The University of Texas at Austin
McCombs School of Business

MIS 381N INTRO. TO DATABASE MANAGEMENT

Big Data

Spark

Tayfun Keskin
Visiting Clinical Professor, The University of Texas at Austin, McCombs School of Business
Associate Teaching Professor, University of Washington Seattle, Foster School of Business

QUESTIONS

Any questions before we begin?

AGENDA

Lecture	**Big Data**
Discussion	**Article:** Why Big Data Isn't Enough
Looking Forward	**Exam 2** **HW5, Cases, Project**

THE 3Vs OF BIG DATA

VOLUME

- Amount of data generated
- Online & offline transactions
- In kilobytes or terabytes
- Saved in records, tables, files

VELOCITY

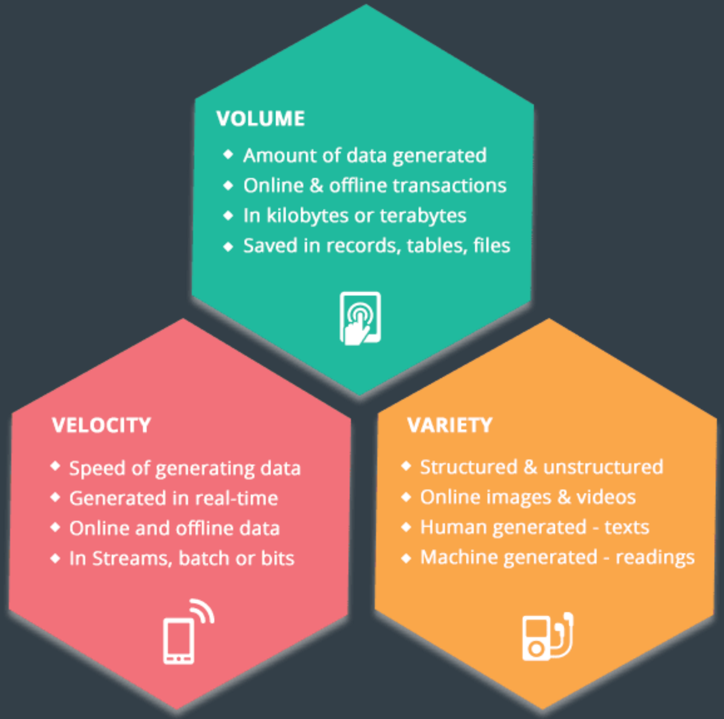
- Speed of generating data
- Generated in real-time
- Online and offline data
- In Streams, batch or bits

VARIETY

- Structured & unstructured
- Online images & videos
- Human generated - texts
- Machine generated - readings

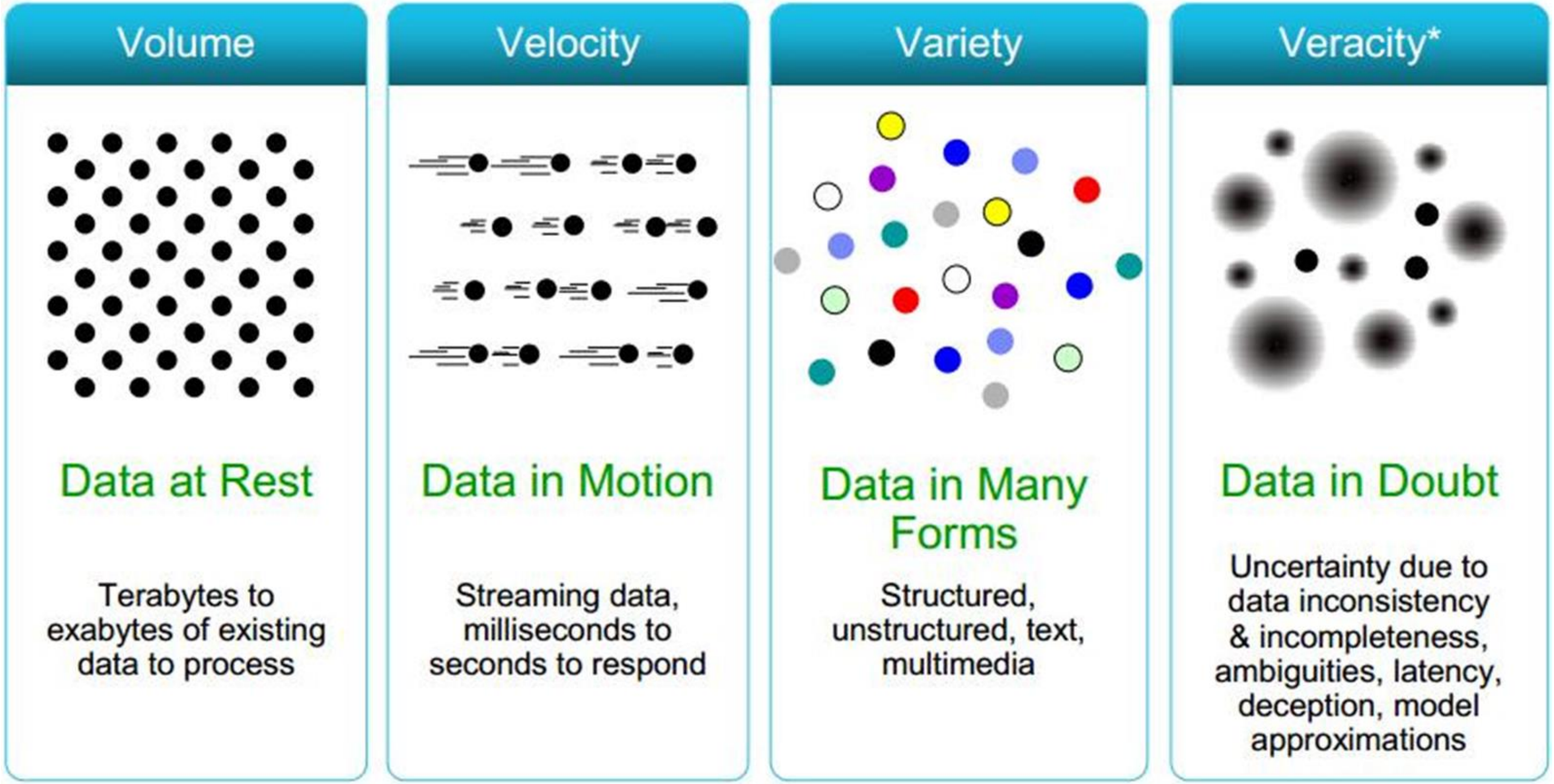
QUESTION

Can you think of other aspects for big data?



- Veracity? (uncertainty)
- Value?
- …?

ARE 4Vs MORE APPROPRIATE?



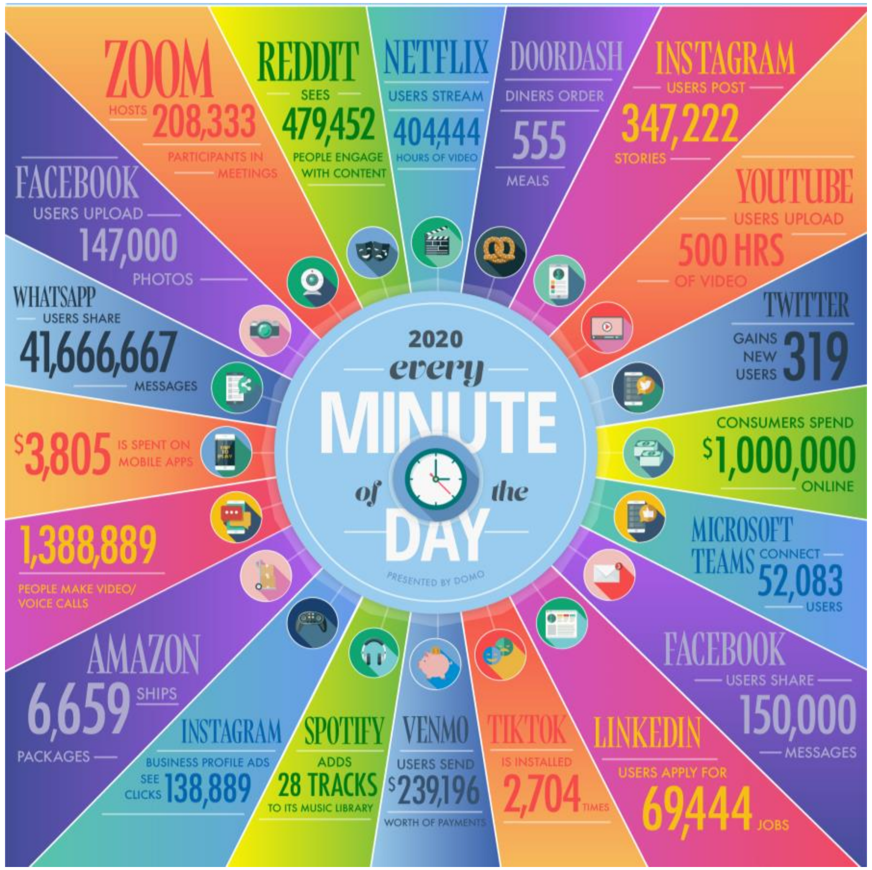
OR ARE THERE 5Vs?

