Apache Flink
Streaming Done Right

Till Rohrmann
trohrmann@apache.org
@stsffap

dataArtisans

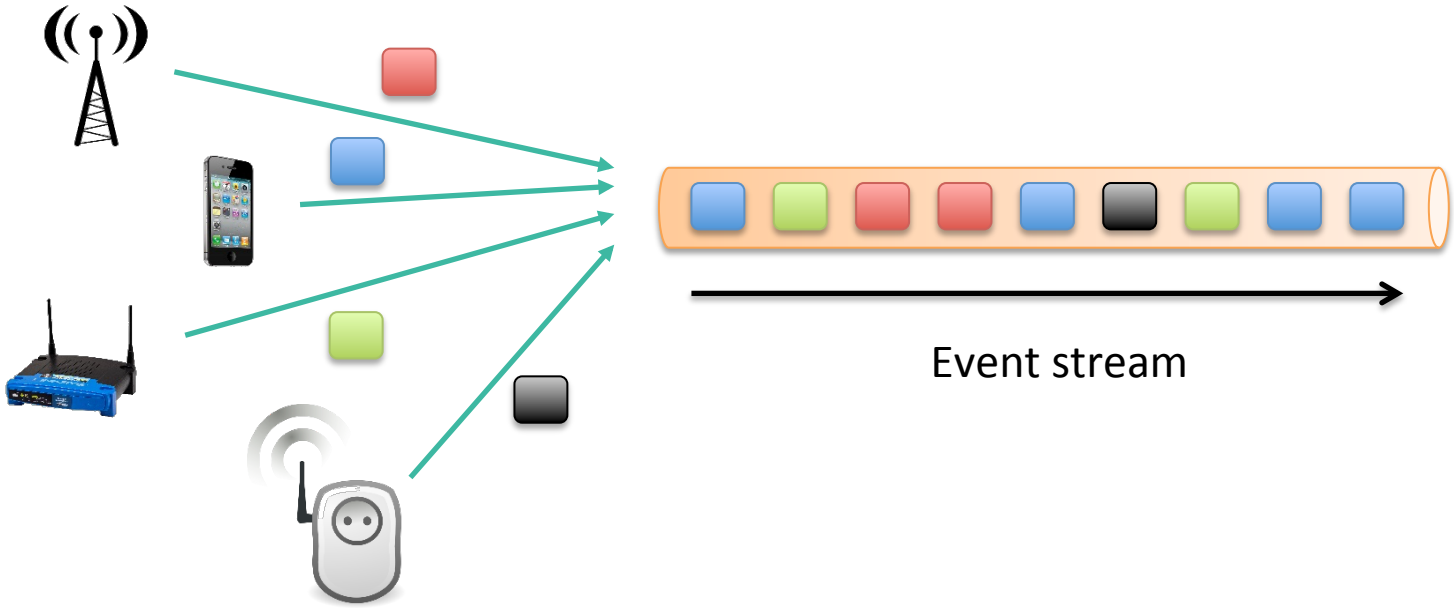
What Is Apache Flink?

- Apache TLP since December 2014

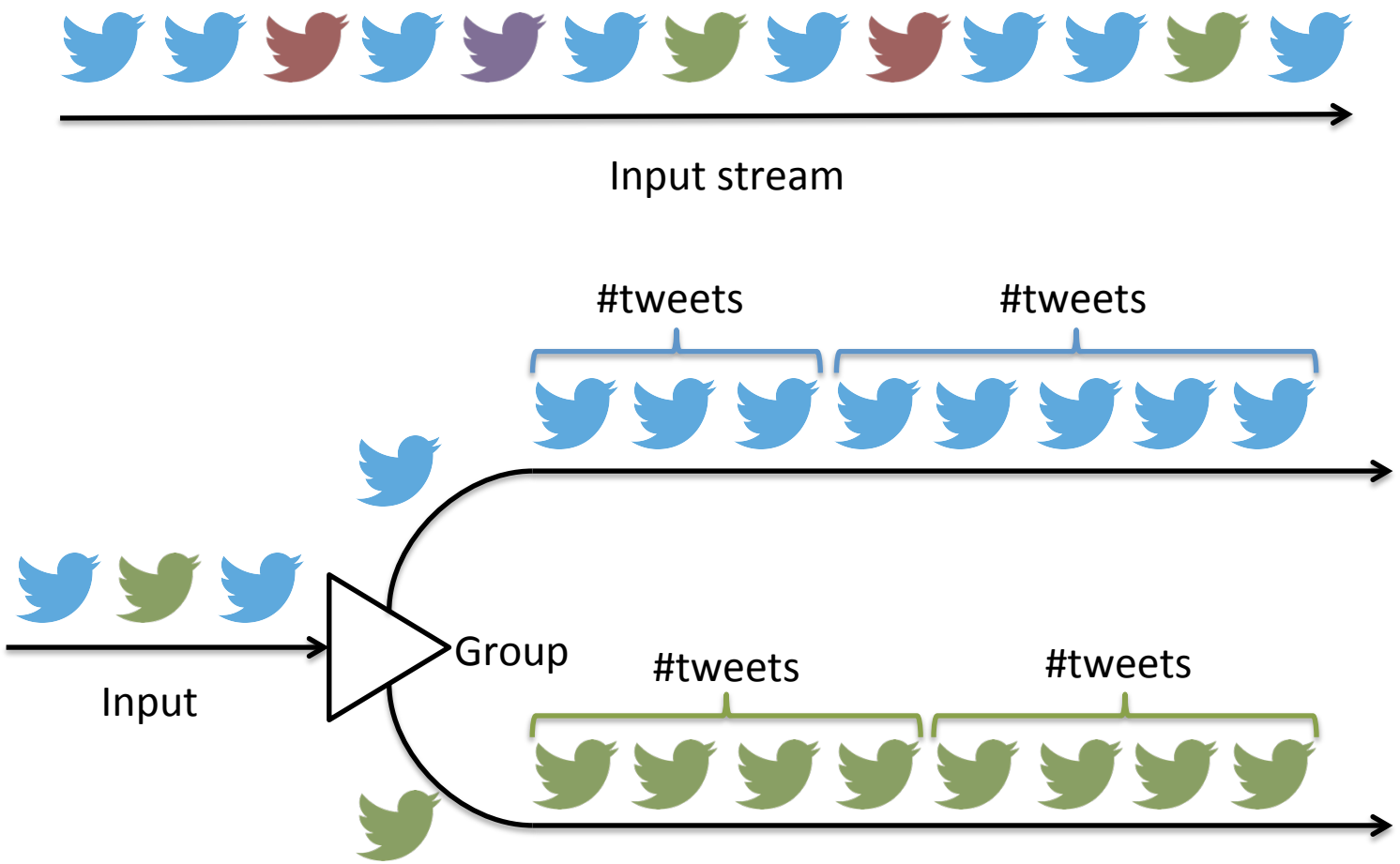
- Parallel streaming data flow runtime
- Low latency & high throughput
- Exactly once semantics
- Stateful operations

Why Stream Processing?

