وحید امیری

کارشناس ارشد فناوری اطلاعات - تجارت الکترونیک

کارشناس، مدرس و مشاور در حوزه کلان داده

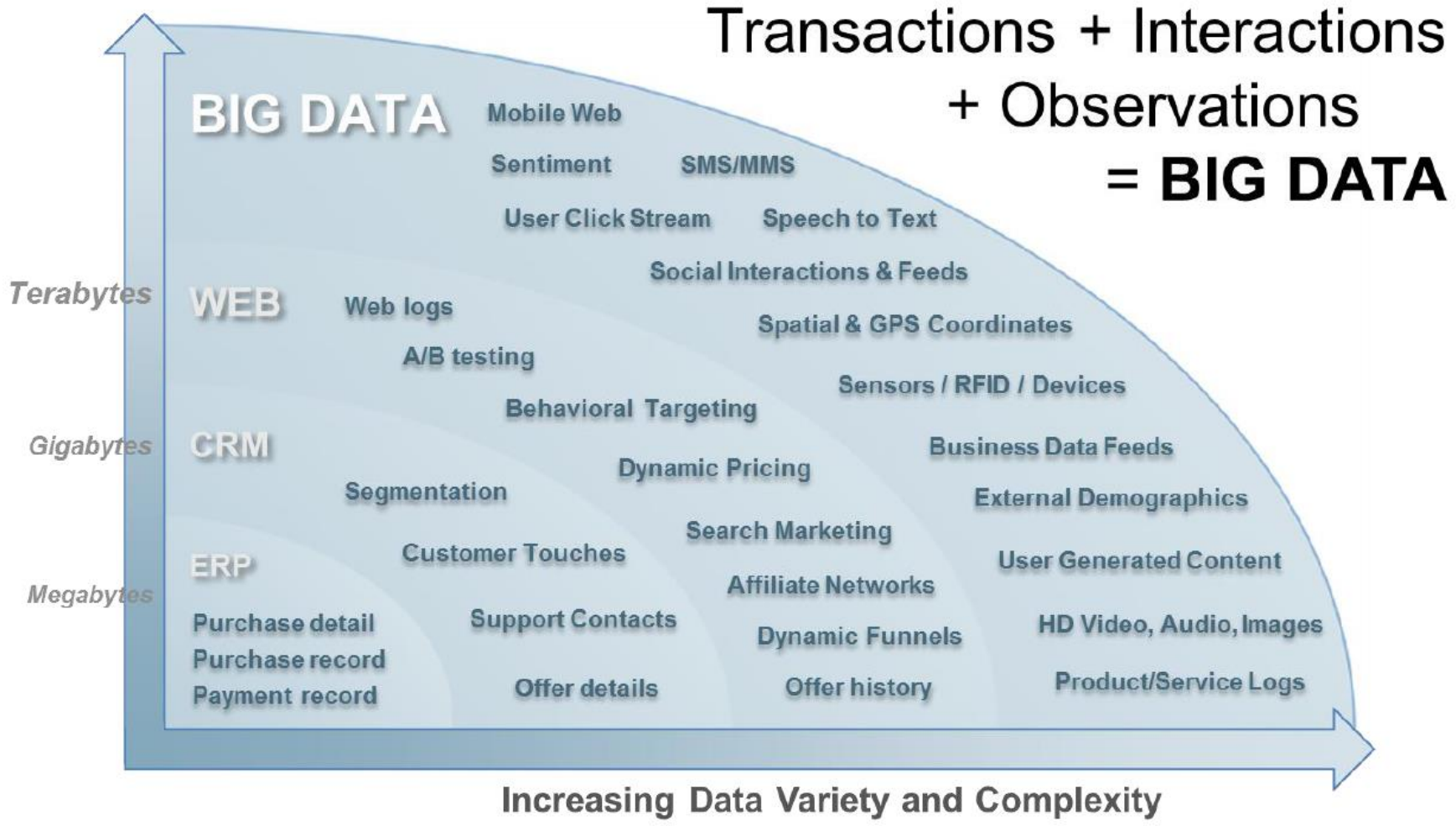
VahidAmiry.ir

@vahidamiry

Big Data Architecture

What is big data and why is it valuable to the business

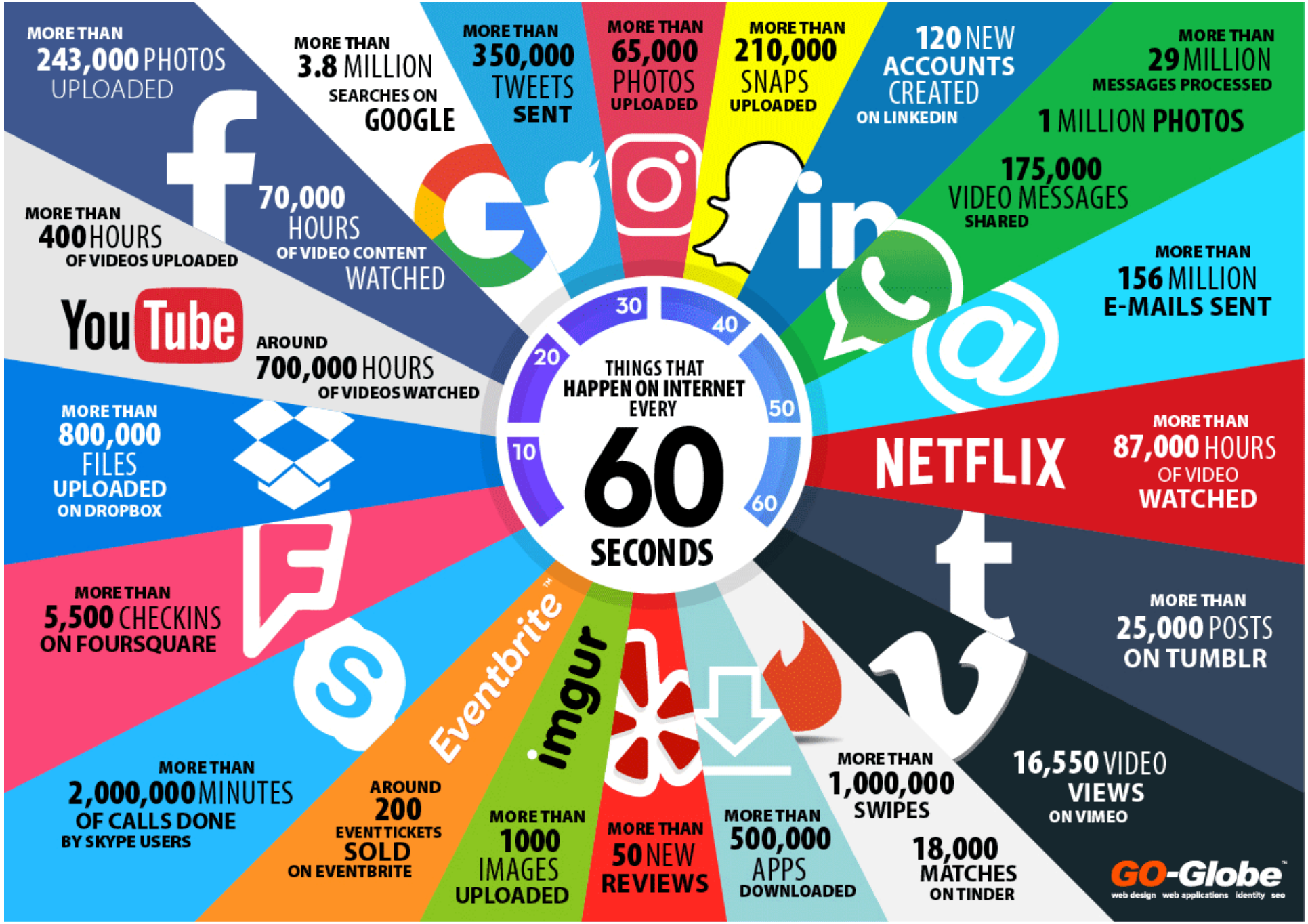
A evolution in the nature and use of data in the enterprise



Need to collect any data

Harness the growing and changing nature of data

- Challenge is combining transactional data stored in relational databases with less structured data
- Get the right information to the right people at the right time in the right format


MORE THAN 243,000 PHOTOS UPLOADED
MORE THAN 3.8 MILLION SEARCHES ON GOOGLE
MORE THAN 350,000 TWEETS SENT
MORE THAN 65,000 PHOTOS UPLOADED
MORE THAN 210,000 SNAPS UPLOADED
120 NEW ACCOUNTS CREATED ON LINKEDIN
MORE THAN 29 MILLION MESSAGES PROCESSED
1 MILLION PHOTOS
175,000 VIDEO MESSAGES SHARED
70,000 HOURS OF VIDEO CONTENT WATCHED
MORE THAN 400 HOURS OF VIDEOS UPLOADED
YouTube
AROUND 700,000 HOURS OF VIDEOS WATCHED
MORE THAN 156 MILLION E-MAILS SENT
THINGS THAT HAPPEN ON INTERNET EVERY 60 SECONDS
MORE THAN 800,000 FILES UPLOADED ON DROPBOX
NETFLIX
MORE THAN 87,000 HOURS OF VIDEO WATCHED
MORE THAN 5,500 CHECKINS ON FOURSQUARE
MORE THAN 25,000 POSTS ON TUMBLR
MORE THAN 2,000,000 MINUTES OF CALLS DONE BY SKYPE USERS
AROUND 200 EVENT TICKETS SOLD ON EVENTBRITE
Eventbrite
imgur
MORE THAN 1000 IMAGES UPLOADED
MORE THAN 50 NEW REVIEWS
MORE THAN 500,000 APPS DOWNLOADED
MORE THAN 1,000,000 SWIPES
18,000 MATCHES ON TINDER
16,550 VIDEO VIEWS ON VIMEO
GO-Globe
web design web applications identity seo

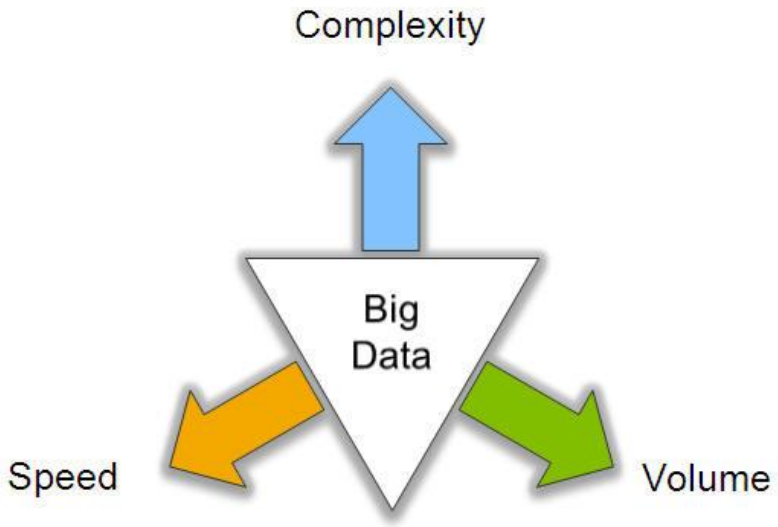
Big Data Definition

- No single standard definition…

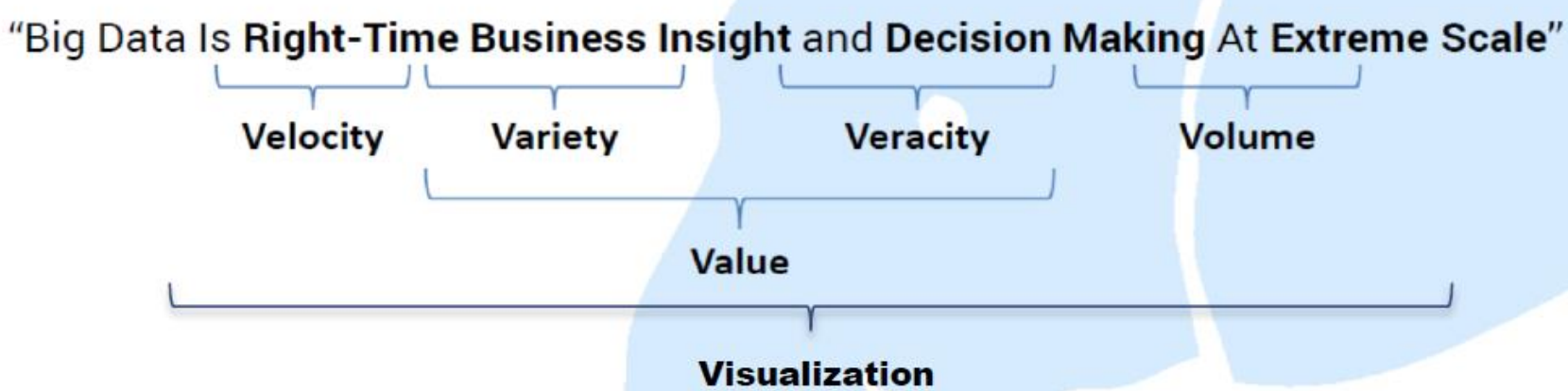
"***Big Data***" is data whose scale, diversity, and complexity require new architecture, techniques, algorithms, and analytics to manage it and extract value and hidden knowledge from it…

Big Data: 3V's


BIG
DATA?
VOLUME
Large amounts of data.
VELOCITY
Needs to be analyzed quickly.
VARIETY
Different types of structured
and unstructured data.

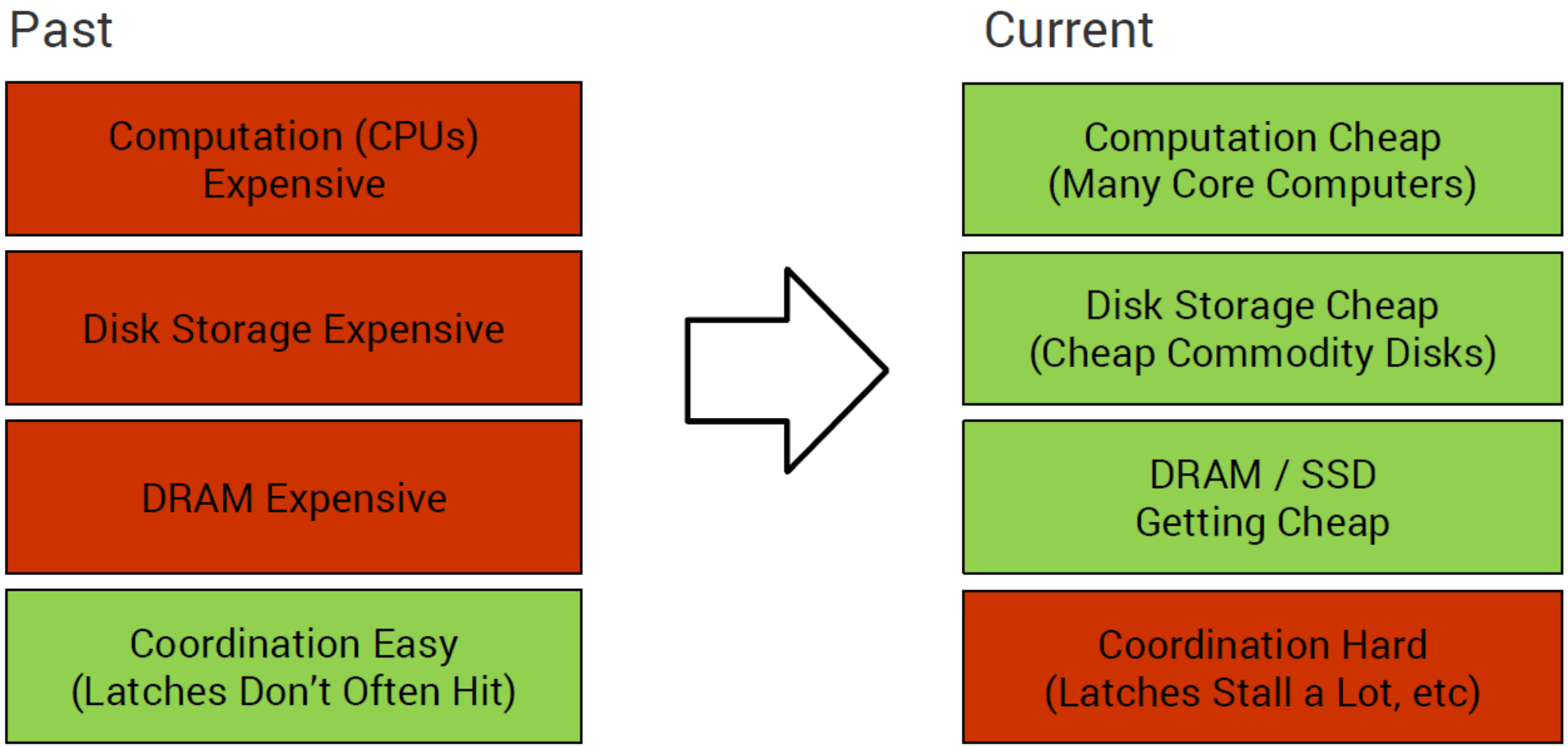

Complexity
Big
Data
Speed
Volume

THE 6 V'S OF BIG DATA: PUTTING IT TOGETHER




BIG DATA
with 8 V's
01
VOLUME
Can you find the information you are looking for?
02
VALUE
Can you find it when you most need it?
03
VERACITY
Are you dealing with information or disinformation?
04
VISUALISATION
Can you make sense at a glance? Does it trigger a decision?
05
VARIETY
Is a picture worth a thousand words in 70 languages? Is your information balanced?
06
VELOCITY
Information gains momentum and crises & opportunities evolve in real time. How is outlook for today?
07
VISCOSITY
Does it stick with you? Does it call for action?
08
VIRALITY
AHA to-go? Does it convey a message that can be pasted into a presentation or Instgrammed?

The Model Has Changed…

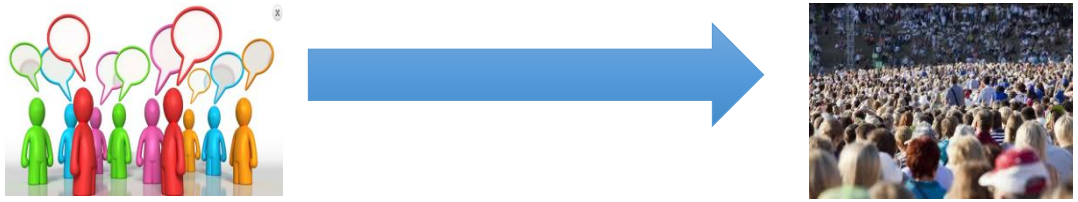


The Model Has Changed…

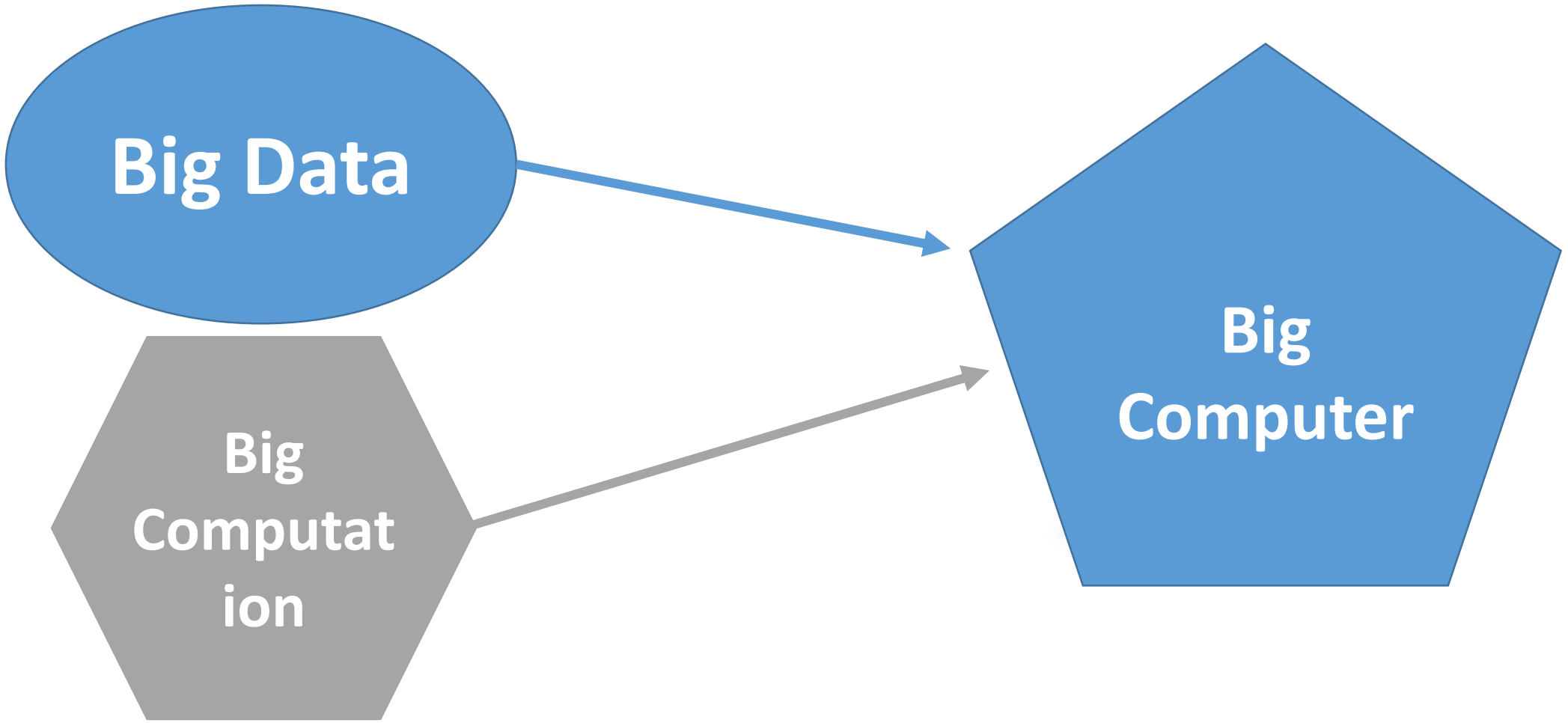
- **The Model of Generating/Consuming Data has Changed**

