Unified Stream & Batch Processing with Apache Flink

Ufuk Celebi
dataArtisans

Hadoop Summit Dublin
April 13, 2016

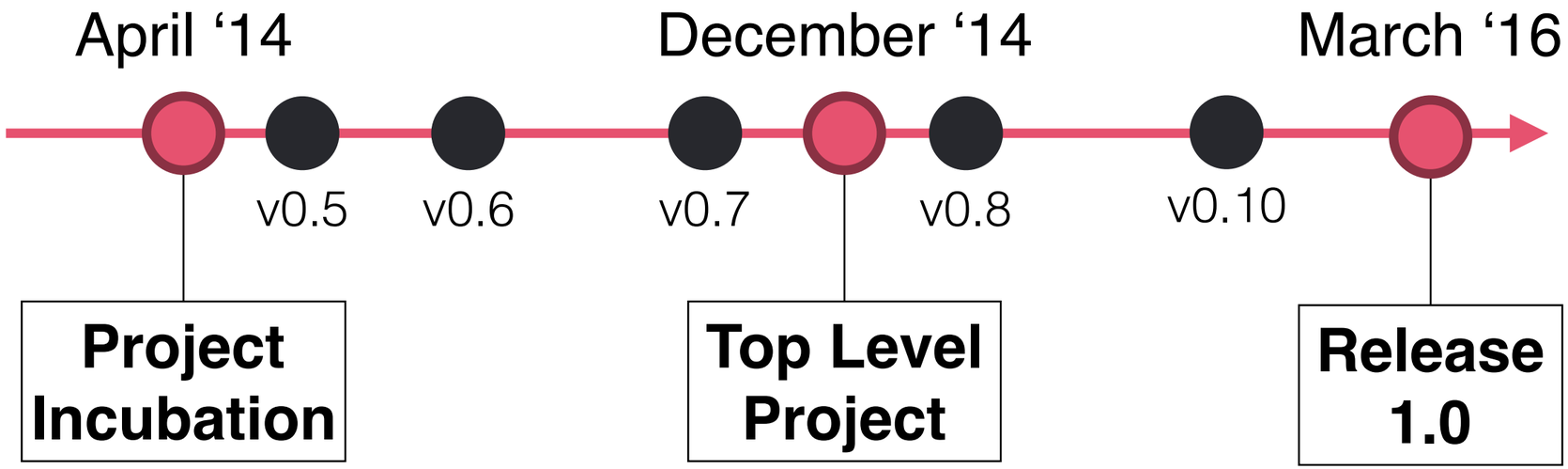
What is Apache Flink?

Apache Flink is an open source *stream processing* framework.

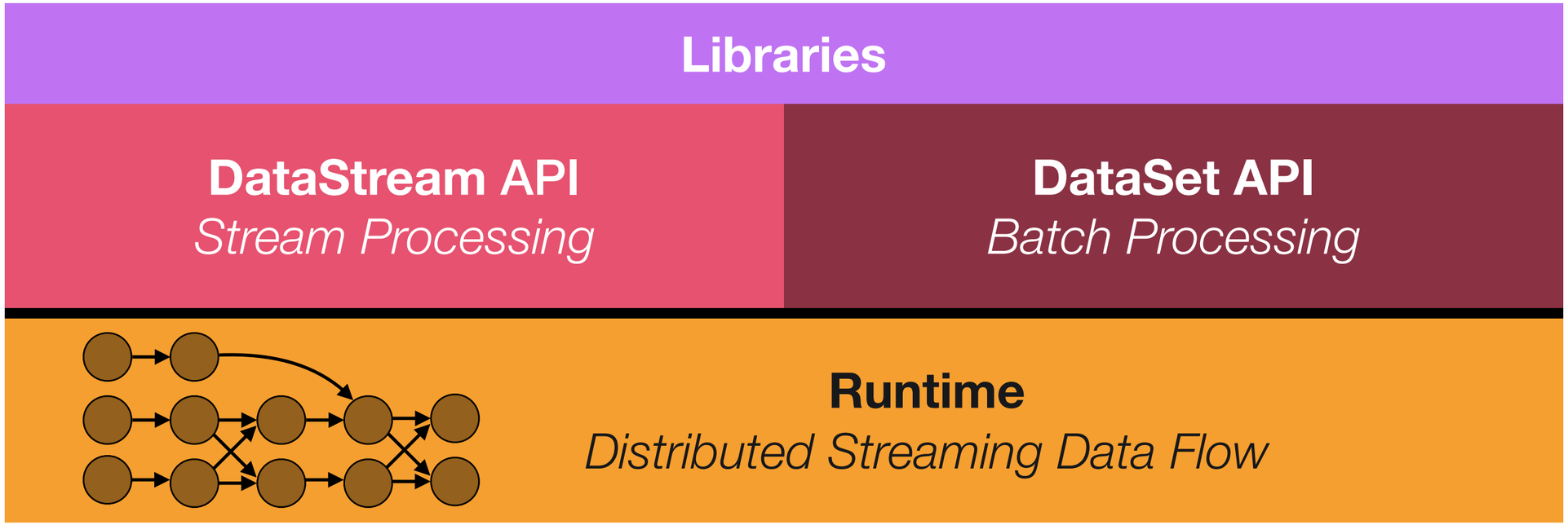
- Event Time Handling
- State & Fault Tolerance
- Low Latency
- High Throughput

Developed at the *Apache Software Foundation*.

Recent History

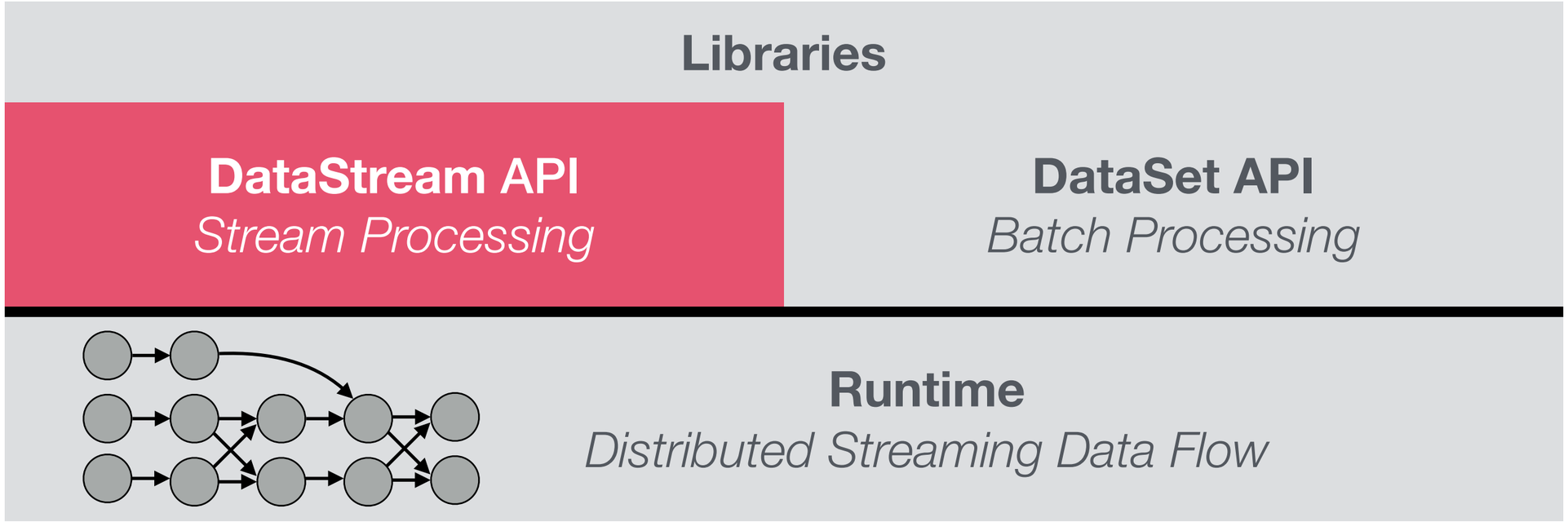


Flink Stack



Streaming and **batch** as **first class citizens**.

Today



Streaming and **batch** as **first class citizens**.

Counting

Seemingly **simple application**:
Count visitors, ad impressions, etc.

But **generalizes** well to other problems.

Batch Processing

All Input

Hadoop,
Spark,
Flink

All Output

Batch Processing

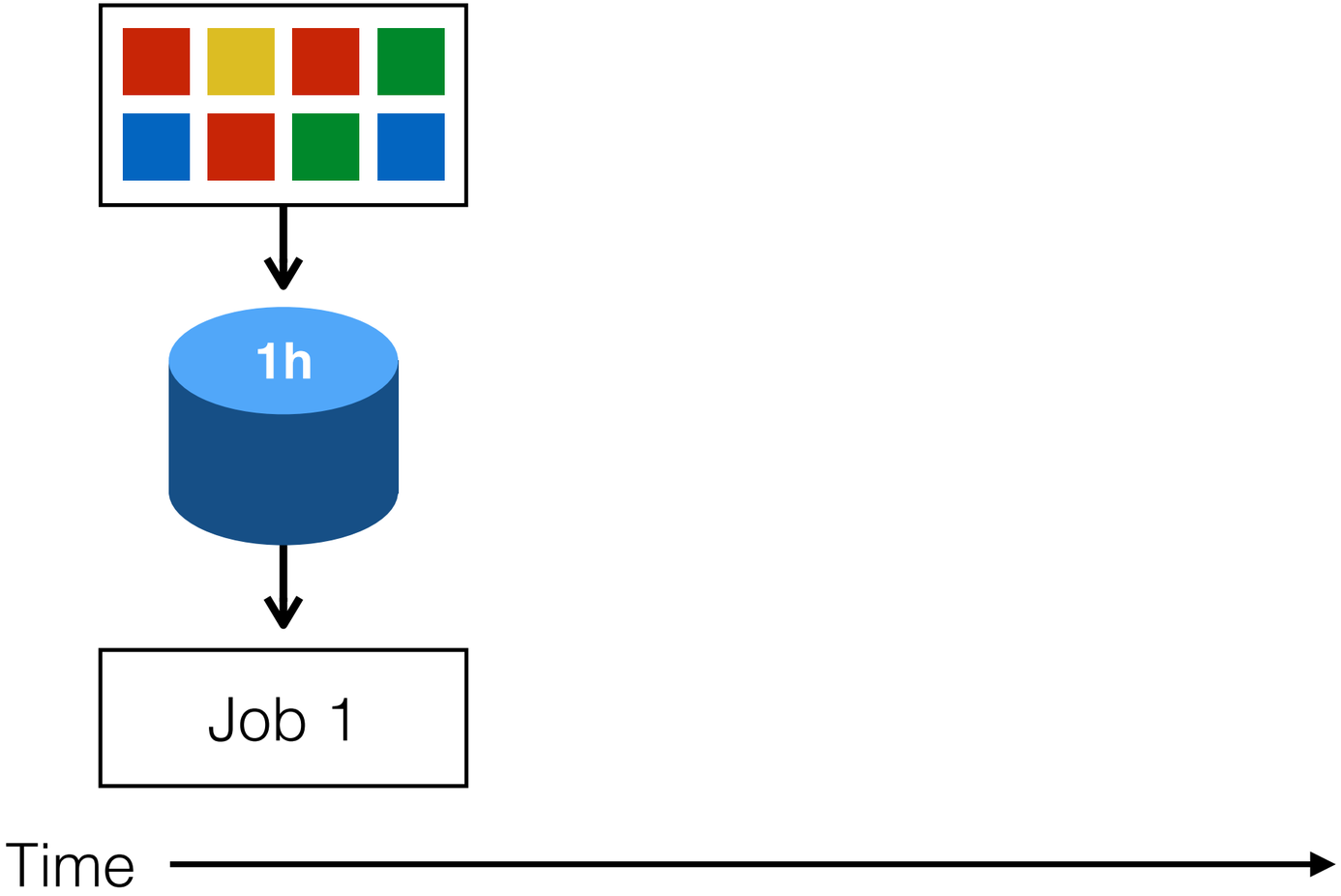
```
DataSet<ColorEvent> counts = env
  .readFile("MM-dd.csv")
  .groupBy("color")
  .count();
```

