S-Store: Streaming Meets Transaction Processing

By Meehan et al.

CS590-BDS

Thamir Qadah

Some slides contains material from the original authors' slides.

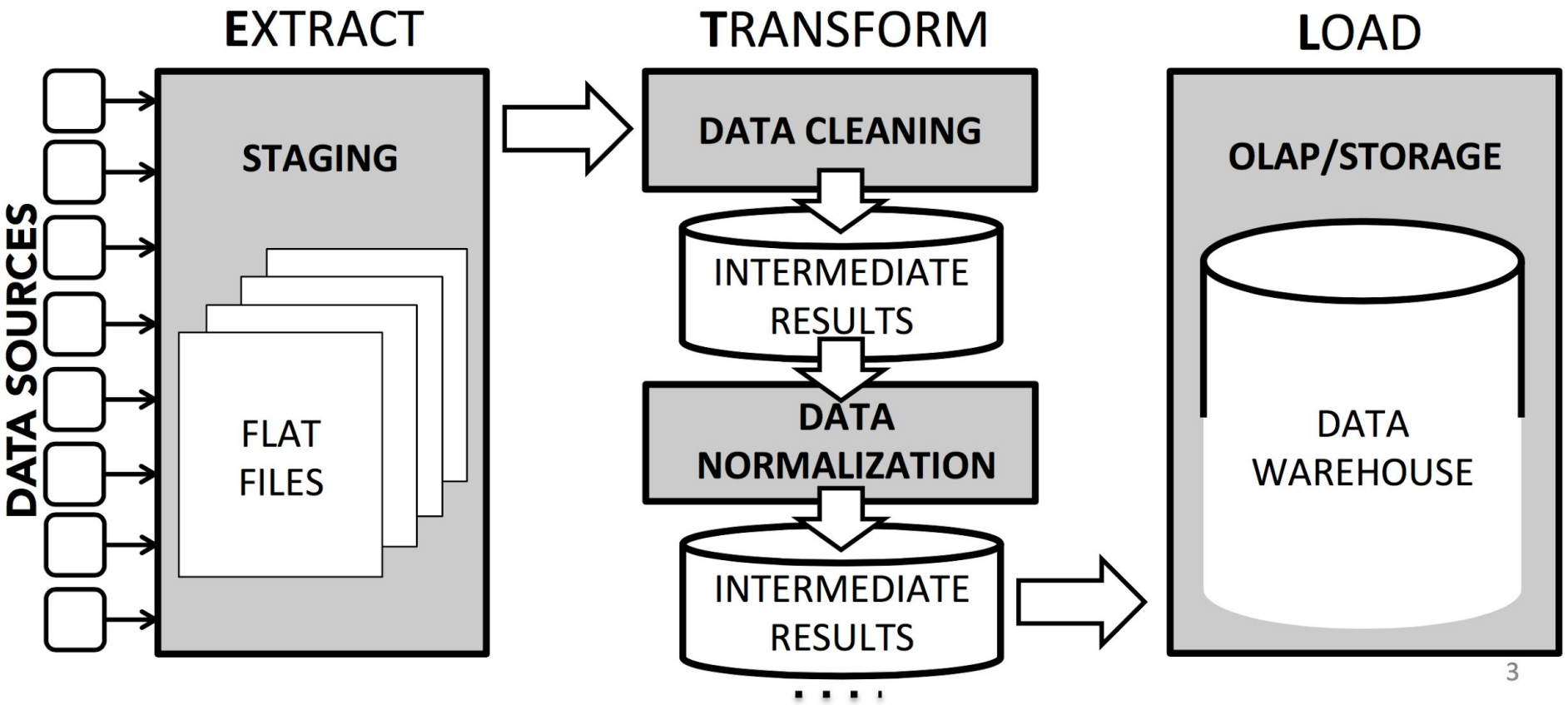
Project Website: http://sstore.cs.brown.edu/

Introduction

- What is S-Store?
  - A data processing system that combines stream processing and transaction processing.
  - Extends H-Store to support streaming semantics
- Why is it useful?
  - Traditional stream processing system: No or limited support for transactional guarantees
  - Traditional OLTP systems: No support for data-driven processing

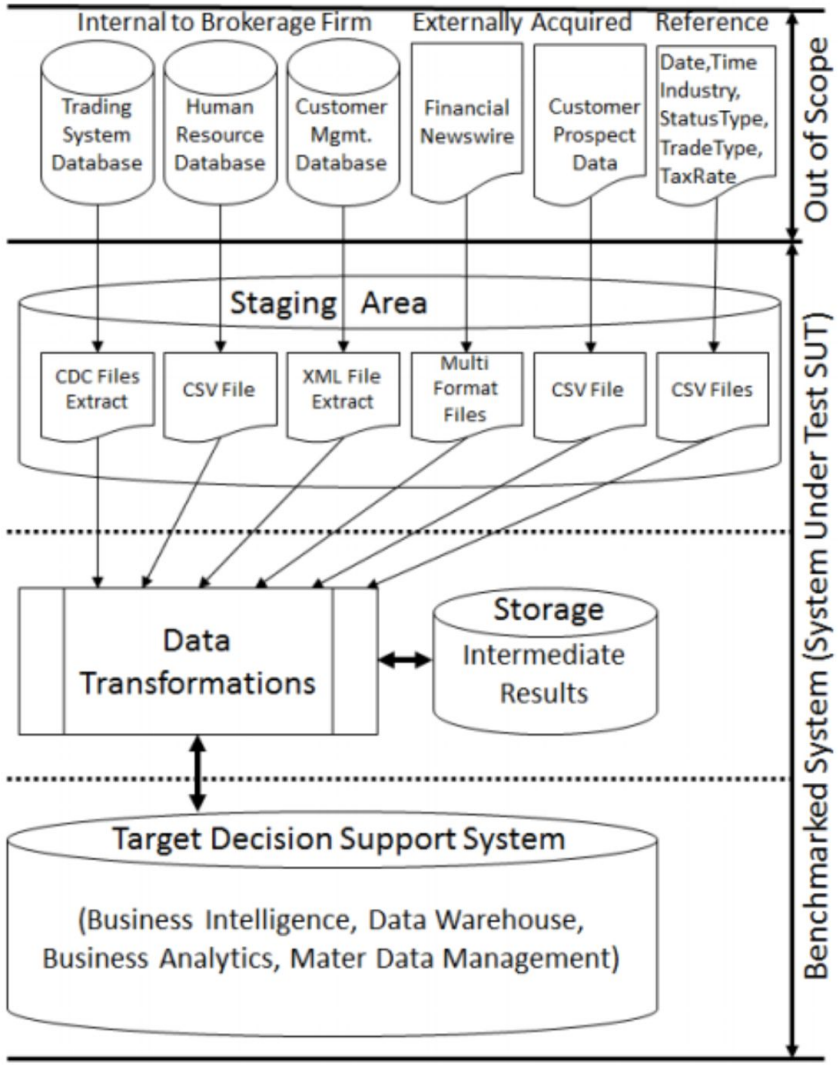
The Era of IoT

Traditional Extract-Transform-Load (ETL)



An Example: TPC-DI

- Brokerage firm
- 6 heterogeneous sources
- 3 key parts:
  1. Ingest raw data
  2. ETL transform
  3. Update warehouse



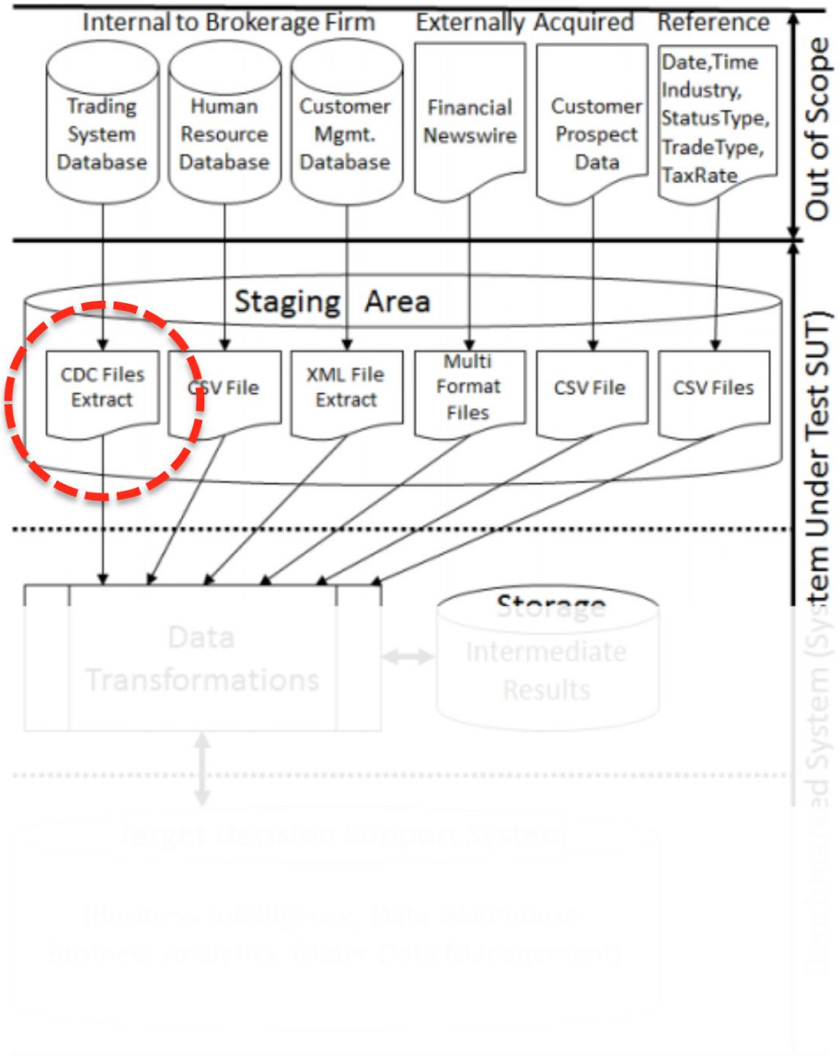
TPC Transaction Processing Performance Council

http://www.tpc.org/tpcdi/
Poess et al, VLDB 2014

An Example: TPC-DI

- Brokerage firm
- 6 heterogeneous sources

- 3 key parts:
  1. Ingest raw data



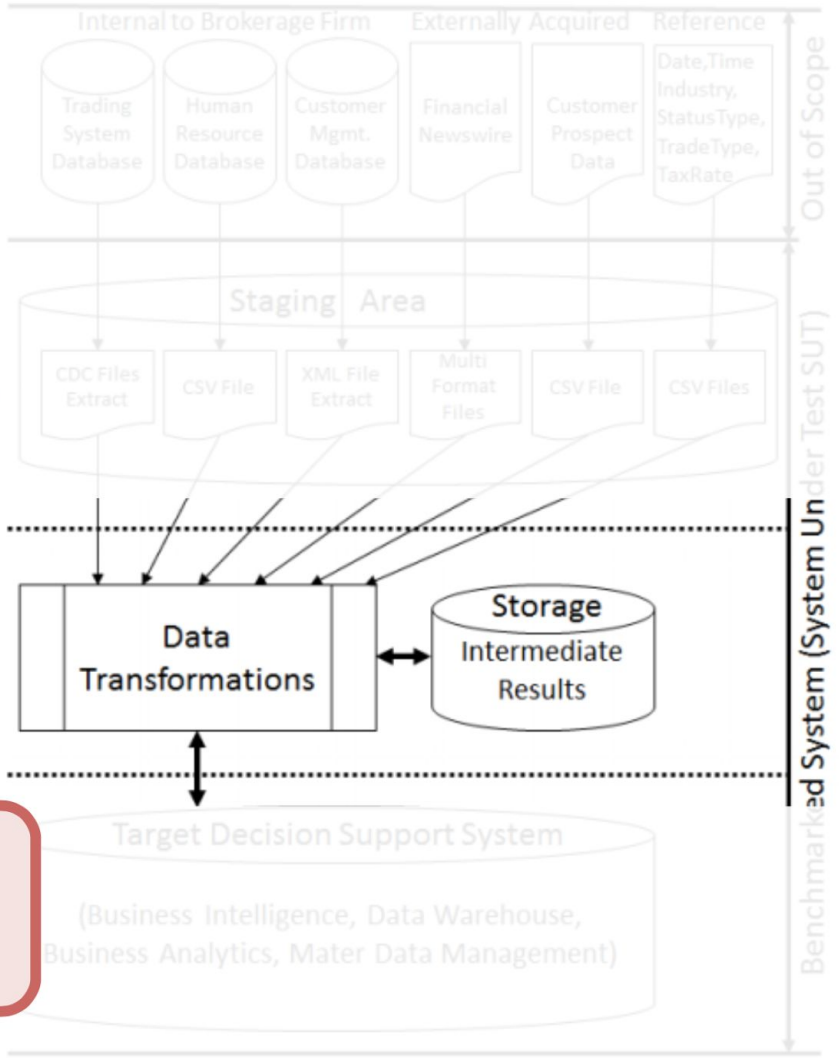
- ✓ Data collected into flat files
- ✓ Heterogeneous data types
- ✓ Incremental update from an OLTP source, once a day