Old Model: Few companies are generating data, all others are consuming data

New Model: all of us are generating data, and all of us are consuming data

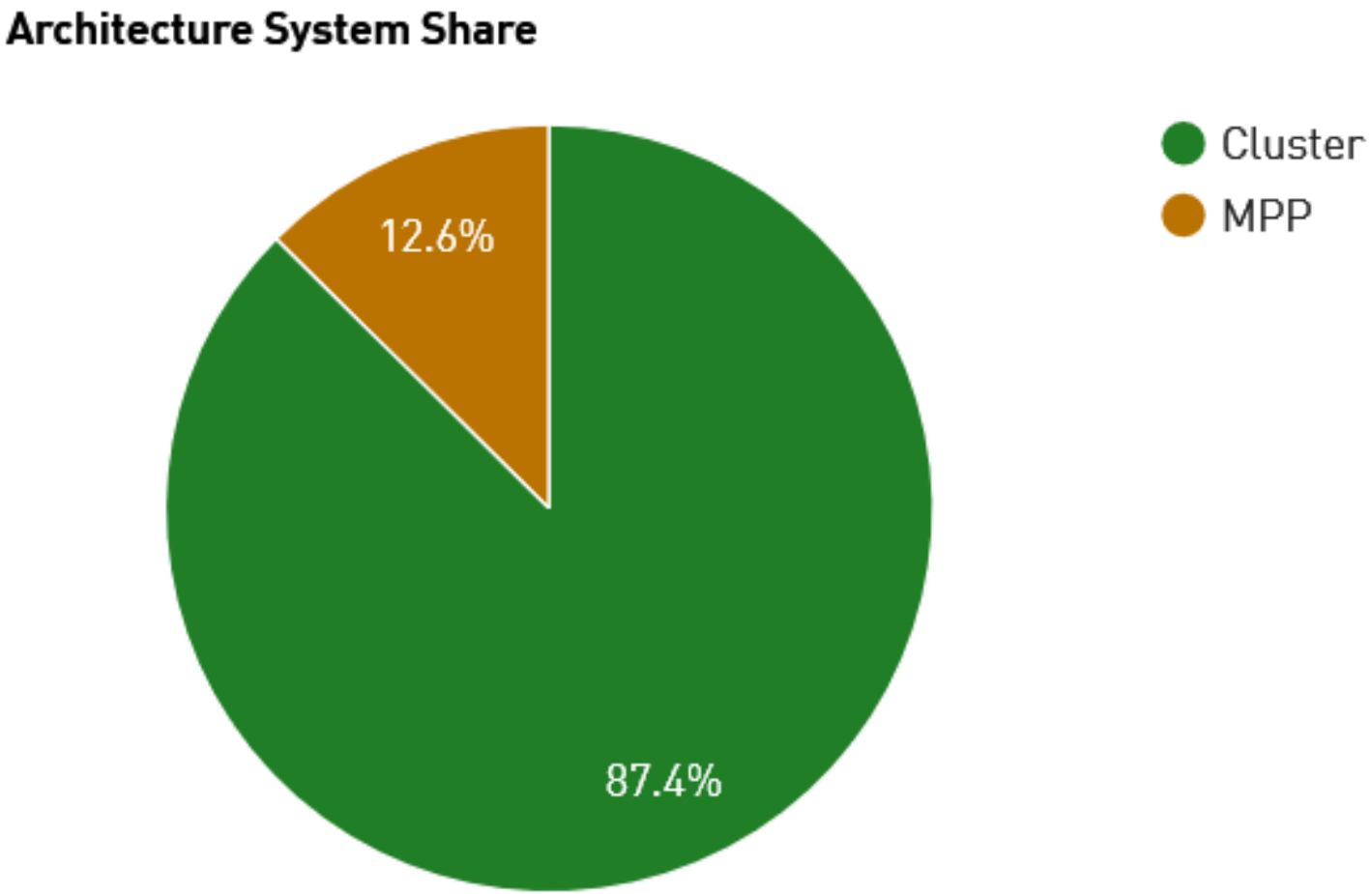


Solution

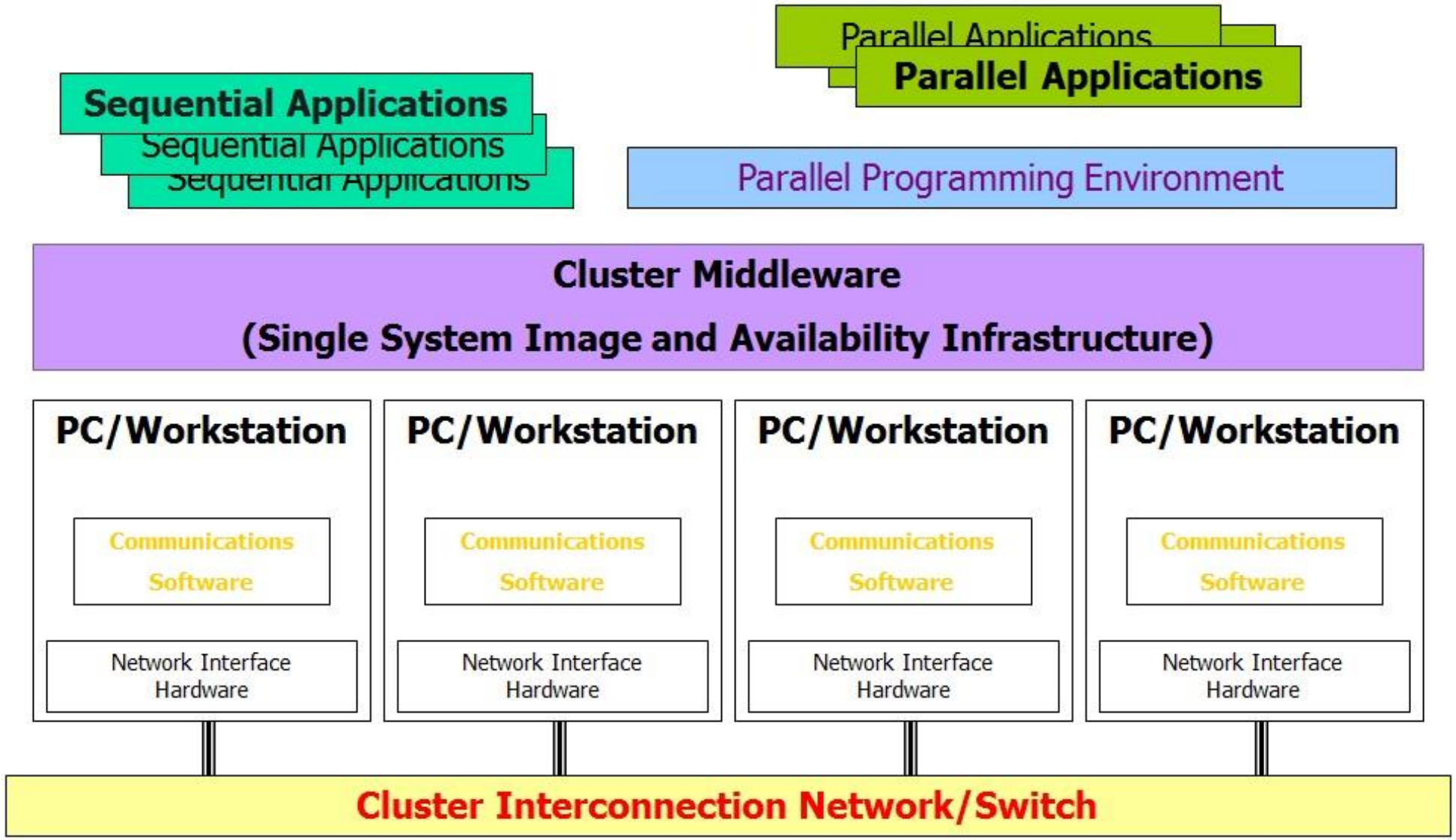



天河高效能计算机系统
天河高效能计算机系统
天河高效能计算机系统
天河
天河

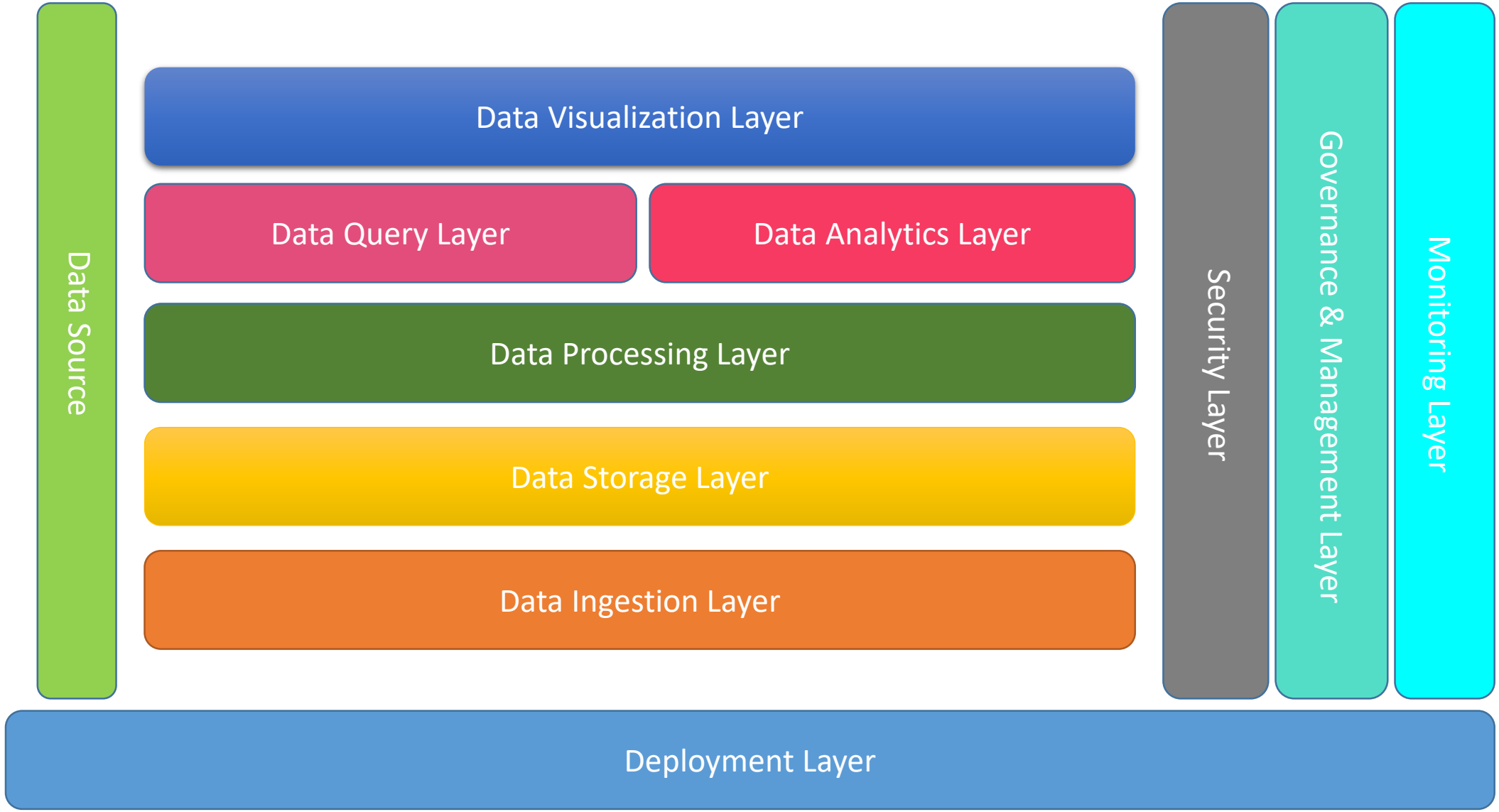
TOP500 SUPERCOMPUTER



Cluster Architecture



Big Data Layers



Data Ingestion Layer

Data Ingestion Layer

- Scalable, Extensible to capture streaming and batch data
- Provide capability to business logic, filters, validation, data quality, routing, etc. business requirements

Data Ingestion Layer

- Amazon Kinesis – real-time processing of streaming data at massive scale.
- Apache Chukwa – data collection system.
- Apache Flume – service to manage large amount of log data.
- Apache Kafka – distributed publish-subscribe messaging system.
- Apache Sqoop – tool to transfer data between Hadoop and a structured datastore.
- Cloudera Morphlines – framework that help ETL to Solr, Hbase and HDFS.
- Facebook Scribe – streamed log data aggregator.
- Fluentd – tool to collect events and logs.
- Google Photon – geographically distributed system for joining multiple continuously flowing streams of data in real-time with high scalability and low latency.

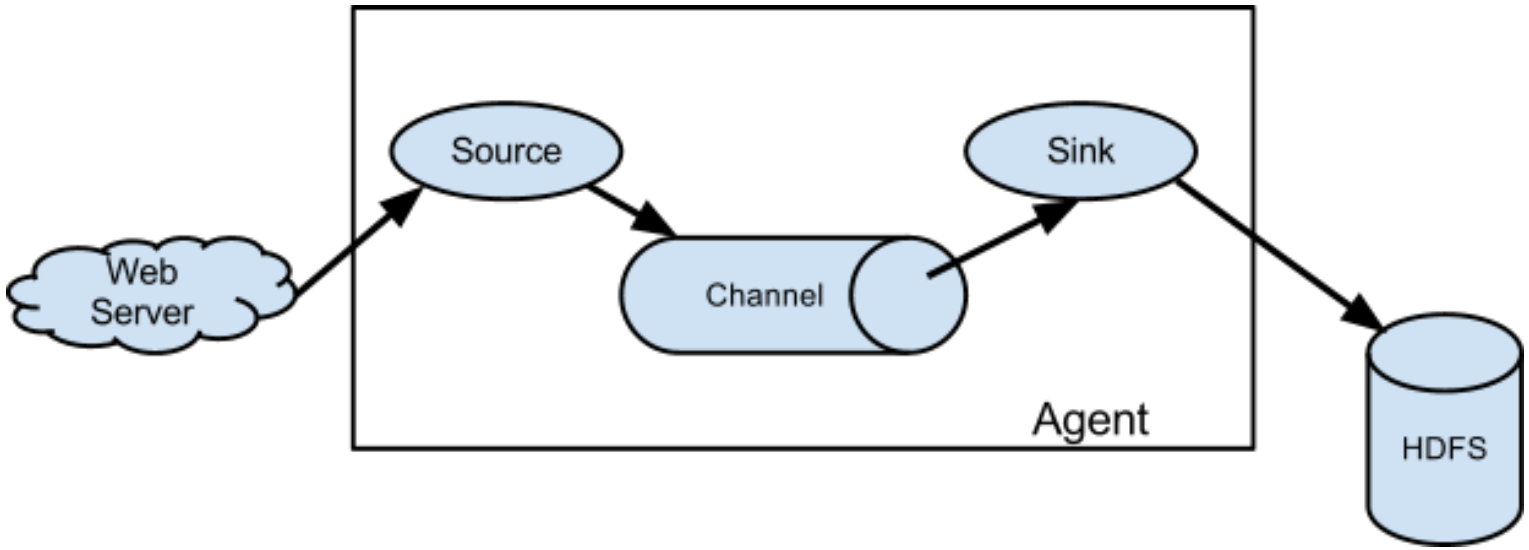
Data Ingestion Layer

- Heka – open source stream processing software system.
- HIHO – framework for connecting disparate data sources with Hadoop.
- Kestrel – distributed message queue system.
- LinkedIn Data bus – stream of change capture events for a database.
- LinkedIn Kamikaze – utility package for compressing sorted integer arrays.
- LinkedIn White Elephant – log aggregator and dashboard.
- Logstash – a tool for managing events and logs.
- Netflix Suro – log aggregator like Storm and Samza based on Chukwa.
- Pinterest Secor – is a service implementing Kafka log persistence.
- Linkedin Gobblin – linkedin's universal data ingestion framework.

Apache Flume

- Apache Flume is a distributed and reliable service for efficiently collecting, aggregating, and moving large amounts of streaming data into HDFS (especially “logs”).
  - Data is pushed to the destination (Push Mode).
  - Flume does not replicate events - in case of flume-agent failure, you will lose events in the channel

Sources	Channels	Sinks
Avro	Memory	HDFS
Thrift	JDBC	Logger
Exec	File	Avro
JMS		Thrift
NetCat		IRC
Syslog TCP/UDP		File Roll
HTTP		Null
		HBase
Custom		**Custom**



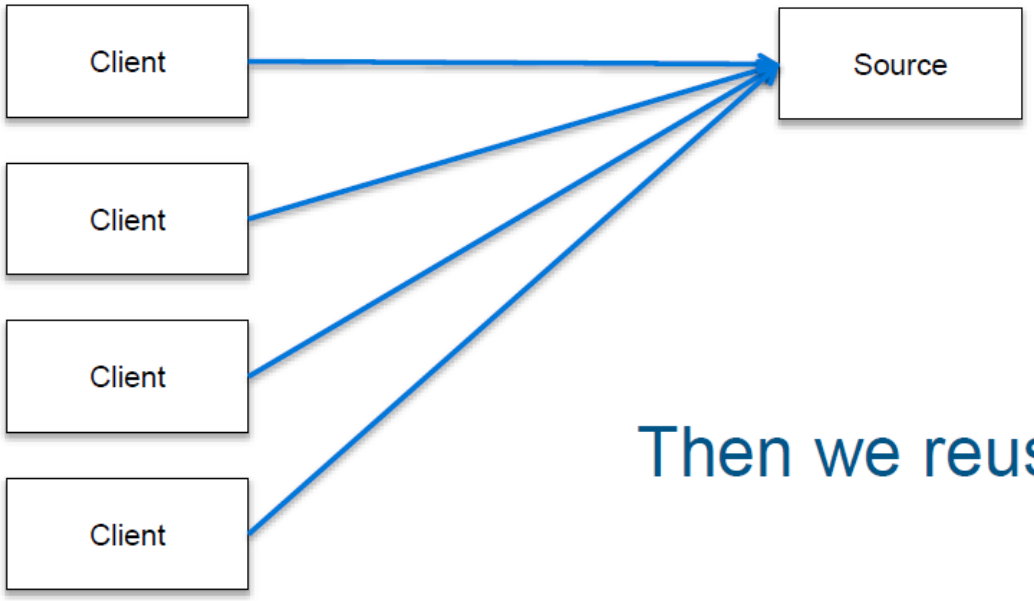
Apache Kafka

- A messaging system is a system that is used for transferring data from one application to another
  - … so applications can focus on data
  - … and not how to share it

Why Kafka

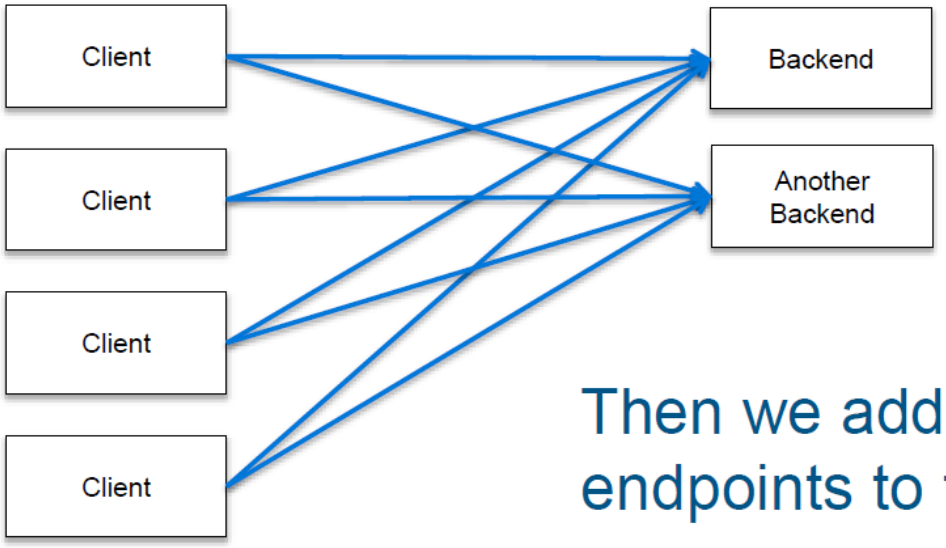
Data Pipelines Start like this.

Why Kafka



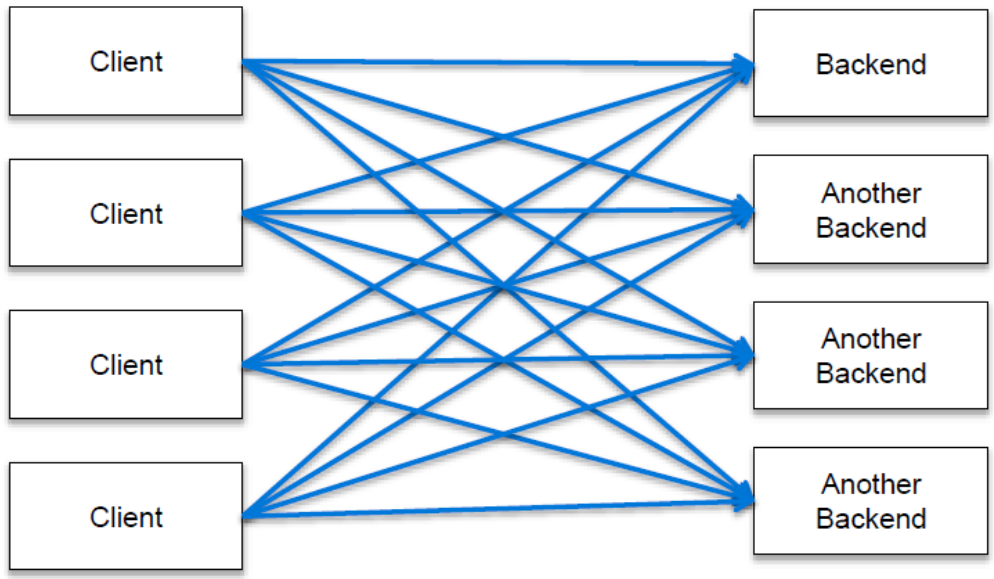
Then we reuse them

Why Kafka



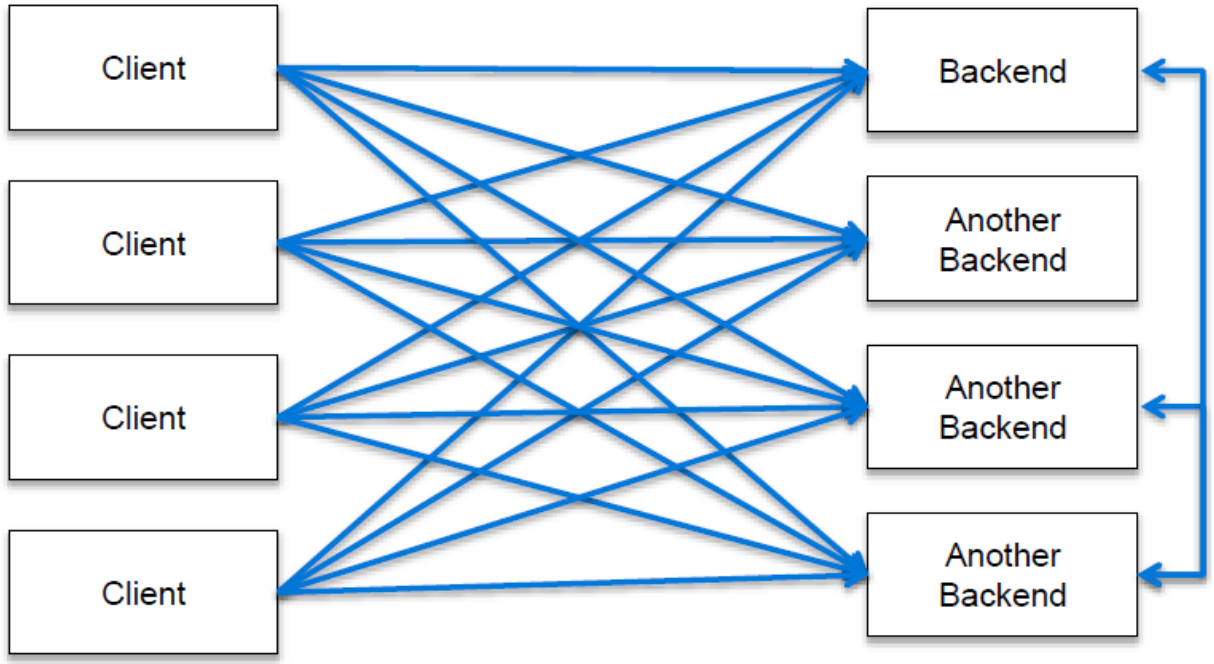
Then we add additional endpoints to the existing sources

