Triton Join: Efficiently Scaling to a Large Join State on GPUs with Fast Interconnects

Clemens Lutz, Sebastian Breß, Steffen Zeuch, Tilmann Rabl, Volker Markl

German Research Center for Artificial Intelligence

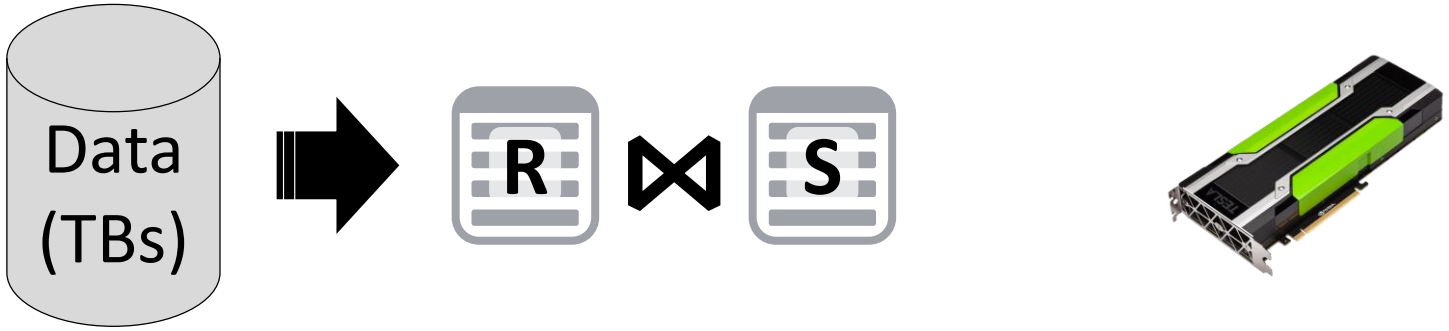
BIFOLD

Goal

*Scale GPU-accelerated joins to **a large join state**.*

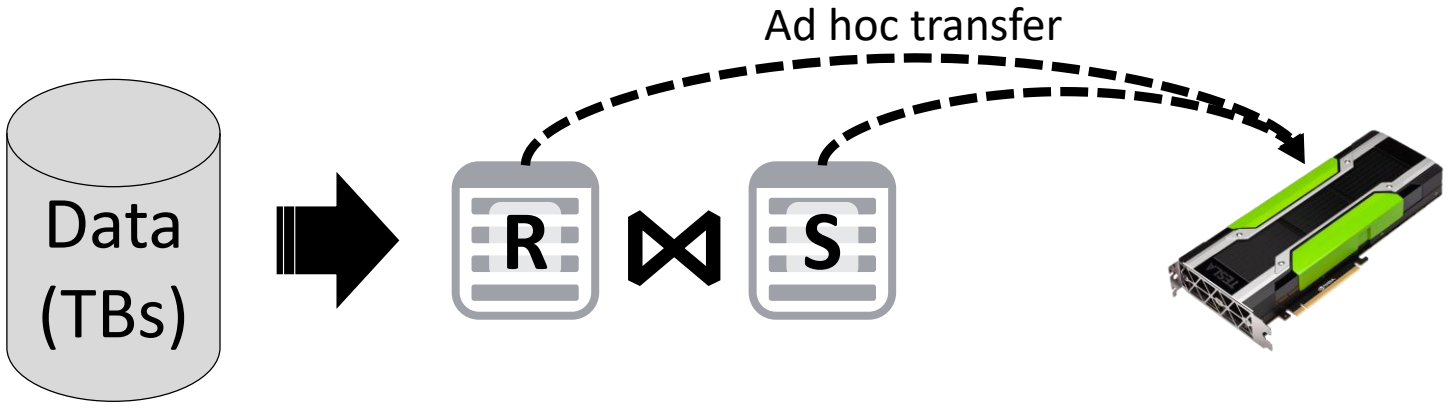
Goal

*Scale GPU-accelerated joins to **a large join state**.*



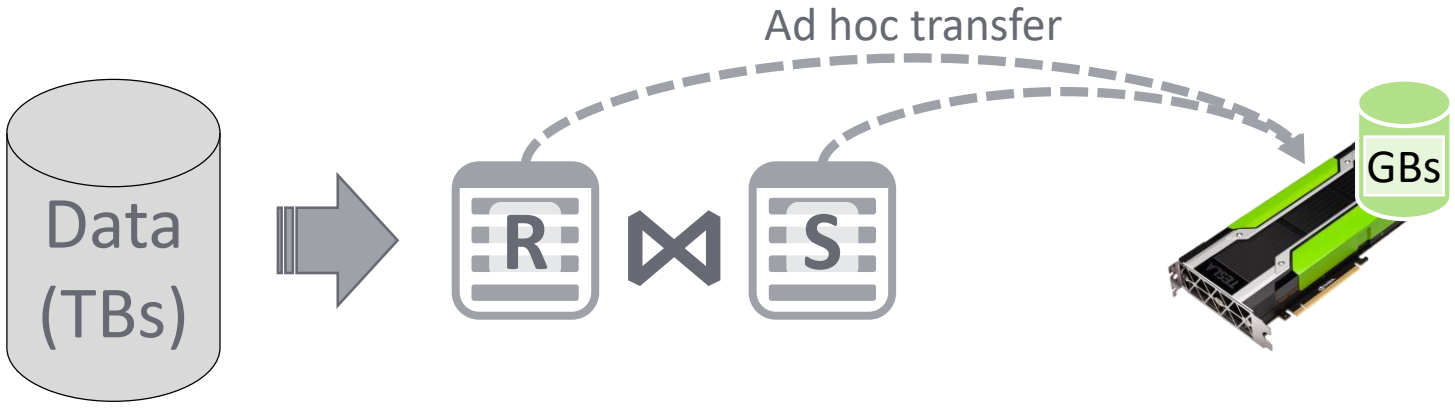
Research Problem

*Scale GPU-accelerated joins to **a large join state**.*



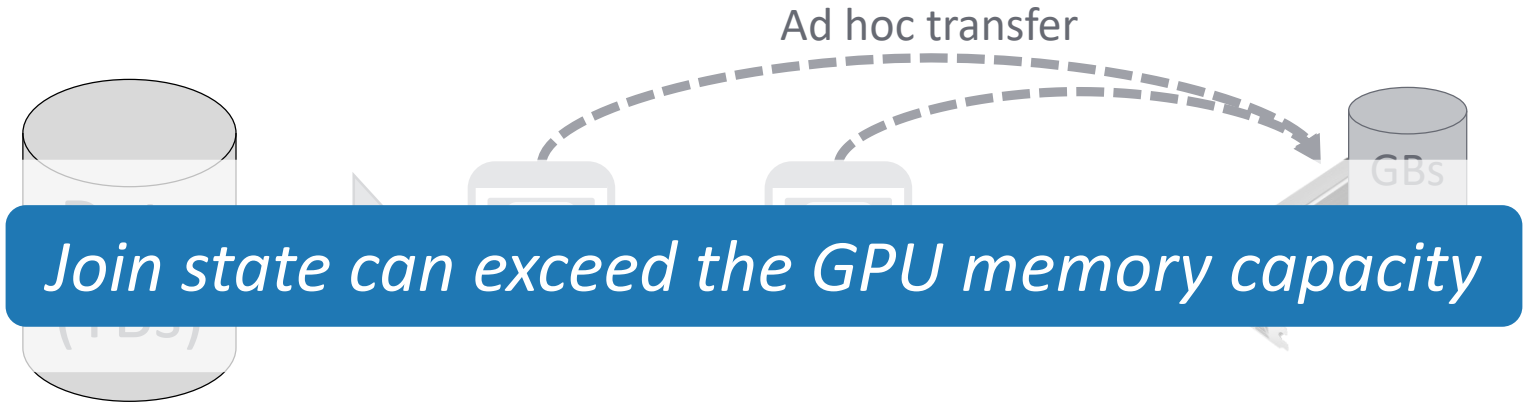
Research Problem

*Scale GPU-accelerated joins to **a large join state**.*



Research Problem

*Scale GPU-accelerated joins to **a large join state**.*



State-of-the-Art Premise

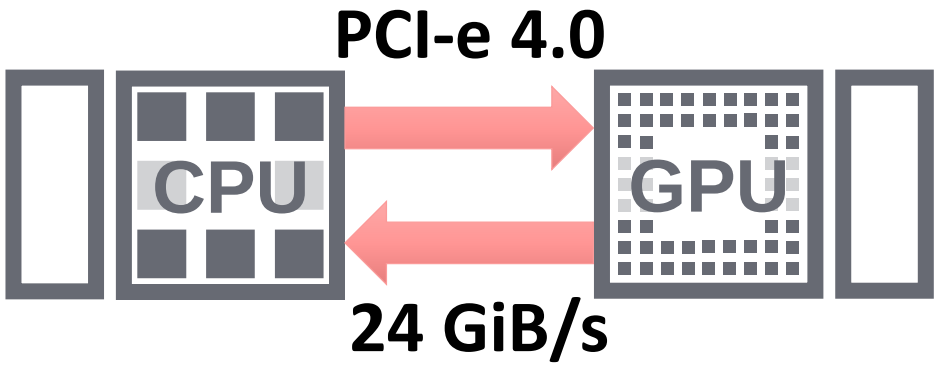
Memory capacity

Interconnect bandwidth

State-of-the-Art Premise

Memory capacity

Interconnect bandwidth



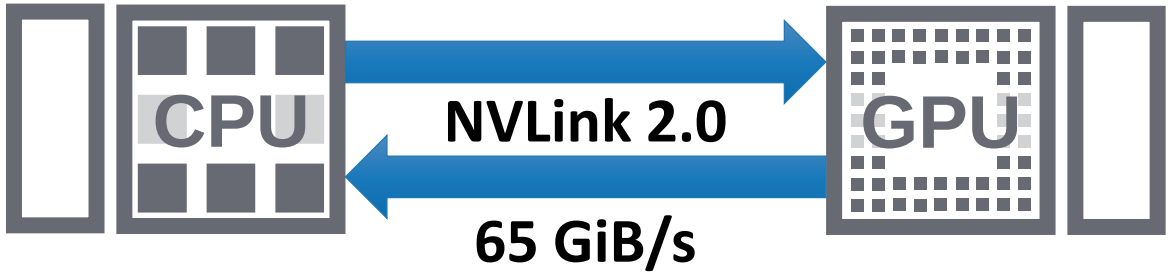
State-of-the-Art Premise

Memory capacity

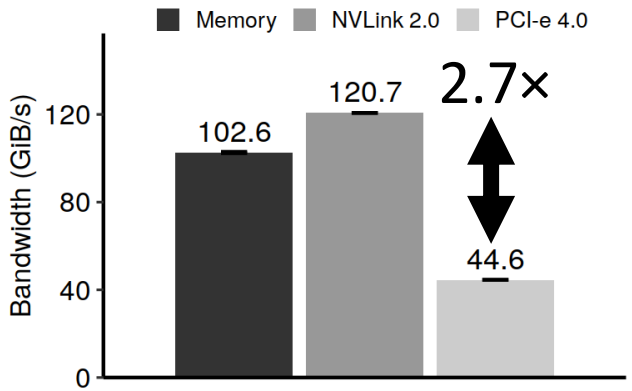
Interconnect bandwidth

Data Transfer Bottleneck

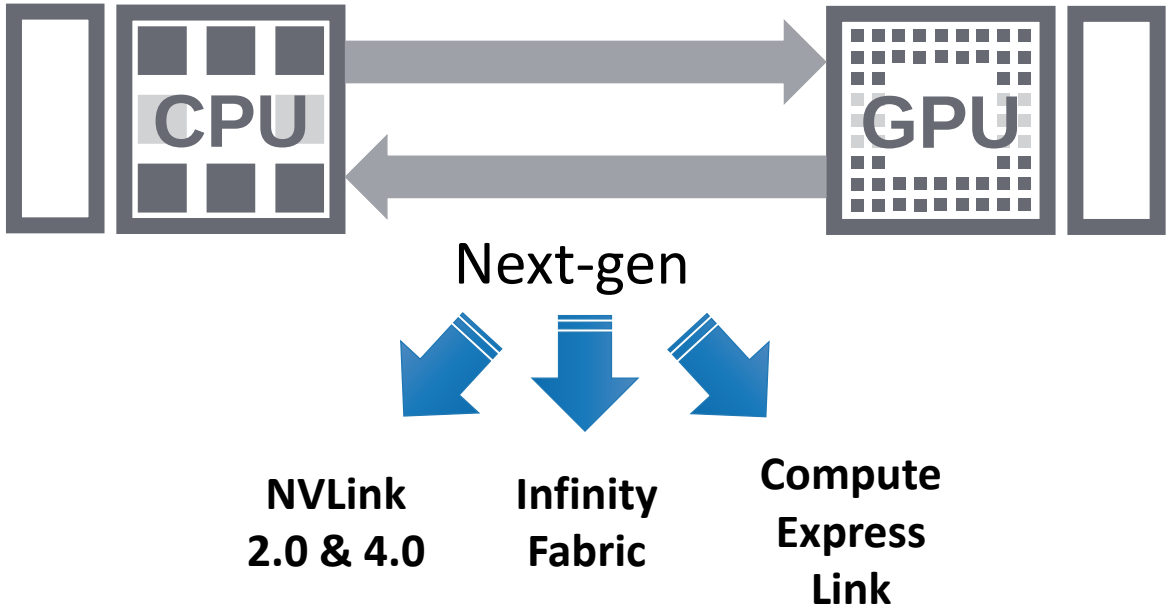
Game Changer: Fast Interconnects



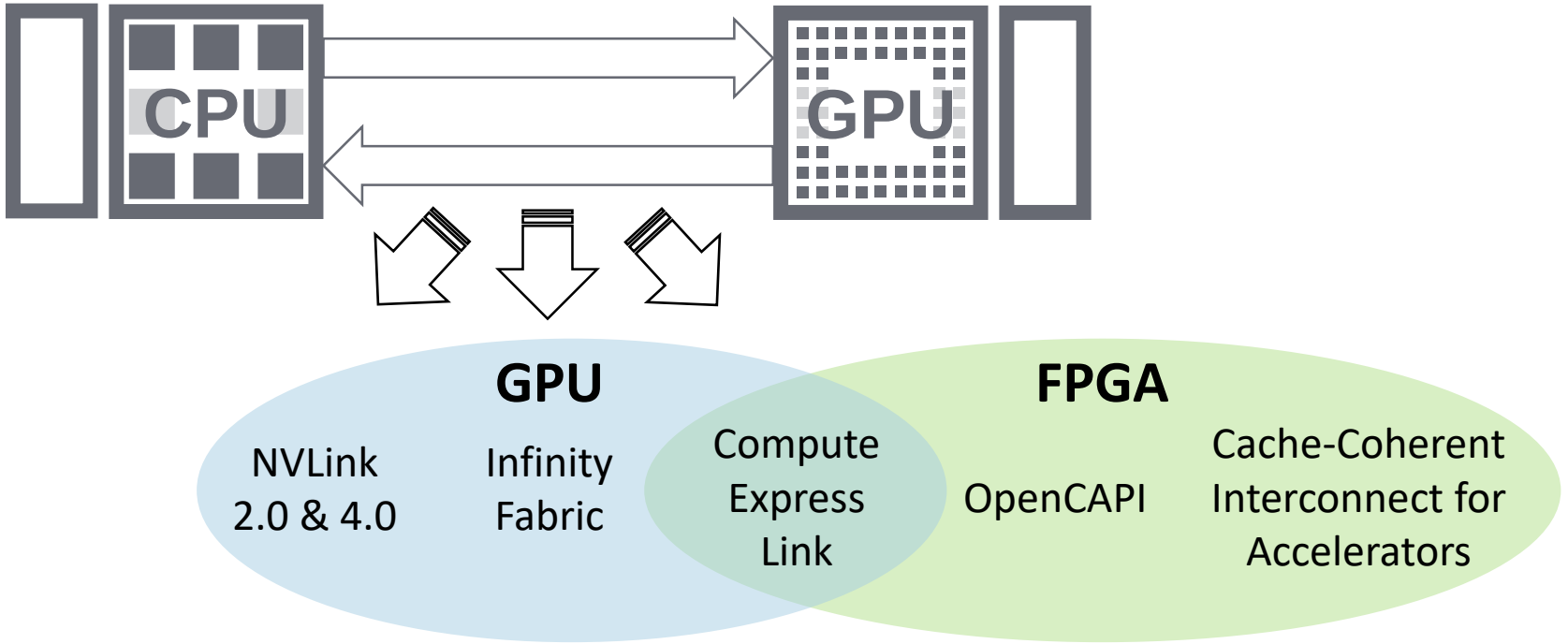
Game Changer: Fast Interconnects



Game Changer: Fast Interconnects



Game Changer: Fast Interconnects



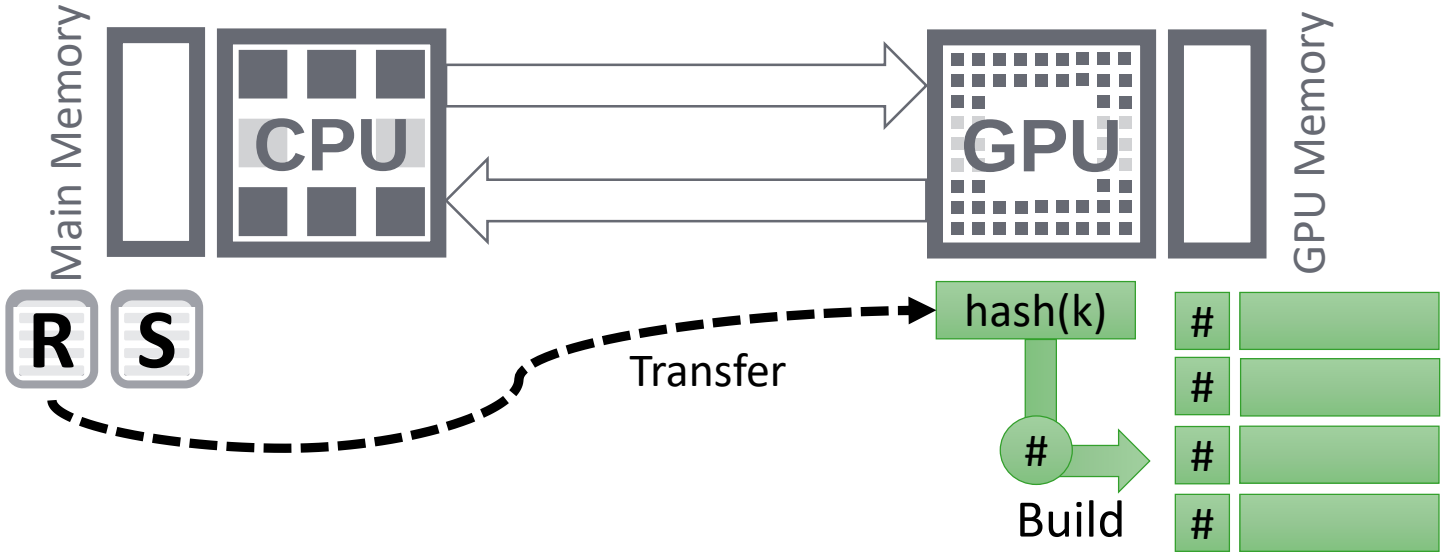
Contributions

- Revisit GPU Hash Joins
- Identify Hardware Bottlenecks
- Out-of-core Radix Partitioning
- Triton Join Algorithm