- Most problems have streaming nature
- Stream processing gives lower latency
- Data volumes more easily tamed
- More predictable resource consumption



Counting Tweet Impressions



How Would I Do It with Flink?

```
case class Tweet(id: Long, timestamp: Long, count: Long)

val env = StreamExecutionEnvironment.getEnvironment()

val tweets: DataStream[Tweet] = env.addSource(
  new MyTwitterSource())

val result: DataStream[Tweet] = tweets
  .keyBy("id")
  .timeWindow(Time.minutes(10))
  .sum("count")

result.print()

env.execute()
```

How Would I Do It with Flink?

```
case class Tweet(id: Long, timestamp: Long, count: Long)

val env = StreamExecutionEnvironment.getEnvironment()

val tweets: DataStream[Tweet] = env.addSource(
  new MyTwitterSource())

val result: DataStream[Tweet] = tweets
  .keyBy("id")
  .timeWindow(Time.minutes(10))
  .sum("count")

result.print()

env.execute()
```

How Would I Do It with Flink?

```
case class Tweet(id: Long, timestamp: Long, count: Long)

val env = StreamExecutionEnvironment.getEnvironment()

val tweets: DataStream[Tweet] = env.addSource(
  new MyTwitterSource())

val result: DataStream[Tweet] = tweets
  .keyBy("id")
  .timeWindow(Time.minutes(10))
  .sum("count")

result.print()

env.execute()
```

How Would I Do It with Flink?

```
case class Tweet(id: Long, timestamp: Long, count: Long)

val env = StreamExecutionEnvironment.getEnvironment()

val tweets: DataStream[Tweet] = env.addSource(
  new MyTwitterSource())

val result: DataStream[Tweet] = tweets
  .keyBy("id")
  .timeWindow(Time.minutes(10))
  .sum("count")

result.print()

env.execute()
```

We have to define a time frame for the aggregation

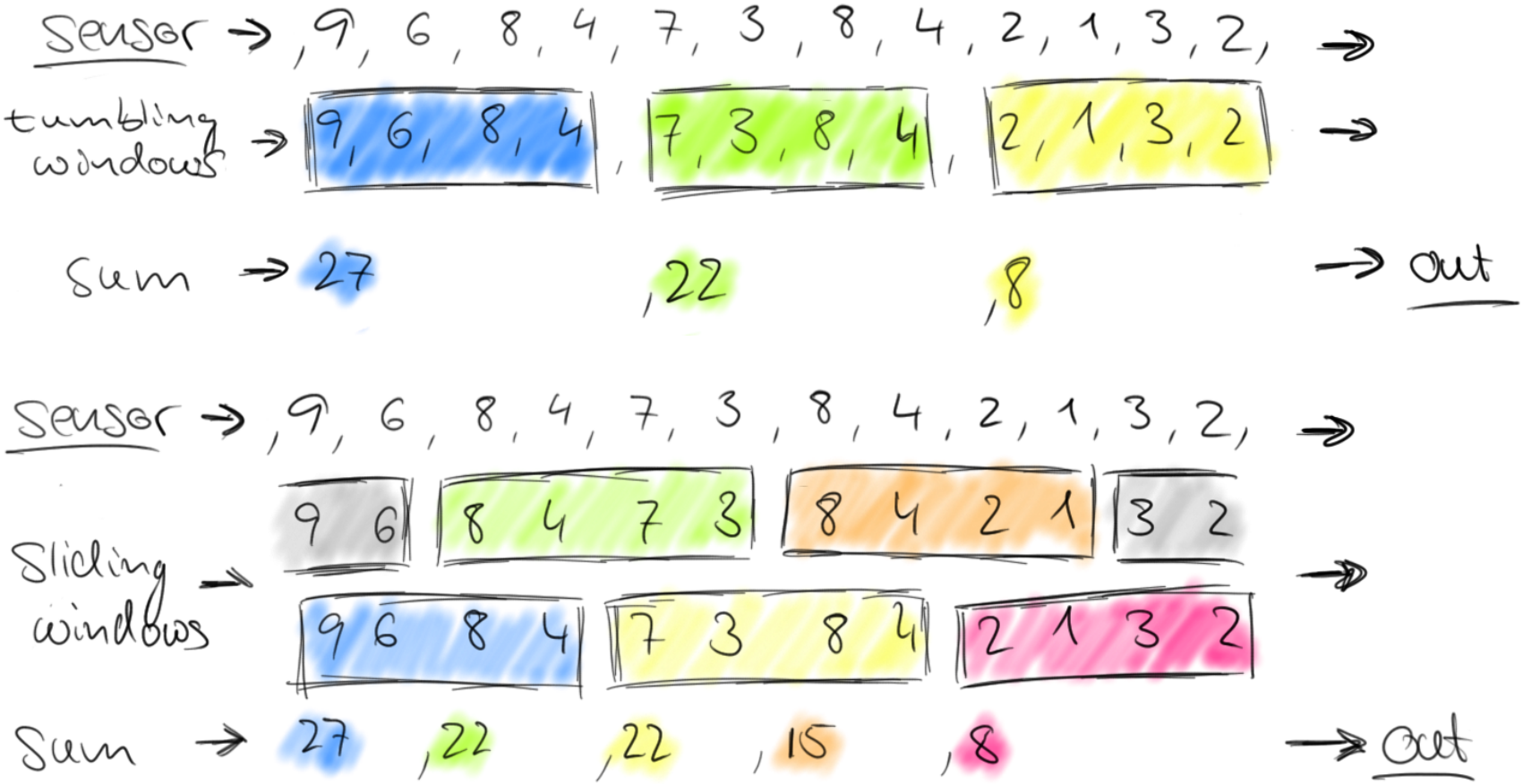
How Would I Do It with Flink?

```
case class Tweet(id: Long, timestamp: Long, count: Long)

val env = StreamExecutionEnvironment.getEnvironment()

val tweets: DataStream[Tweet] = env.addSource(
  new MyTwitterSource())

val result: DataStream[Tweet] = tweets
  .keyBy("id")
  .timeWindow(Time.minutes(10))
  .sum("count")

result.print()

env.execute()
```

How Would I Do It with Flink?

```
case class Tweet(id: Long, timestamp: Long, count: Long)

val env = StreamExecutionEnvironment.getEnvironment()

val tweets: DataStream[Tweet] = env.addSource(
  new MyTwitterSource())

val result: DataStream[Tweet] = tweets
  .keyBy("id")
  .timeWindow(Time.minutes(10))
  .sum("count")

result.print()

env.execute()
```

