Spark Streaming

Prof. Dr. Stephan Trahasch
Hochschule Offenburg

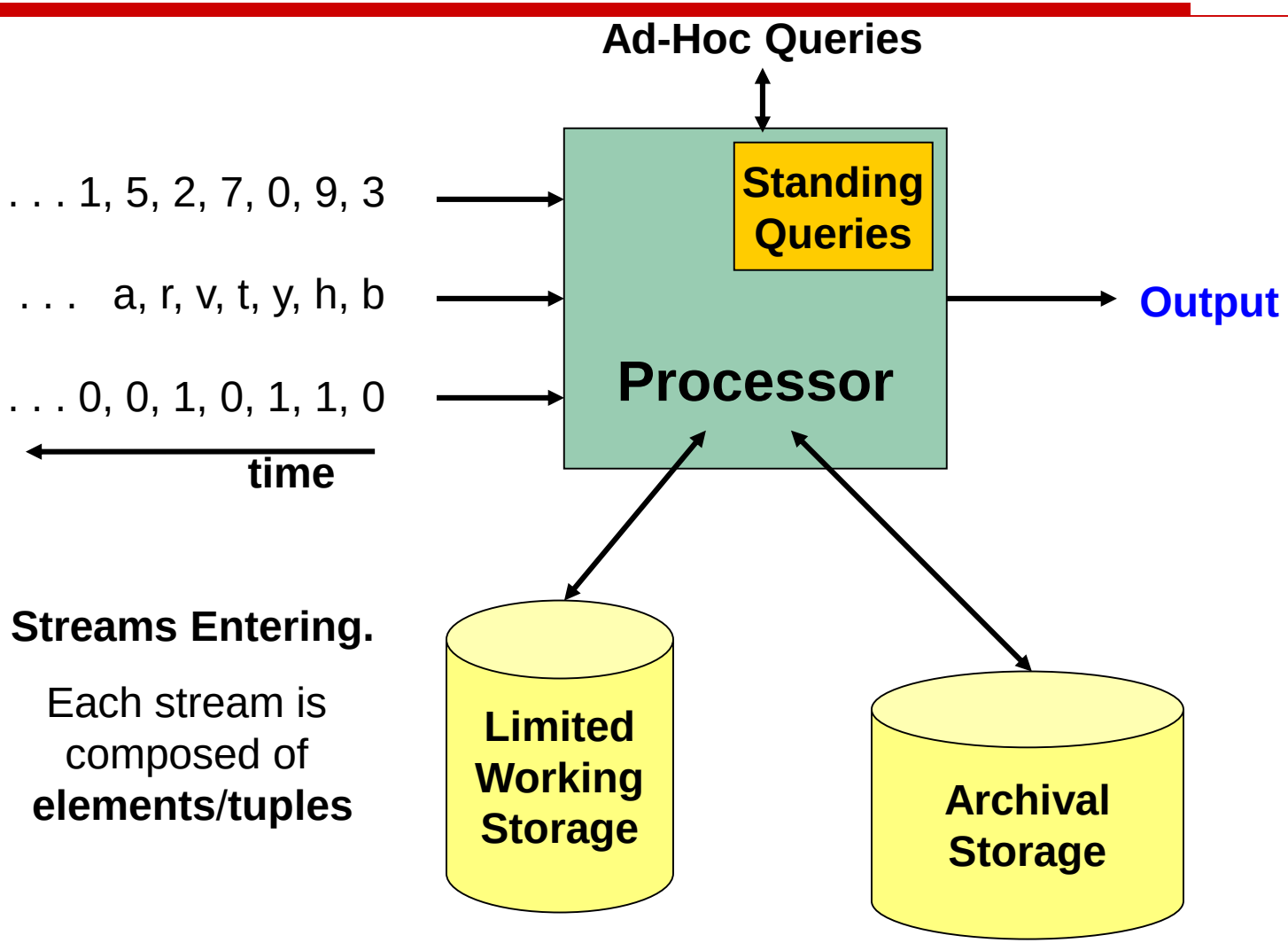
Outline

- Streaming
- Spark Streaming

What is a data stream?

- Input elements enter at a rapid rate, at one or more input ports
  - Structured (e.g., tuples)
  - Ordered (implicitly or timestamped)
  - Arriving continuously at high volumes
  - Not possible to store entirely
  - Sometimes not possible to even examine all items
- We call elements of the stream tuples
- The system cannot store the entire stream accessibly

How do you make critical calculations about the stream using a limited amount of (secondary) memory?



Streams Entering.

Each stream is composed of **elements/tuples**

What to do with data streams?

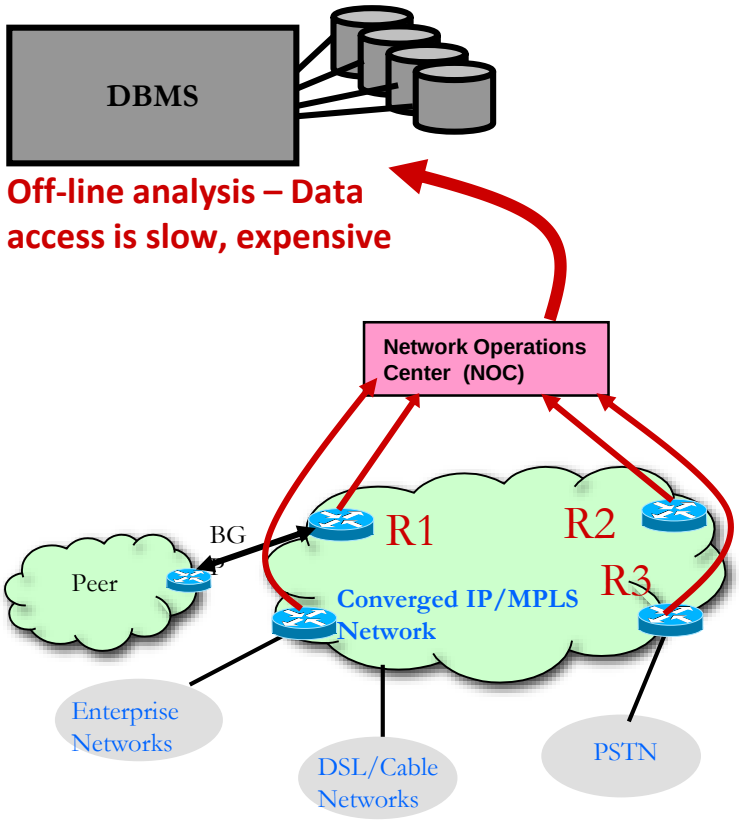
- Network traffic monitoring
- Datacenter telemetry monitoring
- Sensor networks monitoring
- Credit card fraud detection
- Stock market analysis
- Online mining of click streams
- Monitoring social media streams

What's the scale? Packet data streams

- Single 2 Gb/sec link; say avg. packet size is 50 bytes
  - Number of packets/sec = 5 million
  - Time per packet = 0.2 microseconds
- If we only capture header information per packet: source/destination IP, time, no. of bytes, etc. – at least 10 bytes
  - 50 MB per second
  - 4+ TB per day
  - **Per link!**

What if you wanted to do deep-packet inspection?

