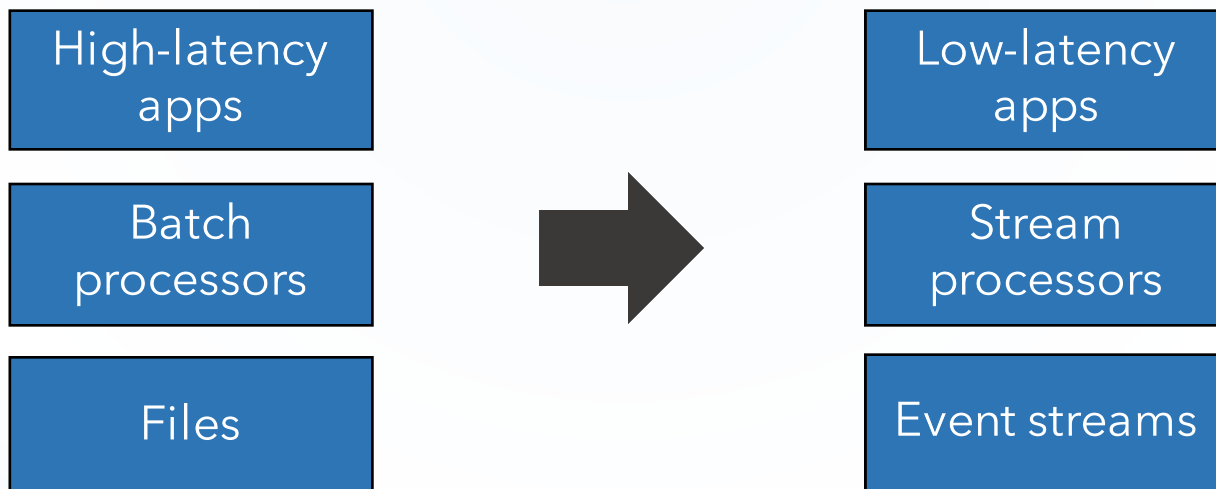
Building Stream Processing Pipelines

Gyula Fóra

gyfora@sics.se

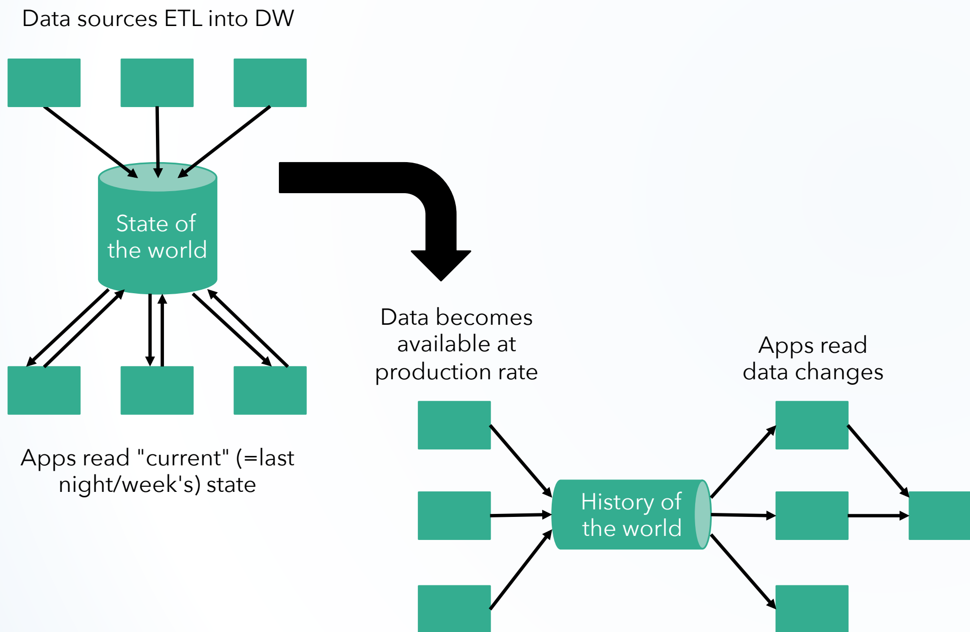
Introduction

- Stream processing is getting extremely relevant
- New open source systems triggered an application shift towards "real-time"
- We are seeing more and more complex streaming applications



What does streaming enable?

