Real-time Stream Processing with Apache Flink

Marton Balassi – data Artisans
Gyula Fora - SICS
Flink committers
mbalassi@apache.org / gyfora@apache.org

Stream Processing

- ***Data stream***: Infinite sequence of data arriving in a continuous fashion.
- ***Stream processing***: Analyzing and acting on real-time streaming data, using continuous queries

Streaming landscape

Apache Storm

- True streaming, low latency - lower throughput
- Low level API (Bolts, Spouts) + Trident

Spark Streaming

- Stream processing on top of batch system, high throughput - higher latency
- Functional API (DStreams), restricted by batch runtime

Apache Samza

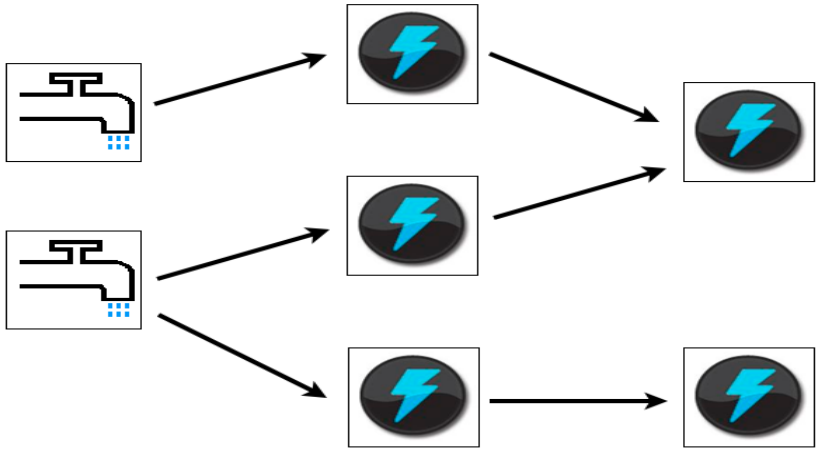
- True streaming built on top of Apache Kafka, state is first class citizen
- Slightly different stream notion, low level API

Apache Flink

- True streaming with adjustable latency-throughput trade-off
- Rich functional API exploiting streaming runtime; e.g. rich windowing semantics

Apache Storm

- True streaming, low latency - lower throughput
- Low level API (Bolts, Spouts) + Trident
- At-least-once processing guarantees

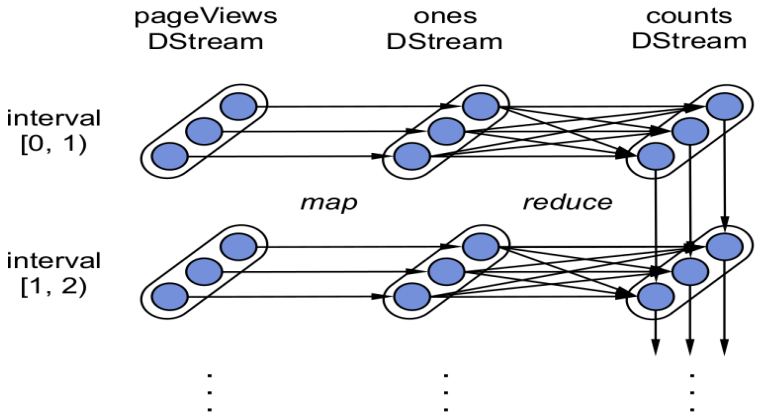


Issues

- Costly fault tolerance
- Serialization
- Low level API

Spark Streaming

- Stream processing emulated on a batch system
- High throughput - higher latency
- Functional API (DStreams)
- Exactly-once processing guarantees