Continuous Counting

Continuous ingestion

Periodic files

Periodic batch jobs

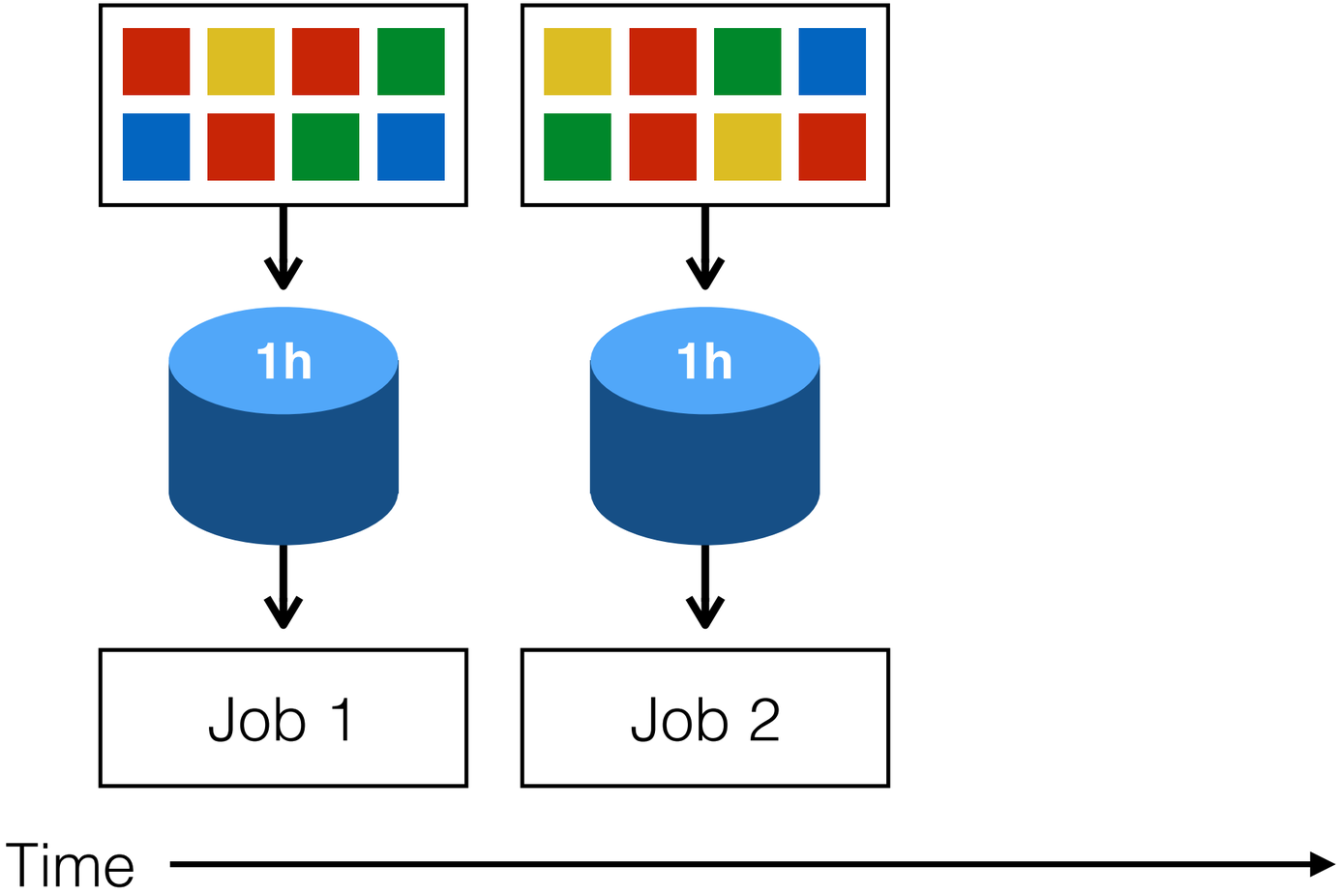


Continuous Counting

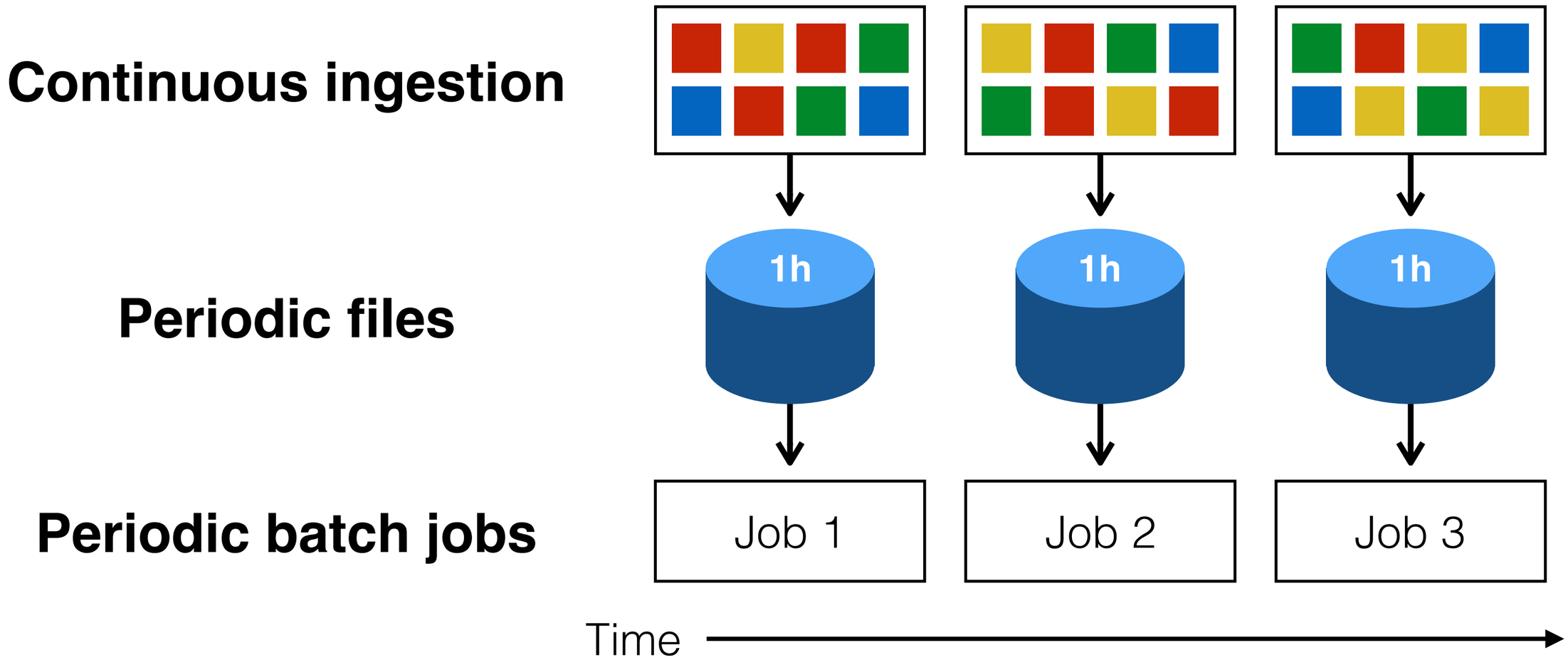
Continuous ingestion

Periodic files

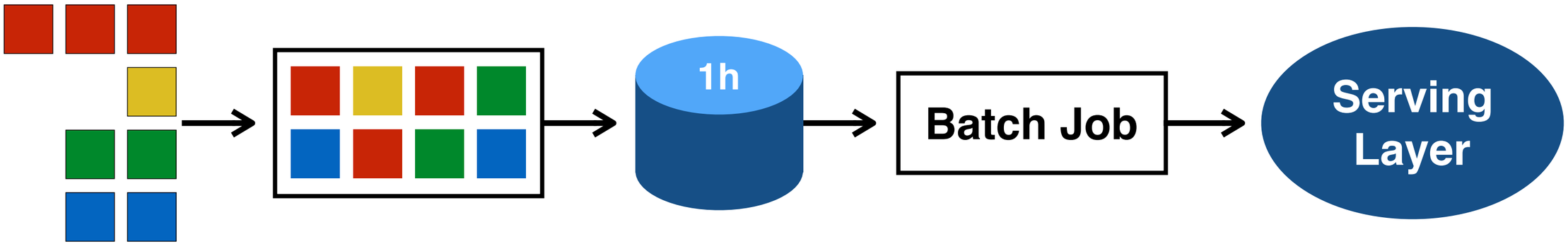
Periodic batch jobs



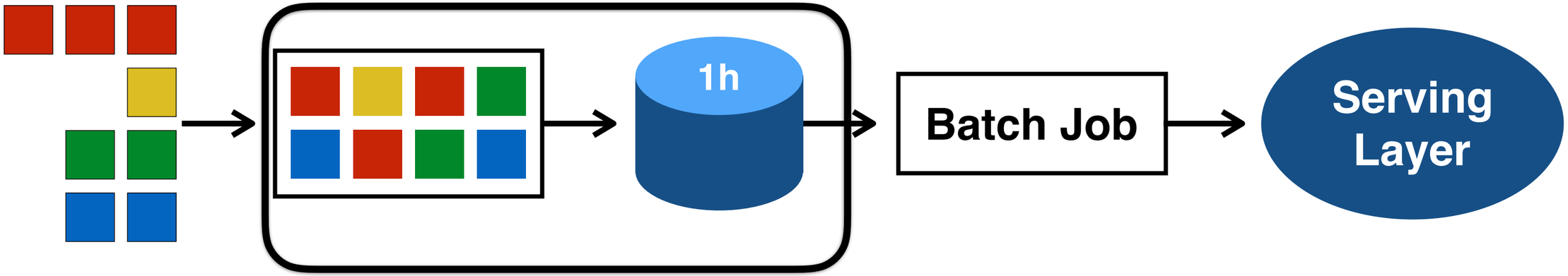
Continuous Counting



Many Moving Parts

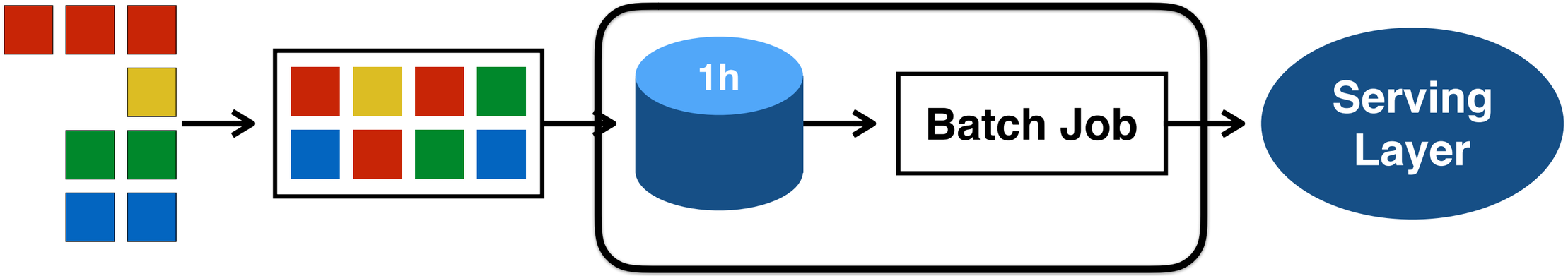


Many Moving Parts



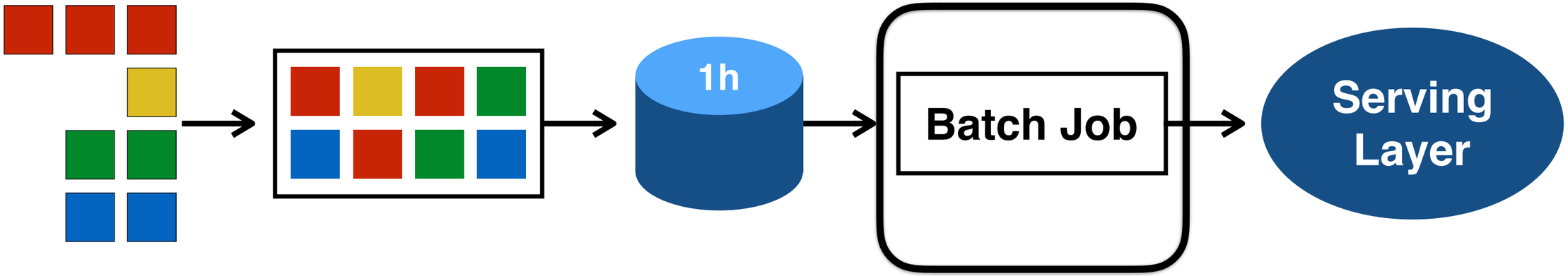
Data loading into HDFS
(e.g. Flume)

