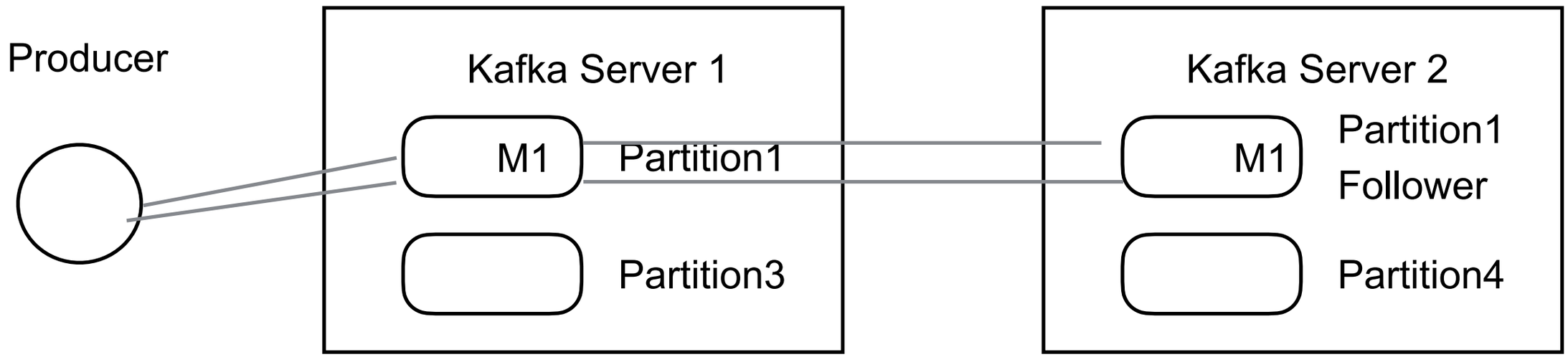
CS 696 Intro to Big Data: Tools and Methods
Spring Semester, 2020
Doc 21 Kafka, Spark Streaming
Apr 9, 2020

Copyright ©, All rights reserved. 2020 SDSU & Roger Whitney, 5500 Campanile Drive, San Diego, CA 92182-7700 USA. OpenContent (http://www.opencontent.org/openpub/) license defines the copyright on this document.

Message Delivery Semantics

How to guarantee delivery producer -> Kafka

When do we consider message delivered?

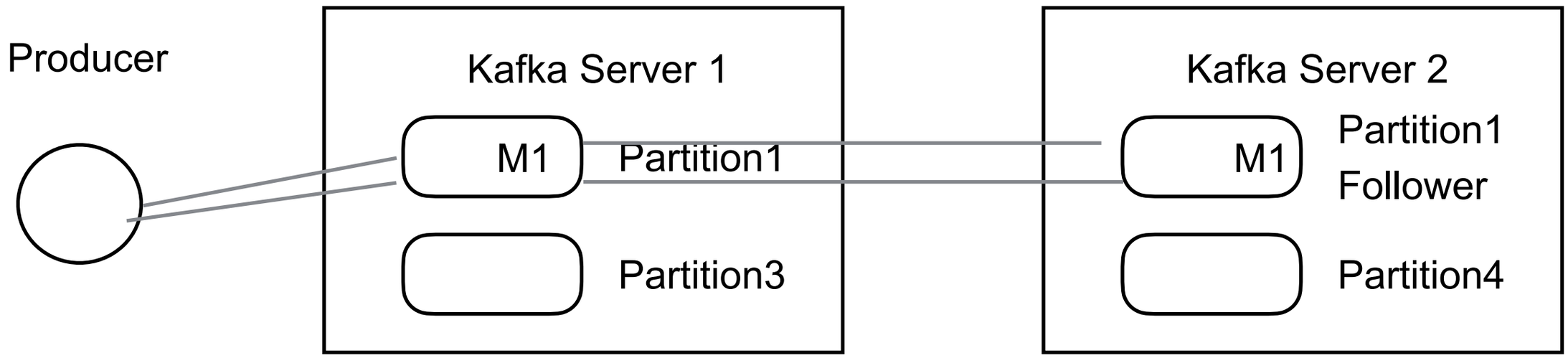


When leader get the message

When all replicated partitions get the message

Message Delivery Semantics

How to guarantee delivery producer -> Kafka



Producer does not know if message was delivered

Producer needs to resend message

Prior to 0.11.0 would create duplicate message in log

Message Delivery Semantics

How to guarantee delivery producer -> Kafka

Idempotent Delivery - Kafka 0.11.0

Each producer
- Has an ID
- Adds sequence number to messages

Kafka server checks for duplicates

Transactions - Kafka 0.11.0
- Producer can send messages to multiple topics
- All either succeed or all fail

Message Delivery Semantics

How to guarantee delivery producer -> Kafka

Producer can specify

- Be notified when leader and all followers have message
- Be notified when leader has message
- Provide a time out
- Receive no notifications

Replication

Each partition has a single leader and zero or more followers (slaves)

Followers are consumers of the leader partition
- Allows for batch reads

Live or in-sync node
- Node maintains its session with ZooKeeper
- If follower must not be too far behind leader
  - replica.lag.time.max.ms

Failed node
- One that is not in-sync

When a Leader Dies

In-sync replicas (ISR)