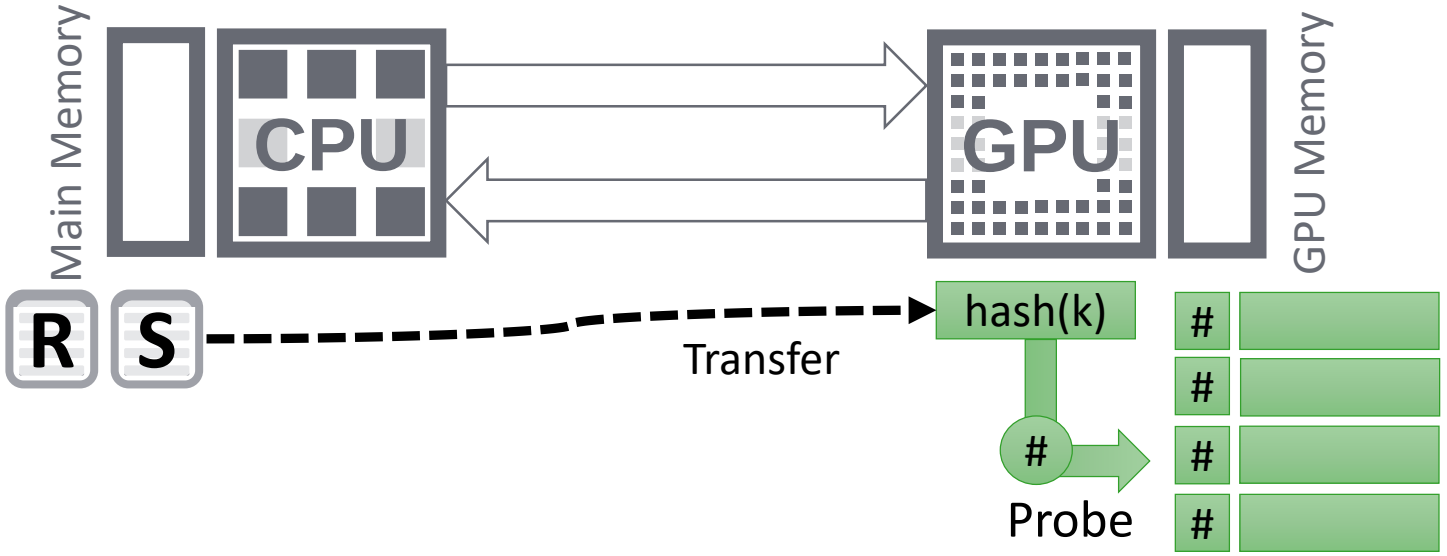
Agenda

- Revisit GPU Hash Joins
- Identify Hardware Bottlenecks
- Out-of-core Radix Partitioning
- Triton Join Algorithm

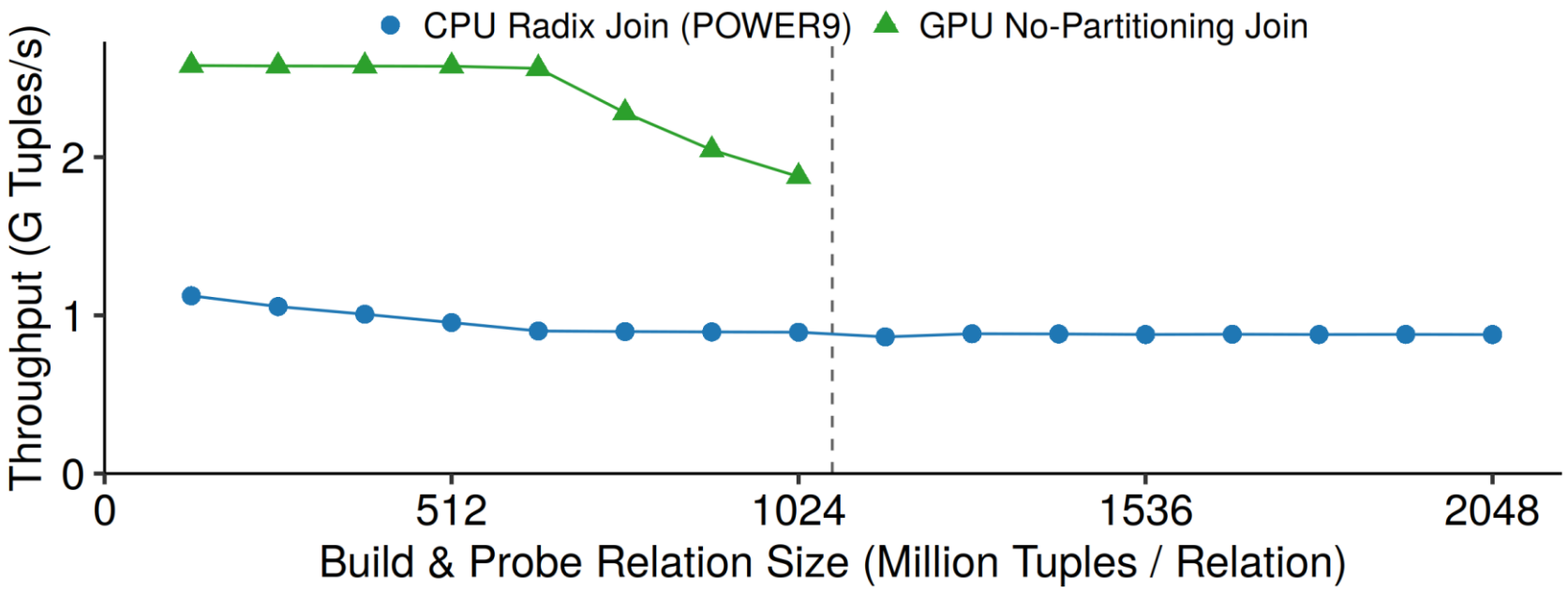
Approach 1: Hash Table in GPU Memory



Approach 1: Hash Table in GPU Memory

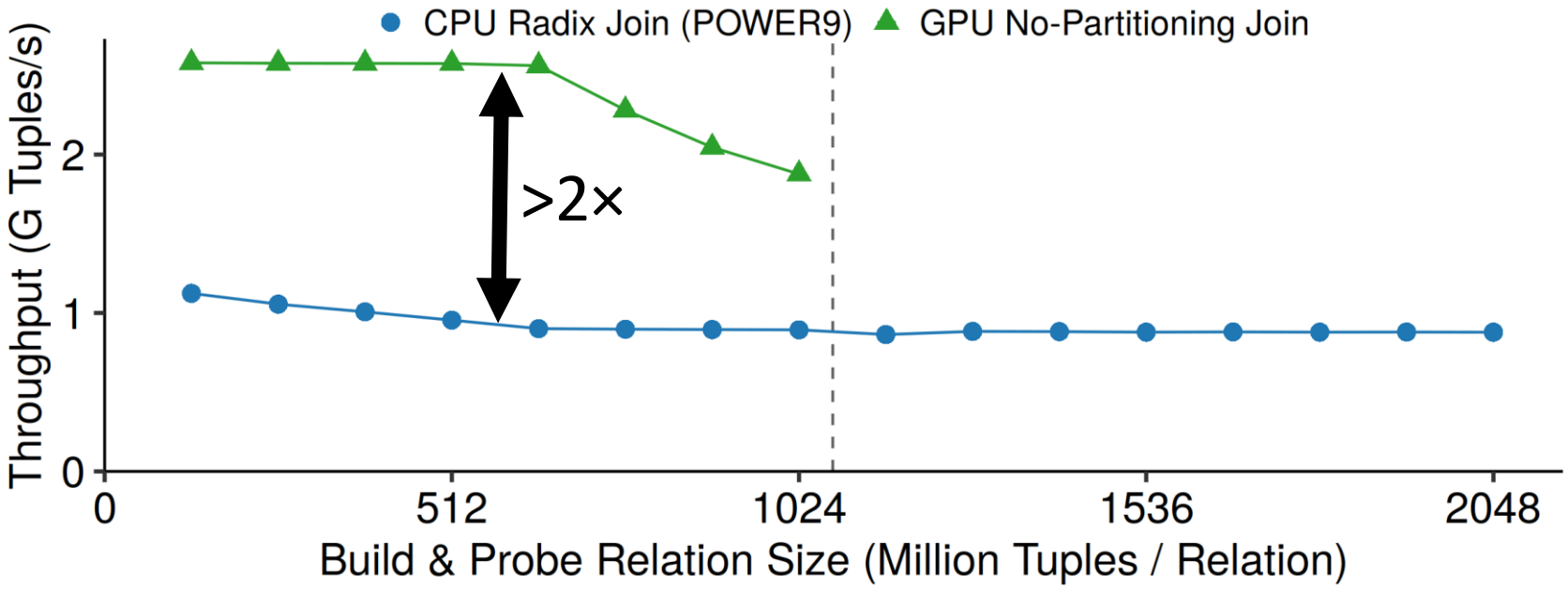


Challenge 1

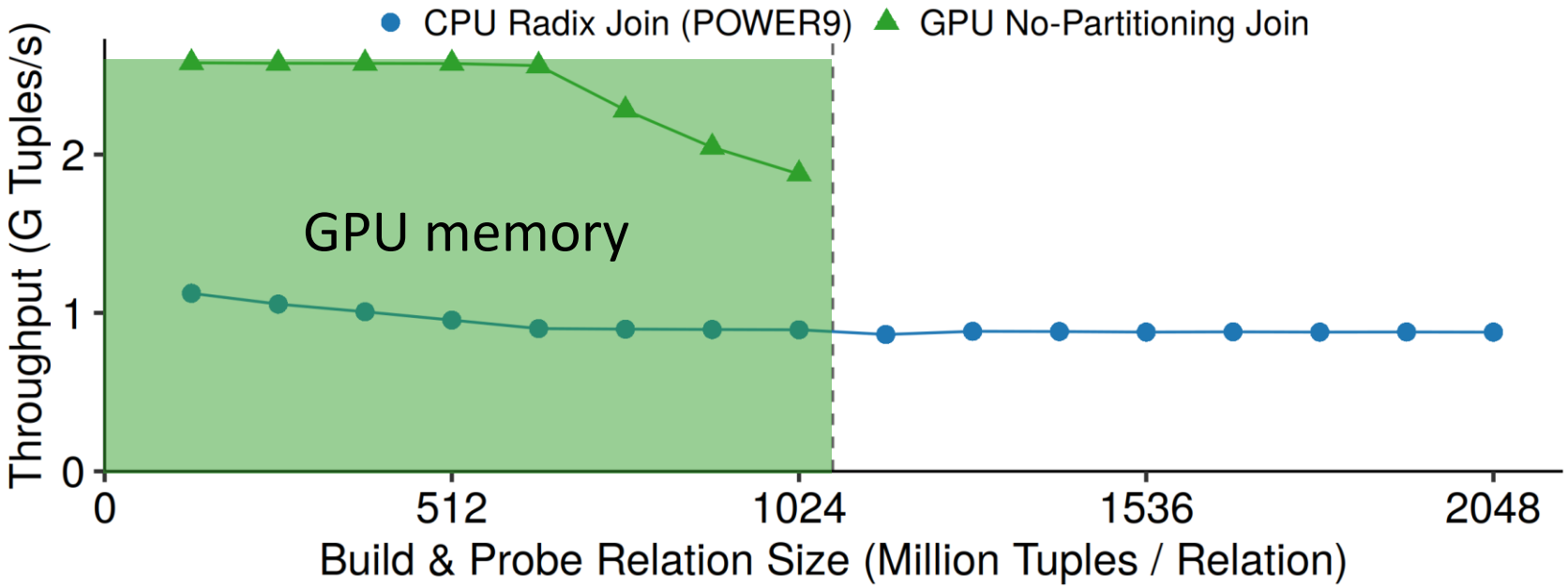


Data:	30 GiB ⋈ 30 GiB
CPU:	IBM POWER9 with 16 cores
GPU:	Nvidia V100 with NVLink 2.0

Challenge 1

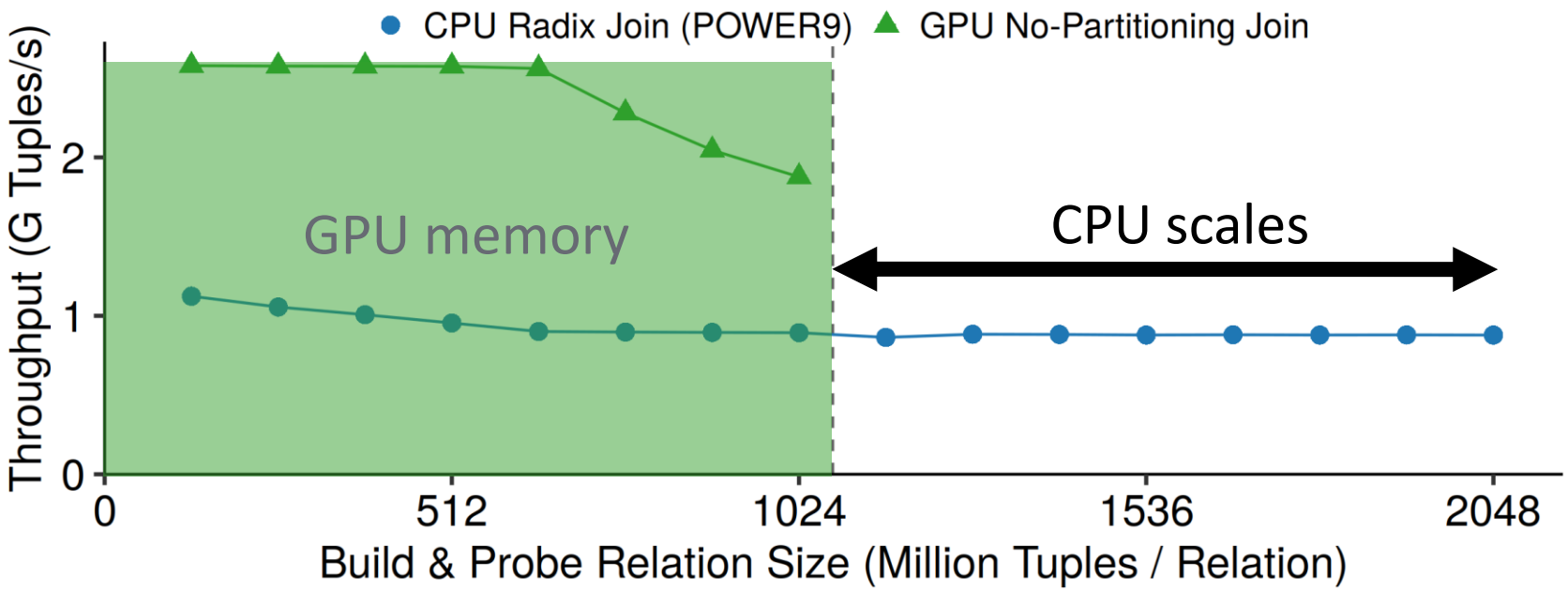

CPU Radix Join (POWER9)
GPU No-Partitioning Join
Throughput (G Tuples/s)
0
1
2
>2×
0
512
1024
1536
2048
Build & Probe Relation Size (Million Tuples / Relation)

