Understanding Big Data Streaming and Apache Flink

Dr. Vladimir Bacvanski

vladimir.bacvanski@scispike.com

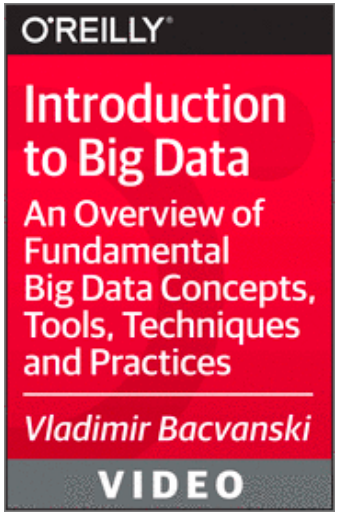
 @OnSoftware

 https://www.linkedin.com/in/vladimirbacvanski

With contributions from:
@StephanEwen
@kostas_tzoumas

Dr. Vladimir Bacvanski

- Founder of SciSpike, a development, consulting, and training firm
- Passionate about software and data
- PhD in computer science RWTH Aachen, Germany
- Architect, consultant, mentor



SciSpike

- Custom development: Scalable Web and IoT systems
- Training and mentoring in Big Data, Scala, node.js, software architecture

@OnSoftware

https://www.linkedin.com/in/vladimirbacvanski

SciSpike Training on Reactive Systems & Big Data

- **Developing with Scala**

  Transform a Java developer into a Scala developer

  - Get productive in Scala quickly
  - Non-threatening, pragmatic Functional Programming

- **Scala Reactive Systems: Futures, Akka, Spray / akka-http**

  Concurrency, Microservices, Reactive Systems

- **Big Data with Spark and Scala**

  Batch and streaming processing with intro to ML and graph processing

- **Big Data with Flink**

  In planning…

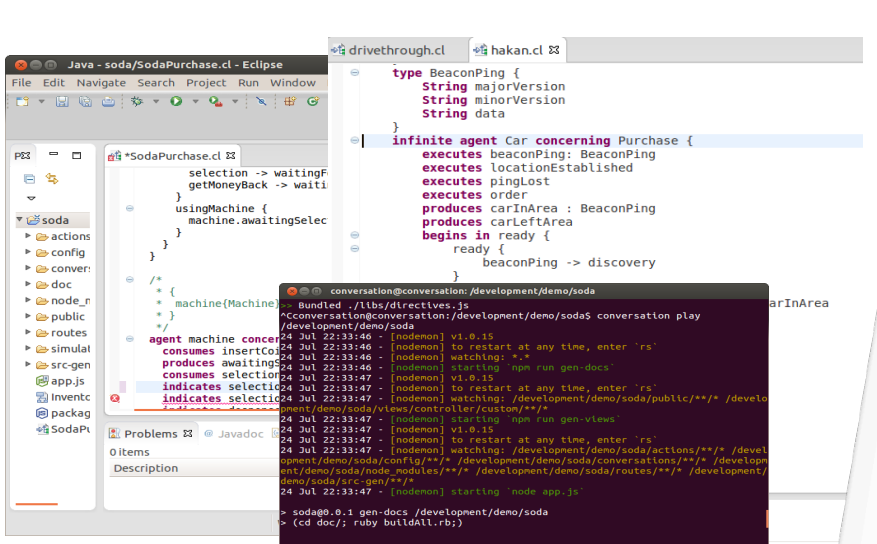
Our courses are often customized to match the application area of the client

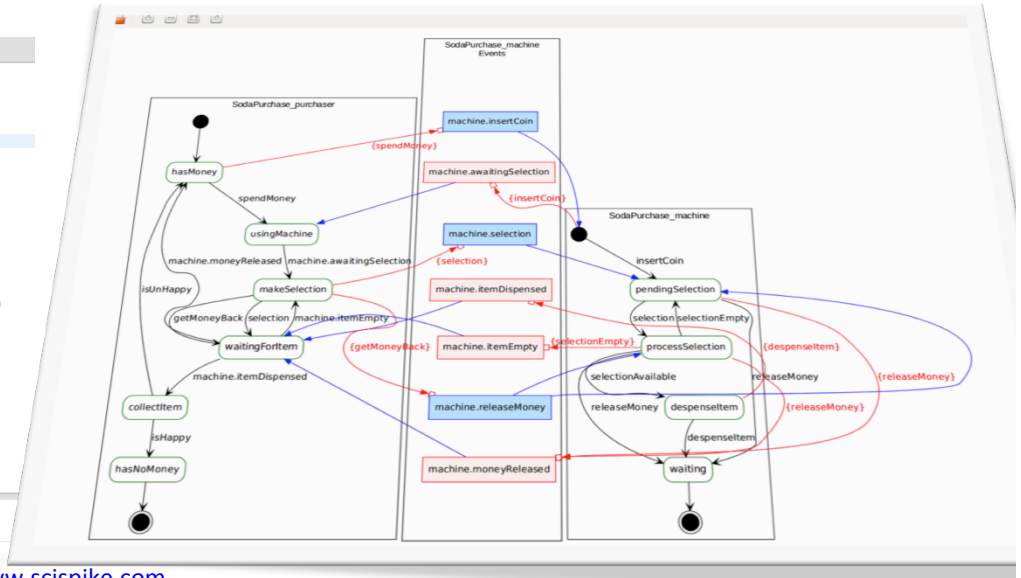
Announcement: Yaktor – Agent Platform for node.js

- **http://yaktor.io**
- Agent based model for scalable web and IoT apps
  - Related to actors, but different
- "Hyper-agile" approach to software development
- Agent conversation and domain languages
- IDEs, visualization, runtime

