DREAM:Lab

Siddhi: A Complex Event Processing Engine

Presenter:
Rajrup Ghosh
M.Tech. Computational Science,
CDS, IISc Bangalore

15-JAN-16

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

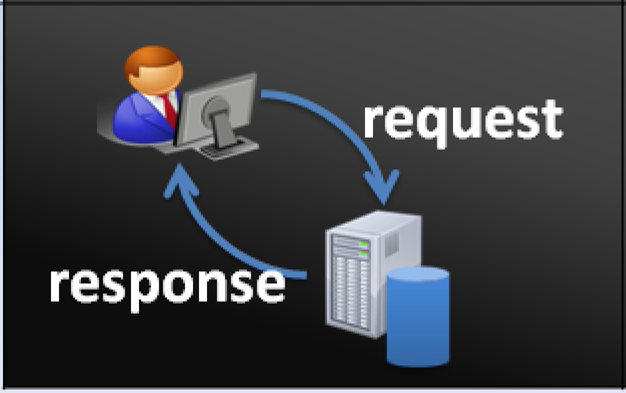
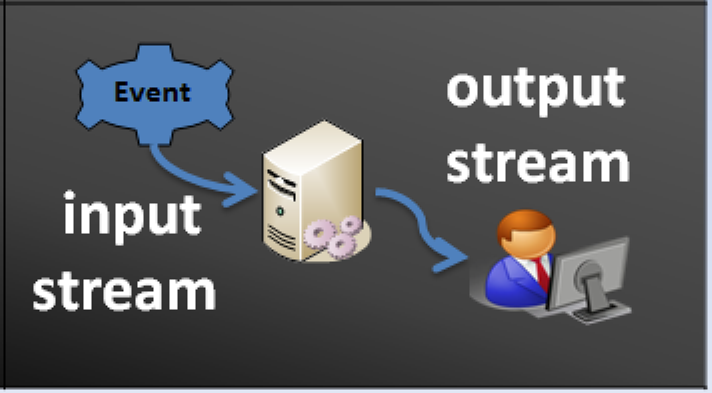
https://docs.wso2.com/display/CEP400/Understanding+Event+Streams

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.

https://en.wikipedia.org/wiki/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.

https://en.wikipedia.org/wiki/Complex_event_processing

Complex Event Processing Use Cases

Utilities

Meter alarm filtering

Power restoration confirmation

Mobile work unit tracking

Telecommunications & Services

Fraud Detection, digital marketing

- *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"*

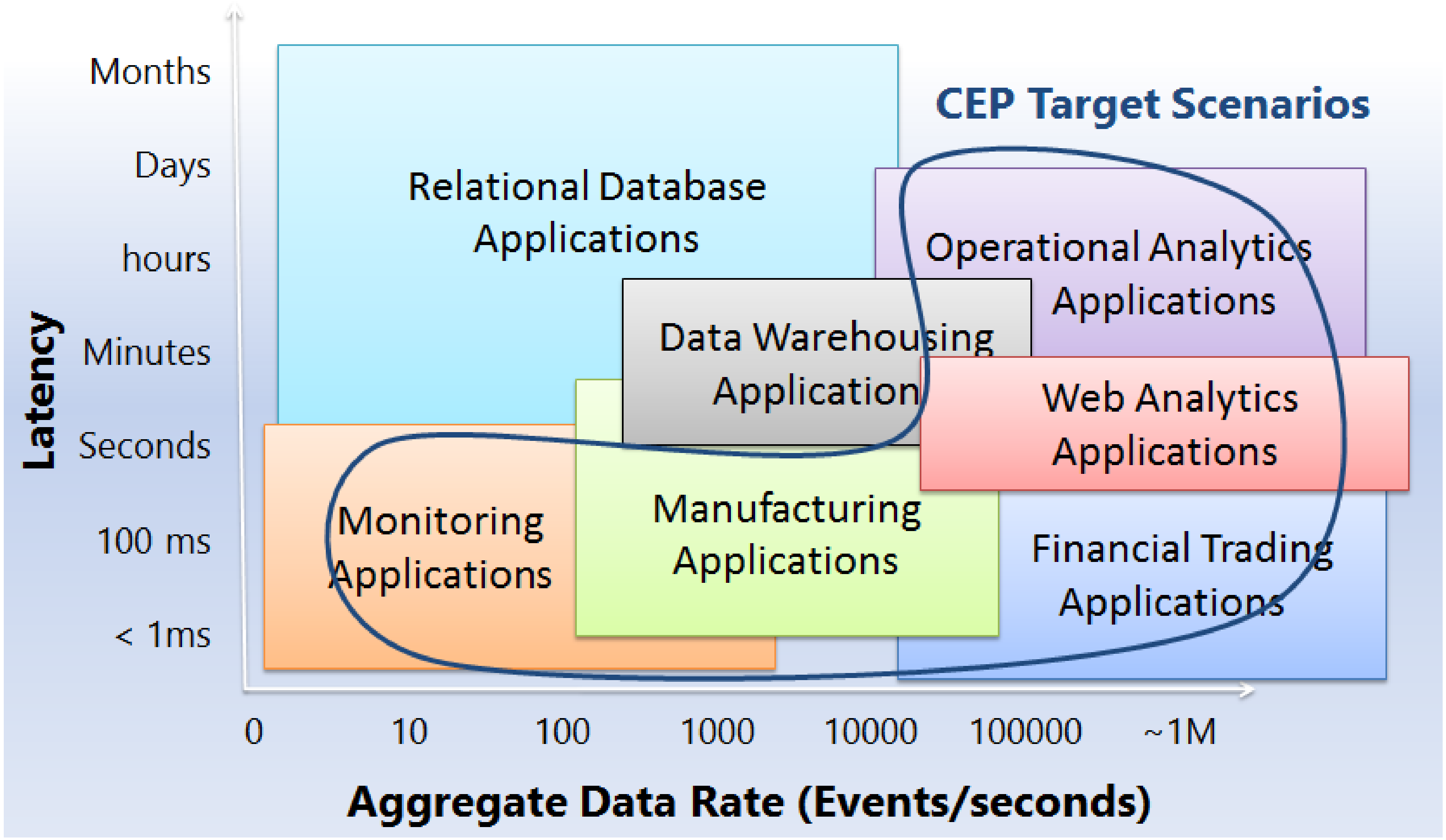
Homeland Security

Threat detection, sensor data correlation, emergency response

Financial Services

Algorithmic trading

Scenarios of Complex Event Processing


Months
Days
hours
Minutes
Seconds
100 ms
< 1ms
Latency
CEP Target Scenarios
Relational Database Applications
Operational Analytics Applications
Data Warehousing Application
Web Analytics Applications
Monitoring Applications
Manufacturing Applications
Financial Trading Applications
0
10
100
1000
10000
100000
~1M
Aggregate Data Rate (Events/seconds)

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.

https://aws.amazon.com/kinesis/streams/

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.

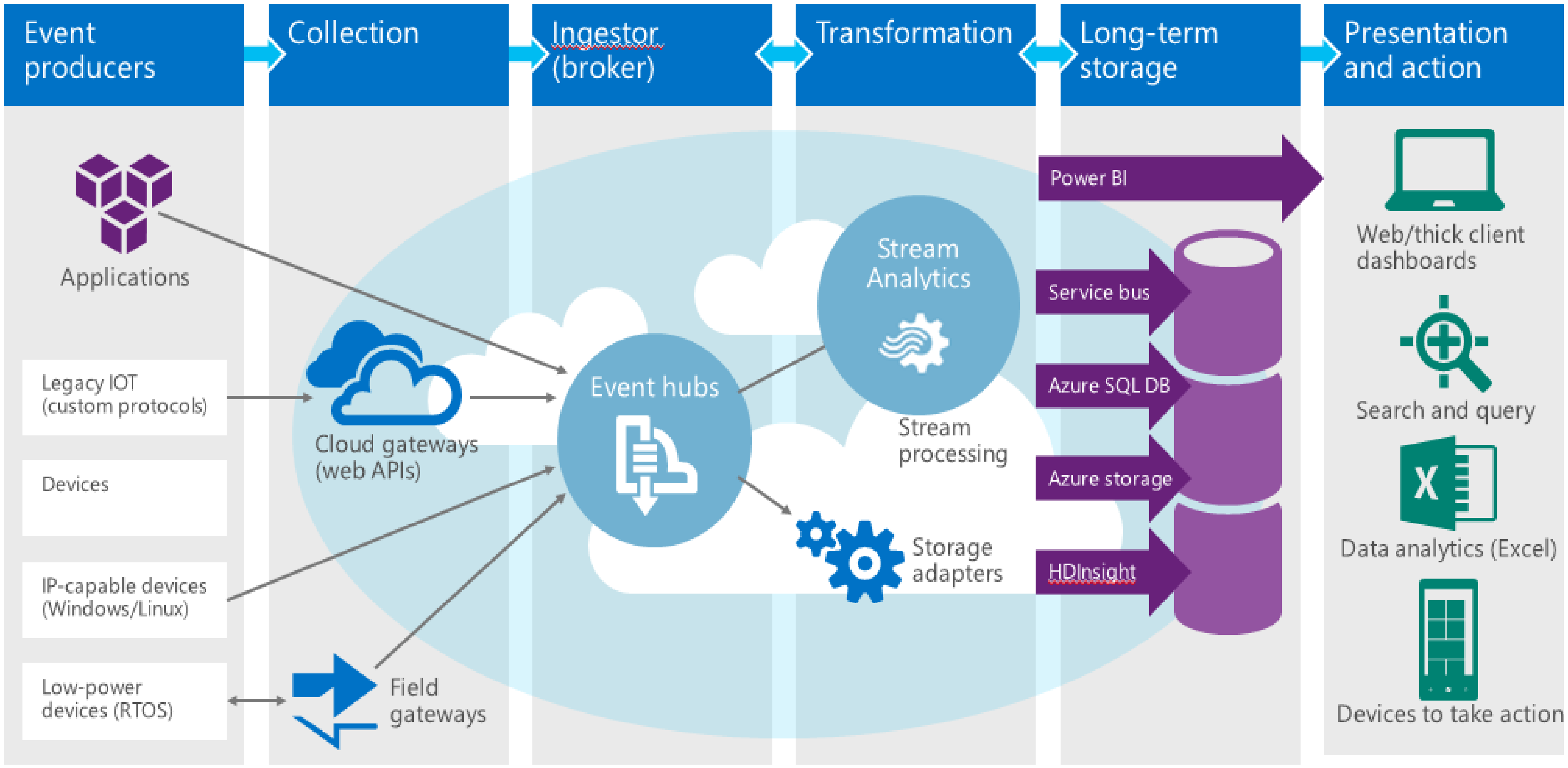
https://aws.amazon.com/kinesis/streams/

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

https://azure.microsoft.com/en-in/services/stream-analytics/

End to End processing on Microsoft Azure



Azure Stream Analytics, James Serra, Big Data Evangelist, Microsoft

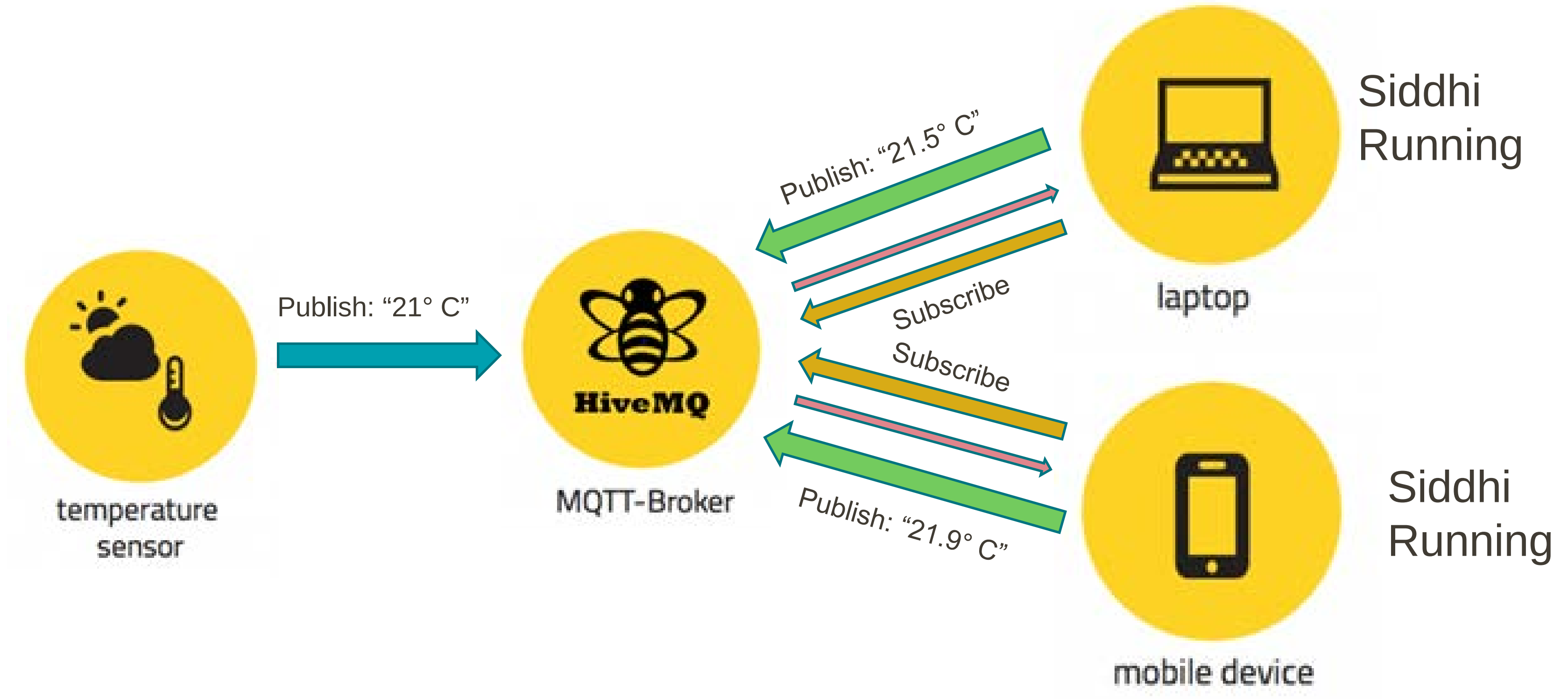
Problems with current CEP solutions

- 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



⬢ Subscribe to topic: “temperature”