TPC Transaction Processing Performance Council

http://www.tpc.org/tpcdi/
Poess et al, VLDB 2014.

An Example: TPC-DI

- Brokerage firm
- 6 heterogeneous sources
- 3 key parts:
  1. Ingest raw data
  2. ETL transform
  3. Update warehouse



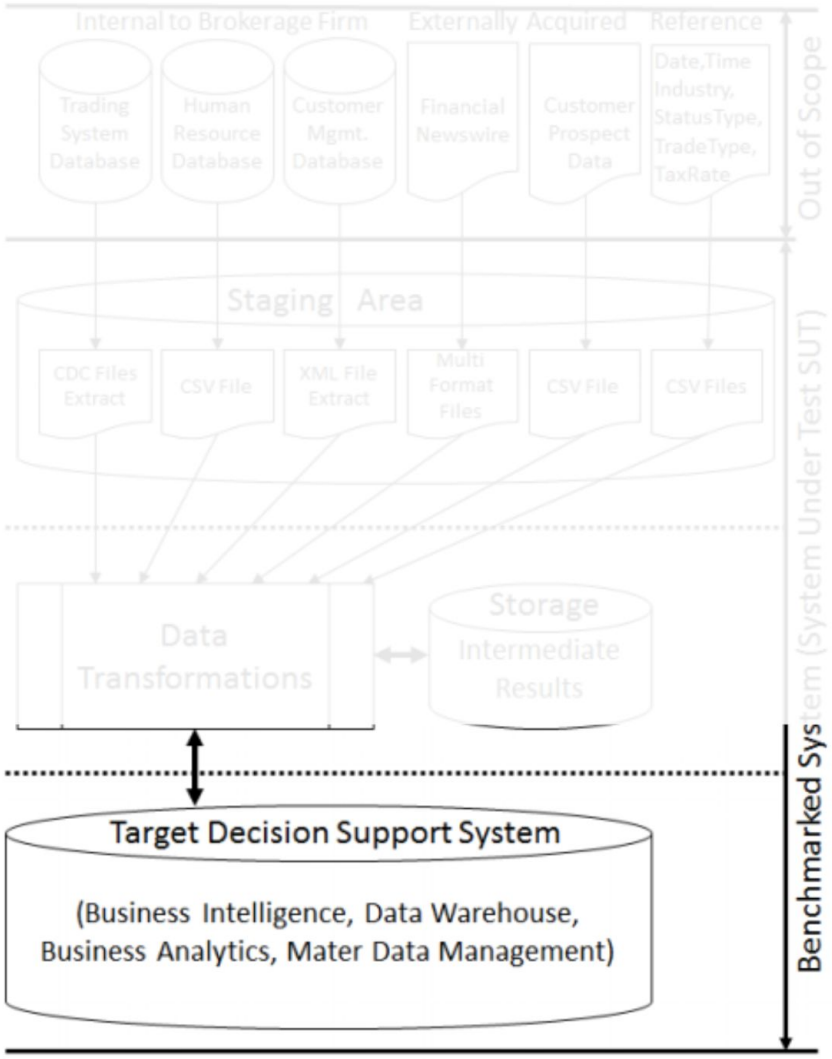
- ✓ Storage for intermediate results
- ✓ Transactional state management

Poess et al, VLDB 2014.

An Example: TPC-DI

- Brokerage firm
- 6 heterogeneous sources
- 3 key parts:
  1. Ingest raw data
  2. ETL transform
  3. Update warehouse

✓ Bulk loading



TPC Transaction Processing Performance Council

http://www.tpc.org/tpcdi/
Poess et al, VLDB 2014.

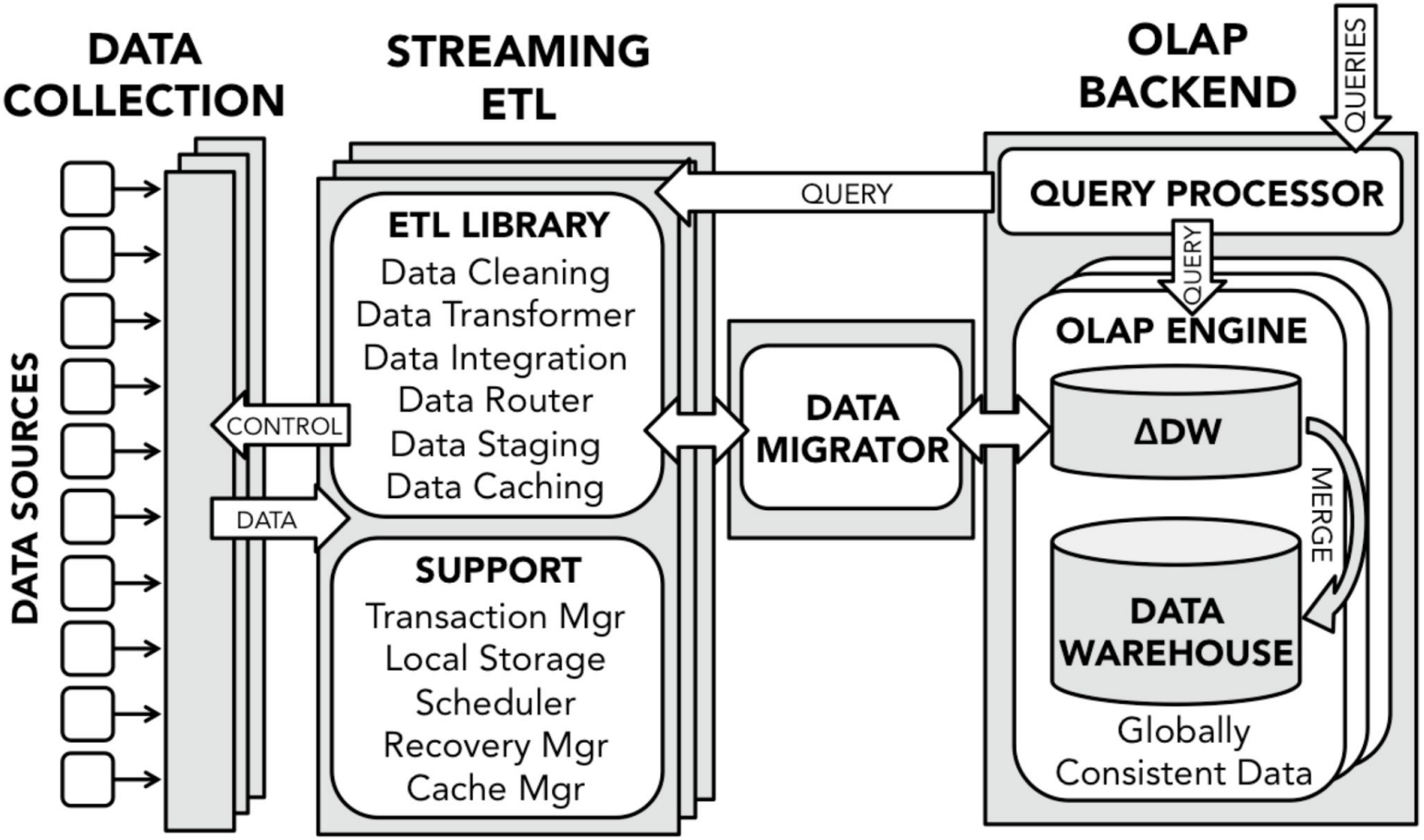
Streaming Data Ingestion

- In modern apps such as IoT:
  - real-time streams of data from a large number of sources
  - majority of these sources report in the form of time-series
  - data currency & low latency is key for real-time decision making & control

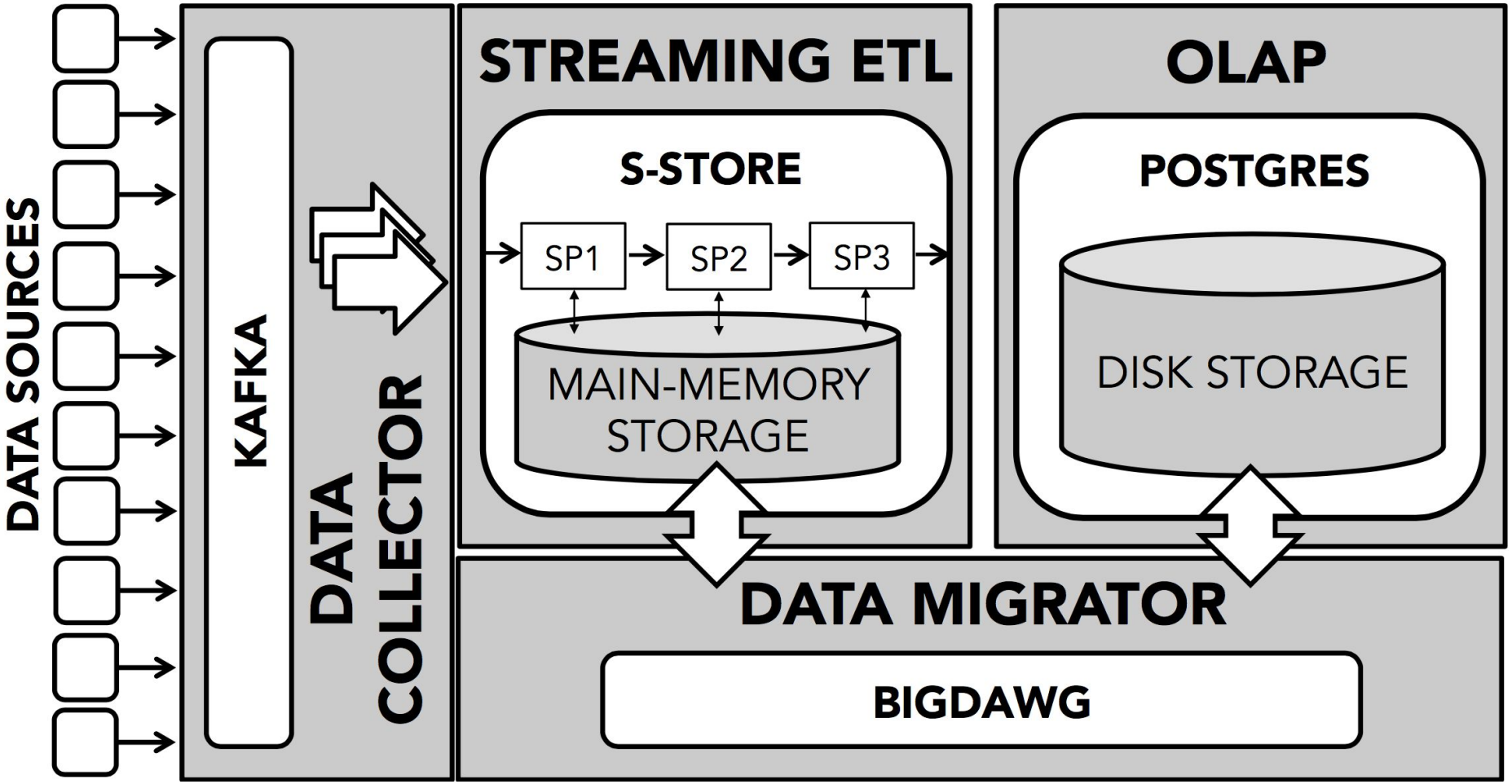
✓ Need a stream-based ingestion architecture

