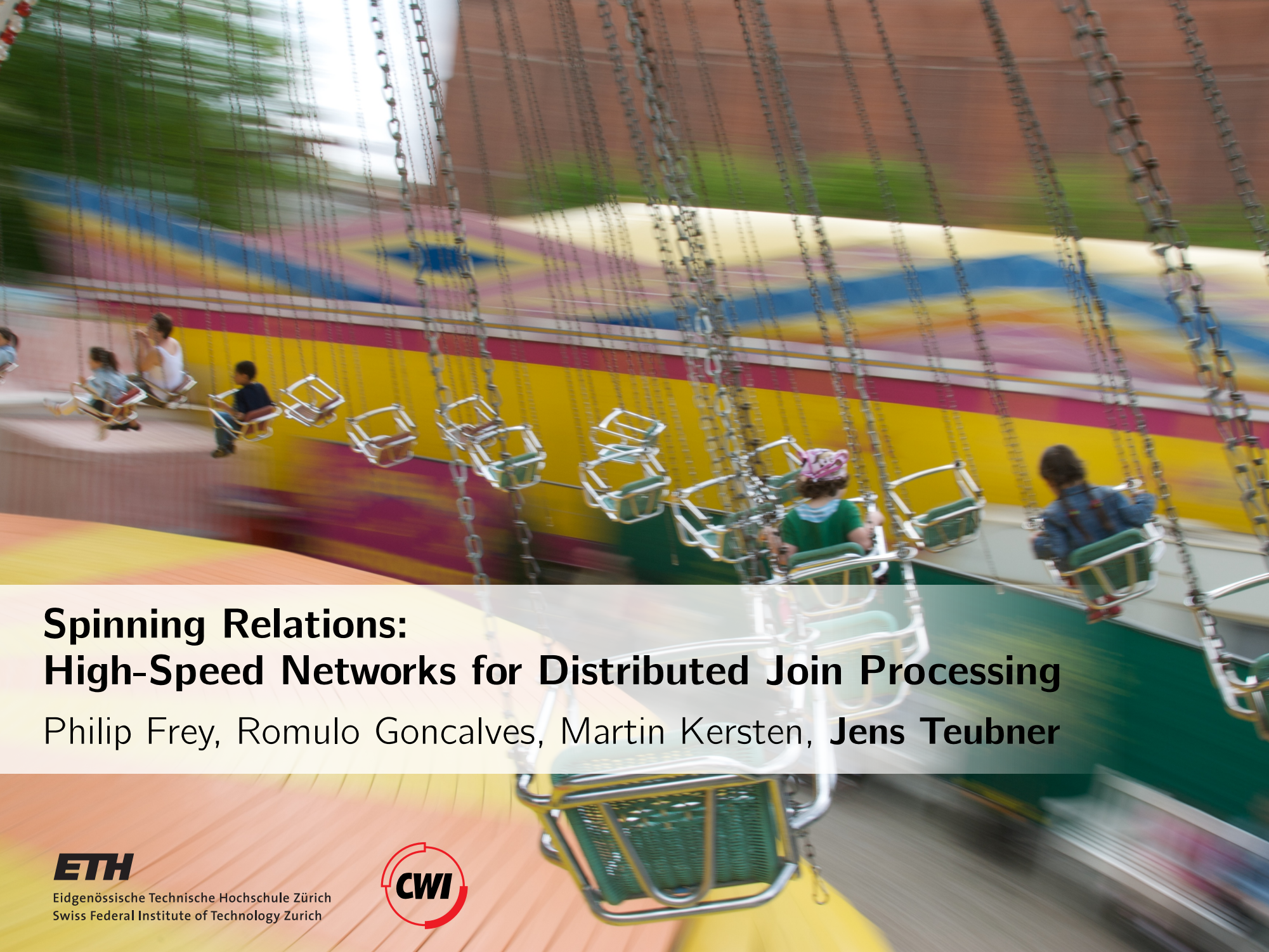


Spinning Relations: High-Speed Networks for Distributed Join Processing

Philip Frey, Romulo Goncalves, Martin Kersten, **Jens Teubner**

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

CWI

Problem Statement

We address a core database problem, but for large problem sizes:

- Process a **join** $R \bowtie_\theta S$ (arbitrary join predicate).
- R and S are **large** (many gigabytes, even terabytes).

