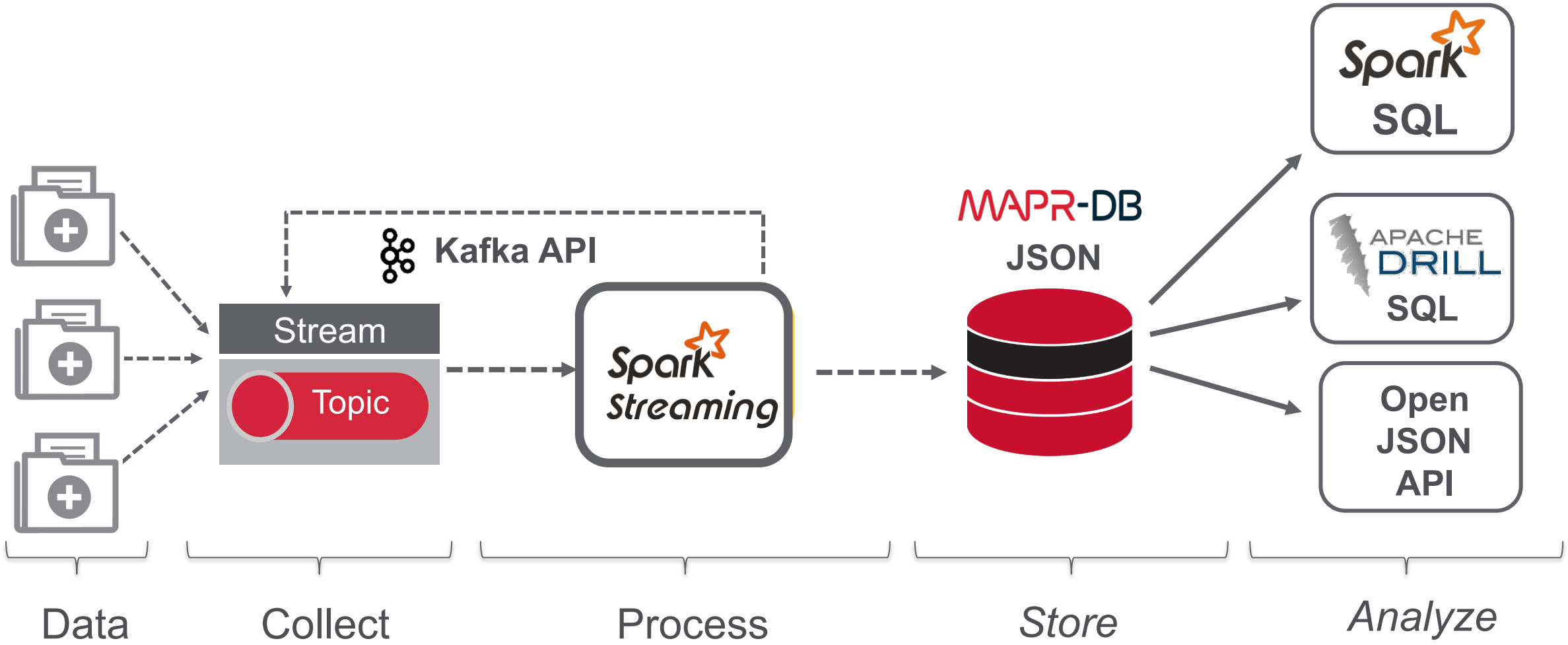
Data Pipeline Using Apache APIs: Kafka, Spark, and MapR-DB

Data Pipeline Using Apache APIs: Kafka, Spark, and MapR-DB

- Kafka
- Spark Streaming
- Spark SQL

Streaming ETL Pipeline



Traditional ETL

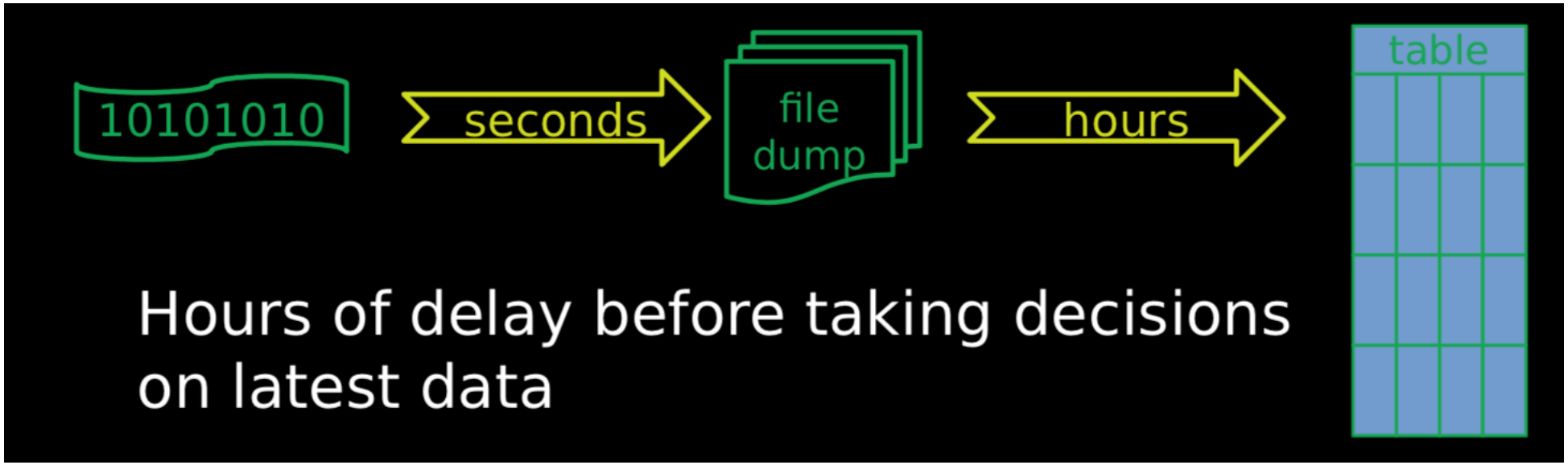


Image Reference: Databricks

Streaming ETL

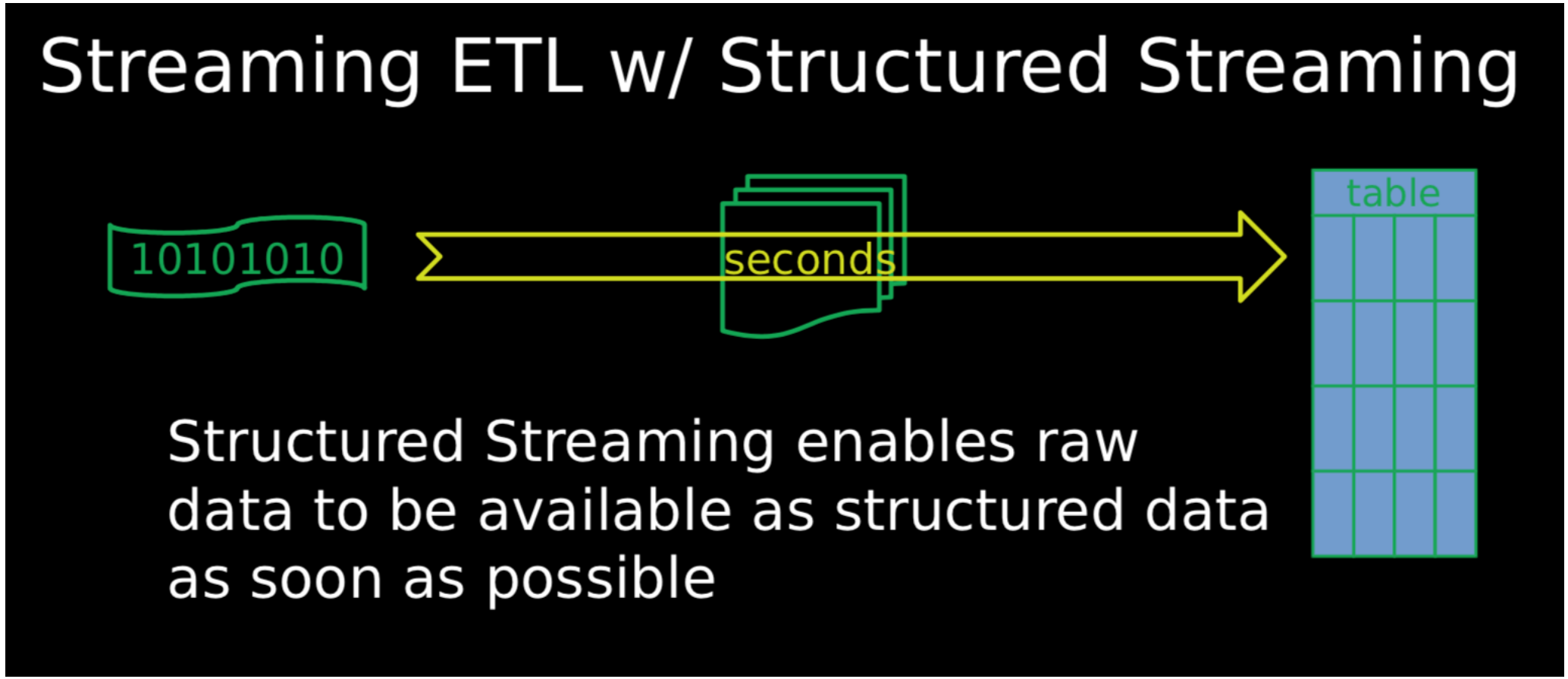
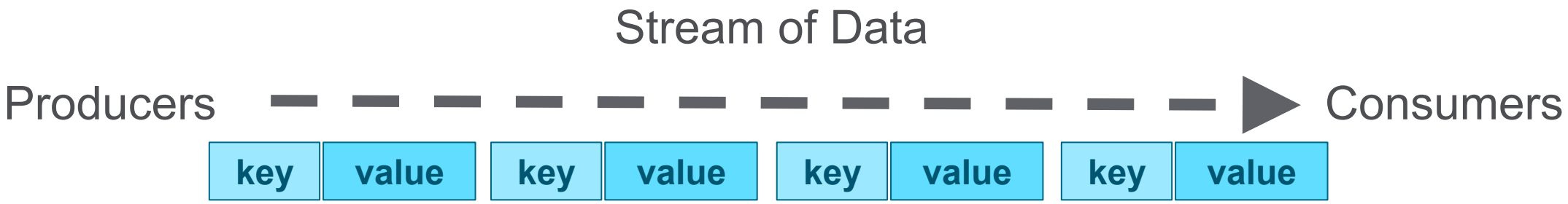


Image Reference: Databricks

What is a Stream ?

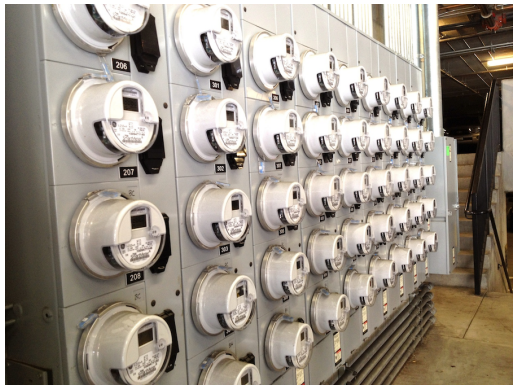


- A **stream** is an **continuous** sequence of events or records
- Records are key-value pairs

Examples of Streaming Data

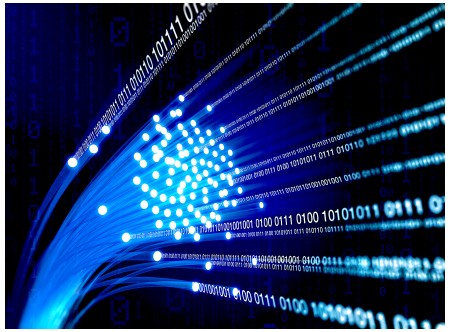
Fraud detection

Smart Machinery



Smart Meters

Home Automation



Networks

Manufacturing

Security Systems

Patient Monitoring

Examples of Streaming Data

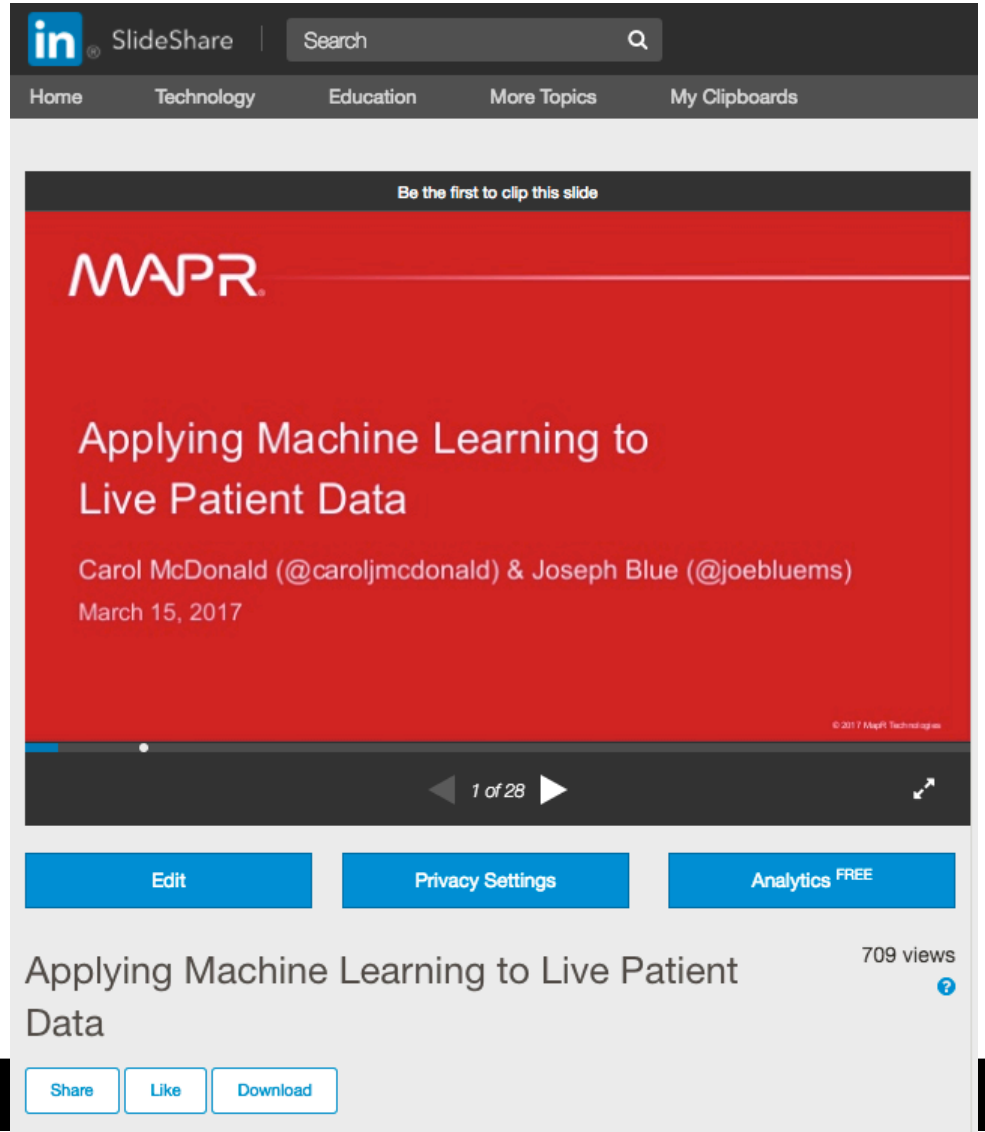
- Monitoring devices combined with **ML** can provide **alerts** for Sepsis, which is one of the leading causes for death in hospitals
  - http://www.computerweekly.com/news/450422258/Putting-sepsis-algorithms-into-electronic-patient-records

Examples of Streaming Data

- A Stanford team has shown that a machine-learning model can identify **heart arrhythmias** from an electrocardiogram (**ECG**) better than an expert
  - https://www.technologyreview.com/s/608234/the-machines-are-getting-ready-to-play-doctor/

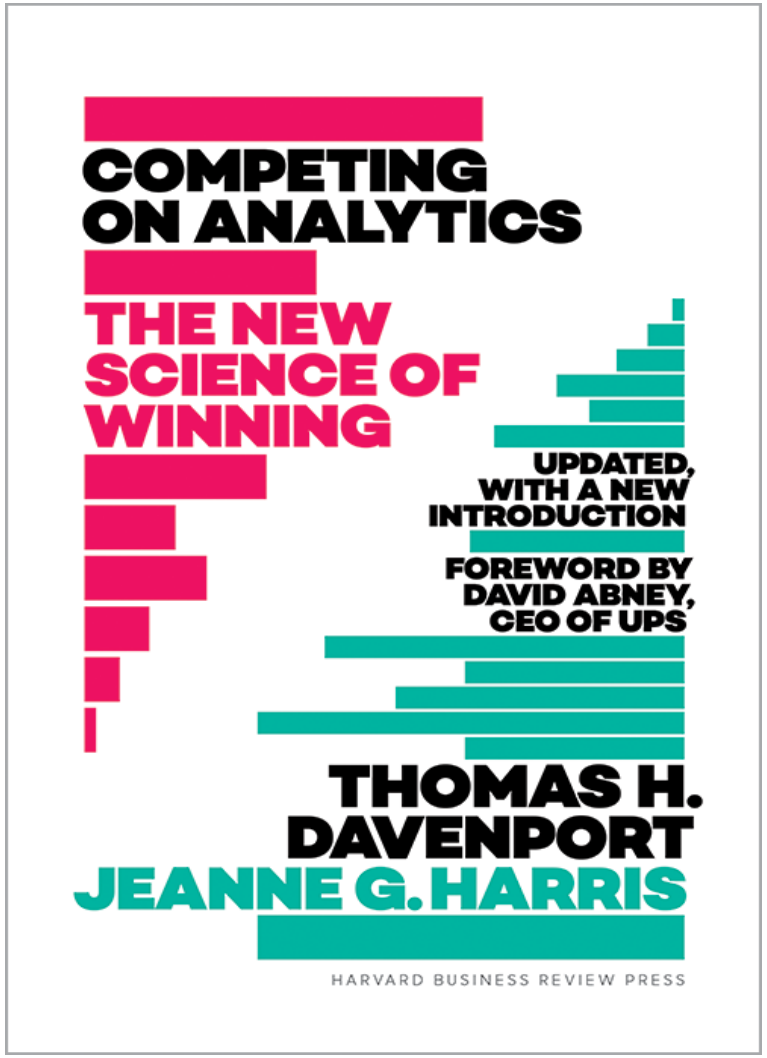
Applying Machine Learning to Live Patient Data

- **https://www.slideshare.net/caroljmcdonald/applying-machine-learning-to-live-patient-data**

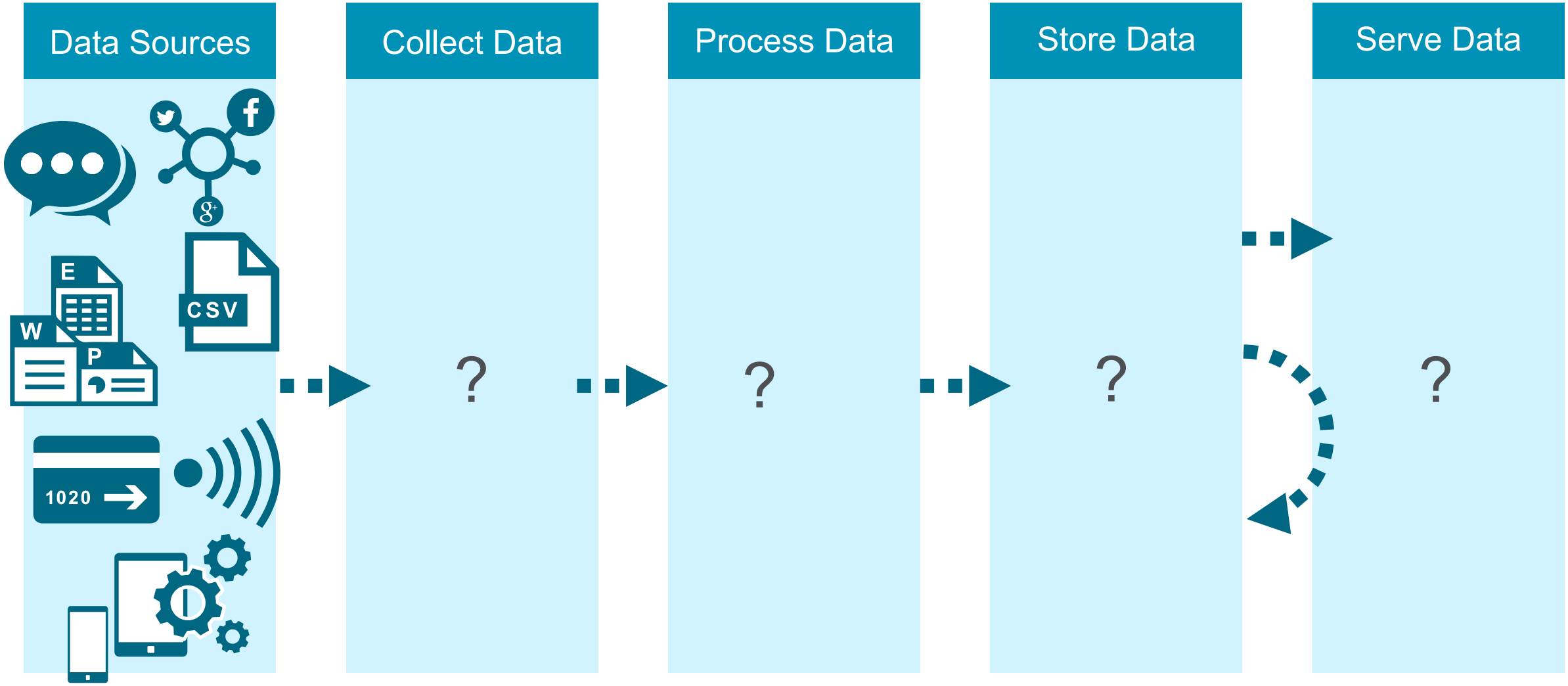


What has changed in the past 10 years?

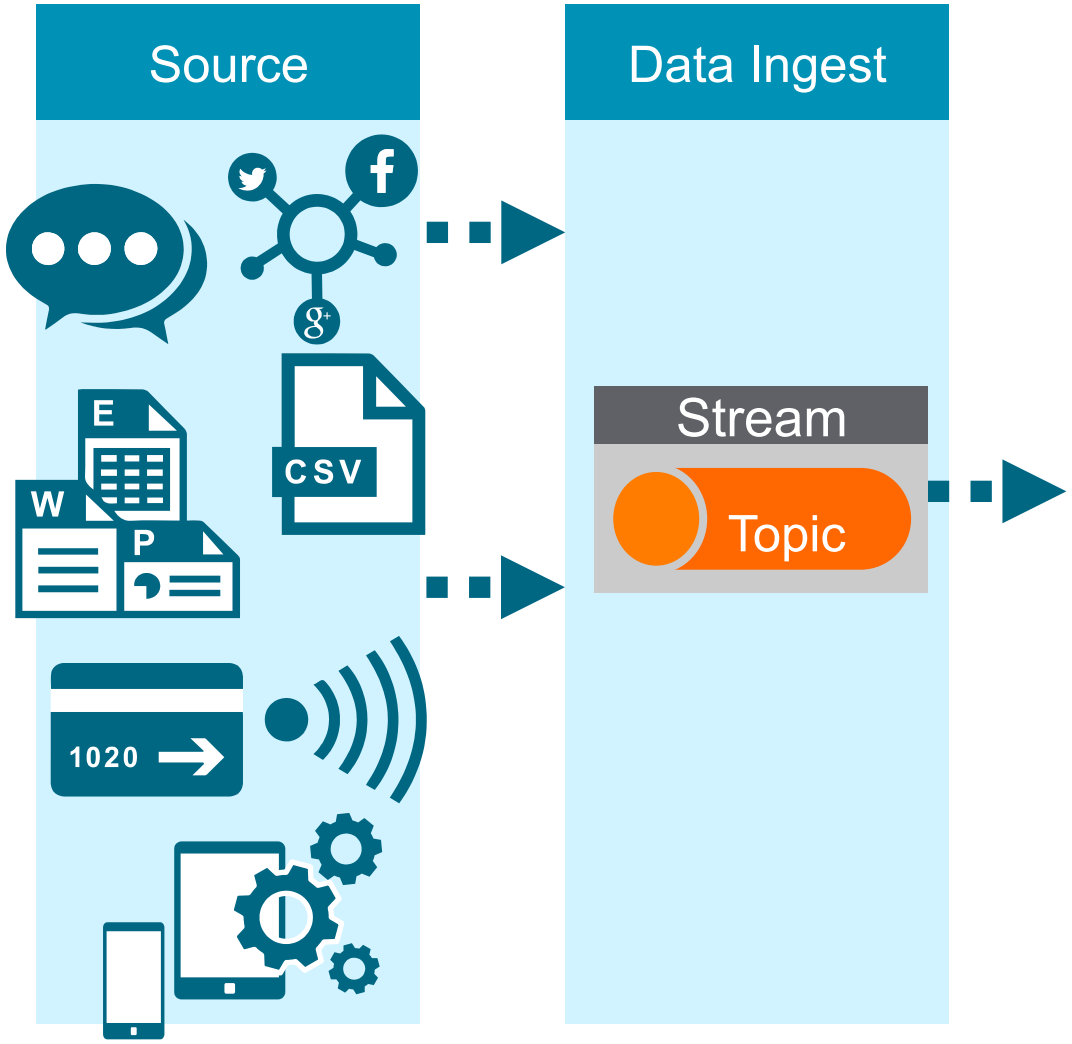
- Distributed computing
- Streaming analytics
- Improved machine learning



What Do We Need to Do ?



