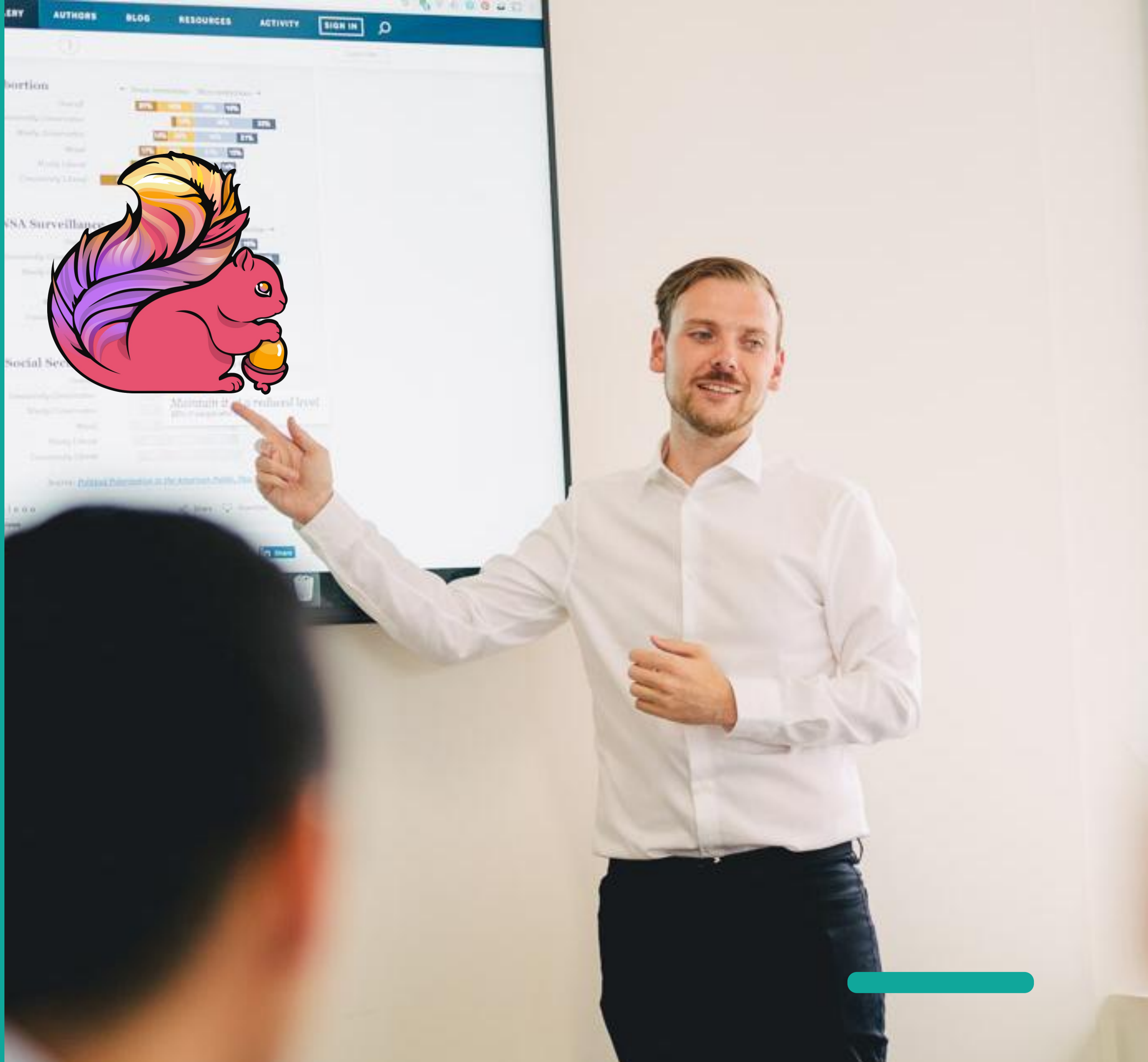
Scalable stream processing at Fairfax Media with Apache Flink



servian_

Agenda

- Introduction
- Motivating example
- Basic streaming concepts
- Streaming with Spark
- Move to Flink

Hello!

- Guy Needham
- Big data engineer since 2013
- Worked with many Hadoop ecosystem components:
  - Spark
  - MapReduce
  - Hive
  - Flume
  - Kafka
  - Flink
- Data science work in R and Python
- Scuba diver

Motivating Example

Streaming at Fairfax Media

Fairfax Media

servian_

Streaming at Fairfax Media

- Wish to process data in real time to better understand important metrics around content delivery
- Have around 600,000 events/minute at peak time, usually above 400,000 events/minute
- Need to
  - calculate metrics in real time based on a week long window
  - keep metrics up to date (sliding window)
  - enrich incoming events with results of aggregation