Challenge 1



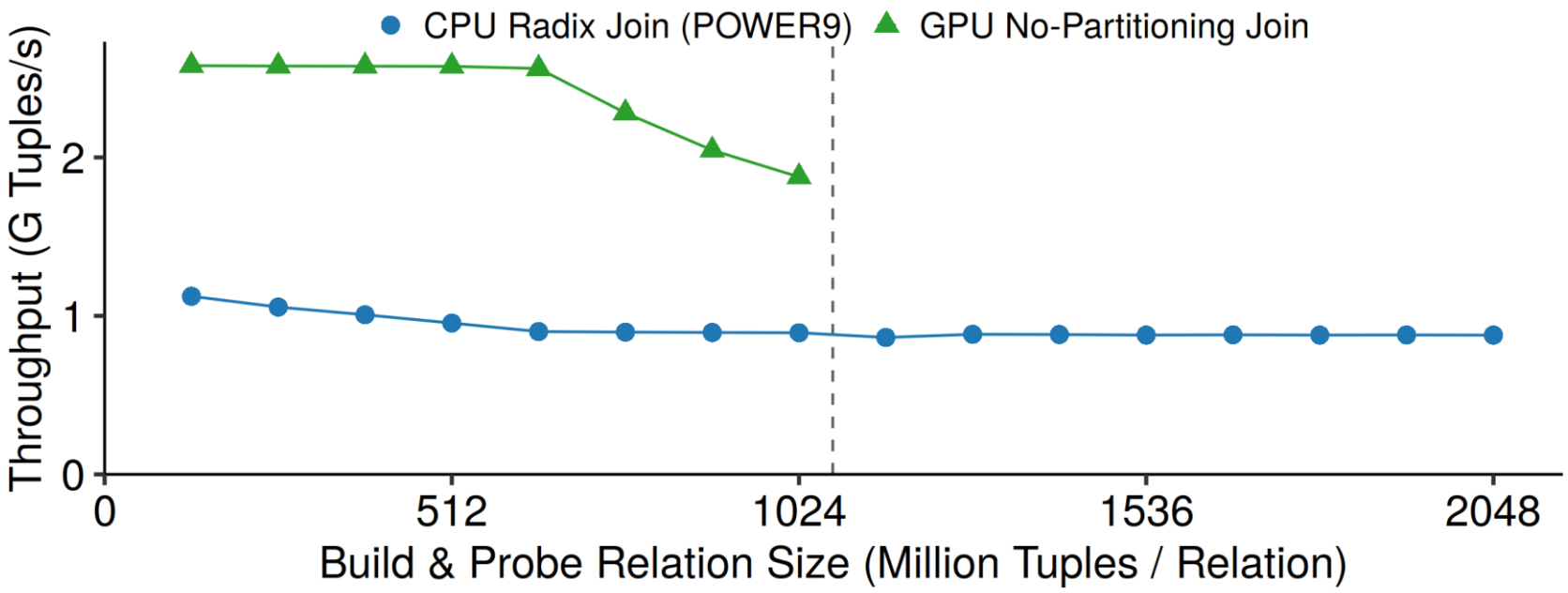
- Join state has limited size

Challenge 1



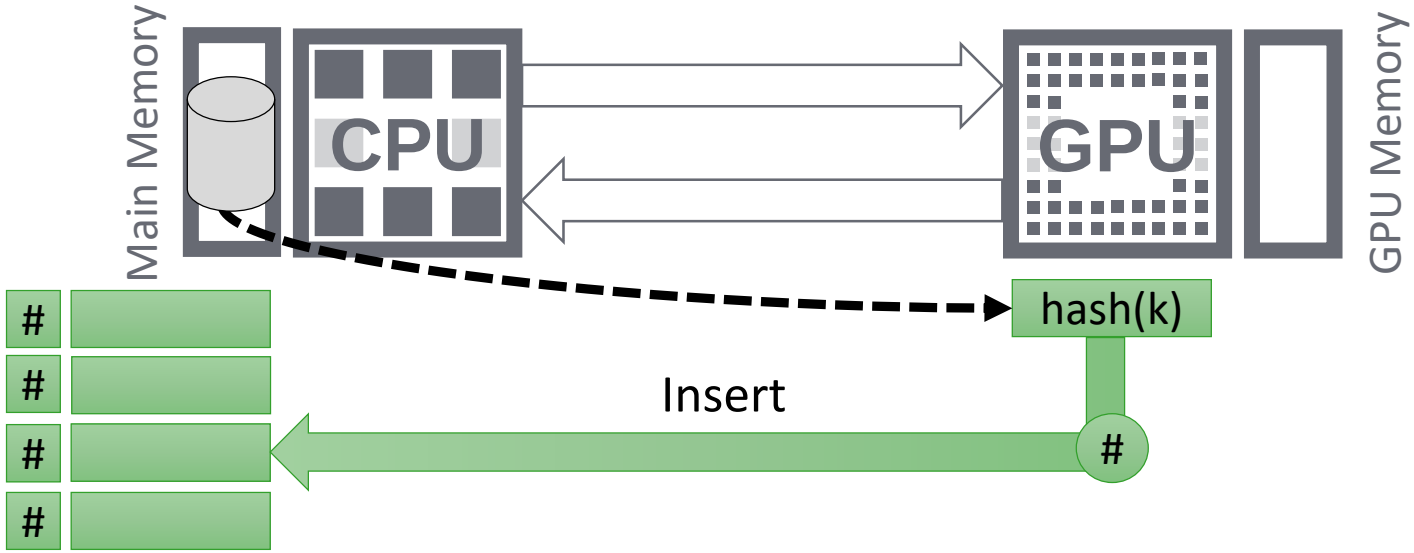
- Join state has limited size

Challenge 1

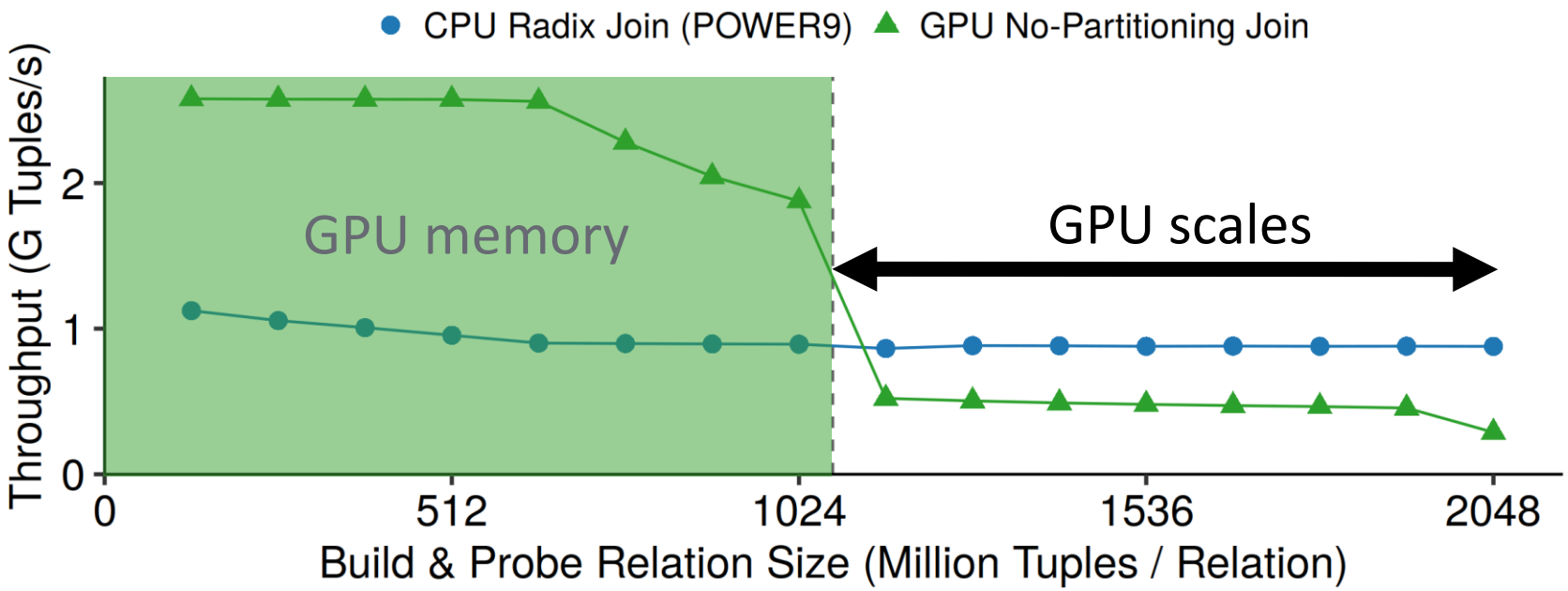


GPU hash join does not scale to a large join state

Approach 2: Spill Hash Table

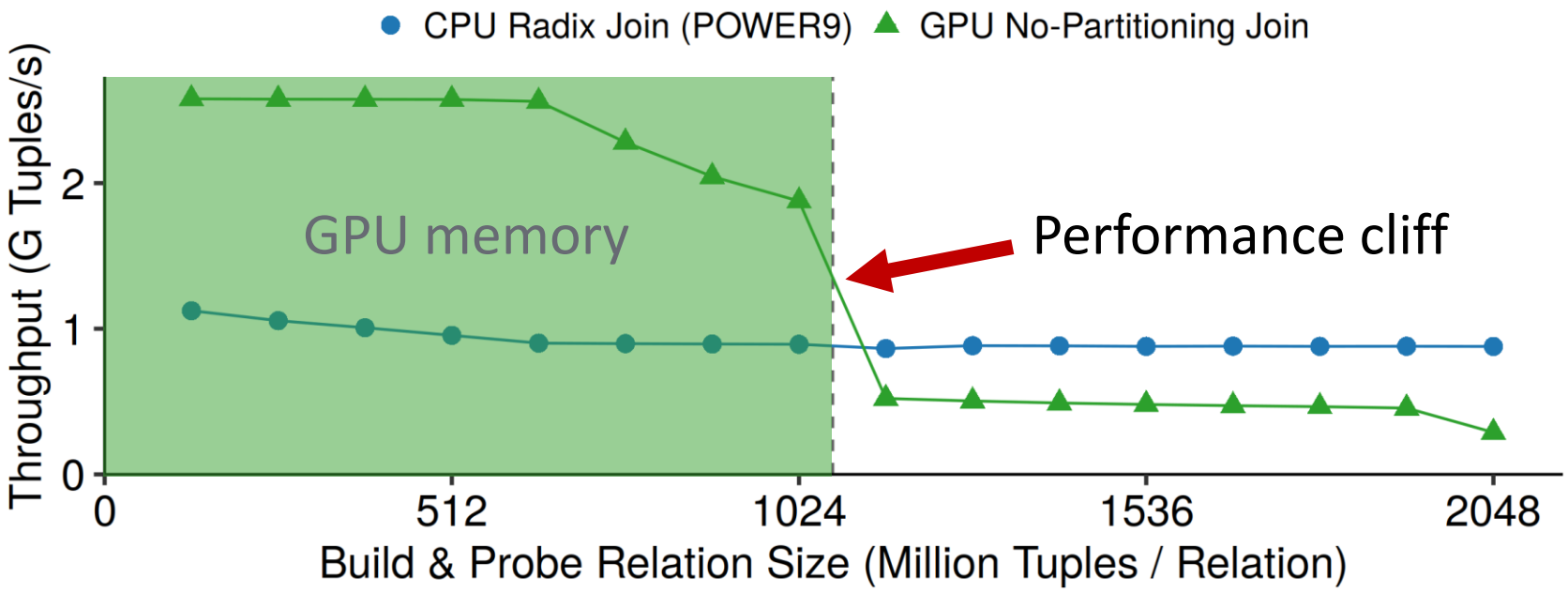


Challenge 2



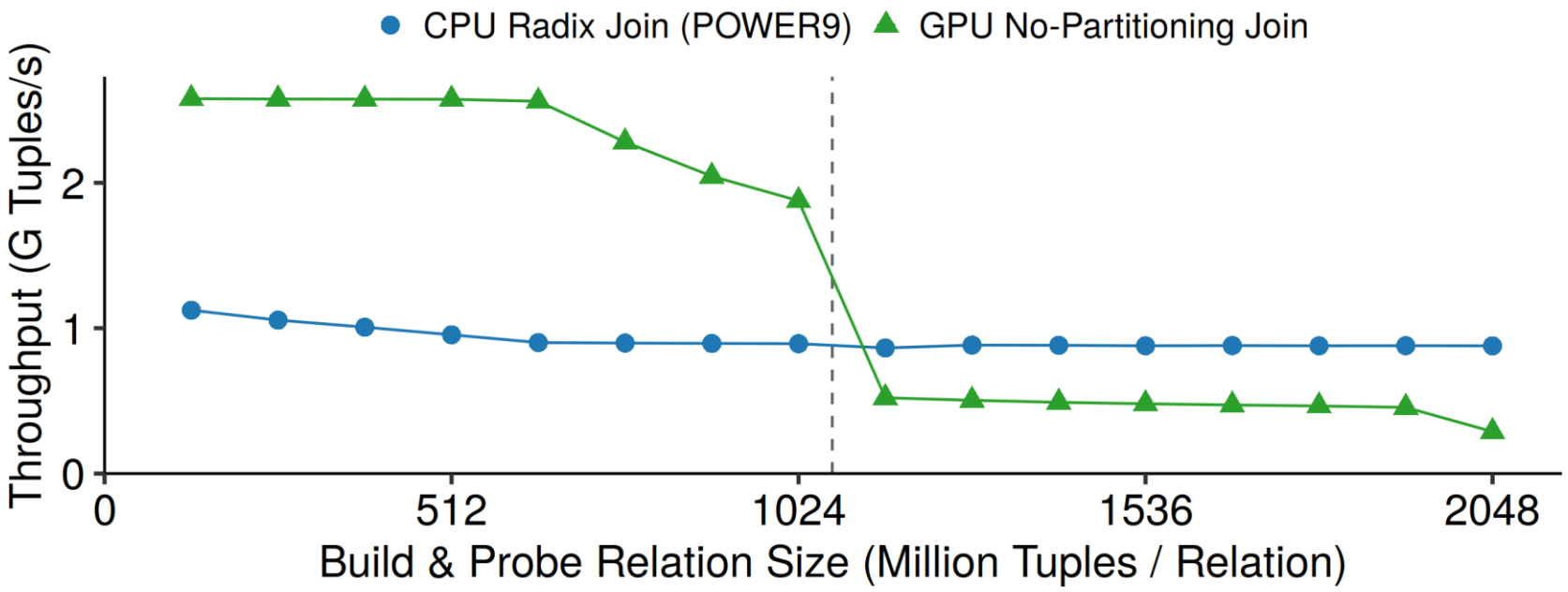
- Scalable ✓

Challenge 2



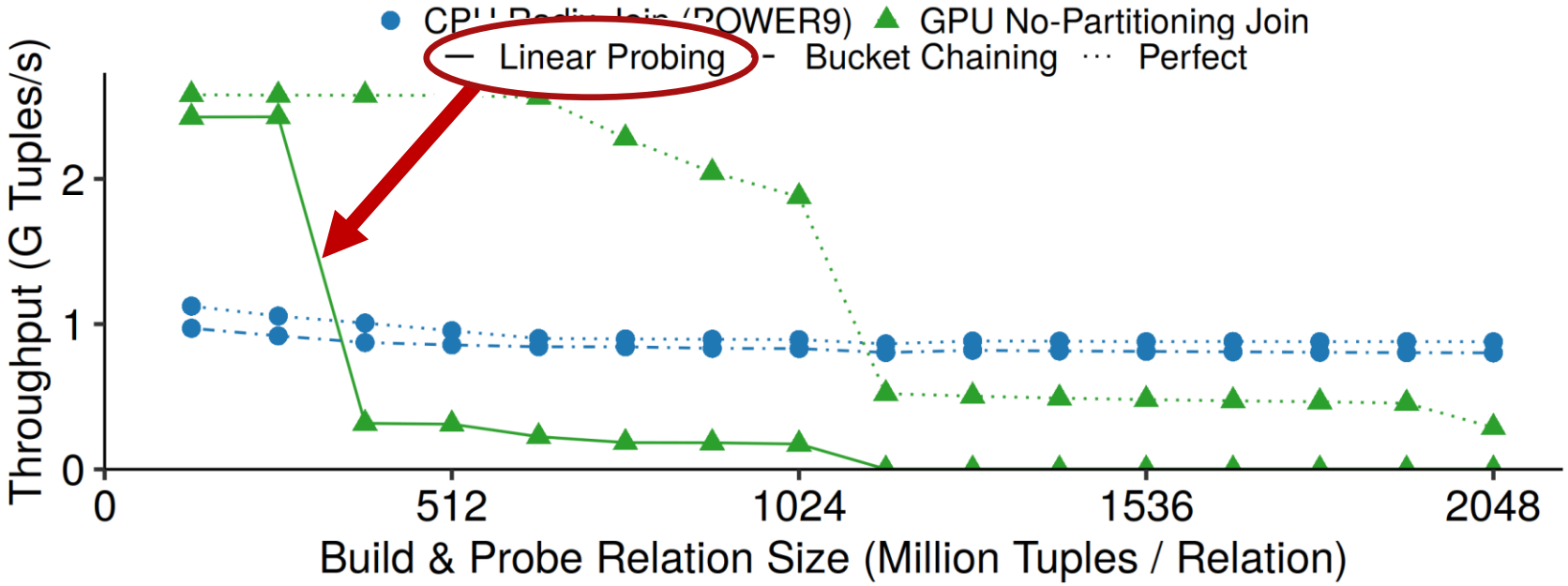
- Scalable ✓
- But: Performance cliff for large hash tables

Challenge 2



Spilling hash table does not lead to robust performance

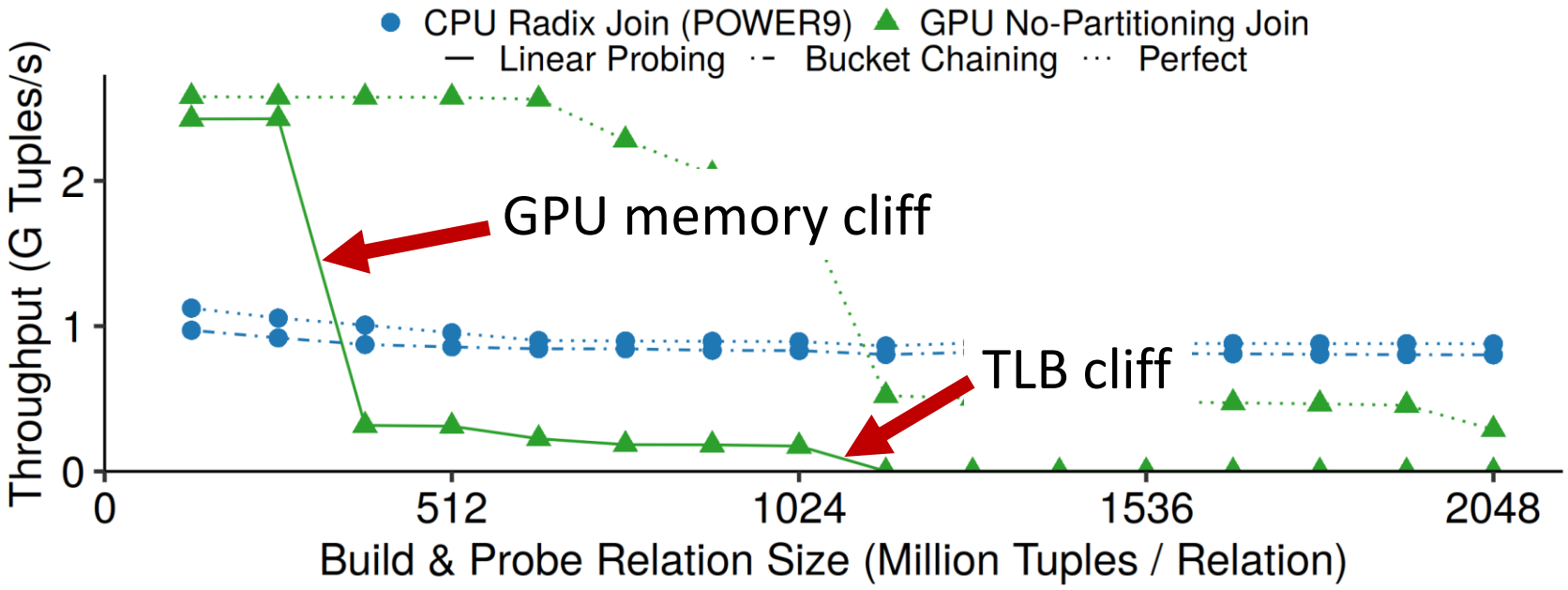
Challenge 2



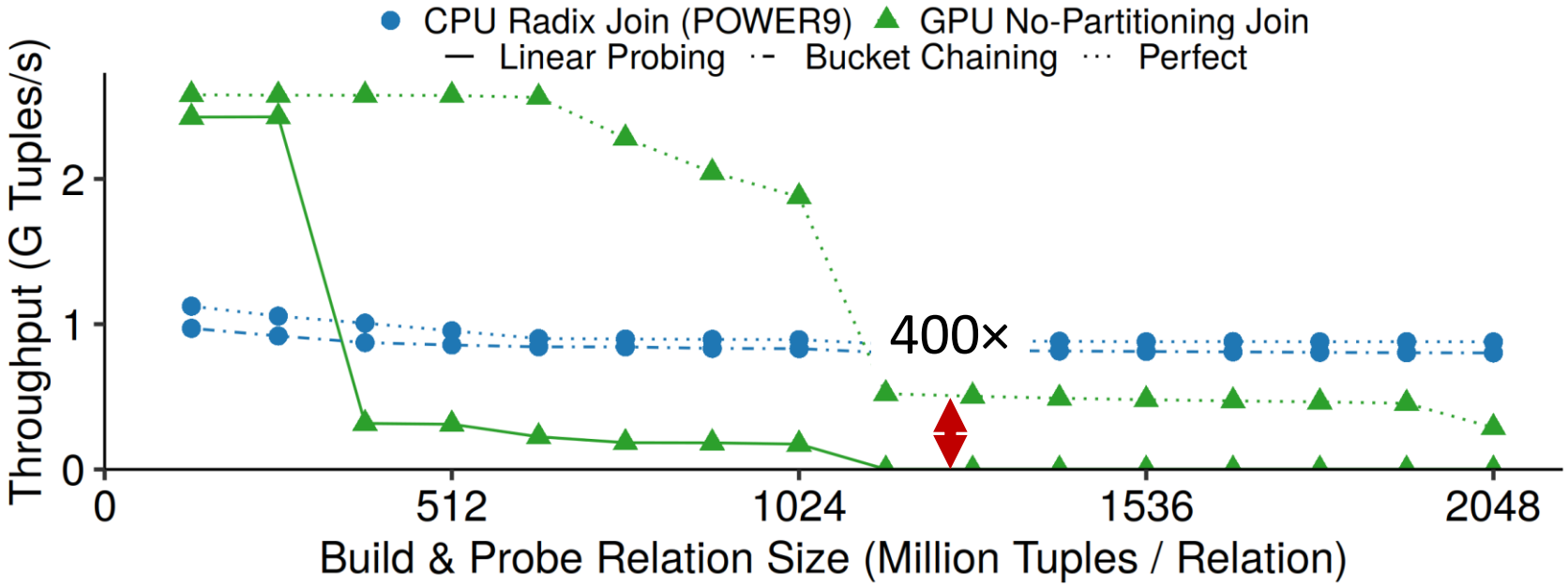
- Linear probing instead of perfect hashing

