Hopsworks Feature Store after 4 years: Lessons learned and what's next

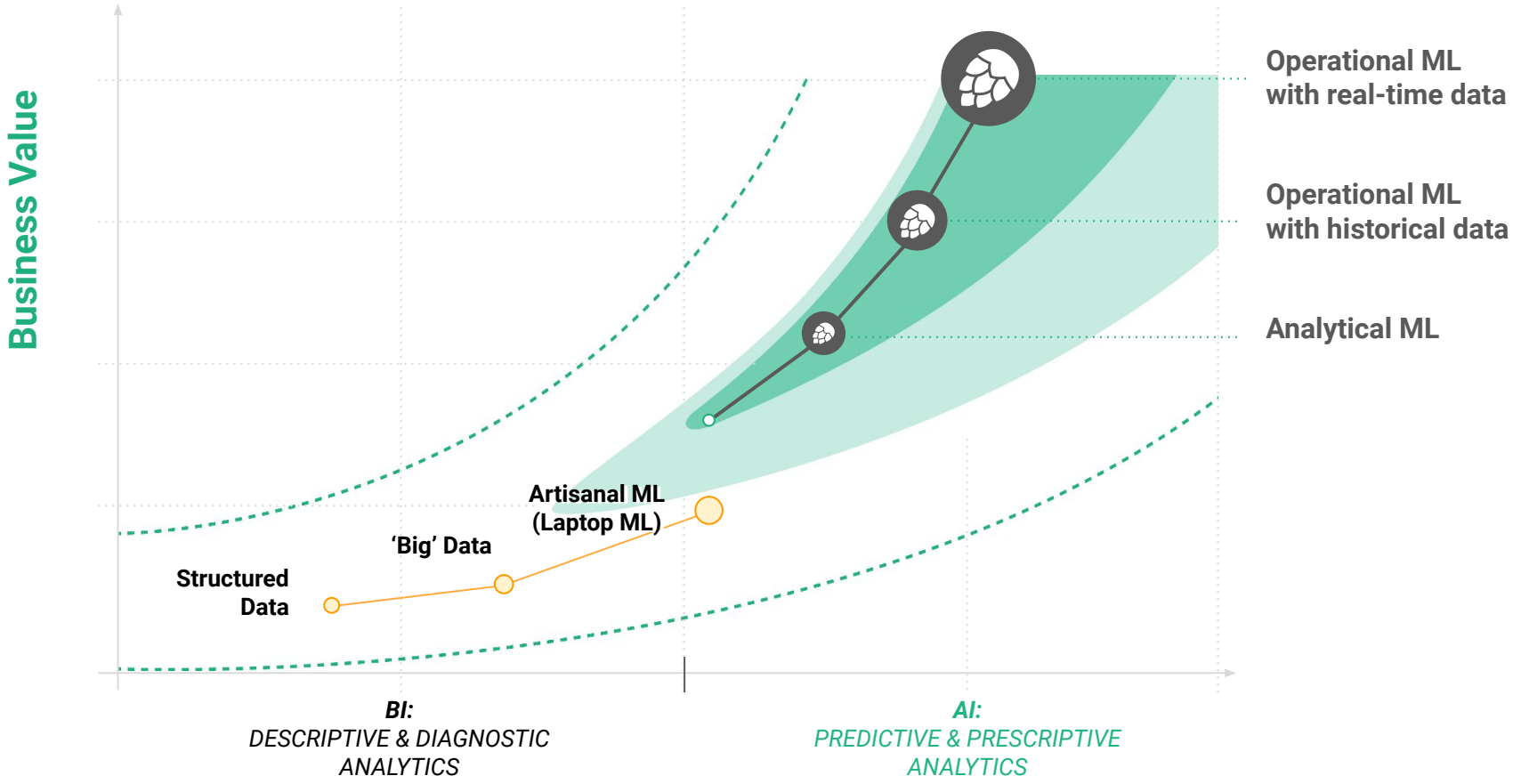
Fabio Buso, VP of Engineering, Hopsworks