1. Data integration



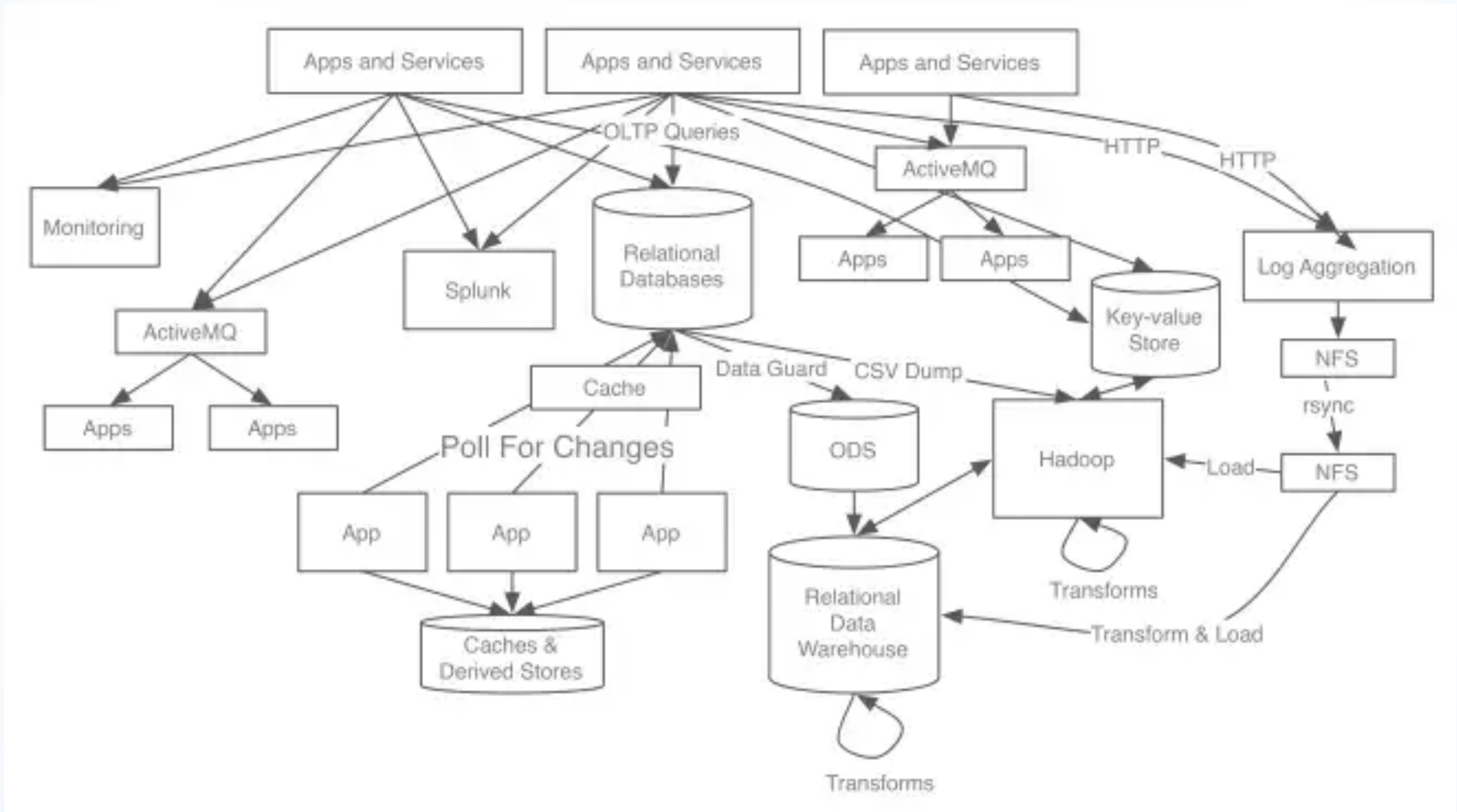
cf. Kleppmann: "Turning the DB inside out with Samza"

2. Low latency applications

- Fresh recommendations, fraud detection, etc.
- Internet of Things, intelligent manufacturing
- Results “right here, right now”

