

---




# Fast Cars, Big Data

## How Streaming Can Help Formula 1

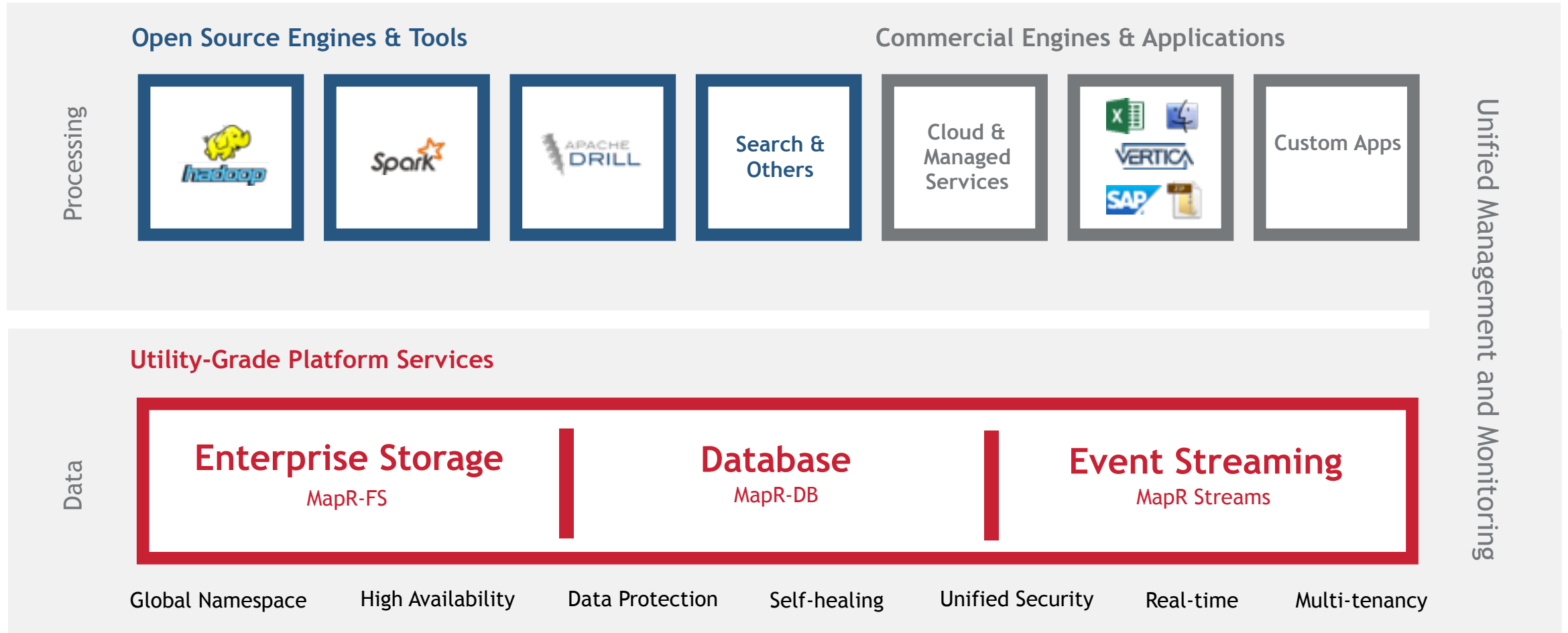
Tugdual Grall  
@tgrall

# { "about" : "me" }

## Tugdual "Tug" Grall

- MapR
    - Technical Evangelist
  - MongoDB
    - Technical Evangelist
  - Couchbase
    - Technical Evangelist
  - eXo
    - CTO
  - Oracle
    - Developer/Product Manager
    - Mainly Java/SOA
  - Developer in consulting firms
- Web
    -  @tgrall
    -  <http://tgrall.github.io>
    -  tgrall
  - NantesJUG co-founder
  - Pet Project :
    - <http://www.resultri.com>
  - [tug@mapr.com](mailto:tug@mapr.com)
  - [tugdual@gmail.com](mailto:tugdual@gmail.com)