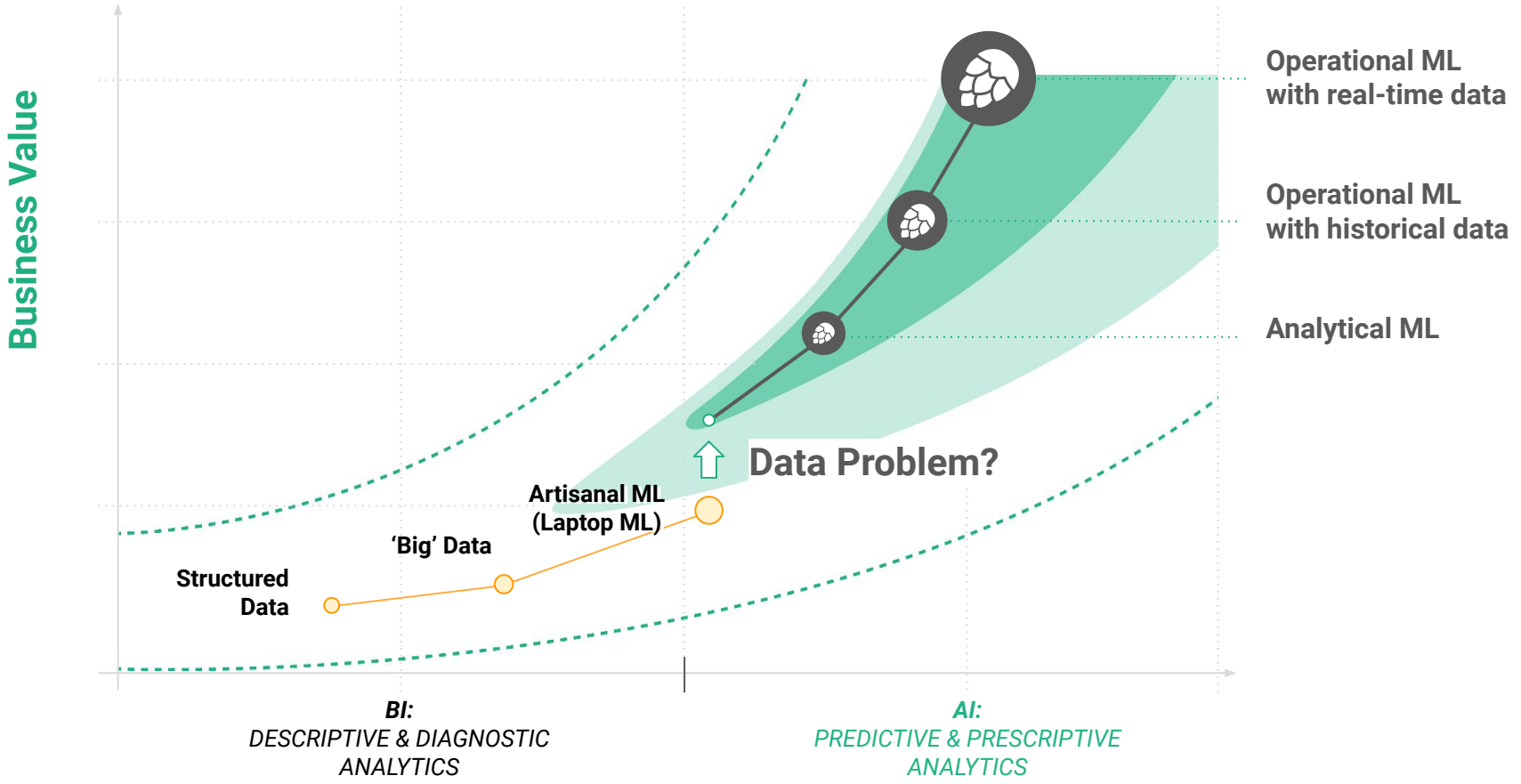
Moritz Meister, Head of Feature Store Engineering, Hopsworks

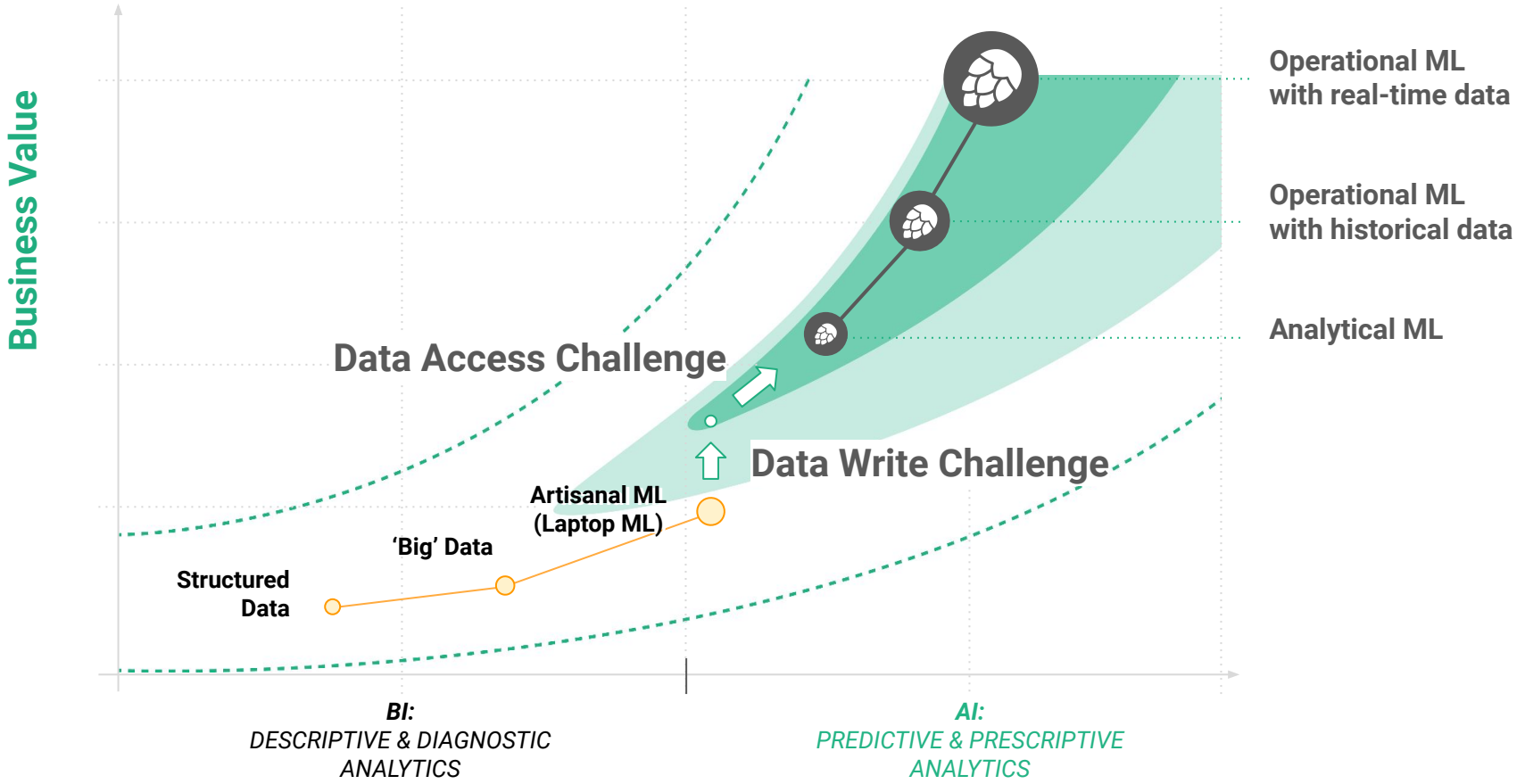
FEATURE STORE SUMMIT 2022

Organized by HOPSWORKS

Feature Stores and the Data Problem


Business Value
Operational ML with real-time data
Operational ML with historical data
Analytical ML
Artisanal ML (Laptop ML)
'Big' Data
Structured Data
BI:
DESCRIPTIVE & DIAGNOSTIC ANALYTICS
AI:
PREDICTIVE & PRESCRIPTIVE ANALYTICS