  → 2× hash table size

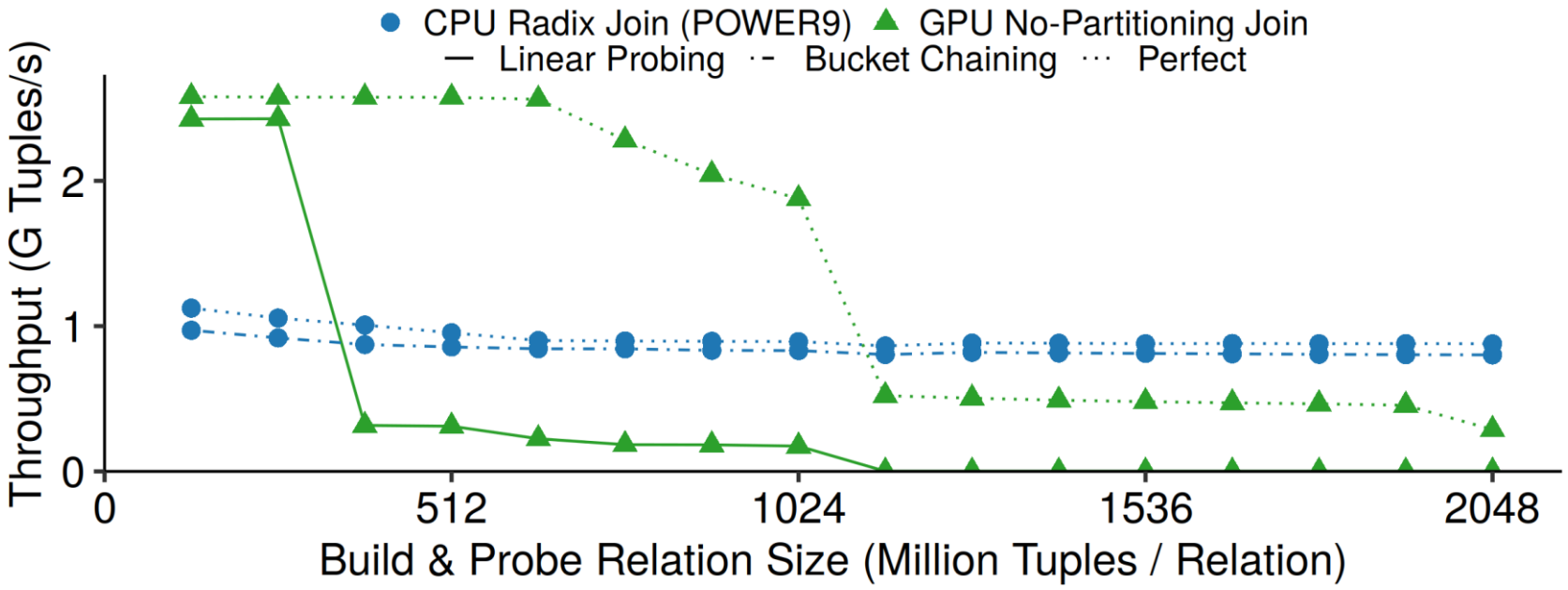
Challenge 2


CPU Radix Join (POWER9)
GPU No-Partitioning Join
Linear Probing
Bucket Chaining
Perfect
Throughput (G Tuples/s)
0
1
2
GPU memory cliff
TLB cliff
0
512
1024
1536
2048
Build & Probe Relation Size (Million Tuples / Relation)

Challenge 2

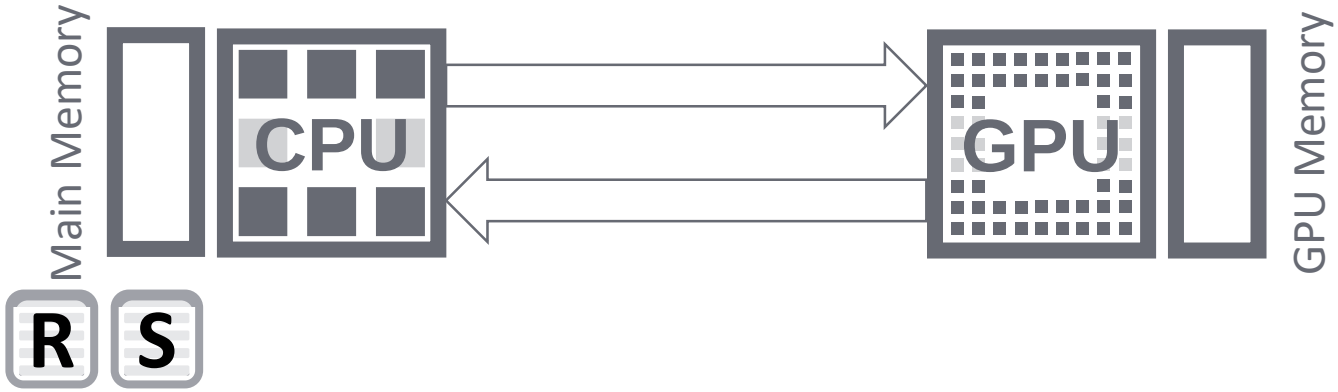

CPU Radix Join (POWER9)
GPU No-Partitioning Join
Linear Probing
Bucket Chaining
Perfect
Throughput (G Tuples/s)
0
1
2
400×
0
512
1024
1536
2048
Build & Probe Relation Size (Million Tuples / Relation)

