INTRODUCTION TO SPARK WITH SCALA

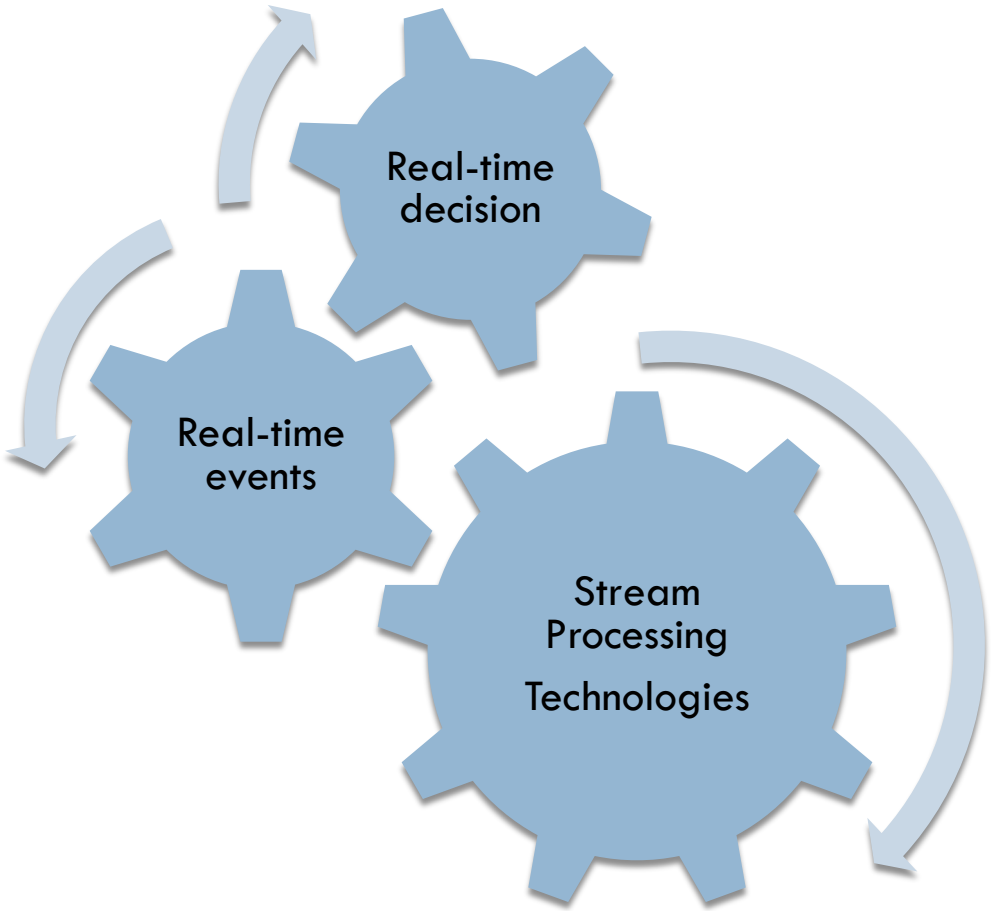
Spark Structured Streaming

Agenda

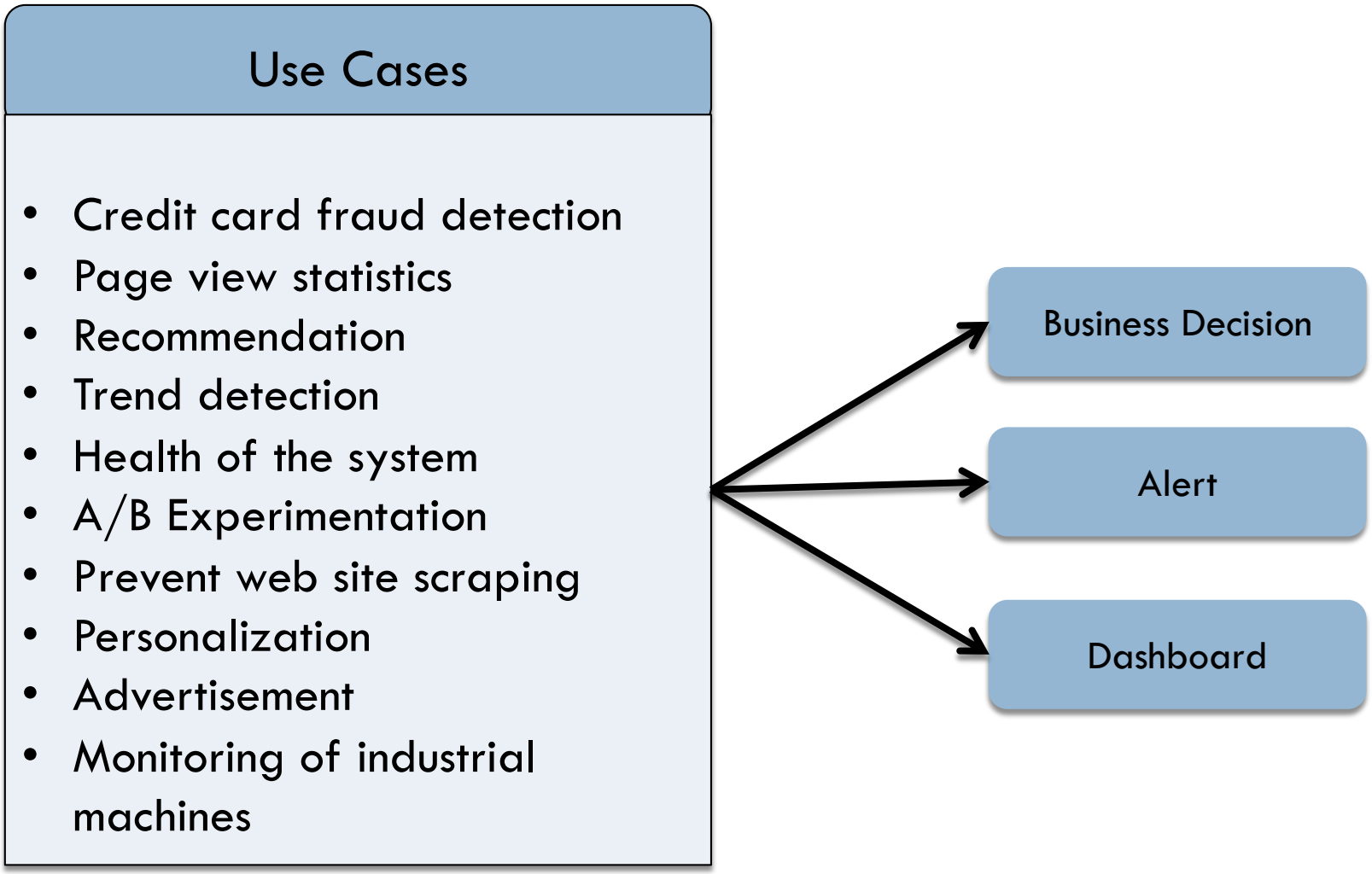
- Stream Processing
- Overview of Structured Streaming
- Structured Streaming Concepts
- Structured Streaming API
  - Datasets & DataFrames
- Working with Streaming Queries
- Execution Model
- Fault Tolerance
- Resources

