WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

CS 839: Design the Next-Generation Database

Lecture 19: RDMA for OLAP

Xiangyao Yu

3/31/2020

Discussion Highlights

SmartNIC vs. SmartSSD

- Different application scenarios: one for storage, one for network
- SATA vs. PCIe?
- SmartNICs used for reducing CPU overhead; SmartSSD used for reducing data movement
- SmartNIC seems more popular among hardware vendors
- Computation in SmartNIC is stronger than SmartSSD

Database operators pushed to SmartNIC

- Common: encryption, caching
- OLTP: filtering, aggregation, locking, indexing
- OLAP: filtering, project, aggregation, compression

Benefits of putting smartness into the NIC

- Packet processing, latency reduction
- Effect of SmartSSD is limited due to caching; caching does not apply in SmartNIC
- Isolate security checks from CPU
- Collect run time statistics such as network usage and latencies
- Reduces burden on PCIe

Today’s Paper

Distributed Join Algorithms on Thousands of Cores

Claude Barthels, Ingo Müller[‡], Timo Schneider, Gustavo Alonso, Torsten Hoefler
Systems Group, Department of Computer Science, ETH Zurich
{firstname.lastname}@inf.ethz.ch

ABSTRACT

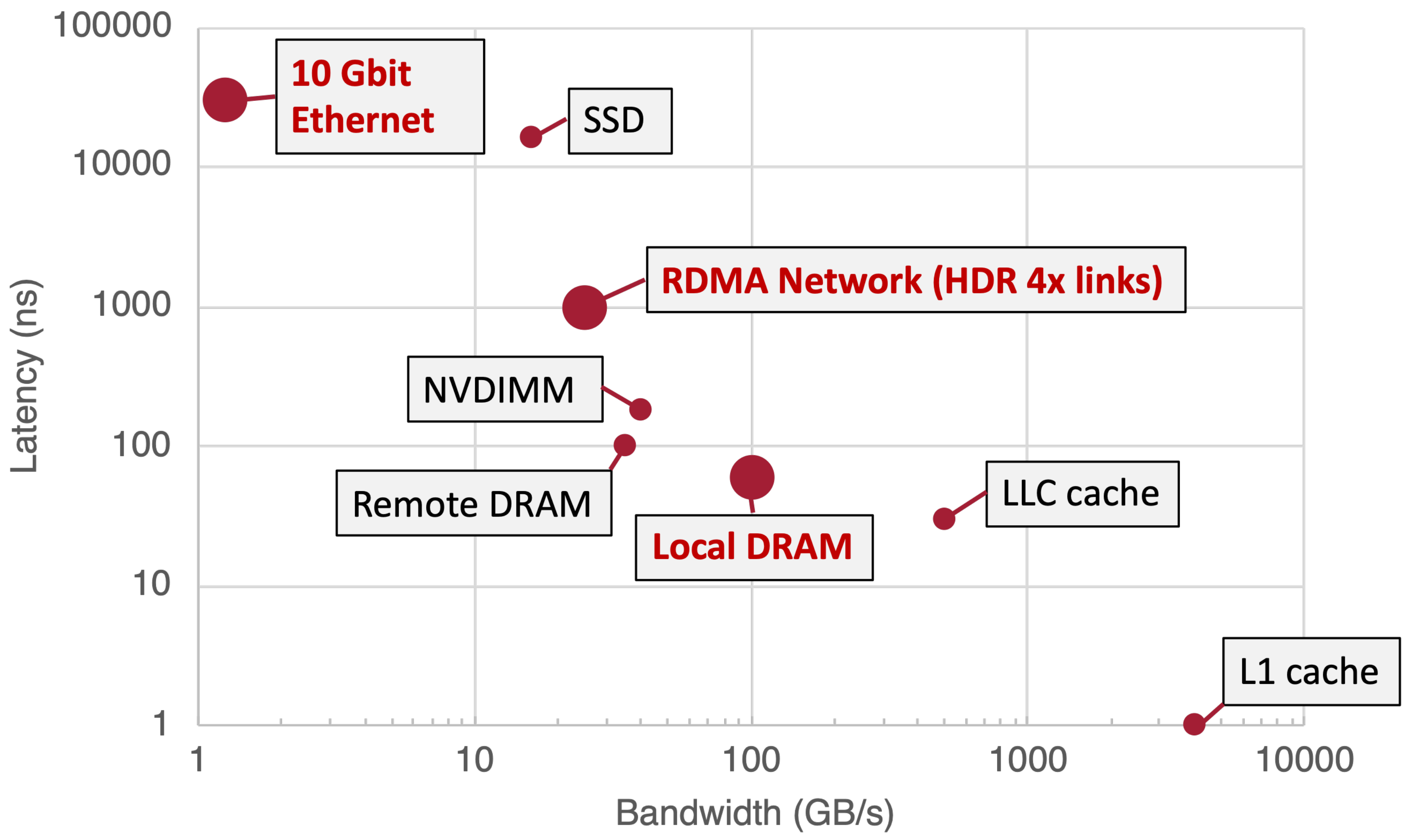
Traditional database operators such as joins are relevant not only in the context of database engines but also as a building block in many computational and machine learning algorithms. With the advent of big data, there is an increasing demand for efficient join algorithms that can scale with the input data size and the available hardware resources.

In this paper, we explore the implementation of distributed join algorithms in systems with several thousand cores connected by a low-latency network as used in high performance computing systems or data centers. We compare radix hash join to sort-merge join algorithms and discuss their imple-

This paper addresses the challenges of running state-of-the-art, distributed radix hash and sort-merge join algorithms at scales usually reserved to massively parallel scientific applications or large map-reduce batch jobs. In the experimental evaluation, we provide a performance analysis of the distributed joins running on 4,096 processor cores with up to 4.8 terabytes of input data. We explore how join algorithms behave when high-bandwidth, low-latency networks are used and specialized communication libraries replace hand-tuned code. These two points are crucial to understand the evolution of distributed joins and to facilitate the portability of the implementation to future systems.

VLDB 2017

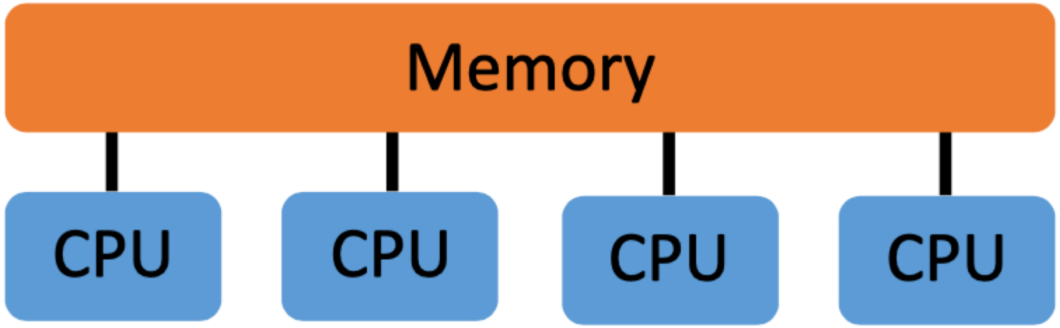
Bandwidth and Latency



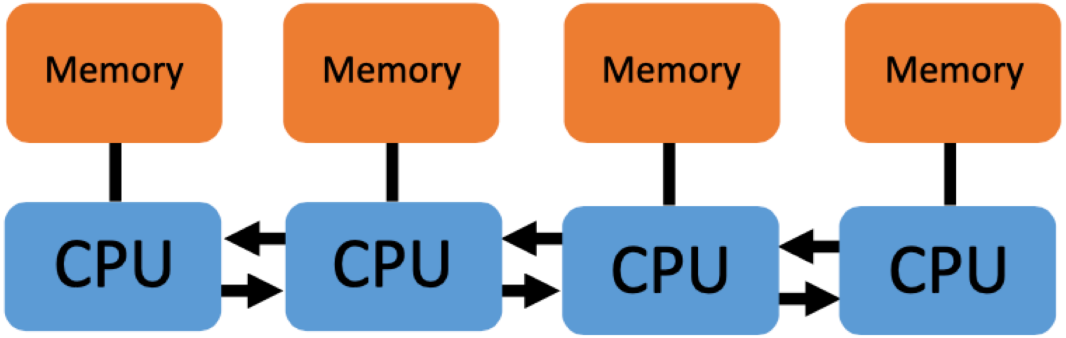
Algorithm Designs

	Shared Memory	RDMA & SmartNIC	Distributed System
Concurrency Control	Shared lock table	???	Partitioned lock table
Fault Tolerance	Shared log	???	Two-phase commit
Join	Radix join	???	Bloom-filter + semi-join

Message Passing



Shared memory



Message Passing

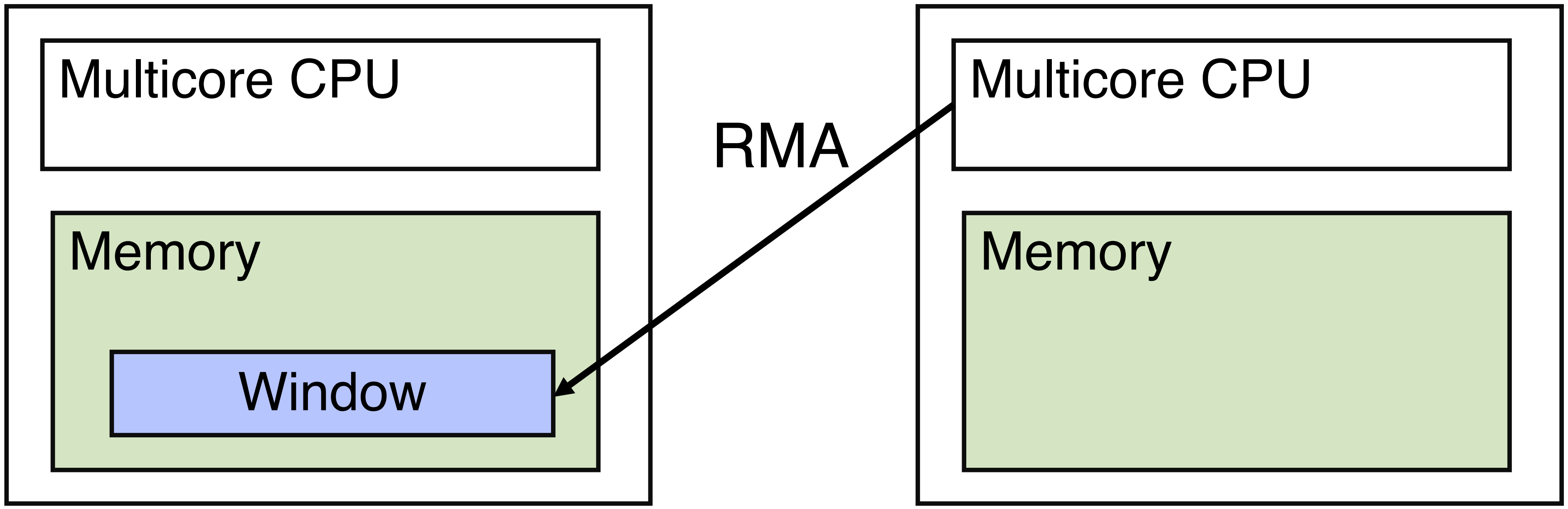
Message Passing Interface (MPI)

Standard library interface for writing parallel programs in high-performance computing (HPC)

- Hardware independent interface
- Can leverage performance of underlying hardware

MPI One-Sided Operations

Memory Window: memory that is accessible by other processes through RMA operations



MPI One-Sided Operations

MPI_Win_create: exposes local memory to RMA operation by other processes.

- Collective operation
- Creates window object

MPI_Win_free: deallocates window object

MPI_Put: moves data from local memory to remote memory

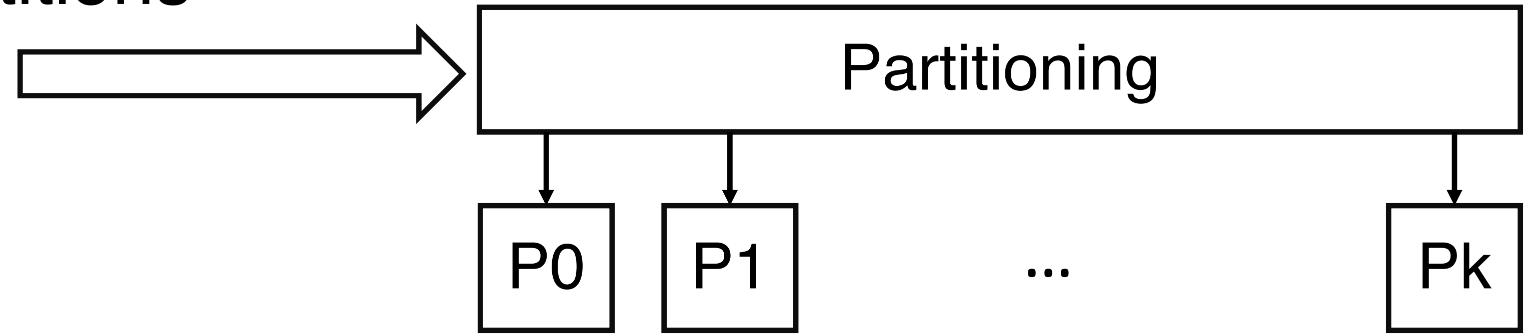
MPI_Get: retrieves data from remote memory into local memory

MPI_Win_lock and **MPI_Win_unlock** to protect RMA operations on a specific window

Radix Hash Join

Partitioned hash join achieves the best performance when each partition of the inner relation fits in cache

