

Interview Notes on Big Data Architecture

Core Hadoop Functionality:

The core Hadoop project consists of a

- 1) way to store data, known as the Hadoop distributed file system, or HDFS
- 2) way to process data with MapReduce.

Key Concept:

we split the data up and store it across the collection of machines (mid-range/commodity) known as a cluster. Then when we want to process the data, we process it where it's actually stored. Rather than retrieving the data from a central server, the data's already on the cluster, so we can process it in place.

You can add more machines to the cluster as the amount of data you're storing grows.

HDFS:

When a file is loaded into HDFS, it's split into chunks, which we call '**blocks**'. Each block is pretty big; the default is 64 megabytes. So, imagine we're going to store a file called 'mydata.txt', which is 150 megabytes in size. As it's uploaded to the cluster, it's split into 64 megabyte blocks, and each block will be stored on one node in the cluster. Each block is given a unique name by the system: it's actually just 'blk', then an underscore, then a large number. In this case, the file will be split into three blocks: the first will be 64 megabytes, the second will be 64 megabytes, and the third will be the remaining 22 megabytes.

Mapreduce Design Patterns

1. Filtering Patterns - Sampling, Top N
2. Summarization Patterns - Counting, MinMax, Statistics, Index
3. Structural Patterns - Combining Data Sets


Building a Large Scale Data Architecture - Application Considerations

1. Capacity
2. Performance
 - a. Latency
 - b. Bandwidth
3. Access Patterns:
 - a. Granularity Of Access
 - b. Frequency Of Access
 - c. Locality Of Access
4. Durability
5. Fault Tolerance
 - a. Reliability
 - b. Availability
 - c. Accuracy
6. Security
7. Provenance

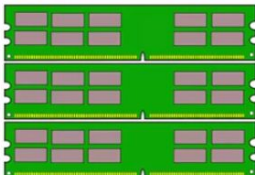
Types of Storage

☰ Caching in Web Services


Storage Devices




Registers and CPU Caches
 Kilobytes to Megabytes
 ~ \$100 per MB
 1 – 5 ns




Main Memory (DRAM)
 4 GB to 1 TB
 ~ \$10 per GB
 10 – 50 ns



Solid State Drives
 64 GB to 1TB
 ~ \$2 per GB
 50 – 150 ns



Magnetic Hard Drives
 1 TB to 6 TB
 ~ \$0.04 per GB
 5 – 20 ms



Remote Secondary Storage (Cloud/Offline)
 Virtually Infinite
 Virtually Free
 > 100 ms

⏮ ⏪ ⏩ ⏭ 3:31 / 13:48

CC HD 🔊 📶

Caching

1. Types of Storage devices and different layers of caches.
2. Question what data to store at which layer of cache. Answers boils down to locality.
3. Temporal Locality (Asking for exact same address at different times), Spatial Locality (On the space dimension - copying a block to cpu). Caching at Client side Technologies to pre-fetch user data
4. Web server cache.

Design Considerations in Local File Systems

1. Think about storage medium - Disk, Magnetic tapes reduce Seek, Rotational times.
2. Maximize amount of useful data transferred.

Design Considerations to build a Distributed File System

Distributed file system, the client views a single, global namespace that encompasses all the files across all the file system servers.

Design considerations:

Fault tolerance

Replication - overcomes fault tolerance

Consistency - created because of replication

File-sharing semantics

- a. **UNIX semantics: Immediate consistency**
- b. **Session semantics:** changes to an open file are initially visible only to the process that modified the file. Once the file is closed, the changes are made visible to other processes. Session semantics relaxes the strict requirements employed by UNIX semantics, but the question of conflict handling emerges: When two clients are simultaneously editing the same file, whose session is honored? Some approaches honor only the last client to close the file, while others might even be unspecified.
- c. **Immutable semantics:** Versioning of files. Followed in GFS
- d. **Atomic transactions:** Ensure Atomicity

