Java is a trademark of Sun Microsystems, Inc.

JavaOne℠

Design Patterns for Complex Event Processing (CEP)

Alexandre Alves

Shailendra Mishra

Oracle Corporation

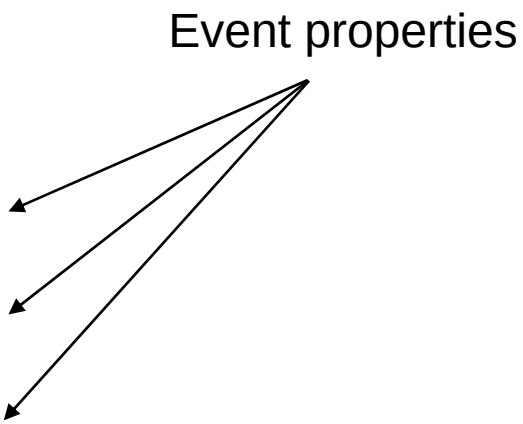
Agenda

- **Building blocks**
- Design Patterns
- Conclusion

Building blocks

- EVENT
  - Defined by a schema (i.e. event type)
  - Tuple of event properties

StockEventType	
symbol	string
lastBid	float
lastAsk	float



Building blocks

- STREAM
  - Time ordered sequence of events in time
  - APPEND-only
    - One cannot remove events, just add them to the sequence
  - Unbounded
    - There is no end to the sequence

    {event1, event2, event3, event4, …, eventN}

Building blocks

- STREAM
  - Examples:
    - {{1s, event1}, {2s, event2}, {4s, event3}}
      - Valid STREAM
    - {{1s, event1}, {4s, event2}, {2s, event3}}
      - Not a STREAM, this is a EVENT CLOUD.

Building blocks

- RELATION
  - Bag of events at some instantaneous time T
  - Allow for INSERT, DELETE, and UPDATE
  - Example:
    - At T=1: {{event1}, {event2}, {event3}}
    - At T=2: {{event1}, {event3}, {event4}}
      - No changes to event1 and event3
      - Event2 was deleted
      - Event4 was inserted

Building blocks

- OPERATORS
  - **Transform STREAMS and RELATIONS**
  - **Types of operators:**
    - **RELATION to RELATION**
      - **Well-known from RDBMS (e.g. project, filter, join)**
    - **STREAM to RELATION**
      - **WINDOW operator bounds STREAMS into RELATIONS (e.g. NOW)**
    - **RELATION TO STREAM**
    - **STREAM to STREAM**

Summary

- CEP is about continuous processing of online streaming events
- STREAM is a APPEND-only time-ordered sequence of events
- RELATION is a INSERT/DELETE/UPDATE-able bag of events at some time *t*

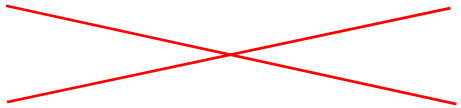
Design Patterns

- Event Filtering
- New event detection
- Event partitioning
- Event enrichment
- Event aggregation
- Event correlation
- Application-time events
- Missing event detection

1. Event filtering

- Problem:
  - Look for specific data on a stream, dropping the unwanted events.
- Scenario:
  - Consider STOCKSTREAM a stream of stock events (symbol, lastBid, lastAsk)
  - Look for all stocks whose symbol is ‘AAA’

1. Event filtering

Time	Input	Output
1	{“AAA”, 10.0, 10.5}	{“AAA”, 10.0, 10.5}
2	{“AAA”, 10.0, 10.5}	{“AAA”, 10.0, 10.5}
3	{“BBB”, 11.0, 12.5}	

1. Event filtering

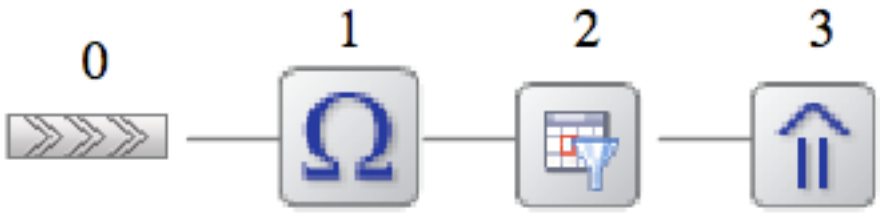
- Solution:

