# A BIG DATA STREAMING RECIPE

## WHAT TO CONSIDER WHEN BUILDING A REAL TIME BIG DATA APPLICATION

Konstantin Gregor / konstantin.gregor@tngtech.com

# ABOUT ME

- Software developer for TNG in Munich
- Client in telecommunication business
- Mobile phone events (SMS sent, call placed etc)
- ~5 billion events per day
- Deriving statistical insights: e.g. age distribution at an event
- Ensuring absolute privacy and anonymity of customers


LET'S COOK
© Sony, Wallpapervortex.com

# INGREDIENT #1:
# STREAM PROCESSOR

# STREAM PROCESSORS

- There are many ...
- Each has its own benefits
- Thoroughly analyze which features you need!
- In this talk, focus lies on



# FUNCTIONALITY

- Generally provided functions:
  - map
  - window
  - aggregate
  - ...
- Framework-specific functionality:
  - Machine learning
  - Graph processing
  - ...

# APACHE FLINK


- Originated at TU Berlin
- Written in Java and Scala
- Processes records one-at-a-time
- APIs: Java, Scala, Python(experimental)
- SQL on Streams
- New feature: "Queryable state"

# HELLO TWITTER IN FLINK

```
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionE
DataStream<String> tweets = env.addSource( new TwitterSource(authProps));

tweets.flatMap(new JSONParseFlatMap<String, Tuple2<String, Long>>() {
                @Override
                public void flatMap(String tweet, Collector<Tuple2<Strin
                try {
                                out.collect(new Tuple2<>(getString(tweet, "lar
                } catch { ... }
        }
 })
        .keyBy(0)
        .timeWindow(Time.seconds( 10))
        .sum(1).print();

env.execute("Streaming WordCount Example");
```

# APACHE STORM


- Originated at Twitter
- Originally in Clojure, now Java
- Processes records one-at-a-time
- Can be used with any language
- Storm SQL

# HELLO TWITTER IN STORM (1/2)

```
TopologyBuilder builder = new TopologyBuilder();
builder.setSpout("twitterSpout", new MyTwitterSpout(...), 1);

builder.setBolt("countBolt", new TwitterLanguageCountBolt()
                .withTumblingWindow(new BaseWindowedBolt.Duration(10, Ti
        .fieldsGrouping("twitterSpout", new Fields("lang"));

        builder.setBolt("output", new OutputBolt(), 1).localOrShuffleGro
```

# HELLO TWITTER IN STORM (2/2)

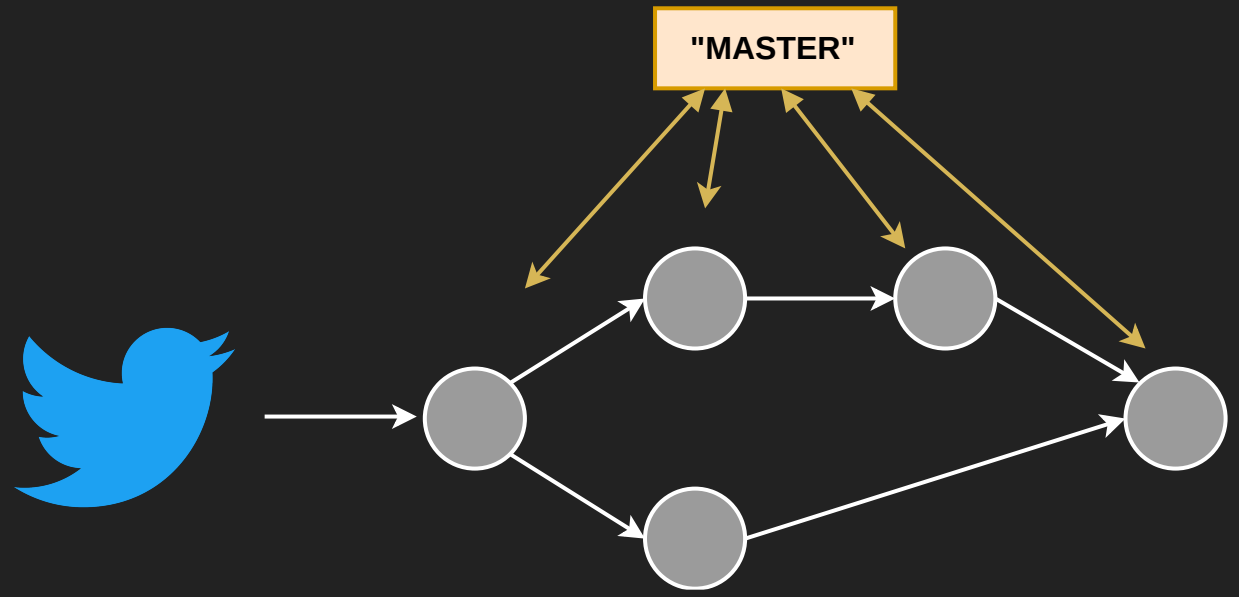
```
class TwitterLanguageCountBolt extends BaseWindowedBolt {
        private OutputCollector collector;
        Map<String, Integer> counts = new HashMap<>();
        ...
        @Override
        public void execute(final TupleWindow tupleWindow) {
                for (final Tuple tuple : tupleWindow.get()) {
                        final String lang = tuple.getStringByField( "lang
                        Integer count = counts.get(lang);
                        if (count == null)
                                count = 0;
                        count++;
                        counts.put(lang, count);
                }

                for (final Map.Entry<String, Integer> langCount : counts
                        collector.emit(new Values(langCount.getKey(), la
        }

        @Override
        public void declareOutputFields(final OutputFieldsDeclarer decla
                declarer.declare(new Fields("lang", "count"));
        }
}
```

# APACHE SPARK STREAMING


- Originated at UC Berkeley
- Written in Scala
- Processes micro-batches
- APIs: Java, Scala, Python, R
- Also provides "SQL on streams"

# HELLO TWITTER IN SPARK

```
JavaReceiverInputDStream twitterStream = TwitterUtils.createStream(sc, t
twitterStream.map(new Function() {
                @Override
                public String call(Status tweet) throws Exception {
                                return tweet.getUser().getLang();
                }
})
        .window(new Duration(10 * 1000))
        .countByValue()
        .print(100);
sc.start();
sc.awaitTermination();
```


"MASTER"