Follower partitions that are caught-up with leader

A machine may handle 1,000's of topics and even more partitions

ZooKeeper maintains the ISR set for each partition

When partition leader fails

Nodes in ISR vote on new leader

Message Delivery Semantics

How to guarantee delivery producer -> Kafka

Producer can specify

- Be notified when leader and all followers have message
- Be notified when leader has message
- Provide a time out
- Receive no notifications

Producer specifies ack parameter

- 0 - no acknowledgement
- 1 - acknowledgement when leader has message
- all (-1) - acknowledgement when all ISR partitions received the message

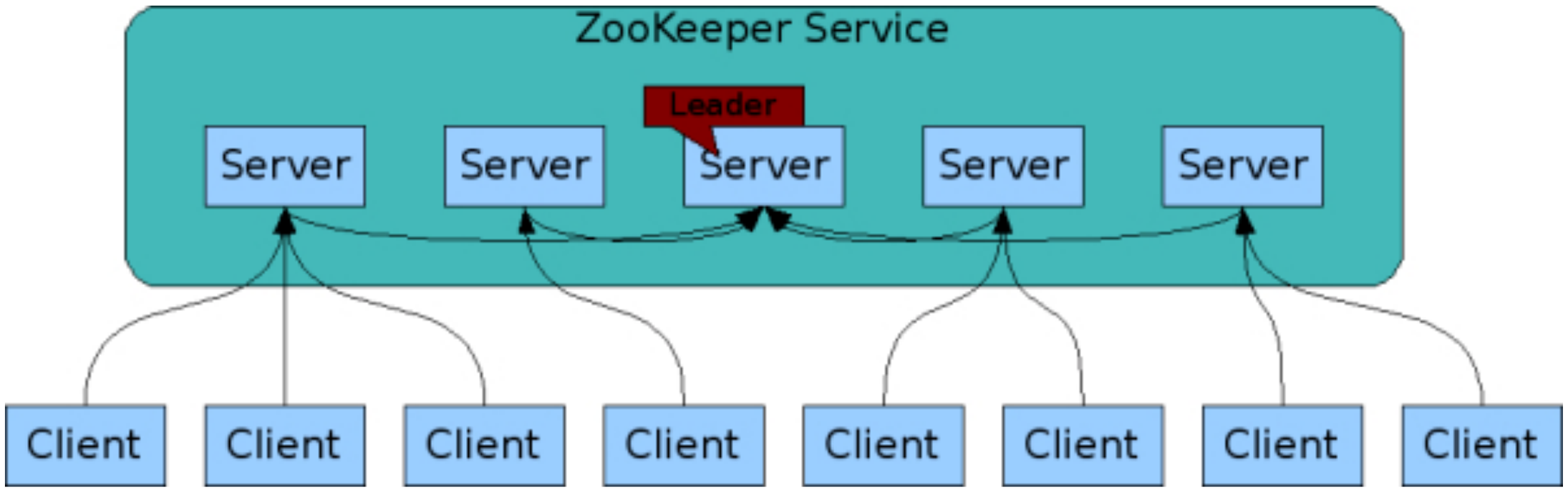
Zookeeper

Because coordinating distributed systems is a Zoo

Distributed hierarchical key-value store

For large distributed systems
- Distributed configuration service,
- Synchronization service,
- Naming registry

Zookeeper


ZooKeeper Service
Leader
Server
Server
Server
Server
Server
Client
Client
Client
Client
Client
Client
Client
Client

Running Kafka

Download and unpack Kafka

In Kafka directory

Start Zookeeper (it comes with Kafka)

```
bin/zookeeper-server-start.sh config/zookeeper.properties
```

Start Kafka Server

```
bin/kafka-server-start.sh config/server.properties
```

Create a topic

```
bin/kafka-topics.sh --create --zookeeper localhost:2181
                    --replication-factor 1 --partitions 1 --topic test
```

You can configure producers to auto-create topics when publish to new topic

Running Kafka

List of topics

```
bin/kafka-topics.sh --list --zookeeper localhost:2181
```

Send some messages

```
bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test
>this is a test
>this is a message
>Hi mom
```

Start a client

```
bin/kafka-console-consumer.sh --bootstrap-server localhost:9092    --topic test --from-begin
this is a test
this is a message
Hi mom
```

Running Kafka

Setting up a multi-broker cluster

Edit configuration so can run on same machine

cp config/server.properties config/server-1.properties

cp config/server.properties config/server-2.properties

Start the servers

bin/kafka-server-start.sh config/server-1.properties &

bin/kafka-server-start.sh config/server-2.properties &

Running Kafka

Create a new topic with a replication factor of three

```
bin/kafka-topics.sh --create --zookeeper localhost:2181
                    --replication-factor 3
                    --partitions 1
                    --topic my-replicated-topic
```

Get information about topic

```
bin/kafka-topics.sh --describe --zookeeper localhost:2181
                    --topic my-replicated-topic

Topic:my-replicated-topic      PartitionCount:1    ReplicationFactor:3      Configs:
        Topic: my-replicated-topic    Partition: 0    Leader: 1       Replicas: 1,2,0      Isr: 1,2,0
```

Send Records in Code

Records have
- Topic
- Key
- Value

Records are serialized

Kafka has serializers for base types

If want key or value to be compound type
- Need to provide serialized
- Consumer needs access to serializer
- Either
  - Attach to each message
  - Register with ZooKeeper

