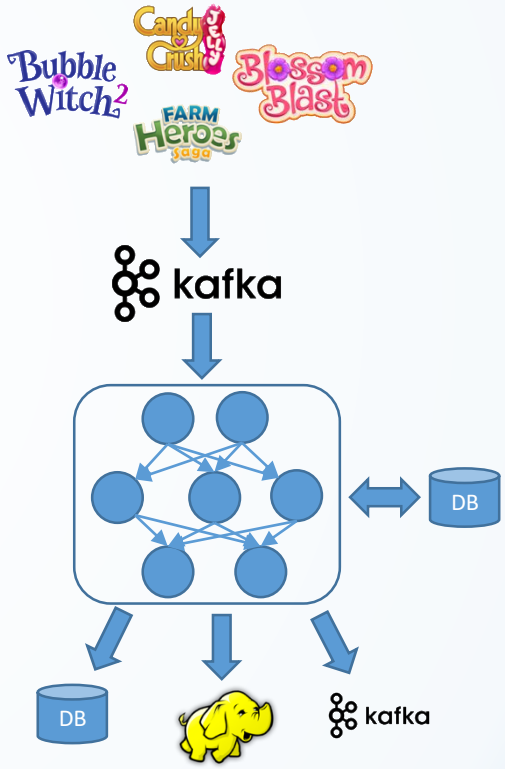
Streaming training

Márton Balassi
mbalassi@apache.org
@MartonBalassi

Stream processing by example

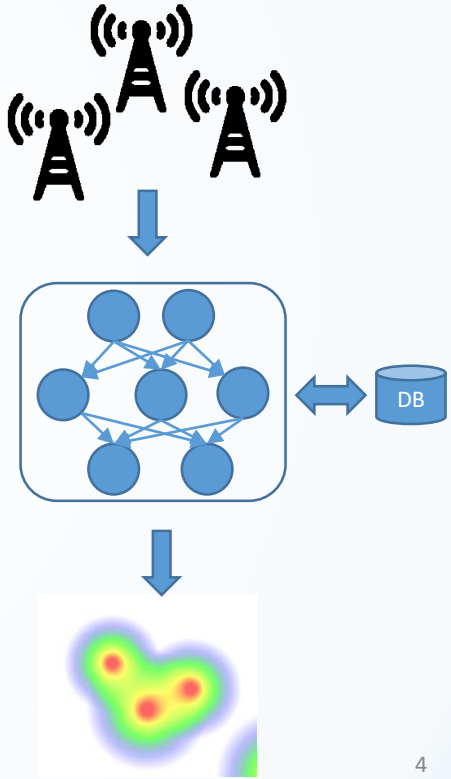
Real-Time Player Statistics

- Compute real-time, queryable statistics
  - Billions of events / day
  - Millions of active users / day
- State quickly grows beyond memory
- Complex event processing logic
- Strong consistency requirements



Real-time dashboard of telco network

- Example query: Download speed heatmap of premium users in the last 5 minutes
- Dependant on ~1 TB slowly changing enrichment data
- Multi GB/s input rate
- Some of the use cases require complex windowing logic



Open source stream processors

Apache Streaming Landscape

Flink

Spark

STORM

samza

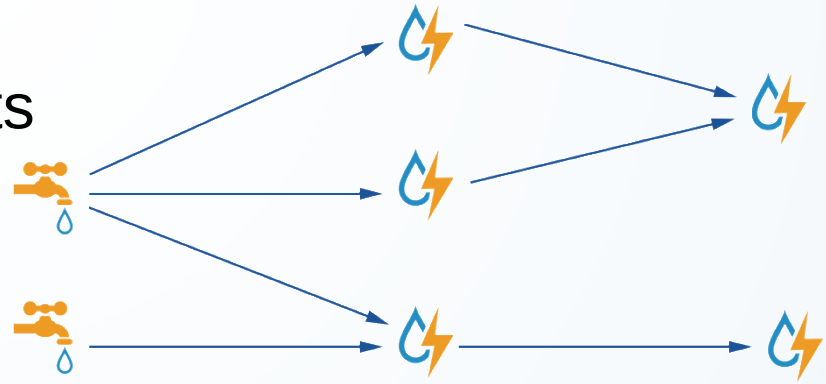
kafka

Apache Storm

- Pioneer of real-time analytics
- Distributed dataflow abstraction with low-level control
- Time windowing and state introduced recently

