Rethinking Stateful Stream Processing with RDMA

Bonaventura Del Monte - Steffen Zeuch - Tilmann Rabl - Volker Markl

2020 ACM SIGMOD Conference

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

Disclaimer

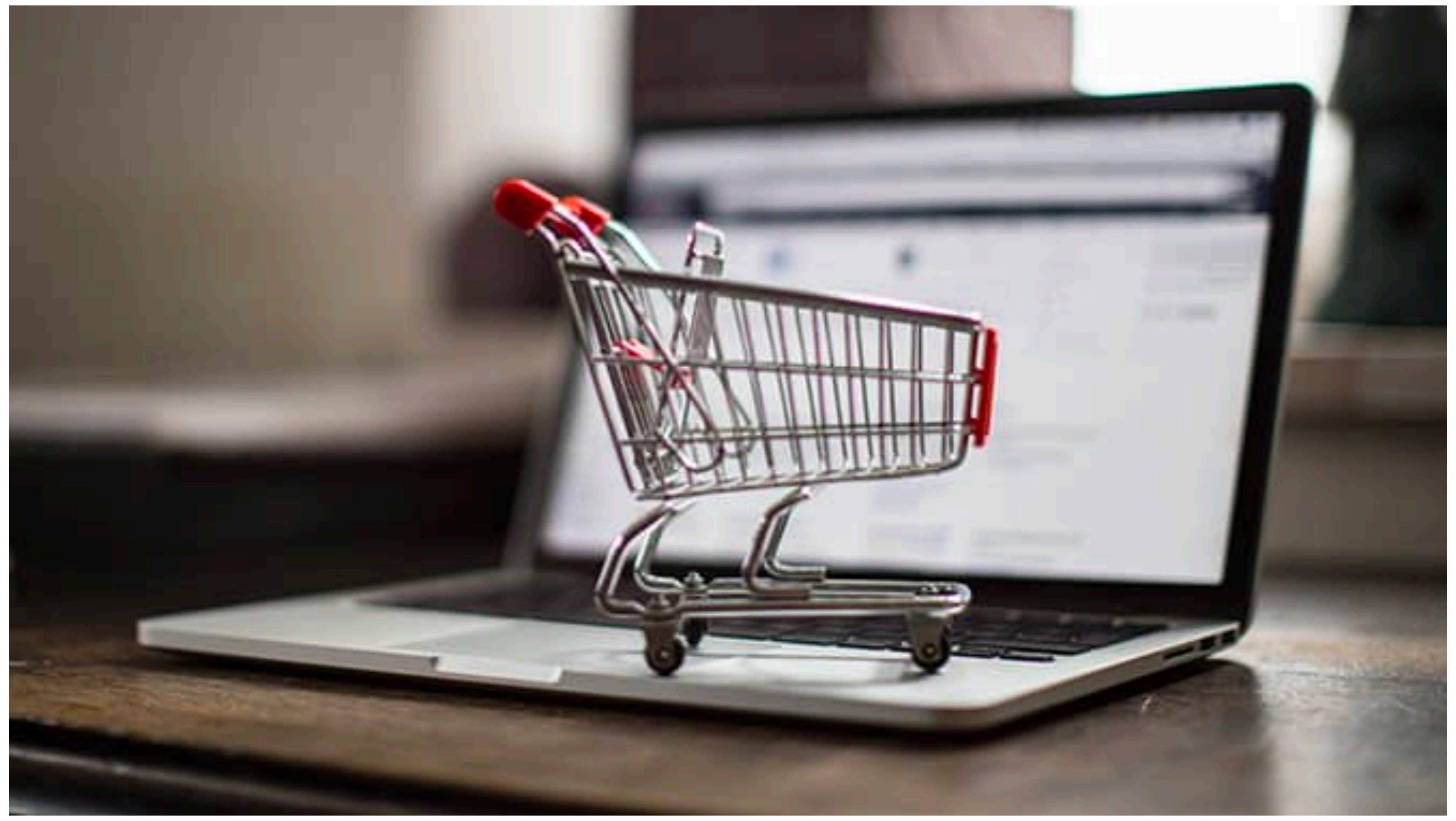
The work behind and content of this presentation were carried out while I was employed at TU Berlin

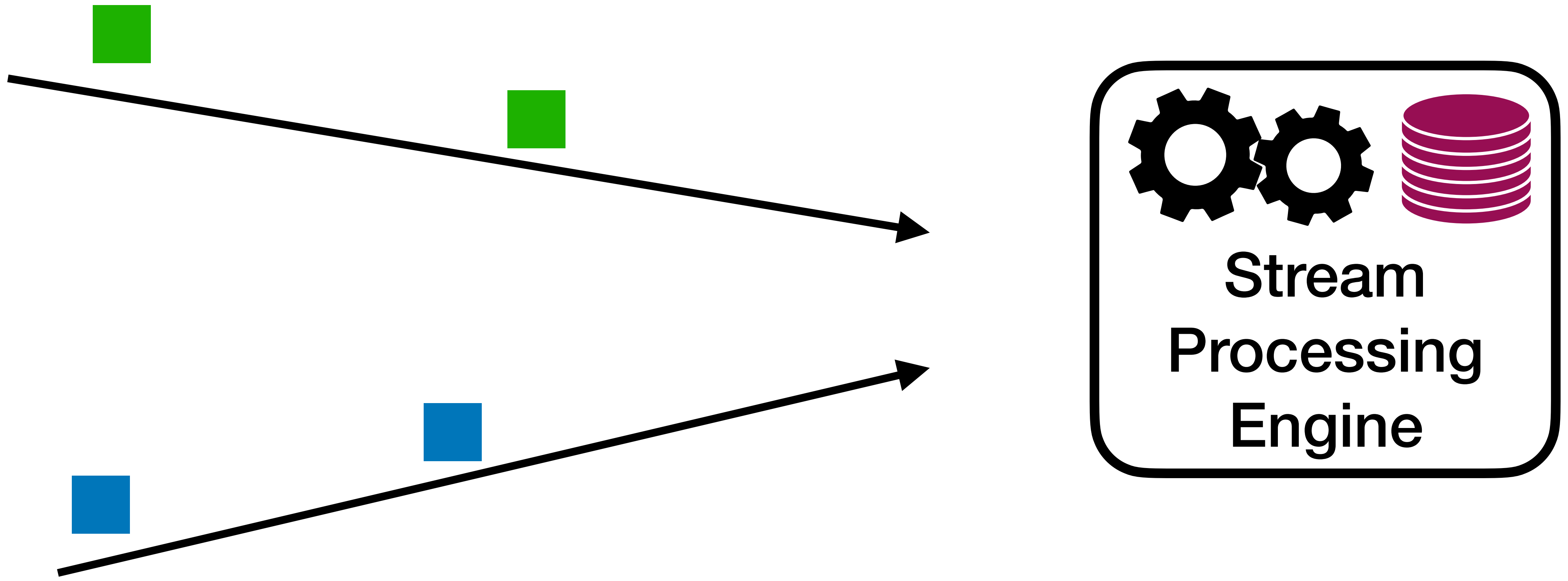
The content and opinions expressed in this talk do not represent Observe Inc.

What is this talk about?

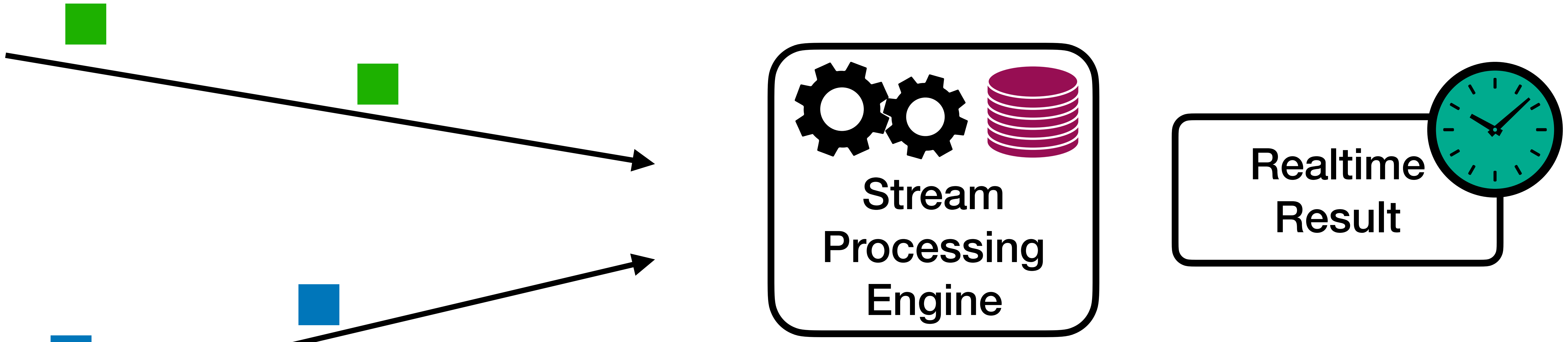
Enable *robust scale-out performance* for stateful streaming queries using *high-speed networks*

Stateful Streaming Analytics



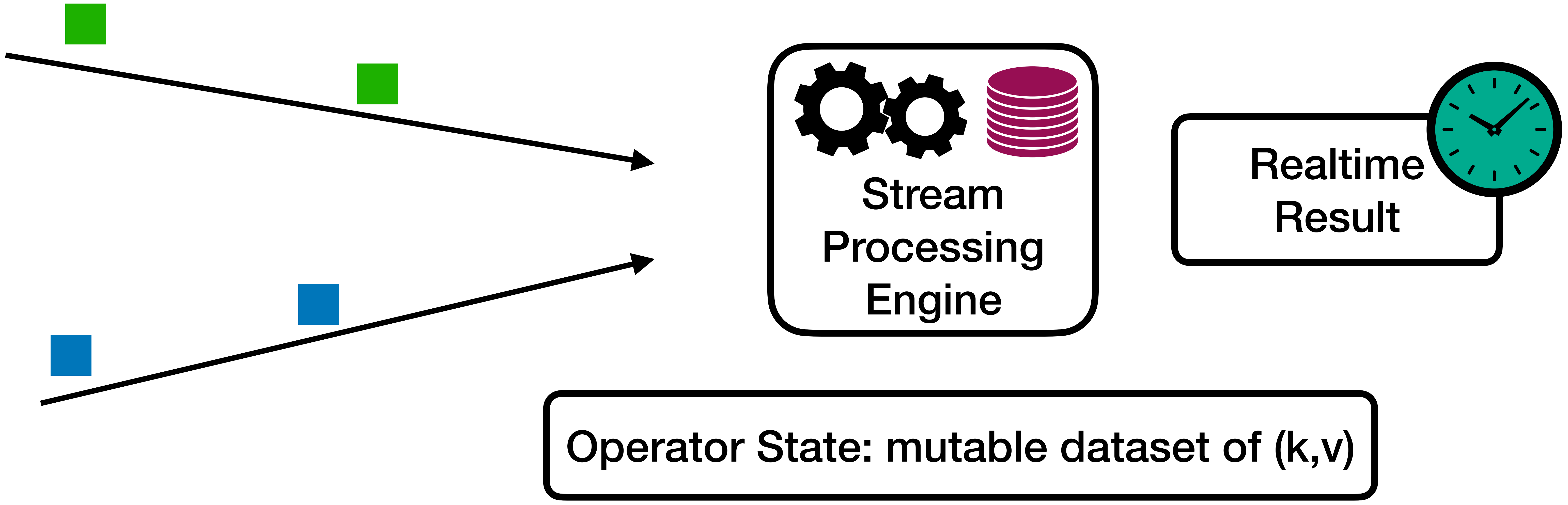


Stateful Streaming Analytics



Stateful Streaming Analytics

Credit Card Fraud Detection



Stateful Streaming Analytics

Credit Card Fraud Detection

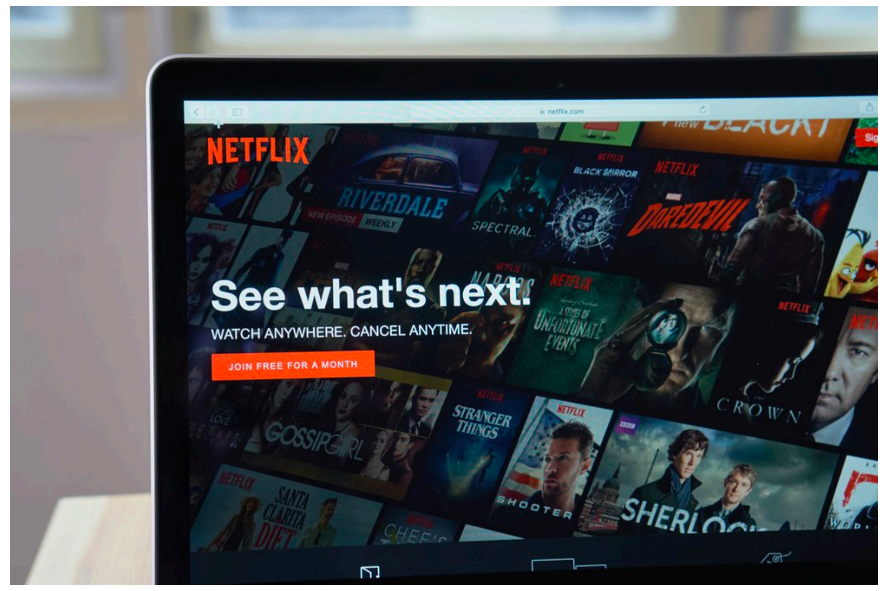
Stream Processing Engine

Realtime Result

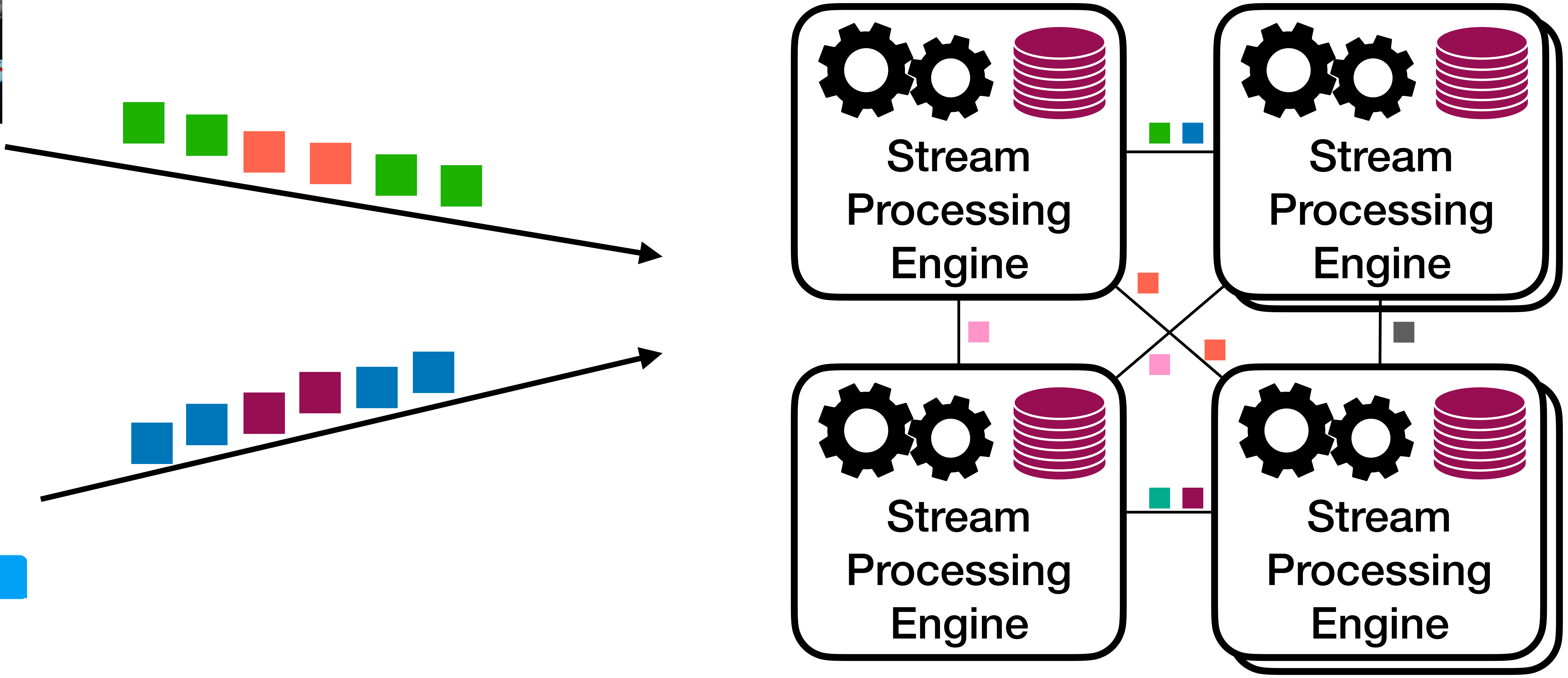
Operator State: mutable dataset of (k,v)