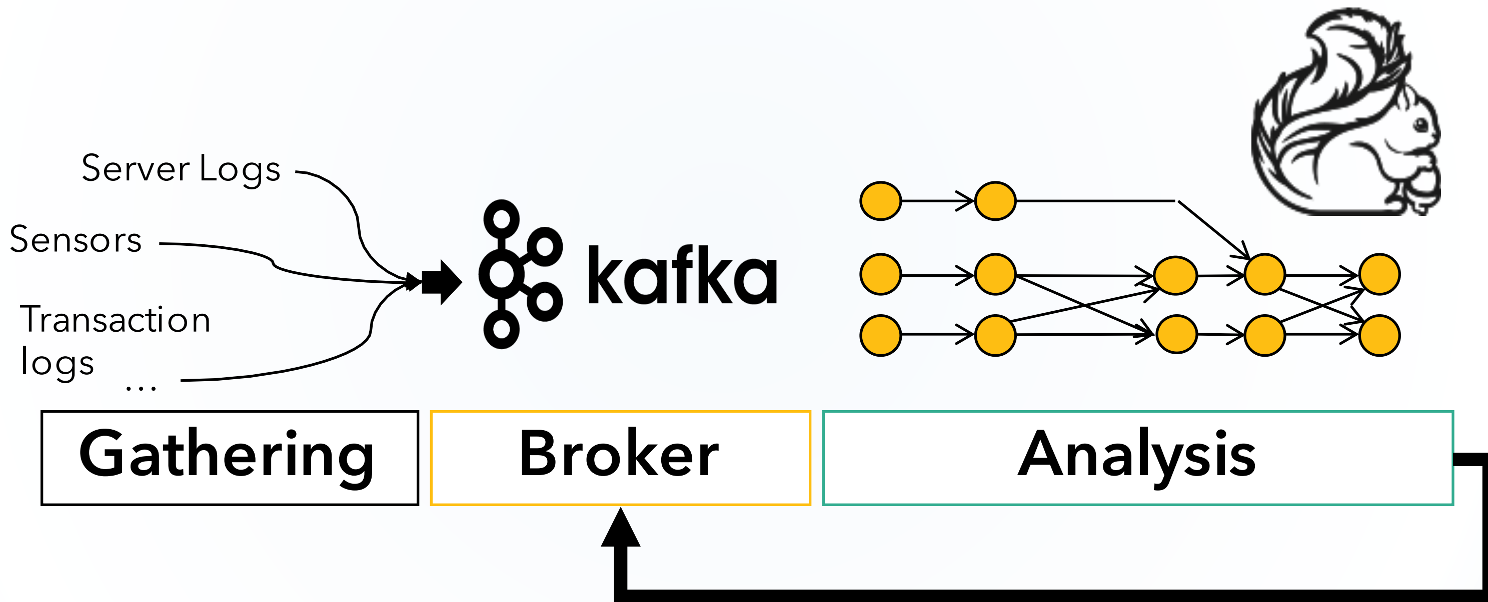
3. Batch < Streaming

What can go wrong?