Design Considerations to build a Database

1. Schema vs Flexible Schema/Schemaless architecture
2. Transactional Data Vs Operational Data

OLTP vs. OLAP

	OLTP	OLAP
Source of Data	Operational Data from source (Transactions)	Data consolidated from OLTP, logs or other databases
Purpose of Data	Fundamental business tasks	Helps with planning, management and decision support
Insert and Updates	Short and fast inserts as part of transactional queries, often in real-time	Long running batch jobs to ingest data from various sources. Run periodically.
Queries	Relatively simple queries that return few records	Complex queries that may require data to be joined and aggregated from multiple tables
Speed of Queries	Typically very fast, number of transactions per second is a typical performance metric	Typically slower than OLTP, dependent on the query length and complexity
Design of Databases	Highly normalized, consists of multiple tables	Fewer tables, use of star-style schemas

8:46 / 10:33

CC HD

Scaling Databases

Motivation

1. Performance Bottleneck
2. Capacity Bottlenecks

Types

1. Vertical Scaling
2. Horizontal Scaling
 - a. Replication - Consistency problems
 - b. Sharding - Data Partitioned between individual Database servers - Horizontal partitioning - Split rows. Vertical Partitioning - Split columns and assign to different servers.

Consistency Formula

$$R + W > N$$

N = Number of Database agents used for replication.
W = Number of Database agents that agree on update
R = Number of Database agents that agree on a read

Distributed Concurrency Control

(Important when We scale out/ Scale up the database)

TimeStamp adherence

Strict - >Global Timestamp - Slowest ,Strong Consistency - Logical Timestamp - less slower - linearized, Sequential Consistency - Logical Timestamp - faster than Strong Consistency, Causal Consistency - unordered - vector clock , Eventual consistency - no locking

<http://www.bailis.org/blog/linearizability-versus-serializability/>

Two Phase locking = Coordinator - Global Commit , Global Abort, Participant - Local Commit, Local Abort

Two Phase locking affects performance and increases Latency.

NoSQL Databases

CAP Theorem - > Cannot guarantee Consistency and Availability (both) in Distributed Databases when tolerant to network partitions.

BASE -> Basically Available, Soft state and Eventual Consistency.

Example Case => E-Commerce Stock Availability

The typical characteristics of NoSQL databases include

- No strict schema requirements
- No strict adherence to ACID properties for transactions
- Consistency traded in favor of availability

The tradeoff with NoSQL databases is between ACID properties (most notably consistency) and performance and scalability.

Types of NoSQL Databases

A limited taxonomy of NoSQL databases is illustrated in Figure 4.20.