Windowed Aggregations, Windowed Joins, or Machine Learning Tasks

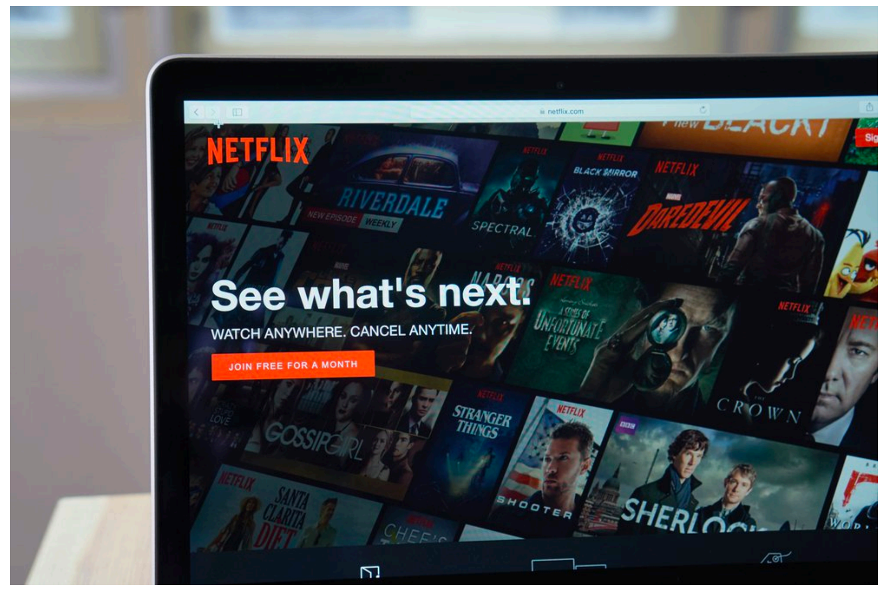
Stateful Streaming Analytics at Scale



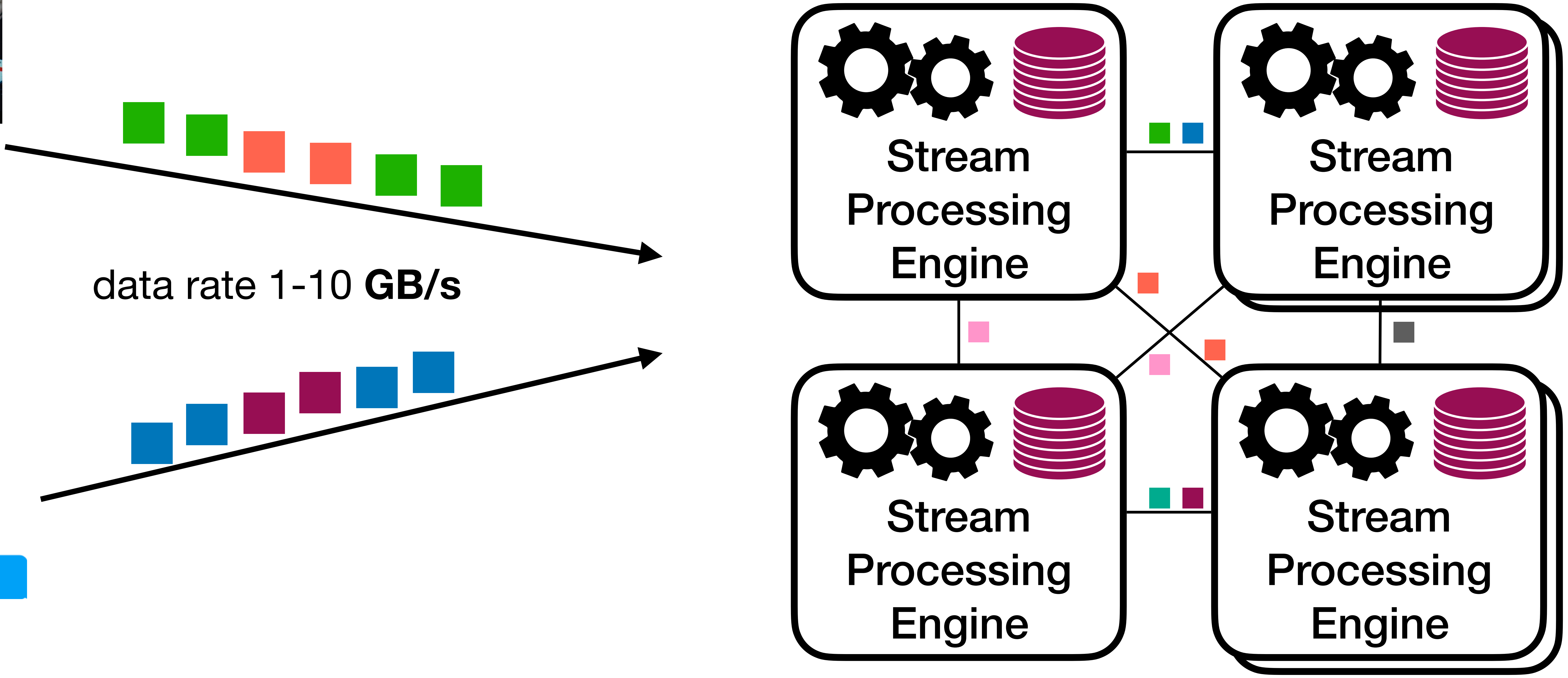
billions of events
high cardinality



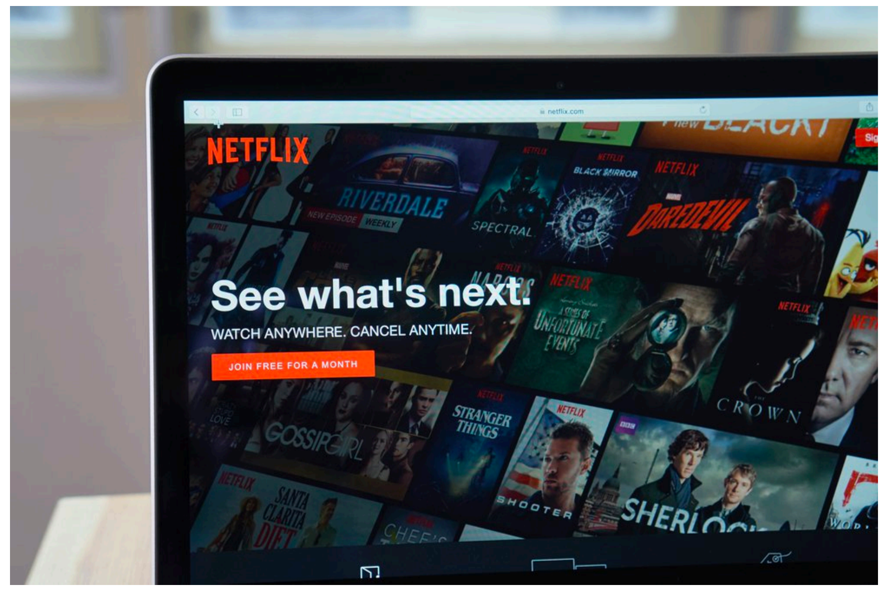
Stateful Streaming Analytics at Scale



billions of events
high cardinality



Stateful Streaming Analytics at Scale

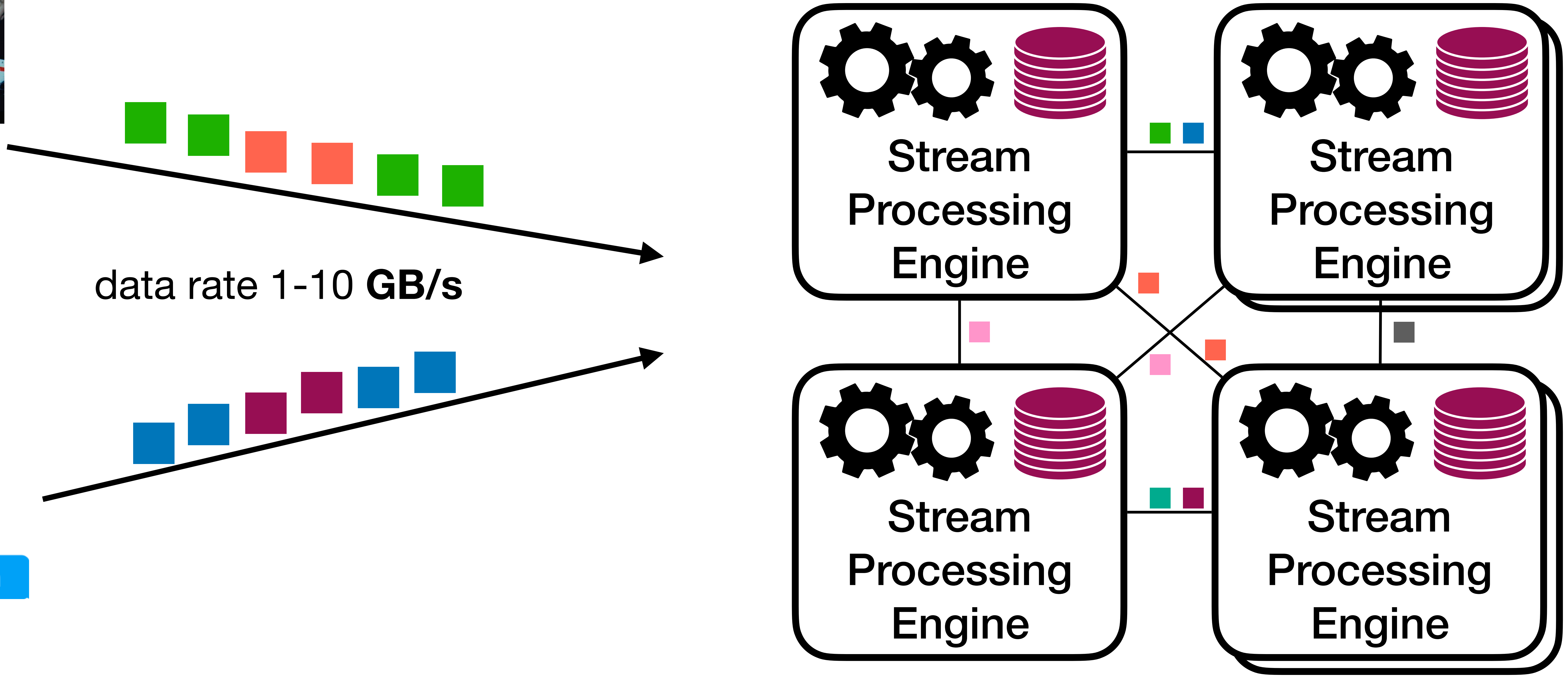


billions of events
high cardinality

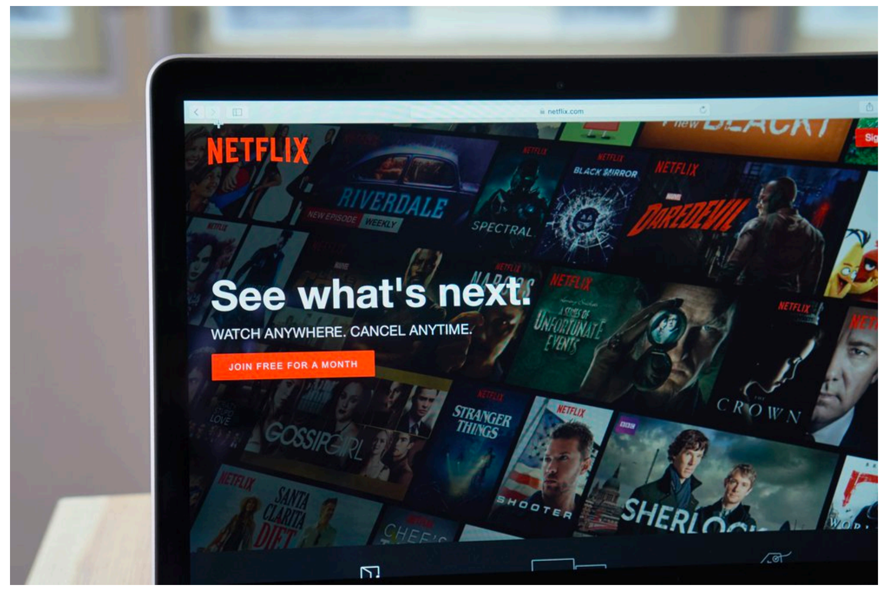
Credit Card Fraud Detection

state size 1-10 **TB**

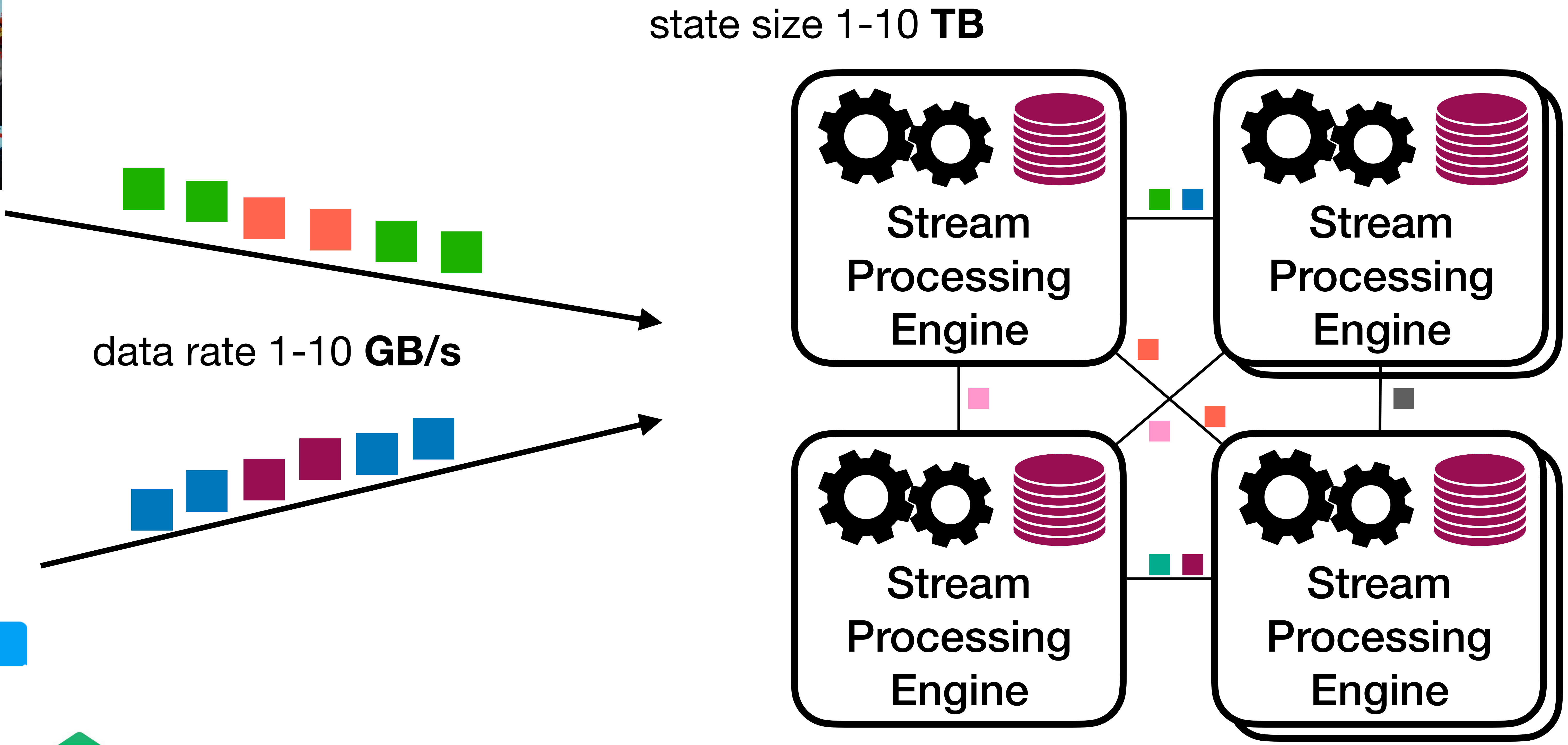
data rate 1-10 **GB/s**



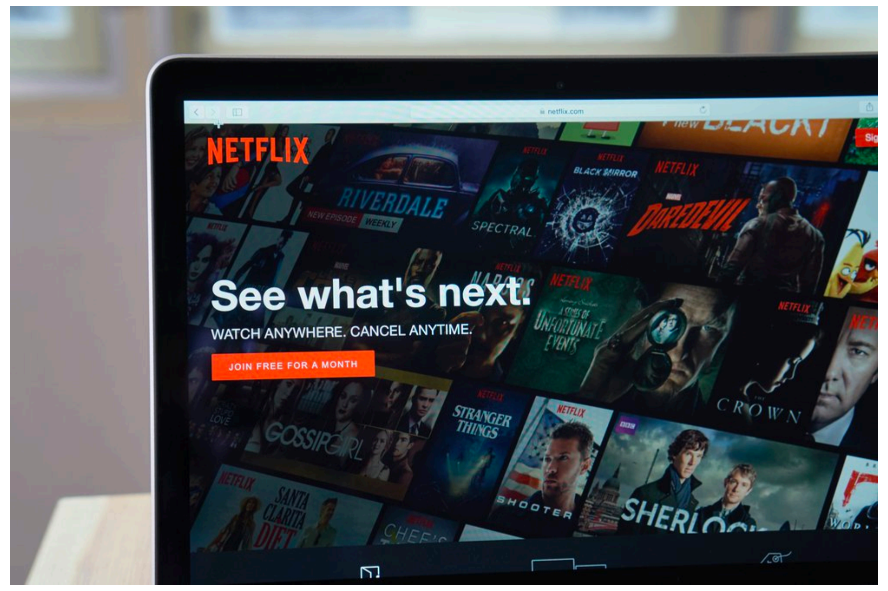
Stateful Streaming Analytics at Scale



billions of events
high cardinality



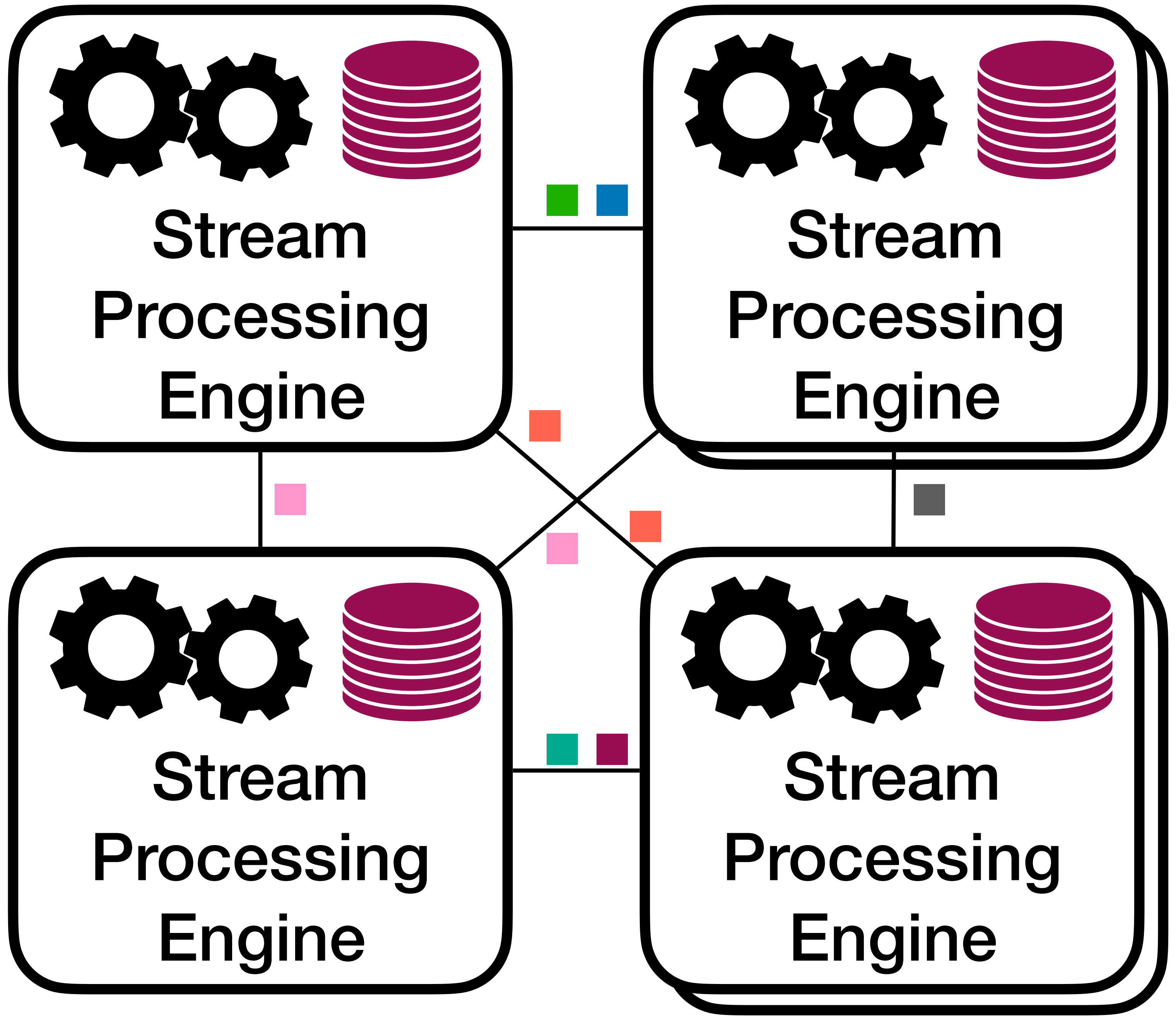
Stateful Streaming Analytics at Scale



billions of events
high cardinality

data rate 1-10 **GB/s**

state size 1-10 **TB**

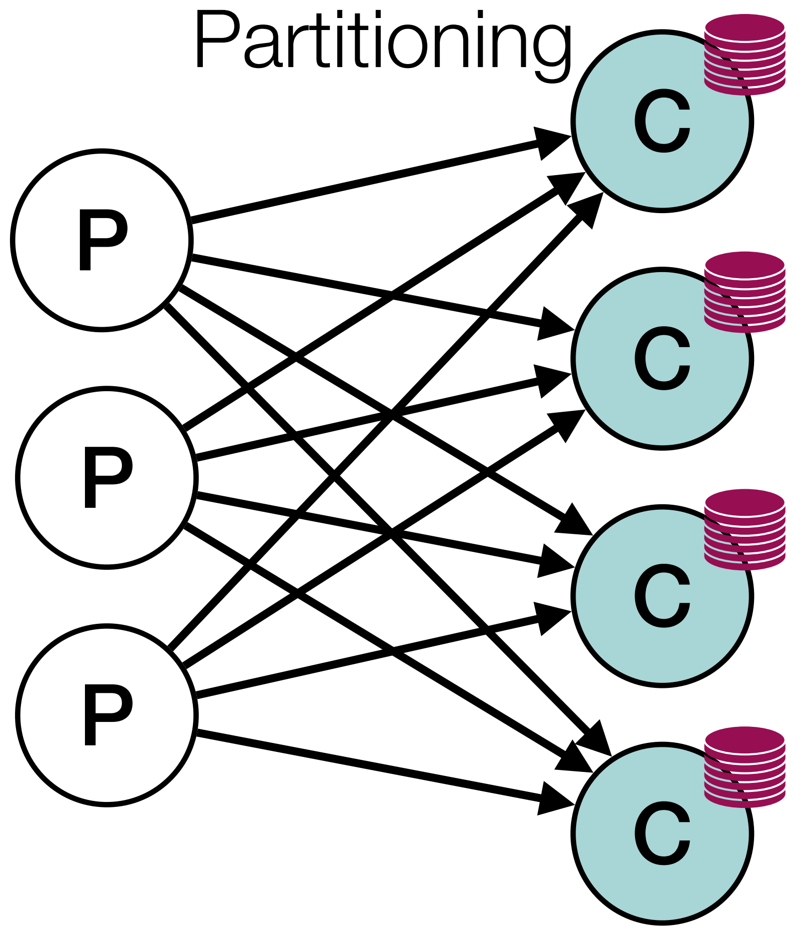


Sustain high-throughput stream processing with low-latency

We need <u>efficient</u>
stateful stream processing

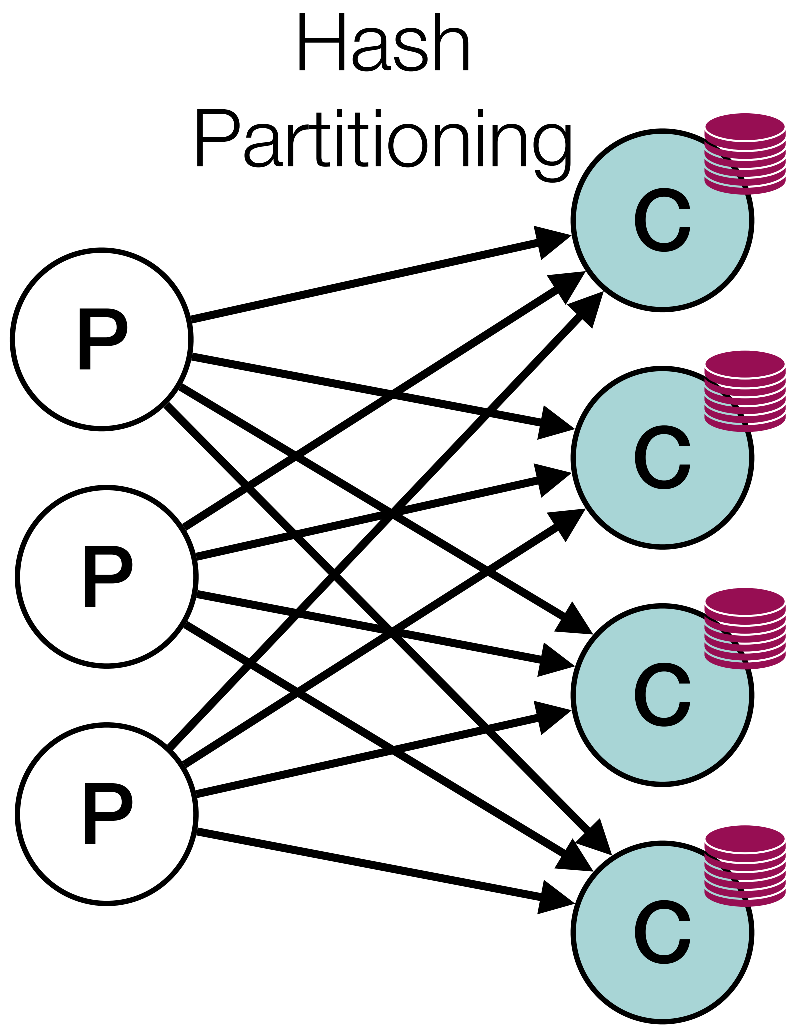
Stream Processing with high-speed network

Hash Partitioning



Data partitioning is a network intensive

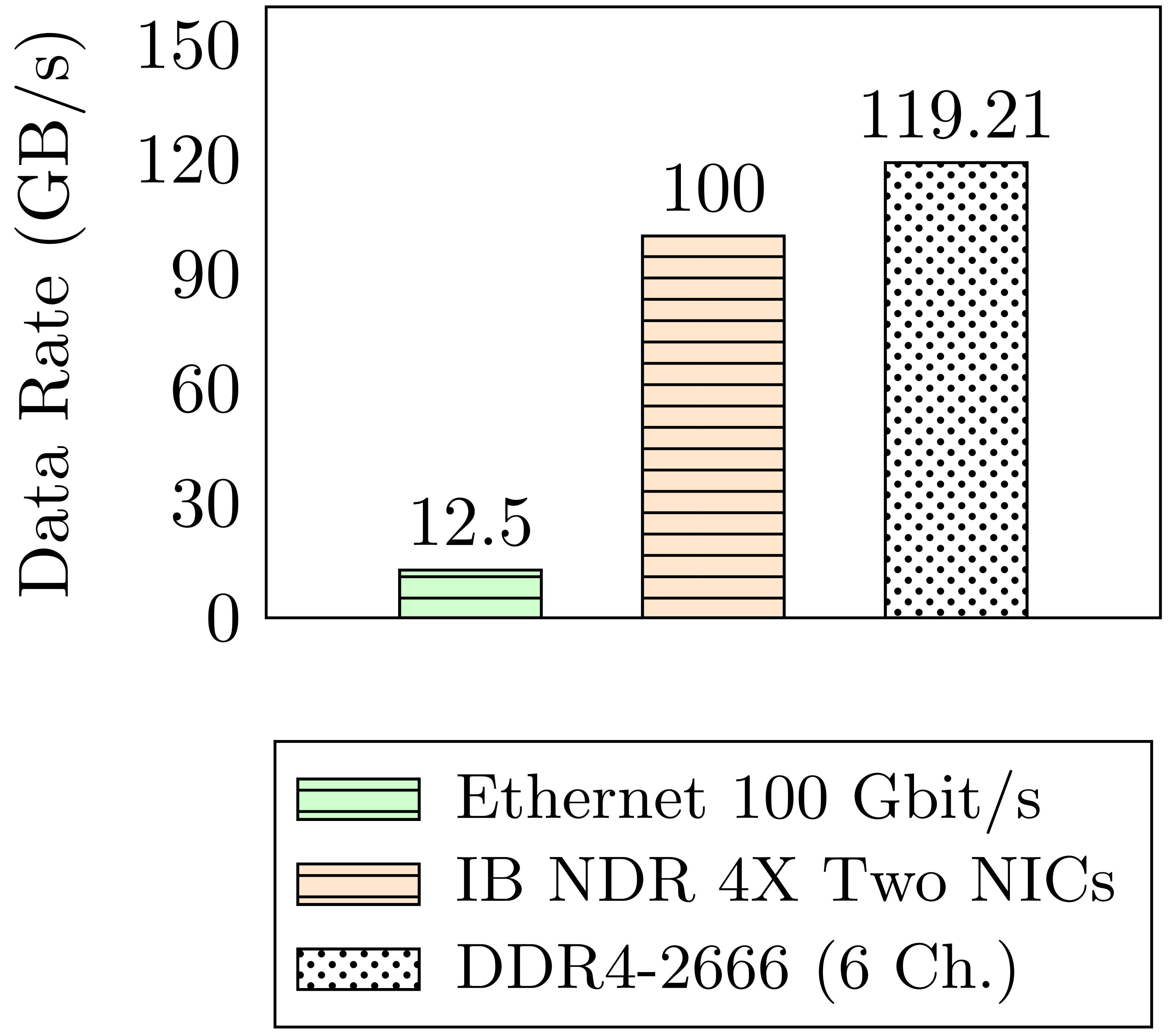
Stream Processing with high-speed network



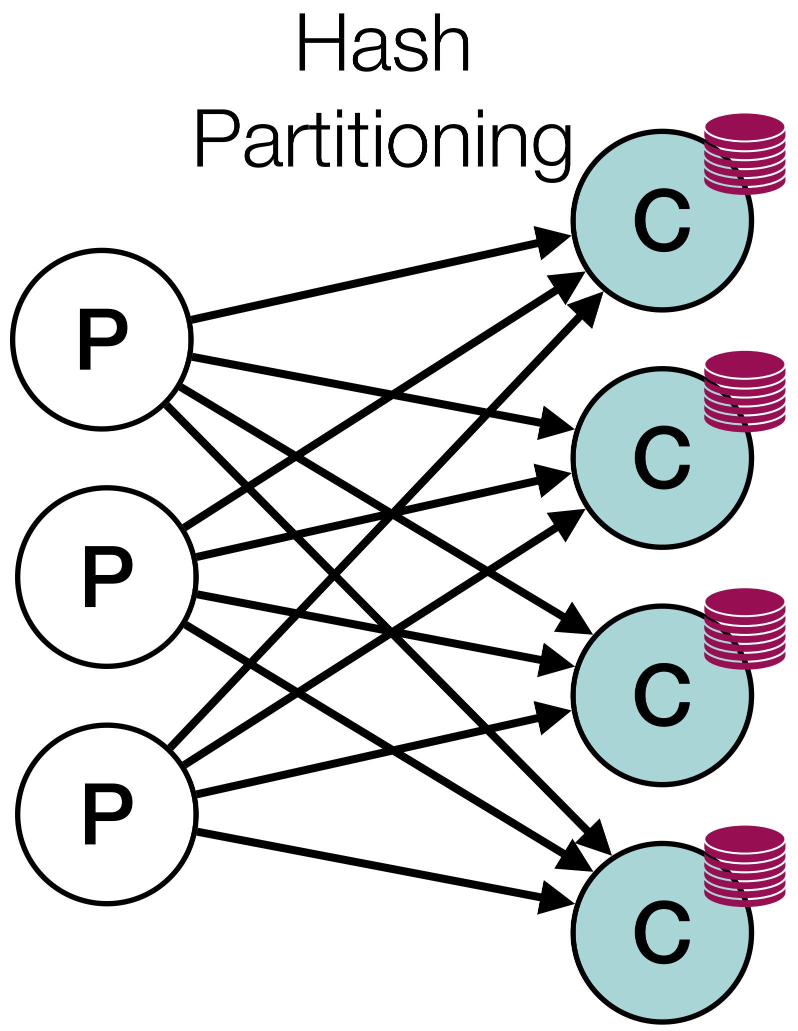
Data partitioning is a network intensive

