Rhino: Efficient Management of Very Large Distributed State for Stream Processing Engines

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

ACM SIGMOD 2020

TU berlin

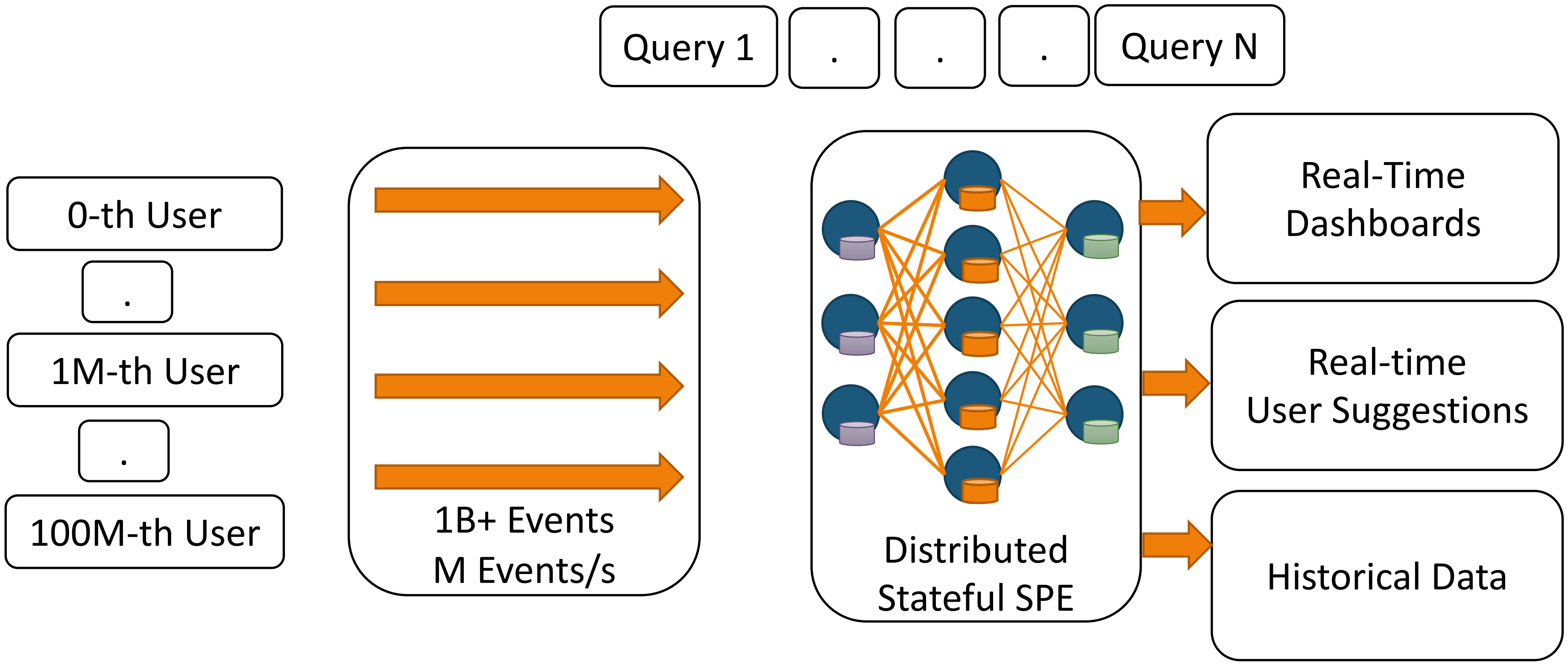
German
Research Center
for Artificial
Intelligence

HPI Hasso
Plattner
Institut
Digital Engineering · Universität Potsdam

What is this talk about?

Enabling *Continuous Stateful Stream Processing*
in the presence of *TB-sized operator state*,
regardless of *failures* and *data rate fluctuations*

Use case: a real-time bidding platform



Query 1

.

.

.

Query N

0-th User

.

1M-th User

.

