Real-time Stream Processing with Apache Flink

Timo Walther

Flink committer
twalthr@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

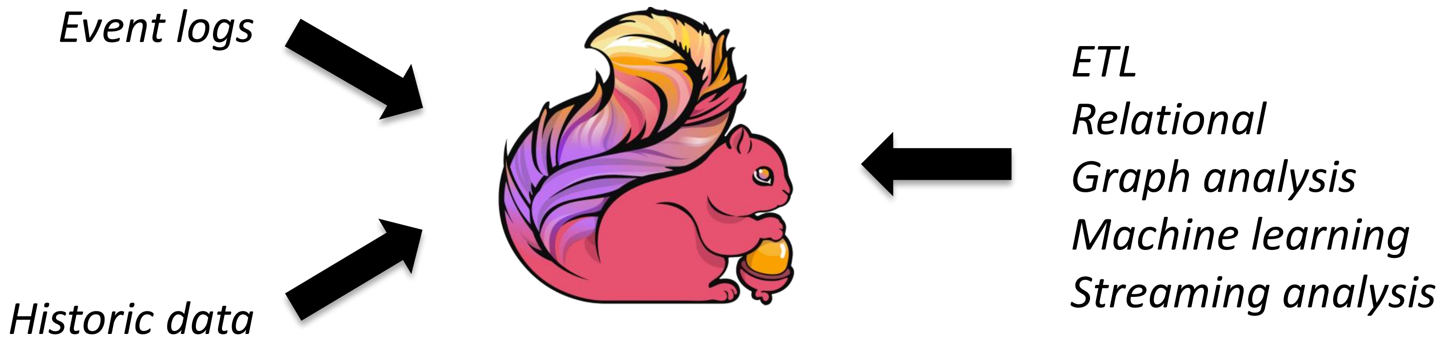
- True streaming built on top of Apache Kafka, state is first class citizen
- Slightly different stream notion, low level API