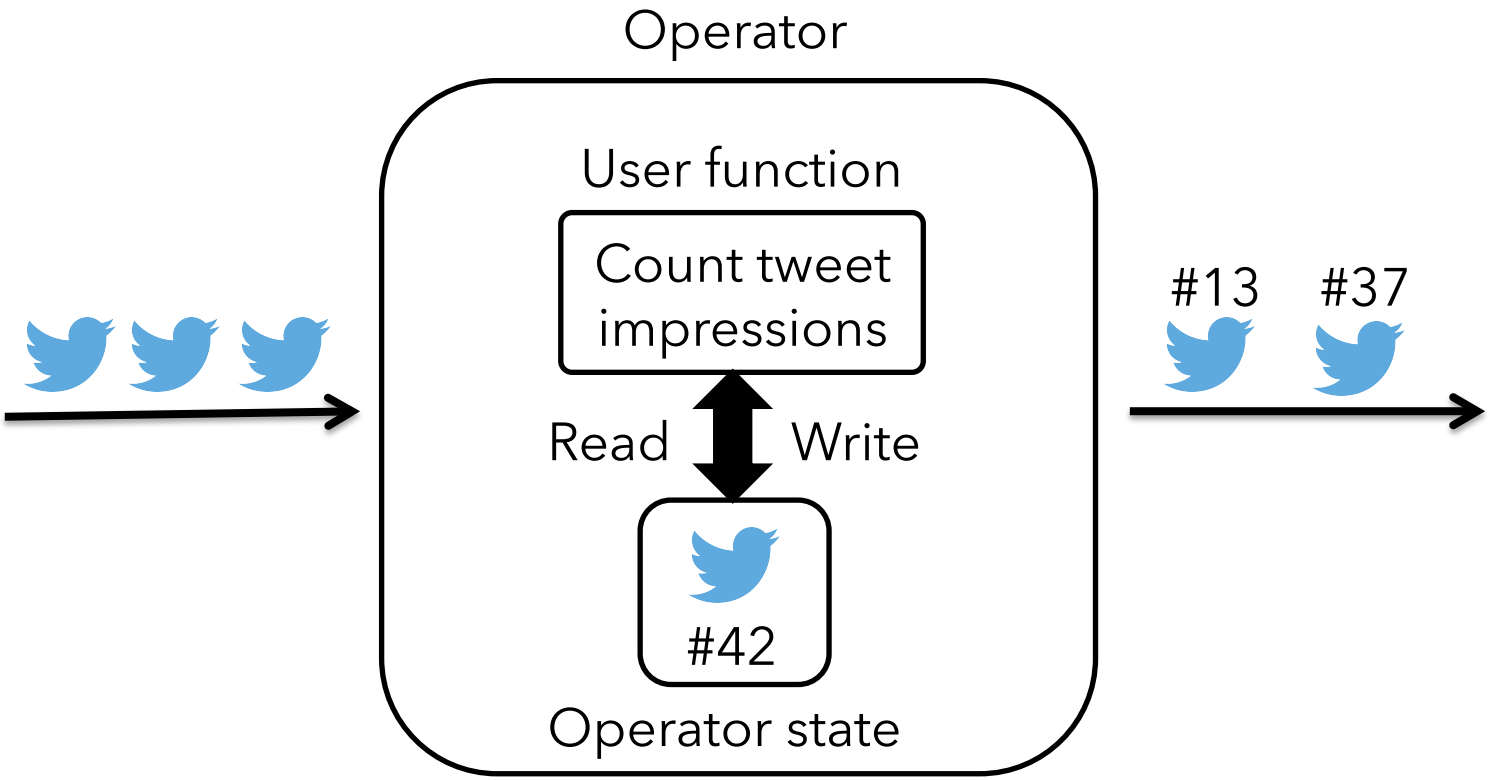
Expressive Windows



Time and count based; Event-time support; Custom windows

Stateful Operators

- What if a window grows too large?
- Solution: Stateful mapper with counter



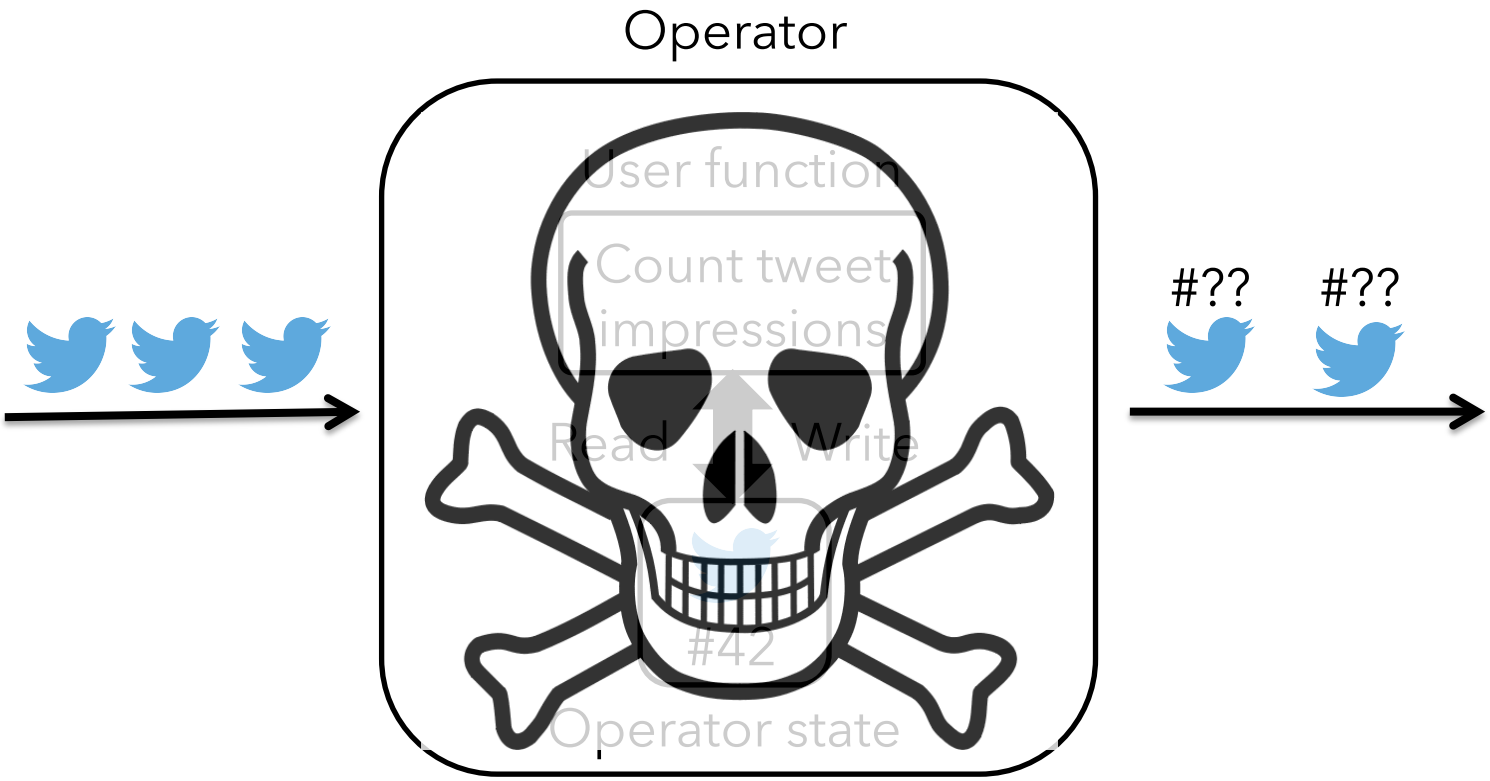
Intuitive API

```
case class Tweet(id: Long, timestamp: Long, count: Long)

val env = StreamExecutionEnvironment.getEnvironment()

val tweets: DataStream[Tweet] = env.addSource(
  new MyTwitterSource())

val result: DataStream[Tweet] = tweets
  .keyBy("id")
  .mapWithState {
    (tweet, state: Option[Long]) =>
      state match {
        case Some(counter) =>
          (tweet.copy(count = counter + 1L), Some(counter + 1L))
        case None => (tweet, Some(1L))
      }
  }
```

