Machine Learning in Production

Agenda

00. Course Introduction & workspace set up

01. Experimentation

02. Model Management

03. Deployment Paradigms

04. Production

Introductions

- Name/Role
- Experience with:
  - MLflow?
  - Deploying/productionizing models?
- Expectations?

Databricks Certified ML Professional

Certification helps you gain industry recognition, competitive differentiation, greater productivity, and results.

- This course helps you prepare for the **Databricks Certified Machine Learning Professional exam**
- Please see the Databricks Academy for additional prep materials

For more information visit:
databricks.com/learn/certification

00 Course Introduction

Hardest Part of ML isn't ML, it's Data

"Hidden Technical Debt in Machine Learning Systems," Google NIPS 2015

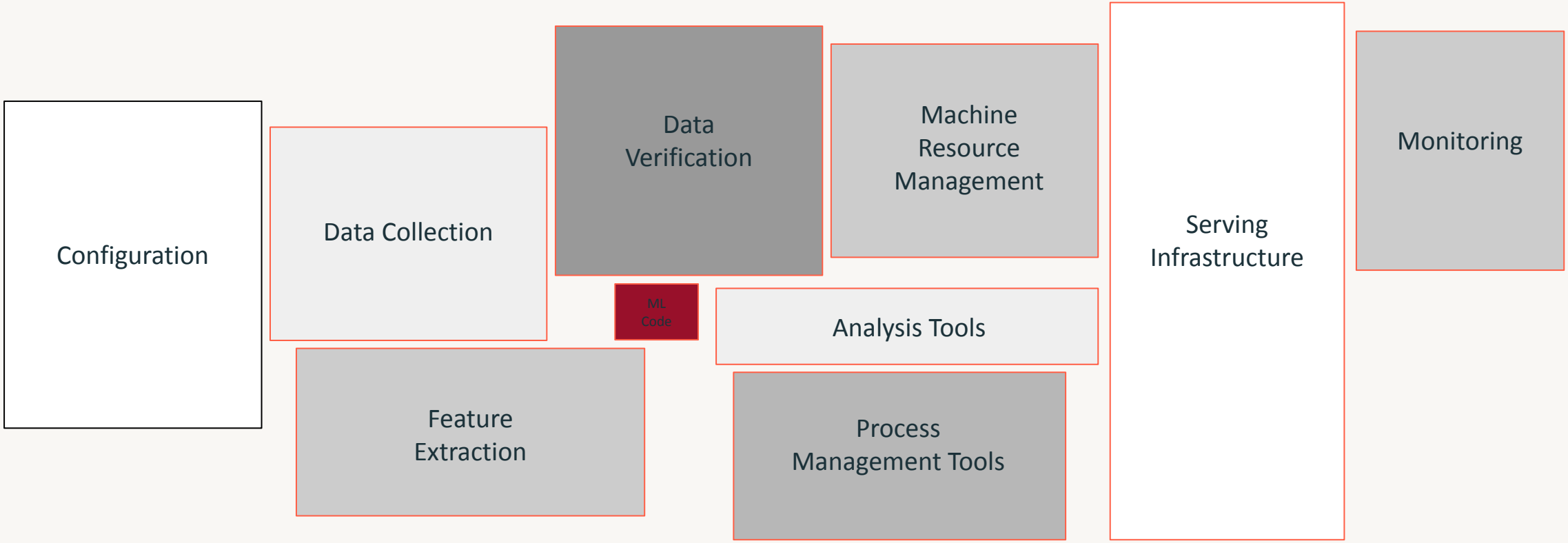


Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small green box in the middle. The required surrounding infrastructure is vast and complex.

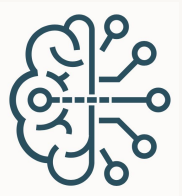
MLOps Introduction

- What is MLOps?
- Why should we care about MLOps?
- People and process
- Guiding principles

What is MLOps?

MLOps is a set of **processes and automation** for **managing code, data, and models** to **improve performance, stability and long-term efficiency** of ML systems

MLOps = DevOps + DataOps + ModelOps



Why should we care about MLOps?

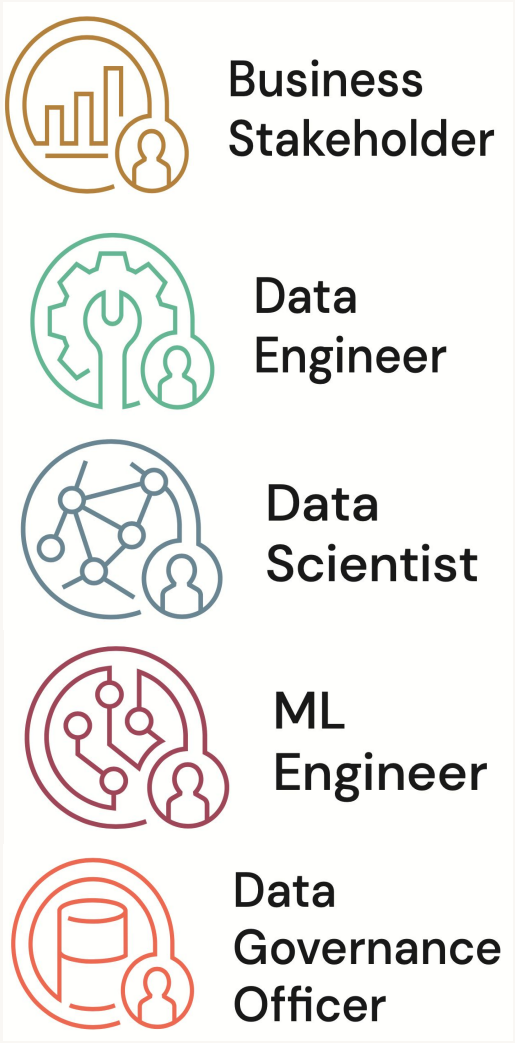
MLOps helps you reduce risk

- Technical risk - poorly performing models, fragile infrastructure
- Compliance risk - violating regulatory or corporate policies

MLOps improves long-term efficiency through automation

- Catch errors before they hit production
- Avoid slow, manual processes

People and process



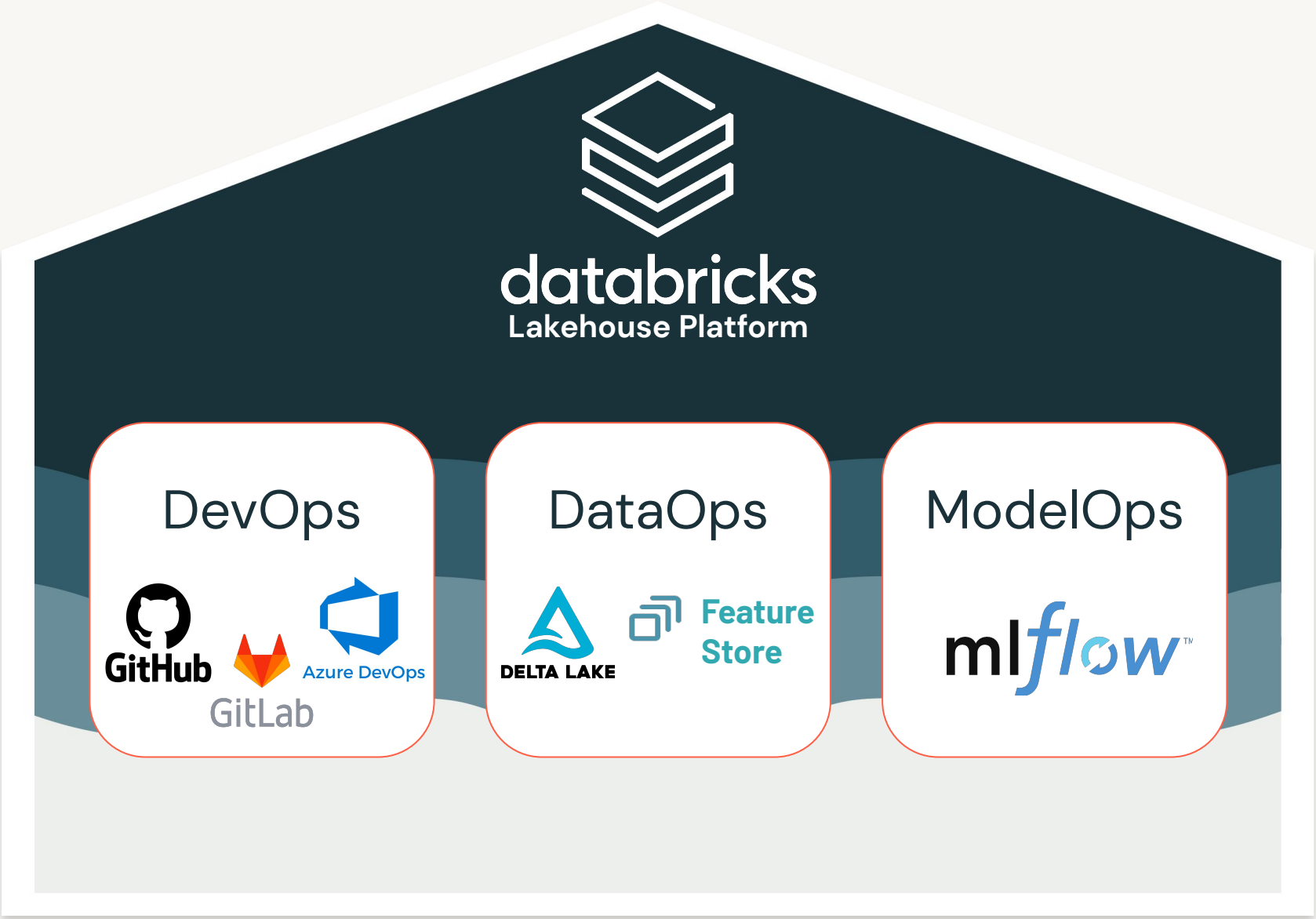
→ Responsible for business value of the ML solution

→ Builds data pipelines

→ Translates business problem; trains, tunes model

→ Deploys ML model to production

→ Responsible for data governance and compliance



Data-centric ML platform

Guiding principles

Always keep your business goals in mind

Take a data-centric approach to machine learning

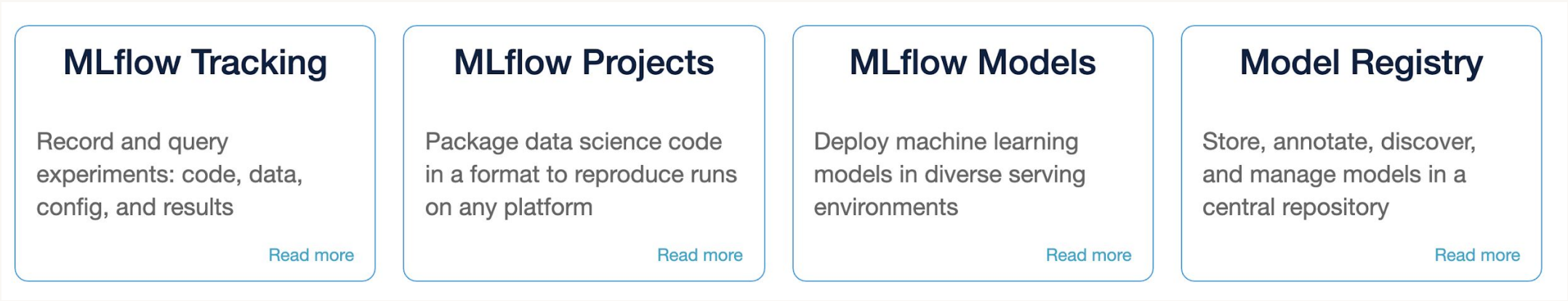
Implement MLOps in a modular fashion

Process should guide automation

Data & Model Management and Reproducibility

Managing the machine learning lifecycle means:

- Reproducibility of data -- data management
- Reproducibility of code -- version control, source tracking
- Reproducibility of models -- model management
- Automated integration with production systems -- CI/CD, testing

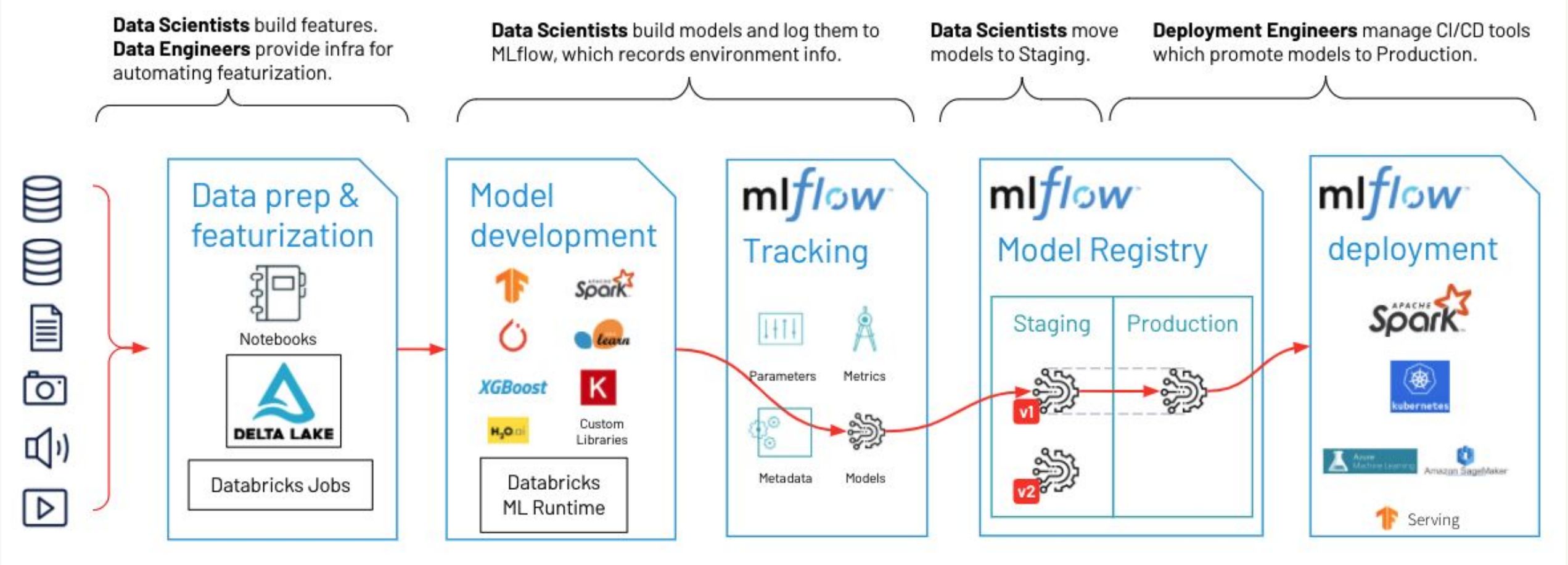


- Open-source platform for machine learning lifecycle
- Operationalizing machine learning
- Developed by Databricks
- Pre-installed on the Databricks Runtime for ML
- APIs: CLI, Python, R, Java, REST

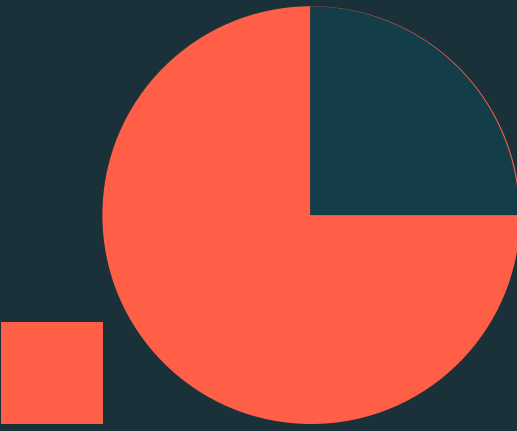
Data is never static

Model Centric → Data Centric Machine Learning

The Full ML Lifecycle



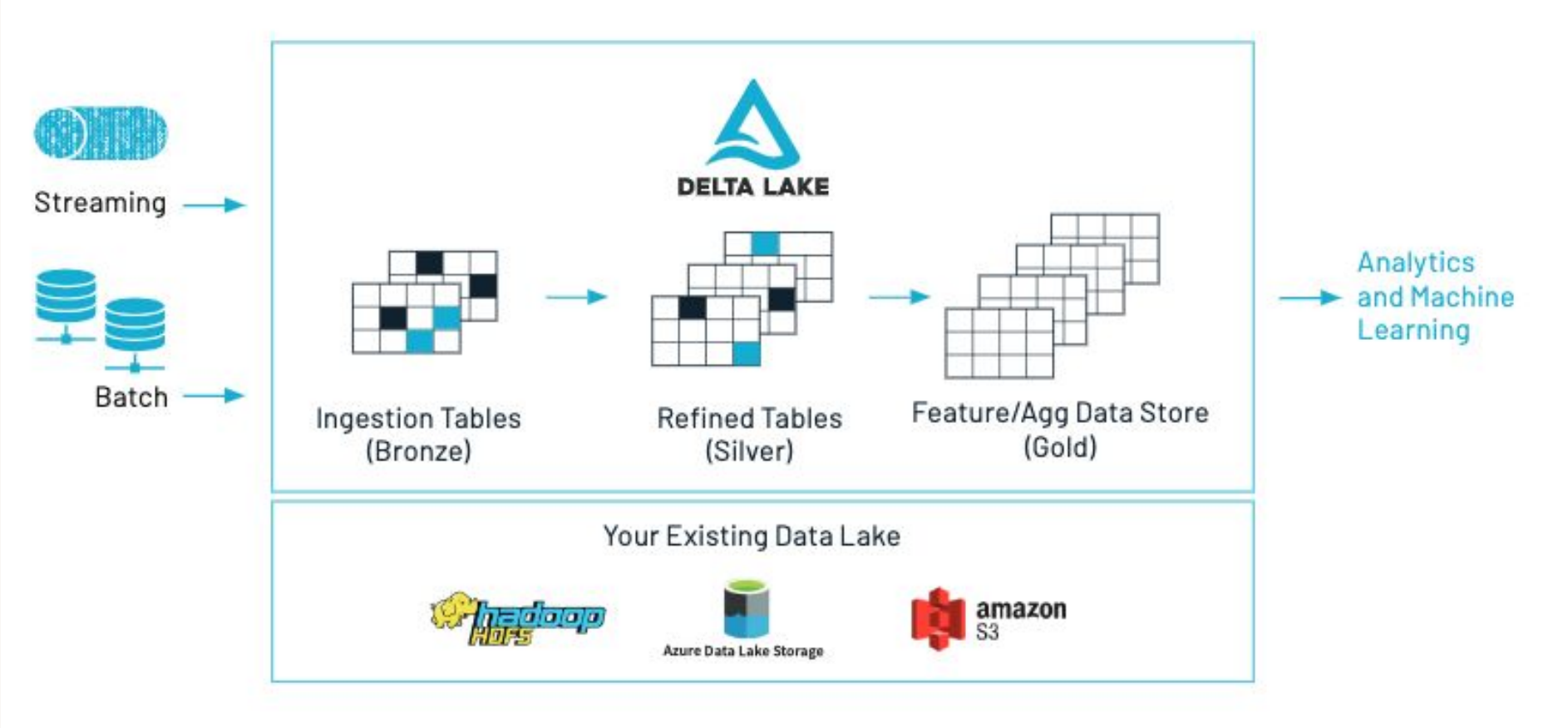
Workspace Setup



01 Experimentation

Delta Lake

Open-source Storage Layer



Delta Lake's Key Features

- ACID transactions → reliability
- Time travel (data versioning) → reliability, error recovery, synchronization
- Schema enforcement and evolution → reliability
- Audit history → data governance and compliance
- Parquet format + transaction log
- Compatible with Apache Spark API