Challenge 2

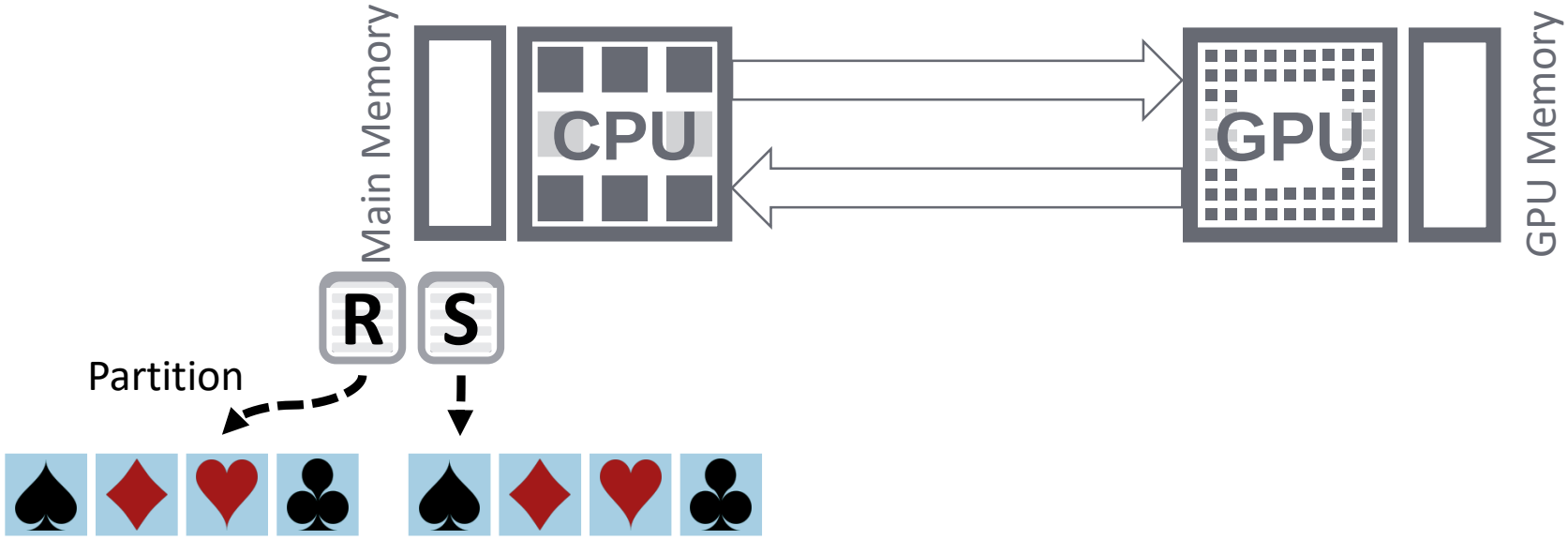


GPU TLB misses cause a slow down for large hash tables

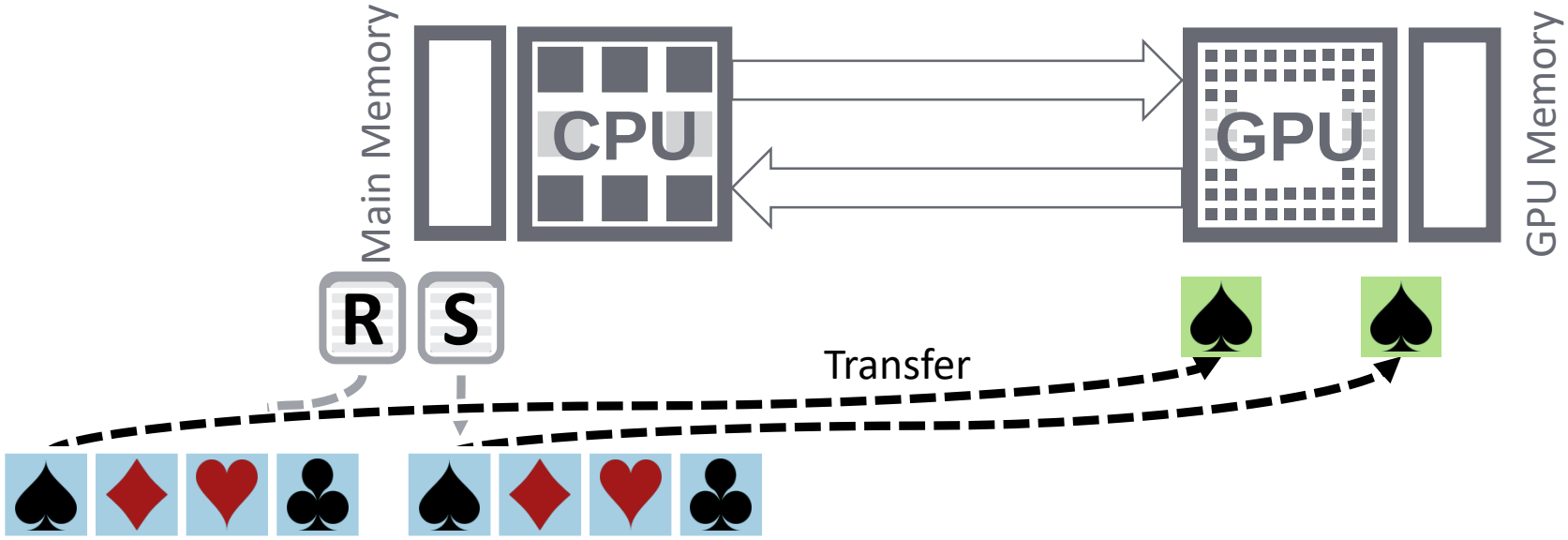
Approach 3: Partition Data using CPU



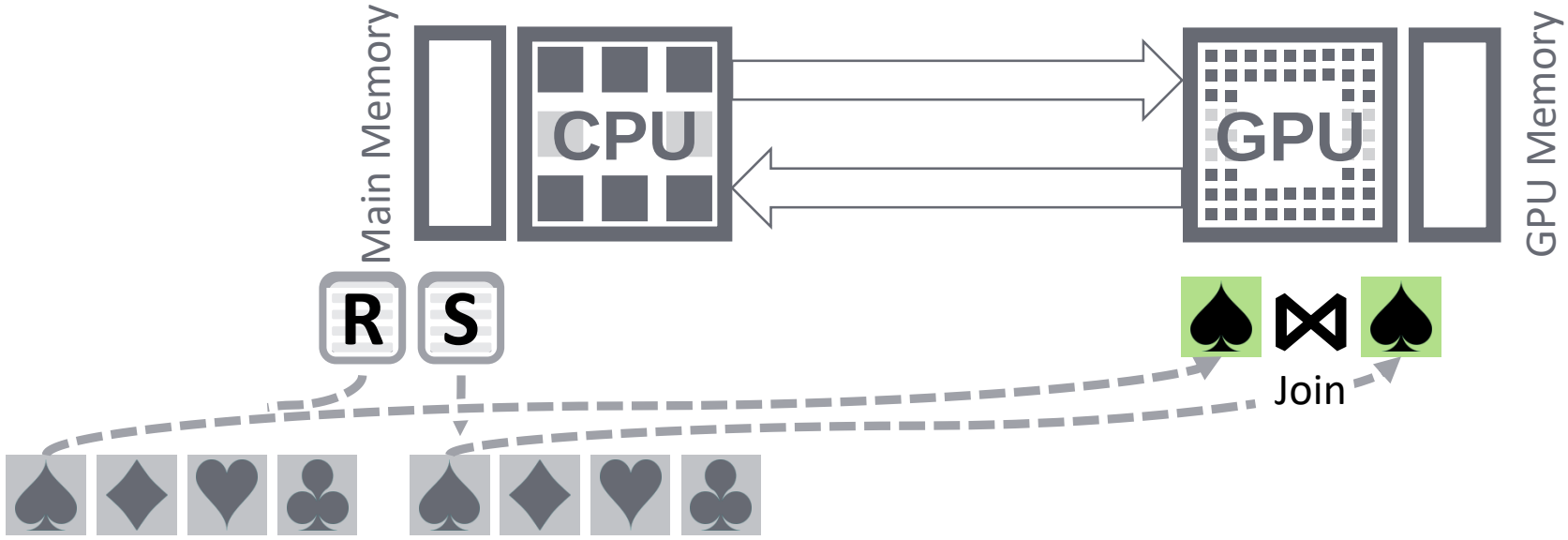
Approach 3: Partition Data using CPU



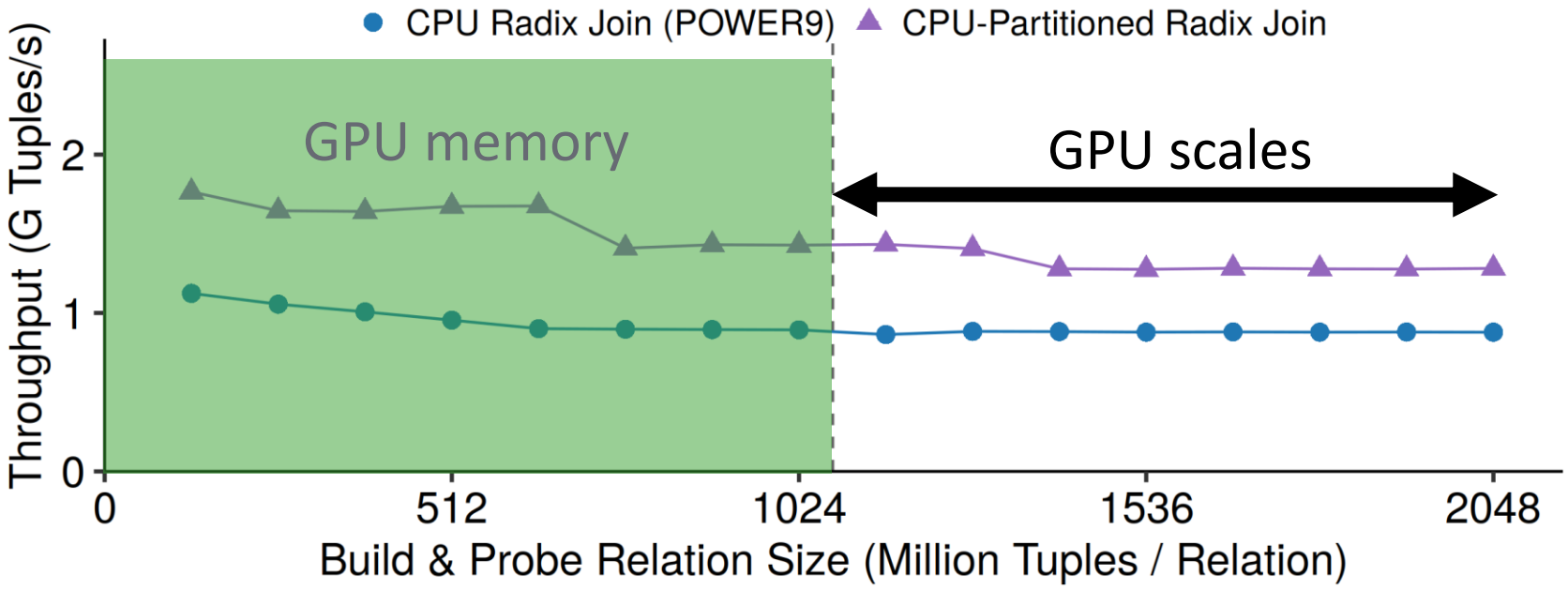
Approach 3: Partition Data using CPU



Approach 3: Partition Data using CPU

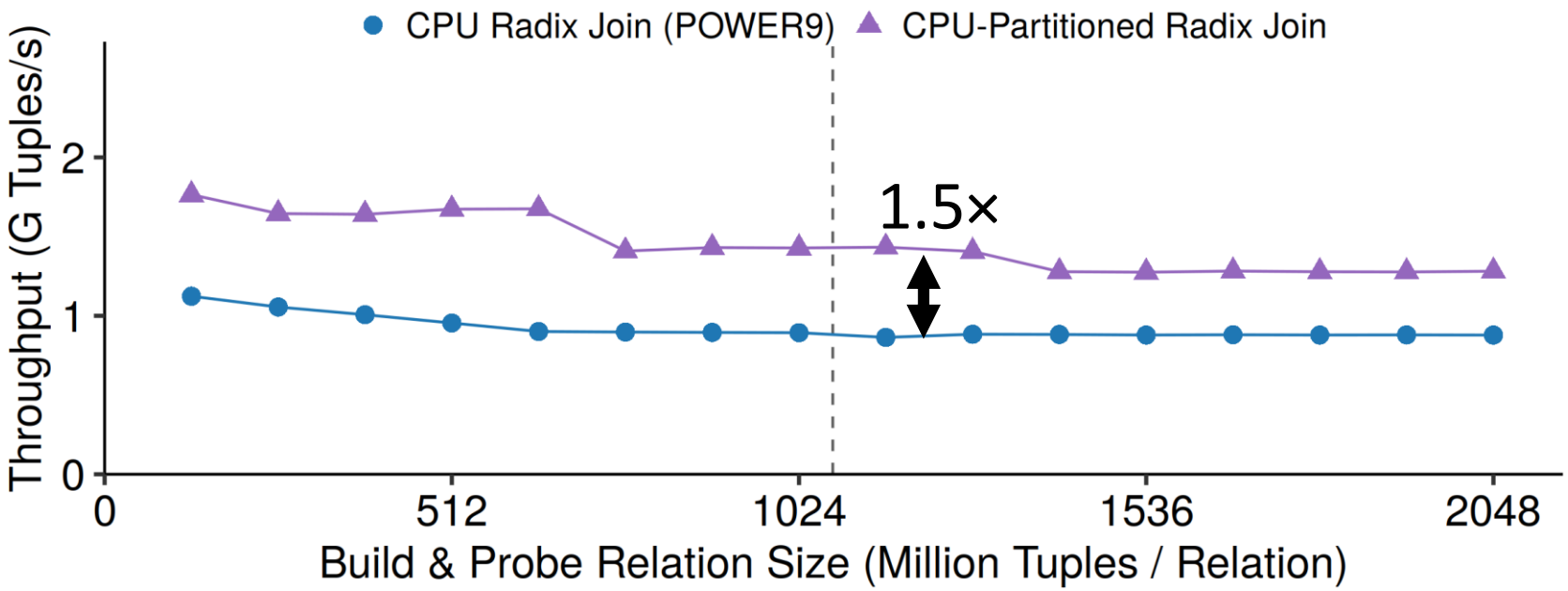


Challenge 3



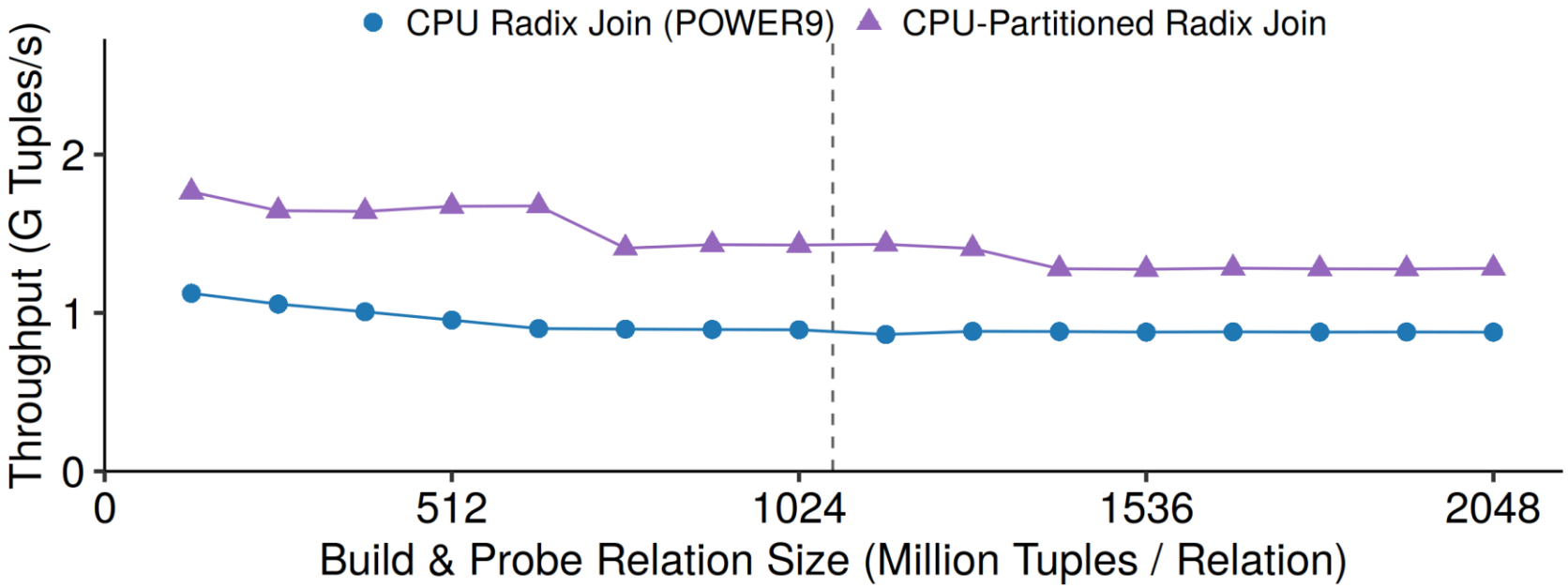
- Scalable ✓

Challenge 3



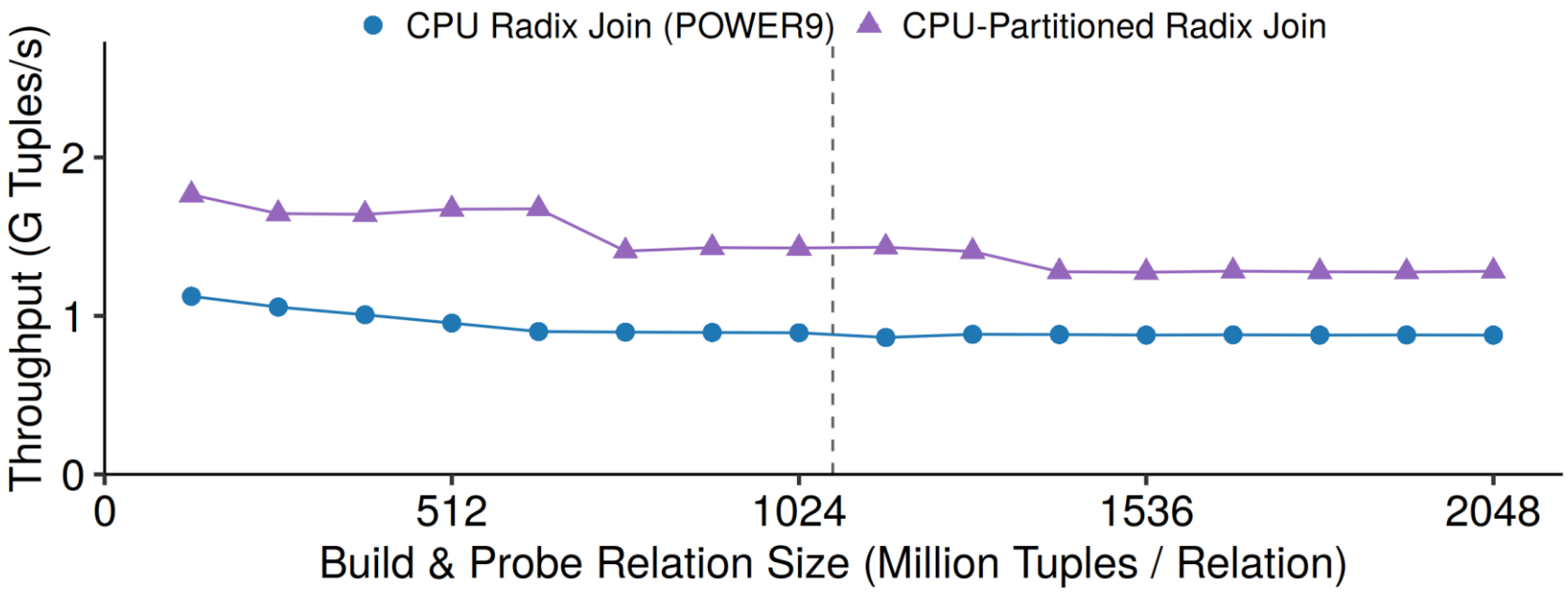
- Scalable ✓
- Robust ✓

Challenge 3



- Scalable ✓
- Robust ✓
- But: Requires a fast CPU and a fast GPU

Challenge 3



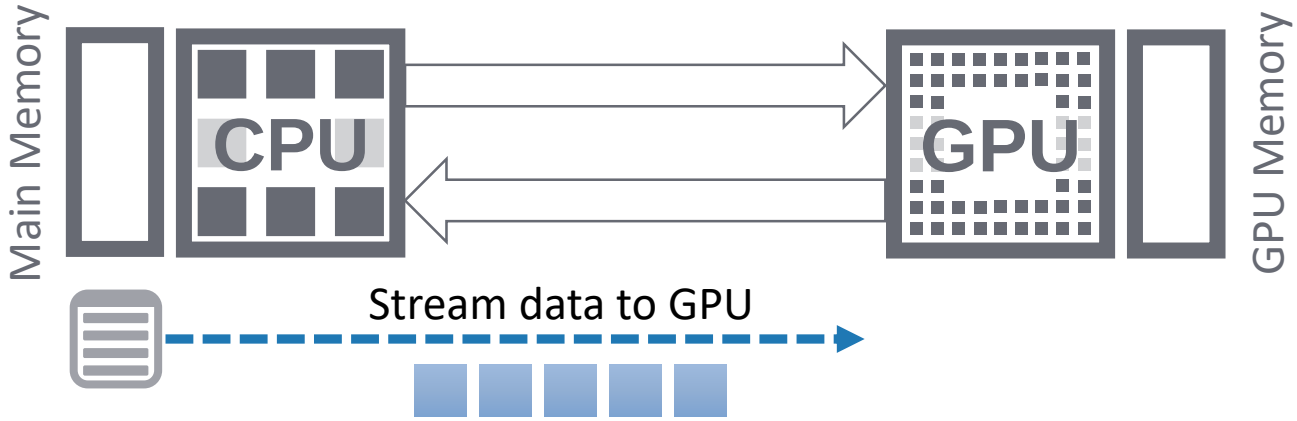
CPU-partitioned radix join is not resource-efficient

Challenge Summary

*Higher interconnect bandwidth is **necessary,***
***but not sufficient** to achieve high scalability.*