Specify STREAM source

Specify a WINDOW operator

```
SELECT *
FROM stockstream [NOW]
WHERE symbol = 'AAA'
```

Define predicate for filtering



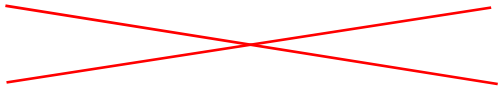
1. Event filtering

- Why do we need the window operator [NOW]?
  - Filtering works on relations, hence we need to convert the stock stream into a relation.
  - Logically this makes sense, as you cannot work on a unbound set of things, you need to constraint it to a bounded set.
  - Note that in some cases short-cuts are allowed; for example, where [NOW] is assumed if not specified.

2. New event detection

- Problem:
  - Look for specific data on a stream, notify only if it is new data
- Scenario:
  - Report only those stocks whose price have changed since the last event

2. New event detection

Time	Input	Output
1	{“AAA”, 10.0, 10.5}	{“AAA”, 10.0, 10.5}
2	{“AAA”, 10.0, 10.5}	
3	{“BBB”, 11.0, 12.5}	{“BBB”, 11.0, 12.5}

2. New event detection

- SELECT * FROM stockstream [NOW]
  - Generates RELATION or STREAM?
  - Generates a RELATION, which includes all events at time *t*
  - What we need are the events that exist at time *t*, but do not exist at time (*t - 1*)
  - Use RELATION-STREAM operator **ISTREAM** (insert stream)
  - As we are interested on the last event, use the window operator **[ROWS 1]** instead of **[NOW]**.

2. New event detection

- Solution

```
ISTREAM(
   SELECT *
   FROM stockstream [ROWS 1]
)
```

3. Event partitioning

Although similar to event at t=2, it is != then event at t=3…

- There is a problem with the current solution…

Time	Input	Output
1	{“AAA”, 10.0, 10.5}	{“AAA”, 10.0, 10.5}
2	{“AAA”, 10.0, 10.5}	
3	{“BBB”, 11.0, 12.5}	{“BBB”, 11.0, 12.5}
4	{“AAA”, 10.0, 10.5}	{“AAA”, 10.0, 10.5}

3. Event partitioning

- We need to partition the window by stock symbol

```
ISTREAM (
   SELECT *
   FROM stockstream
     [PARTITION BY symbol ROWS 1]
)
```

3. Event partitioning

Time	Input	Output
1	{“AAA”, 10.0, 10.5}	{“AAA”, 10.0, 10.5}
2	{“AAA”, 10.0, 10.5}	
3	{“BBB”, 11.0, 12.5}	{“BBB”, 11.0, 12.5}
4	{“AAA”, 10.0, 10.5}	

4. Event enrichment

- Problem:
  - Enrich events from a stream with (somewhat) static data from a relation.

- Scenario:
  - Consider STOCKRELATION (relation):

    symbol,
    company full-name,
    headquarters' location

  - For every stock event from STOCKSTREAM whose lastBid is greaten than 5.0, find its company's location.

4. Event enrichment

Query is trigged by stream, and not by relation

STOCKRELATION

Symbol	Full-name	Location
AAA	The AAA company	San Francisco
BBB	The BBB company	San Jose

Time	Input (STOCKSTREAM)	Output (QUERY)
1	{“AAA”, 10.0, 10.5}	{“AAA”, “San Francisco”}
2	{“BBB”, 11.0, 12.5}	{“BBB”, “San Jose”}
3	{“AAA”, 4.0, 4.5}	

4. Event enrichment

- Solution:

```
SELECT event.symbol, location
FROM stockstream [NOW] AS event,
        stockrelation AS data
WHERE event.symbol = data.symbol AND
        event.lastBid > 5.0
```

  - Why do we need to specify a WINDOW operator for the STREAM and not for the RELATION?
    - Because a RELATION is instantaneous, it is already bound to time *t.*

5. Event aggregation

- Problem:
  - Aggregate several (simple) events from a stream into a single new (complex) event summarizing event properties of the simple events
- Scenario:
  - Output the average bid and ask price of a stock of the last 10 seconds.