Running KafkaProducer Java Methods

initTransactions()

beginTransaction()

abortTransaction()

commitTransaction()

flush()

send(ProducerRecord<K,V> record)

send(ProducerRecord<K,V> record, Callback callback)

KafkaProducer(java.util.Properties properties)

KafkaProducer(java.util.Properties properties, Serializer<K> keySerializer,
Serializer<V> valueSerializer)

Simple Producer

```
Properties props = new Properties();
props.put("bootstrap.servers", "localhost:9092");
props.put("acks", "all");
props.put("retries", 0);
props.put("batch.size", 16384);
props.put("linger.ms", 1);
props.put("buffer.memory", 33554432);
props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");

Producer<String, String> producer = new KafkaProducer<>(props);
for (int i = 0; i < 100; i++)
    producer.send(new ProducerRecord<String, String>("my-topic", Integer.toString(i),
                                                     Integer.toString(i)));

producer.close();
```

With Transactions

```
Properties props = new Properties();
props.put("bootstrap.servers", "localhost:9092");
props.put("transactional.id", "my-transactional-id");
Producer<String, String> producer = new KafkaProducer<>(props, new StringSerializer(),
                                                                 new StringSerializer());
producer.initTransactions();

try {
    producer.beginTransaction();
    for (int i = 0; i < 100; i++)
        producer.send(new ProducerRecord<>("my-topic", Integer.toString(i), Integer.toString(i)));
    producer.commitTransaction();
} catch (ProducerFencedException | OutOfOrderSequenceException | AuthorizationException e) {
   producer.close();
} catch (KafkaException e) {
   producer.abortTransaction();
}
producer.close();
```

Consumer

```
Properties props = new Properties();
props.put("bootstrap.servers", "localhost:9092");
props.put("group.id", "test");
props.put("enable.auto.commit", "true");
props.put("auto.commit.interval.ms", "1000");
props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
props.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);
consumer.subscribe(Arrays.asList("foo", "my-topic"));
while (true) {
    ConsumerRecords<String, String> records = consumer.poll(100);
    for (ConsumerRecord<String, String> record : records)
        System.out.printf("offset = %d, key = %s, value = %s%n", record.offset(), record.key(),
                                record.value());
}
```

Python Client(s)

confluent-kafka-python

- From company that started Kafka
- Commercial support available
- Uses C library librdkafka
- Apache Kafka >= 0.9

https://github.com/confluentinc/confluent-kafka-python

kafka-python

- Maintained by Dana Powers
- 2x github stars as confluent-kafka-python
- Pure Python

https://github.com/dpkp/kafka-python

Using kafka-python Producer

Assume Kafka server is running

```
from kafka import KafkaProducer

producer = KafkaProducer(bootstrap_servers='localhost:9092',
                         compression_type='gzip')

producer.send('demo', 'hello'.encode() )
```

topic

message
Can only send bytes
b'hello' == 'hello'.encode()

Using kafka-python Consumer

```
from kafka import KafkaConsumer
consumer = KafkaConsumer('demo',
                         bootstrap_servers='localhost:9092')
for msg in consumer:
    print (msg)
    print (msg.value.decode())
```

Infinite Loop

```
ConsumerRecord(topic='demo', partition=0,
offset=15, timestamp=1556041866952,
timestamp_type=0, key=None, value=b'hello',
headers=[], checksum=None, serialized_key_size=-1,
serialized_value_size=5,
serialized_header_size=-1)

hello
```

Some kafka-python Producer Details

send(topic, value=None, key=None, headers=None, partition=None, timestamp_ms=None)

value

bytes or serializable by configured value_serializer

partition (int, optional) – optionally specify a partition

key (optional) – a key to associate with the message.

If partition is None (and producer's partitioner config is left as default),

then messages with the same key will be delivered to the same partition

headers (optional) – a list of header key value pairs.

Some kafka-python Producer Details

producer.send('demo', 'hello'.encode())

- Creates topic if 'demo' does not exist
- Done asynchronously
- Return a future object

Future

```
from kafka import KafkaProducer
from kafka.errors import KafkaError

producer = KafkaProducer(bootstrap_servers=['broker1:1234'])

# Asynchronous by default
future = producer.send('my-topic', b'raw_bytes')

# Block for 'synchronous' sends
try:
    record_metadata = future.get(timeout=10)
except KafkaError:
    # Decide what to do if produce request failed...
    log.exception()
    pass
```

Using Serializer

```
import json
producer = KafkaProducer(bootstrap_servers='localhost:9092',
                         value_serializer=lambda m: json.dumps(m).encode('ascii'))
producer.send('demo', [12,'cat', {'a':10}])
```

consumer get result as bytes.

