Spark and Kafka for Data Analysis and Machine Learning

Cork JUG, 2019-02-19

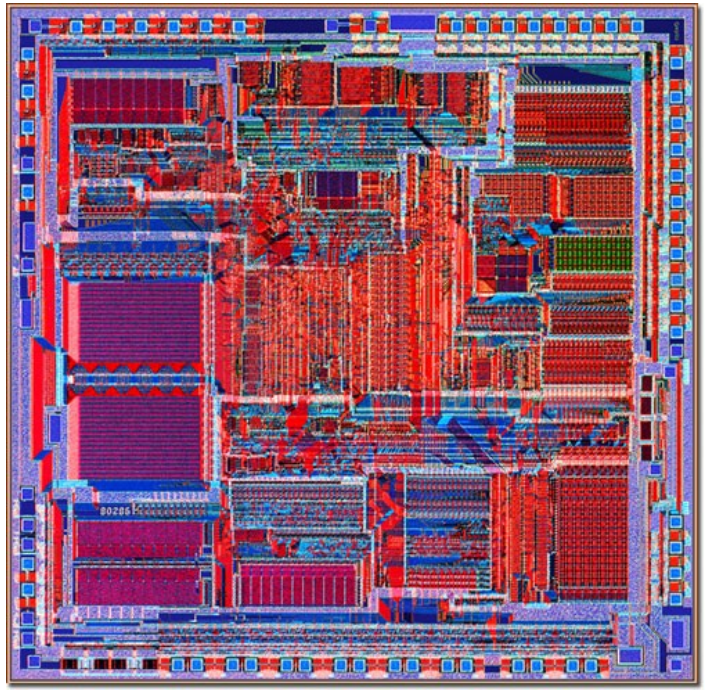
Johannes Ahlmann

Image:
https://softwareengineeringdaily.com/wp-content/uploads/2015/08/kafka-logo-wide.png
https://commons.wikimedia.org/wiki/File:Apache_Spark_logo.svg

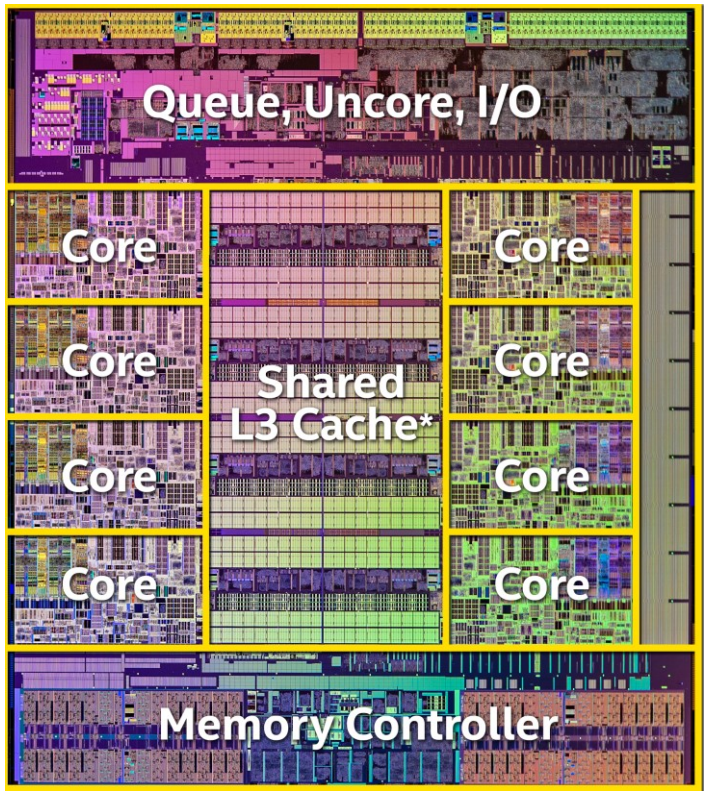
About Me

- Johannes Ahlmann
- sensatus.io
  - On-Prem AI Models
  - Gathering and Enriching Web Data
  - Sales & Client Intelligence
- webdata.org
  - Share Libraries and Best Practices
  - Bring Data Scientists and SME Companies together
  - ForDevelopers
  - AwesomeAvailableDatasets
- Contact:
  johannes@sensatus.io

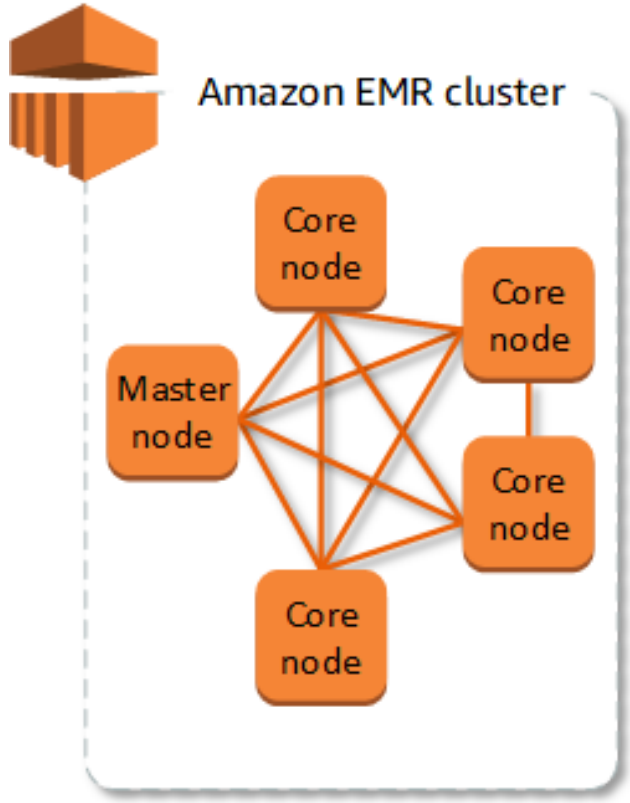
Times Have Changed



Intel 80286
1 core, 0.13M transistors



Intel i7-9900K
8 cores, 3000M transistors



Challenges 1 / 2

Many fantastic libraries written in C, Fortran, Python, Go

- How can we leverage libraries together that are written in different languages, or for different (virtual) machines?

Multiprocessing and concurrency are hard in C, Python, etc.

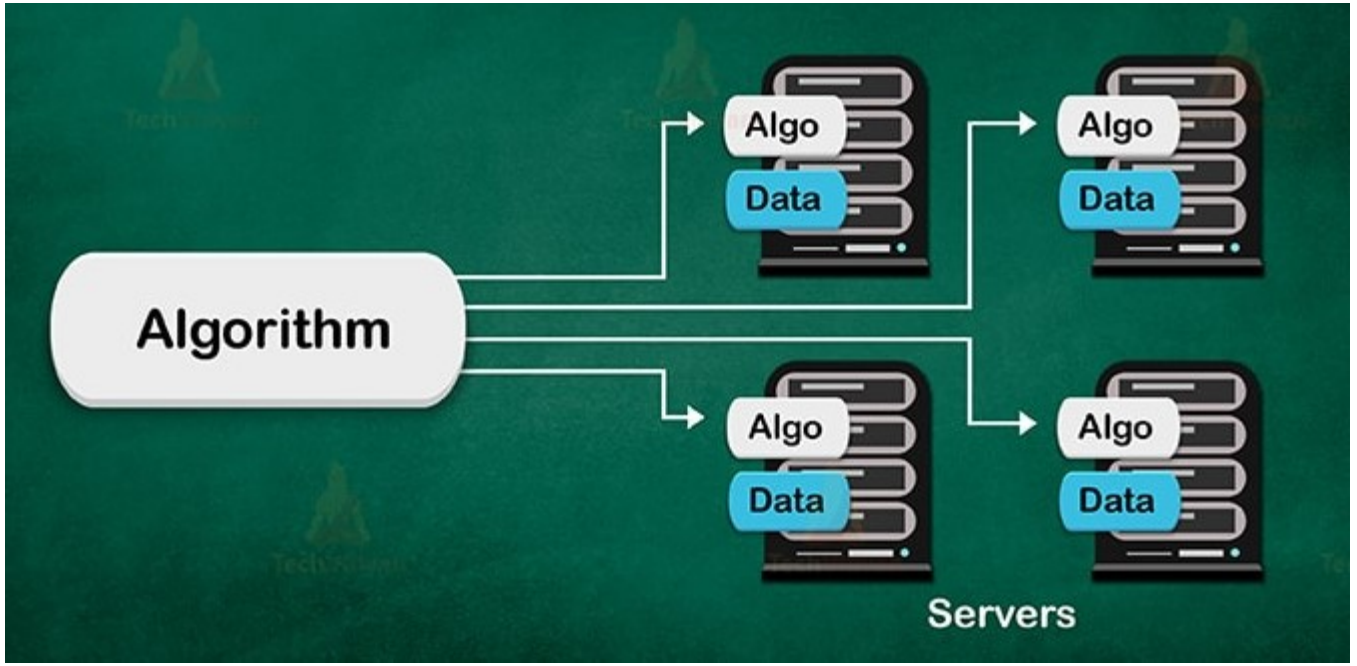
- How can we leverage compute of multiple cores per machine, let alone multiple machines