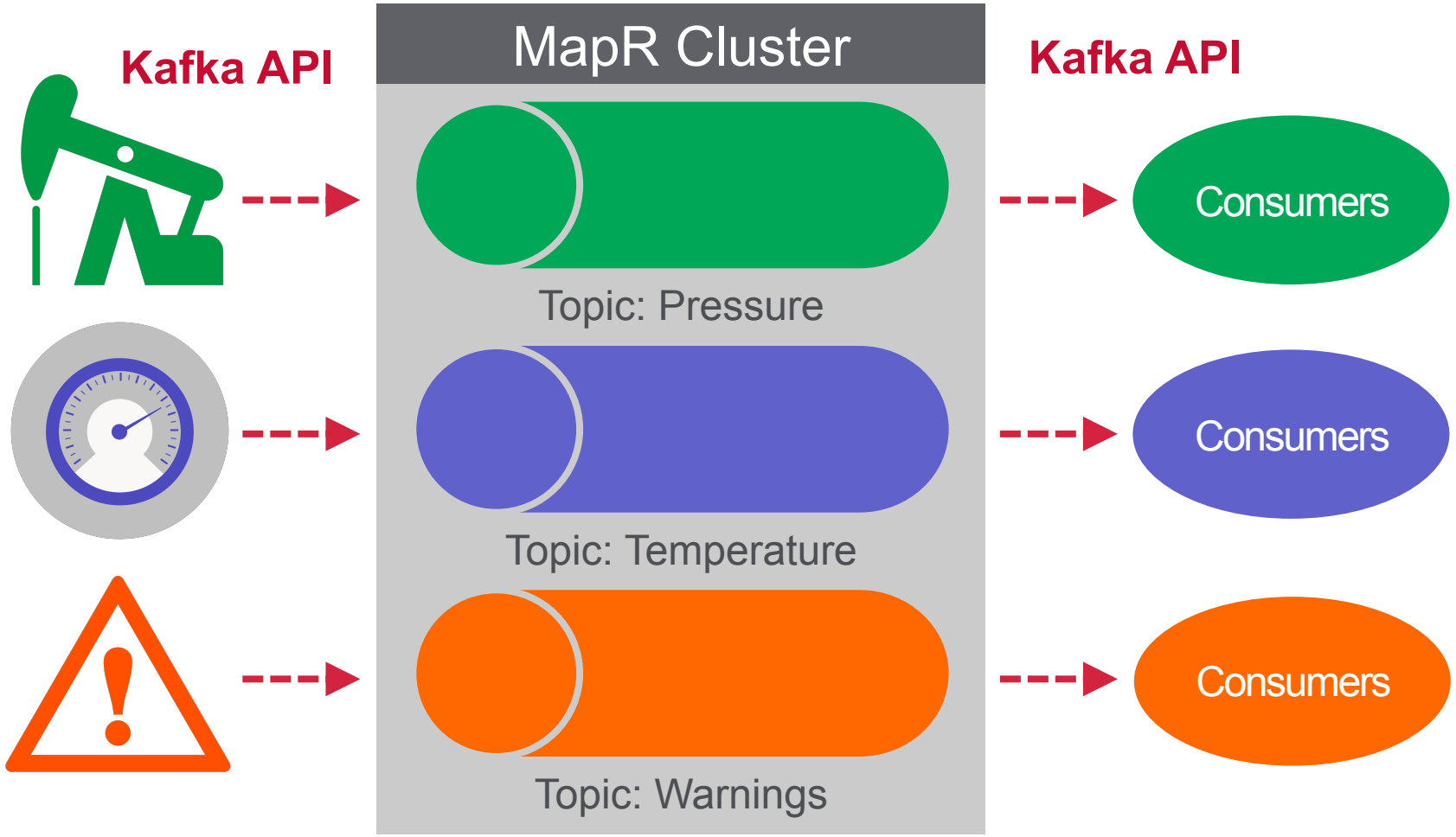
Collect the Data



- **Data Ingest**:
  - Using the Kafka API

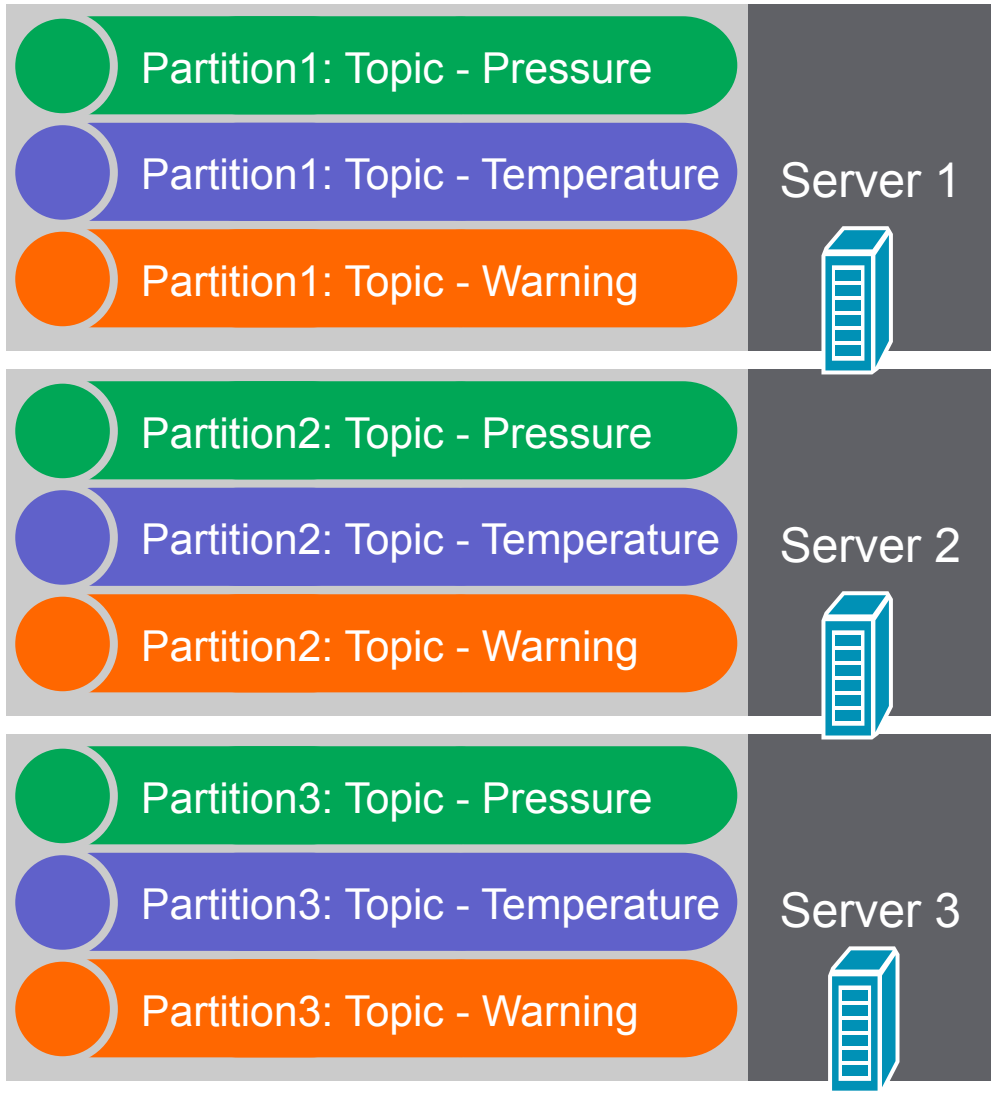
Organize Data into Topics with MapR-Event Streams

Topics: Logical collection of events, Organize Events into Categories

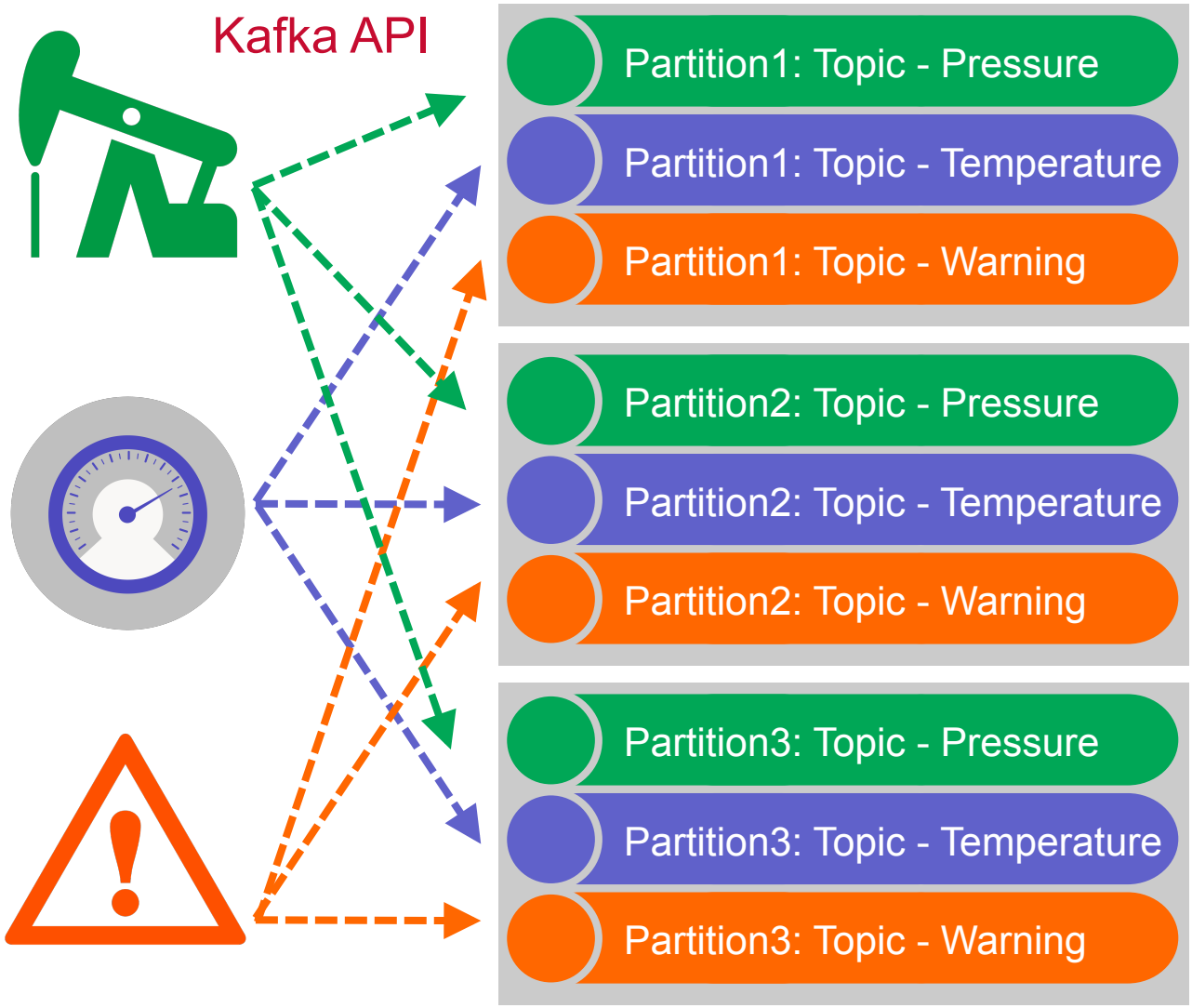


Scalable Messaging with MapR Event Streams

Topics are partitioned for throughput and scalability

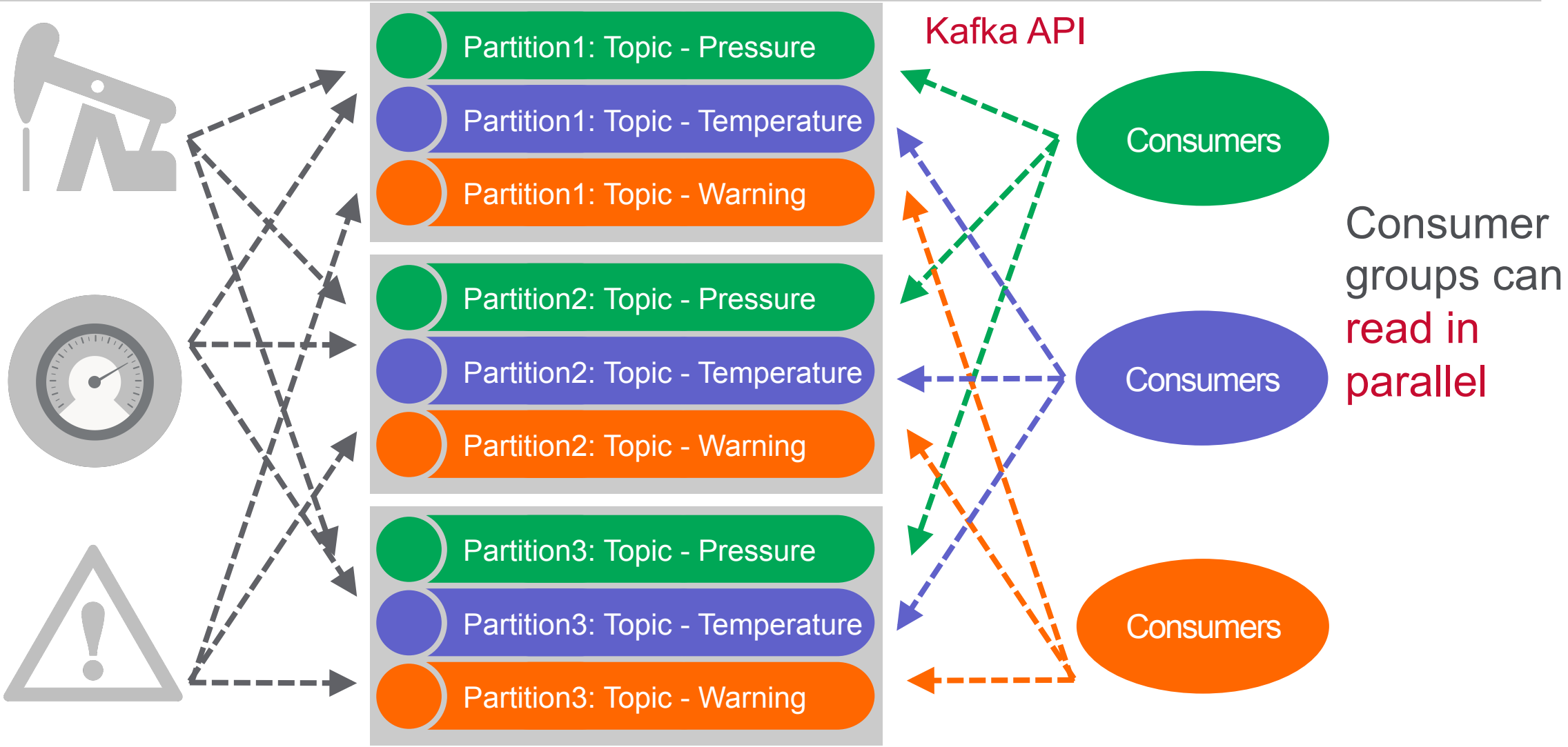


Scalable Messaging with MapR Event Streams



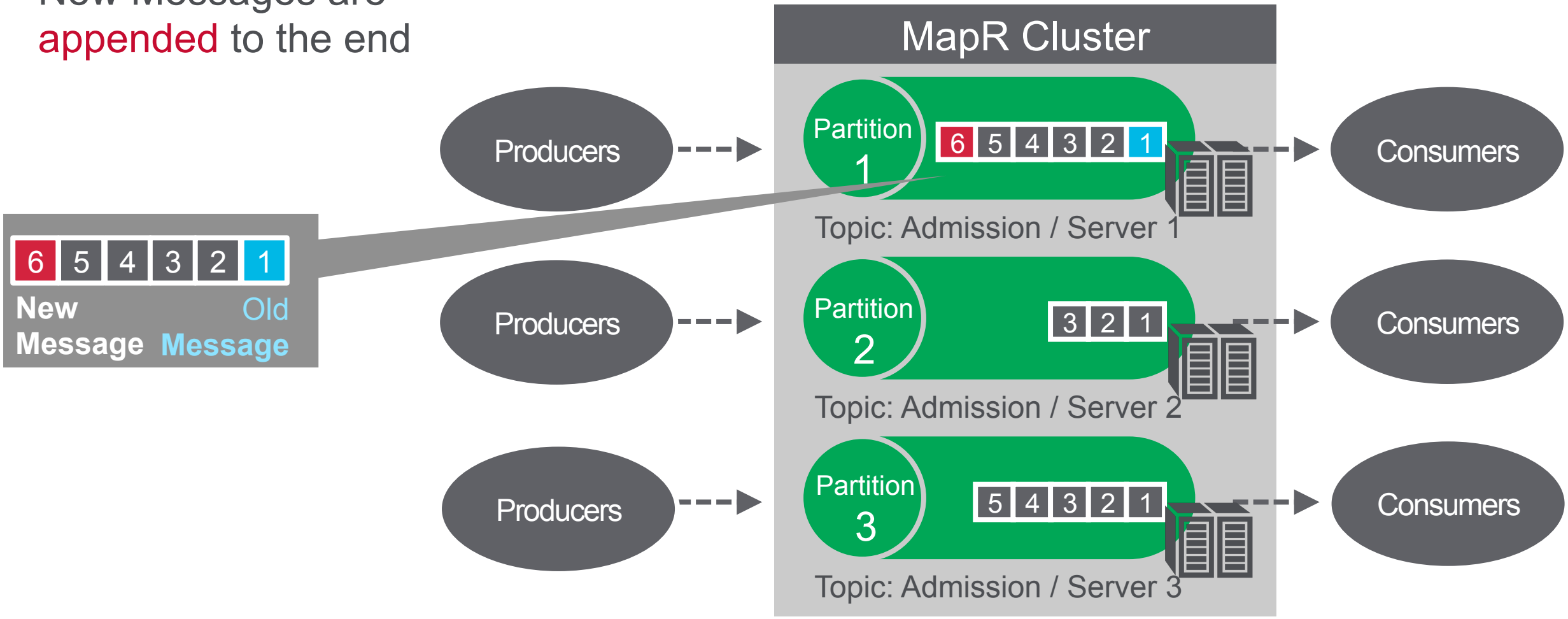
Producers are **load balanced** between partitions

Scalable Messaging with MapR Event Streams



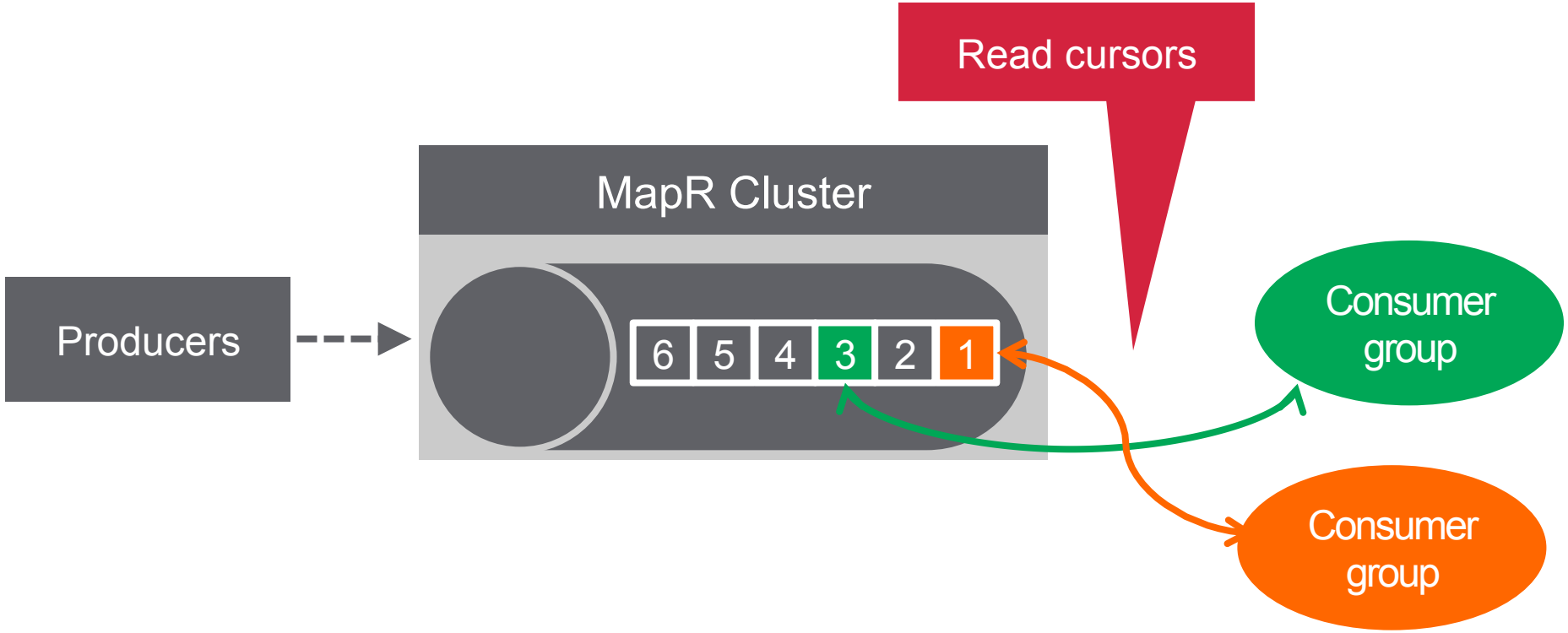
Partition is like an Event Log

New Messages are appended to the end



Partition is like a Queue

Messages are delivered in the order they are received

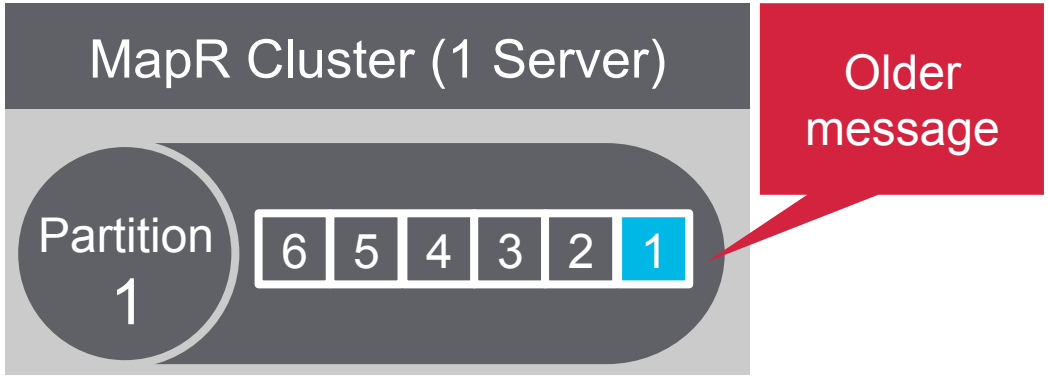


Unlike a queue, events are still persisted after they're delivered

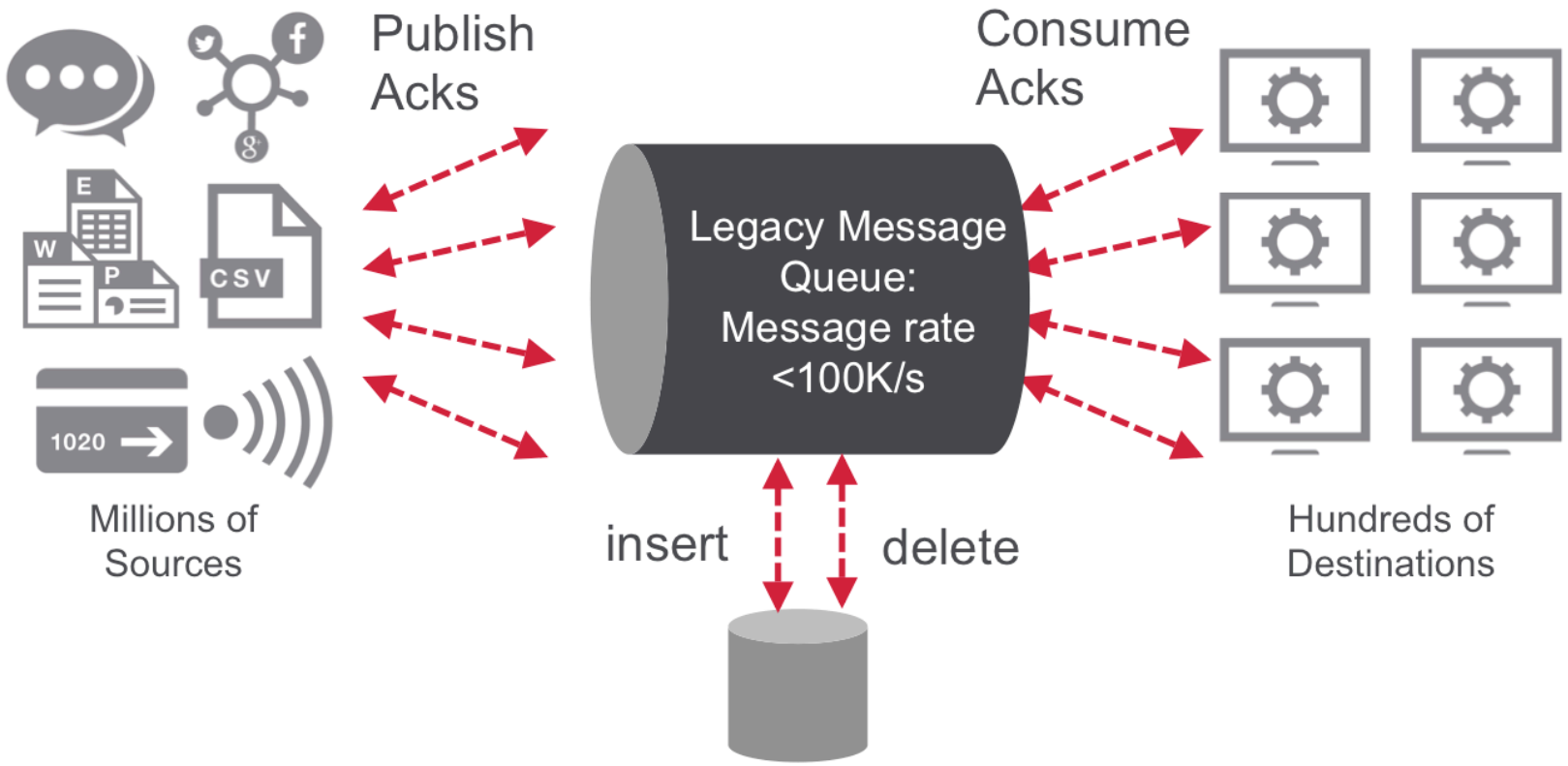
Messages remain on the partition, available to other consumers

When Are Messages Deleted?

