King

RBea: Scalable Real-Time Analytics at King

Gyula Fóra

Data Warehouse Engineer

Apache Flink PMC

About King

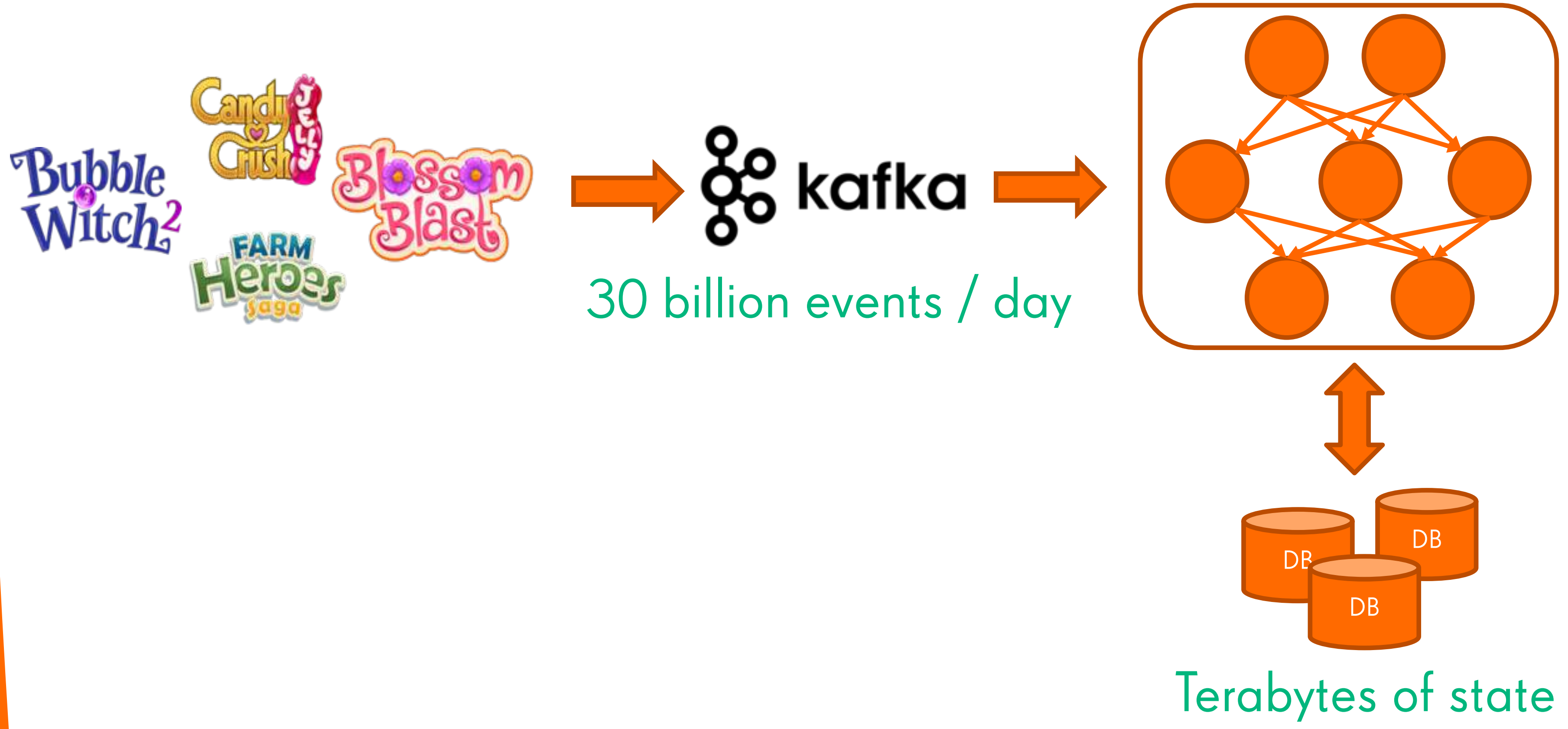
We make awesome mobile games

463 million monthly active users

30 billion events per day

And a lot of data...