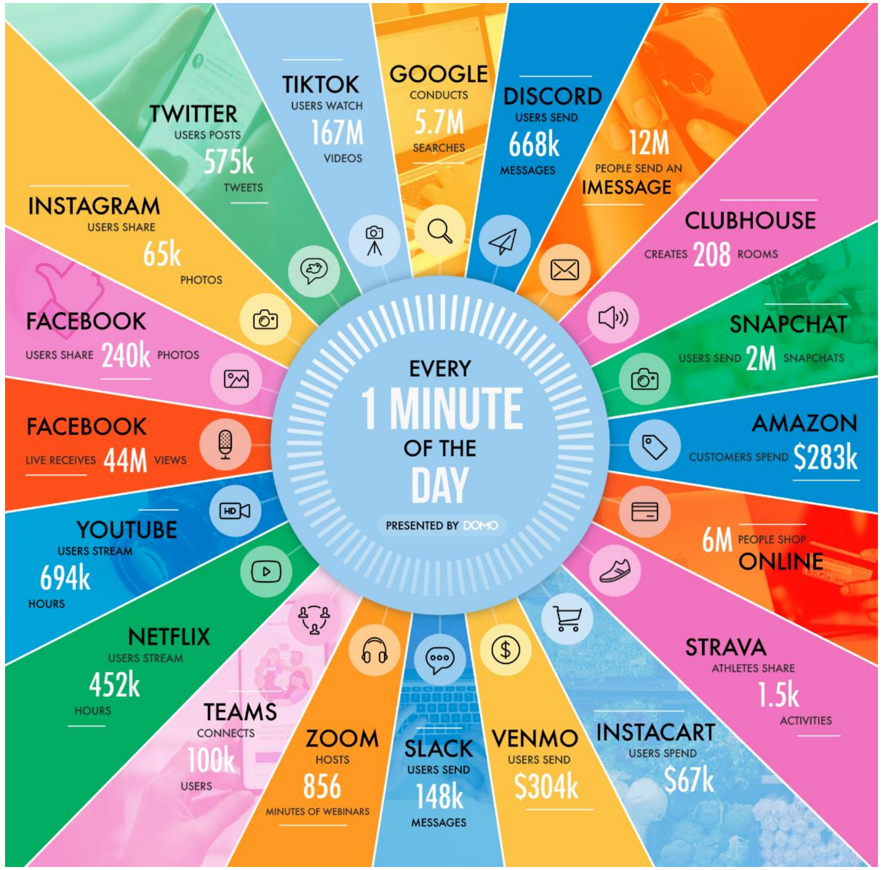
Value

Data is Money

All data does not produce the same value but some data produces direct value

DATA NEVER SLEEPS

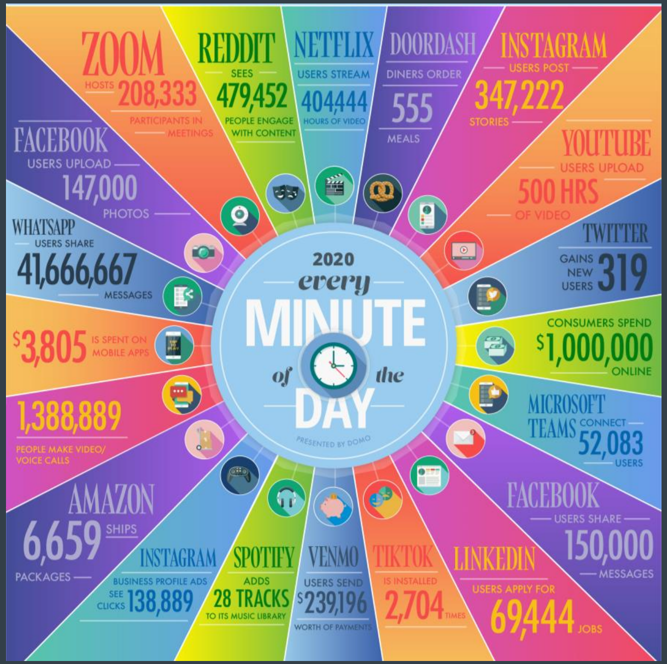




The University of Texas at Austin
McCombs School of Business

QUESTION

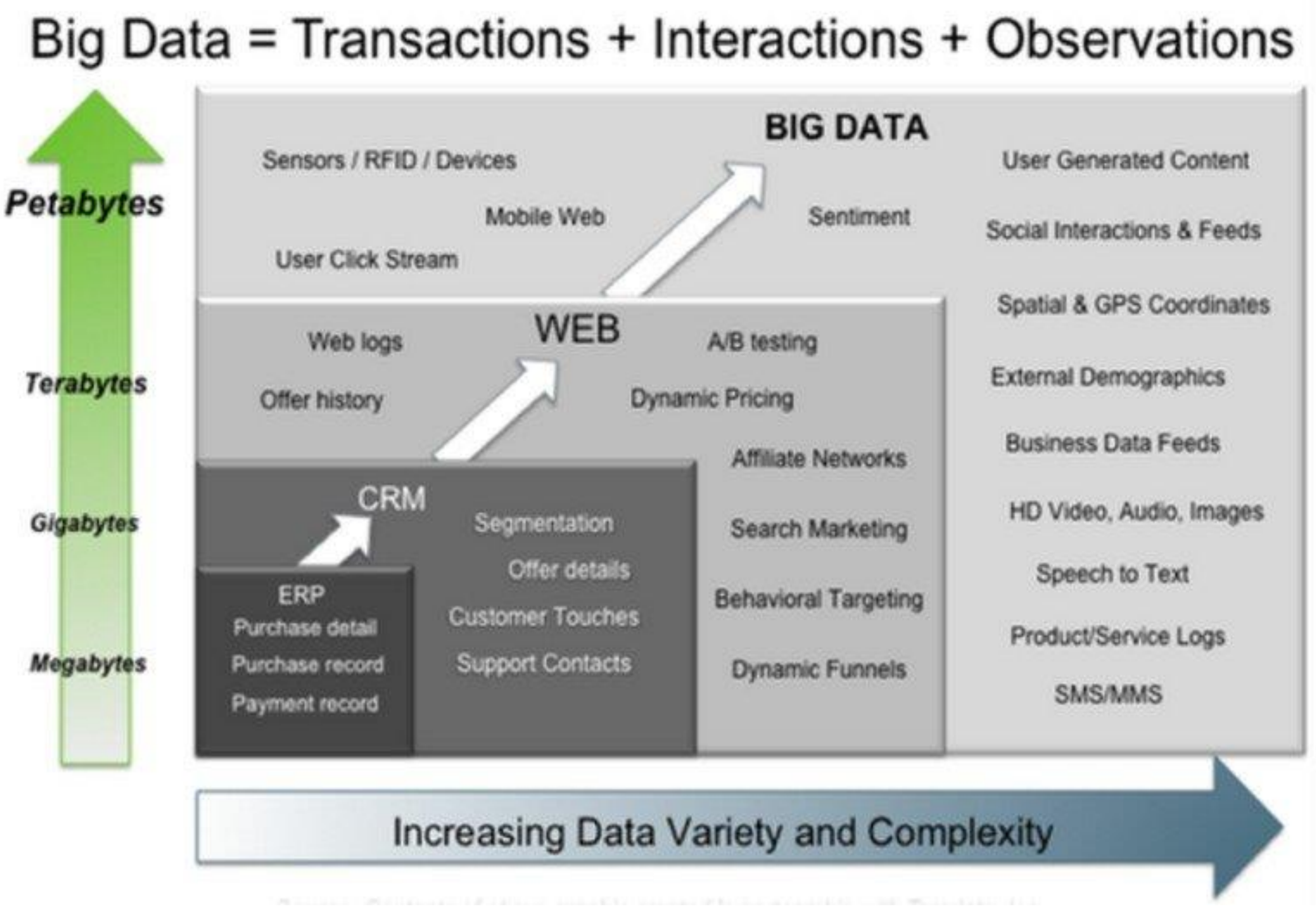
What will the graph for 2021 (or 2030) look like?



WHAT IS BIG DATA? (FROM WIKIPEDIA)

- Big data is the term for a collection of data sets so large and complex that it becomes difficult to process using on-hand database management tools or traditional data processing applications.
- The challenges include capture, curation, storage, search, sharing, transfer, analysis, and visualization.
- The trend to larger data sets is due to the additional information derivable from analysis of a single large set of related data, as compared to separate smaller sets with the same total amount of data, allowing correlations to be found to "spot business trends, determine quality of research, prevent diseases, link legal citations, combat crime, and determine real-time roadway traffic conditions."

IF YOU NEED TO DEFINE BIG DATA IN A SIMPLE EQUATION, THEN IT WOULD BE



TRANSACTIONS

- This is highly structured data related to events. It always includes: Time, a numerical value and refers to an objective, or objectives. Examples of this include, invoices, travel plans, activity records, payments, etc. The vast majority of this information is stored in databases and can be accessed quickly and easily, usually through SQL (Structured Query Language).

INTERACTIONS

- This covers how people interact with one another, or with your business. This includes interactions such as Facebook posts and Likes, social feeds, generated content and even blogs. Basically, this encompasses any data you can collect through any type of interaction that this isn’t limited to business transactions. As social networks become ever more integrated with our lives, the Interactions will play a key role in the Big Data Success Story.

OBSERVATIONS

- This is information gathered from the Internet of Things. The Internet of Things is associated with unique, individual things that have a virtual component that can be observed, and are connected in an Internet-like structure. Some examples of this include GPS coordinates from a person that visits your website on their mobile phone, or RFID chips in ATM cards. This data can be stored and potentially used to make better, more informed decisions.

QUESTION

Where do we get the most data today?

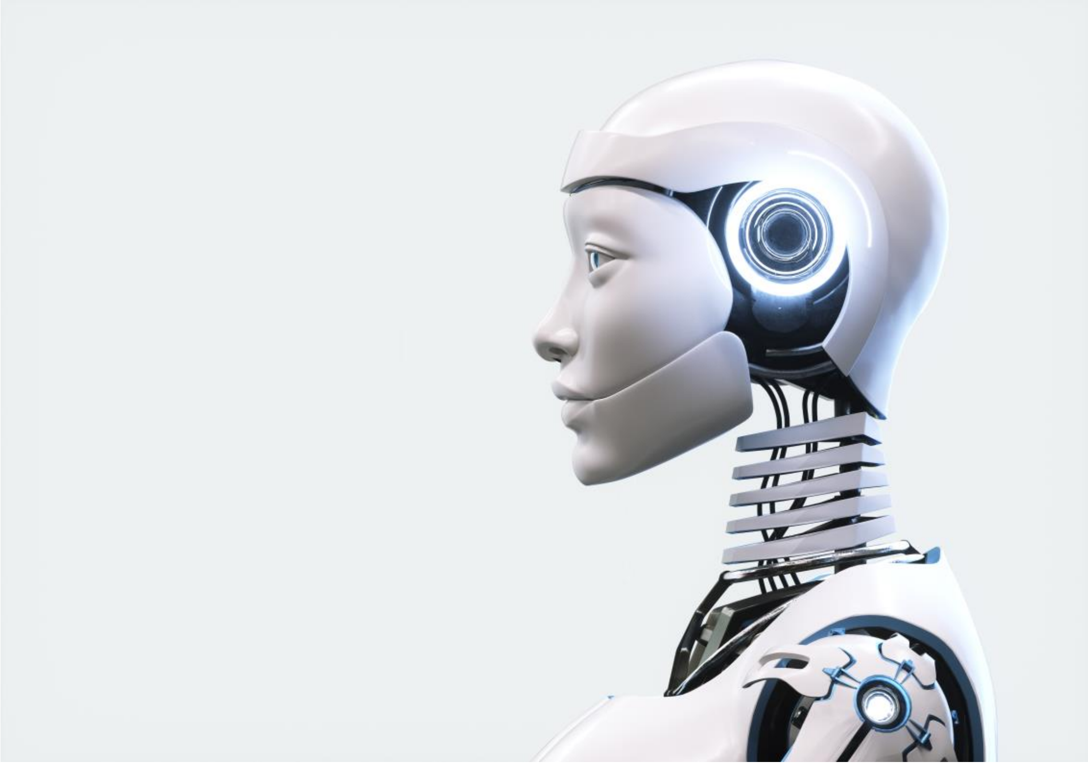
What is the value?

Where will we get the most data in 10 years?