5. Event aggregation

The event for AAA is also included in the result

Time	Input	Output
1s	{“AAA”, 10.0, 12.0}	{“AAA”, 10.0, 12.0}
4s	{“AAA”, 12.0, 14.0}	{“AAA”, 11.0, 13.0}
9s	{“BBB”, 4.0, 5.0}	{“AAA”, 11.0, 13}, {“BBB”, 4.0, 5.0}
15s	{“BBB”, 8.0, 10.0}	{“BBB”, 6.0, 7.5}
20s		{“BBB”, 8.0, 10.0}

> 10s

5. Event aggregation

- Solution:

```
SELECT symbol, AVG(lastBid), AVG(lastAsk)
FROM stockstream [RANGE 10 seconds]
GROUP BY symbol
```

  - The RANGE WINDOW operator is a STREAM to RELATION operator
    - It defines a range of time in which events are kept, events older than range are removed from window.
  - As we are interested on the average price per symbol, we need to use the GROUP BY operator.

6. Event correlation

- Problem:
  - Correlate events from one or several streams on a common set of values, that is, a common pattern.
- Scenario:
  - Consider a BIDSTREAM and ASKSTREAM, respectively containing bid (symbol, bidPrice, customer) and ask requests (symbol, askPrice, customer).
  - Correlate bids with asks that are related to same symbol and occurring within 10 seconds of each other; anything older is considered stale.

6. Event correlation

Time	Input (BIDS)	Input (ASKS)	Output
1s	{“AAA”, 12.0, cust1}	{“BBB”, 9.0, cust2}	
5s	{“AAA”, 10.0, cust3}	{“AAA”, 10.0, cust2}	{“AAA”, 10.0, cust3, 10.0, cust2} {“AAA”, 12.0, cust1, 10.0, cust2}
15s	{“BBB”, 10.0, cust4}	{“CCC”, 11.0, cust5}	{“AAA”, 10.0, cust3, 10.0, cust2}
20s		{“BBB”, 10.0, cust6}	{“BBB”, 10.0, cust4, 10.0, cust6}

> 10s

6. Event correlation

- Solution:

```
SELECT bid.symbol, bidPrice, bid.cust, askPrice, ask.cust,
FROM bidstream [RANGE 10 seconds] AS bid,
        askstream [RANGE 10 seconds] AS ask
WHERE bid.symbol = ask.symbol
```

