

Siddhi: A Complex Event Processing Engine

Presenter:

Rajrup Ghosh

M.Tech. Computational Science,
CDS, IISc Bangalore

15-JAN-16

©DREAM:Lab, 2014

This work is licensed under a [Creative Commons Attribution 4.0 International License](https://creativecommons.org/licenses/by/4.0/)





Introduction

- Event Stream and Event Stream Processing
- Complex Event Processing (CEP)
- Why to use Siddhi – CEP?
- How to use Siddhi?
- A quick look into Siddhi Architecture
- Siddhi + MQTT
- Hands On demo on event processing using CEP



Event Stream

- An event stream is a sequence of events ordered with time.
- One or more event streams can be imported and manipulated using queries.
- Identification of patterns on these events is required.
- Examples:
 - Stock quotes
 - Click streams
 - Sensor network data

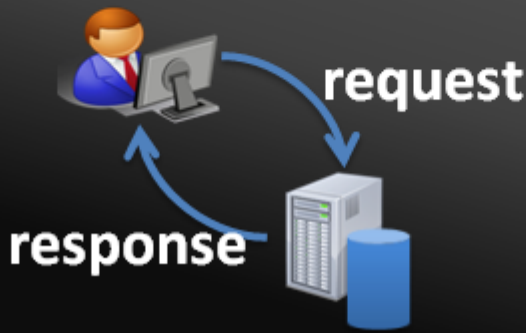
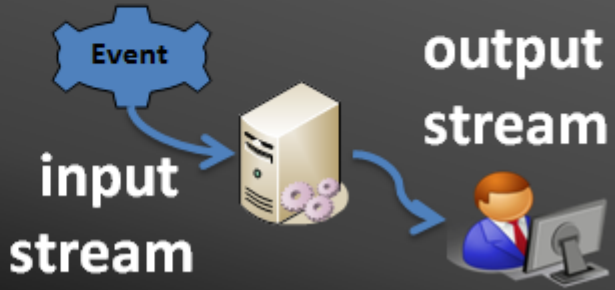


Event Processing

- **Event processing** is a method of tracking and analyzing (processing) streams of information (data) about things that happen (events) and deriving a conclusion from them.
- Events may be of varied form such as data from environmental sensors.
- Finding patterns in this data requires complex event processing.



Database vs Event-driven Applications

	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
		



Complex Event Processing

- **Complex event processing** or **CEP**, is event processing that combines data from multiple sources to infer events or patterns that suggest more complicated circumstances.
- In abstract, the tasks of the CEP is to identify meaningful patterns, relationships and data abstractions among unrelated events and fire an immediate response such as an Alert message.



Complex Event Processing Use Cases



Utilities

Meter alarm filtering
Power restoration confirmation
Mobile work unit tracking



Homeland Security

Threat detection, sensor data
correlation, emergency response



Telecommunications &
Services

Fraud Detection,
digital marketing



Financial Services

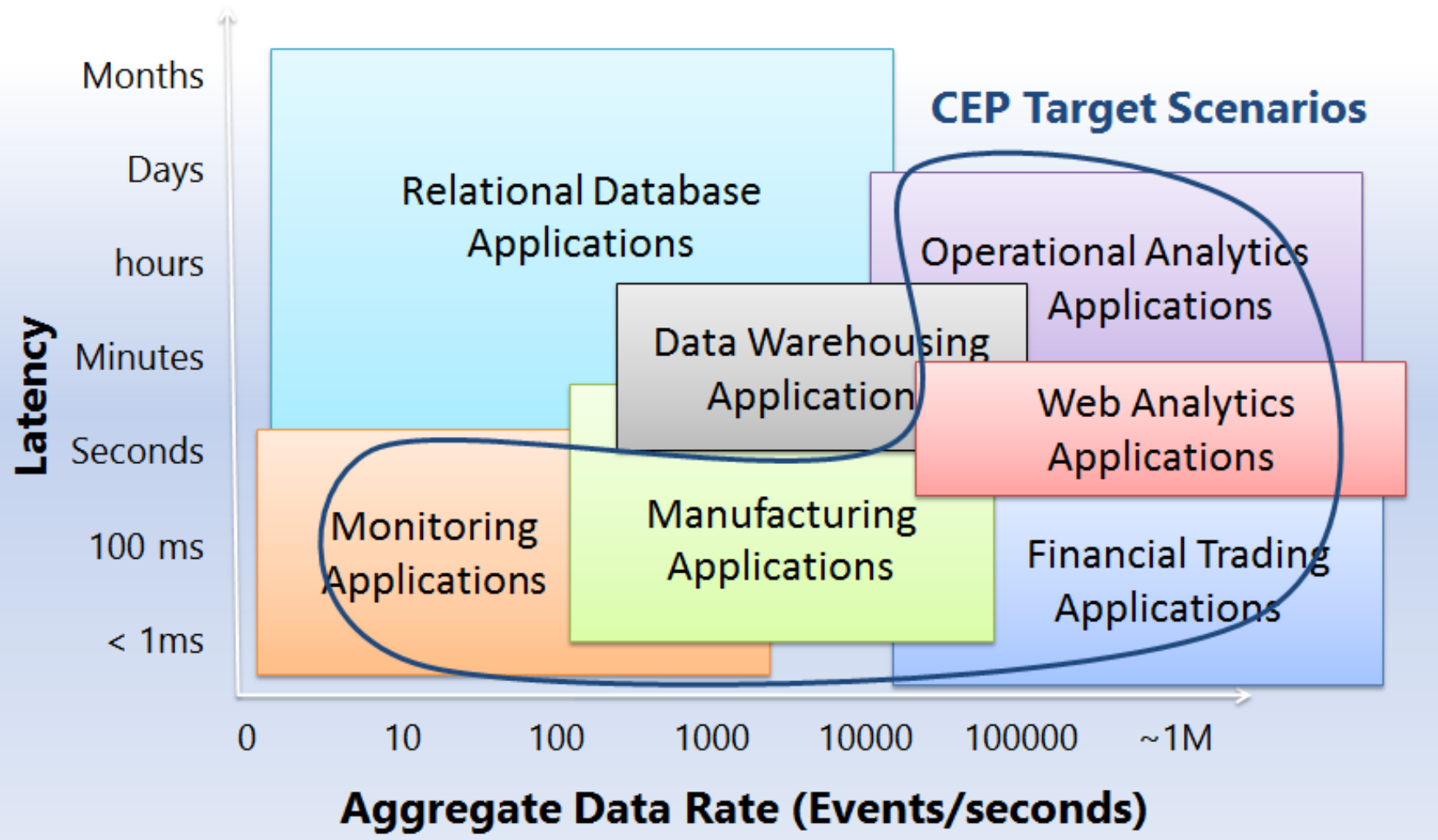
Algorithmic trading



- *Filter, correlate and aggregate events from high volume streams with consistent low latency*
- *Handle disparate event and data sources*
- *Facilitate time window processing*
- *Recognize and act on complex patterns*
- *Manage your data before it reaches your database*
- *Protect your core business processes from the "data tsunami"*



Scenarios of Complex Event Processing





Current CEP Solutions

- S4
- STREAM
- Esper
- SASE
- SAP ESP
- Oracle Event Processing
- HiFi
- Aurora
- CompAS.
- Niagara



Amazon Kinesis

- Amazon Kinesis Streams is a CEP solution from Amazon.
- Amazon Kinesis Streams can continuously capture and store terabytes of data per hour from sources such as
 - » website clickstreams
 - » financial transactions
 - » social media feeds
 - » IT logs
 - » location-tracking events.
- Supported as a service from Amazon EC2.