From streaming perspective...



How do we use Flink

Standalone deployment

Few heavy streaming jobs

RocksDB state backend with caching

Lot of custom tooling

The RBea platform

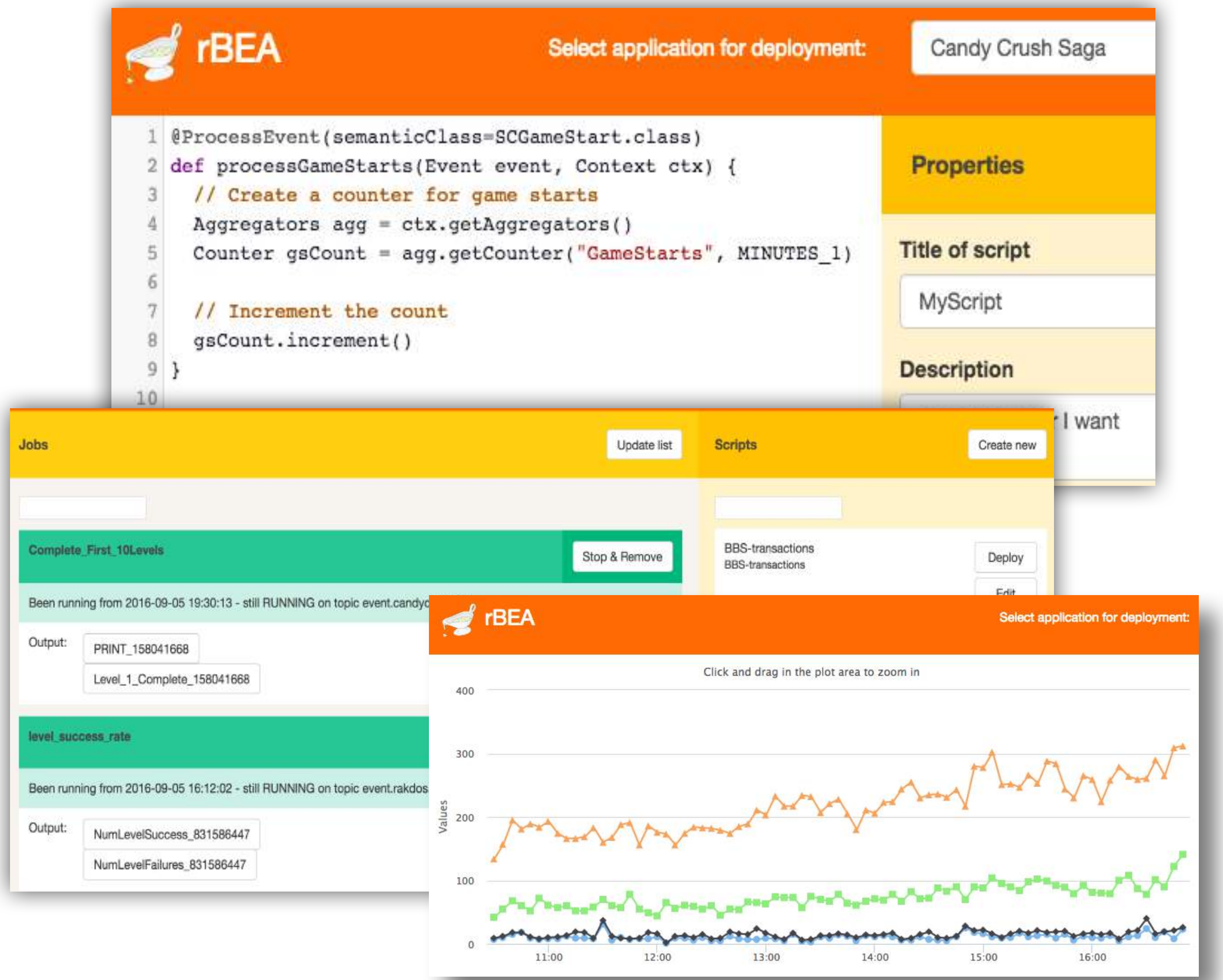
Scripting on the live streams

Deploy from browser/python

Window aggregates

Stateful computations

Scalable + fault tolerant



```
@ProcessEvent(semanticClass=SCGameStart.class)
def processGameStarts(Event event, Context ctx) {