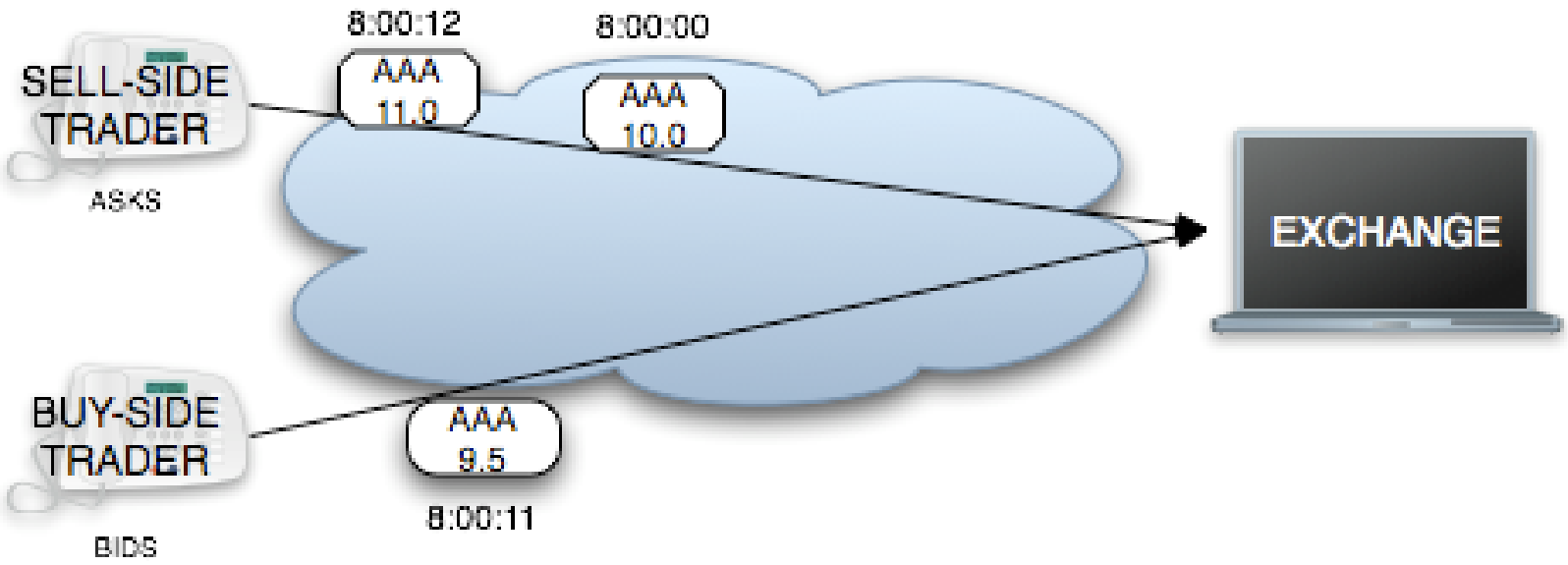
  - You can perform regular join operations between streams, including outer joins.

7. Application time

- Problem:
  - Need to use application's view of time, instead of CPU wall-clock.
  - This is particularly useful when events originate from different machines and need some way of synchronizing.
- Scenario:
  - Again, consider bid and ask stream…
  - However, bid and ask requests are time-stamped on trader at the time that the order is placed.

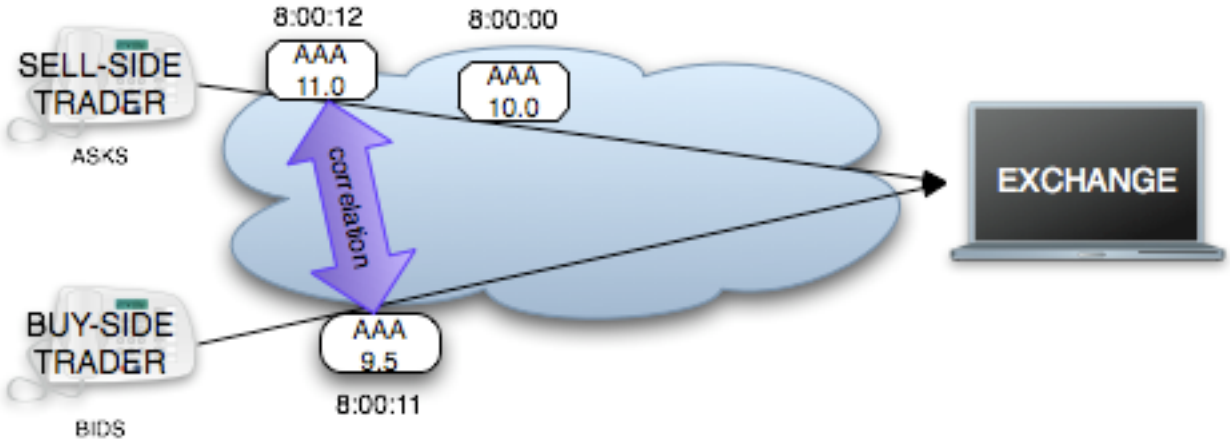
7. Application time

- Seller places two ask requests respectively at time 8:00:00 and 8:00:12
- Buyer places one bid request at time 8:00:11