Amazon Kinesis: Benefits

- Amazon Kinesis Client Library (KCL) support, which allows streaming data to power real-time dashboards, generate alerts, implement dynamic pricing and advertising, and more.
- Emit data from Amazon Kinesis Streams to other AWS services such as
 - Amazon Simple Storage Service (Amazon S3)
 - Amazon Redshift
 - Amazon Elastic Map Reduce (Amazon EMR)
 - AWS Lambda.

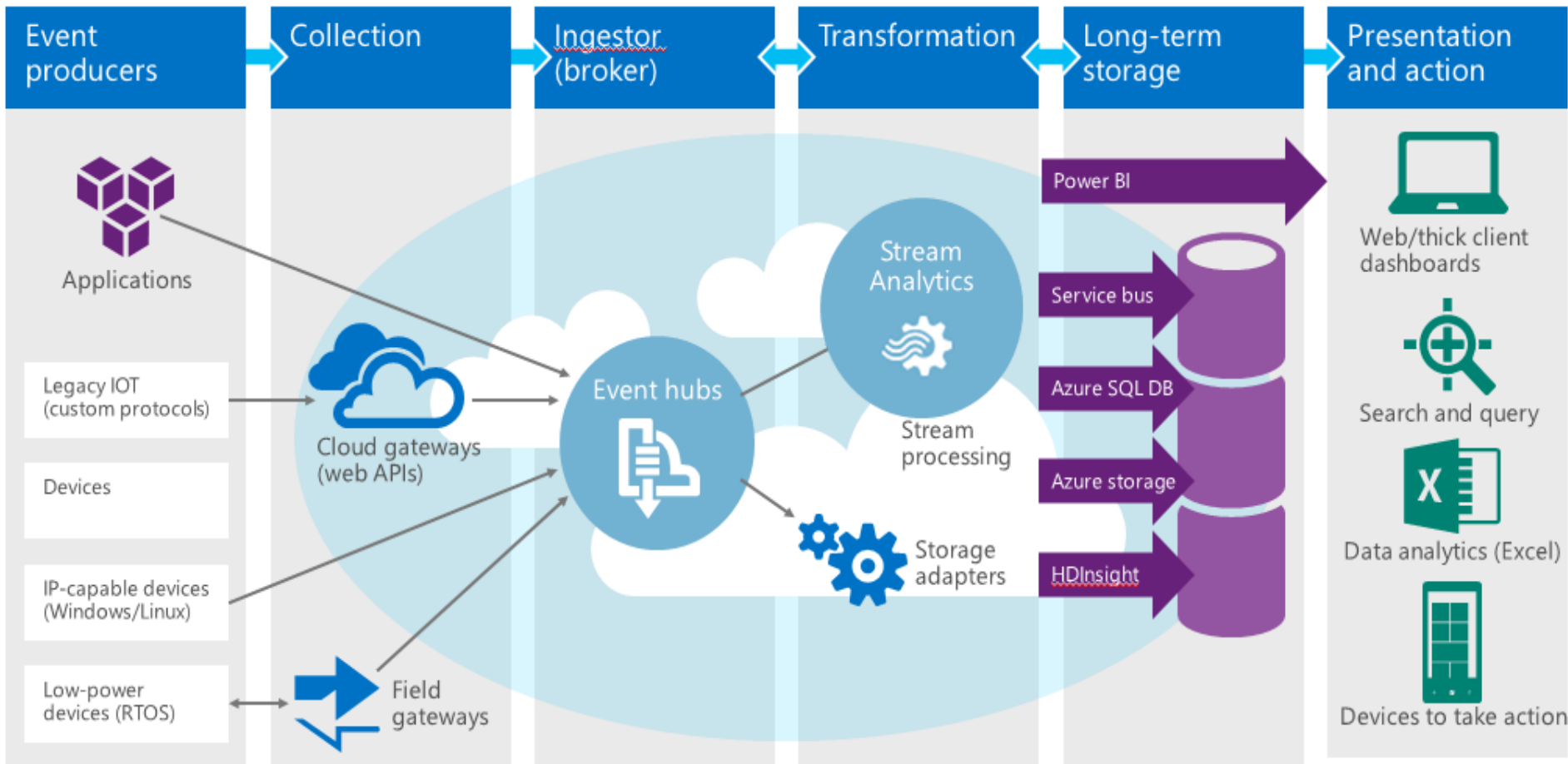


Azure Streaming

- Real time processing engine in the cloud.
- Service provided by Microsoft.
- Advantages:
 - Real-time analytics for IoT solutions
 - Stream millions of events per second
 - Achieve mission-critical reliability and scale
 - Real-time dashboards and alerts over data
 - Correlate across multiple streams of data
 - Use SQL-based language for rapid development



End to End processing on Microsoft Azure





Problems with current CEP solutions

- Many are **proprietary**
- Not enough support for complex queries
- Less efficient
- High latency
- High memory consumption

