ISTC BIG DATA

S-Store: Streaming meets Transaction Processing

Nesime Tatbul (Intel Labs & MIT)

joint work with

John Meehan, Stan Zdonik, Cansu Aslantas, Ugur Cetintemel, Tim Kraska (Brown)

Mike Stonebraker, Sam Madden, Hao Wang (MIT)

Kristin Tufte, Dave Maier (PSU)

Andy Pavlo (CMU)

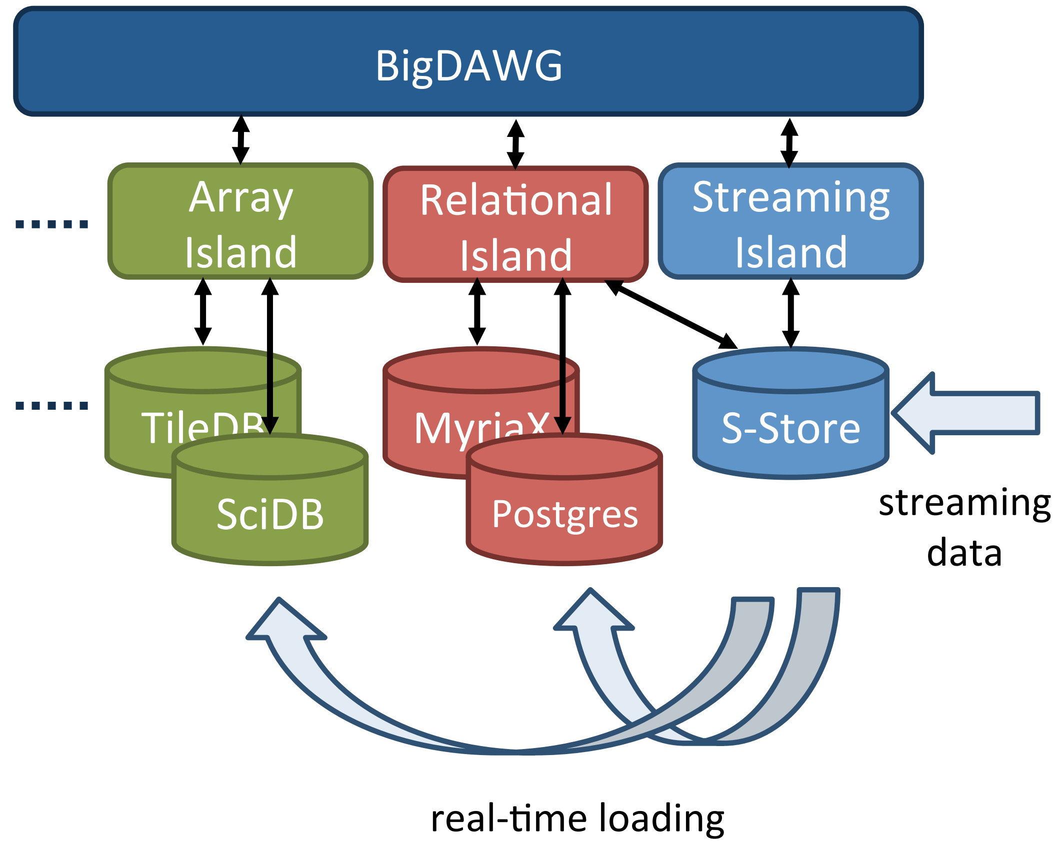
ISTC for Big Data

- One of Intel's 4 current Science and Technology Centers in the US (+6 similar ones world-wide)
- MIT as main hub + 8 other universities
- Launched in 2012, 3+2 years of funding
- Research themes:
  - Data analytics & processing platforms
  - Scalable math & algorithms
  - Visualization
  - Architecture
  - Benchmarks & testbeds
  - **Integration across multiple data processing systems**

The BigDAWG Polystore System

S-Store: BigDAWG's Streaming Data Store

- Reliable, real-time ingest of streaming data
- In-memory processing of all streaming analytics workloads
- Support for transactional state management and relational OLTP workloads
- Real-time ETL of new data into other BigDAWG stores
- Critical enabler for joining current data with past data



The Big Velocity Challenge

- Data is coming too fast!
  - Sensors, Smart phones, Internet of Things, Web clicks, Stock tickers, Social media feeds, News feeds, …
- Applications need:
  - scalable data ingest, processing, and storage
  - real-time, complex data analytics
  - high-throughput, transactional processing
  - data-driven, continuous, incremental processing models

State of the Art & Recent Trends

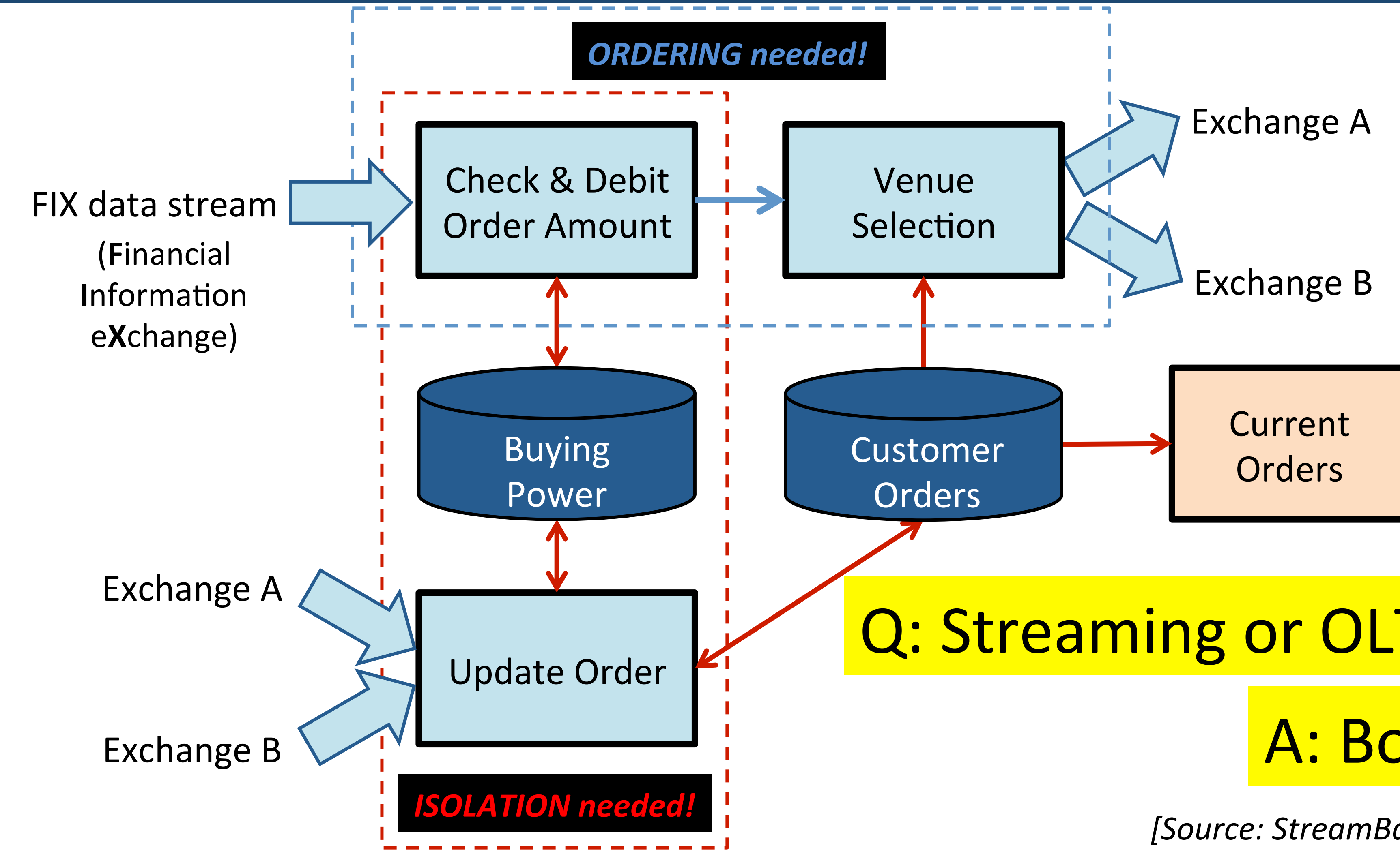
- Stream processing
  - in-memory, low-latency processing
  - fine-grained batching of inputs, complex datafl … utations
  - scalab … s

....

