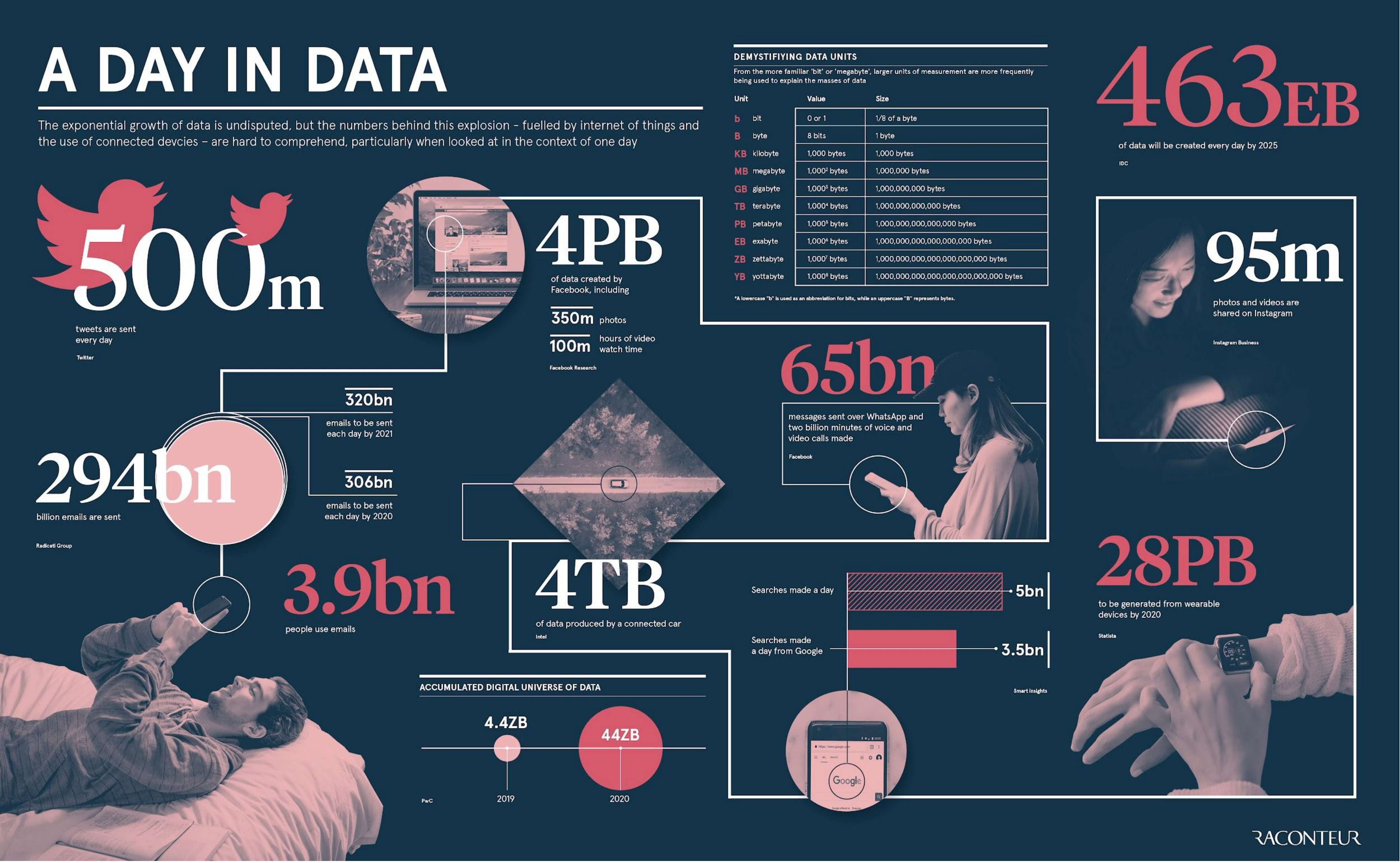
ETL
Extract, transform, and load Data

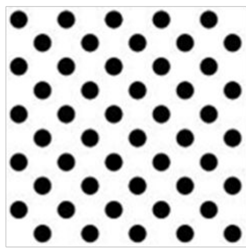
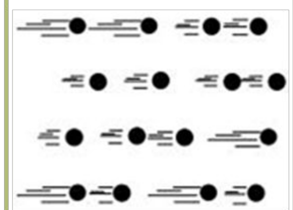
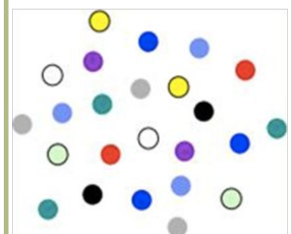
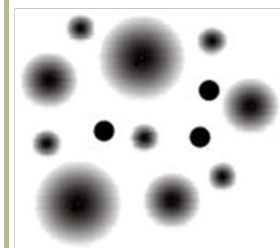
FSD312 Data Engineering



https://everysecond.io/the-internet

5 V's of Big Data

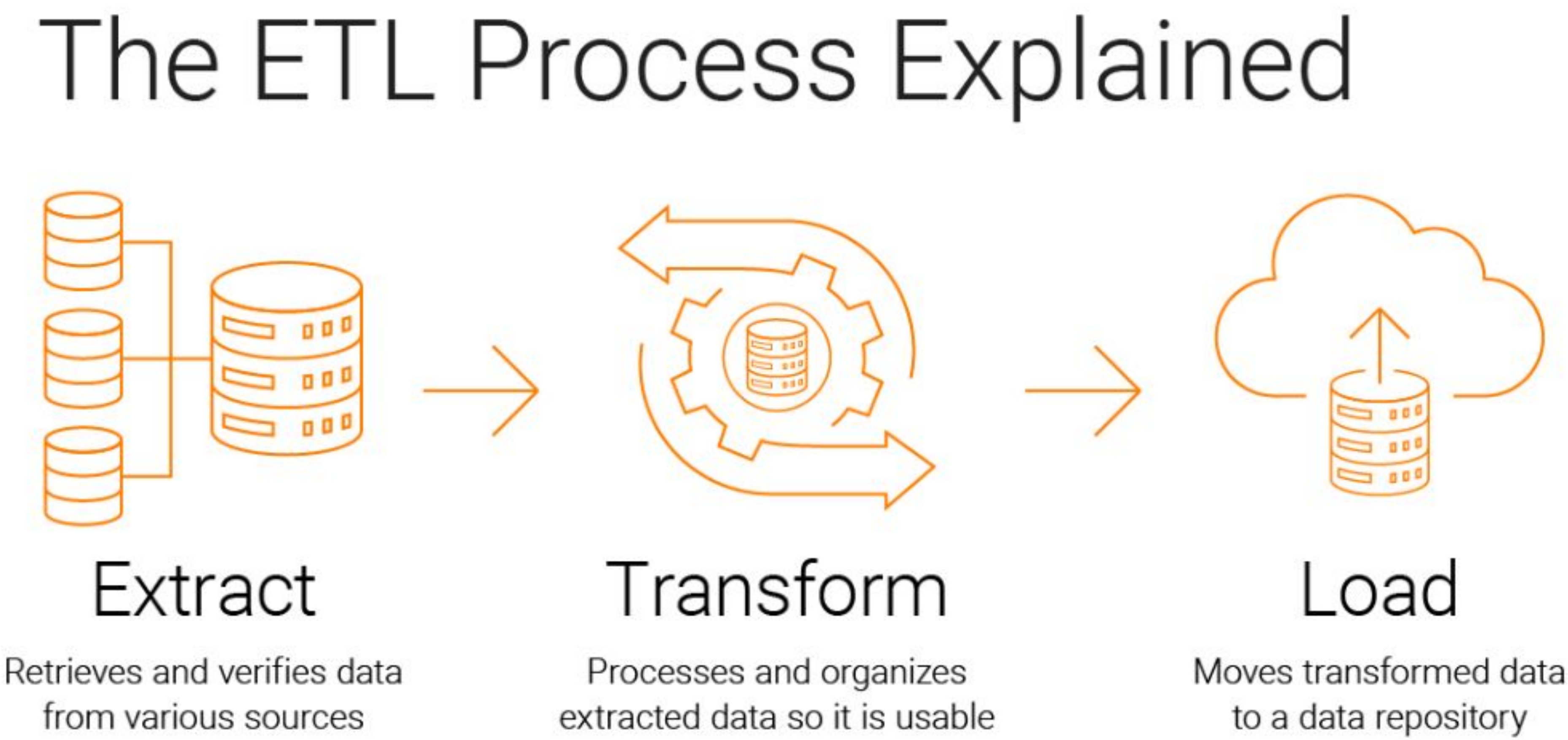
1. **Volume**
2. **Velocity**
3. **Variety**
4. **Veracity**
5. **Value**

Volume	Velocity	Variety	Veracity	Value
				
Data at Rest	**Data in Motion**	**Data in Many Forms**	**Data in Doubt**	**Data into Money**
Terabytes to Exabytes of existing data to process	Streaming data, requiring milliseconds to seconds to respond	Structured, unstructured, text, multimedia,…	Uncertainty due to data inconsistency & incompleteness, ambiguities, latency, deception, model approximations	Business models can be associated to the data