Source: Minos Garofalakis, Berkeley CS 286



What are the top (most frequent) 1000 (source, dest) pairs seen by R1 over the last month?

How many distinct (source, dest) pairs have been seen by both R1 and R2 but not R3?

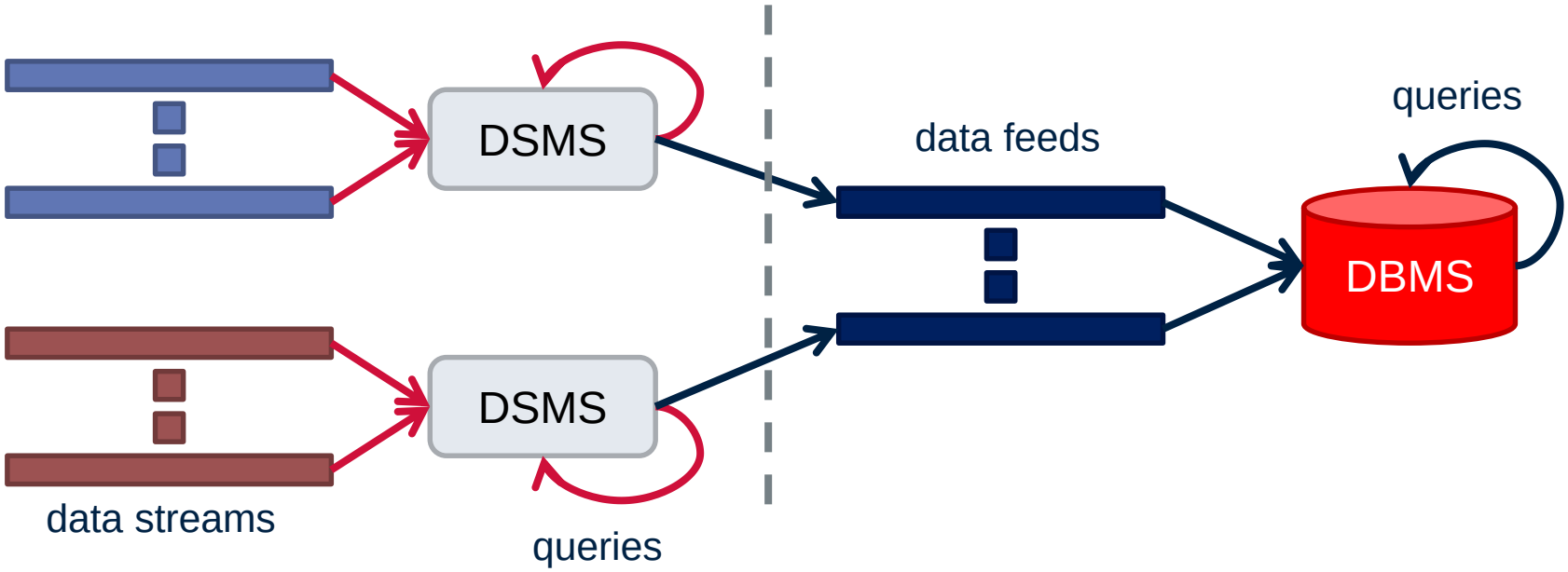
Set-Expression Query

```
SELECT COUNT (R1.source, R1.dest)
FROM  R1, R2
WHERE R1.source = R2.source
```

SQL Join Query

Source: Minos Garofalakis, Berkeley CS 286

Common Architecture



- Data stream management system (DSMS) at observation points
  - Voluminous streams-in, reduced streams-out
- Database management system (DBMS)
  - Outputs of DSMS can be treated as data feeds to databases

Source: Peter Bonz

DBMS vs. DSMS

DSMS

- Model: (mostly) transient relations
- Relation: tuple sequence
- Data update: appends
- Query: persistent
- Query answer: approximate
- Query evaluation: one pass
- Query plan: adaptive

DBMS

- Model: persistent relations
- Relation: tuple set/bag
- Data update: modifications
- Query: transient
- Query answer: exact
- Query evaluation: arbitrary
- Query plan: fixed

Source: Peter Bonz

What makes it hard?

- Intrinsic challenges:
  - Volume
  - Velocity
  - Limited storage
  - Strict latency requirements
  - Out-of-order delivery
- System challenges:
  - Load balancing
  - Unreliable message delivery
  - Fault-tolerance
  - Consistency semantics (lossy, exactly once, at least once, etc.)

What exactly do you do?

- "Standard" relational operations:
  - Select
  - Project
  - Transform (i.e., apply custom UDF)
  - Group by
  - Join
  - Aggregations

Issues of Semantics

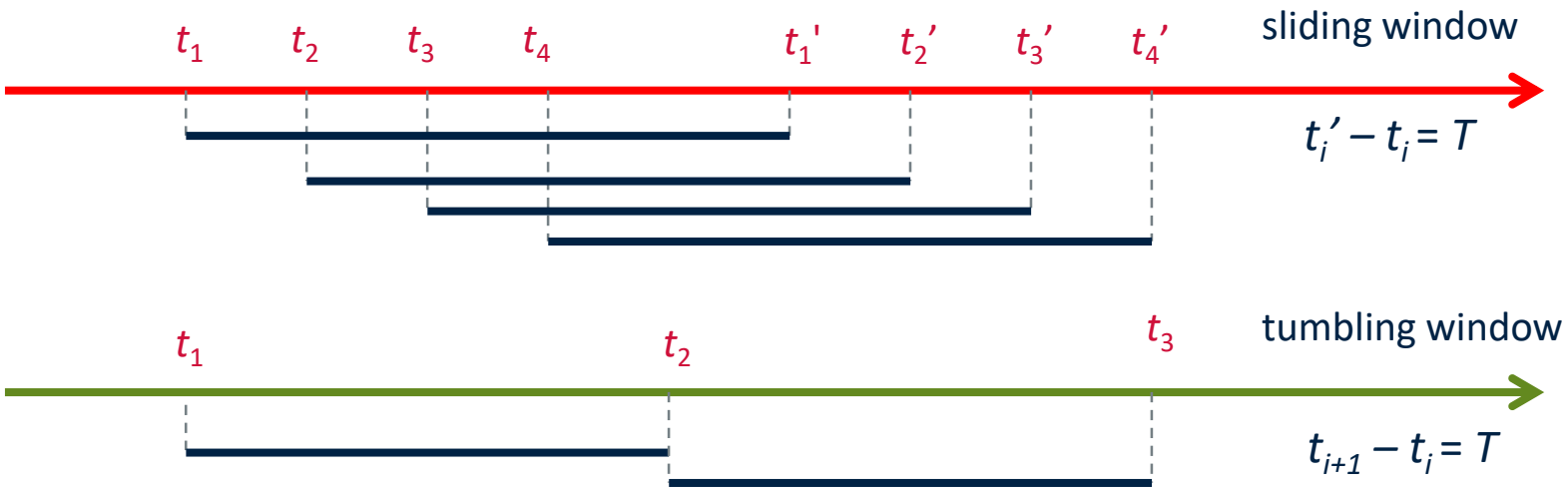
- Group by… aggregate
  - When do you stop grouping and start aggregating?
- Joining a stream and a static source
  - Simple lookup
- Joining two streams
  - How long do you wait for the join key in the other stream?
- Joining two streams, group by and aggregation
  - When do you stop joining?