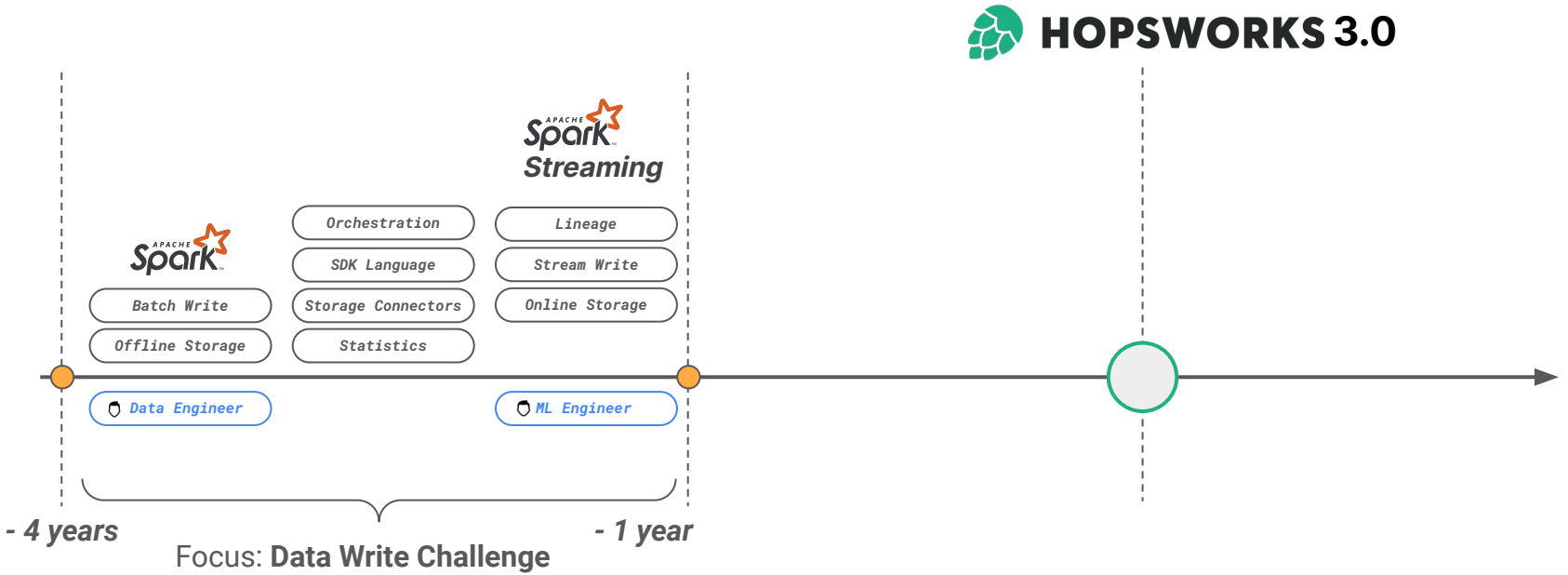

Business Value
Operational ML with real-time data
Operational ML with historical data
Analytical ML
Data Problem?
Artisanal ML (Laptop ML)
'Big' Data
Structured Data
BI:
DESCRIPTIVE & DIAGNOSTIC ANALYTICS
AI:
PREDICTIVE & PRESCRIPTIVE ANALYTICS


Business Value
Operational ML with real-time data
Operational ML with historical data
Analytical ML
Data Access Challenge
Data Write Challenge
Artisanal ML (Laptop ML)
'Big' Data
Structured Data
BI:
DESCRIPTIVE & DIAGNOSTIC ANALYTICS
AI:
PREDICTIVE & PRESCRIPTIVE ANALYTICS