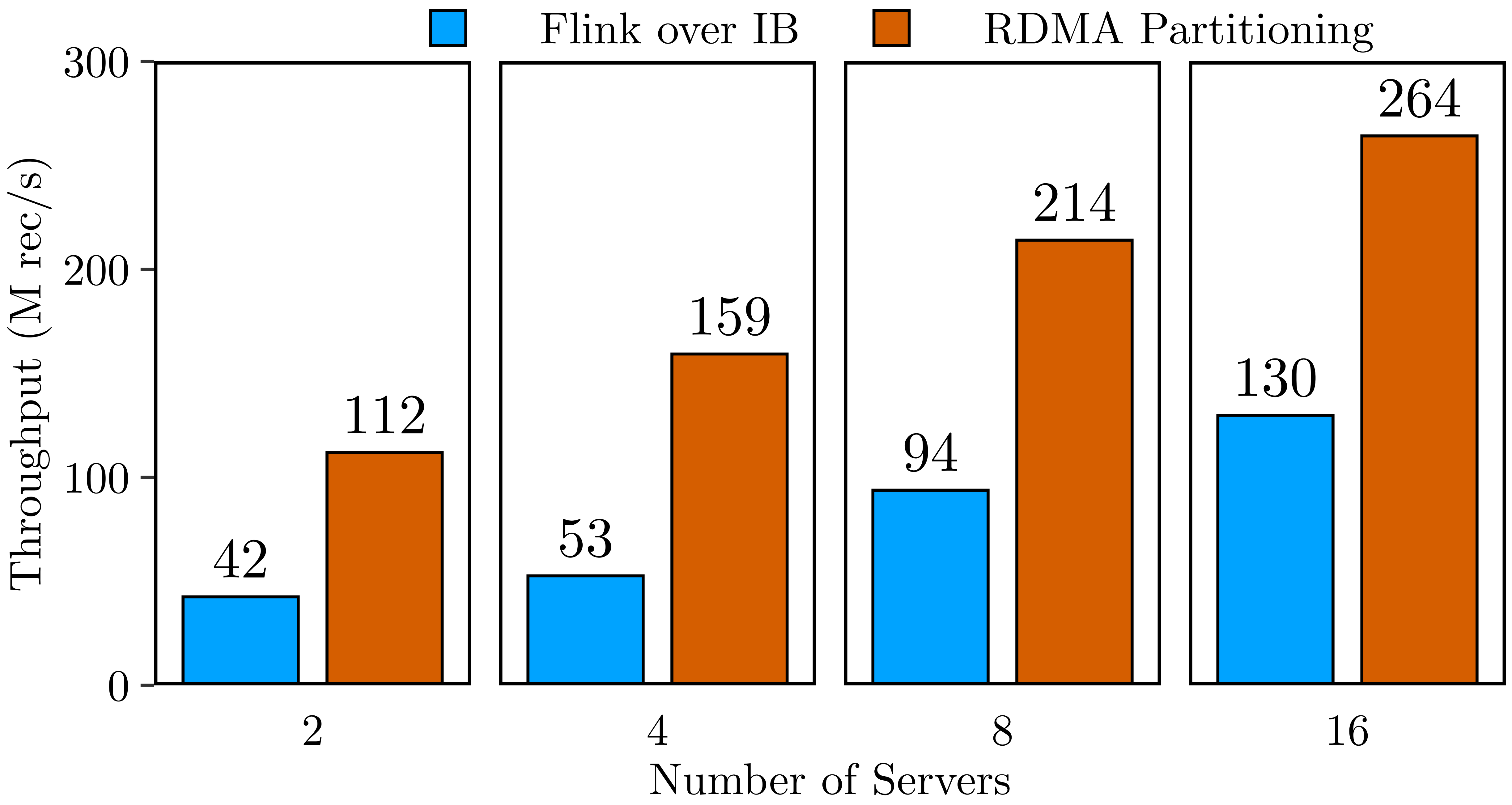
Network

High-speed Networking
Close to memory bandwidth
Faster than 10Gbps Ethernet



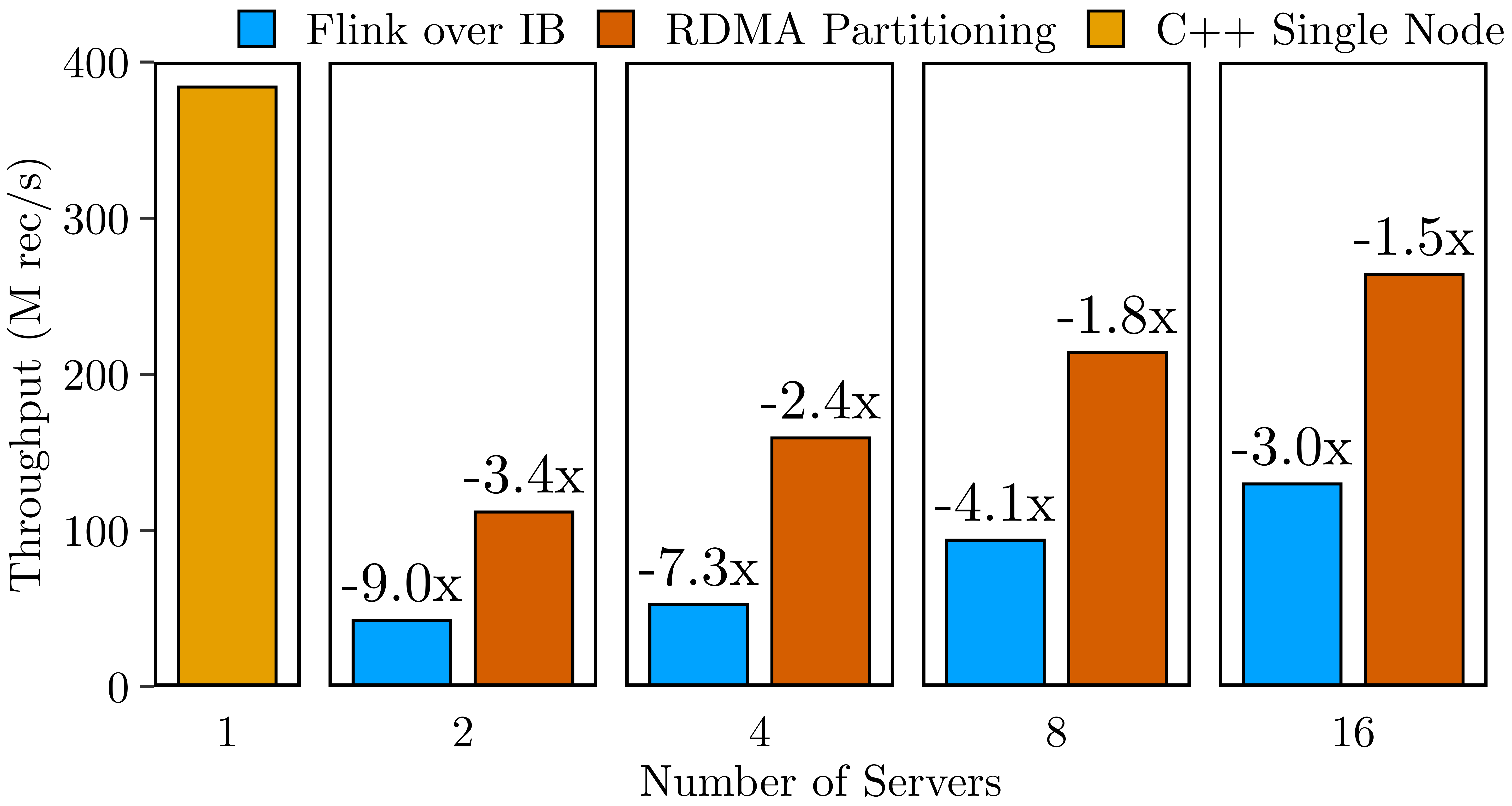
Stream Processing with high-speed network





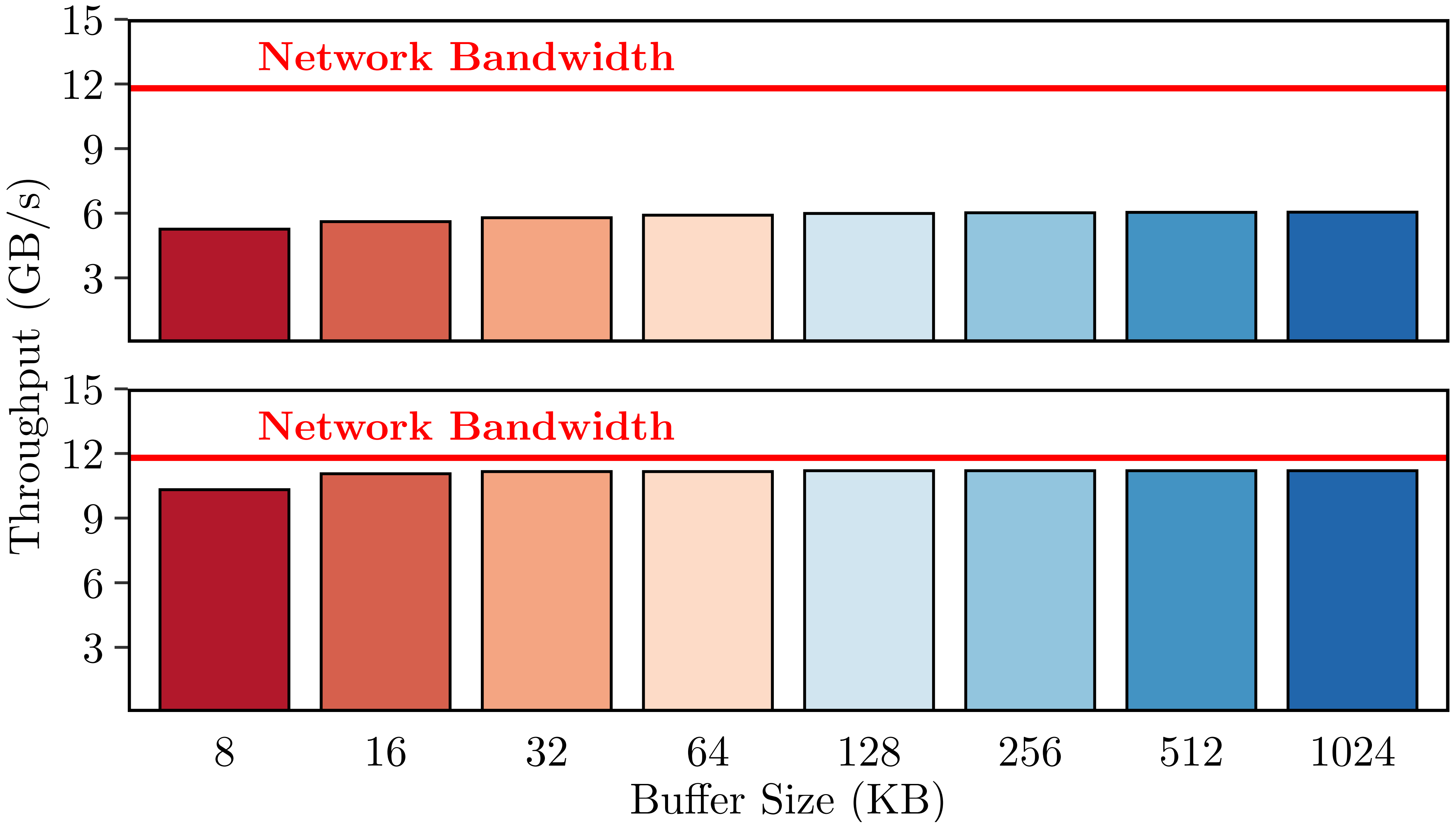
Intel Xeon Gold 5115 @ 2.4 Ghz 10-cores
RAM: 96GB
RNIC: Mellanox Connect-X4 EDR 100Gbps

Stream Processing with high-speed network



Simply using a high-speed network is not enough

Finding the bottleneck

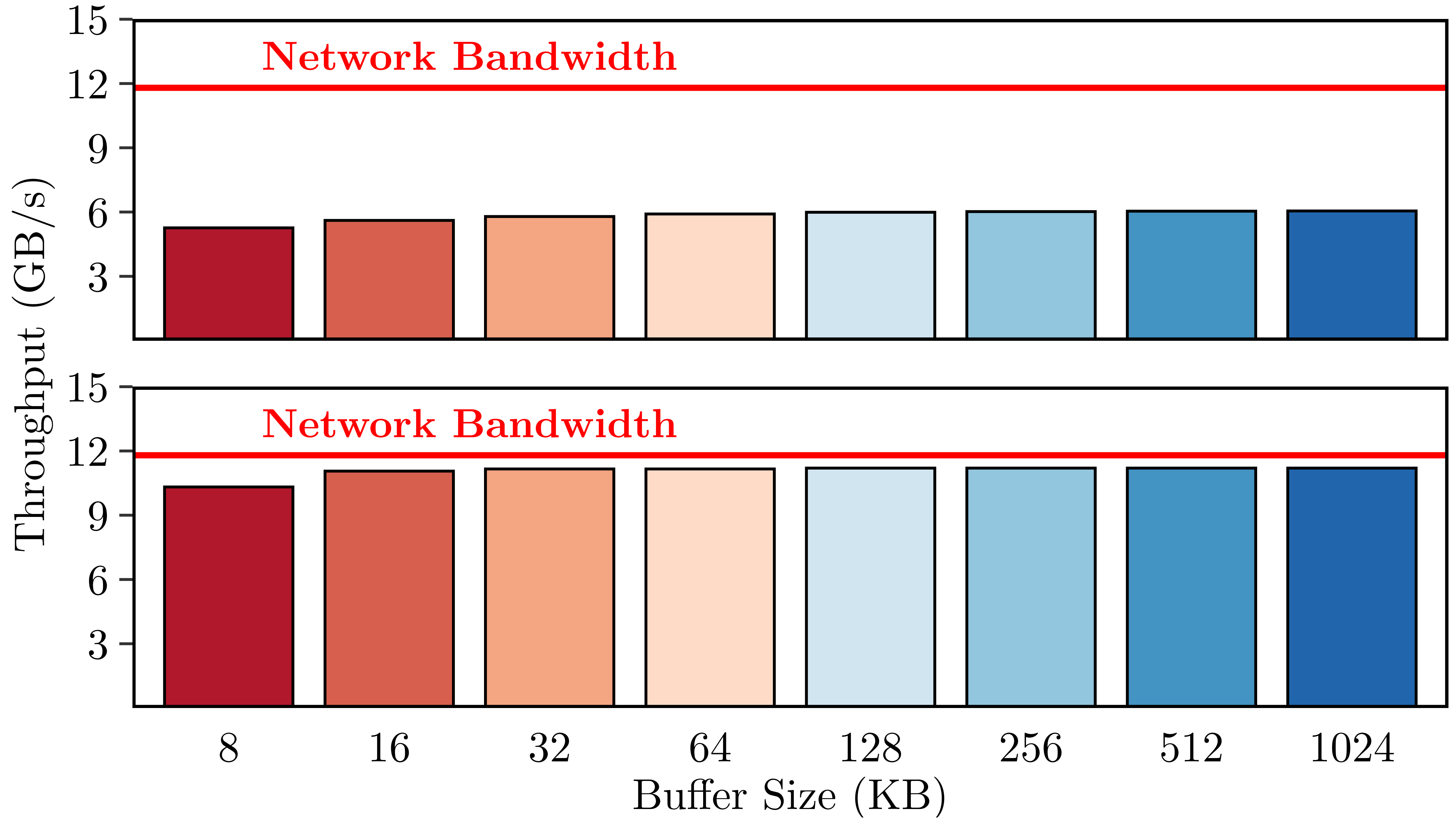


Partitioning
Servers=2
Threads=10
Partitions=100

No Partitioning
Servers=2
Thread=10

Data partitioning is a bottleneck also on two nodes

Finding the bottleneck



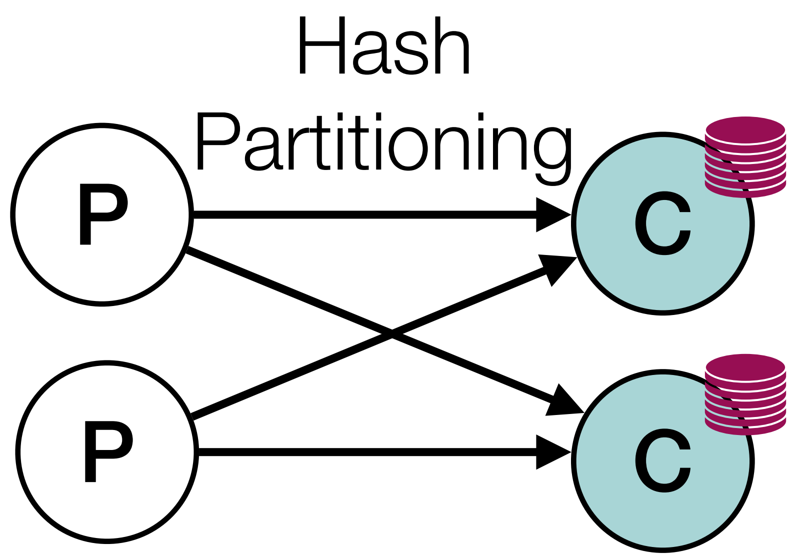
Partitioning
Servers=2
Threads=10
Partitions=100