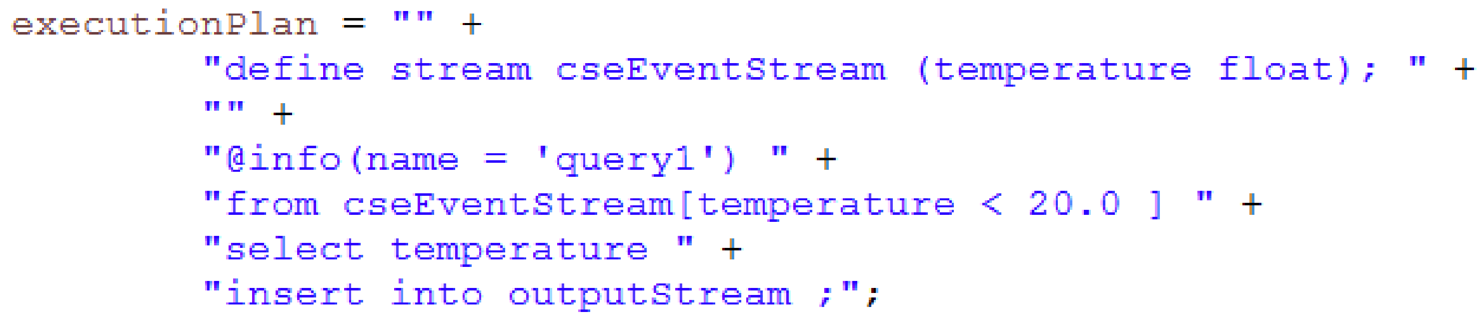
⬢ Publish to topic: “temperature”

⬢ Publish to topic: “Avg. temperature”

http://www.hivemq.com/blog/how-to-get-started-with-mqtt

Siddhi CEP Queries

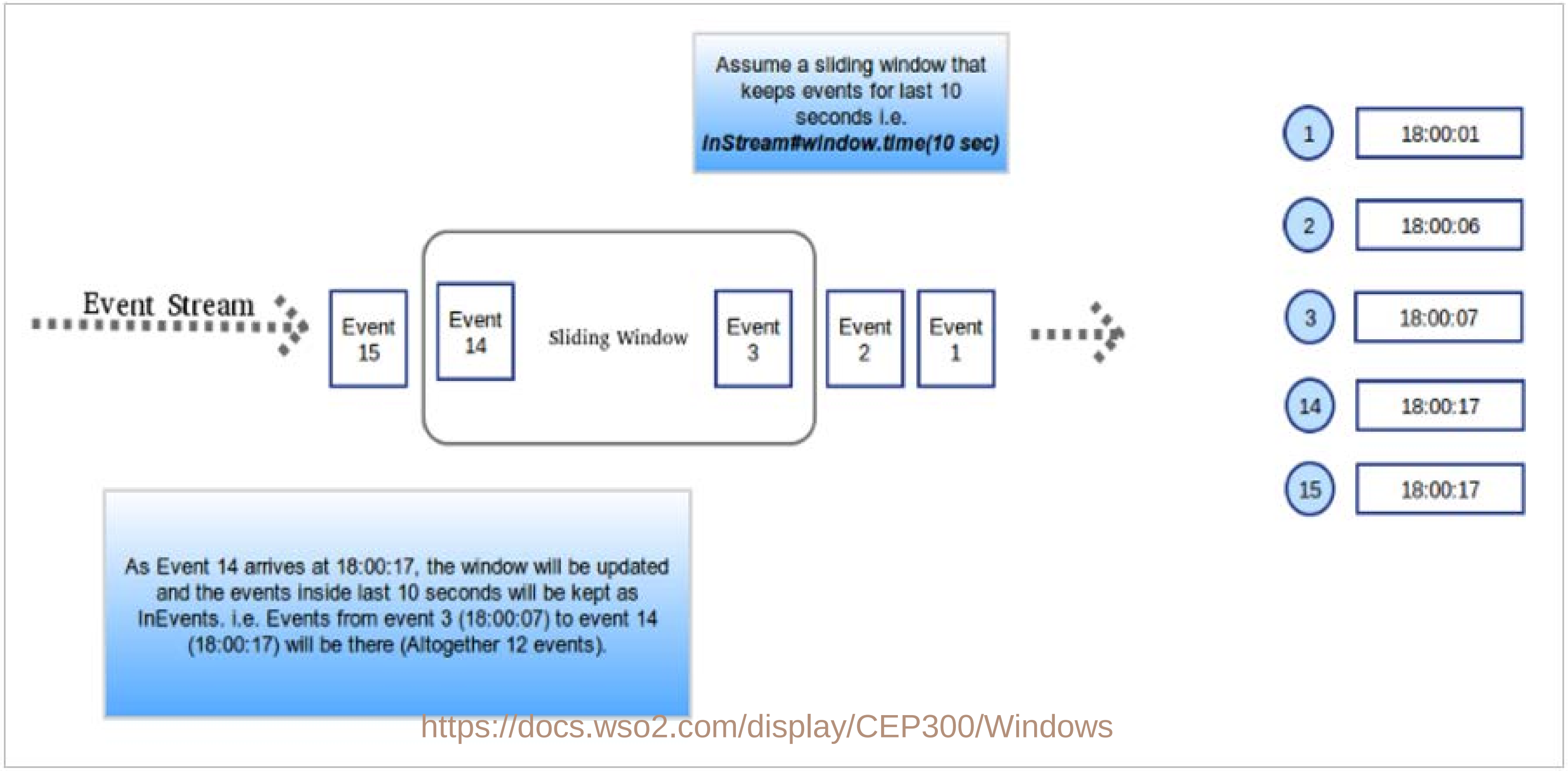
- Selection or filtering and projection (like select in SQL)
- Filter query creates an output stream and inserts any events from the input stream that satisfies the conditions defined.
- Filters support following types of conditions
  - » >, <, ==, >=, <=, !=
  - » contains, instanceof
  - » and, or,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



```
executionPlan = "" +
        "define stream cseEventStream (temperature float); " +
        "" +
        "@info(name = 'query1') " +
        "from cseEventStream[temperature < 20.0 ] " +
        "select temperature " +
        "insert into outputStream ;";
```

https://docs.wso2.com/display/CEP300/Filters

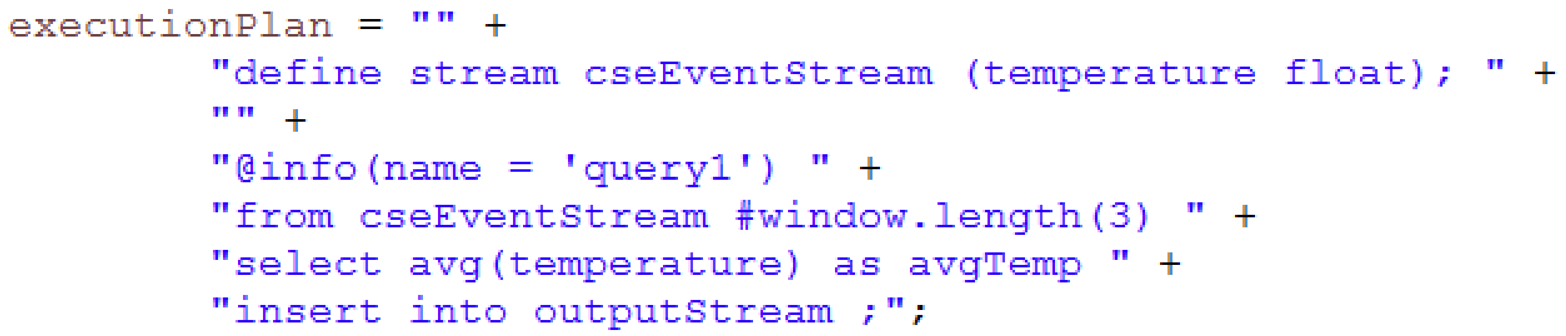
- Windows – events are processed within a window (e.g. for aggregation).
  - » Time window
  - » Length window
  - » Time batch window
  - » Length batch window


Assume a sliding window that keeps events for last 10 seconds i.e. *InStream#window.time(10 sec)*
Event Stream
Event 15
Event 14
Sliding Window
Event 3
Event 2
Event 1
1 18:00:01
2 18:00:06
3 18:00:07
14 18:00:17
15 18:00:17
As Event 14 arrives at 18:00:17, the window will be updated and the events inside last 10 seconds will be kept as InEvents. i.e. Events from event 3 (18:00:07) to event 14 (18:00:17) will be there (Altogether 12 events).

https://docs.wso2.com/display/CEP300/Windows

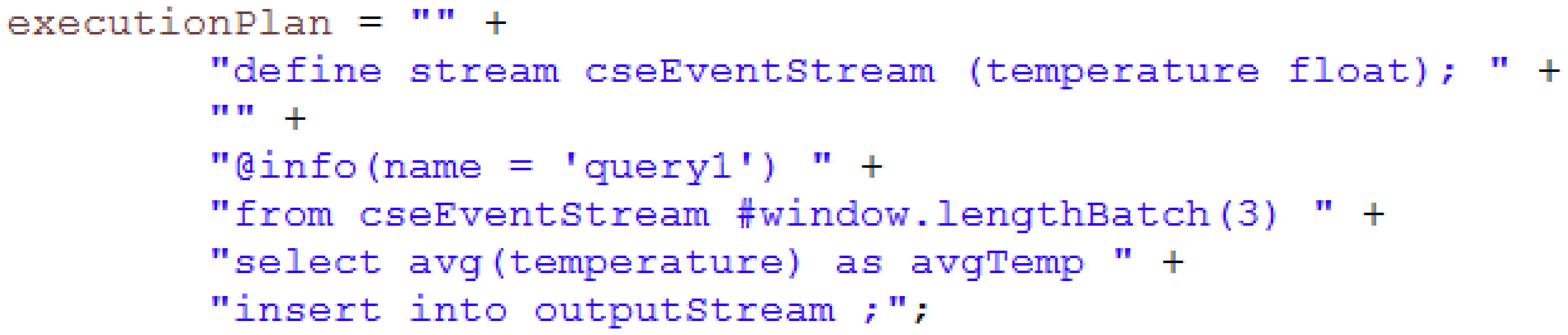
Window Query Examples

- Window Query:
  - Length Sliding Window



```
executionPlan = "" +
           "define stream cseEventStream (temperature float); " +
           "" +
           "@info(name = 'query1') " +
           "from cseEventStream #window.length(3) " +
           "select avg(temperature) as avgTemp " +
           "insert into outputStream ;";
```

  - Length Batch Window



```
executionPlan = "" +
           "define stream cseEventStream (temperature float); " +
           "" +
           "@info(name = 'query1') " +
           "from cseEventStream #window.lengthBatch(3) " +
           "select avg(temperature) as avgTemp " +
           "insert into outputStream ;";
```

- Time Sliding Window:

```
executionPlan = "" +
        "define stream cseEventStream (temperature float); " +
        "" +
        "@info(name = 'query1') " +
        "from cseEventStream #window.time(10 sec) " +
        "select avg(temperature) as avgTemp " +
        "insert into outputStream ;";
```