INTERNET OF THINGS

ARTIFICIAL INTELLIGENCE

- Techniques that allow computers to do things typically done by humans
- Show “adaptability” or “intelligence”

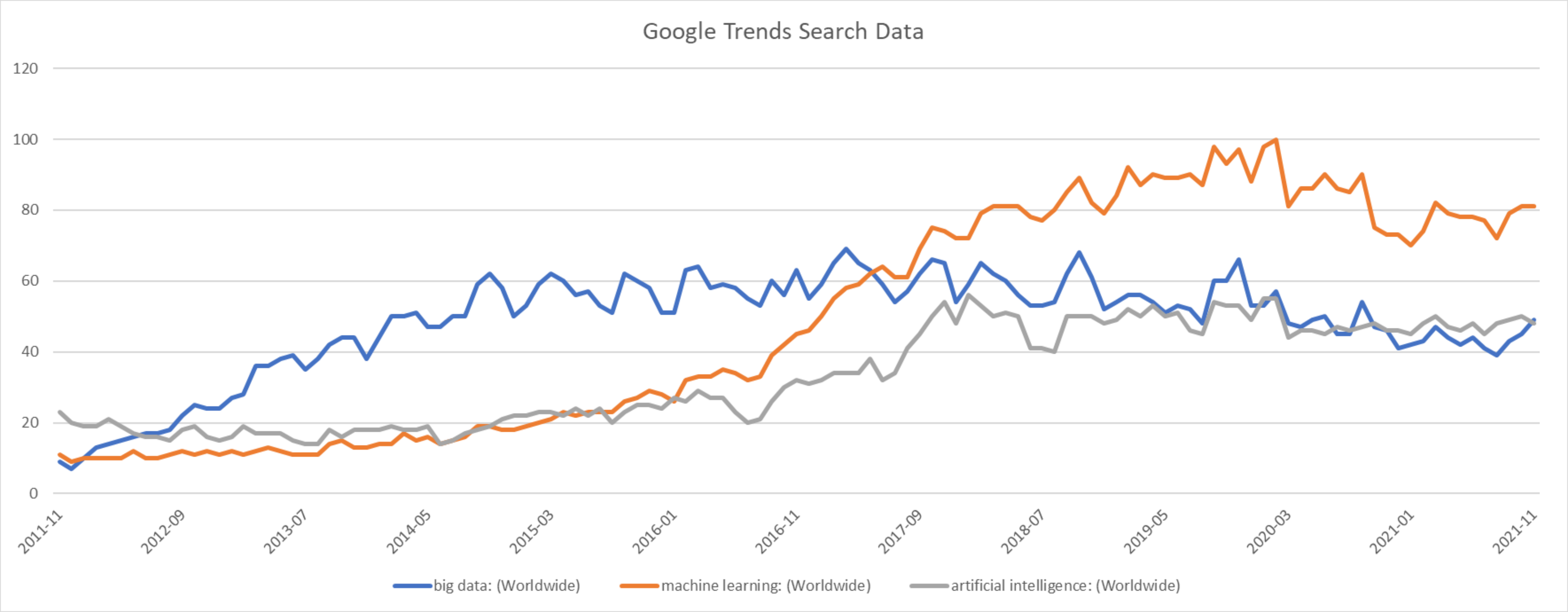


MACHINE LEARNING

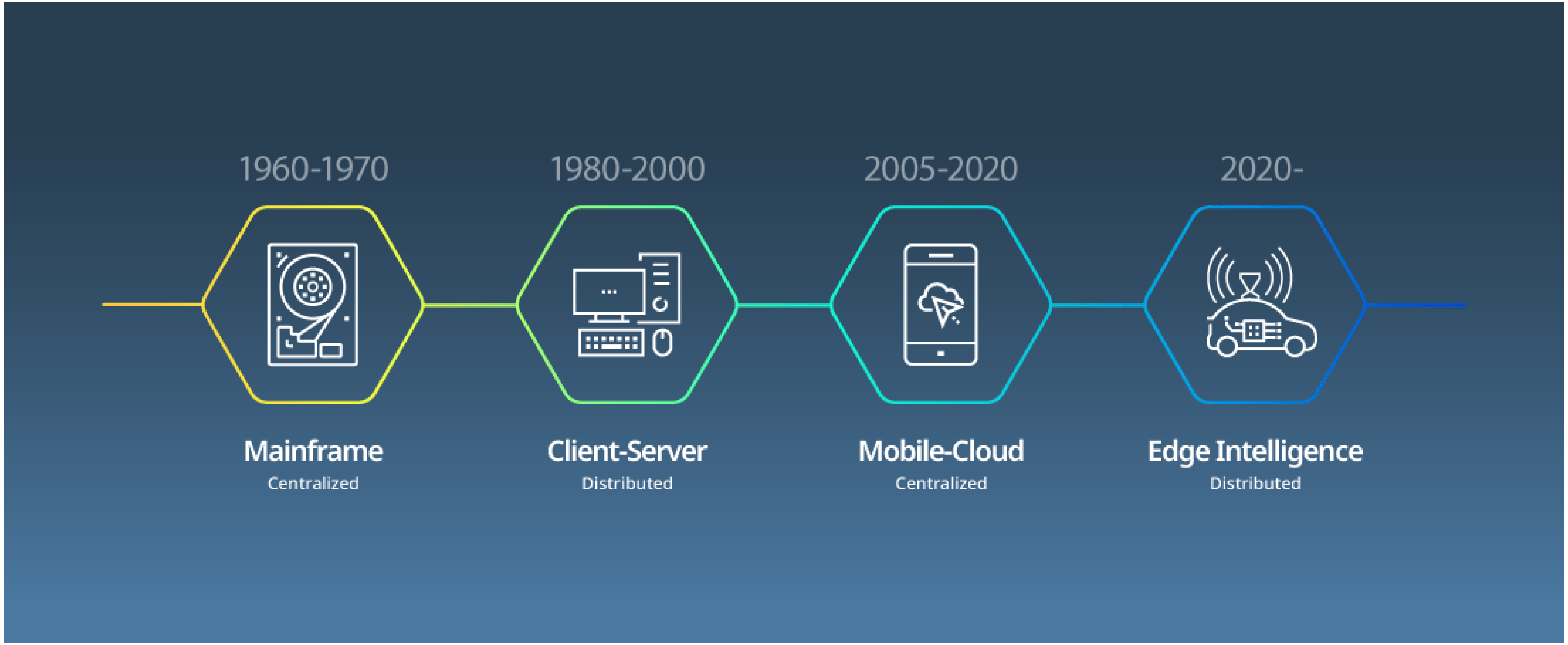
- Algorithms that can find patterns in data to predict outcomes
- Improve over time
    - Simple: regression
    - Complex: deep learning

ARE THEY THE SAME?

Big Data, Machine Learning, Artificial Intelligence



IS THERE A CYCLE?



CONCERNS LED TO EDGE/FOG

Low latency

Poor connectivity

Reduced server load

Privacy

HOW BIG DATA IS DIFFERENT

- Big data is not just quantitatively larger
  1. Small data has a specific goal, big data goals evolve
  2. Small data is usually in one place, big data is distributed
  3. Small data is typically in a table, big data can be unstructured
  4. Small data is prepared by the end user, big data is a team sport
  5. Small data is kept for a short time, big data may be perpetual

HOW BIG DATA IS DIFFERENT

6. Small data is measured in standardized units, big data can come in many formats gathered with many protocols
7. Small data is reproducible, big data replication may not be possible
8. Small data risks are limited, big data can make or break an industry
9. Small data is introspective (organized, easy to locate, with clear metadata), big data can be difficult to locate or interpret
10. Small data can be analyzed at once, big data need to be broken apart

ANALYZING BIG DATA

- Data cleaning / wrangling takes time
  - 80 percent of a typical big data project is spent on preparing the data
- Visualization
- Data mining
- Sentiment analysis
- Predictive analytics

PRIMER ON DISTRIBUTED SYSTEMS

DEFINITION

- The primary goal of distributed systems is to make multiple independent machines (i.e., computers) **interconnected** through a network **coordinate** in a **coherent** way to achieve a **common goal**.
- Optimist: A distributed system is a collection of independent computers that appears to its users as a single coherent system.
- Pessimist: “You know you have one when the crash of a computer you’ve never heard of stops you from getting any work done.” (Lamport)

DISTRIBUTED SYSTEM BENEFITS AND ISSUES

Real world issue	Distributed system goal	Distributed system approach
Software complexity, data generated & used, number of users is increasing and unpredictable	Scalability – overcome expensive network coordination, insufficient parallelism & bottlenecks	Partitioning of data and computational, special purpose (lightweight) processes
Failures – network, computer, data center, software – are inevitable	Fault tolerance – availability, durability, uncertain information	Replication – redundant copies of data and processes
Consistency becomes a problem with replicated redundant systems	Meaningful consistency semantics – eventual consistency	Rigorous agreement protocols – commitment, consensus

BASIC TERMS

- **Fault (or Failure):** Any error in the system, including device failures, software failures (including bugs) and protocol failures.
  This is true for any computing system.
- **Partition:** Any disruption in the network between nodes
- **Partition Tolerance:** System is resilient/robust to partitions
- **Availability:** Data/State always observable/measurable
- **Scaling/Scalability:** Operations scale regardless of number of nodes in the network
- **Replicas:** Replication of data and/or state is an essential component of distributed systems. The mechanism of sharing data and/or state defines the distributed system

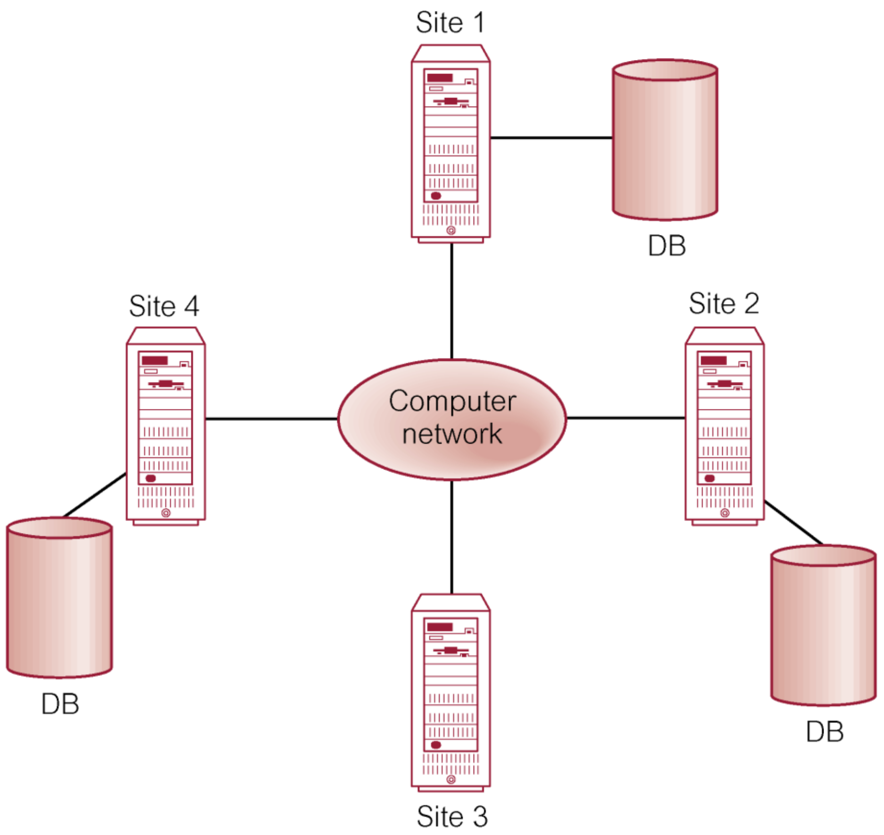
BASIC TERMS

- **Node:** Any participant in a distributed system
- **Network:** Connectivity among nodes in a distributed system
- **Atomicity:** An atomic transaction is an indivisible and irreducible series of operations such that either all occur, or nothing occurs
- **Consistency:** After each operation, all replicas reach the same state
- **Strict Consistency:** Immediate consistency after each operation
- **Eventual Consistency:** Consistent over time, but can take an arbitrary amount of time
- **Liveness:** People actually use the distributed system regularly. If left pristine with wrapping paper, it is a perfect system . Liveness means active use

