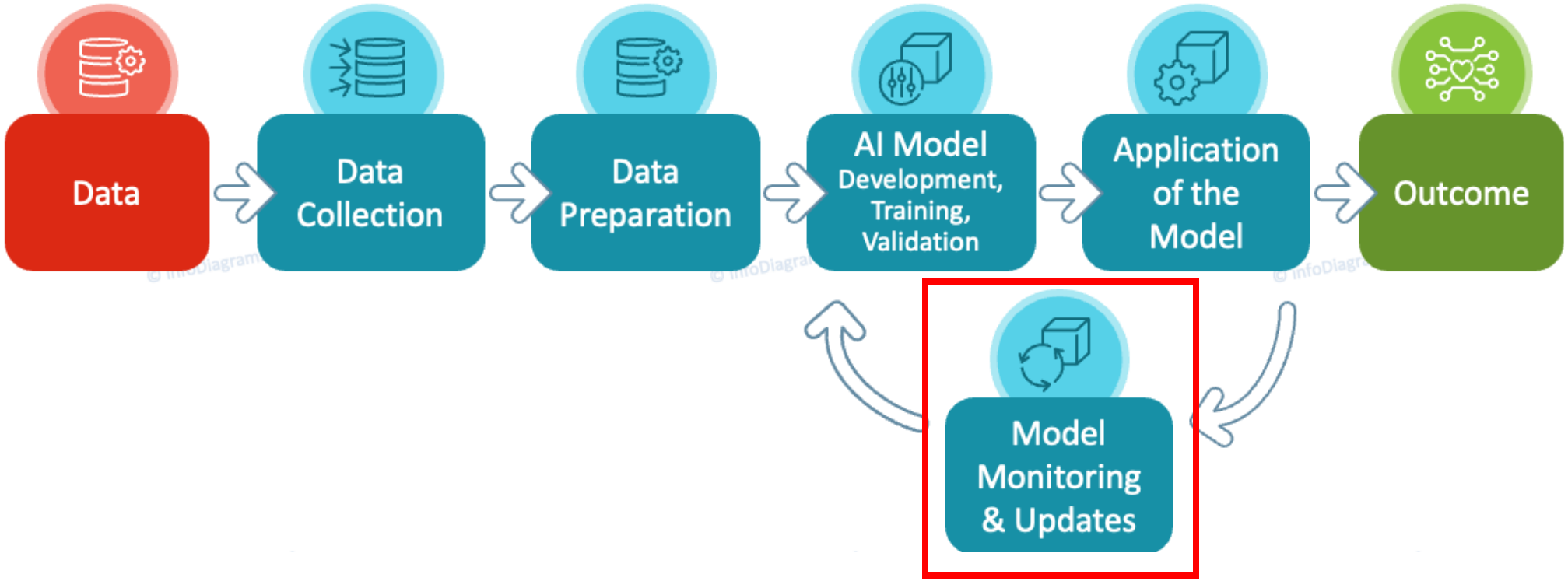
IMEN891K/IDSC501 Data Engineering

16. **ML Systems**

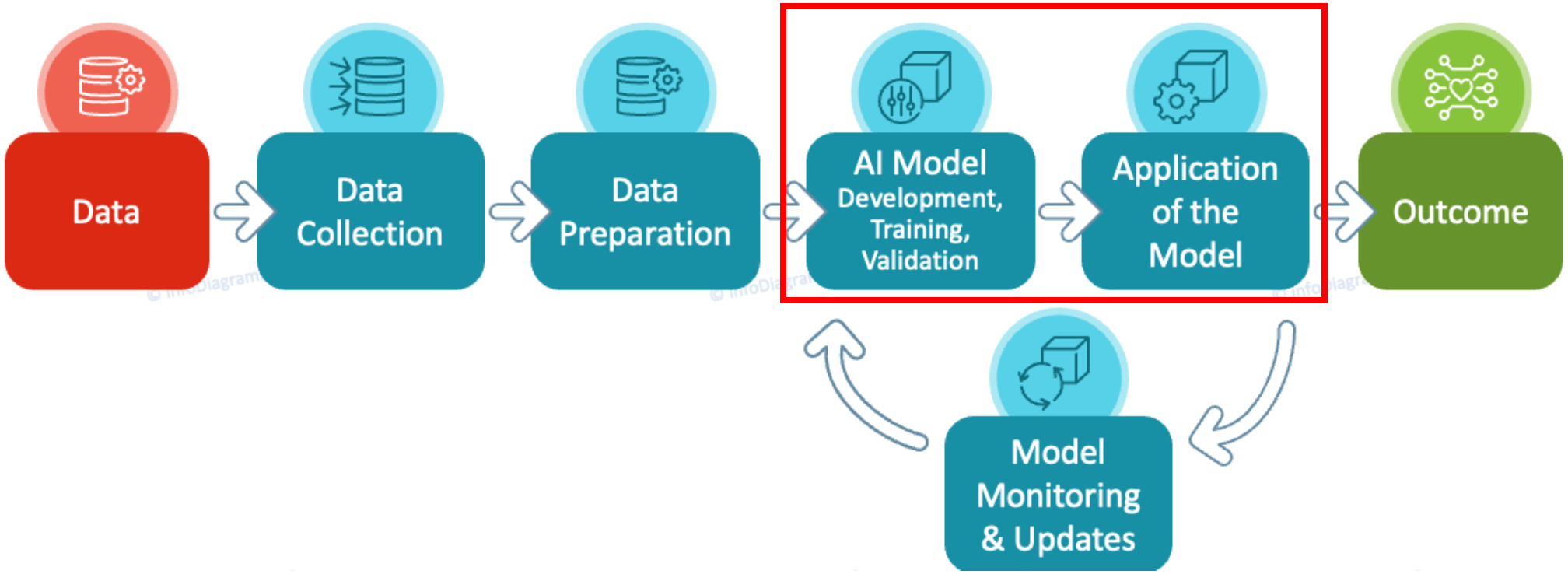
Fall 2025

Instructor: **Sangwoo Mo**

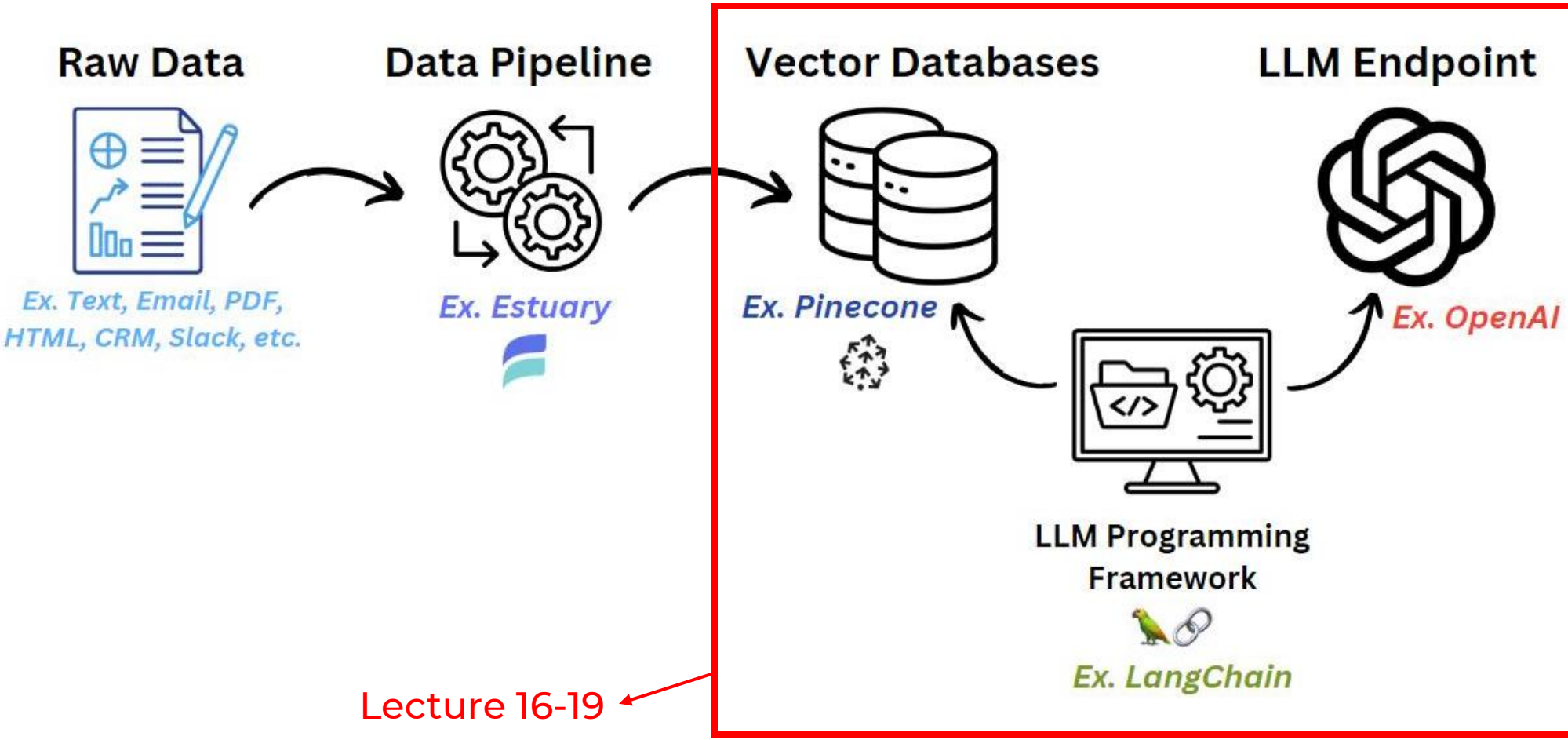
Recap: Course Overview



Advertisement: IMEN574/IDSC574 Programming for Data Science (Spring 2026)

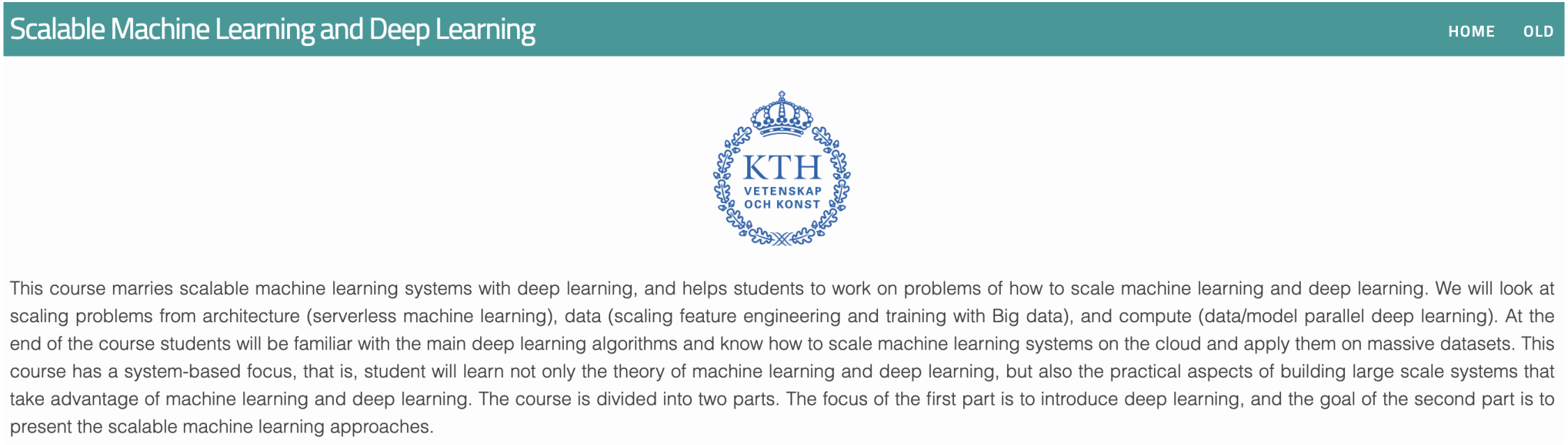


Recap: Course Overview

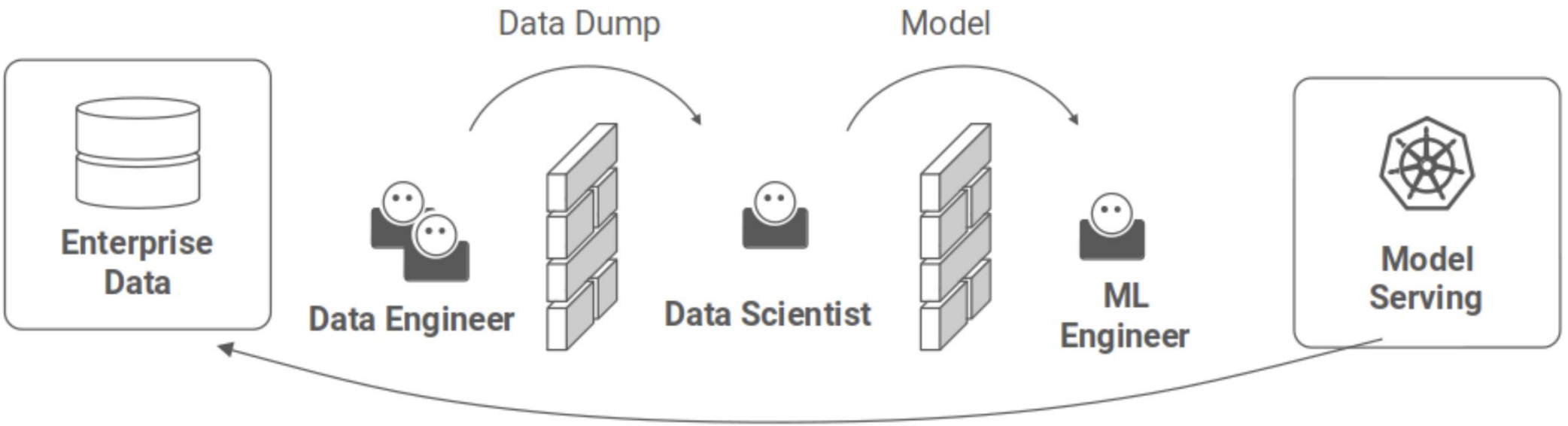


Disclaimer

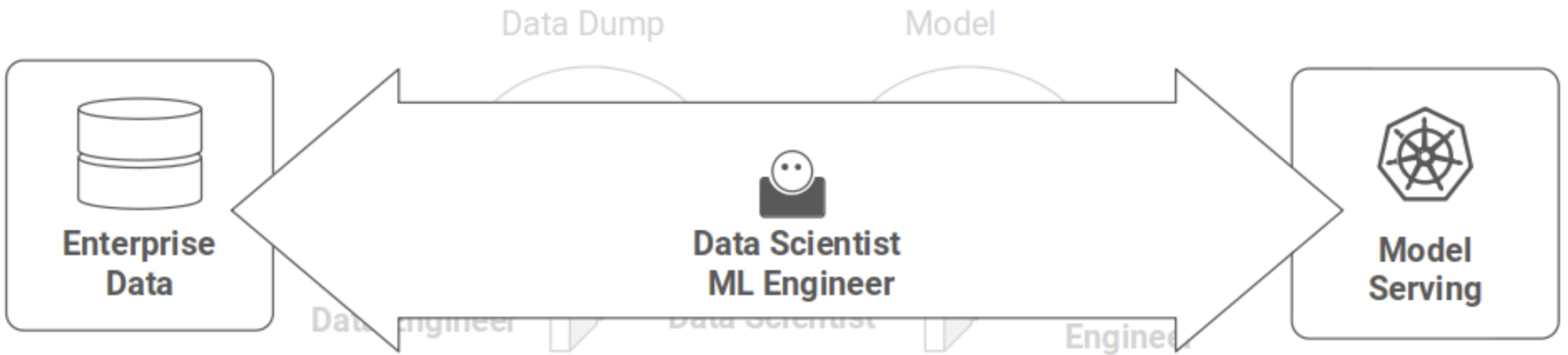
The ML systems part was heavily copy-pasted from KTH ID2223