Distributed computing is hard

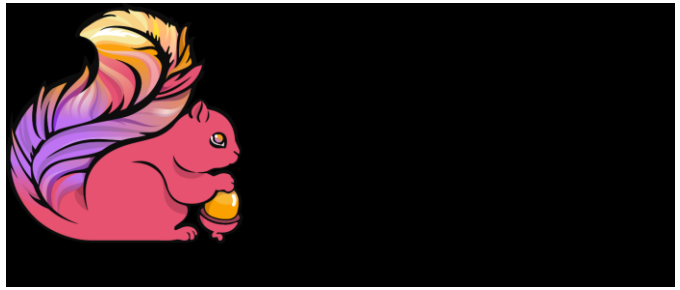
- How can we monitor/ manage/ debug/ reason about it?

Challenges 2 / 2

- Data Locality
- Load Balancing
- Scaling
- State Persistence
  - Start/ Stop
  - Error Recovery



JVM Frameworks to the Rescue

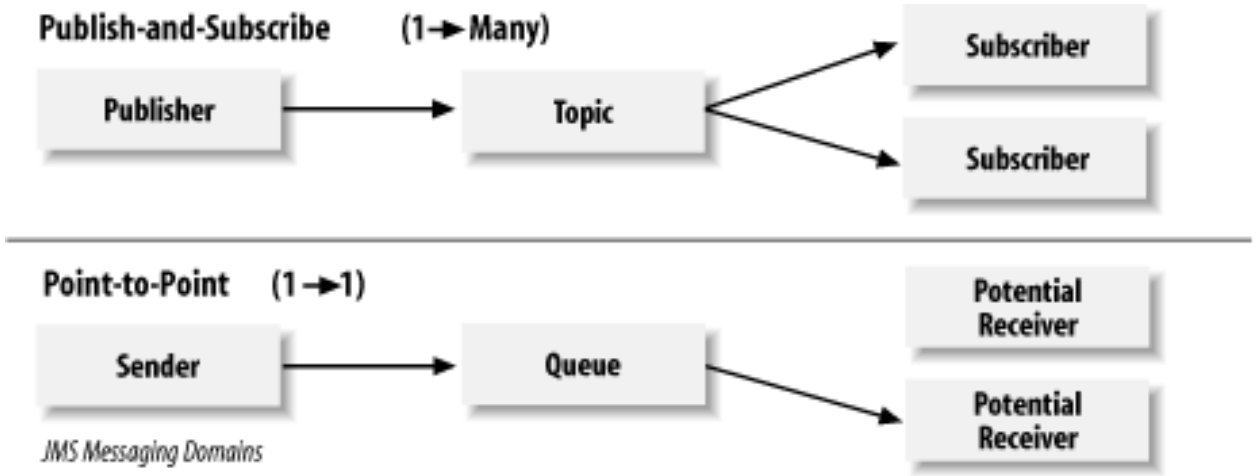



kafka

Background: Queues & PubSub

- PubSub
  - multiple subscribers
  - no scaling

- Queues
  - multiple consumers
  - single-subscriber
  - scaling, load balancing



A high-throughput distributed messaging system

- Decouples Data Pipelines
- Scalable & Fault-Tolerant
- Kafka Functionalities
  - Messaging
  - Processing
  - Storing
- Performance (>100k/s)
  - Batching
  - Zero Copy I/O
  - Leverages OS Cache
- Durability

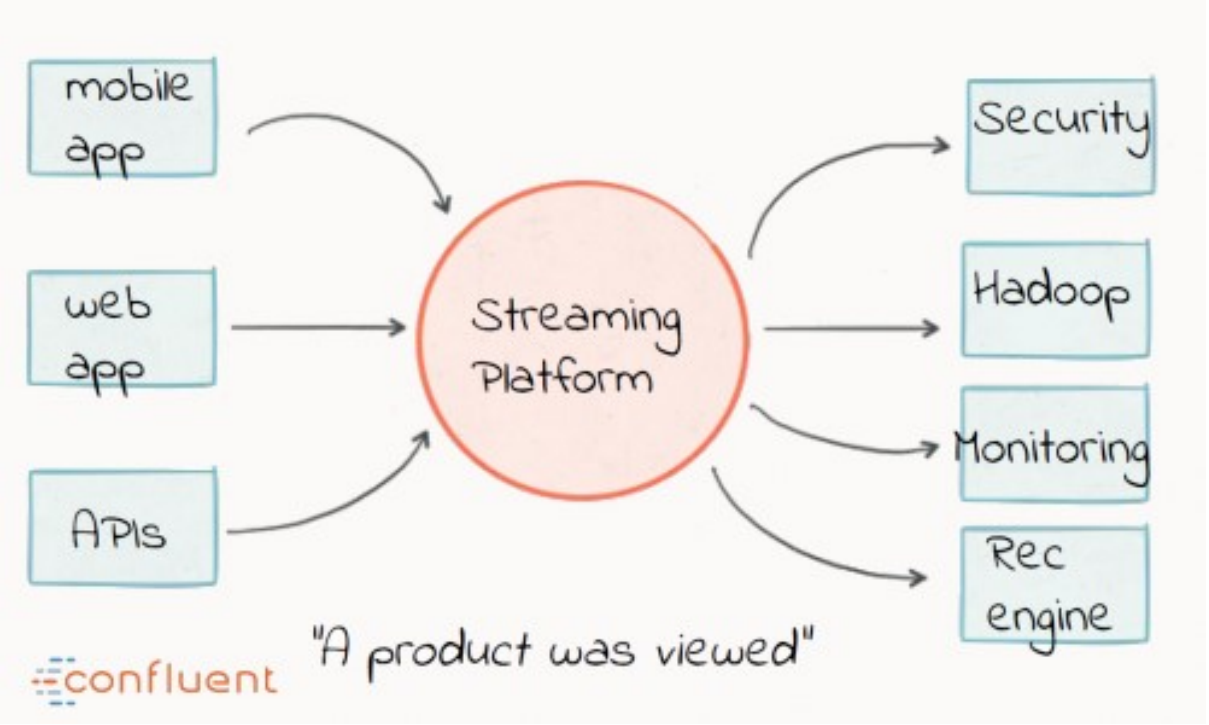


Image Credit: Confluent

Core Concepts

- Producers
- Consumers
- Brokers
- Topics
- Zookeeper
  - Offsets
  - Broker Addresses