Adapted by a post of Michael Walker on 28 November 2012

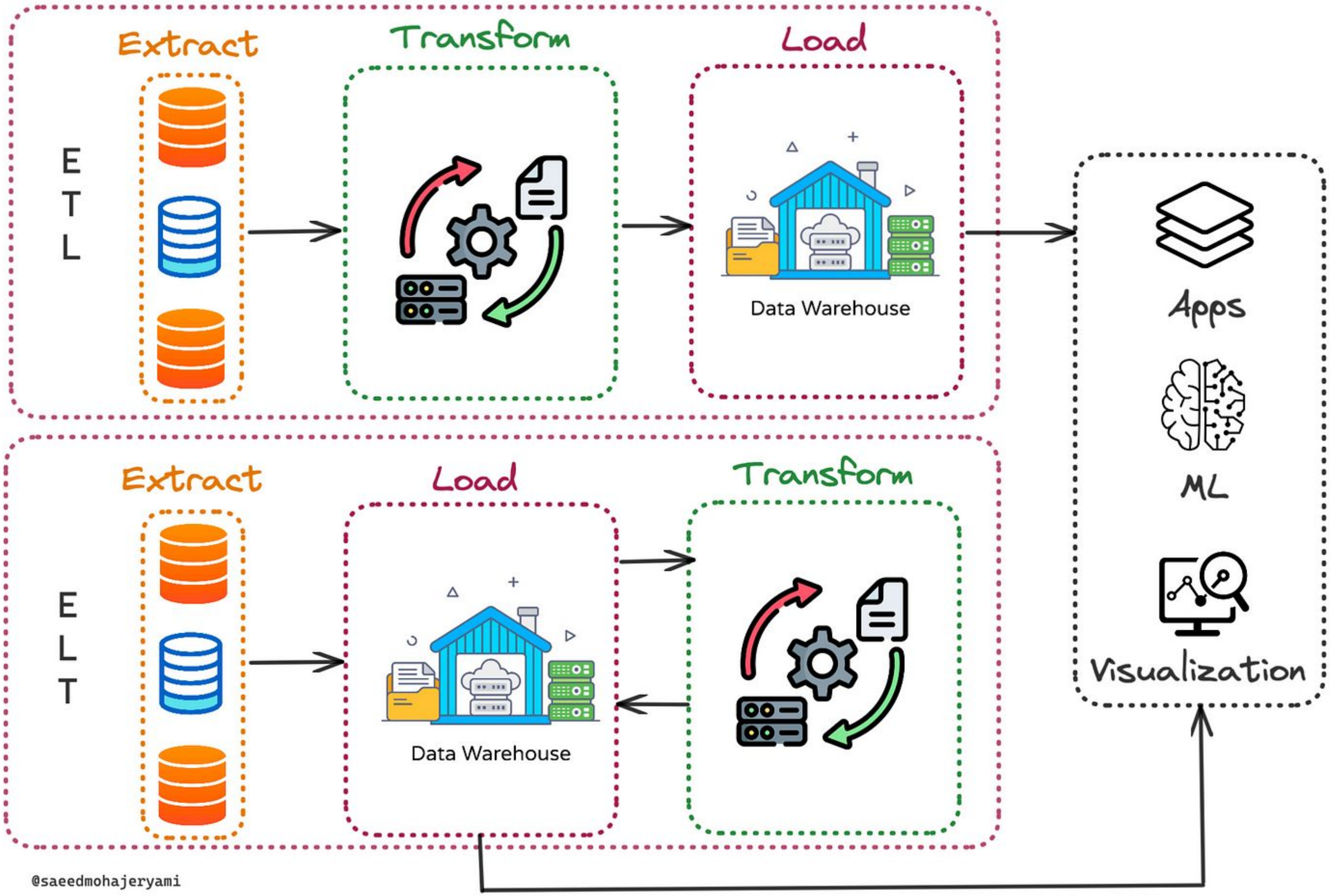
Institut Supérieur de l'Aéronautique et de l'Espace
SUPAERO

Extract, transform, and load



https://www.informatica.com/tw/resources/articles/what-is-etl.html

Extract, load, and transform



https://medium.datadriveninvestor.com/data-processing-evolution-etl-vs-elt-which-one-suits-your-business-needs-ea4082c8d36

Extraction

Get data from:

- A database
- A file system
- API: Application Programming Interface
- Web scraping

Check data sources:

- Is the format correct?
- Did the data arrive when expected?
- Did all data sources return data?
- Do the data make sense?
- Is the data complete, part of a batch, or part of a stream?

Transformation

- Transform data between formats
- Validate and clean data
- ETL: performed through scripts on temporary data, before loading data into long-term storage
- ELT: performed on stored data by scripts in the data platform

Transformation examples:

- Selecting data features
- Validating data
- Replacing missing values
- Encoding free-form values
- Calculating a new data value
- Joining data from multiple sources
- Deduplicating data
- Transposing or pivoting data

Loading

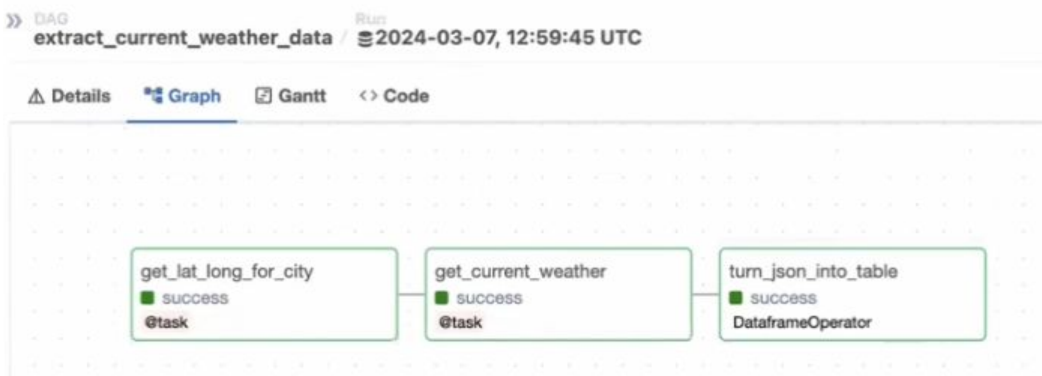
- Store recovered data
- Storage structures:
  - DBMS
  - Data warehouse
  - Data lake
- Allow analysis on stored data

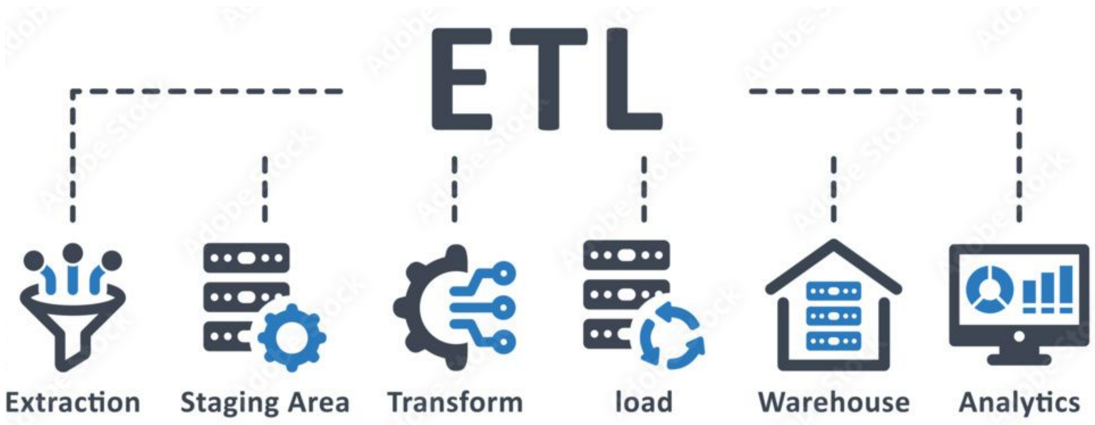
Data warehouse examples:

- Amazon Redshift
- Google BigQuery
- IBM Db2 Warehouse
- Microsoft Azure Synapse
- Oracle Autonomous Data Warehouse
- Snowflake
- Teradata Vantage

ETL tools

- Scripts
- Python
- Apache Airflow
- AWS Glue
- Google Dataflow
- Microsoft SQL Server Integration Services (SSIS)
- Airbyte
- Stitch





Storage structures

factor-bytes.com	Database	Data Lake	Data Warehouse
Data	Structured	Raw & Unstructured	Structured
Data Processing	Schema-on-write	Schema-on-read	Schema-on-write
Scalability	Varies	High	High
Cost	0 - $	$$	$$$
Users	Anyone	Data Scientists	Business Users
Use Cases	Real-time data processing, high transactional throughput, strong data consistency	Exploratory analysis, machine learning, data mining, or data science research	Reporting, analytics, and business intelligence

Database Management Systems (DBMS)

Système de gestion de base de données (SGBD)

Relational

MySQL

PostgreSQL

NoSQL (Not only SQL)