What about streams + transactions?

- Transaction processing
  - disk-based OLTP -> main-memory OLTP
  - multi-core, shared-nothing clusters
  - NewSQL architectures (scalable SQL and ACID)

Shared Mutable State in Streaming
A Real-World Example: Financial Order Routing



Q: Streaming or OLTP?

A: Both!

[Source: StreamBase, Inc.]

S-Store in a Nutshell

- A single system for transaction & stream processing
- A novel computational model for supporting hybrid workloads with well-defined correctness guarantees
  - **ACID** guarantees for individual transactions (OLTP + streaming)
  - **ordered execution** guarantees for dataflow graphs of streaming transactions
  - **exactly-once processing** guarantees for streams (no loss or duplication)
- A flexible and expressive programming interface
  - transactions as user-defined stored procedures (Java) w/ SQL-based data access
  - support for dataflow graphs and nested transactions
- Scalable software architecture and implementation
  - distributed main-memory OLTP system as foundation (H-Store)
  - clean and general architectural extensions (e.g., triggers, windowing)

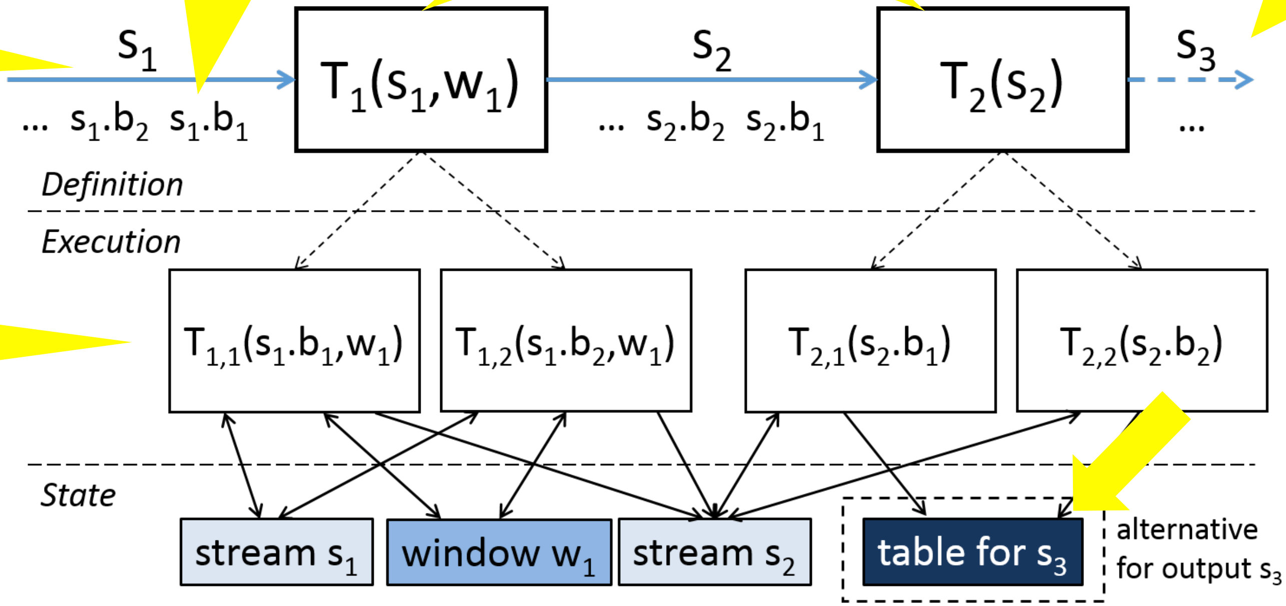
Hybrid Computational Model

Batch-id's are used to track lineage and order

Dataflow Graphs of **Streaming Transactions**

Push outputs to an external sink (or store in a Table)

Stream as a sequence of **Atomic Batches**



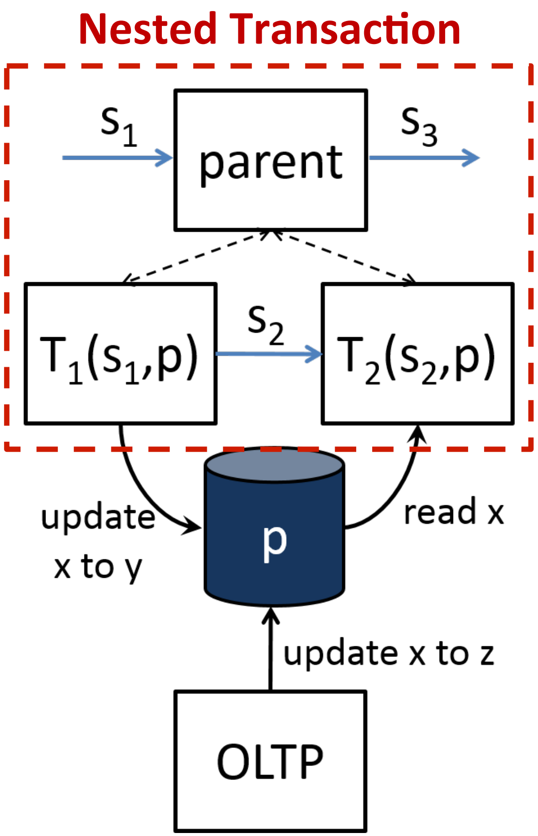
Each batch leads to a **Transaction Execution (TE)**

Three kinds of state: **Streams, Windows, and Tables**
- All physically kept as in-memory tables
- Tables can be publicly shared among all transactions (**OLTP or Streaming)**
- Streams & Windows are not publicly shareable