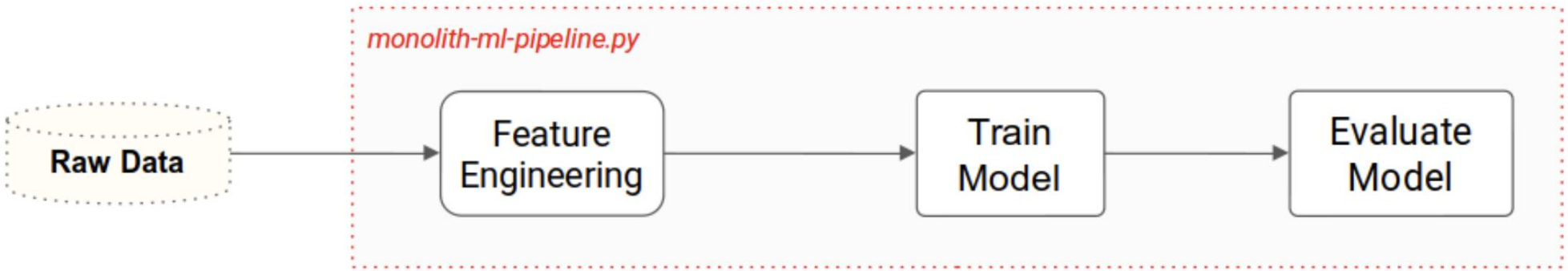
Data Scientist vs ML Engineer



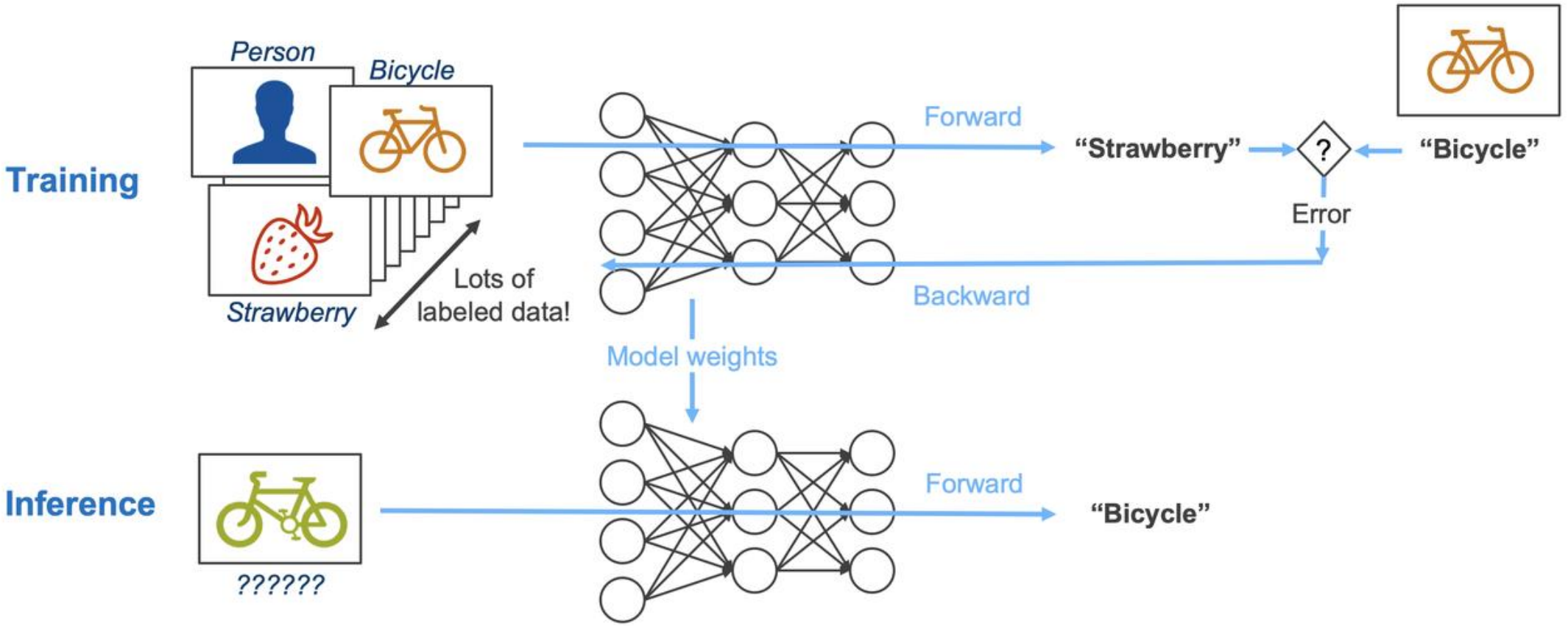
Data Scientist vs ML Engineer



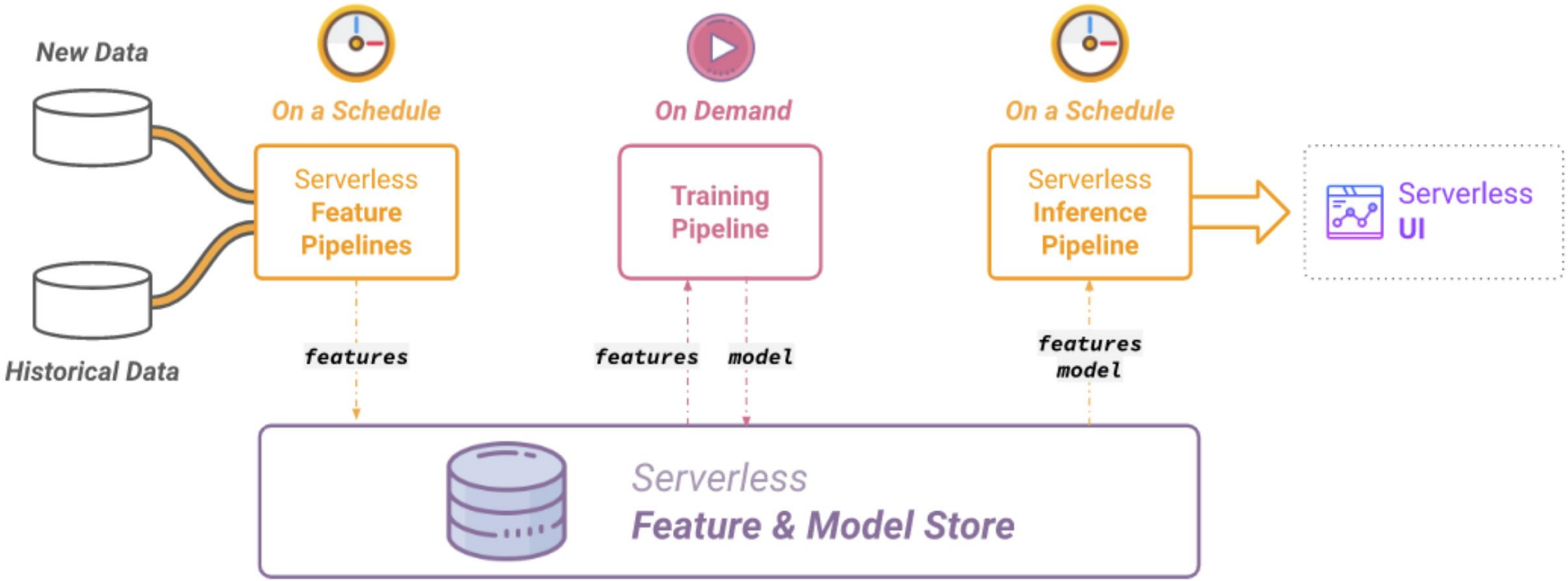
ML Pipeline


monolith-ml-pipeline.py
Raw Data
Feature Engineering
Train Model
Evaluate Model

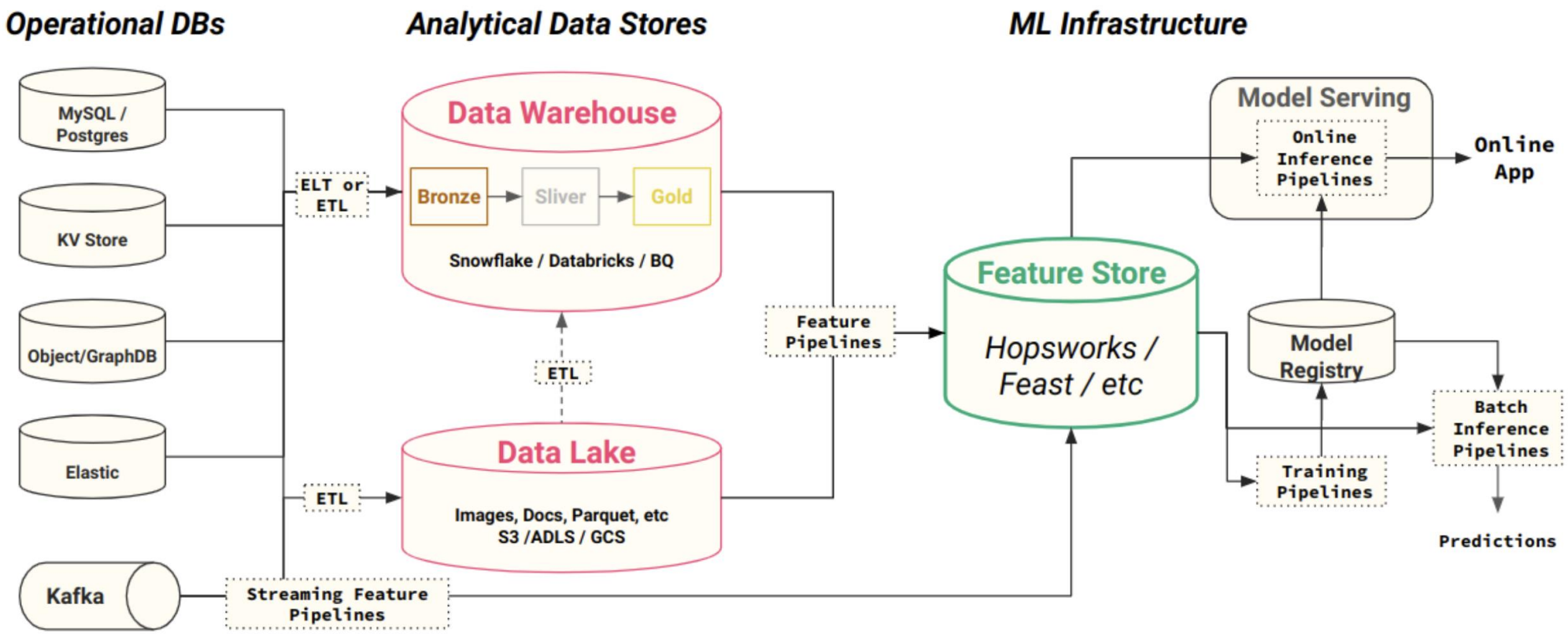
ML Pipeline


Training
Person
Bicycle
Strawberry
Lots of
labeled data!
Forward
"Strawberry"
?
"Bicycle"
Error
Backward
Model weights
Inference
??????
Forward
"Bicycle"