Delta Lake Optimize

- Compaction
- Data Skipping
- Z-ordering
- FAQs on “OPTIMIZE”

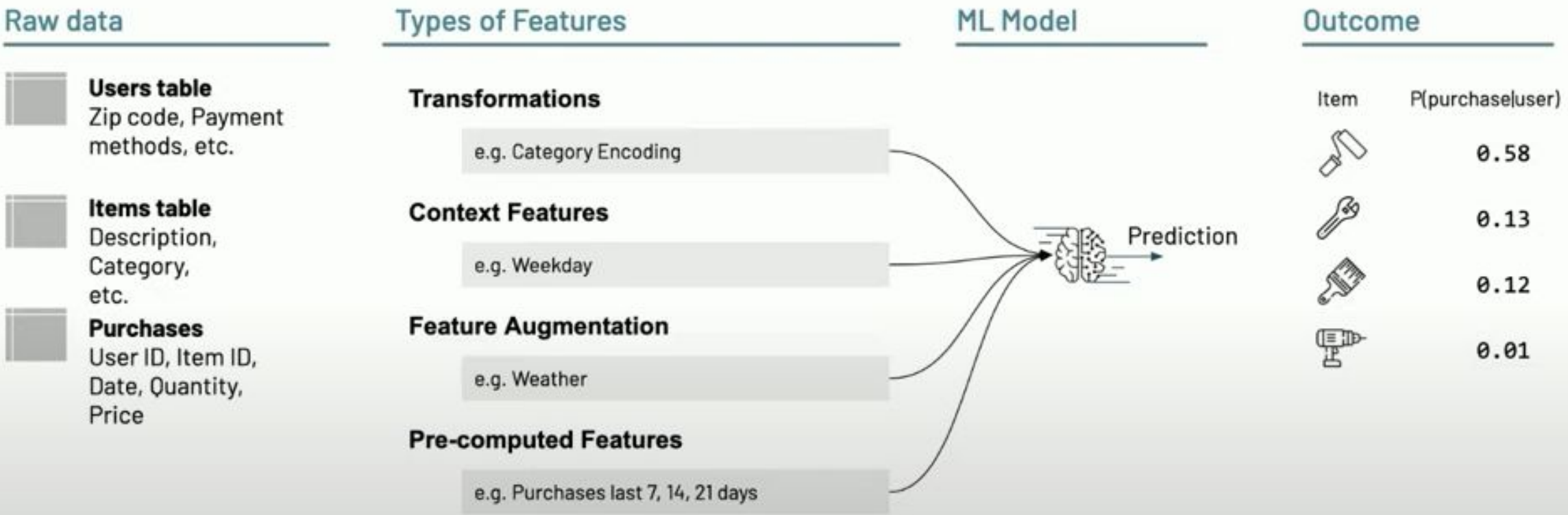
Feature Store

Production Machine Learning

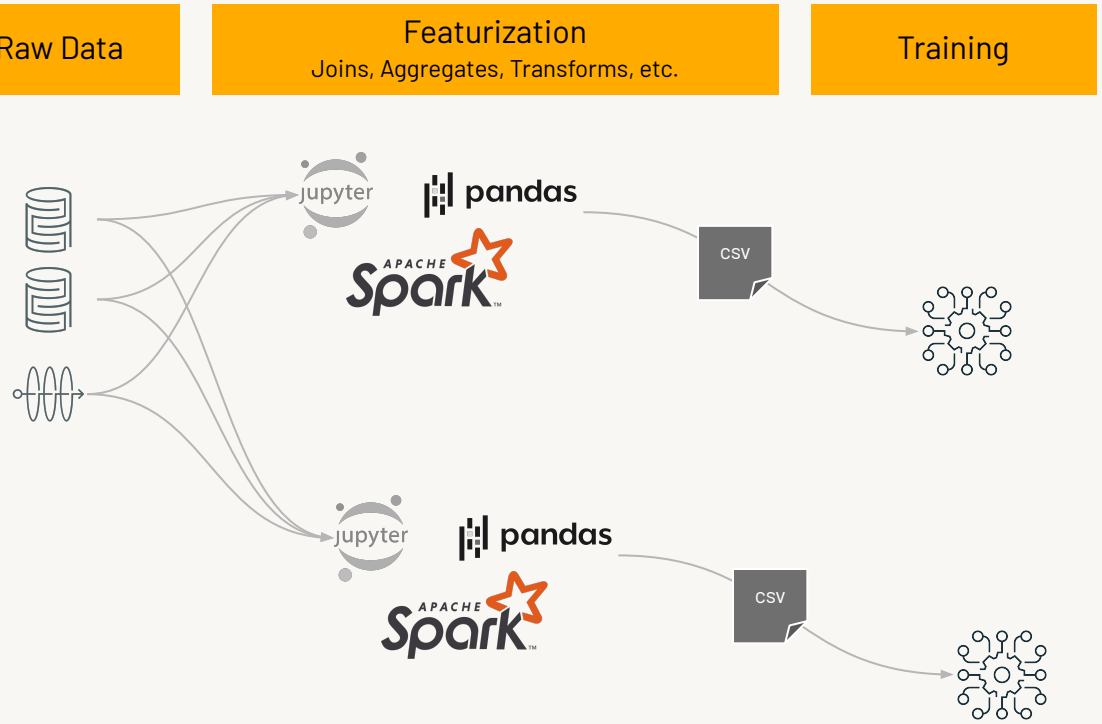
= Production Software + Data

First things first: What is a feature?

On the example of a recommendation system

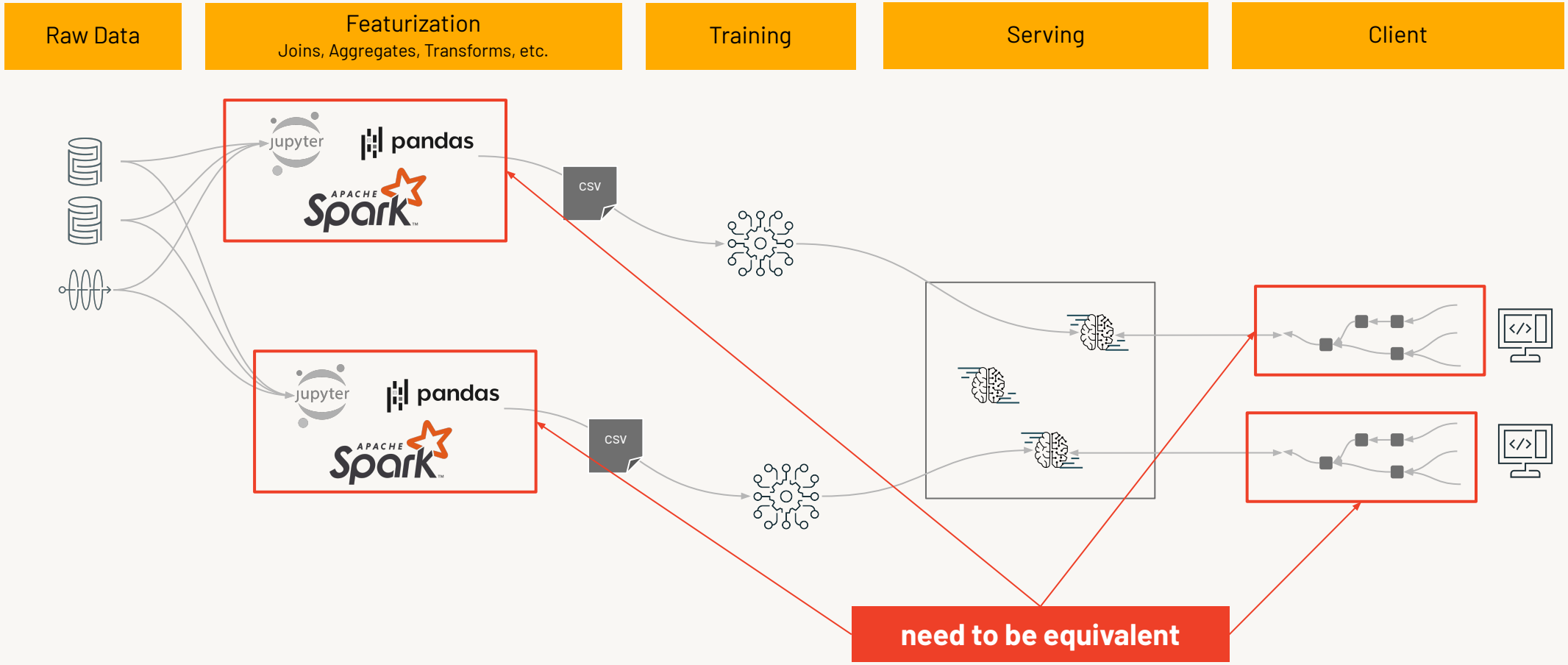


Data Challenges in ML



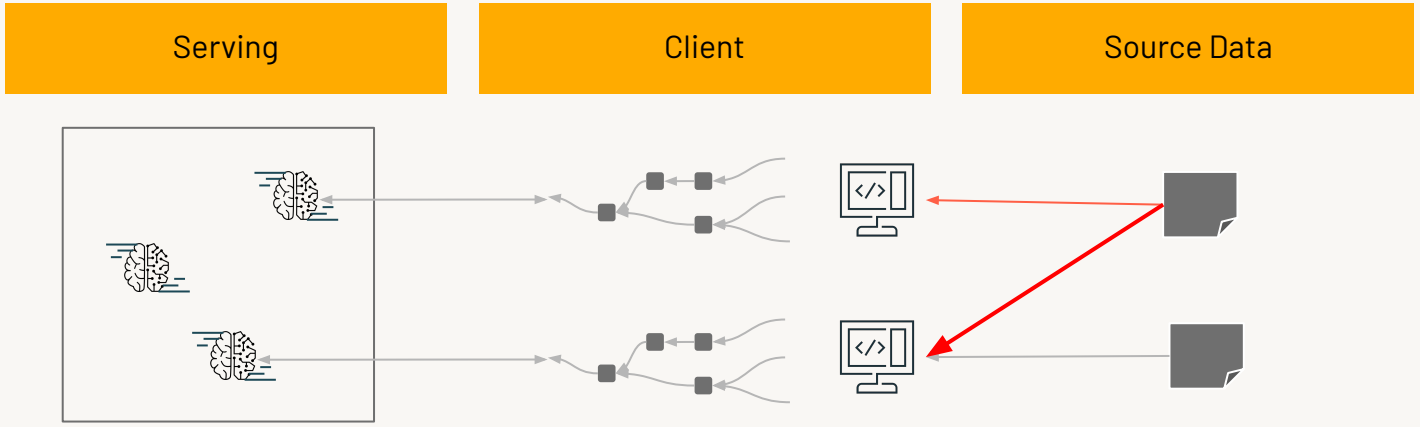
Challenge 1: Data silos

Data Challenges in ML



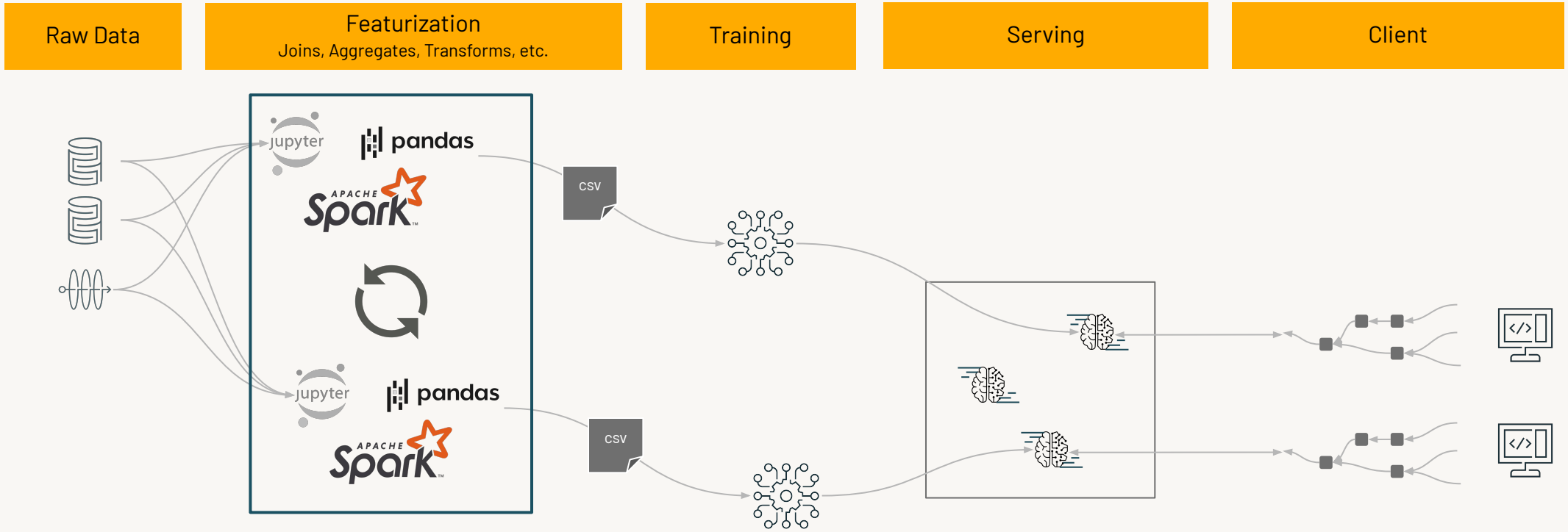
Challenge 2: Online/offline skew

Data Challenges in ML



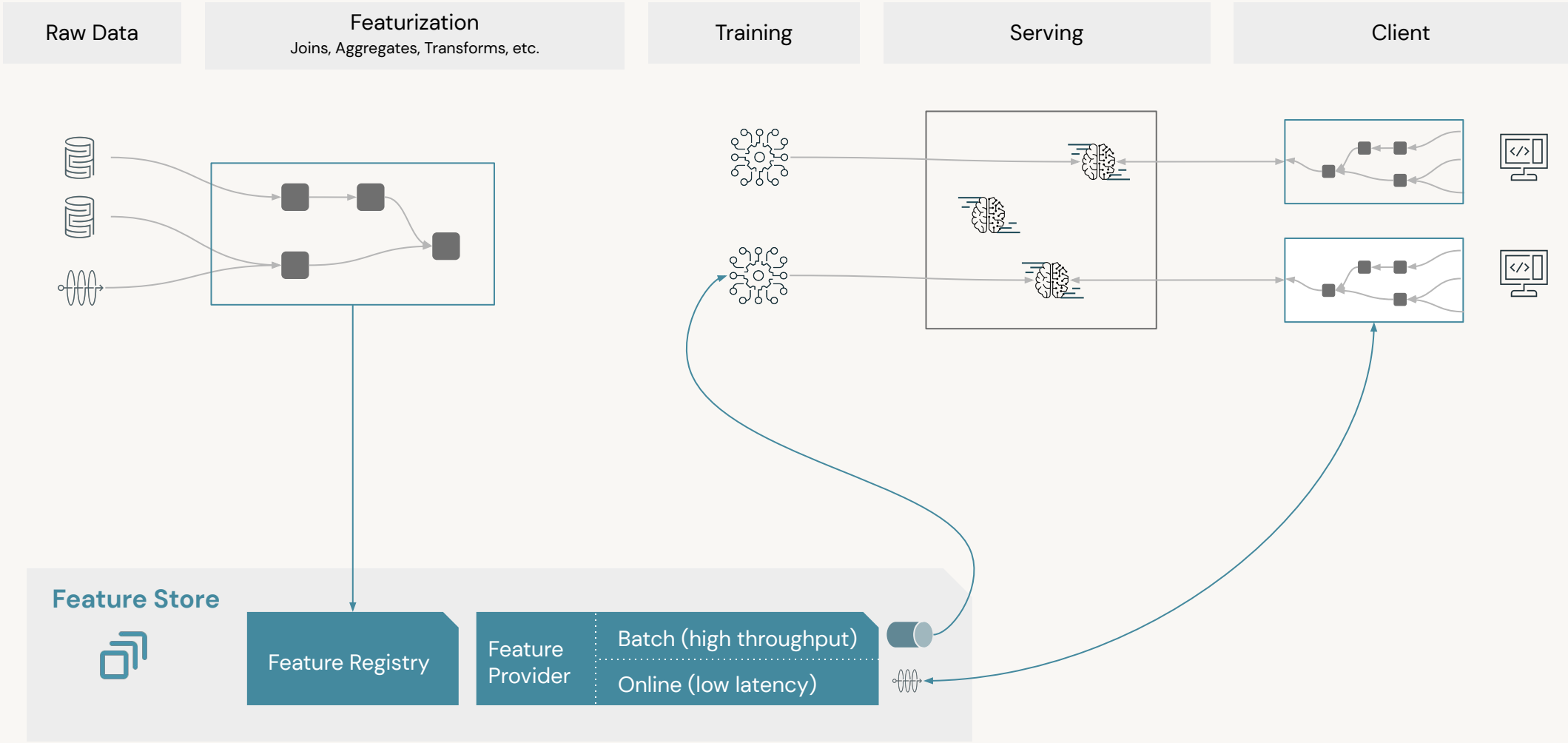
Challenge 3: Client Configuration

Data Challenges in ML



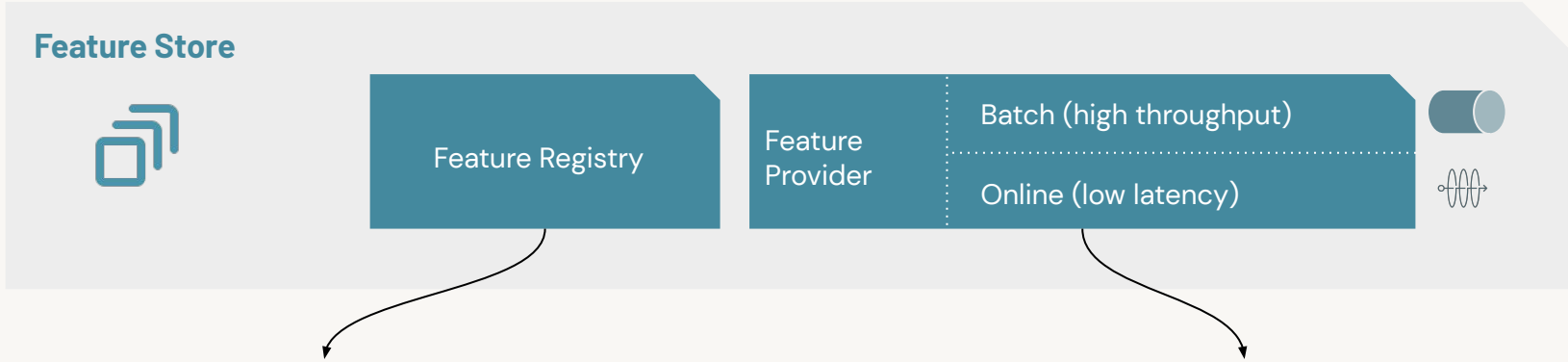
Challenge 4: Data Freshness

Solving the Feature Store Problem



Feature Store

The first Feature Store codesigned with a Data and MLOps Platform



Feature Registry

- Discoverability and Reusability
- Versioning
- Upstream and downstream Lineage

Co-designed with DELTA LAKE

- Open format
- Built-in data versioning and governance
- Native access through PySpark, SQL, etc.