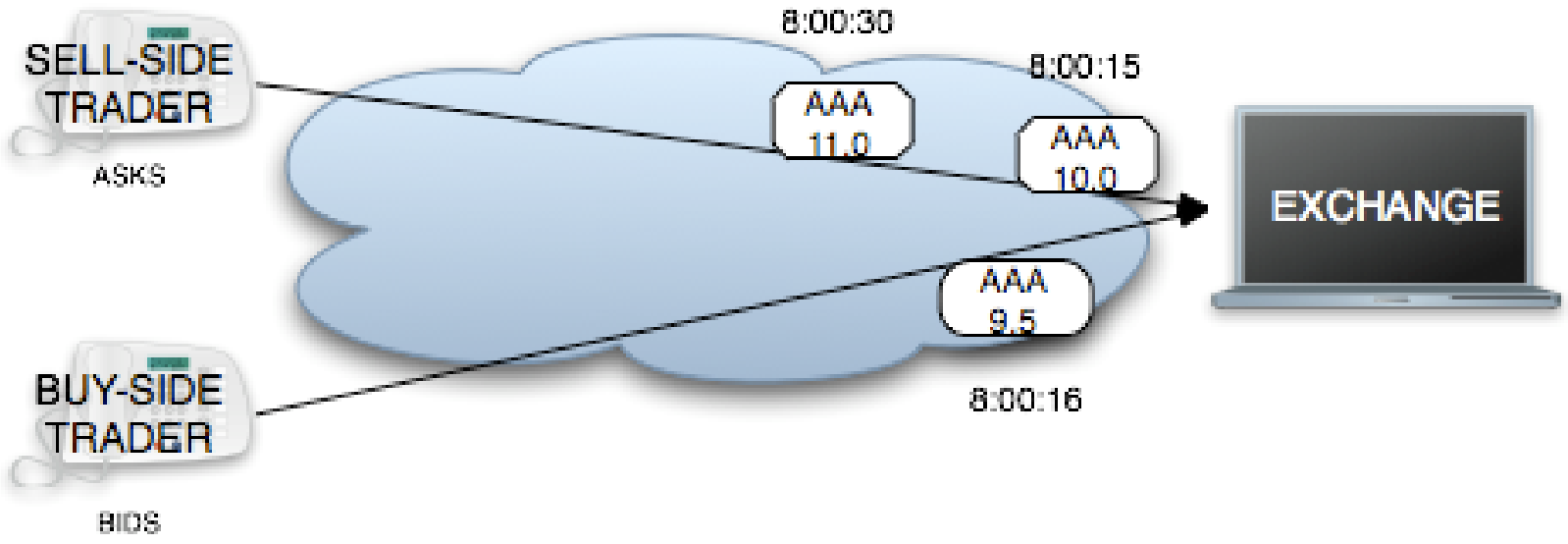
7. Application time

- Remember that we want to correlate using a 10 seconds window, as anything older is stale…
- Hence first event from seller should not be considered, and we should correlate the ask price of 11.0 with the bid price of 9.5



