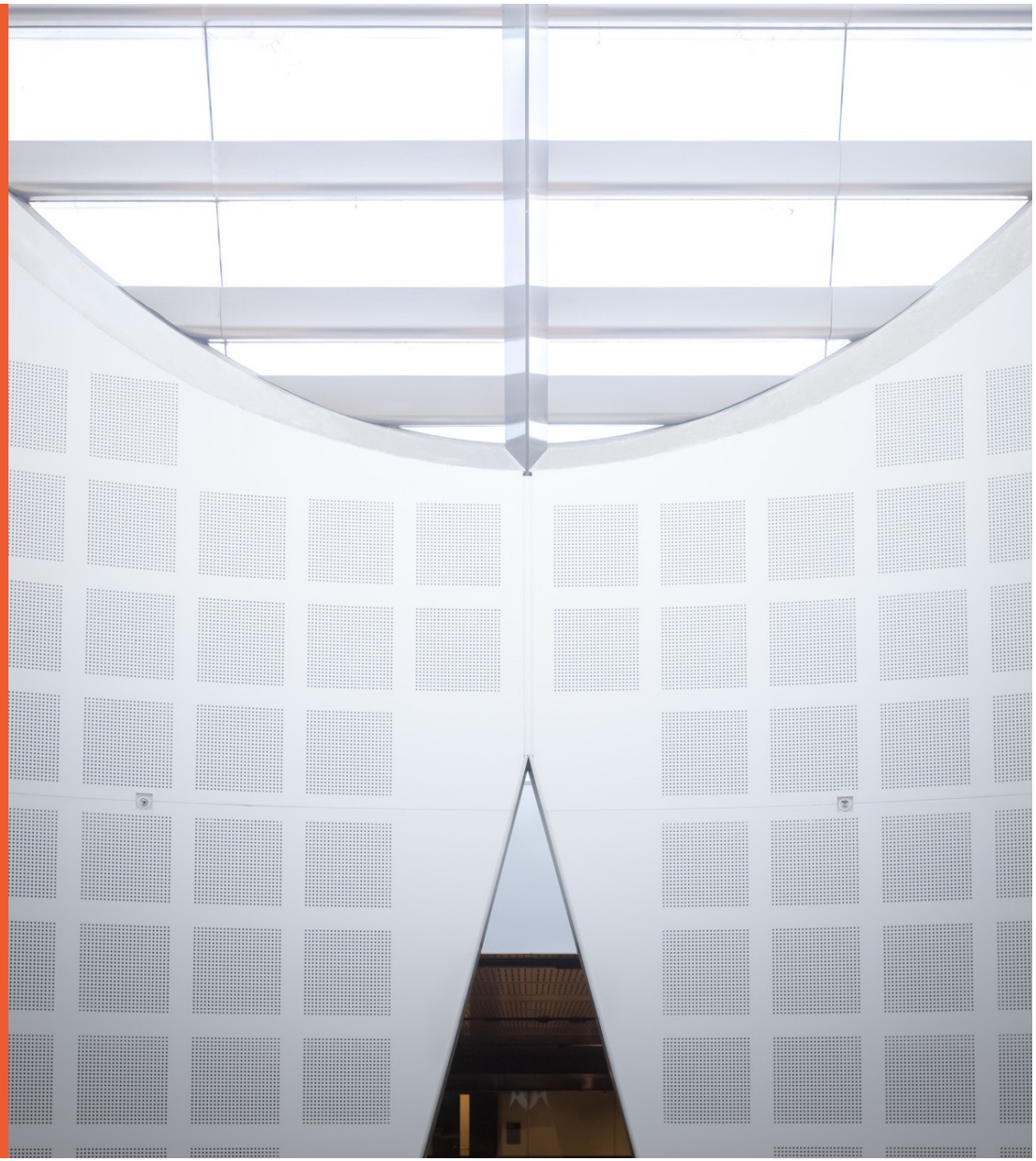
COMP5339: Data Engineering

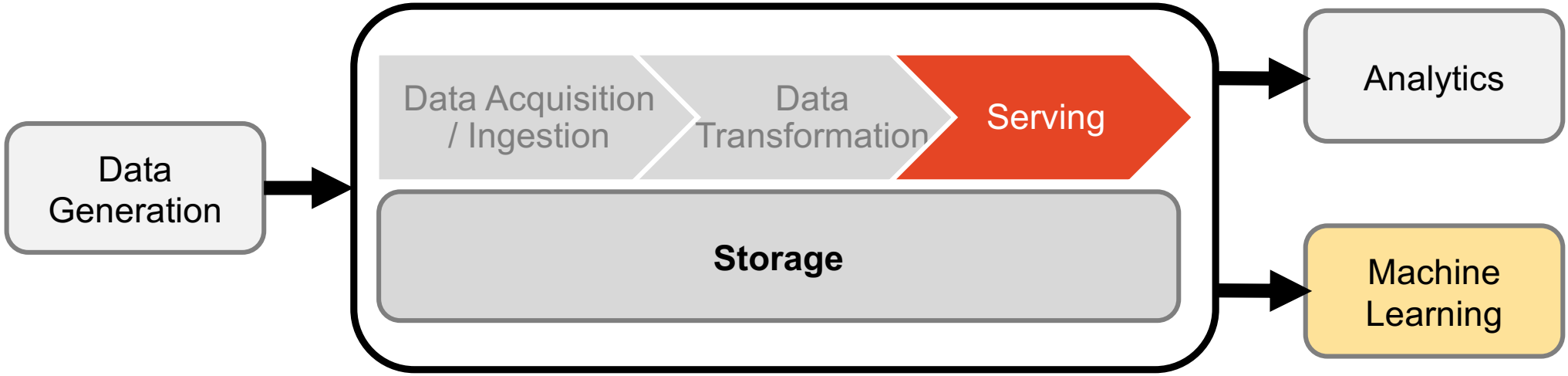
Week 11: DataOps, Data Architectures, and ML

Presented by
Uwe Roehm
School of Computer Science

THE UNIVERSITY OF SYDNEY



Data Engineering Lifecycle – Today's Focus



[J. Reis and M. Housley: Fundamentals of Data Engineering, 2022]

Agenda

- Data Architectures
- Data Ops
- Scaling ML
- Serving Data for Data Analytics: Feature Stores

Slides based in parts on Elliot Zhu's "DE in Practice" talk (2025)

Data Architectures

Data Architecture

- "Architecture represents the significant design decisions that shape a system, where significant is measured by cost of change."

  [Grady Booch, "On design", IBM Community Blog, 2006.]

- "Identifying the data needs of the enterprise (regardless of structure) and designing and maintaining the master blueprints to meet those needs. Using master blueprints to guide data integration, control data assets, and align data investments with business strategy."

  [DAMA DMBOK, call no 005.74 304]

- **Data Architecture**
  - Design decisions that support evolving data needs of an enterprise / organisation
  - Good data architecture is flexible and easily maintainable

Data Architecture Aspects

- Operational Architecture
  - Functional requirements regarding the data architecture; "***What*** needs to be done?"
  - For example:
    - What business / data analytical processes does the data serve?
    - How does the organization manage data quality?
    - What is the latency requirement for data becoming being available to query?
- Technical Architecture
  - ***How*** data is ingested, stored, transformed, and served along the data pipeline
  - For example:
    - how will you move 10 TB of data every hour from a source database to your data lake?