Traditional approach:

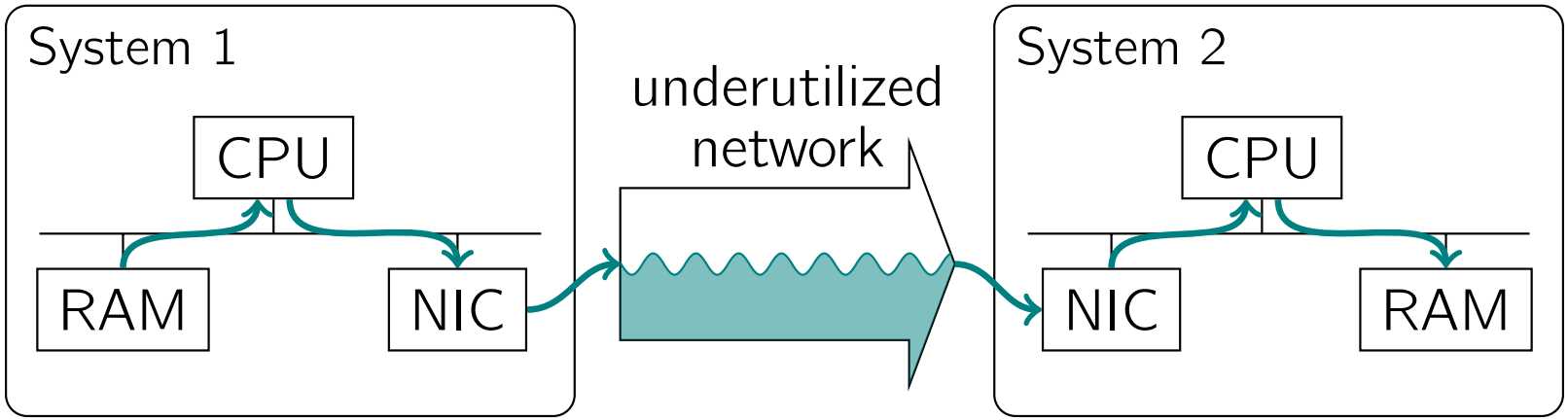
- Use a **big** machine and/or suffer the severe disk I/O bottleneck of block nested loops join.
- Can do **distributed evaluation** only for certain θ or certain data distributions (or suffer high network I/O cost).

Today:

- Assume a **cluster** of **commodity machines** only.
- Leverage modern **high-speed networks** (10 Gb/s and beyond).

Modern Networks: High Speed?

It is actually very hard to saturate modern (*e.g.*, 10 Gb/s) networks.