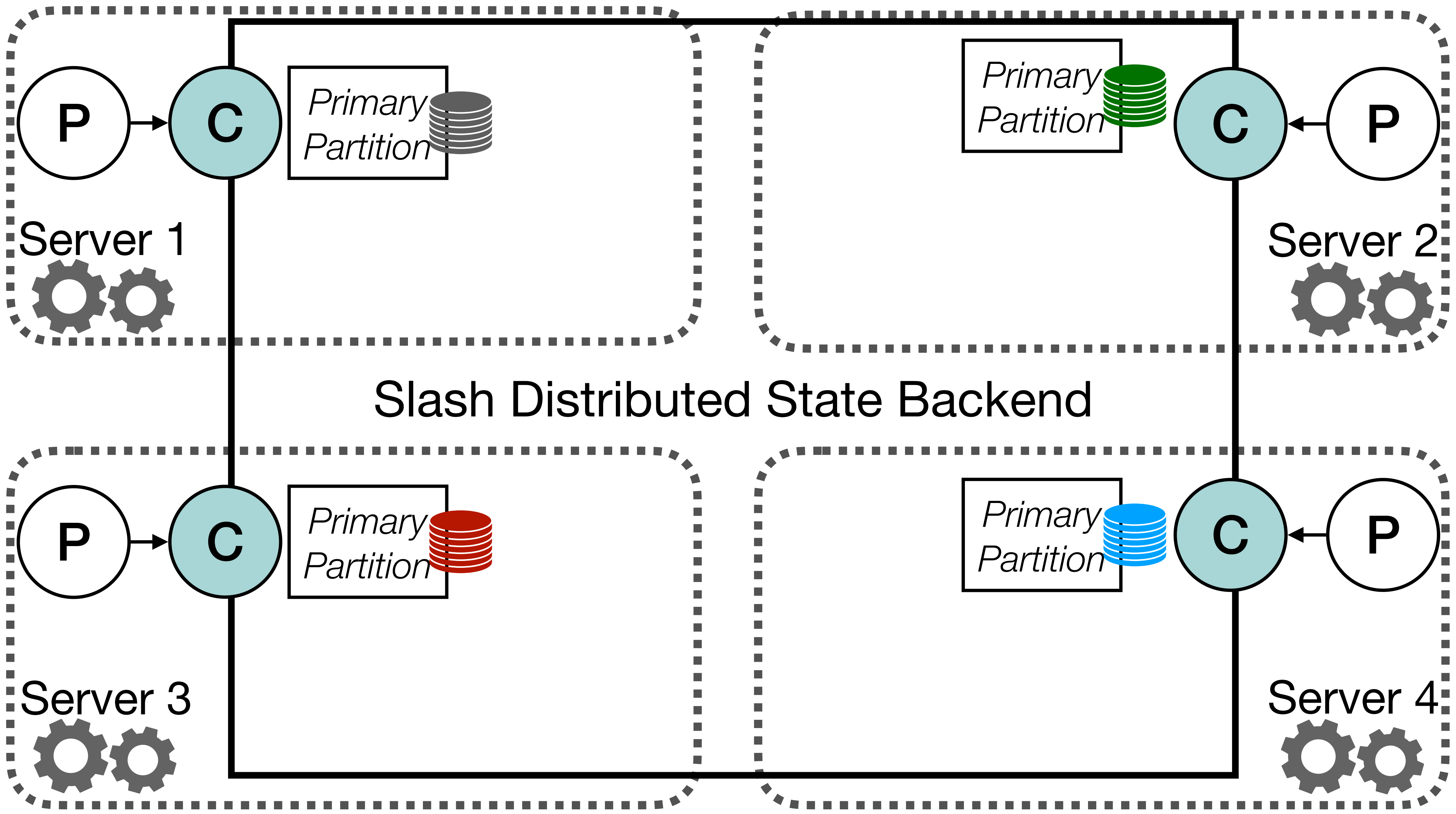
No Partitioning
Servers=2
Thread=10

Data partitioning is a <u>bottleneck already using two nodes</u>

Late Merge and Global Merge <u>using distributed memory with RDMA</u>

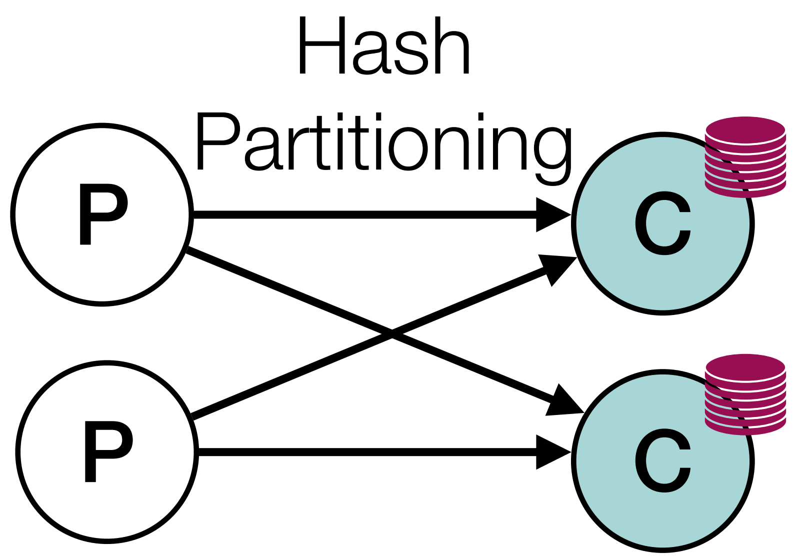
Slash: network-conscious SPE

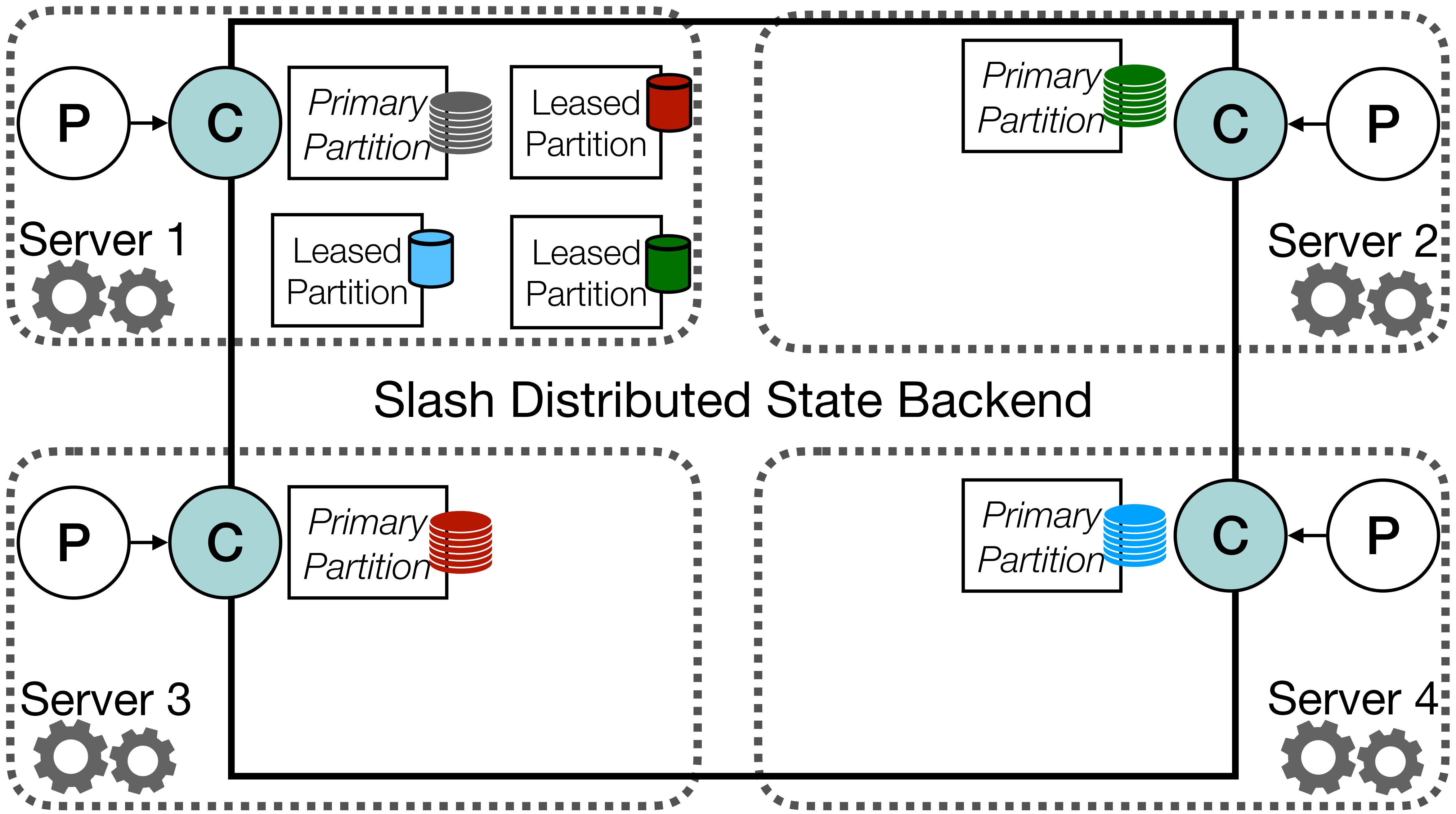




Primary Partitions: disjoint shards of operator state

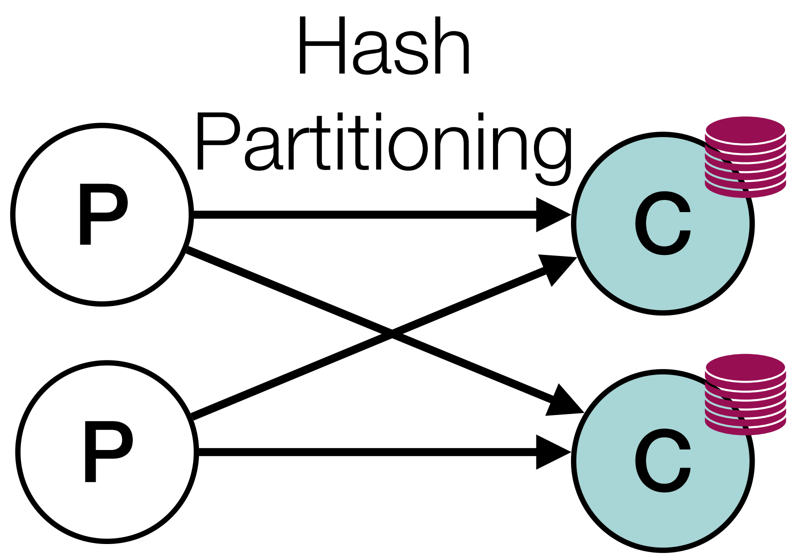
Slash: network-conscious SPE

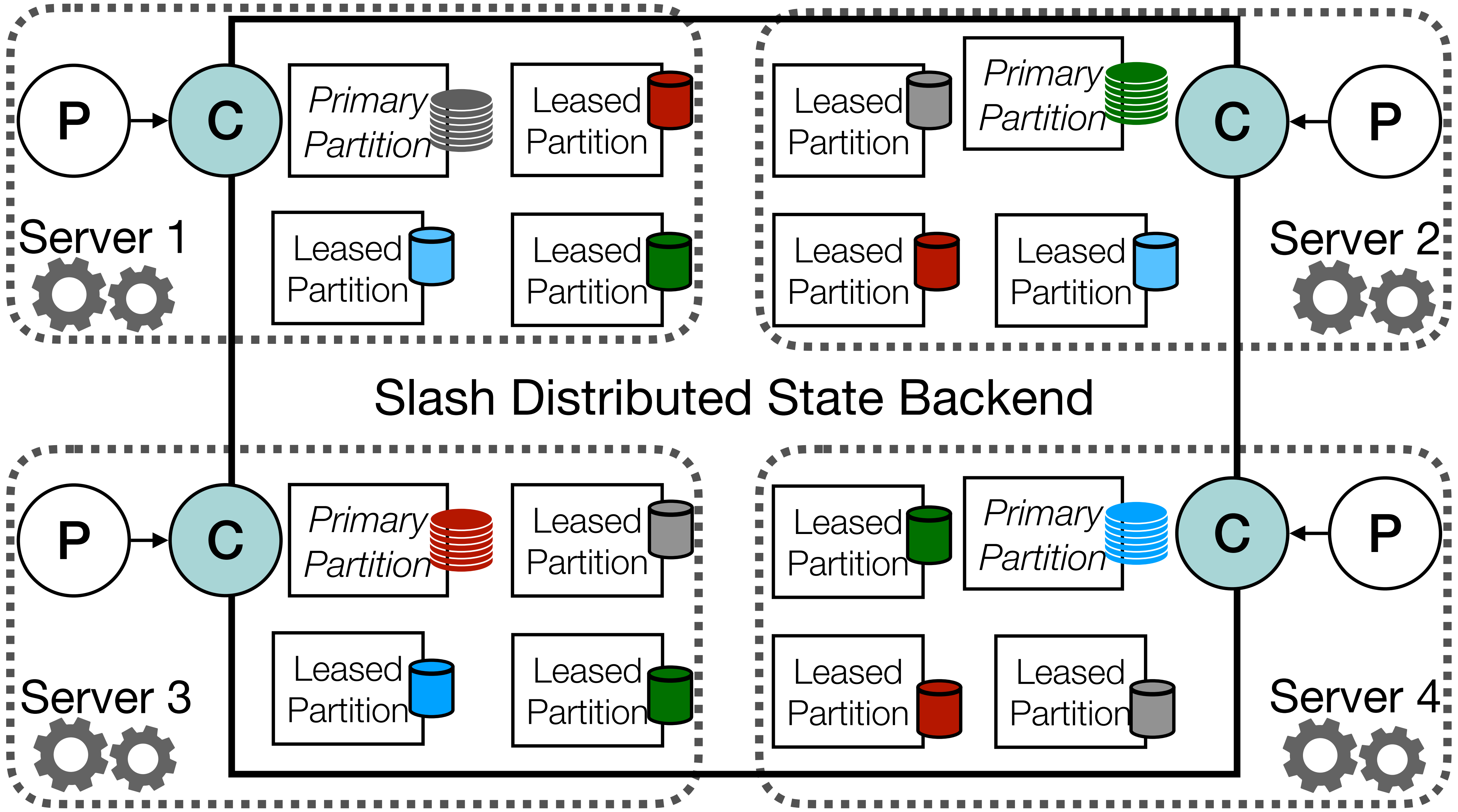




Primary Partitions: disjoint shards of operator state

Slash: network-conscious SPE

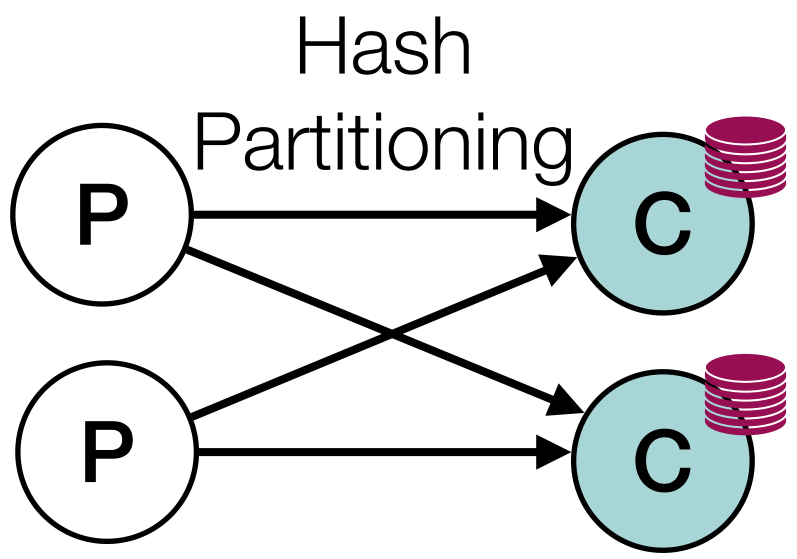


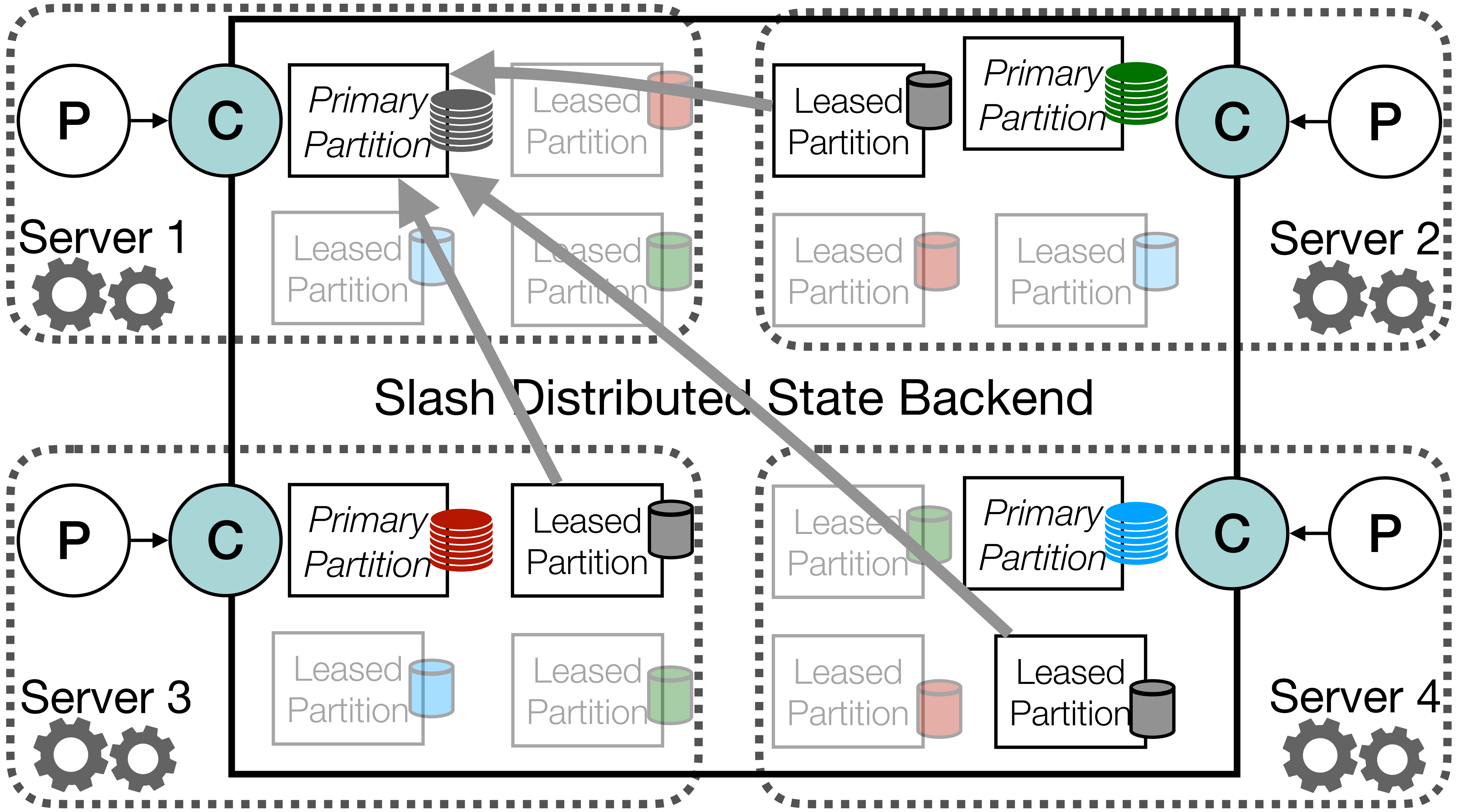


Primary Partitions: disjoint shards of operator state

Replace partitioning with eager computation of partial states followed by lazy merge

Slash: network-conscious SPE



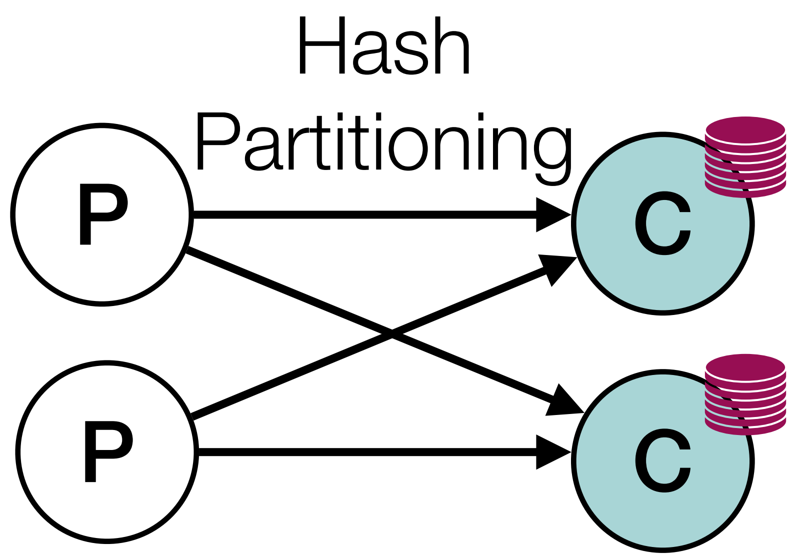


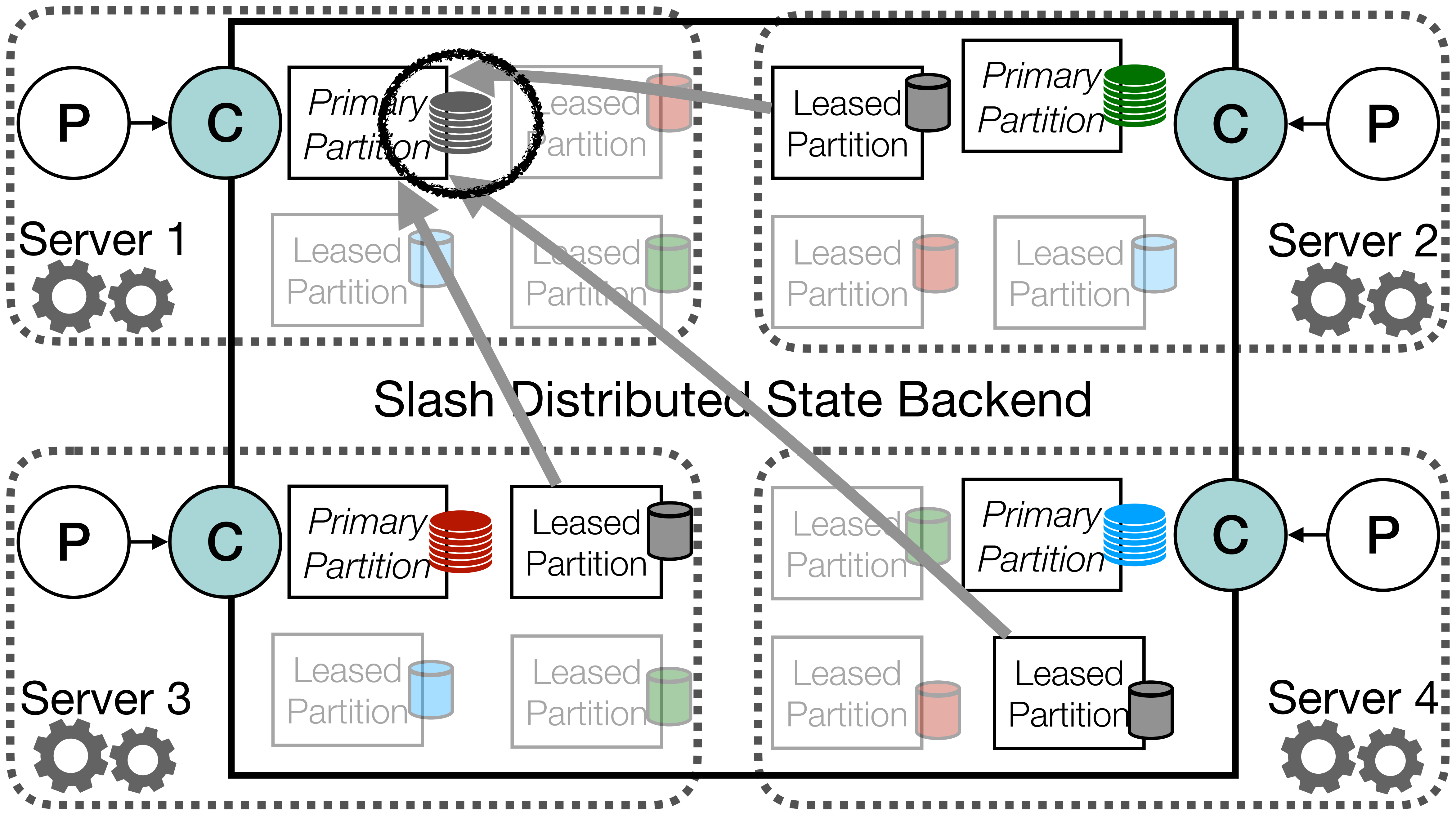
Primary Partitions: disjoint shards of operator state

Epoch-based synchro-nisation: to merge leased and primary partitions

Replace partitioning with eager computation of partial states followed by lazy merge

Slash: network-conscious SPE





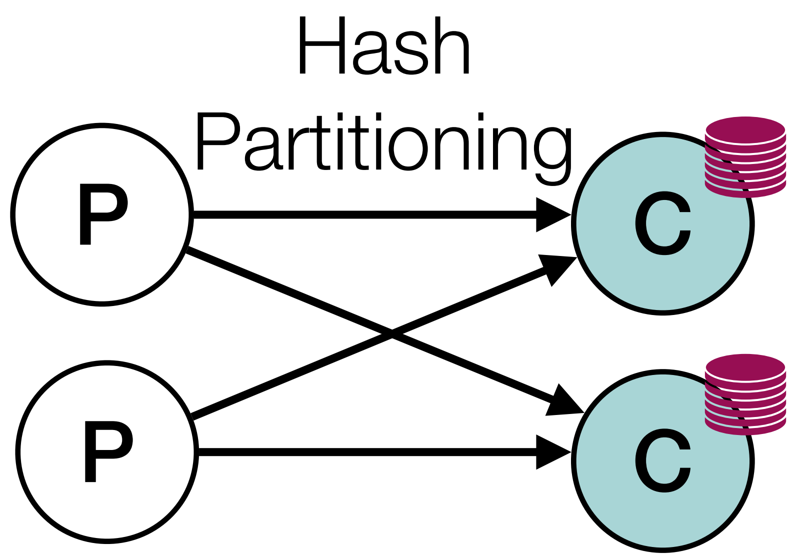
Primary Partitions: disjoint shards of operator state

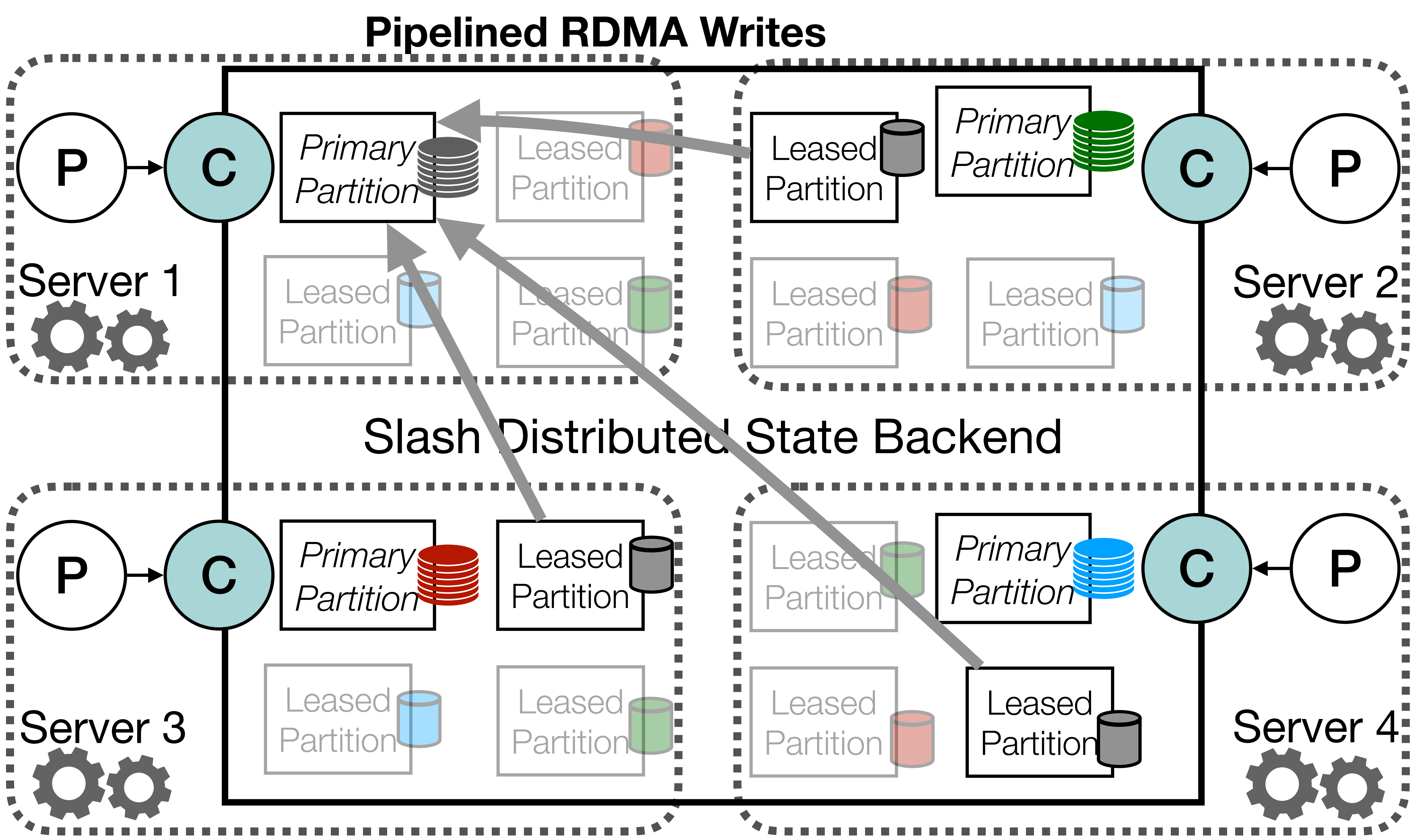
Epoch-based synchronisation: to merge leased and primary partitions

Conflict-free Replicated Data Types: to solve merge conflicts

Replace partitioning with eager computation of partial states followed by lazy merge

Slash: network-conscious SPE





Primary Partitions: disjoint shards of operator state

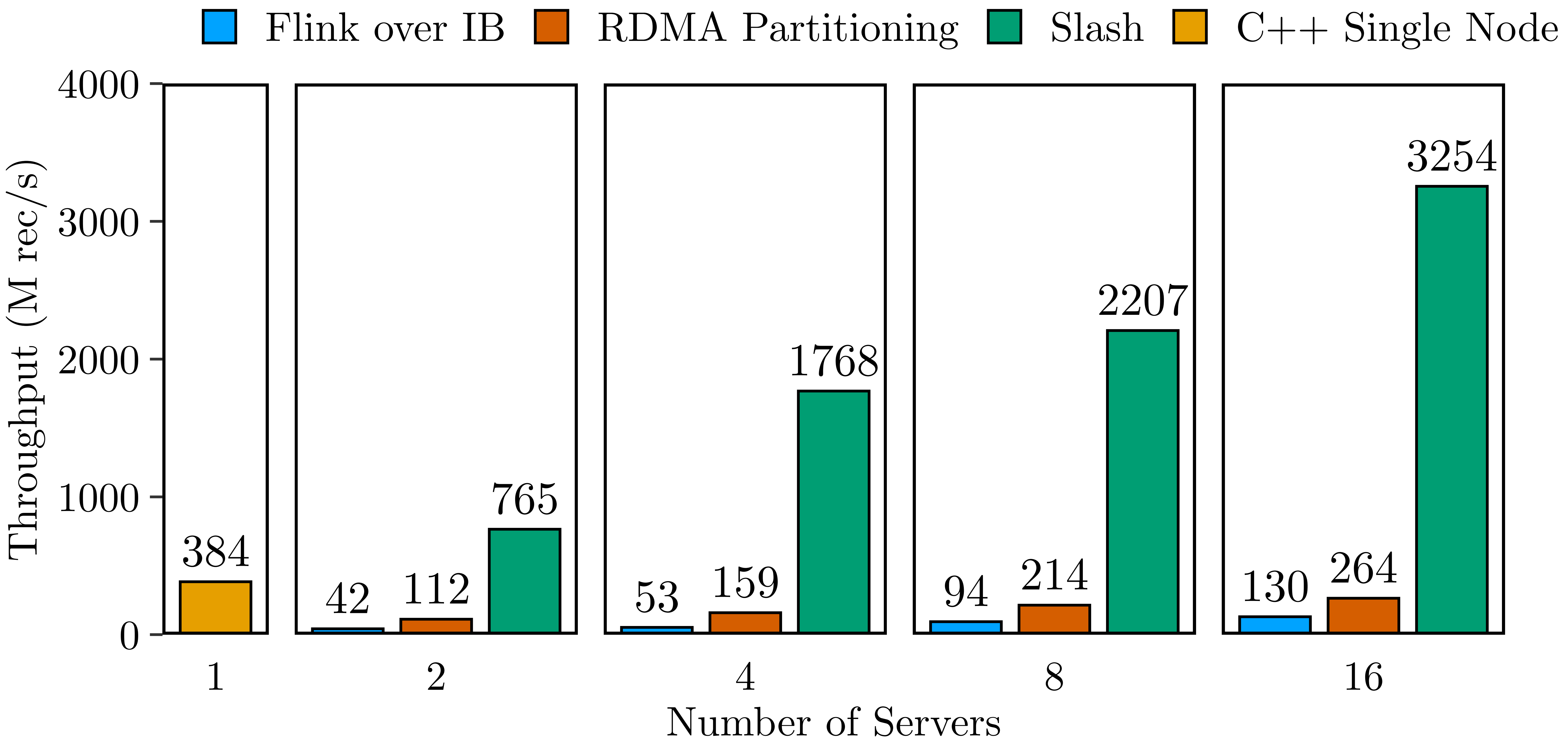
Epoch-based synchro-nisation: to merge leased and primary partitions

Conflict-free Replicated Data Types: to solve merge conflicts

Pipelined RDMA Writes: to transfer state chunks asynchronously

Replace partitioning with eager computation of partial states followed by lazy merge

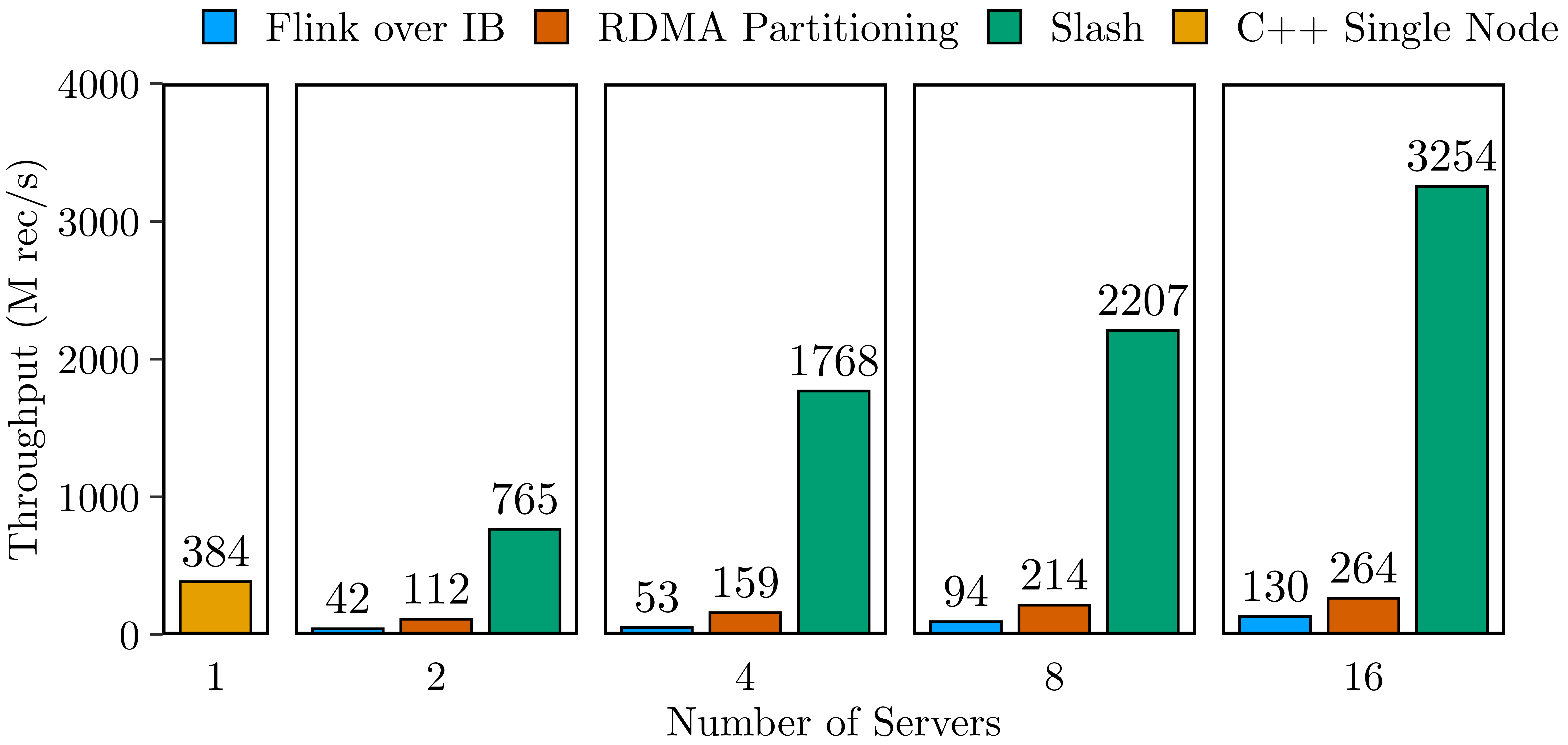
Performance of Slash



12x throughput improvement using 16 nodes

16-node Slash is 8x faster than optimised single node

Performance of Slash



RDMA baseline limited by partitioning speed (CPU-Bound)

Slash is limited by memory speed

Summary

- SPE design to **accelerate streaming workloads using RDMA** at rack-scale
- **No free lunch**: SPEs cannot efficiently scale-out using high-speed networks out-of-the-box
- Achieve **12x** throughput improvement over strongest baseline
- **Slash is memory-bound; baseline is bound by partitioning speed**

Summary

- SPE design to **accelerate streaming workloads using RDMA** at rack-scale
- **No free lunch**: SPEs cannot efficiently scale-out using high-speed networks out-of-the-box
- Achieve **12x** throughput improvement over strongest baseline
- **Slash is memory-bound; baseline is bound by partitioning speed**

NebulaStream

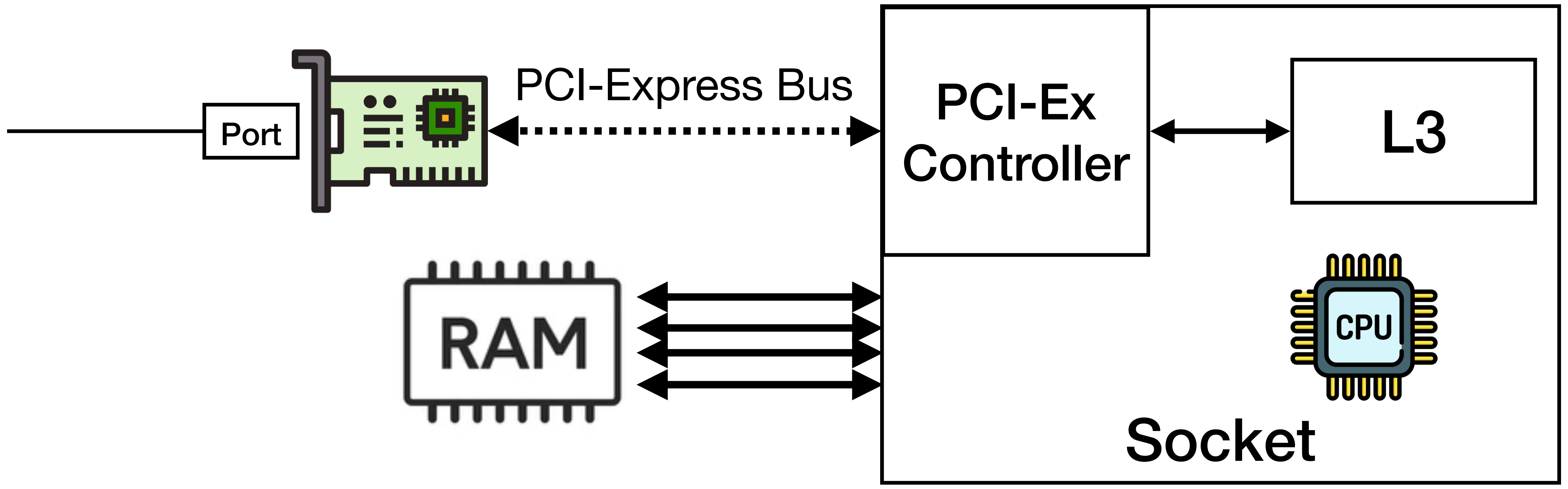
Thank you!