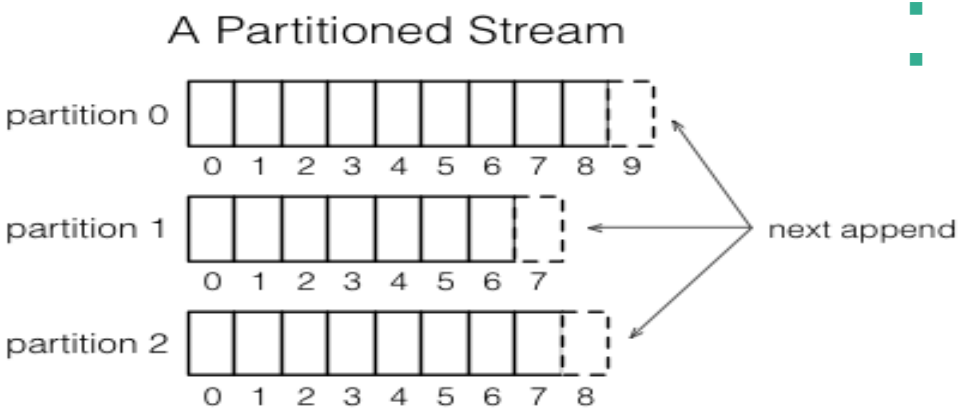


Issues

- Restricted streaming semantics
- Windowing
- High latency

Apache Samza

- True streaming built on top of Apache Kafka
- Slightly different stream notion, low level API
- At-least-once processing guarantees with state

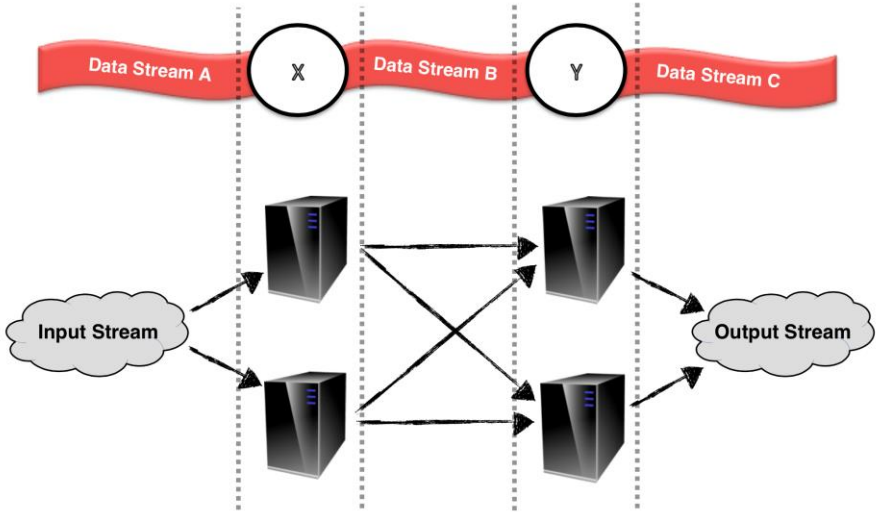


Issues

- High disk IO
- Low level API

Apache Flink

- True streaming with adjustable latency and throughput
- Rich functional API exploiting streaming runtime
- Flexible windowing semantics
- Exactly-once processing guarantees with (small) state