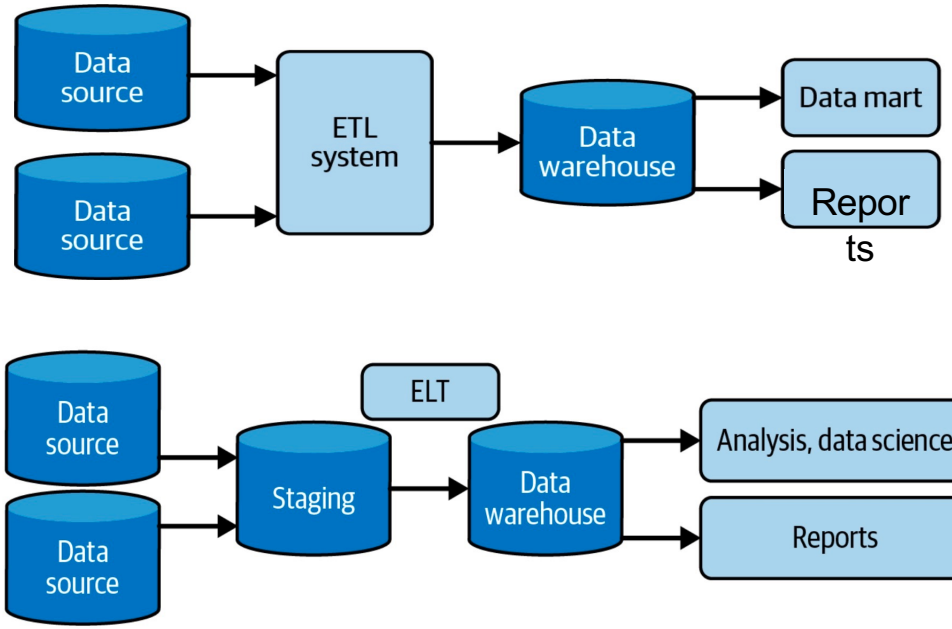
What makes a Good Architecture?

- principles of data architecture
  - Design for **automation**
  - **Availability** / Plan for failure
  - Priorities **Security** / **Data Privacy**
  - Architect for **Scalability**
  - favor **loosely coupled systems**
  - **always** be architecting

- cf. also AWS Well-Architected Framework or Google Cloud's Five Principles for Cloud-Native Architecture

Data Architecture Example: Data Warehouse

- Data Warehouse
  - central data hub used for reporting & analysis
  - Separates production systems from OLAP – Online Analytical Processing
  - **ETL** – Extract-Transform-Load vs. **ELT** with a separate staging to move data more or less directly (faster) from production systems

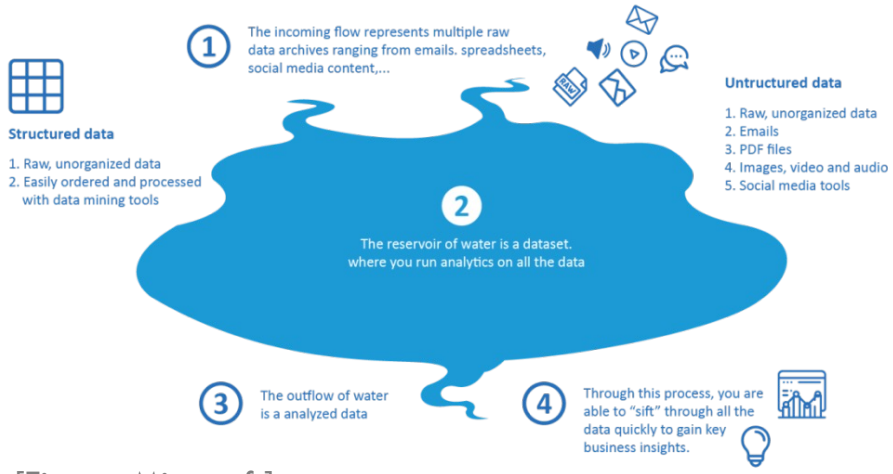


[Figures: Reis/Housley: Fundamentals of Data Engineering, 2022]

- Modern evolution: **Cloud Data Warehouse**
  - Scalability by separating compute from storage

Data Architecture Example: Data Lake

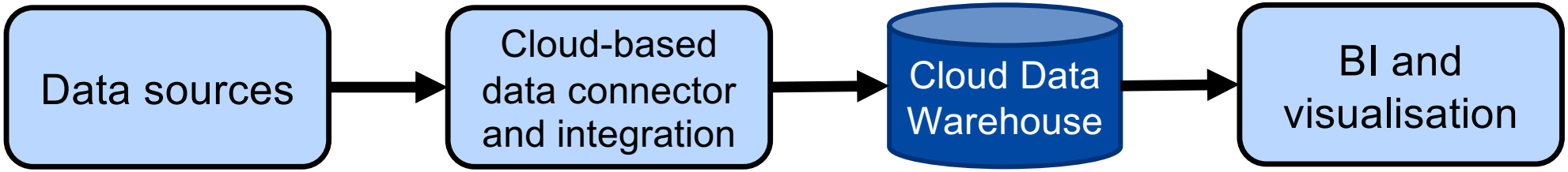
- Big Data architecture designed to handle the ingestion, processing, and analysis of data that is too large or complex for traditional database system
- Data Lake
  - dump all of your data – structured and unstructured – into a central location instead of relying on a monolithic data warehouse with tightly coupled storage and compute
  - First generation:
    HDFS + Map Reduce
- Cons:
  - Write only
  - danger of deteriorating into a *data swamp*



[Figure: Microsoft]

Modern Data Stack

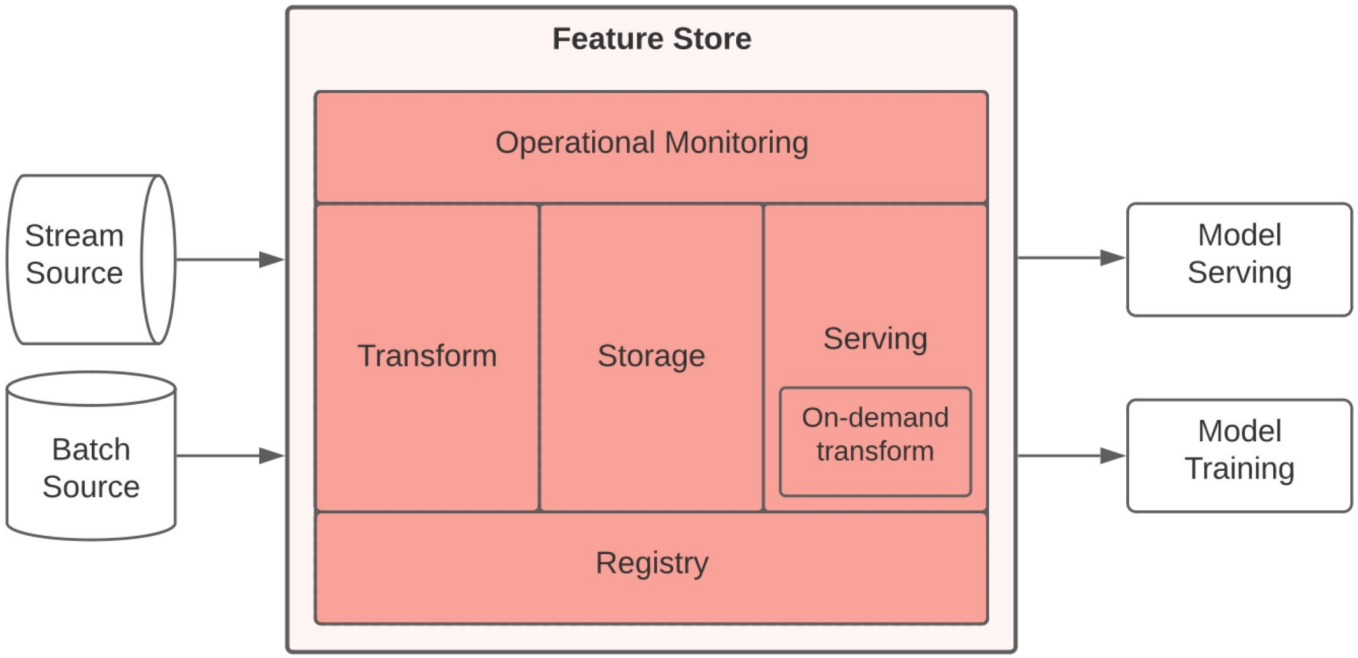
- Does not rely on expensive, monolithic toolsets
- Objective: cloud-based, plug-and-play, easy-to-use, off-the-shelf components
  - include data pipelines, storage, transformation, data management/governance, monitoring, visualization, and exploration.
- Key outcomes of the modern data stack are self-service (analytics and pipelines), agile data management, and using open source tools or simple proprietary tools with clear pricing structures.