# INGREDIENT #2:
# SOURCES AND SINKS

# SOURCES AND SINKS

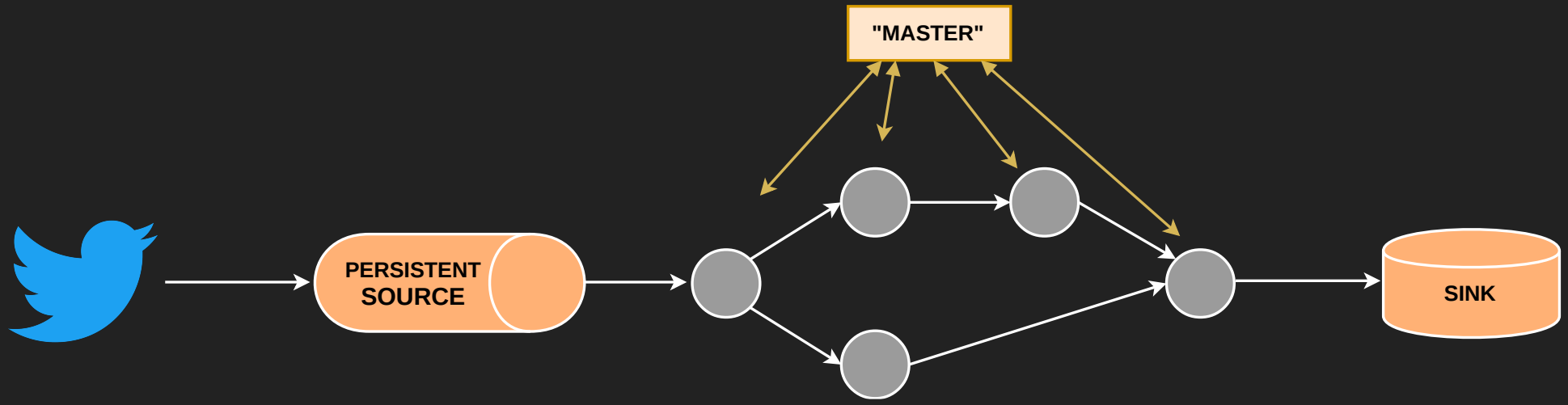
- Kafka
- HDFS
- Databases
- S3
- ...

→ A lot is already provided

→ Building custom sinks is usually easy

# WHAT TO CONSIDER?

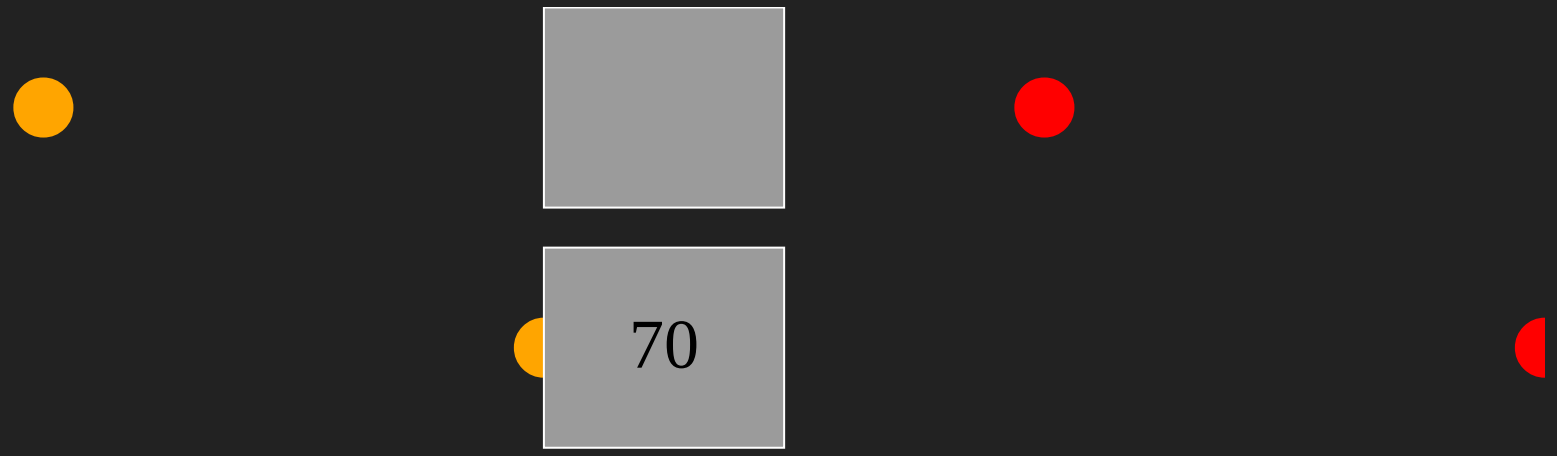
- Fault tolerance, i.e. rewindable sources like Kafka (more later)
- Data quality, i.e. idempotent sinks like databases (more later)
- Decoupling of modules
- Latency, i.e. making data quickly available


"MASTER"
PERSISTENT
SOURCE
SINK

# INGREDIENT #3:
# STATEFUL OPERATIONS

# WHAT IS STATE?

- Stateless Operators: map tweet → language, record encryption, ...
- Stateful Operators: tweet language count, machine learning model, ...



# HOW DO WE GET STATE?

- Could store it in a variable in the operator
- Don't forget: Operators on many machines, but 1 total state
- Central "hub" that stores all the state, e.g. a database
- What happens when the application crashes?

→ Do we really want to manage that ourselves?