Feature Provider

- Batch and online access to Features
- Feature lookup packaged with Models
- Simplified deployment process

Co-designed with mlflow

- Open model format that supports all ML frameworks
- Feature version and lookup logic hermetically logged with Model

MLflow Tracking

Experiment Tracking

Record runs, and keep track of models parameters, results, code, and data from each experiment and in one place.

Provides:

- Pre-configured MLflow tracking server
- Databricks Workspace & Notebooks UI integration
- S3, Azure Blob Storage, Google Cloud for artifacts storage
- Experiments management via role based Access Control Lists (ACLs)

MLflow tracking and autologging

Ensure reproducibility

Track ML development with one line of code: parameters, metrics, data lineage, model, and environment.

```
mlflow.autolog()
```

/Users/joseph@databricks.com/demos/MLflow/MLflow E... > Run 2e9f9391da1b48d891fa50126143afed

Reproduce Run

Date: 2020-11-23 11:28:01 Source: MLflow Example Notebook User: joseph@databricks.com

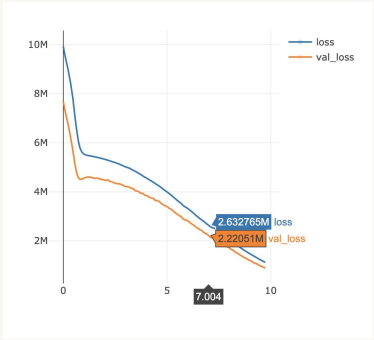
Duration: 3.1s Status: FINISHED

Parameters and tags, including data version

2020-04-06 10:14:06 PDT

- batch_size: 64
- class_weight: None
- epochs: 100
- epsilon: 1e-07
- initial_epoch: 0
- learning_rate: 0.001
- max_queue_size: 10

Metrics



Model, environment, and artifacts

- model
  - data
  - MLmodel
  - conda.yaml
  - model_summary.txt

Full Path: dbfs:/databricks/mlflow/25120103/c39...
Size: 303B

```
artifact_path: model
flavors:
  keras:
    data: data
    keras_module: keras
    keras_version: 2.2.5
  python_function:
    data: data
    env: conda.yaml
    loader_module: mlflow.keras
    python_version: 3.7.3
run_id: c3995babe1754238acd7b241cad12d7c
utc_time_created: '2020-04-06 17:14:18.343005'
```

Analyze results in UI or programmatically

- How does tuning parameter X affect my metric?
- What is the best model?
- Did I run training for long enough?

Workflow overview

- Write code to convert raw data into features and create a Spark DataFrame
- Write the DataFrame as a feature table in Feature Store
- Create a training set based on features from feature tables
- Train model and log it with MLflow
- Perform batch inference on new data. The model automatically retrieves the features it needs from Feature Store.
- (Optional) Publish the features to an online store for batch or real-time inference

02 Model Management

Data Transformation

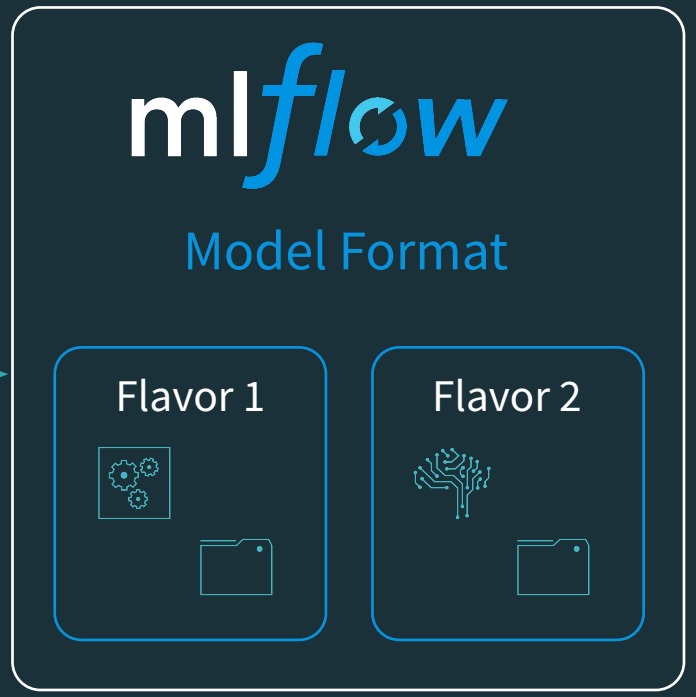
Within the model or prior to model training? It depends!

	Pros	Cons
Prior to Model Training	- Compute once and reuse - Leverages full dataset	- Increases production footprint - Slower to iterate
Within the Model	- Easier to iterate - Smaller production footprint	- Model latency depends upon transformation overheads - Data visibility

mlflow Models

ML Libraries

scikit learn
TensorFlow
Spark
H2O.ai
PyTorch
Keras
mleap
ONNX



Simple model flavors usable by many tools

In-Line Code

docker
kubernetes

Containers

SELDON
mleap
Spark

Batch & Stream Scoring

Azure Machine Learning
Amazon SageMaker

Cloud Inference Services

databricks

End to End Model Management

Setup MLflow Model webhook
Slack notifications, Trigger Testing Jobs

Schedule Monthly Retrain Job
Databricks Job

Data Prep & Featurization
ETL + EDA, Feature Engineering

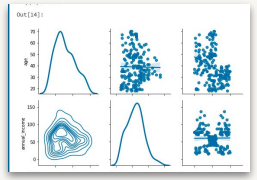
Build baseline Model with AutoML
MLflow autologging, Hyperopt + Spark

Promote Best Run to Registry
Annotate model. Request transition to staging

Automated Model testing
Schema, Demographic accuracy, Docs & artifacts...
Approve/reject request

Run inferences
Load model
Batch or real-time

Feature Store



XGBoost · K · learn · Spark · ...

mlflow Tracking

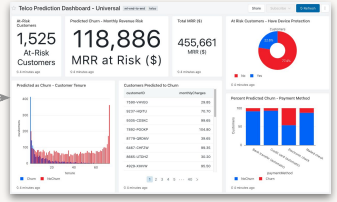
STAGING Request
Model Registry
Webhook triggers test

Approved.
Move to STAGING

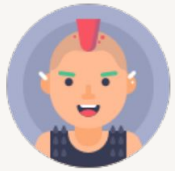
Rejected

mlflow STAGING

{ api }
Realtime
HTTP inference




Data Scientist


ML Engineer


Data Engineer

Model Management with Model Registry

Use one central place to collaboratively manage ML models, from experimentation to online testing and production.

One Collaborative Hub: The Model Registry provides a central hub for making models discoverable, improving **collaboration** and **knowledge sharing** across the organization.

Manage the entire Model Lifecycle (MLOps): The Model Registry provides lifecycle management for models from experimentation to deployment, improving **reliability** and robustness of the model deployment process.

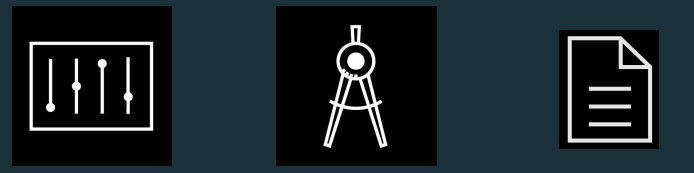
Visibility and Governance: The Model Registry provides full visibility into the deployment stage of all models, who requested and approved changes, allowing for full governance and auditability.

mlflow Model Registry

VISION: Centralized and collaborative model lifecycle management

mlflow

Tracking Server

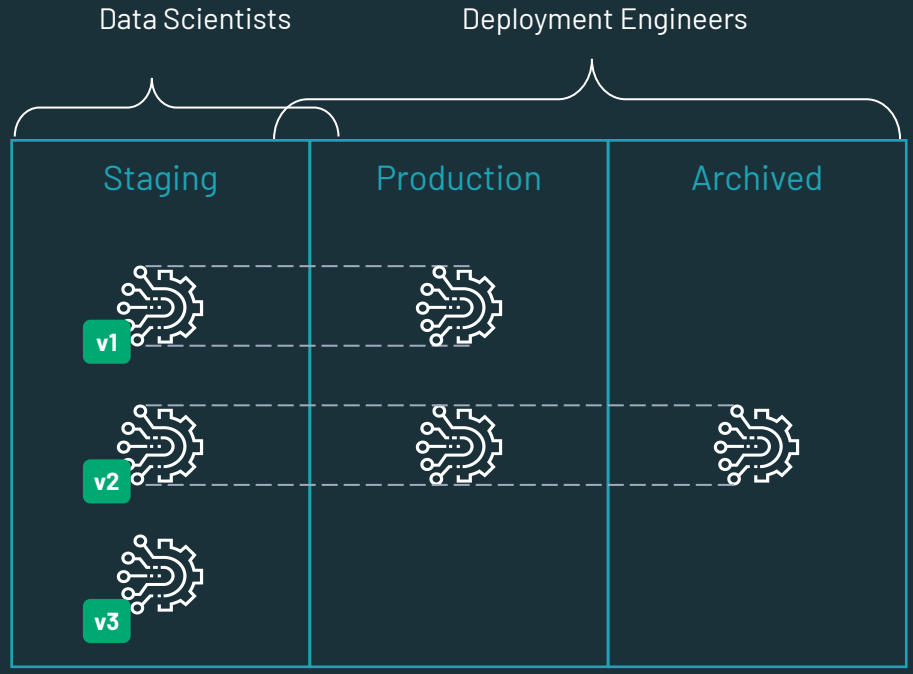


Parameters Metrics Artifacts

Metadata Models

mlflow

Model Registry



databricks

Model Management Lesson

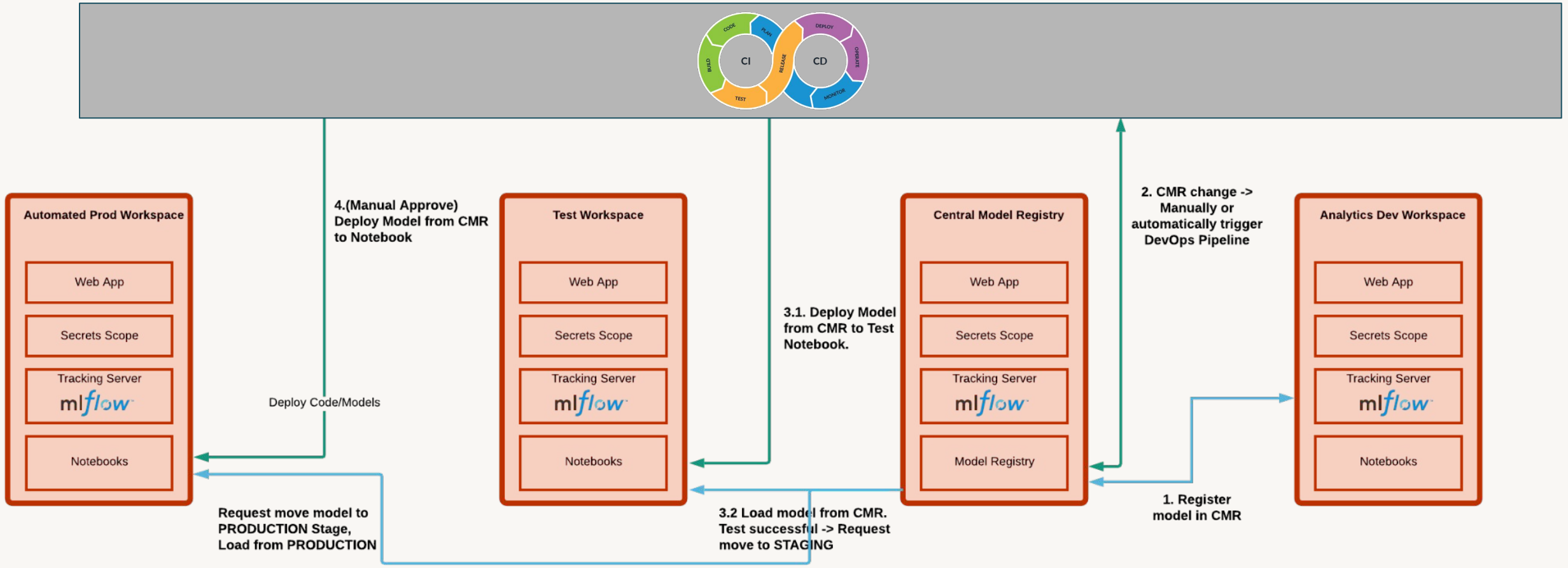
Model Registry Workflow Options

1. **Central Model Registry**: Dedicated workspace(s) to store models and experiments
   a. Refer to this documentation

2. **Per-workspace Model Registries**: Each workspace has its own model registry, model transition is made via CI/CD

Central Model Registry

Dedicated workspace(s) to store models and experiments



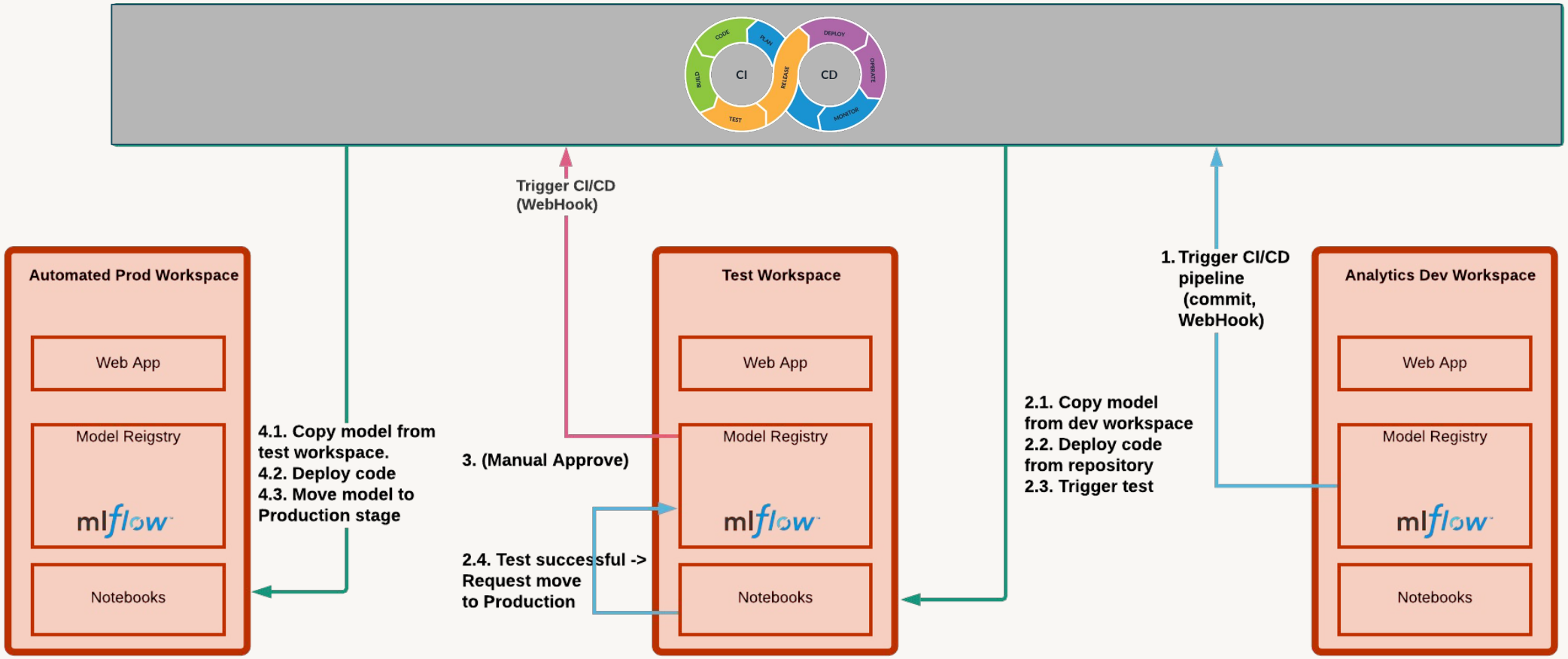
Central Model Registry

Dedicated workspace(s) to store models and experiments

- Pros:
  - Visibility into current state of all models
  - Central management of models
  - Cross cloud
- Cons:
  - Setup is a bit complicated:
    - We need PAT & other configuration in a secret scope
    - We can have max 100 secret scopes per workspace
    - Using shared PATs isn't good from compliance point of view
  - Could be cluttered, especially if we'll use it for tracking of experiments
  - Control of permissions could be complex if we have many teams

Per-workspace Model Registries

Each workspace has its own model registry, model transitions via CI/CD



Per-workspace Model Registries

Each workspace has its own model registry, model transitions via CI/CD

- Pros:
  - Narrowed access to staging/production workspace
  - PAT is required only for a system account (service principals, for example)
- Cons:
  - Limited observability - you need to visit staging/production workspaces to get information about model state
  - Approvals needs to be done in relevant workspaces
  - There are no built-in tools for transitioning of models/experiments between workspaces, although there is some tooling
  - Requires implementation as part of CI/CD pipeline

03 Deployment Paradigms

What is ML Deployment?