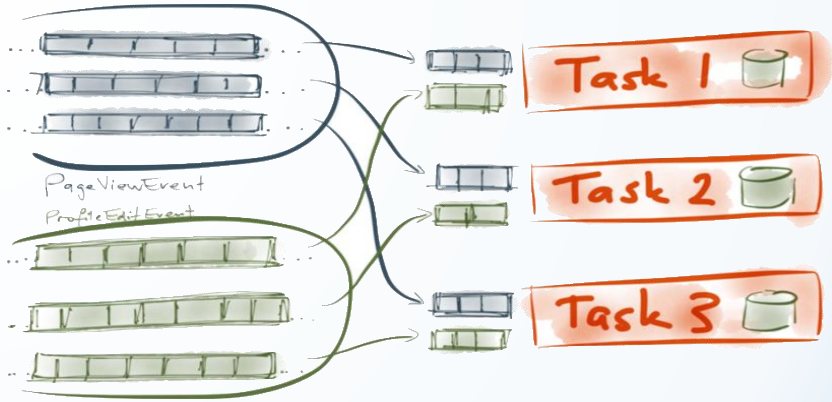
When to use Storm

- Very low latency requirements
- No need for advanced state/windowing
- At-least-once is acceptable



Apache Samza

- Builds heavily on Kafka's log based philosophy
- Pluggable components, but runs best with Kafka
- Scalable operator state with RocksDB
- Basic time windowing



When to use Samza

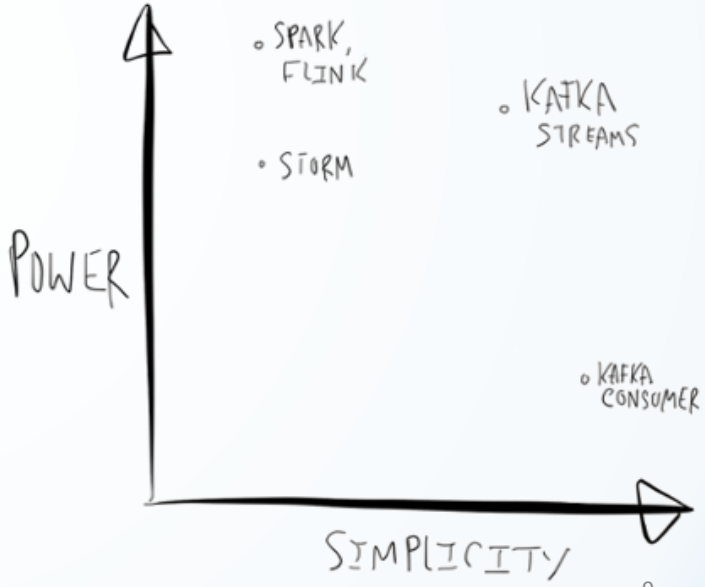
- Join streams with large states
- At-least-once is acceptable

Kafka Streams

- Streaming *library* on top of Apache Kafka
- Similar features to Samza but nicer API
- Big win for operational simplicity

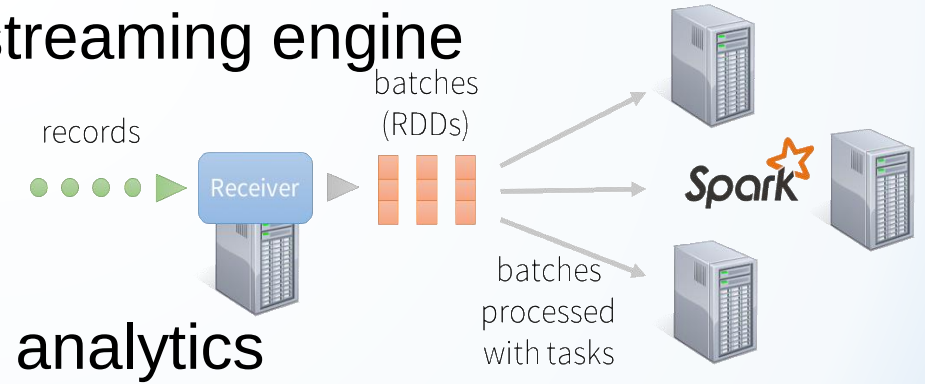
<u>When to use Kafka Streams</u>

- Kafka based data infrastructure
- Join streams with large states
- At-least-once is acceptable



Apache Spark

- Unified batch and stream processing over a batch runtime
- Good integration with batch programs
- Lags behind recent streaming advancements but evolving quickly
- Spark 2.0 comes with new streaming engine



When to use Spark

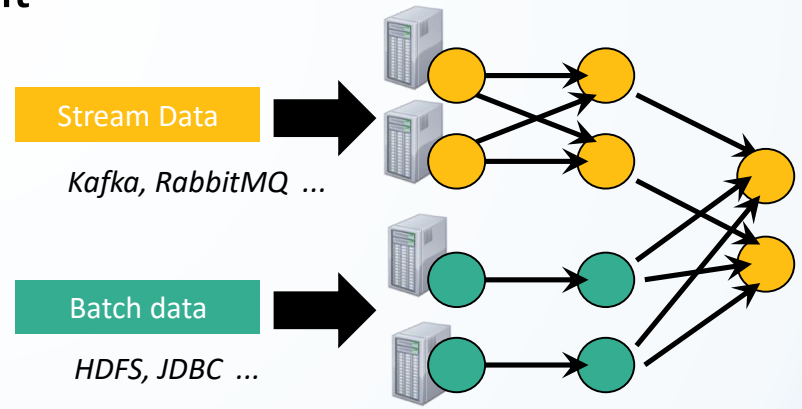
- Simpler data exploration
- Combine with (Spark) batch analytics
- Medium latency is acceptable