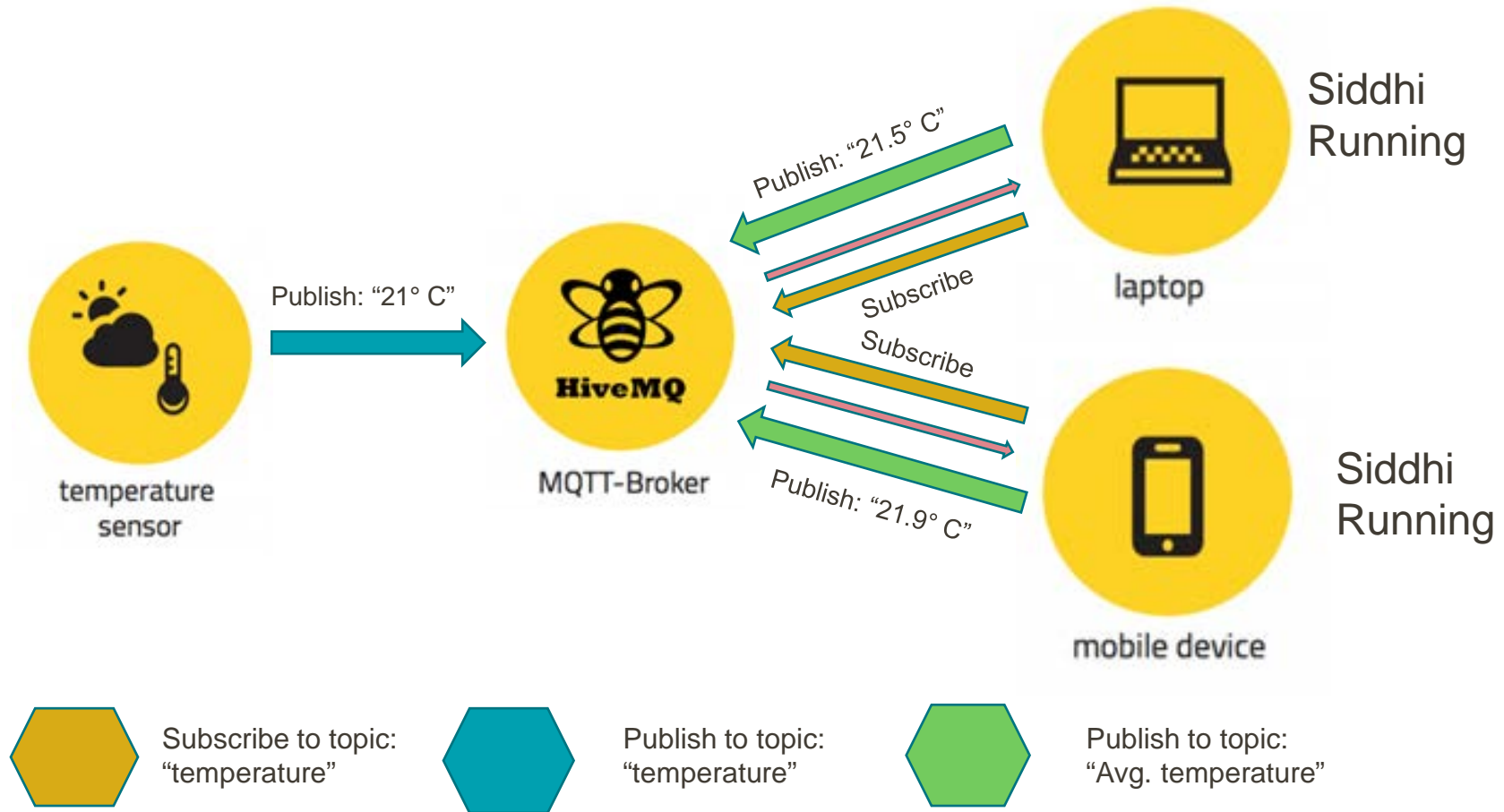


Advantages of using Siddhi

- Siddhi is **open-source**
- A lightweight engine for identifying complex events
- Useful to deploy in low end computational resources such as:
 - » Raspberry Pi
 - » UAVs
 - » mobile phones, etc.
- Supports a query language to detect patterns in events.



How can we apply Siddhi CEP





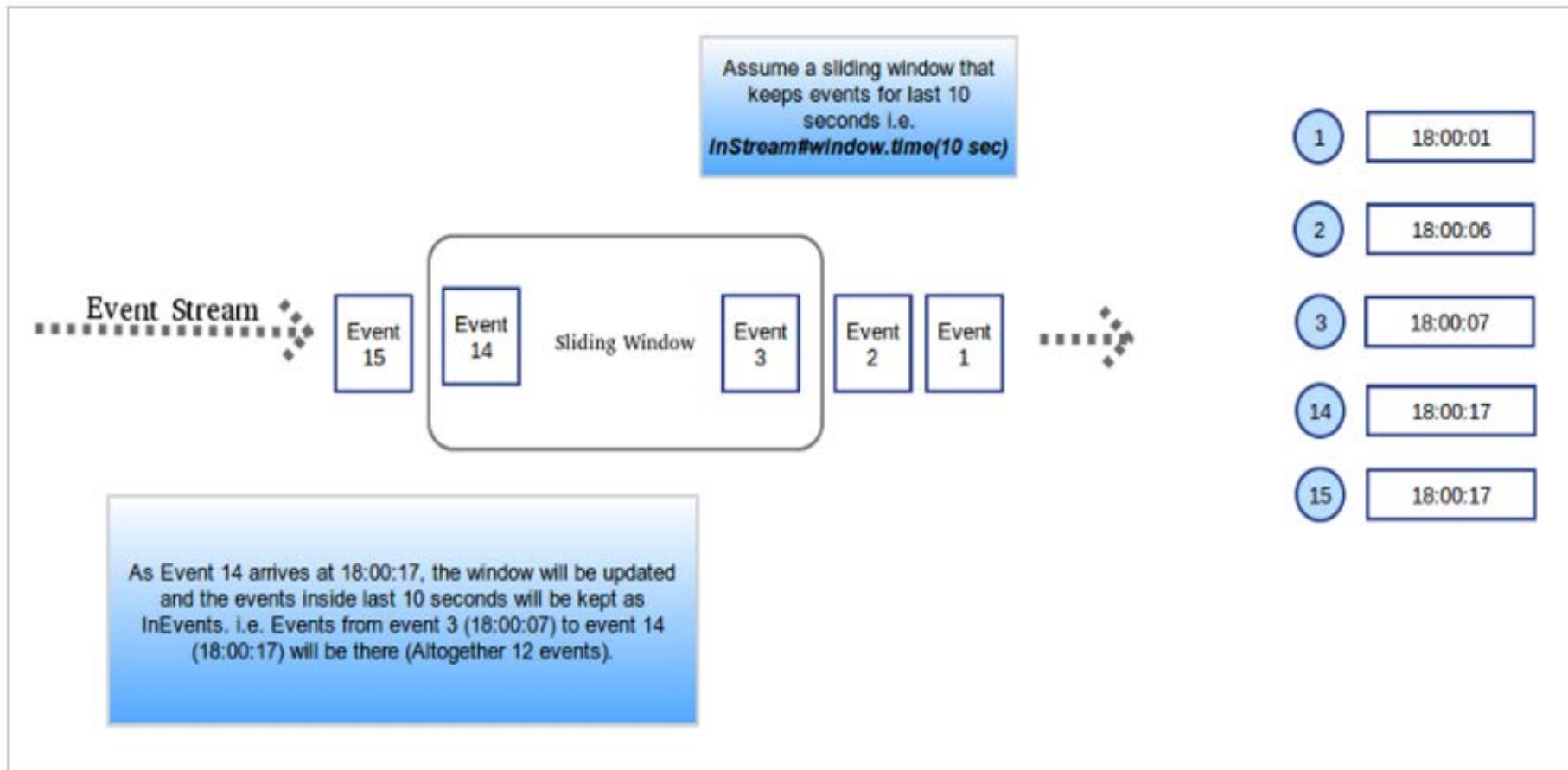
Siddhi CEP Queries

- Selection or filtering and projection (like select in SQL)
- Filter query creates an output stream and inserts any events from the input stream that satisfies the conditions defined.
- Filters support following types of conditions
 - » >, <, ==, >=, <=, !=
 - » contains, instanceof
 - » and, or, not

```
String executionPlan = "" +  
    "define stream cseEventStream (height int); " +  
    "" +  
    "@info(name = 'query1') " +  
    "from cseEventStream[height < 300 ] " +  
    "select height " +  
    "insert into outputStream ;";
```



- Windows – events are processed within a window (e.g. for aggregation).
 - » Time window
 - » Length window
 - » Time batch window
 - » Length batch window





Window Query Examples

- Window Query:
 - Length Sliding Window

```
String executionPlan = "" +  
    "define stream cseEventStream (height int); " +  
    "" +  
    "@info(name = 'query1') " +  
    "from cseEventStream #window.length(6) " +  
    "select avg(height) as avgHt " +  
    "insert into outputStream ;";
```

- Length Batch Window

```
String executionPlan = "" +  
    "define stream cseEventStream (height int); " +  
    "" +  
    "@info(name = 'query1') " +  
    "from cseEventStream #window.lengthBatch(6) " +  
    "select avg(height) as AvgtHeight " +  
    "insert into outputStream ;";
```