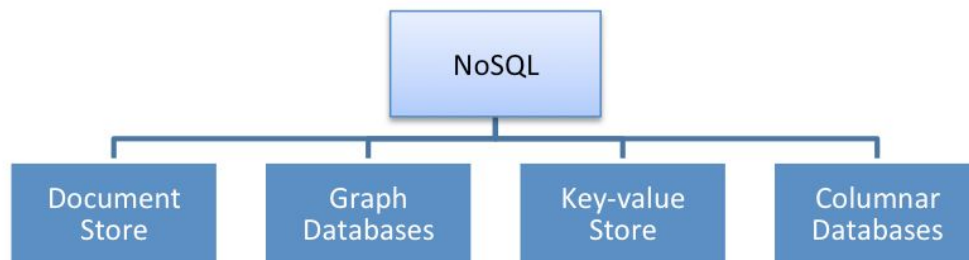


Figure 4.20: A taxonomy of NoSQL databases

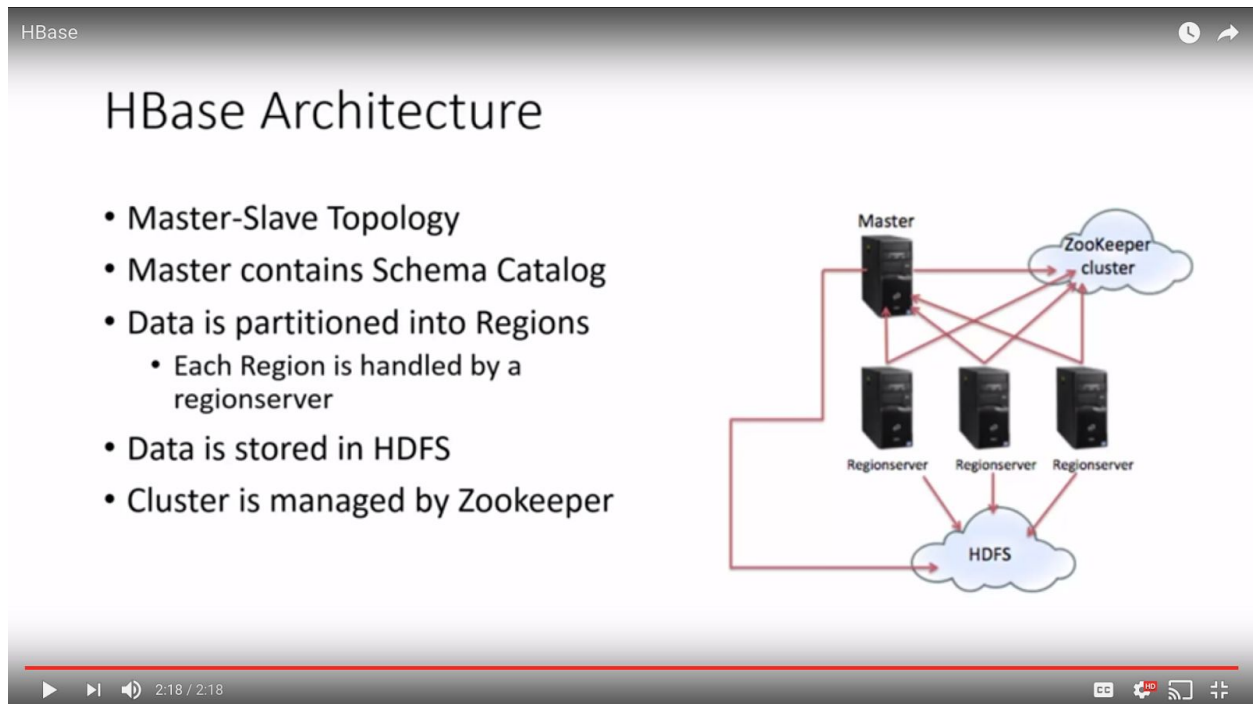
Document Stores

In contrast to RDBMSs, where data is stored as records with fixed-length fields, document stores store a document in

HBase Architecture

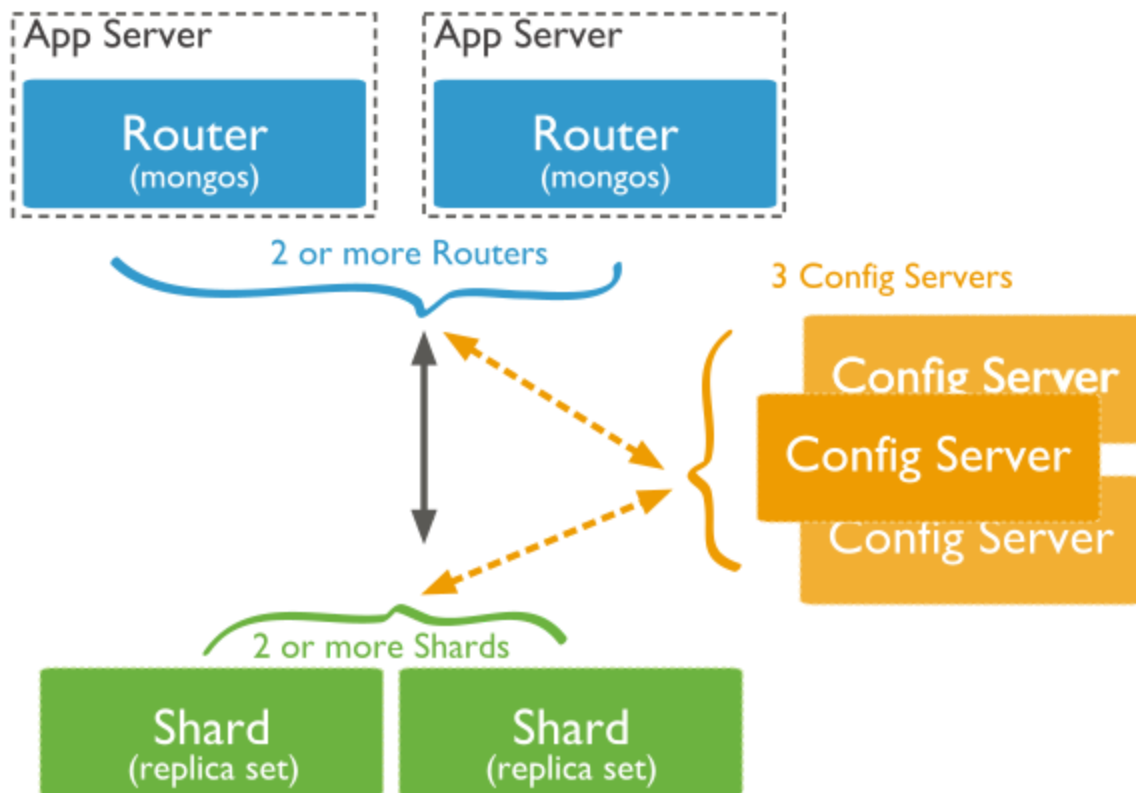
1. Distributed Key - Column Family values
2. Keys are unique
3. Follows Master slave model.
4. Master contains schema Catalog. Decided by zookeeper, a distributed coordination service. Master handles such functions as region allocation, failover, and load balancing.
5. Slaves are region servers. They have independent HDFS servers or they may be connected to remote HDFS servers.