Backup

Slash

Remote Direct Memory Access

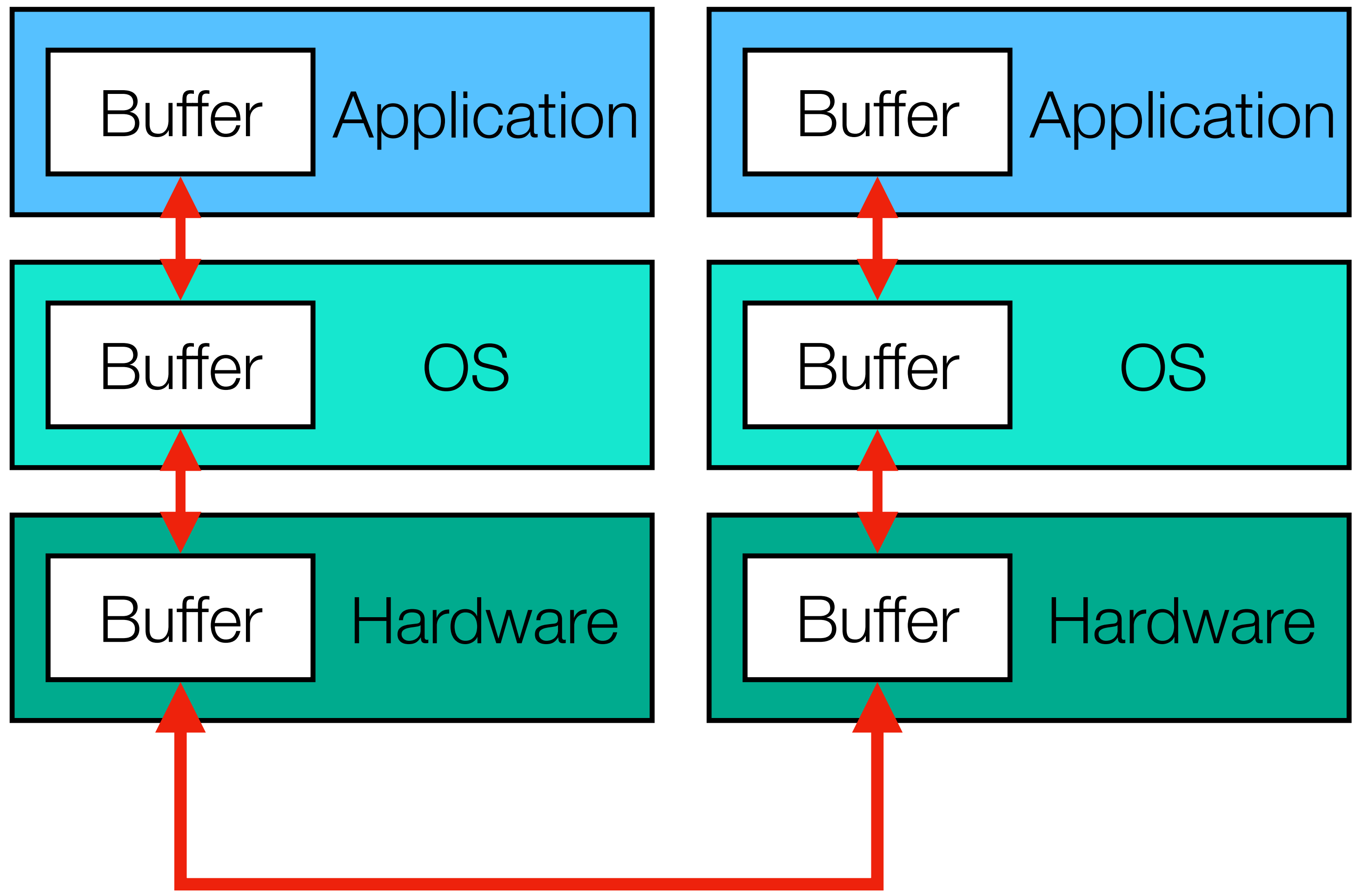
Infiniband EDR 100Gbps (12.5 GB/s)
Infiniband HDR 200 Gbps (25 GB/s)
Infiniband NDR 400 Gbps (50 GB/s)



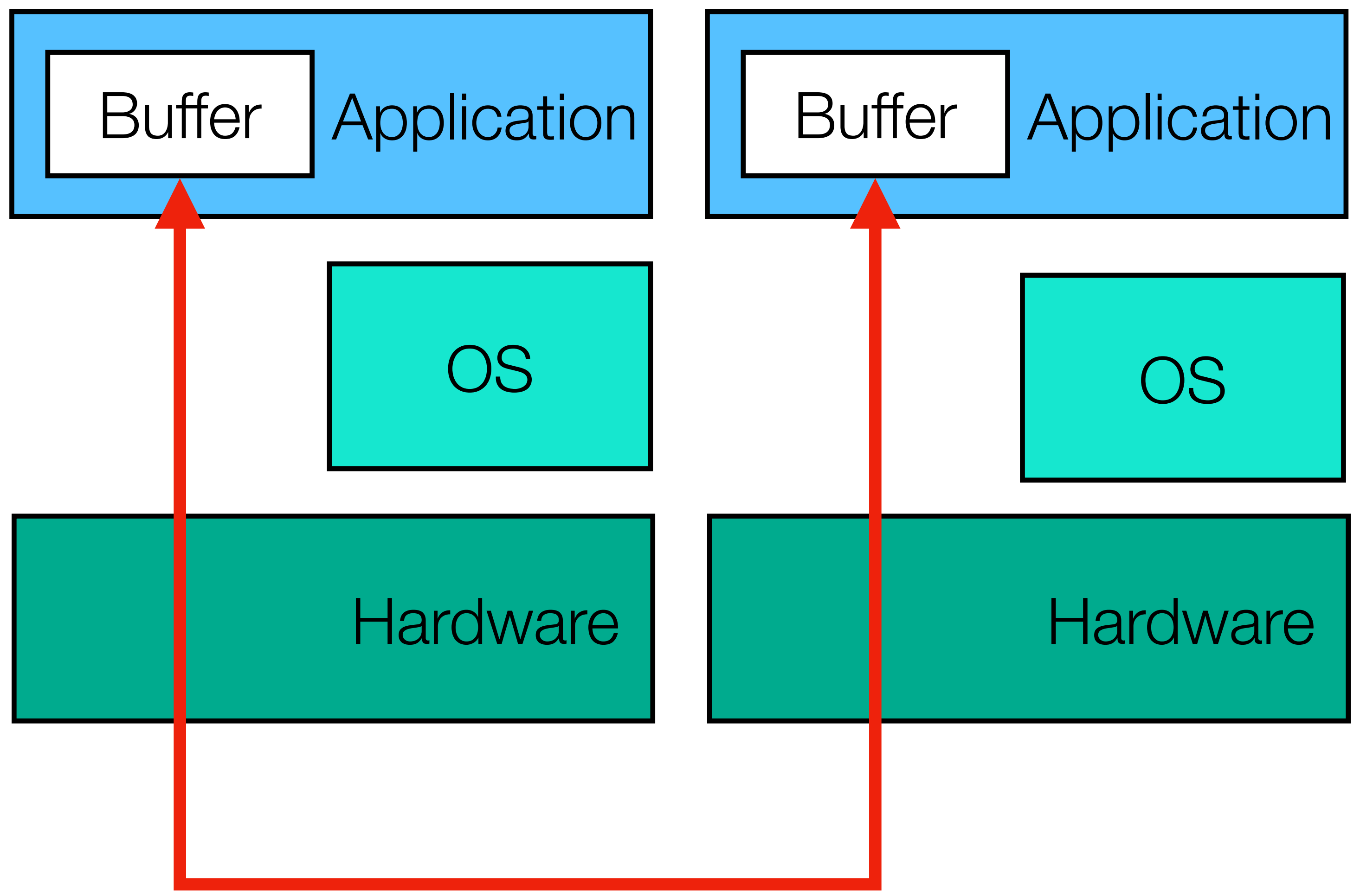
PCI-Express 3.0 Bandwidth: 984.6 MB/s per lane (16x: 15.74 GB/s)

PCI-Express 5.0 Bandwidth: 3.93 GB/s per lane in each direction (16x: 63 GB/s)

Socked-based vs. RDMA



Kernel-space Networking
Multiple Data Copies



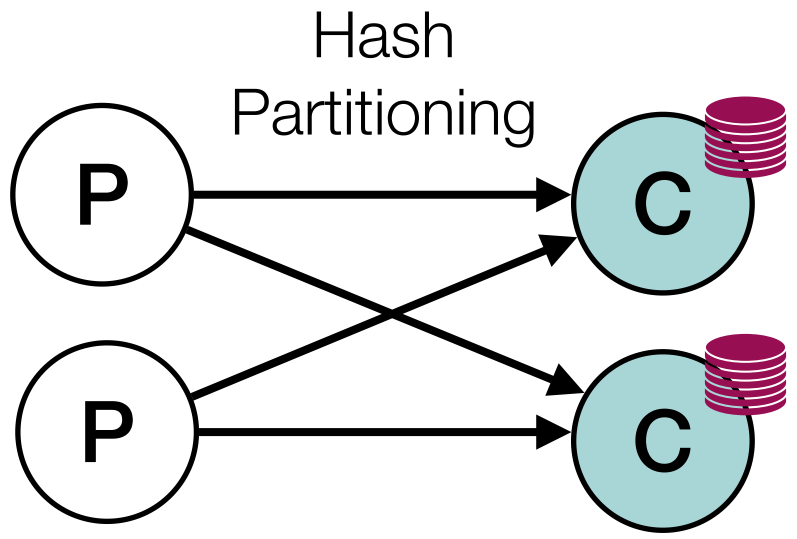
User-space Networking (low latency)

Zero-copy (low latency)

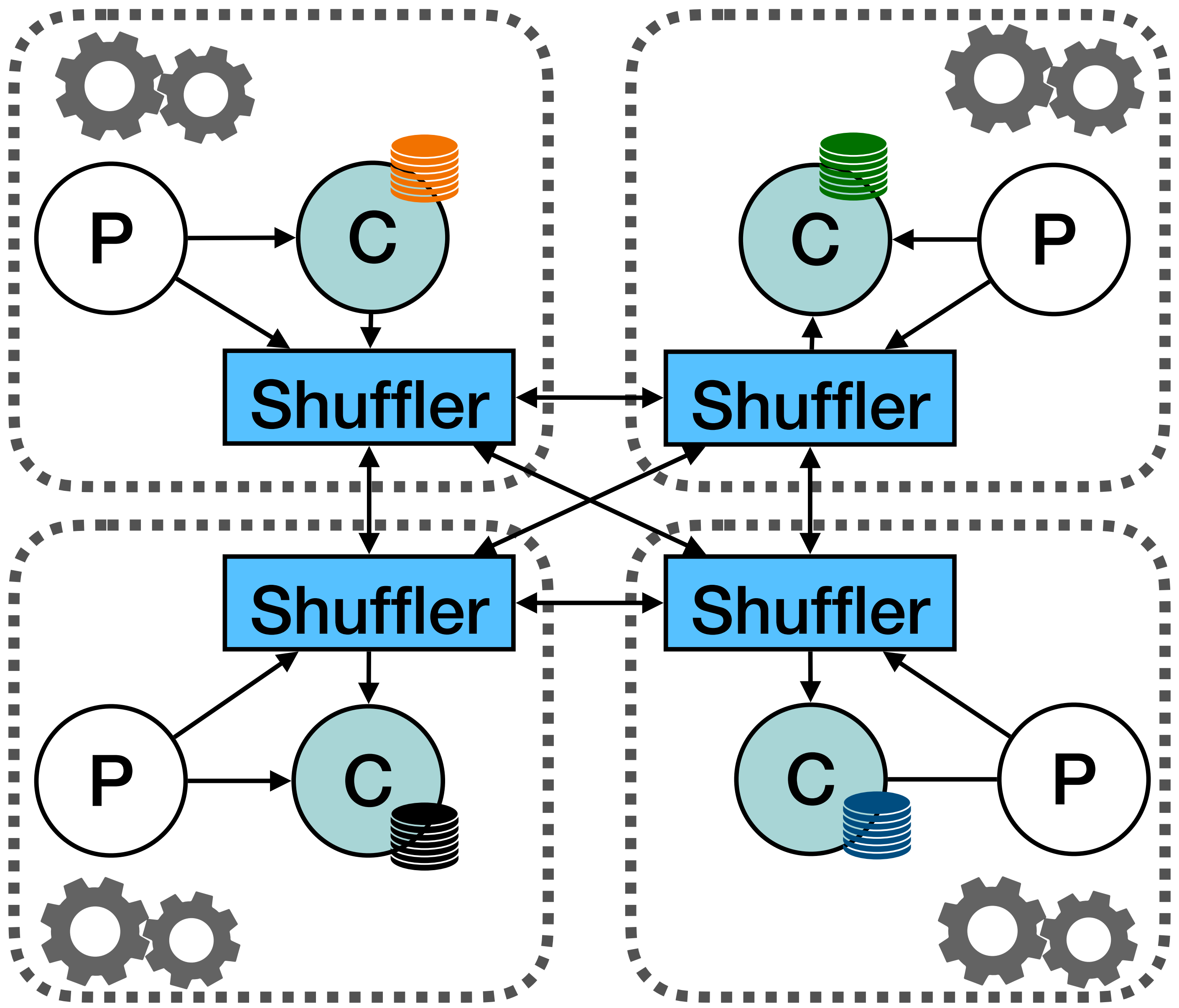
Enables distributed memory (non coherent)

Two-sided verbs: Send/Recv
One-sided verbs: Read/Write/Atomic

Distributed Streaming Query Execution



Partitioning-based Execution

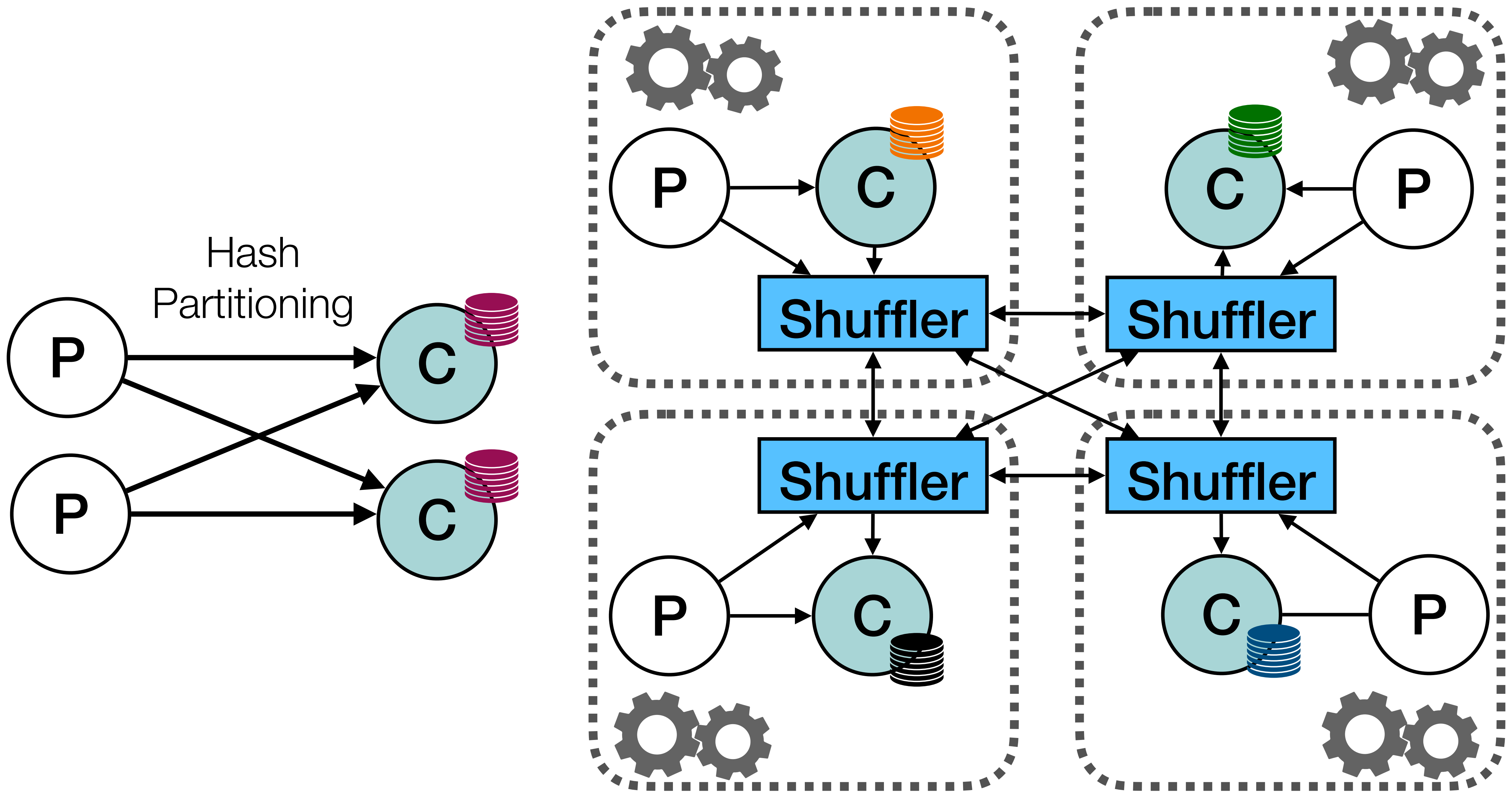


Thread-local State Partitions

Disjoint State Partitions

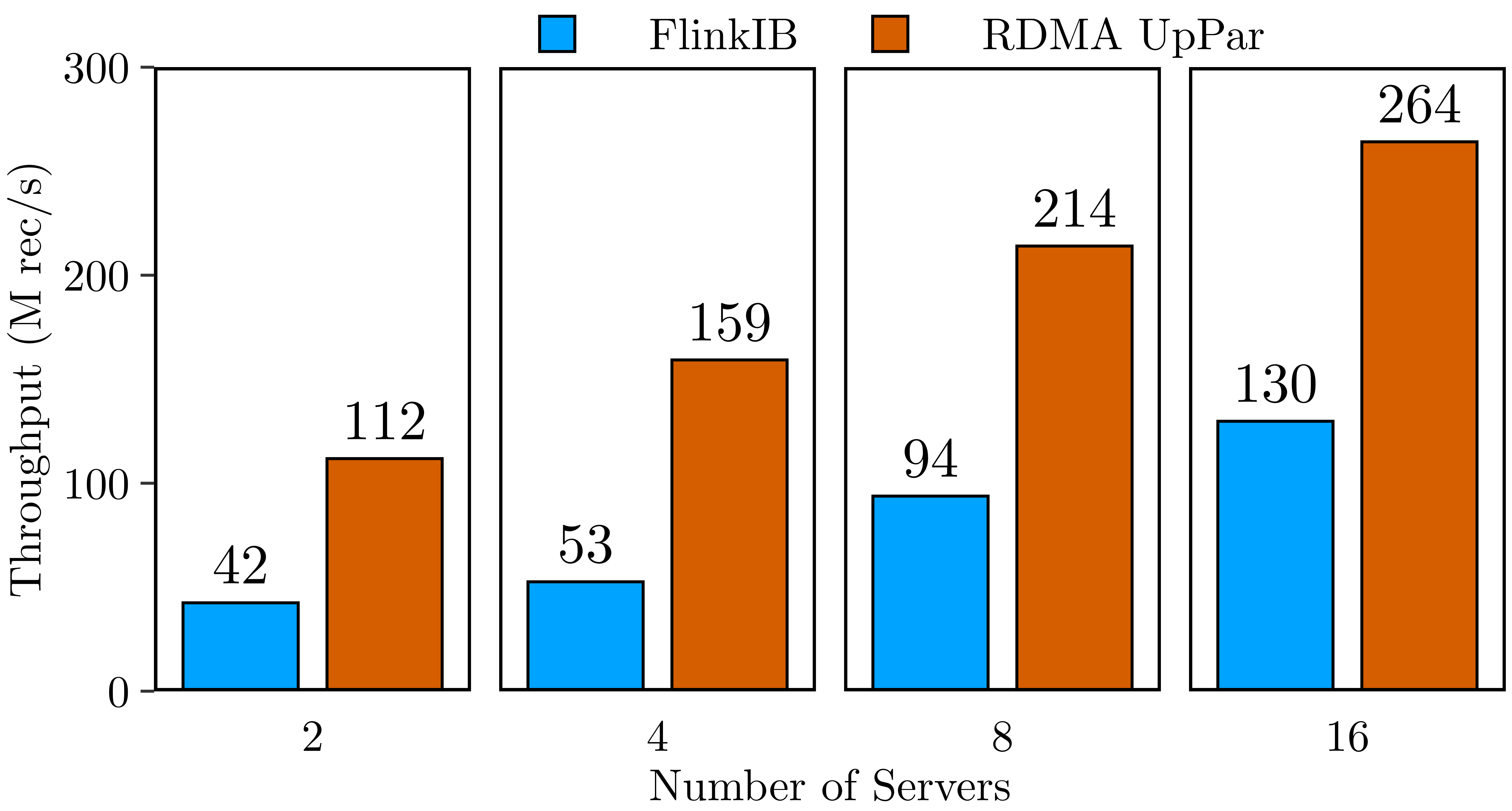
Distributed Streaming Query Execution

Partitioning-based Execution



Thread-local State Partitions

Disjoint State Partitions

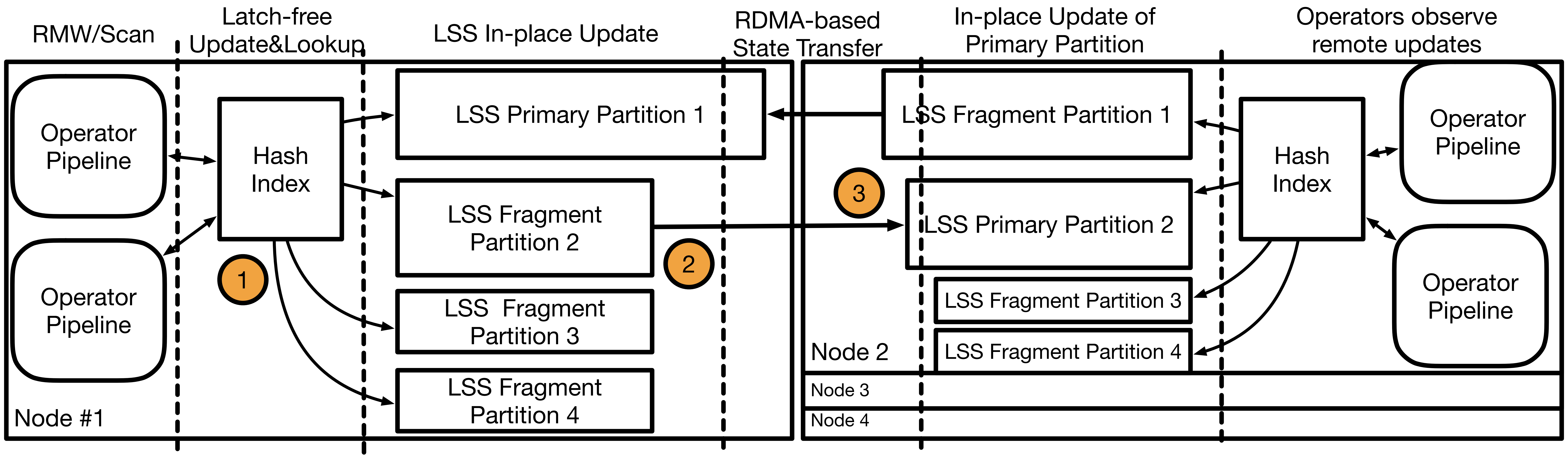


Intel Xeon Gold 5115 @ 2.4 Ghz 10-cores
L1: 32KB L2: 10MB
L3: 13.75MB RAM: 96GB
NIC: Mellanox Connect-X4 EDR 100Gbps

When Slash make sense

- Cost(Partitioning) + Cost(Local Computation) > Cost(Partial Computation) + Cost(Lazy Merge)
- Keyed Aggregation or Joins (Streaming ETL)
    - Define State as a CRDT
- New operators need to use our distributed state abstraction
    - Network-hungry such as Cross-Product
    - ML Operators

