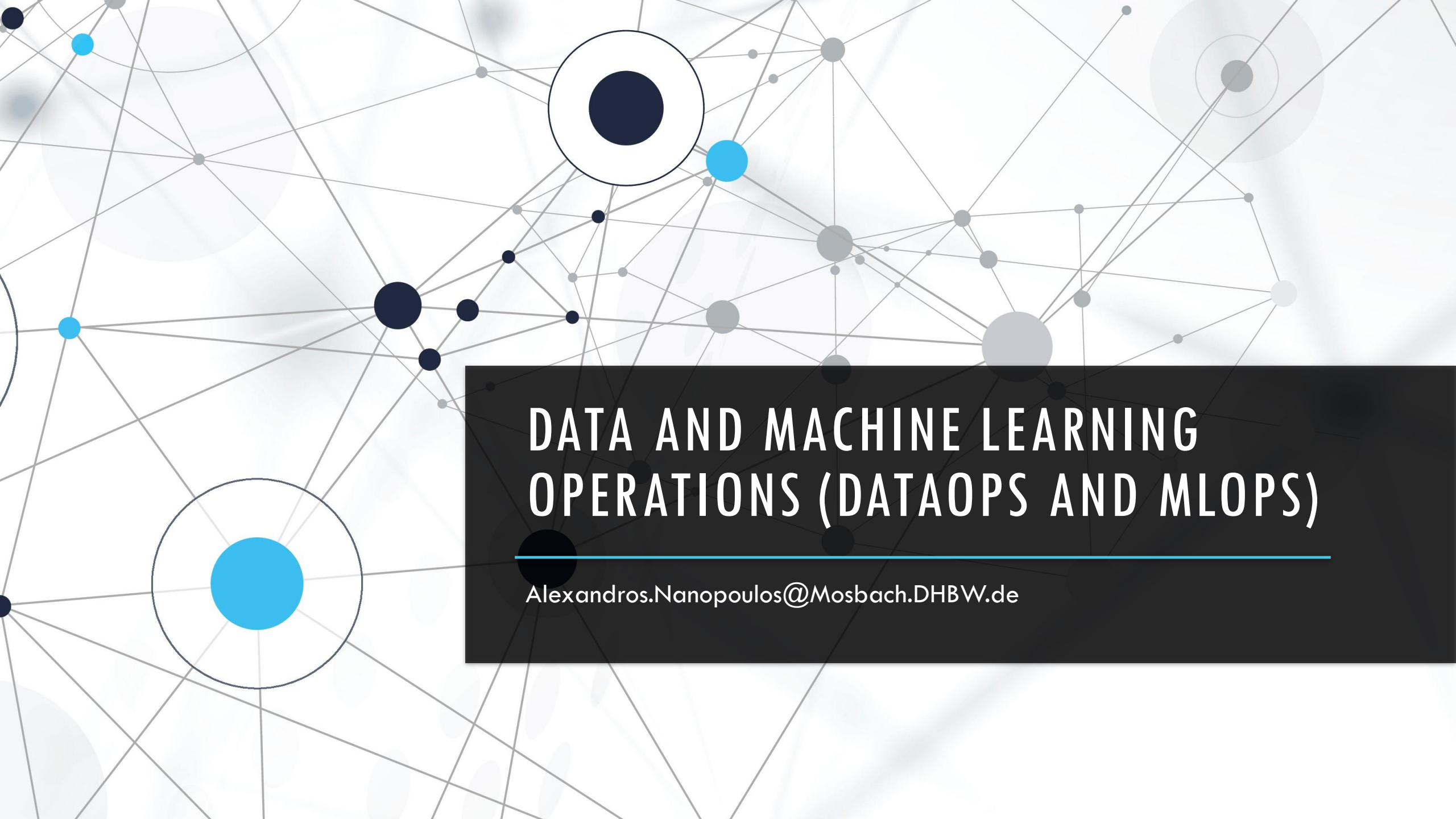


DATA AND MACHINE LEARNING OPERATIONS (DATAOPS AND MLOPS)