Apache Flink

- Unified batch and stream processing over dataflow engine
- Leader of open source streaming innovation
- Highly flexible and robust stateful and windowing computations
- Savepoints for state management

<u>When to use Flink</u>

- Advanced streaming analytics
- Complex windowing/state
- Need for high TP - low latency

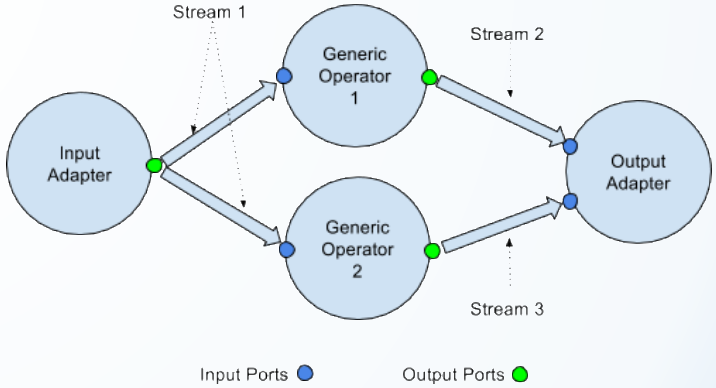


Apache Apex

- Native streaming engine built natively on YARN
- Stateful operators with checkpointing to HDFS
- Advanced partitioning support with locality optimizations

When to use Apex

- Advanced streaming analytics
- Very low latency requirements
- Need extensive operator library

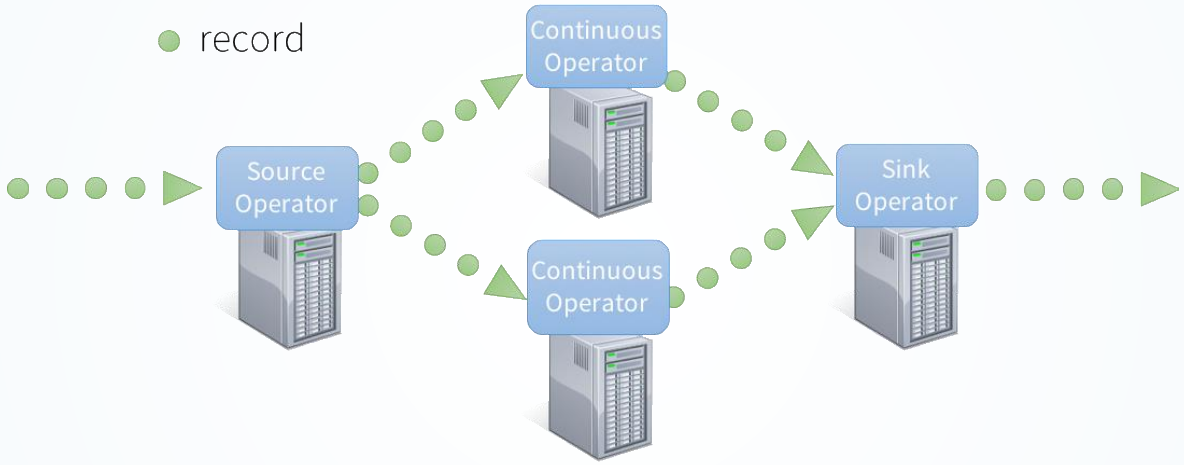


System comparison

	STORM	Spark	samza	Flink	APEX
Model	Native	Micro-batch	Native	Native	Native
API	Compositional	Declarative	Compositional	Declarative	Compositional
Fault tolerance	Record ACKs	RDD-based	Log-based	Checkpoints	Checkpoints
Guarantee	At-least-once	Exactly-once	At-least-once	Exactly-once	Exactly-once
State	Stateful operators	State as DStream	Stateful operators	Stateful operators	Stateful operators
Windowing	Time based	Time based	Time based	Flexible	Time based
Latency	Very-Low	High	Low	Low	Very-Low
Throughput	Medium	Very-High	High	Very-High	Very-High