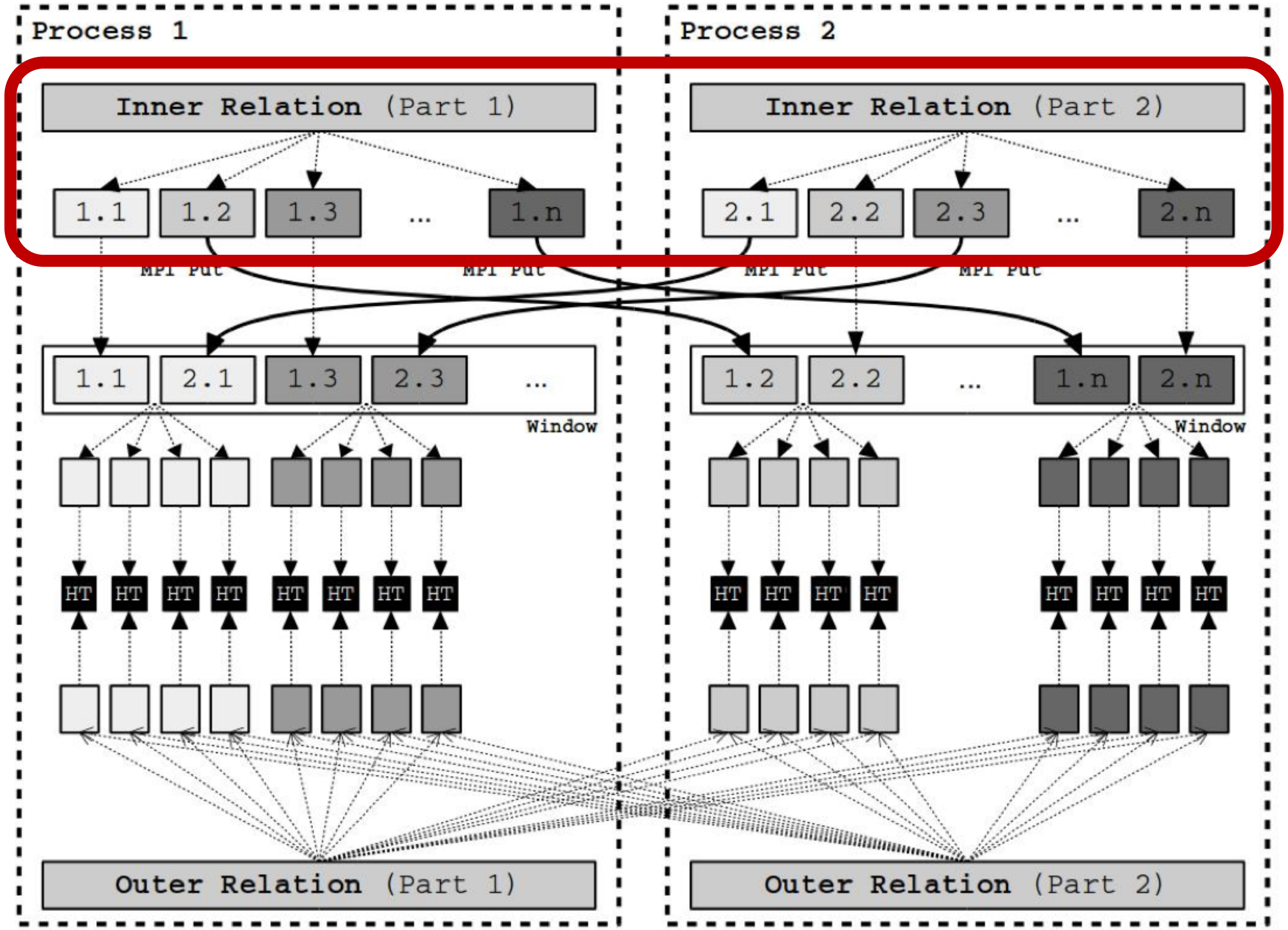
⇒ A large number of partitions



⇒ Performance suffers when the # partitions > # TLB entries or # of cachelines in the cache

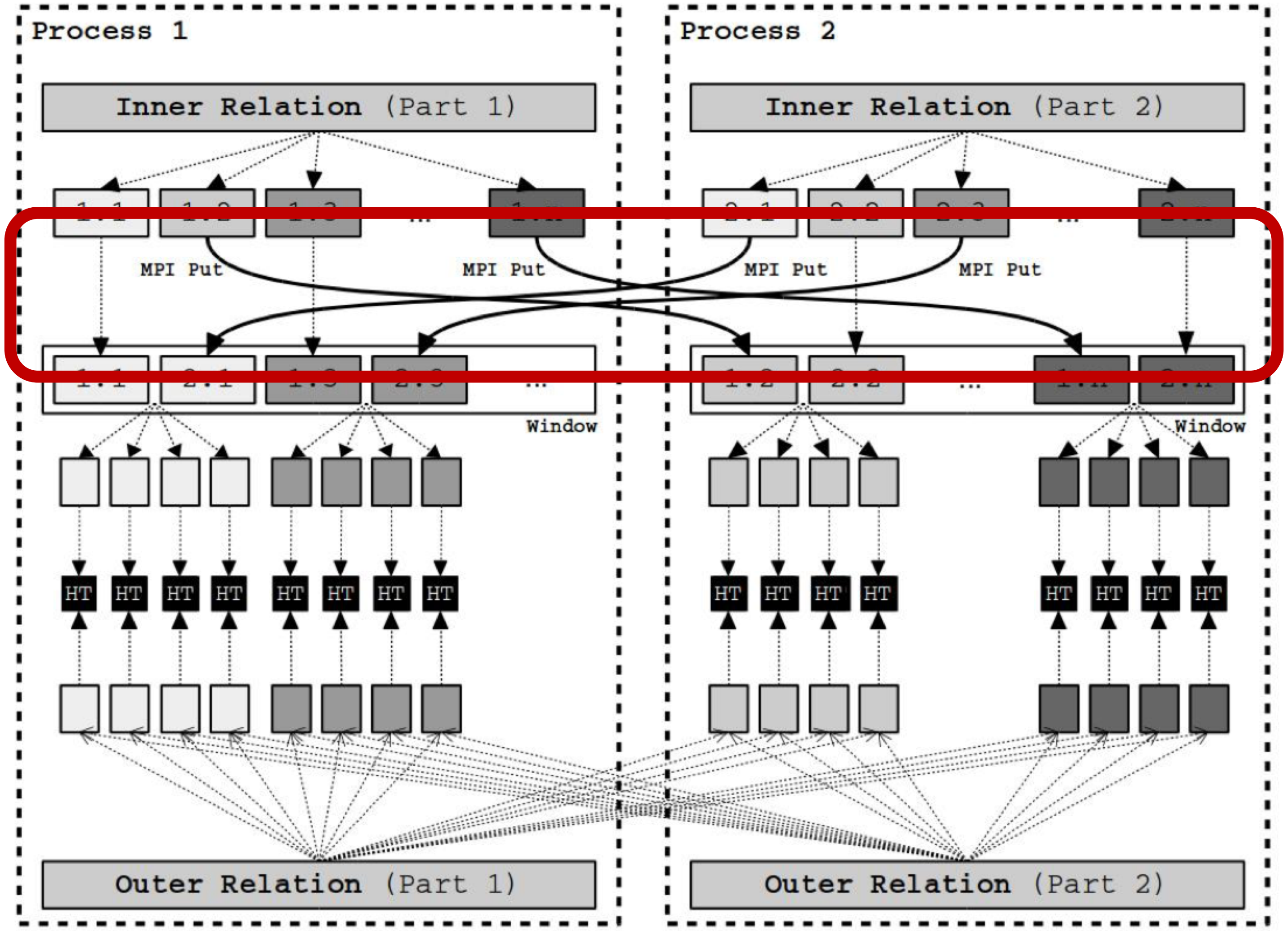
Radix Join: Partition through multiple passes

