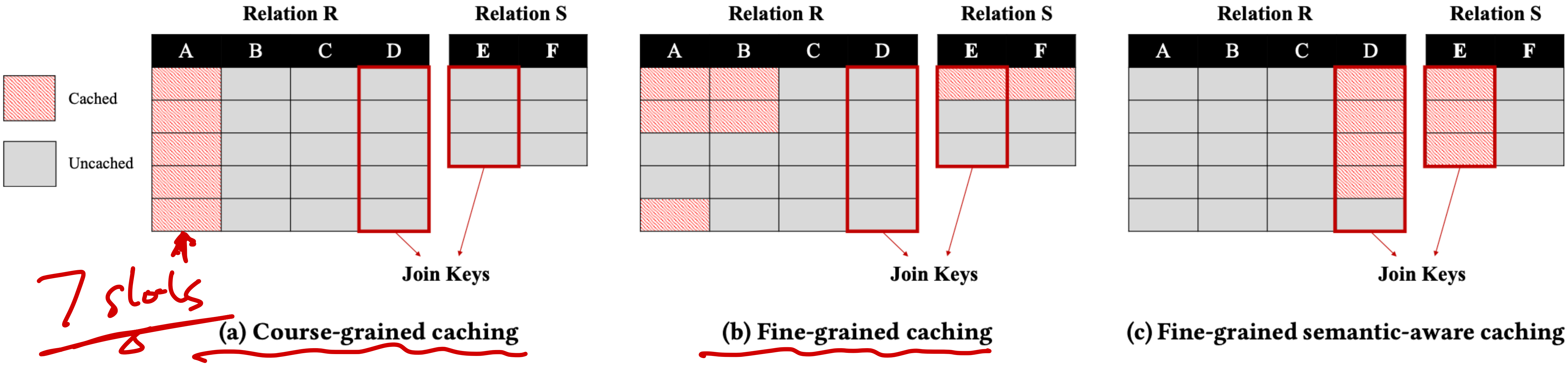
Heterogeneous GPU-CPU Data Analytics



Challenges in heterogeneous CPU-GPU data analytics

- **Data placement**: hot data should be cached in GPU memory
- **Heterogeneous query execution**: hybrid execution with minimal inter-device data transfer

Data Placement



(a) Course-grained caching

(b) Fine-grained caching

(c) Fine-grained semantic-aware caching

GPU data caching should be:

1. **Fine-grained**: segment instead of column
2. **Semantic-aware**: priority of caching depends on the query pattern

Fine-grained Semantic-Aware Caching

Algorithm 1: Update the *weighted frequency counter* for segment *S*

```
UpdateWeightedFreqCounter(segment S)
    # estimate query runtime when S is not cached.
    RT_uncached = estimateQueryRuntime(cached_segments \ S)
    # estimate query runtime when S and segments correlated with S
      are cached.
    RT_cached = estimateQueryRuntime(cached_segments ∪ S ∪
      correlated_segments)
    weight = RT_uncached − RT_cached
    S.weighted_freq_counter += weight
    for C in correlated_segments do
        # evenly distribute weight to all segments correlated with S
        C.weighted_freq_counter += weight / |correlated_segments|
```

Weighted LFU replacement

- Each segment is assigned a different weight (higher weight -> higher priority)

The weight of a segment reflects:

1. The relative speedup of caching a segment.
2. Correlation among segments from different columns

estimateQueryRuntime() uses a model to predict runtime, assuming bandwidth as the bottleneck

Heterogeneous Query Execution

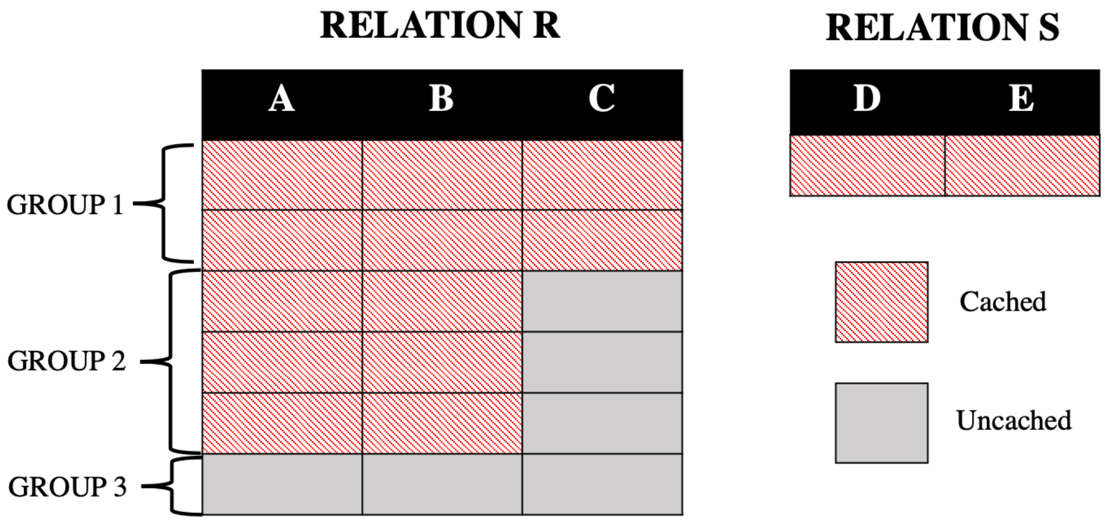
Challenge:

- Extra complexity of query execution due to only subset of data cached in GPU
- Query executor should fully exploit the data in GPU and coordinate query execution across two devices

Solution:

- Segment-level query execution

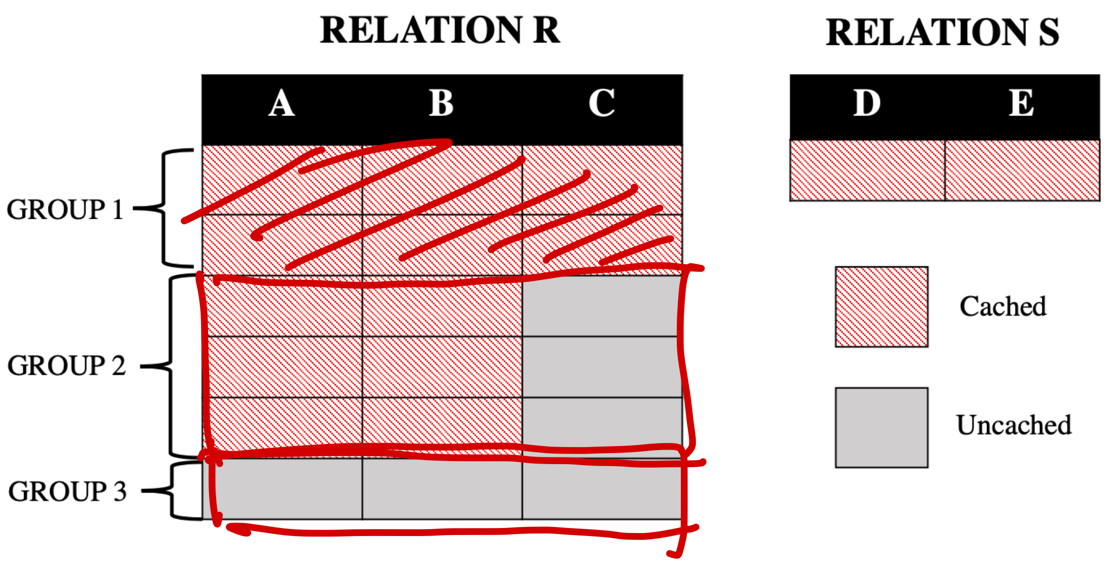
