FAST DATA SMACK DOWN

Data Done Right

Created by Stefan Siprell / @stefansiprell

SMACK?

- Spark
- Mesos
- Akka
- Cassandra
- Kafka

Spark

Swiss Army Knife for Data

- ETL Jobs? No problem.
- µ-Batching on Streams? No problem.
- SQL and Joins on non-RDBMS? No problem.
- Graph Operations on non-Graphs? No problem.
- Map/Reduce in super fast? Thanks.

Mesos

Distributed Kernel for the Cloud

- Links machines to a logical instance
- Static deployment of Mesos
- Dynamic deployment of the workload
- Good integration with Hadoop, Kafka, Spark, and Akka
- Lots and lots of machines - data-centers

Akka

Framework for reactive applications

- Highly performant
- Simple concurrency via asynchronous processing
- elastic and without single point of failure
- resilient
- Lots and lots of performance - 50 million messages per machine and second

Cassandra

Performant and Always-Up No-SQL Database

- Linear scaling - approx. 10'000 requests per machine and second
- No Downtime
- Comfort of a column index with append-only performance
- Data-Safety over multiple data-centers
- Strong in denormalized models

Kafka

Messaging for Big Data

- Fast - Delivers hundreds of MegaBytes per second to 1000s of clients
- Scales - Partitions data to *manageable* volumes
- Data-Safety - Append Only allows buffering of TBs without a performance impact
- Distributed - from the ground up

In the beginning there was

HADOOP

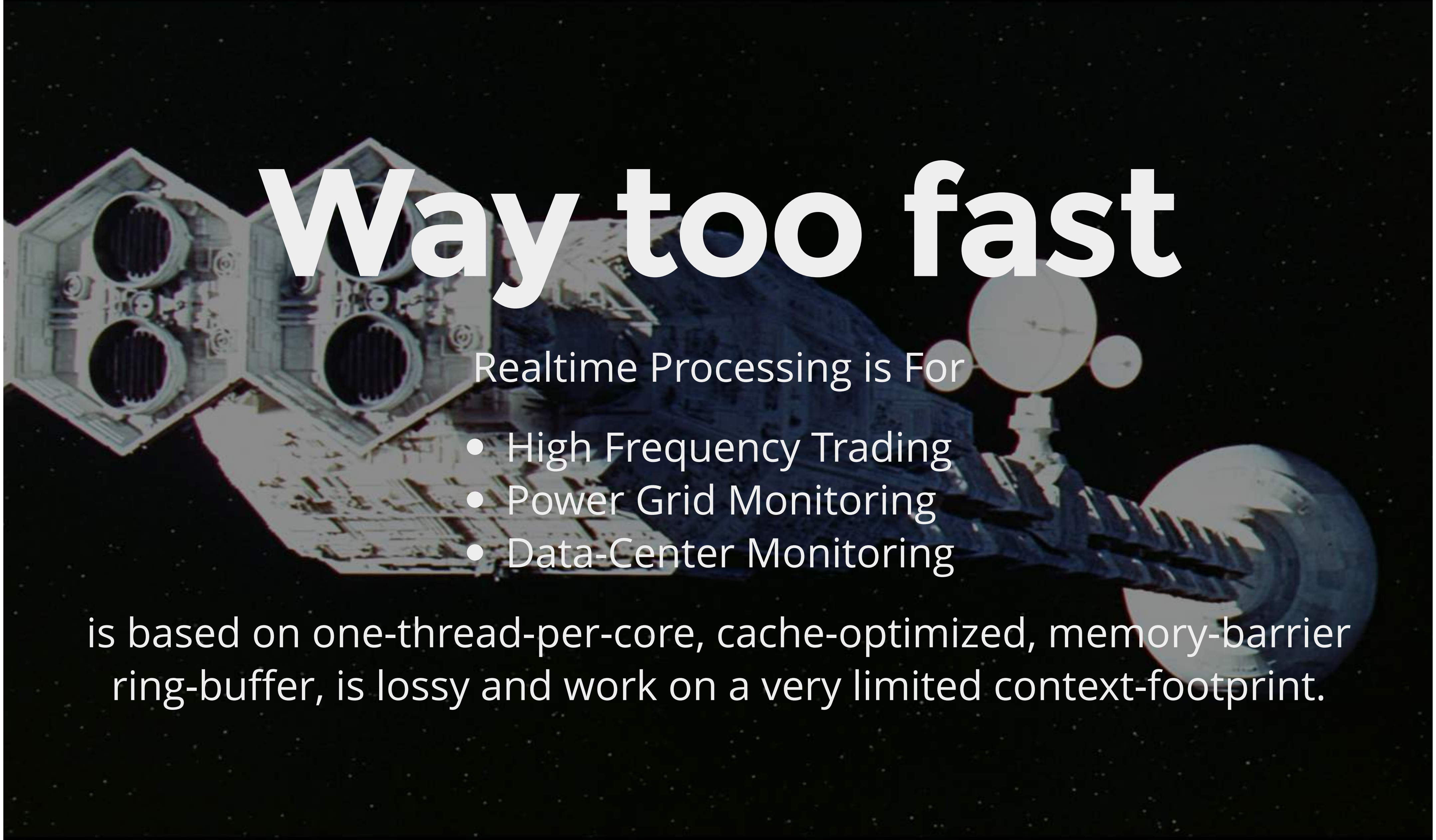
Big-Batch saw that this was good

und has mapped and reduced ever since

But Business does not wait.