Where RDMA comes into play



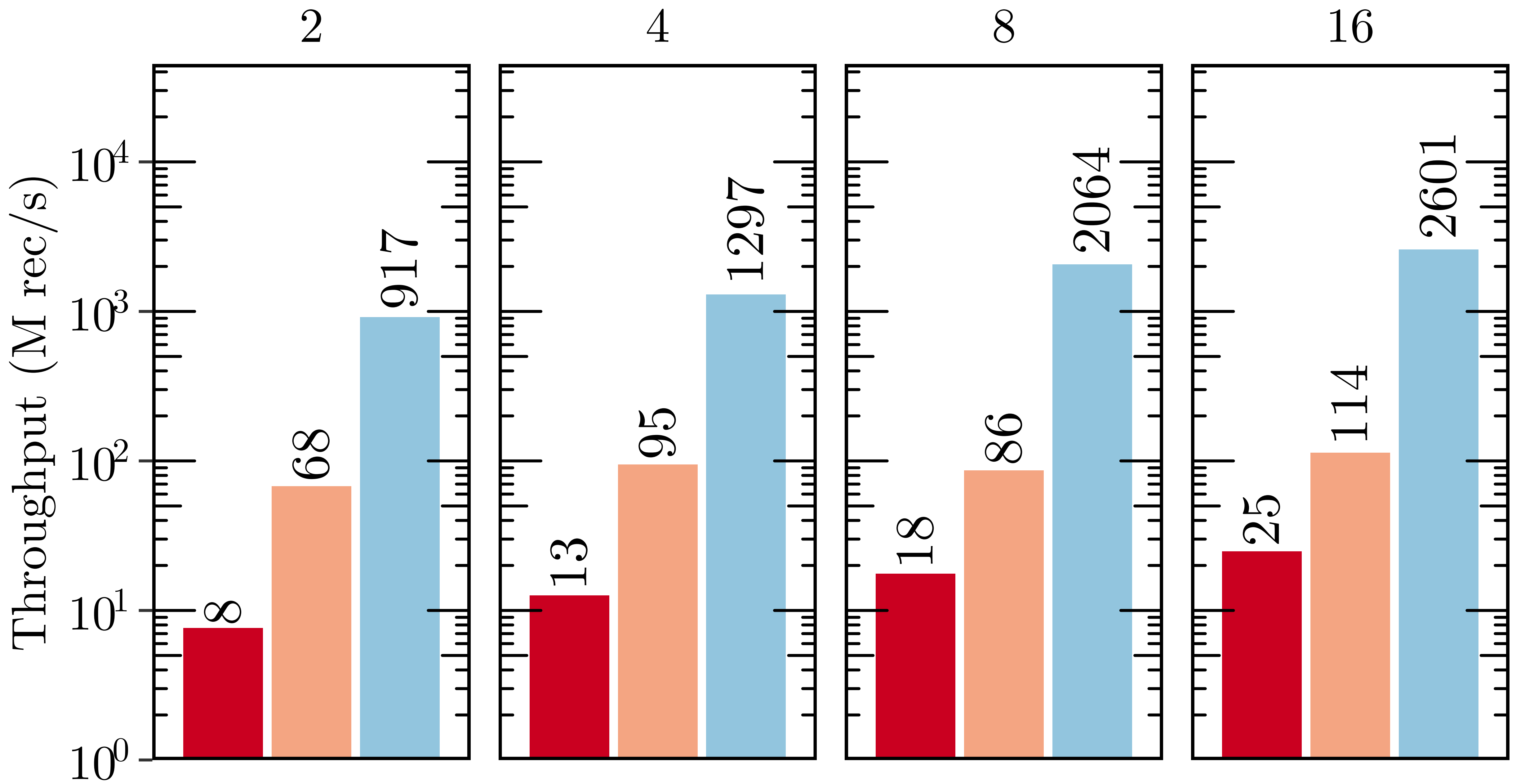
Cost of RDMA

- Mellanox (now Nvidia) Connect X-6 200Gpbs sold at about 1200$
- Azure RDMA-capable H/HB instances: 800/1600$/mo
- AWS has Elastic Fabric Adapter (Send/Recv): 2180$/mo (m6in.32xlarge)

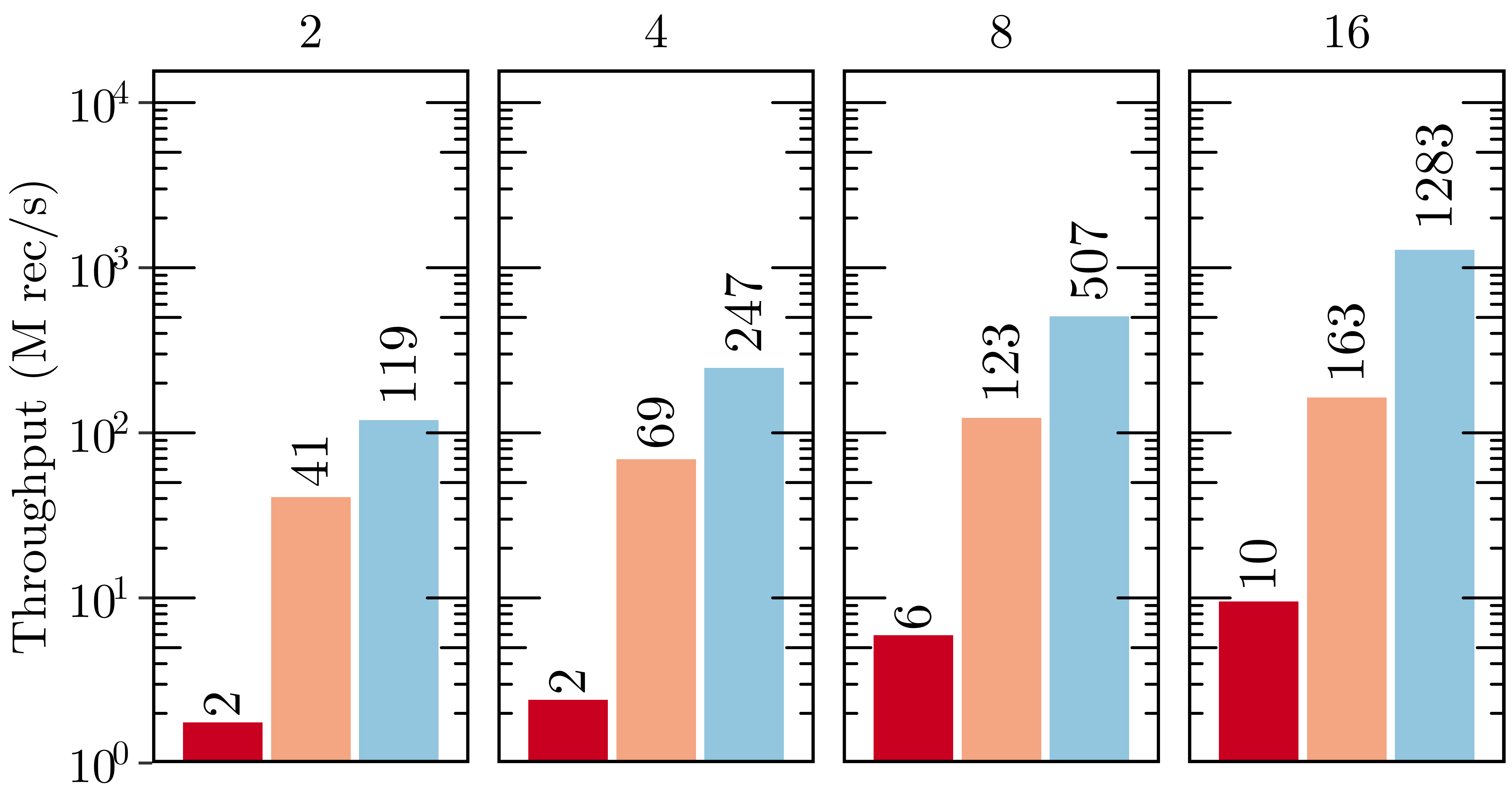
Large SPE deployments

- Alibaba: 1.5M CPU for Flink (35000 jobs)
- Netflix: 14k nodes with 22k CPU (100s jobs)

Slash Performance

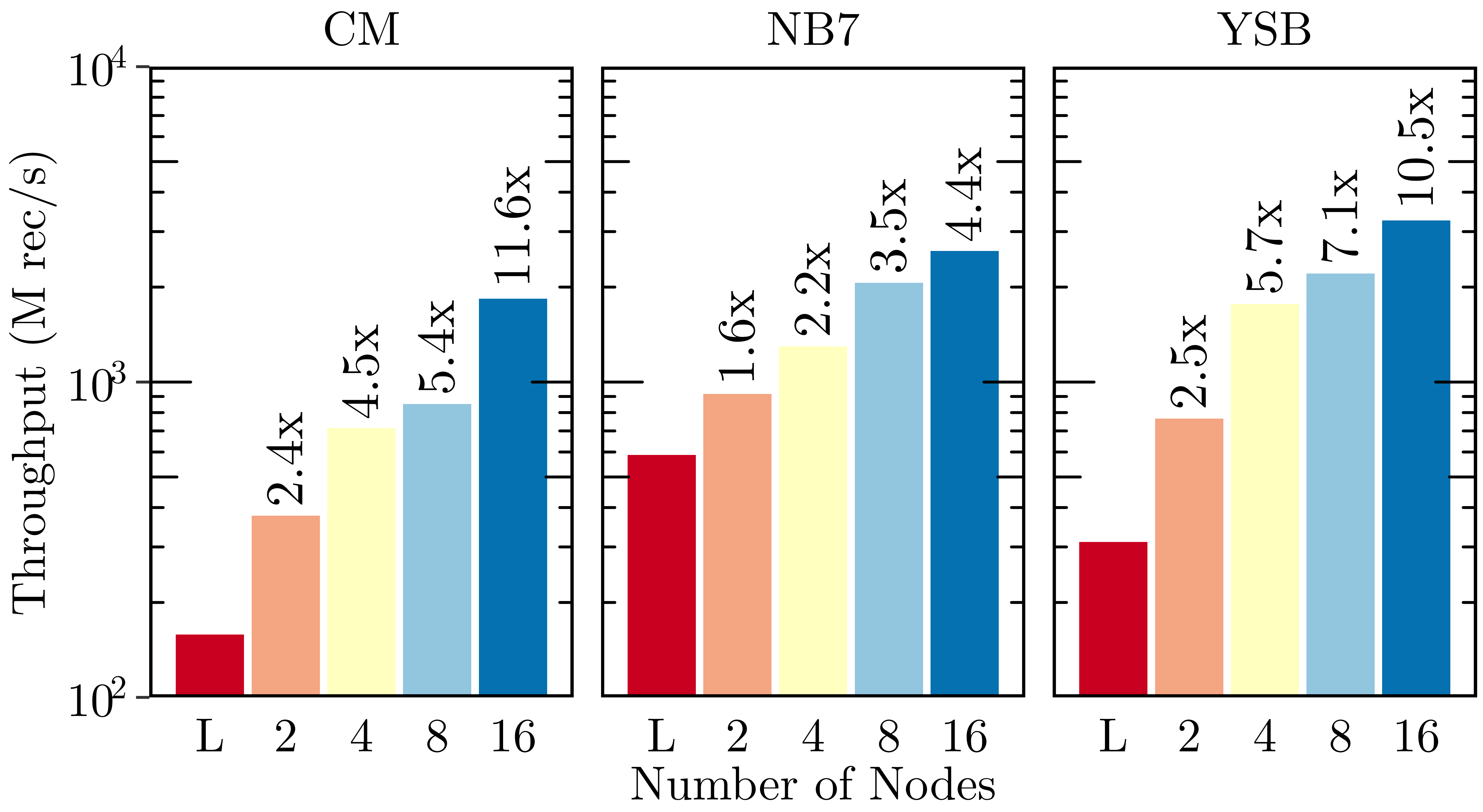


Nexmark Query 7

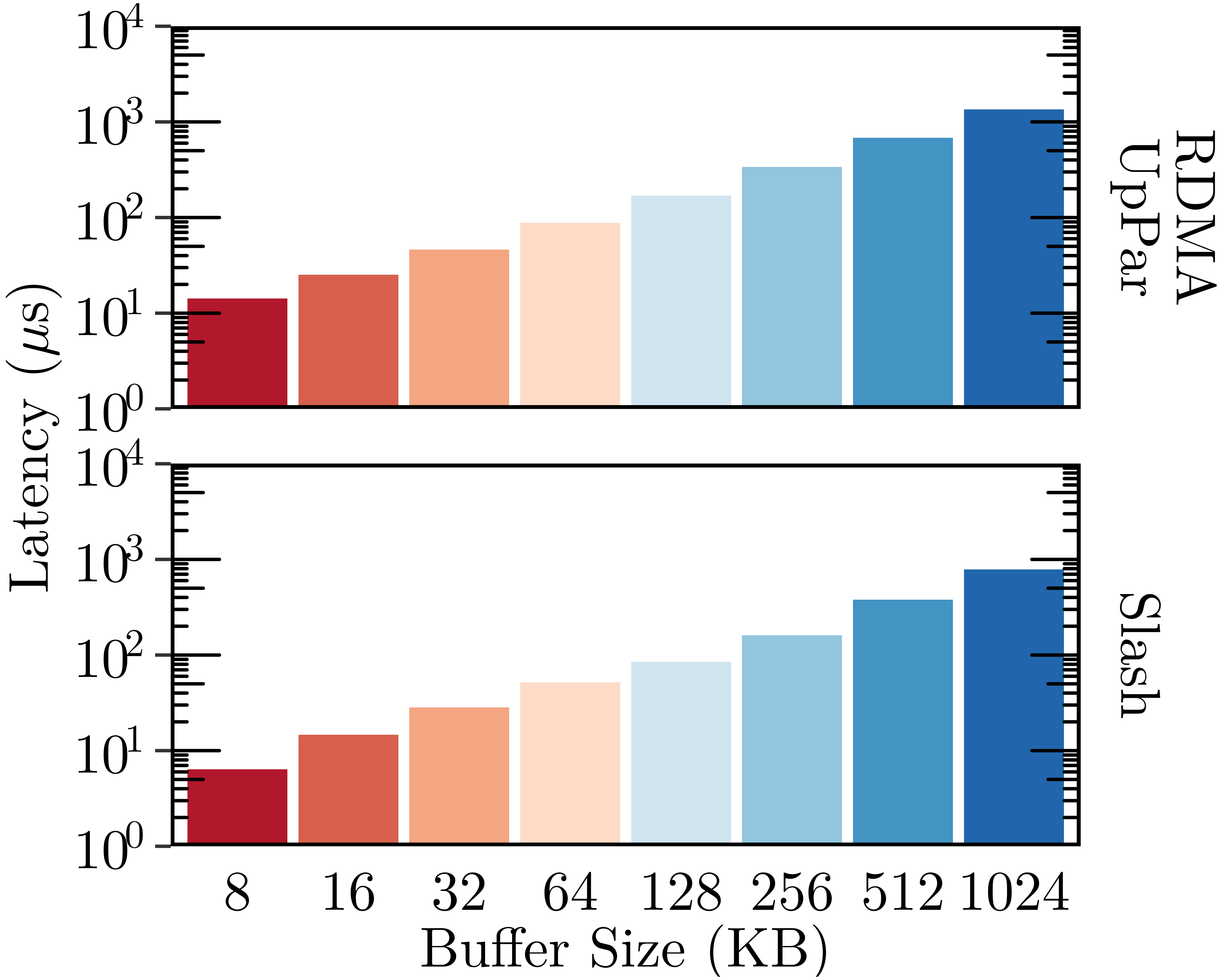


Nexmark Query 8

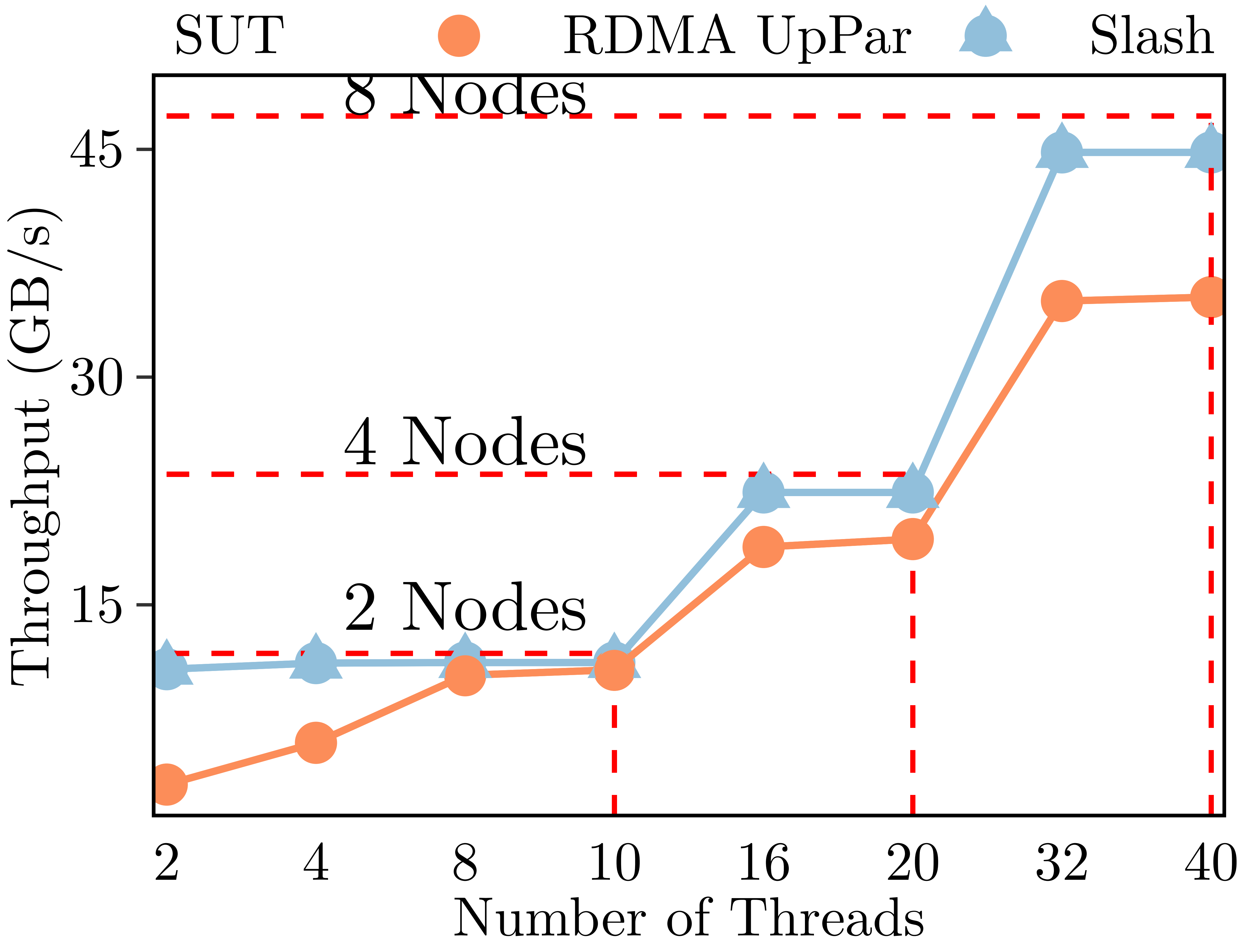
Slash Microbenchmarks: COST



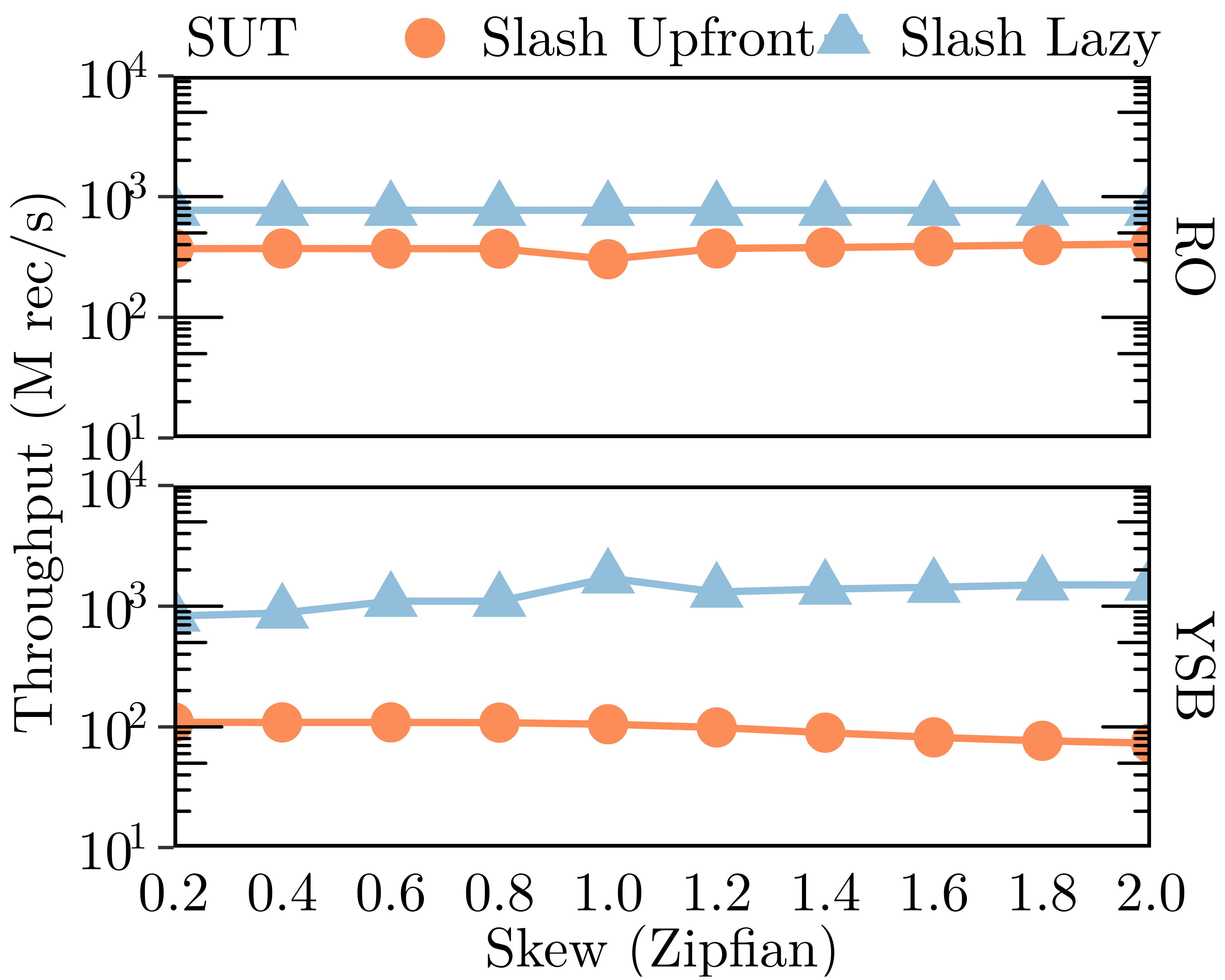
Slash Microbenchmarks: Latency



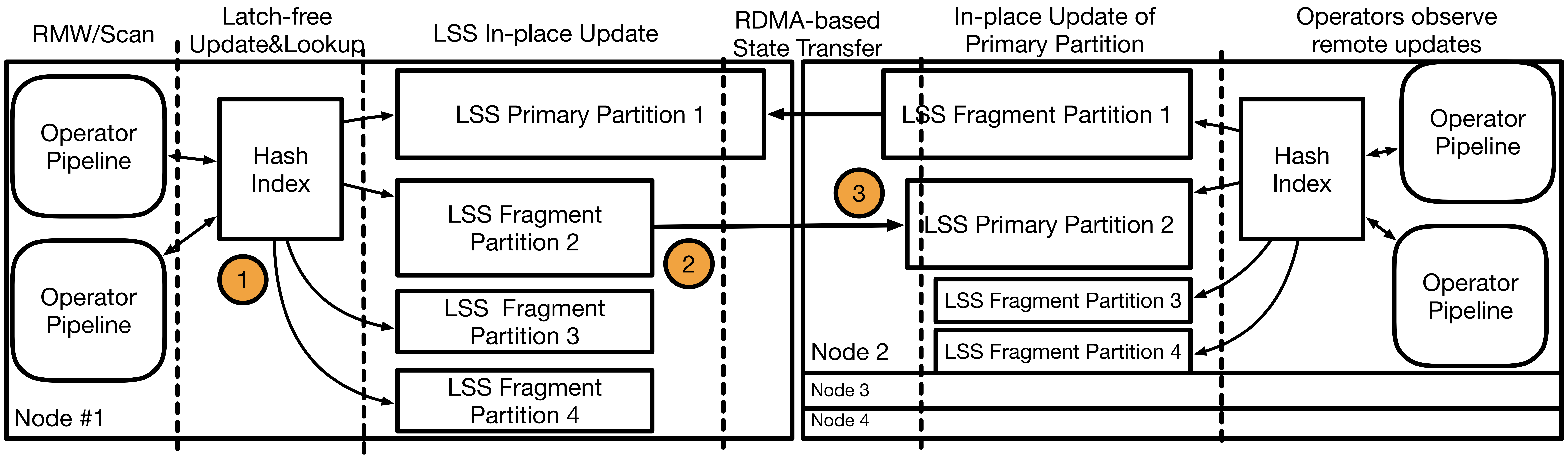
Slash Microbenchmarks: Node Parallelism



Slash Microbenchmarks: Skew

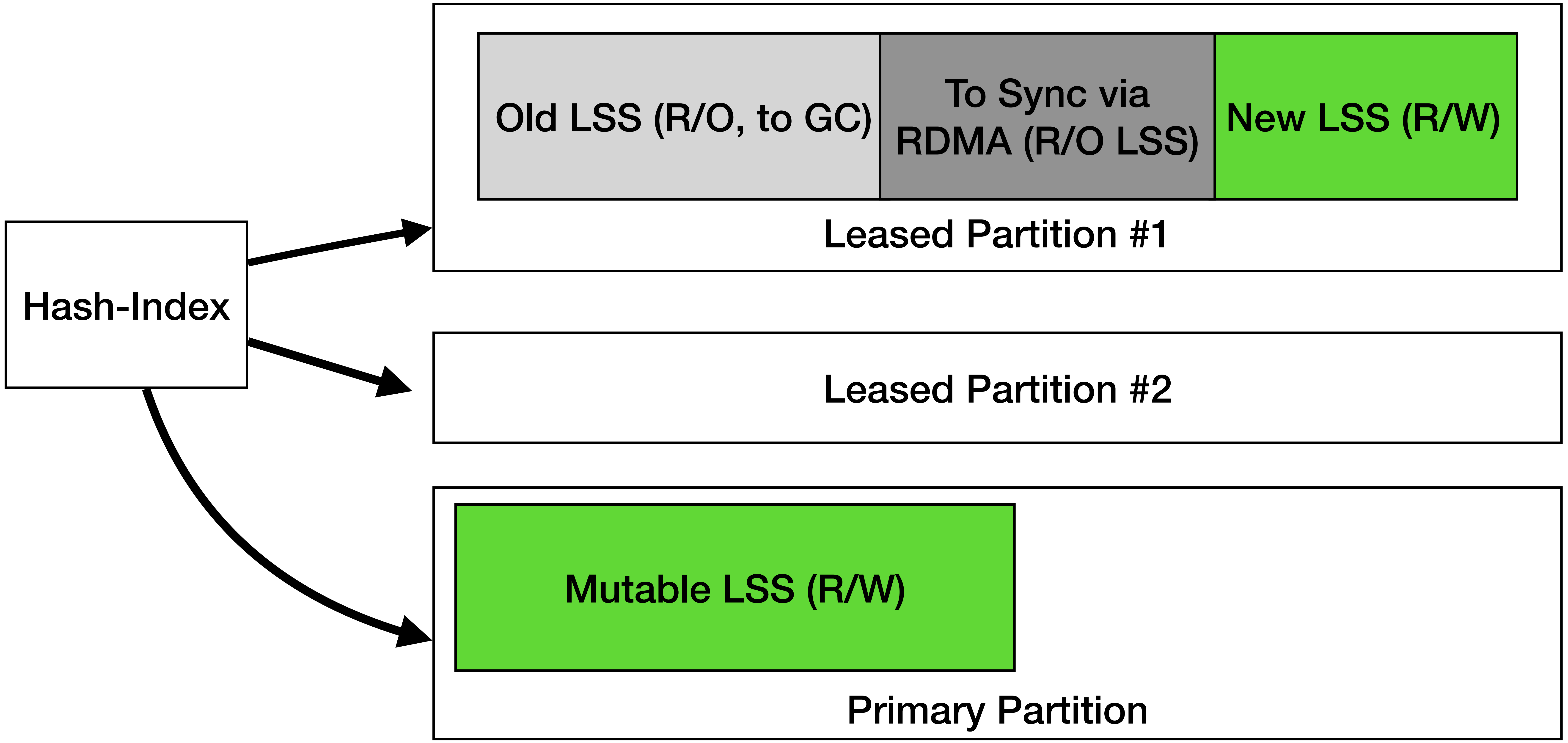


Slash State Backend Internals



Fragment Partition (thesis)
is the Leased Partition (talk)