mongoDB

CouchDB relax

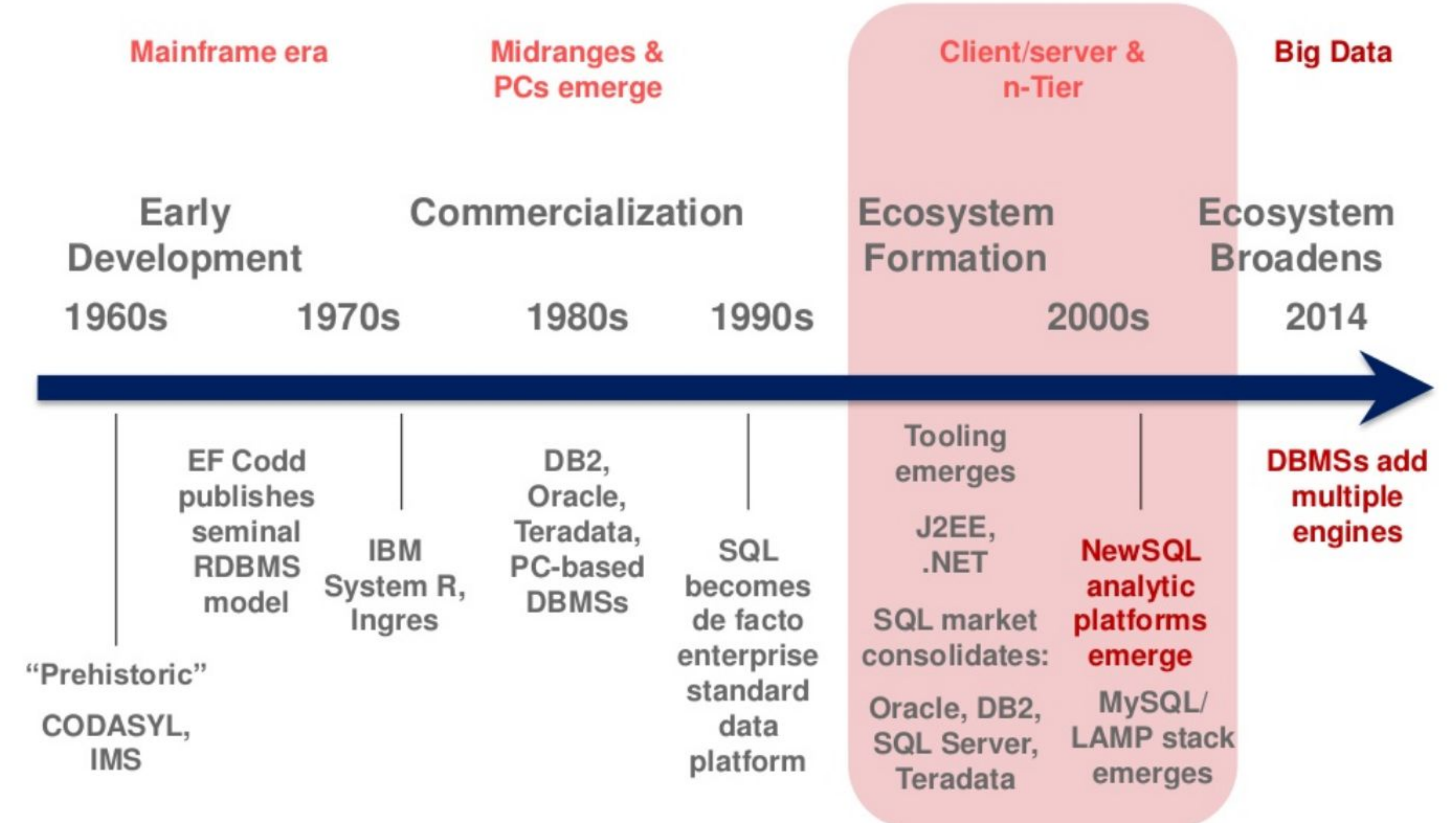
redis

NewSQL

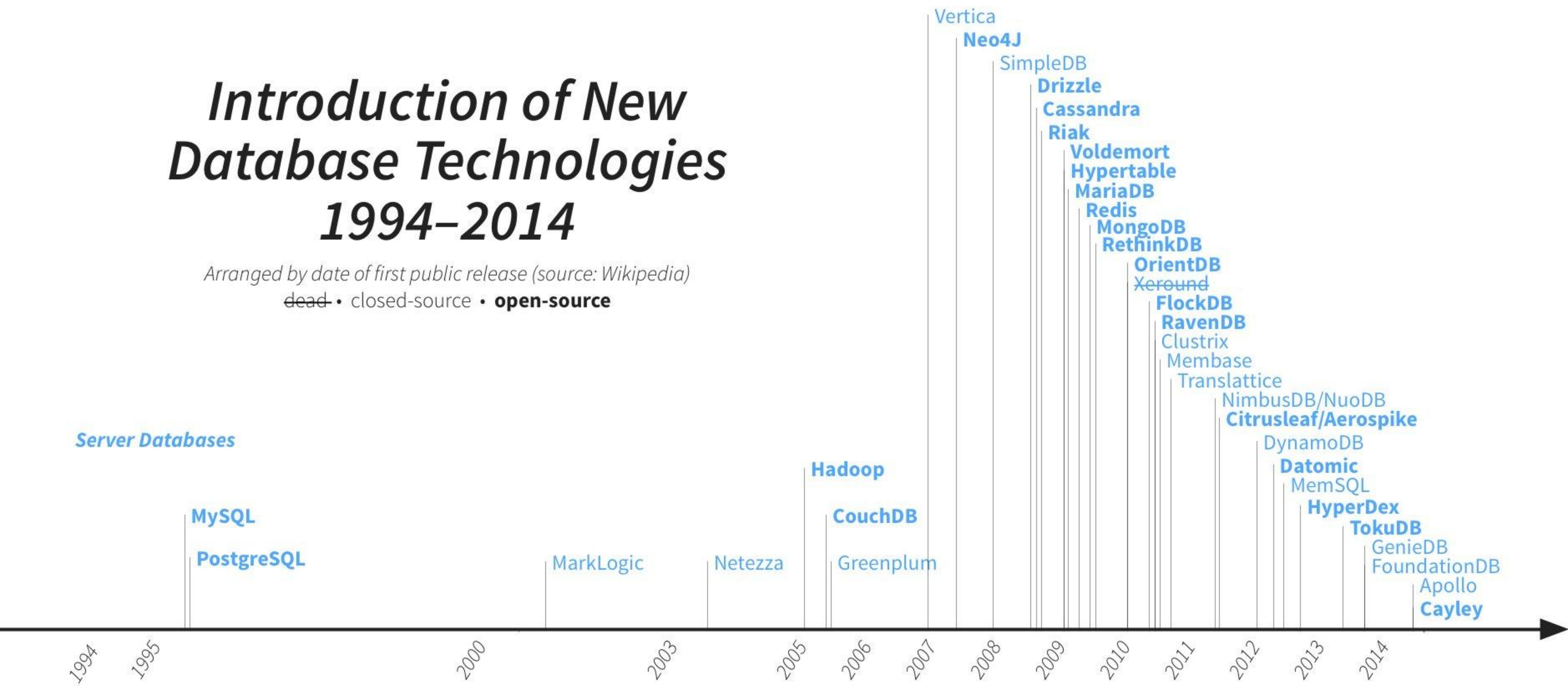
Cockroach DB

Apache Trafodion

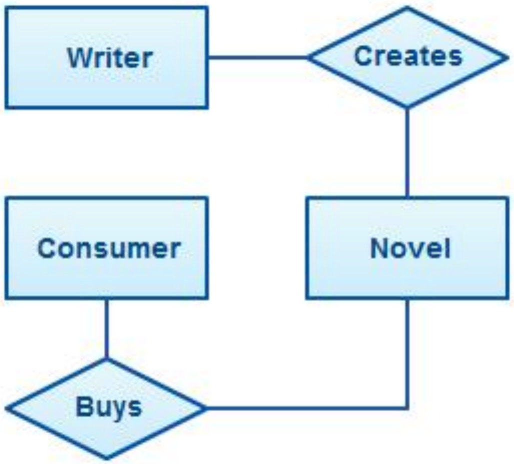
DBMS history



Recent DBMS history



RDBMS : relations



Writer

ID	First Name	Last Name	Affiliation	Age
382	George	Martin	ASOIAF	72

Creates

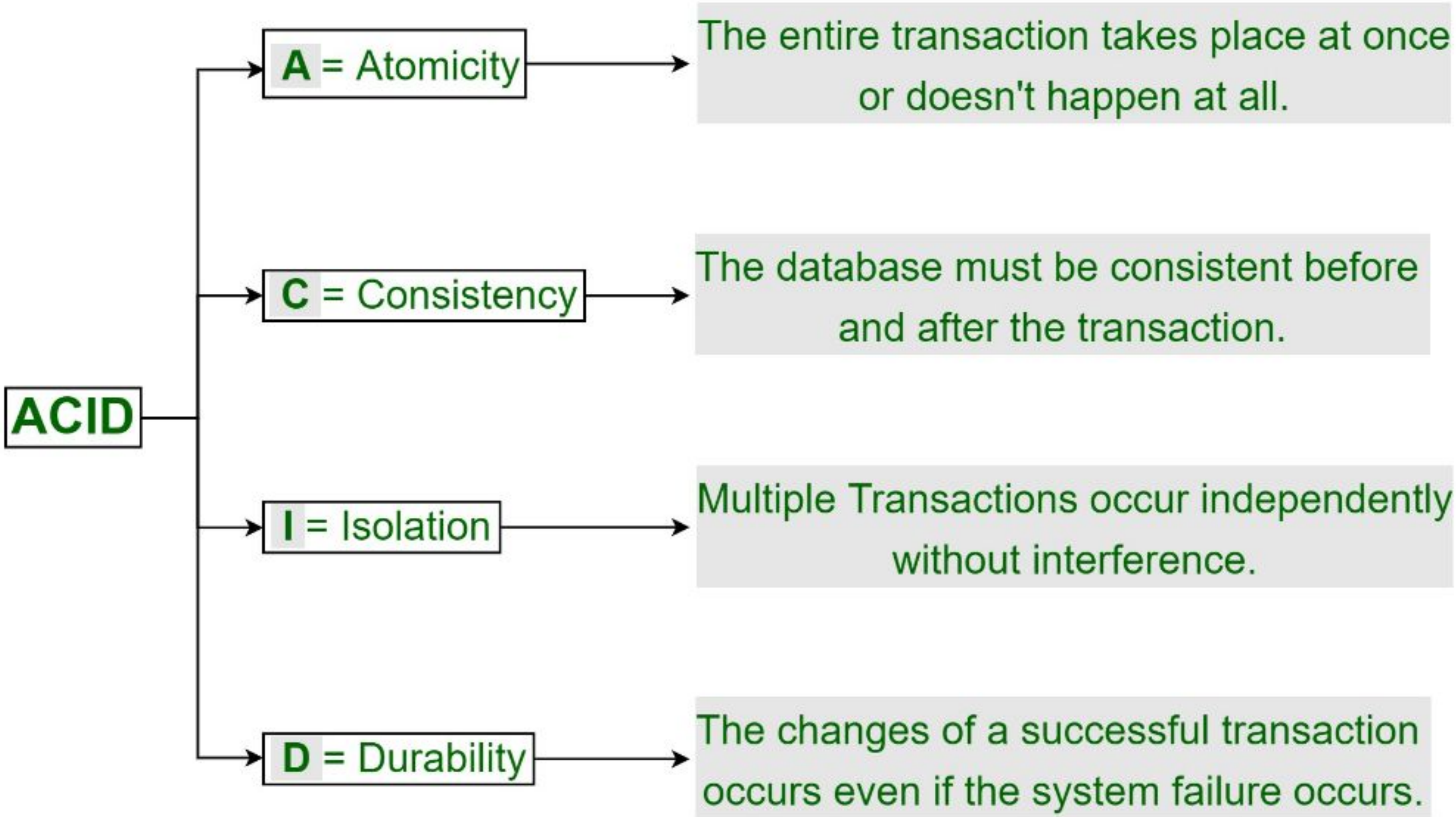
Writer_ID	Novel_ID
382	9772

Novel

ID	Title	Publication Year	Genre
9772	A Game of Thrones	1996	Fantasy

PostgreSQL MySQL

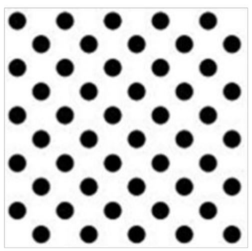
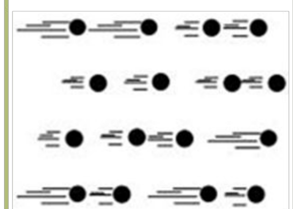
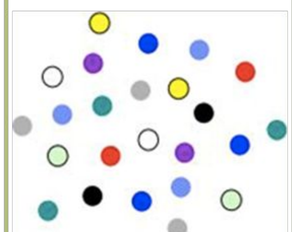
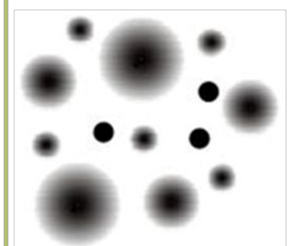
ACID



SQL Server, Oracle, MySQL, PostgreSQL, others...