  // Create a counter for game starts
  Aggregators agg = ctx.getAggregators()
  Counter gsCount = agg.getCounter("GameStarts", MINUTES_1)

  // Increment the count
  gsCount.increment()
}
```

Complete_First_10Levels

Been running from 2016-09-05 19:30:13 - still RUNNING on topic event.candyc

Output: PRINT_158041668, Level_1_Complete_158041668

level_success_rate

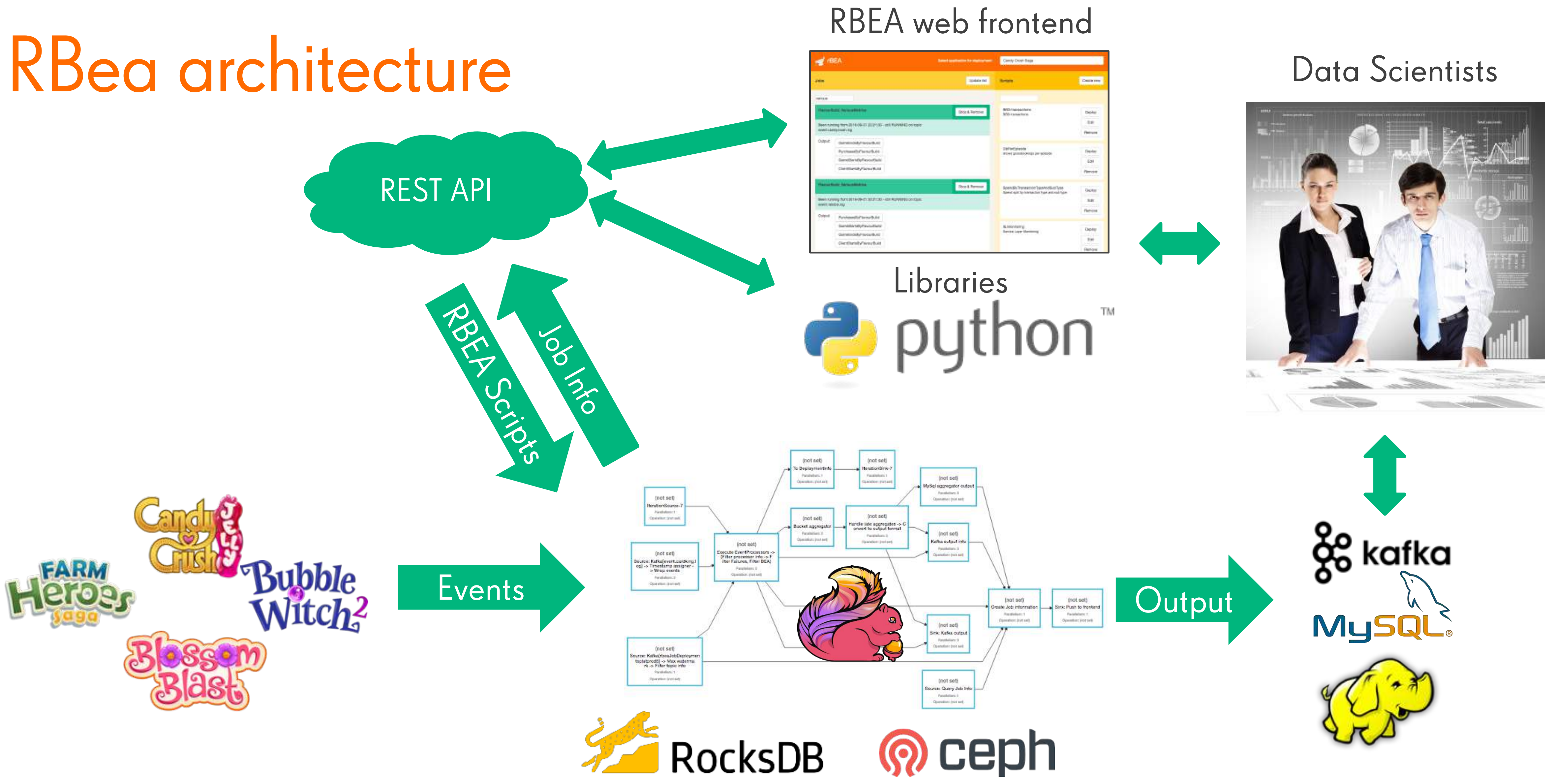
Been running from 2016-09-05 16:12:02 - still RUNNING on topic event.rakdos

