Rethinking Stateful Stream Processing with RDMA

<u>Bonaventura Del Monte</u> · Steffen Zeuch · Tilmann Rabl · Volker Markl

ACM SIGMOD 2022

TU berlin

German Research Center for Artificial Intelligence

HPI Hasso Plattner Institut
Digital Engineering · Universität Potsdam

BIFOLD
The Berlin Institute for the Foundations of Learning and Data

What is this talk about?

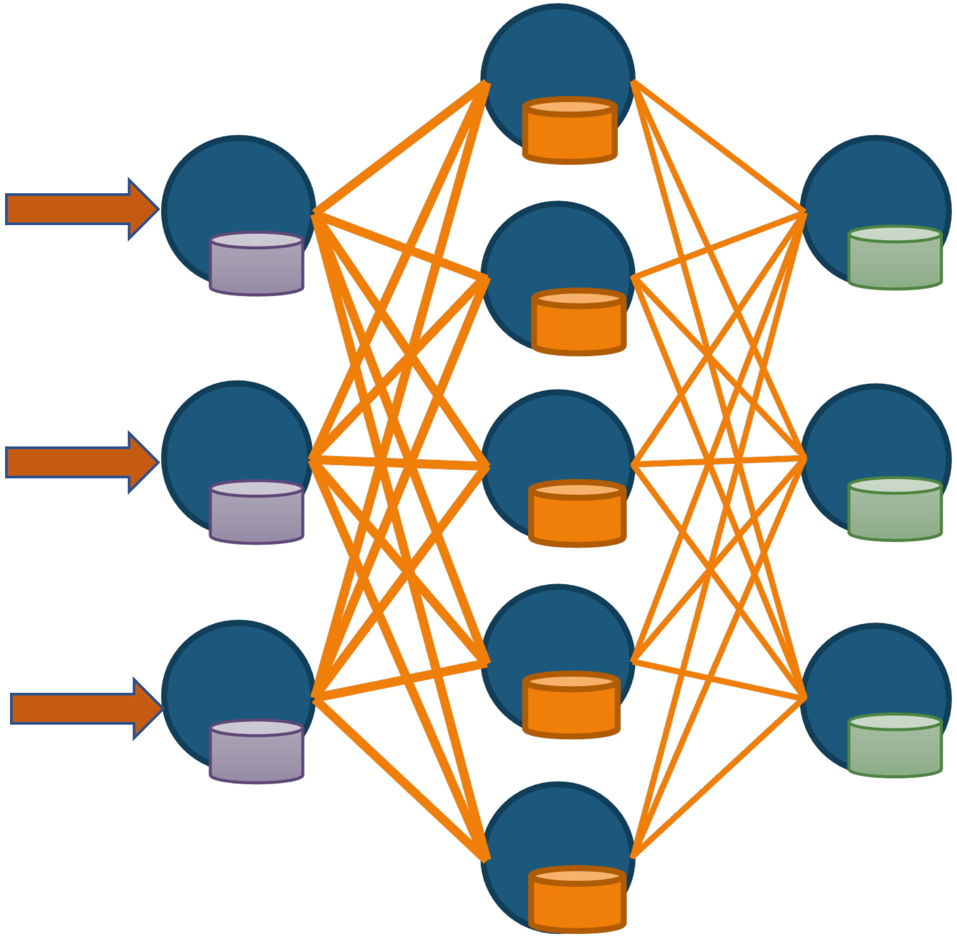
Enable *robust scale-out performance* of
stateful streaming queries using *high-speed networks*

Motivation

Distributed Stream Processing Engines are network-hungry!

Data Repartitioning as primitive for aggregations and joins.

Often the network is a bottleneck!

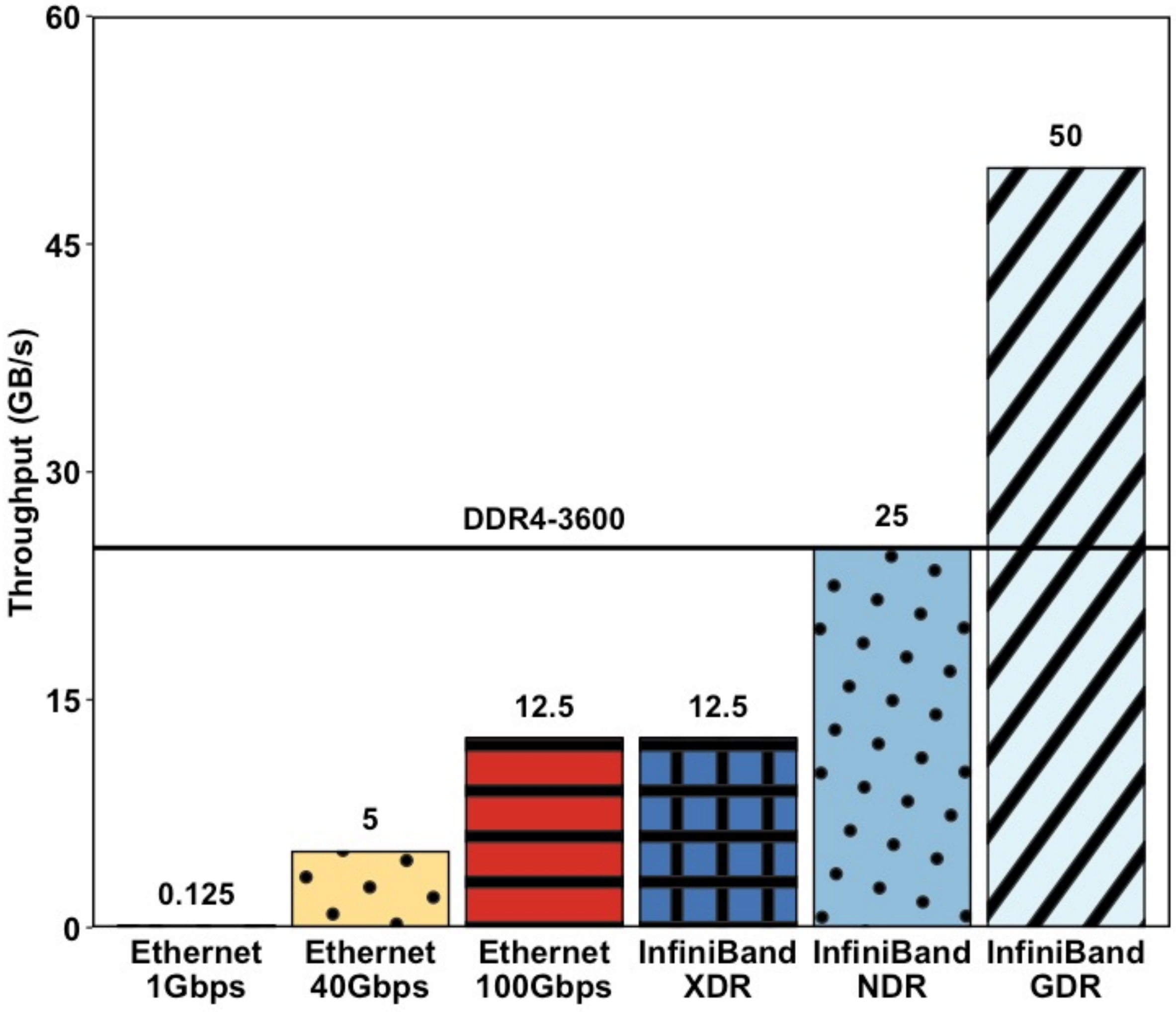


Motivation

Distributed Stream Processing Engines are network-hungry!

Data Repartitioning as primitive for aggregations and joins.