Possible Solution: Windows

- Mechanism for extracting finite relations from an infinite stream
- Windows restrict processing scope:
  - Windows based on ordering attributes (e.g., time)
  - Windows based on item (record) counts
  - Windows based on explicit markers (e.g., punctuations)
  - Variants (e.g., some semantic partitioning constraint)

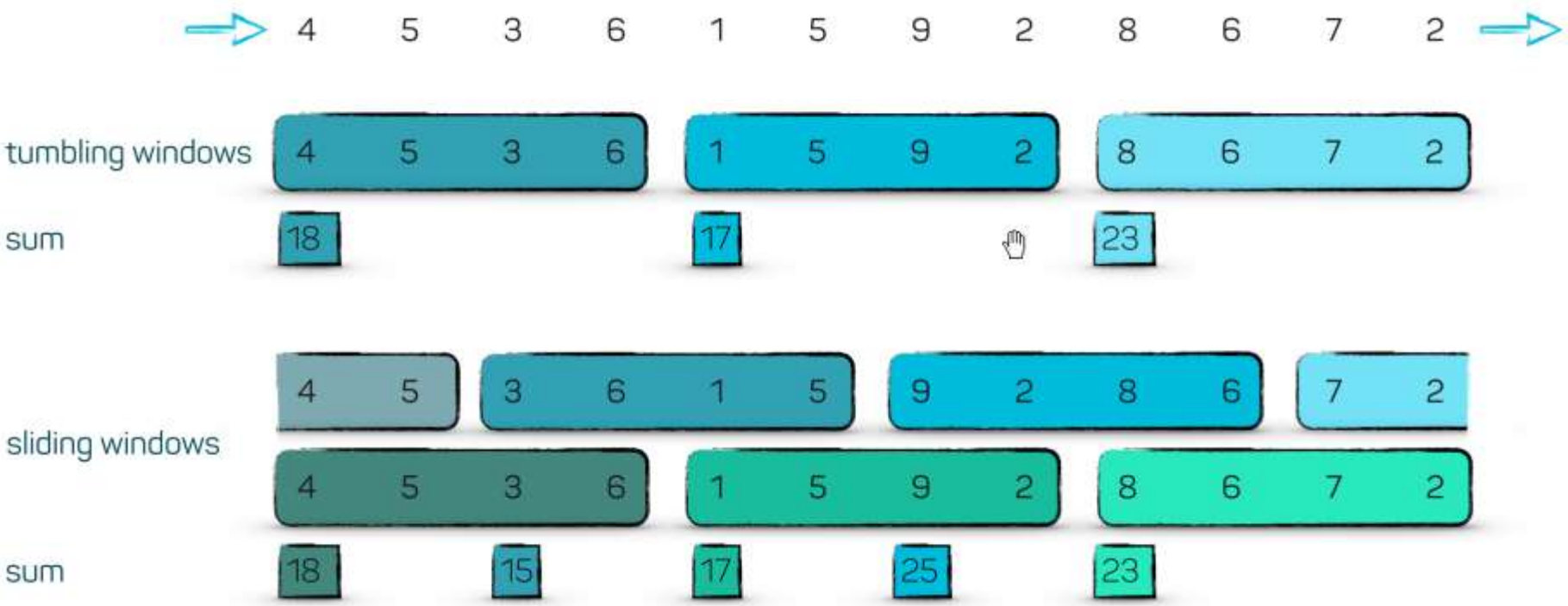
Windows on Ordering Attributes

- Assumes the existence of an attribute that defines the order of stream elements (e.g., time)
- Let *T* be the window size in units of the ordering attribute



Source: Peter Bonz

Tumbling & Sliding Windows



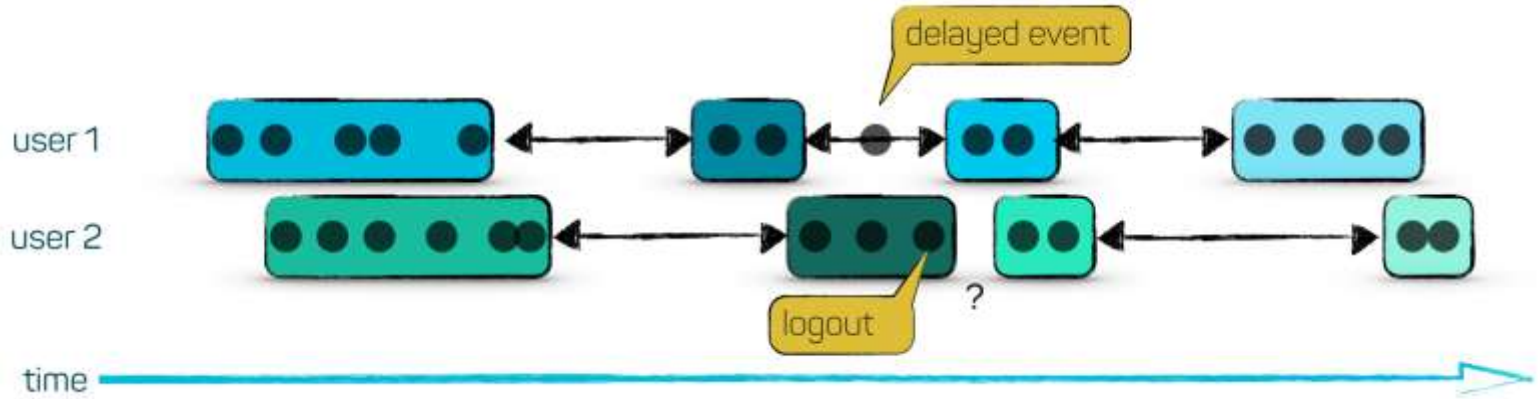
Windows on Counts

- Window of size *N* elements (sliding, tumbling) over the stream
- Challenges:
  - Problematic with non-unique timestamps: non-deterministic output
  - Unpredictable window size (and storage requirements)

Source: Peter Bonz

Windows from “Punctuations”

- Application-inserted “end-of-processing”
  - Example: stream of actions… “end of user session”
- Properties
  - Advantage: application-controlled semantics
  - Disadvantage: unpredictable window size (too large or too small)



The Dataflow Model:

A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing

[...] stop trying to groom unbounded datasets into finite pools of information that eventually become complete, and instead live and breathe under the assumption that we will never know if or when we have seen all of our data, only that new data will arrive, old data may be retracted [...]