SPARK
FEATURE STORES
imgflip.com

A (short) history of Feature Store capabilities

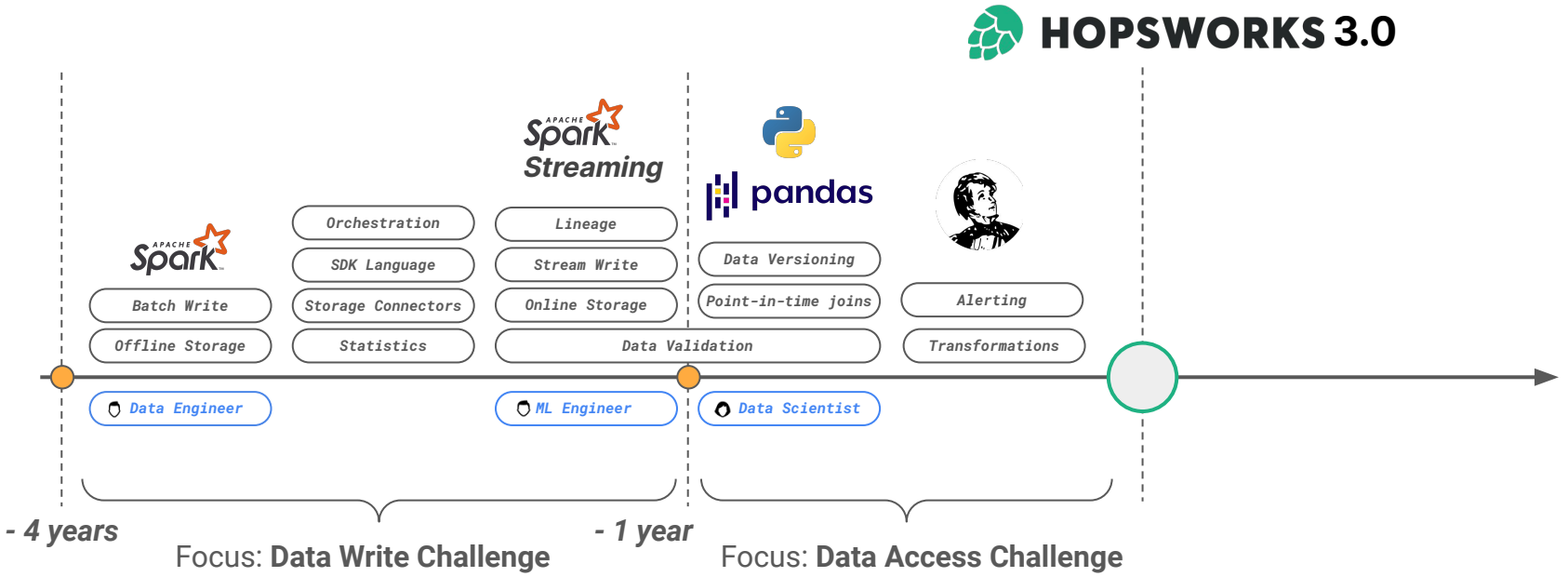
From batch to prediction service



Focus: **Data Write Challenge**

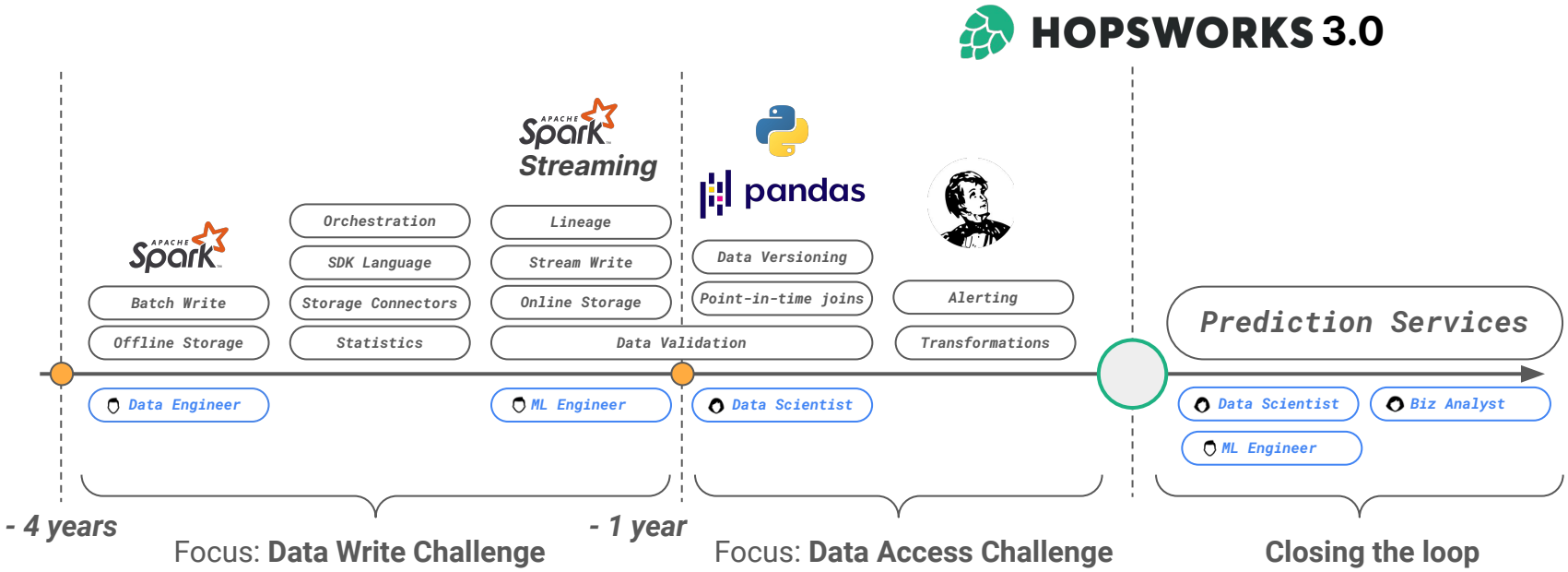
A (short) history of Feature Store capabilities

From batch to prediction service



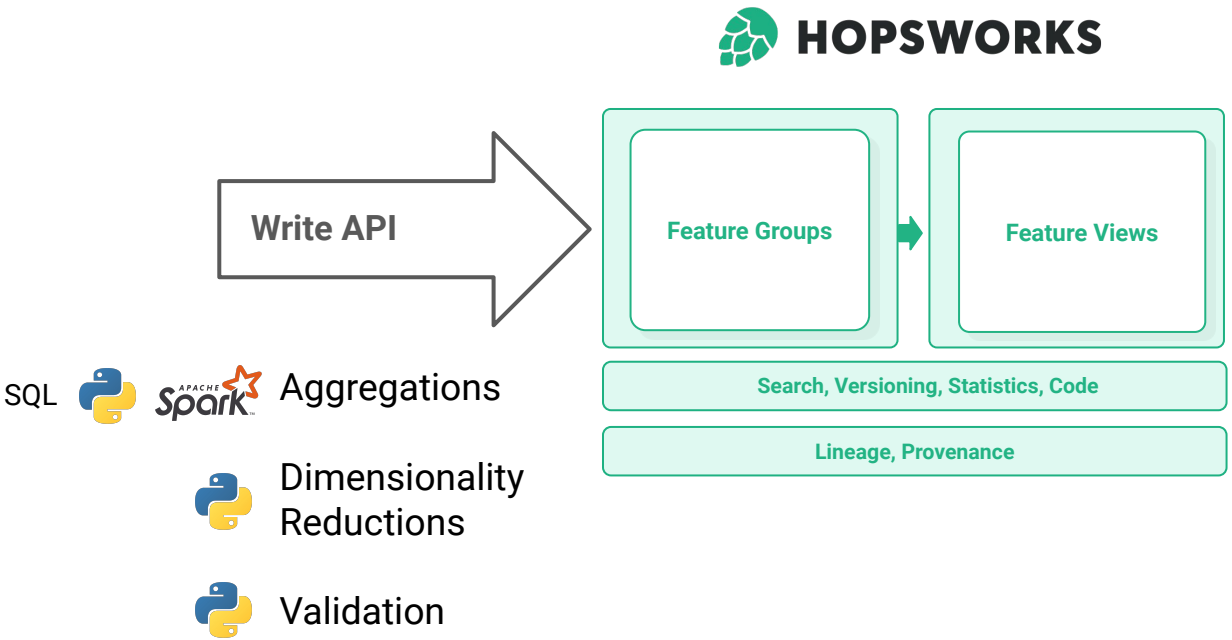
A (short) history of Feature Store capabilities