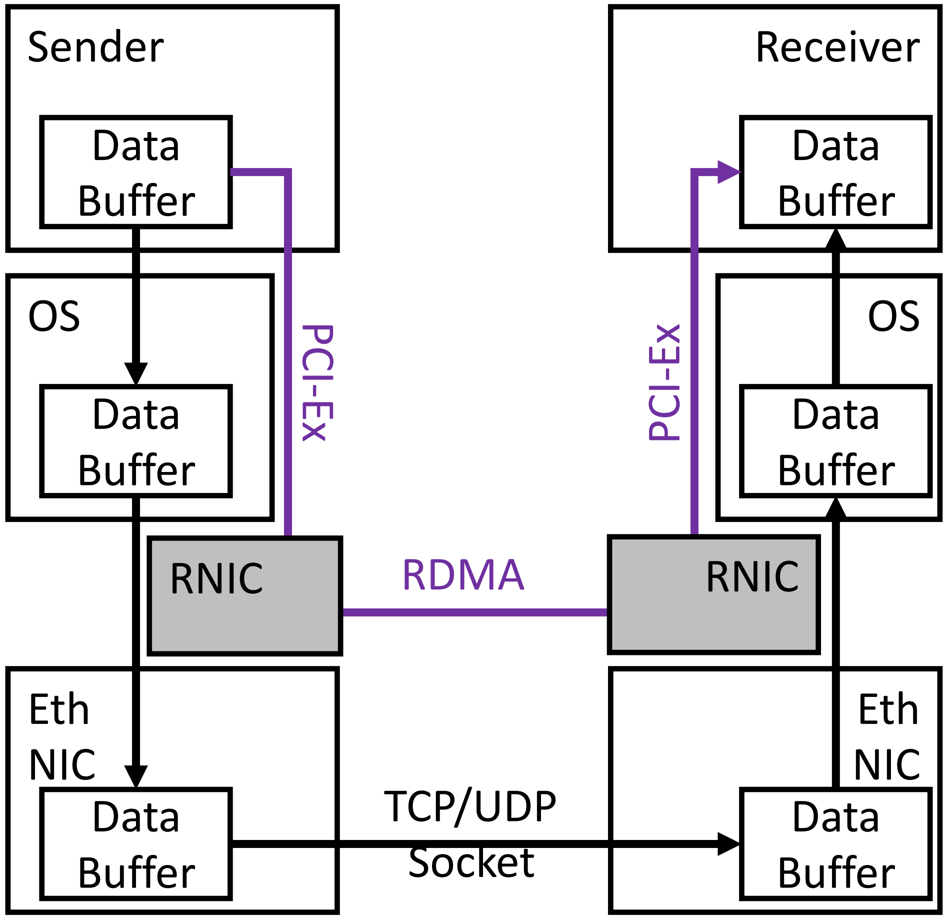
Often the network is a bottleneck!



Source: InfiniBandTA

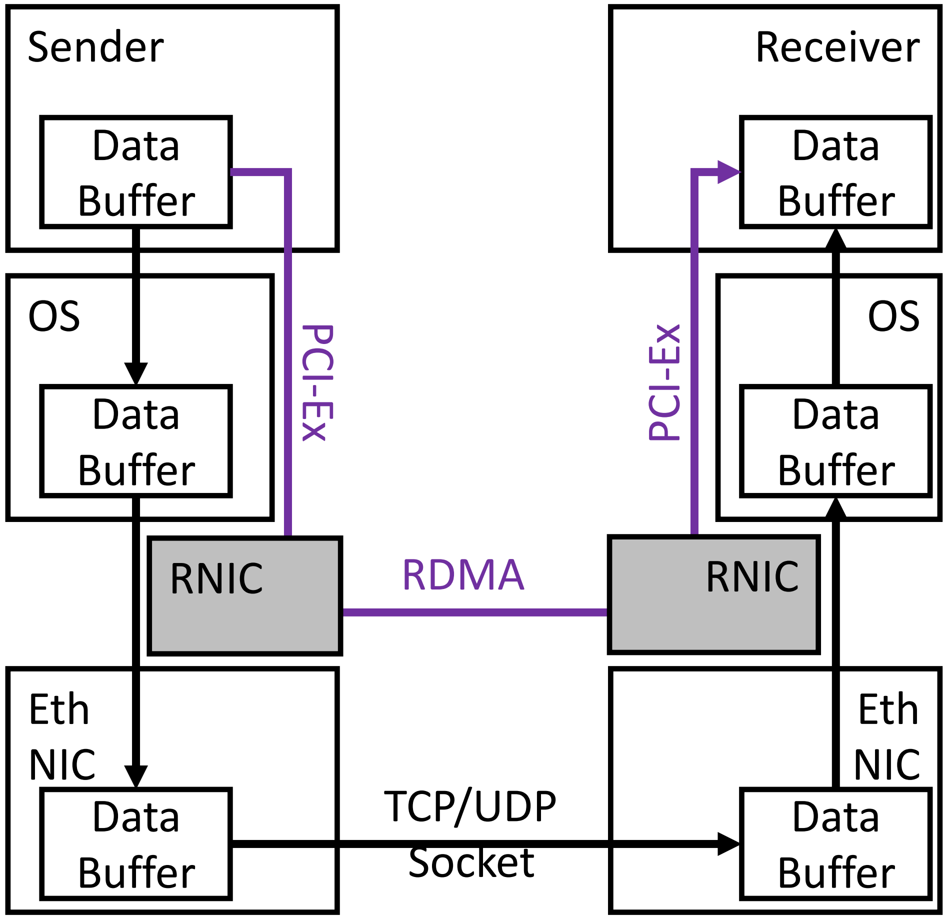
What is RDMA?

- OS kernel stack bypass and zero-copy transfer
- Message-oriented via verbs API
- Current DBMS use RDMA to accelerate batch OLAP and OLTP workloads



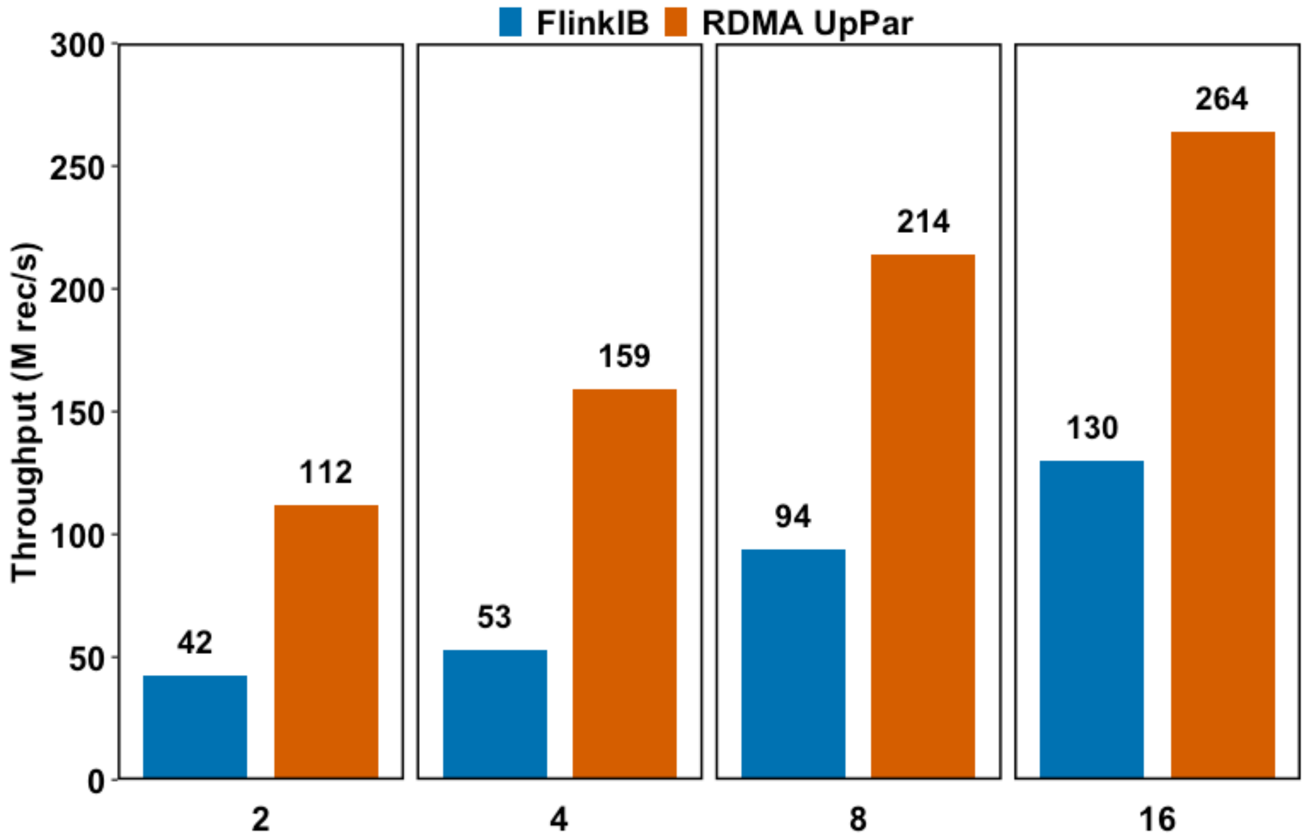
What is RDMA?