# MapR Converged Data Platform



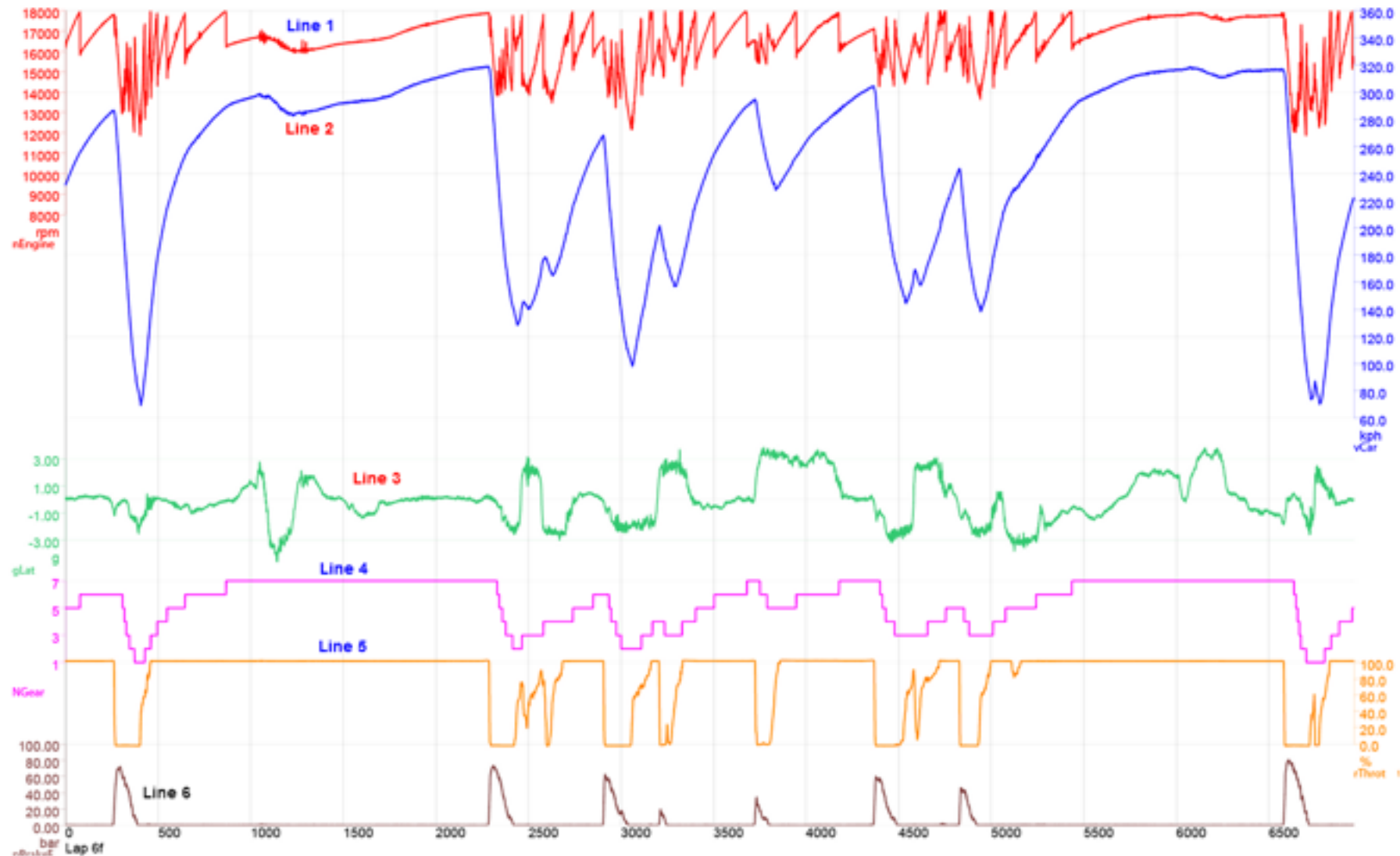
# Agenda

- What's the point of data in motorsports?
- Live demo
- Architecture
- What's next?

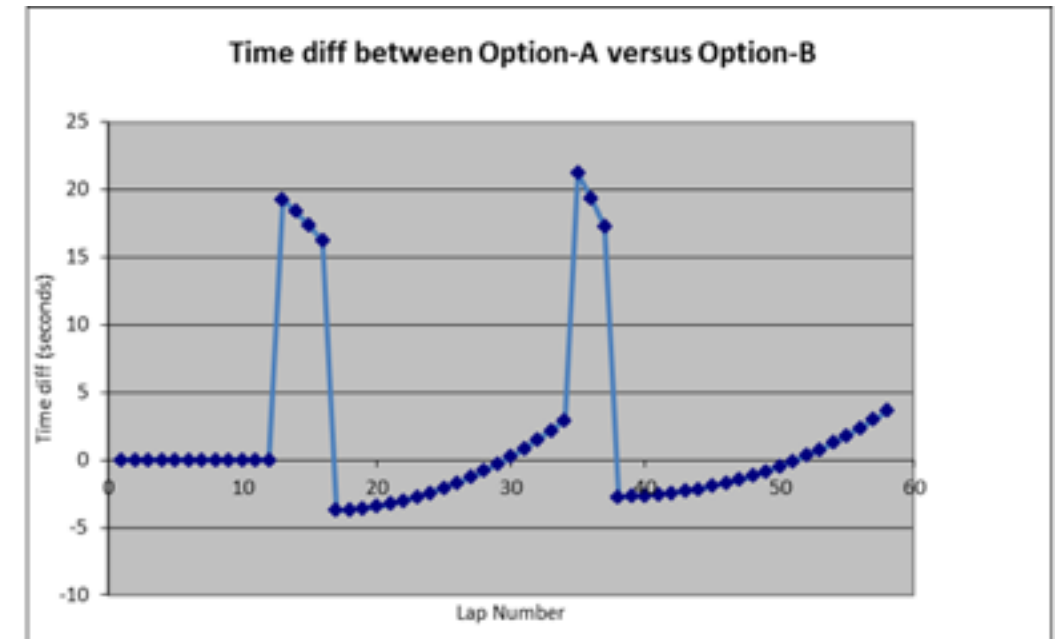
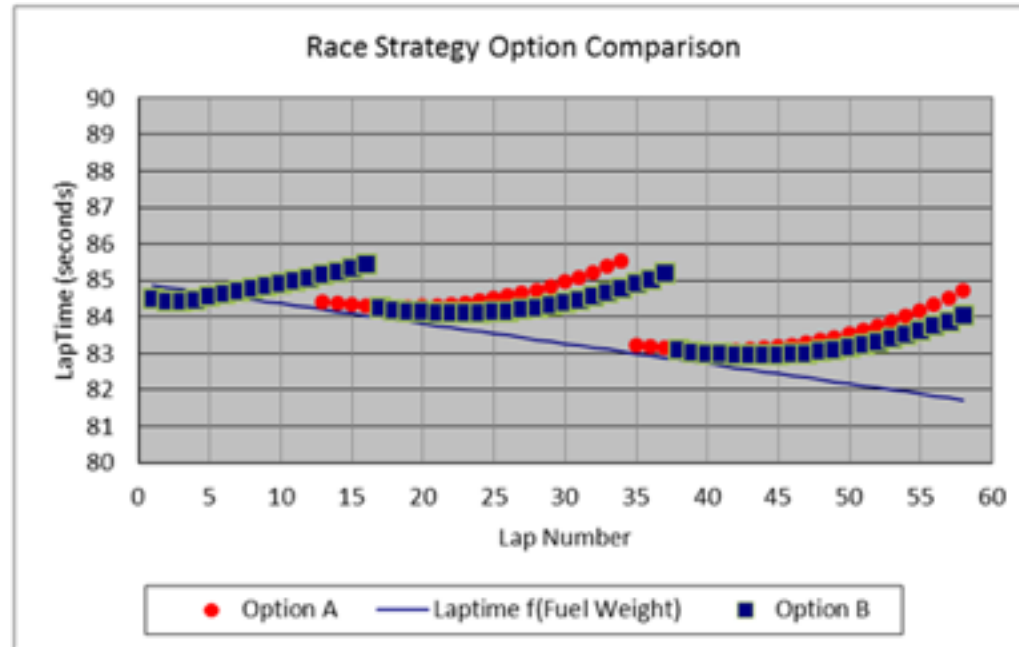
# How data plays in F1 motorsports