✓ Must pay attention to time-series data type and operations (both during ingestion & analytics)

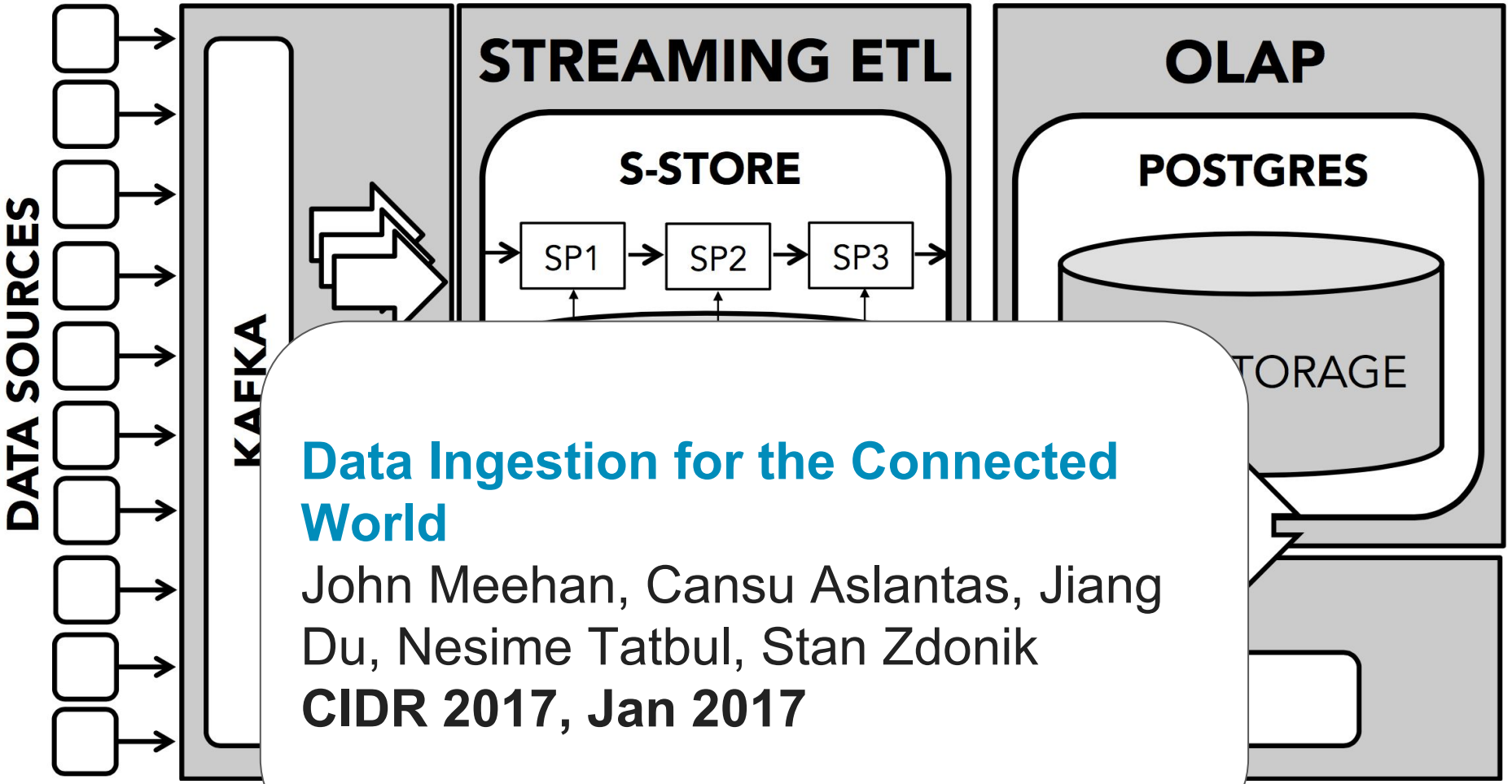
An Architecture for Streaming Data Ingestion



S-Store in BIGDAWG



S-Store in BIGDAWG



Data Ingestion for the Connected World

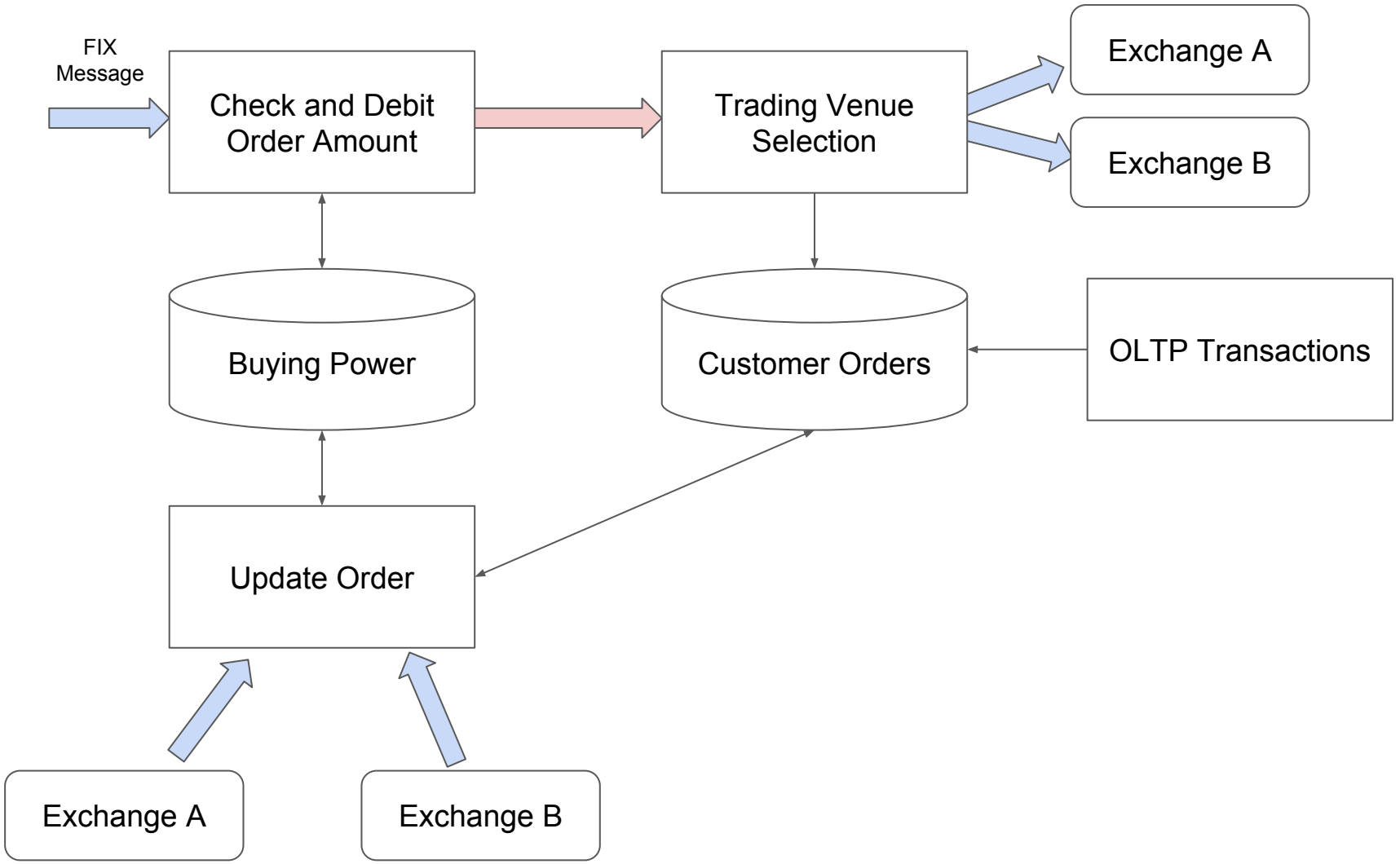
John Meehan, Cansu Aslantas, Jiang Du, Nesime Tatbul, Stan Zdonik