# STATE MANAGEMENT: STORM

- Need to specifically add state to bolts
- Store state in KeyValueState
- State backend is Redis
- State buffered in memory, periodically checkpointed to Redis

# HELLO TWITTER IN STORM (REVISED)

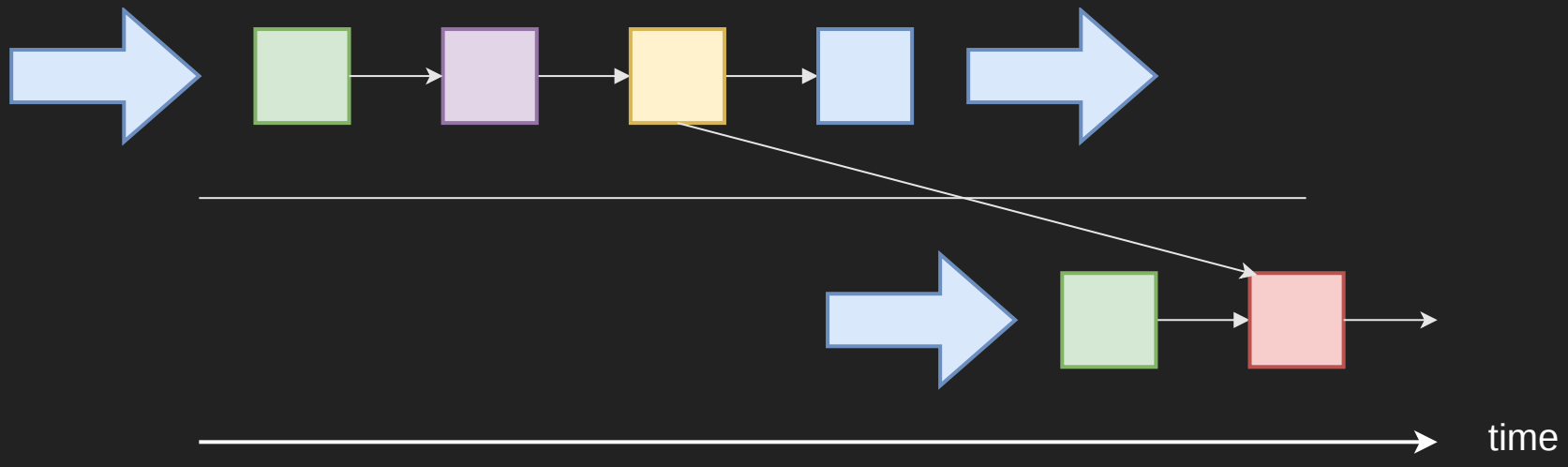
```
class TwitterLanguageCountBolt extends BaseStatefulWindowedBolt<KeyValue
        private OutputCollector collector;
        private KeyValueState<String, Long> counts ;

        @Override
        public void initState(KeyValueState<String, Long> state) {
                wordCounts = state;
        }

        @Override
        public void execute(final TupleWindow tupleWindow) {
                //same as before
        }

        @Override
        public void declareOutputFields(final OutputFieldsDeclarer decla
                //same as before
        }
}
```

# STATE MANAGEMENT: FLINK

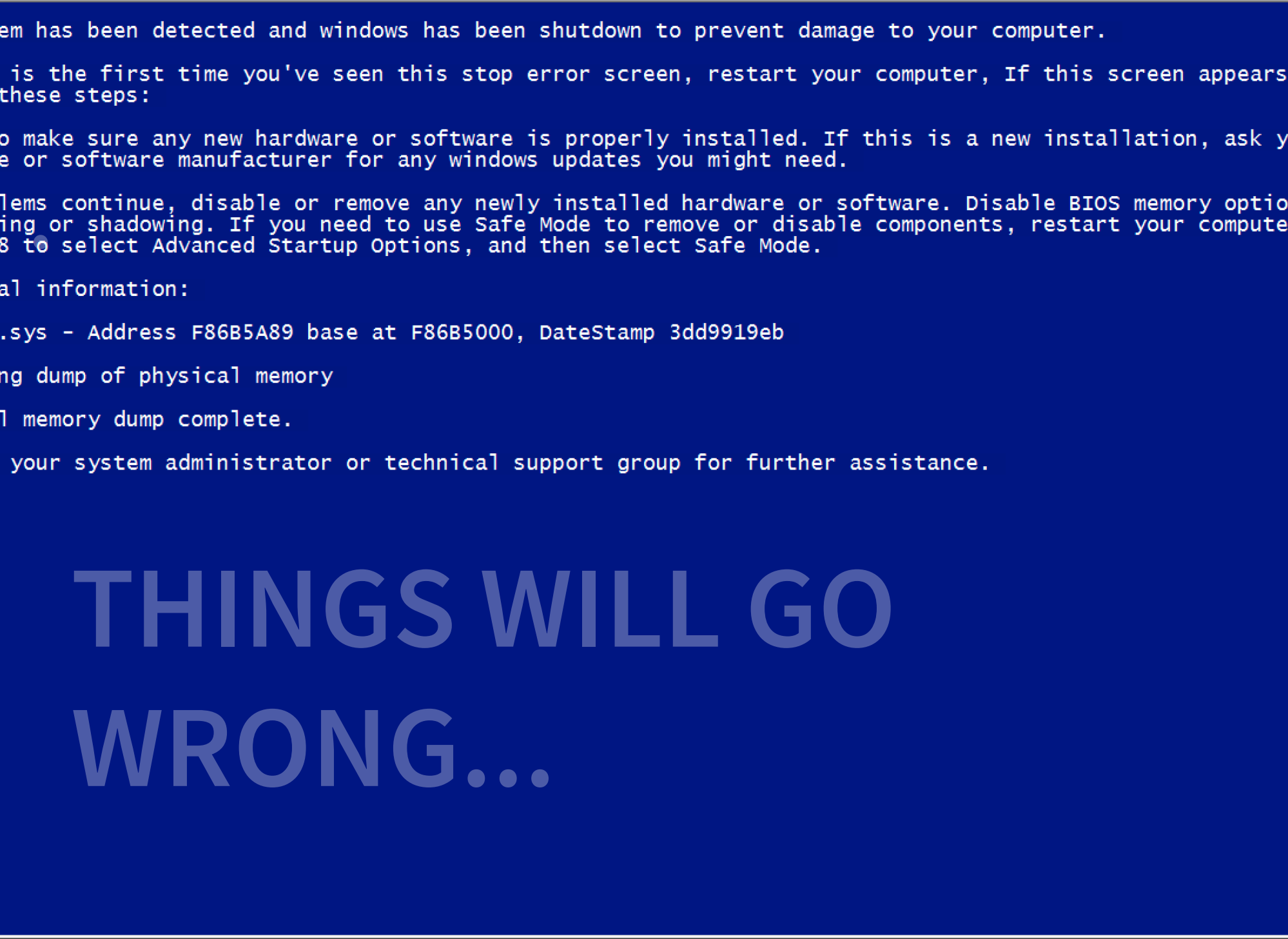
- Provided functions ("window", "sum", ...): out-of-the box
- Own functions: need to care about state yourself
- State is kept in memory (RocksDB allows for "infinite" state)
- Periodically checkpointed to HDFS, JobManager, S3, or RocksDB
- Keyed state vs. operator state