[Reis/Housley: Fundamentals of Data Engineering, 2022]

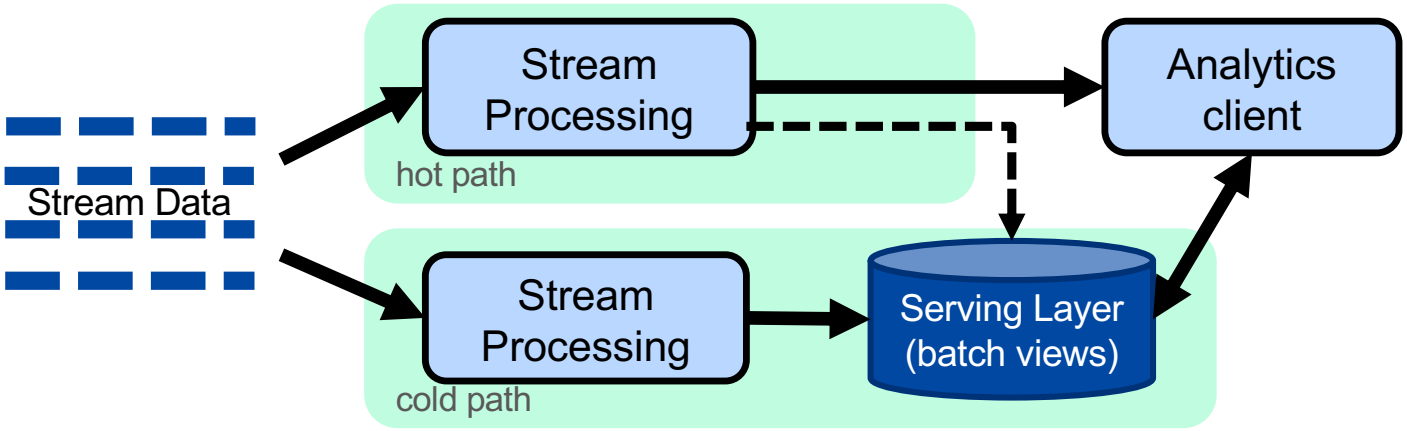
Serving ML: Feature Store

- Some data pipelines specifically server for training and serving machine learning models => Feature Store architecture
  - More in separate content section this week

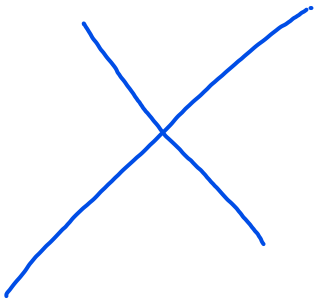


Data Streaming: Lambda Architecture

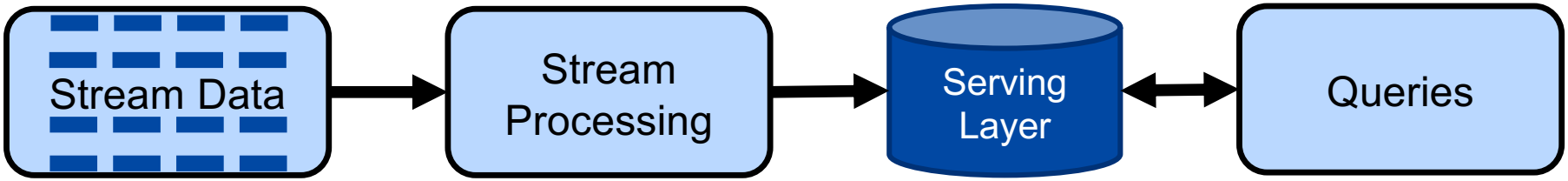
- One drawback of batch-oriented architectures can be that it introduces latency
- **lambda architecture** (Nathan Marz) addresses this problem by creating two paths for data flow.
  - A **batch layer** (cold path) stores all of the incoming data in its raw form and performs batch processing on the data. The result of this processing is stored as a batch view.
  - A **speed layer** (hot path) analyzes data in real time. This layer is designed for low latency, at the expense of accuracy.



Data Streaming: Kappa Architecture



- **Event-driven architecture** that uses a stream-processing platform as backbone for all data handling
  - Same idea than Lambda Architecture, but all data flows through a single path
- Termed by Jay Krebs as **Kappa Architecture**



Summary

- Data Architecture Principles

- Example Architectures
  - Data Warehousing Architectures
  - Big Data Architectures
    - Batch-driven: Data Lake
    - Event-driven: Data Stream Processing Architectures

Data Ops

Slides based on Elliot Zhu's COMP5339 presentations.

CI / CD

- **Continuous Integration (CI):** Automatically testing and integrating new code into a shared repository

- **Continuous Deployment (CD):** Automatically deploying tested code to production environments.

- Relevance with regard to machine learning (ML):
  - Frequent model updates
  - Automating training and deployment processes

Data Engineering Challenges in Traditional ML Pipelines

- **Data Quality and Validation:** Ensuring clean, validated data flows into training models.
- **Data Pipeline Complexity:** Managing the flow from raw data ingestion to feature engineering and model training.
- **Scaling and Performance**: Efficiently processing large datasets across distributed systems.

- *Example*: Ingesting streaming data from IoT devices, processing it with Spark, and ensuring model retraining every hour.

What is DataOps?

- **DataOps** is a set of collaborative data management practices designed to speed up data delivery, maintain quality, and foster collaboration.
  - Modelled after DevOps, but focused on automating data management and analytics processes.
- **Key Goal**: Break down silos between data producers and consumers, and ensure smooth, automated data workflows.
- **DataOps Life Cycle:**

Why is DataOps Important?

- **Data Challenges**: Manual data management tasks are time-consuming, and data silos restrict access.
- **Core Purpose**: Automate data pipelines to handle explosive data growth, maintain quality, and foster collaboration between teams.
- **Agility and Scalability**: Enables teams to adapt quickly to changing business needs and evolving datasets.

DataOps in Action: Breaking Down Silos

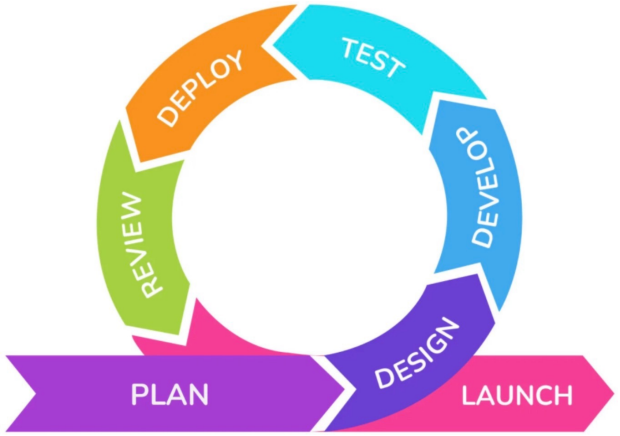
- **Unifying Data**: Removes silos between upstream (data producers) and downstream (data consumers) teams.
- **Collaborative Framework**: DataOps encourages cross-team collaboration, aligning data scientists, data engineers, and business stakeholders.
- Result: A more holistic view of data, leading to insightful analysis and improved decision-making.
- *Example*: A financial institution uses DataOps to streamline collaboration between data engineering and risk analysis teams to improve fraud detection systems.

Key Functions of DataOps

1. **Pipeline Orchestration**: Manages the flow of data, automating extraction, transformation, and loading (ETL).
   - *Example*: A logistics company automates its ETL pipelines to transform raw shipment data into actionable delivery insights.
2. **Data Quality Monitoring**: Real-time checks on data quality to ensure accuracy and reliability.
   - *Example*: An airline company monitors real-time flight data for consistency and accuracy, ensuring accurate delay predictions.
3. **Governance and Security**: Protects data, ensuring it complies with regulations and policies.
4. **Self-Service Data Access**: Empowers users to explore data without relying on IT.