100M-th User

1B+ Events
M Events/s

Distributed
Stateful SPE

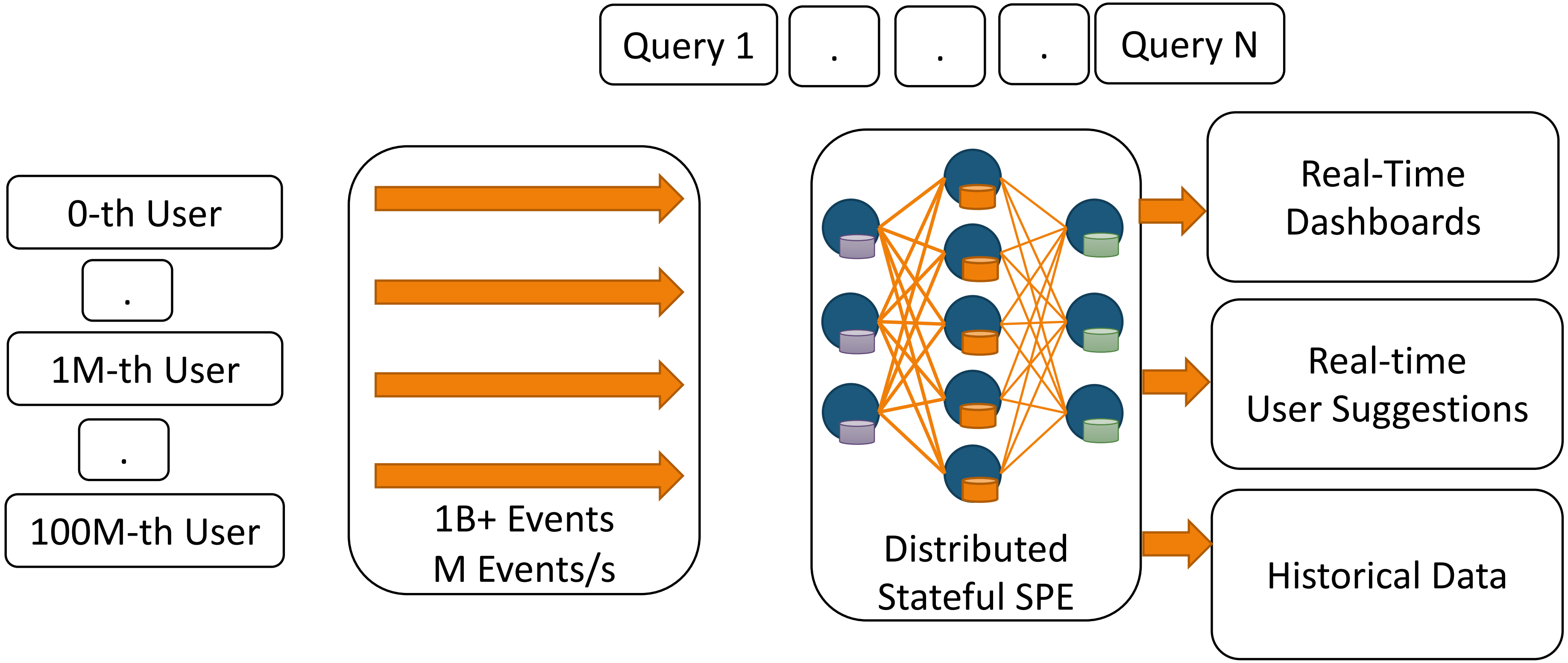
Real-Time
Dashboards

Real-time
User Suggestions

Historical Data

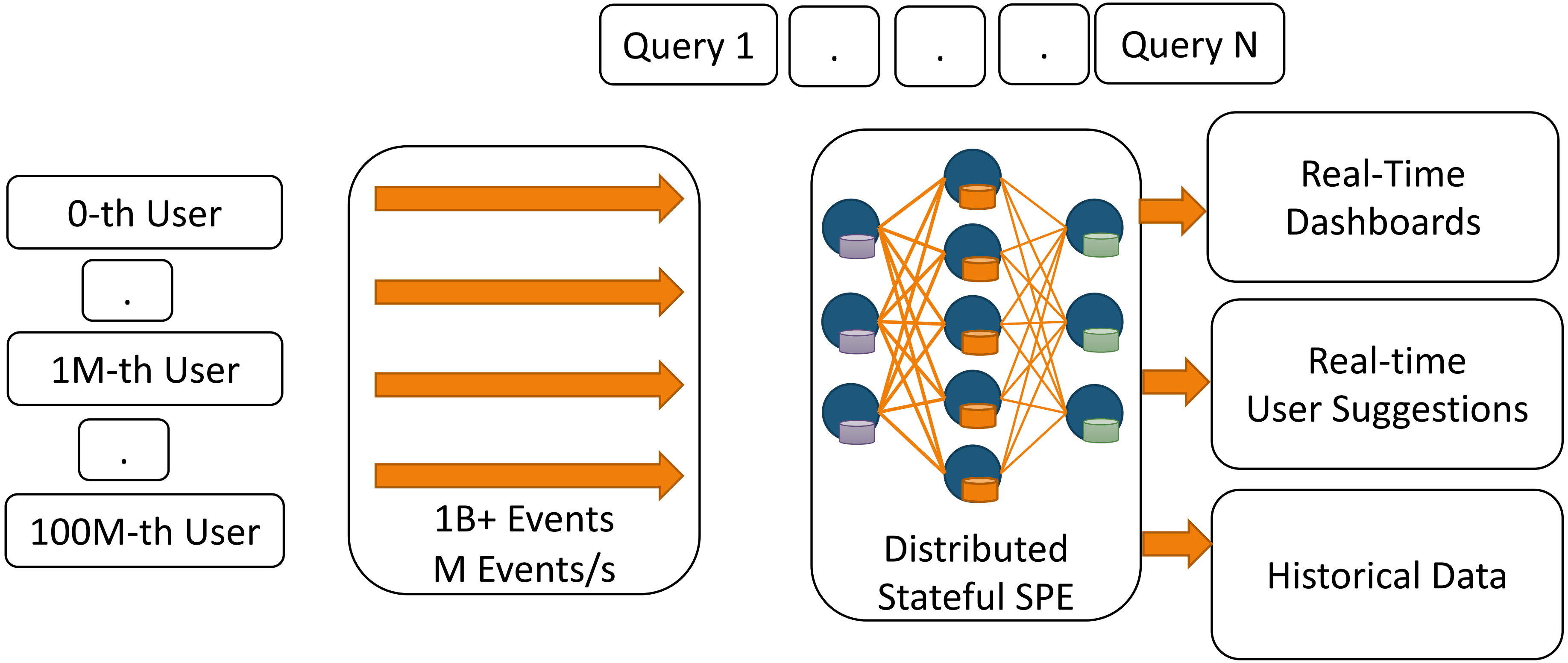
High-cardinality data stream

Use case: a real-time bidding platform



High-cardinality data stream + long-running stateful queries + large temporal aggregations or joins

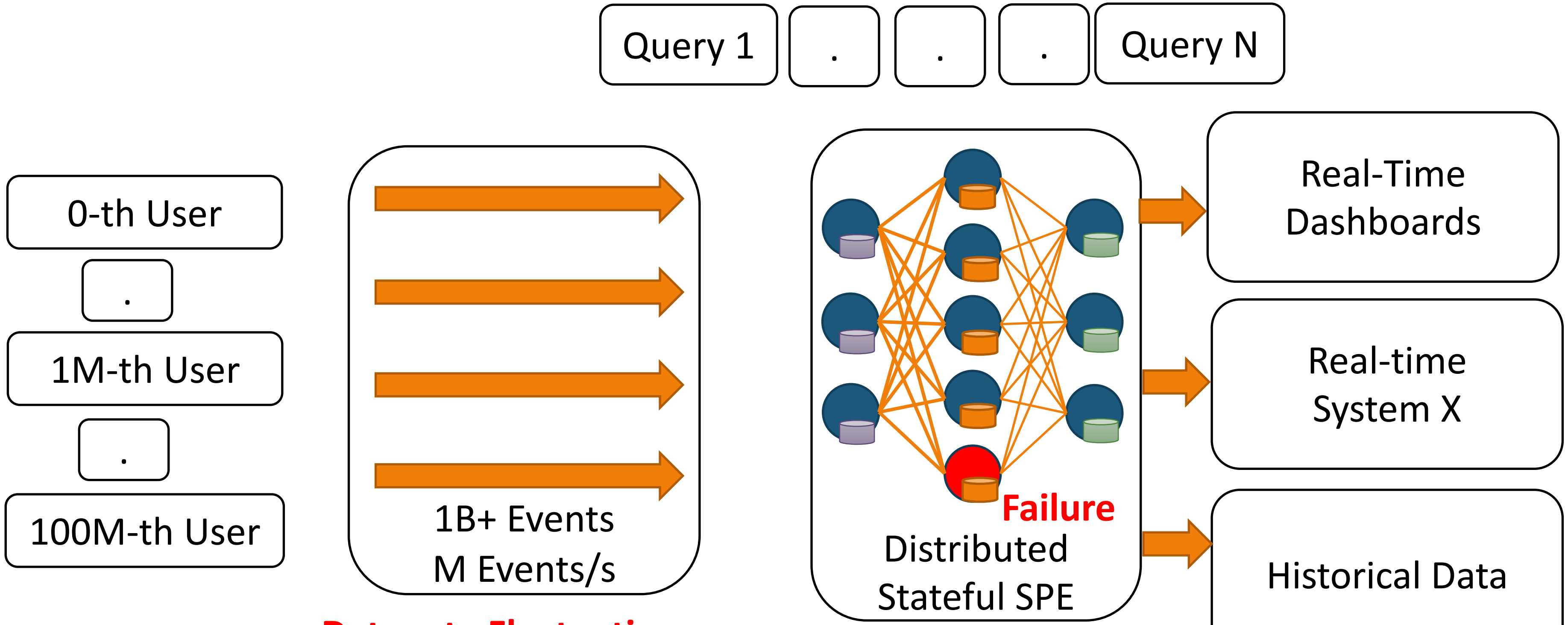
Use case: a real-time bidding platform



High-cardinality data stream + long-running stateful queries + large temporal aggregations or joins =

Very Large Distributed State

What could go wrong?



Query 1 . . . Query N

0-th User

.

1M-th User

.