Alexandros.Nanopoulos@Mosbach.DHBW.de


IF YOU DEVELOPERS COULD GO AHEAD AND WRITE PRODUCTION READY CODE
YEAAAAAH... THAT WOULD BE GREAT


RELEASE READY FOR PRODUCTION?
MERGE INTO MASTER AND DEV

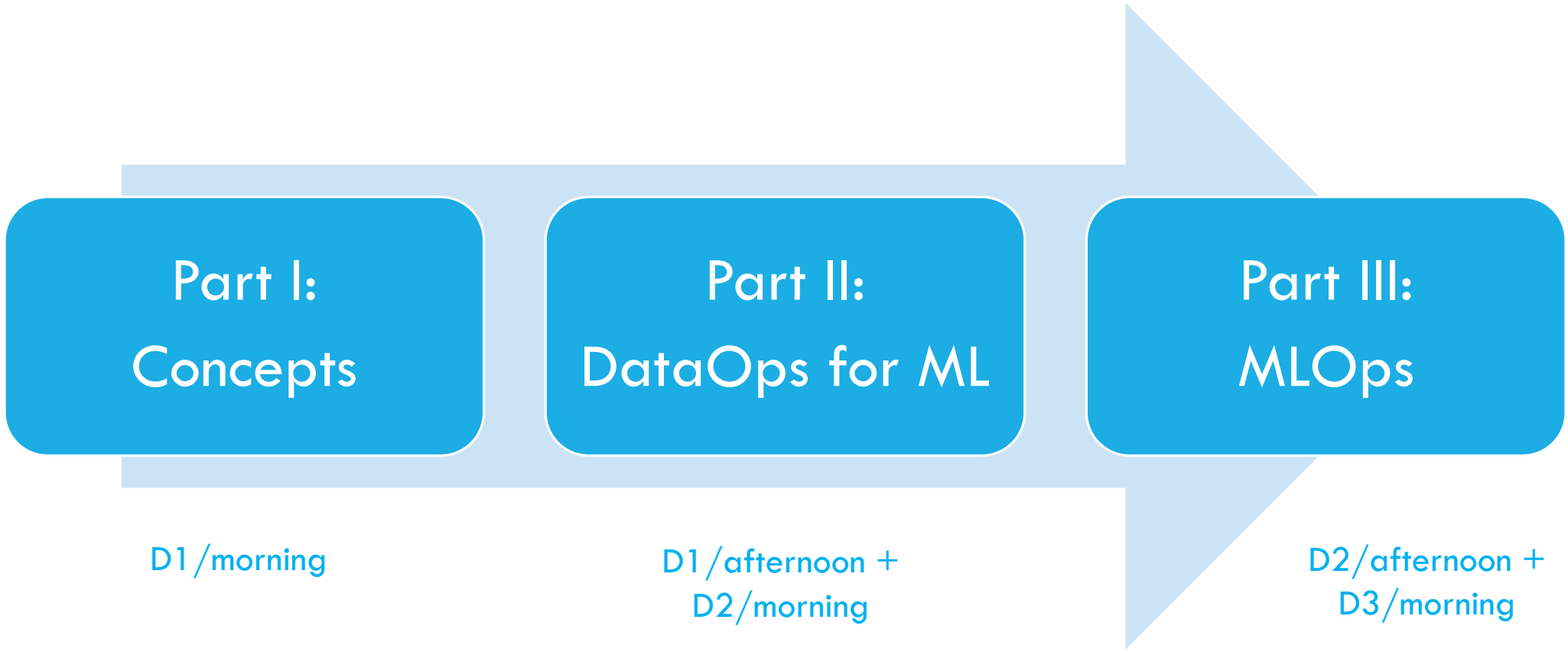

WE TOLD THEM WE WERE
PRODUCTION READY

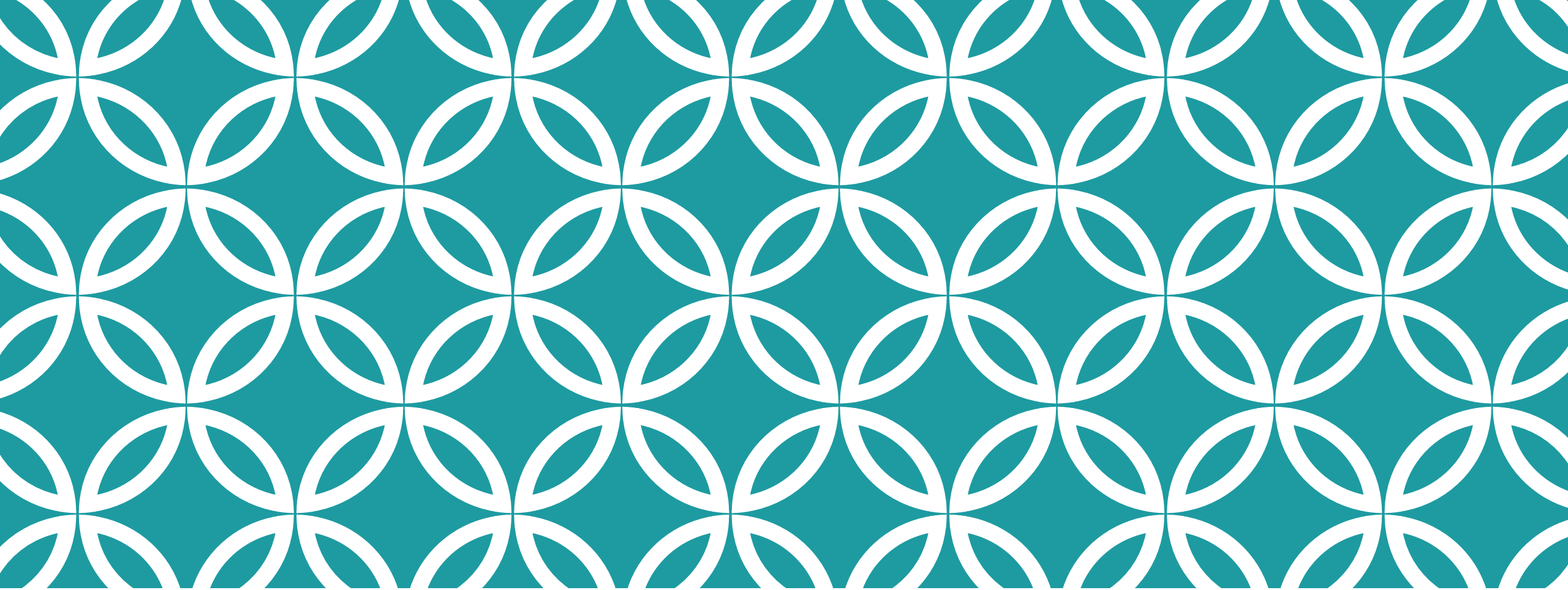

WORKED FINE IN DEV
OPS PROBLEM NOW


SAY "PRODUCTION READY?"
ONE MORE TIME


THERE IS NO
PRODUCTION READY
makeameme.org

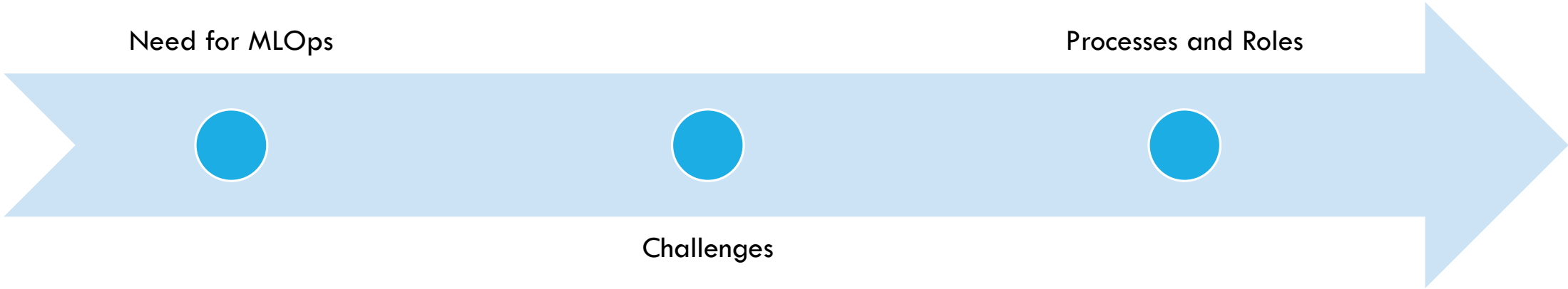
OUTLINE

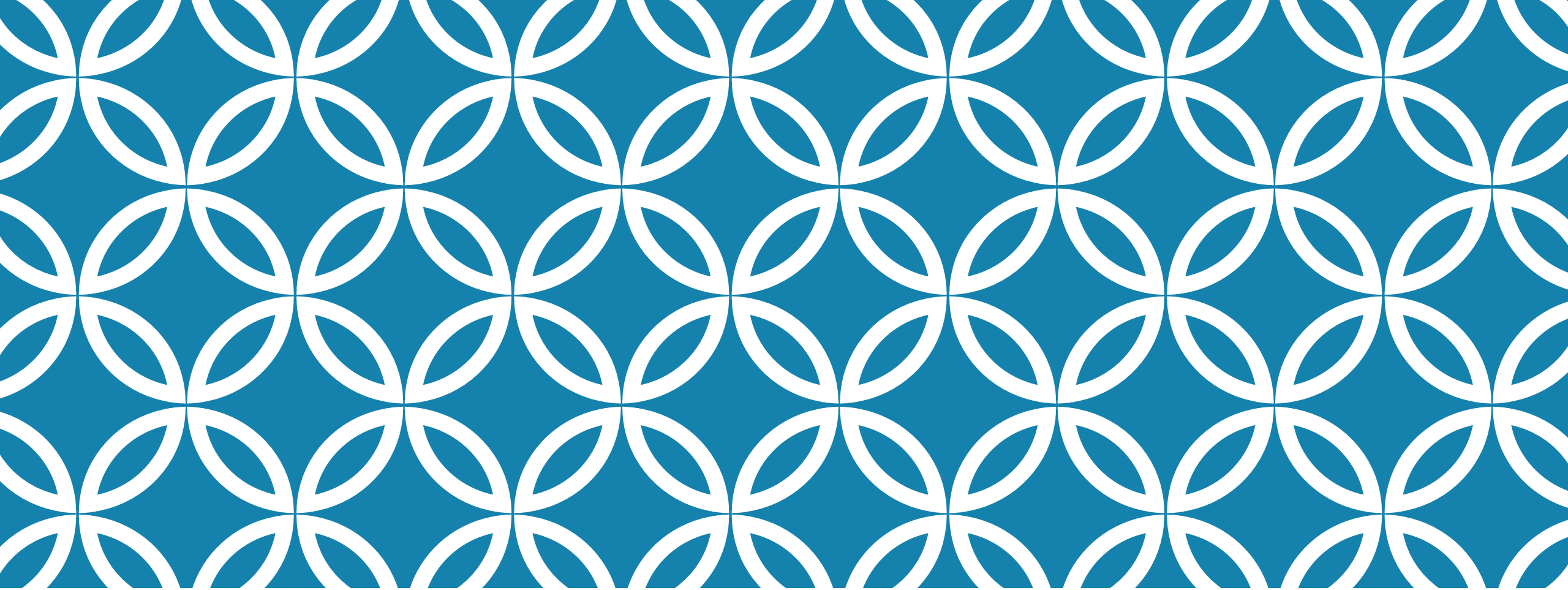

Part I:
Concepts
Part II:
DataOps for ML
Part III:
MLOps
D1/morning
D1/afternoon +
D2/morning
D2/afternoon +
D3/morning



PART I: CONCEPTS

PART I: CONCEPTS



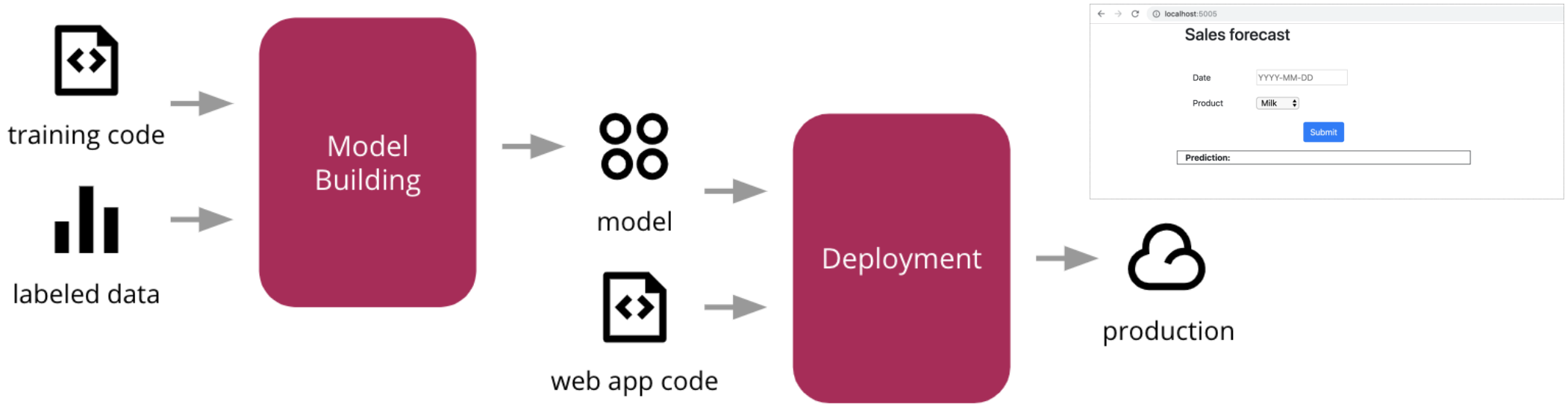


NEED FOR MLOPS

Part I: Concepts

NEED FOR MLOPS

"Push ML to production": A simplistic view

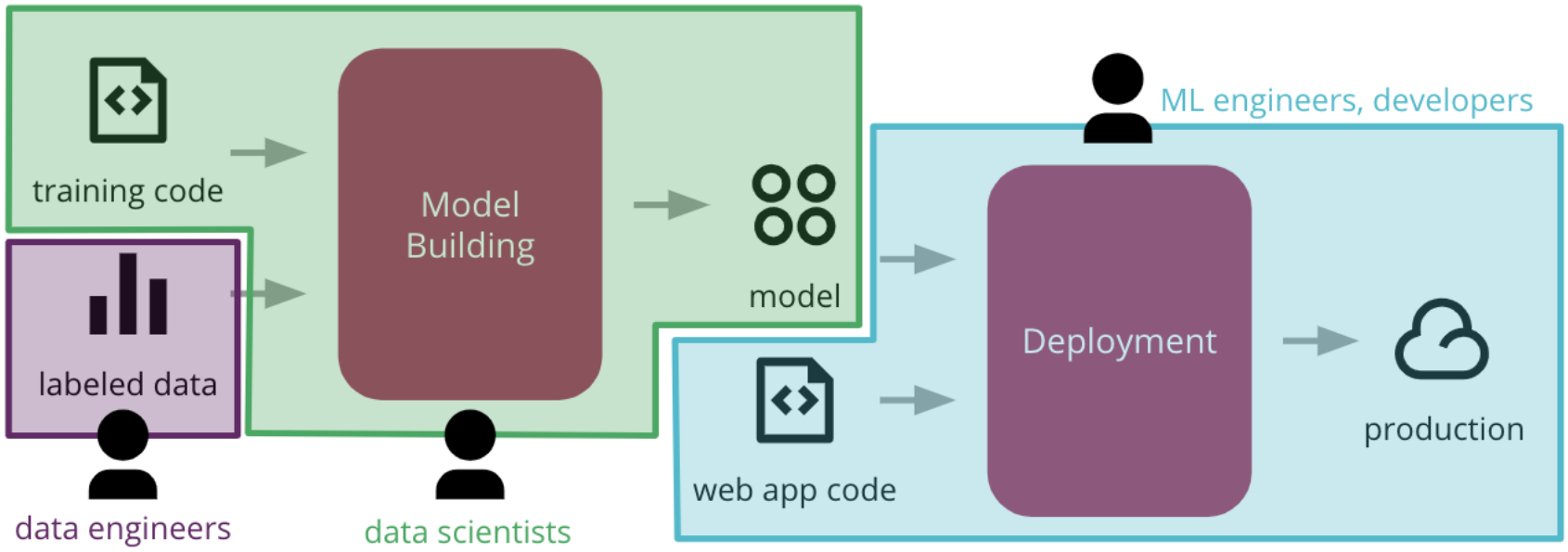


https://martinfowler.com/articles/cd4ml.html

NEED FOR MLOPS

organizational structure and responsibilities,
silos, "throw over the wall“

different tools/workflows,
no reproducible/auditable process,
no end-to-end automation



unclear operations: what kind of issues and whose responsibility?

NEED FOR MLOPS

DevOps mentality: releasing software in an agile manner while **constantly** improving the quality of both business outcomes and the software itself

Deep **knowledge** of ingredients and years of practical **experience** creating delicious meals

Can make meals in an **industrialized** and **repeatable** manner that enables **profitability**

NEED FOR MLOPS

	ML Research	MLOps
Requirements	State-of-the-art model performance on benchmark datasets	Different stakeholders have different requirements
Computational priority	Model development	Deployment, fast inference, and maintenance
Data	Static (and clean)	Constantly shifting (and messy)
Infrastracture	Laptop/workstation	Cloud/On-prem/Hybrid
Fairness/Interpretability	Often not a focus	Must be considered

NEED FOR MLOPS

Models are not making it into production; if they do, they are brittle and fall apart when faced with the complexities of the actual world.

of all intelligent projects fail and deliver erroneous outcomes due to biased data.

Gartner

companies report significant financial gains from developing AI strategy.

BCG

data science projects sink into oblivion at the ideation stage.

VentureBeat

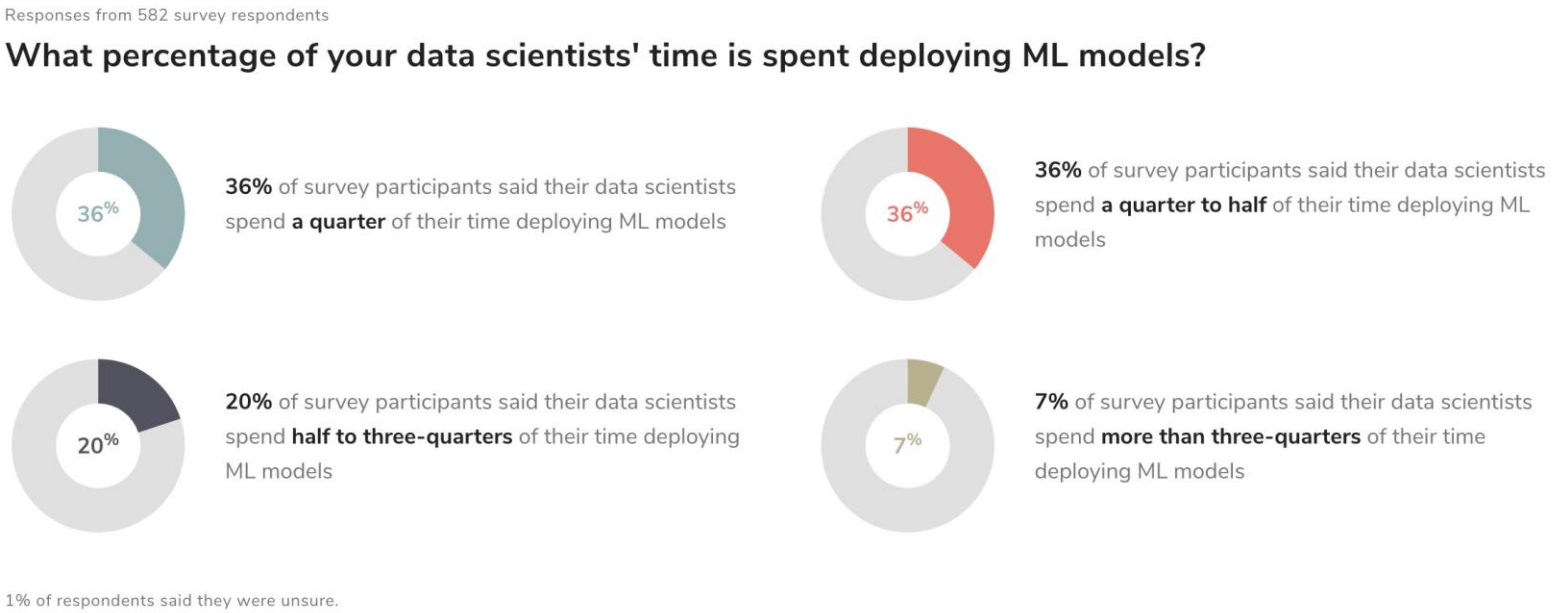
More failure statistics

NEED FOR MLOPS

22 percent of companies that use machine learning have successfully deployed an ML model into production.

72% of organizations that began AI pilots before 2019 have not been able to deploy even a single application in production.