YAKTOR

- In development and use since 2010, open-sourced in Summer 2016

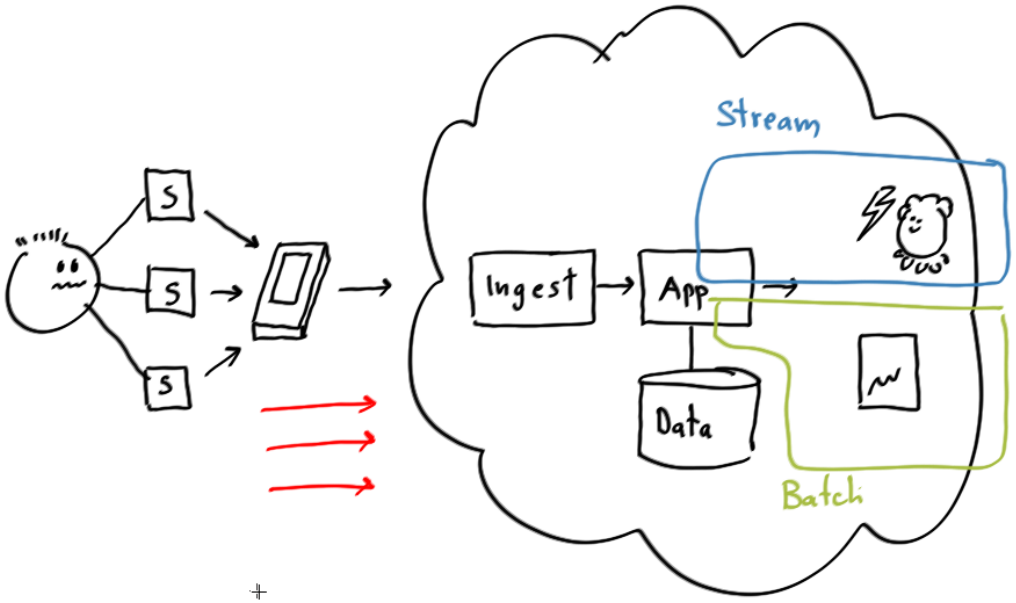




SciSpike

www.scispike.com

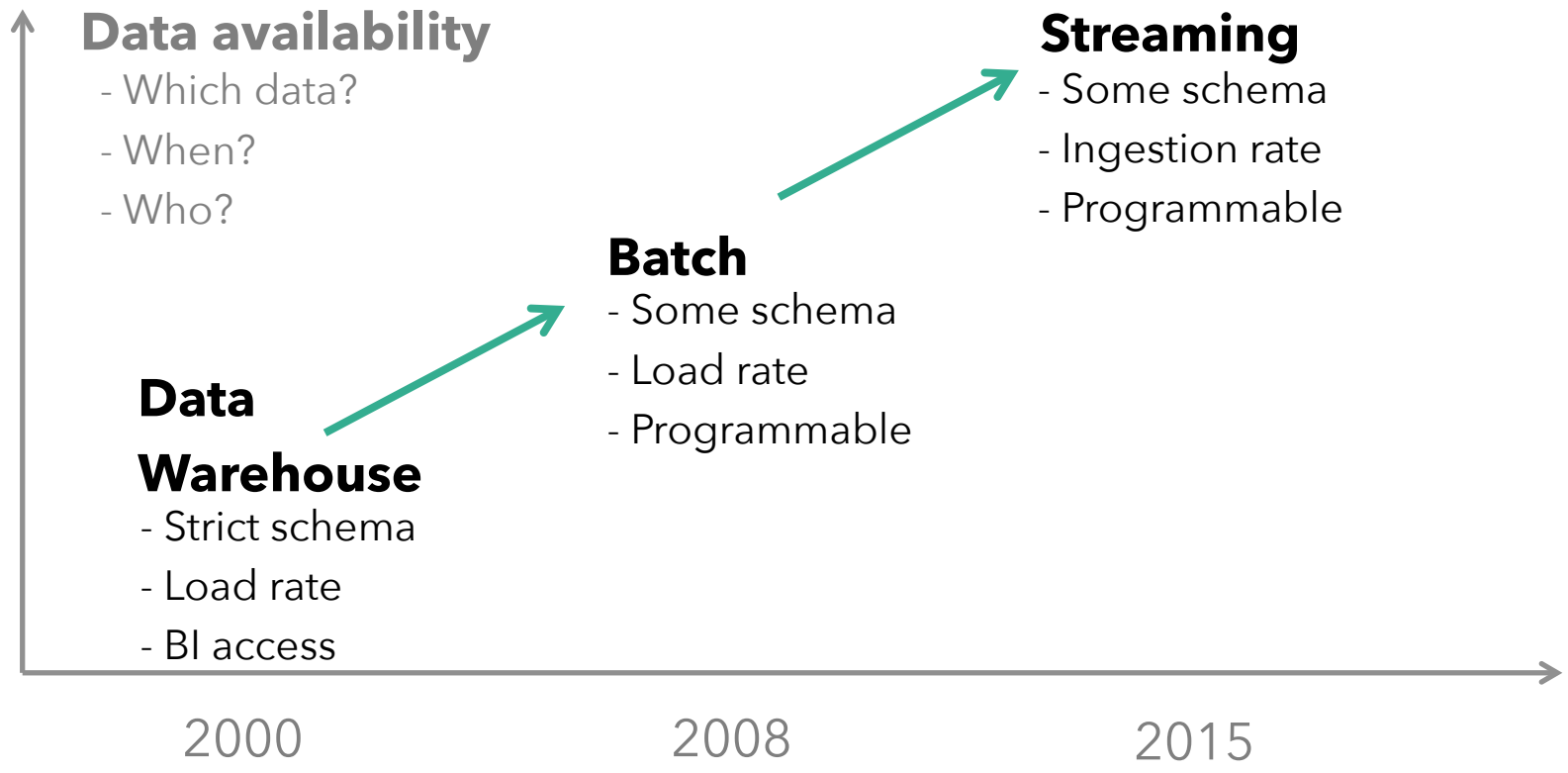
Some of Our Use Cases



- IoT applications
  - Healthcare
  - Automotive
- Big Data Applications

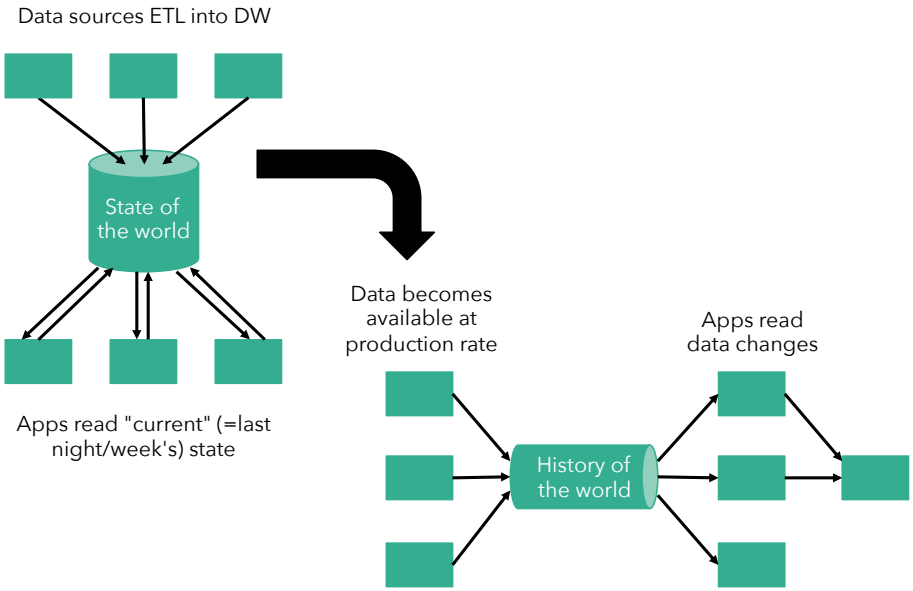
- Mix of workloads
  - Batch
  - Streaming
  - Real-Time Streaming

Why Streaming?