Stream Processing

The Rise of Real Time

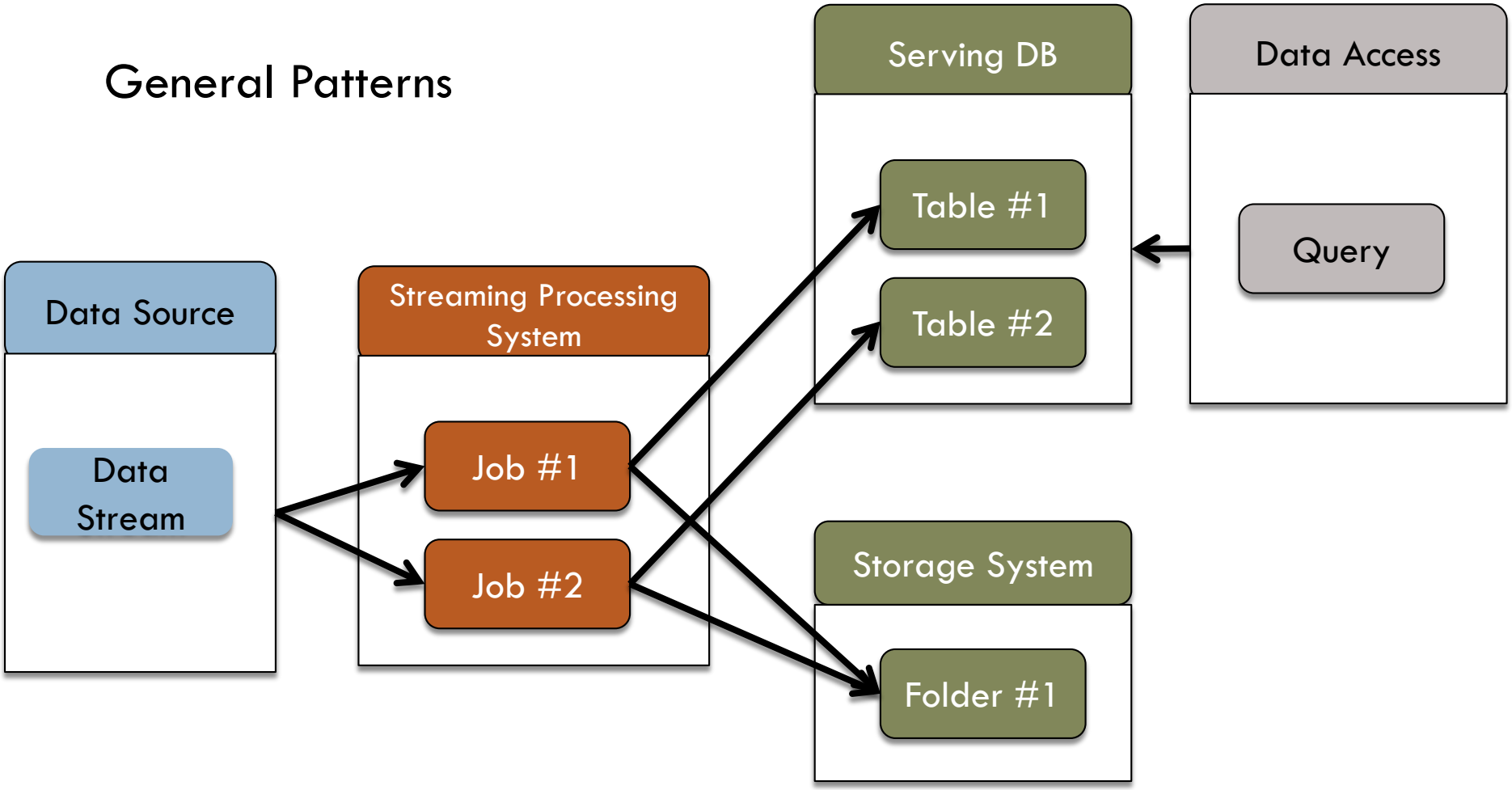


Stream Processing



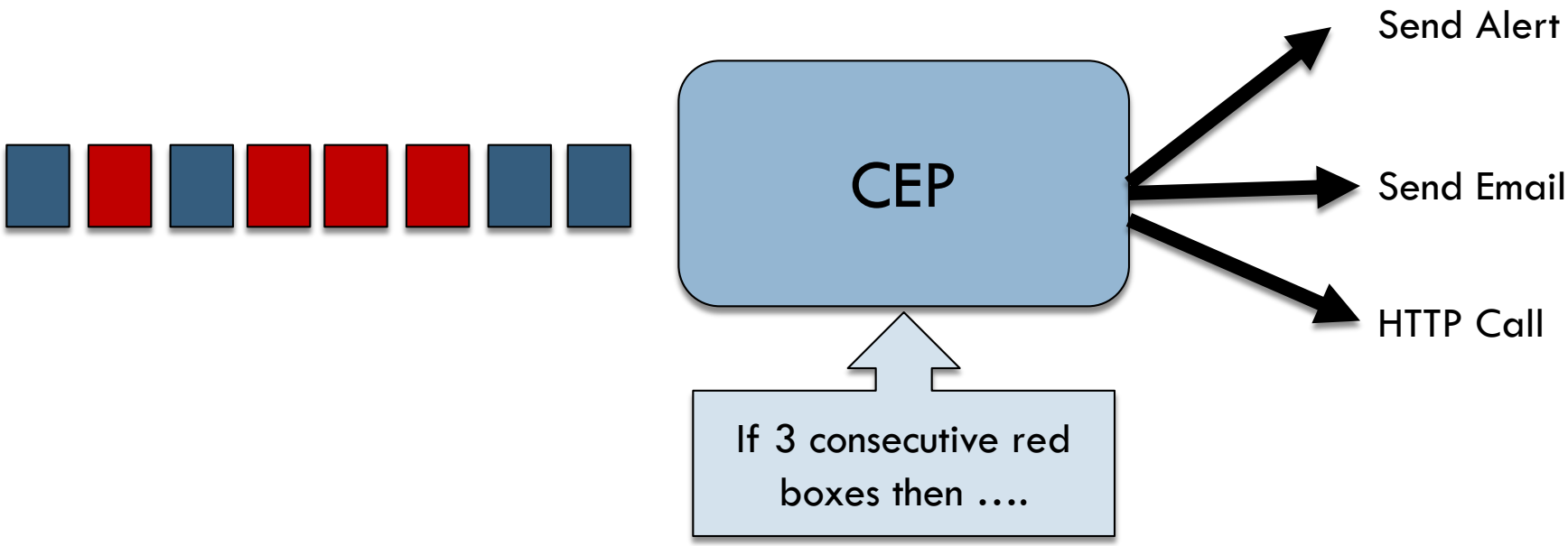
Stream Processing

General Patterns



Stream Processing

- Complex Event Processing
  - Detect complex event patterns in streams of data



Stream Processing

Messaging Semantics

Exactly once => 1 only

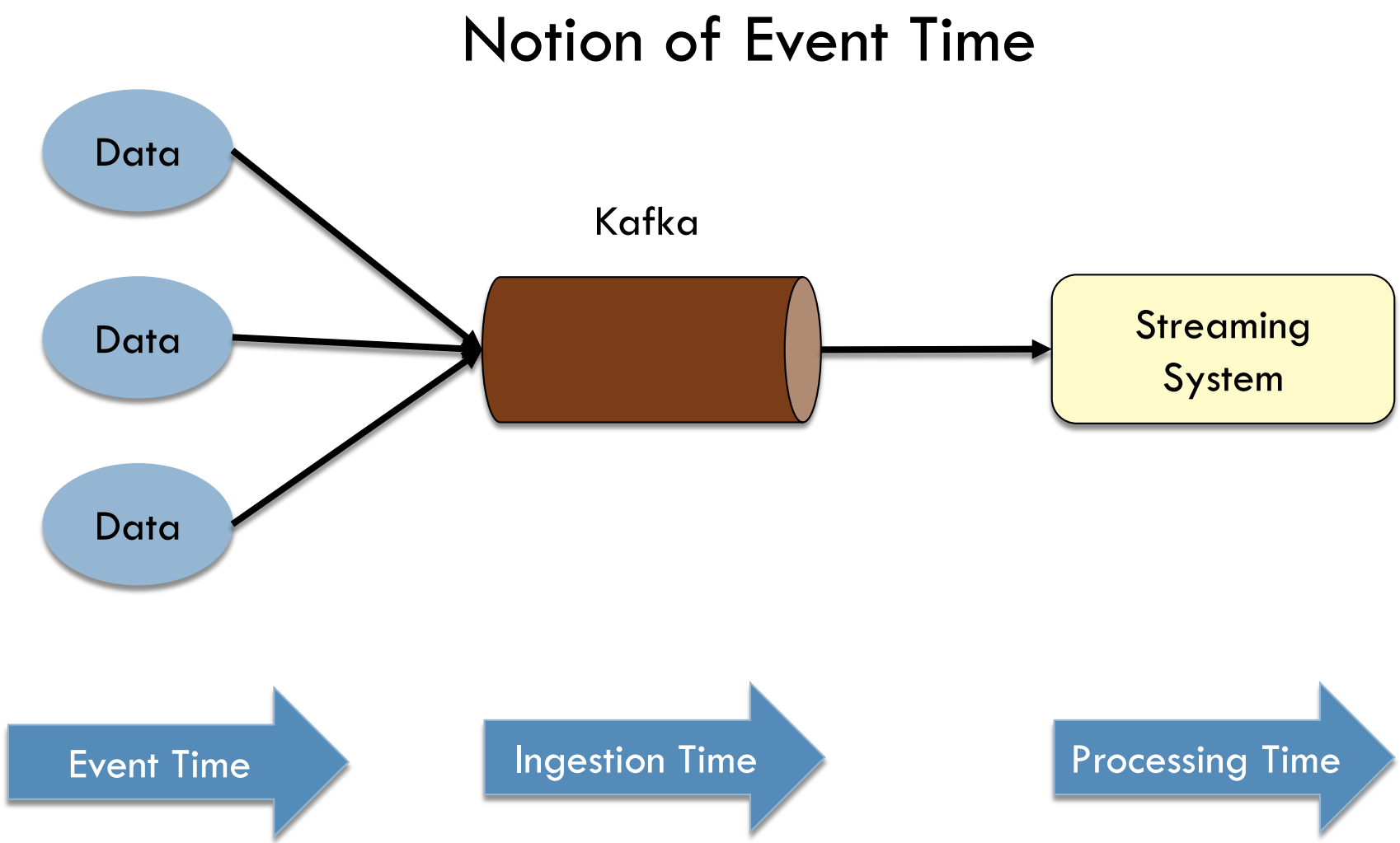
At least once => 1 or n

At most once => 1 or 0

Strongest

Weak

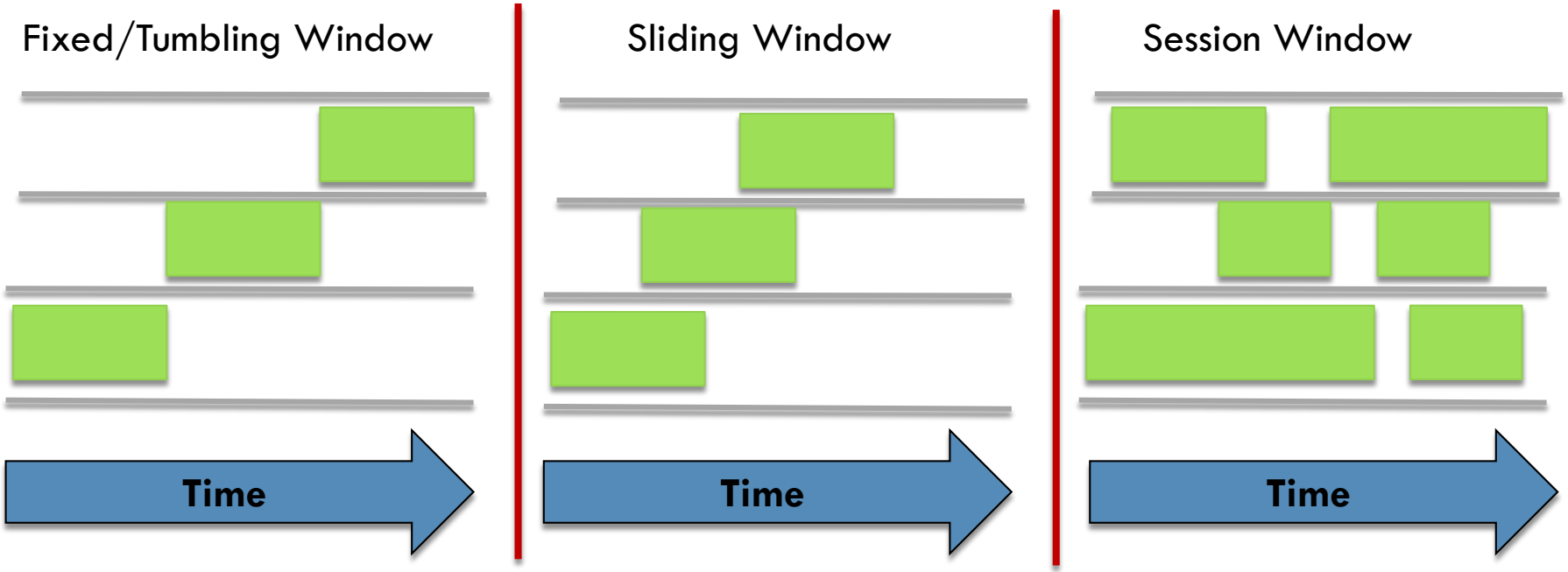
Stream Processing



Stream Processing

Windowing

- Slicing up incoming data into chunks for processing
- Delineate finite boundaries for grouping



Stream Processing

Data Streaming Processing Systems

(Heron)

