Siddhi-CEP

A SECOND LOOK AT COMPLEX EVENT PROCESSING

Srinath Perera,
Senior Software Architect WSO2 Inc.
Visiting Faculty, University of Moratuwa
Member, Apache Software Foundation

http://siddhi.sourceforge.net/

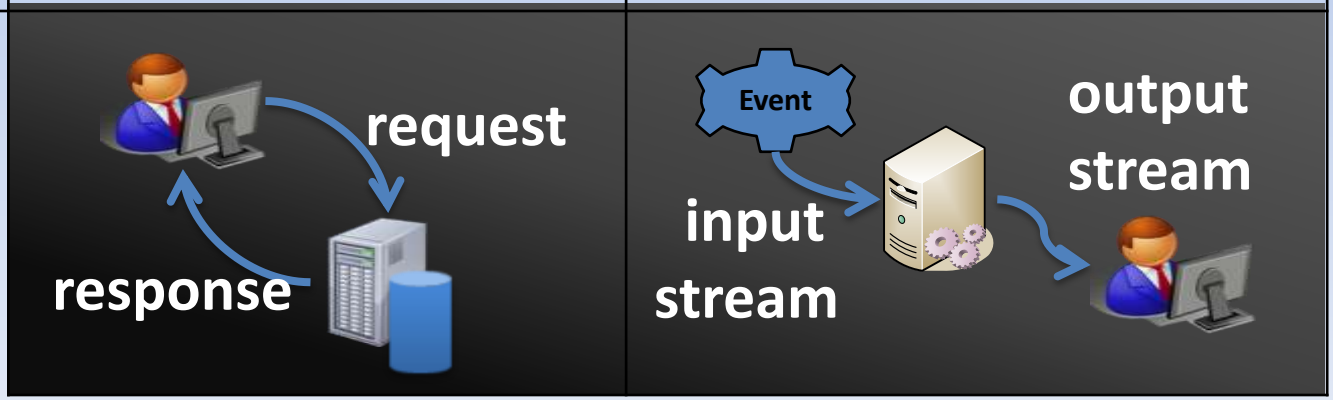
OUTLINE

- Introduction to CEP
- Second look at CEP Implementations
- Siddhi Architecture
- Siddhi Performance
- Conclusion and Future Topics

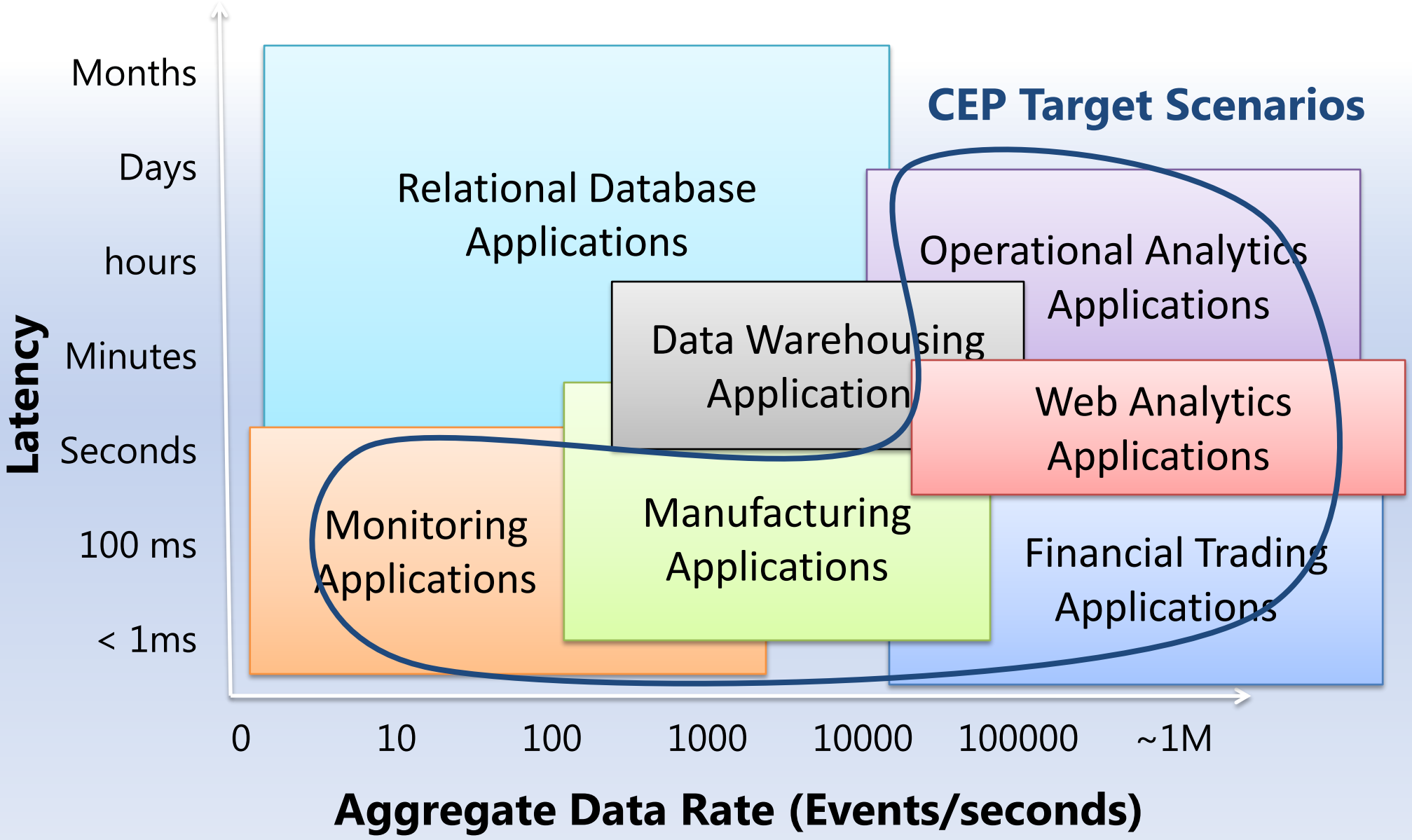
WHAT IS CEP ?

Complex Event Processing (CEP) is identifying meaningful patterns, relationships and data abstractions among unrelated events and fire an immediate response.