What Does Streaming Enable?

1. Data integration



2. Low latency applications

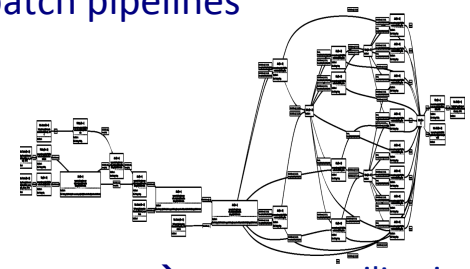
- Fresh recommendations, fraud detection, etc
- Internet of Things, intelligent manufacturing
- Results “right here, right now”

3. Batch < Streaming

Kleppmann: "Turning the DB inside out with Samza"

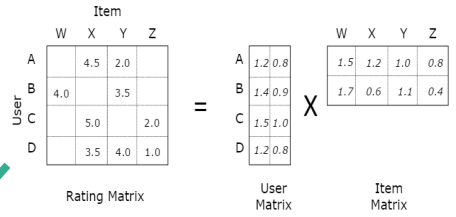
Requirements For The New Big Data Engine

Long batch pipelines



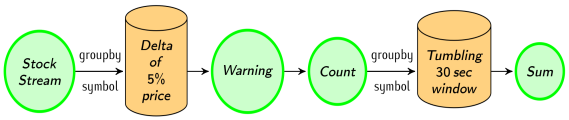
→ resource utilization

Machine Learning at scale



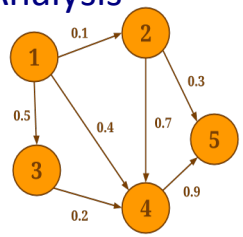
→ iterative algorithms

Streaming topologies



→ Low latency

Graph Analysis



→ Mutable state

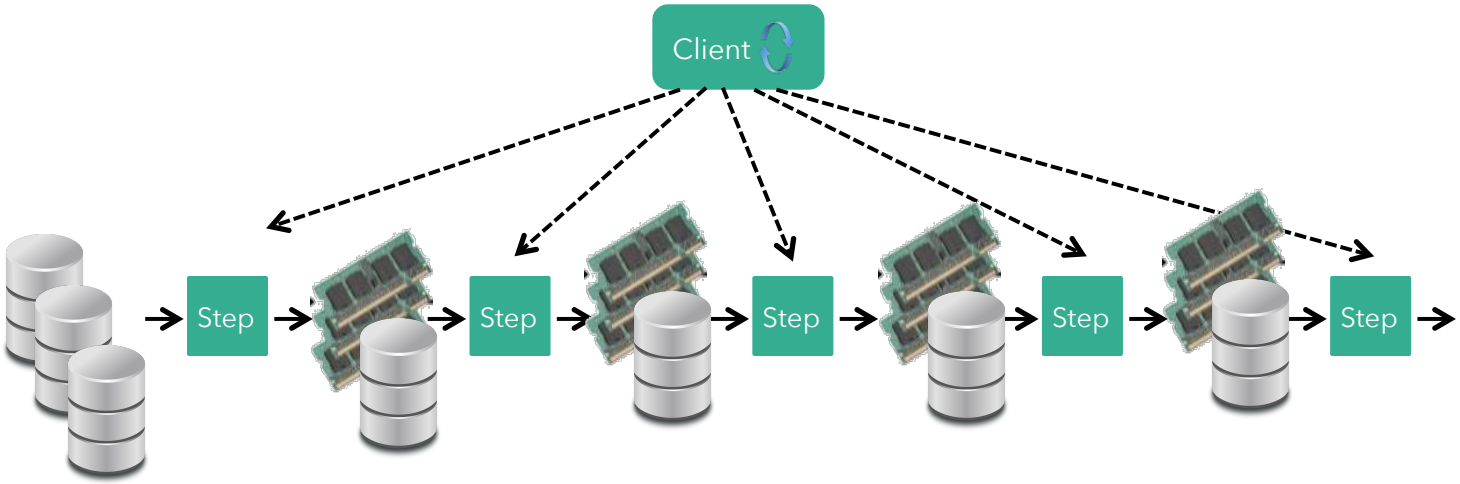
Flink

How can an engine **natively** support all these workloads?

And what does "native" **mean**?

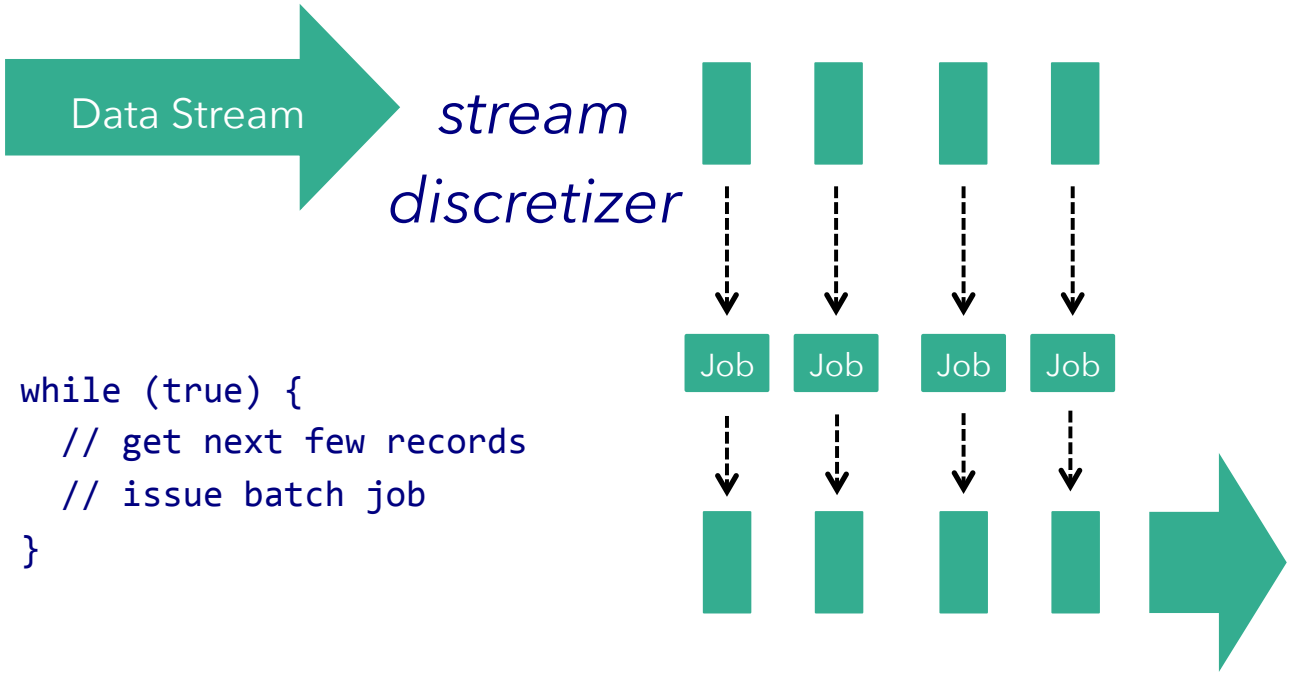
Not This: Non-Native Iterations

```
for (int i = 0; i < maxIterations; i++) {
        // Execute MapReduce job
}
```



- Iterations are needed:
  - ML: clustering, gradient descent,…
  - Graph processing: page rank, path algorithms,...