- Time Batch Window:

```
executionPlan = "" +
        "define stream cseEventStream (temperature float); " +
        "" +
        "@info(name = 'query1') " +
        "from cseEventStream #window.timeBatch(10 sec) " +
        "select avg(temperature) as avgTemp " +
        "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 (temperature float); " +
        "" +
        "@info(name = 'query1') " +
        "from every e1 =  cseEventStream " +
        "-> e2 = cseEventStream[e1.temperature == e2.temperature]" +
        "-> e3 = cseEventStream[e2.temperature == e3.temperature] "+
        "select e1.temperature as temp1, e2.temperature as temp2, e3.temperature as temp3  " +
        "insert into outputStream ;";
```

  - Sequence Query

```
executionPlan = "" +
        "define stream cseEventStream (temperature float); " +
        "" +
        "@info(name = 'query1') " +
        "from every e1 =  cseEventStream, "+
        "e2 = cseEventStream[e1.temperature == e2.temperature], "+
        "e3 =  cseEventStream[e3.temperature == e2.temperature] " +
        "select e1.temperature as temp1, e2.temperature as temp2, e3.temperature as temp3 " +
        "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;
```

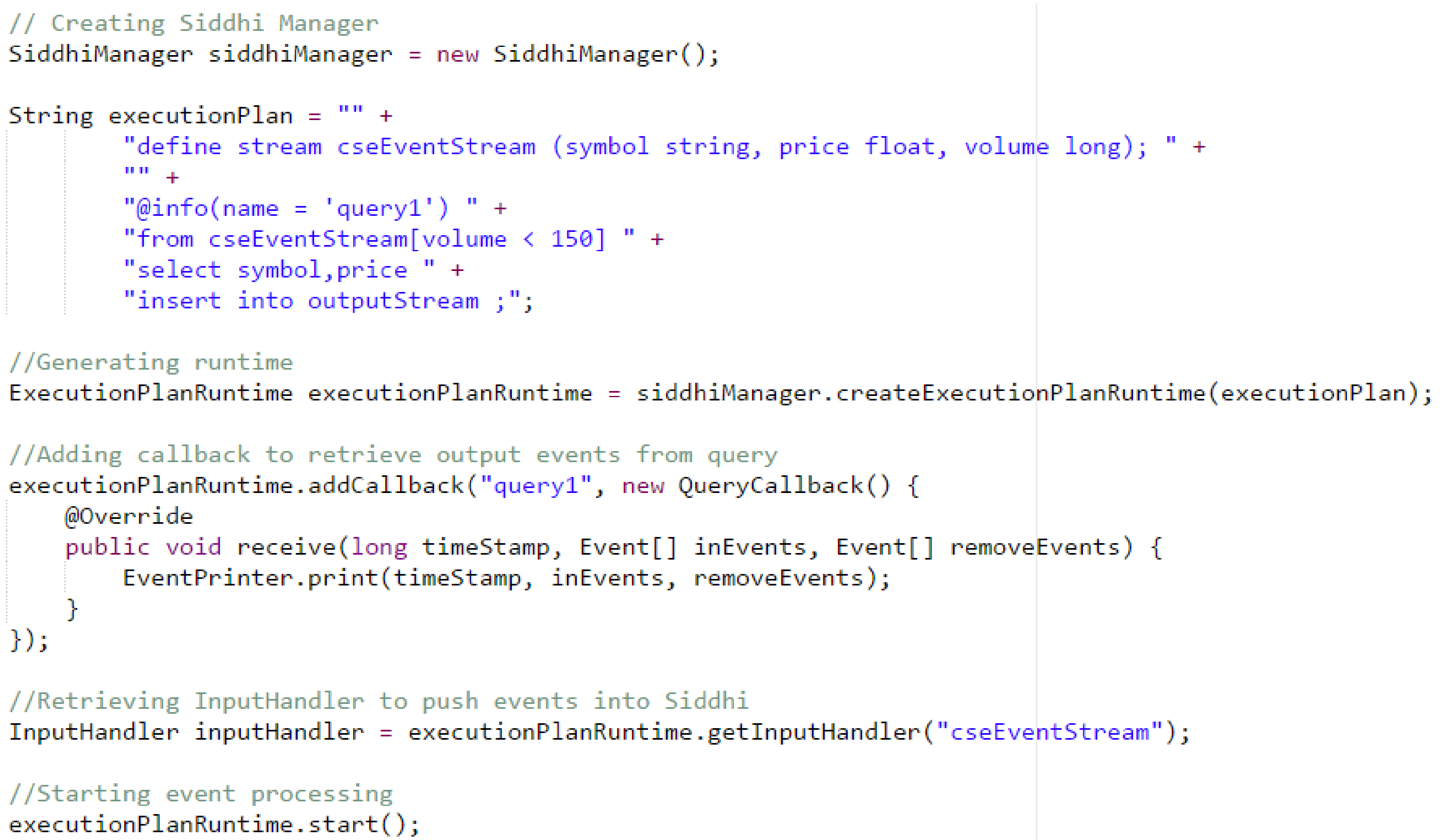
- Maven to get Siddhi dependencies:

```
<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>
```

https://docs.wso2.com/display/CEP400/Using+Siddhi+as+a+Library

Use Siddhi API

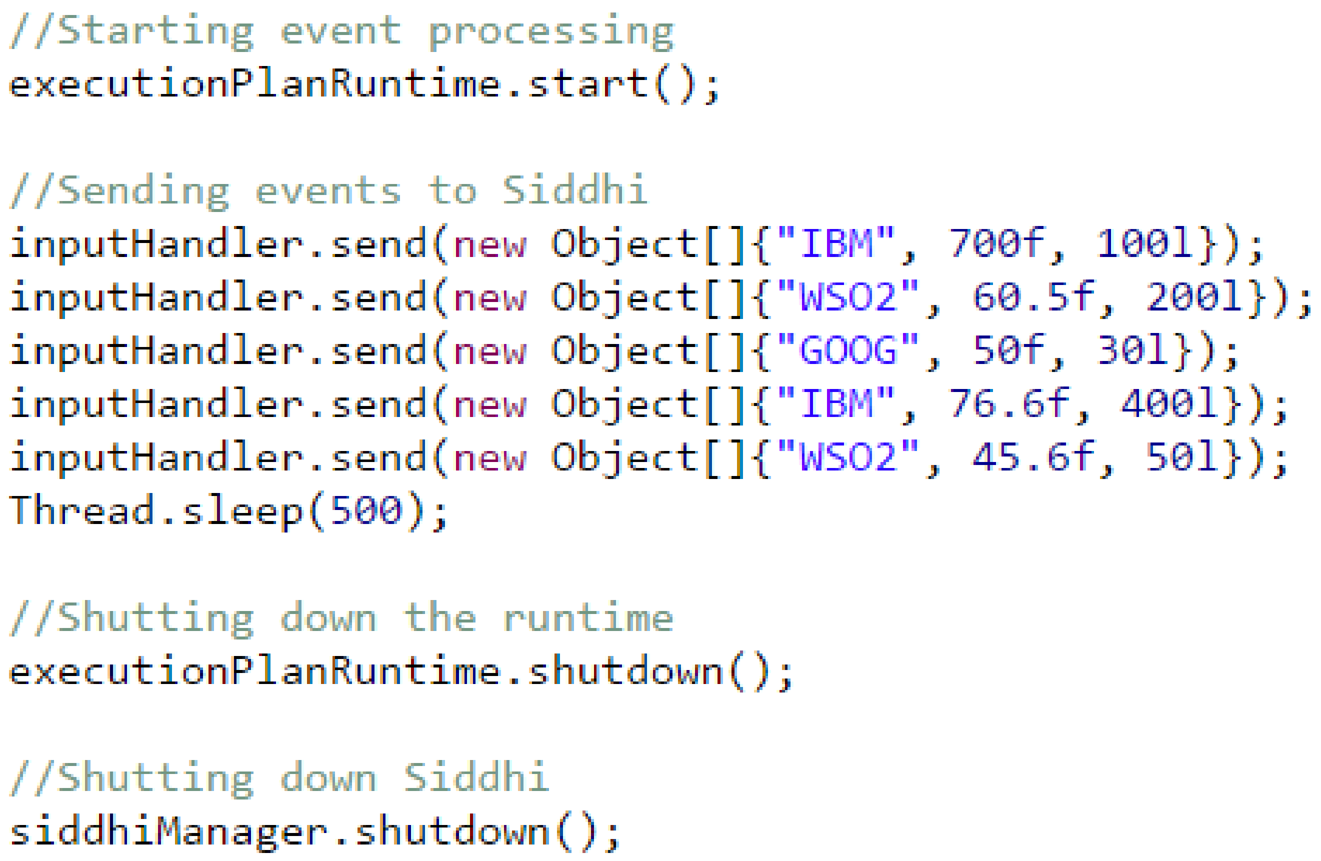
- Starting Siddhi Manager:



```
// 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();
```

https://github.com/wso2/siddhi/blob/master/modules/siddhi-samples/quick-start-samples/src/main/java/org/wso2/siddhi/sample/SimpleFilterSample.java

Input to Siddhi



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

//Sending events to Siddhi
inputHandler.send(new Object[]{"IBM", 700f, 100l});
inputHandler.send(new Object[]{"WSO2", 60.5f, 200l});
inputHandler.send(new Object[]{"GOOG", 50f, 30l});
inputHandler.send(new Object[]{"IBM", 76.6f, 400l});
inputHandler.send(new Object[]{"WSO2", 45.6f, 50l});
Thread.sleep(500);

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

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

https://github.com/wso2/siddhi/blob/master/modules/siddhi-samples/quick-start-samples/src/main/java/org/wso2/siddhi/sample/SimpleFilterSample.java

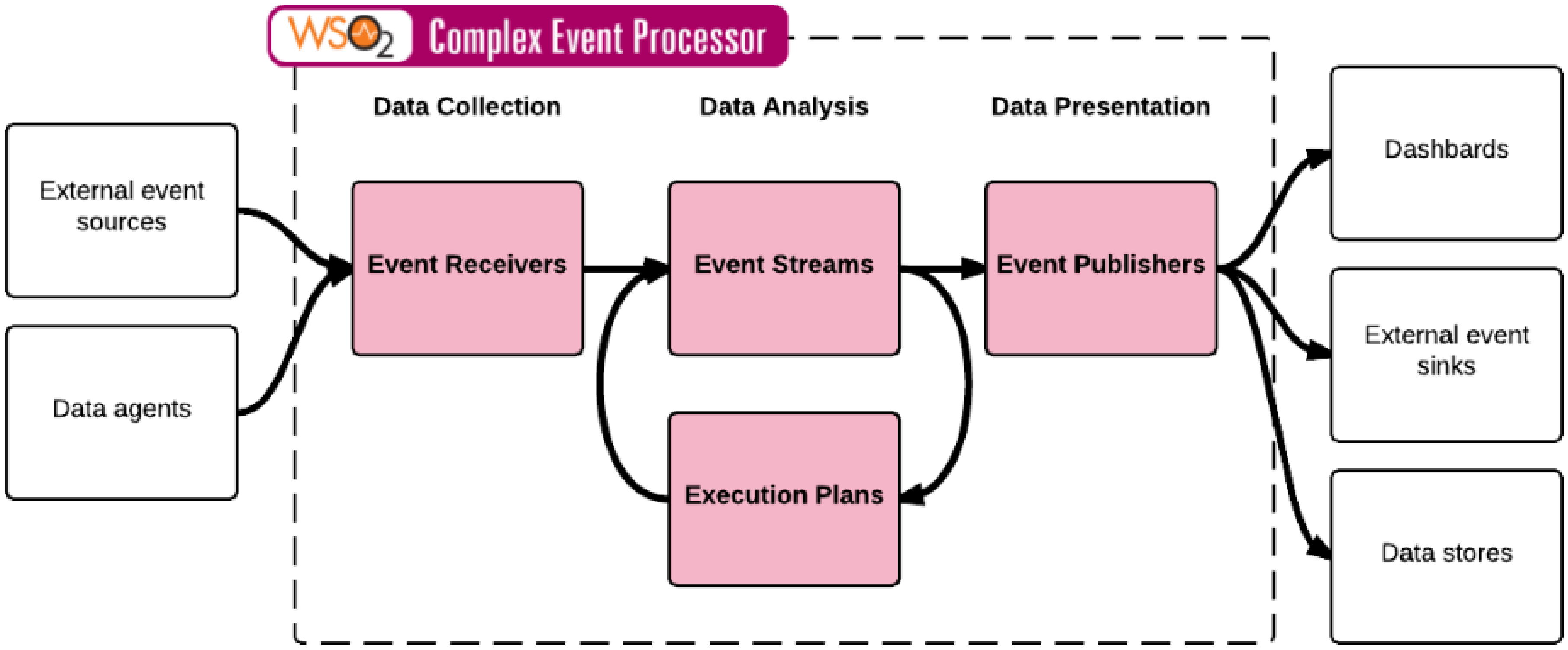
Some More on Siddhi

- Connected Queries:

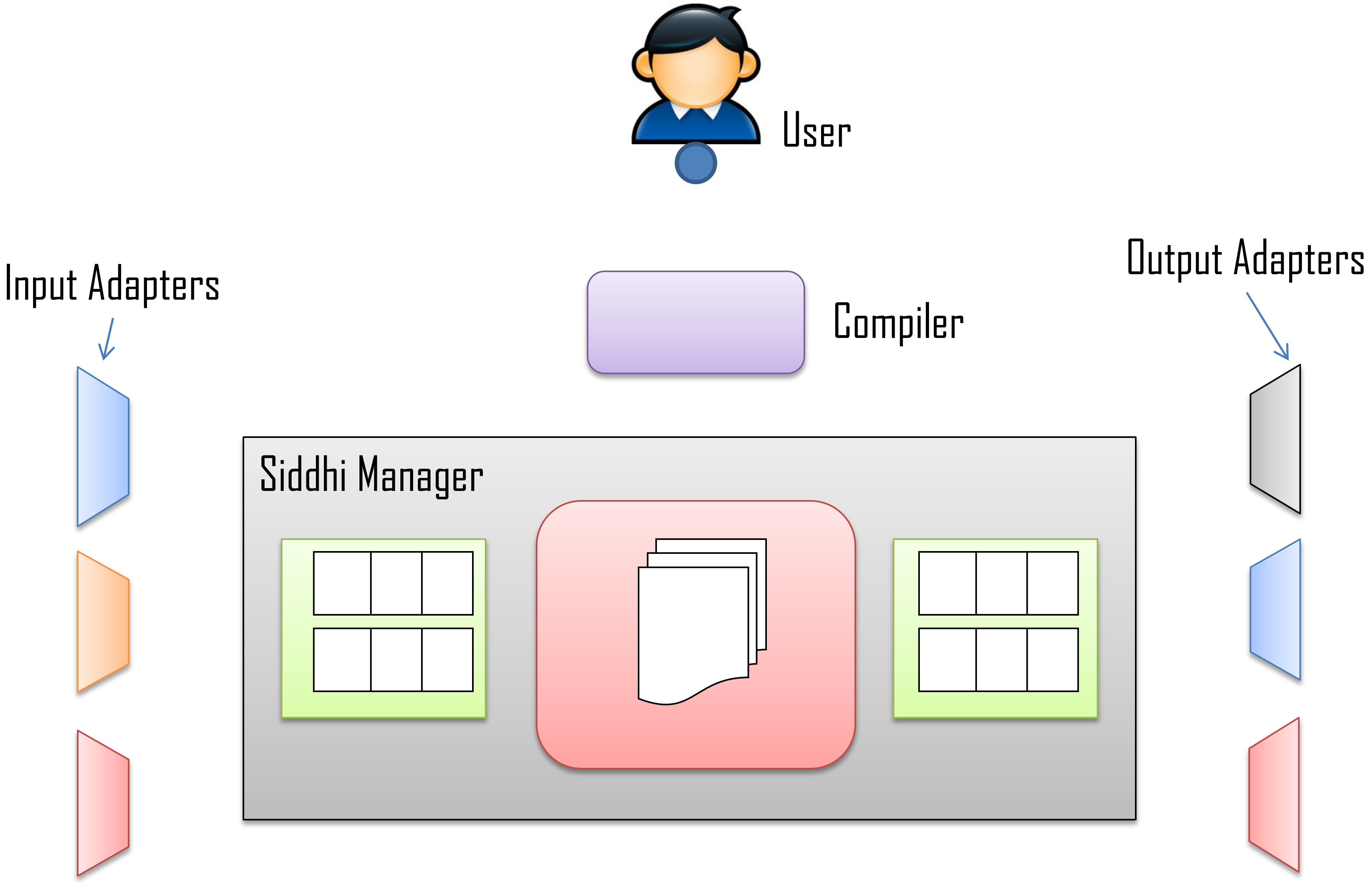
```
executionPlan = "" +
        "define stream InputEventStream1 (temperature double); " +
        "define stream InputEventStream2 (temperature double); " +
        "" +
        "@info(name = 'query1') " +
        "from InputEventStream1 #window.length(3) " +
        "select avg(temperature) as avgTemp " +
        "insert into AvgTempStream ;" +
        "" +
        "@info(name = 'query2') " +
        "from e = InputEventStream2 " + ", a = AvgTempStream [ (a.avgTemp - e.temperature) < 5.0  ]  " +
        "select a.avgTemp as avgTemperature, e.temperature as Temperature "+
        "insert into outputStream ;";
```

  - This requires appropriate manipulation of output callbacks.
  - Requires 2 input handlers to send streams.

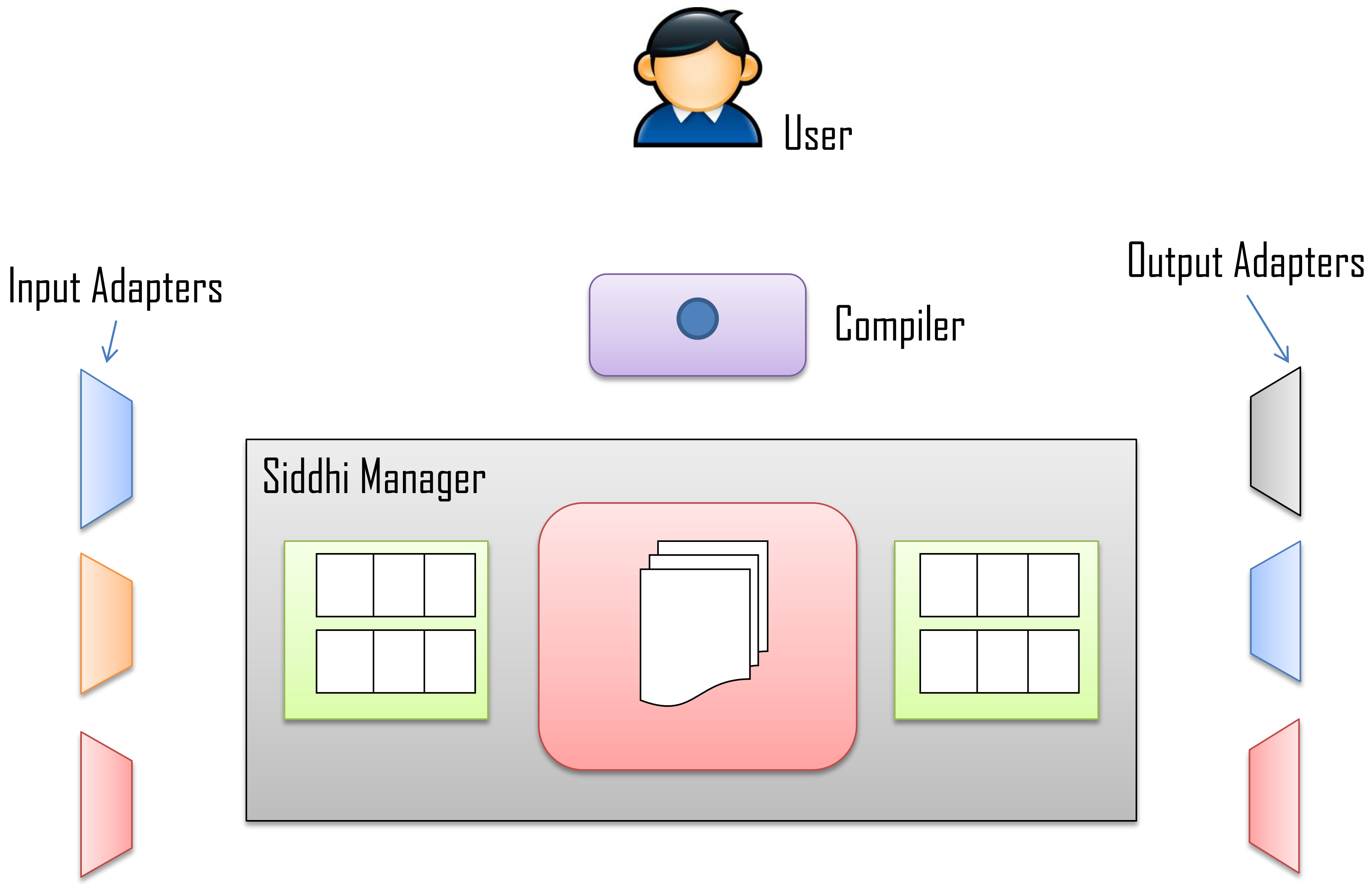
Siddhi Architecture and Event Flow



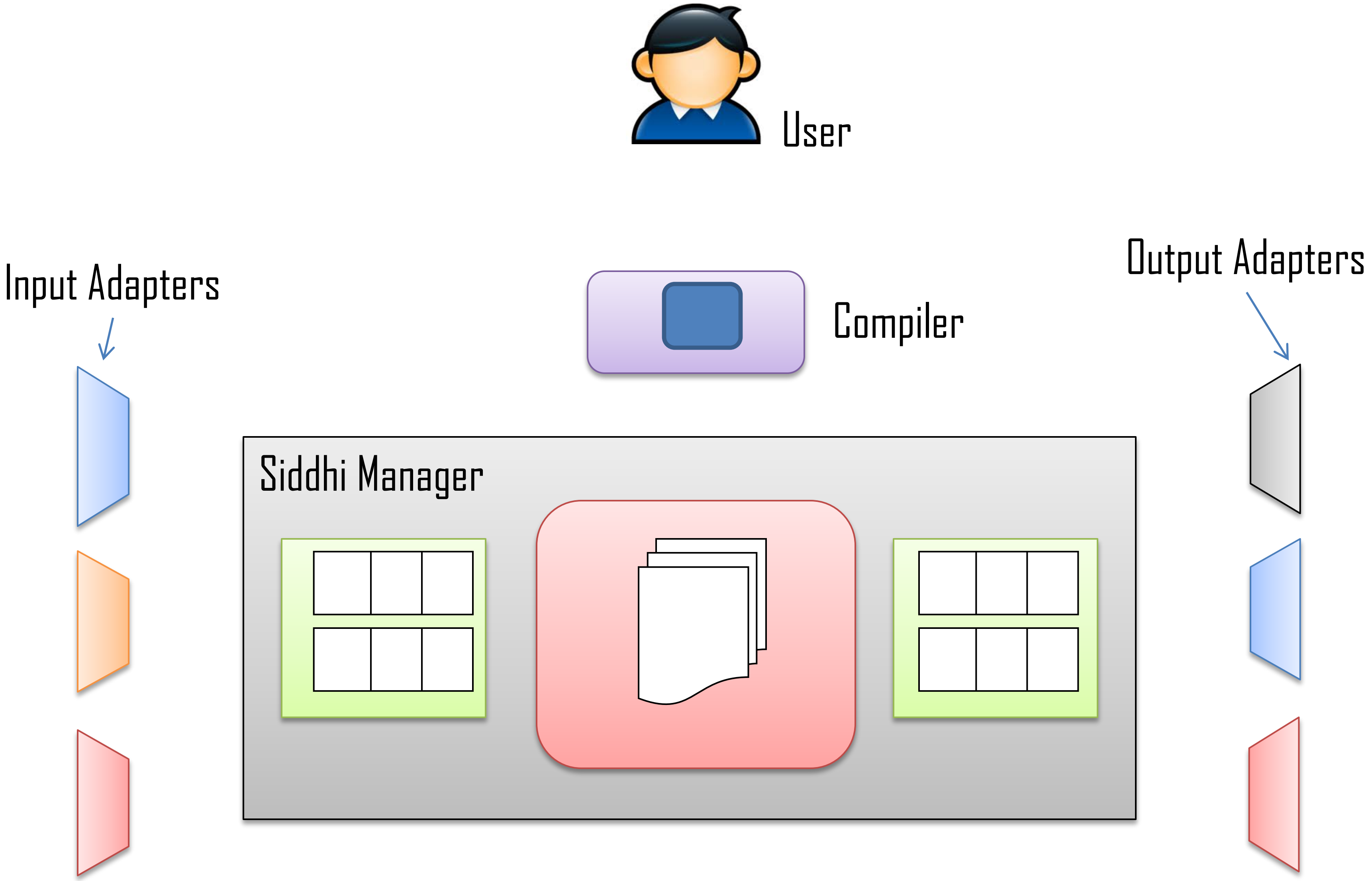
https://docs.wso2.com/display/CEP410/User+Guide



