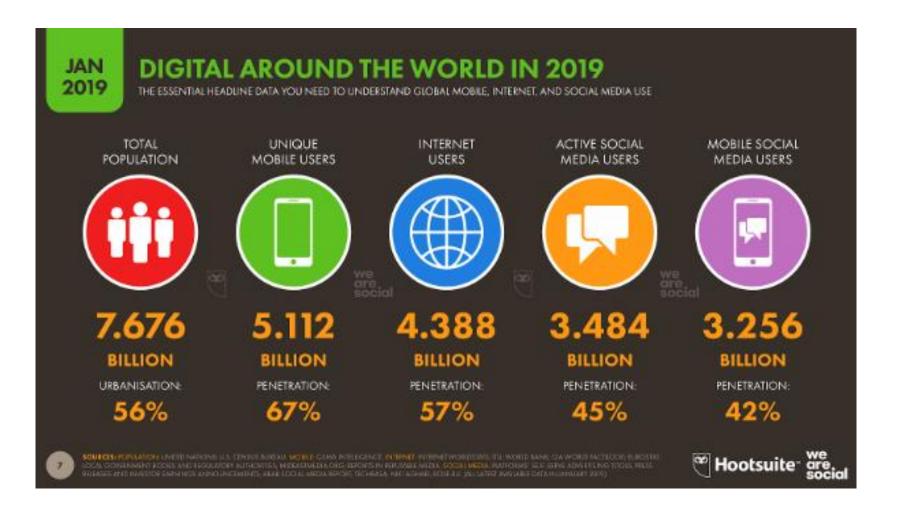
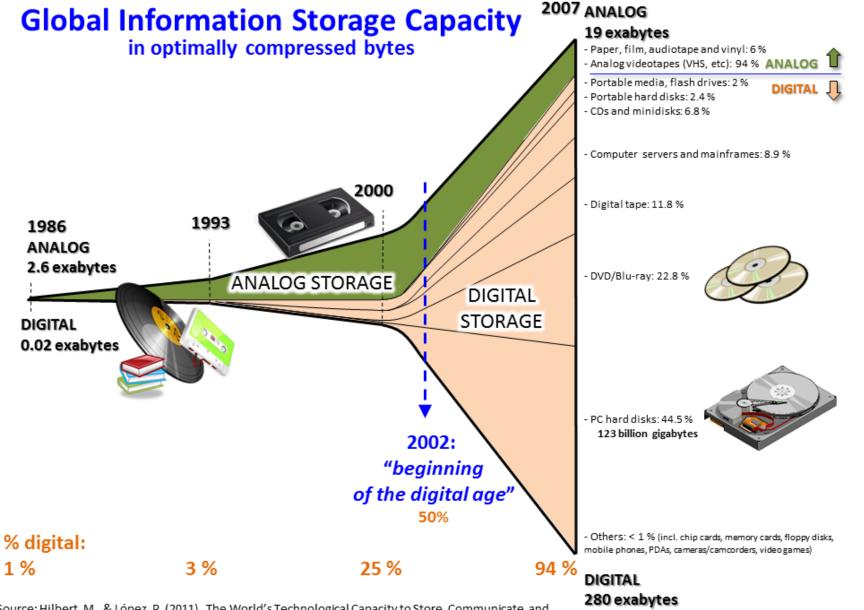
# Introduction to Data Systems & Data Architectures

**Bachar Wehbi** 

bachwehbi@gmail.com

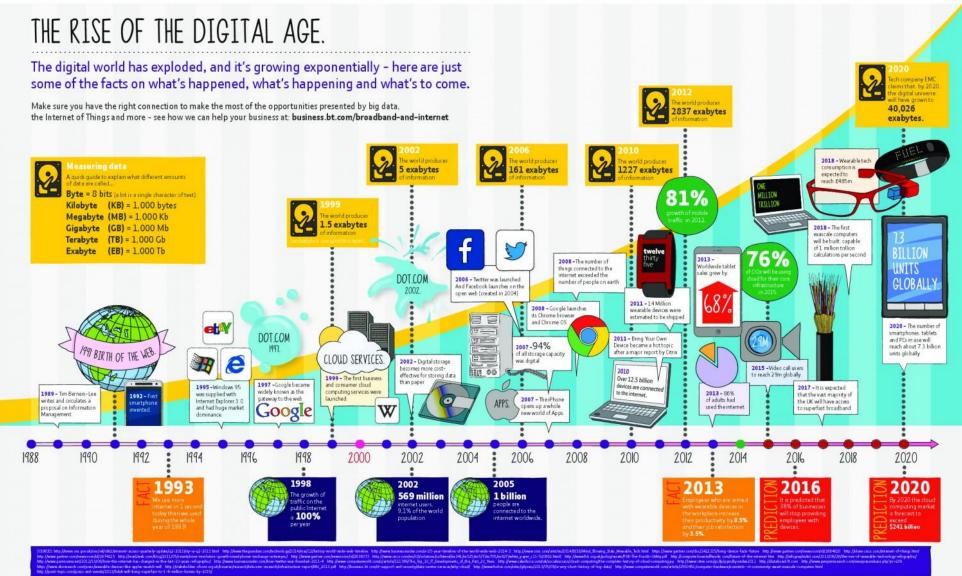
## We live in a global digital age





Source: Hilbert, M., & López, P. (2011). The World's Technological Capacity to Store, Communicate, and Compute Information. *Science*, 332(6025), 60 –65. <a href="http://www.martinhilbert.net/WorldInfoCapacity.html">http://www.martinhilbert.net/WorldInfoCapacity.html</a>
Bachar Wehbi





## (Big) Data Systems

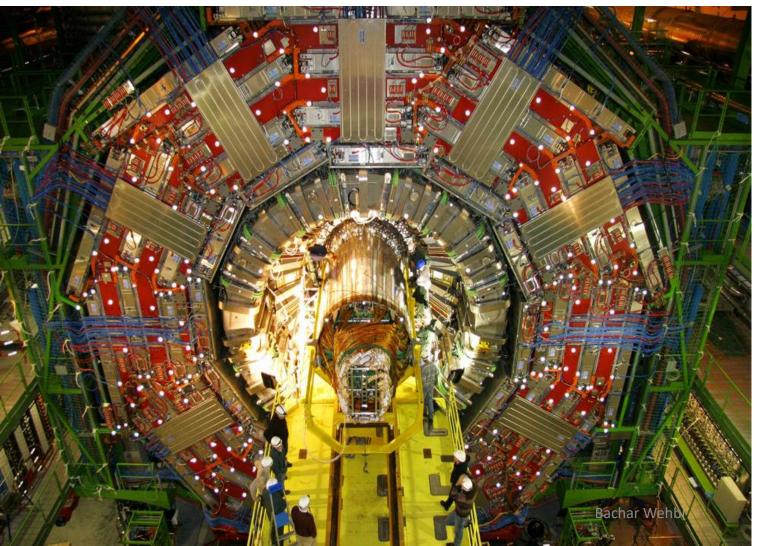
## What is Big Data

Big data is **data sets** that are so **voluminous and complex** that traditional data-processing application software are inadequate to deal with them.

Big data challenges include **capturing data**, **data storage**, **data analysis**, **search**, **sharing**, transfer, visualization, querying, updating, information privacy and data source.

There are five concepts associated with big data: volume, variety, velocity and, the recently added, veracity and value

## Big Volume



Cern's Large Hydron Collider (LHC)

Particles collide with sensors 1 Billion times per second

Generate 1 PB of raw data per second

7.5 million seconds of experiments in 2016

49 PB of cleaned experiments data in 2016

## Big Volume

#### Imagine The size of a cluster with these specs!

#### Nimbus 100TB SSD Disk!



AWS EC2: u-12tb1.metal

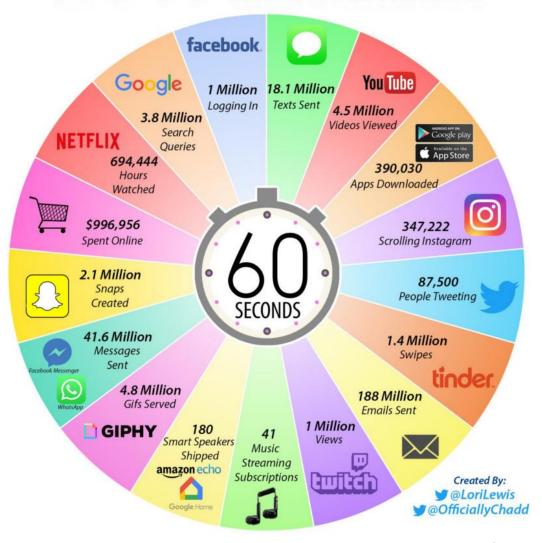


## Big Velocity

 Velocity: the speed at which the data is generated and processed.

 Big data is often available in real-time.

