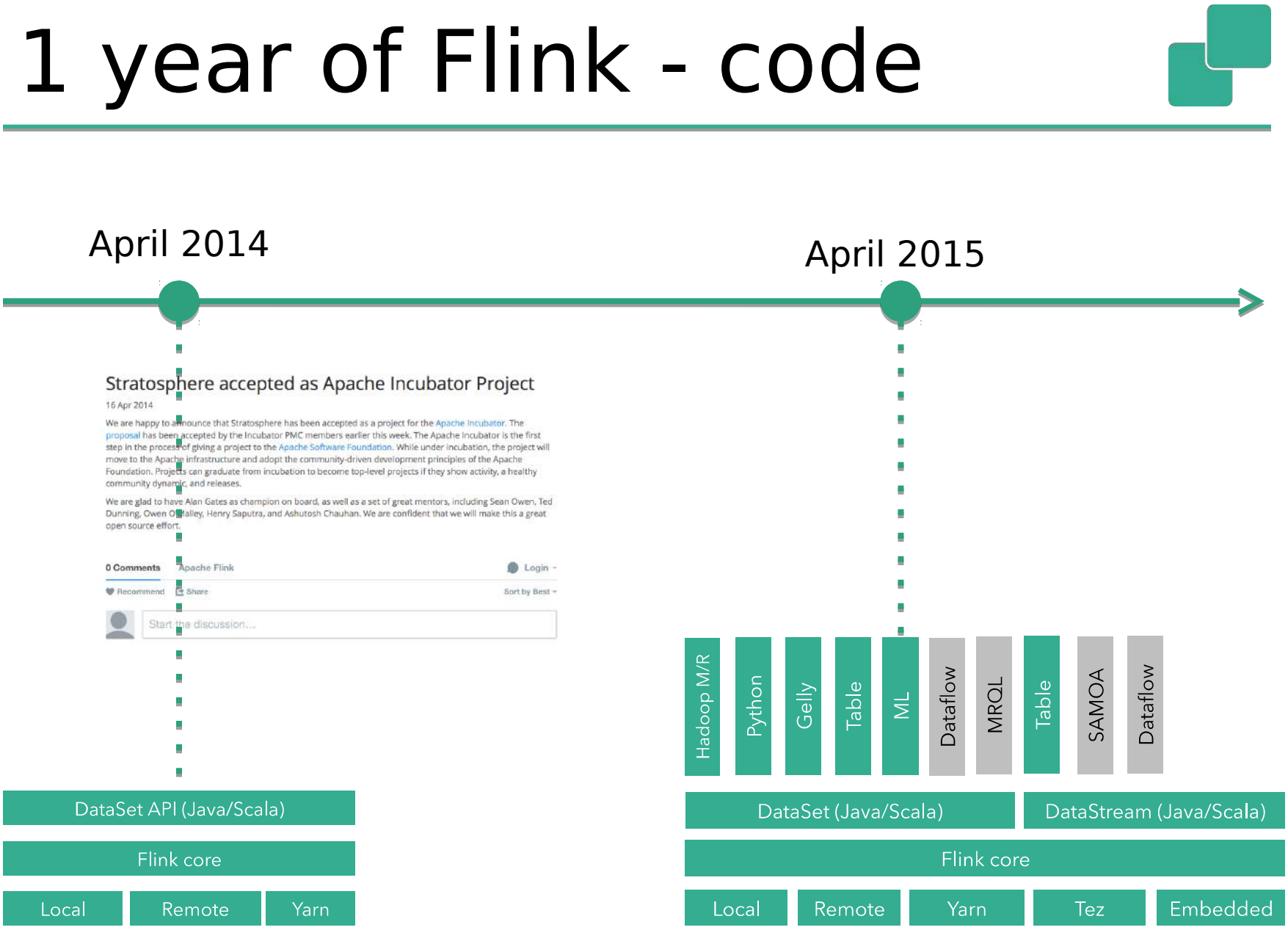
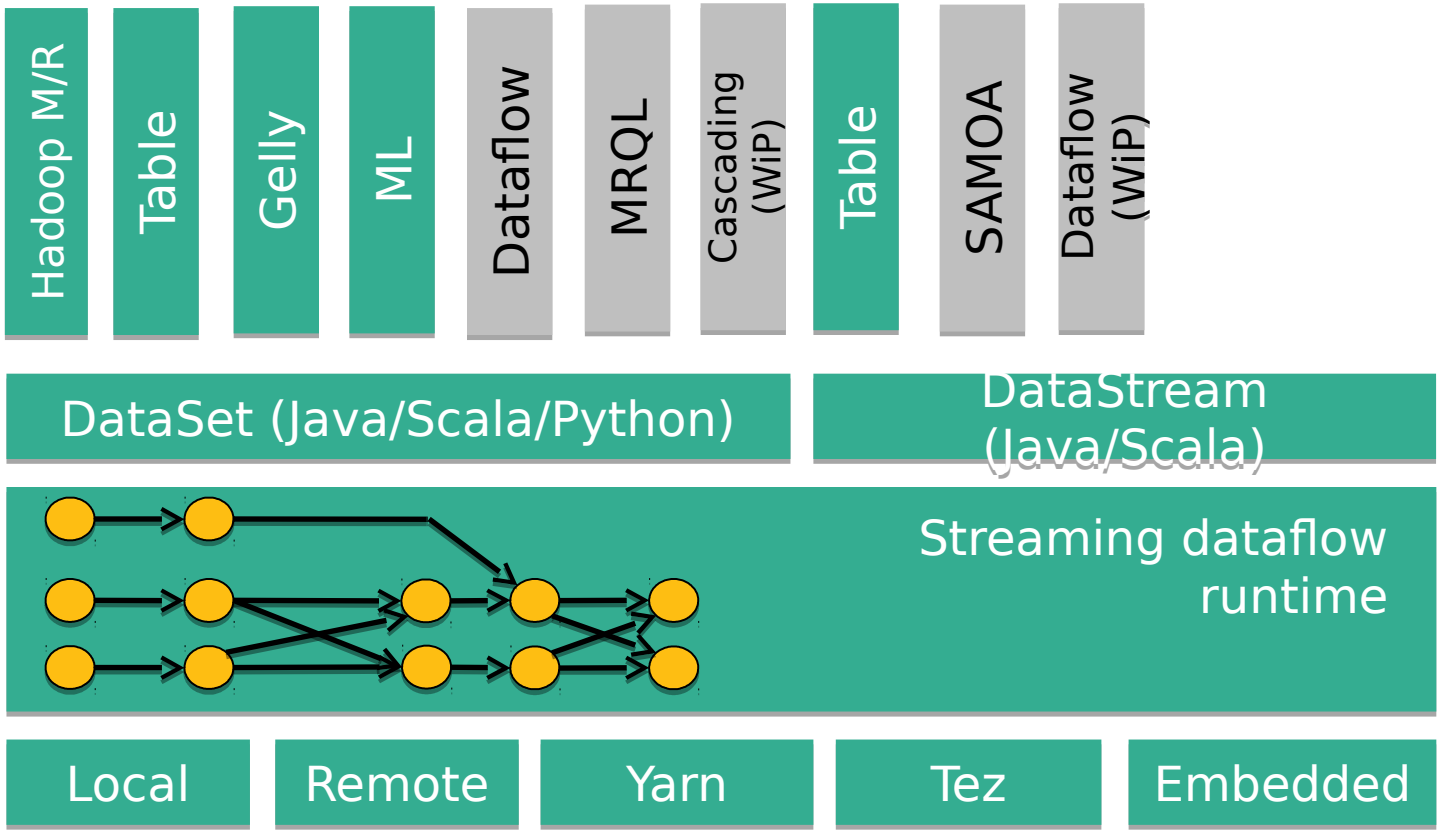
Apache Flink

Stephan Ewen
Flink committer
co-founder @ *data Artisans*
@StephanEwen

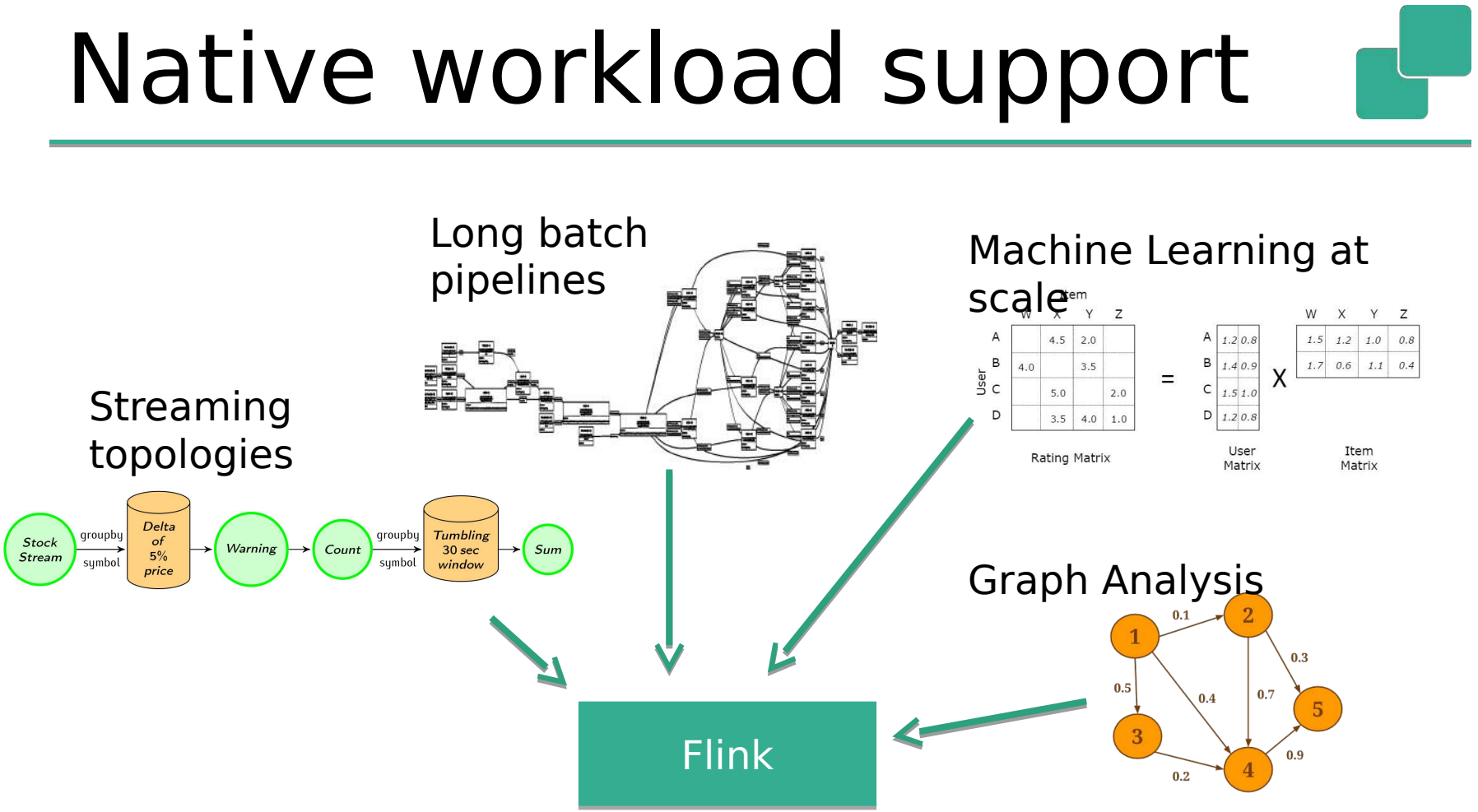
1 year of Flink - code



What is Flink



Native workload support

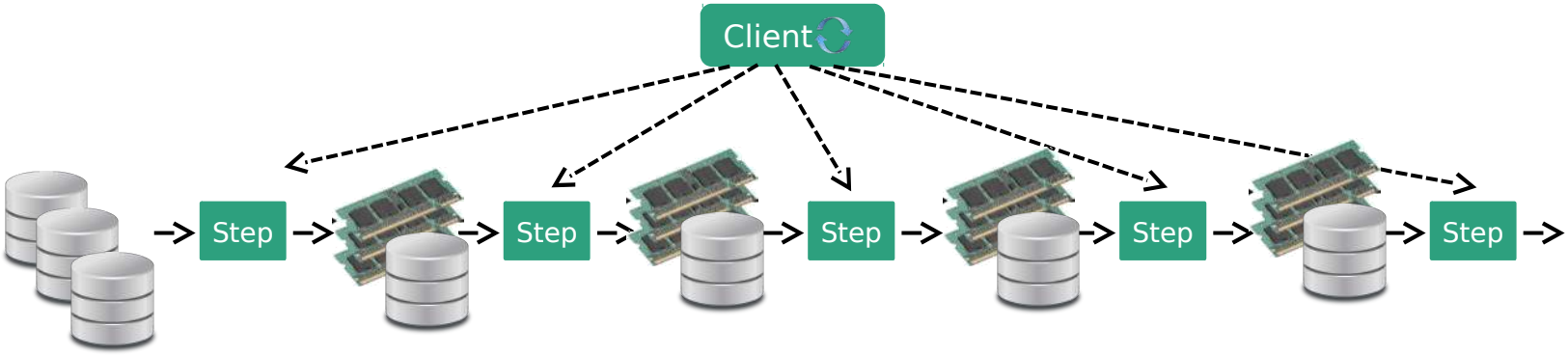


ow can an engine **natively** support all these workloads?

