Scalable Data Management using GPUs with Fast Interconnects

Clemens Lutz

PhD Defense

10. November 2022

GPUs

Compute:	3×
Memory bandwidth:	10×

GPUs

Compute: 3×
Memory bandwidth: 10×

TensorFlow

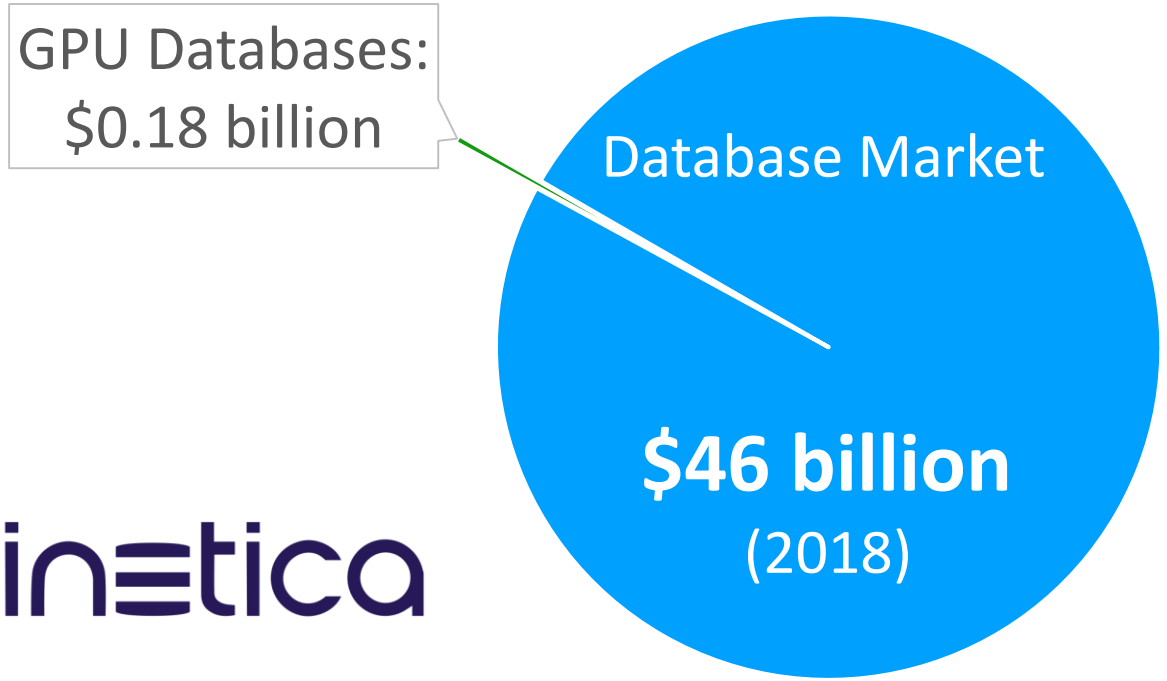
GPUs

Compute: 3×
Memory bandwidth: 10×

?

TensorFlow

Data Management using GPUs



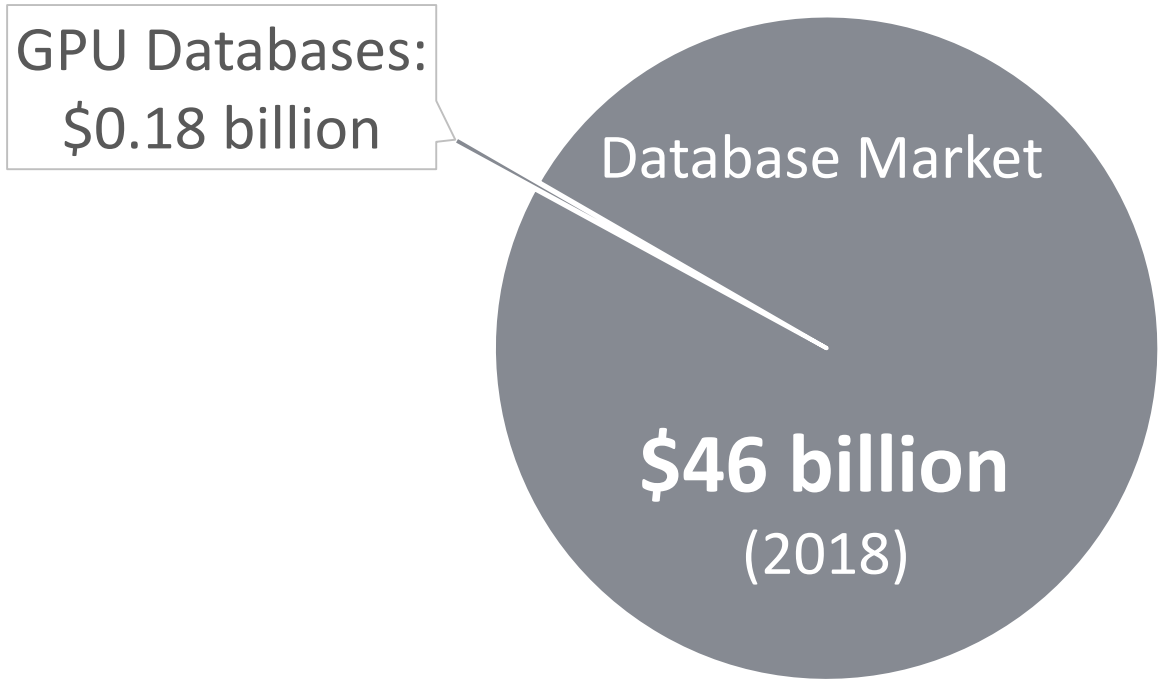
kinetica

blazingSQL

SQREAM

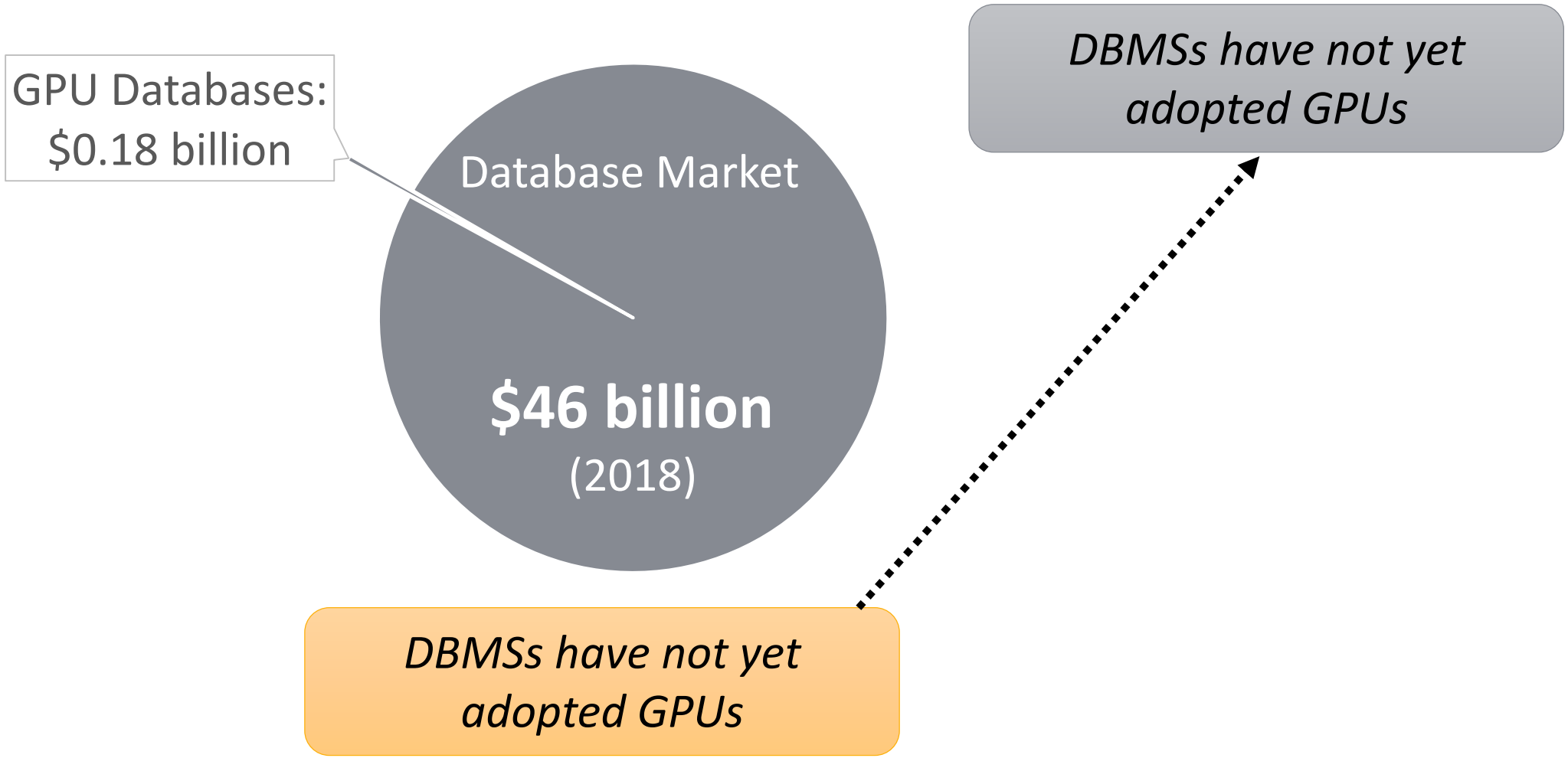
omni·sci

Data Management using GPUs

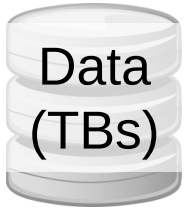


DBMSs have not yet adopted GPUs

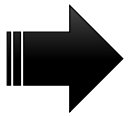
Data Management using GPUs

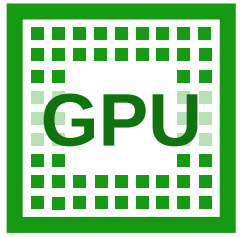


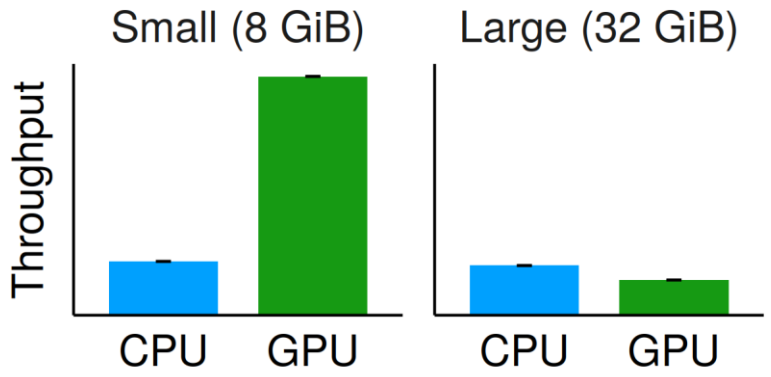
Why DBMSs haven't adopted GPUs?



Data-intensive queries

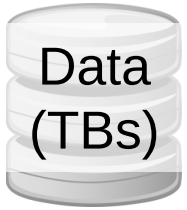




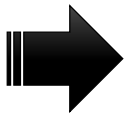


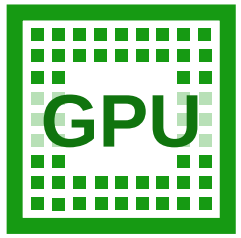
DBMSs have not adopted GPUs

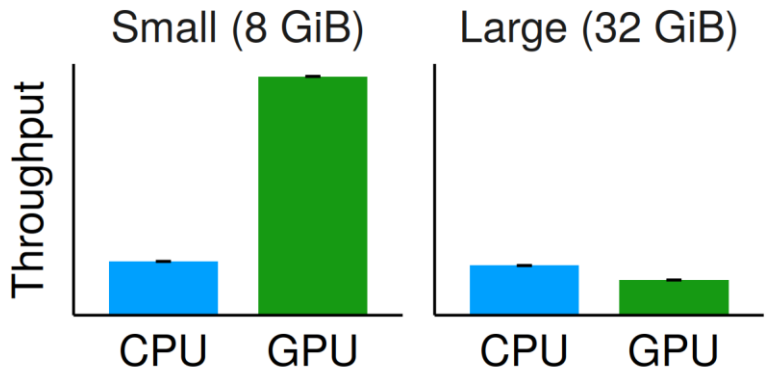
Why DBMSs haven't adopted GPUs?



Data-intensive queries



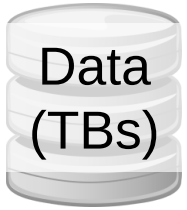




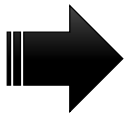
DBMSs have not adopted GPUs

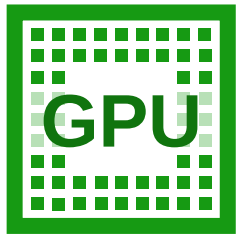
Because GPU-enabled data management doesn't scale

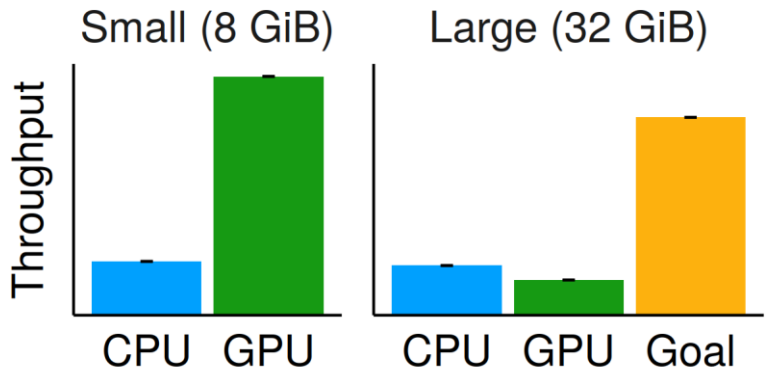
Why DBMSs haven't adopted GPUs?



Data-intensive queries







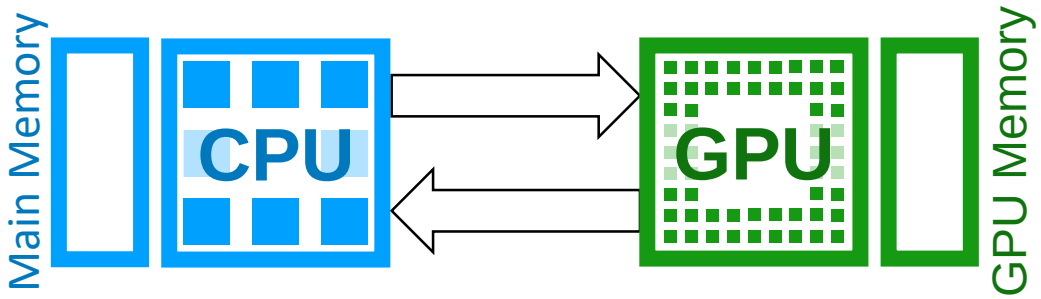
DBMSs have not adopted GPUs

Because GPU-enabled data management doesn't scale

Thesis Goal

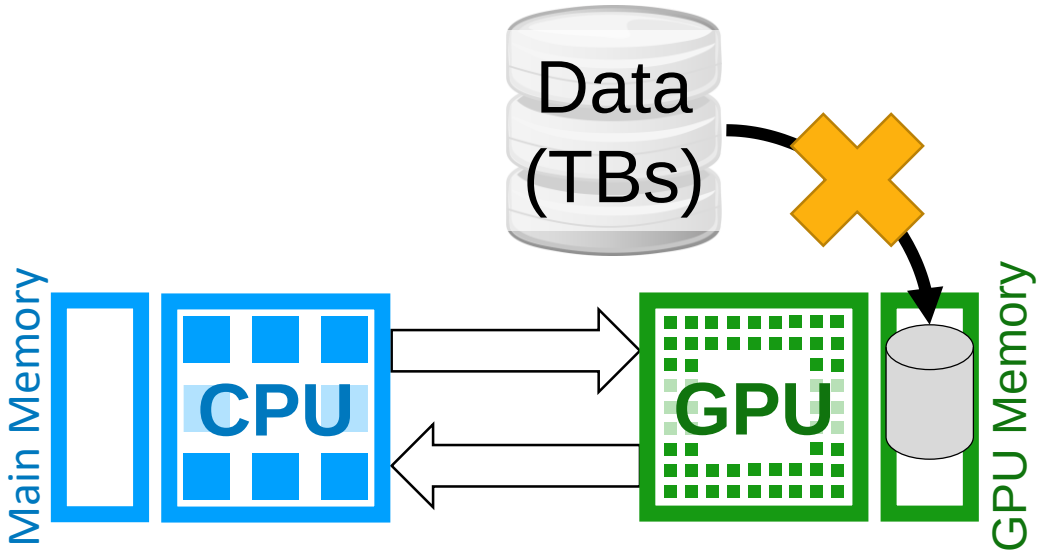
Scale data management using GPUs to large data volumes

Why it doesn’t scale



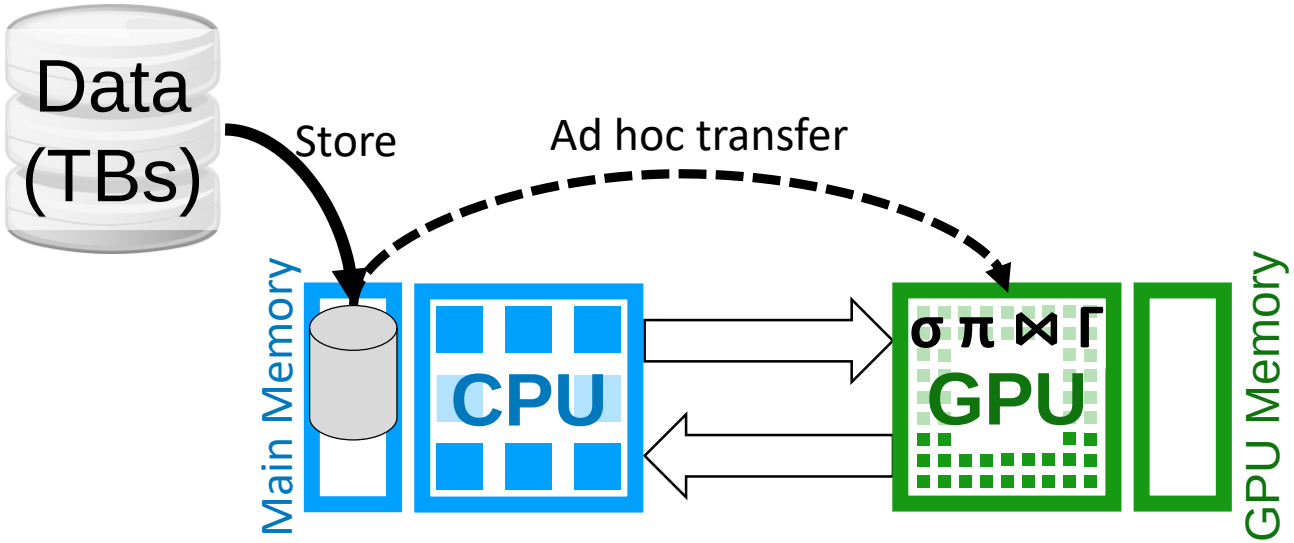
- Memory capacity
- Interconnect bandwidth

Why it doesn’t scale



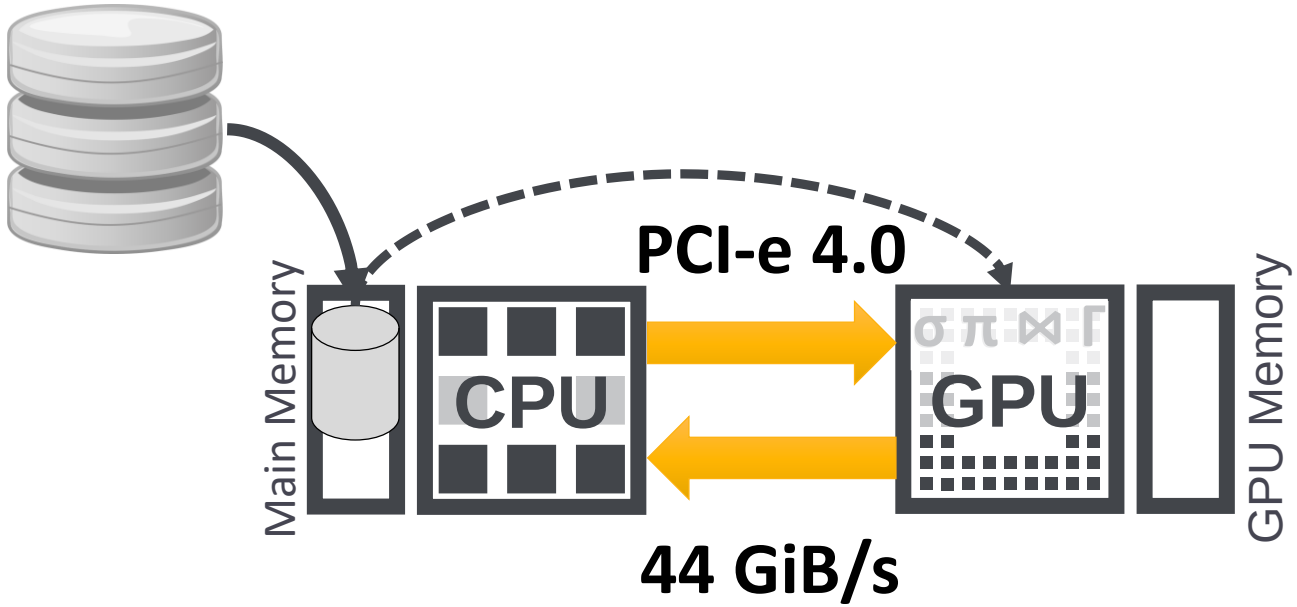
- Memory capacity
- Interconnect bandwidth

Why it doesn't scale



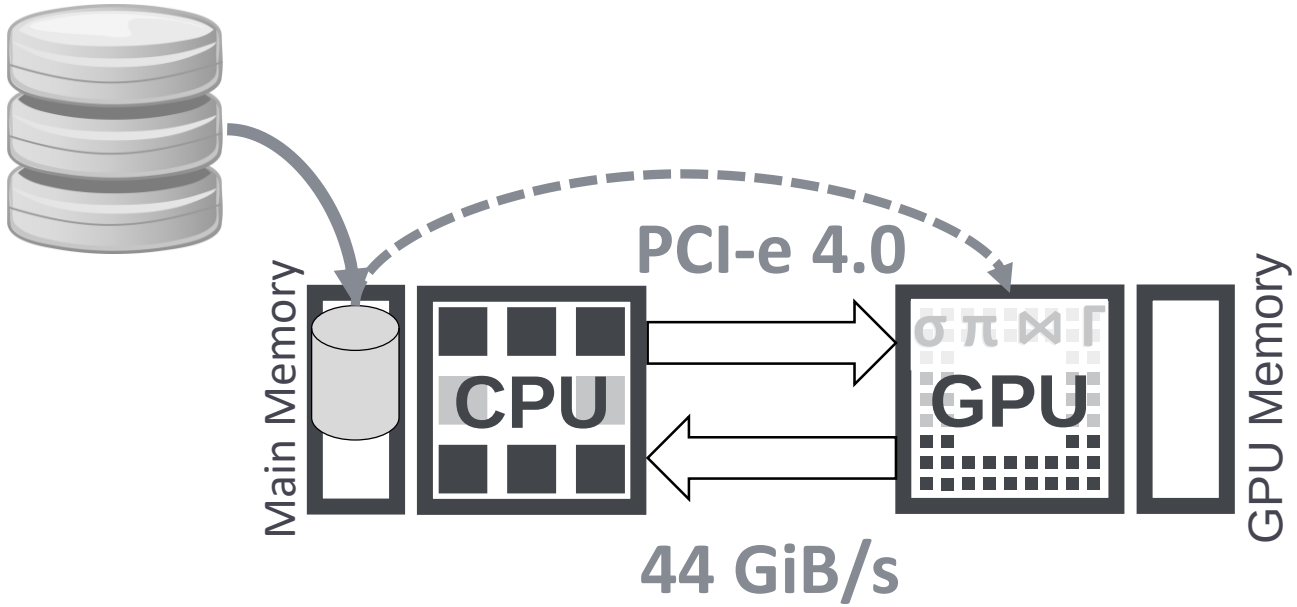
- Memory capacity
- Interconnect bandwidth