Fairfax Media

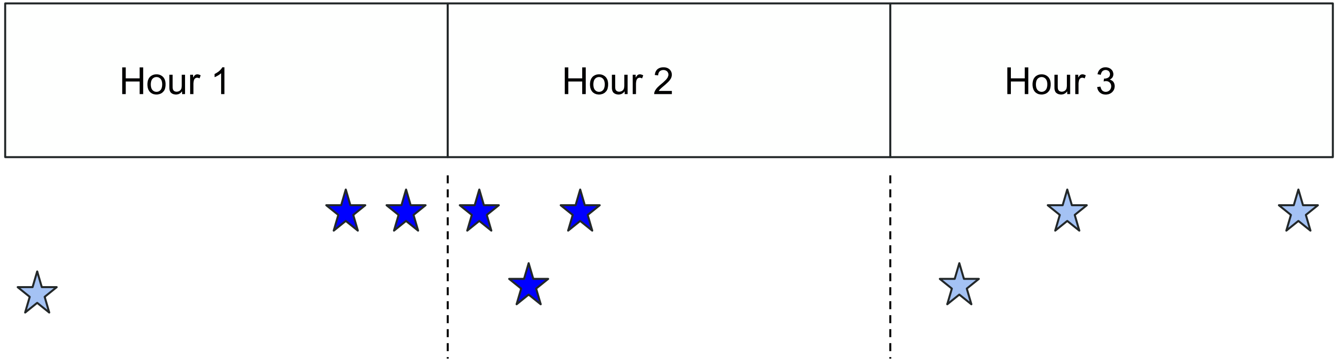
servian_

Streaming Basics

Key Concepts in Streaming

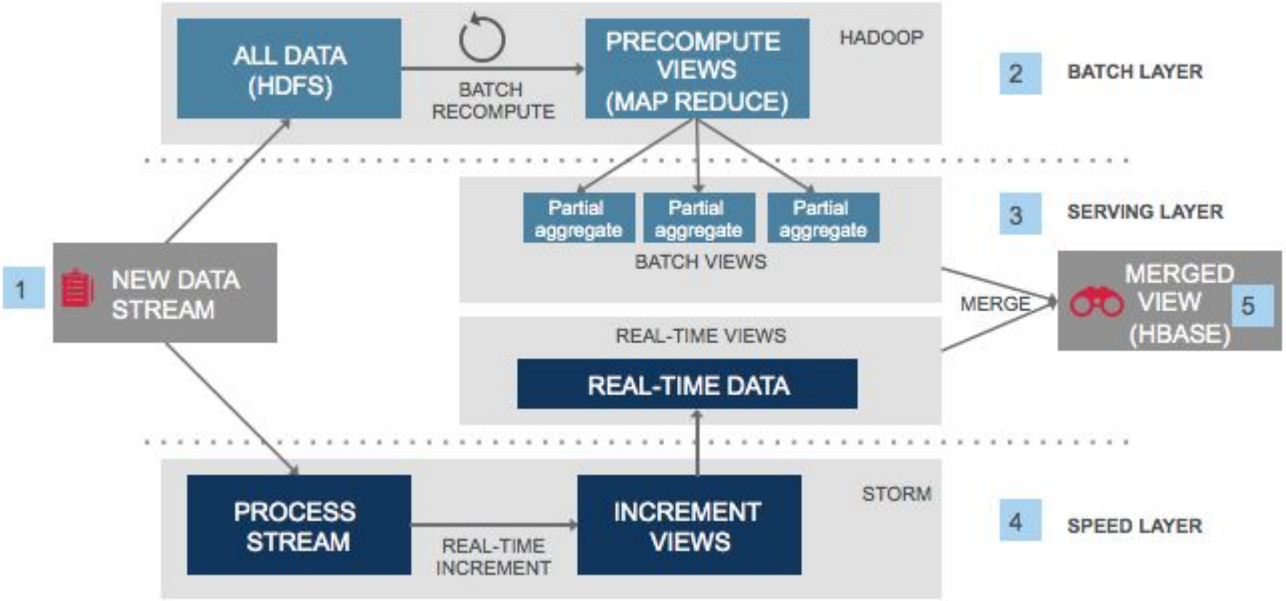
- Time:
  - Event time: when things actually happened
  - Processing time: when we observed the event
- Watermarks: how the system keeps track of time
- State: keeping track of computations
- Windows
- Microbatching vs true streaming