Radix Hash Join



1st pass of partitioning

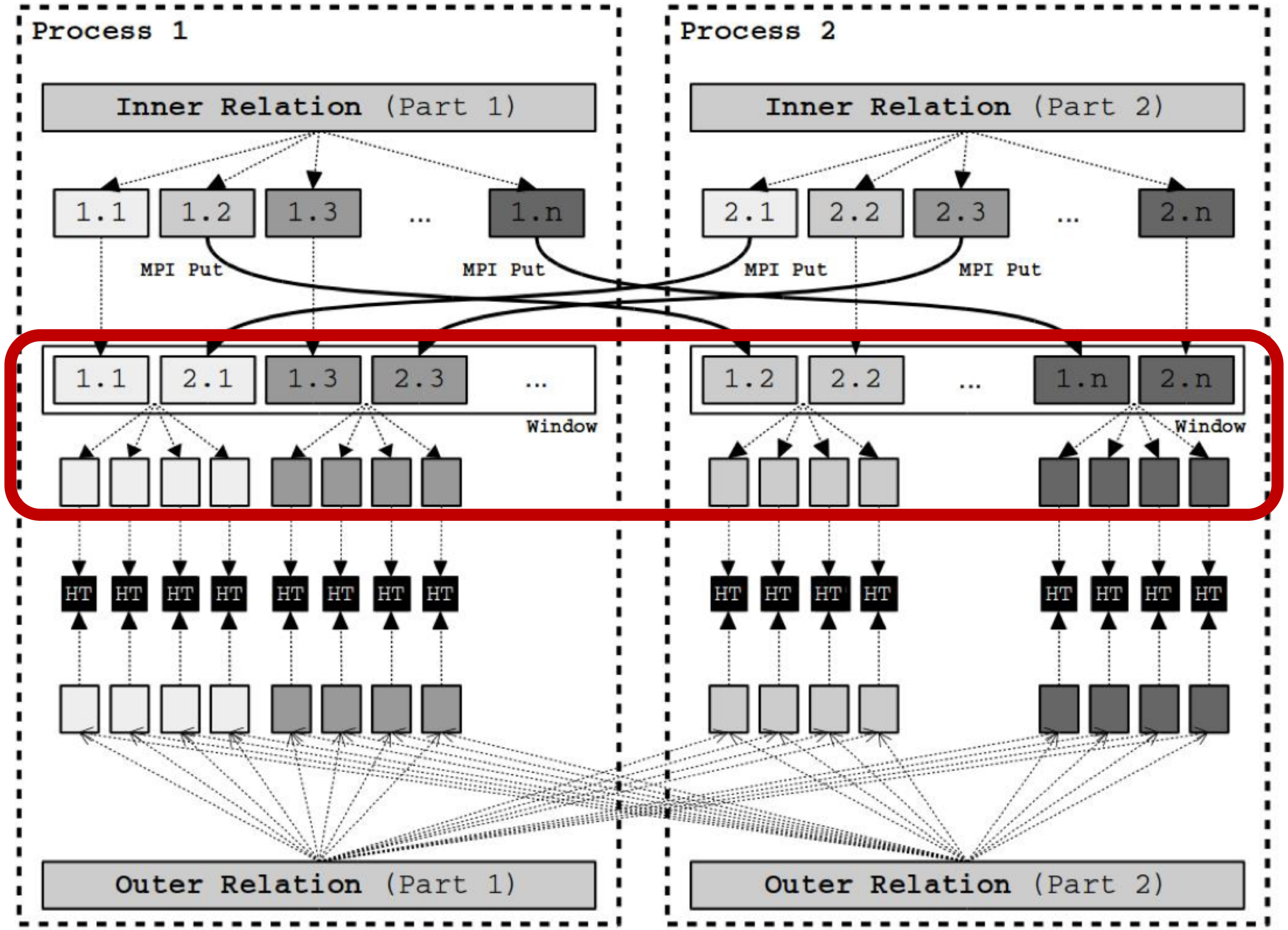
Radix Hash Join



1st pass of partitioning

Data shuffle

Radix Hash Join

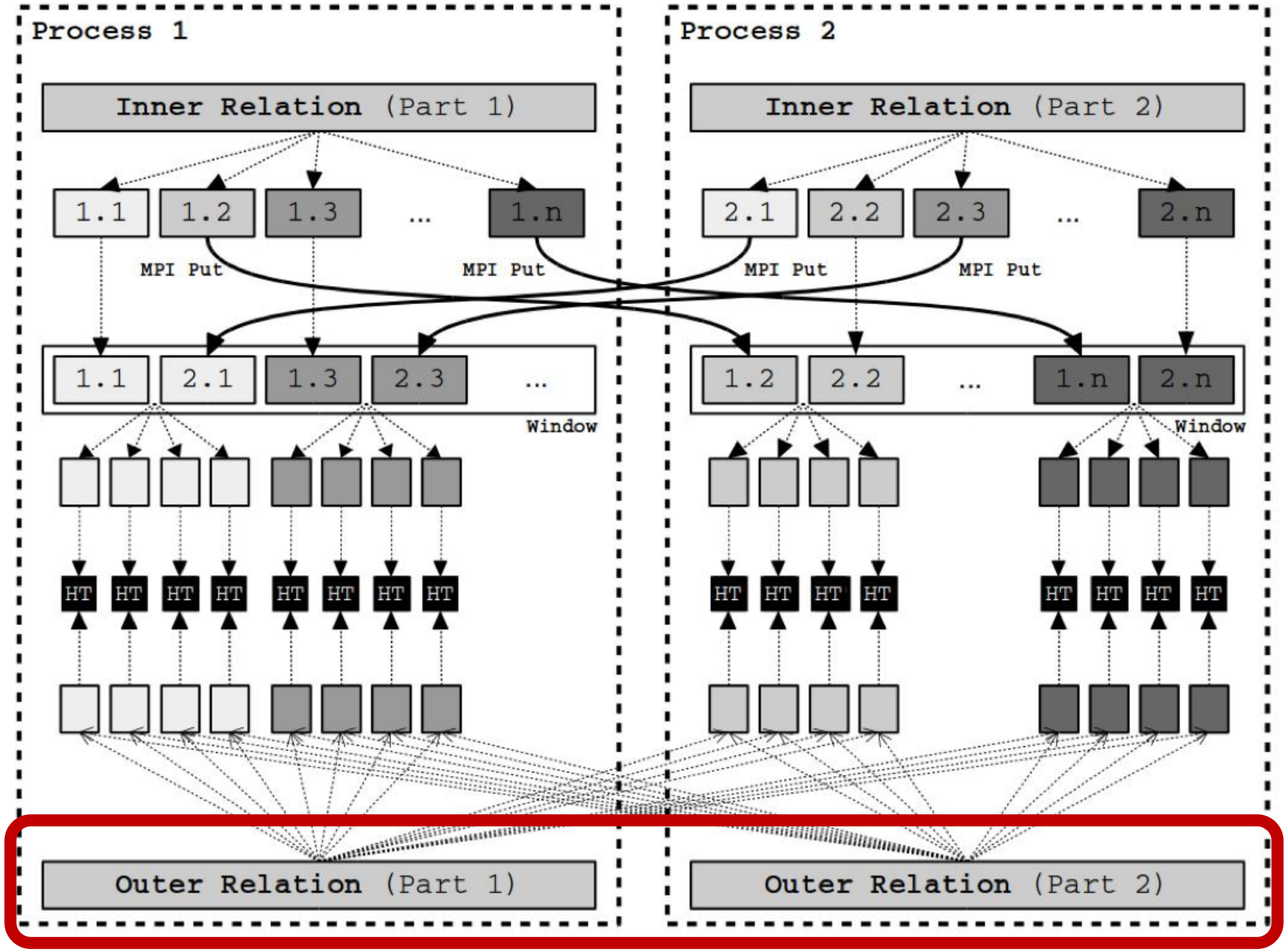


1^{st} pass of partitioning

Data shuffle

Following passes of partitioning

Radix Hash Join



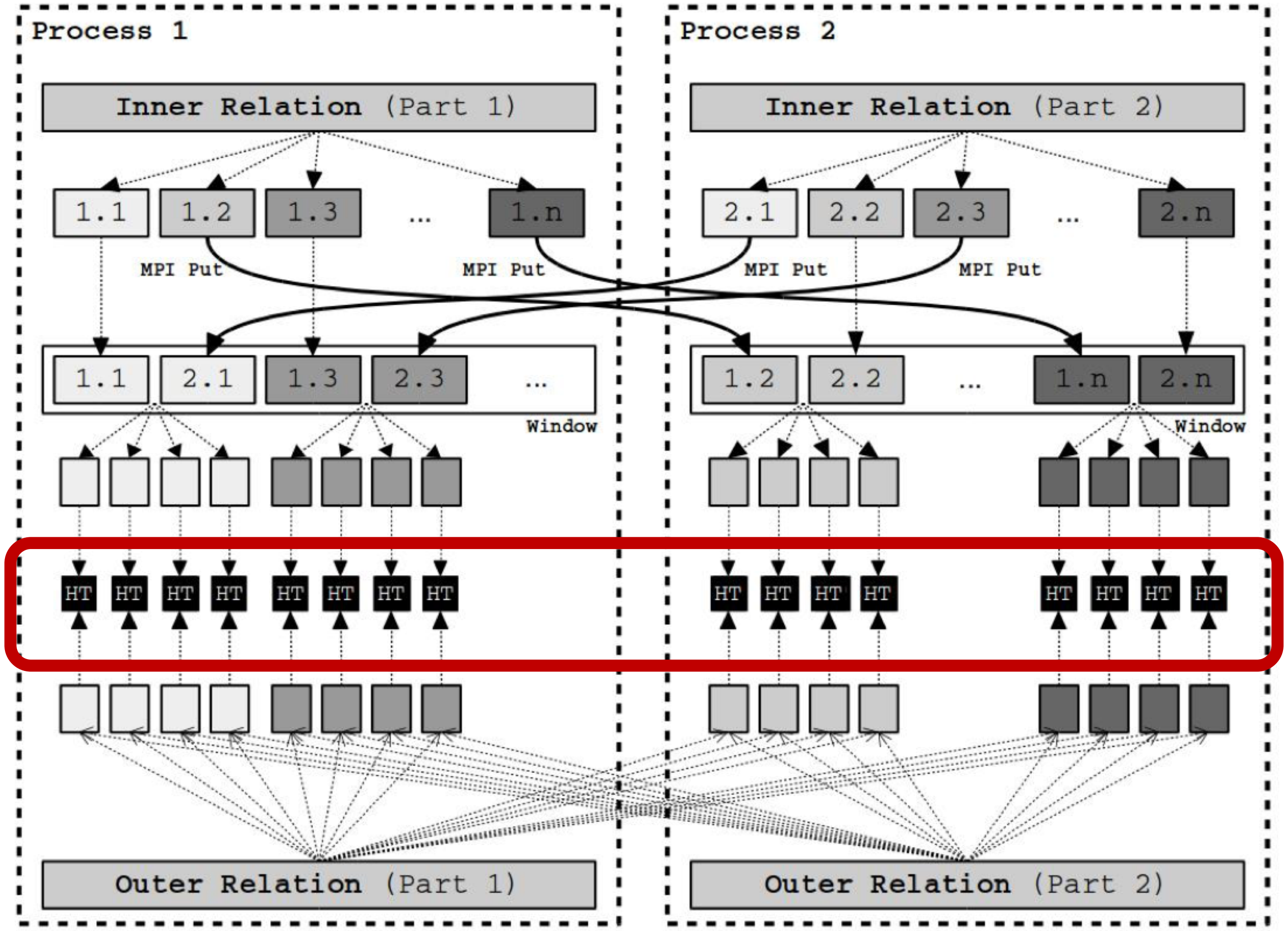
1st pass of partitioning

Data shuffle

Following passes of partitioning

Partition outer relation

Radix Hash Join



1st pass of partitioning

Data shuffle

Following passes of partitioning

Build and probe

Partition outer relation

Radix Hash Join – Performance Model

Compute the histogram

- Determine the size of memory windows
- Assignment of partitions to nodes
- Offsets within memory windows into which each process writes exclusively

$$T_{\text{hist}} = \frac{|R| + |S|}{p \cdot P_{\text{scan}}}$$

Radix Hash Join – Performance Model

Multi-pass partitioning

Number of passes

$$d = \left\lceil \log_{F_P} \left(|R| / \text{cache size} \right) \right\rceil$$

F_P : partitioning fan-out

Time of partitioning

$$T_{\text{part}} = \left(\frac{1}{p \cdot P_{\text{net}}} + \frac{d-1}{p \cdot P_{\text{part}}} \right) \cdot (|R| + |S|)$$

$$P_{\text{net}} = \min \left(P_{\text{part}}, \frac{BW_{\text{node}}}{t} \right)$$

Radix Hash Join – Performance Model

Build and Probe

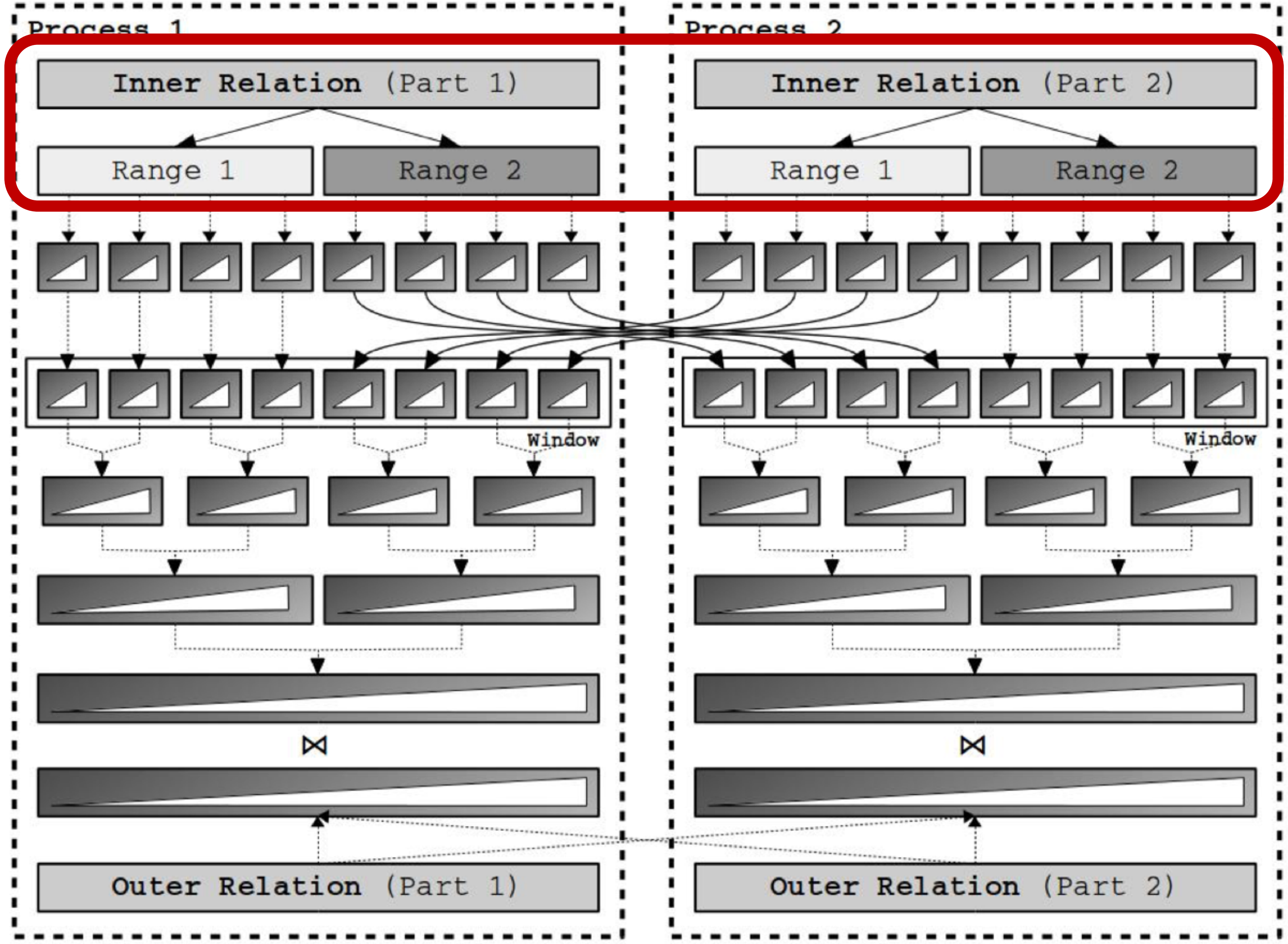
Build Time $$T_{\text{build}} = (F_P)^d \cdot \frac{|R_p|}{p \cdot P_{\text{build}}} = \frac{|R|}{p \cdot P_{\text{build}}}$$

Probe Time $$T_{\text{probe}} = (F_P)^d \cdot \frac{|S_p|}{p \cdot P_{\text{probe}}} = \frac{|S|}{p \cdot P_{\text{probe}}}$$

Radix Hash Join – Performance Model

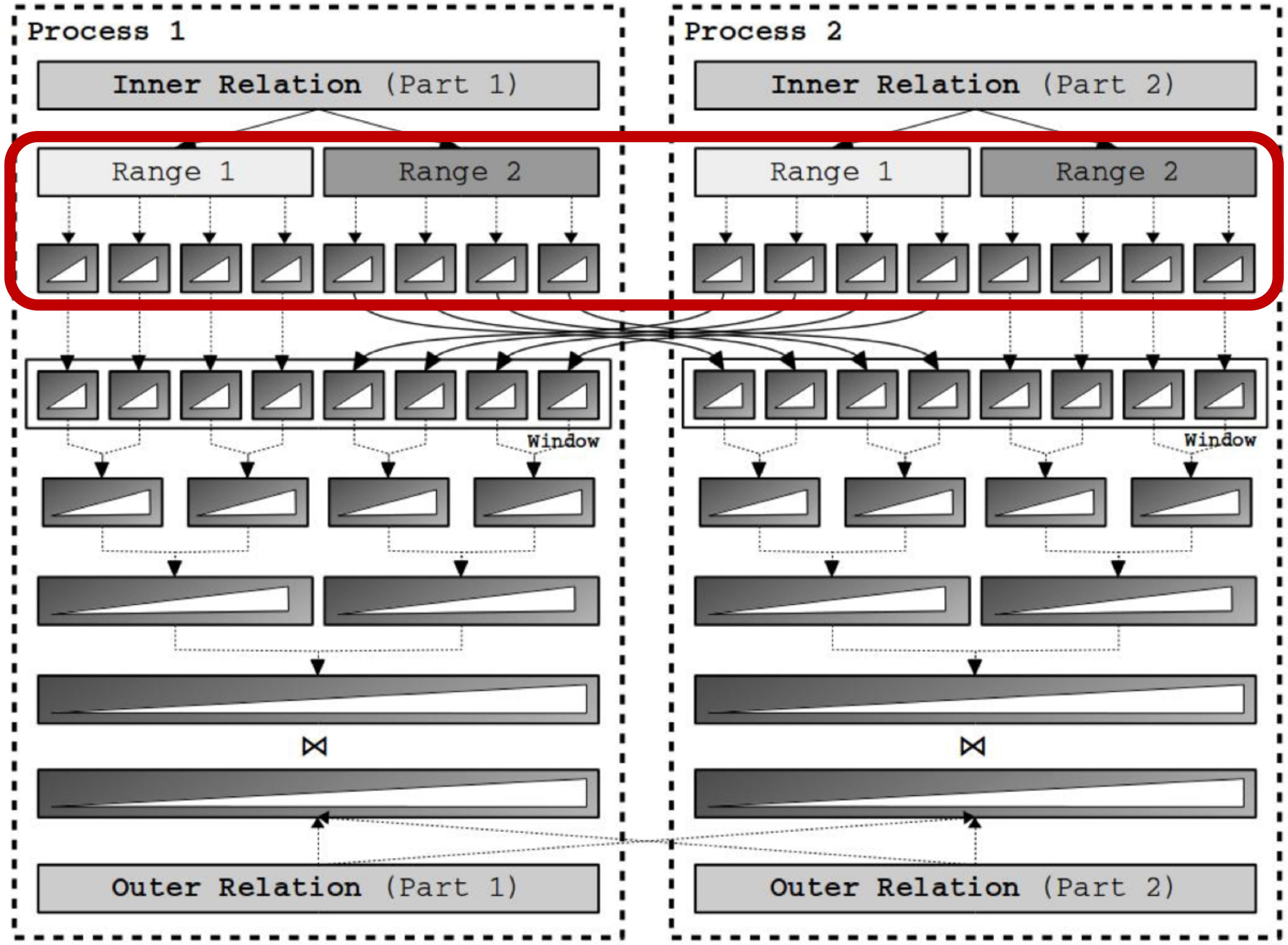
$$
\begin{aligned}
T_{\mathrm{rdx}} &= T_{\mathrm{hist}} + T_{\mathrm{part}} + T_{\mathrm{build}} + T_{\mathrm{probe}} \\
&= \frac{|R| + |S|}{p \cdot P_{\mathrm{scan}}} \\
&+ \left(\frac{1}{p \cdot P_{\mathrm{net}}} + \frac{d-1}{p \cdot P_{\mathrm{part}}} \right) \cdot (|R| + |S|) \\
&+ \frac{|R|}{p \cdot P_{\mathrm{build}}} \\
&+ \frac{|S|}{p \cdot P_{\mathrm{probe}}}
\end{aligned}
$$