TRANSPARENCY IN DISTRIBUTED DATABASES

Transparency	Description
Access	Hide differences in data representation and how a resource is accessed
Location	Hide where a resource is located
Migration	Hide that a resource may move to another location
Relocation	Hide that a resource may be moved to another location while in use
Replication	Hide that a resource may be shared by several competitive users
Concurrency	Hide that a resource may be shared by several competitive users
Failure	Hide the failure and recovery of a resource
Persistence	Hide whether a (software) resource is in memory or on disk

BASIC DISTRIBUTED DATA TOPOLOGY



The University of Texas at Austin
McCombs School of Business

DISTRIBUTED DATA MANAGEMENT SYSTEM CONCEPTS

- A distributed data management system manages a collection of logically-related shared data.
- These data are **partitioned** into **fragments**.
- These fragments may be **replicated**.
- Fragments/replicas are allocated to **sites**.
- Sites are linked by a communication network.
- Data at each site is under the control of a database management system.
- This database management system handles local applications autonomously.
- The distributed data management system manages
  - Fragmentation, replication and synchronization of data across all sites
  - Orchestration of read requests across all sites

FRAGMENTATION AND REPLICATION STRATEGIES

	Locality of reference	Reliability and availability	Performance	Storage costs	Communication costs
Centralized	Lowest	Lowest	Unsatisfactory	Lowest	Highest
Fragmented	High[a]	Low for item; high for system	Satisfactory[a]	Lowest	Low[a]
Complete replication	Highest	Highest	Best for read	Highest	High for update; low for read
Selective replication	High[a]	Low for item; high for system	Satisfactory[a]	Average	Low[a]

[a] Indicates subject to good design.

ACID (transaction) SEMANTICS

- Atomicity – an operation is performed on all replicas or not performed at all
- Consistency – after each operation all replicas have the same state
- Isolation – no operation can see data from another operation in an intermediate state
- Durability – once a write have been successful, data will persist indefinitely

QUESTION

What are the ACID (atomicity, consistency, isolation, durability) requirements for a Facebook news feed?

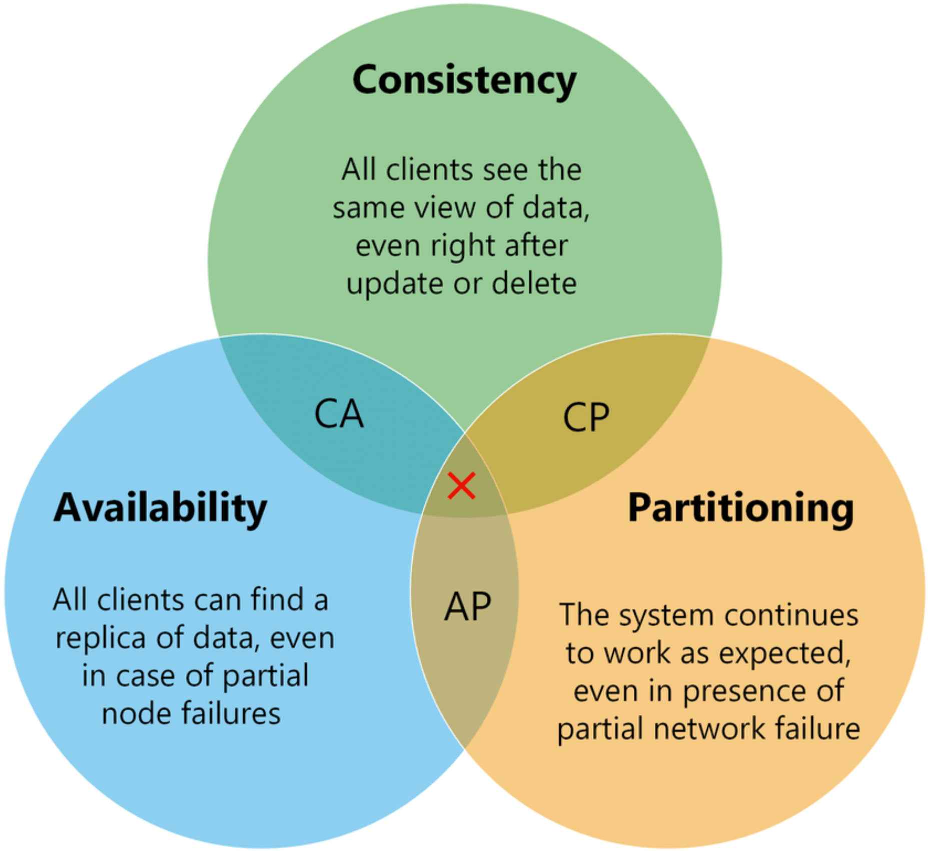
BASE SEMANTICS

- Basically available – users perceive the system as available
- Soft state – the state of the system may change over time even without any input
- Eventually consistent – given enough time and no input all replicas will become consistent

CAP THEOREM (Brewer)

- A system can only achieve two of the three goals of:

1. Strong Consistency
2. High Availability
3. Partition tolerance.



CONSISTENCY AND AVAILABILITY

- Providing transactional semantics requires all functioning nodes to be in contact with each other (no partition)
- Examples:
  Single-site and clustered databases
  Other cluster-based designs
- Typical features:
  Two-phase commit
  Cache invalidation protocols
  Classic DS style

PARTITION-TOLERANCE AND AVAILABILITY

- Once consistency is sacrificed, life is easy …
- Examples:

  DNS

  Web cache

  Mobile environments
- Typical features:

  Optimistic updating with conflict resolution

  “Internet design style”

  TTLS and lease cache management

IMPLEMENTING EVENTUAL CONSISTENCY

1. All writes eventually propagate to all replicas
2. Writes, when they arrive, are written to a log and are applied in the same order at all replicas

   Easily done with timestamps and “reversing” optimistic writes

CONFLICT RESOLUTION

- Replication not transparent to the application

  Only application knows how to resolve conflicts

  Application can do record-level conflict detection, not just file-level conflict detection

- Split of responsibility

  Replication system: propagates updates

  Application: resolves conflict

- Optimistic application of writes requires that writes be “reversible”

SUMMARY… (AND GOOD NEWS)

- We can assume that the distributed data management systems are under the covers taking care of partitioning, fragmentation, replication and synchronization
- Distributed data management systems are available for relational and non-relational databases—they are however the norm for non-relational (NoSQL) data stores.
- For transactional applications using NoSQL data stores, developers need to design the data model and application to account for the ACID requirements they need to meet
- For analytics, we do not need to worry about ACID requirements

ARTICLE: WHY BIG DATA ISN'T ENOUGH

QUESTION

Can you give me a summary / synopsis of the article?

WHAT DO YOU THINK?

This article have been very controversial in my field and overall research area

MITSloan Management Review

MENU

MAGAZINE WINTER 2017 ISSUE • RESEARCH FEATURE

Why Big Data Isn't Enough

There is a growing belief that sophisticated algorithms can explore huge databases and find relationships independent of any preconceived hypotheses. But in businesses that involve scientific research and technological innovation, the authors argue, this approach is misguided and potentially risky.

Sen Chai and Willy Shih • November 14, 2016

QUESTION

Have you ever run a regression in a large dataset?

- If yes, what were the correlation values?
- How do spurious correlations emerge in open-ended search?

SCIENTIFIC HYPOTHESIS

- An idea that proposes a tentative explanation about a phenomenon or a narrow set of phenomena observed in the natural world
  1. **FALSIFIABILITY**
  2. **TESTABILITY**
- …reflected in an “If…then” statement summarizing the idea and, in the ability to be supported or refuted through observation and experimentation

IT'S NOT ALL BAD

How else can we use big data analytics?

- Strengthening existing models?
- Creating new models?

LOOKING FORWARD

- **Exam 2**
- Homework 5
- Cases
- Final Project

THANK YOU

BACKUP SLIDES

PART 1

Spark

SQL → MapReduce → …

Understanding the progression of tools …

Data Interaction
Data Management
Data Model
Compute/ Storage Infrastructure

Understanding the progression of tools ...

	Structured Data
Data Interaction	Structured Query Language (SQL)
Data Management	Relational Database Management System (RDBMS)
Data Model	Entity-Relationship Tables, columns, rows
Compute/ Storage Infrastructure	Centralized, Proprietary physical storage, Vertical scaling

Understanding the progression of tools …

	Structured Data	Unstructured Data
Data Interaction	Structured Query Language (SQL)	Python, Java, …
Data Management	Relational Database Management System (RDBMS)	MapReduce Framework of Stack
Data Model	Entity-Relationship Tables, columns, rows	Key-value pairs, documents, graphs, …
Compute/ Storage Infrastructure	Centralized, Proprietary physical storage, Vertical scaling	Distributed, File based non-proprietary storage, horizontal scaling

Understanding the progression of tools …

	Structured Data	Unstructured Data	
Data Interaction	Structured Query Language (SQL)	Pig, Hive, Python, Java, …	• In the real-world, data is structured and unstructured. • Should we use disparate tools or should tools be integrated? • Should the compute/ storage structure be integrated?
Data Management	Relational Database Management System (RDBMS)	MapReduce Framework of Stack	
Data Model	Entity-Relationship Tables, columns, rows	Key-value pairs, documents, graphs, …	
Compute/ Storage Infrastructure	Centralized, Proprietary physical storage, Vertical scaling	Distributed, File based non-proprietary storage, horizontal scaling	

Apache Big Data Ecosystem (partial)

Data Interaction	Pig, Hive, Impala	
Data Management	Computing Framework (MapReduce, Spark, Storm) Distributed Resource Management (YARN, ZooKeeper, Oozie)	NoSQL Database (Hbase)
Compute/ Storage Infrastructure	Hadoop Distributed File System (HDFS)	

MapReduce Issues

- All data processing tasks need to be decomposed into Map and Reduce steps
- “Acyclic Data Flow” from Disk to Disk (HDFS)
- Read and write to disk before and after Map and Reduce, making it inefficient for iterative tasks such as the ones required for Machine Learning algorithms
- Efficient for streaming data, but less efficient for interactivity and batch processing

SQL → MapReduce → Spark

Spark facts

- A general framework for distributed computing leveraging the Hadoop ecosystem
- In-memory caching of data for efficient iterative, graph, and other types of tasks needed to support machine learning algorithms
- Supports interactive data analysis required for exploratory data analysis
- High level and low level APIs encourages integration
- Native Scala – supports Java, Python, SQL and R
- Developed at AMPLab UC Berkeley and is currently supported commercially by Databricks