Differences to batch processing



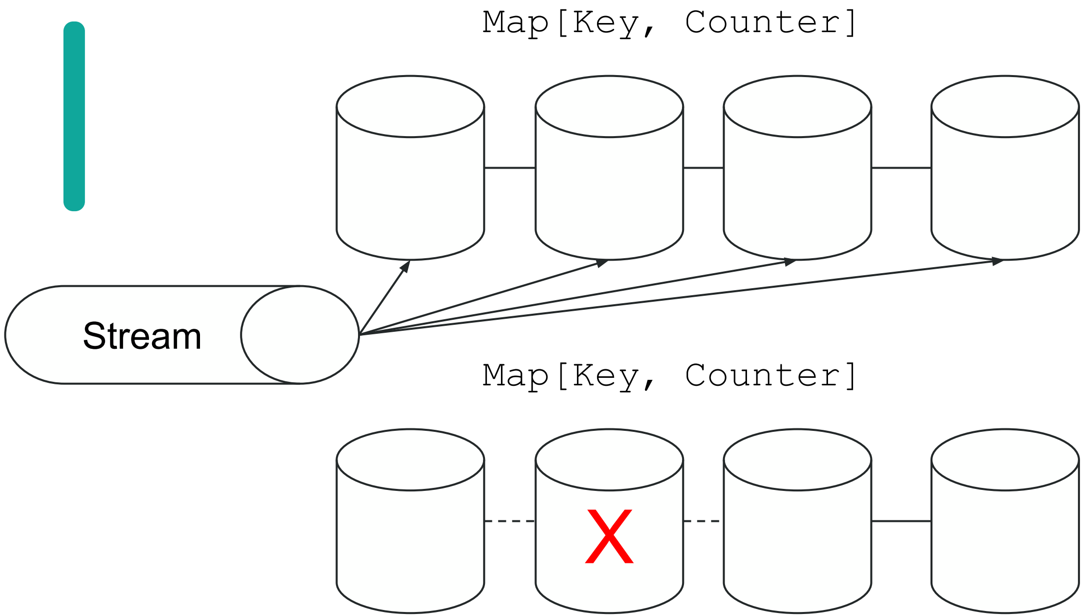
We should not introduce **unnatural partitions** into our data

Differences to batch processing



There should be no need for a complex architecture - the stream processor should be **correct**

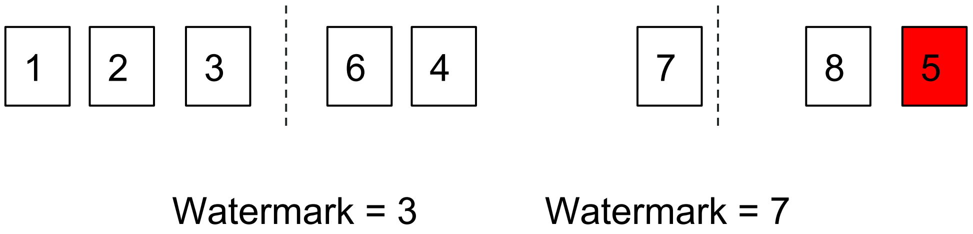
Differences to batch processing



In streaming, it’s often not an option to replay data if a node is lost.
We must be **fault tolerant** and handle **state**

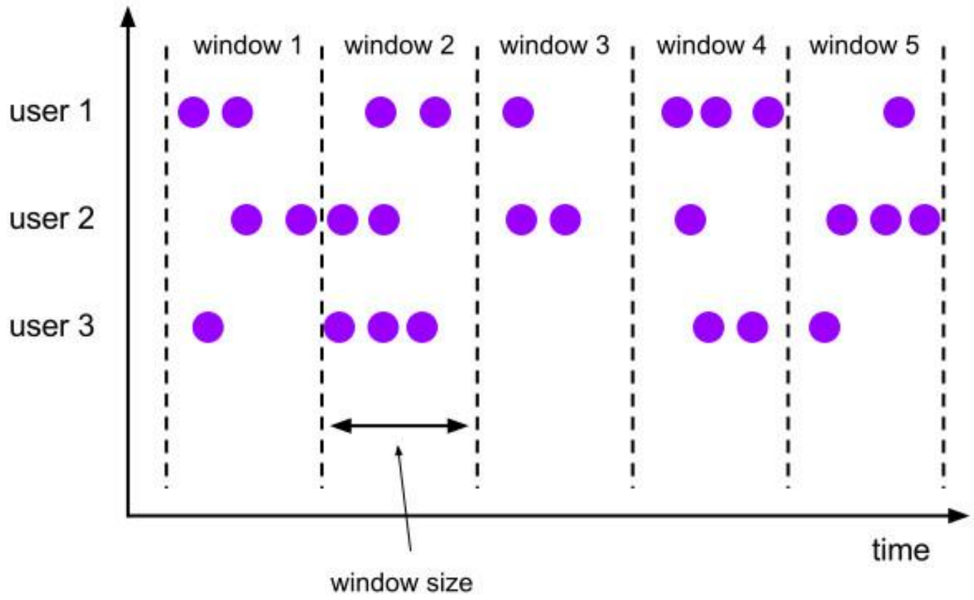
Watermarks

- How the system knows what time to process
- Mechanism for dealing with out of order data
- How to flag late events