Heterogeneous Query Execution



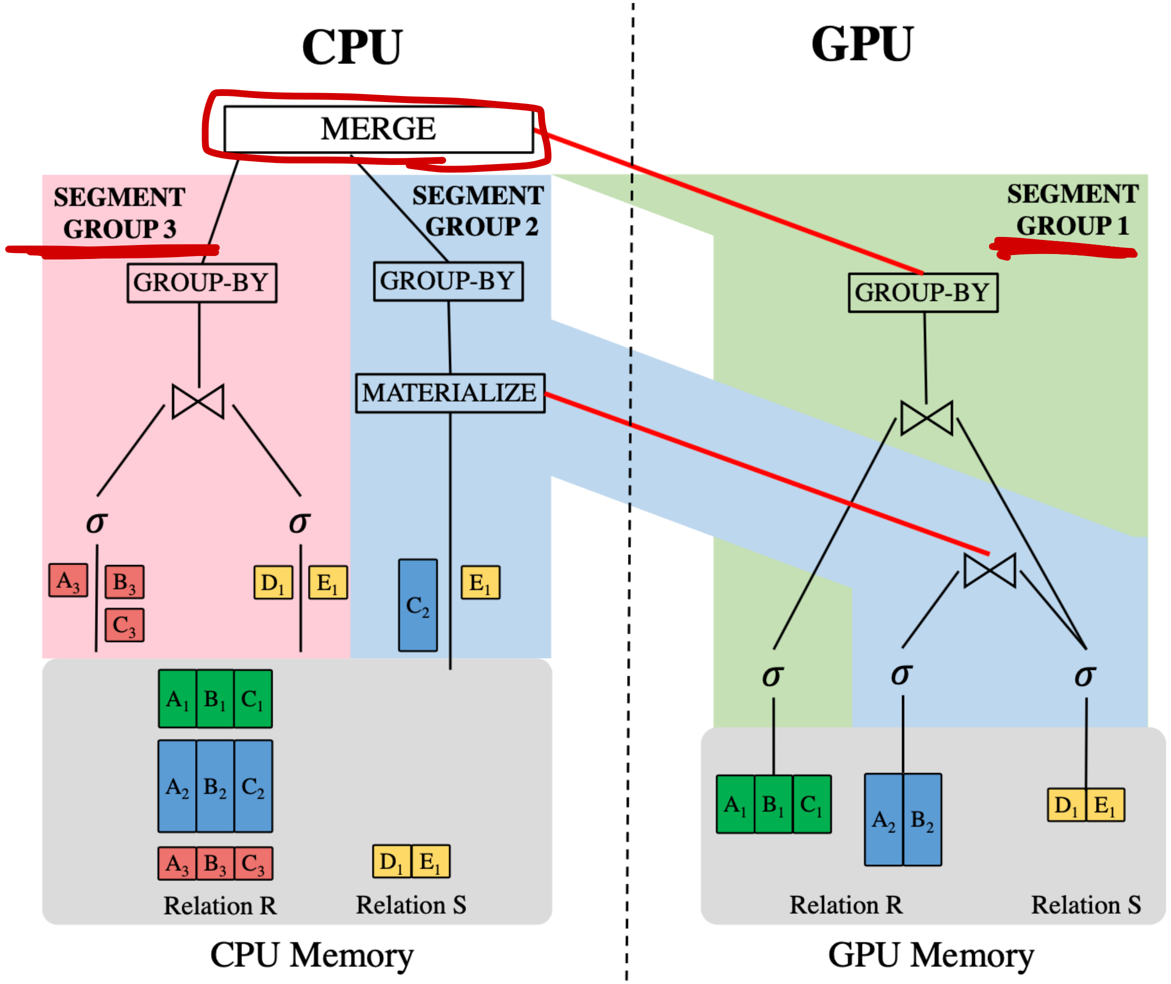
Segment level execution

– Group segments with the same execution plan into **segment groups**

Heterogeneous Query Execution




```
SELECT S.D, SUM(R.C) FROM R,S
WHERE R.B = S.D AND R.A > 10 AND S.E > 20
GROUP BY S.E
```



