The Stream Processor as a Database

Ufuk Celebi
@iamuce

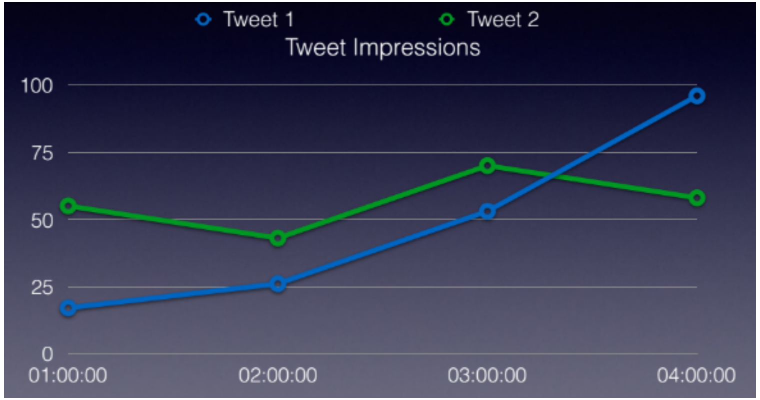
dataArtisans

Realtime Counts and Aggregates

The (Classic) Use Case

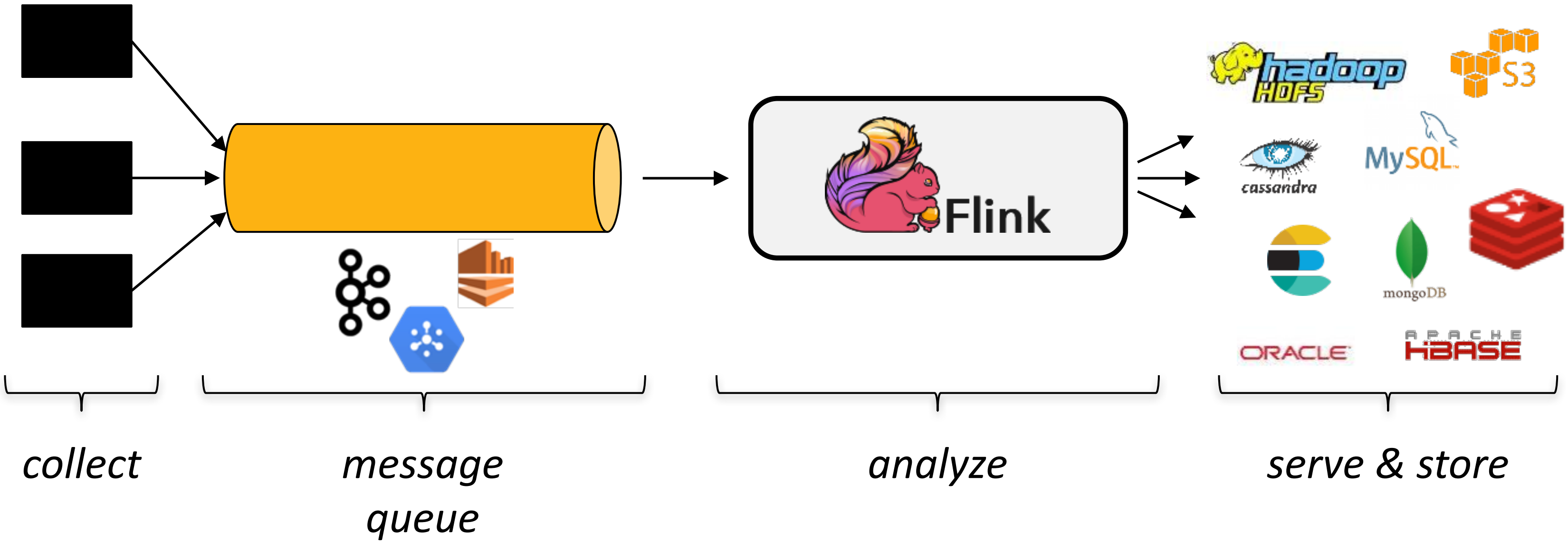
(Real-)Time Series Statistics

Stream of Events



Real-time Statistics

The Architecture



The Flink Job

```scala
case class Impressions(id: String, impressions: Long)

val events: DataStream[Event] = env
    .addSource(new FlinkKafkaConsumer09(…))

val impressions: DataStream[Impressions] = events
    .filter(evt => evt.isImpression)
    .map(evt => Impressions(evt.id, evt.numImpressions)

val counts: DataStream[Impressions]= stream
    .keyBy("id")
    .timeWindow(Time.hours(1))
    .sum("impressions")
```

The Flink Job

```scala
case class Impressions(id: String, impressions: Long)

val events: DataStream[Event] = env
    .addSource(new FlinkKafkaConsumer09(…))

val impressions: DataStream[Impressions] = events
    .filter(evt => evt.isImpression)
    .map(evt => Impressions(evt.id, evt.numImpressions)

val counts: DataStream[Impressions]= stream
    .keyBy("id")
    .timeWindow(Time.hours(1))
    .sum("impressions")
```

The Flink Job

```
case class Impressions(id: String, impressions: Long)

val events: DataStream[Event] = env
   .addSource(new FlinkKafkaConsumer09(…))

val impressions: DataStream[Impressions] = events
   .filter(evt => evt.isImpression)
   .map(evt => Impressions(evt.id, evt.numImpressions)

val counts: DataStream[Impressions]= stream
   .keyBy("id")
   .timeWindow(Time.hours(1))
   .sum("impressions")
```