Algorithmia "2020 State of Enterprise Machine Learning"



manual and one-off work (e.g., script-based data pipelines)

no reproducible components (e.g., boilerplate aspects)

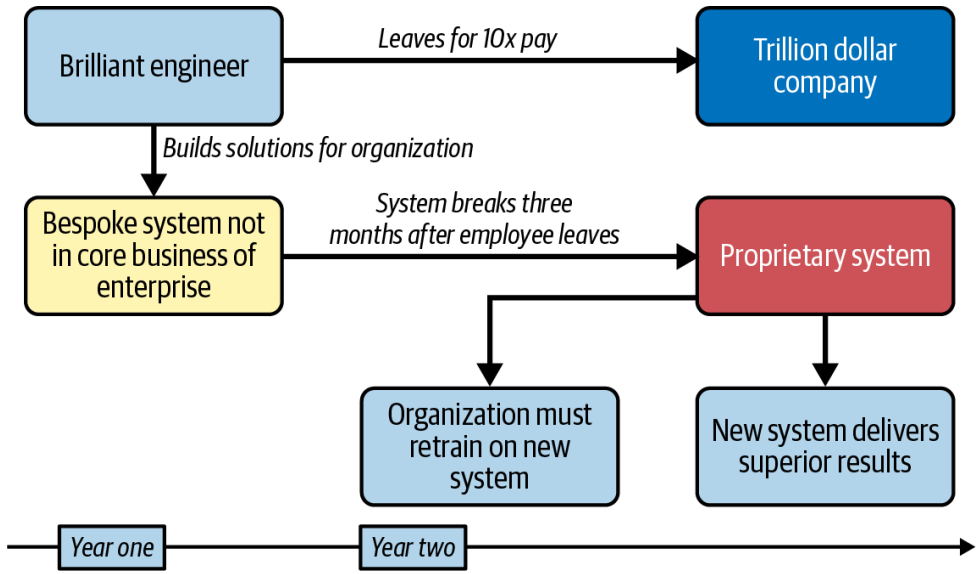
hard handoffs between data scientists and IT

NEED FOR MLOPS

ML is **accelerator** (and not creator) of business value for organizations in other industries

Unbounded risk of (bespoke) ML solutions: No delivery or lousy solution that worsens business value

ML adoption with **calculated risk**: limit downsides of technology change management



NEED FOR MLOPS

No silver bullets
Better data (ingredients) or a specific deep learning framework (oven)?

Business and production-first mindset + comprehensive strategy

"MLOps supports ML development like DevOps supports software development"

NEED FOR MLOPS

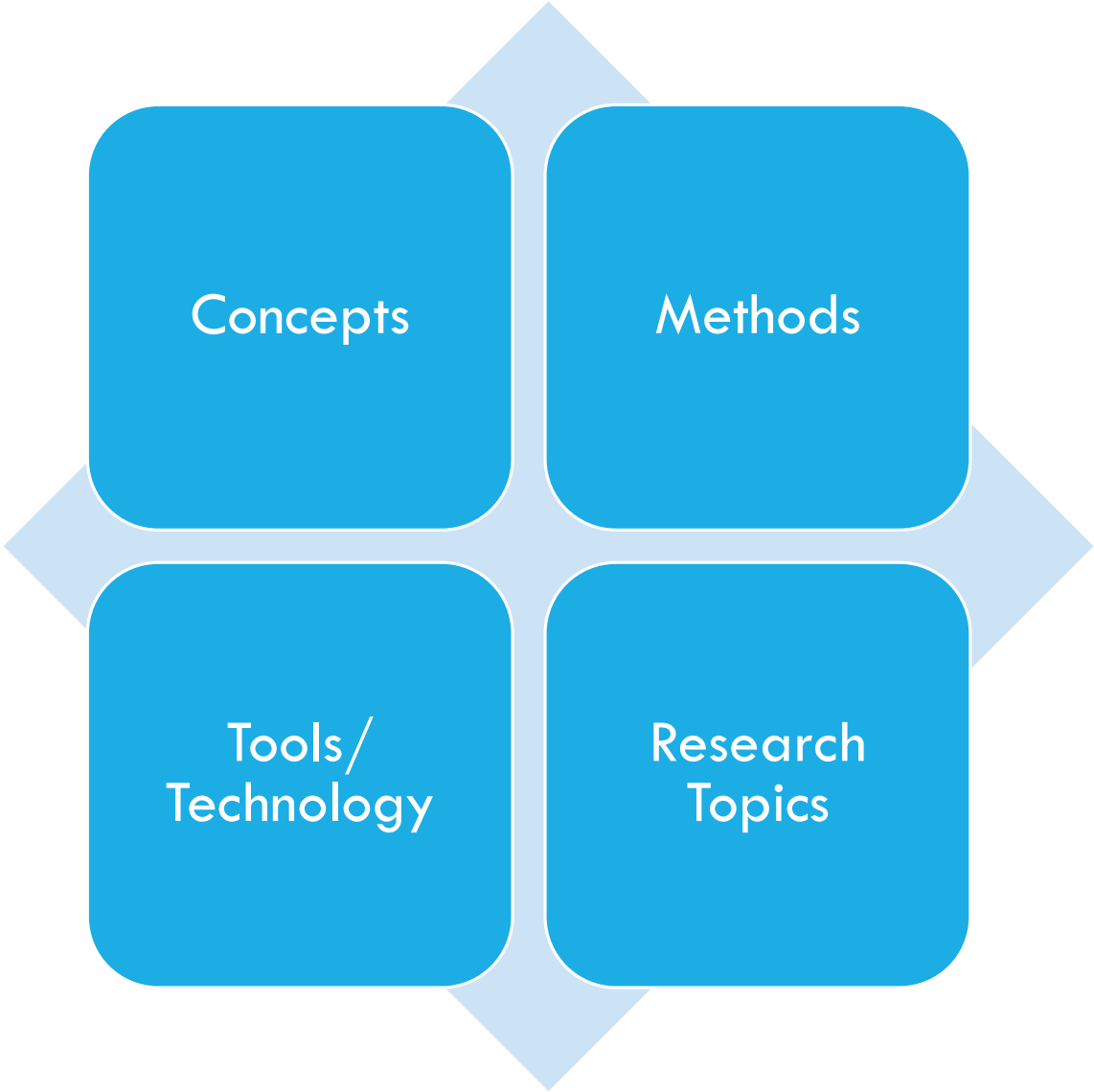
MLOps = Machine Learning (ML) + Operations (Ops): processes and practices for designing, building, enabling, and supporting the efficient deployment of ML models in production, to continuously improve business activity:

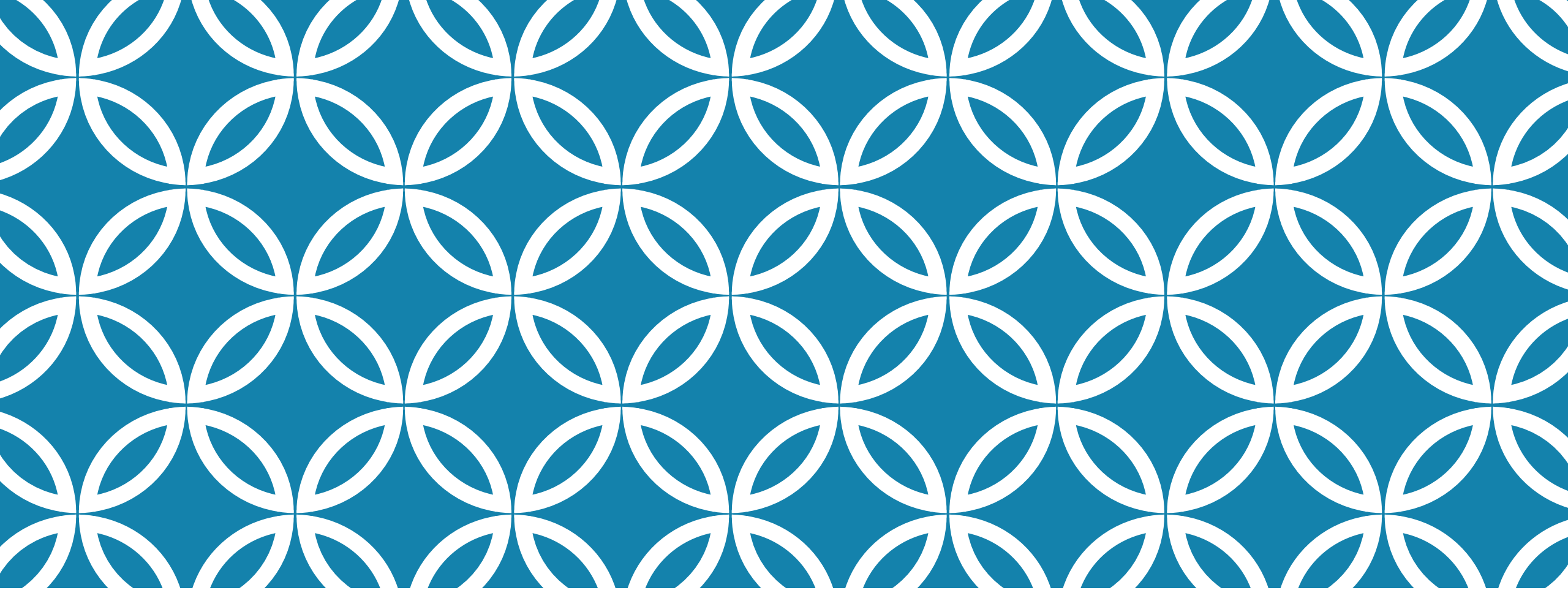
- Model deployment and development time
- Collaboration between different teams
- Operational excellence of ML systems
- Data governance
- Enhancing the ROI

Gartner: "MLOps aims to standardize the deployment and management of ML models alongside the operationalization of the ML pipeline. It supports the release, activation, monitoring, performance tracking, management, reuse, maintenance, and governance of ML artifacts".

NEED FOR MLOPS

Dimensions of MLOps

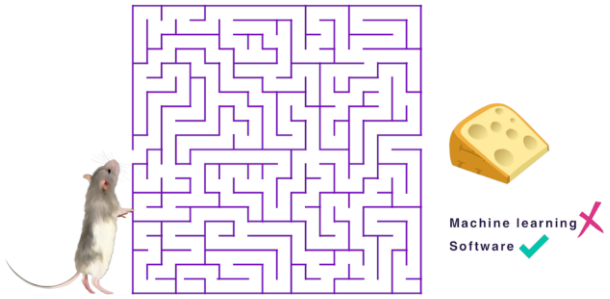


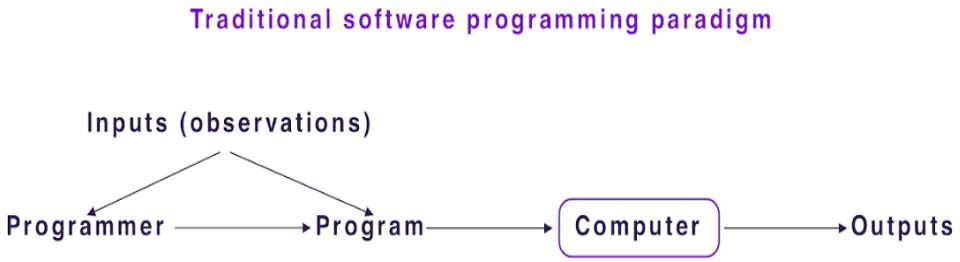


CHALLENGES

Part I: Concepts

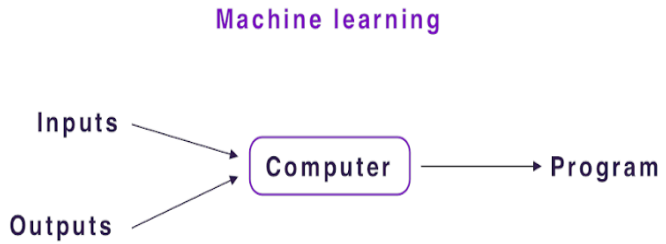
CHALLENGES





clear mathematical specification

more data doesn't improve the quality of the algorithm



no clear mathematical specification

best solved with (lots of) clean, structured and well annotated data

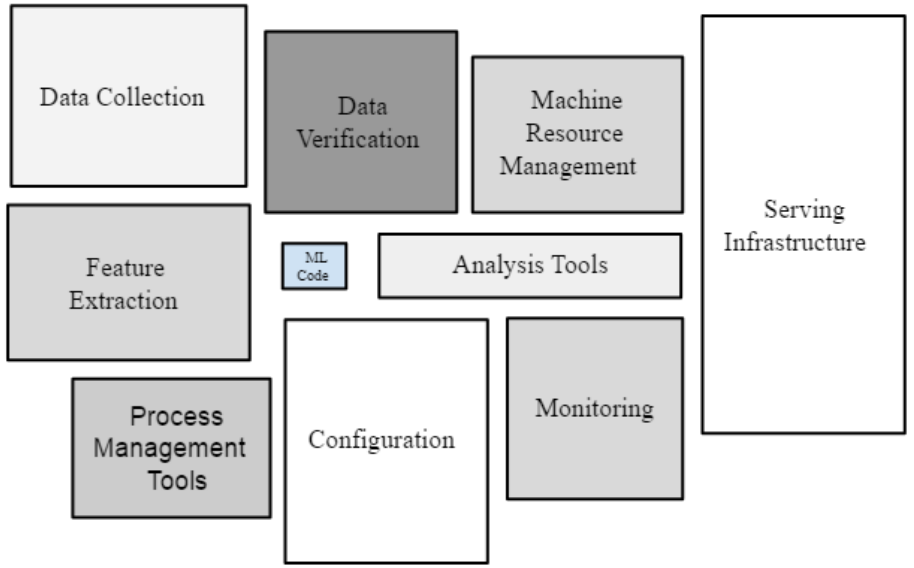
CHALLENGES

- Benefits **and problems** of ML stem from the same fact: "programs" (i.e., models) are generated automatically from labeled data, not by developer's code
- ML "programs" are difficult to test, debug, maintain, and understand