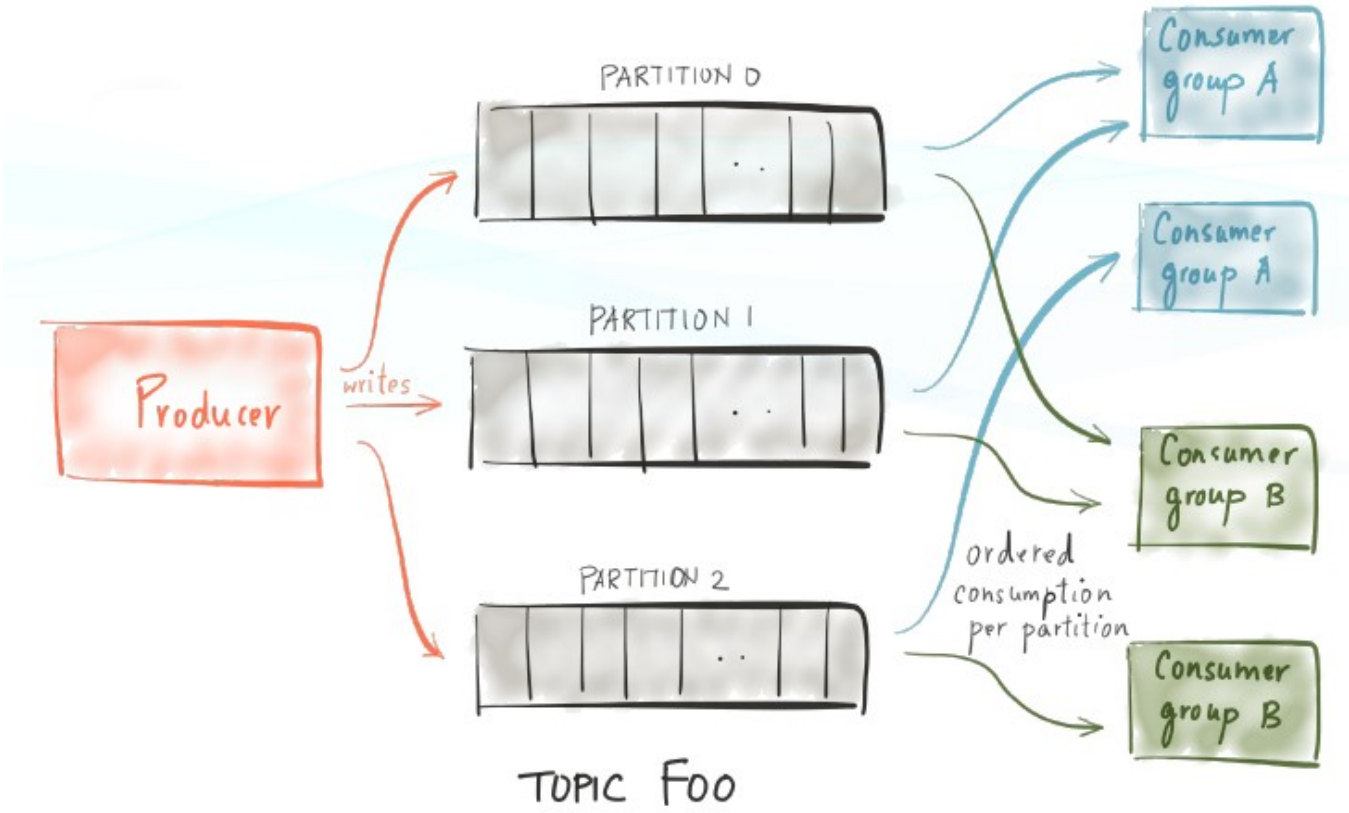
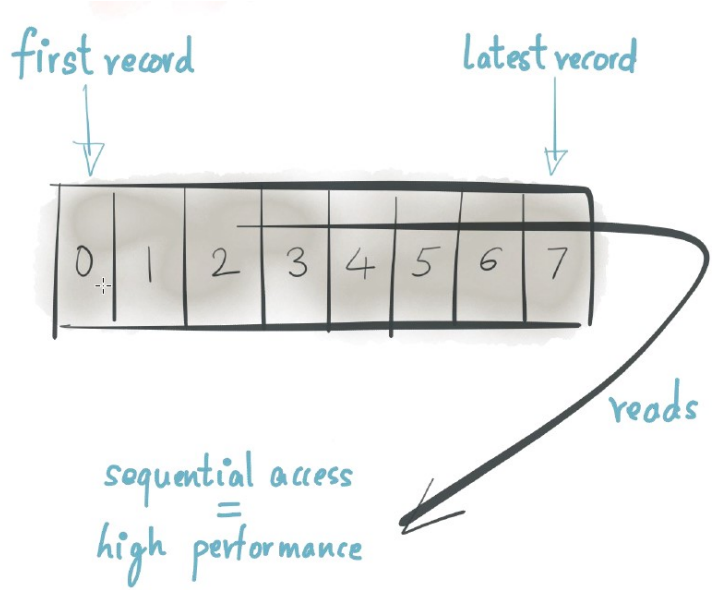


Image Credit: Confluent

Key Idea: Partitioned Log

- Very fast, due to zero copy I/O and batching
- Uses sendfile and OS buffer cache
- Sequential writes to FS
- Order guaranteed within partition
- Scaling



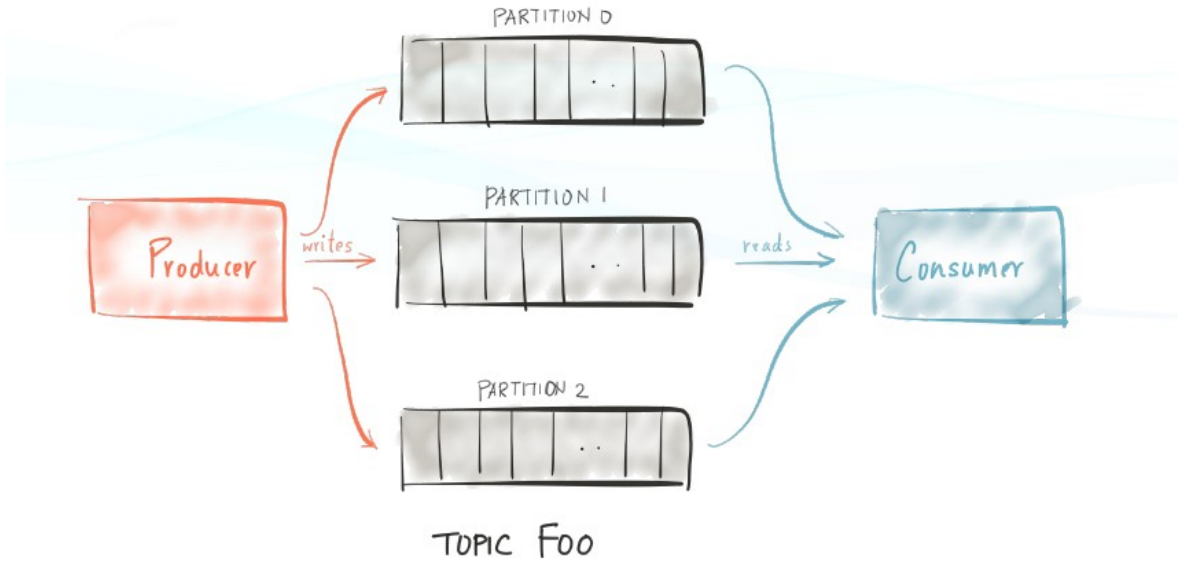
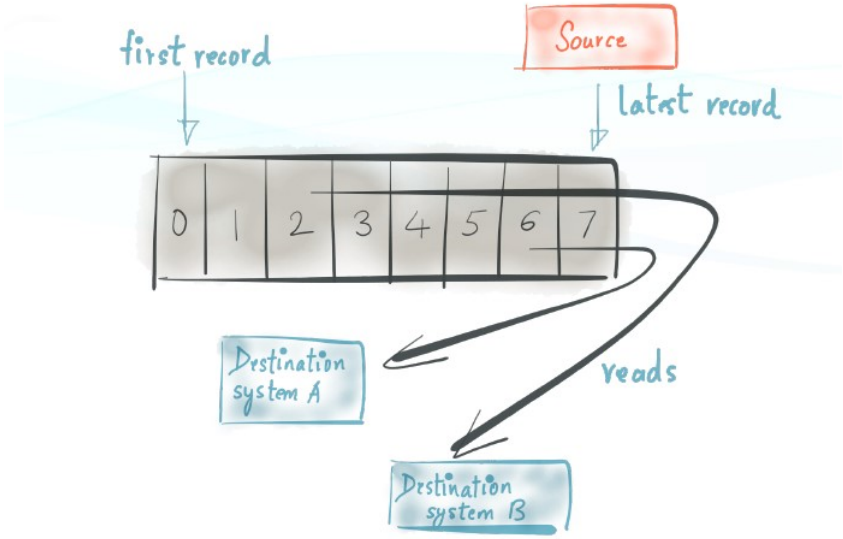