http://www.confluent.io/blog/stream-data-platform-1/

Parts of a streaming infrastructure



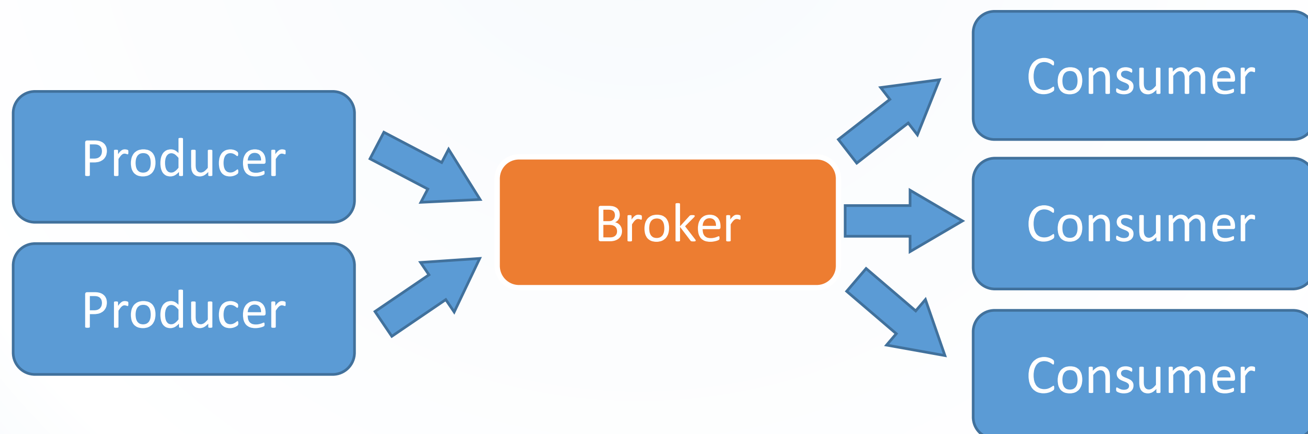
Parts of a streaming infrastructure

- Gathering
  - Collect data from different sources: Logs, Sensors etc.
  - Integration frameworks (Apache Camel)
- Broker service
  - Store the collected data and make it available for processing
  - Fault-tolerant message queues and logging services
- Stream Analysis
  - Analyze the incoming data on-the-fly
  - Feed results back into the broker for other systems

Broker Systems

Broker systems

- Middle-man between data gathering and processing
- Decouples data collection from analysis
- Different systems can analyze the data in different ways
- Stored data can be further analyzed later

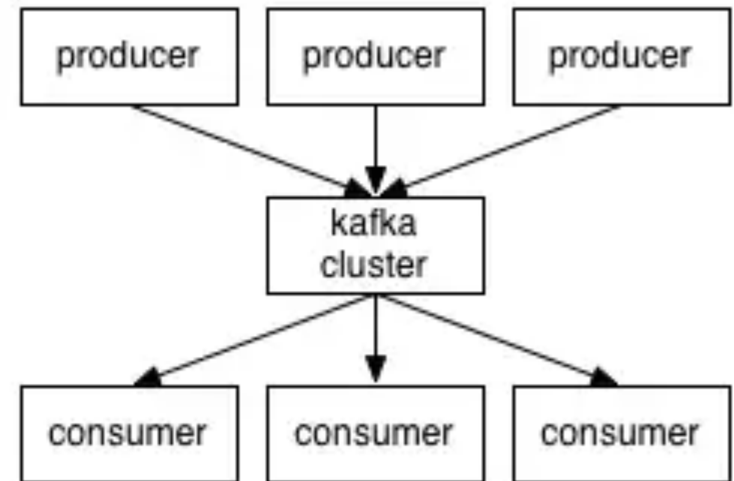


Requirements

- Persistence
  - Durability: Replay any set of messages on demand
  - This is critical to tolerate failures or modifications in other parts of the pipeline
- High-throughput, low-latency
  - As the primary data hub it needs to provide high read/write throughput with low latency
- Scalability
- Flexible APIs
- Commonly used broker systems: Kafka, RabbitMQ, ActiveMQ

Apache Kafka

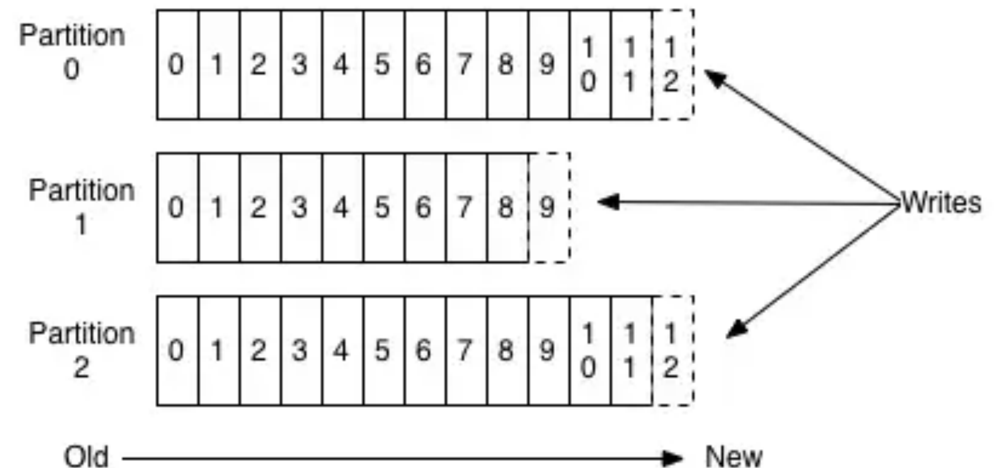
- Distributed, partitioned, replicated commit log service
- Very high read/write throughput
- Main concepts
  - Topic
  - Producer
  - Consumer (Consumer group)
- Data is partitioned so producers and consumers work in parallel