- Data Science != ML Engineering
- Data science is scientific
  - Business problems → data problems
  - Model mathematically
  - Optimize performance
- ML engineers are concerned with
  - Reliability
  - Scalability
  - Maintainability
  - SLAs
  - …

DevOps vs. ModelOps

DevOps

Software development + IT operations

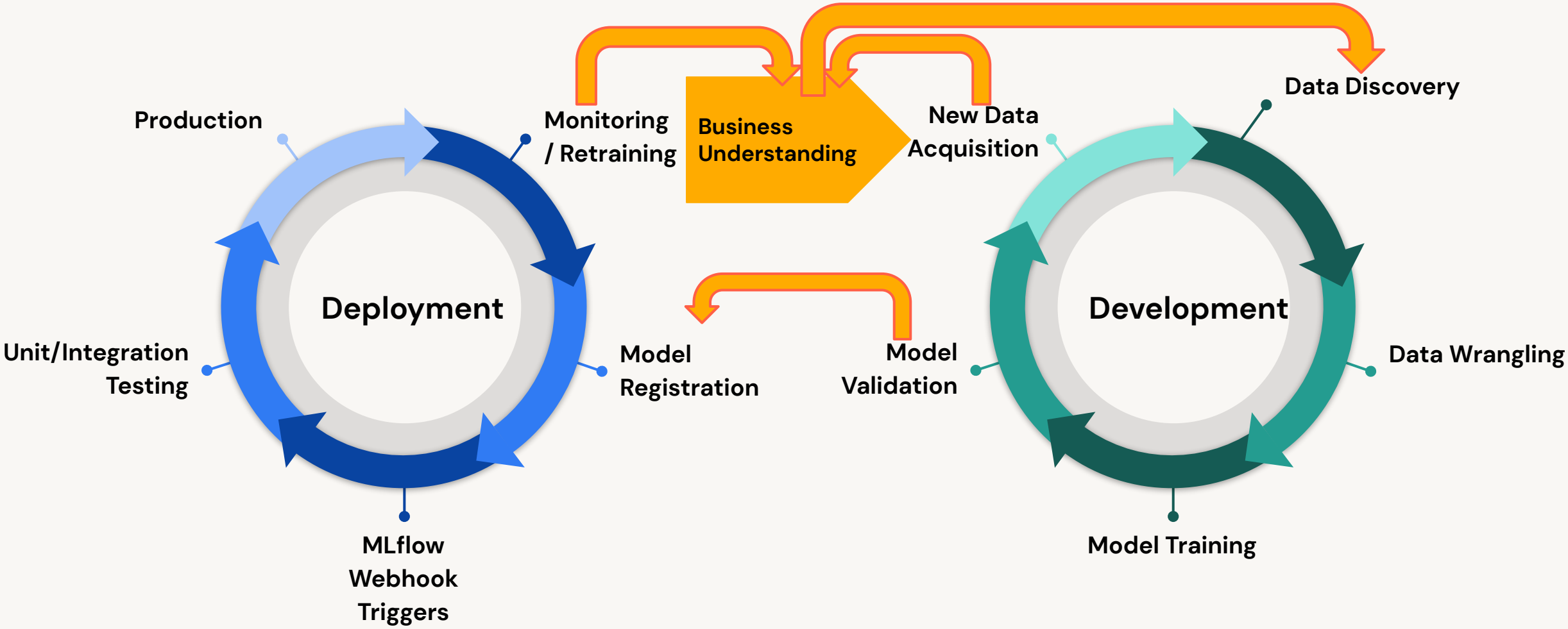
- Manages deployments
- CI/CD of features, patches, updates, rollbacks

ModelOps

Data modeling + deployment operations

- Artifact management (Continuous Training)
- Model performance monitoring (Continuous Monitoring)
- Data management
- Use of containers and managed services

Closed Loop Systems

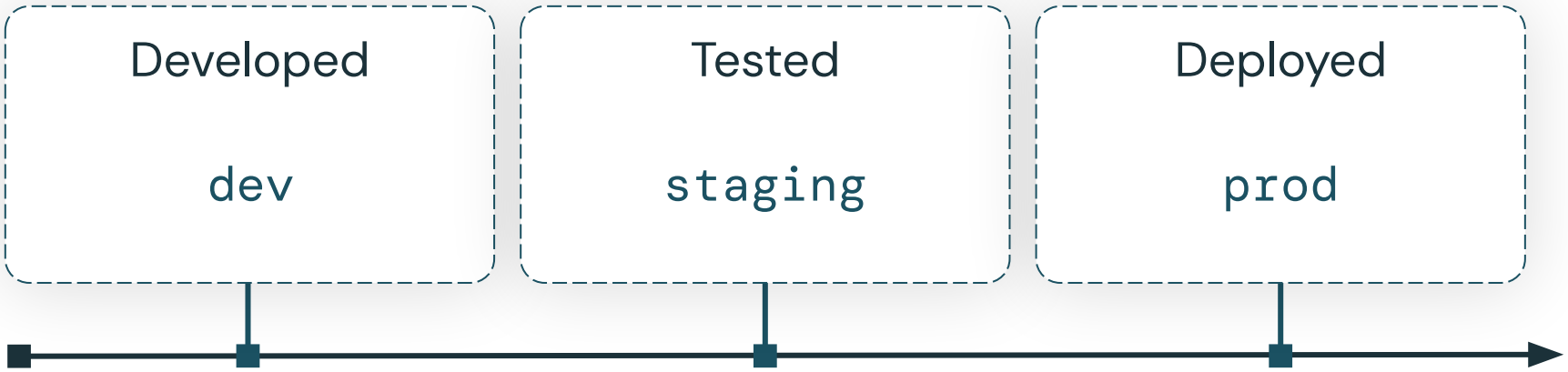


Deployment Patterns

Semantics of dev, staging, and prod

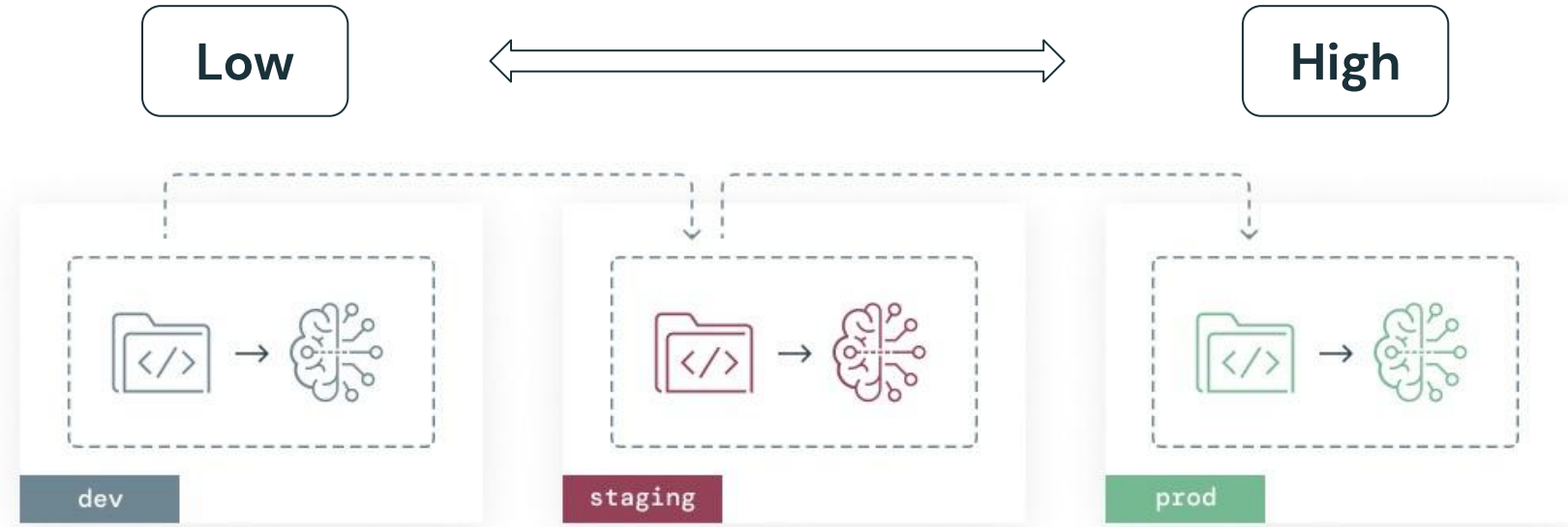
ML Workflow Assets:

Assets need to be:



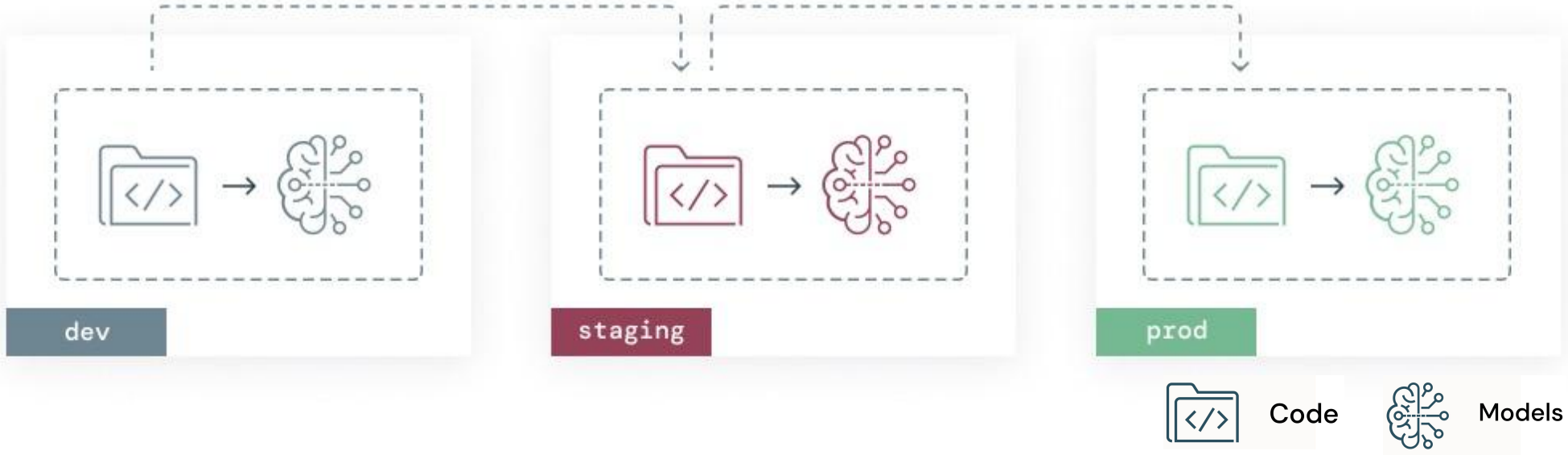
Dev vs. staging vs. prod

Level of trust, quality and testing:



Openness of access:

Model vs. code lifecycles



Model and code lifecycles often operate asynchronously

- Weekly fraud detection model
  - Model update; no code change
- Computer vision model, large language model
  - No model update; code change

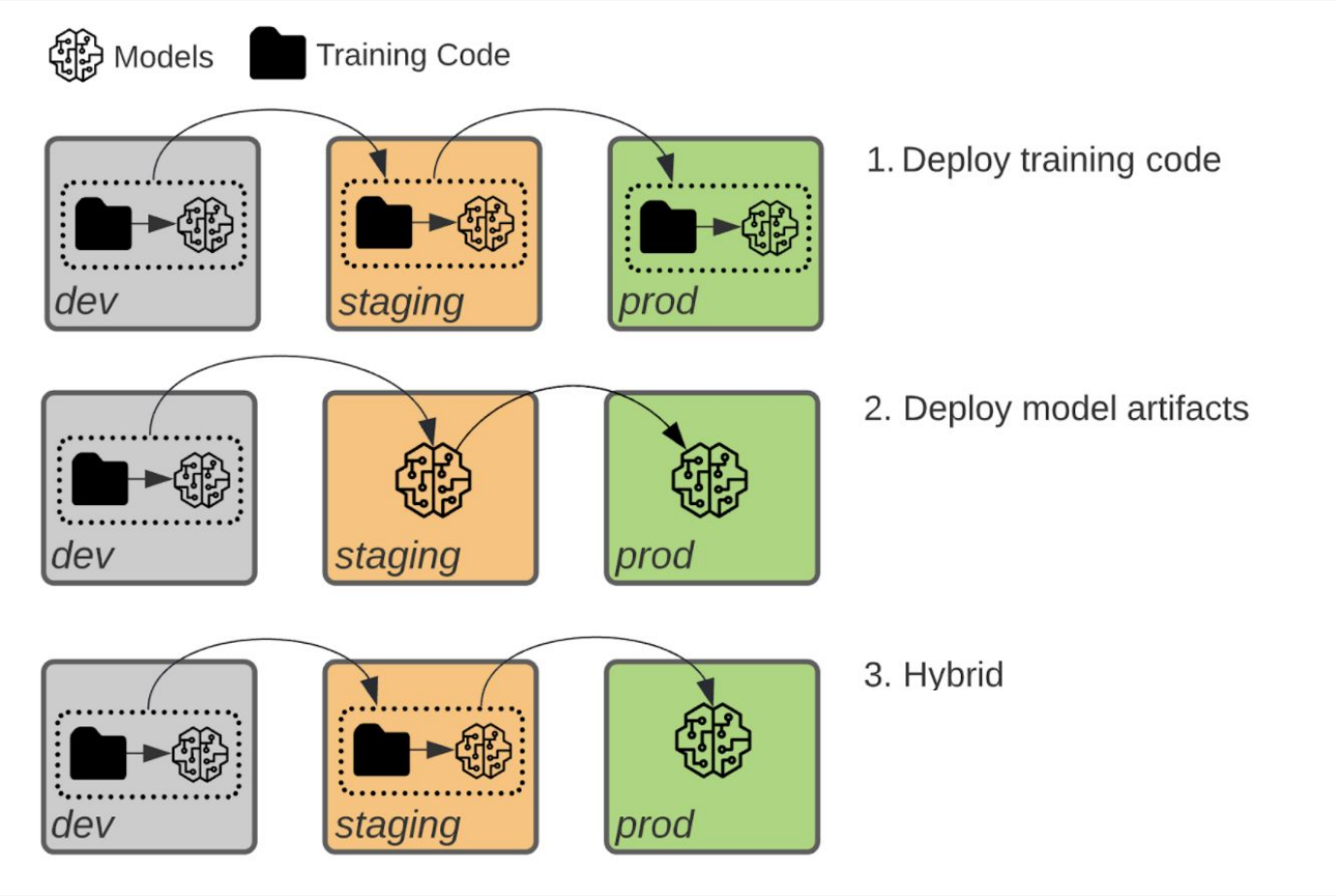
→ mlflow™

Dev vs. staging vs. prod: managing assets

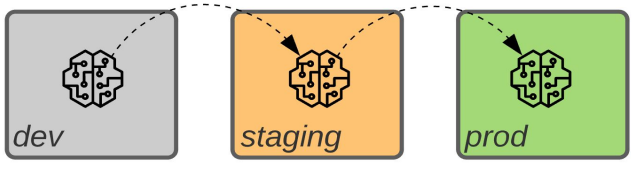
ASSET	SEMANTICS	SEPARATED BY
Execution environments	Labeled according to where development, testing and connections with production systems happen	Cloud provider and Databricks Workspace access controls
Models	Labeled according to model lifecycle phase	MLflow access controls or cloud storage permissions
Data	Labeled according to its origin in dev, staging or prod execution environments	Table access controls or cloud storage permissions
Code	Labeled according to software development lifecycle phase	Git repository branches

Three model deployment patterns

More information on The Big Book of MLOps published by Databricks

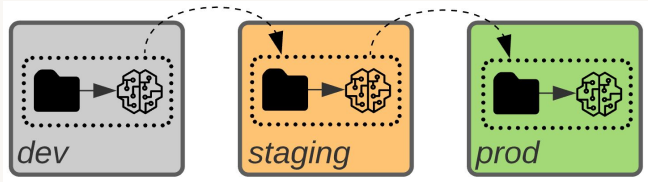


What moves towards production?



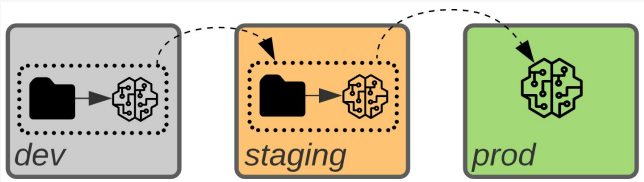
	Deploy models
Process	1. Models are trained in dev and promoted to staging 2. Models are tested in staging 3. Models are promoted to prod
Trade-offs	↑ Simplicity ↑ DS familiarity ↑ Computational cost ↓ Automation ↓ Scalability ↓ Reproducibility

What moves towards production?



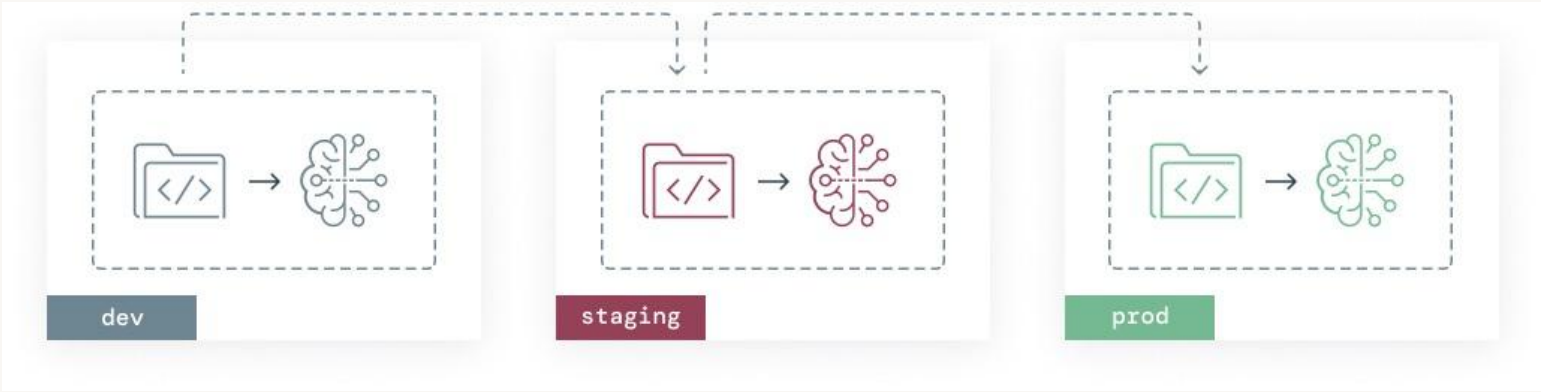
	Deploy code
Process	1. Code is promoted from dev to staging 2. Models are retrained and tested in staging 3. Code is promoted to prod 4. Models are retrained in prod
Trade-offs	↑ Only prod env needs prod data ↑ Automation ↑ Reproducibility ↑ Scalability ↓ Engineering setup and maintenance ↓ DS familiarity

What moves towards production?