Anatomy of Slash Partitions



Conflict Free Replicated Data Types

- Inspired by AnnaKVS and FASTER design
- Define a “merge function” *f(k, v1, v2)* to merge v1 and v2 within the same window
- Windowed aggregation:
  - Average, Sum, Count
- Windowed Join:
  - List of segments

RDMA Data Channel details

- Pipelined RDMA Writes of data chunks arranged in a circular queue
  - Keep the RNIC well-fed with data
  - Async: too little -> low bandwidth; too much -> RNIC cache trashing
- Polling on footer
- Zero-copy
- Credit-based flow control to avoid producer overwhelm consumer

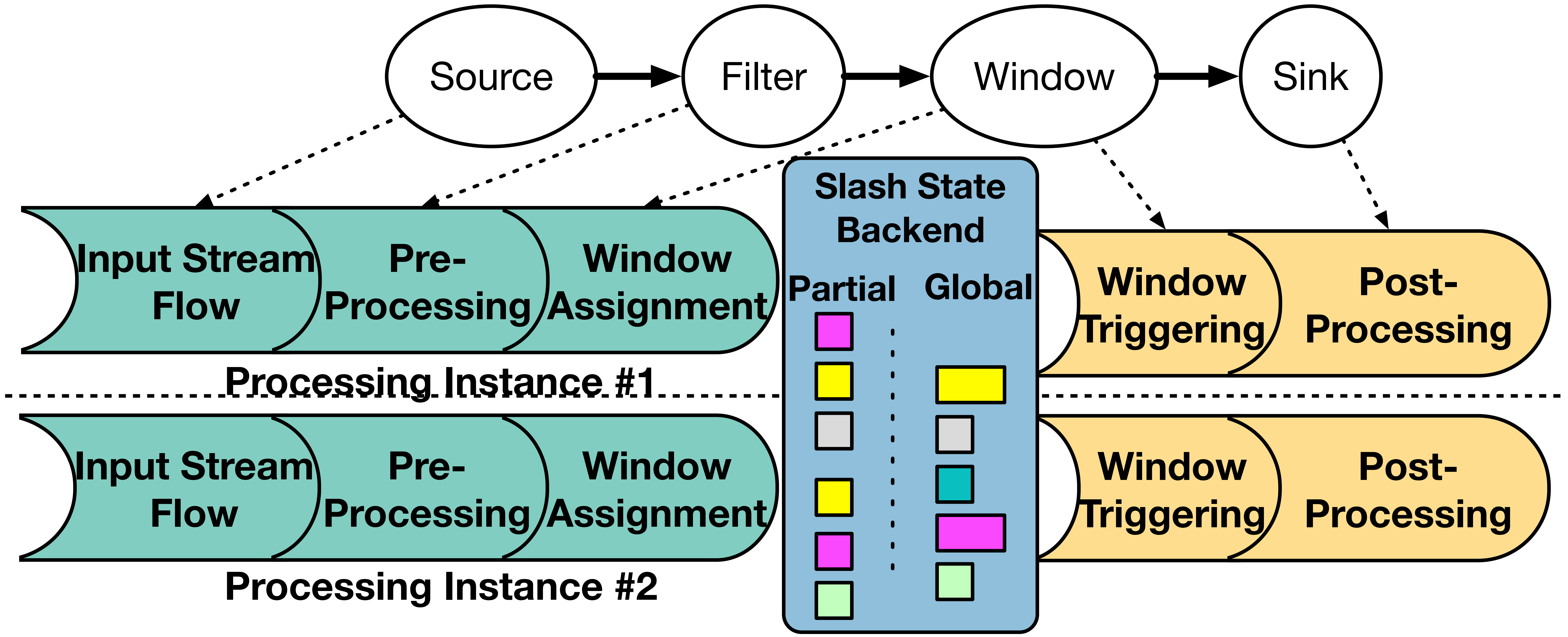
Going beyond rack-scale

- Slash requires a number of RDMA connections quadratic in the num of nodes
- Use Two-sided (Send/Recv) instead of RDMA Write/Read
  - Kalia et al.: RDMA requires NIC-managed connection state (a Connect-X5 RNIC drop 50% throughput with 5000 connections = 70 Slash instances)
  - RNIC SRAM: ~2 MB for connection and data structures, connection state ~375 bytes
  - Switch to application-managed connection state (datagram)
  - Requires software Congestion Control (e.g., rate-based) and achieves 70-92% of network throughput

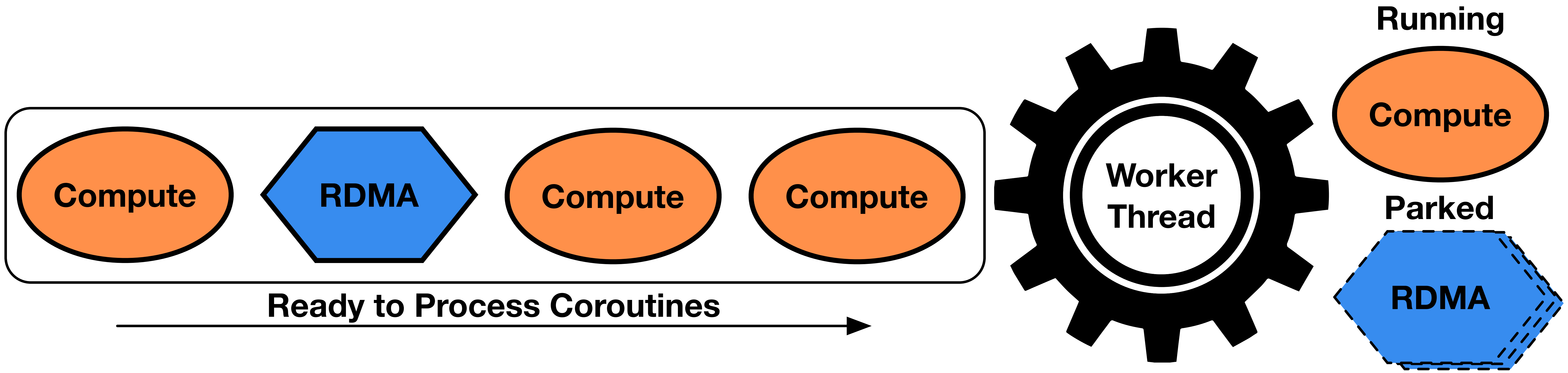
RDMA Atomics

- Bound by PCI-Ex RTT as a lock in the RNIC is held until the op is completed
  - 100s of ns with PCI-Express 3.0
  - Should evaluate with PCI-Express 5.0 and newer models?
  - Atomic semantics are atomic only among RNICs not CPU
  - Consensus in the Network community on avoiding them

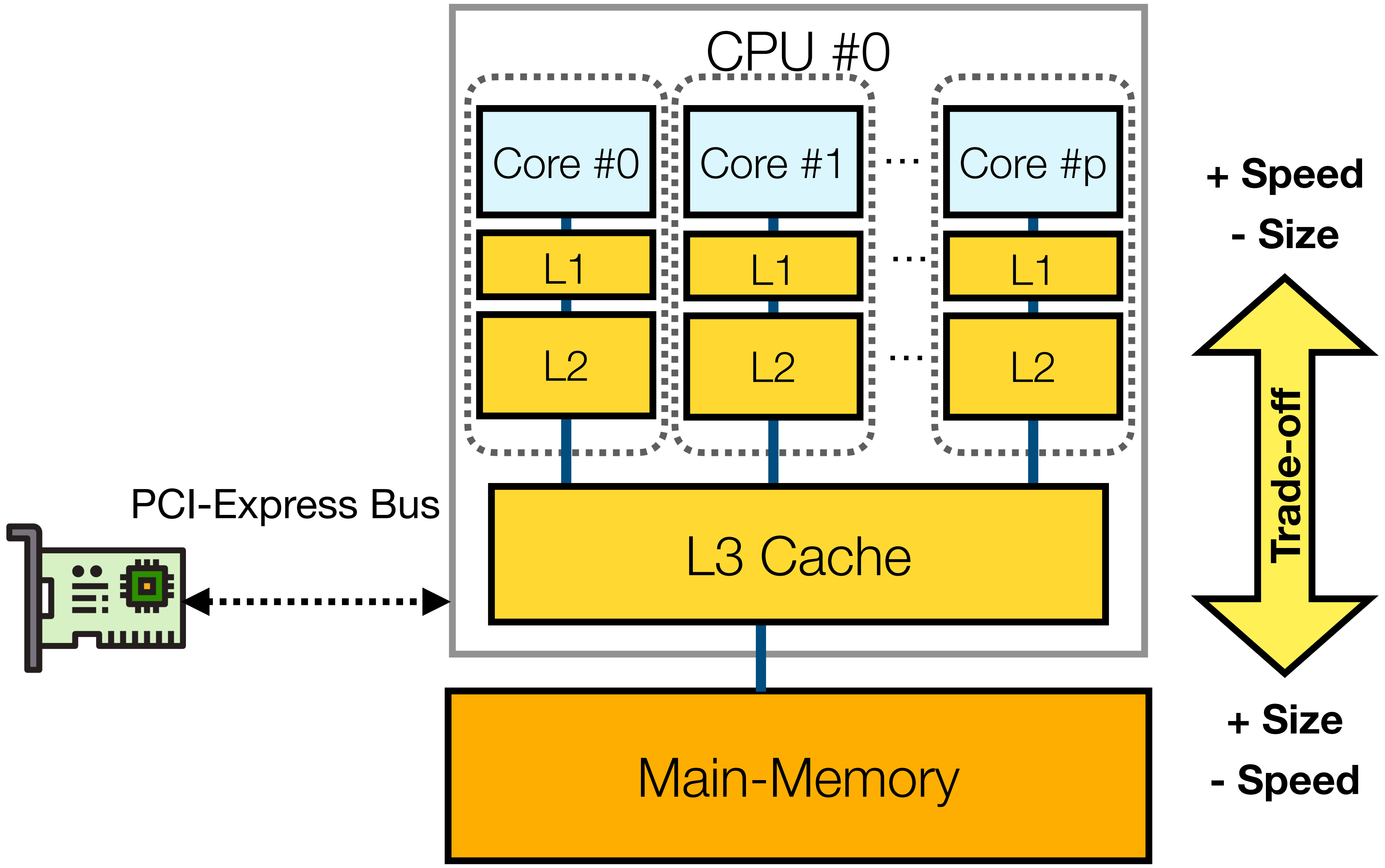
Slash internal processing



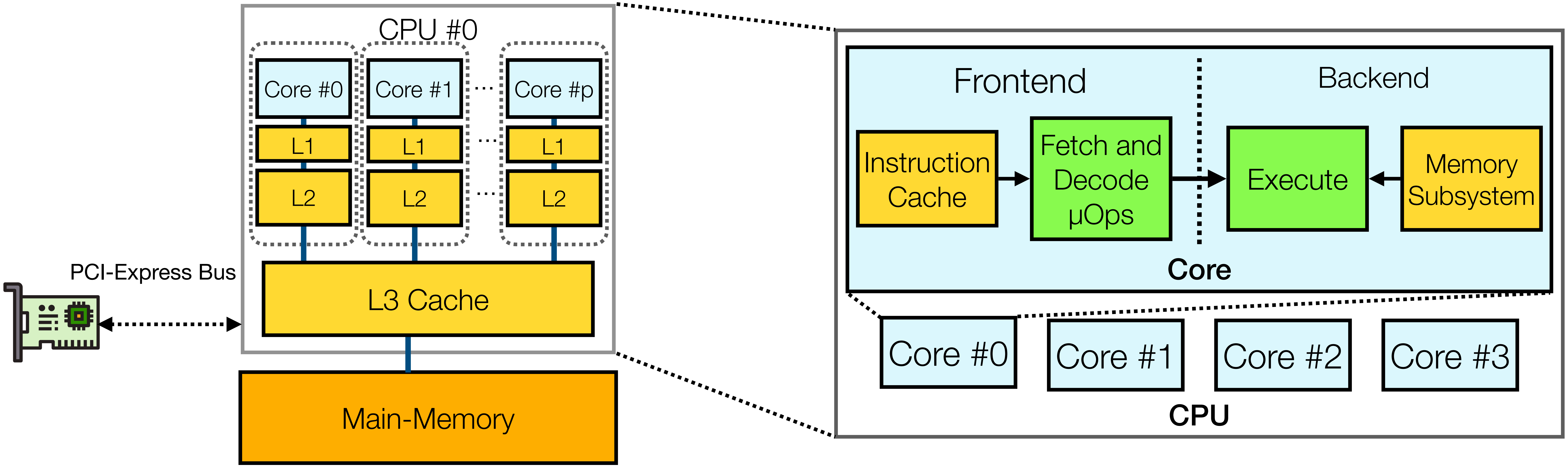
Slash internal processing



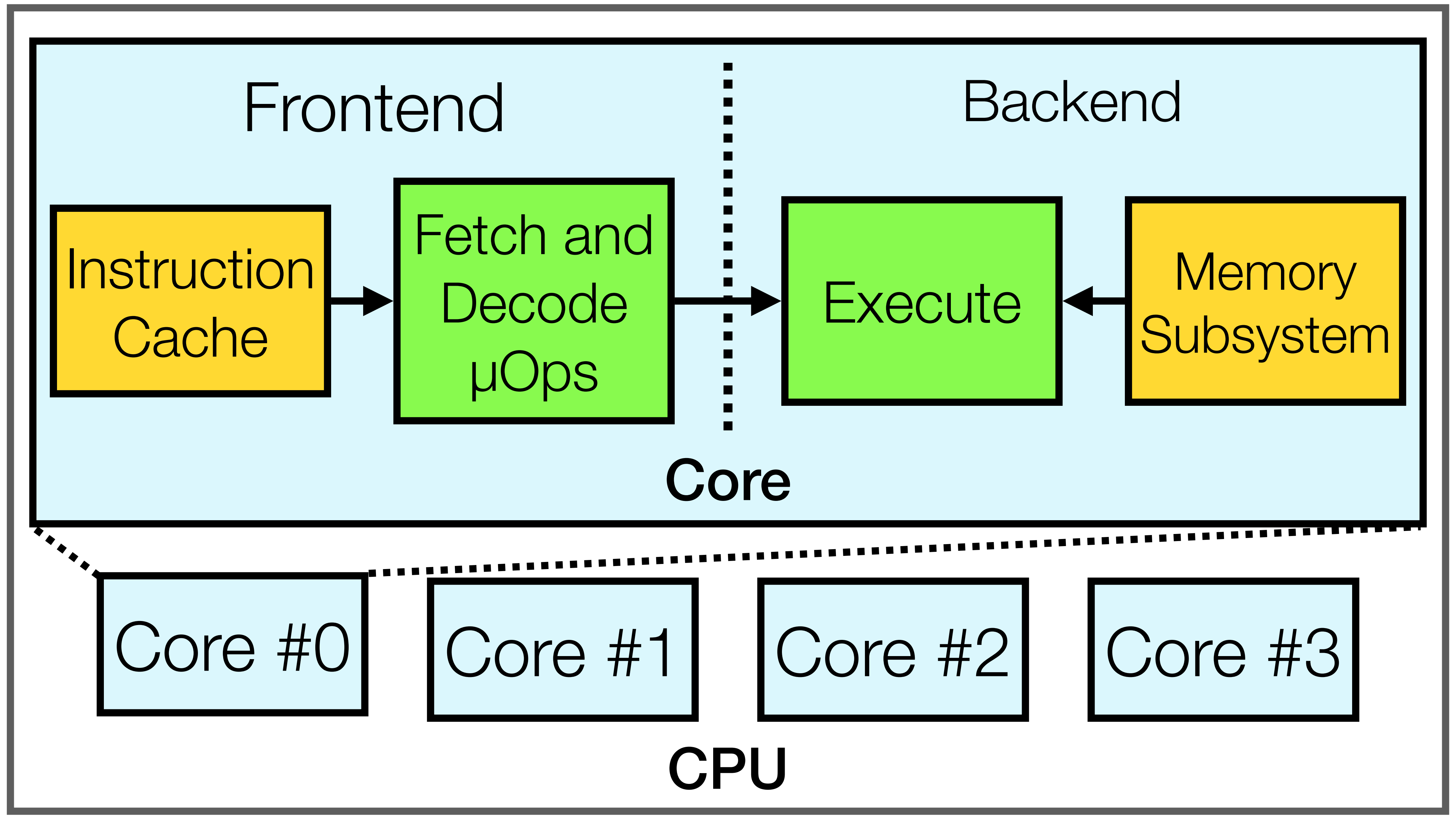
Micro-architecture Analysis



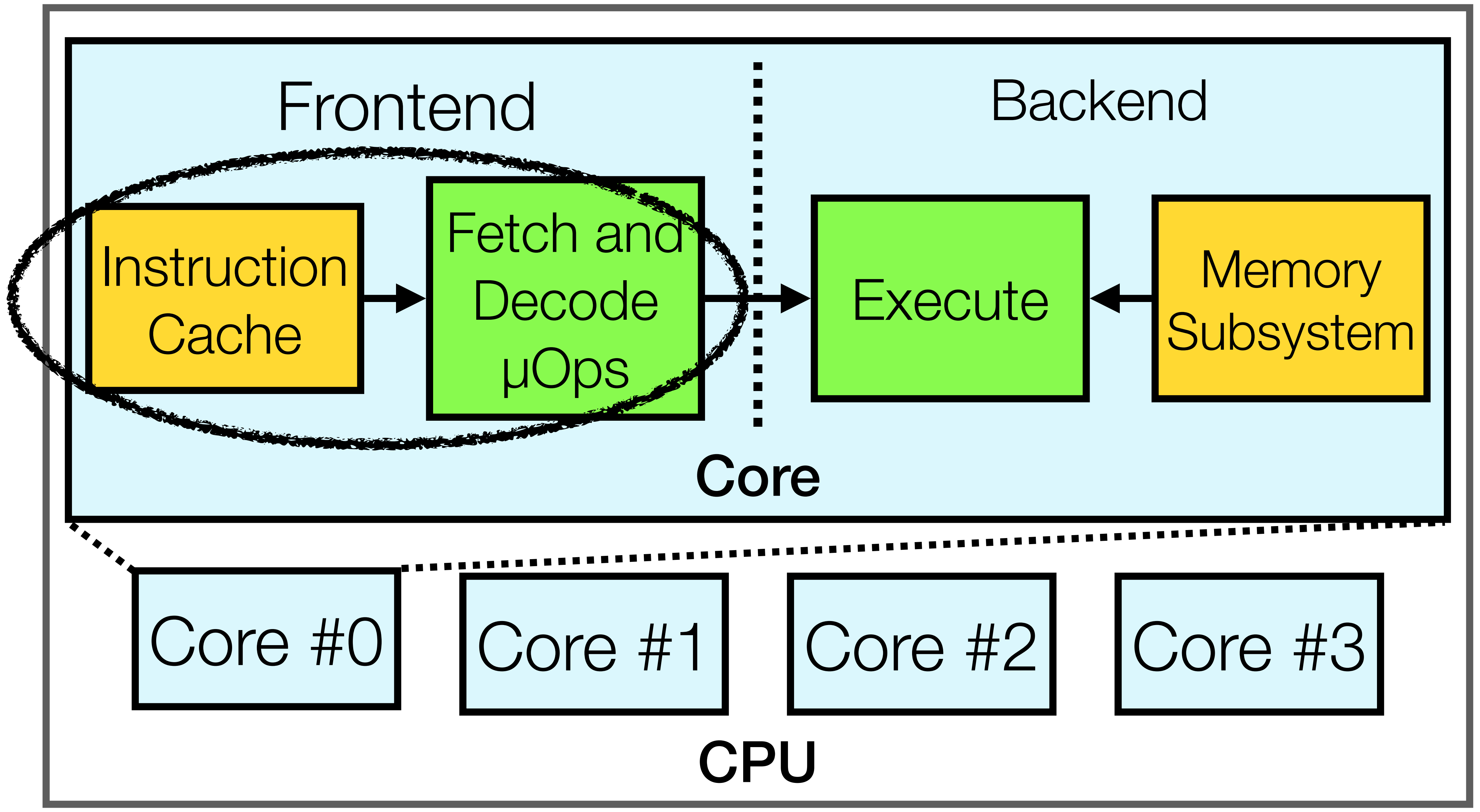
Micro-architecture Analysis



Micro-architecture Analysis

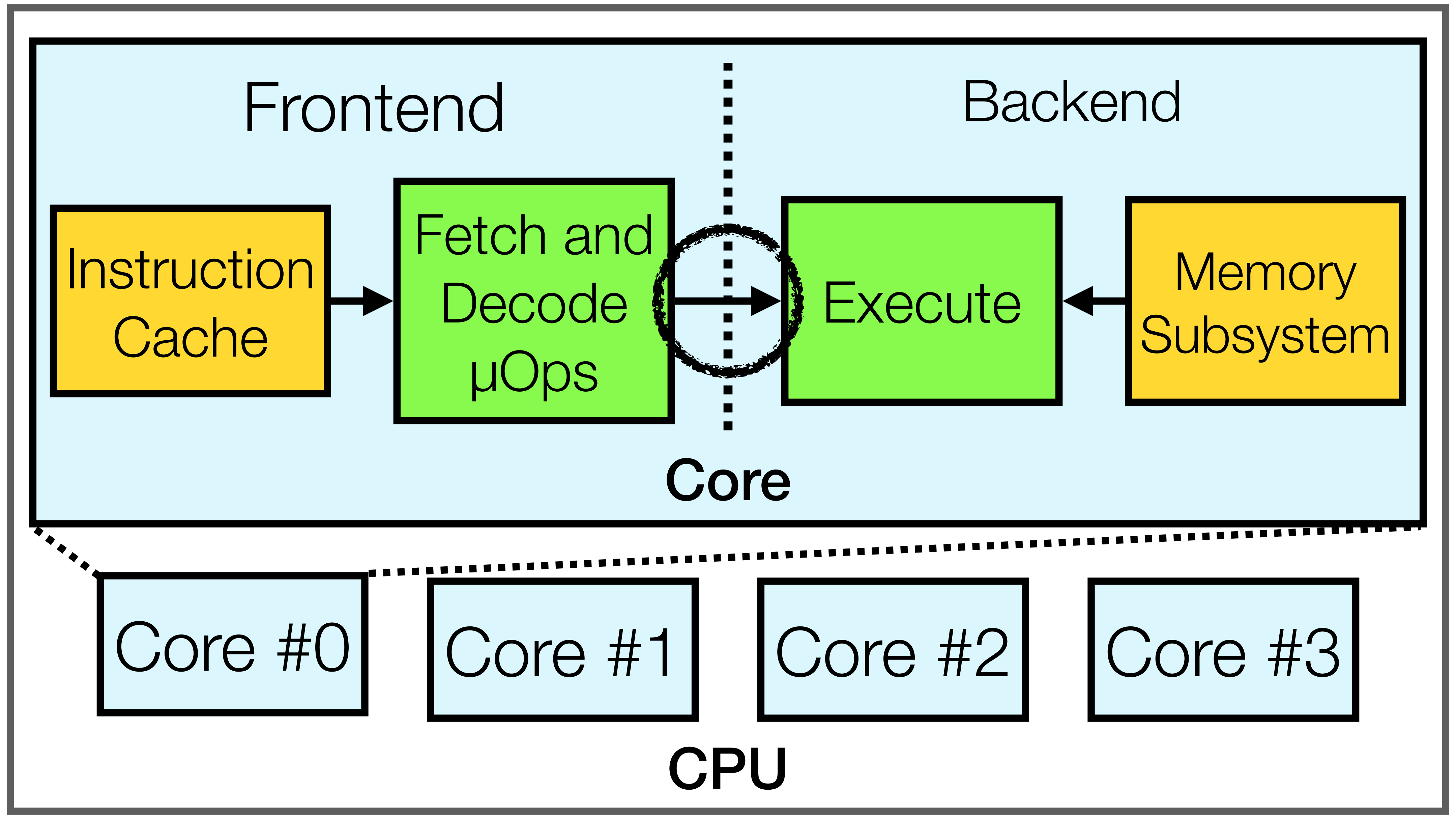


Micro-architecture Analysis



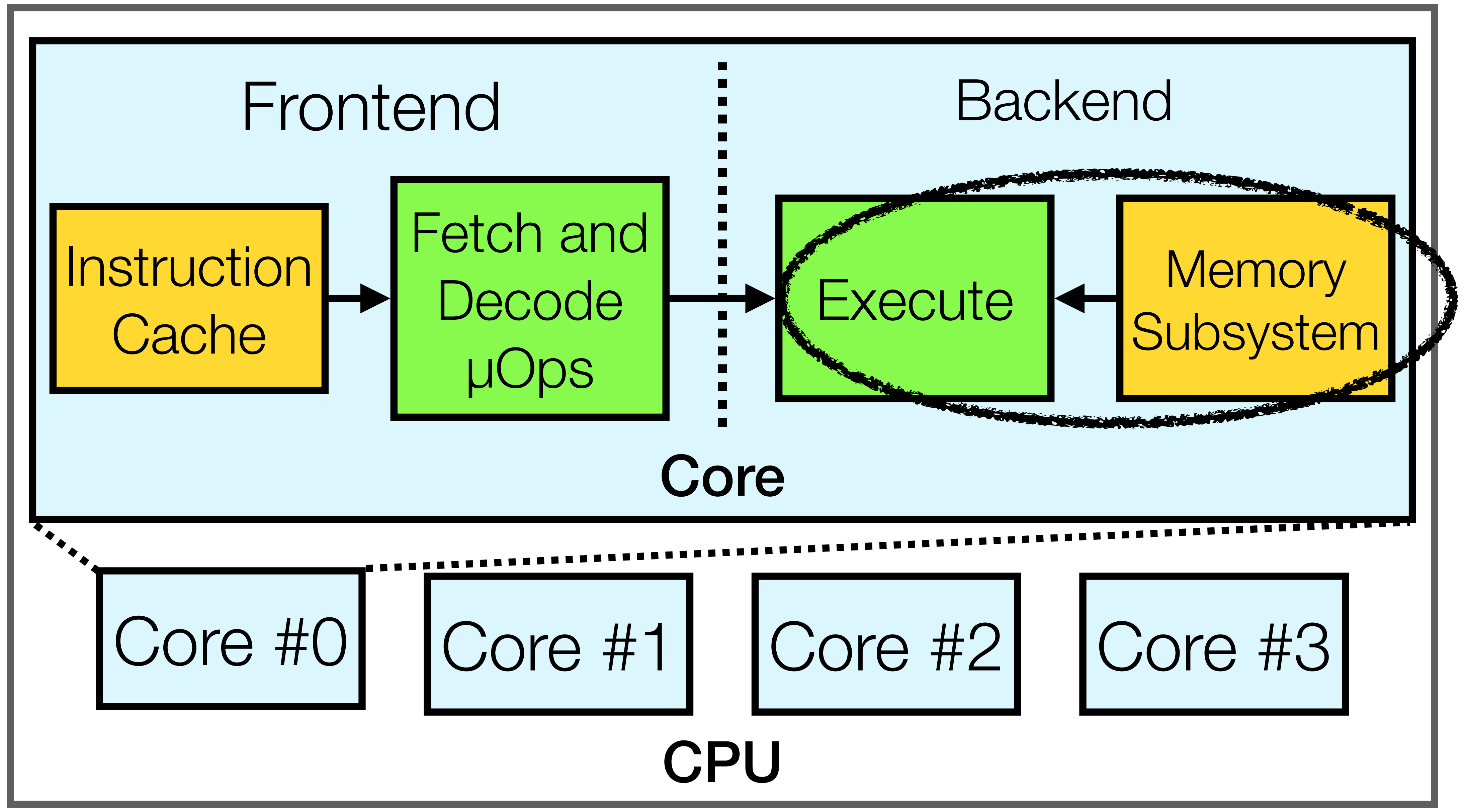
Complex instructions in L1i decoded in µOps