Apache Beam

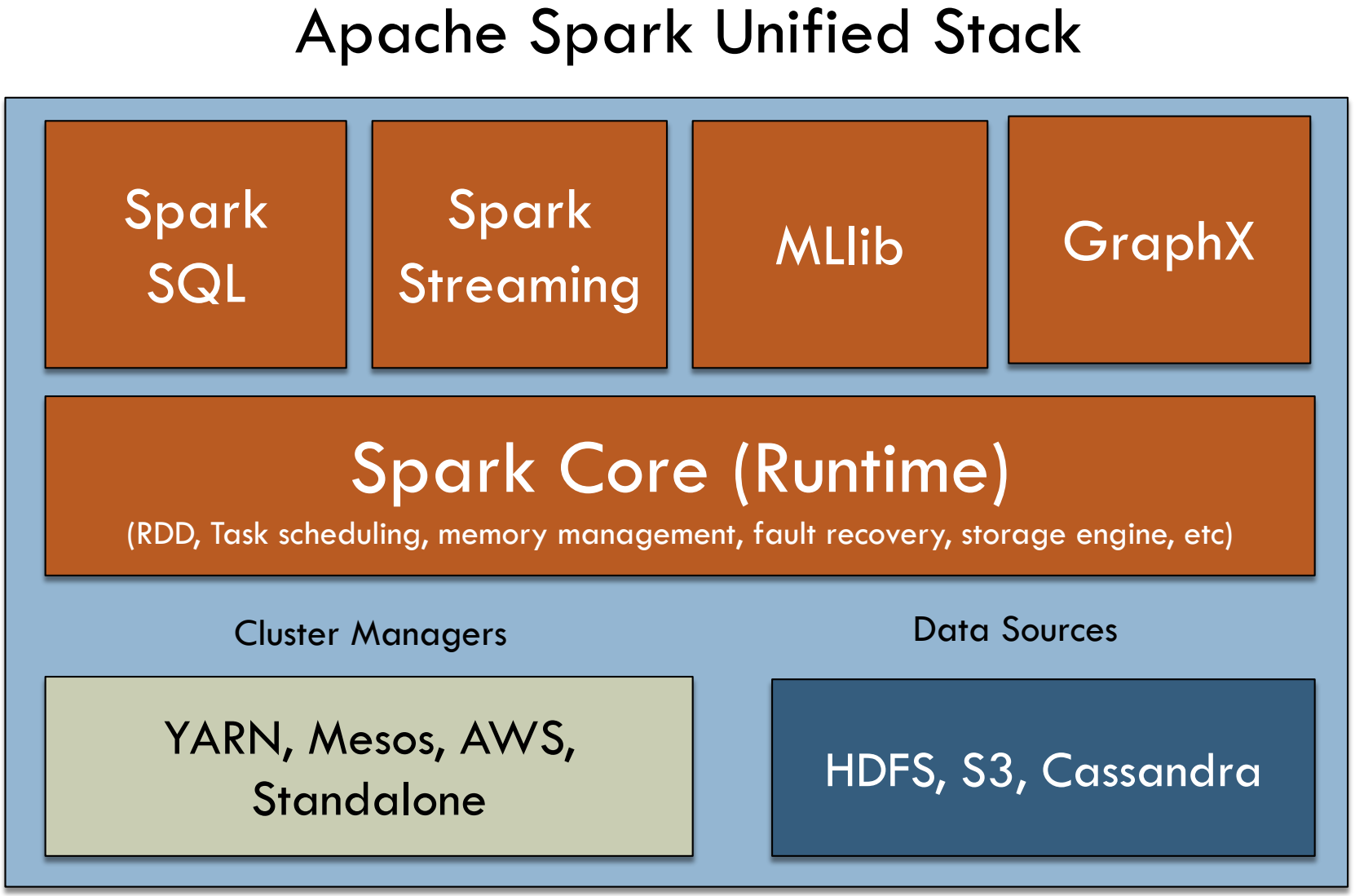
Stream Processing

Requirements for Streaming Processing Engine

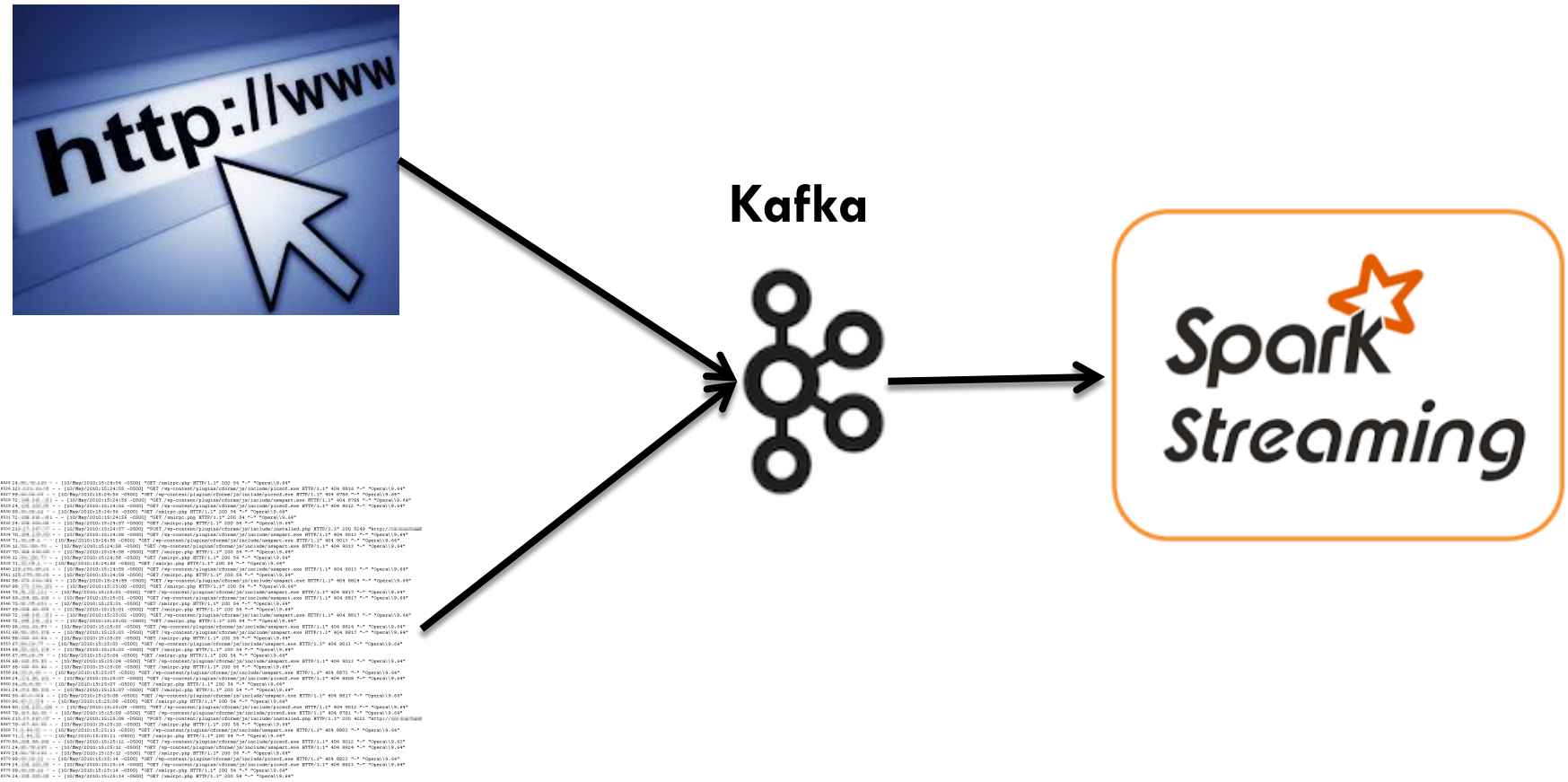
- Keep the data moving
  - Streaming architecture
- Declarative access
  - E.g. StreamSQL, CQL
- Handle imperfections
  - Late, missing, unordered items
- Predictable outcomes
  - Consistency, event time
- Integrate stored and streaming data
  - Hybrid stream and batch
- Data safety and availability
  - Fault tolerance, durable state
- Automatic partitioning and scaling
  - Distributed processing
- Instantaneous processing and response

