Lesson 2.2: Data Pipelines

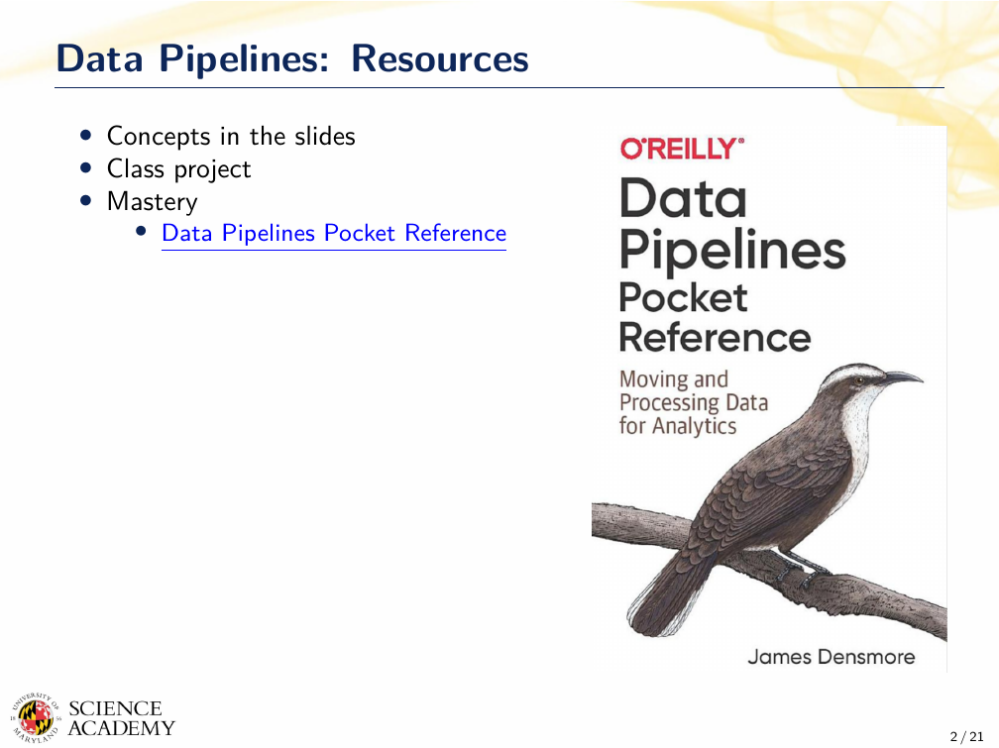
UMD DATA605 - Big Data Systems

Lesson 2.2: Data Pipelines

Instructor: Dr. GP Saggese, gsaggese@umd.edu

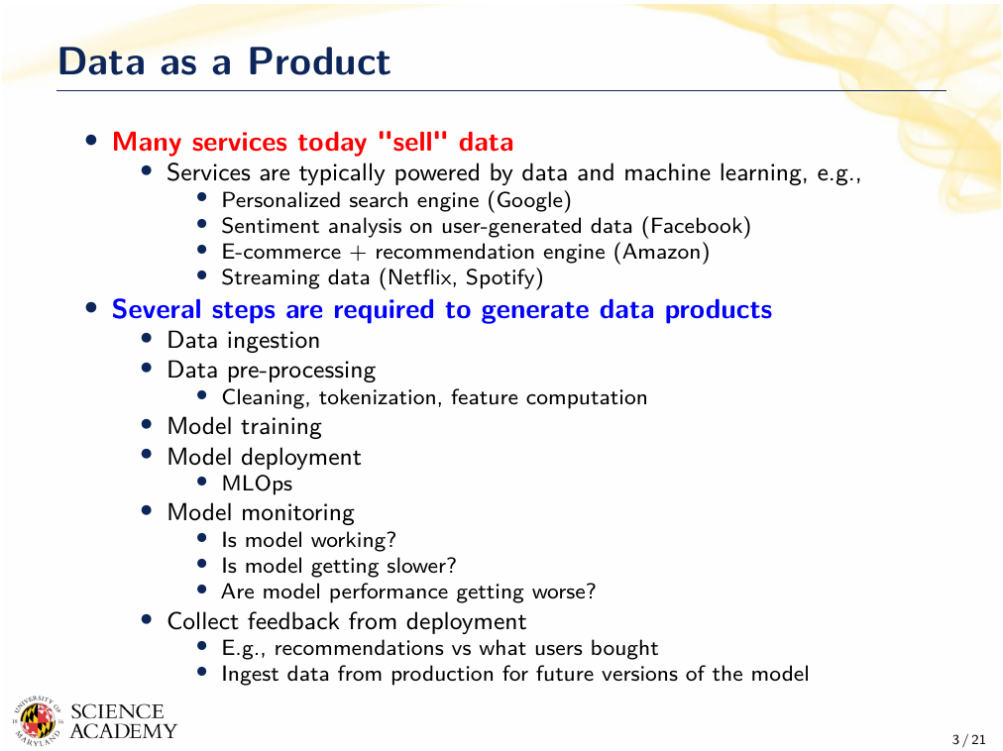
SCIENCE ACADEMY

2 / 21: Data Pipelines: Resources



- **Concepts in the slides**
  - This bullet point suggests that the slides themselves contain important concepts related to data pipelines. Data pipelines are essential in managing and processing large volumes of data efficiently. They automate the flow of data from one system to another, ensuring that data is clean, organized, and ready for analysis. Understanding these concepts is crucial for anyone working with big data or machine learning, as they form the backbone of data-driven decision-making.
- **Class project**
  - The mention of a class project indicates that students will have a practical opportunity to apply what they've learned about data pipelines. This hands-on experience is invaluable because it allows students to see how theoretical concepts are implemented in real-world scenarios. Working on a project helps solidify understanding and provides a chance to troubleshoot and solve problems that may arise in data pipeline creation and management.
- **Mastery**
  - The reference to the *Data Pipelines Pocket Reference* book suggests a resource for students to deepen their understanding and achieve mastery in the subject. This book likely offers detailed explanations, examples, and best practices for building and managing data pipelines. It's a useful tool for students who want to go beyond the basics and gain a comprehensive understanding of the topic.

3 / 21: Data as a Product



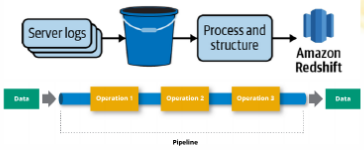
- **Many services today "sell" data**
  - In today's digital world, many services are built around the concept of using data to provide value. These services often rely heavily on data and machine learning to function effectively.
    - **Personalized search engine (Google):** Google uses data to tailor search results to individual users, making searches more relevant and efficient.
    - **Sentiment analysis on user-generated data (Facebook):** Facebook analyzes user posts and interactions to understand public sentiment and improve user experience.
    - **E-commerce + recommendation engine (Amazon):** Amazon uses data to recommend products to users based on their browsing and purchasing history, enhancing the shopping experience.
    - **Streaming data (Netflix, Spotify):** These platforms use data to suggest content that aligns with user preferences, keeping users engaged.
- **Several steps are required to generate data products**
  - Creating data-driven products involves multiple stages, each crucial for ensuring the final product is effective and reliable.
  - **Data ingestion:** This is the process of collecting raw data from various sources, which will be used for analysis and model training.
  - **Data pre-processing:** Before data can be used, it needs to be cleaned and organized. This includes tasks like removing errors, breaking down text into tokens, and computing features that will be useful for the model.
  - **Model training:** This step involves using the pre-processed data to train machine learning models, teaching them to make predictions or decisions.

- **Model deployment:** Once trained, models need to be deployed into a production environment where they can be used in real-time applications. This involves MLOps, which is the practice of managing the lifecycle of machine learning models.
- **Model monitoring:** After deployment, it's important to continuously monitor the model to ensure it is performing well. This includes checking if the model is functioning correctly, if it's slowing down, or if its performance is degrading over time.
- **Collect feedback from deployment:** Gathering feedback is essential for improving models. For example, comparing recommendations with actual user purchases helps refine future versions of the model. This feedback loop involves ingesting new data from the production environment to enhance the model's accuracy and relevance.