http://www.vldb.org/pvldb/vol8/p1792-Akidau.pdf

Outline

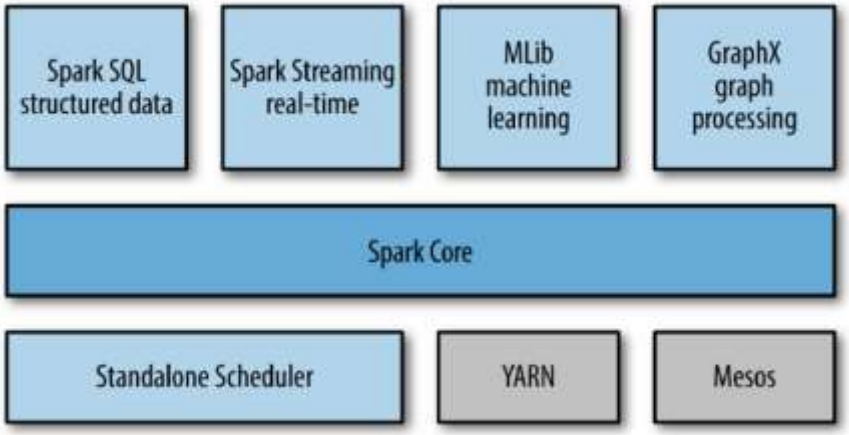
- Streaming
- Spark Streaming

Spark Streaming

Stream Processing with Structured Streaming

- Scalable
- Fault-tolerant
- “Easy”

→ Building continuous applications



COMPLEX DATA

Diverse data formats (json, avro, binary, …)

Data can be dirty, late, out-of-order

COMPLEX WORKLOADS

Combining streaming with interactive queries

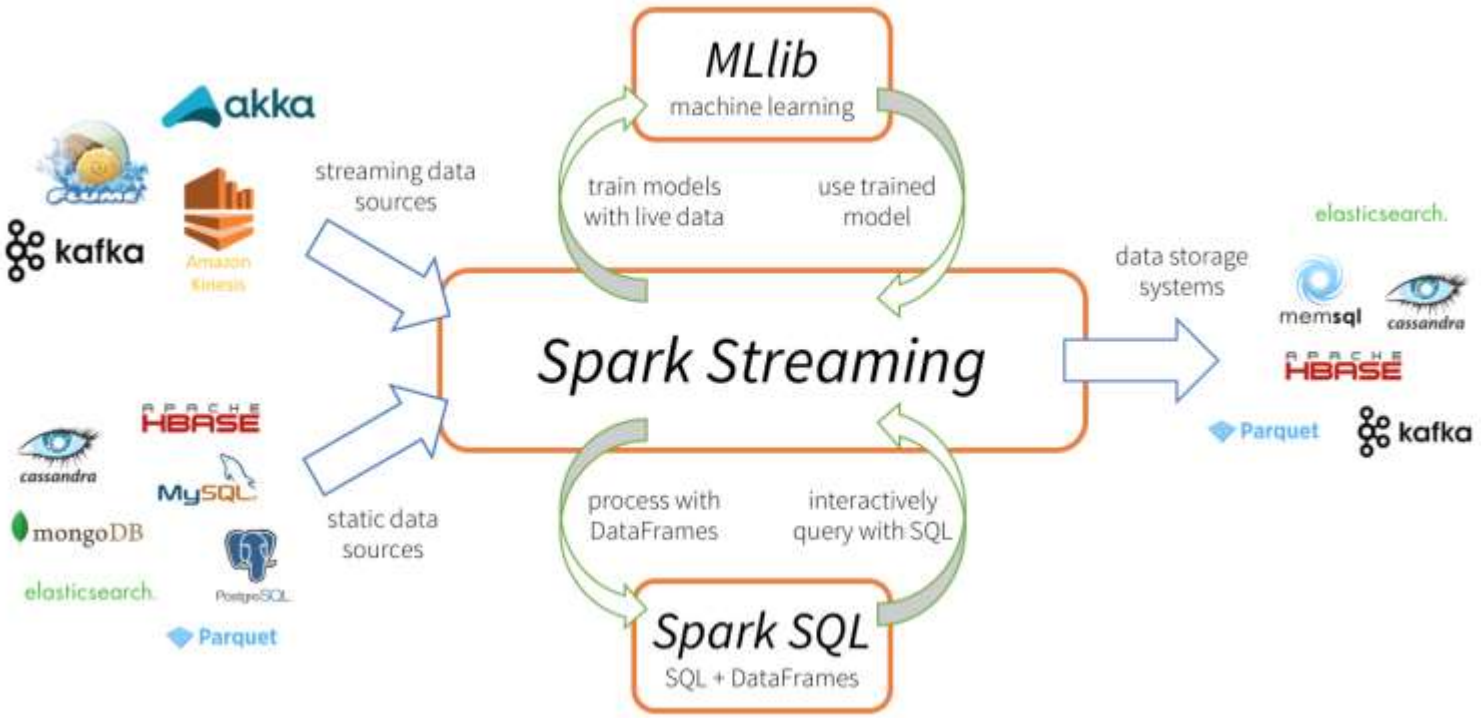
Machine learning

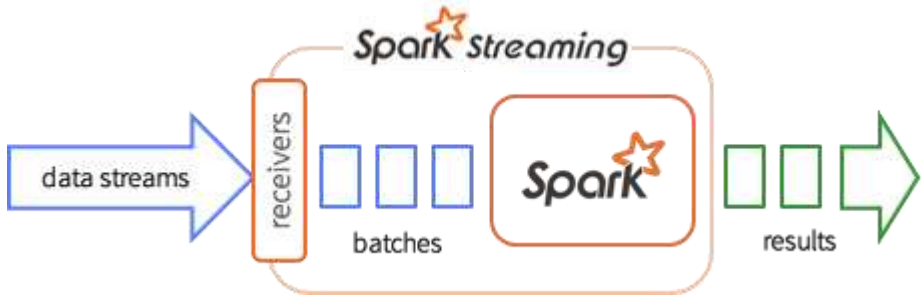
COMPLEX SYSTEMS

Diverse storage systems (Kafka, S3, Kinesis, RDBMS, …)