Limits of RDBMS

1. RDBMS can not handle ‘Data Variety’
2. Difficult to change tables and relationships
3. Structural limitations
4. Strict application of ACID principles

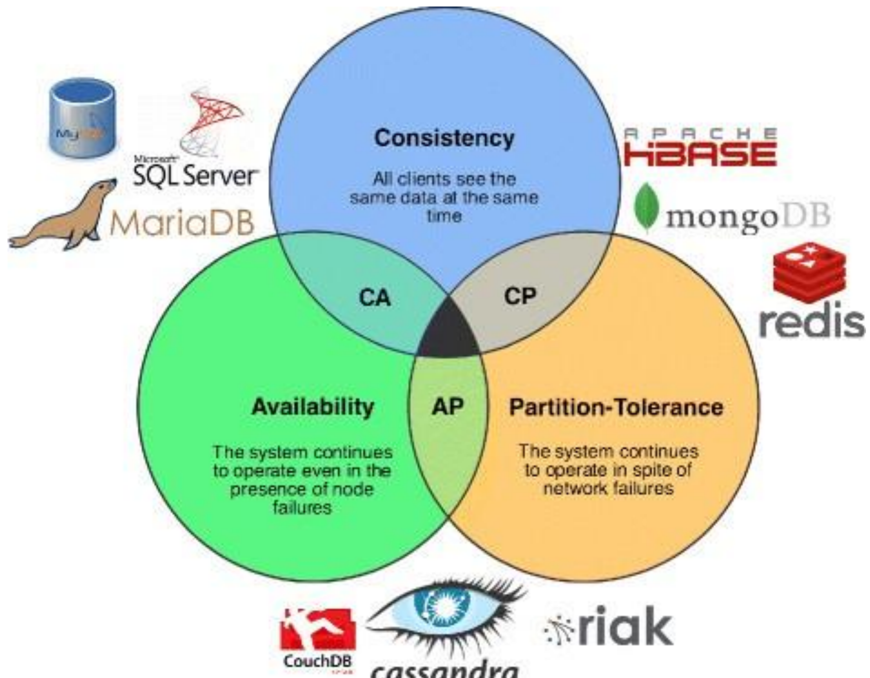
Volume	Velocity	Variety	Veracity	Value
				
Data at Rest	**Data in Motion**	**Data in Many Forms**	**Data in Doubt**	**Data into Money**
Terabytes to Exabytes of existing data to process	Streaming data, requiring milliseconds to seconds to respond	Structured, unstructured, text, multimedia,…	Uncertainty due to data inconsistency & incompleteness, ambiguities, latency, deception, model approximations	Business models can be associated to the data

Adapted by a post of Michael Walker on 28 November 2012

Brewer's theorem (CAP)

It is impossible for a distributed data store to simultaneously provide more than two out of the following three guarantees:

- **C**onsistency: Every read receives the most recent write or an error
- **A**vailability: Every request receives a (non-error) response, without the guarantee that it contains the most recent write
- **P**artition tolerance: The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes



BASE

- **B**asically **A**vailable: basic reading and writing operations are available as much as possible (using all nodes of a database cluster), but without any kind of consistency guarantees (the write may not persist after conflicts are reconciled, the read may not get the latest write)
- **S**oft state: without consistency guarantees, after some amount of time, we only have some probability of knowing the state, since it may not yet have converged
- **E**ventually consistent: If the system is functioning and we wait long enough after any given set of inputs, we will eventually be able to know what the state of the database is, and so any further reads will be consistent with our expectations

Main objectives for NoSQL databases

NoSQL (Not Only SQL)

Characteristics of NoSQL:

- Schema free
- Eventually consistent (as in the BASE property)
- Replication of data stores to avoid Single Point of Failure.
- Can handle Data variety and huge amounts of data.

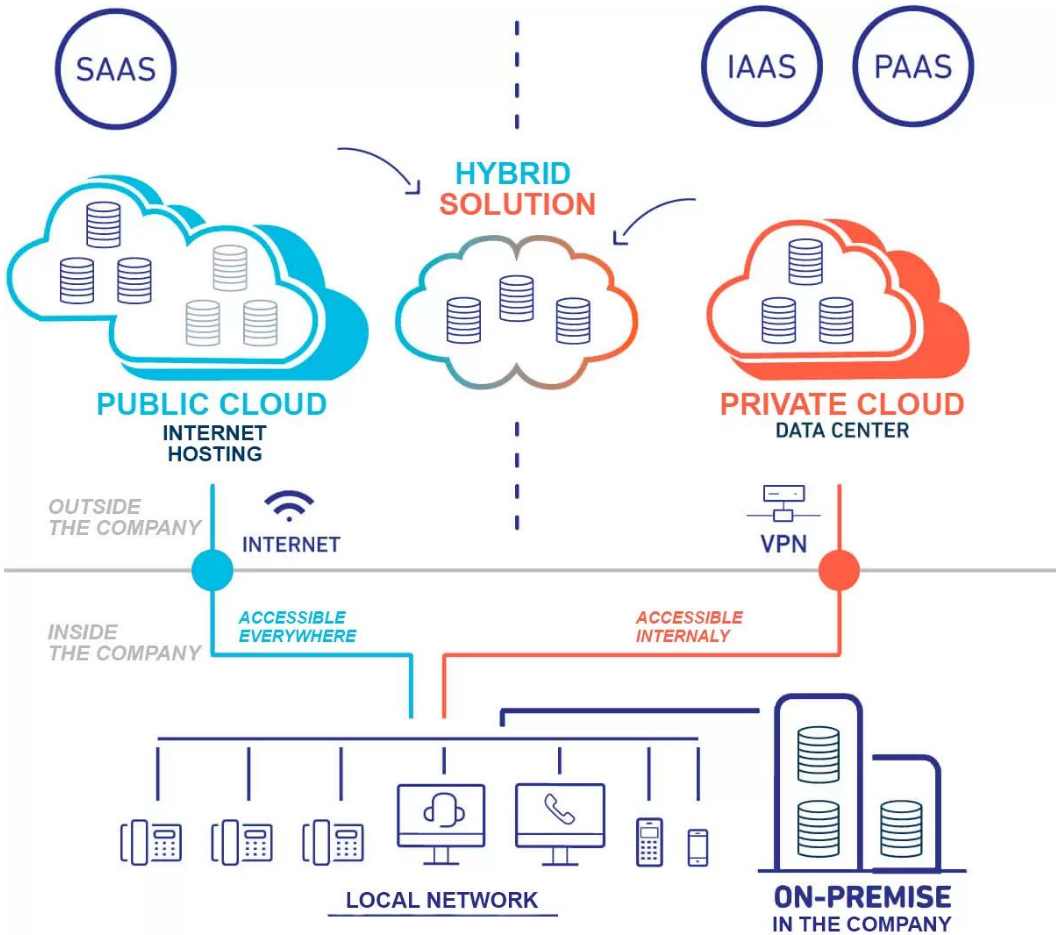
Key value Stores — Riak, Voldemort, and Redis

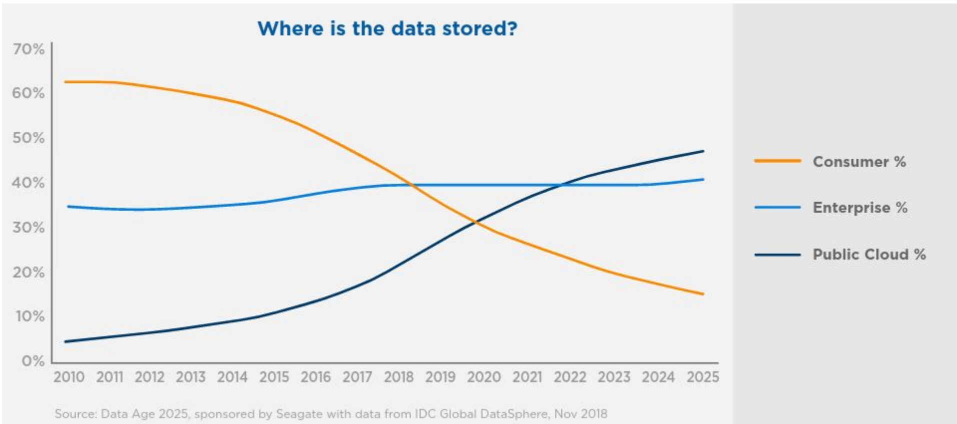
Wide Column Stores — Cassandra and HBase

Document databases — MongoDB

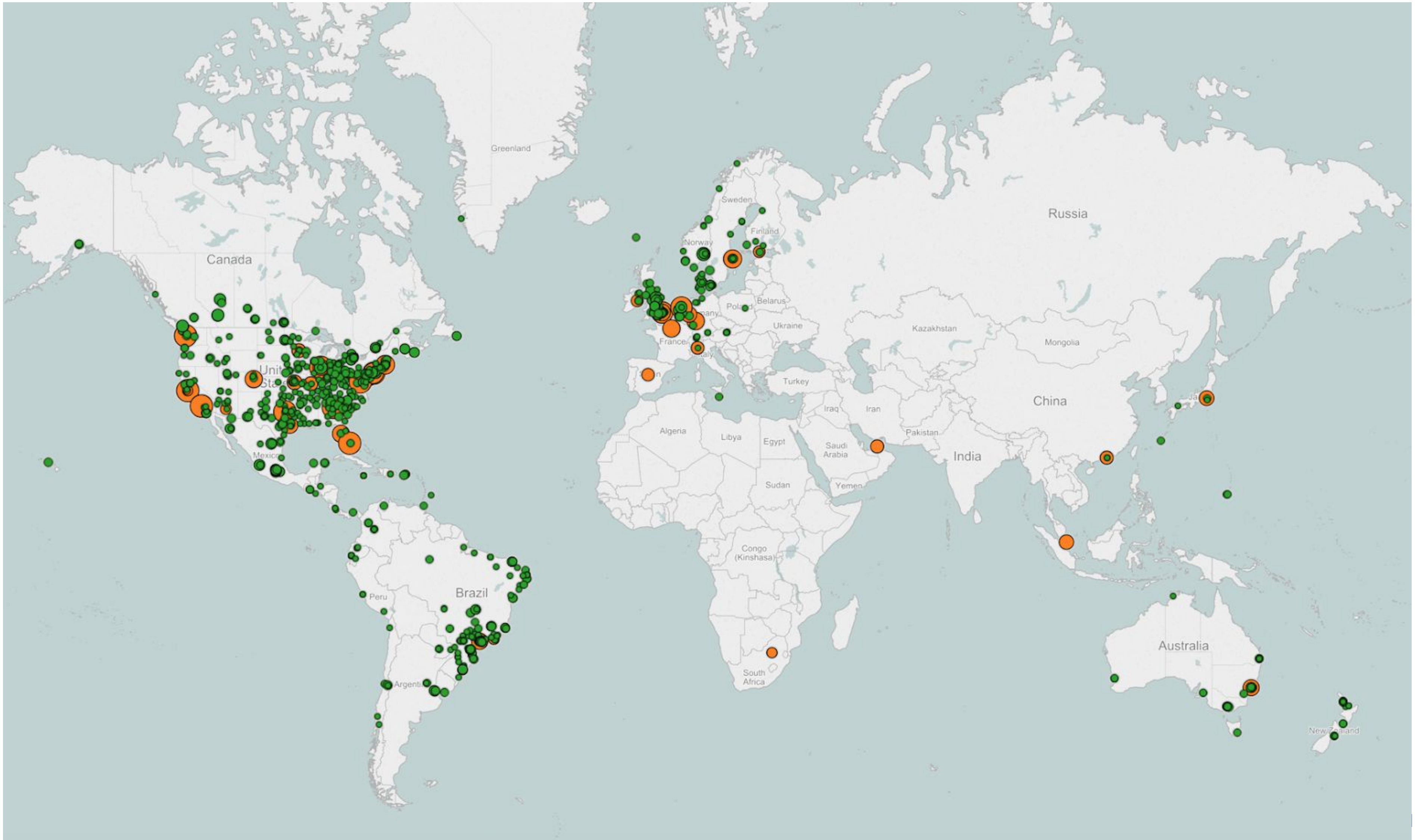
Graph databases — Neo4J and HyperGraphDB