Issues

- Limited state size
- HA issue

Apache Flink

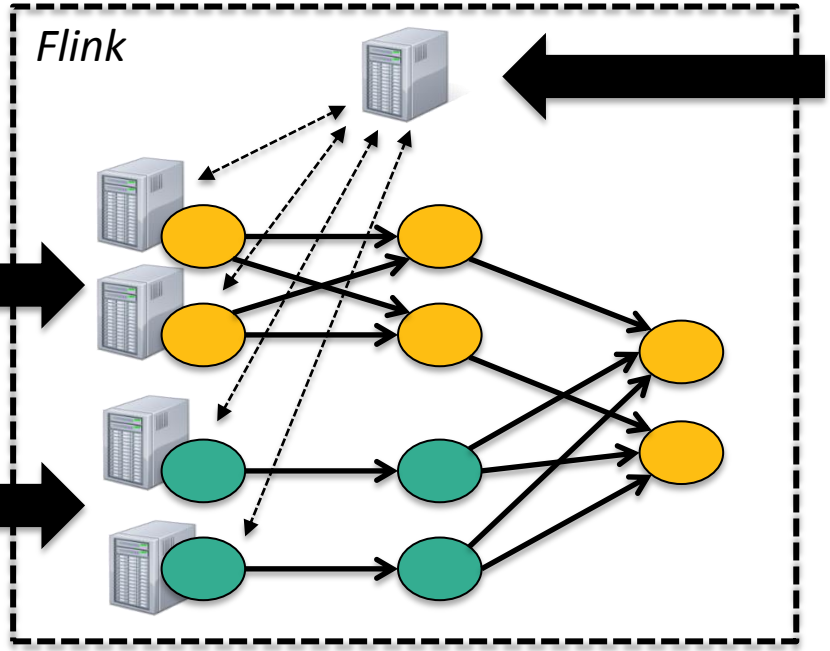
What is Flink

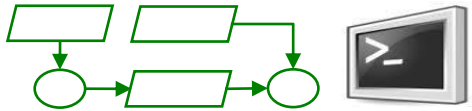
A "use-case complete" framework to unify batch and stream processing

What is Flink

Real-time data streams

An engine that puts equal emphasis to streaming and batch





ETL, Graphs,
Machine Learning
Relational, …

Low latency
windowing,
aggregations, ...

Event logs

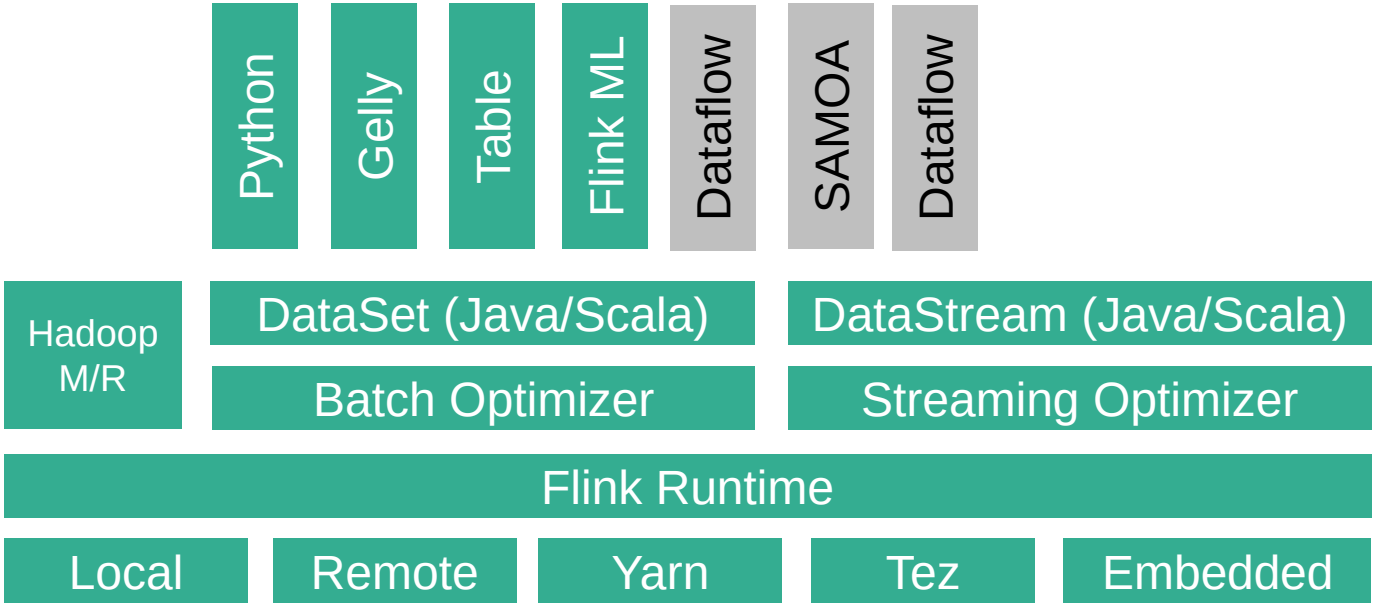
Kafka, RabbitMQ, ...