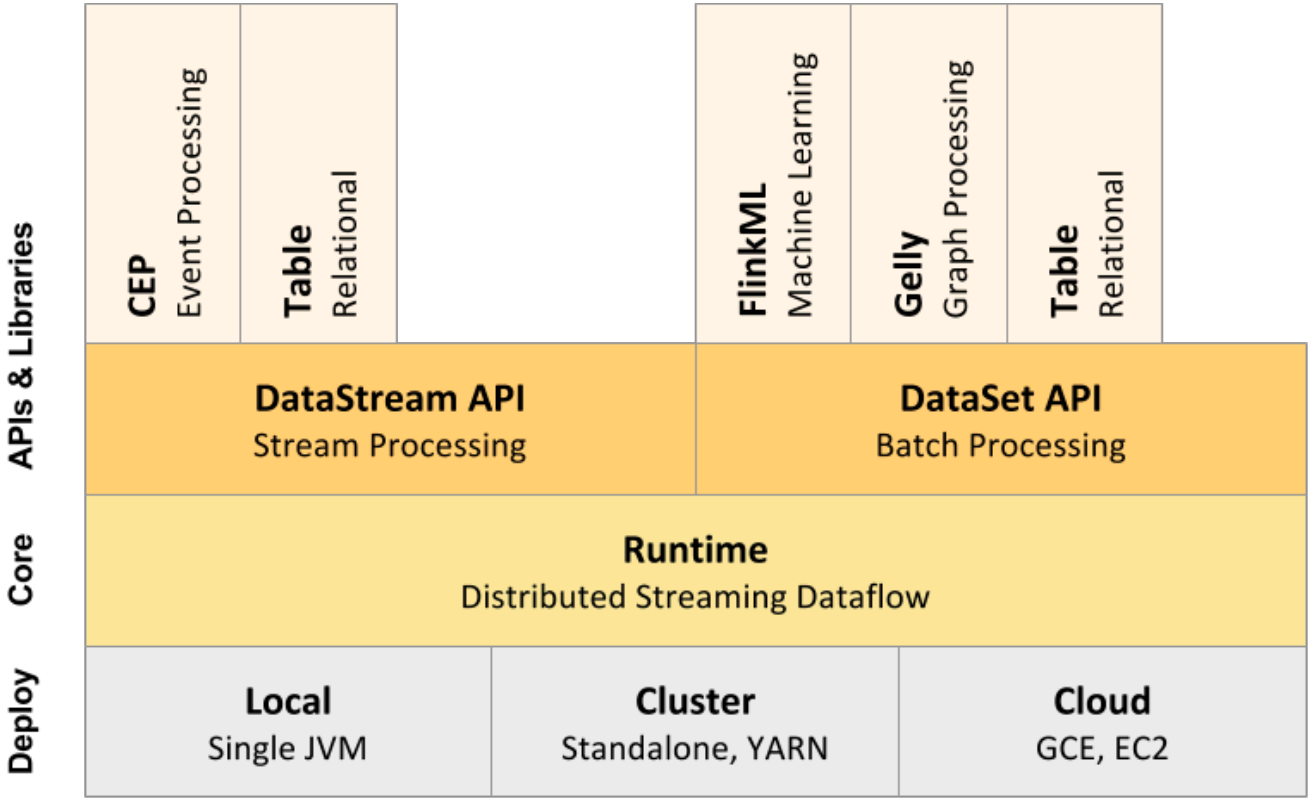
Not This: Non-Native Streaming




```
while (true) {
  // get next few records
  // issue batch job
}
```

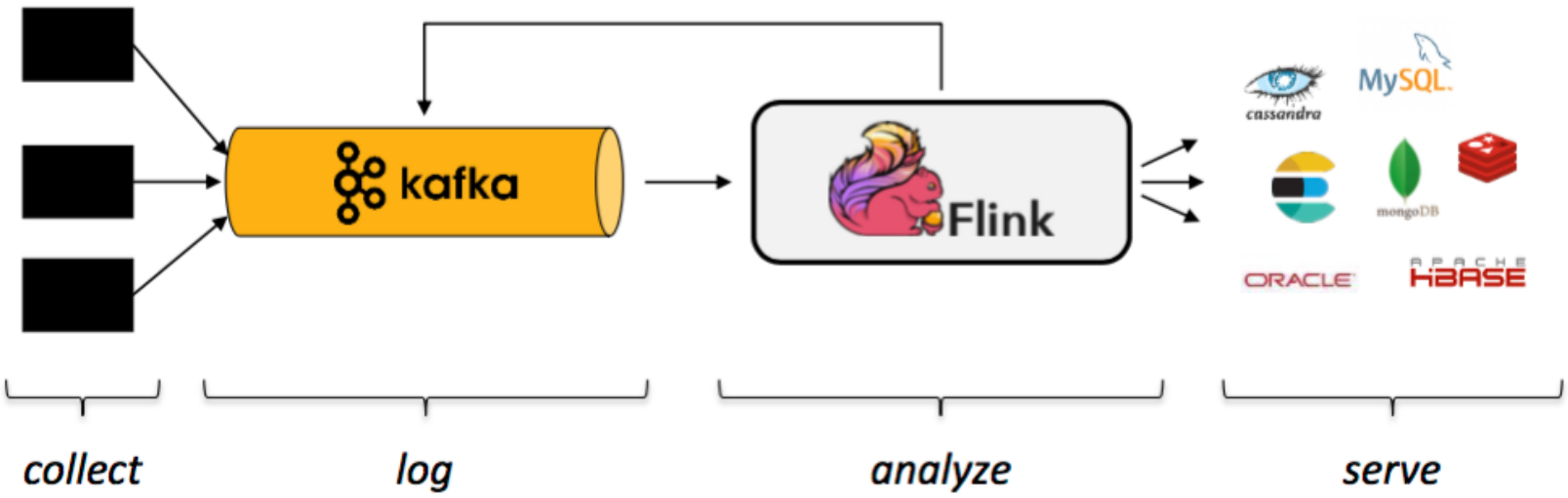
- Native streaming is needed for real-time response

Apache Flink Components



- Integration with Hadoop YARN, MapReduce, HBase, Cassandra, Kafka, …
- Execution engine for Apache Beam (Google Dataflow)