4 / 21: Data Pipelines

Data Pipelines

- *"Data is the new oil"* . . . but oil needs to be refined
- **Data pipelines**
  - Processes that move and transform data
  - **Goal**: derive new value from data through analytics, reporting, machine learning



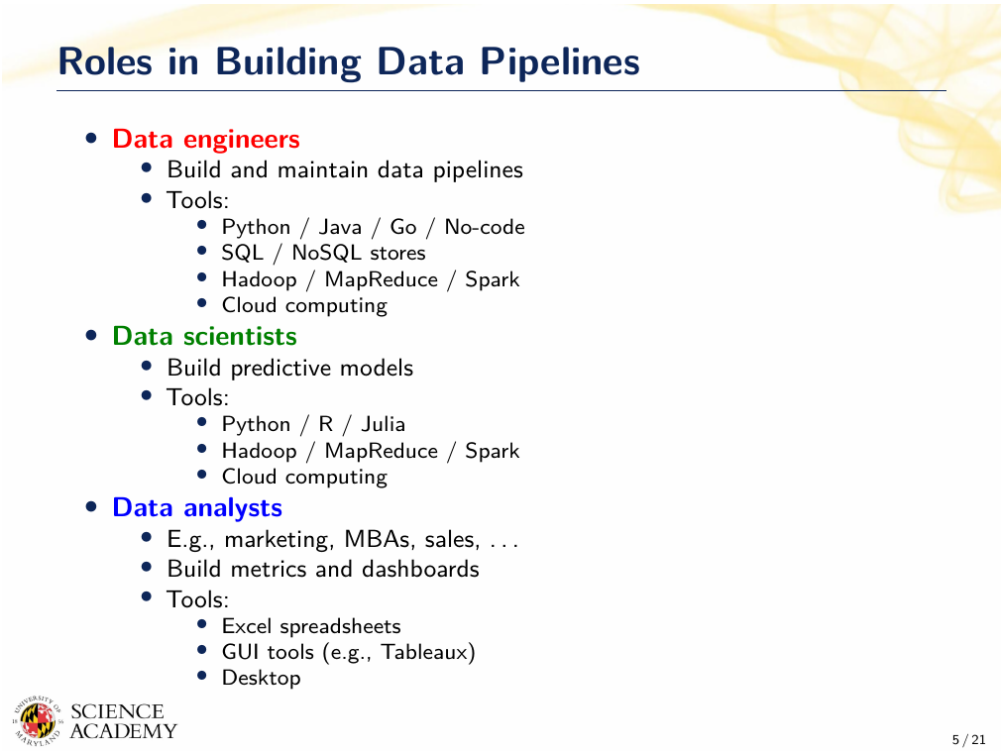
- **Data needs to be**:
  - Collected
  - Pre-processed / cleaned
  - Validated
  - Processed
  - Combined
- **Data ingestion**
  - Simplest data pipeline
  - Extract data (e.g., from REST API)
  - Load data into DB (e.g., SQL table)

SCIENCE ACADEMY

4 / 21

- **Data Pipelines**
  - Think of data pipelines as the systems that help us move and change data from one place to another. Just like oil needs to be refined to be useful, data needs to be processed to be valuable.
  - **Goal**: The main aim of data pipelines is to help us get new insights from data. This can be through analytics, creating reports, or using machine learning to make predictions or decisions.
- **Data needs to be**:
  - **Collected**: Gathering data from various sources, like sensors, databases, or user inputs.
  - **Pre-processed / cleaned**: Removing errors or inconsistencies in the data to make sure it's accurate and ready for analysis.
  - **Validated**: Checking that the data is correct and meets the required standards.
  - **Processed**: Transforming the data into a format that is useful for analysis or reporting.
  - **Combined**: Merging data from different sources to provide a complete picture.
- **Data Ingestion**
  - This is the simplest form of a data pipeline. It involves taking data from a source, like a REST API, and putting it into a database, such as a SQL table. This is the first step in making data ready for further processing and analysis.

5 / 21: Roles in Building Data Pipelines

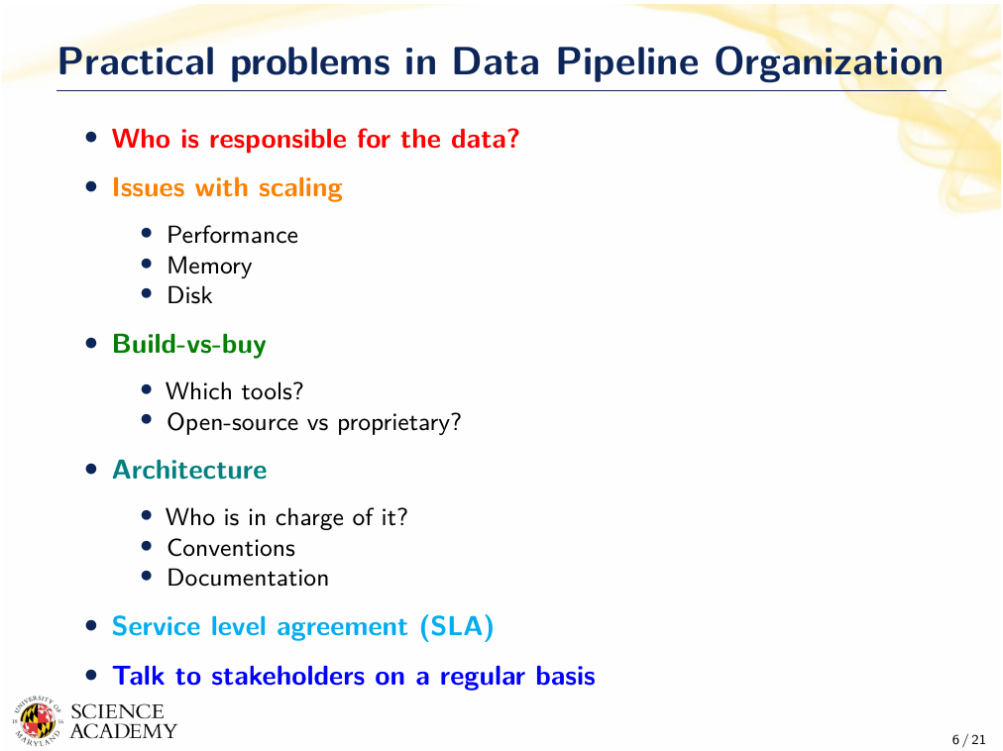


- **Data engineers**
  - *Role*: Data engineers are responsible for creating and maintaining the systems that allow data to be collected, stored, and processed efficiently. They ensure that data pipelines are robust and scalable, which is crucial for handling large volumes of data.
  - *Tools*:
    - **Programming Languages**: They often use languages like Python, Java, and Go for scripting and automation. These languages are versatile and widely used in the industry.
    - **Data Storage**: SQL and NoSQL databases are essential for storing structured and unstructured data, respectively.
    - **Big Data Frameworks**: Hadoop, MapReduce, and Spark are popular for processing large datasets. These frameworks allow for distributed computing, which is necessary for handling big data.
    - **Cloud Computing**: Cloud platforms provide scalable resources and services, making it easier to manage data infrastructure without the need for physical hardware.
- **Data scientists**
  - *Role*: Data scientists focus on analyzing data and building models that can predict future trends or behaviors. Their work often involves statistical analysis and machine learning.
  - *Tools*:
    - **Programming Languages**: Python, R, and Julia are commonly used for data analysis and modeling due to their rich libraries and ease of use.
    - **Big Data Frameworks**: Like data engineers, data scientists also use Hadoop, MapReduce, and Spark to handle large datasets.
    - **Cloud Computing**: Cloud services provide the computational power needed for

complex data analysis and model training.

- **Data analysts**
  - *Role*: Data analysts interpret data and create reports that help businesses make informed decisions. They often work closely with business teams to understand their data needs.
  - *Tools*:
    * **Excel**: A staple tool for data analysis, Excel is used for its simplicity and powerful data manipulation capabilities.
    * **GUI Tools**: Tools like Tableau allow analysts to create interactive dashboards and visualizations, making it easier to communicate insights.
    * **Desktop**: Many data analysis tasks are performed on desktop computers, using software that provides a user-friendly interface for data exploration.