Historic data

HDFS, JDBC, ...

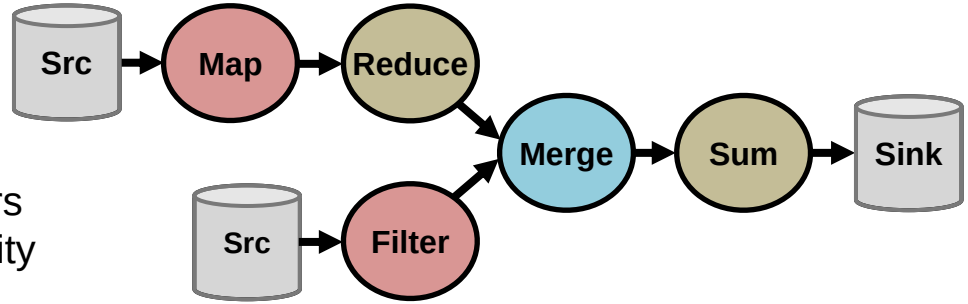
Flink stack

**current Flink master + few PRs*



Flink Streaming

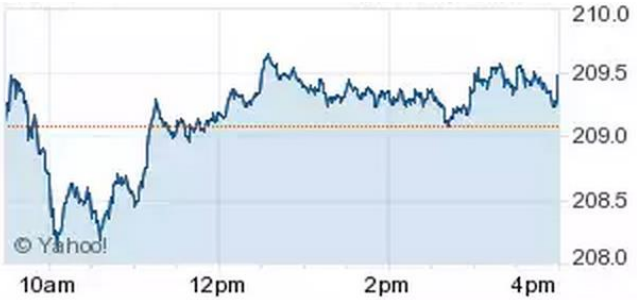
Overview of the API



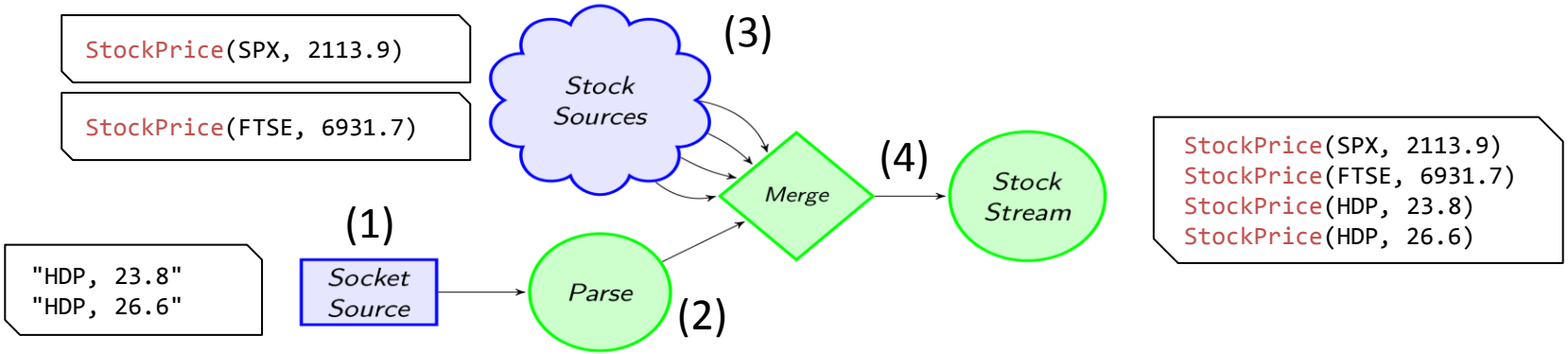
- Data stream sources
  - File system
  - Message queue connectors
  - Arbitrary source functionality
- Stream transformations
  - Basic transformations: *Map, Reduce, Filter, Aggregations*…
  - Binary stream transformations: *CoMap, CoReduce*…
  - Windowing semantics: *Policy based flexible windowing (Time, Count, Delta…)*
  - Temporal binary stream operators: *Joins, Crosses*…
  - Native support for iterations
- Data stream outputs
- For the details please refer to the programming guide:
  - http://flink.apache.org/docs/latest/streaming_guide.html