Why it doesn't scale



- Memory capacity
- Interconnect bandwidth

Why it doesn't scale

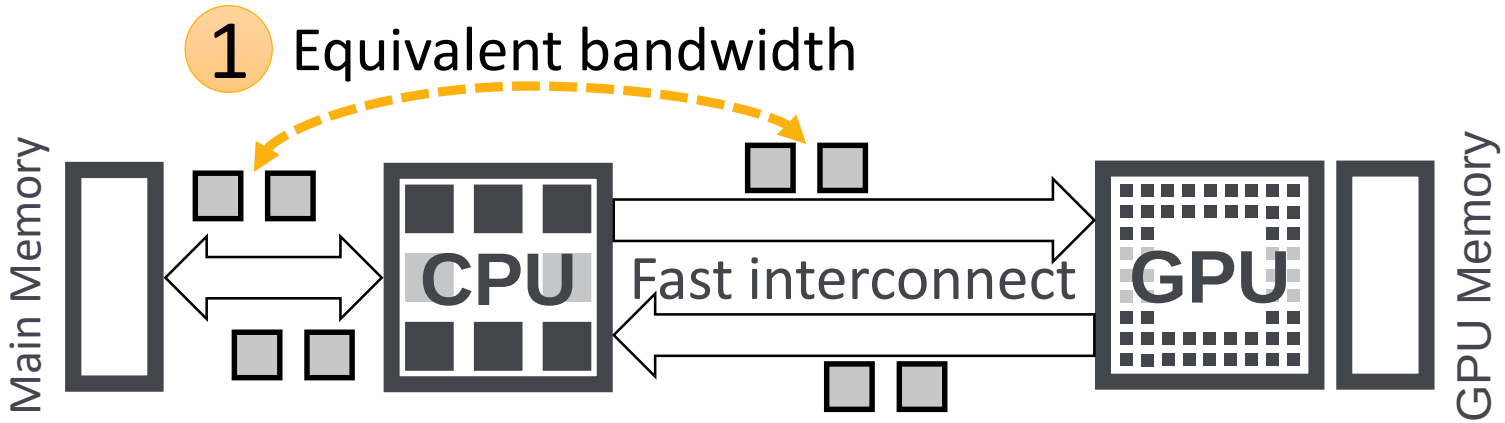


- Memory capacity
- Interconnect bandwidth

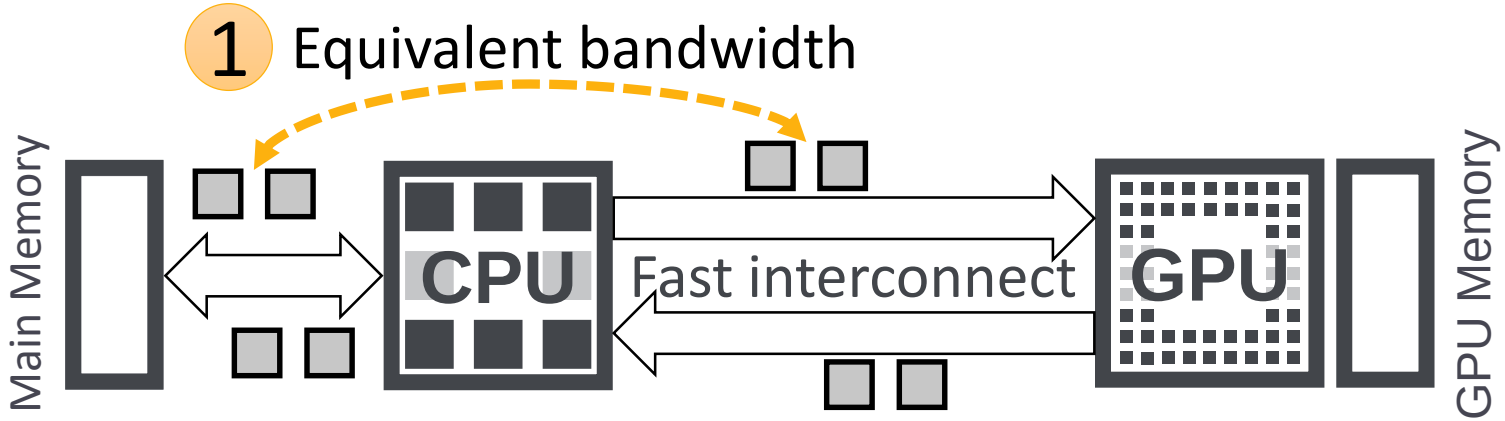
Data transfer bottleneck

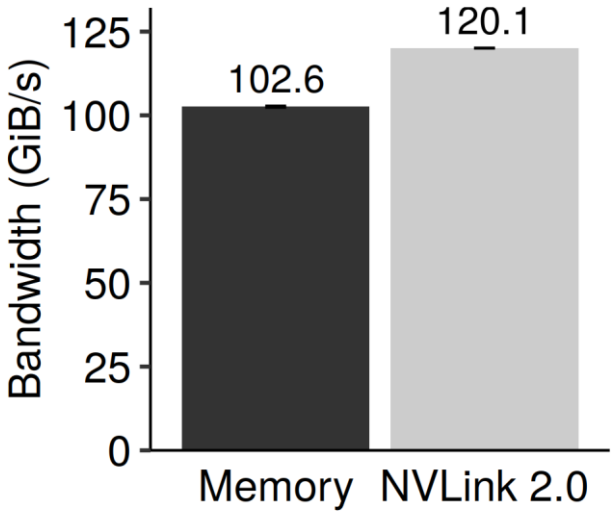
Game Changer: Fast Interconnects

Game Changer: Fast Interconnects

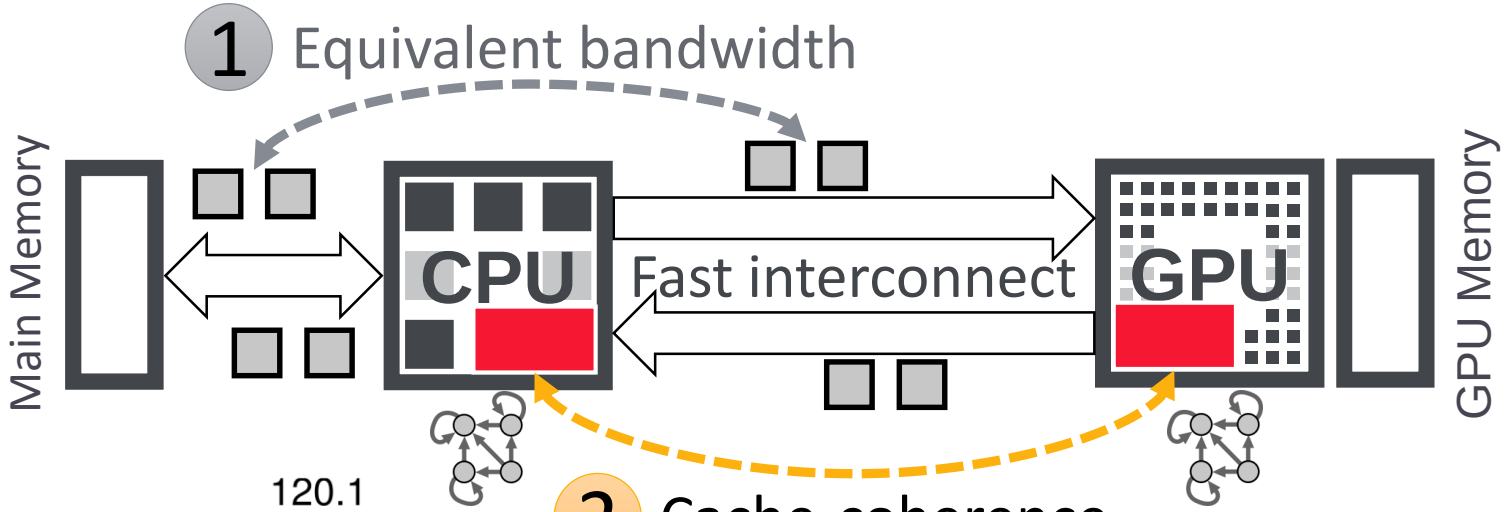


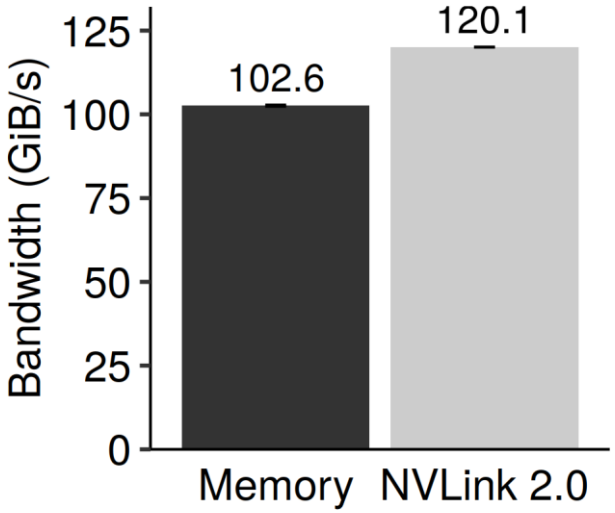
Game Changer: Fast Interconnects





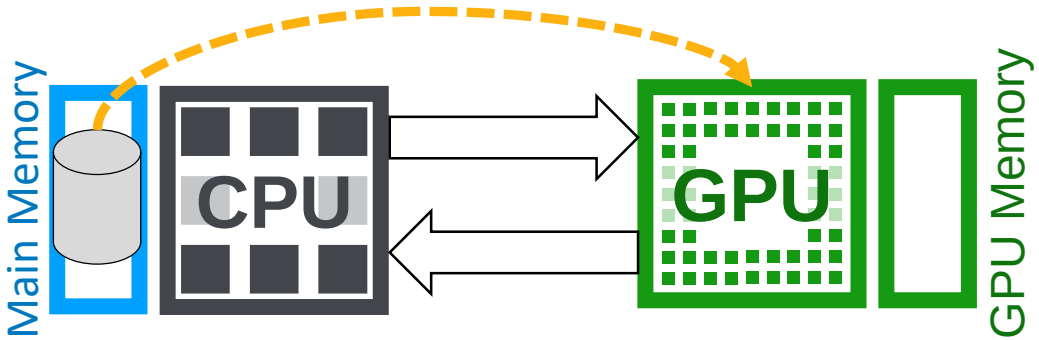
Game Changer: Fast Interconnects



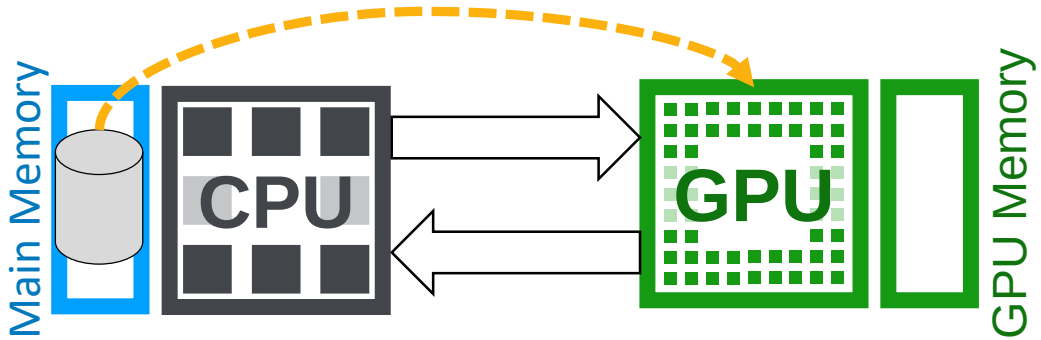


- Unified address space
- Data-dependent memory accesses

Why a Fast Interconnect is not Sufficient



Why a Fast Interconnect is not Sufficient

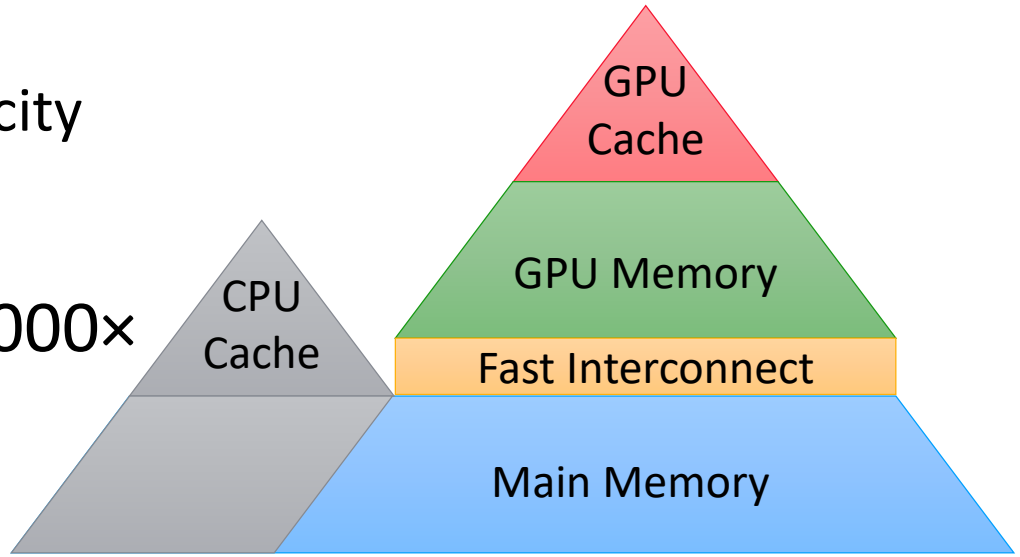


Overview

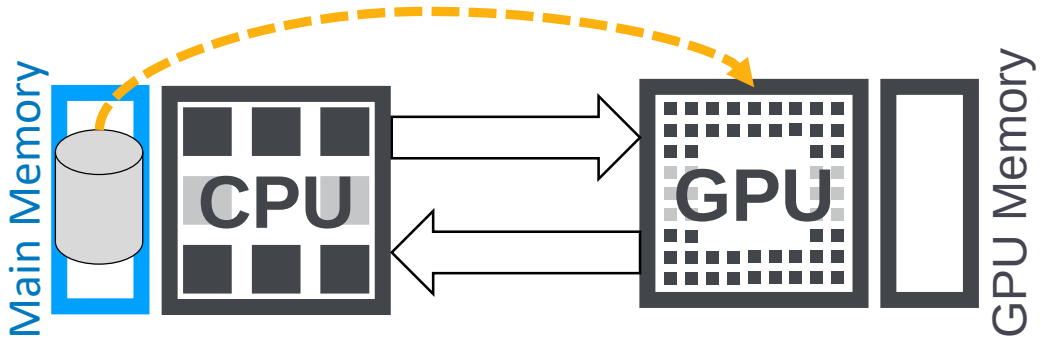
Memory Hierarchy

Bandwidth Capacity

1000× 1000×



Why a Fast Interconnect is not Sufficient

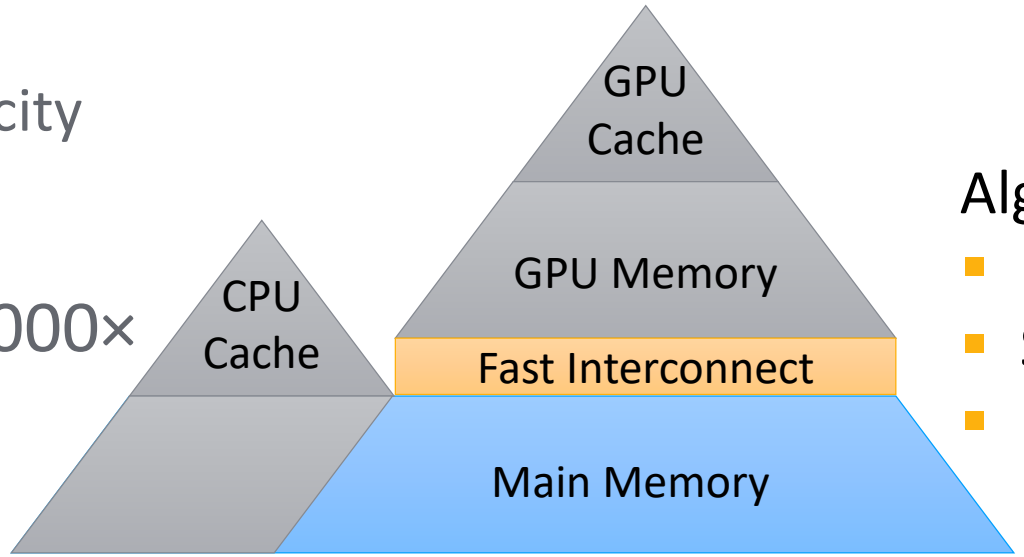


Overview

Memory Hierarchy

Bandwidth Capacity

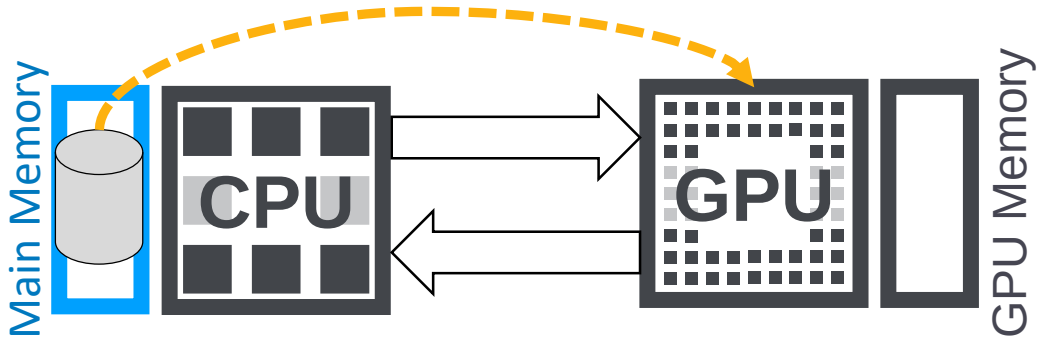
1000× 1000×



Algorithm design:

- Data access
- State access
- Data locality

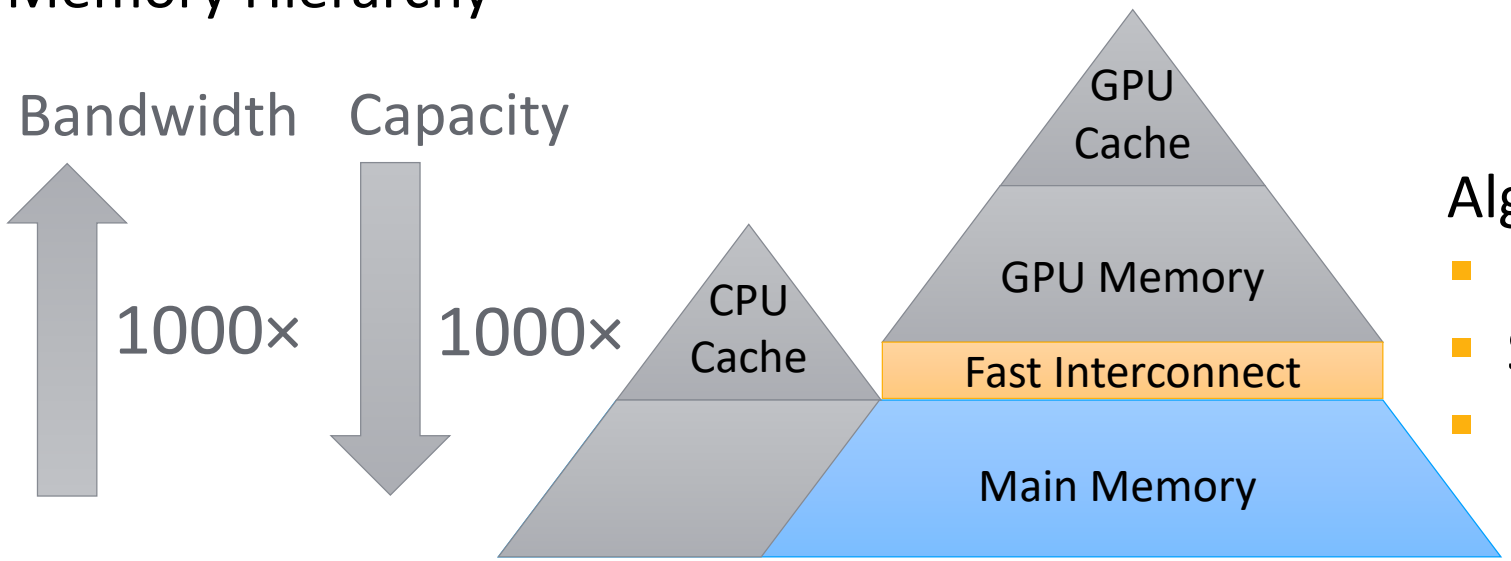
Why a Fast Interconnect is not Sufficient



Interconnect-conscious algorithm design

Overview

Memory Hierarchy

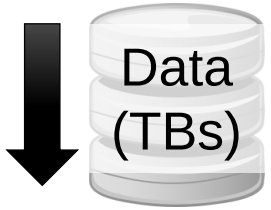


Algorithm design:

- Data access
- State access
- Data locality

Contributions

Data Management Problem



Data-intensive query processing

>

Our Solution

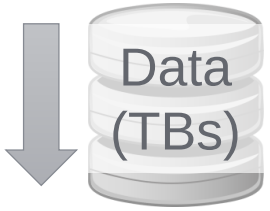
Pump Up the Volume

SIGMOD 2020

Interconnect-conscious algorithm design

Contributions

Data Management Problem

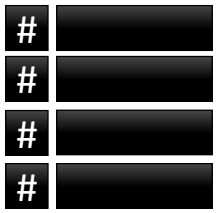


Data-intensive query processing

Our Solution

> Pump Up the Volume

SIGMOD 2020



Stateful data processing

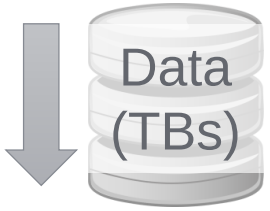
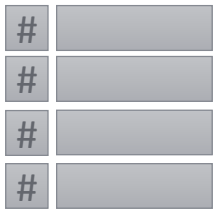
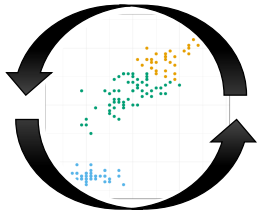
> Triton join

SIGMOD 2022

Interconnect-conscious algorithm design

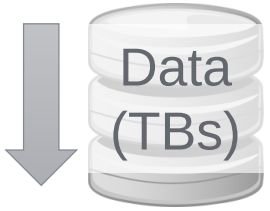
Contributions

Interconnect-conscious algorithm design

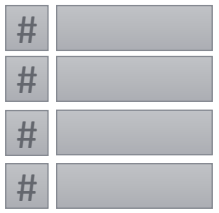
Data Management Problem		Our Solution
 Data-intensive query processing	>	Pump Up the Volume SIGMOD 2020
 Stateful data processing	>	Triton join SIGMOD 2022
 Iterative algorithms	>	Single-pass *k*-means strategy DaMoN 2018 DB Spektrum 2018

Contributions

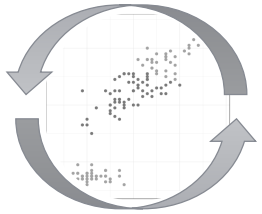
Data Management Problem



Data-intensive query processing



Stateful data processing



Iterative algorithms

Our Solution

> Pump Up the Volume — SIGMOD 2020

> Triton join — SIGMOD 2022

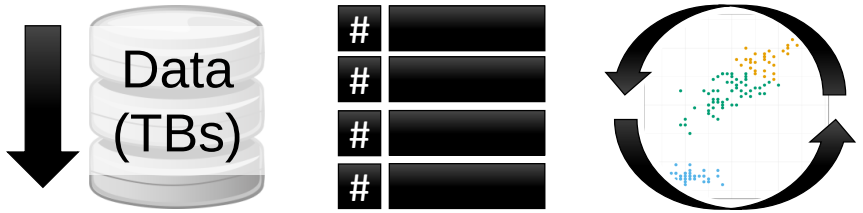
> Single-pass *k*-means strategy — DaMoN 2018; DB Spektrum 2018

Interconnect-conscious algorithm design

Solution: Efficient out-of-core algorithms

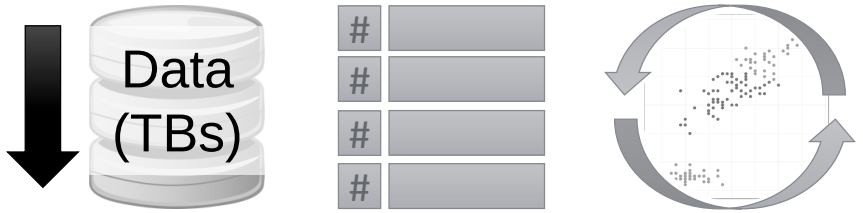
Agenda

1. Motivation
2. Data-intensive query processing
3. Stateful data processing
4. Iterative algorithms
5. Conclusion

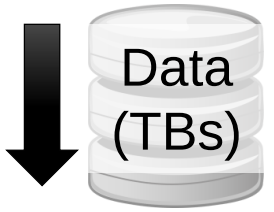


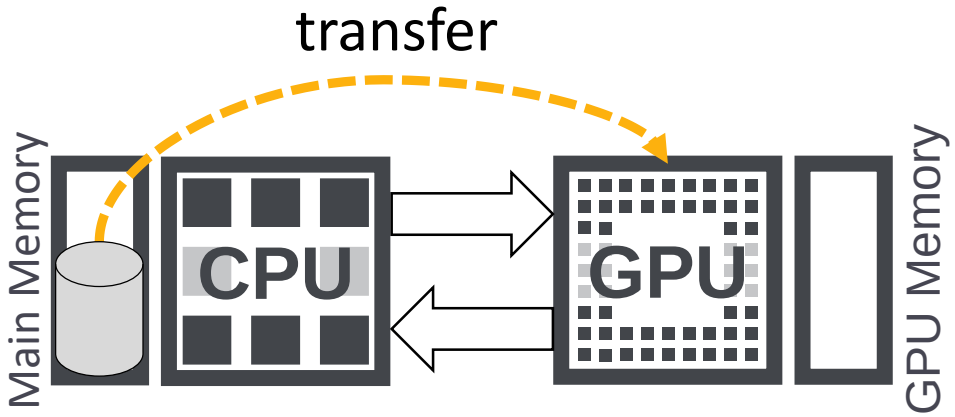
Agenda

1. Motivation
2. **Data-intensive query processing**
3. Stateful data processing
4. Iterative algorithms
5. Conclusion

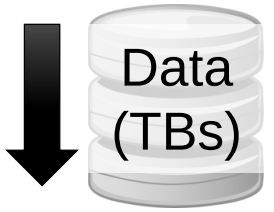


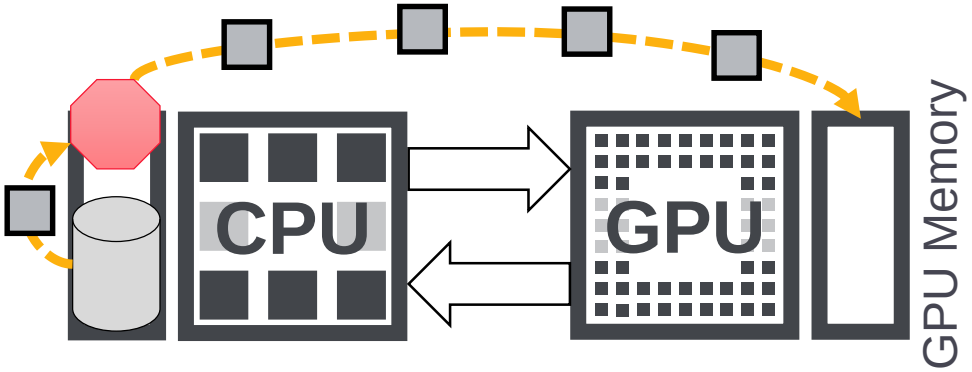
Efficient Data Access





Efficient Data Access

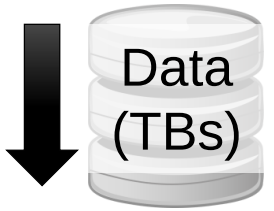


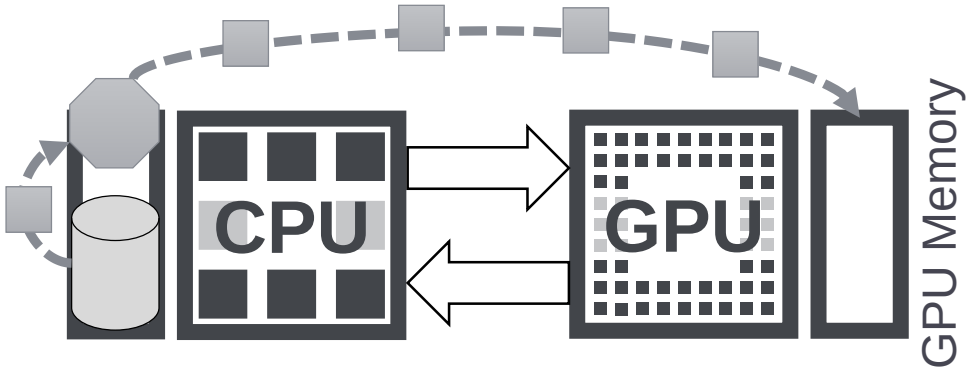


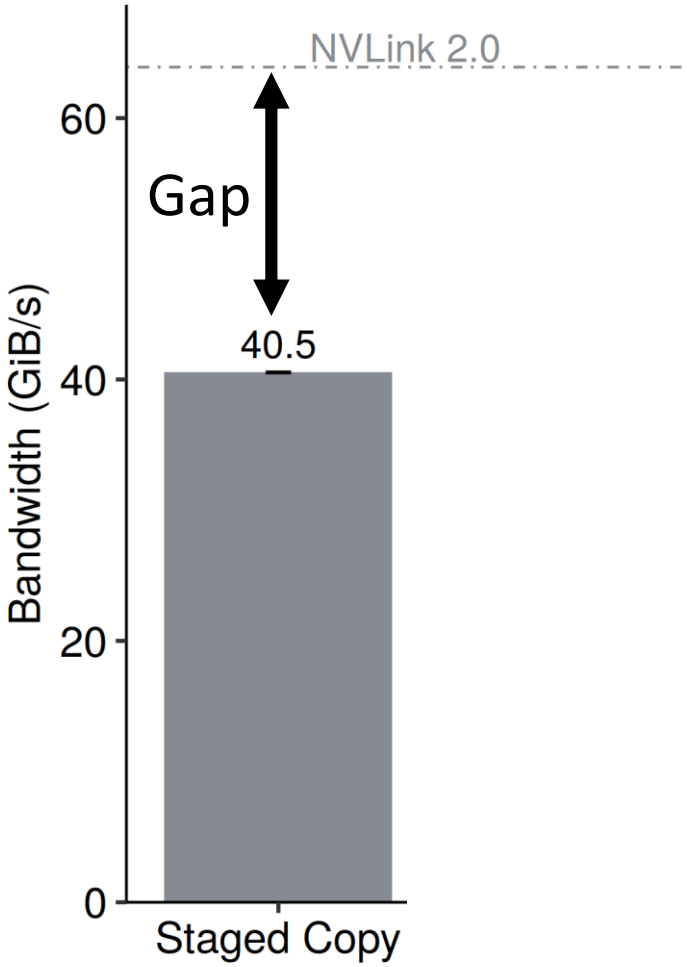
Transfer methods:

- Staged copy

Efficient Data Access



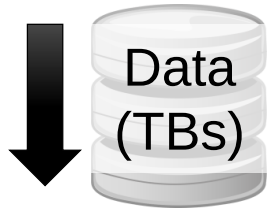




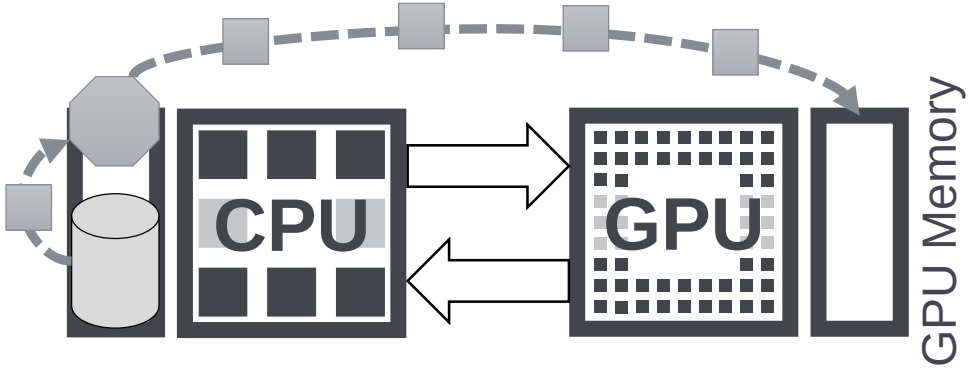
Transfer methods:

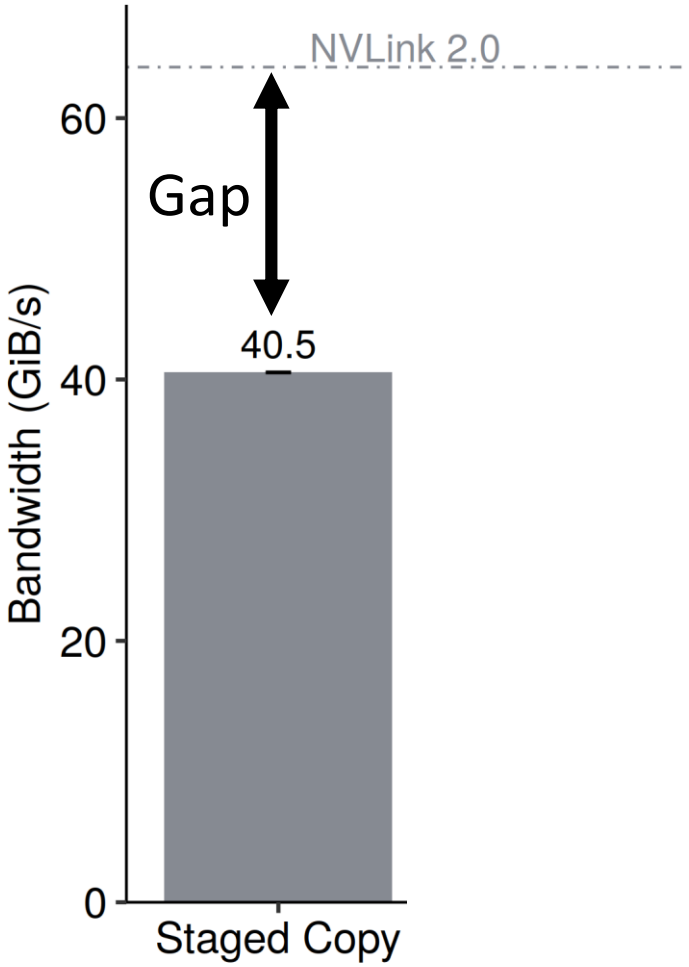
- Staged copy

Efficient Data Access



How can GPUs efficiently access data?

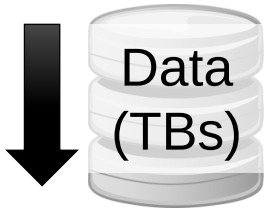


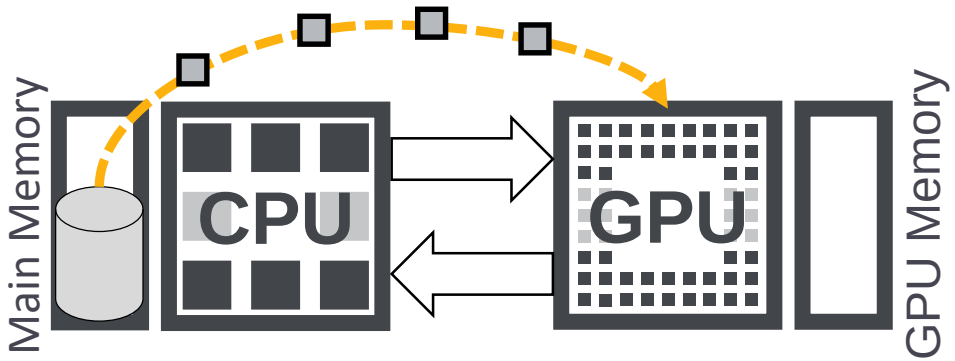


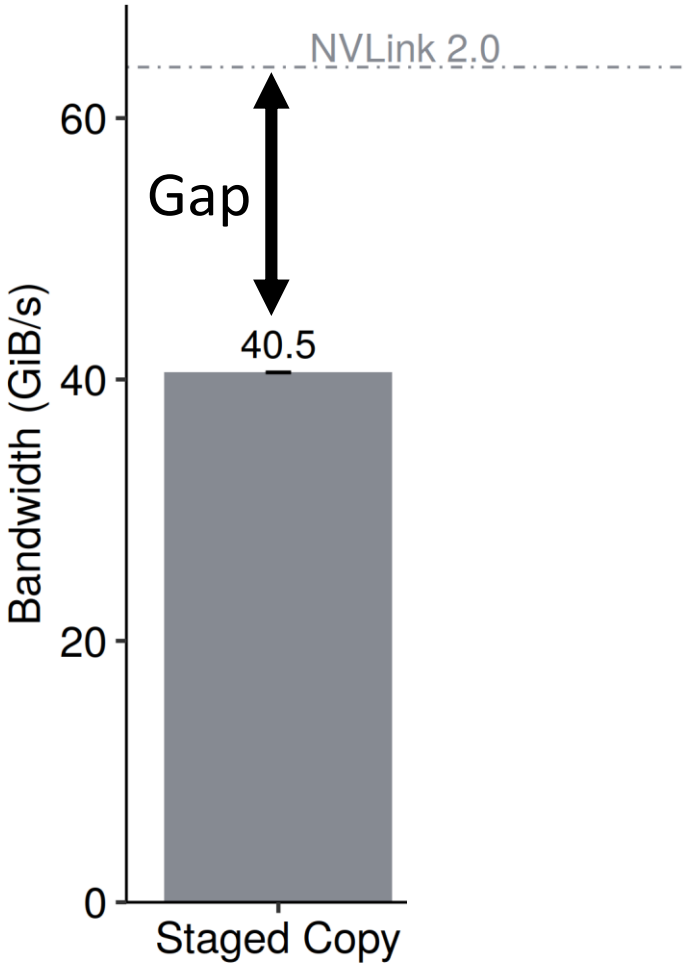
Transfer methods:

- Staged copy

Efficient Data Access







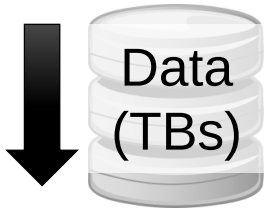
How can GPUs efficiently access data?

Insight: Exploit cache-coherence

Transfer methods:

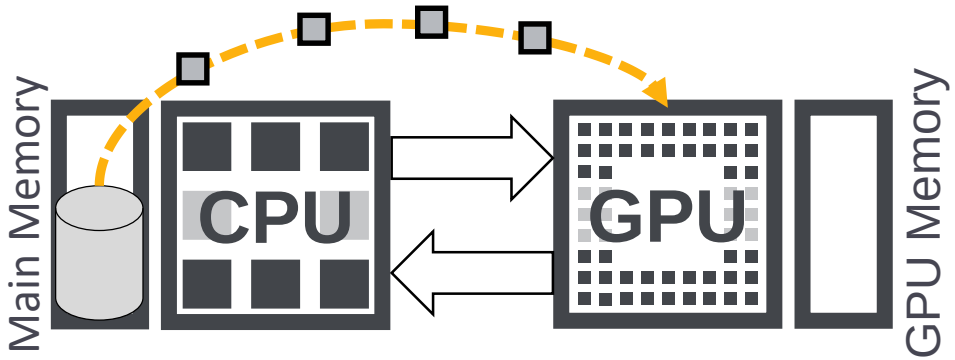
- Staged copy
- **Coherence**

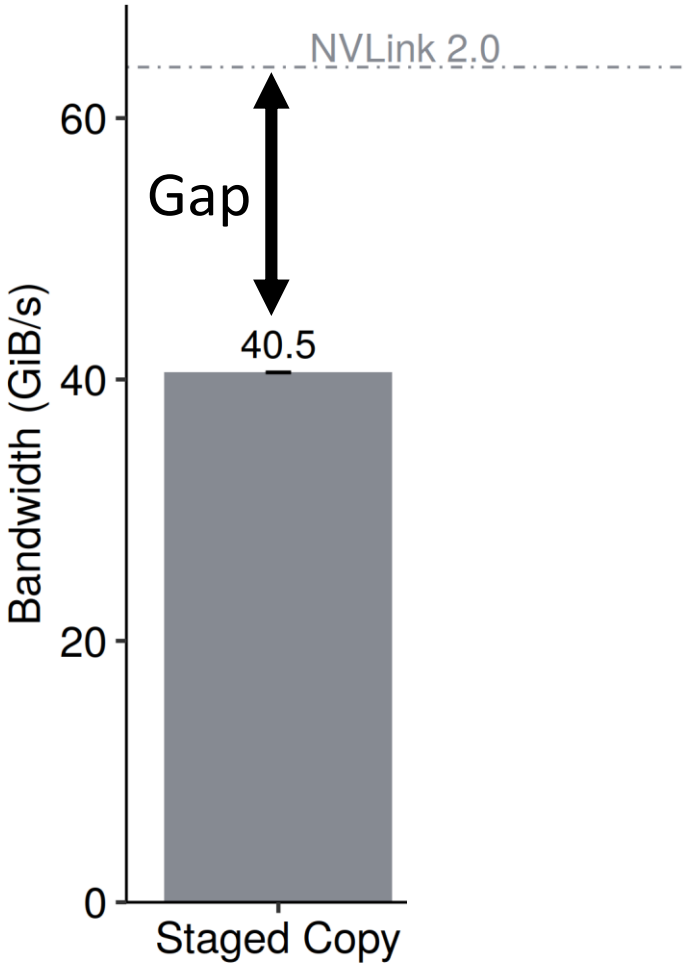
Efficient Data Access



Example:

```
sum += *x
```





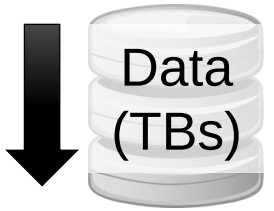
How can GPUs efficiently access data?

Insight: Exploit cache-coherence

Transfer methods:

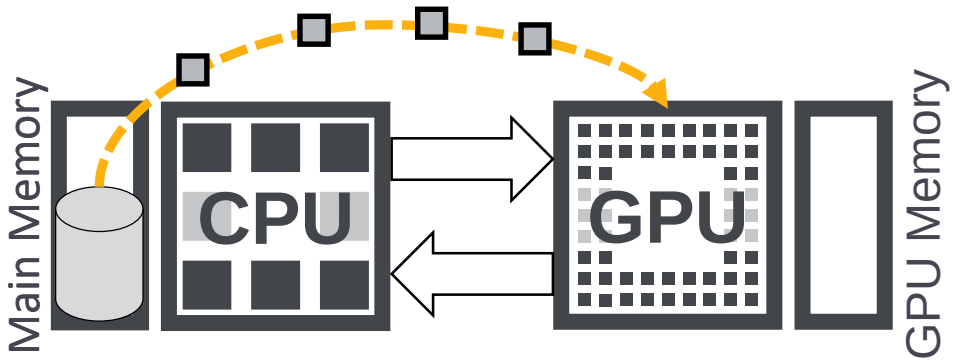
- Staged copy
- **Coherence**

Efficient Data Access



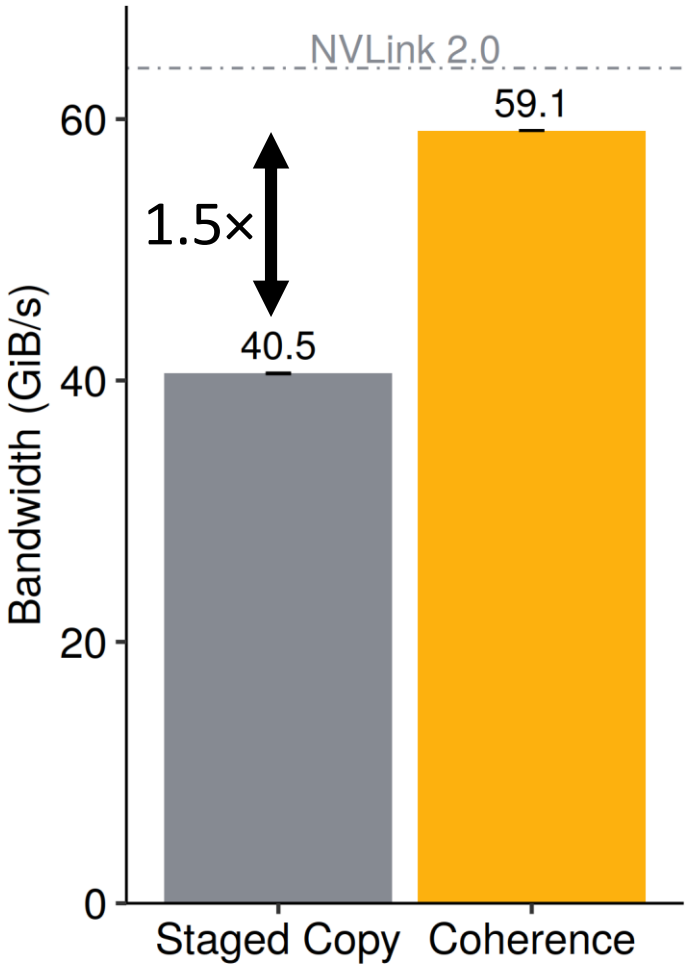
Example:

```
sum += *x
```



Transfer methods:

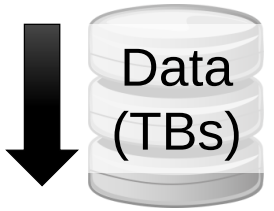
- Staged copy
- **Coherence**



How can GPUs efficiently access data?