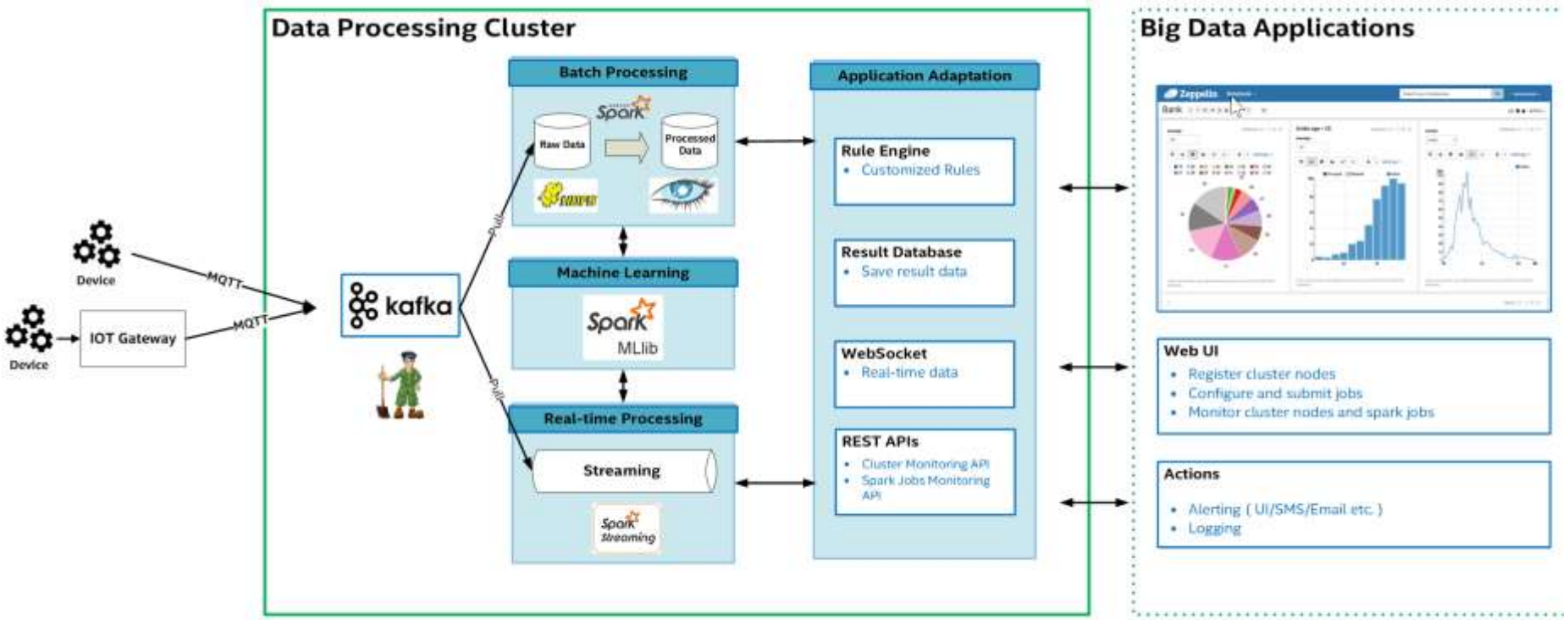
System failures

Spark Streaming: Micro Batches



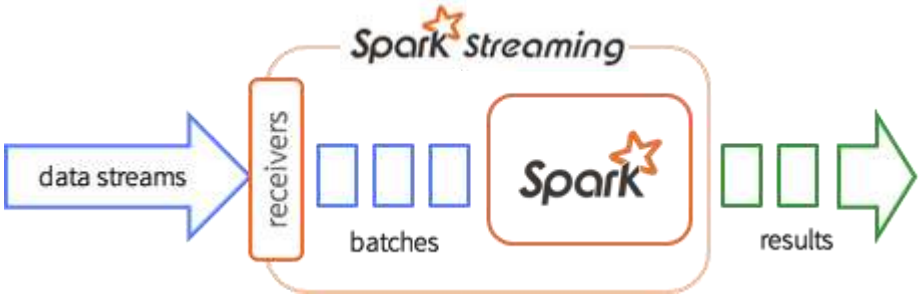


Example


Data Processing Cluster
Batch Processing
Spark
Raw Data
Processed Data
Machine Learning
Spark
MLlib
Real-time Processing
Streaming
Spark Streaming
kafka
Pull
Pull
Device
Device
IOT Gateway
MQTT
MQTT
Application Adaptation
Rule Engine
Customized Rules
Result Database
Save result data
WebSocket
Real-time data
REST APIs
Cluster Monitoring API
Spark Jobs Monitoring API
Big Data Applications
Web UI
Register cluster nodes
Configure and submit jobs
Monitor cluster nodes and spark jobs
Actions
Alerting (UI/SMS/Email etc.)
Logging

Apache Spark Streaming

- Microbatching
- Similiar, partly unified, programming model as with batch processing
- State and window operations
- Missing support for event time
- DStream and Structured Streaming



Spark Structured Streaming

- Incrementally and continuously updating the final result
- Handling Event-time and Late Data
- Delivering end-to-end exactly-once fault tolerance semantics
- Output modes: Append, Complete, Update
- Support Stateless and Stateful operations
- Dataset/DataFrame API in Scala, Java, Python or R
- Express streaming the same way as batch

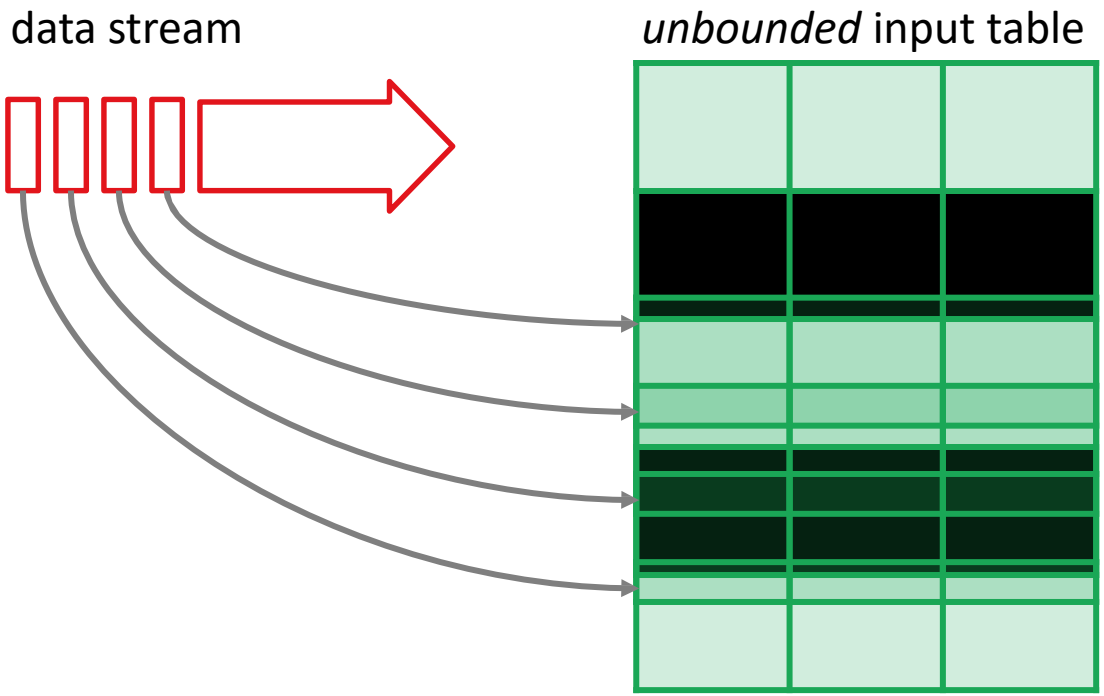
```
val ds = sparkSession
  .read
  .json("someFile.json")

val ds = sparkSession
  .readStream
  .format("kafka")
  .option("...","...")
  .load
```

```
ds
  .write
  .json("otherFile.json")

ds
  .writeStream
  .outputMode("complete")
  .format("console")
  .start()
```