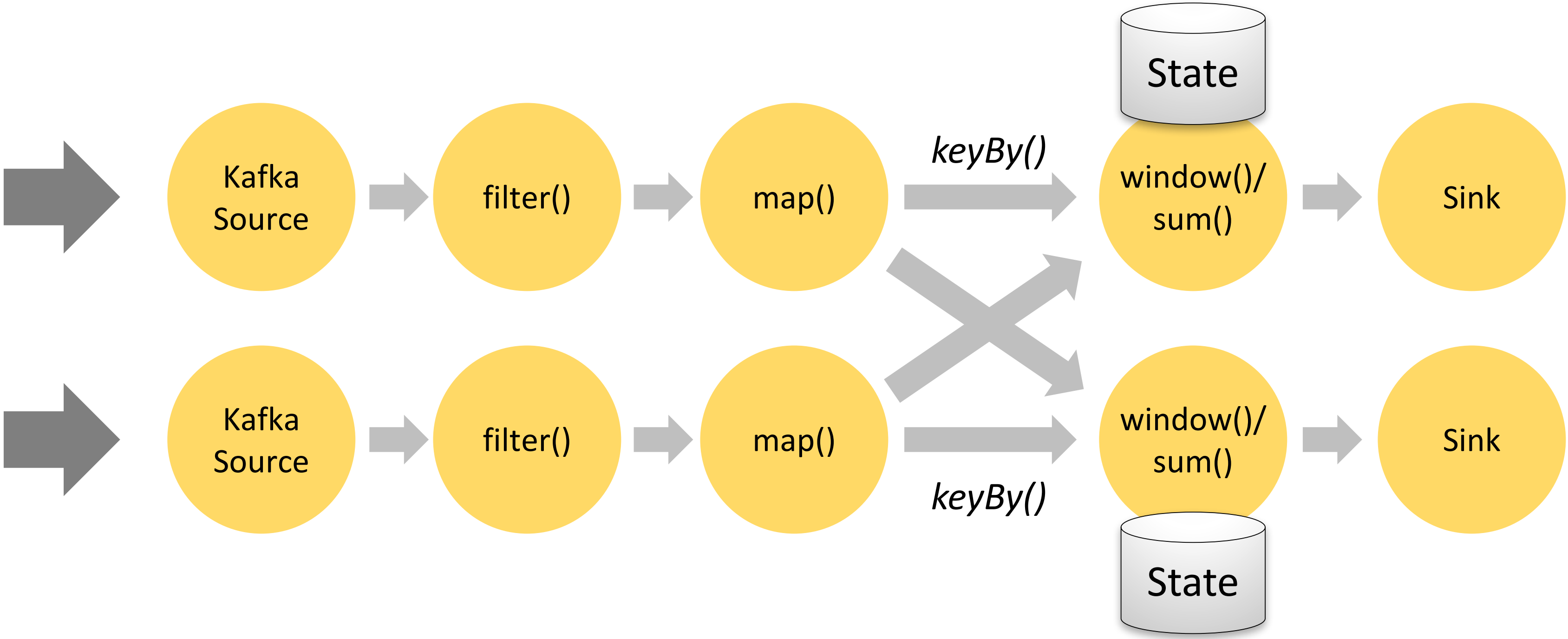
The Flink Job

```scala
case class Impressions(id: String, impressions: Long)

val events: DataStream[Event] = env
    .addSource(new FlinkKafkaConsumer09(…))

val impressions: DataStream[Impressions] = events
    .filter(evt => evt.isImpression)
    .map(evt => Impressions(evt.id, evt.numImpressions)

val counts: DataStream[Impressions]= stream
    .keyBy("id")
    .timeWindow(Time.hours(1))
    .sum("impressions")
```

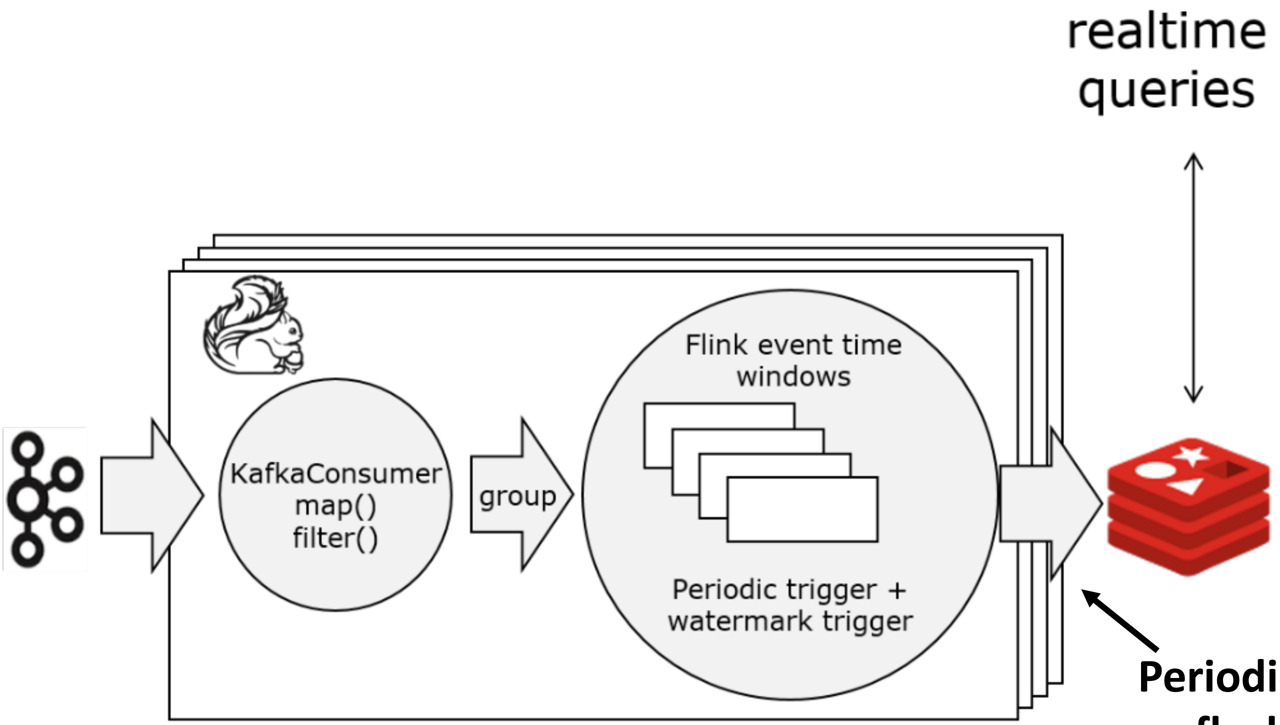
The Flink Job

```
case class Impressions(id: String, impressions: Long)

val events: DataStream[Event] = env
   .addSource(new FlinkKafkaConsumer09(…))

val impressions: DataStream[Impressions] = events
   .filter(evt => evt.isImpression)
   .map(evt => Impressions(evt.id, evt.numImpressions)

val counts: DataStream[Impressions]= stream
   .keyBy("id")
   .timeWindow(Time.hours(1))
   .sum("impressions")
```