Image Credit: Confluent

Logs & PubSub

- Consumers can be transient
- Consumer Groups
- Delivery Semantics
  - at least once (default)
  - at most once
  - exactly once
- Retention Policy
- Reprocessing



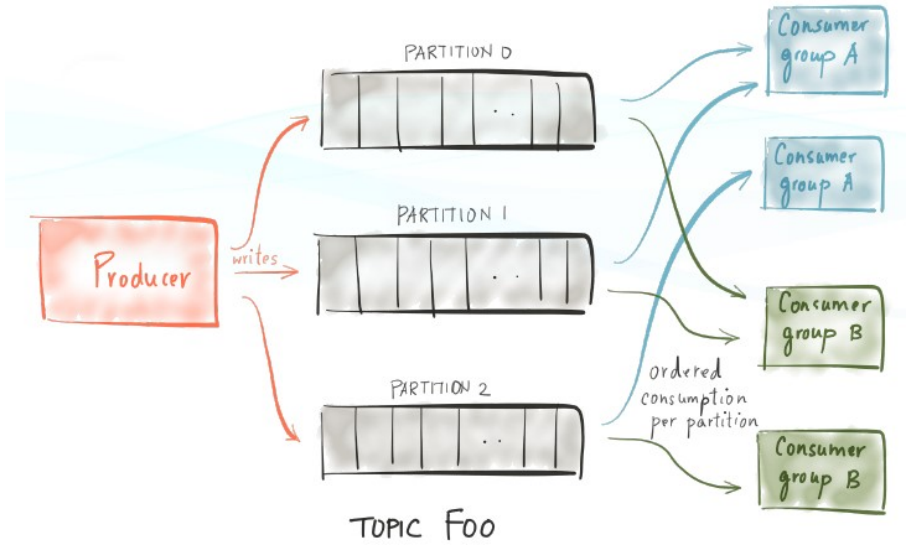


Image Credit: Confluent

Partitions & Replication

- Partitions configurable
- Partition allocation
  - round-robin
  - semantic partition by key
- Replication
  - optional
  - 1 leader, 0 or more followers
  - sync or async
  - flush delay configurable