The 8 Requirements of Real-Time Stream Processing – Stonebraker et al. 2005

Spark Streaming Concepts



Spark Streaming Concepts

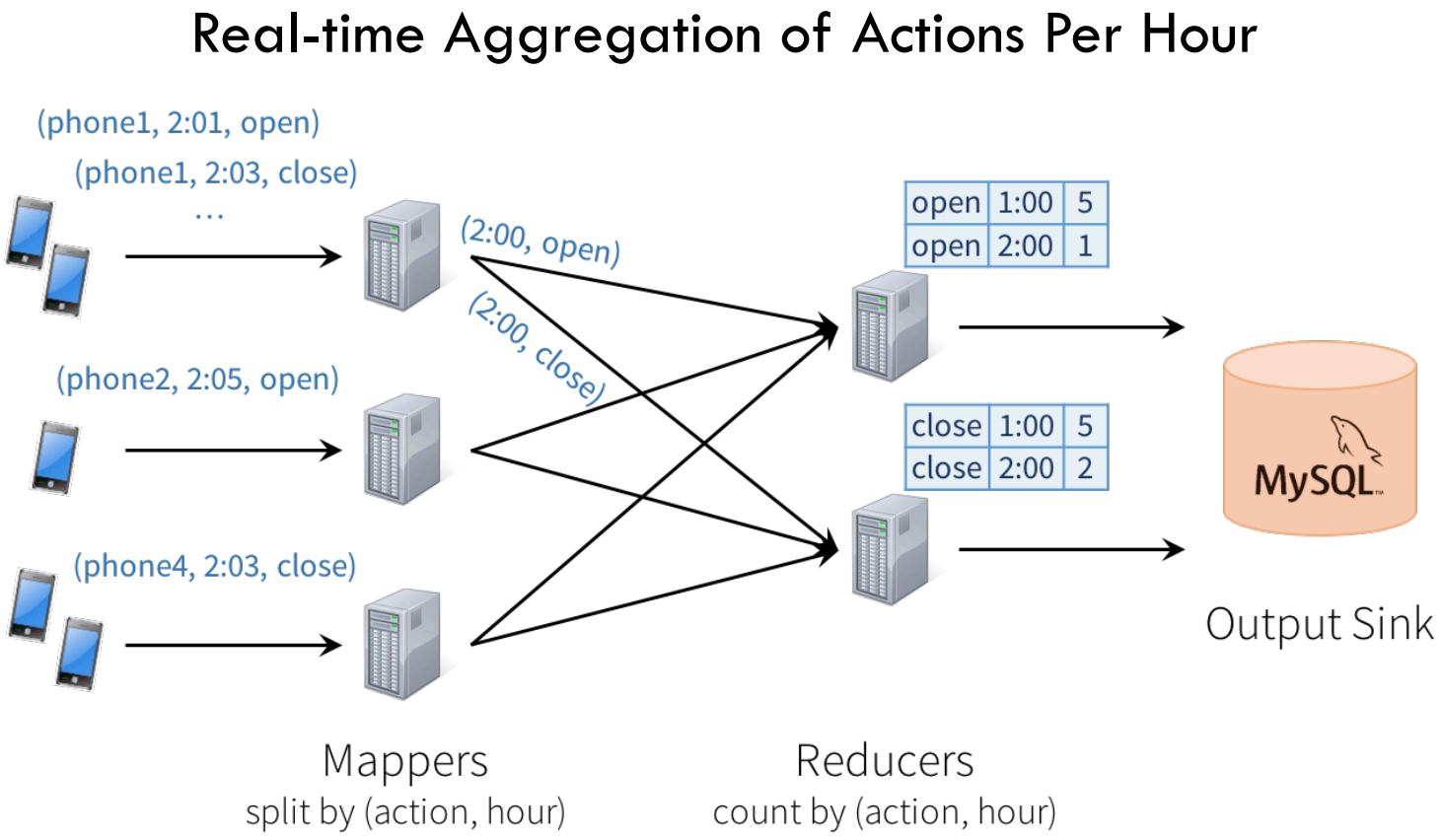


Event data

Buffering

Processing

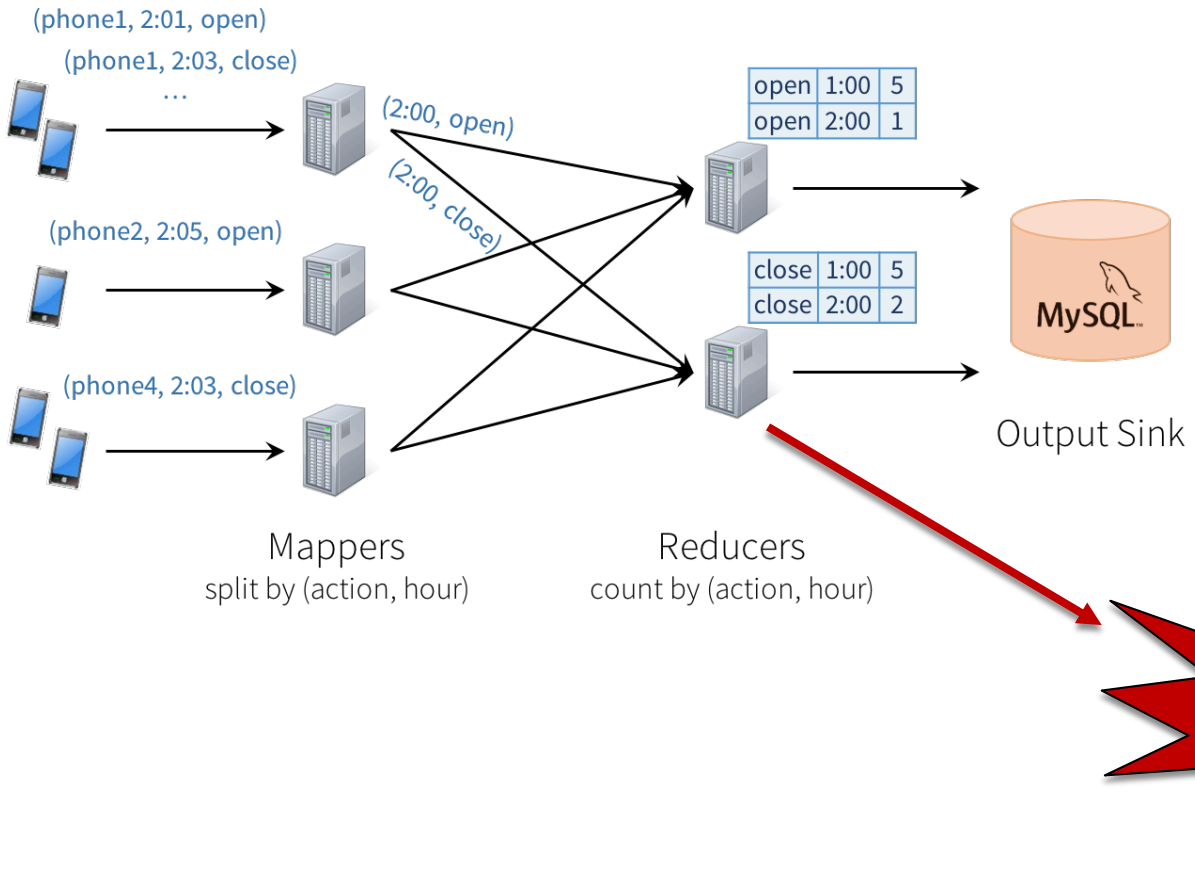
Overview of Structured Streaming



https://databricks.com/blog/2016/07/28/structured-streaming-in-apache-spark.html

Overview of Structured Streaming

Real-time Aggregation of Actions/Hour



Potential Issues:

- Consistency
- Fault tolerance
- Out-of-order data

Overview of Structured Streaming

- Streaming applications are complex
  - Integrating with other systems
    - Interactive, batch, RDBMS, machine learning
  - Dealing with late event or out-of-order data
  - Fault tolerance & consistency
- DStream
  - No event time processing support
  - Limited interoperability between batch and interactive
  - Difficult to reasoning about end-to-end guarantees

Overview of Structured Streaming

*The simplest way to perform streaming analytics is not having to **reason** about streaming*

- Reynold Xin, chief architect @ Databricks

Overview of Structured Streaming

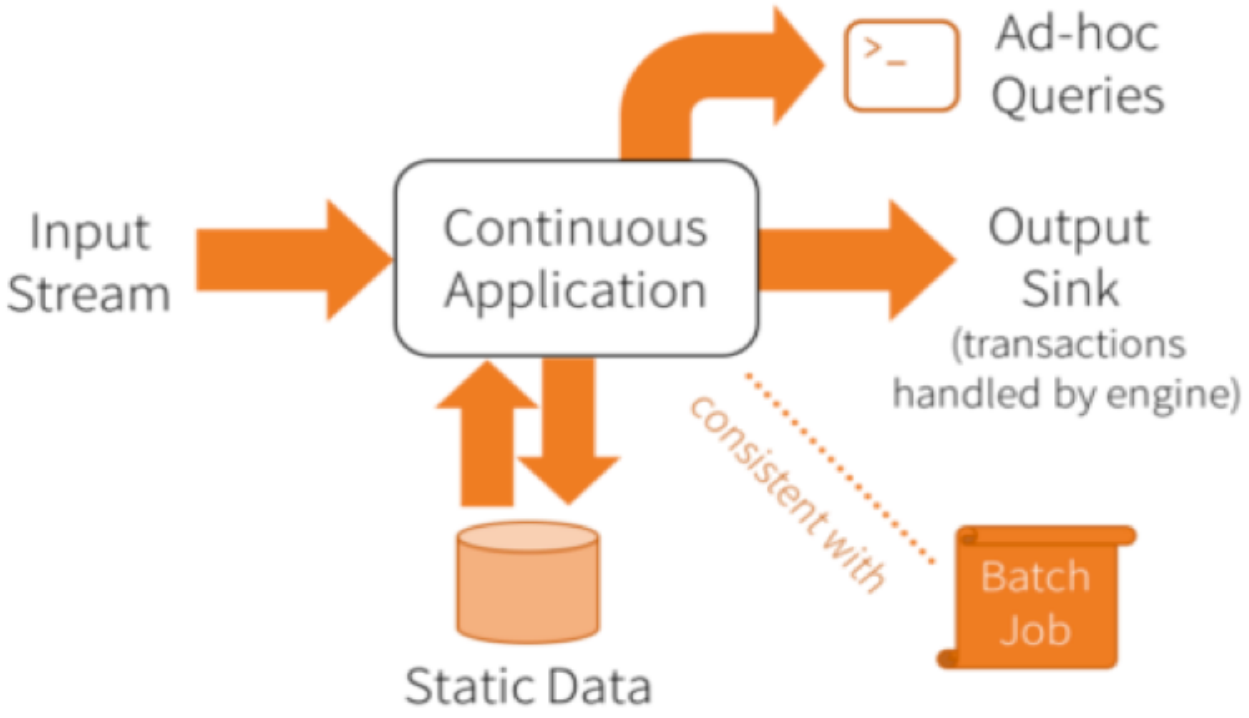
Structured Streaming provides fast, scalable, fault-tolerant, end-to-end exactly once streaming processing without user having to reason about streaming

Overview of Structured Streaming

- High-level API built on top of Spark SQL engine
  - Streaming computation similar to batch computation
- Strong guarantees about consistency with batch jobs
- Ensure end-to-end exactly-once guarantees
  - Transactional integration with storage system
- Handling late or out-of-order data
- Tight integration with the rest of Spark
  - Serving interactive queries on streaming state
  - Integrate with MLlib

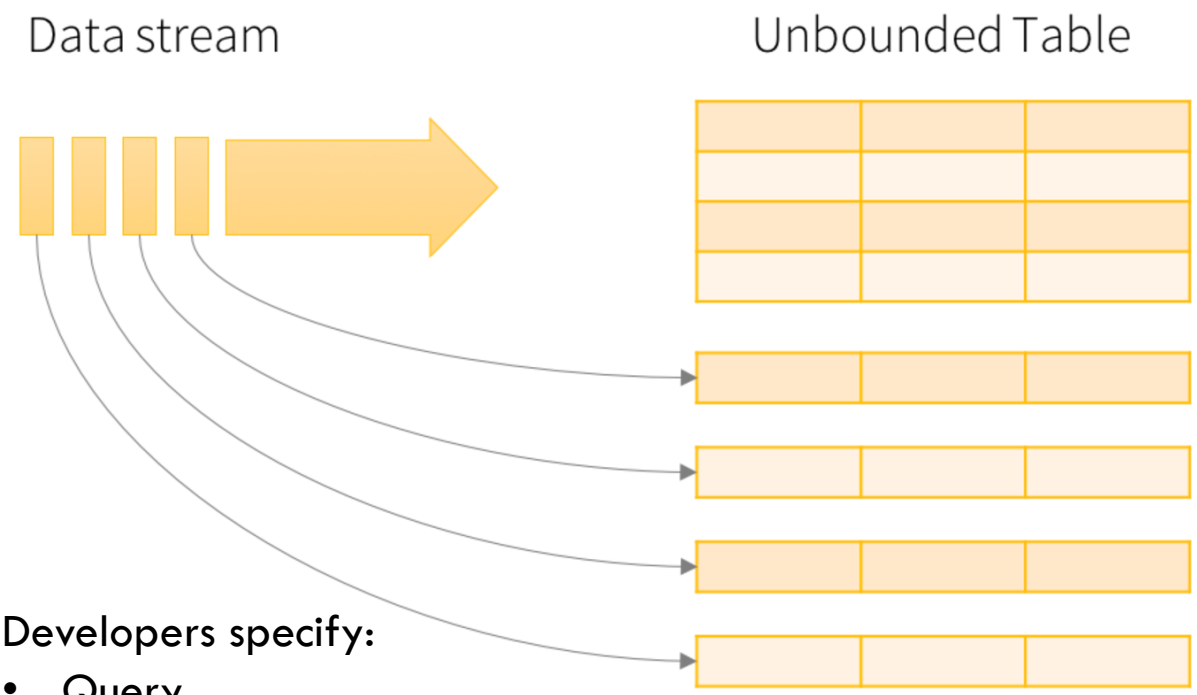
Overview of Structured Streaming

Continuous application – an end-to-end application that reacts to data in real-time



Structured Streaming Concepts

Data Stream as An Unbounded Table



new data in the data stream

=

new rows appended to a unbounded table

Developers specify:

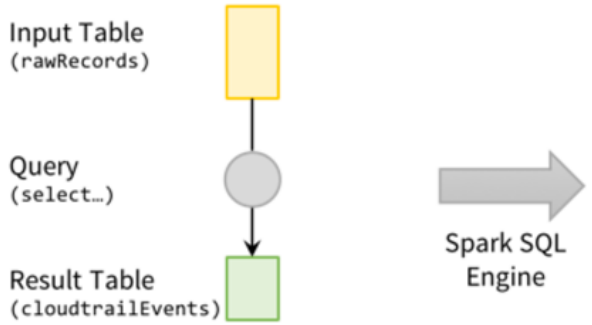
- Query
- Trigger point

http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html

Structured Streaming Concepts

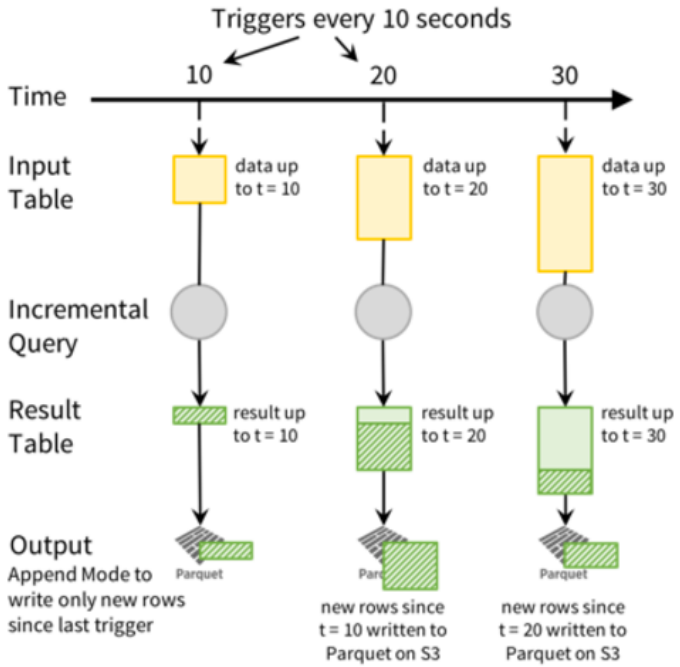
Structured Streaming Processing Model

Batch Processing



user's query on Input Table using batch API

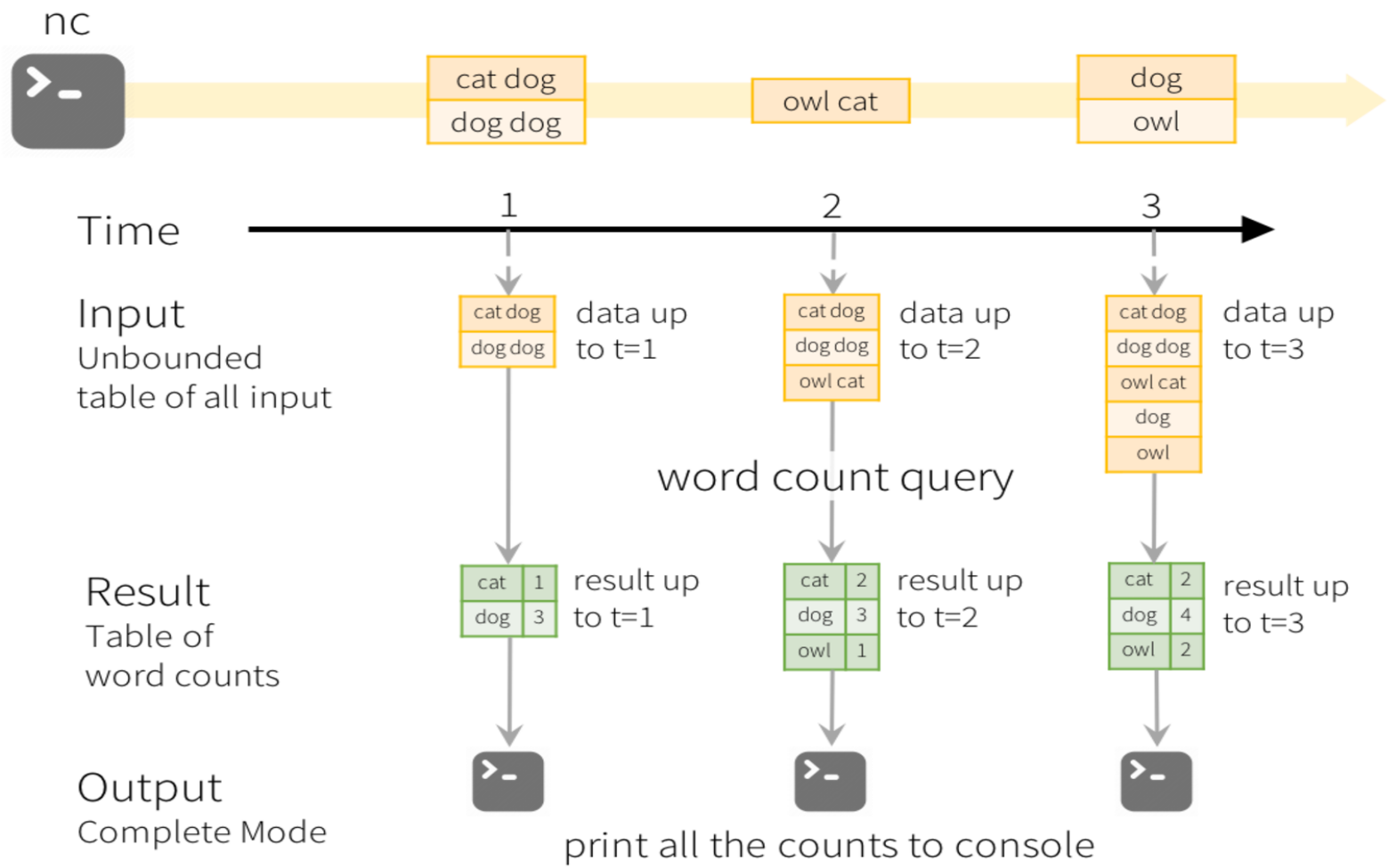
Stream Processing



incremental execution on streaming data

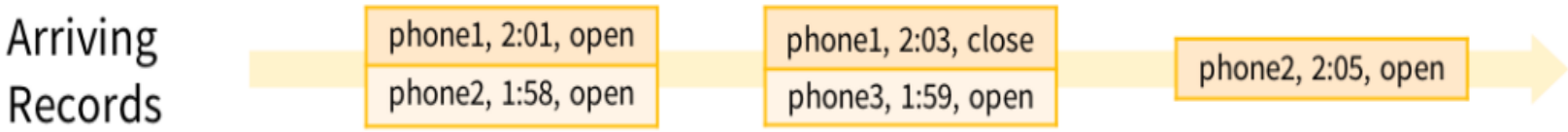
Structured Streaming Concepts

Structured Streaming Word Count Example



Structured Streaming Concepts

Events from Mobile Application

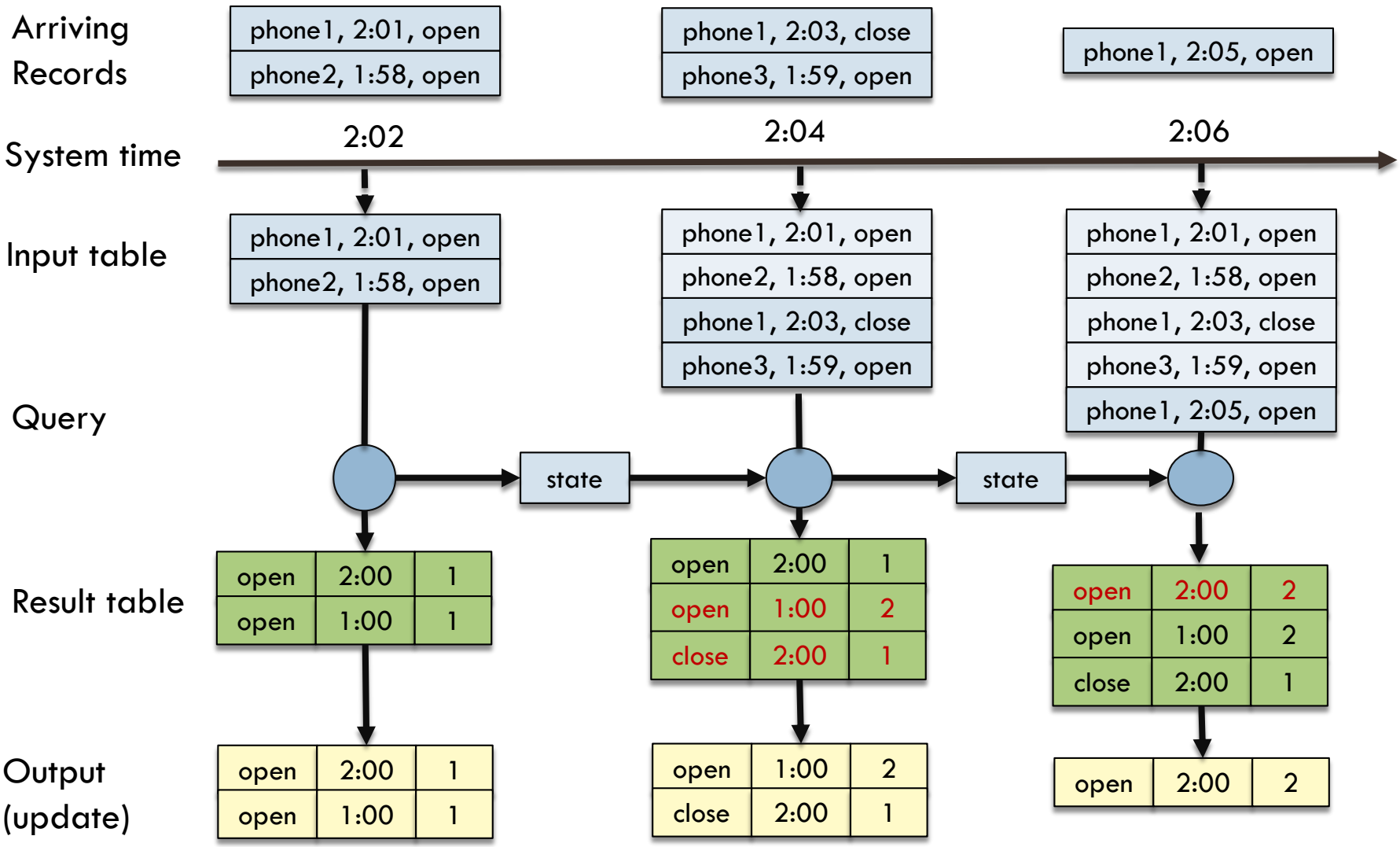



```
// schema (phone id, time, action)

val inputDF = spark.readStream.json("<path>")

val countDF = inputDF.groupBy($"action",
                window($"time", "1 hour")).count()
```

Structured Streaming Concepts



Structured Streaming Concepts

- Fault Recovery & Storage System Requirements
  - Results must still be valid even if machine failed
  - Input sources must be replayable
    - Amazon Kinesis, Apache Kafka, File system
  - Output sinks must support transactional updates
    - Idempotent
  - Spark will manage its internal storage
    - On a reliable storage system
    - S3 or HDFS

Structured Streaming Concepts

Output Modes

Mode	Description
Complete	The entire updated result table will be written to external storage. (Monitoring)
Append	Only the new rows appended in the result table since the last trigger. Rows will be output only once. (ETL)
Update	Only updated rows in the result table since the last trigger.. Applicable for output sinks that can be updated in place, such as RDBMS table. (Alerting)

Structured Streaming API

API built on top of Spark SQL engine

Batch ETL

```
val input = spark.read.json("<path>")

val result = input.select("phoneId", "action").where($"action" === "open")

result.write.format("parquet").save("<path>")
```

Streaming

```
val input = spark.readStream.json("<path>")

val result = input.select("phoneId", "action").where($"action" === "open")

result.writeStream.format("parquet").option("path","<path>")
```

Structured Streaming API

Word Count Example

```
import org.apache.spark.sql.functions._
import org.apache.spark.sql.SparkSession
import spark.implicits._

val lines = spark.readStream.format("socket")
       .option("host", "localhost").option("port", 9999).load()

// Split the lines into words
val words = lines.as[String].flatMap(_.split(" "))

// Generate running word count
val wordCounts = words.groupBy("value").count()

val query = wordCounts.writeStream.outputMode("complete")
            .format("console").start()
```