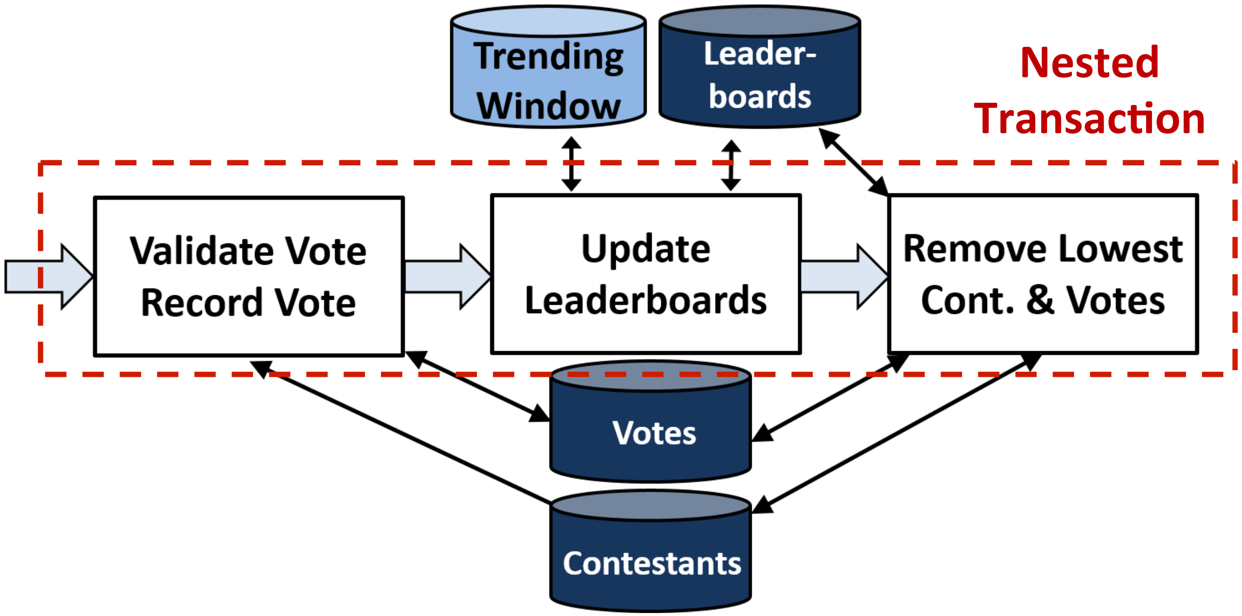
Nested Transactions for coarse-grained isolation

Example Uses for Nested Transactions

Use 1: To protect parts of a dataflow graph from other OLTP or Streaming transactions



Use 2: To protect one instance of a dataflow graph from its subsequent instances (e.g., Leaderboard Benchmark)



Triple Correctness Guarantees

- **ACID** from traditional OLTP
  - Failure recovery (Atomicity and Durability)
  - Concurrency control (Consistency and Isolation)
- **Ordered execution** from Streaming
  - Atomic batches of a stream must be processed in order (stream order constraint)
  - For a given atomic batch, transactions in a dataflow graph must be processed in topological order (dataflow order constraint)
  - Nested transactions require strict serial ordering
- **Exactly-once processing** from Streaming
  - Recovering from failures (i.e., replay of streams) should not cause lost or

➢ S-Store provides efficient scheduling and recovery mechanisms to ensure these guarantees.

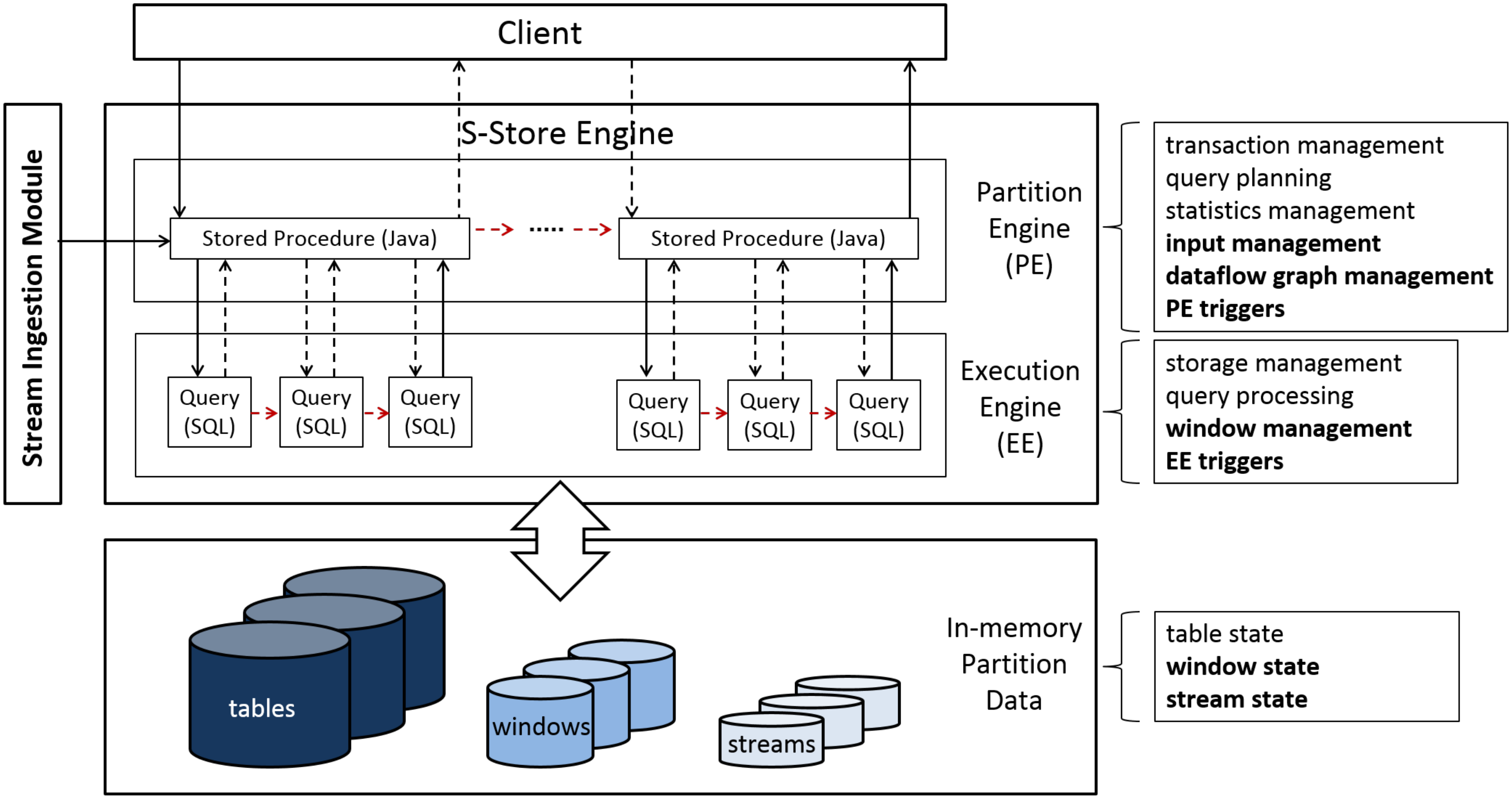
H-Store as System Foundation

- main-memory OLTP system developed at Brown & MIT
- base design for the VoltDB NewSQL database system
- programming model: stored procedures (Java + SQL)
- database partitioned across multiple sites in a way to minimize the number of distributed transactions
- single-threaded transaction execution per partition
- recovery via checkpointing + command-logging
- anti-caching to disk if all data does not fit in memory