Under the hood

Native Streaming



records processed one at a time

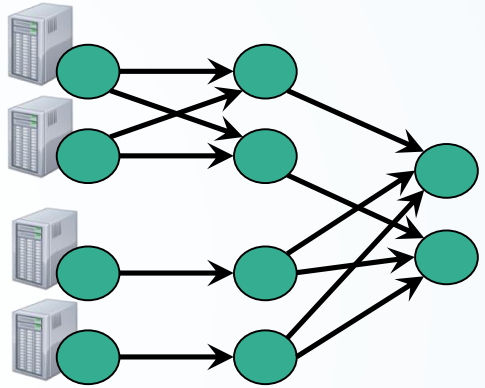
STORM

samza

APEX

Distributed dataflow runtime

- Long standing operators
- Pipelined execution
- Usually possible to create cyclic flows



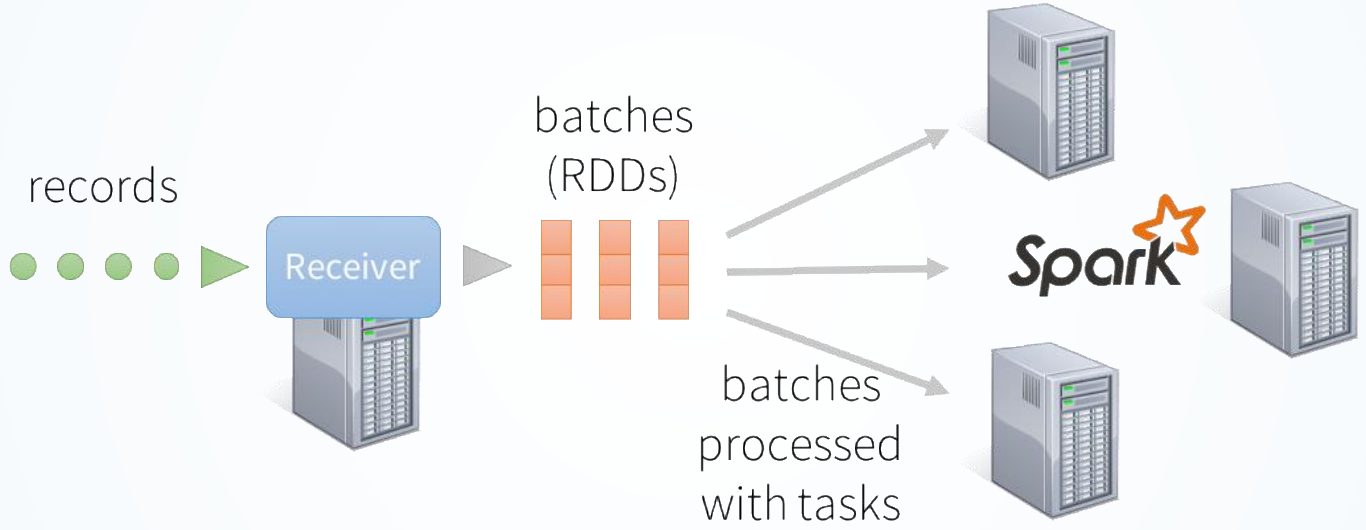
Pros

- Full expressivity
- Low-latency execution
- Stateful operators

Cons

- Fault-tolerance is hard
- Throughput may suffer
- Load balancing is an issue

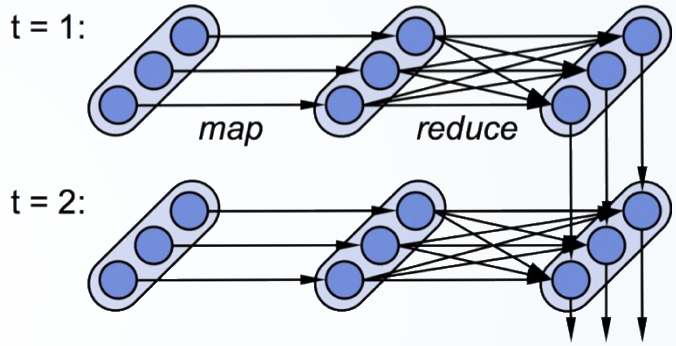
Micro-batching



records processed in batches with short tasks

Micro-batch runtime

- Computation broken down to time intervals
- Load aware scheduling
- Easy interaction with batch



Pros

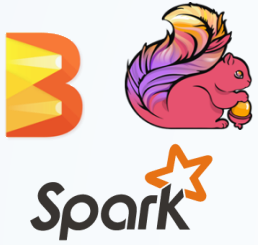
- Easy to reason about
- High-throughput
- FT comes for “free”
- Dynamic load balancing