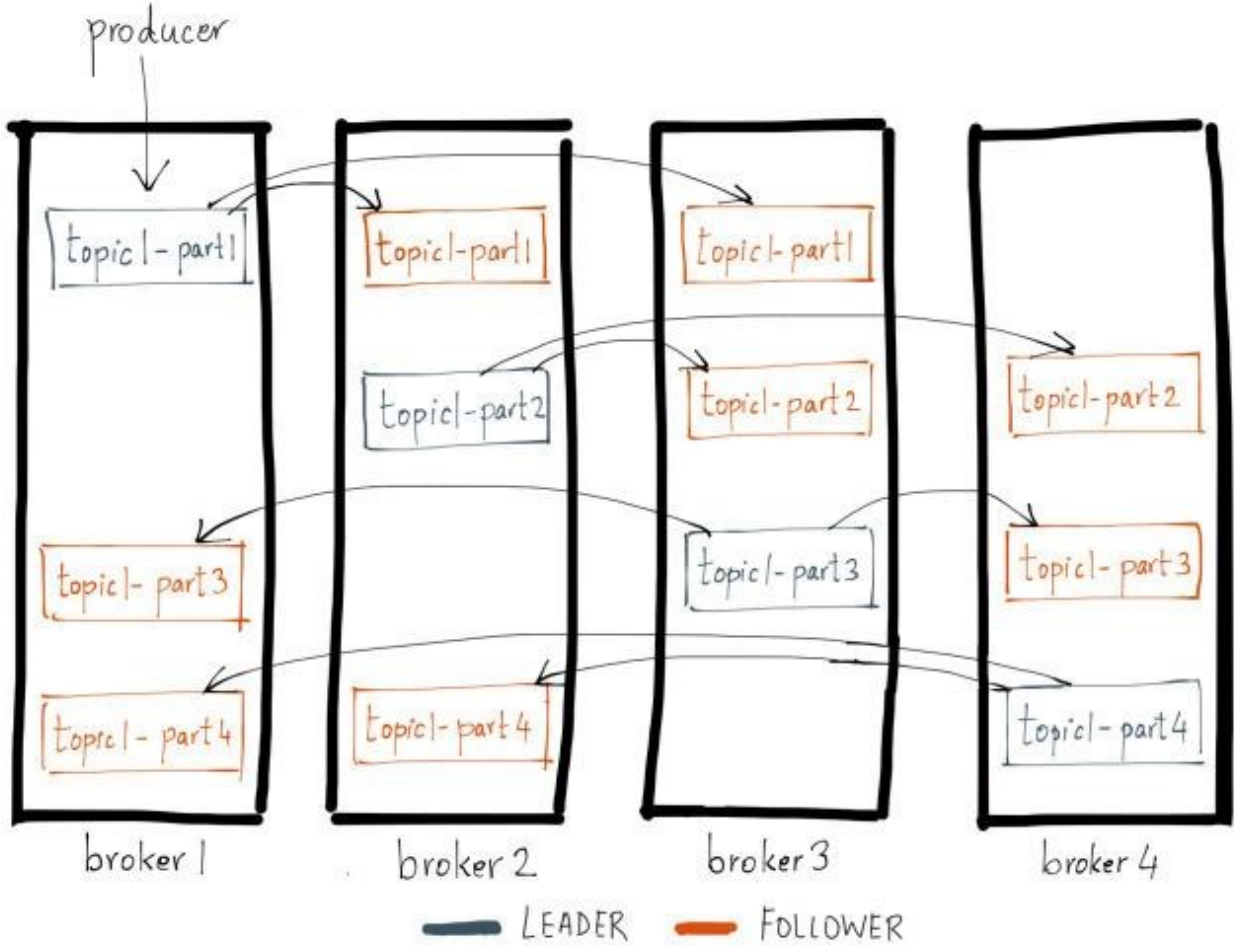


Image Credit: Confluent

Kafka Connect

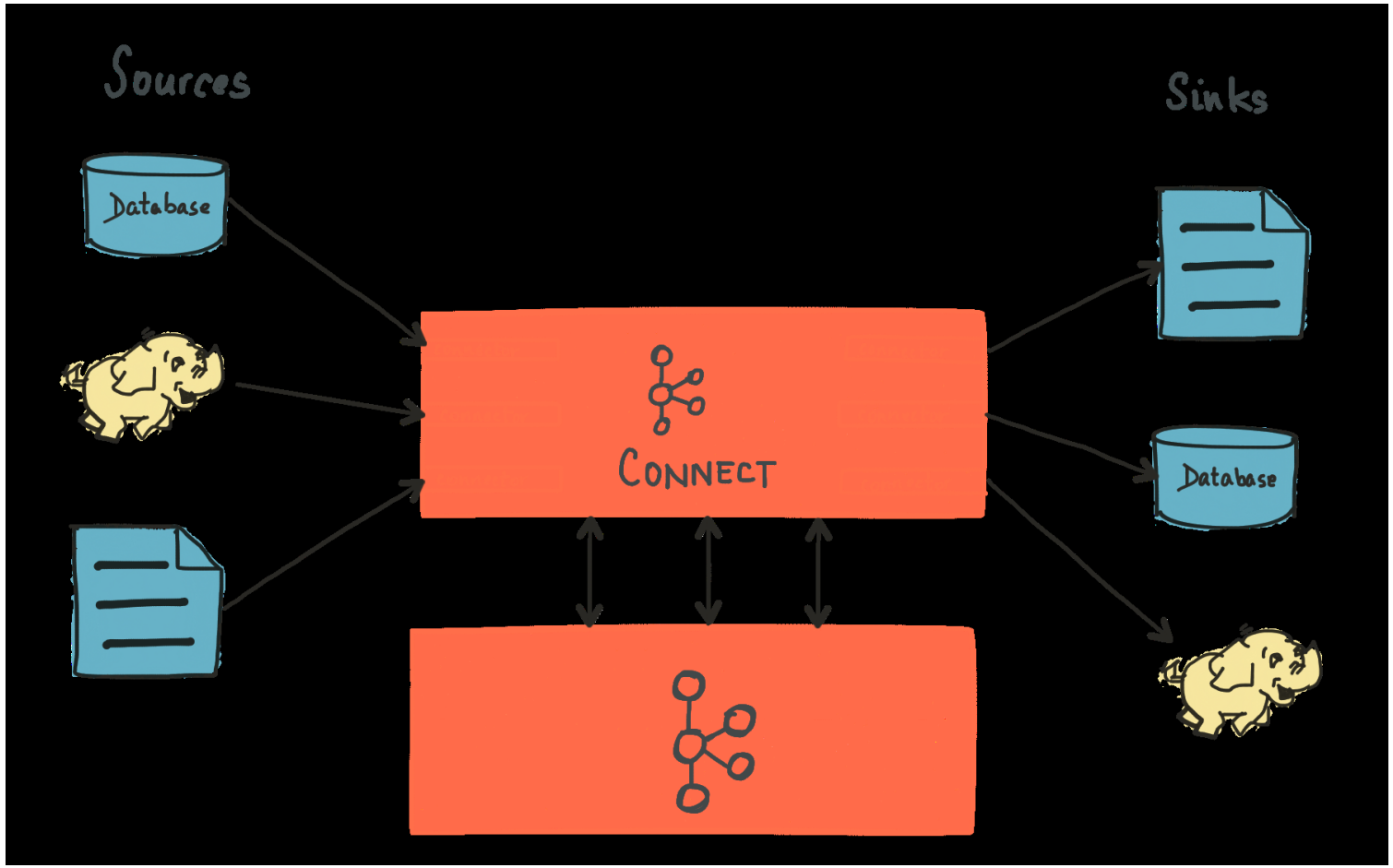


Image Credit: Confluent

Kafka Streams

- Operations
  - filter
  - map
  - join
  - aggregate

  } stateful
- KStream
- KTable
  - manages local state
- Windows

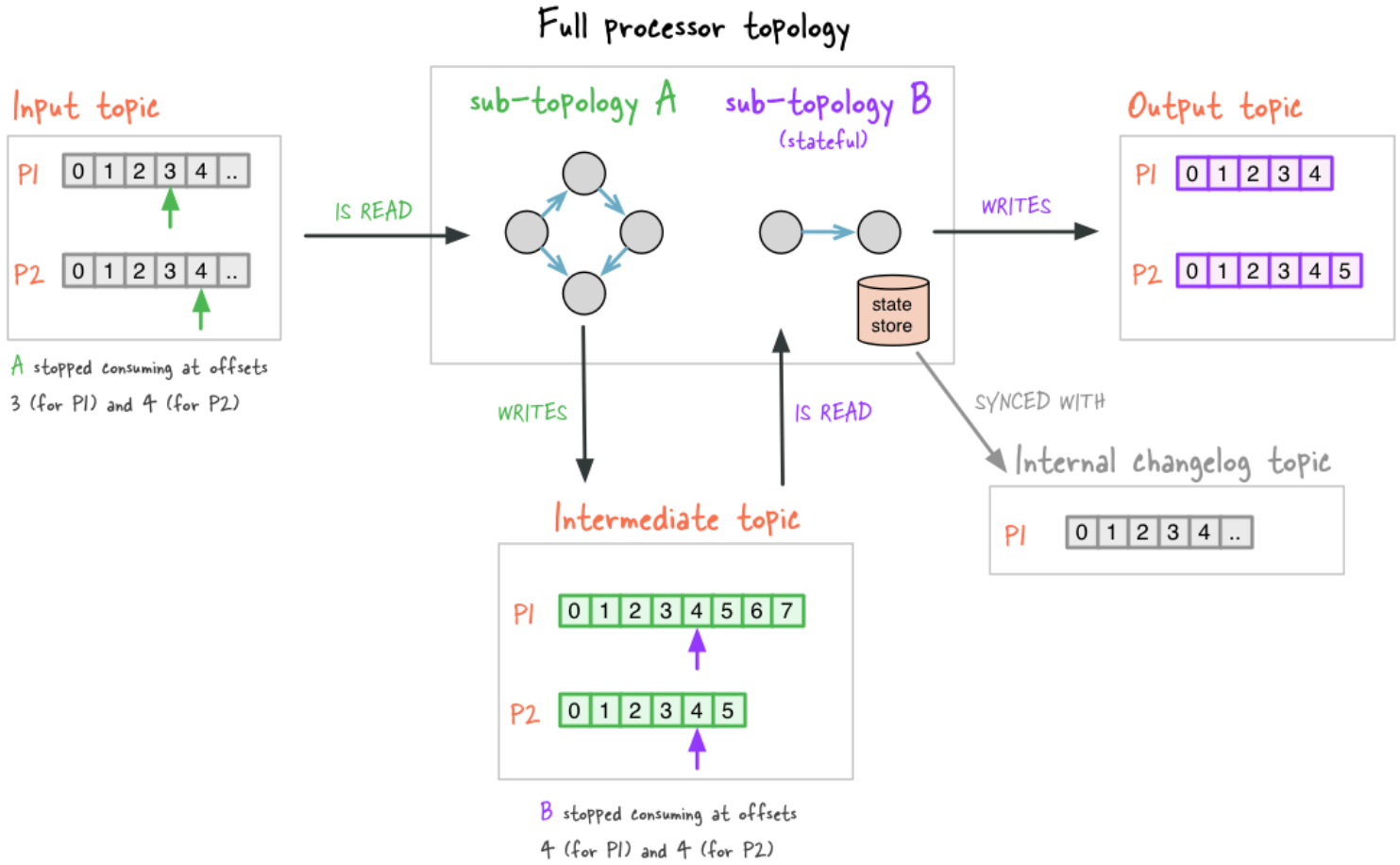


Image Credit: Confluent

Summary

scalability of a **filesystem**

- hundreds of MB/s
- many TBs per server
- commodity hardware

guarantees of a **database**