DataOps and Agile Methodology

- **Agile Principles**: DataOps borrows agile principles like iterative development, continuous feedback, and collaboration.
- **Faster Data Insights**: Iterative data pipelines allow for quicker data processing and insight delivery.
- **Continuous Monitoring**: Automated feedback loops ensure pipelines are continuously improved and adapted to new requirements.
- *Example:* A product development team uses DataOps to continuously refine product features based on user data from multiple sources, improving the product iteratively.



Benefits of DataOps

- **Increased Agility**: Speeds up data delivery cycles by automating repetitive tasks.
- **Better Data Quality**: Ensures clean, reliable data through continuous testing and monitoring.
- **Fostering Collaboration**: Encourages communication between data teams and business users, enhancing data-driven decision-making.
- **Governance & Compliance**: Ensures transparency and security in data handling, addressing regulatory challenges like GDPR and CCPA

Best Practices for Implementing DataOps

- **Define Data Standards:** Establish clear rules for data quality and metadata early on.
- **Automate Processes:** Leverage tools to automate data processing and validation.
- **Build Cross-functional Teams:** Assemble a diverse team of data engineers, data scientists, and business stakeholders.
- **Focus on Scalability:** Design pipelines that can handle increasing data volumes and complexity.
- **Continuous Improvement:** Regularly review and optimize data workflows to improve efficiency and performance.

DataOps Lifecycle: From Planning to Monitoring

- **Plan**: Define data quality metrics and availability goals through collaboration.
  - *Example*: A financial services firm defines KPIs for risk management, ensuring accurate data flows for regulatory reports.
- **Develop**: Build data products and models with input from engineers and scientists.
- **Integrate**: Connect data pipelines to the existing tech stack for seamless operation.
  - *Example*: A retail company integrates product inventory models into its workflow automation tools to trigger alerts for low stock levels.
- **Test**: Validate data accuracy and integrity with automated tests.
- **Deploy**: Move data models to production after thorough testing.
- **Monitor**: Continuously monitor pipeline performance and data quality to catch issues early.

Key Technologies Supporting DataOps

- **Data Ingestion:** Tools like Apache NiFi or Kafka to streamline data ingestion at scale.
- **Pipeline Orchestration:** Apache Airflow or Luigi to automate complex workflows.
- **Data Transformation:** Tools like Apache Spark for efficient data cleaning and transformation.
- **Data Cataloging:** Tools like Alation or Apache Atlas to manage and search for data assets.

Automating Data Curation and Metadata Management

数据管理

- **Data Curation**: Automate data cleansing, transformation, and standardization to ensure high-quality data.
- **Metadata Management:** Track data lineage and changes across the pipeline using automated metadata tools.
  - Example: An energy company uses metadata management to track changes in energy usage data, ensuring accurate billing.
- **Active Metadata**: Dynamic metadata that provides real-time insights into data assets and their transformations.

Governance, Security, and Master Data Management

- **Governance Automation**: Enforce data quality rules and access control policies to maintain compliance.
- **Security Automation**: Use encryption, access control, and data loss prevention to secure data pipelines.
- **Master Data Management**: Ensure consistency and reliability by automating tasks like deduplication and synchronisation across systems.

- **Example**: A healthcare provider automates master data management to synchronise patient records across departments, ensuring a single source of truth.

Data Observability: The Five Pillars

- **Freshness**: Ensure data is updated and ingested in a timely manner.
- **Distribution**: Check if data values fall within acceptable ranges.
- **Volume**: Monitor for missing or incomplete data.
- **Schema**: Track structural changes to data to ensure integrity.
- **Lineage**: Understand how data has moved and transformed across systems.
  - *Example*: A marketing analytics firm tracks data lineage to ensure accurate attribution of customer conversion data across campaigns.

Examples of DataOps in ML Workflows

- **Real-time Fraud Detection**: Using Kafka to stream transaction data into a Spark pipeline, performing real-time data validation and triggering model retraining.

- **Retail Analytics**: Automating the ingestion and cleaning of sales data, ensuring consistency and scalability with growing datasets, and using ML models for demand forecasting.

Scaling ML: Data to ML or ML to Data?

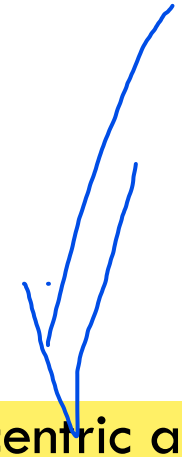
Large-scale Data Analysis: in Data Store or in Platform?

传统是前者
现在更多是后者，比如Shared-Nothi

- Challenge with large-scale data analytics:
  Does the data come to the computation, or the computation to the data?
- Standard approach in many data science projects is:
  - Keep data in files or in databases / data warehouses purely for storage
  - Load data completely into Python program (e.g. using Pandas)
  - Then do all computation / analysis in data platform (e.g. using Python)
- Example:

```
import pandas
from sklean.feature_extraction.text import CountVectorizer
engine = create_engine('postgresql://login@localhost:5432/postgres')
text = pandas.read_sql_query('SELECT * FROM DocTable', con=engine)
v = CountVectorizer(binary=False, lowercase=True)
vector_matrix = v.fit_transform(text.message)
```

SQL vs Python



- The computation-centric approach has some advantages, such as:
  - a single code base;
  - good support for data ingest in, e.g., Python; and
  - good visualization support
- But: it is **not easily scalable**
  - Requires data to fit into the available main memory...
  - It also makes no use of multi-core machines (no parallelism in the programs)
  - Also data needs to be shipped; no use of the query capabilities of databases.

- On the other hand: databases are not good for iterative algorithms and out-of-the-box no support machine learning algorithms…

Apache MADlib

https://madlib.apache.org

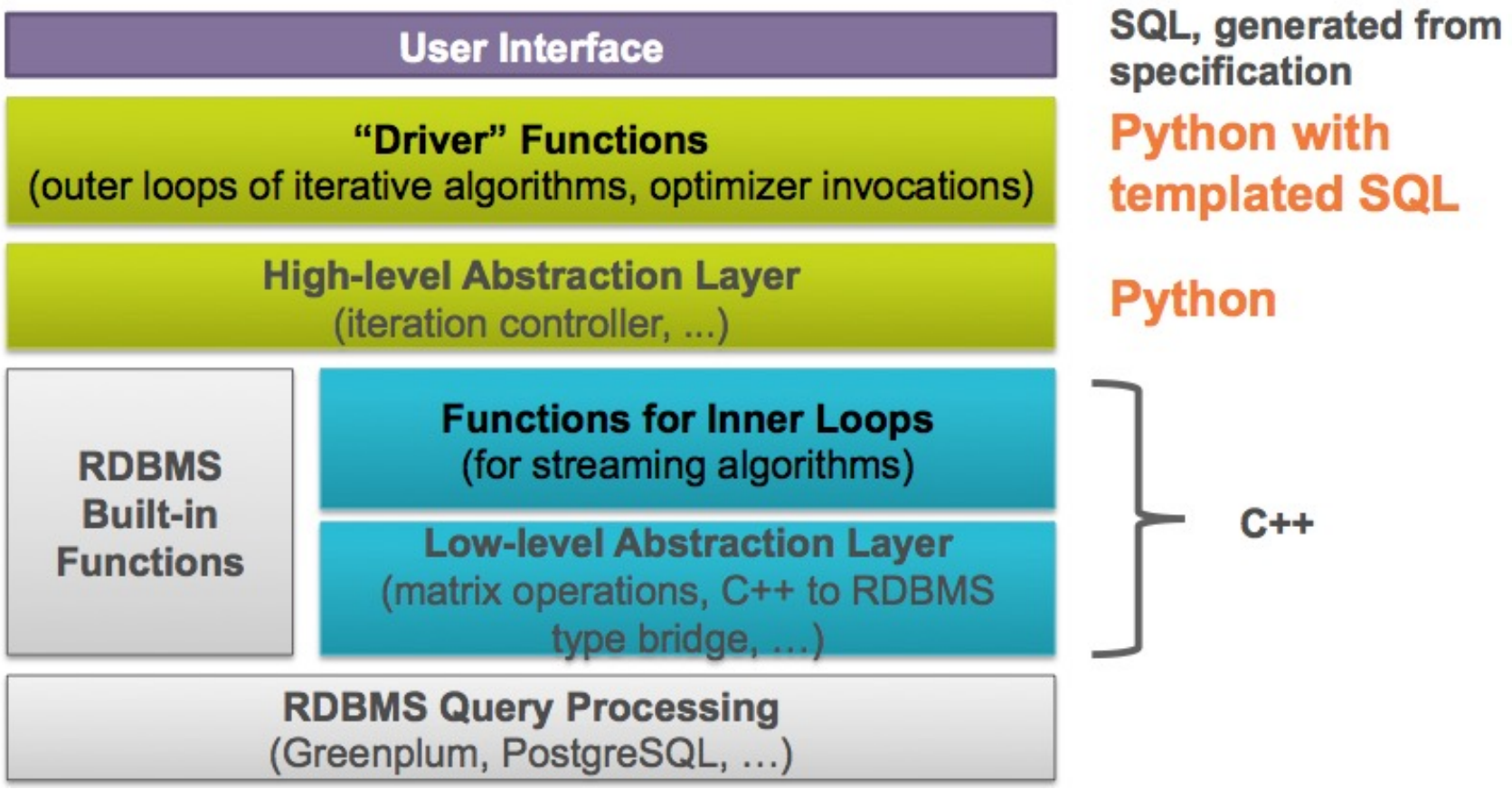
- Example: Open-source library for scalable in-database analytics
  - Data-parallel implementations of mathematical, statistical, and machine learning methods for PostgreSQL and Greenplum database
  - Data-parallel problems allow parallel execution of algorithm on different subsets of data
  - Supports **PostgreSQL 9.6 or v10**
- MADlib functionality includes:
  - Descriptive statistics, Classification, Regression, Clustering
  - Topic Modeling: attempts to identify clusters of documents that are similar
  - Association Rule Mining (aka market basket analysis or frequent itemset mining)
  - many more…
- MADlib functions are parallelisable implemented
  - Claim much better scalability with MADlib than stand-alone Python or R programs
- Documentation: https://cwiki.apache.org/confluence/display/MADLIB/Installation+Guide