Intuitive API

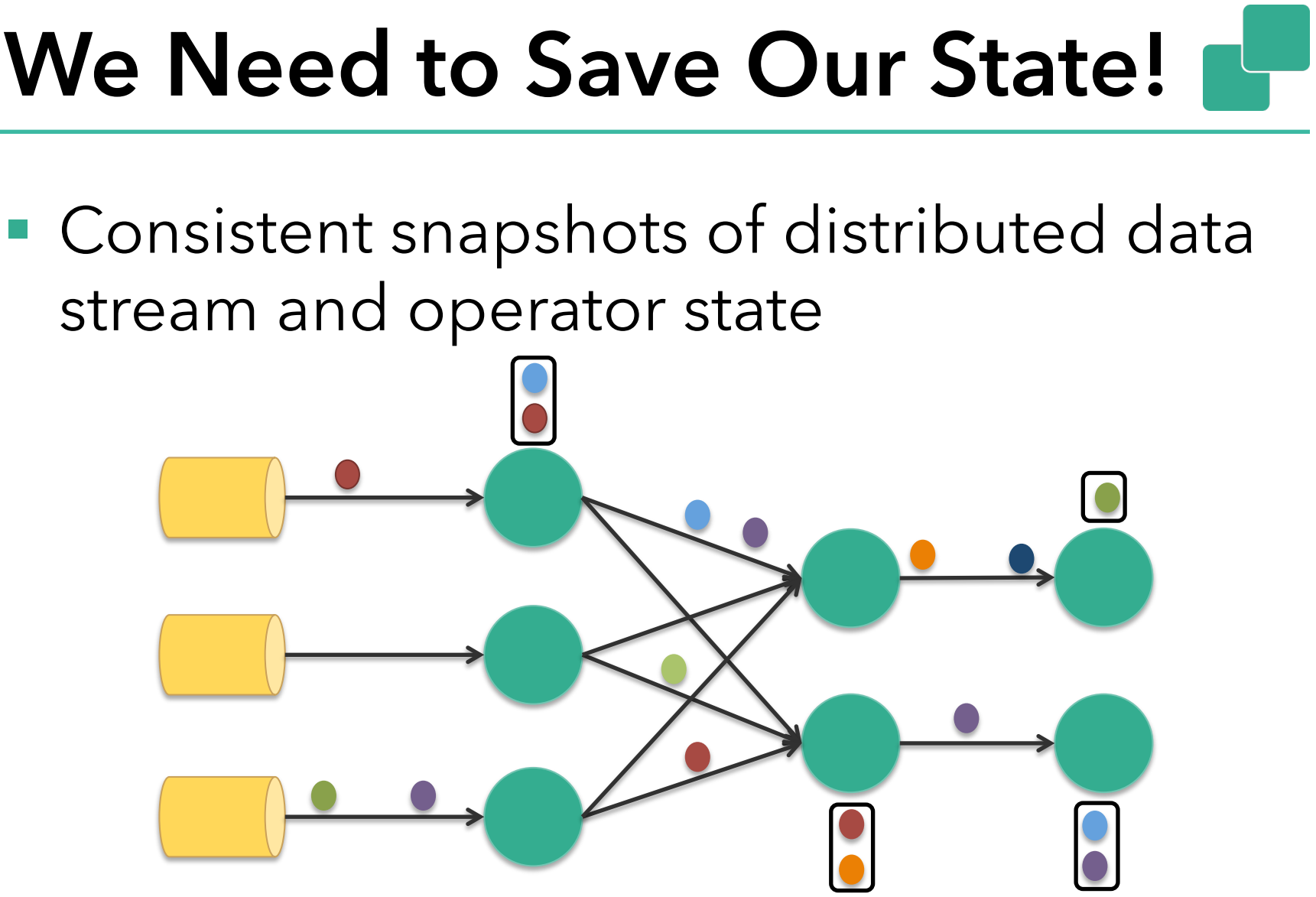
```
case class Tweet(id: Long, timestamp: Long, count: Long)

val env = StreamExecutionEnvironment.getEnvironment()

val tweets: DataStream[Tweet] = env.addSource(
  new MyTwitterSource())

val result: DataStream[Tweet] = tweets
  .keyBy("id")
  .mapWithState {
    (tweet, state: Option[Long]) =>
      state match {
        case Some(counter) =>
          (tweet.copy(count = counter + 1L), Some(counter + 1L))
        case None => (tweet, Some(1L))
      }
  }
```

What If a Failure Occurs?



Loss of data and incorrect count!