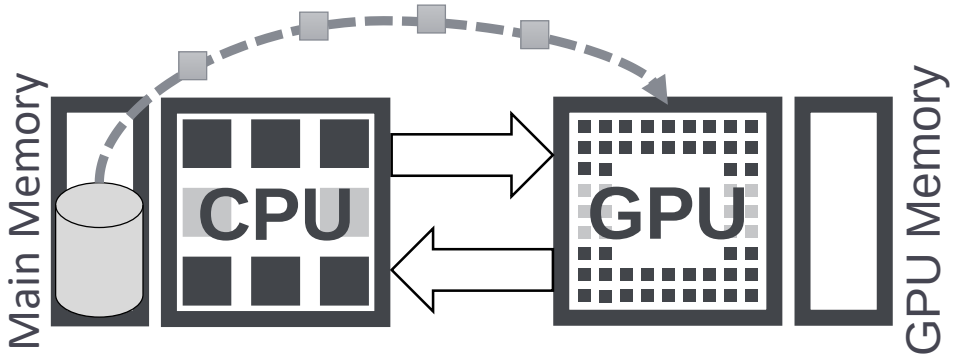
Insight: Exploit cache-coherence

Efficient Data Access



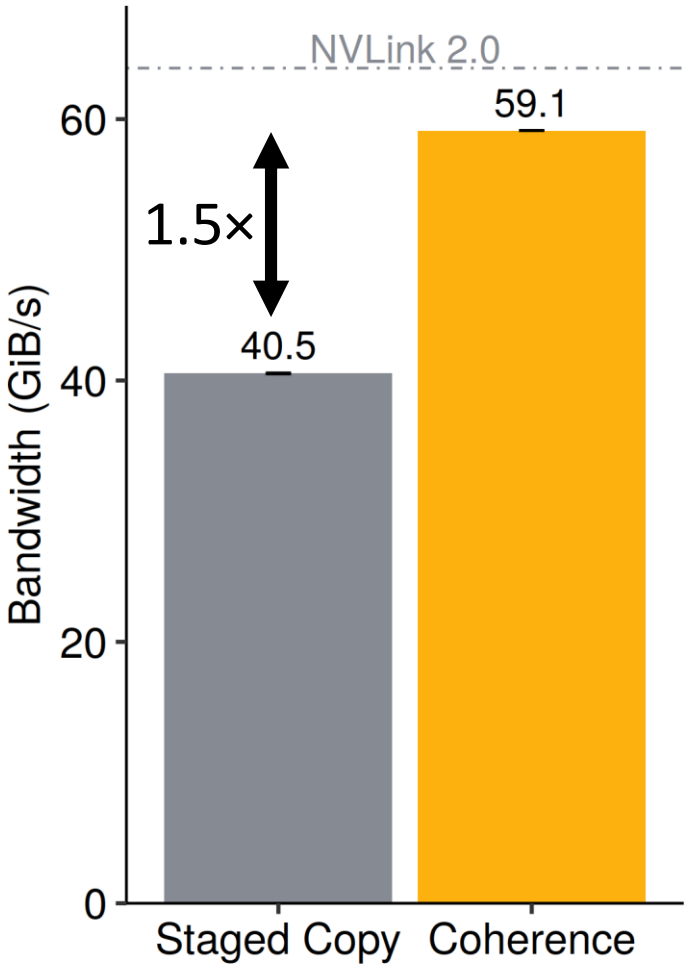
Example:

```
sum += *x
```



Transfer methods:

- Staged copy
- Coherence

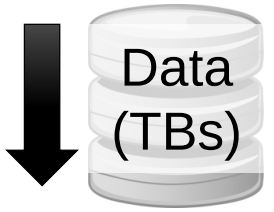


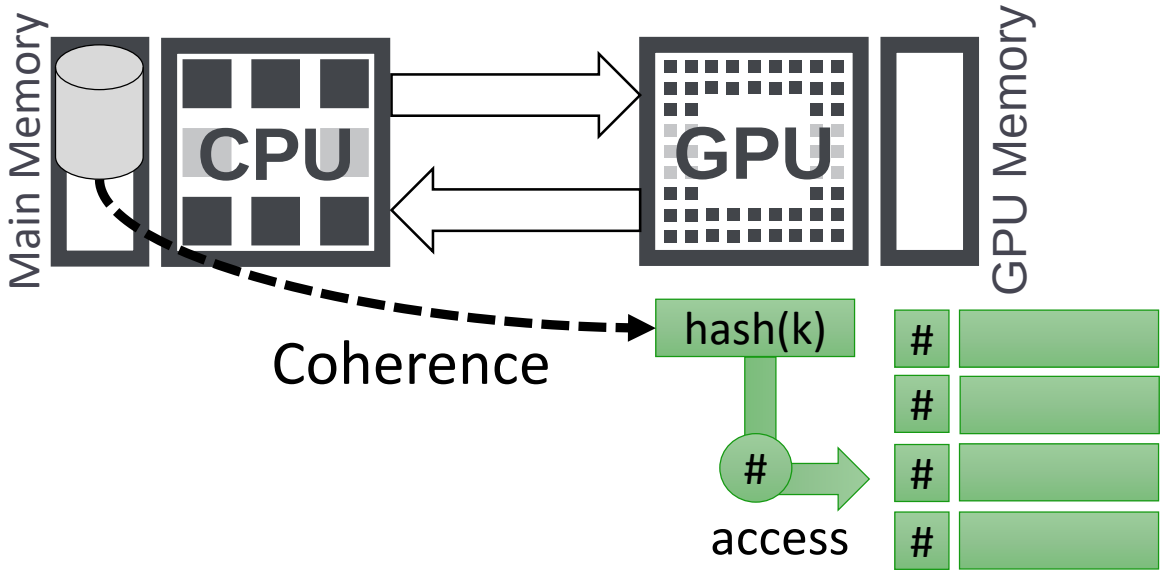
How can GPUs efficiently access data?

Insight: Exploit cache-coherence

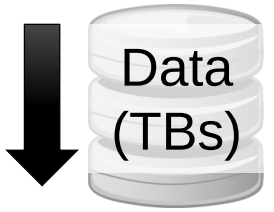
Interconnect-conscious data access

Hash Join: Scaling Outer Relation

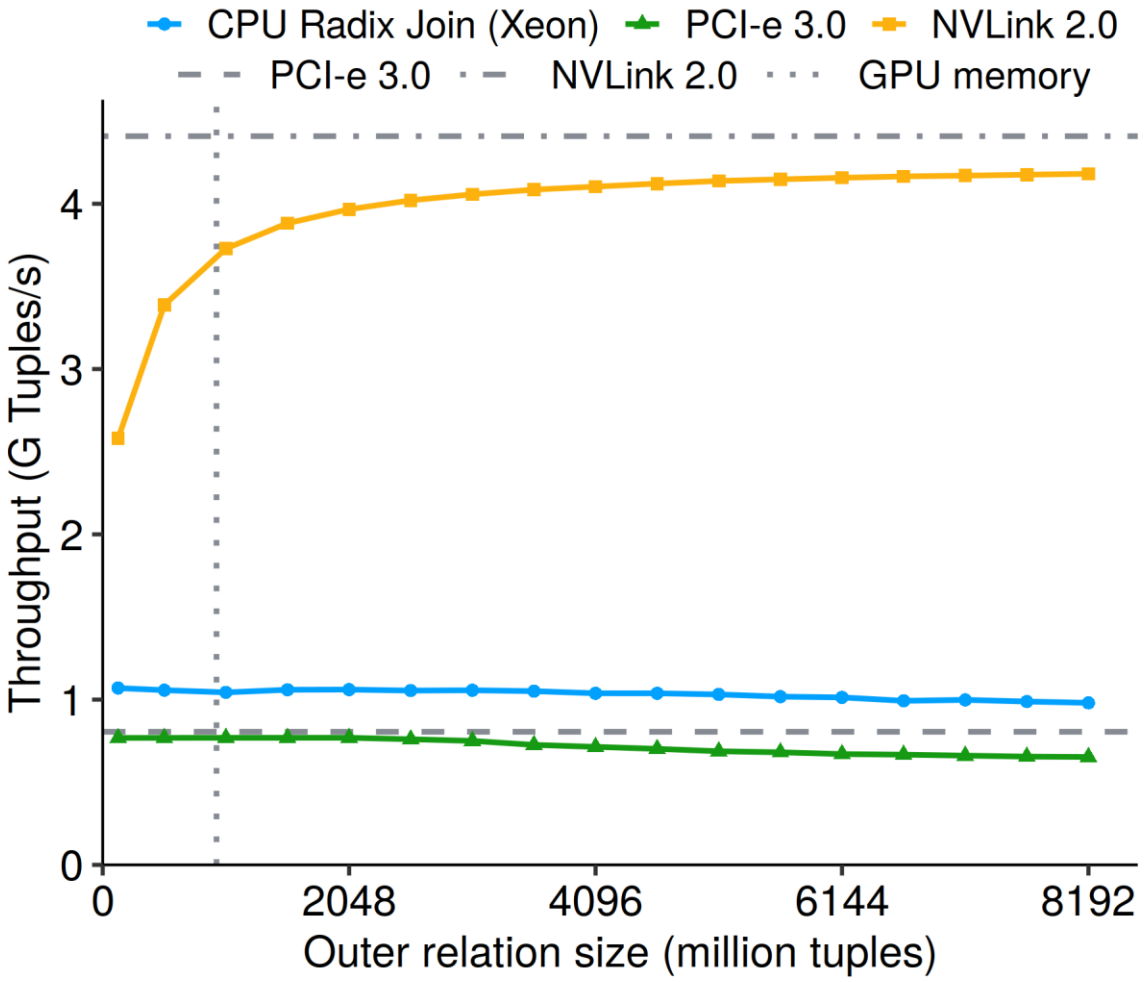




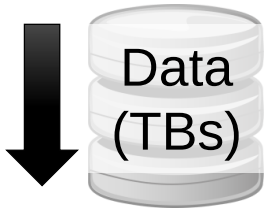
Hash Join: Scaling Outer Relation



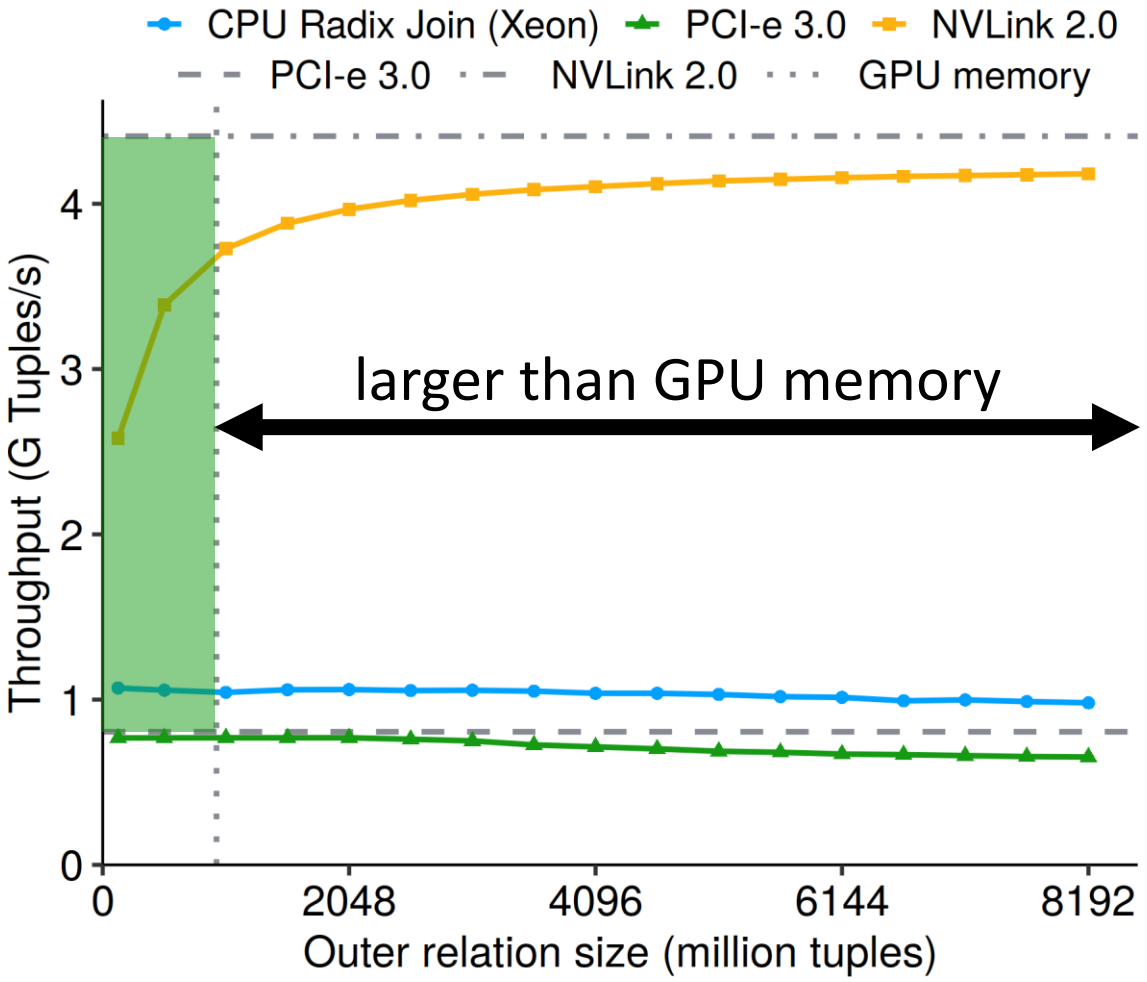
Data: 2 GiB ⋈ 122 GiB



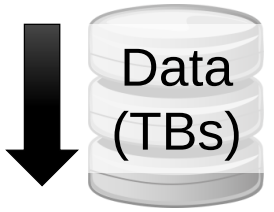
Hash Join: Scaling Outer Relation



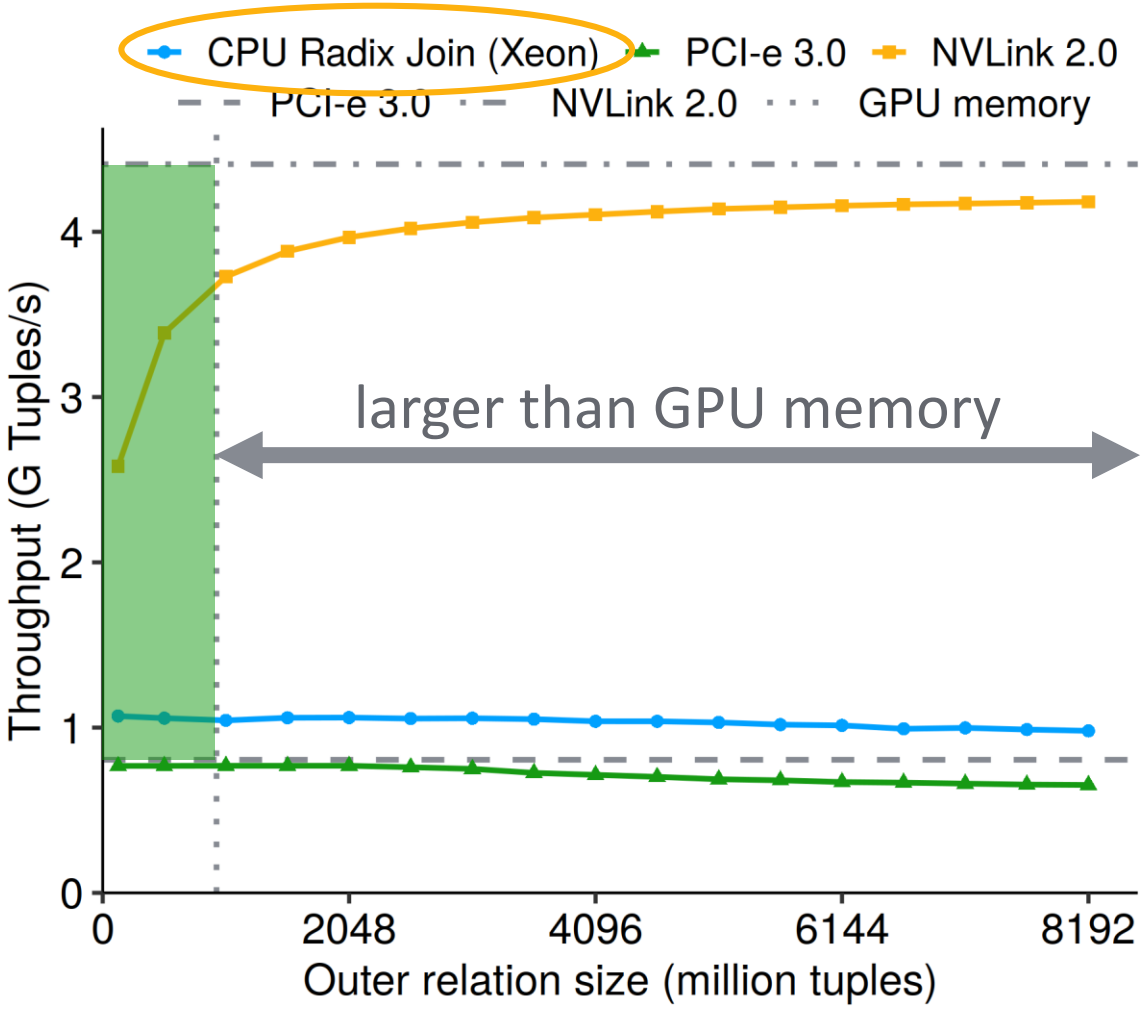
Data: 2 GiB ⋈ 122 GiB



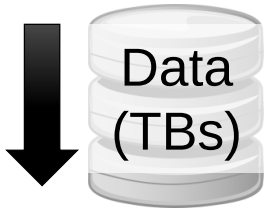
Hash Join: Scaling Outer Relation



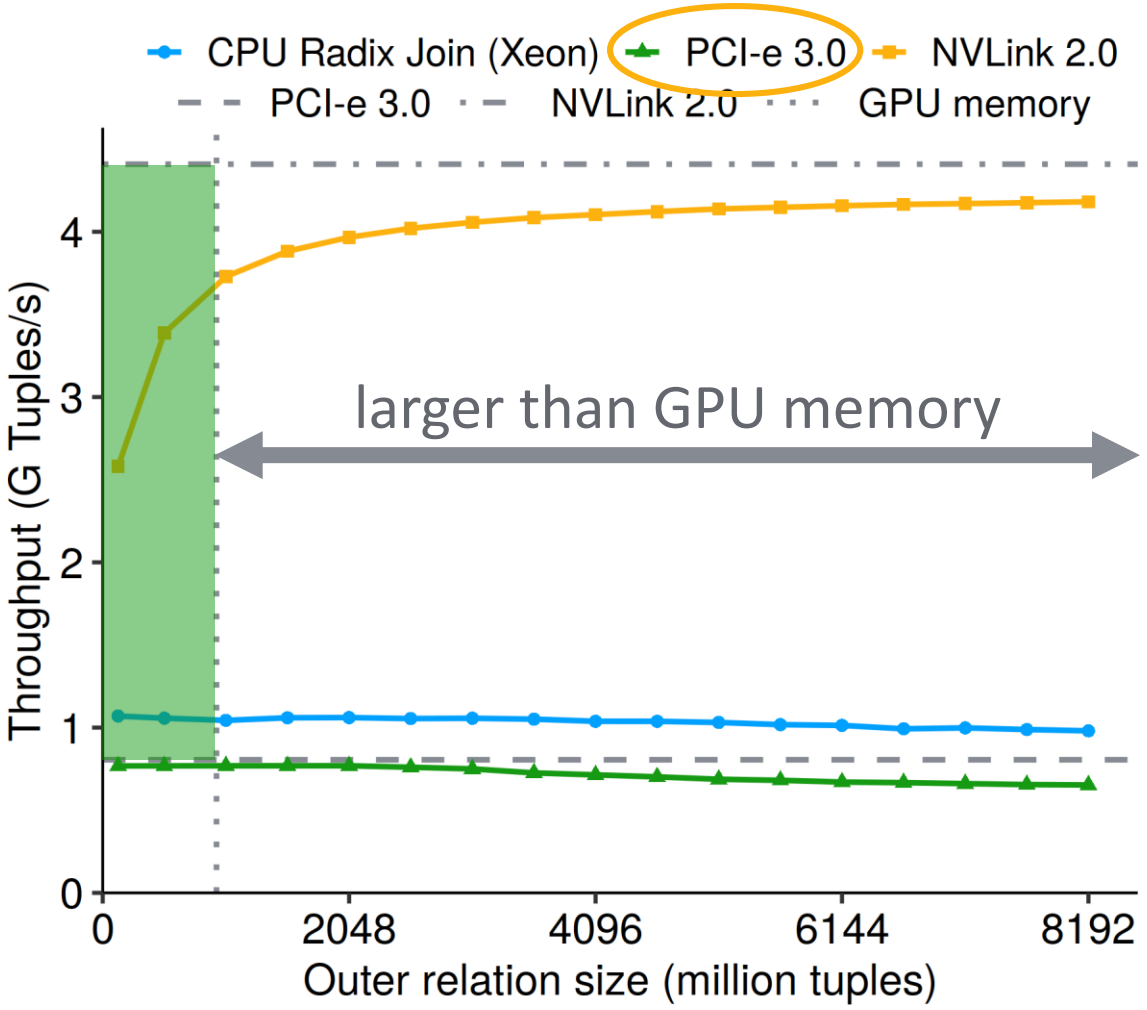
Data: 2 GiB ⋈ 122 GiB
CPU: Intel Xeon
with 12 cores



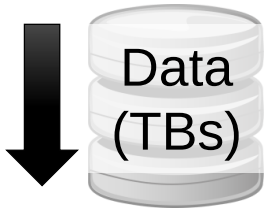
Hash Join: Scaling Outer Relation



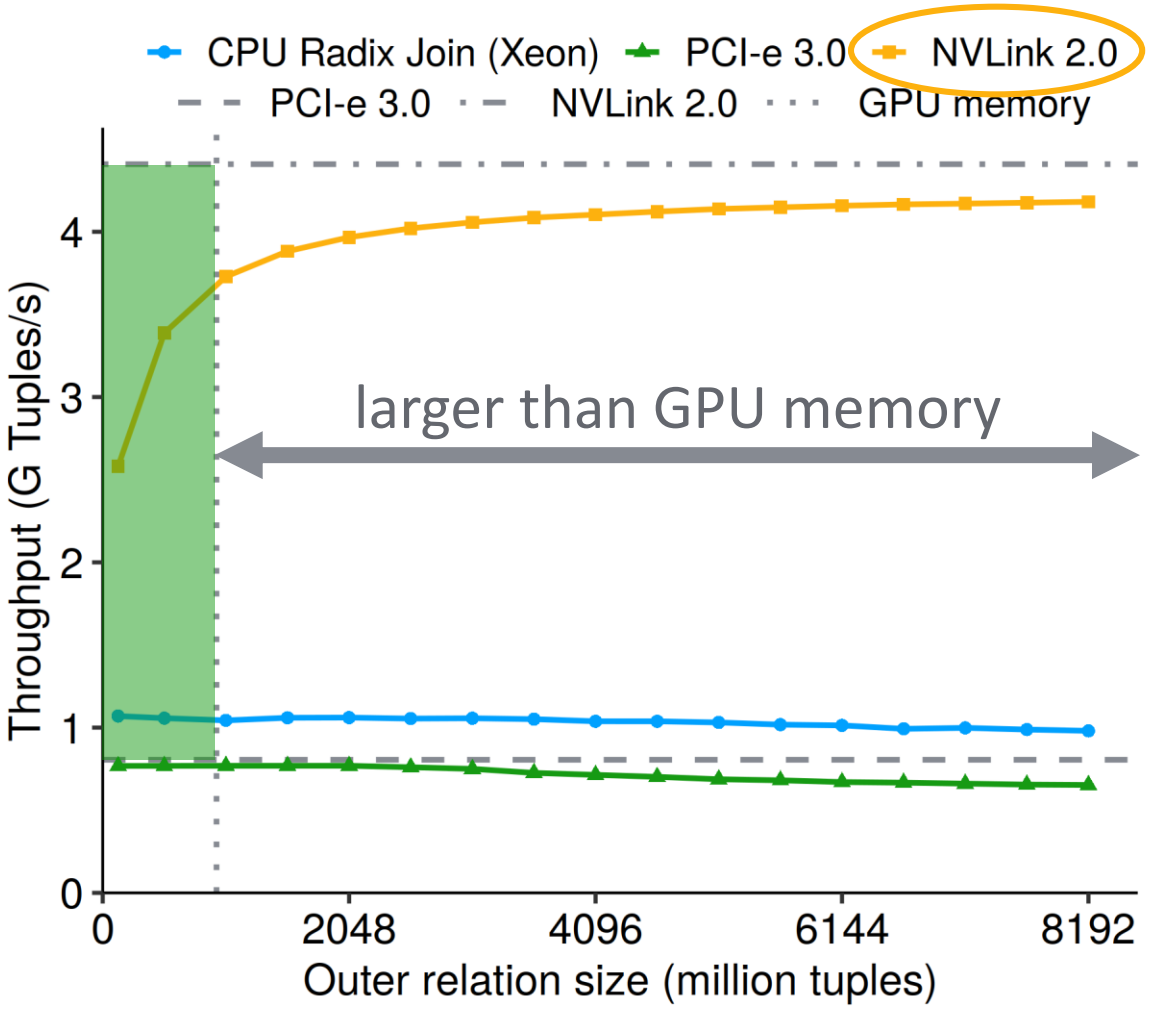
Data: 2 GiB ⋈ 122 GiB
CPU: Intel Xeon
with 12 cores
GPU: Nvidia V100



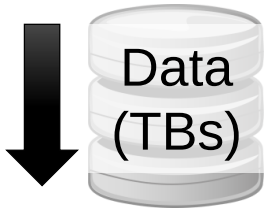
Hash Join: Scaling Outer Relation



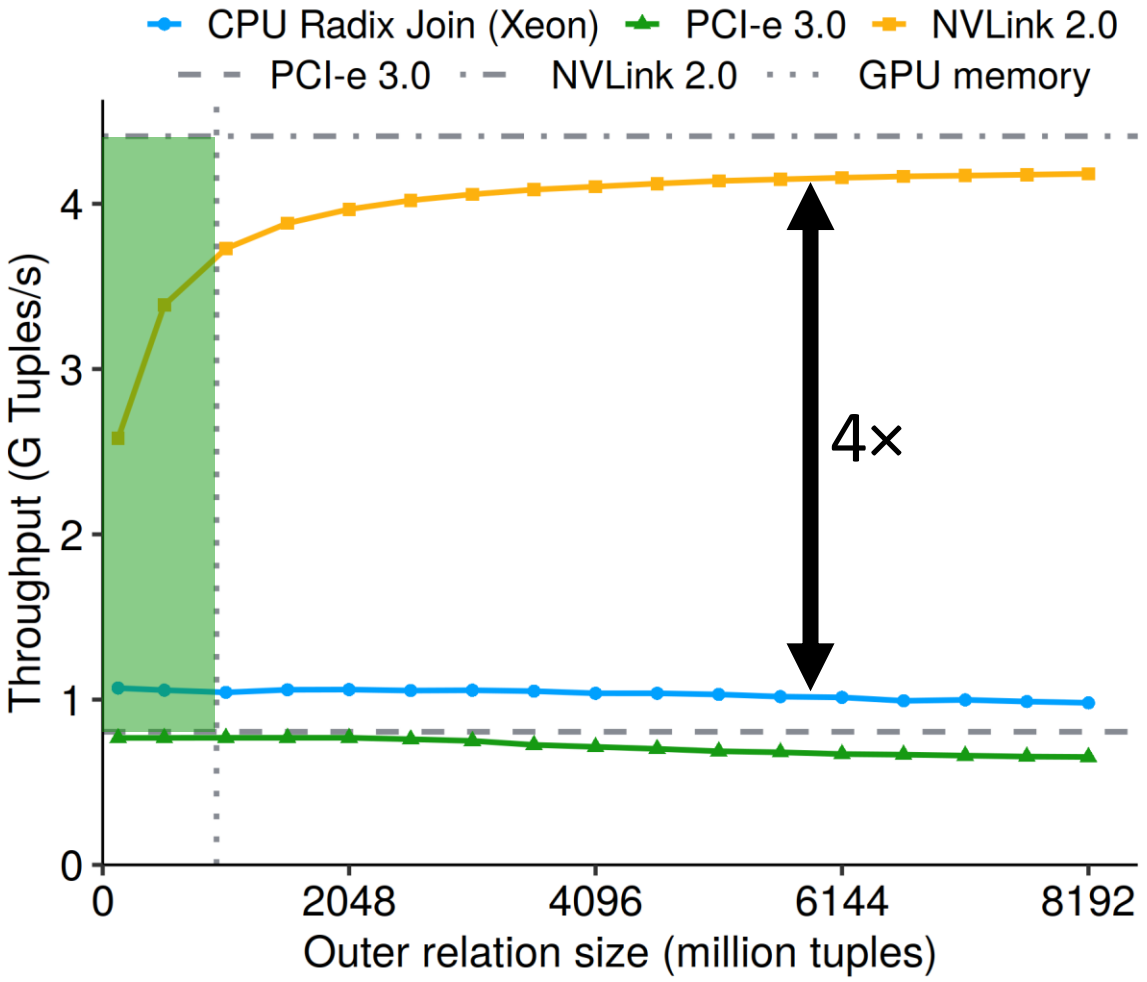
Data: 2 GiB ⋈ 122 GiB
CPU: Intel Xeon
with 12 cores
GPU: Nvidia V100



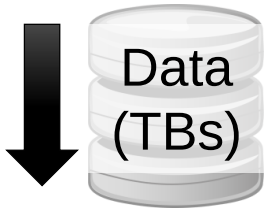
Hash Join: Scaling Outer Relation



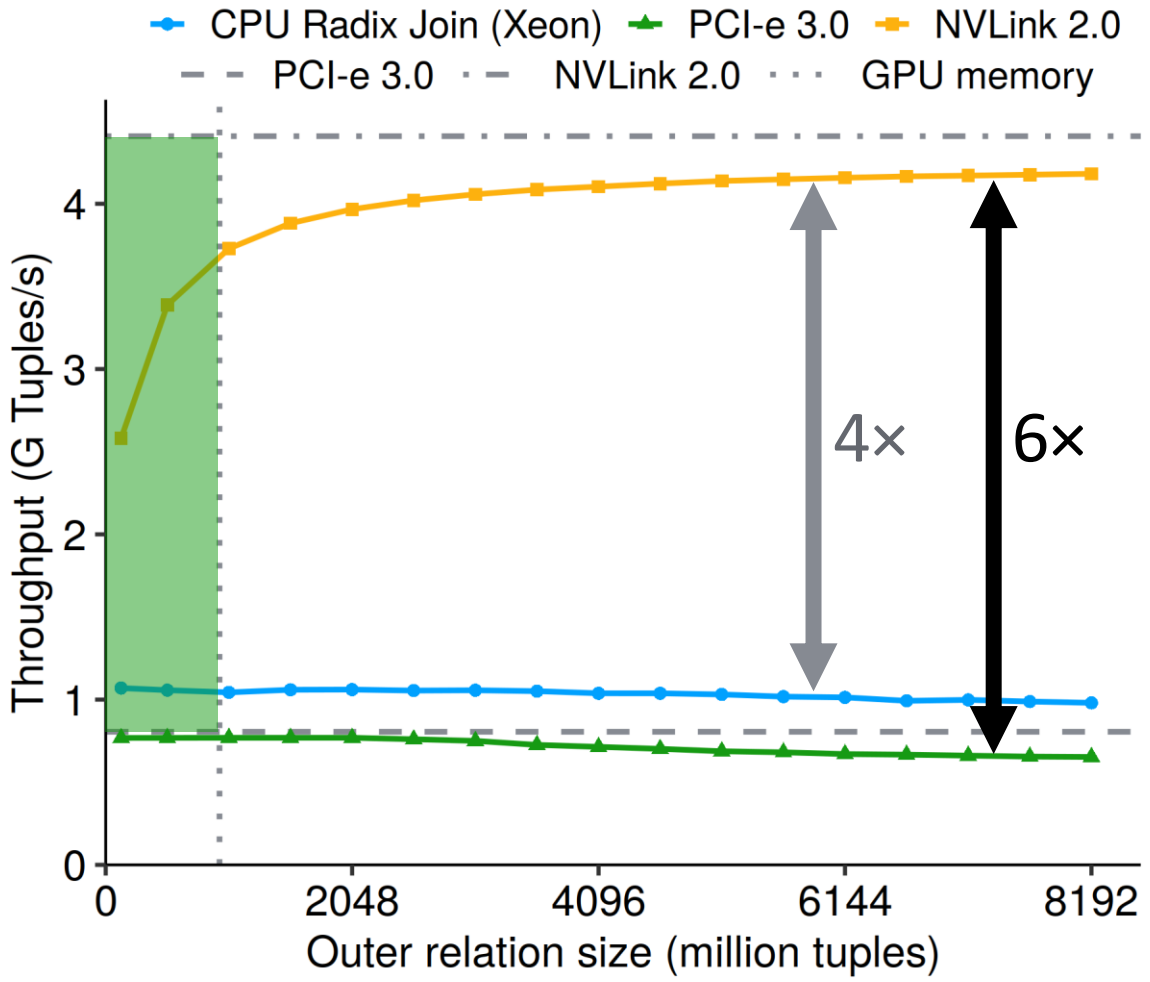
Data: 2 GiB ⋈ 122 GiB
CPU: Intel Xeon
with 12 cores
GPU: Nvidia V100