	Database Applications	Event-driven Applications
Query Paradigm	Ad-hoc queries or requests	Continuous standing queries
Latency	Seconds, hours, days	Milliseconds or less
Data Rate	Hundreds of events/sec	Tens of thousands of events/sec or more



Scenarios of Event Processing



E-SCIENCE AND CEP

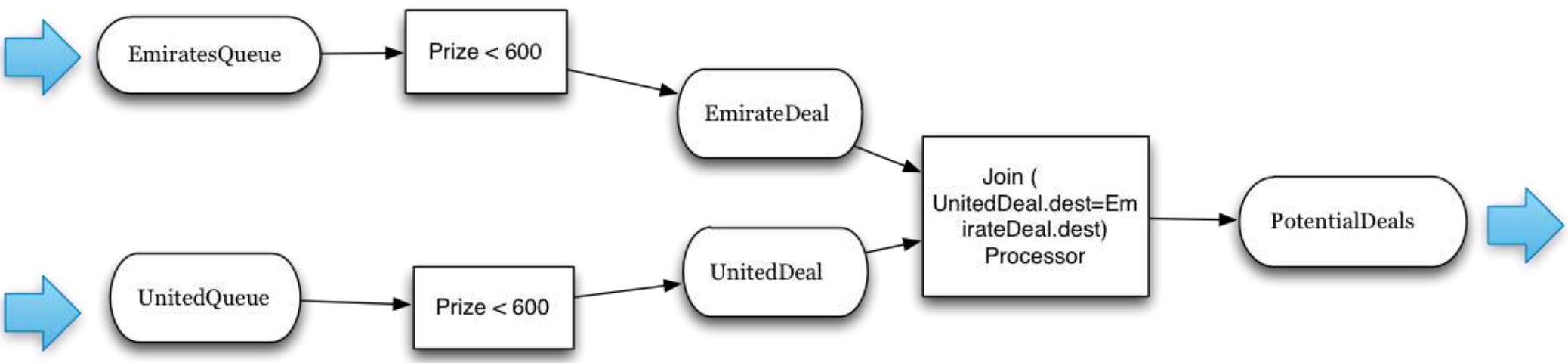
- E-Science often deals with National and Global scale usecases while we try to understand the world around us better.
- Following is an important class of E-science applications
  - Receives data about the world around us from sensors deployed across the country/world
  - Try to make sense, react to, predict, and/or control the world around us
  - Examples: Weather, Traffic Data, Surveillance, Smart Grid etc..
- CEP is a powerful enabling technologies for these usecases

CEP Queries

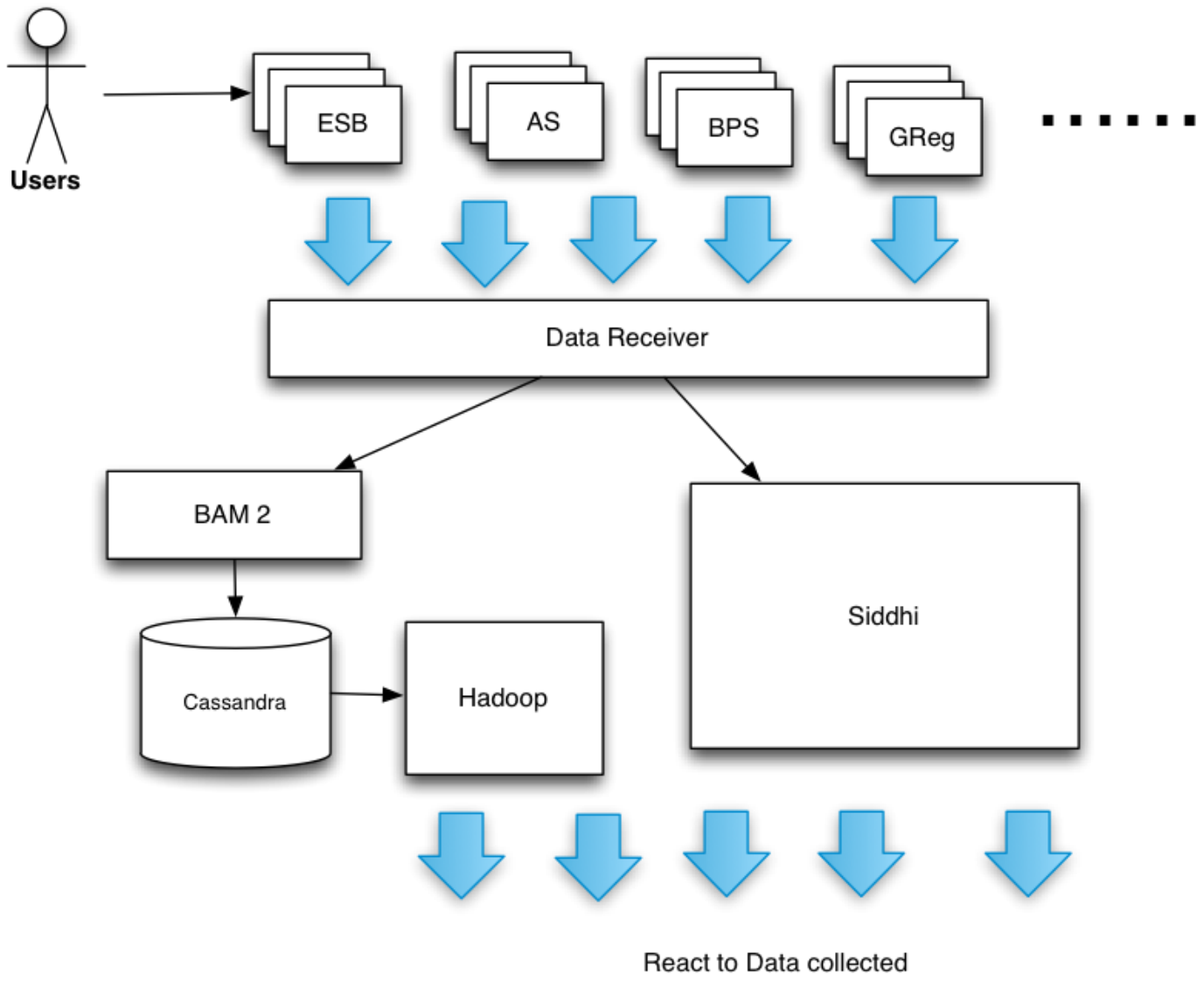
- Each Event consists of properties (name value pairs)
- We separate different events to streams
- Use a SQL like query language, but queries are evaluated on continuous event streams
- Types of queries are following
  - Selection (filtering) and projection (e.g. like select in SQL)
  - Windows – events are processed within a window (e.g. for aggregation and joins). Time window vs. length window.
  - Ordering – sequences and patterns (before, followed by conditions e.g. new location followed by small and a large purchase might suggest a fraud)
  - Join and split
  - Aggregation

EXAMPLE QUERY

```
EmirateDeal= Select * from Emirate where prize < 600
UnitedDeal = Select * from United where prize < 600
PotentialDeal = Select * from EmirateDeal join UnitedDeal
where  UnitedDeal .dest=EmirateDeal .dest
```