Apache Kafka

Guarantees

- Partial message ordering
- At-least-once delivery by default
- Exactly-once delivery can be implemented by the applications



Running Kafka

- Running Kafka:
  1. Start Zookeper server
  2. Start Kafka server
  3. Create Kafka topics
  4. Setup Producers/Consumers

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

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

> bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test
> bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic test --from-beginning
```

Stream processors

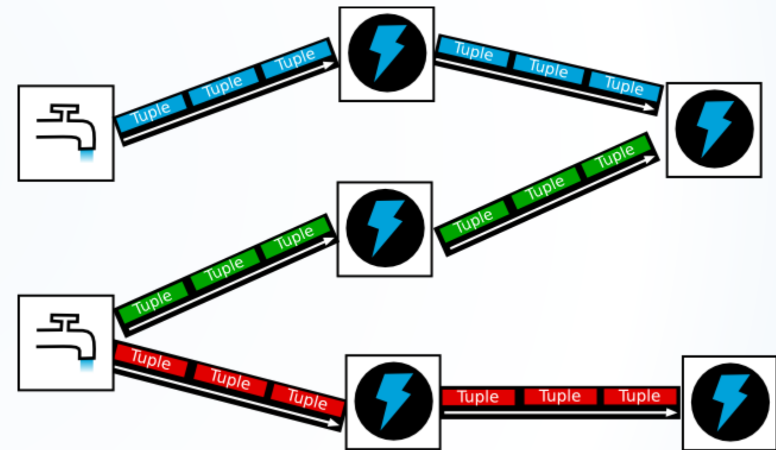
Large-scale stream processing

- Now that the data is sitting in a Broker system we need something to process it
- General requirements
  - High-throughput (keep up with the broker + more)
  - Expressivity
  - Fault-tolerance
  - Low-latency
- There are plenty of stream processing systems tailored more towards specific applications

Apache Storm

Overview

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



Storm – Word Count

Rolling word count - Standard Storm API

```
TopologyBuilder builder = new TopologyBuilder();

builder.setSpout("spout", new RandomSentenceSpout(), 5);

builder.setBolt("split", new SplitSentence(), 8).shuffleGrouping("spout");
builder.setBolt("count", new WordCount(), 12).fieldsGrouping("split", new Fields("word"));
```

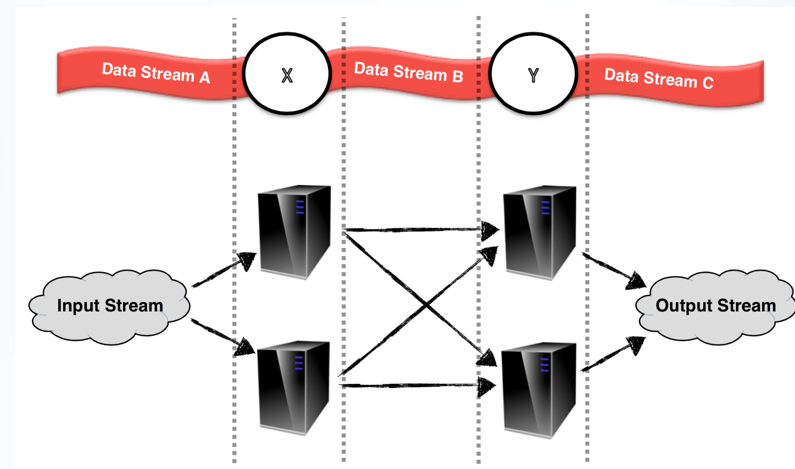
Rolling word count - Trident

```
TridentTopology topology = new TridentTopology();
TridentState wordCounts = topology.newStream("spout1", spout).parallelismHint(16).each(new Fields("sentence"),
    new Split(), new Fields("word")).groupBy(new Fields("word")).persistentAggregate(new MemoryMapState.Factory(),
    new Count(), new Fields("count")).parallelismHint(16);
```

Apache Flink

Overview

- True streaming with adjustable latency and throughput
- Rich functional API
- Fault-tolerant operator states and flexible windowing
- Exactly-once processing guarantees



Flink – Word Count

```
case class Word (word: String, frequency: Int)
```

Rolling word count

```
val lines: DataStream[String] = env.fromSocketStream(...)