Segment level execution

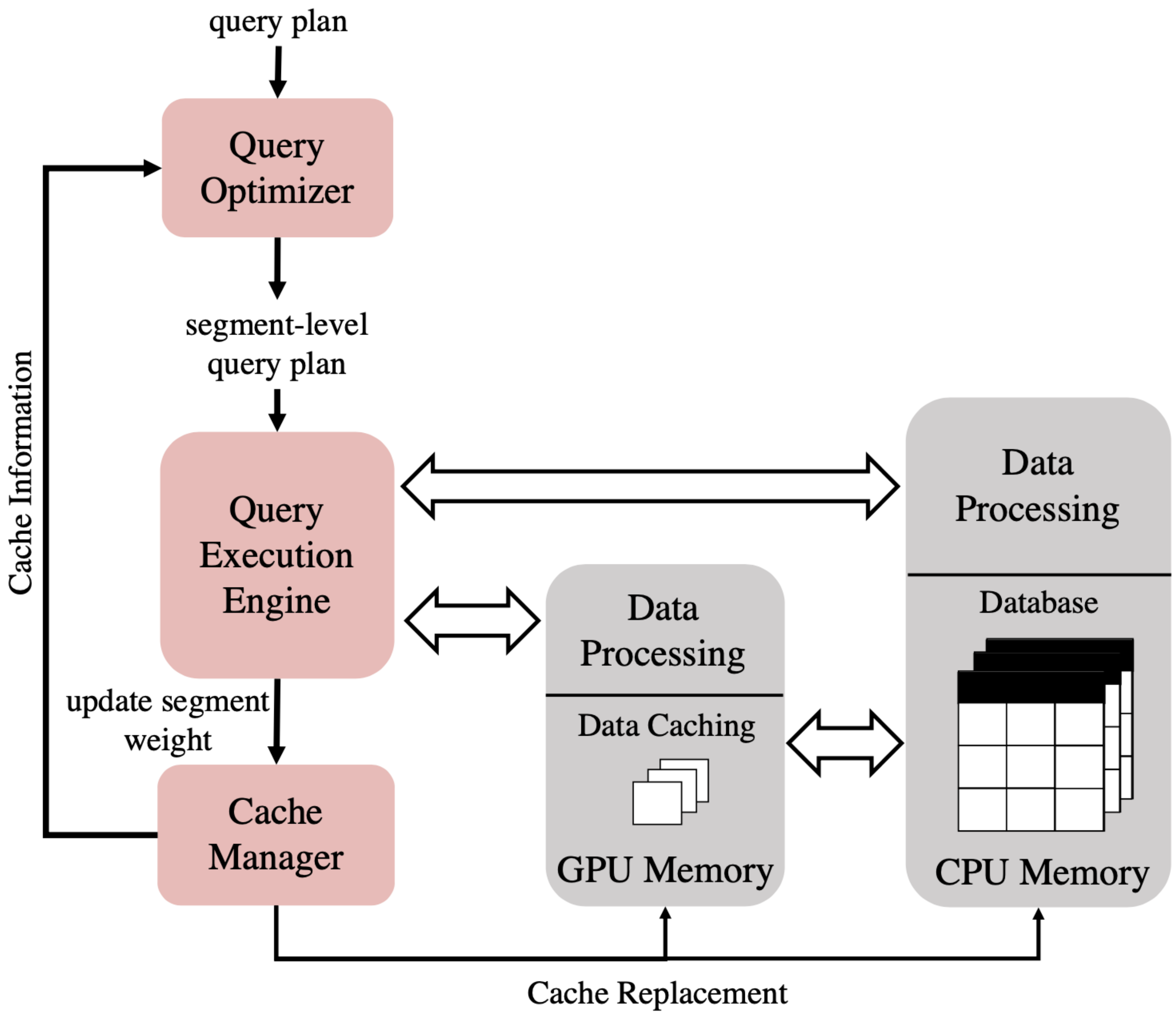
- Group segments with the same execution plan into **segment groups**
- Execute each segment group and merge the results

Heterogeneous Query Execution

Other optimizations

- **Late materialization**: express intermediate relation in the form of row ID to reduce the total data transfer
- **Operator pipelining**: pipelining consecutive operators on the same device whenever possible
- **Segment skipping**: apply minmax pruning both for predicate evaluation and join operator