Treat Streams as Unbounded Tables

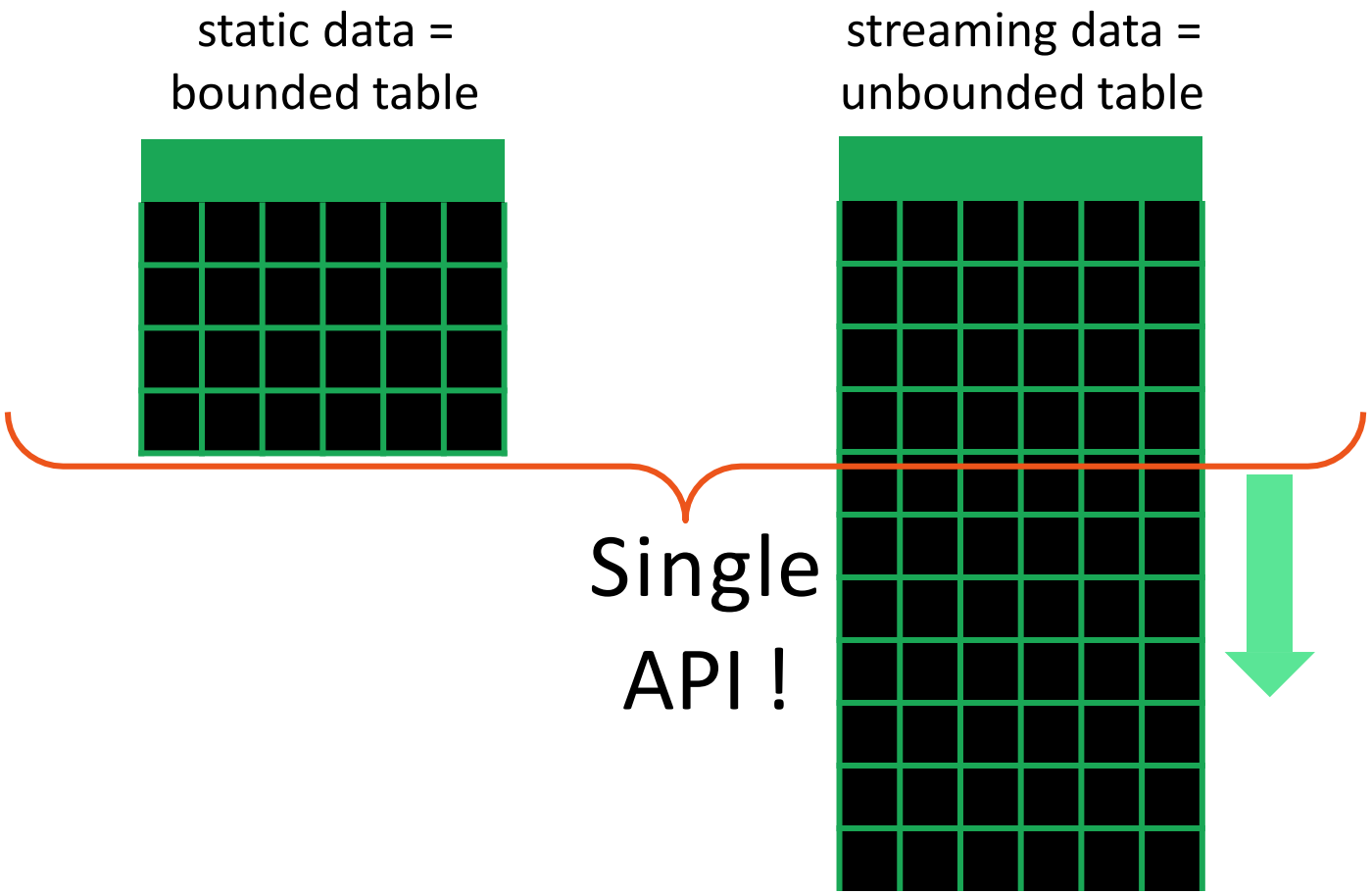


new data in the
data stream

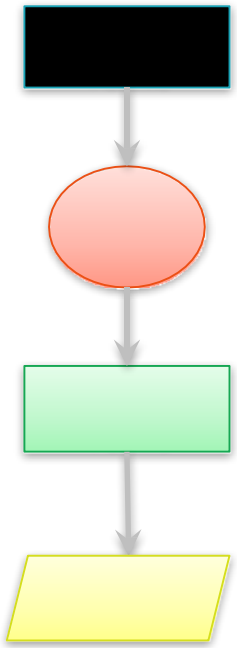
=

new rows appended
to a unbounded table

Single API for bounded and unbounded Table



Batch Queries with DataFrames



```
input = spark.read
        .format("json")
        .load("source-path")
```

Create input DF from JSon file

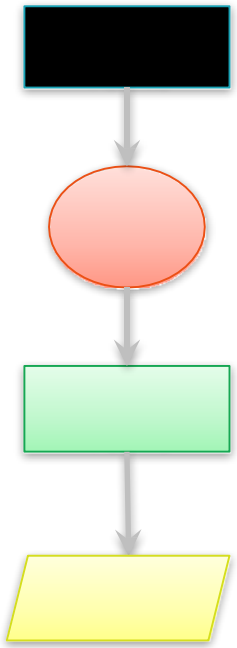
```
result = input
         .select("device", "signal")
         .where("signal > 15")
```

Create result DF by querying for some devices to create

```
result.write
         .format("parquet")
         .save("dest-path")
```

Output result to parquet file

Streaming Queries with DataFrames



```
input = spark.readStream
        .format("json")
        .load("source-path")
```

Create input DF from Kafa

Replace read with readStream

```
result = input
         .select("device", "signal")
         .where("signal > 15")
```

Select some devices

Query does not change

```
result.writeStream
         .format("parquet")
         .start("dest-path")
```

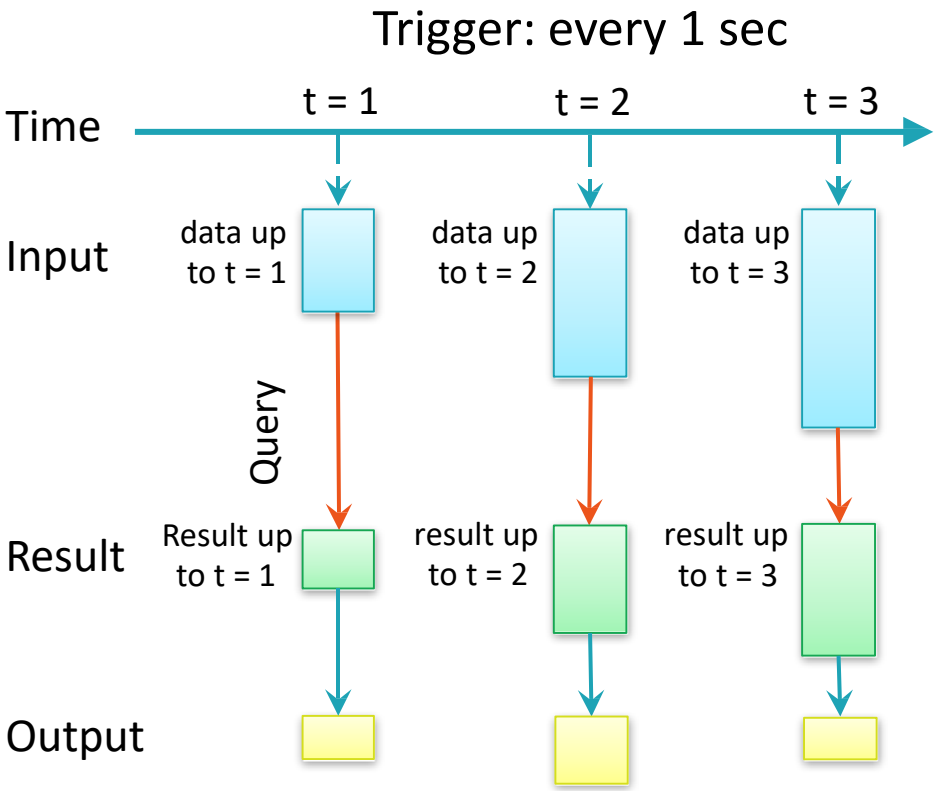
Write to Parquet file stream

Replace save() with start()