- Messages can be persisted forever
- Or
- Older messages can be deleted automatically based on time to live



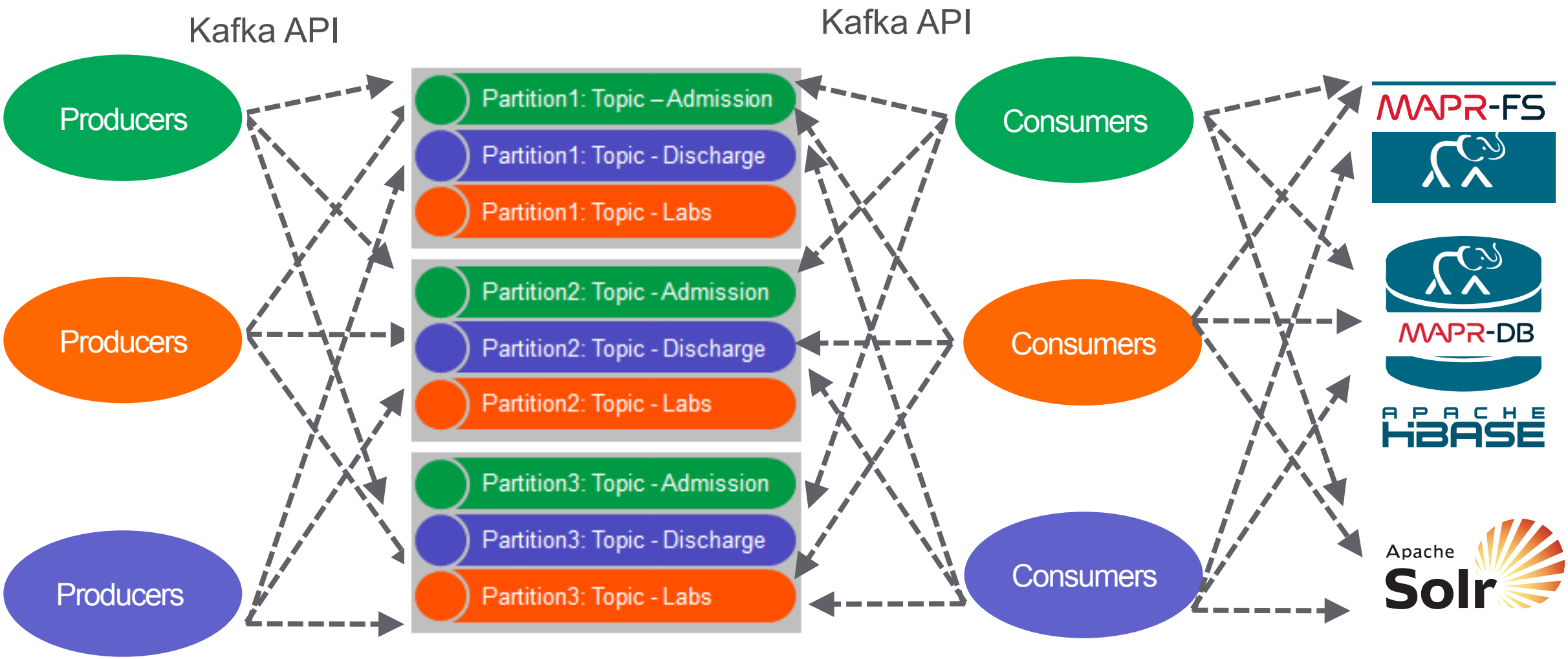
Traditional Message queue



How do we do this with High Performance at Scale?

- Parallel operations
- minimizes disk read/writes

Processing Same Message for Different Purposes



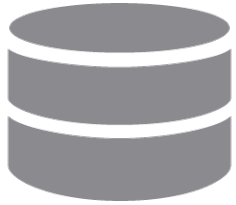
Stream as the System of Record

A Table is a Snapshot of a Stream

Imagine each event as a change to an entry in a database.

1: **WillO : Deposit : 100.00**
2: **BradA : Deposit : 50.00**
3: **BradA : Withdraw : 30.00**
4: **WillO : Withdraw: 20.00**

Change log

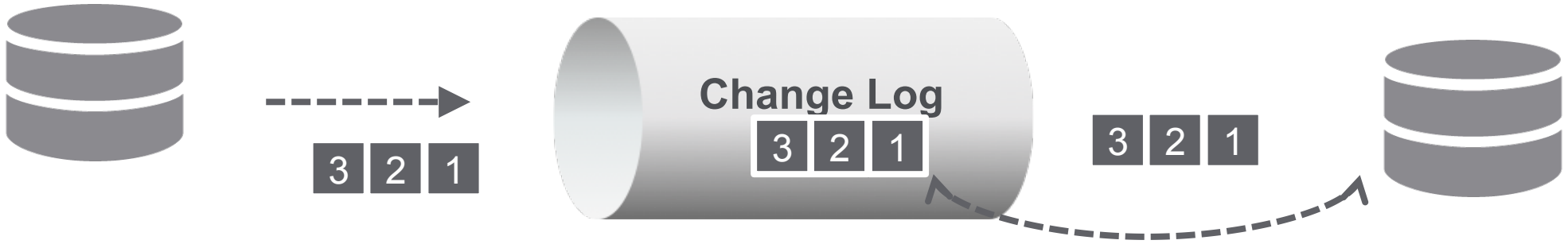


Account Id	Balance
WillO	80.00
BradA	20.00

https://engineering.linkedin.com/distributed-systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying

A Stream is a Change Log of a Table

Replication of changes



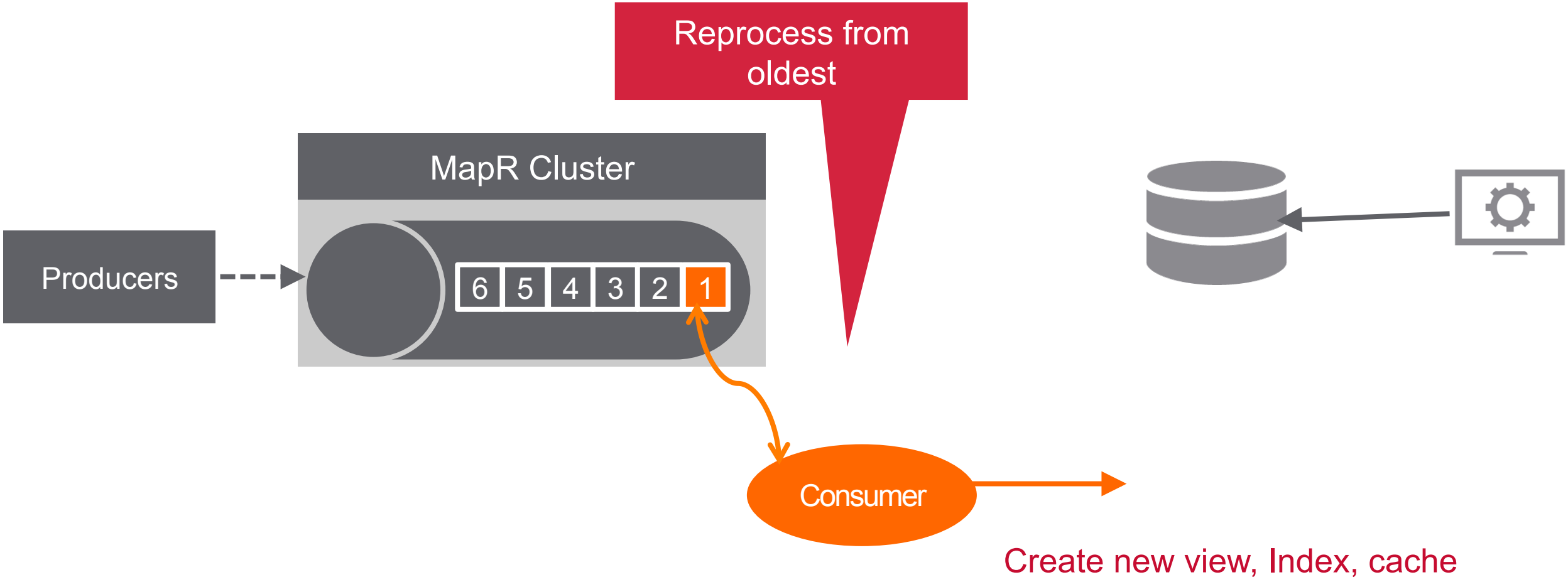
Master:
Append writes

Slave:
Apply writes in order

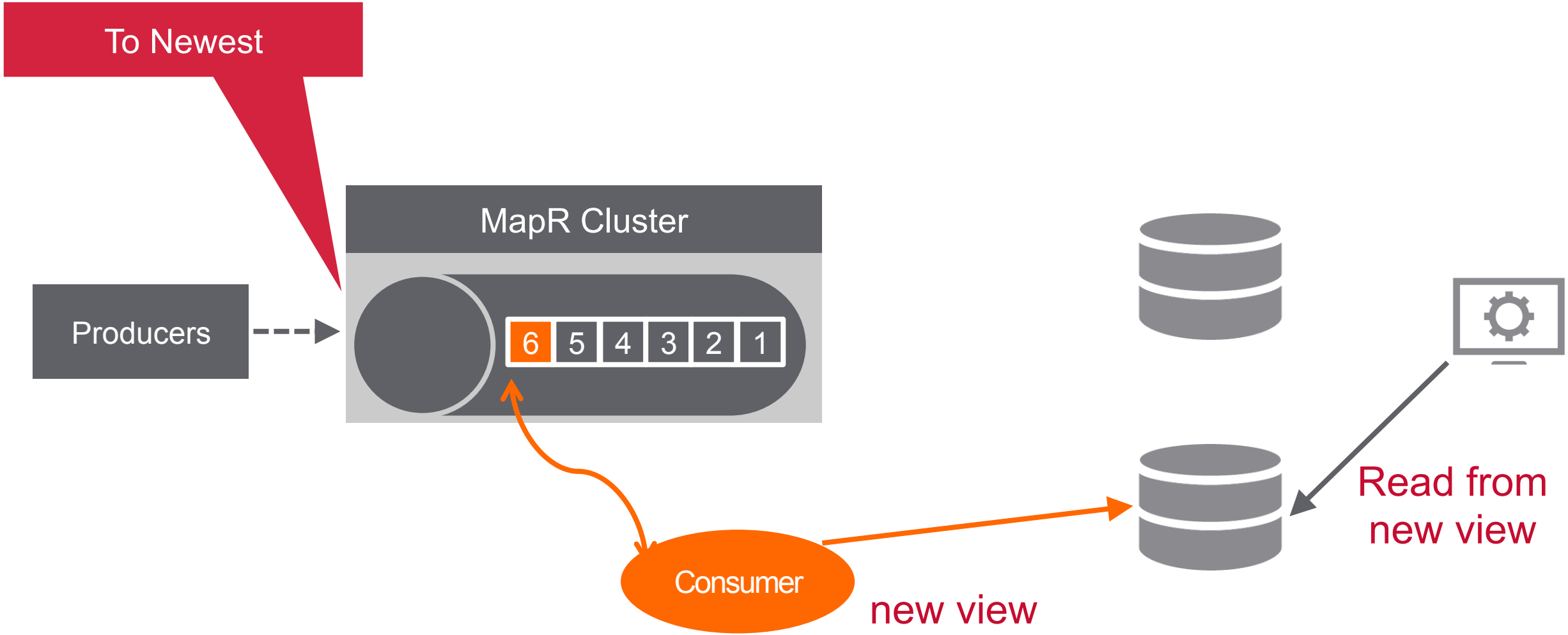
Duality of Streams and Tables

https://engineering.linkedin.com/distributed-systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying

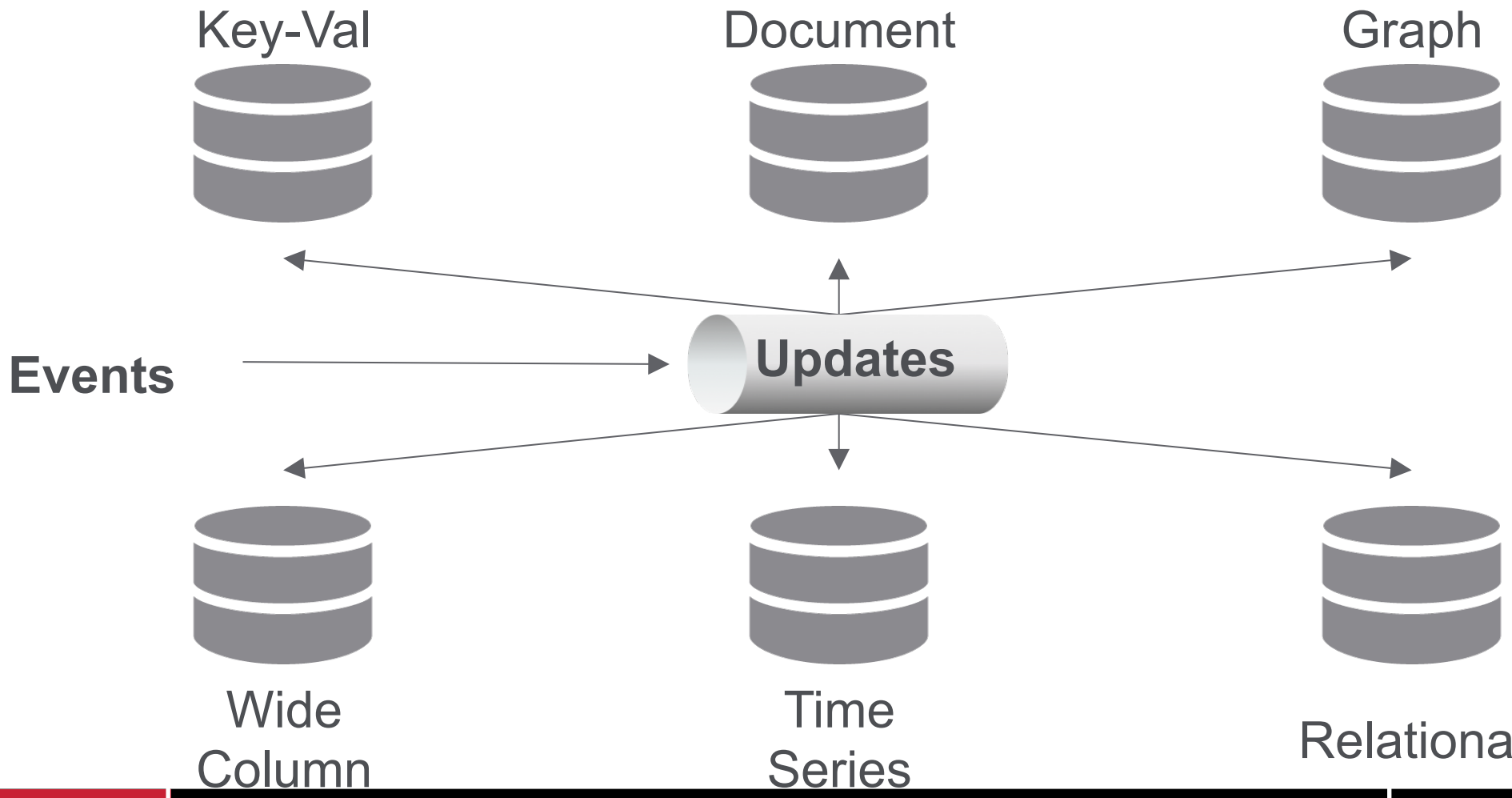
Rewind: Reprocessing Events



Rewind Reprocessing Events



Event Sourcing, Command Query Responsibility Separation:
Turning the Database Upside Down



Use Case: Streaming System of Record for Healthcare

Objective:

- Build a flexible, secure healthcare exchange

Challenges:

- Many different data models
- Security and privacy issues
- HIPAA compliance

What are the outcomes in the entire state on diabetes?

Clinical Data Viewer

Are there doctors that are doing this better than others?

Georgia Health Connect

Provider Organizations

Clinical Data

Financial Data

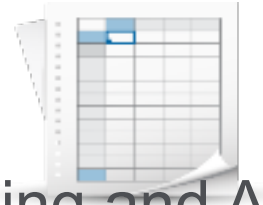
GaHC Georgia Health Connect POWERED BY LIAISON

ALLOY Health: Exchange

IHE

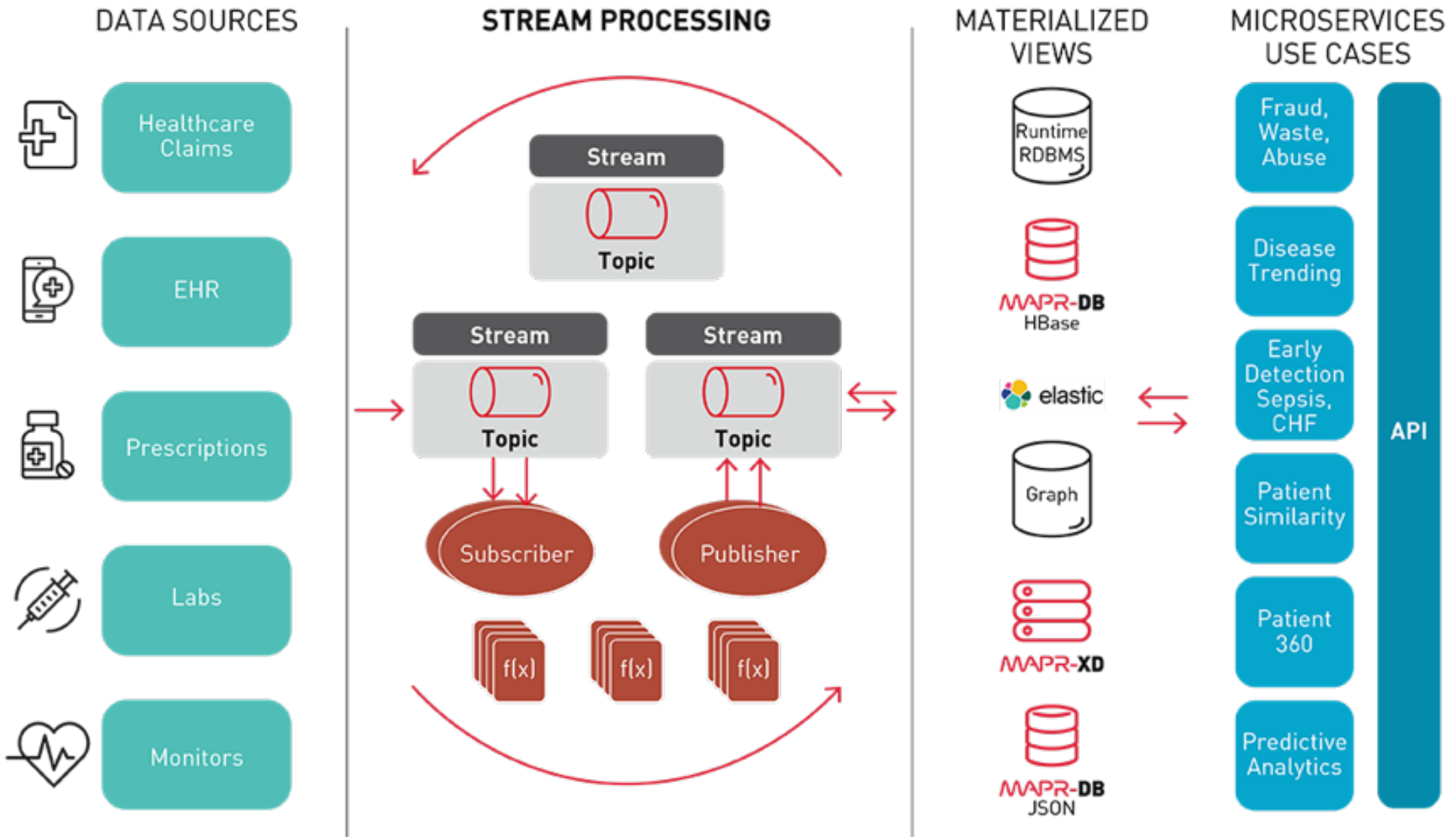
State HIE

GEORGIA HEALTH INFORMATION NETWORK



Reporting and Analytics

Use Case: Streaming System of Record for Healthcare



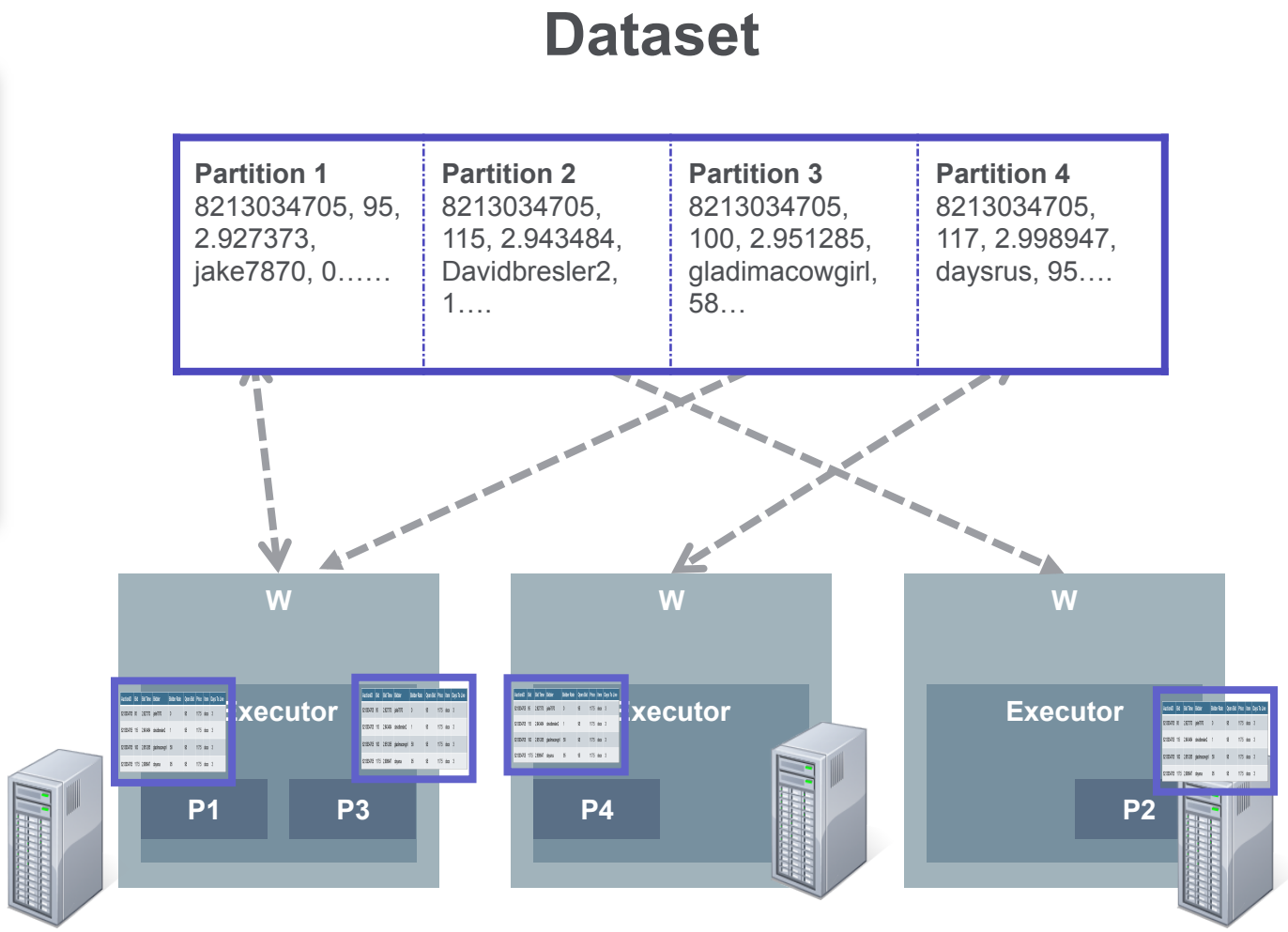
Spark Dataset

Spark Distributed Datasets

- Read only **collection** of **typed objects Dataset[T]**
- **Partitioned** across a cluster
- Operated on in **parallel**
- can be **Cached**

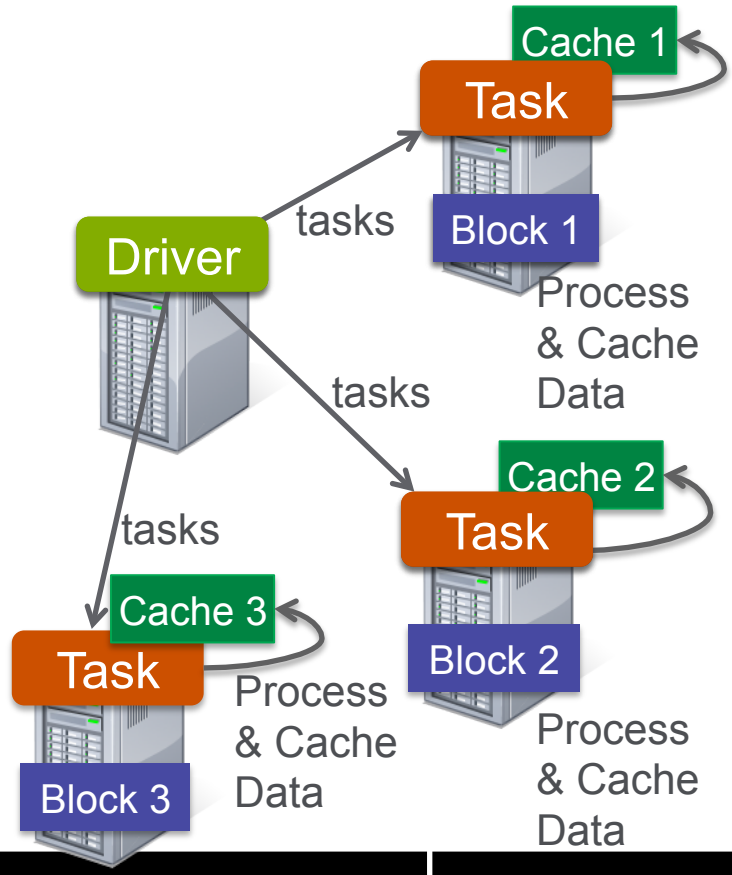
partitioned

AuctionID	Bid	Bid Time	Bidder	Bidder Rate	Open Bid	Price	Item	Days To Live
8213034705	95	2.927373	jake7870	0	95	117.5	xbox	3
8213034705	115	2.943484	davidbresler2	1	95	117.5	xbox	3
8213034705	100	2.951285	gladimacowgirl	58	95	117.5	xbox	3
8213034705	117.5	2.998947	daysrus	95	95	117.5	xbox	3