Many Moving Parts



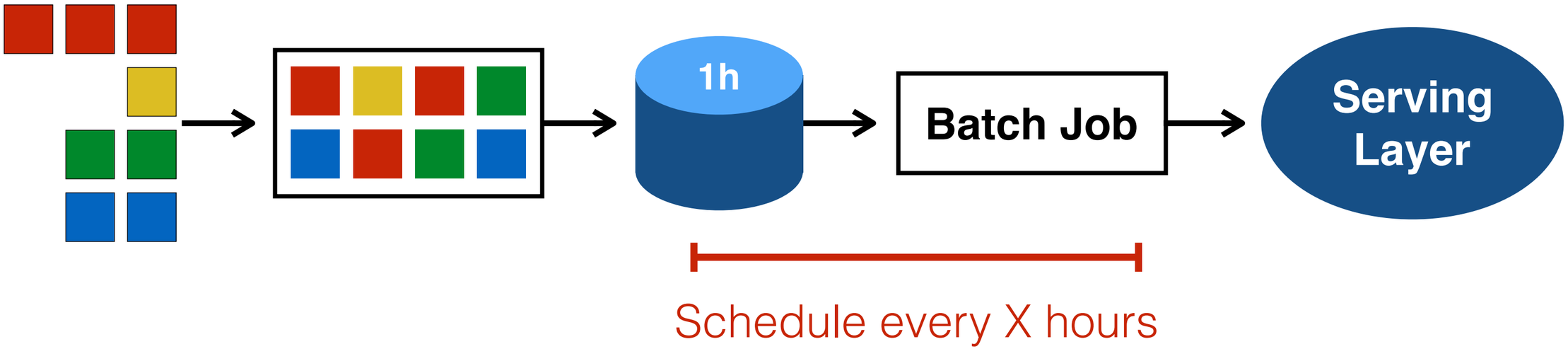
Periodic job scheduler
(e.g. Oozie)

Many Moving Parts



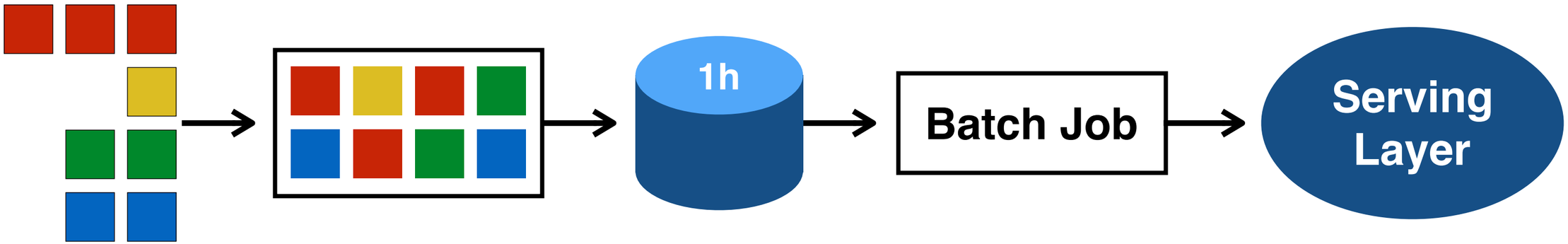
Batch processor
(e.g. Hadoop, Spark, Flink)

High Latency



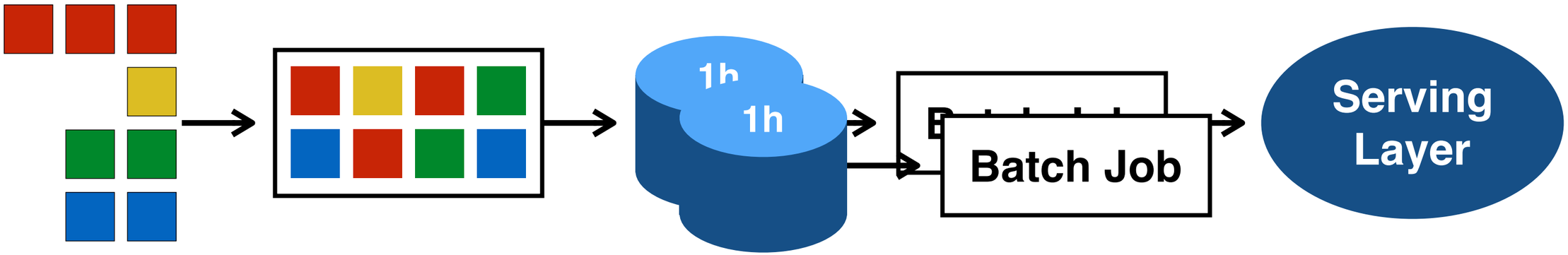
Latency from *event to serving layer* usually in the **range of hours.**

Implicit Treatment of Time



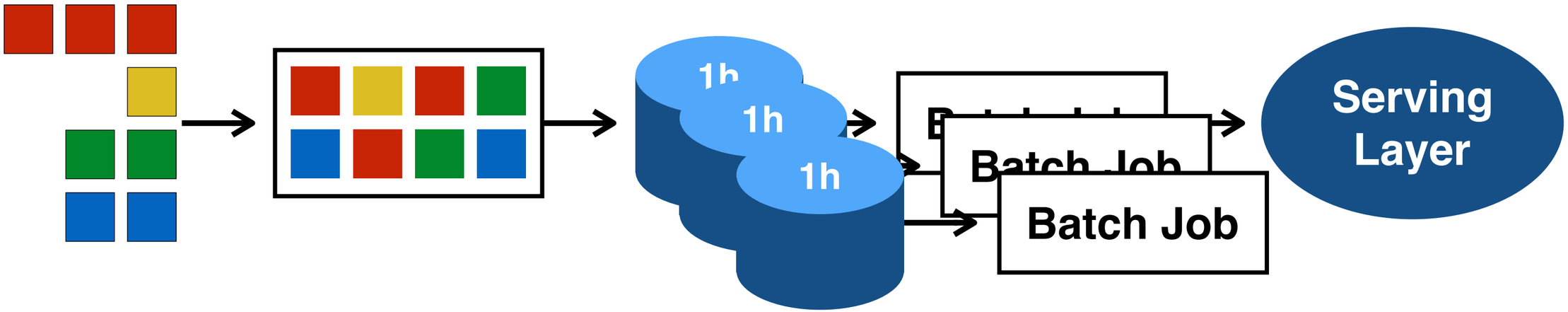
Time is treated **outside** of your application.

Implicit Treatment of Time



Time is treated **outside** of your application.

Implicit Treatment of Time



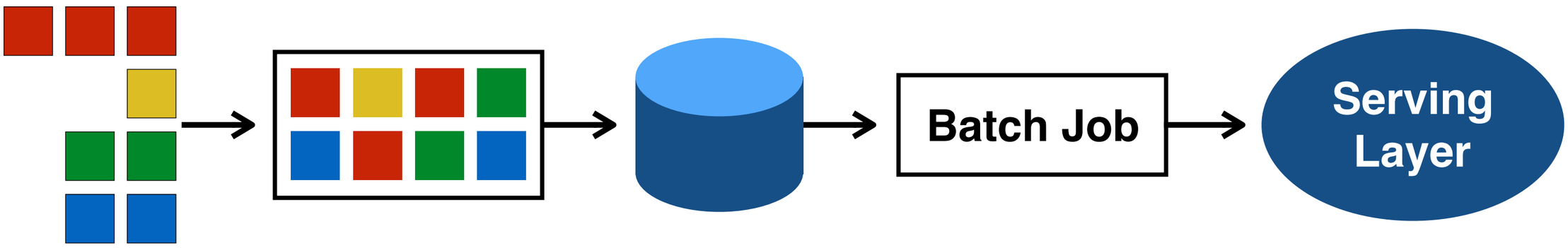
Time is treated **outside** of your application.

Implicit Treatment of Time

```
DataSet<ColorEvent> counts = env
  .readFile("MM-dd.csv")
  .groupBy("color")
  .count();
```

Time is **implicit**
in input file

Streaming over Batch



Continuously produced

Files are **finite streams**

Periodically executed

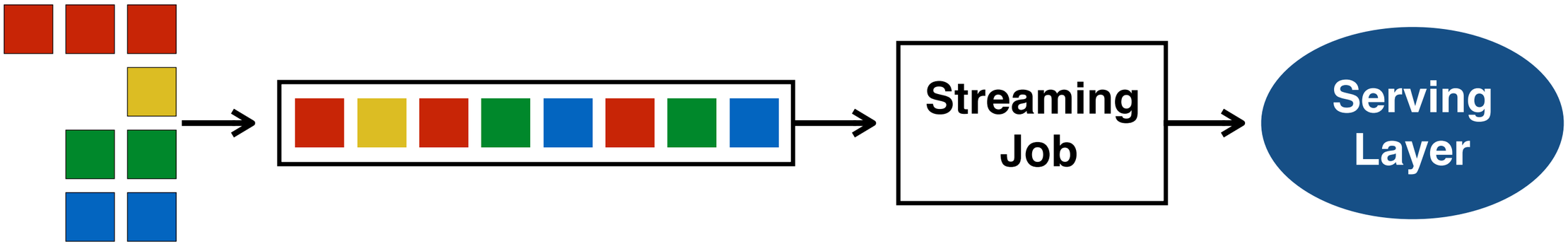
Streaming

Until now, **stream processors** were **less mature** than their batch counterparts. This led to:

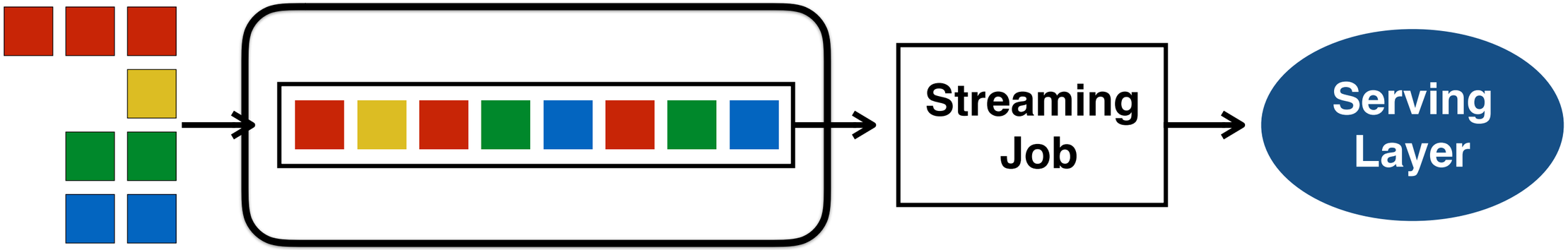
- **in-house solutions**,
- **abuse of batch processors**,
- **Lambda architectures**

This is **no longer** needed with new generation stream processors like Flink.