Conceptual Model

As the input table grows with new data, the result table changes

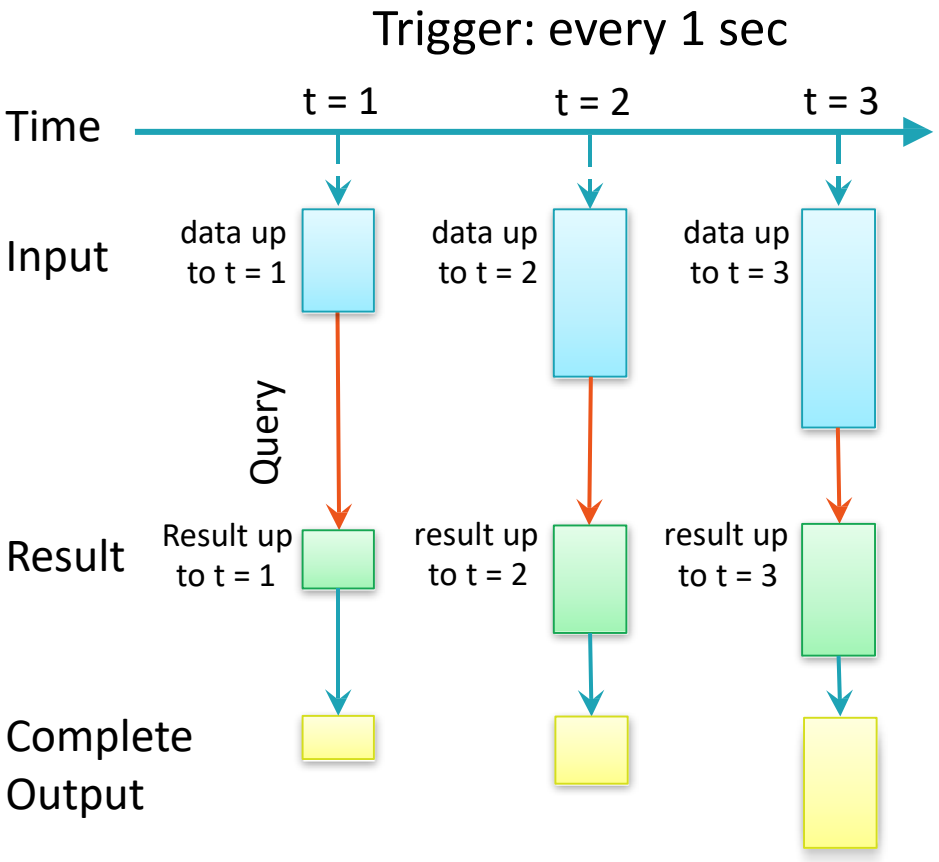
Every trigger interval, we can output the changes in the result



Conceptual Model

Output mode defines what changes to output

Complete mode outputs the entire result

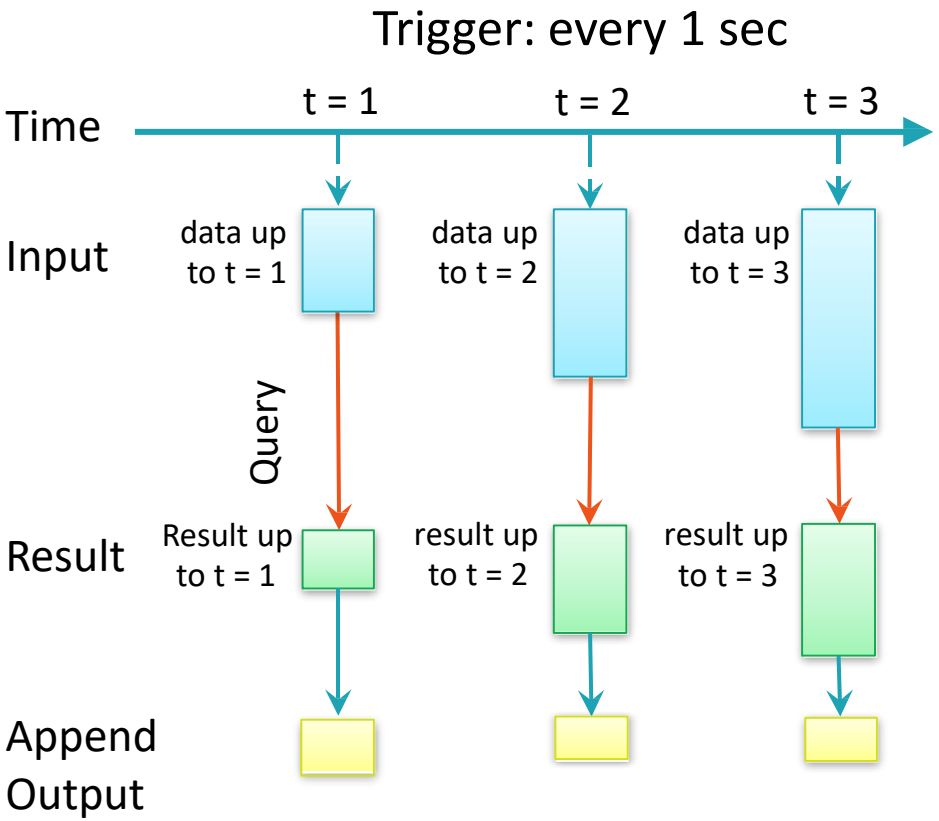


Conceptual Model

Output mode defines what changes to output

Append mode outputs new tuples only

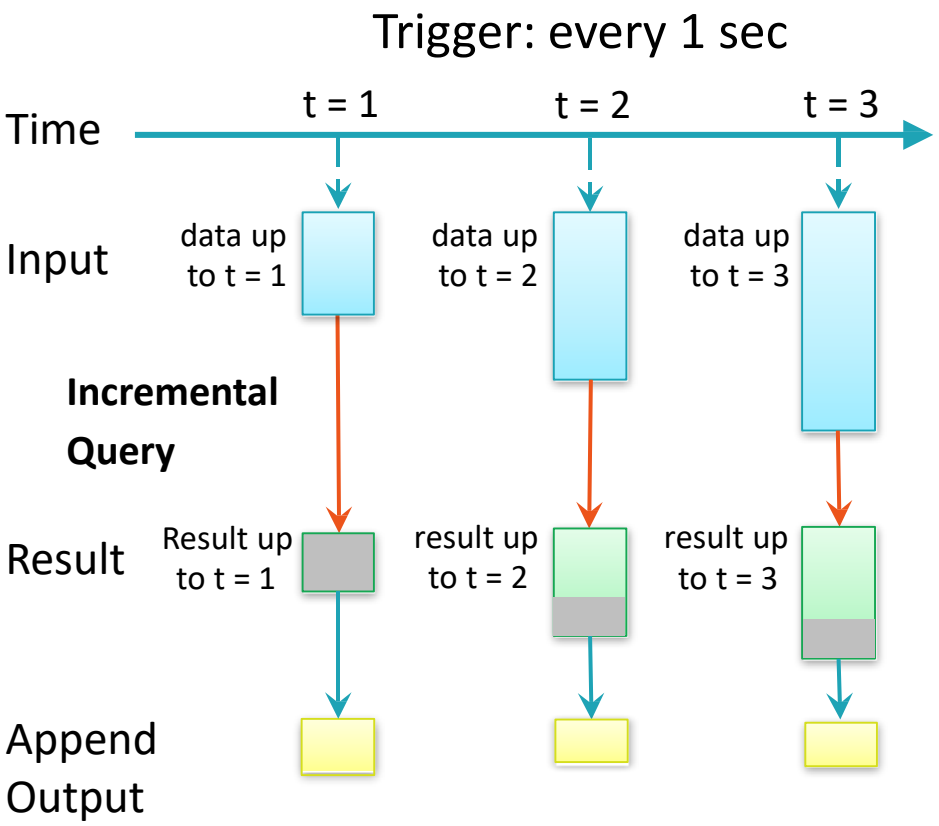
Update mode output tuples that have changed since the last trigger