- persistence
- ordering

distributed by **design**

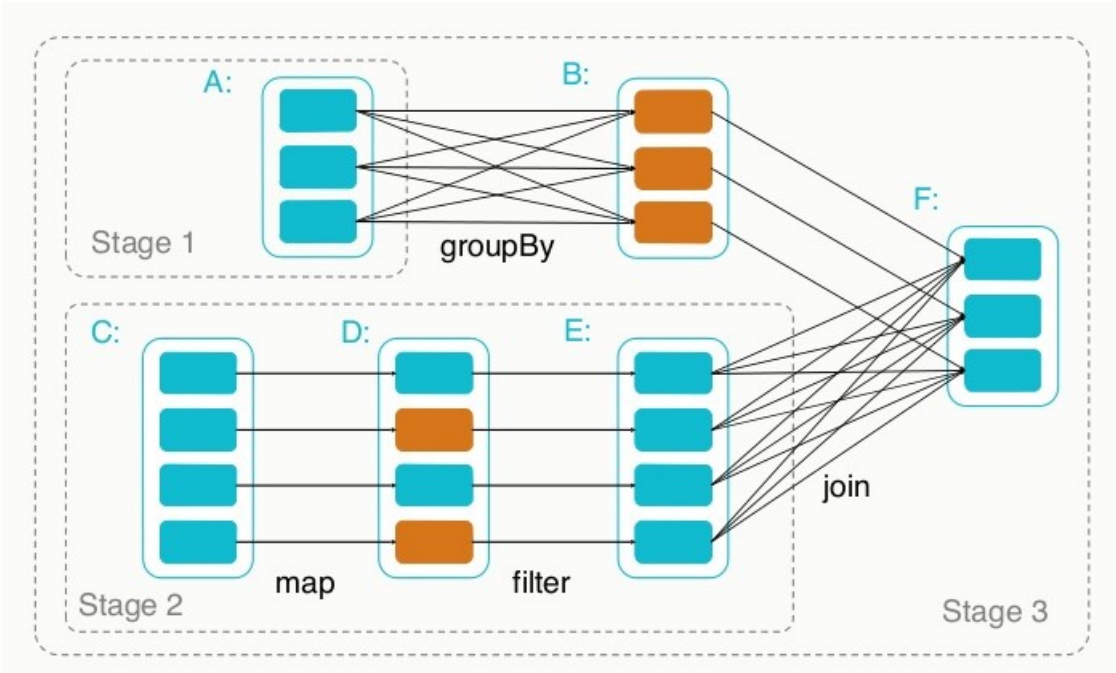
- replication
- partitioning
- horizontal scalability
- fault tolerance

Image Credit: Confluent

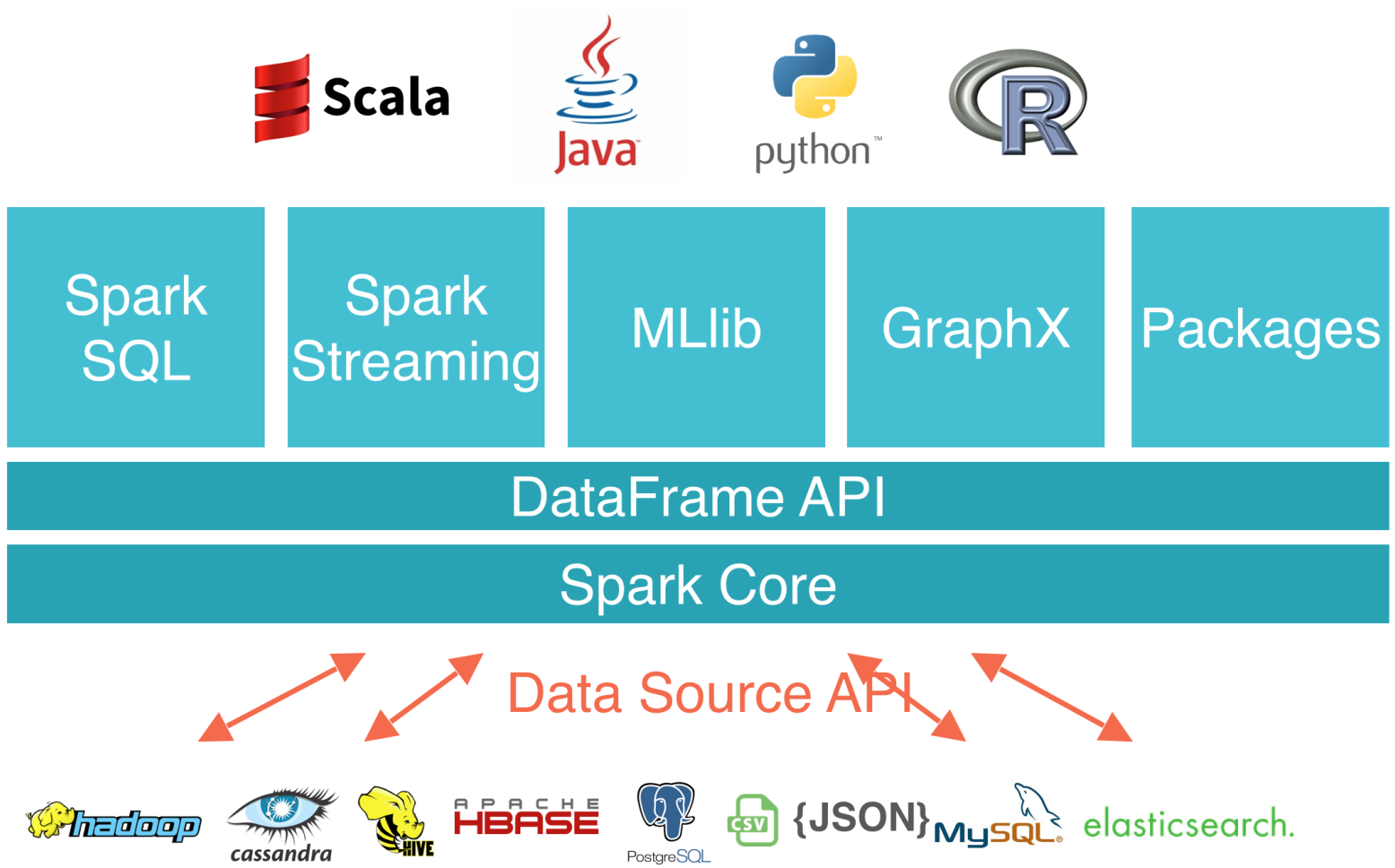

APACHE
Spark
TM

What is Apache Spark

- Fast and general engine for large-scale data processing
- Multi-stage in-memory primitives
- Supports Iterative Algorithms
- High-Level Abstractions
- Extensible; integrated stack of libraries