100M-th User

1B+ Events
M Events/s

Failure

Distributed
Stateful SPE

Real-Time
Dashboards

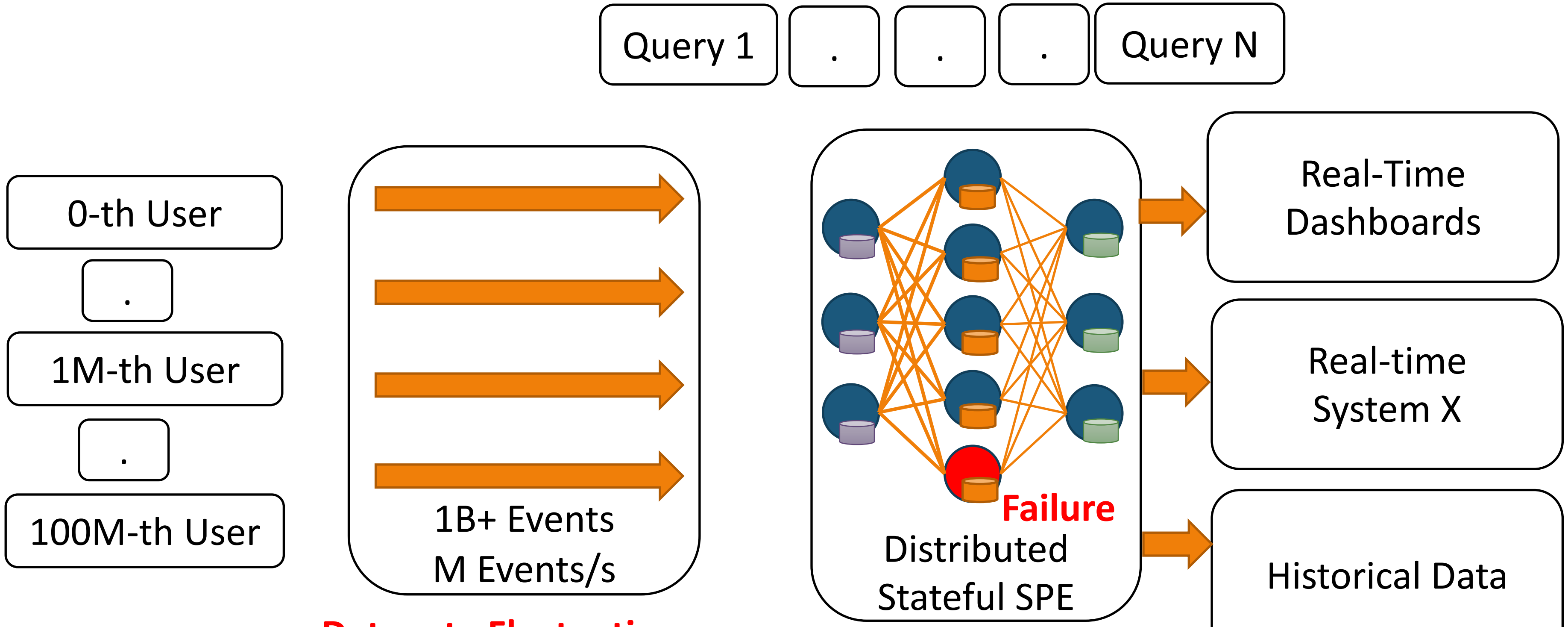
Real-time
System X

Historical Data

Data rate Fluctuation

Very Large Distributed State + anomalous operational events

What could go wrong?



Query 1 . . . Query N

0-th User

.

1M-th User

.

100M-th User

1B+ Events
M Events/s

Failure

Distributed
Stateful SPE

Real-Time
Dashboards

Real-time
System X

Historical Data

Data rate Fluctuation

Very Large Distributed State + anomalous operational events =
slow reconfiguration = high latency + downtime + data loss = **DISASTER**

What about current SPEs?

Production-ready SPEs

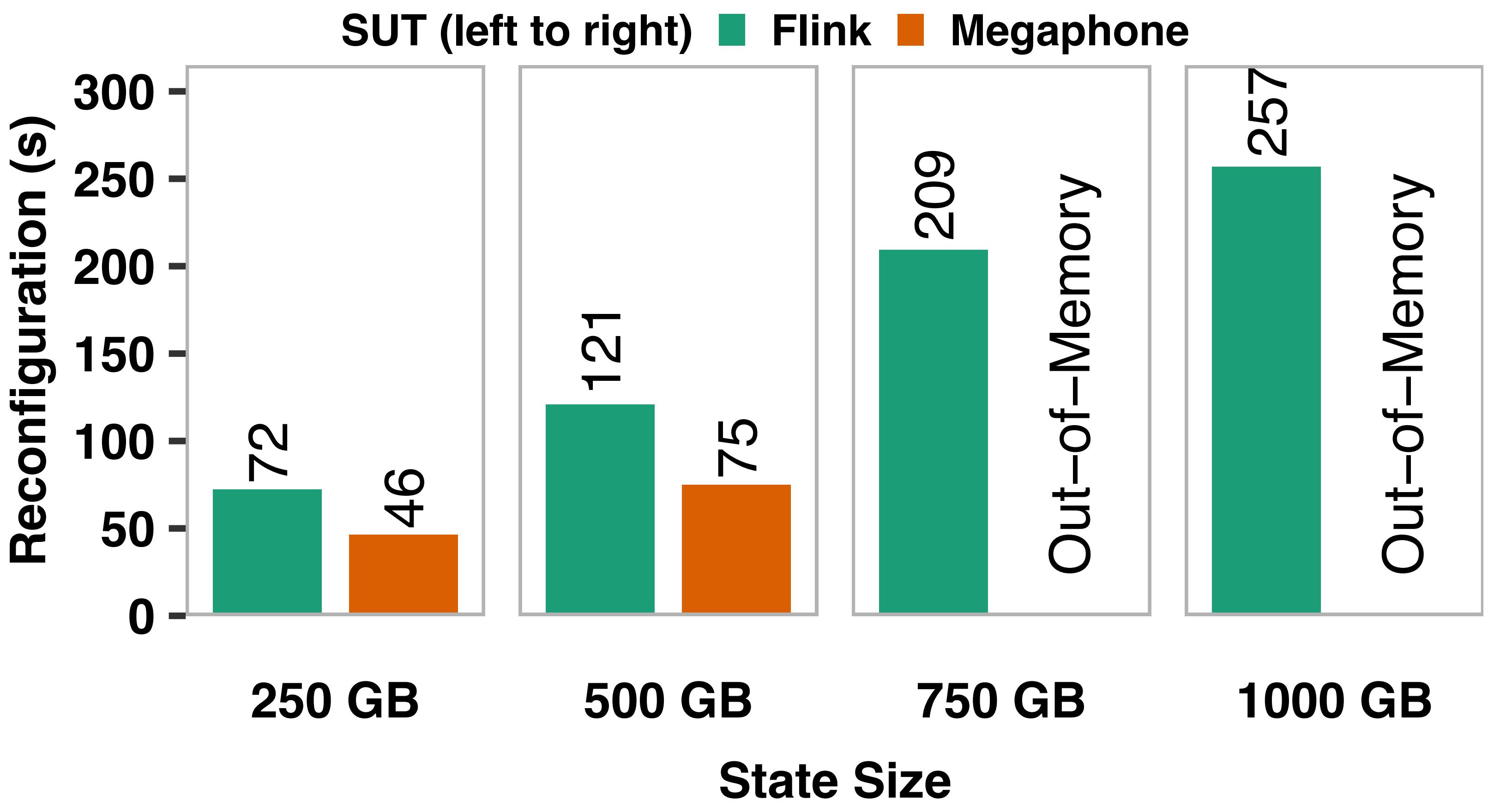
- Spark/Flink/Storm
- Reconfiguration via restart ❌
- Support TB-sized State 🙂

Research Prototypes

- Megaphone/Chi/SDG/SEEP
- Fine-grained reconfiguration 🙂
- Small state size ❌

We want to efficiently support TB-sized state and provide fine-grained reconfiguration of running queries

Do we really need yet a new system?