It always demands more...

ever faster



Way too fast

Realtime Processing is For

- High Frequency Trading
- Power Grid Monitoring
- Data-Center Monitoring

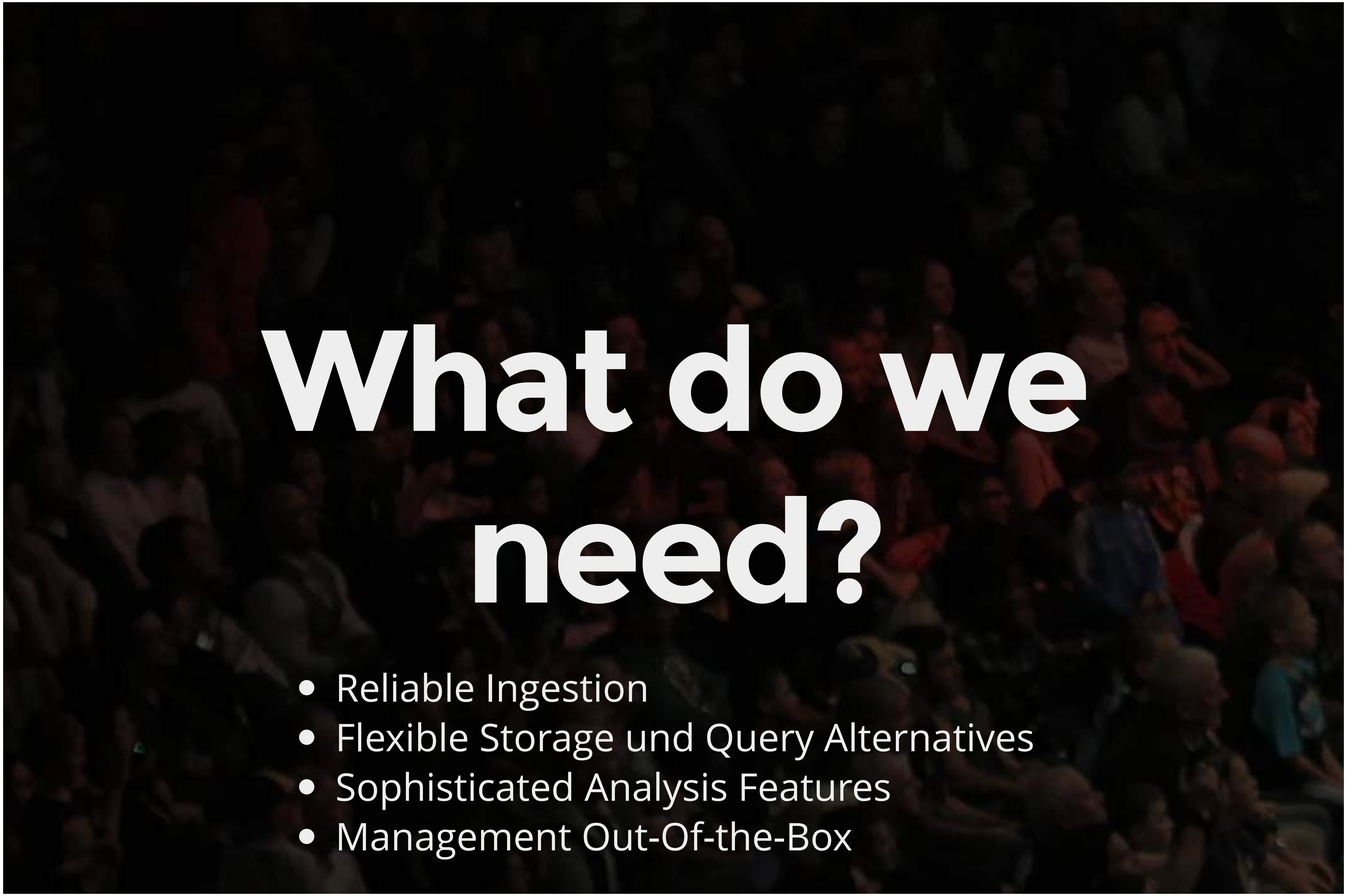
is based on one-thread-per-core, cache-optimized, memory-barrier ring-buffer, is lossy and work on a very limited context-footprint.

Between Batch and Realtime...

...lies a new Sweet Spot

SMACK

- Updating News Pages
- User Classification
- (Business) Realtime Bidding for Advertising
- Automative IoT
- Industry 4.0



What do we need?

- Reliable Ingestion
- Flexible Storage und Query Alternatives
- Sophisticated Analysis Features
- Management Out-Of-the-Box

What is SMACK - Part II

- Architecture Toolbox
- Best-of-Breed Platform
- *Also a Product*

μ-Batching

When do I need this?

I don't want to react on individual events.

I want to establish a context.