6. To distribute data stored in HBase across the nodes in a cluster, HBase automatically partitions (shards) tables into regions, which are groups of consecutive rows in a table.
7. Client -> Identifies Root Table (Via zookeeper) -> identifies Meta Table which is partitioned -> Meta table returns the Region where IO has to be performed
8. HBase does not guarantee ACID Properties on Scan operation
9. Best use case, Web Analytics. Does not support joins. Supports versioning



MongoDB

1. Based on Document Model, Precisely Document Driven Design. It's a Database of a group of collections.
2. Basic operations: Find, insert, remove, update
3. Architecture: Two types of deployments
 - a. Single Node - Default Mode
 - b. Multiple Node - To Scale
 - i. Sharding
 - ii. Replication



Architecture of MultiNode MongoDB Deployment contains

1. Routers - (Mongos) - Routes queries to individual mongoDB servers
2. Many mongoDb servers forms a cluster whose configuration are saved in a config cluster
3. The DbServer can be sliced to multiple shards who gets called by Routers/ Notified by Config servers to access/compute data
4. Replication - > Shards themselves have replicas, a primary shard and a set of secondary shard. If primary shard fails, the remaining set of secondary shards conduct a vote to determine new primary shard.
5. Data is sent over to a particular shard based on a hash key =. Pointing to a chunk in a group of shard or range index.

Useases:

1. Large amounts of evolving, Flexible Data with ever changing schema
2. Very good for location based indexing [The Weather channel](#) (cook county - Emergency alert)

“The apps typically handle two million requests per minute, including weather data and social sign-ins. As the user base scales, so will MongoDB. With its native scale-out capabilities, MongoDB can support thousands of nodes, petabytes of data and hundreds of thousands of ops per second.”