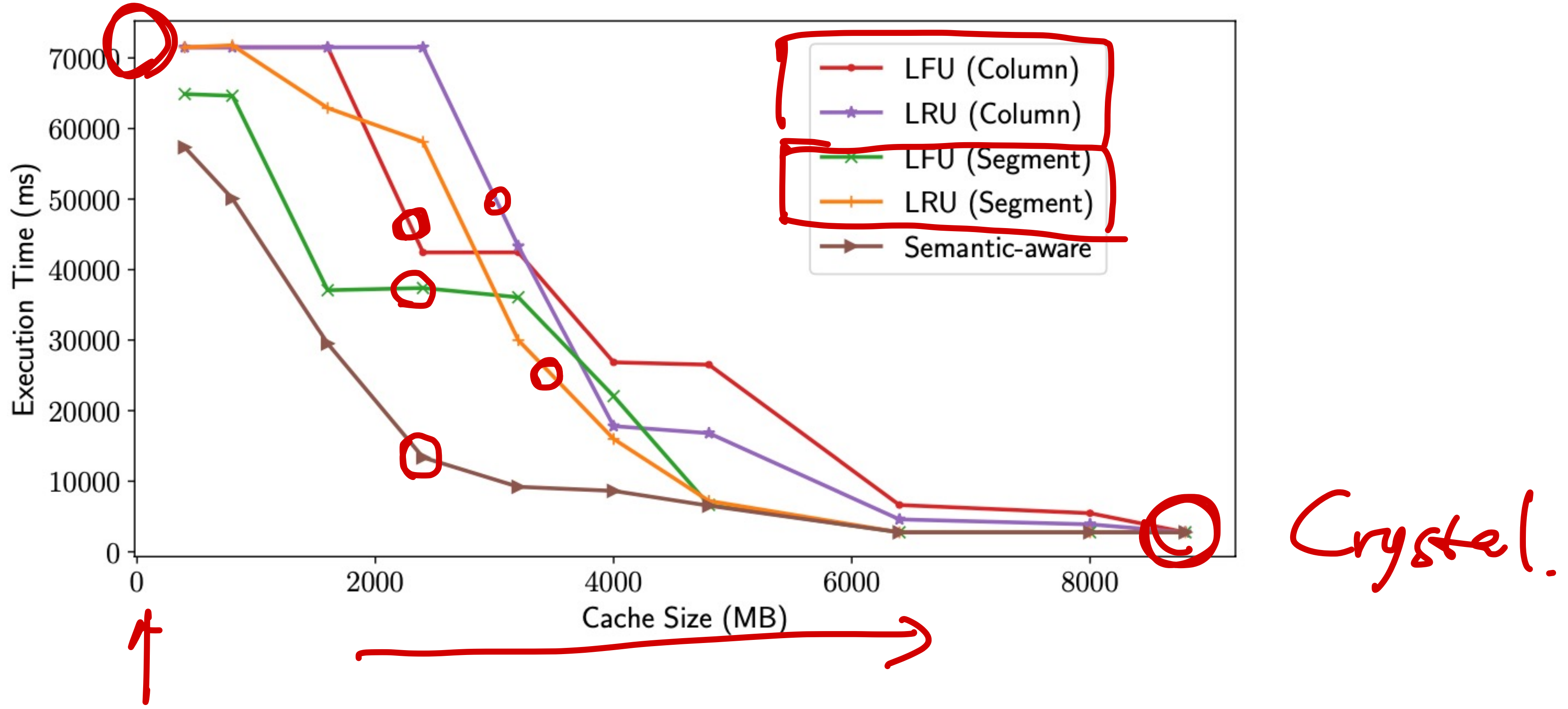
Mordred Architecture



Three components:

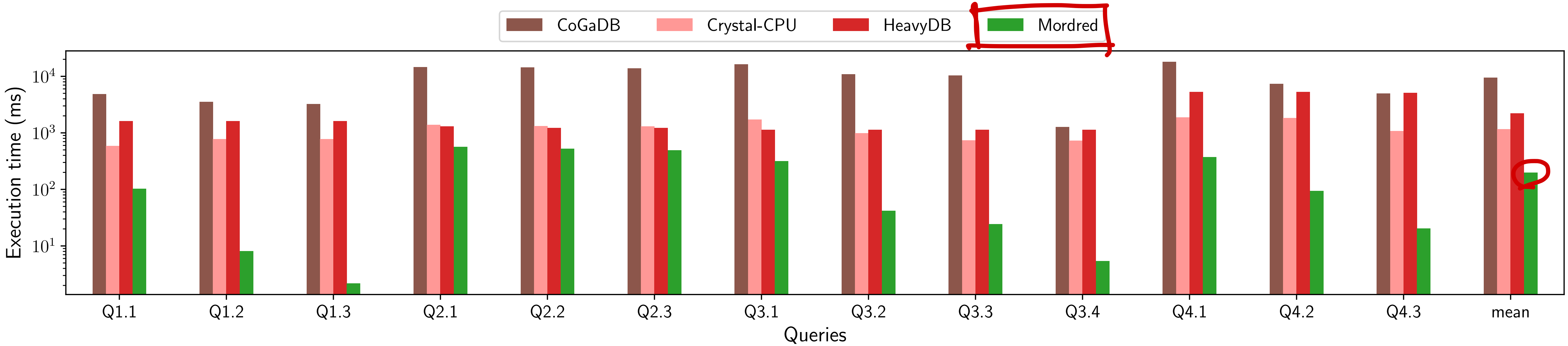
- Cache Manager
- Query Optimizer
- Query Execution Engine