CIDR 2017, Jan 2017

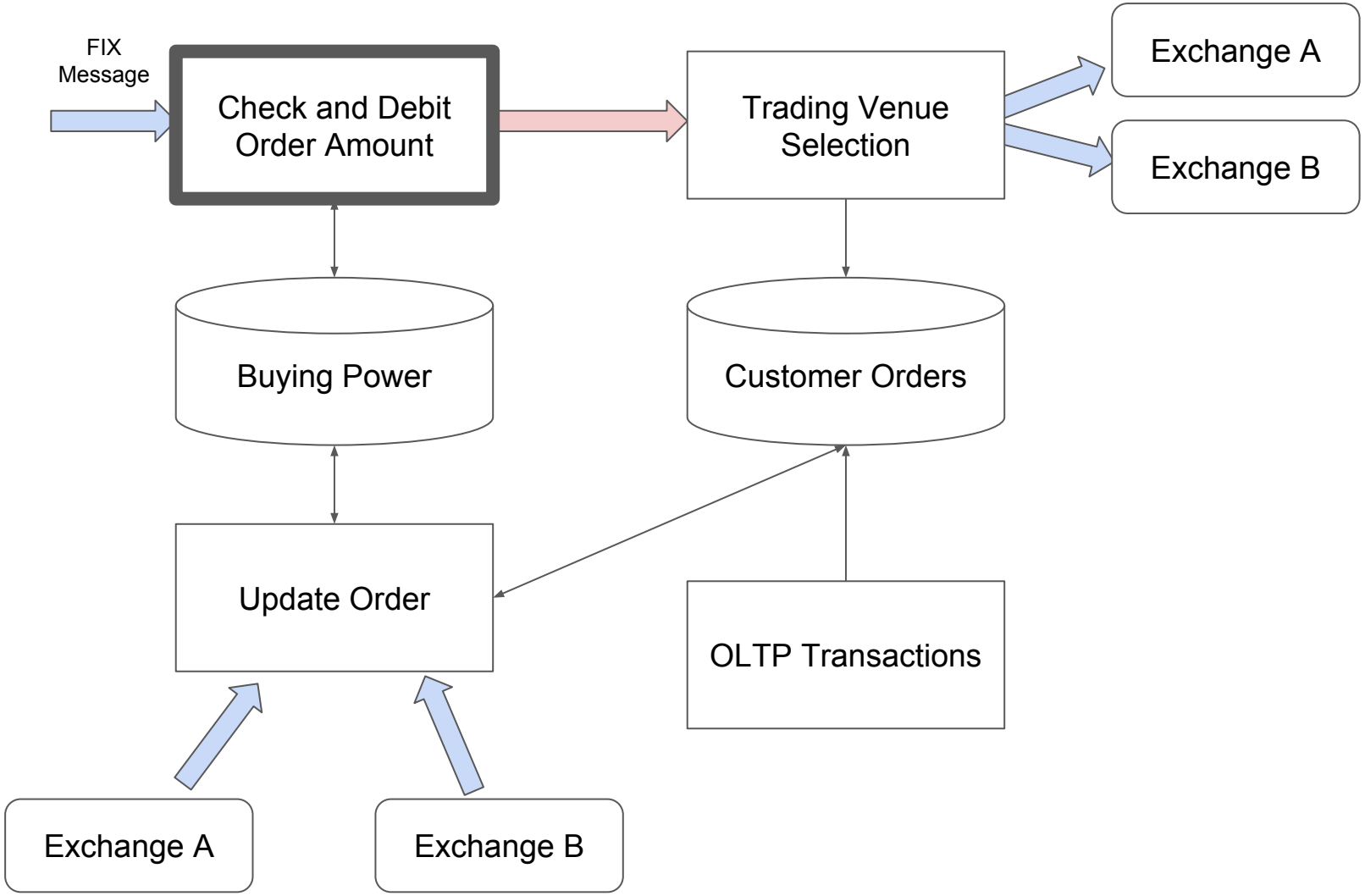
Smart Order Routing (SOR) Application

- Same stocks can be traded at different trading venues independently
- A SOR systems takes the client order, and routes it to the venue what provides the most benefit the client.