- OS kernel stack bypass and zero-copy transfer
- Message-oriented via verbs API
- Current DBMS use RDMA to accelerate batch OLAP and OLTP workloads



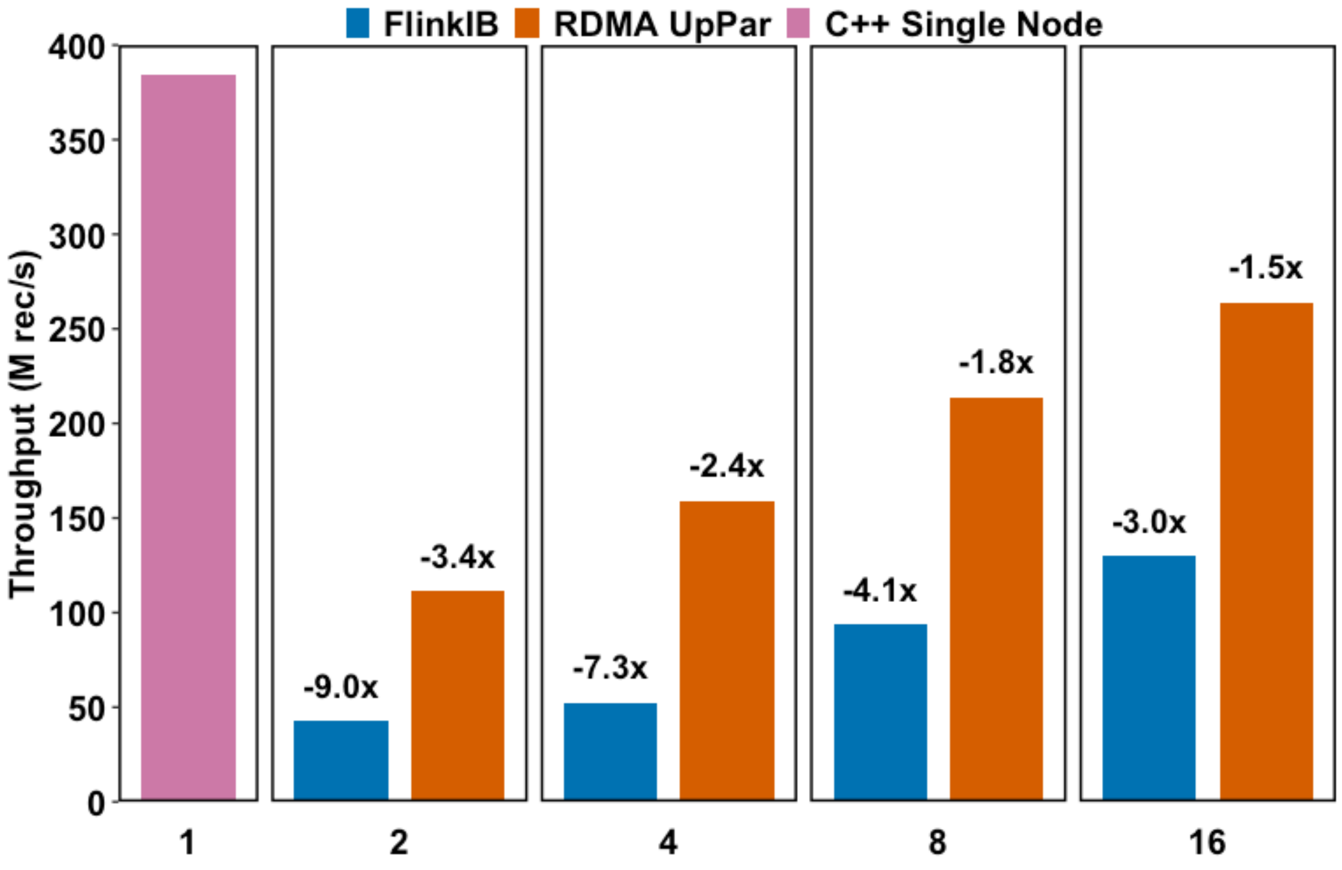
Can I apply RDMA acceleration to Stream Processing Engines (SPEs)?
If SPEs are often network-bound, adding a fast network is a good idea!

Can SPEs benefit from a fast network?



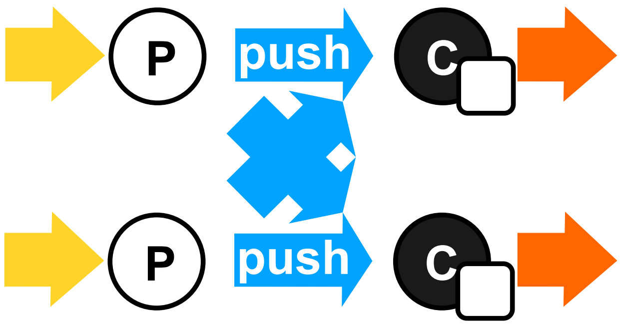
YSB Benchmark on a 16-node cluster with 100 Gbit Mellanox NICs using FlinkIB and RDMA UpPar

Can SPEs benefit from a fast network?



Adding a fast network to an SPE does not generally make it run faster, if execution is CPU-Bound.

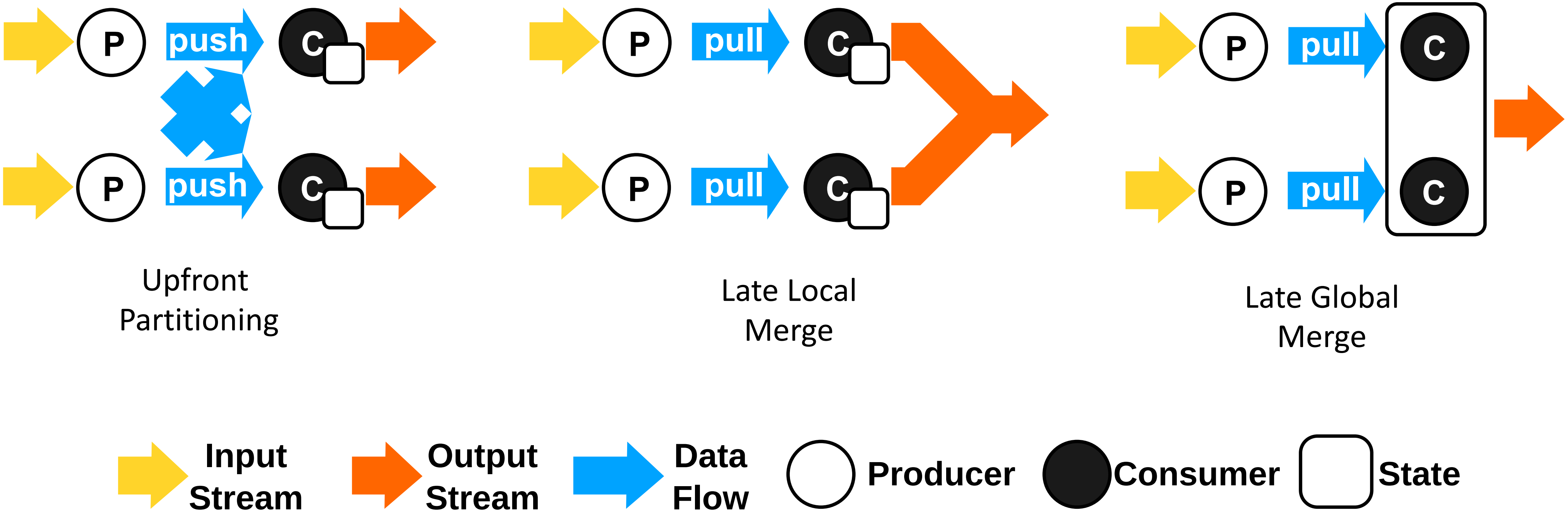
Can we reuse any insight from scale-up SPEs?



Upfront
Partitioning

Source: Analyzing efficient stream processing on modern hardware, S. Zeuch, B. Del Monte, et al., VLDB 2019

Can we reuse any insight from scale-up SPEs?