CEP Role within WSO2 Platform



MOTIVATING USE CASE: LOS ANGELES SMART GRID DEMONSTRATION PROJECT

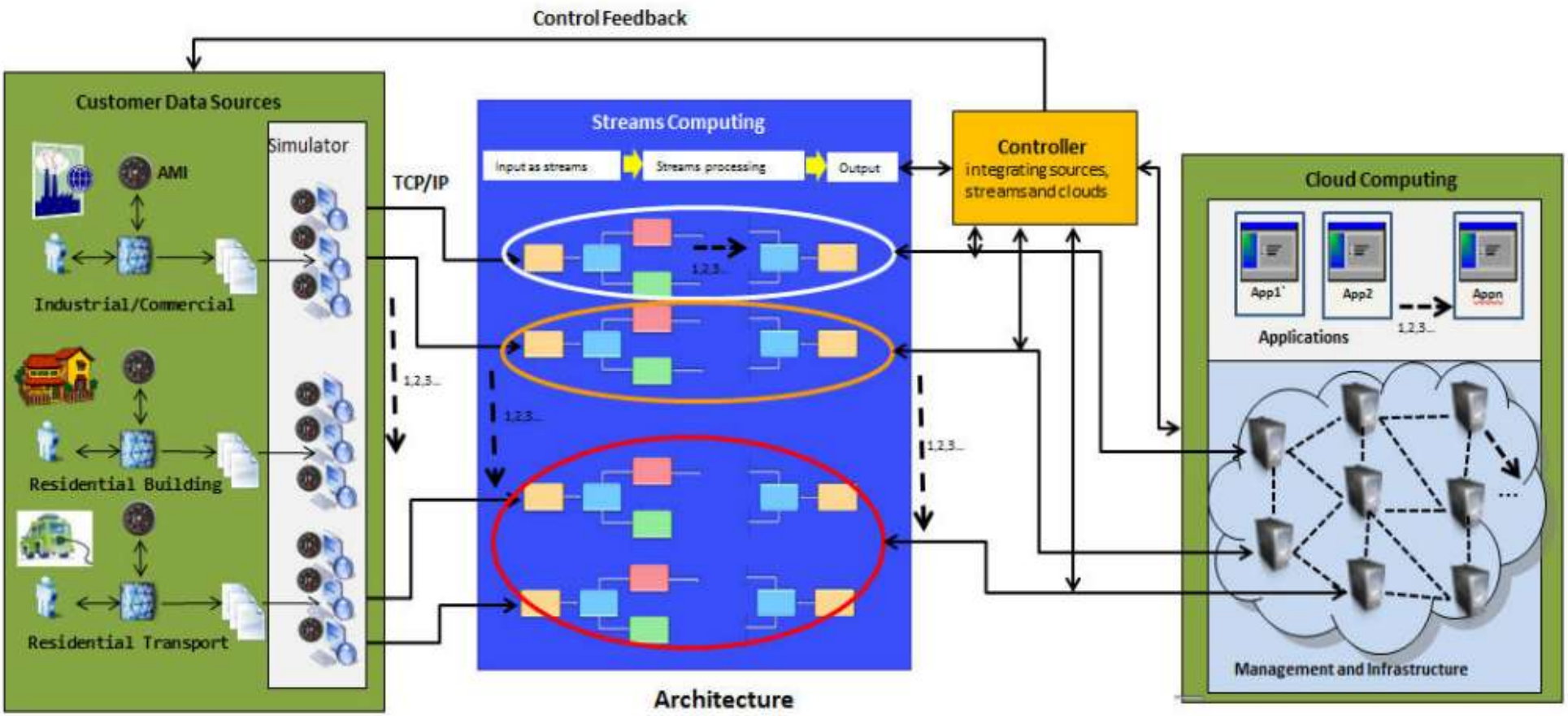


Figure 2: Design for adaptive rate control feedback from stream processing system to AMIs to optimize bandwidth into Cloud platform

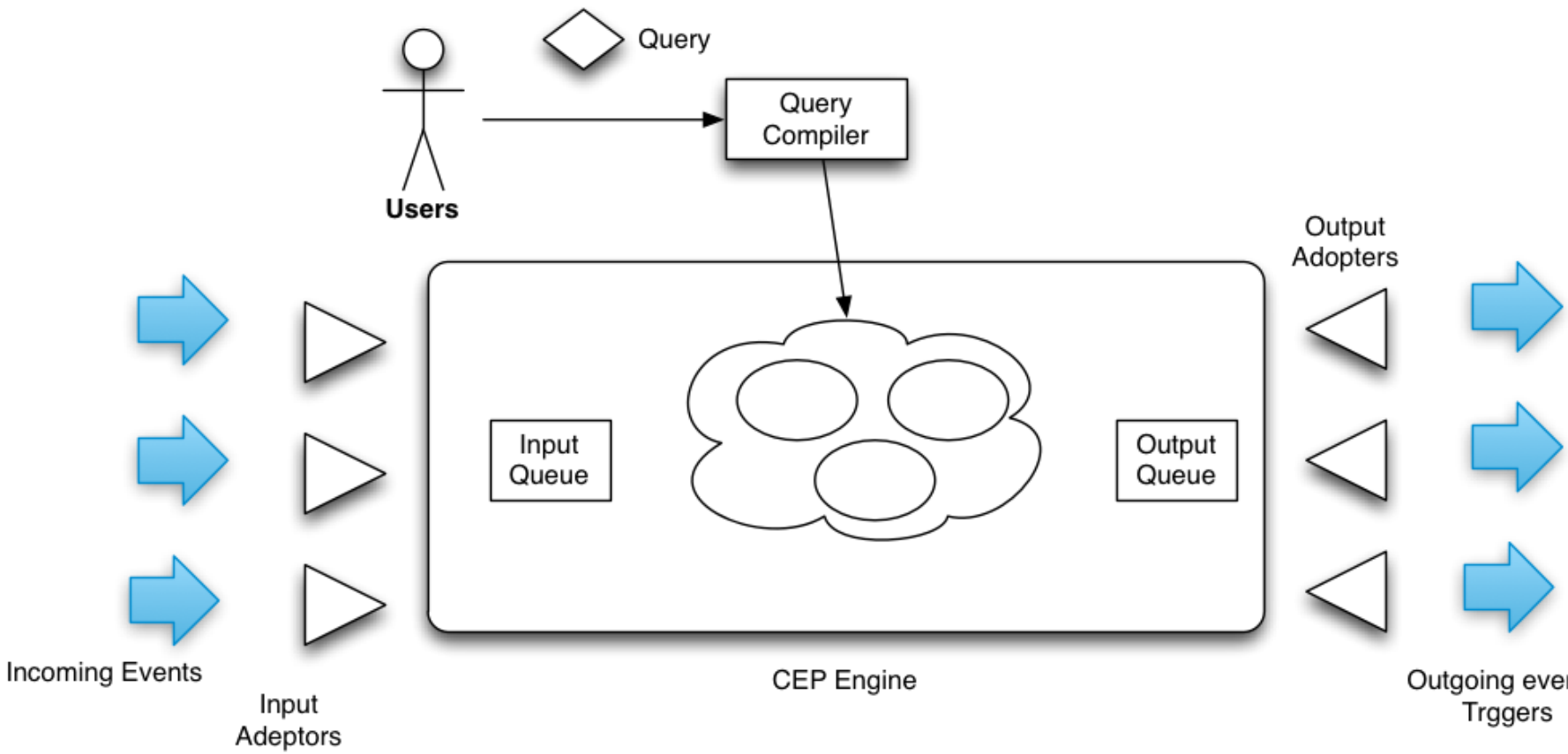
Related Work on CEP Implementations

- A very good reference: “G. Cugola and A. Margara. Processing flows of information: From data stream to complex event” processing. ACM Computing Surveys, 2011.
- Initial work from Databases, active databases. But the “store and process” model was too slow.
- Stream processing uses a pub/sub model, very scalable but lacks temporal support
  - (Aurora [5, 2], PIPES [23, 7], STREAM [30], Borealis [6,1] S4 [25], and Storm.)
- CEP engines
  - Mainly uses a NFA based model
  - SASE (data flow and NFA)
  - Esper – NFA and Delta networks
  - Cayuga – NFA and re-subscriptions (single thread)