Using deserializer to receive & decode JSON

```
import json
from kafka import KafkaConsumer
consumer = KafkaConsumer('demo',
                         bootstrap_servers='localhost:9092',
                         value_deserializer=lambda m: json.loads(m.decode('ascii')))
msg = next(consumer)
print(msg.value)
```

Callback on send completion

```
def on_send_success(record_metadata):
    print(record_metadata.topic)
    print(record_metadata.partition)
    print(record_metadata.offset)

def on_send_error(excp):
    log.error('I am an errback', exc_info=excp)

producer.send('demo', b'foo' ) \
    .add_callback(on_send_success)\
    .add_errback(on_send_error)
```

```
demo
0
22
```

asks & retries

```
producer = KafkaProducer(retries=5, acks='all')
producer.send('demo', 'with retries'.encode())
producer.close()
```

localhost:9092 is default

will retry 5 times to send message

acks

0 send do not wait for response from server

1 Wait for response that leader wrote message to log

all Wait for all partitions have written message to log

Spark Streaming & Spark Structured Streaming

Spark Streaming

Stream data to RDDs

Older

Spark Structured Streaming

Stream data to Dataframes

Real Streaming - Spark Streaming

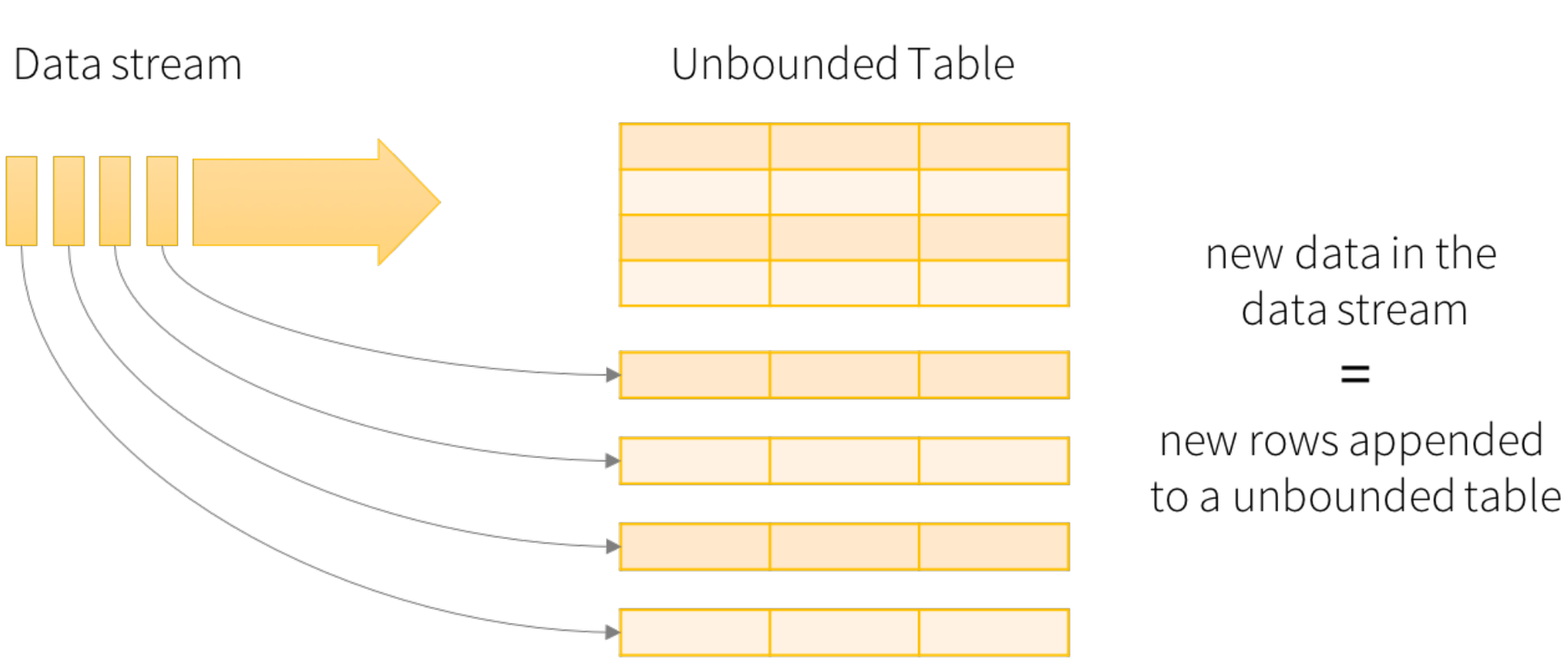
Data in stream is processed when received

Spark Streaming
- Groups input into batches
- Processes batch as group

Real Streaming - Spark Structured Streaming

Data in stream is processed when received

Spark Structured Streaming
- Processes data as it arrives
- Adds to unbounded table



Data stream as an unbounded table

Event Time - Late Data

Event time

- When the event happened
- When the data was created

Spark Streaming

- Processes data based on when it arrives

Spark Structured Streaming

- Can process data base on event time
- Even if data is late

End-to-end Guarantees

What happens when Spark application fails?