	Hybrid
Process	1. Code is promoted from dev to staging 2. Models are retrained and tested in staging 3. Models are promoted to prod
Trade-offs	↑ Scalability ↑ Automation ↑ Reproducibility ↑ ML Eng familiarity ↑ Computational cost ↓ Simplicity ↓ DS familiarity

Recommend deploy code pattern



dev

Develop training code
Develop ancillary code
→ Promote code.

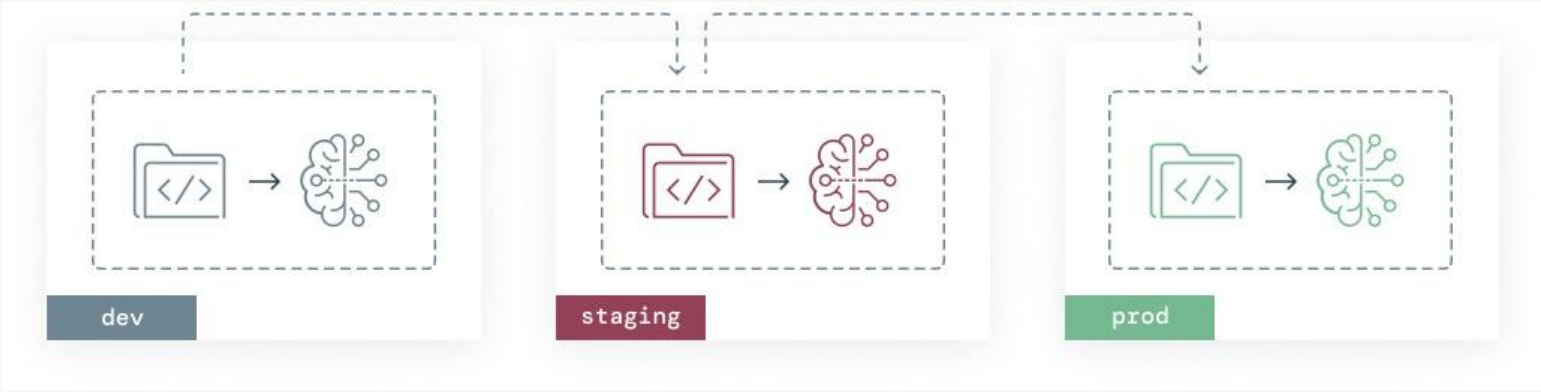
staging

✔ Test model training code on subset of data
✔ Test ancillary code
→ Promote code.

prod

✔ Train model on production data
✔ Test model
→ Deploy model
→ Deploy ancillary code

Deeper dive into benefits of “deploy code”



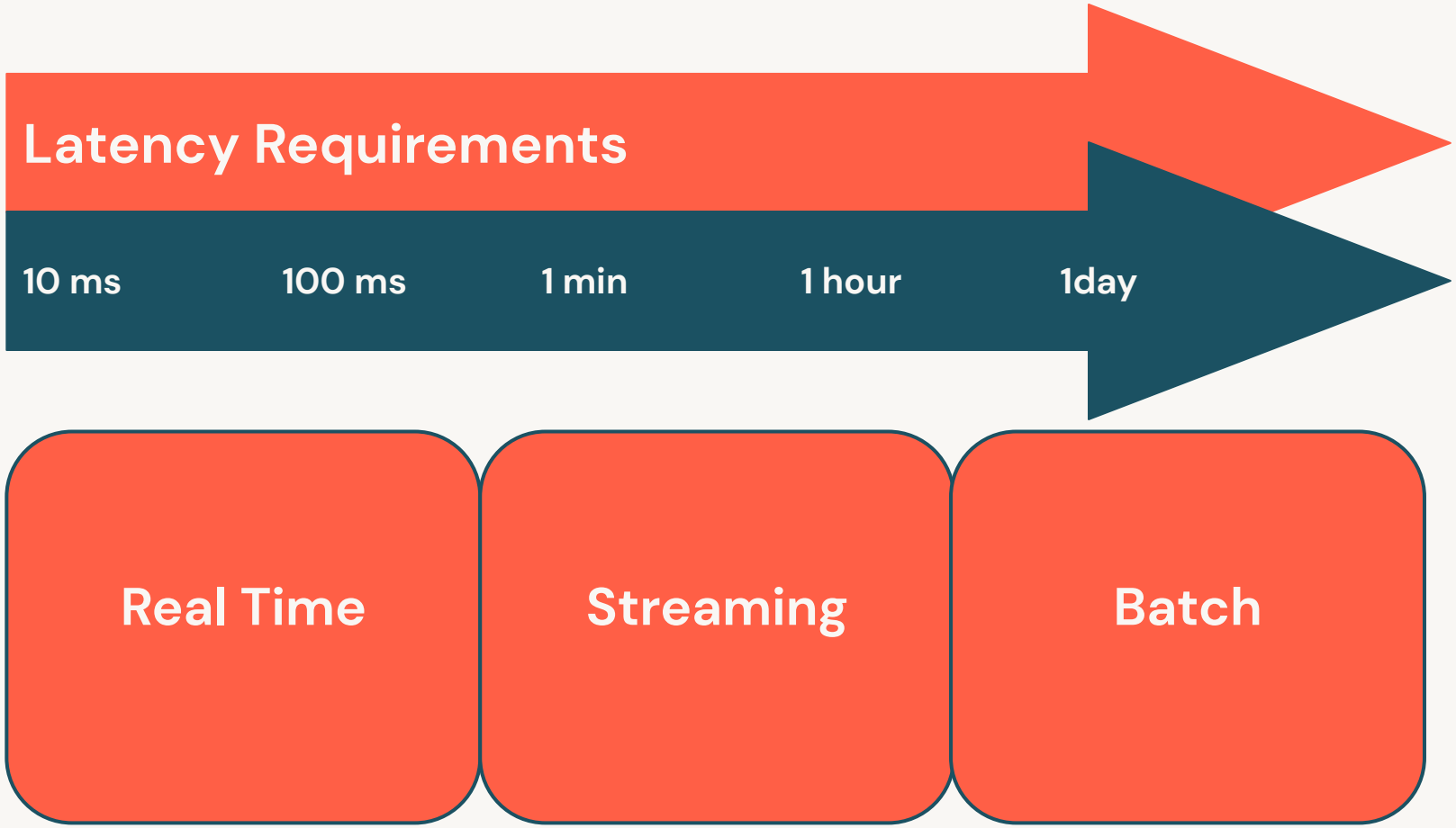
Automation	↑	Supports automated retraining in locked-down env.
Data access control	↑	Only prod env needs read access to prod training data.
Reproducible models	↑	Eng control over training env, which helps to simplify reproducibility.
Support for large projects	↑	This pattern forces the DS team to use modular code and iterative testing, which helps with coordination and development in larger projects.
Data science familiarity	↓	DS team must learn to write & hand off modular code to Eng.
Eng setup & maintenance	↓	Requires CI/CD infra for unit and integration tests, even for one-off models.

Deployment Paradigms

The Four Deployment Paradigms

1. **Batch**
   - 80-90% of deployments
   - Leverages databases and object storage
   - Fast retrieval of stored predictions
2. **Streaming (continuous)**
   - 10-15% of deployments
   - Moderately fast scoring on new data
3. **Real Time**
   - 5-10% of deployments
   - Usually using REST (Azure ML, SageMaker, containers)
4. **On-device (edge)**

Latency Requirements (roughly)

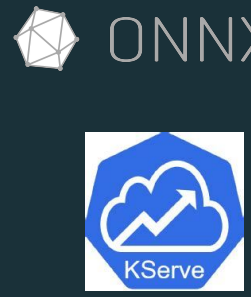


mlflow Model Deployment Options

In-Line Code

Containers

Batch & Stream Scoring



OSS Inference Solutions

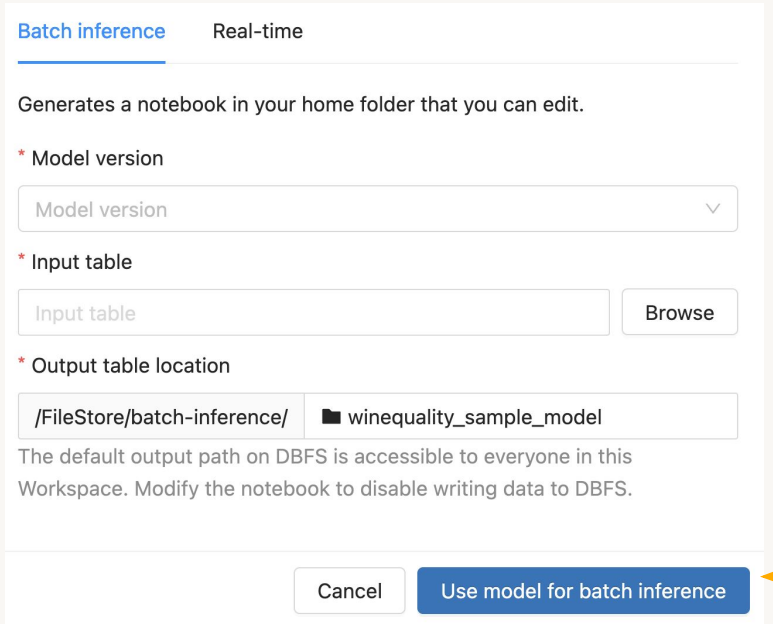
Cloud Inference Services

Online/Offline Model Serving with Databricks

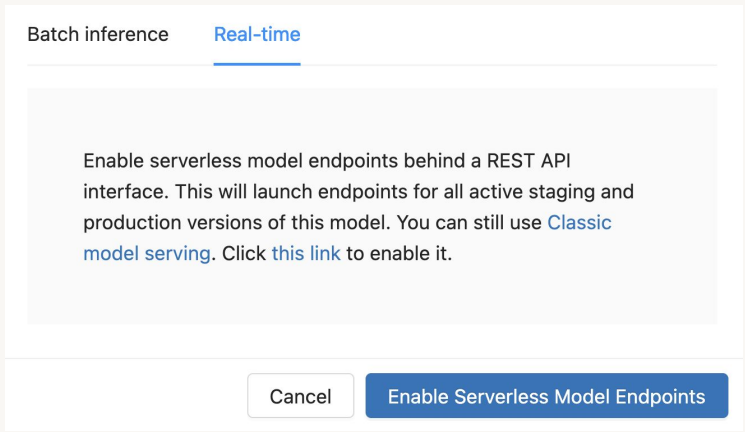
Deploy any ML model at large scale AND low latency

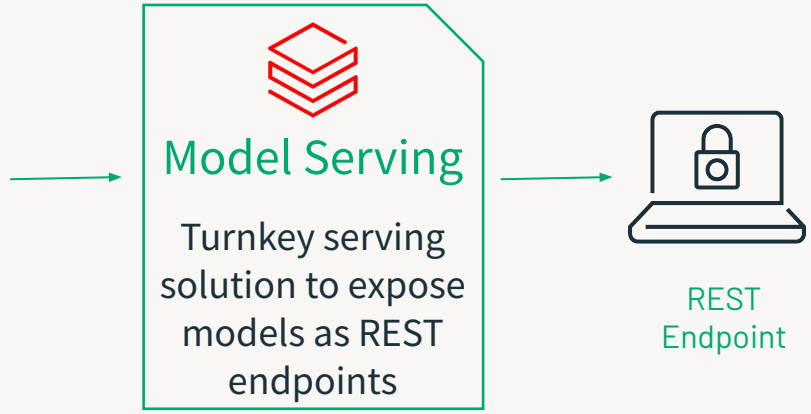
Use model for inference — One click model deployment directly from the Model Registry