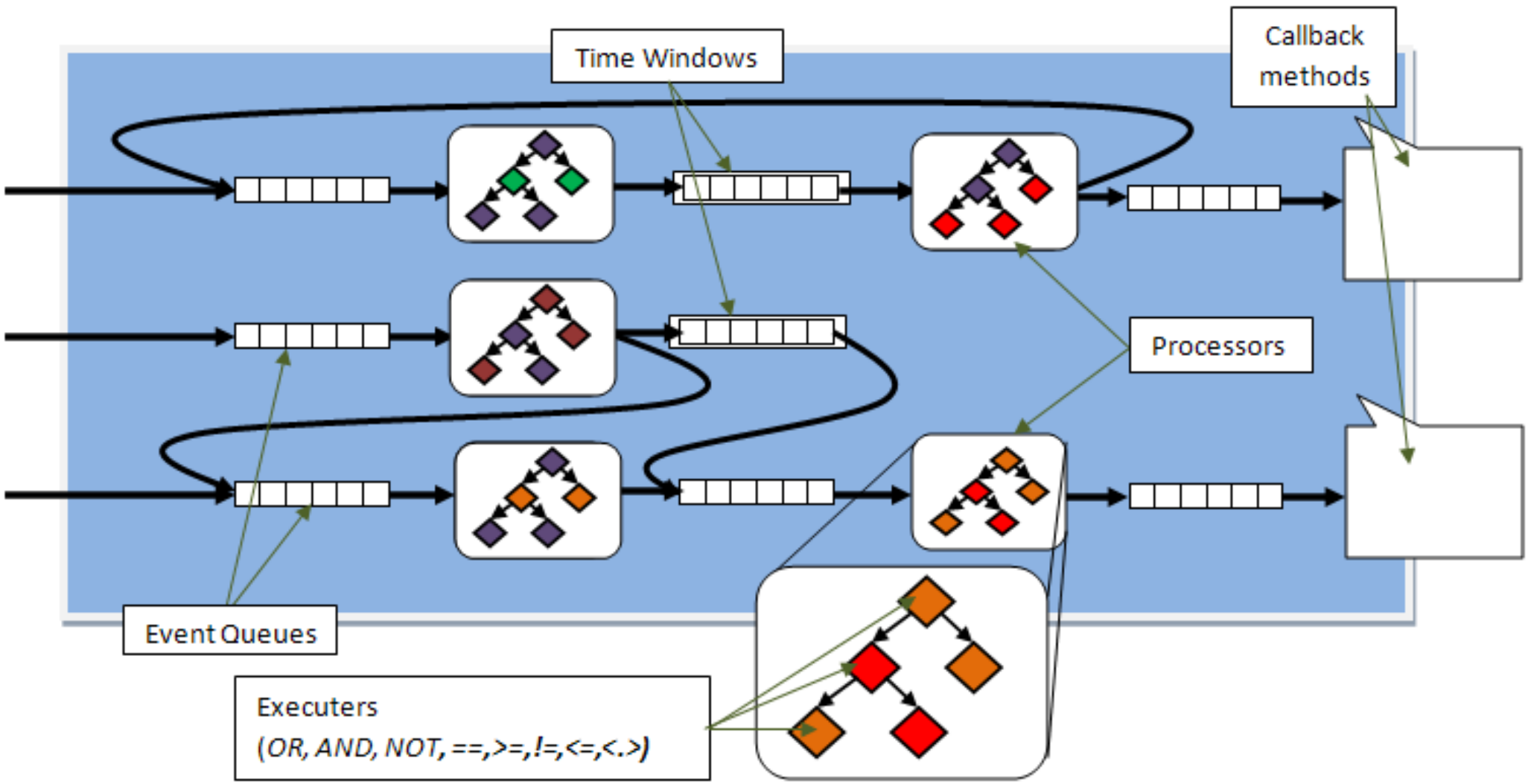
GOAL: IDEAS FOR IMPROVEMENT

- Borrowing ideas from Stream processing: Pipeline based architecture with a pub/sub model inspired by Stream processing.
- Breaking single thread evaluations: Improving parallelism in processing through a pipeline based model
- A New State machine implementation that avoids checking a series of conditions
- Improved support for query chaining with complex queries

SIDDHI HIGH LEVEL ARCHITECTURE



SIDDHI PROCESSING IMPLEMENTATION



SIDDHI PROCESSING IMPLEMENTATION

- Uses a pipeline architecture
- Consist of a pipeline of processors (support all other operators except windows)
- Processors are connected by queues (they implement windows)
- Processors take inputs from a queue by subscribing to them and placing results in a output queue
- Many threads are assigned to processors, where they take data from input queues, process them, and place them in output queues.

API TO WRITE QUERIES

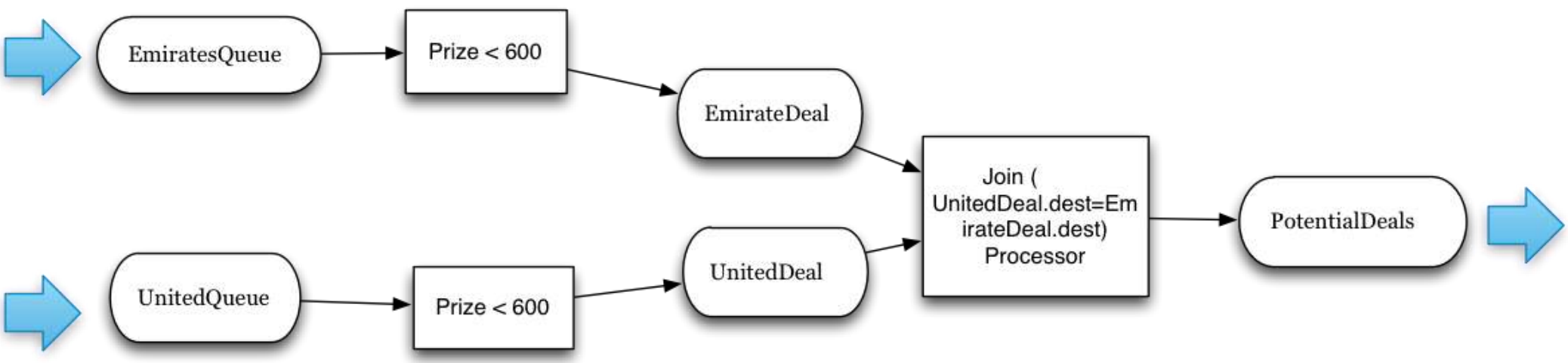
```
//Instantiate SiddhiManager
SiddhiManager siddhiManager = new SiddhiManager();
//Get the QueryFactory to create queries
QueryFactory qf = siddhiManager.getQueryFactory();

InputEventStream stockStream = new InputEventStream(
    "StockStream",
    new String[]{"symbol", "price", "volume"},
    new Class[]{String.class, Float.class, Long.class}
);
//Setting a time window of 1 hour
stockStream.setTimeWindow(3600000);
//Assign input stream
siddhiManager.addInputEventStream(stockStream);

//Create Query
Query query = qf.createQuery(
    "StockQuote",
    qf.output("symbol=StockStream.symbol","price=avg(StockStream.price)"),
    qf.inputStream(stockStream),
    qf.and(
            qf.condition("StockStream.symbol", EQUAL, "IBM"),
            qf.condition("StockStream.volume", GREATERTHAN, "10000")
    )
);
//Assign query
siddhiManager.addQuery(query);
```