Why Kafka



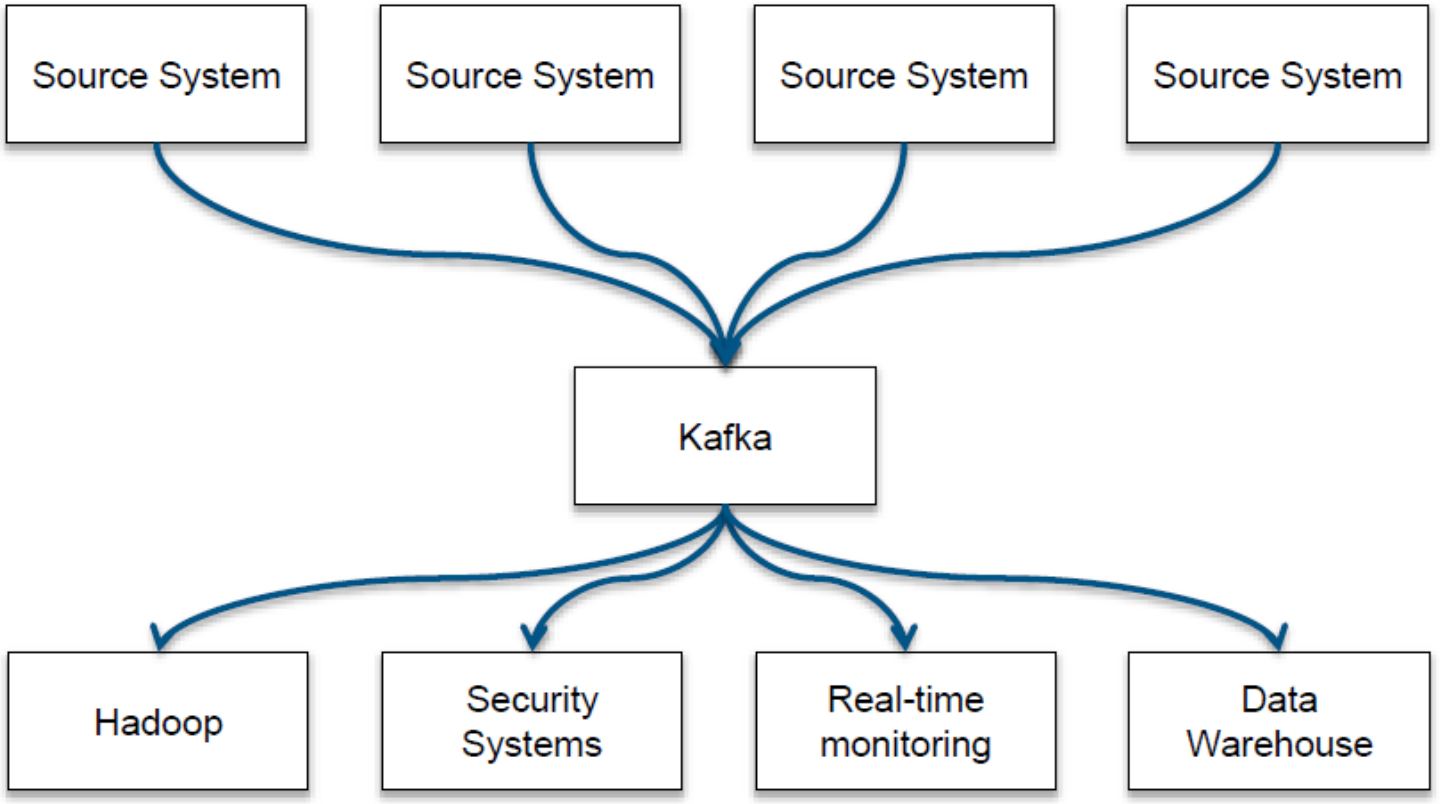
Then it starts to look like this

Why Kafka



With maybe some of this

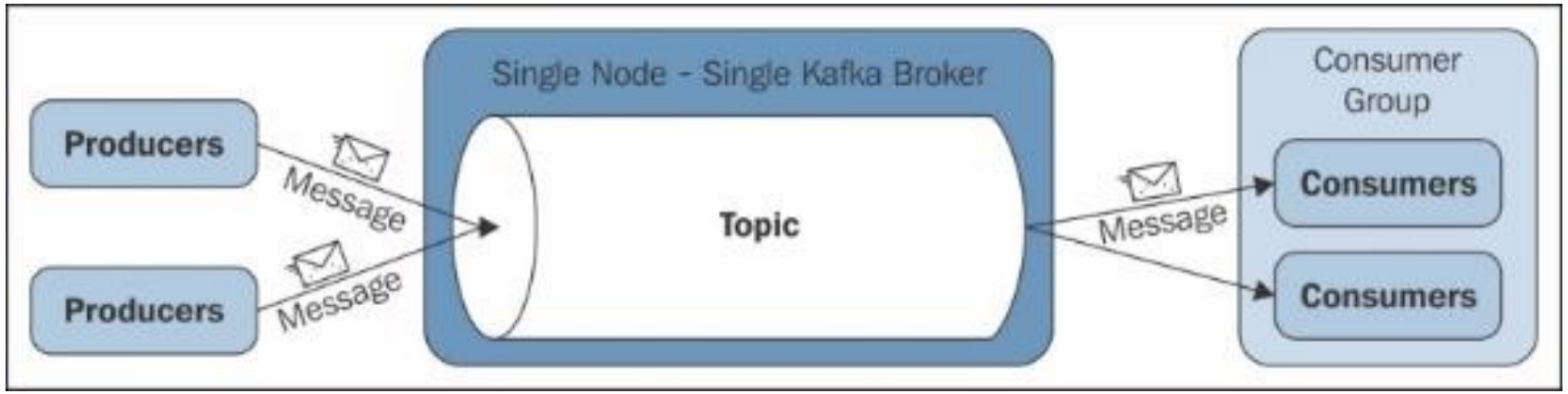
Why Kafka



Kafka decouples Data Pipelines

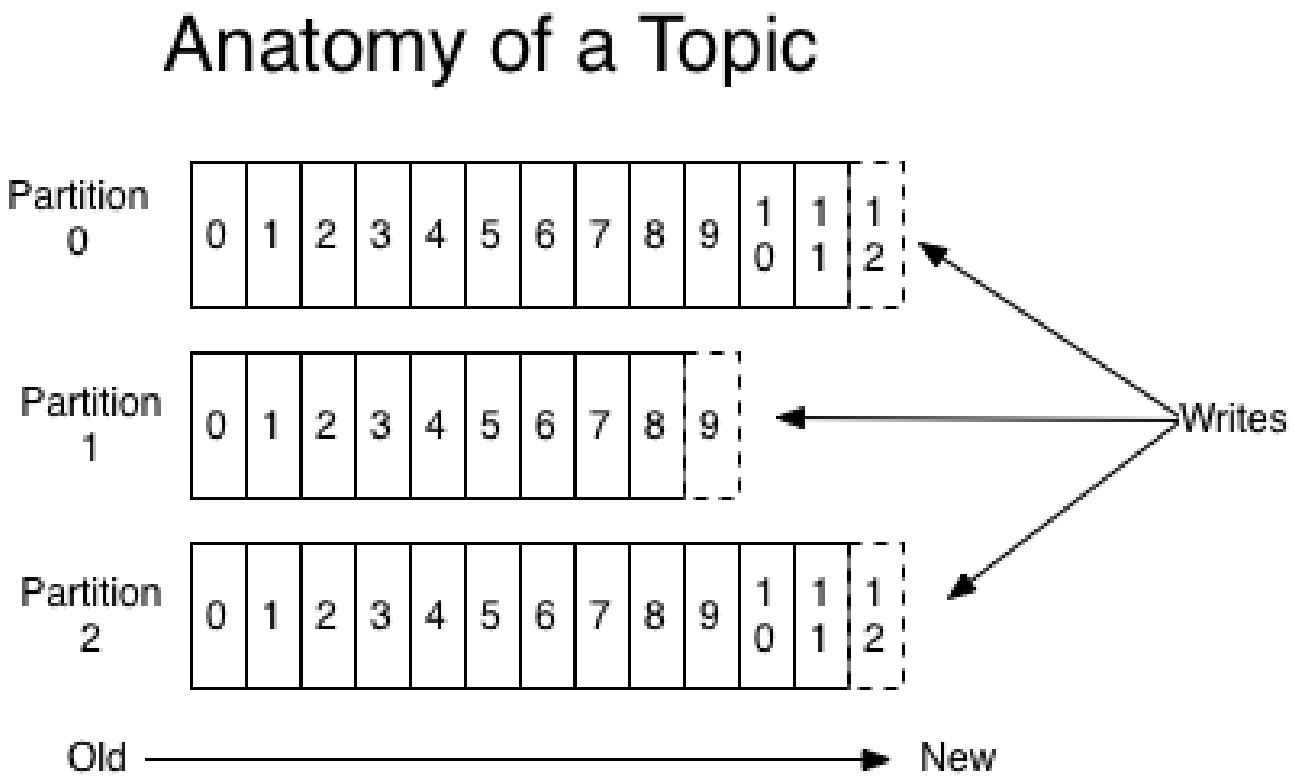
Kafka – Topics

- Kafka clusters store a stream of records in categories called **topics**
  - A topic is a feed name or category to which records are published (a named stream of records)
  - Topics are always multi-subscriber (0 or many consumers)
  - For each topic the Kafka cluster maintains a log



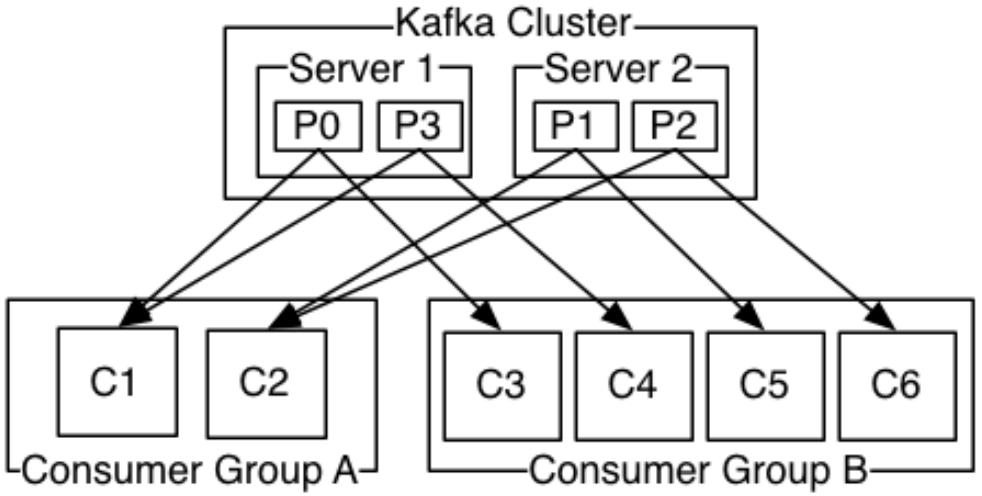
Kafka Basics – Partitions

- A topic consists of **partitions.**
- Partition: **ordered + immutable** sequence of messages that is continually appended to



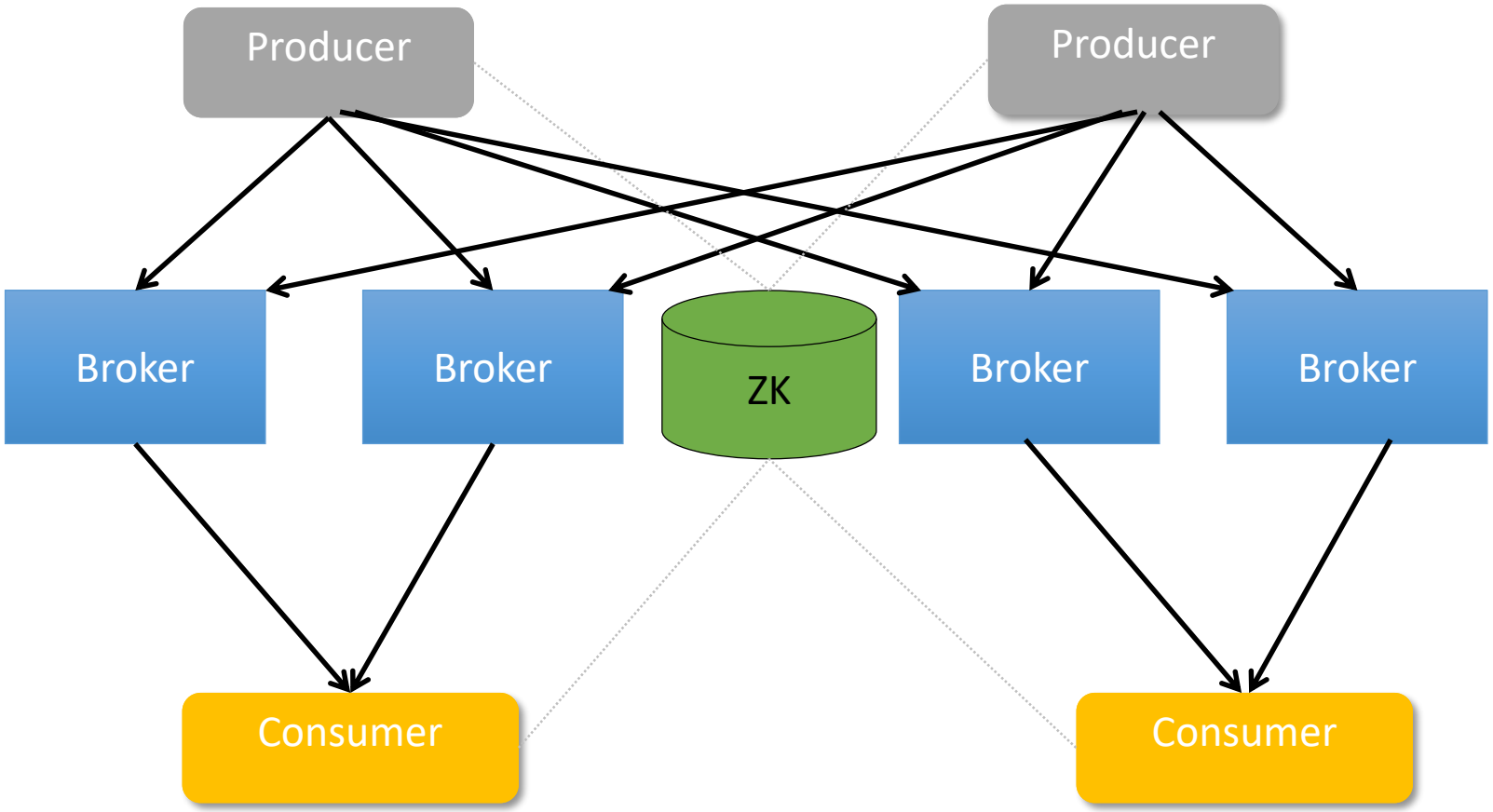
Kafka Basics – Partitions

- #partitions of a topic is configurable
- #partitions determines **max** consumer (group) parallelism



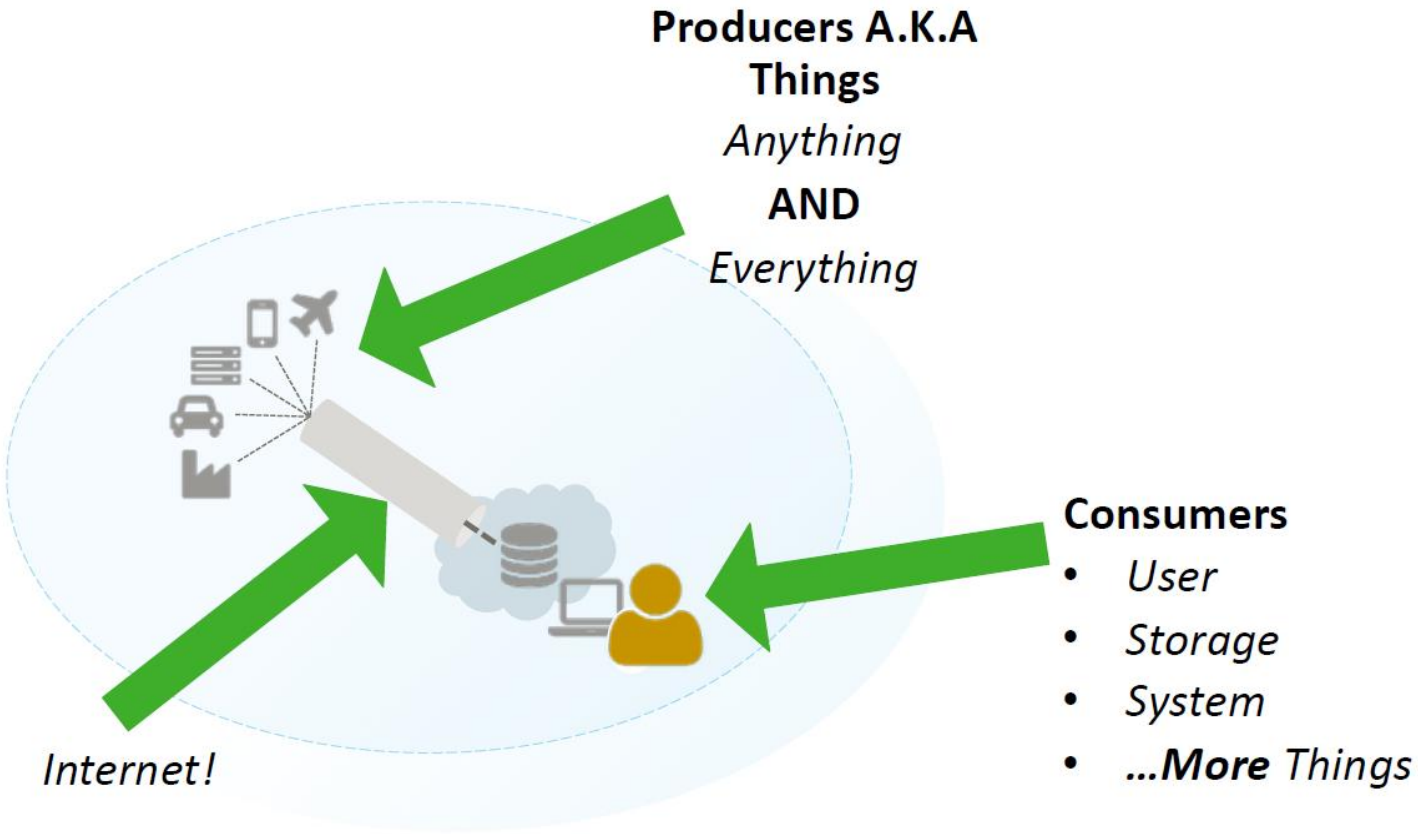
- Consumer group A, with 2 consumers, reads from a 4-partition topic
- Consumer group B, with 4 consumers, reads from the same topic

Kafka Architecture



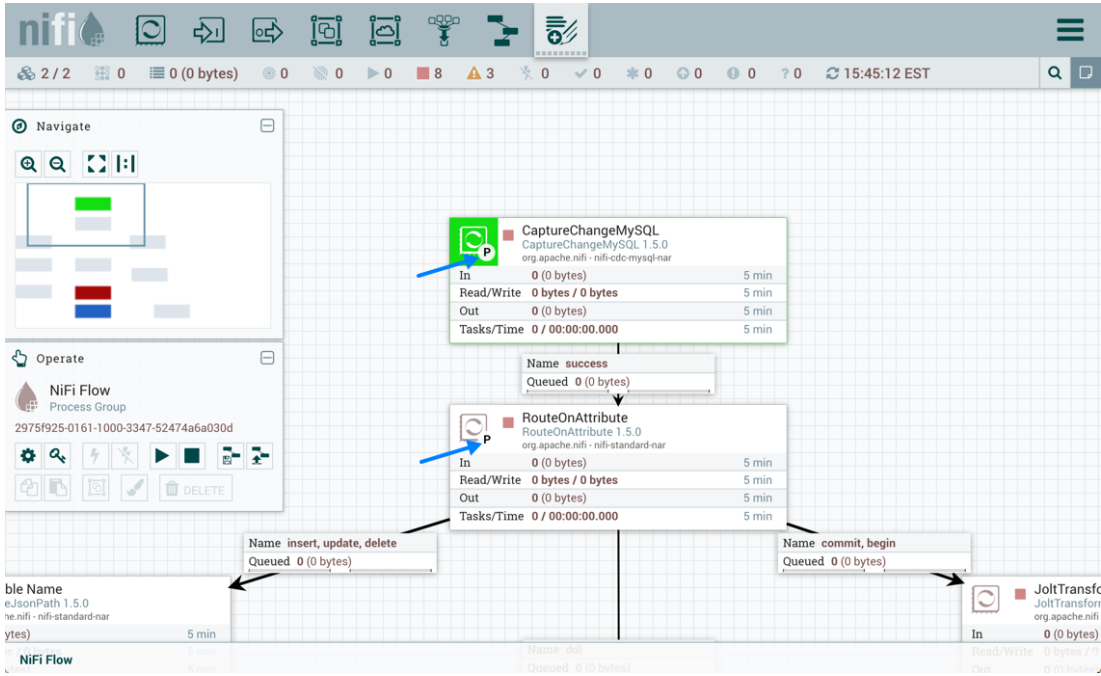
Apache NiFi

- Moving some content from A to B
  - **Content** could be any bytes
    - Logs
    - HTTP
    - XML
    - CSV
    - Images
    - Video
    - Telemetry
  - **Consideration**
    - Standards
    - Formats
    - Protocols
    - Veracity of Information
    - Validity of Information
    - Schemas

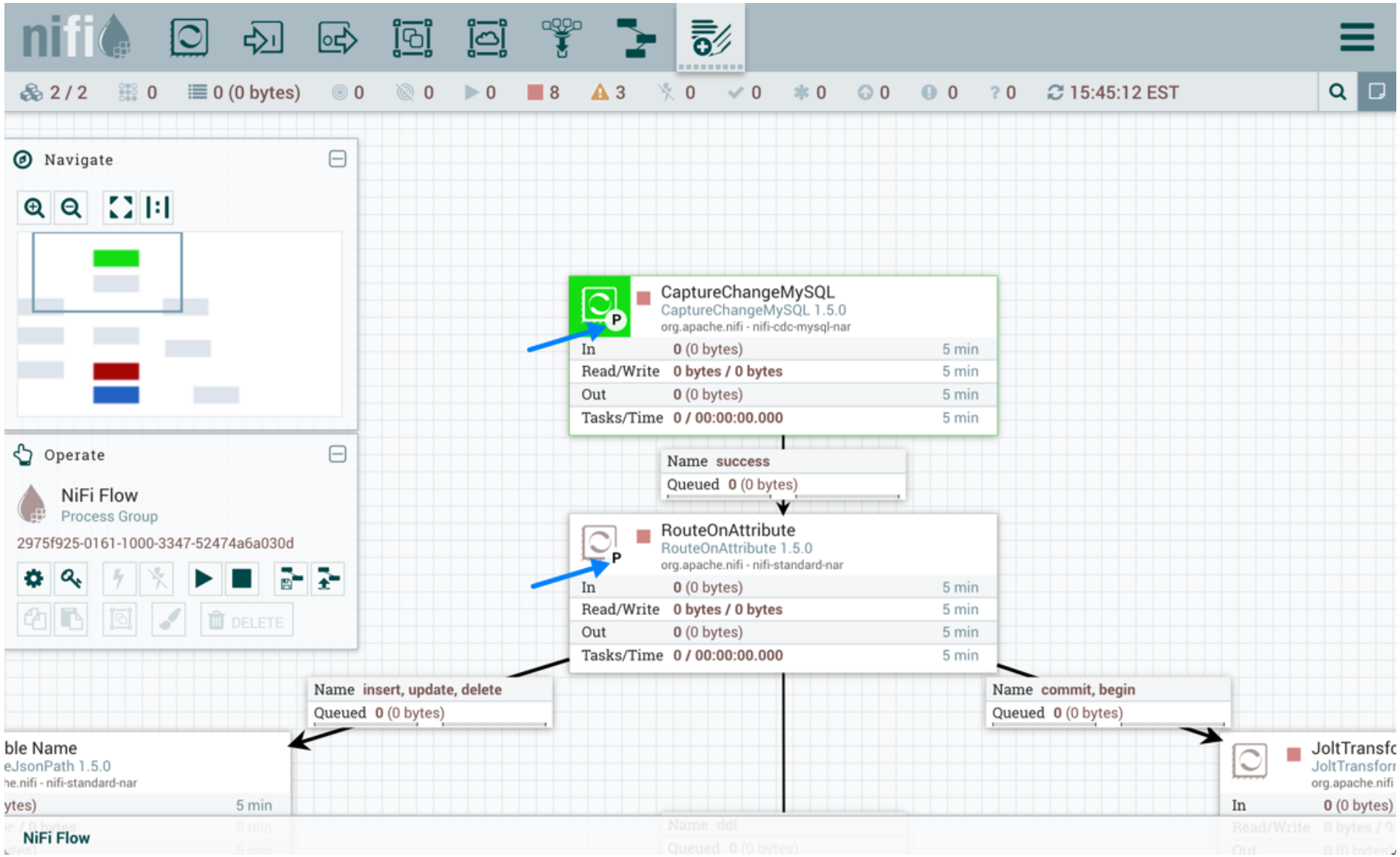


Apache NiFi

- Powerful and reliable system to process and distribute data
- Directed graphs of data routing and transformation
- Web-based User Interface for creating, monitoring, & controlling data flows
- Highly configurable - modify data flow at runtime, dynamically prioritize data
- Easily extensible through development of custom components

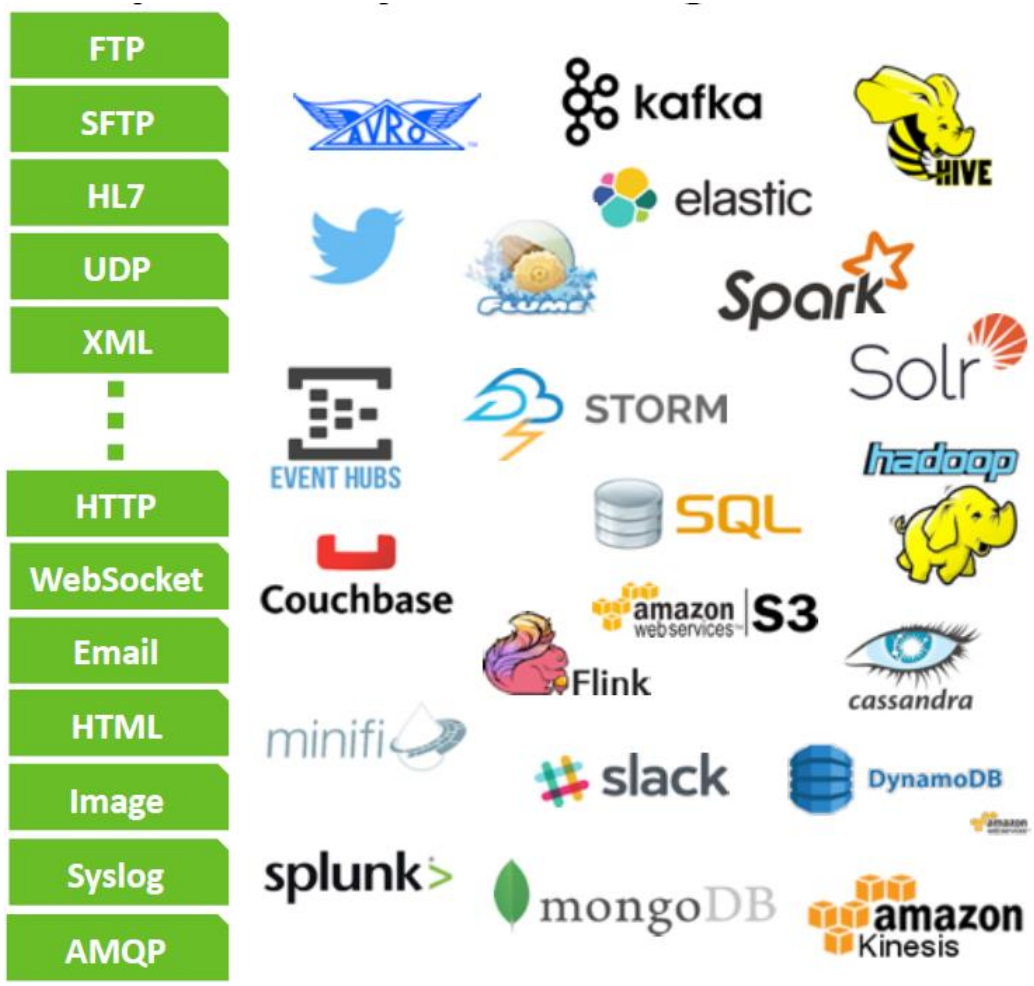


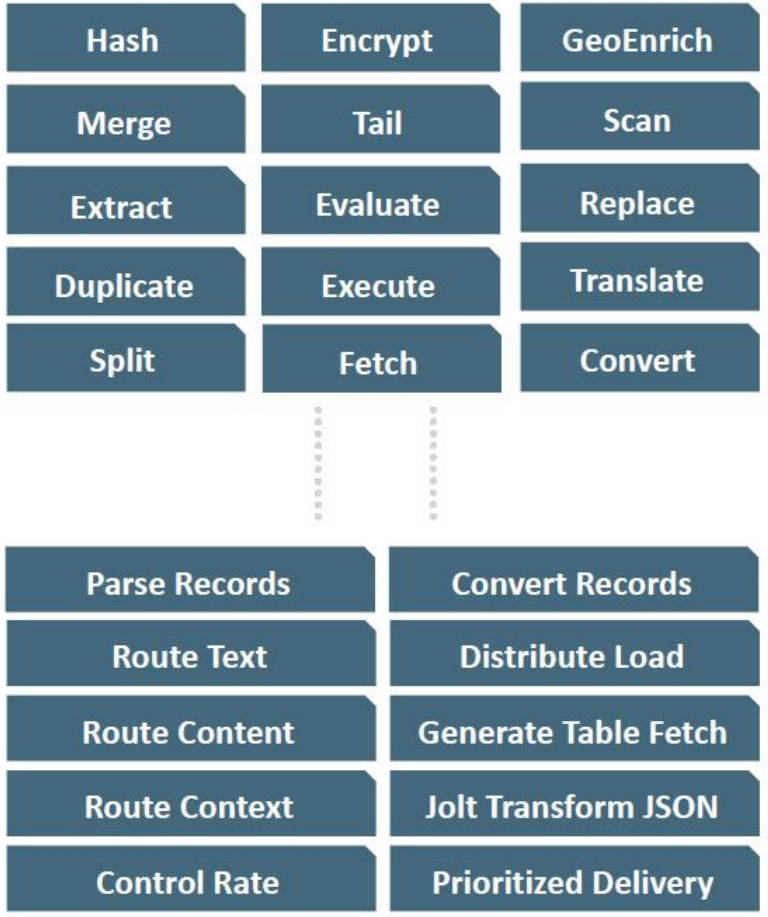
NiFi User Interface



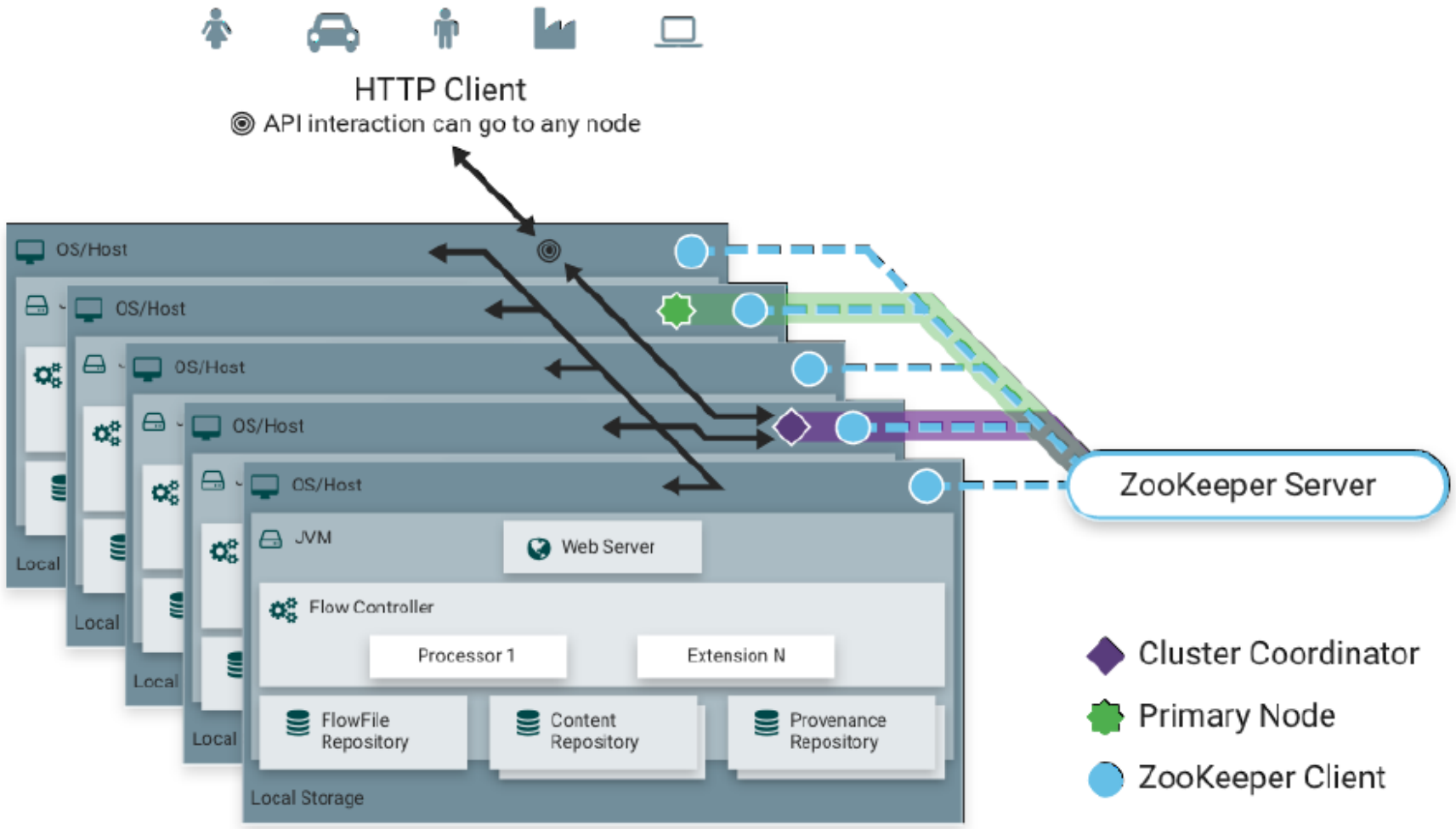
NiFi Processors and Controllers

- 200+ built in Processors
- 10+ built Control Services
- 10+ built in Reporting Tasks