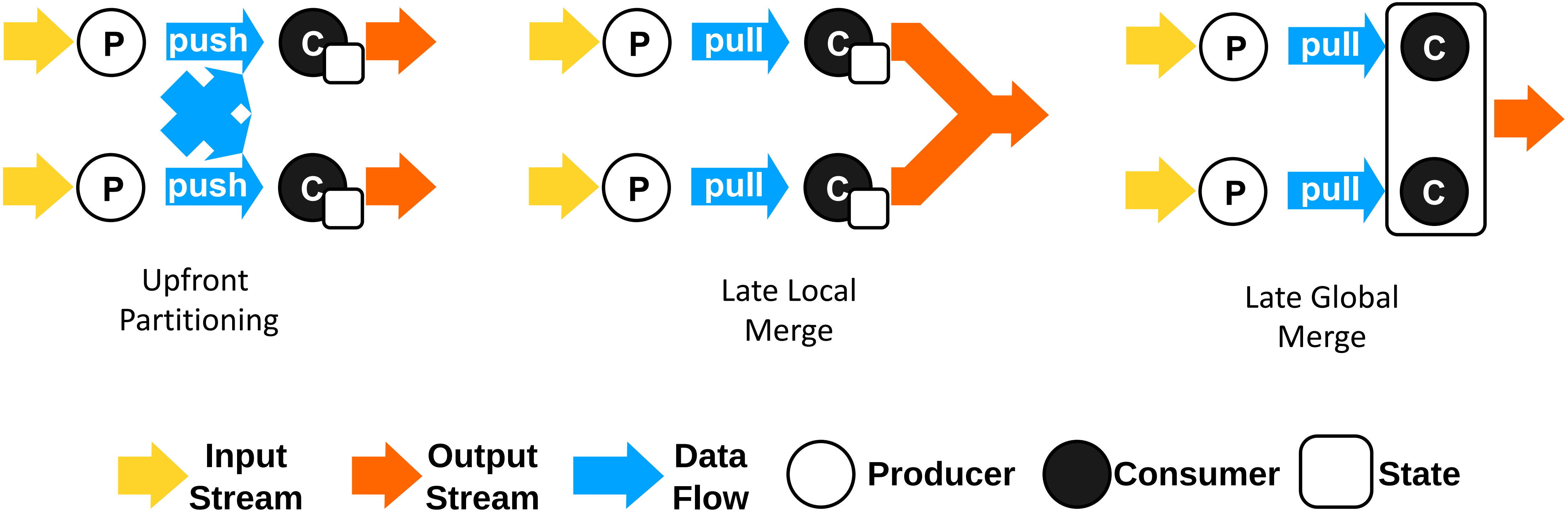


Upfront Partitioning

Late Local Merge

Late Global Merge

Can we reuse any insight from scale-up SPEs?



Partitioning makes SPEs CPU-Bound, when processing high-speed data streams.
Use alternative processing model for scale-up SPEs.

Architectural Change: Design Challenges

1. Efficient streaming computations
   - Replace data re-partitioning with RDMA-enabled late merge

Architectural Change: Design Challenges

1. Efficient streaming computations
   - Replace data re-partitioning with RDMA-enabled late merge

2. Efficient data transfer
   - RDMA performance depends on low-level factors

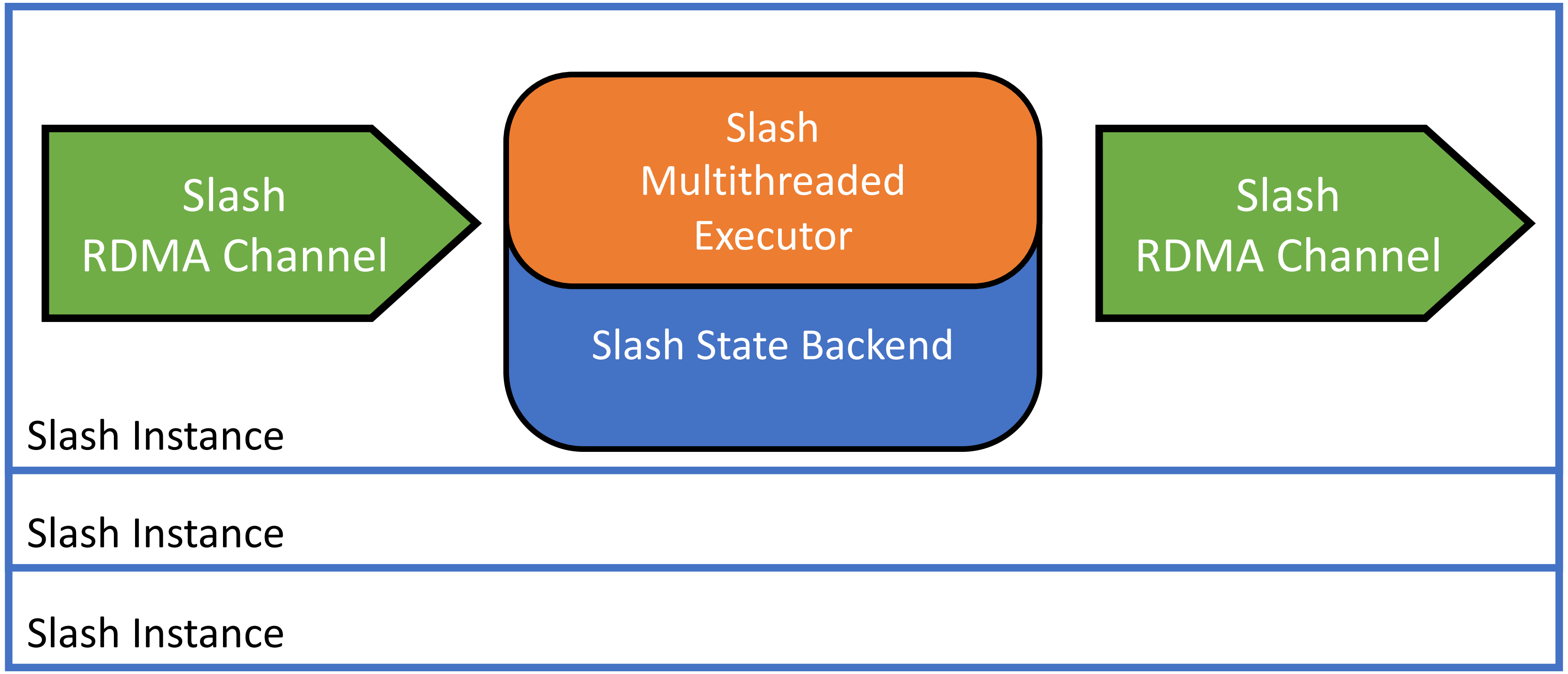
Architectural Change: Design Challenges

1. Efficient streaming computations
   - Replace data re-partitioning with RDMA-enabled late merge

2. Efficient data transfer
   - RDMA performance depends on low-level factors

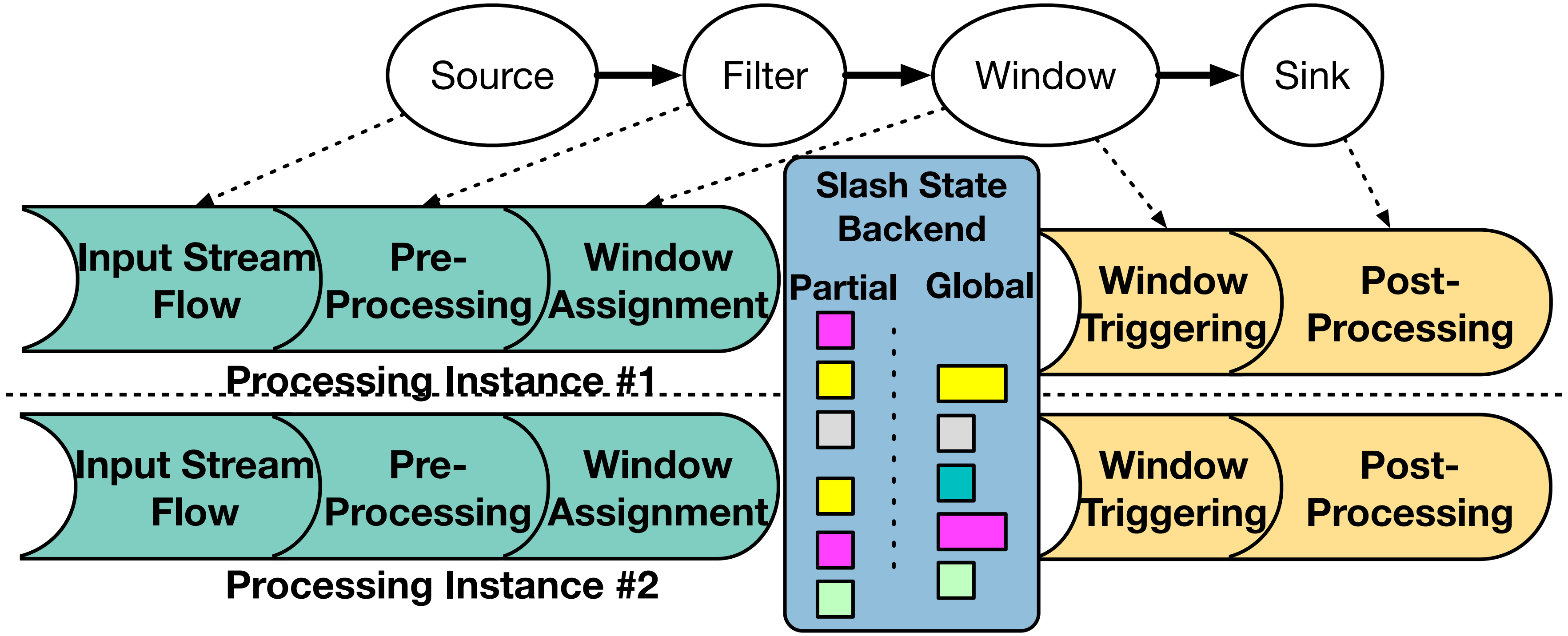
3. Consistent stateful computations
   - Progress tracking and exactly-once state updates