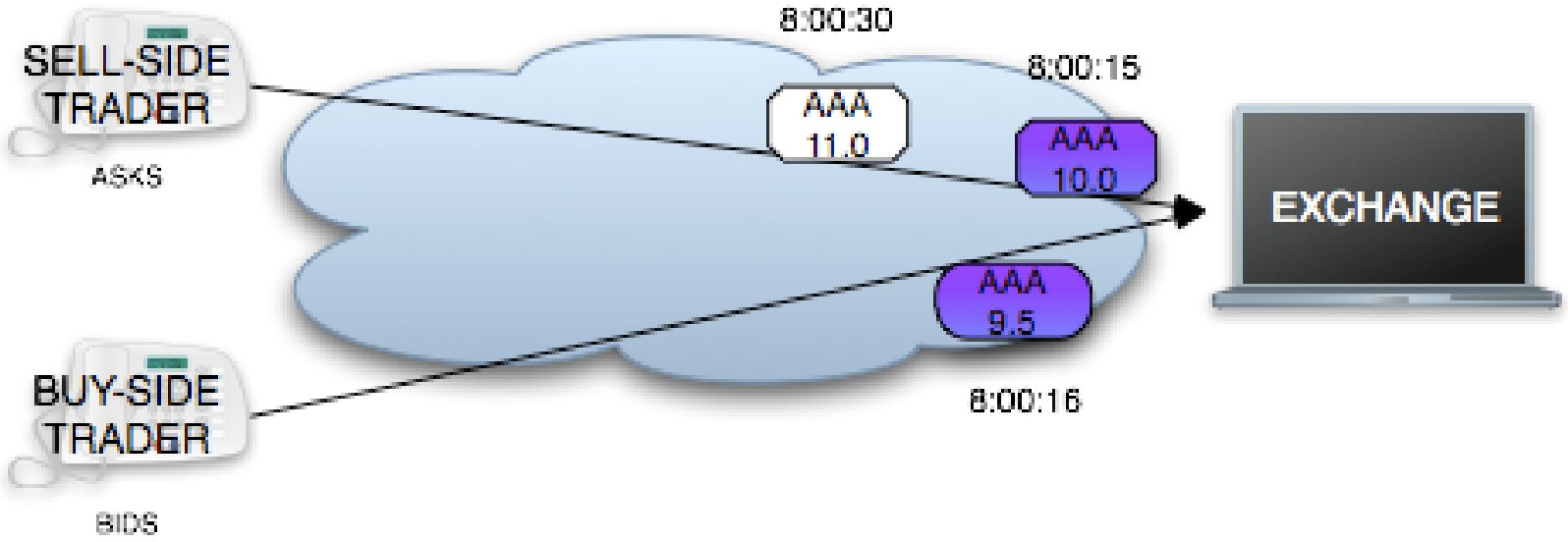
7. Application time

- However, the reality is that the first two events could arrive together in a burst in the exchange, and the third could be delayed in the cloud…



7. Application time

- In this case, bid price of 9.5 would correlate to ask price of 10.0 and the exchange would loose money as the spread is lower…



7. Application time

- Solution:
  - The query does not change…
  - However STREAMS must be configured to use application time-stamps based upon some event property, instead of having events being system-timestamped…

BidEventType	
symbol	string
customer	string
bidPrice	float
orderTime	dateTime

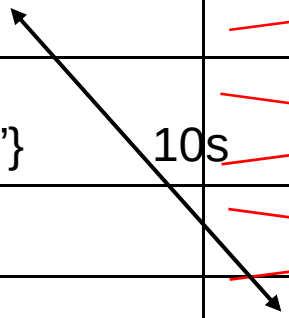
8. Missing event detection

- Problem:
  - Alert if an expected event is missing after some amount of time.

- Scenario:
  - Consider a SALESTREAM stream, containing transaction type (customer order or shipment), and order id.
  - Alert if an order is not followed by a shipment within 10 seconds.

8. Missing event detection

Time	Input	Output
1s	{1, “ORDER”}	
5s	{2, “ORDER”}	
10s	{1, “SHIPMENT”}	
15s	{3, “ORDER”}	
15+t		{“DELAYED”, 2}
20s	{3, “SHIPMENT”}	



8. Missing event detection

- Solution:

```
SELECT  "DELAYED" as alertType, orders.orderId,
FROM salestream MATCH_RECOGNIZE (
        PARTITION BY orderId

        MEASURES
                CustOrder.orderId AS orderId

        PATTERN (CustOrder NotTheShipment*) DURATION 10 SECONDS

        DEFINE
                CustOrder AS (type = 'ORDER'),
                NotTheShipment AS ((NOT (eventType = ' SHIPMENT')))

) AS orders
```

Conclusion

- Online processing, that is, no need to store to disk first
- Declarative approach
- Able to deal with unbounded data-sets (streams)
- Powerful temporal constructs
- Able to find complex relationships using pattern matching
- Leverage Relational algebra from DBMS by means of converting STREAM-RELATION and RELATION-STREAM
- Rich library of event processing design patterns

Thank You

Alexandre Alves
Shailendra Mishra
Oracle Corporation