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

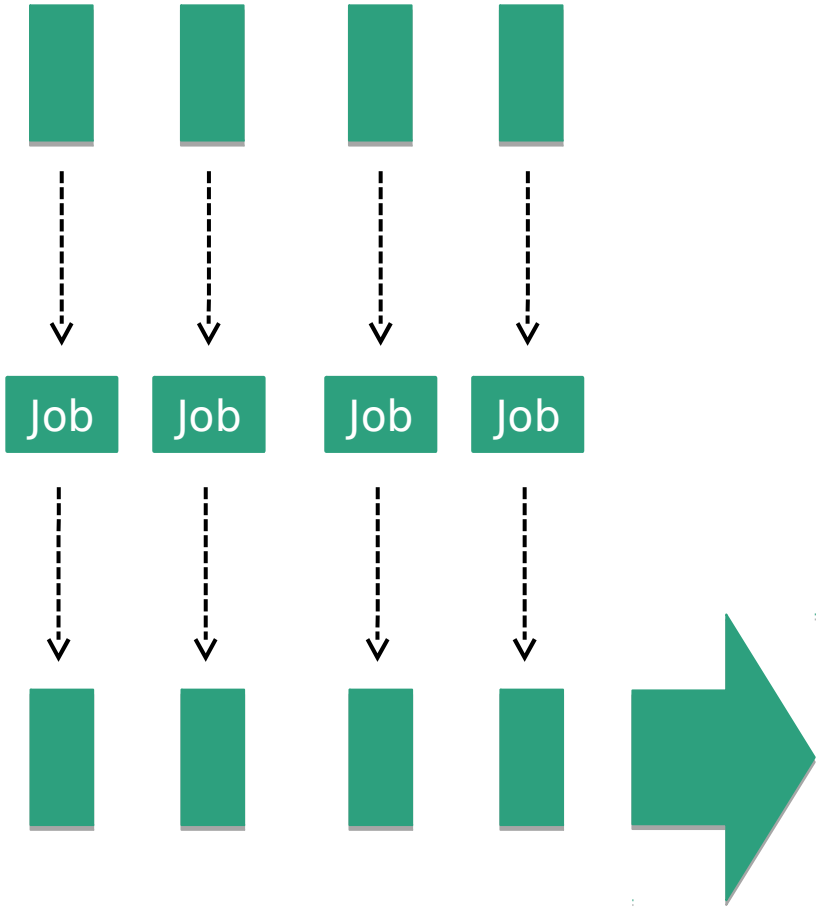
E.g.: Non-native iterations

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



E.g.: Non-native streaming

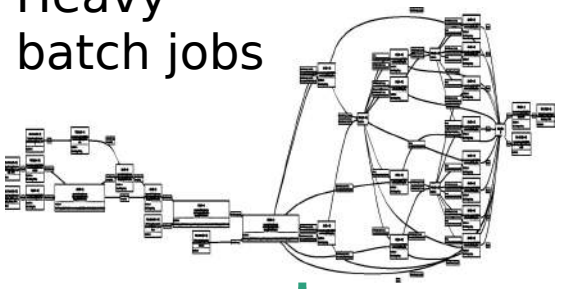
stream discretizer



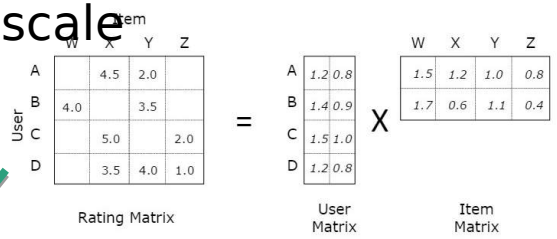

```
while (true) {
  // get next few records
  // issue batch job
}
```

Native workload support

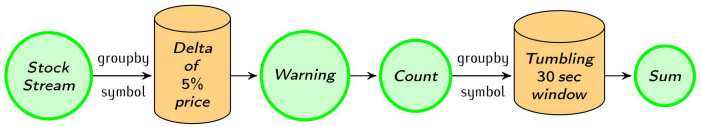
Heavy batch jobs



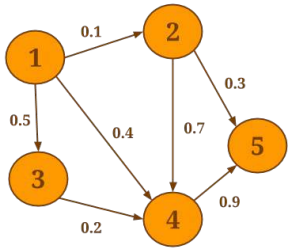
Machine Learning at scale



Streaming topologies



Flink



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

And what does native **mean**?

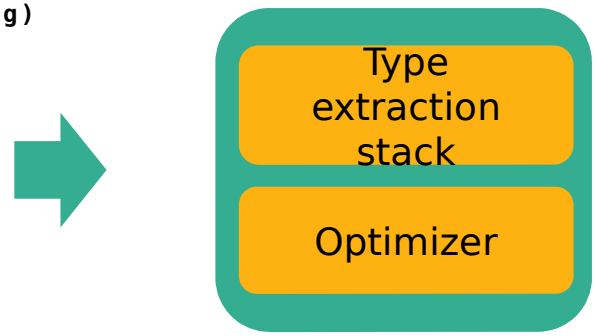
Flink Engine

1. Execute everything as streams
2. Allow some iterative (cyclic) dataflows
3. Allow and handle (mutable) state
4. Operate on managed memory

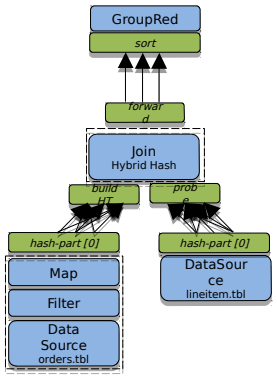
Program compilation

```
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
 }
```

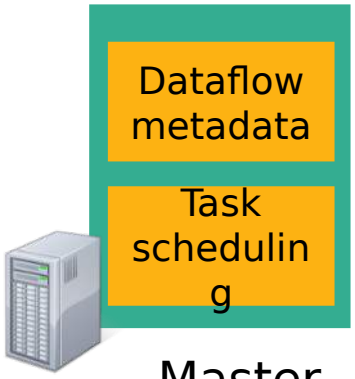
Program



Pre-flight (Client)