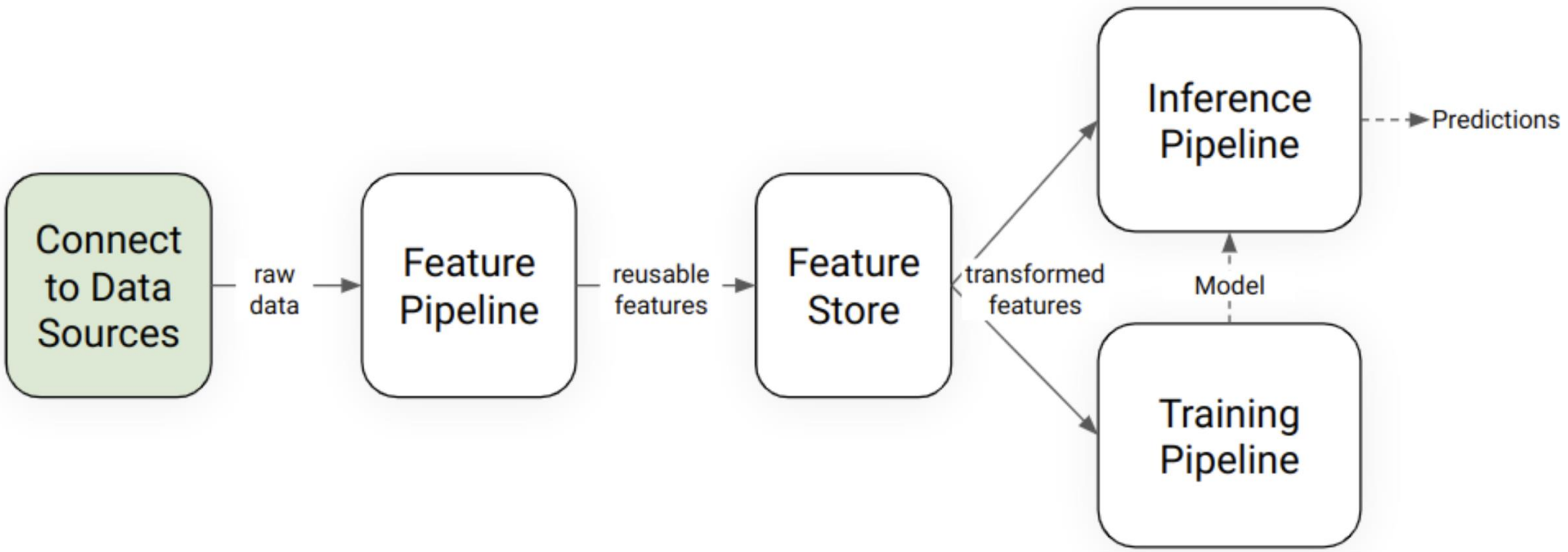
ML Pipeline at Scale



ML Pipeline at Scale



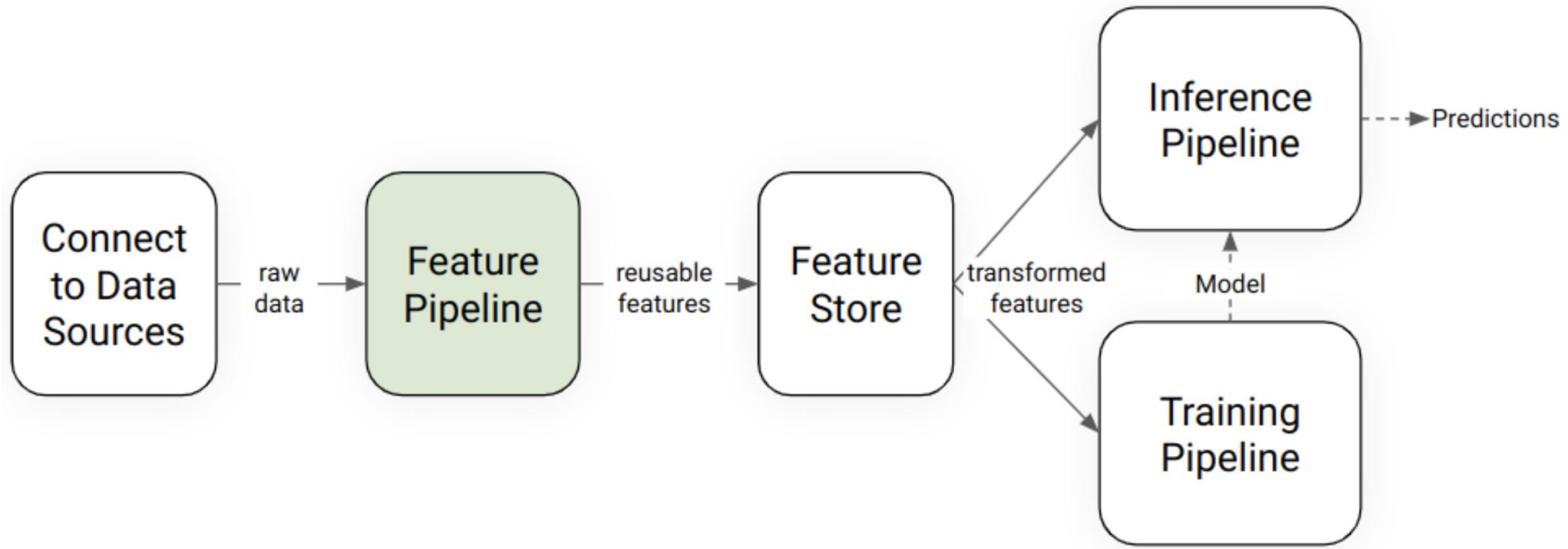
ML Pipeline Stages



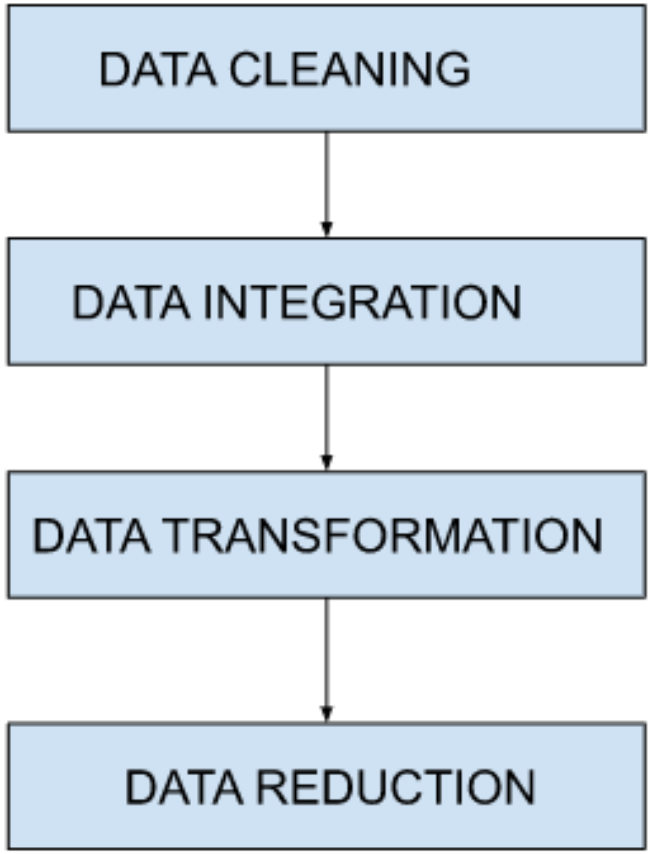
Recap: Data Collection

Type of Data	File Formats	Example Systems
Tabular data	.csv, .parquet, .tfrecord, .avro	Snowflake, Databricks, BQ, Redshift, S3, ADLS, GCS
Unstructured data	images, sound, video	S3, GCS, ADLS, HDFS
Free-text search	application/service logs	Elasticsearch, Solr
Documents	JSON	MongoDB
Graph data	Social networks	Neo4J
Time-series data	Performance metrics	InfluxDB, Prometheus
Queued data	Avro	Kafka, Kinesis
REST APIs	REST API with API key	Saas Platform
Web scraped	N/A	Websites publishing data

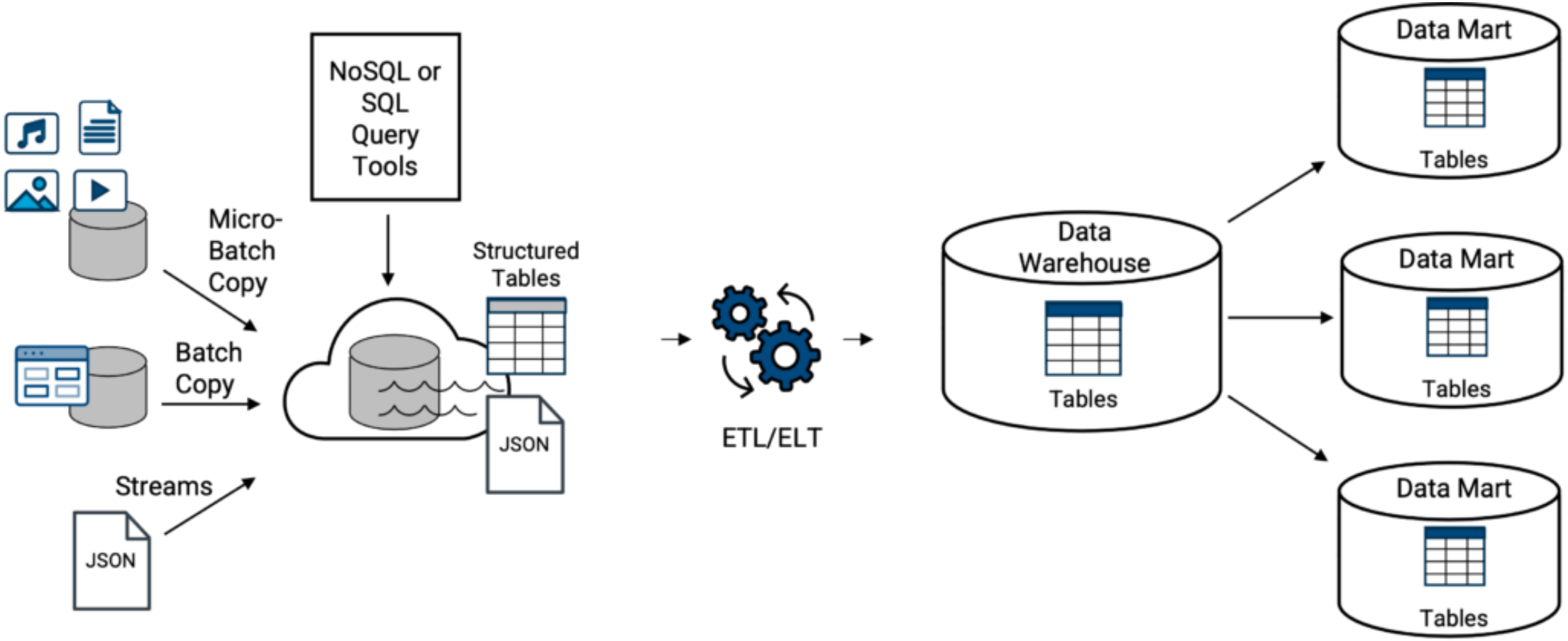
ML Pipeline Stages



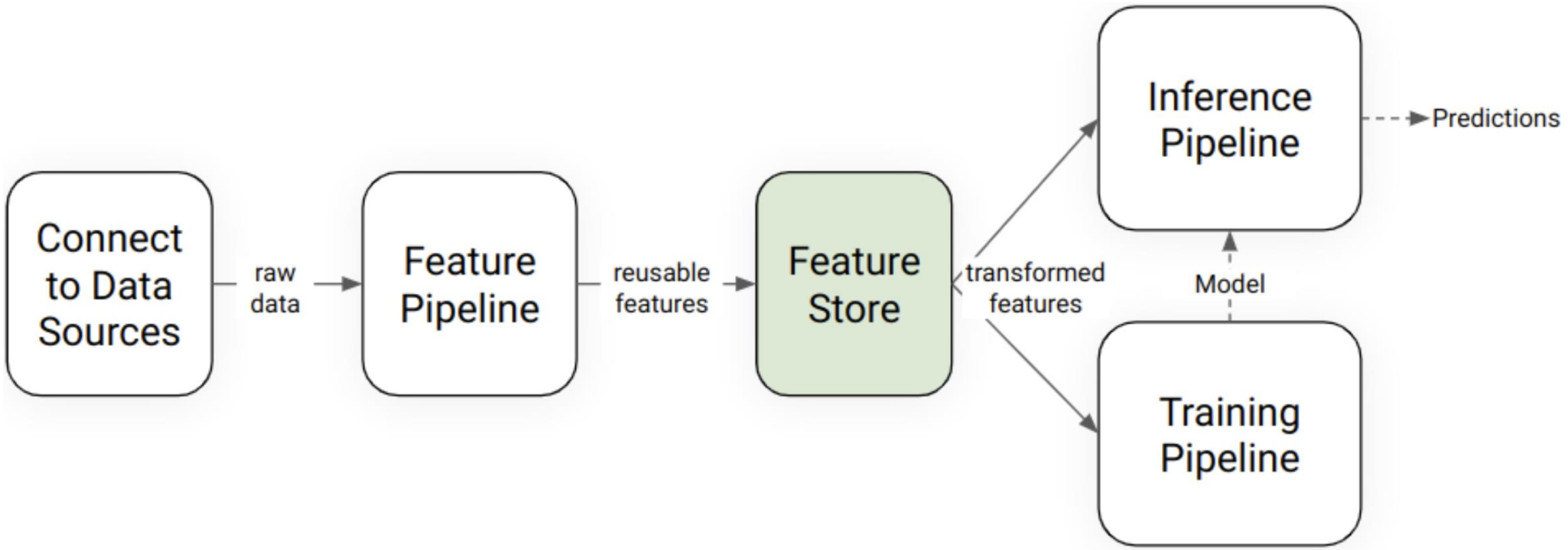
Recap: Data Pre-processing



Recap: Unstructured → Structured Data



ML Pipeline Stages



Recap: Data Storage

Data for a prediction service to identify if a credit card transaction is **suspected of fraud or not.**

transactions table

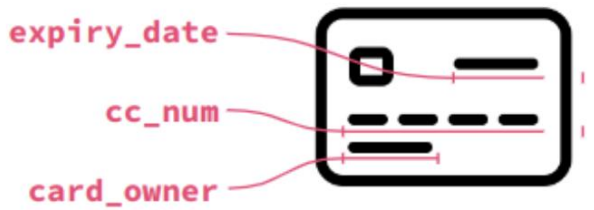


user_profile table

Name
Sex
Date of Birth
City



credit_card_details table



Recap: Data Storage – Tables



Updated
once/hour

tid	cc_num	datetime	category	amount	long/lat	fraud
1	11111	2022-09-01 11:24	food	45.33	53N,6W	No
2	11111	2022-09-02 09:17	clothing	183.12	52N,6W	No
3	11111	2022-09-04 19:33	entertain	63.33	51N,7W	yes
..	…	…	…	…	…	…



Updated
once/week

credit_cards

cc_num	provider	expires
11111	visa	24/05
…	…	…

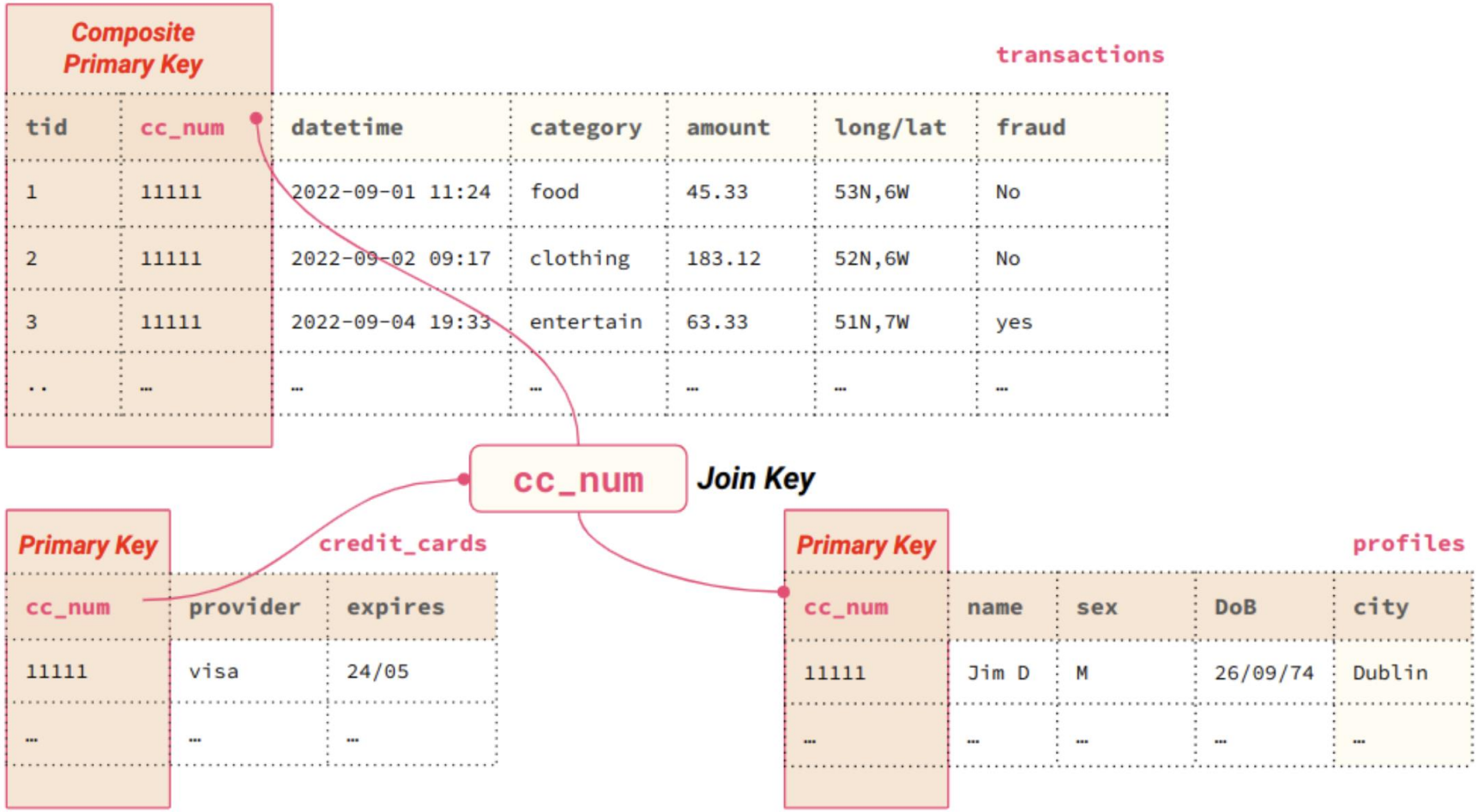


Updated
once/day

profiles

cc_num	name	sex	DoB	city
11111	Jim D	M	26/09/74	Dublin
…	…	…	…	…

Recap: Data Storage – Tables



Recap: Data Storage – One Big Table

Key	Primary Key	Event Time	Features			Label	Features			
tid	cc_num	datetime	category	amount	long/lat	fraud	days_until_card_expires	age_at_transaction	sex	lives_city
1	1111 ...	2022-09-01 11:24	food	45.33	53N,6W	No	1011	47	M	Dublin
2	1111 ...	2022-09-02 09:17	clothing	183.12	52N,6W	No	1010	47	M	Dublin
3	1111 ...	2022-09-04 19:33	entertain	63.33	51N,7W	yes	1008	47	M	Dublin
..	...	...	...	...	...	...	...	...	...	...