From batch to prediction service



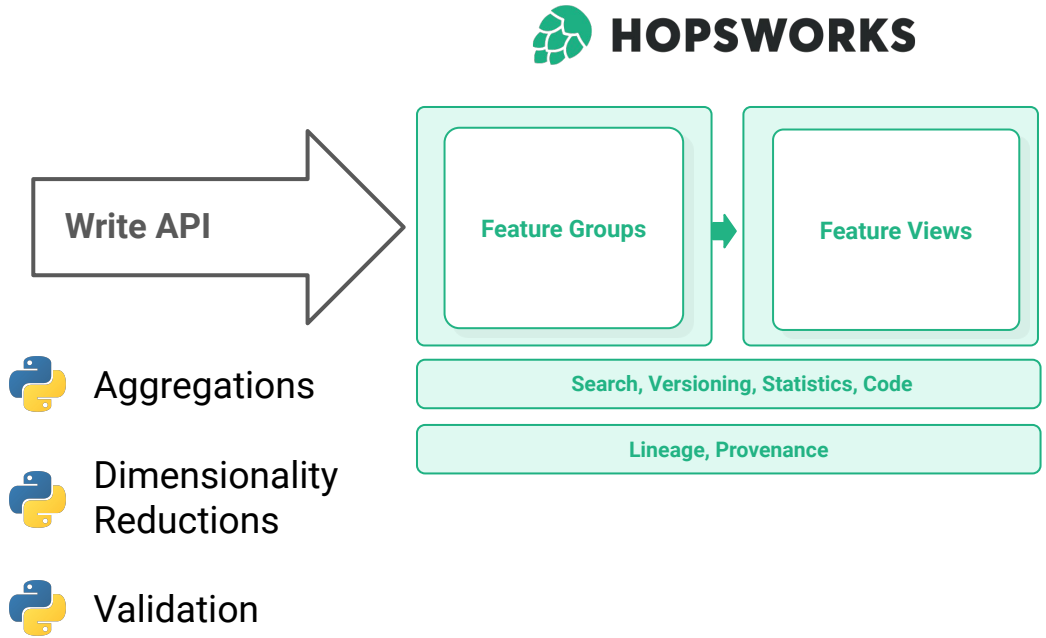
Write to Feature Groups, read from Feature Views

Effort 1. Making Feature Pipelines accessible to Data Scientists



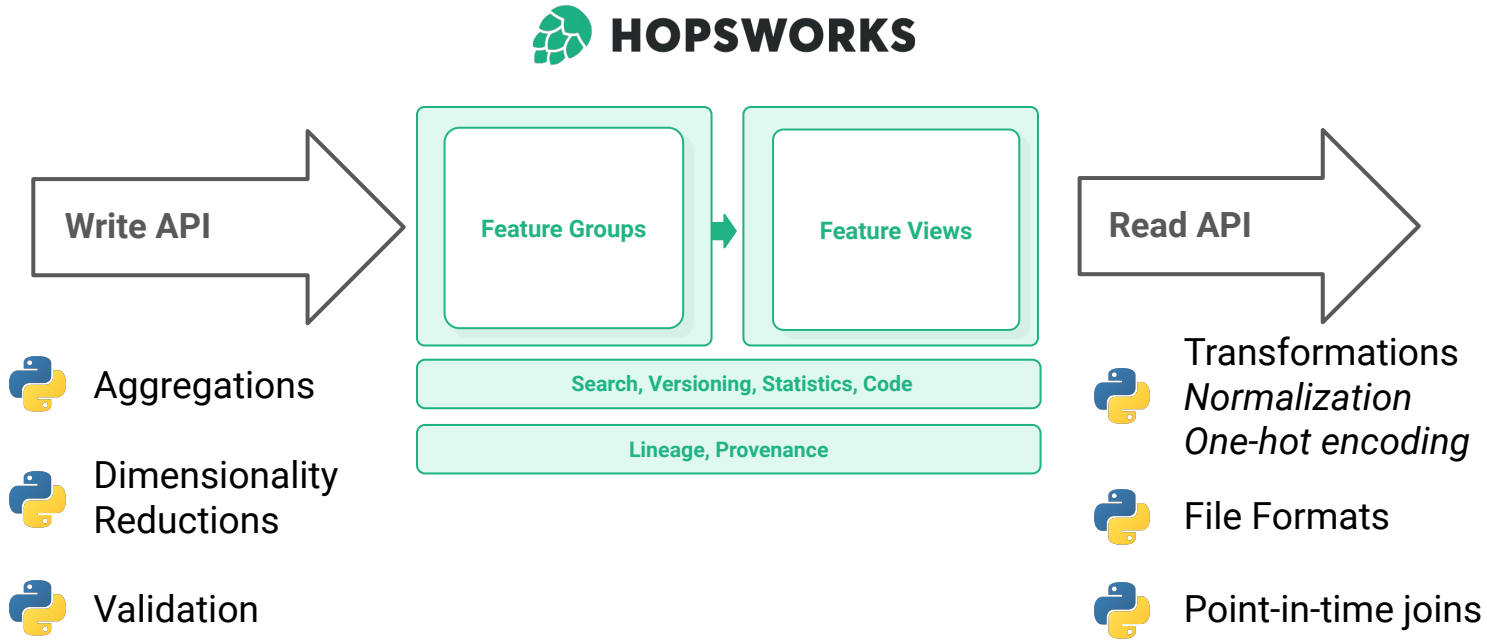
Write to Feature Groups, read from Feature Views