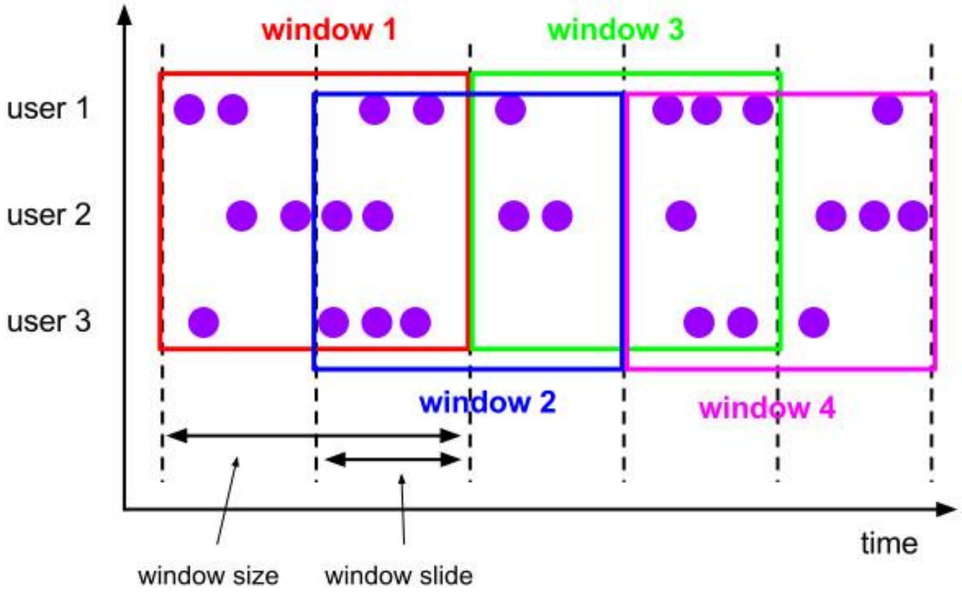
Windowing Operations

Tumbling Window



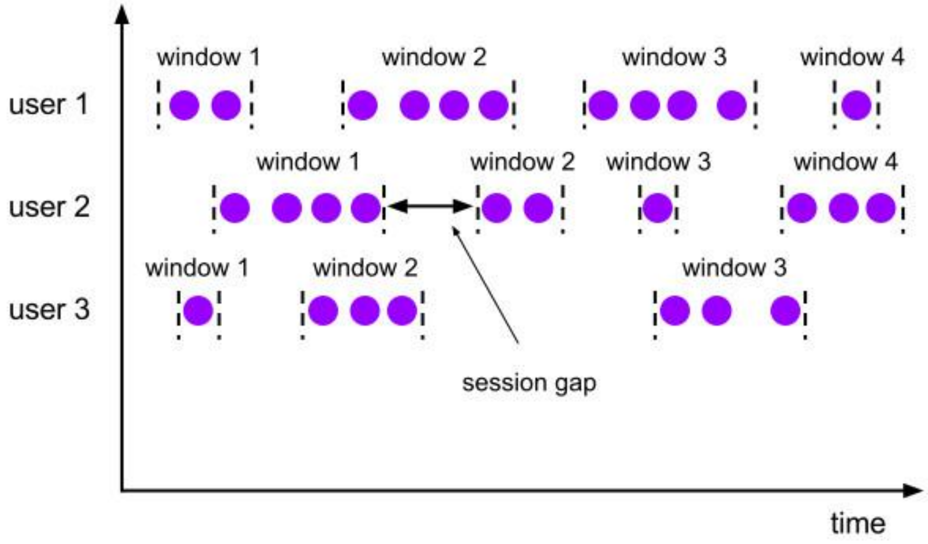
Windowing Operations

Sliding Window



Windowing Operations

Session Window



Microbatching and true streaming

- Microbatching discretises events into ever smaller batches
- Usually there is an overhead to creating the batch and cleaning up
- High latency but can be very high throughput
- True streaming processes events as they happen, so can achieve very low latency

Spark Streaming

Spark Streaming

- Microbatching framework
- Discretises the stream into batches
- Operate on the batches
- No support for event time - so can't do event time based windowing