Recap: Data Storage – One Big Table

Our main Feature group is `transactions`. The columns for this one big table (OBT) have different data sources, that are updated at different cadences. It is inefficient to update all columns in every update, but less JOINs will be required for training data.

Updated Frequently | *Updated Less Frequently*

Source: cc_transactions							Source: credit_card	Source: profile		
tid	cc_num	datetime	category	amount	long/lat	fraud	days_until_card_expires	age_at_transaction	sex	lives_city
1	1111 ...	2022-09-01 11:24	food	45.33	53N,6W	No	1011	47	M	Dublin
2	1111 ...	2022-09-02 09:17	clothing	183.12	52N,6W	No	1010	47	M	Dublin
3	1111 ...	2022-09-04 19:33	entertain	63.33	51N,7W	yes	1008	47	M	Dublin
..	...	...	...	...	...	...	...	...	...	...

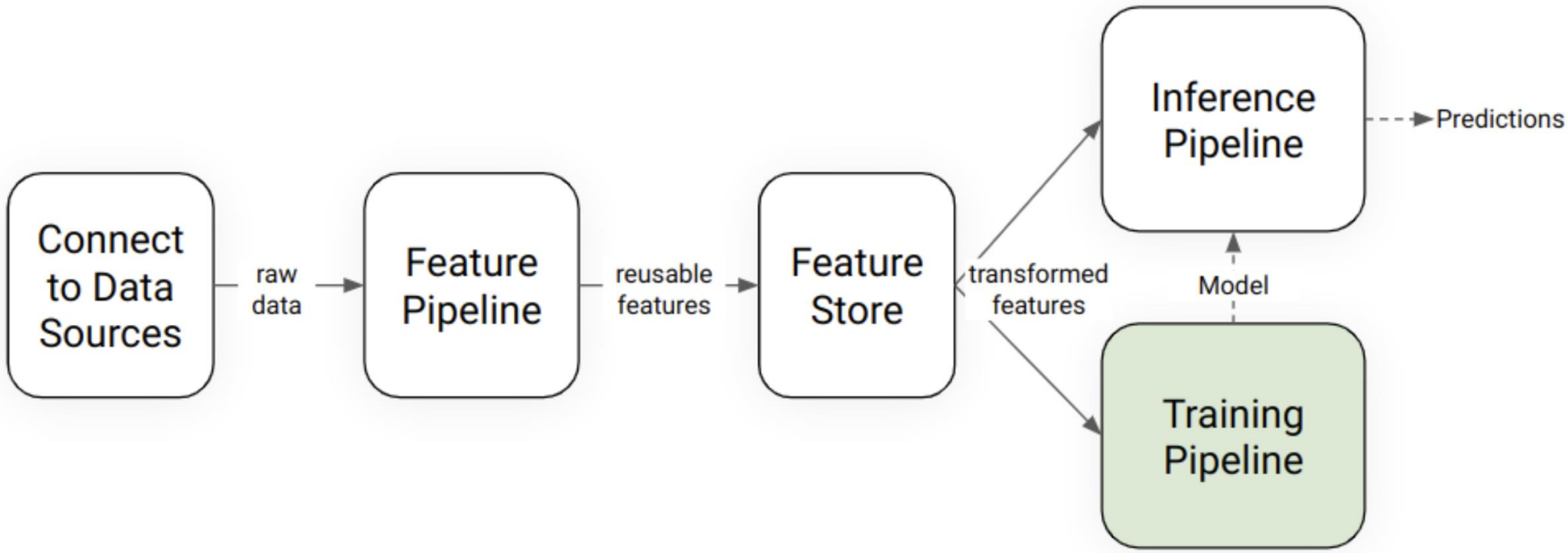
Recap: Data Storage – One Big Table

`transactions_4h_aggs` contains aggregated features computed over a 4h time window for each credit card

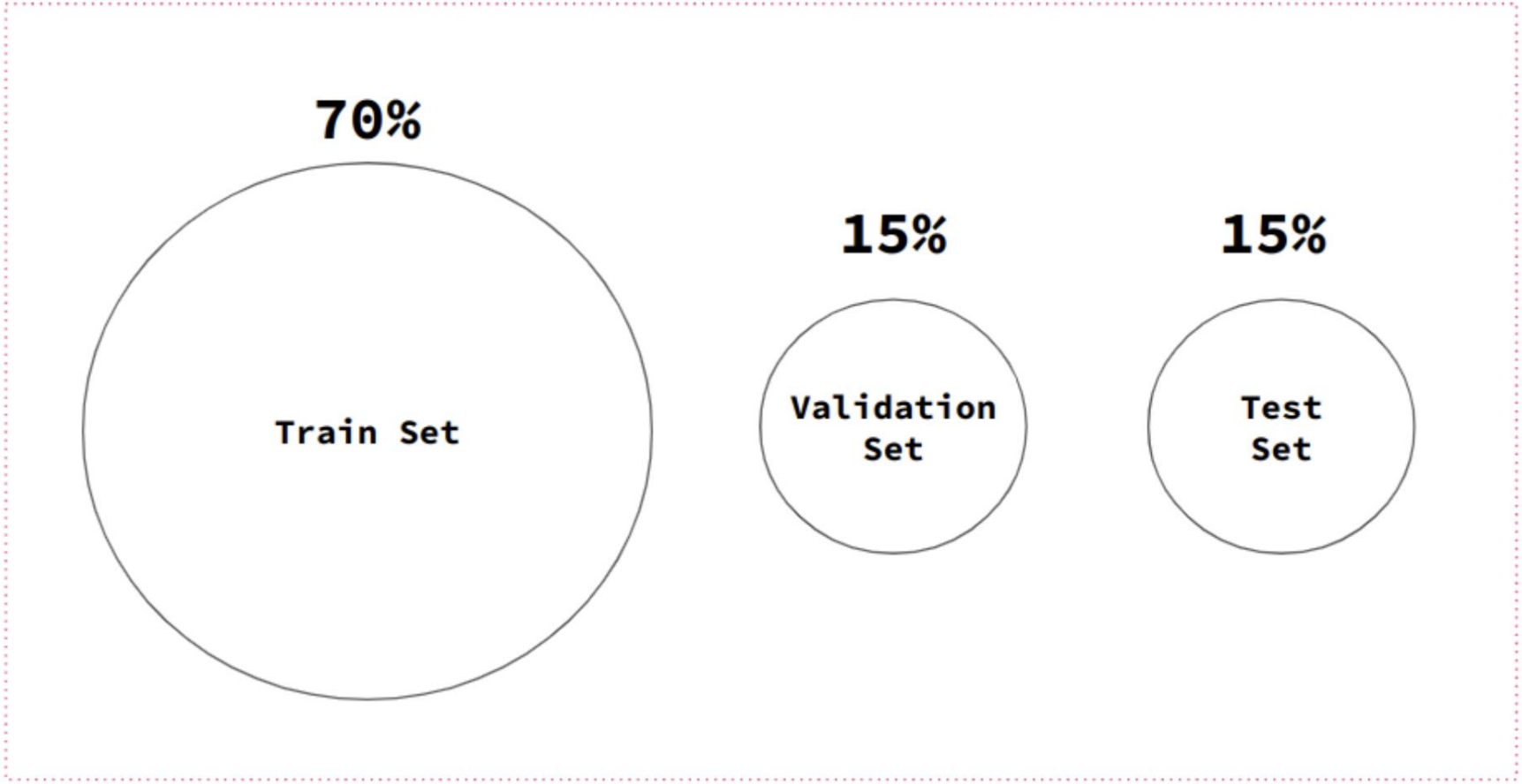
Primary Key *Event Time* *Features*

cc_num	datetime	loc_delta_mavg	trans_volume_mstd	trans_volume_mavg	trans_freq
1111 ...	2022-09-01 11:24	53N,6W	3.4	8	6
1111 ...	2022-09-02 09:17	52N,6W	3.6	3	5
1111 ...	2022-09-04 19:33	51N,7W	4.1	33	45
...	...	...	...	...	...

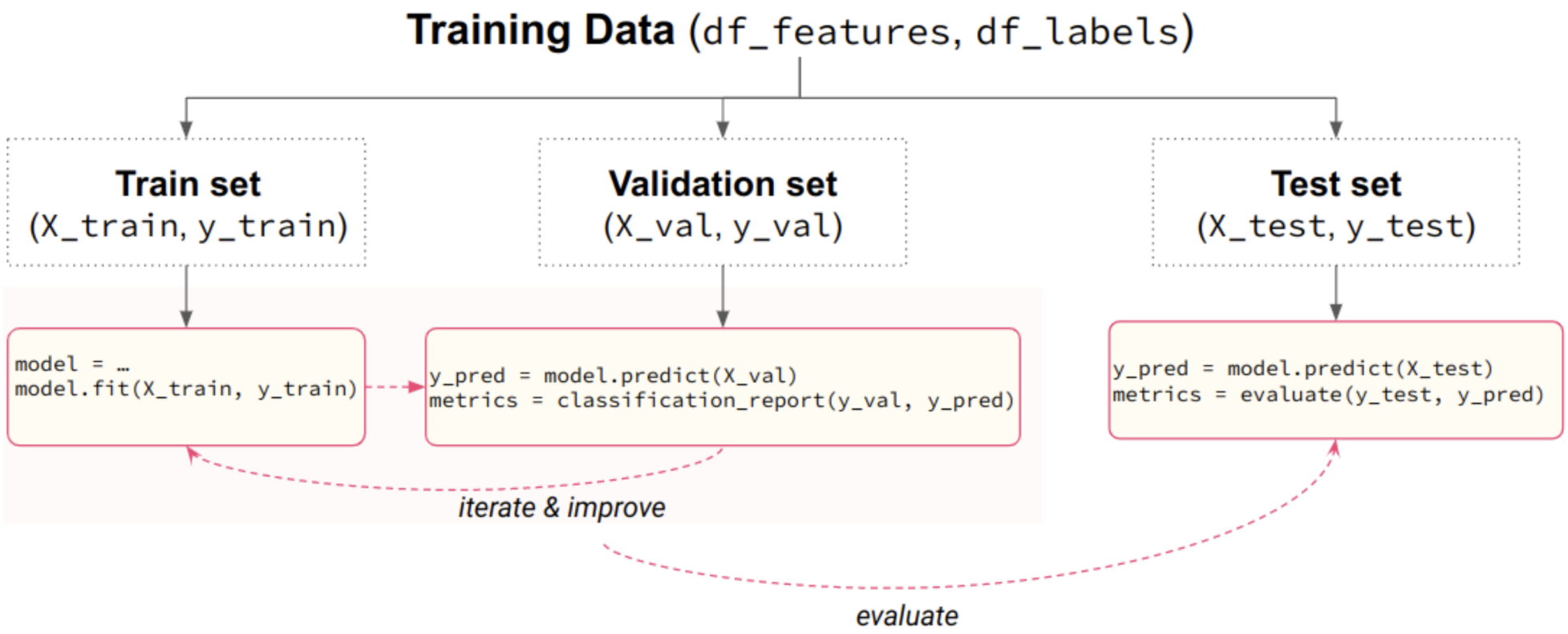
ML Pipeline Stages



Model Training is an Iterative Process



Model Training is an Iterative Process



Model Training is an Iterative Process

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import xgboost as xgb

X_train,X_test,y_train,y_test =
   train_test_split(features, labels, test_size=0.2)

model = xgb.XGBClassifier()

model.fit(X_train,y_train)

y_pred = model.predict(X_test)