The Flink Job

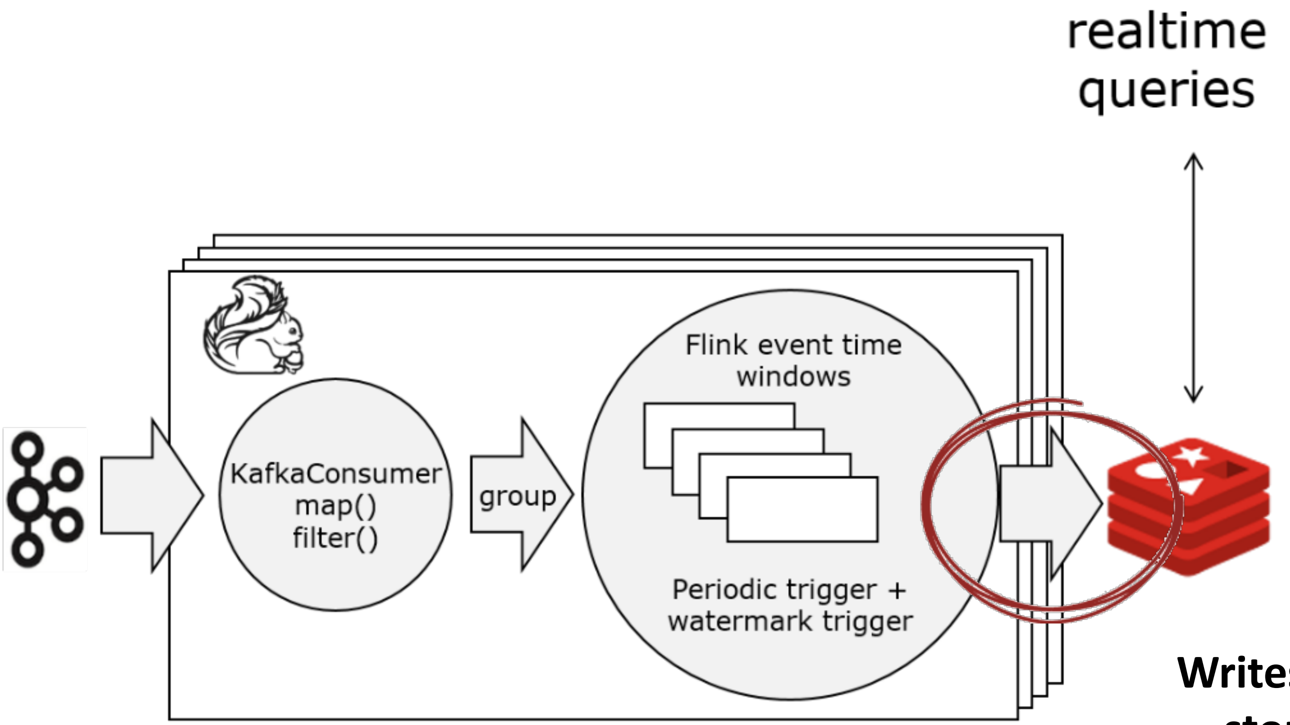


Putting it all together



Periodically (every second) flush new aggregates to Redis