NeXMark Query 8 (Large Windowed Join) on 8+1 cloud instances
State-of-the-art SPEs are not ready to handle reconfiguration with TB-sized state

Research Goal

Efficient State Management and *on-the-fly Query Reconfiguration*
in the presence of TB-sized Operator State to support:

Fault-tolerance
Resource Elasticity
Runtime Optimizations

Research Challenge

1. *Processing overhead*: minimal impact on query processing performance

2. *Consistency*: do not break exactly-once progressing semantics

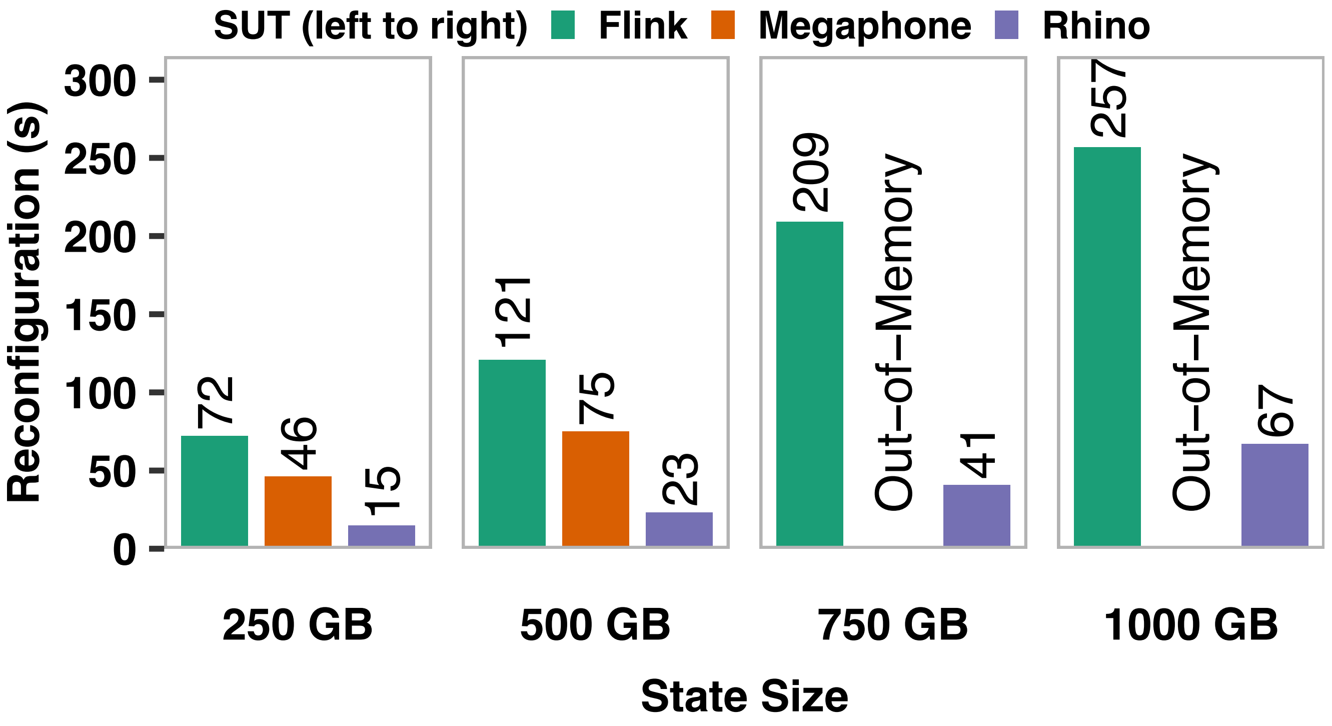
3. *Network Overhead*: state migration

Our Solution: Rhino

1. Handover Protocol
    - Consistent reconfiguration without halting query execution

Our Solution: Rhino

1. Handover Protocol
   - Consistent reconfiguration without halting query execution

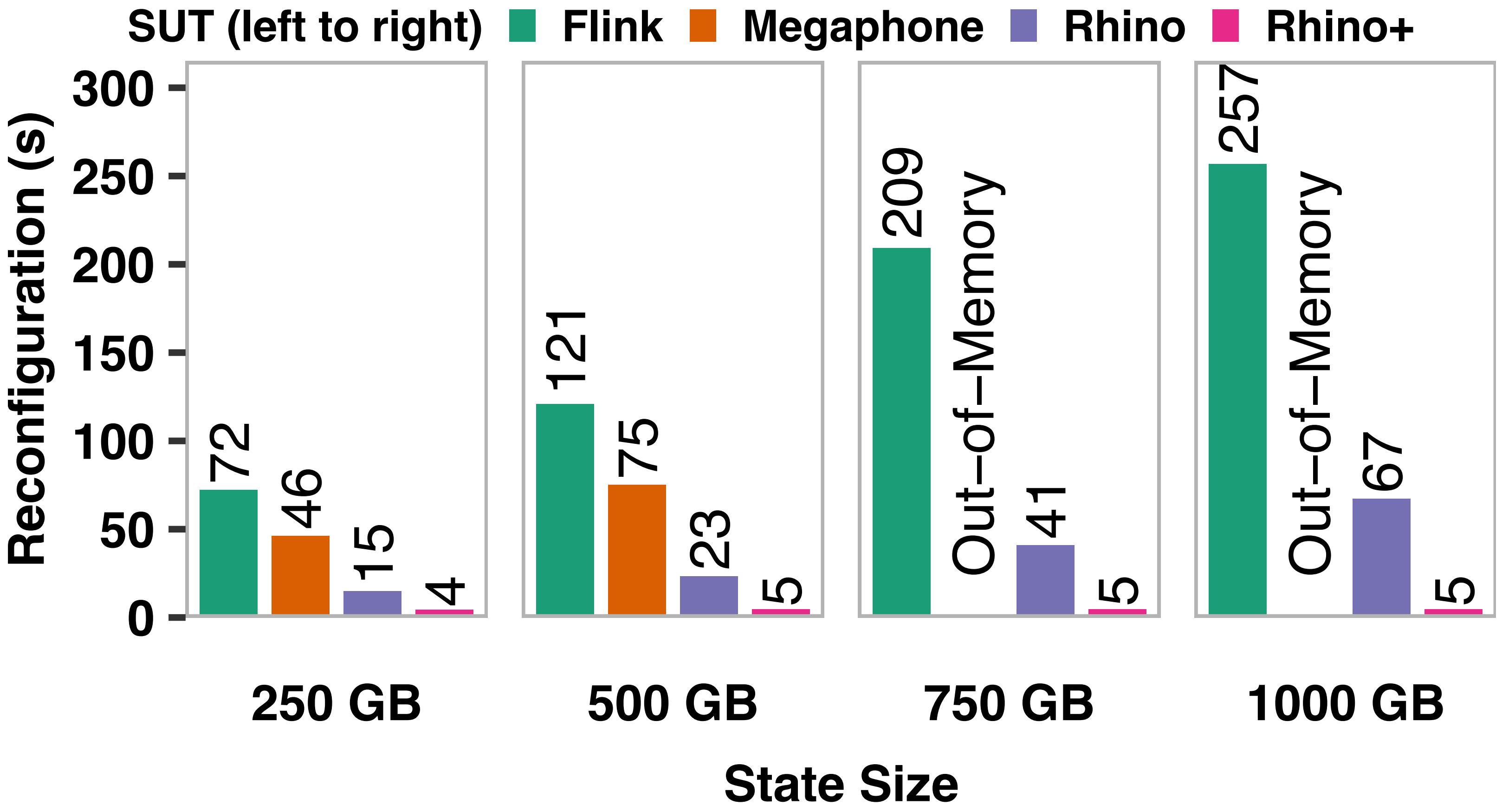


Our Solution: Rhino

1. Handover Protocol
   - Consistent reconfiguration without halting query execution

2. Proactive, Incremental State Migration Protocol
   - Tailored to efficiently transfer large state for future reconfigurations