## 2019 This Is What Happens In An Internet Minute



## Big Velocity

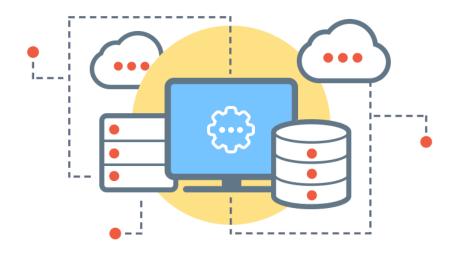


"From the dawn of civilization to 2003, five exabytes of data were created. The same amount was created in the last two days."

(Eric Schmidt - 2010)

## Big Variety

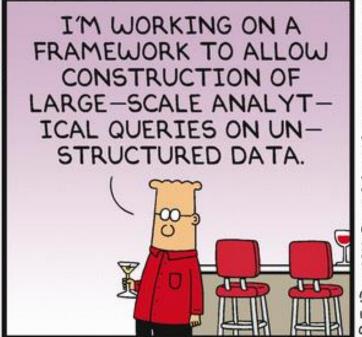
- Variety: The type and nature of the data.
- Big data draws from text, images, audio, video.
- Data sources are both internal and external to the organization.
- Data is structured, semi-structured and unstructured.

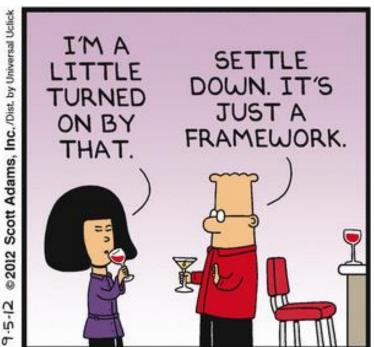




## Big Variety

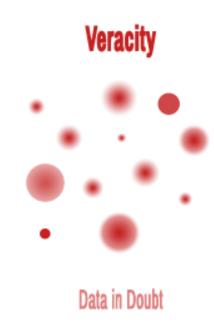






## Big Veracity

- **Veracity:** refers to the quality of trustworthiness of the data.
- Big Data gives you the opportunity to analyse all of the data, but on the other hand the data contains lots of noise.
- Company has to be able to transform the data into trustworthy insight and discard noise.
- Example of untrustworthy noise "fake news", Spam, Bots, etc.



## Big Value

#### **Datafication**

- CRM
- Sales
- Products
- ERP
- Email
- Social
- Video
- Audio
- Click Stream

Volume

Velocity

Variety

Veracity

#### **Analysing Data**

- Reporting
- Text mining
- Search & Indexing
- Predictive Analytics
- Recommendation
- Sentiment Analysis
- Object recognition
- Artificial Intelligence

**Value** 



## Properties of Modern Data Systems

## Data Systems: Properties

#### Reliability

The system should continue to work correctly even in the presence of faults & errors

#### Scalability

 As the system grows there should be a reasonable and affordable ways of dealing with this growth

#### Maintainability

 When different people work on the system over time, they should all be able to work productively

## Data Systems: Reliability

- Reliable systems can tolerate and cope with faults. These systems are called fault tolerant or resilient.
- Faults have different natures:
  - Hardware faults: when a HW dies (Hard disk, RAM, ...) we can:
    - Use redundancy: i.e. RAID 5
    - Software fault-tolerance in distributed systems.
    - Usually random and independent from each other
  - Software Errors: like software bugs, unresponsive dependency, cascading failures
    - Difficult to prevent. Testing, process isolation, measuring and monitoring.
  - Human errors: Humans are known to be unreliable ©
    - A study has shown that configuration errors by operators are the leading cause of outages.
       HW faults (servers & network) responsible for only 10 25% of outages.

## Data Systems: Scalability

- Scalability isn't one-dimentional either a system scales or not!
- Rather it is: if a system grows in a particular way, what are the options for coping with that growth.
- OR, how can we add resources to handle the additional load
- Scalability is all about coping with load. But what is load?
  - Requests per second, rows per second, volume per second
  - Read/write ratio in a database
  - Number of simultaneous connections/requests
  - Average, peak, percentile

## Data Systems: Scalability

- In order to understand how the system behaves when the load changes, we need to understand the system performance
  - When you increase the load and keep the resources unchanged, how is the performance affected
  - When you increase the load, how much you need to increase the resources to keep the same performance
- Performance is measured using:
  - throughput (volume/sec), latency, response time
  - Measured in median values and percentiles (more important)
- How to cope with load?
  - Scale up (vertical scaling): Use high end machine. A system that can be run on the same machine is usually simpler.
  - Scale out (horizontal scaling): known as share nothing architecture. Use a distributed system to cope with load.
  - Both: use a few high end machines: simpler than using a large number of low end machines.

## Data Systems: Scalability

- Example:
  - System A: 1,000,000 requests/sec with 1KB per request
  - System B: 10 requests/min with 6GB per request

Both systems have the same throughput: 1GBps but are fundamentally different

## Data Systems: Maintainability

- The major cost of a system is not the initial development
- Rather, the debug and maintenance phases are those that cost the most

#### Dev cost < debug cost < maintenance cost

- Operability: make it easy for operations teams to keep the system running smoothly
- Simplicity: Make it easy for new devs to understand the system by avoiding unnecessary complexity
- Evolvability or extensibility: make it easy to make changes in the future.
- Recommendations for good maintainability:
  - Continuous monitoring of the system
  - Tracking the root cause of issues
  - Keeping systems up-to-data including security patches
  - Use abstractions to hide the complexity whenever possible
  - Precise and up-to-date documentation

- Twitter was born in 2006. It has two main operations:
- Post a tweet:
  - user publishes a message to their followers.
  - Average 4.6K requests/sec (2012), 7.5K requests/sec (2016). 12K requests/sec in peak time (2012)
- Home timeline:
  - A user can view tweets posted by the accounts they follow.
  - 300K requests/sec (2012)
- Discussion around designing a Data system for Twitter.

- Handling tens of thousands of writes per seconds (posting tweets) is fairly easy. This is not the challenge!
- The challenge is due to the fan-out:
  - Each user follows many accounts and is followed by many accounts.
  - Constructing the user timeline is equivalent to reading the tweets from the accounts the user follows sorted by time.

- Strategy 1: Use a regular relational database
- Tweets, users and follows tables.
- Post a tweet:

```
INSERT INTO tweets (sender_id, text)
     VALUES (current_user, curent_tweet)
```

 Read timeline: find the accounts the user follows and find he tweets for each of those accounts then merge them while sorting by time:

```
Select tweets.*, users.* FROM tweets

JOIN users ON tweets.sender_id = users.id

JOIN follows ON follows.followee_id = users.id

WHERE follows.followee_id = current_user
```

- Strategy 2: Maintain a cache for each user's timeline
- When a user posts a tweet, check all of its followers and insert the tweet to their timelines.
- Reading the timeline is then very straight forward.
- The downside is that posting a tweet now requires a lot of writes (writing to followers caches)
  - Average 345K writes/sec (2012)
  - Some accounts (celebrities) have up to 30million followers!!!
    - Twitter tries to deliver tweets to followers within 5 seconds

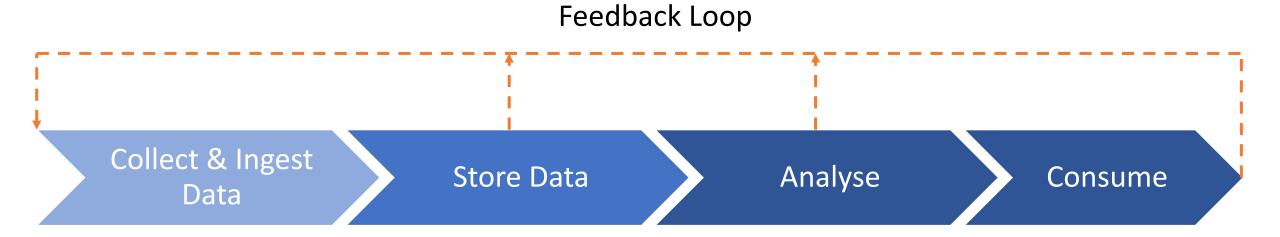
- Strategy 3: Hybrid approach
- Posted tweets continue to be fanned out to followers timelines for most accounts
- Accounts with a large number of followers (celebrities) are excepted from this fan-out
- Tweets from celebrities are fetched separately.
- Reading the timeline is now:
  - Read from account timeline
  - Read from celebrities the account follows
  - Merge both to construct the final home timeline

## Components of a (BIG) Data Platform

## Big Data: Approach

Iterative Process

Continuous Improvement



## Components of Data Platform

## Collect & Ingest Data

#### Store Data

#### Analyse

#### Consume

- Load data from source to destination
- Batch ingestion for mass of data
- Streaming ingestion for real-time data

- Massively scalable
- Structured and non structured data
- Varying performance objectives (bulk/random read write)

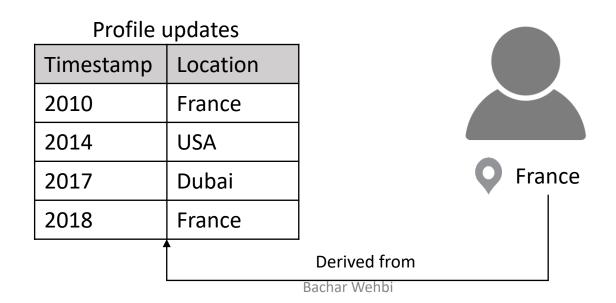
- Distributed and massively scalable
- Batch processing for large amounts of data (periodic)
- Streaming processing for realtime or close to real time

- Querying data in SQL or NoSQL stores
- Feeds reporting and business intelligence tools
- Search data

## What is Data?

#### Data

- Data is more than Information
- Not all information is equal
  - Some information is derived from other pieces of information



#### Data

- An indivisible unit that holds to be true simply because it exists
  - Time based: fact known to be true at some moment of time
  - Inherently immutable: no such thing as update or delete!!!