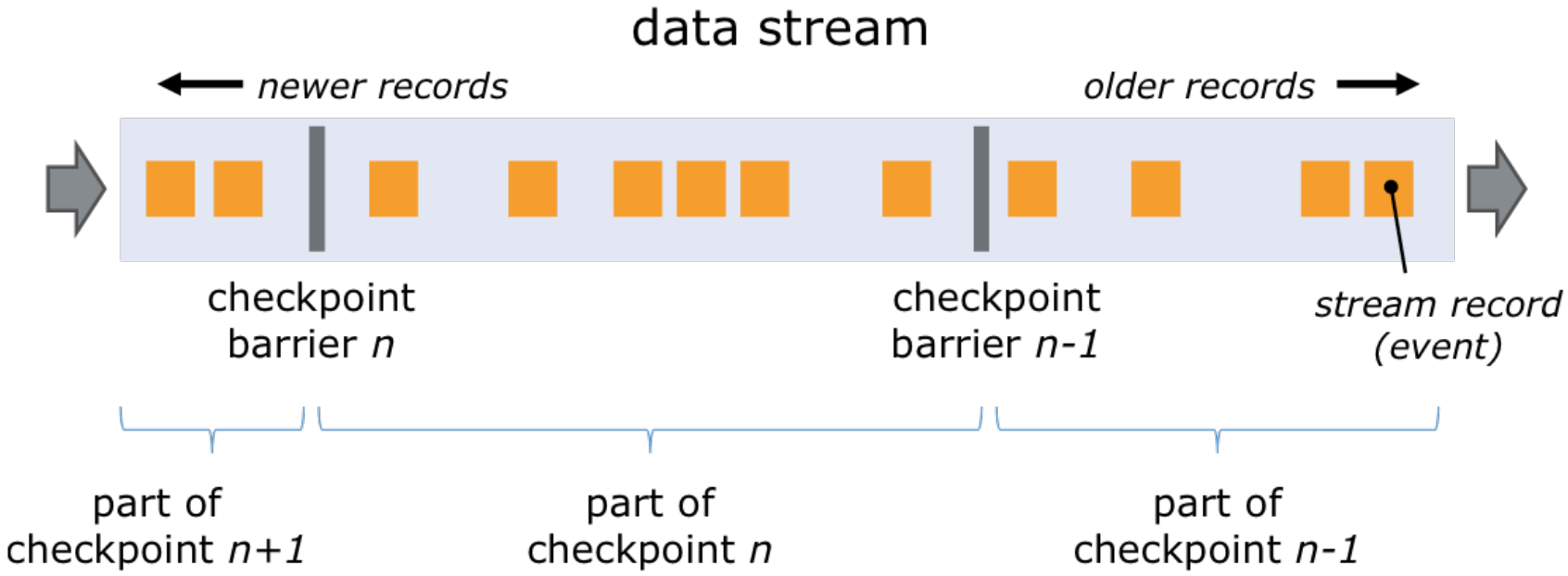
We Need to Save Our State!



- Consistent snapshots of distributed data stream and operator state

How Does It Work?

- Markers for checkpoints
- Injected in the data flow



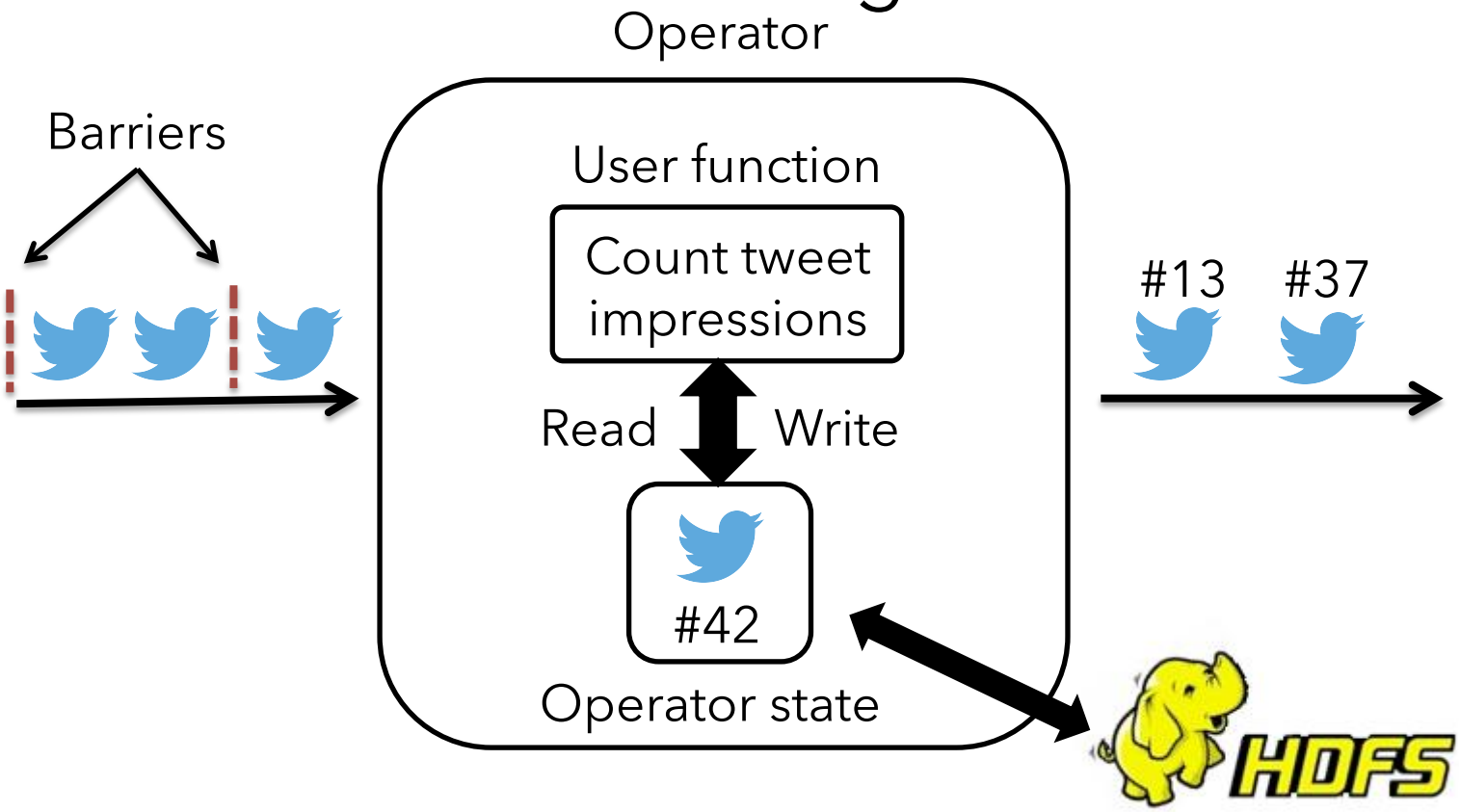
Processing Guarantees

- **At most once**
  - No guarantees at all
- **At least once**
  - Ensure that all operators see all events
- **Exactly once**

Flink gives you **all** guarantees

Checkpointed Operators

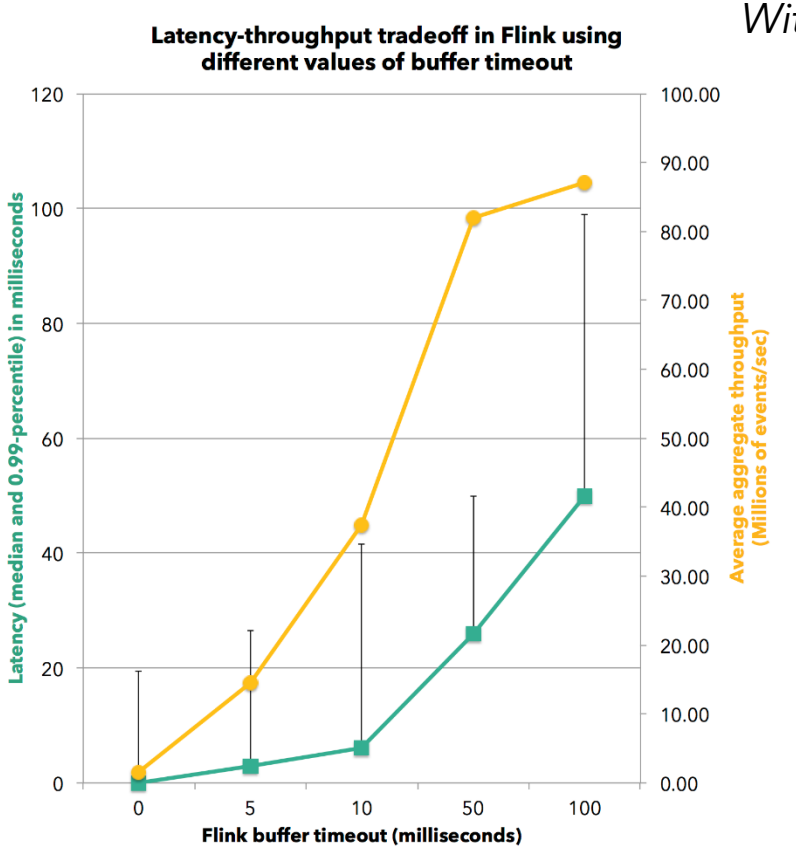
- State is automatically checkpointed
- State backend is configurable