Impact of Handover and State Migration



Reconfiguring a query with large operator state is feasible with minimal impact

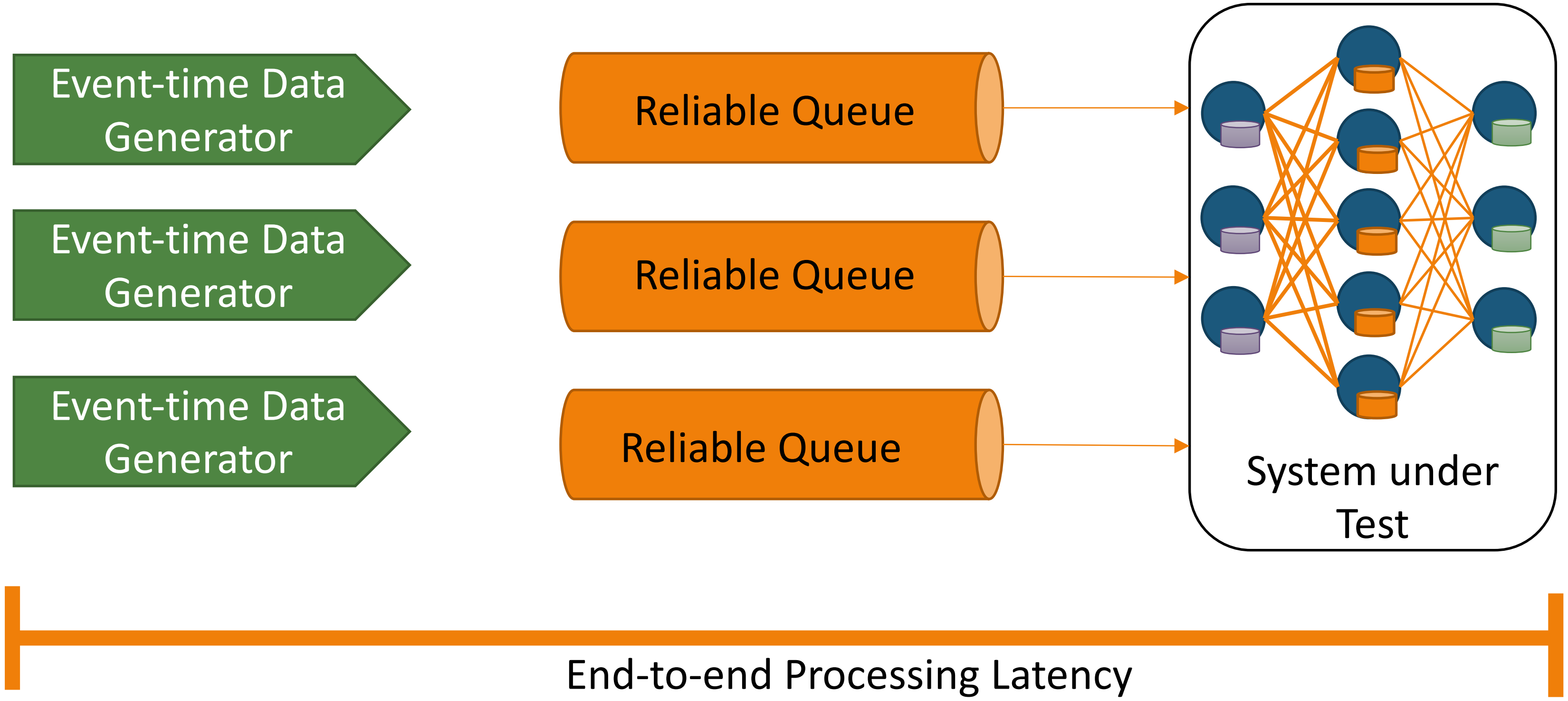
Our contribution

- Enable on-the-fly reconfiguration of running queries with large stateful operators

- Support for fault tolerance, resource elasticity, and runtime optimizations for running queries with large stateful operators

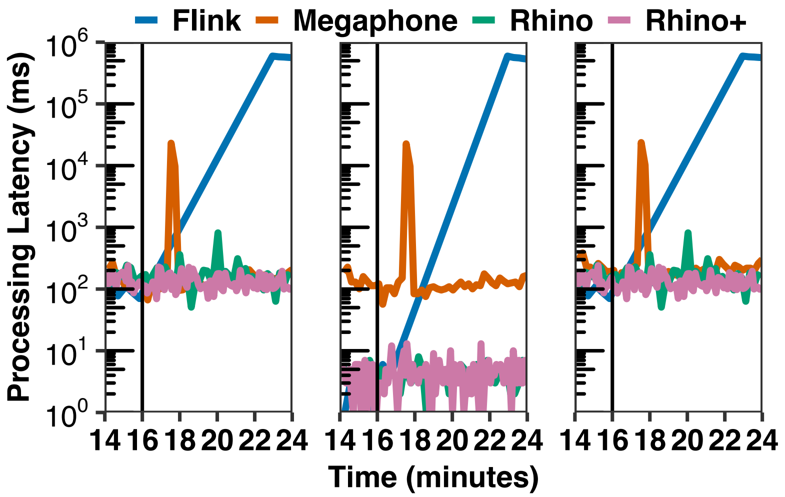
- Validation of our system design at TB scale

Experiments


Event-time Data Generator
Event-time Data Generator
Event-time Data Generator
Reliable Queue
Reliable Queue
Reliable Queue
System under Test
End-to-end Processing Latency

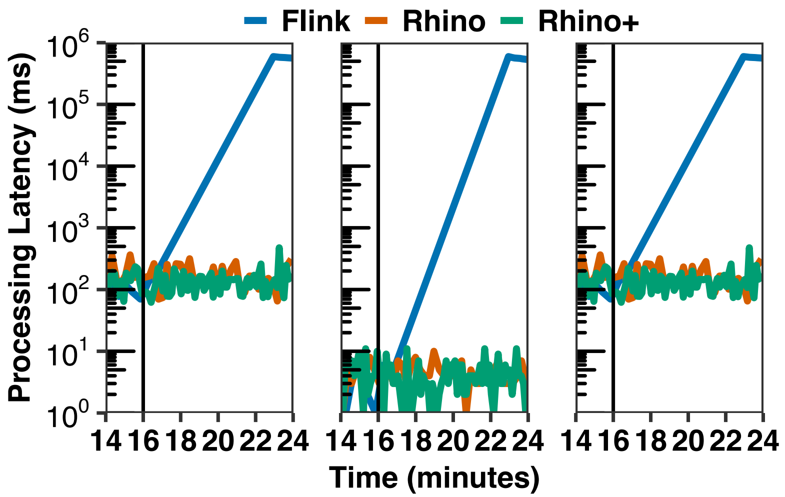
NeXMark Benchmark Suite (Q5-**<u>Q8</u>**-QX)
Distributed setting (16 VMs on GCP)

SUT just below saturation point on NBQ8



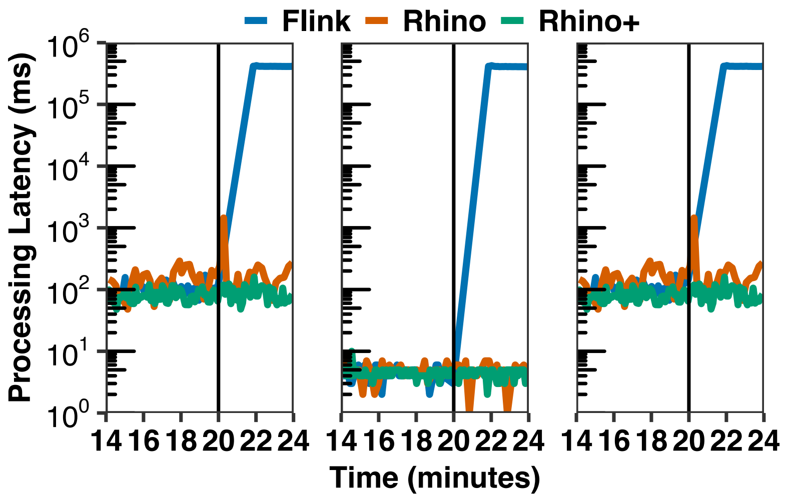
Avg Min P99

Load Balancing



Avg Min P99

Scaling Out



Avg Min P99

Fault Tolerance

Rhino keeps latency in check whereas baseline shows up to 3 orders of magnitude increment in latency

Conclusion

Rhino removes the bottleneck due to large state transfer upon a query reconfiguration

Enables fault-tolerance, resource elasticity, runtime optimizations for running stateful queries

Up to 3 orders of magnitude latency reduction upon a reconfiguration

Future Work on Stream Processing

- Rhino in Action: demo paper in 2021
  - Show-case of Rhino

Future Work on Stream Processing

- Rhino in Action: demo paper in 2021
  - Show-case of Rhino

- RDMA-enabled Stream Processing Engine
  - Can we perform stream processing at line rate?