3. Applications with high write load and insert rate (they have bulk inserts with flexible transactions)

Apache Cassandra

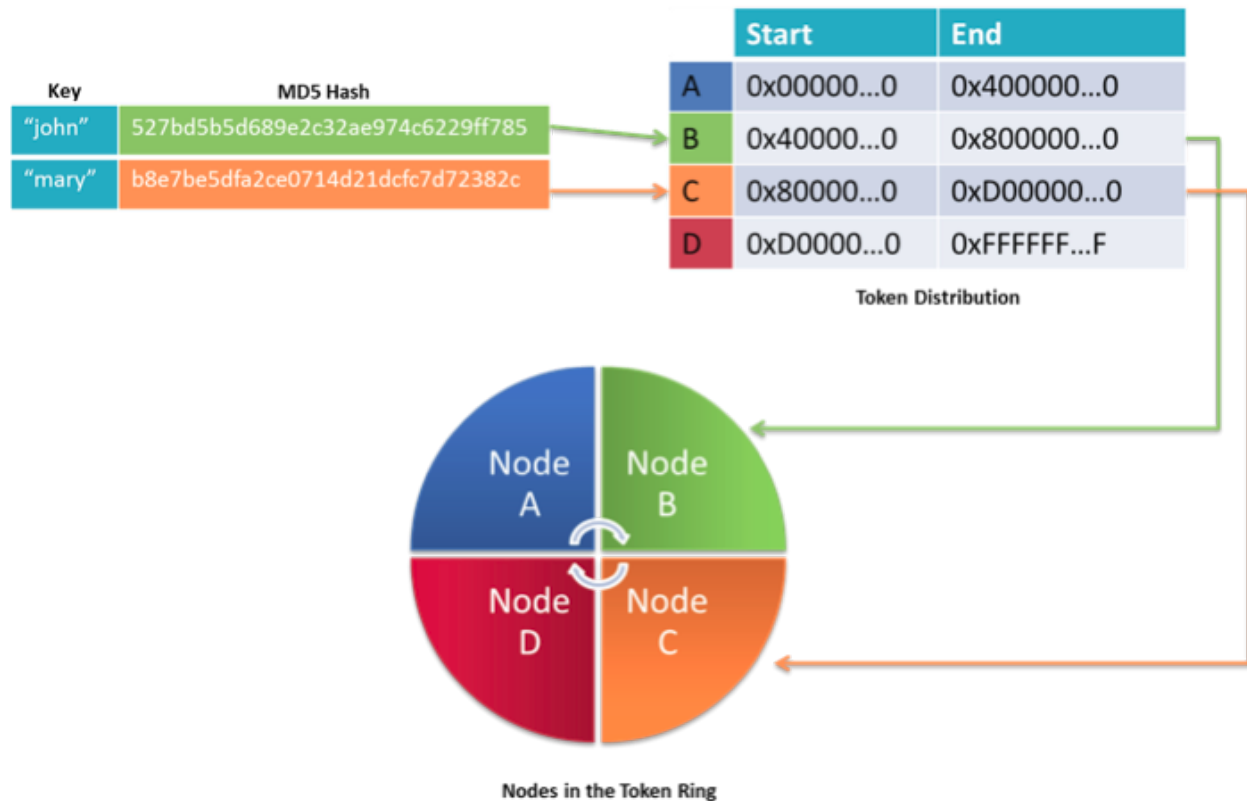
Fully Distributed Datastore. Motivation from Google Big Table (Replication part) and Amazon Dynamo (Performance Tuning on per operation basis). Shards Data using Consistent Hashing.
(Number of cluster nodes that agree upon read + Number of cluster nodes that agree upon write > (always) Number of clusters that are used for replication)

Data Model

1. Similar to HBase Key Value (As a Column Family => Super columns (nested data as in multiple sub-columns, columns))
2. Operations => Get, insert Delete (Can perform these operations on individual rows, Columns, column family, Slices (a complete column/column family) and range (a group of rows))

Architecture

1. Fully distributed Architecture, No Single point of failure
2. Node communicates each other to update the cluster state and distribute data
3. Data Distributed using Consistent hashing



Consistency for Read or Write operation can be Tuneable

For Read - One, Quorum (Vote), All

For Write - Zero (Write without confirmation), Any, One, Quorum, All

Failure Detection:

1. Special Gossip protocol to communicate the failure of a node . No Master. Accrual Failure Detection.

Hinted handoff:

1. A write operation has been sent to a node by another to perform the operation in the destination location.
2. But that location is not reachable anymore
3. So the sender would take a hint info (Like a proxy person attending your call)

Hint Like

"I have this write information that is intended for node X. I am going to hold on to this write, and when I notice node X has come back online, I will send the write request to node X."

Acid Properties:

Atomic - Achieved

Durable - Achieved

Consistency - Depends on levels specified. Each level has its own tradeoff in consistency and performance

Isolation - Achieved

Use-cases

Facebook - Writes are frequent and reads are less predictable

out-of-the-box support for replicating across multiple data centers (Property from DynamoDB)