Use-case: Financial analytics

- Reading from multiple inputs
  - Merge stock data from various sources
- Window aggregations
  - Compute simple statistics over windows of data
- Data driven windows
  - Define arbitrary windowing semantics
- Combine with sentiment analysis
  - Enrich your analytics with social media feeds (Twitter)
- Streaming joins
  - Join multiple data streams
- Detailed explanation and source code on our blog
  - http://flink.apache.org/news/2015/02/09/streaming-example.html



Reading from multiple inputs

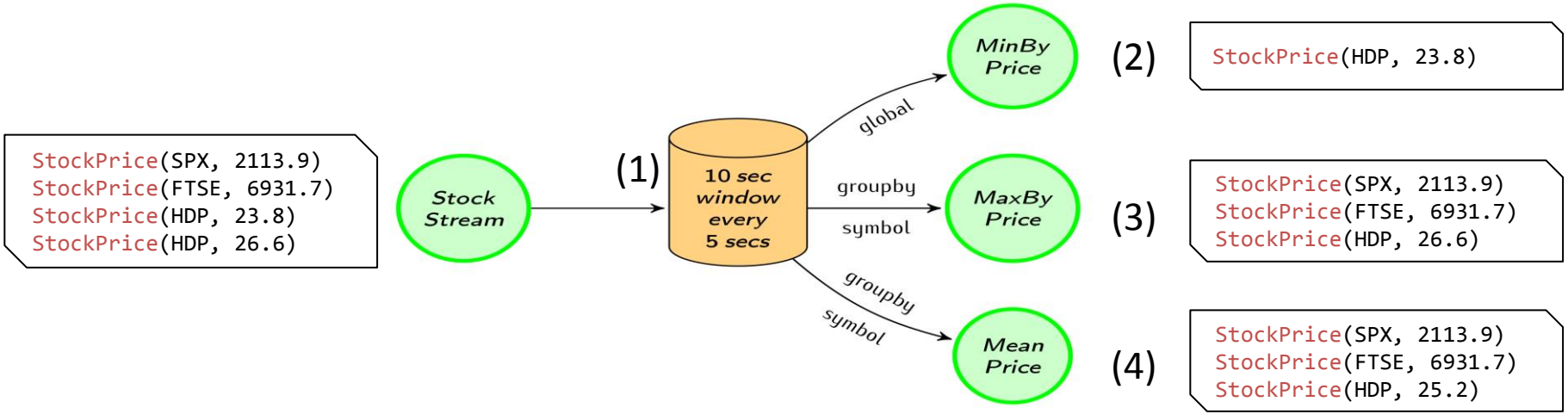



```
case class StockPrice(symbol : String, price : Double)
val env = StreamExecutionEnvironment.getExecutionEnvironment

(1) val socketStockStream = env.socketTextStream("localhost", 9999)
(2)     .map(x => { val split = x.split(",")
            StockPrice(split(0), split(1).toDouble) })

(3) val SPX_Stream = env.addSource(generateStock("SPX")(10) _)
    val FTSE_Stream = env.addSource(generateStock("FTSE")(20) _)
(4) val stockStream = socketStockStream.merge(SPX_Stream, FTSE_STREAM)
```