Output: NumLevelSuccess_831586447, NumLevelFailures_831586447

BBS-transactions
BBS-transactions

rBEA
Select application for deployment:
Click and drag in the plot area to zoom in
Values
400
300
200
100
0
11:00
12:00
13:00
14:00
15:00
16:00

RBea architecture



http://hpc-asia.com/wp-content/uploads/2015/09/World-Class-Consultancy-Seeking-Data-Scientist-CA-Hobson-Associates-Matthew-Abel-Recruiter.jpg

RBea backend implementation

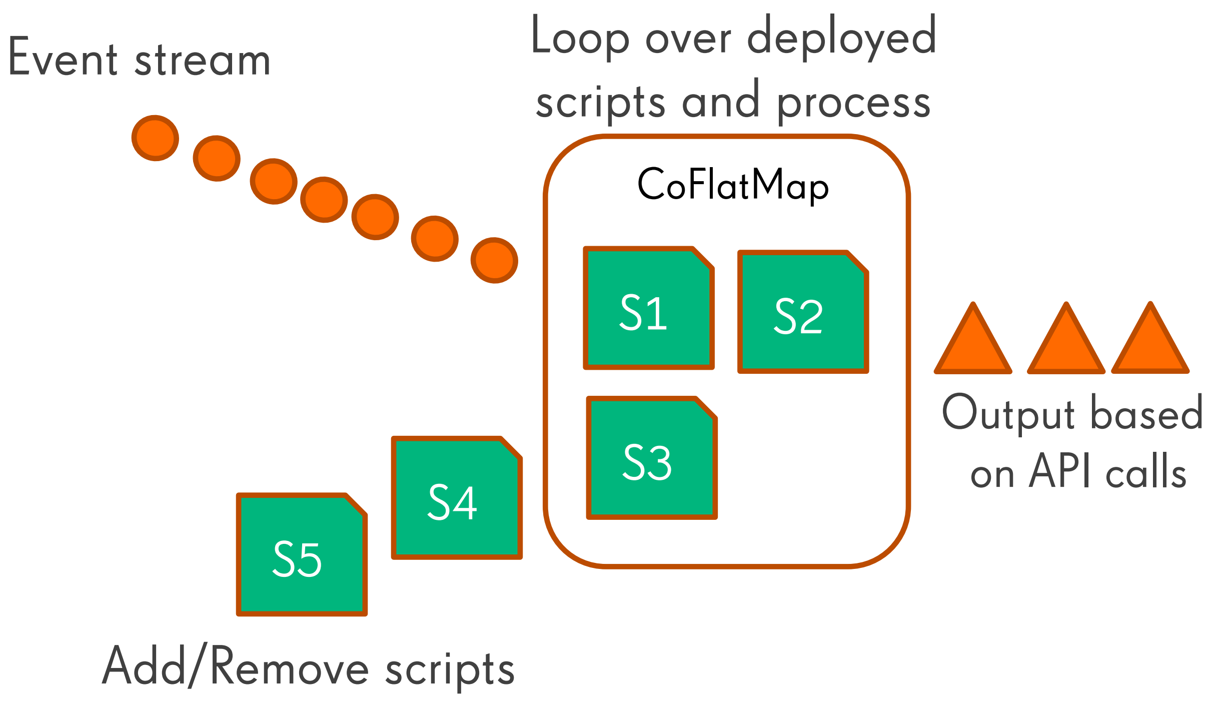
One stateful Flink job / game

Stream events and scripts

Events are partitioned by user id

Scripts are broadcasted

Output/Aggregation happens downstream


Event stream
Loop over deployed scripts and process
CoFlatMap
S1
S2
S3
S4
S5
Output based on API calls
Add/Remove scripts

Dissecting the DSL

```
@ProcessEvent(semanticClass=SCPurchase.class)
def process(SCPurchase purchase,
            Output out,
            Aggregators agg) {

   long amount = purchase.getAmount()
   String curr = purchase.getCurrency()
   out.writeToKafka("purchases", curr + "\t" + amount)

   Counter numPurchases = agg.getCounter("PurchaseCount", MINUTES_10)
   numPurchases.increment()
}
```

Dissecting the DSL

```
@ProcessEvent(semanticClass=SCPurchase.class)
def process(SCPurchase purchase,
            Output out,
            Aggregators agg) {

   long amount = purchase.getAmount()