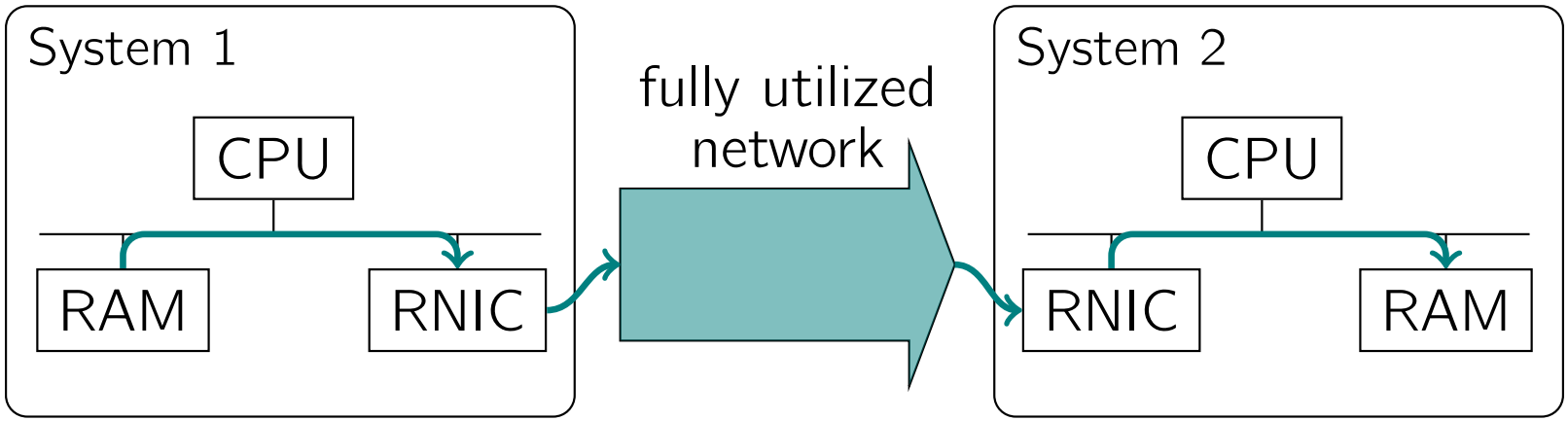


- **High CPU demand**
  - Rule of thumb: 1 GHz CPU per 1 Gb/s network throughput (!)
- **Memory bus contention**
  - Data typically has to cross the memory bus **three times**