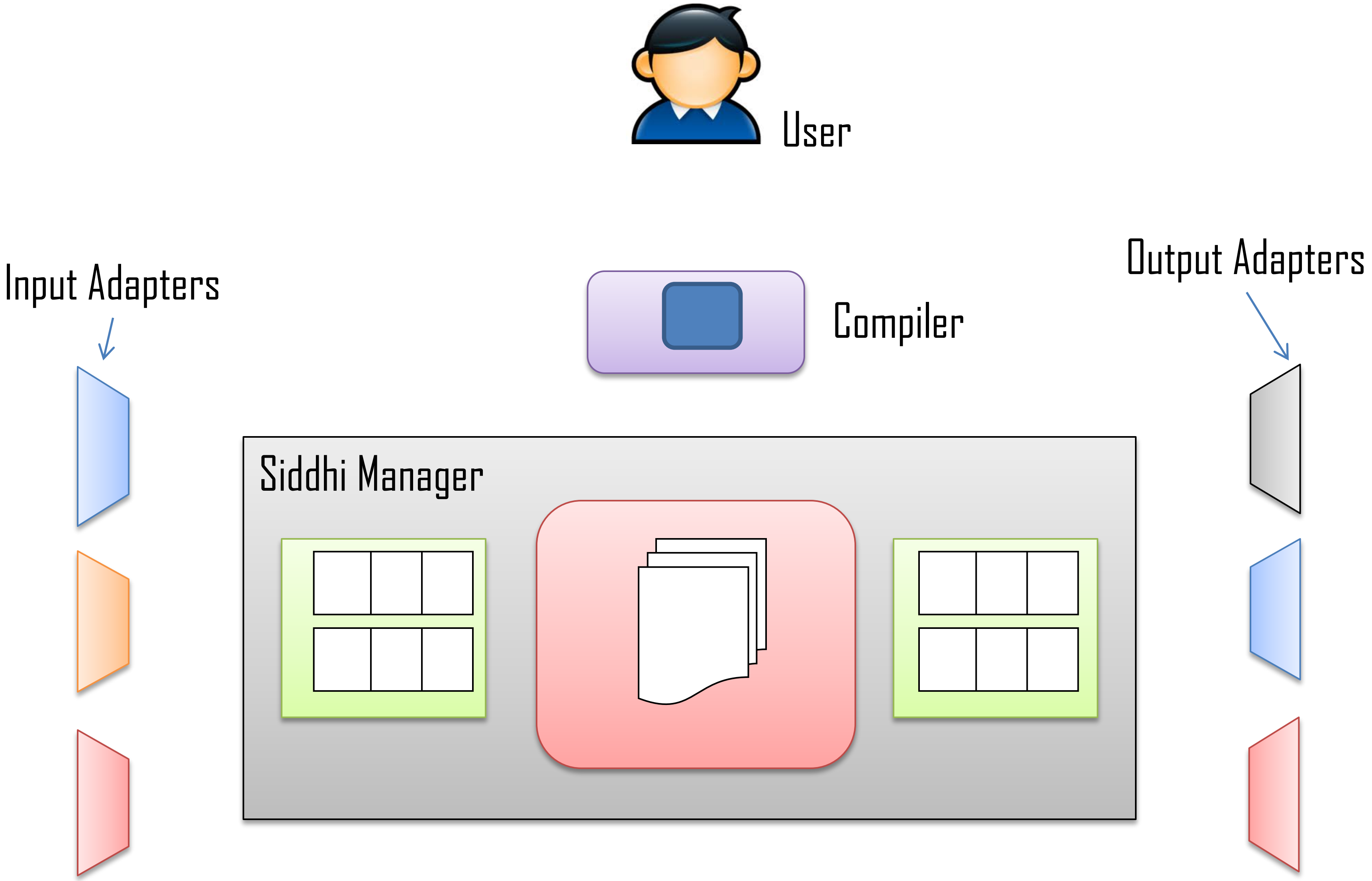
User Input a Query



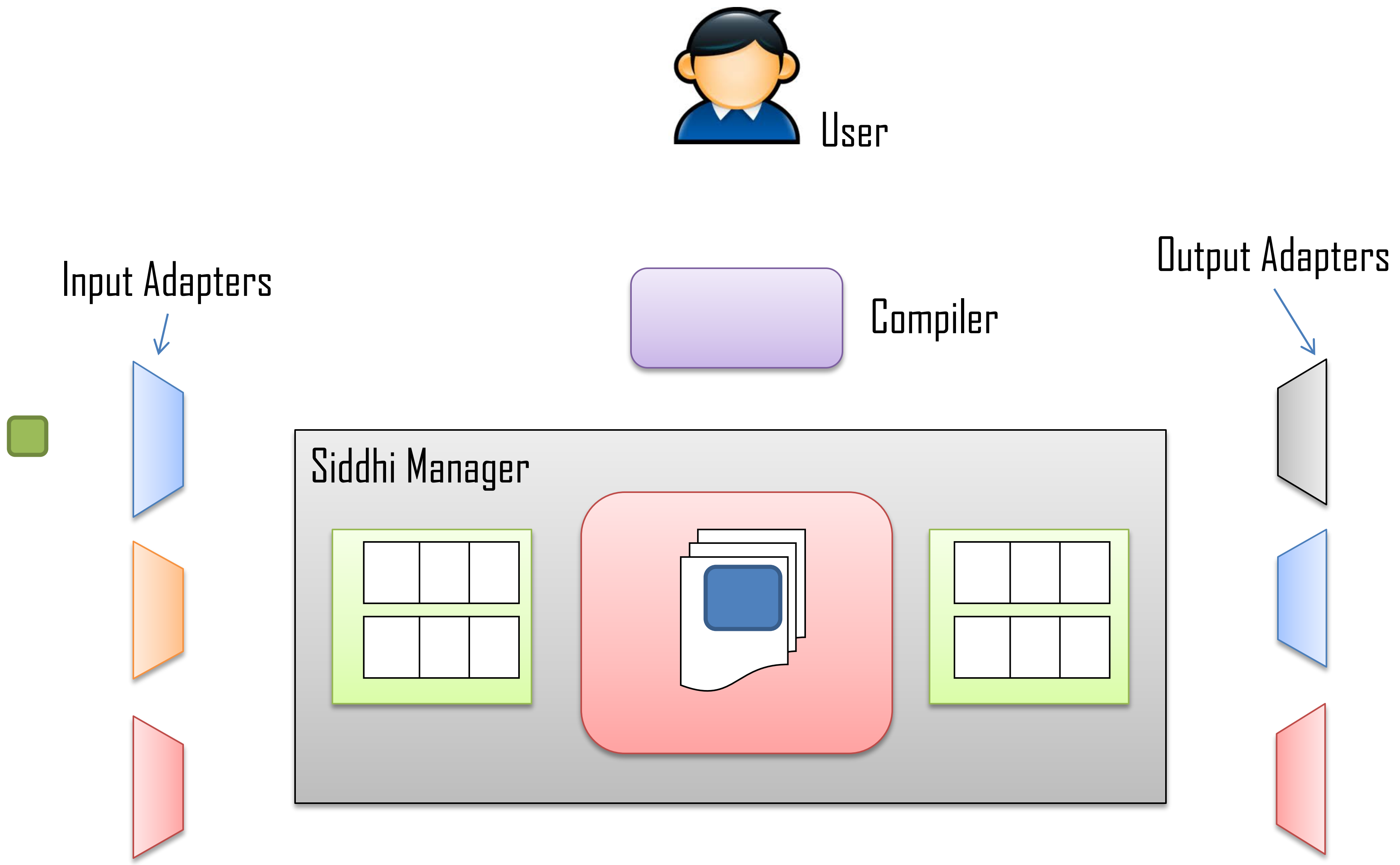
Compiling the query



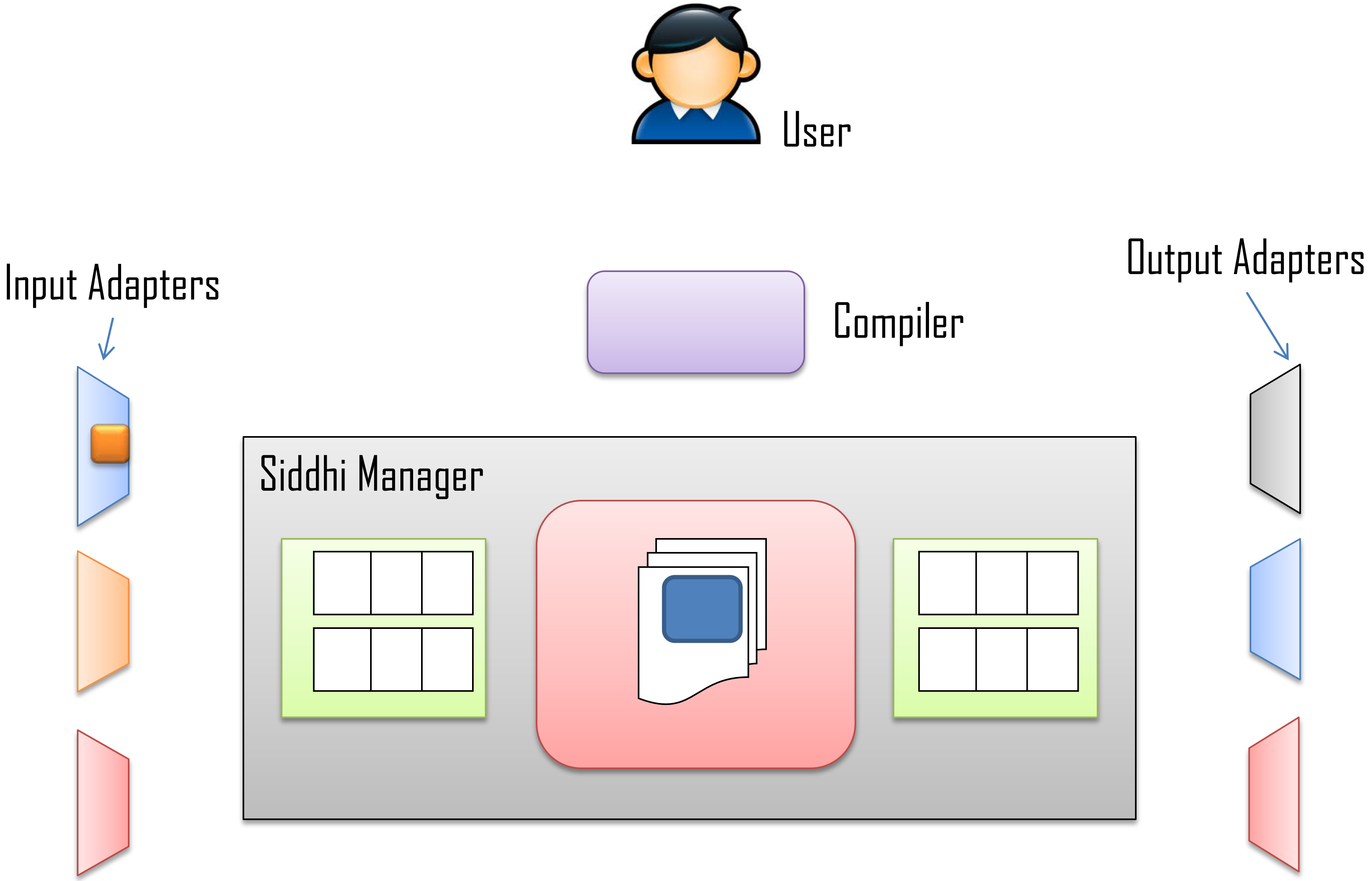
Query get compiled into an object model



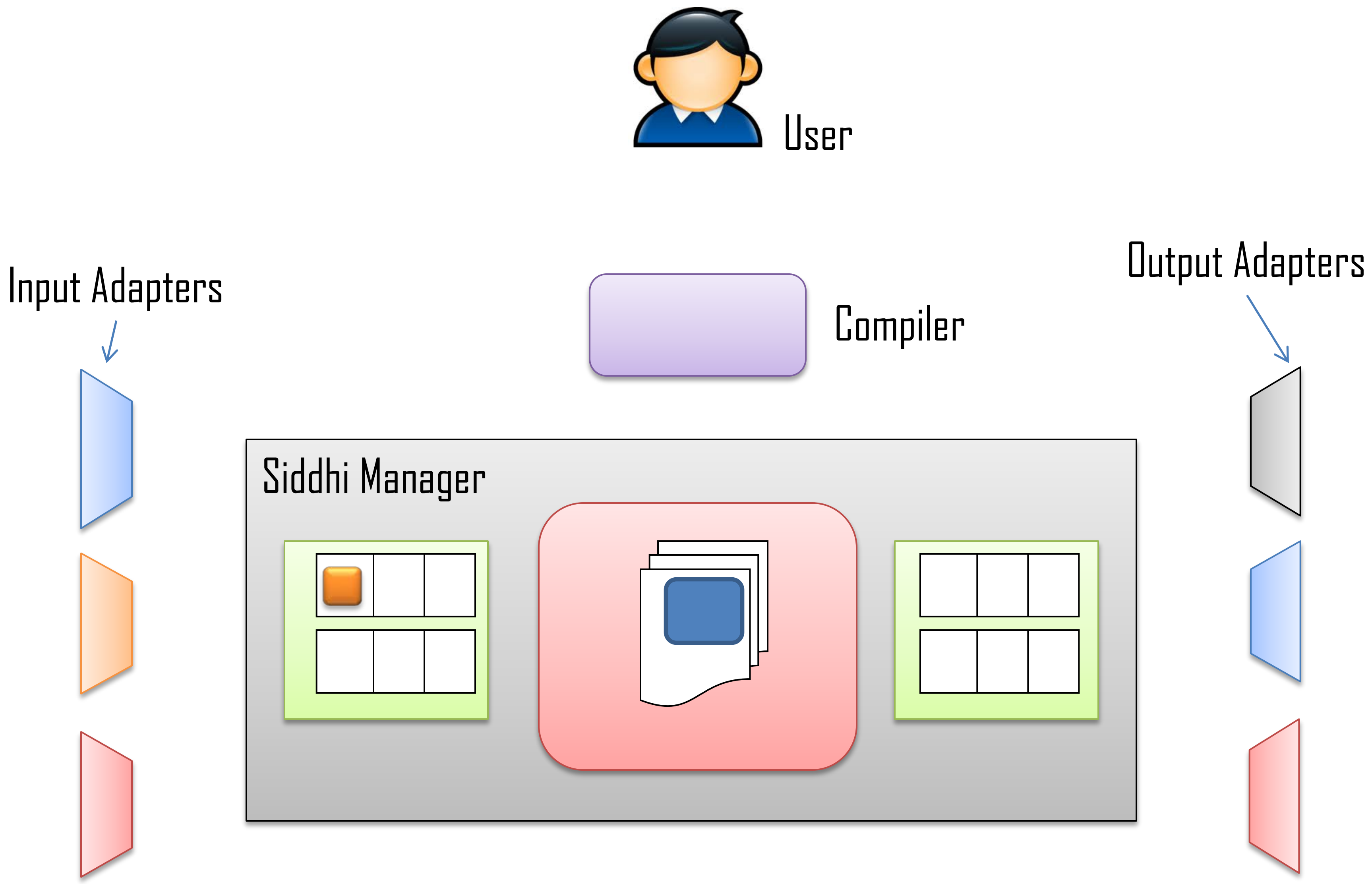
Query Object Model is parsed to the Siddhi Manager