Effort 1. Making Feature Pipelines accessible to Data Scientists



Write to Feature Groups, read from Feature Views

Effort 2. Making Data accessible to Data Scientists



Feature Groups and the Dataframe API

Data Write Challenge

Why DSLs don’t make the cut

Advantages of Dataframes

1. **Flexibility** - *DSLs are use case specific*
2. **User Experience** - *No additional learning curve and no lock in*
3. **Bring your own pipeline** - *Keep existing libraries and pipelines*

Why Spark is not suitable for Data Scientists

1. **Resource Estimation**
   - *Distributed environment resources don't map 1:1 to local environments*
2. **Debugging**
   - *Debugging in distributed systems quickly becomes a complex task*
3. **Models are rarely trained on Spark Dataframes**
   - *Modeling Frameworks are often not distributed tools*
4. **Wrapping SQL in Python functions is not pythonic**
   - *At serving time there is often no Spark context available*

Feature engineering

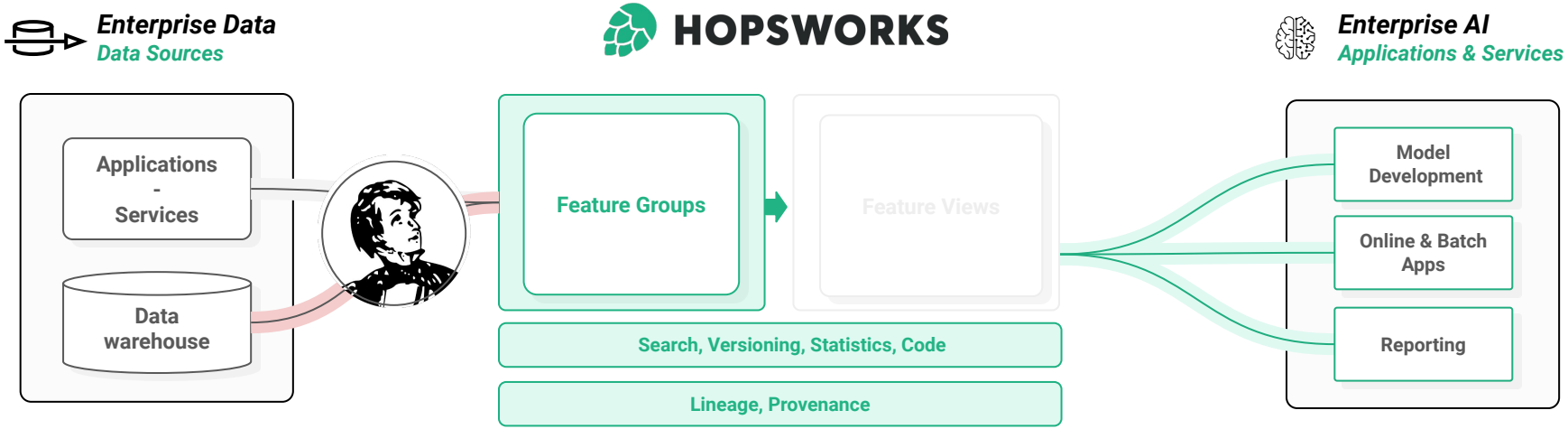
Initialize Feature Group metadata

Write feature data

```python
window_len = "4h"
cc_group = trans_df[["cc_num", "amount", "datetime"]] \
    .groupby("cc_num").rolling(window_len, on="datetime")

# Compute aggregations
df_4h_count = pd.DataFrame(cc_group.mean())
df_4h_count.columns = ["trans_freq", "datetime"]
df_4h_count = df_4h_count.reset_index(level=["cc_num"])
df_4h_count = df_4h_count.drop(columns=["cc_num", "datetime"])
df_4h_count = df_4h_count.sort_index()
window_aggs_df = window_aggs_df.merge(df_4h_count,left_index=True, right_index=True)

# Initialize metadata of feature group
window_aggs_fg = fs.get_or_create_feature_group(
    name="transactions_4h_aggs_fraud_batch_fg",
    version=1,
    description="Aggregate transaction data over 4hwindows.",
    primary_key=["cc_num"],
    event_time="datetime"
)

# Write Dataframe to the feature group
window_aggs_fg.insert(window_aggs_df)
```

Where does Great Expectations fit in?