# Data in Motorsports

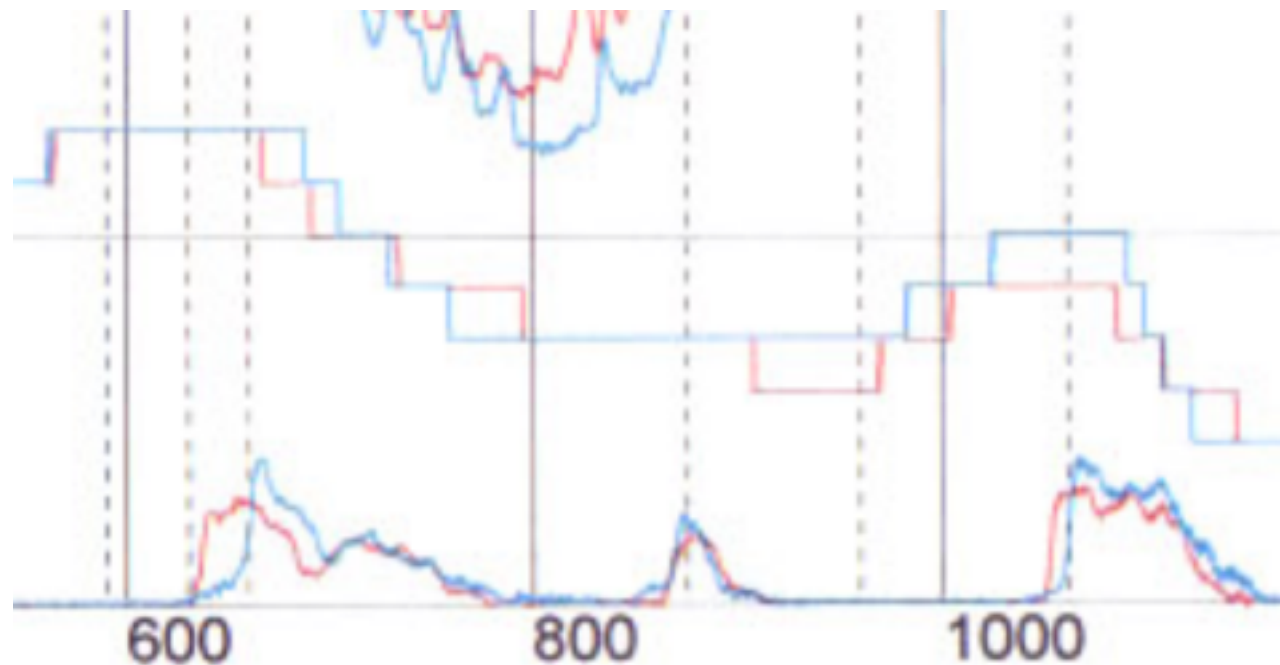


# Race Strategy



F1 Framework - <http://f1framework.blogspot.be/2013/08/race-strategy-explained.html>





Difference is due to  
later and sharper  
braking

# Got examples?

- Up to **300 sensors** per car
- Up to **2000 channels**
- Sensor data are sent to the paddock in **2ms**
- **1.5 billions** of data points for a race
- **5 billions** for a full race weekend
- **5/6Gb** of compressed data per car for 90mn

# Got examples?

- Up to **300 sensors** per car

# Got examples?

- Up to **300 sensors** per car
- Up to **2000 channels**

# Got examples?

- Up to **300 sensors** per car
- Up to **2000 channels**
- Sensor data are sent to the paddock in **2ms**

# Got examples?

- Up to **300 sensors** per car
- Up to **2000 channels**
- Sensor data are sent to the paddock in **2ms**
- **1.5 billions** of data points for a race

# Got examples?

- Up to **300 sensors** per car
- Up to **2000 channels**
- Sensor data are sent to the paddock in **2ms**
- **1.5 billions** of data points for a race
- **5 billions** for a full race weekend

# Got examples?

- Up to **300 sensors** per car
- Up to **2000 channels**
- Sensor data are sent to the paddock in **2ms**
- **1.5 billions** of data points for a race
- **5 billions** for a full race weekend
- **5/6Gb** of compressed data per car for 90mn