   String curr = purchase.getCurrency()
   out.writeToKafka("purchases", curr + "\t" + amount)

   Counter numPurchases = agg.getCounter("PurchaseCount", MINUTES_10)
   numPurchases.increment()
}
```

Processing methods by annotation

Event filter conditions

Flexible argument list

Code-generate Java classes

=> void processEvent(Event e, Context ctx);

Dissecting the DSL

```
@ProcessEvent(semanticClass=SCPurchase.class)
def process(SCPurchase purchase,
            Output out,
            Aggregators agg) {

   long amount = purchase.getAmount()
   String curr = purchase.getCurrency()
   out.writeToKafka("purchases", curr + "\t" + amount)

   Counter numPurchases = agg.getCounter("PurchaseCount", MINUTES_10)
   numPurchases.increment()
}
```

Output calls create *Output* events

Output(KAFKA, "purchases", "…")

These events are filtered downstream and sent to a Kafka sink

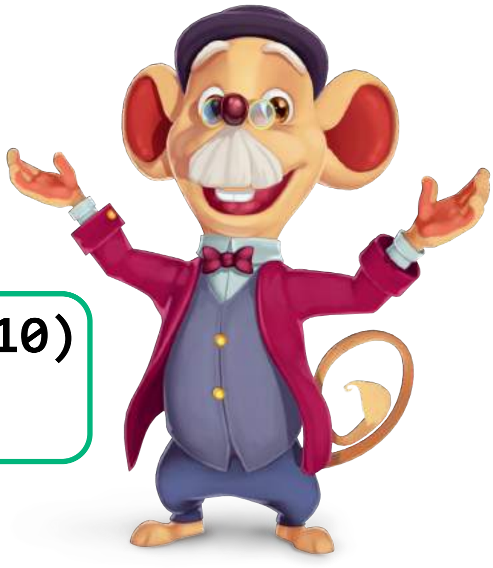
Dissecting the DSL

```
@ProcessEvent(semanticClass=SCPurchase.class)
def process(SCPurchase purchase,
            Output out,
            Aggregators agg) {

   long amount = purchase.getAmount()
   String curr = purchase.getCurrency()
   out.writeToKafka("purchases", curr + "\t" + amount)

   Counter numPurchases = agg.getCounter("PurchaseCount", MINUTES_10)
   numPurchases.increment()
}
```