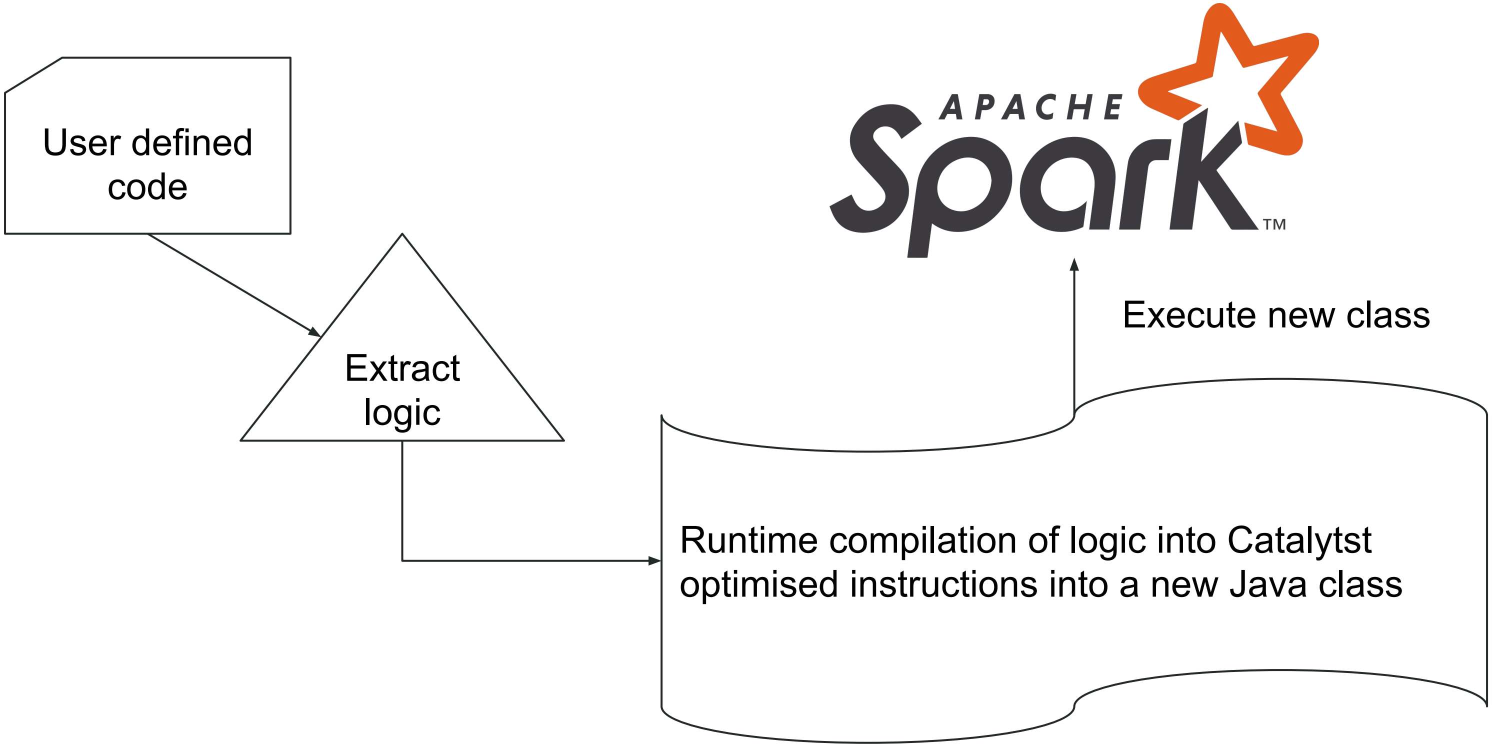
Spark Structured Streaming

- New in Spark 2
- Extension of Spark SQL to perform microbatches over streams
- Supports event time
- Spark SQL operations
    - Catalyst optimiser - lots of promise, many issues
    - In many cases lose the type safe guarantees
    - Tricky to fully test and make guarantees about what Spark will do with the user provided code

Spark Structured Streaming

- Attempted to build a:
  - count distinct of value by user ID
  - Sliding window length: 1 hour
  - Sliding window duration: 1 minute
  - using event time
- Not that complicated

SparkSQL under the hood



Spark Structured Streaming

Spark / SPARK-18492

GeneratedIterator grows beyond 64 KB

Export

Details

Type:	Bug	Status:	OPEN
Priority:	Major	Resolution:	Unresolved
Affects Version/s:	2.0.1	Fix Version/s:	None
Component/s:	SQL		
Labels:	None		
Environment:	CentOS release 6.7 (Final)		

People

Assignee:	Unassigned
Reporter:	Norris Merritt
Votes:	11 Vote for this issue
Watchers:	30 Start watching this issue

Dates

Created:	17/Nov/16 18:47
Updated:	26/Sep/18 18:50

Description

spark-submit fails with ERROR CodeGenerator: failed to compile: org.codehaus.janino.JaninoRuntimeException: Code of method "(I[Lscala/collection/Iterator;)V" of class "org.apache.spark.sql.catalyst.expressions.GeneratedClass$GeneratedIterator" grows beyond 64 KB

Error message is followed by a huge dump of generated source code.

The generated code declares 1,454 field sequences like the following:

Enter Flink

Apache Flink

- True stream processor
- Fine grained control over the processor if required
- Growing SQL support, strong batch support
- Fully unit testable:
  - Can define tests and run the entire job as tests
  - Speeds up development cycles
  - Gives guarantees about processing before building a cluster
- Included with EMR, so no need to install ourselves