- Testing software:
  Unit test for each piece of code
- Testing ML:
  - With each input, a new program
  - A unit test for every single input?
  - Not even clear what we should test

CHALLENGES

- ML is not just different, it is also not sufficient: you still need “standard” software
- ML code accounts for only 5% of ML systems
- You’ve just trained a model? Congratulations! You are only 5% of the way to creating a real-world product capable of providing value to customers



CHALLENGES

Technical debt: long-term costs incurred by **moving quickly** in software engineering

Technical dept in ML: **all** of the maintenance problems of traditional code **plus** ML-specific issues

Hidden debt at the *system level*

ML developers as persons with a high interest credit card (and ignorant of how 20% compound interest works)

CHALLENGES

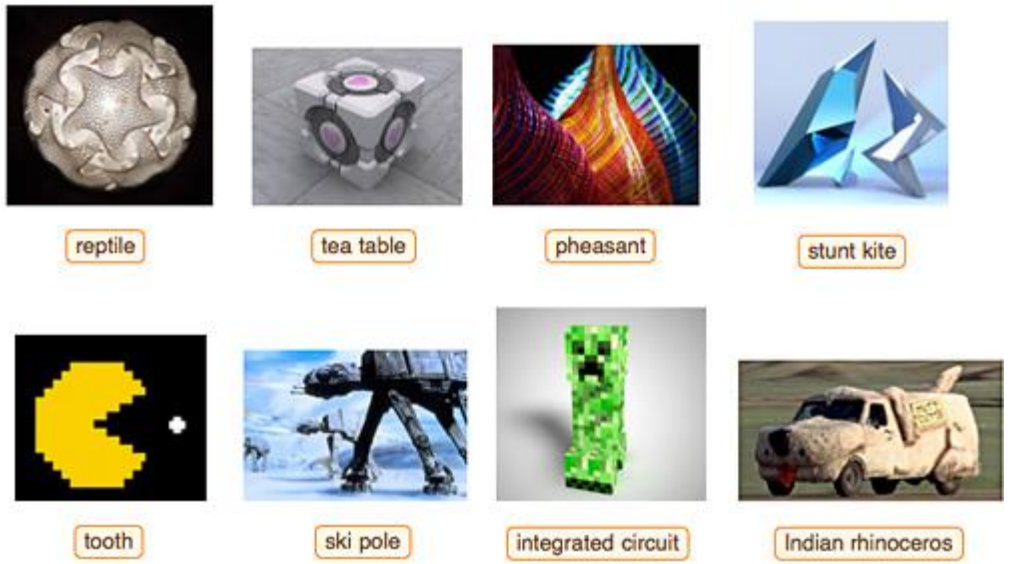
ML models often don't make it into production, and if they do, they break because they fail to adapt to **changes** in the environment

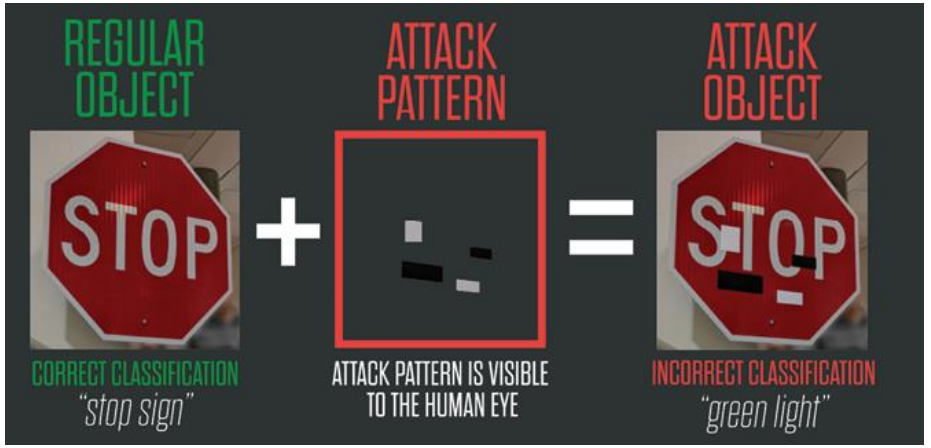
The failure modes could either be:

- System failure: One that breaks down the production system due to errors such as slow loading or scoring times, exception errors, etc.
- Statistical failure: Or "**silent**" failure where the model consistently outputs wrong predictions

CHALLENGES

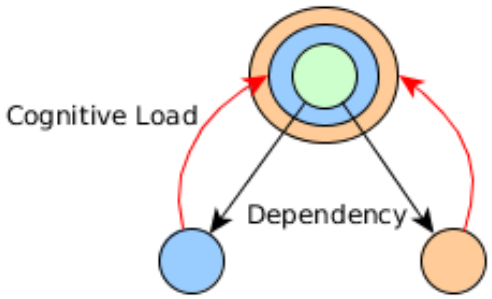
Silent failures…



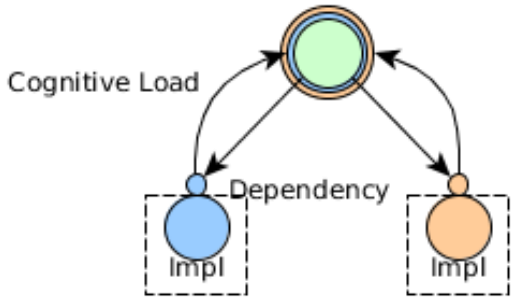


CHALLENGES

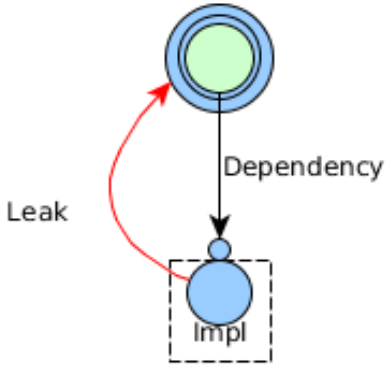
ML forms leaky abstractions and weak contracts (compared to "standard" software)



(a) High complexity and cognitive load when there's no abstraction.



(b) Abstractions reduce cognitive load by hiding details..



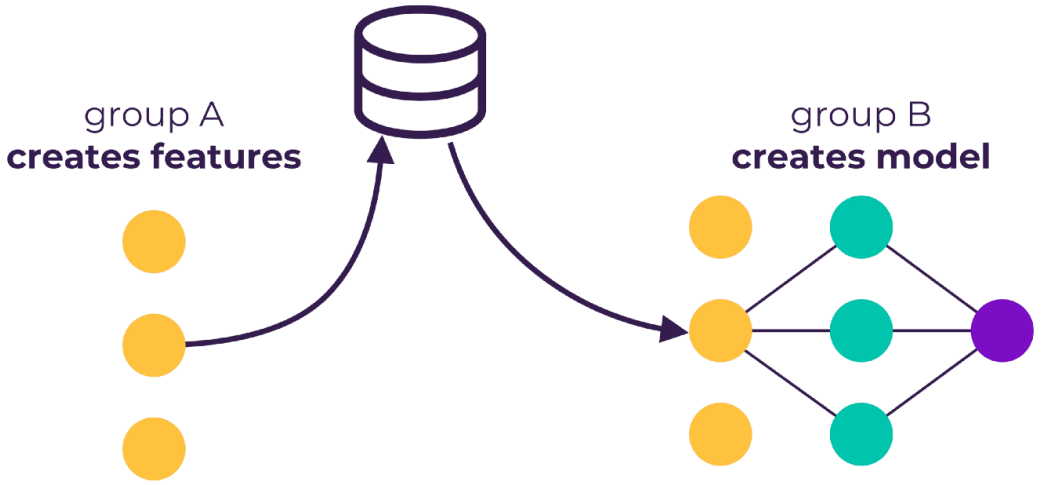
(c) But abstractions are not perfect and some details may leak.

Let's examine concrete examples that occur in *production*

CHALLENGES

ML forms leaky abstractions

- Customer churn model: a column for gender of customers with two values (male and female)
- Frontend adds “Transgender”, “Other”, and “Prefer not to say” to the dropdown list
- “Hey, our model didn’t crush, because our code handles unseen values. That’s good!”

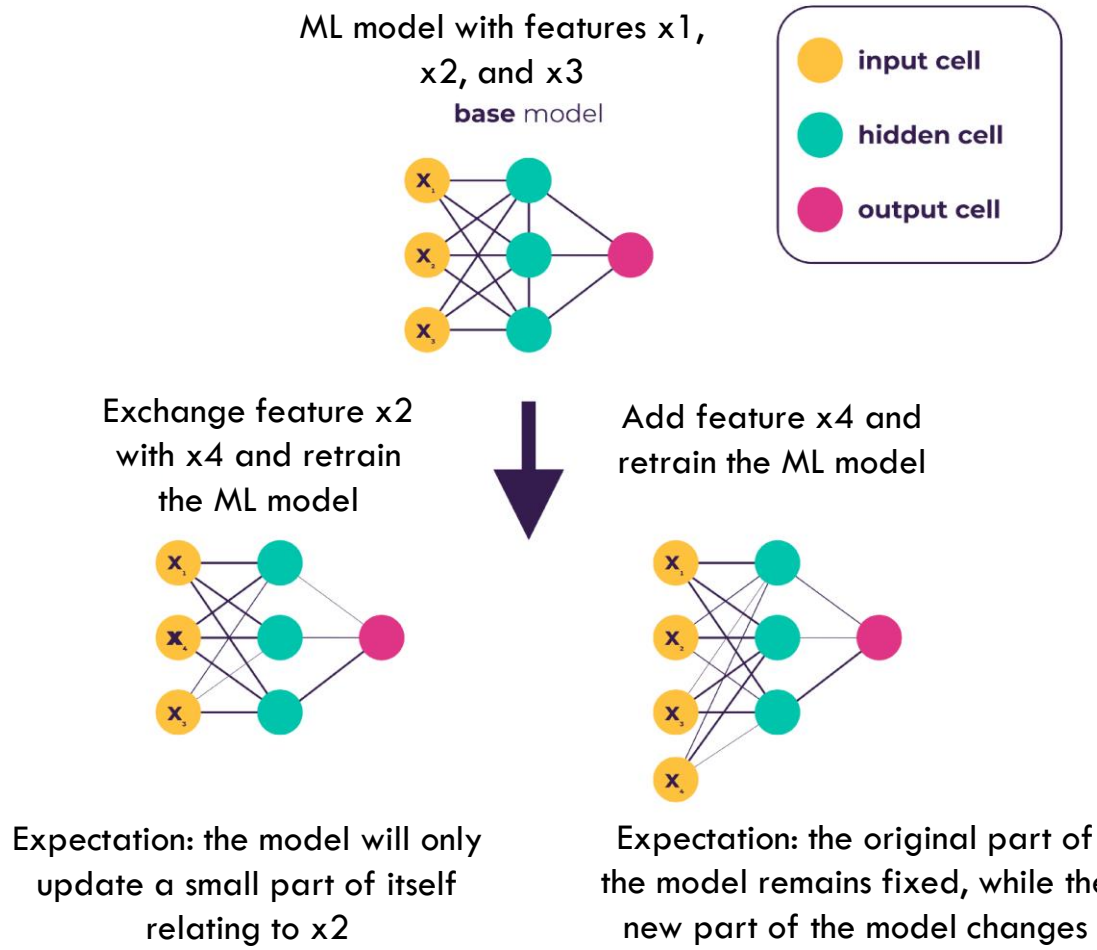


Different groups are responsible for feature creation and model creation; such updates to external systems can happen at any time.