Profile updates			
Timestamp	Location		
2010	France	Still true We	\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
2014	USA		We are now in 2018
2017	Dubai		2010
2018	France		

In its raw form, Data can only be created and read This simplifies dramatically the CRUD model used in RDB

#### Data

- Data is therefore timestamped events
- In the past, events used to change the master data
  - CRUD model in relational databases
- In Data Architecture: events are the master data, the ground truth



## Data System

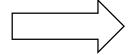
### Query = Function(All Data)

Simple definition that includes every usage of data from joins, aggregations to advanced analytics and machine learning

## Data System

#### Number of persons living in each country

Timestamp	Name	Location
2010	Sara	France
2014	Bachar	USA
2016	Sara	USA
2016	Nadine	France
2017	Nadine	Emirates
2018	Bachar	France



Country	Count
France	1
USA	1
Emirates	1

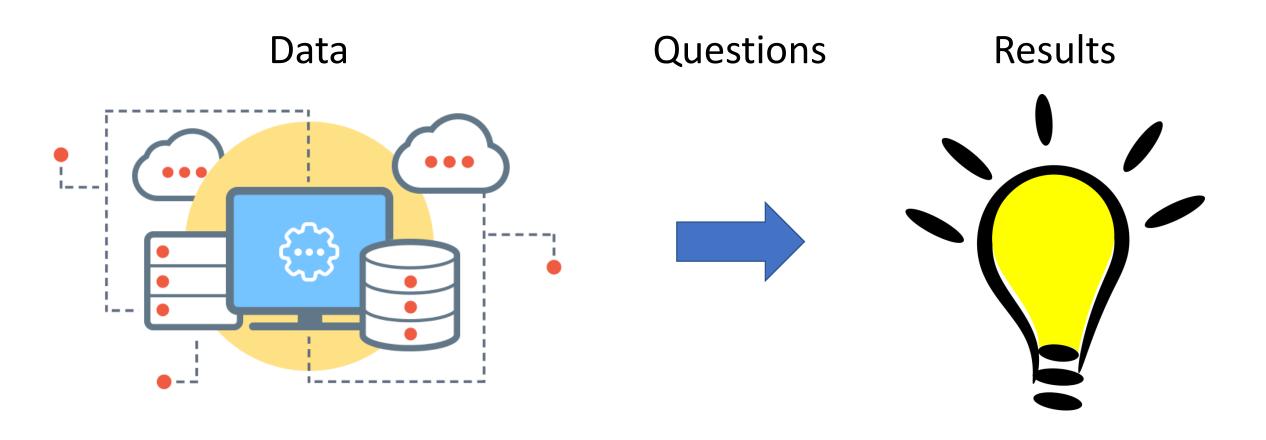
## Data System

## Query = Function(All Data)

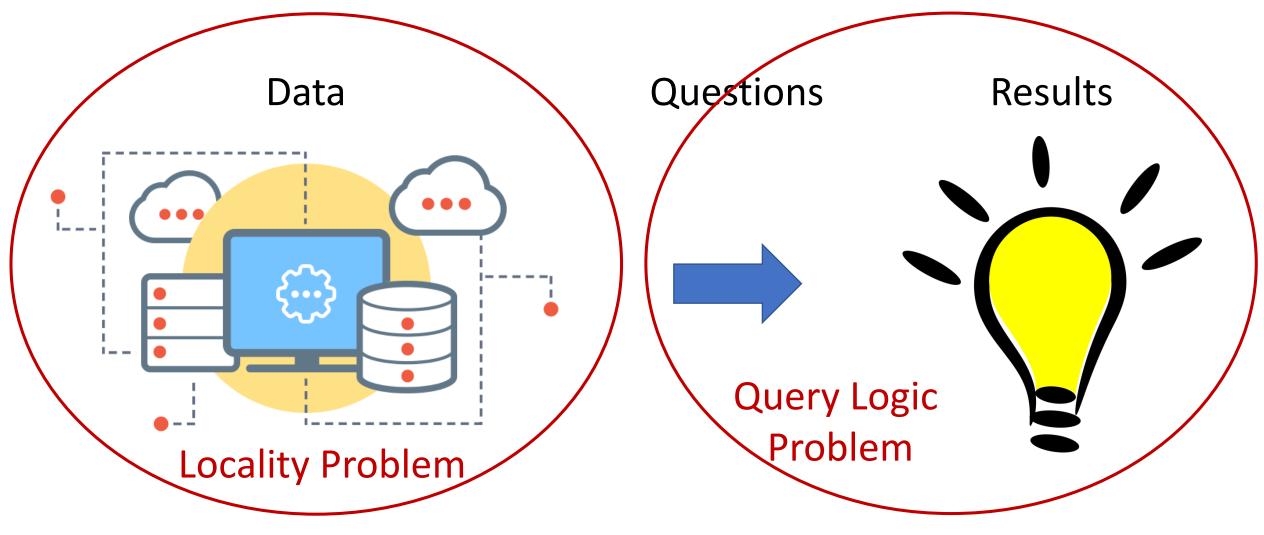


# Data Architecture

## Data is a value store



## Data is a value store

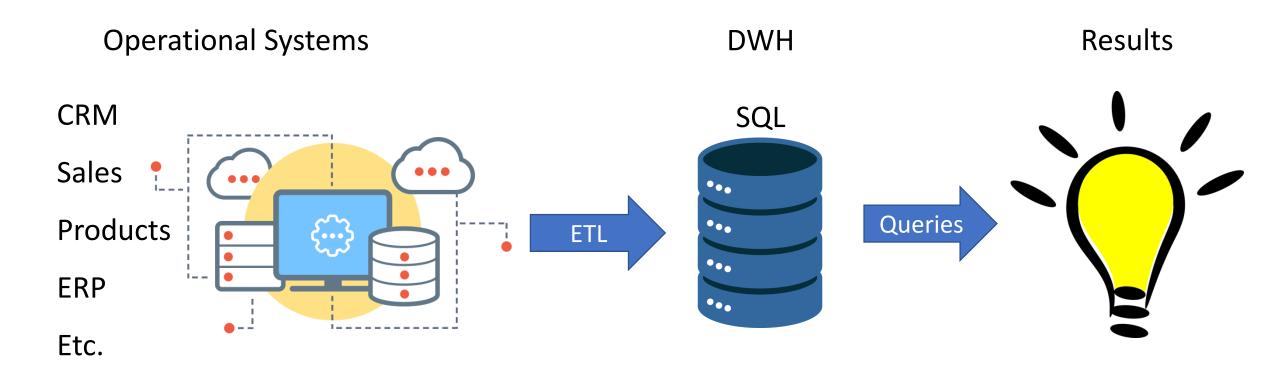


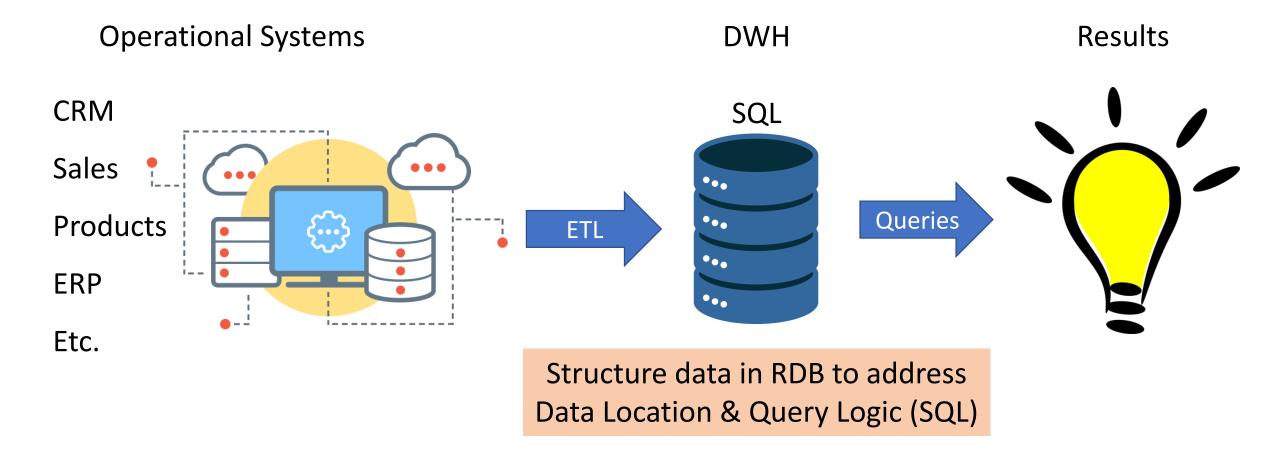
# Introducing the Data Warehouse

Centralise operational data

to enable

Business Intelligence & decision making





Load data from operational systems to central DWH and get Business Intelligence

#### Traditional process when building DWH

- 1. Start with use cases to identify required analysis and reports
- 2. Design DB schema and queries
- 3. Identify the required data sources
- 4. Create ETL pipelines to extract data from sources, transform it into target schema and load it in DWH (schema-on-write)
- 5. Query DWH for defined reports and analysis