- Time Sliding Window:

```
String executionPlan = "" +  
    "define stream cseEventStream (height int); " +  
    "" +  
    "@info(name = 'query1') " +  
    "from cseEventStream #window.time(1 sec) " +  
    "select avg(height) as avgHeight " +  
    "insert into outputStream ;";
```

- Time Batch Window:

```
String executionPlan = "" +  
    "define stream cseEventStream (height int); " +  
    "" +  
    "@info(name = 'query1') " +  
    "from cseEventStream #window.timeBatch(1 sec) " +  
    "select avg(height) as AvgHeight " +  
    "insert into outputStream ;";
```



- Ordering – sequences and patterns (before, followed by conditions e.g. new location followed by small and a large purchase might suggest a fraud)

➤ Pattern Query

```
executionPlan = "" +  
    "define stream cseEventStream (height int); " +  
    "" +  
    "@info(name = 'query1') " +  
    "from every e1 = cseEventStream " +  
    "-> e2 = cseEventStream[e1.height == e2.height]" +  
    "-> e3 = cseEventStream[e2.height == e3.height]" +  
    "select e1.height as height1, e2.height as height2, e3.height as height3 " +  
    "insert into outputStream ;";
```

➤ Sequence Query

```
executionPlan = "" +  
    "define stream cseEventStream (height int); " +  
    "" +  
    "@info(name = 'query1') " +  
    "from every e1 = cseEventStream, "+  
    "e2 = cseEventStream[e1.height == e2.height], "+  
    "e3 = cseEventStream[e3.height == e2.height] " +  
    "select e1.height as height1, e2.height as height2, e3.height as height3 " +  
    "insert into outputStream ;";
```



Other Siddhi CEP Queries

- Split
- Join
- Partition
- **Advanced Queries:**
<https://docs.wso2.com/display/CEP300/Advanced+Queries>



Use Siddhi API

- Packages to be imported:

```
import org.wso2.siddhi.core.ExecutionPlanRuntime;
import org.wso2.siddhi.core.SiddhiManager;
import org.wso2.siddhi.core.event.Event;
import org.wso2.siddhi.core.query.output.callback.QueryCallback;
import org.wso2.siddhi.core.stream.input.InputHandler;
import org.wso2.siddhi.core.util.EventPrinter;
```

- Maven to get Siddhi dependencies:

```
<dependency>
  <groupId>org.wso2.siddhi</groupId>
  <artifactId>siddhi-query-api</artifactId>
  <version>3.0.5</version>
</dependency>
<dependency>
  <groupId>org.wso2.siddhi</groupId>
  <artifactId>siddhi-query-compiler</artifactId>
  <version>3.0.5</version>
</dependency>
<dependency>
  <groupId>org.wso2.siddhi</groupId>
  <artifactId>siddhi-core</artifactId>
  <version>3.0.5</version>
</dependency>
```



Use Siddhi API

■ Starting Siddhi Manager:

```
// Creating Siddhi Manager
SiddhiManager siddhiManager = new SiddhiManager();

String executionPlan = "" +
    "define stream cseEventStream (symbol string, price float, volume long); " +
    "" +
    "@info(name = 'query1') " +
    "from cseEventStream[volume < 150] " +
    "select symbol,price " +
    "insert into outputStream ;";

//Generating runtime
ExecutionPlanRuntime executionPlanRuntime = siddhiManager.createExecutionPlanRuntime(executionPlan);

//Adding callback to retrieve output events from query
executionPlanRuntime.addCallback("query1", new QueryCallback() {
    @Override
    public void receive(long timeStamp, Event[] inEvents, Event[] removeEvents) {
        EventPrinter.print(timeStamp, inEvents, removeEvents);
    }
});

//Retrieving InputHandler to push events into Siddhi
InputHandler inputHandler = executionPlanRuntime.getInputHandler("cseEventStream");

//Starting event processing
executionPlanRuntime.start();
```




Input to Siddhi

```
//Starting event processing
executionPlanRuntime.start();

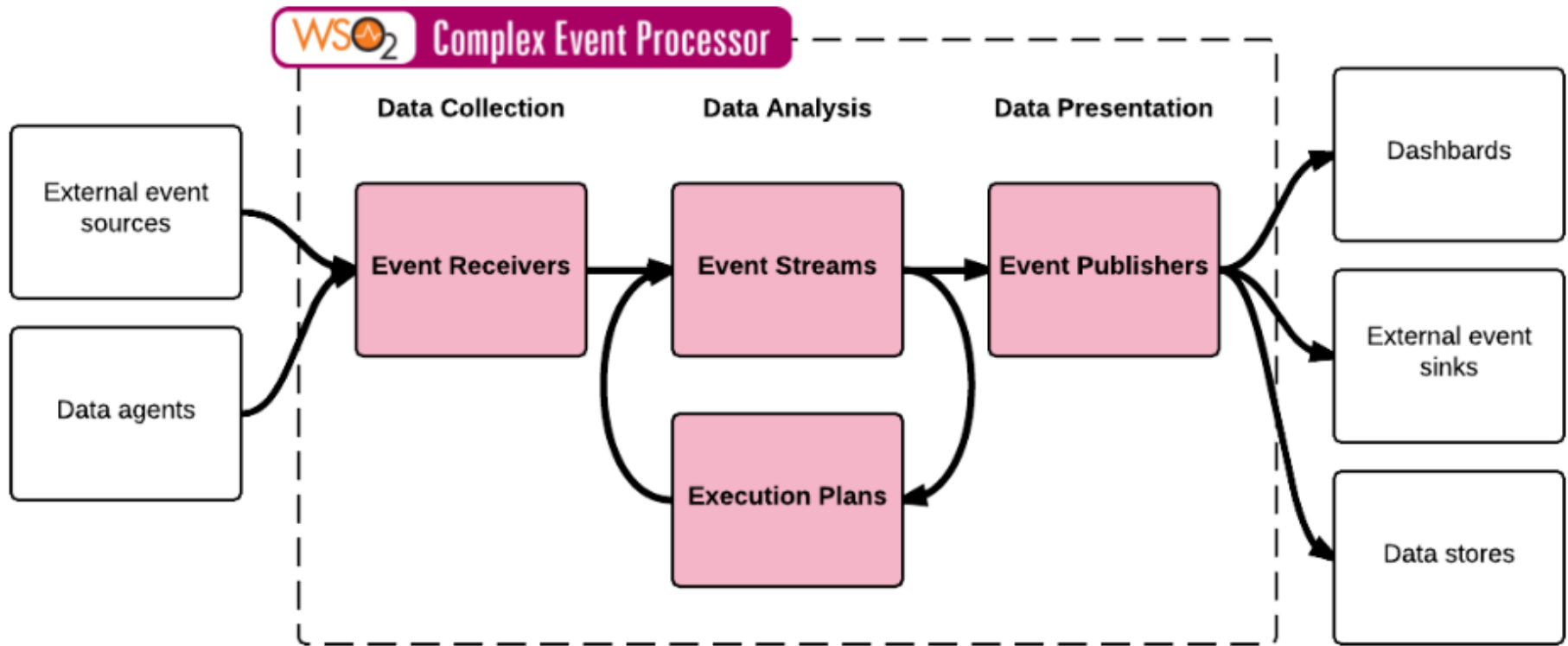
//Sending events to Siddhi
inputHandler.send(new Object[]{"IBM", 700f, 1001});
inputHandler.send(new Object[]{"WSO2", 60.5f, 2001});
inputHandler.send(new Object[]{"GOOG", 50f, 301});
inputHandler.send(new Object[]{"IBM", 76.6f, 4001});
inputHandler.send(new Object[]{"WSO2", 45.6f, 501});
Thread.sleep(500);

//Shutting down the runtime
executionPlanRuntime.shutdown();

//Shutting down Siddhi
siddhiManager.shutdown();
```