Apache Flink

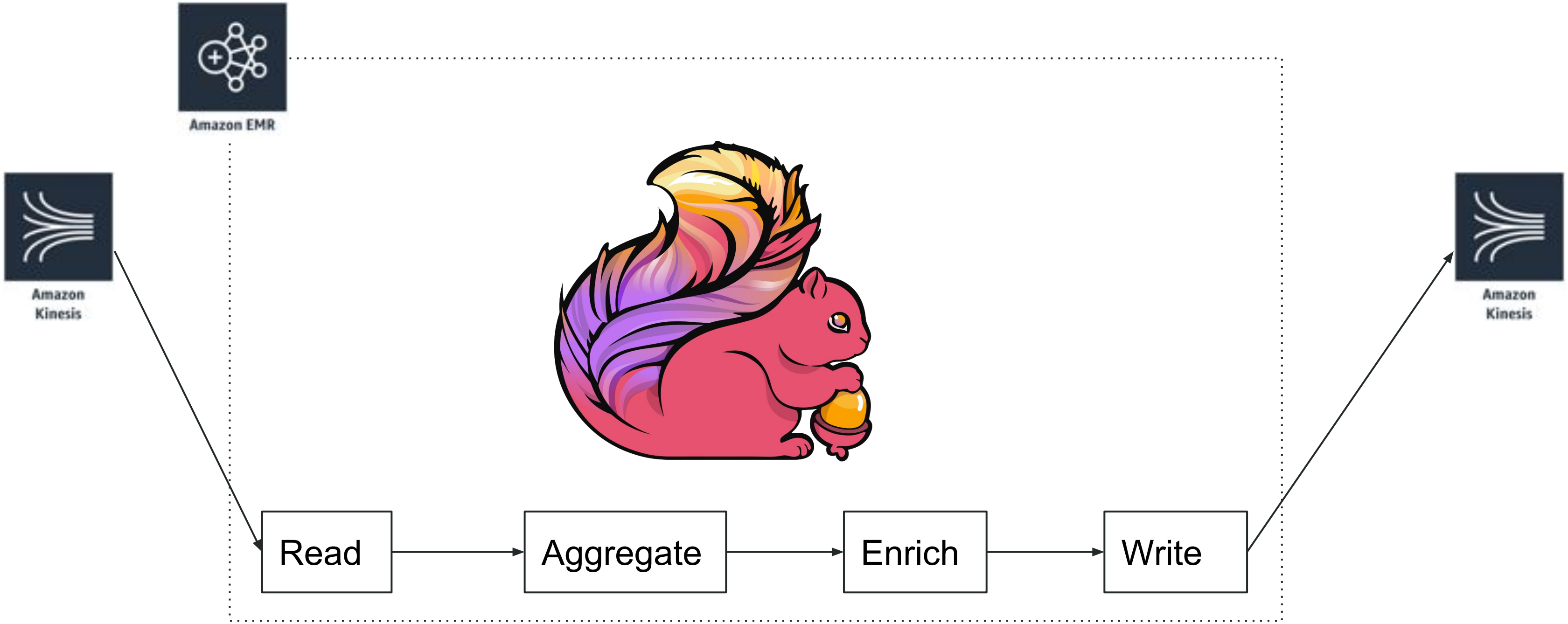
Flink Delivers ACID Transactions on Streaming Data - Datanami
https://www.datanami.com/2018/.../flink-delivers-acid-transactions-on-streaming-data/
Sep 4, 2018 - **Flink** Delivers **ACID Transactions** on Streaming Data ... DB2 to serve their needs (although NoSQL databases are **adding ACID** support too).

Serializable ACID Transactions on Streaming Data - data Artisans
https://data-artisans.com/blog/serializable-acid-transactions-on-streaming-data
Sep 4, 2018 - In this post, we will explain why serializable **ACID transactions** are an extremely ... data Artisans Streaming Ledger builds on Apache **Flink** and provides ... For each row being accessed, you **add** a call that specifies the access: ...

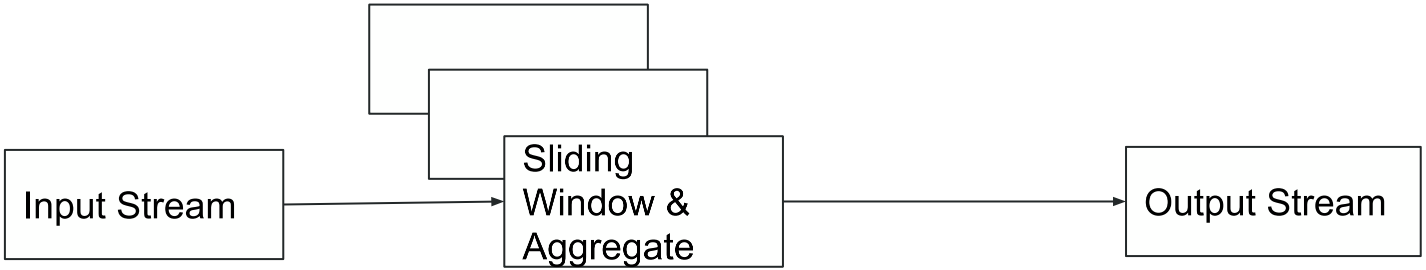
Apache Flink takes ACID | ZDNet
https://www.zdnet.com/article/apache-flink-takes-acid/
Sep 4, 2018 - Unlike distributed databases, the Streaming Ledger does not use normal database locks or approaches like multi-version concurrency control (MVCC). As **Flink** is not a database, **transaction** logic is contained in the application code (the **transaction** function), with data persisted in memory or in RocksDB.