Streaming All the Way



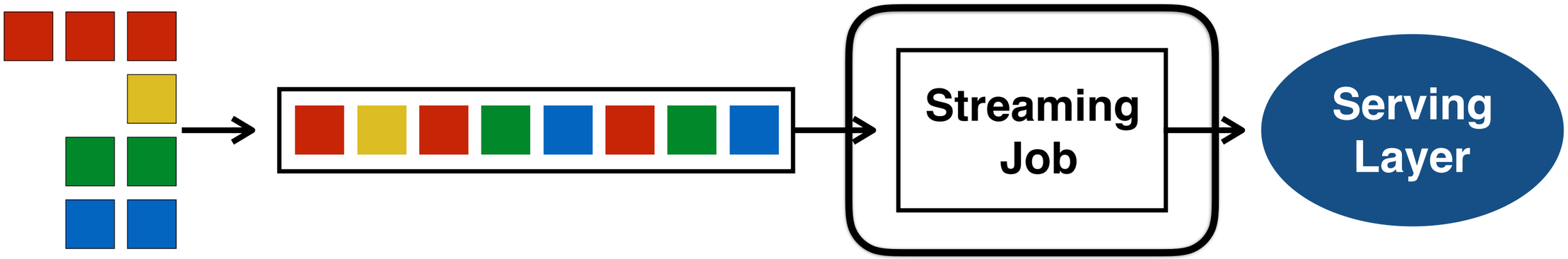
Streaming All the Way



Message Queue
(e.g. Apache Kafka)

Durability and Replay

Streaming All the Way



Stream Processor
(e.g. Apache Flink)

Consistent Processing

Building Blocks of Flink

Explicit Handling of Time

State & Fault Tolerance

Performance

Building Blocks of Flink

Explicit Handling of Time

State & Fault Tolerance

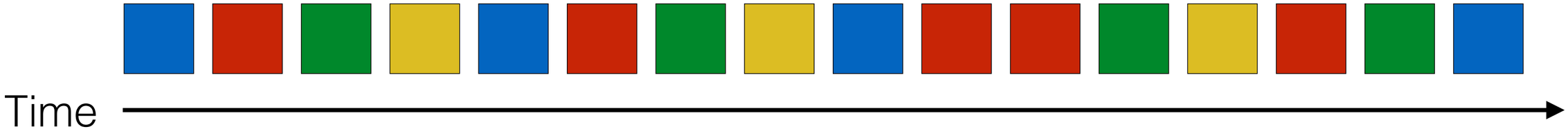
Performance

Windowing

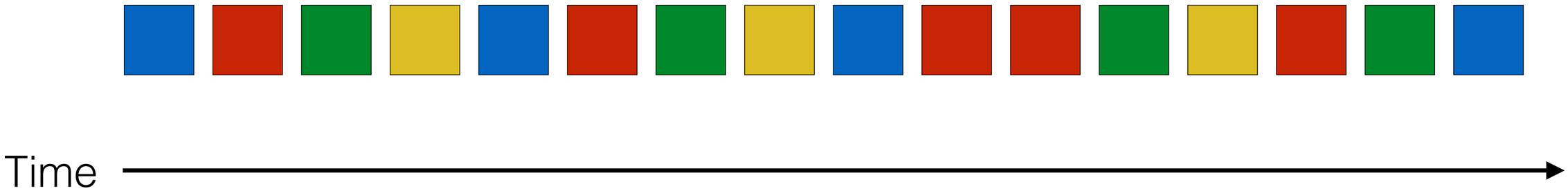
Aggregates on **streams**
are scoped by **windows**

Time-driven
e.g. last X minutes

Data-driven
e.g. last X records



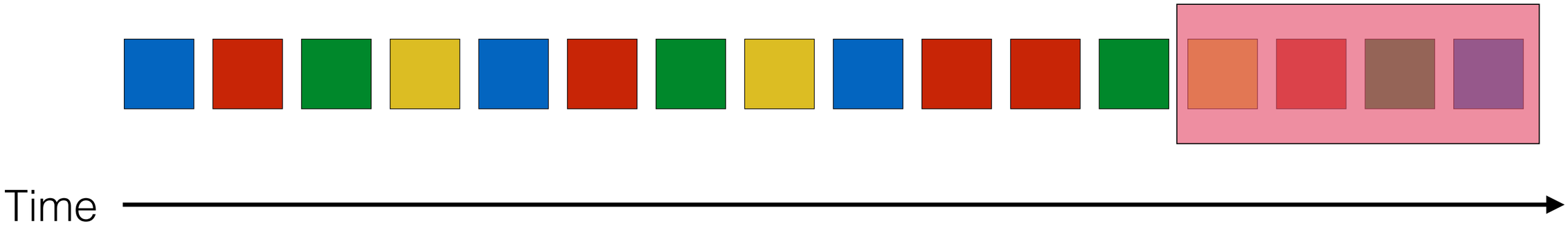
Tumbling Windows (No Overlap)



e.g. *"Count over the **last 5 minutes**",*

*"Average over the **last 100 records**"*

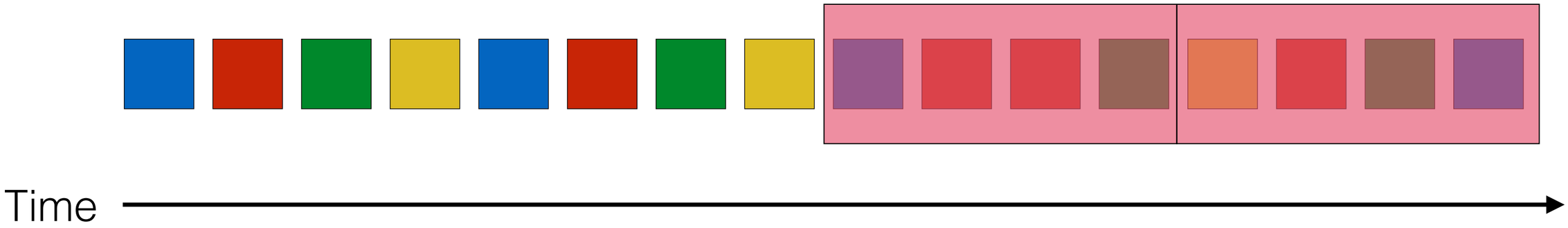
Tumbling Windows (No Overlap)



e.g. *"Count over the **last 5 minutes**",*

*"Average over the **last 100 records**"*

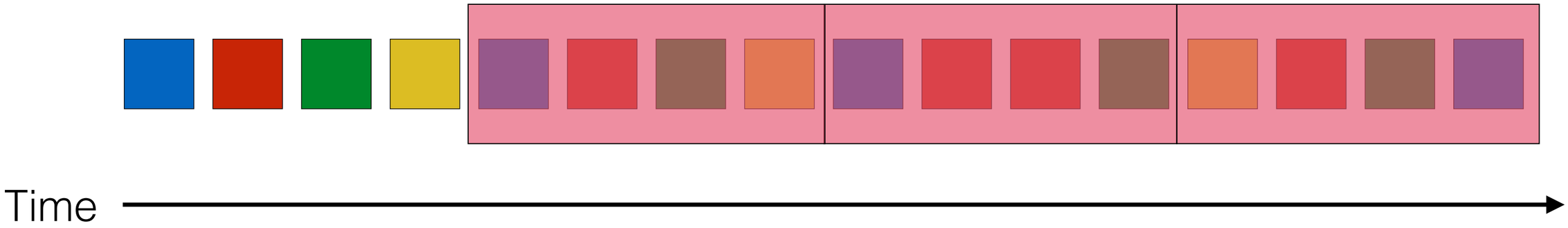
Tumbling Windows (No Overlap)



e.g. *"Count over the **last 5 minutes**",*

*"Average over the **last 100 records**"*

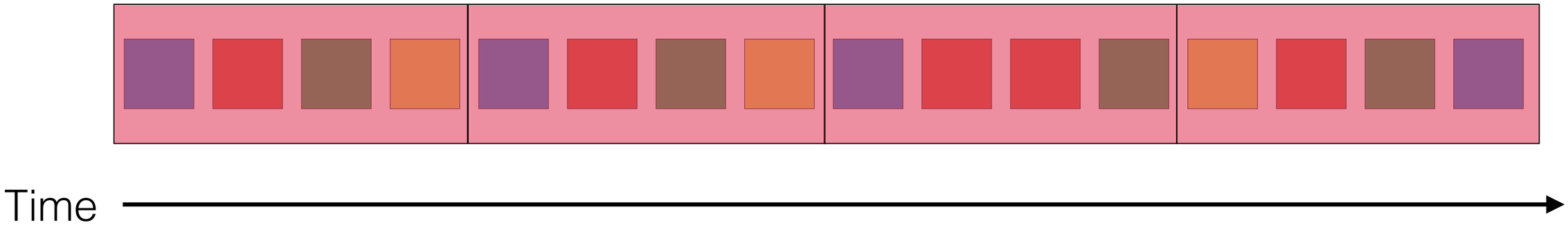
Tumbling Windows (No Overlap)



e.g. *"Count over the **last 5 minutes**",*

*"Average over the **last 100 records**"*

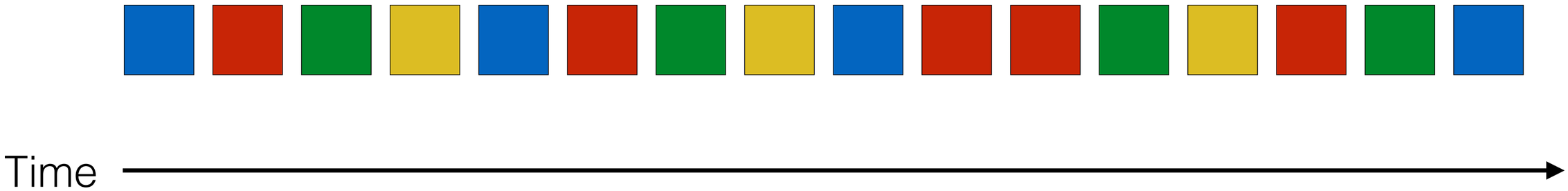
Tumbling Windows (No Overlap)



e.g. *"Count over the* ***last 5 minutes****"*,

"Average over the ***last 100 records****"*

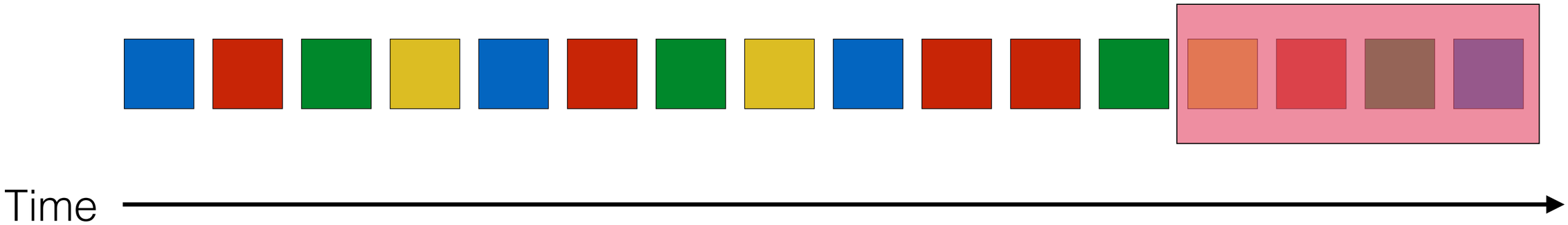
Sliding Windows (with Overlap)



e.g. *"Count over the **last 5 minutes**, updated **each minute**."*,

*"Average over the **last 100 elements**, updated every **10 elements**"*

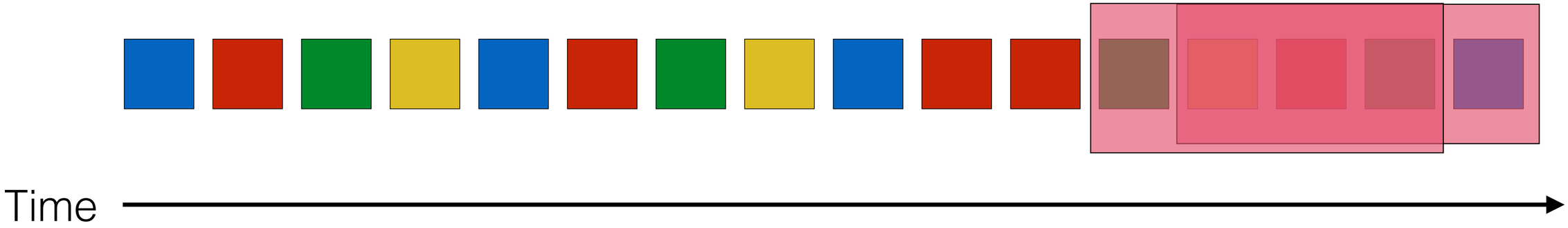
Sliding Windows (with Overlap)