Cloud computing dominance

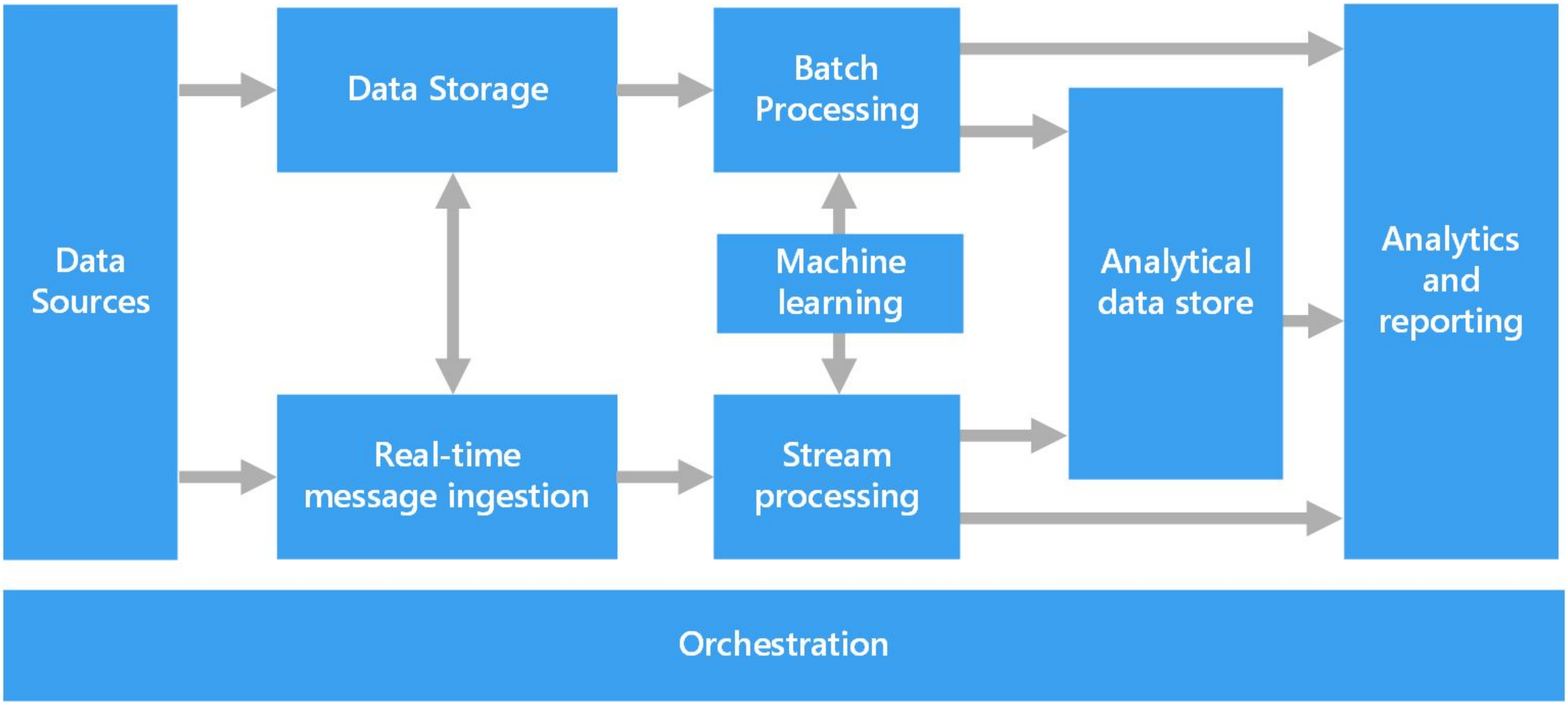




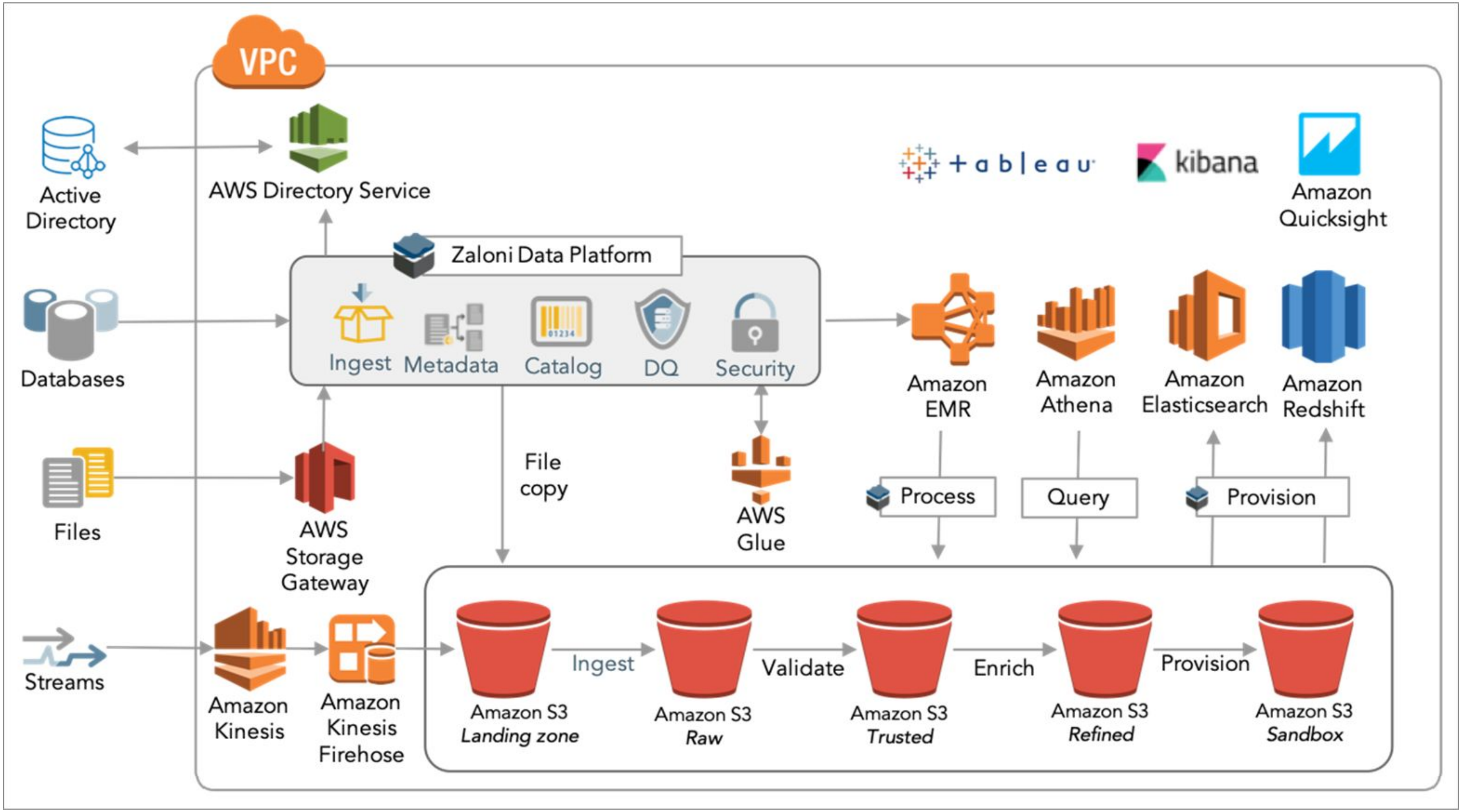
Distributed data: Netflix CDN



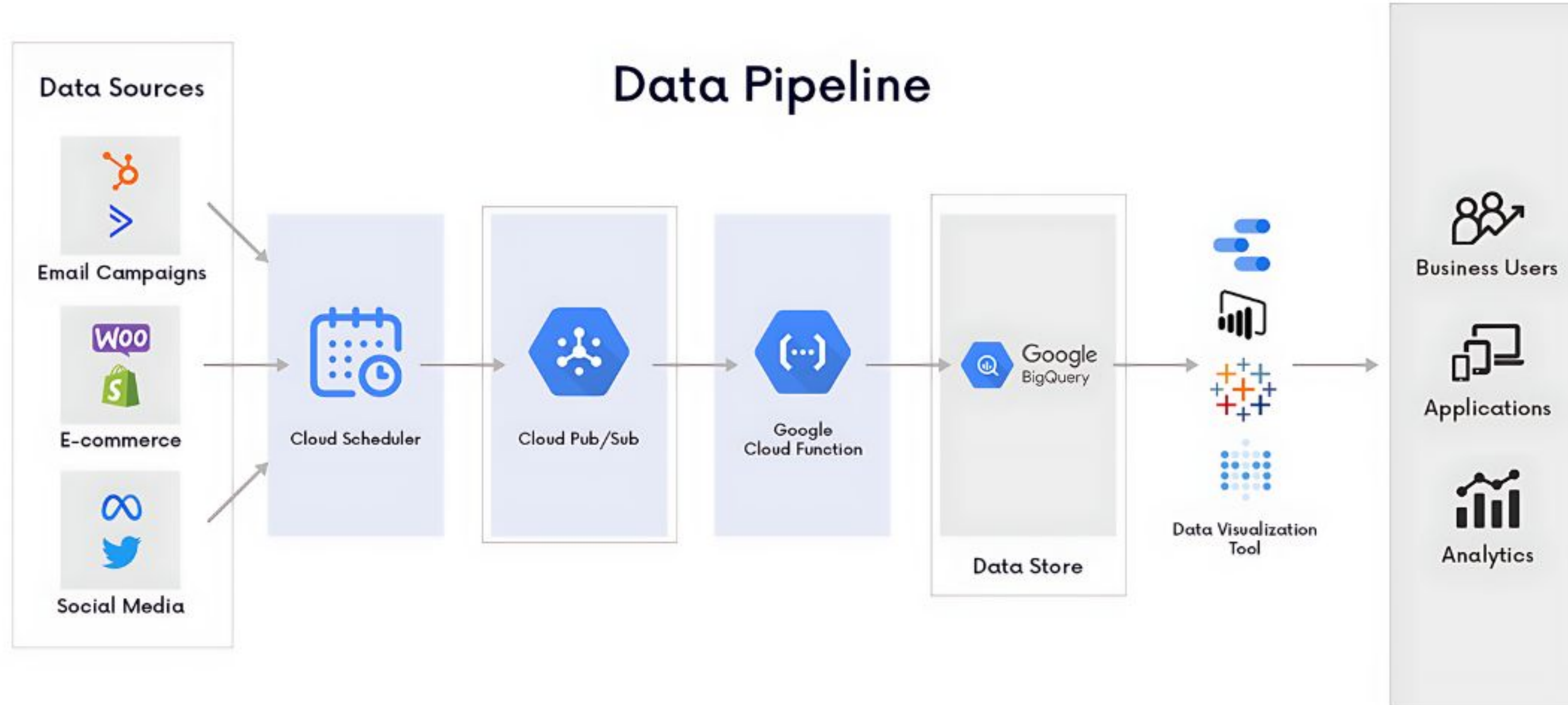
Modern data pipelines



Data pipelines in AWS



ETL on Google Cloud Platform

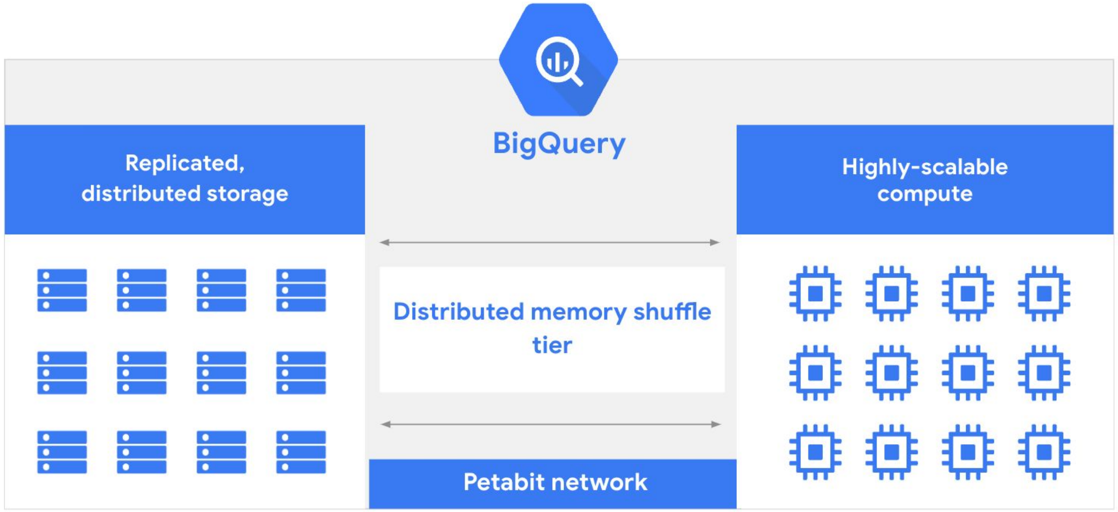


- Extract: Data comes from various sources (databases, APIs, files)
- Transform: Dataflow or BigQuery processes and cleans the data
- Load: Clean data goes into BigQuery for analysis
- Orchestrate: Composer schedules and monitors the entire process
- Build: Data Fusion provides a visual way to design these pipelines

GCP BigQuery

Role in ETL: The "Load" destination

- Stores large amounts of structured data in tables
- Allows fast SQL queries on billions of rows
- Can also perform transformations using SQL (ELT approach)



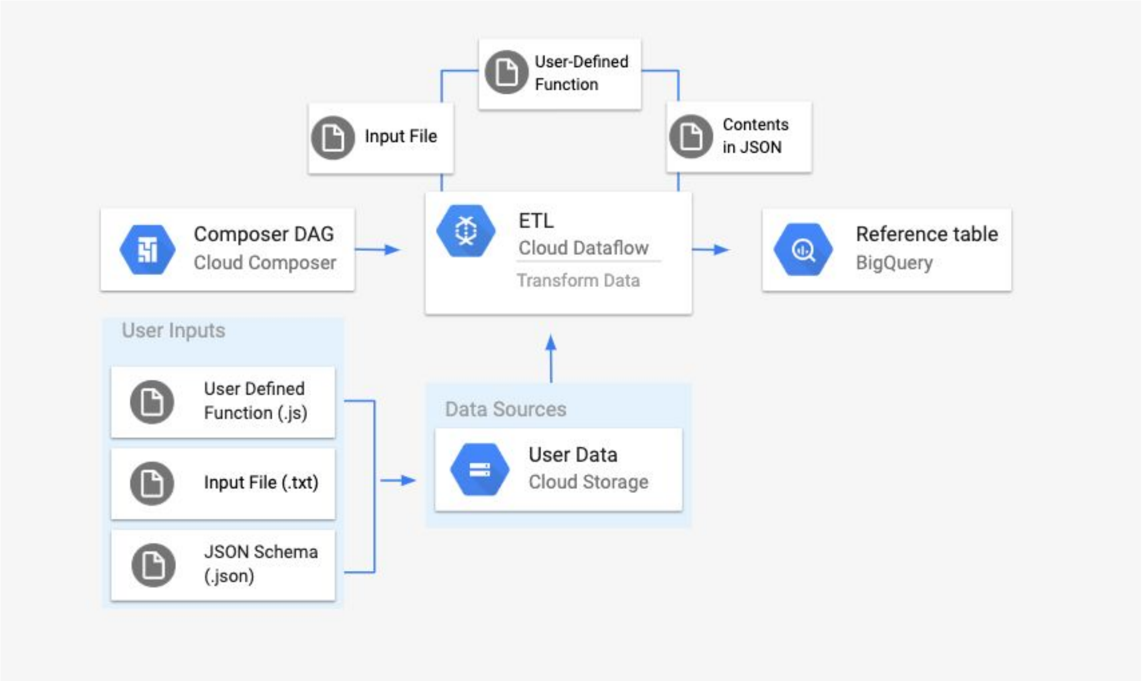
Example use: A retail company loads daily sales data into BigQuery tables, then runs SQL queries to calculate monthly revenue by region.