- Abuse Classification
- User Classification
- Site Classification

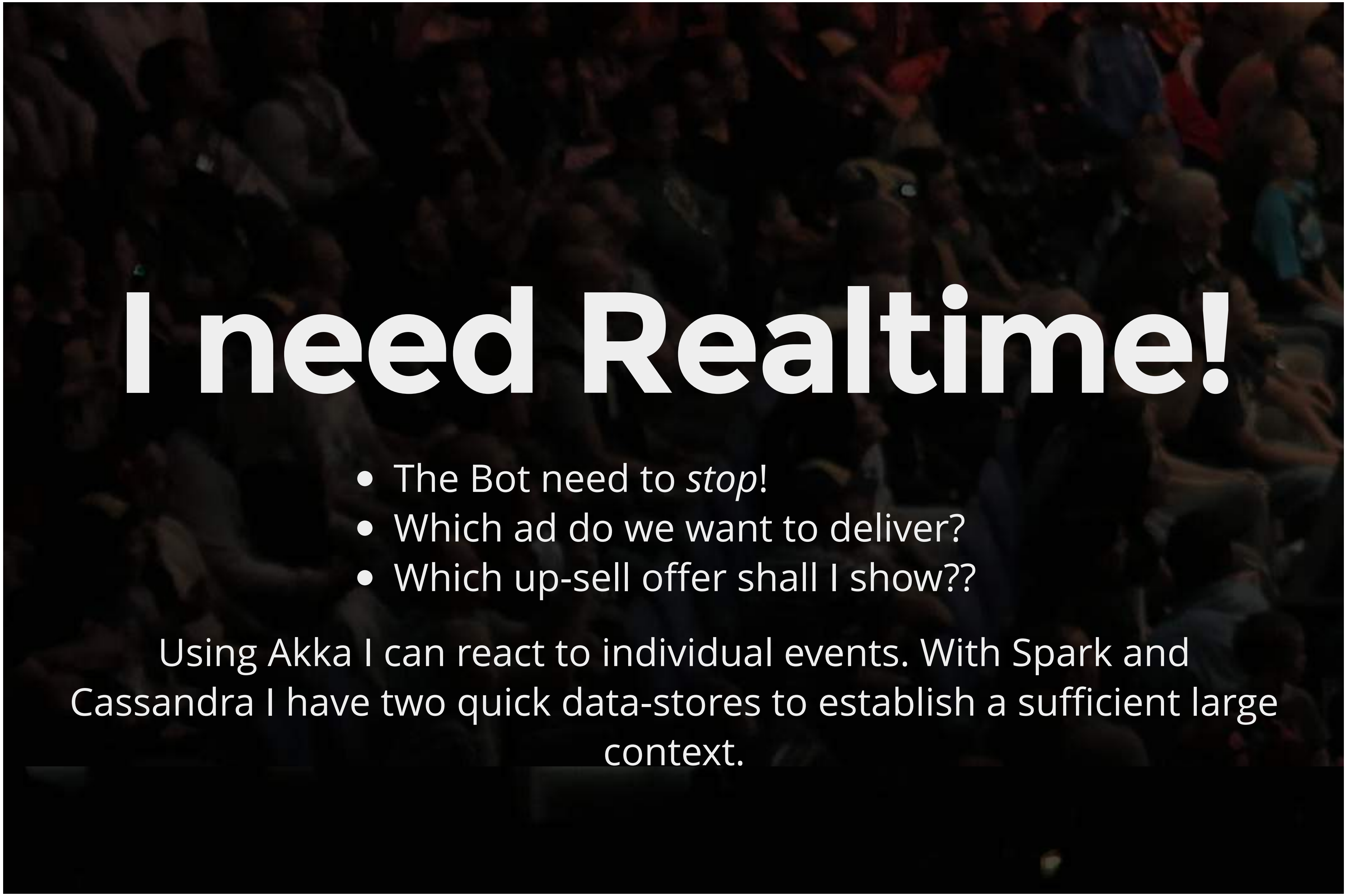
How does it work?

Spark Streaming collects events und generates RDDs out of windows.

- Uses Kafka, Databases, Akka or streams as input
- Windows can be flushed to persistent storage
- Windows can be queried / modified per SQL In-Memory/Disk
- Cascades of aggregations / classifications can be assembled / flushed

What about λ-Architectures?

Spark Operations cab be run unaltered in either batch or stream mode - it is always an RDD!