e.g. *"Count over the **last 5 minutes**, updated **each minute**."*,

*"Average over the **last 100 elements**, updated every **10 elements**"*

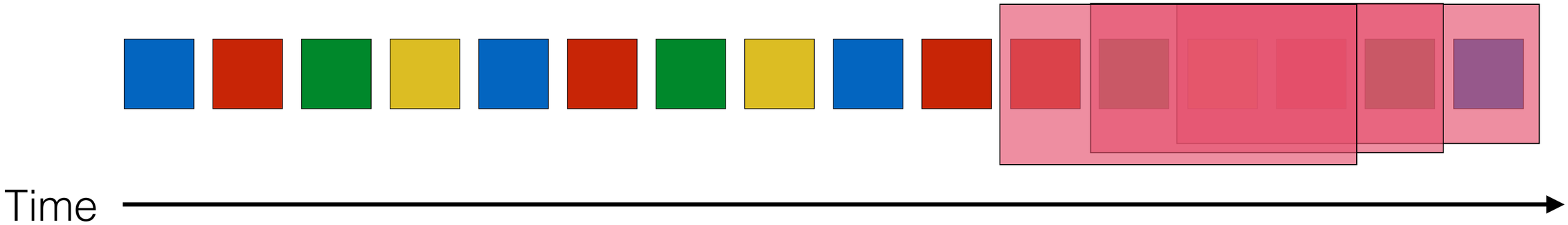
Sliding Windows (with Overlap)



e.g. *"Count over the **last 5 minutes**, updated **each minute**."*,

*"Average over the **last 100 elements**, updated every **10 elements**"*

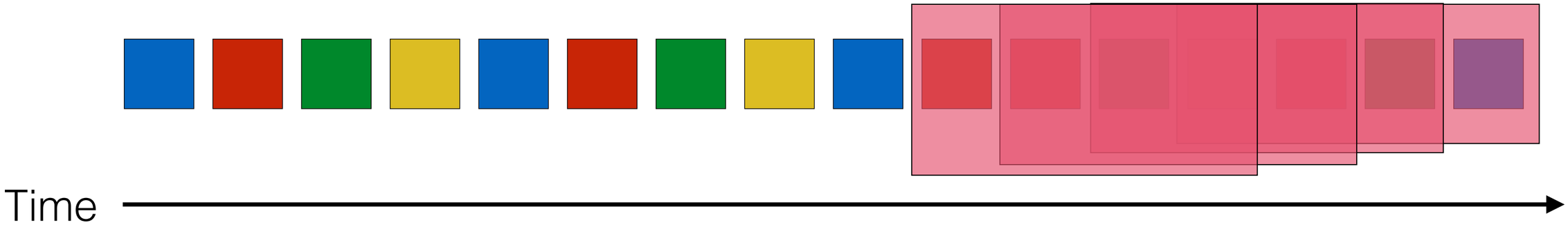
Sliding Windows (with Overlap)



e.g. *"Count over the **last 5 minutes**, updated **each minute**."*,

*"Average over the **last 100 elements**, updated every **10 elements**"*

Sliding Windows (with Overlap)



e.g. *"Count over the **last 5 minutes**, updated **each minute**."*,

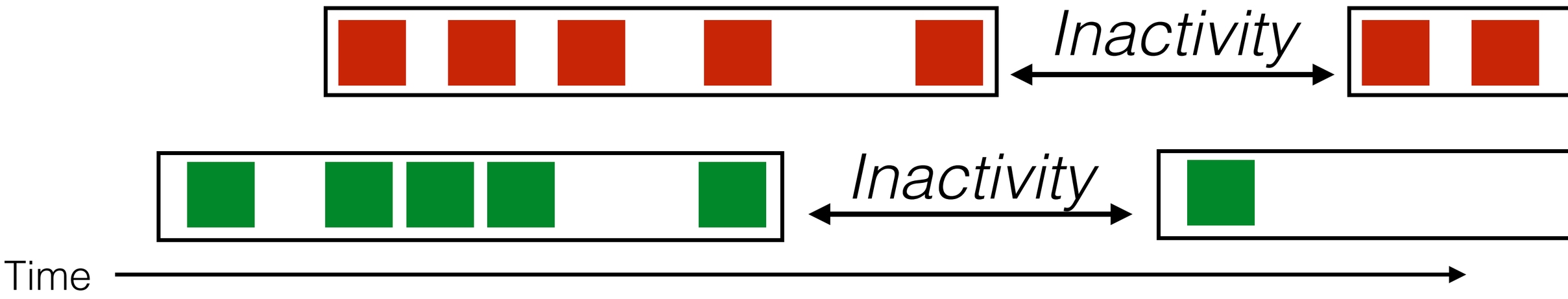
*"Average over the **last 100 elements**, updated every **10 elements**"*

Explicit Handling of Time

```
DataStream<ColorEvent> counts = env
  .addSource(new KafkaConsumer(…))
  .keyBy("color")
  .timeWindow(Time.minutes(60))
  .apply(new CountPerWindow());
```

Time is **explicit**
in **your program**

Session Windows



Sessions **close** after *period of inactivity*.

e.g. "Count activity from login until time-out or logout."

Session Windows

```
DataStream<ColorEvent> counts = env
  .addSource(new KafkaConsumer(…))
  .keyBy("color")
  .window(EventTimeSessionWindows
    .withGap(Time.minutes(10))
  .apply(new CountPerWindow());
```

Notions of Time

Event Time

Time when event happened.

12:23 am

Notions of Time

Event Time

Time when event happened.

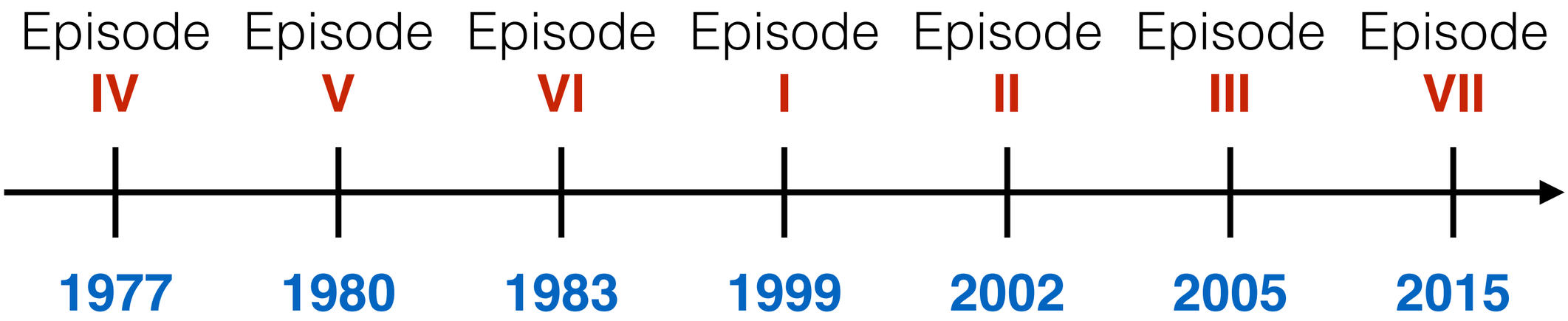
12:23 am

1:37 pm

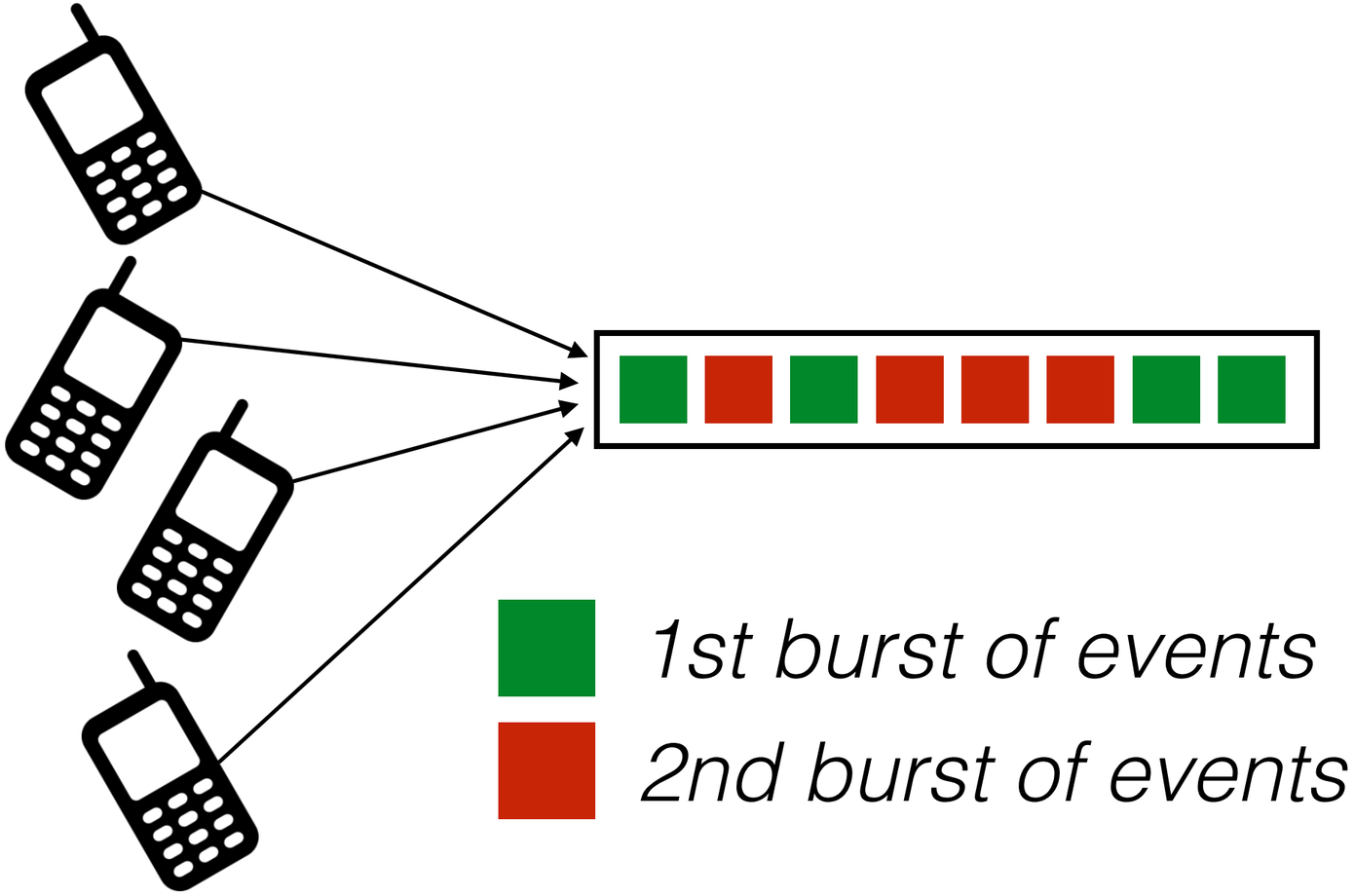
Processing Time

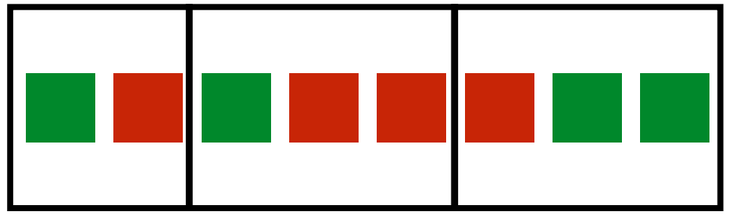
Time measured by system clock

Out of Order Events

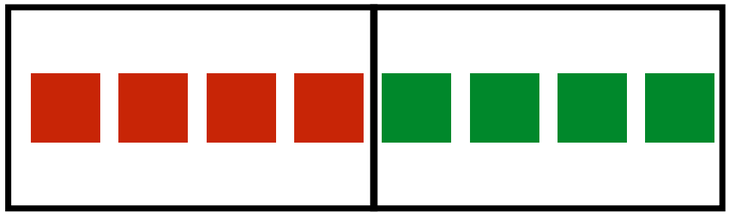


Out of Order Events





Processing Time Windows



Event Time Windows

Notions of Time

```
env.setStreamTimeCharacteristic(
  TimeCharacteristic.EventTime);

DataStream<ColorEvent> counts = env
  ...
  .timeWindow(Time.minutes(60))
  .apply(new CountPerWindow());
```

Explicit Handling of Time

1. **Expressive windowing**
2. **Accurate results** for out of order data
3. **Deterministic** results