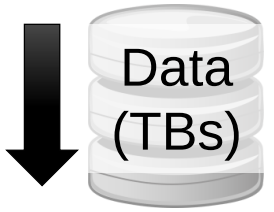
Hash Join: Scaling Outer Relation



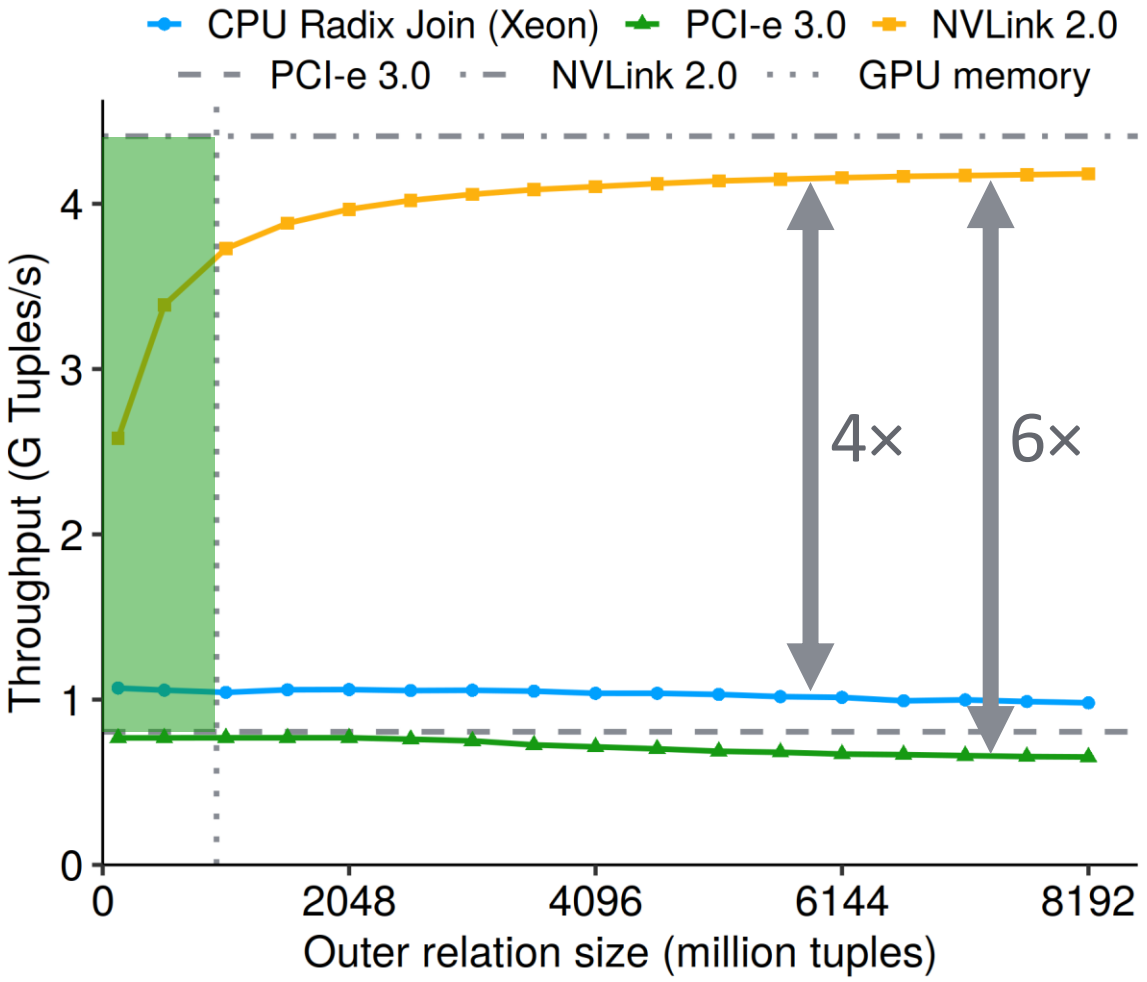
Data: 2 GiB ⋈ 122 GiB
CPU: Intel Xeon
with 12 cores
GPU: Nvidia V100



Hash Join: Scaling Outer Relation

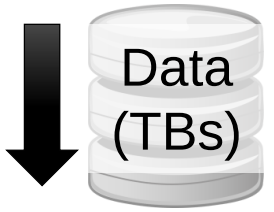


Data: 2 GiB ⋈ 122 GiB
CPU: Intel Xeon
with 12 cores
GPU: Nvidia V100



GPUs can efficiently process large, out-of-core data sets

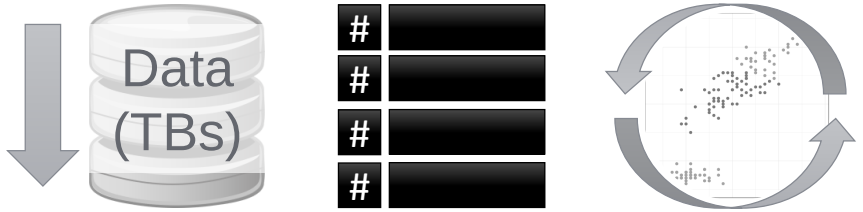
Findings Summary



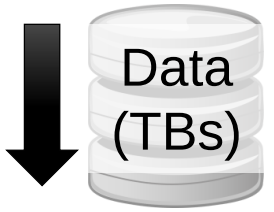
- Fast interconnect enables scalability
- Interconnect-conscious data access
  - Coherence
- Scale outer relation of hash join
  - 4× speedup

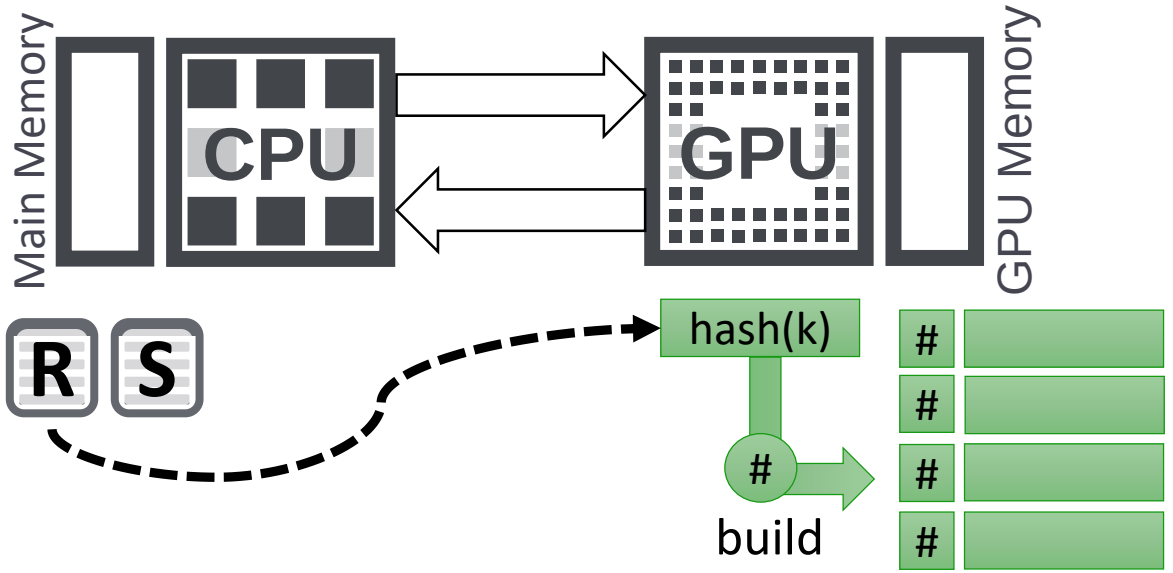
Agenda

1. Motivation
2. Data-intensive query processing
3. **Stateful data processing**
4. Iterative algorithms
5. Conclusion

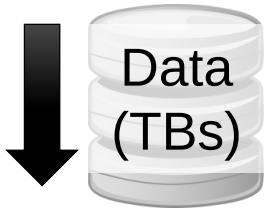


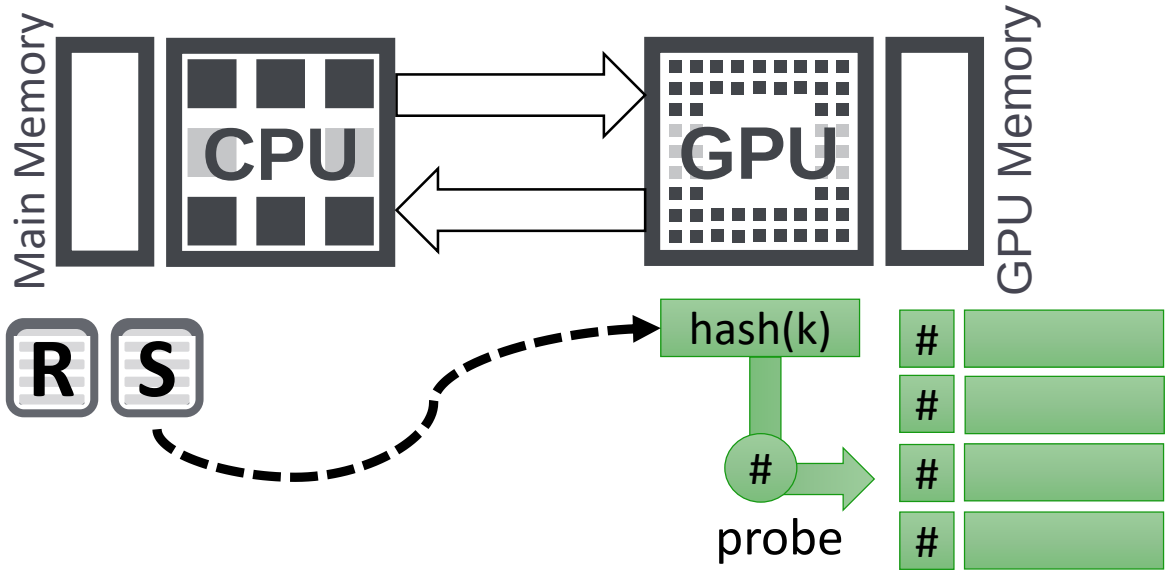
Hash Join: Scaling Inner Relation



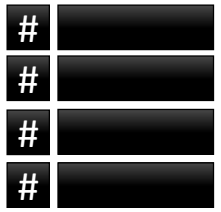


Hash Join: Scaling Inner Relation

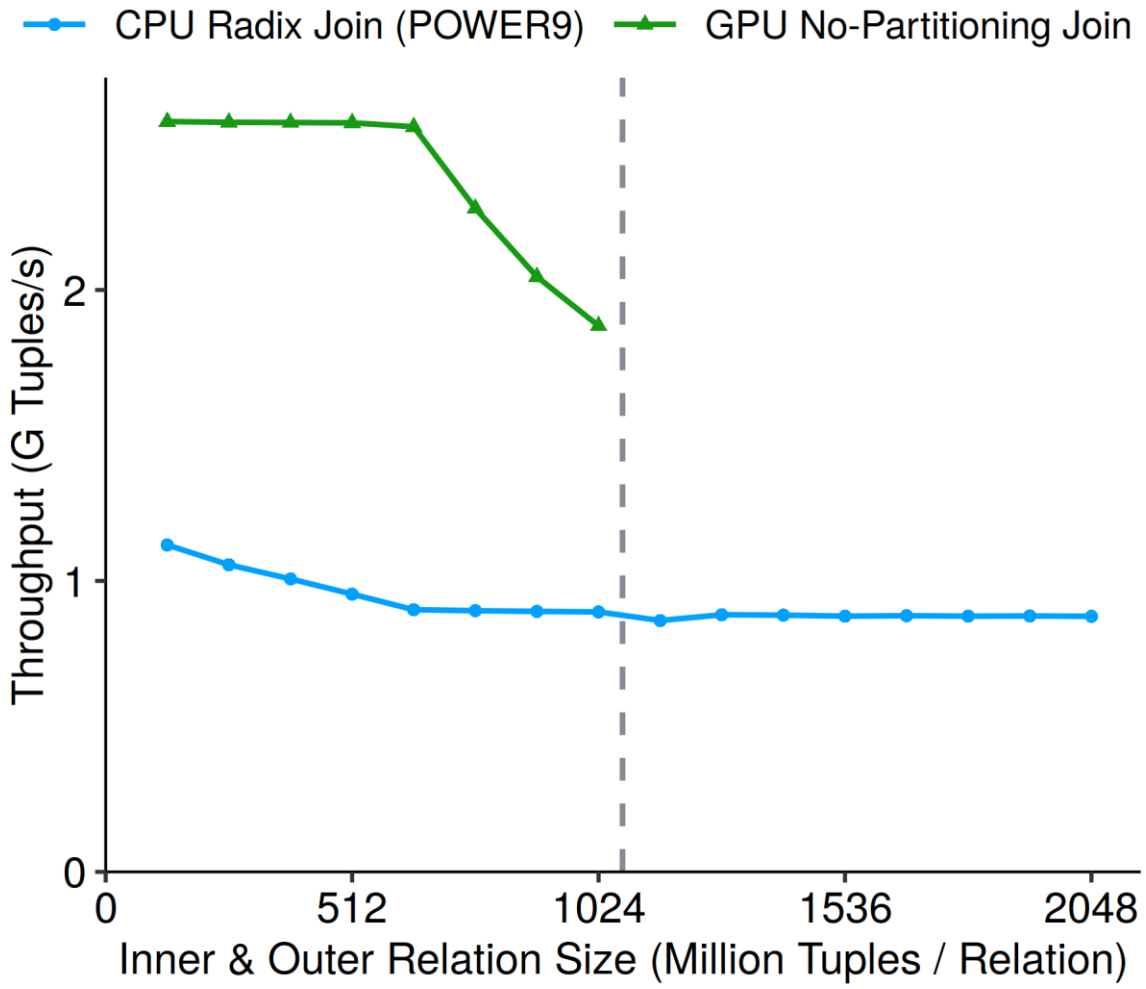




Hash Join: Scaling Inner Relation

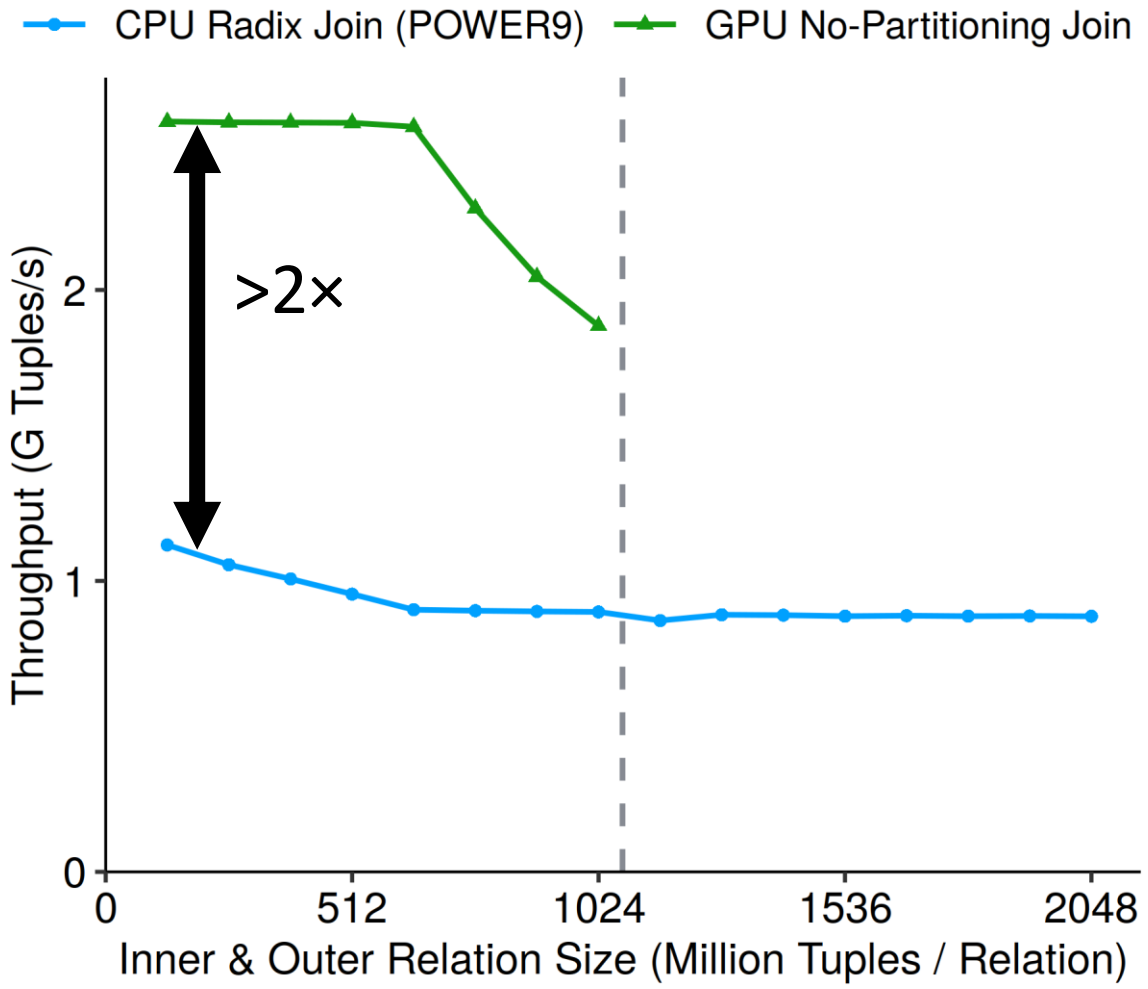


Data: 30 GiB ⋈ 30 GiB
CPU: IBM POWER9
with 16 cores
GPU: Nvidia V100
with NVLink 2.0

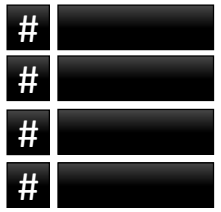


Hash Join: Scaling Inner Relation

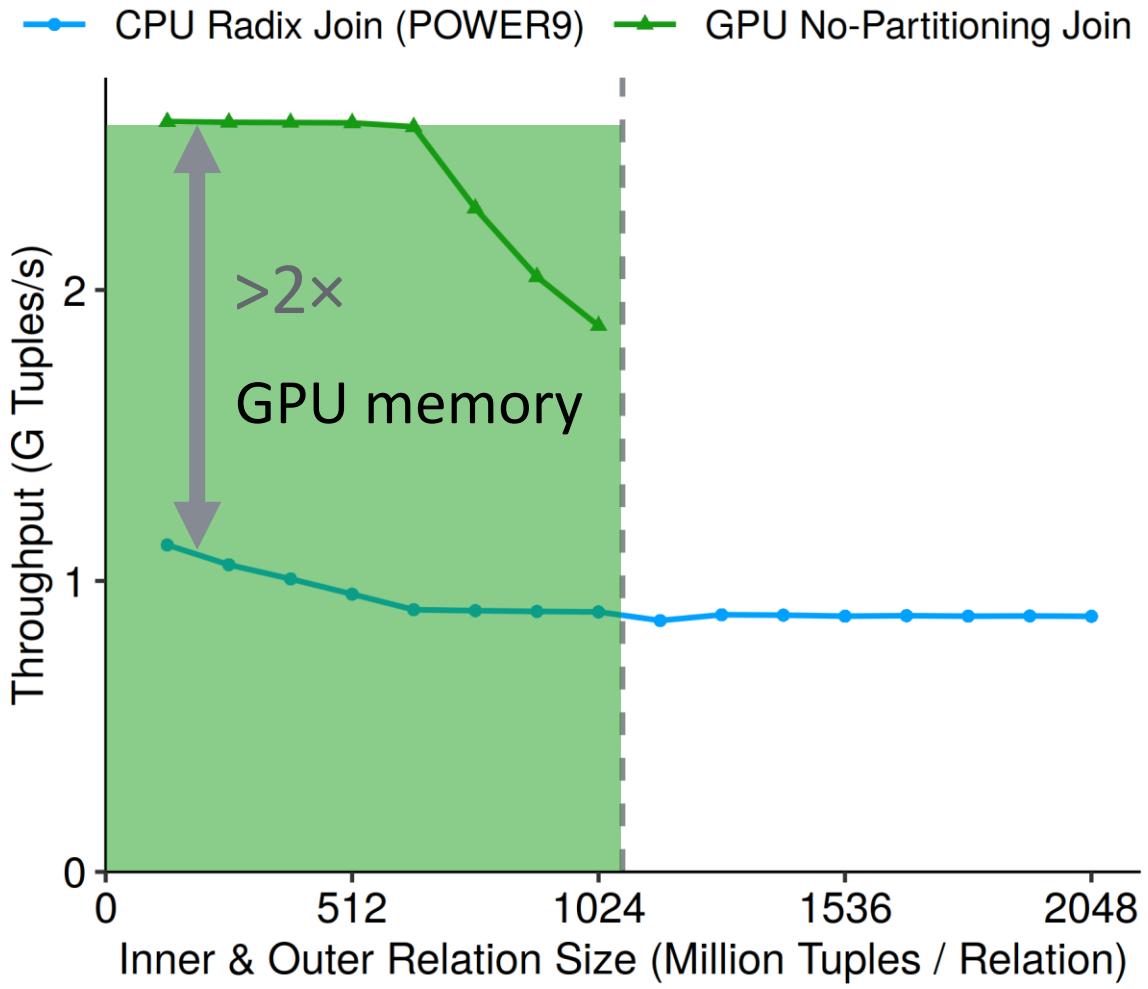
Data: 30 GiB ⋈ 30 GiB
CPU: IBM POWER9
with 16 cores
GPU: Nvidia V100
with NVLink 2.0



Hash Join: Scaling Inner Relation

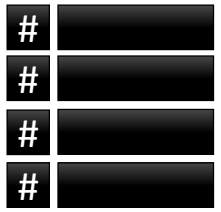


Data: 30 GiB ⋈ 30 GiB
CPU: IBM POWER9
with 16 cores
GPU: Nvidia V100
with NVLink 2.0

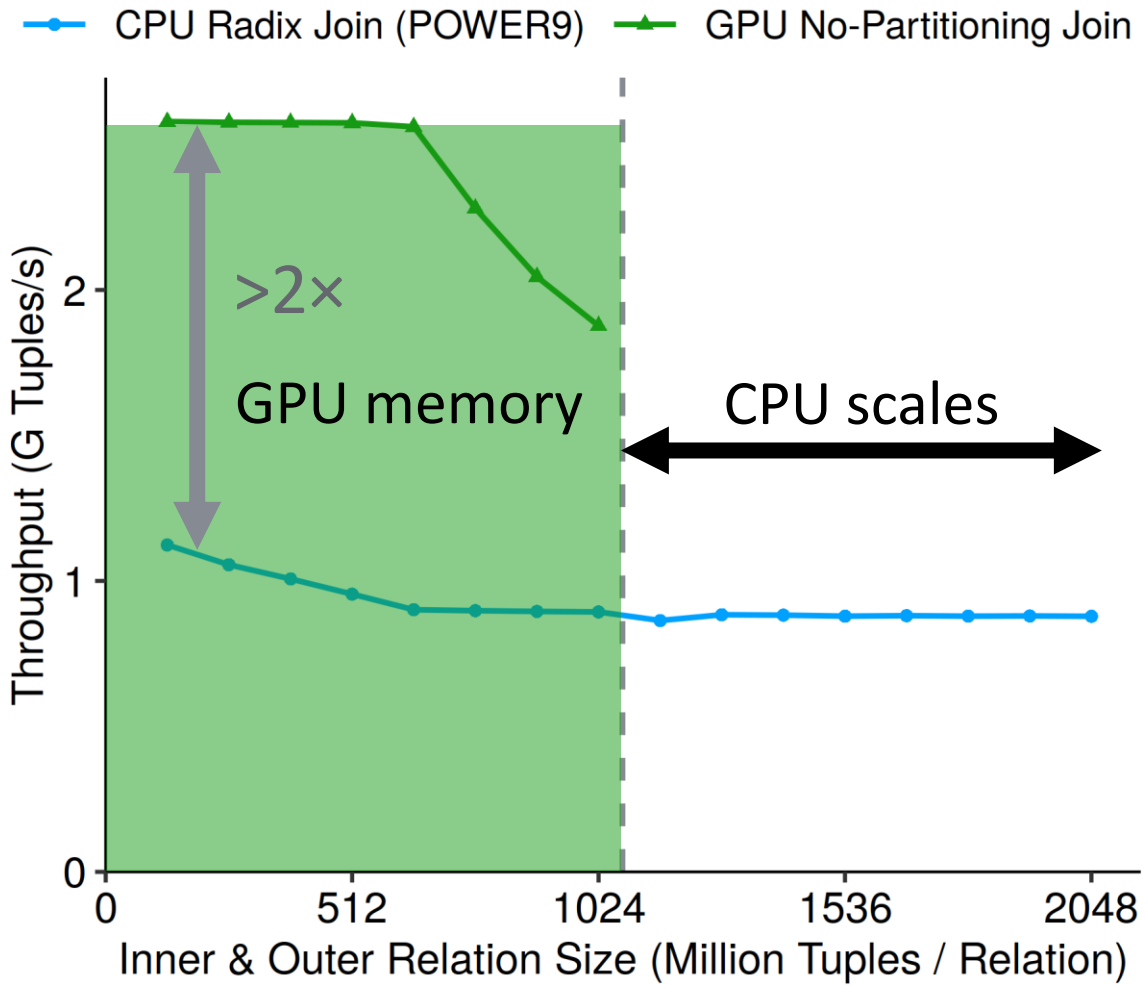


- Join state has limited size

Hash Join: Scaling Inner Relation

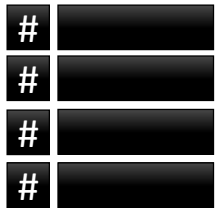


Data: 30 GiB ⋈ 30 GiB
CPU: IBM POWER9
with 16 cores
GPU: Nvidia V100
with NVLink 2.0

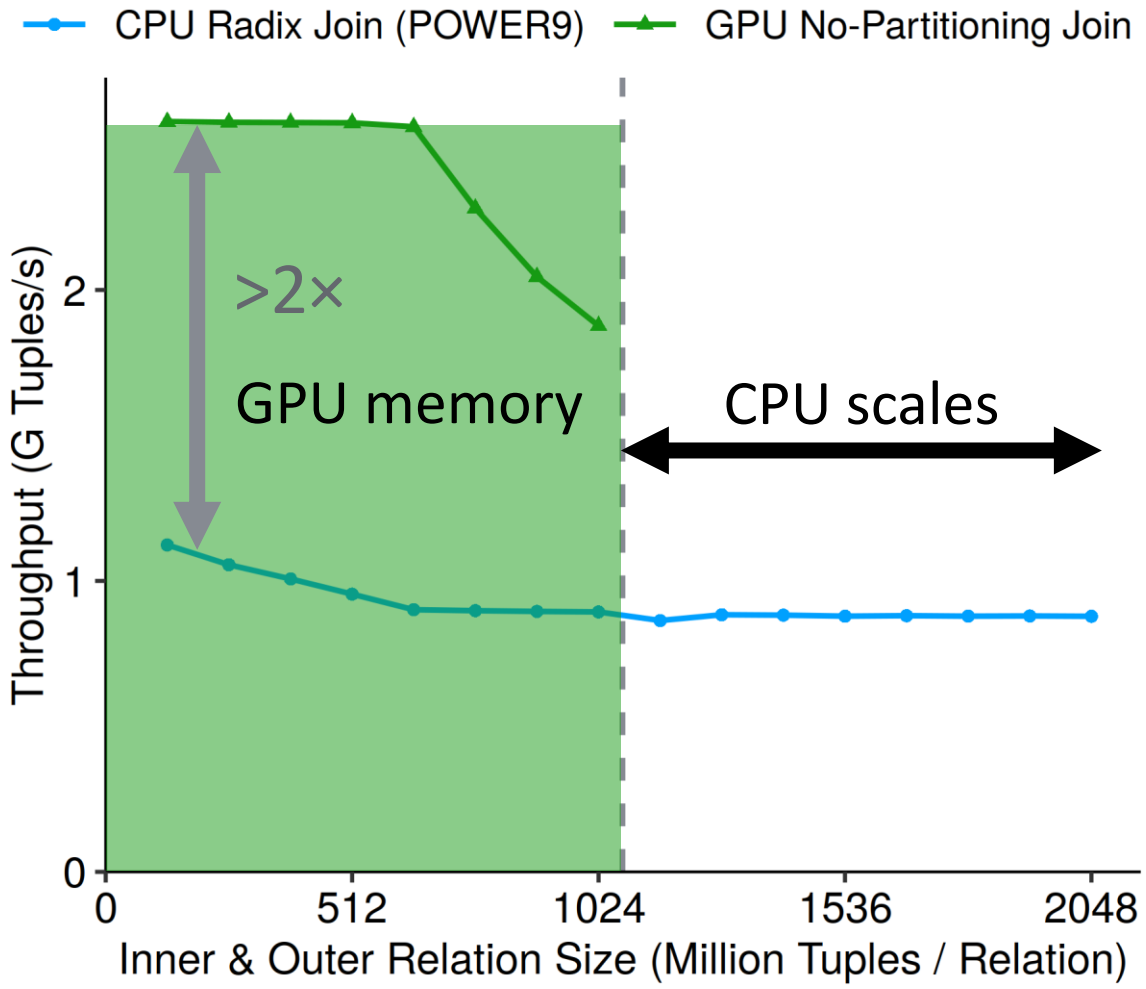


- Join state has limited size

Hash Join: Scaling Inner Relation



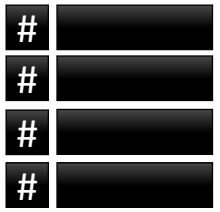
Data: 30 GiB ⋈ 30 GiB
CPU: IBM POWER9
with 16 cores
GPU: Nvidia V100
with NVLink 2.0

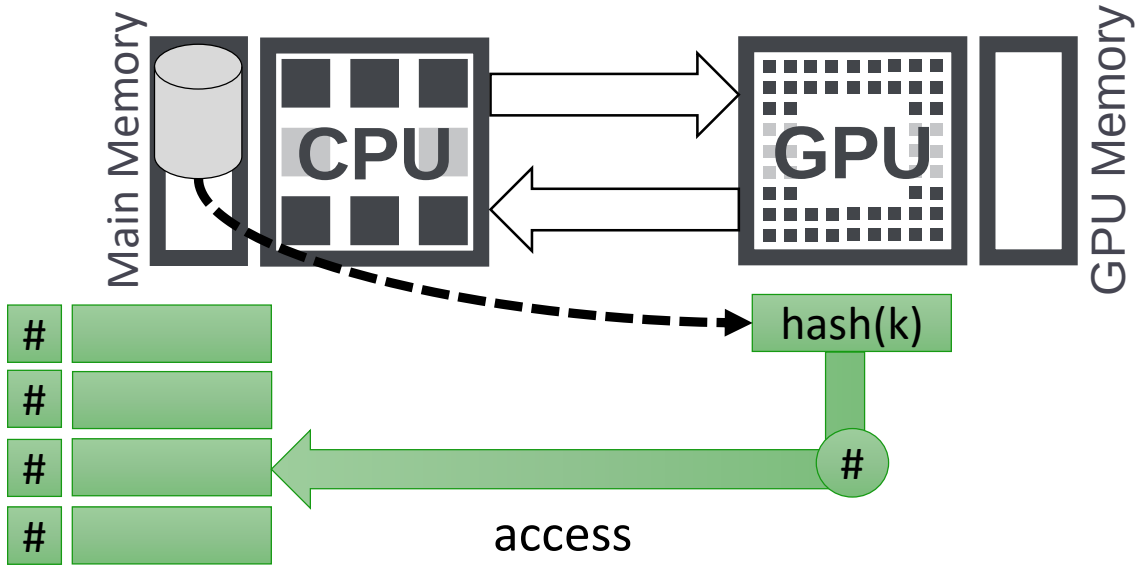


How can GPU hash join scale to a large join state?

- Join state has limited size

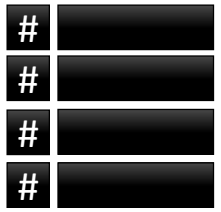
Approach: Spill Hash Table



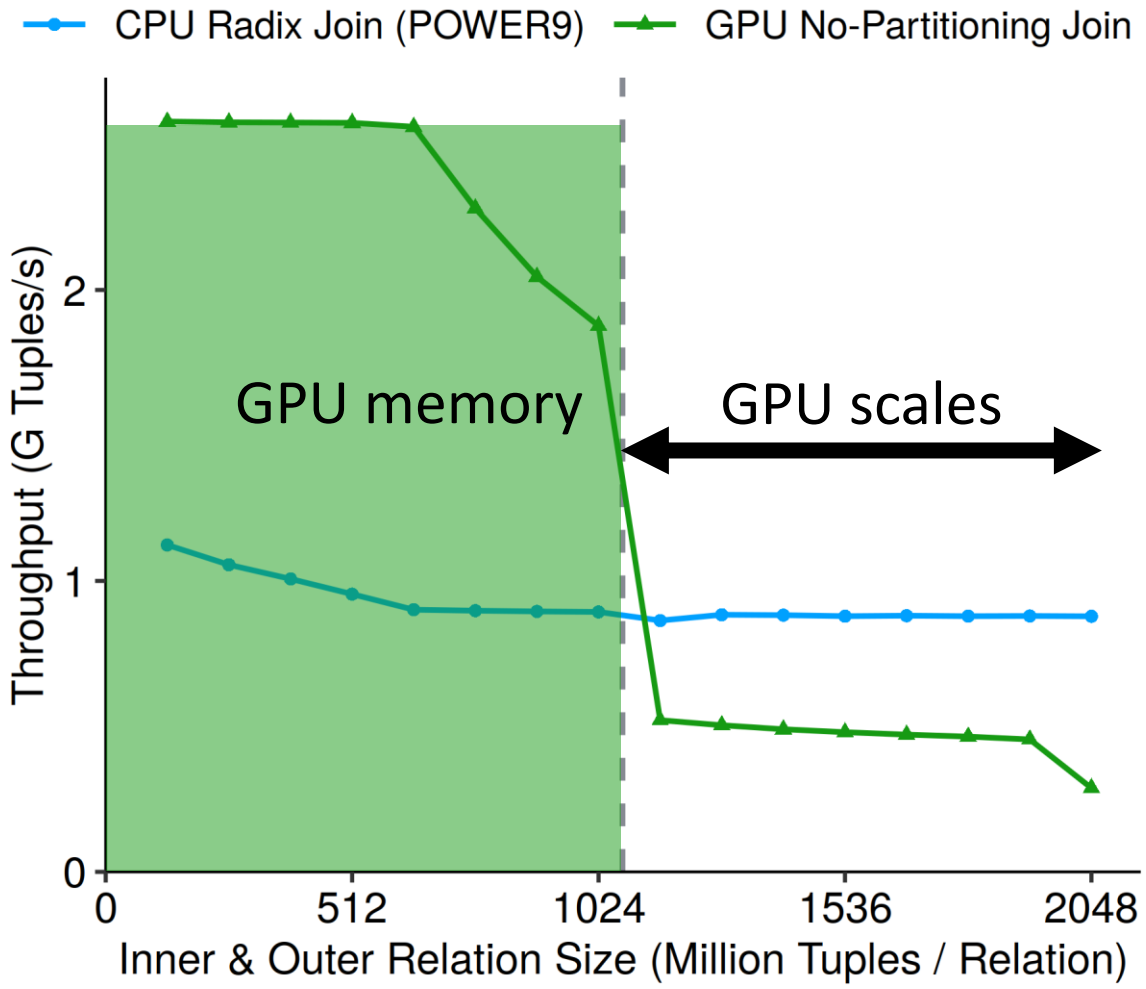


How can GPU hash join scale to a large join state?