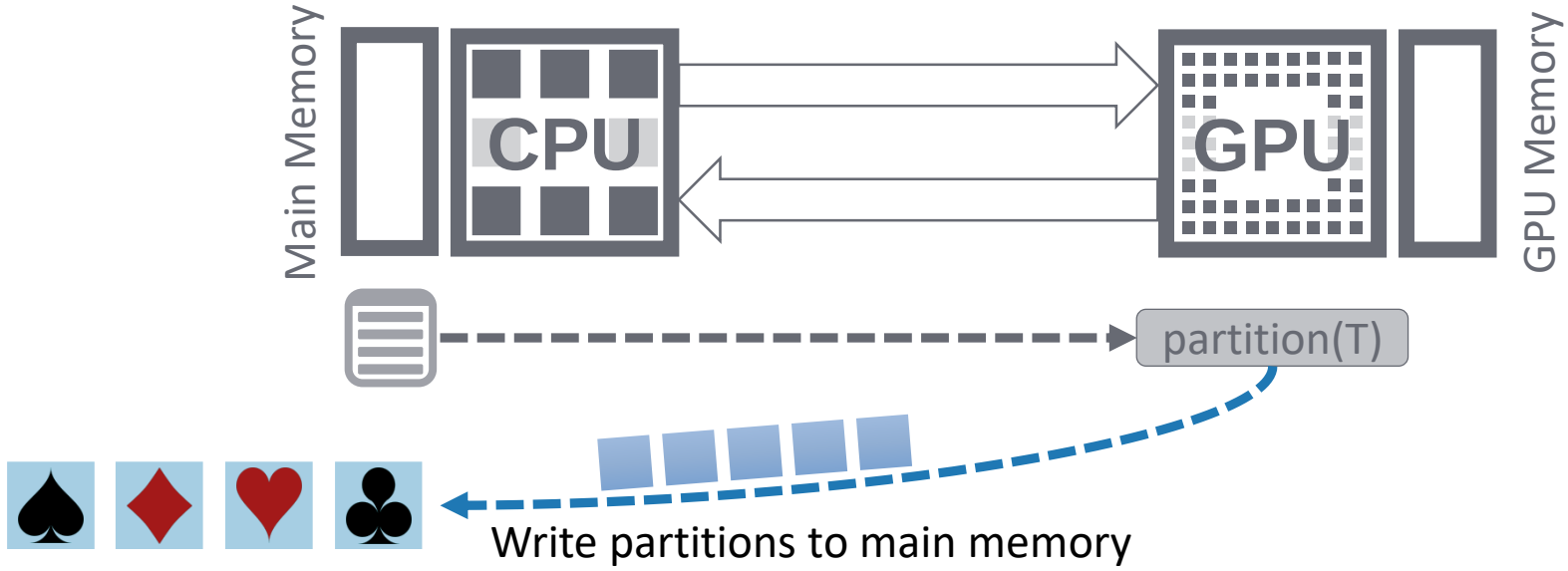
Key Insight

Key Insight

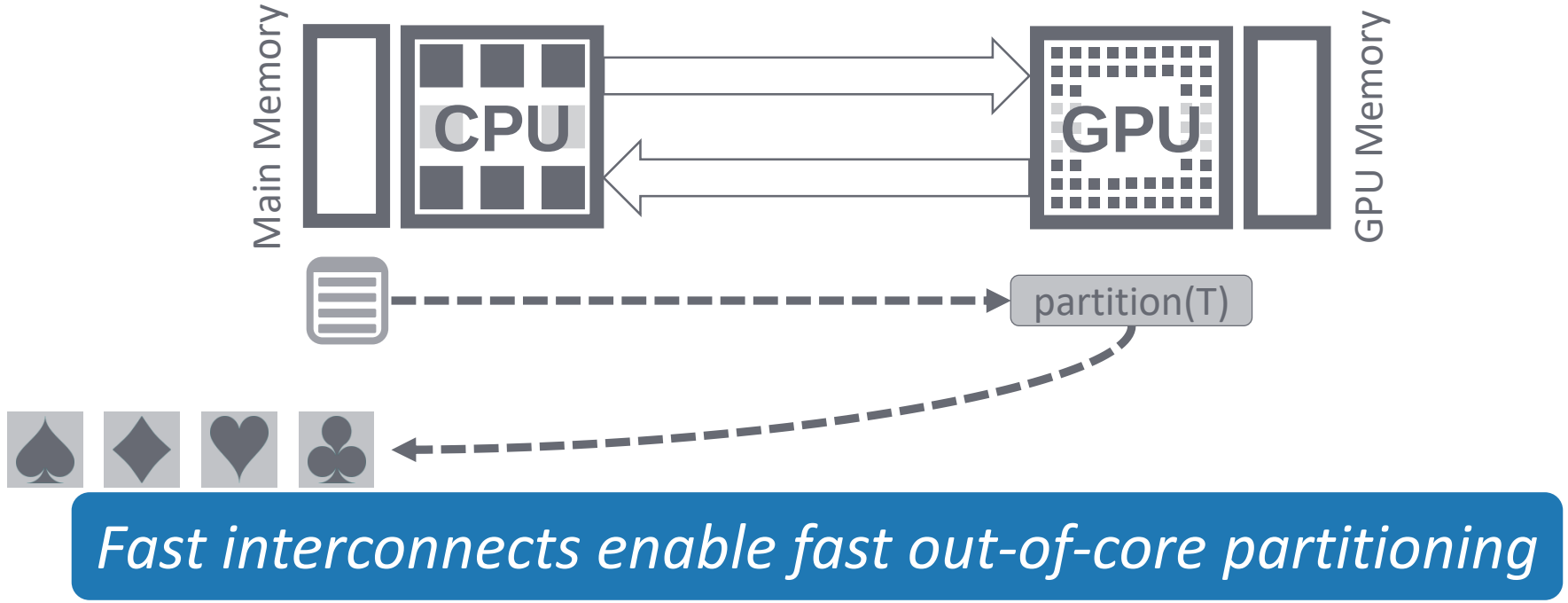


Key Insight

Key Insight

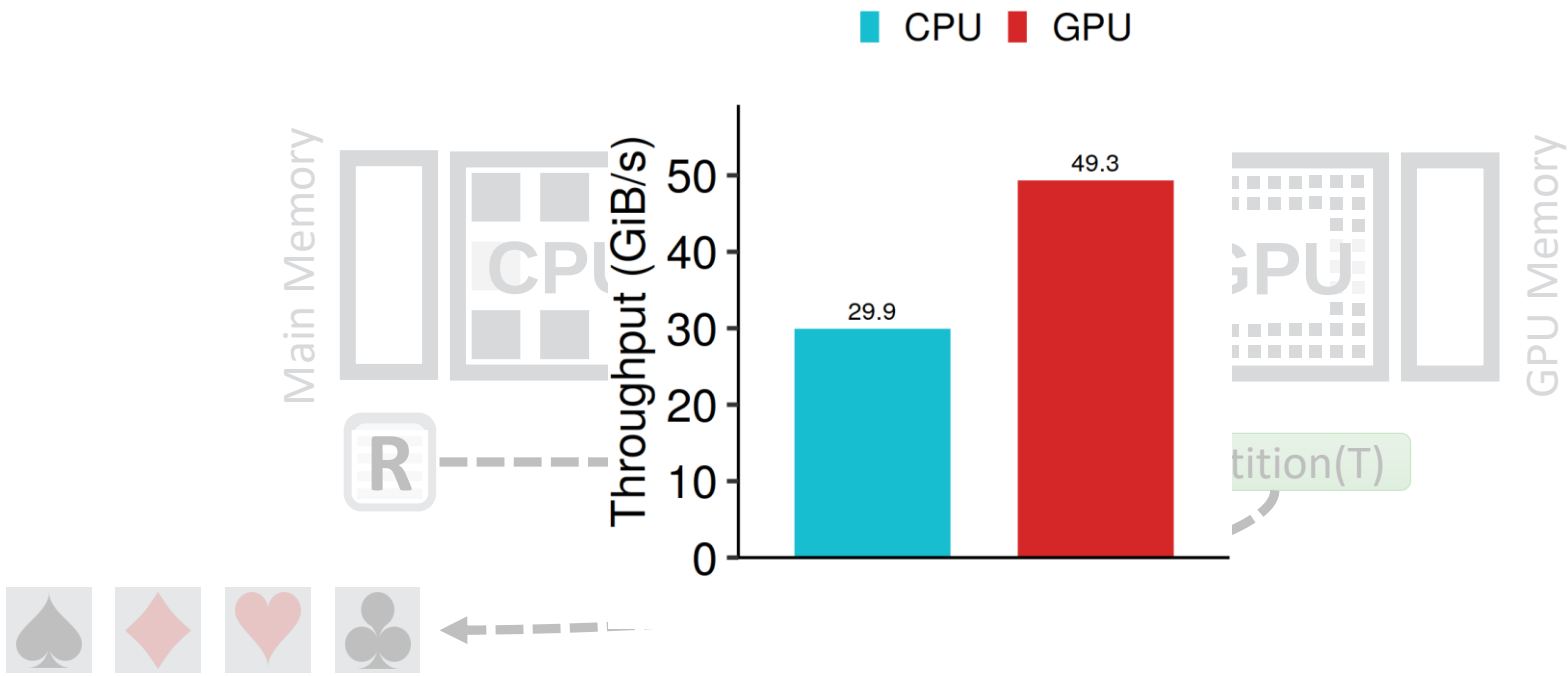

Main Memory
CPU
GPU
GPU Memory
partition(T)
Write partitions to main memory

Key Insight



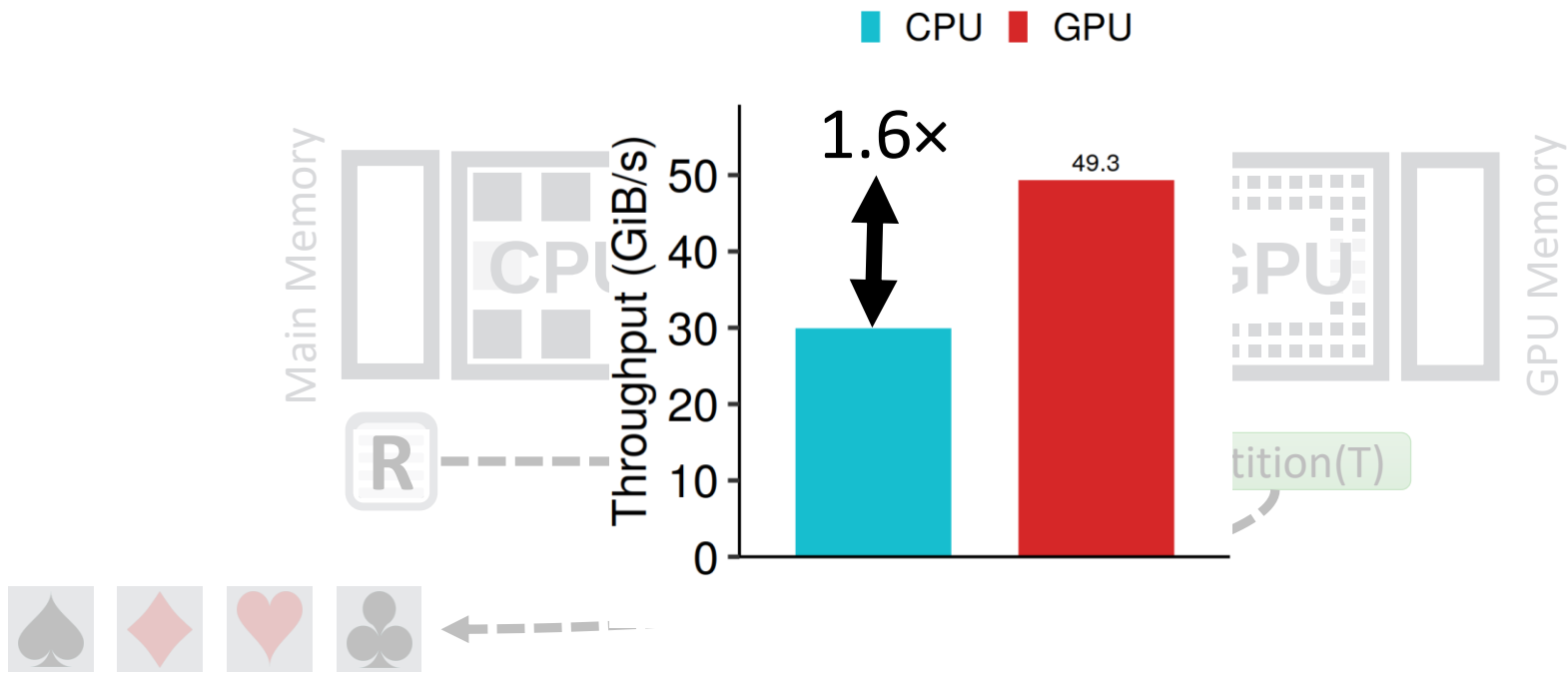
Fast interconnects enable fast out-of-core partitioning

Key Insight

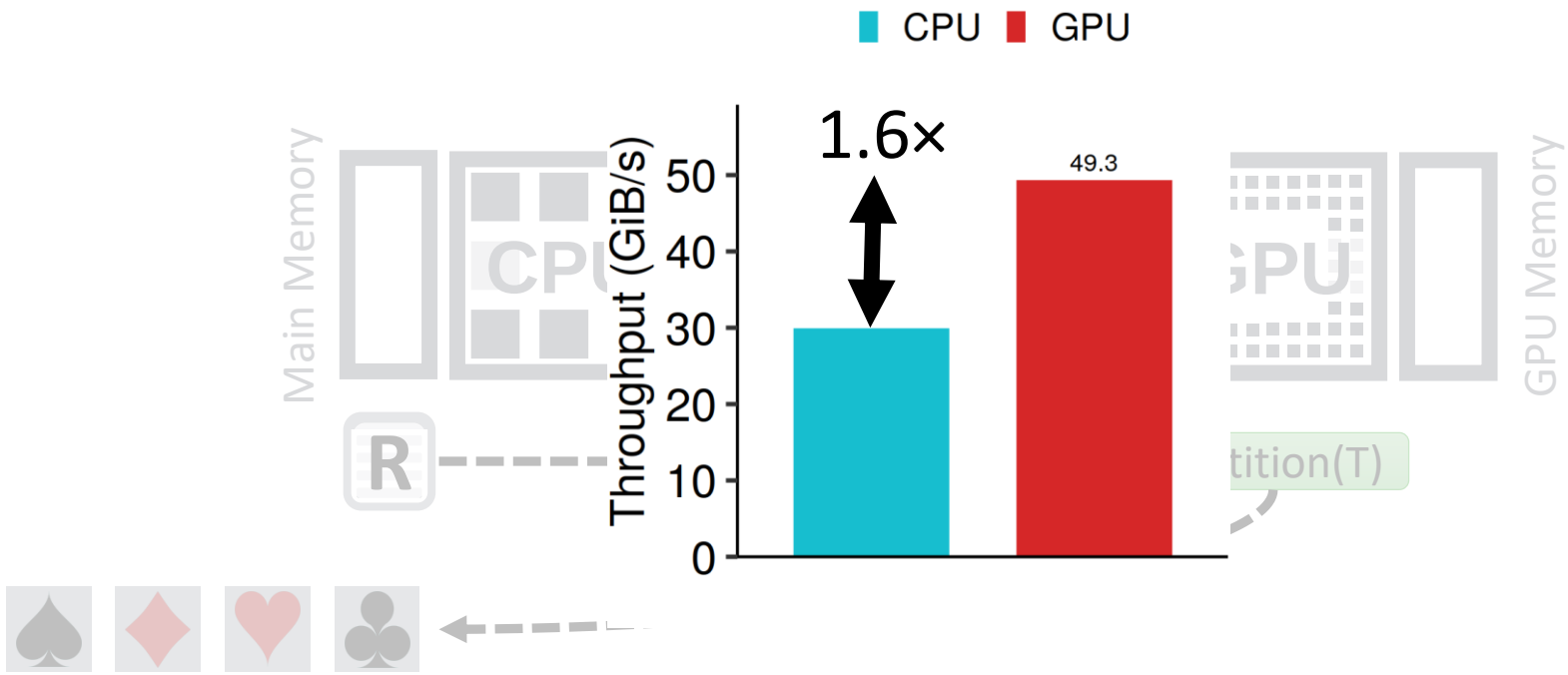


Data: 15 GiB
Fanout: 512 partitions

Key Insight


CPU
GPU
Throughput (GiB/s)
0
10
20
30
40
50
1.6×
49.3
Main Memory
GPU Memory
R
tition(T)

Key Insight



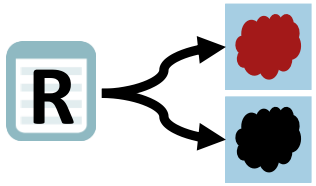
GPU with fast interconnect outperforms CPU