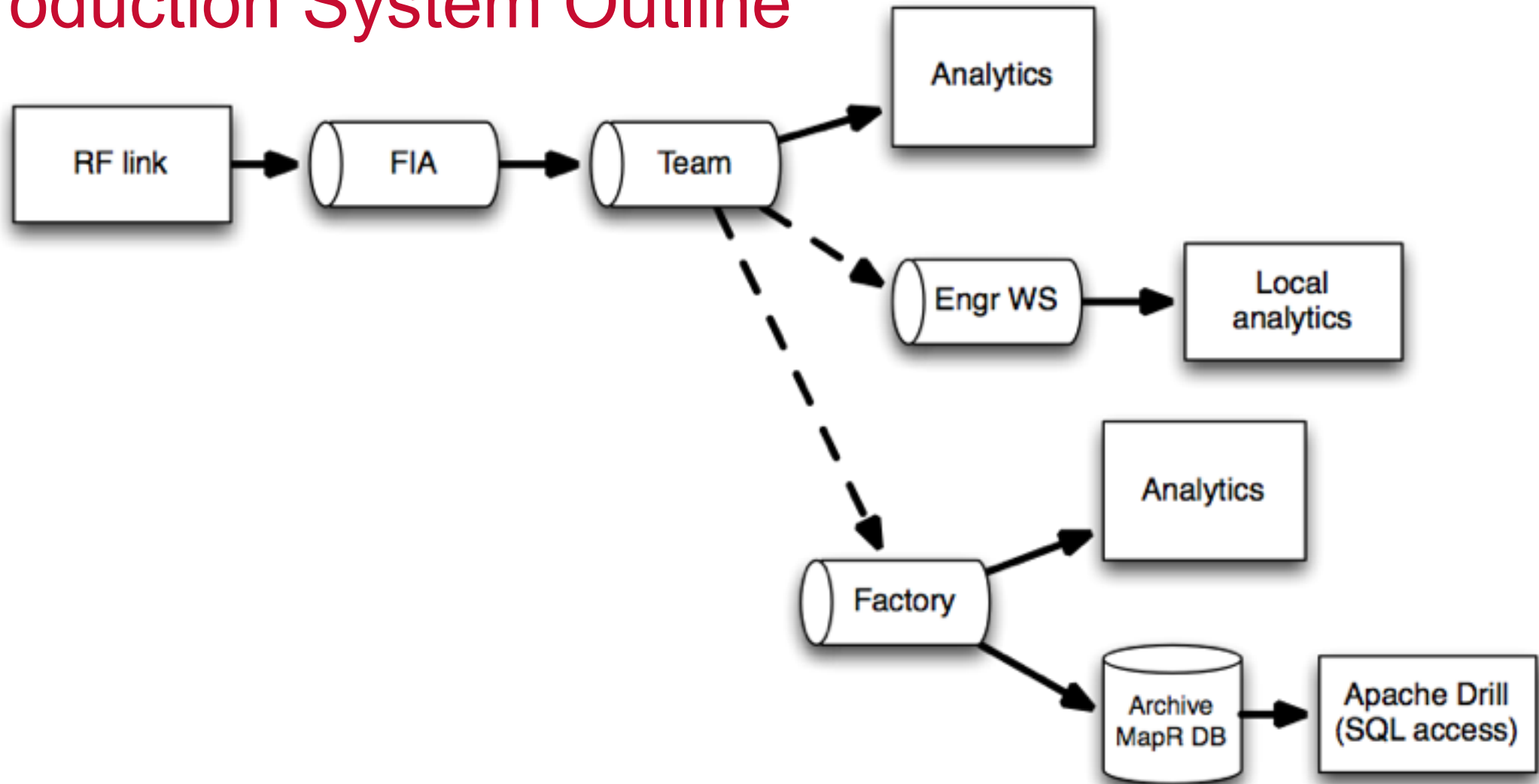
## Got examples?

- Up to **300 sensors** per car
- Up to **2000 channels**
- Sensor data are sent to the paddock in **2ms**
- **1.5 billions** of data points for a race
- **5 billions** for a full race weekend
- **5/6Gb** of compressed data per car for 90mn

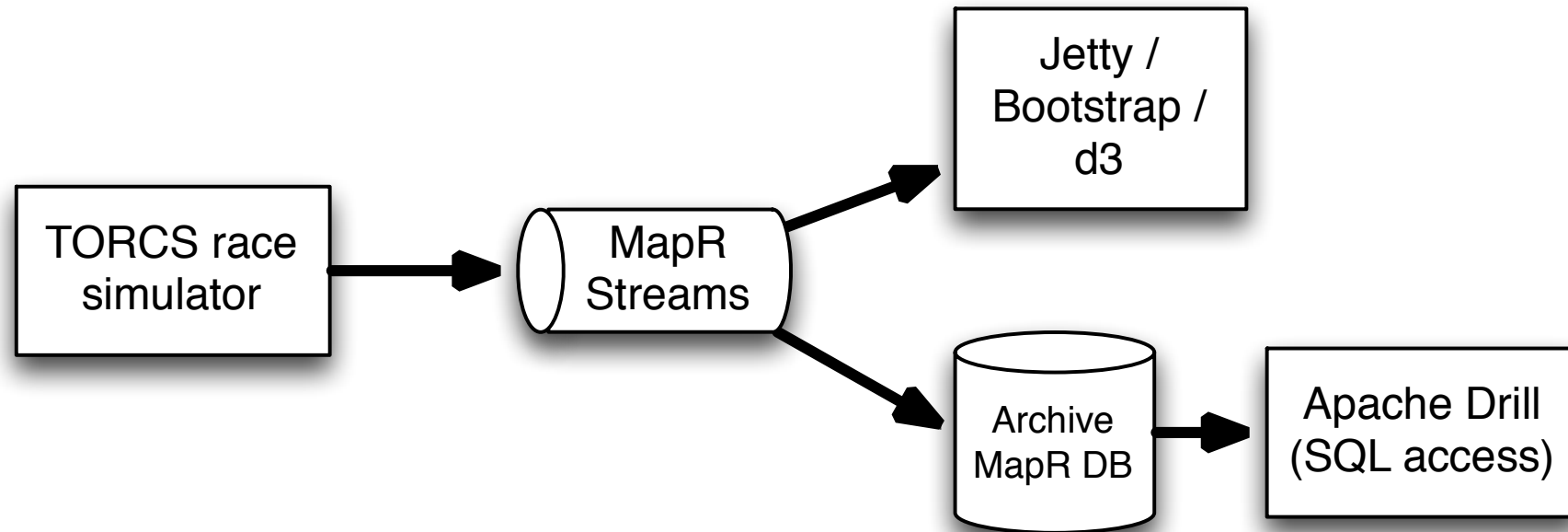
**US Grand Prix 2014 : 243 Tb (race teams combined)**

So how does that work?  
Especially for real-time data?

# Production System Outline



# Simplified Demo System Outline



# TORCS for Cars, Physics and Drivers

The Open Racing Car Simulator

<http://torcs.sourceforge.net/>

Car racing game

AI & Research Platform



Demo time!