Best for: Analytics, reporting, data warehousing, and ML model training with SQL syntax

GCP Cloud Dataflow

Role in ETL: The "Transform" step

- Takes raw data and cleans, filters, or reshapes it
- Handles both real-time streaming data and large batch files
- Automatically scales up when processing large amounts of data



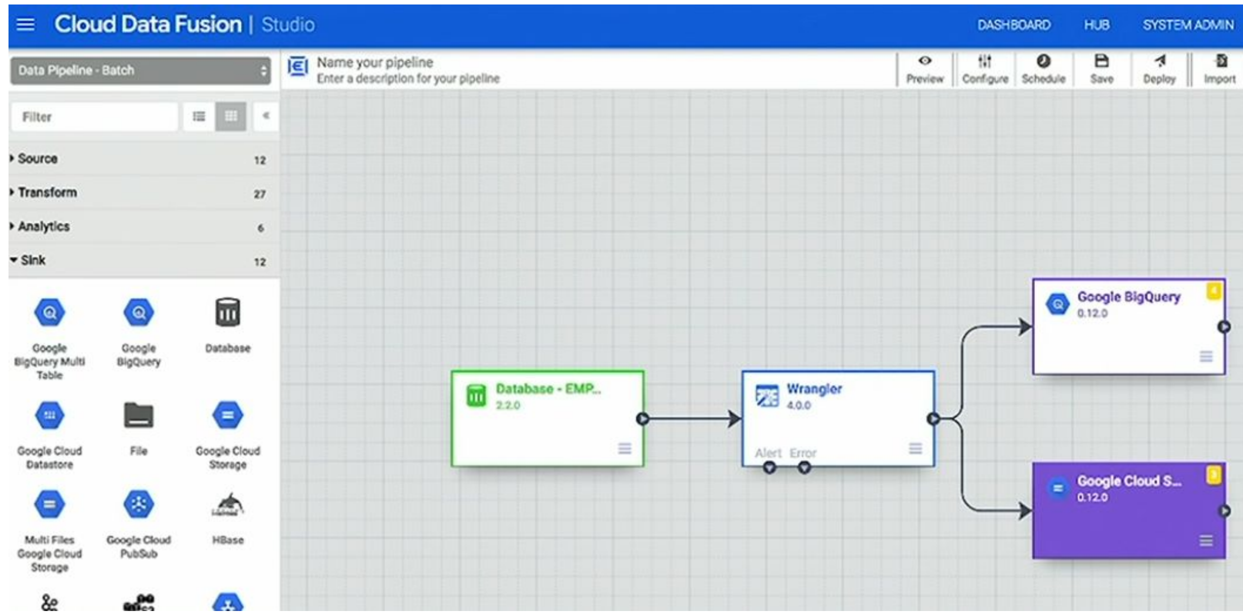
Example use: Processing millions of website clicks per hour, filtering out invalid clicks, and calculating user session statistics.

Best for: Real-time ETL, streaming analytics, fraud detection, and continuous data processing

GCP Cloud Data Fusion

Role in ETL: Drag-and-drop pipeline creation

- Build ETL pipelines by connecting visual blocks
- No programming required - point and click interface
- Connects different data sources and destinations easily



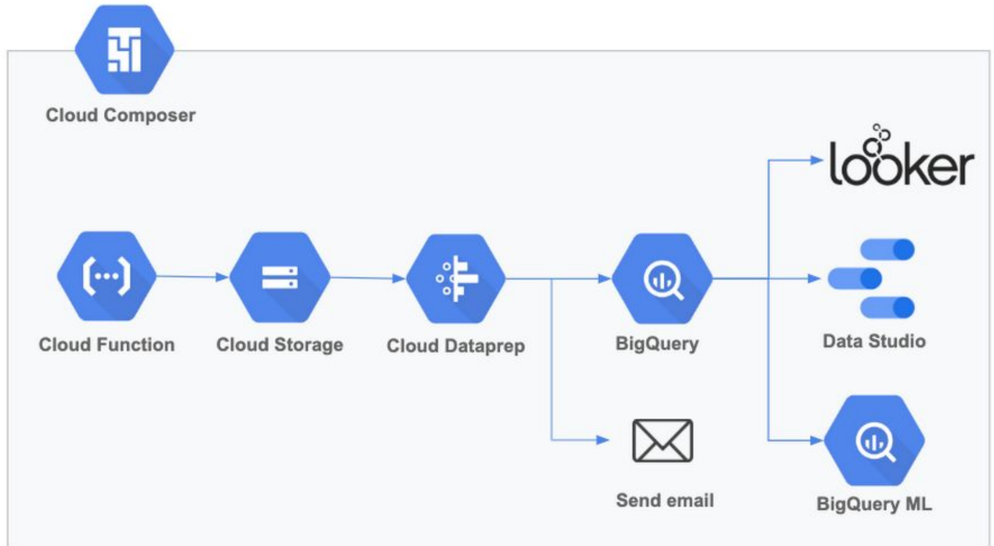
Example use: A marketing analyst connects customer data from a CRM system to sales data from a database, joins them together, and loads the result into BigQuery - all by dragging boxes and drawing connections.