Apache Flink

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

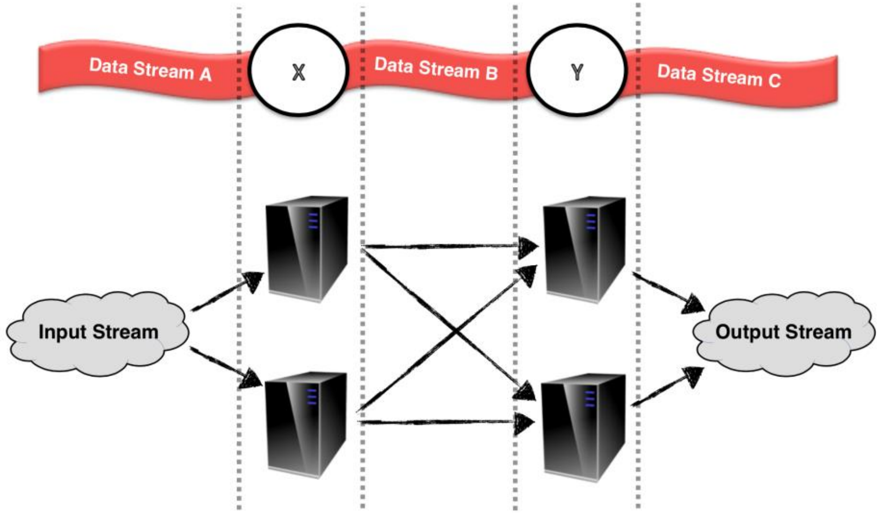
What is Flink

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



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

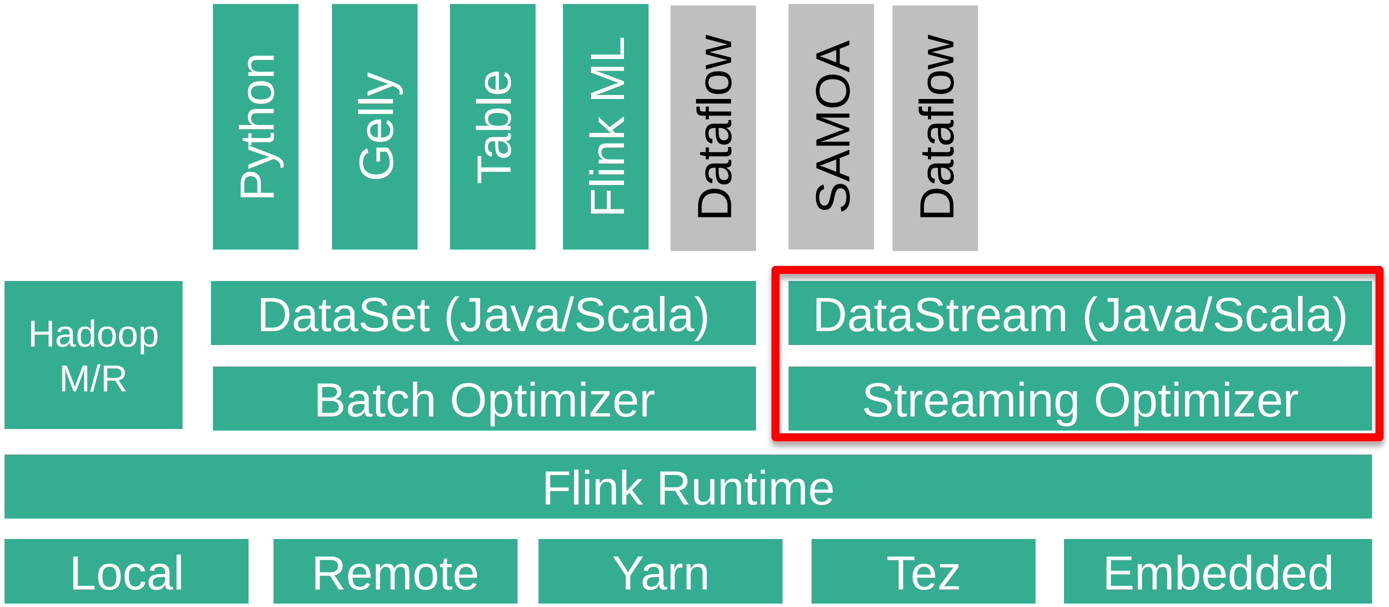


Issues

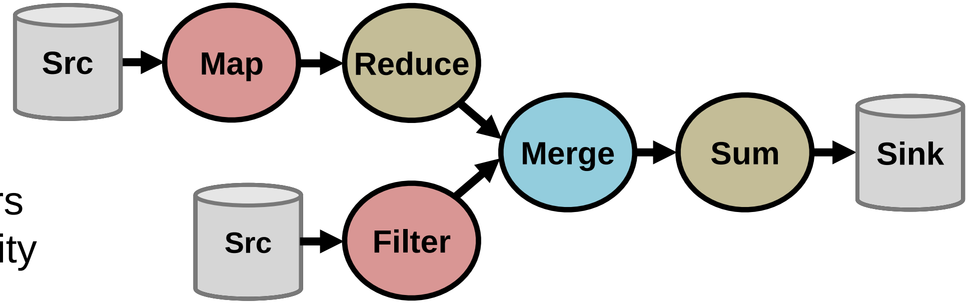
- Limited state size
- HA issue

Flink stack

**current Flink master + few PRs*



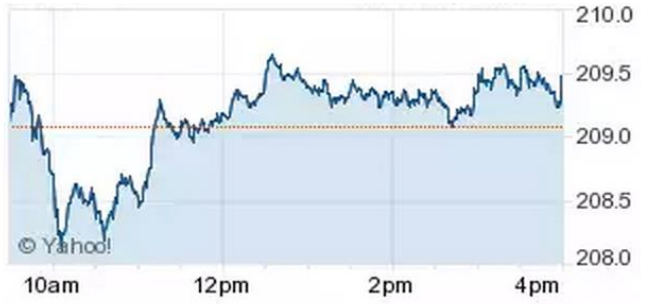
Overview of the API



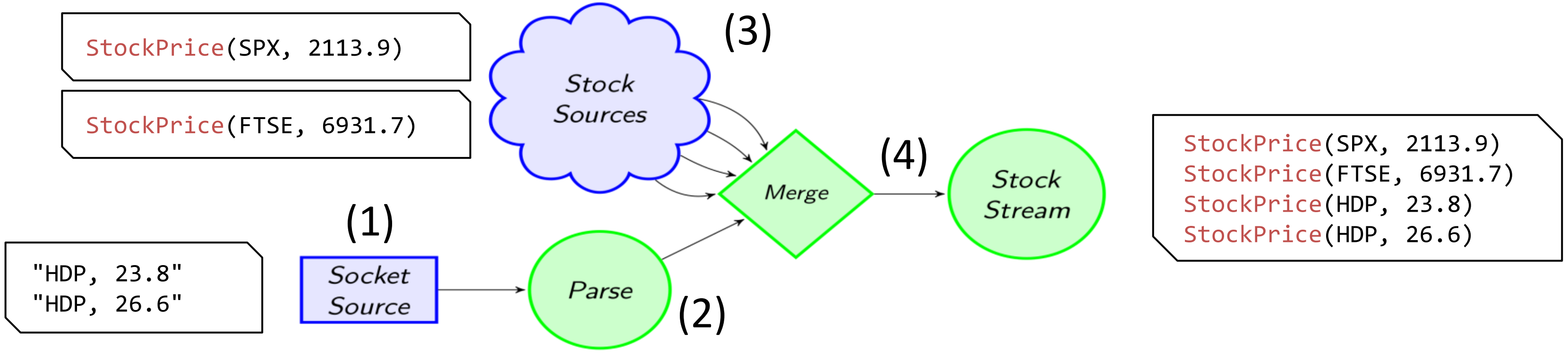
- Data stream sources
  - File system
  - Message queue connectors
  - Arbitrary source functionality
- Stream transformations
  - Basic transformations: *Map, Reduce, Filter, Aggregations*…