Validate data before it is made available to data scientists



Where does Great Expectations fit

Validate data before it is made available to data scientists

Pandas Dataframe

Expectation

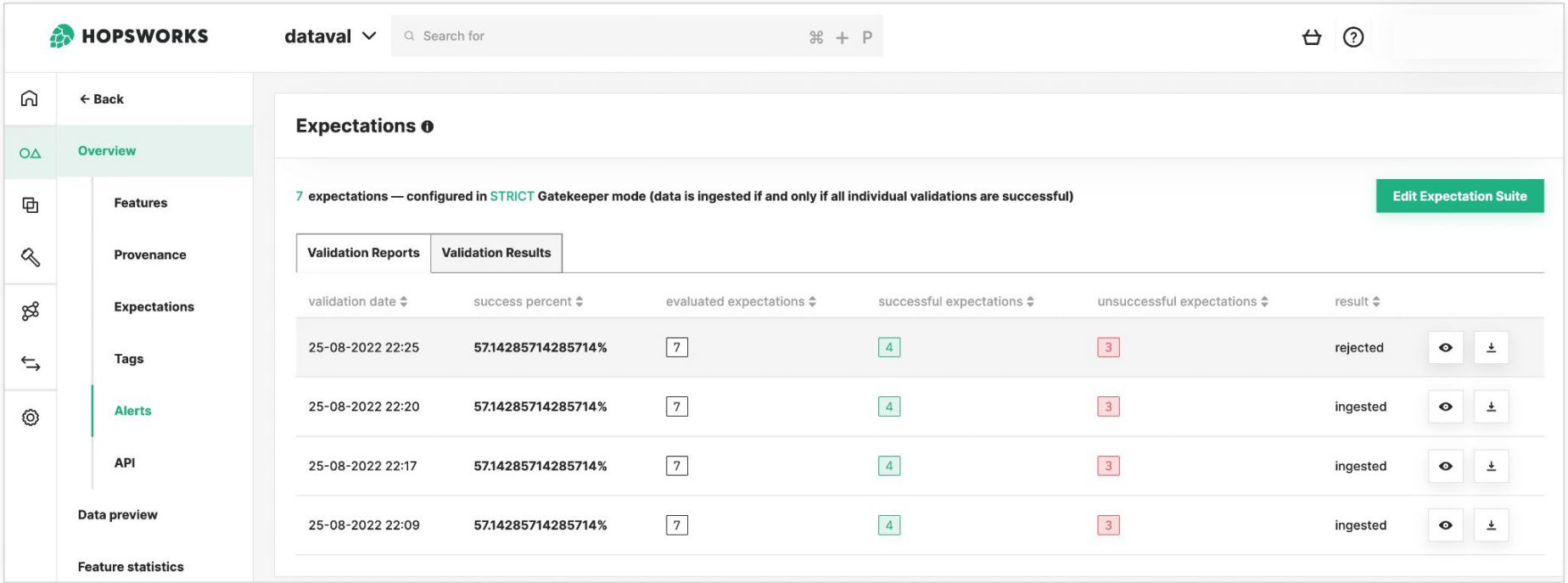
```
ExpectationConfiguration(
    expectation_type="expect_column_min_to_be_between",
    kwargs={
        "column":"age_at_transaction",
        "min_value": 18,
        "max_value": 130
    }
)
```

Result

```
validation_result = {
  "success": false,
  "result": {
    "observed_value": 17.738671,
    "element_count": 106020,
    "missing_count": null,
    "missing_percent": null
  },
  ...
}
```

Where does Great Expectations fit

Validate data before it is made available to data scientists



Python, Pandas and Feature Views

Data Access Problem

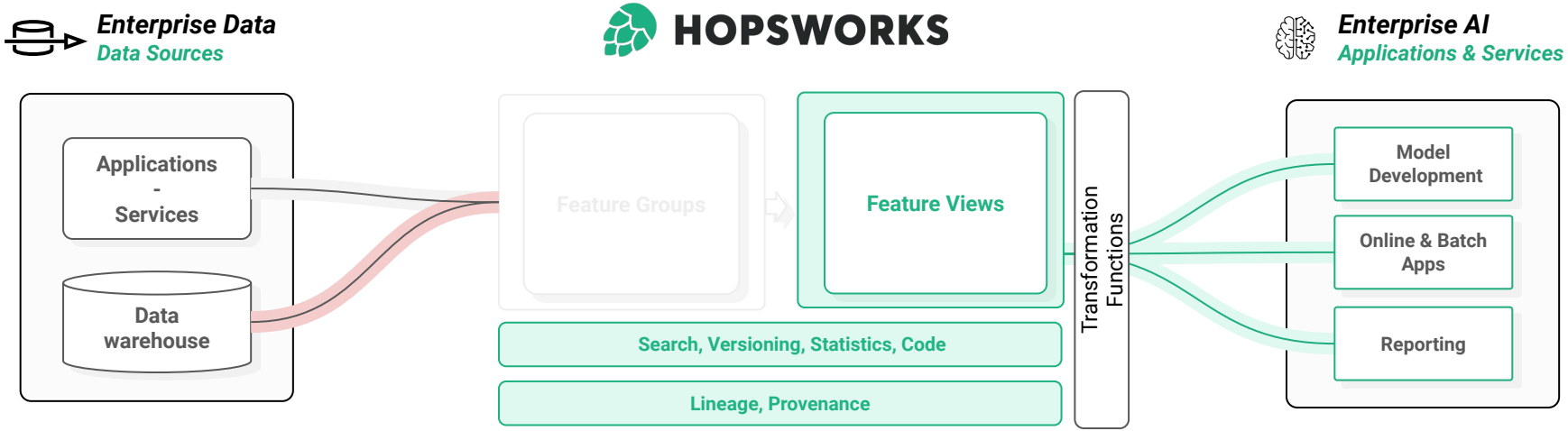
Feature Reusability requires Views - Feature Views

Why Data Scientists want Python-centric APIs

1. **Models are trained with Python libraries** - *DSLs are use case specific*
2. **Training frameworks might require file formats** - Scikit-learn & Pandas, Tensorflow & Tfrecord, Pytorch & Numpy
3. **Point-in-time joins are hard** - *Writing PIT joins in SQL, Spark or Python is error prone*
4. **Transformations happen at feature retrieval time** - *Reusing a feature often means limiting the set of entities a model is trained on, which again leads to different statistics*
5. **Spark not available at feature retrieval time** - *At serving time there is often no Spark context available*