Our prototype: Slash



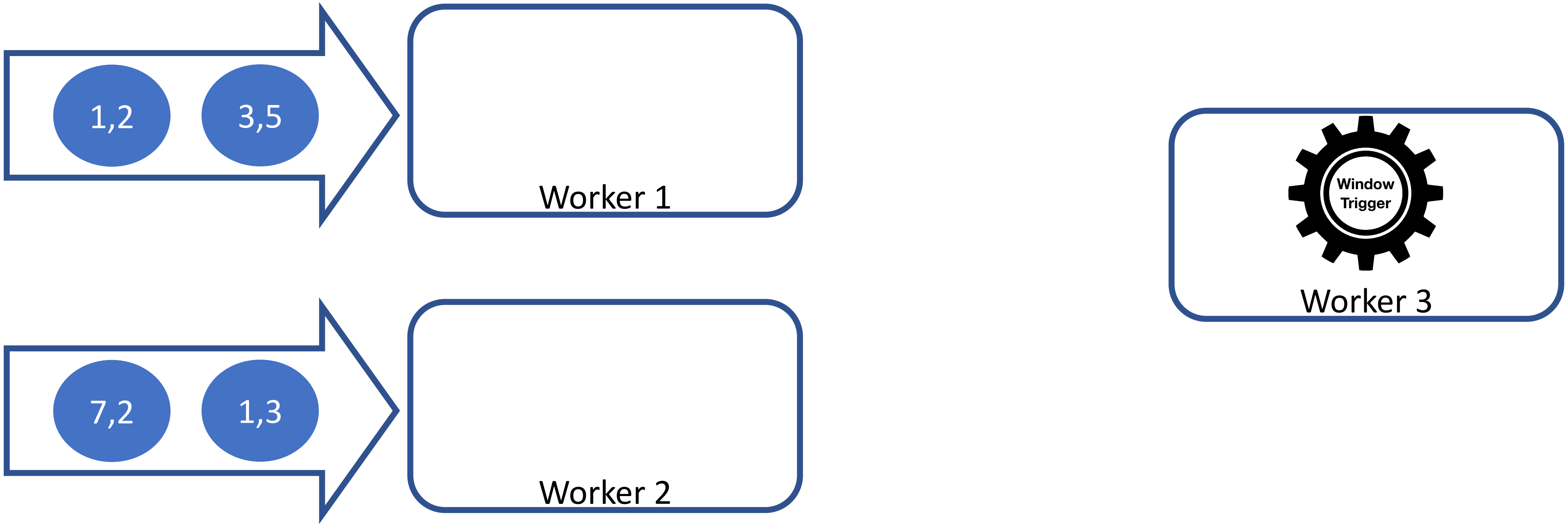
Slash’s design principle: make the common case fast

Slash: Stateful Query Execution

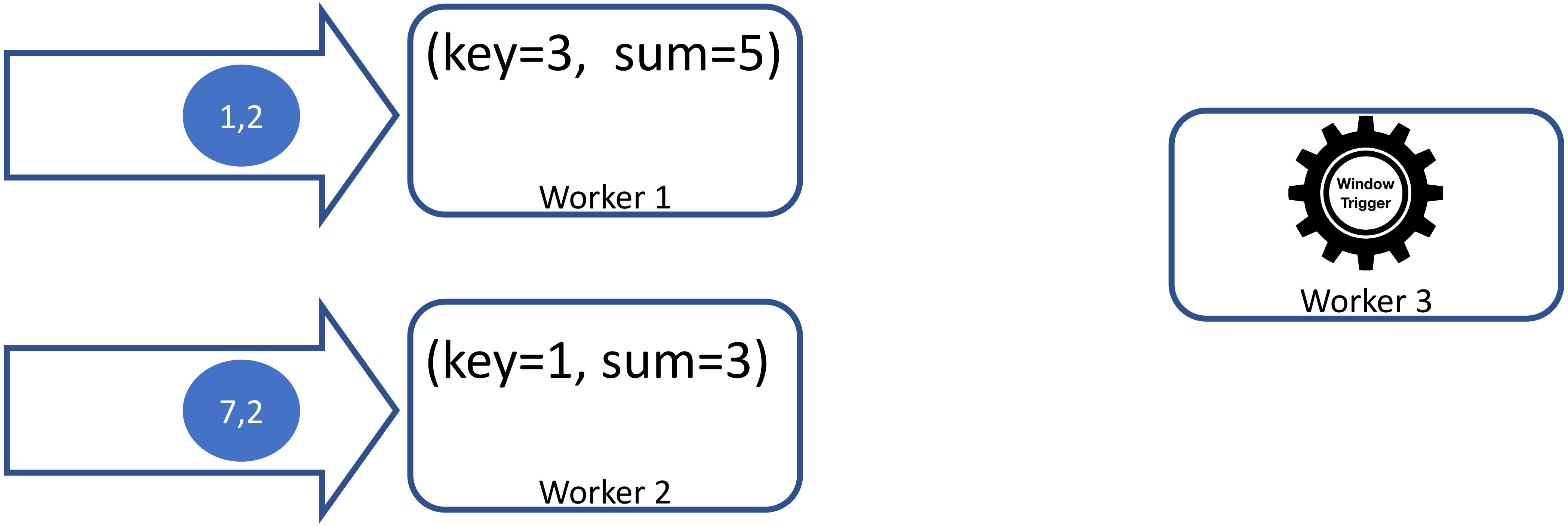


Eager computation of partial states followed by
lazy late merging of partial states in a consistent state