Structured Streaming API

- Data Sources
  - File – txt, json, csv, parquet
  - Kafka
  - Socket source – for testing purpose
- Output Sinks
  - File – store output to a directory
  - Console – for debugging
    - Print output every time there is a trigger
  - Memory – for debugging
    - In memory table for low volume data

Structured Streaming API


```
// socket
val dataDF = spark.readStream.format(“socket”).option(“host”,
“localhost”).option(“port”, 8888).load()

val dataSchema = new StructType().add(“id”,StringType).add(“time”,
TimestampType).add(“action”, StringType)

// file
val dataDF = spark.readStream.schema(dataSchema).json(”<path>”)

val dataDF = spark.readStream.schema(dataSchema )
       .option(“maxFilesPerTrigger”, 1).json(”<path>”)

// is DF streaming?
dataDF.isStreaming

// File sink
writeStream.format(“parquet”).start()
// Console sink
writeStream.format(“console”).start()
```

Structured Streaming API

- Selection, Project, Aggregation
  - Most common operations are supported for streaming
  - select, map, filter, flatMap
- Window Operations on Event Time

```
// schema {timestamp:Timestamp, word:String}
val wordsDF = ..

// aggregation over sliding window
val windowCount = dataDF.groupBy(
        window($"timestamp", "10 minutes", "5 minutes"),
         $"word").count()
```

Structured Streaming API

- Joining streams with static data
  - Very easy and without having to think

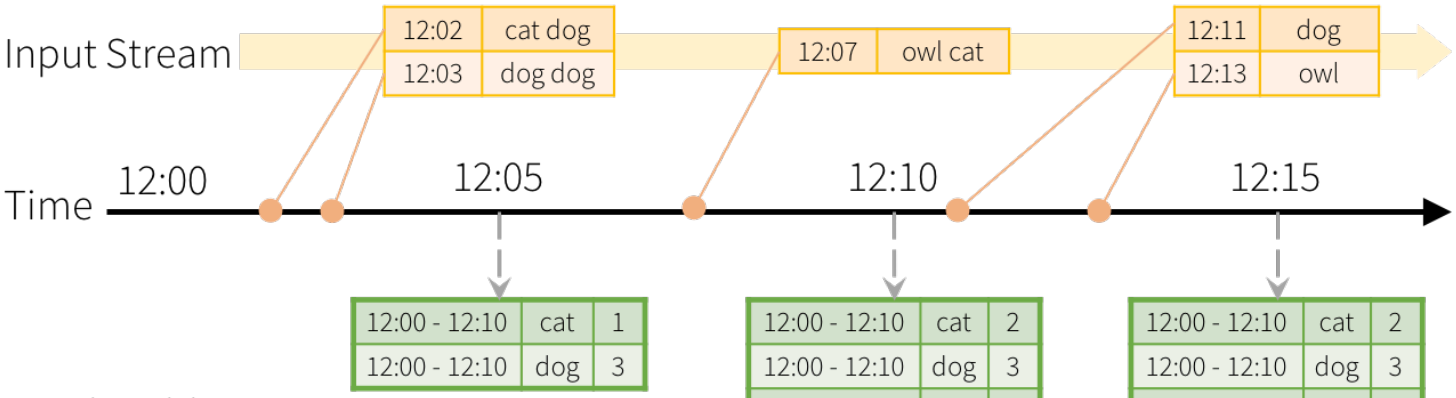
```
val pageViewEvents = spark.readStream.format("kafka").load()

val userInfo = spark.read.parquet("<path>")

val pageViewWithInfo = pageViewEvents.join(userInfo, "userId")
```

Structured Streaming API

Aggregation Over Sliding Window



Result Tables
after 5 minute triggers

counts incremented for windows
12:00 - 12:10 and 12:05 - 12:15

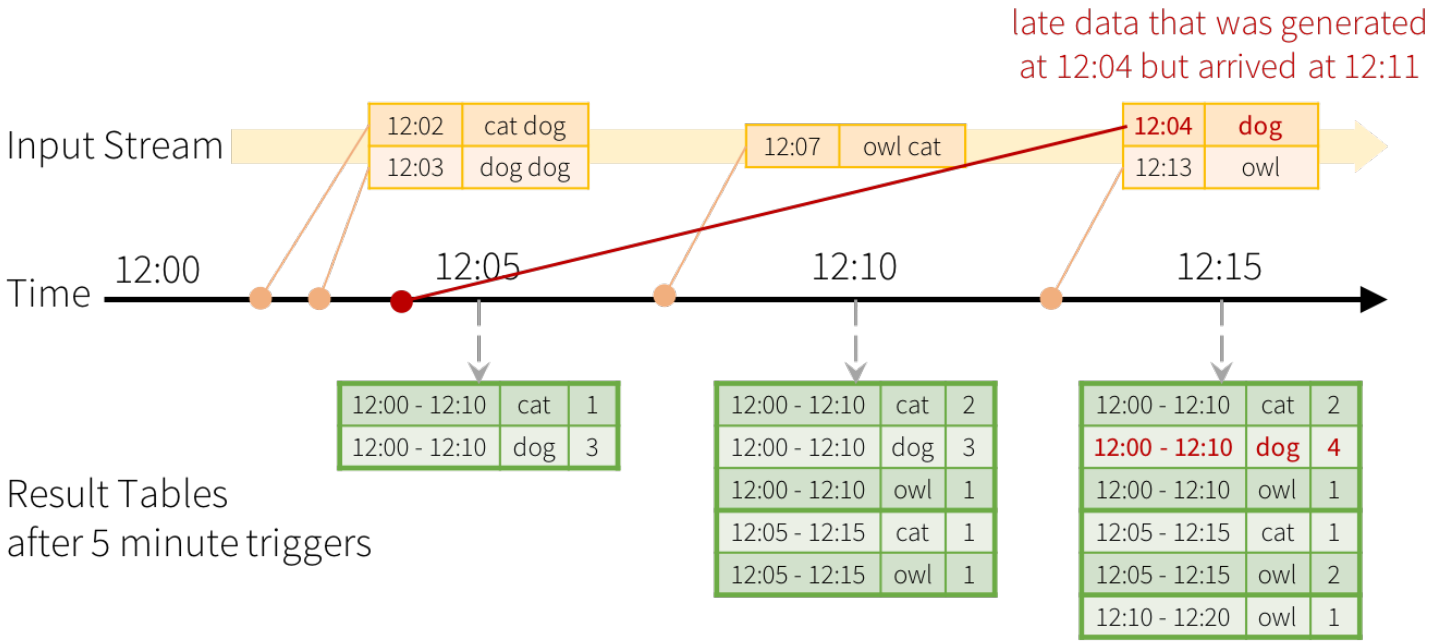
counts incremented for windows
12:05 - 12:15 and 12:10 - 12:20

Windowed Grouped Aggregation
with 10 min windows, sliding every 5 mins

http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html

Structured Streaming API

Handling Late Data & Watermarking



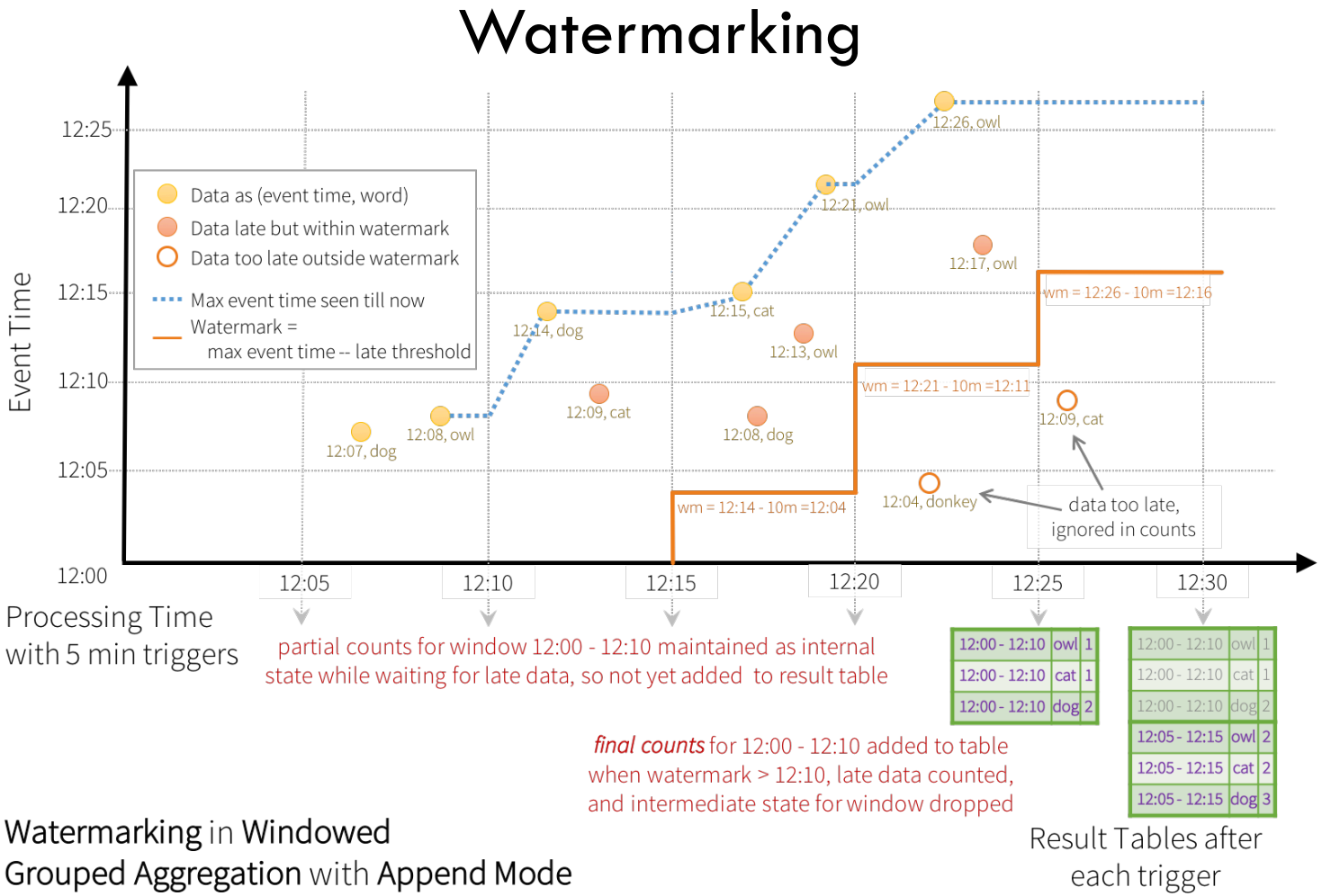
Late data handling in
Windowed Grouped Aggregation

Structured Streaming API

Watermarking

- A moving threshold trailing behind the max seen event time
- A gap defines how late the data is expected to be
- Data older than the watermark will be ignored
- State older than watermark is removed to reduce intermediate state size
- Of course useful only for stateful operation

Structured Streaming API



Watermarking in **Windowed Grouped Aggregation** with **Append Mode**

https://spark.apache.org/docs/2.3.0/structured-streaming-programming-guide.html#handling-late-data-and-watermarking