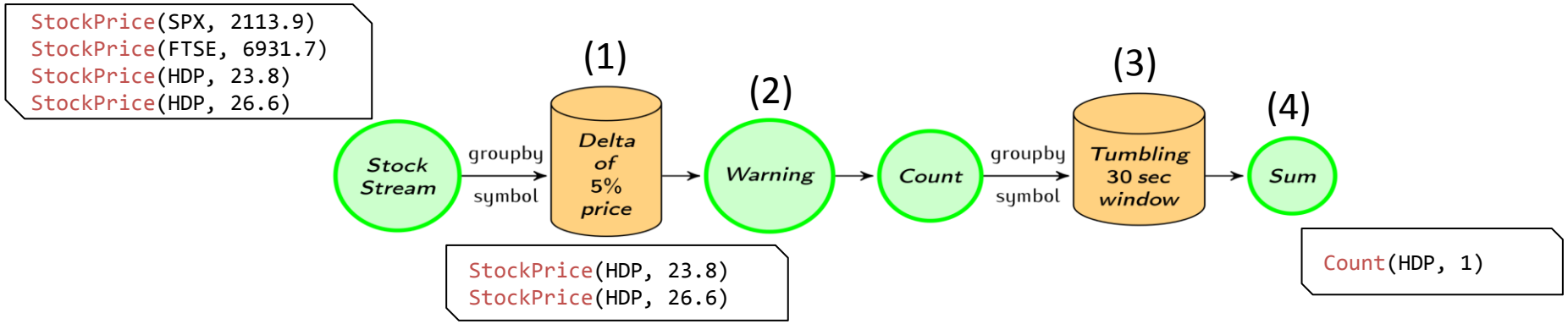
Window aggregations




```
    val windowedStream = stockStream
(1)   .window(Time.of(10, SECONDS)).every(Time.of(5, SECONDS))

(2) val lowest = windowedStream.minBy("price")
(3) val maxByStock = windowedStream.groupBy("symbol").maxBy("price")
(4) val rollingMean = windowedStream.groupBy("symbol").mapWindow(mean _)
```

Data-driven windows

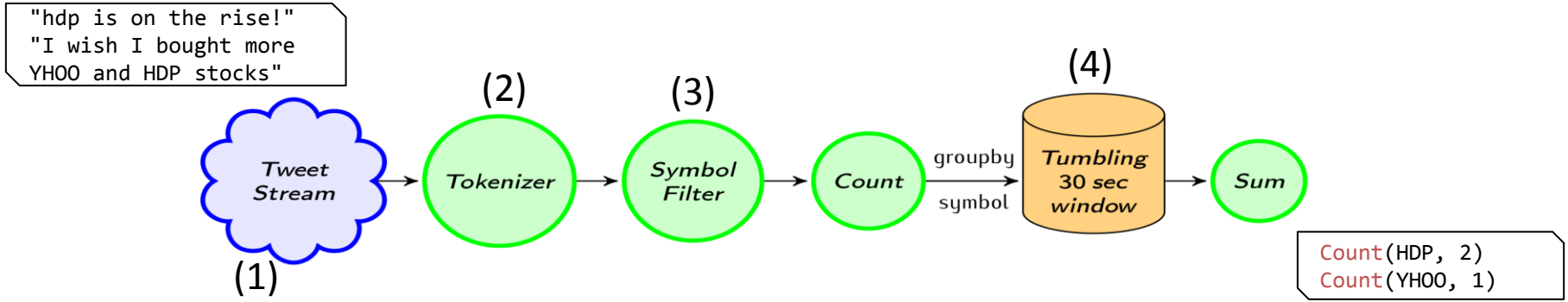



```
case class Count(symbol : String, count : Int)

val priceWarnings = stockStream.groupBy("symbol")
(1)     .window(Delta.of(0.05, priceChange, defaultPrice))
(2)     .mapWindow(sendWarning _)

val warningsPerStock = priceWarnings.map(Count(_, 1)) .groupBy("symbol")
(3)     .window(Time.of(30, SECONDS))
(4)     .sum("count")
```