lines.flatMap {line => line.split(" ")
                                .map(word => Word(word,1))}
     .groupBy("word").sum("frequency")
     .print()
```

Window word count

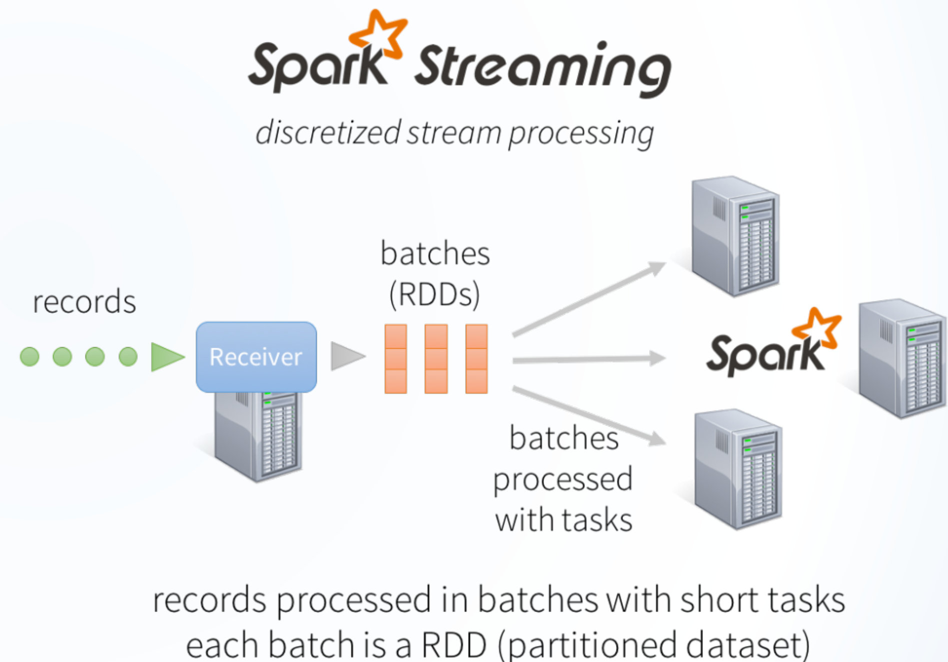
```
val lines: DataStream[String] = env.fromSocketStream(...)

lines.flatMap {line => line.split(" ")
                                .map(word => Word(word,1))}
     .window(Time.of(5,SECONDS)).every(Time.of(1,SECONDS))
     .groupBy("word").sum("frequency")
     .print()
```

Apache Spark Streaming

Overview

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



Spark – Word Count

Window word count

```
val lines = ssc.socketTextStream(args(0), args(1).toInt, StorageLevel.MEMORY_AND_DISK_SER)
val words = lines.flatMap(_.split(" "))
val wordCounts = words.map(x => (x, 1)).reduceByKey(_ + _)
wordCounts.print()
```

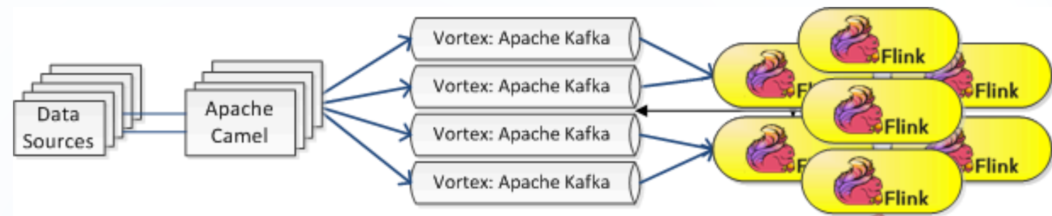
Rolling word count (kind of)

```
val lines = ssc.socketTextStream(args(0), args(1).toInt)
val words = lines.flatMap(_.split(" "))
val wordDstream = words.map(x => (x, 1))