Structured Streaming API

- Watermarking example
  - Add call to watermark API before calling groupBy
  - Watermark event-time column must be the same as the one groupBy

```
// schema {timestamp:Timestamp, action:String}
val actiongLogDF = ..

// aggregation over sliding window
val countDF = actiongLogDF
       .watermark("timestamp", "15 minutes")
       .groupBy(
       window($"timestamp", "10 minutes", "5 minutes"),
        $"word").count()
```

Structured Streaming API

- Arbitrary Stateful Processing
  - User defined processing logic on each group of data
  - Call once per group in batch
  - mapGroupsWithState
  - flatmapGroupsWithState
  - Support in Scala and Java only
  - Need to set time out in order to reduce amount of state to maintain

Working with Streaming Queries

- Expose the result table
  - To interactive queries via Spark's JDBC server
  - Spark 2.0 – just rudimentary "memory" output sink

```
// schema {timestamp:Timestamp, action:String}
val actiongLogDF = ..

// aggregation over sliding window
val countDF = actiongLogDF.groupBy(
        window($"timestamp", "10 minutes", "5 minutes"),
         $"word").count()

countDF.writeStream.format("memory").queryName("count")
                           .outputMode("complete").start()

sql("select sum(count) from count where action='login'")
```

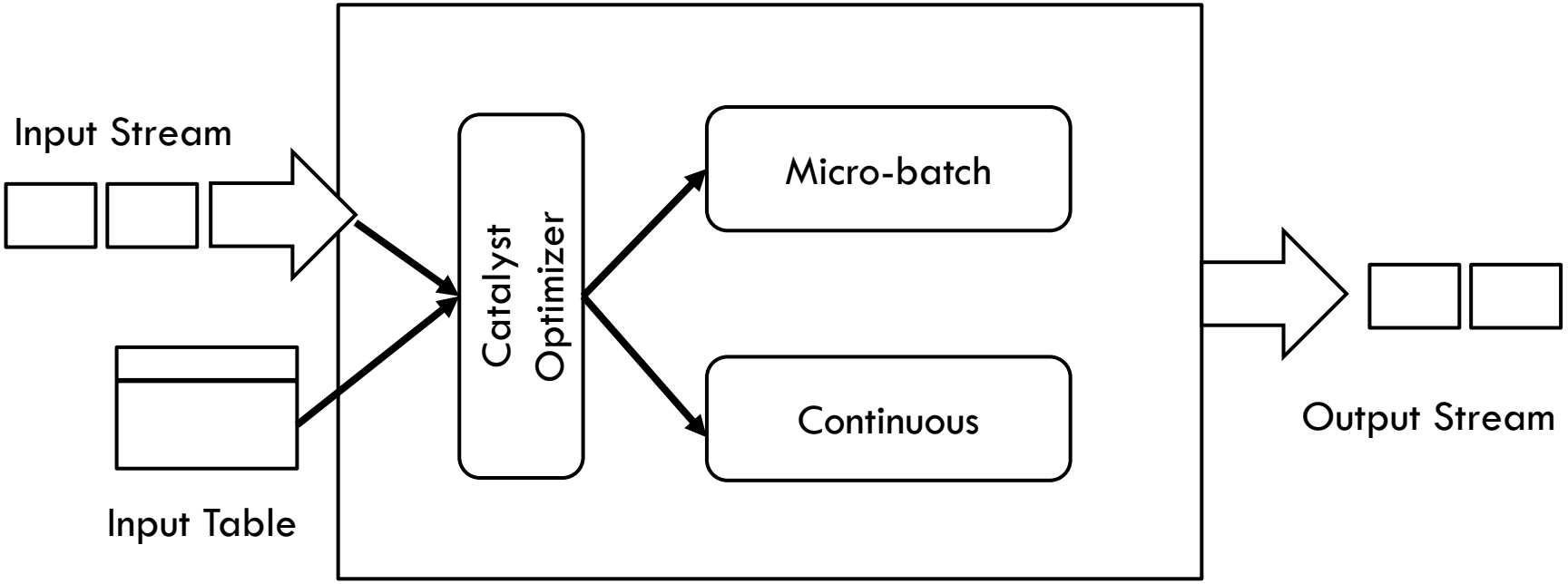
Working with Streaming Queries

Triggers

Type	Description
unspecified (default)	Micro-batch mode. Trigger as soon as the next batch has completed
Fixed-interval	Micro-batch mode. Trigger based on the specified interval. For example – every 5 seconds.
One-time	Trigger only once, and then stop. Good for low volume data stream that need to process a few times a day
Continuous	(Experimental 2.3) Continuous processing of new incoming data. Low latency.

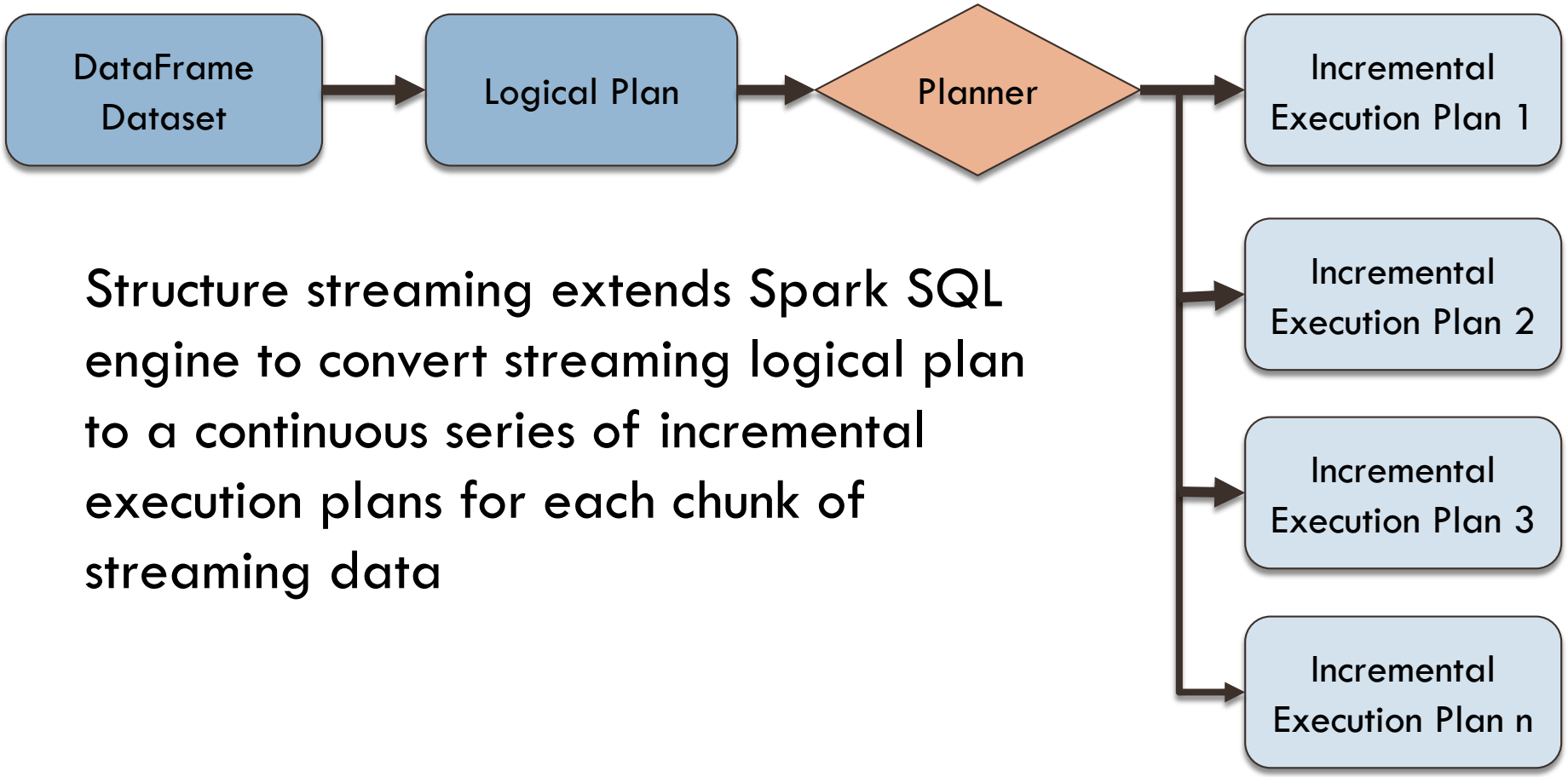
Working with Streaming Queries

- Continuous processing mode (Spark 2.3)
  - For low latency use cases - < 100ms
  - Real-time fraud detection
  - Long running tasks that process events as they come in



Execution Model

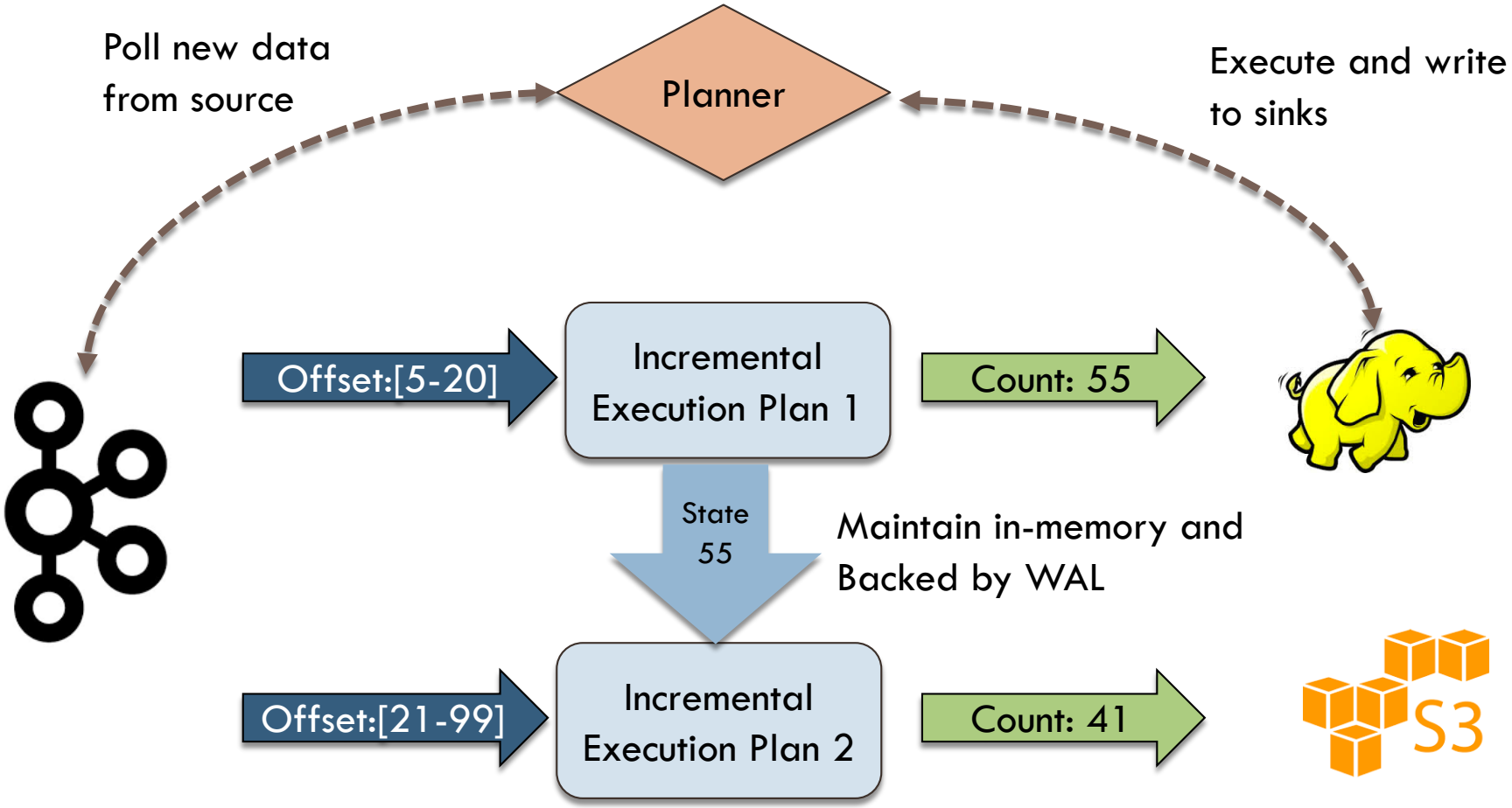
Continuous & Incremental Execution



Structure streaming extends Spark SQL engine to convert streaming logical plan to a continuous series of incremental execution plans for each chunk of streaming data