#### The Good

- Centralized and fresh data
- Fast queries
- Transactional

#### The Bad

- Hard to scale & non Elastic
- Mostly limited to internal structured data
- Not for advanced analytics & machine learning and
- Closed format (locked in)

# Need to collect a variety of data



Challenge to store structured transactional data with less structured data

Do not necessary know in advance what use cases to make with the data

Big Data = All Data

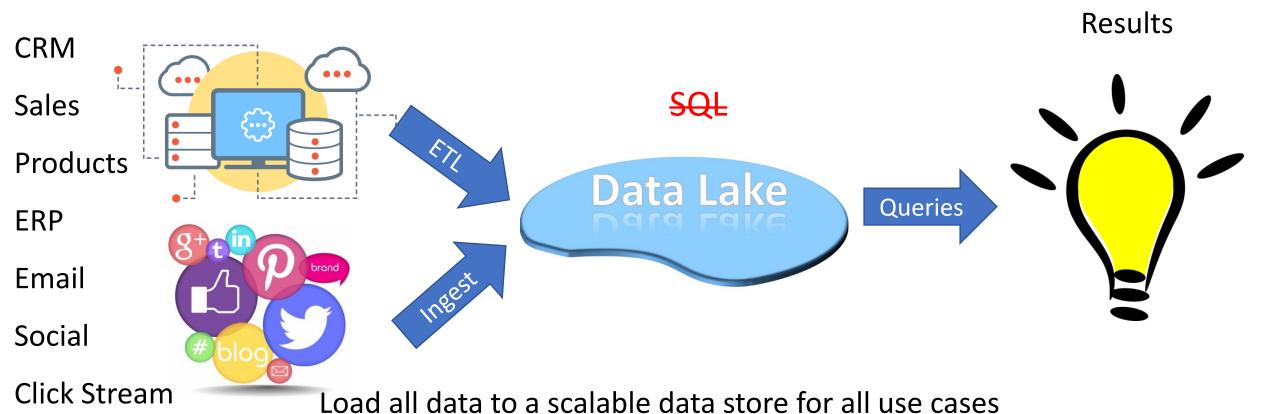
# So was born the Data Lake

Get <u>ALL DATA</u> in one scalable store to enable

Any future use case

## Data Lake

Internal + External Data



Ingest all data

Store all Data

Perform Analysis

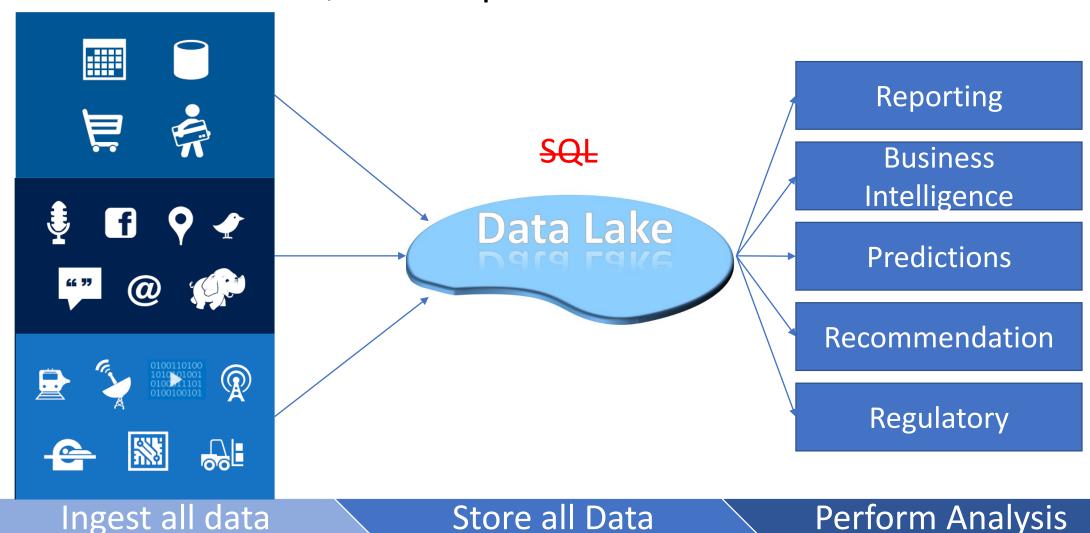
No structuring required

In native format (no schema)

On distributed computing engines

## One Data Lake, Multiple use cases

No structuring required



In native format (no schema)

On distributed computing engines

### Data Lake

#### The Good

- Massive scalability
- Relatively inexpensive (per TB)
- Built on open formats
  - Avro, ORC, Parquet
- Adapted for advanced analytics and machine learning
- Raw data (all data)

#### The Bad

- Raw data
  - Inconsistent
  - no schema
- Requires processing
  - Analytics
  - Reporting
- Performance

### Data Lake

#### Modern approach for data systems

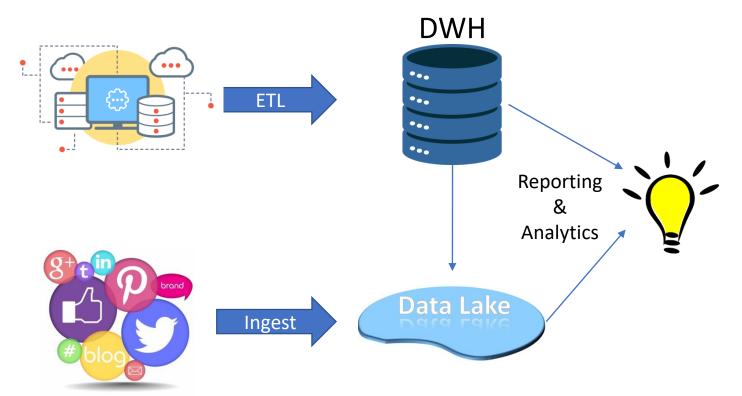
- Use cases may be defined in the future → store data in native format (raw data), do not impose a rigid schema
- Schema is imposed on data read (queries, analysis, etc.)
- schema-on-read



# Building a Data Lake is Complex

And there is no ONE way to do it

## Augmented Data Warehouse

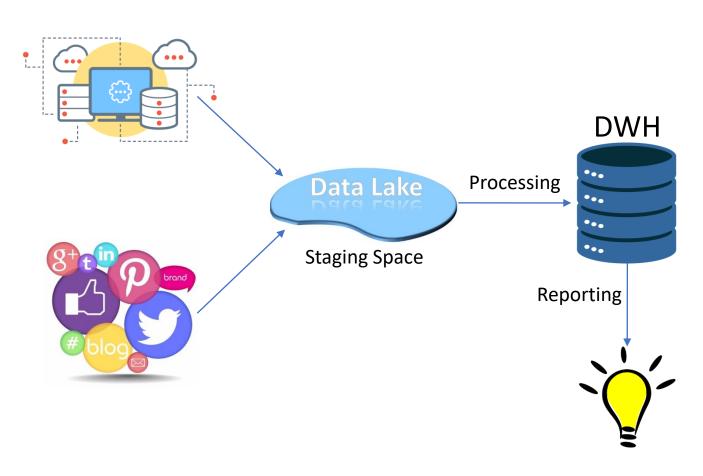


- When DWH already exists
- DWH incapable of handling new data
  - New data goes directly to Data lake

#### Cons

- Complex architecture: two systems
- Complex ingestion (3 ways)
- Data with different maturity