Data Artisans Announces Serializable ACID Transactions on ... - InfoQ
https://www.infoq.com/news/2018/09/data-artisans-acid-streaming
Sep 12, 2018 - The patent-pending technology is a proprietary **add**-on for **Flink** and allows ... **Flink** with capabilities to perform serializable **ACID transactions** ...

Aggregating stream processor



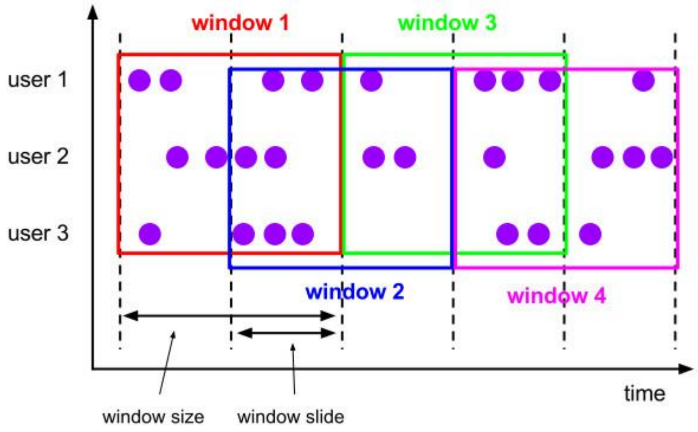
Logical flow, take 1



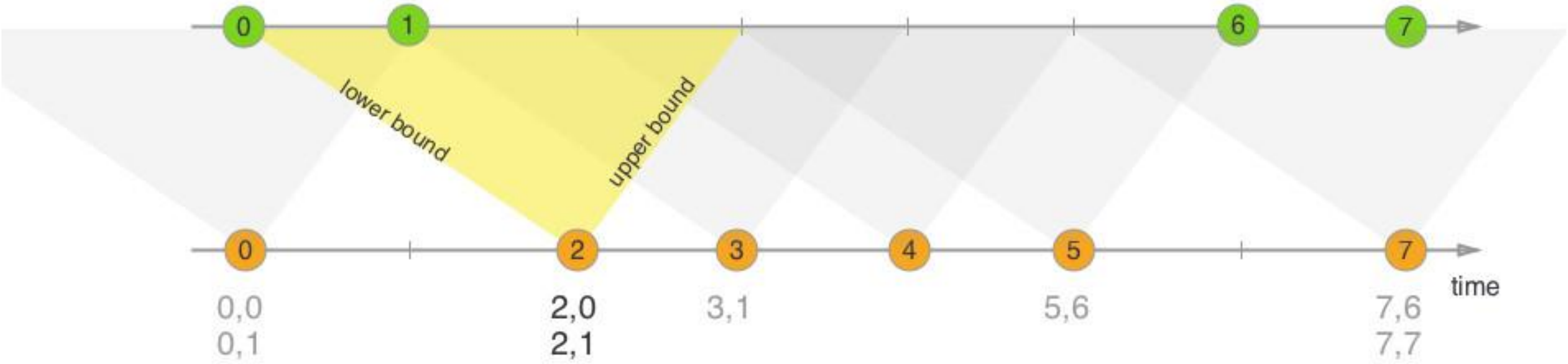
- Builds up window, computes aggregate
- Leads to many duplicates, events output once per window

Sliding windows and duplicates

- The boxes in the diagram overlap
- If the raw window was output, each point would be produced many times
- Not what we want
- Must work out how to join the most recent aggregate results onto the input data