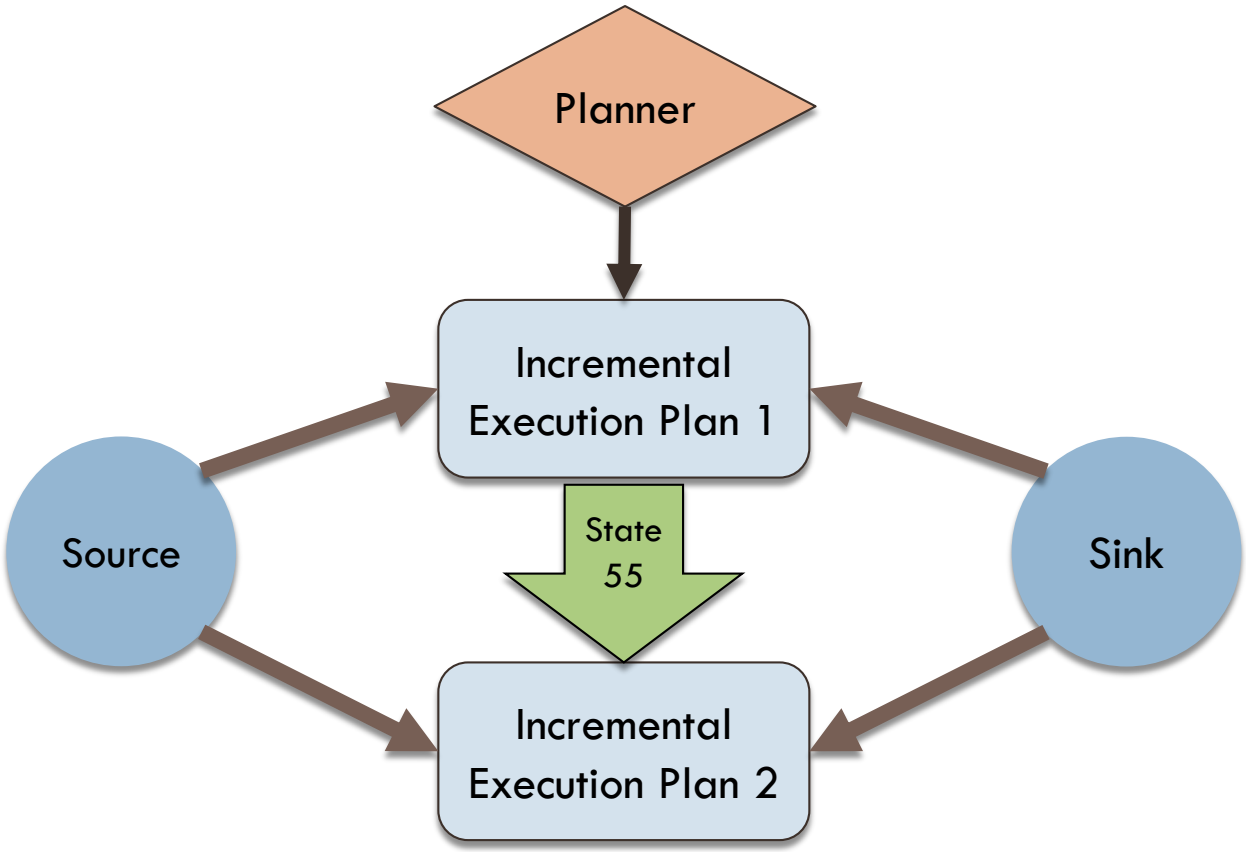
Execution Model

Continuous & Incremental Execution



Fault Tolerance

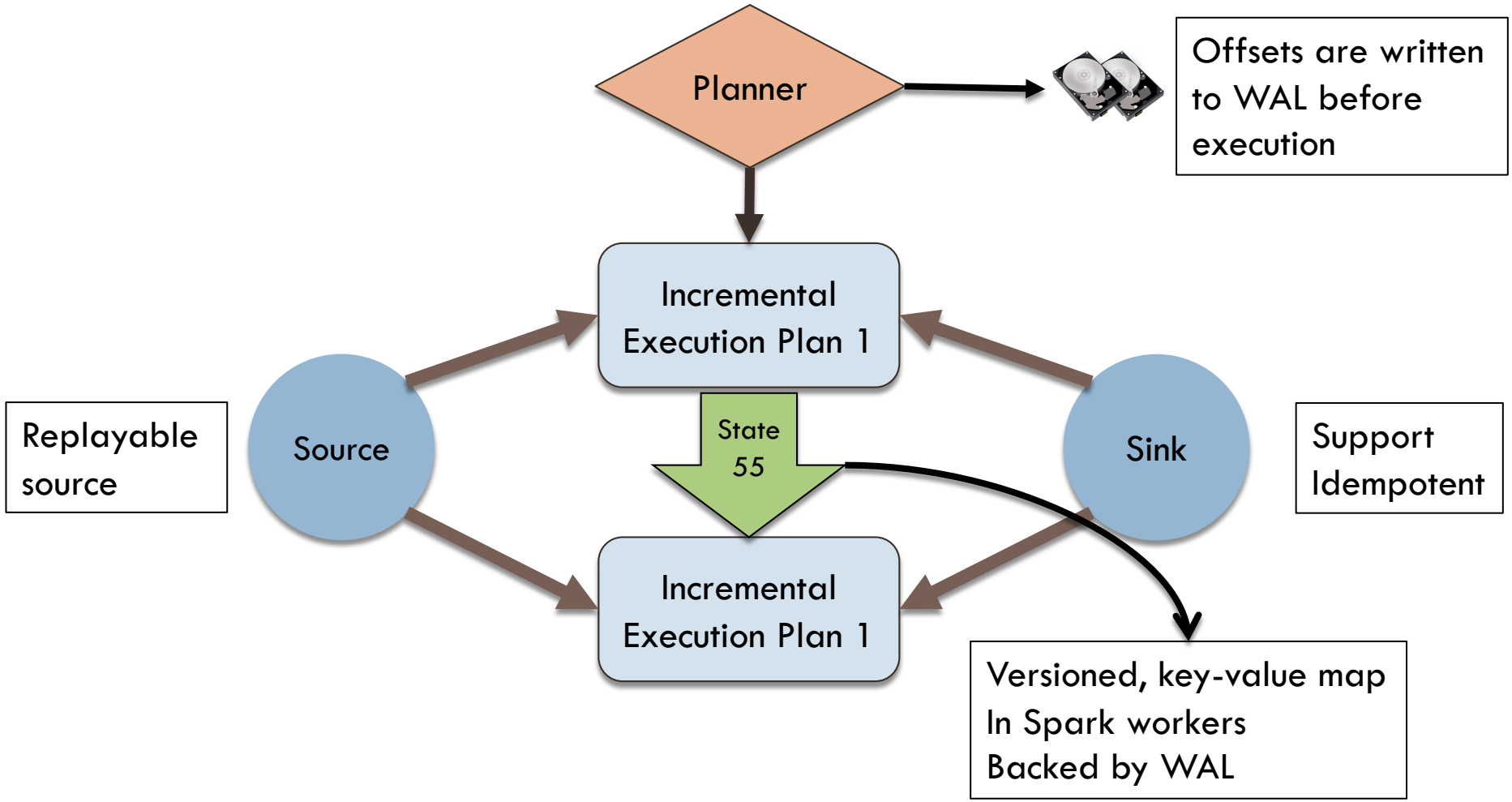
End-to-End Exactly Once Guarantees



All data & metadata must be recoverable & replayable

Fault Tolerance

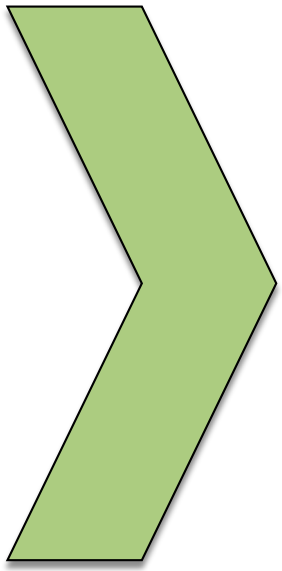
End-to-End Exactly Once Guarantees

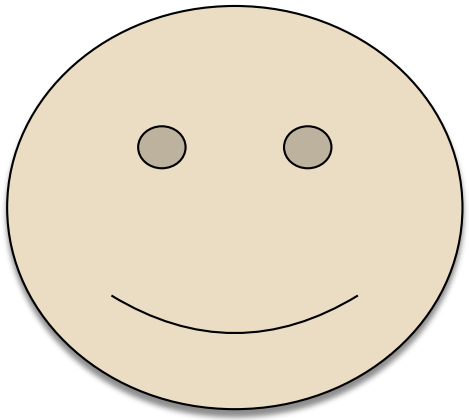


Fault Tolerance

End-to-End Exactly Once Guarantees

- Offset tracking in WAL
- State Management
- Replayable sources
- Idempotent sinks





Happy Customers

Spark vs Others

Comparing With Other Streaming Engines

Property	Structured Streaming	Spark Streaming	Apache Storm	Apache Flink	Kafka Streams	Google Dataflow
Streaming API	incrementalize batch queries	integrates with batch	separate from batch	separate from batch	separate from batch	integrates with batch
Prefix Integrity Guarantee	✔	✔	✘	✘	✘	✘
Internal Processing	exactly once	exactly once	at least once	exactly once	at least once	exactly once
Transactional Sources/Sinks	✔	some	some	some	✘	✘
Interactive Queries	✔	✔	✘	✘	✘	✘
Joins with Static Data	✔	✔	✘	✘	✘	✘

Resources

- Programing guide
  - http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html
- Blogs
  - https://databricks.com/blog/2016/07/28/structured-streaming-in-apache-spark.html
  - https://databricks.com/blog/2017/01/19/real-time-streaming-etl-structured-streaming-apache-spark-2-1.html