Cons

- Latency depends on batch size
- Limited expressivity
- Stateless by nature

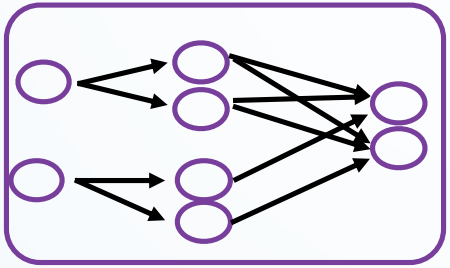
Programming models

Hierarchy of Streaming APIs



DataStream
DStream

Spout, Consumer, Bolt, Task, Topology



Declarative

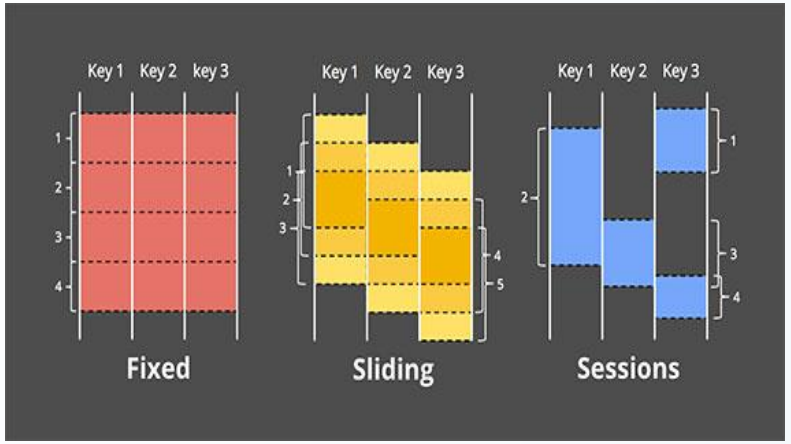
- Transformations, abstract operators
- For both engineers and data analysts
- Allows (some) automatic query optimization

Compositional

- Direct access to the execution graph
- Suitable for engineers
- Fine grained access but lower productivity

Apache Beam

- One API to rule them all: combined batch and streaming analytics
- Open sourced by Google, based on DataFlow
- Advanced windowing
- Runners on different systems
  - Google Cloud
  - Flink
  - Spark
  - (Others to follow…)
- Useful for benchmarking?



Apache Beam

- *What* results are calculated?
- *Where* in event time are results calculated?
- *When* in processing time are results materialized?
- *How* do refinements of results relate?

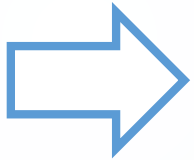
	Beam Model	Google Cloud Dataflow	Apache Flink	Apache Spark
ParDo	✓	✓	✓	✓
GroupByKey	✓	✓	✓	~
Flatten	✓	✓	✓	✓
Combine	✓	✓	✓	✓
Composite Transforms	✓	~	~	~
Side Inputs	✓	✓	~	~
Source API	✓	✓	~	✓
Aggregators	~	~	~	~
Keyed State	×	×	×	×

	Beam Model	Google Cloud Dataflow	Apache Flink	Apache Spark
Global windows	✓	✓	✓	✓
Fixed windows	✓	✓	✓	~
Sliding windows	✓	✓	✓	×
Session windows	✓	✓	✓	×
Custom windows	✓	✓	✓	×
Custom merging windows	✓	✓	✓	×
Timestamp control	✓	✓	✓	×

Counting words…

WordCount

storm dublin flink
apache storm spark
streaming samza storm
flink apache flink
bigdata storm
flink streaming



(storm, 4)
(dublin, 1)
(flink, 4)
(apache, 2)
(spark, 1)
(streaming, 2)
(samza, 1)
(bigdata, 1)

Storm

Assembling the topology

```
TopologyBuilder builder = new TopologyBuilder();

builder.setSpout("spout", new SentenceSpout(), 5);
builder.setBolt("split", new Splitter(), 8).shuffleGrouping("spout");
builder.setBolt("count", new Counter(), 12)
       .fieldsGrouping("split", new Fields("word"));
```

Rolling word count bolt

```
public class Counter extends BaseBasicBolt {
    Map<String, Integer> counts = new HashMap<String, Integer>();

    public void execute(Tuple tuple, BasicOutputCollector collector) {
        String word = tuple.getString(0);
        Integer count = counts.containsKey(word) ? counts.get(word) + 1 : 1;
        counts.put(word, count);
        collector.emit(new Values(word, count));
    }
}
```

Samza

Rolling word count task

```
public class WordCountTask implements StreamTask {
  private KeyValueStore<String, Integer> store;

  public void process(        IncomingMessageEnvelope
envelope,
          MessageCollector collector,
                    TaskCoordinator coordinator) {
          String word = envelope.getMessage();
          Integer count = store.get(word);
          if(count == null){count = 0;}
          store.put(word, count + 1);
          collector.send(new OutgoingMessageEnvelope(new
SystemStream("kafka", ”wc"), Tuple2.of(word, count)));
          }
}
```