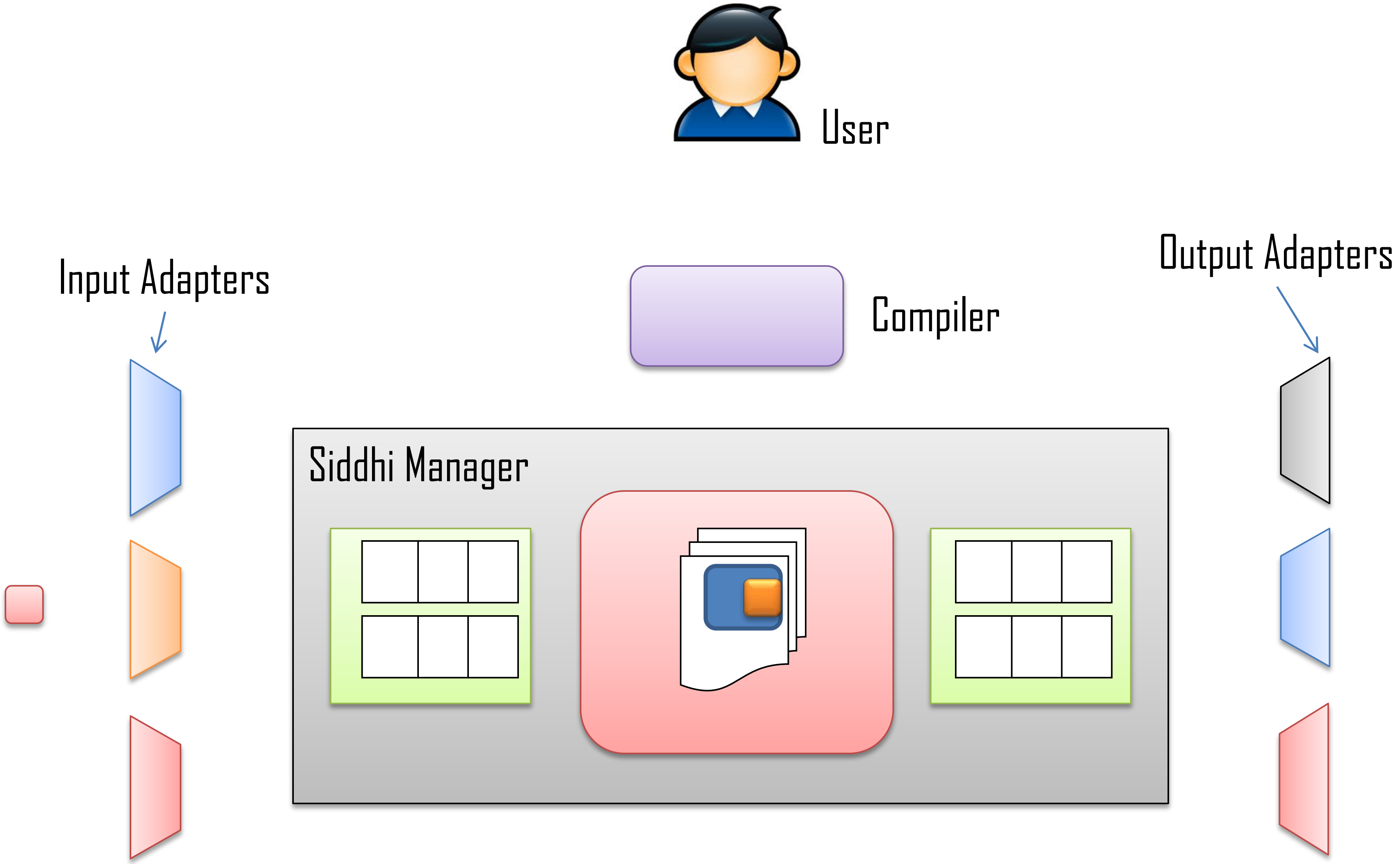
Event arrives to the Input Adapter



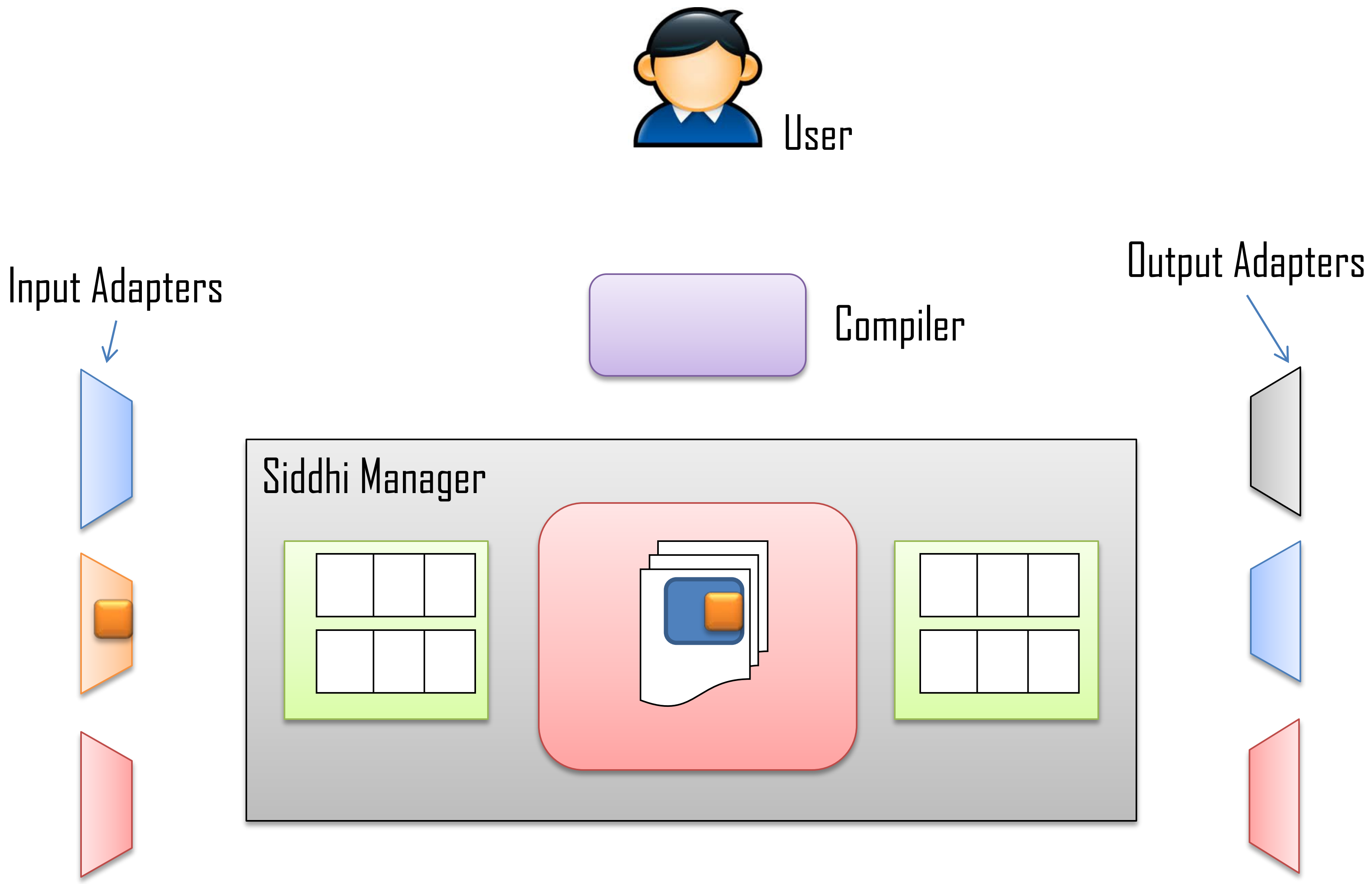
Convert Event to a Tuple and place it to the input event queue



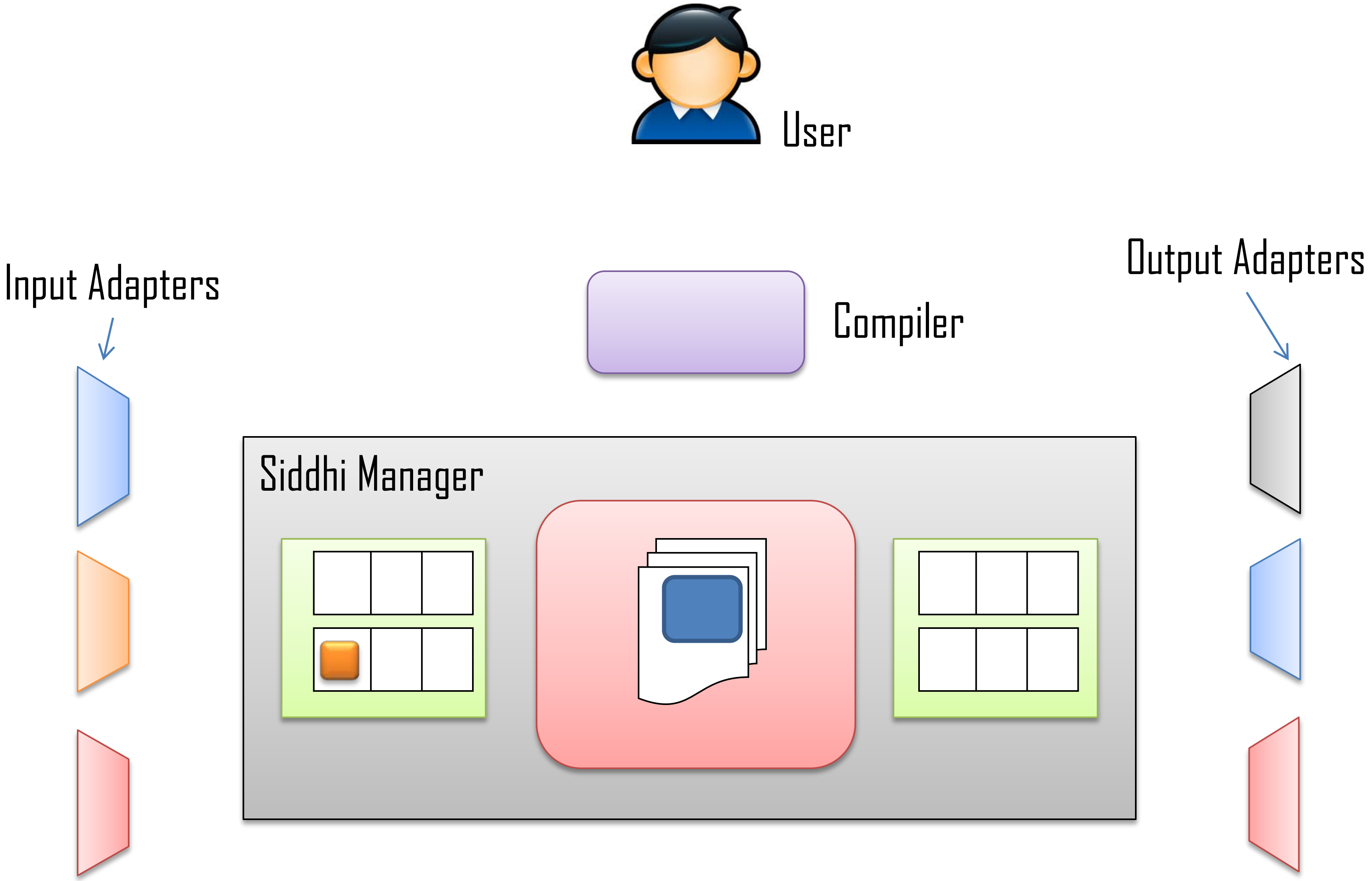
Processor takes the tuple from the queue