Stream Platform Architecture



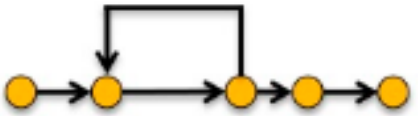
- Gather and backup streams
- Offer streams for consumption
- Provide stream recovery

- Analyze and correlate streams
- Create derived streams and state
- Provide these to downstream systems

Flink Characteristics: Native Support

1. **Execute everything as streams**
   - Pipelined execution,
     backpressure or buffered, push/pull model
   - Batch / streaming / relational / ML / graph

2. **Automatic batch job optimization,**
   fault tolerance
3. **Iterative (cyclic) dataflows**
4. **Mutable state**
5. **Managed memory**

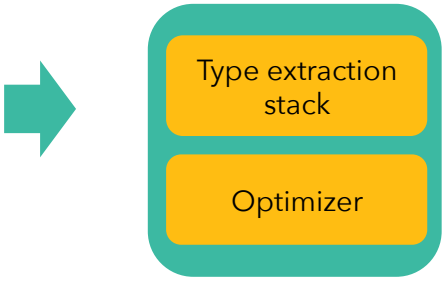


Little configuration needed:
- Memory
- Network
- Serialization

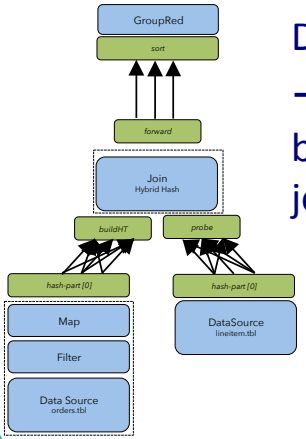
Program Compilation

```
case class Path (from: Long, to: Long)
val tc = edges.iterate(10) {
  paths: DataSet[Path] =>
    val next = paths
      .join(edges)
      .where("to")
      .equalTo("from") {
        (path, edge) =>
          Path(path.from, edge.to)
      }
      .union(paths)
      .distinct()
    next
  }
```

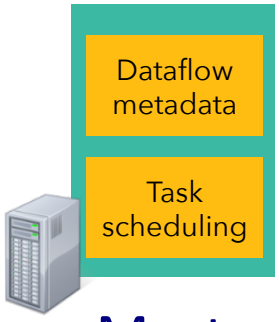
Program



Pre-flight (Client)



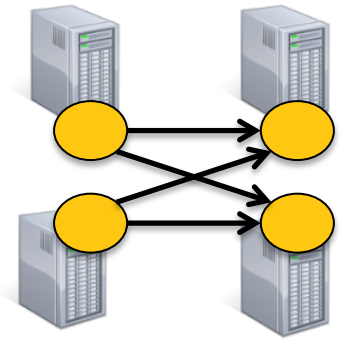
Dataflow Graph
→Independent of batch or streaming job



Master

deploy operators

track intermediate results



Workers

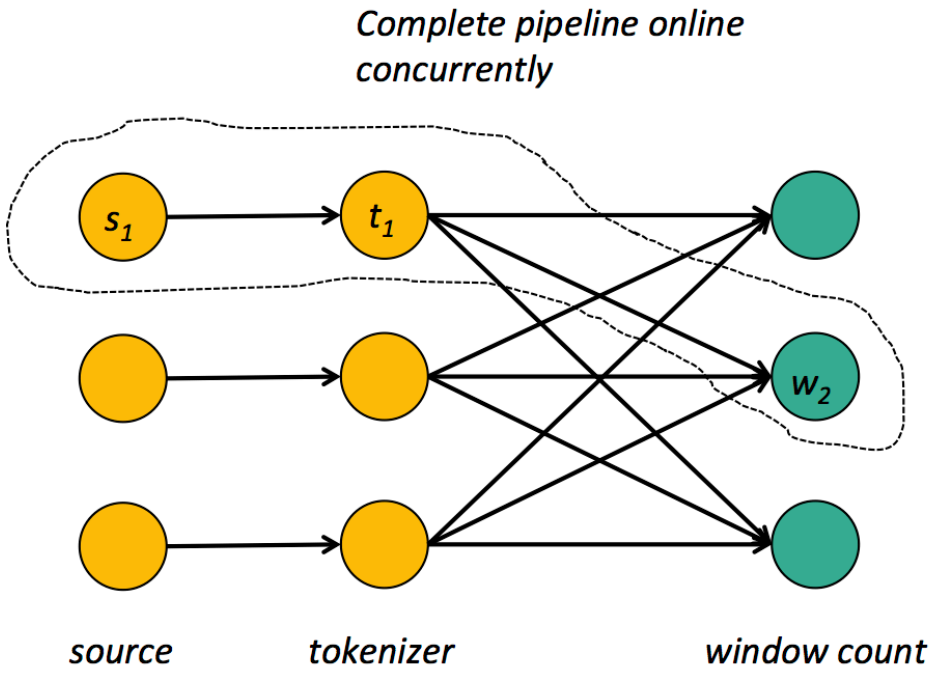
What Is A Stream Processor?

1. Pipelining 2. Stream replay	}	*Basics*
3. Operator state 4. Backup and restore	}	*State*
5. High-level APIs 6. Integration with batch	}	*App development*
7. High availability 8. Scale-in and scale-out	}	*Large deployments*

See http://data-artisans.com/stream-processing-with-flink.html

Parallelism and Stream Processing: Pipelining

Basic building block to “keep the data moving”



Transformations are executed concurrently

Pipelined systems do not usually transfer individual tuples, but buffers that batch several tuples

Origins in parallel databases

Operator State

- **User-defined state**
  - Flink transformations (map/reduce/etc) are long-running operators, feel free to keep around objects
  - Hooks to include in system's checkpoint
- **Windowed streams**
  - Time, count, data-driven windows
  - Managed by the system
- **Out of core state**
  - State checkpoints store to RocksDB

Windows

- **Tumbling windows**
  - No overlap
- **Sliding windows**
  - Overlap
- **Time based windows**
  - Based on the real world time when the event occurred

Time Characteristics

- Different notions of time:
  - **Processing** time
  - **Ingestion** time
  - **Event** time
    - Extract from the event data
- The events can arrive out of order
- **Watermark(X)**: mechanism to control the window completeness
  - "All input data with event times < X have been observed"
  - Generate watermarks yourself, based on your application

First open source streaming engine that can do this

Iterations

Iterate

- In each iteration:
  - The function uses the entire input
    - Data from the previous iteration, or initial data
  - Computes the result of the iteration

Delta iterate

- Runs only in data that changed
- Significant speedup

Streaming Fault Tolerance

- Ensure that operators see all events
  - "At least once"
  - Solved by replaying a stream from a checkpoint, e.g., from a past Kafka offset
- Ensure that operators do not perform duplicate updates to their state
  - **"Exactly once"**
  - Flink uses Chandy-Lamport inspired distributed snapshots
- Flink: end-to-end exactly once from sources to sinks:
  - e.g. Kafka → Flink → HDFS