Apex

```
@ApplicationAnnotation(name="MyFirstApplication")
public class Application implements StreamingApplication
{
  @Override
  public void populateDAG(DAG dag, Configuration conf)
  {
    LineReader lineReader = dag.addOperator("input", new LineReader());
    Parser parser = dag.addOperator("parser", new Parser());
    UniqueCounter counter = dag.addOperator("counter", new UniqueCounter());
    ConsoleOutputOperator cons = dag.addOperator("console", new ConsoleOutputOperator());
    dag.addStream("lines", lineReader.output, parser.input);
    dag.addStream("words", parser.output, counter.data);
    dag.addStream("counts", counter.count, cons.input);
  }
}
```

Flink

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

Rolling word count

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

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

Window word count

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

lines.flatMap {line => line.split(" ")
                       .map(word => Word(word,1))}
     .keyBy("word").
     .timeWindow(Time.seconds(5))
     .sum("frequency").print()
```

Spark

Window word count

```
val lines = env.fromSocketStream(...)

val words = lines.flatMap(line => line.split(" "))
                 .map(word => (word,1))
val wordCounts = words.reduceByKey(_ + _)
wordCounts.print()
```

Rolling word count (new feature ☺)

```
val func = (word: String, one: Option[Int], state: State[Int]) => {
      val sum = one.getOrElse(0) + state.getOption.getOrElse(0)
      val output = (word, sum)
      state.update(sum)
      output
    }
val stateDstream = wordDstream.mapWithState(
      StateSpec.function(func).initialState(initialRDD))
stateDstream.print()
```

Beam

Window word count (minimalistic version)

```
PCollection<String> windowedLines = input
          .apply(Window.<String>into(
                    FixedWindows.of(Duration.standardMinutes(5))));