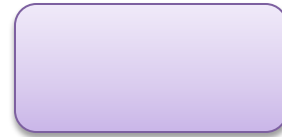
Siddhi Architecture and Event Flow





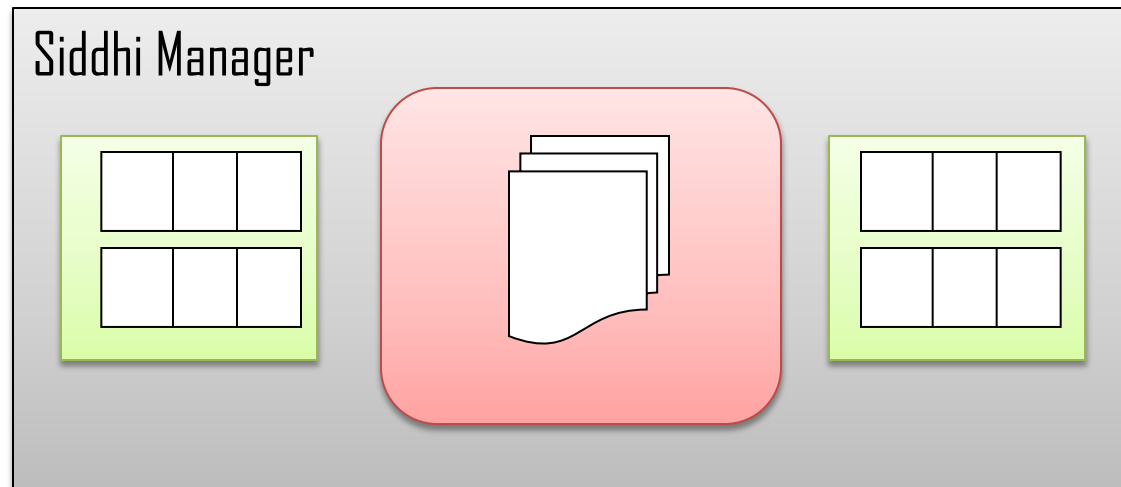
User

Input Adapters



Compiler

Output Adapters



User Input a Query



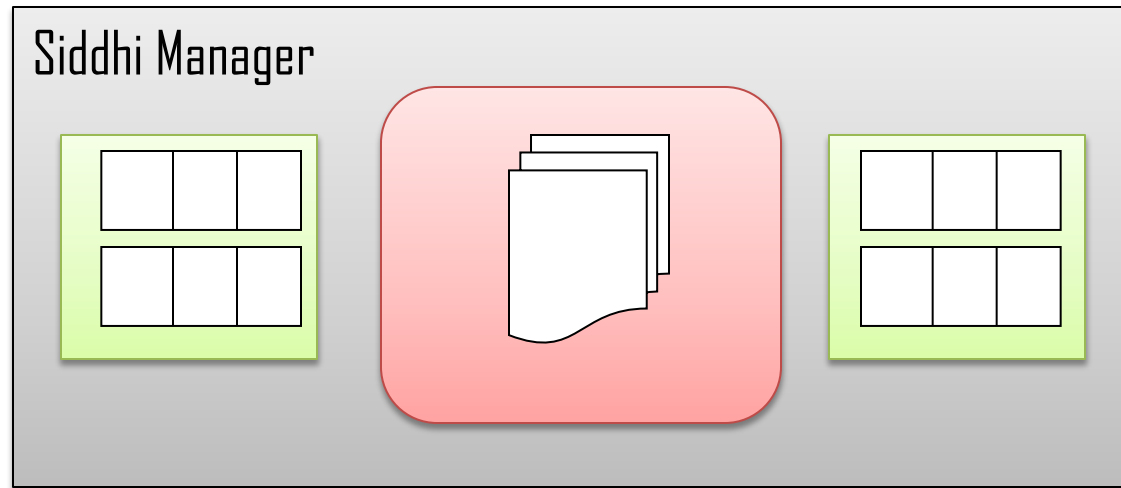
User

Input Adapters



Compiler

Output Adapters



Compiling the query



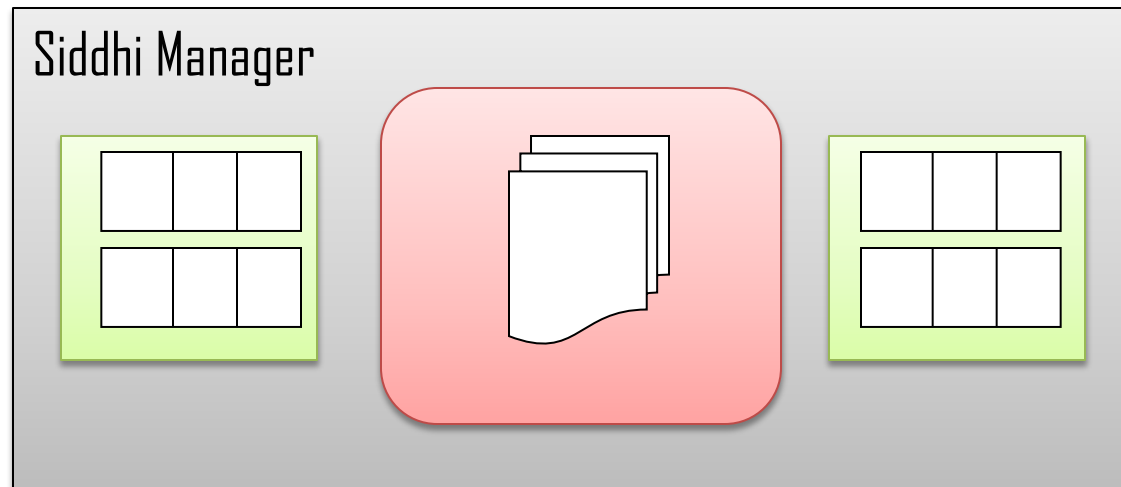
User

Input Adapters



Compiler

Output Adapters



Query get compiled into an object model



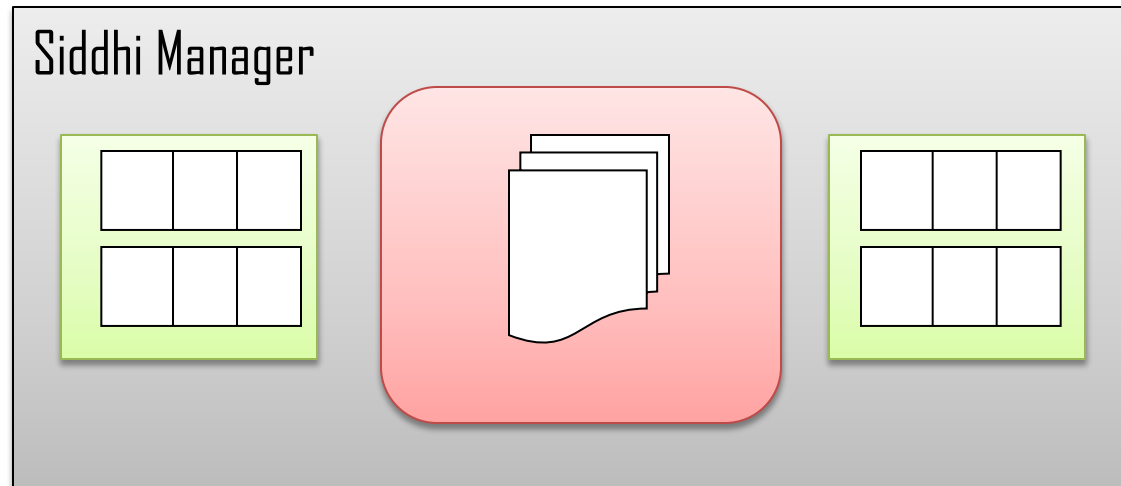
User

Input Adapters



Compiler

Output Adapters

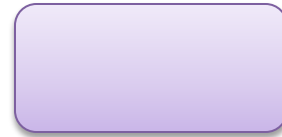