  - Binary stream transformations: *CoMap, CoReduce*…
  - Windowing semantics: *Policy based flexible windowing (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



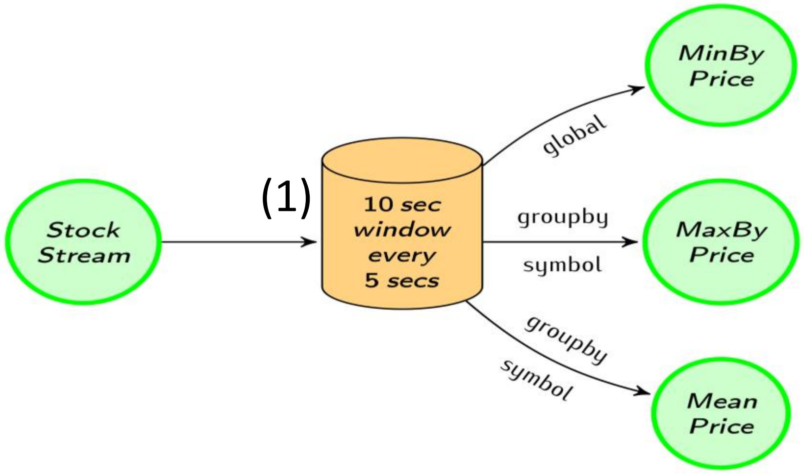
```
    case class StockPrice(symbol : String, price : Double)
    val env = StreamExecutionEnvironment.getExecutionEnvironment

(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

```
StockPrice(SPX, 2113.9)
StockPrice(FTSE, 6931.7)
StockPrice(HDP, 23.8)
StockPrice(HDP, 26.6)
```



(2)
```
StockPrice(HDP, 23.8)
```

(3)
```
StockPrice(SPX, 2113.9)
StockPrice(FTSE, 6931.7)
StockPrice(HDP, 26.6)
```

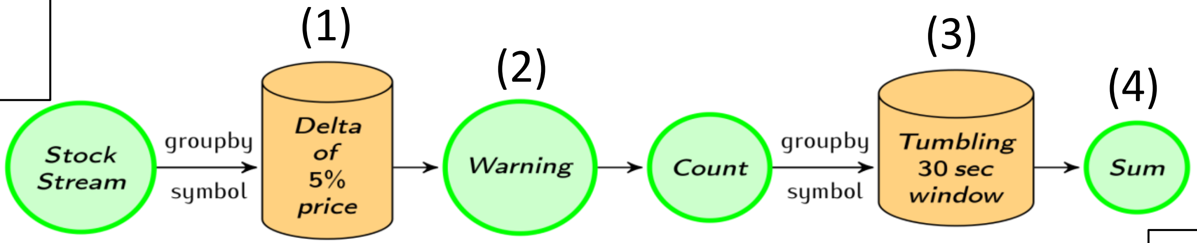
(4)
```
StockPrice(SPX, 2113.9)
StockPrice(FTSE, 6931.7)
StockPrice(HDP, 25.2)
```