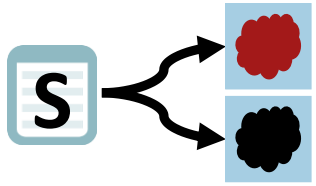
The Triton Join

The Triton Join

The Triton Join

Out-of-core
radix partitioning

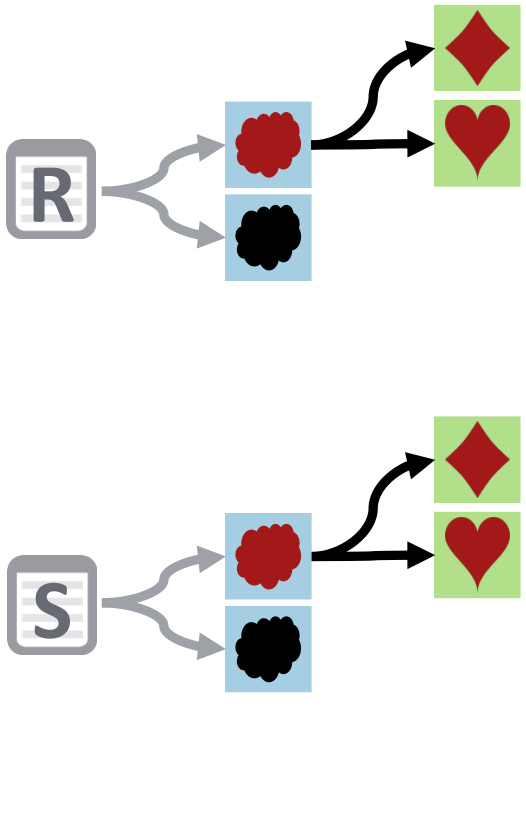




1st Pass
GPU Partitioning

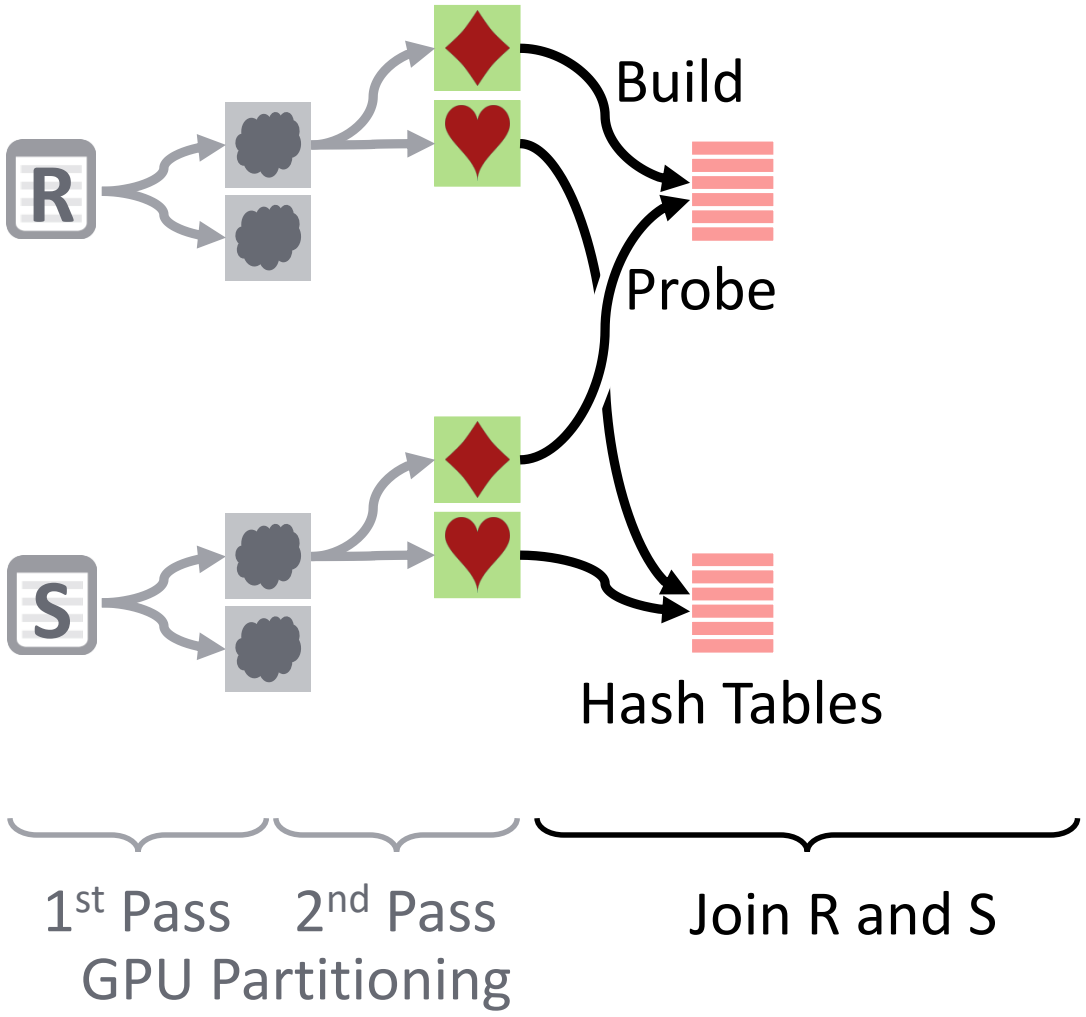
The Triton Join

Out-of-core radix partitioning



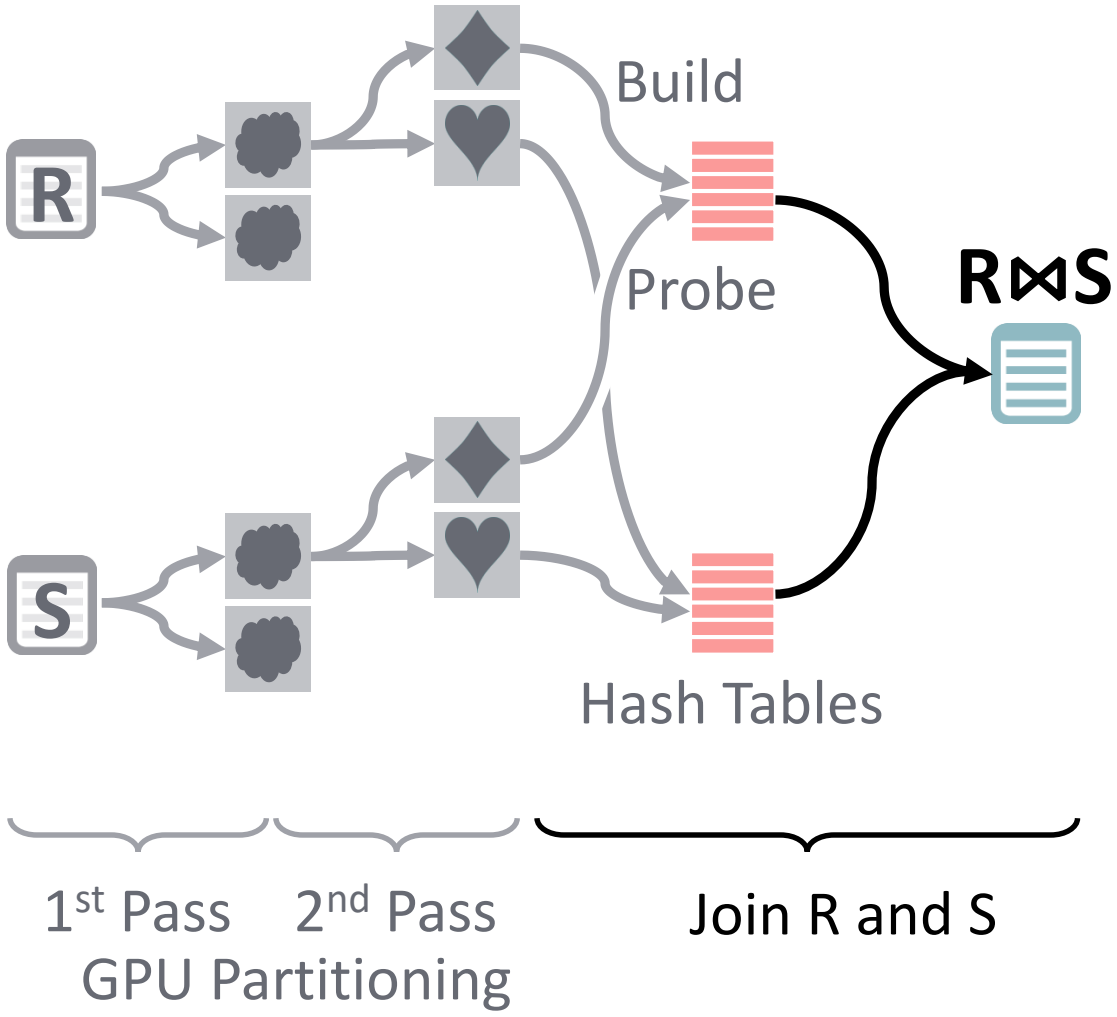
The Triton Join

Out-of-core radix partitioning



The Triton Join

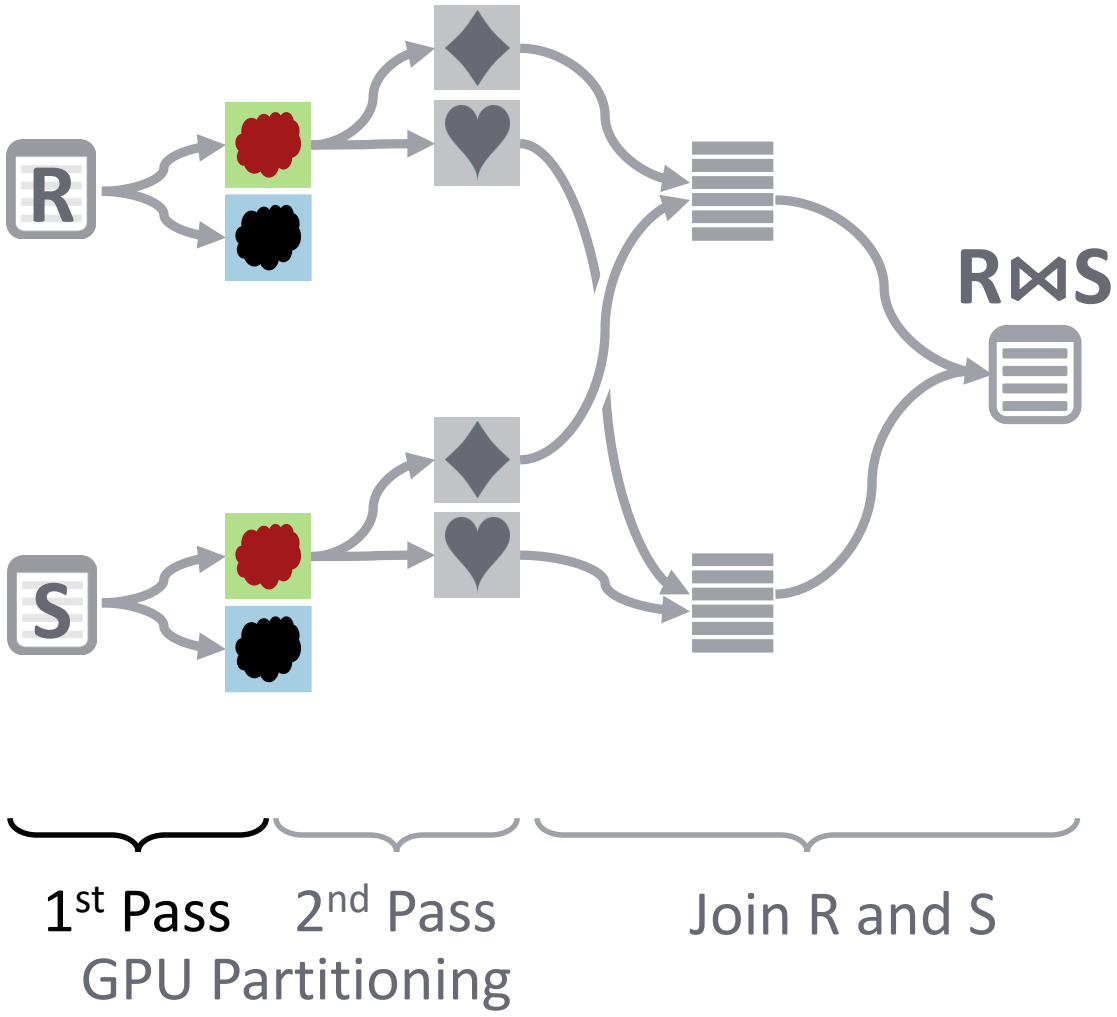
Out-of-core radix partitioning



The Triton Join

Out-of-core radix partitioning

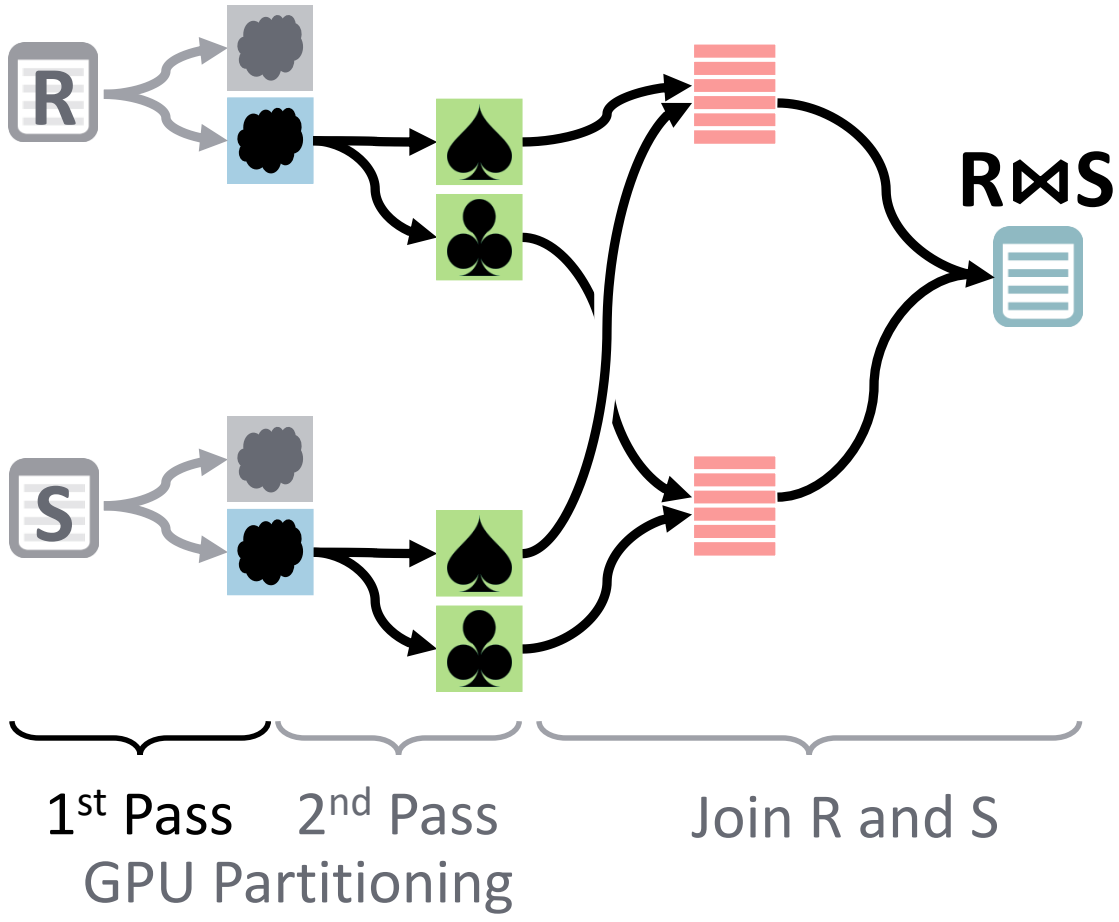
Caching partitions in GPU memory



The Triton Join

Out-of-core radix partitioning

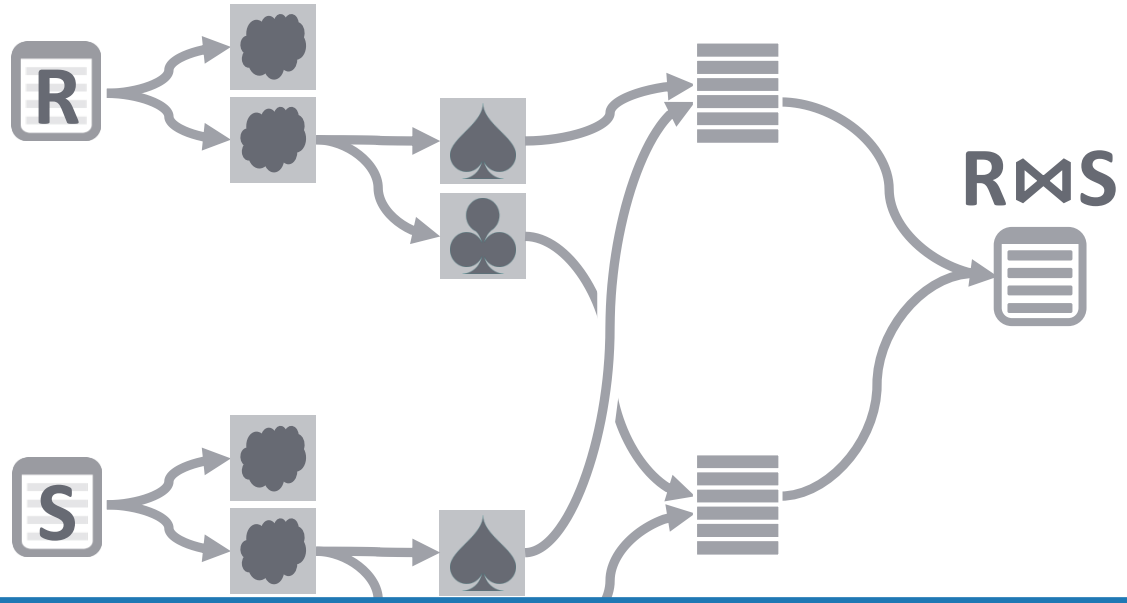
Caching partitions in GPU memory



The Triton Join

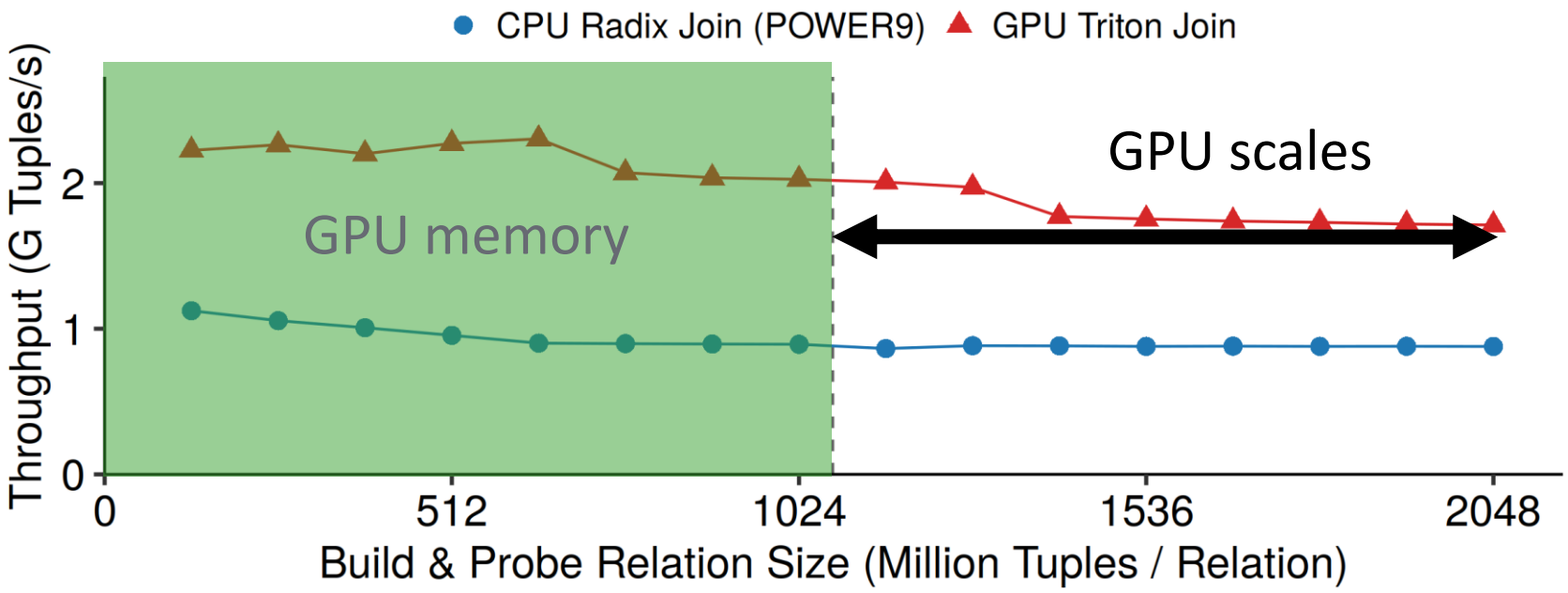
Out-of-core radix partitioning

Caching partitions in GPU memory

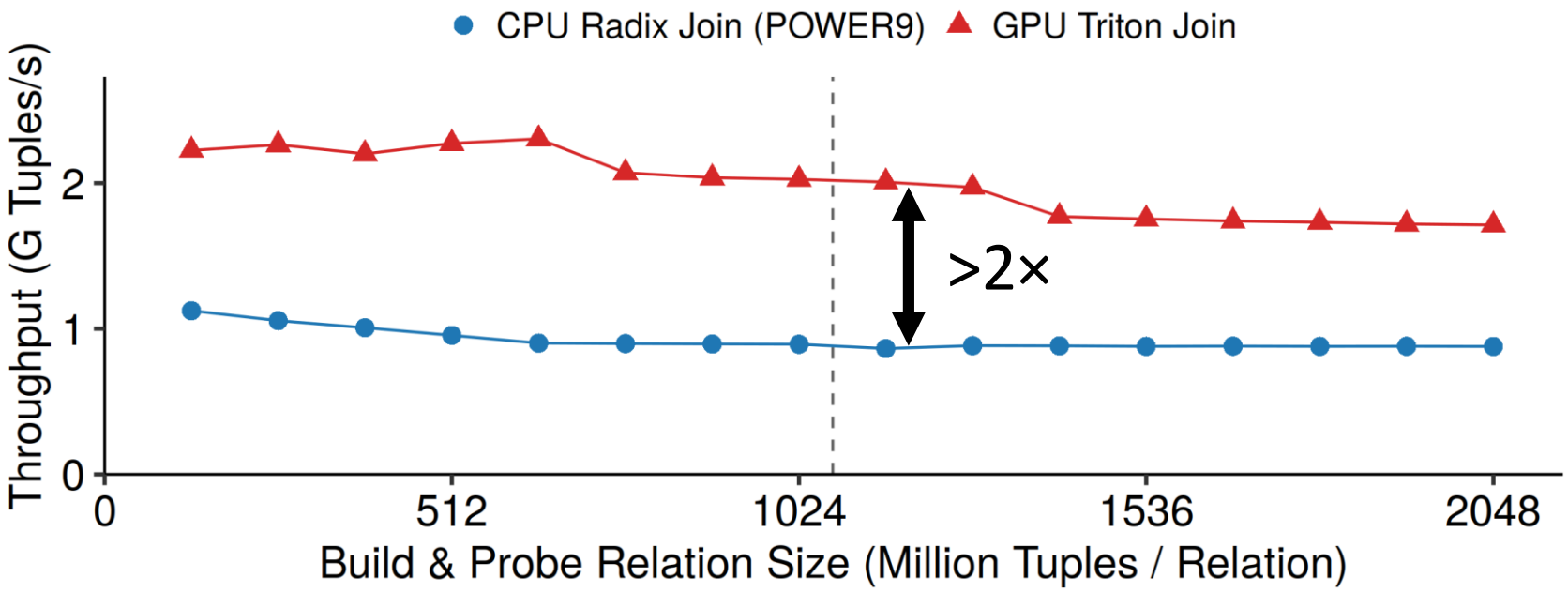


Triton join is new hierarchical hybrid hash join for GPUs

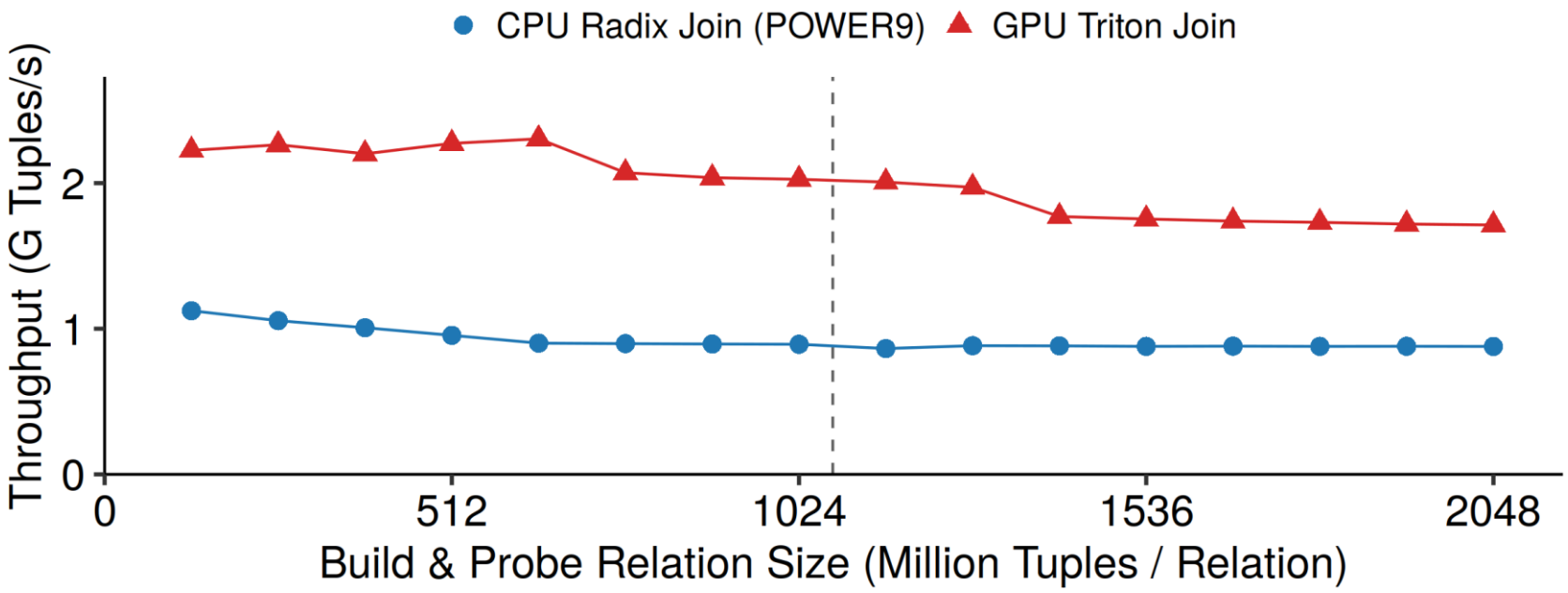
Evaluation: Scalability



Evaluation: Scalability