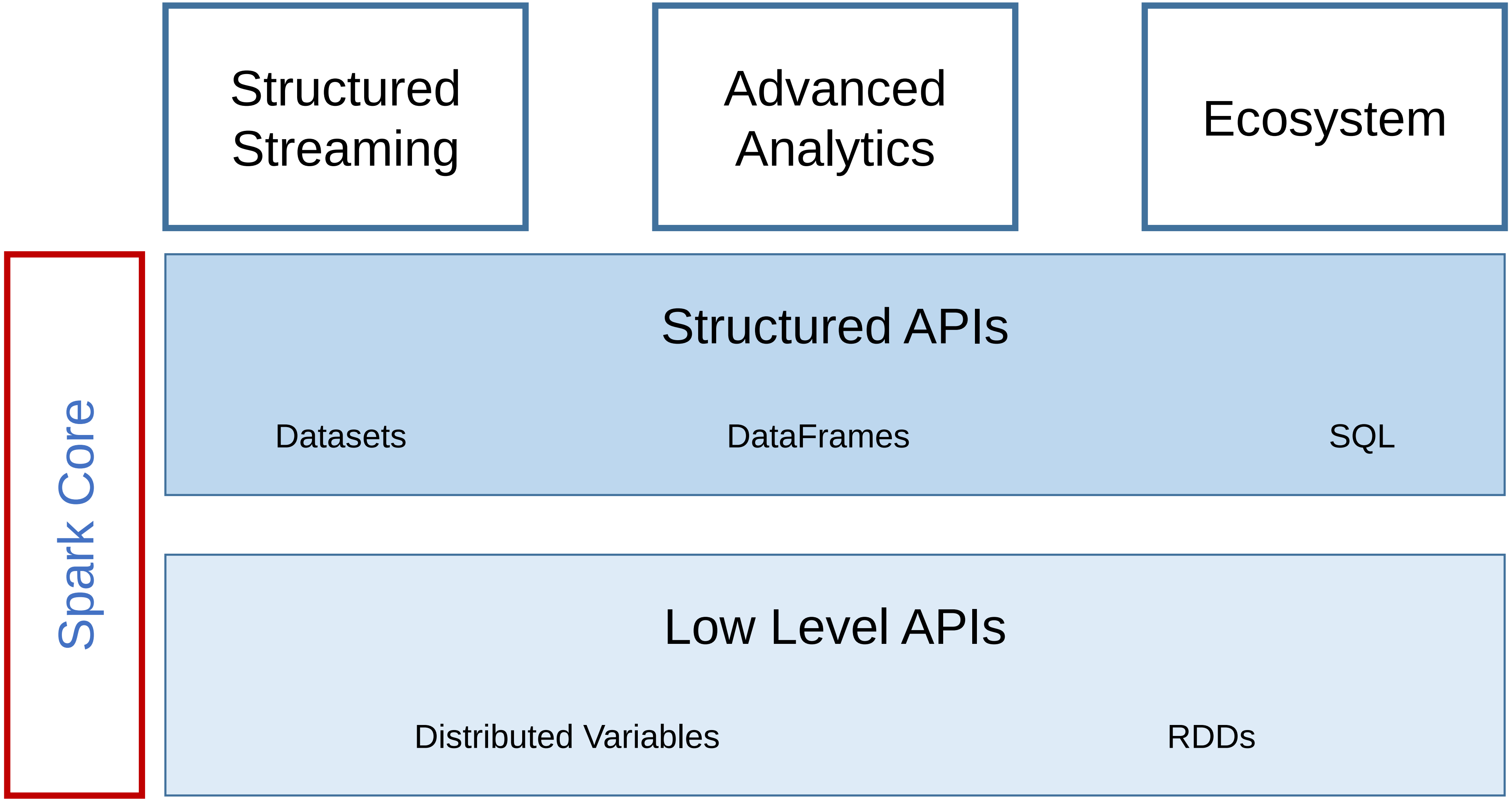
Production applications of Spark

- Uber gathers terabytes of event data from its mobile users every day
  - Continuous ETL pipeline using Kafka, Spark Streaming, and HDFS
  - Raw unstructured event data converted to structured data in real-time
  - These data are used for complex analytics and operations optimization
- Pinterest – Uses a Spark ETL pipeline
  - Leverages Spark Streaming to gain immediate insight into how users all over the world are engaging with Pins—in real time.
  - Can make more relevant recommendations as people navigate the site
  - Recommends related Pins
  - Determine which products to buy, or destinations to visit

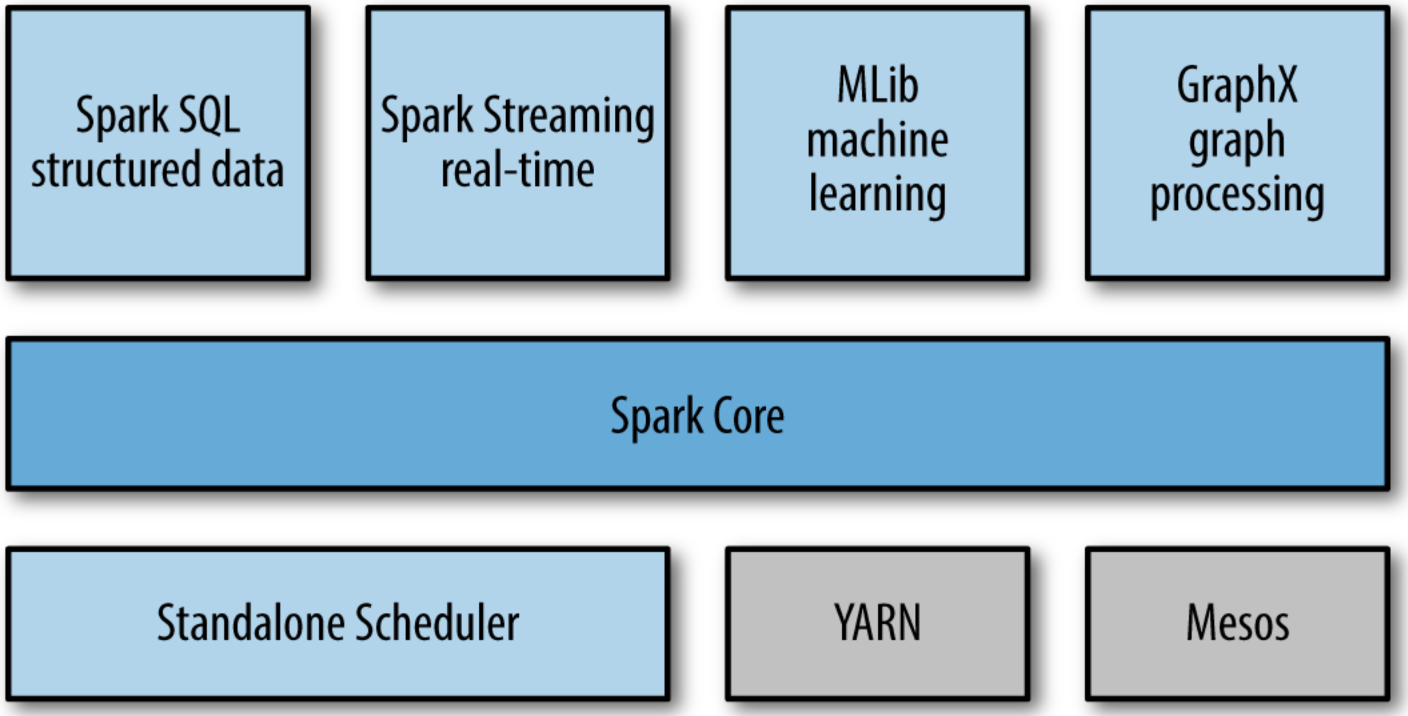
Production applications of Spark

- Conviva, streaming video company second only to YouTube with 4 million video feeds per month
  - Uses Spark to reduce customer churn by optimizing video streams and managing live video traffic
  - Maintains a consistently smooth, high quality viewing experience
- Capital One
  - Uses Spark and data science algorithms to understand customer behavior, develop new financial products and services and find attributes and patterns that point to an increased probability for fraud
- Netflix
  - Leverages Spark to understand viewing habits, recommends content and drive new content creation