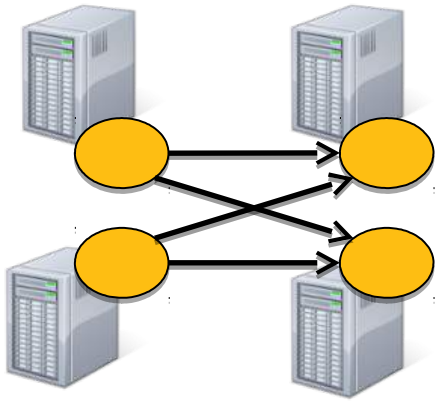
Dataflow Graph



Master

deploy operators

track intermediate results



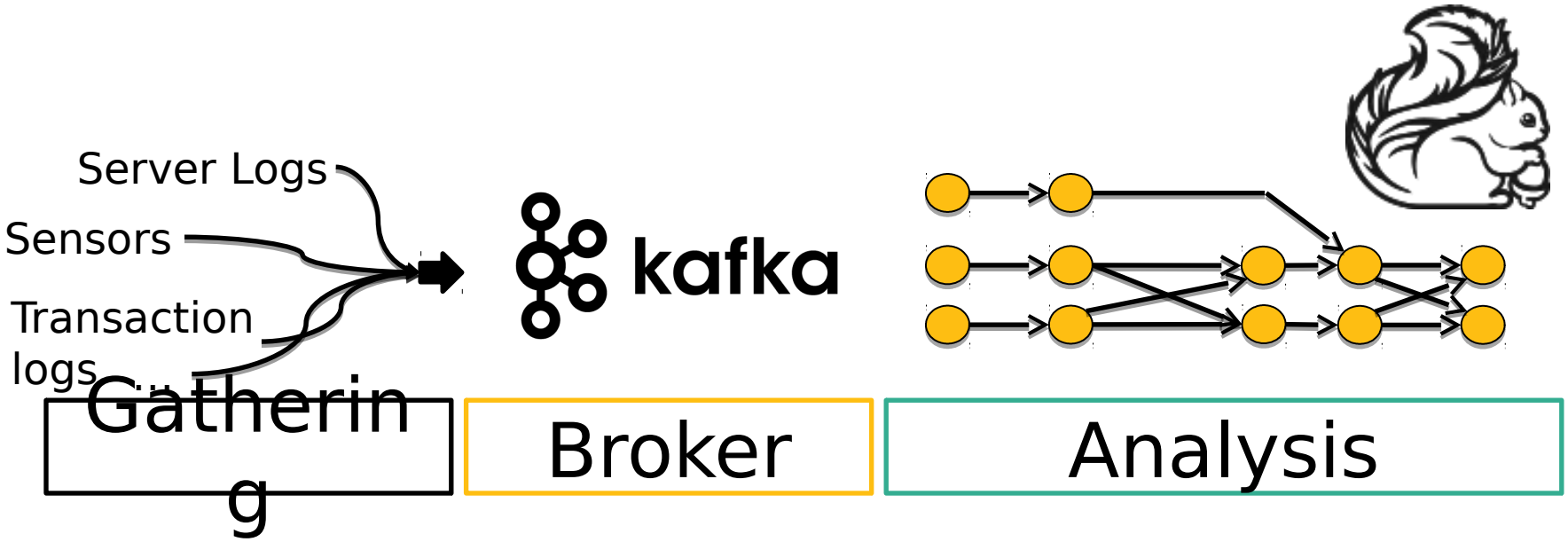
Workers

Flink by Use Case

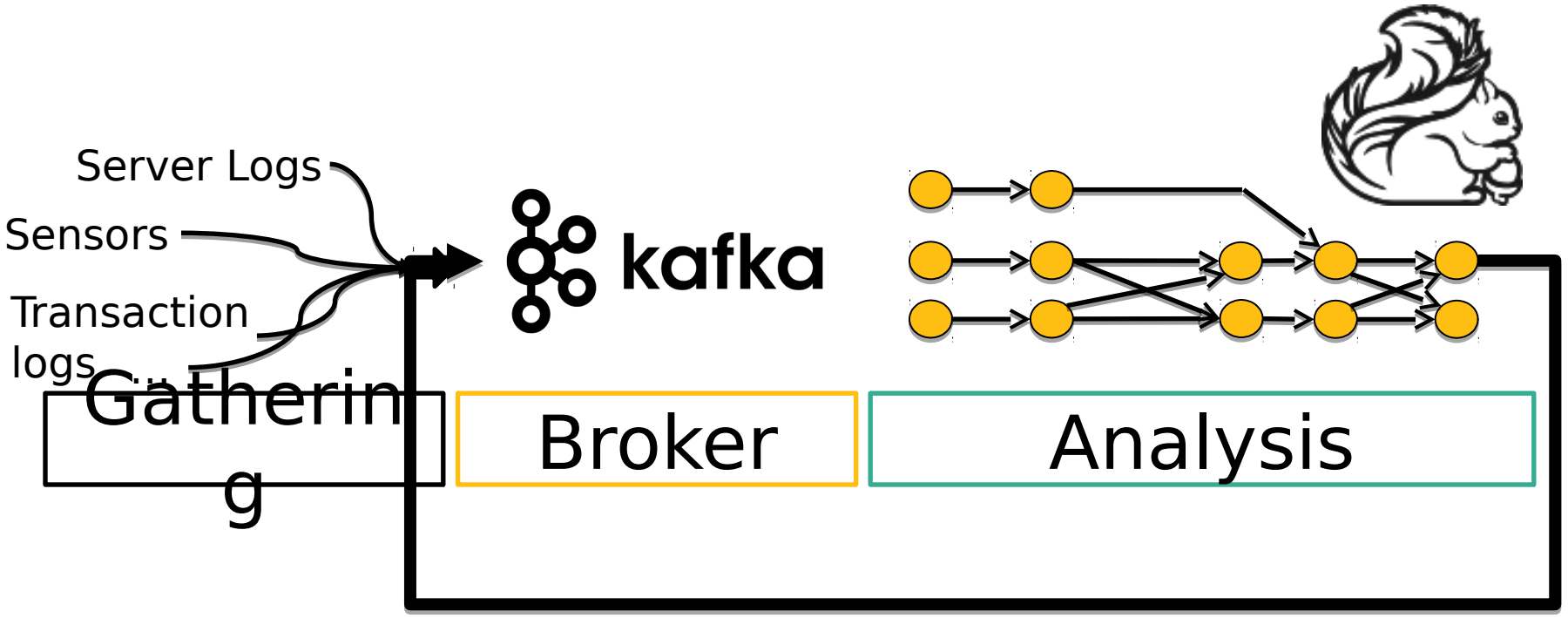
streaming dataflows

Data Streaming Analysis

3 Parts of a Streaming Infrastructure



3 Parts of a Streaming Infrastructure

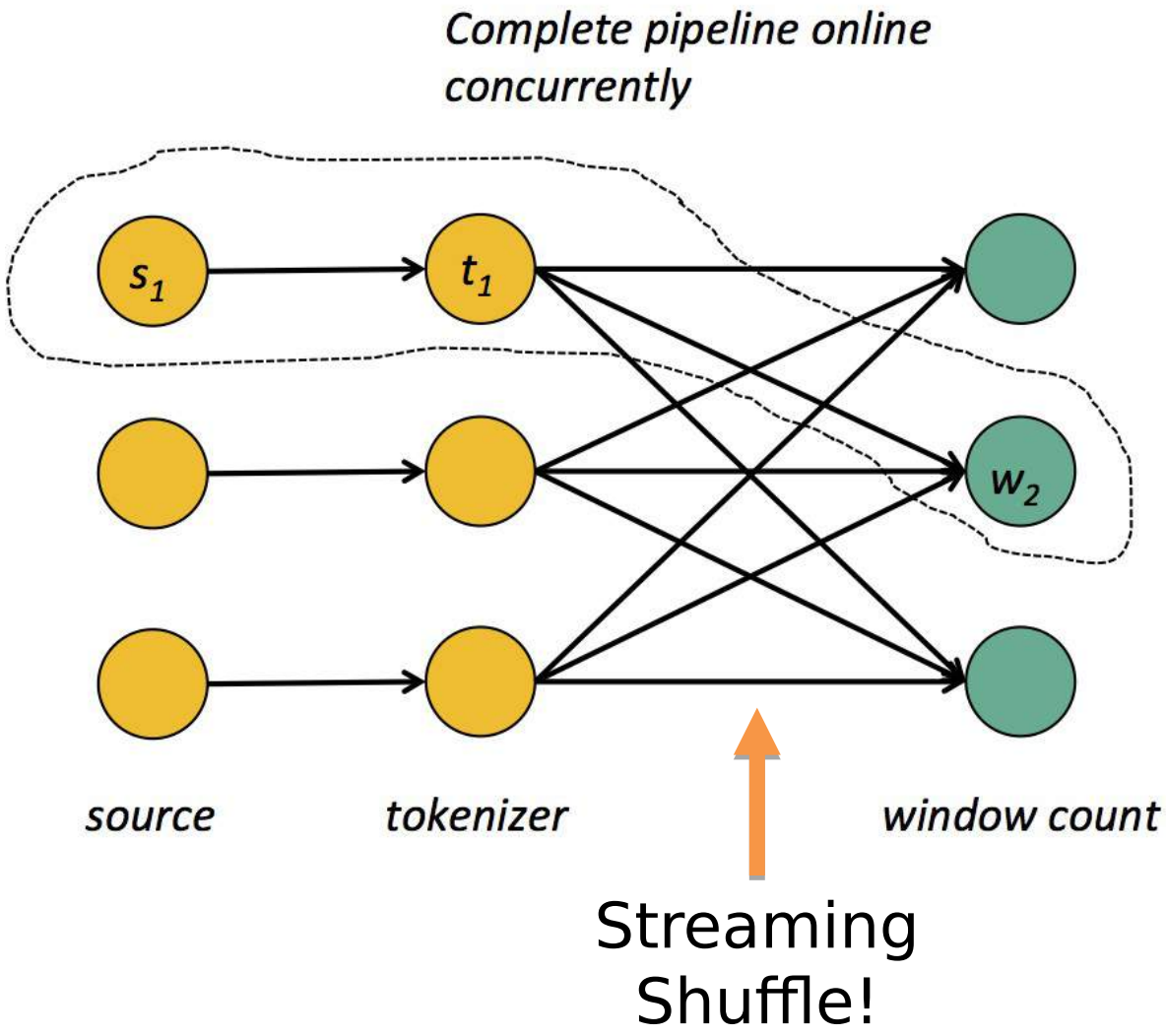


Result may be fed back to the broker

Cornerstones of Flink Streaming

- Pipelined stream processor (low latency)
- Expressive APIs
- Flexible operator state, streaming windows
- Efficient fault tolerance for streams and state.

Pipelined stream processor



Expressive APIs

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

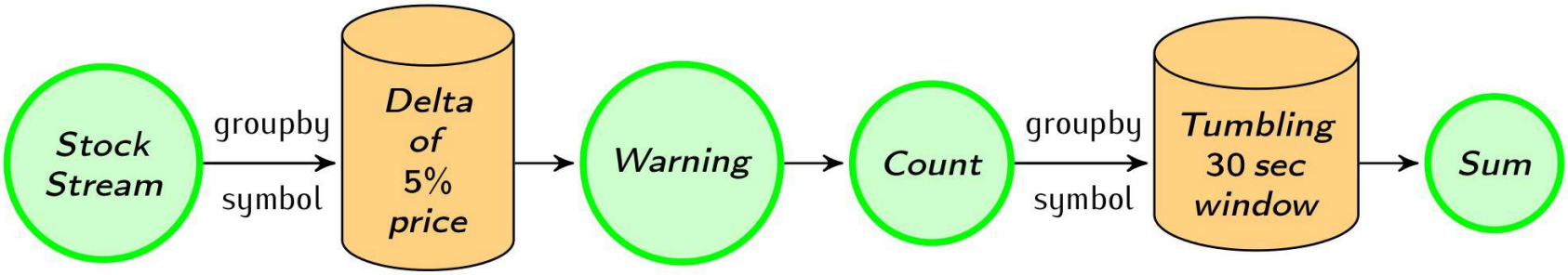
DataSet API (batch):

```
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()
```

Windows

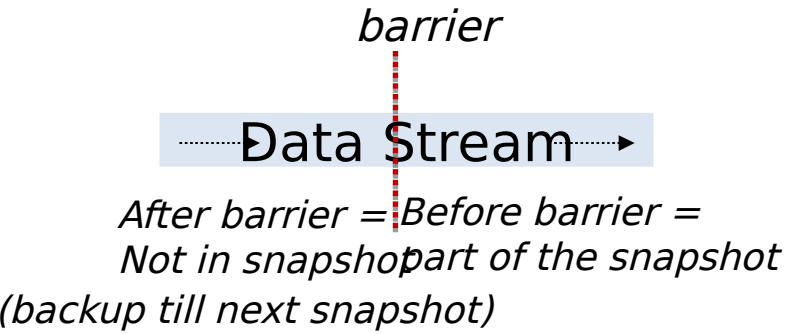


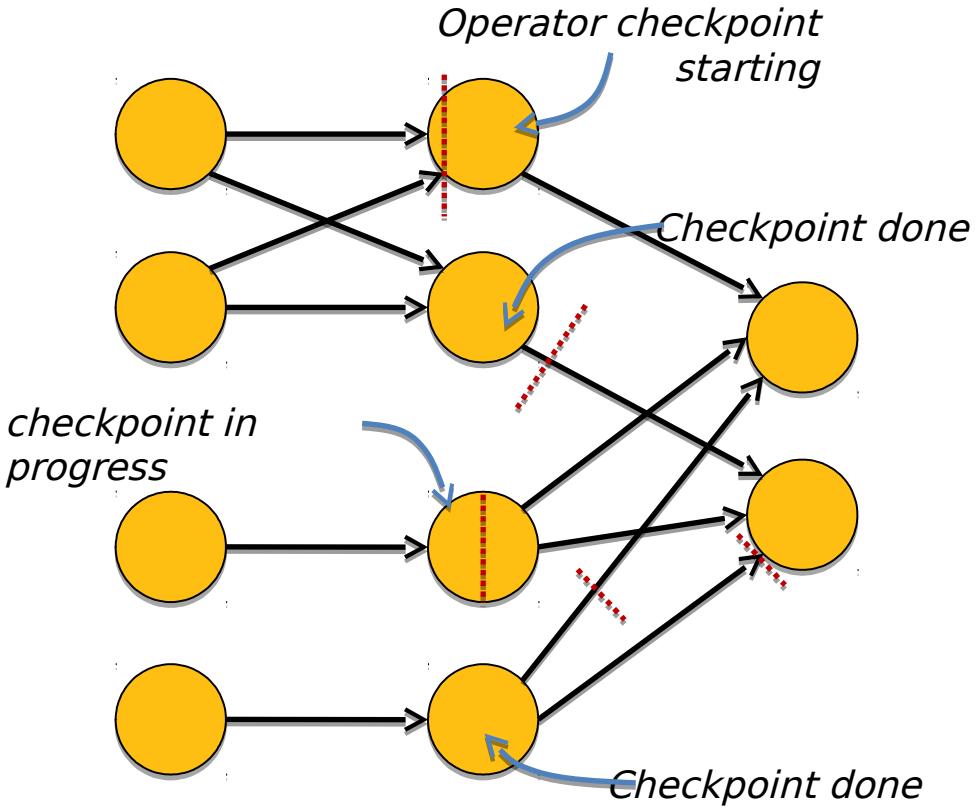

```
case class Count(symbol: String, count: Int)
val defaultPrice = StockPrice("", 1000)

//Use delta policy to create price change warnings
val priceWarnings = stockStream.groupBy("symbol")
  .window(Delta.of(0.05, priceChange, defaultPrice))
  .mapWindow(sendWarning _)

//Count the number of warnings every half a minute
val warningsPerStock = priceWarnings.map(Count(_, 1))
  .groupBy("symbol")
  .window(Time.of(30, SECONDS))
  .sum("count")
```