# STATE MANAGEMENT: SPARK

- Every n seconds one program
- Operations made on whole RDD
- Spark saves "lineage" of RDDs
- Too large RDDs can be spilled to disk
- Stateful functions provided: updateStateBykey
- RDDs and lineage periodically checkpointed to state backend (HDFS, S3)




Text


em has been detected and windows has been shutdown to prevent damage to your computer.
is the first time you've seen this stop error screen, restart your computer, If this screen appears
these steps:
o make sure any new hardware or software is properly installed. If this is a new installation, ask y
e or software manufacturer for any windows updates you might need.
lems continue, disable or remove any newly installed hardware or software. Disable BIOS memory optio
ing or shadowing. If you need to use Safe Mode to remove or disable components, restart your compute
8 to select Advanced Startup Options, and then select Safe Mode.
al information:
.sys - Address F86B5A89 base at F86B5000, DateStamp 3dd9919eb
ng dump of physical memory
l memory dump complete.
your system administrator or technical support group for further assistance.
THINGS WILL GO
WRONG...

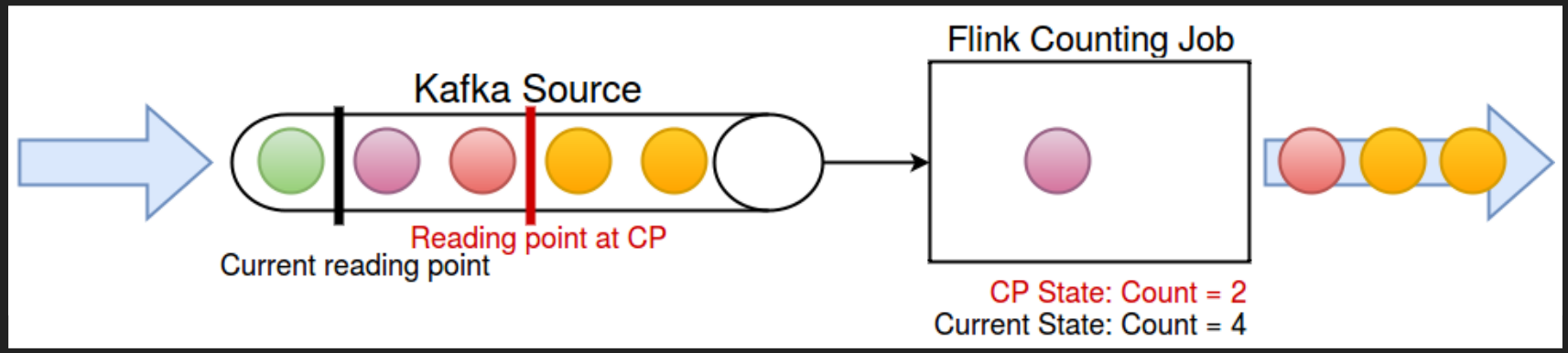
# INGREDIENT #4:
# FAILURE TOLERANCE

# FAILURE TOLERANCE

- State is periodically checkpointed to a persistent backend
- Have a persistent, rewindable source

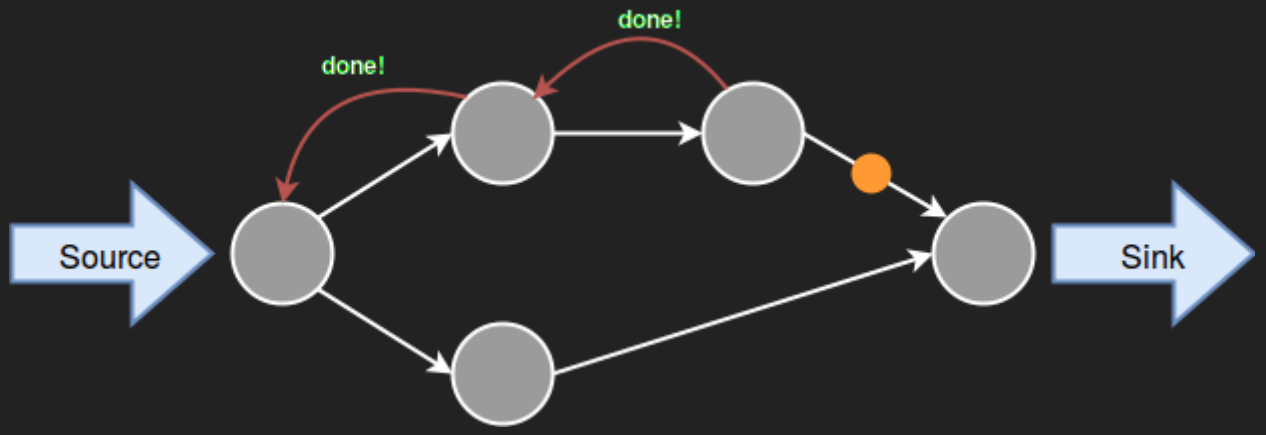
# FAULT-TOLERANCE: FLINK

- After failure:
  - Restart job
  - Read checkpointed state
  - Rewind source (reading offsets are part of the state)
  - Continue processing