    $\rightarrow \approx 3$ GB/s bus capacity needed for 10 Gb/s network

RDMA: Remote Direct Memory Access

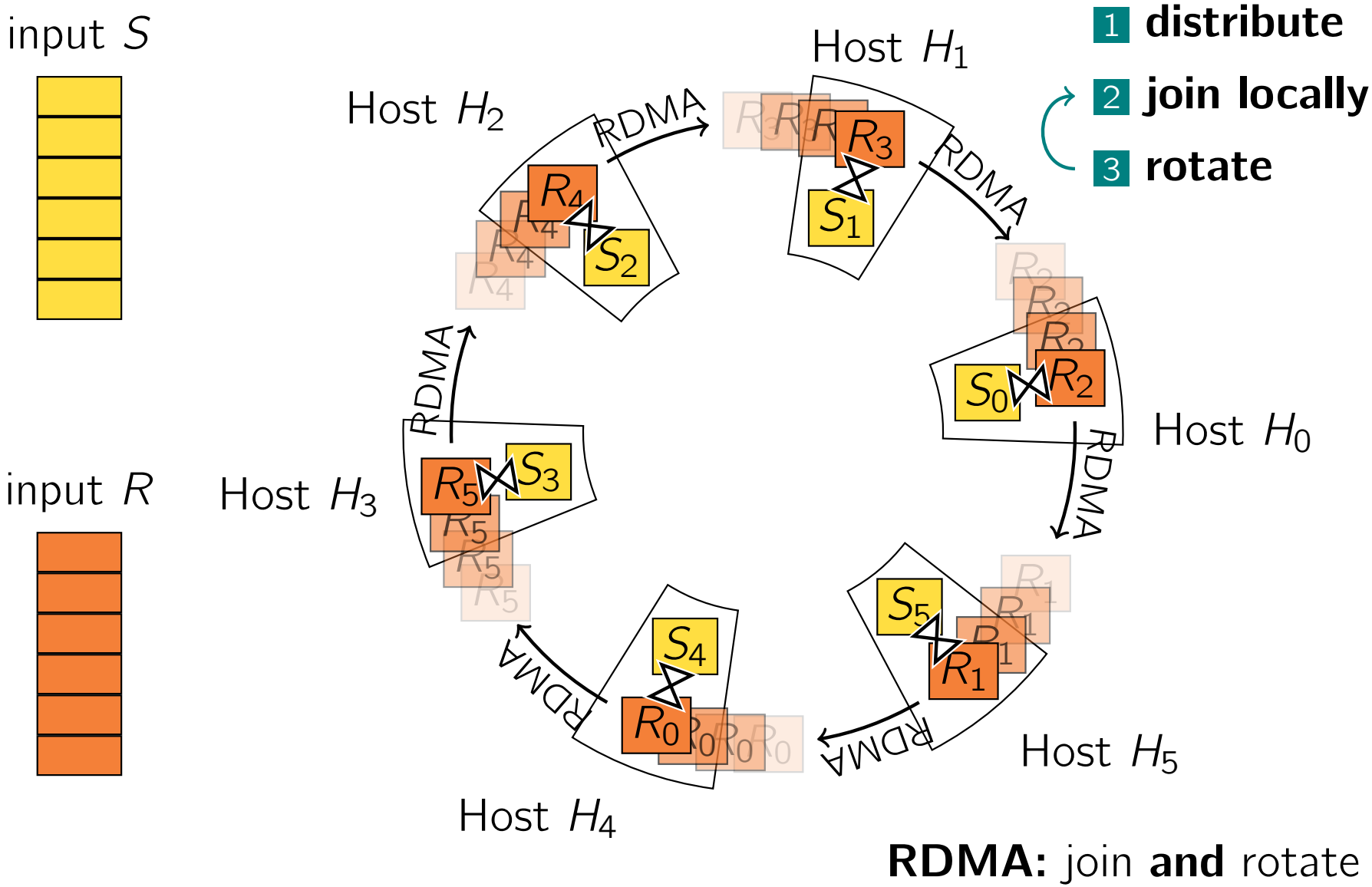
RDMA-capable network cards (RNICs) can saturate the link using