report_dict = classification_report(
              y_test, y_pred, output_dict=True)
```

Get train and test data sets as features (X) and labels (y)

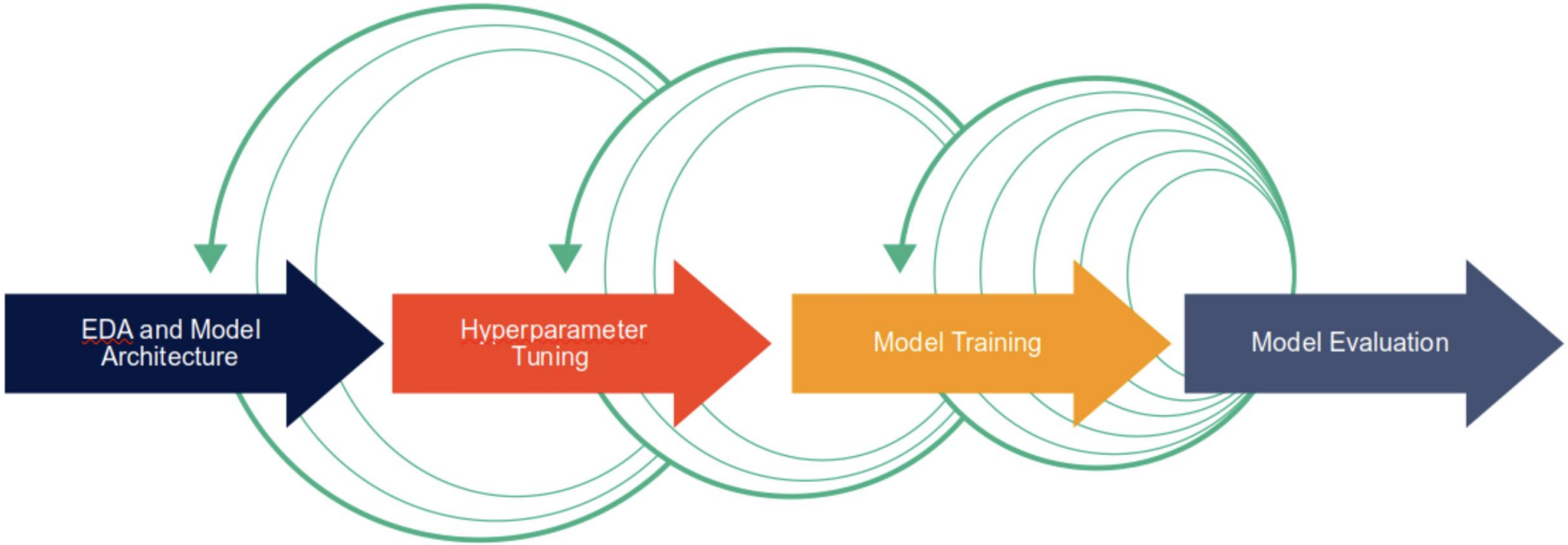
Use XGBoost as modelling algorithm

Train supervised ML classifier with features and labels from train set

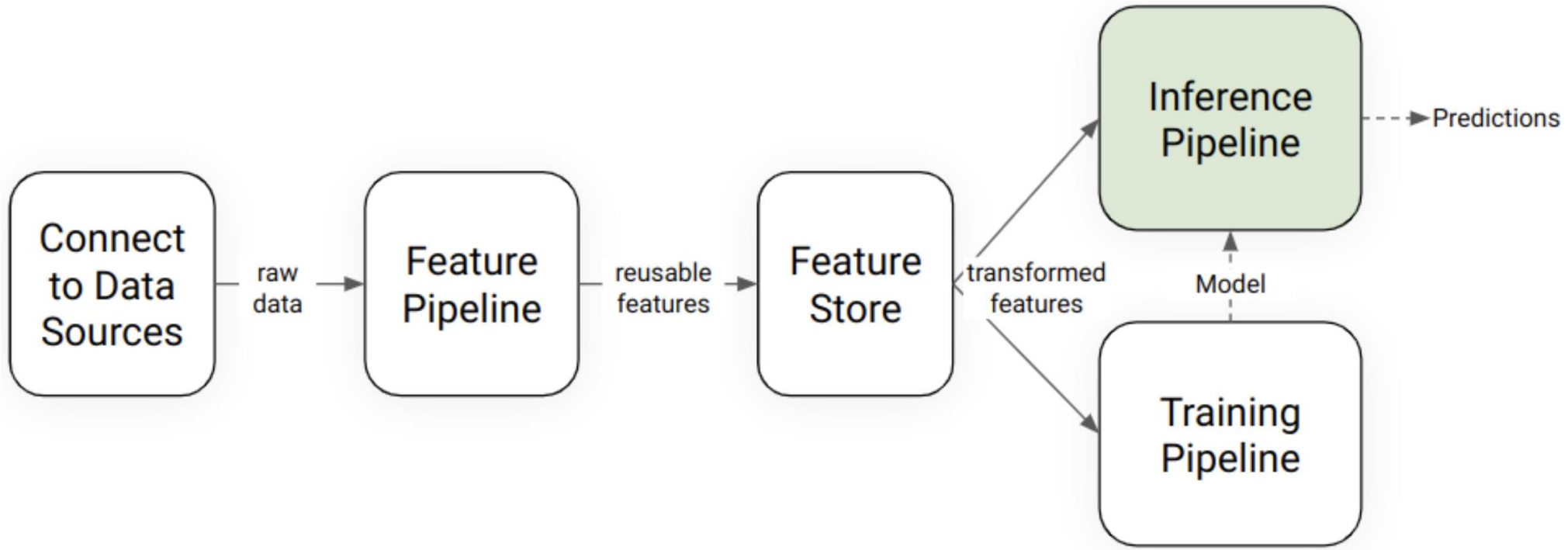
Generate predictions with model on test features (X_test)

Evaluate model performance by comparing predictions (y_pred) and labels (y_test) for the test set

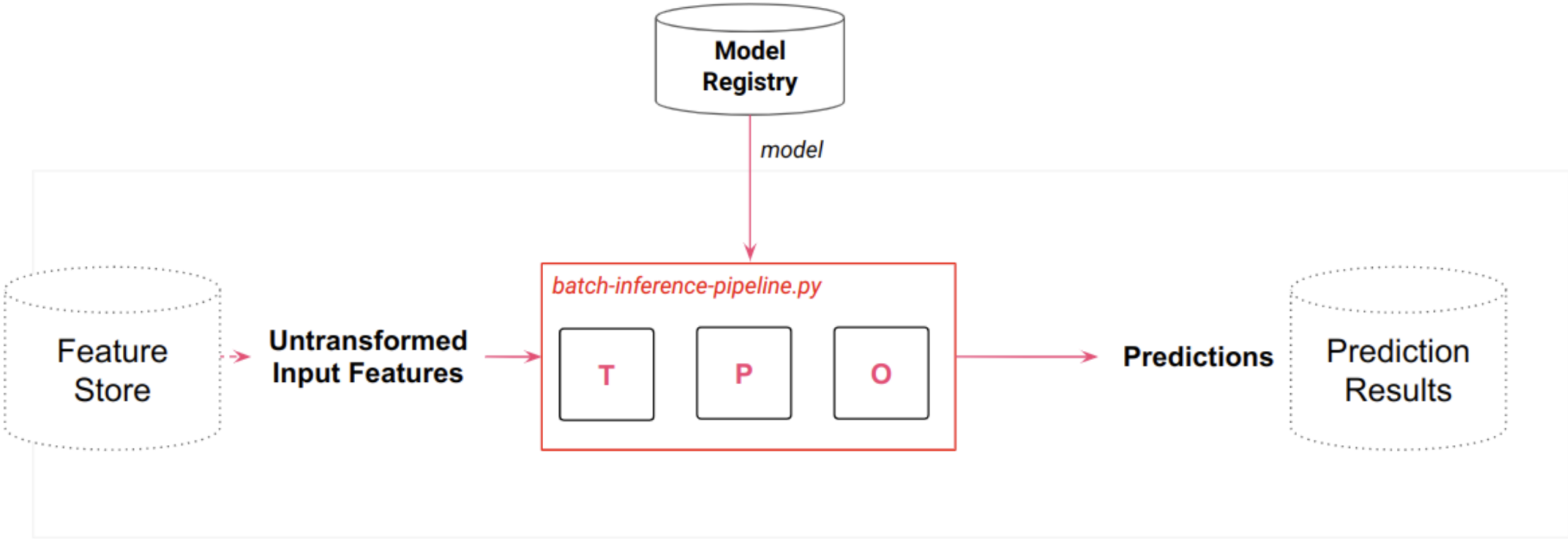
Model Training is an Iterative Process



ML Pipeline Stages

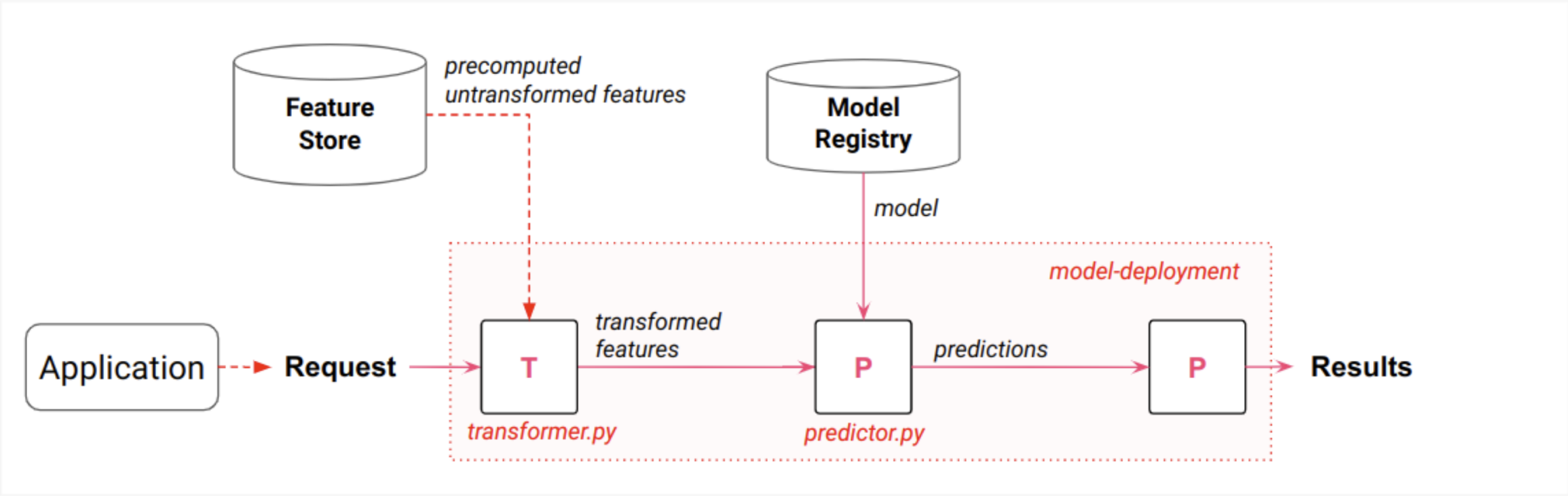


Batch Inference



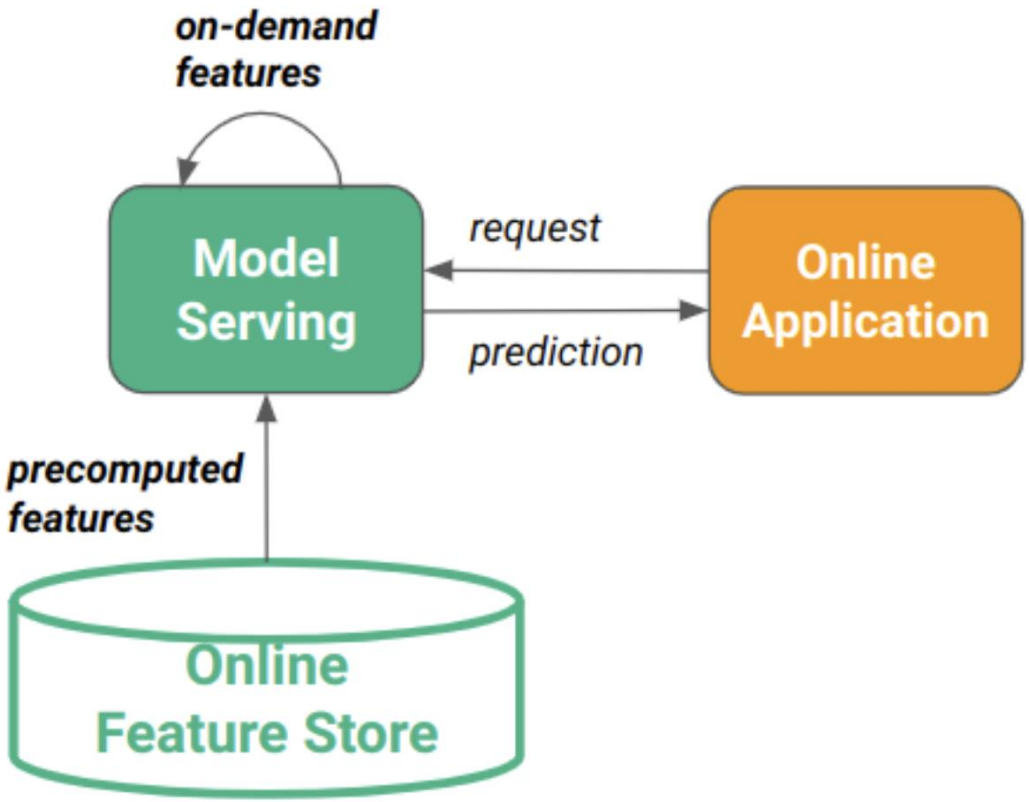
`TPO = Transform features, Predict, Output.`

Online Inference

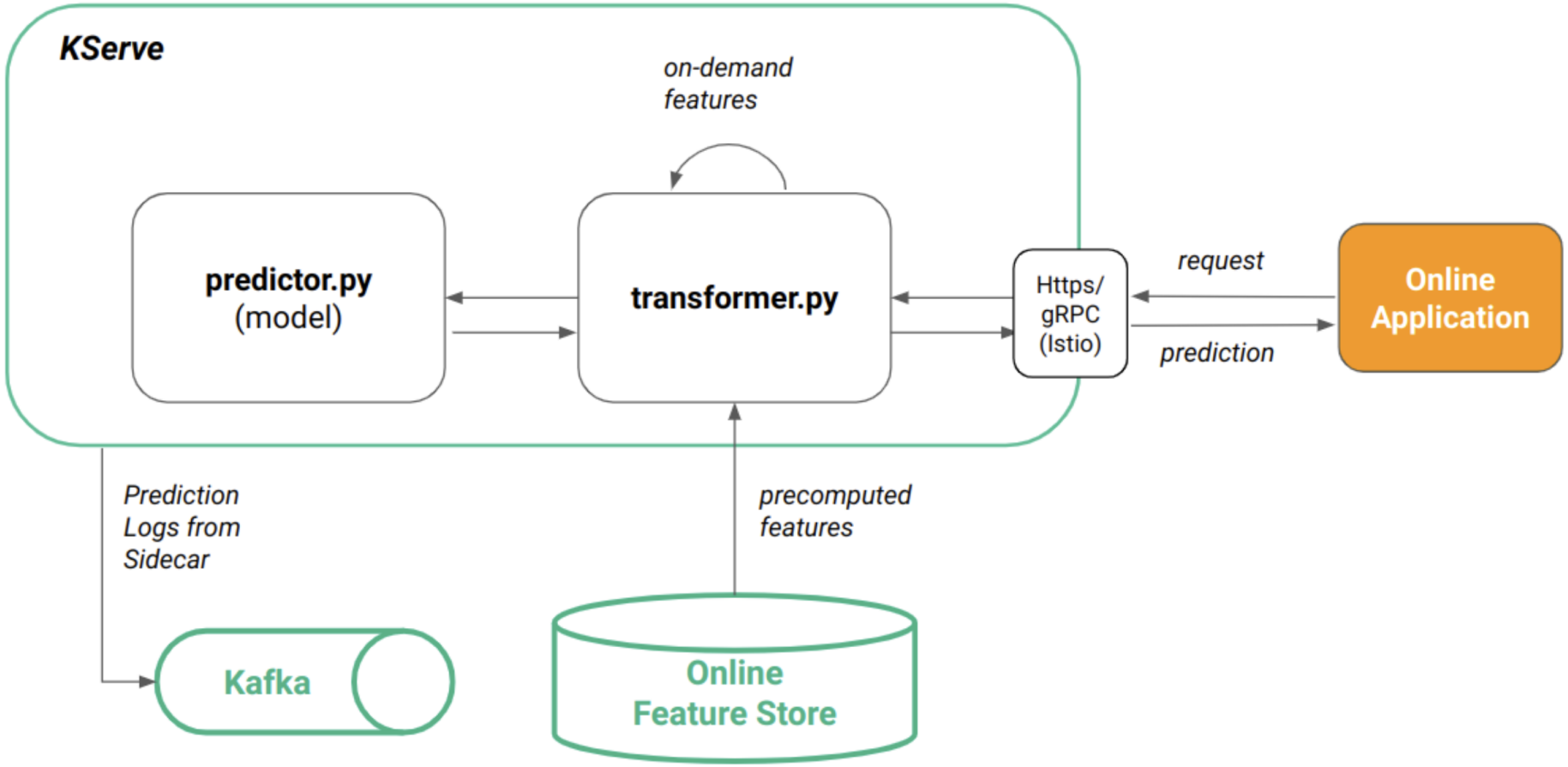


TPP = Transform the input request into features, Predict using input features and the model, Post-process predictions, before output results.

Real-time ML Systems

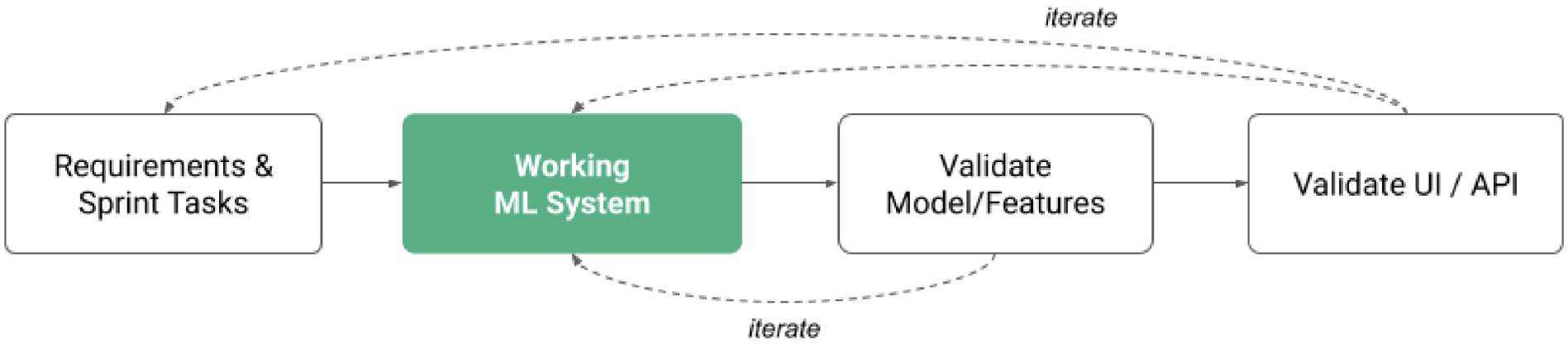


Real-time ML Systems



MLOps

- A goal of MLOps is to improve both iteration speed and quality when developing ML



MLOps

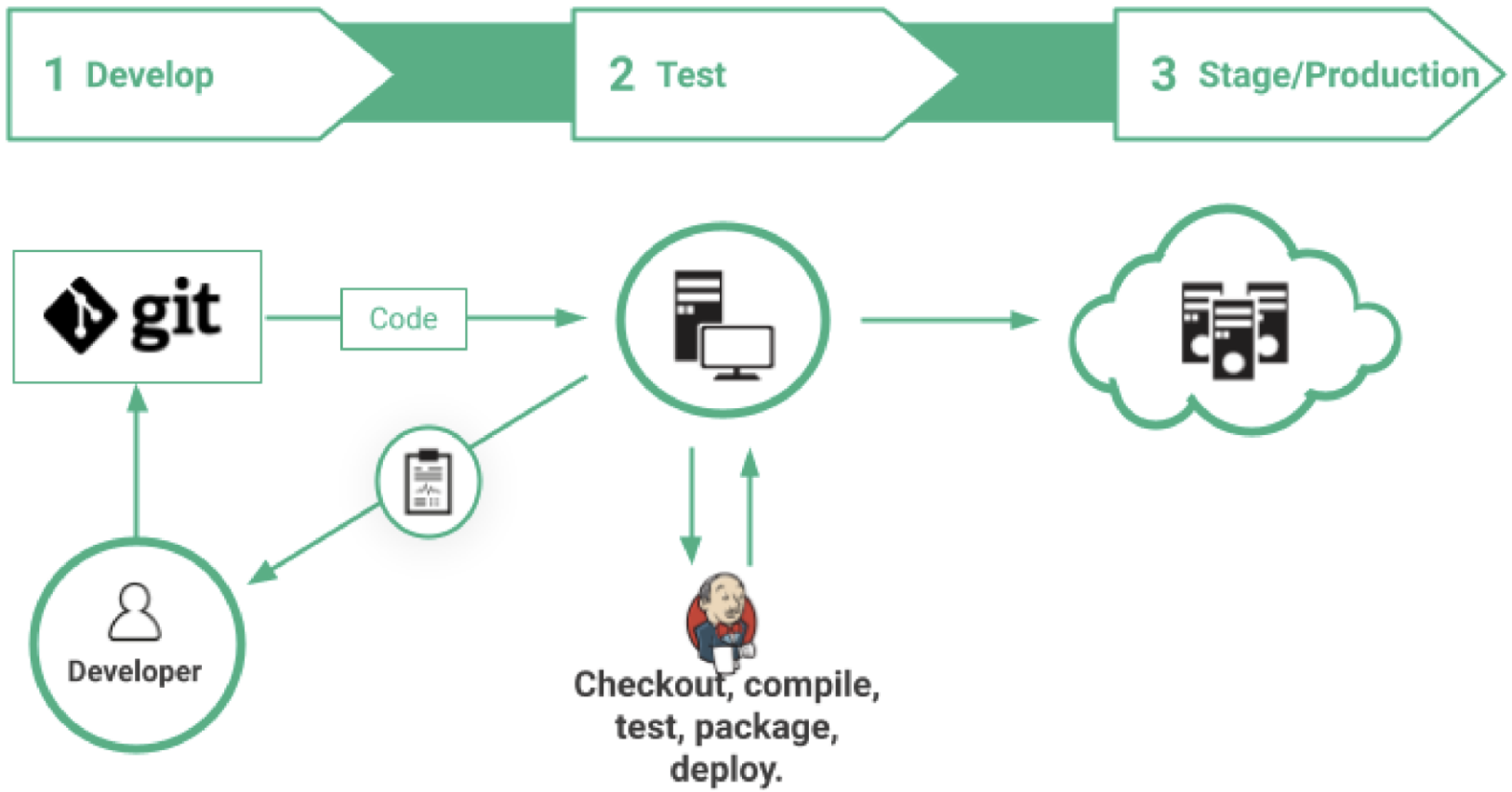
- ML-enabled products **evolve over time**:
  - The available input data (features) change over time
  - The target you are trying to predict changes over time
  - With the help of automation, how can quickly and reliably develop, test, and deploy ML-enabled products without affecting their ongoing operation?
- We should aim to **automate the testing and deployment** of ML-enabled Products

MLOps

- Safe incremental updates: make small changes to your source code with confidence that your changes will not break anything (downstream clients, deployments on different platforms), performance regressions, etc)