MADlib in PostgreSQL

- PostgreSQL, as many other databases, is extensible
  - Supports *user-defined functions (UDFs)* which extend the SQL capabilities
    - Many different languages supported: SQL, PL/pgSQL, Perl, Python, C/C++ (https://www.postgresql.org/docs/current/server-programming.html)
  - UDFs come in different flavours
    - functions, aggregates, table-valued functions, types, operators…
  - UDFs are executed inside the database with direct access to the data
    - No need for data export / network traffic
    - Can process data larger than main memory because of DBMS buffering

- MADlib loads many pre-compiled functions
  - After installation, shows 1459 functions available in install schema
  - Written and compiled in C++

Architecture



[Source: https://cwiki.apache.org/confluence/display/MADLIB/Architecture]

MADlib Usage in SQL

- MADlib functions can be used as part of the standard SQL queries
  - All functions in namespace/schema: madlib
  - load data into a database table, often with array types (eg. multi-dim points)
  - Call matlib function with parameters specifying source table, source column(s), algorithm parameters and target table

Example: `SELECT * FROM madlib.knn( parameters );`

```
SELECT * FROM madlib.knn('knn_train_data',      -- Table of training data,
                         'data',                -- Col name of training data,
                         'id',                  -- Col name of id in train data,
                         'label',               -- Training labels,
                         'knn_test_data',       -- Table of test data,
                         'data',                -- Col name of test data,
                         'id',                  -- Col name of id in test data,
                         'knn_result',          -- Output table,
                         3,                     -- Number of nearest neighbors,
                         True,                  -- True to list nearest-neighbors by id,
                         'madlib.squared_dist_norm2' -- Distance function");
```

Example: ARIMA

- Given a time series of data X, the **Autoregressive Integrated Moving Average (ARIMA)** model is a tool for understanding/predicting future values in time series.
- Prerequisite:
  Time series in a relation *input_table* in *point representation*:
  One time column, one value column, and optional one or more grouping columns.
- MADlib's ARIMA training function has the following syntax:

```
madlib.arima_train( input_table, training_output_table,
                    timestamp_column, timeseries_column, grouping_columns,
                    include_mean, non_seasonal_orders, optimizer_params)
```

- We can then use this output table to do some forecast using MADlib's forecast function:

```
SELECT madlib.arima_forecast( training_output_table,
                              forecast_output_table, k );
SELECT * FROM forecast_output_table;
```

Data-Parallelism in MADlib

- Included functions in principle ready for parallelism
- works with Greenplum parallel database
  - Greenplum is a shared-nothing database using postgres per each node

- On PostgreSQL server itself, runs into limits with parallel query execution
  - PostgreSQL since version 9.6 supports parallel queries, e.g. parallel scans on multiple CPU cores
  - BUT: many limitations, e.g. no updates allowed during parallel query
    - -> which is exactly what the MADlib functions are doing…

Ongoing Research

- Research into in-DBMS ML
  - MASQ, SIGMOD 2021
    approach to translate selected scikit-learn classifiers and regressors to SQL
  - DB4ML, SIGMOD2019 (own work)
    An In-Memory Database Kernel with Machine Learning Support
  - Multiple works in integration of machine learning via UDFs, such as Bismark, SIGMOD 2012: "Towards a unified architecture for in-RDBMS analytic"

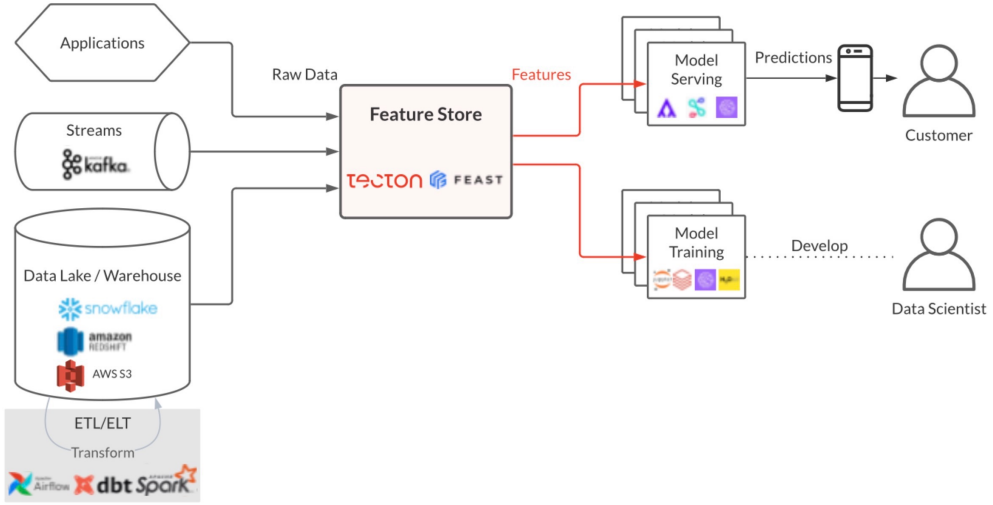
- Vendors strive to support existing ML runtimes in their systems
  - E.g. Microsoft SQLServer ML Services, VerticaML, Greenplum MADlib
  - Overview: Yuhao Zhang et al (VLDB 2021): "Distributed Deep Learning on Data Systems: A Comparative Analysis of Approaches."