Sort-Merge Join



Range partitioning

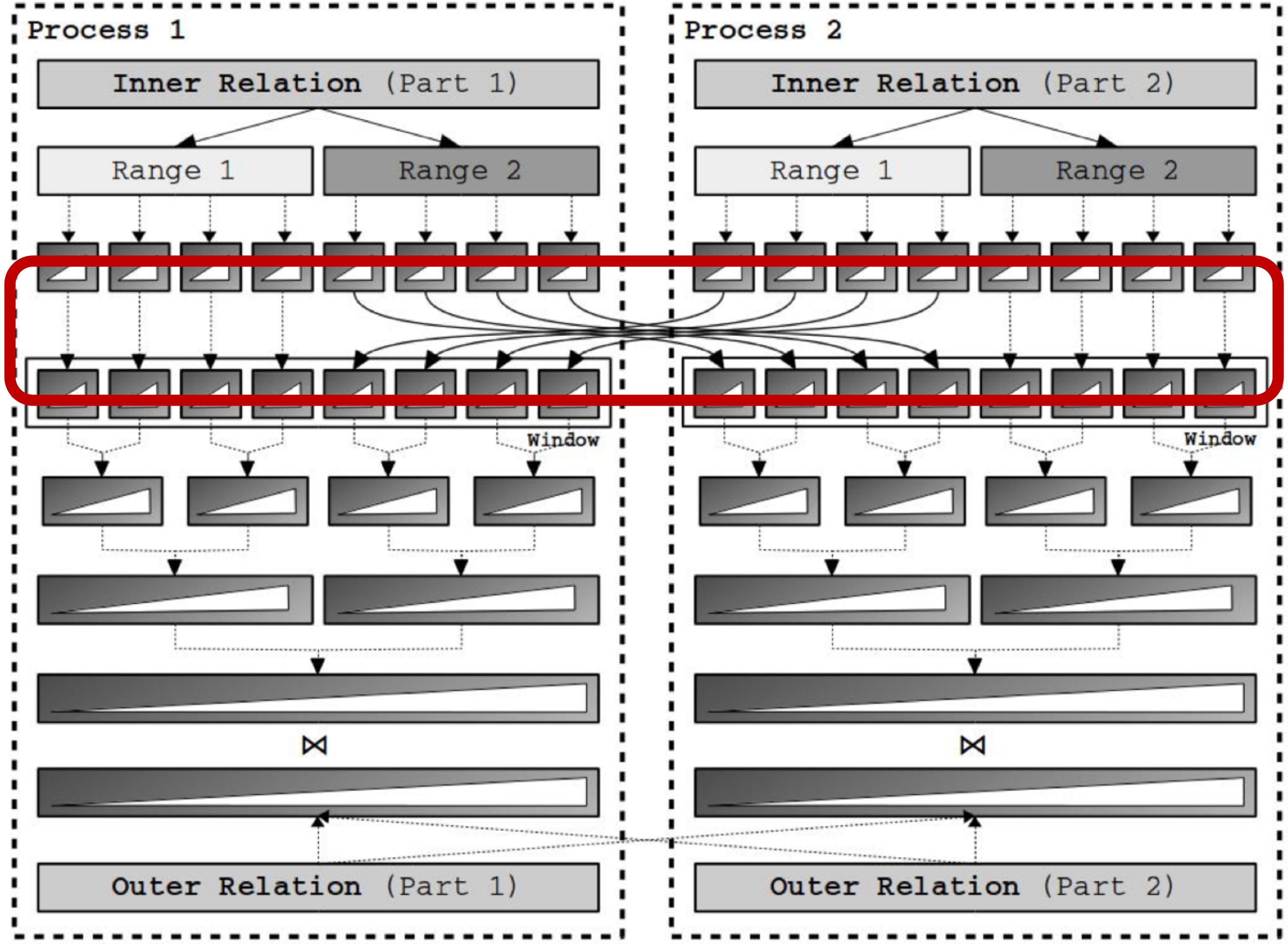
Sort-Merge Join



Range partitioning

Sort individual runs

Sort-Merge Join

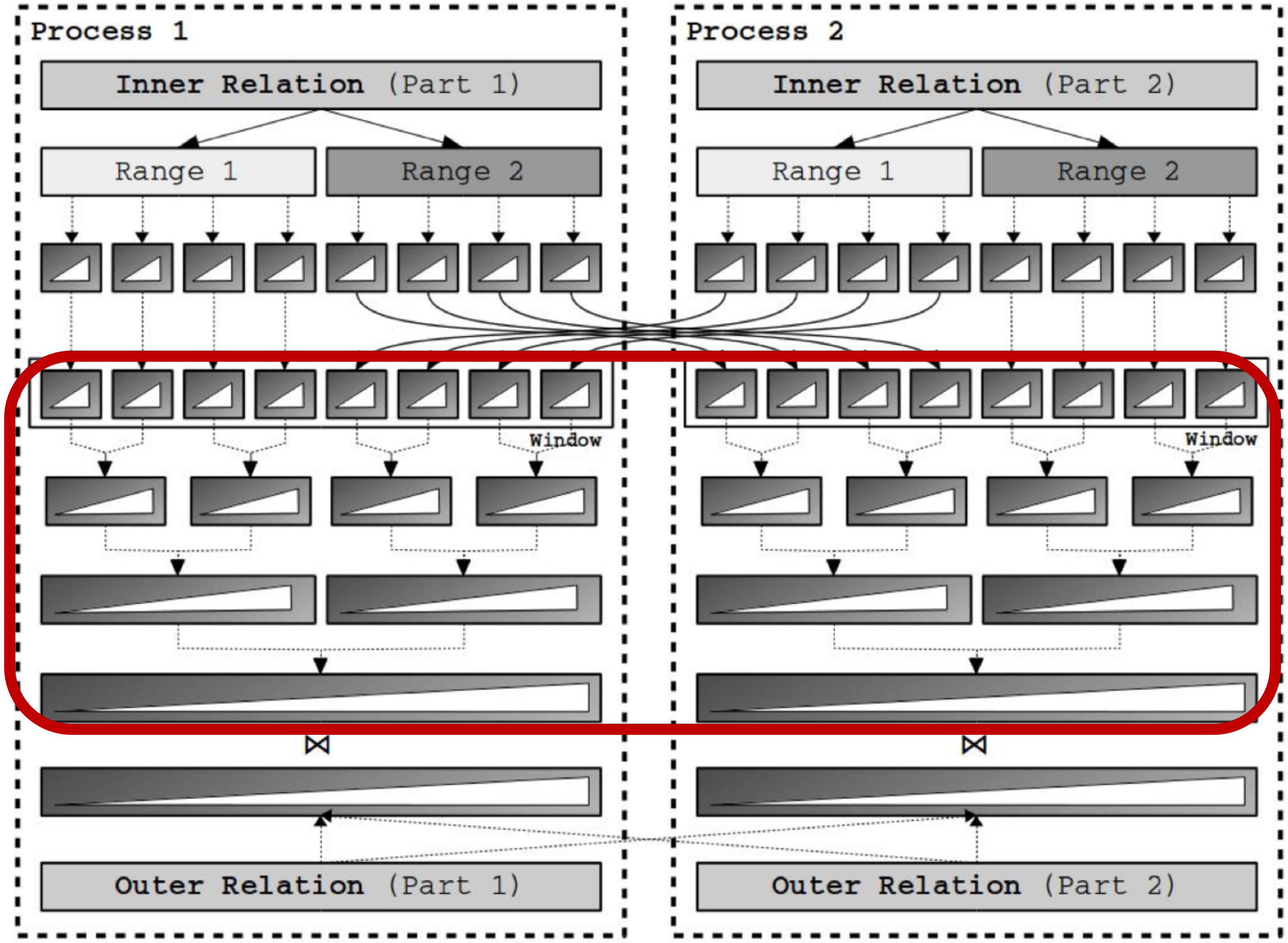


Range partitioning

Sort individual runs

Data shuffle

Sort-Merge Join



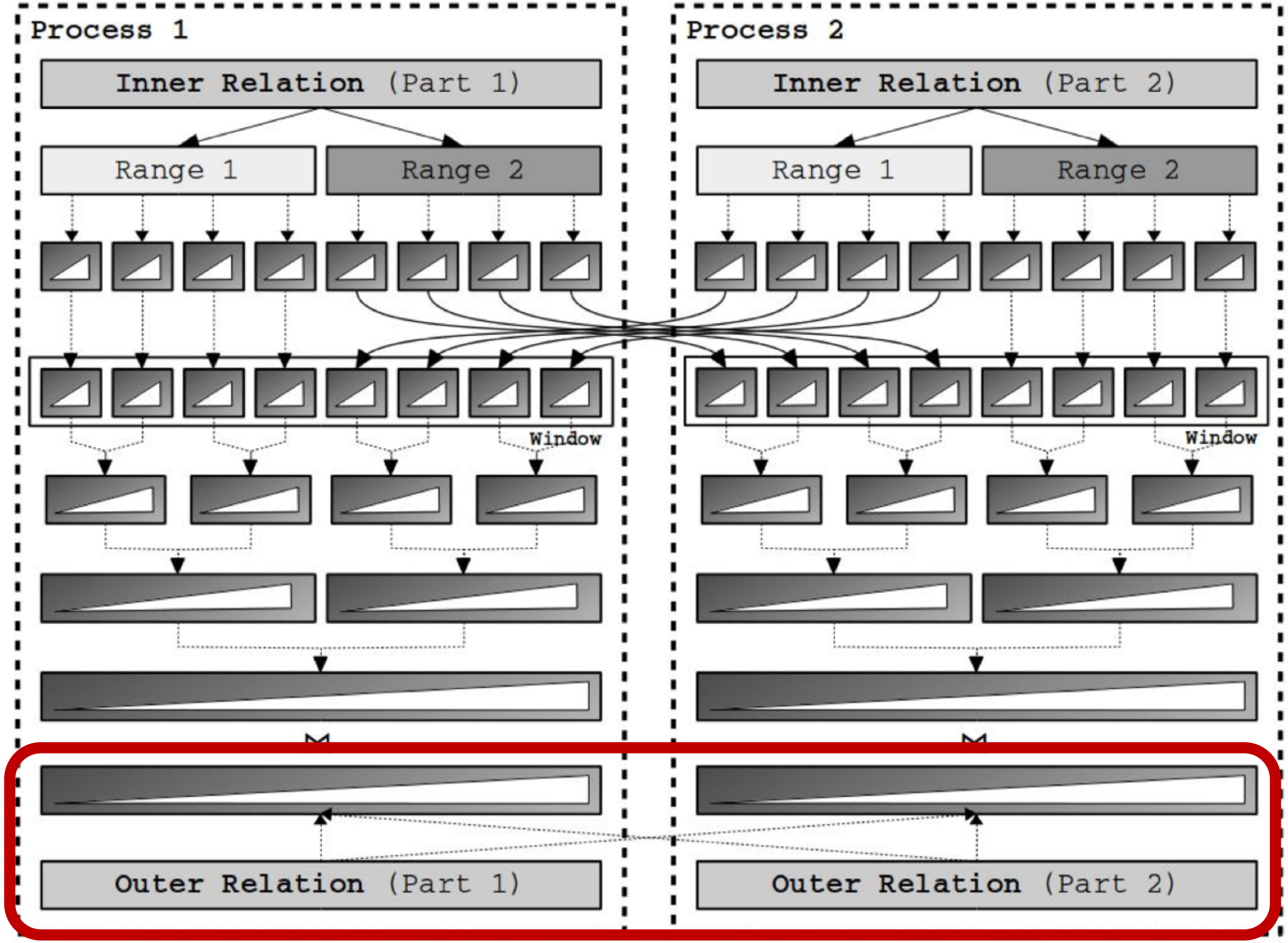
Range partitioning

Sort individual runs

Data shuffle

Merge

Sort-Merge Join



Range partitioning

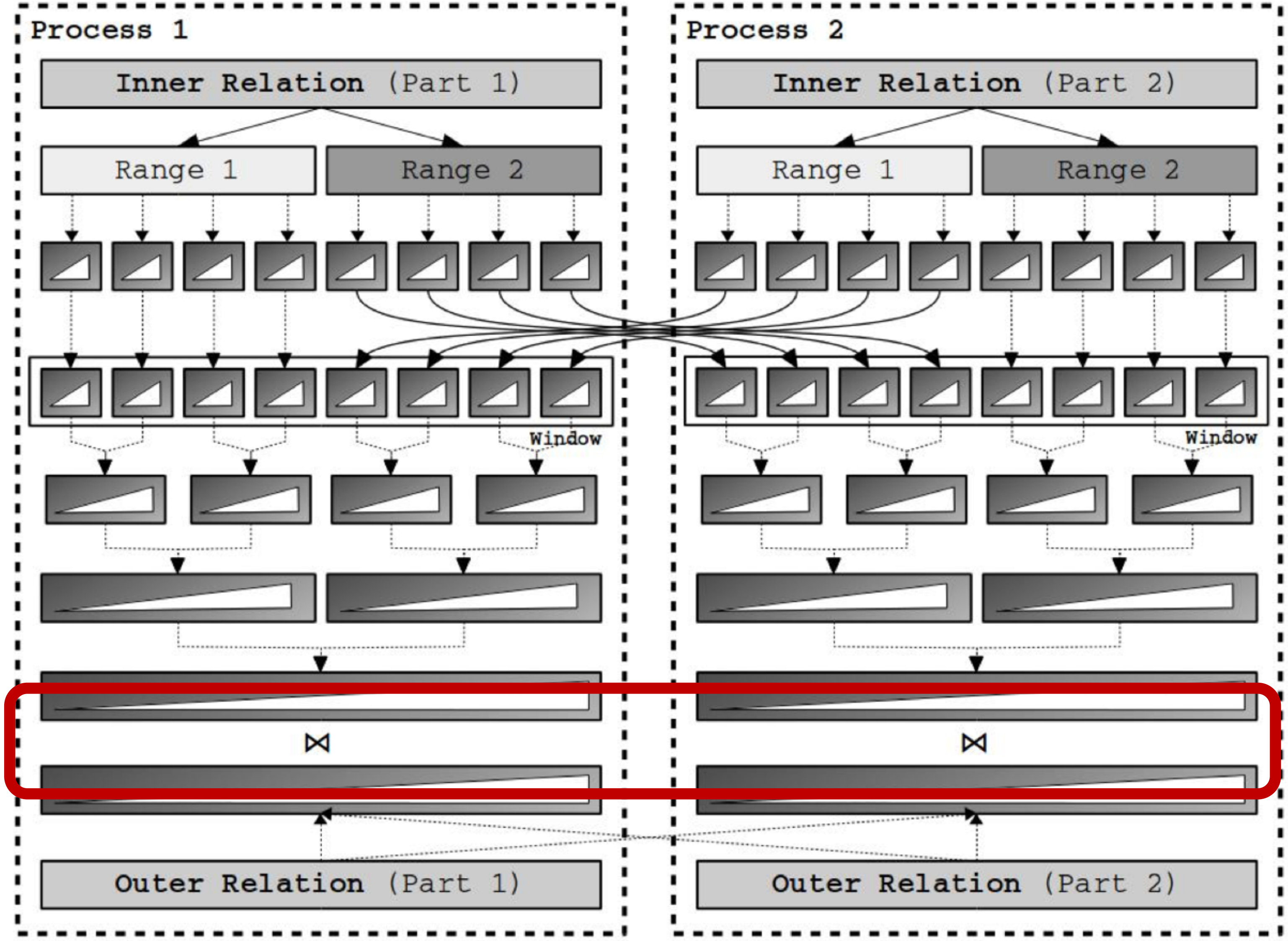
Sort individual runs

Data shuffle

Merge

Sort-merge outer relation

Sort-Merge Join



Range partitioning

Sort individual runs

Data shuffle

Merge

Join

Sort-merge outer relation

Sort-Merge Join – Performance Model

Partitioning

$$T_{\text{part}} = \frac{|R| + |S|}{p \cdot P_{\text{part}}}$$

Sort-Merge Join – Performance Model

Sorting individual runs of length *l*

Number of runs $$N_R = \frac{|R|}{l} \quad \text{and} \quad N_S = \frac{|S|}{l}$$

Sorting performance $$P_{\text{sort}} = \min\left(P_{\text{run}}(l), \frac{BW_{\text{node}}}{t}\right)$$

Sorting time $$T_{\text{sort}} = (N_R + N_S) \cdot \frac{l}{p \cdot P_{\text{sort}}} = \frac{|R| + |S|}{p \cdot P_{\text{sort}}}$$

Sort-Merge Join – Performance Model

Merging multiple runs into a sorted output

Number of iterations $d_R = \lceil \log_{F_M}(N_R/p) \rceil \quad \textbf{and} \quad d_S = \lceil \log_{F_M}(N_S/p) \rceil$

F_M : Merge fan-in

Merge time $$T_{\text{merge}} = d_R \cdot \frac{|R|}{p \cdot P_{merge}} + d_S \cdot \frac{|S|}{p \cdot P_{merge}}$$

Sort-Merge Join – Performance Model

Joining sorted relations

$$T_{\text{match}} = \frac{|R| + |S|}{p \cdot P_{\text{scan}}}$$

Sort-Merge Join – Performance Model

Total execution time

$$
\begin{aligned}
T_{\mathrm{sm}} &= T_{\mathrm{part}} + T_{\mathrm{sort}} + T_{\mathrm{merge}} + T_{\mathrm{match}} \\
&= \frac{|R| + |S|}{p \cdot P_{\mathrm{part}}} \\
&+ \frac{|R| + |S|}{p \cdot P_{\mathrm{sort}}} \\
&+ d_R \cdot \frac{|R|}{p \cdot P_{merge}} + d_S \cdot \frac{|S|}{p \cdot P_{merge}} \\
&+ \frac{|R| + |S|}{p \cdot P_{\mathrm{scan}}}
\end{aligned}
$$