EXAMPLE QUERY IMPLEMENTATION

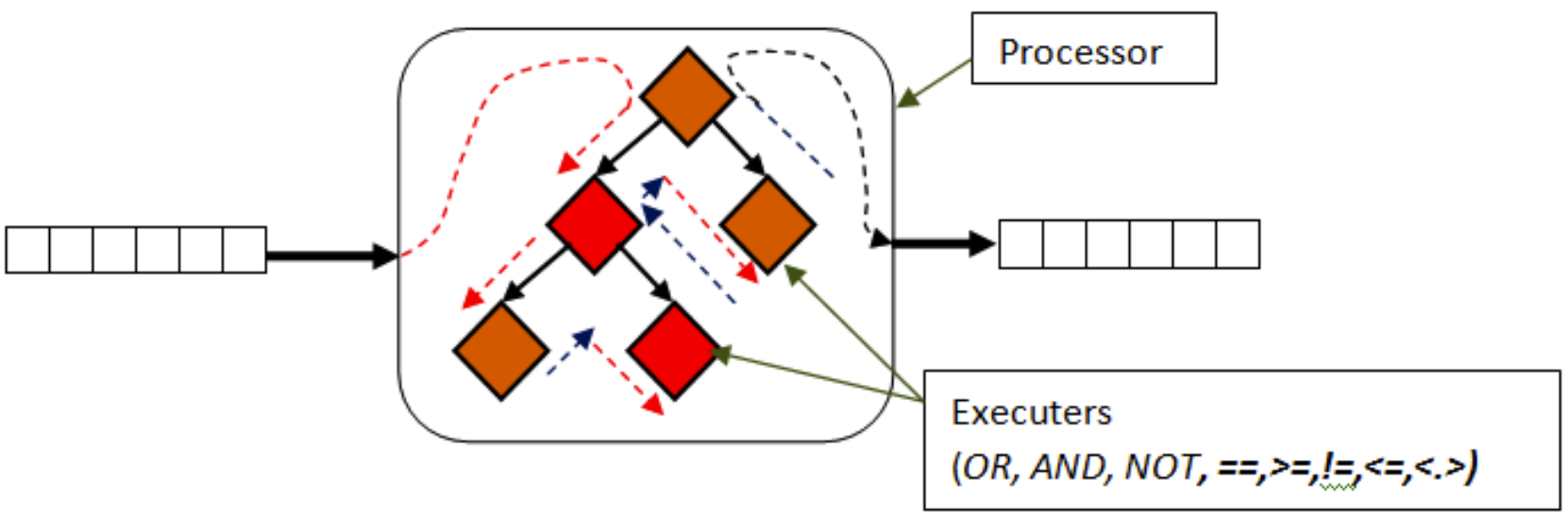
```
EmirateDeal= Select * from Emirate where prize < 600
UnitedDeal = Select * from United where prize < 600
PotentialDeal = Select * from EmirateDeal join UnitedDeal
where  UnitedDeal .dest=EmirateDeal .dest
```



IMPLEMENTING FILTERS

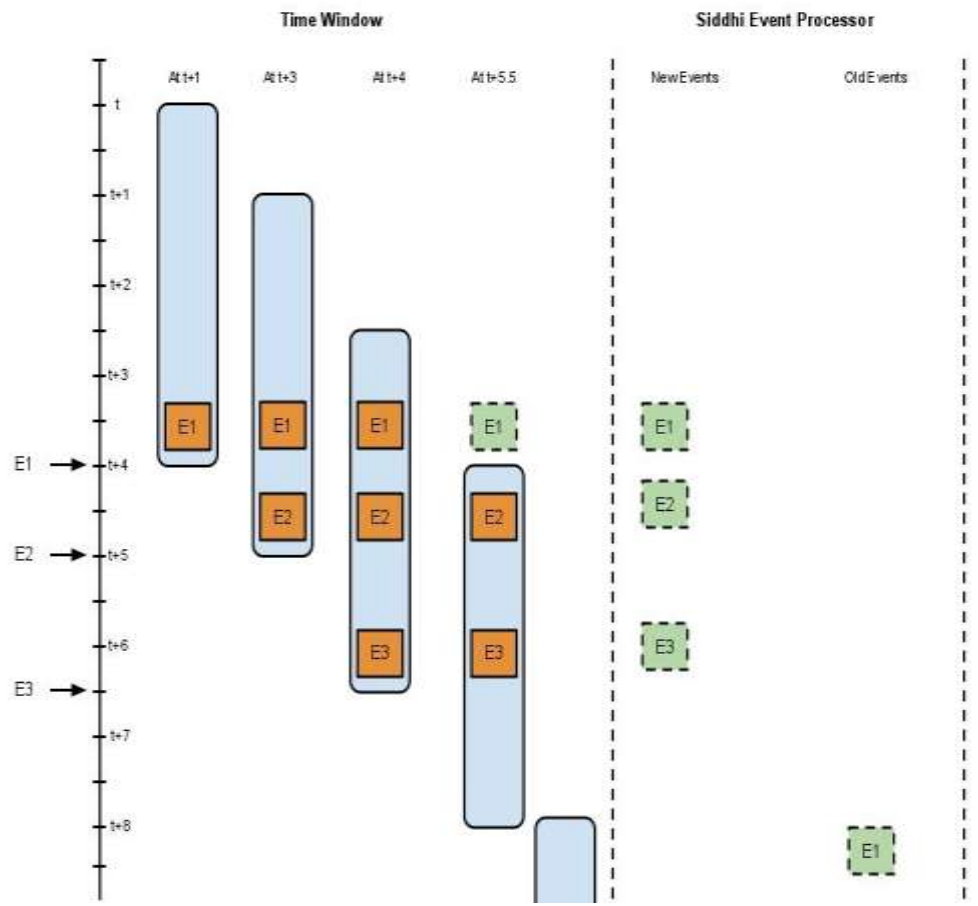
```
Query query = qf.createQuery(
    "StockQuote",
    qf.output("symbol=StockStream.symbol","price=avg(StockStream.price)"),
    qf.inputStream(stockStream),
    qf.and(
            qf.condition("StockStream.symbol", EQUAL, "IBM"),
            qf.condition("StockStream.volume", GREATERTHAN, "10000")
    )
);
```

- Evaluate the tree, and optimizations to stop the evaluations as soon as possible