Serving Data for Analytics: Feature Stores

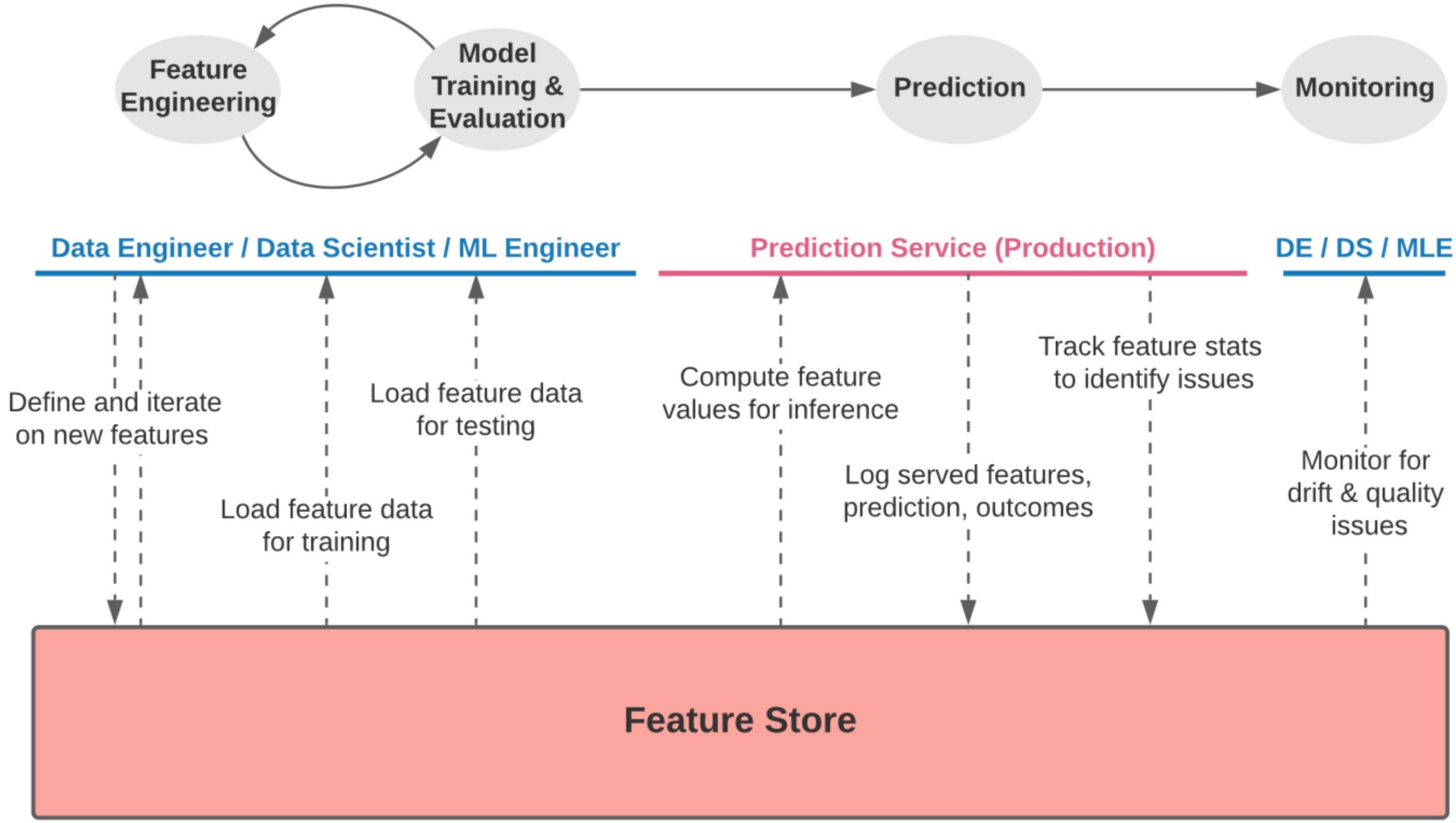
Slides based on Elliot Zhu's COMP5339 presentations.

Serving Data for ML: Feature Store

- Feature stores aim to manage data for operational machine learning (ML).
  - This is a 'Data to ML' approach.
- **Challenge:** Deploying ML into production introduces new data infrastructure requirements, necessitating a specialised system for feature management.
- **Definition of Features:** A feature is an input signal (feature vector) used by ML models.
  - Example: "Is this transaction in a foreign country?" for fraud detection.

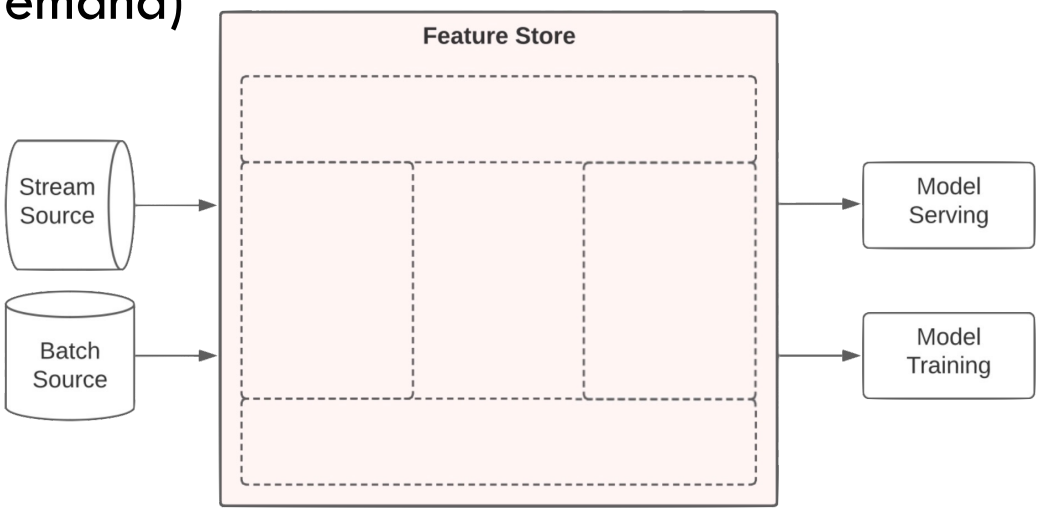


Feature Store Lifecycle



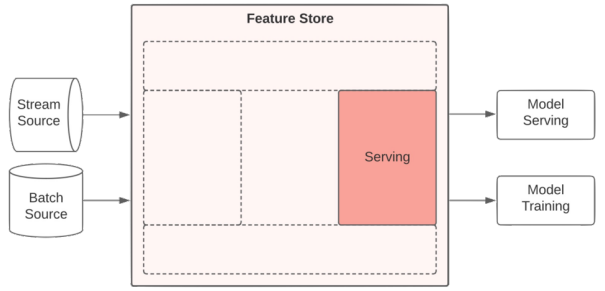
Components of a Feature Store

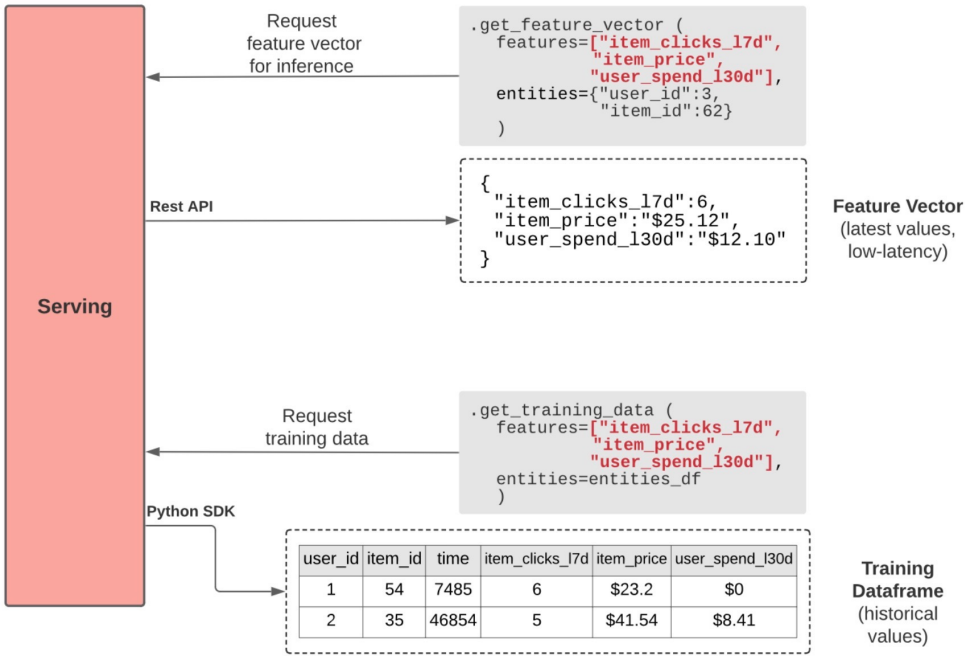
- **Transformation:** Handles various data transformations (batch, streaming, on-demand)
- **Storage:** Manages both online and offline storage to meet the needs of different ML applications.
- **Serving:** Delivers feature data in real time for model inference and training.
- **Monitoring:** Tracks feature health, correctness, and data quality.
- **Feature Registry:** Acts as a central source of truth for all feature definitions and metadata.