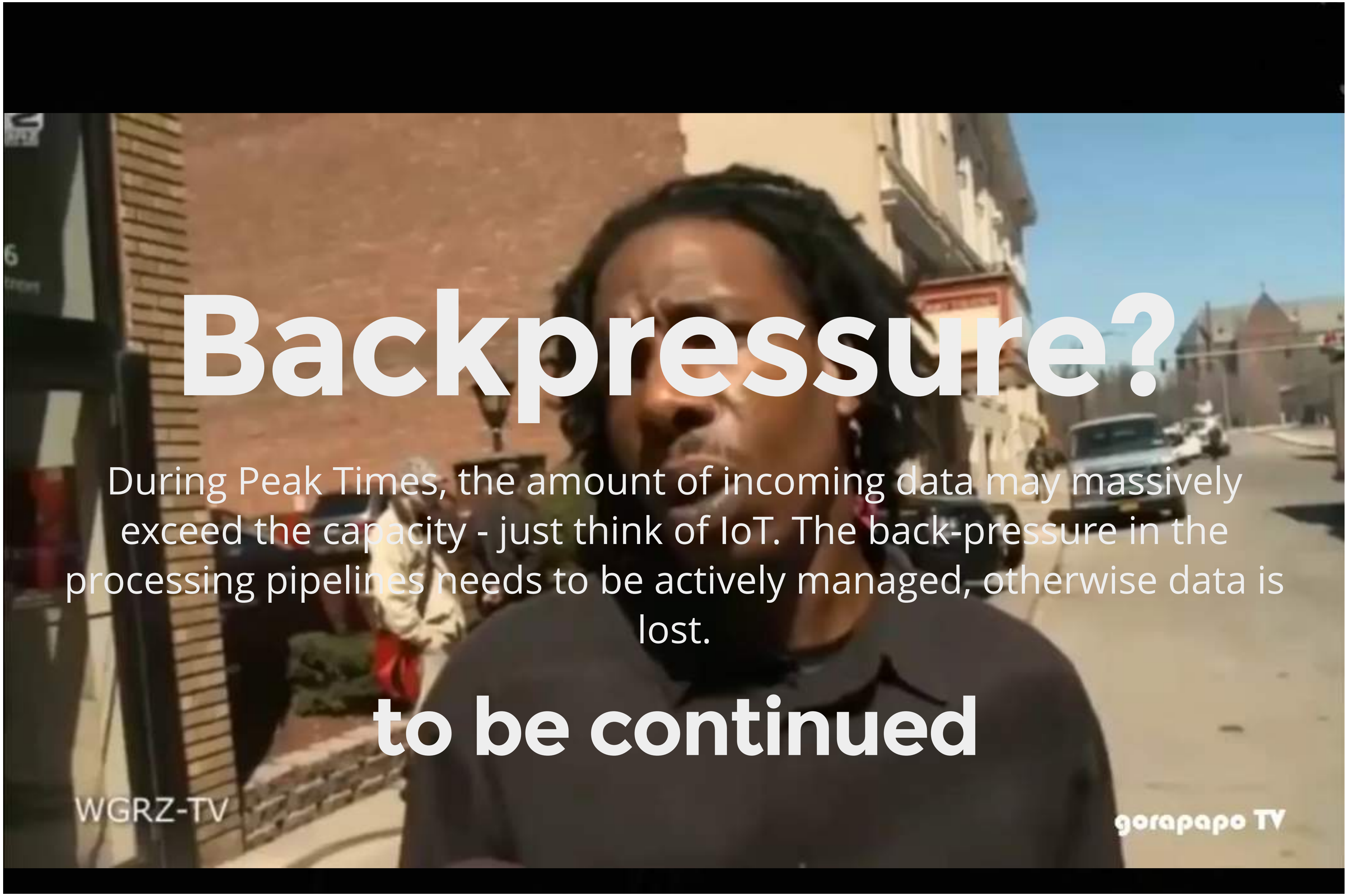
I need Realtime!

- The Bot need to *stop*!
- Which ad do we want to deliver?
- Which up-sell offer shall I show??

Using Akka I can react to individual events. With Spark and Cassandra I have two quick data-stores to establish a sufficient large context.

3 Streams of Happiness

- *Direct streams* between Kafka and Spark.
- *Raw streams* for TCP, UDP connections, Files, etc.
- *Reactive streams* for *back-pressure support*.
- Kafka can translate between *raw* und *reactive* streams.



Backpressure?

During Peak Times, the amount of incoming data may massively exceed the capacity - just think of IoT. The back-pressure in the processing pipelines needs to be actively managed, otherwise data is lost.