IMPLEMENTING WINDOWS

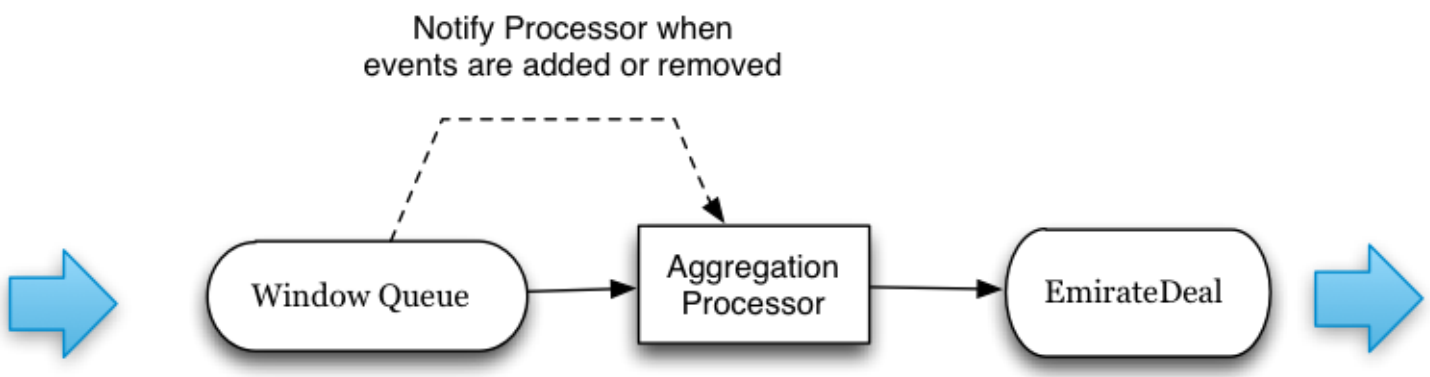
- Implemented as a queue
- Use events to notify the windows event addition and removals using a pub/sub pattern to processors



AGGREGATION

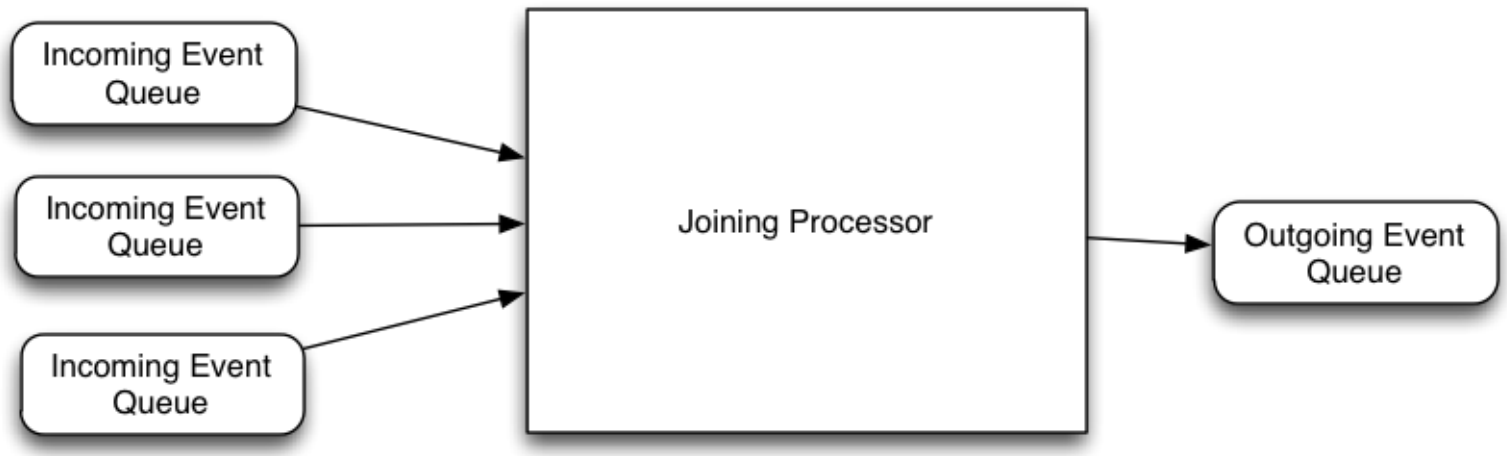
- Siddhi supports avg, sum, count, max, min

```
Query query = qf.createQuery(
        "StockQuote",
        qf.output("symbol=CSEStream.symbol", "avgPrice=avg(CSEStream.price)"),
        qf.inputStream(cseEventStream),
        qf.condition("CSEStream.price", GREATERTHAN_EQUAL, "20")
);
query.groupBy("CSEStream.symbol");
query.having(qf.condition("StockQuote.avgPrice", GREATERTHAN, "50"));
```



- These are implemented within processor and each addition or removal of a event updates the value (if it is a window based, queues notify subscribers when thing change).

IMPLEMENTING JOINS



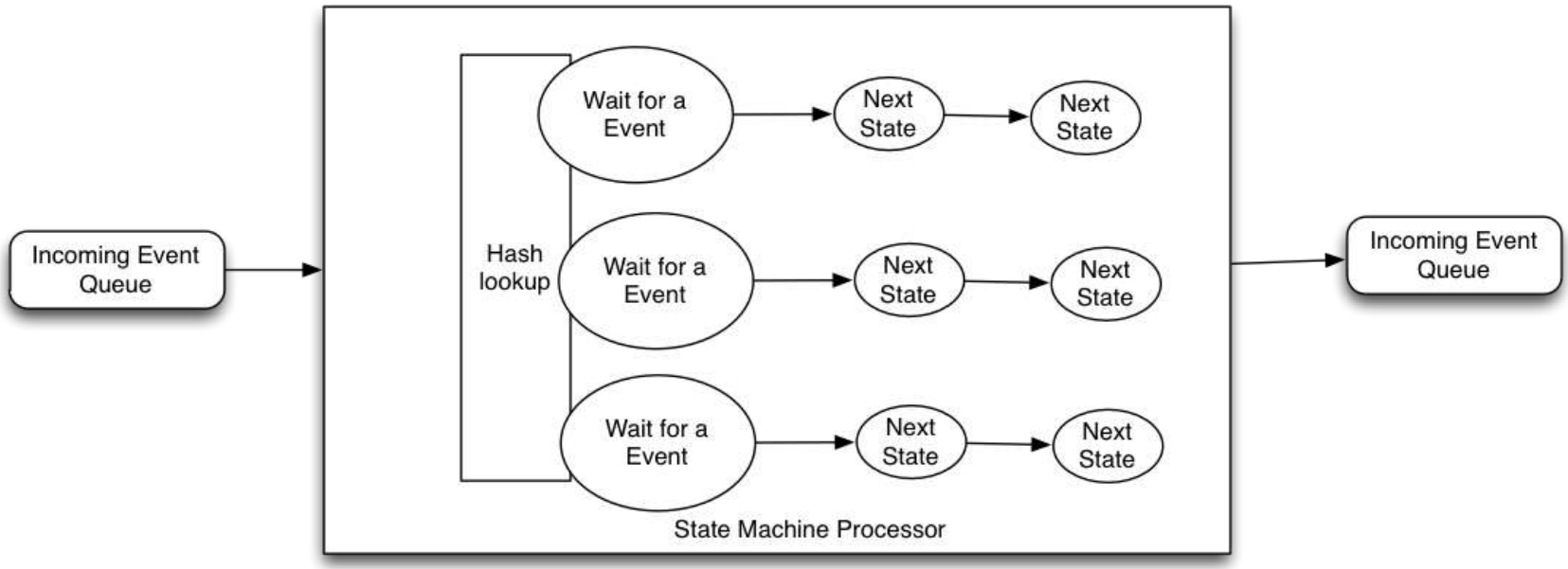
- Implemented as a processor that moves data from several queues to a different queue after joining
- When a event arrives, if it matches the join condition, we send them over to the next queue, else keeps them to match with the future events.
- We keep them as long as the window condition allows (this is done by keeping references to queues, and checking within them for matching)