Hash Join: Scaling Inner Relation



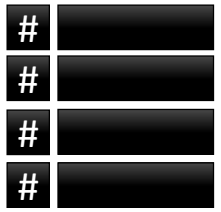
Data: 30 GiB ⋈ 30 GiB
CPU: IBM POWER9
with 16 cores
GPU: Nvidia V100
with NVLink 2.0



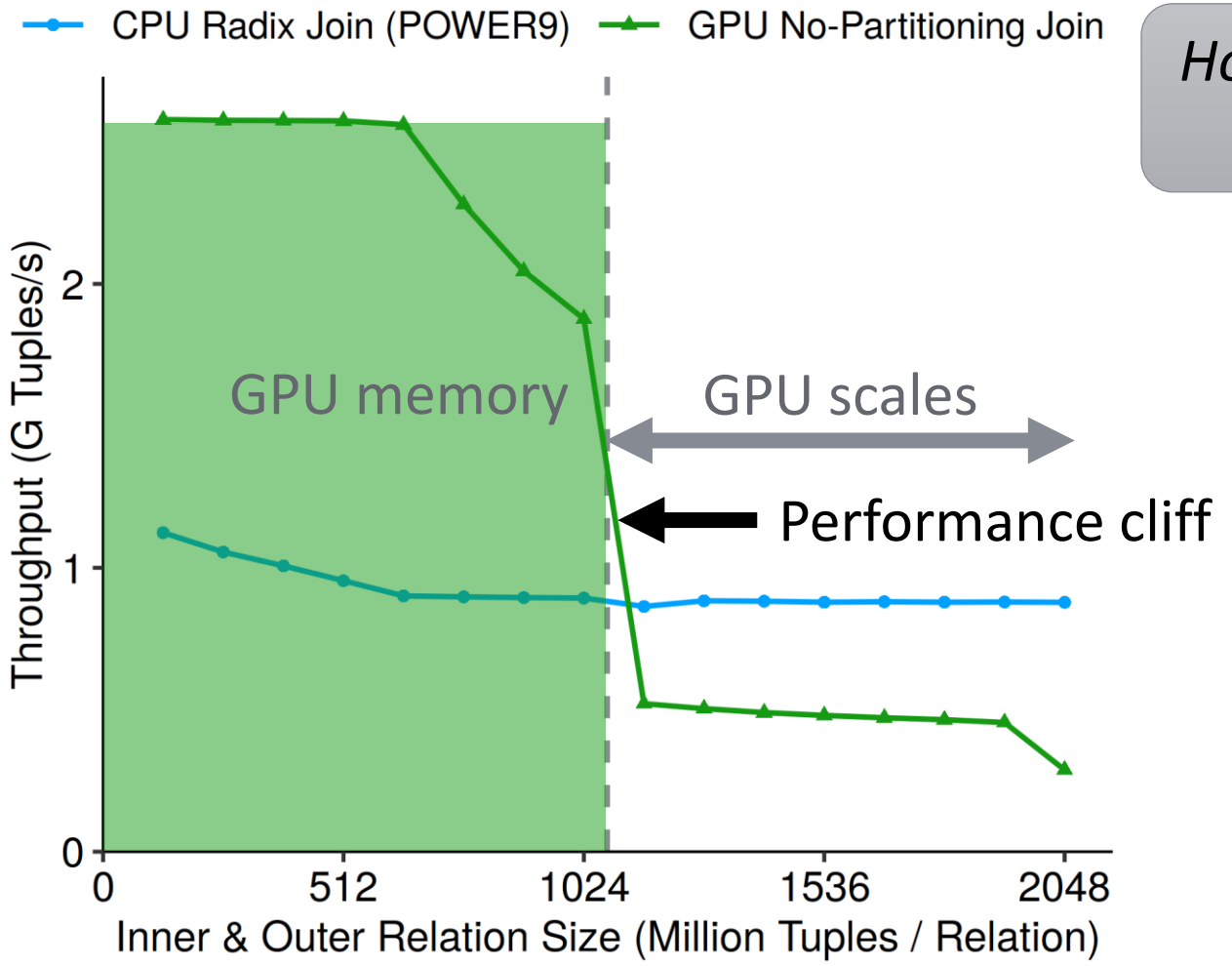
How can GPU hash join scale to a large join state?

- Scalable ✓

Hash Join: Scaling Inner Relation



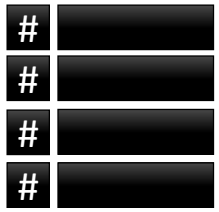
Data: 30 GiB ⋈ 30 GiB
CPU: IBM POWER9
with 16 cores
GPU: Nvidia V100
with NVLink 2.0



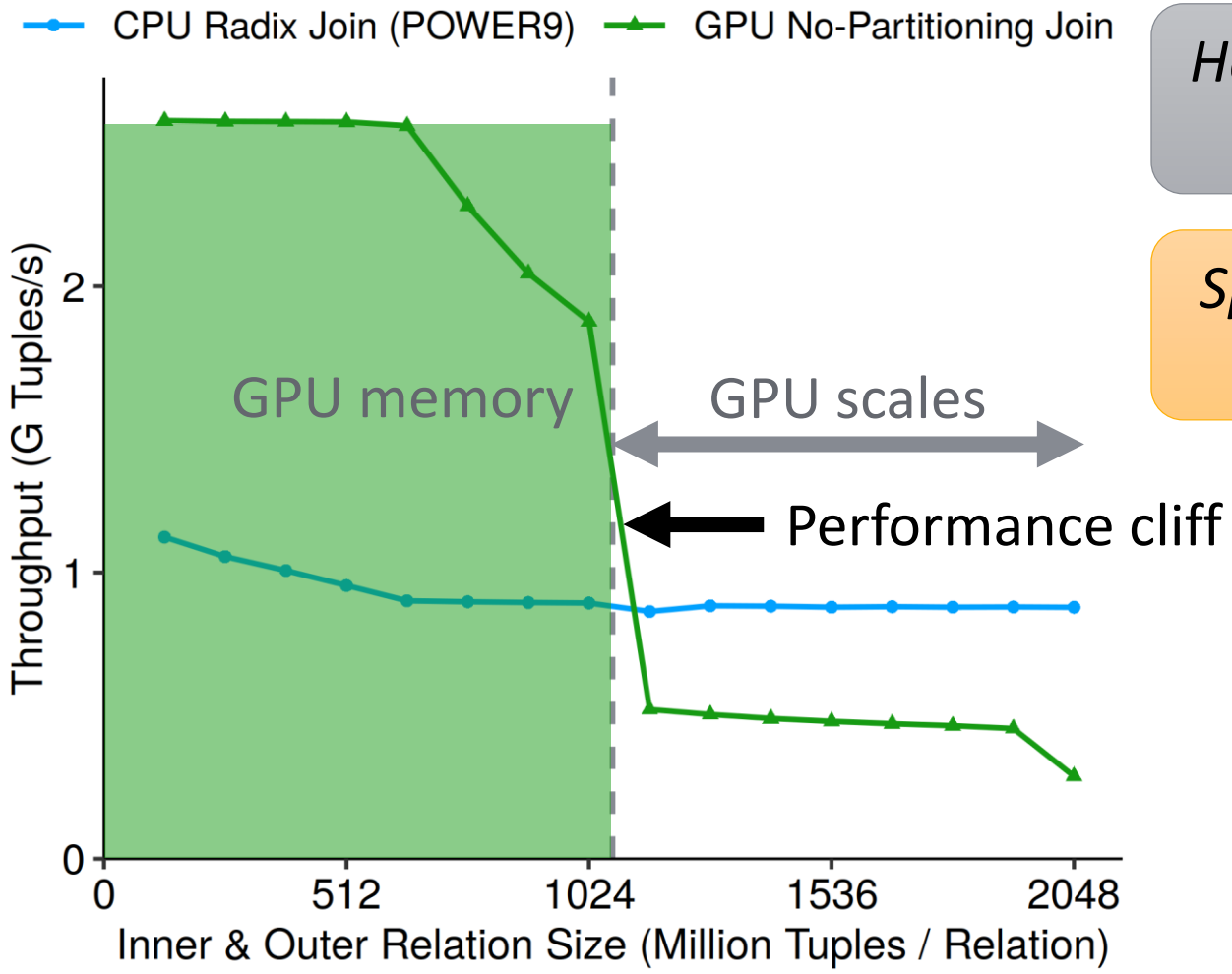
How can GPU hash join scale to a large join state?

- Scalable ✓

Hash Join: Scaling Inner Relation



Data: 30 GiB ⋈ 30 GiB
CPU: IBM POWER9
with 16 cores
GPU: Nvidia V100
with NVLink 2.0

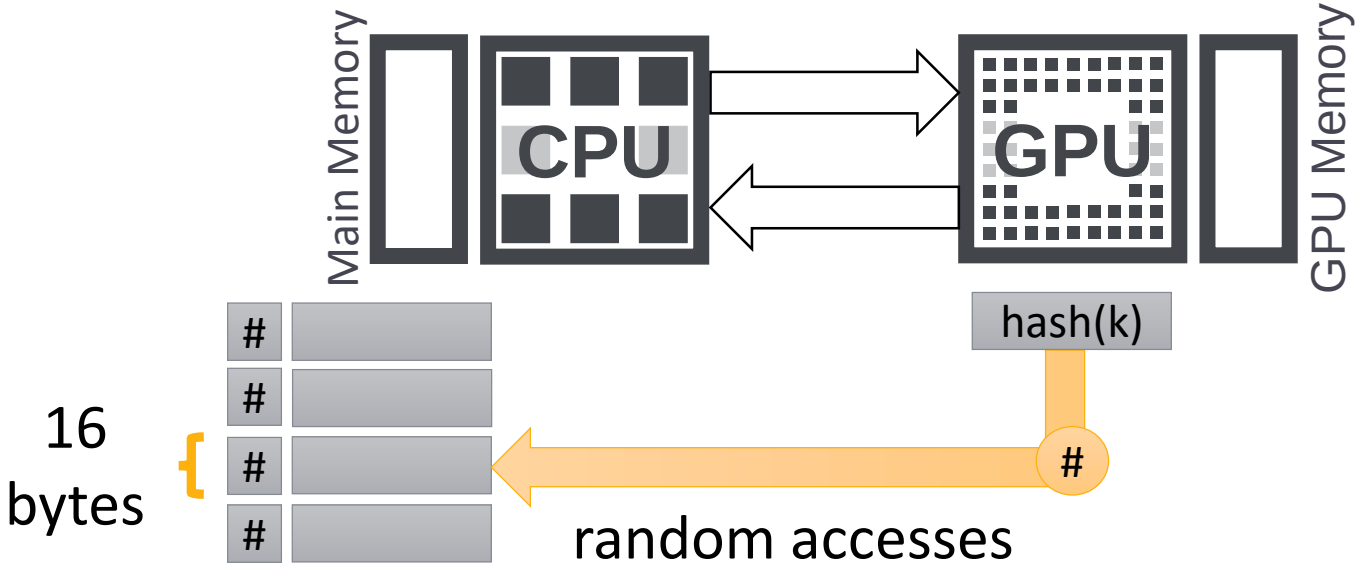


How can GPU hash join scale to a large join state?

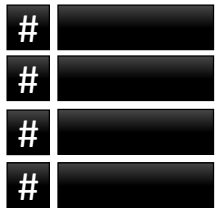
Spilling the hash table is not efficient

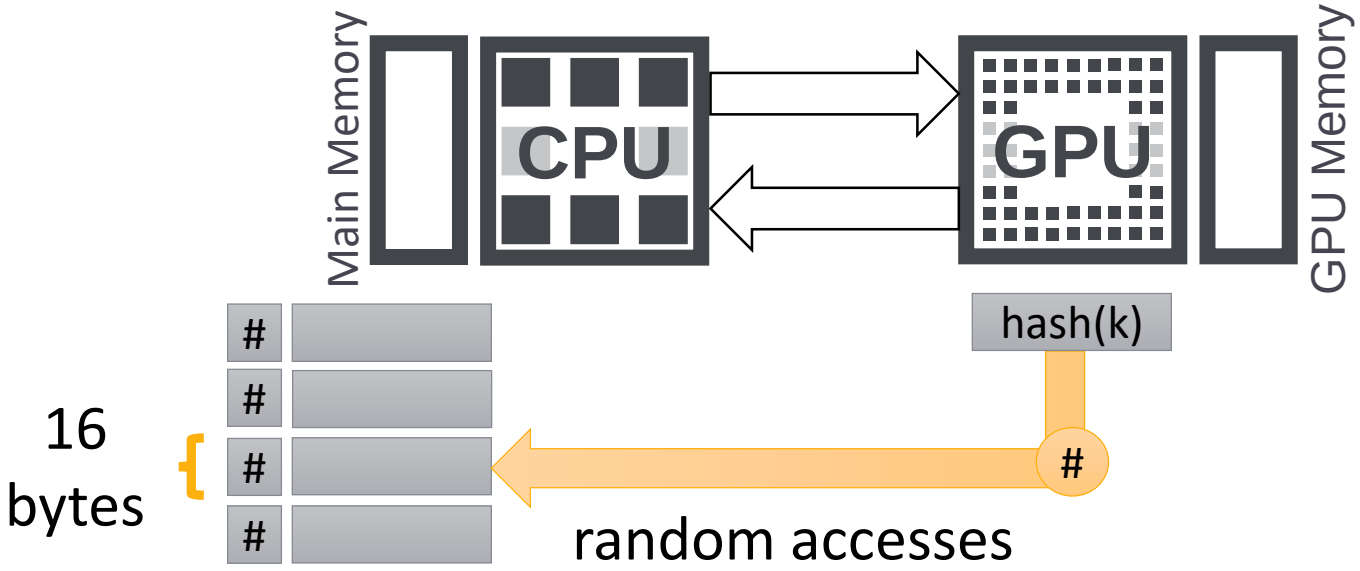
- Scalable ✓
- Not efficient

Why is Spilling not Efficient?



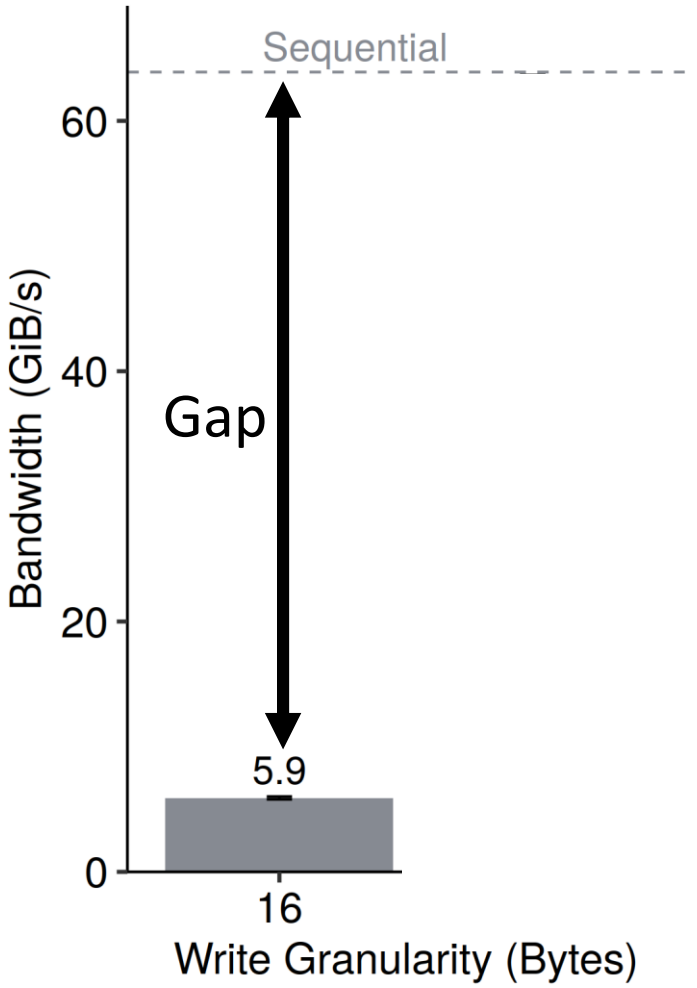
Why is Spilling not Efficient?





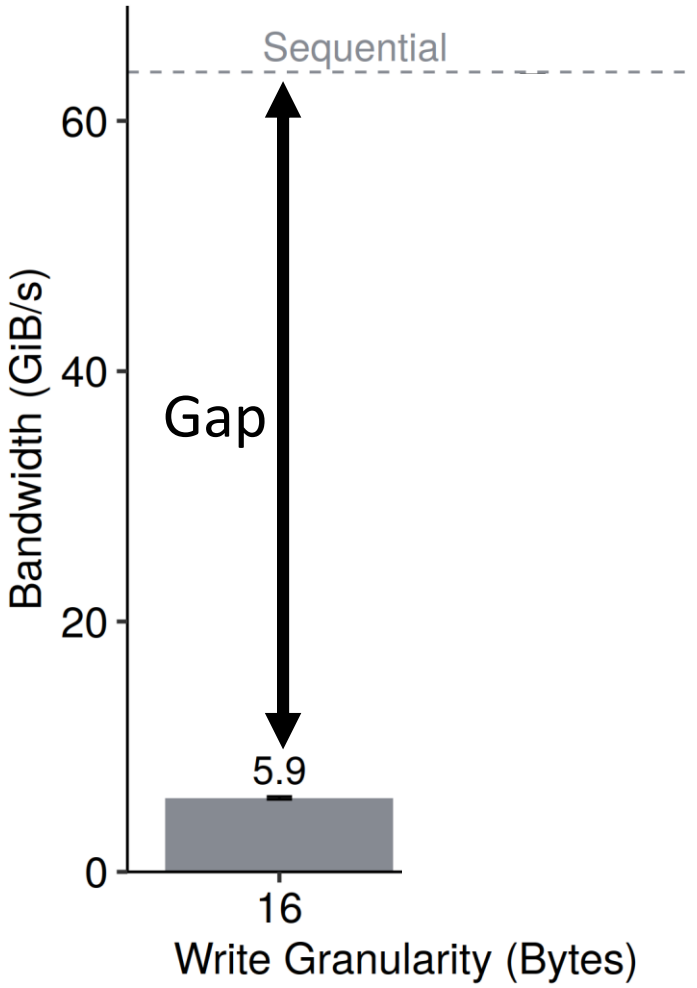
Hash join incurs fine-grained random access pattern

Random Access Bandwidth Gap



Hash join incurs fine-grained random access pattern

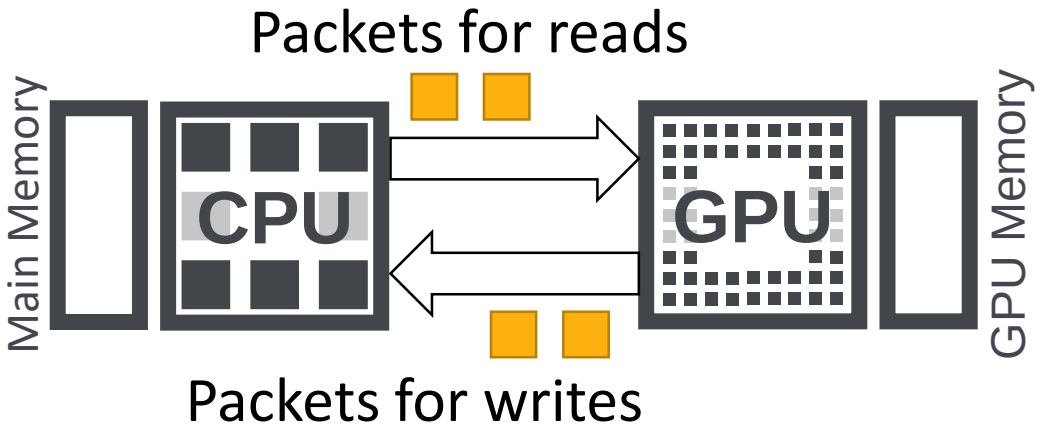
Random Access Bandwidth Gap



Hash join incurs fine-grained random access pattern

How can GPUs efficiently access out-of-core state?

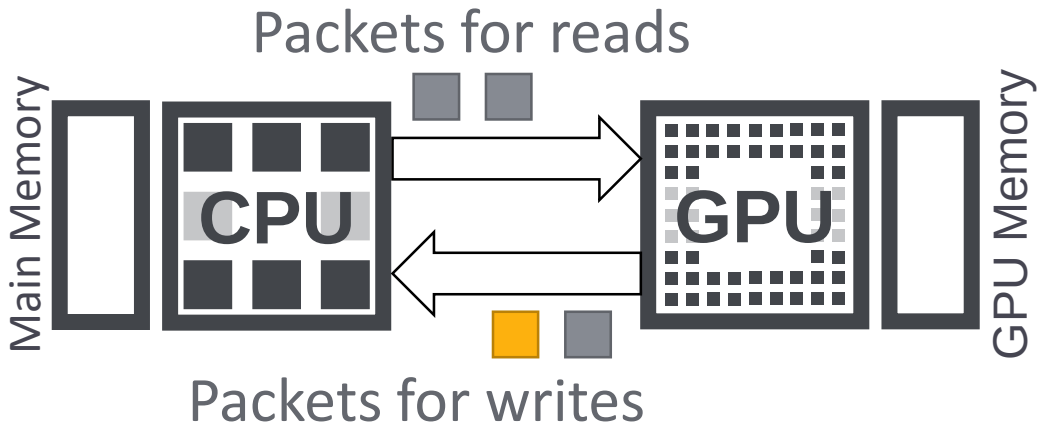
Efficient State Access



Hash join incurs fine-grained random access pattern

How can GPUs efficiently access out-of-core state?

Efficient State Access

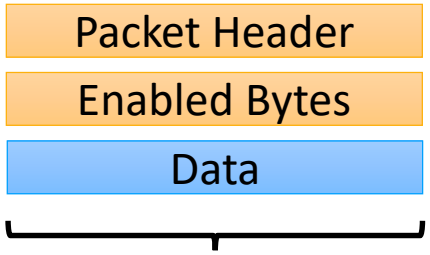


Hash join incurs fine-grained random access pattern

How can GPUs efficiently access out-of-core state?

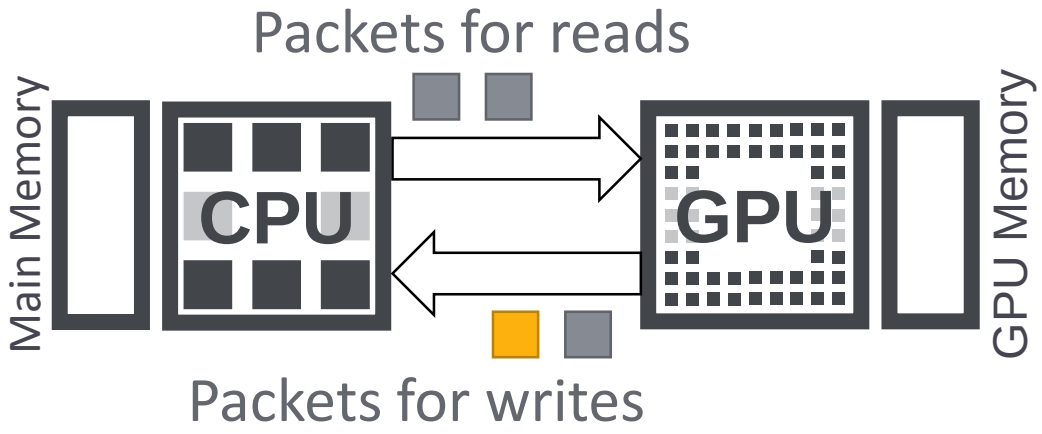
16-byte write packet

67% overhead



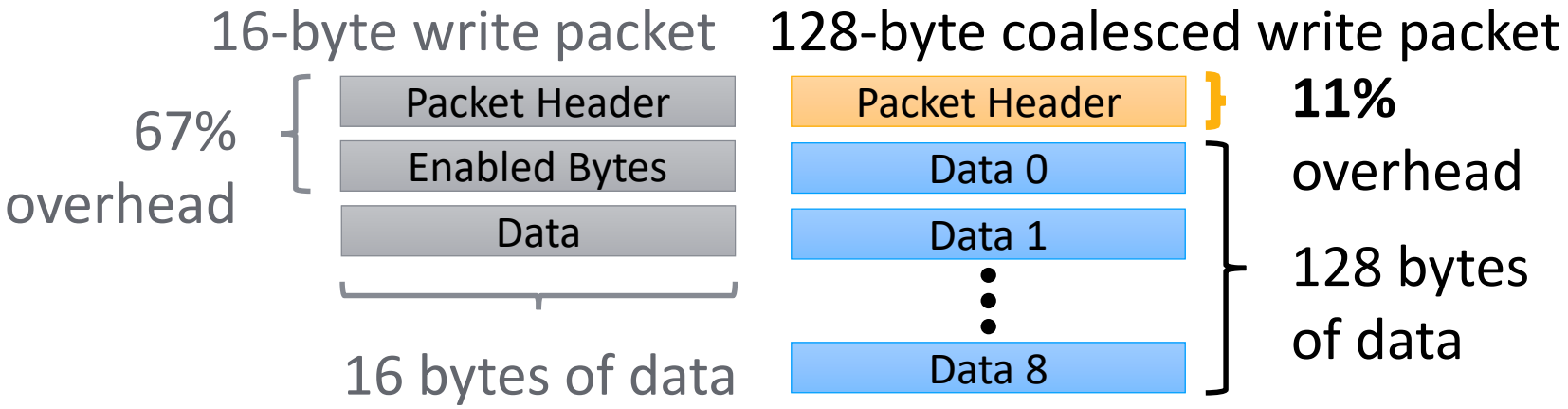
16 bytes of data

Efficient State Access

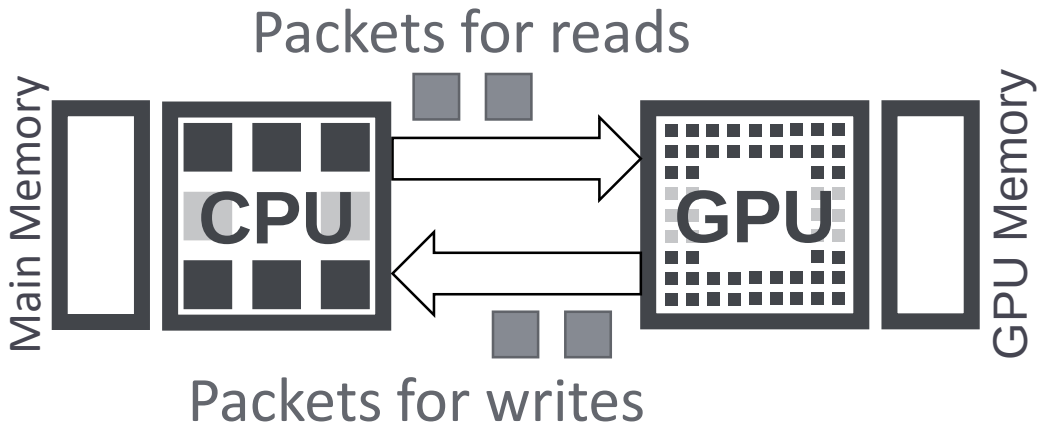


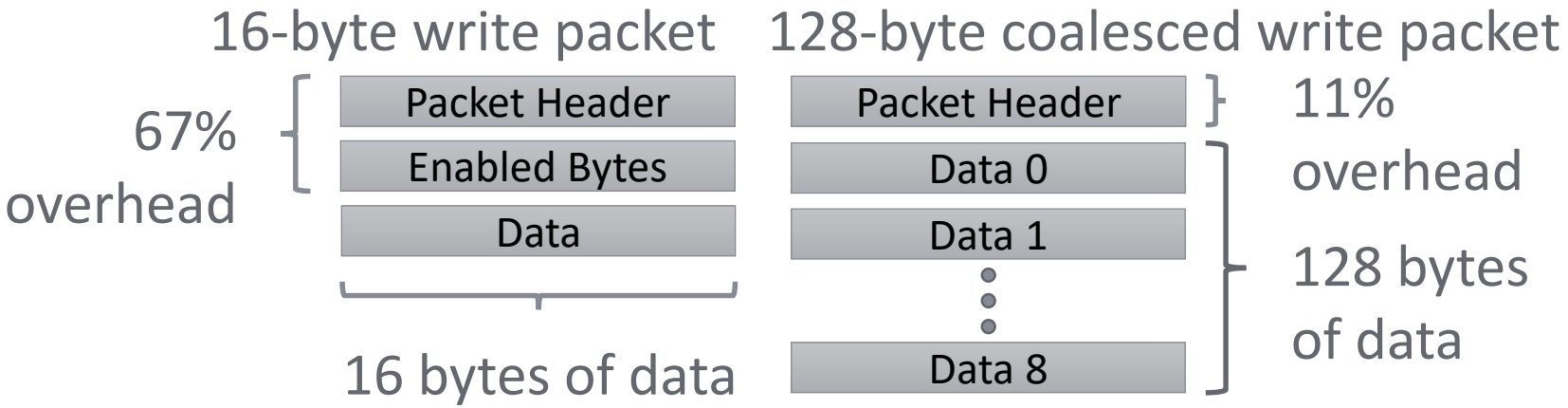
Hash join incurs fine-grained random access pattern

How can GPUs efficiently access out-of-core state?



Efficient State Access



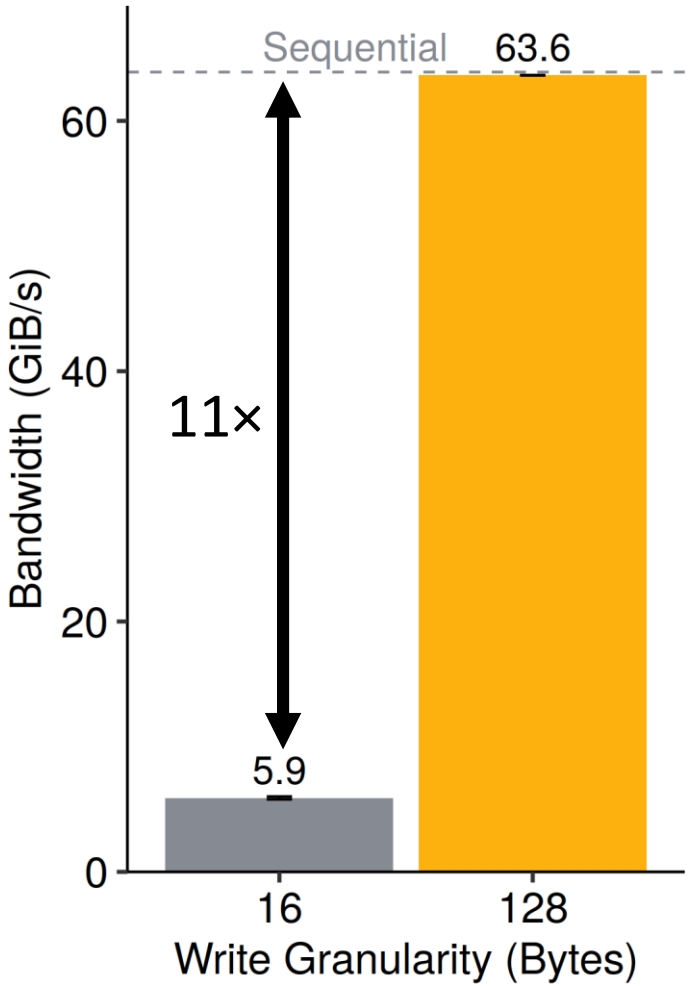


Hash join incurs fine-grained random access pattern

How can GPUs efficiently access out-of-core state?

Insight: Perfect coalescing reduces overhead

Perfect Coalescing

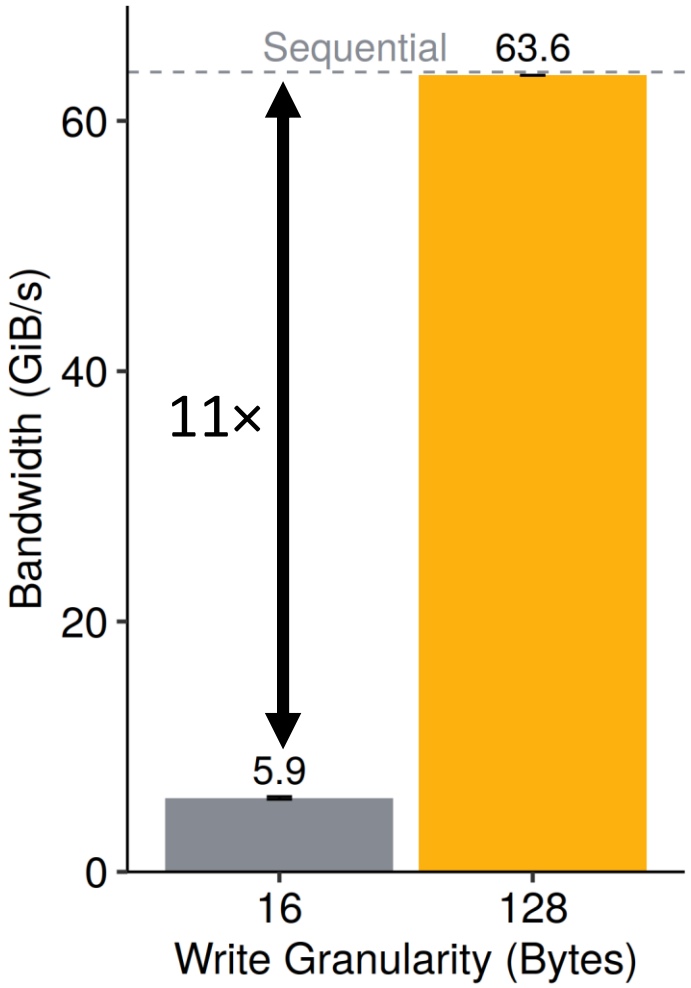


Hash join incurs fine-grained random access pattern

How can GPUs efficiently access out-of-core state?

Insight: Perfect coalescing reduces overhead

Perfect Coalescing



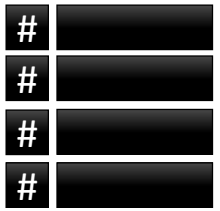
Hash join incurs fine-grained random access pattern

How can GPUs efficiently access out-of-core state?