Evaluation — Caching Policies



Semantic-aware fine-grained caching achieves the best performance

Evaluation — SSB Runtime

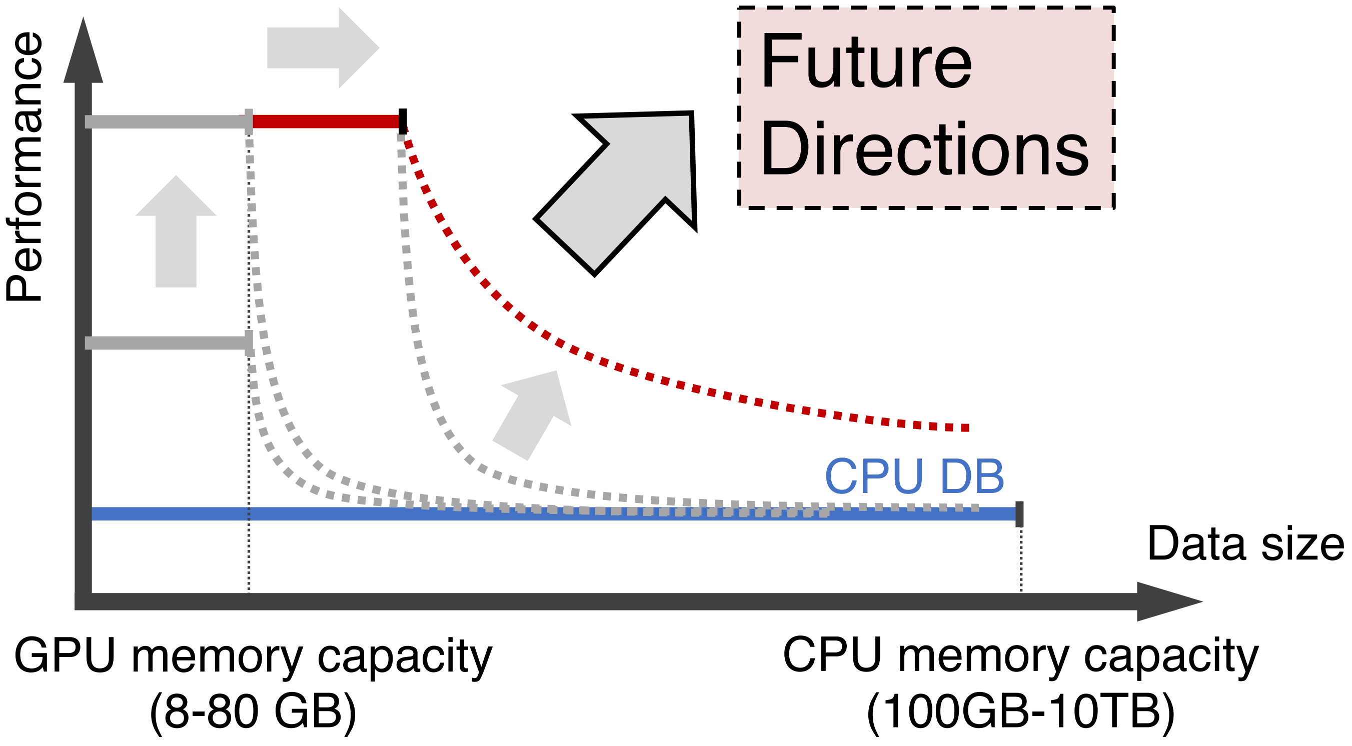


Query runtime on SSB (SF = 160)

Mordred (our design) is 6× faster than the best-previous design

- 8GB cache size, ~32GB data size

Conclusion



GPU database has great performance potential

Key challenge: large data sets do not fit in GPU memory

– Project 1: **Saturate GPU memory** [1]
– Project 2: **Data compression** [2]
– Project 3: **Hybrid CPU-GPU DB** [3]

[1] Anil Shanbhag, Samuel Madden, Xiangyao Yu, *A Study of the Fundamental Performance Characteristics of GPUs and CPUs for Database Analytics*, **SIGMOD 2020**
[2] Anil Shanbhag*, Bobbi Yogatama*, Xiangyao Yu, Samuel Madden, *Tile-based Lightweight Integer Compression in GPU*, **SIGMOD 2022**
[3] Bobbi Yogatama, Weiwei Gong, Xiangyao Yu, *Orchestrating Data Placement and Query Execution in Heterogeneous CPU-GPU DBMS*, **VLDB 2022**

GPU Databases – Q/A

Special hardware designed for SQL analytics?

Transfer between CPU and GPU becomes a bottleneck?

How popular are GPUs used in industrial databases? What are the main barriers?

Next Lecture

DAWN workshop

– Reserve a presentation slot using the following google sheet

https://docs.google.com/spreadsheets/d/1Re1M9FmJwI_YkidhNgeV0iKn-clssFrK_J1PMidaAuw/edit?usp=sharing

– 8-min per group (presentation + QA)

Project report (DDL: Dec. 19)

– **Submit to the hotcrp website** (like the proposal)

Submit course evaluation on aefis.wisc.edu