Best for: Business users without programming experience, rapid pipeline prototyping, and hybrid cloud deployments

GCP Cloud Composer

Role in ETL: Orchestration and scheduling

- Coordinates multiple ETL steps in the right order
- Handles failures and retries automatically
- Schedules pipelines to run daily, weekly, or when triggered



Example use: Every night at 2 AM, extract data from three different databases, run transformations in Dataflow, load results into BigQuery, then send an email report - all coordinated automatically.

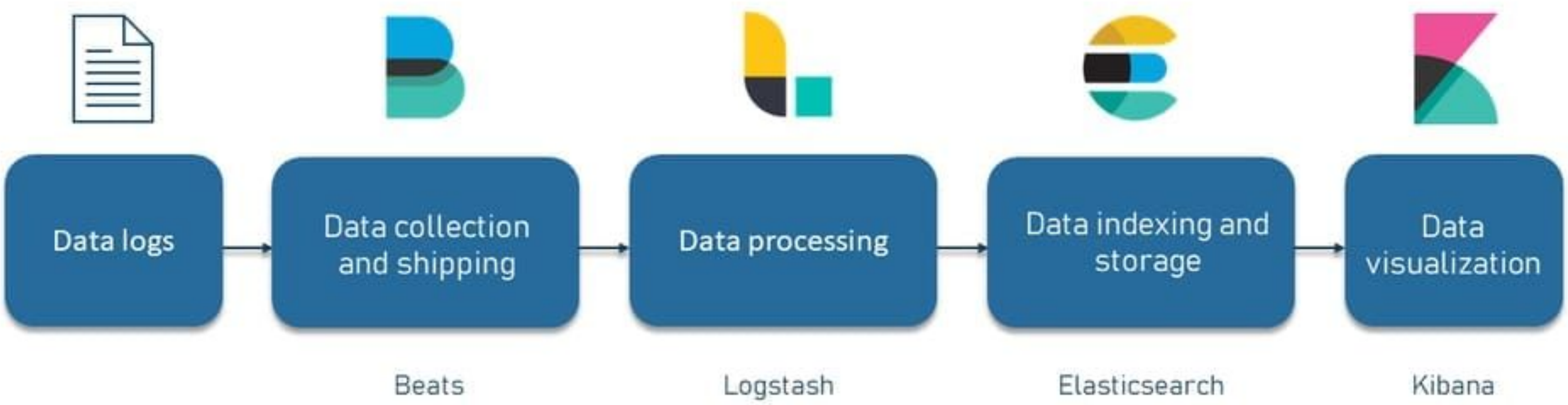
Best for: Complex workflow orchestration, scheduling dependencies between services, and managing enterprise-scale data pipelines

ETL for big data with Hadoop and Spark

- ETL tools that can handle massive amounts of data
- data from distributed systems
- Hadoop
  - distributed storage
- Apache Spark
  - in-memory processing
- allow ETL pipelines to scale horizontally
  - distributes data and the computational workload across many nodes

Loading with Elasticsearch

- a distributed search and analytics engine
- indexing structured and unstructured data for real-time analytics
- Use Cases: Real-time log analysis, full-text search, monitoring and alerting.
- ETL Role: "Load" destination in ETL workflows
- Advantages: Horizontal scalability, distributed architecture, and near-instant search results on large datasets.



Real-time ETL with Kafka and Spark Streaming

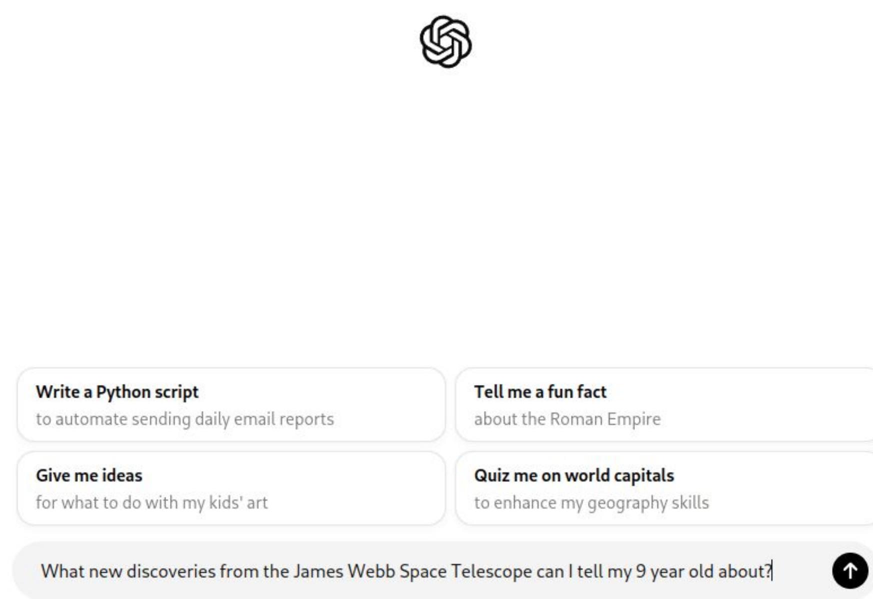
- Traditional ETL pipelines often work in batch mode
- Real-time ETL is becoming increasingly important
- real-time ETL pipelines: data is extracted, transformed, and loaded continuously as new data flows in
- Apache Kafka
  - is a distributed streaming platform that enables real-time data ingestion
- Apache Spark Streaming
  - allows for real-time data processing
- use cases
  - monitoring financial transactions
  - real-time analytics
  - IoT applications

ETL Use-Cases

- Airnity
- CNES
- LLM

ETL Use-Case: LLMs

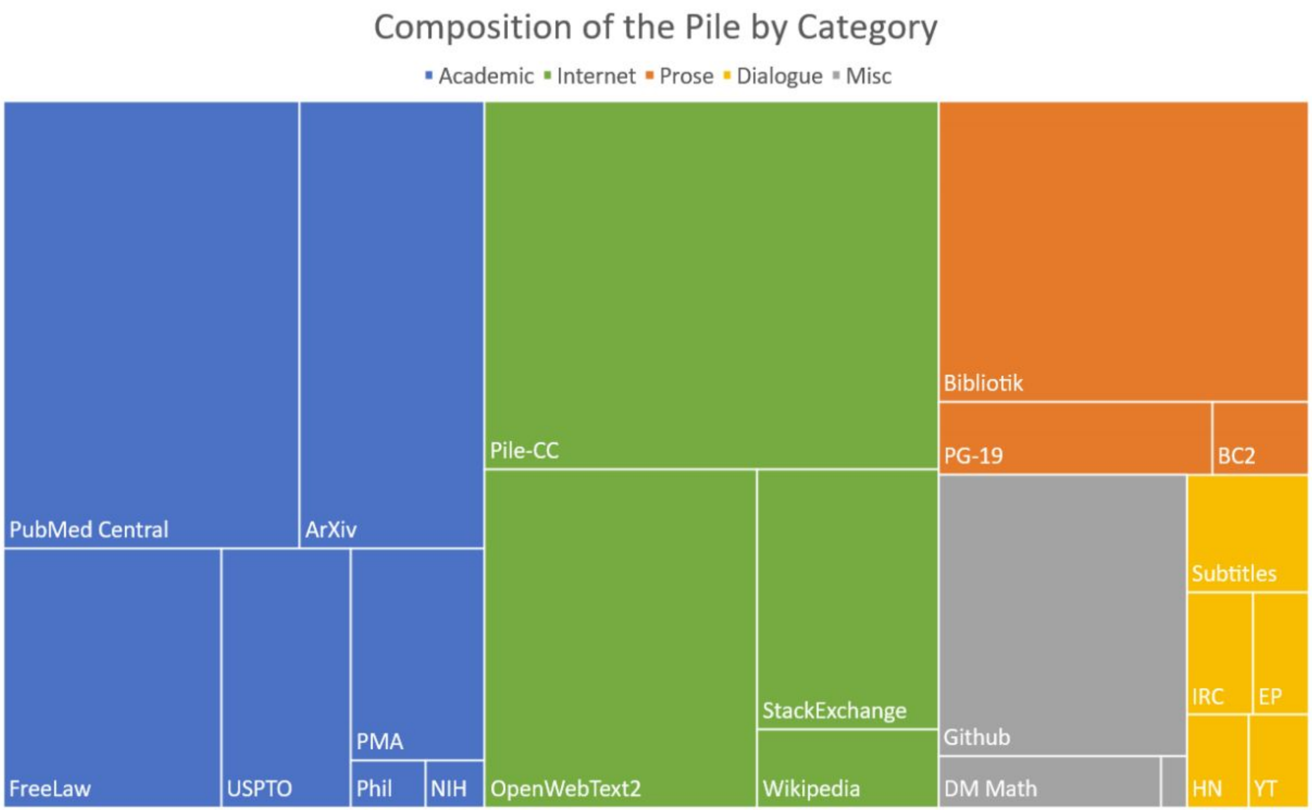
- Scale Challenge: LLMs require processing trillions of tokens from diverse data sources (web crawls, documents, enterprise data)
- Specialized Pipeline: Traditional ETL must adapt to handle unstructured text, massive compute requirements, and continuous model updates
- Modern Architecture: Combines data lakes, vector databases, distributed processing (Spark), and specialized ML infrastructure