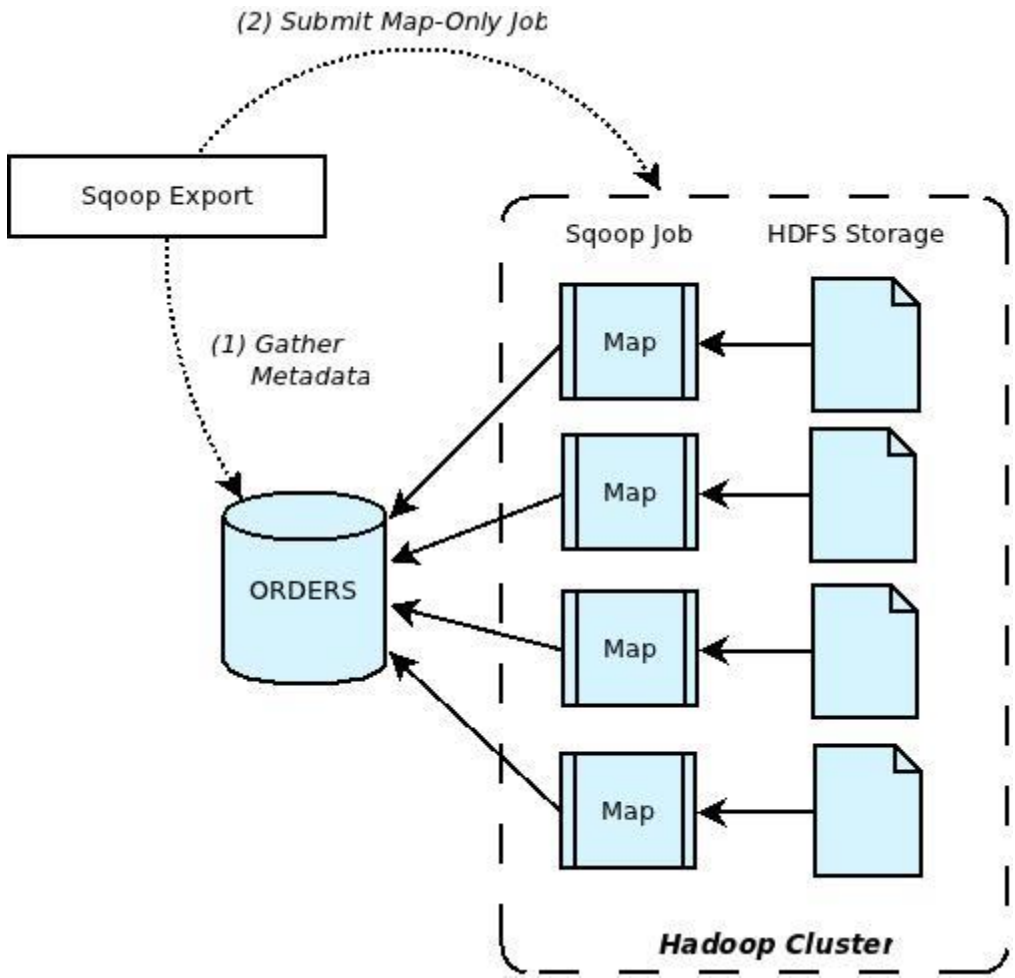


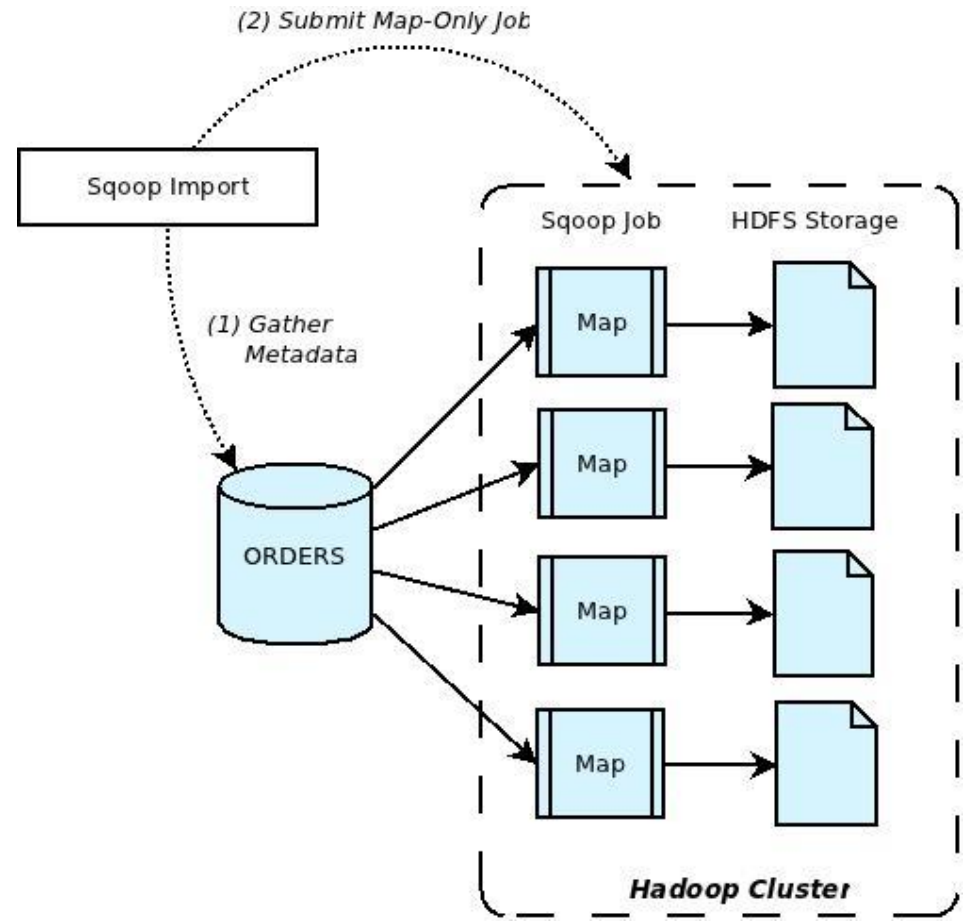
NiFi Architecture



Cluster

Apache Sqoop



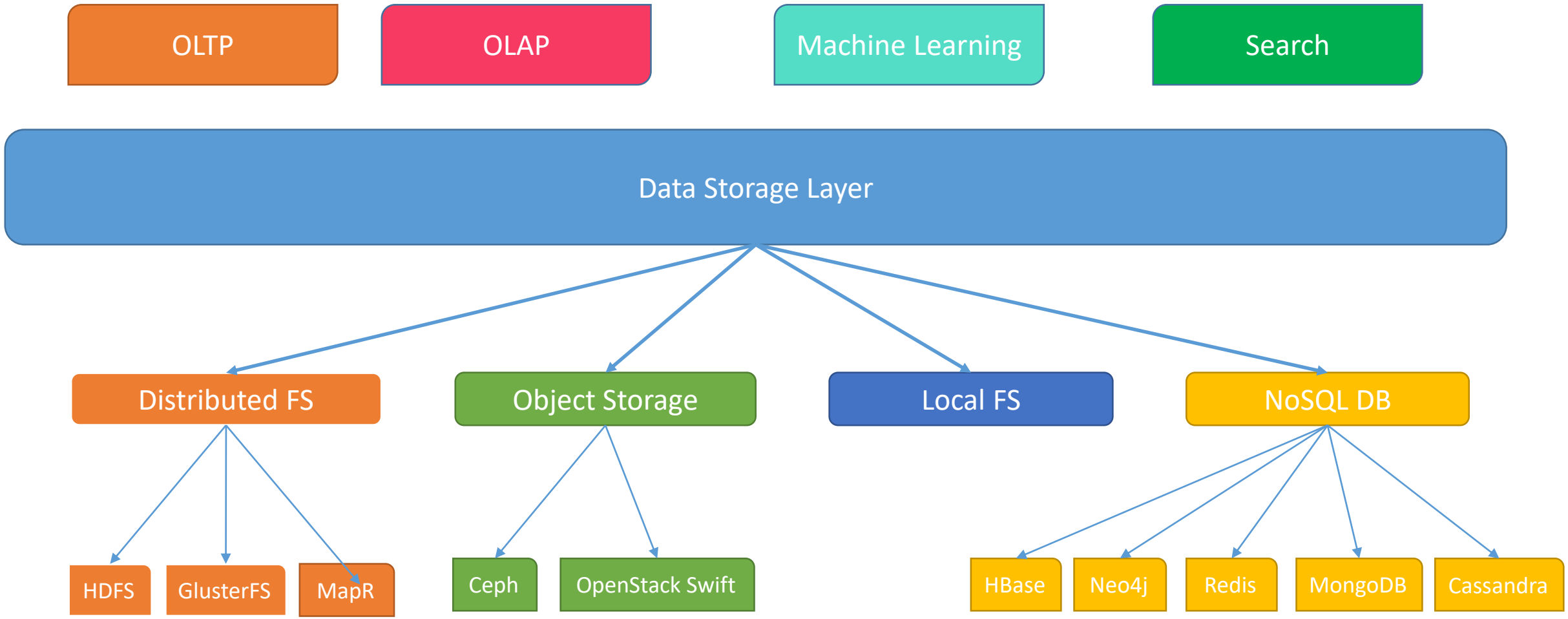


Data Storage Layer

Data Storage Layer

- Depending on the requirements data can placed into Distributed File System, Object Storage, NoSQL Databases, etc.

Data Storage Layer



Rise of the Immutable Datastore

- In a relational database, files are mutable, which means a given cell can be overwritten when there are changes to the data relevant to that cell.
- New architectures offer accumulate-only file system that overwrites nothing. Each file is immutable, and any changes are recorded as separate timestamped files.
- The method lends itself not only to faster and more capable stream processing, but also to various kinds of historical time-series analysis.

Why is immutability so important?

- Fewer dependencies & Higher-volume data handling and improved site-response capabilities
  - Immutable files reduce dependencies or resource contention, which means one part of the system doesn't need to wait for another to do its thing. That's a big deal for large, distributed systems that need to scale and evolve quickly.
- More flexible reads and faster writes
  - writing data without structuring it beforehand means that you can have both fast reads and writes, as well as more flexibility in how you view the data.
- Compatibility with Hadoop & log-based messaging protocols
  - A popular method of distributed storage for less-structured data.
- Suitability for auditability and forensics
  - Only the fully immutable shared log systems preserve the history that is most helpful for audit trails and forensics.

Hadoop Distributed FS

- **Appears as a single disk**
- **Runs on top of a native filesystem**
  - Ext3,Ext4,…
- **Based on Google's Filesystem GFS**
- **Fault Tolerant**
  - Can handle disk crashes, machine crashes, etc...
- **portable Java implementation**

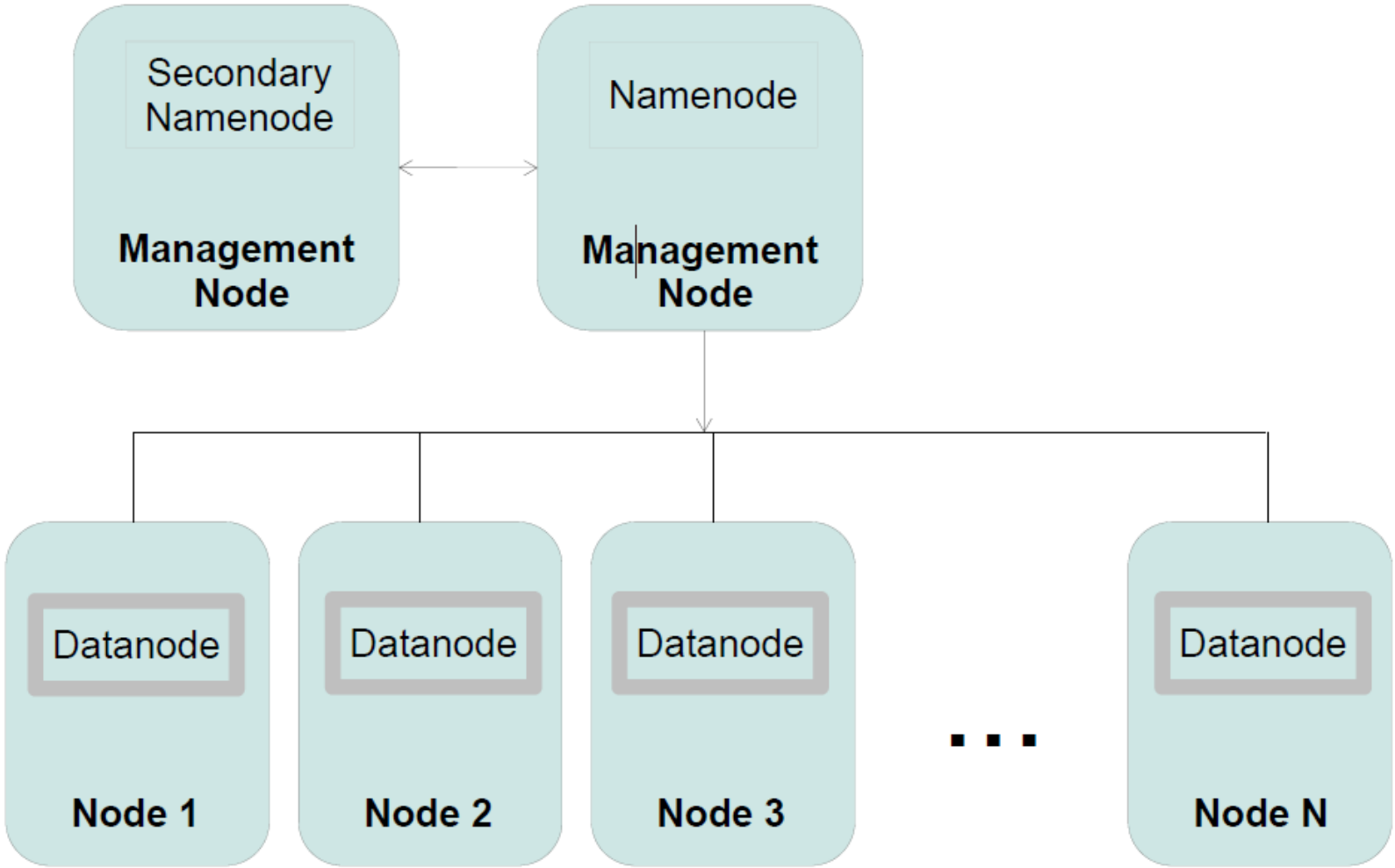
HDFS Architecture

- Master-Slave Architecture

- HDFS Master “Namenode”
  - Manages all filesystem metadata
  - File name to list blocks + location mapping
  - File metadata (i.e. “inode”)
  - Collect block reports from Datanodes on block locations
  - Controls read/write access to files
  - Manages block replication

- HDFS Slaves “Datanodes”
  - Notifies NameNode about block-IDs it has
  - Serve read/write requests from clients
  - Perform replication tasks upon instruction by namenode
  - Rack-aware

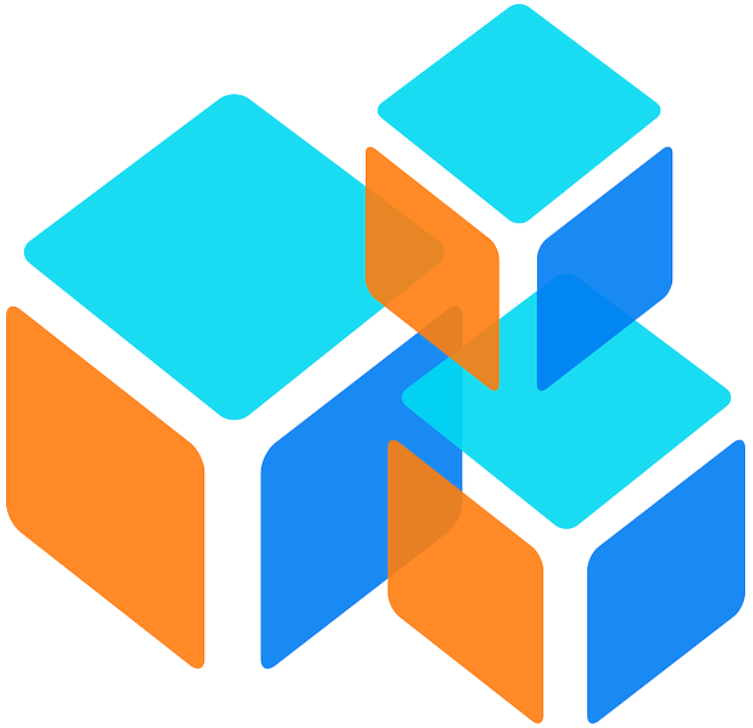
HDFS Daemons



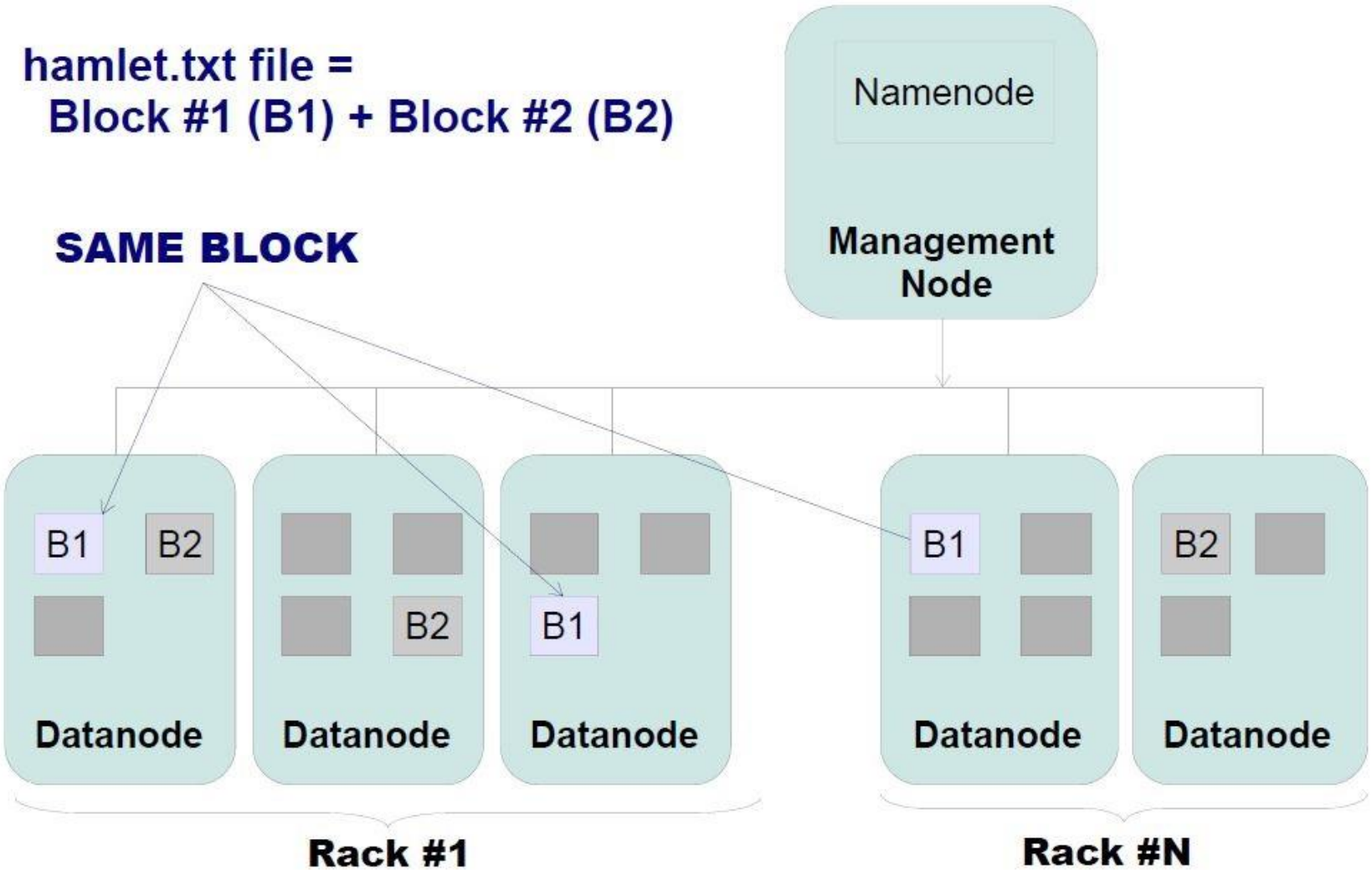
Files and Blocks

- **Files are split into blocks (single unit of storage)**
  - Managed by NameNode, stored by DataNode
  - Transparent to user

- **Replicated across machines at load time**
  - Same block is stored on multiple machines
  - Good for fault-tolerance and access
  - Default replication is 3