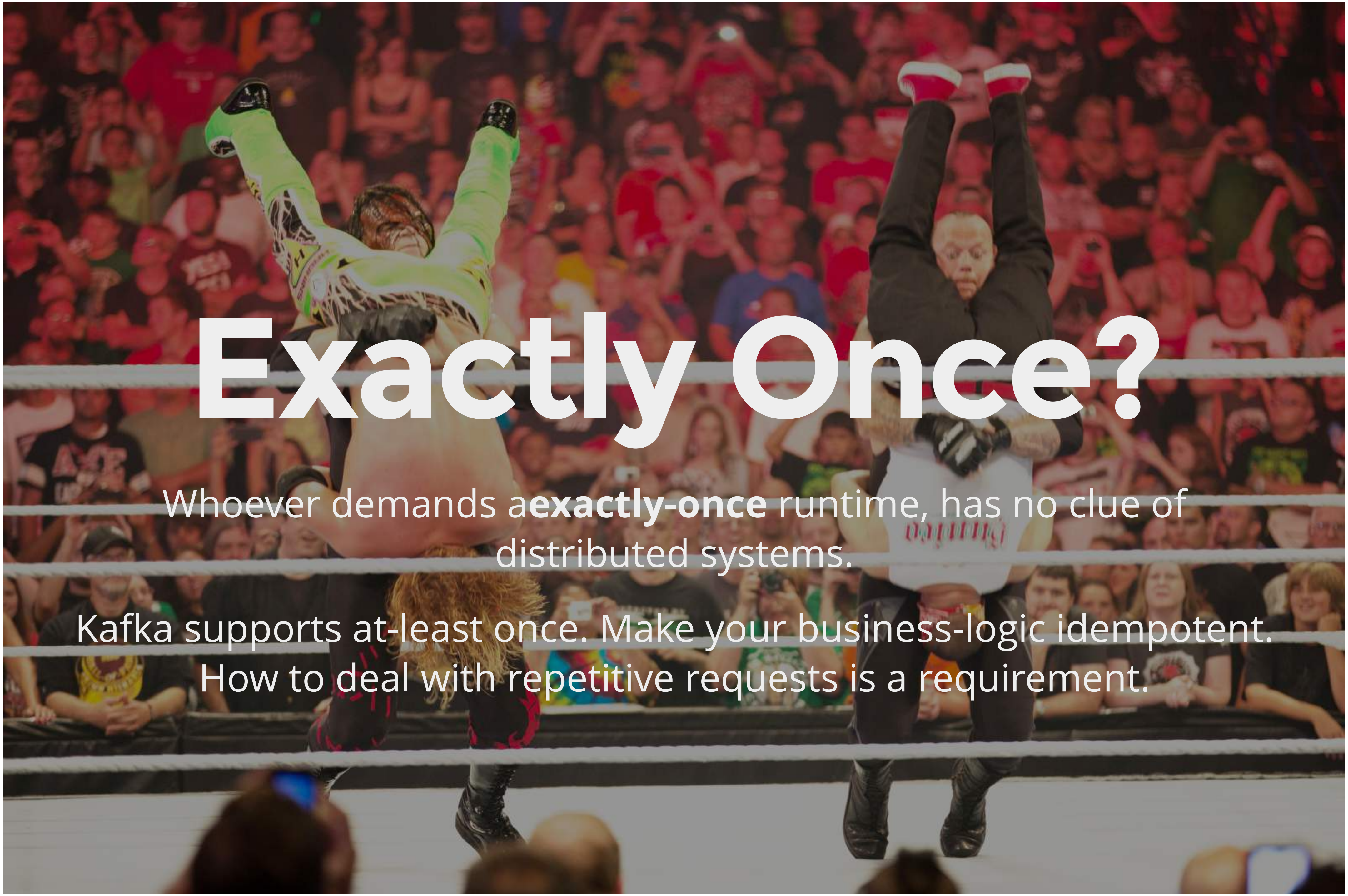
to be continued

Flow

If a an event needs specific handling - *a reaction* - it needs to be dealt with in Akka.

Why Kafka?

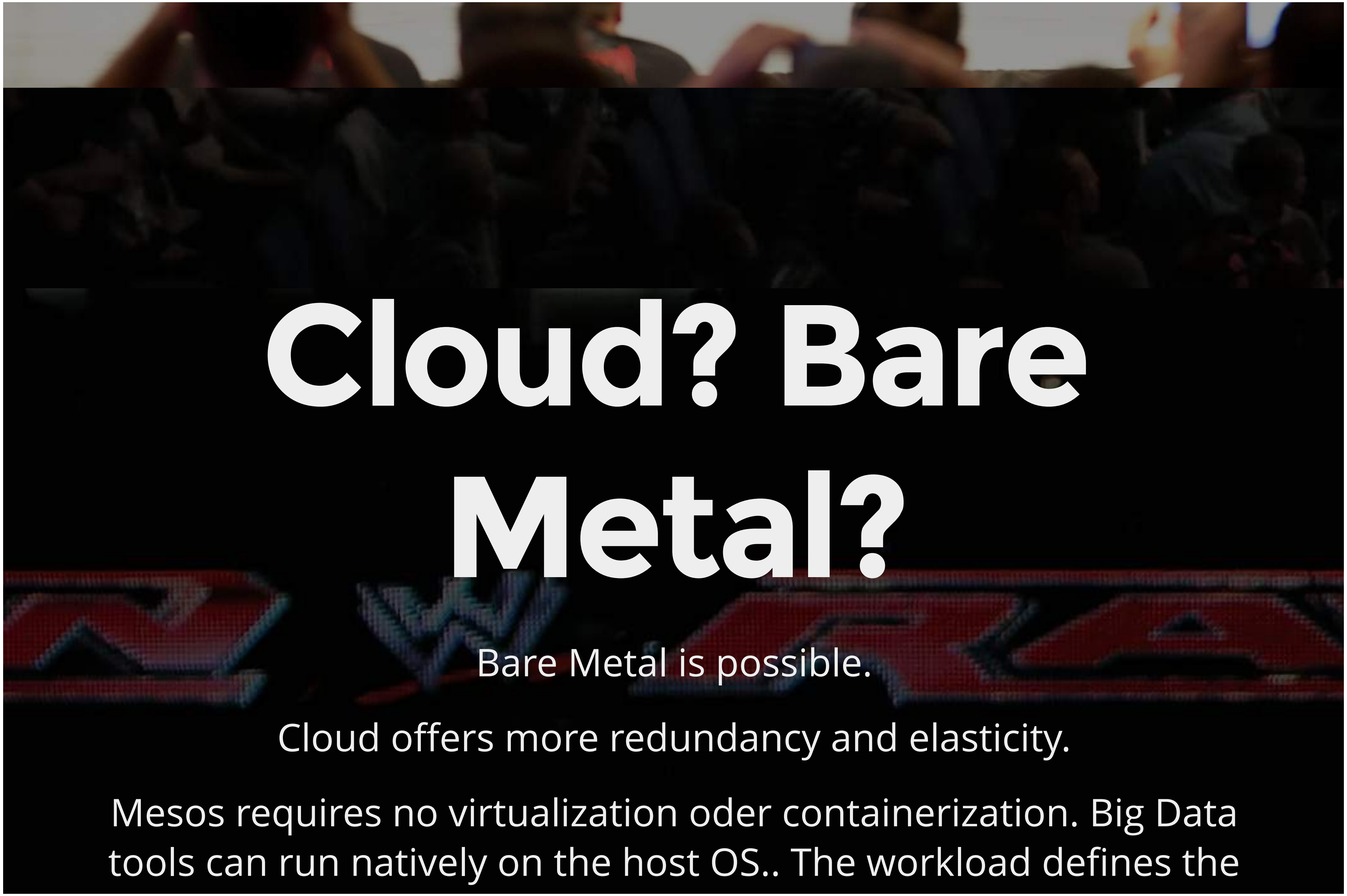
- *Append Only:*Consumer may be offline for days.
- *Broker:*Reduction of point-to-point connections.
- *Router:*Routing of Streams including (de-)multiplexing.
- *Buffer:*Compensate short overloads.
- *Replay:*Broken Algorithm on incorrect data? Redeploy and Replay!