More at: http://flink.apache.org/news/2015/02/09/streaming-example.html

Checkpointing / Recovery

Pushes checkpoint barriers through the data flow



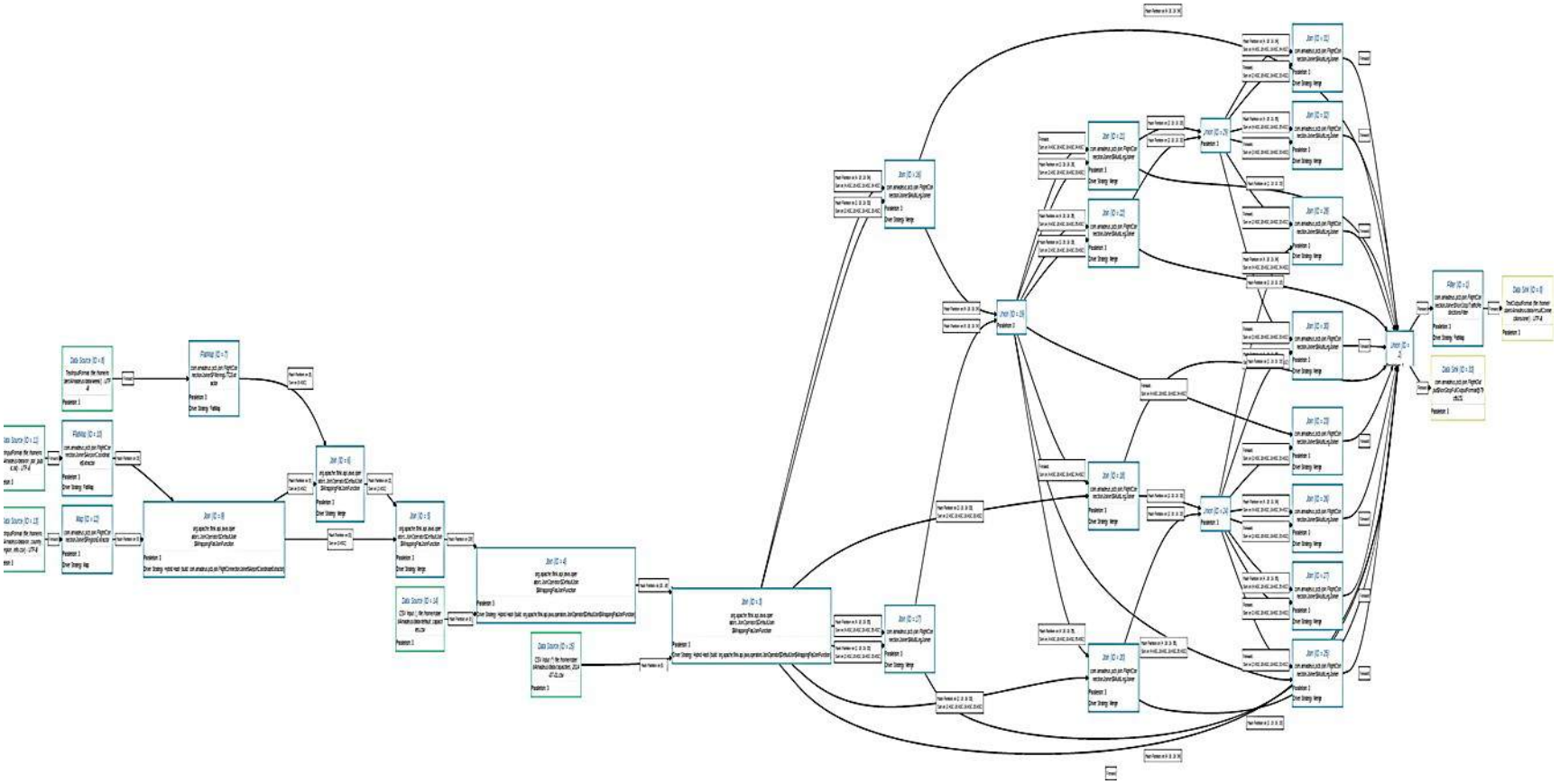


Chandy-Lamport Algorithm for consistent asynchronous distributed snapshots

Batch on Streaming

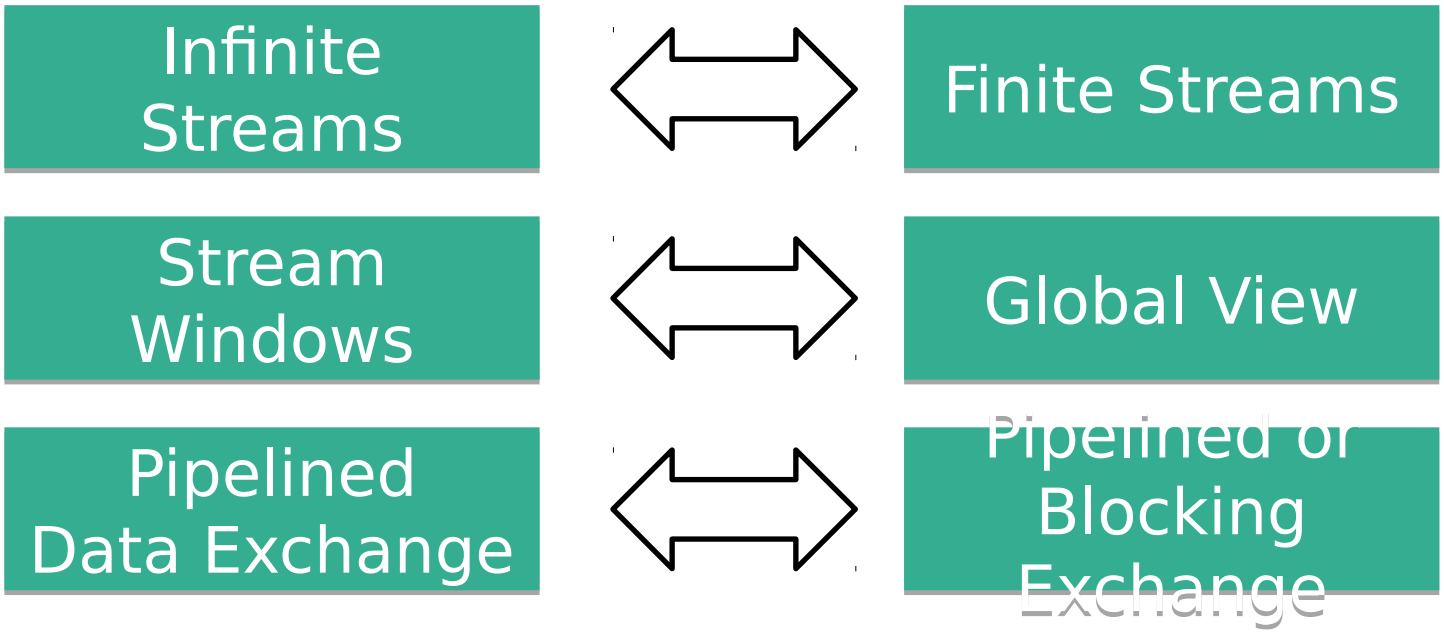
Long batch pipelines

Batch Pipelines

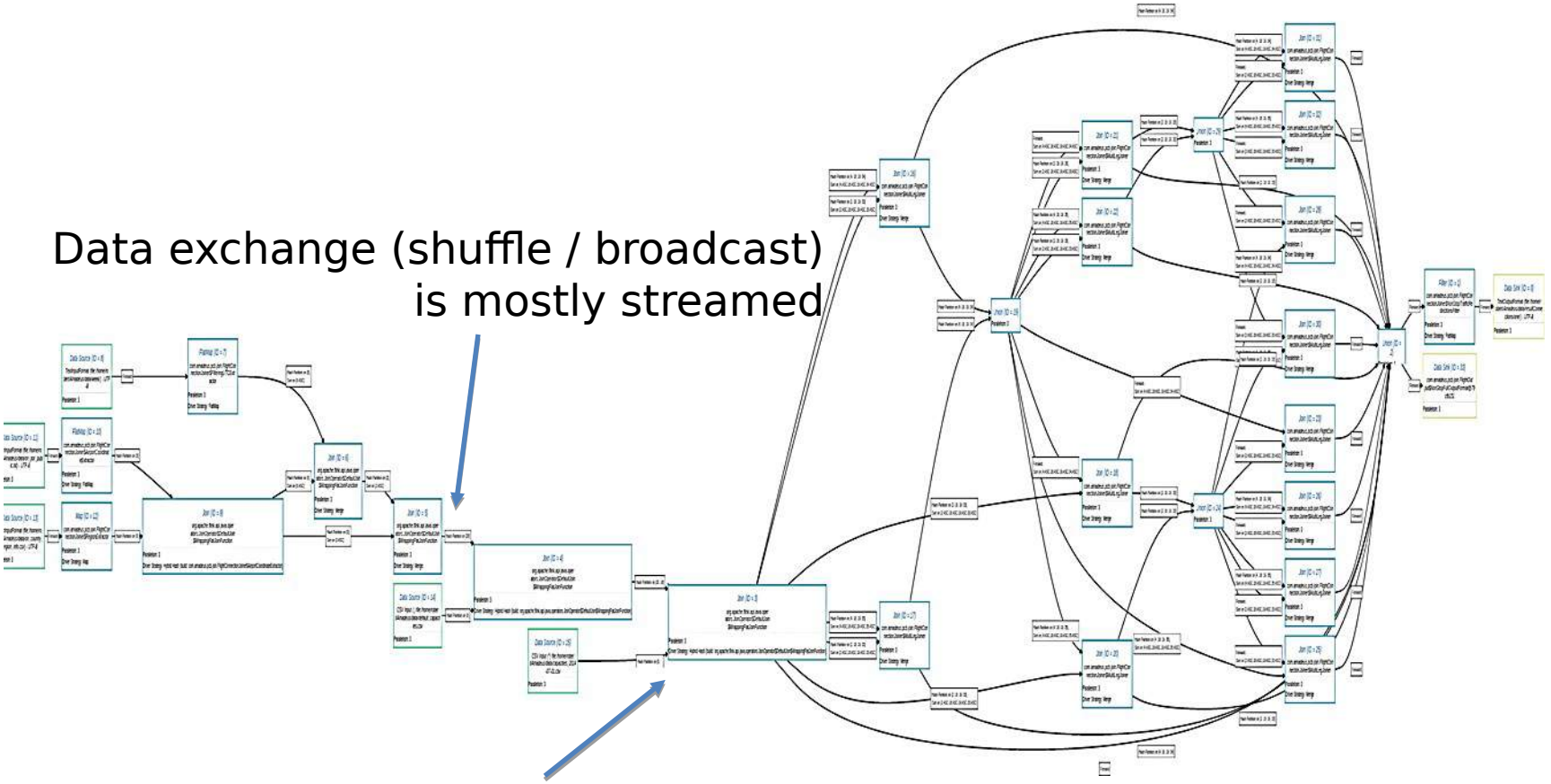


Batch on Streaming

- Batch programs are a special kind of streaming program

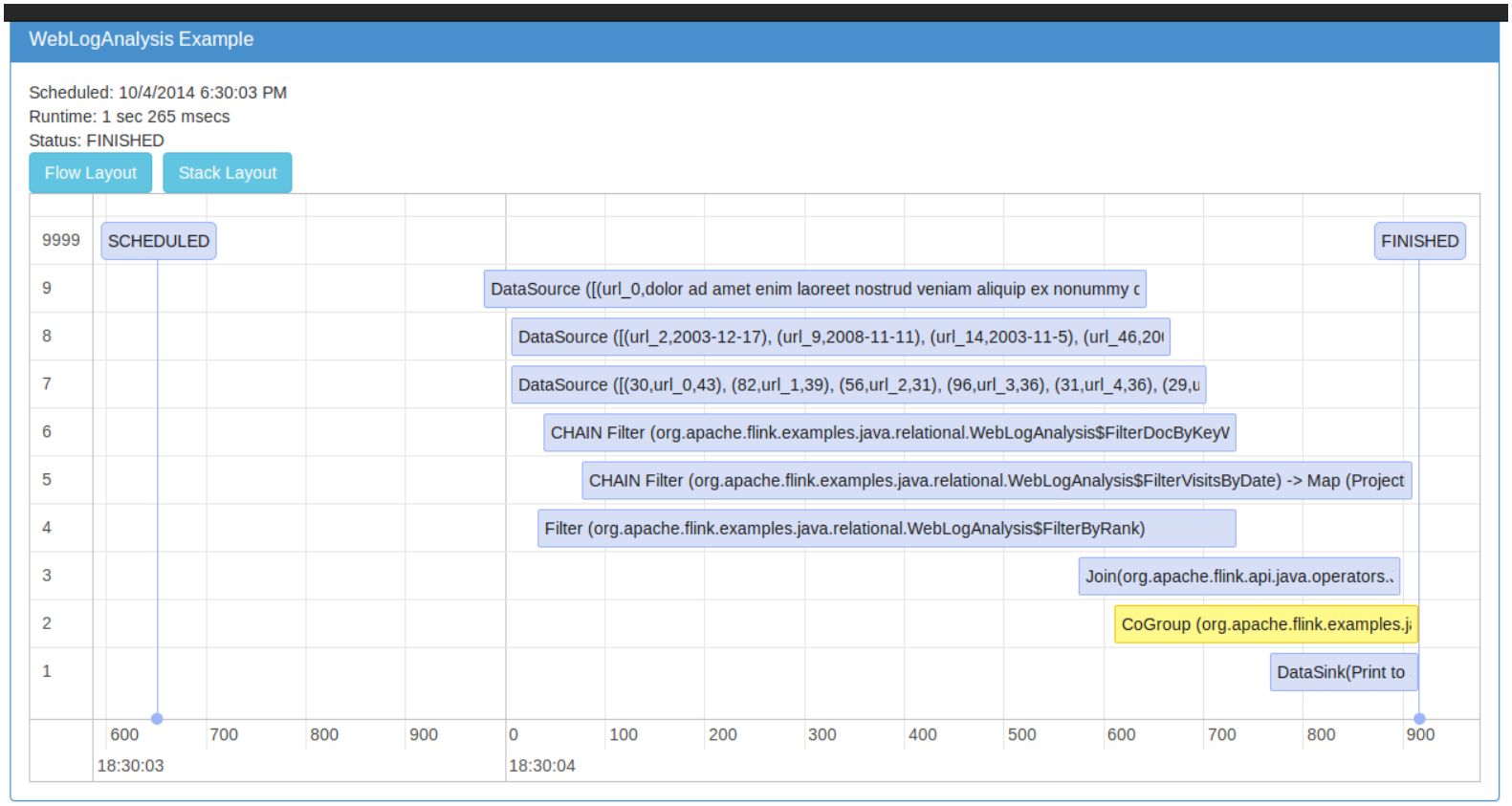


Batch Pipelines

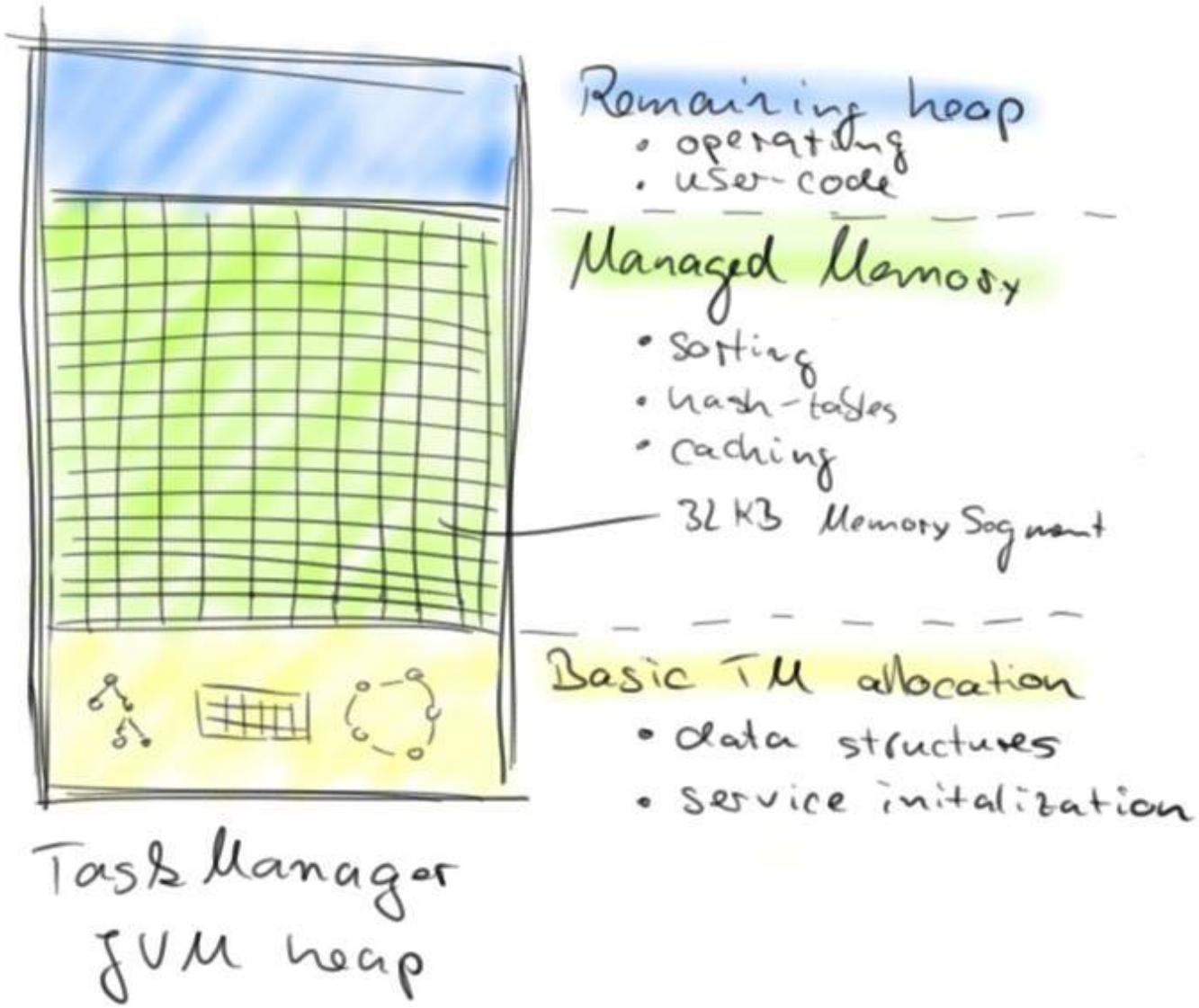


Some operators block (e.g. sorts / hash tables)