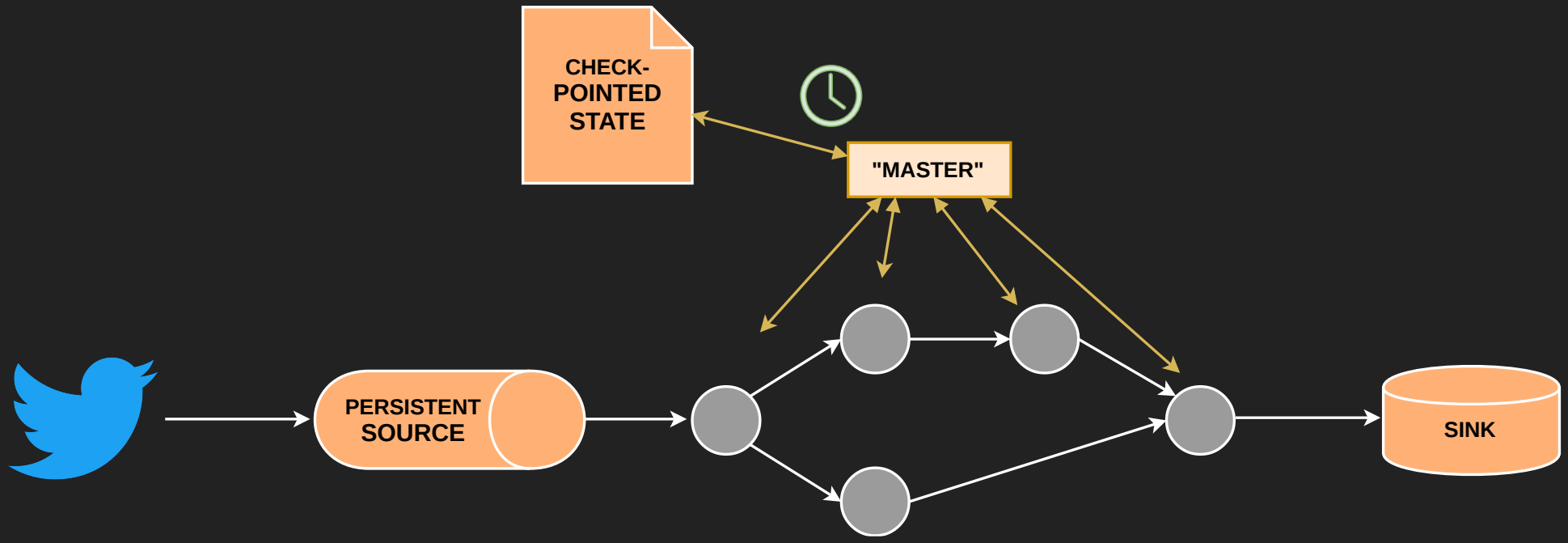
# FAULT-TOLERANCE: STORM

- After Failure:
  - Restart worker
  - Read checkpointed state
  - First node saves backup of records not yet "acked"
  - (some bolt "ack" automatically, others don't → watch out)
  - Records not fully acked after some time will be resent



# FAULT-TOLERANCE: SPARK

- After failure of a batch:
  - Read checkpointed lineage (original batch + transformations)
  - Recompute failed batch according to lineage
  - Spark recomputes the failed batch quickly → keep up


CHECK-
POINTED
STATE
"MASTER"
PERSISTENT
SOURCE
SINK

# SINGLE POINTS OF FAILURE

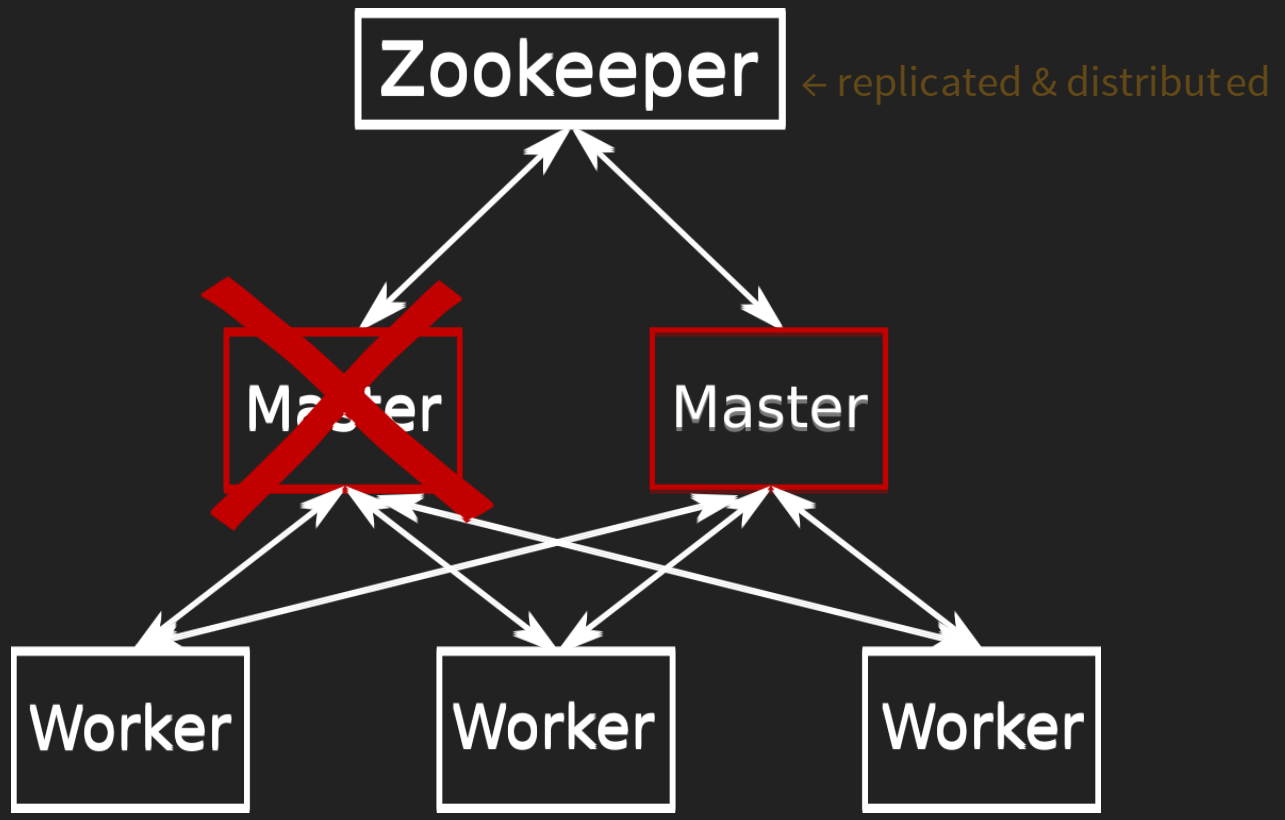
© Wookiepedia
(starwars.wikia.com)