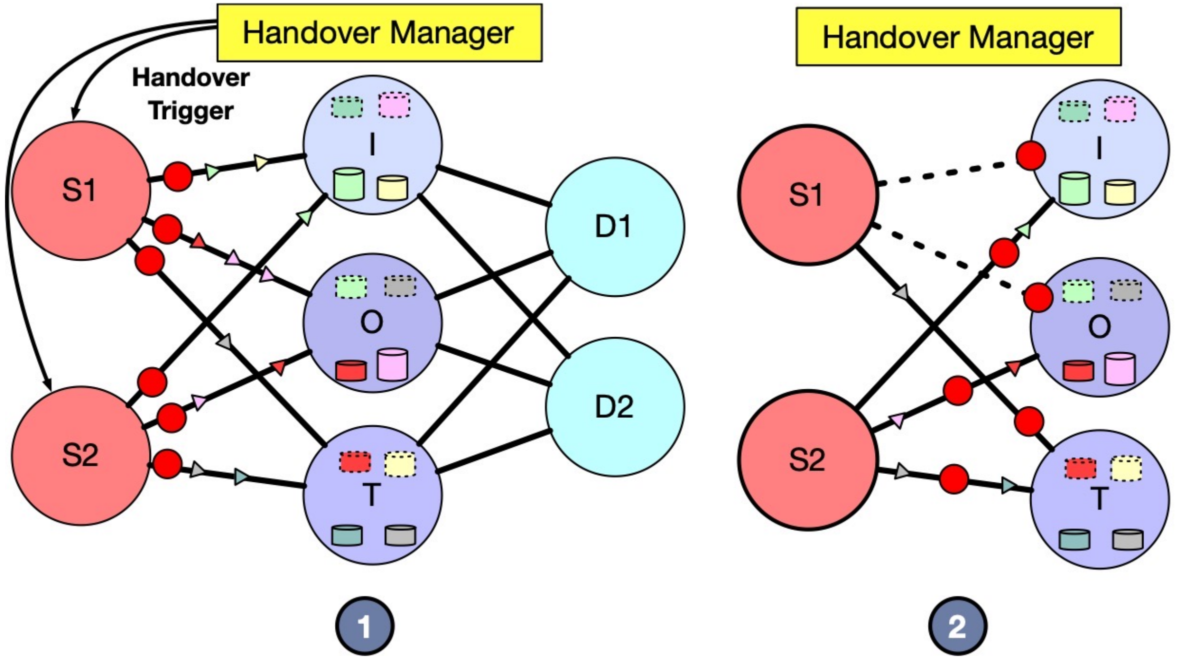
Conclusion

Rhino removes the bottleneck due to large state transfer upon a query reconfiguration

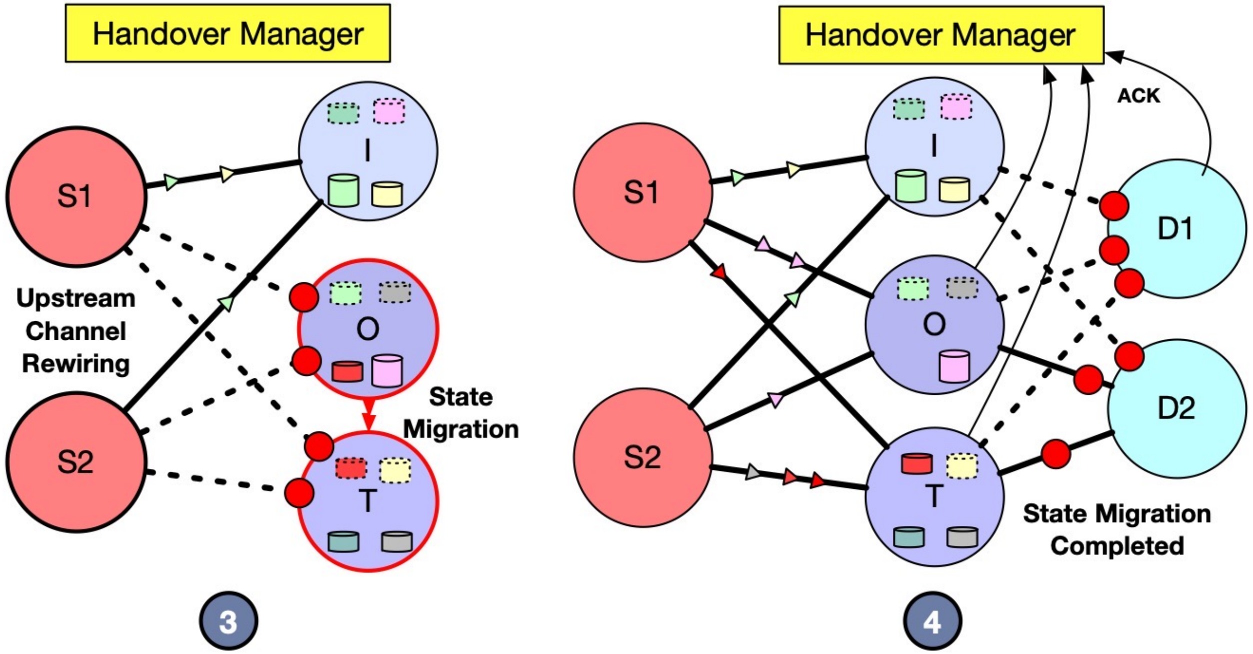
Enables fault-tolerance, resource elasticity, runtime optimizations for running stateful queries