Radix-Hash Join vs. Sort-Merge Join

Radix join

$$
\begin{aligned}
T_{\text{rdx}} &= T_{\text{hist}} + T_{\text{part}} + T_{\text{build}} + T_{\text{probe}} \\
&= \frac{|R| + |S|}{p \cdot P_{\text{scan}}} \\
&+ \left(\frac{1}{p \cdot P_{\text{net}}} + \frac{d-1}{p \cdot P_{\text{part}}} \right) \cdot (|R| + |S|) \\
&+ \frac{|R|}{p \cdot P_{\text{build}}} \\
&+ \frac{|S|}{p \cdot P_{\text{probe}}}
\end{aligned}
$$

Sort-merge join

$$
\begin{aligned}
T_{\text{sm}} &= T_{\text{part}} + T_{\text{sort}} + T_{\text{merge}} + T_{\text{match}} \\
&= \frac{|R| + |S|}{p \cdot P_{\text{part}}} \\
&+ \frac{|R| + |S|}{p \cdot P_{\text{sort}}} \\
&+ d_R \cdot \frac{|R|}{p \cdot P_{merge}} + d_S \cdot \frac{|S|}{p \cdot P_{merge}} \\
&+ \frac{|R| + |S|}{p \cdot P_{\text{scan}}}
\end{aligned}
$$

Radix-Hash Join vs. Sort-Merge Join

Radix join

$$T_{\text{rdx}} = T_{\text{hist}} + T_{\text{part}} + T_{\text{build}} + T_{\text{probe}}$$

$$= \frac{|R| + |S|}{p \cdot P_{\text{scan}}}$$

$$+ \left(\frac{1}{p \cdot P_{\text{net}}} + \frac{d-1}{p \cdot P_{\text{part}}}\right) \cdot (|R| + |S|)$$

$$+ \frac{|R|}{p \cdot P_{\text{build}}}$$

$$+ \frac{|S|}{p \cdot P_{\text{probe}}}$$

Sort-merge join

$$T_{\text{sm}} = T_{\text{part}} + T_{\text{sort}} + T_{\text{merge}} + T_{\text{match}}$$

$$= \frac{|R| + |S|}{p \cdot P_{\text{part}}}$$

$$+ \frac{|R| + |S|}{p \cdot P_{\text{sort}}}$$

$$+ d_R \cdot \frac{|R|}{p \cdot P_{merge}} + d_S \cdot \frac{|S|}{p \cdot P_{merge}}$$

$$+ \frac{|R| + |S|}{p \cdot P_{\text{scan}}}$$

Radix-Hash Join vs. Sort-Merge Join

Radix join

$$T_{\text{rdx}} = T_{\text{hist}} + T_{\text{part}} + T_{\text{build}} + T_{\text{probe}}$$

$$= \frac{|R| + |S|}{p \cdot P_{\text{scan}}}$$

$$+ \left(\frac{1}{p \cdot P_{\text{net}}} + \frac{d-1}{p \cdot P_{\text{part}}} \right) \cdot (|R| + |S|)$$

$$+ \frac{|R|}{p \cdot P_{\text{build}}}$$

$$+ \frac{|S|}{p \cdot P_{\text{probe}}}$$

Sort-merge join

$$T_{\text{sm}} = T_{\text{part}} + T_{\text{sort}} + T_{\text{merge}} + T_{\text{match}}$$

$$= \frac{|R| + |S|}{p \cdot P_{\text{part}}}$$

$$+ \frac{|R| + |S|}{p \cdot P_{\text{sort}}}$$

$$+ d_R \cdot \frac{|R|}{p \cdot P_{merge}} + d_S \cdot \frac{|S|}{p \cdot P_{merge}}$$

$$+ \frac{|R| + |S|}{p \cdot P_{\text{scan}}}$$

Radix-Hash Join vs. Sort-Merge Join

Radix join

$$T_{\text{rdx}} = T_{\text{hist}} + T_{\text{part}} + T_{\text{build}} + T_{\text{probe}}$$

$$= \frac{|R| + |S|}{p \cdot P_{\text{scan}}}$$

$$+ \left(\frac{1}{p \cdot P_{\text{net}}} + \frac{d-1}{p \cdot P_{\text{part}}} \right) \cdot (|R| + |S|)$$

$$+ \frac{|R|}{p \cdot P_{\text{build}}}$$

$$+ \frac{|S|}{p \cdot P_{\text{probe}}}$$

Sort-merge join

$$T_{\text{sm}} = T_{\text{part}} + T_{\text{sort}} + T_{\text{merge}} + T_{\text{match}}$$

$$= \frac{|R| + |S|}{p \cdot P_{\text{part}}}$$

$$+ \frac{|R| + |S|}{p \cdot P_{\text{sort}}}$$

$$+ d_R \cdot \frac{|R|}{p \cdot P_{merge}} + d_S \cdot \frac{|S|}{p \cdot P_{merge}}$$

$$+ \frac{|R| + |S|}{p \cdot P_{\text{scan}}}$$

Radix-Hash Join vs. Sort-Merge Join

Radix join

$$T_{\text{rdx}} = T_{\text{hist}} + T_{\text{part}} + T_{\text{build}} + T_{\text{probe}}$$

$$= \frac{|R| + |S|}{p \cdot P_{\text{scan}}}$$

$$+ \left(\frac{1}{p \cdot P_{\text{net}}} + \frac{d-1}{p \cdot P_{\text{part}}}\right) \cdot (|R| + |S|)$$

$$+ \frac{|R|}{p \cdot P_{\text{build}}}$$

$$+ \frac{|S|}{p \cdot P_{\text{probe}}}$$

Sort-merge join

$$T_{\text{sm}} = T_{\text{part}} + T_{\text{sort}} + T_{\text{merge}} + T_{\text{match}}$$

$$= \frac{|R| + |S|}{p \cdot P_{\text{part}}}$$

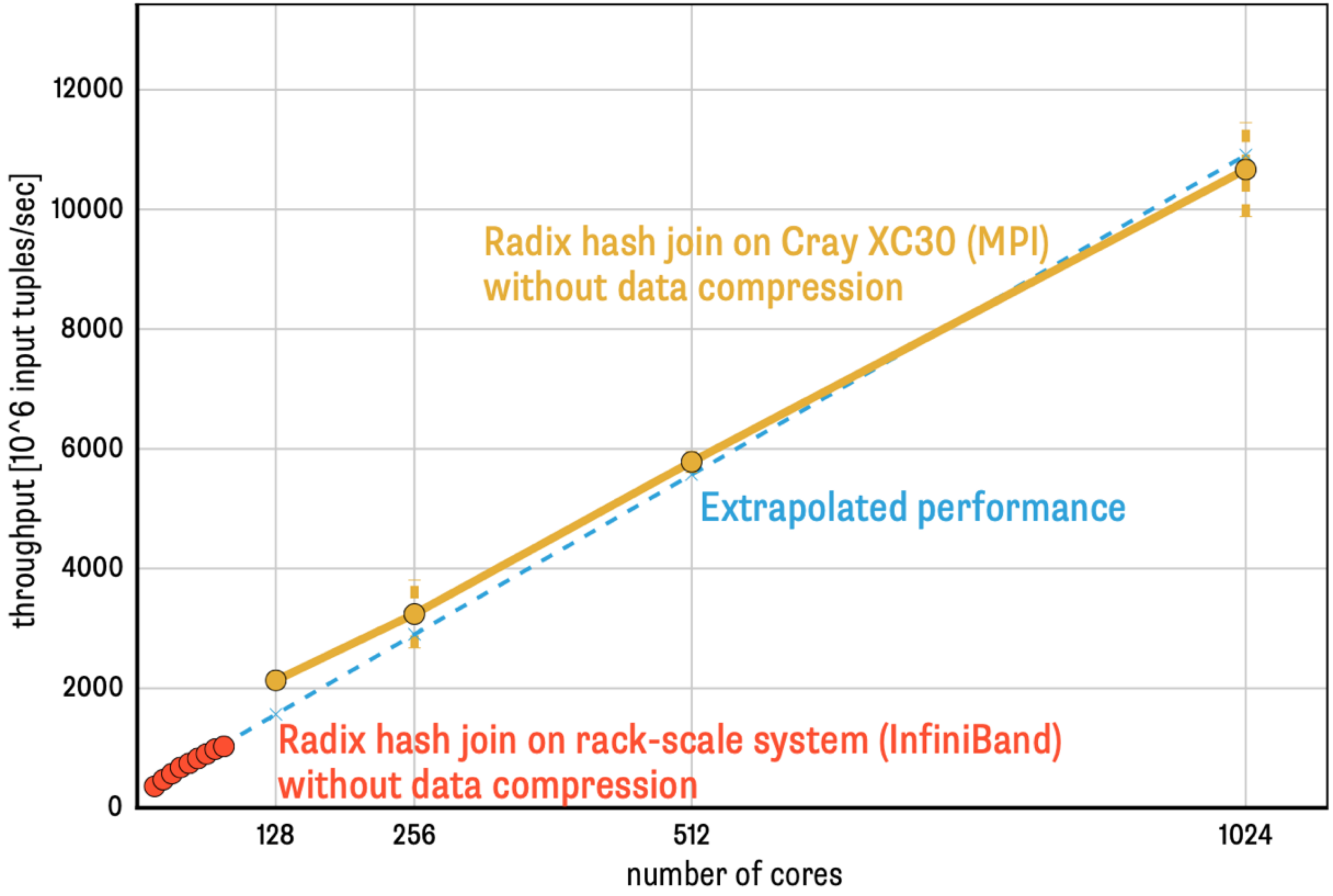
$$+ \frac{|R| + |S|}{p \cdot P_{\text{sort}}}$$

$$+ d_R \cdot \frac{|R|}{p \cdot P_{merge}} + d_S \cdot \frac{|S|}{p \cdot P_{merge}}$$

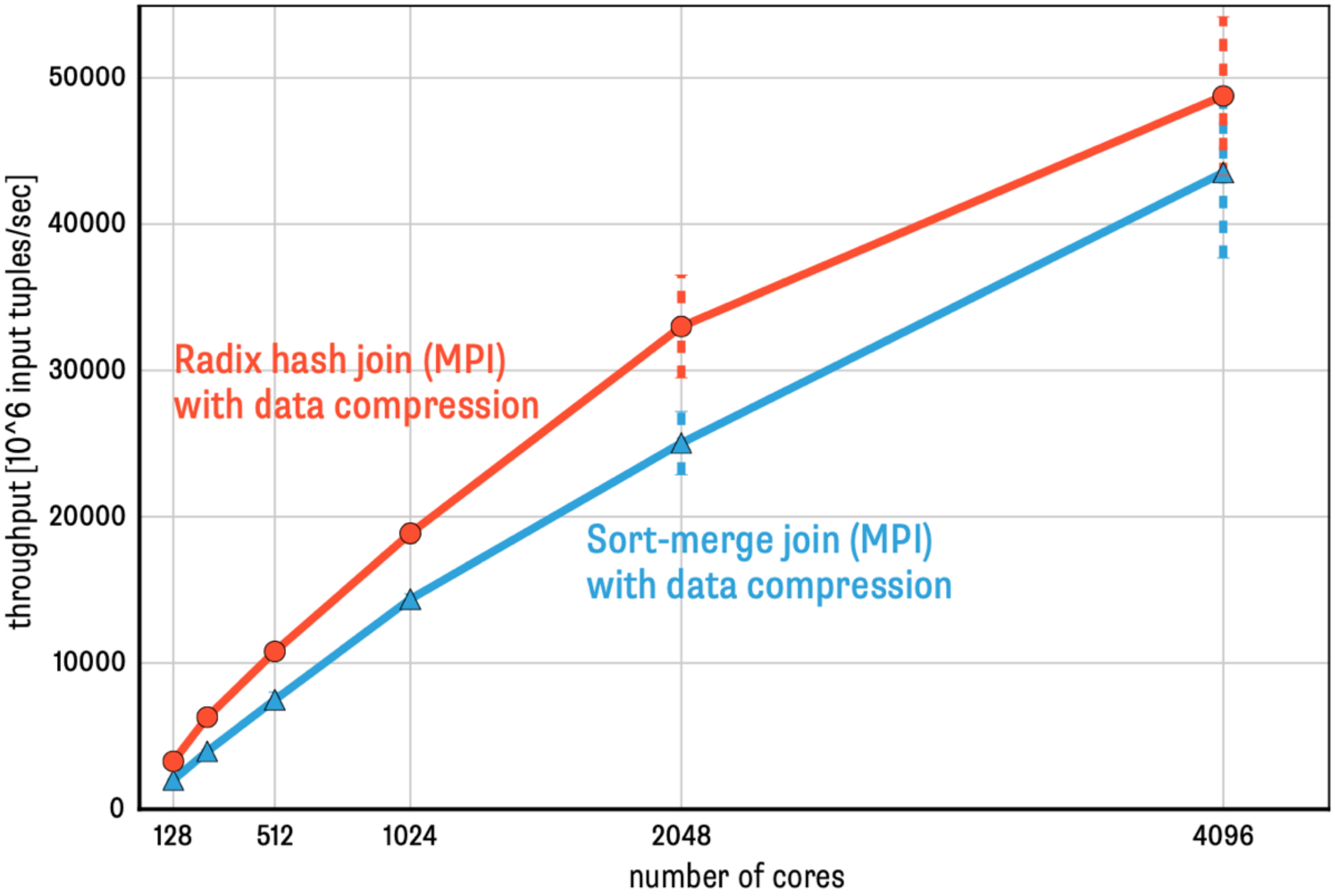
$$+ \frac{|R| + |S|}{p \cdot P_{\text{scan}}}$$