Broad Data, Language Support

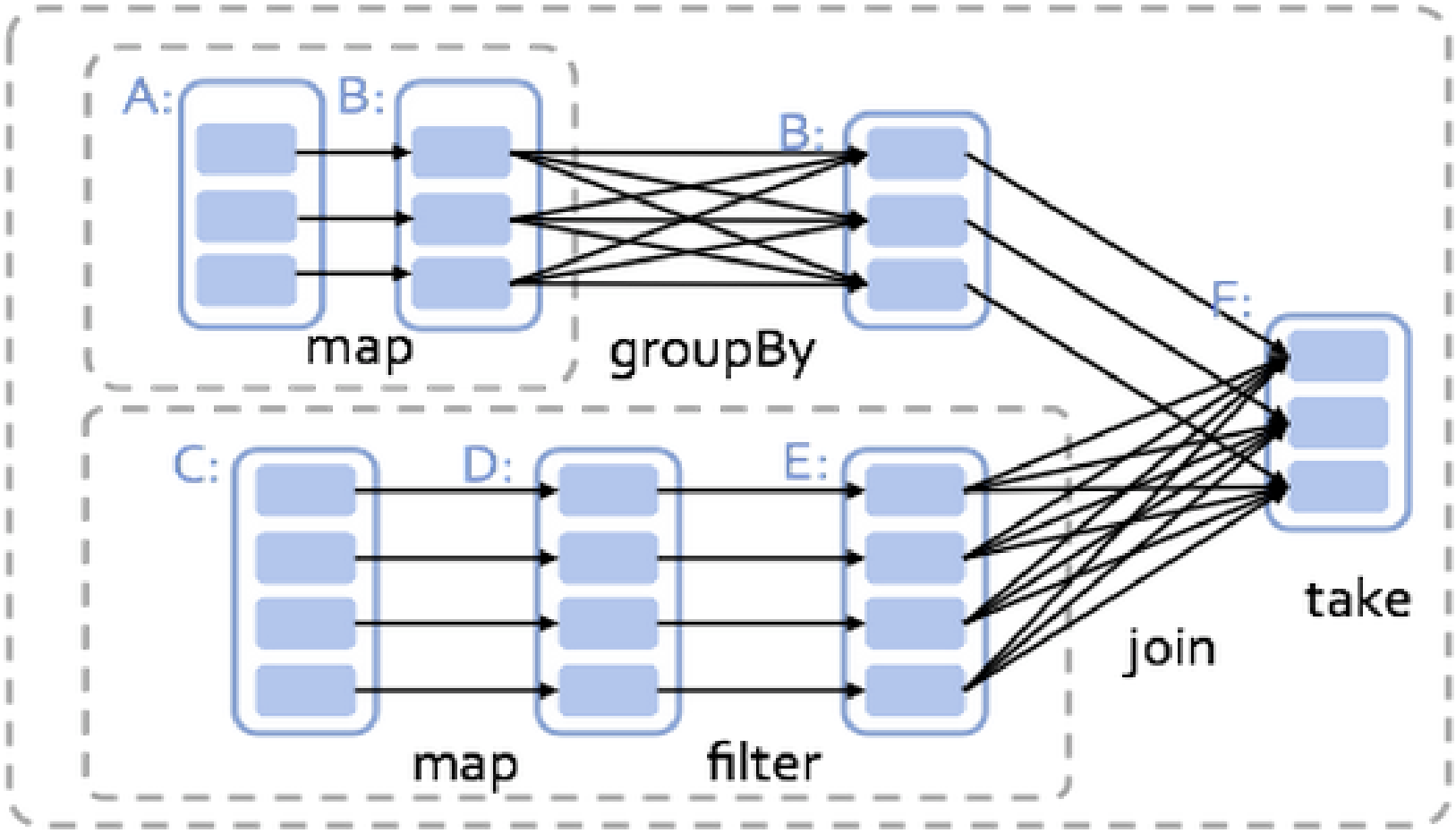


Spark Example

```
from pyspark import SparkContext
sc = SparkContext("local", "Log Analyzer")

rdd1 = sc.textFile('testfile1.txt')
rdd2 = sc.textFile('testfile2.txt')


def splitLine(line):
    key,val = line.split('|')
    return (key, int(val))

a = rdd1.filter(lambda line: not "ERROR" in line).map(splitLine)
b = rdd2.map(splitLine).reduceByKey(lambda a,b: a+b)

res = a.join(b)
#print res.toDebugString()
res.take(10)
```

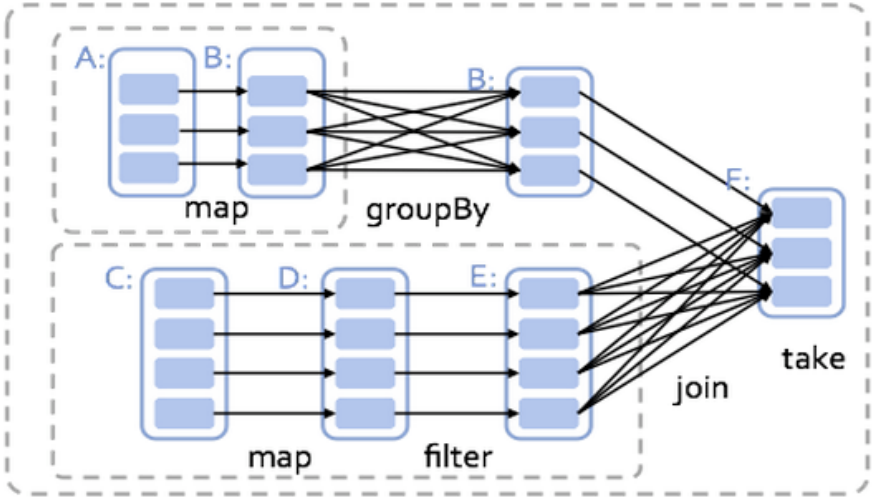
```
[(u'key3', (50, 30)), (u'key1', (20, 60))]
```

Operator Graph



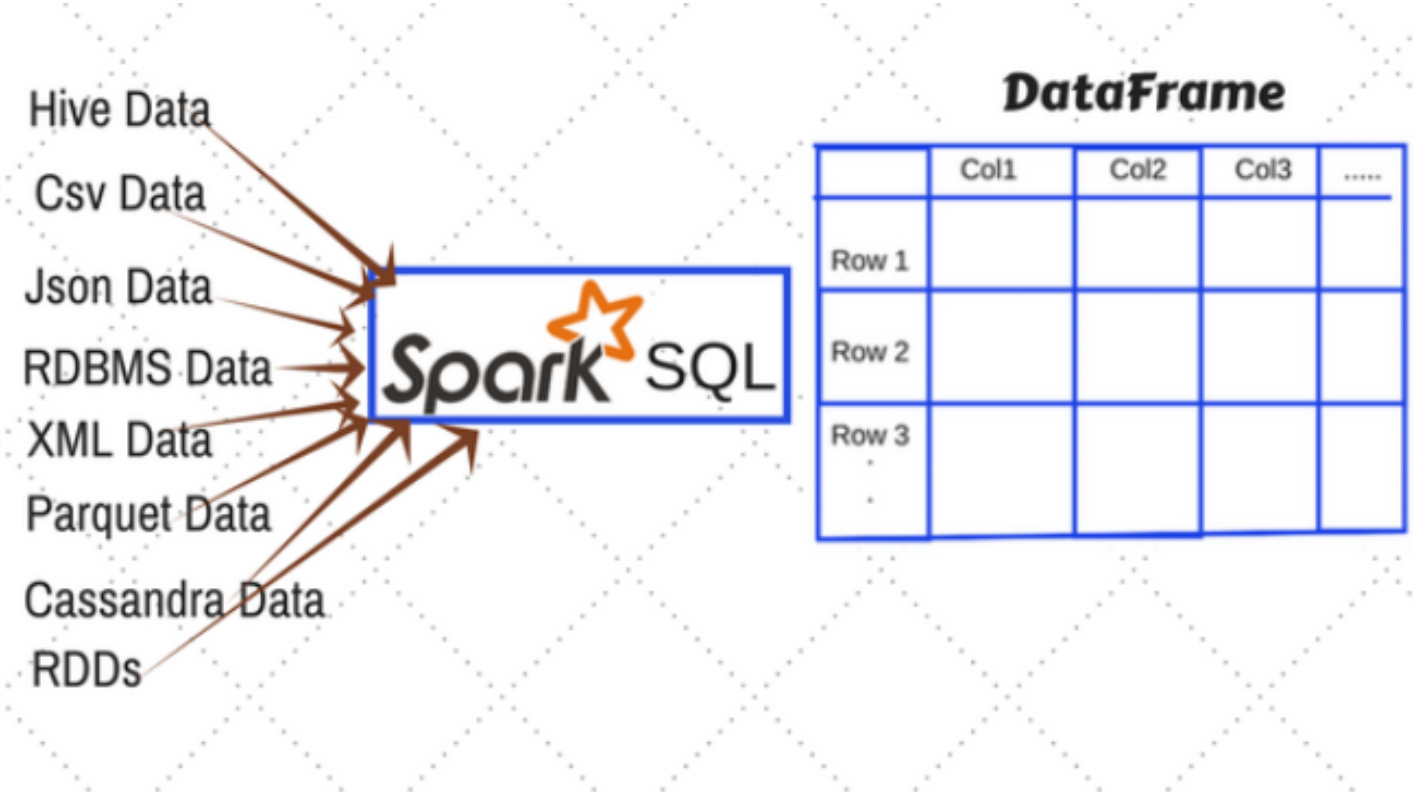
vs.