Exactly Once?

Whoever demands a**exactly-once** runtime, has no clue of distributed systems.

Kafka supports at-least once. Make your business-logic idempotent. How to deal with repetitive requests is a requirement.



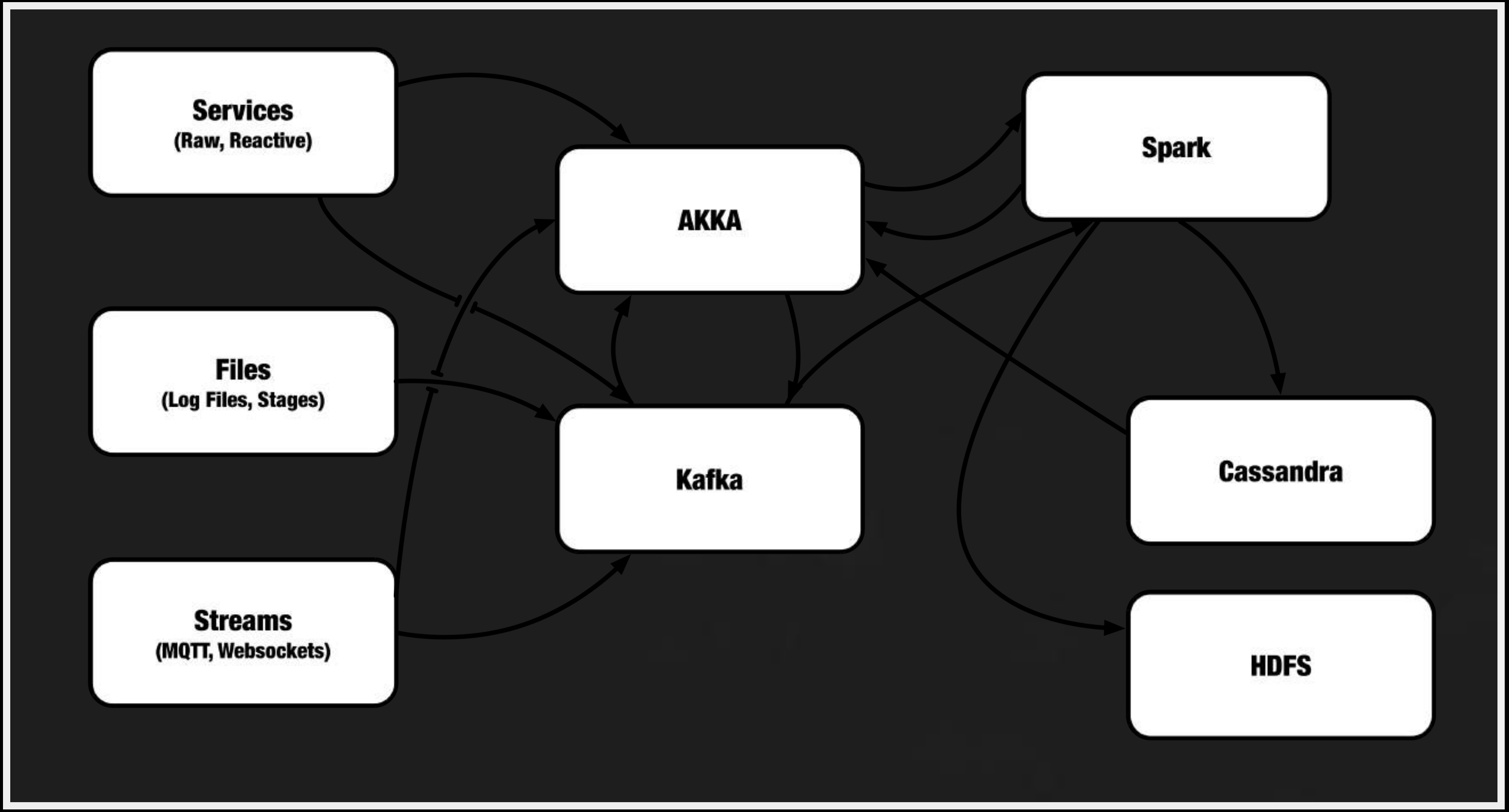
Cloud? Bare Metal?