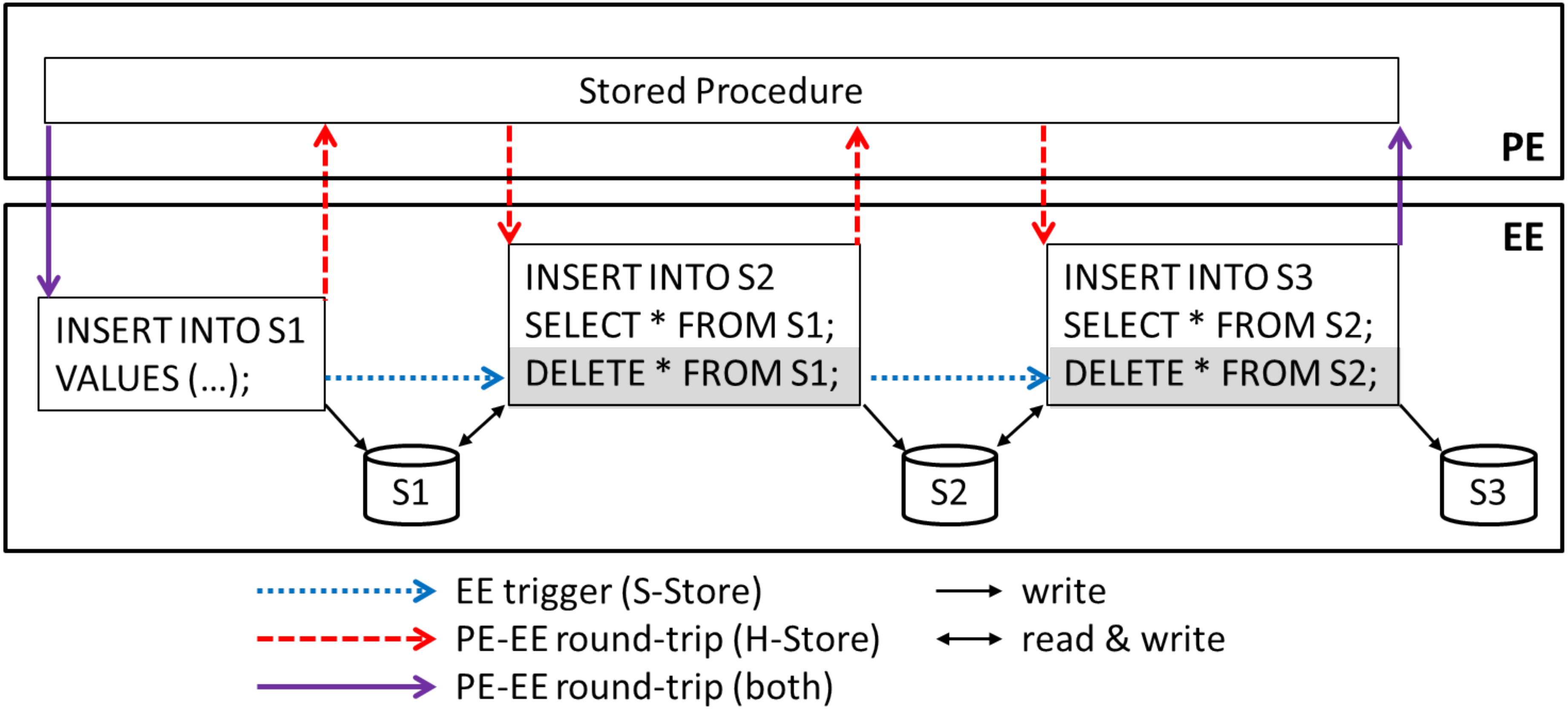
S-Store = H-Store + Streaming

intel MIT BROWN Carnegie Mellon University W UT Portland State UNIVERSITY

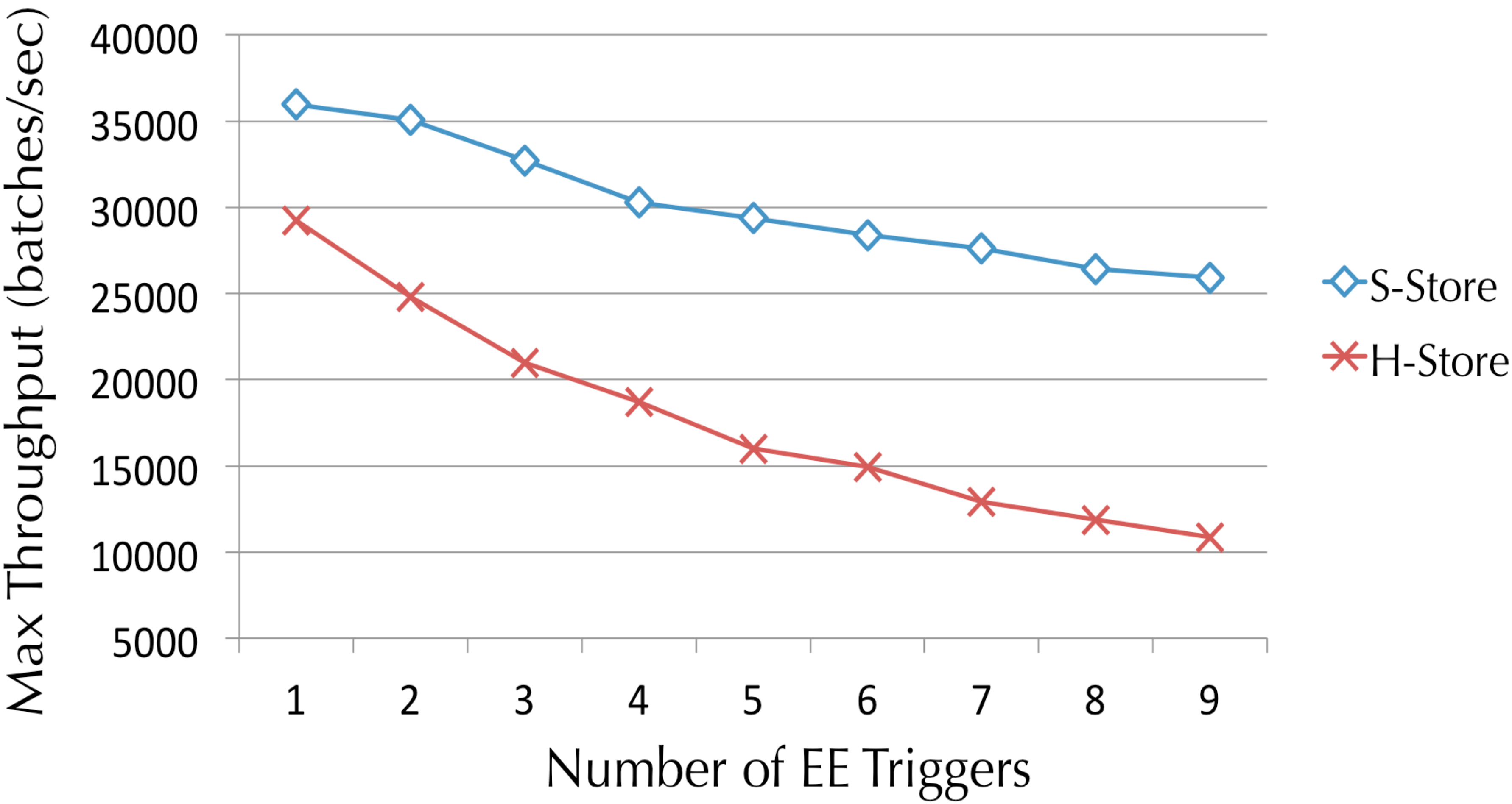
S-Store's Extended Architecture



S-Store vs. H-Store: EE Triggers

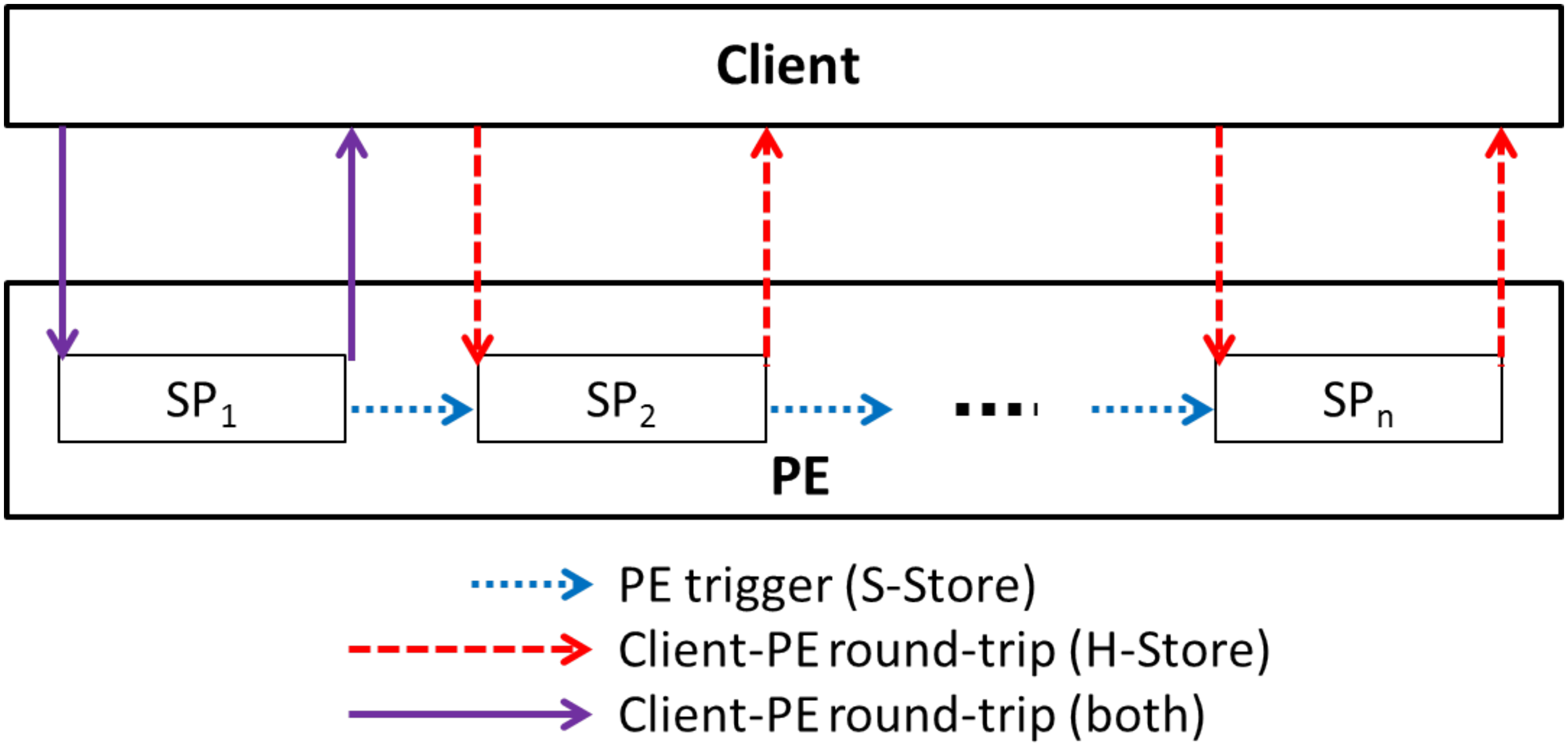


S-Store vs. H-Store: EE Triggers

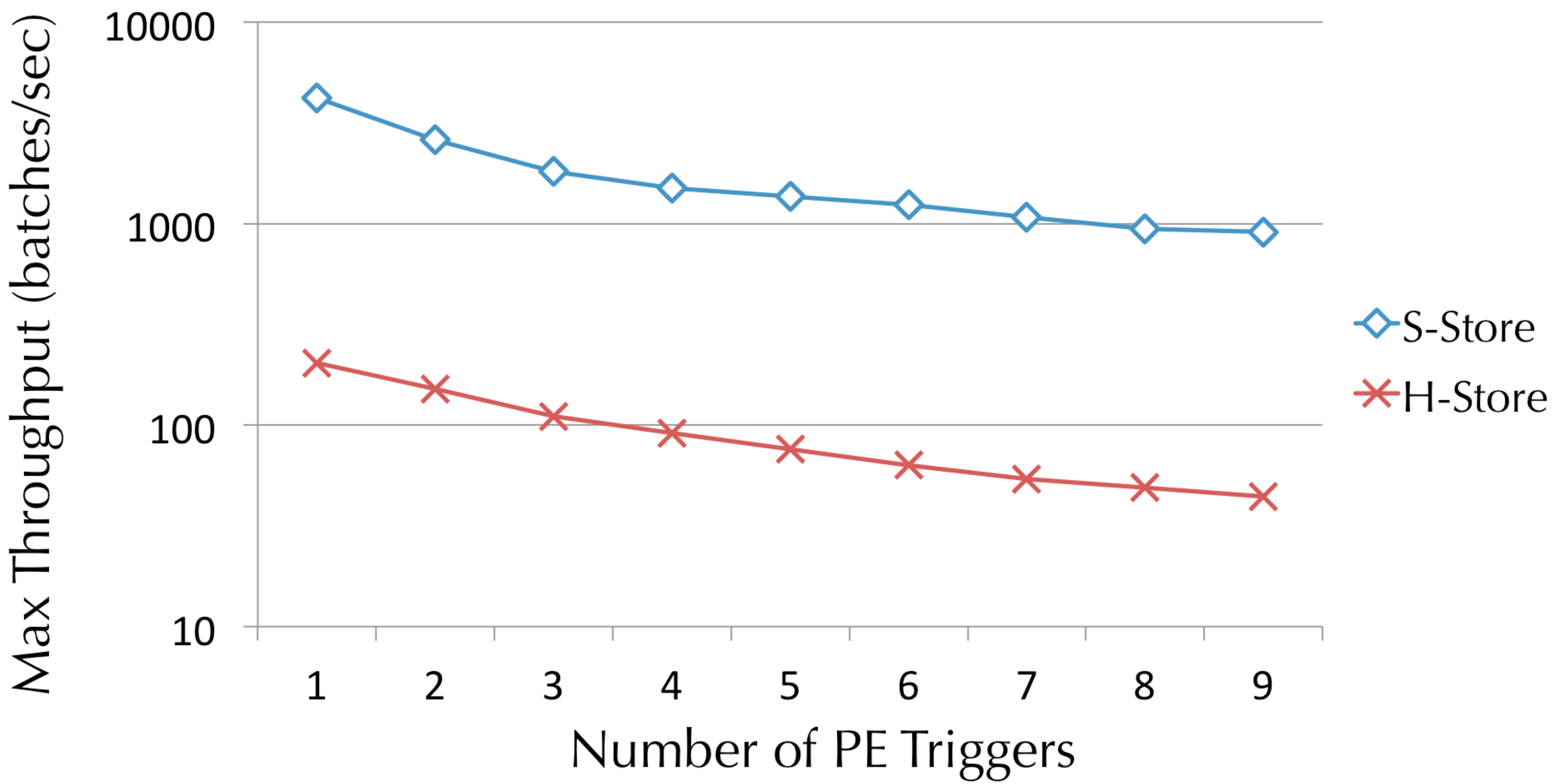


Portland State
UNIVERSITY

S-Store vs. H-Store: PE Triggers



S-Store vs. H-Store: PE Triggers

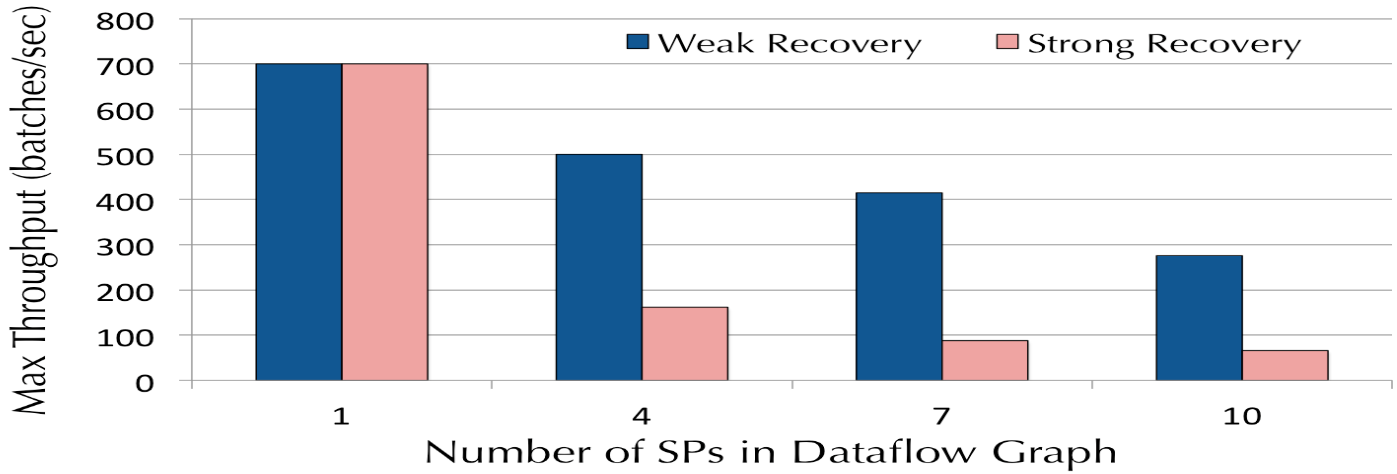