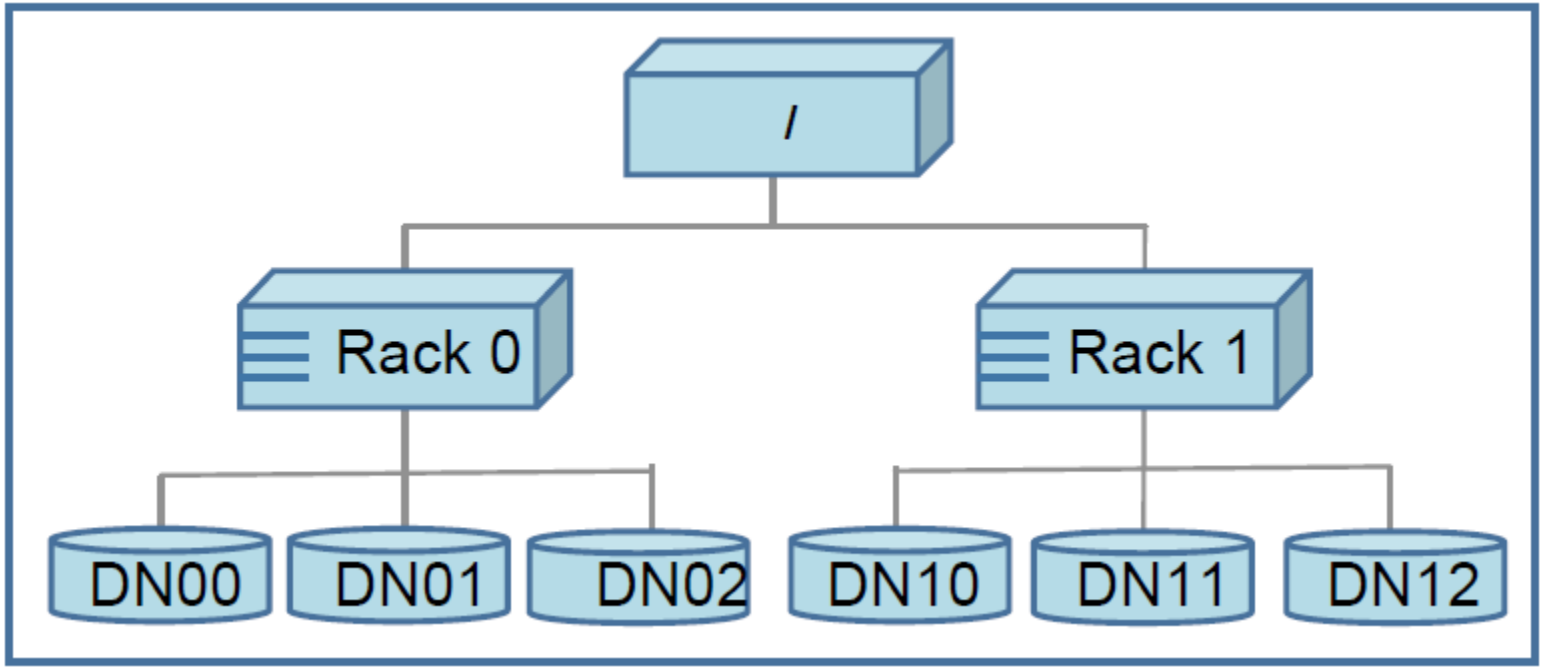


Files and Blocks

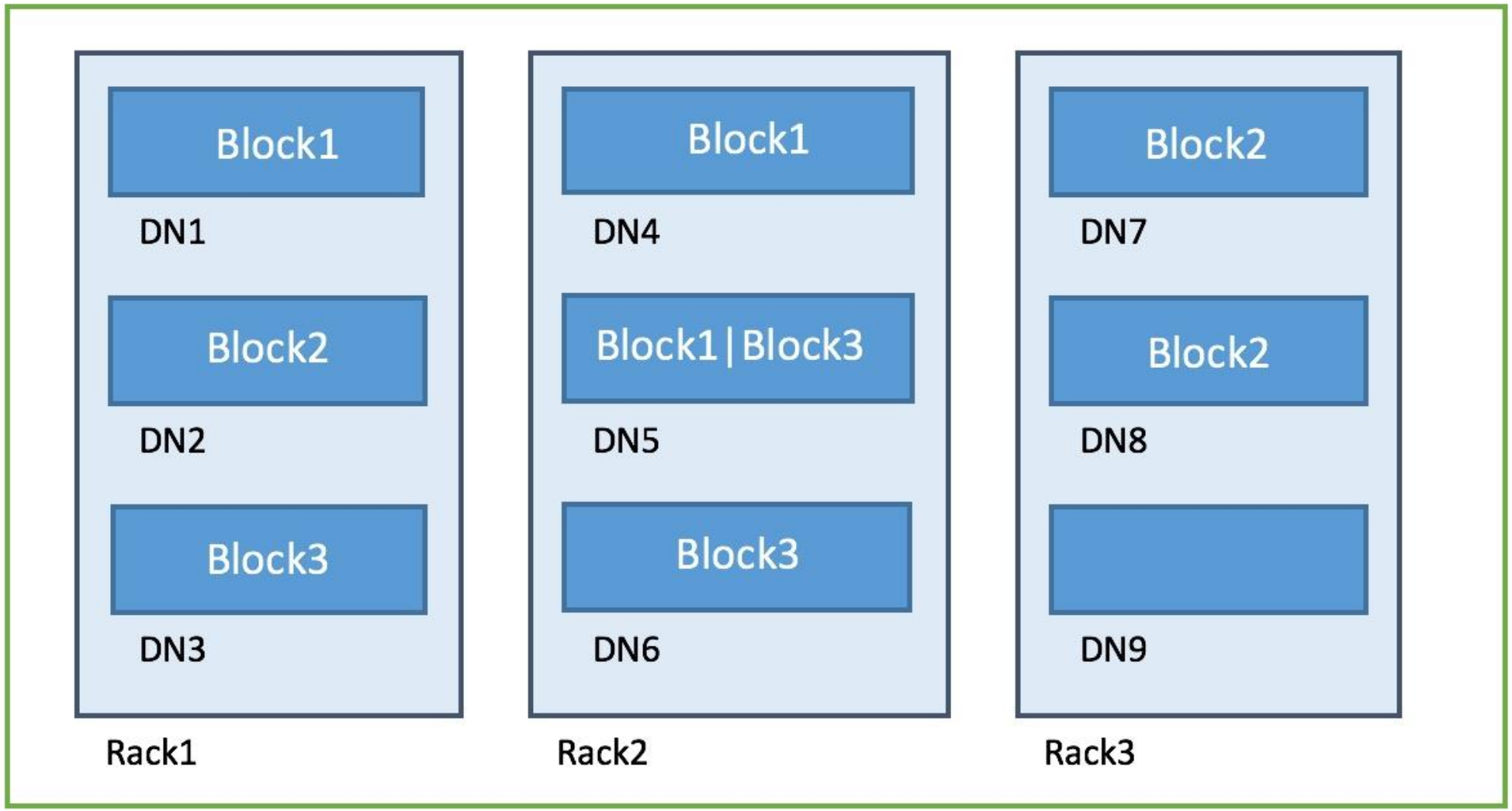


REPLICA MANGEMENT

- A common practice is to spread the nodes across multiple racks
  - A good replica placement policy should improve **data reliability**, **availability**, and **network bandwidth** utilization
    - **Namenode** determines replica placement



Rack Aware


Block1
DN1
Block2
DN2
Block3
DN3
Rack1
Block1
DN4
Block1 | Block3
DN5
Block3
DN6
Rack2
Block2
DN7
Block2
DN8
DN9
Rack3

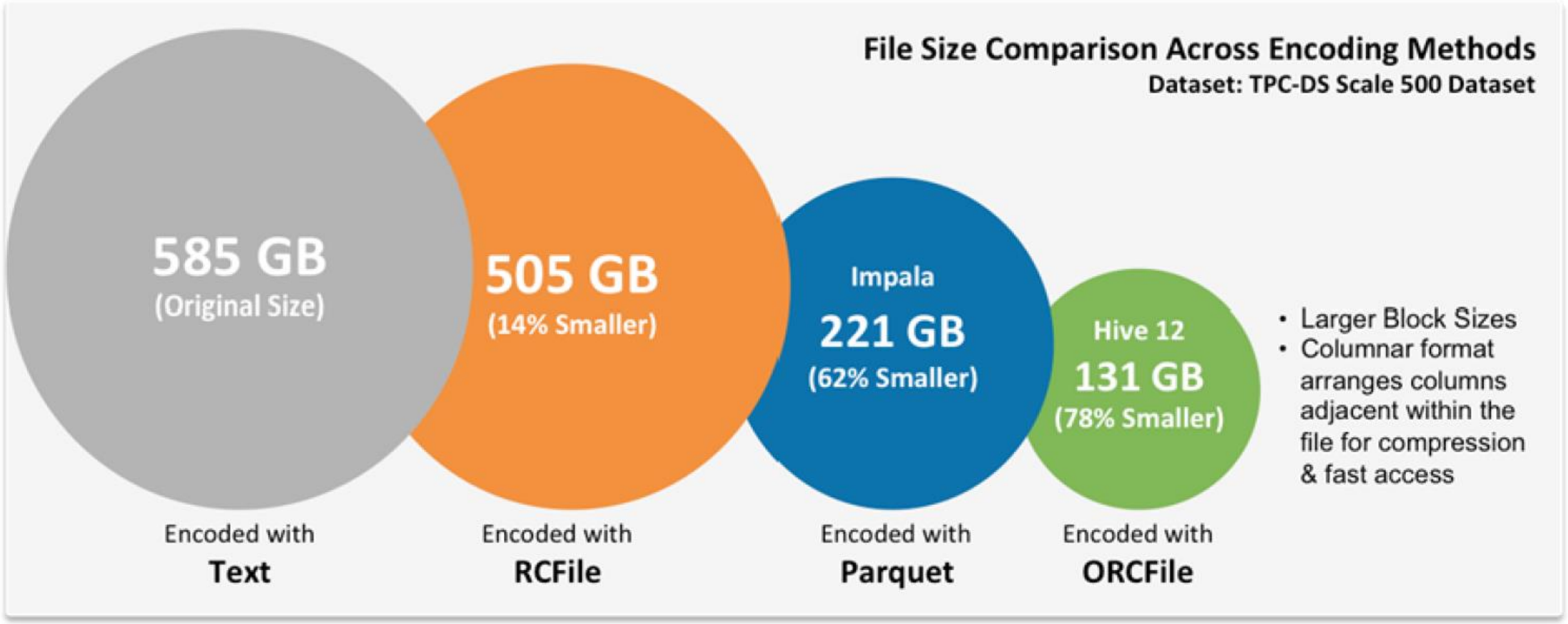
Data Encoding

- A huge bottleneck for HDFS-enabled applications like MapReduce and Spark is the time it takes to find relevant data in a particular location and the time it takes to write the data back to another location.
- Choosing an appropriate file format can have some significant benefits:
  - **Faster read times**
  - **Faster write times**
  - **Splittable files** (so you don't need to read the whole file, just a part of it)
  - **Schema evolution support** (allowing you to change the fields in a dataset)
  - Advanced **compression** support (compress the files with a compression codec without sacrificing these features)

Data Encoding

- **The format of the files you can store on HDFS, like any file system, is entirely up to you.**
  - However unlike a regular file system, HDFS is best used in conjunction with a data processing toolchain like MapReduce or Spark.
  - These processing systems typically (although not always) operate on some form of textual data like webpage content, server logs, or location data.

Encoding Technologies



EncodingTechnologies

- **TextFiles (E.G. CSV, JSON)**
- **Sequence files were originally designed for MapReduce**
- **Avro**
- **Columnar File Formats**
  - RCFile
  - Apache Orc
  - Apache Parquet

Apache orc™

NOSQL Databases

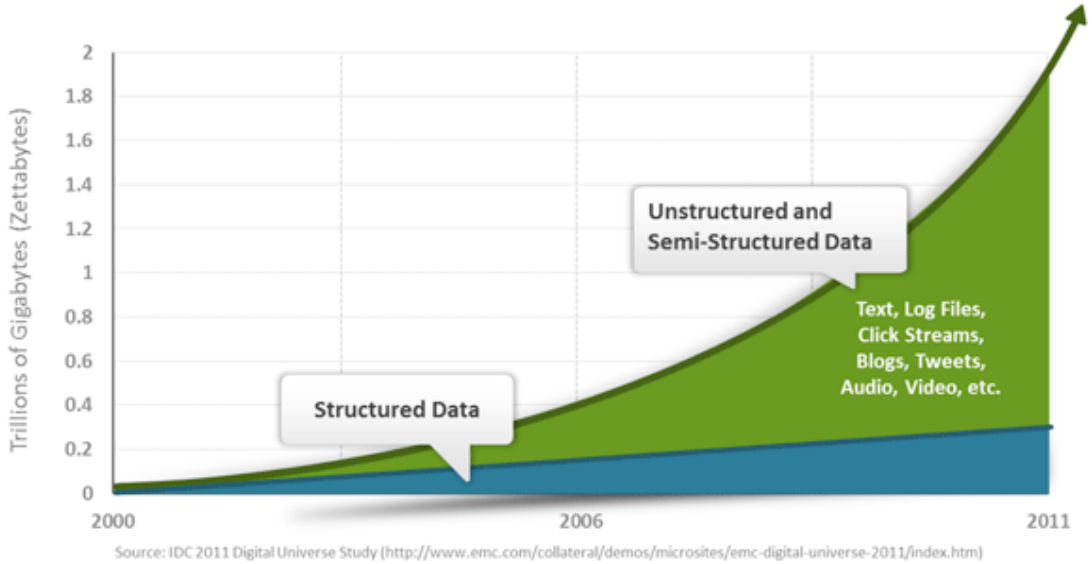
Nosql

- Relational database (RDBMS) technology
  - Has not fundamentally changed in over 40 years
  - Default choice for holding data behind many web apps
  - Handling more users means adding a bigger server
- Extend the Scope of RDBMS
  - Caching
  - Master/Slave
  - Table Partitioning
  - Federated Tables
  - Sharding

Something Changed!

- Organizations work with different type of data, often semi or un-structured.
- And they have to store, serve and process huge amount of data.

- There were a need for systems that could:
  - work with different kind of data format,
  - Do not require strict schema,
  - and are easily scalable.



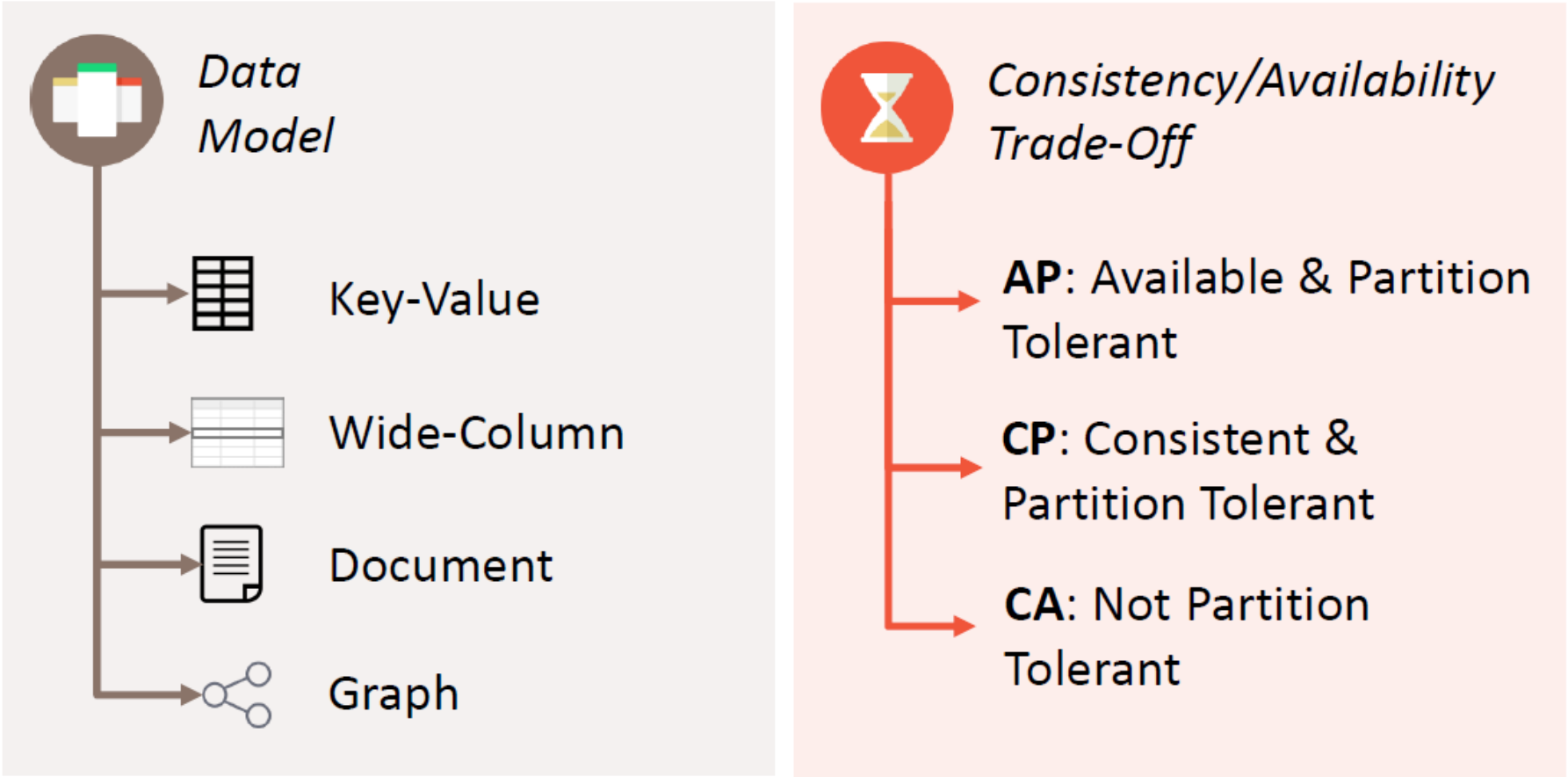
RDBMS with Extended Functionality

Vs.

Systems Built from Scratch with Scalability in Mind

Nosql System Classification

- Tow common criteria:



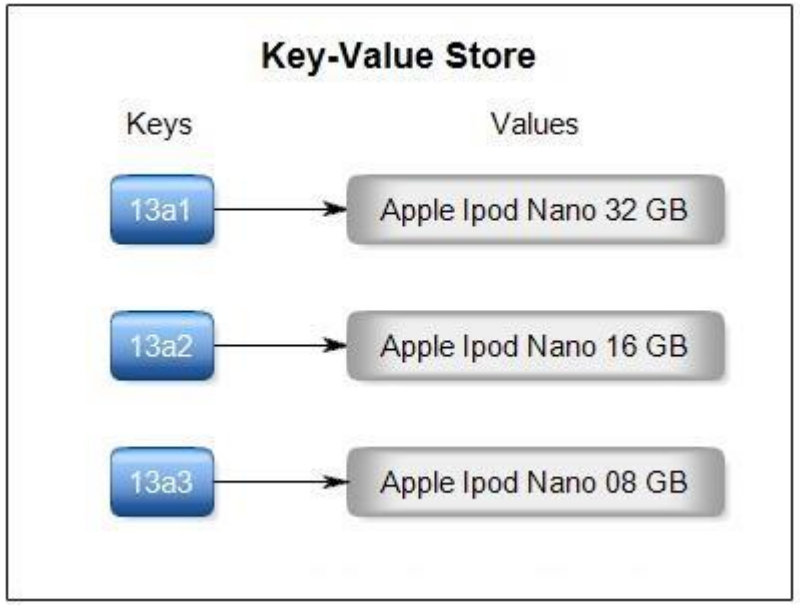
Nosql Data Model

Document Database	Graph Databases
Couchbase MarkLogic mongoDB	Neo4j InfiniteGraph The Distributed Graph Database
Key-Value Databases	**Wide Column Stores**
redis amazon DynamoDB AEROSPIKE riak	accumulo HYPERTABLE INC Cassandra APACHE HBASE Amazon SimpleDB

Key Value Store

- Extremely simple interface:
  - Data model: (key, value) pairs
  - Basic Operations: : Insert(key, value), Fetch(key), Update(key), Delete(key)

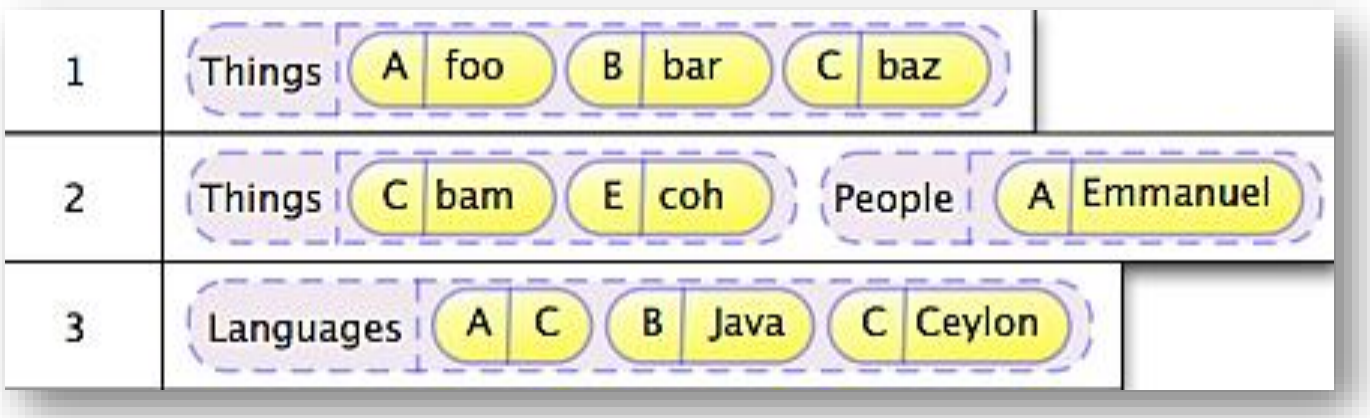
- Pros:
  - very fast
  - very scalable
  - simple model
  - able to distribute horizontally
- Cons:
  - many data structures (objects) can't be easily modeled as key value pairs



key-value

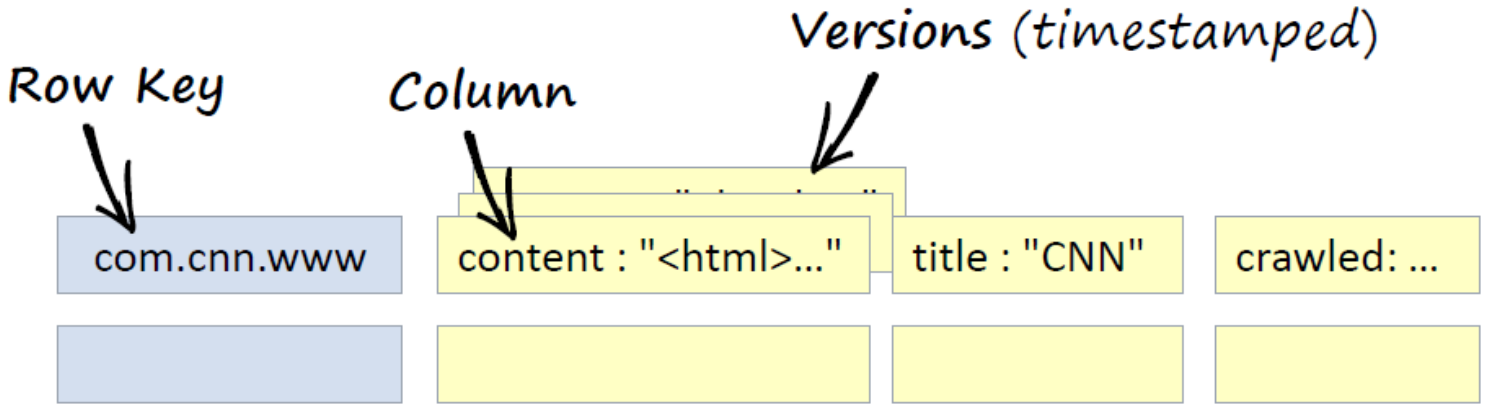
Amazon DynamoDB (Beta) ORACLE BERKELEY DB 11g redis

Column Oriented Store



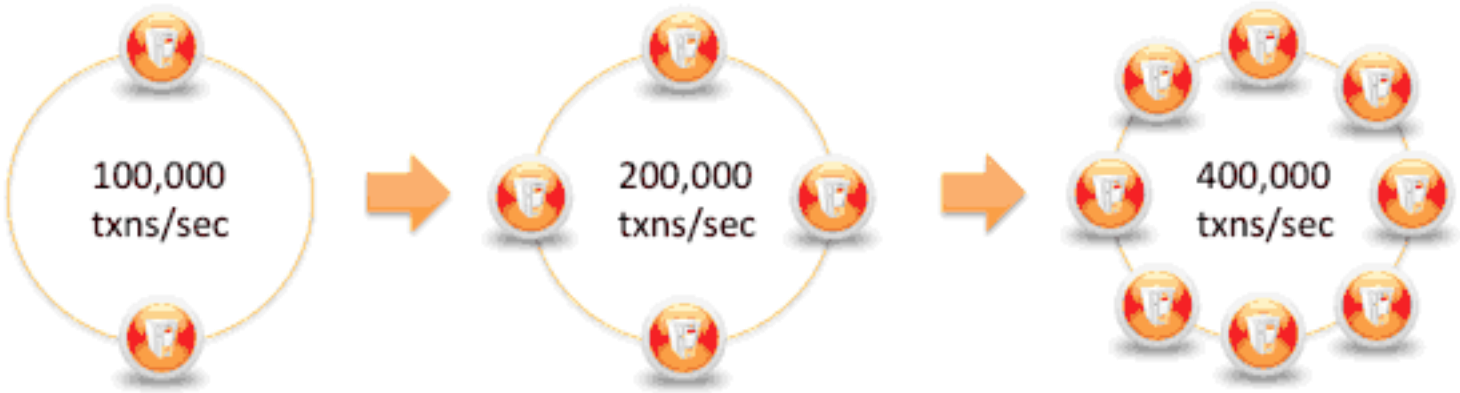
Column-oriented

- Store data in columnar format
- Allow key-value pairs to be stored (and retrieved on key) in a massively parallel system
  - data model: families of attributes defined in a schema, new attributes can be added online
  - storing principle: big hashed distributed tables
  - properties: partitioning (horizontally and/or vertically), high availability etc. completely transparent to application
- Column Oriented Store
  - BigTable
  - Hbase
  - Hypertable
  - Cassandra



Cassandra

- All nodes are similar
- Data can have expiration (set on INSERT)
- Map/reduce possible with Apache Hadoop
- Rich Data Model (columns, composites, counters, secondary indexes, map, set, list, counters)