# IoT : Racing Cars

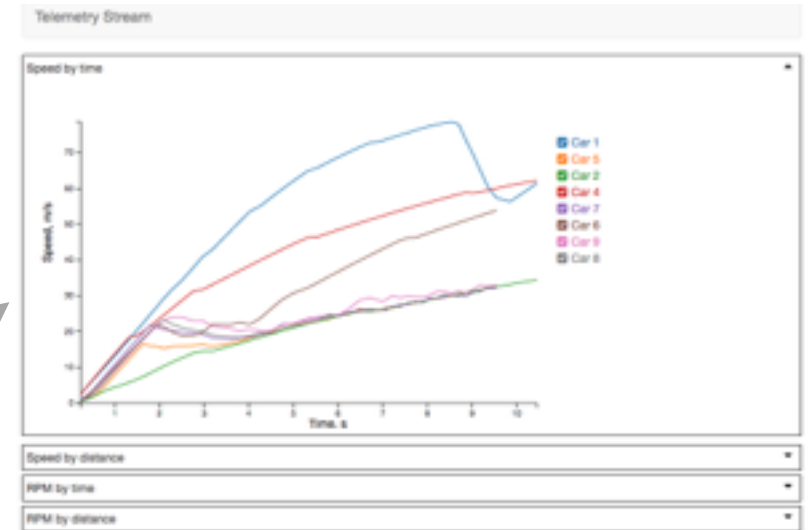


Events Producers

sensors data



Real Time



Analytics



Analytics  
with SQL

Events Consumers

# IoT : Racing Cars



kafka  
Producer

Events Producers

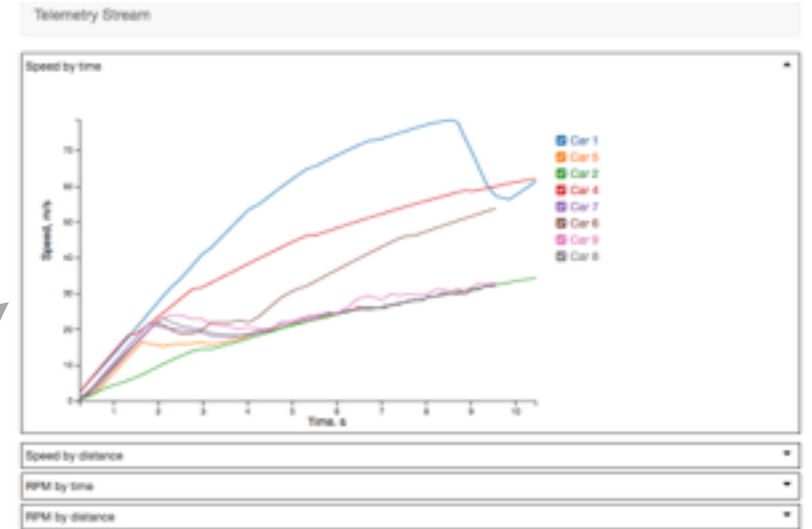
sensors data



Real Time

kafka  
Consumer

Analytics



MAPR-DB  
JSON



Events Consumers



Where/How to store data ?

# Big Datastore



Distributed File System  
HDFS/MapR-FS



NoSQL Database  
HBase/MapR-DB

....

# Data Streaming

# Data Streaming

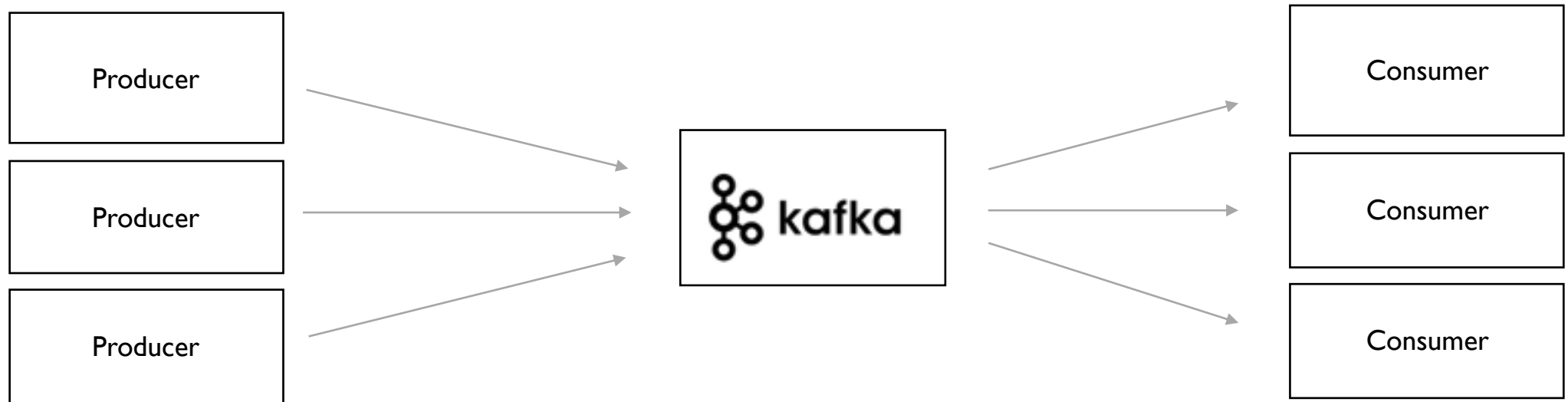
Moving millions of events per h|mn|s

# What is Kafka?

- <http://kafka.apache.org/>
- Created at LinkedIn, open sourced in 2011
- Implemented in Scala / Java
- Distributed messaging system built to scale