Spark use checkpointing

Save state of job

Structured Streaming requires

Source to be repayable

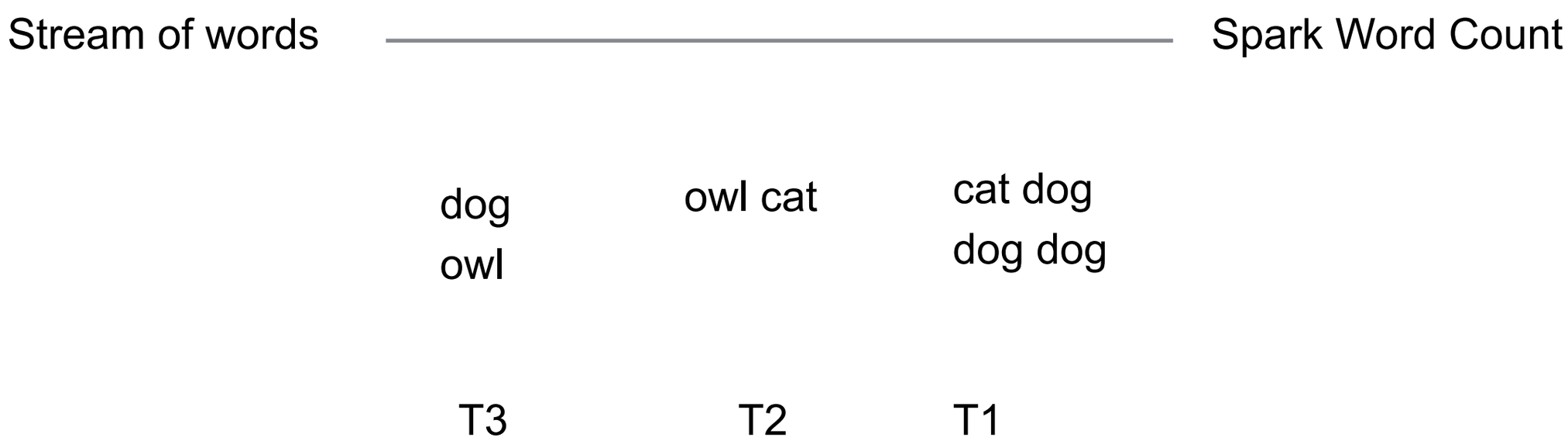
Sink support idempotent operations to support reprocessing

Spark Structured Streaming

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

Spark SQL engine
- Runs computation incrementally & continuously
- Updates the final result as streaming data continues to arrive

Spark Structured Streaming - Example



Word Count Example

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import explode
from pyspark.sql.functions import split

spark = SparkSession.builder  \
        .appName("StreamingWordCount") \
        .getOrCreate()

lines = spark \                                    # lines - DataFrame[value: string]
    .readStream \
    .format("socket") \
    .option("host", "localhost") \
    .option("port", 9999) \
    .load()

# Split the lines into words
words = lines.select(                              # Words - DataFrame[word: string]
   explode(
       split(lines.value, " ")
   ).alias("word")
)
```

Word Count Example

```
wordCounts = words.groupBy("word").count()   # DataFrame[word: string, count: bigint]

query = wordCounts \                    #pyspark.sql.streaming.StreamingQuery
    .writeStream \
    .outputMode("complete") \
    .format("console") \
    .start()

query.awaitTermination()
```

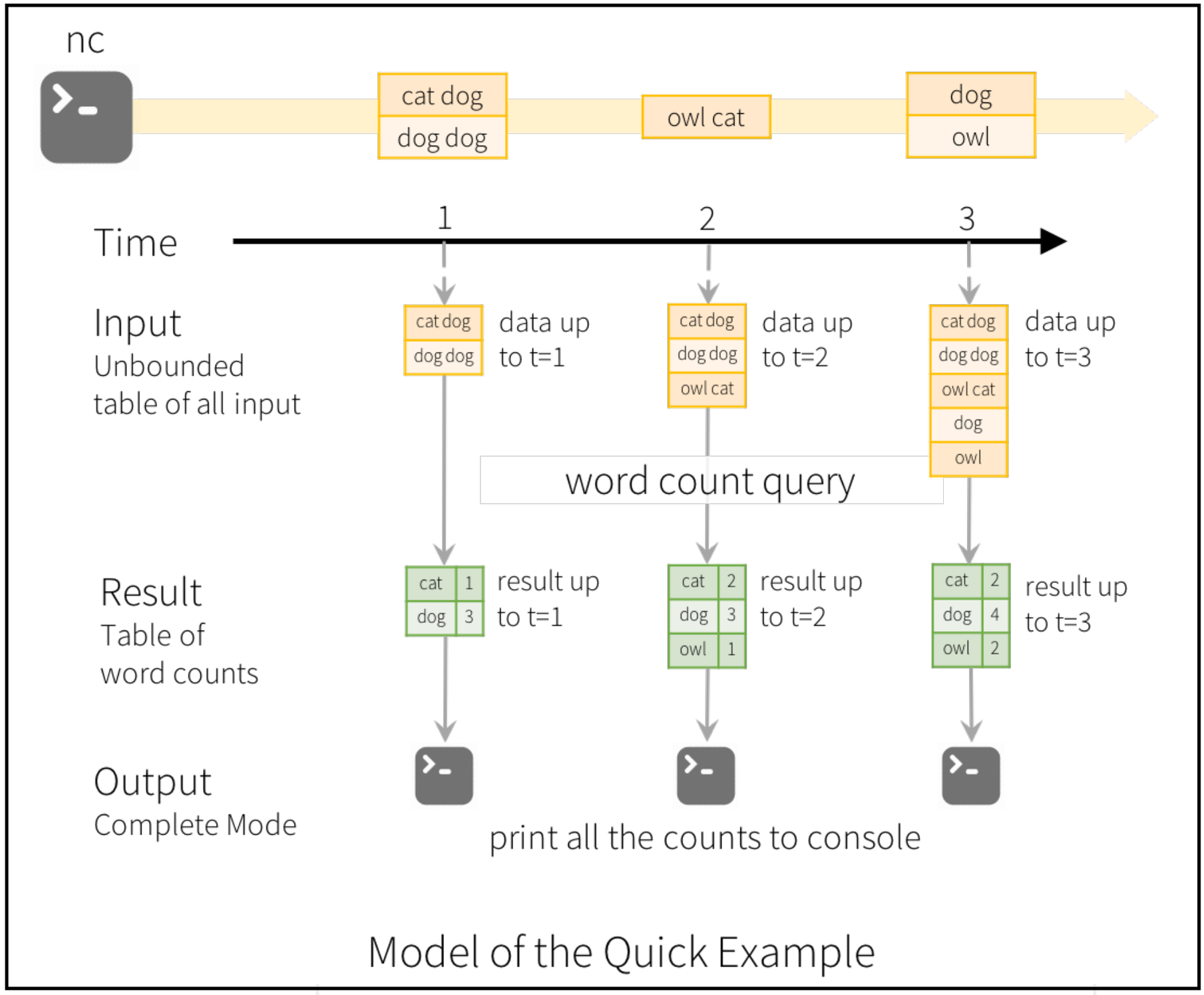
query.awaitTermination()

Runs until

query.stop()

Exception occurs

Output



Model of the Quick Example

Complete

Input

cat dog
dog dog

owl cat

dog owl

Output