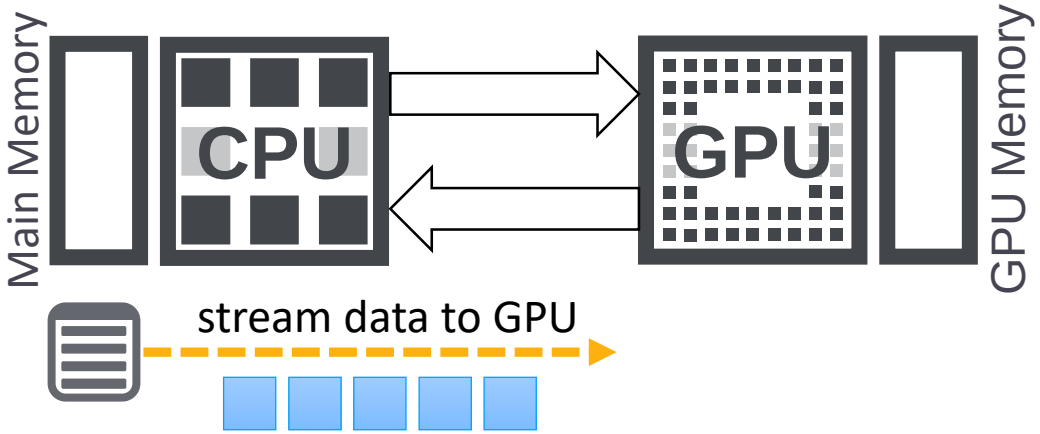
Insight: Perfect coalescing reduces overhead

Interconnect-conscious state access

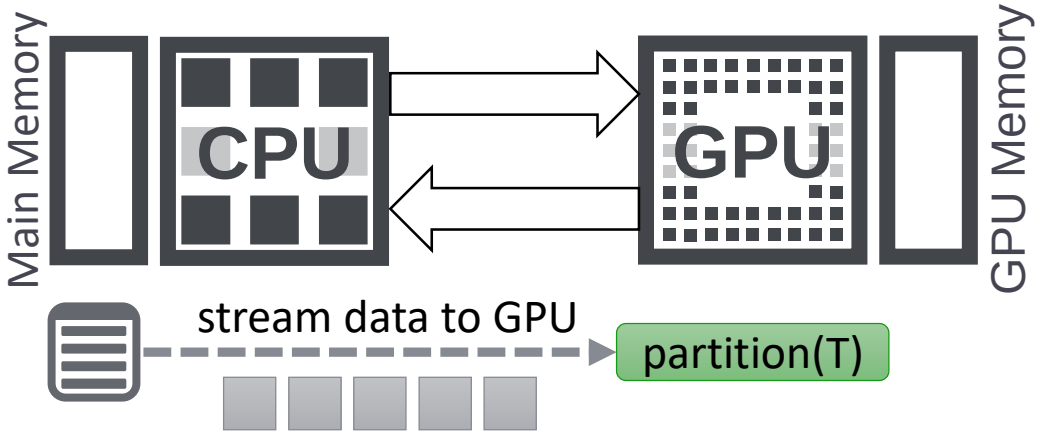
Approach: Out-of-Core Partitioning



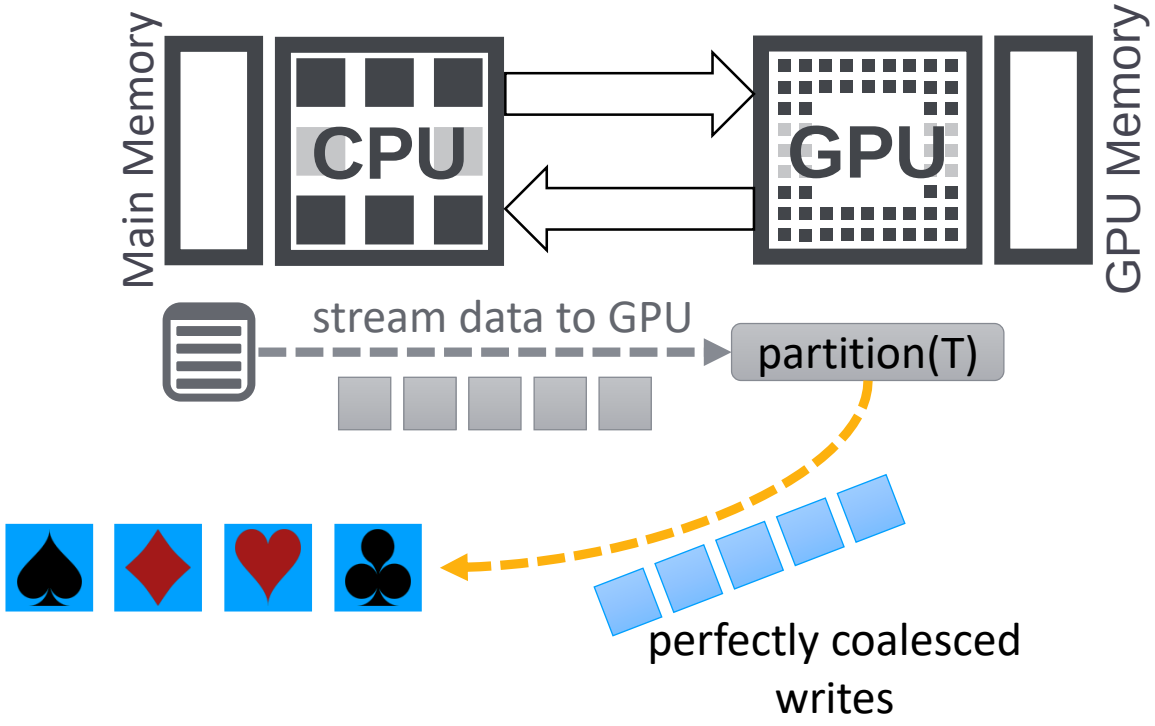
Approach: Out-of-Core Partitioning



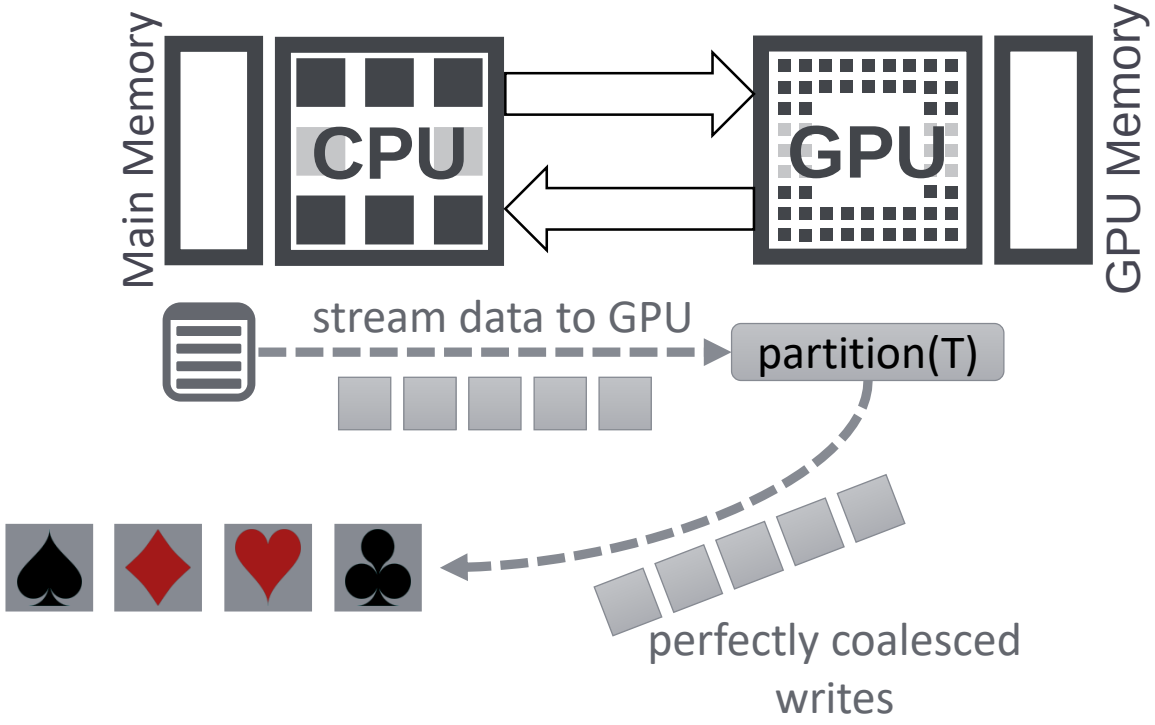
Approach: Out-of-Core Partitioning



Approach: Out-of-Core Partitioning

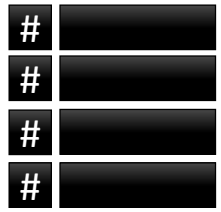


Approach: Out-of-Core Partitioning

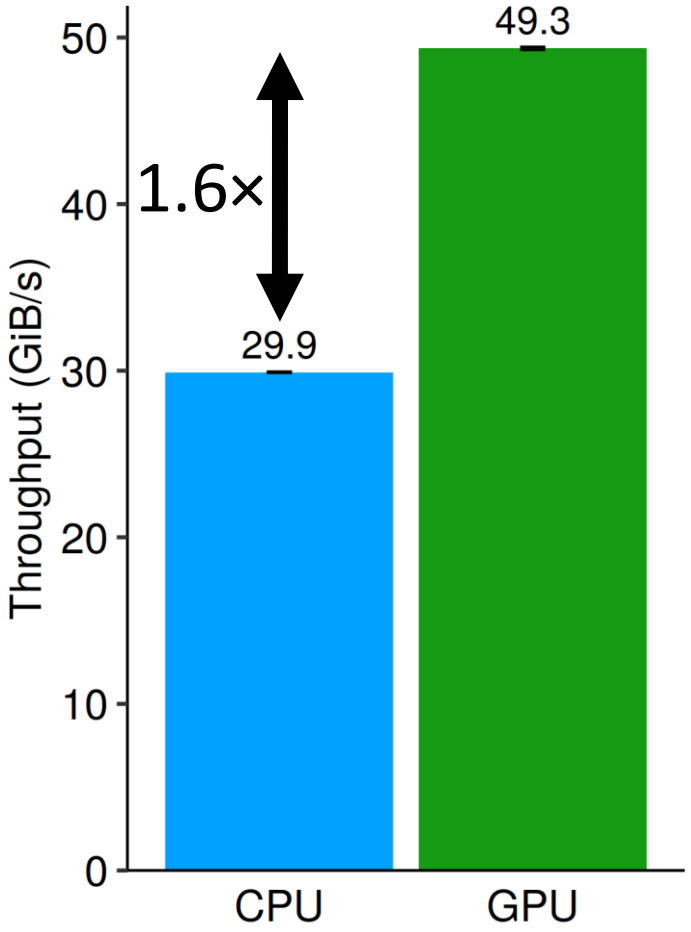


Exploit perfect coalescing using out-of-core partitioning

Out-of-Core Partitioning Performance

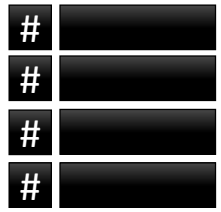


Data: 15 GB
Fanout: 512 partitions

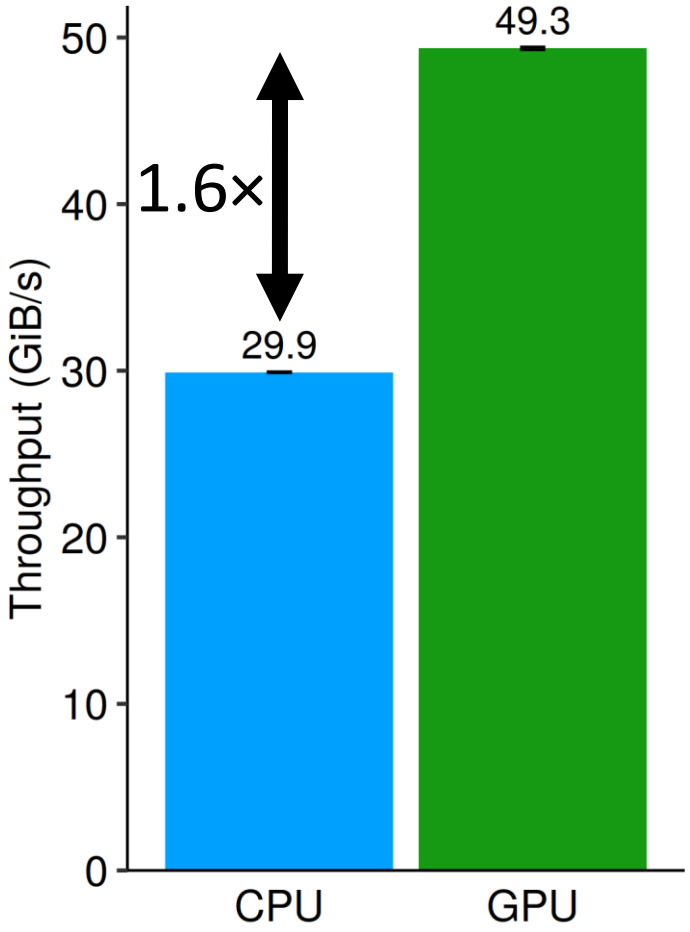


Exploit perfect coalescing using out-of-core partitioning

Out-of-Core Partitioning Performance



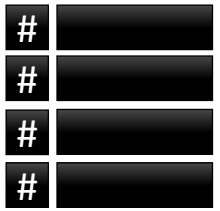
Data: 15 GB
Fanout: 512 partitions



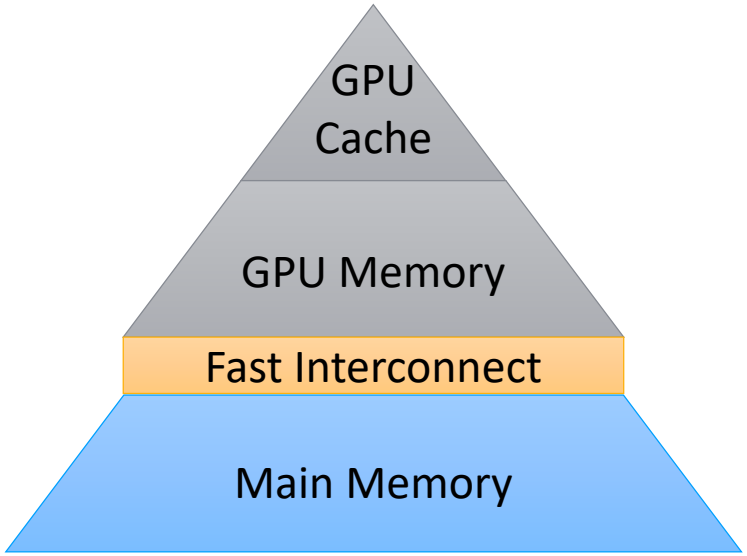
Exploit perfect coalescing using out-of-core partitioning