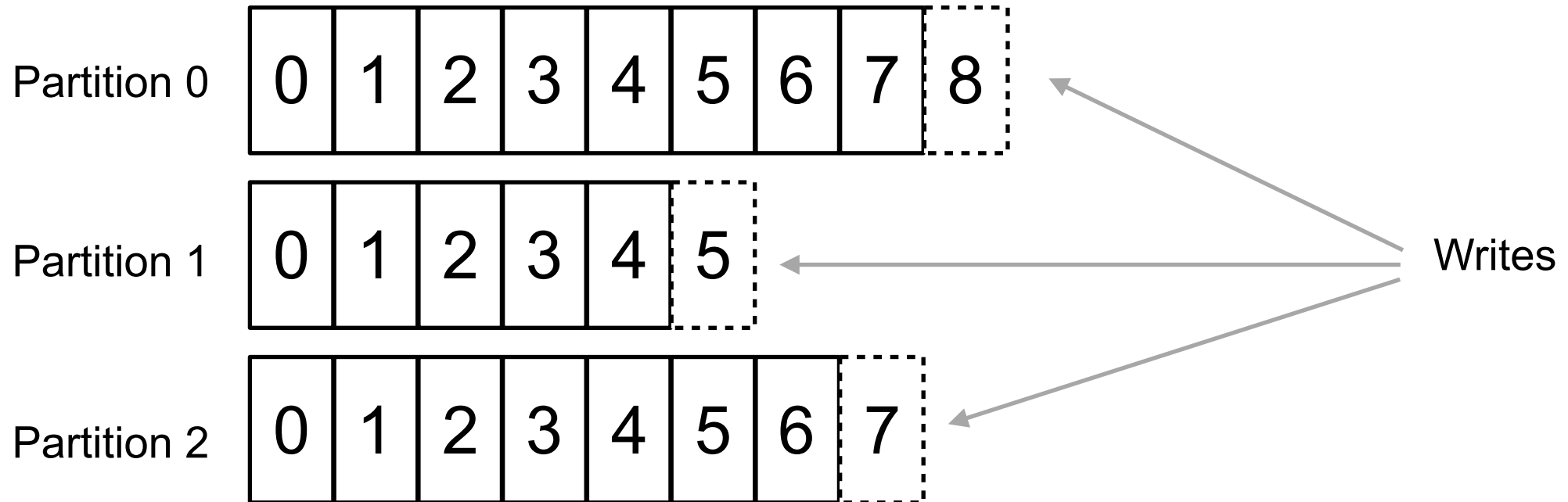


# Big Picture



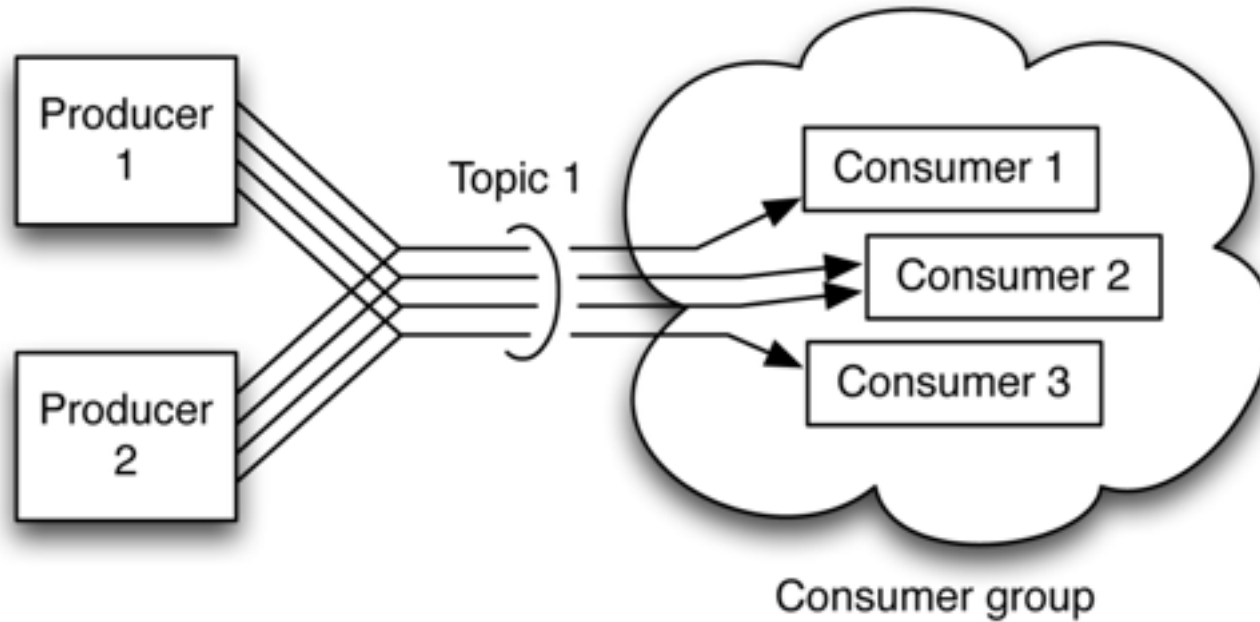
# Topics and Partitions

- Split topics into partitions for scalability



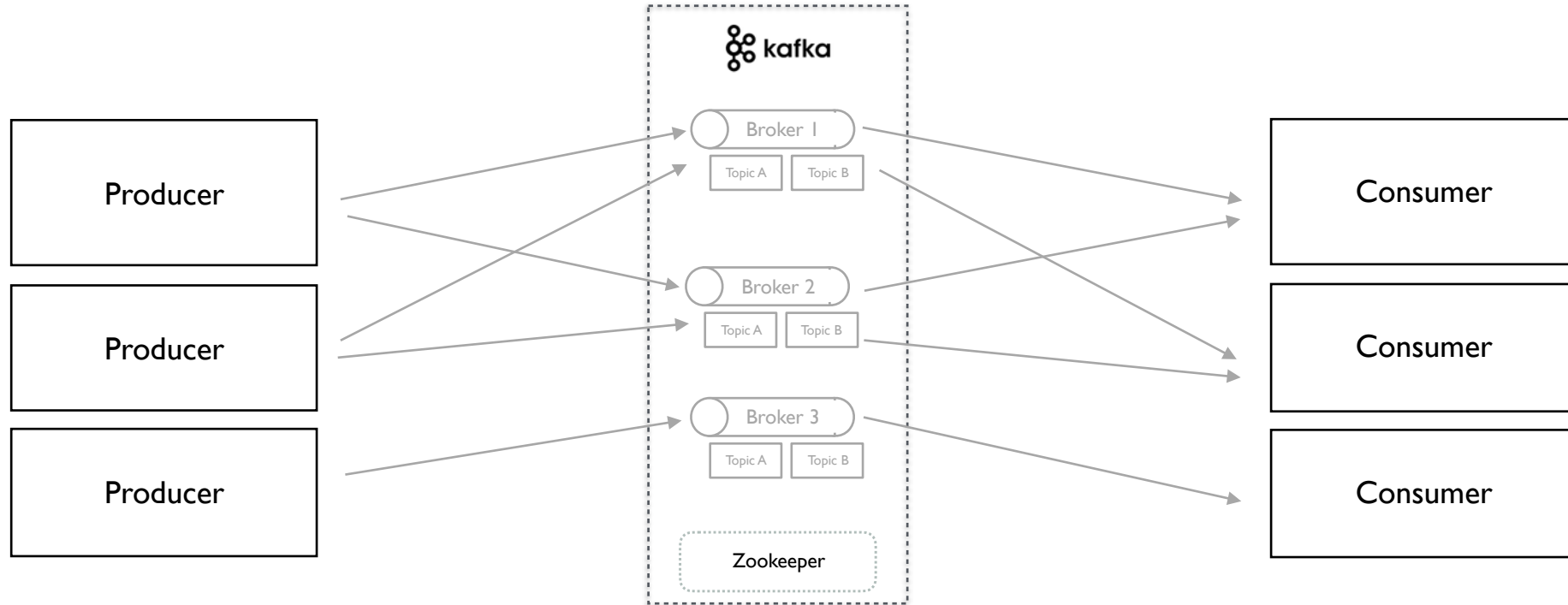
# Consumer Groups

- Single cor
- Max 1 cor
- Any numk





# More real life Kafka ...



# MAPR STREAMS

- Distributed messaging system built to scale
- Use Apache Kafka API 0.9.0
- No code change
- Does not use the same “broker” architecture
  - Log stored in MapR Storage  
(Scalable, Secured, Fast, Multi DC)
  - No Zookeeper

# Produce Messages

```
ProducerRecord<String, byte[]> rec = new ProducerRecord<>(
    "/apps/racing/stream:sensors_data",
    eventName,
    value.toString().getBytes());

producer.send(rec, (recordMetadata, e) -> {
    if (e != null) { ... });

producer.flush();
```



# Consume Messages

```
long pollTimeOut = 800;
while(true) {
    ConsumerRecords<String, String> records = consumer.poll(pollTimeOut);
    if (!records.isEmpty()) {
        Iterable<ConsumerRecord<String, String>> iterable = records.iterator();
        StreamSupport.stream(iterable.spliterator(), false).forEach((record) -> {
            // work with record object
            ... record.value();
            ...

        });
        consumer.commitAsync();
    }
}
```



What 's next?

# Sensor Data V1

- 3 main data points:
  - Speed (m/s)
  - RPM
  - Distance (m)