- **direct data placement** (avoid unnecessary bus transfers),
- **OS bypassing** (avoid context switches), and
- **TCP offloading** (avoid CPU load).



- Data is read/written on both ends using intra-host **DMA**.
- **Asynchronous** transfer after **work request** issued by CPU.

Cyclo-Join Idea



Analysis

Cyclo-join has similarities to **block nested loops join**.

- Cut input data into **blocks** R_i and S_j.
- Join all combinations $R_i \bowtie S_j$ **in memory**.

As such, *cyclo-join*

- can be paired with **any in-memory join algorithm**,
- can be used to distribute the processing of **any join predicate**.

Cyclo-join fits into a "cloud-style" environment:

- **additional nodes** can be hooked in as needed,
- arbitrary assignment host $\leftrightarrow$ task,
- *cyclo-join* consumes **and** produces distributed tables
  $\rightarrow$ *n*-way joins.

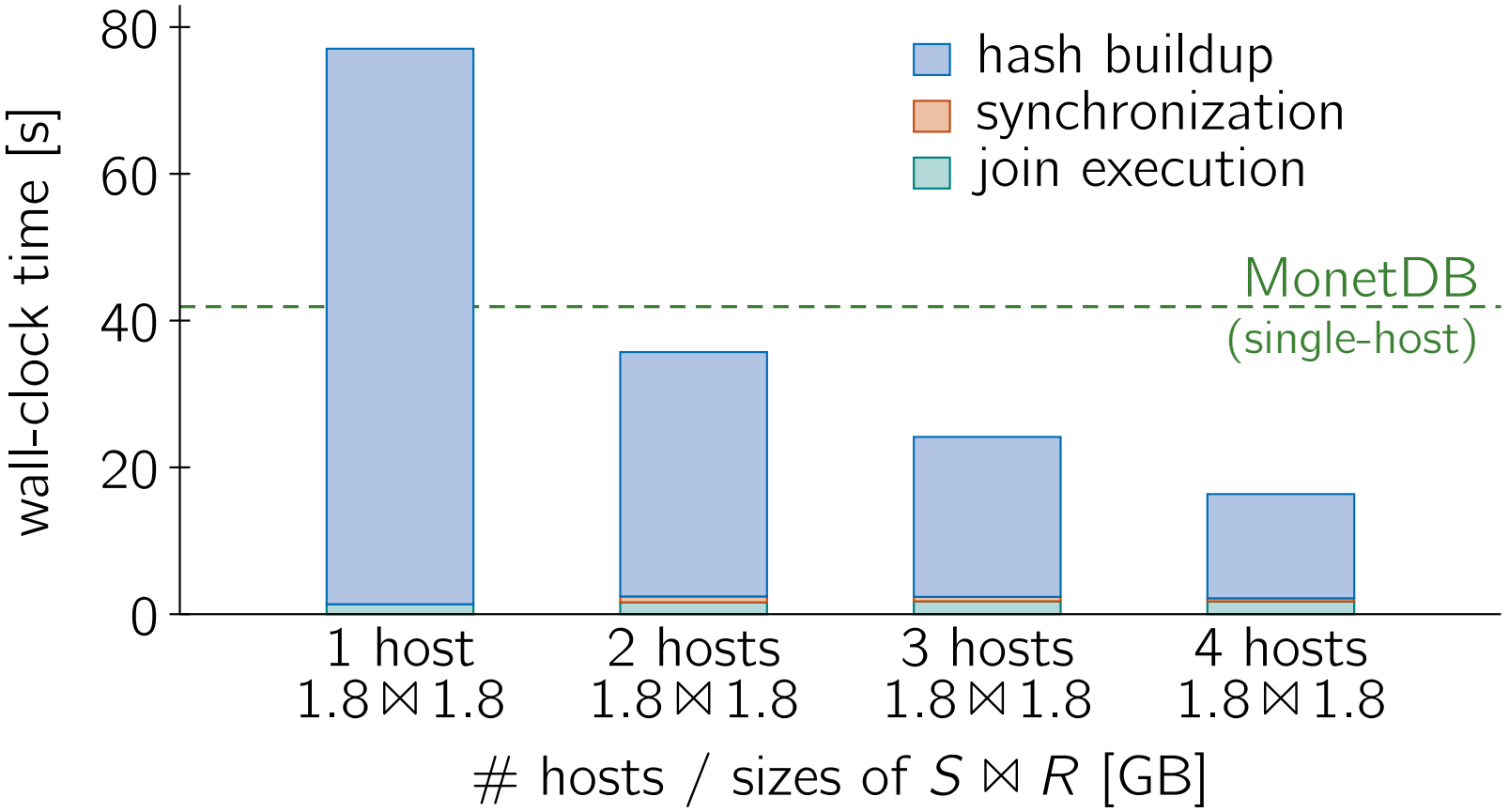
Cyclo-Join Put Into Practice

We implemented a prototype of *cyclo-join*:

- **four processing nodes**
  - Intel Xeon quad-core 2.33 GHz
  - 6 GB RAM per node; memory bandwidth: 3.4 GB/s (measured)
- **10 Gb/s Ethernet**
  - Chelsio T3 RDMA-enabled network cards
  - Nortel 10 Gb/s Ethernet switch
- in-memory **hash join**
  - hash phase **physically re-organizes data** (on each node)
    $\rightarrow$ better **cache efficiency** during join phase
  - I/O complexity: $\mathcal{O}(|R| + |S|)$