Spark Distributed Datasets read from a file

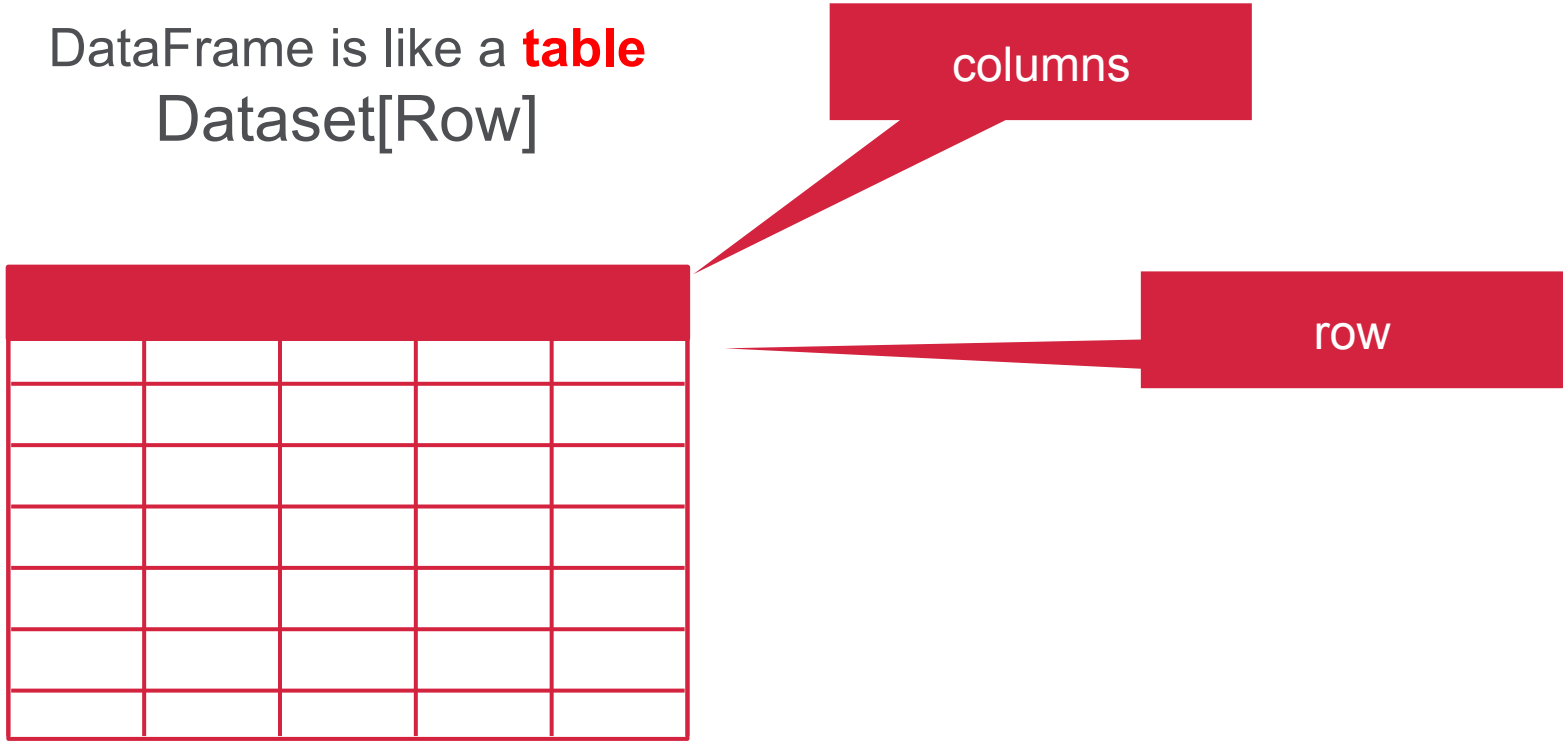
```
val df: Dataset[Payment] = spark.read.json(”/p/file.json").as[Payment]
```



DataFrame = Dataset[Row] can use Spark SQL

DataFrame is like a **table**

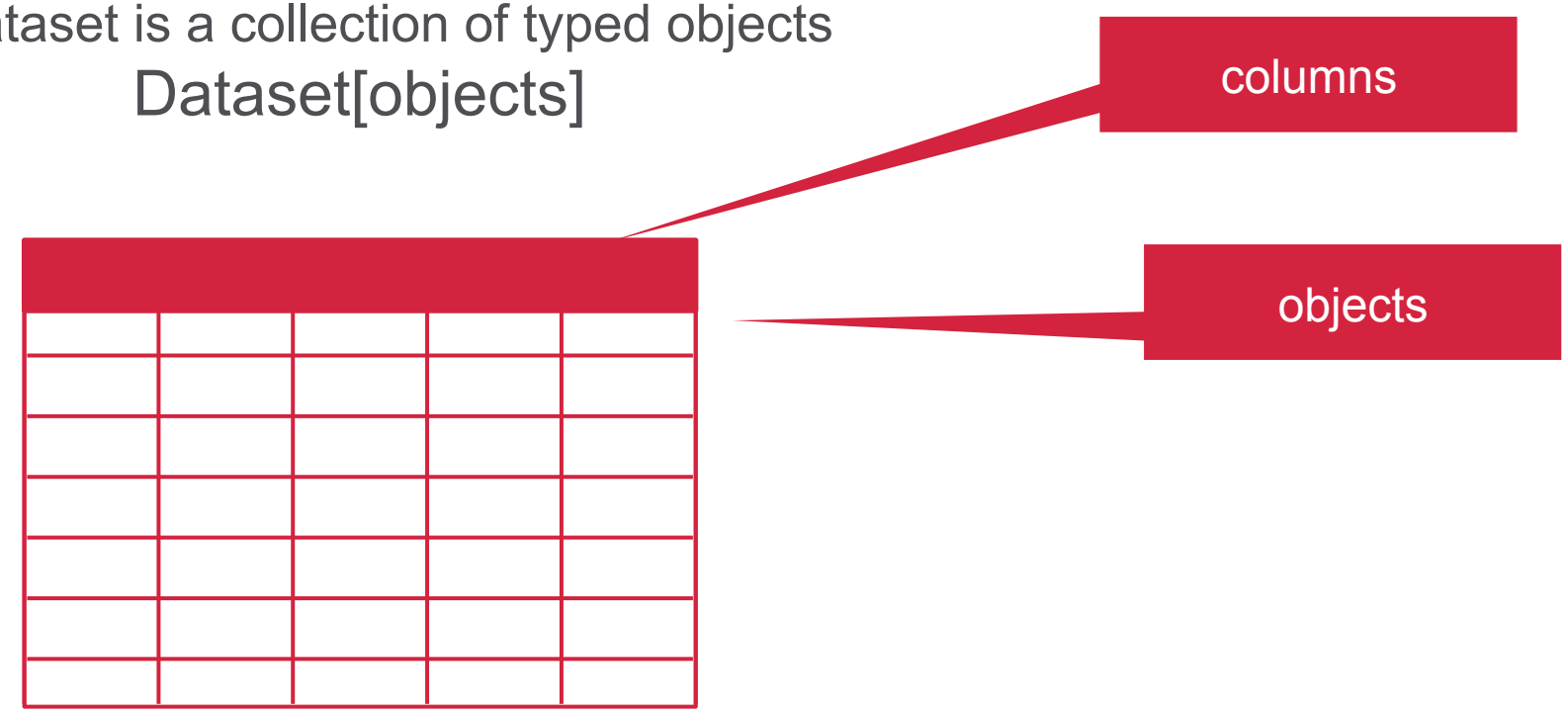
Dataset[Row]



Dataset[Typed Object] can use Spark SQL and Functions

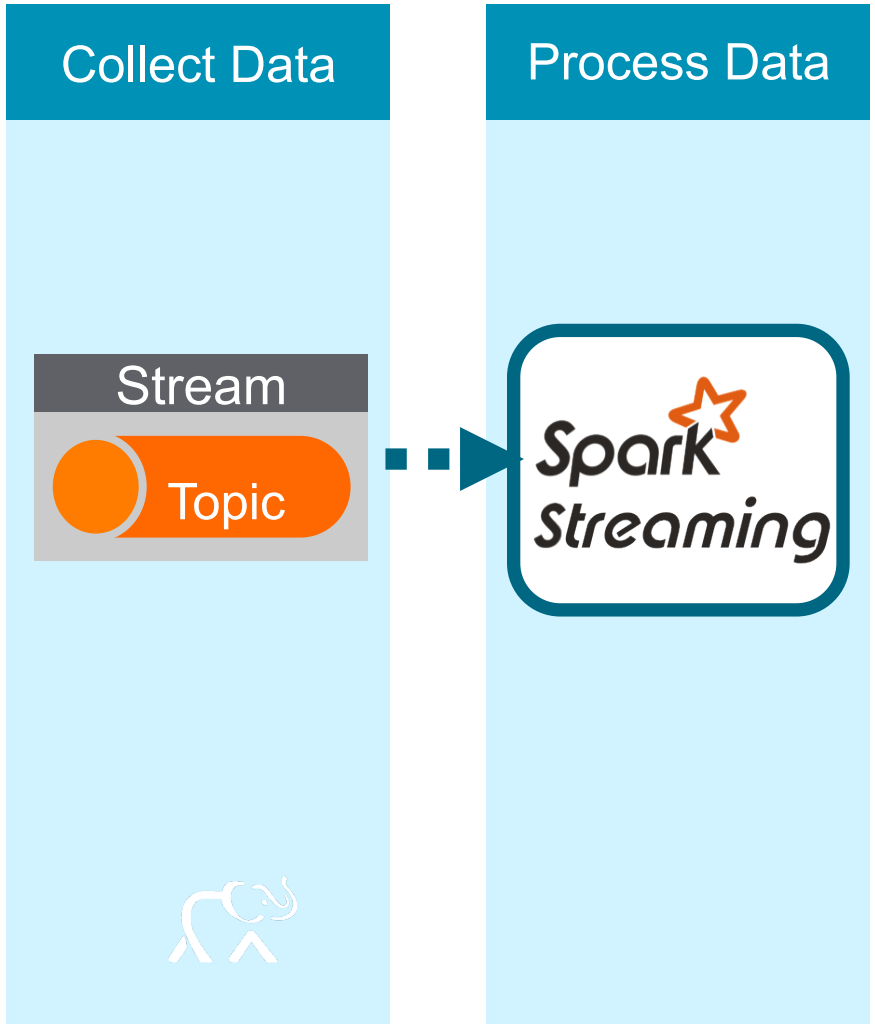
A Dataset is a collection of typed objects

Dataset[objects]



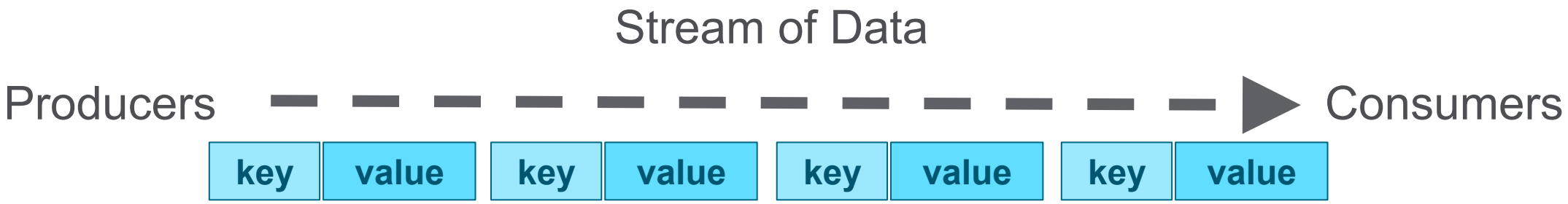
Spark Streaming

Process the Data with Spark Streaming



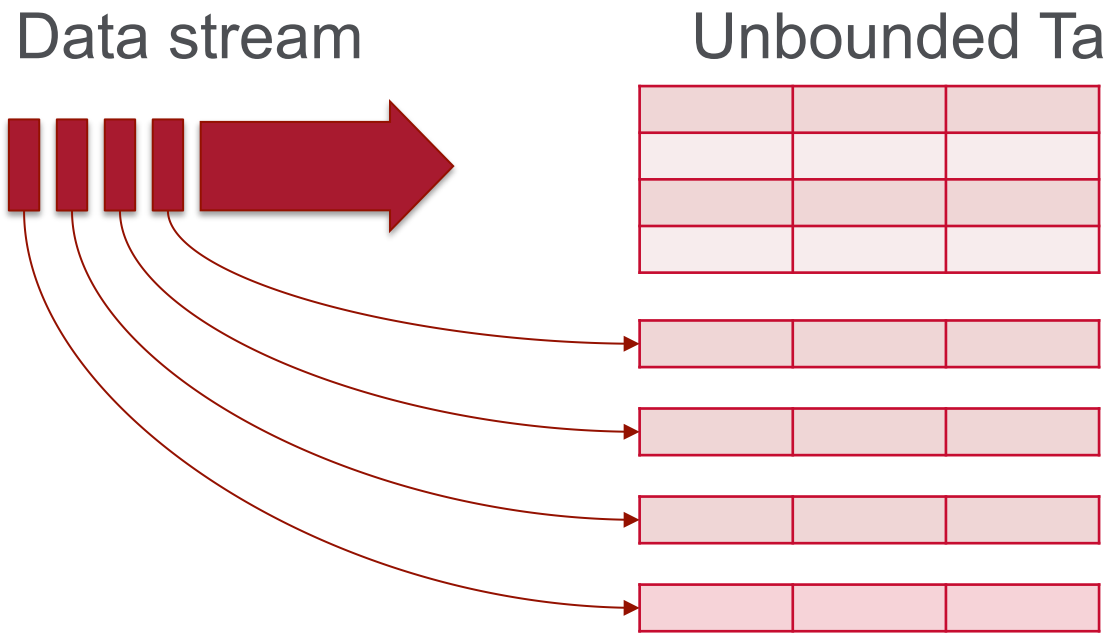
- scalable, high-throughput, stream processing of live data

What is a Stream ?



- A **stream** is an **continuous** sequence of events or records
- Records are key-value pairs

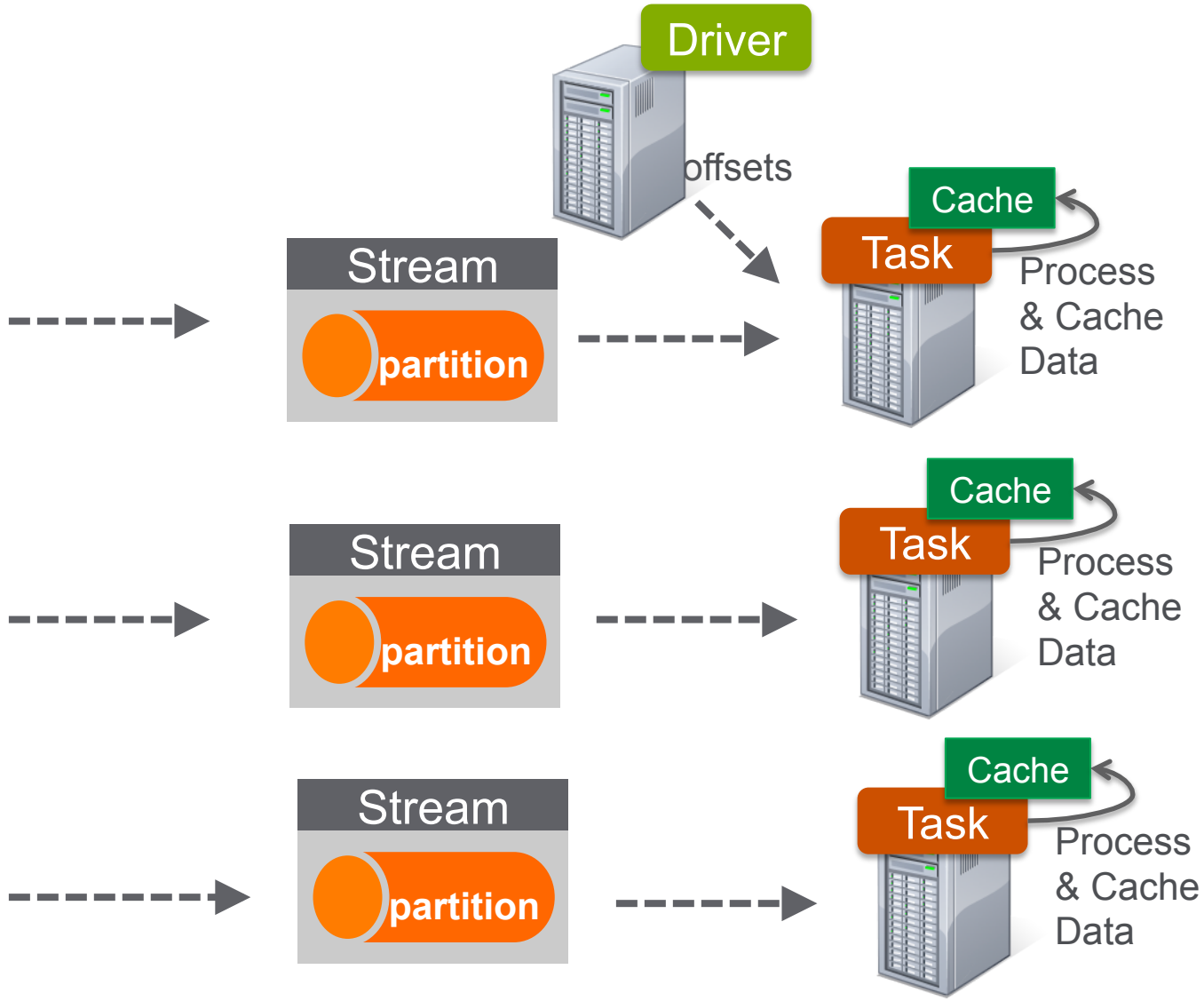
Treat Stream as Unbounded Tables



Data stream as an unbounded table

new data in the
data stream
=
new rows appended
to an unbounded table

Spark Distributed Datasets read from Stream partitions



Data is cached for **aggregations**
And windowed functions

Stream Processing on Spark SQL Engine

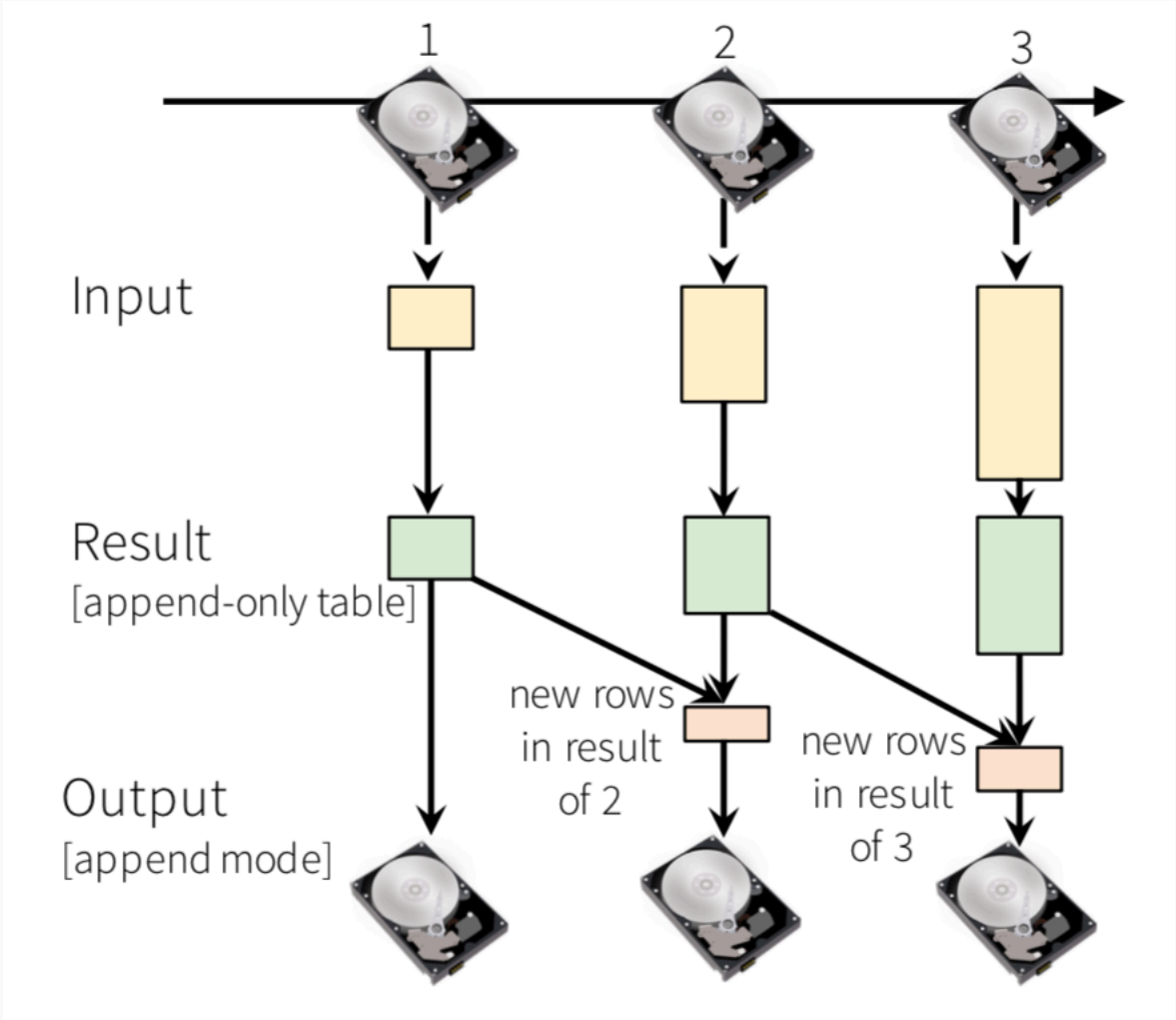
Static Data =
bounded table

Streaming data =
Unbounded table

Same Dataset operations & SQL

Conceptual model

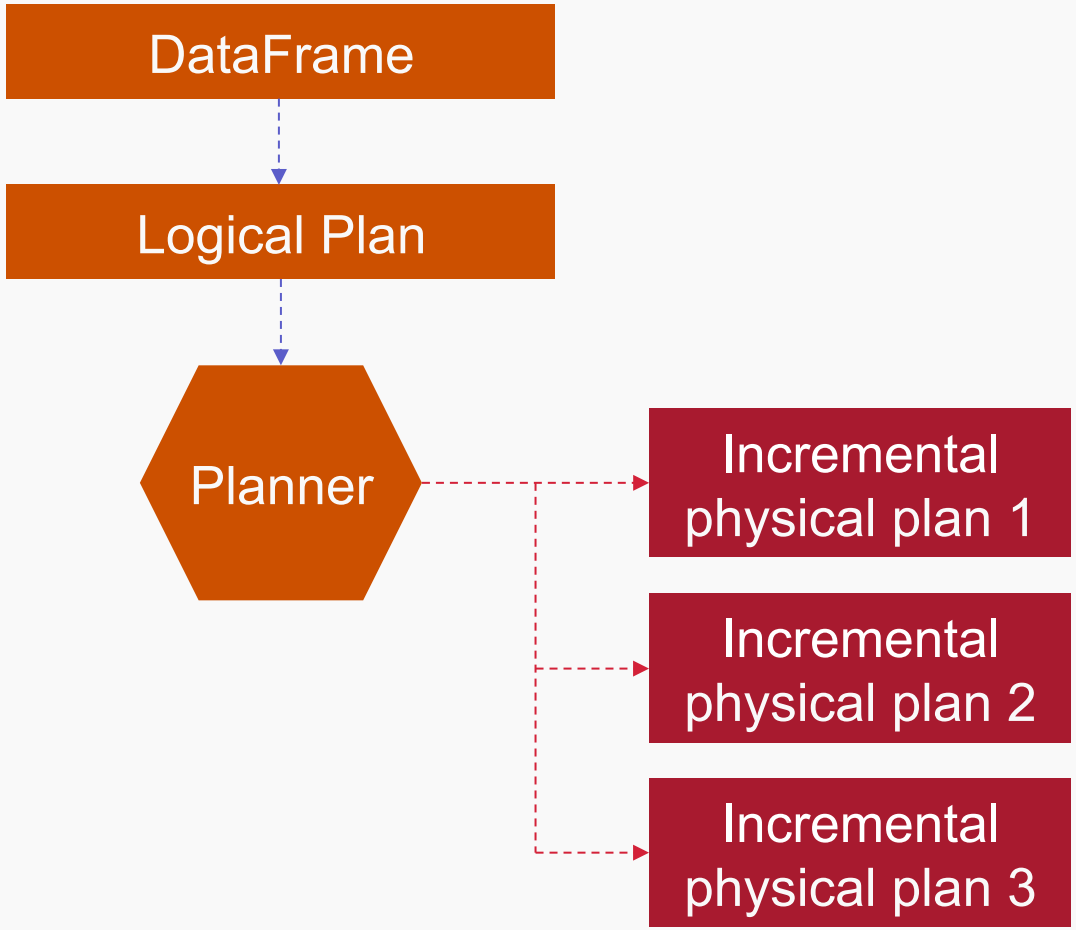
1. For new input
2. Apply **incremental query** changes
3. **Append** the result to Output