## Data Lake & Data Warehouse

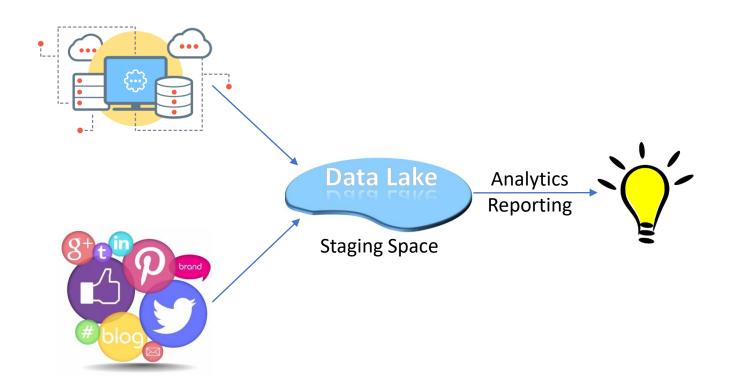


- Data Lake centralizes raw data in one place
- Data processed and moved to DWH
  - Batch processing

#### • Cons

- Not using all the potential of Data Lake
- Not suited for any data use case (advanced analytics and machine learning)

## All in one Data Lake

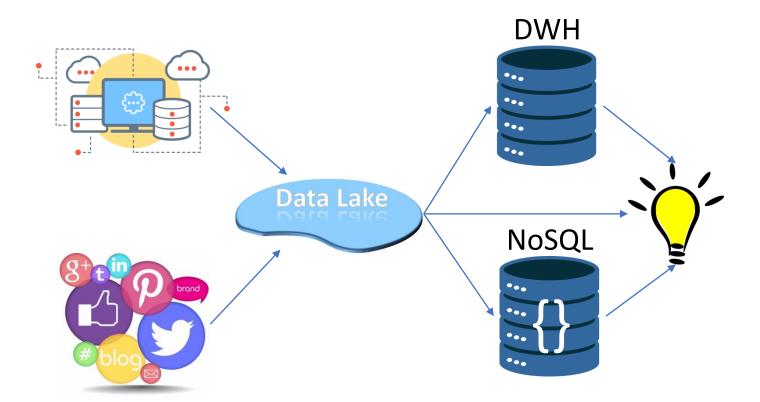


- When there is no existing DWH
- Data Lake for all usages
  - Staging, processed data, reporting, analytics

#### Cons

- Performance (especially for interactive reporting)
- Can't use existing BI tools (or requires major adaptation)

# Target Architecture



- Use the complete Data ecosystem (Data Lake, DWH, NoSQL)
- Data centralized in the Data Lake
- Data harmonized in the Data Lake
- Processing results copied to DWH or NoSQL DB as required
  - Read Only
- Future proof: adapted for today and future use cases

## Data Lake

Paradigm shift

Before: Structure → Ingest → Analyse

Now: Ingest → Analyse → Structure

## Data Lake

#### Application data

Business logic on cleansed data. Ready for consumption by applications.

#### Governance

#### Cleansed area

Uniform the way files are stored (format, encoding, ...). Also called conformed layer

Staging area – Raw data (immutable)

Also called landing zone

# Lambda Architecture

Lambda architecture is a data-processing architecture designed to

handle **massive quantities** of data by taking advantage of both **batch** and **stream-processing** methods." (Wikipedia)



#### Nathan marz

- > Ex- Twitter Engineer
- Creator of Apache Storm

http://nathanmarz.com/blog/how-to-beat-the-cap-theorem.html

# Lambda Architecture: Why

- Fault-tolerance
  - against hardware failures
  - and against human errors (which occur more frequently: bugs, errors, etc)
- Support for a variety of use cases
  - Low latency querying as well as updates
- Linear scale-out capabilities
  - Adding more machines 
     speed up the whole process
- Extensibility to support new use cases easily

# Lambda Architecture: Properties

- Scalable data store
  - Easily store and scale immutable, constantly growing dataset
- Append only data store
  - The primary write operation is adding new immutable data
  - Updates and deletes are no different than creating new data
- Avoid complexity of the CAP theorem (and human errors)
  - recomputing queries from raw data
- Incremental algorithms
  - Lower latency of queries to an acceptable level

## Lambda Architecture: Master Dataset

Master Dataset: the source of truth in the Lambda Architecture.

• If we loose the computed views (failure) we could reconstruct the application from the master dataset.

#### Data in master dataset is:

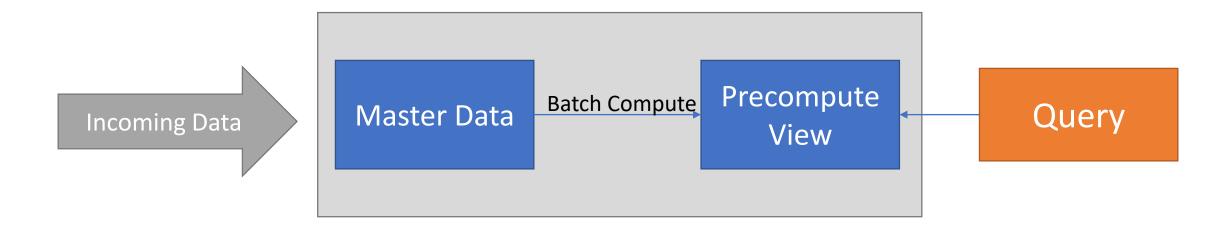
- Raw in its original format: append only
- Immutable: write once read many
- Eternally true

### Lambda Architecture

Lambda is a layered architecture composed of 3 components

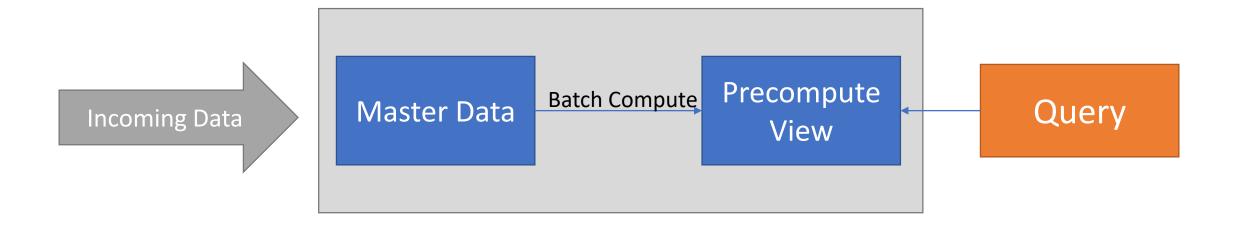


- Batch layer precomputes the master dataset into batch views
- Queries can be resolved with low latency
- Unrestrained computation: can compute anything given enough time



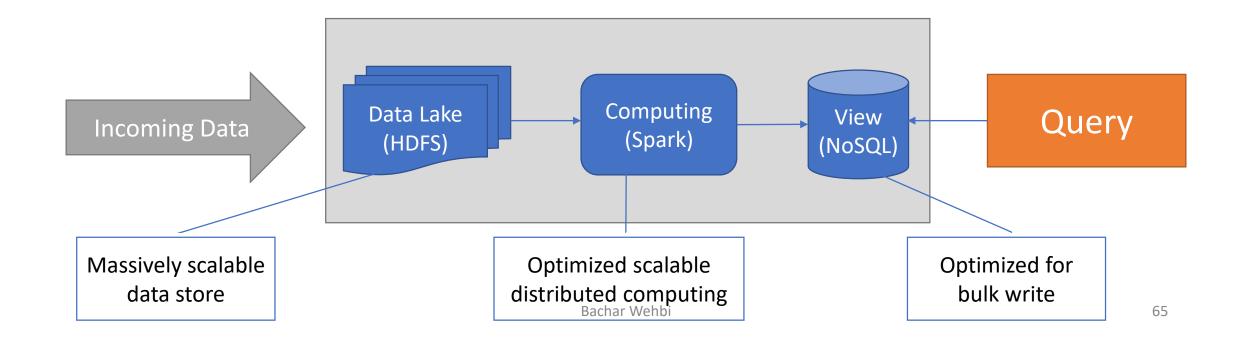
(Re)-Computation based on <u>immutable</u> dataset is <u>idempotent</u>

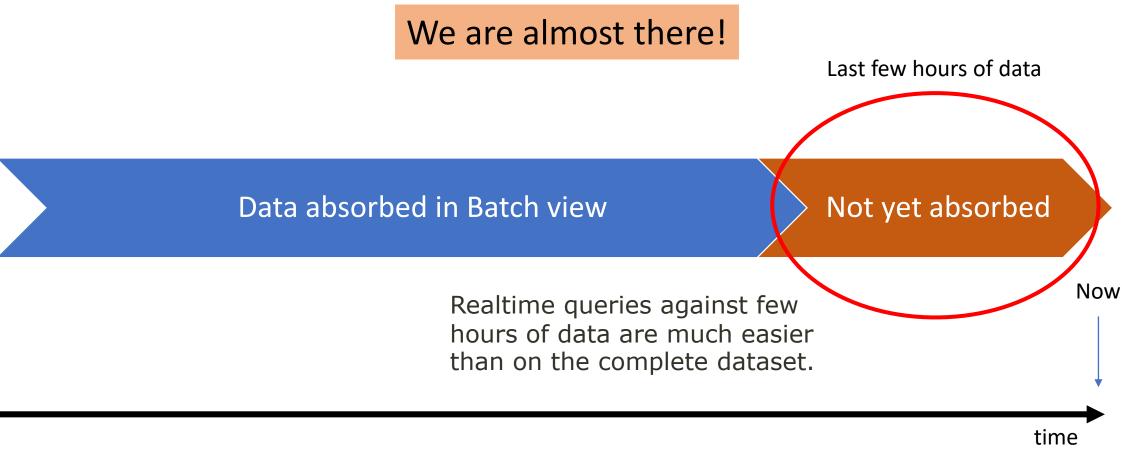
→ Recomputing will yield to the same result