- Tighter iteration loop: the time taken to run tests or experiments should not dominate the time taken to make the source code changes
- A faster iteration loop makes developers happier and more productive

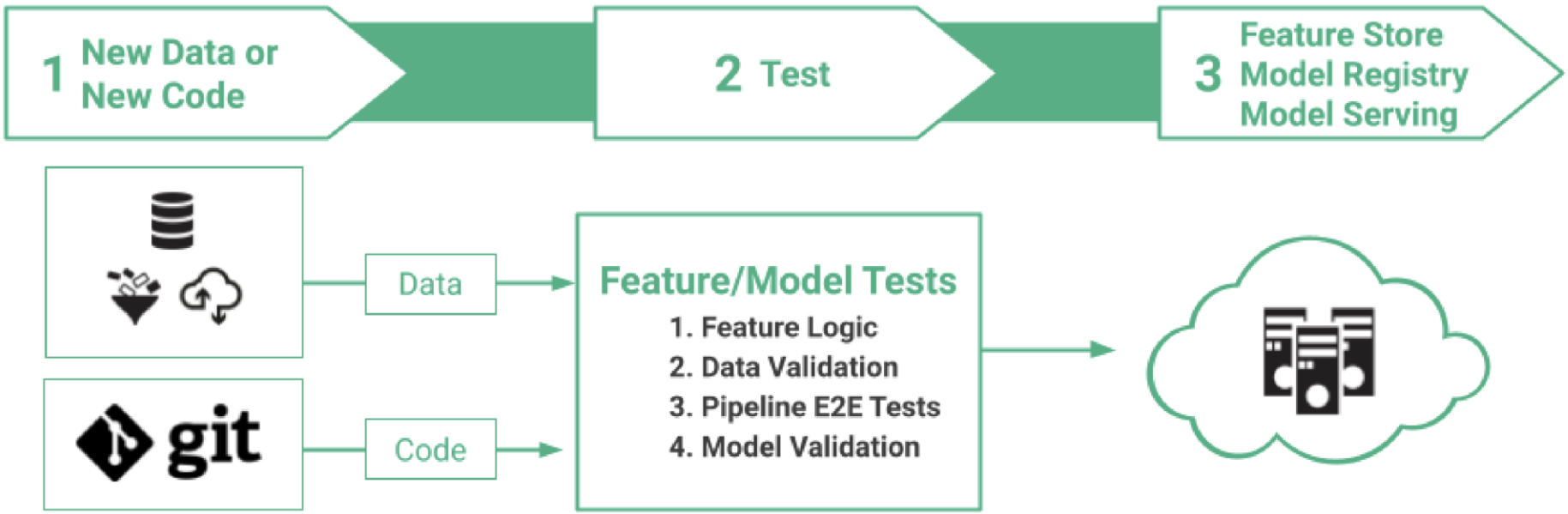
DevOps vs MLOps

- DevOps is a set of practices, tools, and a cultural philosophy that automate and integrate the processes between software development and IT teams. Key technologies are **version control, automated testing, versioning** of production deployments.



DevOps vs MLOps

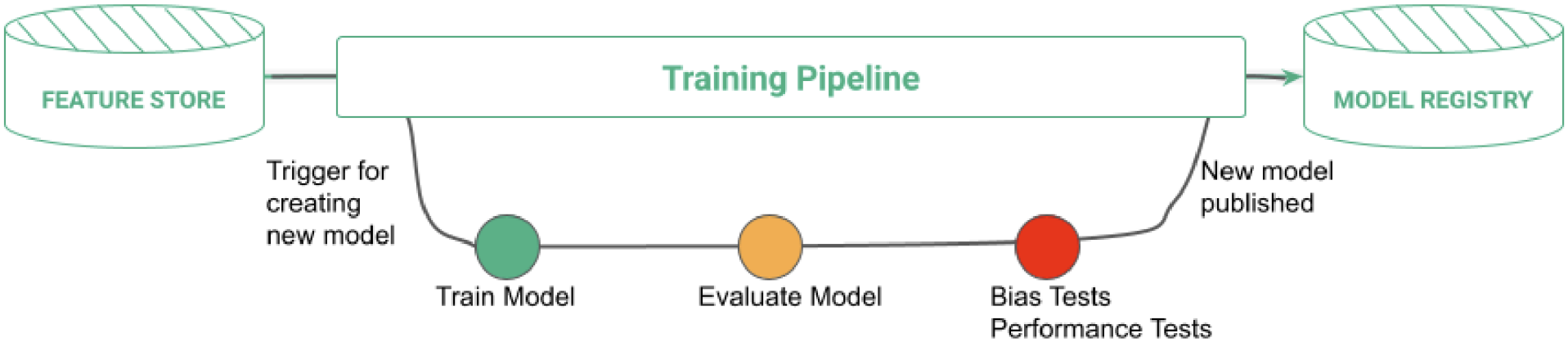
- In DevOps, changes in source code trigger automated testing and deployment.
- In MLOps, changes in either source code or incoming data trigger automated testing and deployment.



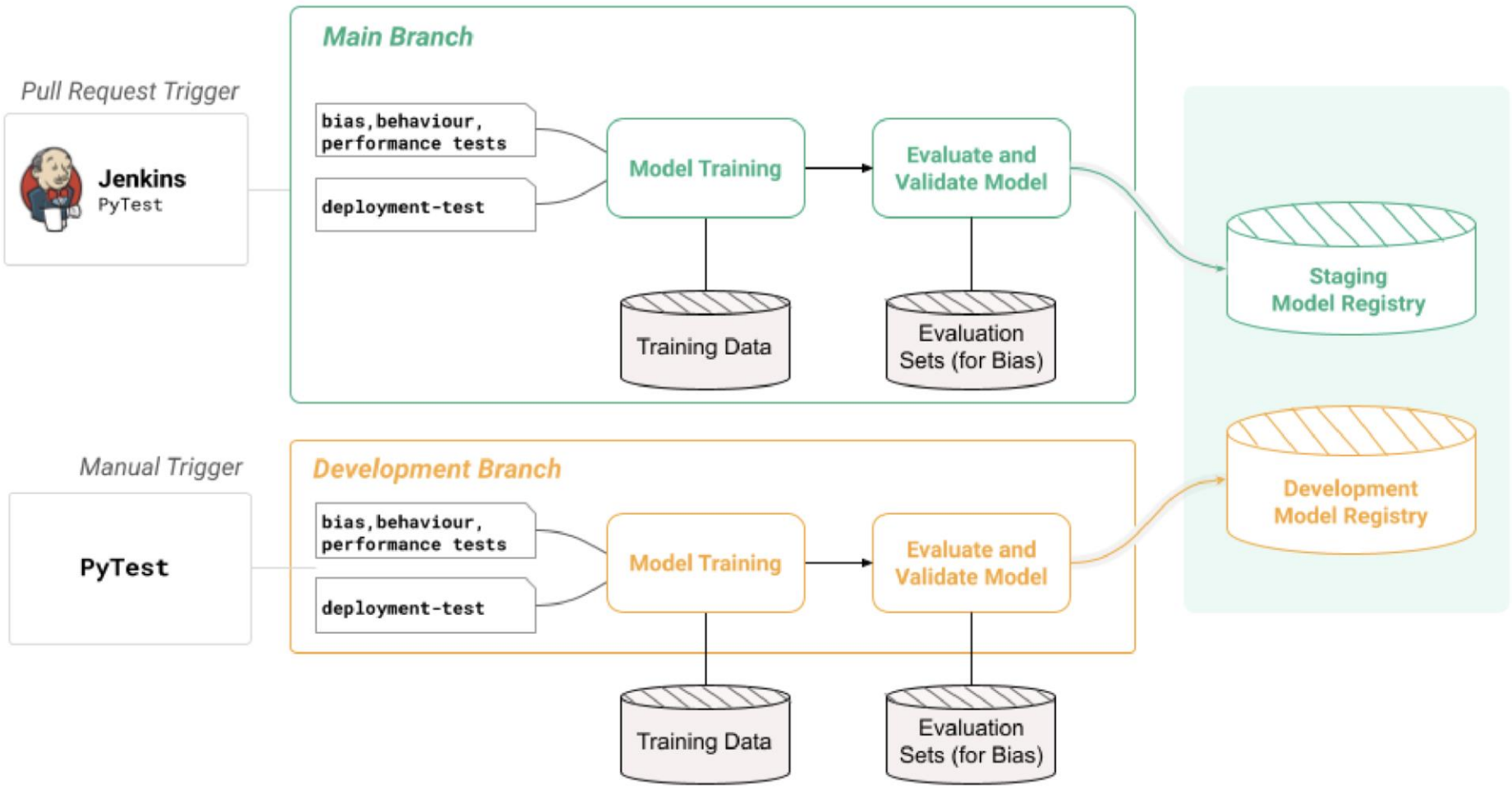
1. Make Everything Reproducible

Model Development

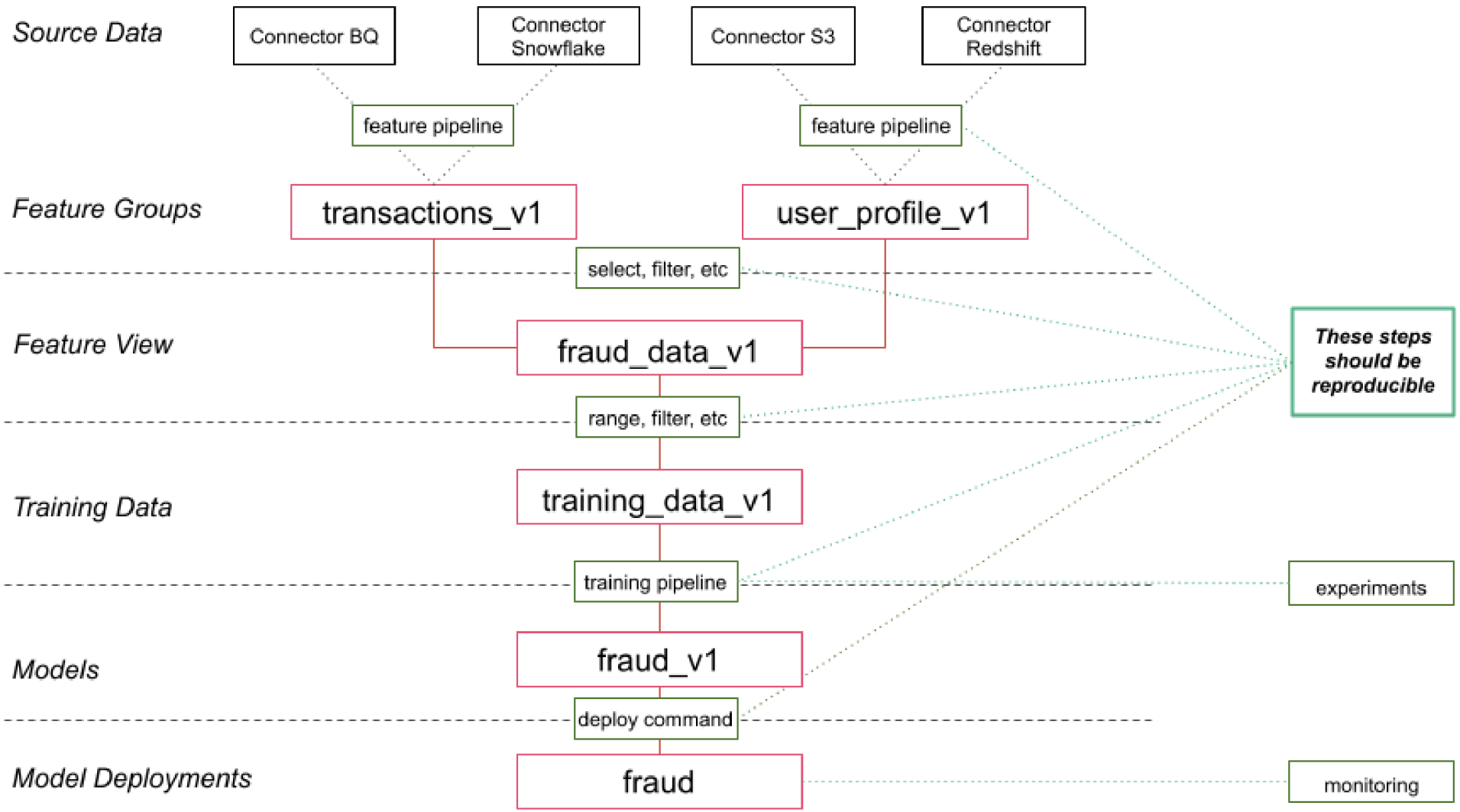
- Save the hyperparams and training data used to train the model,
- Only deploy a model that outperforms an existing model deployment (or baseline), assuming both trained on same dataset.
- Only deploy a model if it is free from bias and trained on ethical data.



Model Development



Reproducible ML



Reproducible ML

ML Asset	Schema Versions	Data Versions	How to reproduce
Feature Groups	Yes	Hudi Commits	Re-run the feature pipeline with the same backfill data
Feature Views	Yes	No	Re-run the same feature view creation commands
Training Data	Yes	No	The parent feature view can recreate training datasets.
Models	Yes	No	Re-run the training pipeline with the same training data,hyperparams, and random number seeds
Model Deployments	No	No	Deploy the same model name/version with the same transformer/predictor code
Prediction Logs	No	No	Not possible if model is stochastic. Emulate by re-running inference pipeline on prediction requests.

Schema Versioning