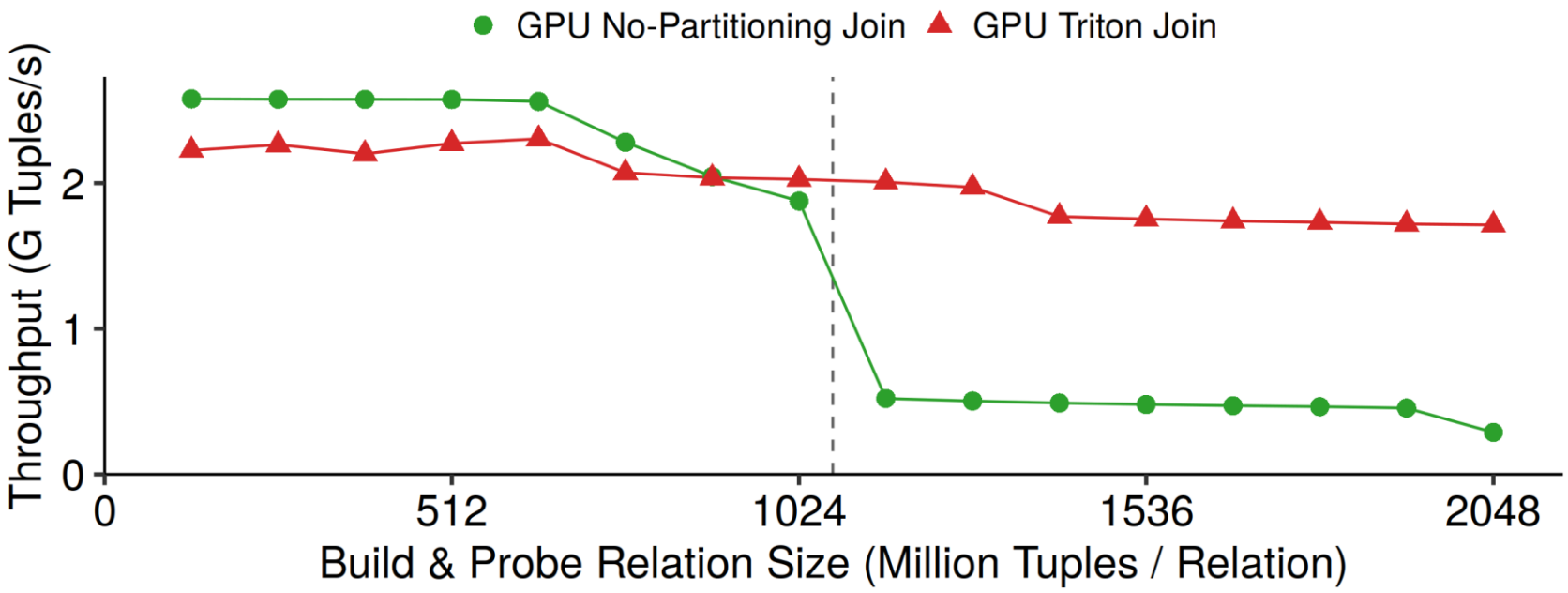


Evaluation: Scalability

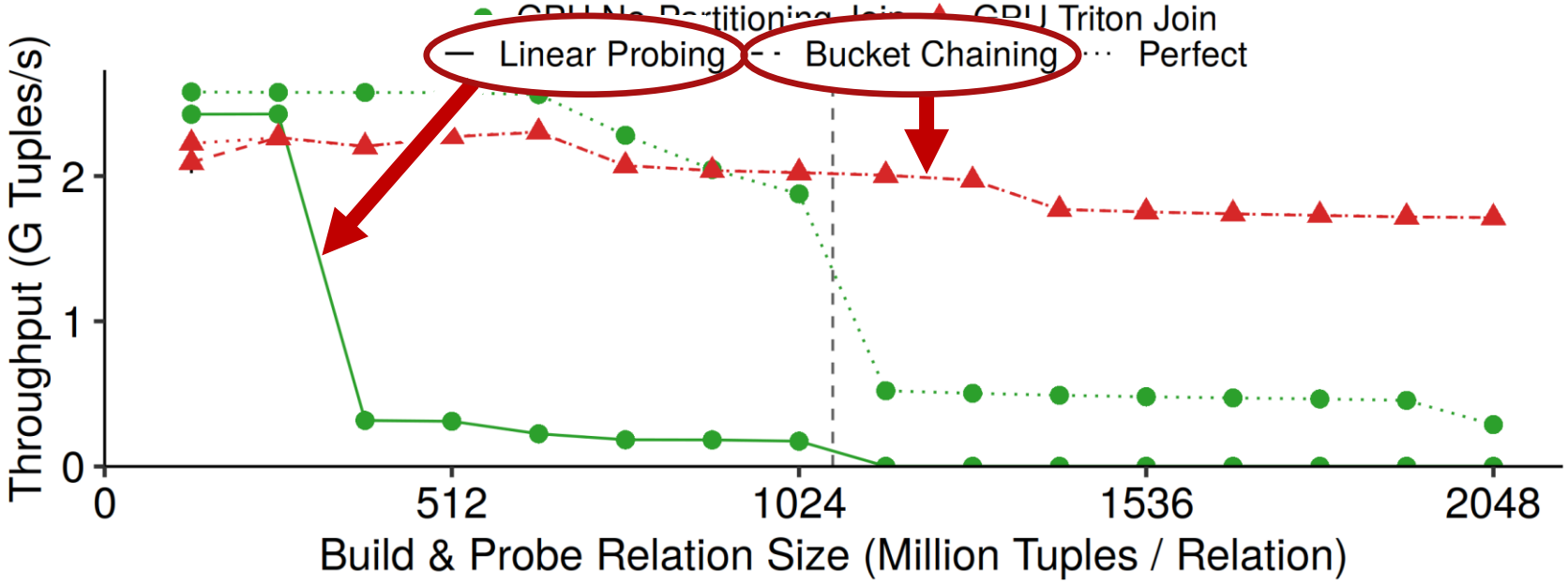


Triton join scales to a large join state

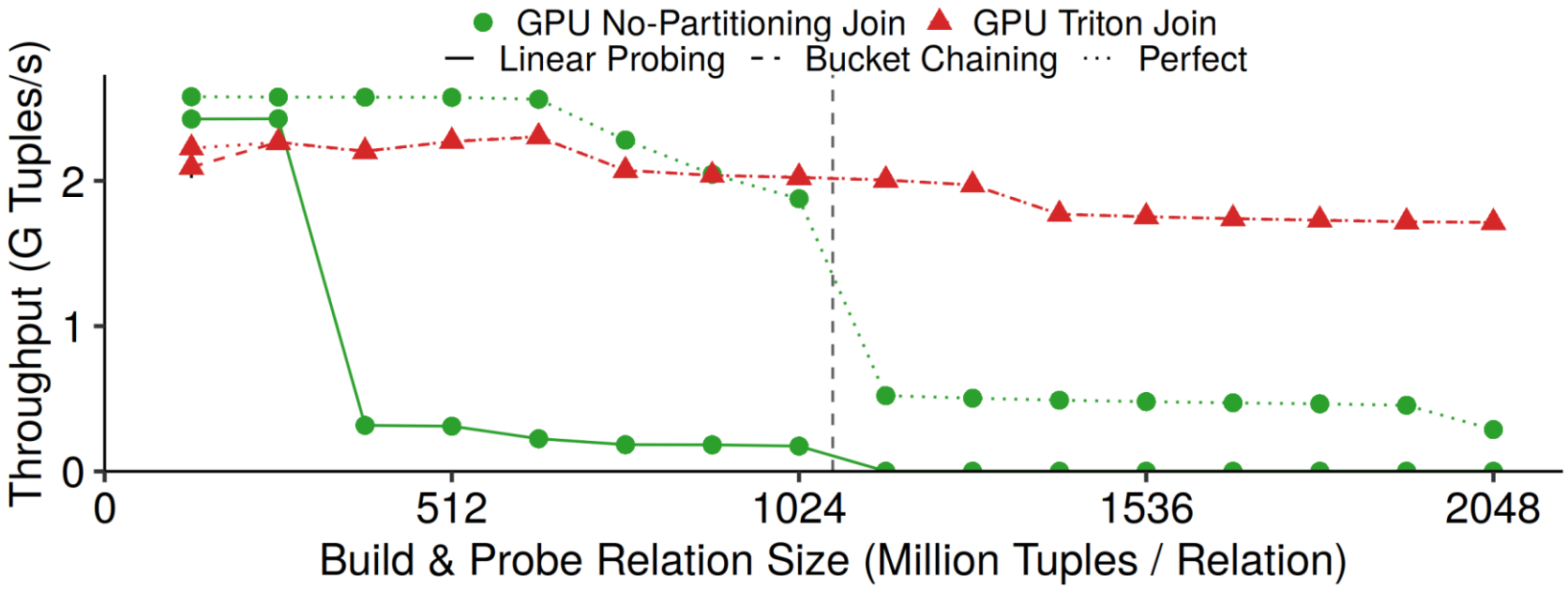
Evaluation: Robustness



Evaluation: Robustness

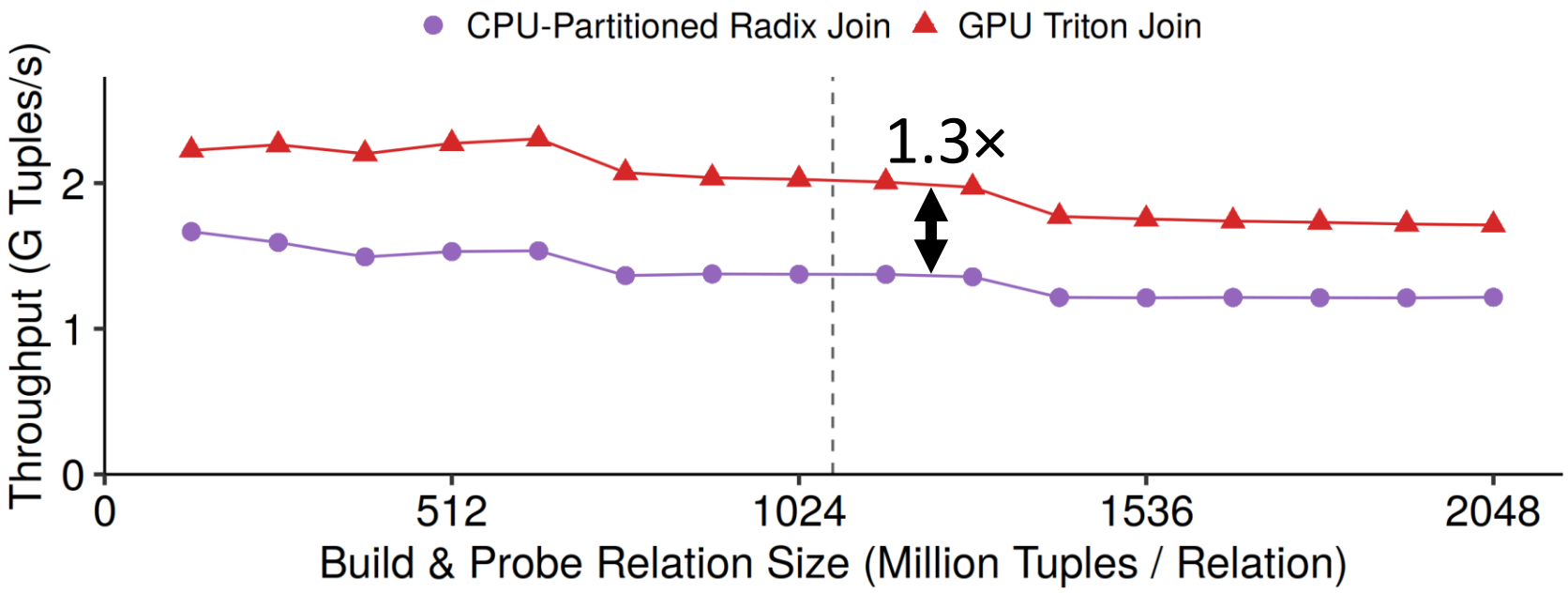


Evaluation: Robustness

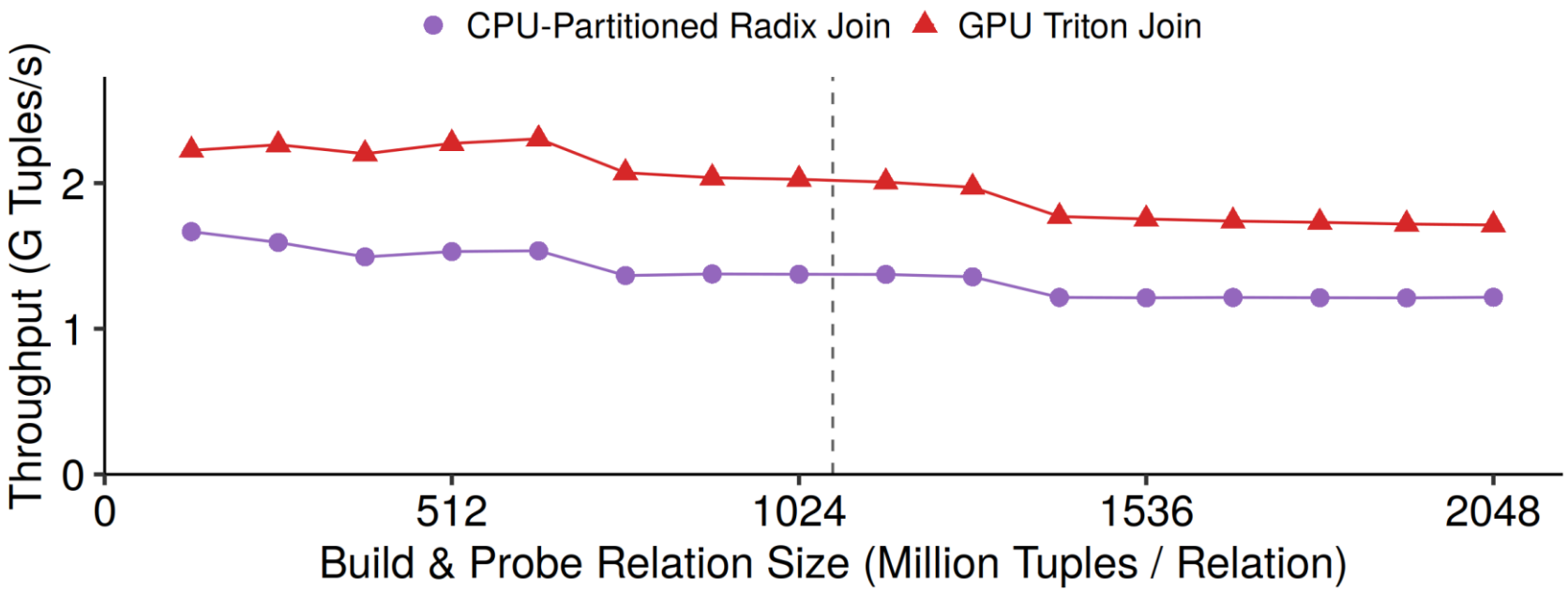


Performance of Triton join degrades gracefully

Evaluation: Resource-Efficiency

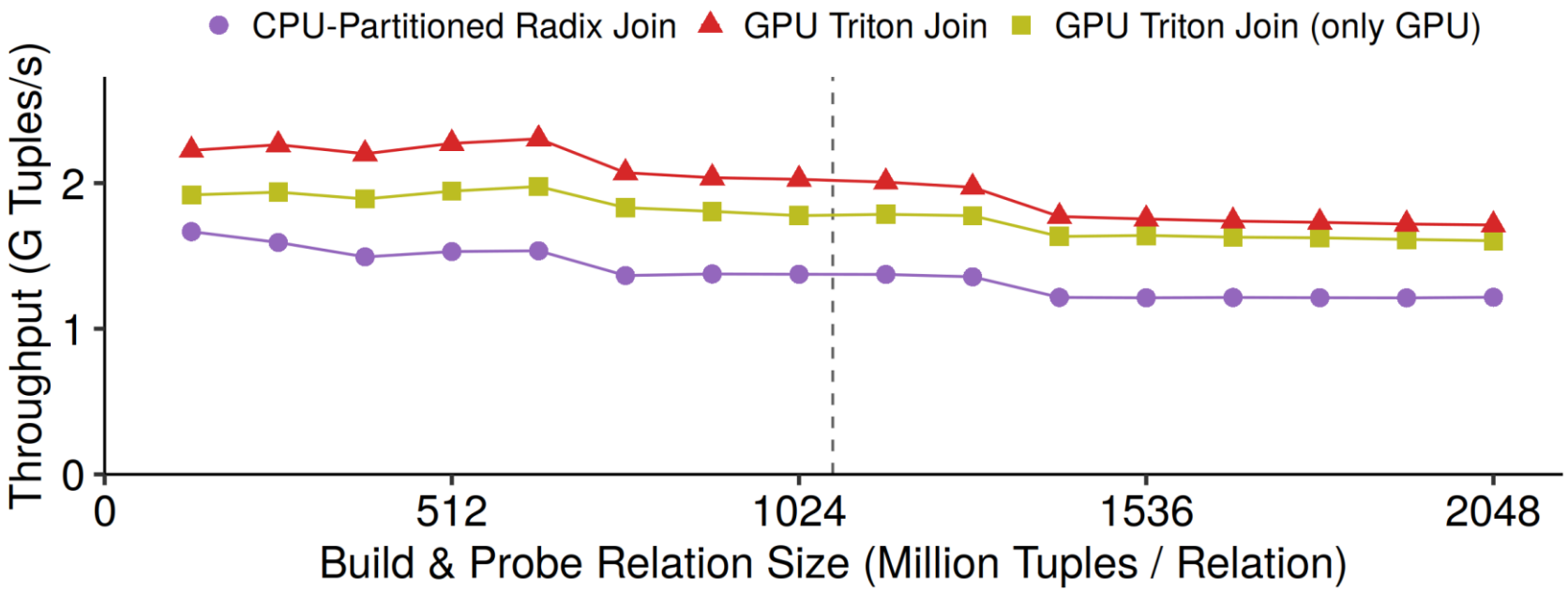


Evaluation: Resource-Efficiency



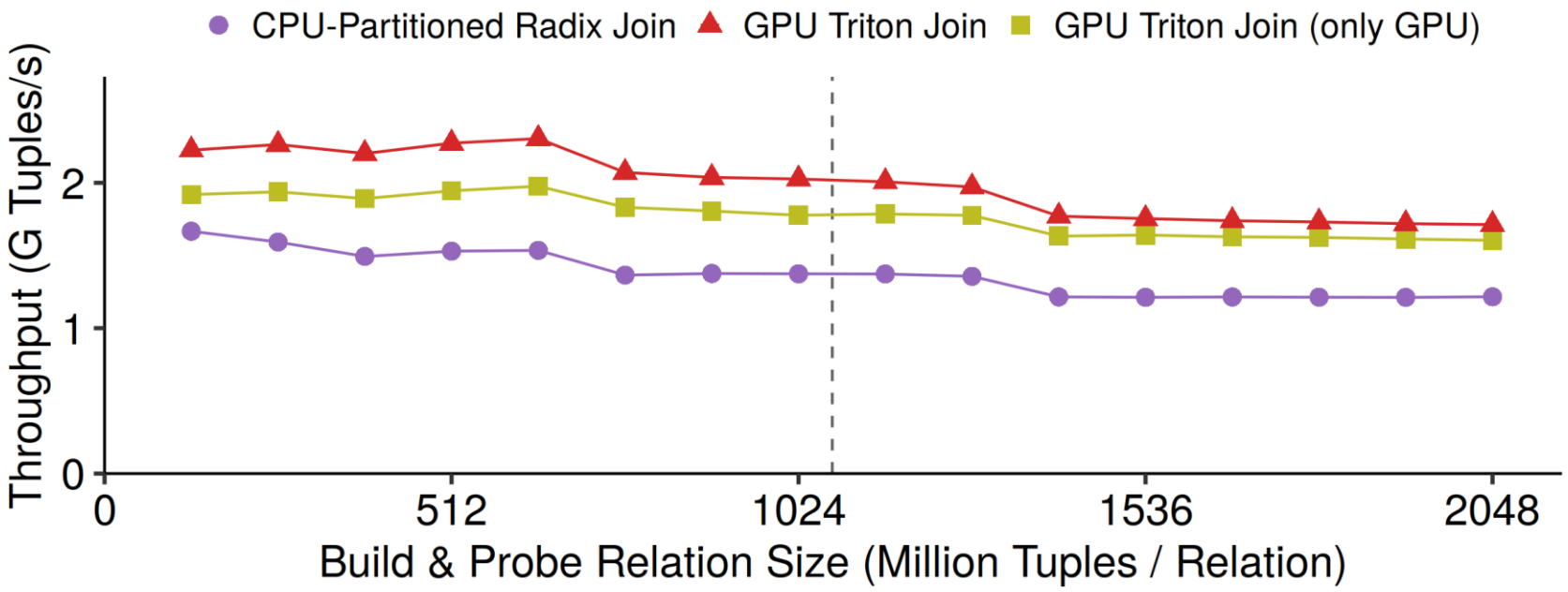
- CPU-Partitioned Radix Join: CPU + GPU
- GPU Triton Join: GPU only*

Evaluation: Resource-Efficiency



- CPU-Partitioned Radix Join: CPU + GPU
- GPU Triton Join: GPU only*

Evaluation: Resource-Efficiency



Triton join efficiently processes joins end-to-end on GPU

Conclusion

Our Triton join exploits fast interconnects to:

- **Scale** to a large **join state**
- **Robustly spill** state to main memory
- **Efficiently** process the join using **the GPU**

www.clemenslutz.com

github.com/TU-Berlin-DIMA/fast-interconnects