Bare Metal is possible.

Cloud offers more redundancy and elasticity.

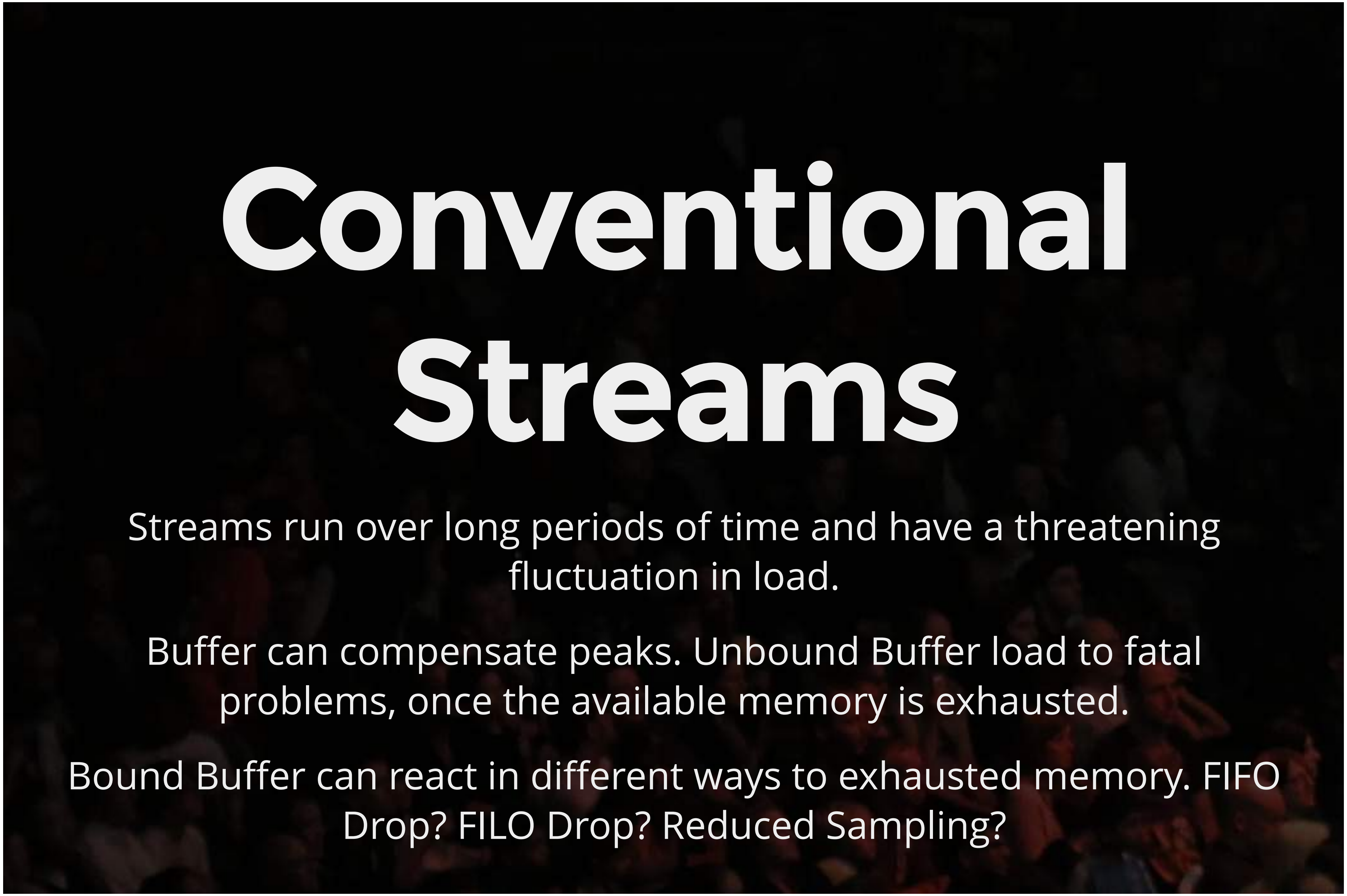
Mesos requires no virtualization oder containerization. Big Data tools can run natively on the host OS.. The workload defines the

cluster setup.



Streams

... are an unbound and continuous sequence of events. Throughput and end are not defined.