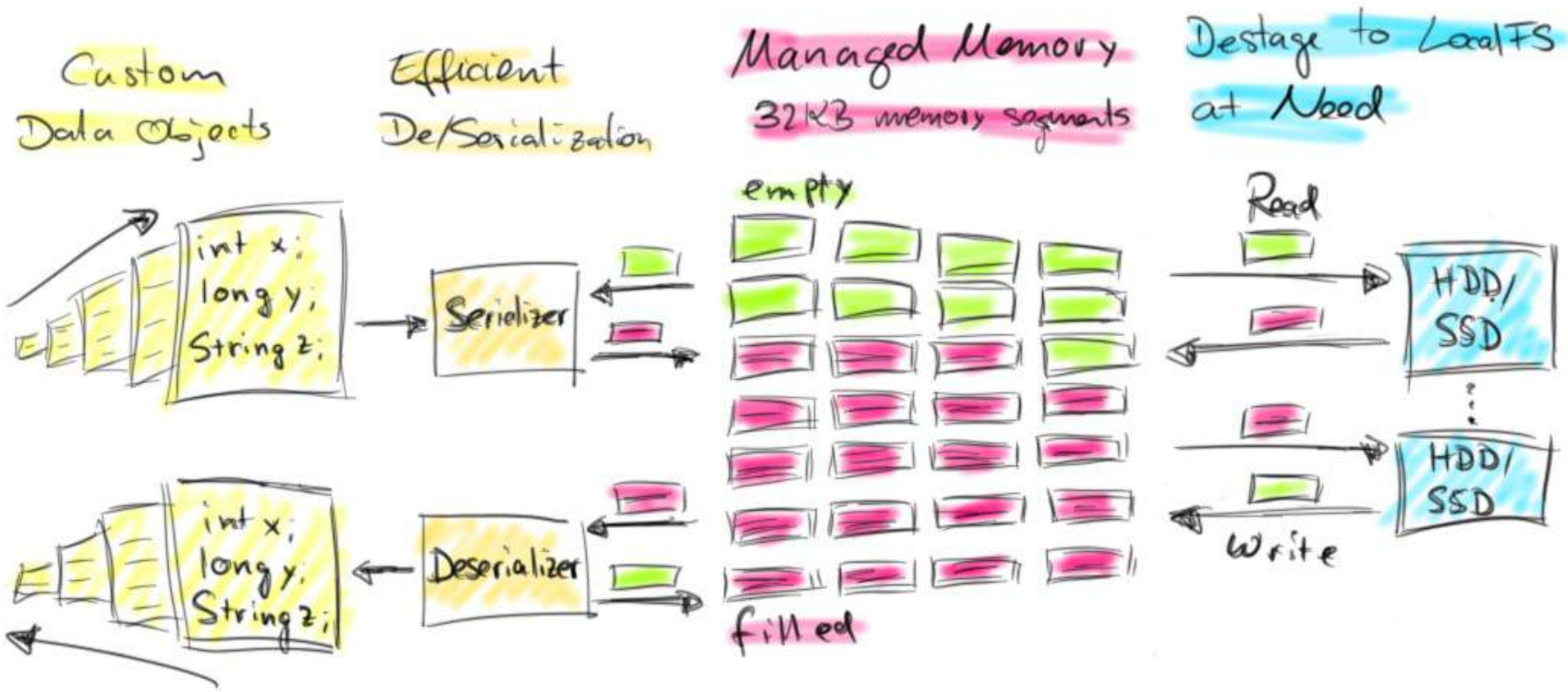
Operators Execution Overlaps



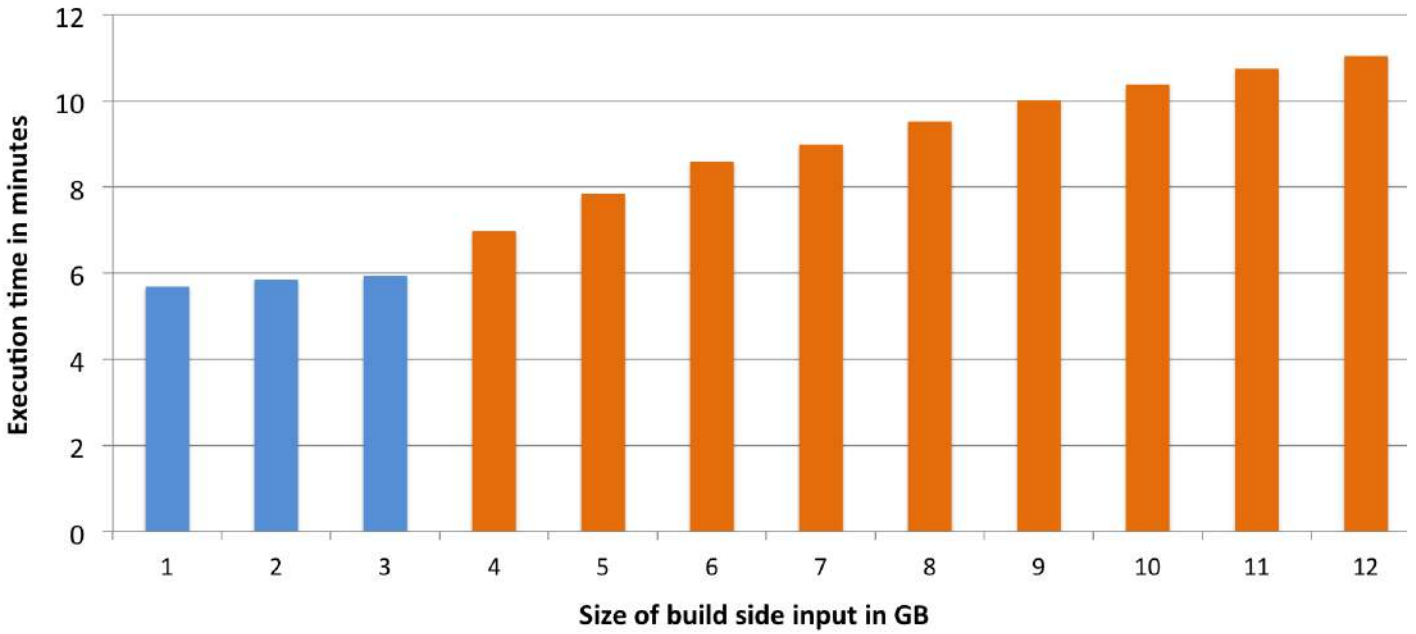
Memory Management



Memory Management



Smooth out-of-core performance



Blue bars are in-memory, orange bars (partially) out-of-co

t: http://flink.apache.org/news/2015/03/13/peeking-into-Apache-Flinks-Engine-Room.html

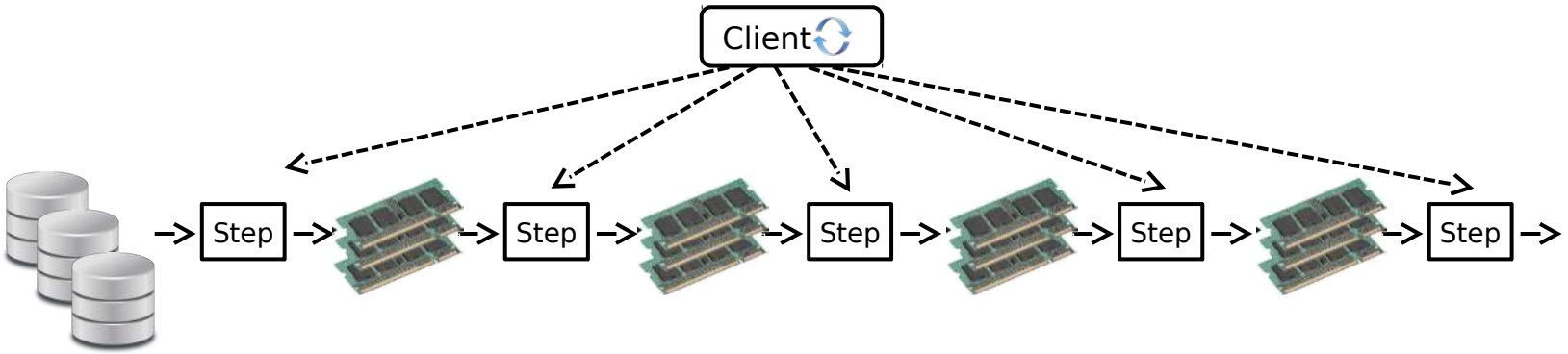
Table API

```
val customers = env.readCsvFile(...).as('id, 'mktSegment)
    .filter("mktSegment = AUTOMOBILE")

val orders = env.readCsvFile(...)
    .filter( o => dateFormat.parse(o.orderDate).before(date) )
    .as("orderId, custId, orderDate, shipPrio")

val items = orders
    .join(customers).where("custId = id")
    .join(lineitems).where("orderId = id")
    .select("orderId, orderDate, shipPrio,
      extdPrice * (Literal(1.0f) - discount) as revenue")

val result = items
    .groupBy("orderId, orderDate, shipPrio")
    .select('orderId, revenue.sum, orderDate, shipPrio")
```

Iterative data flows

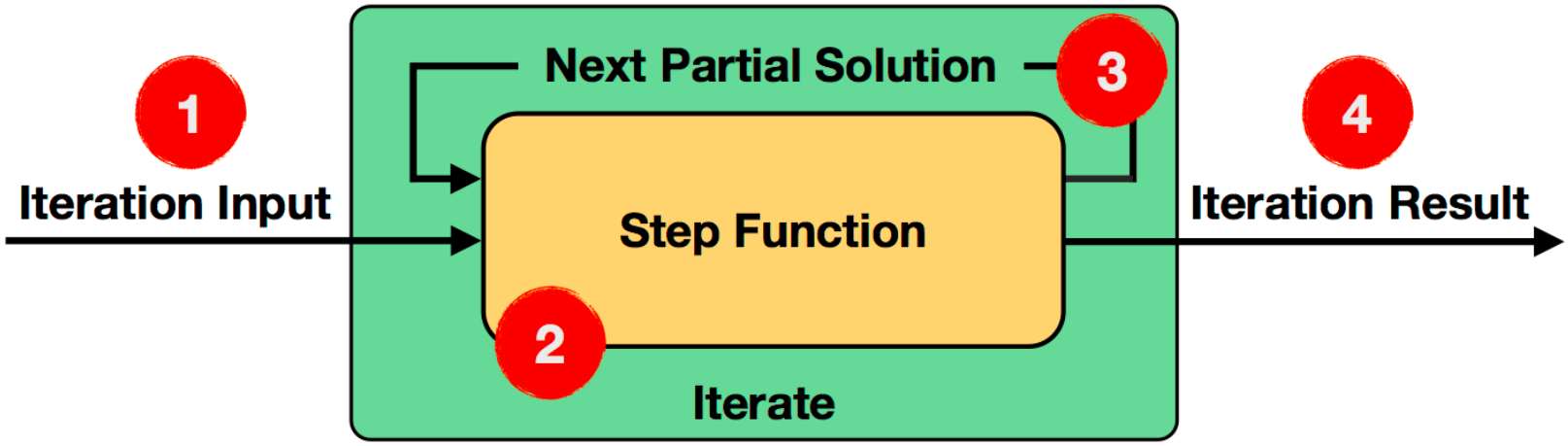
Machine Learning Algorithms

Iterate by looping



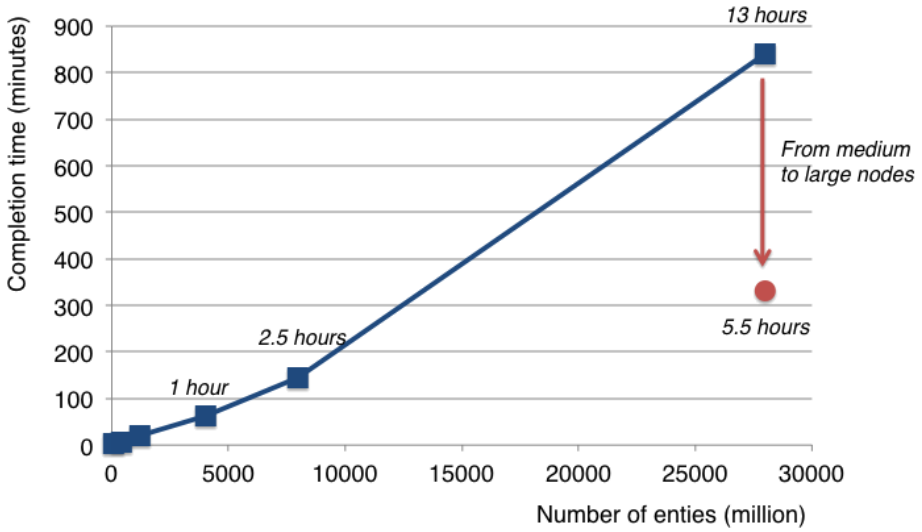
- for/while loop in client submits one job per iteration step
- Data reuse by caching in memory and/or disk