- Buffered

```
{  "_id": "1.458141858E9/0.324",
   "car" = "car1",
   "timestamp": 1458141858,
   "racetime": 0.324,
   "records":
     [
       {
         "sensors": {
           "Speed": 3.588583,
           "Distance": 2003.023071,
           "RPM": 1896.575806
         },
         "racetime": 0.324,
         "timestamp": 1458141858
       },
       {
         "sensors": {
           "Speed": 6.755624,
           "Distance": 2004.084717,
           "RPM": 1673.264526
         },
         "racetime": 0.556,
         "timestamp": 1458141858
       },
     ]
 }
```



Capture more data

# Sensor Data V2

- 3 main data points:
  - Speed (m/s)
  - RPM
  - Distance (m)
  - **Throttle**
  - **Gear**
  - ...
- Buffered

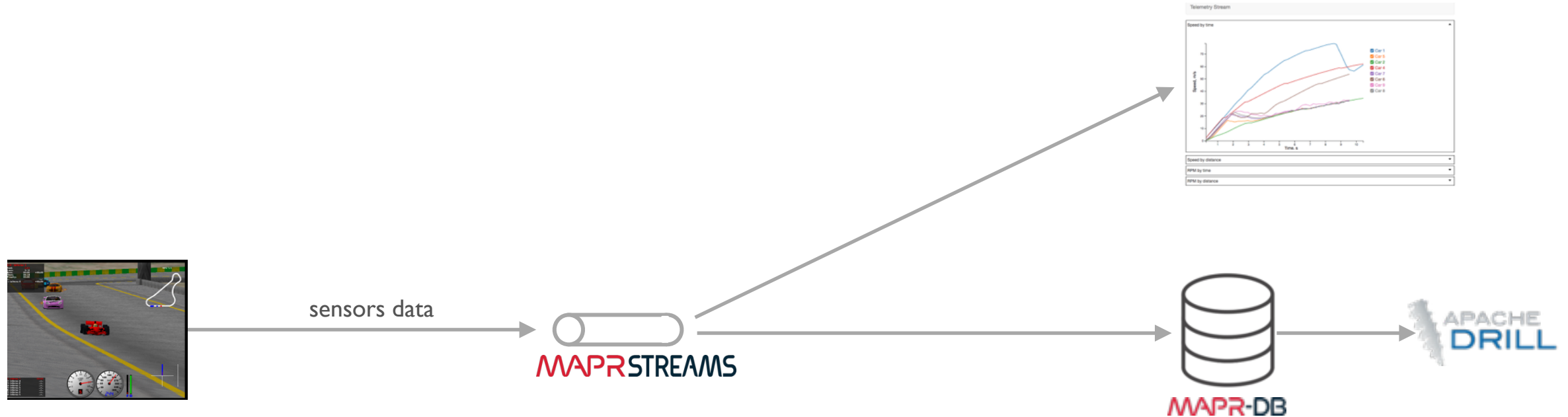
```
{  "_id": "1.458141858E9/0.324",
   "car" = "car1",
   "timestamp": 1458141858,
   "racetime": 0.324,
   "records":
     [
       {
         "sensors": {
           "Speed": 3.588583,
           "Distance": 2003.023071,
           "RPM": 1896.575806,
           "gear" : 2
         },
         "racetime": 0.324,
         "timestamp": 1458141858
       },
       {
         "sensors": {
           "Speed": 6.755624,
           "Distance": 2004.084717,
           "RPM": 1673.264526,
           "gear" : 2
         },
         "racetime": 0.556,
         "timestamp": 1458141858
       },
     ]
}
```