PATTERNS AND SEQUENCES

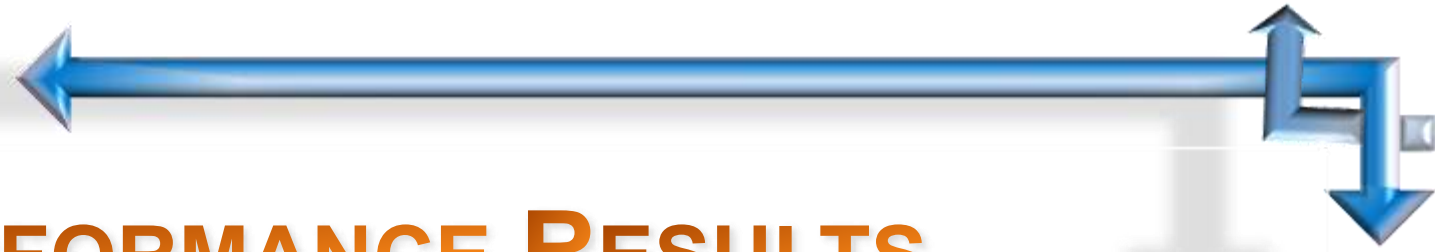
- Patterns and sequences handle series of events like A->B->C, there are two classes patterns and sequences
- Patterns matches with events ignoring events in between while sequences matches exactly. You can use star (*) with sequences to mimic patterns
- Supported through a state machine (NFA)

```
Query query = qf.createQuery(
        "StockQuote",
        qf.output("priceA=$0.first.price", "priceA=$0.last.price", "priceB=$1.price"),
        qf.inputStreams(cseEventStream, infoStock),
        qf.sequence(
                qf.star(
                        qf.condition("CSEStream.price", GREATERTHAN, "$0.prev.price")
                ),
                qf.condition("CSEStream.price", LESSTHAN, "$0.last.price")
        )
);
```

Implementing Patterns and Sequences



- Use a event model
- Say we need A->B->C, then we break down it to list of listeners, which when triggered remove itself and registers the next in the list.
- We apply different variations of this to support *, and sequences .



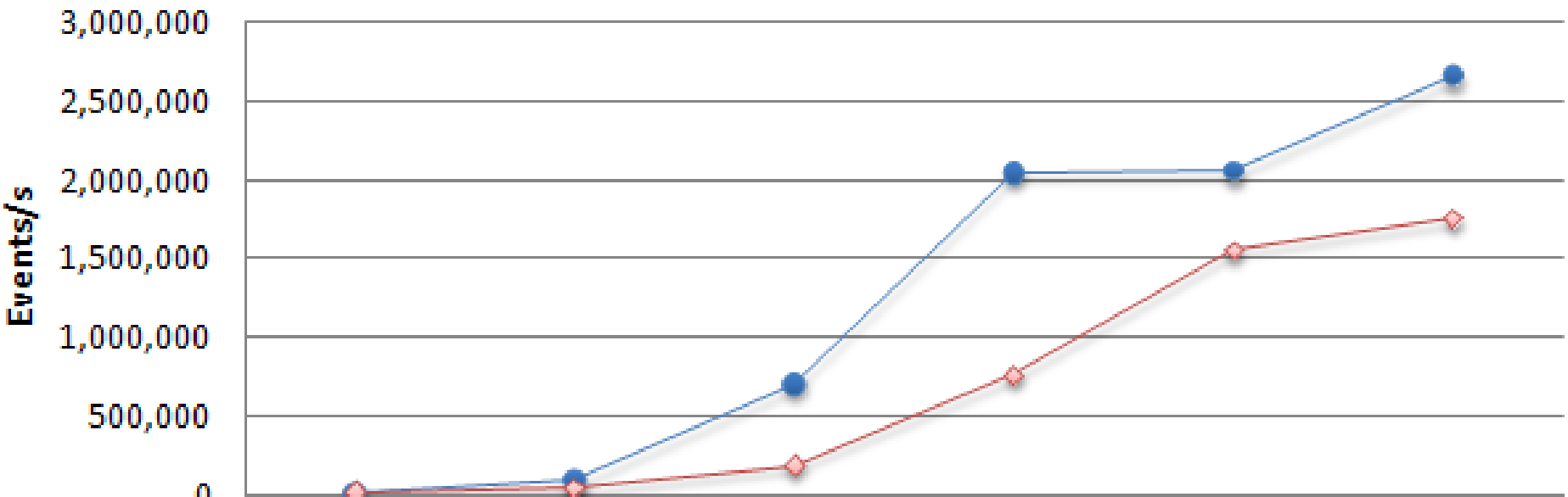
PERFORMANCE RESULTS

- We compared Siddhi with Esper, the widely used opensource CEP engine
- For evaluation, we did setup different queries using both systems, push events in to the system, and measure the time till all of them are processed.
- We used Intel(R) Xeon(R) X3440 @2.53GHz , 4 cores 8M cache 8GB RAM running Debian 2.6.32-5-amd64 Kernel

PERFORMANCE COMPARISON WITH ESPER

Simple filter without window

```
select symbol, price
from StockTick(price>6)
```

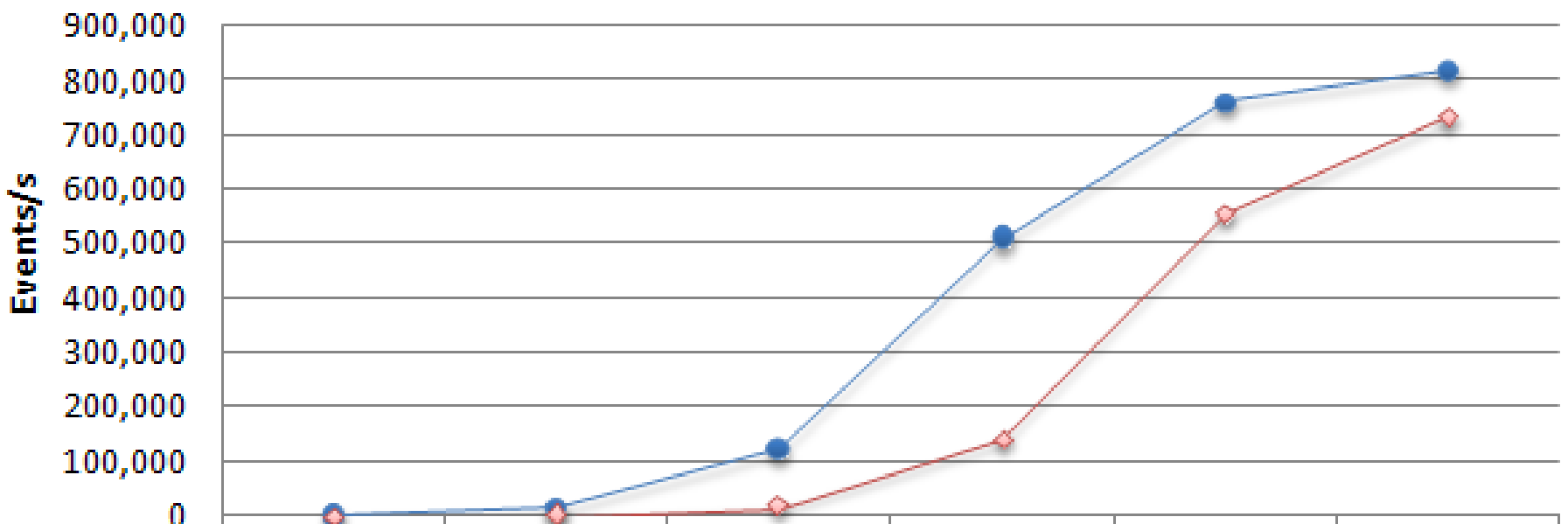


# of Events	10	100	1,000	10,000	100,000	1,000,000
Siddhi	21,739	92,593	704,225	2,053,388	2,061,856	2,662,336
Esper	16,667	36,900	185,185	760,456	1,560,062	1,752,265