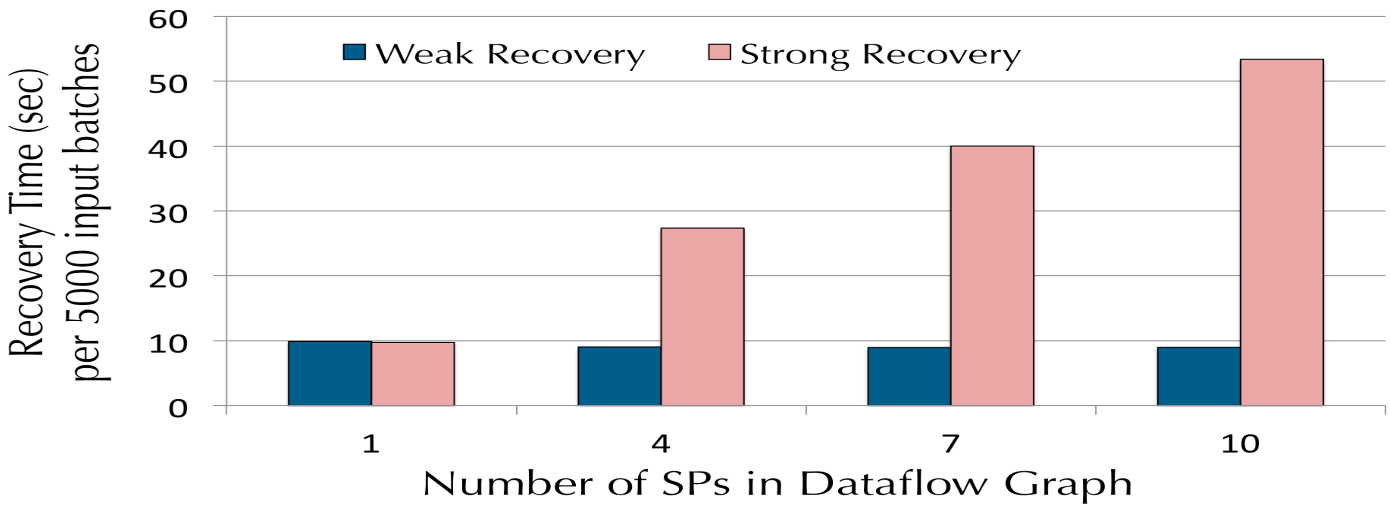
Fault Tolerance in S-Store

Check-pointing + Command-logging + Upstream backup

- Periodic check-pointing of in-memory tables to disk
- Strong recovery
  - Log all committed transactions (OLTP + streaming)
  - Upon failure, log replay reproduces the exact pre-failure state
  - To avoid redundancy, must turn off triggers during recovery
- Weak recovery
  - Log transactions selectively (all OLTP + "border" streaming)
  - Upon failure, log replay may lead to a different, but correct state
  - No need to turn off triggers
- Upstream backup for streaming inputs that have not yet been accounted for in downstream logs

Weak Recovery vs. Strong Recovery





S-Store vs. State of the Art

Better Correctness Guarantees & Better Performance

Correctness Guarantees (ACID, Order, Exactly-Once)

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

~ 10 x OLTP
~ 4 x SPE

Leaderboard Benchmark on a single-node Intel® Xeon® E7-4830 at 2.13 GHz

"S-Store: Streaming Meets Transaction Processing", Research Track, PVLDB Vol. 8 No. 13, September 2015.