// Update the cumulative count using updateStateByKey
// This will give a Dstream made of state (which is the cumulative count of the words)
val stateDstream = wordDstream.updateStateByKey[Int](newUpdateFunc,
  new HashPartitioner (ssc.sparkContext.defaultParallelism), true, initialRDD)
stateDstream.print()
```

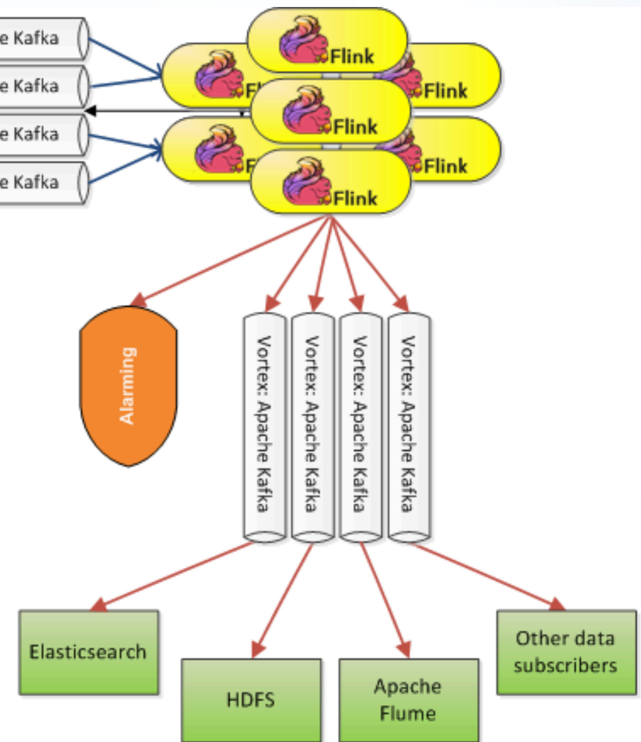
Putting it all together

Streaming pipeline in action



Streaming infrastructure at Bouygues Telecom

- Network and subscriber data gathered
- Added to Broker in raw format
- Transformed and analyzed by streaming engine
- Stored back for further processing
- Results processed by other systems

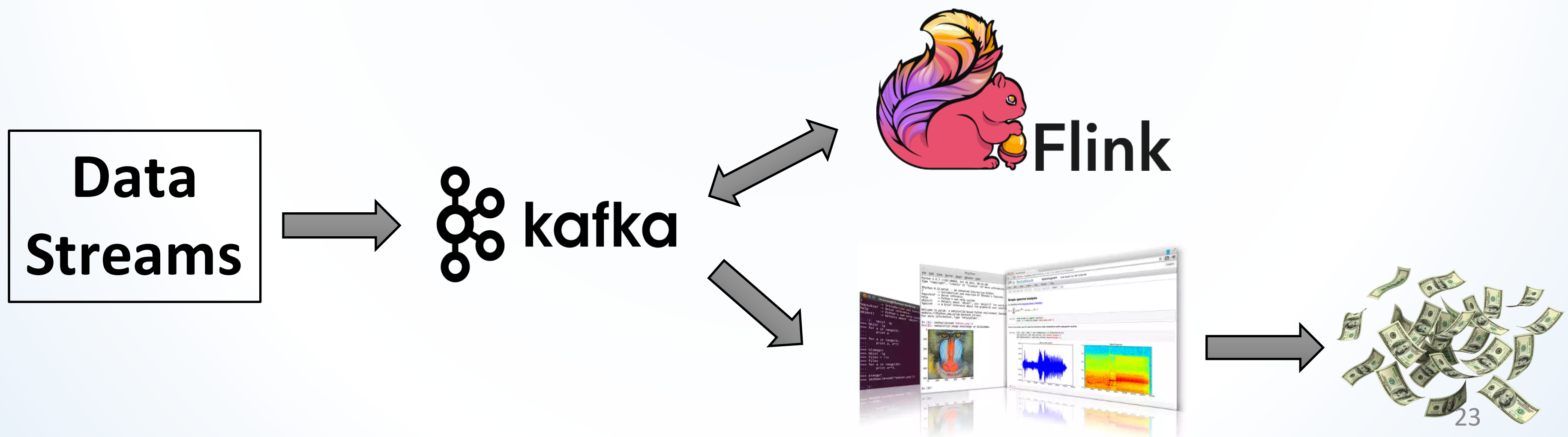


Read more:
http://data-artisans.com/flink-at-bouygues.html

Let's start with something simple

Interactive analysis using Flink, Kafka and Python

1. Load stream into Kafka
2. Create a Flink job to process the data
3. Store results back into Kafka
4. Analyze results using Python notebook



Demo

https://github.com/gyfora/summer-school

What have we learnt?

- Building a proper streaming infrastructure is not trivial (but it’s certainly possible)
- Stream processors are just part of the big picture, other components are critical as well
- There is no single system to provide an end-to-end solution
  - Mix and match the different components

Thank you!