Schema Free Data model

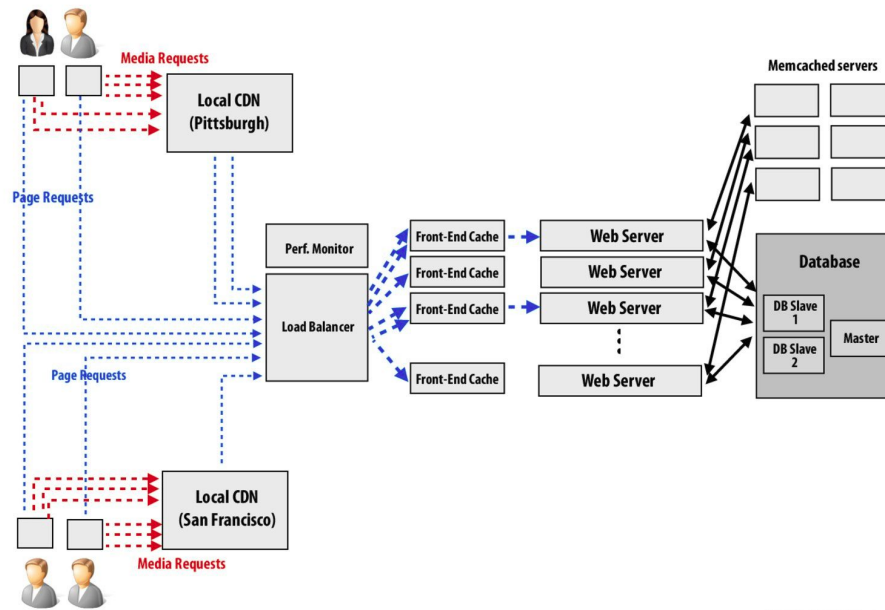
Dynamic Fault Tolerant Mechanisms (apparently better than Hadoop/HBase)

Scaling websites

Principles to Scale a website

Design Considerations on Performance.

CDN integration

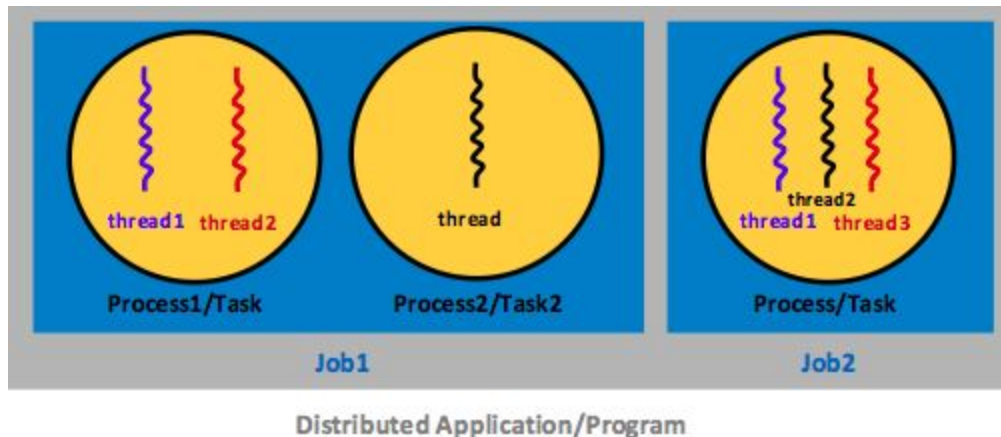


CMU 15-418/618, Spring 2017

Think about

1. Web server Auto Scaling
2. Load Balancing
3. Distributing Binary Objects in CDN
4. MemCaching
5. Front end caching
6. Data Replication

MultiProcessing - MultiTasking - Thread - Job - Application



Design Considerations of a Cloud program:

1. programming model message passing vs shared memory?
2. synchronous or asynchronous computation model?
3. What is the best way to configure data for computational efficiency: by using data parallelism or graph parallelism?
4. Architecture Style: Master-Slave vs Peer to Peer?

Problems:

1. Scaling Cloud applications are hard - Load Imbalance, Synchronization is hard, Lots of communication overheads.
2. Heterogeneity in Computational resources (Mac/ Windows/Linux commodity devices installed with different configs) makes scheduling hard. Creates Hardware/Software differences among nodes.
3. Failure likelihood increase in cloud scale. Robust Fault Tolerance Mechanisms needed.

Shared memory vs Message passing

Shared Memory : Programming models use Distributed Shared memory. Tasks, Semaphore (point-to-point synchronization mechanism that involves two parallel/distributed tasks, operations - post & wait), Critical section (only one task can access a critical section at a time), Locks.

Message Passing: Distributed tasks communicate by sending and receiving messages. Tasks do not share address space. Incurs Communication overheads. Tasks are split into sub-tasks and send their final outputs to a main task via a Message passing interface.