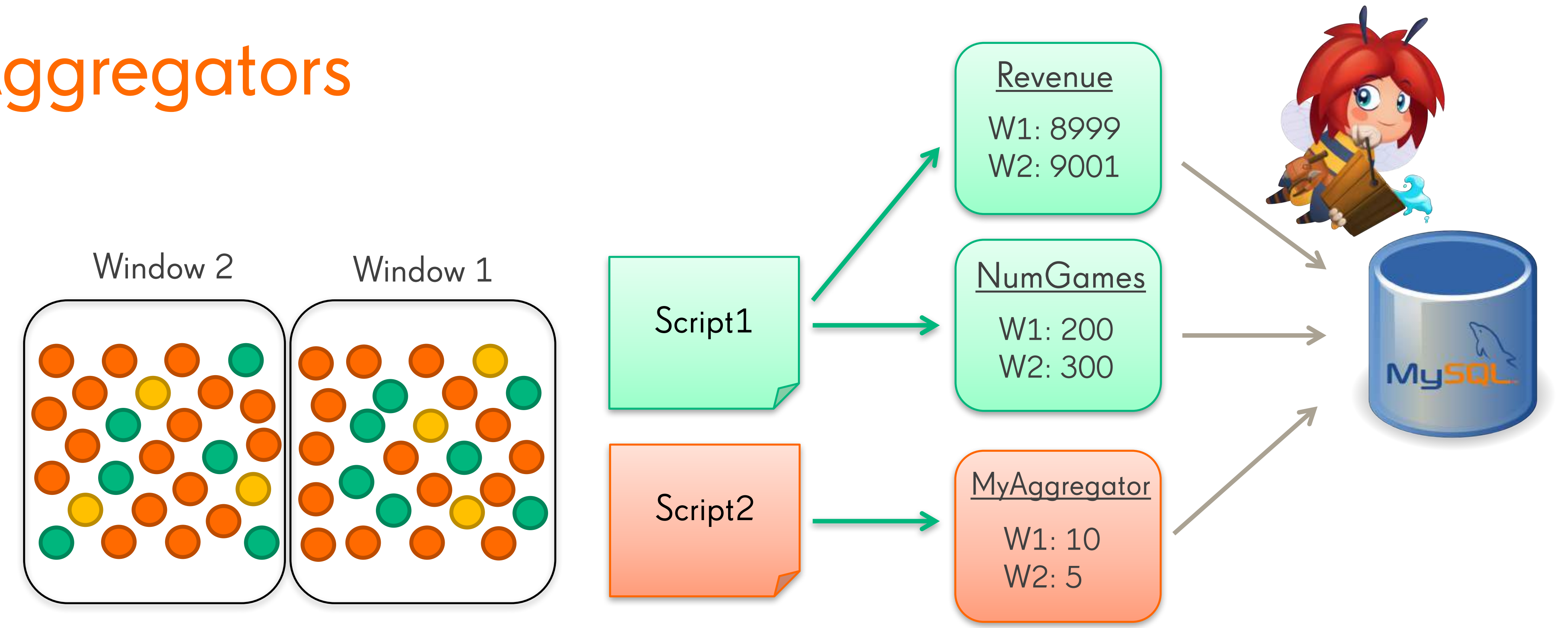
Aggregator calls create *Aggregate* events

Aggr (MYSQL, 60000, "PurchaseCount", 1)