Checkpoints

- Checkpoints taken regularly
  - Application does not need to stop
- At failure:
  - Rewind input stream to the logical time of the last checkpoint

Savepoints

- Checkpoints taken by the user
- Accessible externally
- Never expire

- Save: `flink savepoint <ID>`
- Resume: `flink run -s <pathToID> <jar>`

- Great for production:
  - Upgrades of applications, Flink, migration, what-if simulations, …

Benefits Of Flink's Approach

- **Data processing does not block**
  - Can checkpoint at any interval you like to balance overhead/ recovery time
- **Separates business logic from recovery**
  - Checkpointing interval is a config parameter, not a variable in the program (as in discretization)
- **Can support richer windows**
  - Session windows, event time, etc

- True streaming latency, exactly-once semantics, and low overhead for recovery

Code: Batch and Streaming

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

DataSet API (batch):

```
val lines: DataSet[String] = env.readTextFile(...)

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

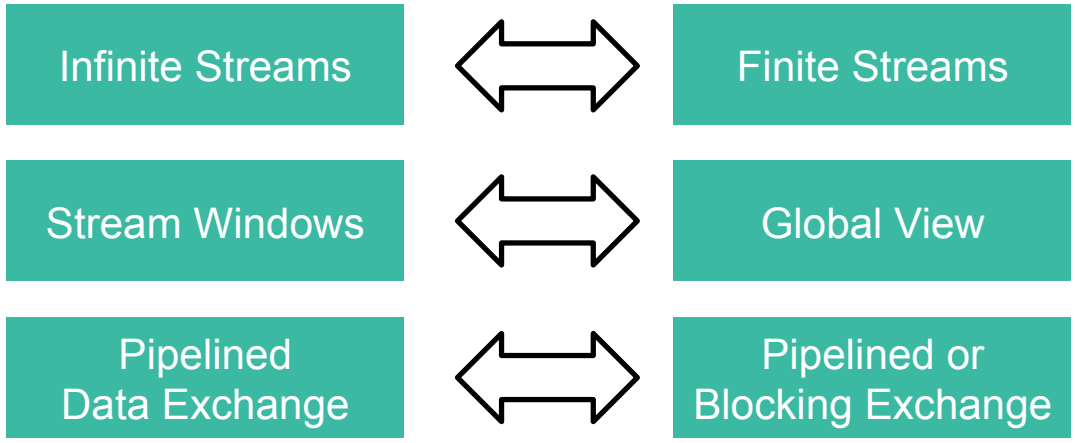
DataStream API (streaming):

```
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()
```

Batch on Streaming

- Batch programs are a special kind of streaming program



Flink Complex Event Processing (CEP): API

- Detecting event patterns with CEP queries in Real-Time
  - Match incoming events against patterns

Operation	Effect
Begin	Start state
Next	An event directly succeeds the previous matching event
FollowedBy	Like Next, but there may be other events in between
Where	Filter condition
Within	Time interval for a sequence to match the pattern
Subtype	Condition for event kind

Flink Complex Event Processing (CEP): Example