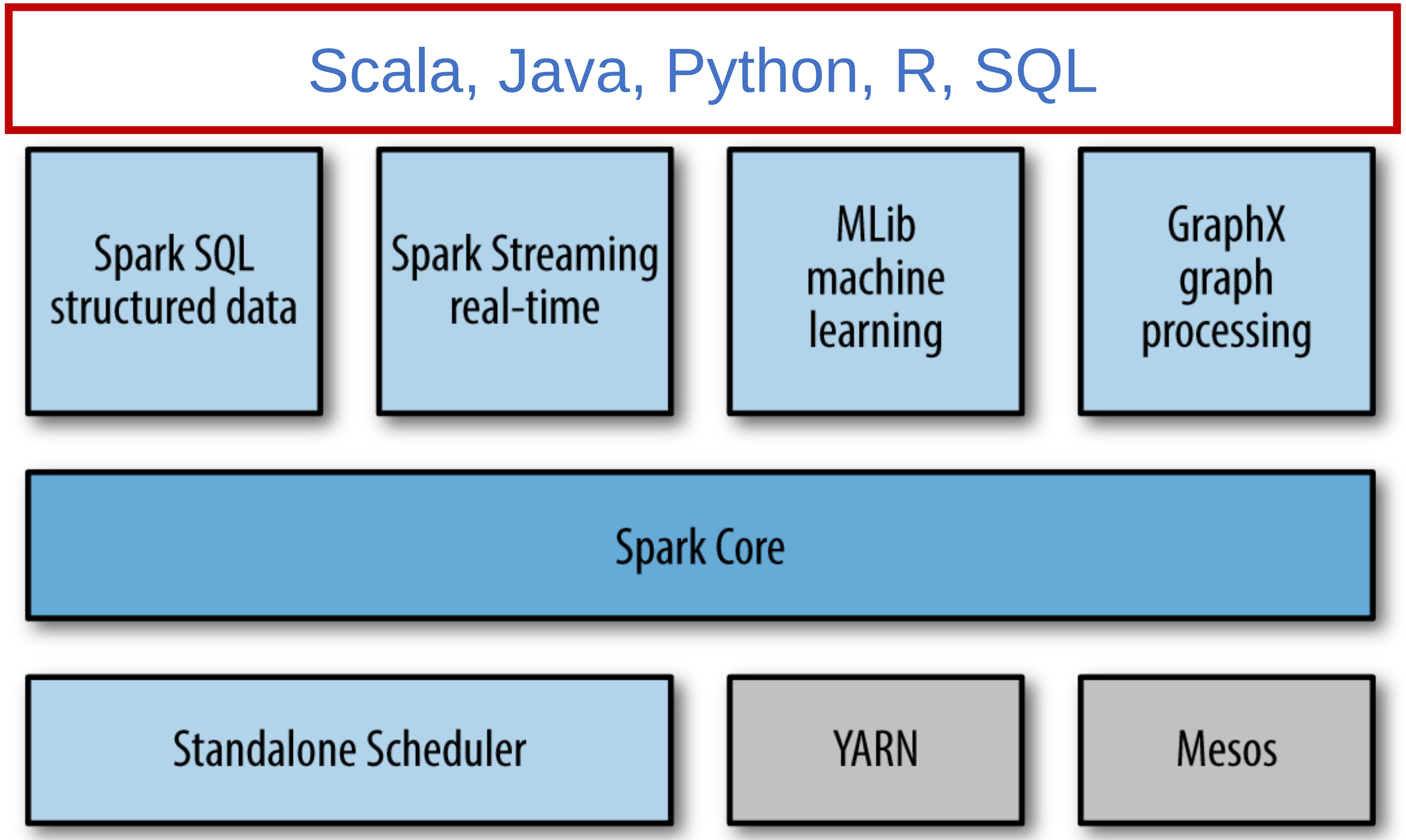
Spark High Level Structure



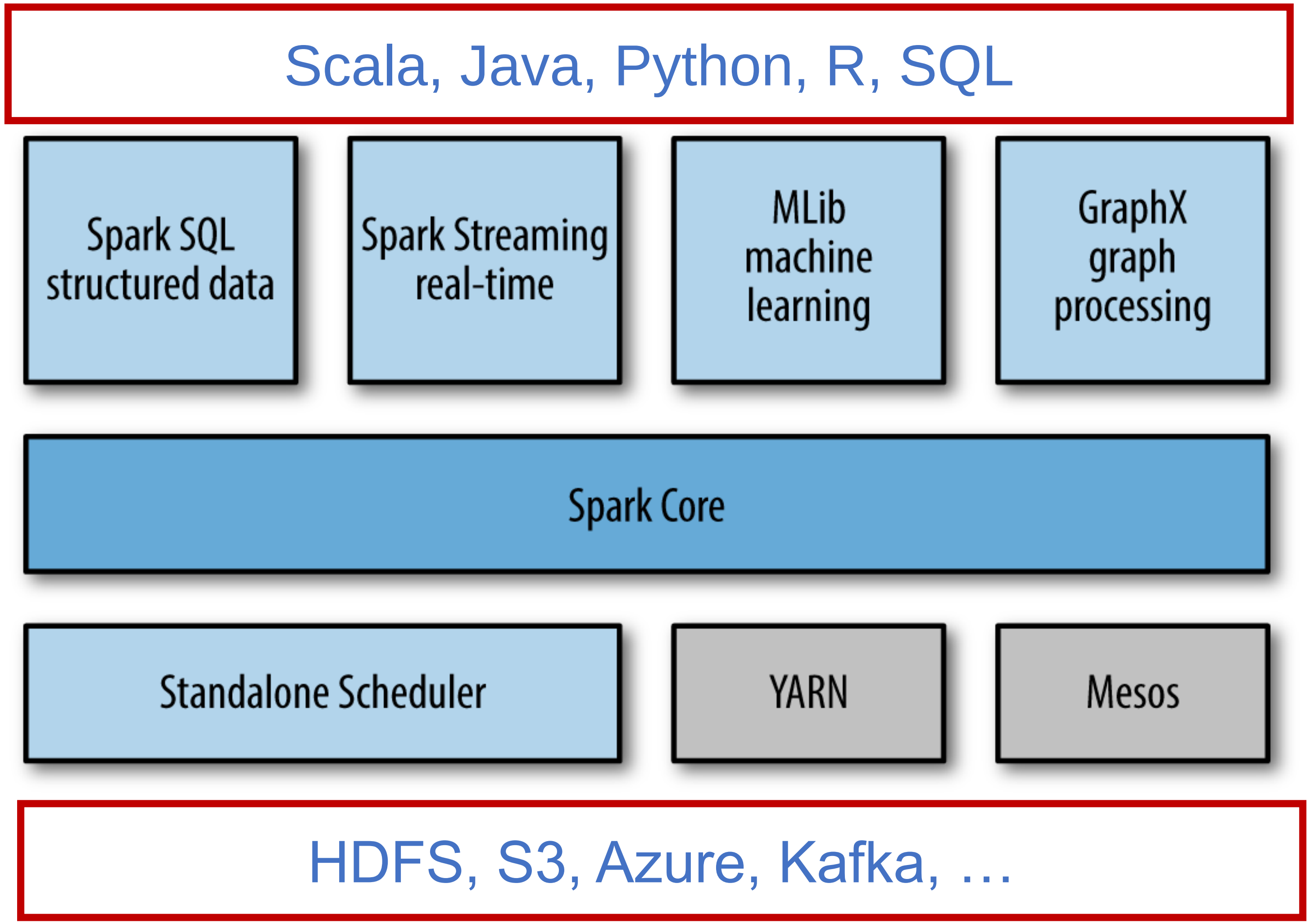
Spark Components



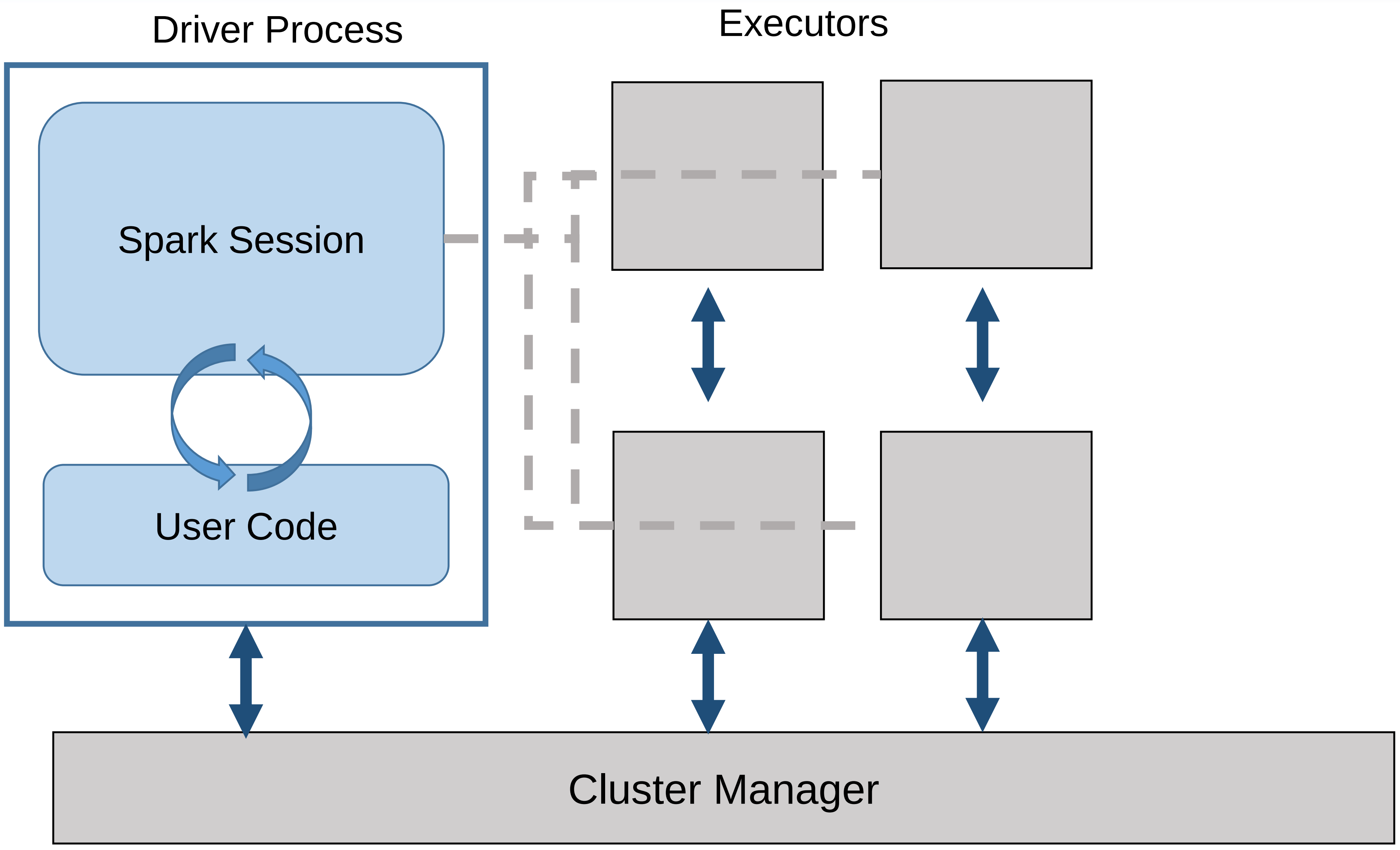
Spark API Language Support



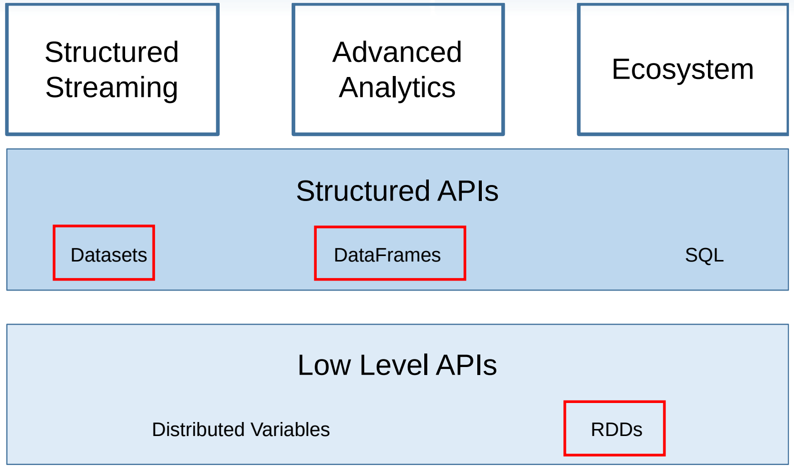
Spark Storage Support



Spark Architecture



Data Representation in Spark



- Resilient Distributed Datasets (RDDs)
  - Fundamental data structure of Spark
  - Read-only partition collection of records.
  - A programmer can perform in-memory computations on large clusters in a fault-tolerant manner
  - Efficient and performant
- DataFrame API
  - Data organized into named columns
  - Immutable distributed collection of data.
  - A programmer can impose structure onto a distributed collection of data, allowing higher-level abstraction
- Dataset:
  - Extension of DataFrame API
  - Type-safe, object-oriented programming interface

Resilient Distributed Datasets (RDDs)

- RDDs (Resilient Distributed Datasets) play the role of data containers
- All the different processing components in Spark share the same abstraction called RDD
- As applications share the RDD abstraction, you can mix different kind of transformations to create new RDDs
- Created by parallelizing a collection or reading a file
- Fault tolerant

DataFrames

- DataFrames (DFs) are distributed datasets organized in named columns
- They are similar to tables in a relational database, Python Pandas Dataframe or DataTables in R
  - Immutable once constructed
  - Track lineage
  - Enable distributed computations
- How to construct DataFrames?
  - Read from file(s)
  - Transform an existing DF(Spark or Pandas)
  - Parallelizing a python collection list
  - Apply transformations and actions