```
-------------------
Batch: 1
-------------------
+-----+-----+
|value|count|
+-----+-----+
|  dog|    3|
|  cat|    2|
|  owl|    1|
+-----+-----+
```

```
-------------------
Batch: 2
-------------------
+-----+-----+
|value|count|
+-----+-----+
|  dog|    3|
|  cat|    2|
|  owl|    1|
+-----+-----+
```

```
-------------------
Batch: 3
-------------------
+-----+-----+
|value|count|
+-----+-----+
|  dog|    4|
|  cat|    2|
|  owl|    2|
+-----+-----+
```

Output Modes

Append

Only new rows in DataFrame are written to output

Complete

All rows are written to output

Update

Only rows that were updated will be written to output

Valid output modes

Depend on source & sink

Most sinks require watermark

```
query = wordCounts \                    #pyspark.sql.streaming.StreamingQuery
    .writeStream \
    .outputMode("complete") \
    .format("console") \
    .start()
```

Output Sinks

File

Kafka

Foreach

Console (debugging)

Memory (debugging)

Input Sources

File source -

Reads files written in a directory as a stream of data

File formats are text, csv, json, parquet

Kafka source -

Poll data from Kafka

Kafka versions 0.10.0 or higher

Socket source (for testing)

Reads UTF8 text data from a socket connection

Using Kafka as Source

Need spark-sql-kafka jar file

spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.11:2.4.2 kafkaWordCount.py

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import explode
from pyspark.sql.functions import split

if __name__ == "__main__":
    spark = SparkSession.builder.appName("StructuredKafkaWordCount").getOrCreate()
    lines = spark\
        .readStream\
        .format("kafka")\
        .option("kafka.bootstrap.servers", "localhost:9092")\
        .option('subscribe', 'test')\
        .load()\
    .selectExpr("CAST(value AS STRING)")

words = lines.select( explode( split(lines.value, ' ')).alias('word'))
wordCounts = words.groupBy('word').count()
query = wordCounts\
    .writeStream\
    .outputMode('complete')\
    .format('console')\
    .start()
query.awaitTermination()
```

Used To Reduce Logging Output

```
logger = spark.sparkContext._jvm.org.apache.log4j
logger.LogManager.getLogger("org"). setLevel( logger.Level.ERROR )
logger.LogManager.getLogger("akka").setLevel( logger.Level.ERROR )
```

Output Mode Complete

cat cat cat cat bat

cat cat cat mat

```
+----+-----+
|word|count|
+----+-----+
+----+-----+
-------------------------------------
Batch: 1
-------------------------------------
+----+-----+
|word|count|
+----+-----+
| bat|    1|
| cat|    4|
+----+-----+
-------------------------------------
Batch: 2
-------------------------------------
+----+-----+
|word|count|
+----+-----+
| bat|    1|
| cat|    7|
| mat|    1|
+----+-----+
```