Combining with a Twitter stream



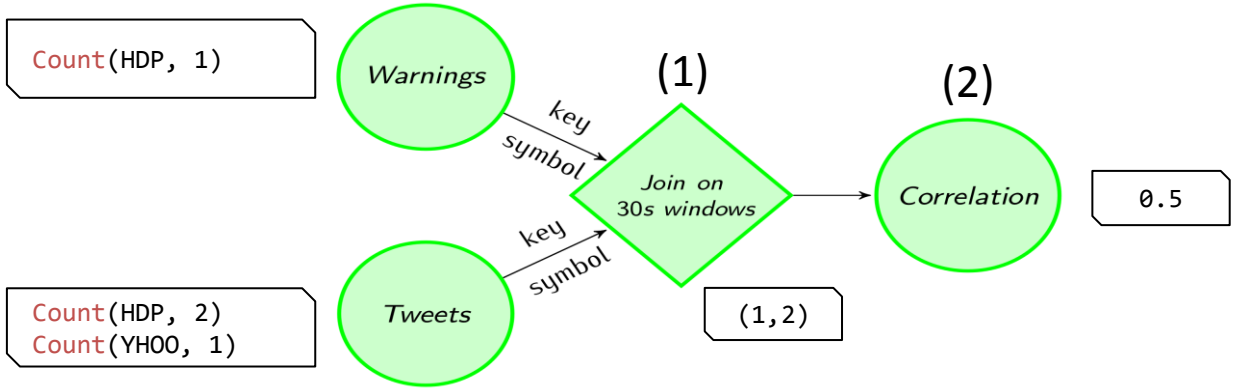
(1)
```
val tweetStream = env.addSource(generateTweets _)
```

(2)
```
val mentionedSymbols = tweetStream.flatMap(tweet => tweet.split(" "))
    .map(_.toUpperCase())
```

(3)
```
    .filter(symbols.contains(_))
```

(4)
```
val tweetsPerStock = mentionedSymbols.map(Count(_, 1)).groupBy("symbol")
    .window(Time.of(30, SECONDS))
    .sum("count")
```

Streaming joins



(1)

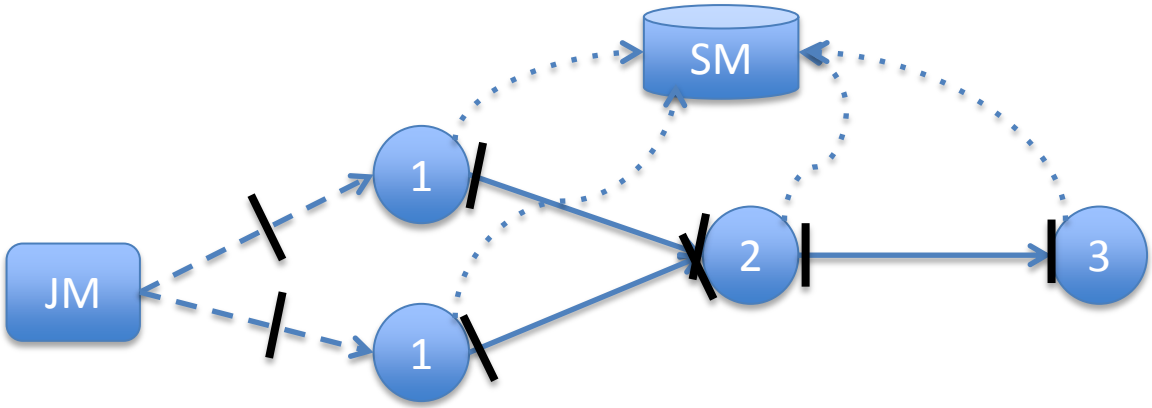
```
val tweetsAndWarning = warningsPerStock.join(tweetsPerStock)
    .onWindow(30, SECONDS)
    .where("symbol")
    .equalTo("symbol"){ (c1, c2) => (c1.count, c2.count) }
```