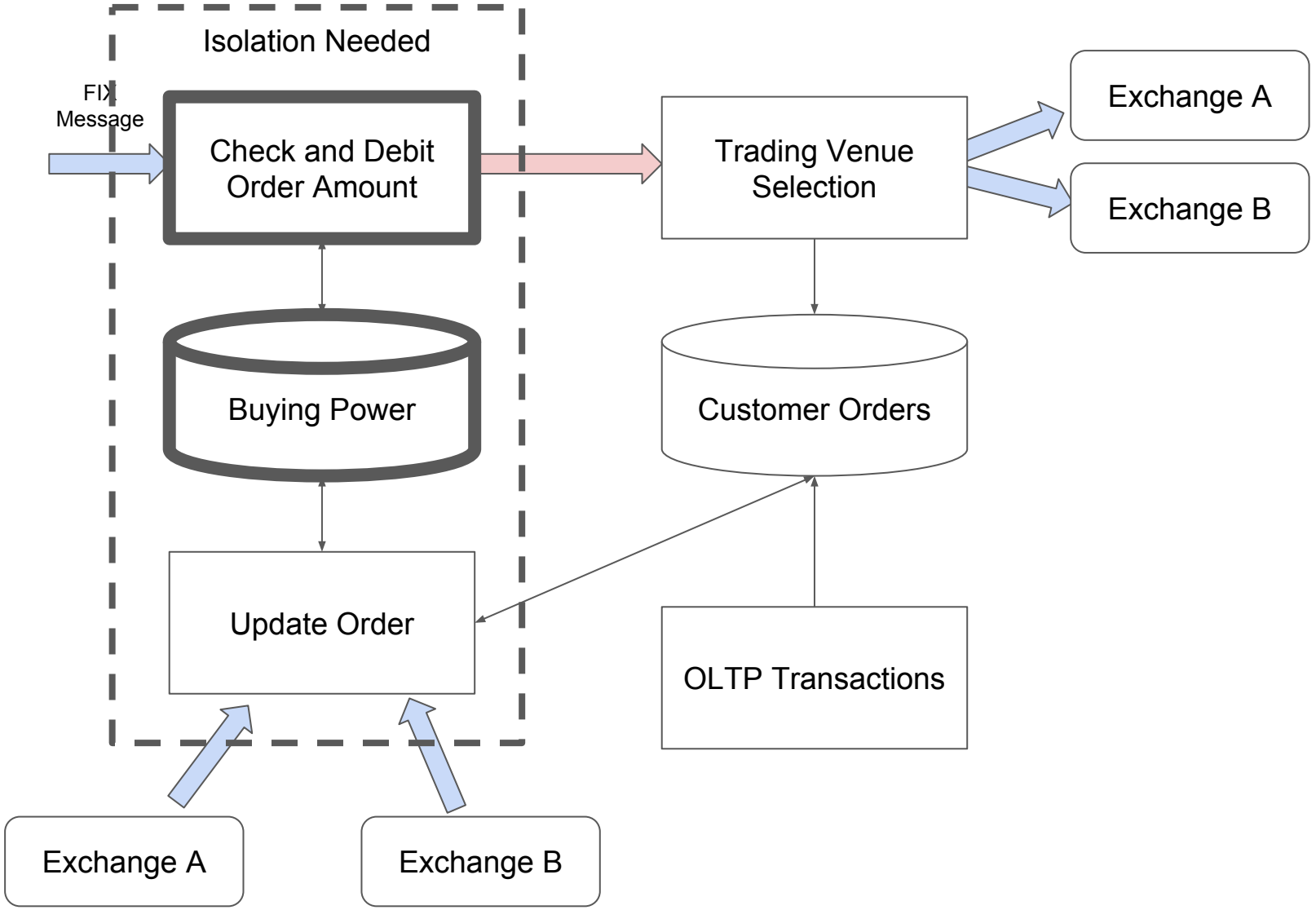
FIX trading Example



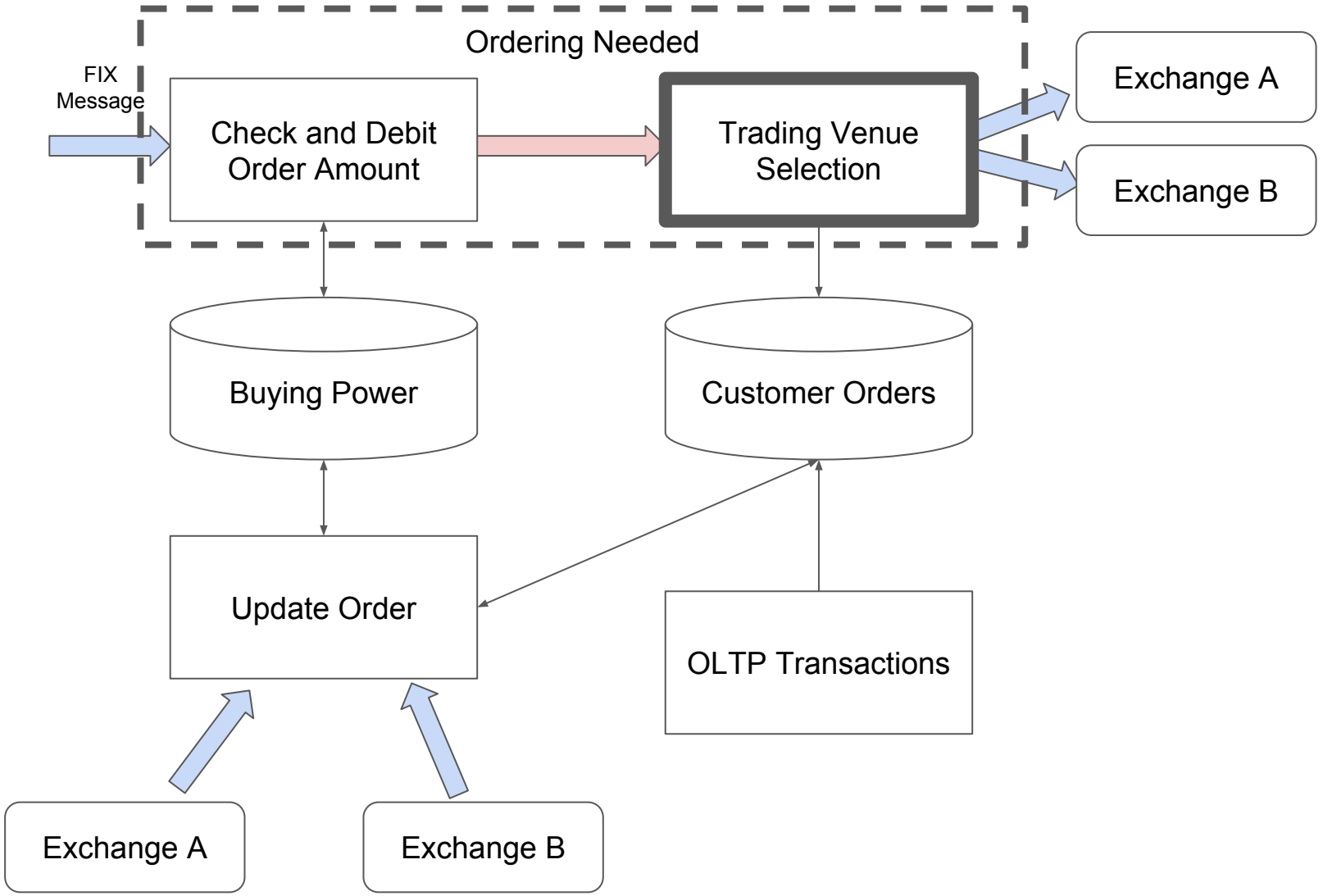
FIX trading Example



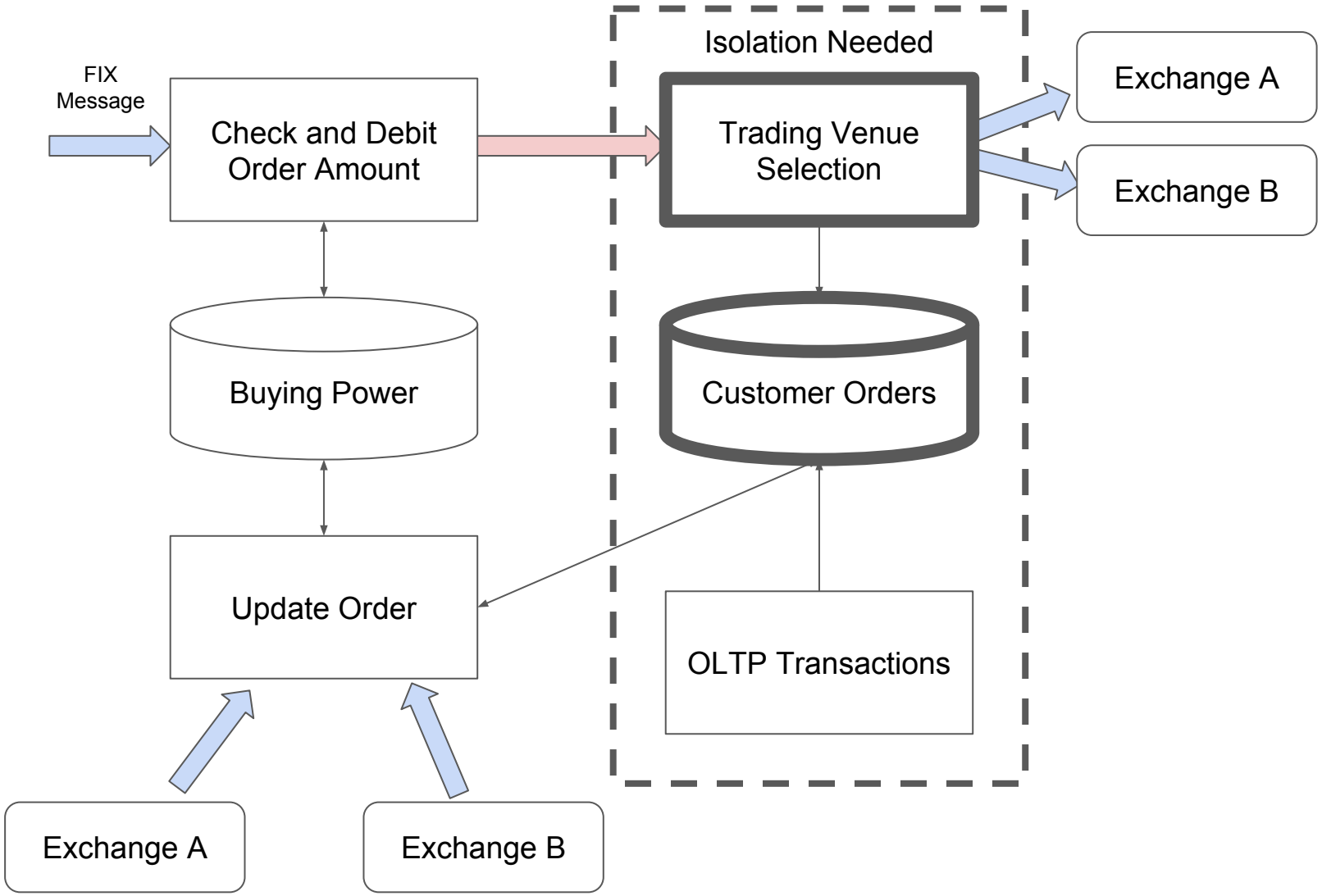
FIX trading Example



FIX trading Example



FIX trading Example



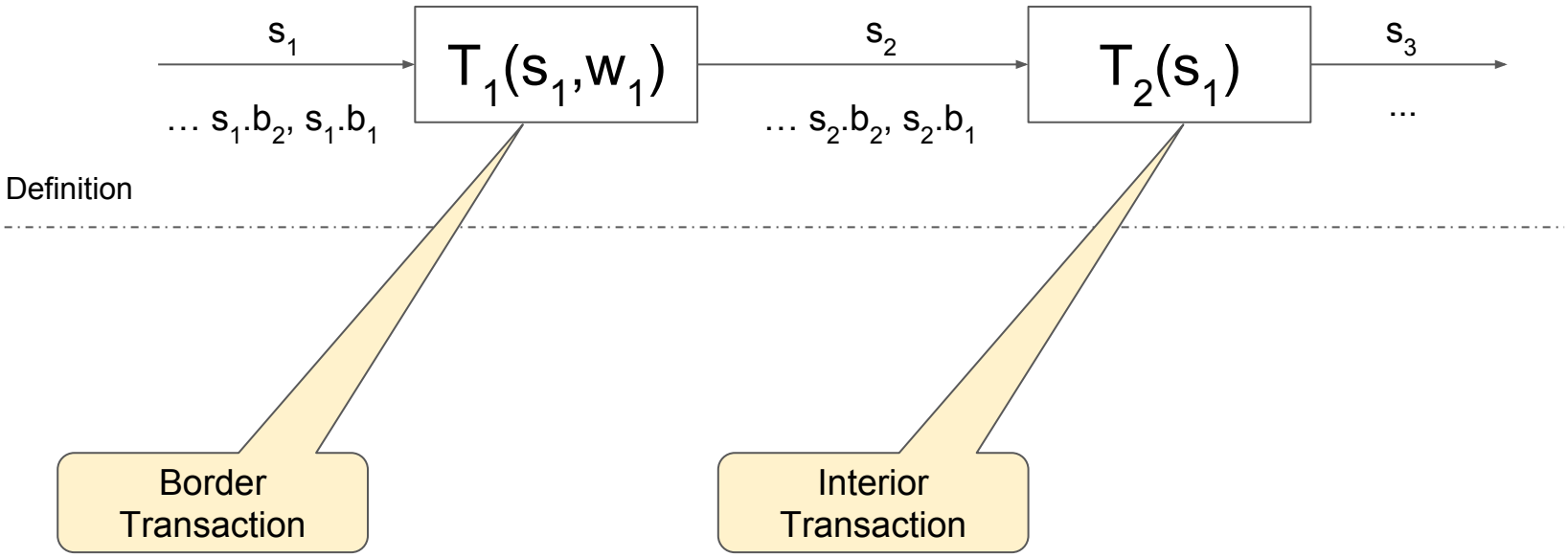
The Computational Model

- Guarantees:
  - ACID guarantees for OLTP and Streaming
  - Ordered Execution guarantees
    - Executions follow the dataflow graph for streaming transactions
  - Exactly once processing guarantees for streams
    - No loss or duplication
- 3 kinds of states:
  - Public tables
  - Windows
  - Streams
- 2 kinds of transactions:
  - **OLTP transactions**: can only access public tables
  - **Streaming transactions**: can access all kinds of state

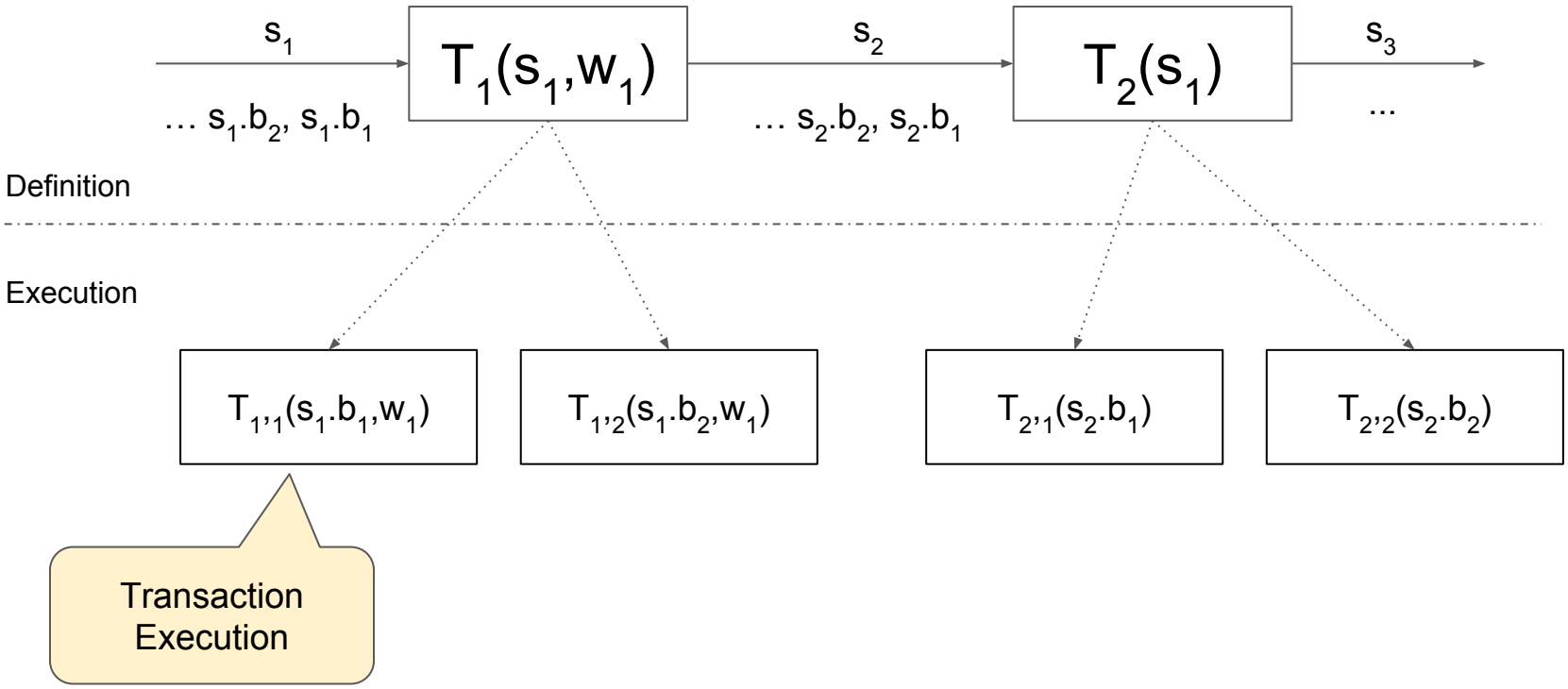
Data and Processing Models

- A stream is an **ordered** collection of tuples
- Each tuple is associated with a batch-id (e.g. timestamp) that specifies the simultaneity and ordering
- Streaming transactions operates on non-overlaping **atomic batches** of tuples.
- An atomic batch is a finite contiguous subsequence of a stream
  - External to a streaming transaction
- A window is finite contiguous subsequence of a stream
  - Internal to a streaming transaction
  - Have a slide parameter => (sliding window)
  - If slide == window size => (tumbling window)
- Data-driven execution represented as a **dataflow** (DAG) with nodes representing streaming transactions and edges represent the flow of data among nodes.

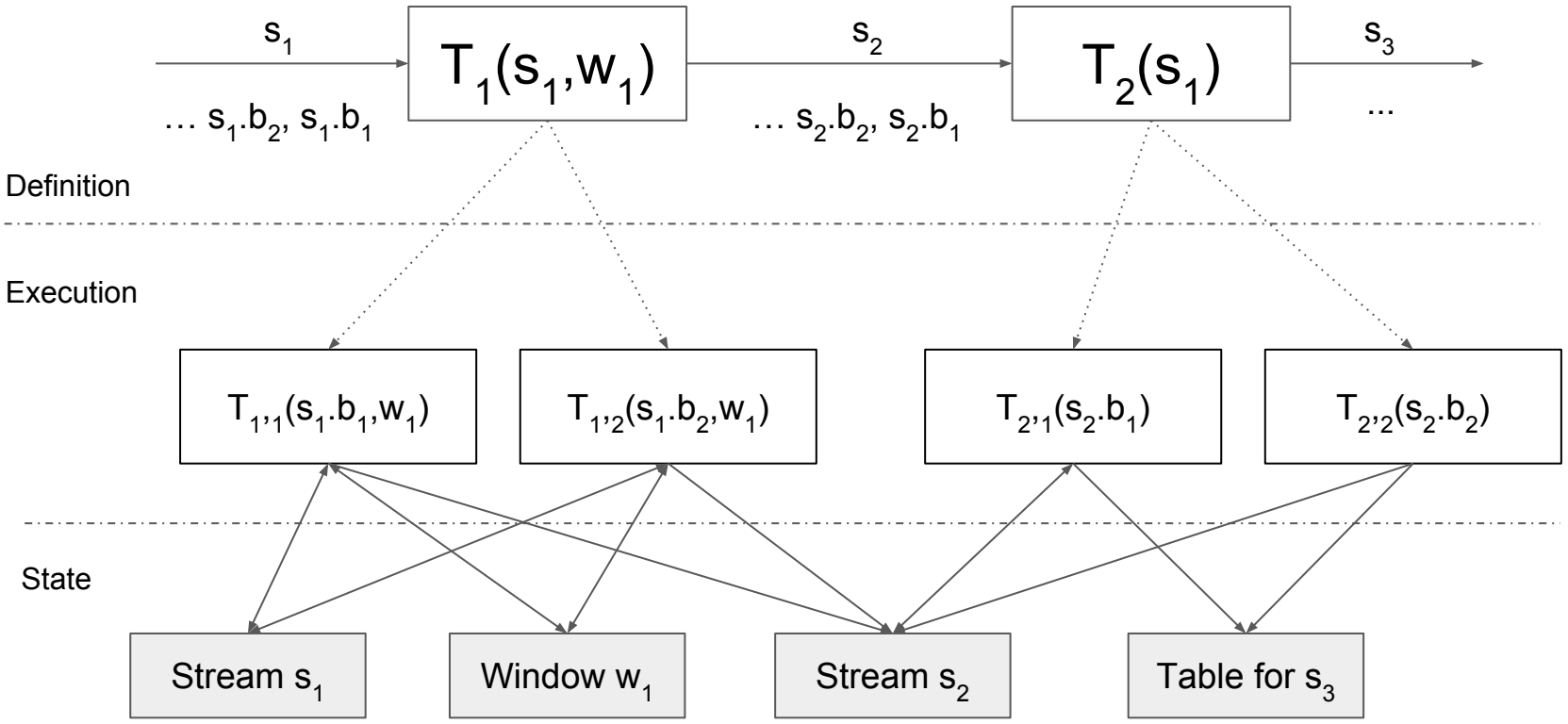
Abstract Example



Abstract Example



Abstract Example



Correct Execution

- A dataflow graph is executed in rounds of atomic batches.
- Unlike traditional ACID, the execution is constrained by:
  - DAG order constraint
  - Stream order constraint
- In hybrid workloads, an OLTP transaction $T_{i,j}(p_i)$ can be interleave anywhere in the schedule.
- Nested transactions can only commit if all of its sub-transactions commit.