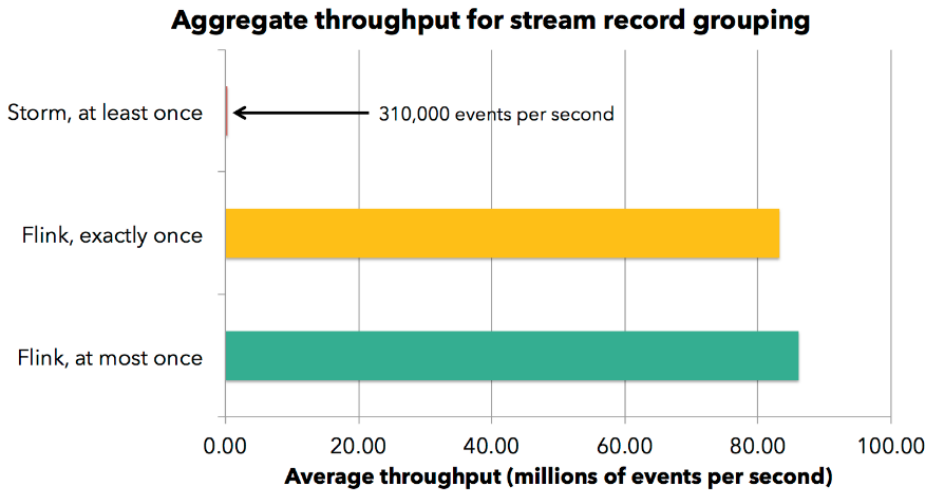


What About Performance?

Continuous streaming + Latency-bound buffering + Distributed Snapshots = High Throughput & Low Latency