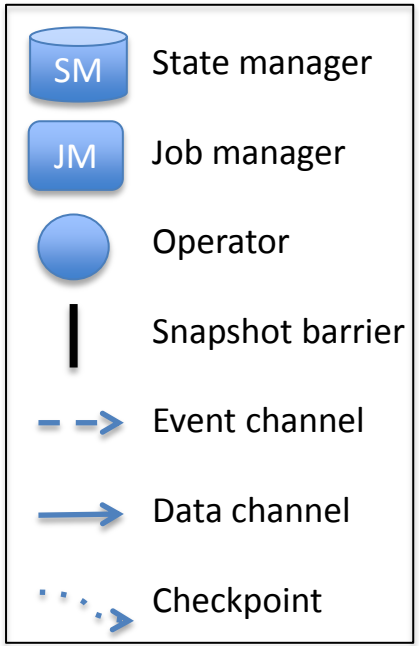
(2)

```
val rollingCorrelation = tweetsAndWarning
    .window(Time.of(30, SECONDS))
    .mapWindow(computeCorrelation _)
```

Fault tolerance

- Exactly once semantics
  - Asynchronous barrier snapshotting
  - Checkpoint barriers streamed from the sources
  - Operator state checkpointing + source backup
  - Pluggable backend for state management





Performance

- Performance optimizations
  - Effective serialization due to strongly typed topologies
  - Operator chaining (thread sharing/no serialization)
  - Different automatic query optimizations
- Competitive performance
  - ~ 1.5m events / sec / core
  - As a comparison Storm promises ~ 1m tuples / sec / node

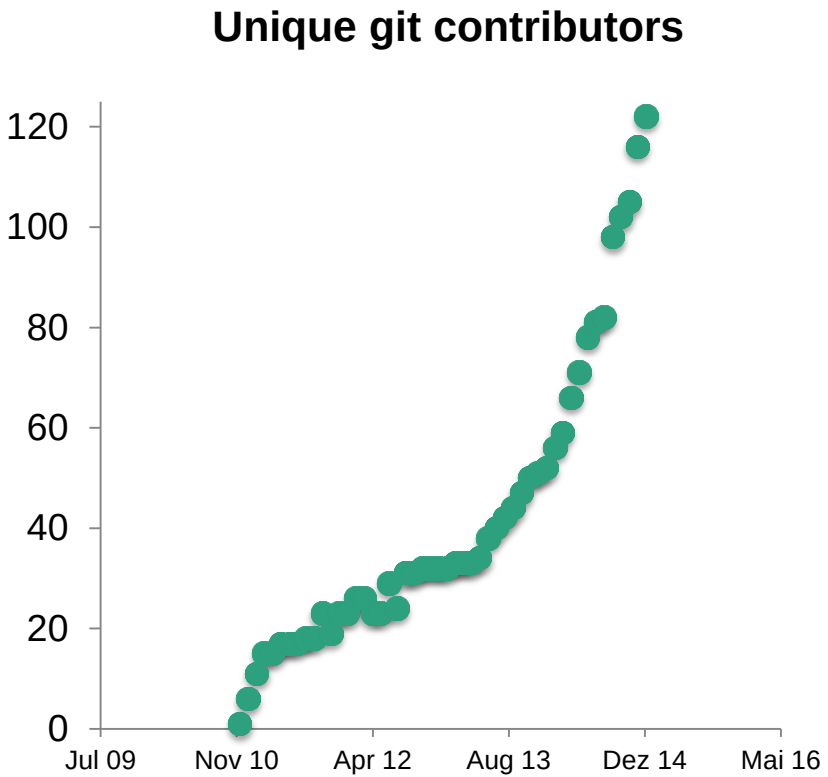
Roadmap

- Persistent, high-throughput state backend
- Job manager high availability
- Application libraries
  - General statistics over streams
  - Pattern matching
  - Machine learning pipelines library
  - Streaming graph processing library
- Integration with other frameworks
  - Zeppelin (Notebook)
  - SAMOA (Online ML)

Summary

- Flink is a use-case complete framework to unify batch and stream processing
- True streaming runtime with high-level APIs
- Flexible, data-driven windowing semantics
- Competitive performance
- We are just getting started!

Flink Community



flink.apache.org

@ApacheFlink