Building Blocks of Flink

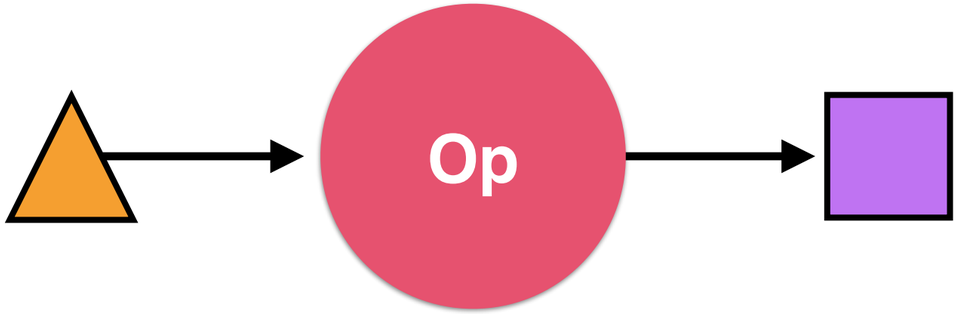
Explicit Handling of Time

State & Fault Tolerance

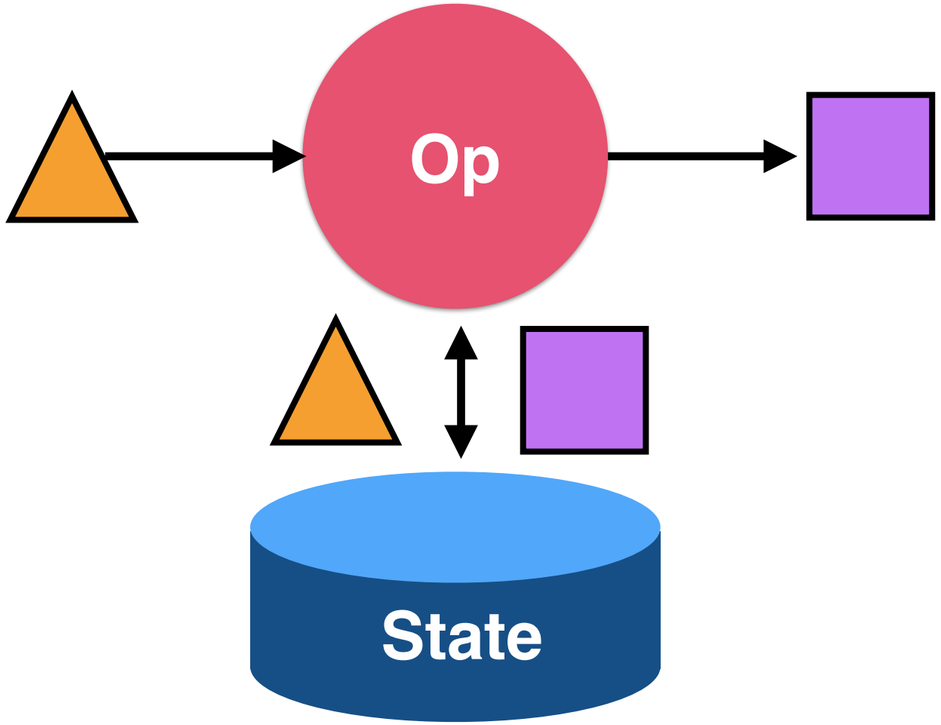
Performance

Stateful Streaming

Stateless Stream Processing



Stateful Stream Processing



Processing Semantics

At-least once
May over-count
after failure

Exactly Once
Correct counts
after failures

End-to-end exactly once
Correct counts in external system
(e.g. DB, file system) after failure

Processing Semantics

- Flink guarantees **exactly once** (can be configured for *at-least once* if desired)
- **End-to-end exactly** once with specific sources and sinks (e.g. Kafka -> Flink -> HDFS)
- Internally, Flink periodically takes **consistent snapshots** of the state without ever stopping computation

Building Blocks of Flink

Explicit Handling of Time

State & Fault Tolerance

Performance

Yahoo! Benchmark

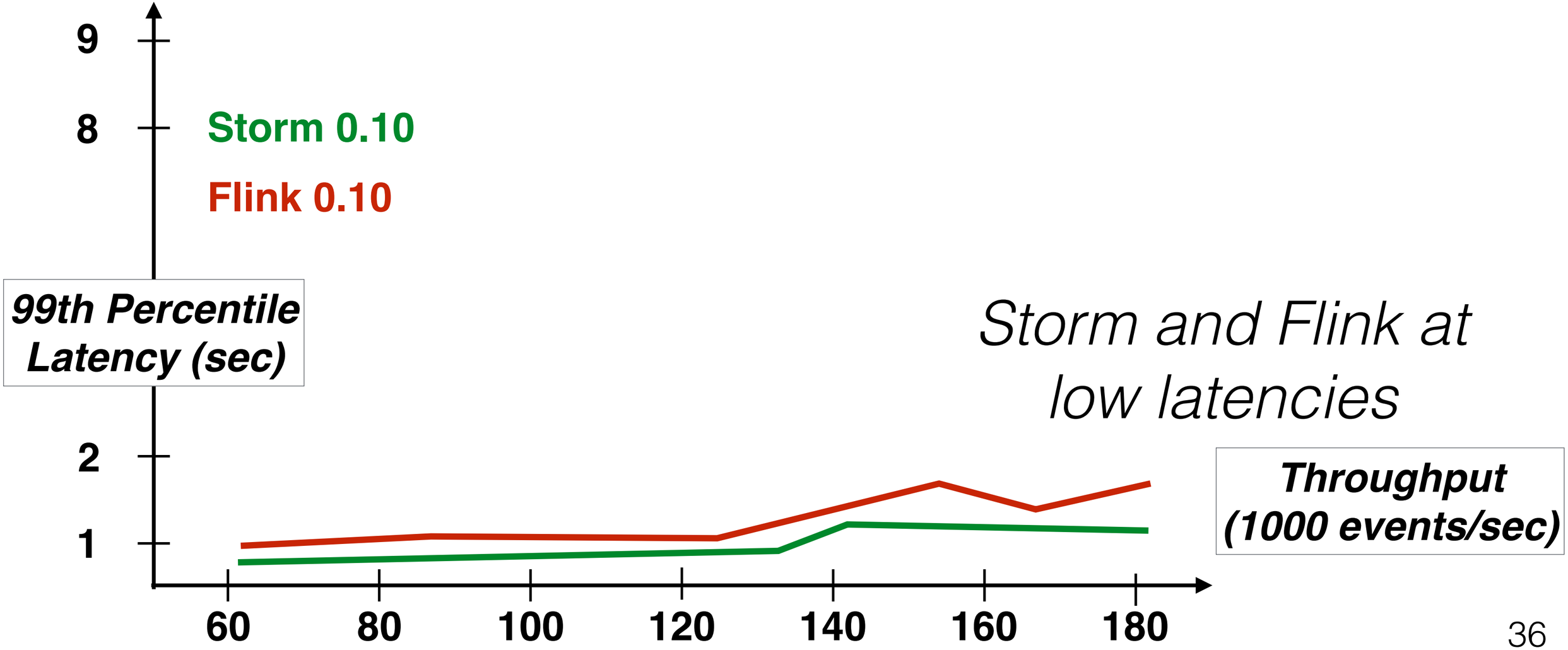
- **Storm 0.10**, **Spark Streaming 1.5**, and **Flink 0.10** benchmark by Storm team at Yahoo!
- Focus on measuring **end-to-end latency** at **low throughputs** (~ 200k events/sec)
- First benchmark modelled after a **real application**

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

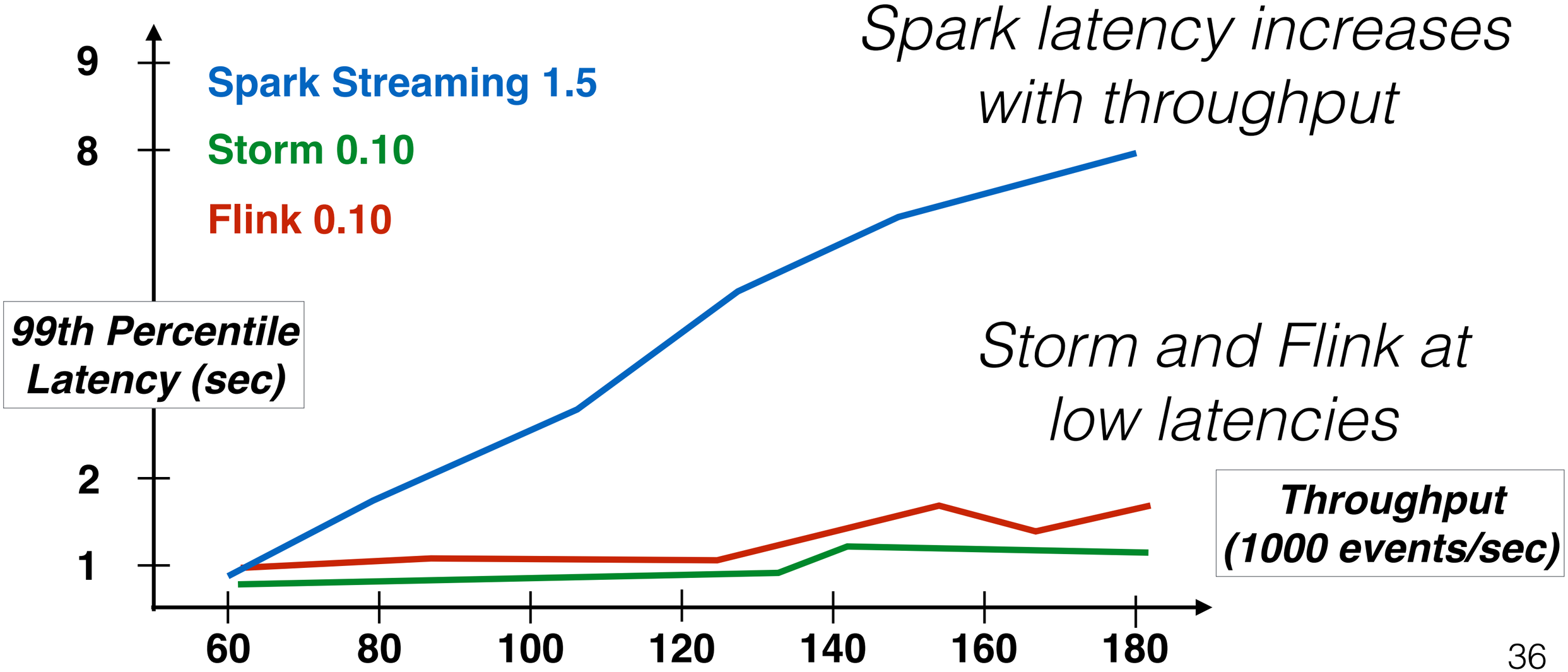
Yahoo! Benchmark

- **Count ad impressions** grouped by campaign
- Compute **aggregates** over last 10 seconds
- Make aggregates available for **queries** in Redis

Latency (Lower is Better)



Latency (Lower is Better)