6 / 21: Practical problems in Data Pipeline Organization



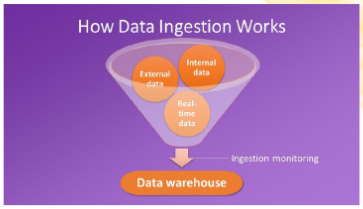
- **Who is responsible for the data?**
  - It's crucial to identify who owns the data within an organization. This person or team is responsible for ensuring data quality, security, and compliance. Without clear ownership, data management can become chaotic, leading to errors and inefficiencies.
- **Issues with scaling**
  - *Performance*: As data volume grows, systems may slow down. It's important to optimize processes to maintain speed.
  - *Memory*: Large datasets require more memory. Efficient data handling and processing techniques are necessary to prevent system overloads.
  - *Disk*: Storage capacity can become a bottleneck. Planning for scalable storage solutions is essential to accommodate growing data needs.
- **Build-vs-buy**
  - *Which tools?*: Deciding on the right tools is critical. Consider factors like ease of use, integration capabilities, and community support.
  - *Open-source vs proprietary?*: Open-source tools are often cost-effective and flexible, while proprietary solutions may offer better support and features. Weigh the pros and cons based on your organization's needs.
- **Architecture**
  - *Who is in charge of it?*: Assigning a person or team to oversee the architecture ensures consistency and alignment with business goals.
  - *Conventions*: Establishing standards and best practices helps maintain a coherent and efficient data pipeline.
  - *Documentation*: Proper documentation is vital for understanding and maintaining the data pipeline, especially as team members change.

- **Service level agreement (SLA)**
  - SLAs define the expected performance and availability of data services. They help set clear expectations and accountability between data providers and users.
- **Talk to stakeholders on a regular basis**
  - Regular communication with stakeholders ensures that the data pipeline aligns with business objectives and adapts to changing needs. It also helps in identifying potential issues early and fosters collaboration.

7 / 21: Data Ingestion

Data Ingestion

- **Data ingestion**
  - = extract data from one source and load it into another store
- **Data sources / sinks**
  - DBs
    - E.g., Postgres, MongoDB
  - REST API
    - Abstraction layer on top of DBs
  - Network file system / cloud
    - E.g., CSV files, Parquet files
  - Data warehouses
  - Data lakes
- **Source ownership**
  - An organization can use 10-1000s of data sources
  - Internal
    - E.g., DB storing shopping carts for a e-commerce site
  - 3rd-parties
    - E.g., Google analytics tracking website usage



SCIENCE ACADEMY

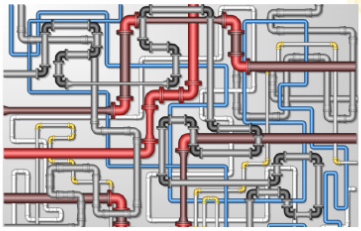
7 / 21

- **Data ingestion**
  - *Data ingestion* is the process of taking data from one place and moving it to another. This is a crucial step in data processing because it allows data to be collected from various sources and stored in a centralized location where it can be analyzed or used for other purposes.
- **Data sources / sinks**
  - **DBs**: Databases like Postgres and MongoDB are common sources and destinations for data. They store structured data and are often used in applications that require quick access to data.
  - **REST API**: This is a way to access data over the internet. It acts as a bridge between databases and applications, allowing data to be retrieved or sent using HTTP requests.
  - **Network file system / cloud**: Data can also be stored in files, such as CSV or Parquet files, which can be located on a network or in the cloud. These files are often used for large datasets that need to be processed in bulk.
  - **Data warehouses**: These are large storage systems designed to hold vast amounts of data from different sources, optimized for query and analysis.
  - **Data lakes**: Unlike data warehouses, data lakes store raw data in its native format, which can include structured, semi-structured, and unstructured data.
- **Source ownership**
  - Organizations often deal with a wide range of data sources, sometimes numbering in the thousands. These sources can be internal, such as a database that keeps track of shopping carts for an e-commerce site, or external, like third-party services such as Google Analytics, which provides insights into website usage. Understanding who owns the data source is important for managing access and ensuring data quality.

8 / 21: Data Pipeline Paradigms

Data Pipeline Paradigms

- There are **several styles of building data pipelines**
- **Multiple phases**
  - Extract
  - Load
  - Transform
- **Phases arranged in different ways** depending on philosophy about data / roles
  - ETL
  - ELT
  - EtLT

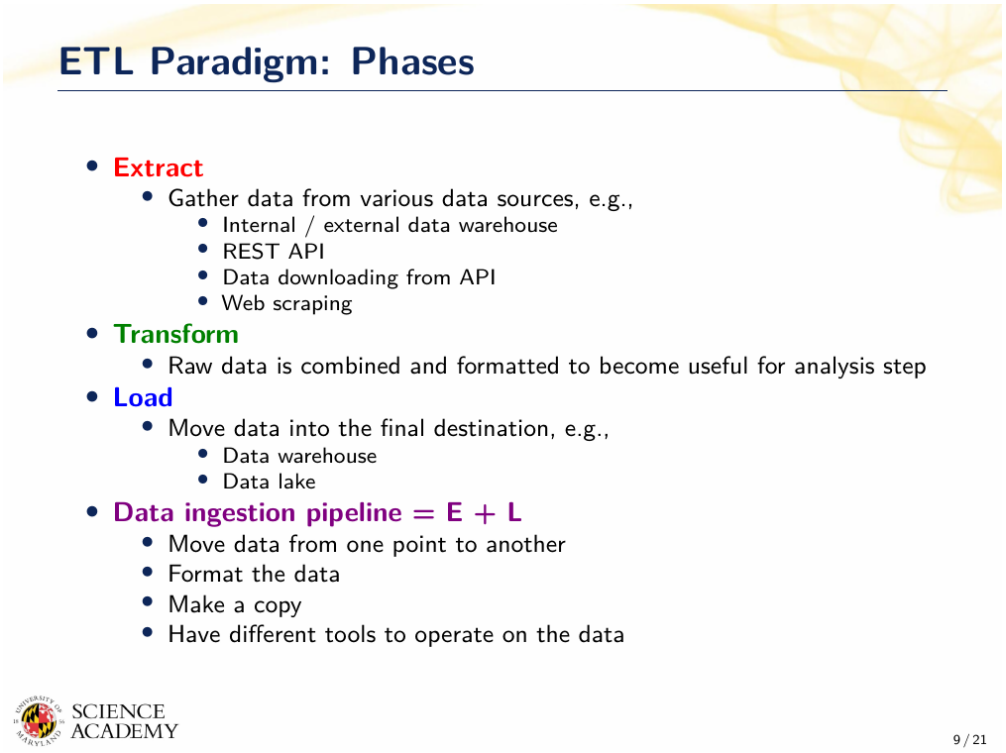


SCIENCE ACADEMY

8 / 21

- **Several styles of building data pipelines**: In the world of data processing, a data pipeline is a series of steps that data goes through from its raw form to a usable format. There are different ways to design these pipelines, and each style has its own advantages and use cases. Understanding these styles helps in choosing the right approach for specific data needs.
- **Multiple phases**: Data pipelines typically involve three main phases:
  - **Extract**: This is the first step where data is collected from various sources. These sources can be databases, APIs, or even files.
  - **Load**: After extraction, the data is loaded into a storage system. This could be a data warehouse, a data lake, or any other storage solution.
  - **Transform**: In this phase, the data is cleaned, organized, and converted into a format that is suitable for analysis or further processing.
- **Phases arranged in different ways**: The order and emphasis of these phases can vary based on the philosophy or the roles involved in the data processing:
  - **ETL (Extract, Transform, Load)**: This traditional approach involves transforming data before loading it into the storage system. It's useful when data needs to be cleaned and structured before storage.
  - **ELT (Extract, Load, Transform)**: In this modern approach, data is loaded in its raw form and transformed later. This is beneficial when dealing with large volumes of data, as it allows for more flexible and scalable processing.
  - **EtLT (Extract, then Load and Transform)**: This is a hybrid approach where some transformation occurs before loading, and additional transformations happen afterward. It combines the benefits of both ETL and ELT, providing a balanced approach.