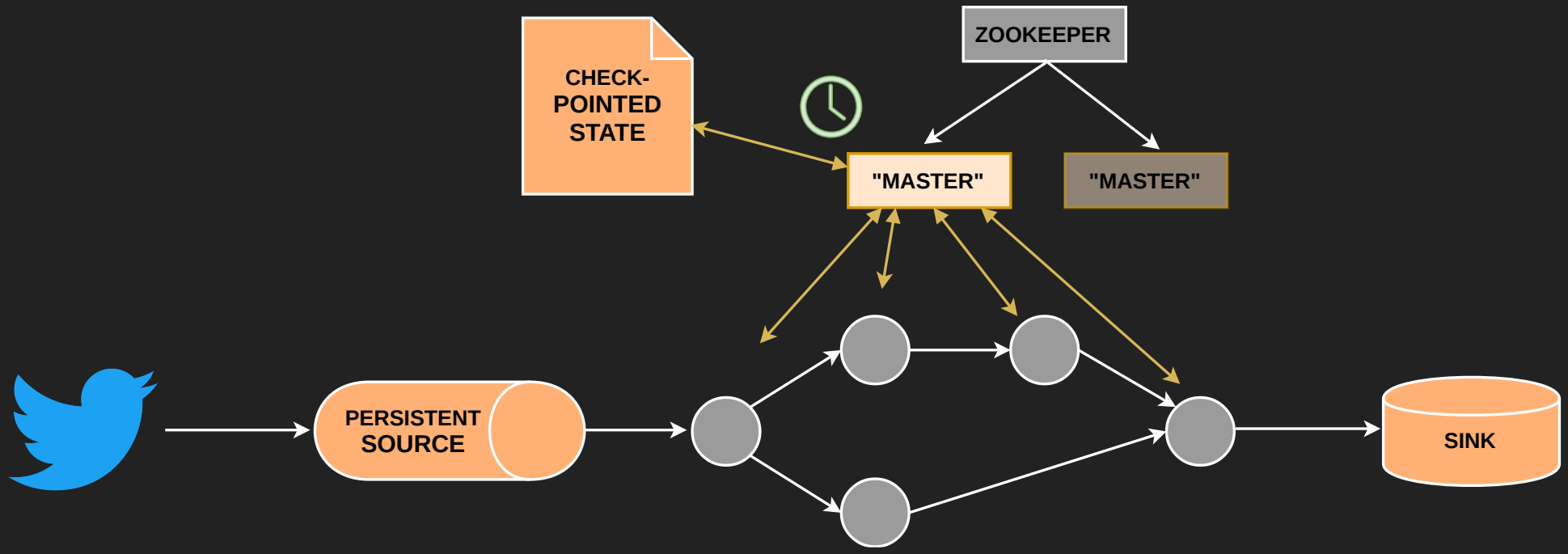
"The application is unbreakable! Really
Well, except for this one little thing, bu
that's nothing..."

# INGREDIENT #5:
# HIGH AVAILABILITY

# HIGH AVAILABILITY

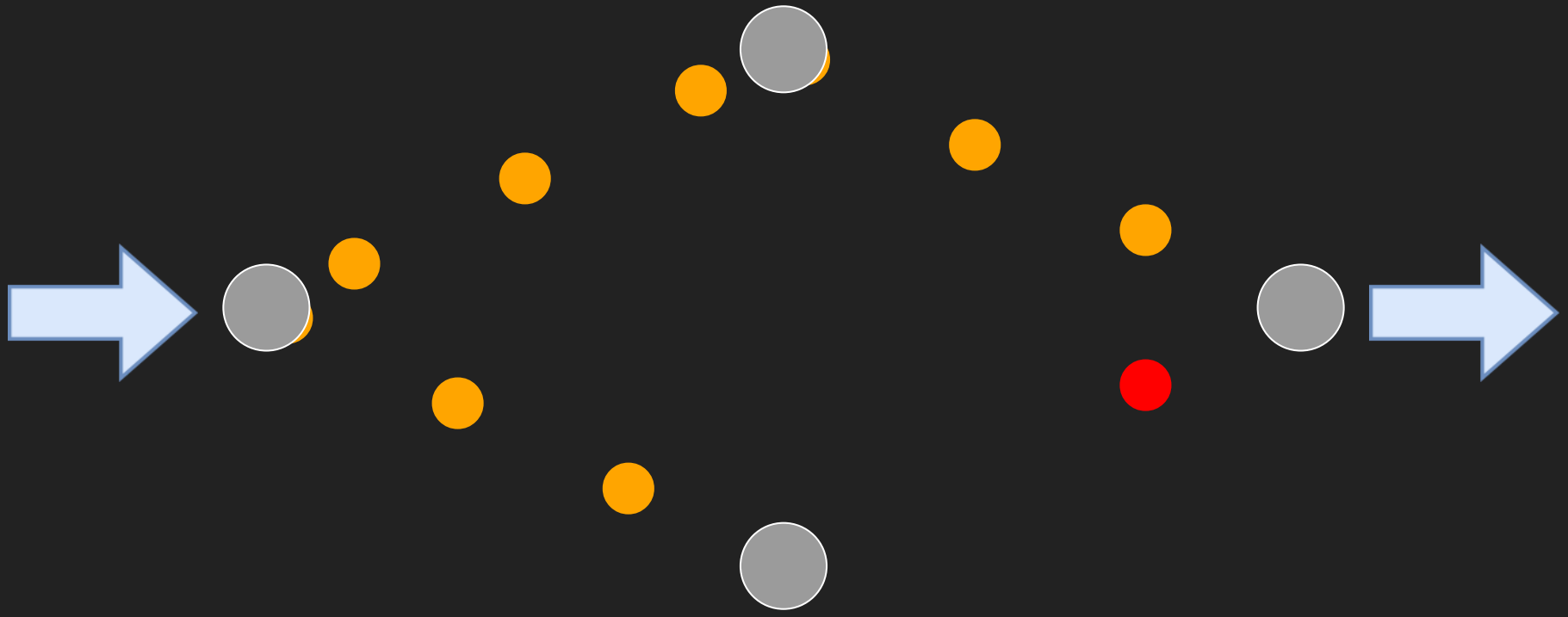
- You don't want single points of failures
- "Masters" manage the program, they are SPOFs
- Job Manager (Flink), Nimbus (Storm), Master (Spark)
  → Use Zookeeper framework