Conceptual Model

Full input does not need to be processed every trigger

Engine converts query to an incremental query that operates only on new data to generate output



Anatomy of a Streaming Query

```
spark.readStream
  .format("kafka")
  .option("subscribe", "input")
  .load()
```

Source

- Specify one or more locations to read data from
- Built in support for Files/Kafka/Socket, pluggable.
- Can include multiple sources of different types using union()

Anatomy of a Streaming Query

```
spark.readStream
  .format("kafka")
  .option("subscribe", "input")
  .load()
  .groupBy(value.cast("string") as key)
  .agg(count("*") as value)
```

Transformation

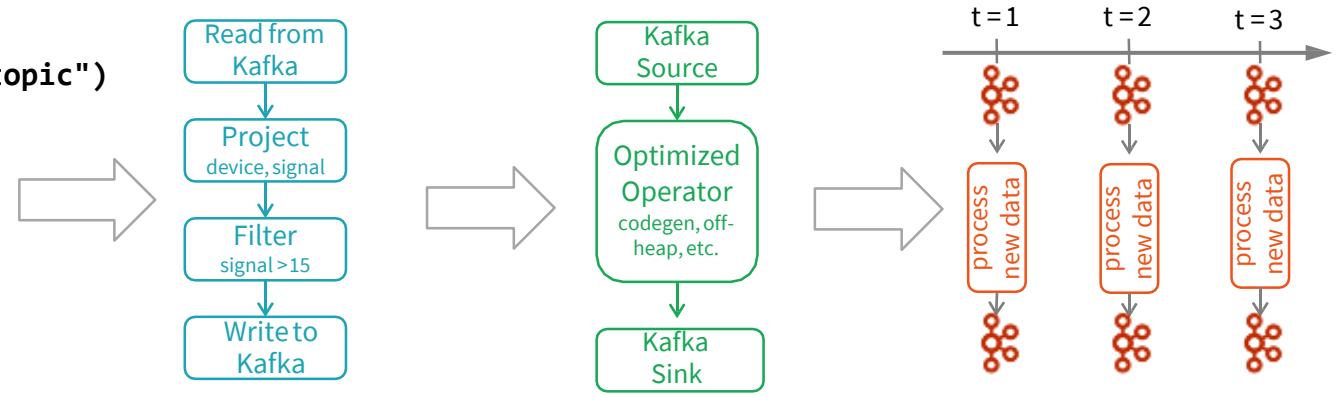
- Using DataFrames, Datasets and/or SQL.
- Catalyst figures out how to execute the transformation incrementally.
- Internal processing always exactly-once.

Spark automatically streamifies!

```
input = spark.readStream
  .format("kafka")
  .option("subscribe", "topic")
  .load()

result = input
  .select("device",
  "signal")
  .where("signal > 15")

result.writeStream
  .format("parquet")
  .start("dest-path")
```



DataFrames, Datasets, SQL

Logical Plan

Optimized Physical Plan

Series of Incremental Execution Plans

Spark SQL converts batch-like query to a series of incremental execution plans operating on new batches of data

Anatomy of a Streaming Query

```
spark.readStream
  .format("kafka")
  .option("subscribe", "input")
  .load()
  .groupBy(value.cast("string") as key)
  .agg(count("*") as value)
  .writeStream
  .format("kafka")
  .option("topic", "output")
```

Sink

- Accepts the output of each batch.
- When supported sinks are transactional and exactly once (Files)
- Use foreach to execute arbitrary code.

Anatomy of a Streaming Query

```
spark.readStream
  .format("kafka")
  .option("subscribe", "input")
  .load()
  .groupBy(value.cast("string") as key)
  .agg(count("*") as value)
  .writeStream
  .format("kafka")
  .option("topic", "output")
  .trigger("1 minute")
  .outputMode("update")
```

Output mode – What's output

- Complete – Output the whole answer every time
- Update – Output changed rows
- Append – Output new rows only

Trigger – When to output

- Specified as a time, eventually supports data size
- No trigger means as fast as possible

Anatomy of a Streaming Query

```
spark.readStream
  .format("kafka")
  .option("subscribe", "input")
  .load()
  .groupBy(value.cast("string") as key)
  .agg(count("*") as value)
  .writeStream
  .format("kafka")
  .option("topic", "output")
  .trigger("1 minute")
  .outputMode("update")
  .option("checkpointLocation", "…")
  .start()
```

Checkpoint

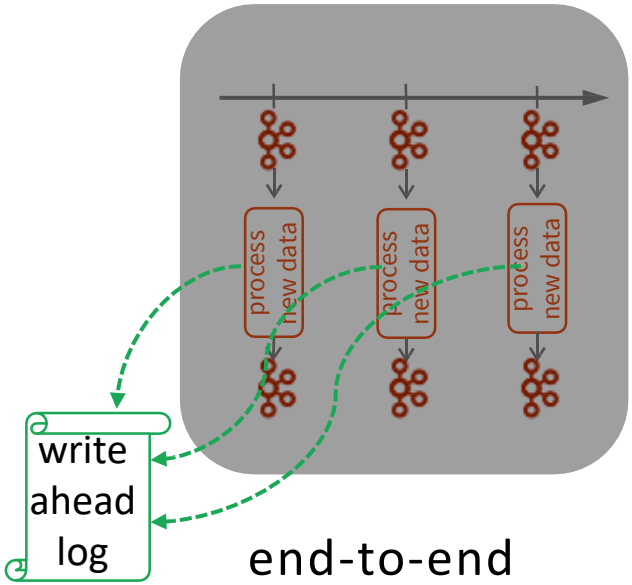
- Tracks the progress of a query in persistent storage
- Can be used to restart the query if there is a failure.

Fault-tolerance with Checkpointing

Checkpointing – tracks progress (offsets) of consuming data from the source and intermediate state.

Offsets and metadata saved as JSON

Can resume after changing your streaming transformations



end-to-end exactly-once guarantees

More Info

Structured Streaming Programming Guide

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