(Re)-Computation based on immutable dataset is idempotent

→ Recomputing will yield to the same result

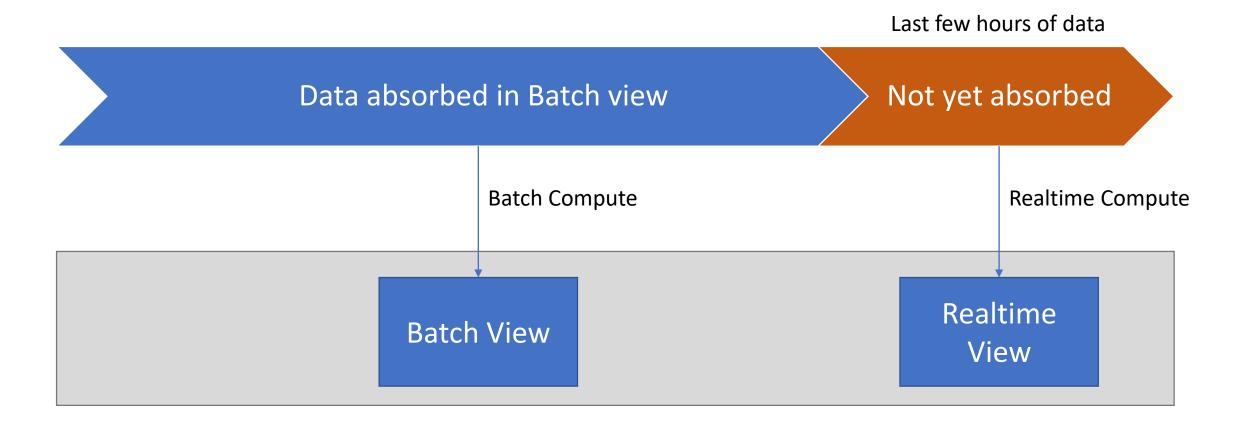




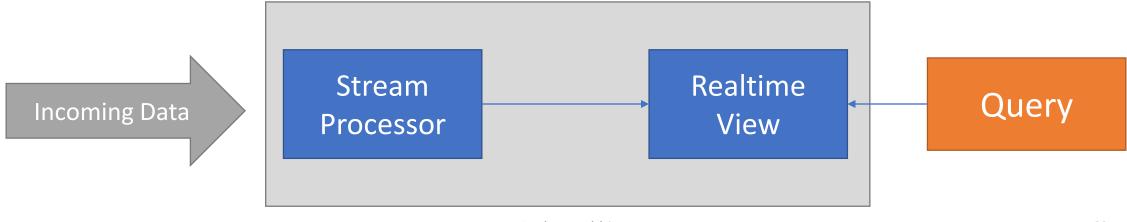
# Speed Layer

The Batch layer processes data up to few hours precision.

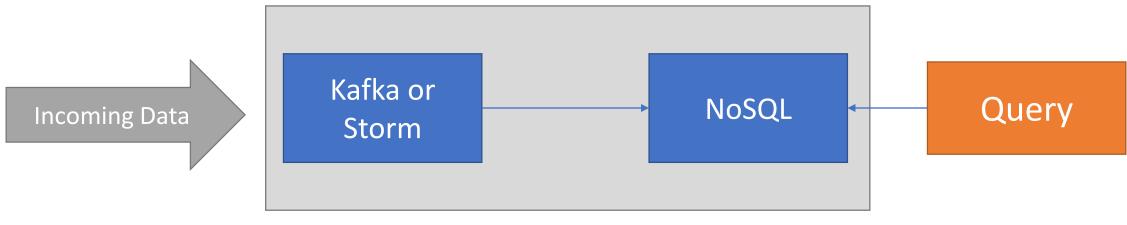
The Speed layer's objective is to compensate for the last few hours of data



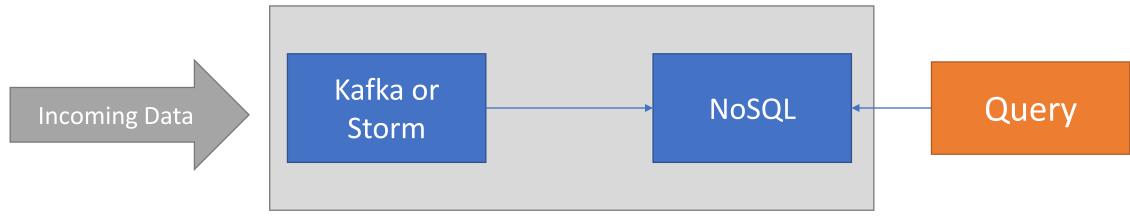
- Two main tasks for the Speed Layer:
  - Continuously process incoming data stream
  - Store the results of the real-time views
- Stores a limited window data (to compensate data not considered in Batch layer)
- If anything goes wrong, it is autocorrected in the next Batch View



- Stream Processing
  - PubSub Architecture is usually fine
  - Can retain (store) data for a relatively short period
  - Realtime processing capability of data as it comes
  - Incremental algorithms



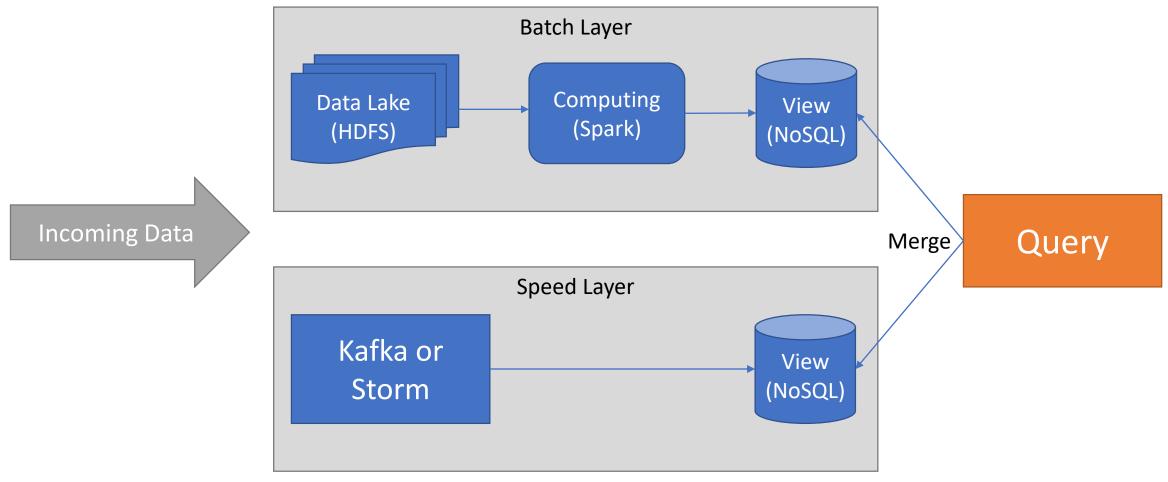
- Realtime views
  - Stored in random write database (incremental calculations)
  - Should support random read (to serve queries)
  - Performance should account large amount of random read & write operations



# Serving Layer

Queries the Batch and Speed layers and merges their results

# Lambda Architecture: Serving Layer



Bachar Wehbi

73

### Lambda Architecture: Advantages

- Data is immutable
  - Simplifies reasoning about data (no updates, no deletes)
  - Modelling data transformations from source data
  - Transformations are idempotent, repeat them any number of times leads to the same result
- Complexity isolated in the Realtime layer
  - Batch layer corrects any coherence issues in Realtime views
- Fault-tolerance (Human & System)
  - Discard views (Batch and Realtime) and recompute everything
  - Takes into account reprocessing data

#### Lambda Architecture: Cons

- Two different layers to maintain
  - 2 codes one for Batch Layer one for Speed Layer
  - They need to produce the same result
  - More technologies to deal with
  - Merging views is not easy and brings additional complexity of maintaining potentially 2 databases for the serving layer
- Two different and diverging programming paradigms to build a Lambda architecture.
- Requires multiple competences to put it to work.