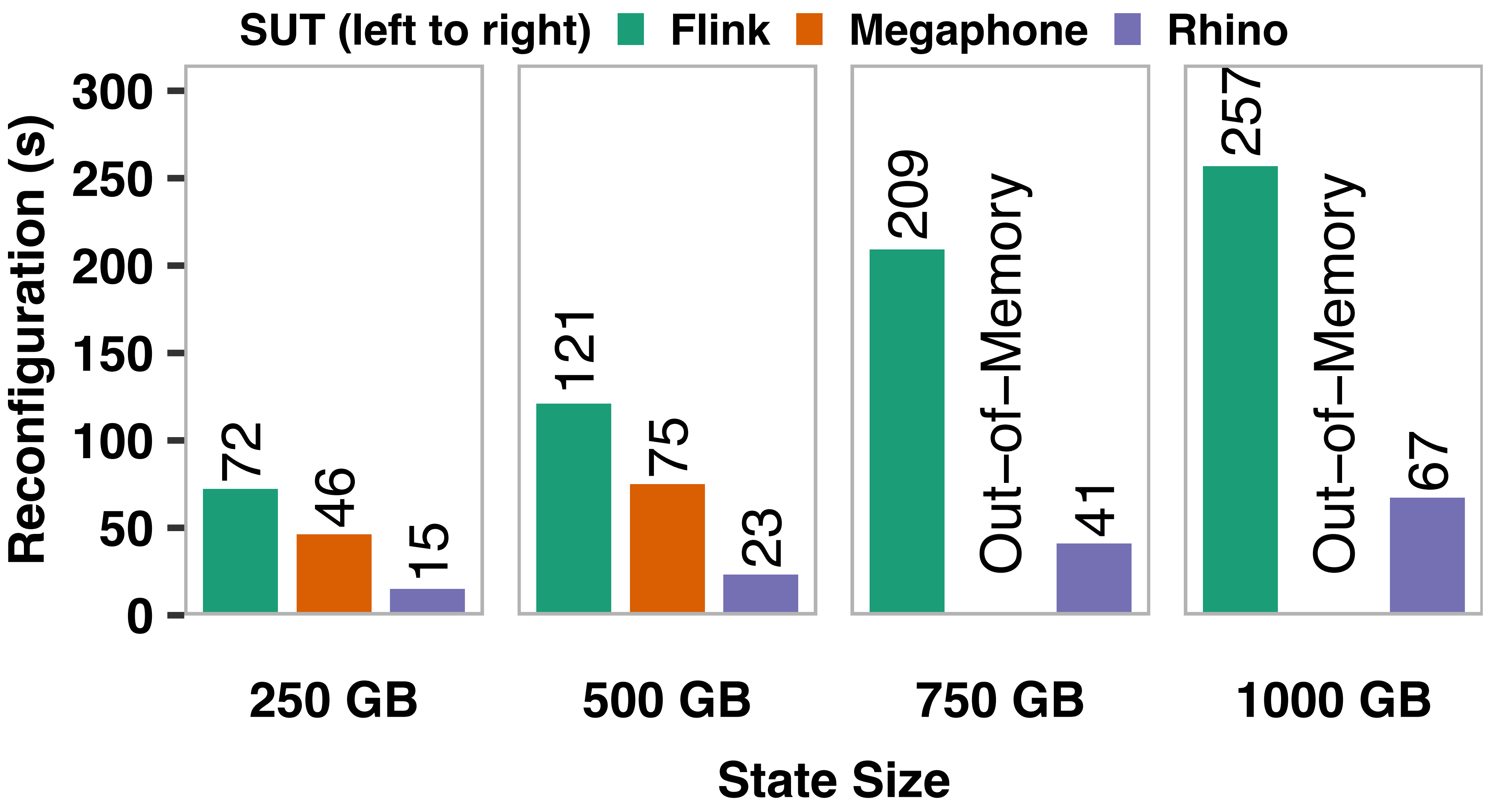
Up to 3 orders of magnitude latency reduction upon a reconfiguration