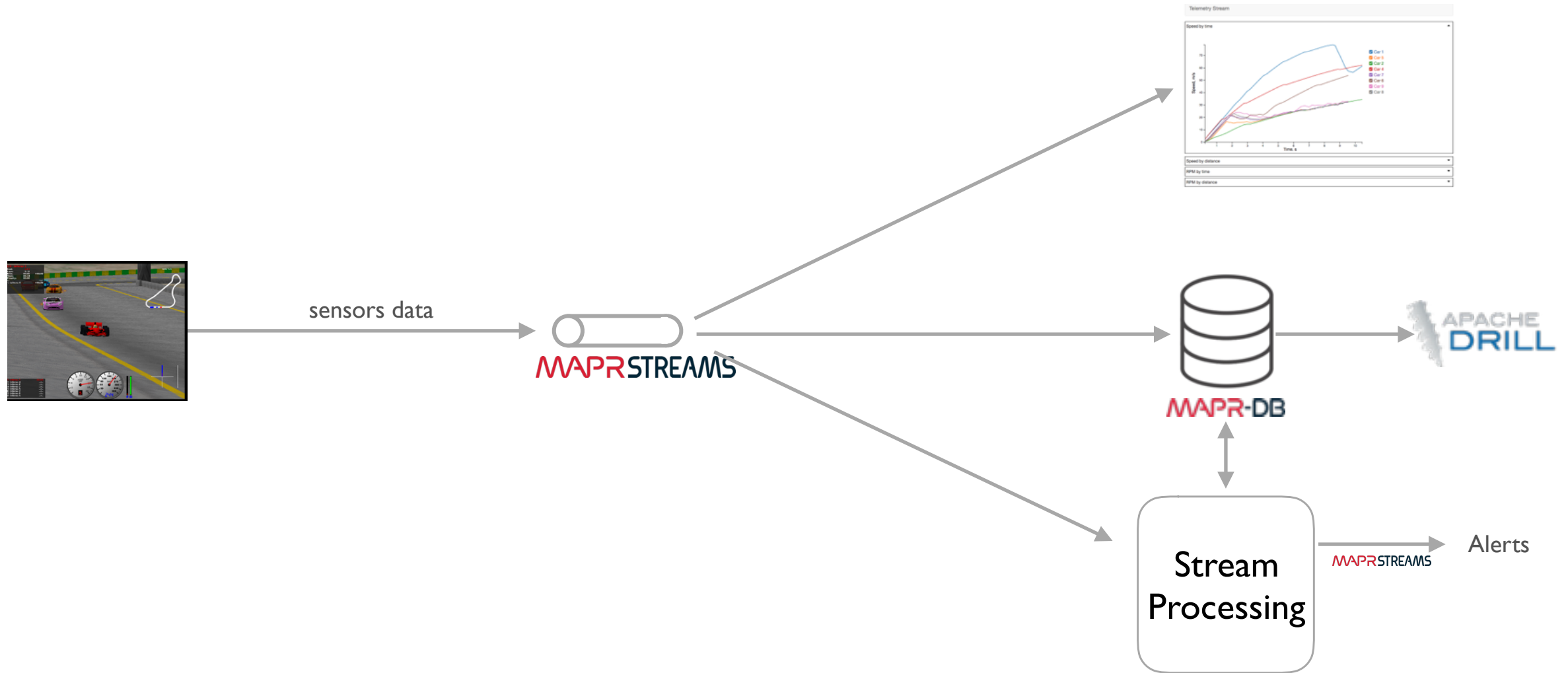


Add new services

# New Data Service



# New Data Service



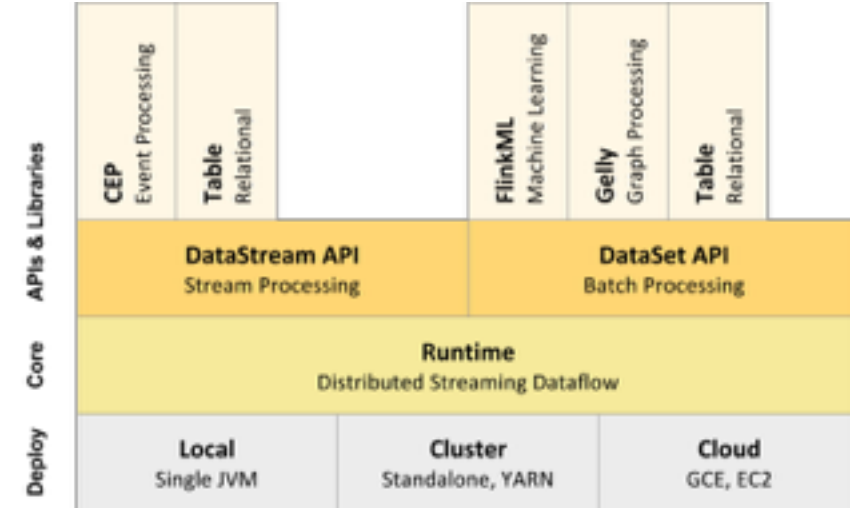


- Cluster Computing Platform
- Extends “MapReduce” with extensions
  - Streaming
  - Interactive Analytics
- Run in Memory
- <http://spark.apache.org/>





- Streaming Dataflow Engine
  - Datastream/Dataset APIs
  - CEP, Graph, ML
- Run in Memory
- <https://flink.apache.org/>



# DEMO

## Stream Processing with Flink

# Streaming Architecture & Formula 1

- Stream data in real time
- Big Data Store to deal with the scale
  - NoSQL Database, Distributed File System
- Decouple the source from the consumer(s)
  - Dashboard, Analytics, Machine Learning
  - Add new use case....

# Streaming Architecture & Formula 1

- Stream data in real time
- Big Data Store to deal with the scale
  - NoSQL
- Decoupled
  - Data Lake
  - Add new use cases...

**This is not only about Formula 1!**  
**(Telco, Finance, Retail, Content, IT)**



---

# Fast Cars, Big Data

## How Streaming Can Help Formula 1

Tugdual Grall  
@tgrall