Feature Views for training data, batch scoring and low latency serving

The API for Data Scientists



Feature Reusability with Feature Views

Pandas like join making PIT joins transparent

Get Feature Group metadata

Select features

Create Feature View along with transformations

```python
# Select feature groups
trans_fg = fs.get_feature_group("transactions_fraud_batch_fg",
    version=1)
window_aggs_fg = fs.get_feature_group("transactions_4h_aggs_fraud_batch_fg",
    version=1)

# Join the feature gorups and select the features
ds_query = trans_fg.select(["fraud_label", "category", "amount",
        "age_at_transaction", "days_until_card_expires", "loc_delta"]) \
    .join(window_aggs_fg.select_except(["cc_num"]))

feature_view = fs.create_feature_view(
    name="transactions_view",
    query=ds_query,
    labels=["fraud_label"],
    transformation_functions={
        "category": label_encoder,
        "amount": min_max_scaler,
        "days_until_card_expires": min_max_scaler,
    }
)
```

Reading from the Feature View

One API to rule them all

Training Data as files

Training Data in-memory as Pandas Dataframe

New batches of data for scoring

Real-time feature lookup

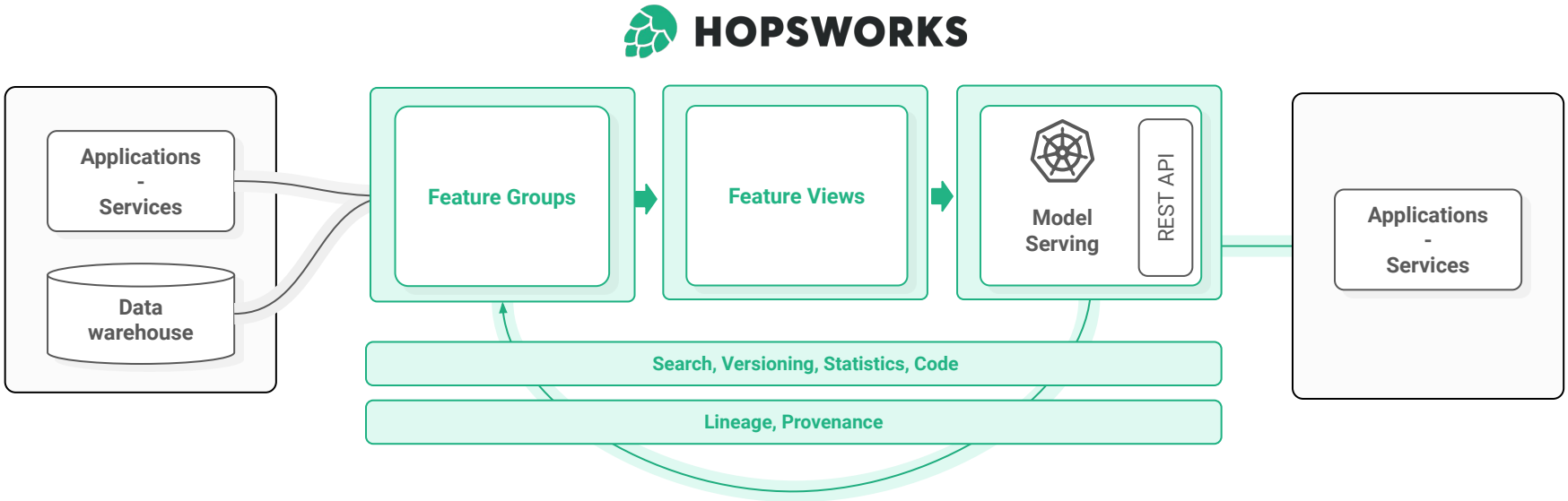
```python
# Create training data in a specific file format
td_version, td_job = feature_view.create_train_validation_test_split(
    description = 'transactions fraud batch training dataset',
    data_format = 'csv',
    validation_size = 0.2,
    test_size = 0.1
)

# Retrieve previoulsy materialised training data
X_train, y_train, X_val, y_val, X_test, y_test = feature_view.get_train_validation_test_split(1)

# Generate batches of data from a certain time range
start_time = int(float(datetime.strptime("2022-01-03 00:00:01",
    date_format).timestamp()) * 1000)
end_time = int(float(datetime.strptime("2022-03-31 23:59:59",
    date_format).timestamp()) * 1000)
march_transactions = feature_view.get_batch_data(
    start_time = start_time,  end_time = end_time)

# Initialize with training data version for transformations and get single feature vectors
feature_view.init_serving(1)
feature_array = feature_view.get_feature_vector(
    {"primary_key": 1})
```

Endboss: Prediction Service

Integrating Data APIs with Model APIs

Building Prediction Services with KServe Integration

Closing the loop



Tying it all together

The predictor interface

Establish connection to Feature Store

Initialize Feature View

Loading the model from a model registry

Implementing the predict interface using the Feature View

```python
import os
import numpy as np
import hsfs
import joblib

class Predict(object):

    def __init__(self):
        """ Initializes the serving state, reads a trained model"""
        # get feature store handle
        fs_conn = hsfs.connection()
        self.fs = fs_conn.get_feature_store()

        # get feature views
        self.fv = self.fs.get_feature_view("transactions_view", 1)

        # initialise serving
        self.fv.init_serving(1)

        # load the trained model
        self.model = joblib.load(
            os.environ["ARTIFACT_FILES_PATH"] + "/fraud.model.pkl")
        print("Initialization Complete")

    def predict(self, inputs):
        """ Serves a prediction request usign a trained model"""
        # Numpy Arrays are not JSON serializable
        return self.model.predict(
            np.asarray(
                self.fv.get_feature_vector({"cc_num": inputs[0]}))
                    .reshape(1, -1)
            ).tolist()
```

app.hopsworks.ai

Try it out! Serverless and free!