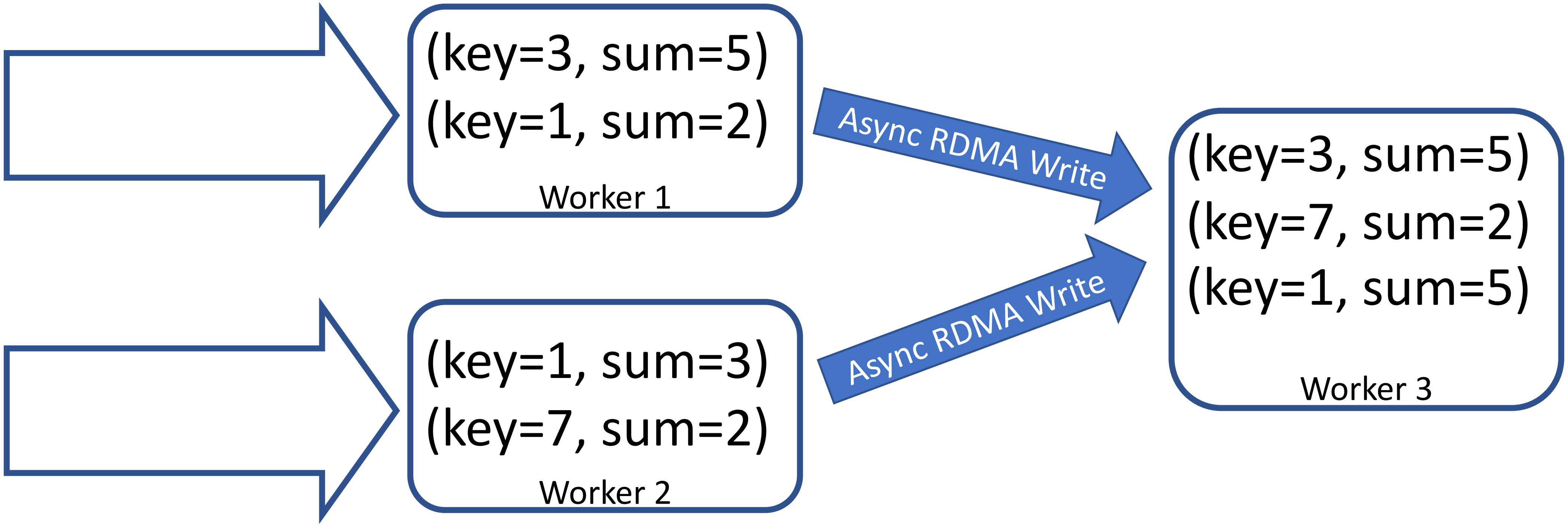
Slash in action: Window Aggregation



Slash in action: Window Aggregation

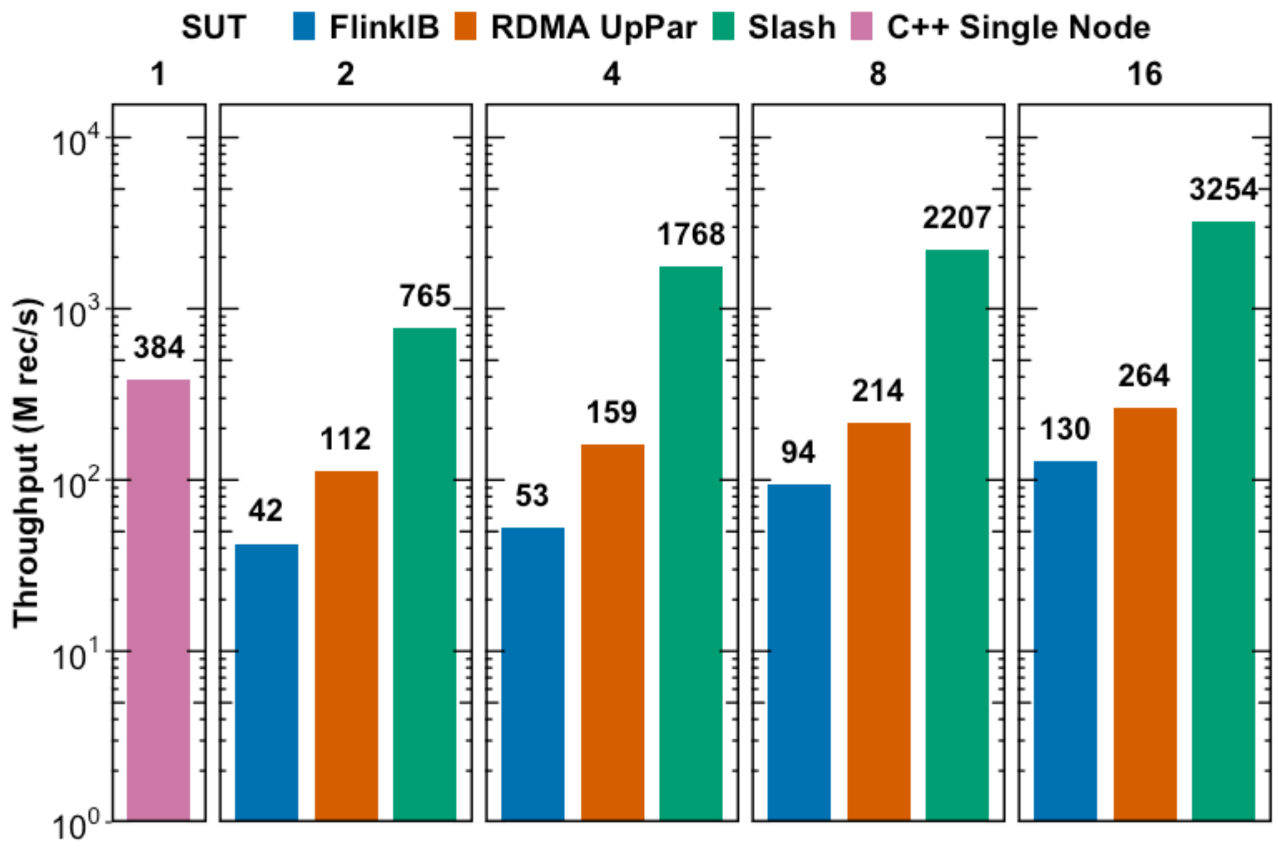


Slash in action: Window Aggregation



Slash relies on log-structured storage, epoch-based synchronization, and CRDT for late lazy merging.

Performance Evaluation



Slash outperforms the baselines by a factor up to 25x.

Performance gain explained

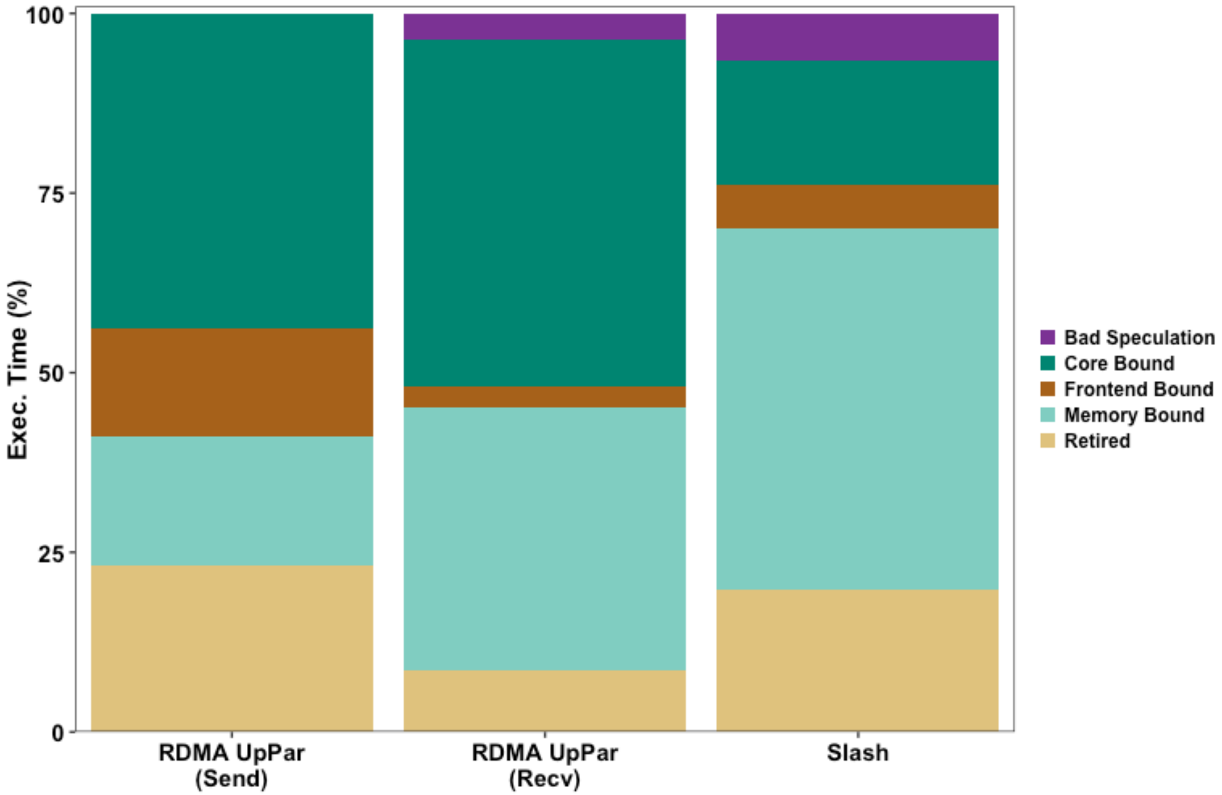
	IPC	Instr./ Rec.	Cyc./ Rec.
RDMA UpPar	0.6	166	274
	0.4	78	276
Slash	0.9	42	53

- RDMA UpPar needs to execute partitioning logic on every record. Next, it computes a thread-local result on pre-partitioned data.
- Slash computes thread-local results that lazily merges (less data moved around).

Slash requires less instructions and cycles to process a single record.
Partitioning for RDMA UpPar is an expensive operation.