Windows

Most sinks require a timestamp

Allows Spark to know how to group data

Window

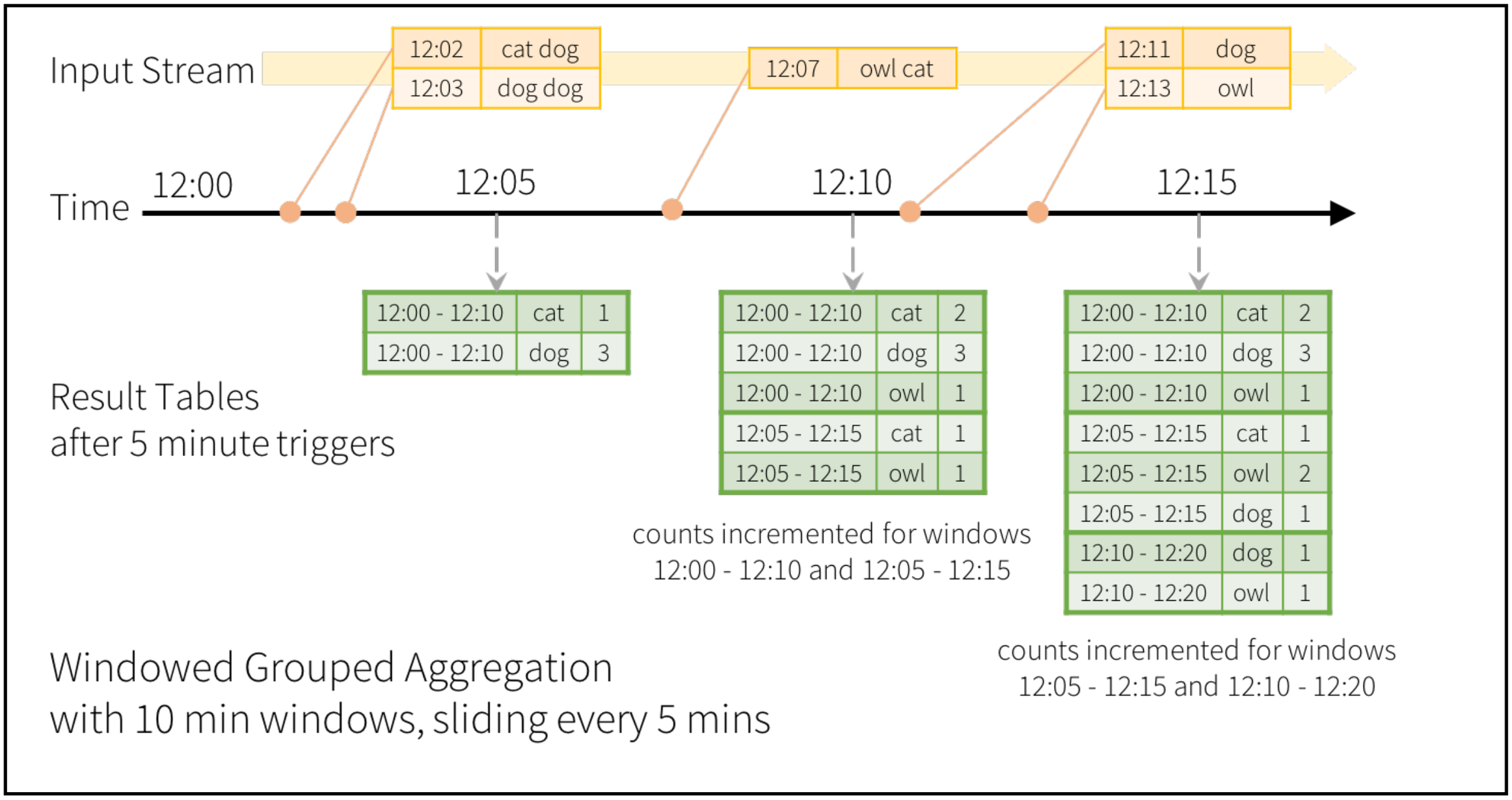
Time interval that data is grouped

Slide time

Once we reach end of window how far do we move the start of the next window

```
windowDuration = '5 seconds'
    slideDuration = '2 seconds'
    wordCounts =  words.groupBy(
        window(words.timestamp, windowDuration, slideDuration),
        words.word
    ).count().orderBy('window')
```

Window Operations on Event Time



```
spark = SparkSession.builder.appName("StructuredKafkaWordCount").getOrCreate()

 # Create DataSet representing the stream of input lines from kafka
 lines = spark\
     .readStream\
     .format("kafka")\
     .option("kafka.bootstrap.servers", "localhost:9092")\
     .option('subscribe', 'test')\
     .option('includeTimestamp', 'true')\                    # Include Kafka timestamp
     .load()

words = lines.select( explode(split(lines.value, ' ')).alias('word'),
    lines.timestamp                                          # keep Kafka timestamp
)

windowDuration = '5 seconds'
slideDuration = '2 seconds'
wordCounts =  words.groupBy(
    window(words.timestamp, windowDuration, slideDuration),
    words.word
).count().orderBy('window')
```

```
# Start running the query that prints the running counts to the console
   query = wordCounts\
       .writeStream\
       .outputMode('complete')\
       .format('console')\
       .start()

   query.awaitTermination()
```

```
-------------------------------------
Batch: 1
-------------------------------------
+------------------+----+-----+
|              window|word|count|
+------------------+----+-----+
|[2019-04-29 22:21...| cat|    4|
|[2019-04-29 22:21...| bat|    1|
|[2019-04-29 22:21...| bat|    1|
|[2019-04-29 22:21...| cat|    4|
+------------------+----+-----+
-------------------------------------
Batch: 2
-------------------------------------
+------------------+----+-----+
|              window|word|count|
+------------------+----+-----+
|[2019-04-29 22:21...| cat|    4|
|[2019-04-29 22:21...| bat|    1|
|[2019-04-29 22:21...| bat|    1|
|[2019-04-29 22:21...| cat|    4|
|[2019-04-29 22:21...| cat|    3|
|[2019-04-29 22:21...| mat|    1|
|[2019-04-29 22:21...| mat|    1|
|[2019-04-29 22:21...| cat|    3|
+------------------+----+-----+
```