BROWN
Carnegie Mellon University
Portland State UNIVERSITY

Current Work in Progress
Scaling to Multiple Nodes

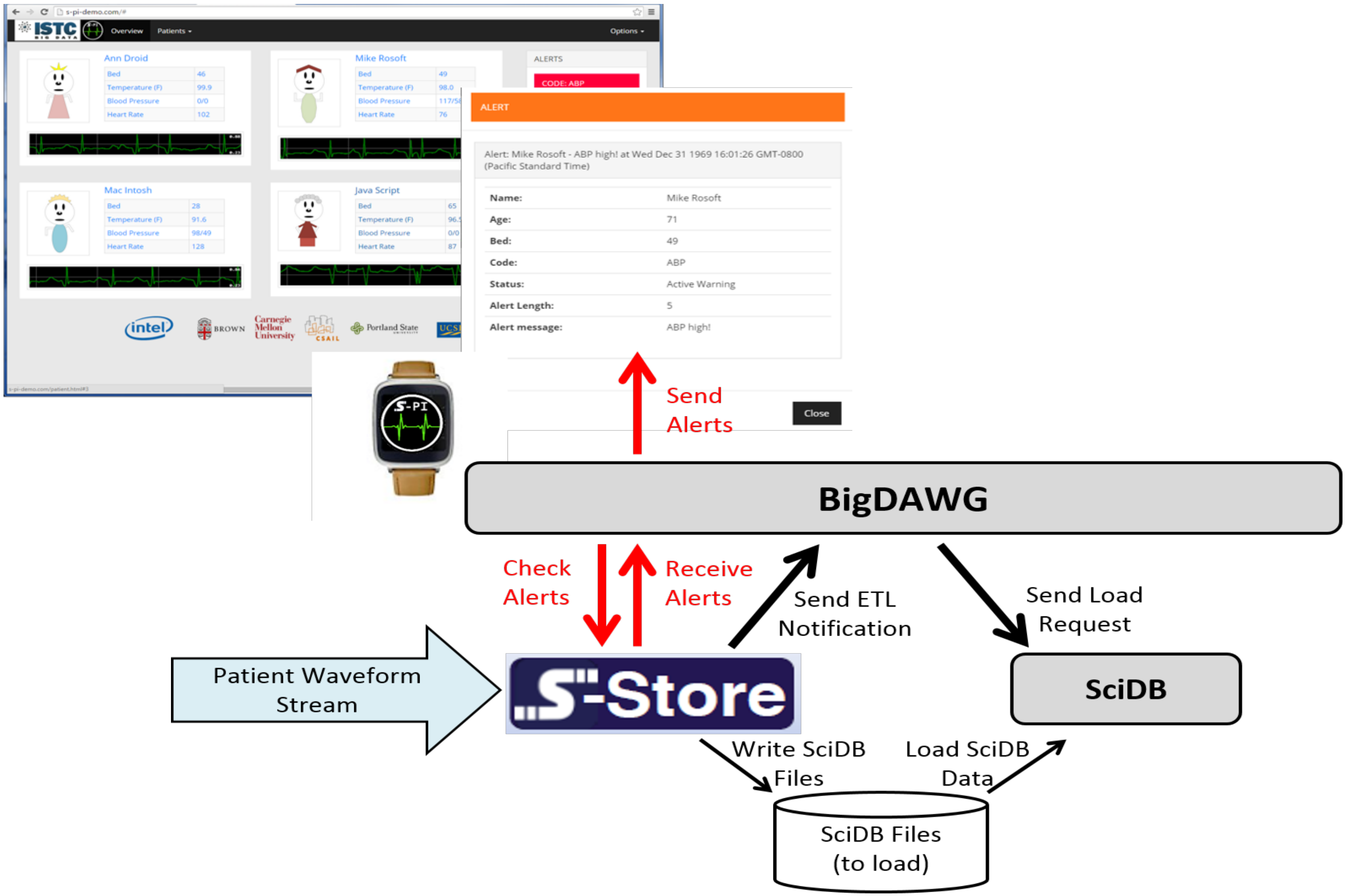
- Three basic primitives to partition a streaming workload:
  - **Move:** Move a stream from one node to another (distributed transaction)
  - **Demux:** Split a stream into multiple partitions
  - **Mux:** Merge multiple streams into one
- Both pipelined (Move) & partitioned parallelism (Demux+Move)
- Research question #1: Given a dataflow graph and a set of processing nodes, where to place Move/Demux/Mux + how to partition public Tables in order to maximize performance and load balance?
- Research question #2: How to ensure correct and efficient scheduling and recovery at all nodes?

Future Directions

- Extend our support for streaming analytics
- Tighter integration with BigDAWG (e.g., optimizing cross-system workloads)
- Hardware-aware S-Store (NVM, many-core, fast networks)
- Handling mixed and dynamic workloads
- Building novel and challenging use cases