DataFrame examples

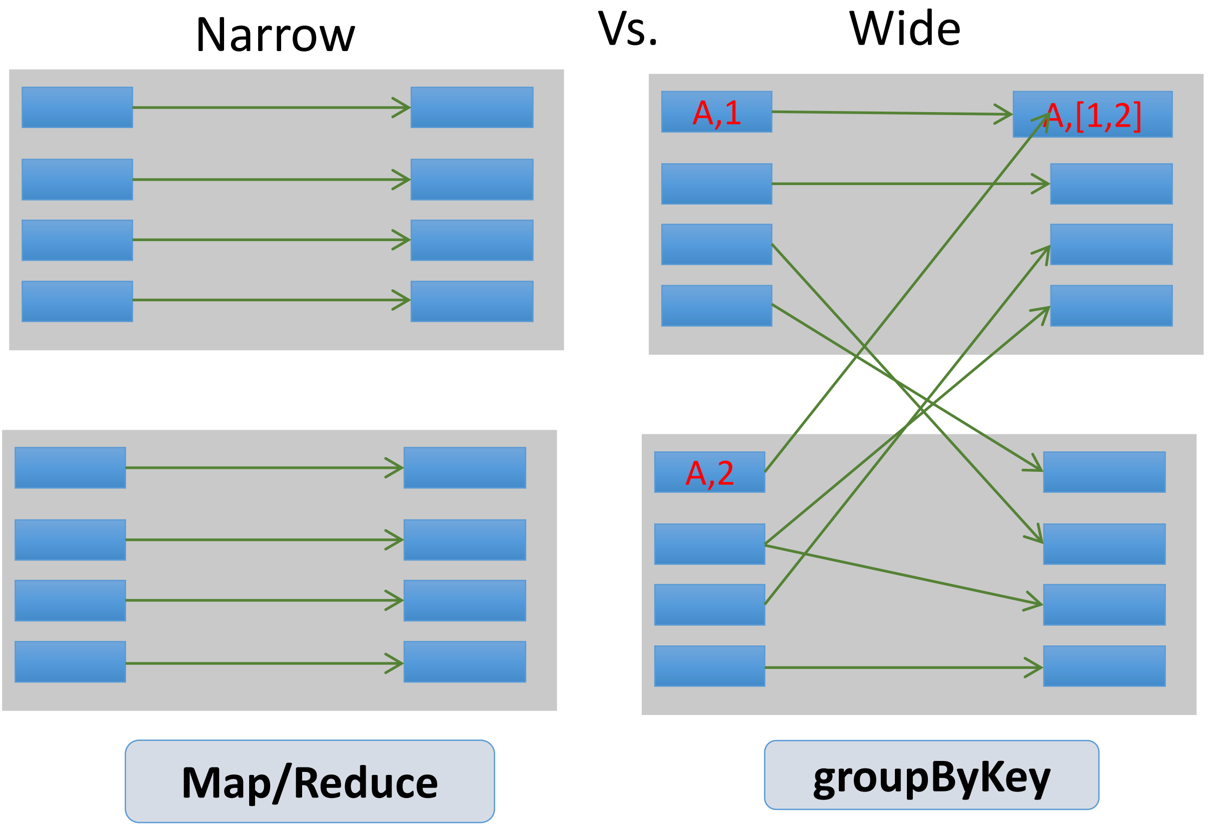
```
// Create a new DataFrame that contains "students"
students = users.filter(users.age < 21)

//Alternatively, using Pandas-like syntax
students = users[users.age < 21]

//Count the number of students users by gender
students.groupBy("gender").count()

// Join young students with another DataFrame called logs
students.join(logs, logs.userId == users.userId,
"left_outer")
```

Spark Operations

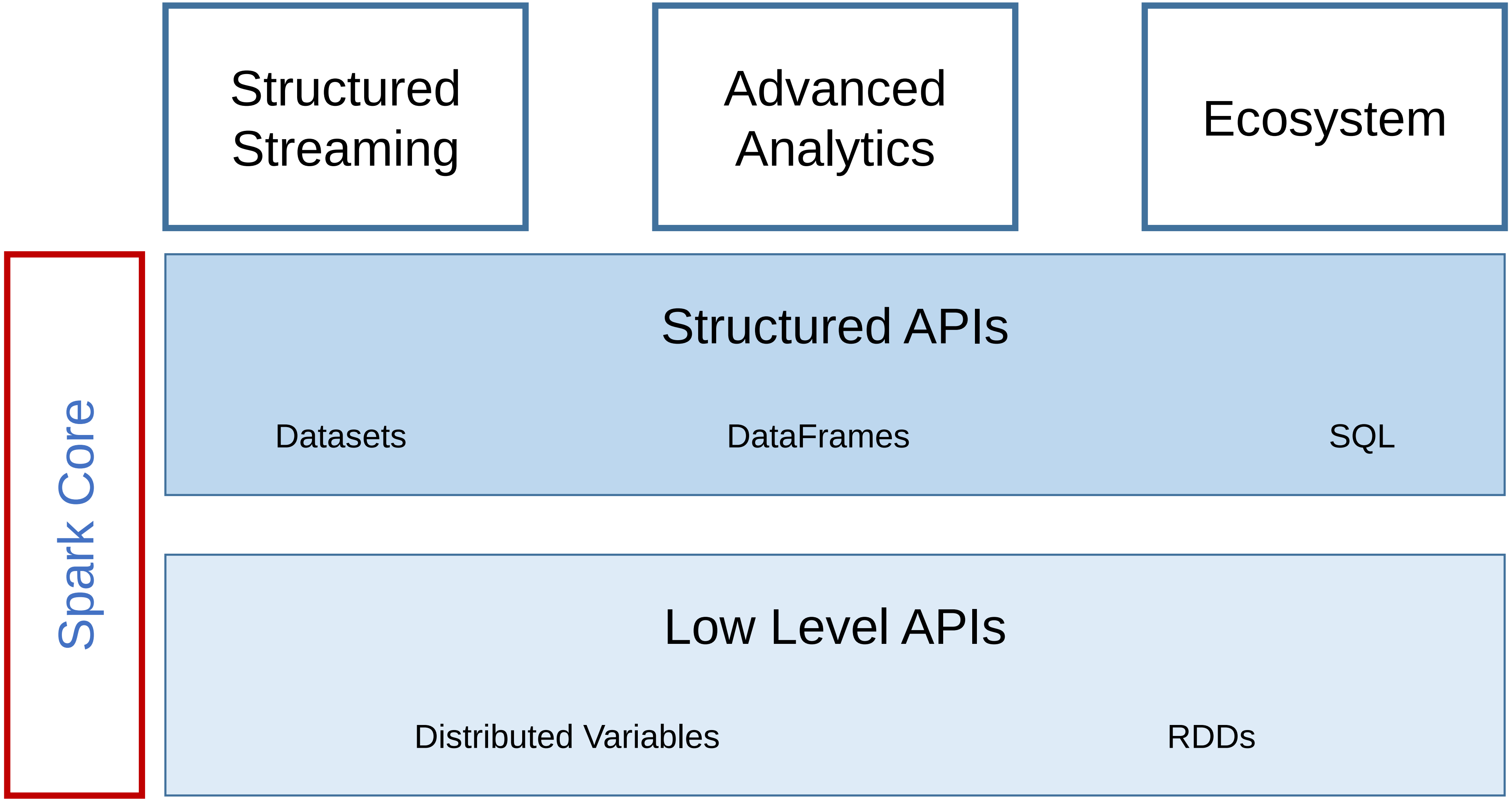
Transformations (create a new RDD)	map filter sample groupByKey reduceByKey sortByKey intersection	flatMap union join cogroup cross mapValues reduceByKey
Actions (return results to driver program)	collect Reduce Count takeSample take lookupKey	first take takeOrdered countByKey save foreach

Narrow vs. Wide transformations

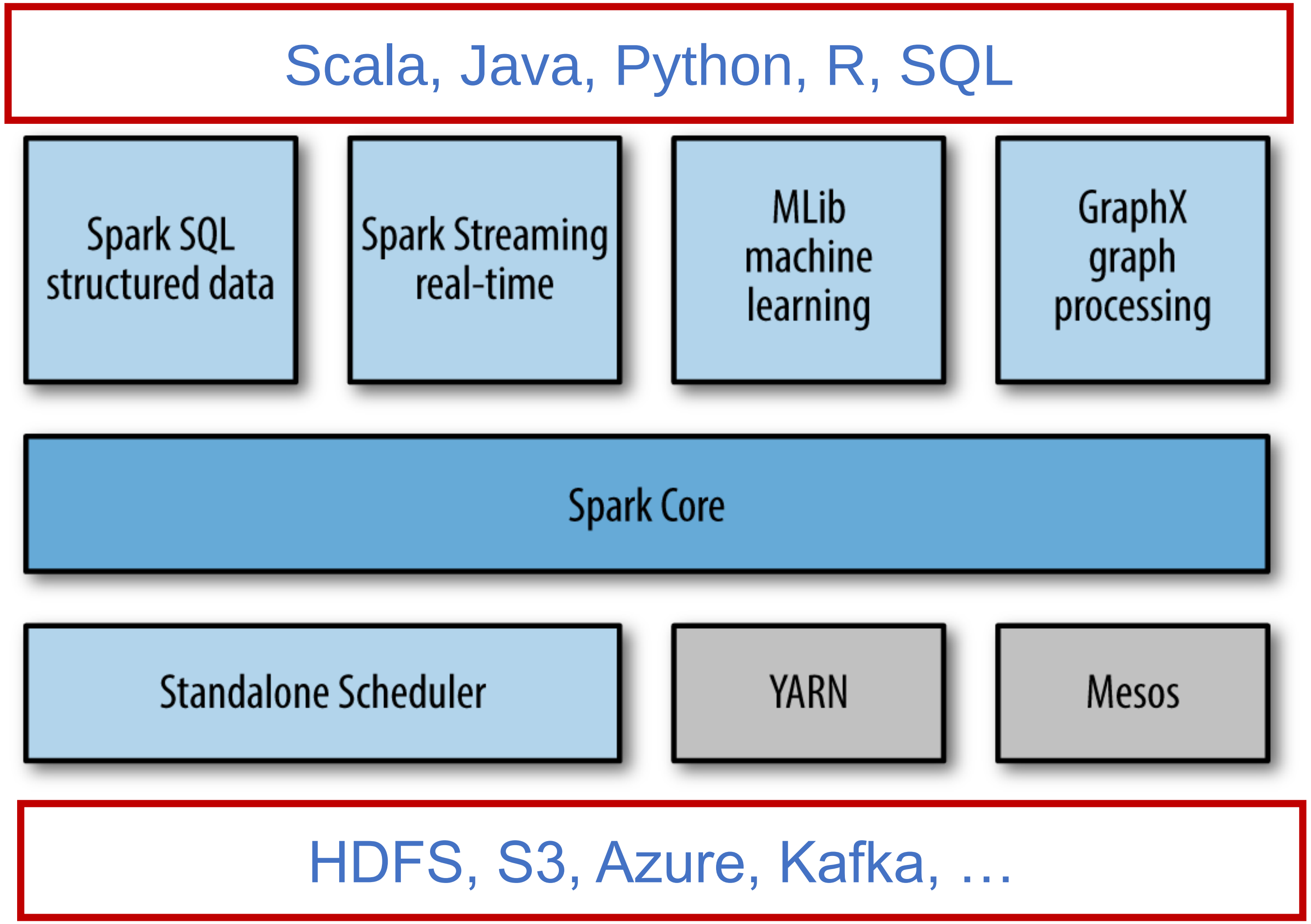
- Narrow transformations do not require shuffling of data across a partition – group into single stage
- Wide transformations cause data shuffles – results in stage boundaries.
- Each RDD maintains a pointer to one or more parents along with metadata about what type of relationship it has with the parent.
- if we call val b=a.map() on an RDD, the RDD b keeps a reference to its parent RDD a, that's an RDD lineage.