Large-scale batch scoring

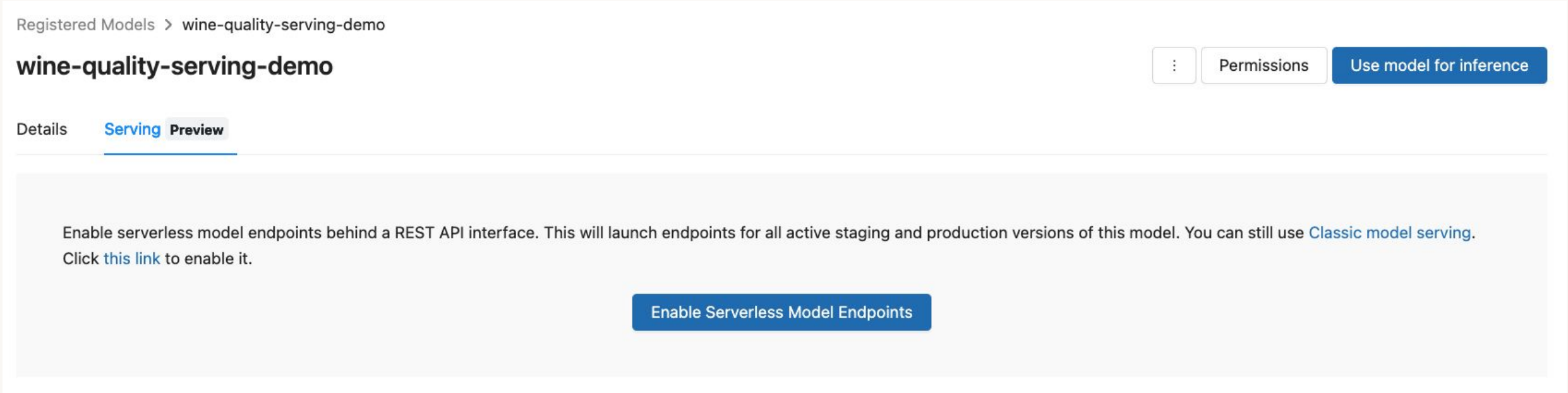


Low-latency online serving

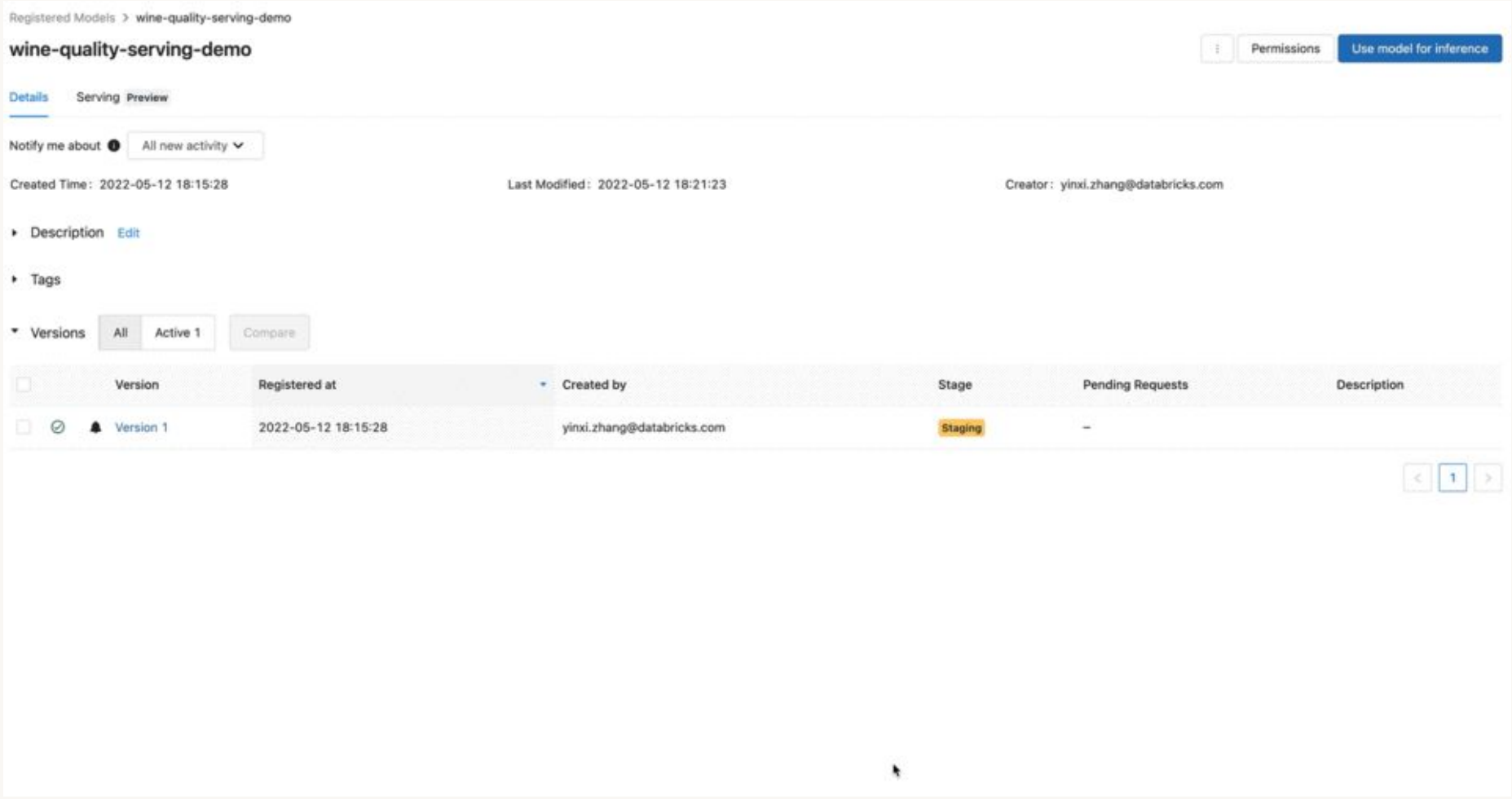




Enable model serving with 1 click



Validate Model Serving with Input Examples



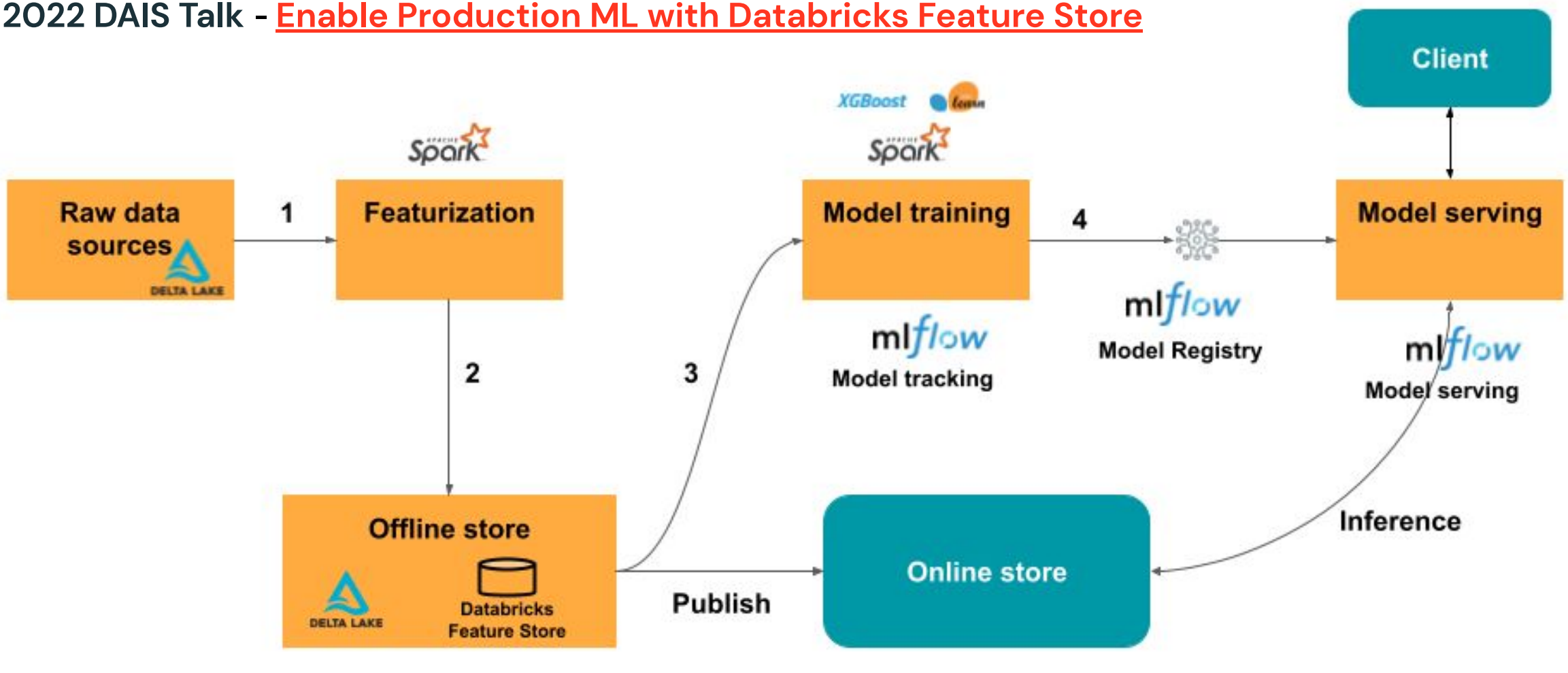
Serving Models on Databricks

Serverless Model Endpoint - Preview in AWS & Azure

- Production-grade serving
- Low latency, high availability and scalable
- HTTP endpoints accessible with Databricks authentication
- Scoring validation from the UI (same UI across clouds)
- 1-click enable serving
- Autoscaling clusters
- Endpoint latency & QPS monitoring

Serve Feature Store Models Online

2022 DAIS Talk - [Enable Production ML with Databricks Feature Store]



04 Production

Production Requirements

Core Requirements

- Model registry
- Data and model drift
- Interpretability
- Reproducibility
- Security
- Environment management

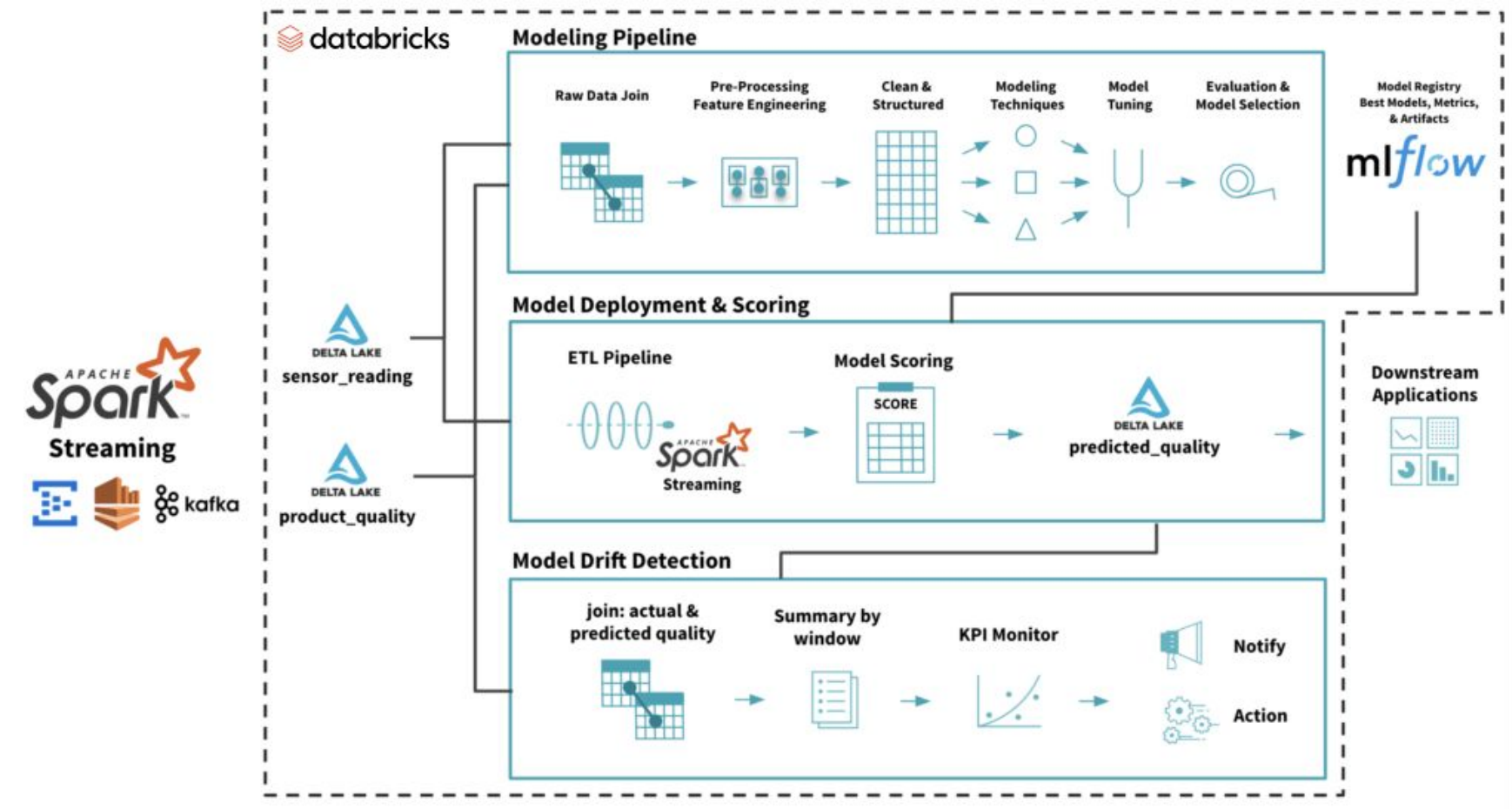
Core +

- ML pipeline with featurization logic
- CI/CD pipeline for automation
- Monitoring and alerting
- Testing framework
- Version control

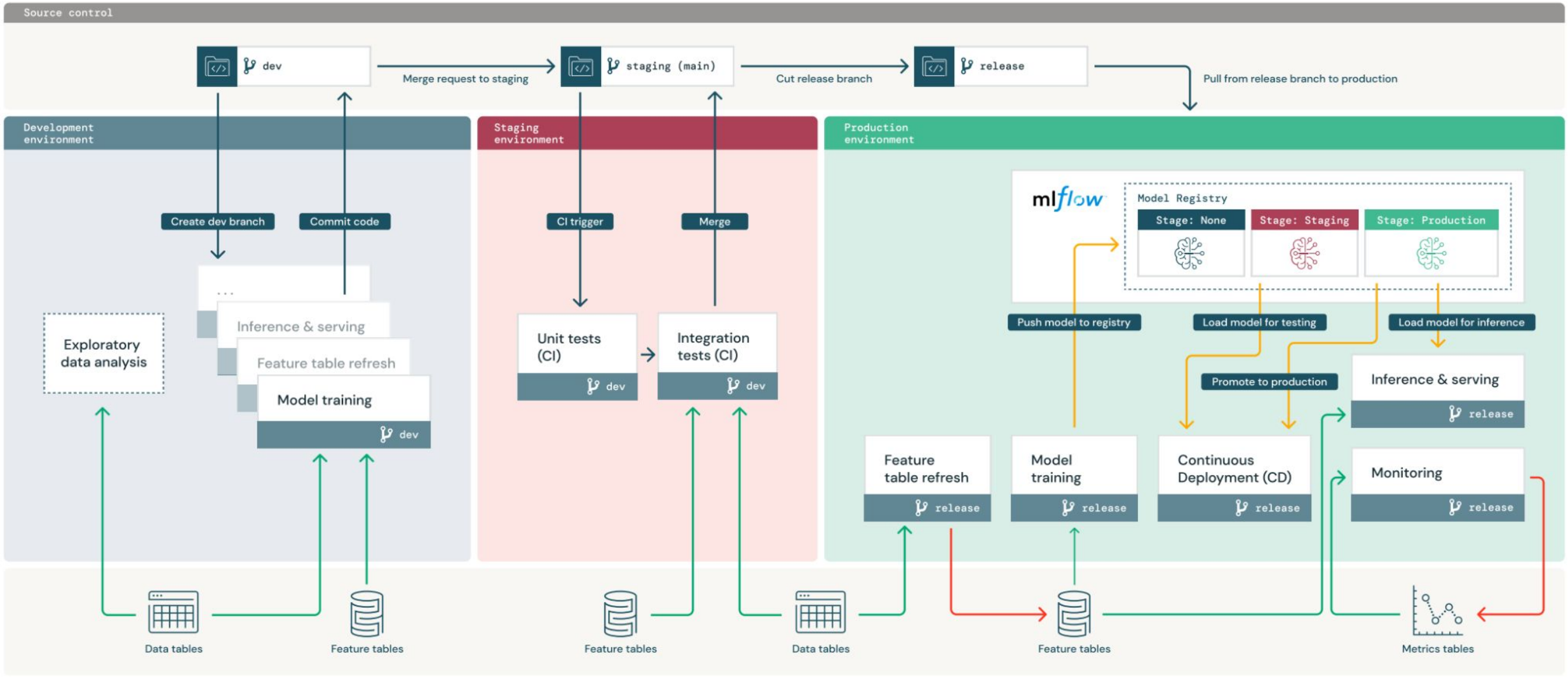
Specialized

- Feature dictionary
- Cost management
- A/B testing
- Performance optimization

Architecture Example

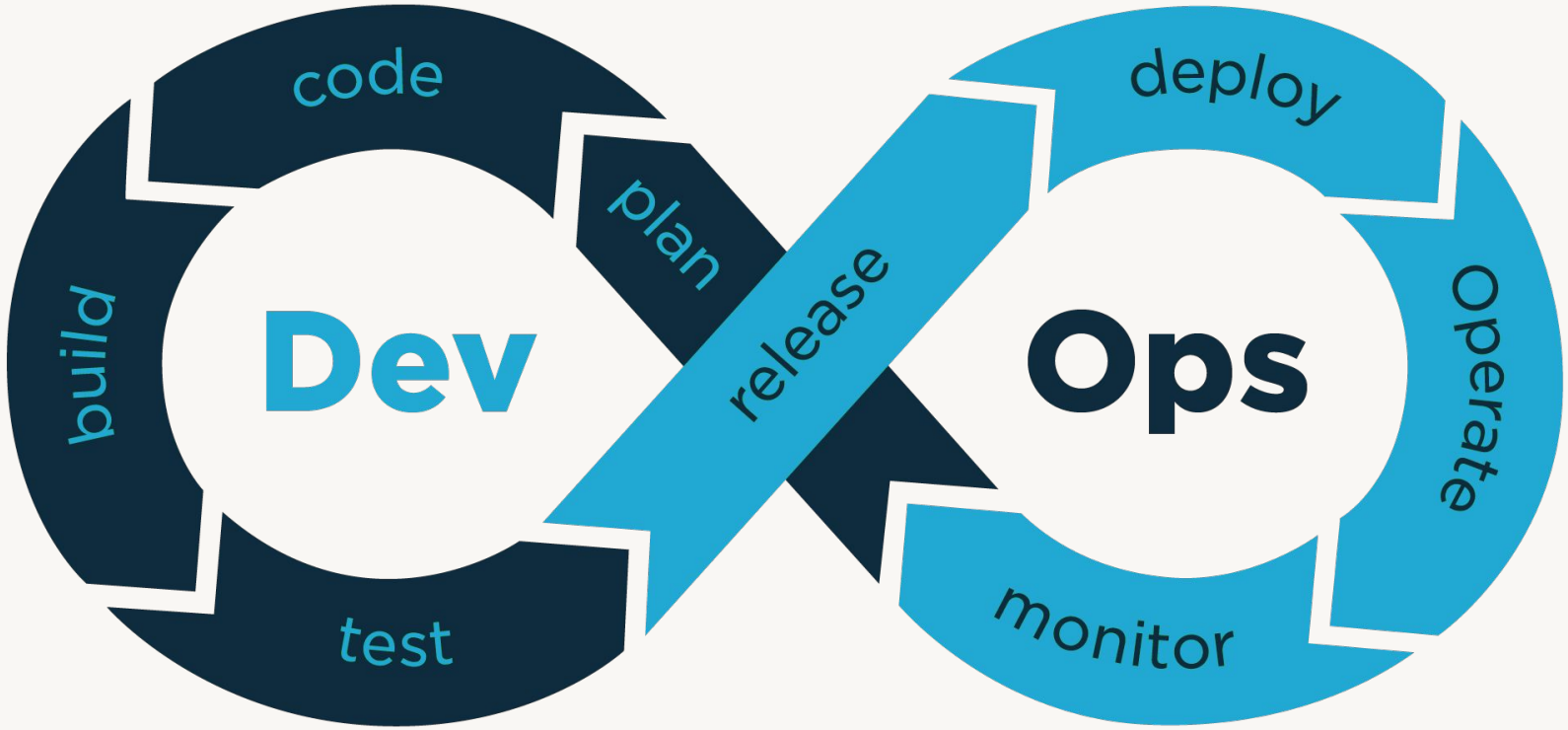


Architecture Perspective

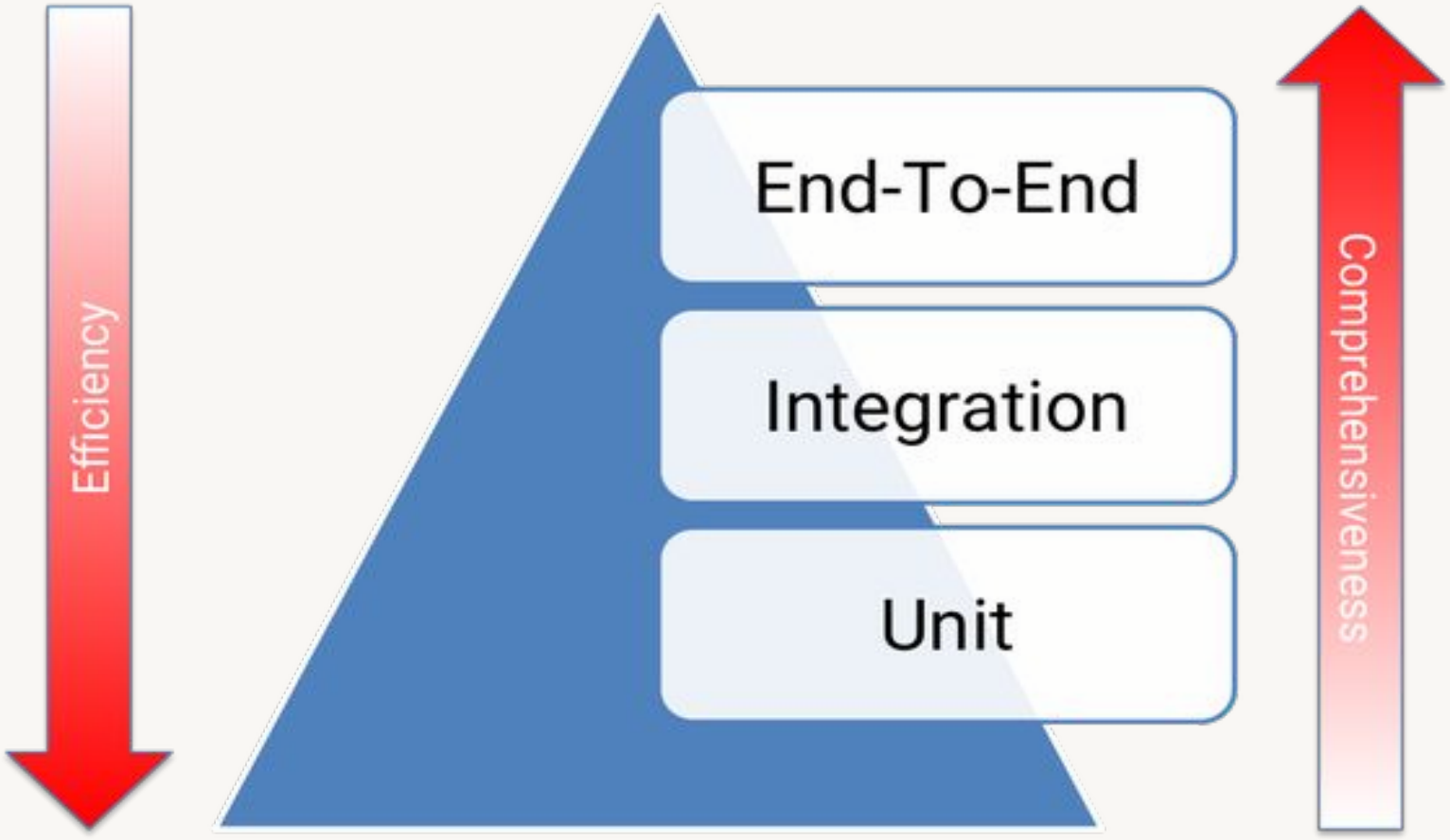


CI/CD

DevOps...