# KAPPA Architecture

"The Lambda Architecture has its merits, but alternatives are worth exploring." (Jey Kreps)



#### Jey Kreps

- > Ex- LinkedIn
- > Co-Creator of Apache Kafka
- > CEO of Confluent

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

## Kappa Architecture: Why

• Why can't the stream processing system be improved to handle the full problem of maintaining two systems in Lambda Architecture?

• If you have a sufficiently powerful stream processing system you can do most of the view building (transformation & computing) directly in the stream processing system.

➤ Simplify the Lambda Architecture by removing the batch layer.

### Kappa Architecture: Properties

- The datastore is retained in an append only Log (Write Ahead Log)
  - Immutable data similar to Lambda Architecture
  - Stores data in the stream system as a Log instead of an object store
- Processing results are stored in real-time views (usually NoSQL)
  - Processing data is idempotent as in Lambda Architecture
- Reprocessing can be done by consuming the entire history of data
- Fault Tolerant: throw the view are recompute the views!

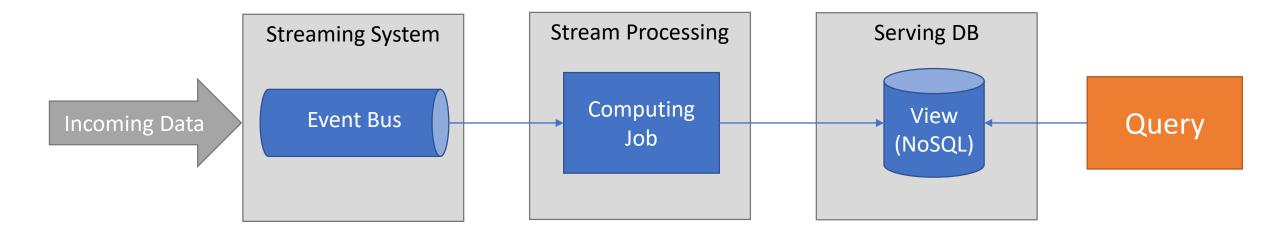
# Kappa Architecture: How (1/2)

- Use a Stream system as Apache Kafka
  - This is a publish subscribe paradigm
  - System should allow multiple subscribers
- Retain data for as long as needed
  - example 30 days, 1 year, ... This depends on the application
  - Data can be retained with unlimited history as well
    - Directly in the stream system (Kafka can handle Peta Bytes) or in an massively scalable object store like HDFS

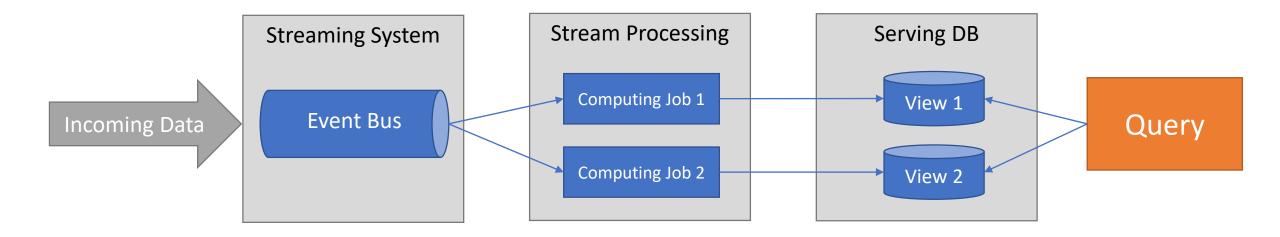
# Kappa Architecture: How (2/2)

- Start a computing job by subscribing to the data topic
- Write the results in a view (usually NoSQL)
  - Random read, Random write database
- Reprocessing can simply be done:
  - start a new instance of the stream processing
  - Point to the start of the retained data
  - Store results in a new destination
  - When the result is ready, switch from old job to new job

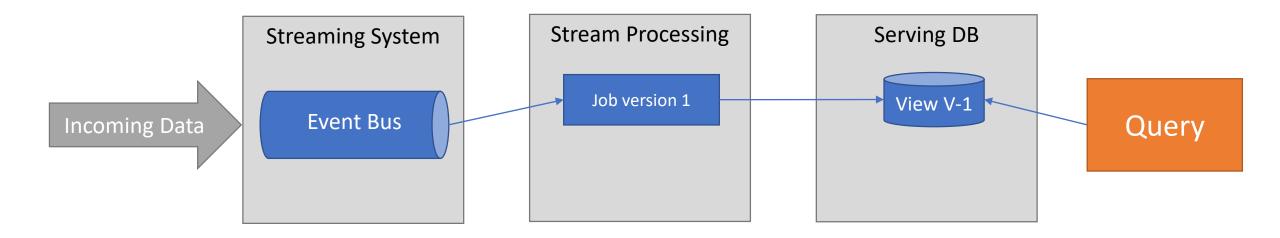
Kappa architecture high level conceptual architecture



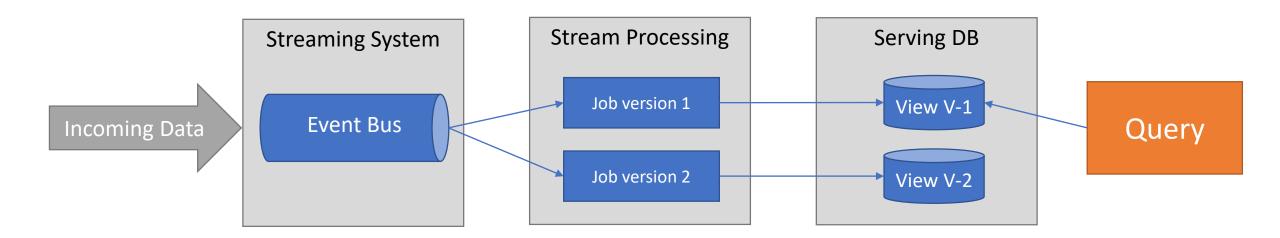
One data stream – multiple processing jobs



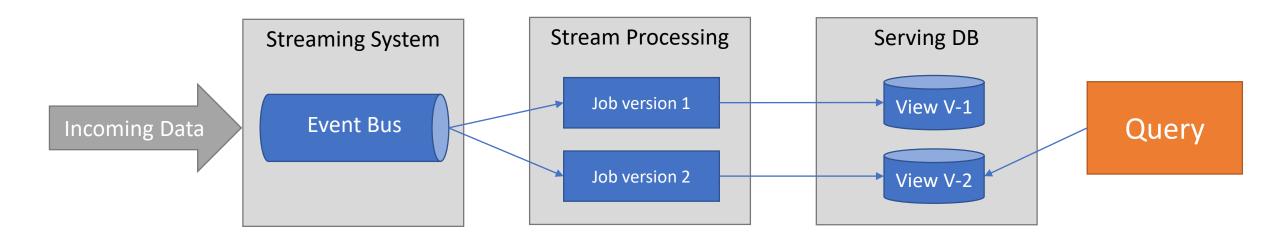
How reprocessing works?



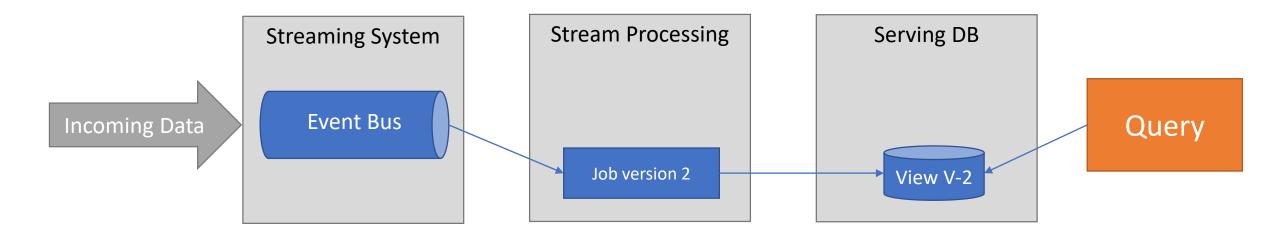
Reprocessing data gets very simple: change your code and run it against retained data



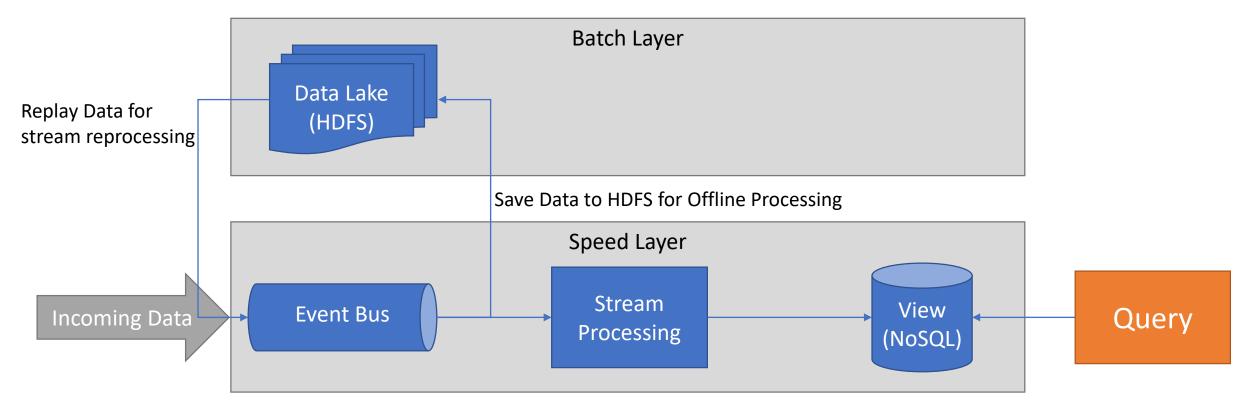
Reprocessing data gets very simple: change your code and run it against retained data When the second job is ready, read from new table (view V-2)