Query Object Model is parsed to the Siddhi Manager



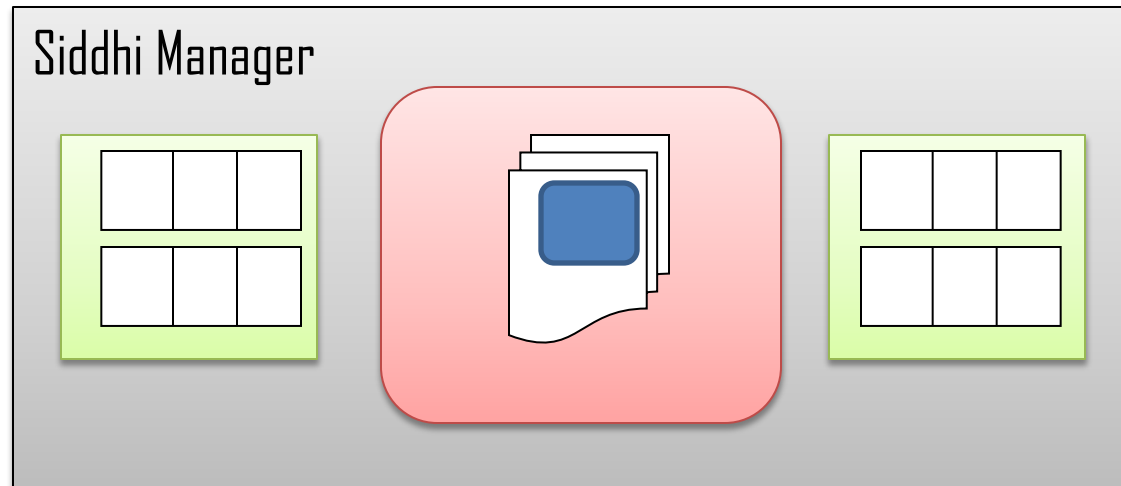
User

Input Adapters



Compiler

Output Adapters

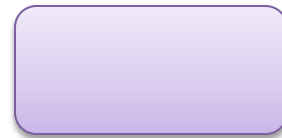


Event arrives to the Input Adapter



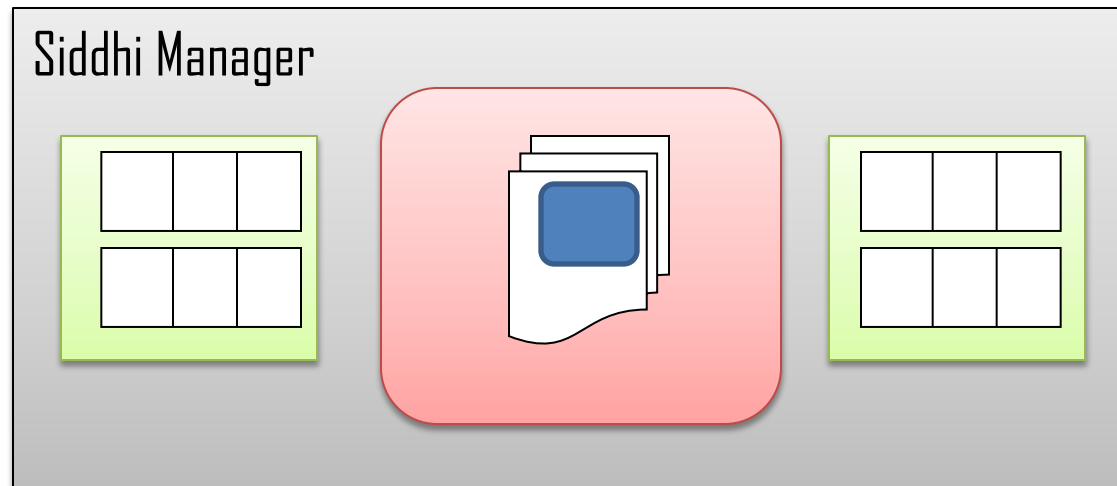
User

Input Adapters



Compiler

Output Adapters

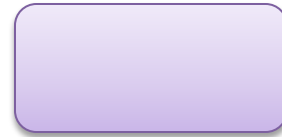


Convert Event to a Tuple and place it to the input event queue



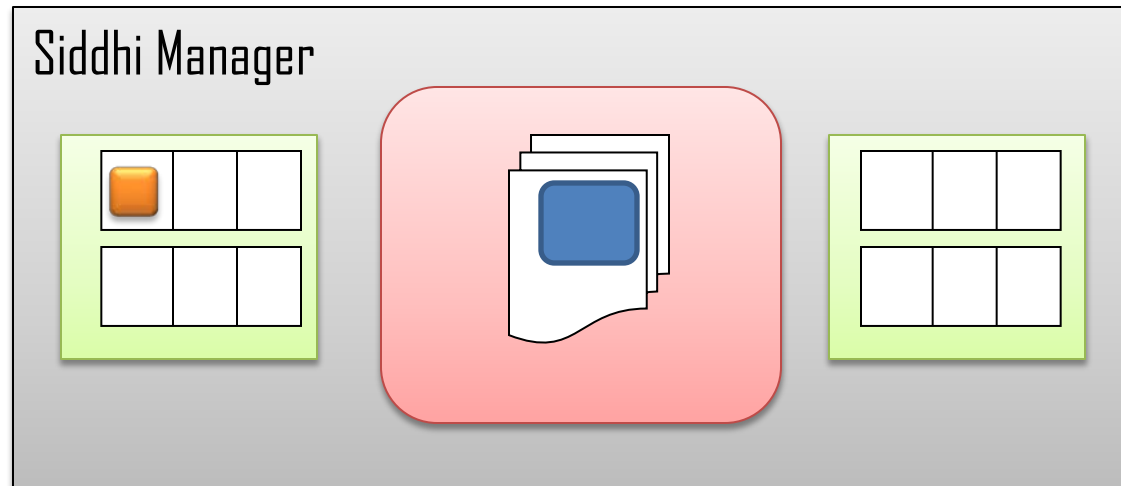
User

Input Adapters



Compiler

Output Adapters

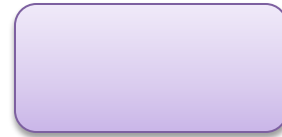


Processor takes the tuple from the queue



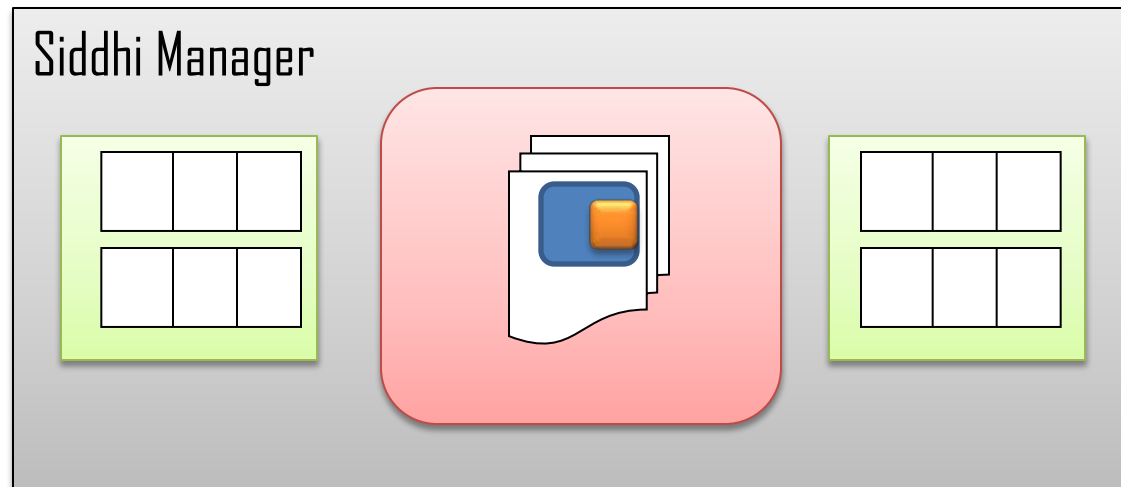