Demos

S-Store in Action
The MIMIC Demo



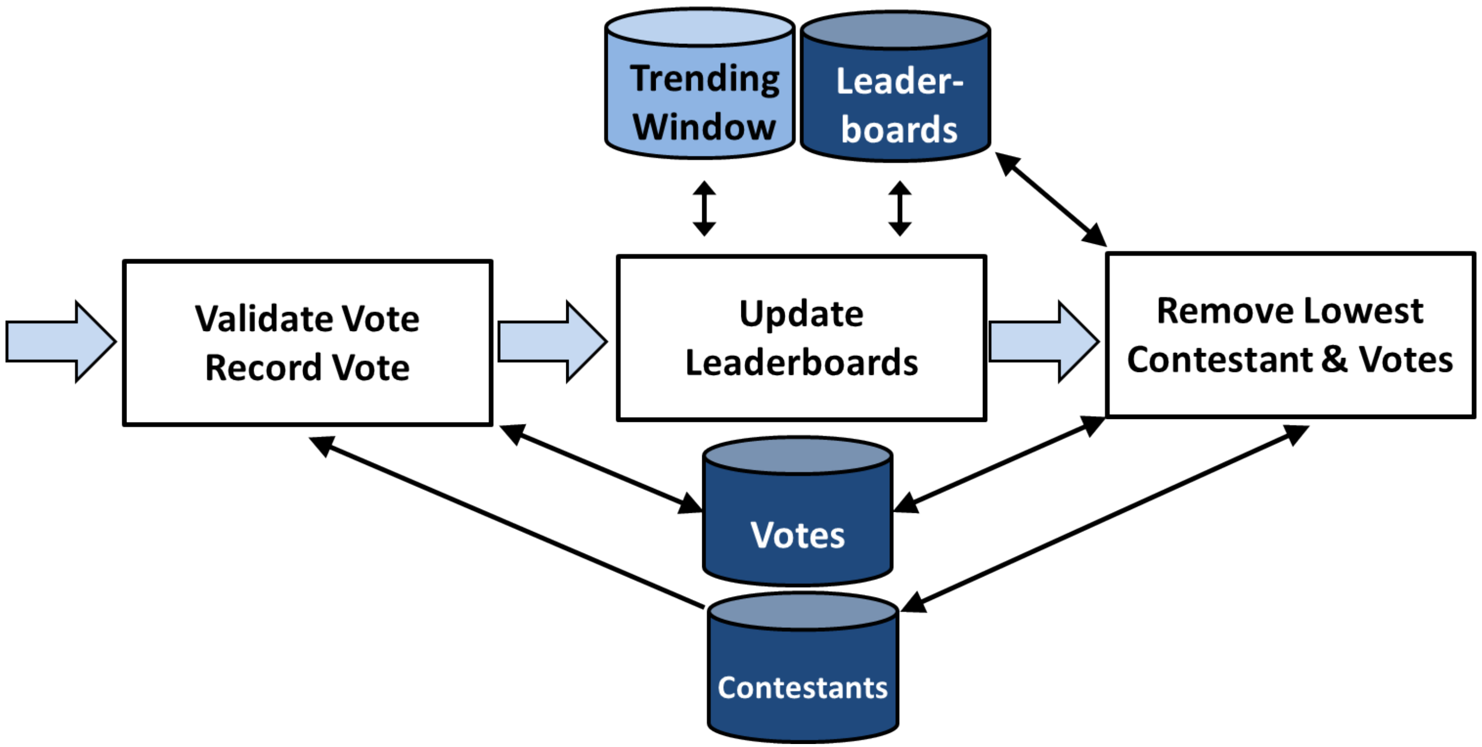
S-Store in Action

The MIMIC Demo

S-Store

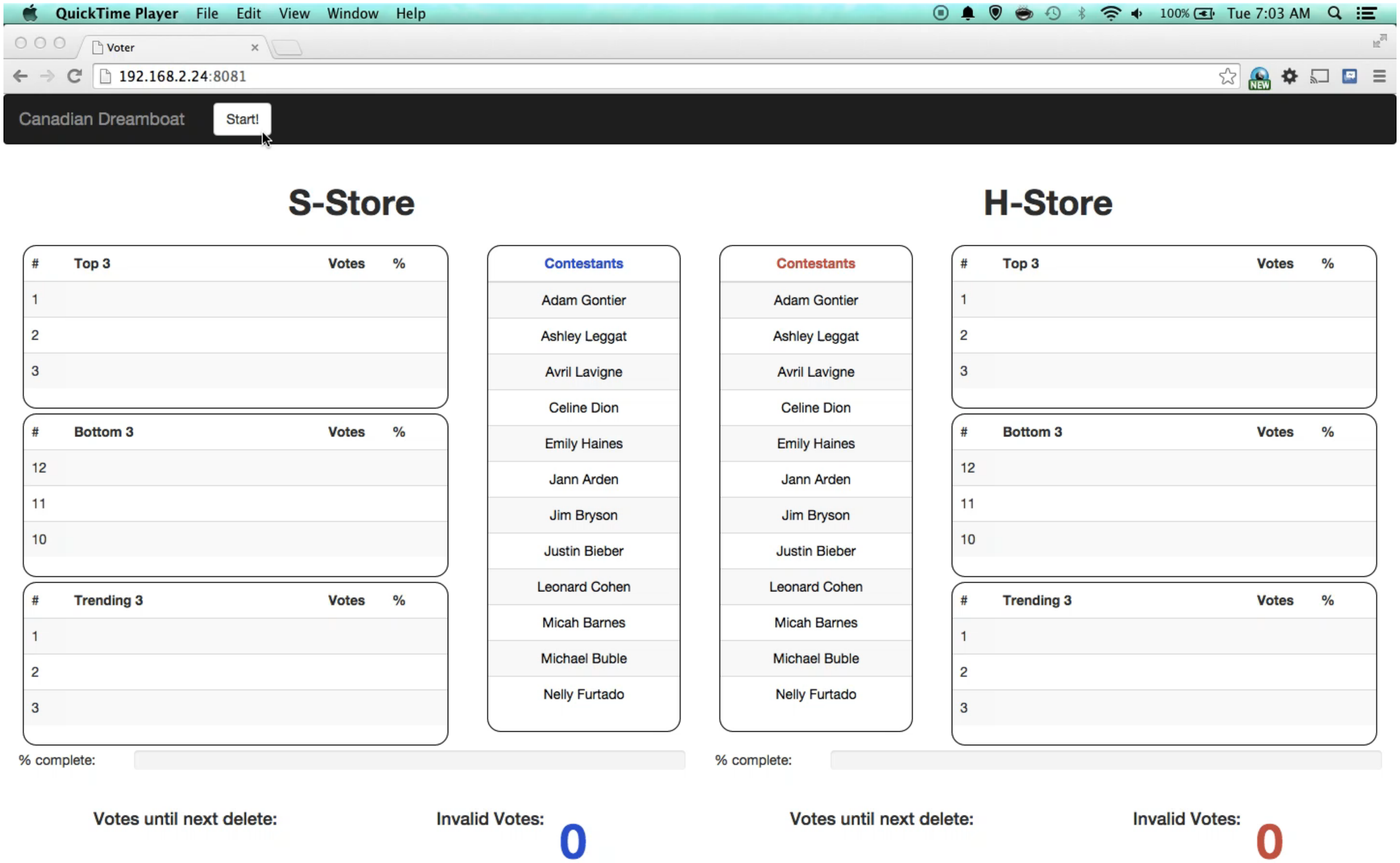
S-Store in Action

The Canadian Dreamboat Demo



S-Store in Action

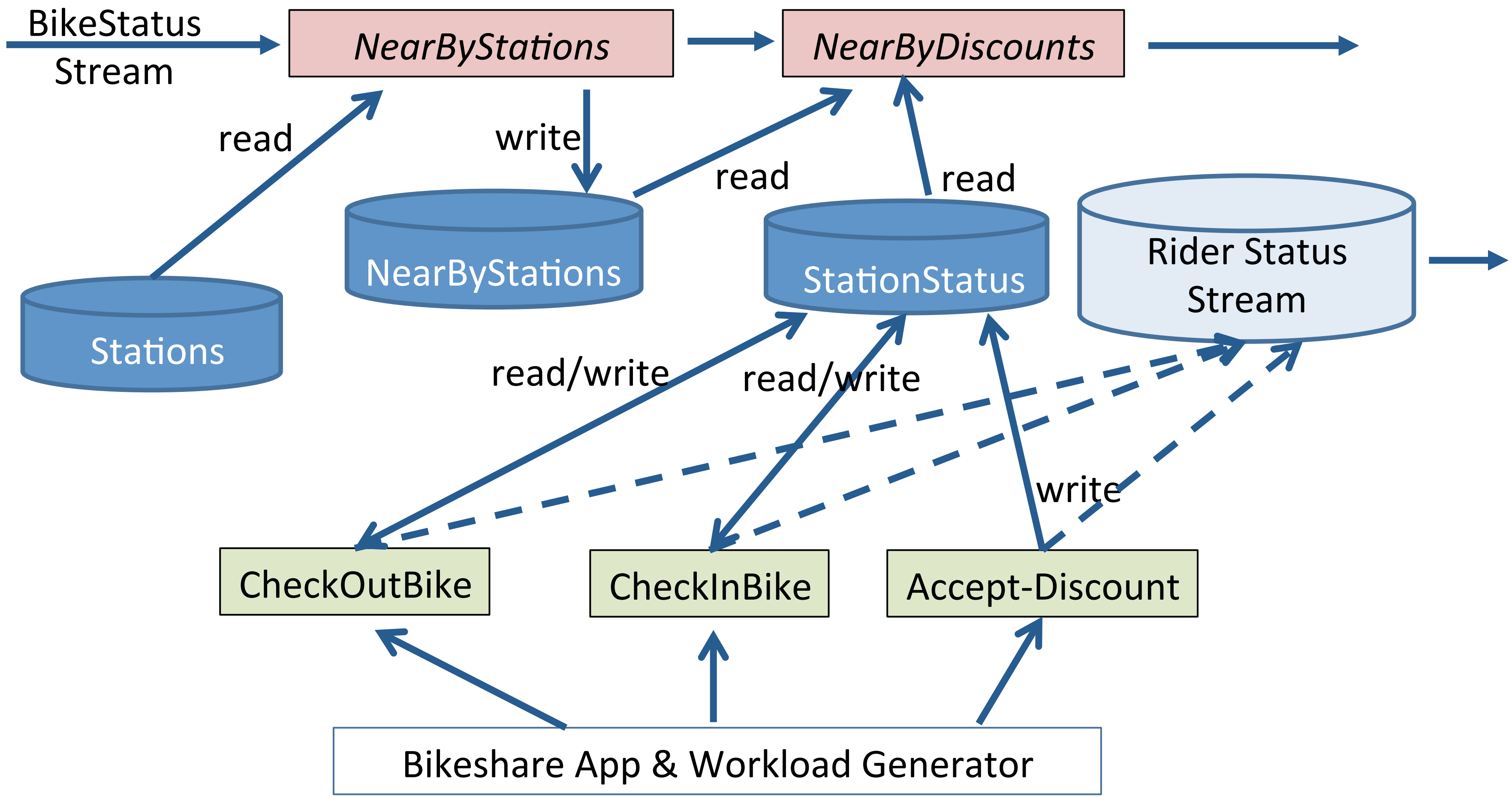
The Canadian Dreamboat Demo



intel BROWN Carnegie Mellon University W Portland State UNIVERSITY

S-Store in Action

The BikeShare Demo



S-Store in Action

The BikeShare Demo