Performance Evaluation

Baseline Experiments

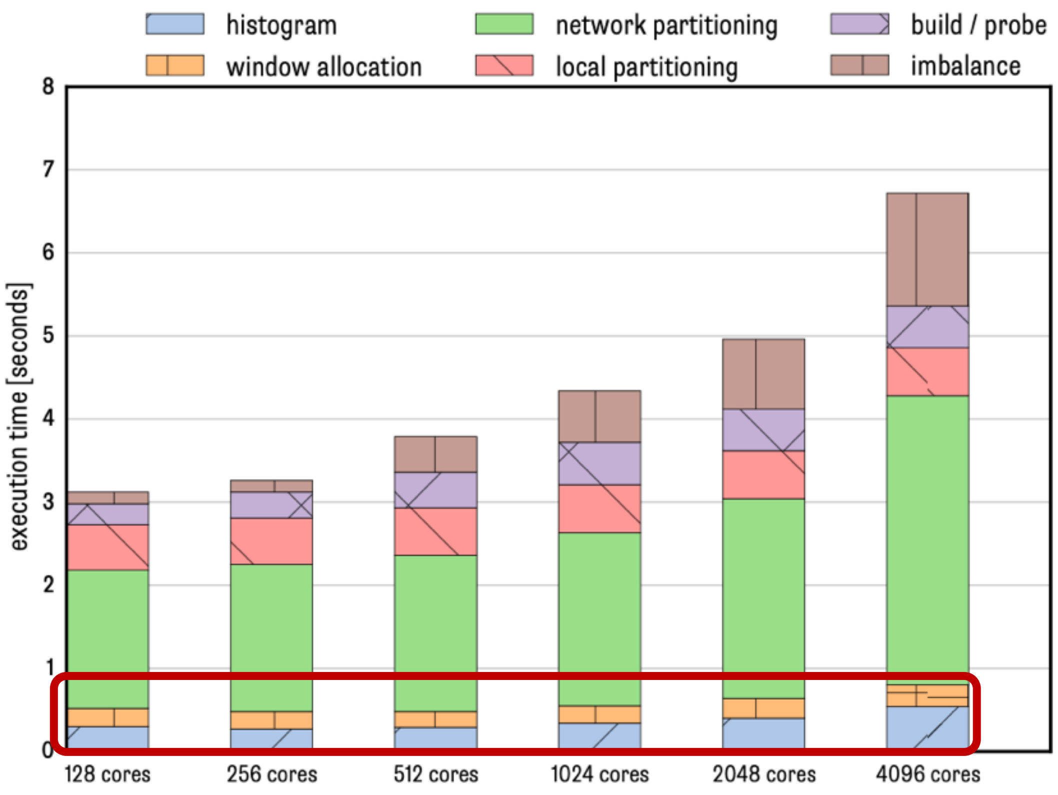


Scale-Out Experiments

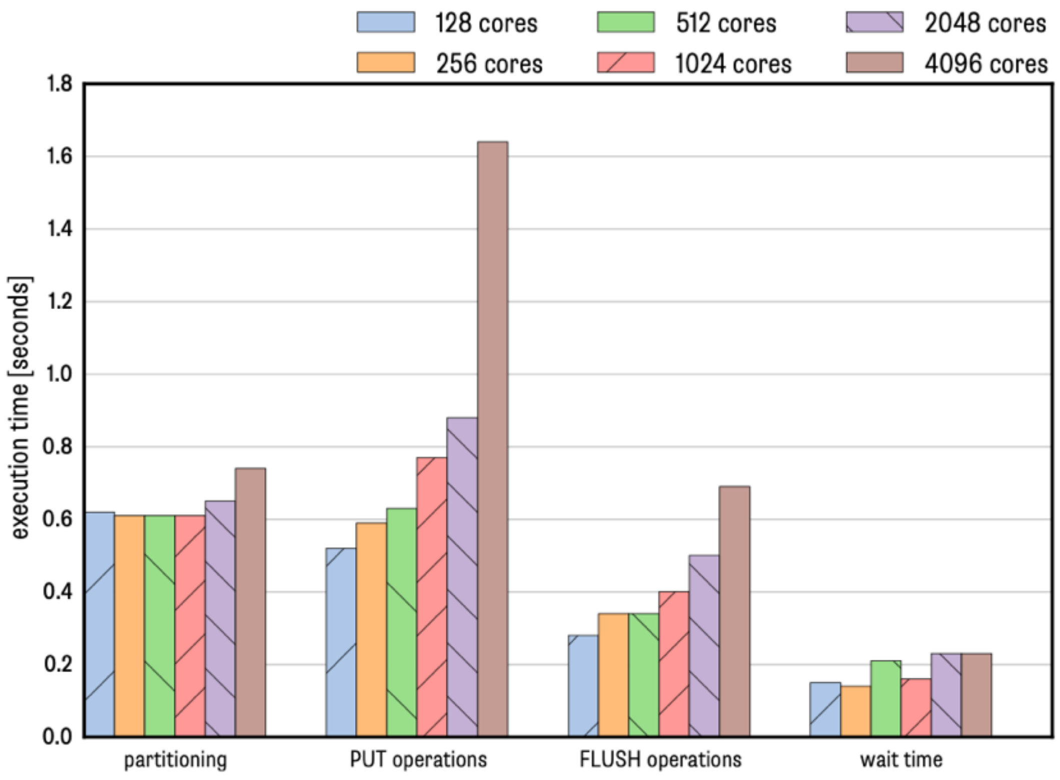


- Compression improves performance
- Radix join outperforms sort-merge join

Radix Join Execution Time Breakdown



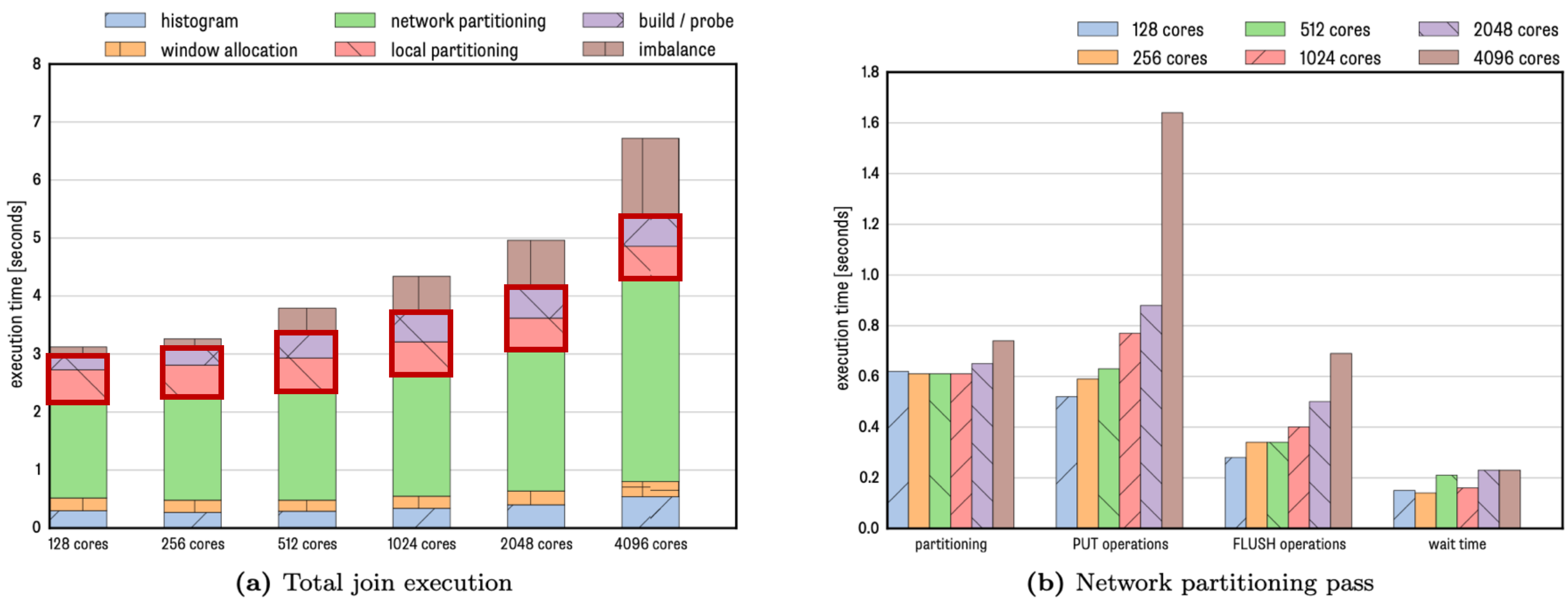
(a) Total join execution



(b) Network partitioning pass

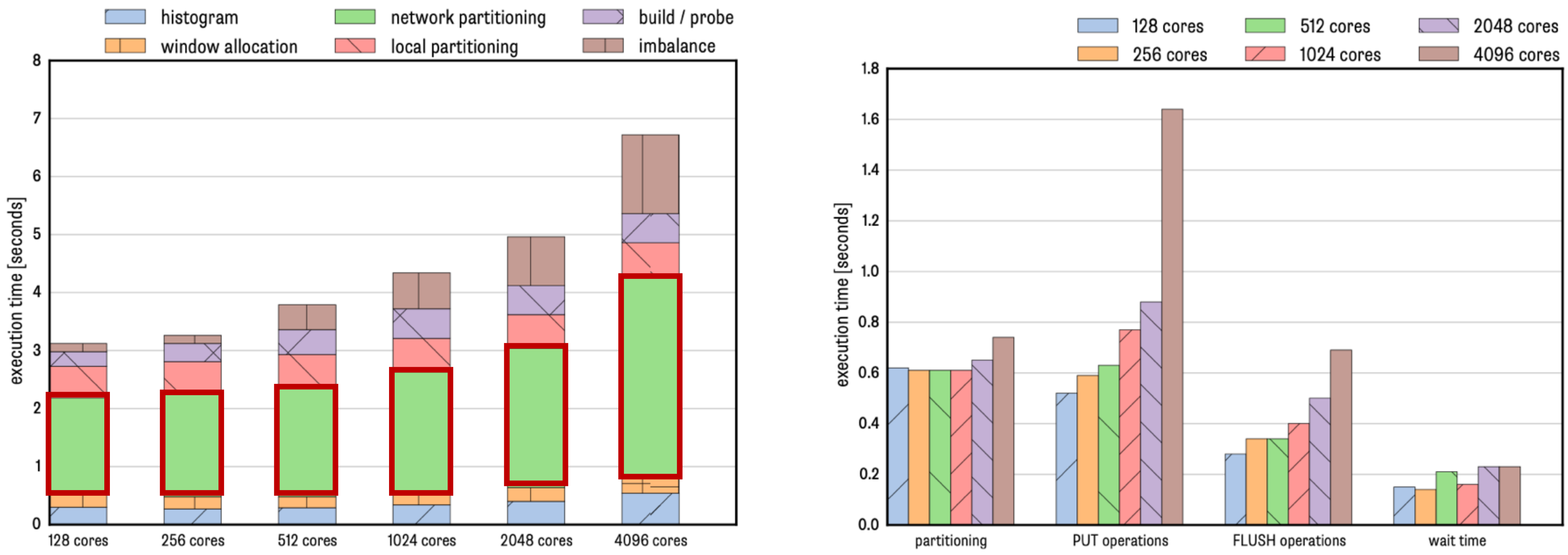
Time of Histogram computation and window allocation largely remains constant