Flink window operators do the aggregation



Aggregators

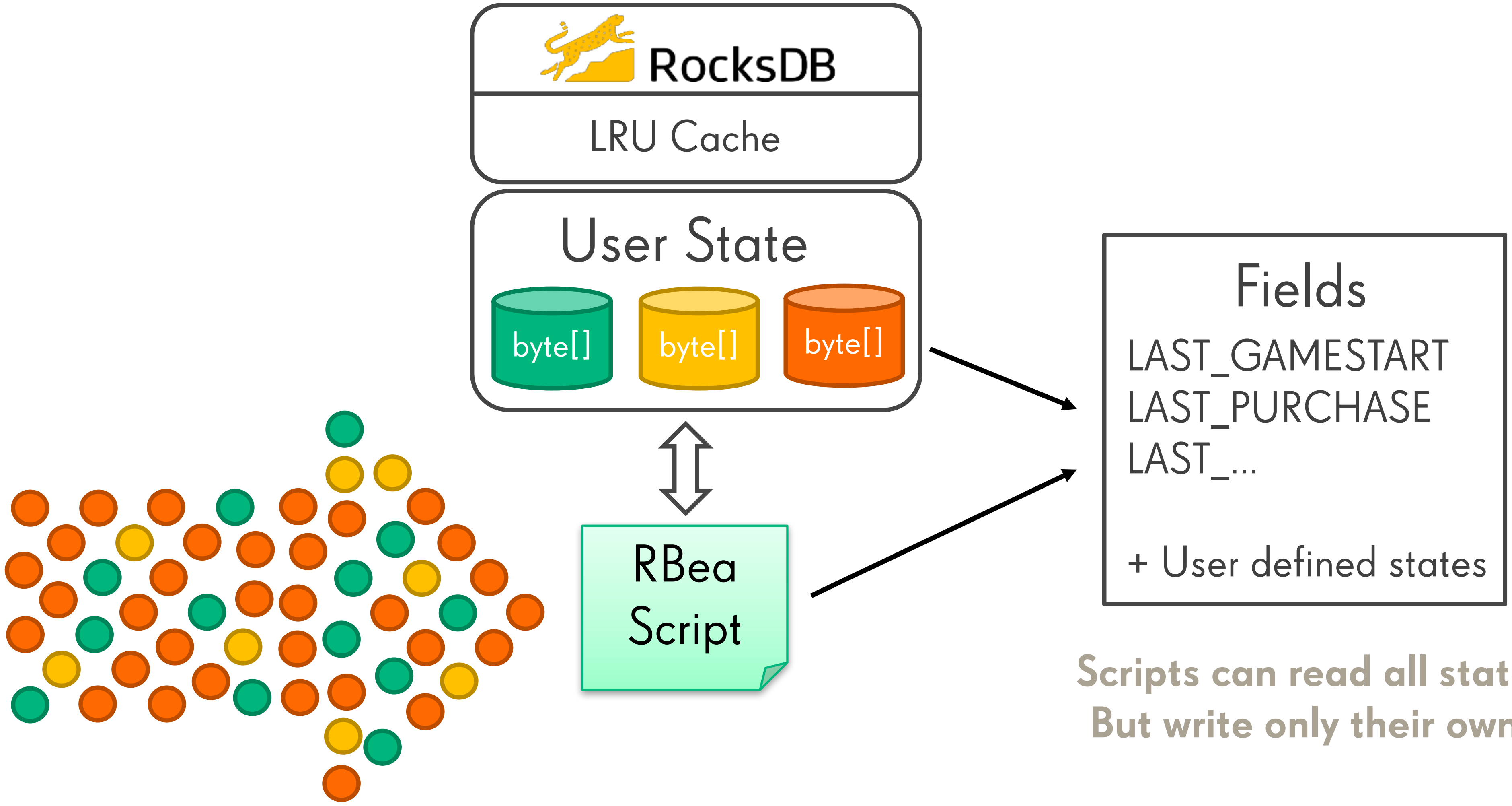


Event time windows
Window size / aggregator

Dynamic window assignment

```
long size = aggregate.getWindowSize();
long start = timestamp - (timestamp % size);
long end = start + size;
TimeWindow tw = new TimeWindow(start, end);
```

User states



Scripts can read all state
But write only their own

Things you might wonder...

Can slow scripts affect other scripts?