Micro-architecture Analysis



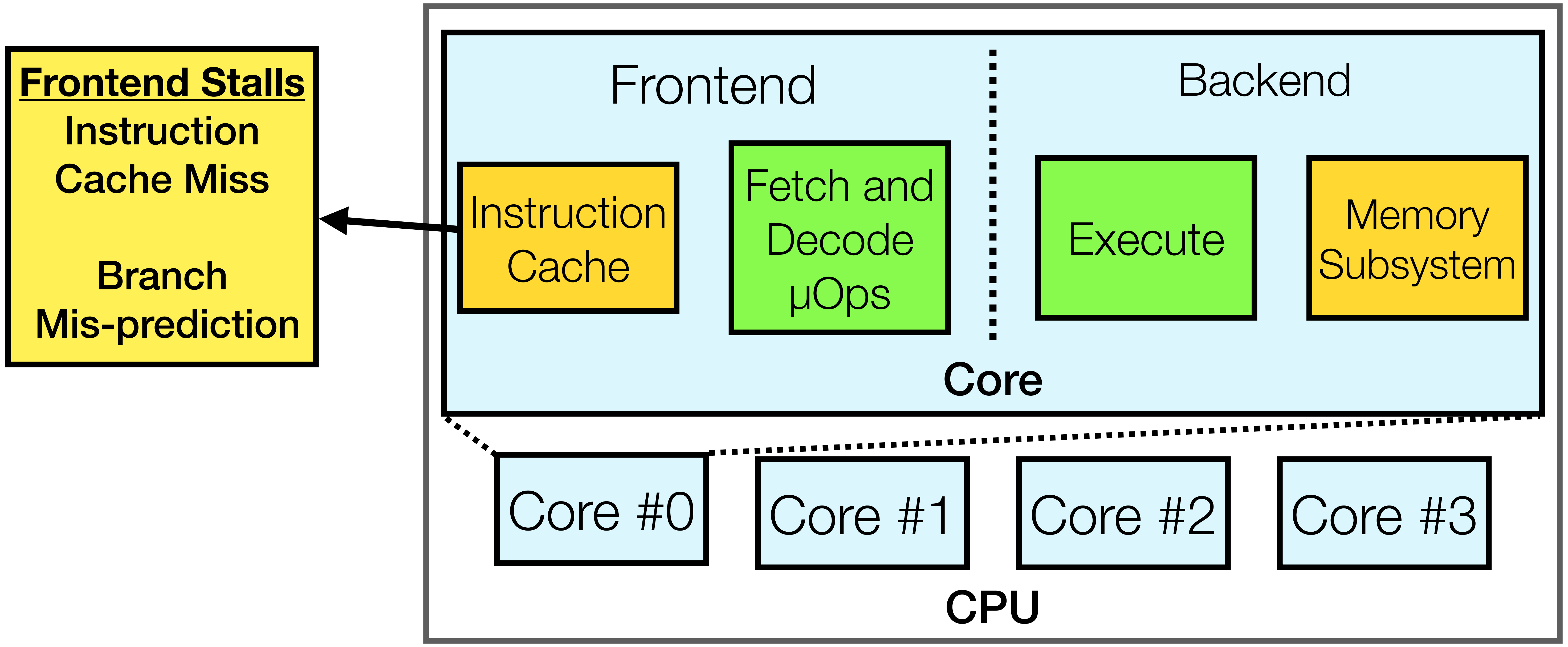
Frontend delivers up to 4 µOps per cycle to backend (Intel)

Micro-architecture Analysis

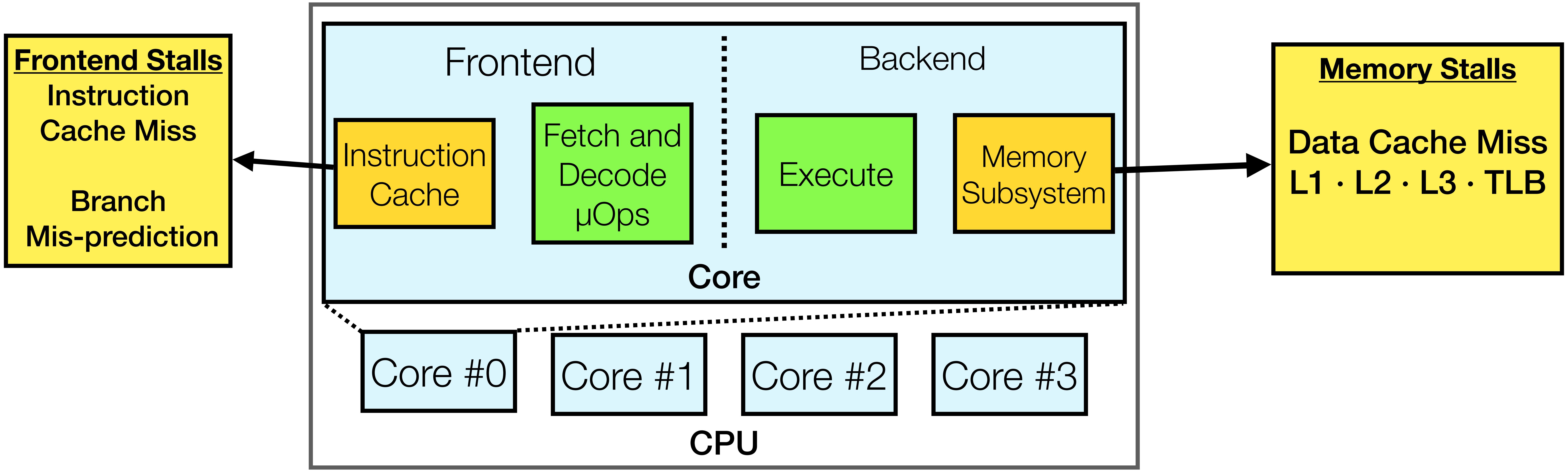


Provides data to registers from L1d, L2, LLC, and Main-Memory

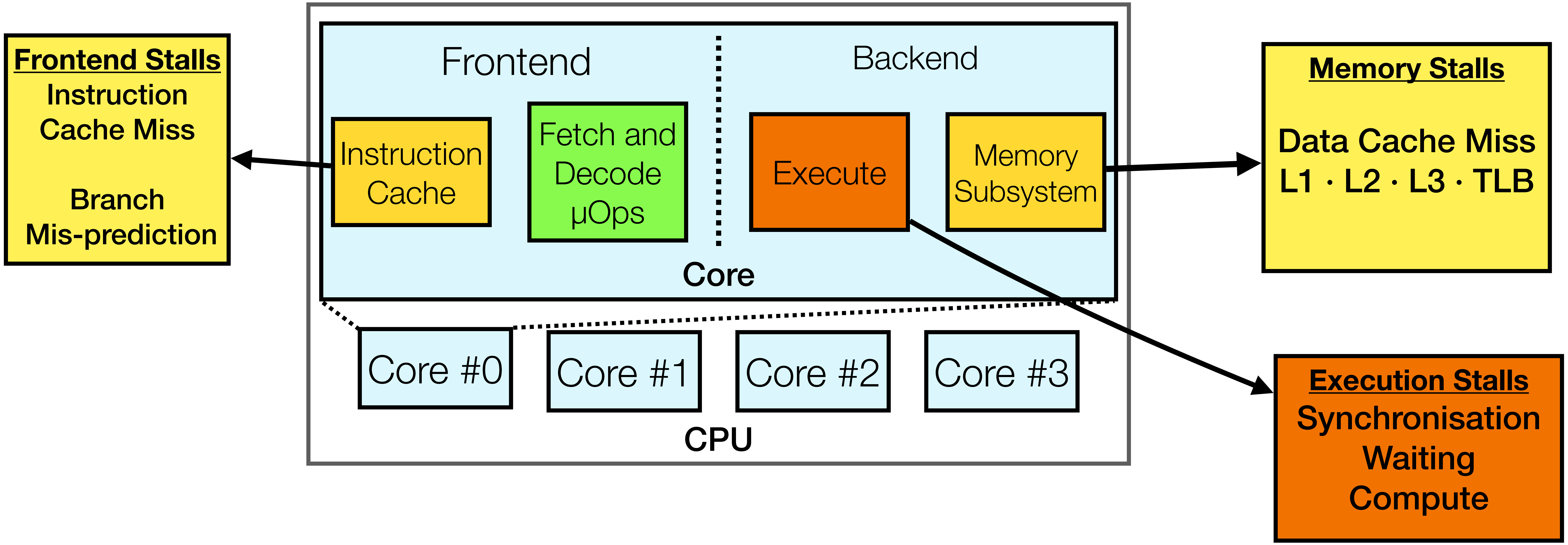
Micro-architecture Analysis



Micro-architecture Analysis

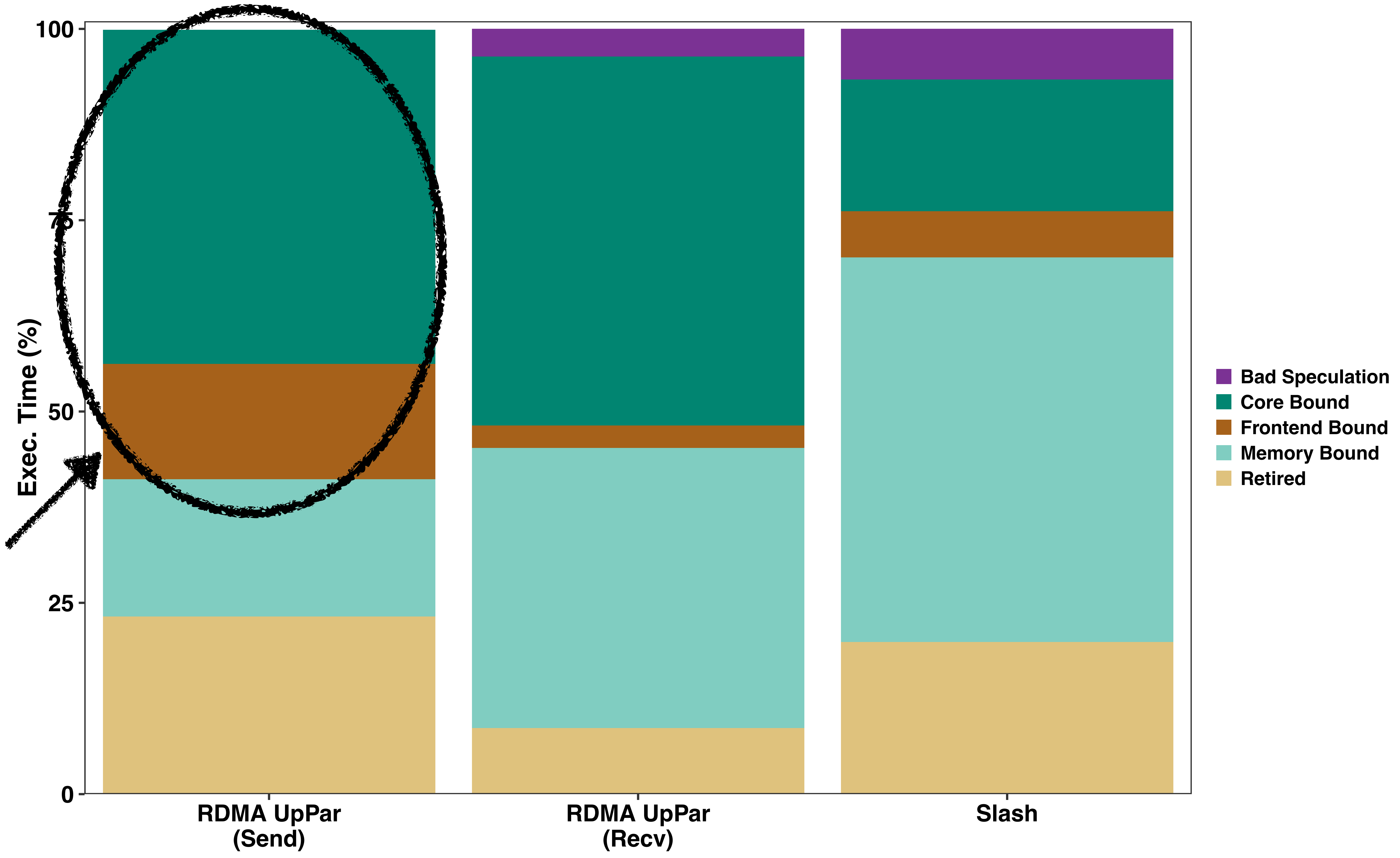


Micro-architecture Analysis



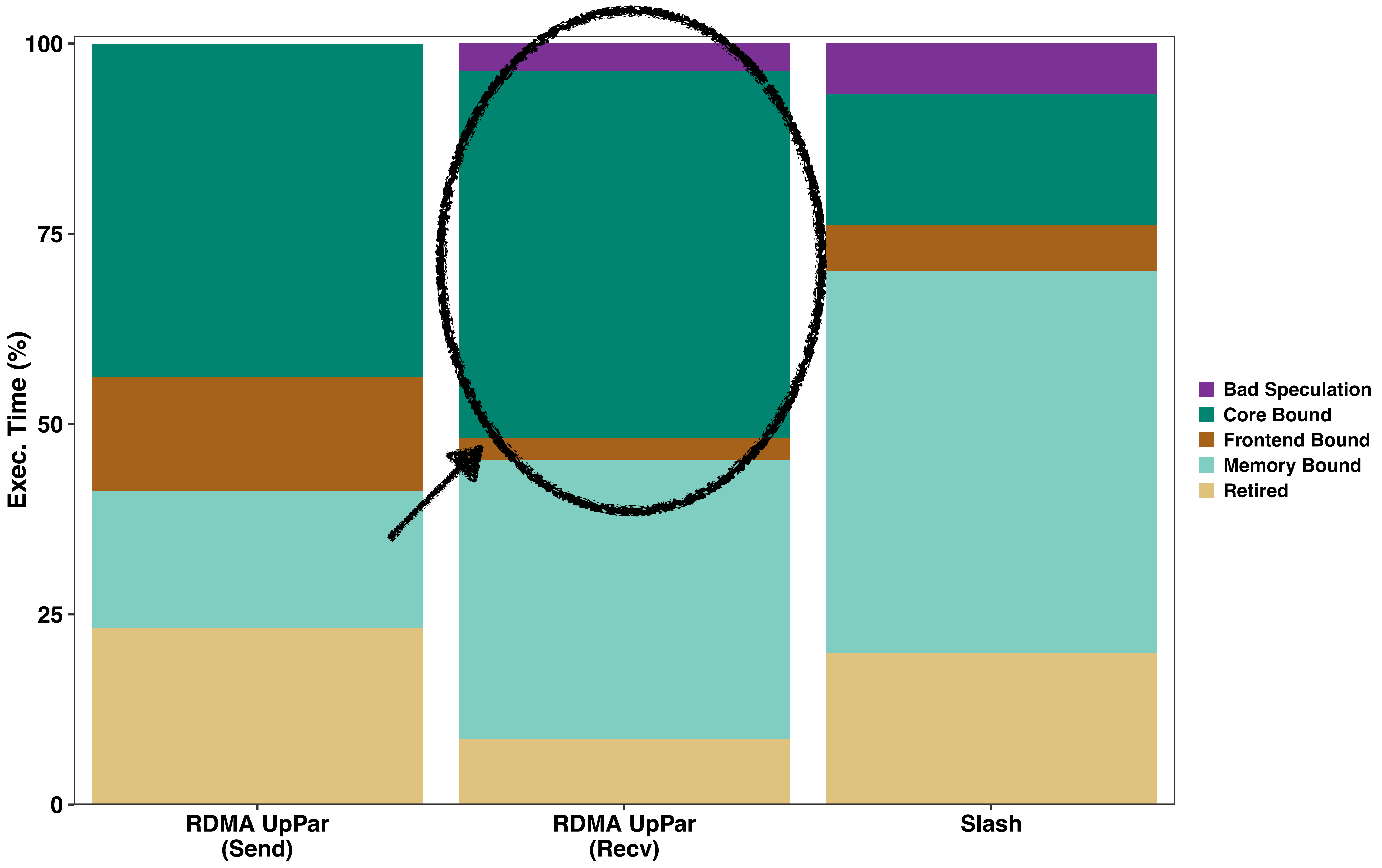
Hardware Performance Counters help us understand CPU performance

Slash Performance Gain Explained



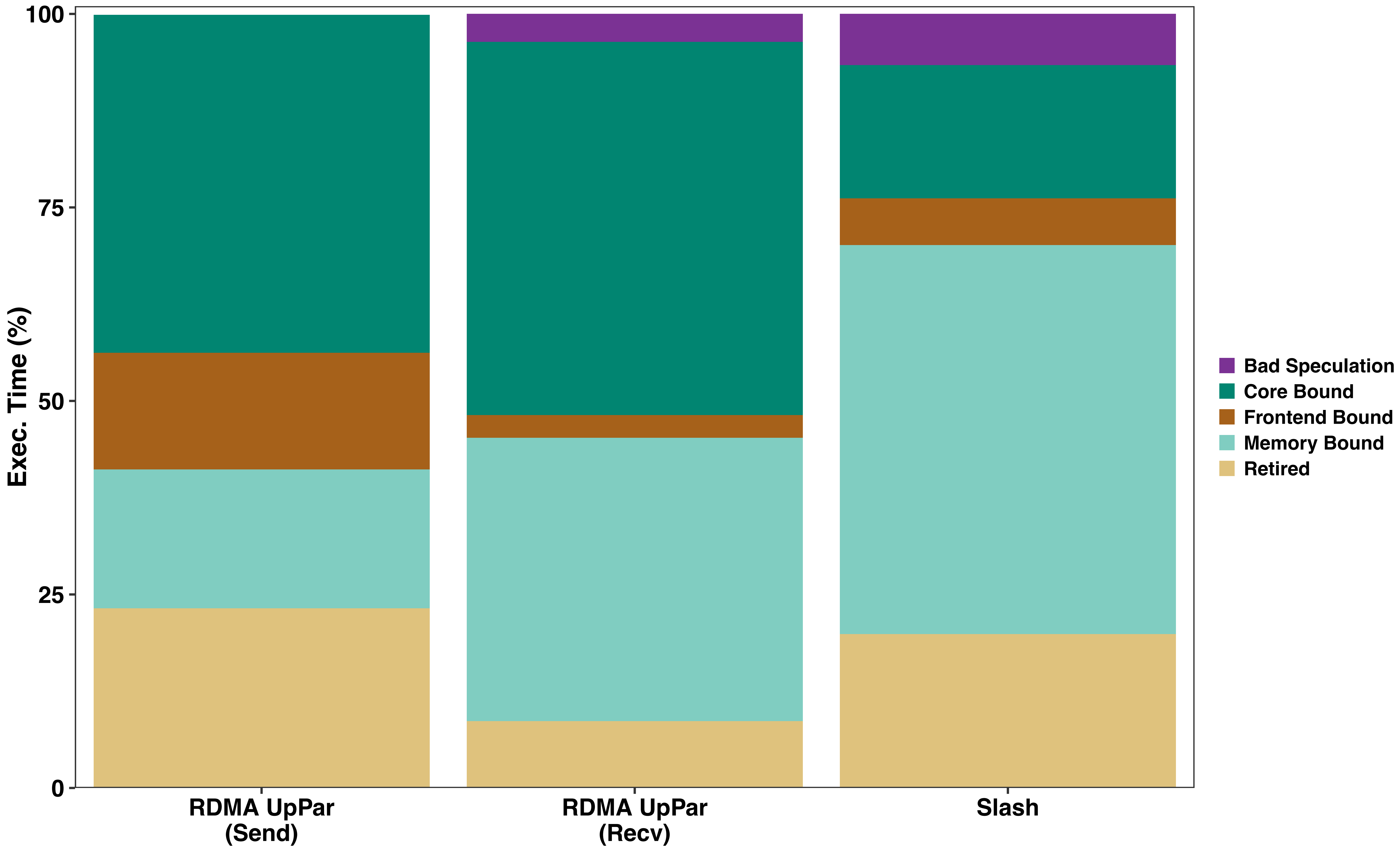
Sender of RDMA UpPar is Frontend and Core Bound
Partitioning involves complex code and spin waiting

Slash Performance Gain Explained



Receiver of RDMA UpPar spin waits on data from the sender

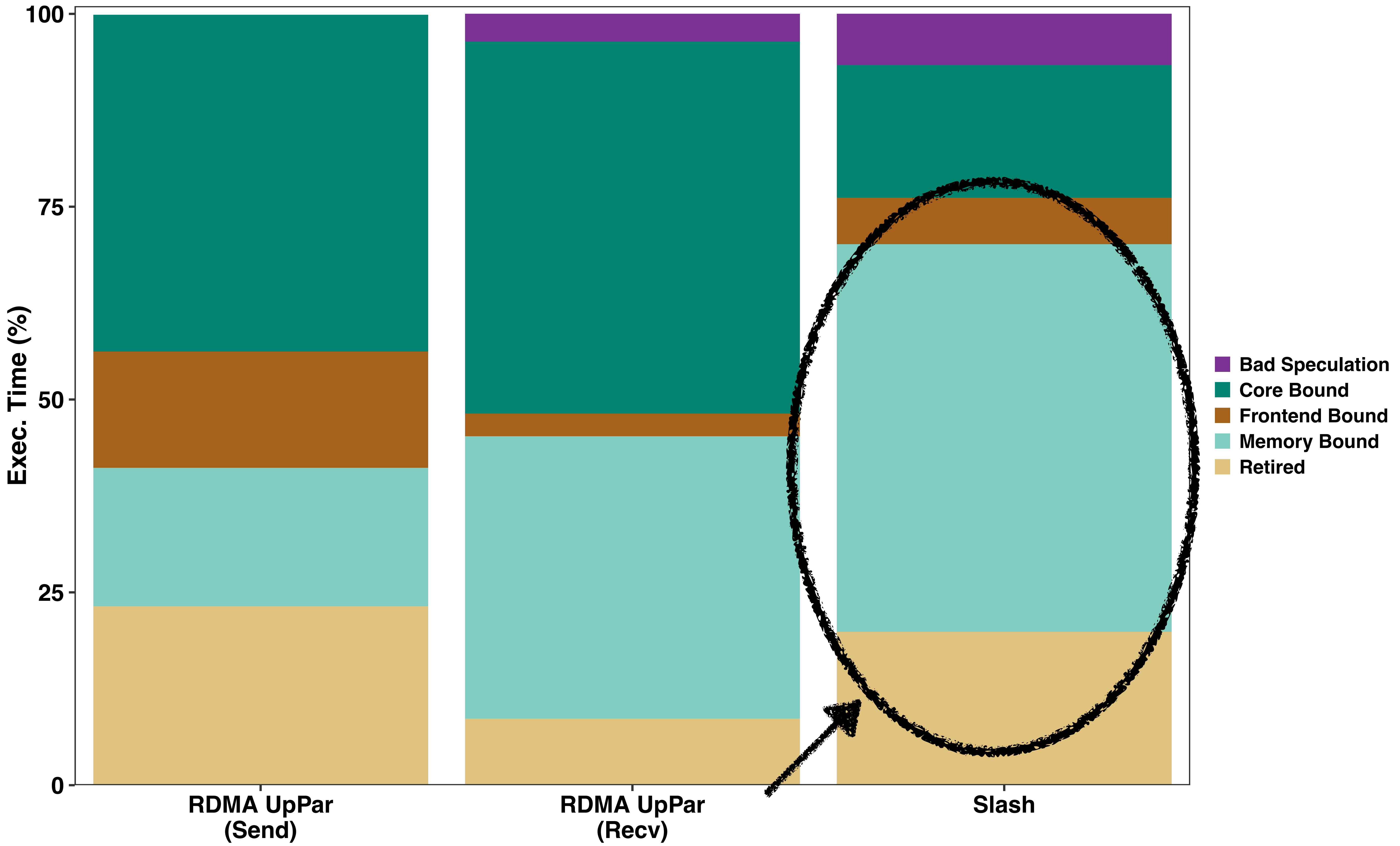
Slash Performance Gain Explained



RDMA UpPar limited by partitioning speed (CPU-Bound)

Receiver of RDMA UpPar spin waits on data from the sender

Slash Performance Gain Explained



RDMA UpPar limited by partitioning speed (CPU-Bound)

Slash is bound by memory speed

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

Slash waits for data to be materialized for processing