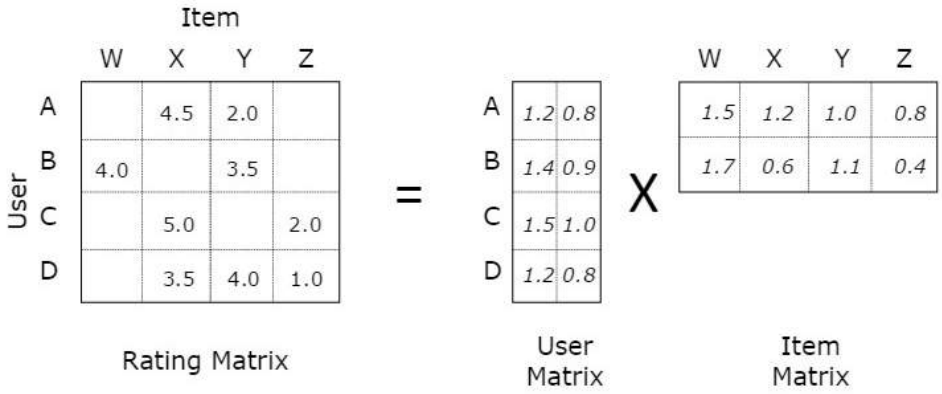
Iterate in the Dataflow



Example: Matrix Factorization

Factorizing a matrix
with
28 billion ratings for
recommendations



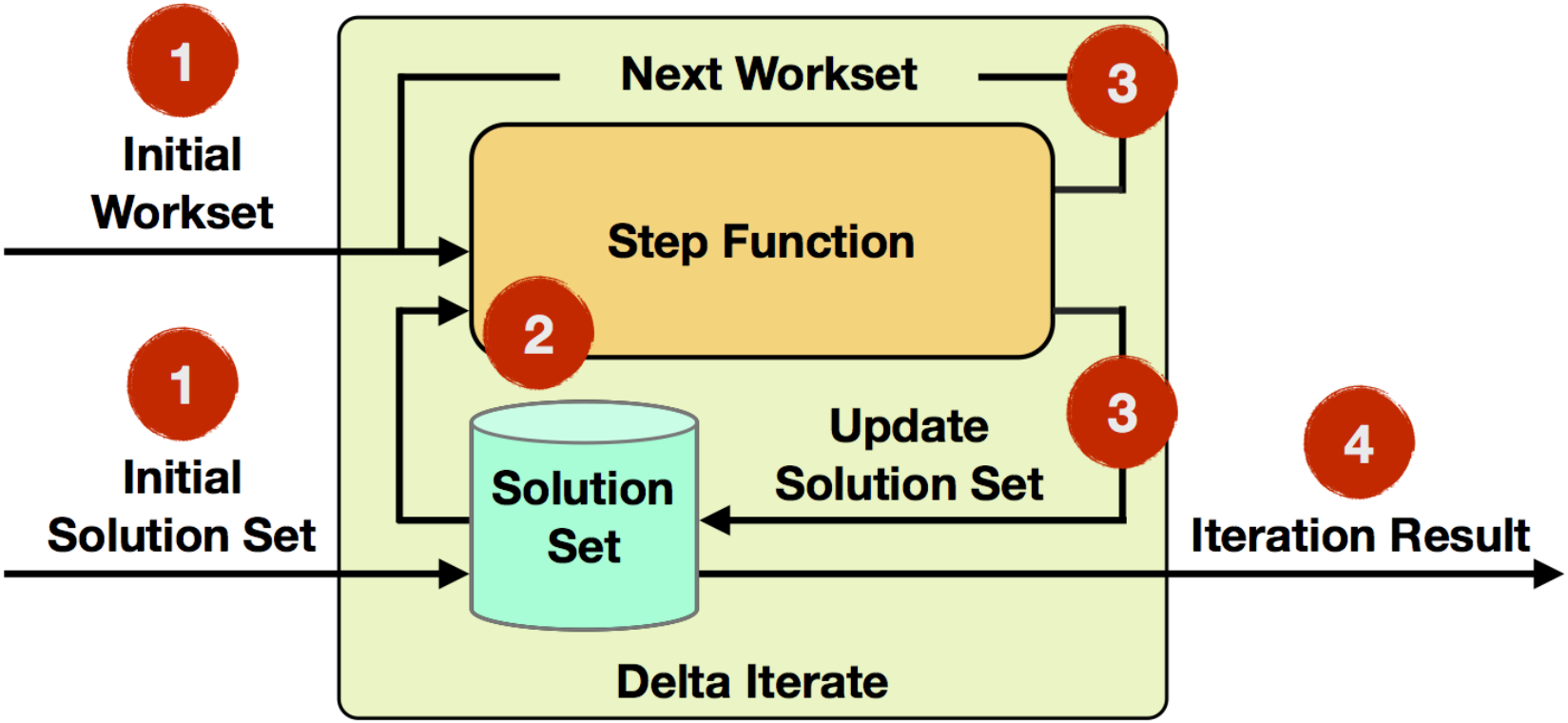


e at: http://data-artisans.com/computing-recommendations-with-flink.html

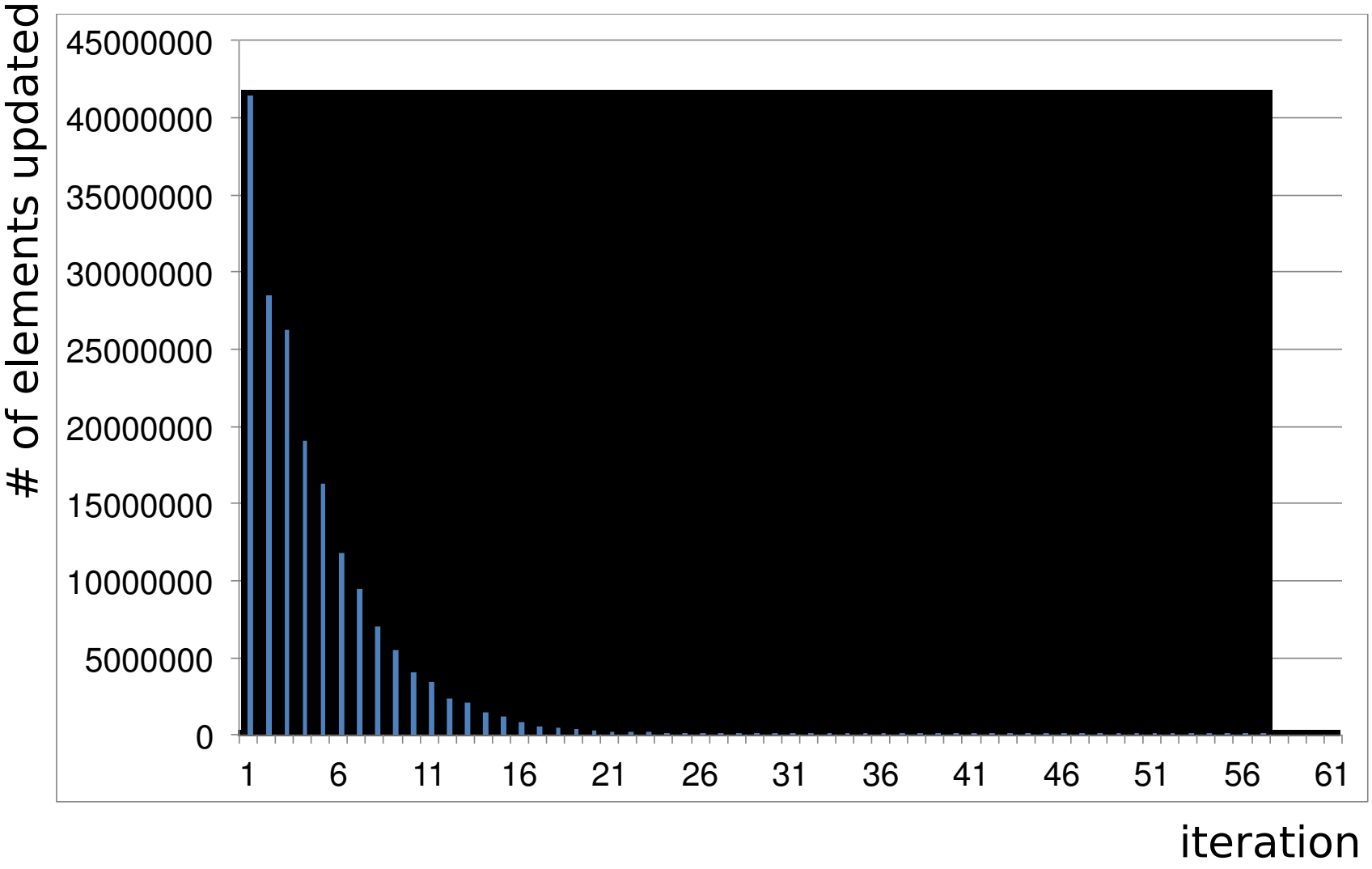
Stateful Iterations

Graph Analysis

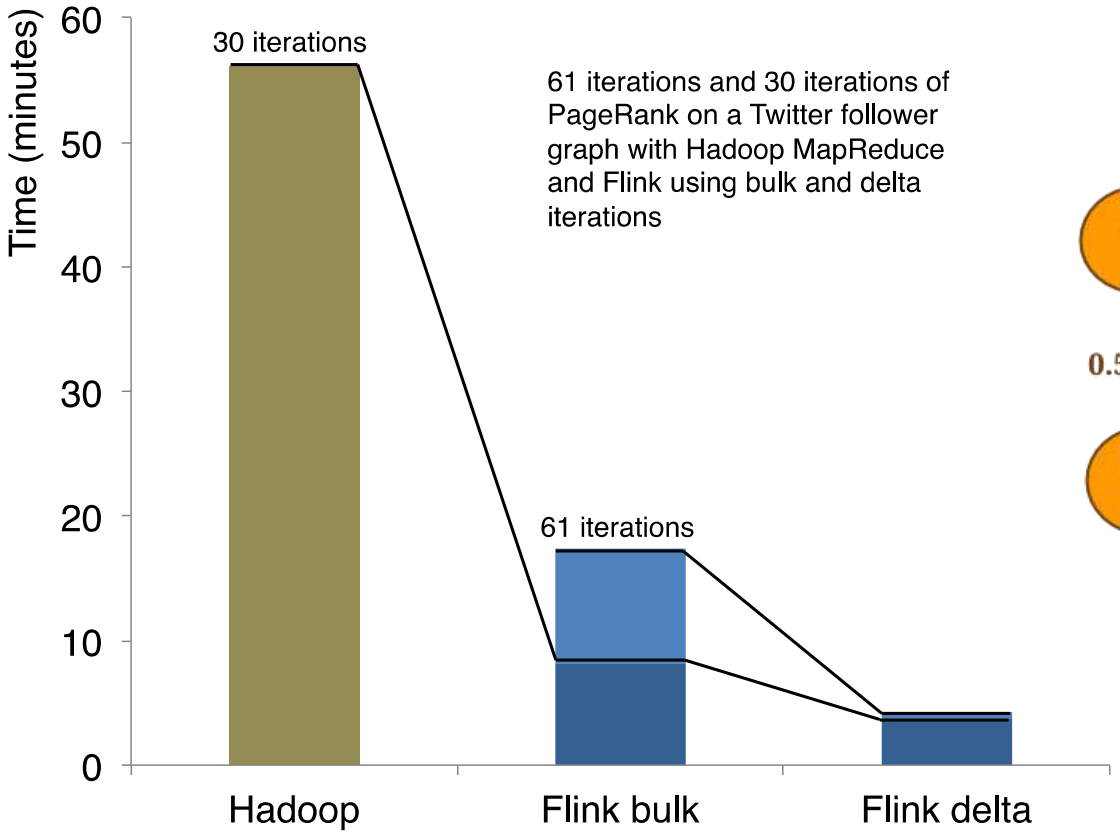
Iterate natively with state/deltas

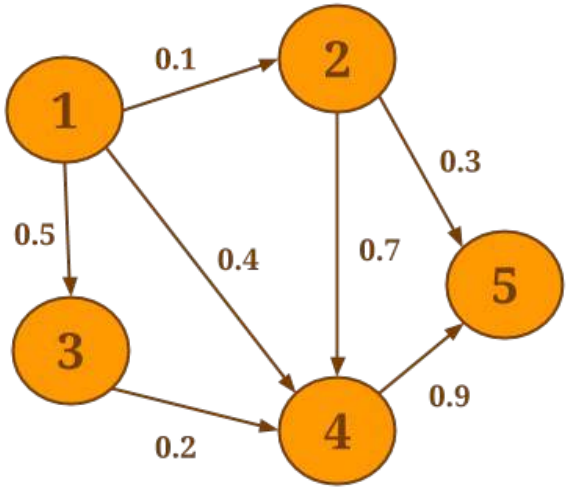


Effect of delta iterations...



… fast graph analysis





More at: http://data-artisans.com/data-analysis-with-flink.html

Closing

Flink Roadmap for 2015

Some highlights that we are working on

- More flexible state and state backends in streaming
- Master Failover
- Improved monitoring
- Integration with other Apache projects
  - SAMOA, Zeppelin, Ignite
- More additions to the libraries

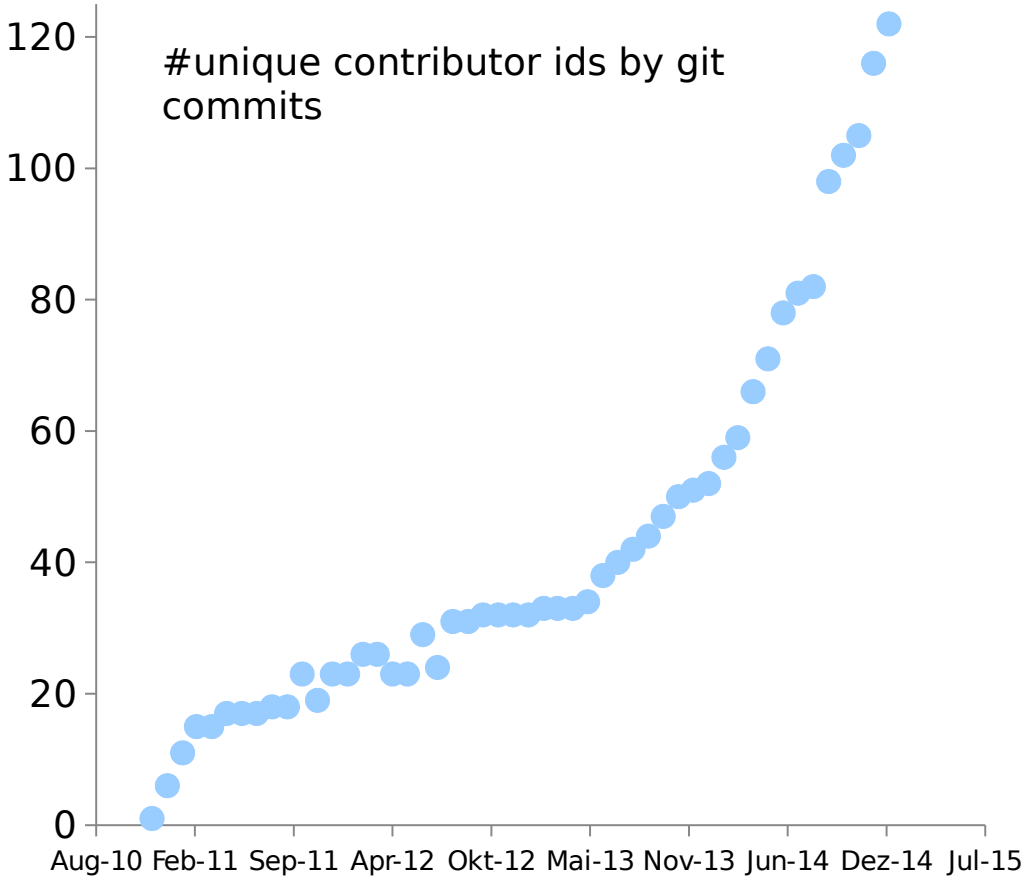
Flink Forward

BERLIN 12/13 OCT 2015

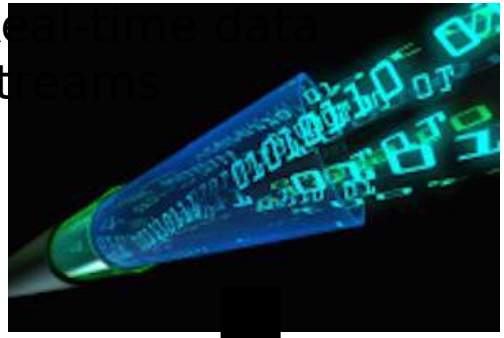
flink.apache.org
@ApacheFlink

Backup

Flink community



What is Apache Flink?