… And the Test Pyramid…



Overview of a typical Databricks CI/CD pipeline

Continuous integration

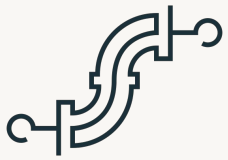
Continuous delivery

Code

Build

Release

Deploy



Test

Operate

Types of Pipelines

- Scheduled by a workflow manager

- Types of pipelines
  - ETL: heavy lifting data transformation
  - Featurization: Generate features for model development
  - Training: Produce models for deployment
  - Scoring: Inference pipeline post deployment
  - Monitoring: recurring jobs post deployment with alerts set up

Key Benefits of Continuous Integration

- Regressions are captured early by the automated tests
- Building the release is easy as all integration issues have been solved early
- Your QA team spends less time testing and can focus on significant improvements to the quality culture

Key Benefits of Continuous Delivery

- The complexity of deploying software has been taken away
- You can release more often, thus accelerating the feedback loop with your customers
- There is much less pressure on decisions for small changes, hence encouraging iterating faster

CI/CD terminology

- **Build pipeline** – set of steps that build, test (usually only unit tests) & package code or notebooks
- **Release pipeline** – set of steps that performing deployment of the assets
- **Trigger** – event or action that initiates the execution of pipeline. It could be a commit to repository, or explicit execution of the pipeline
- **Asset / Artifact** – deliverable produced by build pipeline, or some other process (model training)
- **Environment** – typically:
  - Development – where development happens
  - Staging – for integration & E2E testing
  - Production – actual work

CI/CD Technologies

	OSS Standard	Databricks	AWS	Azure	Third Party
Orchestration	Airflow, Jenkins	Jobs, notebook workflows	CodePipeline, CodeBuild, CodeDeploy	DevOps, Data Factor	
Git Hooks		MLflow Webhooks			Github Actions, Gitlab, Travis CI
Artifact Management	PyPi, Maven	MLflow Model Registry			Nexus
Environment Management	Docker, Kubernetes, Conda, pyenv		Elastic Container Repository	Container Registry	DockerHub
Testing	pytest				Sonar
Alerting		Jobs	CloudWatch	Monitor	PagerDuty, Slack integrations

Databricks Resources

- Implementing CI/CD on Databricks Using Databricks Notebooks and Azure DevOps Part 1, Part 2 (blog)
- Continuous Integration and Delivery on Databricks using Jenkins (blog)
- Automate Continuous Integration and Continuous Delivery on Databricks using Databricks Labs CI/CD Templates (blog)

Recommended books

- Machine Learning Engineering in Action by Ben Wilson
- Designing Data-Intensive Applications by Martin Kleppmann
- Clean Code: A Handbook of Agile Software Craftsmanship by Robert C. Martin
- Refactoring: Improving the Design of Existing Code by Martin Fowler, Kent Beck, Don Roberts
- Test-Driven Development: By Example by Kent Beck
- Code Complete, 2nd ed. by Steve McConnell
- The Pragmatic Programmer: From Journeyman to Master by Andy Hunt, Dave Thomas

Monitoring

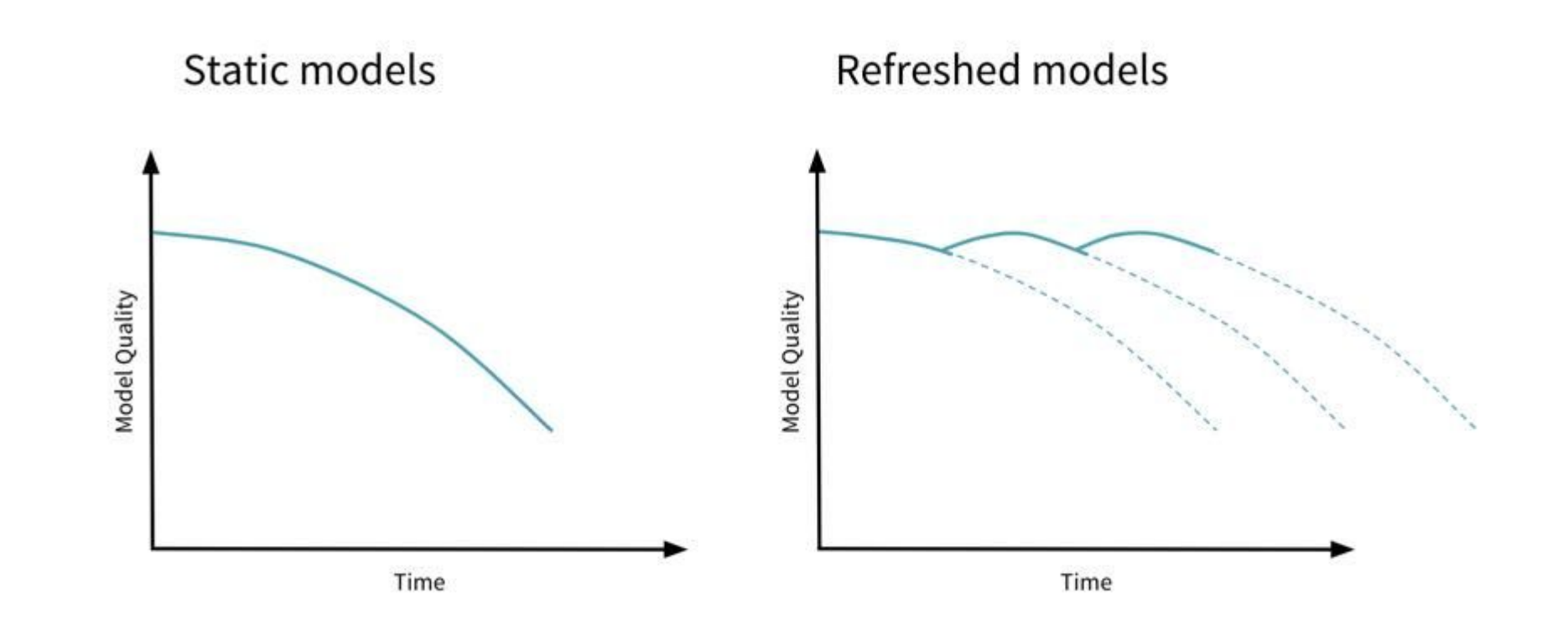
Big Data & AI Present new challenges…

3 changeable ingredients go into training an ML model…

Ingredient	Frequency of change	Well defined process/tools for managing change?
Code	~Daily?	Yes!
Configuration	~Daily/Weekly?	Emerging
Data	~Every second?	No!

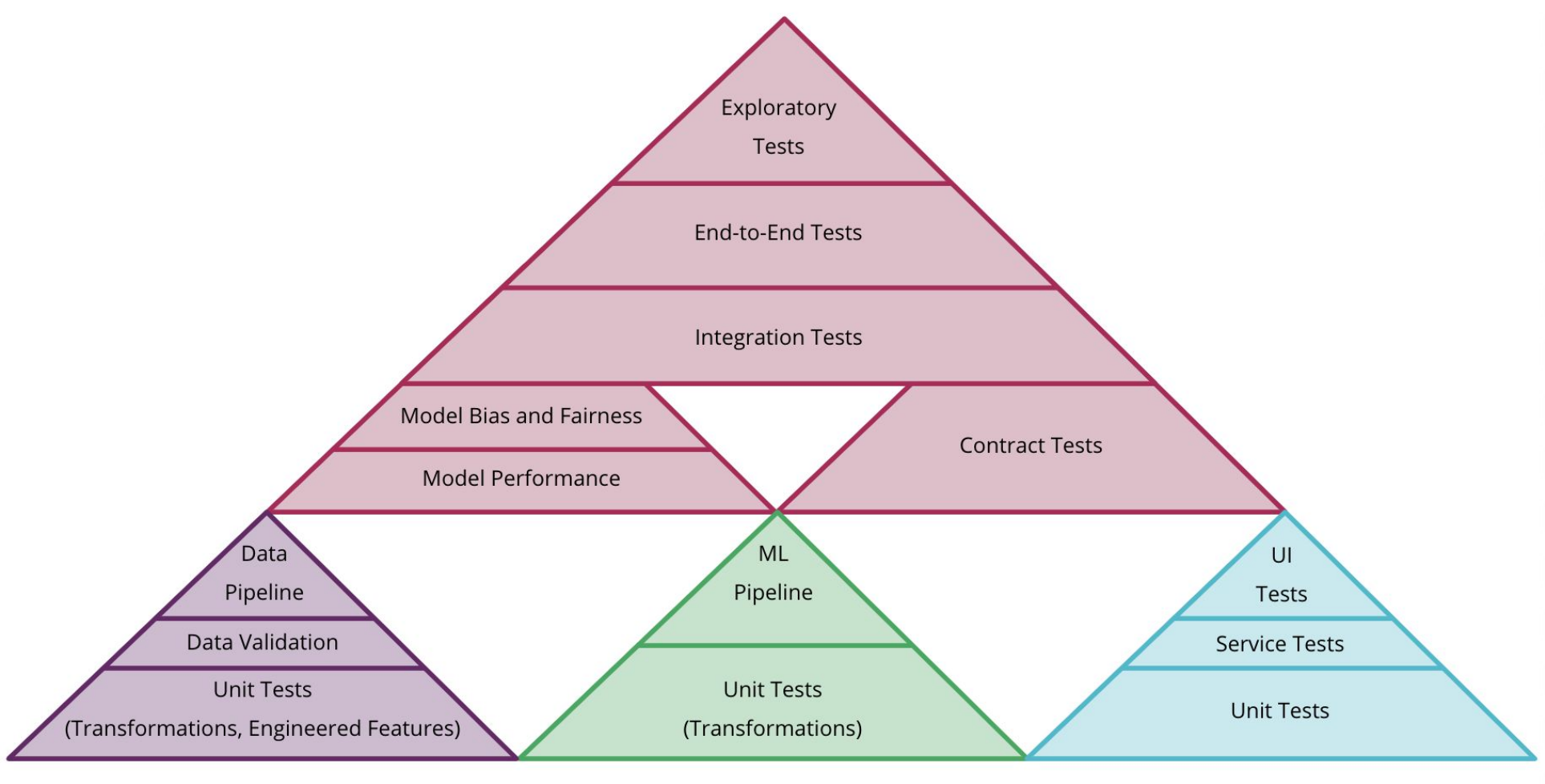
Why do ML projects fail in production?

Neglect maintenance: Lack of re-training and testing



Source

And these make for one complicated test pyramid...



MLOps: Continuous …

- Continuous Integration (CI) extends the testing and validating code and components by adding testing and validating data and models
- Continuous Delivery (CD) concerns with delivery of an ML training pipeline that automatically deploys another the ML model prediction service.
- **Continuous Training** (CT) automatically retrains ML models for redeployment
- **Continuous Monitoring** (CM) concerns with monitoring production data and model performance metrics, which are bound to business metrics

Types of drift

Feature Drift

Input feature(s) distributions deviate

Label Drift

Label distribution deviates

Prediction Drift

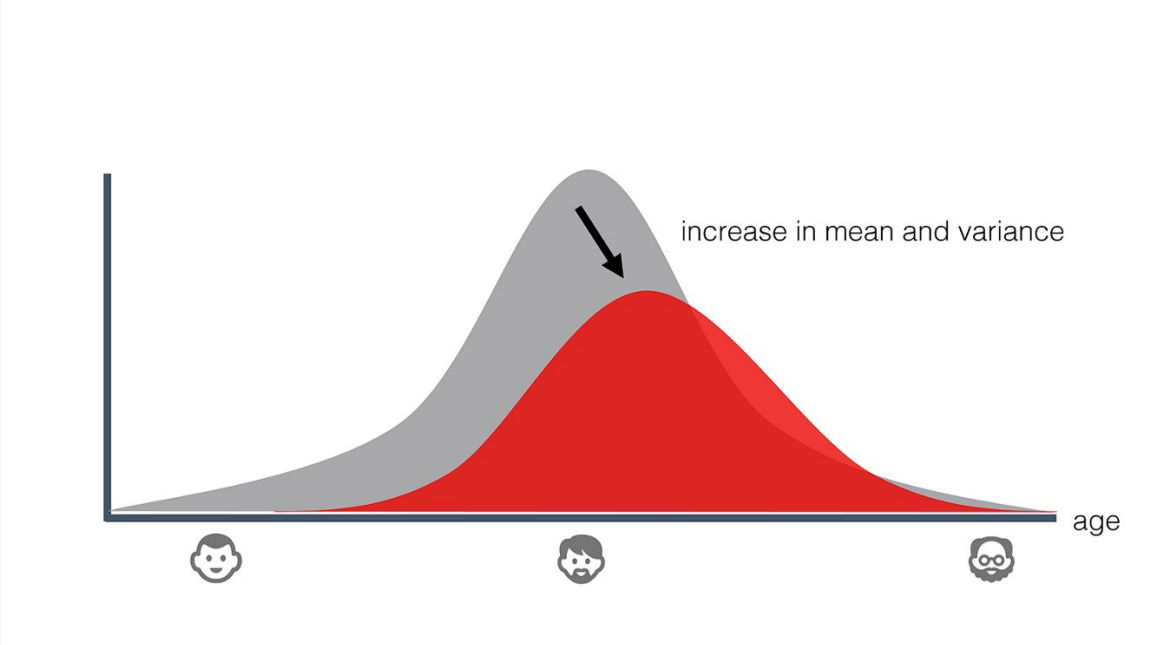
Model prediction distribution deviates

Concept Drift

External factors cause the label to evolve

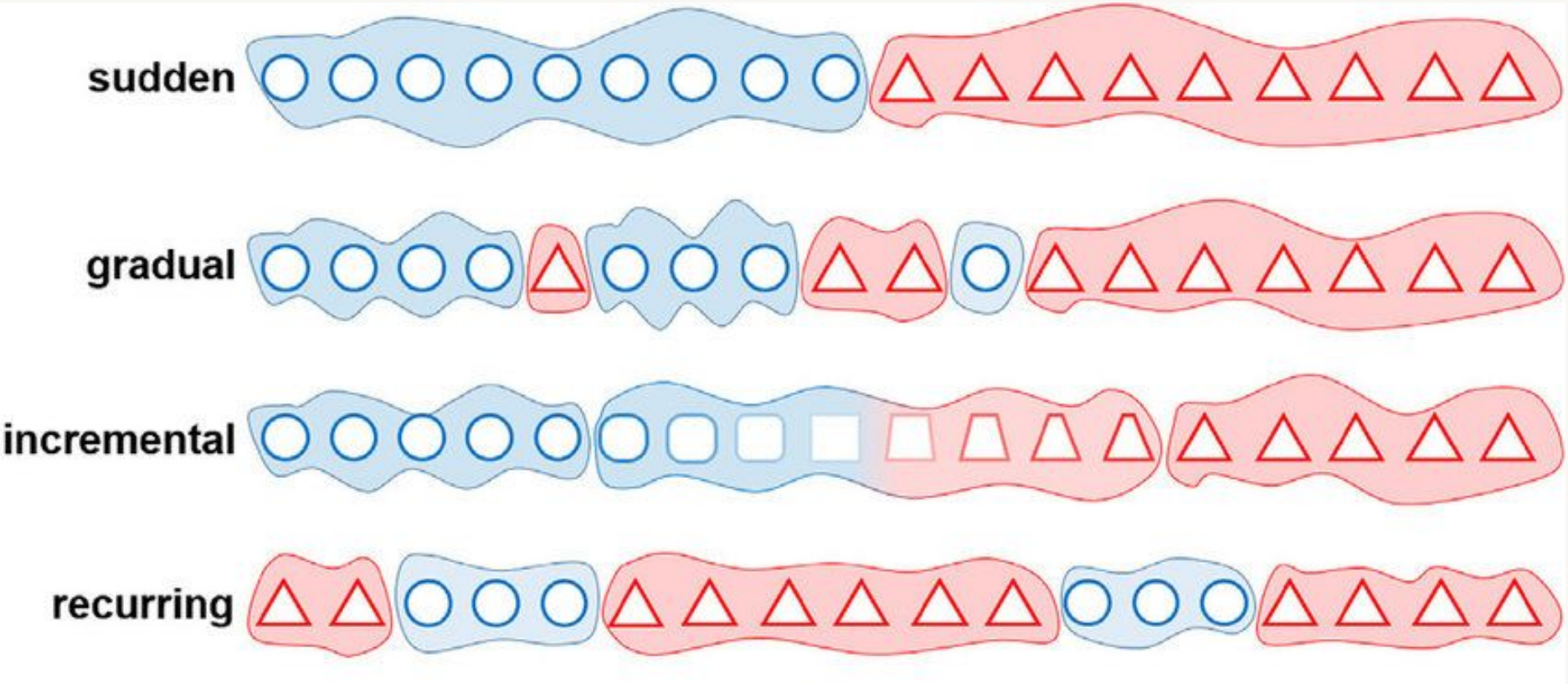
Feature, Label, and Prediction Drift

Categories	Expected	Observed	Total
A	25	35	60
B	25	20	45
C	25	25	50
D	25	20	45
Total	100	100	200



Source

Concept drift



Source

Drift types and actions to take

Drift Type Identified	Action
Feature Drift	• Investigate feature generation process • Retrain using new data
Label Drift	• Investigate label generation process • Retrain using new data
Prediction Drift	• Investigate model training process • Assess business impact of change in predictions
Concept Drift	• Investigate additional feature engineering • Consider alternative approach/solution • Retrain/tune using new data

What to monitor?