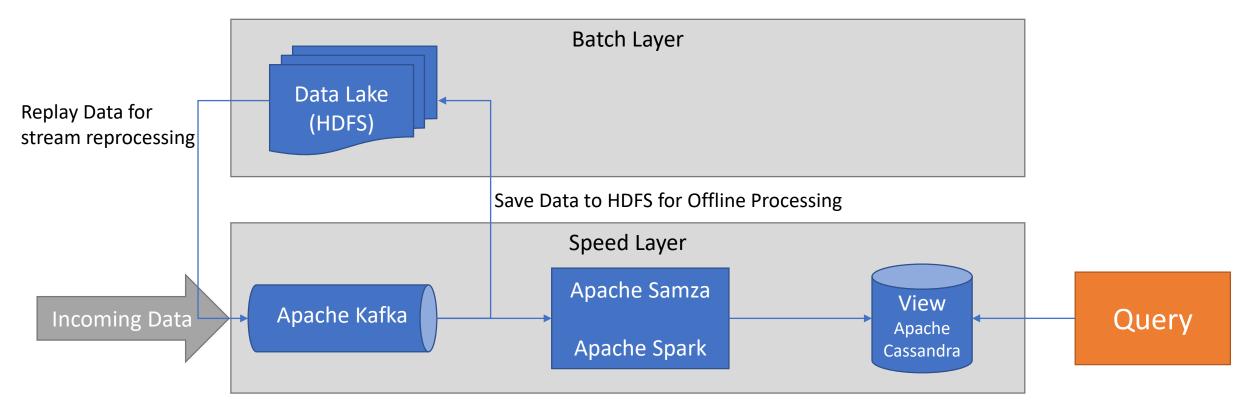
Reprocessing data gets very simple: change your code and run it against retained data When the second job is ready, read from new table (view V-2) Stop the old version and delete its results



Kappa architecture as a simplified view of Lambda architecture



Kappa architecture as a simplified view of Lambda architecture



#### Kappa Architecture: Pros

- Simplicity: single processing platform
  - Only one paradigm and one code to maintain
- Latency: well adapted for real-time and time sensitive applications
- Configurable data retention and possibility to mirror data to HDFS to keep a full history
  - Data in both cases is immutable
- No periodic reprocessing as in Lambda
  - Reprocess data only when the code changes
- Run in parallel multiple versions of the processing job
  - Good for A/B testing

### Kappa Architecture: Cons

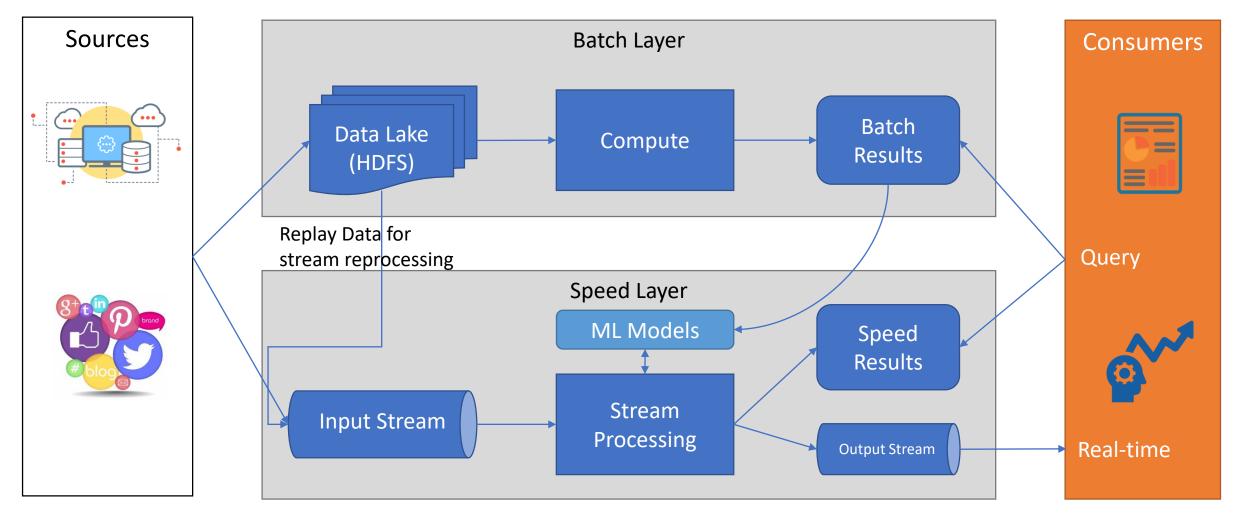
Effort to adapt processing for real time

- Not a silver bullet for every Big Data project
  - Not meant for machine learning projects
  - More adapted for event based and streaming projects

# Unified Architecture

Combines the advantages of both Lambda and Kappa architectures

#### Unified Architecture



### Data Systems are Complex

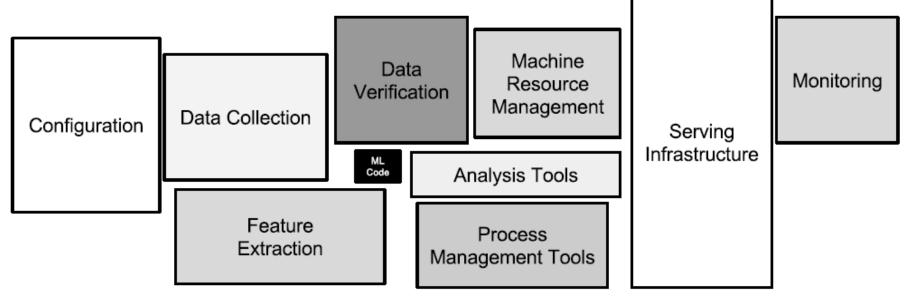


Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

Big Data is hard, really hard!

#### References

- http://milinda.pathirage.org/kappa-architecture.com/
- https://www.youtube.com/watch?v=fU9hR3kiOK0
- https://www.youtube.com/watch?v=aJuo bLSW6s&feature=youtu.be
- https://www.oreilly.com/ideas/questioning-the-lambda-architecture
- https://engineering.linkedin.com/distributed-systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying
- https://ferd.ca/beating-the-cap-theorem-checklist.html
- https://martinfowler.com/eaaDev/EventSourcing.html
- https://martinfowler.com/bliki/CQRS.html
- http://lambda-architecture.net/stories/
- http://nathanmarz.com/blog/how-to-beat-the-cap-theorem.html
- https://codahale.com/you-cant-sacrifice-partition-tolerance/
- https://www.voltdb.com/blog/2010/10/21/clarifications-cap-theorem-data-related-errors/
- <a href="http://blog.cloudera.com/blog/2010/04/cap-confusion-problems-with-partition-tolerance/">http://blog.cloudera.com/blog/2010/04/cap-confusion-problems-with-partition-tolerance/</a>
- https://docs.microsoft.com/en-us/azure/architecture/patterns/event-sourcing
- https://mapr.com/developercentral/lambda-architecture/
- http://cs-www.cs.yale.edu/homes/dna/papers/abadi-pacelc.pdf
- <a href="https://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed">https://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed</a>

#### References

- https://databricks.com/blog/2015/04/21/analyzing-apache-accesslogs-with-databricks-cloud.html
- https://www.oreilly.com/ideas/applying-the-kappa-architecture-inthe-telco-industry
- http://samza.apache.org/learn/documentation/0.7.0/jobs/reprocessing.html

#### References - Data

- https://www.ibm.com/developerworks/library/wa-introhdfs/
- https://cdn.oreillystatic.com/en/assets/1/event/160/Parquet%20performance%20tuning %20The%20missing%20guide%20Presentation.pdf
- https://techmagie.wordpress.com/category/big-data/avro-parquet/
- https://www.dataengineeringpodcast.com/data-serialization-with-dougcutting-and-julien-le-dem-episode-8/
- http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoophdfs/HdfsDesign.html#Data Replication
- http://hadooptalk.com/awesome-introduction-of-parquet/
- https://akiselev87.wordpress.com/2016/10/26/spark-and-data-formats-introduction/