Serving Feature Data for ML Models

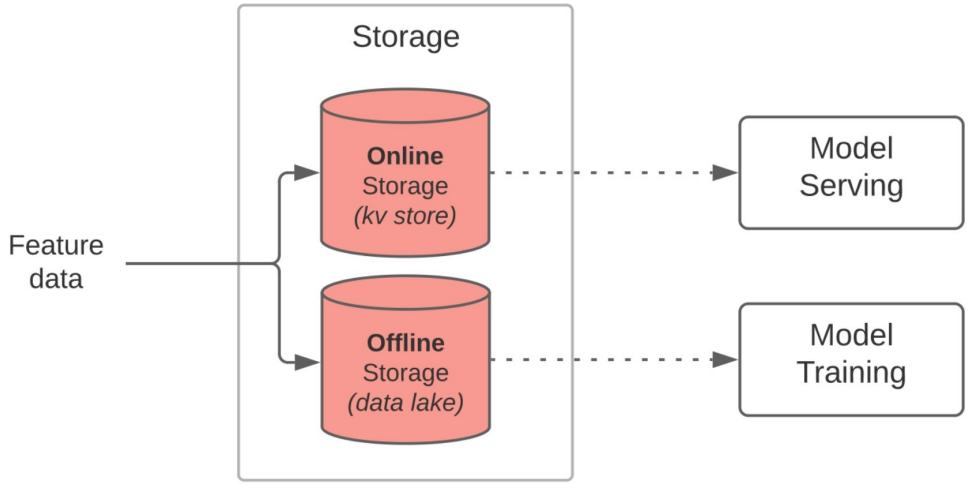
- **Consistency is Key:** Ensures that features used during training are the same as those served in production (avoids training-serving skew).
- **Offline Serving:** Access historical data for training using point-in-time accuracy (time travel).
- **Online Serving:** Provide fresh feature values in real time via low-latency APIs.





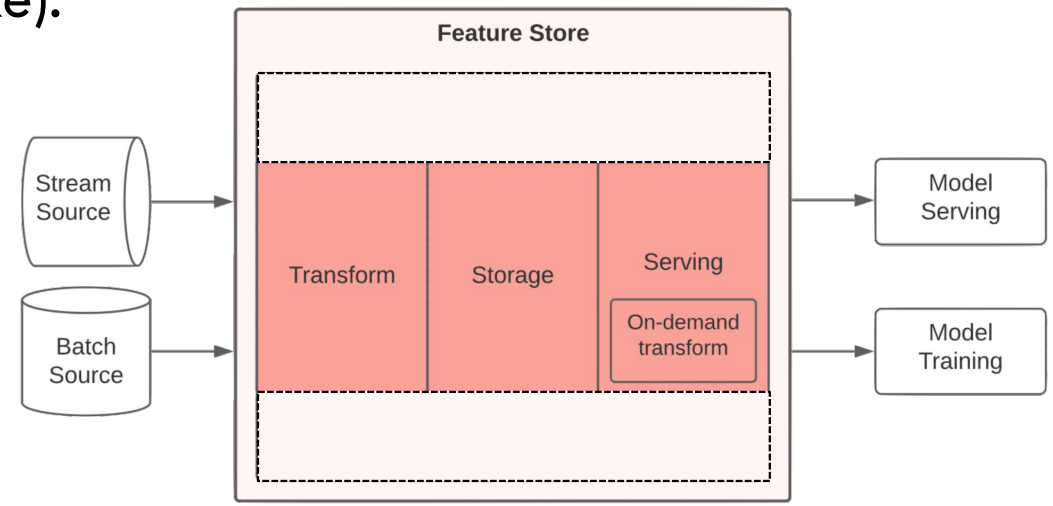
Data Storage in a Feature Store

- **Offline Storage**: Supports storing large volumes of historical feature data for training (e.g., S3, BigQuery, Snowflake).
- **Online Storage:** Optimised for low-latency feature retrieval during inference (e.g., DynamoDB, Redis).
- **Entity-Based Data Model:** Each feature is associated with an entity (e.g., user) and a timestamp.

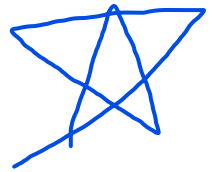


Feature Transformation in Machine Learning

- **Batch Transform**: Applied to data at rest (e.g., user country from a data lake).
- **Streaming Transform:** Applied to streaming sources (e.g., clicks per user in the past 30 minutes).
- **On-Demand Transform:** Used at the moment of prediction (e.g., whether a user is currently in a specific location).
- Feature stores simplify the process of combining various types of transformations in a model.



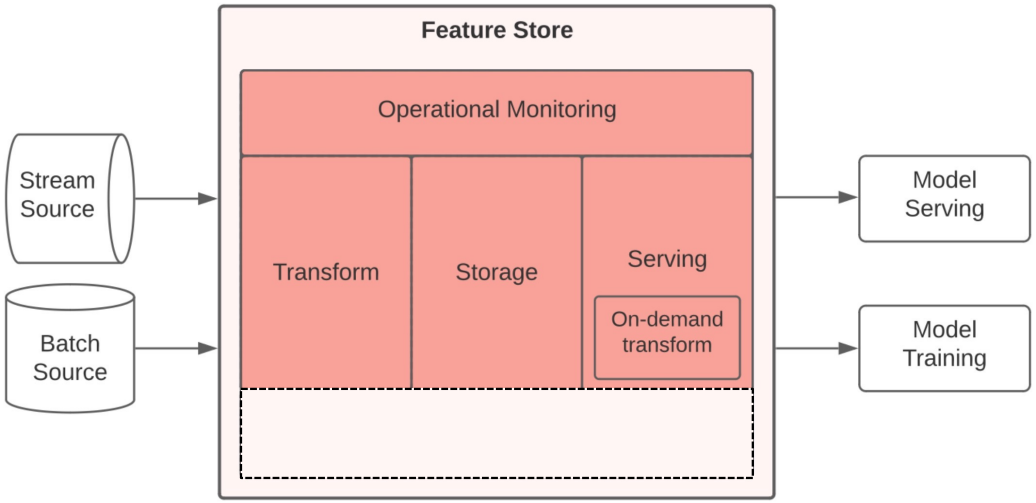
Feature Transformation in Machine Learning