GPU efficiently partitions data out-of-core

The Triton Join: Overview

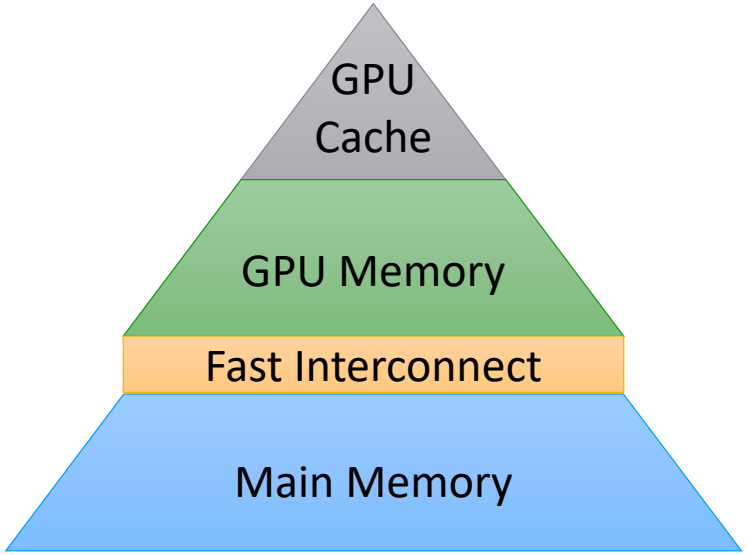


The Triton Join: Overview



Out-of-core radix partitioning

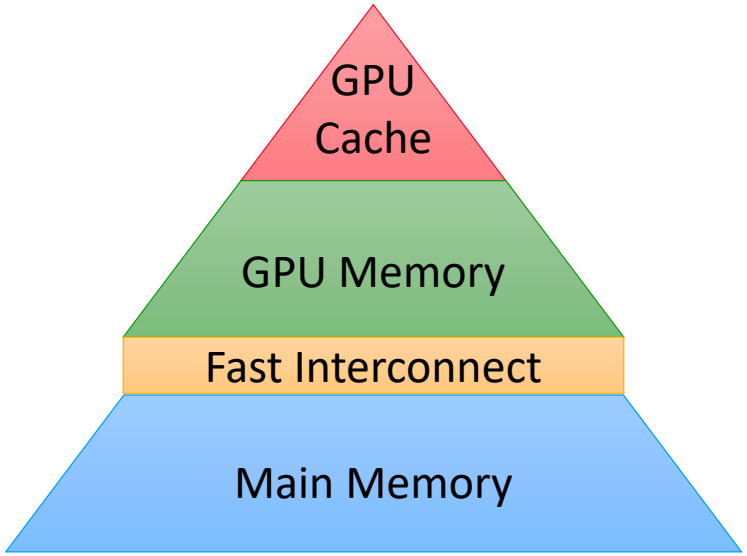
The Triton Join: Overview



Caching partitions in GPU memory

Out-of-core radix partitioning

The Triton Join: Overview

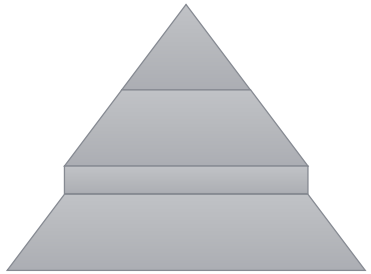


Build & probe hash table

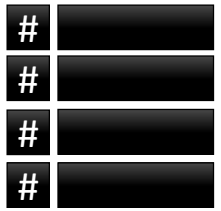
Caching partitions in GPU memory

Out-of-core radix partitioning

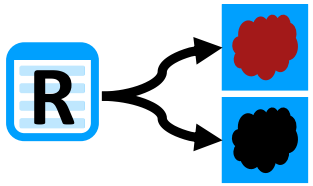
The Triton Join: Deep Dive

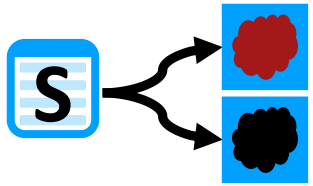


The Triton Join: Deep Dive

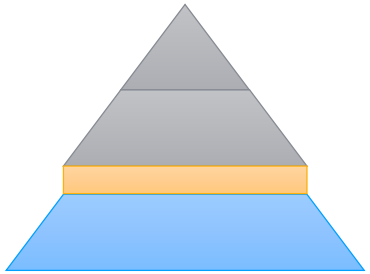


Out-of-core radix partitioning

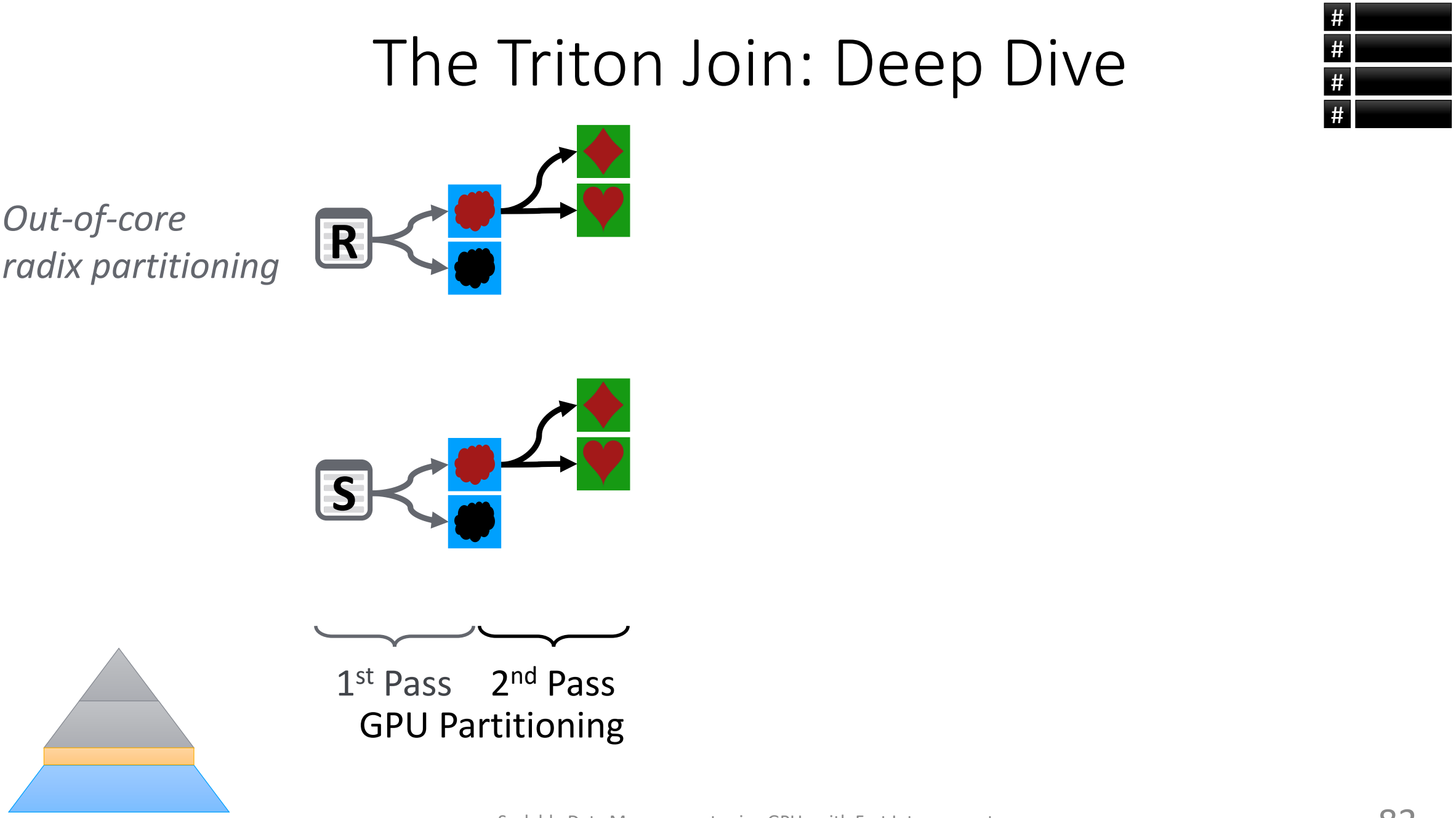




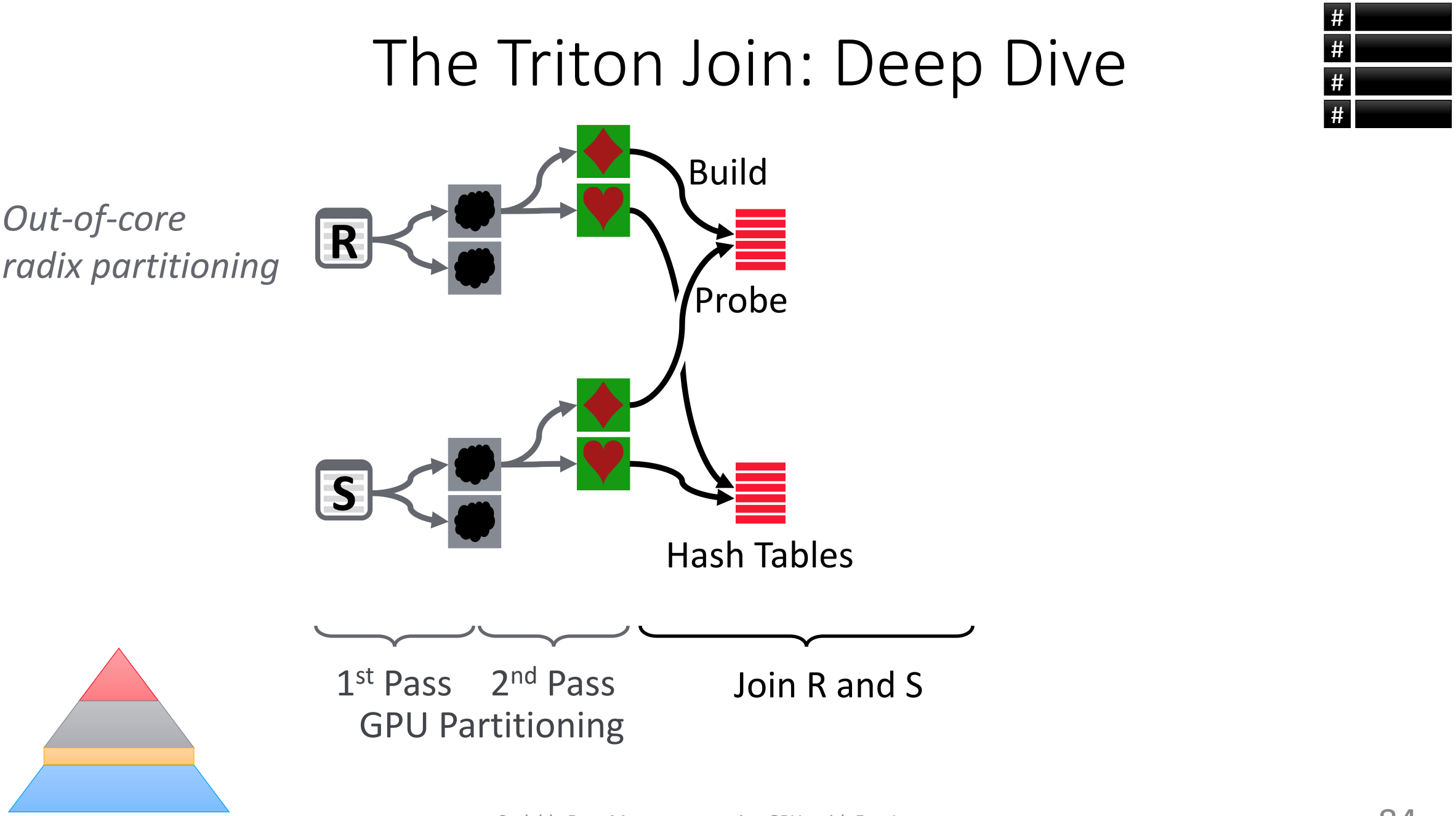
1st Pass
GPU Partitioning



The Triton Join: Deep Dive

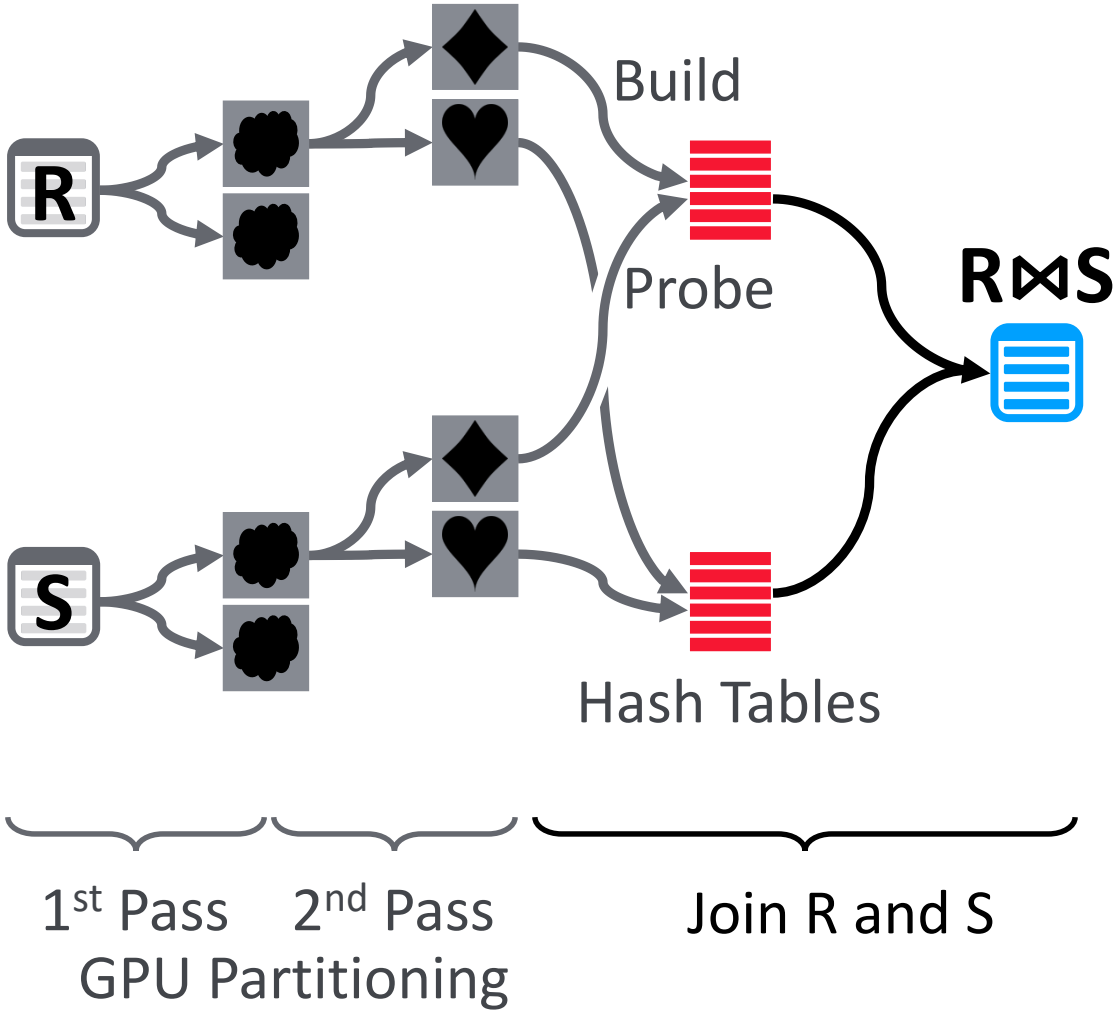


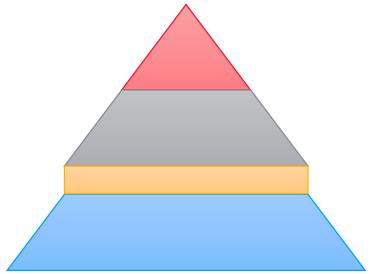
The Triton Join: Deep Dive



The Triton Join: Deep Dive

Out-of-core radix partitioning

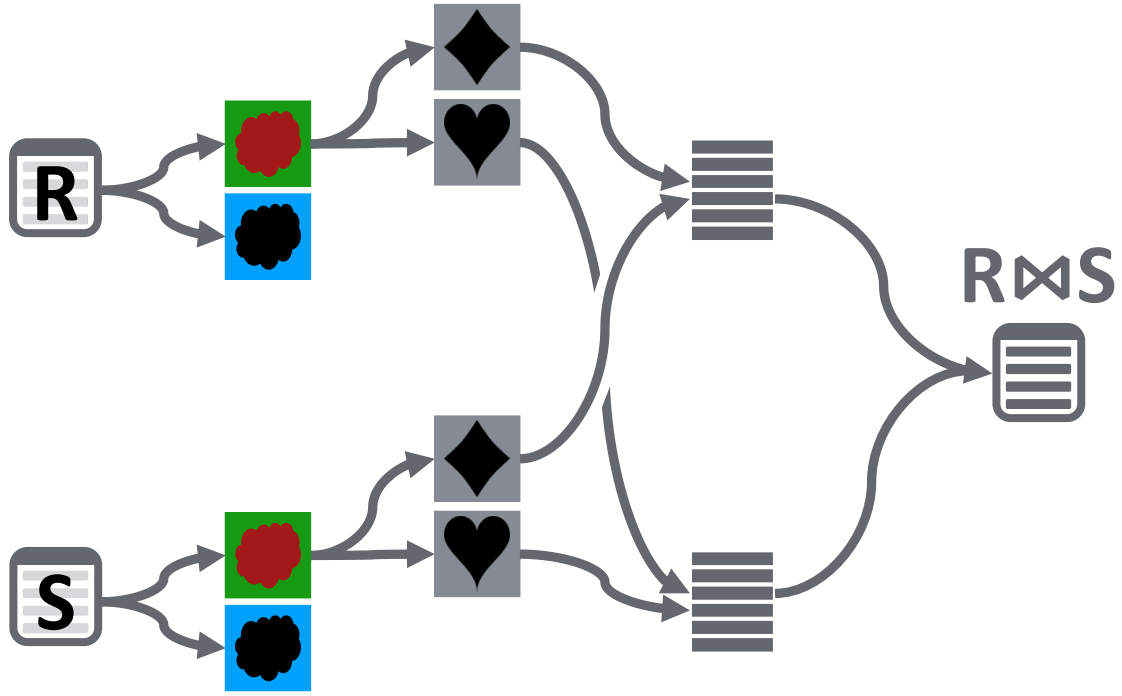


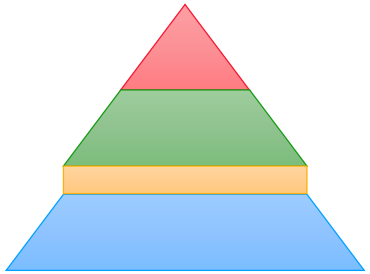


The Triton Join: Deep Dive

Out-of-core radix partitioning

Caching partitions in GPU memory





1st Pass GPU Partitioning

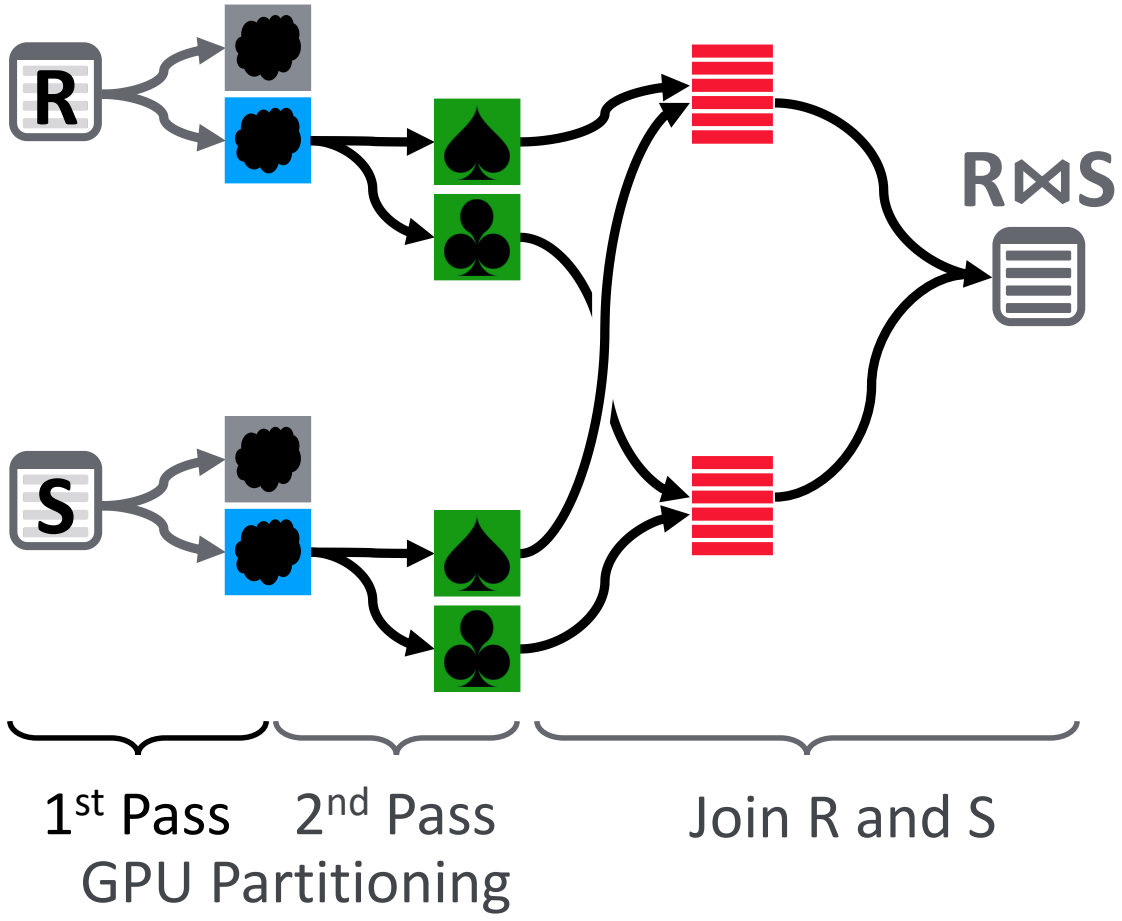
2nd Pass

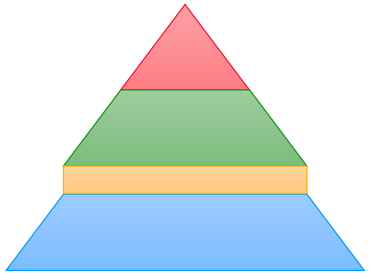
Join R and S

The Triton Join: Deep Dive

Out-of-core radix partitioning

Caching partitions in GPU memory

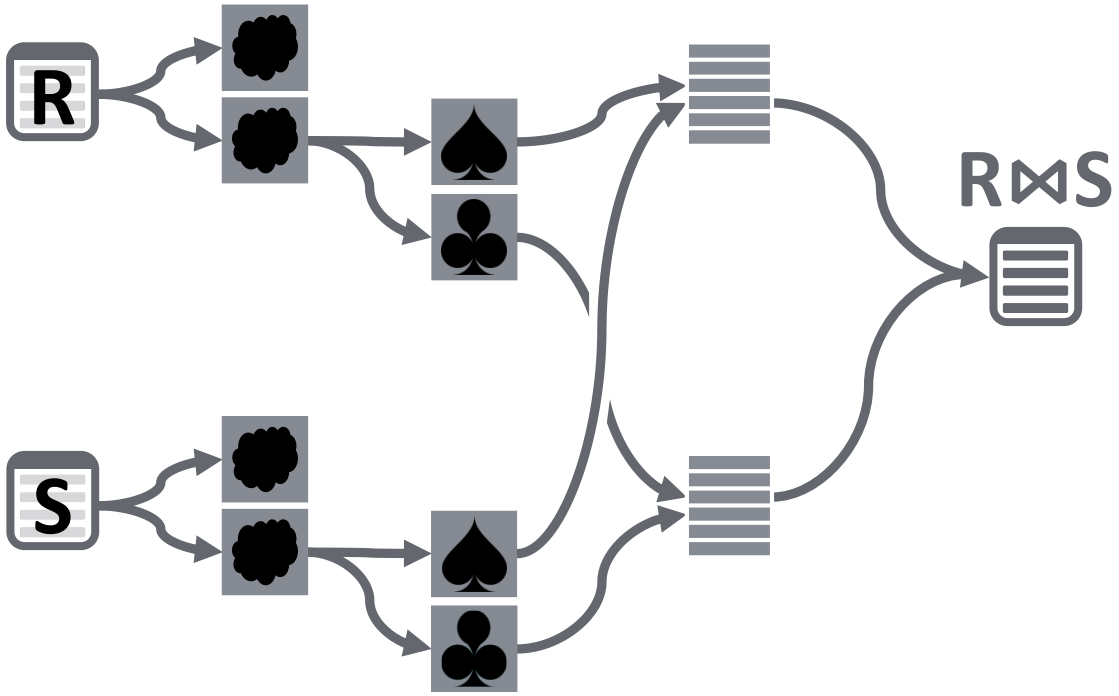




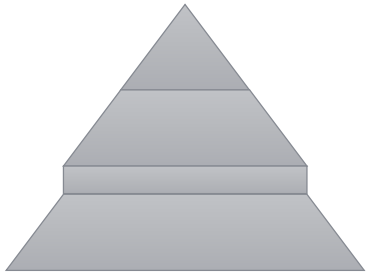
The Triton Join: Deep Dive

Out-of-core radix partitioning

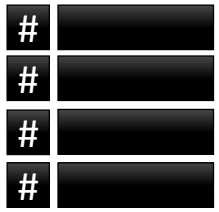
Caching partitions in GPU memory



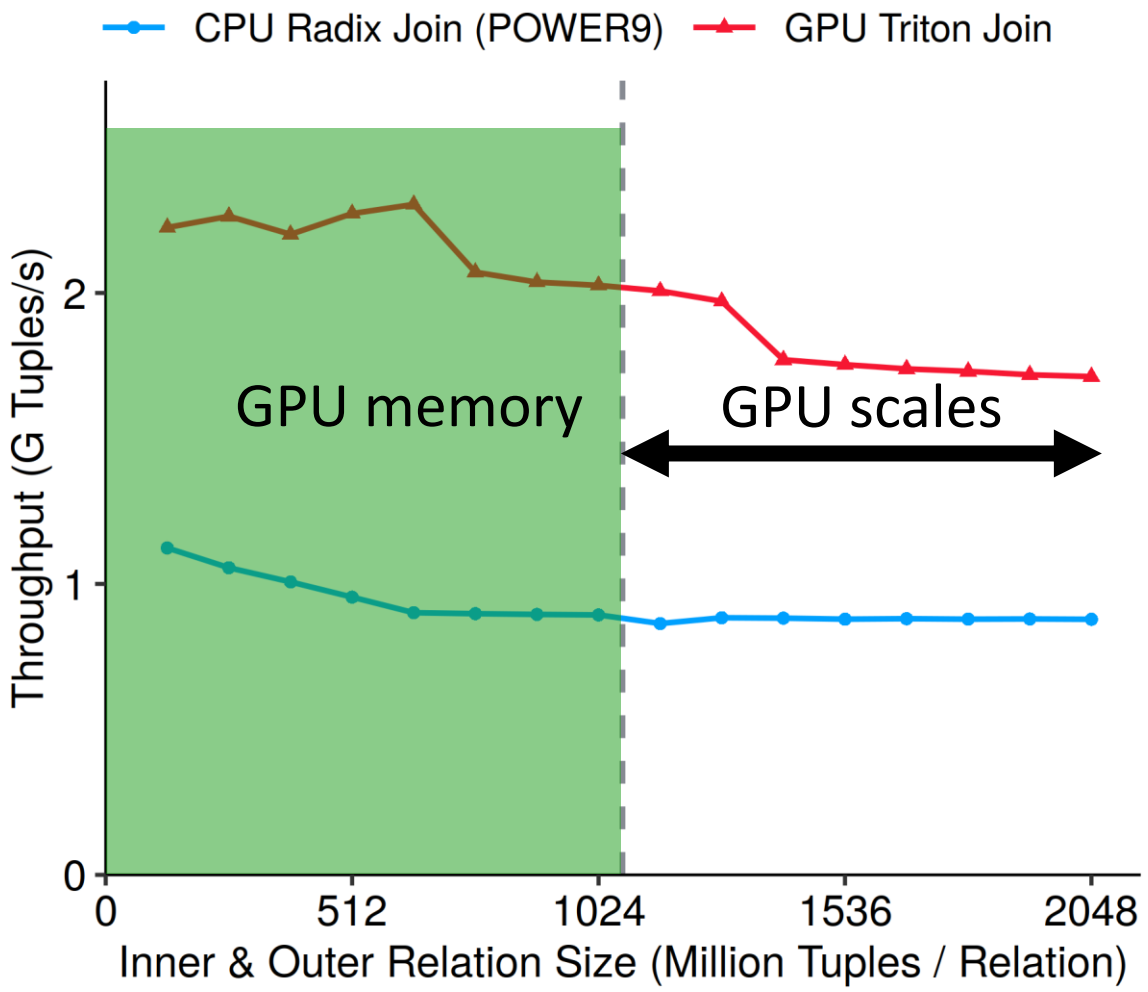
Triton join is new hierarchical hybrid hash join for GPUs



Triton Join Performance



Data: 30 GiB ⋈ 30 GiB
CPU: IBM POWER9
with 16 cores
GPU: Nvidia V100
with NVLink 2.0

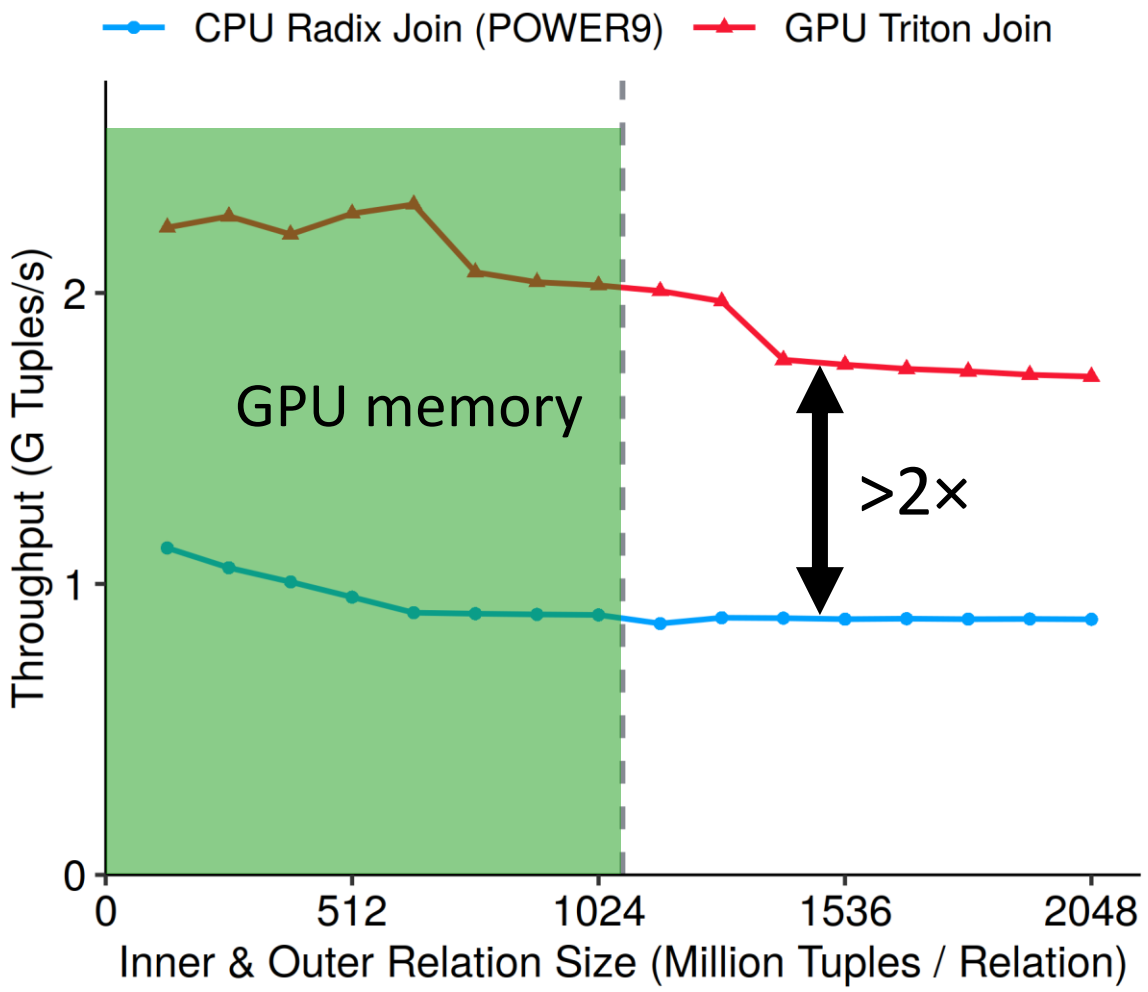


Triton join is new hierarchical hybrid hash join for GPUs

- Scalable ✓

Triton Join Performance

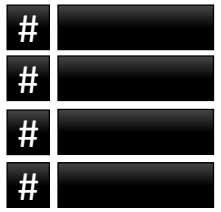
Data: 30 GiB ⋈ 30 GiB
CPU: IBM POWER9
with 16 cores
GPU: Nvidia V100
with NVLink 2.0



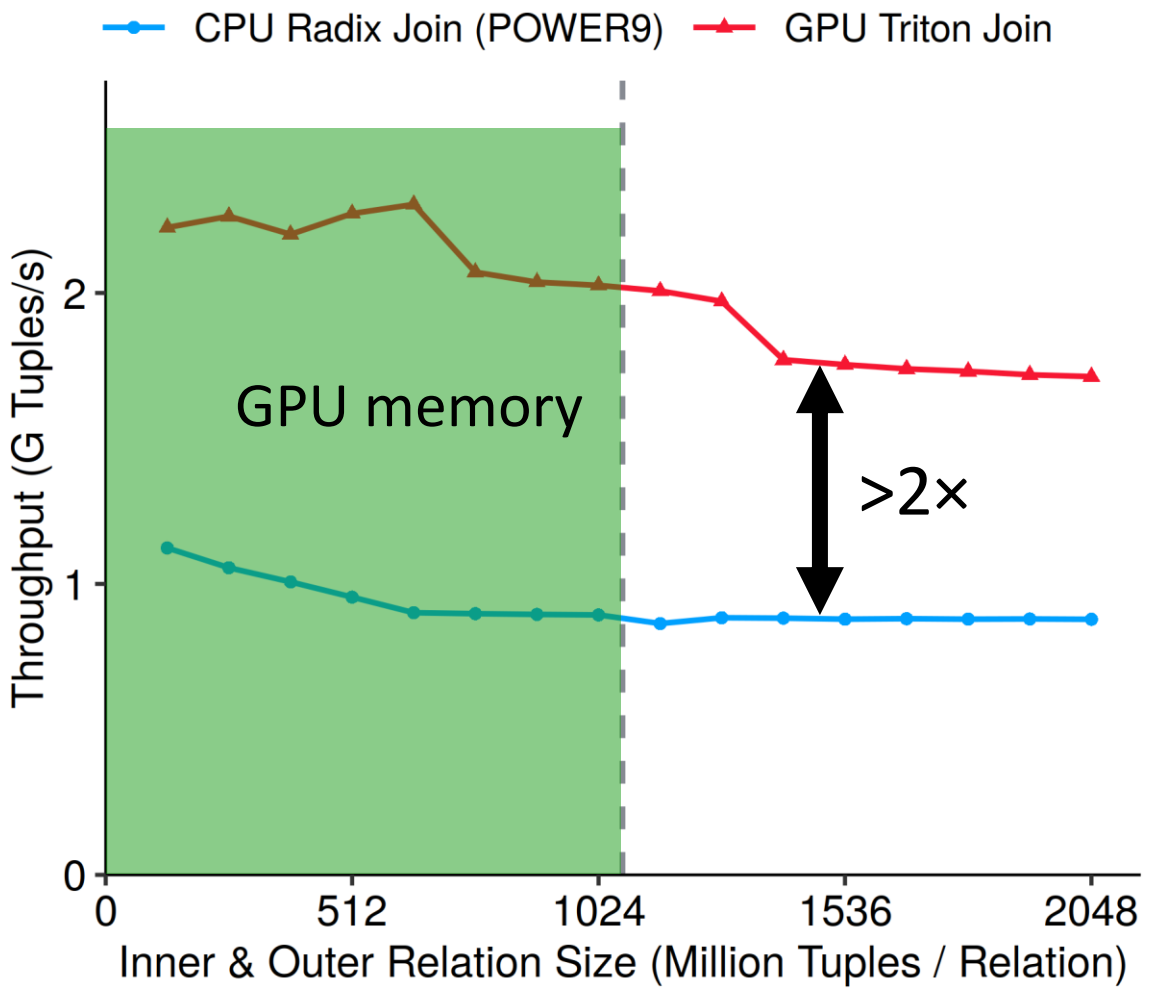
Triton join is new hierarchical hybrid hash join for GPUs

- Scalable ✓
- Efficient ✓

Triton Join Performance



Data: 30 GiB ⋈ 30 GiB
CPU: IBM POWER9
with 16 cores
GPU: Nvidia V100
with NVLink 2.0

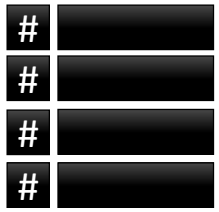


Triton join is new hierarchical hybrid hash join for GPUs

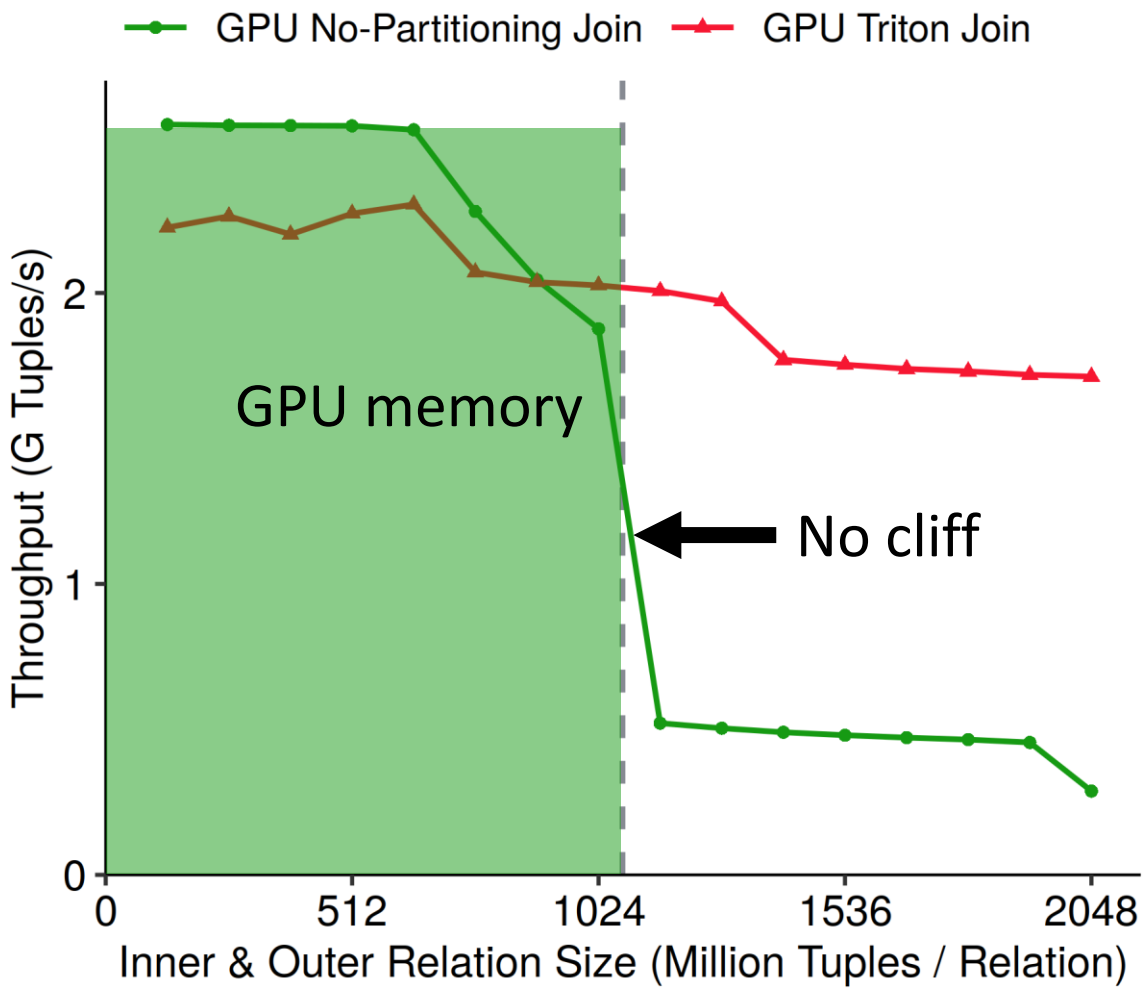
Efficiently scales to large out-of-core join state

- Scalable ✓
- Efficient ✓

Triton Join Performance



Data: 30 GiB ⋈ 30 GiB
CPU: IBM POWER9
with 16 cores
GPU: Nvidia V100
with NVLink 2.0

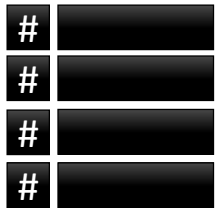


Triton join is new hierarchical hybrid hash join for GPUs

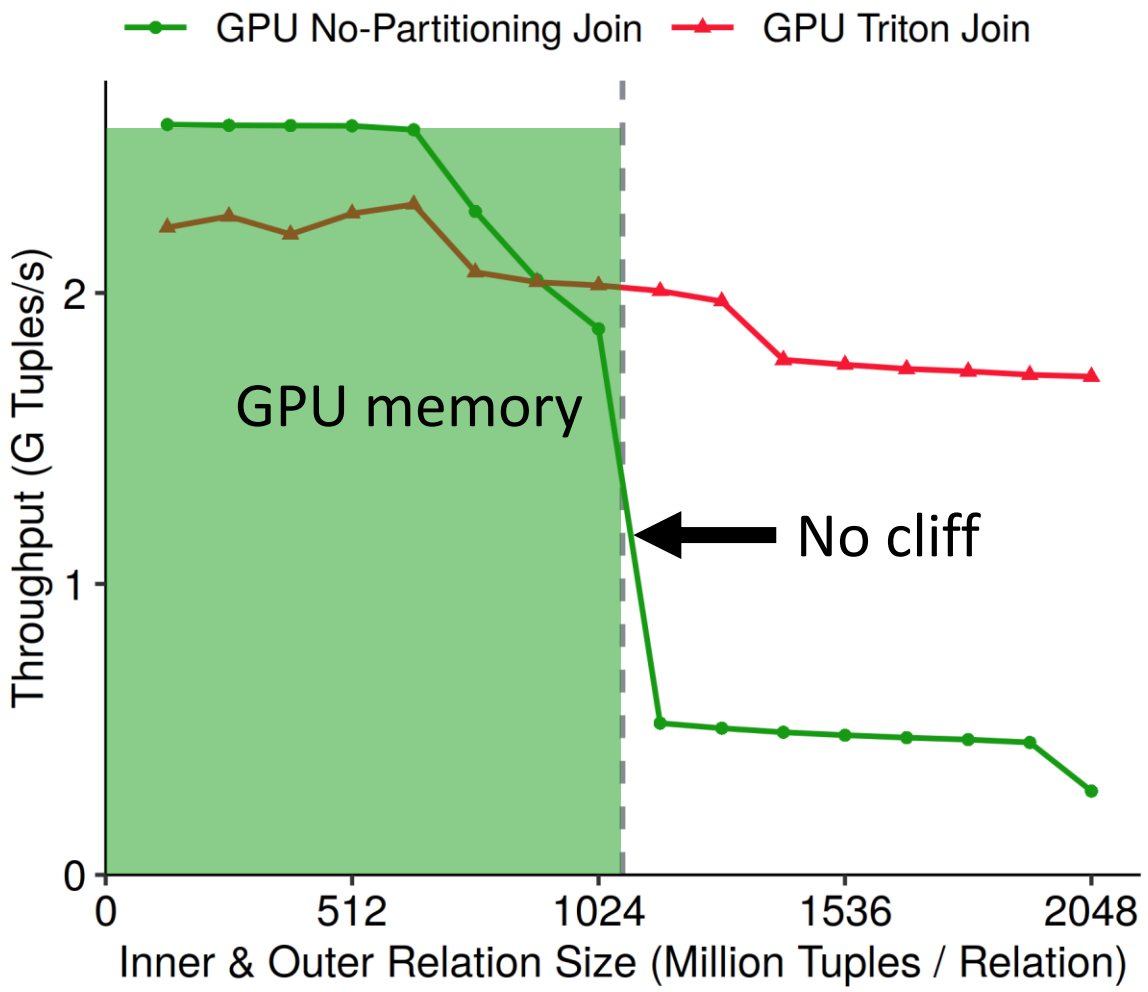
Efficiently scales to large out-of-core join state

- Scalable ✓
- Efficient ✓

Triton Join Performance



Data: 30 GiB ⋈ 30 GiB
CPU: IBM POWER9
with 16 cores
GPU: Nvidia V100
with NVLink 2.0



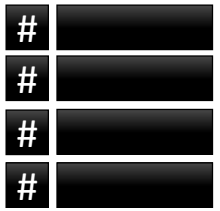
- Scalable ✓
- Efficient ✓

Triton join is new hierarchical hybrid hash join for GPUs

Efficiently scales to large out-of-core join state

Performance degrades gracefully

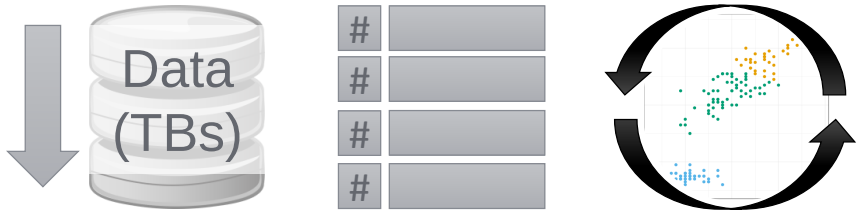
Findings Summary



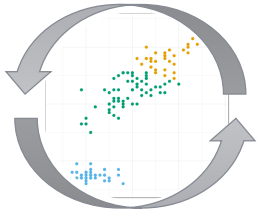
- Interconnect-conscious state access
  - Perfect coalescing
- Scalable join algorithm
  - Out-of-core partitioning
  - 2× speedup