- Arbitrary operator graph
- Lazy eval of lineage graph => optimization
- Off-heap use of large memory
- Native integration with python

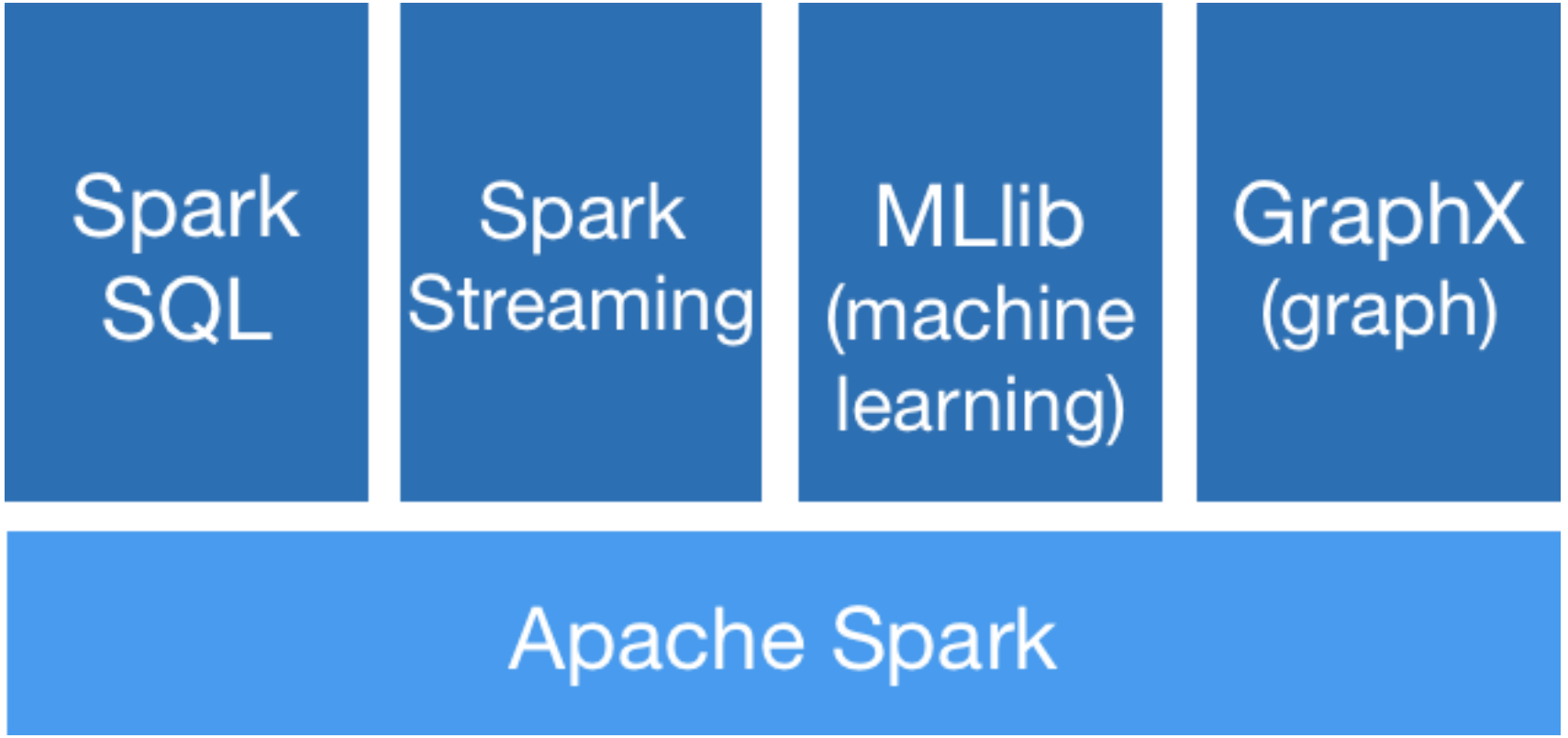


Spark DataFrame

- Distributed Collection
- Named Columns
- Rows, Column, Schema



Integrated Libraries



Spark Challenges 1 / 2

ExecutorLostFailure (executor 16 exited caused by one of the running tasks) Reason: Container killed by YARN for exceeding memory limits. 10.4 GB of 10.4 GB physical memory used. Consider boosting spark.yarn.executor.memoryOverhead.

Spark Challenges 2 / 2

```
17/09/05 16:40:33 ERROR Utils: Aborting task
java.lang.NullPointerException
          at org.apache.parquet.it.unimi.dsi.fastutil.objects.Object2IntLinkedOpenHashMap.getInt(Object2IntLinkedOpenHashMap.java:590)
          at org.apache.parquet.column.values.dictionary.DictionaryValuesWriter$PlainFixedLenArrayDictionaryValuesWriter.writeBytes(DictionaryValuesWriter.java:307)
          at org.apache.parquet.column.values.fallback.FallbackValuesWriter.writeBytes(FallbackValuesWriter.java:162)
          at org.apache.parquet.column.impl.ColumnWriterV1.write(ColumnWriterV1.java:201)
          at org.apache.parquet.io.MessageColumnIO$MessageColumnIORecordConsumer.addBinary(MessageColumnIO.java:467)
          at
org.apache.spark.sql.execution.datasources.parquet.ParquetWriteSupport$$anonfun$org$apache$spark$sql$execution$datasources$parquet$ParquetWriteSupport$$makeWriter$10.apply(ParquetWriteSupport.scala:1
84)
          at
org.apache.spark.sql.execution.datasources.parquet.ParquetWriteSupport$$anonfun$org$apache$spark$sql$execution$datasources$parquet$ParquetWriteSupport$$makeWriter$10.apply(ParquetWriteSupport.scala:1
72)
          at
org.apache.spark.sql.execution.datasources.parquet.ParquetWriteSupport$$anonfun$org$apache$spark$sql$execution$datasources$parquet$ParquetWriteSupport$$writeFields$1.apply$mcV$sp(ParquetWriteSupport.
scala:124)
          at
org.apache.spark.sql.execution.datasources.parquet.ParquetWriteSupport.org$apache$spark$sql$execution$datasources$parquet$ParquetWriteSupport$$consumeField(ParquetWriteSupport.scala:437)
          at org.apache.spark.sql.execution.datasources.parquet.ParquetWriteSupport.org$apache$spark$sql$execution$datasources$parquet$ParquetWriteSupport$$writeFields(ParquetWriteSupport.scala:123)
          at org.apache.spark.sql.execution.datasources.parquet.ParquetWriteSupport$$anonfun$write$1.apply$mcV$sp(ParquetWriteSupport.scala:114)
          at org.apache.spark.sql.execution.datasources.parquet.ParquetWriteSupport.consumeMessage(ParquetWriteSupport.scala:425)
          at org.apache.spark.sql.execution.datasources.parquet.ParquetWriteSupport.write(ParquetWriteSupport.scala:113)
          at org.apache.spark.sql.execution.datasources.parquet.ParquetWriteSupport.write(ParquetWriteSupport.scala:51)
          at org.apache.parquet.hadoop.InternalParquetRecordWriter.write(InternalParquetRecordWriter.java:123)
          at org.apache.parquet.hadoop.ParquetRecordWriter.write(ParquetRecordWriter.java:180)
          at org.apache.parquet.hadoop.ParquetRecordWriter.write(ParquetRecordWriter.java:46)
          at org.apache.spark.sql.execution.datasources.parquet.ParquetOutputWriter.write(Parquet
          at org.apache.spark.sql.execution.datasources.FileFormatWriter$DynamicPartitionWriteTask
          at org.apache.spark.sql.execution.datasources.FileFormatWriter$DynamicPartitionWriteTask
          at scala.collection.Iterator$class.foreach(Iterator.scala:893)
```

...78 more lines of Java stack trace

Takeaways

Spark...

- feels like native python, very nice API
- adds awesome Distributed Computing and Parallel Programming capabilities to python
- comes with batteries included (SQL, GraphX, MLLib, Streaming, etc.)
- can be used from the start for exploratory programming