Radix Join Execution Time Breakdown



Time of local partitioning and build/probe remain constant

Radix Join Execution Time Breakdown

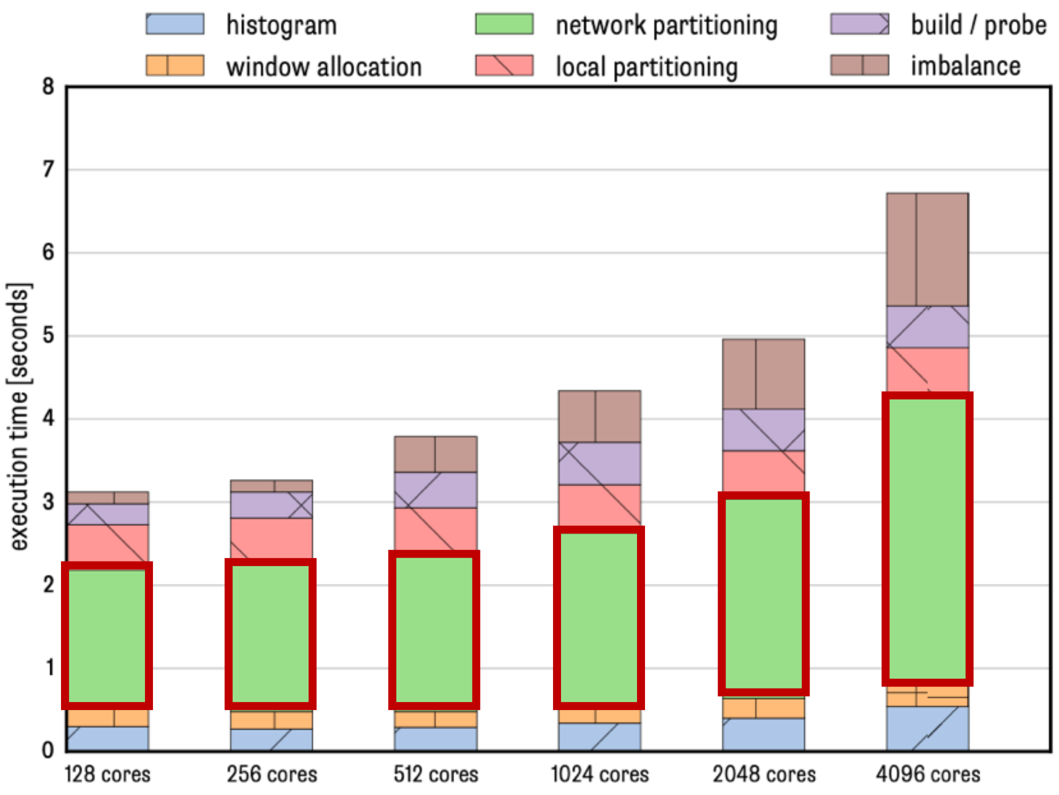


(a) Total join execution

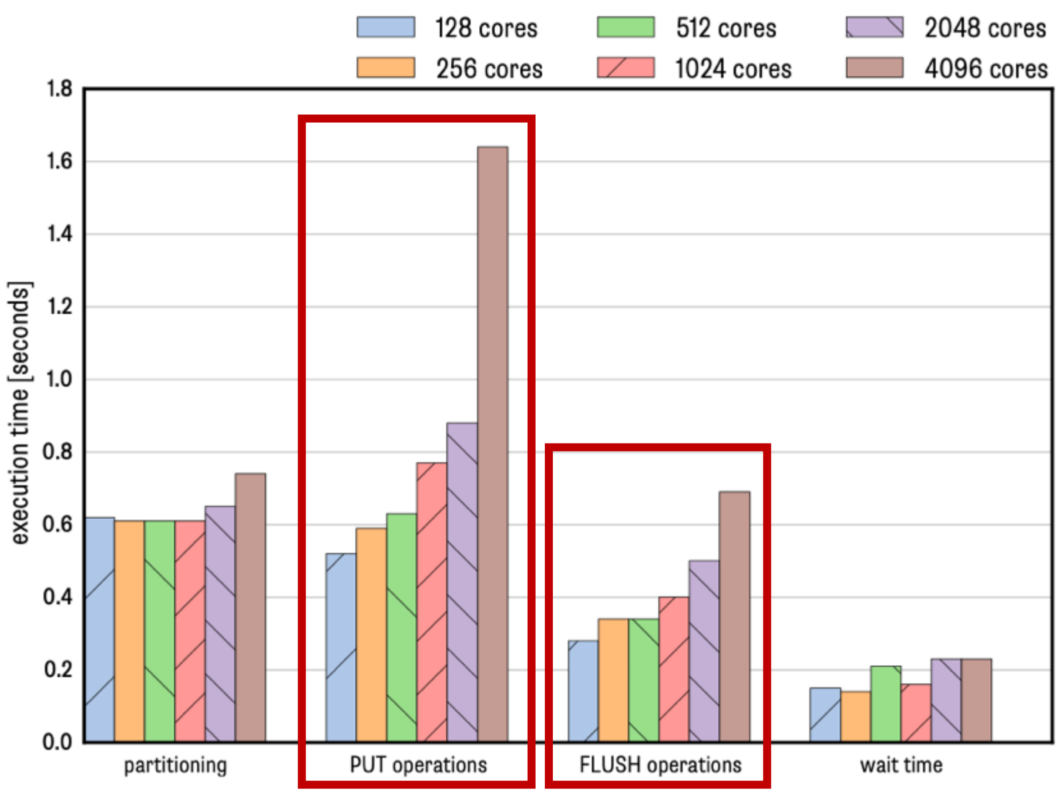
(b) Network partitioning pass

Time of network partitioning increases at more than 1024 cores

Radix Join Execution Time Breakdown



(a) Total join execution

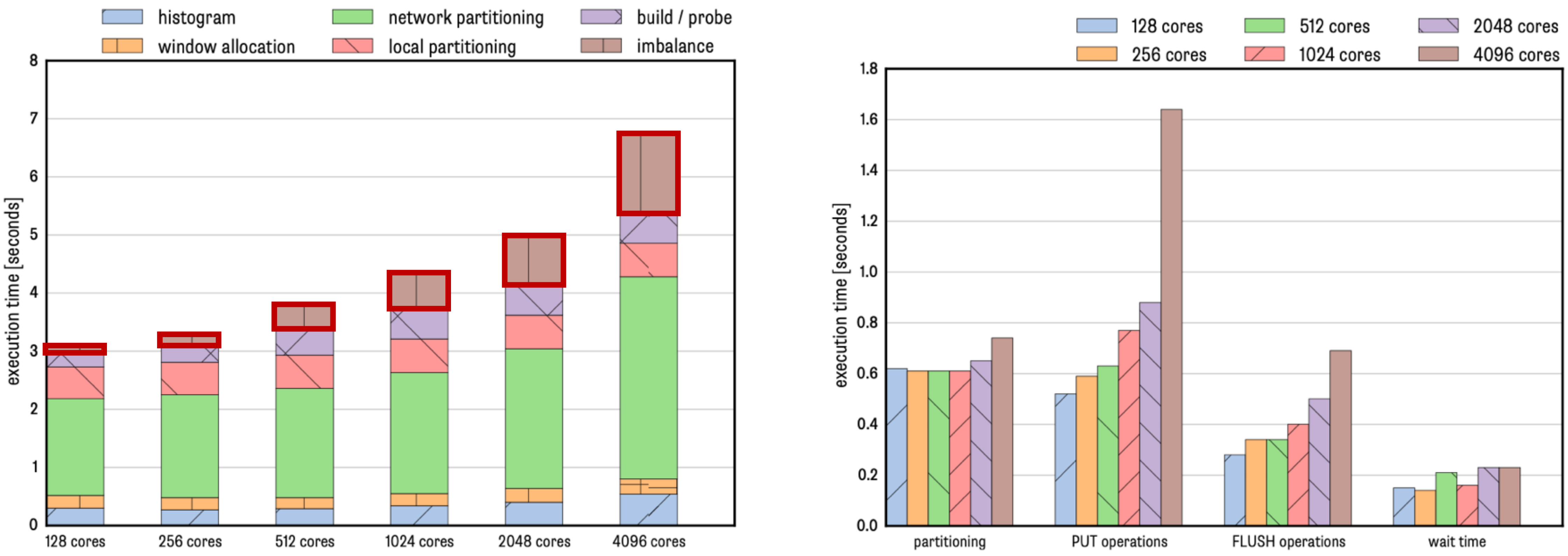


(b) Network partitioning pass

Time of network partitioning increases at more than 1024 cores

- Partitioning fan-out is increased beyond its optimal setting
- Additional time spent in **MPI_Put** and **MPI_Flush**