Software Metrics

- Memory
- Compute
- Latency
- Throughput
- Server load

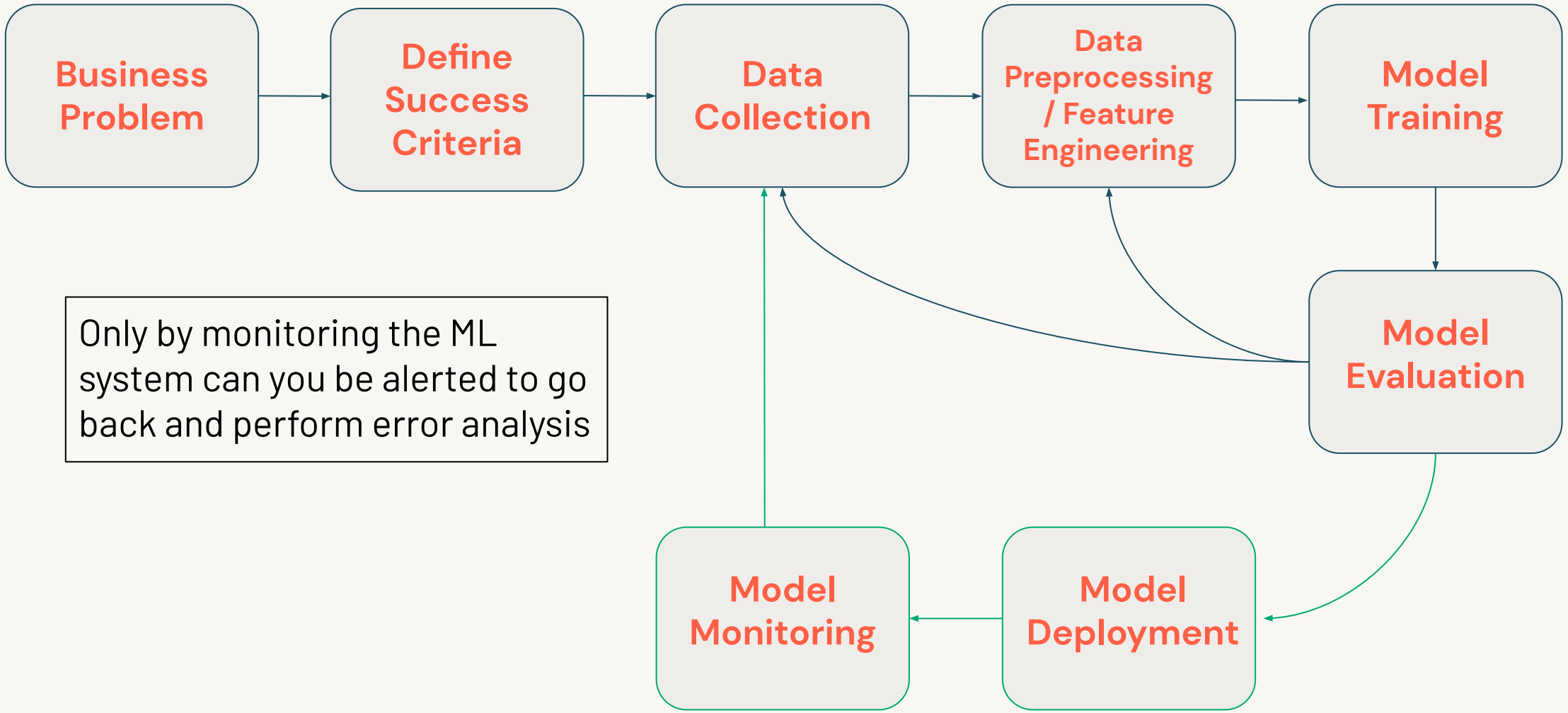
Input Metrics

- Feature summary stats
  - Mean/Median/Min/Max
  - Num missing vals
- Statistical tests
  - KS test/Mann Whitney
  - Jensen Shannon Distance
  - Levene

Output Metrics

- Prediction summary stats
  - Mean/Median/Min/Max
- Business metrics

How to action monitoring metrics



Model Rollout Strategies

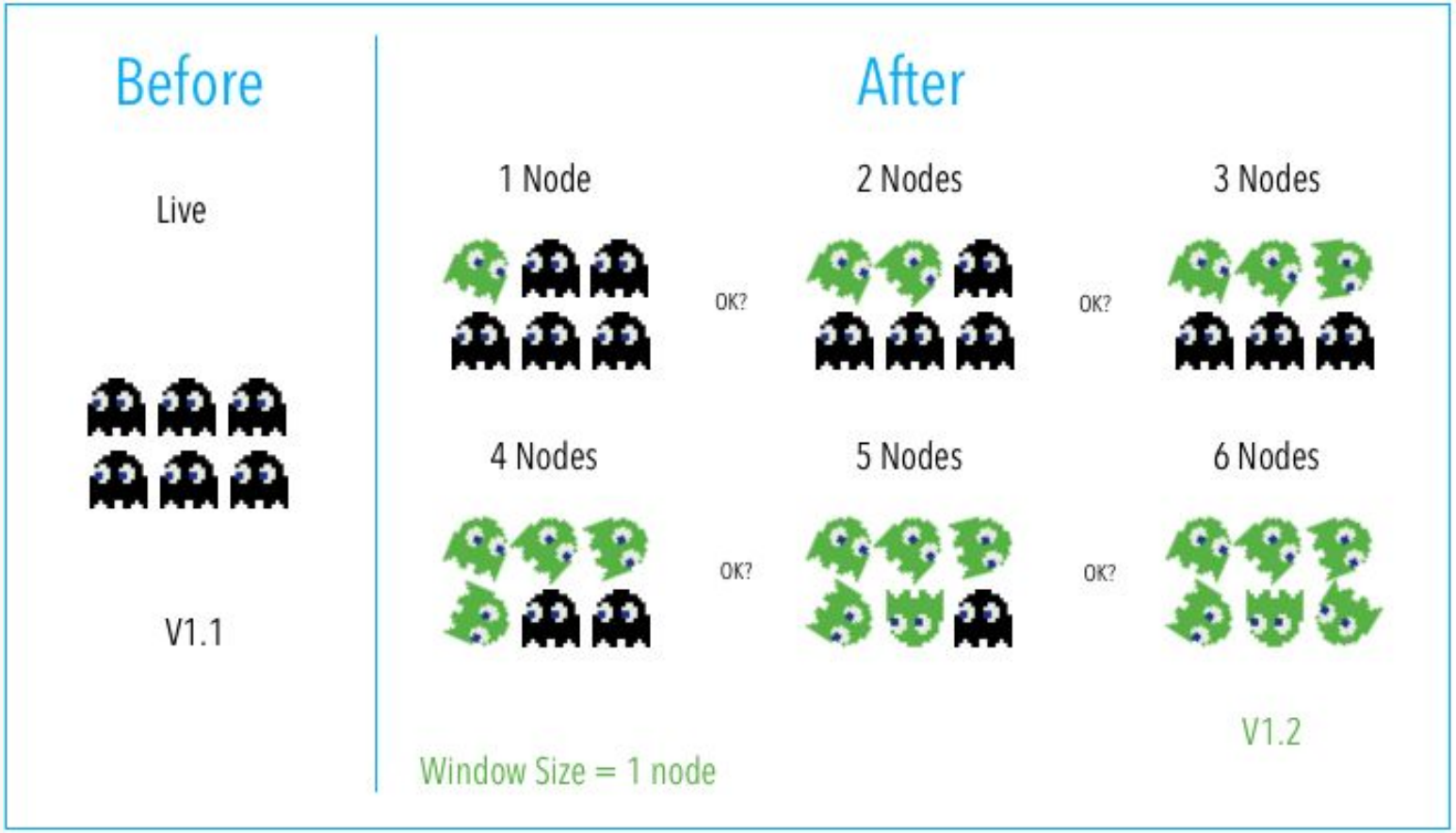
Model Rollout Strategies

- **Shadow**
  - New deployment shows existing system but not used for decision making
- Rolling
- Blue-Green
- Canary
- A/B Testing

Source

Model Rollout Strategies

- Shadow
- **Rolling**
  - Updates nodes in a target environment incrementally in batches with the new service version.
- Blue-Green
- Canary
- A/B Testing



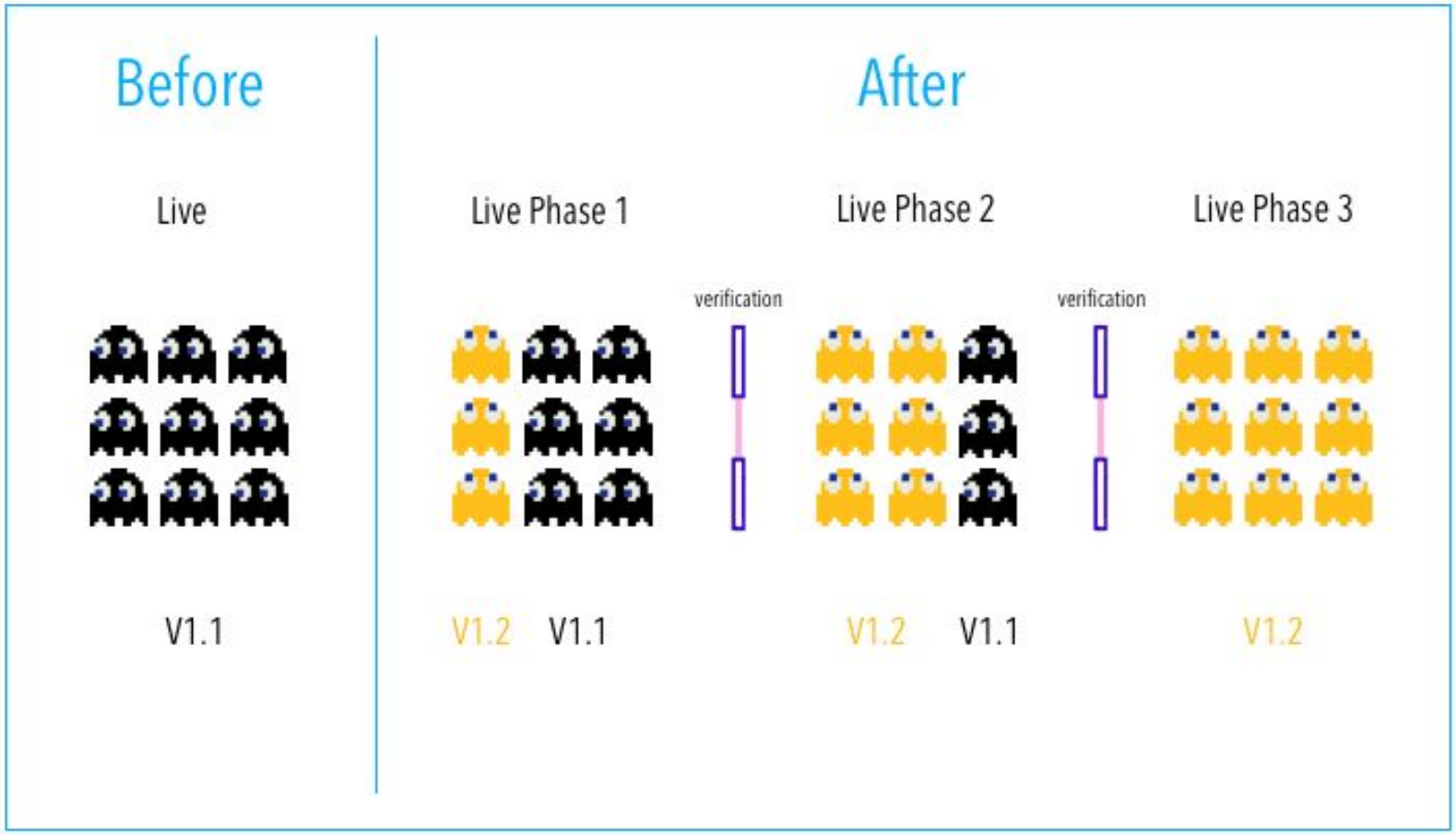
Source

Model Rollout Strategies

- Shadow
- Rolling
- **Blue–Green**
  - Utilizes two identical environments, a “blue” and a “green” environment with different versions of an application or service
- Canary
- A/B Testing

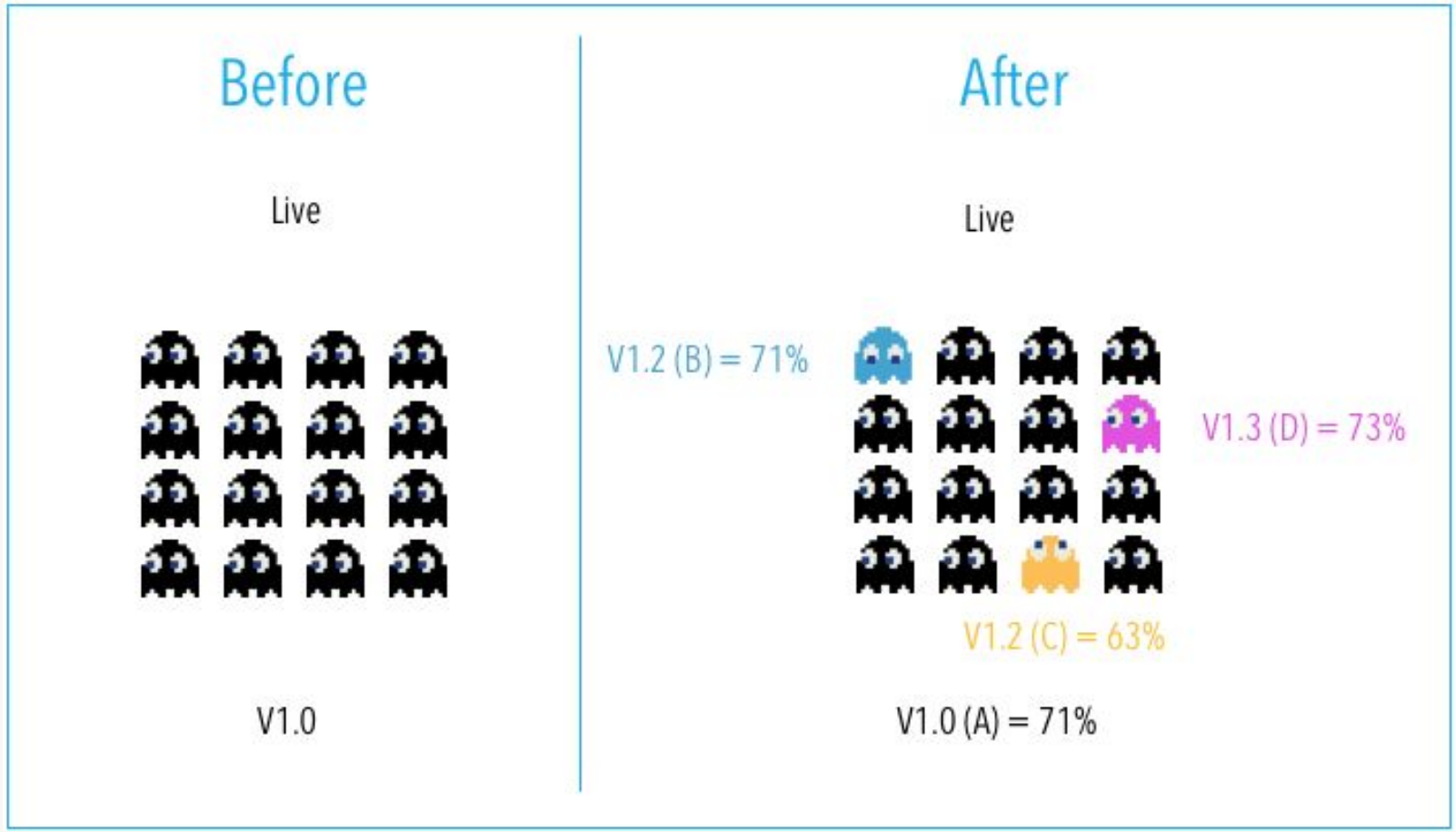
Model Rollout Strategies

- Shadow
- Rolling
- Blue-Green
- **Canary**
  - Releases an application or service incrementally to a subset of users
- A/B Testing



Model Rollout Strategies

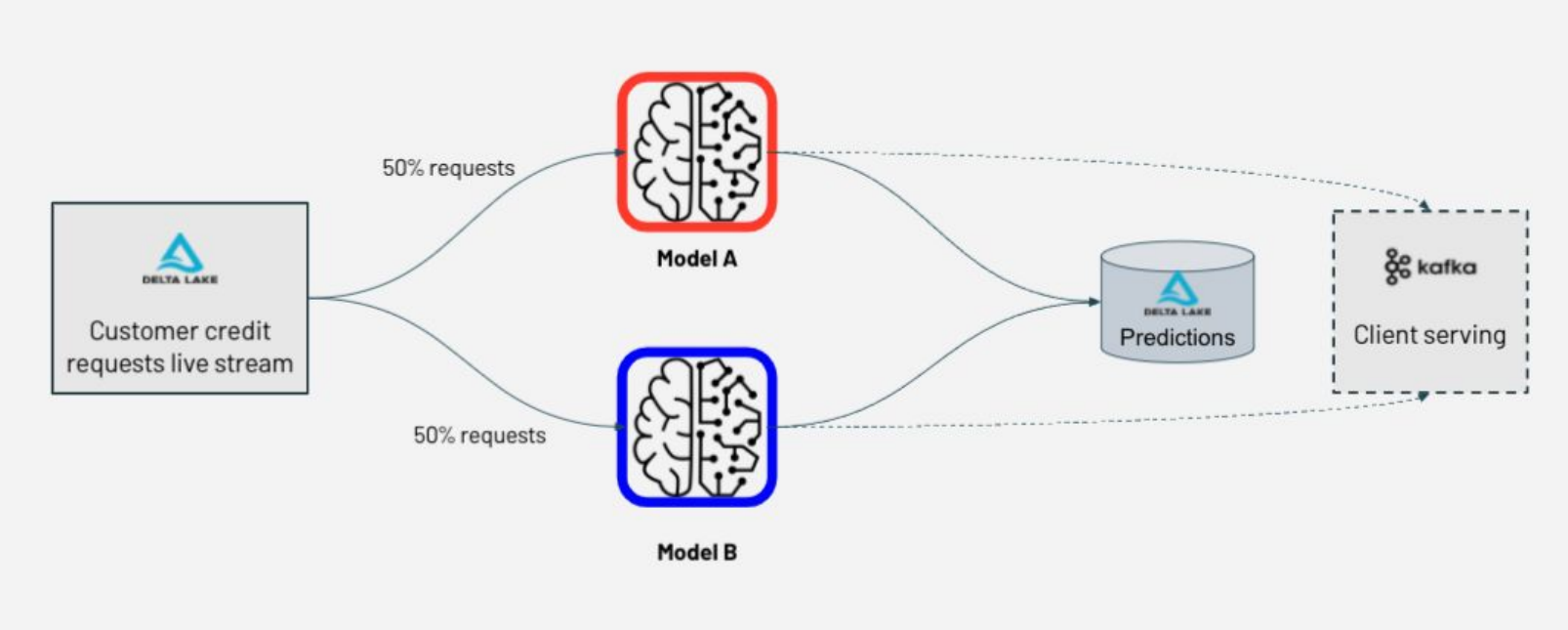
- Shadow
- Rolling
- Blue-Green
- Canary
- **A/B Testing**
  - Different versions of the same service run simultaneously in the same environment for a period of time.



A/B Testing Framework

A/B testing is essentially an experiment where two or more variants of a deployment are shown to users at random, and statistical analysis is used to determine which variation performs better for a given conversion goal.

- Collect data
- Identify goals
- Generate hypothesis
- Create variations
- Run experiment
- Analyze results



A/B Testing Metrics

Project	ML metric	Business Metric
Fraud Detection	Area Under PR, Area Under ROC, f1	Fraud Loss $, # Fraud Investigations
Churn Prediction	Area Under PR, Area Under ROC, f1	Recency of purchases Login events for high churn risk
Sales Forecasting	AIC, BIC, RMSE, et al.	Revenue
Sentiment Analysis	bleurt, bertscore	Number of users of tool, engagement rate
Ice Cream Coupons	MAE, MSE, RMSE	Revenue, Coupon usage

Source

Thank you!