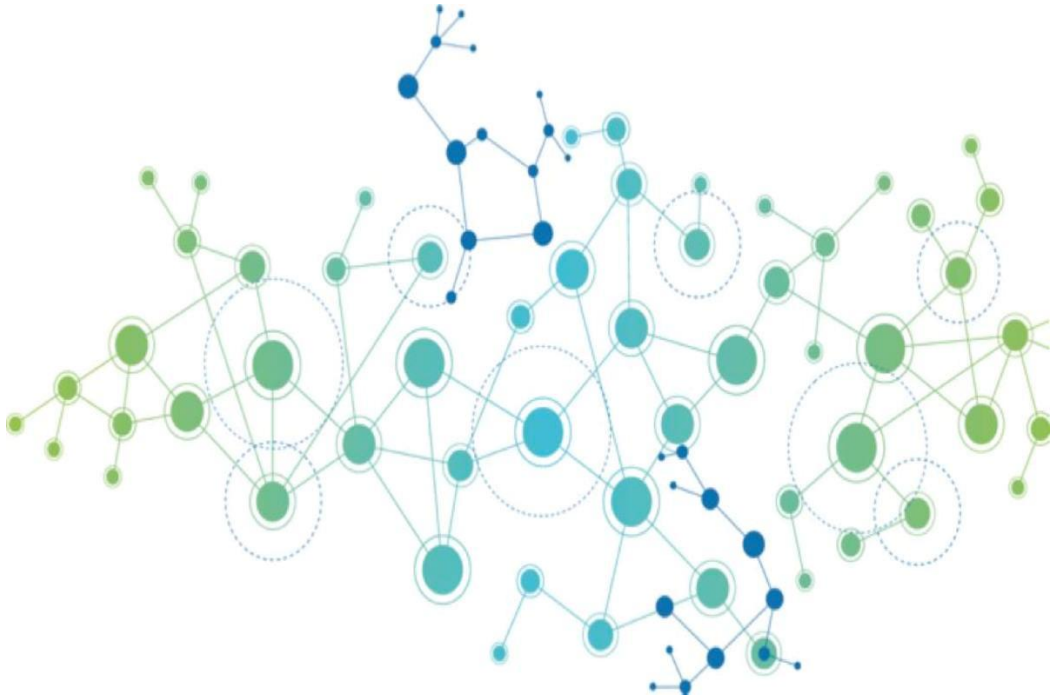


Document Store

Document Store

- Schema Free.
- Usually JSON (BSON) like interchange model, which supports lists, maps, dates, Boolean with nesting
- Query Model: JavaScript or custom.
- Aggregations: Map/Reduce.
- Indexes are done via B-Trees.
- Examples:
  - MongoDB
  - CouchDB
  - CouchBase
  - RethinkDB

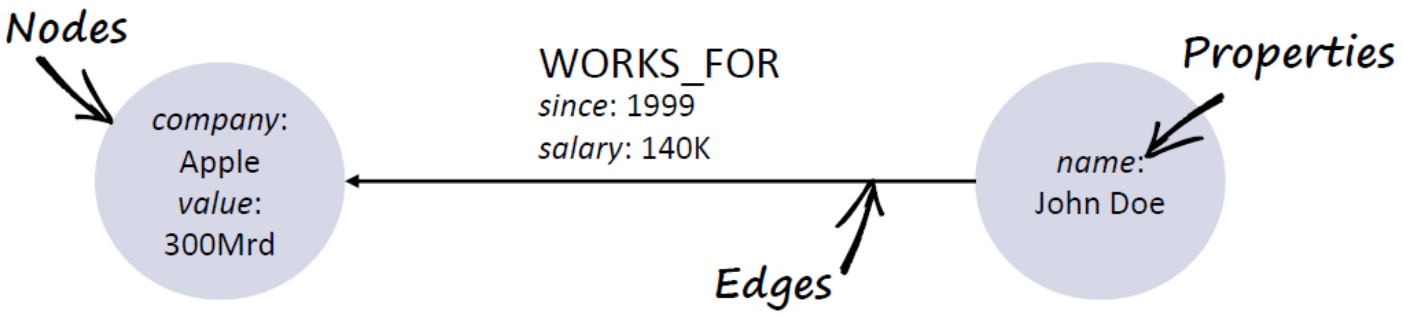
Graph Databases



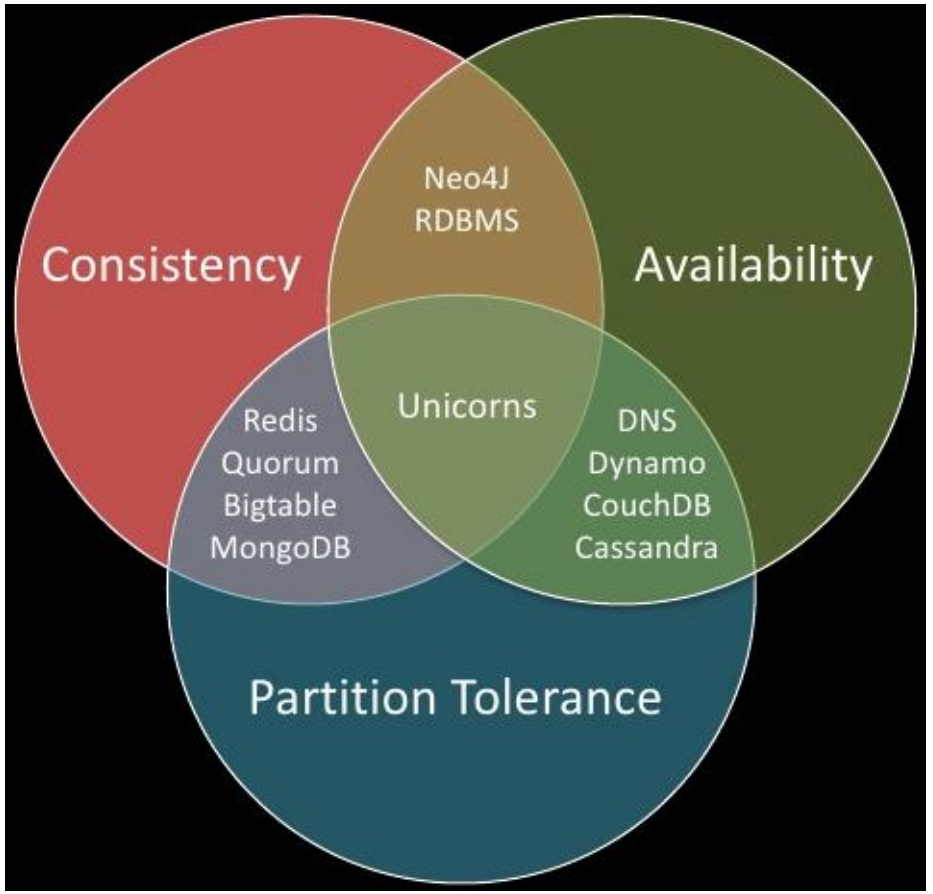
Graph Databases

- They are significantly different from the other three classes of NoSQL databases.
- Are based on the concepts of Vertex and Edges
- Relational DBs can model graphs, but it is expensive.

- Graph Store
  - Neo4j
  - Titan
  - OrientDB



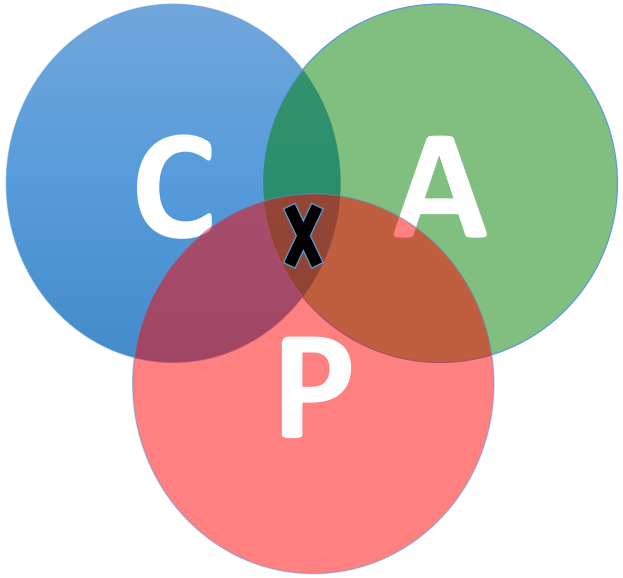
Nosql Implementation Considerations



CAP Theorem

- Conjectured by Prof. Eric Brewer at PODC (Principle of Distributed Computing) 2000 keynote talk
- Described the *trade-offs involved in distributed system*
- It is impossible for a web service to provide following *three guarantees at the same time*:
  - **Consistency**
  - **Availability**
  - **Partition-tolerance**

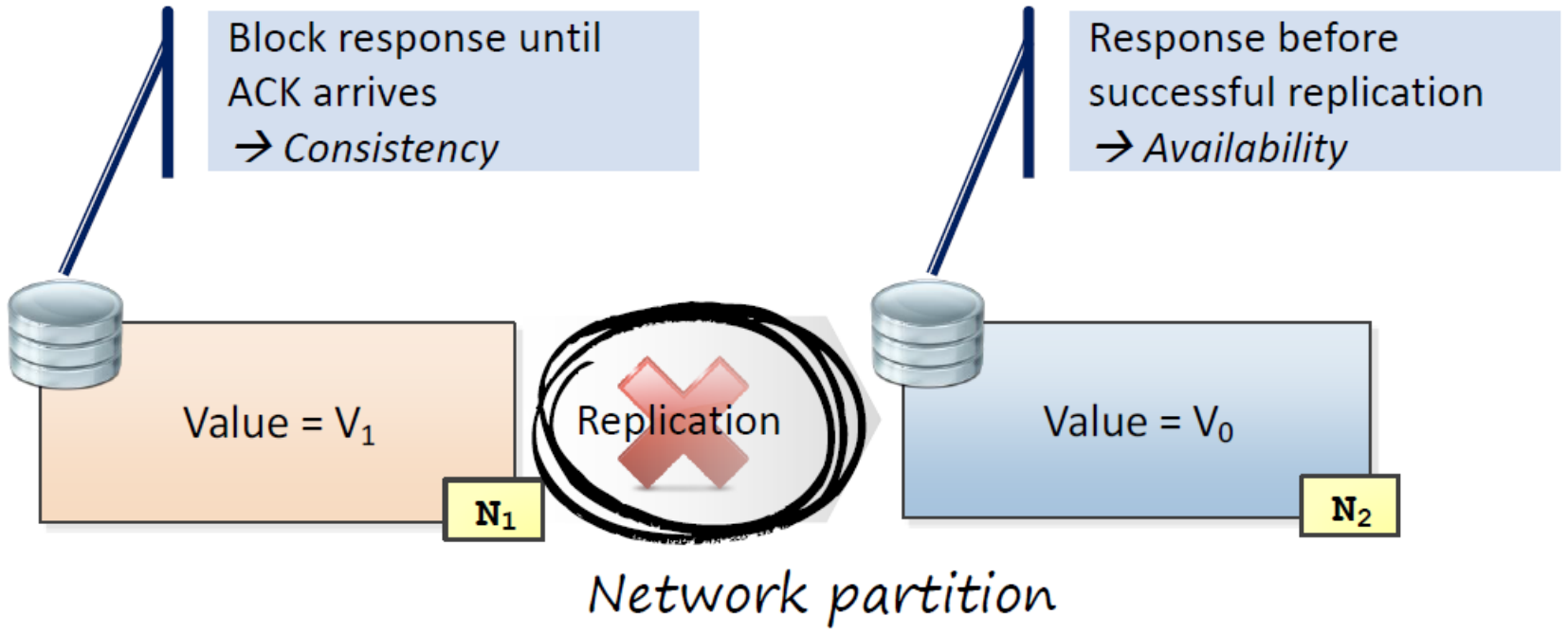
CAP Theorem



- **C**onsistency:
  - All nodes should see the same data at the same time
- **A**vailability:
  - Node failures do not prevent survivors from continuing to operate
- **P**artition-tolerance:
  - The system continues to operate despite network partitions
- A distributed system can satisfy any two of these guarantees at the same time **but not all three**

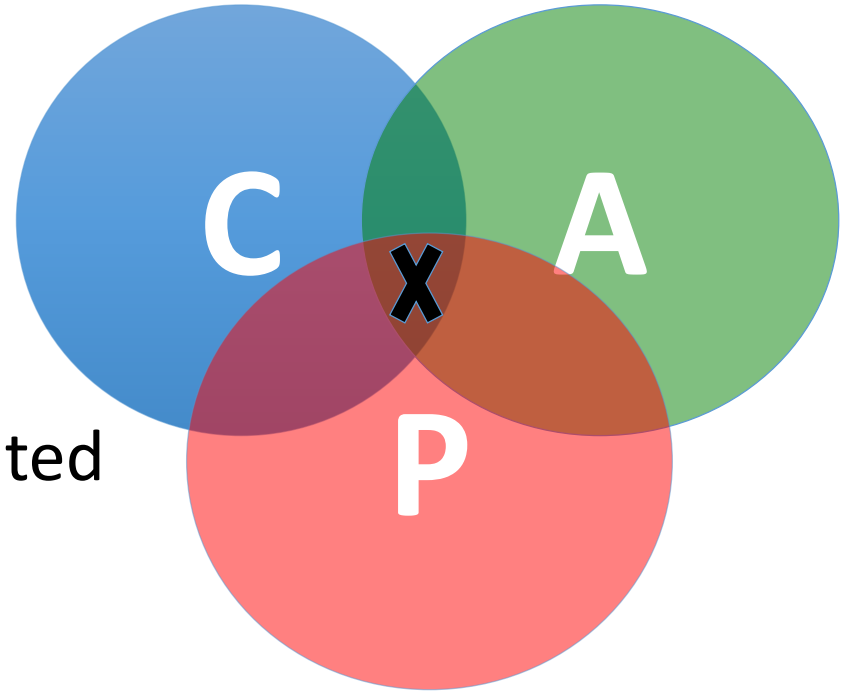
CAP-Theorem: simplified proof

- Problem: when a network partition occurs, either consistency or availability have to be given up



Revisit CAP Theorem

- Of the following three guarantees potentially offered a by distributed systems:
  - Consistency
  - Availability
  - Partition tolerance
- Pick two
- This suggests there are three kinds of distributed systems:
  - CP
  - AP
  - CA



Any problems?

CAP Theorem 12 year later

- Prof. Eric Brewer: father of CAP theorem
  - “The “2 of 3” formulation was always **misleading** because it tended to oversimplify the tensions among properties. ...
  - **CAP prohibits only a tiny part of the design space**: *perfect availability and consistency in the presence of partitions*, which are rare.”

http://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed

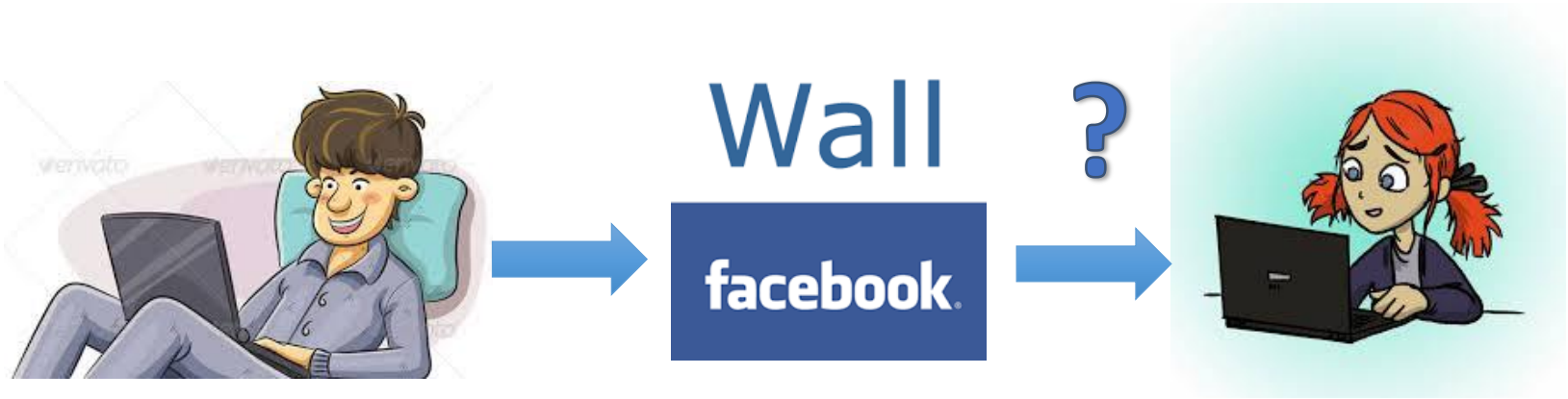
Types of Consistency

- Strong Consistency
  - After the update completes, **any subsequent access** will return the **same** updated value.
- Weak Consistency
  - It is **not guaranteed** that subsequent accesses will return the updated value.
- Eventual Consistency
  - Specific form of weak consistency
  - It is guaranteed that if **no new updates** are made to object, **eventually** all accesses will return the last updated value (e.g., *propagate updates to replicas in a lazy fashion*)

Eventual Consistency - A Facebook Example

- Bob finds an interesting story and shares with Alice by posting on her Facebook wall
- Bob asks Alice to check it out
- Alice logs in her account, checks her Facebook wall but finds:

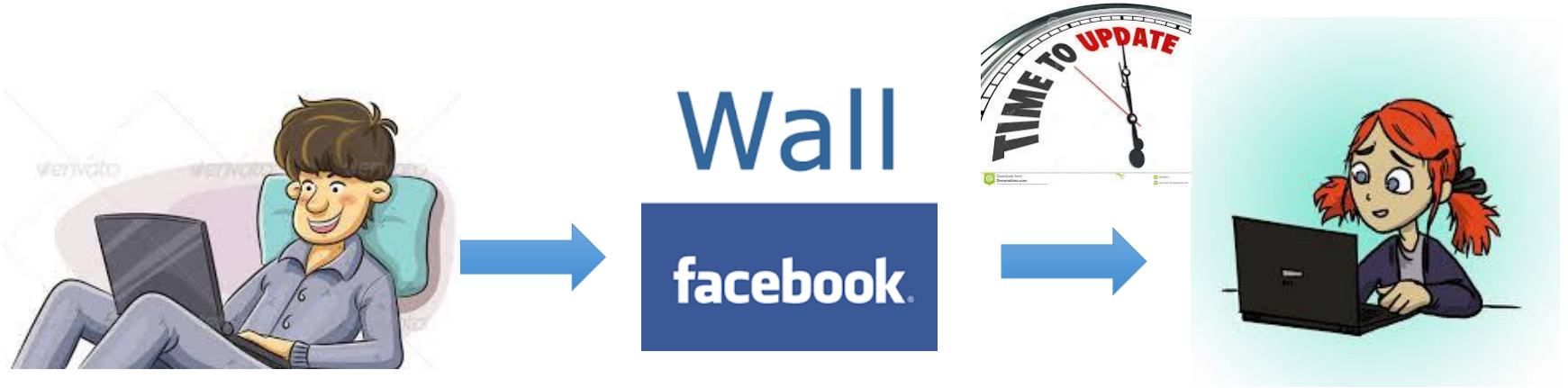
  - **Nothing is there!**



Eventual Consistency - A Facebook Example

- Bob tells Alice to wait a bit and check out later
- Alice waits for a minute or so and checks back:

  - **She finds the story Bob shared with her!**



Eventual Consistency - A Facebook Example

- Reason: it is possible because Facebook uses an **eventual consistent model**
- Why Facebook chooses eventual consistent model over the strong consistent one?
  - Facebook has more than 1 billion active users
  - It is non-trivial to efficiently and reliably store the huge amount of data generated at any given time
  - Eventual consistent model offers the option to **reduce the load and improve availability**

What if there are no partitions?

- Tradeoff between **Consistency** and **Latency**:
- Caused by the **possibility of failure** in distributed systems
  - High availability -> replicate data -> consistency problem
- Basic idea:
  - Availability and latency are arguably **the same thing**: unavailable -> extreme high latency
  - Achieving different levels of consistency/availability takes different amount of time

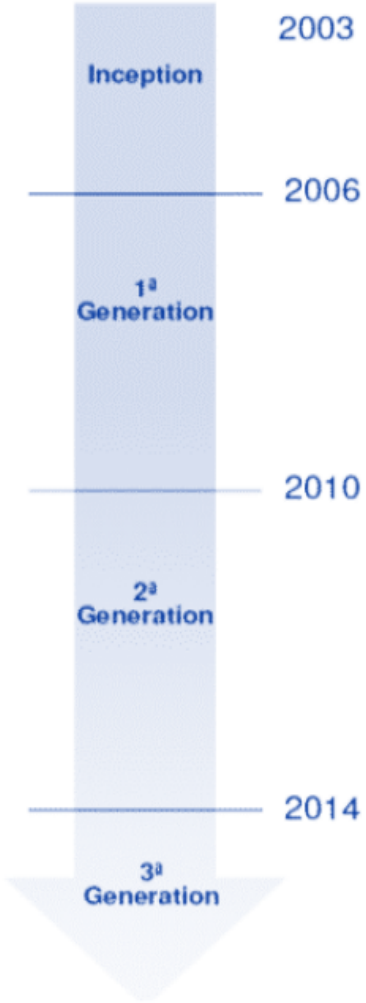
CAP -> PACELC

- A more complete description of the space of potential tradeoffs for distributed system:
  - If there is a **partition (P)**, how does the system trade off **availability and consistency (A and C)**; **else (E)**, when the system is running normally in the absence of partitions, how does the system trade off **latency (L) and consistency (C)**?

Abadi, Daniel J. "Consistency tradeoffs in modern distributed database system design." Computer-IEEE Computer Magazine 45.2 (2012): 37.

Data Processing Layer

Data Processing Layer



Batch processing

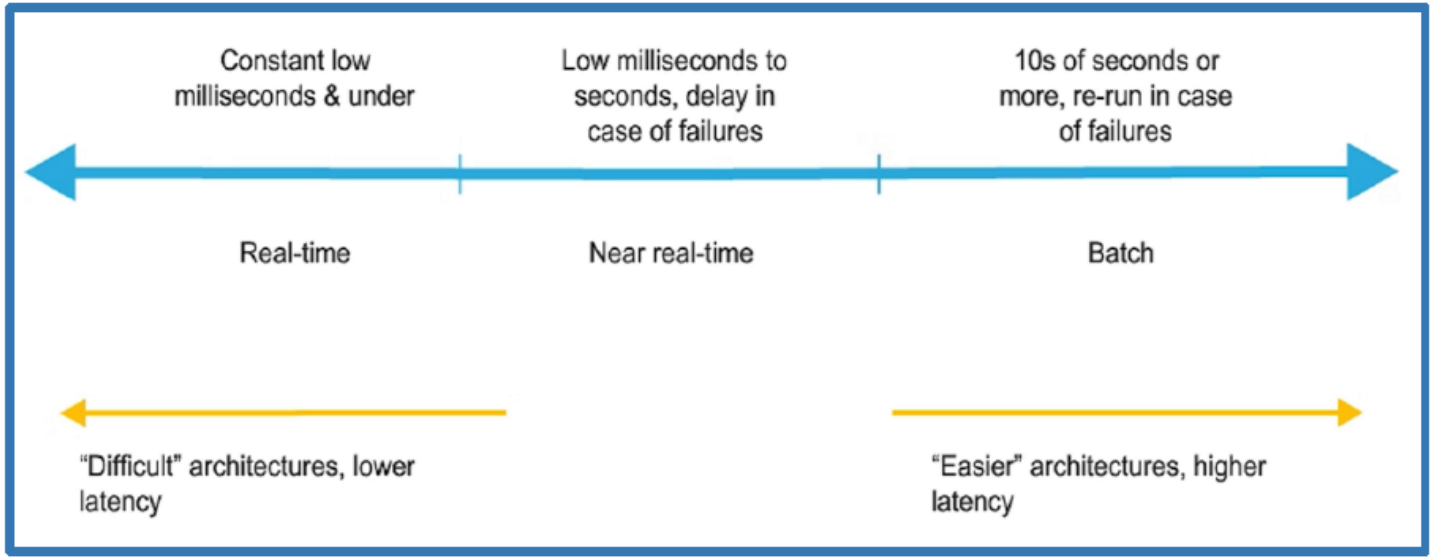
- Large amount of statics data
- Generally incurs a high-latency / Volume

Real-time processing

- Compute streaming data
- Low latency
- Velocity

Hybrid computation

- Lambda Architecture
- Volume + Velocity

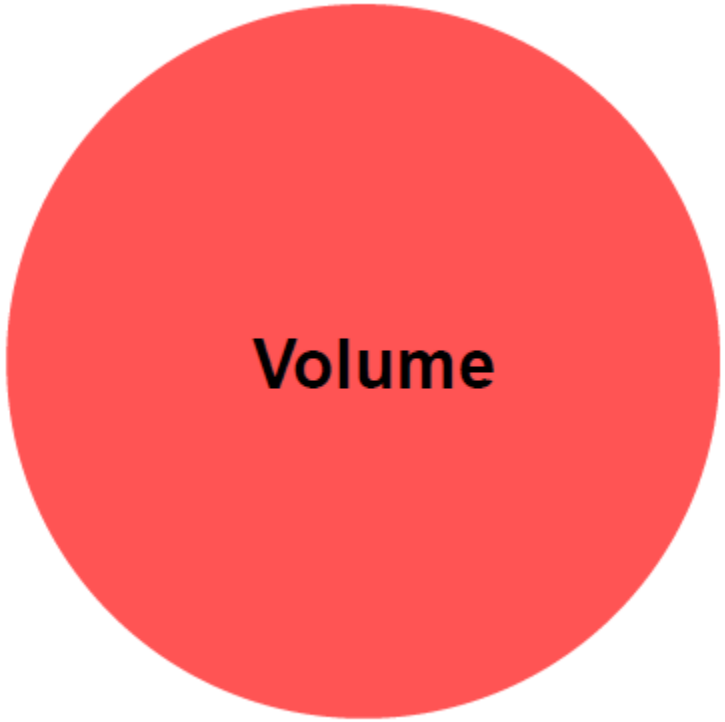


Data Processing Layer

- Processing is provided for batch, streaming and near-realtime use cases
- Scale-Out Instead of Scale-Up
- Fault-Tolerant methods
- Process is moved to data