CHECK-
POINTED
STATE
ZOOKEEPER
"MASTER"
"MASTER"
PERSISTENT
SOURCE
SINK

# INGREDIENT #6:
# FLOW CONTROL

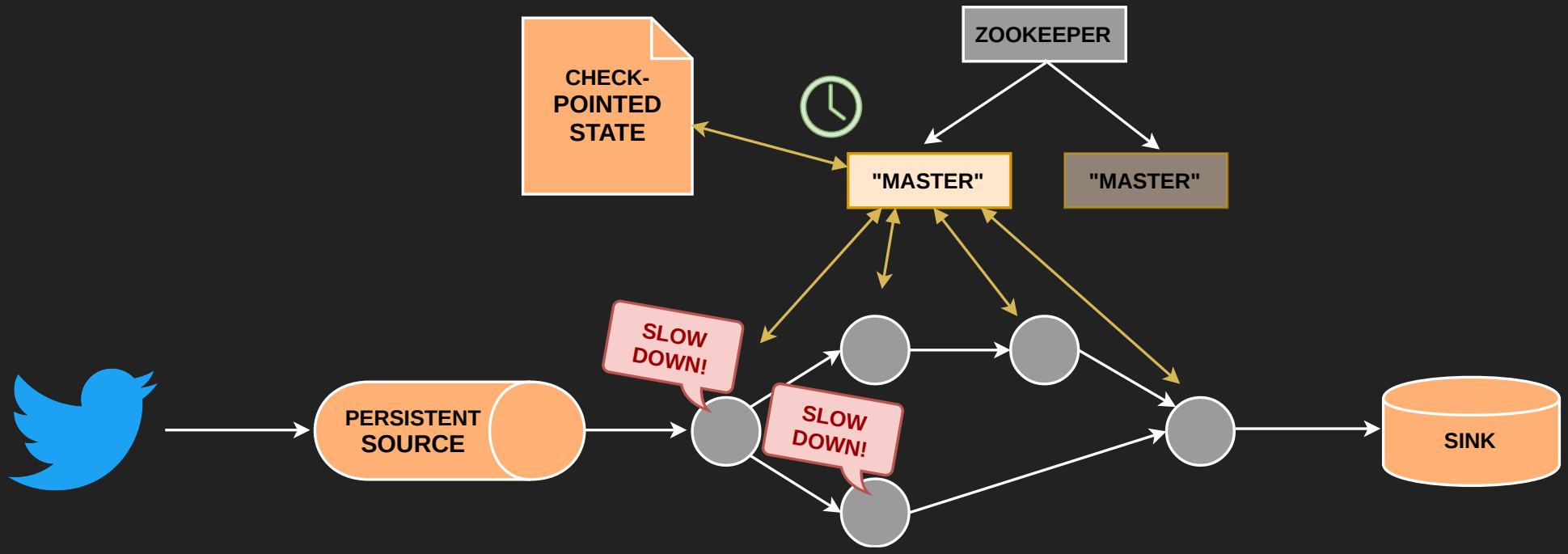
# FLOW CONTROL



# BACK PRESSURE

- Only let as many records enter as the topology can handle
- Needs persistent source
- Provided by Storm, Flink, and Spark
- Storm & Flink: Buffers of limited size between operators
- Spark: Calculate processing rate after each batch → rate limit

- Note: Back pressure between applications is very hard!


CHECK-
POINTED
STATE
ZOOKEEPER
"MASTER"
"MASTER"
SLOW
DOWN!
SLOW
DOWN!
PERSISTENT
SOURCE
SINK

# INGREDIENT #7:
# DELIVERY GUARANTEES

# DATA QUALITY

- Delivery guarantees

  At most once  At least once  Exactly once
- Past: Lambda-Architecture: Streaming and Batch

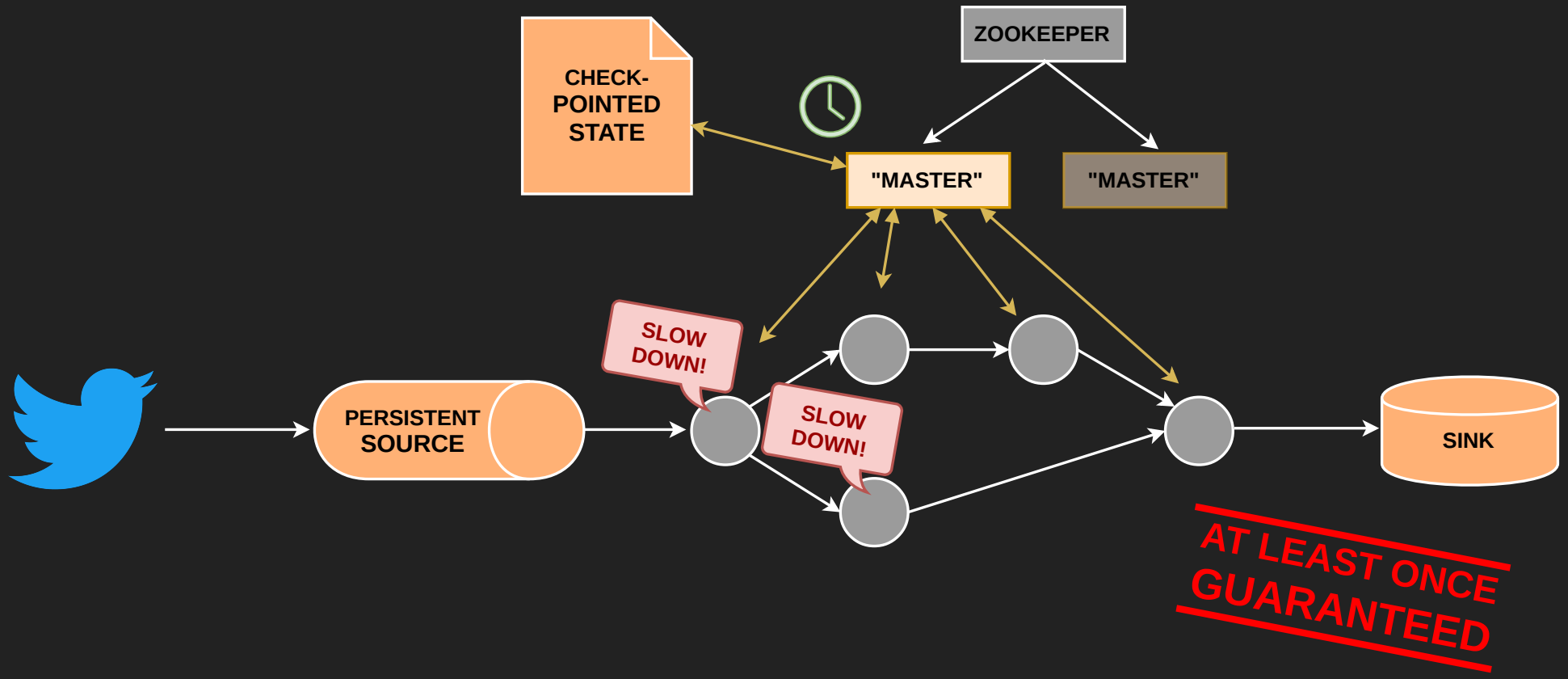
Assuming Kafka source: *

- Spark Streaming: Micro-batching → exactly once
- Flink: Checkpoints → exactly once
- Storm: Acking → at least once

*Note: other guarantees on other sources (complicated!)