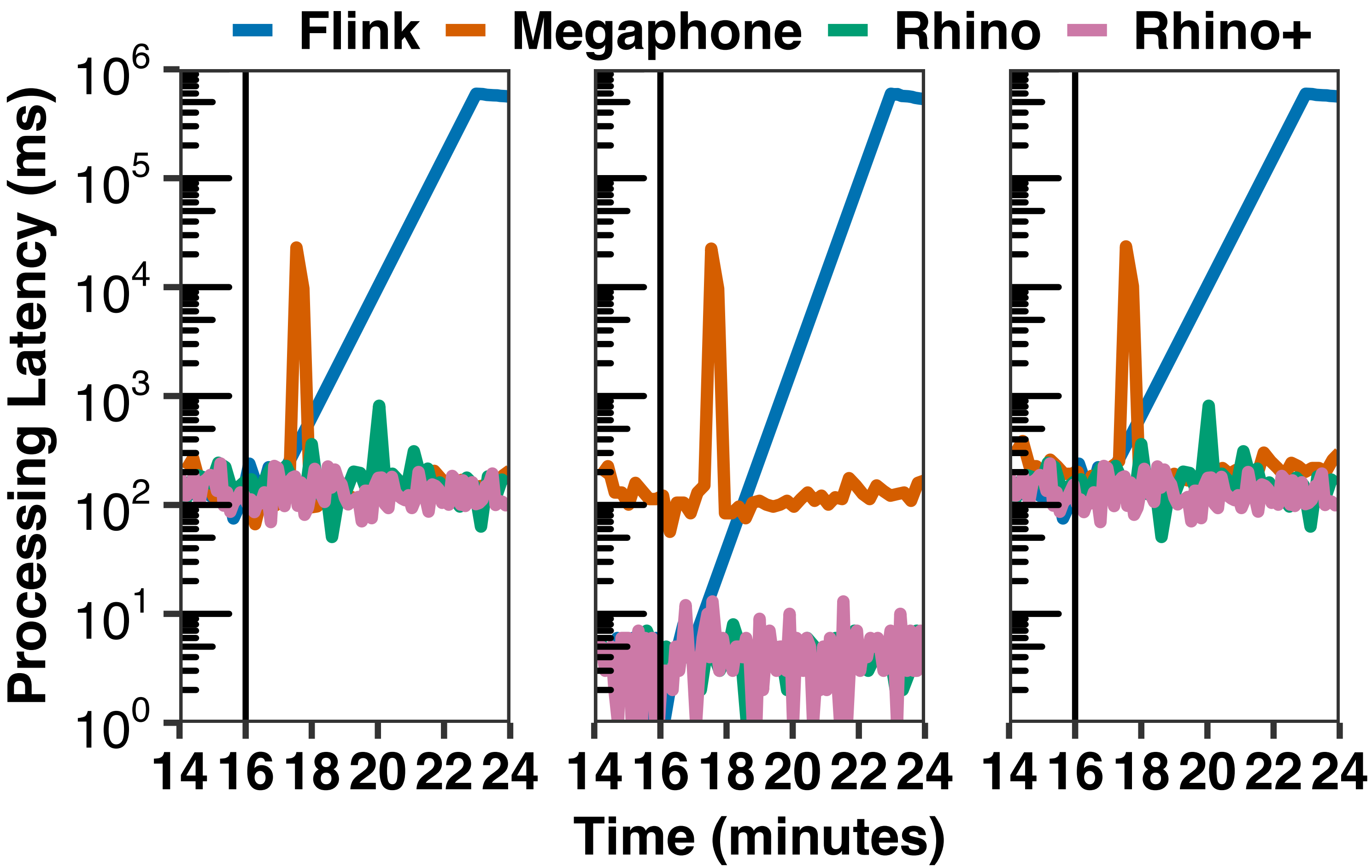
The Handover Protocol

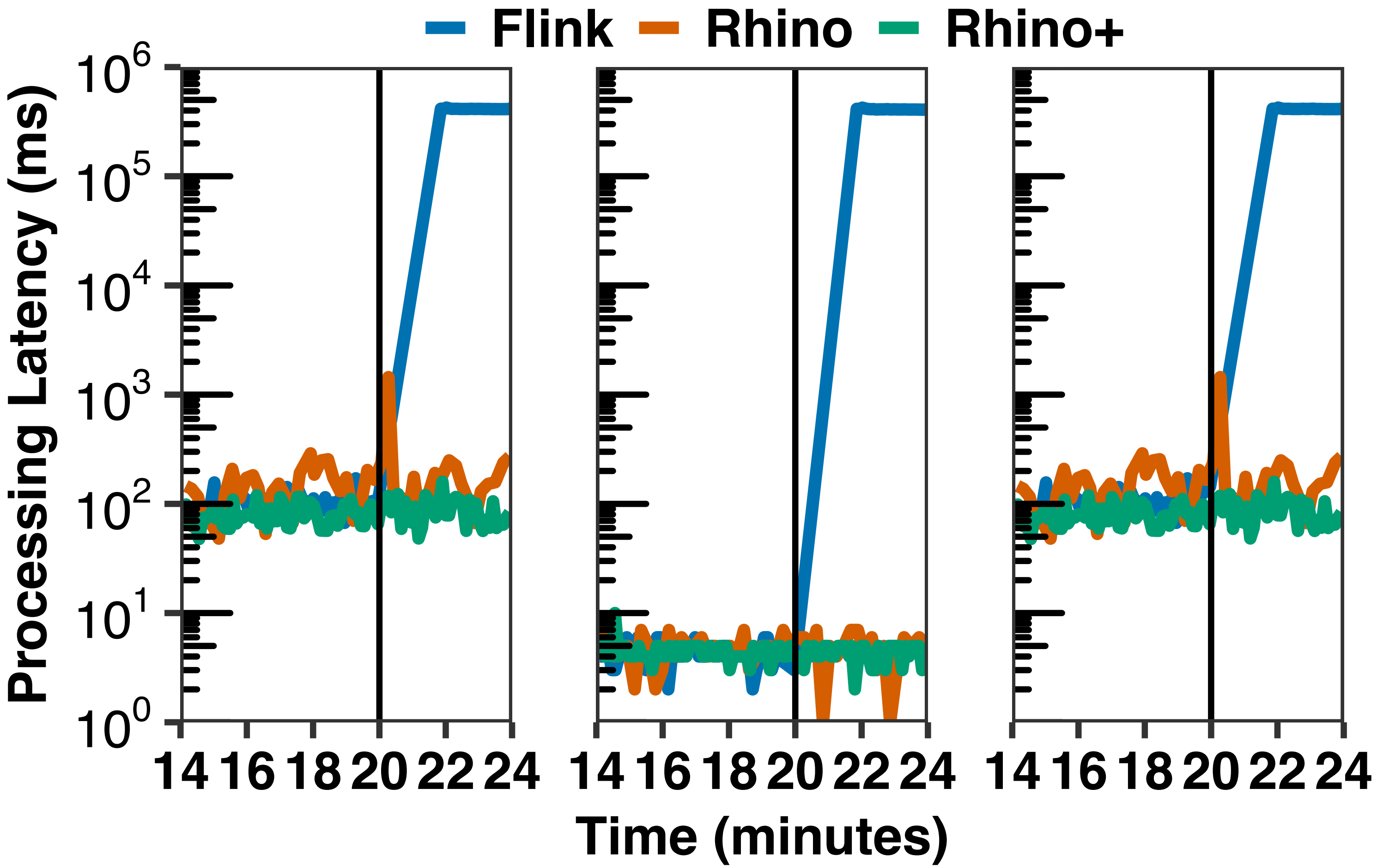


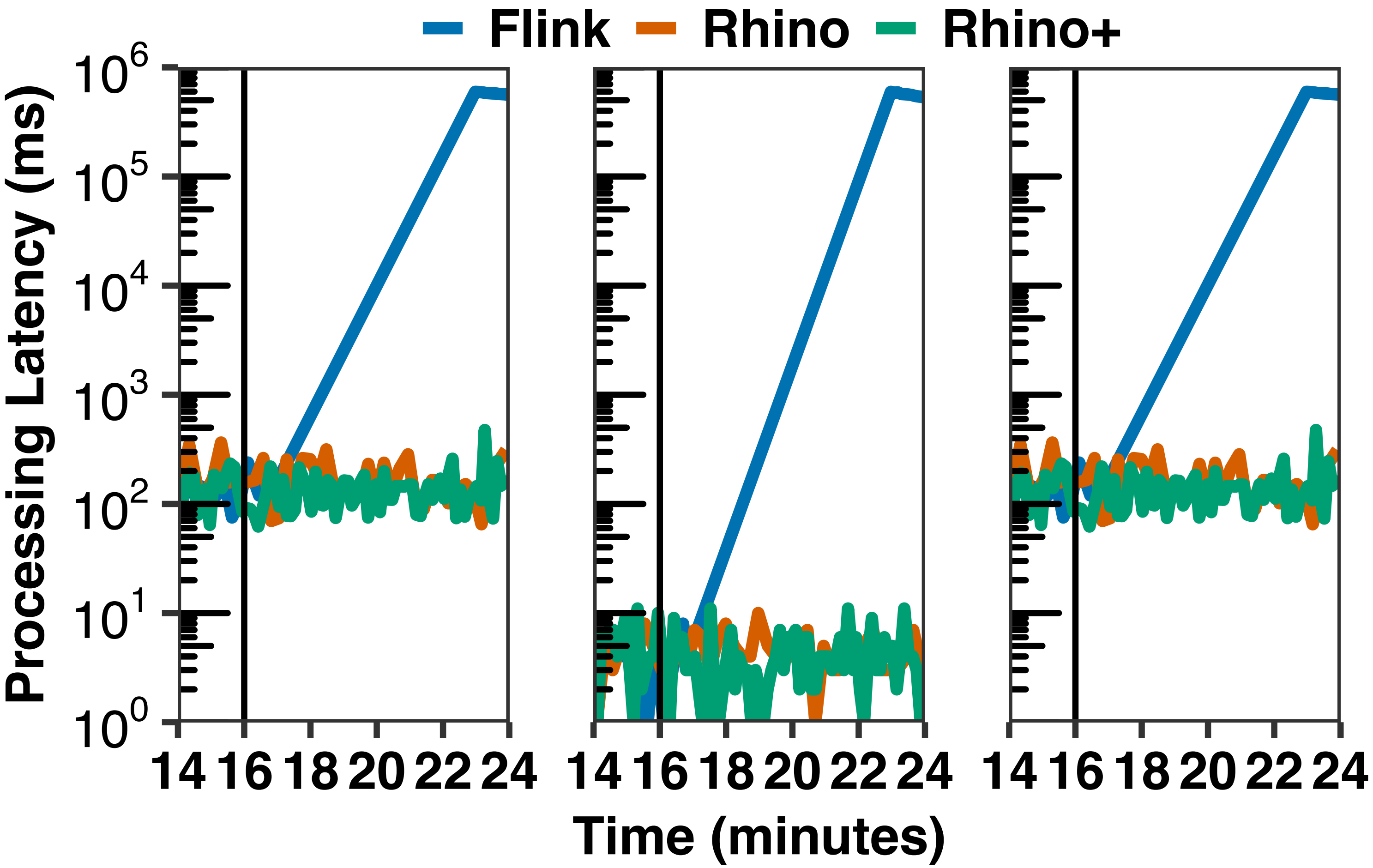
The Handover Protocol




SUT (left to right)
Flink
Megaphone
Rhino
Reconfiguration (s)
300
250
200
150
100
50
0
72
46
15
121
75
23
209
Out–of–Memory
41
257
Out–of–Memory
67
250 GB
500 GB
750 GB
1000 GB
State Size


Flink
Megaphone
Rhino
Rhino+
Processing Latency (ms)
10^0
10^1
10^2
10^3
10^4
10^5
10^6
14
16
18
20
22
24
Time (minutes)


Flink
Rhino
Rhino+
Processing Latency (ms)
10^6
10^5
10^4
10^3
10^2
10^1
10^0
14
16
18
20
22
24
Time (minutes)


Flink
Rhino
Rhino+
Processing Latency (ms)
10^6
10^5
10^4
10^3
10^2
10^1
10^0
14 16 18 20 22 24
14 16 18 20 22 24
14 16 18 20 22 24
Time (minutes)