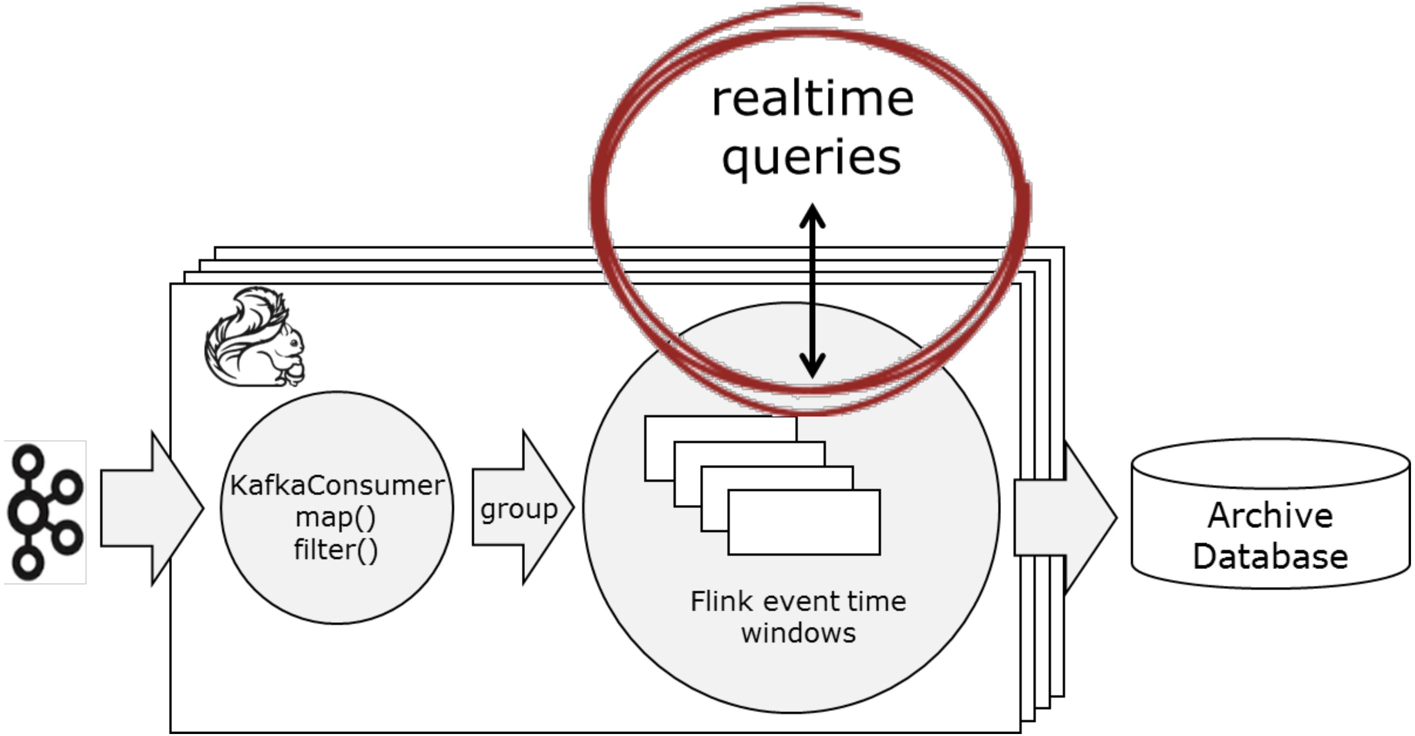
The Bottleneck



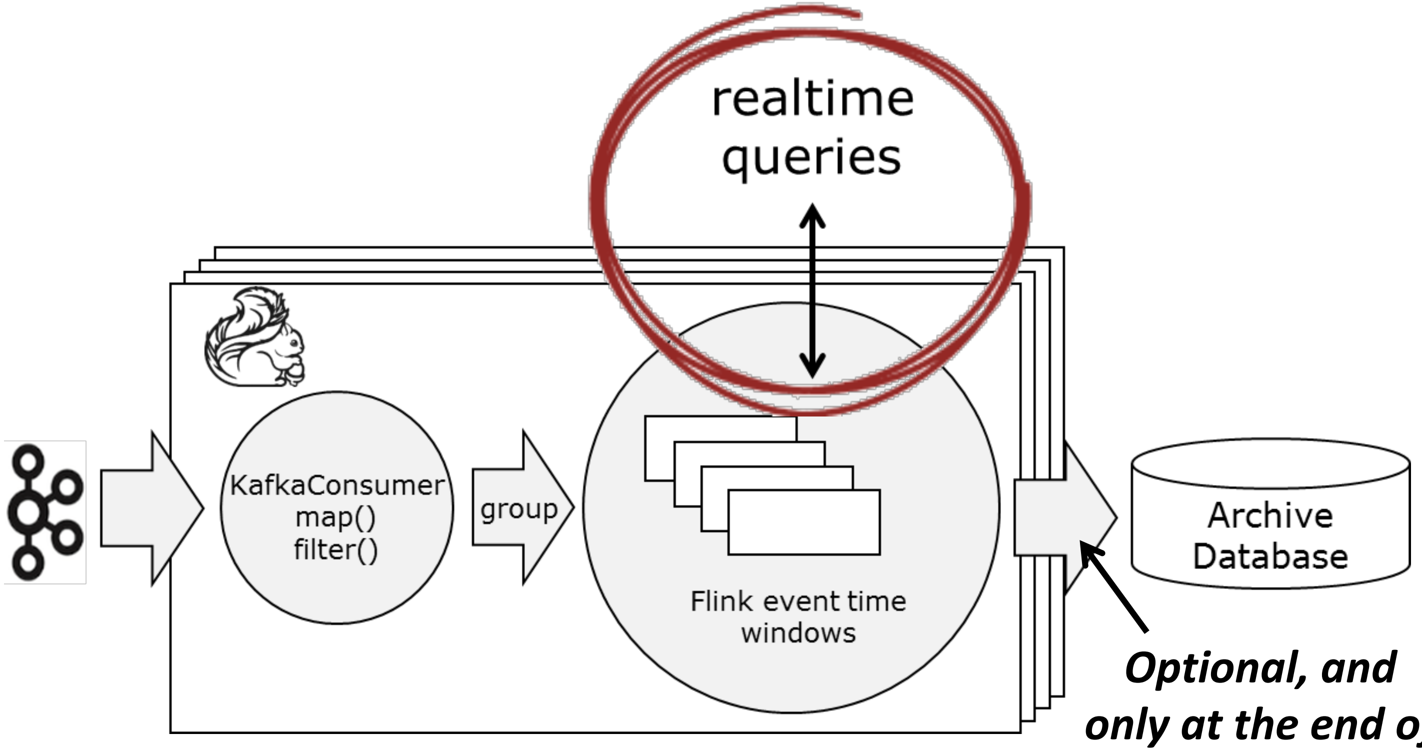
Writes to the key/value store take too long