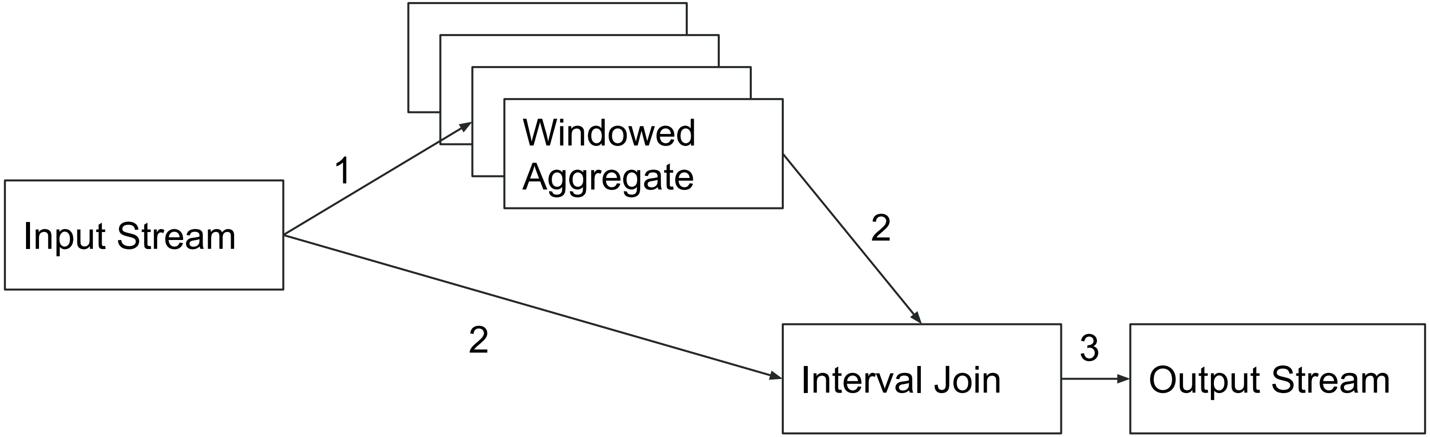
Interval Joining in Flink



- Join the orange stream on the green stream
- Where green time is between orange time +/- some time bound
- **join a on b**

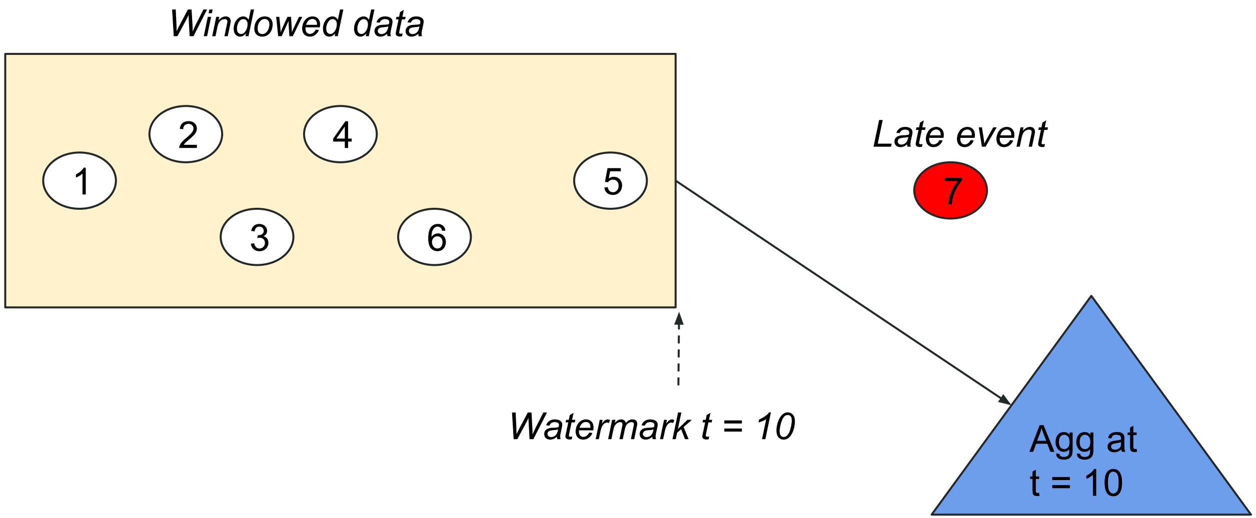
  where a.time between (b.time - lower_bound) and (b.time + upper_bound)

Logical flow, take 2



- Input is windowed and then the input is joined on the aggregate
- Input is multicast as input into the interval join
- Interval join is from event time to the event time plus slide duration - gives exactly one record output per input record

Late Data



Late Data

Options:

1. repeat aggregation
2. ignore late event
3. output late event without modification

Late Data

Flink supports these:

1. repeat aggregation
   - retrigger with allowed lateness
2. ignore late event
   - do nothing, lose data
3. output late event without modification
   - get the late events as a separate stream
   - handle differently

Late Data

```
val lateTag = ...
val agg = keyedStream
    .window(<aggregator>)
    .allowedLateness(Time…)
    .sideOutputLateData(lateTag)
agg…
val lateData = agg.getSideOutput(lateTag)
```

Restartability

- Input data:
  - We need to be able to consume weeks of data on restart
  - Input data is in Kinesis which only stores for 7 days
  - Read in older data from JSON files on S3 and union that stream with the Kinesis data
- State:
  - Want to be able to restart processor from checkpoints or restart when change code
  - Flink checkpoints state regularly into RocksDB
  - Can start up from savepoints (manually taken) or checkpoints

Conclusions

Experience with Apache Flink so far

- Very fast and efficient
  - Needs about 30x fewer resources than Spark to do similar work
- No horrific bugs (so far)
- Fantastic API, as long as you don't mind code being a little verbose - it's a Java project after all
- Operationally much easier to work with than Spark:
  - UI makes understanding what's going on easy
  - Watermarks, data volumes moving are displayed
  - No need to kill a process to end the execution
  - Can make a savepoint and restart from there including all internal state
- Enjoy the more fine grained control of logic compared to SparkSQL
  - No unnecessary over optimisation

Streaming doesn't have to be hard

- Choosing the right technology for your stream processor is key
- With the right technology and skills, stream processing is now mature enough to make a difference
- Life doesn't happen in batch mode - why process data in batch?
- Flink is a fantastic reliably scalable stream processor