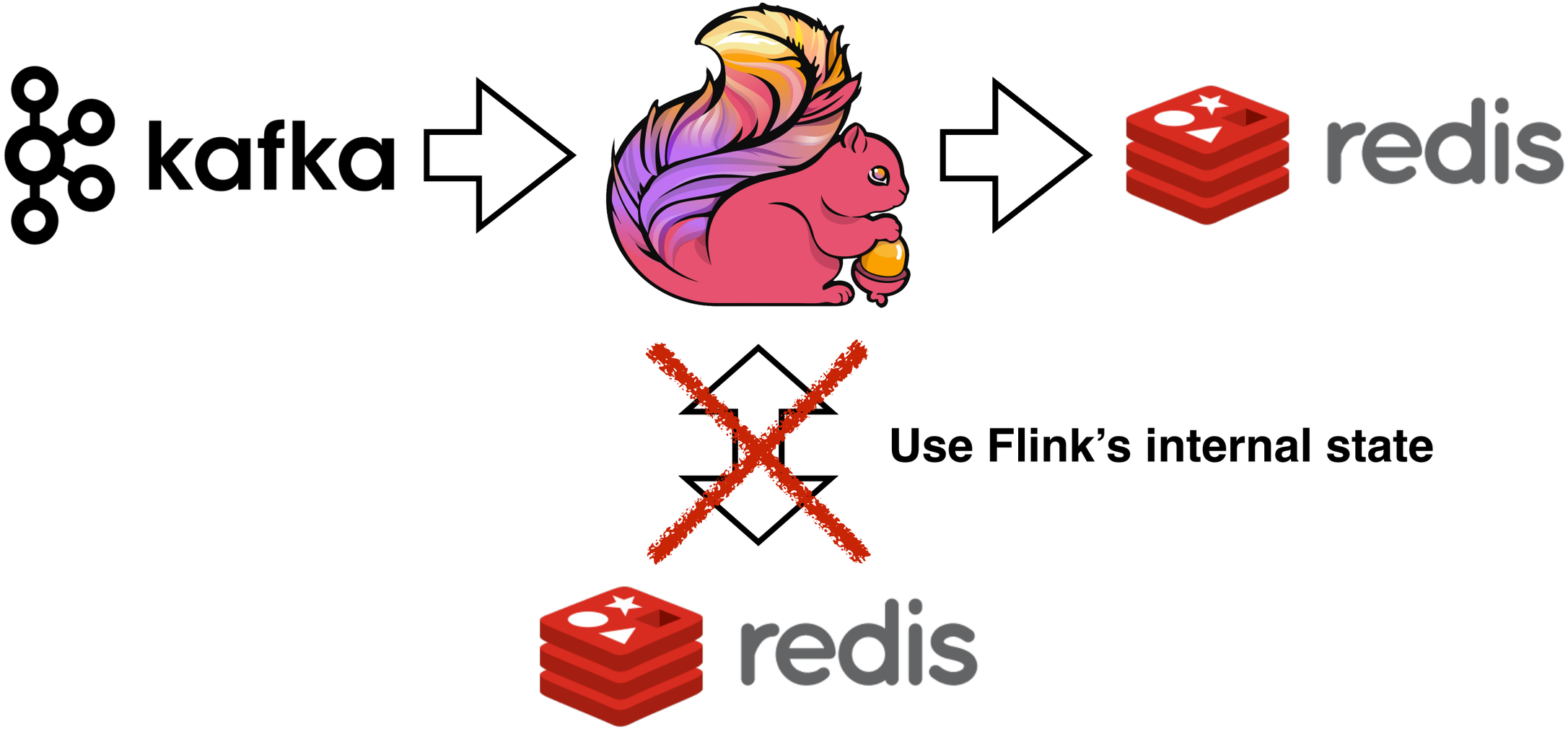


Extending the Benchmark

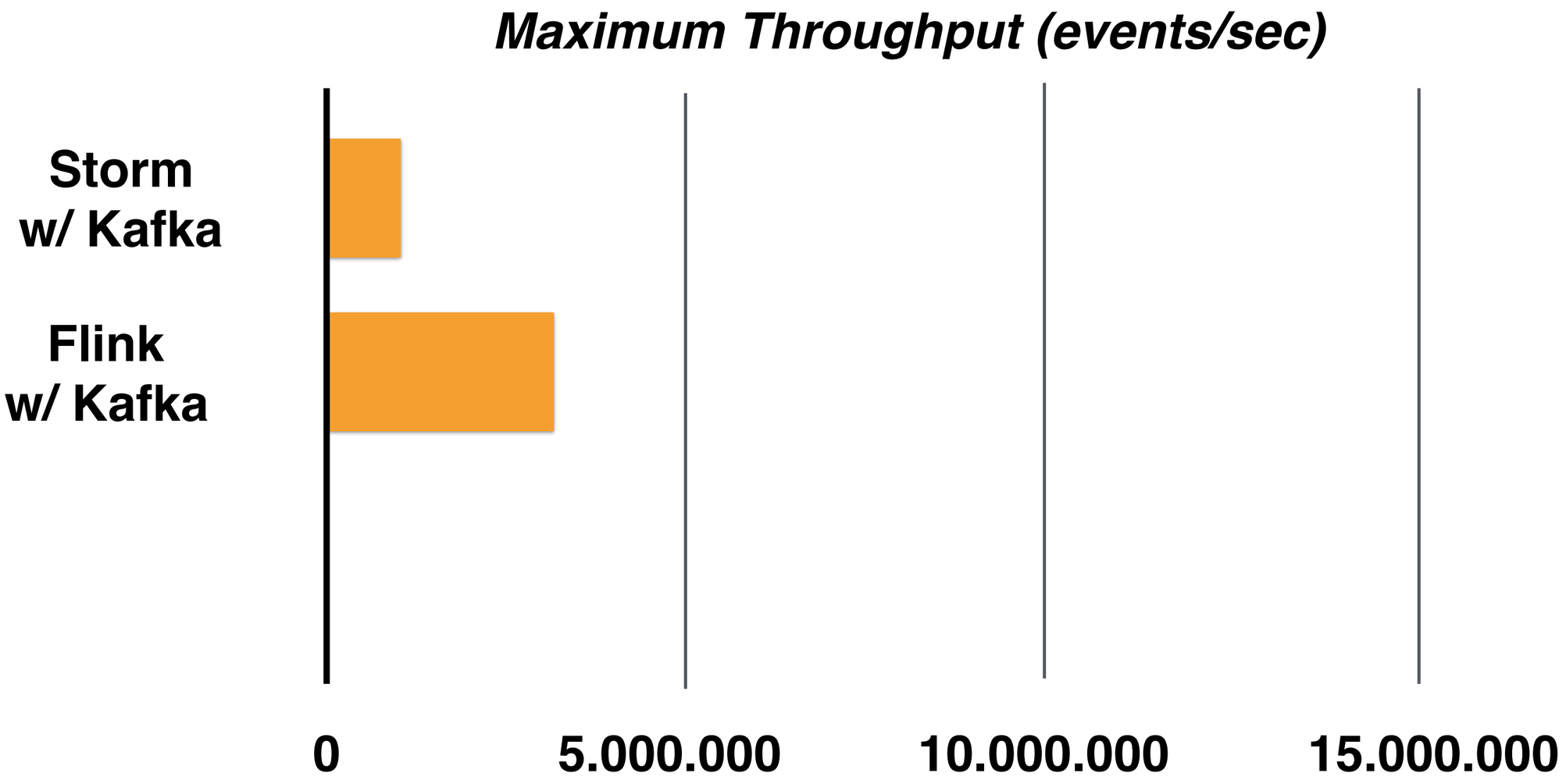
- Great starting point, but benchmark stops at **low write throughput** and programs are **not fault-tolerant**
- Extend benchmark to **high volumes** and Flink's built-in **fault-tolerant state**

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

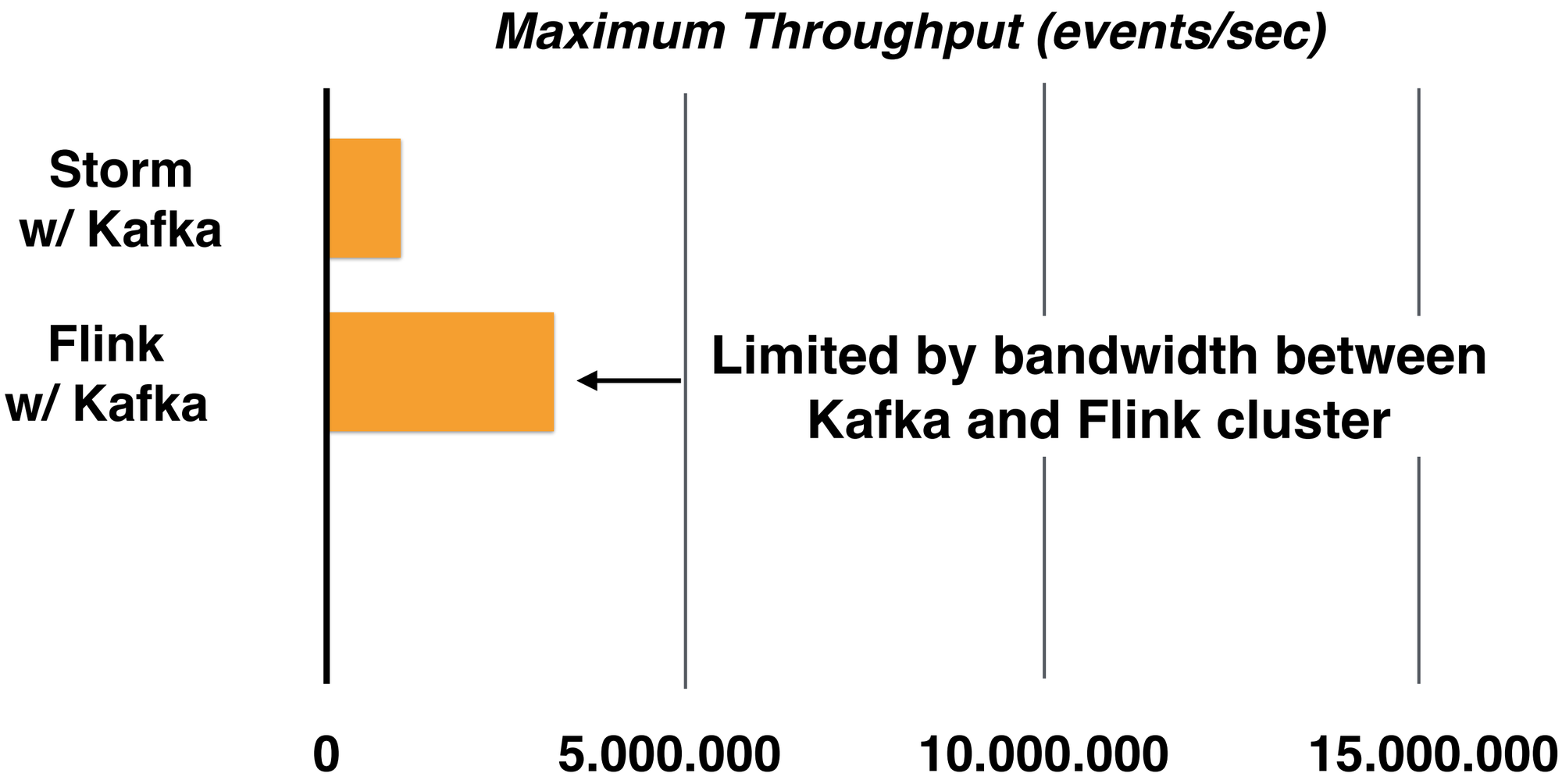
Extending the Benchmark



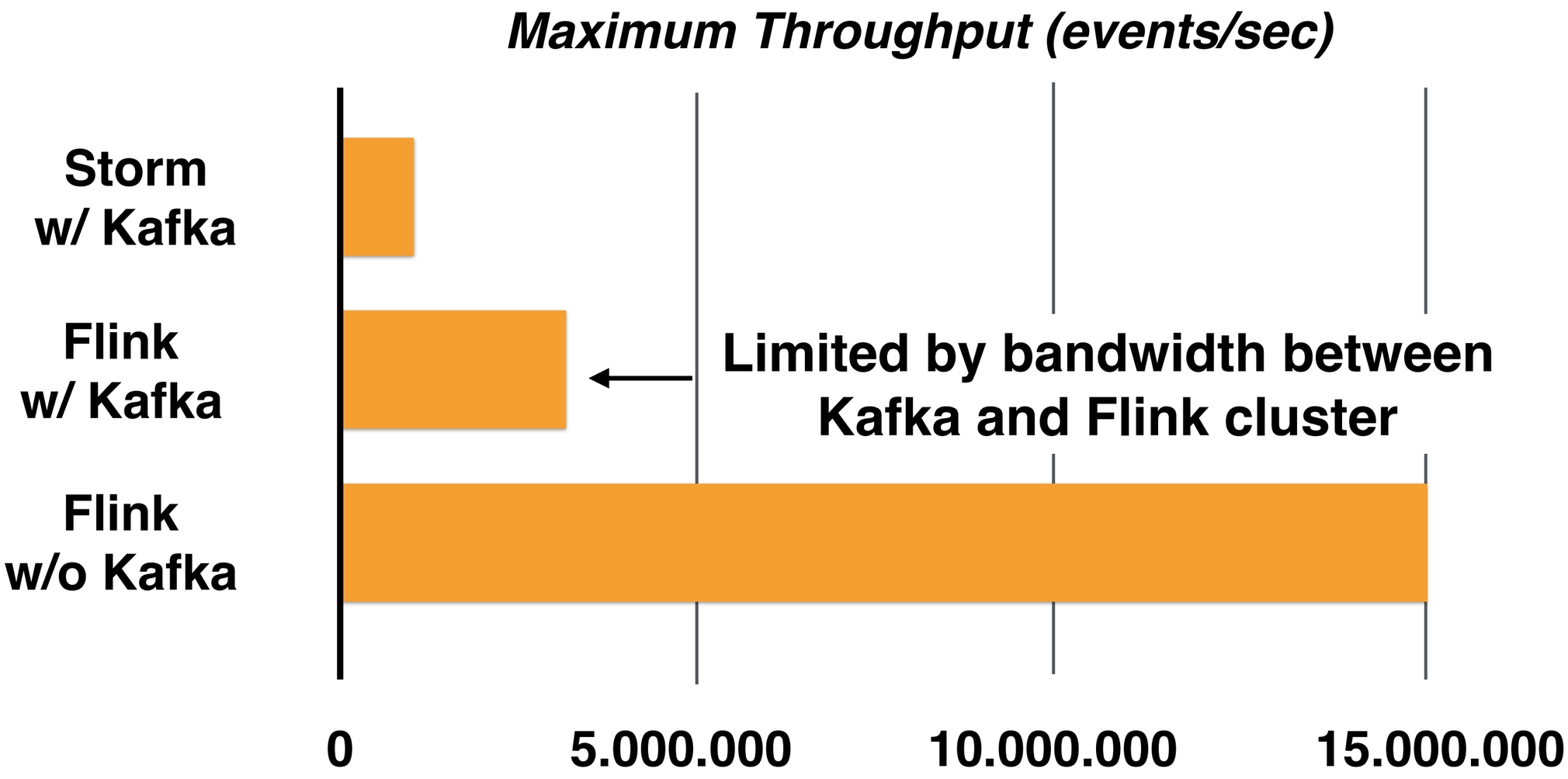
Throughput (Higher is Better)