Event-time and Late Data

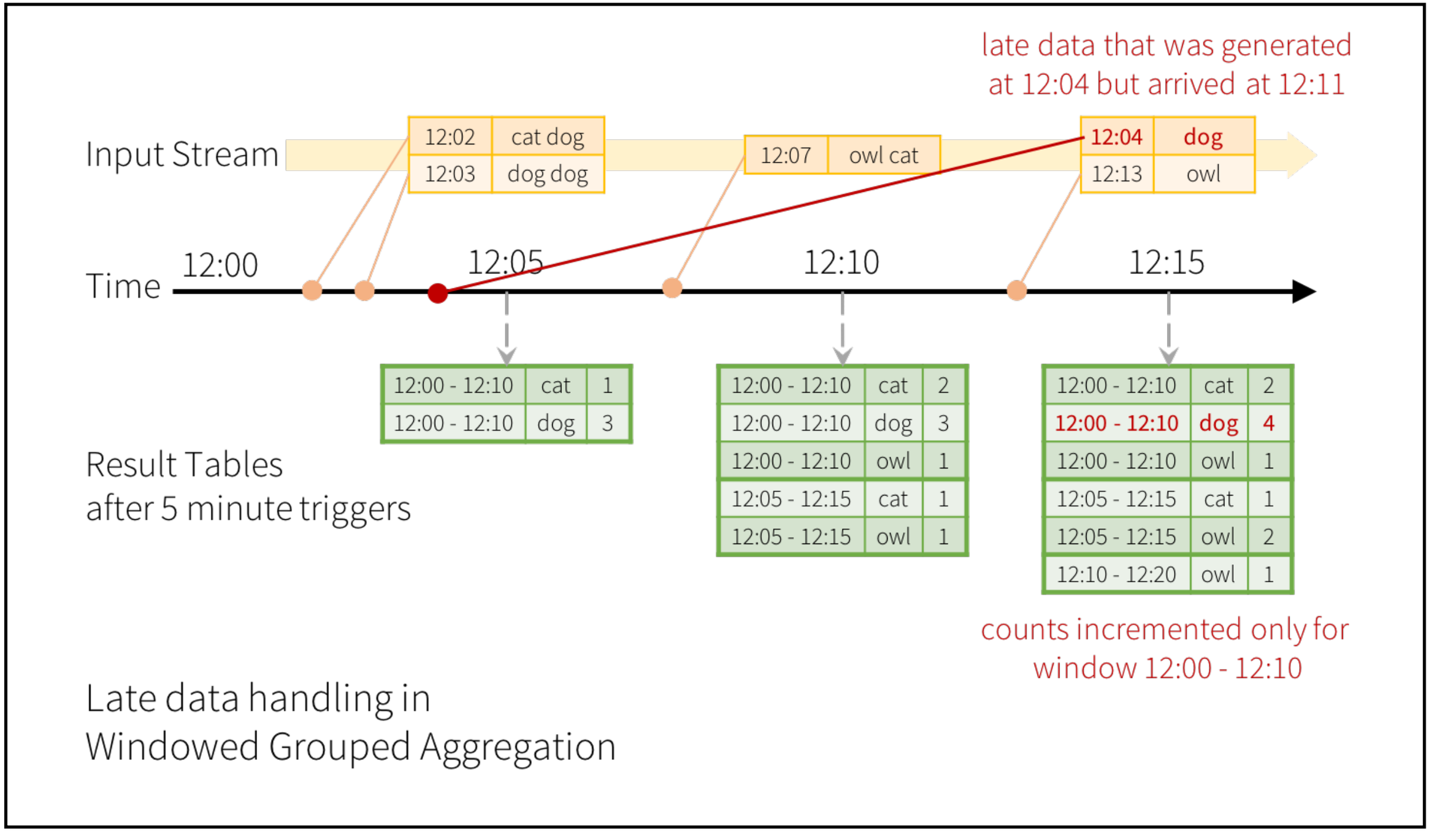
Event-time

- Messages contain timestamp
- Can use timestamp as event time

Event-time

- Message may come out of order

With Late Data



Watermarking

Can specify how late data can be and still be used

```
windowDuration = '5 seconds'
slideDuration = '2 seconds'
wordCounts =  words \
.withWatermark("timestamp", "10 minutes") \
.groupBy(
   window(words.timestamp, windowDuration, slideDuration),
   words.word
).count().orderBy('window')
```

Output Sinks & Modes

Output Modes

Append
  Only new rows in DataFrame are written to output

Complete
  All rows are written to output

Update
  Only rows that were updated will be written to output

Not all modes are available for all Sinks
  Complete - seems Console only
  File
    Append only
    No aggregations

Output Sinks

File

Kafka

Foreach

Console (debugging)

Memory (debugging)

Unsupported Operations

Multiple streaming aggregations

Limit and take first N rows are not supported on streaming Datasets.

Distinct operations on streaming Datasets are not supported.

Sorting operations are supported on streaming Datasets only after an aggregation and in Complete Output Mode.

Any kind of joins between two streaming Datasets is not yet supported.

count()
  use ds.groupBy().count() which returns a streaming Dataset containing a running count.

foreach() - Use ds.writeStream.foreach(...)

show() - Use the console sink