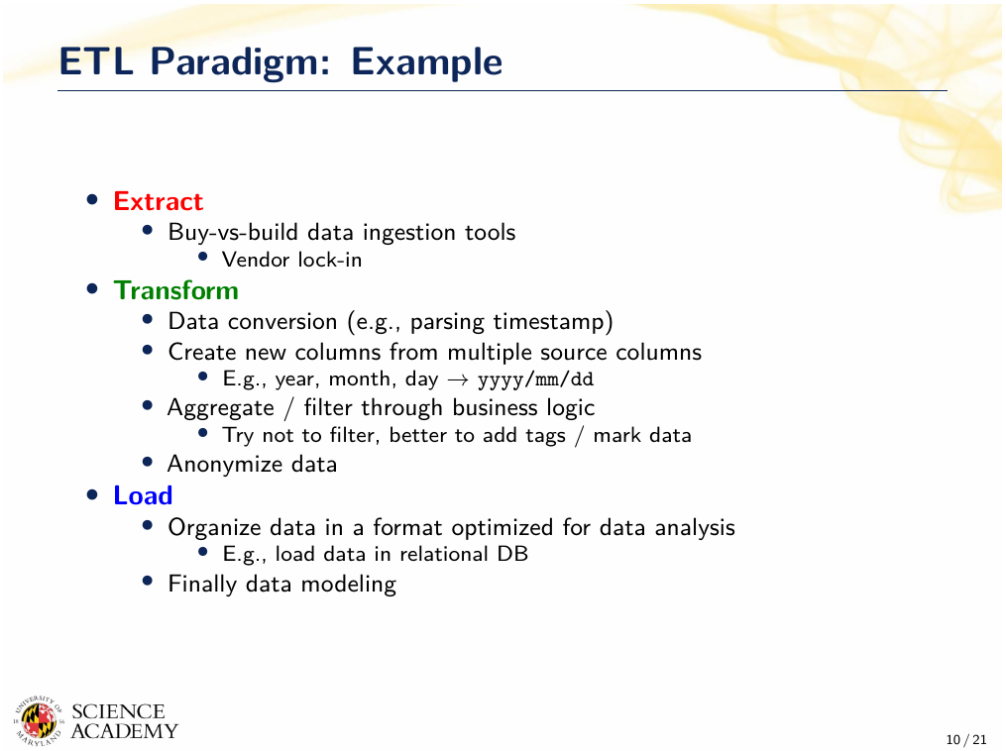
9 / 21: ETL Paradigm: Phases



- **Extract**
  - The first phase of the ETL process involves gathering data from various sources. This is crucial because data can reside in multiple places, such as internal or external data warehouses, which are large storage systems for data. Additionally, data can be collected from REST APIs, which are interfaces that allow different software applications to communicate with each other. Data can also be downloaded directly from APIs or collected through web scraping, which involves extracting data from websites. The goal here is to gather all the necessary data that will be used in later stages.
- **Transform**
  - Once the data is extracted, it needs to be transformed. This means taking the raw data and combining it in a way that makes it useful for analysis. This step often involves cleaning the data, removing duplicates, and converting it into a format that can be easily analyzed. The transformation phase is essential because raw data is often messy and not immediately useful for making decisions or gaining insights.
- **Load**
  - After transforming the data, the next step is to load it into its final destination. This could be a data warehouse, which is used for storing large amounts of structured data, or a data lake, which can store both structured and unstructured data. The loading phase ensures that the data is stored in a place where it can be accessed and analyzed by data analysts and other stakeholders.
- **Data ingestion pipeline = E + L**
  - The data ingestion pipeline focuses on the extraction and loading phases of the ETL process. It involves moving data from one point to another, ensuring that it is formatted correctly, and often involves making a copy of the data. Different tools are used to

operate on the data during this process, ensuring that it is efficiently and accurately transferred to its destination. This pipeline is crucial for maintaining a steady flow of data into systems where it can be used for analysis and decision-making.

10 / 21: ETL Paradigm: Example

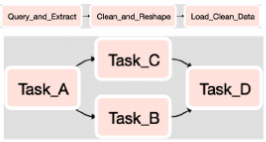


- **ETL Paradigm: Example**
  - **Extract**
    - This step involves gathering data from various sources. You can either buy or build your own data ingestion tools. *Buying* tools can be quicker and easier but may lead to **vendor lock-in**, meaning you become dependent on a specific vendor's technology and may face challenges if you want to switch tools later.
  - **Transform**
    - This is where the raw data is cleaned and converted into a usable format. For example, you might need to convert timestamps into a standard format.
    - You can also create new columns by combining data from multiple sources, such as converting separate year, month, and day columns into a single `yyyy/mm/dd` format.
    - Aggregating or filtering data is done based on business logic. However, it's often better to tag or mark data instead of filtering it out, as this retains more information for future analysis.
    - Anonymizing data is crucial for privacy, ensuring that personal information is protected during analysis.
  - **Load**
    - The final step is to load the transformed data into a system where it can be easily accessed and analyzed. This often involves organizing the data in a format optimized for analysis, such as a relational database.
    - After loading, data modeling can be performed to structure the data in a way that supports business needs and decision-making processes.

11 / 21: Workflow Orchestration

Workflow Orchestration

- Companies have **many data pipelines** (10-1000s)
- **Orchestration tools**, e.g.,
  - Apache Airflow (from AirBnB)
  - Luigi (from Spotify)
  - AWS Glue
  - Kubeflow
- **Schedule and manage** flow of tasks according to their dependencies
  - Pipeline and jobs are represented through DAGs
- Monitor, retry, and send alarms

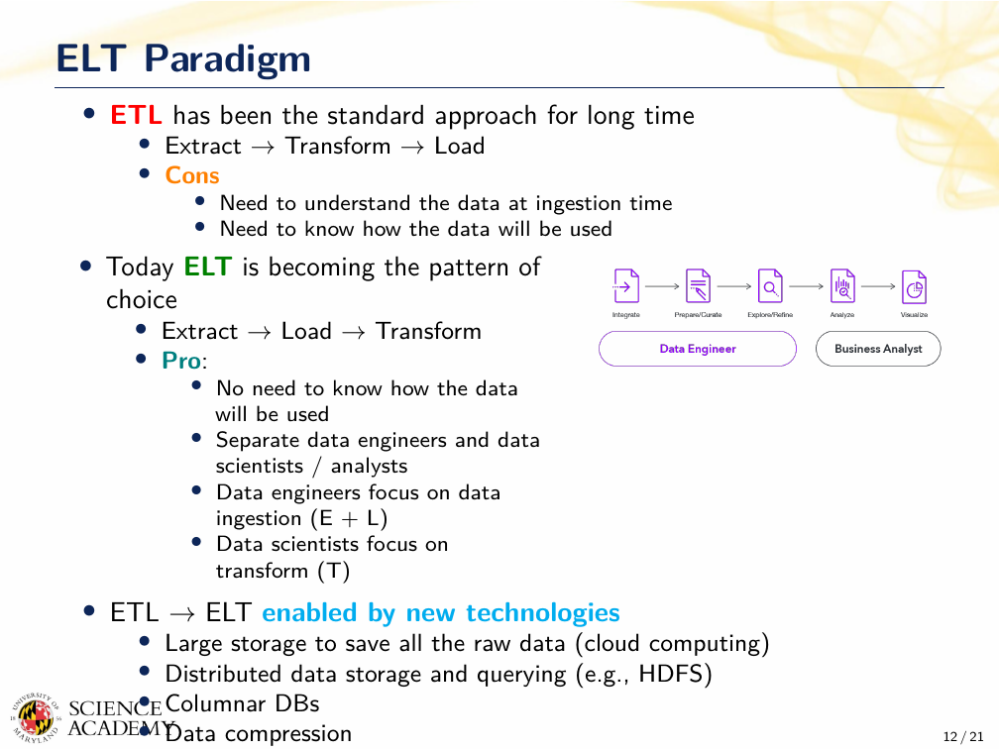


SCIENCE ACADEMY

11 / 21

- **Workflow Orchestration** is crucial for managing complex data processes within companies. Many organizations have **many data pipelines**—ranging from tens to thousands—that need to be efficiently managed and executed. These pipelines are sequences of data processing steps that transform raw data into valuable insights.
- **Orchestration tools** are software solutions that help automate, schedule, and manage these data pipelines. Some popular tools include:
  - **Apache Airflow**: Developed by AirBnB, it's widely used for its flexibility and scalability.
  - **Luigi**: Created by Spotify, it's known for its simplicity and ease of use.
  - **AWS Glue**: A fully managed service by Amazon Web Services that simplifies the process of building data pipelines.
  - **Kubeflow**: Designed for Kubernetes, it focuses on machine learning workflows.
- These tools help **schedule and manage** the flow of tasks by understanding their dependencies. They use Directed Acyclic Graphs (DAGs) to represent pipelines and jobs, ensuring tasks are executed in the correct order.
- Additionally, orchestration tools can **monitor** the execution of tasks, **retry** failed tasks, and **send alarms** to notify users of any issues. This ensures that data processes run smoothly and any problems are quickly addressed.