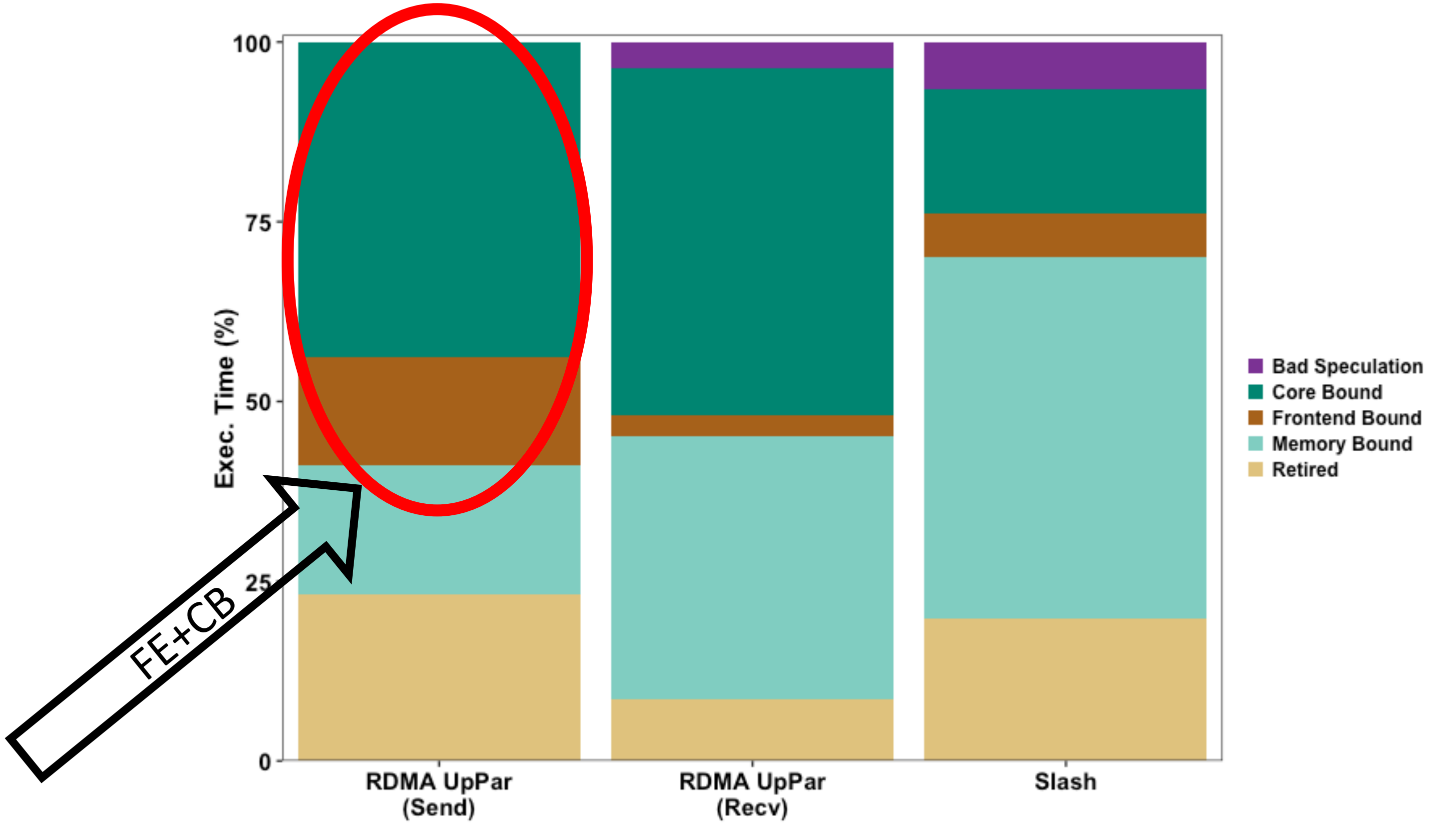
Performance gain explained

	IPC	Instr./ Rec.	Cyc./ Rec.
RDMA UpPar	0.6	166	274
	0.4	78	276
Slash	0.9	42	53



Performance gain explained

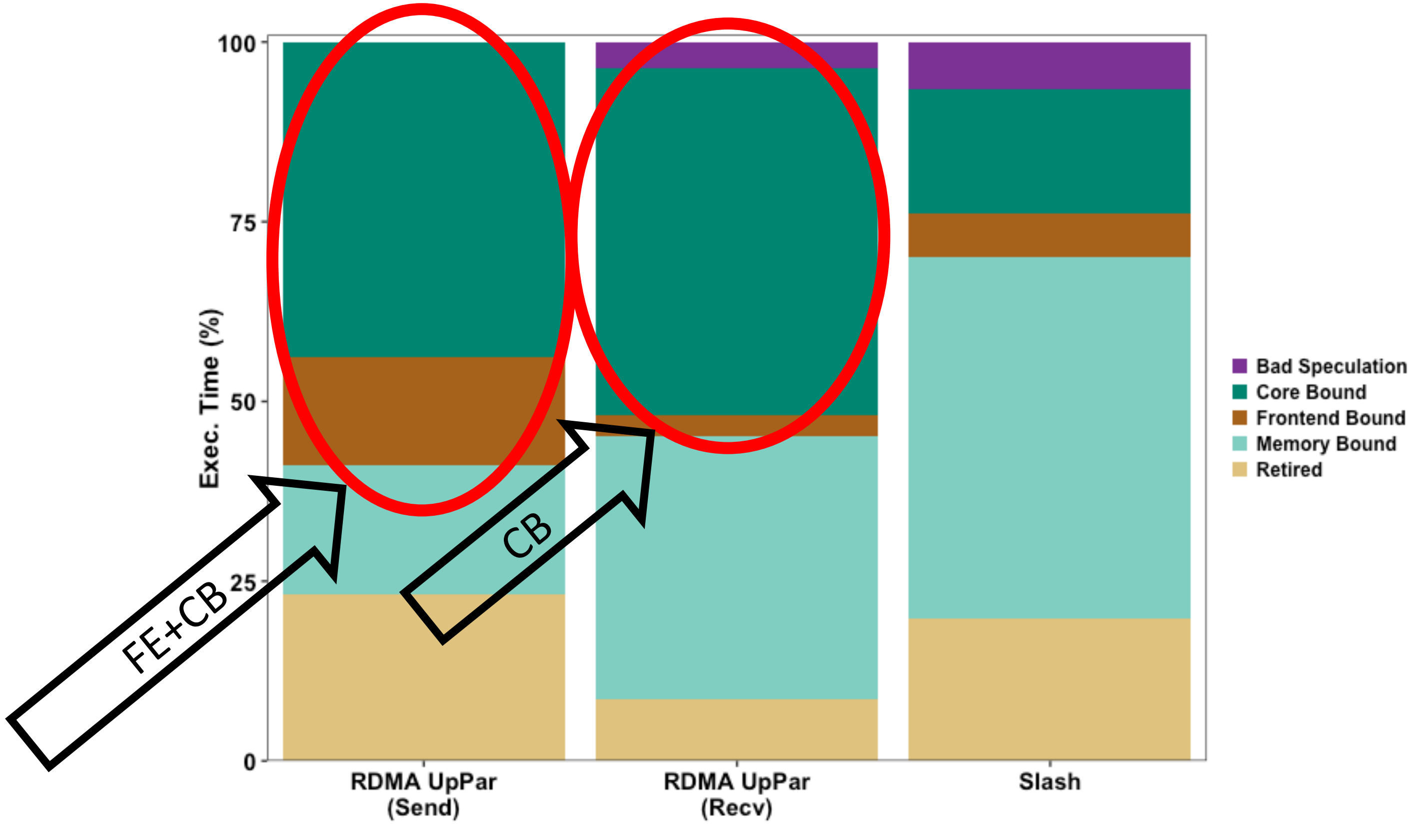
	IPC	Instr./ Rec.	Cyc./ Rec.
RDMA UpPar	0.6	166	274
	0.4	78	276
Slash	0.9	42	53



Execution of RDMA UpPar's sender suffers from complex code and spin waiting.
Reason: data partitioning.