Throughput (Higher is Better)



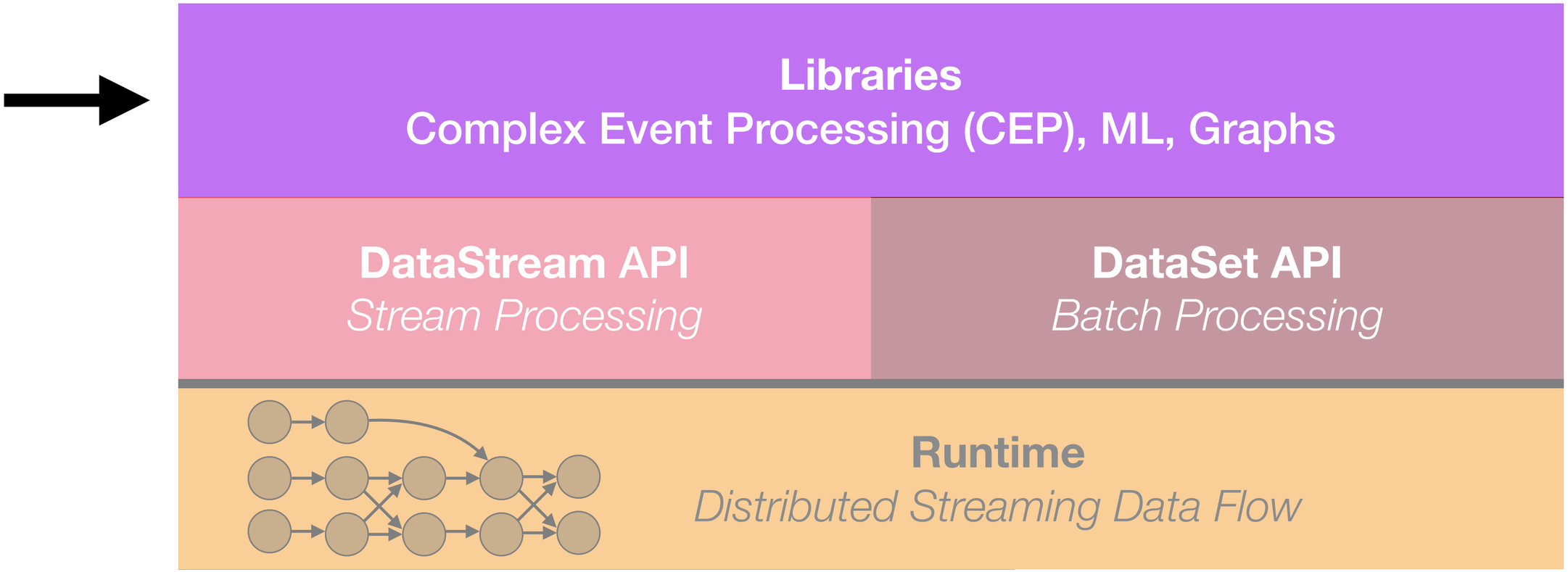
Throughput (Higher is Better)



Summary

- Stream processing is gaining momentum, the **right paradigm** for continuous data applications
- Choice of **framework is crucial** – even seemingly simple applications become complex at scale and in production
- **Flink** offers **unique combination** of *efficiency*, *consistency* and *event time handling*

Libraries


Libraries
Complex Event Processing (CEP), ML, Graphs
DataStream API
Stream Processing
DataSet API
Batch Processing
Runtime
Distributed Streaming Data Flow

Complex Event Processing (CEP)

```
Pattern<MonitoringEvent, ?> warningPattern =
  Pattern.<MonitoringEvent>begin("First Event")
    .subtype(TemperatureEvent.class)
    .where(evt -> evt.getTemperature() >= THRESHOLD)
    .next("Second Event")
    .subtype(TemperatureEvent.class)
    .where(evt -> evt.getTemperature() >= THRESHOLD)
    .within(Time.seconds(10));
```

Upcoming Features

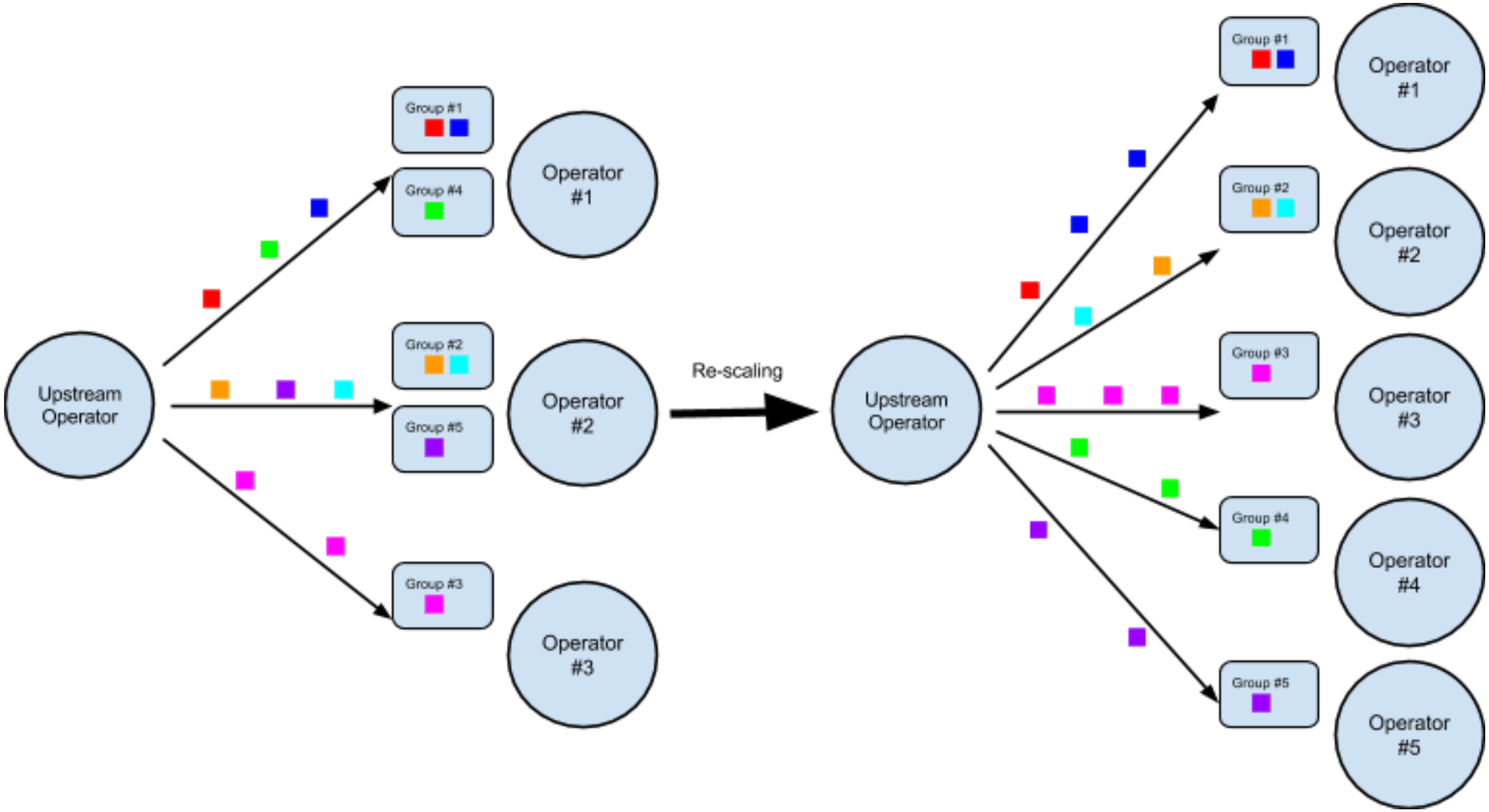
- **SQL**: ongoing work in collaboration with Apache Calcite
- **Dynamic Scaling**: adapt resources to stream volume, scale up for historical stream processing
- **Queryable State**: query the state inside the stream processor

SQL

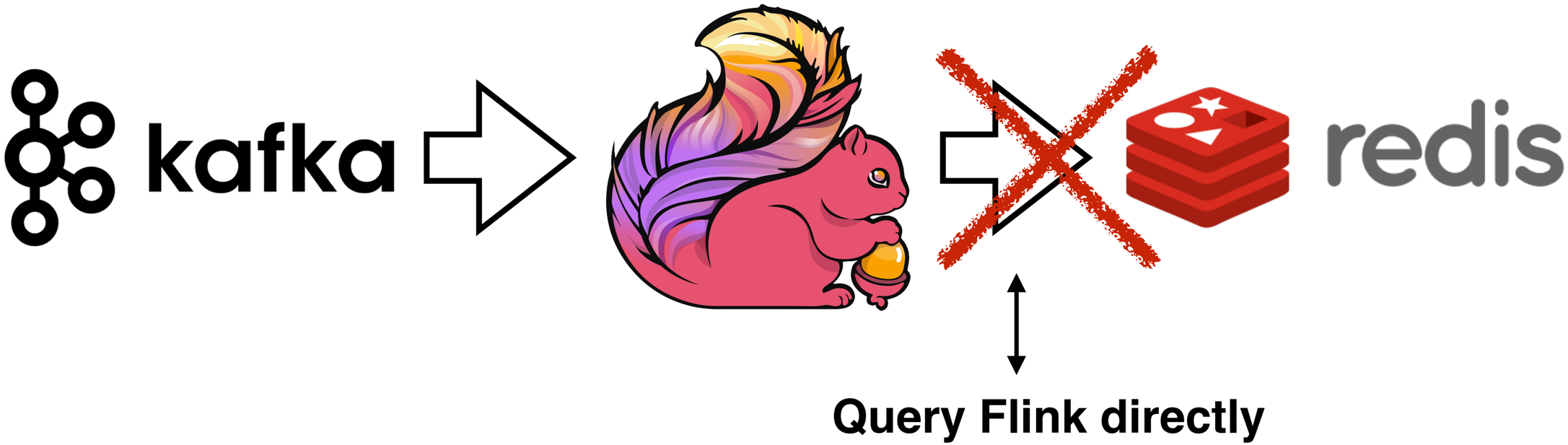
```
SELECT STREAM * FROM Orders WHERE units > 3;
```

rowtime	productId	orderId	units
10:17:00	30	5	4
10:18:07	30	8	20
11:02:00	10	9	6
11:09:30	40	11	12
11:24:11	10	12	4
…	…	…	…

Dynamic Scaling



Queryable State



Join the Community

Read

http://flink.apache.org/blog

http://data-artisans.com/blog

Follow

@ApacheFlink

@dataArtisans

Subscribe

(news | user | dev)@flink.apache.org