Queryable State

Queryable State

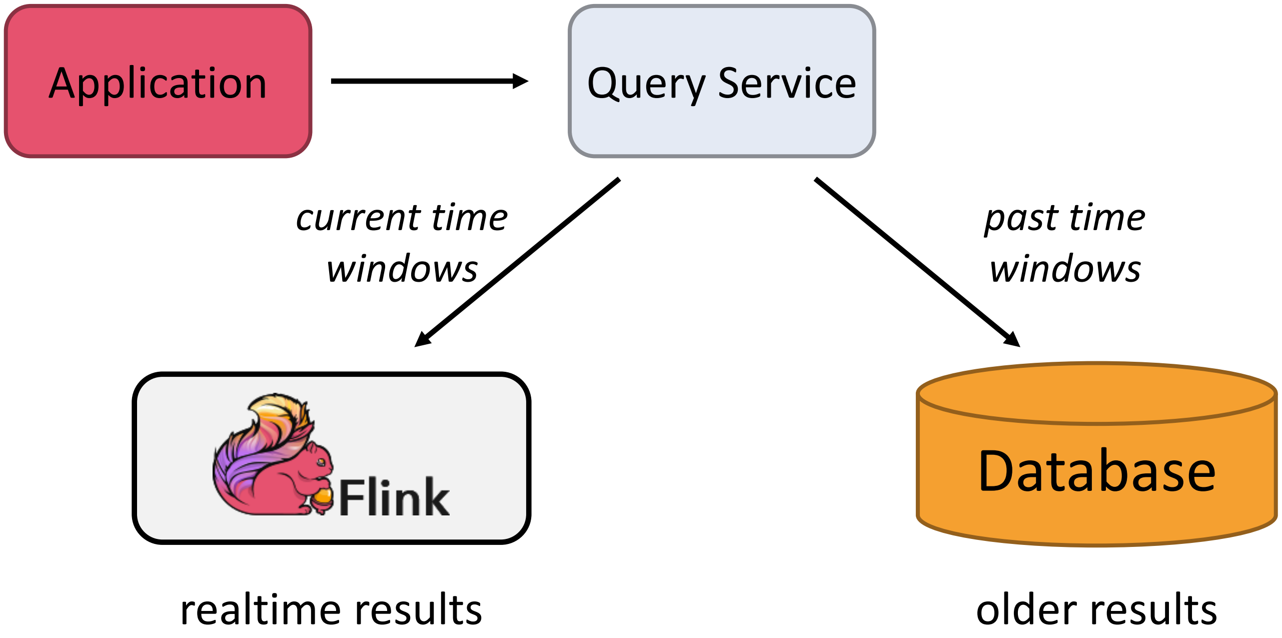


Queryable State



Optional, and only at the end of windows

Queryable State: Application View

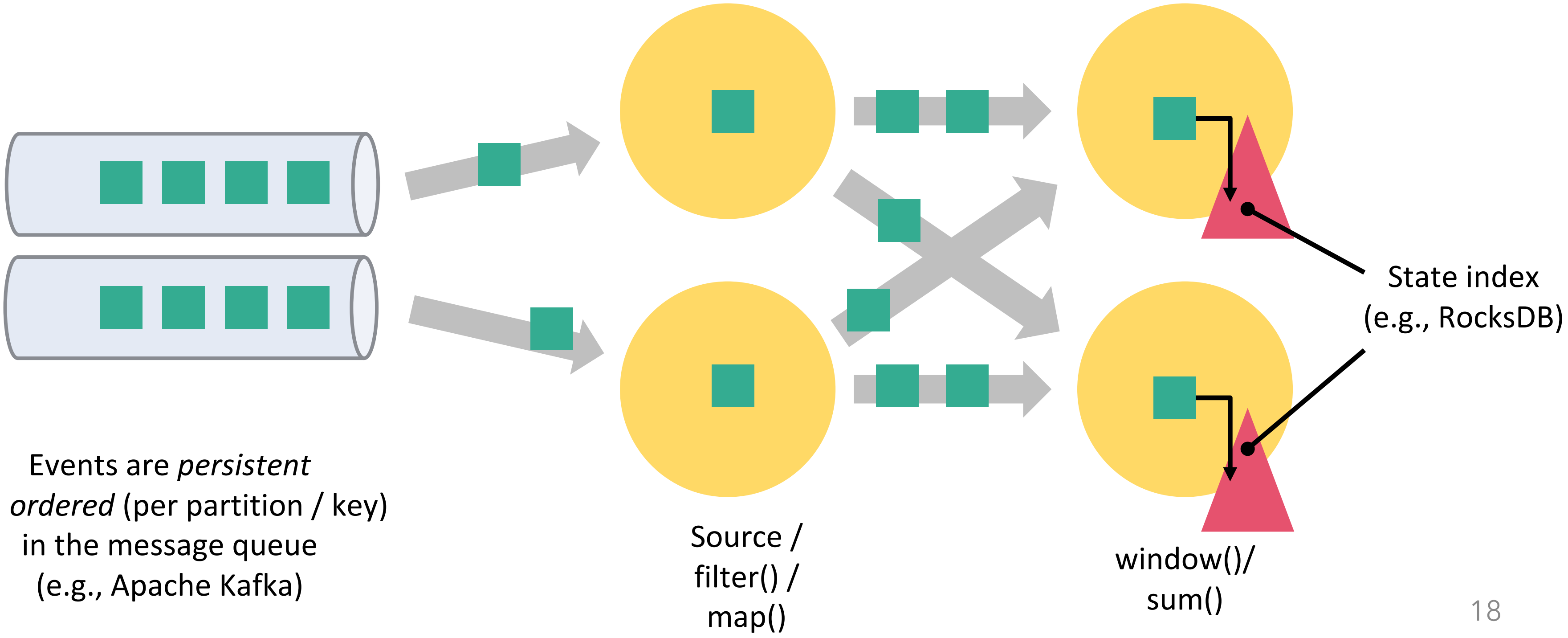


Queryable State Enablers

- Flink has state as a **first class citizen**
- State is **fault tolerant** (exactly once semantics)
- State is **partitioned** (sharded) together with the operators that create/update it
- State is **continuous** (not mini batched)
- State is **scalable**