With configurable throughput/latency tradeoff

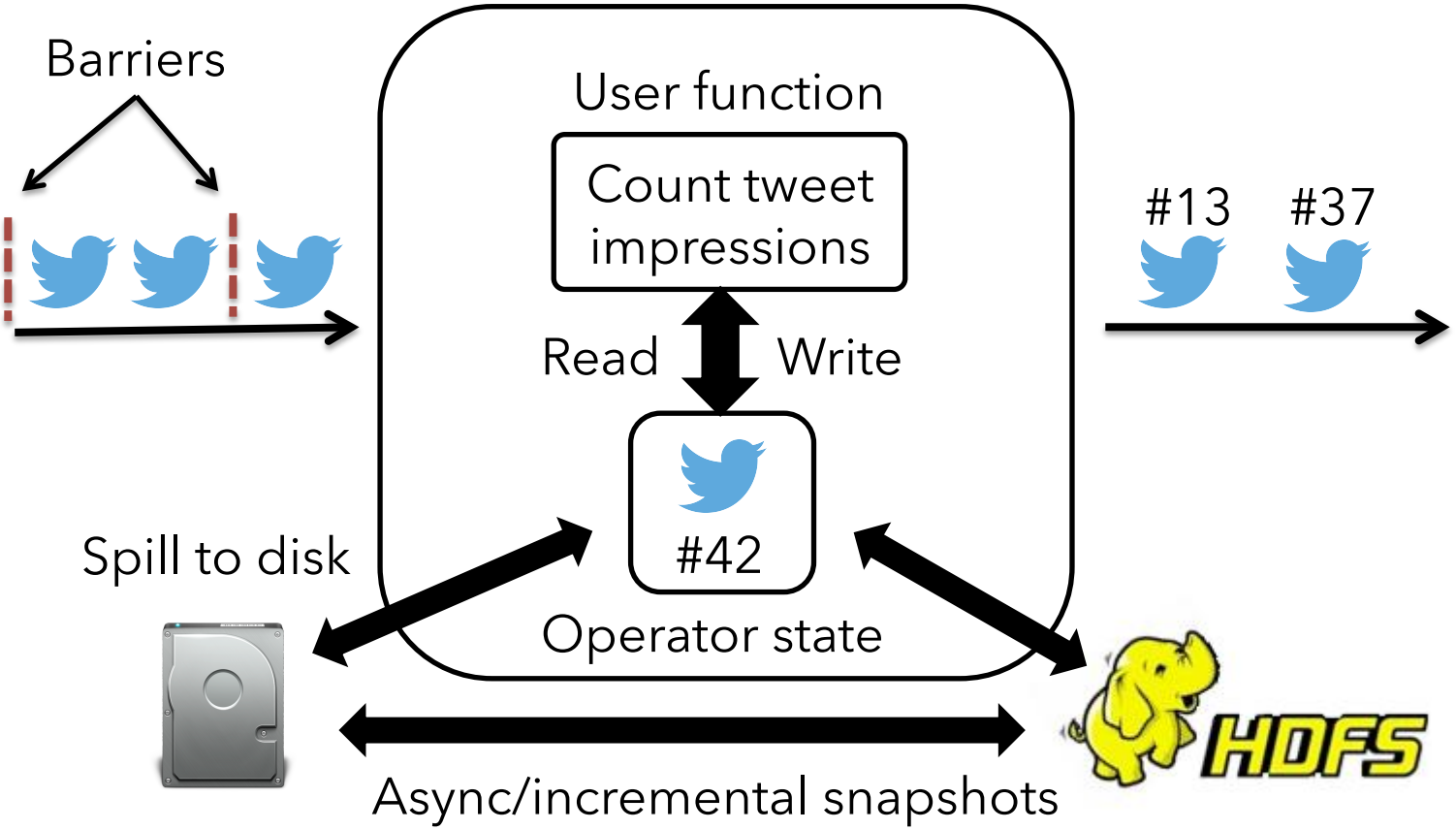




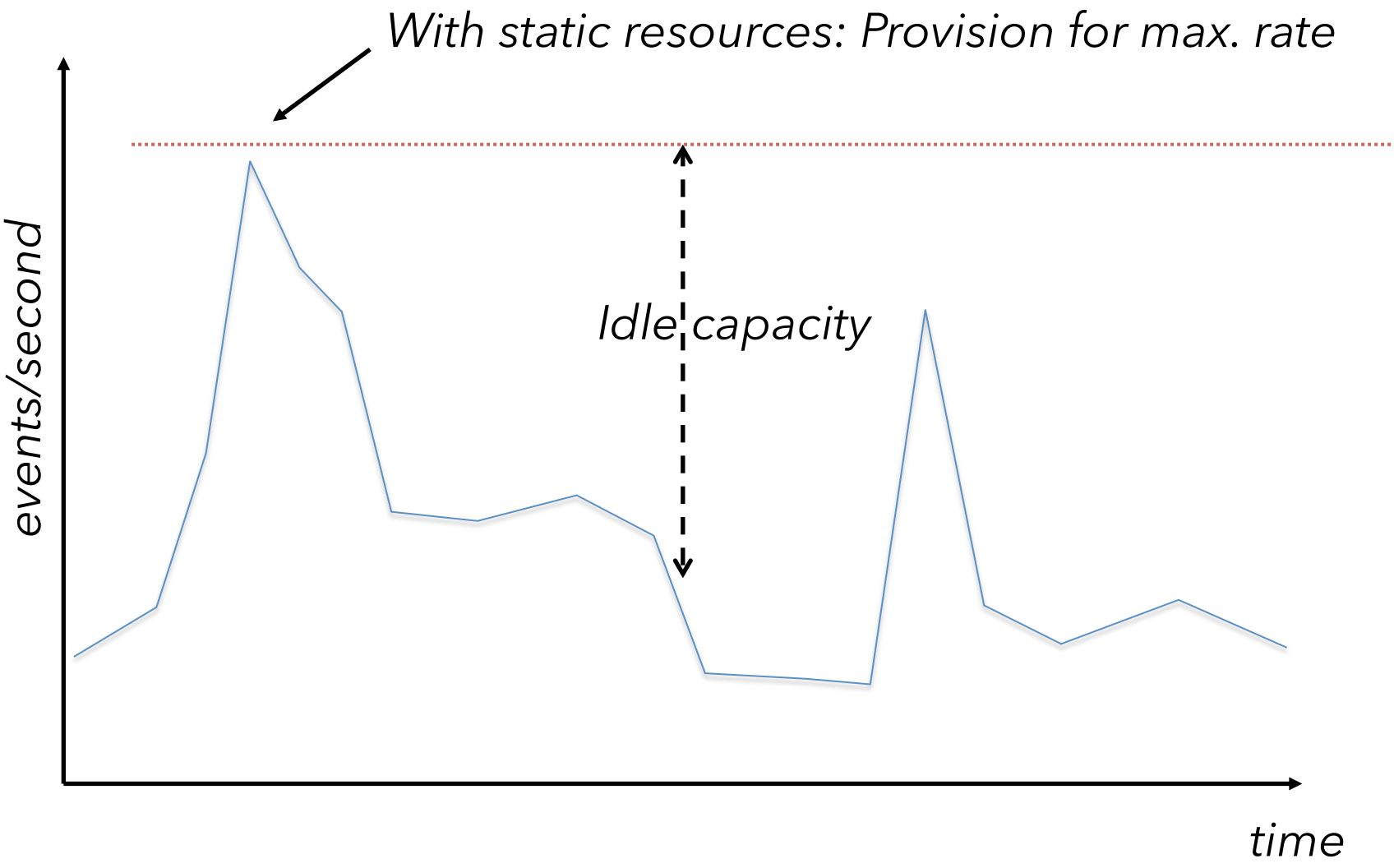
What’s coming next?

Asynchronous Snapshots

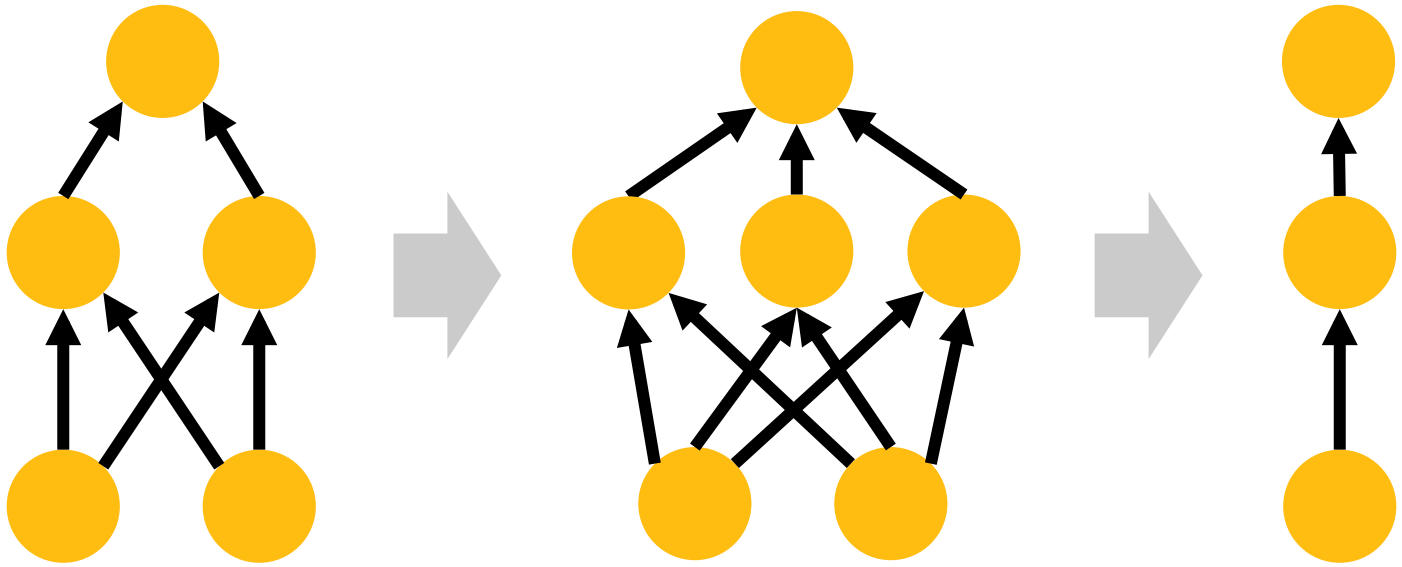
- Taking snapshots stalls the operator
- Solution: Out of core & asynchronous snapshots