CHALLENGES

ML forms leaky abstractions

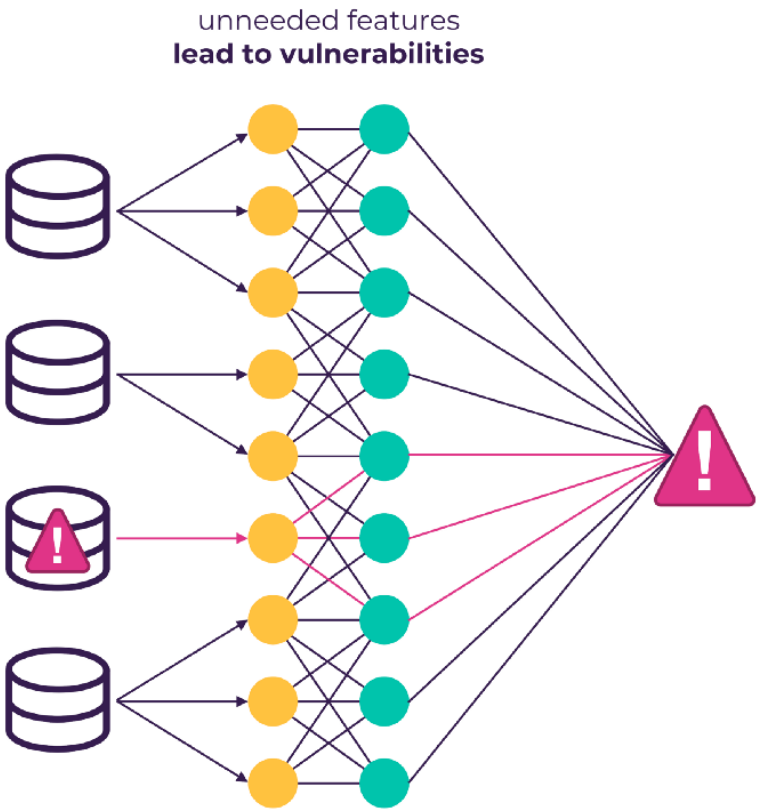
- Better estimate of location from web cookies: Brilliant! Surely, this will improve our model that uses location to predict a propensity score
- Overwrite the previous feature and re-train the model (we can even add the new feature in parallel)
- "The model will stay more or less the same; only the part that was related to the old feature may change a bit"



CHALLENGES

ML forms leaky abstractions

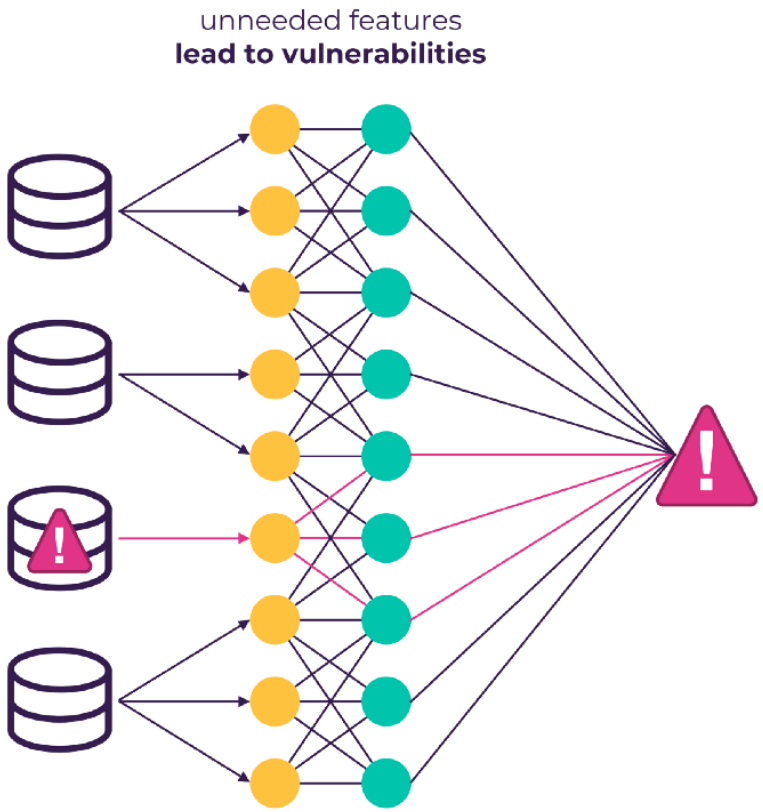
- In first version of a music recommender we found that user's device type is highly predictive feature
- Later, we realize that the listening history and the likes are better features, whereas device type is not that important anymore
- "Just in case, we should keep the device type as feature; this can't hurt, right?"



CHALLENGES

ML forms leaky abstractions

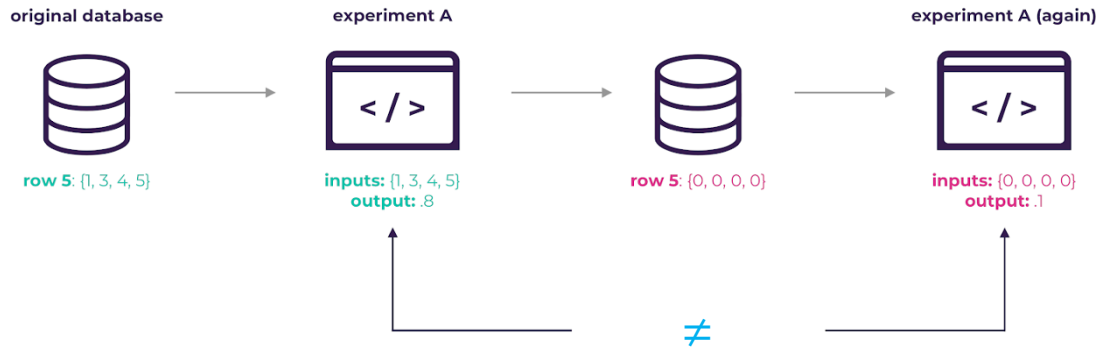
- In a rush to improve sales prediction, we add various statistics (mean, mode, IQR, skew, kurtosis) of past sales per week day
- We wrote *custom code* to calculate these statistics
- Later, we find out that the mean is adequate
- "But the rest statistics still might bring 0.1% improvement, so they can stay"



CHALLENGES

ML forms leaky abstractions

- Model: probability of churn in next 2 months
- For a selected test example, the model predicts 0.8. The features of this customer were added to a nice story/presentation
- Two months later, you pick up the project to improve upon
- You run the same ML model and get an output of 0.1on the same test example
- “I checked twice, it’s clear that I use the same code ☹”



CHALLENGES

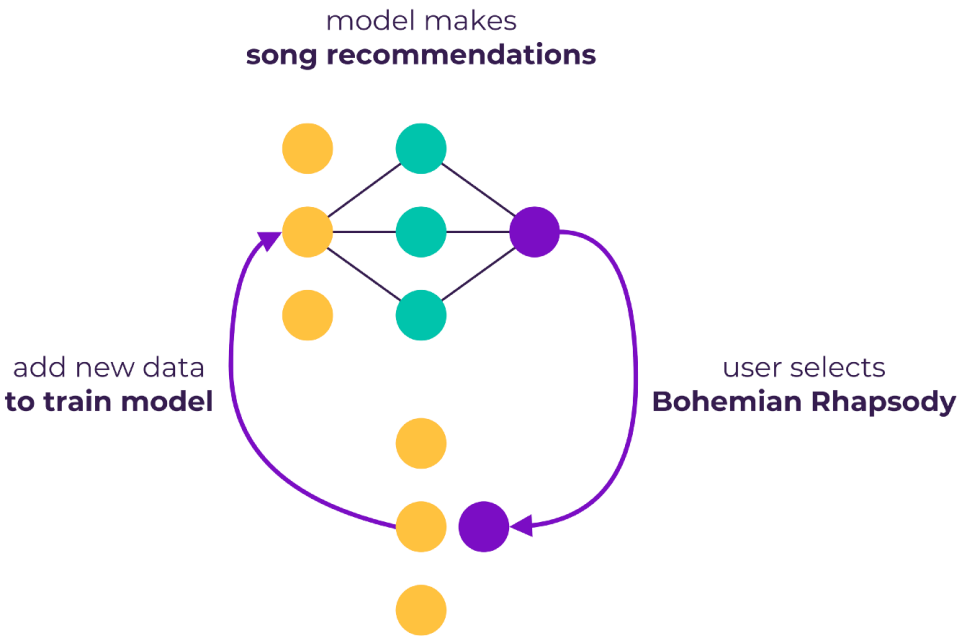
ML forms leaky abstractions

- Success story: one ML team developed a baseline model (Ma) to predict sales under “normal conditions” => Its predictions are made accessible, either online (API) or by writing to a database (batch)
- Our ML teams discovers this asset and (is forced to) consume it in our Mb (downstream) models to predict: out-of-stock, the impact of discounts, in-store personnel planning, etc.
- What we suspect: Ma overestimates **systematically** by 1M€ per week
- “Somehow our Mb model seems able to handle this. Isn’t ML great?”
- At some point, Ma-team realized themselves, corrected and retrains this model. We think: “Finally! All is fine now.”

CHALLENGES

ML forms leaky abstractions

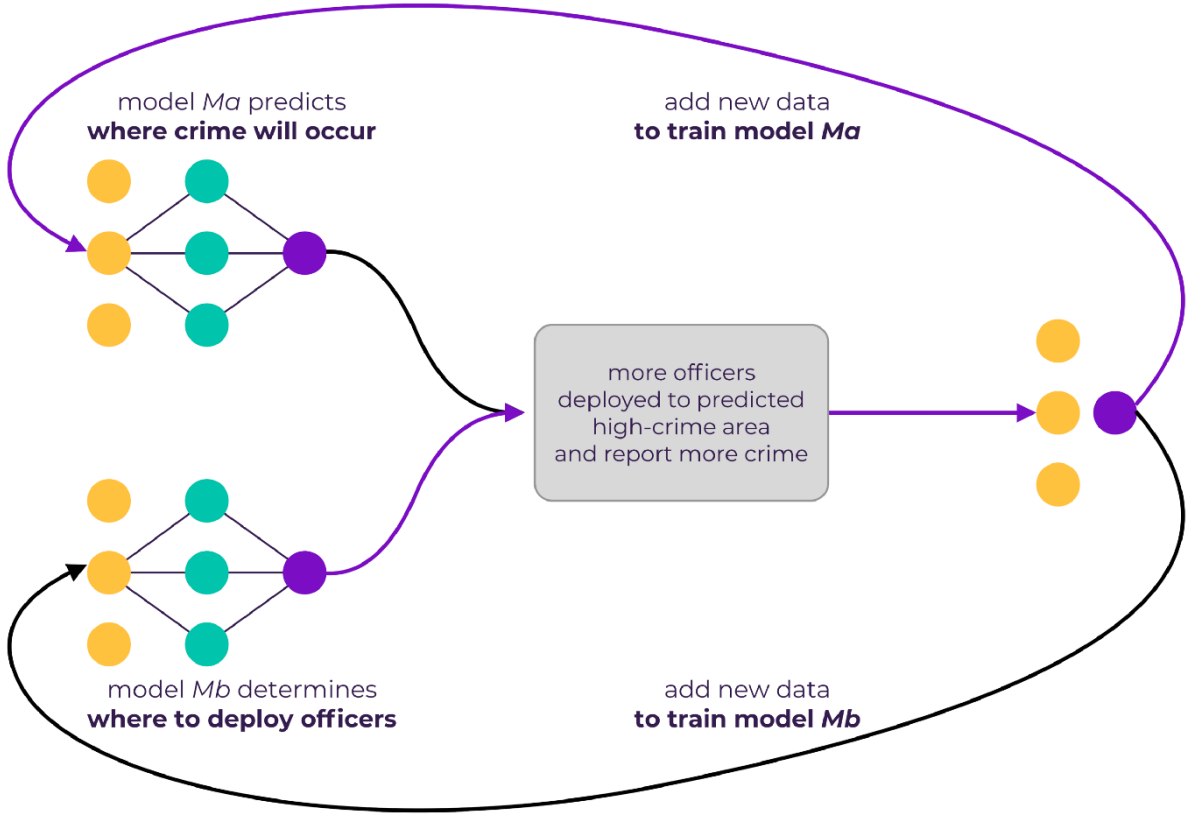
- Our cool song-recommender on a music streaming service
- During user acceptance tests: model presents 5 songs and the user chooses one — Bohemian Rhapsody
- In truth, her ideal song — Kraftwerk's Autobahn — was not recommended
- "Such kind of prediction errors are always expected in ML-systems, right? Hey, there is a positive side-effect of it: the users discover new music and may even evolve their tastes"



CHALLENGES

ML forms leaky abstractions

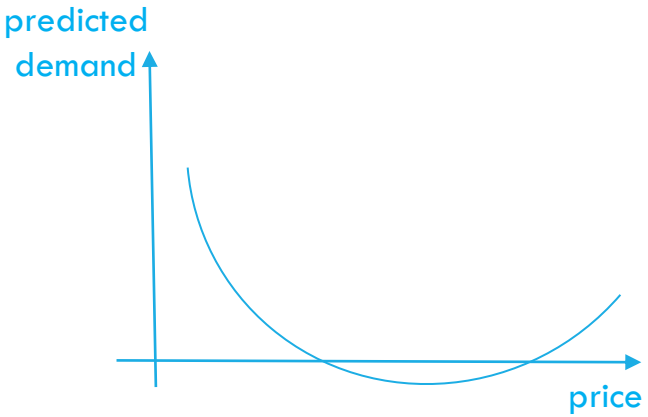
- Critical ML algorithm (Ma): predicts where crimes will occur
- Here comes another model (Mb): depending on Ma, where to *optimally* deploy police officers
- Mb tends to deploys more police officers to areas identified by Ma
- "Success! More crimes uncovered there where more officers were deployed"



CHALLENGES

ML forms leaky abstractions

- XGBoost regression model for demand prediction: dozens of well engineered features, impressive RMSE
- Business wants the isolated effect of price => we develop a script that automatically generates plots for some selected items and emails the pdf
- Reply: they ask if we selected only Veblen goods for good reasons
- We think: “Veblen-what? These business people only use buzzwords and have no clue about data science”



CHALLENGES

ML forms leaky abstractions

- Binary classifier for promotional offers: outputs prediction of some level of confidence
- We define a threshold to convert predicted probabilities to class labels
- We tuned the threshold during ML development with an evaluation set against our business (monetary) KPI
- "Of course, the model will have to be re-trained from time to time in production"

```
X_train, X_test, y_train, y_test = train_test_split(X,
y, test_size=.2, random_state=42)

svc = SVC(max_iter=10000, probability=True)

preds_svc = svc
                .fit(X_train,y_train)
                .predict(X_test)

probs_svc = svc.decision_function(X_test)

probs_svc = (probs_svc - probs_svc.min()) /
(probs_svc.max() - probs_svc.min())

y_pred = (probs_svc >= threshold).astype(int)
```