Continuous incremental execution

Spark SQL converts queries to incremental execution plans

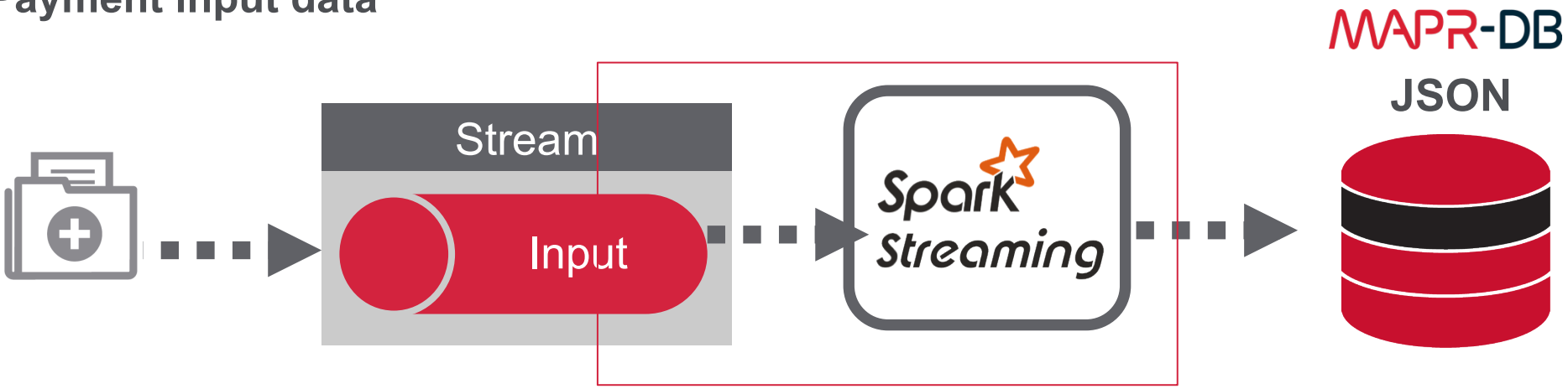
For input of data



Use Case: Payment Data

Payment input data

transform

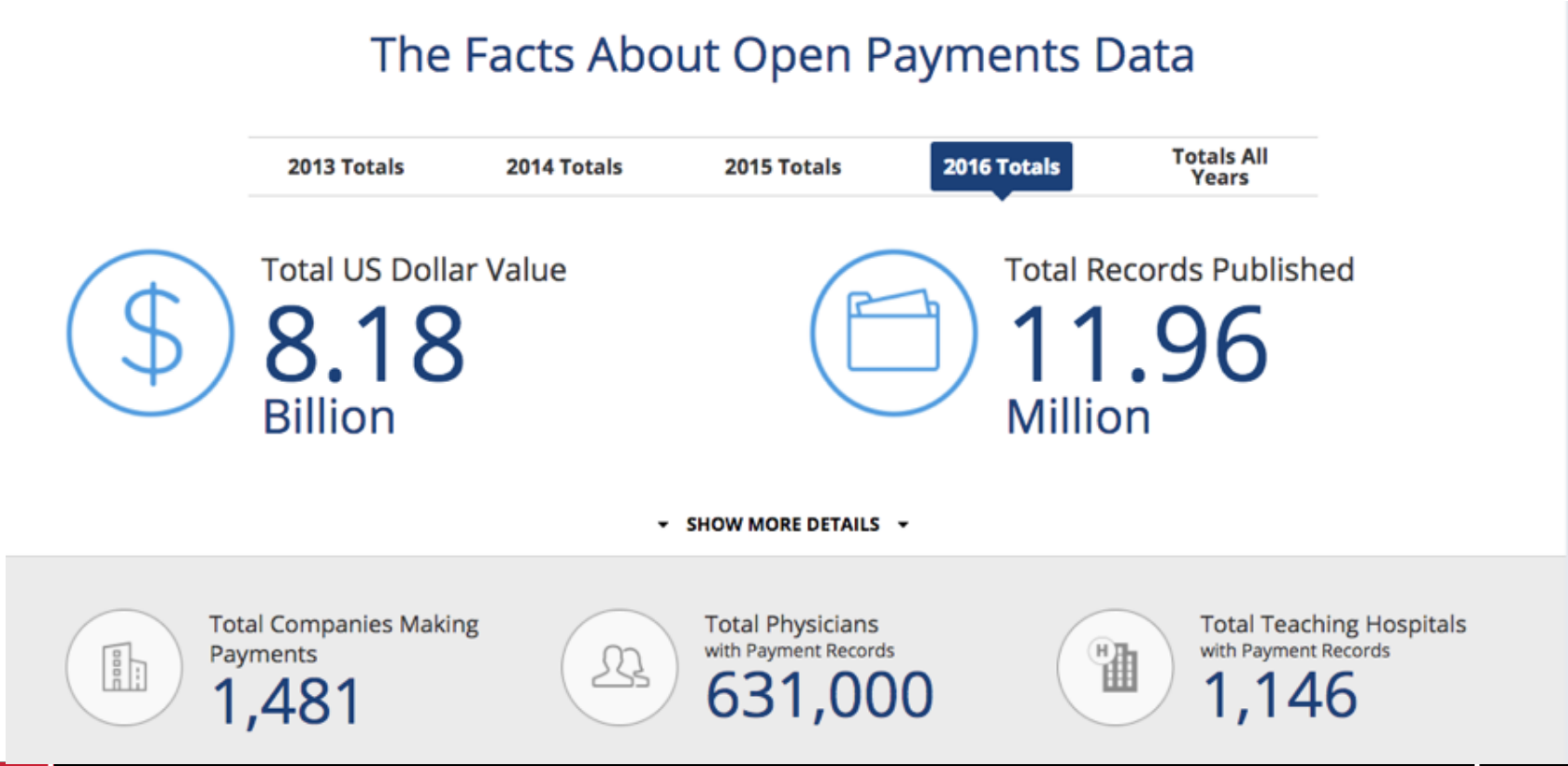


"NEW","Covered Recipient Physician",,,,"132655","GREGG","D","ALZATE",,"8745 AERO DRIVE","STE 200","SAN DIEGO","CA","92123","United States",,,"Medical Doctor","Allopathic & Osteopathic Physicians|Radiology|Diagnostic Radiology","CA",,,,,"DFINE, Inc","100000000326","DFINE, Inc","CA","United States",90.87,"02/12/2016","1","In-kind items and services","Food and Beverage",,,,"No","No Third Party Payment",,,,,"No","346039438","No","Yes","Covered","Device","Radiology","StabiliT",,"Covered","Device","Radiology","STAR Tumor Ablation System",,,,,,,,,,,,,,,,"2016","06/30/2017"


```
{
  "_id":"317150_08/26/2016_346122858",
  "physician_id":"317150",
  "date_payment":"08/26/2016",
  "record_id":"346122858",
  "payer":"Mission Pharmacal Company",
  "amount":9.23,
  "Physician_Specialty":"Obstetrics & Gynecology",
  "Nature_of_payment":"Food and Beverage"
}
```

Use Case: Open Payment Dataset

- Payments Drug and Device companies make to
- Physicians and Teaching Hospitals for
- Travel, Research, Gifts, Speaking fees, and Meals



Scenario: Payment Data

Provider ID	Date	Payer	Payer State	Provider Specialty	Provider State	Amount	Payment Nature
1261770	01/11/2016	Southern Anesthesia & Surgical, Inc	CO	Oral and Maxillofacial Surgery	CA	117.5	Food and Beverage

Stream the data into a Dataframe: Define the Schema

```
case class Payment(physician_id: String,
 date_payment: String, payer: String, payer_state: String
 amount: Double, physician_specialty: String,
 phys_state: String, nature_of_payment:String)
```

```
val schema = StructType(Array(
   StructField("_id", StringType, true),
   StructField("physician_id", StringType, true),
   StructField("date_payment", StringType, true),
   StructField("payer", StringType, true),
   StructField("payer_state", StringType, true),
   StructField("amount", DoubleType, true),
   StructField("physician_specialty", StringType, true),
   StructField("physician_type", StringType, true),
   StructField("physician_state", StringType, true),
   StructField("nature_of_payment", StringType, true)
 ))
```

Function to Parse CSV into Payment Class

```
def parse(str: String): Payment = {
    val td = str.split(",(?=([^\\\"]*\\\"[^\\\"]*\\\")*[^\\\"]*$)")
    val physician_id = td(5)
    val payer = td(27)
    . . .
    val physician_state = td(20)
    var focus =td(19)
    val id =physician_state+'_'+focus+ '_'+ date_payment+'_'
        + record_id
    Payment(id, physician_id, date_payment, payer, payer_state,
     amount, physician_type, focus, physician_state,
     nature_of_payment)
```

Parsed and Transformed Payment Data

Example Dataset Row


```
{
  "_id":”TX_Gynecology_08/26/2016_346122858",
  "physician_id":"317150",
  "date_payment":"08/26/2016",
  "payer":"Mission Pharmacal Company",
  "payer_state":”CO",
  "amount":9.23,
  ”physician_specialty":”Gynecology",
  “physician_state":”TX"
  ”nature_of_payment":"Food and Beverage"
}
```

Streaming pipeline Data source

```
val df1 = spark.readStream.format("kafka")
 .option("kafka.boostrap.servers",...)
 .option("subscribe", "topic")
 .load()
```

Specify data source

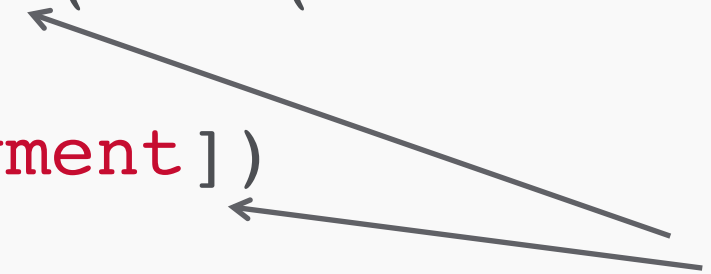
returns a dataframe

Kafka DataFrame

key	value	topic	partition	offset	timestamp
[binary]	*[binary]*	"topic"	0	345	1486087873
[binary]	*[binary]*	"topic"	3	2890	1486086721

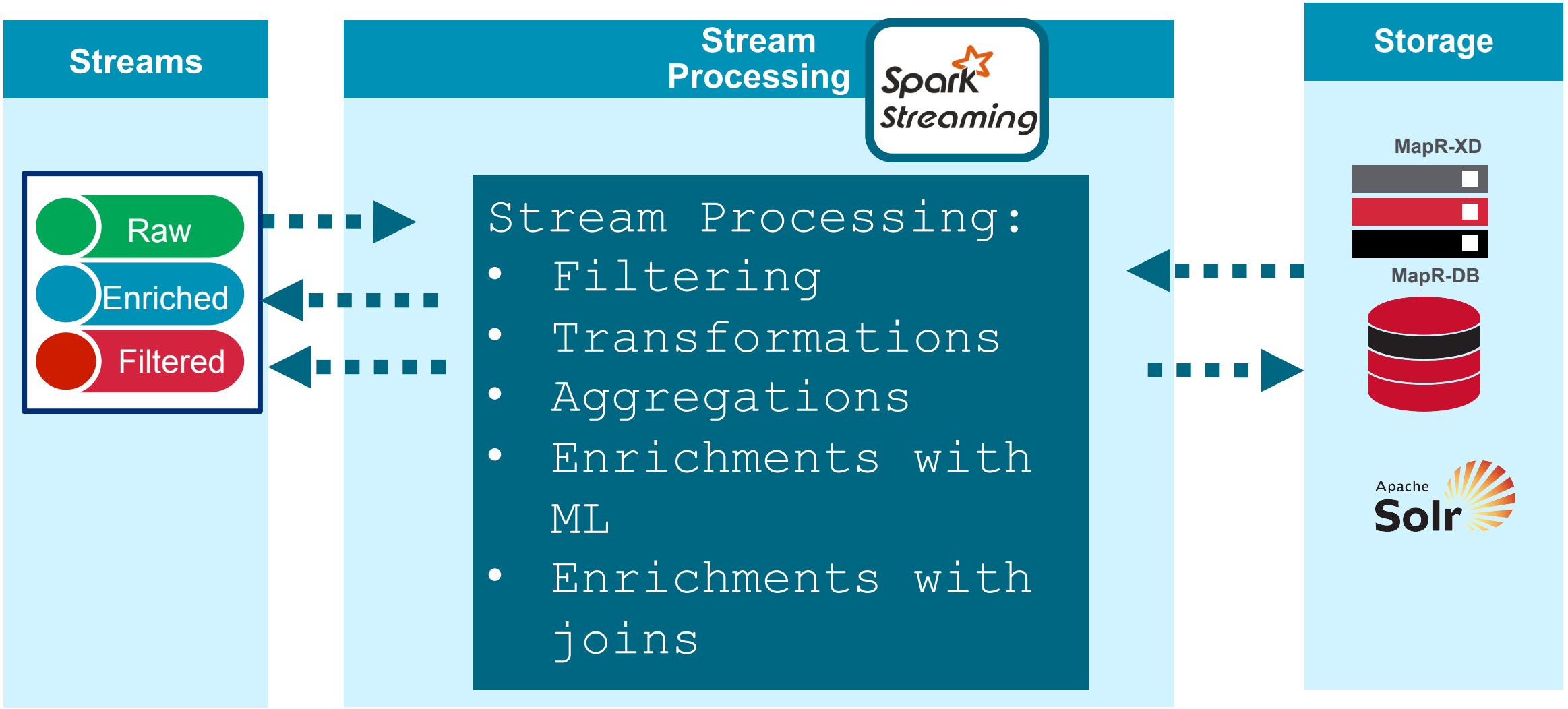
Transformation

```
spark.udf.register("deserialize",
  (message: String) => parse(message))

val df2=df1
  .selectExpr("""deserialize(CAST(value as STRING))
   AS message""")
  .select($"message".as[Payment])
```



Cast bytes from Kafka records to a string, parse csv , and return Dataset[Payment]

Stream Processing



Dataframe Integrated Queries

L I Query	Description
`agg(expr, exprs)`	Aggregates on entire DataFrame
`distinct`	Returns new DataFrame with unique rows
`except(other)`	Returns new DataFrame with rows from this DataFrame not in other DataFrame
`filter(expr); where(condition)`	Filter based on the SQL expression or condition
`groupBy(cols: Columns)`	Groups DataFrame using specified columns
`join (DataFrame, joinExpr)`	Joins with another DataFrame using given join expression
`sort(sortcol)`	Returns new DataFrame sorted by specified column
`select(col)`	Selects set of columns

Continuous aggregations

```
val d3=df2.avg(“amount")
```

Continuously compute **average** payment amount

```
spark.sql("SELECT avg(amount)  FROM payment2").show()

+------------------+
|       avg(amount)|
+------------------+
|662.5487485242039|
+------------------+
```

Continuous aggregations and filter

```
val d3=df2.groupBy(“payer")
  .avg(“amount")
```

Continuously compute **average** payment amount **by payer**

```
spark.sql("SELECT payer, avg(amount)  FROM payment2 group by payer ").show()
```

```
+--------------------+------------------+
|               payer|       avg(amount)|
+--------------------+------------------+
|    Ciel Medical Inc| 4153.666666666667|
|Southern Anesthes...|             670.0|
|HF Acquisition Co...|1708.3333333333333|
|Mission Pharmacal...|11.995000000000001|
|          DFINE, Inc|211.72225728155328|
|Affordable Pharma...|44246.082500000004|
+--------------------+------------------+
```

Continuous aggregations and filter

```
val d3=df2
      .filter($"amount" > 20000)
```

Continuously **filter** large payment amount

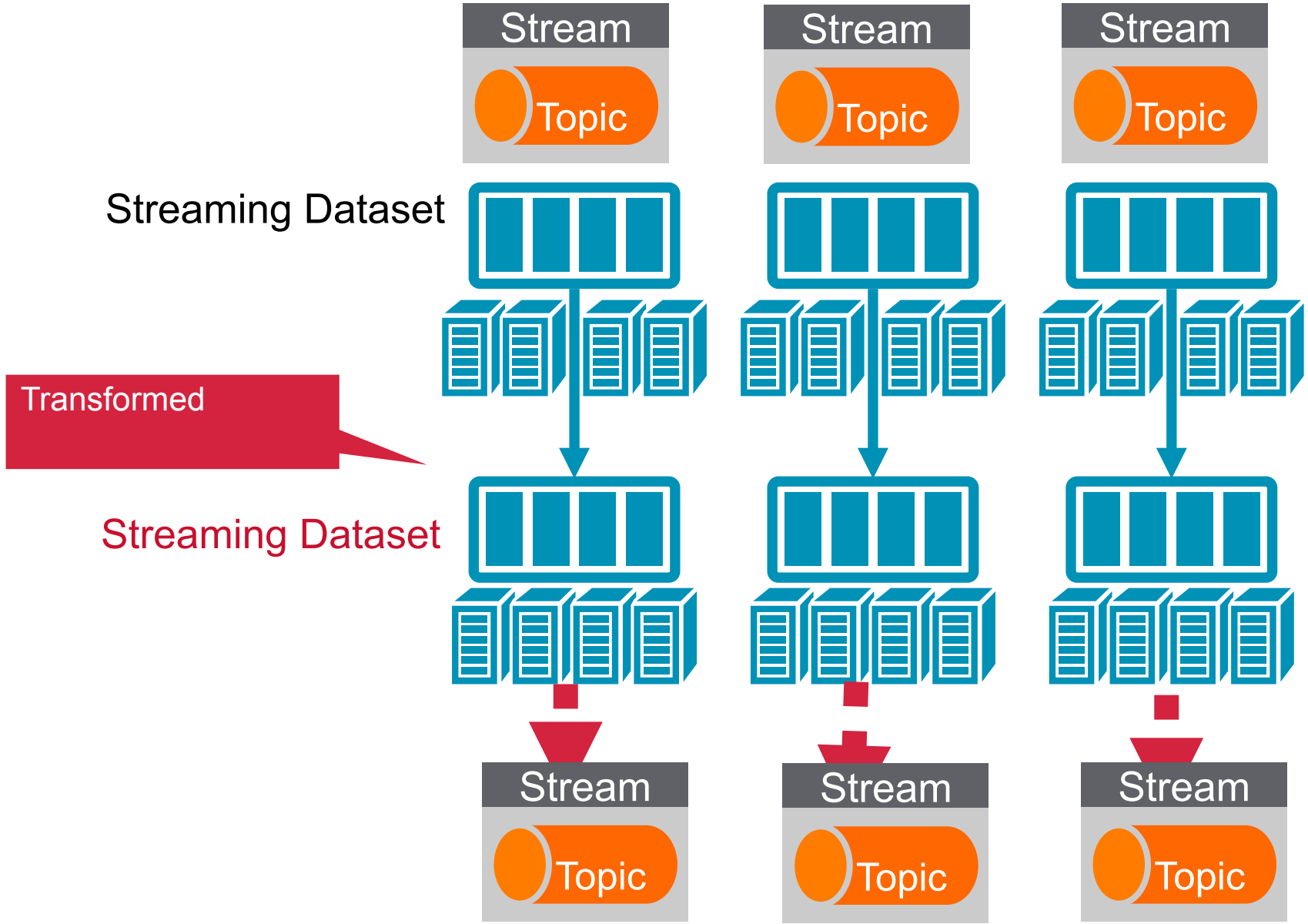
Streaming pipeline Kafka topic Data Sink

```
val query = df3.write
    .format("kafka")
    .option("kafka.bootstrap.servers", "host1:port1,host2:port2")
    .option("topic", "/apps/uberstream:uberp")

query.start.awaitTermination()
```

Write results to Kafka topic

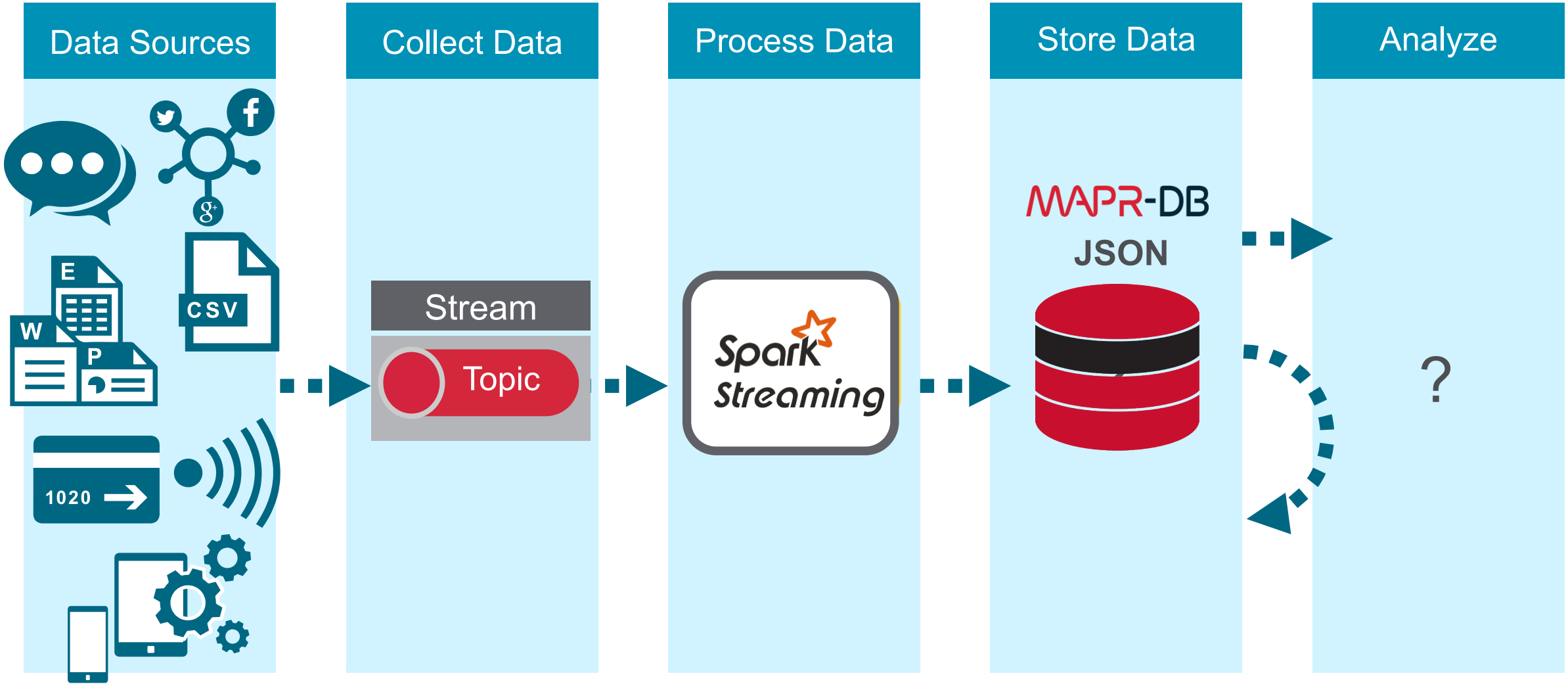
Start running the query

Streaming Applicaton

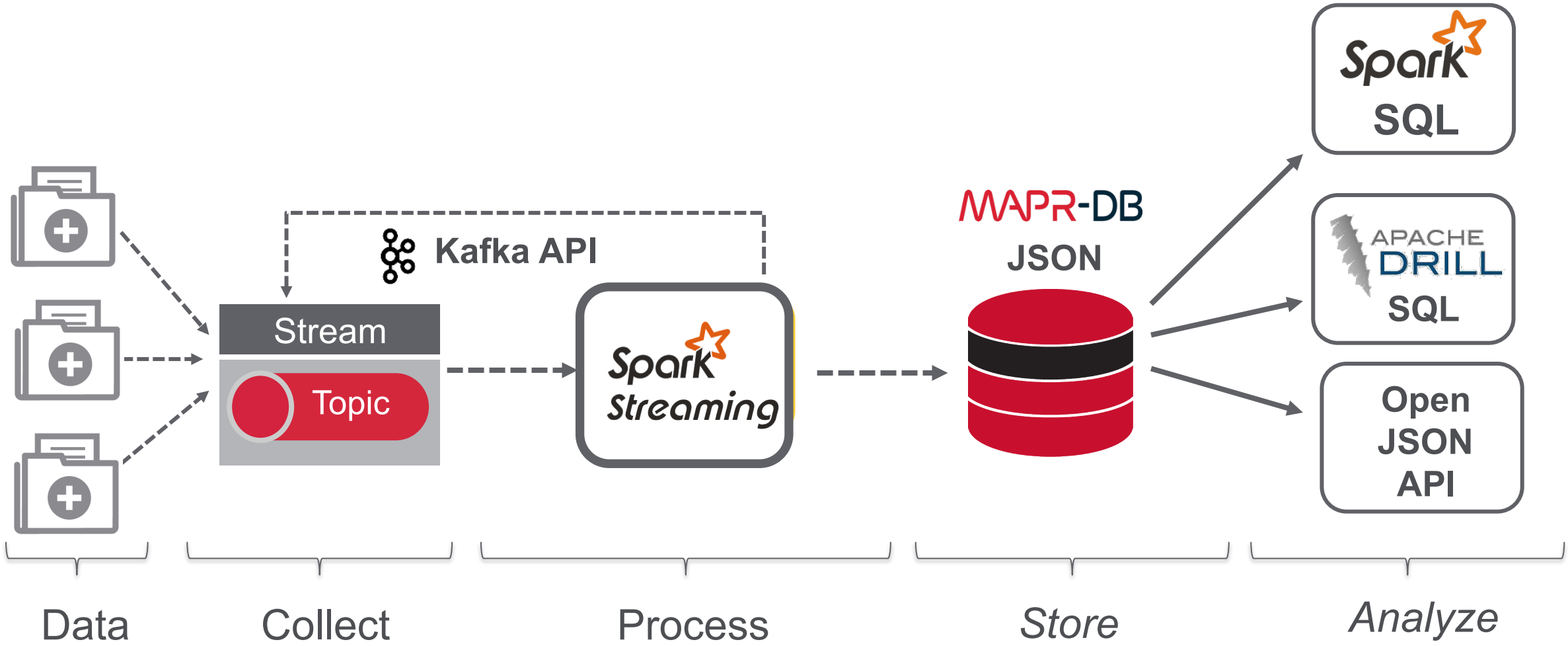


Spark & MapR-DB

What Do We Need to Do ?



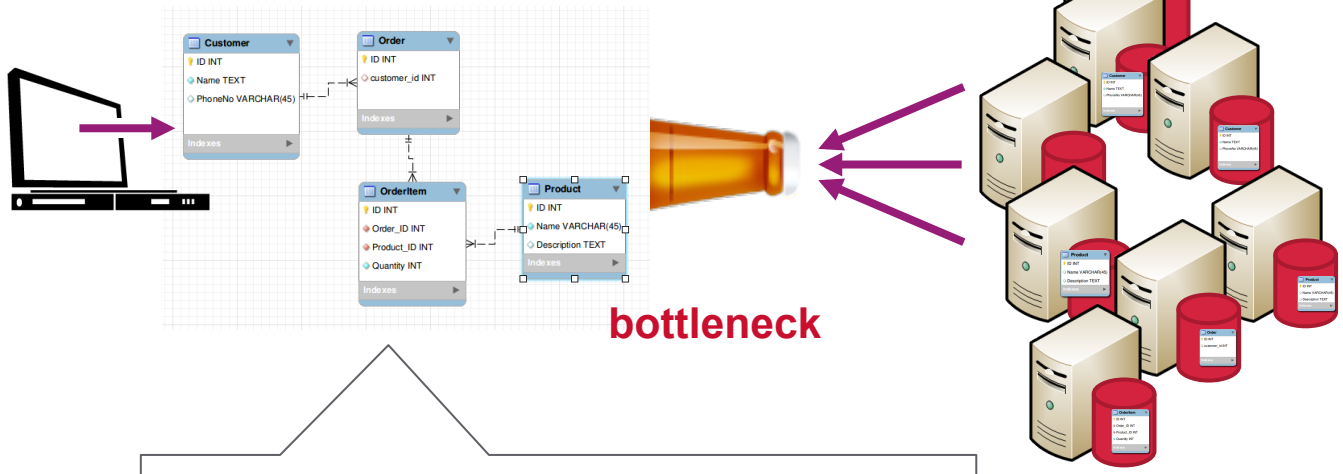
Stream Processing Pipeline



Where/How to store data ?