PCollection<KV<String, Long>> wordCounts = windowedLines
          .apply(ParDo.of(new DoFn<String, String>() {
                        @Override
                        public void processElement(ProcessContext c) {
                          for (String word : c.element().split("[^a-zA-Z']+")){
                            if (!word.isEmpty()) {
                              c.output(word);
                            }
                          }
                        }
                      }))
.apply(Count.<String>perElement());
```

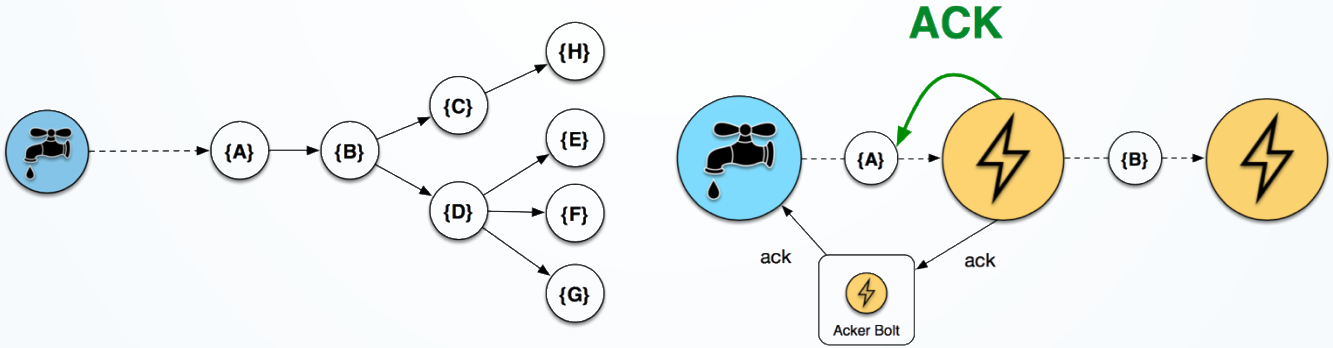
Fault tolerance and stateful processing

Fault tolerance intro

- Fault-tolerance in streaming systems is inherently harder than in batch
  - Can’t just restart computation
  - State is a problem
  - Fast recovery is crucial
  - Streaming topologies run 24/7 for a long period
- Fault-tolerance is a complex issue
  - No single point of failure is allowed
  - Guaranteeing input processing
  - Consistent operator state
  - Fast recovery
  - At-least-once vs Exactly-once semantics

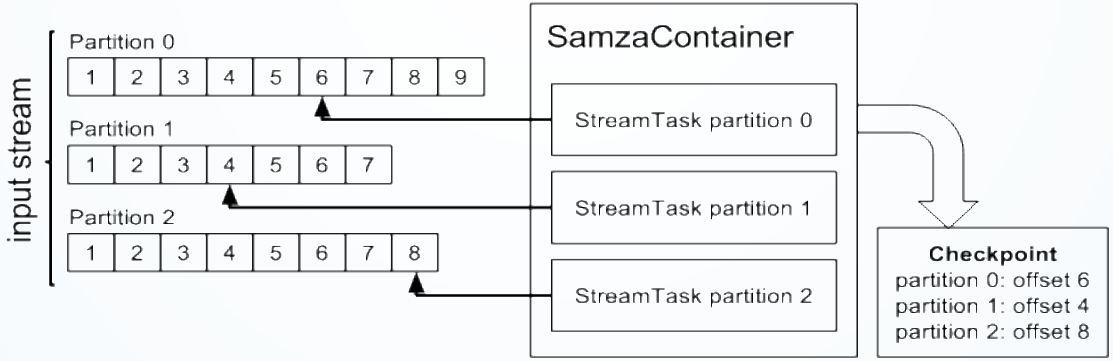
Storm record acknowledgements

- Track the lineage of tuples as they are processed (anchors and acks)
- Special “acker” bolts track each lineage DAG (efficient xor based algorithm)
- Replay the root of failed (or timed out) tuples



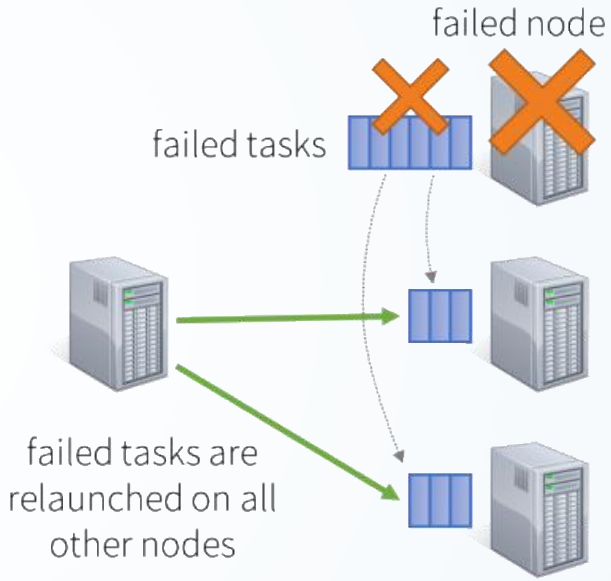
Samza offset tracking

- Exploits the properties of a durable, offset based messaging layer
- Each task maintains its current offset, which moves forward as it processes elements
- The offset is checkpointed and restored on failure (some messages might be repeated)



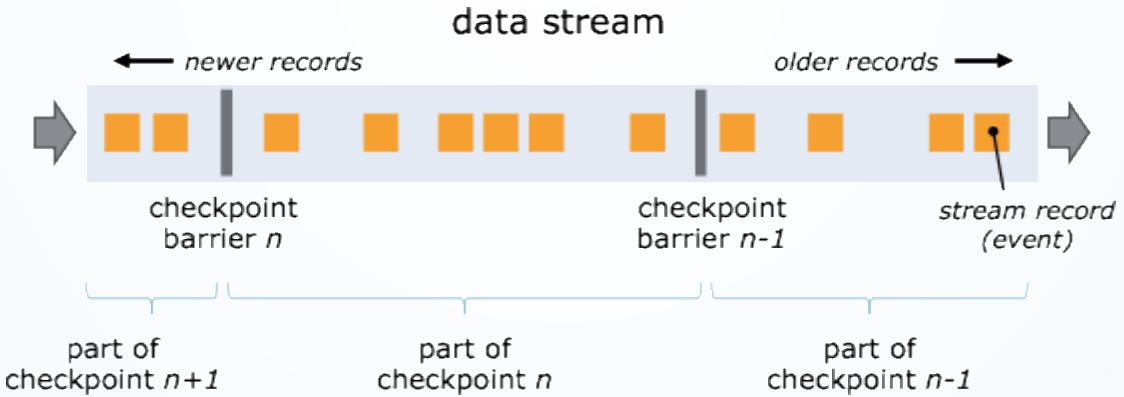
Spark RDD recomputation

- Immutable data model with repeatable computation
- Failed RDDs are recomputed using their lineage
- Checkpoint RDDs to reduce lineage length
- Parallel recovery of failed RDDs
- Exactly-once semantics



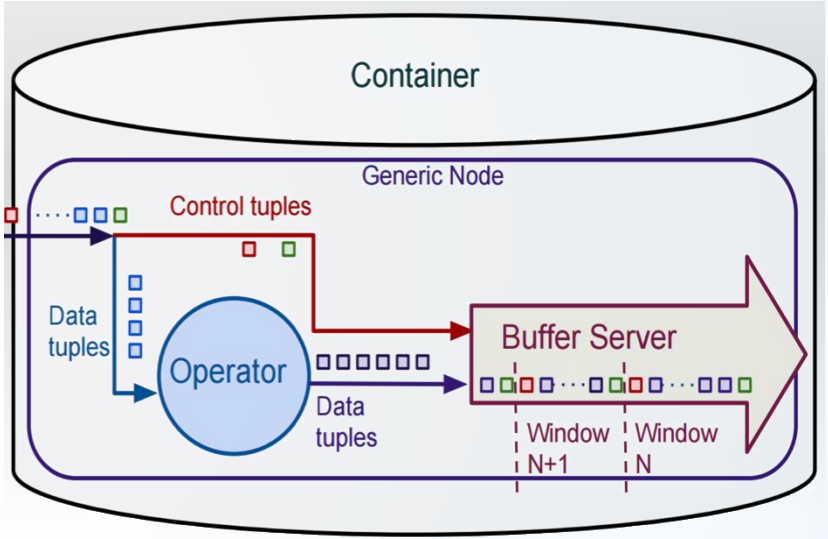
Flink state checkpointing

- Consistent global snapshots with exactly-once semantics
- Algorithm designed for stateful dataflows (minimal runtime overhead)
- Pluggable state backends: Memory, FS, RocksDB, MySQL…



Apex state checkpointing

- Algorithms similar to Flink's but also buffers output windows
- Larger memory overhead but faster, granular recovery
- Pluggable checkpoint backend, HDFS by default



Performance

How much does all this matter?

The winners of last year's Twitter hack week managed to reduce the resources needed for a specific job by 99%. [1]

There are many recent benchmarks out there

- Storm, Flink & Spark by Yahoo [2]
- Apex by DataTorrent [3,4]
- Flink by data Artisans [1,5]

[1] http://data-artisans.com/extending-the-yahoo-streaming-benchmark

[2] https://yahooeng.tumblr.com/post/135321837876/benchmarking-streaming-computation-engines-at

[3] https://www.datatorrent.com/blog/blog-implementing-linear-road-benchmark-in-apex/

[4] https://www.datatorrent.com/blog/blog-apex-performance-benchmark/