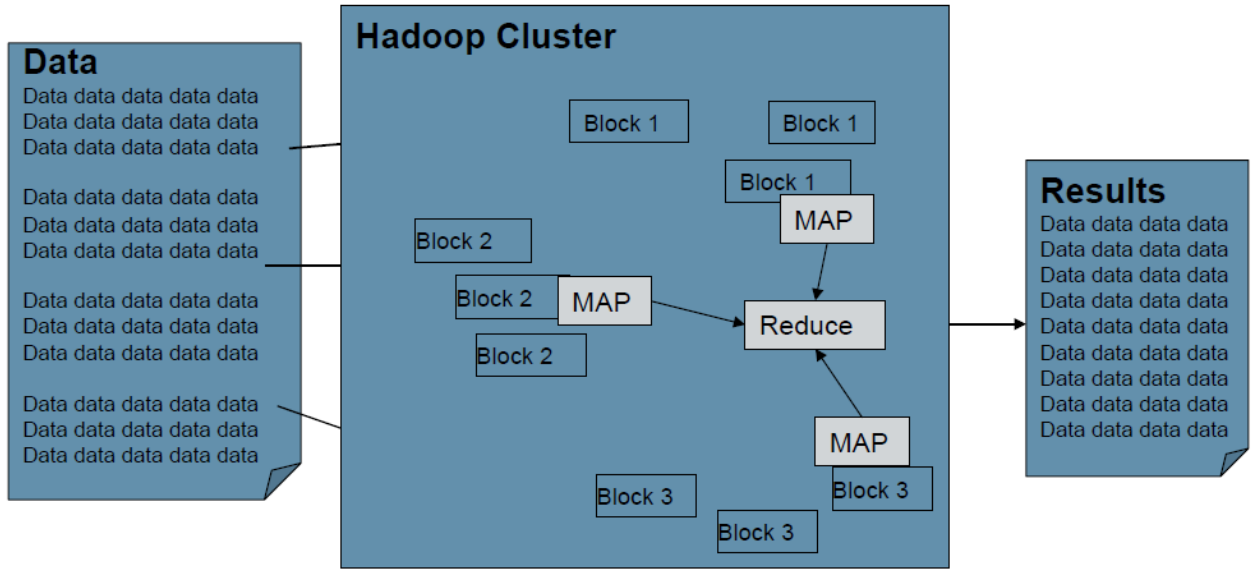
Batch Processing

- **Scalable**
- Large amount of **static** data
- Distributed
- Parallel
- Fault tolerant
- High latency



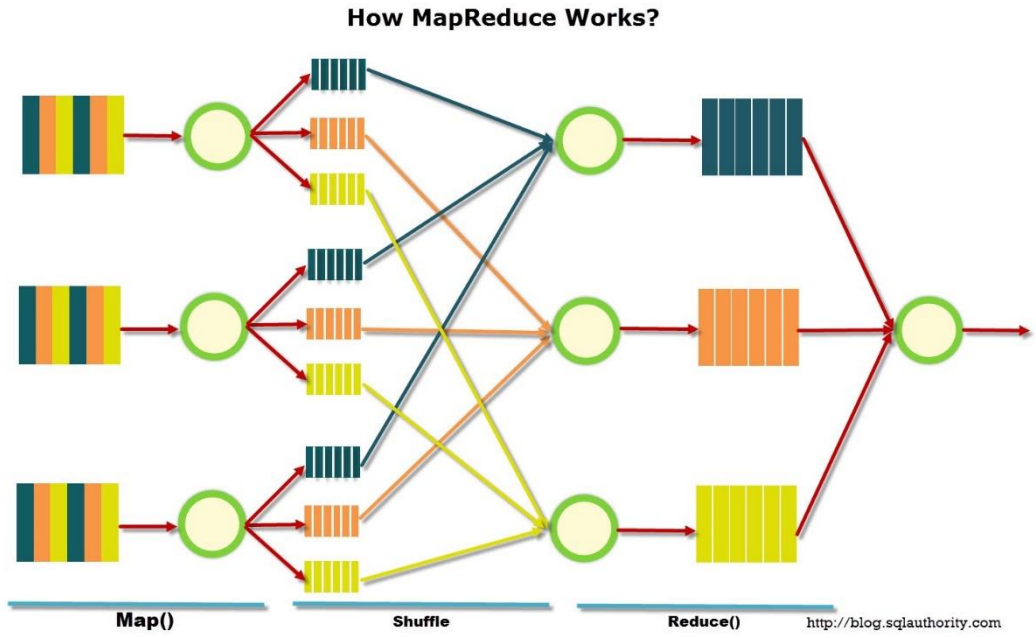
Mapreduce Model

- MapReduce was designed by Google as a programming model for processing large data sets with a parallel, distributed algorithm on a cluster.
  - MapReduce can take advantage of the locality of data

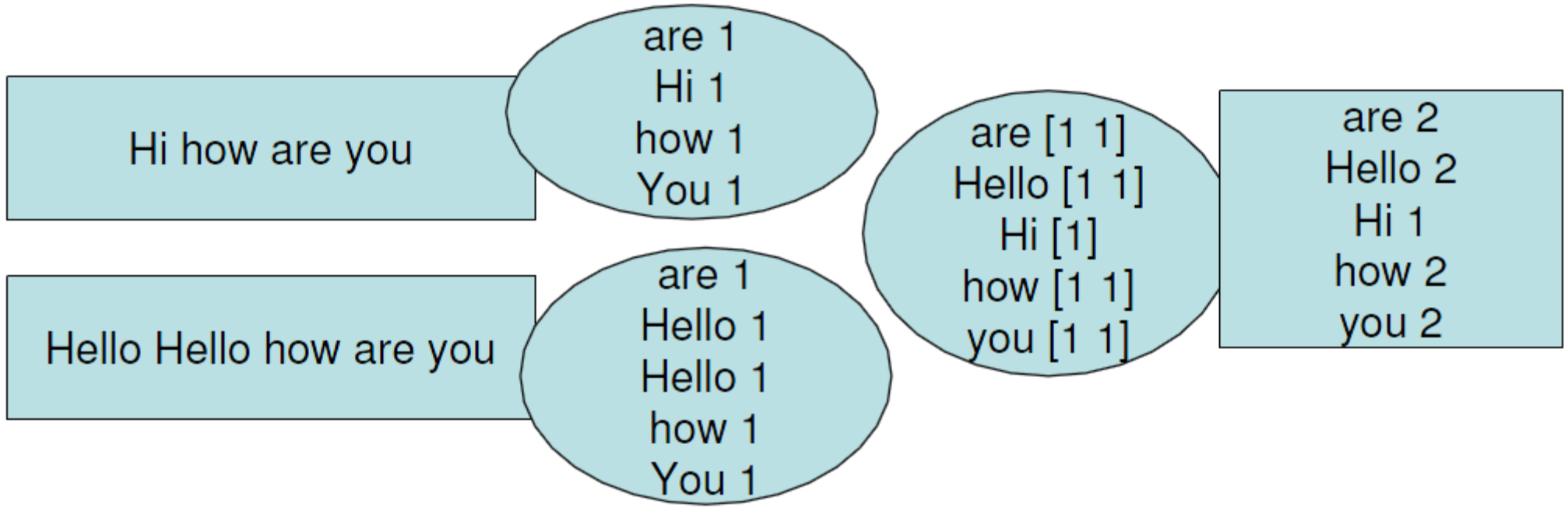


- Map()
  - Process a key/value pair to generate intermediate key/value pairs

- Reduce()
  - Merge all intermediate values associated with the same key
  - eg. <key, [value1, value2,..., valueN]>



Mapreduce Data Flow



Apache Hadoop Ecosystem

Ambari
Provisioning, Managing and Monitoring Hadoop Clusters

Sqoop
Data Exchange

Flume
Log Collector

Zookeeper
Coordination

Oozie
Workflow

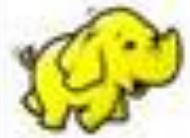
Pig
Scripting

Mahout
Machine Learning

R Connectors
Statistics

Hive
SQL Query

Hbase
Columnar Store



YARN Map Reduce v2
Distributed Processing Framework

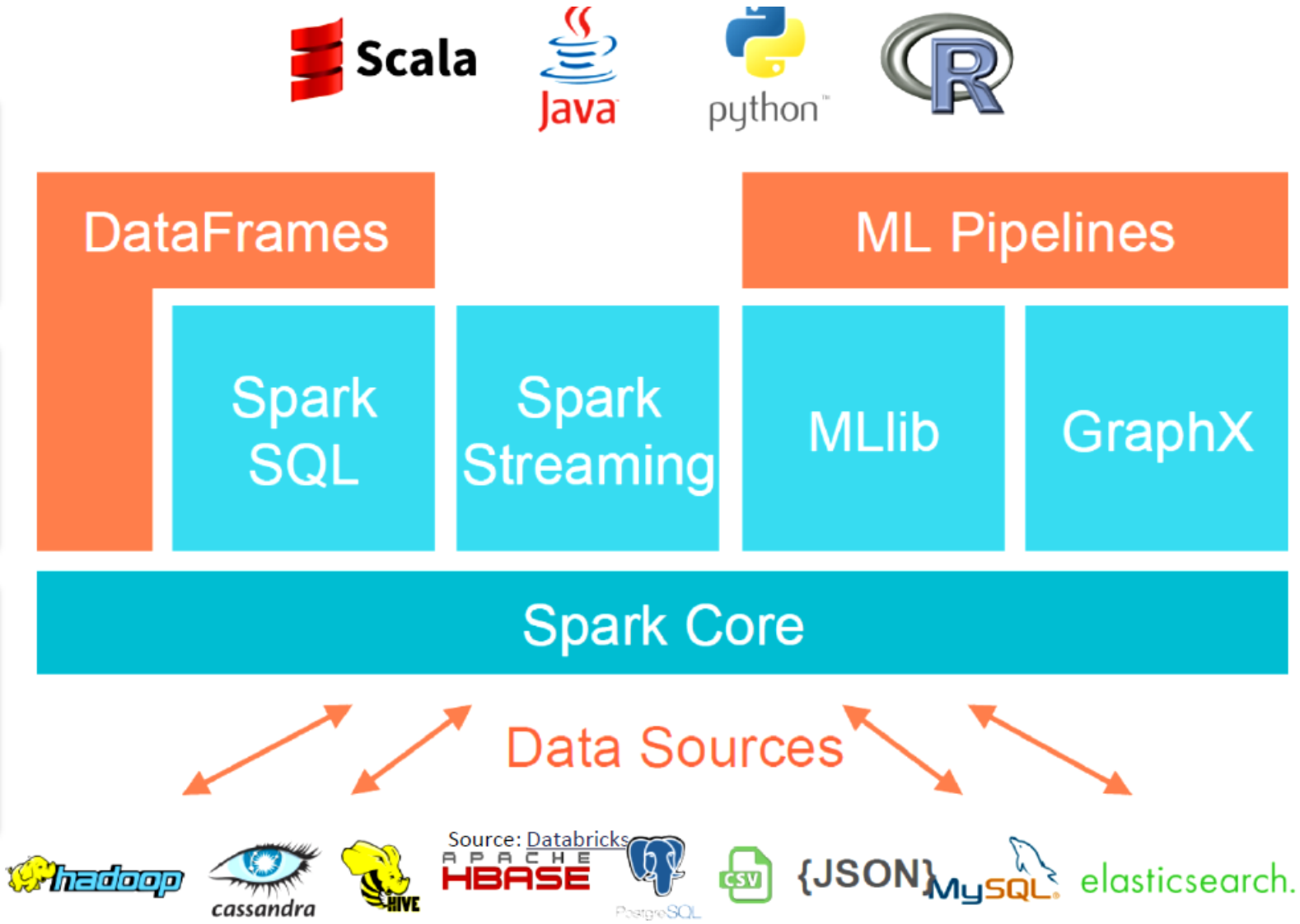
HDFS
Hadoop Distributed File System

Spark Stack

Goal: unified engine across data sources, workloads and environments

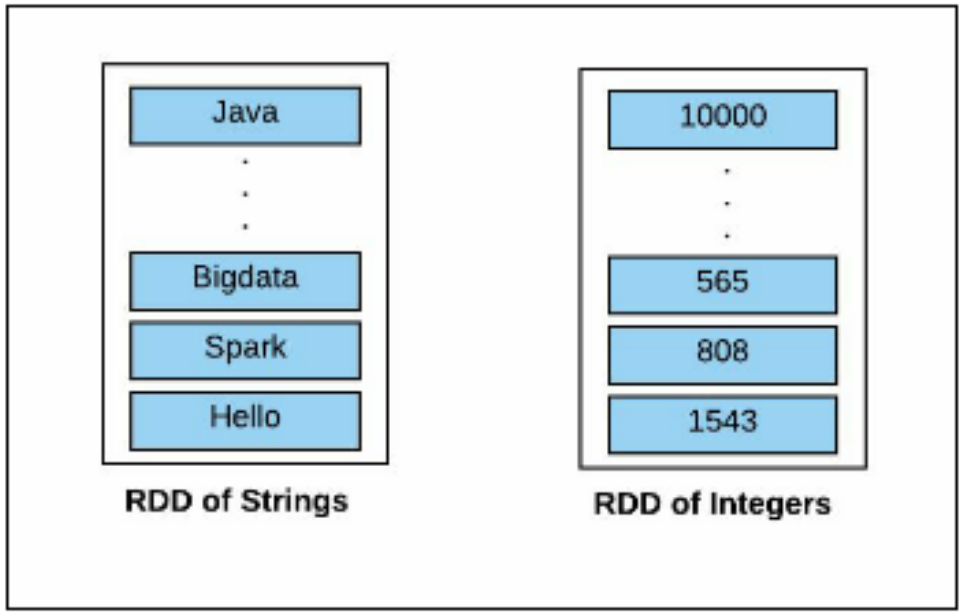
DataFrame is a distributed collection of data organized into named columns

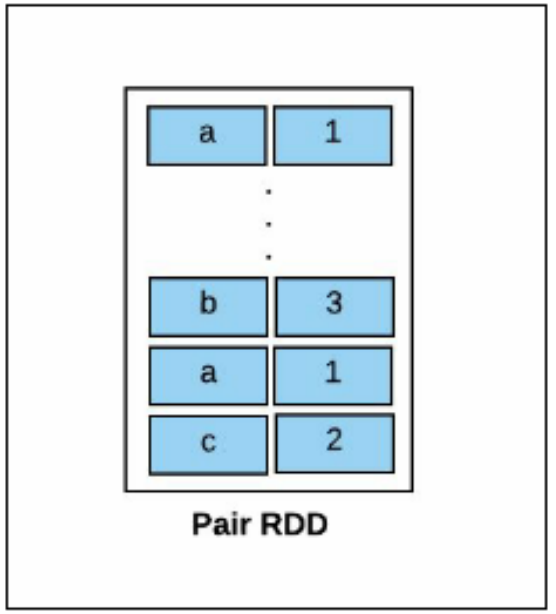
ML pipeline to define a sequence of data pre-processing, feature extraction, model fitting, and validation stages



RDD

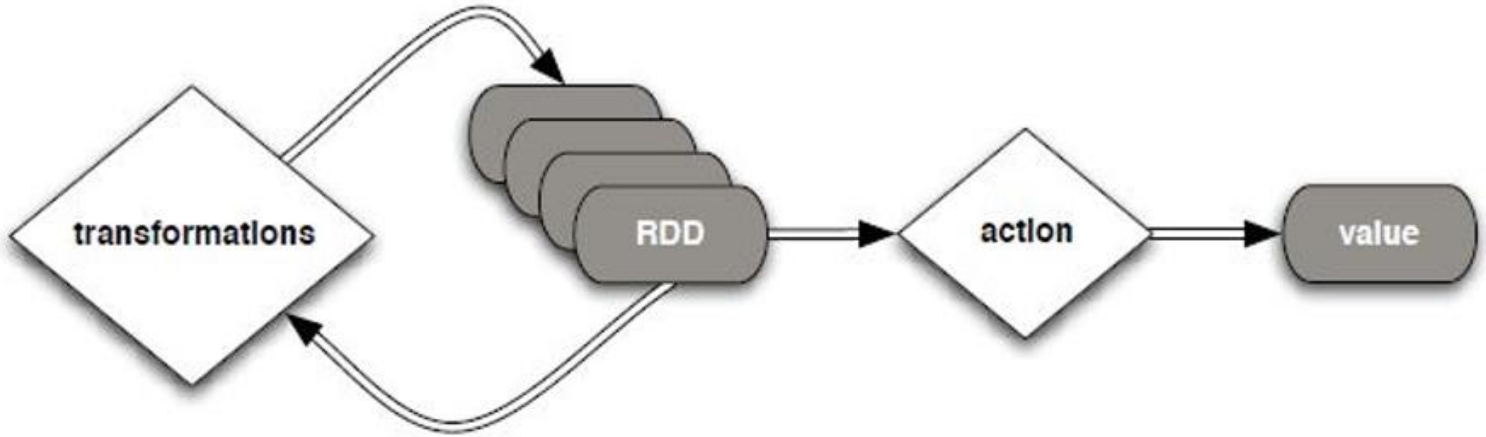
- **R**esilient **D**istributed **D**atasets (RDD) are the primary abstraction in Spark – a fault-tolerant collection of elements that can be operated on in parallel

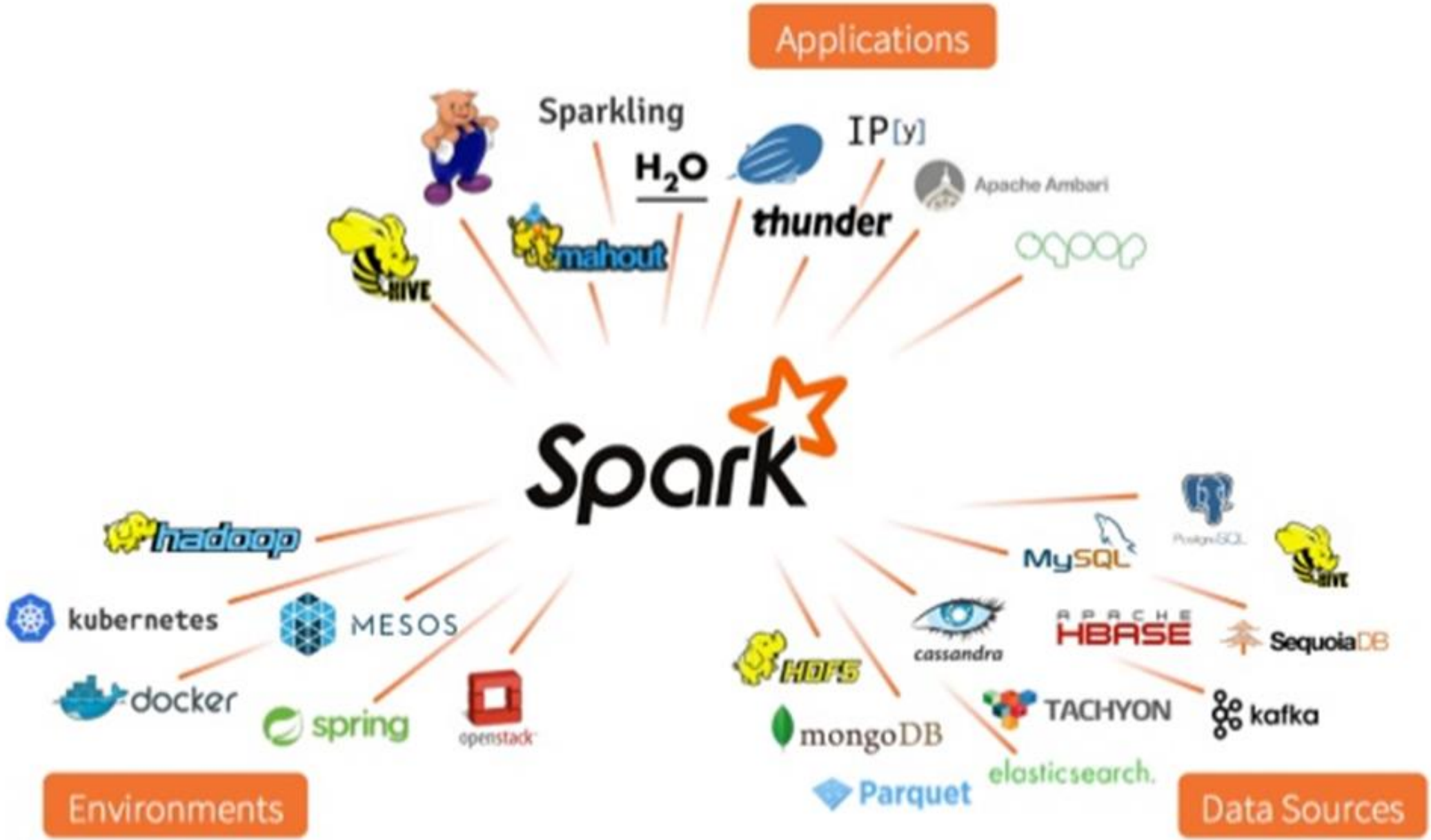




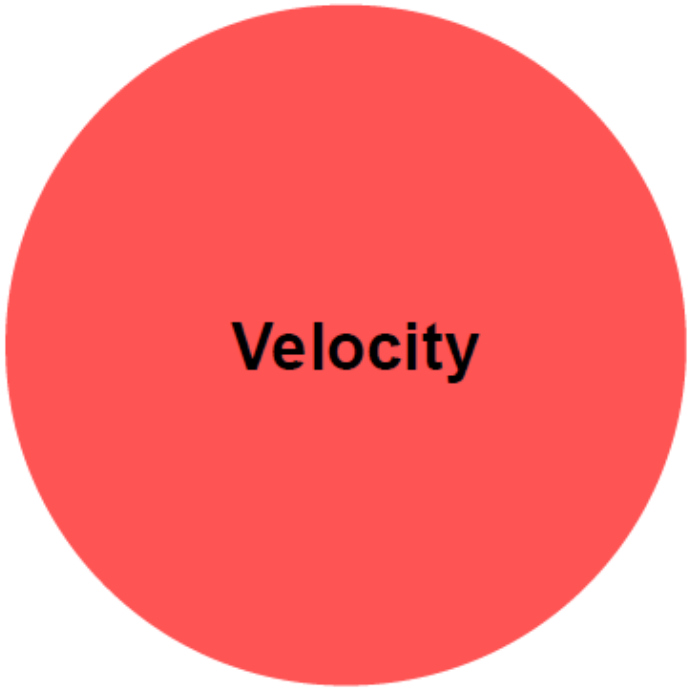
RDD

- two types of operations on RDDs:
  - *transformations* and *actions*
- transformations are lazy
  - (not computed immediately)
- the transformed RDD gets recomputed
  - when an action is run on it (default)
- however, an RDD can be *persisted* into
  - storage in memory or disk




Applications
Sparkling
H2O
IP[y]
Apache Ambari
thunder
mahout
HIVE
oozie
Spark
hadoop
kubernetes
MESOS
docker
spring
openstack
MySQL
PostgreSQL
HIVE
cassandra
APACHE HBASE
SequoiaDB
HDFS
TACHYON
kafka
mongoDB
elasticsearch.
Parquet
Environments
Data Sources

Real-time Processing – Stream Processing



- **Low latency**
- **Continuous unbounded streams of data**
- Distributed
- Parallel
- Fault-tolerant

Stream Data

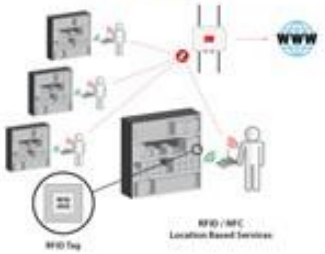
- Stream data can come from:
  - Devices
  - Sensors
  - Web sites
  - Social media feeds
  - Applications

Data Collection Devices

Smart Machinery

Phones and Tablets

Home Automation



RFID Systems

Digital Signage

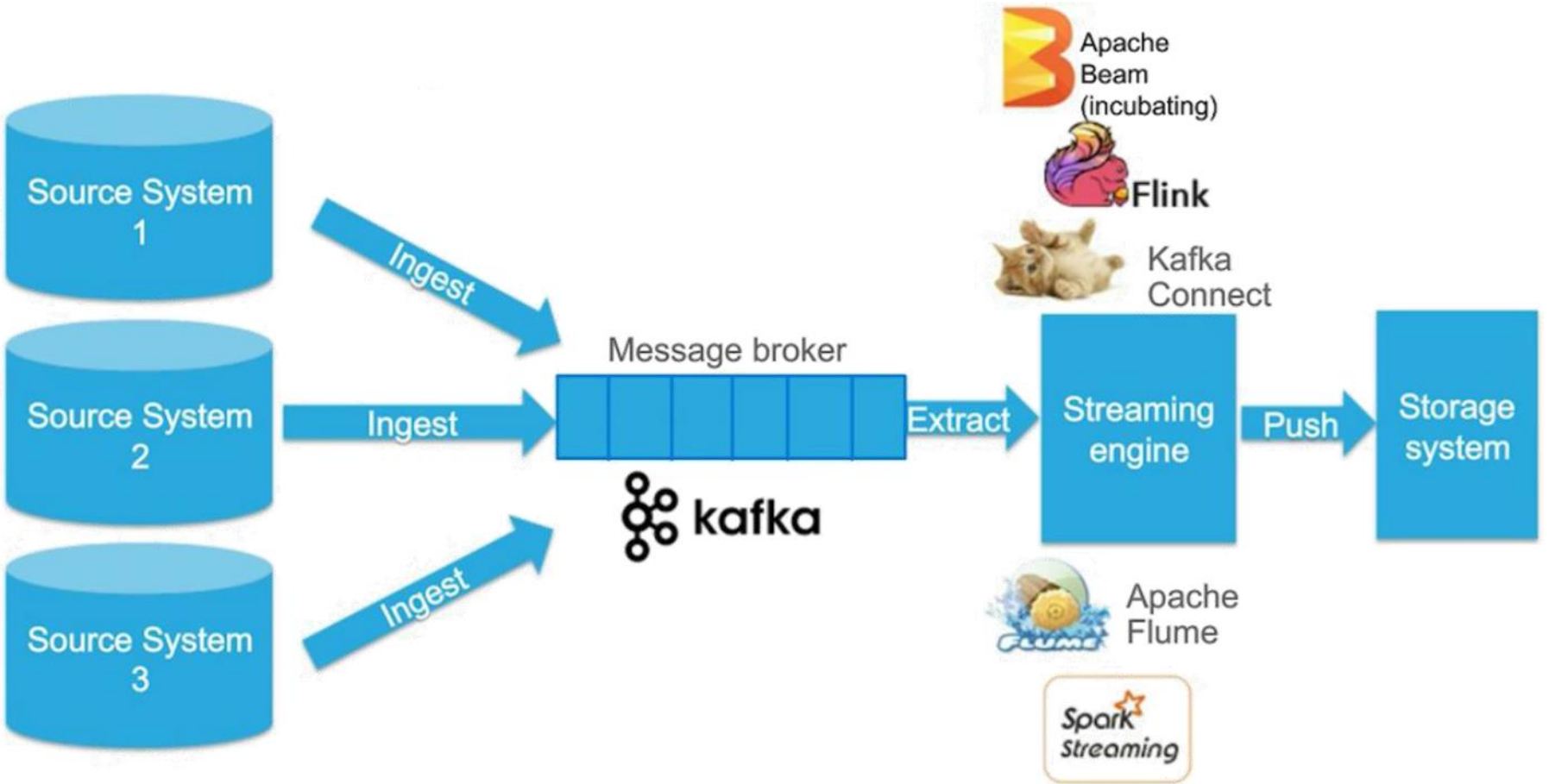
Security Systems

Medical Devices

Real-time (Stream) Processing

- Computational model and Infrastructure for continuous data processing, with the ability to produce low-latency results
  - Data collected continuously is naturally processed continuously (Event Processing or Complex Event Processing -CEP)
- Stream processing and real-time analytics are increasingly becoming where the action is in the big data space.