Relational Database vs. MapR-DB

RDBMS



Normalized schema → Joins for queries can cause bottleneck

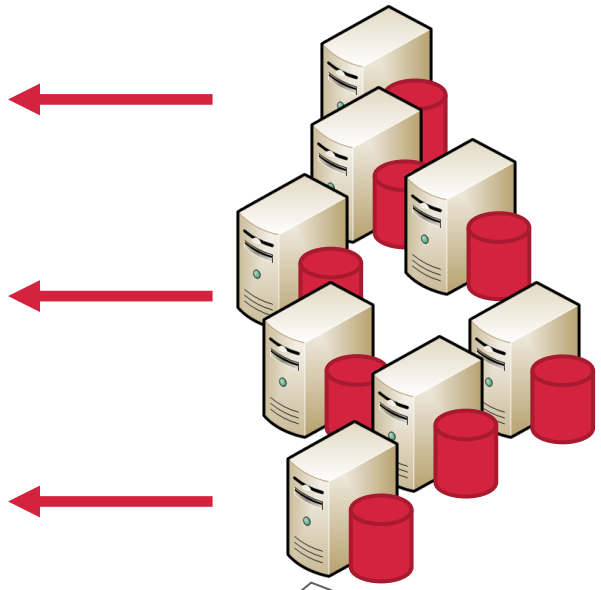
Storage Model

MapR-DB

Key	colB	colC
xxx	val	val
xxx	val	val

Key	colB	colC
xxx	val	val
xxx	val	val

Key	colB	colC
xxx	val	val
xxx	val	val



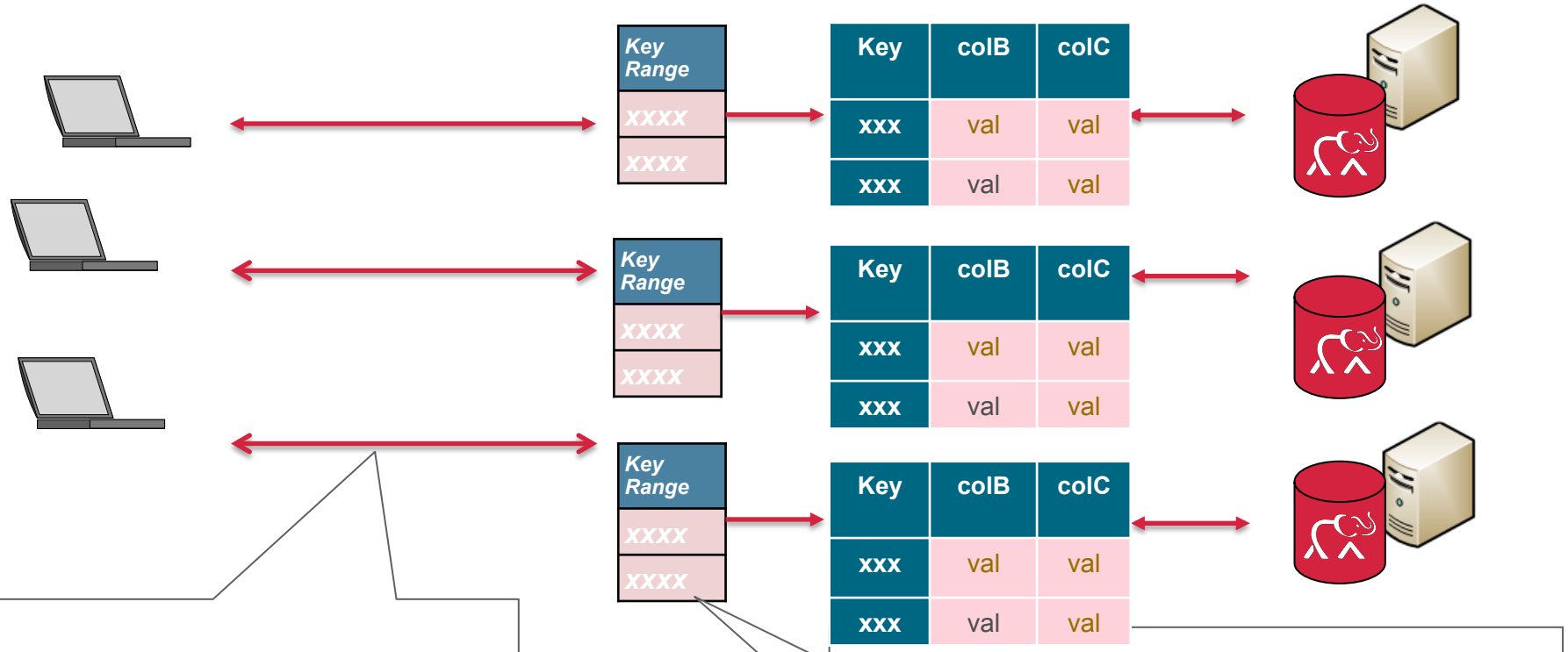
De-Normalized schema → Data that is read together is stored together

MapR-DB JSON Document Store

Data that is read together is stored together

Row Keys: identify the rows in an MapR-DB table.

Designed for Partitioning and Scaling



Fast Reads and Writes by Key

Data is automatically partitioned by Key Range

MapR-DB JSON Document Store

Data is automatically partitioned by Key Range

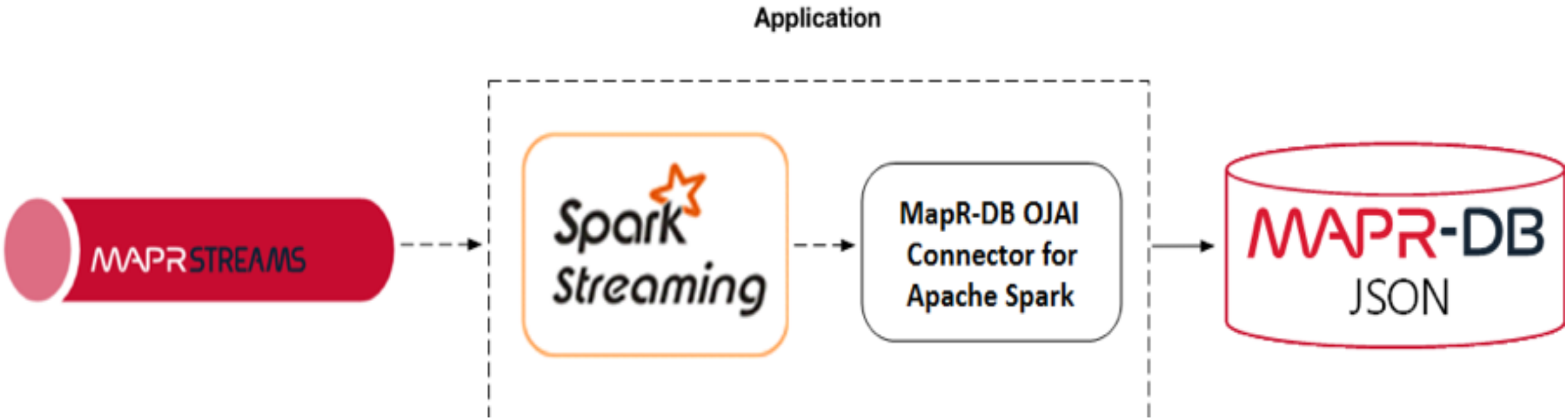
Row Keys: identify the rows in an MapR-DB table.

Payment Data

Automatically sorted and partitioned by Key Range (_id)

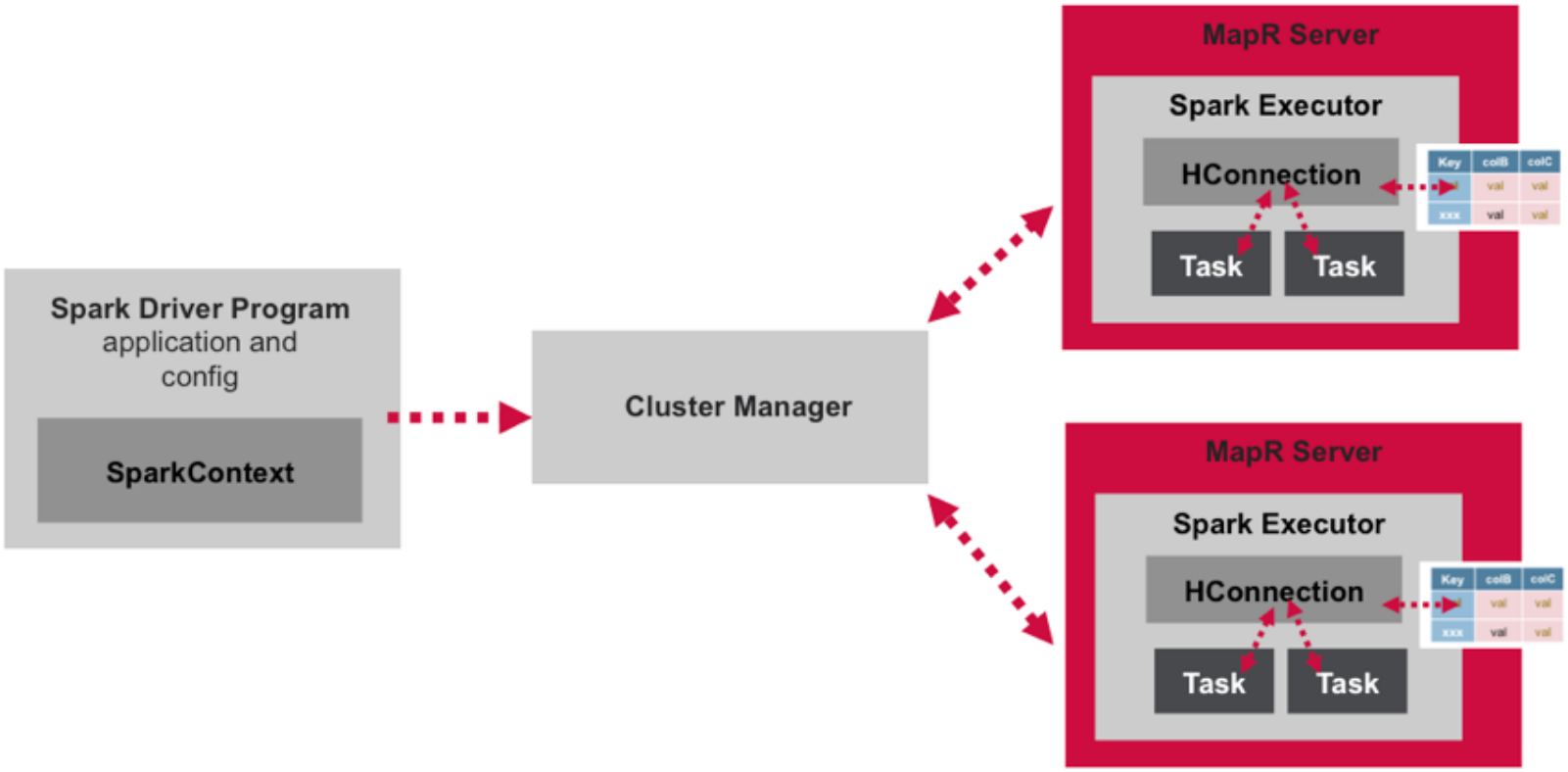

```
{
  "_id":"TX_Gynecology_08/26/2016_346122858",
  "physician_id":"317150",
  "date_payment":"08/26/2016",
  "payer":"Mission Pharmacal Company",
  "payer_state":"CO",
  "amount":9.23,
  "physician_specialty":"Gynecology",
  "physician_state":"TX"
  "nature_of_payment":"Food and Beverage"
}
```

Spark Streaming writing to MapR-DB JSON

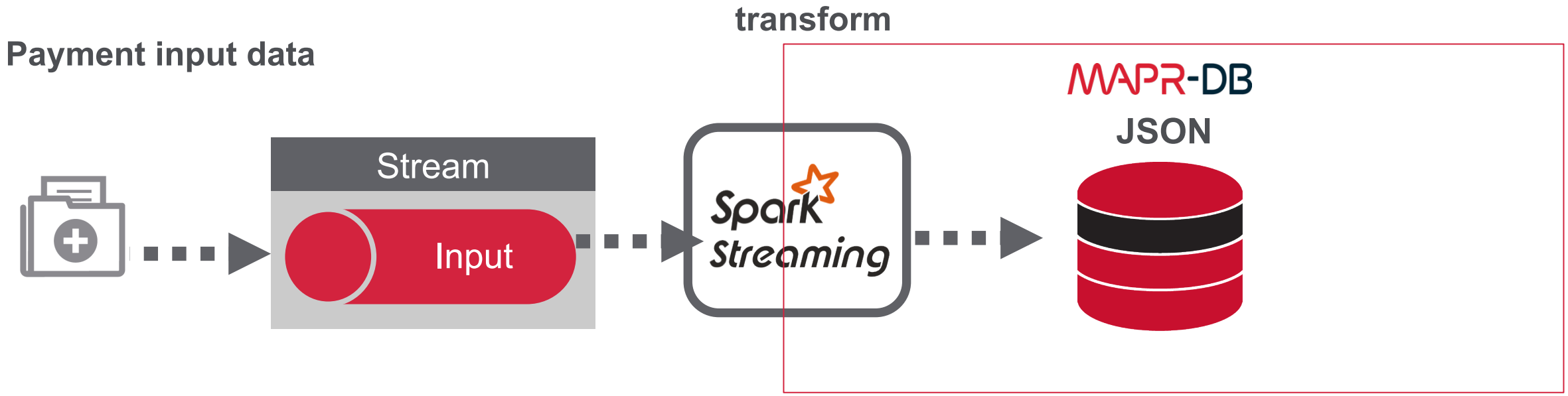


Spark MapR-DB Connector

- Connection object in every Spark Executor:
- **distributed parallel** writes & reads



Use Case: Flight Delays

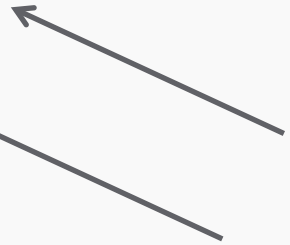


"NEW","Covered Recipient Physician",,,,"132655","GREGG","D","ALZATE",,"8745 AERO DRIVE","STE 200","SAN DIEGO","CA","92123","United States",,,"Medical Doctor","Allopathic & Osteopathic Physicians|Radiology|Diagnostic Radiology","CA",,,,,"DFINE, Inc","100000000326","DFINE, Inc","CA","United States",90.87,"02/12/2016","1","In-kind items and services","Food and Beverage",,,,"No","No Third Party Payment",,,,,"No","346039438","No","Yes","Covered","Device","Radiology","StabiliT",,"Covered","Device","Radiology","STAR Tumor Ablation System",,,,,,,,,,,,,,,,"2016","06/30/2017"


```
{
  "_id":"317150_08/26/2016_346122858",
  "physician_id":"317150",
  "date_payment":"08/26/2016",
  "record_id":"346122858",
  "payer":"Mission Pharmacal Company",
  "amount":9.23,
  "Physician_Specialty":"Obstetrics & Gynecology",
  "Nature_of_payment":"Food and Beverage"
}
```

Streaming pipeline Data Sink

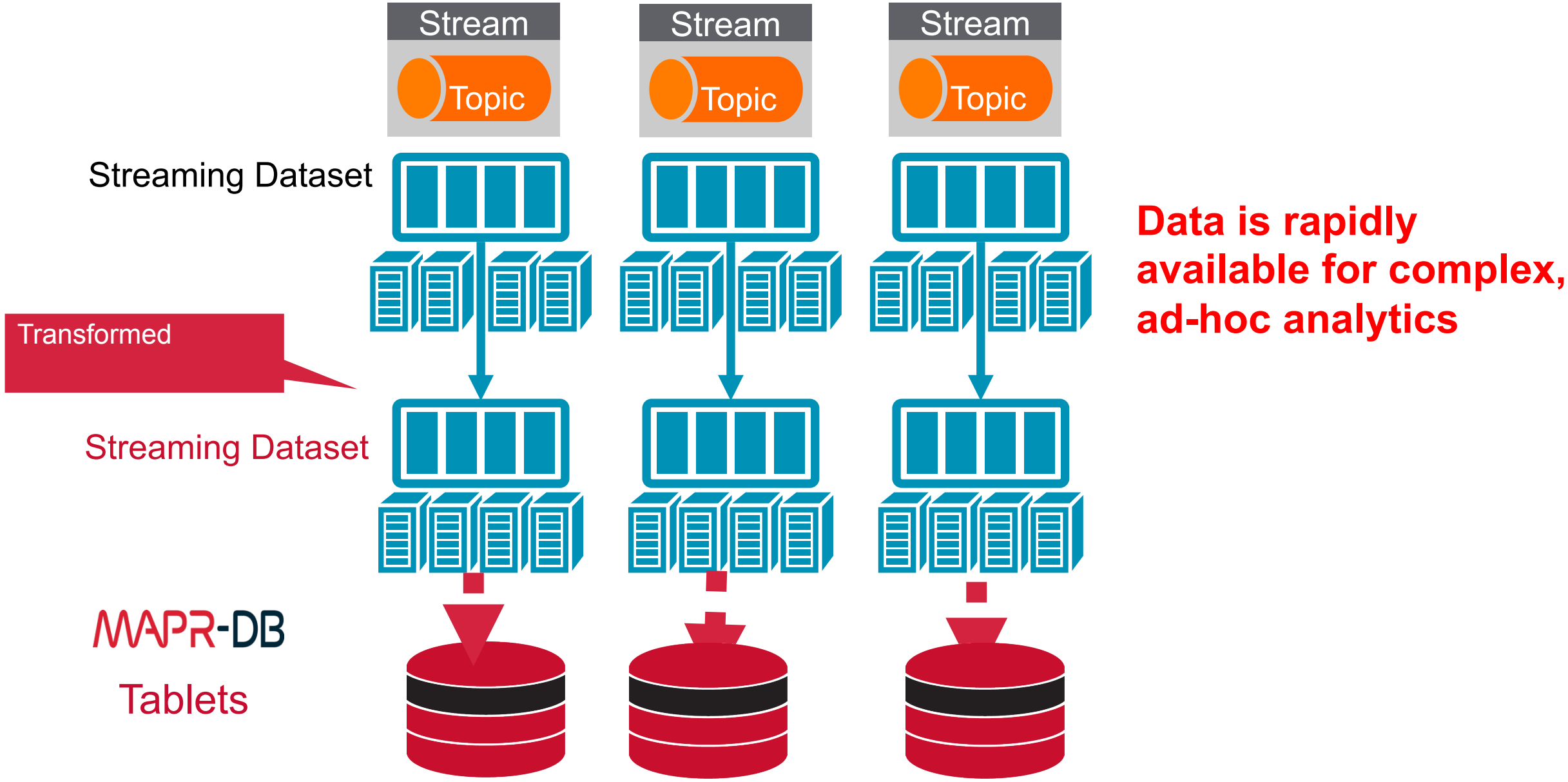
```
val query = df2.writeStream
      .format(MapRDBSourceConfig.Format)
      .option(MapRDBSourceConfig.TablePathOption, "/apps/paytable")
      .option(MapRDBSourceConfig.CreateTableOption, false)
      .option(MapRDBSourceConfig.IdFieldPathOption, "value")
      .outputMode(”append")

query.start().awaitTermination()
```



Write result to maprdb

Start running the query

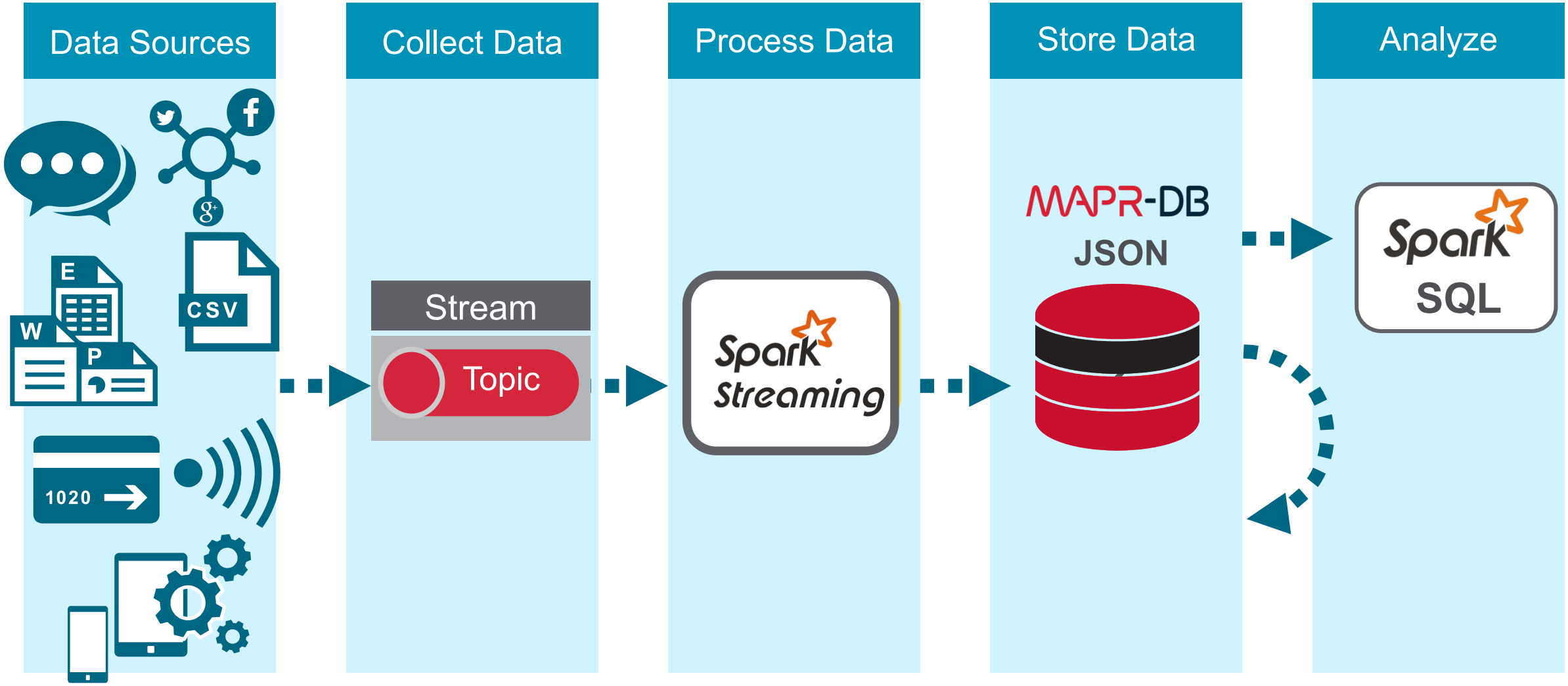
Streaming Applicaton



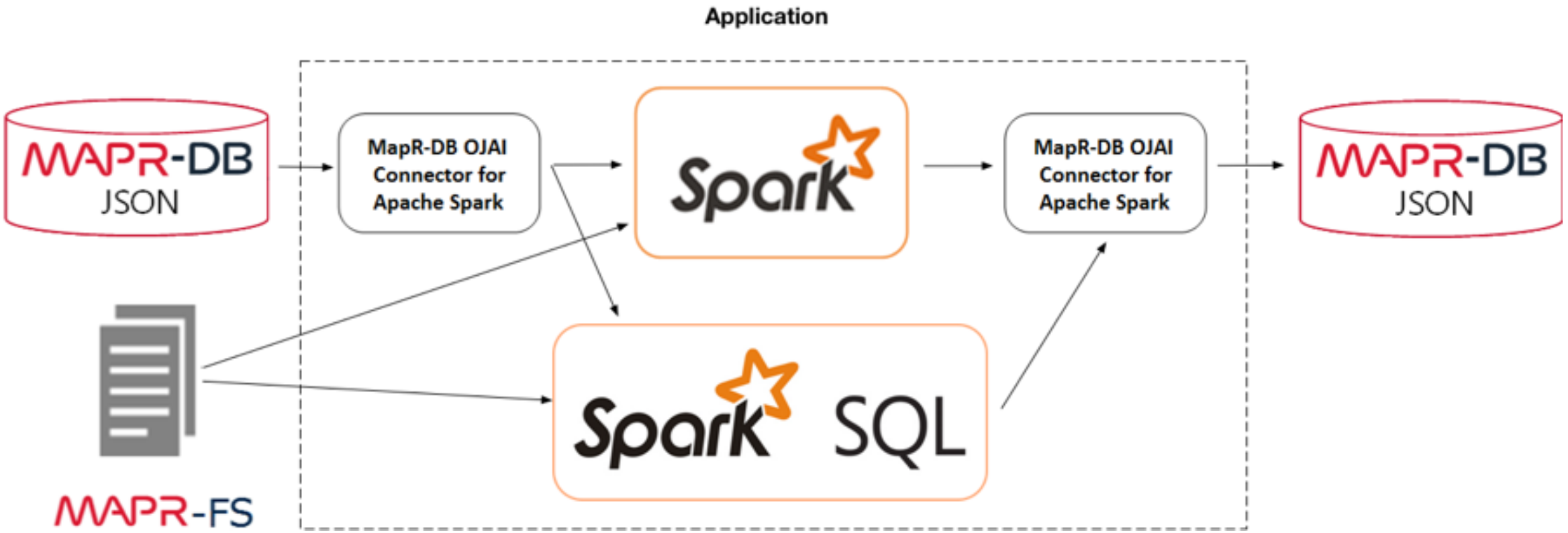
Data is rapidly available for complex, ad-hoc analytics

Explore the Data With Spark SQL

What Do We Need to Do ?