[5] http://data-artisans.com/high-throughput-low-latency-and-exactly-once-stream-processing-with-apache-flink/

Next steps for streaming

- Dynamic scaling (with state)
- Rolling upgrades
- Better state handling
- More Beam runners
- Libraries: CEP, ML
- Better batch integration

Closing

Summary

- Streaming systems are gaining popularity with many businesses migrating some of their infrastructure
- The open source space sees a lot of innovation
- When choosing an application consider your specific use cases, do not just follow the herd
- We have a recommended reading section :)

Thank you!

Recommended reading

Apache Beam

- http://beam.incubator.apache.org/beam/capability/2016/03/17/capability-matrix.html
- https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-101
- https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-102

Apache Spark Streaming

- https://databricks.com/blog/2016/02/01/faster-stateful-stream-processing-in-spark-streaming.html
- http://www.slideshare.net/databricks/2016-spark-summit-east-keynote-matei-zaharia

Apache Flink

- http://flink.apache.org/news/2015/12/04/Introducing-windows.html
- http://data-artisans.com/flink-1-0-0/
- http://data-artisans.com/how-apache-flink-enables-new-streaming-applications/

Apache Storm

- https://community.hortonworks.com/articles/14171/windowing-and-state-checkpointing-in-apache-storm.html
- https://yahooeng.tumblr.com/post/135321837876/benchmarking-streaming-computation-engines-at

Samza / Kafka Streams

- http://docs.confluent.io/2.1.0-alpha1/streams/architecture.html
- http://www.confluent.io/blog/introducing-kafka-streams-stream-processing-made-simple
- http://docs.confluent.io/2.1.0-alpha1/streams/index.html
- http://www.slideshare.net/edibice/extremely-low-latency-web-scale-fraud-prevention-with-apache-samza-kafka-and-friends
- http://radar.oreilly.com/2014/07/why-local-state-is-a-fundamental-primitive-in-stream-processing.html

Apache Apex

- http://docs.datatorrent.com/application_development/#apache-apex-platform-overview
- http://docs.datatorrent.com/application_development/#fault-tolerance
- https://github.com/apache/incubator-apex-malhar/tree/master/demos
- https://www.datatorrent.com/introducing-apache-apex-incubating/

List of Figures (in order of usage)

- https://upload.wikimedia.org/wikipedia/commons/thumb/2/2a/CPT-FSM-abcd.svg/326px-CPT-FSM-abcd.svg.png
- https://storm.apache.org/images/topology.png
- *https://databricks.com/wp-content/uploads/2015/07/image11-1024x655.png*
- *https://databricks.com/wp-content/uploads/2015/07/image21-1024x734.png*
- https://people.csail.mit.edu/matei/papers/2012/hotcloud_spark_streaming.pdf, page 2.
- http://www.slideshare.net/ptgoetz/storm-hadoop-summit2014, page 69-71.
- http://samza.apache.org/img/0.9/learn/documentation/container/checkpointing.svg
- https://databricks.com/wp-content/uploads/2015/07/image41-1024x602.png
- https://storm.apache.org/documentation/images/spout-vs-state.png
- http://samza.apache.org/img/0.9/learn/documentation/container/stateful_job.png