```
// Warning pattern: 2 high heart rate events with a high blood pressure
// within 10 seconds
Pattern<Measurement, ?> alarmPattern =
        Pattern.<Measurement>begin("first")
        .subtype(HeartMeasurement.class)
        .where(evt -> evt.getRisk() >= 1)
        .followedBy("middle")
        .subtype(BloodPressureMeasurement.class)
        .where(evt -> evt.getRisk() >= 2)
        .followedBy("last")
        .subtype(HeartMeasurement.class)
        .where(evt -> evt.getRisk() >= 3)
        .within(Time.seconds(10));
```

https://github.com/gsahbi/flinkicu/blob/master/src/main/java/hes/cs63/CEPMonitor/CEPMonitor.java

Apache Beam Capabilities and Flink

- Apache Beam: a portable API layer for building sophisticated data-parallel processing engines that may be executed across a diversity of execution engines

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

Apache Beam: Flink and Other Runners

(expand details) **What is being computed?**

	Beam Model	Google Cloud Dataflow	Apache Flink	Apache Spark
ParDo	✓	✓	✓	✓
GroupByKey	✓	✓	✓	~
Flatten	✓	✓	✓	✓
Combine	✓	✓	✓	✓
Composite Transforms	✓	~	~	~
Side Inputs	✓	✓	~ (BEAM-102)	~
Source API	✓	✓	✓	✓
Aggregators	~	~	~	~
Keyed State	× (BEAM-25)	×	×	×

(expand details) **When in processing time?**

	Beam Model	Google Cloud Dataflow	Apache Flink	Apache Spark
Configurable triggering	✓	✓	✓	×
Event-time triggers	✓	✓	✓	×
Processing-time triggers	✓	✓	✓	✓
Count triggers	✓	✓	✓	×
[Meta]data driven triggers	× (BEAM-101)	×	×	×
Composite triggers	✓	✓	✓	×
Allowed lateness	✓	✓	✓	×
Timers	× (BEAM-27)	×	×	×

(expand details) **Where in event time?**

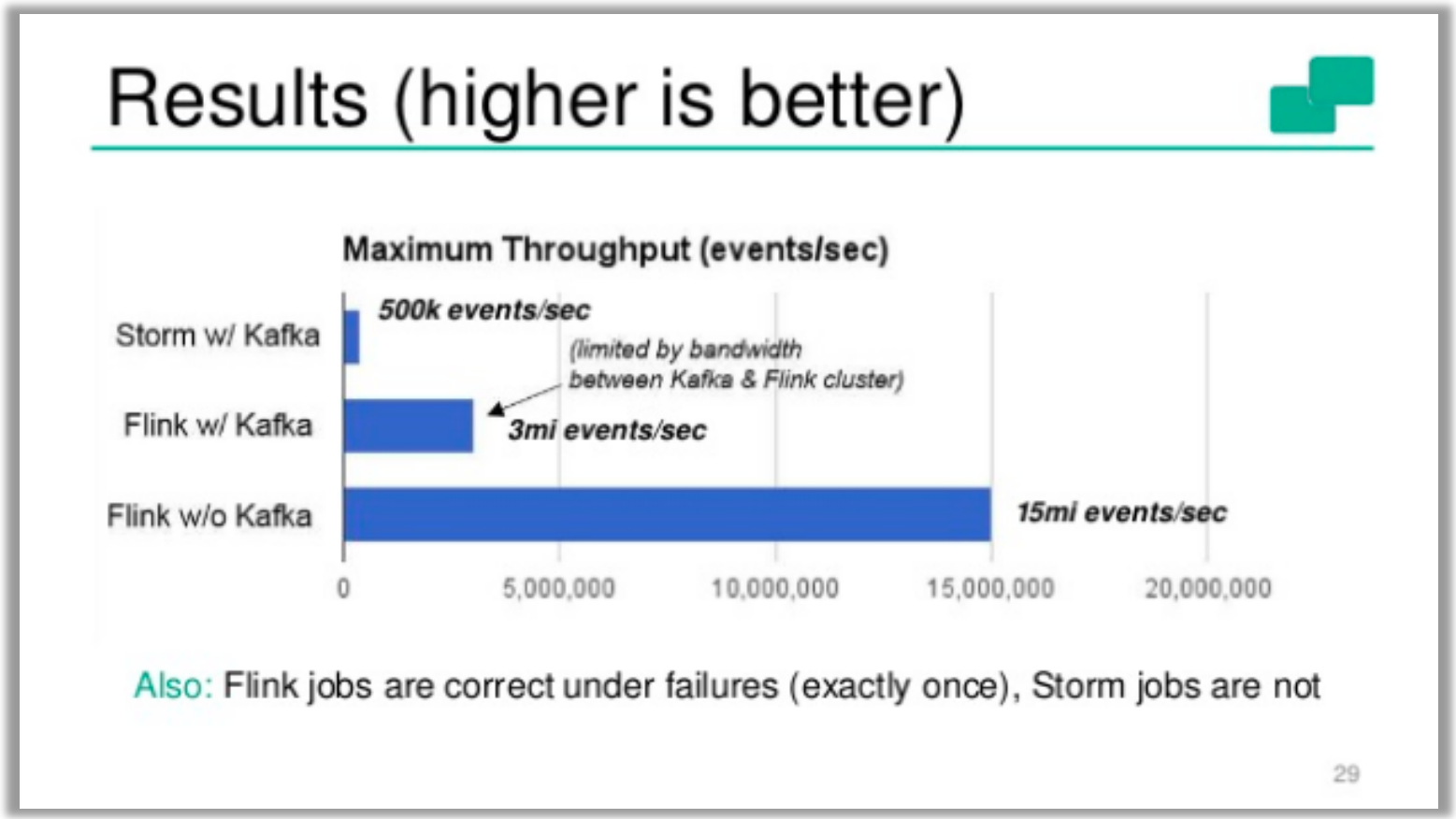
	Beam Model	Google Cloud Dataflow	Apache Flink	Apache Spark
Global windows	✓	✓	✓	✓
Fixed windows	✓	✓	✓	~
Sliding windows	✓	✓	✓	~
Session windows	✓	✓	✓	×
Custom windows	✓	✓	✓	×
Custom merging windows	✓	✓	✓	×
Timestamp control	✓	✓	✓	×

(expand details) **How do refinements relate?**

	Beam Model	Google Cloud Dataflow	Apache Flink	Apache Spark
Discarding	✓	✓	✓	✓
Accumulating	✓	✓	✓	×
Accumulating & Retracting	× (BEAM-91)	×	×	×

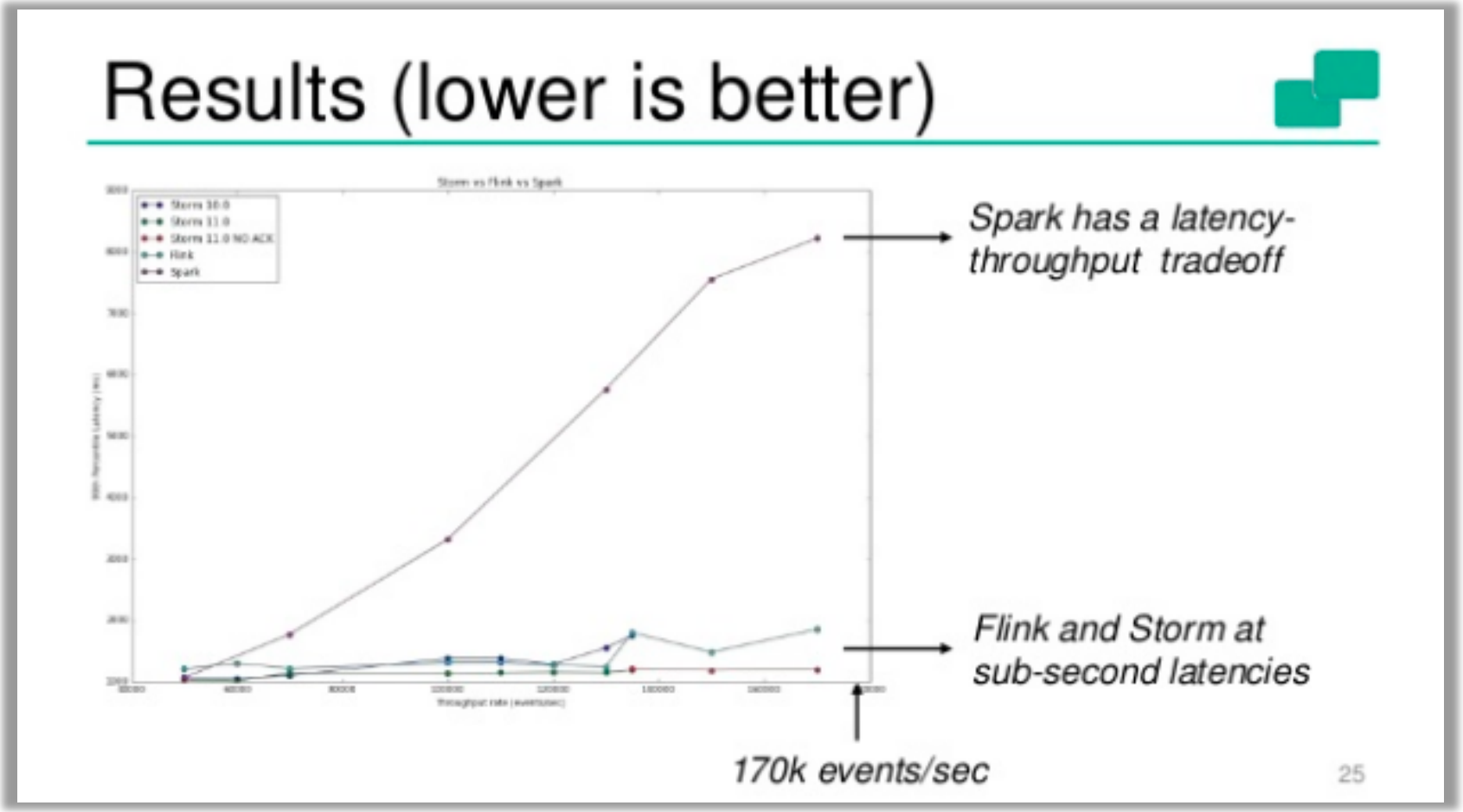