Streams with Varying Data Rate



(1) Adjust Parallelism

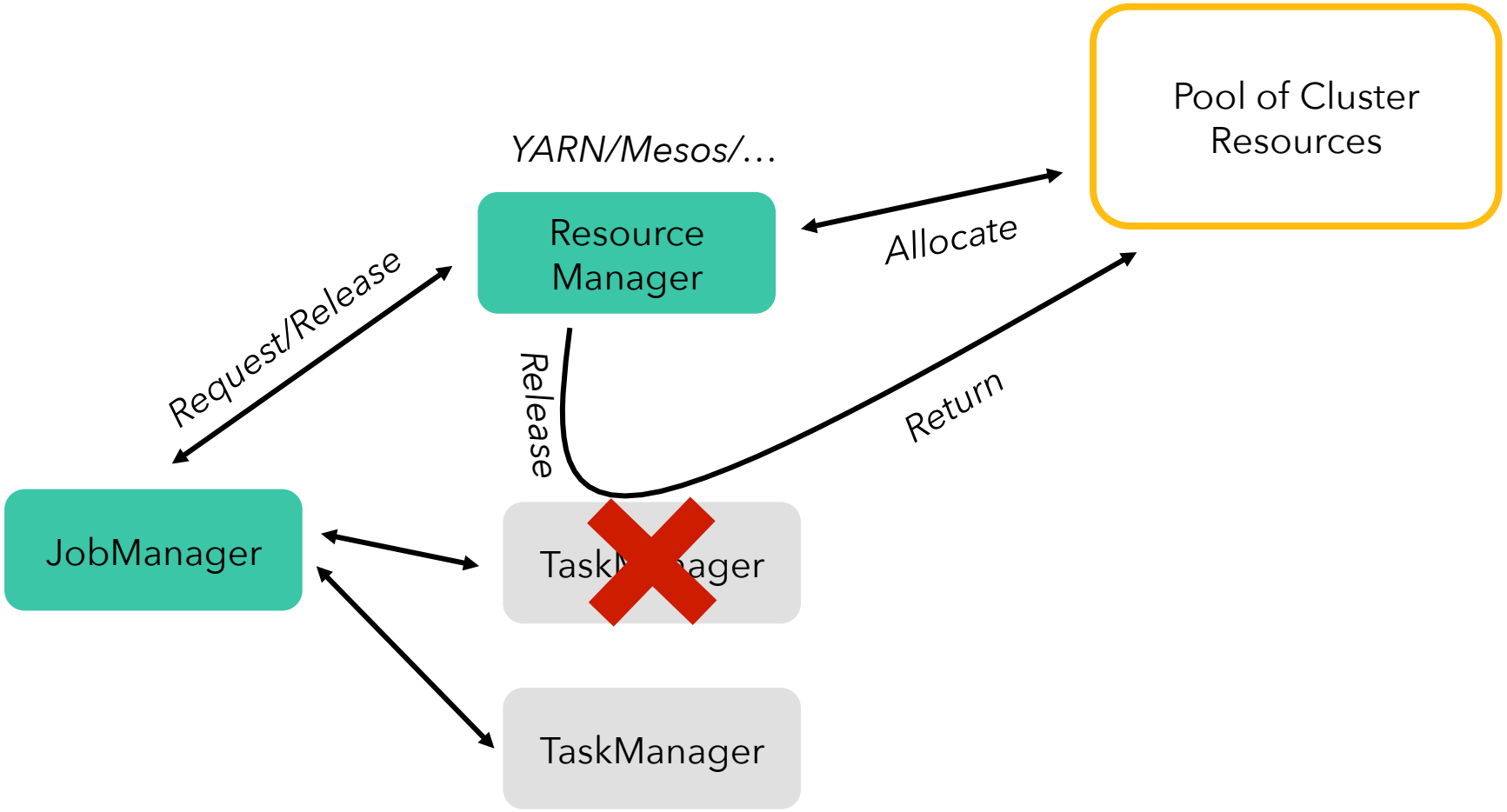


Initial configuration

Scale Out (for load)

Scale In (save resources)

(2) Dynamic Worker Pool

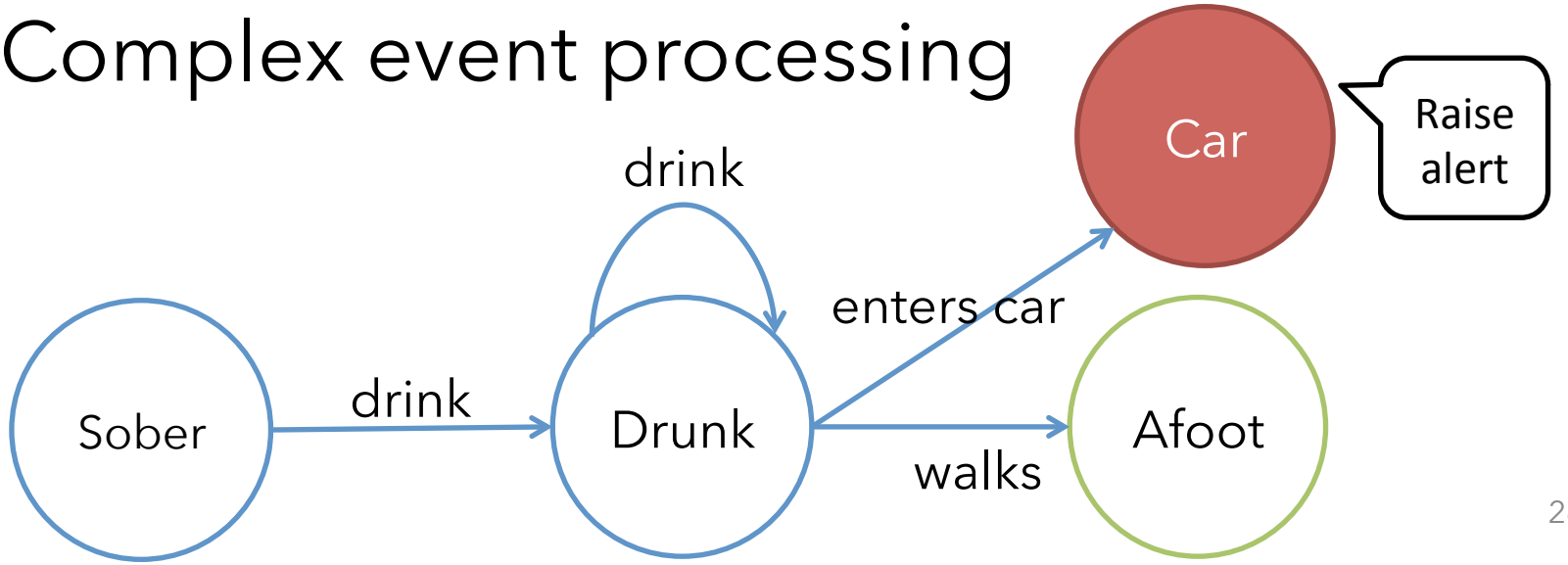


Declarative Queries

- StreamSQL

```
val tabEnv = new TableEnvironment(env)
tabEnv.registerStream(stream, “myStream”,
  (“ID”, “MEASURE”, “COUNT”))

val sqlQuery = tabEnv.sql(
    “SELECT ID, MEASURE FROM myStream WHERE COUNT > 17”)
```

- Complex event processing



Where to Find Us?

www.flink.apache.org

https://github.com/apache/flink

@ApacheFlink

Architecture Overview