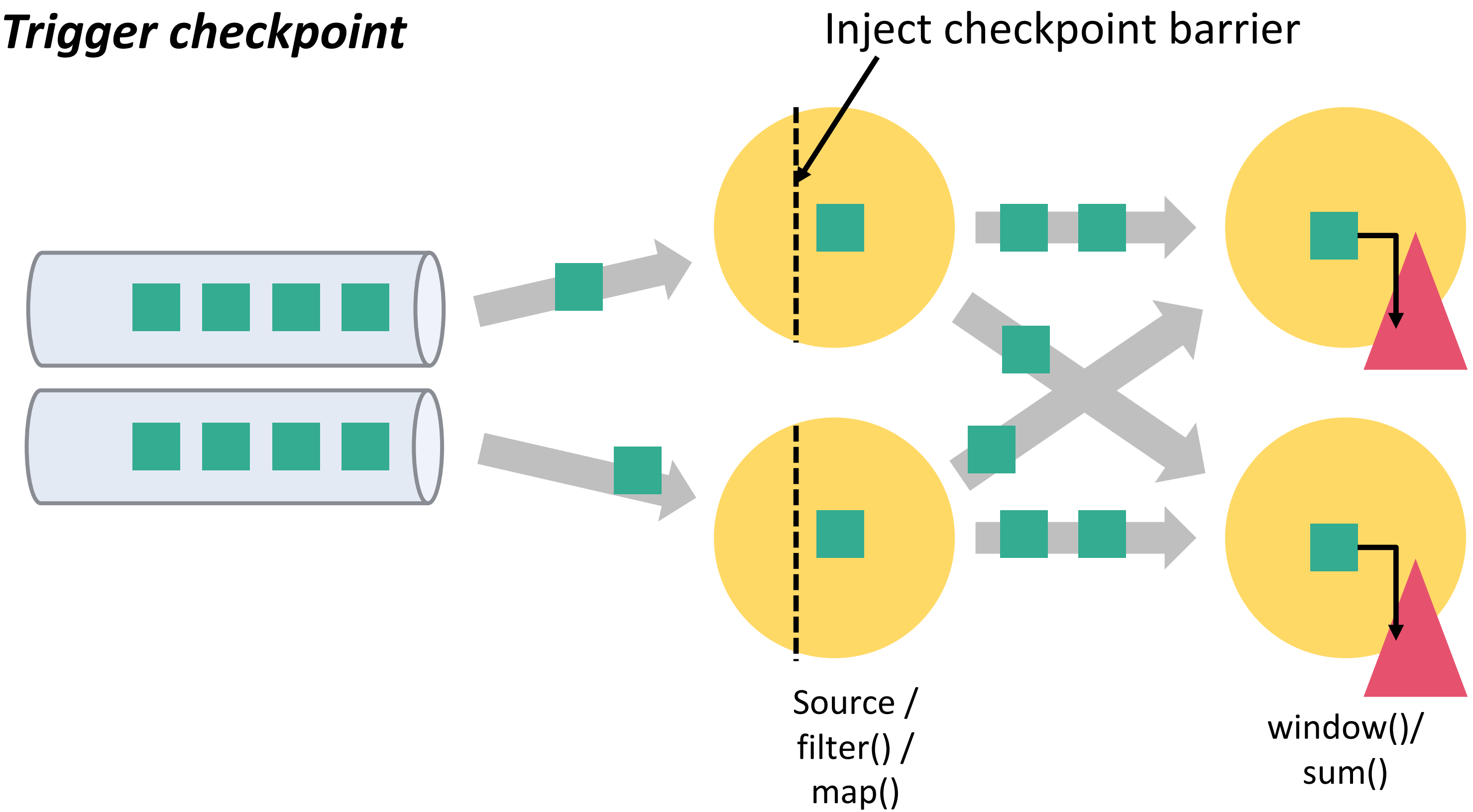
State in Flink

Events flow ***without replication*** *or* ***synchronous writes***



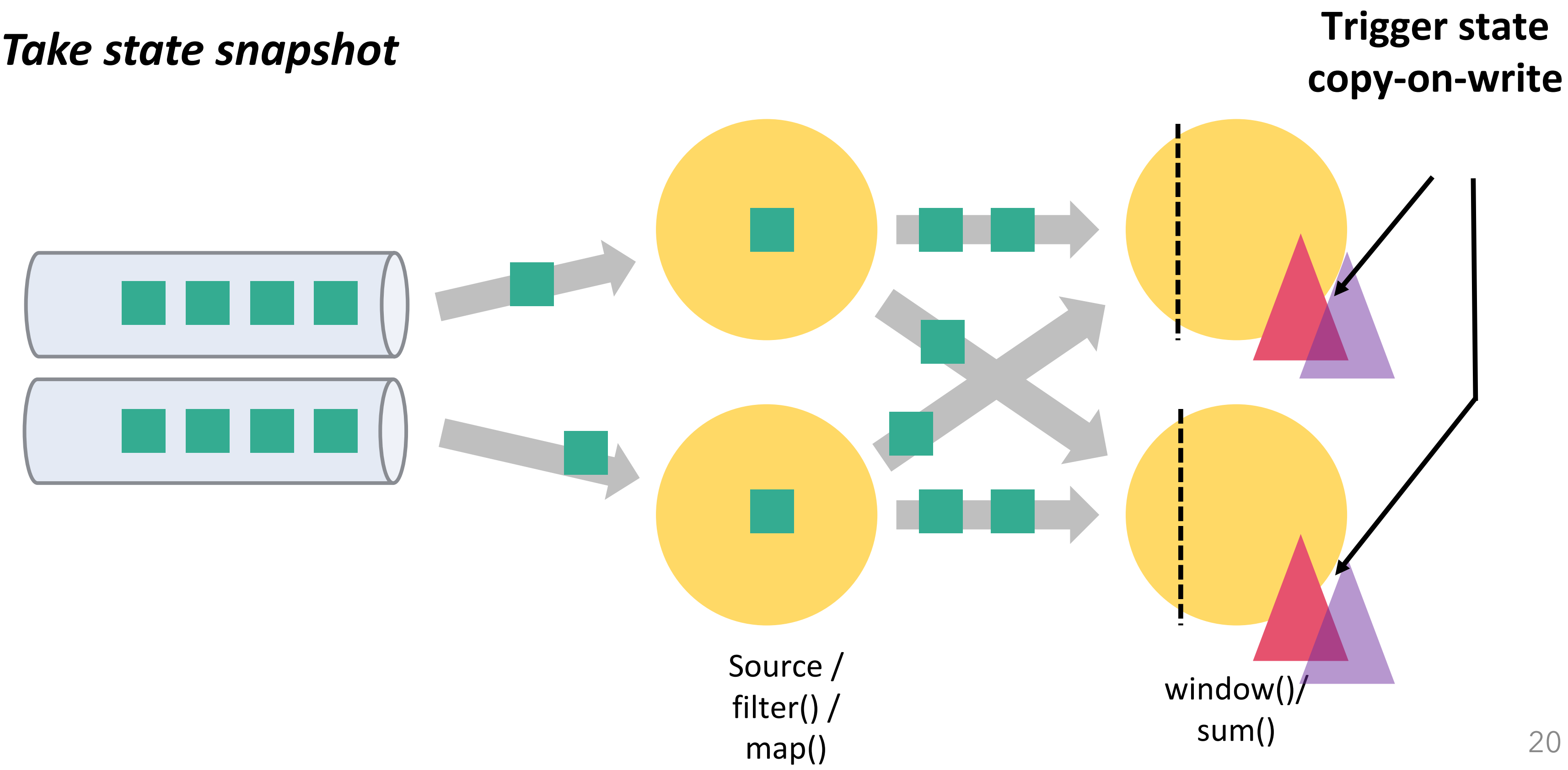
State in Flink

Trigger checkpoint



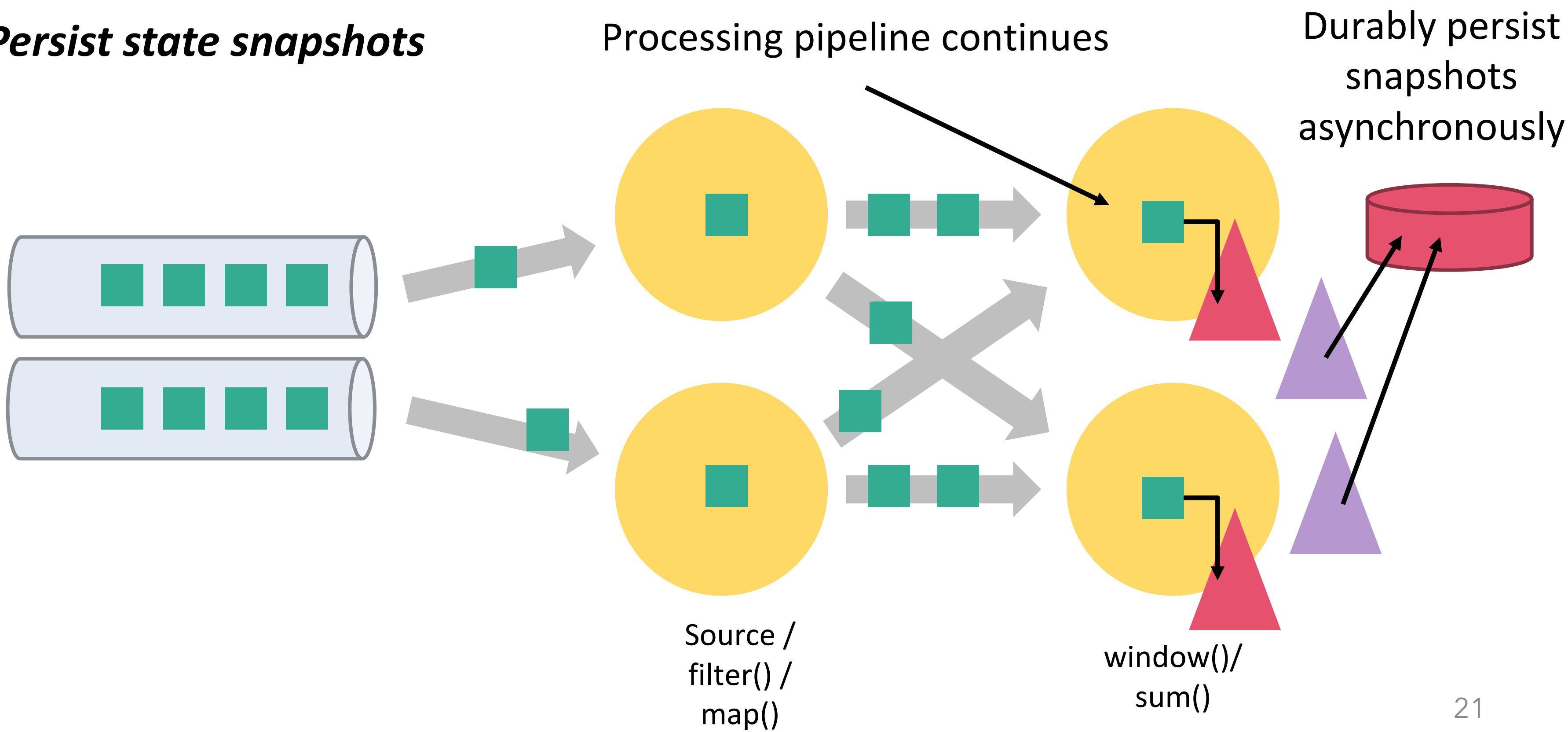
State in Flink

Take state snapshot



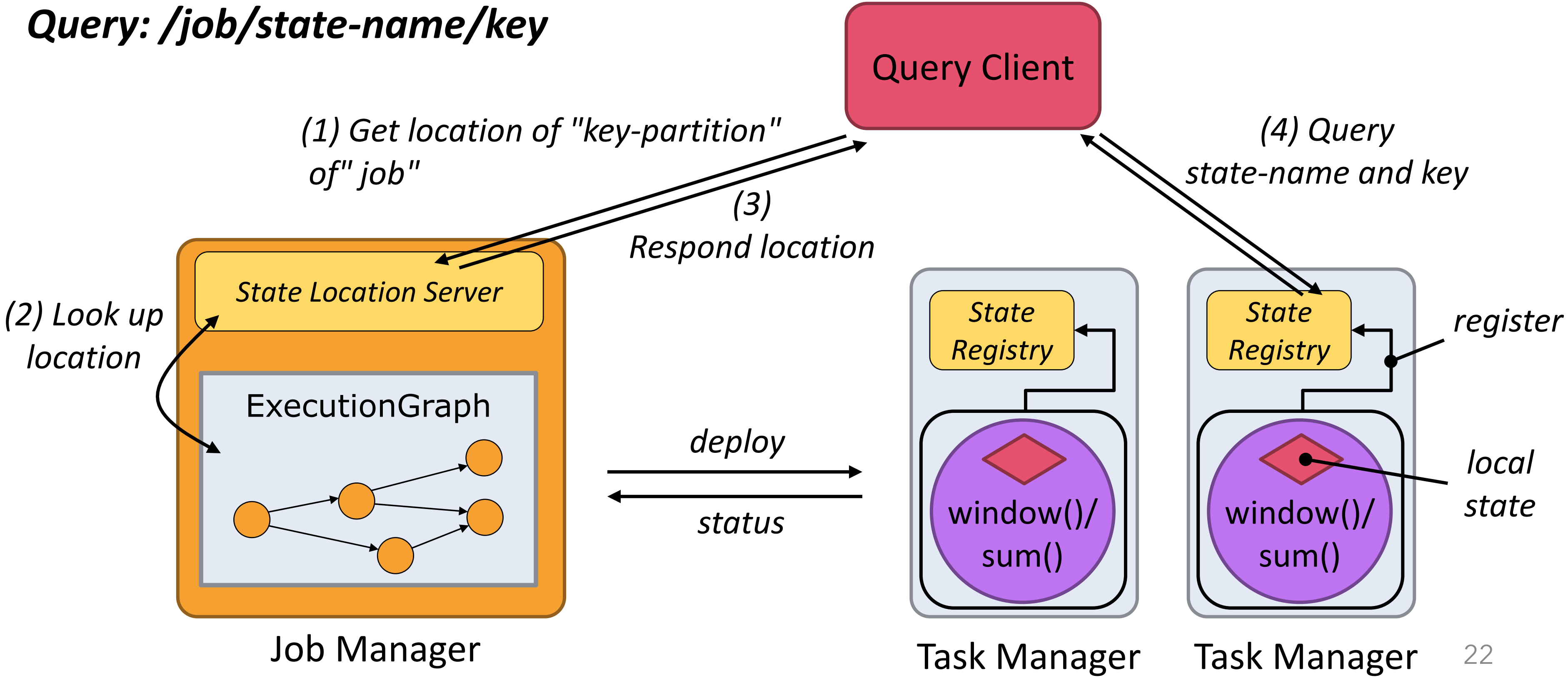
State in Flink

Persist state snapshots

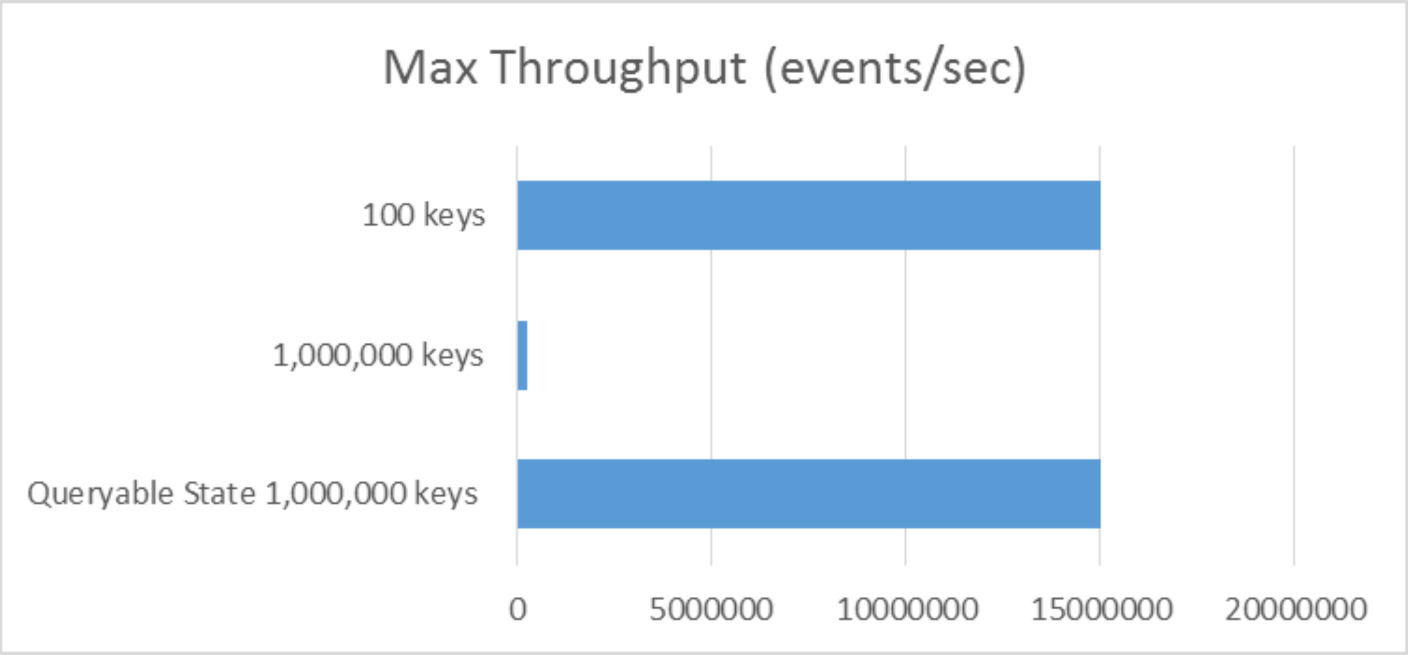


Queryable State: Implementation

Query: /job/state-name/key



Queryable State Performance



Conclusion

Takeaways

- Streaming applications are often not bound by the stream processor itself. **Cross system interaction** is **frequently biggest bottleneck**

- **Queryable state mitigates a big bottleneck**: Communication with external key/value stores to publish realtime results

- **Apache Flink**'s sophisticated support for state makes this possible

Takeaways

Performance of Queryable State

- Data persistence is fast with logs
  - Append only, and streaming replication
- Computed state is fast with local data structures and no synchronous replication
- Flink's checkpoint method makes computed state persistent with low overhead

Questions?

- eMail: uce@apache.org
- Twitter: @iamuce
- Code/Demo: https://github.com/dataArtisans/flink-queryable_state_demo

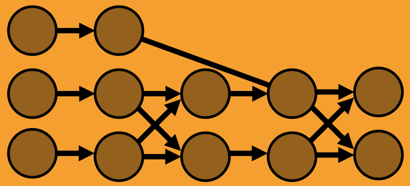
Appendix

Flink Runtime + APIs

Table API & Stream SQL

DataStream API

ProcessFunction API



Runtime

Distributed Streaming Data Flow

Building Blocks: **Streams, Time, State**

Apache Flink Architecture Review