Radix Join Execution Time Breakdown

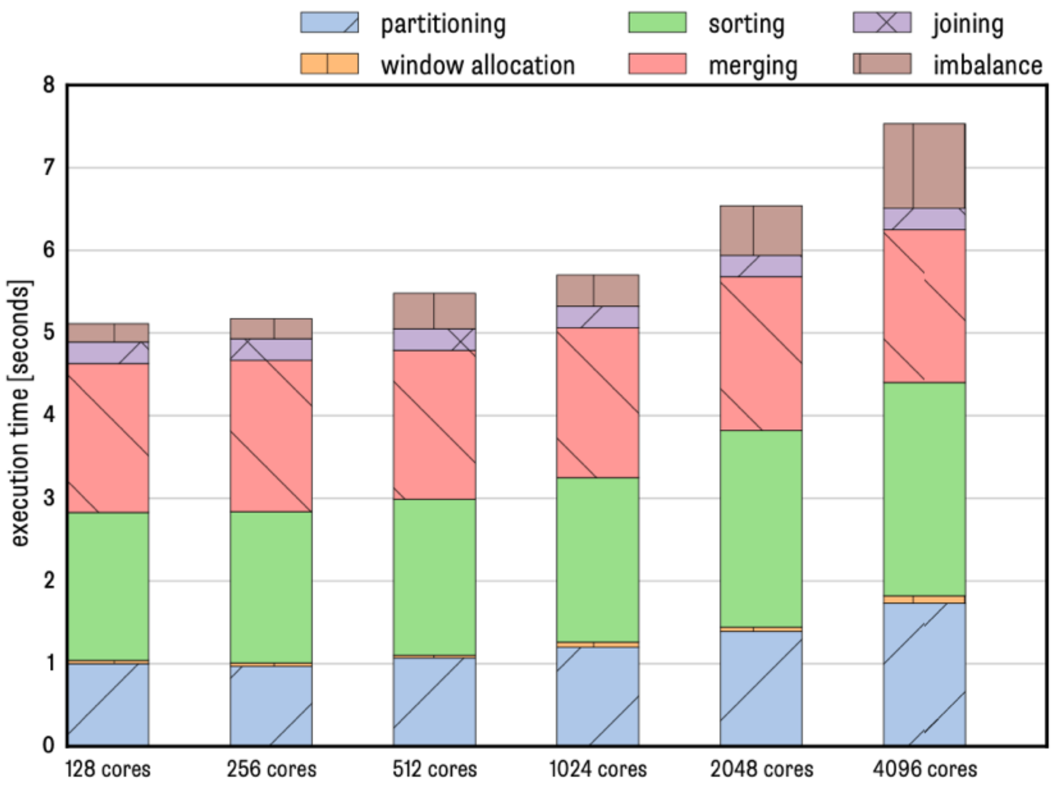


(a) Total join execution

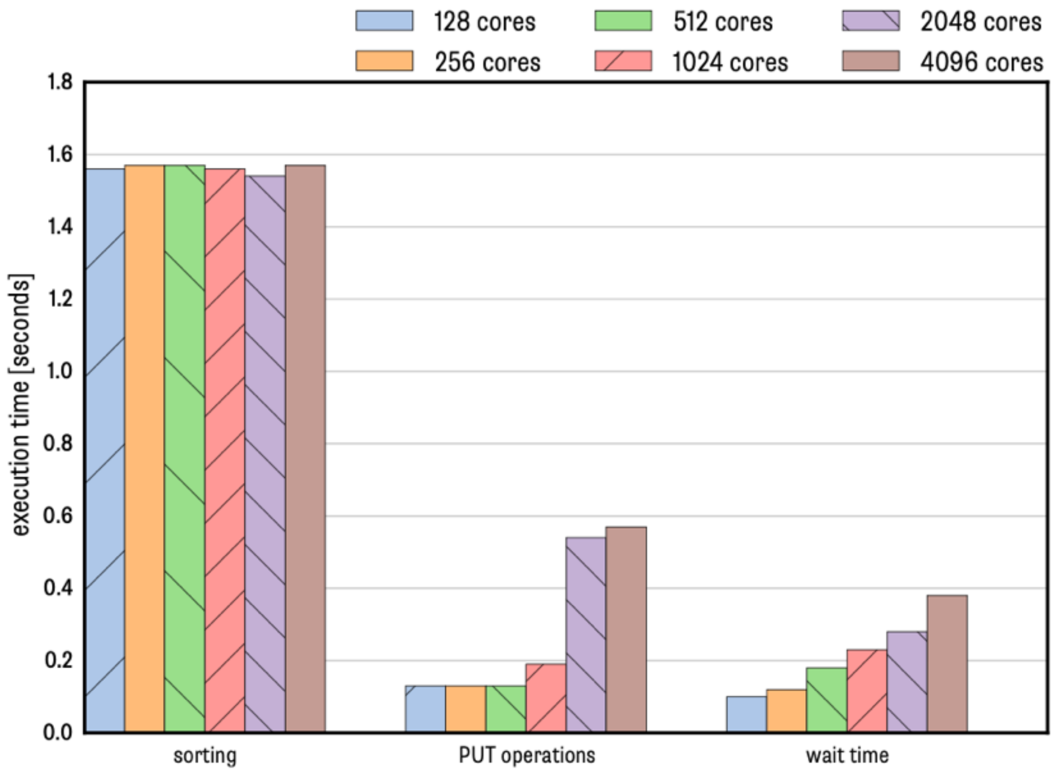
(b) Network partitioning pass

Time due to load imbalance increases with core count

Sort-Merge Join Execution Time Breakdown

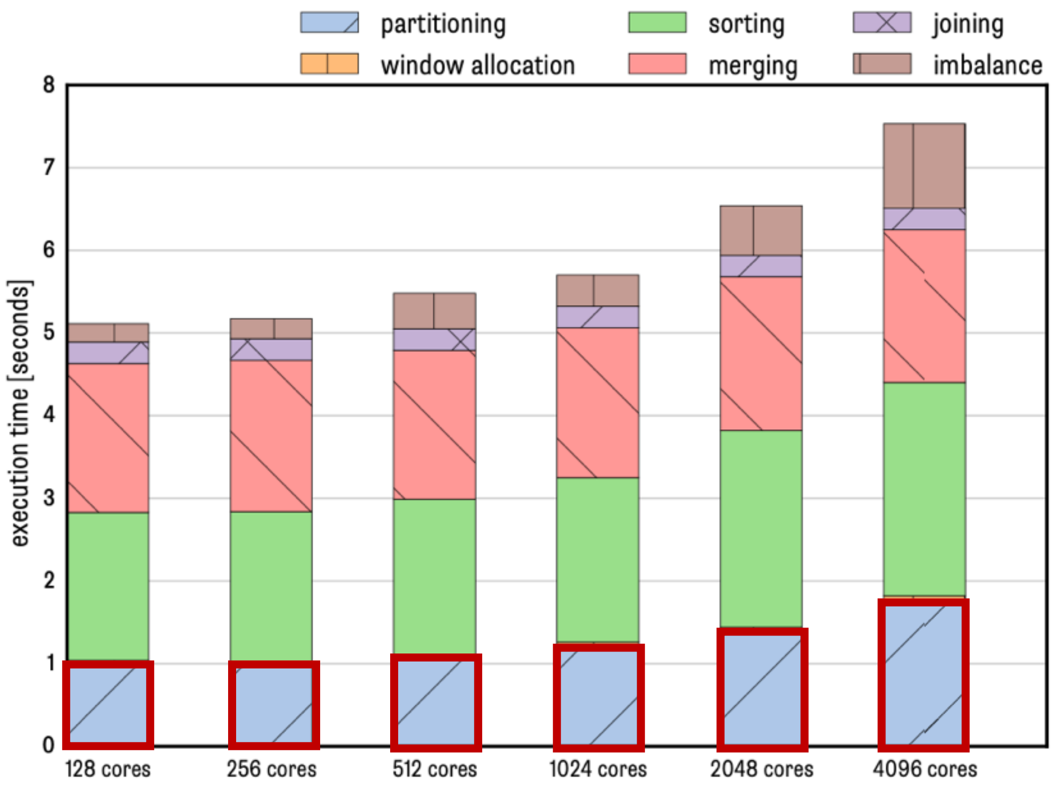


(a) Total join execution

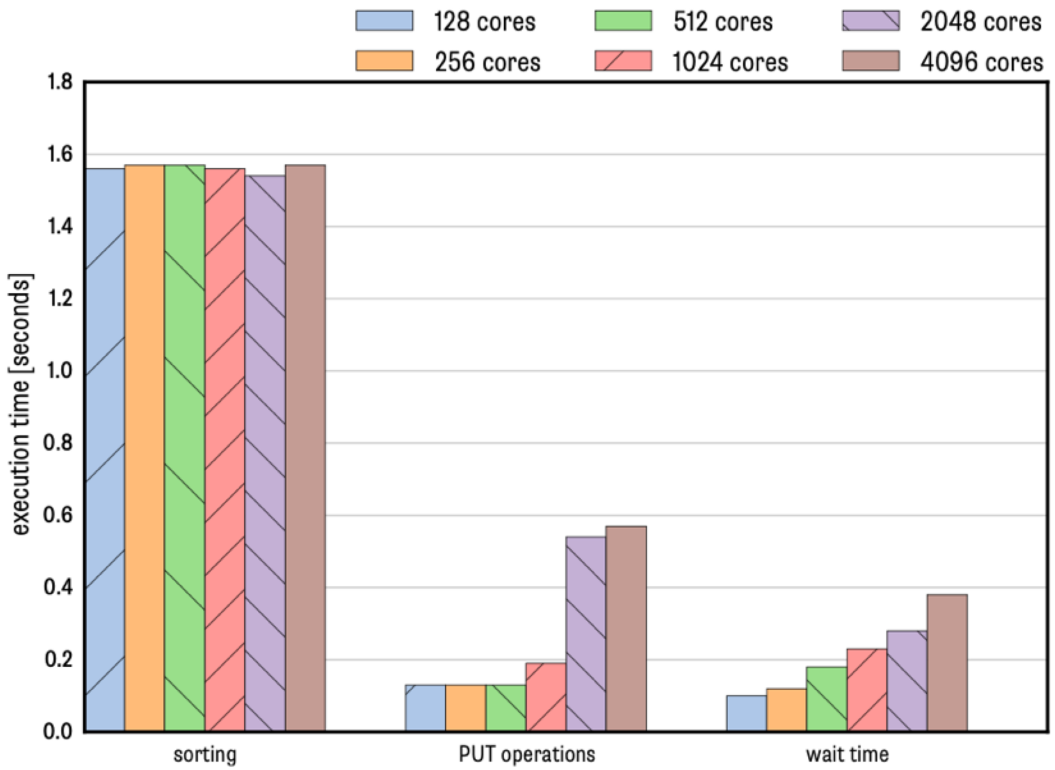


(b) Sorting phase

Sort-Merge Join Execution Time Breakdown



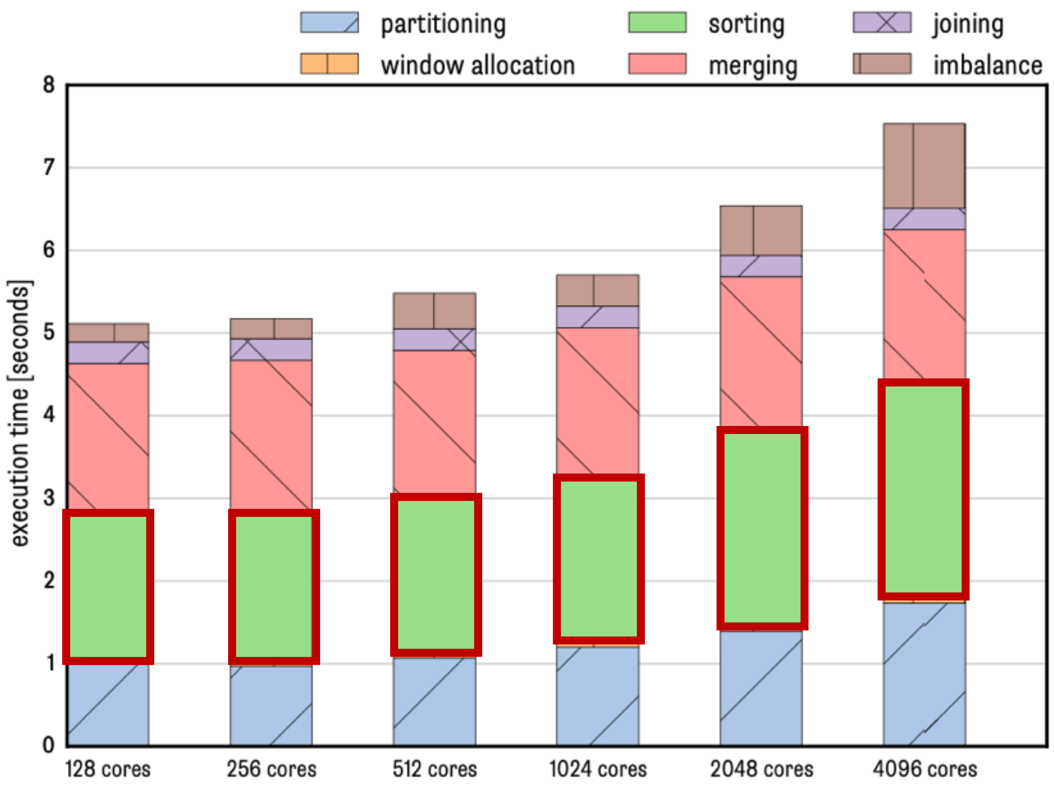
(a) Total join execution



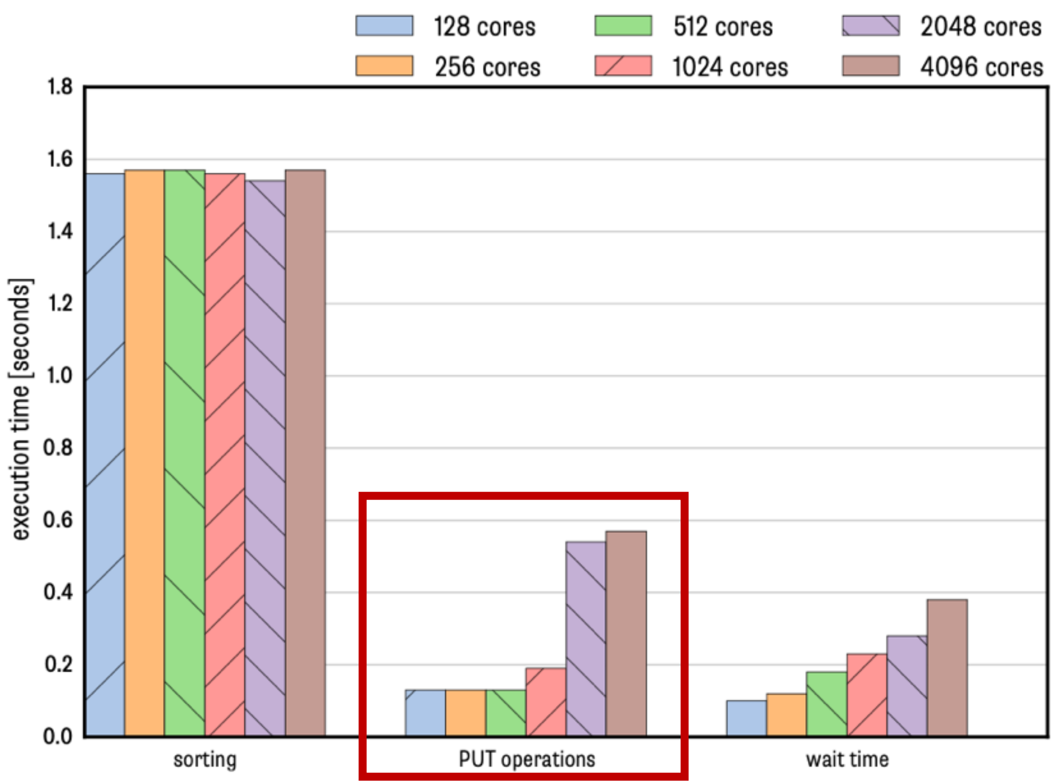
(b) Sorting phase

Partitioning fan-out is pushed beyond its optimal configuration

Sort-Merge Join Execution Time Breakdown



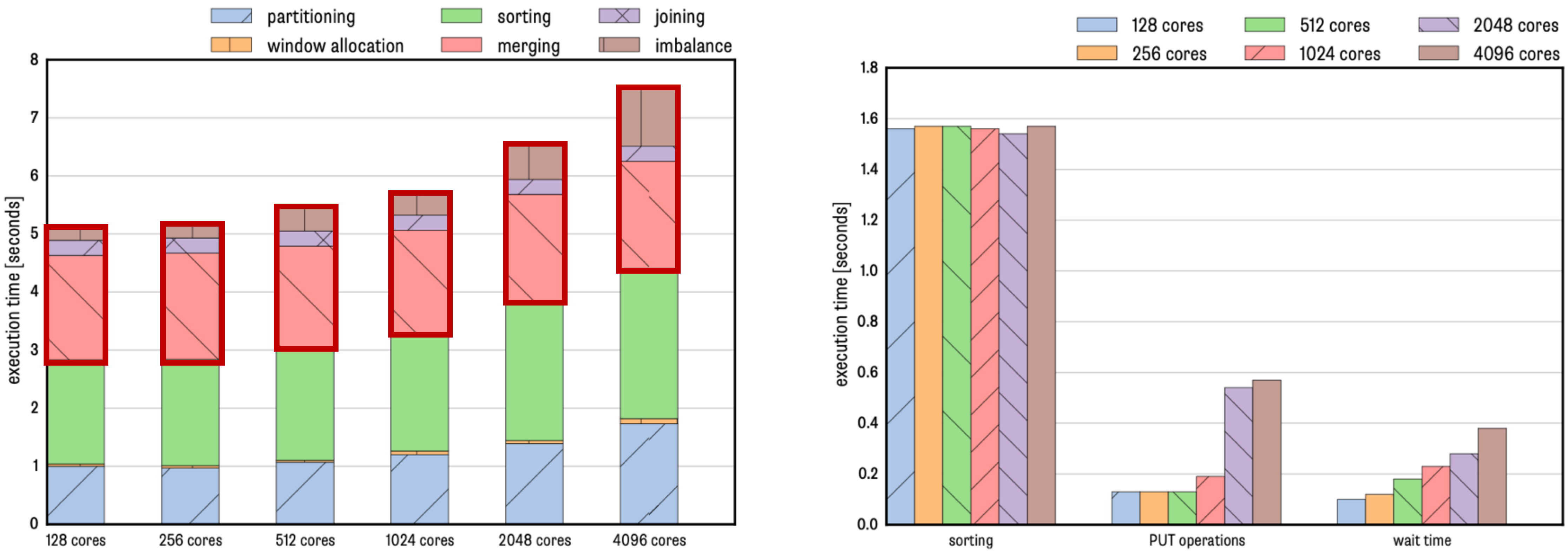
(a) Total join execution



(b) Sorting phase

Within sorting, time of network shuffling increases with core count

Sort-Merge Join Execution Time Breakdown

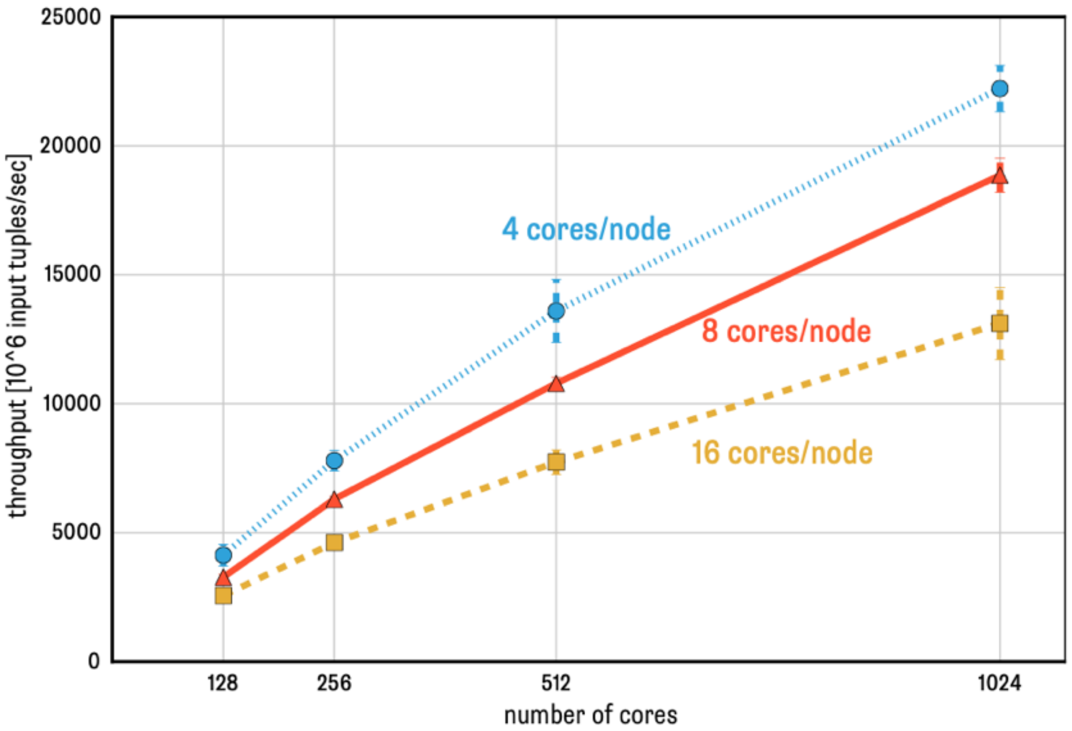


(a) Total join execution (b) Sorting phase

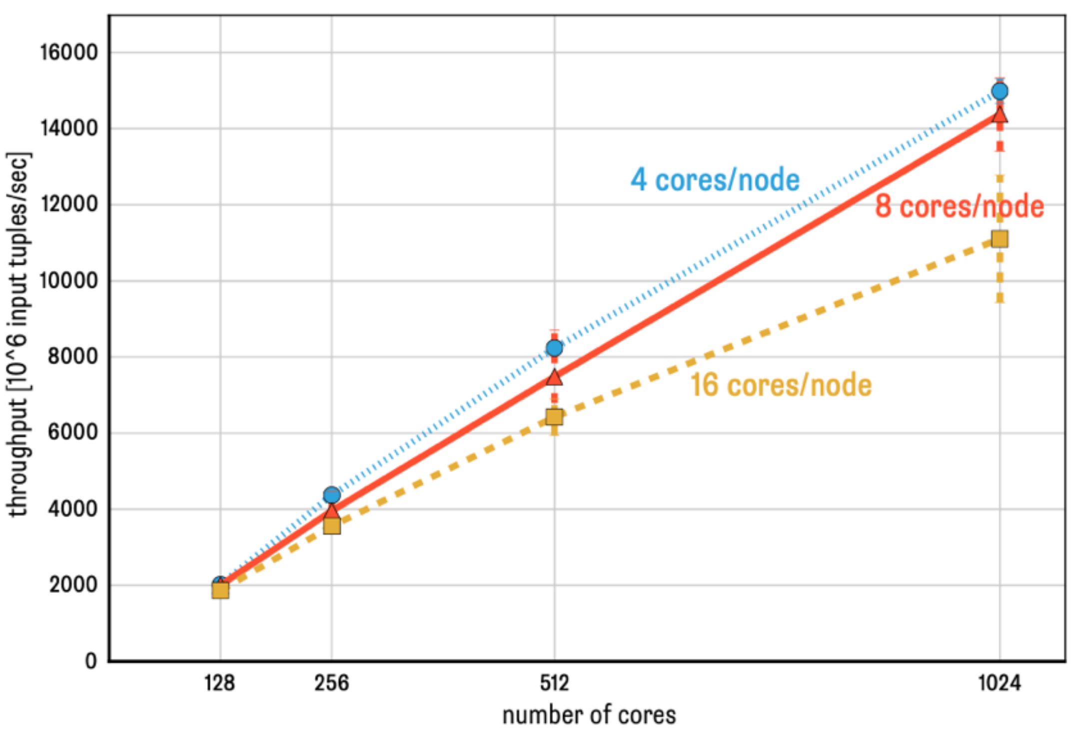
Time of merge and joining stays constant

Time due to load imbalance slightly increases with core count

Scale-Up Experiments



(a) Radix hash join



(b) Sort-merge join

With more cores per machine, considerably more time spent on **MPI_Put** and **MPI_Flush**. Difficult to fully interleave computation and communication

Comparison with the Model

Radix hash join

Phase	Exec. Time	Model	Diff.
Histogram Comp.	0.34s	0.36s	−0.02s
Window Allocation	0.21s	—	+0.21s
Network Partitioning	2.08s	0.67s	+1.41s
Local Partitioning	0.58s	0.67s	−0.09s
Build-Probe	0.51s	0.51s	+0.00s
Imbalance	0.62s	—	+0.62s
Total	4.34s	2.21s	+2.13s

Sort-merge join

Partitoning	1.20s	1.02s	+0.18s
Window Allocation	0.06s	—	+0.06s
Sorting	1.99s	1.45s	+0.54s
Merging	1.81s	1.78s	+0.03s
Matching	0.26s	0.36s	−0.10s
Imbalance	0.38s	—	+0.38s
Total	5.70s	4.61s	+1.09s

Parameters [million tuples per second]

RHJ: $P_{\text{scan}} = 225$, $P_{\text{Part}} = 120$, $P_{\text{net}} = 1024$, $P_{\text{build}} = 120$, $P_{\text{probe}} = 225$

SMJ: $P_{\text{part}} = 78$, $P_{\text{sort}} = 75$, $P_{\text{net}} = 1024$, $P_{\text{merge}} = 45$, $P_{\text{scan}} = 225$

Network shuffling is the bottleneck

RDMA for OLAP – Q/A

Collective communication scheduling for joins?

Supercomputers used in the real world for database workloads?

Radix join vs. hash join?

Radix join does not achieve theoretical maximum performance

What is partition fan-out?

MPI vs. shared memory for join

Group Discussion

How can Smart NICs help improve the performance of joins?

Can you think of any hardware/software techniques that may close the performance gap between radix join and sort-merge join?

Can you think of any hardware/software techniques that may allow radix join to achieve its theoretical maximum performance?

Before Next Lecture

Submit discussion summary to https://wisc-cs839-ngdb20.hotcrp.com

- **Deadline: Wednesday 11:59pm**

Submit review for

- Amazon Aurora: Design Considerations for High Throughput Cloud-Native Relational Databases
- [optional] Amazon Aurora: On Avoiding Distributed Consensus for I/Os, Commits, and Membership Changes