Agenda

1. Motivation
2. Data-intensive query processing
3. Stateful data processing
4. **Iterative algorithms**
5. Conclusion

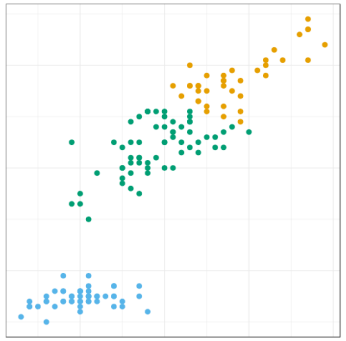


How *k*-Means Works

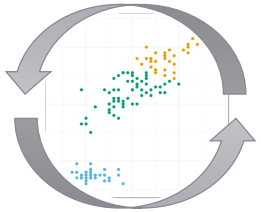


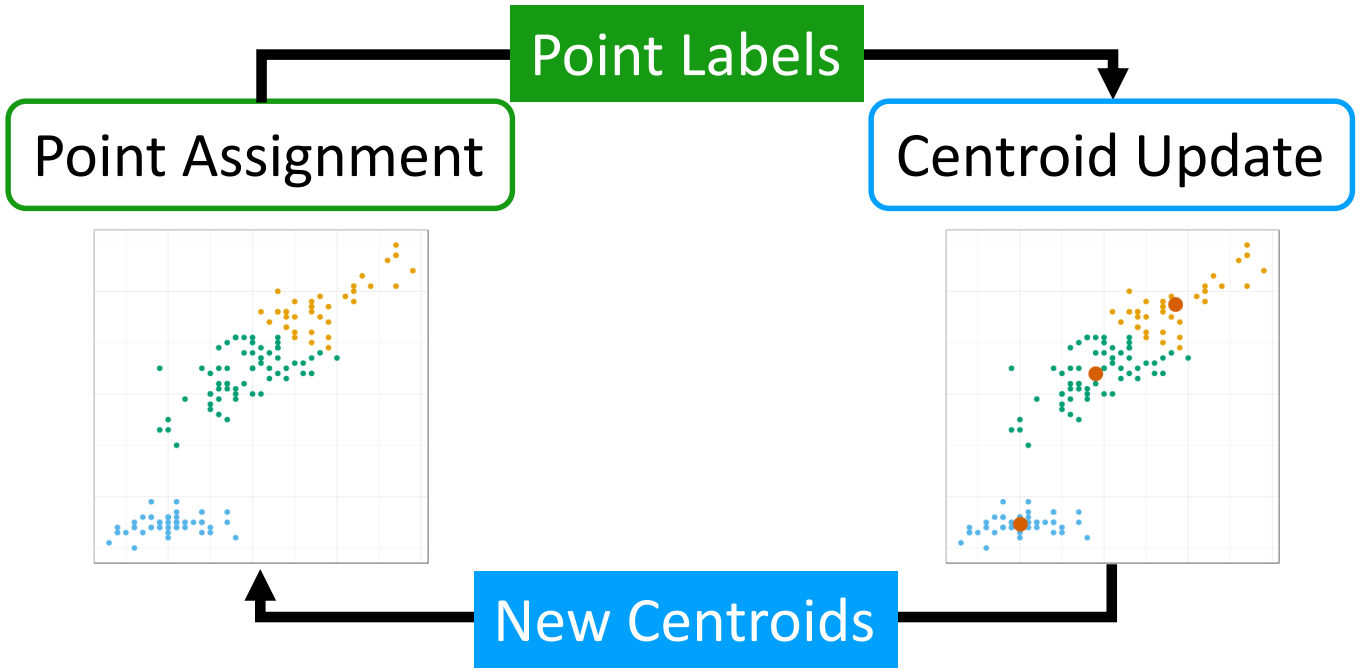
Point Labels

Point Assignment

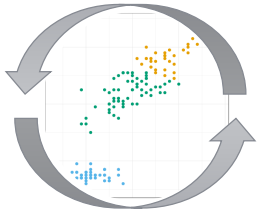


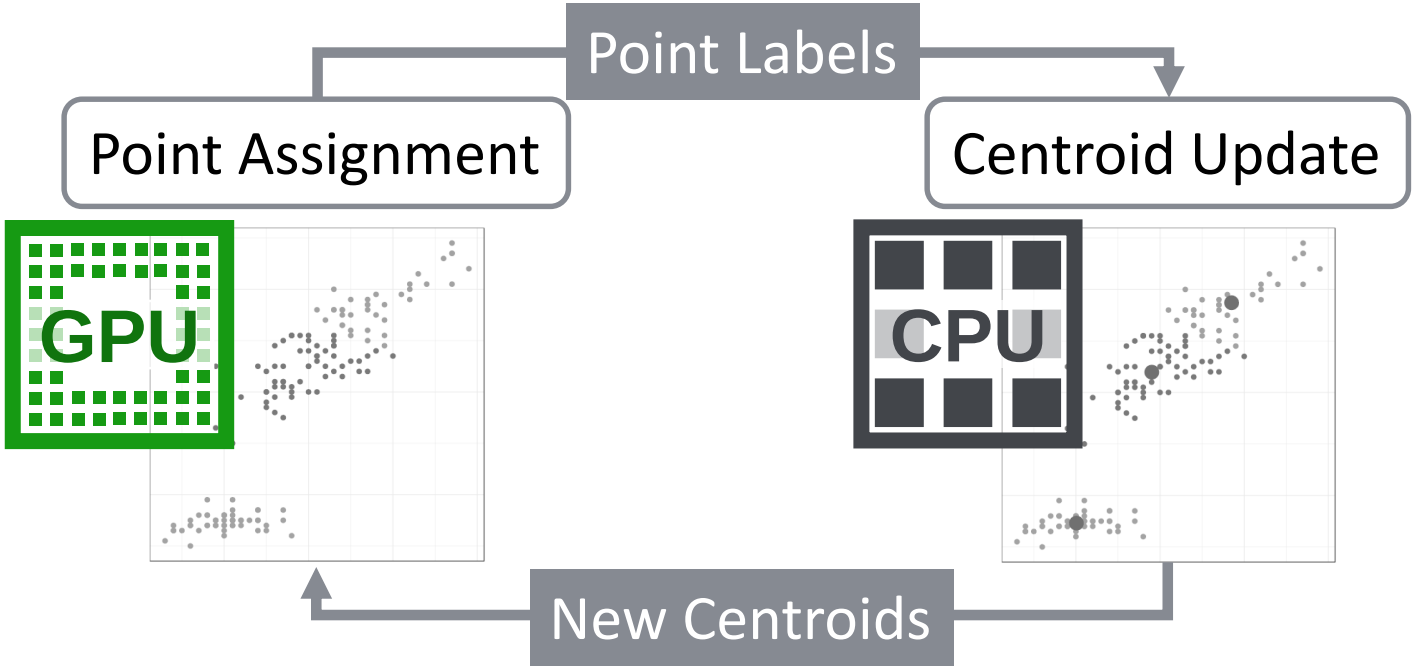
How *k*-Means Works



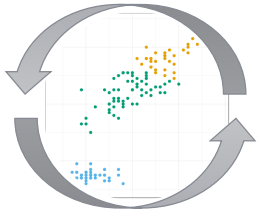


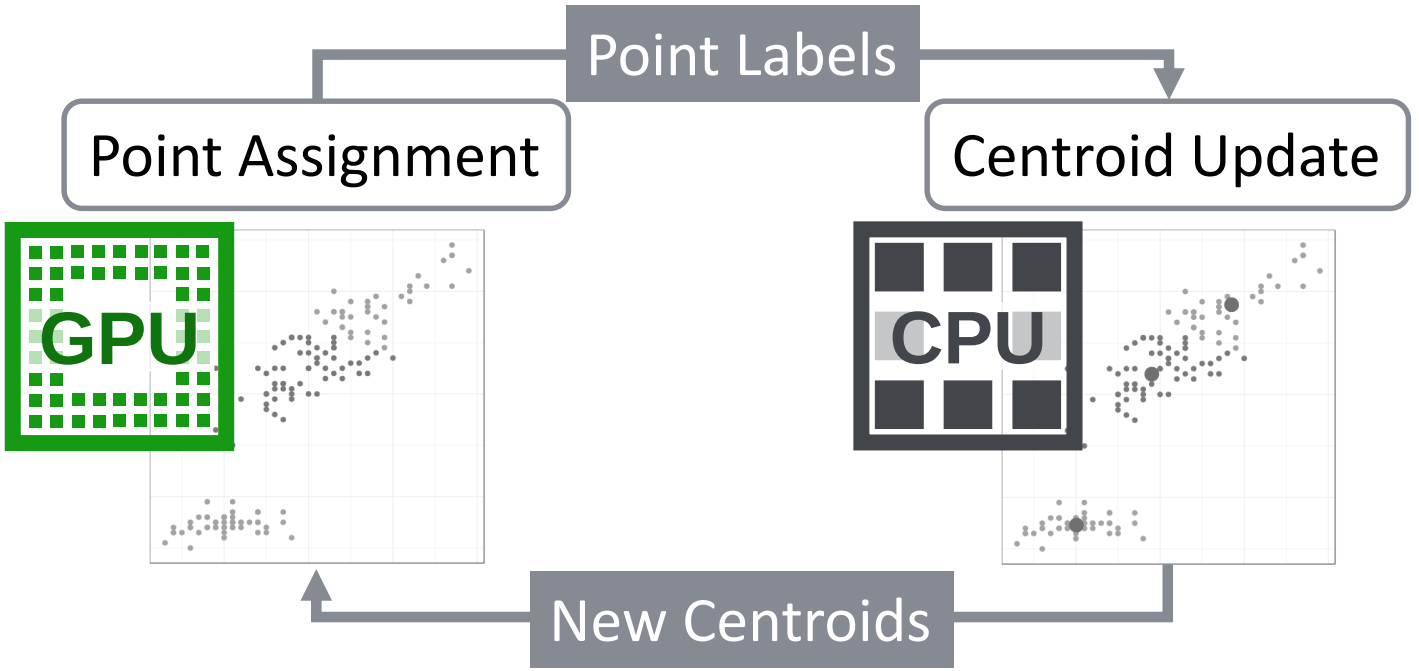
State-of-the-Art *k*-Means Strategy





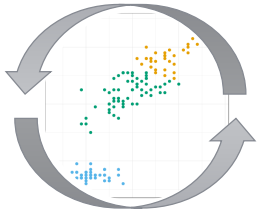
State-of-the-Art *k*-Means Strategy



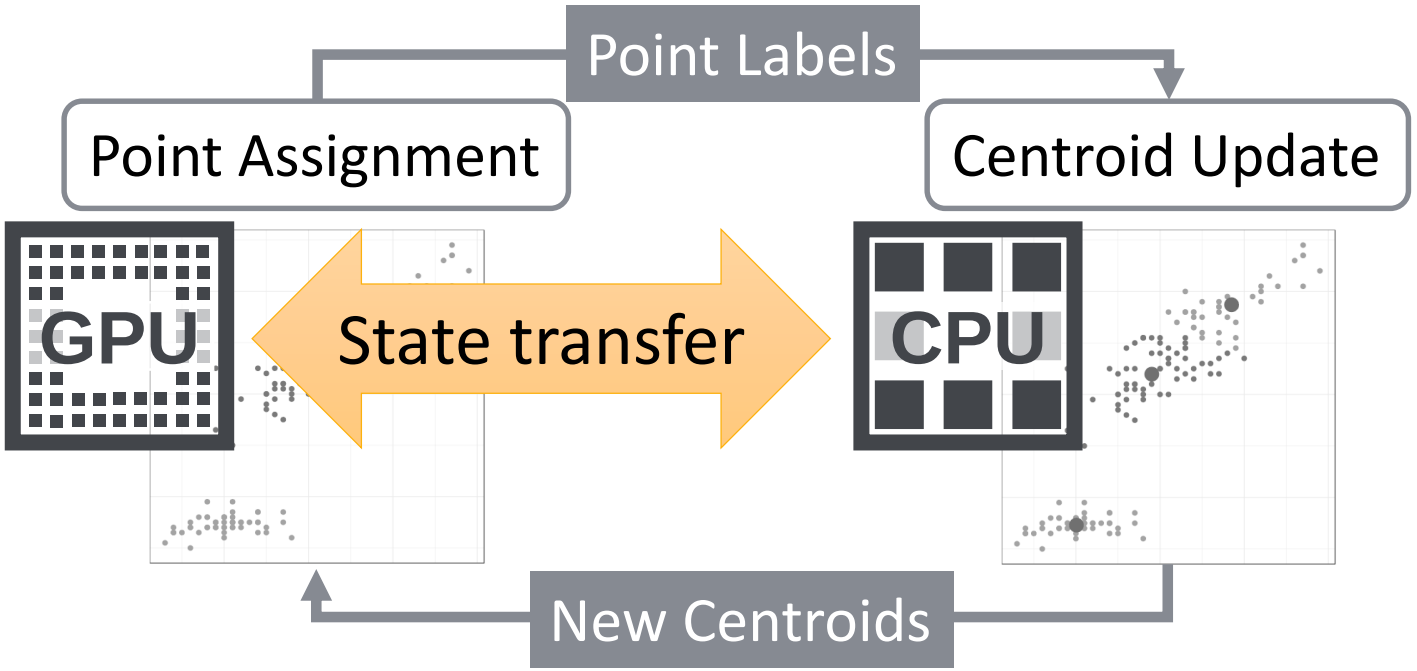


Offload compute intensive task to GPU

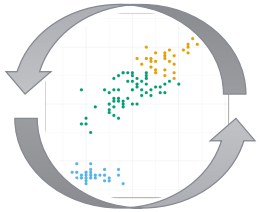
State-of-the-Art *k*-Means Strategy



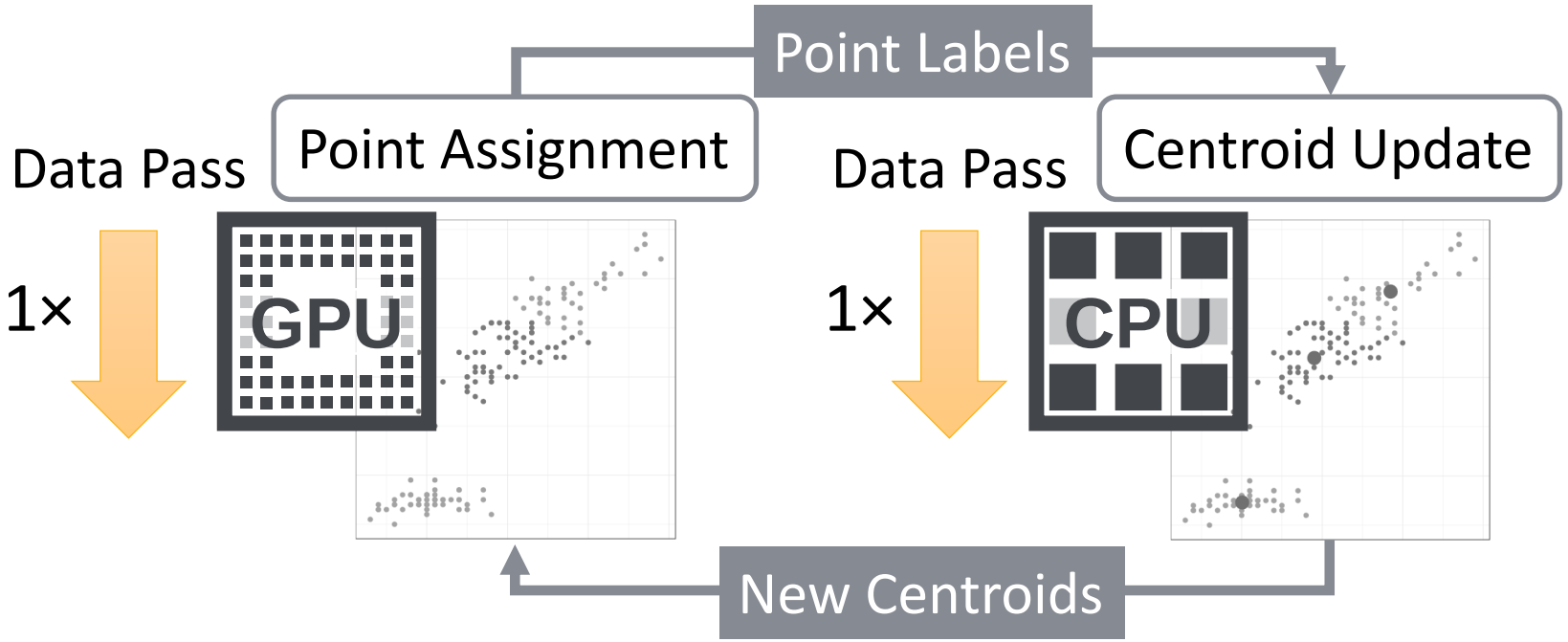
Offload compute intensive task to GPU



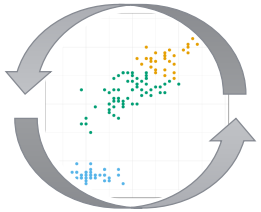
State-of-the-Art *k*-Means Strategy

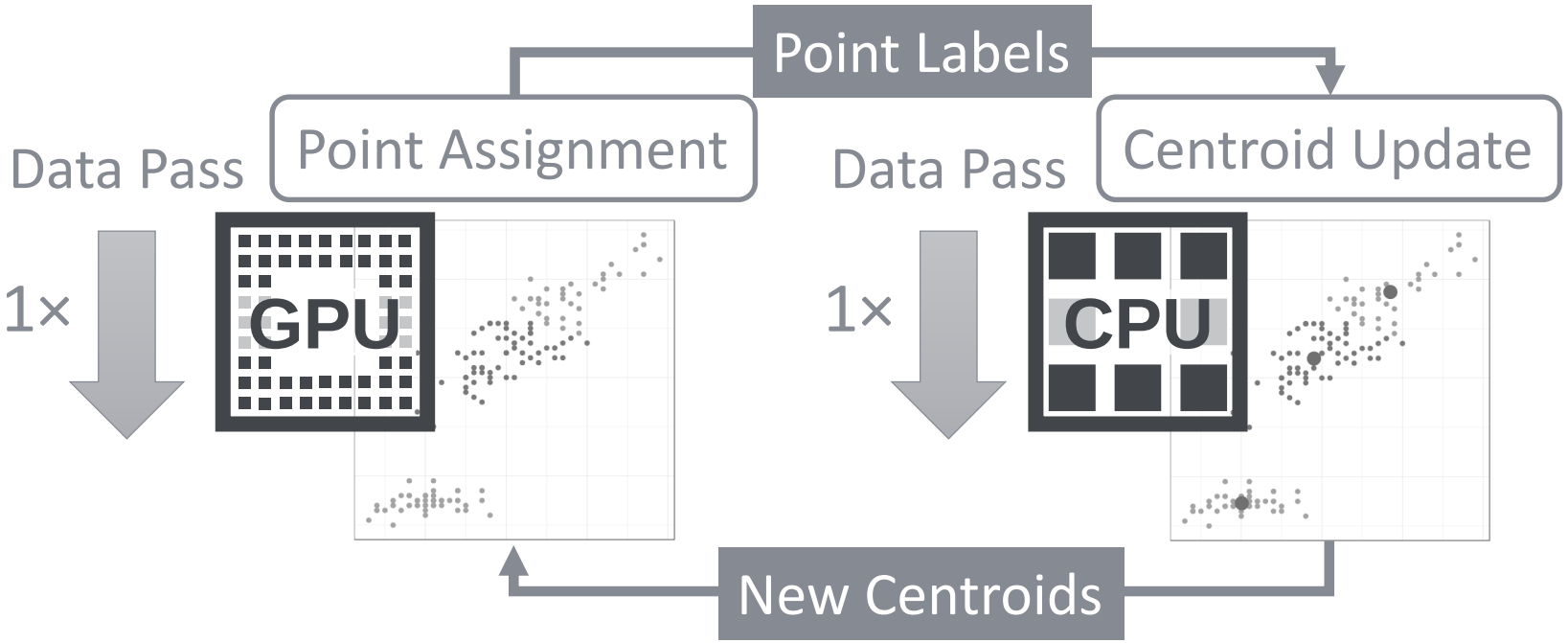


Offload compute intensive task to GPU



State-of-the-Art *k*-Means Strategy

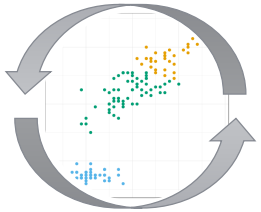


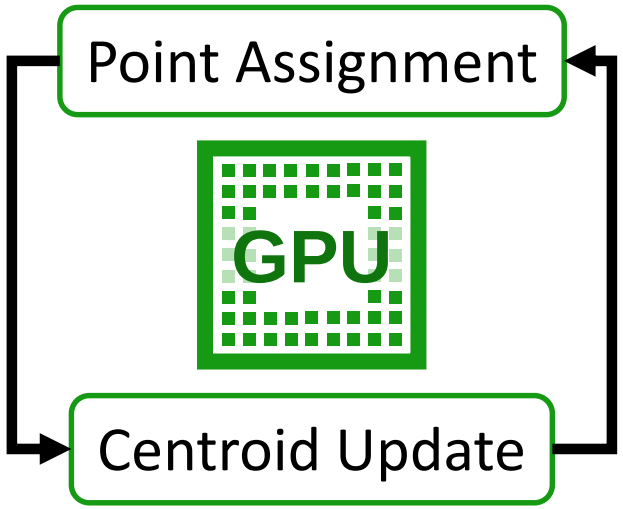


Offload compute intensive task to GPU

Overhead for state transfer and data passes

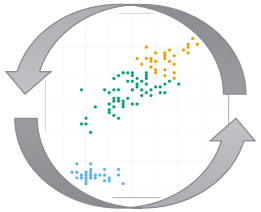
Single-Pass *k*-Means Strategy

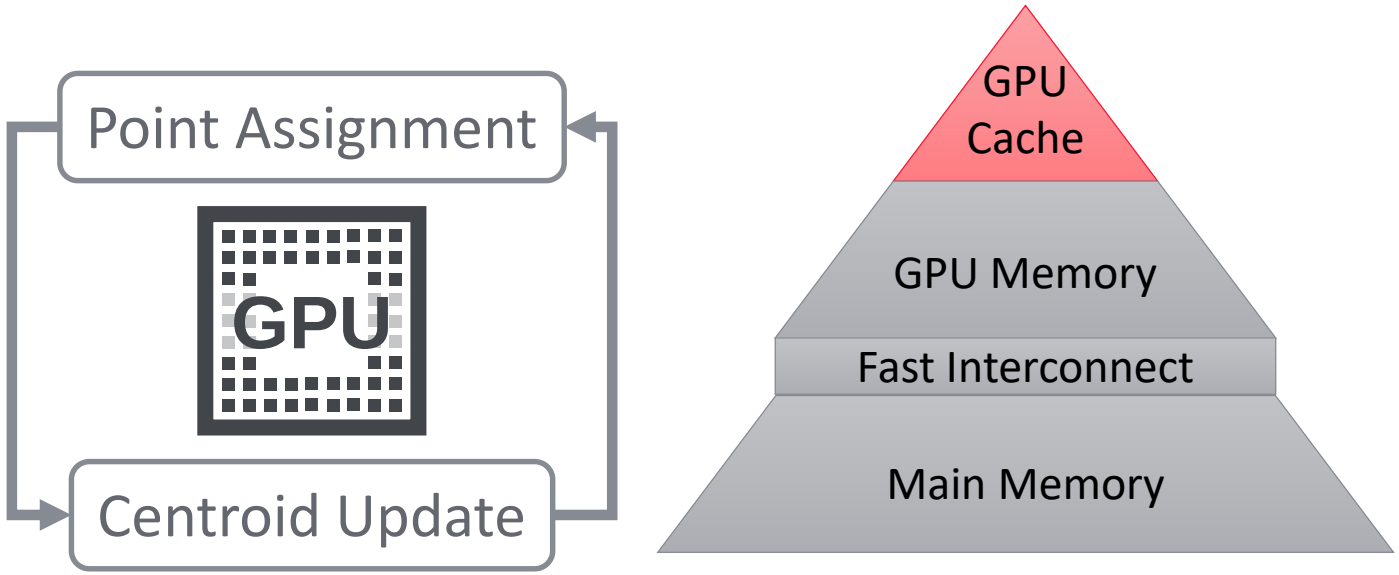




- End-to-end GPU execution
  - Centroid Update algorithm for GPU

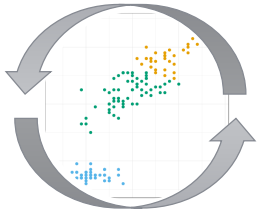
Single-Pass *k*-Means Strategy

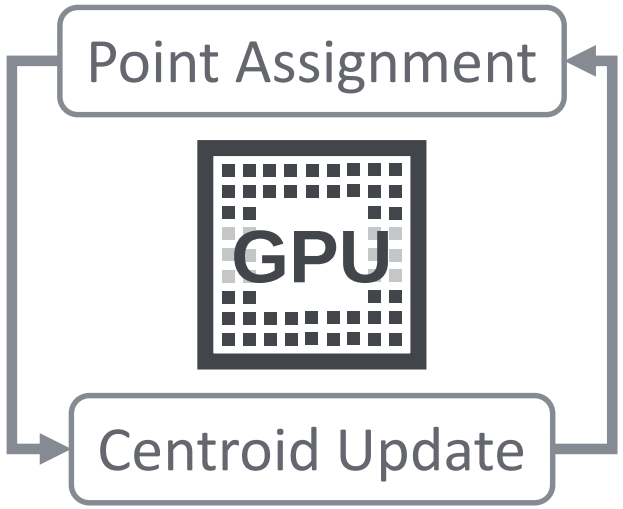


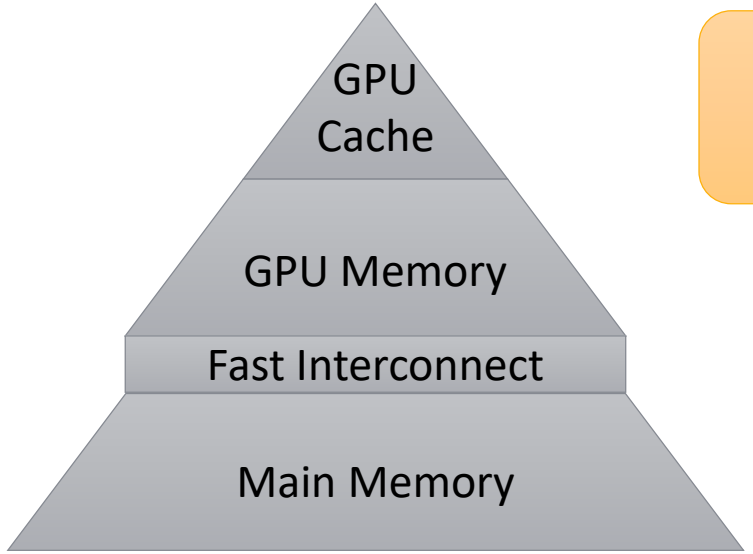


- End-to-end GPU execution
  - Centroid Update algorithm for GPU
- Increase data locality
  - Fuse phases into a single GPU kernel
  - Store state in scratchpad cache

Single-Pass *k*-Means Strategy



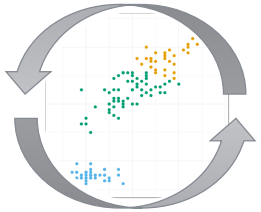


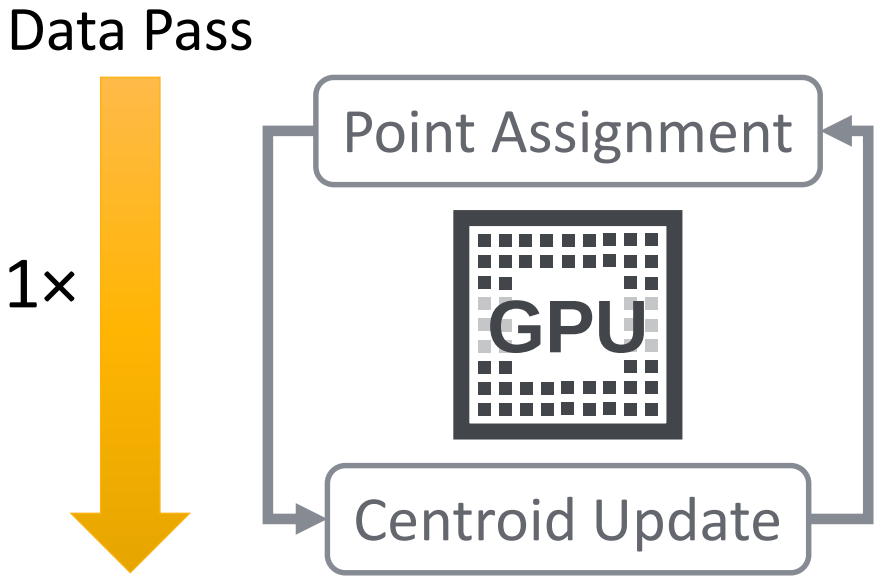


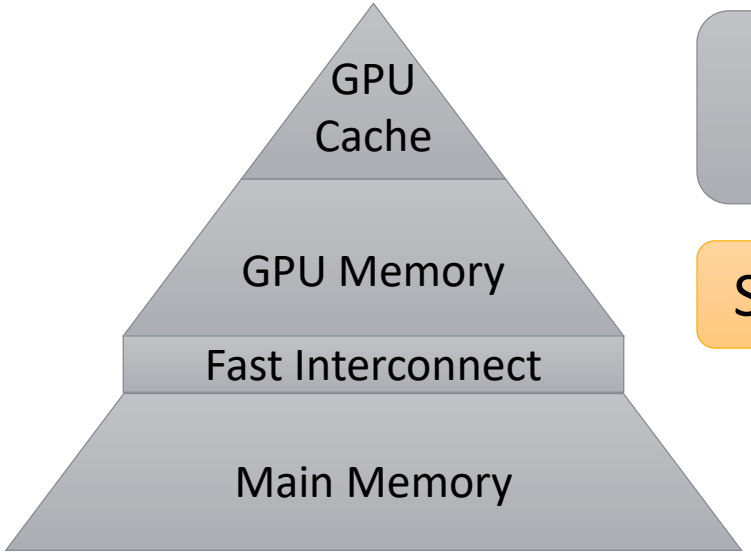
Leverage data locality with end-to-end GPU execution

- End-to-end GPU execution
  - Centroid Update algorithm for GPU
- Increase data locality
  - Fuse phases into a single GPU kernel
  - Store state in scratchpad cache

Single-Pass *k*-Means Strategy







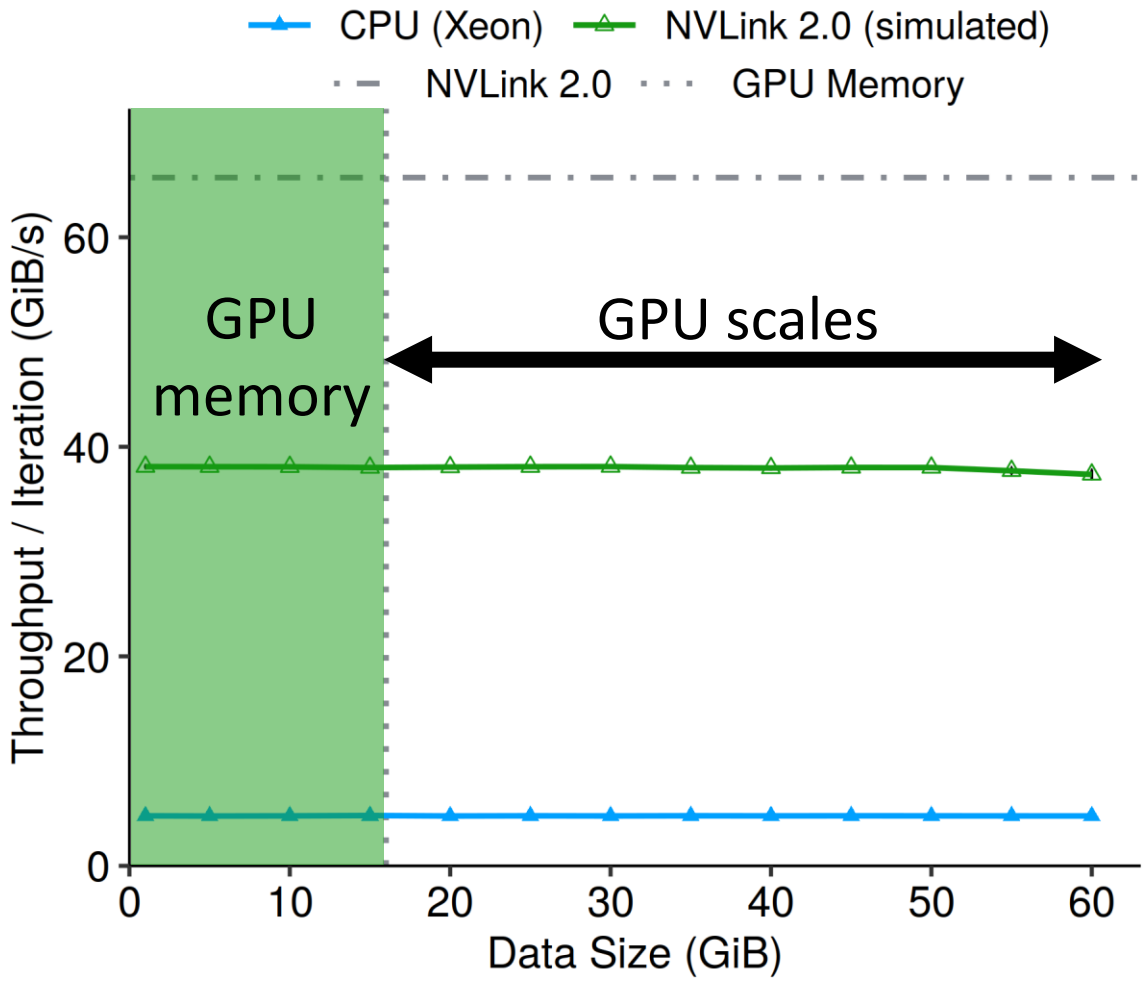
Leverage data locality with end-to-end GPU execution

Single data pass per iteration

- End-to-end GPU execution
  - Centroid Update algorithm for GPU
- Increase data locality
  - Fuse phases into a single GPU kernel
  - Store state in scratchpad cache

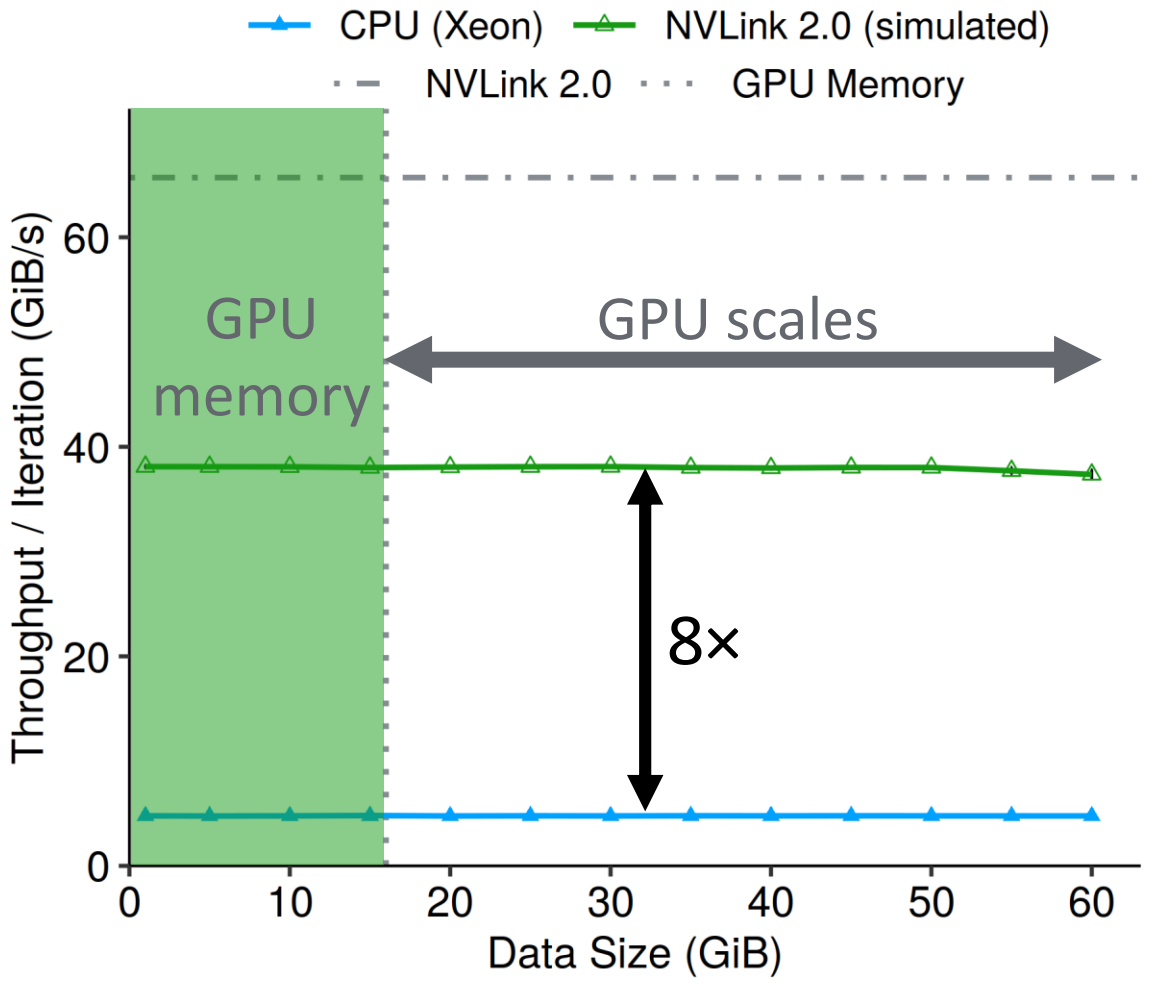
Single-Pass *k*-Means Scalability

Clusters: 64
Features: 4
CPU: Intel Xeon
with 12 cores
GPU: Nvidia V100



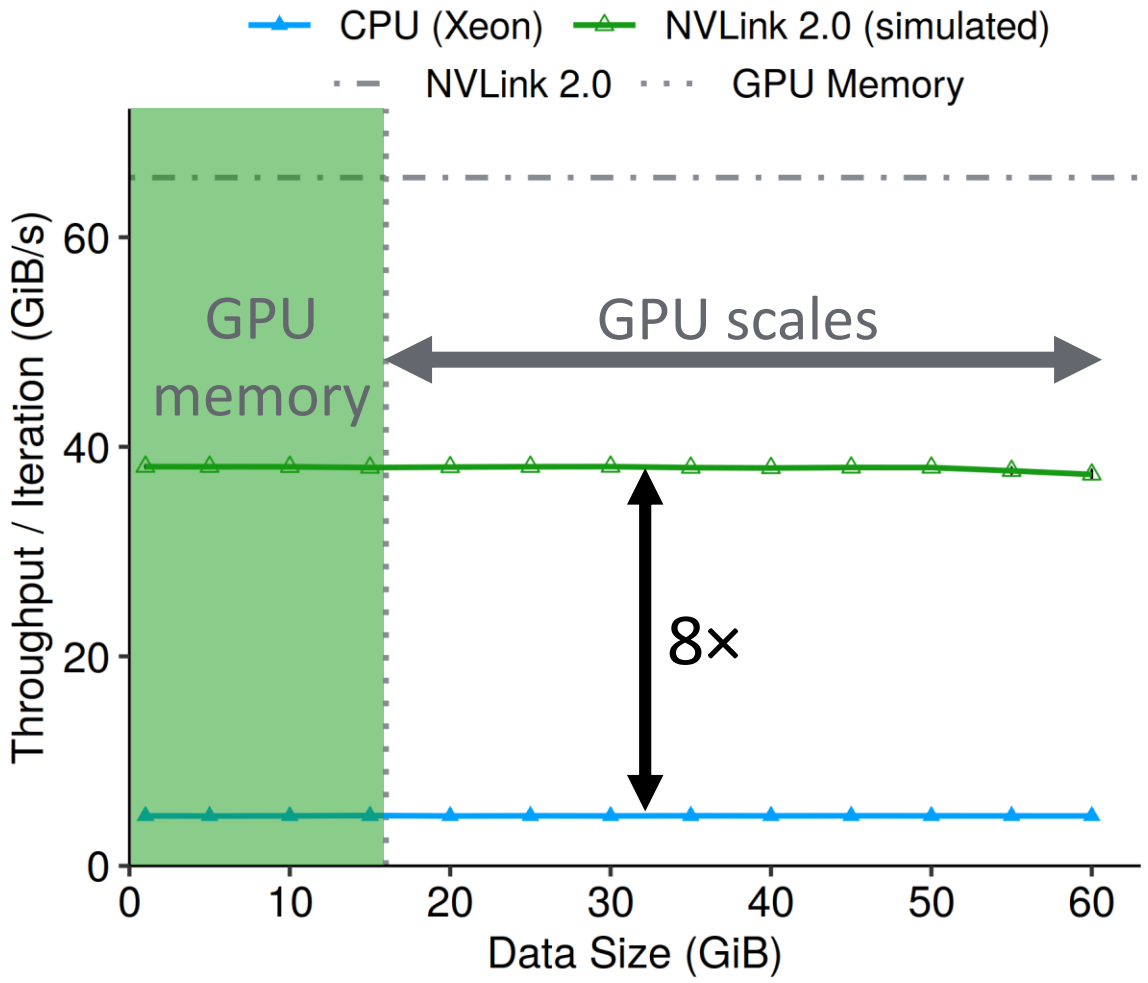
Single-Pass *k*-Means Scalability

Clusters: 64
Features: 4
CPU: Intel Xeon
with 12 cores
GPU: Nvidia V100



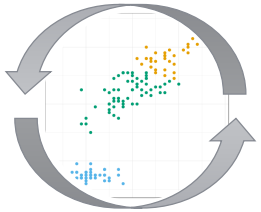
Single-Pass *k*-Means Scalability

Clusters: 64
Features: 4
CPU: Intel Xeon
with 12 cores
GPU: Nvidia V100



GPU efficiently iterates over large working sets

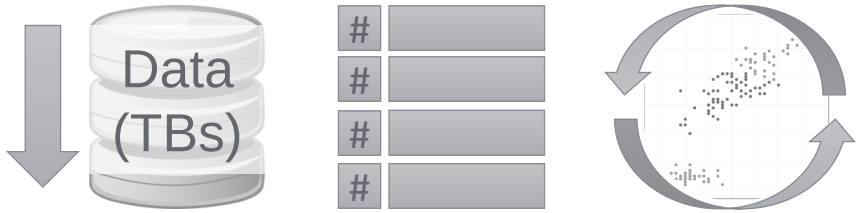
Findings Summary



- Interconnect-conscious data locality
- End-to-end GPU execution
  - Single data pass
  - 8× speedup

Agenda

1. Motivation
2. Data-intensive query processing
3. Stateful data processing
4. Iterative algorithms
5. **Conclusion**



Conclusion

- Scalable data management using GPUs

Conclusion

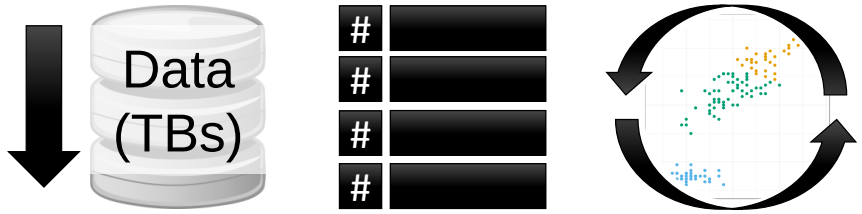
- Scalable data management using GPUs
- Fast interconnect is **necessary, but not sufficient**

Conclusion

- Scalable data management using GPUs
- Fast interconnect is necessary, but not sufficient
- Interconnect-conscious design
  - Data access ➡ Coherence
  - State access ➡ Perfect coalescing
  - Data locality ➡ End-to-end GPU execution

Conclusion

- Scalable data management using GPUs
- Fast interconnect is necessary, but not sufficient
- Interconnect-conscious design
  - Data access ➡ Coherence
  - State access ➡ Perfect coalescing
  - Data locality ➡ End-to-end GPU execution
- Overall, efficient out-of-core algorithms



PhD Thesis Publications

Fast Interconnects

SIGMOD 2020

Triton Join

SIGMOD 2022

Single-Pass *k*-Means

DaMoN 2018

DB Spektrum 2018

Additional Contributions

Data Loading using GPUs

BTW 2021

Efficient Stream Processing

PVLDB 2019

Energy-Efficient Stream Join

DaMoN 2021

Awards

Best Paper Award
& Reproducibility Badge

SIGMOD 2020

Best Paper Award
& Reproducibility Badge

BTW 2021

Conclusion

- Scalable data management using GPUs
- Fast interconnect is **necessary, but not sufficient**
- Interconnect-conscious design
  - Data access ➡ Coherence
  - State access ➡ Perfect coalescing
  - Data locality ➡ End-to-end GPU execution
- Overall, efficient out-of-core algorithms