Real-time (Stream) Processing Arch. Pattern



Real-time (Stream) Processing

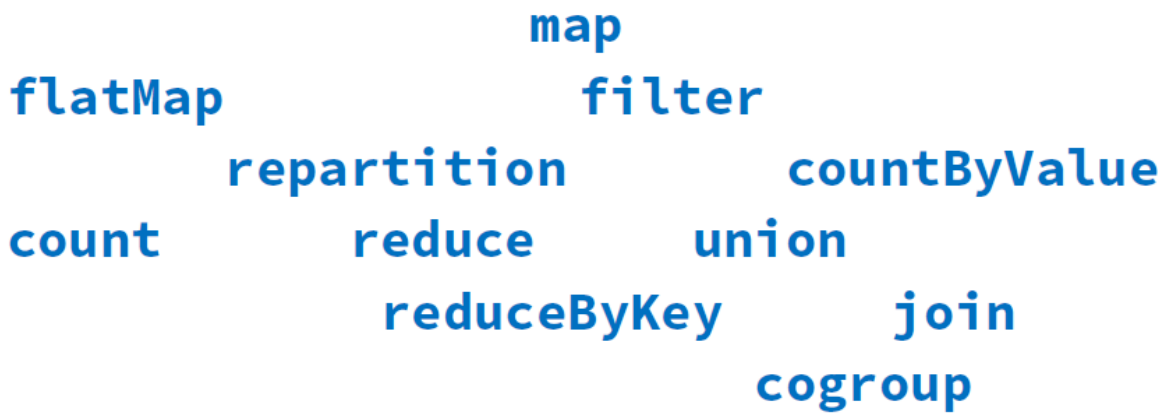
- (Event-) Stream Processing
  - A one-at-a-time processing model
  - A datum is processed as it arrives
  - Sub-second latency
  - Difficult to process state data efficiently
- Micro-Batching
  - A special case of batch processing with very small batch sizes (tiny)
  - A nice mix between batching and streaming
  - At cost of latency
  - Gives statefull computation, making windowing an easy task

Spark Streaming

- Spark streaming receives live input data streams and divides the data into batches (micro-batching)
  - Batches are then processed by the spark engine to create the final stream of data
  - Can use most RDD transformations
  - Also DataFrame/SQL and MLlib operations

Transformations on DStreams

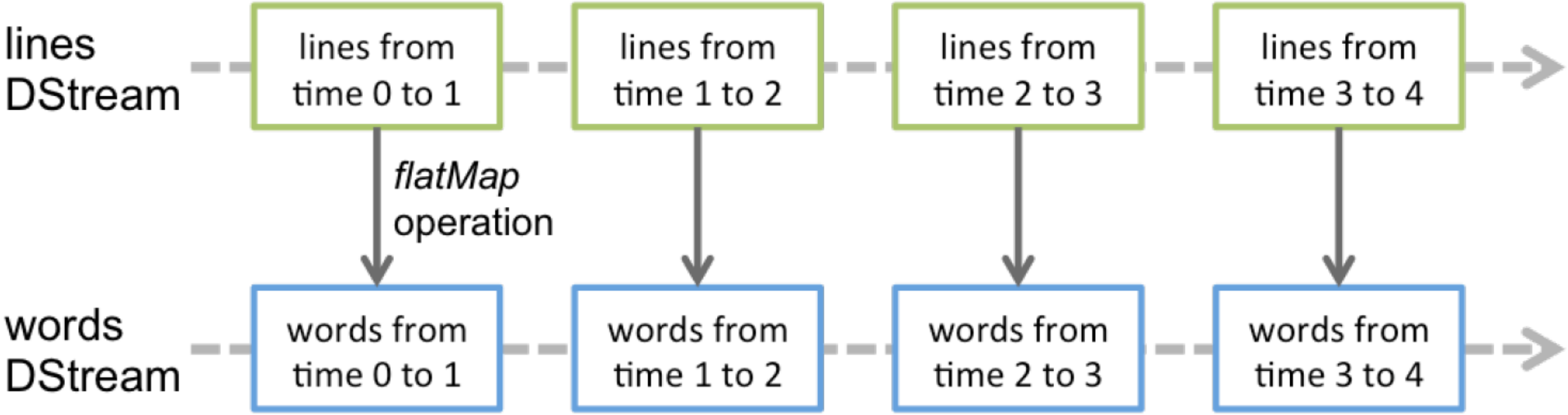
- DStreams support most of the RDD transformations



- Also introduces special transformations related to state & windows

Operations applied on DStream

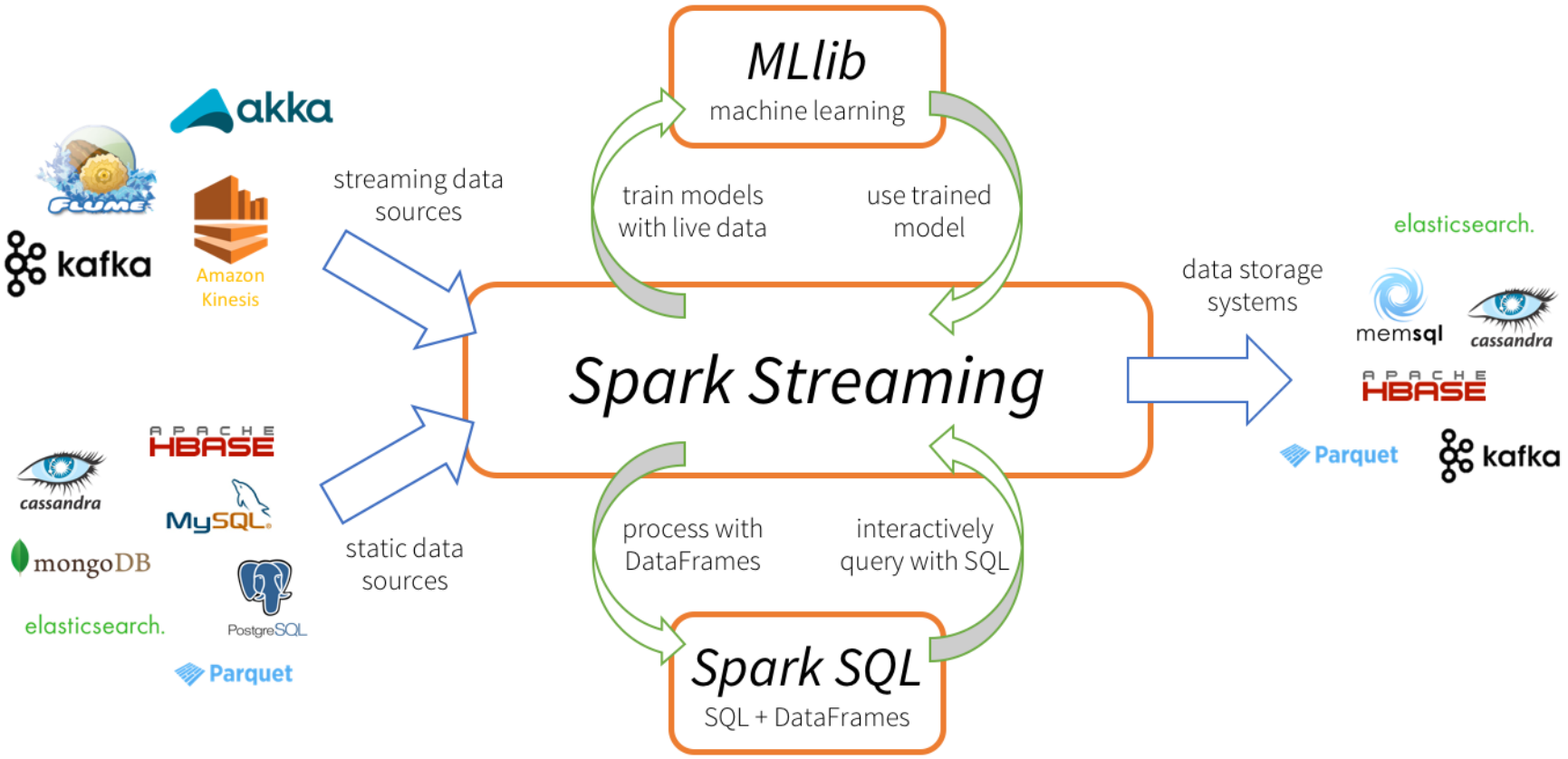
- Operations applied on a DStream are translated to operations on the underlying RDDs
- **Use Case:** Converting a stream of lines to words by applying the operation fatMap on each RDD in the “lines DStream”.



Stateless vs Stateful Operations

- By design streaming operators are stateless
  - they know nothing about any previous batches
- Stateful operations have a dependency on previous batches of data
  - continuously accumulate metadata overtime
  - data check-pointing is used for saving the generated RDDs to a reliable stage

Spark Streaming

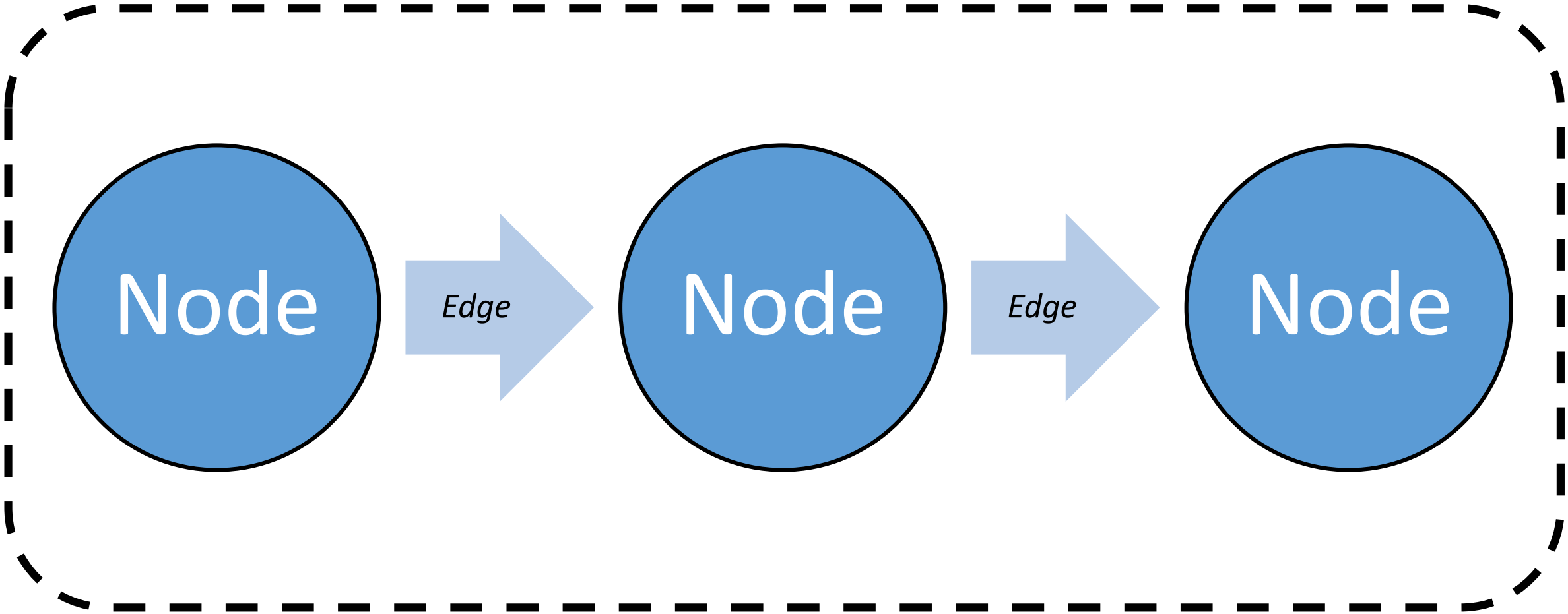


Apache Storm

Distributed, real-time computational framework, used to process unbounded streams.

- It enables the integration with messaging and persistence frameworks.
- It consumes the streams of data from different data sources.
- It process and transform the streams in different ways.

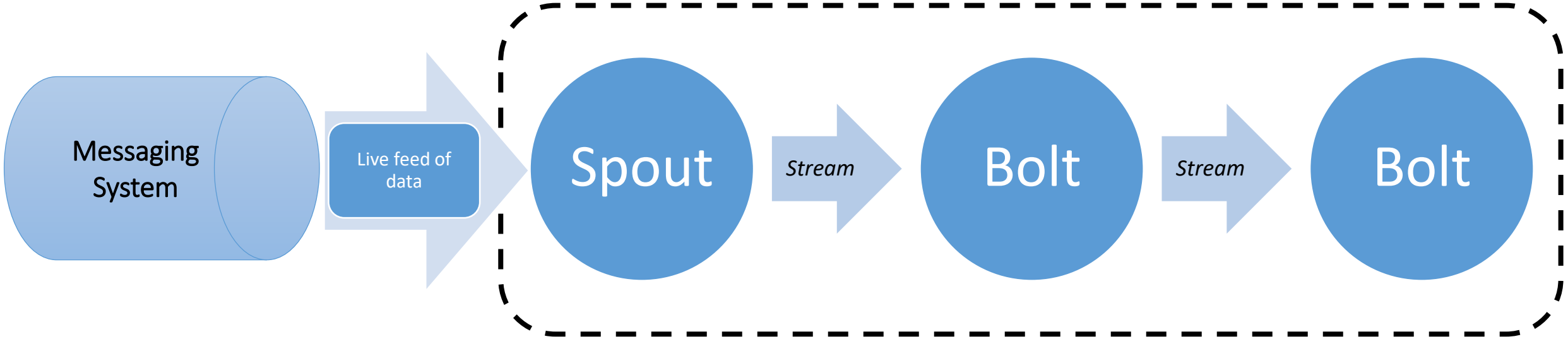
Topology



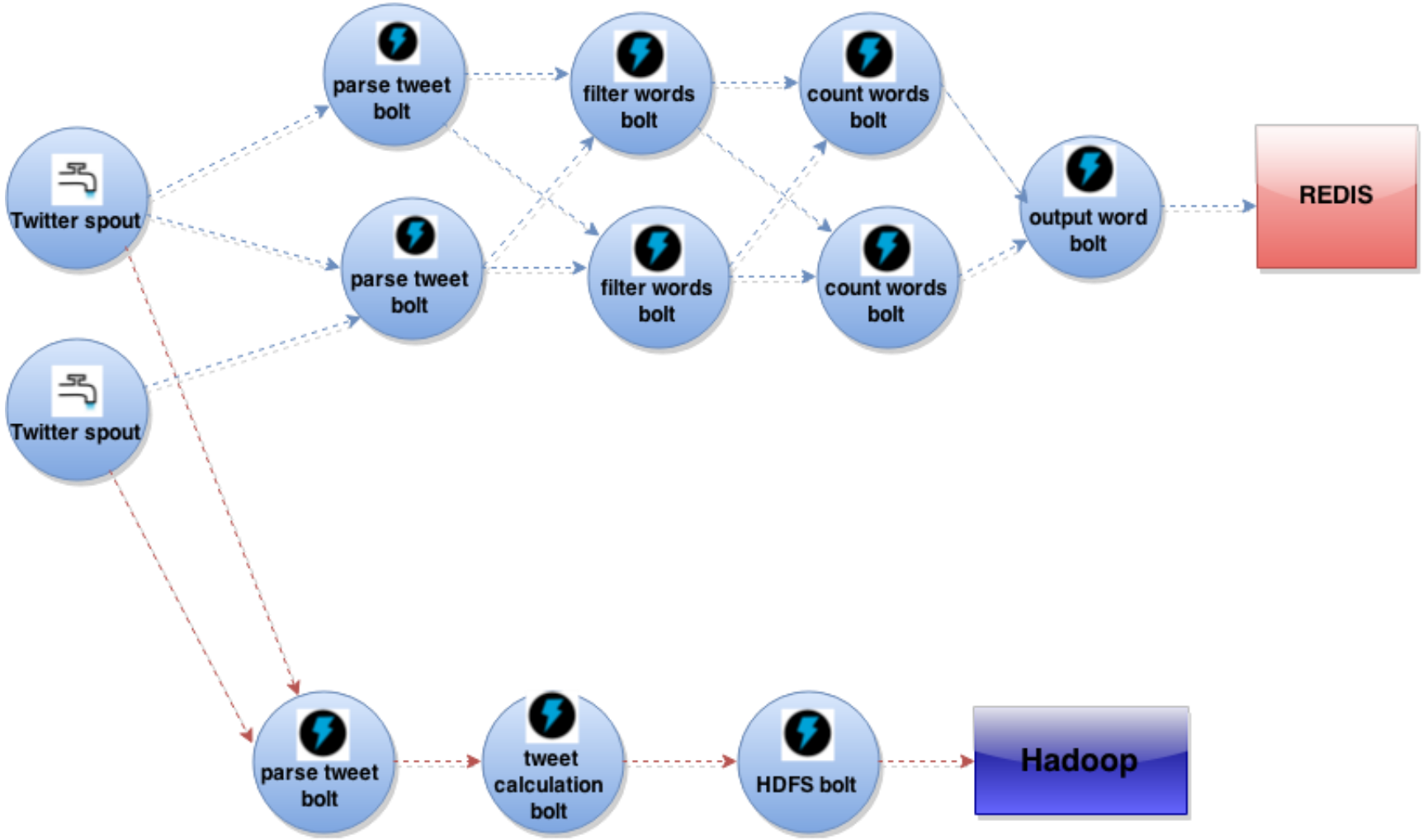
Apache Storm Concepts

- Tuple
  - Data send between nodes in form of Tuples.
- Stream
  - Unbounded sequence of Tuples between two Nodes.
- Spout
  - Source of Stream in Topology.
- Bolt
  - Computational Node, accept input stream and perform computations.

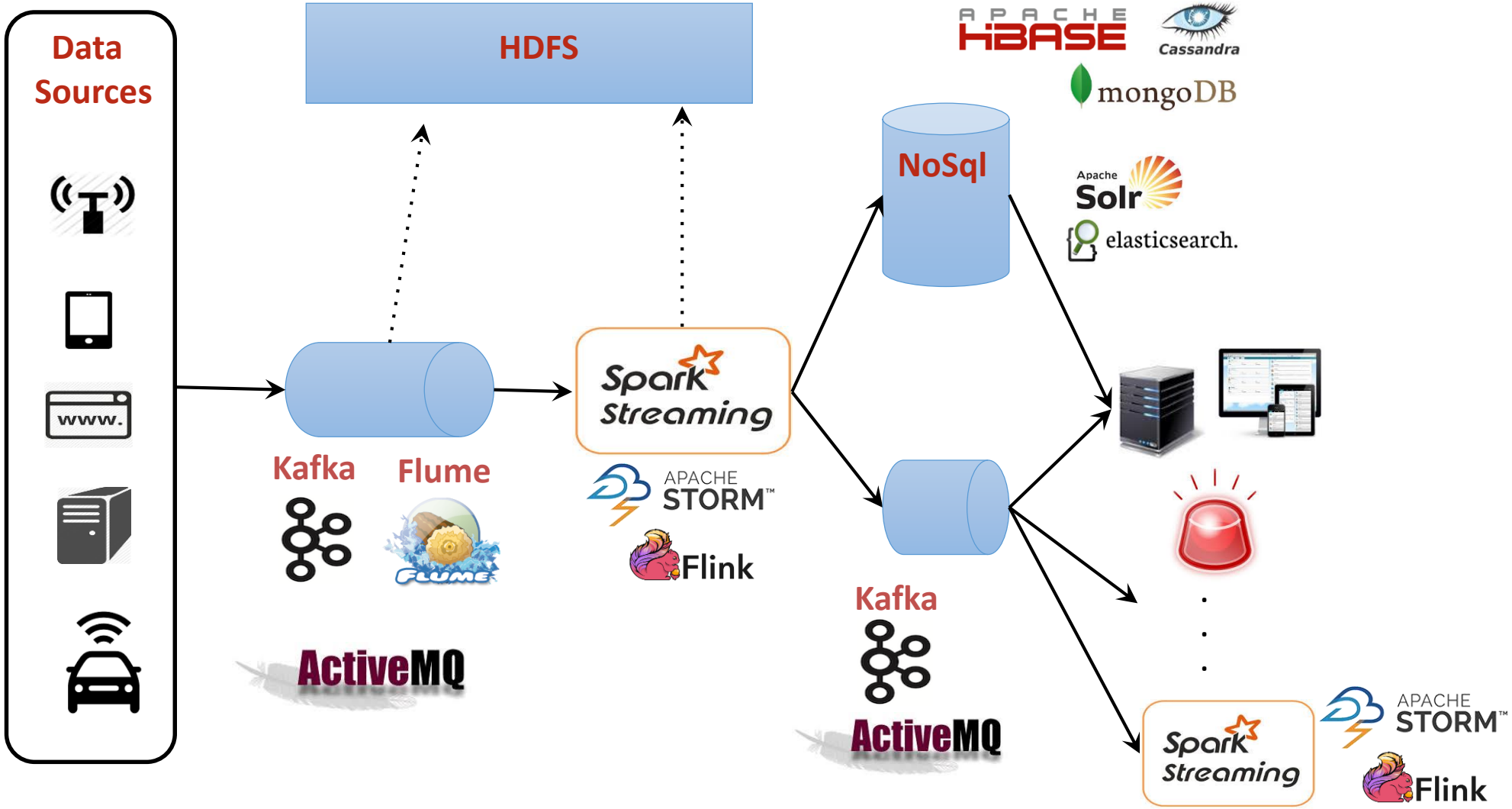
Topology


Messaging System
Live feed of data
Spout
Stream
Bolt
Stream
Bolt

Storm Word Count Topology

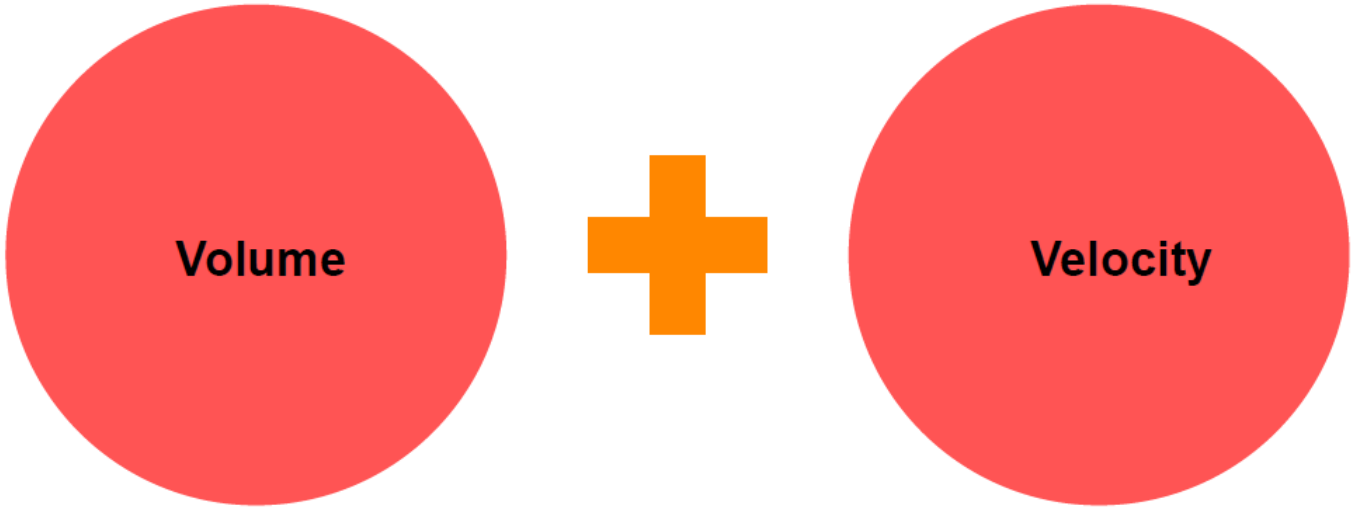


Stream Processing Architecture



Hybrid Computation Model

- **Low latency**
- **Massive** data + **Streaming** data
- Scalable
- **Combine** batch and real-time results



Hybrid Computation

- Data-processing architecture designed to handle massive quantities of data by taking advantage of both batch-and stream-processing methods.
- A system consisting of three layers: batch processing, speed (or real-time) processing, and a serving layer for responding to queries.
  - This approach to architecture attempts to balance latency, throughput, and fault-tolerance by using batch processing to provide comprehensive and accurate views of batch data, while simultaneously using real-time stream processing to provide views of online data.
  - The two view outputs may be joined before presentation.

Data Visualization Layer

Visualization and APIs

- Dashboard and applications that provides valuable bisuness insights
- Data can be made available to consumers using API, messaging queue or DB access

- Technology Stack
  - Qlik/Tableau/Spotifire
  - REST APIs
  - Kafka
  - JDBC

VahidAmiry.ir
@vahidamiry