CHALLENGES

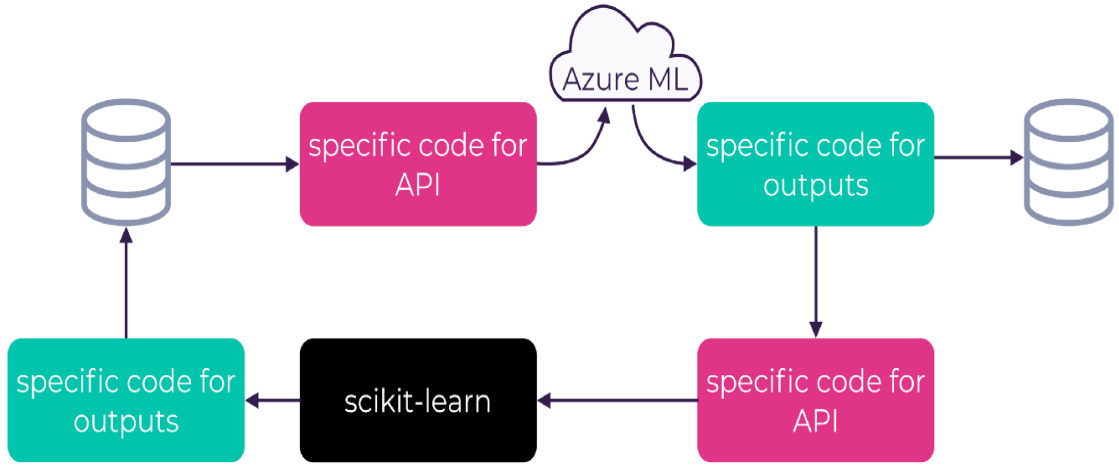
ML forms leaky abstractions

- Model estimates a patient's hospital bill; patients request an estimation at the point of admission
- Its key feature is the **estimated** *hospitalization_days*, provided by the admitting doctor via an app
- Generated values of *hospitalization_days* are stored in the production database, managed by IT
- After a major problem in the app, IT decided to **backfill** the hospitalization_days column for the last 2 years with the **actual** number of days hospitalized
- "Great, otherwise missing values would ruin training!"

CHALLENGES

ML forms leaky abstractions

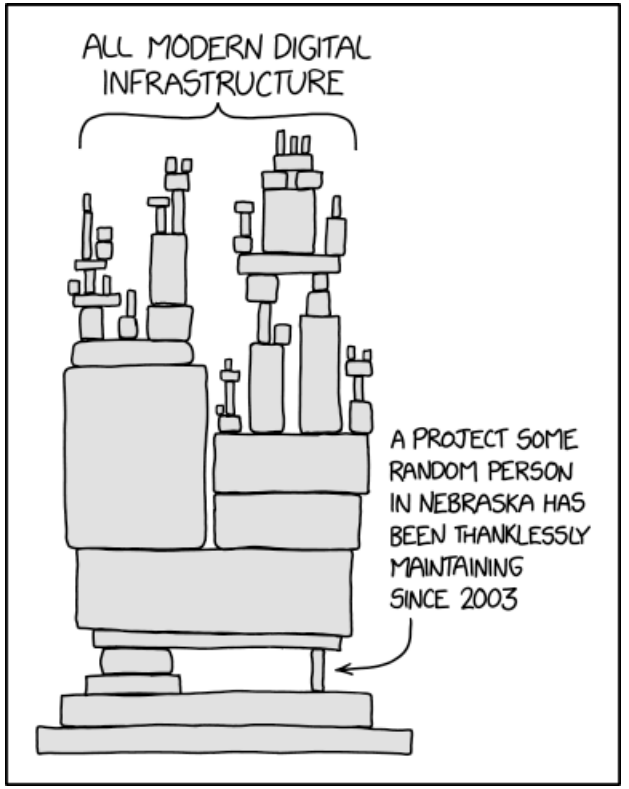
- A prototype for a new NLP use-case: Of course we use general purpose packages to speed up the development.
- Some **"glue code"** emerges (e.g., transform input and output data into the right format, use package-specific API to train and save the model, etc.)
- "What else should we've done? Of course, we tested our glue code, too, so it can make it into production"



CHALLENGES

ML forms leaky abstractions

- The new release of Spark fixes a critical pain point we had lately
- We start using it ASAP in our dev environment
- “We continue with the new release in dev, since the infra team will get convinced and update prod soon”



CHALLENGES

ML forms leaky abstractions

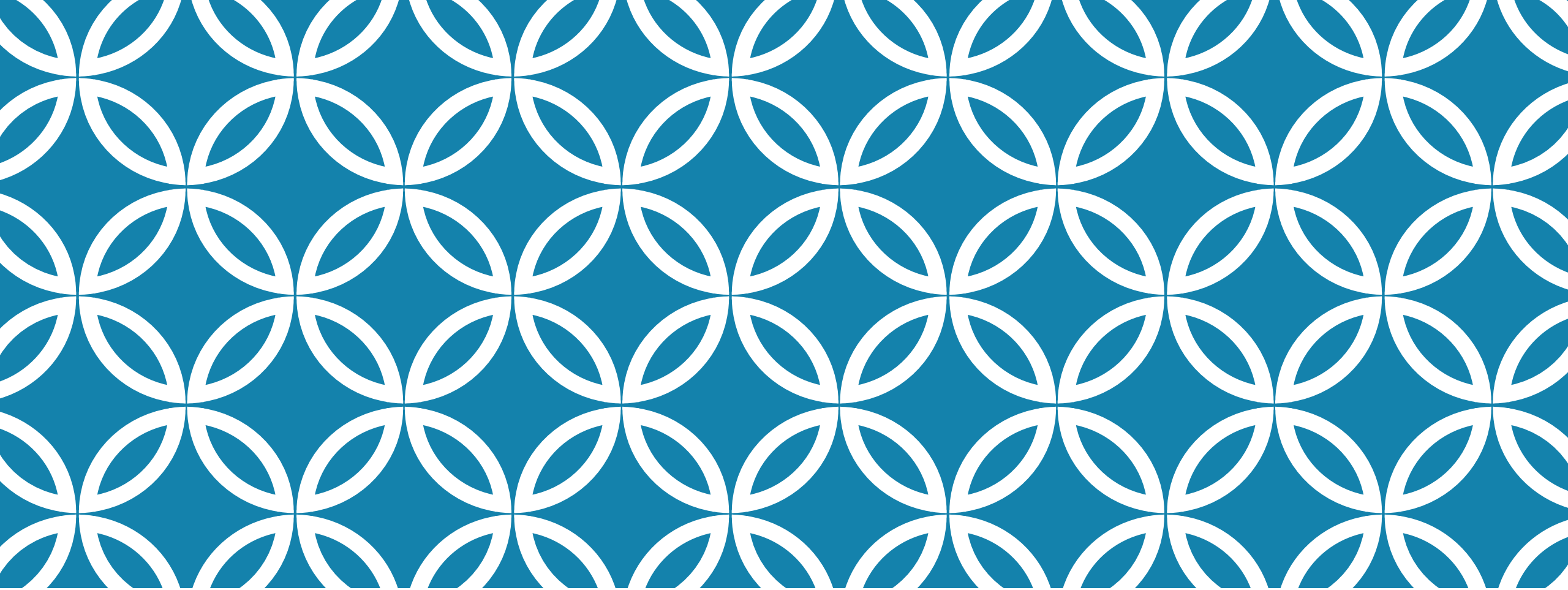
- We've been experimenting with several foundation NLP models for a prototype item-description automated translation service
- We picked the best one based on a thorough evaluation with BLEU
- "This model will definitely deliver the best business value in production later"

CHALLENGES

ML systems cannot be built in an ad hoc manner, isolated from other IT initiatives like DataOps and DevOps.

ML systems cannot be built without adopting and applying sound software engineering practices, while taking into account the factors that make operationalizing ML **different** from operationalizing other types of software

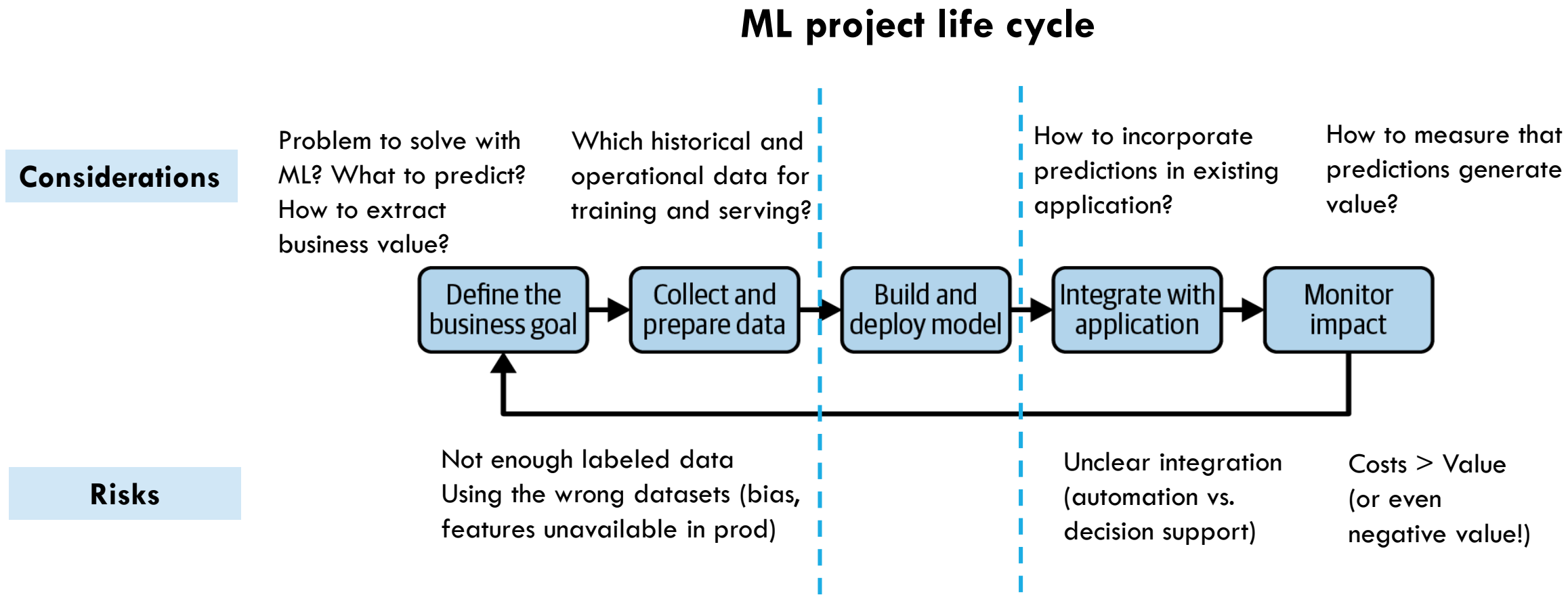
Organizations need an automated and streamlined ML process (+ mid to senior talents and change-management)



PROCESSES AND ROLES

Part I: Concepts

PROCESSES AND ROLES



PROCESSES AND ROLES

Automated ML Pipelines:

No one-off model development

Triggered when data, code, or parameters change

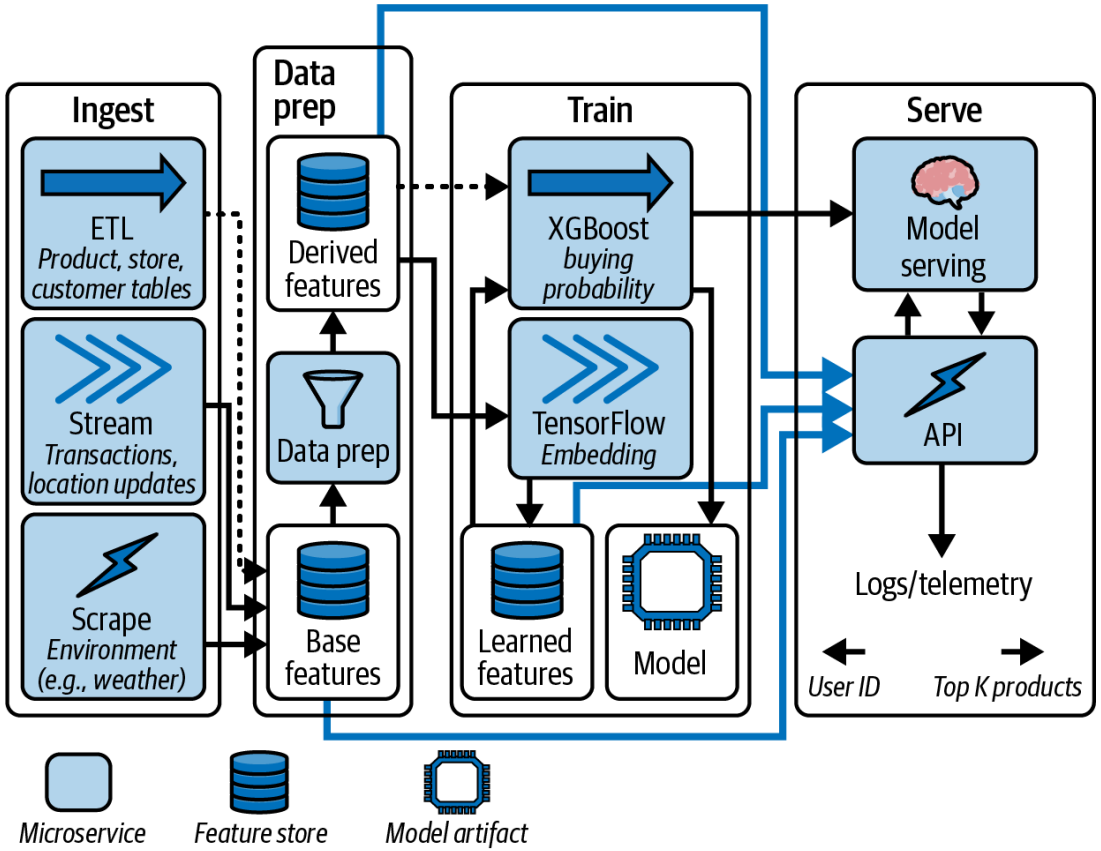
Executed in a **loop** (different data or parameters)