Performance gain explained

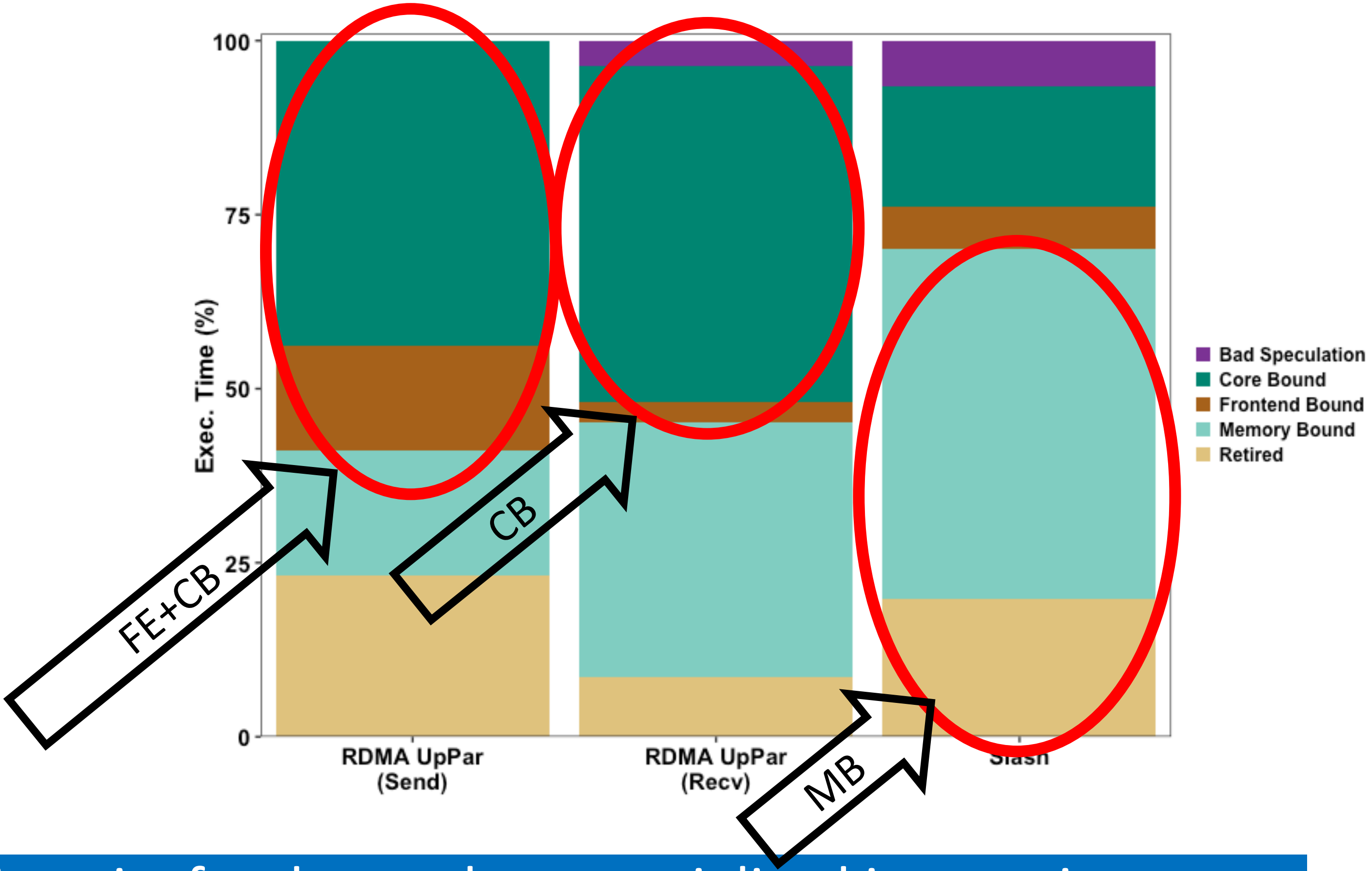
	IPC	Instr./ Rec.	Cyc./ Rec.
RDMA UpPar	0.6	166	274
	0.4	78	276
Slash	0.9	42	53



The execution of RDMA UpPar's receiver stalls due to spin waiting on sender.
RDMA UpPar is bound by partitioning speed (CPU).

Performance gain explained

	IPC	Instr./ Rec.	Cyc./ Rec.
RDMA UpPar	0.6	166	274
	0.4	78	276
Slash	0.9	42	53



Slash is memory-bound: it waits for data to be materialized into registers.
Slash is ultimately bound by memory speed.

Lesson learned

- Apply RDMA native acceleration and redesign internal data structures

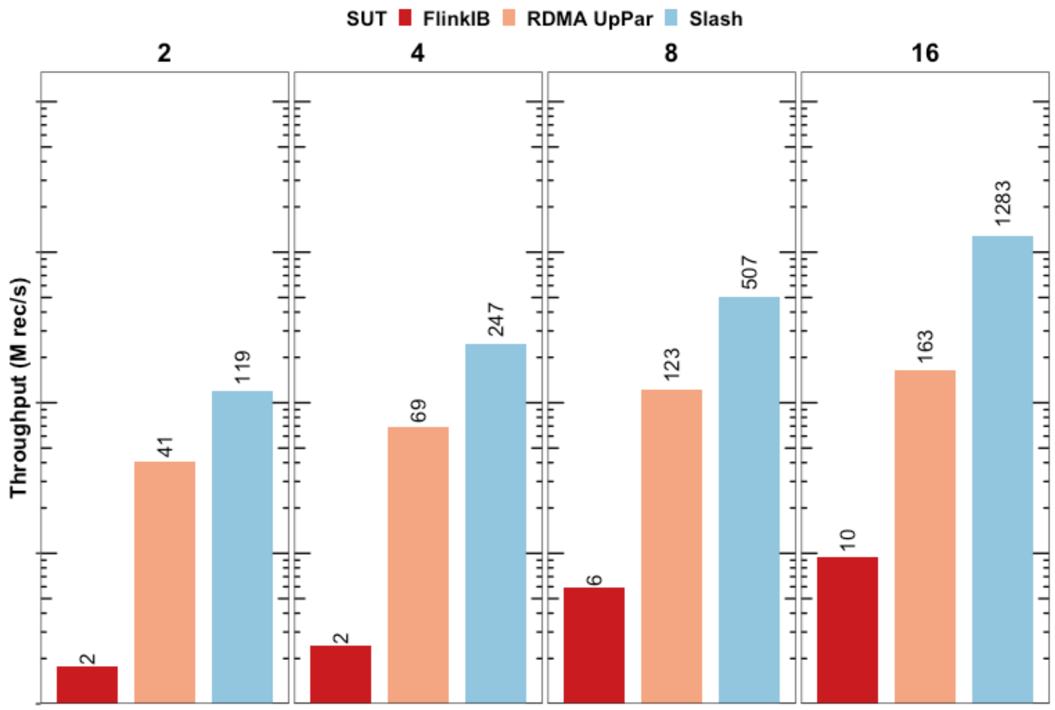
- Avoid data-repartitioning: it induces performance issues!

- Use instead lazy merging of eagerly computed partial state/results

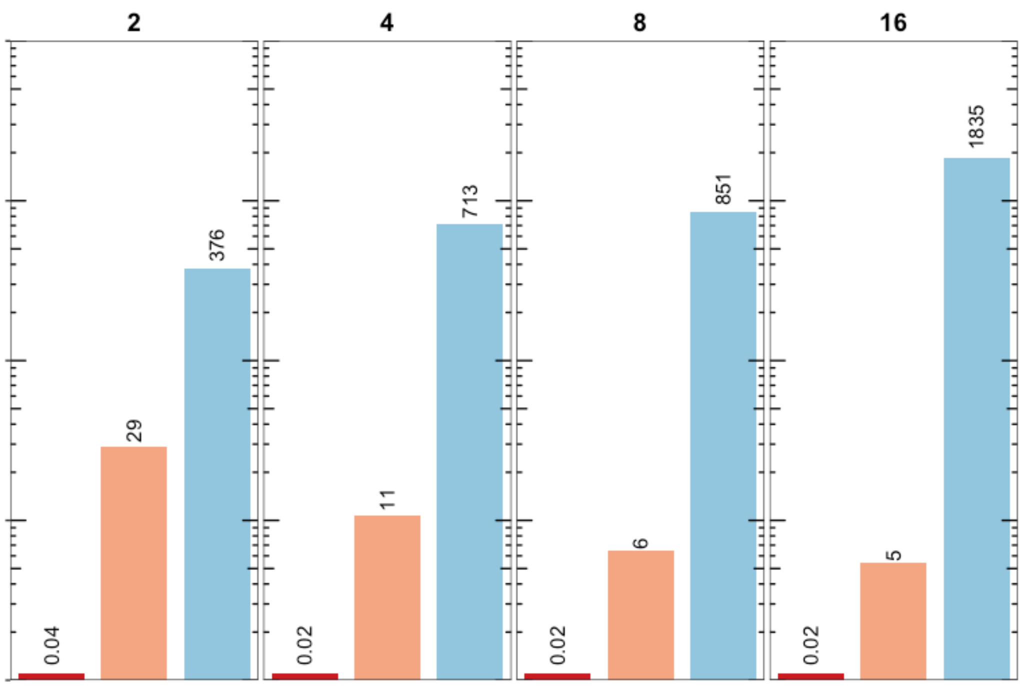
Summary and take-home

- We provide a **new system design** for RDMA-accelerated stateful stream processing.
- Slash attains up to a **factor of 25 increment** in throughput compared to the strongest baseline.
- Our drill-down analysis shows that **Slash is mainly memory-bound**, whereas **our strongest baseline is limited by partitioning speed**.

Show us more numbers



Nexmark Query 8



Google Cluster Monitoring

Slash outperforms baseline executing join operators and real-world workloads