- http://beam.incubator.apache.org/learn/runners/capability-matrix/

Some Benchmarks…



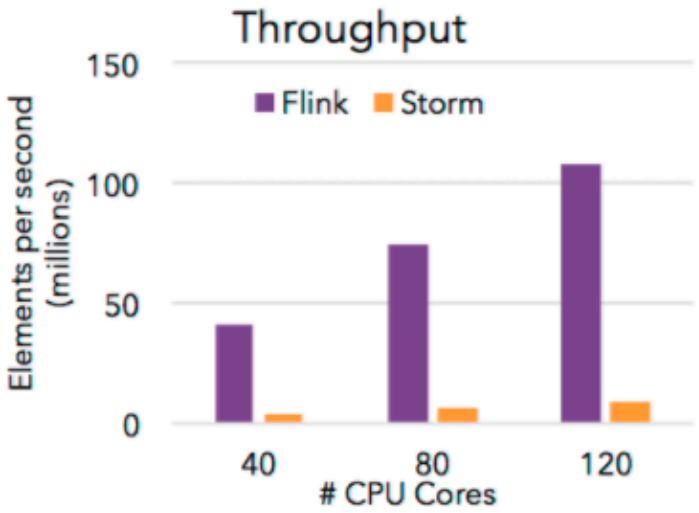
- Strata San Jose 2016
  - http://www.slideshare.net/KostasTzoumas/apache-flink-at-strata-san-jose-2016

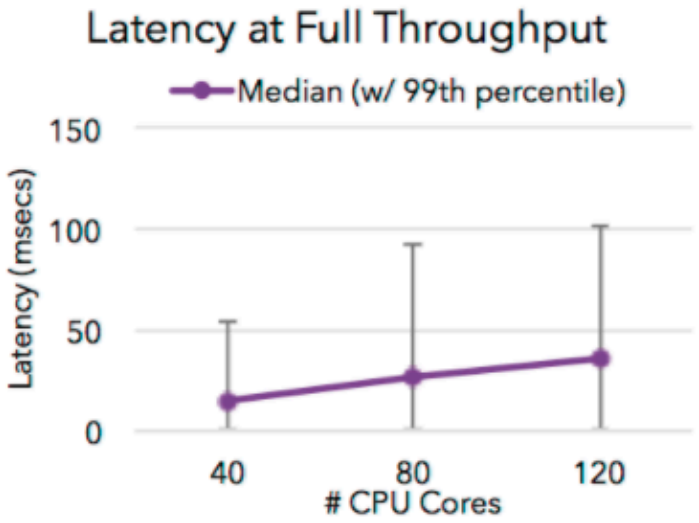
Some Benchmarks…



- Strata San Jose 2016
  - http://www.slideshare.net/KostasTzoumas/apache-flink-at-strata-san-jose-2016

Throughput and Low Latency





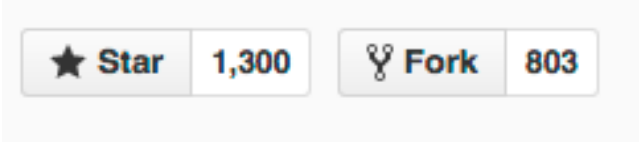
- http://flink.apache.org

Upcoming Features

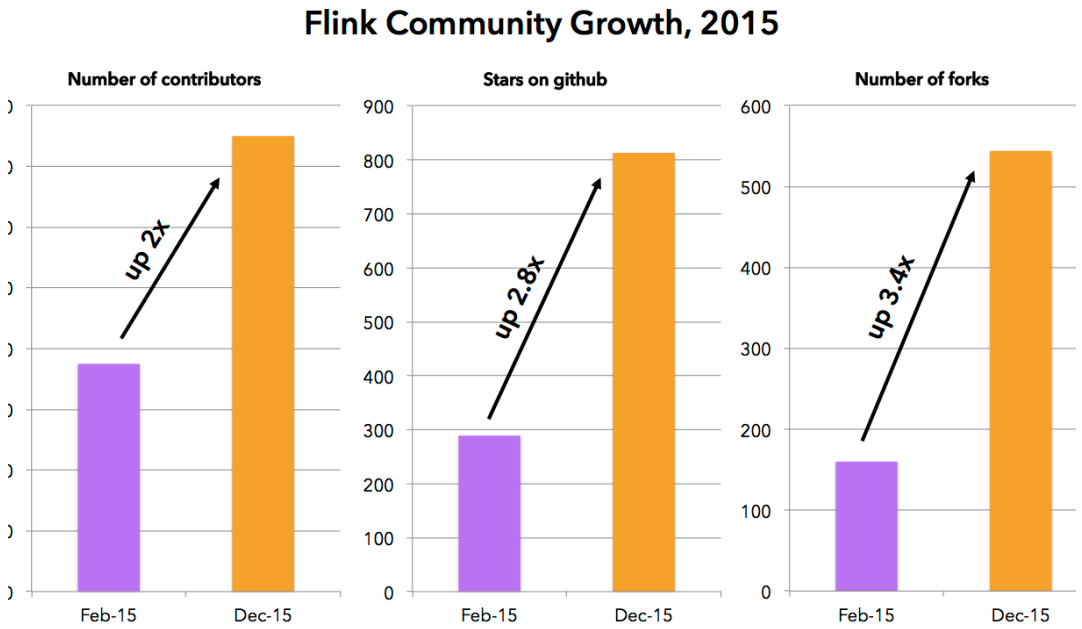
- **SQL for Streaming**
  - Together with Apache Calcite
- **Dynamic Scaling**
  - Adapt resources to stream volume
- **Queryable State**
  - Query the state directly: no need to store the data to an external database
- **Mesos Support**
- **Security**
  - Over the wire encryption

Flink Community

https://github.com/apache/flink



182 contributors



https://flink.apache.org/news/2015/12/18/a-year-in-review.html

Flink Resources

- Apache Flink site: https://flink.apache.org/
- Subscribe to news@flink.apache.org,
- Blog: flink.apache.org/blog
- Blog: http://data-artisans.com/blog/
- Twitter: @ApacheFlink

- Performance comparisons:
  - A Comparative Performance Evaluation of Apache Flink http://www.slideshare.net/ssuser6bb12d/a-comparative-performance-evaluation-of-apache-flink
  - Extending the Yahoo! Streaming Benchmark http://data-artisans.com/extending-the-yahoo-streaming-benchmark/

Resources: Flink and Other Streaming Systems

- The world beyond batch: Streaming 101 & 102
  - https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-101
  - https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-102
- Apache Beam Capability Matrix – framework for comparison of various streaming platforms:
  - http://beam.incubator.apache.org/learn/runners/capability-matrix/
  - Dataflow/Beam & Spark: A Programming Model Comparison https://cloud.google.com/dataflow/blog/dataflow-beam-and-spark-comparison
- A great set of Flink references and presentations from Slim Baltagi:
  - http://sparkbigdata.com/

Thank you!
Questions?

vladimir.bacvanski@scispike.com