User

Input Adapters



Compiler

Output Adapters

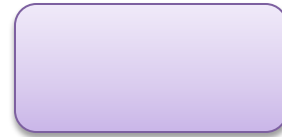


Executing the queries... Other Events arrives at the same time



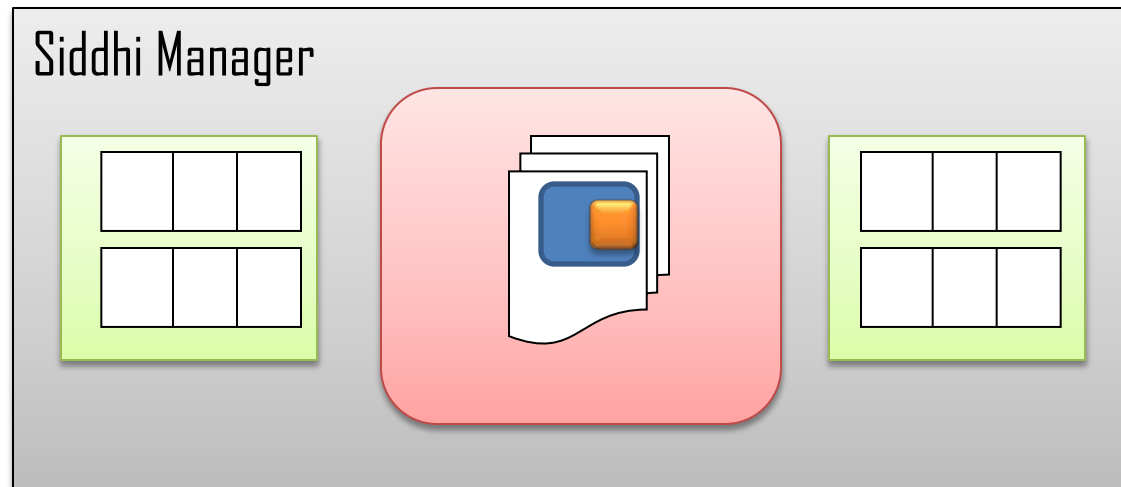
User

Input Adapters



Compiler

Output Adapters

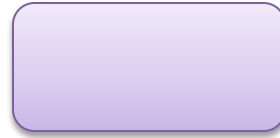


Non matching event thrown away



User

Input Adapters

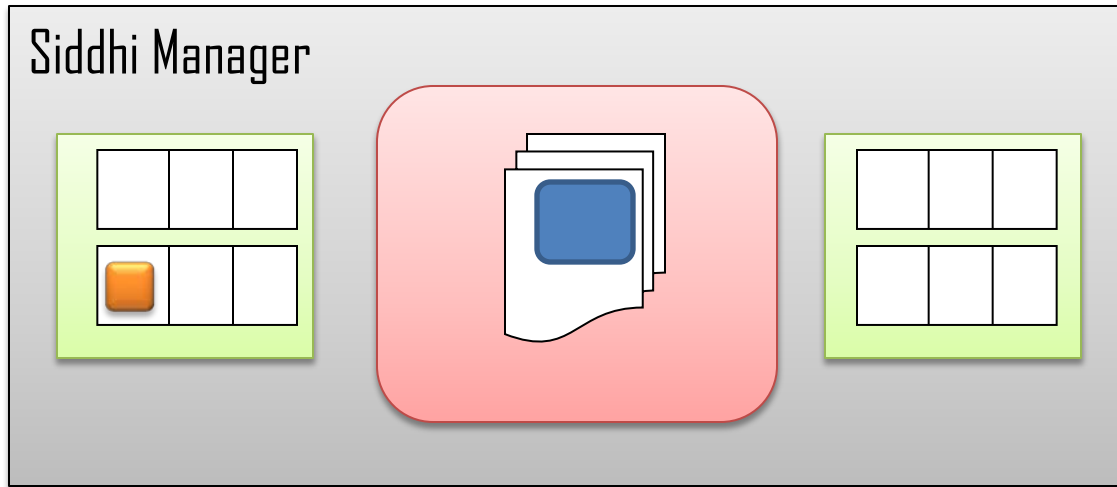


Compiler

Output Adapters



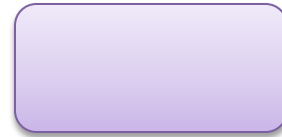
Siddhi Manager





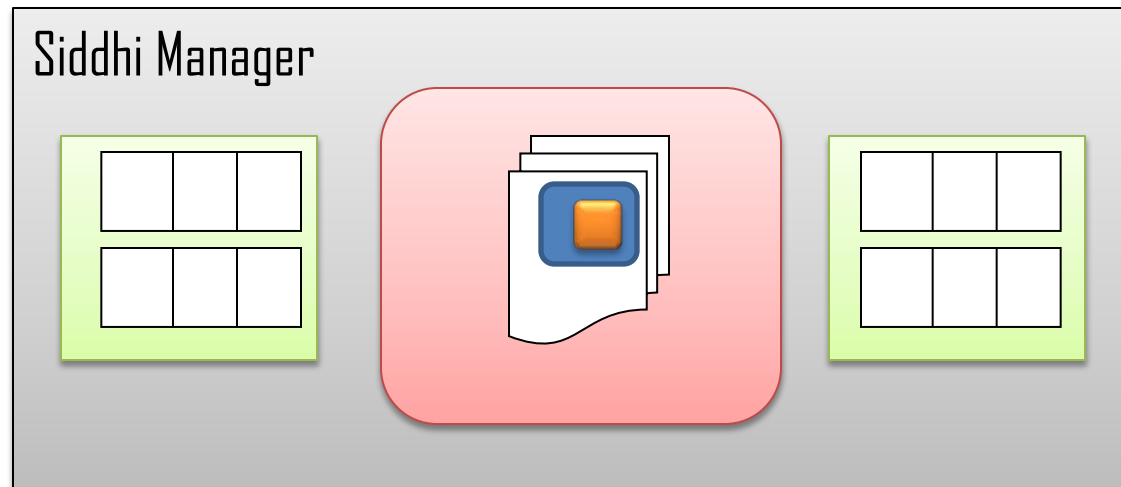
User

Input Adapters



Compiler

Output Adapters