# SOME NOTES ON "EXACTLY ONCE"

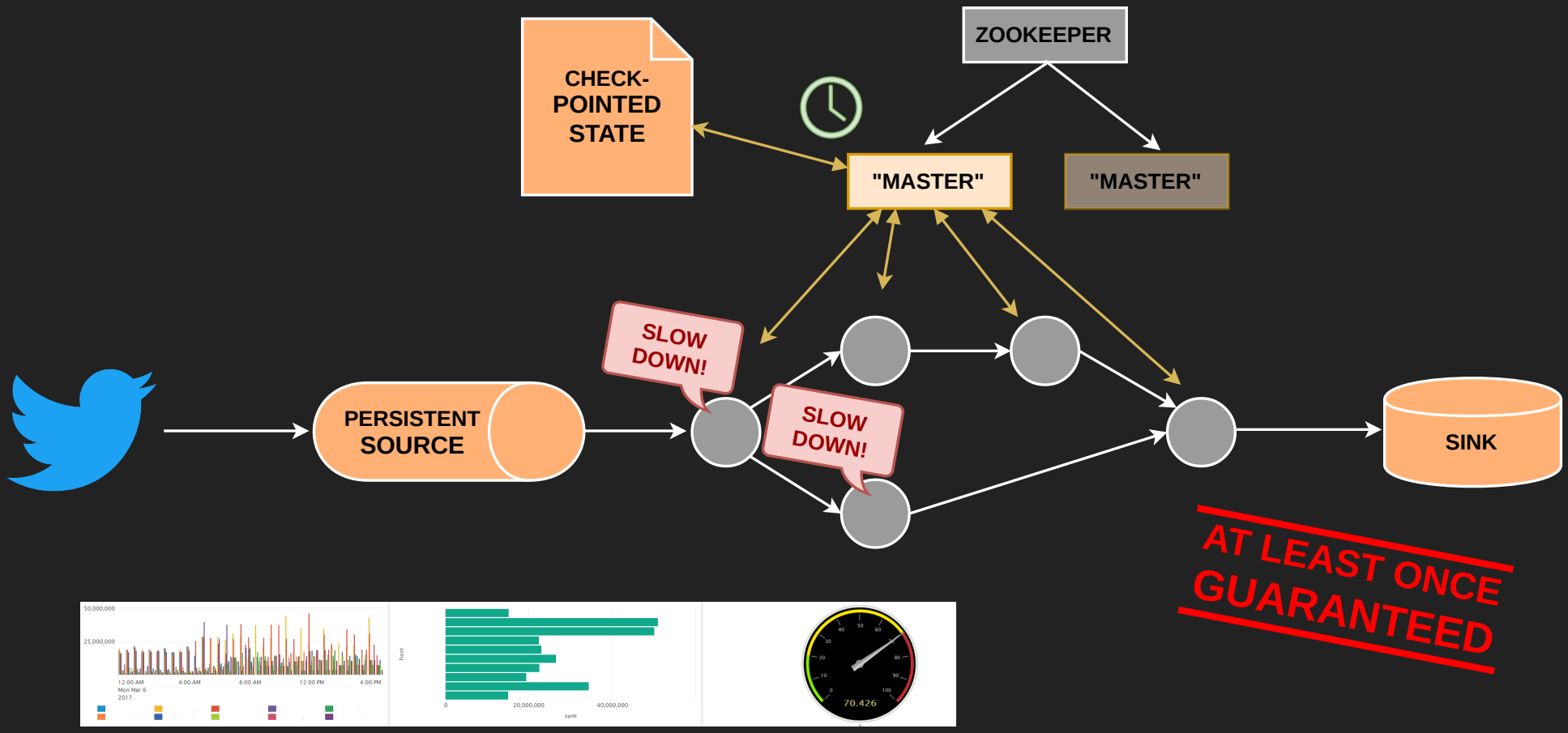
- Exactly once *state* or *delivery*?!
- "Exactly once state" possible with Flink and Spark
- "Exactly once delivery" depends on the architecture
- However, "exactly once delivery" can easily be achieved with idempotent sinks
- HDFS has "truncate" method → exactly once delivery
- Spark offers transactional updates → exactly once delivery


CHECK-
POINTED
STATE
ZOOKEEPER
"MASTER"
"MASTER"
SLOW
DOWN!
SLOW
DOWN!
PERSISTENT
SOURCE
SINK
AT LEAST ONCE
GUARANTEED

# INGREDIENT #8:
# MONITORING

# MONITORING

- Don't underestimate what's necessary
- You need proper metrics to analyze your application
- Business metrics: might not be a by-product
- Metrics and failure tolerance?


CHECK-
POINTED
STATE
ZOOKEEPER
"MASTER"
"MASTER"
SLOW
DOWN!
SLOW
DOWN!
PERSISTENT
SOURCE
SINK
AT LEAST ONCE
GUARANTEED


50,000,000
25,000,000
12:00 AM
Mon Mar 6
2017
4:00 AM
8:00 AM
12:00 PM
4:00 PM
host
0
20,000,000
40,000,000
sum
70.426


ADD SOME SALT

# PRODUCTION CHECKS

- Regularly, inject some dummy records into your flow
- Verify they come out of the pipeline
- Measure their latency

# TL;DL

- **Stream Processor:** Thoroughly analyze what you need
- **State Management:** Make your job properly store its state
- **Checkpointing:** No data loss on failures
- **High Availability:** No single points of failure! → Zookeeper
- **Back Pressure:** Don't overload the system
- **Data Quality:** Know your "exactly once"
- **Monitoring:** Make sure you always know what's happening
- **Production Tests:** Test if prod system is working as expected

# THANK YOU!

## QUESTIONS?

www.tngtech.com / konstantin.gregor@tngtech.com