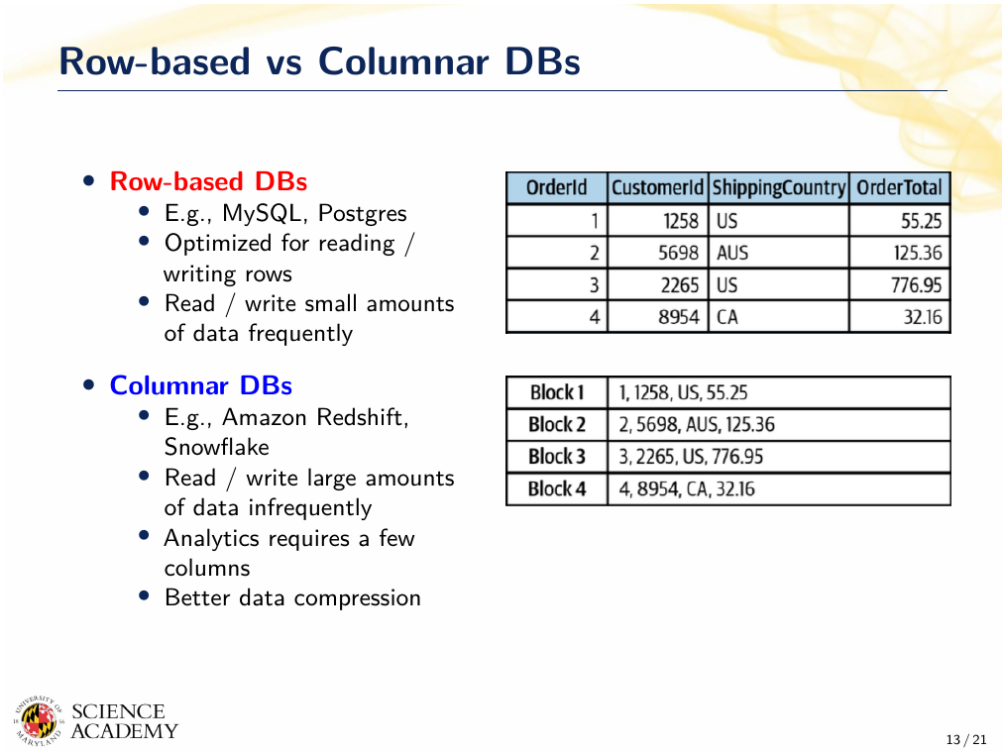
12 / 21: ELT Paradigm



- **ETL** has been the standard approach for a long time
  - *Extract → Transform → Load*: This traditional method involves extracting data from various sources, transforming it into a suitable format, and then loading it into a data warehouse or database.
  - **Cons**:
    - *Need to understand the data at ingestion time*: This means that before you even start processing the data, you need to have a clear understanding of its structure and how it will be used, which can be limiting.
    - *Need to know how the data will be used*: This requires predicting future data needs, which can be challenging and may lead to inefficiencies if the data usage changes.
- Today **ELT** is becoming the pattern of choice
  - *Extract → Load → Transform*: This modern approach involves loading raw data into a storage system first and then transforming it as needed.
  - **Pro**:
    - *No need to know how the data will be used*: This flexibility allows for data to be stored in its raw form and transformed later based on actual needs.
    - *Separate data engineers and data scientists/analysts*: This division of labor allows data engineers to focus on the technical aspects of data ingestion (E + L), while data scientists can concentrate on analyzing and transforming the data (T).
- ETL → ELT **enabled by new technologies**
  - *Large storage to save all the raw data (cloud computing)*: Cloud services provide scalable storage solutions that make it feasible to store vast amounts of raw data.
  - *Distributed data storage and querying (e.g., HDFS)*: Technologies like Hadoop Distributed File System (HDFS) allow for efficient storage and retrieval of large datasets

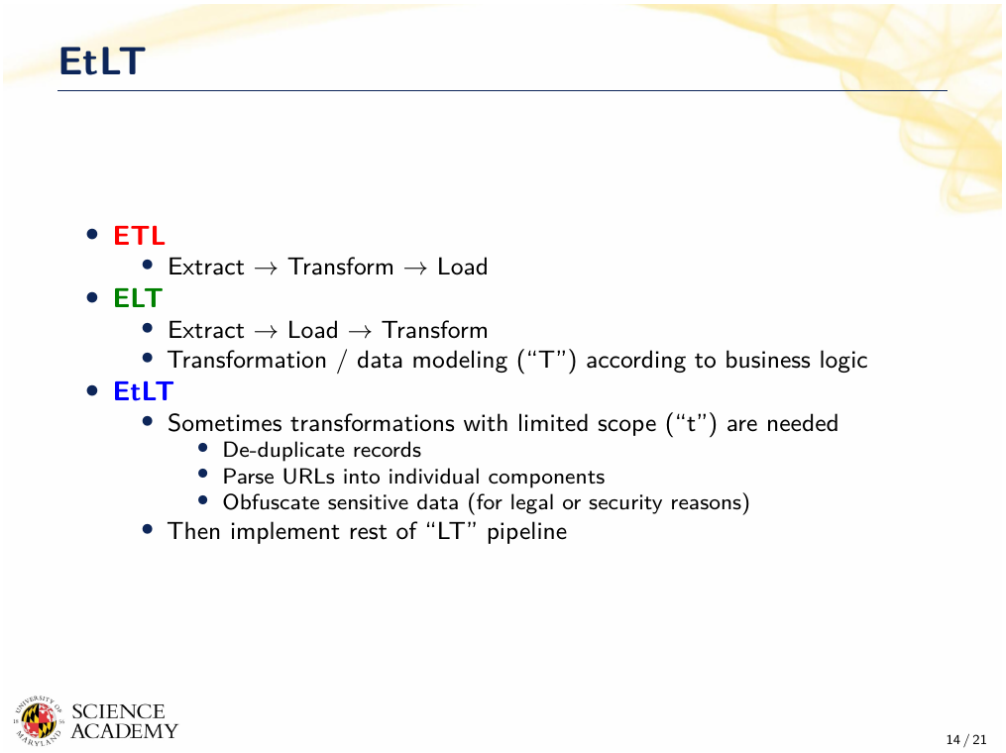
across multiple machines.
- *Columnar DBs*: These databases store data in columns rather than rows, optimizing them for read-heavy operations typical in analytics.
- *Data compression*: This reduces the storage footprint and speeds up data processing by minimizing the amount of data that needs to be read from disk.

13 / 21: Row-based vs Columnar DBs



- **Row-based DBs**
  - *Examples include* MySQL and Postgres. These are traditional databases that store data in rows. This means that all the data for a single record is stored together.
  - They are *optimized for reading and writing rows*, which makes them ideal for applications where you need to access complete records frequently, such as transactional systems.
  - These databases are best suited for scenarios where you need to read or write small amounts of data frequently. This is because accessing a full row is efficient, but reading specific columns from many rows can be slower.
- **Columnar DBs**
  - *Examples include* Amazon Redshift and Snowflake. These databases store data in columns rather than rows, which means that all the data for a single column is stored together.
  - They are designed to *read and write large amounts of data infrequently*, making them ideal for analytical queries where you need to process large datasets.
  - In analytics, you often need only a few columns from a large dataset. Columnar databases excel in these scenarios because they can quickly access the needed columns without reading entire rows.
  - They also offer *better data compression* because similar data types are stored together, which can significantly reduce storage requirements and improve query performance.