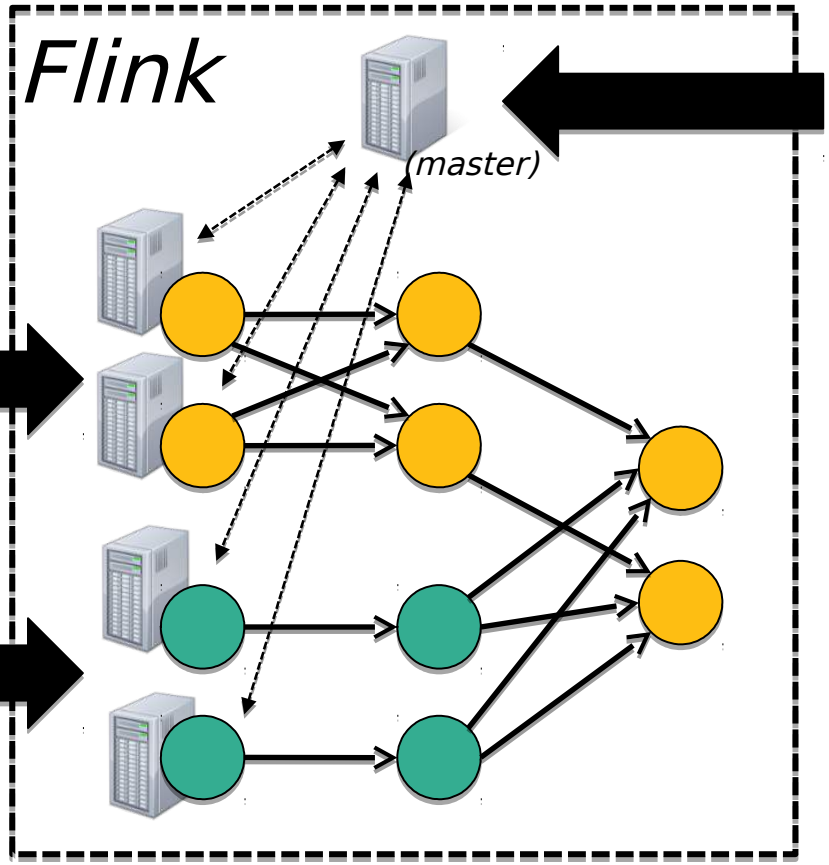
Real-time data streams

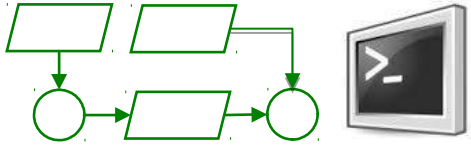
Event logs

Kafka, RabbitMQ, ...

Historic data

HDFS, JDBC, ...





ETL, Graphs,
Machine Learning
Relational, ...

Low latency,
windowing,
aggregations, ...

Cornerpoints of Flink Design

Flexible Data Streaming Engine

→ *Low Latency Steam Proc.*
→ *Highly flexible windows*

Robust Algorithms on Managed Memory

No OutOfMemory Errors
→ *Scales to very large JVMs*
→ *Efficient an robust processing*

High-level APIs, beyond key/value pairs

→ *Java/Scala/Python (upcoming)*
→ *Relational-style optimizer*

Pipelined Execution of Batch Programs

→ *Better shuffle performance*
→ *Scales to very large groups*

Active Library Development

→ *Graphs / Machine Learning*
→ *Streaming ML (coming)*

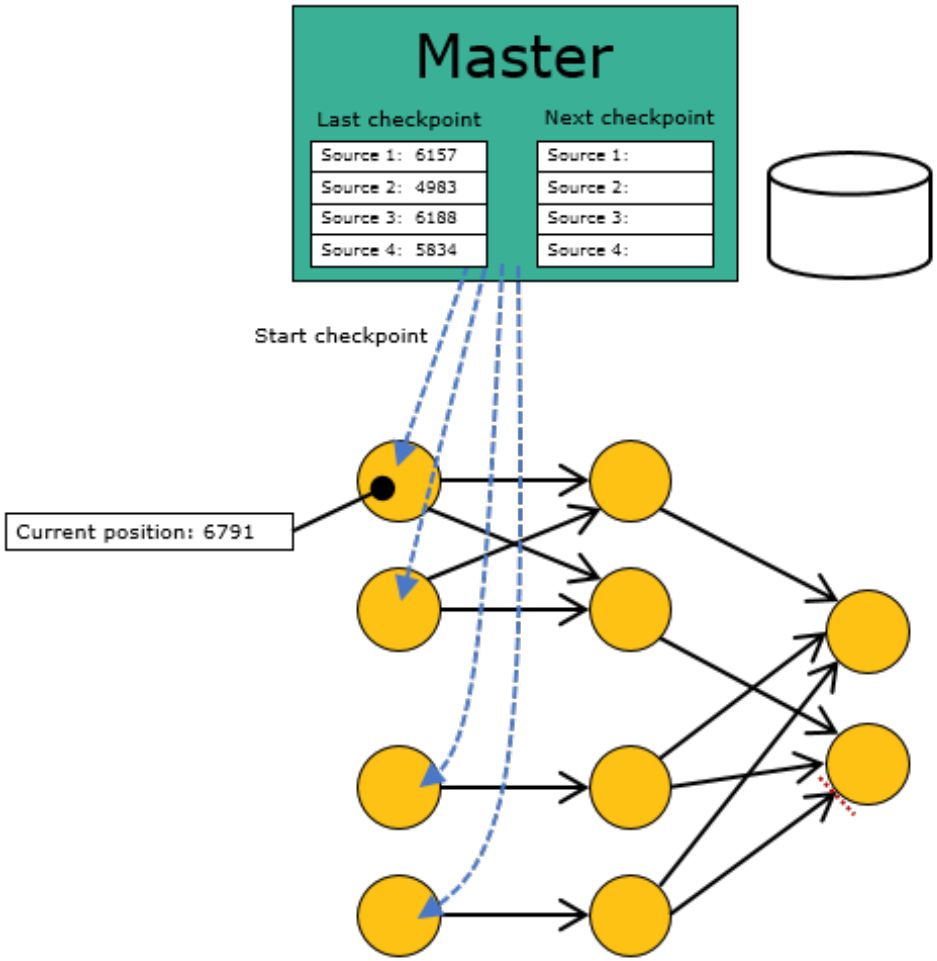
Native Iterations

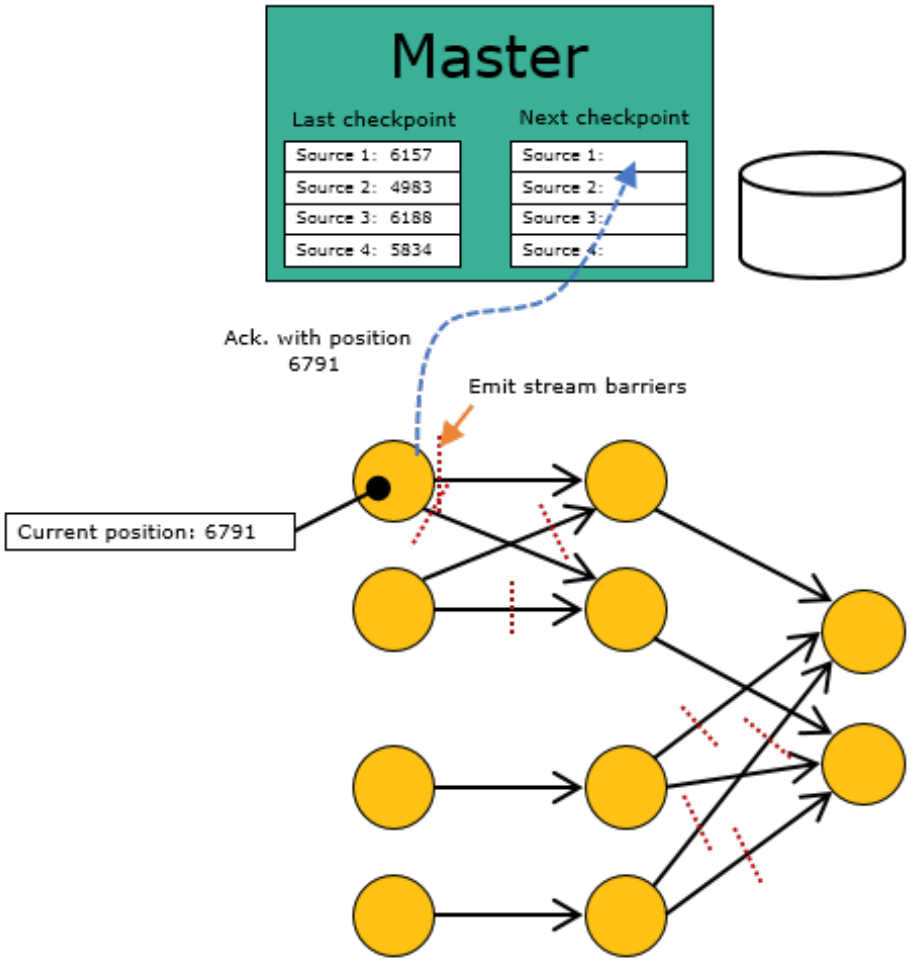
→ *Very fast Graph Processing*
→ *Stateful Iterations for ML*

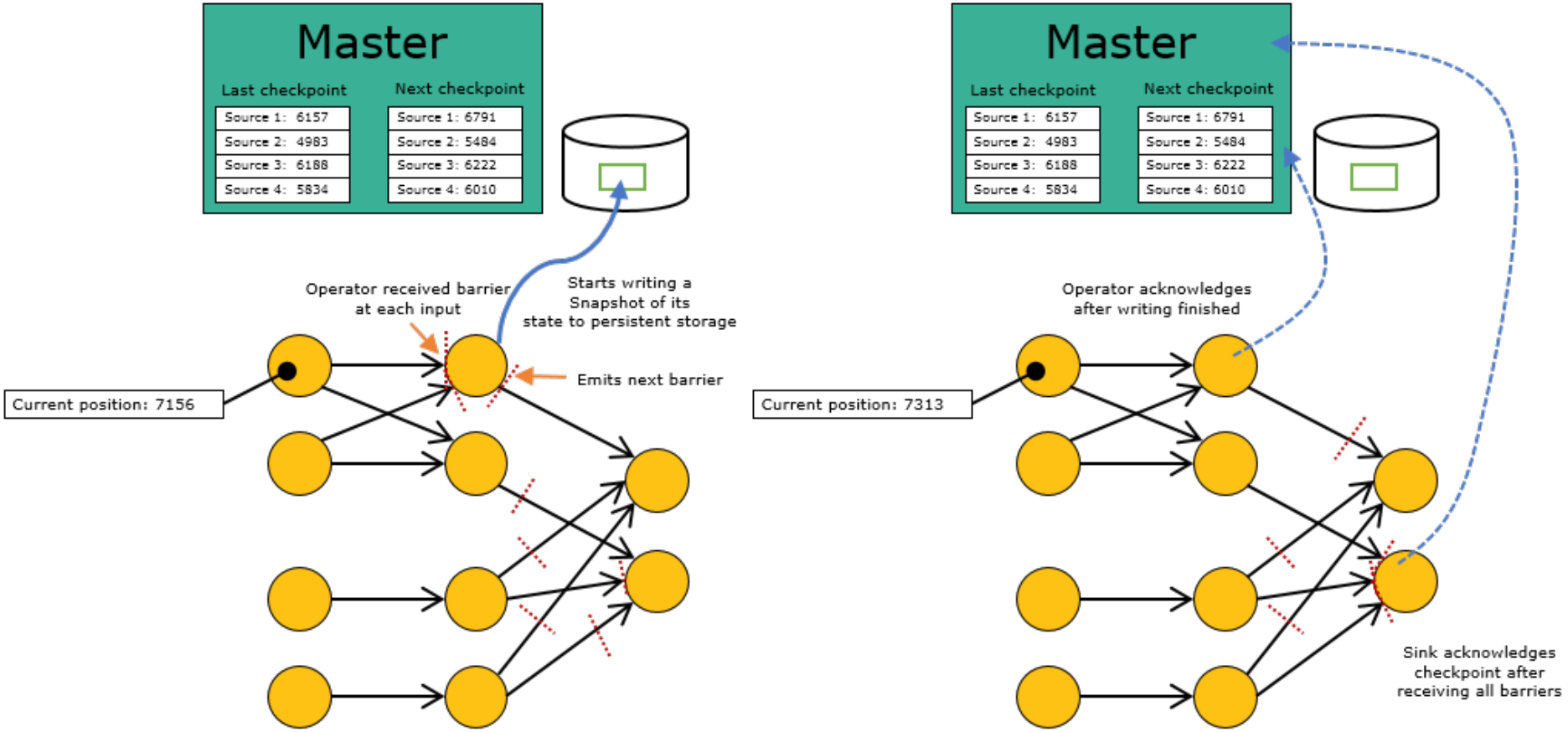
Defining windows in Flink

- Trigger policy
  - When to trigger the computation on current window
- Eviction policy
  - When data points should leave the window
  - Defines window width/size
- E.g., count-based policy
  - evict when #elements > n
  - start a new window every n-th element
- Built-in: Count, Time, Delta policies