All inputs/outputs must be recorded and **versioned**

- code, data, parameters; operational data (type of hardware, logs, etc.); predictions/evaluations

Part of an extensive **app pipeline:**

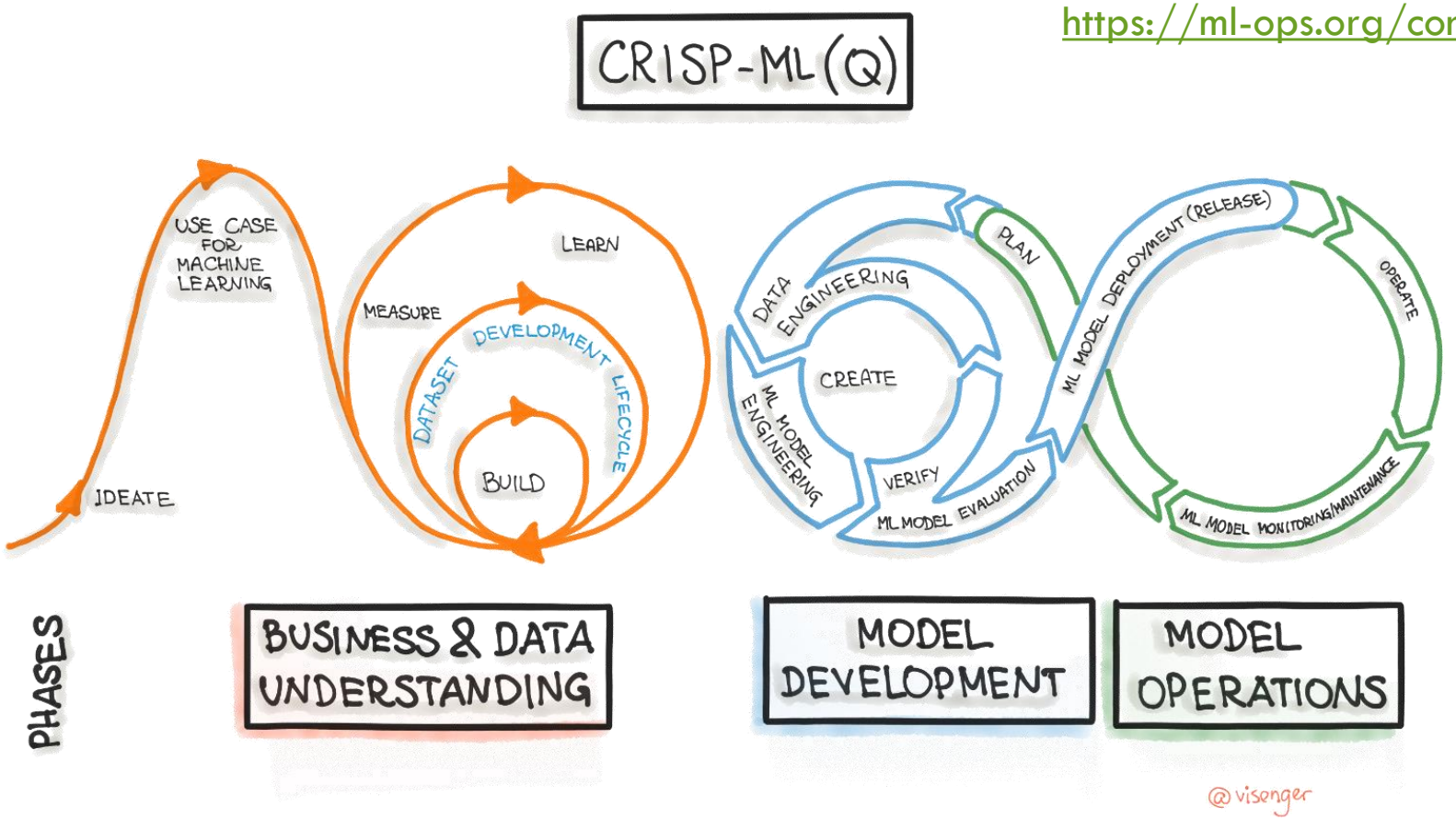
- API integration, real-time data enrichment



ML pipeline example: real-time product recommendations

PROCESSES AND ROLES

https://ml-ops.org/content/crisp-ml



PROCESSES AND ROLES

MLOps Processes

experimenting and developing a reproducible model training **procedure** (training pipeline code), with data prep, model training and evaluation

governing ML artifacts to support shareability, reusability, discovery, auditability, compliance

automating the process of packaging, testing, and deploying training pipelines

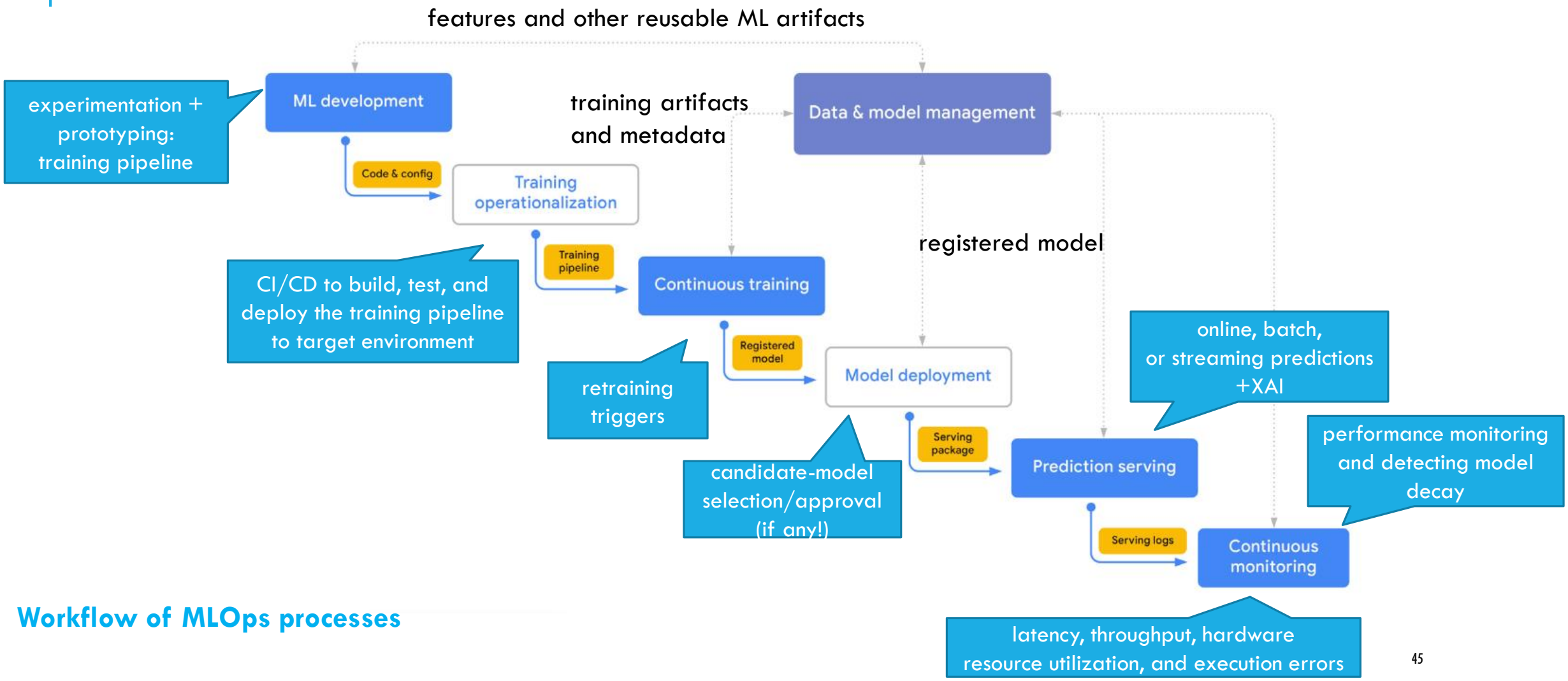
monitoring the effectiveness and efficiency of a deployed model

repeatedly executing the training pipeline in response to new data or to code changes, or on a schedule, potentially with new training settings

serving a model that is deployed in production for inference

packaging, testing, and deploying a model to a serving environment for production serving or online experimentation

PROCESSES AND ROLES



Workflow of MLOps processes

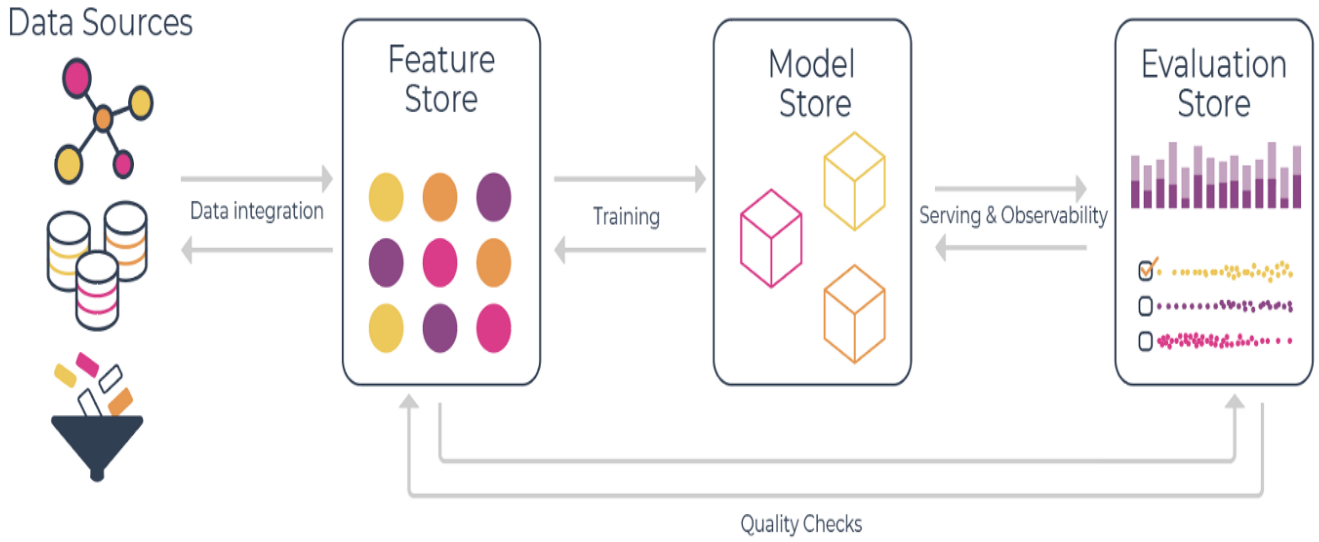
PROCESSES AND ROLES

Overview of major components of ML infrastructure (TBD)