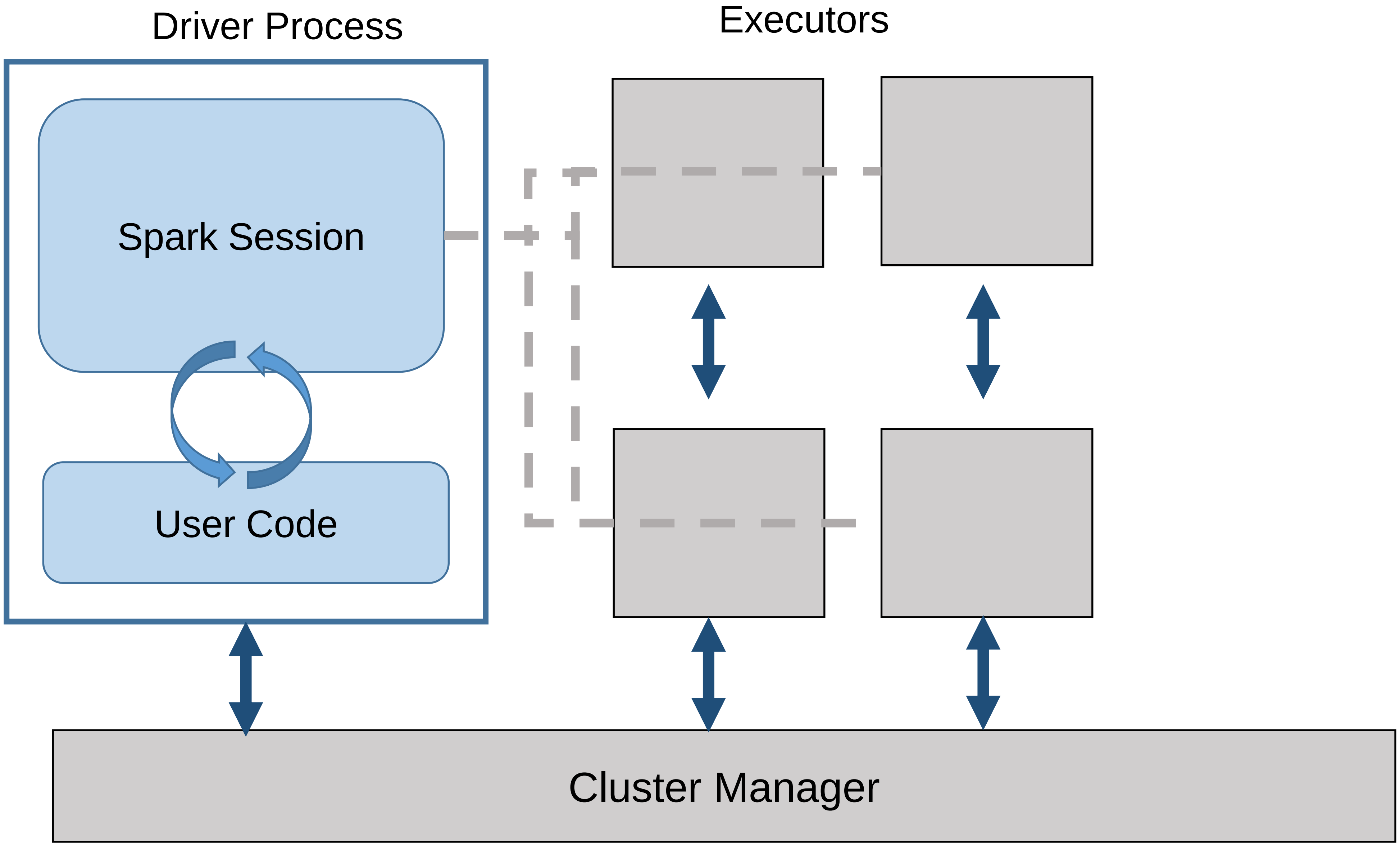
Spark High Level Structure



Spark Components



Spark Architecture



Recap of Spark Concepts …

- **Spark Session**
  Driver process
  Executors – user-defined manipulations across the cluster

- **DataFrame Structured API**
  Table of data with rows and columns
  List of columns and their types is referred to as a schema
  Distributed across the cluster

- **Partitions**
  Data broken up into chunks – a collection of rows
  A partition is physically located on a cluster
  Amount of parallelism depends on the number of partitions
  Programmers specify high-level manipulations on DataFrames
  Performance is not impacted by staying at a high-level

Recap of Spark Concepts …

- Transformations

  DataFrames are immutable

  DataFrames are manipulated using transformations

  Business logic in Spark is expressed as a series of transformations

  Transformations are abstract, they are not acted upon until there is an Action

  Series of transformations can be followed by one action

  Transformations can be wide or narrow

  Lazy computation leads to an optimum execution plan across a series of transformation

  Execution plan performance is independent of programming language

- Actions

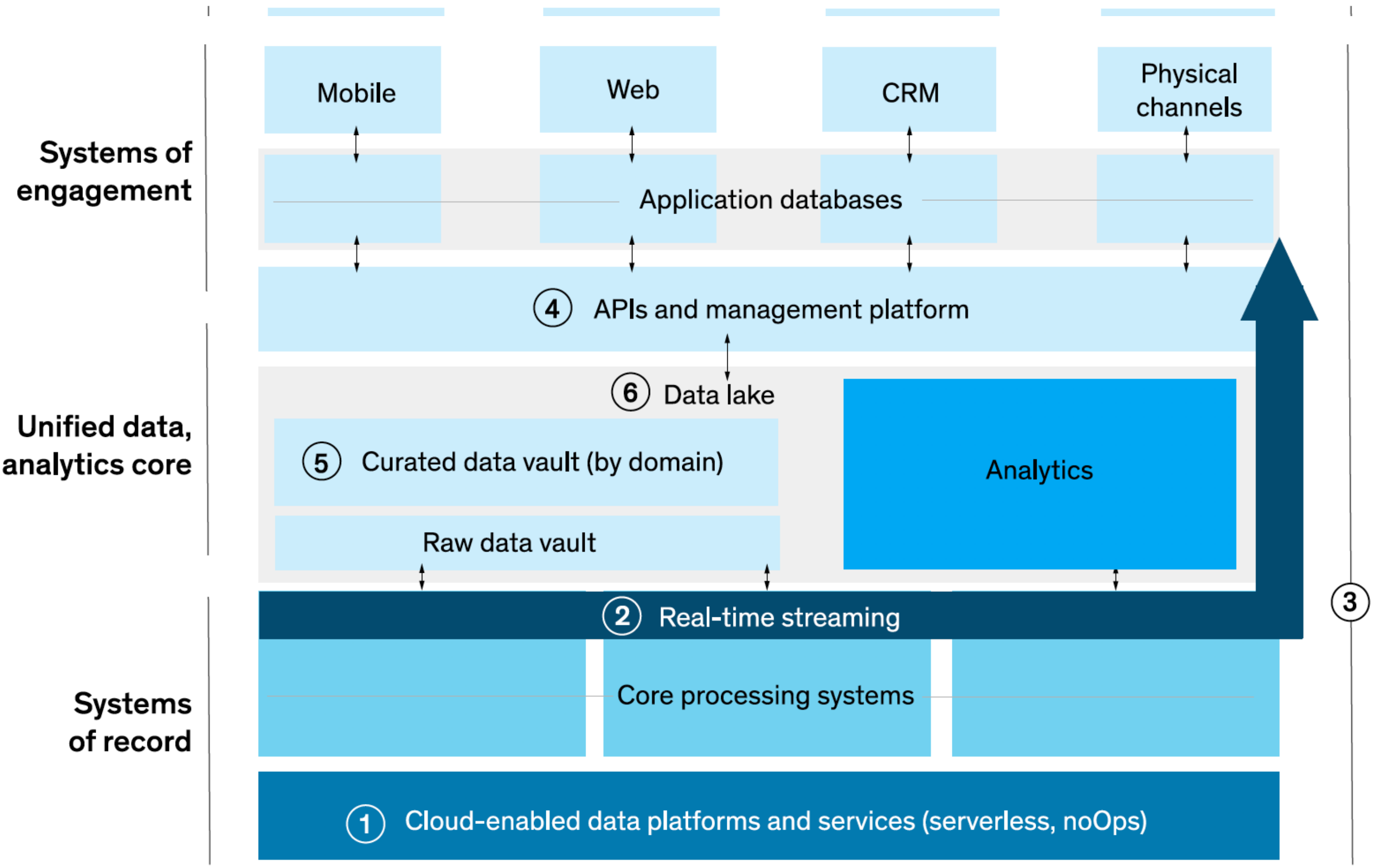
  Trigger the computation plan specified by transformations

  3 types of actions: View data on the console, Collect data to native objects, Write output

Spark Operations

Transformations (create a new RDD)	map filter sample groupByKey reduceByKey sortByKey intersection	flatMap union join cogroup cross mapValues reduceByKey
Actions (return results to driver program)	collect Reduce Count takeSample take lookupKey	first take takeOrdered countByKey save foreach

Spark is an embodiment of the architecture shift



1. In Spark, data and data analytics are both optimized for the Cloud
2. Spark is designed natively to handle streaming data
3. Spark supports multiple programming languages – enabling development of systems engagement
4. Spark is modularized with structured and low-level APIs, with extensibility that has led to the development of special purpose APIs.

Spark Function Reference

Basic RDD Functions

```
rdd.sortBy(function)
filter(function)
map(function)
flatMap(function)
mapToPair(function)
reduceByKey(function)
distinct()
union(otherRDD)
intersection(otherRDD)
subtract(otherRDD)
sample(withReplacement, fraction, [seed])
```

DataFrame Creation …

Read a CSV file with header line

```
sqlContext.read.format('com.databricks.spark.csv')
.options(header='true', inferSchema='true')
.load('filename')
```

DataFrame Creation …

- From a RDD containing lists

```
df = rdd.toDF([‘field1’, ‘field2’, … ])
```

- From a RDD of lists

```
df = sqlContext.createDataFrame(rdd, [‘field1’, ‘field2’, …
])
```

- From a RDD of Rows, named tuple, or dictionary

```
df = sqlContext.createDataFrame(rdd)
```

- From a created RDD of Rows

```
rowRDD = rdd.map(lambda x: Row(field1=x[0], field2=x[1],
...))
```

```
 df = sqlContext.createDataFrame(rowRDD)
```

Row

- Create a Row using named arguments – fields are sorted by name

```
row = Row(make = “Ford”, model=“F-150”)
```

- Fields in a Row can be accessed by:

`row[‘make’]` Or `row.make`

- Checking if a field is in a Row object

```
‘make’ in row
```

Row to Dictionary

- Convert a Row to a dictionary

```
row = Row(vin=ABC, desc=Row(make="Ford", model="F-150"))
```

- Convert top level – only converts the outer-most Row

```
row.asDict()
```

- Convert Row objects recursively

```
row.asDict(True)
```

SQL Query against a DataFrame

- Register a table name

```
sqlContext.registerDataFrameAsTable(df, “table1”)
```

Or

```
df.registerTempTable(“table1”)
```

- Query against a table

```
df2 = sqlContext.sql(“SELECT field1 as f1, field2 as f2 FROM table1”)
```

Resulting DataFrame (df2) has Rows with fields: f1, f2

Save a DataFrame to a CSV

- Output to a CSV file

```
(df.repartition(1)
.write
.format("com.databricks.spark.csv")
.options(header="true")
.save("/mnt/S3/output/dfOut")
```

PART 2

Parallel DB Architectures

PARALLEL DATABASE ARCHITECTURES

a) Shared memory

b) Shared disk

c) Shared nothing