Fault Tolerance

- S-Store must be able to recover its state.
- Exactly once processing guarantees is limited to internal state only
- Strong recovery:
  - Uses command-log for committed transactions
  - Replay commands to restore states
  - Limitation: cannot guarantee same results if non-determinism exist in transaction logic
- Weak Recovery:
  - Perform command logging for border transactions only.
  - Assumes the ability to replay input data streams.

S-Store Architecture

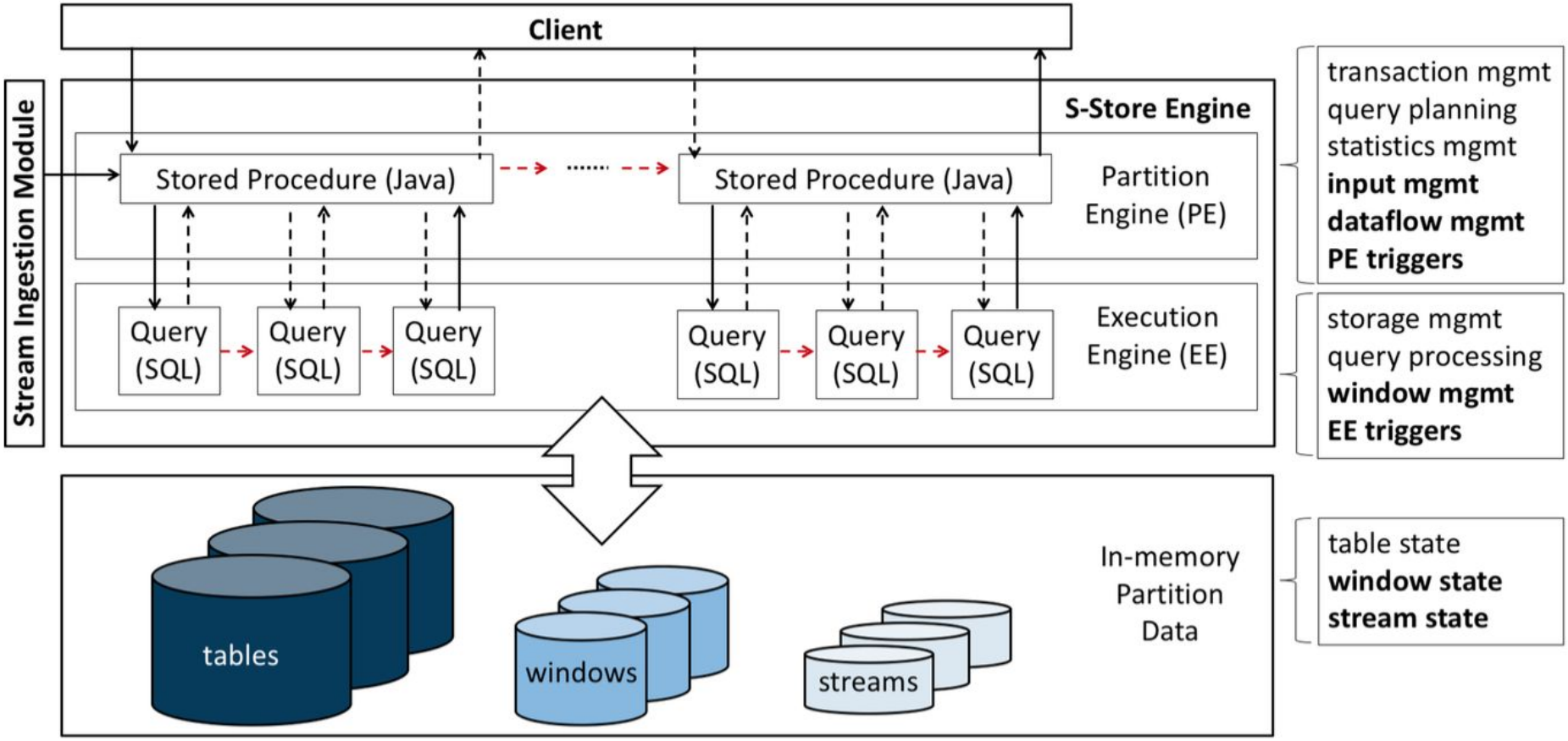


Figure 4: S-Store Architecture

Stream Implementation

Stream 1

TS	A1	A2

$T_1(s_1)$

Stream 2

TS	A3	A4

Stream Implementation

Stream 1

TS	A1	A2
1	...	...
2	...	...

$T_1(s_1)$

Stream 2

TS	A3	A4

Batch $s_1.b_1$ is ready

Stream Implementation

Stream 1

TS	A1	A2
1	...	...
2	...	...

$T_1(s_1)$

$T_{1,1}(s_1.b_1)$

Stream 2

TS	A3	A4

$T_{1,1}$ is scheduled

Stream Implementation

Stream 1

TS	A1	A2
1	...	...
2	...	...
3	...	...
4	...	...

$T_1(s_1)$

$T_{1,1}(s_1.b_1)$

$T_{1,2}(s_1.b_1)$

Stream 2

TS	A3	A4
1	...	...
2	...	...

s_1,b_2 is ready, $T_{1,2}$ is scheduled, $T_{1,1}$ produces output

Stream Implementation

Stream 1

TS	A1	A2
3	...	...
4	...	...

$T_1(s_1)$

$T_{1,1}(s_1.b_1)$

$T_{1,2}(s_1.b_1)$

Stream 2

TS	A3	A4
1	...	...
2	...	...
3	...	...
4	...	...

s_1,b_2 is ready, $T_{1,2}$ is scheduled, $T_{1,1}$ commits

Stream Implementation

Stream 1

TS	A1	A2

$T_1(s_1)$

$T_{1,1}(s_1.b_1)$

$T_{1,2}(s_1.b_1)$

Stream 2

TS	A3	A4
1	...	...
2	...	...
3	...	...
4	...	...

$T_{1,2}$ commits

Experiments

- Single core deployment for data access
- Single core client
- Batch size = 1 tuple
- System comparison used leaderboard benchmark
- Microbenchmarks were used to evaluate triggers and recovery mechanisms

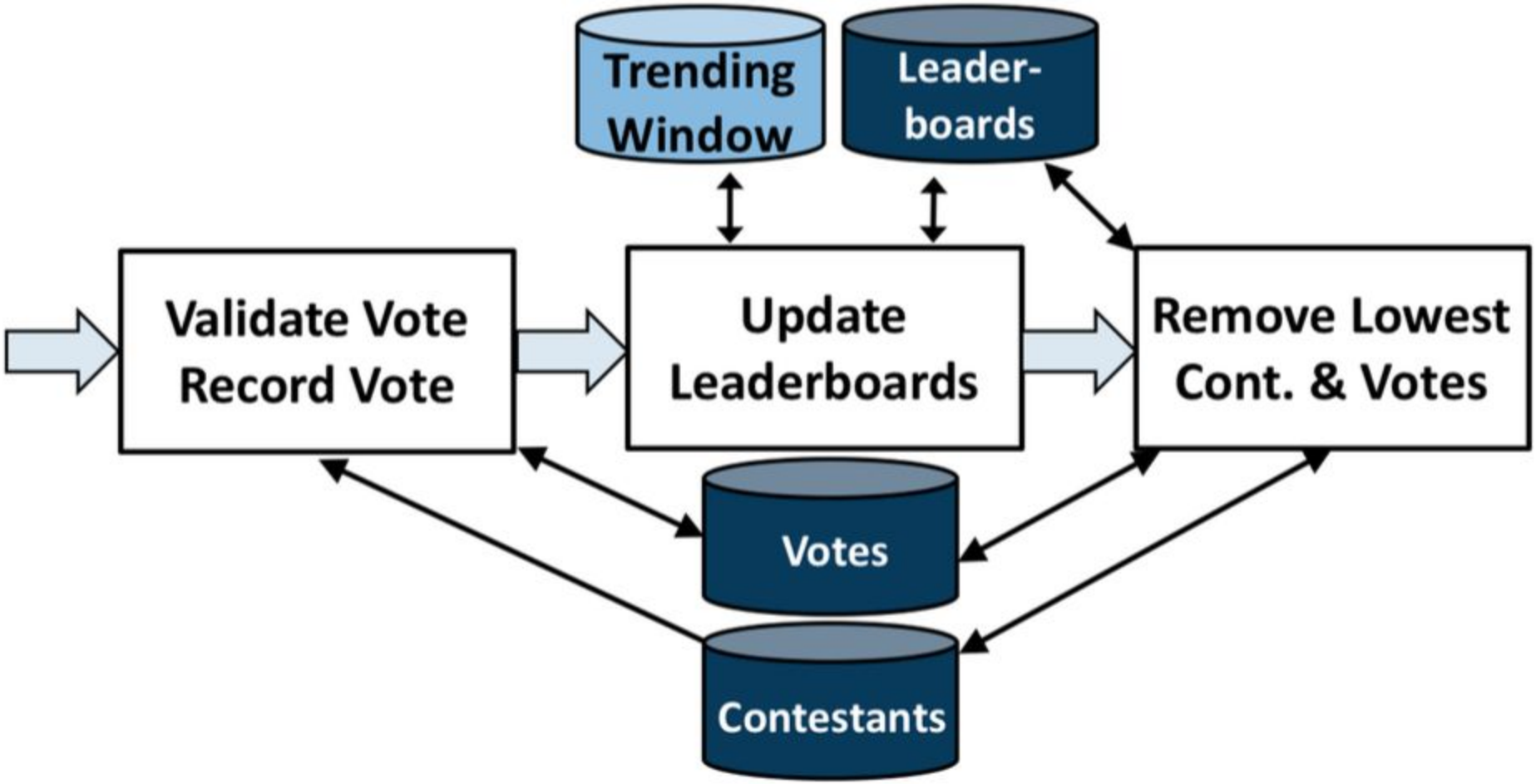
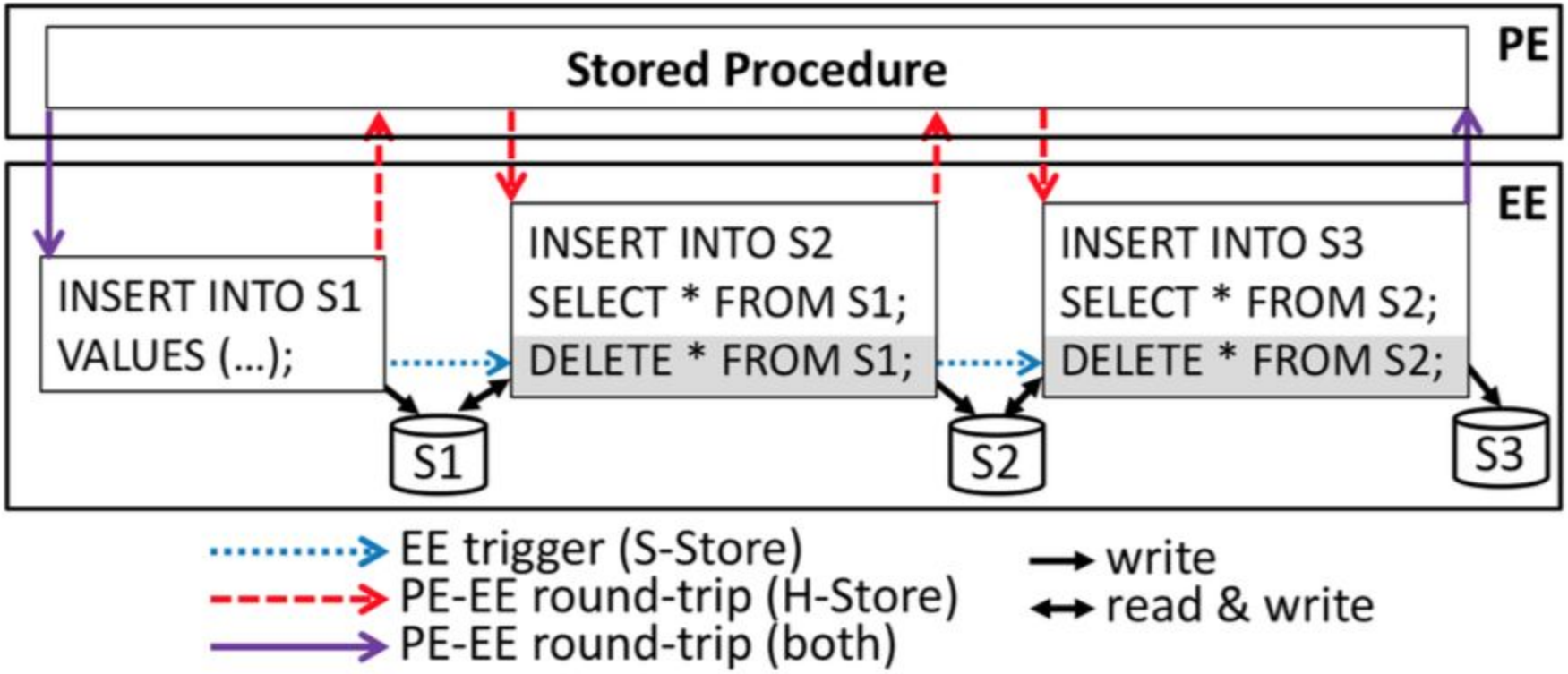