Streaming checkpoints


Master
Last checkpoint
Next checkpoint
Source 1: 6157
Source 2: 4983
Source 3: 6188
Source 4: 5834
Source 1:
Source 2:
Source 3:
Source 4:
Start checkpoint
Current position: 6791


Master
Last checkpoint
Next checkpoint
Source 1: 6157
Source 2: 4983
Source 3: 6188
Source 4: 5834
Source 1:
Source 2:
Source 3:
Source 4:
Ack. with position 6791
Emit stream barriers
Current position: 6791

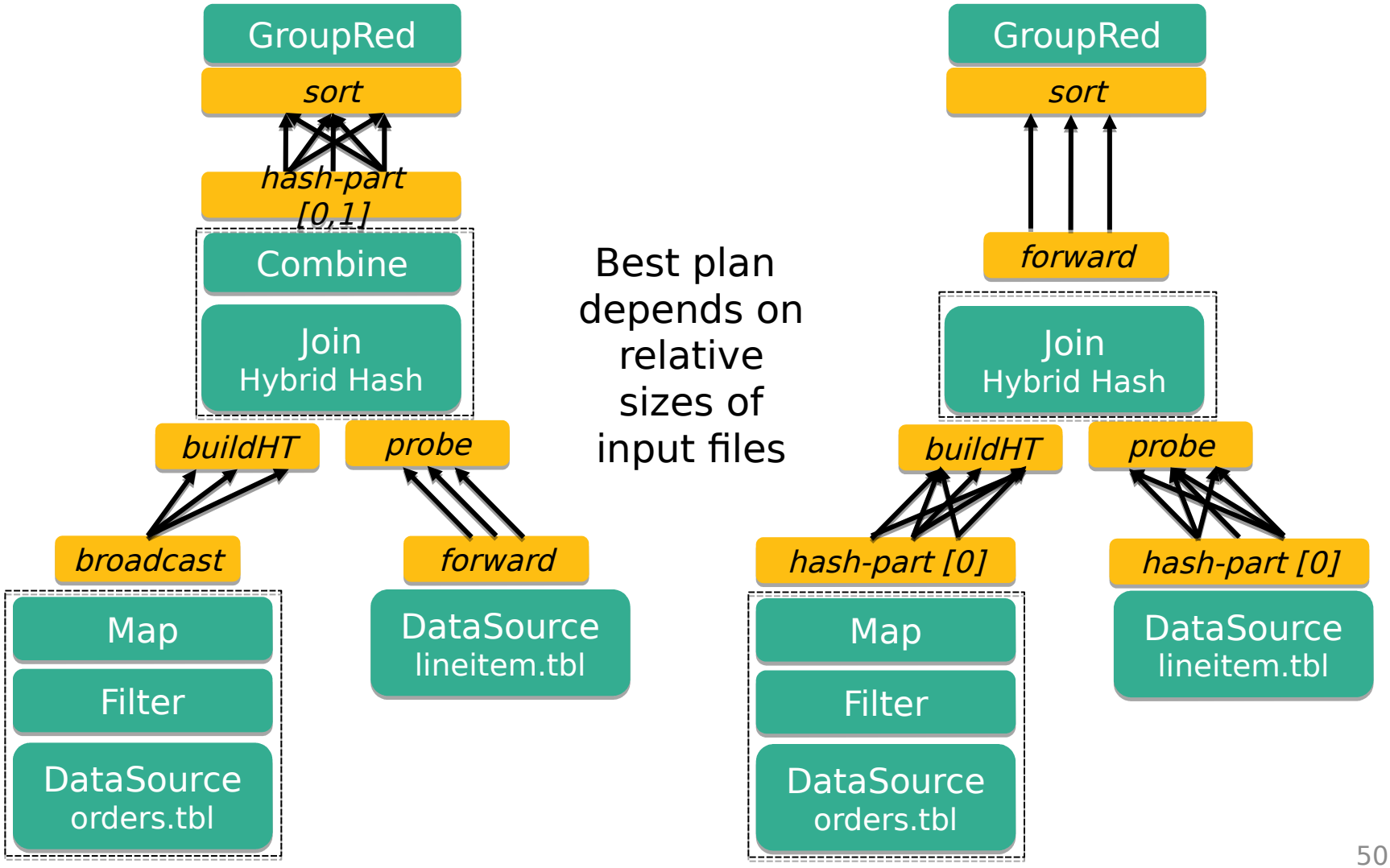

Master
Last checkpoint
Next checkpoint
Source 1: 6157
Source 2: 4983
Source 3: 6188
Source 4: 5834
Source 1: 6791
Source 2: 5484
Source 3: 6222
Source 4: 6010
Operator received barrier at each input
Starts writing a Snapshot of its state to persistent storage
Emits next barrier
Current position: 7156
Master
Last checkpoint
Next checkpoint
Source 1: 6157
Source 2: 4983
Source 3: 6188
Source 4: 5834
Source 1: 6791
Source 2: 5484
Source 3: 6222
Source 4: 6010
Operator acknowledges after writing finished
Current position: 7313
Sink acknowledges checkpoint after receiving all barriers

Program optimization

A simple program

```
val orders = …
val lineitems = …

val filteredOrders = orders
  .filter(o => dataFormat.parse(l.shipDate).after(date))
  .filter(o => o.shipPrio > 2)

val lineitemsOfOrders = filteredOrders
  .join(lineitems)
  .where(“orderId”).equalTo(“orderId”)
  .apply((o,l) => new SelectedItem (o.orderDate, l.extdPrice))

val priceSums = lineitemsOfOrders
  .groupBy(“orderDate”).sum (“l.extdPrice”);
```

Two execution plans

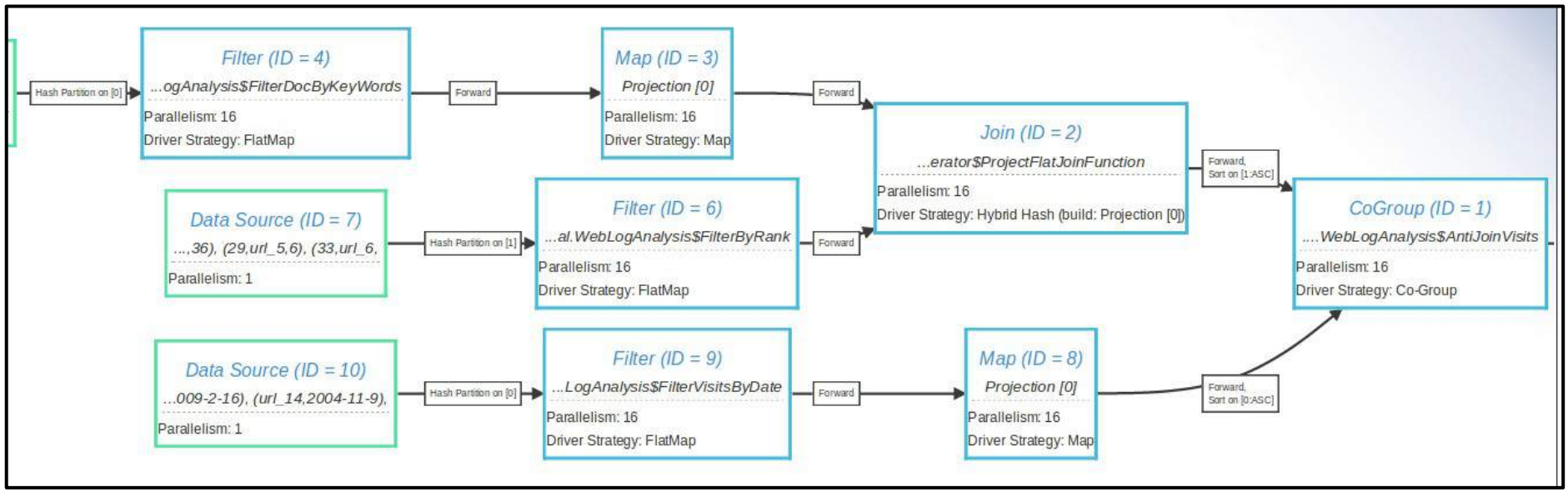


Examples of optimization

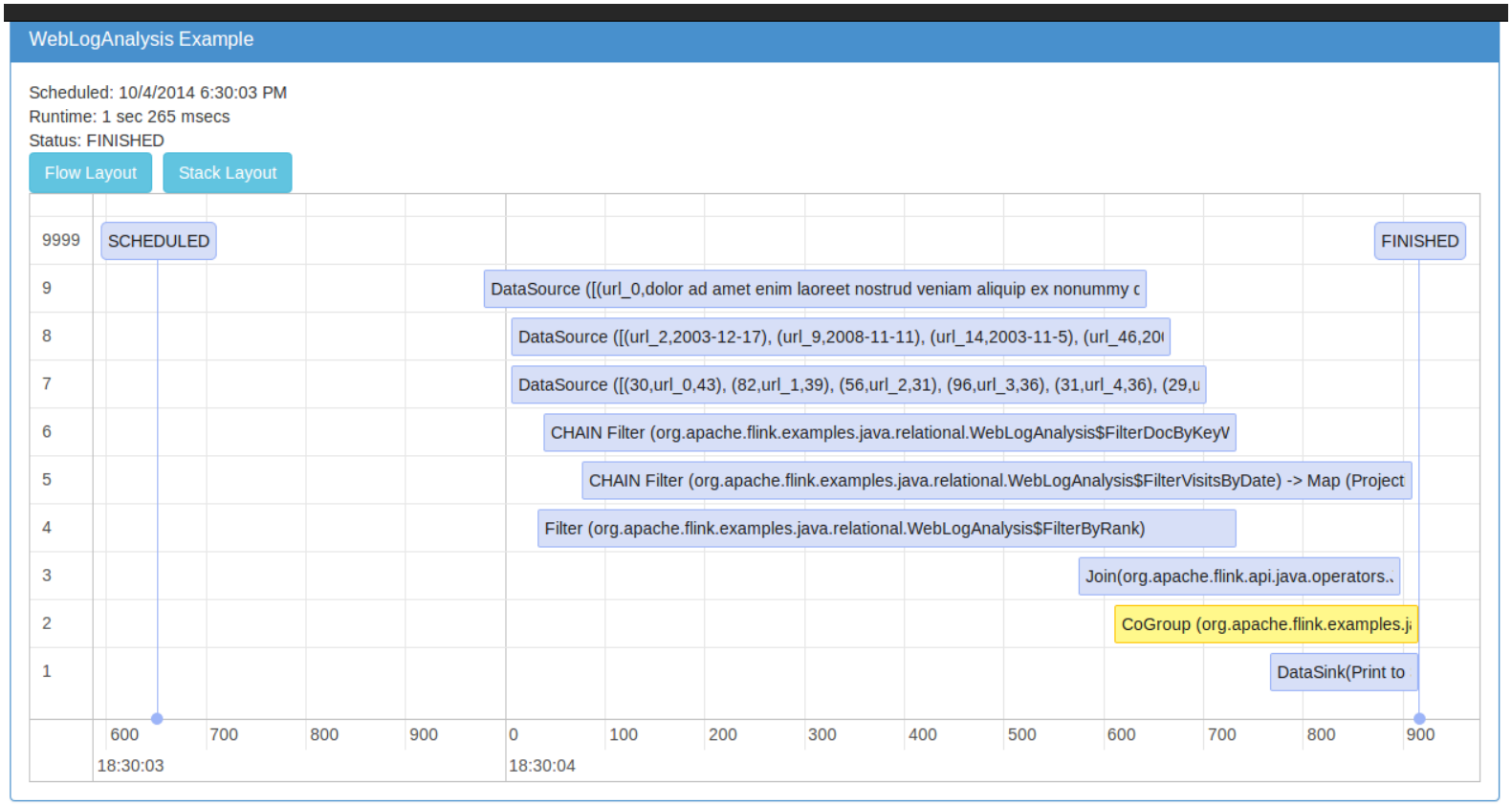
- Task chaining
  - Coalesce map/filter/etc tasks
- Join optimizations
  - Broadcast/partition, build/probe side, hash or sort-merge
- Interesting properties
  - Re-use partitioning and sorting for later operations
- Automatic caching
  - E.g., for iterations

Visualization

Visualization tools



Visualization tools



Visualization tools