14 / 21: EtLT



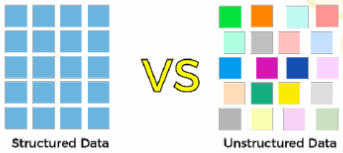
- **ETL**
  - **Extract → Transform → Load**: This is a traditional data processing approach where data is first extracted from various sources. After extraction, the data is transformed into a suitable format or structure, often involving cleaning, aggregating, or enriching the data. Finally, the transformed data is loaded into a target system, such as a data warehouse, where it can be used for analysis or reporting.
- **ELT**
  - **Extract → Load → Transform**: In this approach, data is extracted and immediately loaded into a storage system, like a data lake or cloud storage. The transformation occurs after loading, allowing for more flexibility and scalability. This method is particularly useful when dealing with large volumes of data, as it leverages the processing power of modern storage systems to perform transformations according to specific business logic.
- **EtLT**
  - **Sometimes transformations with limited scope ("t") are needed**: In some cases, minor transformations are necessary before loading data. These transformations might include:
    - **De-duplicate records**: Removing duplicate entries to ensure data quality.
    - **Parse URLs into individual components**: Breaking down URLs into parts like protocol, domain, and path for easier analysis.
    - **Obfuscate sensitive data**: Masking or encrypting sensitive information to comply with legal or security requirements.
  - **Then implement rest of "LT" pipeline**: After these initial transformations, the data is loaded, and further transformations are applied as needed, following the ELT process. This hybrid approach balances the need for immediate data cleaning with the flexibility

of post-load transformations.

15 / 21: Structure in Data (or Lack Thereof)

Structure in Data (or Lack Thereof)

- **Structured data**: there is a schema
  - Relational DB
  - CSV
  - DataFrame
  - Parquet
- **Semi-structured**: subsets of data have different schema
  - Logs
  - HTML pages
  - XML
  - Nested JSON
  - NoSQL data
- **Unstructured**: no schema
  - Text
  - Pictures
  - Movies
  - Blobs of data



SCIENCE ACADEMY

15 / 21

- **Structured data**: This type of data is organized in a highly predictable way, often following a specific schema or format.
  - *Relational DB*: Databases that store data in tables with rows and columns, like a spreadsheet. Each table has a defined structure.
  - *CSV*: Stands for Comma-Separated Values, a simple file format used to store tabular data, where each line is a data record.
  - *DataFrame*: A data structure used in programming languages like Python (Pandas) and R, which organizes data into a 2D table.
  - *Parquet*: A columnar storage file format optimized for use with big data processing frameworks like Apache Hadoop and Apache Spark.
- **Semi-structured**: This data type doesn't fit neatly into tables but still has some organizational properties.
  - *Logs*: Records of events or transactions, often with a timestamp and other metadata, but not always in a uniform format.
  - *HTML pages*: Web pages that have a structured format using tags, but the content within can vary greatly.
  - *XML*: A markup language that defines rules for encoding documents in a format that is both human-readable and machine-readable.
  - *Nested JSON*: A lightweight data interchange format that is easy for humans to read and write, and for machines to parse and generate, often with nested structures.
  - *NoSQL data*: Databases that store data in a format other than tabular relations, often used for large sets of distributed data.
- **Unstructured**: This data lacks a predefined format or organization, making it more challenging to process and analyze.

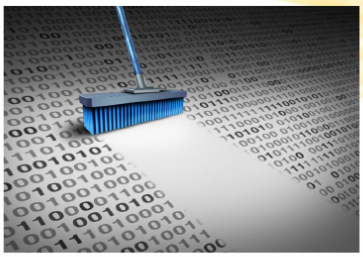
- *Text*: Free-form text data, such as emails, social media posts, or articles.
- *Pictures*: Image files that contain visual data without a specific structure.
- *Movies*: Video files that include a sequence of images and sound, lacking a structured format.
- *Blobs of data*: Binary Large Objects, which can be any type of data stored as a single entity in a database, often unstructured.

This slide highlights the different types of data structures, emphasizing the importance of understanding the format and organization of data when working with machine learning and big data. Each type requires different tools and techniques for processing and analysis.

16 / 21: Data Cleaning

Data Cleaning

- **Data cleanliness**
  - Quality of source data varies greatly
  - Data is typically messy
    - Duplicated records
    - Incomplete or missing records (nans)
    - Inconsistent formats (e.g., phone with / without dashes)
    - Mislabeled or unlabeled data
- **When to clean it?**
  - As soon as possible!
  - As late as possible!
  - In different stages
  - → Pipeline style: ETL vs ELT vs EtLT
- **Heuristics when dealing with data**
  - Hope for the best, assume the worst
  - Validate data early and often
  - Don't trust anything
  - Be defensive



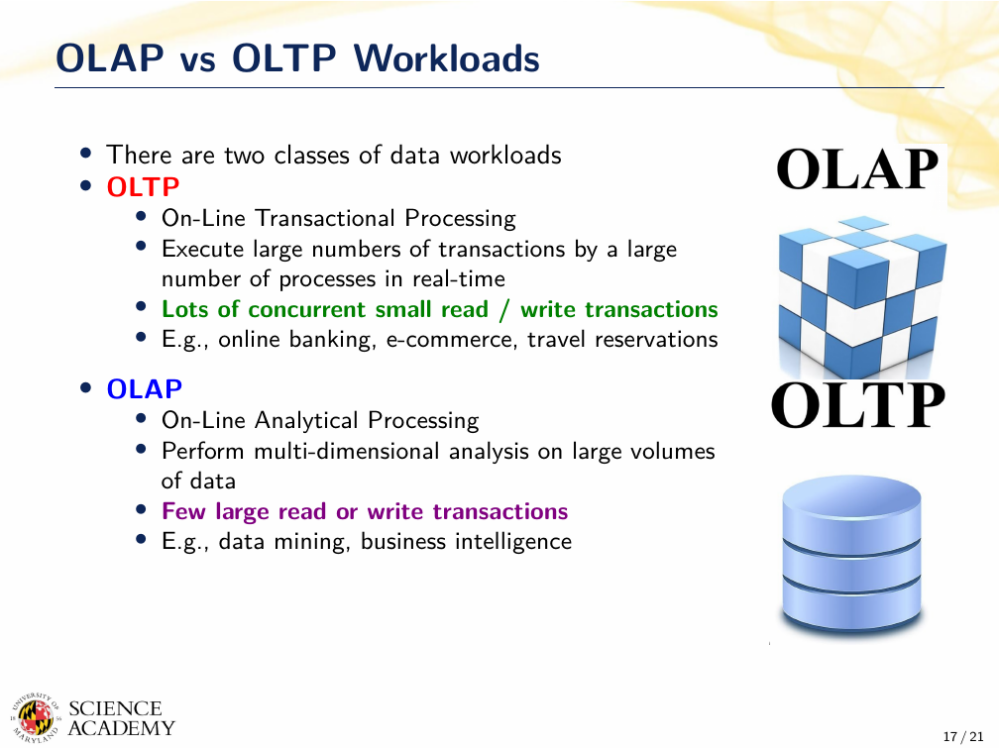
SCIENCE ACADEMY

16 / 21

- **Data cleanliness**
  - The quality of data you receive can vary widely. Some datasets might be well-organized, while others can be quite messy. This messiness can manifest in several ways:
    - **Duplicated records**: Sometimes, the same data entry appears multiple times, which can skew analysis results.
    - **Incomplete or missing records**: Often, datasets have gaps where information is missing, represented as 'nans' (not-a-number).
    - **Inconsistent formats**: Data might be recorded in different formats, such as phone numbers with or without dashes, making it hard to standardize.
    - **Mislabeled or unlabeled data**: Data might be incorrectly labeled or lack labels entirely, complicating analysis.
- **When to clean it?**
  - The timing of data cleaning is crucial and can be approached in several ways:
    - *As soon as possible*: Cleaning early can prevent errors from propagating through your analysis.
    - *As late as possible*: Sometimes, it's better to wait until you have a clearer picture of what data is essential.
    - *In different stages*: Cleaning can be an ongoing process, occurring at various points in your workflow.
    - *Pipeline style*: Different approaches like ETL (Extract, Transform, Load), ELT (Extract, Load, Transform), and EtLT (Extract, transform, Load, Transform) offer structured ways to handle data cleaning.
- **Heuristics when dealing with data**
  - When working with data, it's important to adopt a cautious mindset:

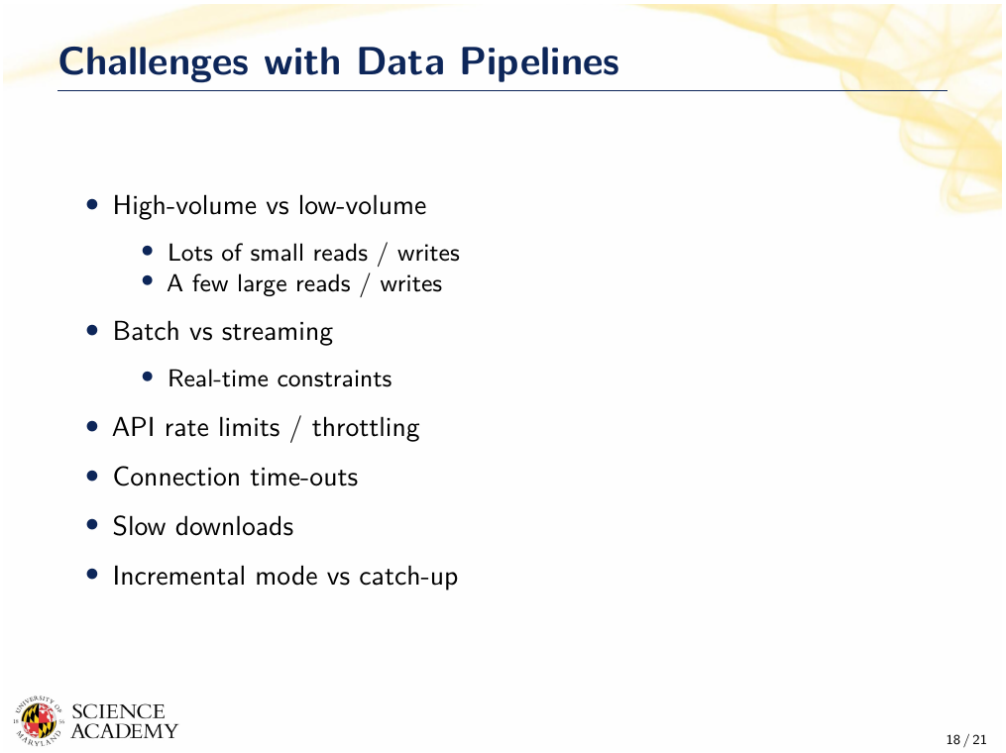
* *Hope for the best, assume the worst*: Be optimistic but prepared for potential issues.
* *Validate data early and often*: Regular checks can catch errors before they become problematic.
* *Don't trust anything*: Always verify data integrity and accuracy.
* *Be defensive*: Anticipate and guard against potential data issues to ensure reliable analysis.

17 / 21: OLAP vs OLTP Workloads



- **Two Classes of Data Workloads**
  - In the world of data processing, there are two main types of workloads: OLTP and OLAP. These are essential for different purposes and understanding them helps in choosing the right system for your needs.
- **OLTP (On-Line Transactional Processing)**
  - This type of workload is designed to handle a large number of transactions. Think of it as the backbone of systems that require real-time processing.
  - **Lots of concurrent small read/write transactions**: OLTP systems are optimized for handling many small transactions simultaneously. This is crucial for applications where speed and efficiency are key.
  - *Examples*: Online banking, e-commerce platforms, and travel reservation systems are typical use cases. These systems need to process numerous transactions quickly and accurately.
- **OLAP (On-Line Analytical Processing)**
  - OLAP is focused on analyzing large volumes of data. It's about understanding data trends and patterns rather than processing transactions.
  - **Few large read or write transactions**: Unlike OLTP, OLAP systems deal with fewer but much larger transactions. This is because they are designed to perform complex queries and data analysis.
  - *Examples*: Data mining and business intelligence applications rely on OLAP to provide insights and support decision-making processes. These systems help businesses understand their data better and make informed decisions.

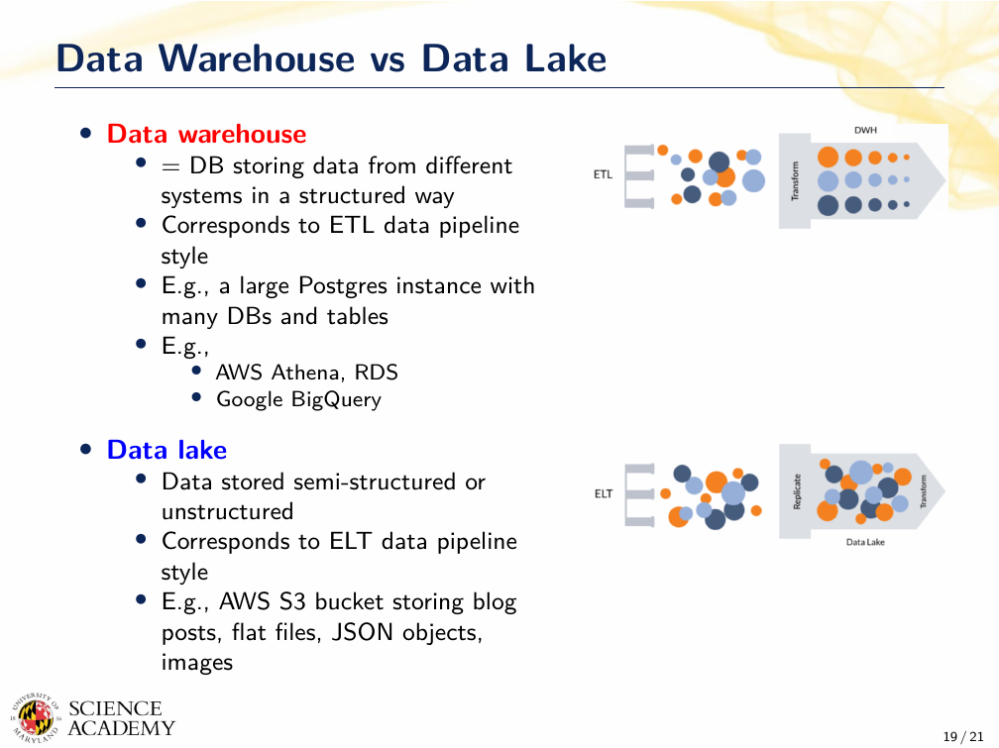
18 / 21: Challenges with Data Pipelines



- **Challenges with Data Pipelines**
  - **High-volume vs low-volume**
    - Data pipelines often have to handle different types of data loads. *High-volume* refers to situations where there are many small data transactions, like numerous small reads and writes. This can be challenging because it requires the system to efficiently manage a large number of operations. On the other hand, *low-volume* involves fewer but larger transactions, which can be easier to manage but require the system to handle large data chunks at once.
  - **Batch vs streaming**
    - Data can be processed in two main ways: *batch* processing and *streaming*. Batch processing involves collecting data over a period and processing it all at once, which is suitable for tasks without real-time constraints. *Streaming* processes data in real-time as it arrives, which is crucial for applications needing immediate insights or actions.
  - **API rate limits / throttling**
    - Many data sources impose *API rate limits*, restricting the number of requests you can make in a given time frame. This is known as throttling and can be a significant challenge when building data pipelines, as it requires careful planning to avoid exceeding these limits and potentially losing access to critical data.
  - **Connection time-outs**
    - Data pipelines often rely on network connections to access data. *Connection time-outs* occur when a connection takes too long to establish or maintain, leading to interruptions in data flow. This can be particularly problematic for time-sensitive applications.