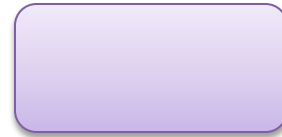


Matching Event creates the output Event



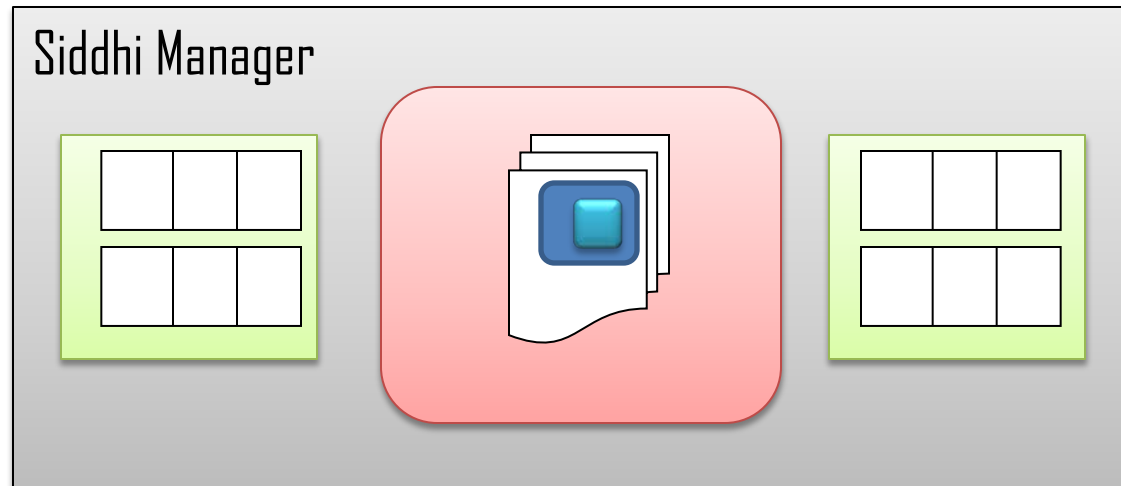
User

Input Adapters



Compiler

Output Adapters

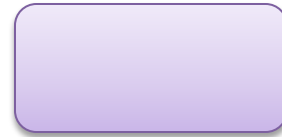


Pushing generated Events to the output queue



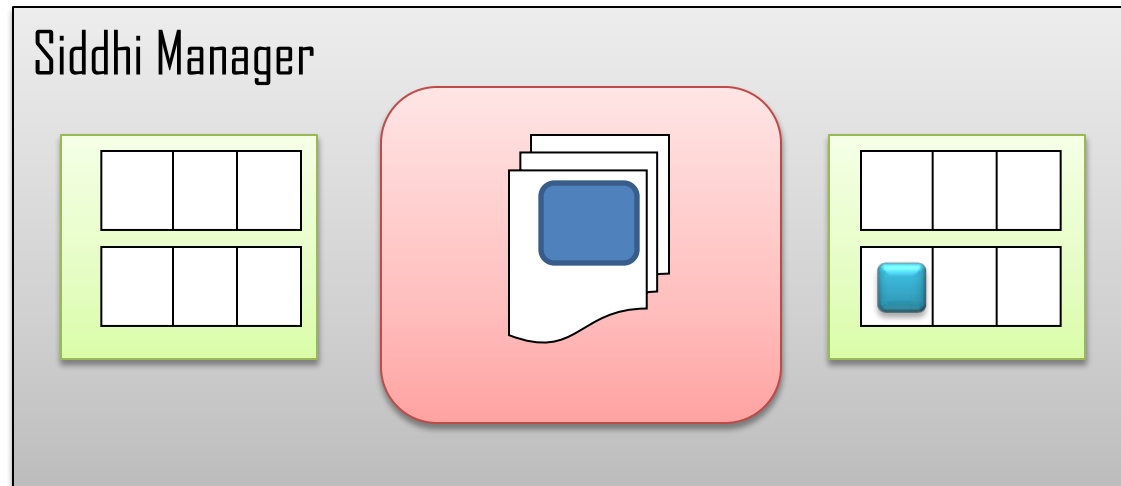
User

Input Adapters



Compiler

Output Adapters

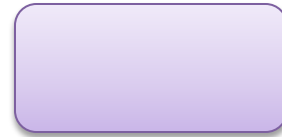


User get notified through output Adapter



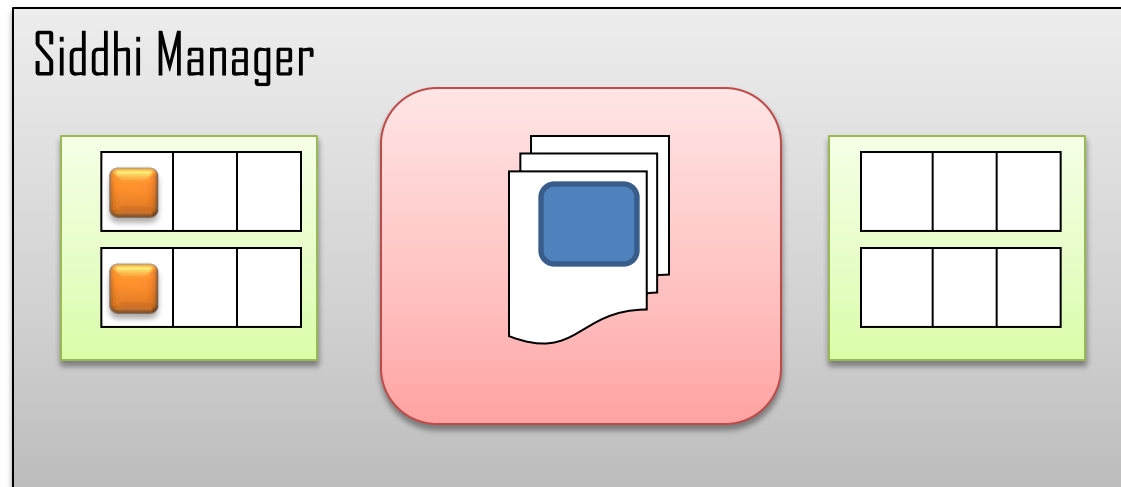
User

Input Adapters



Compiler

Output Adapters



Same procedure happens again and again...



All set for Hands On

Let the data flow