Experiments

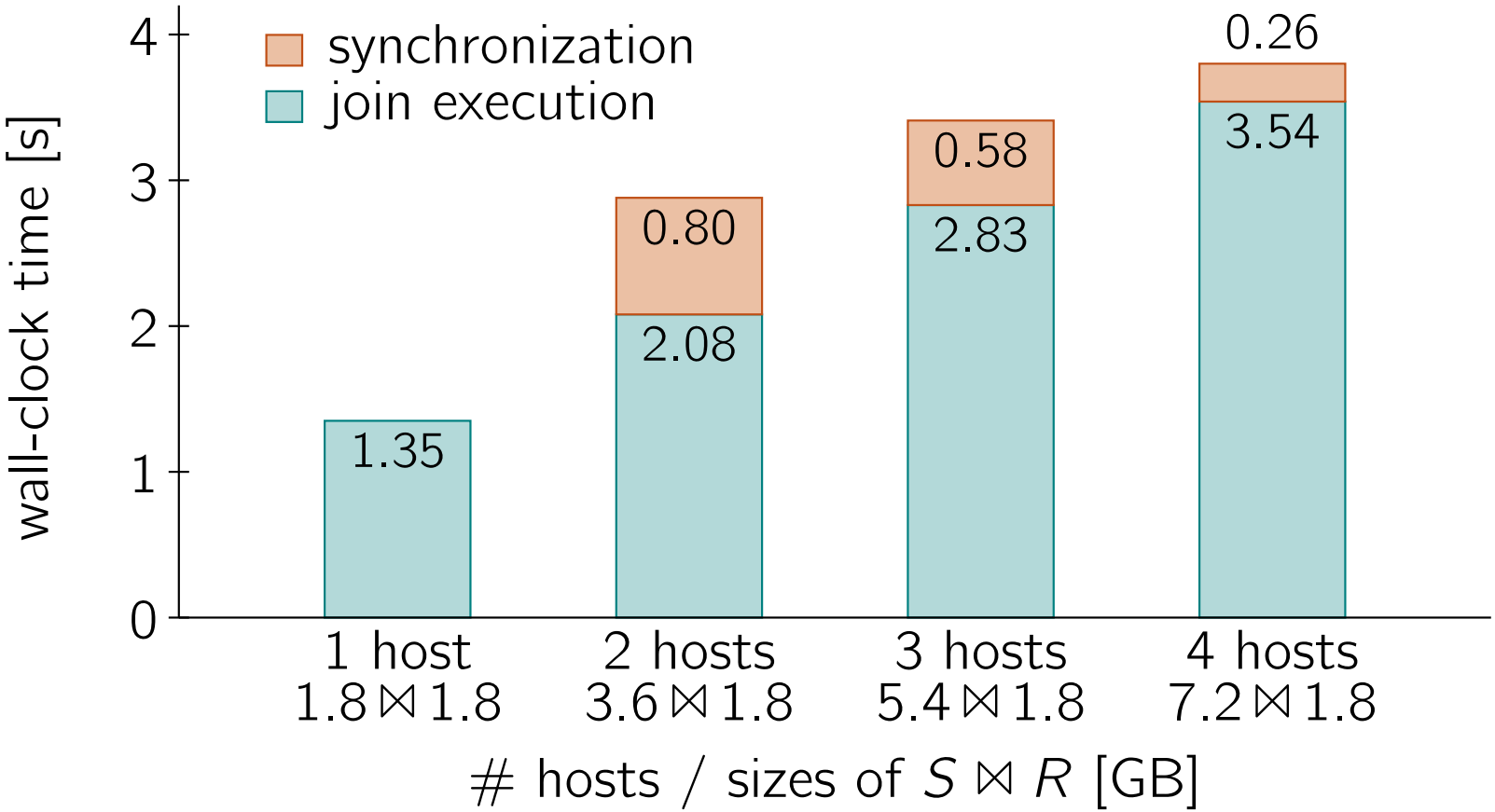
Experiment 1: Distribute evaluation of a join where $|R| = |S| = 1.8\,\text{GB}$.



- Main benefit: reduced **hash buildup time**.

Experiments

Experiment 2: Scale up and join larger S (hash buildup ignored here).