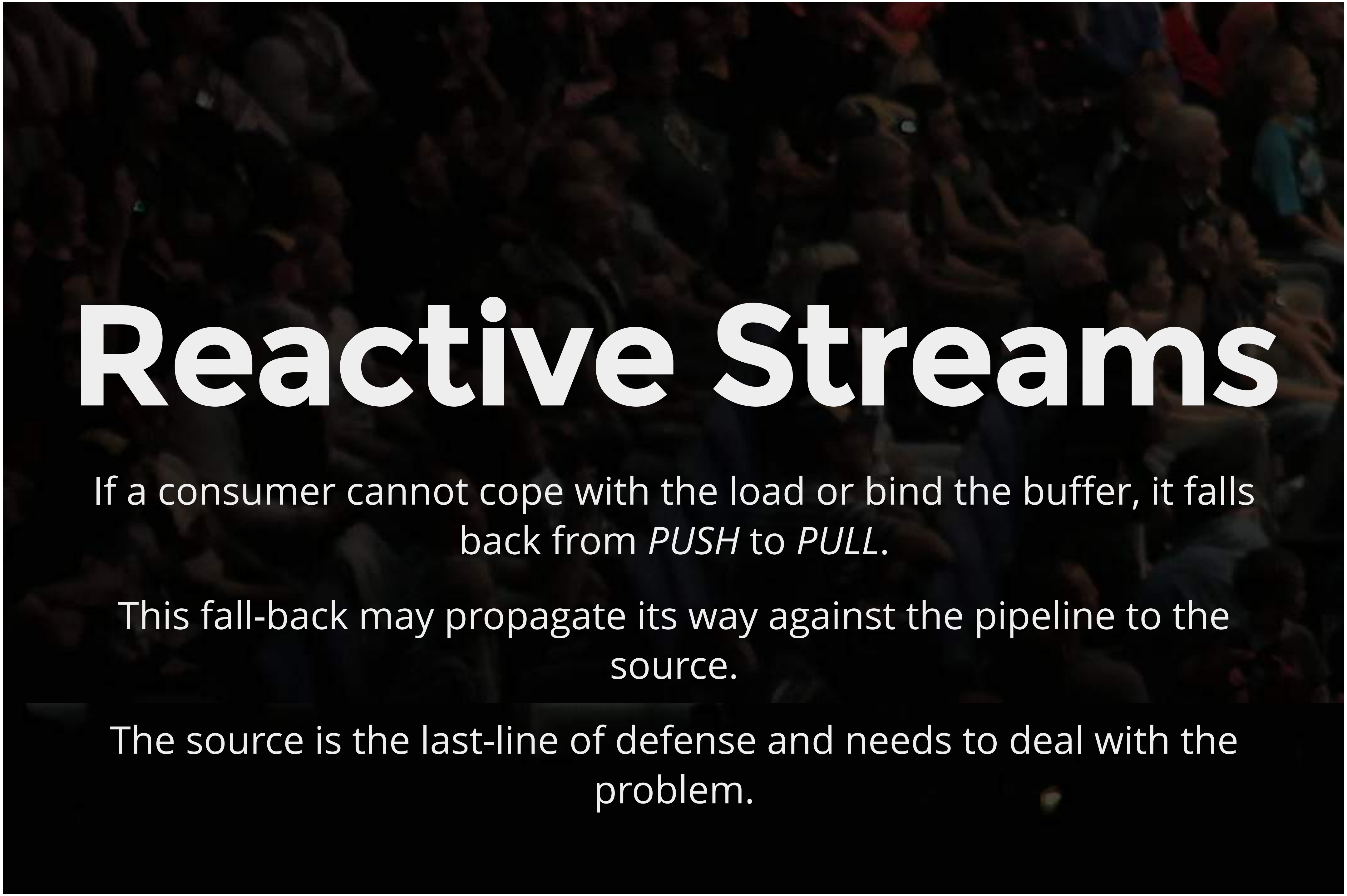


Conventional Streams

Streams run over long periods of time and have a threatening fluctuation in load.

Buffer can compensate peaks. Unbound Buffer load to fatal problems, once the available memory is exhausted.

Bound Buffer can react in different ways to exhausted memory. FIFO Drop? FILO Drop? Reduced Sampling?



Reactive Streams

If a consumer cannot cope with the load or bind the buffer, it falls back from *PUSH* to *PULL*.

This fall-back may propagate its way against the pipeline to the source.

The source is the last-line of defense and needs to deal with the problem.

SMACK Reactive Streams

Akka implements Reactive Streams.

Spark Streaming 1.5 implements Reactive Streams.

Spark Streaming 1.6 allows it's clients to use Reactive Stream as a Protocol.

Kafka is a perfect candidate of a bound buffer for streams - the last line of defense.

Mesos can scale up consumers on-the-fly during the fall-back.

Functional?

Streams love to be processed in parallel. Streams love Scala!

Events therefore need to be immutable - nobody likes production-only concurrency issues.

Functions do not have side-effects - do not track state between function calls!

Functions need to be 1st class citizens - maximize reuse of code.

Reuse?

```
sparkContext.textFile("/path/to/input")
.map { line =>
  val array = line.split(", ", 2)
  (array(0), array(1))
}.flatMap {
  case (id,contents) => toWords(contents).map(w => ((w, id), 1))
}.reduceByKey {
  (count1, count2) => count1 + count2
}.map {
  case ((word, path), n) => (word, (path, n))
}.groupByKey
.map {
  case(word, list) => (word, sortByCount(list))
}.saveAsTextFile("/path/to/output")
```

شكرا

Thank You