```
from hsfs.feature import Feature

features = [
    Feature(name="id",type="int",online_type="int"),
    Feature(name="name",type="string",online_type="varchar(20)")]

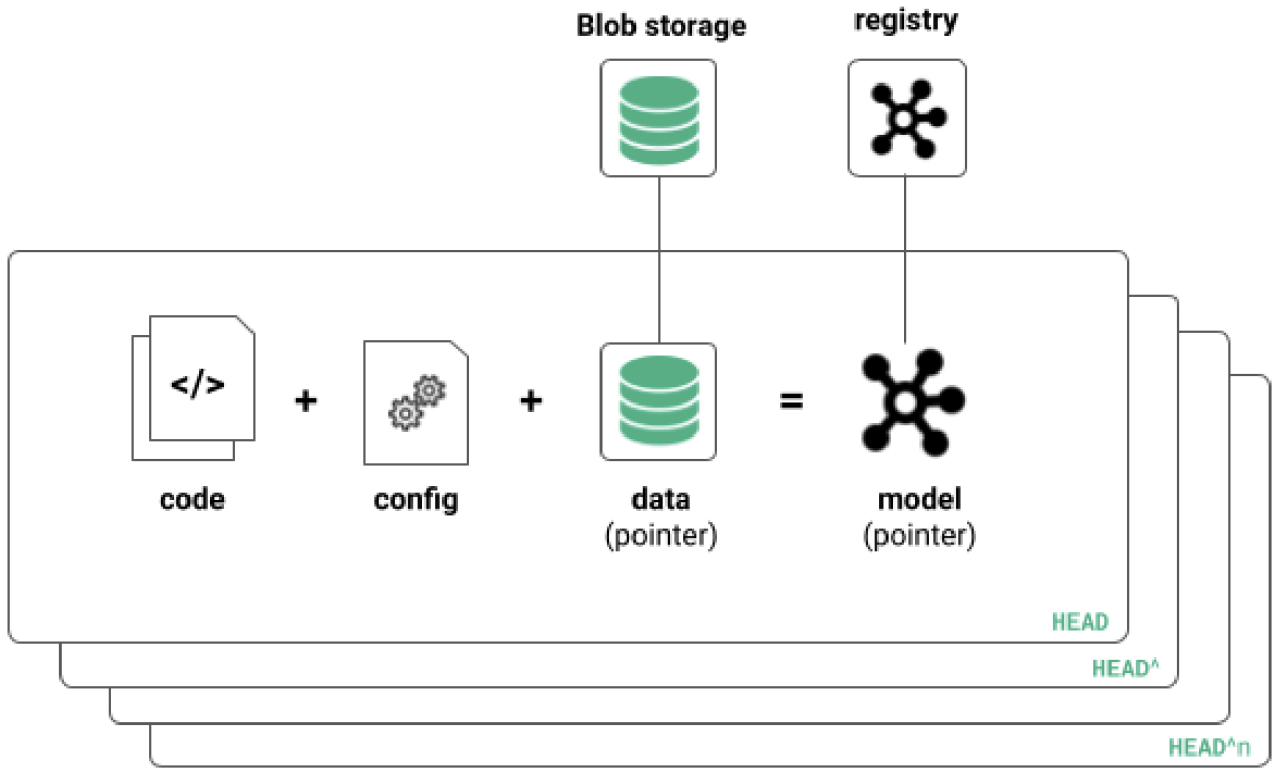
fg = fs.get_feature_group(name="example", version=1)
fg.append_features(features)
```

- Breaking schema changes require updating the schema version for a Feature Group.

```
fg1 = fs.create_feature_group(name="example", version=1)
df = fg1.read()
fg2 = fs.create_feature_group(name="example", version=2, features=new_features, …)
fg2.insert(df) #backfill the new feature group with data from the prev version
```

Data Versioning with Git

- Versioning code and data together using Git, such as Data Version Control (DVC).
- Warning: DVC is impractical and inefficient for large amounts of data, as it only stores pointers to blobs (binary large objects) in Git.



Data Versions with Apache Hudi

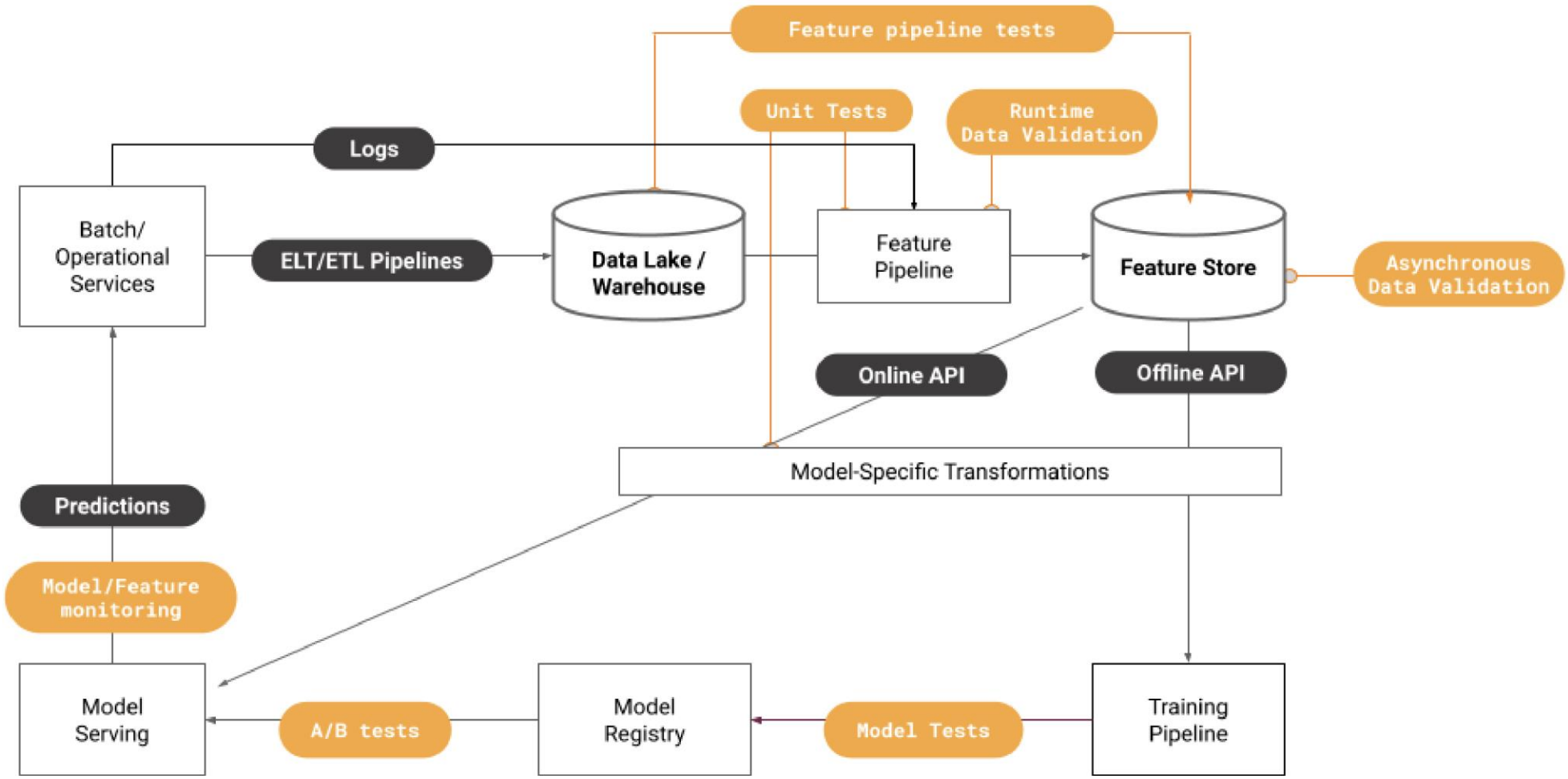


View Commits in the Hopsworks UI in the “Activity” tab of a Feature Group



2. Test Every Step

Tests in ML Systems



ML Test Criteria

Feature Tests

- Test that the distributions of each feature match your expectations
- Test the relationship between each feature and the target
- Test the cost of each feature (e.g., latency)
- Test all code that creates input features

Model Tests

- Test that every model specification undergoes a code review
- Test the relationship between offline proxy metrics and the actual impact metrics
- Test the impact of each tunable hyperparameter
- Test the effect of model staleness
- Test against a simpler model as a baseline
- Test model quality and model bias using important data slices (*evaluation sets*)

ML Test Criteria

Infrastructure Tests

- Integration test the feature, training, and inference pipelines.
- Test the reproducibility of model training.
- Test model quality before attempting to serve it.
- Test models via a canary process before production serving.
- Test that a model deployment can be safely rolled back to a previous version.

Monitoring Tests

- Test data quality for features in feature pipelines
- Test that data invariants hold in training and serving inputs
- Test that your training and serving features compute the same values
- Test for model staleness
- Test for NaNs or infinities appearing in your model during training or serving
- Test for regressions in prediction quality on served data

Refactor Code into Testable Functions