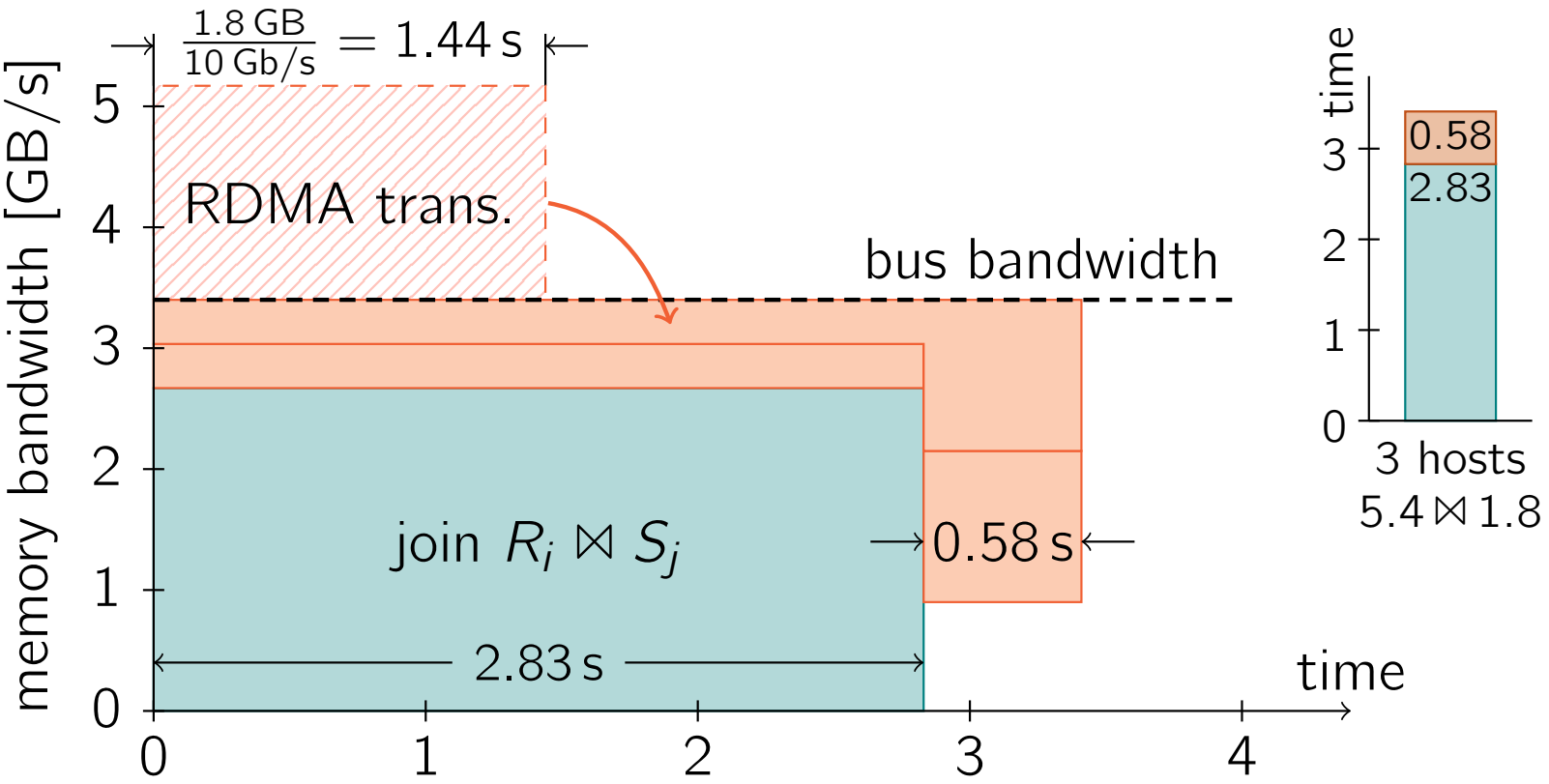


☺ System **scales** like a machine with large RAM would.

☹ CPUs have to **wait for network transfers** ("synchronization").

Memory Transfers

Need to wait for network: Does that mean RDMA doesn't work at all?



- The culprit is the **local memory bus**!
- If RDMA hadn't saved us some bus transfers, this would be **worse**.

Conclusions

I demonstrated *cyclo-join*:

- **ring topology** to process **large joins**,
- use **distributed memory** to process **arbitrary joins**,
- hardware acceleration via **RDMA** is crucial:
  - reduce CPU load **and memory bus contention**.

Cyclo-join is part of the *Data Cyclotron* project:

- support for **more local join algorithms**,
- process **full queries** in a **merry-go-round setup**.