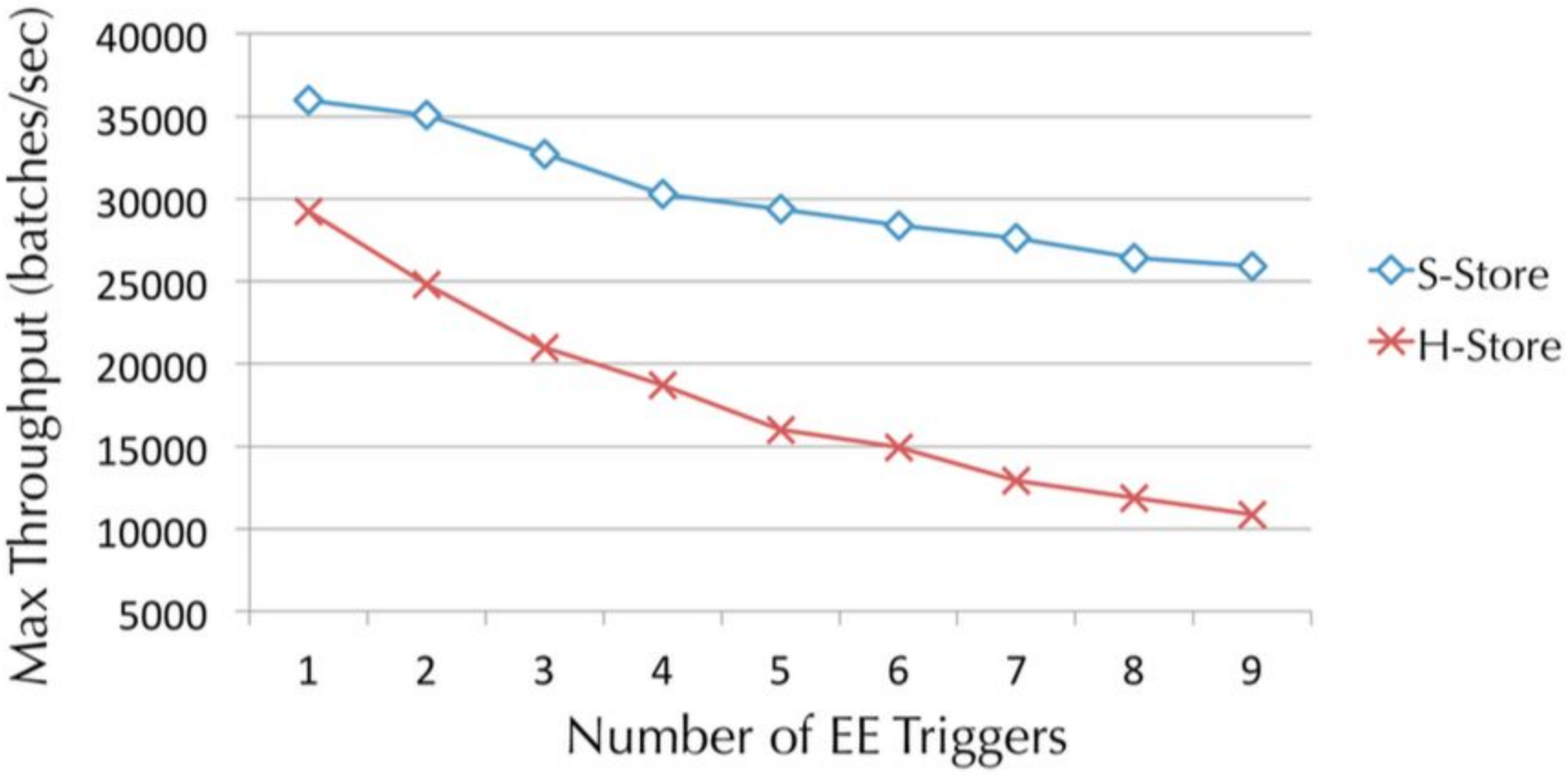
Figure 5: Leaderboard Maintenance Benchmark

System	ACID	Order	Exactly-Once	Max Tput (batches/sec)
H-Store (async)	✓	×	×	5300
H-Store (sync)	✓	✓	×	210
Esper+ VoltDB	✓	✓	×	570
Storm+ VoltDB	✓	✓	✓	600
S-Store	✓	✓	✓	2200

Table 1: Guarantees vs Max Tput (Leaderboard Maintenance)