- Yes, but we are working on it
- Separate test/live environments

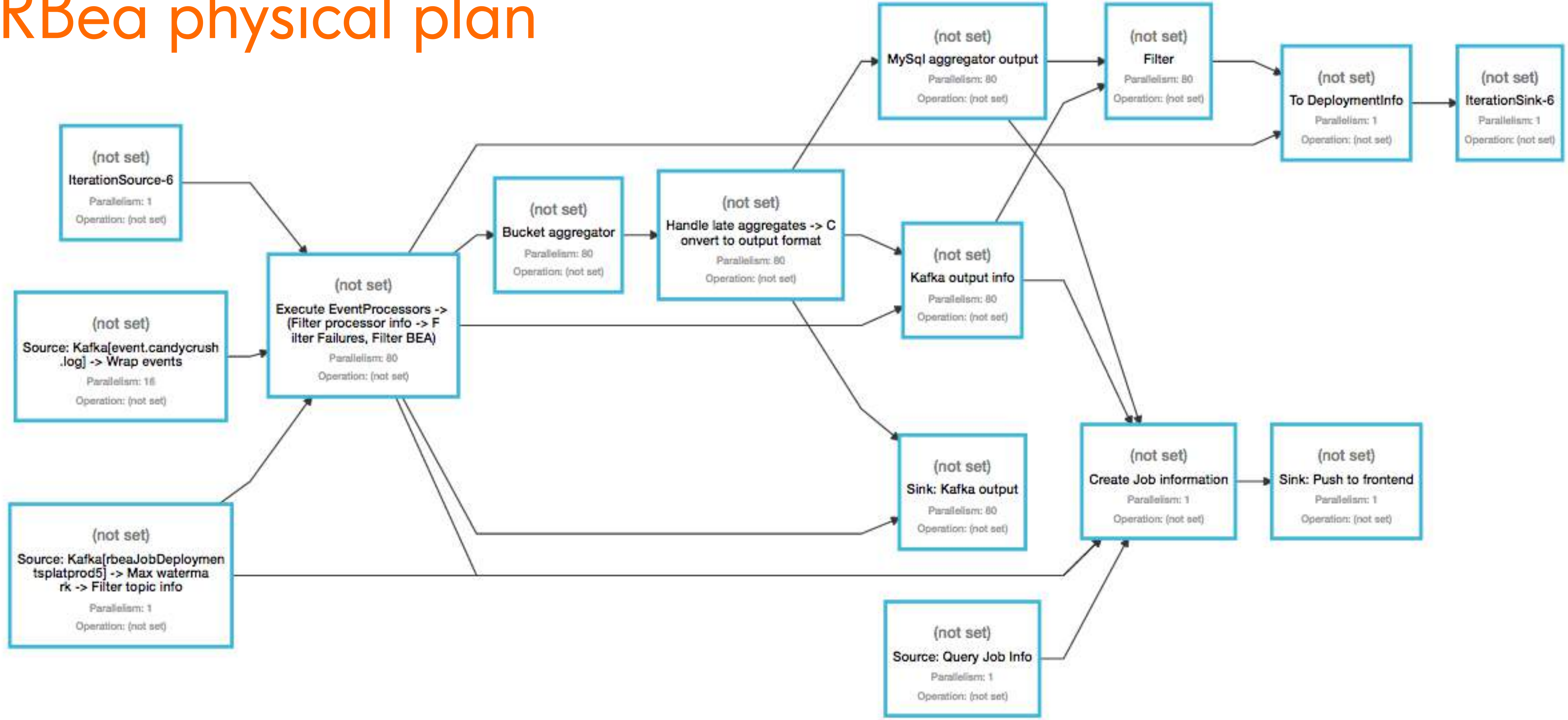
What does the backend know about the scripts?

- Outputs produced
- Failures (causes) => these are propagated
- Runtime stats in the future

Is RBea useful compared to custom Flink jobs?

- Writing + maintaining streaming jobs is hard
- Especially with state, windowing, MySQL etc.

RBea physical plan



Wrap up

RBea makes streaming accessible to every data scientist at King

We leverage Flink's stateful and windowed processing capabilities

People love it because it's simple and powerful

Thank you!