PERFORMANCE COMPARISON WITH ESPER

Average calculation of a given symbol within time window

```
select irstream symbol, price, avg(price)
from StockTick(symbol='IBM').win:time(.005)
```

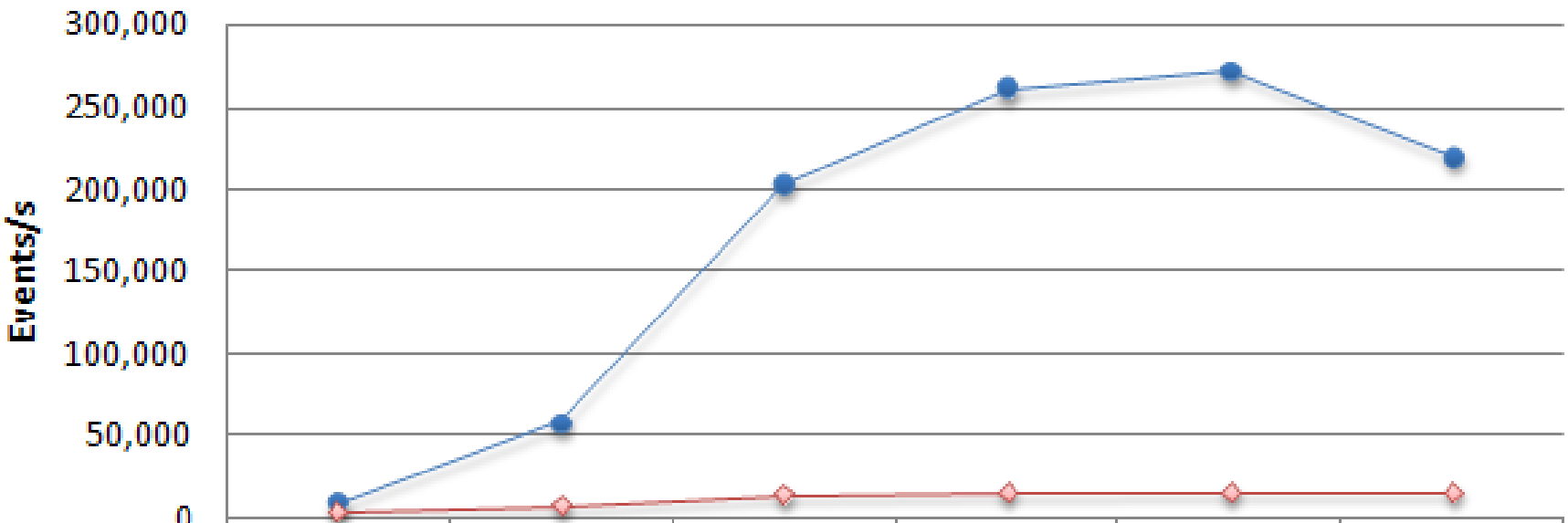


# of Events	10	100	1,000	10,000	100,000	1,000,000
Siddhi	1,613	15,060	122,249	509,424	763,359	820,392
Esper	201	1,838	17,734	142,837	556,824	732,982

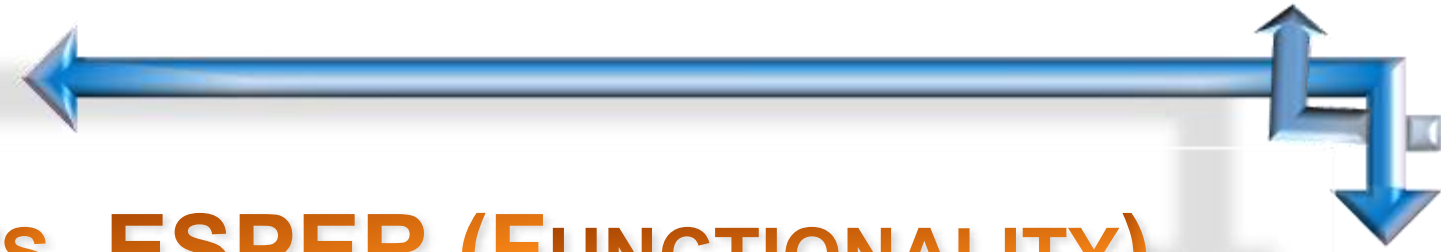
Performance Comparison With ESPER

State machine query for pattern matching

```
select f.symbol, p.accountNumber, f.accountNumber
from pattern [every f=FraudWarningEvent2 -> p=PINChangeEvent2(accountNumber
                                                    = f.accountNumber)]
```



# of Events	10	100	1,000	10,000	100,000	1,000,000
Siddhi	9,709	58,824	204,082	261,097	273,187	220,612
Esper	3,401	6,653	13,676	15,264	15,466	15,623



SIDDHI VS. ESPER (FUNCTIONALITY)

- Following describe Siddhi in contrast to Esper using the comparisons defined in Cugola et al.
- In terms of Functional and Processing Models and Data, Time, and Rule Models (see Table I and III of Cugola et al.), Esper and Siddhi behave the same.
- In terms of Supported language model (table IV),
  - Esper supports Pane and Tumble windows, User Defined operators and Parameterization while Siddhi does not support them
  - Siddhi supports removal of duplicates that is not supported by Siddhi.
- Both support all other language constructs.

FEATURES WE DID NOT DISCUSSED

- Improvements in the Selection Operator
  - using MVEL
- Support conditions on the aggregation functions
  - having, group-by
- Event quantifications in sequential patterns
  - kleene closure (*)
- Unique function

CURRENT USERS OF SIDDHI

- "Los Angeles Smart Grid Demonstration Project"
  - It forecasts electricity demand, respond to peak load events, and improves sustainable use of energy by consumers. http://ceng.usc.edu/~simmhan/pubs/simmhan-usctr2011-smartgridinformatics.pdf
- Open MRS NCD module – idea is to detect and notify patient when certain conditions have occurredhttps://wiki.openmrs.org/display/docs/Notifiable+Condition+Detector+%28NCD%29+Module
- WSO2 CEP Server (Perspective)

FUTURE WORK

http://www.flickr.com/photos/garryknight/3650151941/

- SQL like query language for Siddhi
- Building scalable processing network using Siddhi nodes
  - Siddhi is currently a java library, build a Thrift based server for Siddhi
  - Supporting a network of Siddhi processors (clustering)
  - Query partition and optimization for scale
- Support for Pane and Tumble windows, User Defined operators and Parameterization
- Available under Apache License

Immediate Plans

- Have resource allocations and development efforts to continue next year
- Looking forward to build a opensource community around it
- Looking for usecases, real data feeds, and of course Guinea pigs

Siddhi-CEP

QUESTIONS?