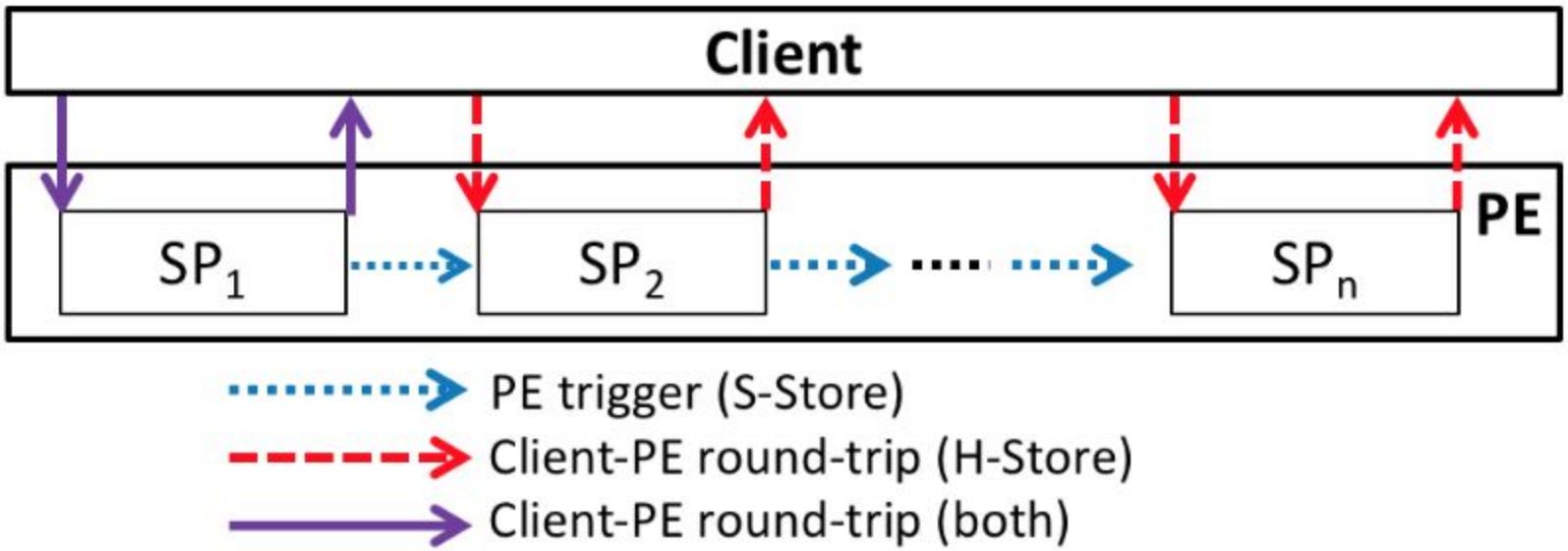


(a) EE Trigger Micro-Benchmark

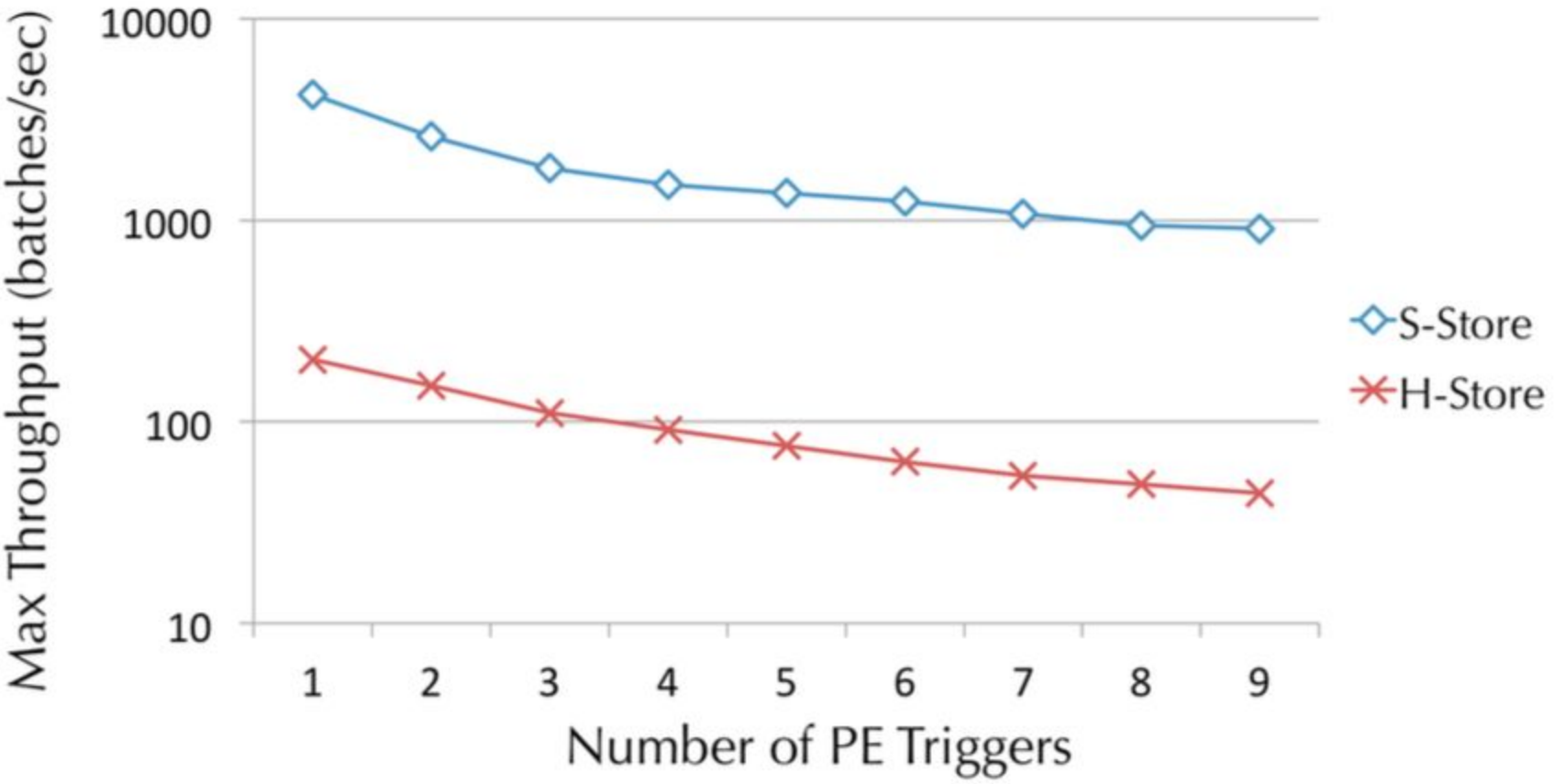


(b) EE Trigger Result

Figure 6: Execution Engine Triggers



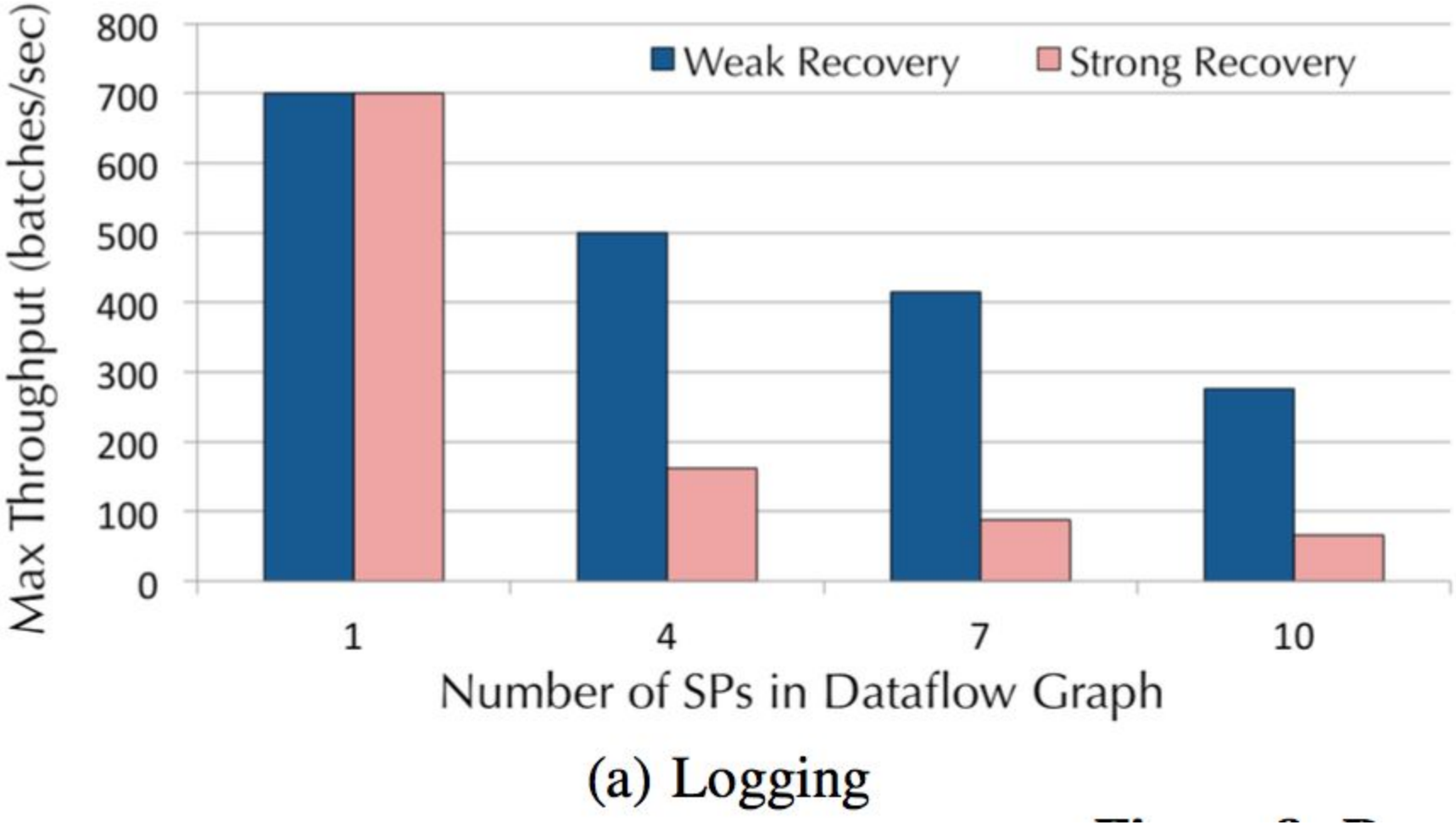
(a) PE Trigger Micro-Benchmark



(b) PE Trigger Result

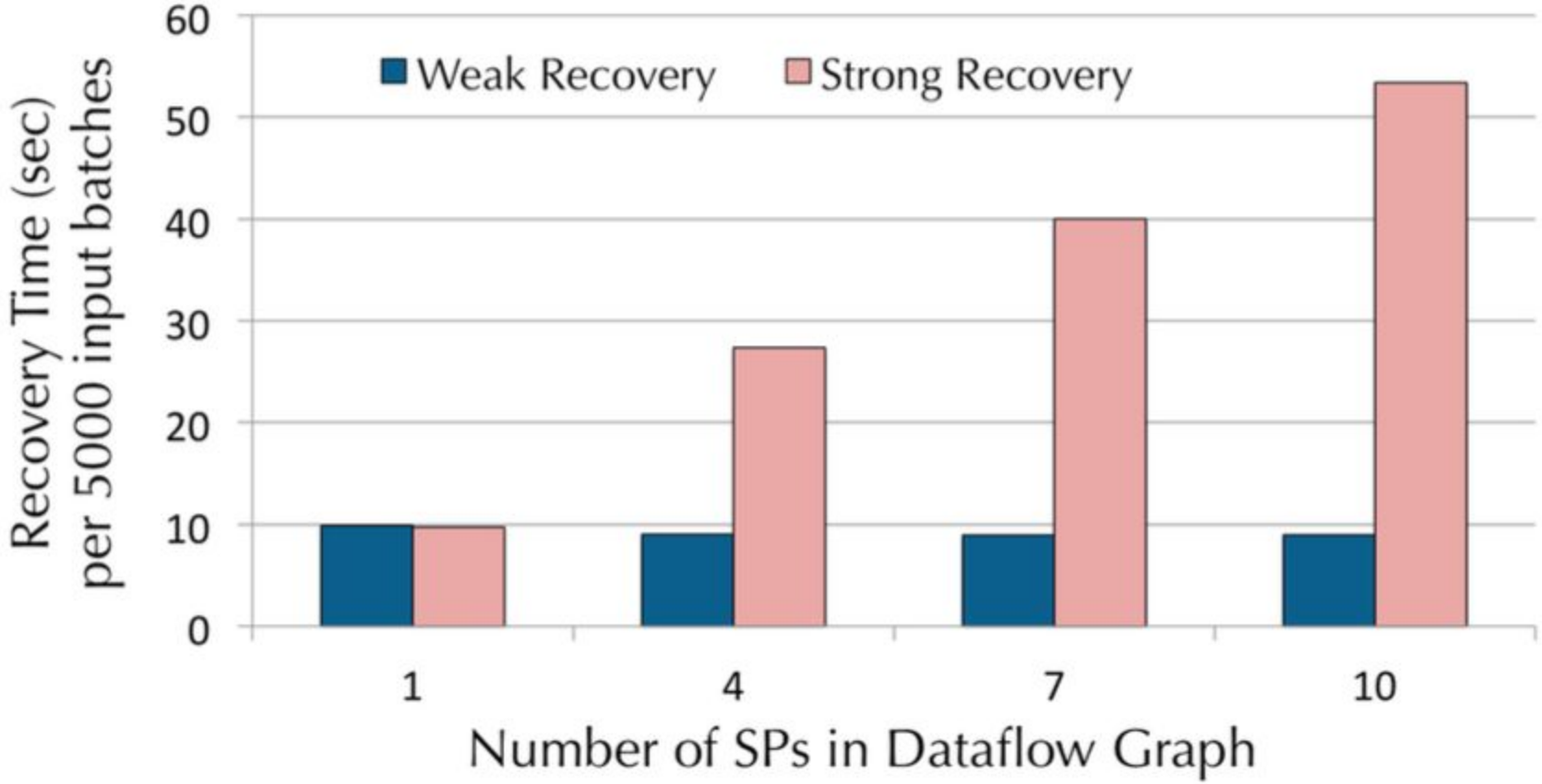
Figure 7: Partition Engine Triggers

Logging becomes a bottleneck



(a) Logging

Strong recovery requires communication with recovery manager for each transaction redone from the log



(b) Recovery

Summary

- Introduces transactional semantics for stream processing
- Introduces push-based for transaction processing
- Enables more efficient processing for emerging applications
- Unified computational model for OLTP and streaming transactions
- Strong Recovery and Weak Recovery

Research Question

- How to support OLAP queries that read from multiple tables in S-Store?
  - OLTP+OLAP+Transactional Streaming
- What is the programming model that is used for programming the dataflow graphs?
- Why not using something like LINQ instead of Java+SQL?

Thanks You