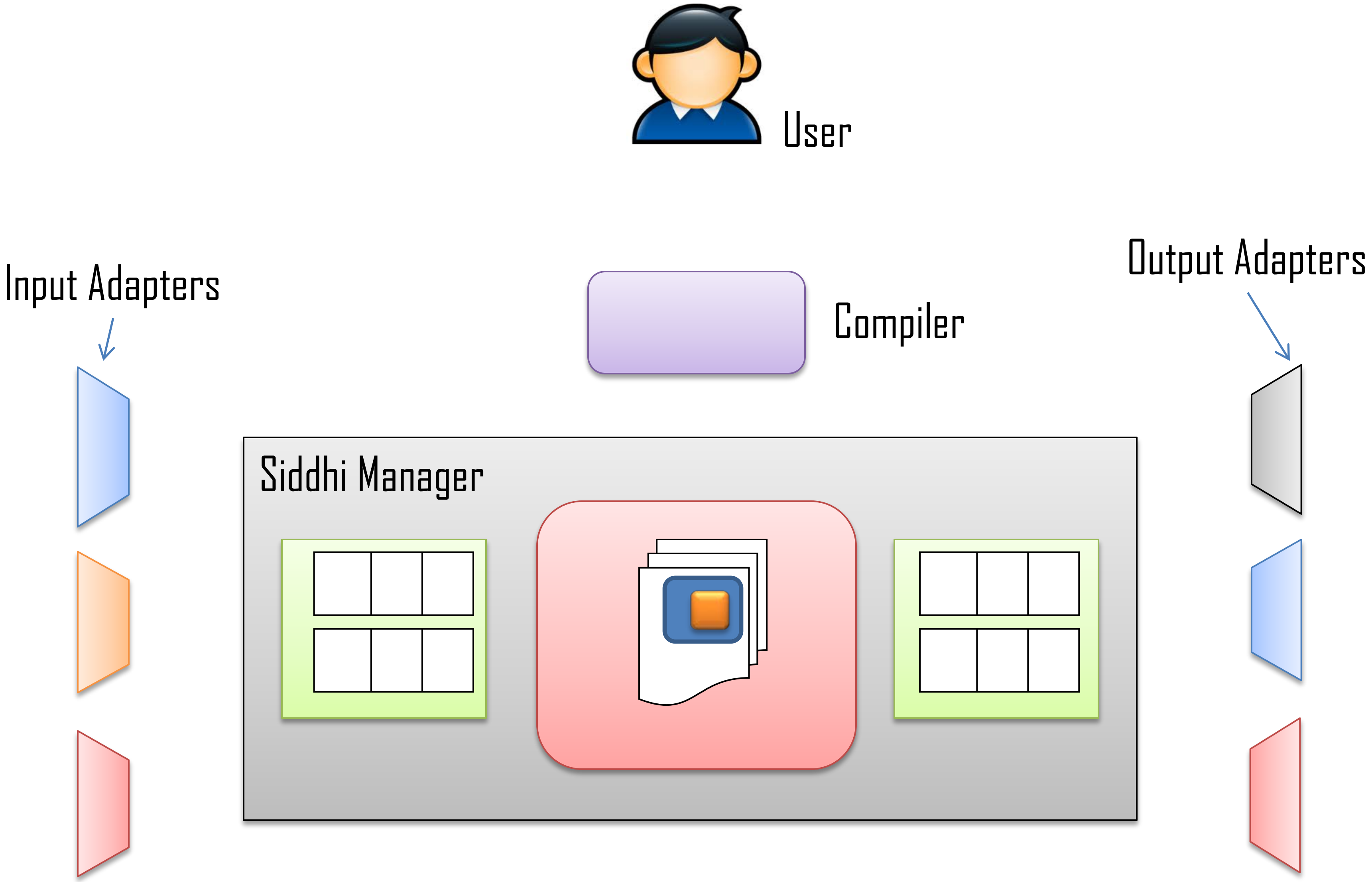


Executing the queries… Other Events arrives at the same time

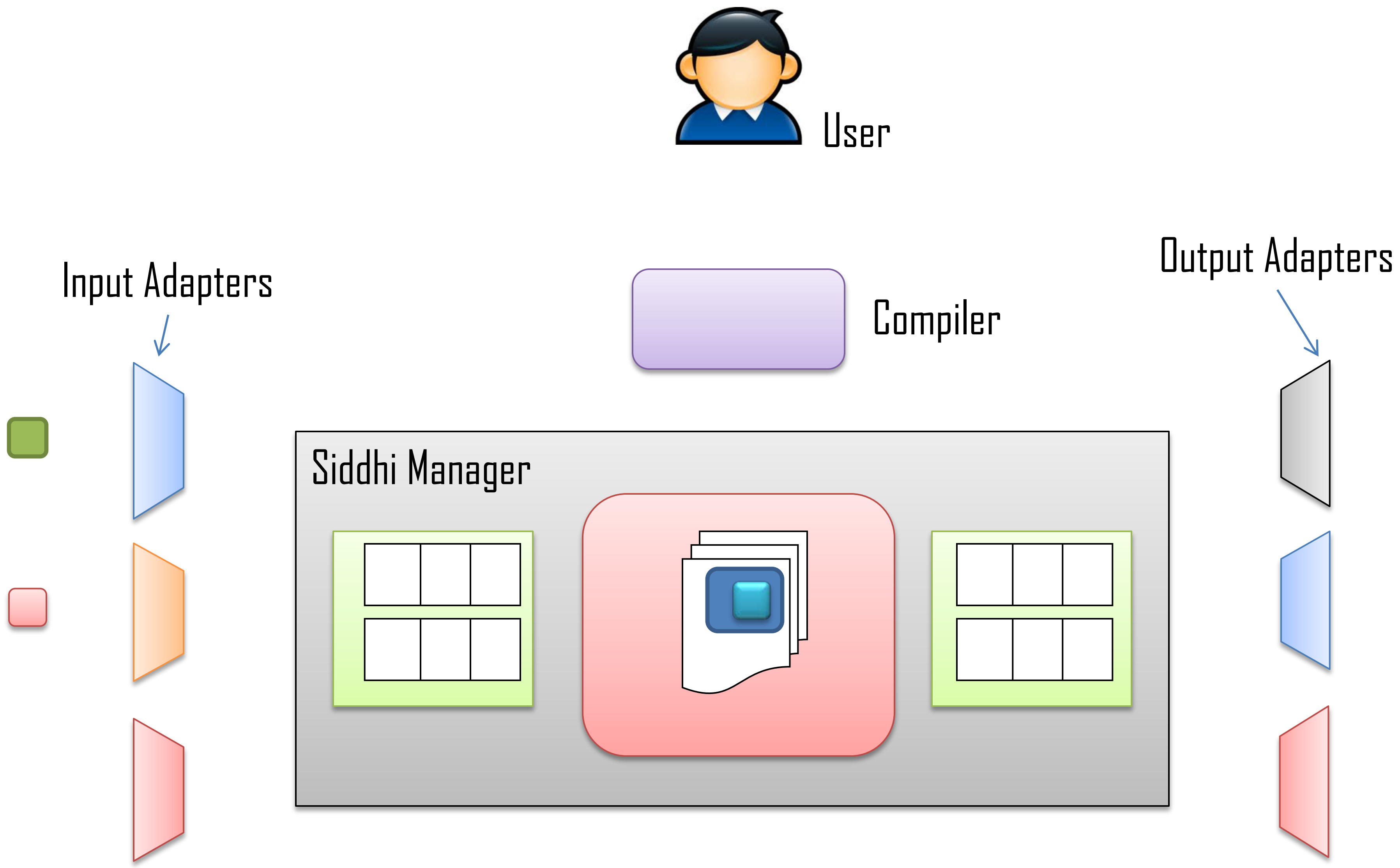


Non matching event thrown away

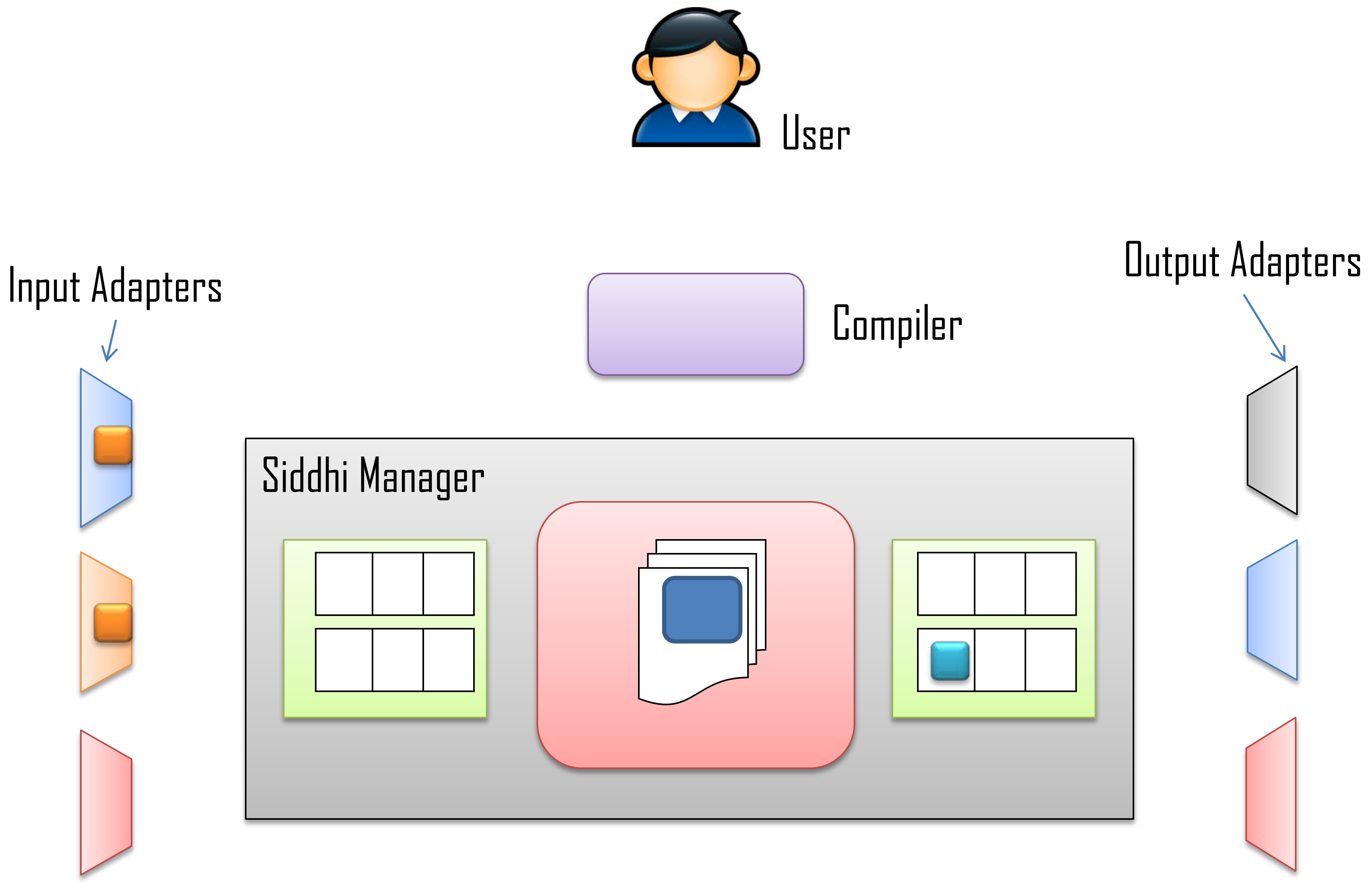

User
Input Adapters
Compiler
Output Adapters
Siddhi Manager



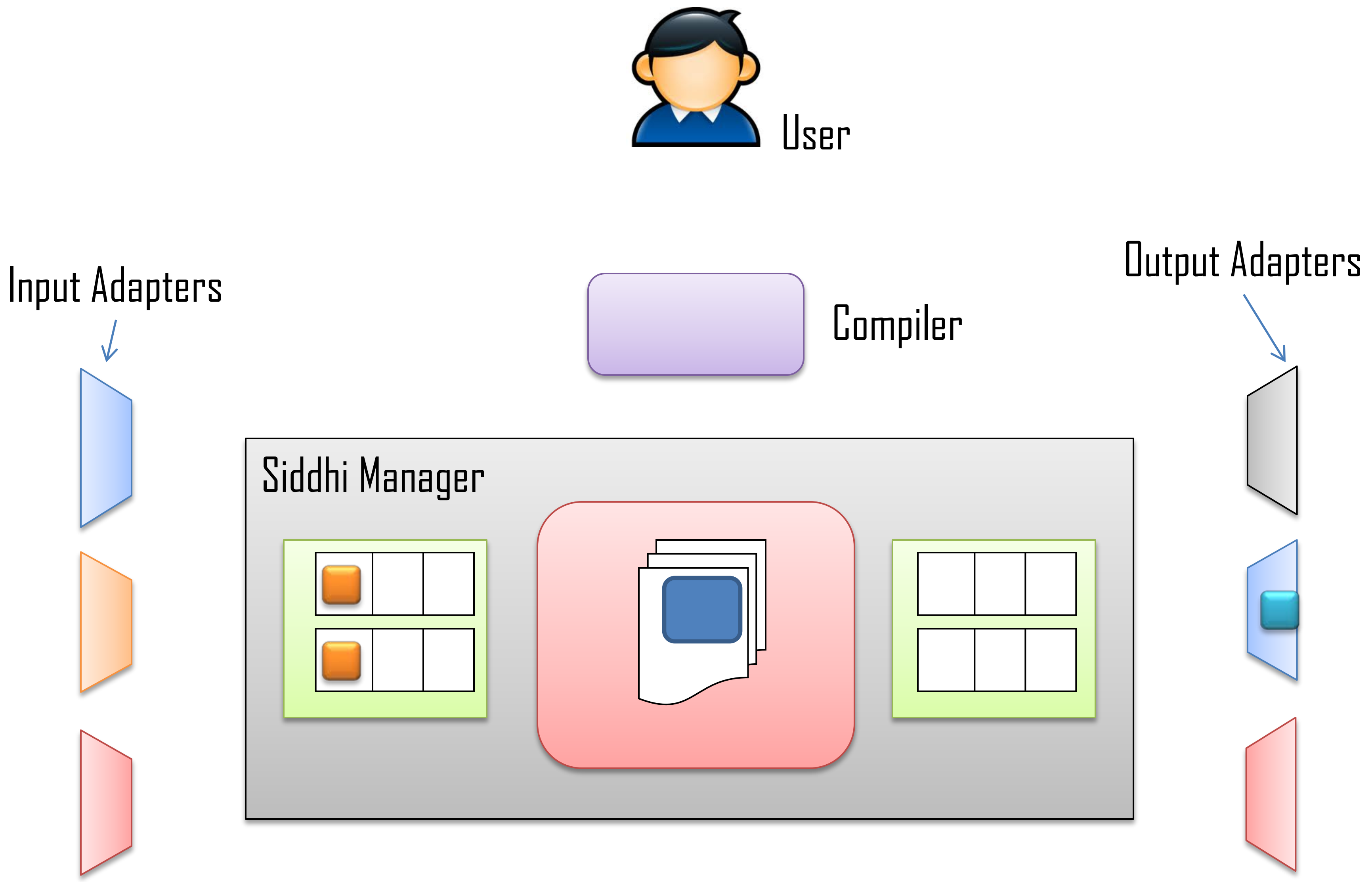
Matching Event creates the output Event



Pushing generated Events to the output queue

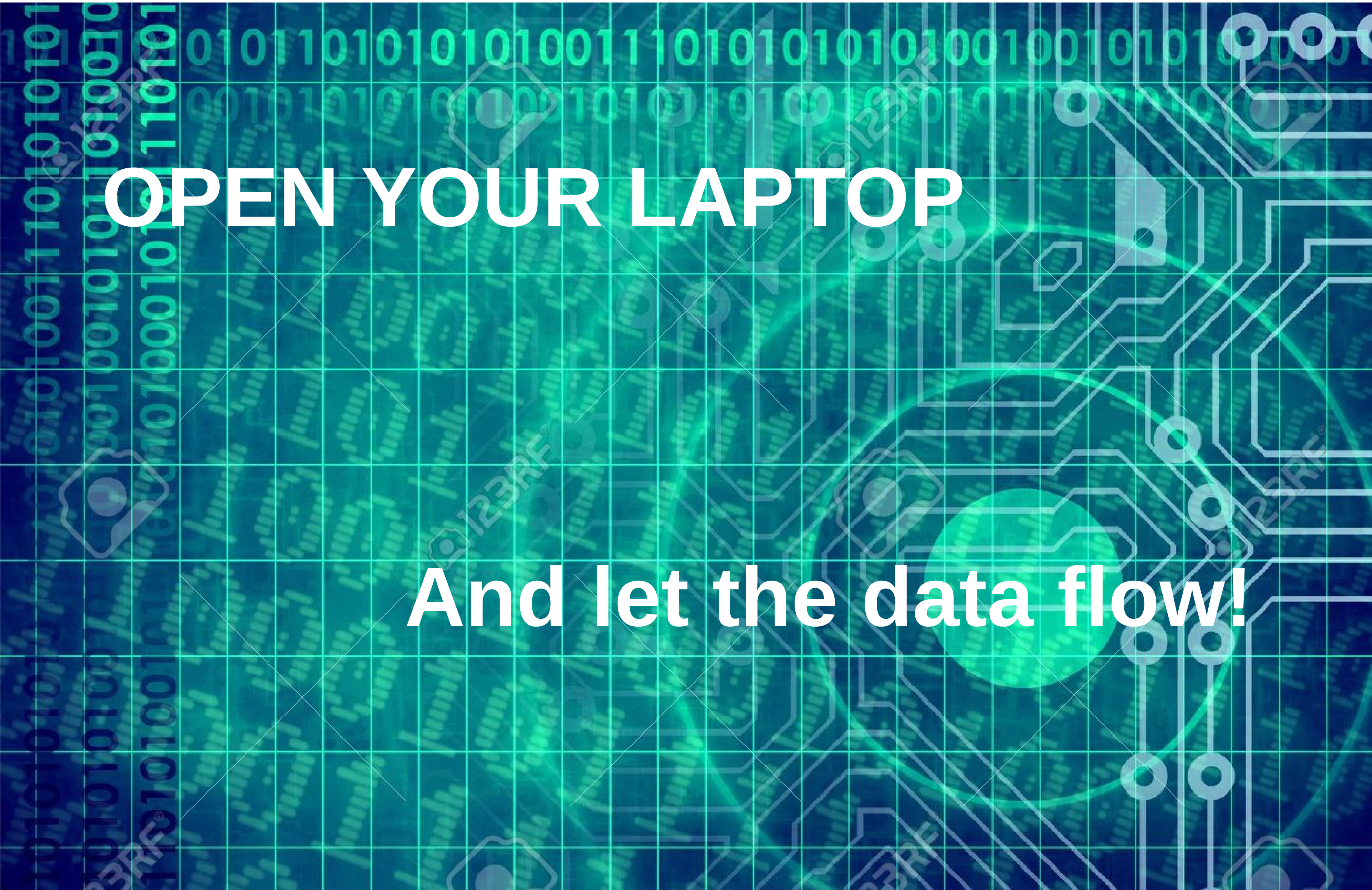


User get notified through output Adapter



Same procedure happens again and again…

Start of Hands On



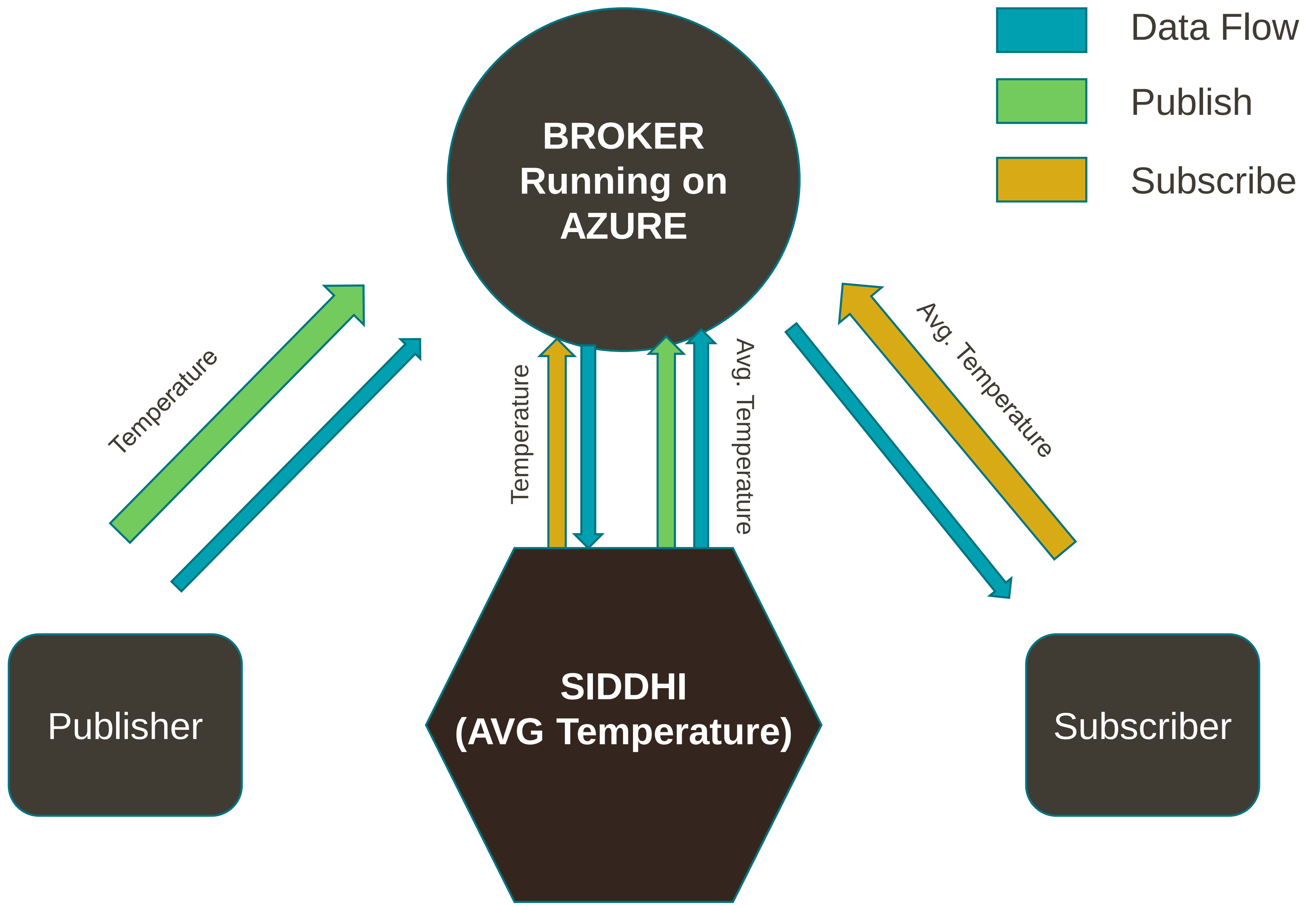
OPEN YOUR LAPTOP

And let the data flow!

DEMO SETUP

- Clone the siddhi-mqtt or iot-school folder
- Open **README.txt** file
  - cd **iot-school**/common/analytics/cep/**siddhi-mqtt**/
  - Find pom.xml here.
  - mvn clean compile package -Dmaven.test.skip=true
  - If possible import the project in JAVA IDE
  - Go through the source code
- Follow the steps in README.txt to
  - Subscribe
  - Perform analytics
  - Publish (result)

Overall Picture



Useful References

- https://github.com/wso2/siddhi
- https://docs.wso2.com/display/CEP400/SiddhiQL+Guide+3.0
- https://docs.wso2.com/display/CEP400/Siddhi+Try+It+Tool
- https://docs.wso2.com/display/CEP300/WSO2+Complex+Event+Processor+Documentation

THANKS!

Any Questions?

Best of luck with CEP!