LLM Extract - Data Sources & Collection

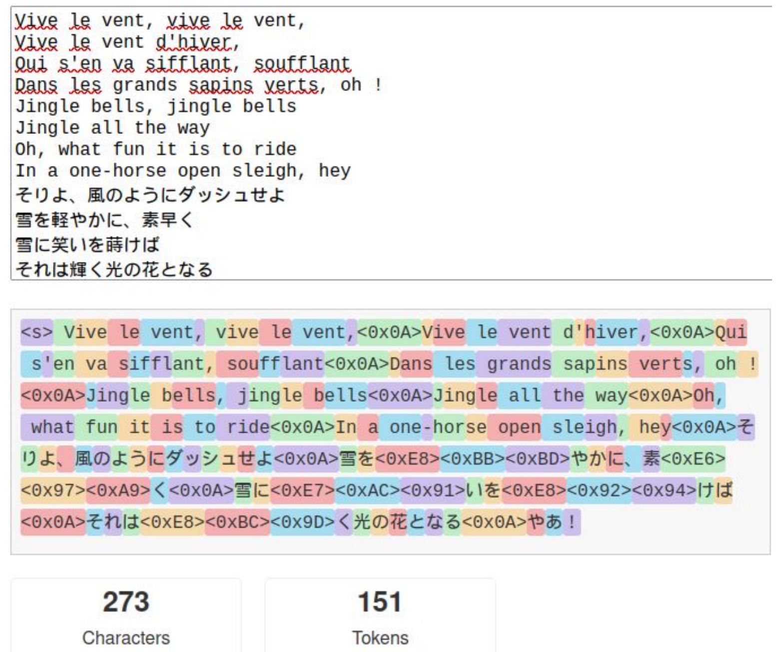
- Massive Web Data: Common Crawl provides terabytes of raw web data from billions of pages (used by GPT-3, LLaMA)
- Format Diversity: PDF, HTML, JSON, Office documents, plus enterprise sources like Slack, Salesforce, internal databases
- Volume Scale: Datasets contain 5+ trillion tokens requiring specialized connectors and parallel extraction methods
- Quality Challenges: Must handle duplicates, privacy concerns (PII), and varying data quality across sources

Dataset	Quantity (tokens)	Weight in training mix	Epochs elapsed when training for 300B tokens
Common Crawl (filtered)	410 billion	60%	0.44
WebText2	19 billion	22%	2.9
Books1	12 billion	8%	1.9
Books2	55 billion	8%	0.43
Wikipedia	3 billion	3%	3.4



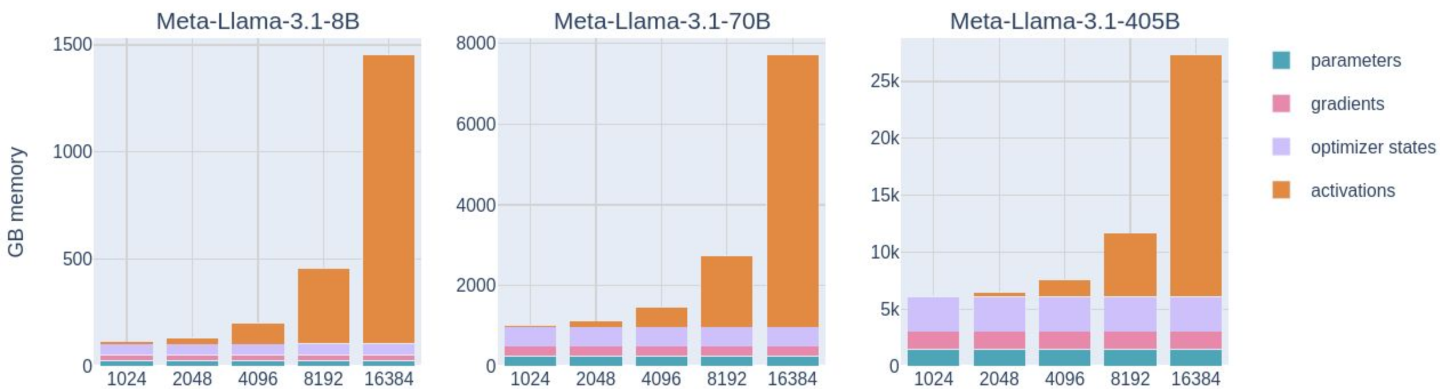
LLM Transform - Data Processing Pipeline

- Text Preprocessing: Normalization, deduplication (exact, fuzzy, semantic), and PII removal to ensure data quality and privacy
- Tokenization: Breaking text into subword tokens using specialized algorithms (SentencePiece, BPE) tailored to the training corpus
- Massive Reduction: Advanced preprocessing can reduce dataset size by 99% (2TB → 25GB) through efficient tokenization and filtering
- Distributed Processing: Apache Spark enables parallel processing across hundreds of nodes for handling trillion-token datasets



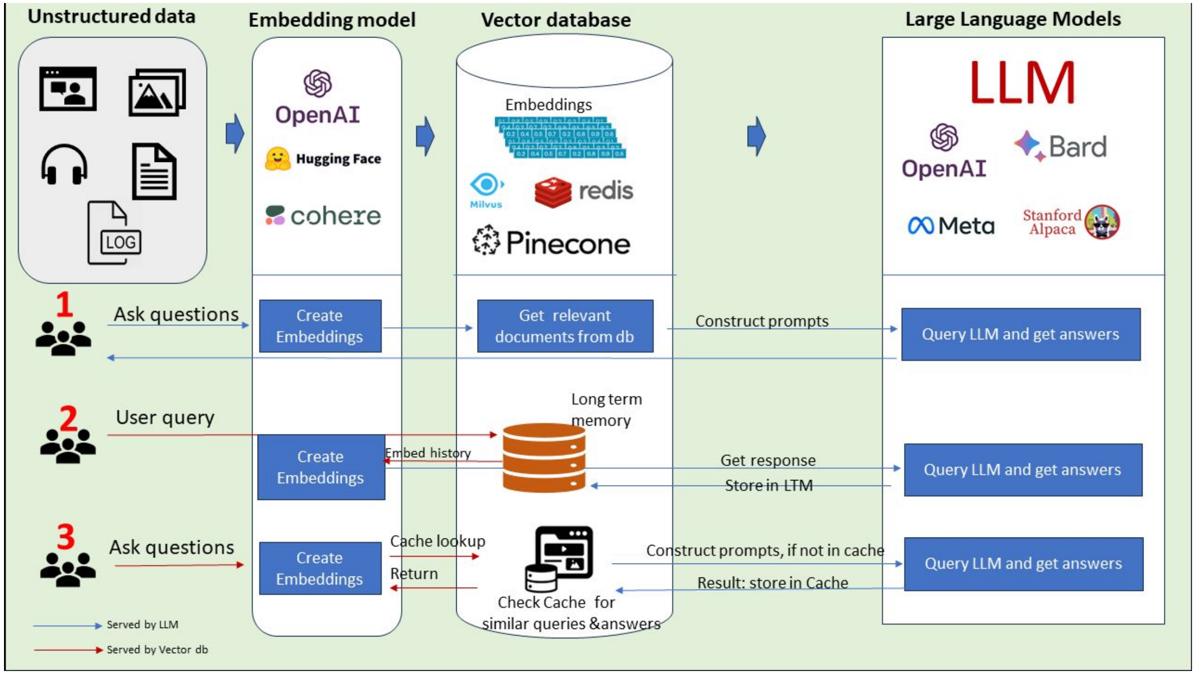
LLM Load - Training Infrastructure

- Distributed Storage: Data lakes (petabyte scale) with local node duplication to avoid network bottlenecks during training
- Specialized Hardware: GPU clusters (H100s) with high-memory capacity for processing large model parameters and batch sizes
- Model Artifacts: Centralized model registry for versioning, lineage tracking, and lifecycle management of trained models
- Cost Optimization: Parameter-efficient fine-tuning (LoRA, QLoRA) reduces storage and compute requirements



LLM Load - Deployment & Serving

- Vector Databases: Specialized storage (Pinecone, Weaviate) for embeddings enabling fast semantic search and RAG applications
- Model Serving: Production deployment with auto-scaling, load balancing, and monitoring for inference workloads
- Continuous Updates: MLOps platforms (MLflow, Databricks) orchestrate model retraining and deployment pipelines



Project: Design a data pipeline

- You are the 4 person data engineering team of a startup
- Startups to be defined in AI Business Models class
- 2 - 4 page report on ETL/ELT pipeline choices
  - Motivate and explain choices:
    - E: where are the data coming from?
    - T: how are the data being transformed?
    - L: how are the data loaded, stored, and used?
- Demo of example database
  - PostgreSQL, Mongo, other, your choice
  - Documented scripts to load and manipulate example data
  - Data should not be exhaustive, should demonstrate ETL choices

Report rigor	6
Report clarity	6
Demo data	4
Demo manipulation	4

Designing a data pipeline: extraction

- What are the various data sources?
- What is the format of the data from each source (e.g., CSV, JSON, XML, database tables)?
- Is the data streaming or can it be loaded in batches?
- What will the data look like on extraction?
- How do you verify the data's accuracy and completeness at the source?
- Are there any data access limitations or security constraints?
- How frequently is the data updated or changed at the source?
- Will you need to deal with incremental data extraction or full data loads?
- What are the volume and velocity of the data (e.g., terabytes per day, real-time streams)?

Designing a data pipeline: transformation

- What data cleansing steps are needed?
- Are there any business rules that need to be applied during transformation?
- Do you need to join or merge data from multiple sources?
- Are there any specific data formats or types that need to be converted?
- How will you handle any data inconsistencies or errors?
- Are there any dependencies between the transformation steps (e.g., one transformation requires another to be completed first)?
- Do you need to enrich the data by adding additional calculated fields?
- How will you track the changes to the data for auditing purposes?
- Will transformation happen before or after loading?

Designing a data pipeline: loading

- What is the target system for the data (e.g., data warehouse, data lake, database)?
- How often will the data be loaded (e.g., real-time, hourly, daily)?
- Should the data be appended to the existing dataset or replace it entirely?
- Are there any schema or structural requirements for the target system?
- How do you ensure data consistency and integrity during loading?
- What are the performance constraints?
- How will you handle schema changes in the target system?
- Is there a need for historical data tracking or versioning in the target?
- How will you monitor the loading process to ensure it runs successfully?

ETL or ELT

Data Format:

- ETL is needed for unstructured data to transform it into a relational format.
- ELT can be used if the data is already in a relational or flat format.

Data Size:

- ELT is suited for large datasets due to enhanced processing capabilities.
- ETL is typically used for smaller datasets.

Cost:

- ETL can be expensive as it involves physically moving data for processing.
- ELT can be more cost-effective, especially when leveraging cloud-based parallel processing without moving data.

Data Source:

- ELT is better for large, batch-wise data from cloud sources.
- ETL is preferred for streaming or messaging data.

Data Destination:

- ETL is often used when the source and destination systems differ.
- ELT is suitable when transferring data between identical systems or products.

Transformation Intensity:

- ELT is preferred for simple transformations and scalability.
- ETL is chosen for more complex transformations that need to be handled in smaller steps.

Transformation Location:

- In ETL, transformations occur before the data reaches the warehouse.
- In ELT, transformations are performed after data import, requiring more processing power but offering faster insights.

Airlife: Discover an Airplane’s History

- Airlife allows users to track the lifetime history of a specific airplane, including total distance flown, estimated carbon footprint, and its most recent location.
- Pull data from OpenFlights (flight routes, airports) and live flight tracking APIs (e.g., FlightAware, OpenSky Network).
- Load data into a database to allow quick queries about an airplane’s history. Visualize key statistics on a dashboard.
- Estimate carbon footprint using factors like aircraft type and distance flown.
- Handle frequent updates to reflect current location and flights.