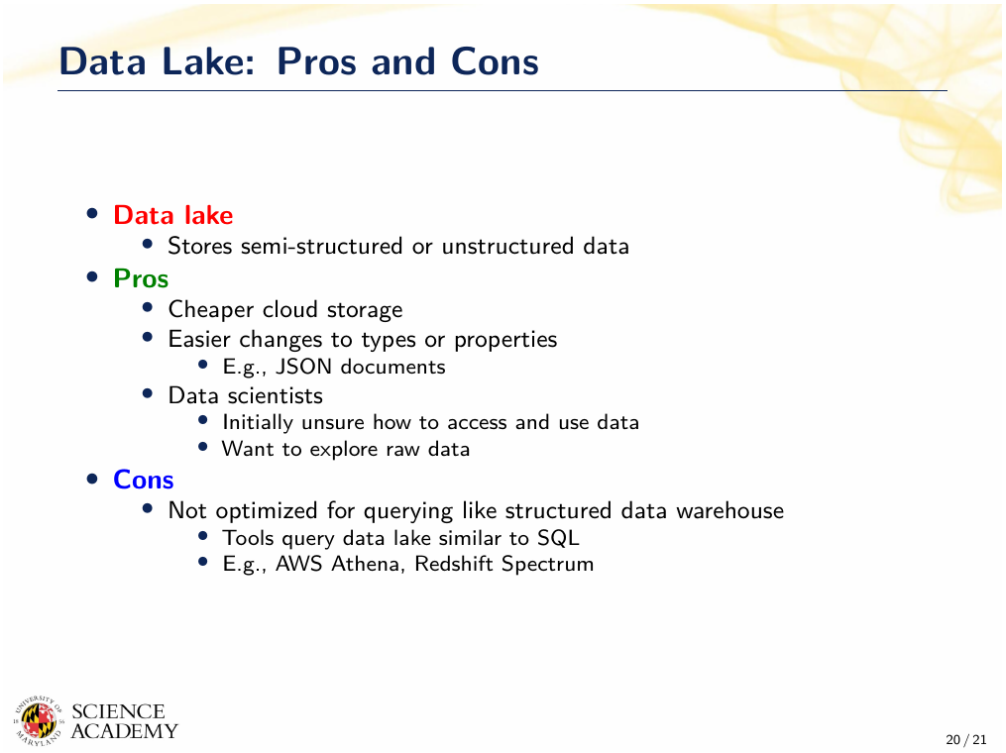
- **Slow downloads**
  - Sometimes, data downloads can be slow due to network issues or server limitations. This can delay the entire data processing pipeline, especially if the pipeline depends on timely data retrieval to function correctly.
- **Incremental mode vs catch-up**
  - Data pipelines can operate in *incremental mode*, where only new or changed data is processed, or in *catch-up* mode, where the pipeline processes all data to bring the system up to date. Choosing the right mode depends on the specific needs of the application and can impact the efficiency and speed of data processing.

19 / 21: Data Warehouse vs Data Lake



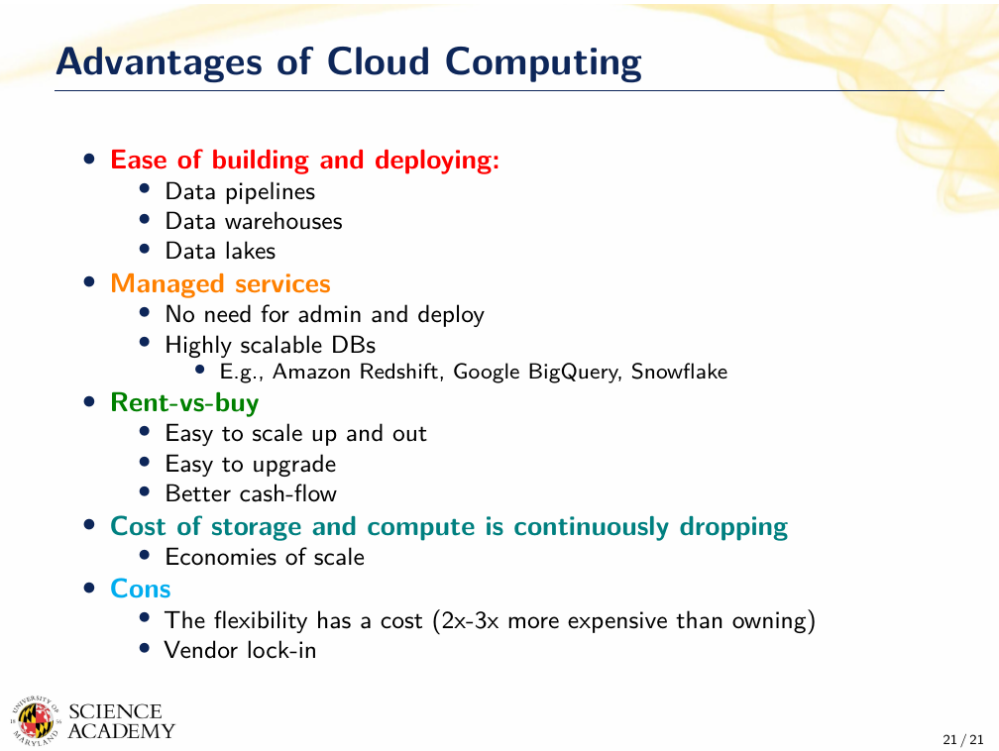
- **Data warehouse**
  - A *data warehouse* is essentially a large database that stores data from various sources in a highly organized and structured manner. This means that the data is cleaned, transformed, and formatted to fit into predefined tables and schemas.
  - It follows the ETL (Extract, Transform, Load) data pipeline style. In this process, data is first extracted from different sources, then transformed into a suitable format, and finally loaded into the data warehouse.
  - An example of a data warehouse could be a large Postgres instance that contains multiple databases and tables, each designed to hold specific types of data.
  - Popular services that provide data warehousing capabilities include AWS Athena and RDS, as well as Google BigQuery. These platforms offer robust solutions for managing and querying large volumes of structured data efficiently.
- **Data lake**
  - A *data lake* is a storage system that holds data in its raw form, which can be semi-structured or unstructured. This means that data is stored as-is, without any transformation or structuring, allowing for a more flexible and scalable storage solution.
  - It aligns with the ELT (Extract, Load, Transform) data pipeline style. Here, data is first extracted and loaded into the data lake, and transformation occurs later, often when the data is needed for analysis.
  - An example of a data lake could be an AWS S3 bucket that stores a variety of data types such as blog posts, flat files, JSON objects, and images. This flexibility allows organizations to store vast amounts of diverse data types in a single repository, making it easier to perform big data analytics.

20 / 21: Data Lake: Pros and Cons



- **Data lake**
  - A *data lake* is a storage system that holds a vast amount of raw data in its native format, which can be either semi-structured or unstructured. This means that the data is stored as-is, without any transformation or structuring, allowing for flexibility in how it can be used later.
- **Pros**
  - **Cheaper cloud storage**: Data lakes are cost-effective because they use cloud storage solutions that are generally cheaper than traditional data warehouses. This makes them an attractive option for storing large volumes of data.
  - **Easier changes to types or properties**: Since data lakes store data in its raw form, it's easier to make changes to data types or properties. For example, JSON documents can be stored without needing to define a schema upfront, allowing for flexibility in data management.
  - **Data scientists**: Data lakes are particularly beneficial for data scientists who may not know exactly what data they need at the outset. They can explore and experiment with the raw data to uncover insights and patterns without being constrained by predefined structures.
- **Cons**
  - **Not optimized for querying like structured data warehouse**: Data lakes are not designed for fast querying like traditional data warehouses. While tools exist to query data lakes using SQL-like syntax (such as AWS Athena and Redshift Spectrum), these queries may not be as efficient or fast as those on structured data warehouses. This can be a drawback for users who need quick access to specific data insights.

21 / 21: Advantages of Cloud Computing



- **Ease of building and deploying:**
  - Cloud computing makes it simpler to create and launch *data pipelines*, which are systems that move data from one place to another, often transforming it along the way. This is crucial for processing large amounts of data efficiently.
  - *Data warehouses* are centralized repositories for storing and analyzing large volumes of structured data. Cloud services provide tools to set these up quickly without needing to manage physical hardware.
  - *Data lakes* are storage systems that hold vast amounts of raw data in its native format. Cloud platforms offer the flexibility to store and process this data easily.
- **Managed services:**
  - Cloud providers offer *managed services*, meaning they handle the maintenance and deployment of infrastructure, freeing users from these tasks.
  - They provide *highly scalable databases*, which can grow with your needs without requiring manual intervention. Examples include Amazon Redshift, Google BigQuery, and Snowflake, which are designed to handle large-scale data processing efficiently.
- **Rent-vs-buy:**
  - Cloud computing allows businesses to *scale up and out* easily, meaning they can increase their resources as needed without significant upfront investment.
  - It also simplifies *upgrading* systems, as cloud providers regularly update their services.
  - This model improves *cash-flow* since businesses pay for what they use rather than investing heavily in infrastructure upfront.
- **Cost of storage and compute is continuously dropping:**
  - The *economies of scale* achieved by cloud providers mean that as they grow, they can offer storage and computing power at lower costs, benefiting users.

- **Cons:**
  - While cloud computing offers flexibility, it can be *2x-3x more expensive* than owning and managing your own infrastructure.
  - There is a risk of *vendor lock-in*, where switching providers becomes difficult due to reliance on a specific provider's services and tools.