```
    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

```
StockPrice(SPX, 2113.9)
StockPrice(FTSE, 6931.7)
StockPrice(HDP, 23.8)
StockPrice(HDP, 26.6)
```



```
StockPrice(HDP, 23.8)
StockPrice(HDP, 26.6)
```

```
Count(HDP, 1)
```

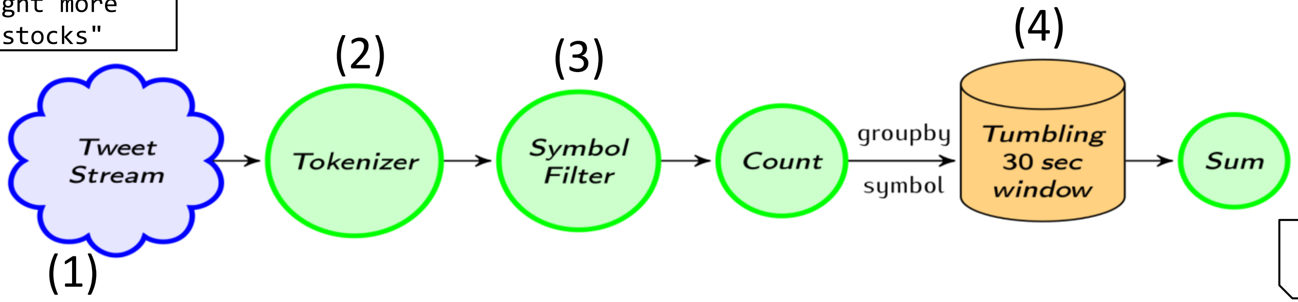
```
    case class Count(symbol : String, count : Int)

    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

```
"hdp is on the rise!"
"I wish I bought more
YHOO and HDP stocks"
```



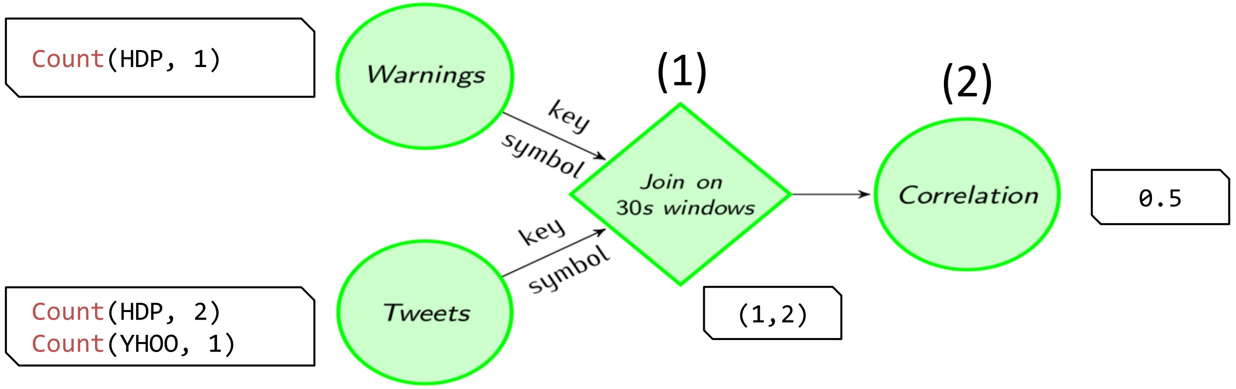
```
Count(HDP, 2)
Count(YHOO, 1)
```

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

(2) (3)
```
val mentionedSymbols = tweetStream.flatMap(tweet => tweet.split(" "))
    .map(_.toUpperCase())
    .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 _)
```

flink.apache.org
@ApacheFlink