Why Spark Transformations are lazy?

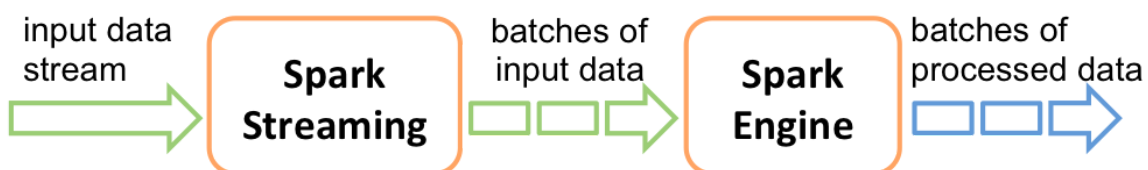
Transformations on data frames are lazily evaluated, meaning that Spark will not begin to execute until it sees an action. So, when we call `sc.textFile()`, the data is not loaded until it is necessary. As with transformations, the operation (in this case, reading the data) can occur multiple times.

example

```
lines = sc.textFile("README.md") #transformation  
pythonLines = lines.filter(lambda line: "Python" in line) #transformation  
pythonLines.first() #action
```

we defined a text file and then filtered the lines that include Python. If Spark were to load and store all the lines in the file as soon as we wrote `lines=sc.textFile(...)`, it would waste a lot of storage space, given that we then immediately filter out many lines. Instead, once Spark sees the whole chain of transformations, it can compute just the data needed for its result. In fact, for the `first()` action, Spark scans the file only until it finds the first matching line; it doesn't even read the whole file.

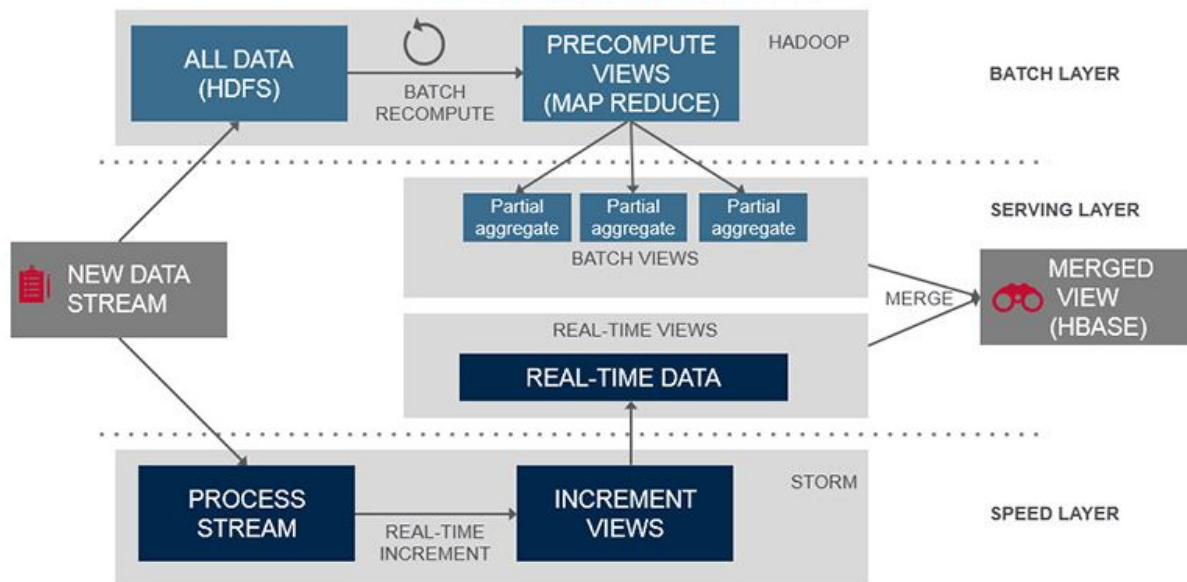
Spark Streaming



Lambda Architecture

Why dual systems? (Batch and Stream)

Lambda Architecture



Sorting/ Linked List / Binary Search 1 hour

Resume Review - 2 hours

Revision Notes - 2 Hours

<https://www.oreilly.com/ideas/questioning-the-lambda-architecture>

[Big O cheat sheet](#)

[Hive vs HBase Comparison](#)