Spark SQL Querying MapR-DB JSON



Load the data into a Dataframe

Load data → **Data Frame**

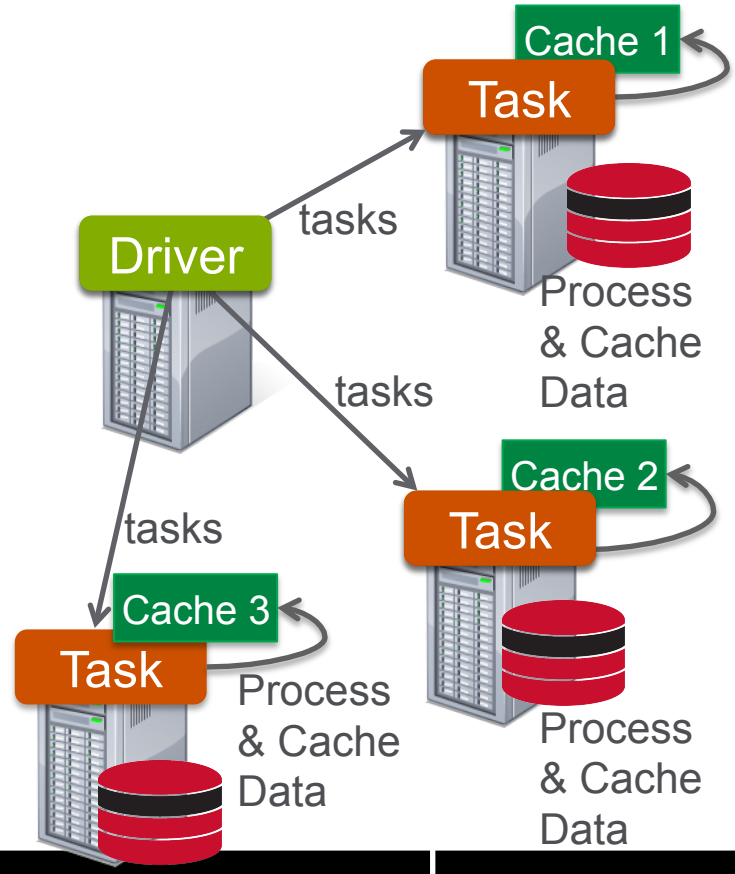
```
val pdf: Dataset[Payment] =
      spark.loadFromMapRDB[Payment]("/apps/paytable", schema)
      .as[Payment]

pdf.select("_id", "payer", "amount").show
```

_id	payer	amount
AK_Anesthesiology_04/01/2016_346088758	Mission Pharmacal Company	19.58
AK_Gastroenterology_03/15/2016_346146914	Braintree Laboratories, Inc.	13.33
AK_General Practice_10/13/2016_346134424	Mission Pharmacal Company	21.6
AK_Gynecology_12/05/2016_346591748	Bovie Medical Corporation	92.71
AK_Obstetrics & Gynecology_12/05/2016_346591744	Bovie Medical Corporation	92.71

Spark Distributed Datasets read from MapR-DB Partitions

```
val pdf: Dataset[Payment] =
      spark.loadFromMapRDB[Payment]("/apps/paytable", schema)
      .as[Payment]
```

_id
AK_Anesthesiology_04/01/2016_346088758
AK_Gastroenterology_03/15/2016_346146914
AK_General Practice_10/13/2016_346134424
AK_Gynecology_12/05/2016_346591748
AK_Obstetrics & Gynecology_12/05/2016_346591744



Language Integrated Queries

L I Query	Description
`agg(expr, exprs)`	Aggregates on entire DataFrame
`distinct`	Returns new DataFrame with unique rows
`except(other)`	Returns new DataFrame with rows from this DataFrame not in other DataFrame
`filter(expr); where(condition)`	Filter based on the SQL expression or condition
`groupBy(cols: Columns)`	Groups DataFrame using specified columns
`join (DataFrame, joinExpr)`	Joins with another DataFrame using given join expression
`sort(sortcol)`	Returns new DataFrame sorted by specified column
`select(col)`	Selects set of columns

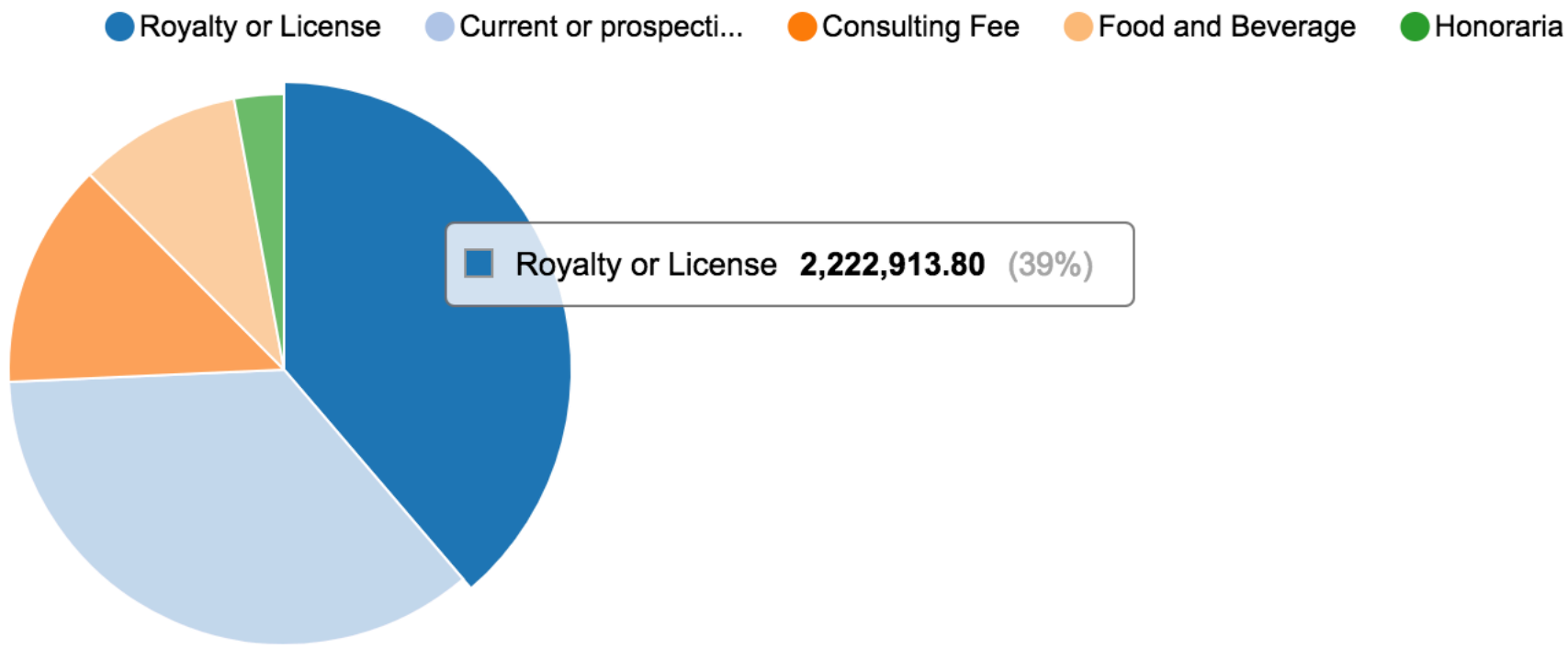
Top 5 Nature of Payment by count

```
val res = pdf.groupBy("Nature_of_Payment")
             .count()
             .orderBy(desc(count))
             .show(5)
```

```
+--------------------+-----+
|   Nature_of_payment|count|
+--------------------+-----+
|   Food and Beverage|28806|
|  Travel and Lodging|  446|
|      Consulting Fee|  259|
|Compensation for ...|  163|
|           Honoraria|  112|
+--------------------+-----+
```

Top 5 Nature of Payment by amount of payment

```
%sql select Nature_of_payment,
     sum(amount) as total from payments
     group by Nature_of_payment order by total desc limit 5
```



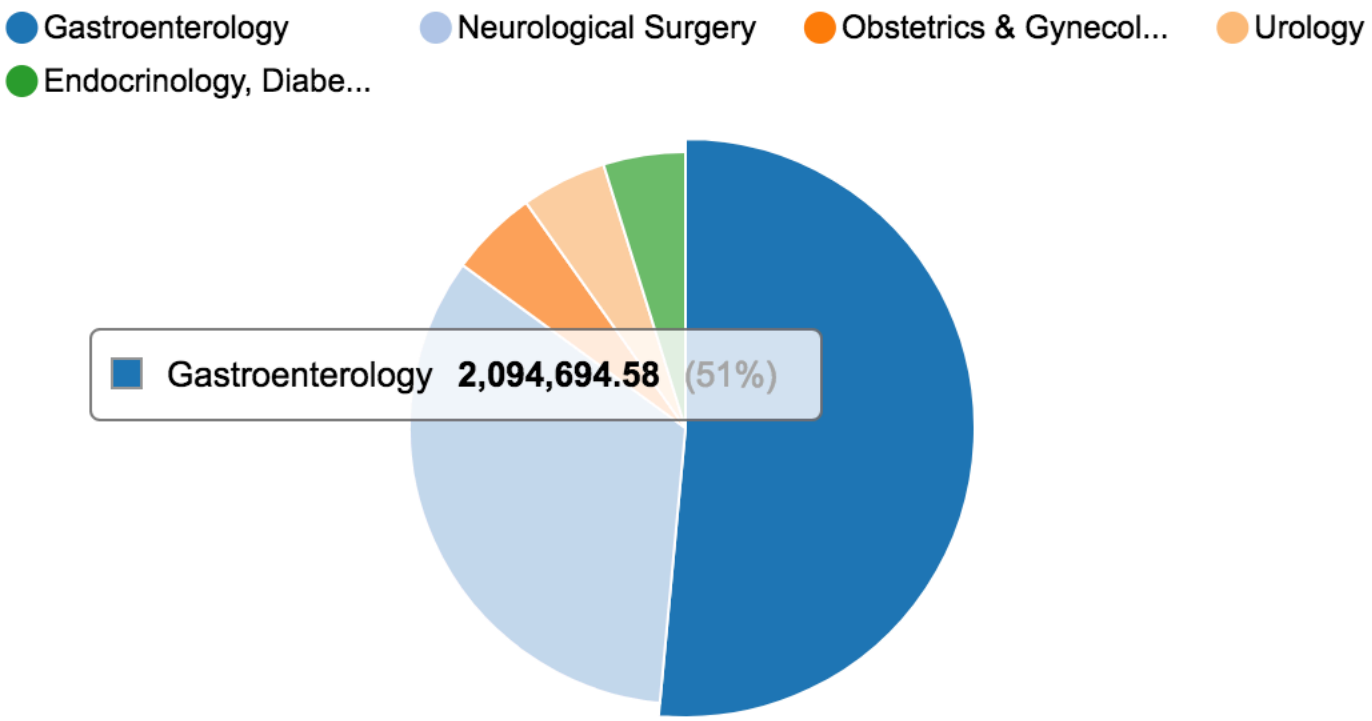
What are the Nature of Payments with payments > $1000 with count

```
pdf.filter($"amount" > 1000)
    .groupBy("Nature_of_payment")
    .count().orderBy(desc("count")).show()
```

```
+--------------------+-----+
|   Nature_of_payment|count|
+--------------------+-----+
|      Consulting Fee|  156|
|           Honoraria|   54|
|  Royalty or License|   38|
|  Travel and Lodging|   32|
|Compensation for ...|   24|
|           Education|   21|
|Current or prospe...|   21|
|Space rental or f...|   10|
|               Grant|    5|
|   Food and Beverage|    1|
+--------------------+-----+
```

Top 5 Physician Specialties by total Amount

```
%sql select physician_specialty, sum(amount) as total
     from payments where physician_specialty IS NOT NULL
     group by physician_specialty order by total desc limit 5
```



Average Payment by Specialty

What is the average Payment by Specialty?

```
%sql select physician_specialty, count(*) as cnt, avg(amount) as avgamount
 from payments where physician_specialty IS NOT NULL group by physician_specialty order by avgamount desc limit 10
```

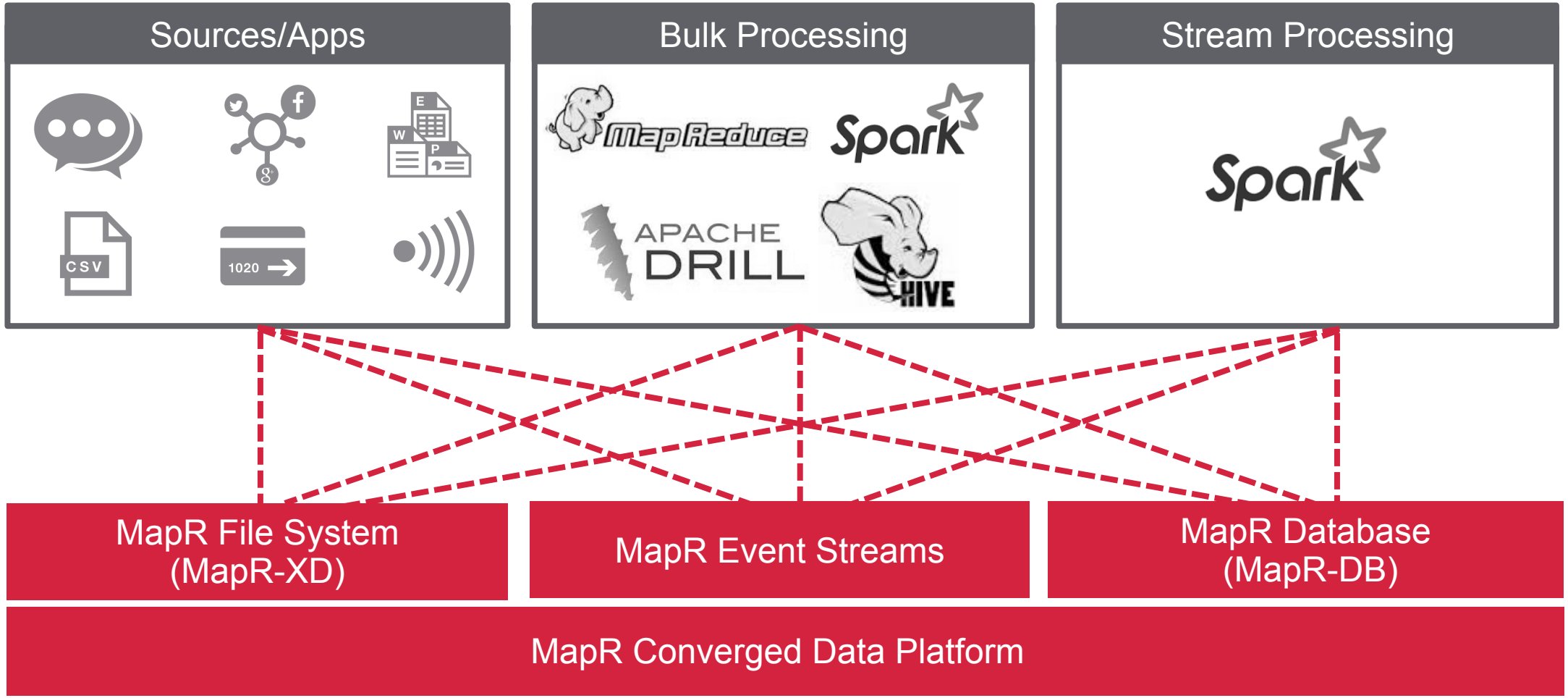
physician_specialty	cnt	avgamount
Neurological Surgery	72	18966.440000000002
Endocrinology, Diabetes & Metabolism	38	5143.806052631579
Blood Banking & Transfusion Medicine	1	2500.0
Emergency Medicine	146	1184.4565753424654
Mental Health	2	589.1999999999999
Podiatrist	294	571.9885374149659

Top Payers by Total Amount with count

```
%sql select payer, payer_state, count(*) as cnt,
     sum(amount) as total from payments
     group by payer, payer_state  order by total desc limit 10
```

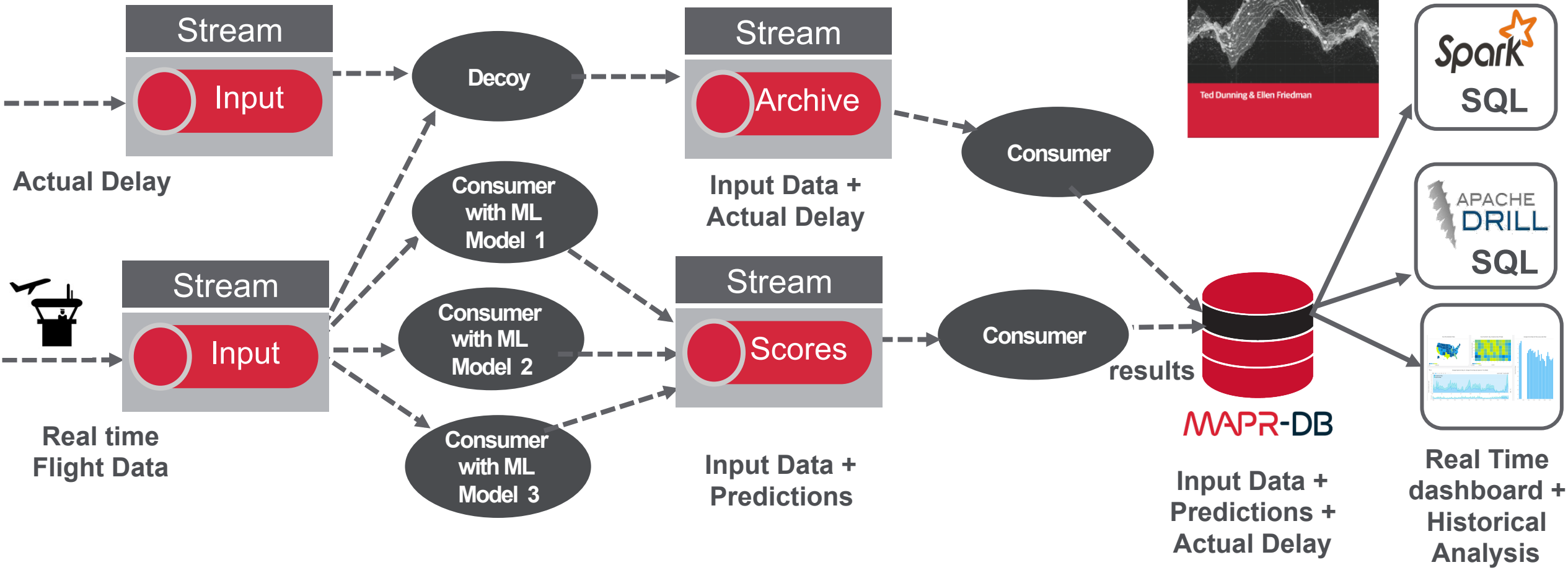
payer	payer_state	cnt	total
Leading Edge Spinal Implants LLC	CO	21	2040000.0
Braintree Laboratories, Inc.	MA	5818	1822474.34000
Mission Pharmacal Company	TX	18818	662132.010000
Affordable Pharmaceuticals, LLC	MA	8	353968.660000
ZOLL Medical Corporation	MA	193	308006.669999
Alliqua BioMedical, Inc.	PA	1013	275810.000000

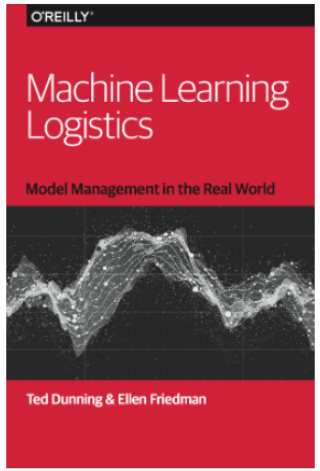
Building a Complete Data Architecture



All of these components can run on the same cluster with the MapR Converged platform.

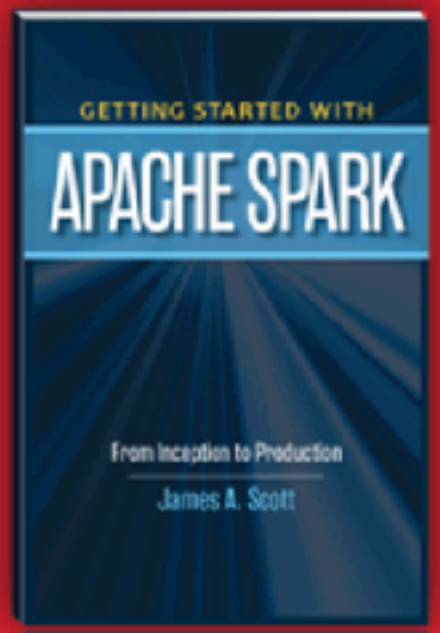
Data Pipelines and Machine Learning Logistics

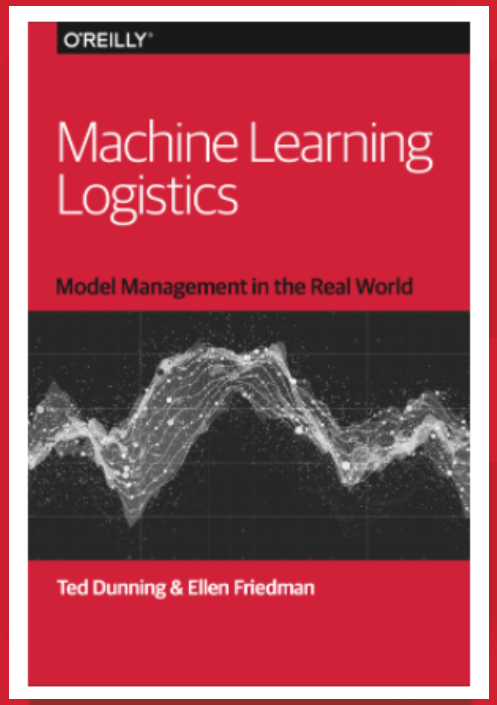




MAPR

Complimentary books



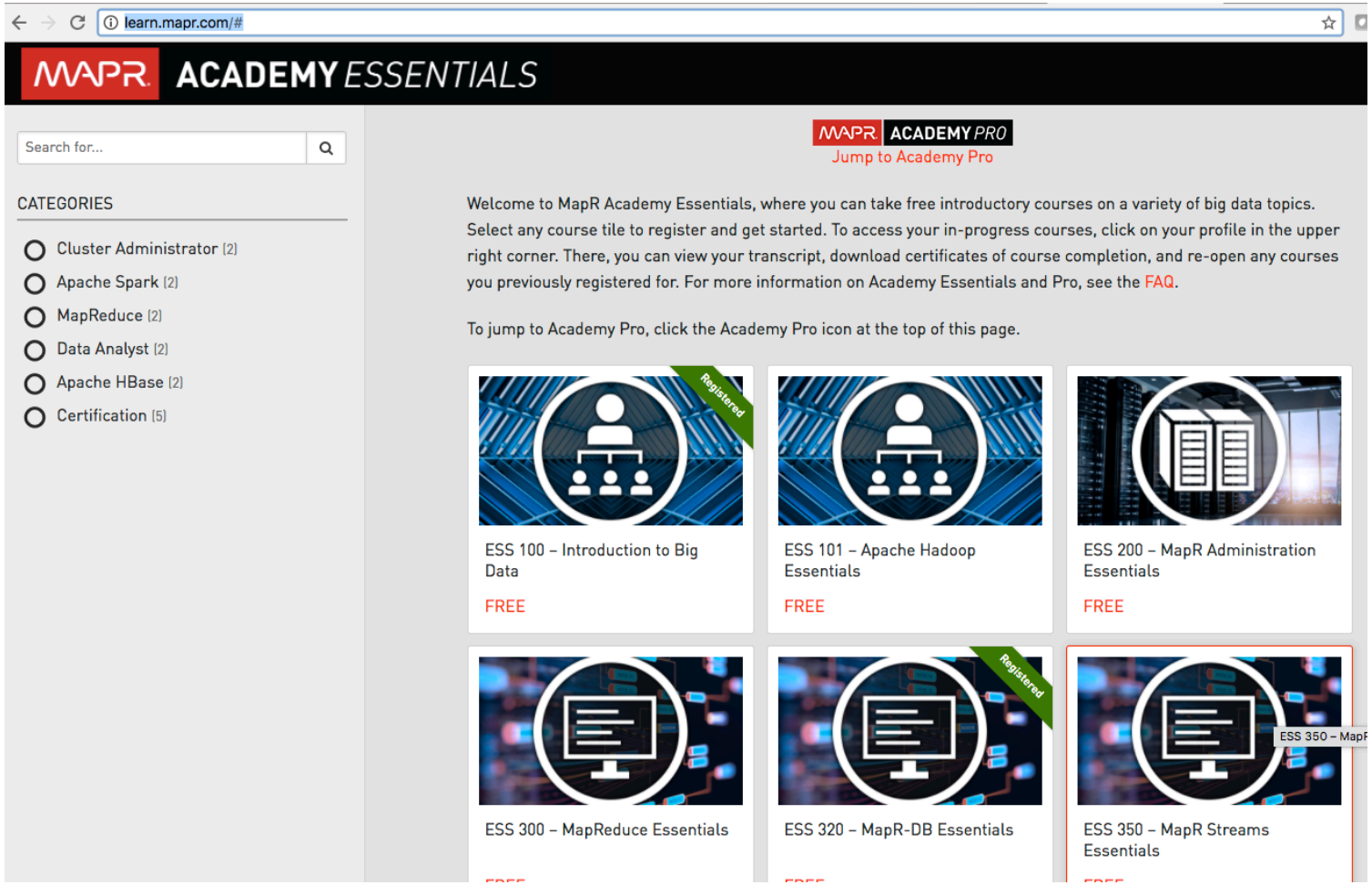


Get your copies

download at mapr.com/ebooks

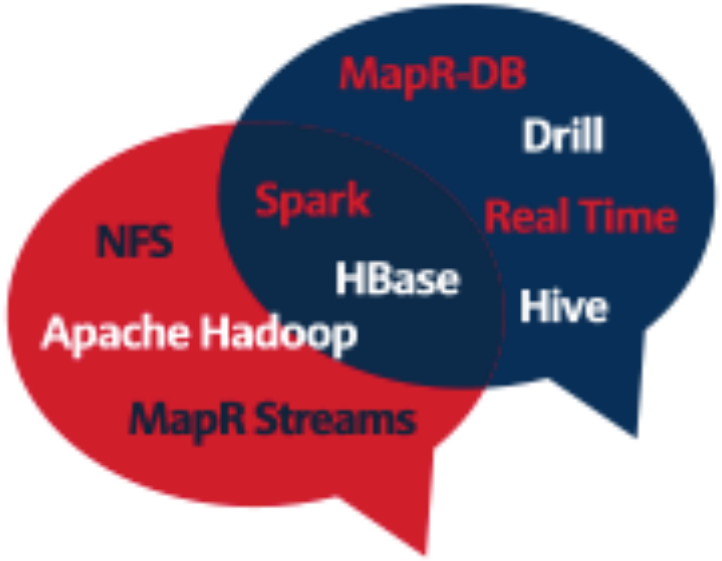
To Learn More:

- MapR Free ODT http://learn.mapr.com/



CONVERGE community
Powered by MapR

…helping you put data technology to work



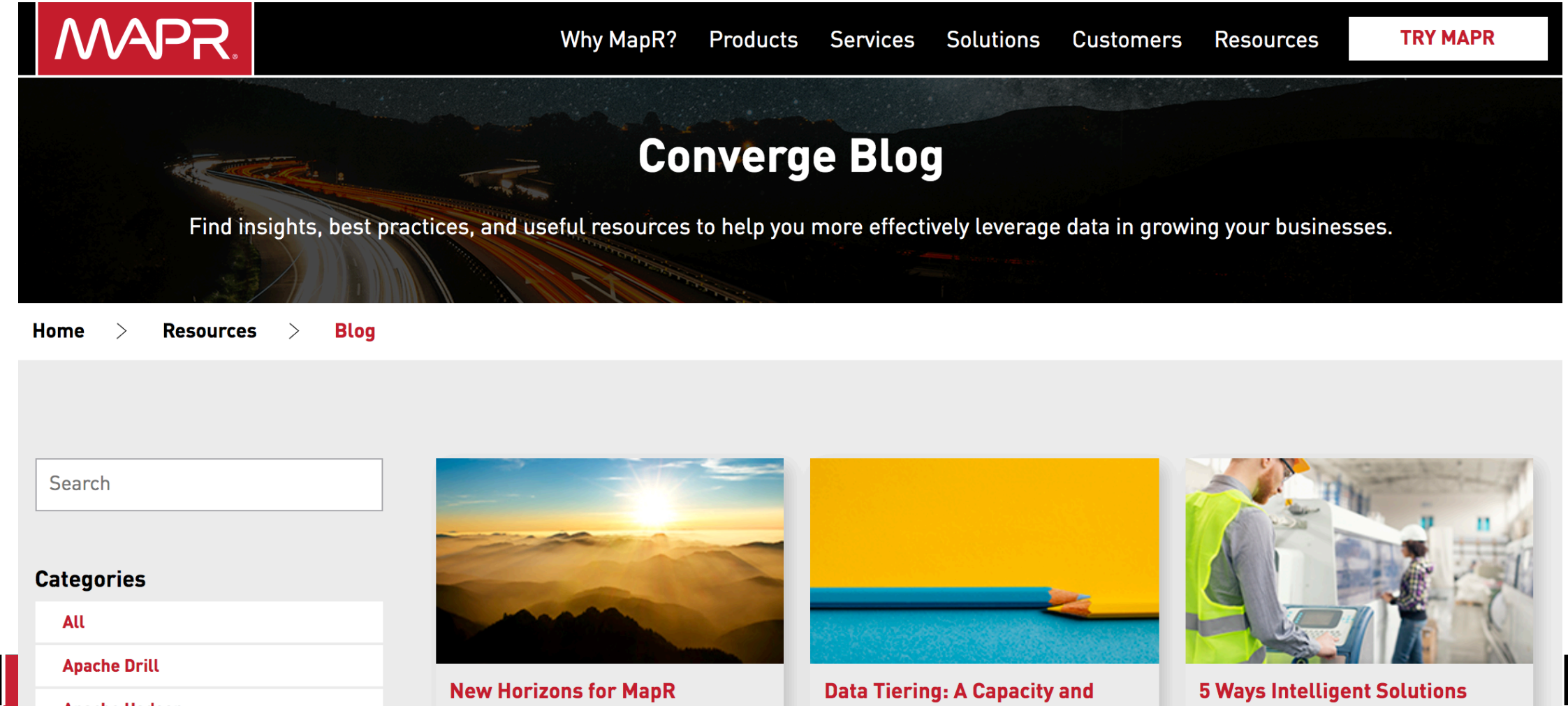
community.mapr.com

Connect with fellow Apache Hadoop and Spark professionals

- Find answers
- Ask technical questions
- Join on-demand training course discussions
- Follow release announcements
- Share and vote on product ideas
- Find Meetup and event listings

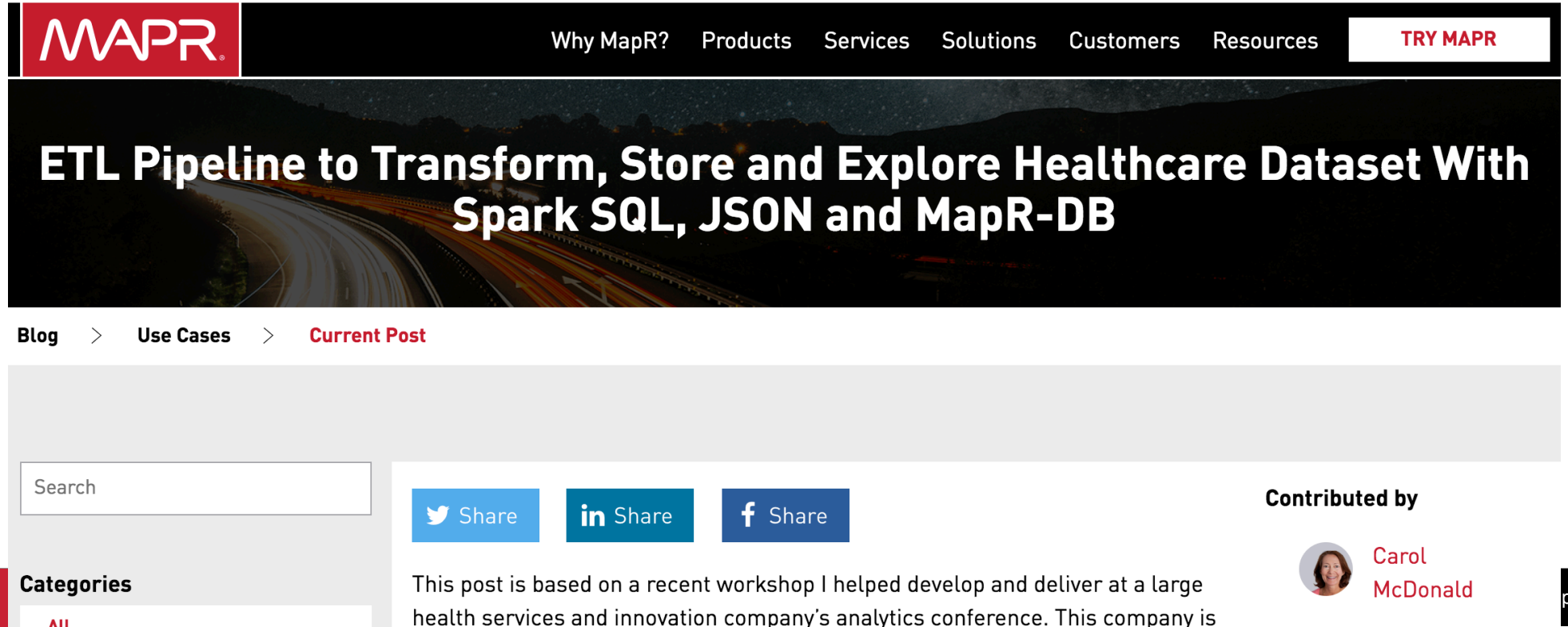
MapR Blog

- https://www.mapr.com/blog/



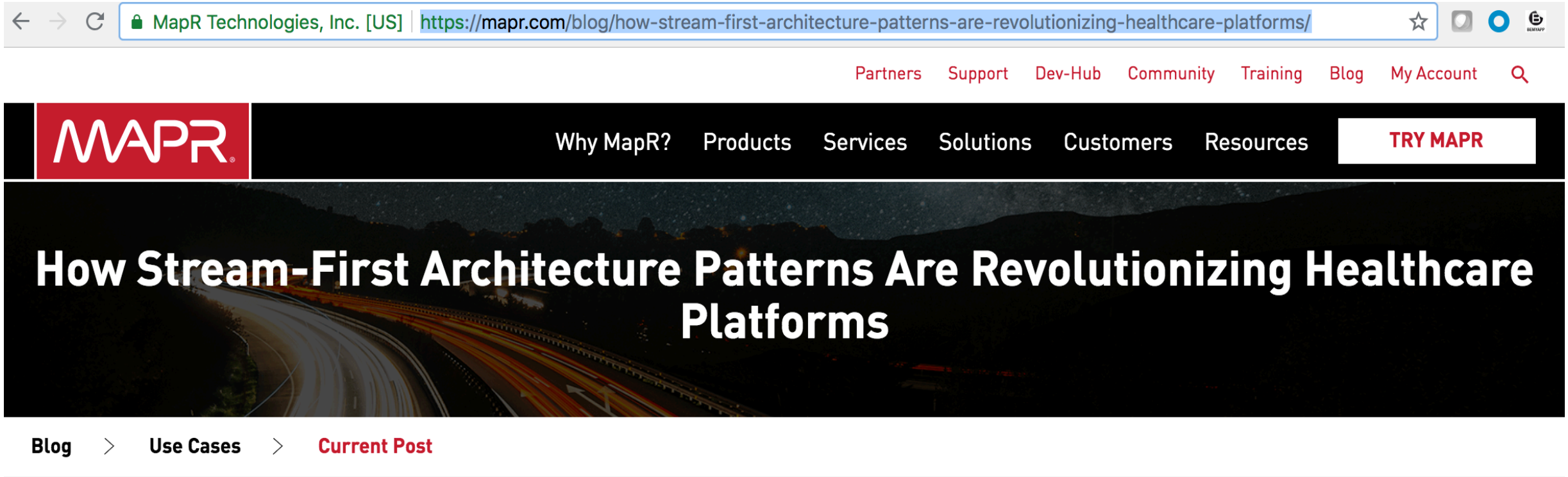
To Learn More: ETL Payment data pipeline

- https://mapr.com/blog/etl-pipeline-healthcare-dataset-with-spark-json-mapr-db/
- https://mapr.com/blog/streaming-data-pipeline-transform-store-explore-healthcare-dataset-mapr-db/



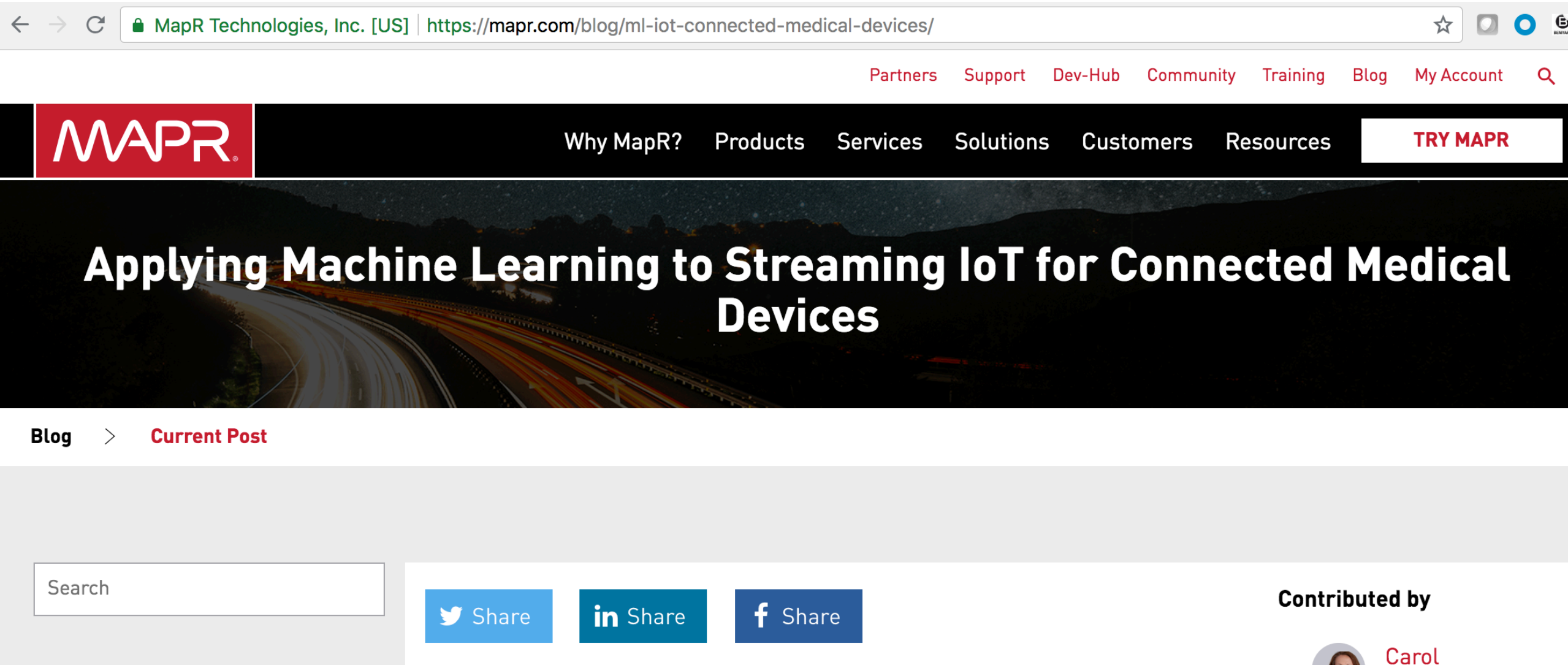
To Learn More:

- **https://mapr.com/blog/how-stream-first-architecture-patterns-are-revolutionizing-healthcare-platforms/**



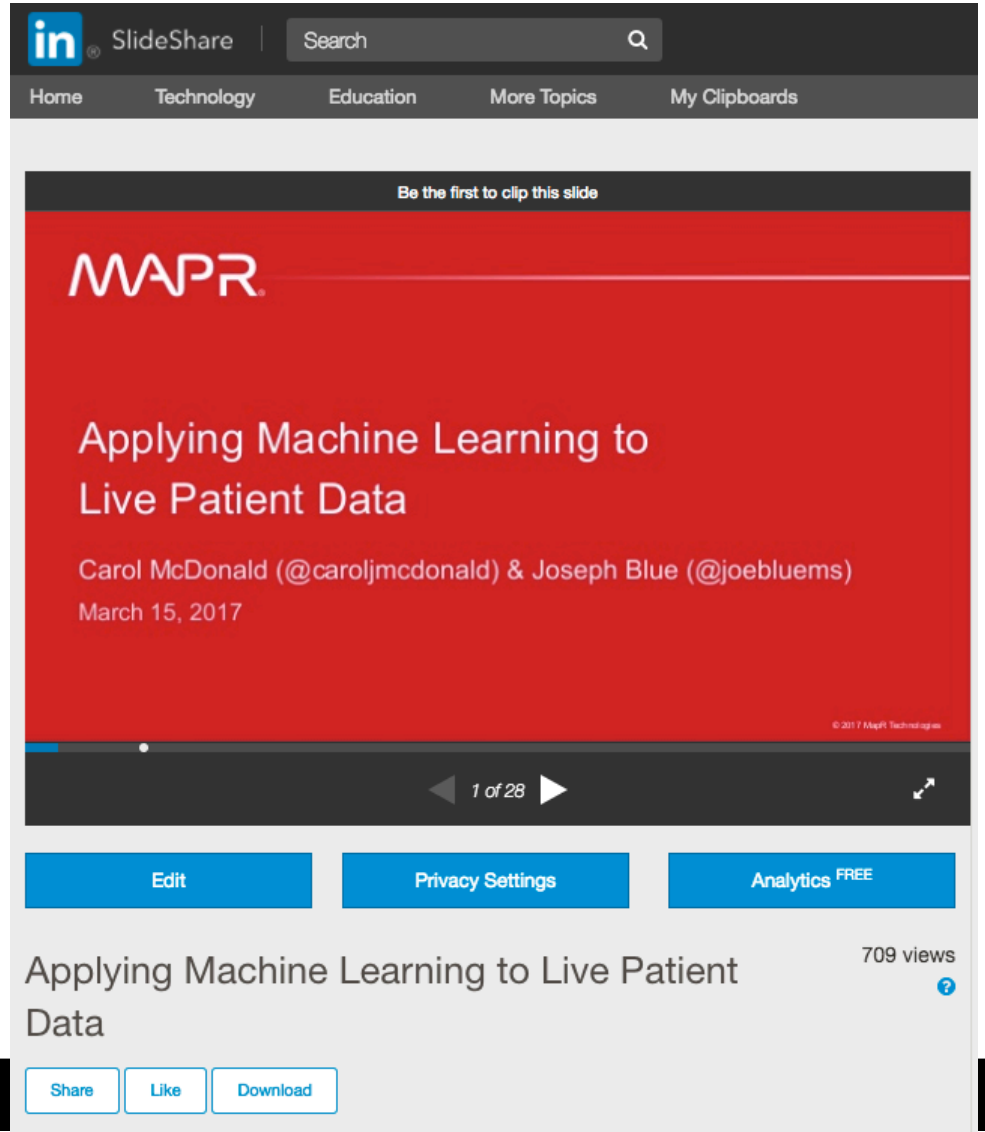
To Learn More:

- **https://mapr.com/blog/ml-iot-connected-medical-devices/**



Applying Machine Learning to Live Patient Data

- **https://www.slideshare.net/caroljmcdonald/applying-machine-learning-to-live-patient-data**



MapR Container for Developers

- https://maprdocs.mapr.com/home/MapRContainerDevelopers/MapRContainerDevelopersOverview.html

MapR Container for Developers

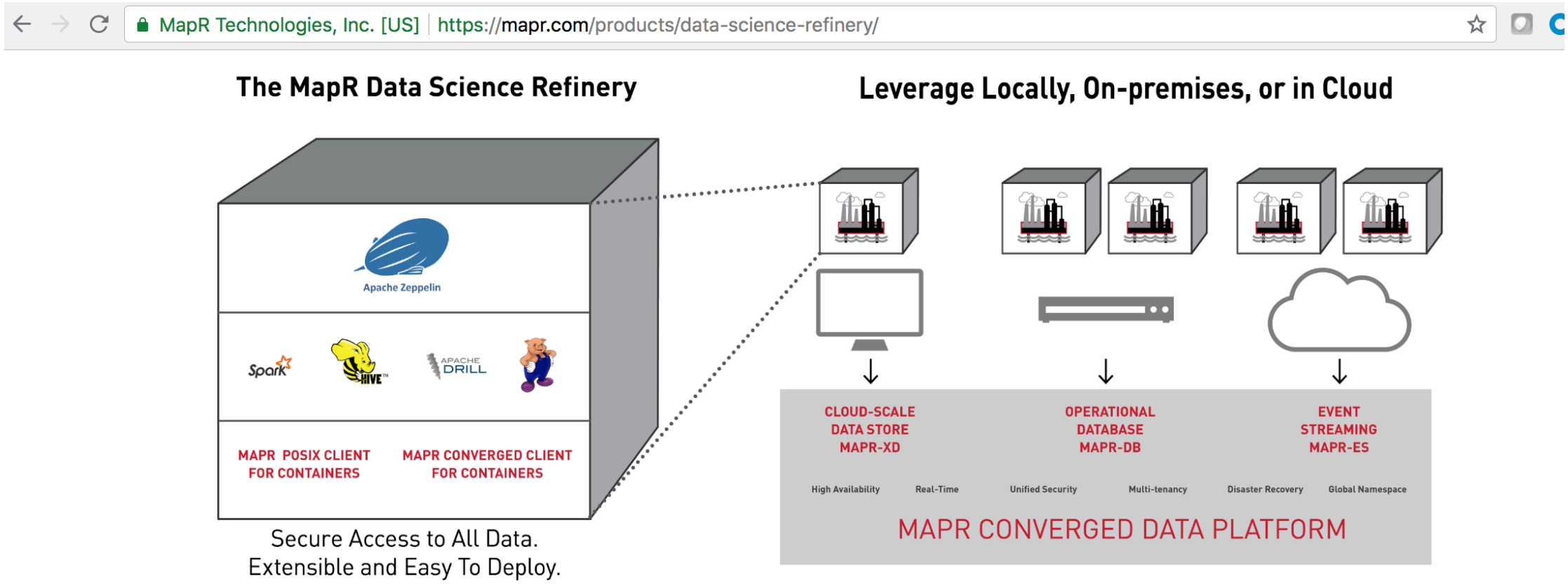
The MapR Container for Developers is a docker container that enables you to create a single node MapR cluster. The container is lightweight and designed to run on your laptop. It requires no additional configuration for you to connect your clients, also running on your laptop, to the cluster.

The MapR cluster created by the docker image includes the following components:

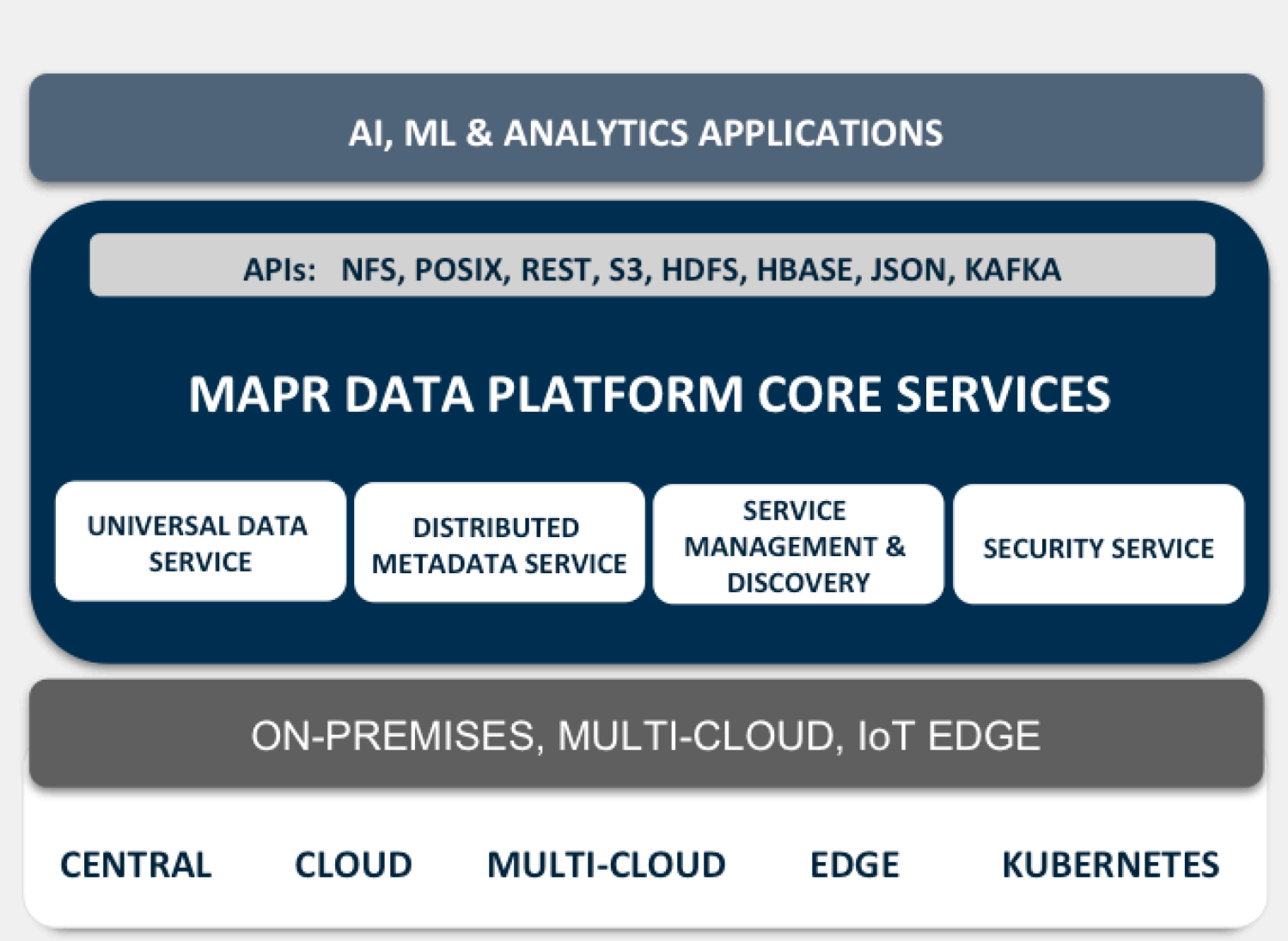
- MapR Core 6.0
  - MapR-XD
  - MapR-DB
  - MapR-ES
  - MapR Control System
  - MapR NFS
- Apache Drill 1.11.0-1710
- Apache Spark 2.1.0-1710

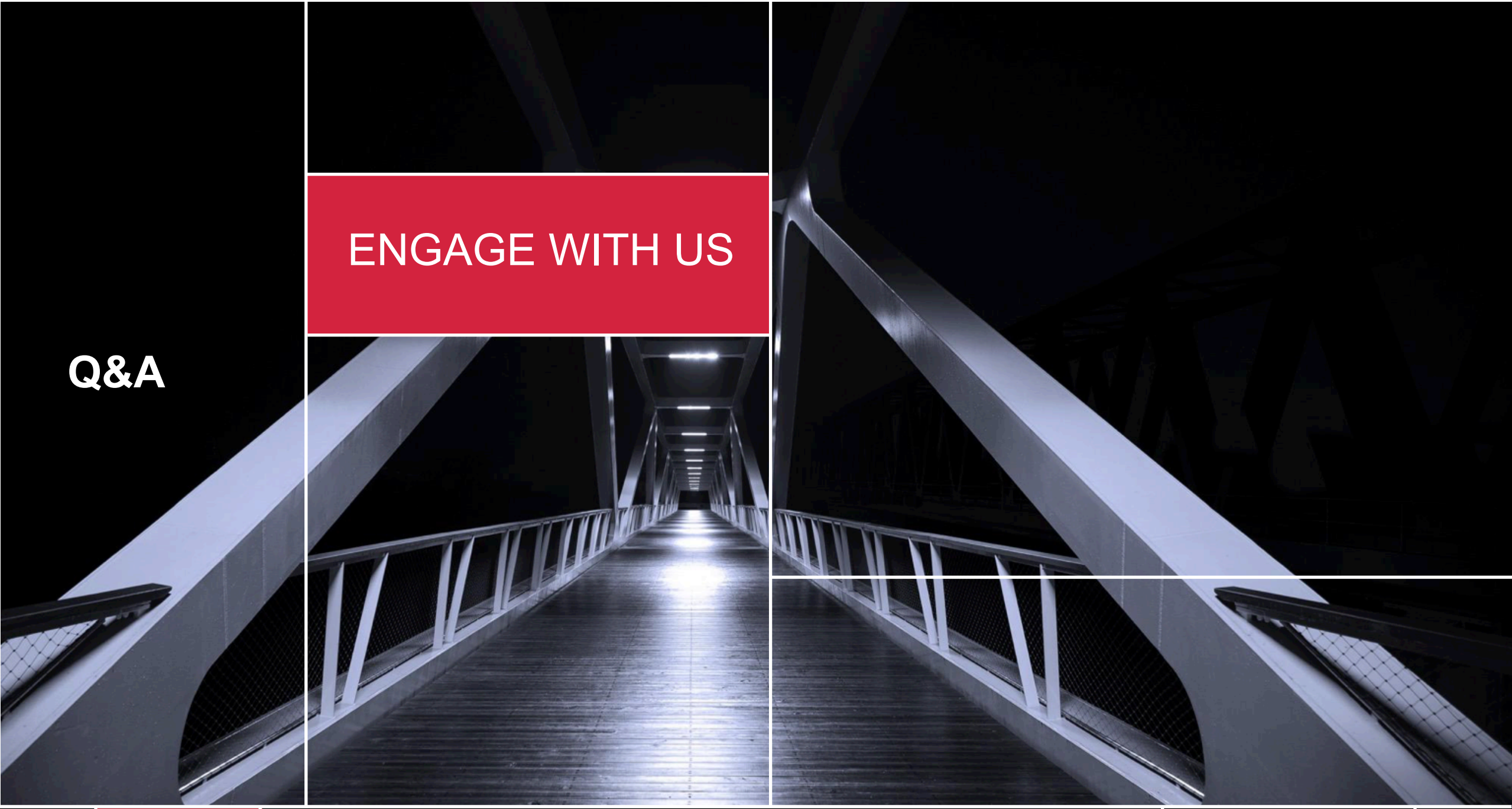
MapR Data Science Refinery

- https://mapr.com/products/data-science-refinery/



MapR Data Platform





ENGAGE WITH US

Q&A