```
def compute_features(df : Dataframe): -> Dataframe
    if config["region"] == "UK":
        df["holidays"] = is_uk_holiday(df["year"], df[" week"])
    else:
        df["holidays"] = is_holiday(df["year"], df["week"])

    df["avg_3wk_spend"] = df["spend"].rolling(3).mean()
    df["acquisition_cost"] = df["spend"] / df["signups"]
    df["spend_shift_3weeks"] = df["spend"].shift(3)
    df["special_feature1"] = compute_bespoke_feature(df)
    df["spend_b"] = multiply_columns(df["acquisition_cost'], df['B'])
    return df

df = loader.load_actuals(dates) # e.g. spend, signups
df = compute_features(df)

feature_group = fs.get_feature_group("customer_features", version=1)
feature_group.insert(df)
```

This feature logic does not contain independently testable features

Refactor Code into Testable Functions

```python
def avg_3wk_spend(spend: pd.Series) -> pd.Series:
    """Rolling 3 week average spend."""
    return spend.rolling(3).mean()
……
```

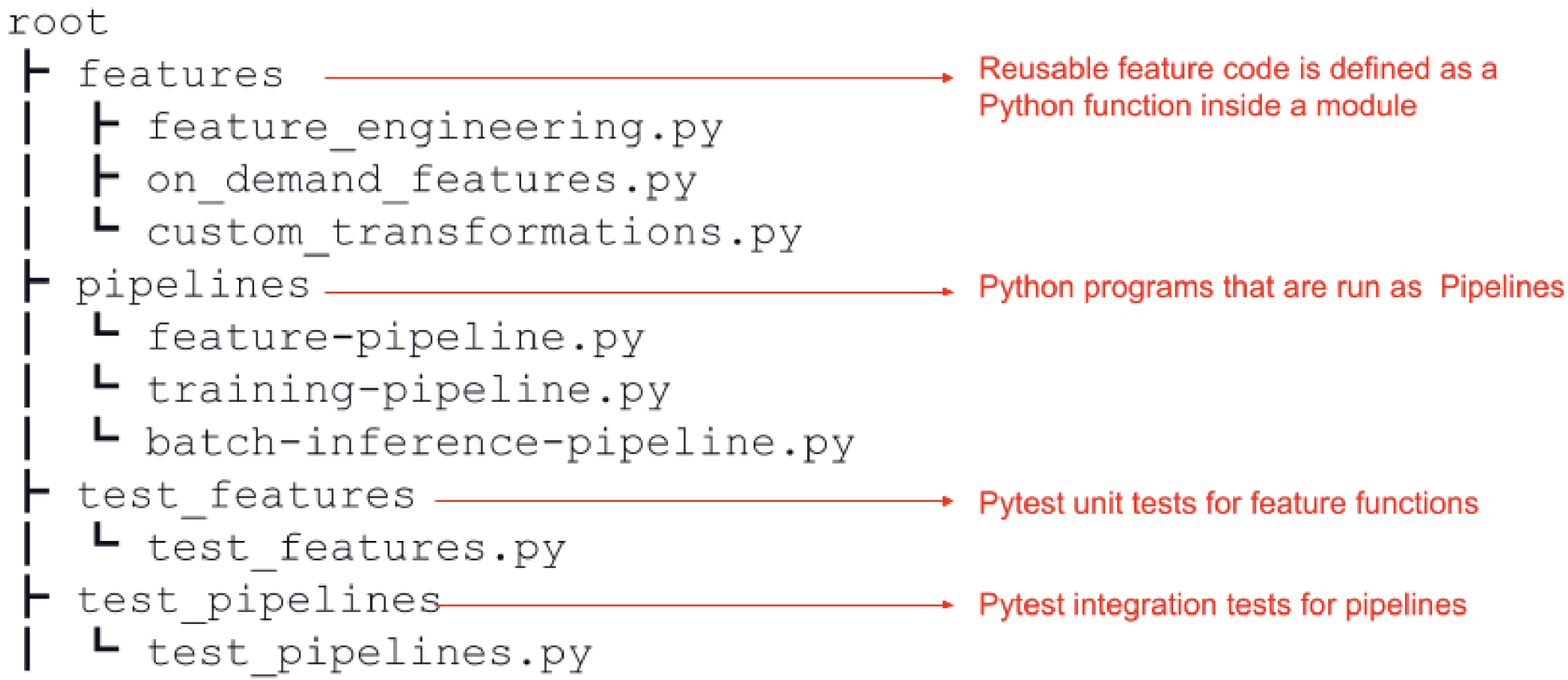
```python
def spend_per_signup(spend: pd.Series, signups: pd.Series) -> pd.Series:
    """The cost per signup in relation to spend."""
    return spend / signups
```

Unit Test

```python
@pytest.fixture
def get_spends(self) -> pd.DataFrame:
    return pd.DataFrame([[20, 40], [5, 4], [4, 10],
                         columns=["spends", "signups", "spend_per_signup"])

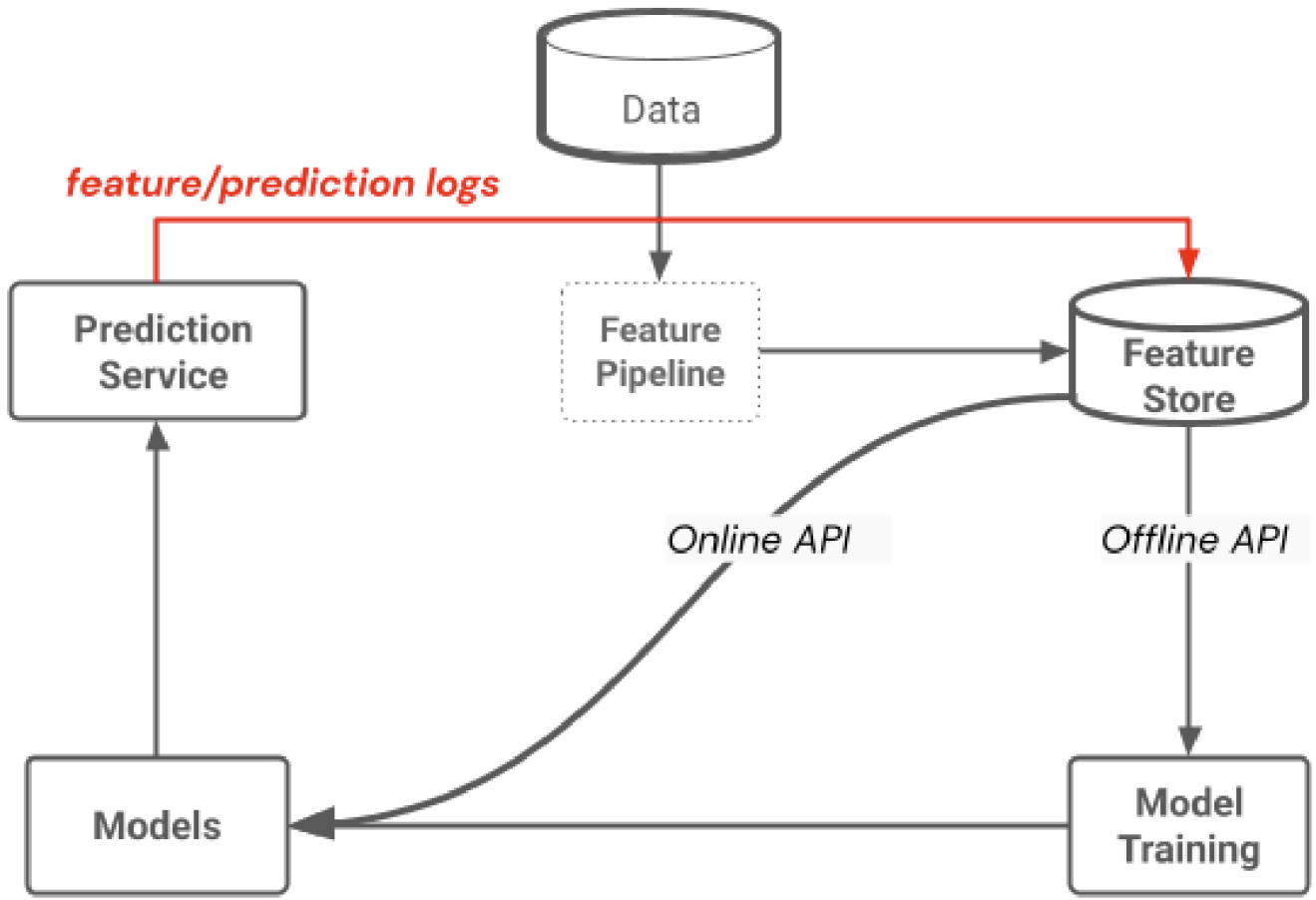
def test_spend_per_signup(get_spends : Callable):
    df = get_spends
    df["res"] = spend_per_signup(df["spends"), df("signups")]
    pd.testing.assert_series_equal(df["res"], df["spend_per_signup"])
```

Refactor Code into Testable Functions



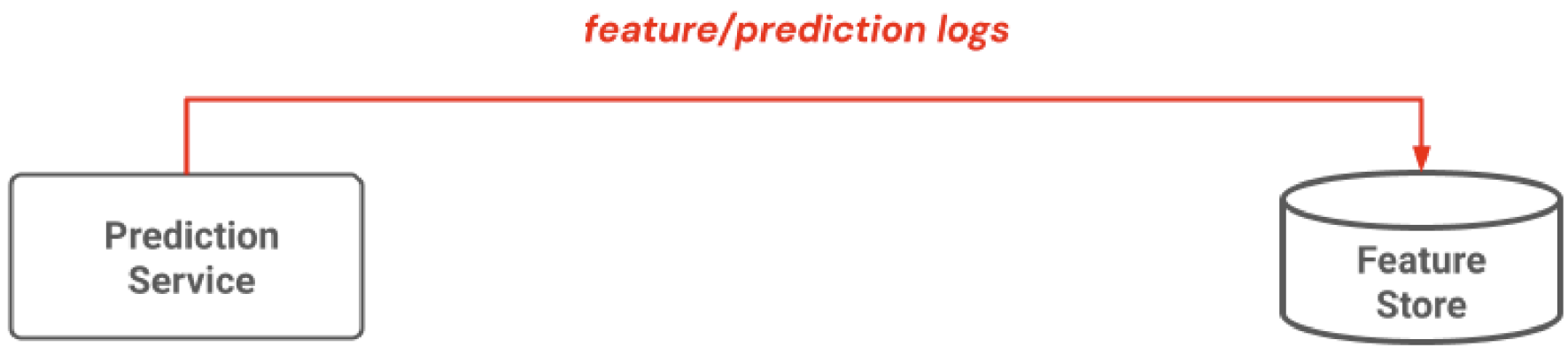
3. Log Everything

Feature & Prediction Logs

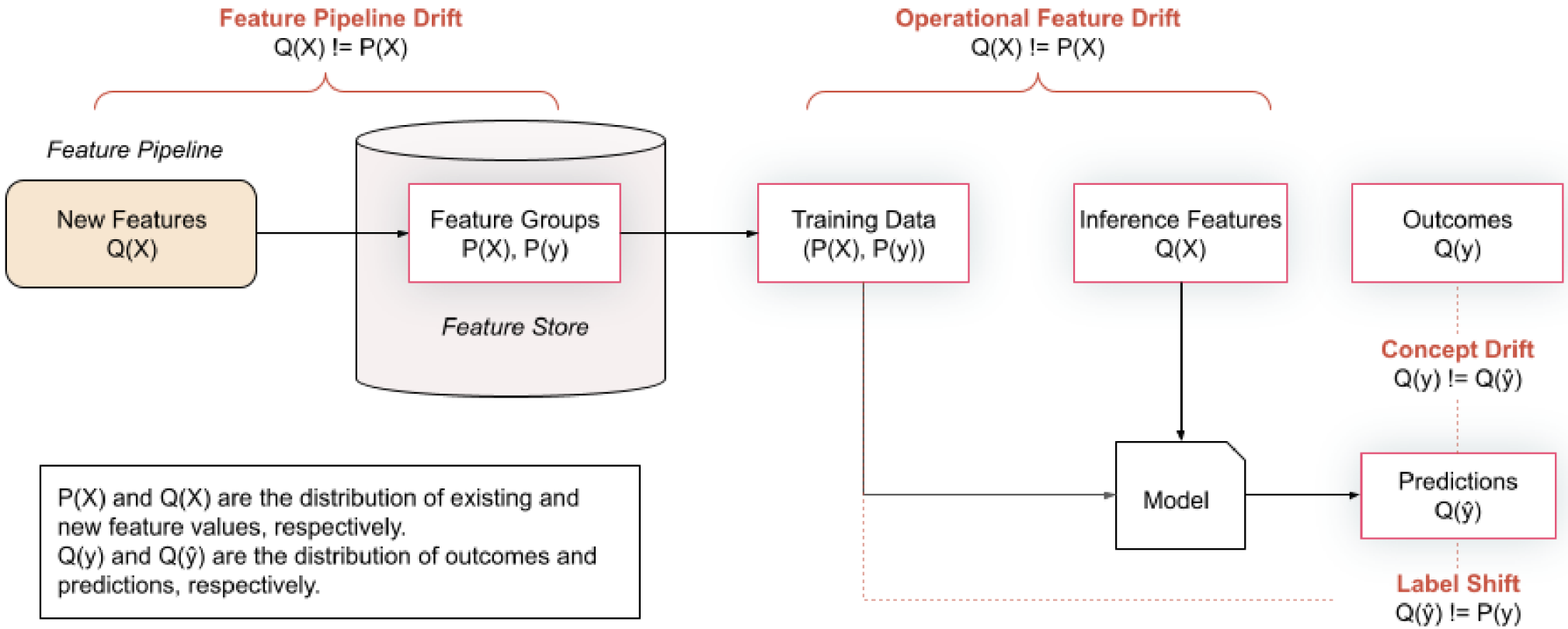


Feature & Prediction Logs

- Logs of predictions and features can be used for:
  - Debugging
  - Model/Feature/Performance Monitoring
  - To create new training data for models
- You should log untransformed feature values and predictions
  - Log before model-specific transformations & reverse_transform predictions



Use Case: Data Drift



Use Case: Data Drift

- **Feature Pipeline Drift**
  - The distribution of a feature in a DataFrame that is being written to a Feature Group is significantly different from the distribution of that feature stored in the Feature Group.

- **Operational Feature Drift** (aka Covariate Shift)
  - The distribution of a feature in a window of Inference Data is significantly different from the distribution of that feature in the models' training data.

- **Label Shift**
  - The distribution of a window of label data in inference is significantly different from the distribution of that label in the models' training data.

- **Concept Drift**
  - The outcomes are significantly different from the model's predictions.