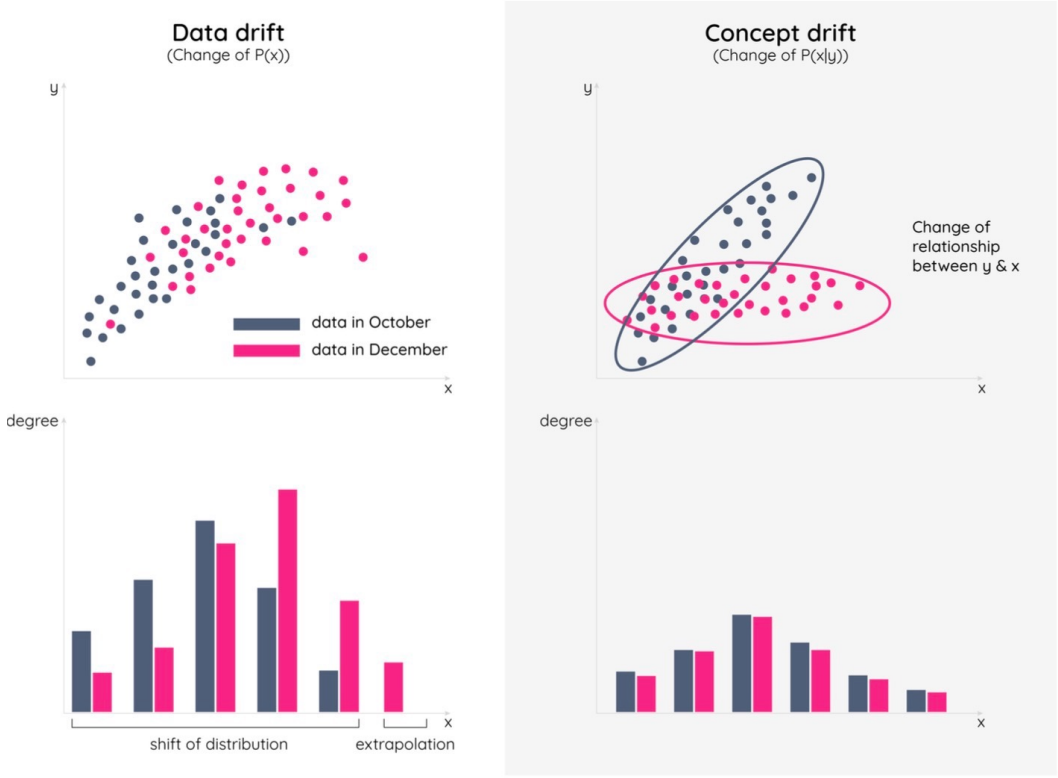
Feature Type	Example
Batch Transform	User's country and product category based on their historical purchase data, retrieved from a data warehouse, and used for personalized marketing campaigns.
Stream Transform	Number of clicks per user in the last 30 minutes within a specific product vertical, used to provide real- time product recommendations on an e-commerce site.
On-Demand Transform	Determining if a user is in a supported location at the time of app access, or calculating the similarity score between a searched listing and the current user query for real-time personalized search results.

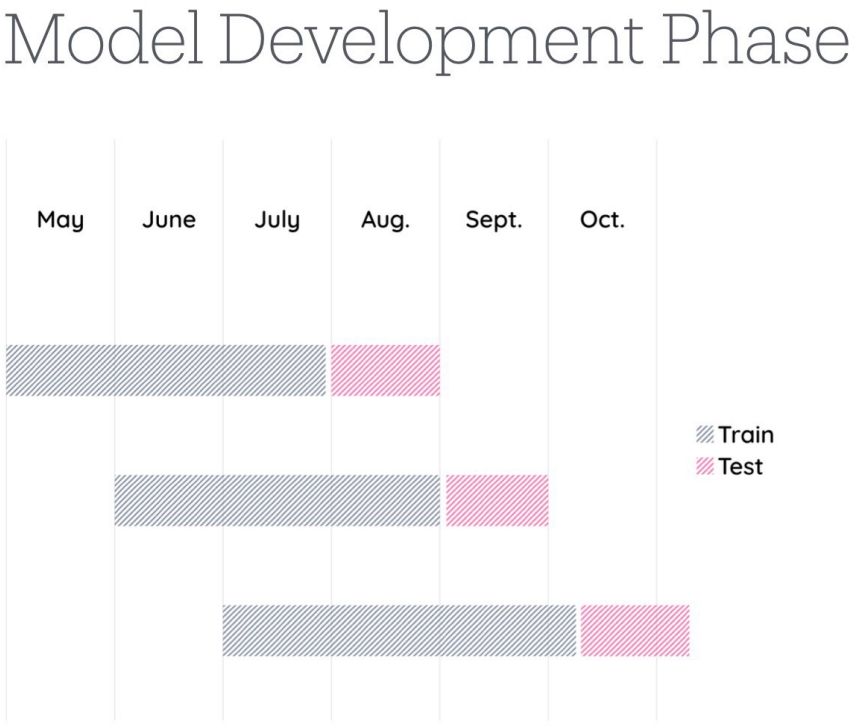
Monitoring Feature Stores in Production

- **Data Monitoring:** Tracks data quality by monitoring for **drift** and **training-serving skew.**
- **Operational Monitoring:** Tracks system metrics such as feature storage availability, capacity, and staleness.
- **Benefit:** Monitoring systems ensure that ML models are using high-quality, up-to-date features for accurate predictions.



Drift and Continuous Model Development

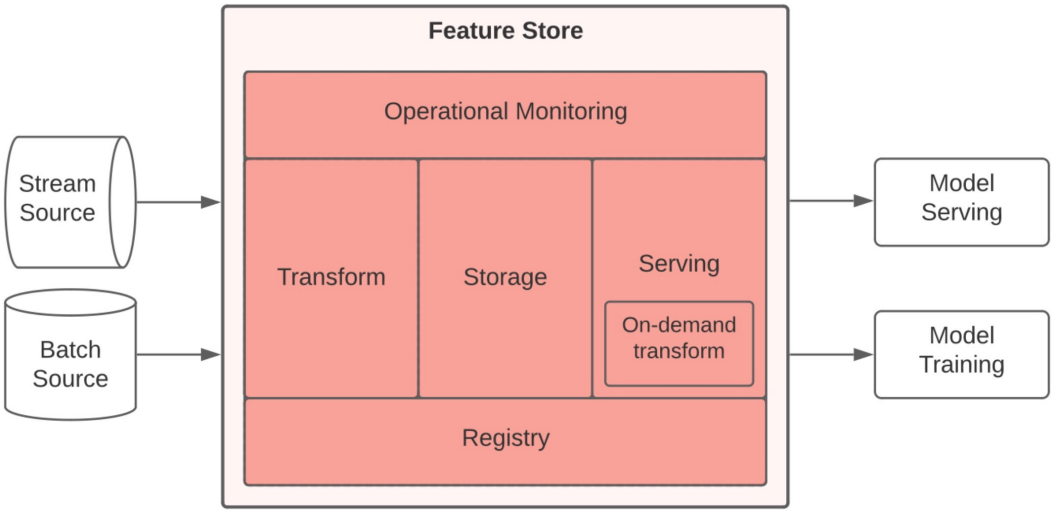




[cf https://dotdata.com/wp-content/uploads/2023/06/Feature-stability_Figure-1-2.svg]

Machine Learning Model Registry

- **Centralised Registry**: A single source of truth for all feature definitions, metadata, and lineage (**only** place for user interactions with feature store)
- **Collaboration**: Enables teams to easily discover, share, and reuse features.
- **Version Control**: Tracks feature versioning, making it easy to debug models and ensure compliance.
- **Automation**: Automates the scheduling of feature ingestion, transformation, and serving jobs.



Why Feature Store in ML Projects?

- **Economies of Scale**: Reusing features across models reduces duplication of effort and accelerates new ML projects.
- **Adaptability**: Feature stores are modular and can be integrated with existing data platforms to simplify operational ML.
- **Impact on Organizations**: Feature stores unlock collaboration across data teams, increase model deployment speed, and streamline ML workflows.

Alternatives to Feature Stores

1. Custom ETL Pipelines
   - **Pro**: High flexibility and control. If using in-store ML, good data security.
   - **Con**: Increased complexity, lacks consistency in serving features across teams.
2. Data Warehouses & Lakes
   - **Pro**: Integrated with existing infrastructures (e.g., S3).
   - **Con**: Not optimised for real-time ML use cases, typically lacks native feature transformation capabilities.
3. In-House Feature Management Systems
   - **Pro**: Complete customisation tailored to business needs.
   - **Con**: High maintenance and scalability concerns as teams grow.

Summary

- We covered two important aspects of Data Engineering Lifecycle:
  - **Data Architecture** design
  - DataOps operation & monitoring of pipeline during development and production
- **DataOps**
  - collaborative data management practices to speed up data delivery and maintain quality
  - Pipeline Orchestration, Data Quality Monitoring, Governance/Security, Data Access
- **Scaling Machine Learning** 把 ML 带到数据那边
  - Approach 1: ML to Data with integrated ML in data stores; example: MADlib for PostgreSQL
  - Approach 2: Serving Data for ML 数据系统只负责"服务"，ML 在外部完成
    - either raw or feature data for training machine learning
    - or serving trained models during deployment
  - **FeatureStore**