Feature Store: handles offline and online (versioned) feature transformations

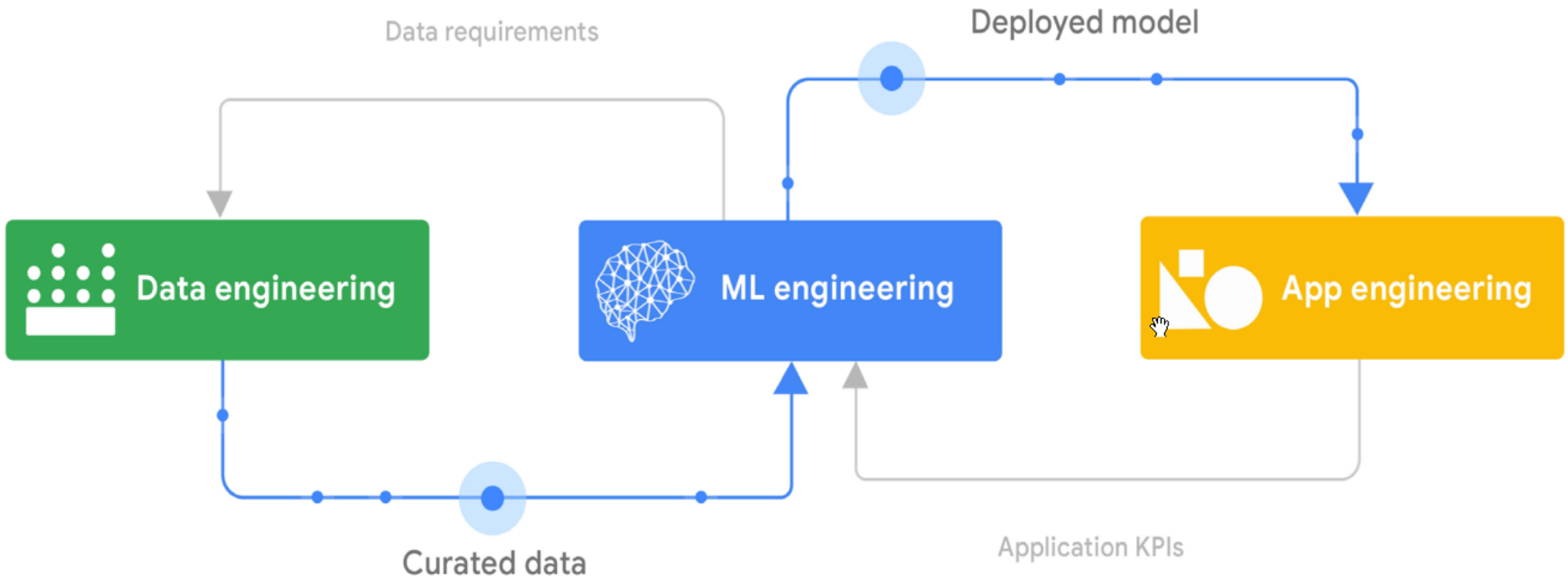
Model Store: central (versioned) model registry

Evaluation Store: central monitor of model performance



PROCESSES AND ROLES

Building an ML-enabled system

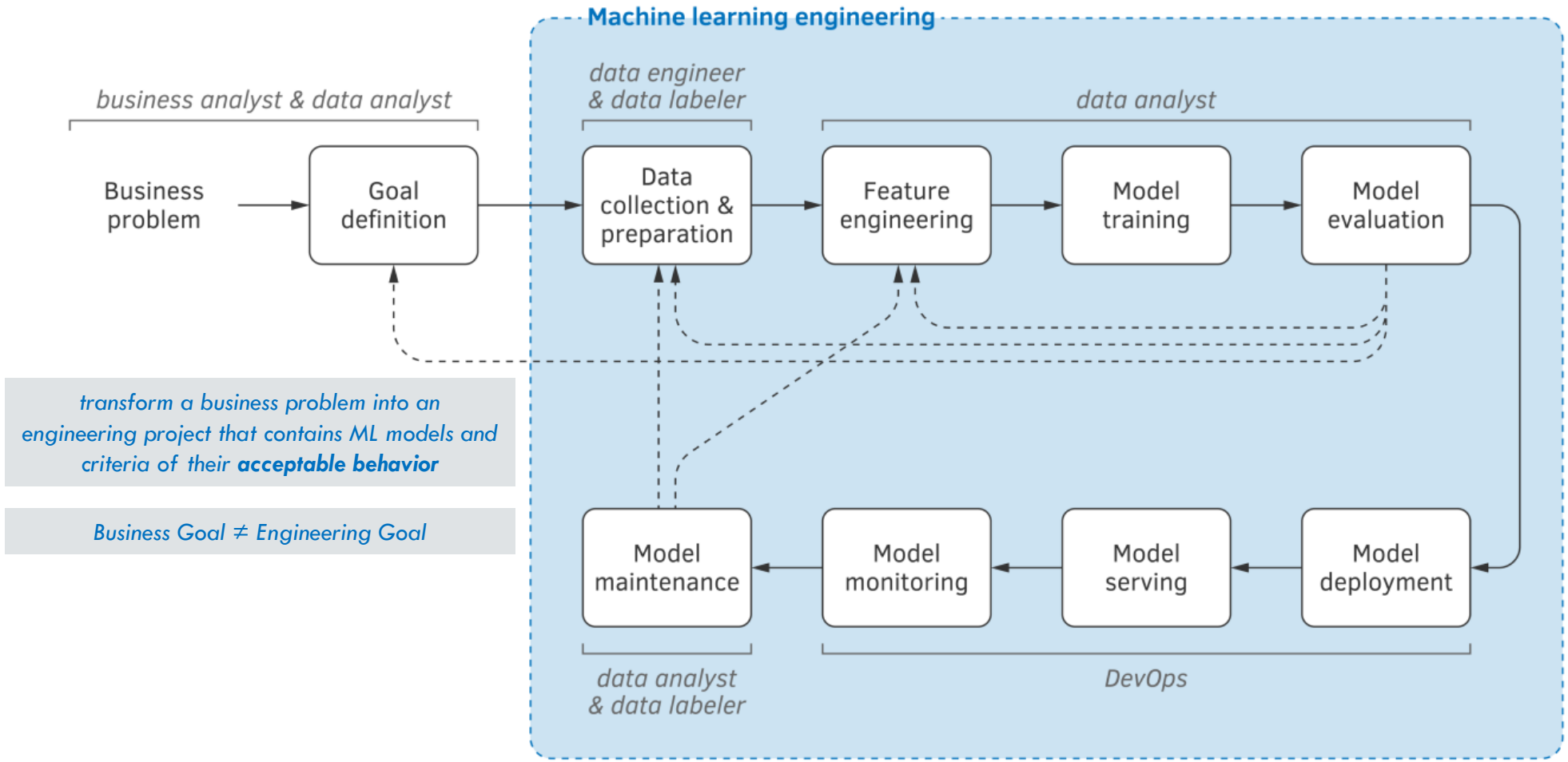


Ingesting, integrating, curating, and refining data to facilitate data analytics and ML

Integrating ML models into applications: deploying and monitoring business KPIs

PROCESSES AND ROLES

Machine Learning Project Life Cycle



*transform a business problem into an engineering project that contains ML models and criteria of their **acceptable behavior***

Business Goal ≠ Engineering Goal

PROCESSES AND ROLES

ML engineering

Development and operationalizing of production-grade ML systems

Superset of software engineering that handles the unique **complexities** of ML:

- Preparing and maintaining high-quality training data
- Tracking models in production to detect performance degradation
- Performing ongoing experimentation (new data sources, ML algorithms, and hyperparameters) and tracking these experiments
- Maintaining the veracity of models by continuously retraining them on fresh data
- Avoiding training-serving skews (inconsistencies between training and serving environments)
- Handling concerns about feeback loops, model fairness, and adversarial attacks

PROCESSES AND ROLES

Data scientists/analysts: develop solutions using ML for various business problems, by using existing algorithms or pre-trained models, and optimize them in the context of the problem statement

MLEs: take the models prepared by the data scientists and take them to production

- Model optimization to make it compatible with the custom deployment constraints
- Building infrastructure for:
  - experimentation, A/B testing, nodel management, containerization, deployment
- Monitoring the model performance once deployed

PROCESSES AND ROLES

Skills

Data Scientist	MLE
Problem-solving	Programming
Programming	Data infrastructure
Statistics	Data modeling
Machine learning	Software engineering
Data Analytics	Machine learning frameworks
Data visualization	Conceptual knowledge of ML
Written and verbal communication skills	Statistics

PROCESSES AND ROLES

Data engineers: software engineers responsible for developing automated data pipelines, in which raw data is transformed into analysis-ready data

- Structure the data and integrate it from various resources
- Write on-demand queries on that data and make sure that the data is easily accessible by data consumers
- Not expected to know machine learning and work separately from machine learning engineers in a data engineering team

PROCESSES AND ROLES

DevOps engineers work closely with MLEs to automate:

- model deployment and serving
- model monitoring
- model maintenance

Smaller companies: MLEs = DevOps Engineers

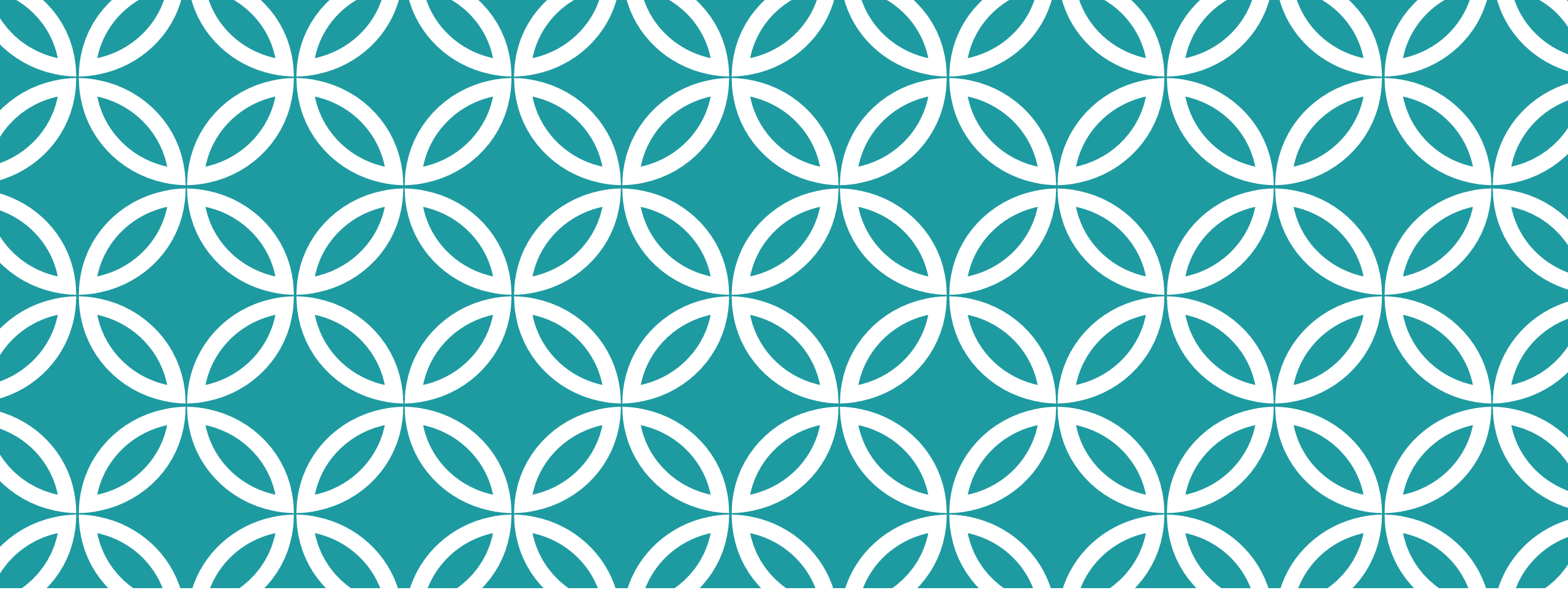
Bigger companies introduce MLOps

PROCESSES AND ROLES

Experts in data labeling are responsible for:

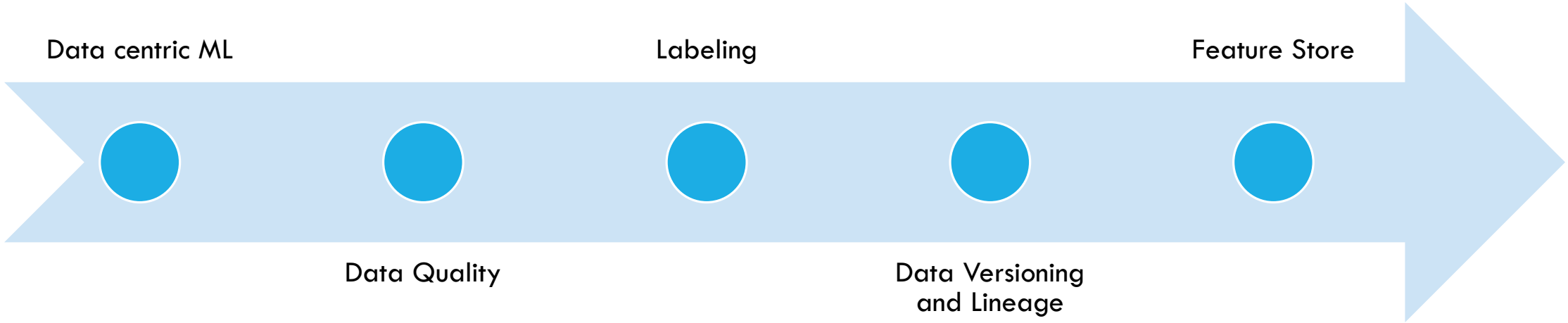
- assigning manually or semi-automatically labels to unlabeled examples
- building labeling tools
- manage outsourced labelers and solutions
- validate labeled examples for quality

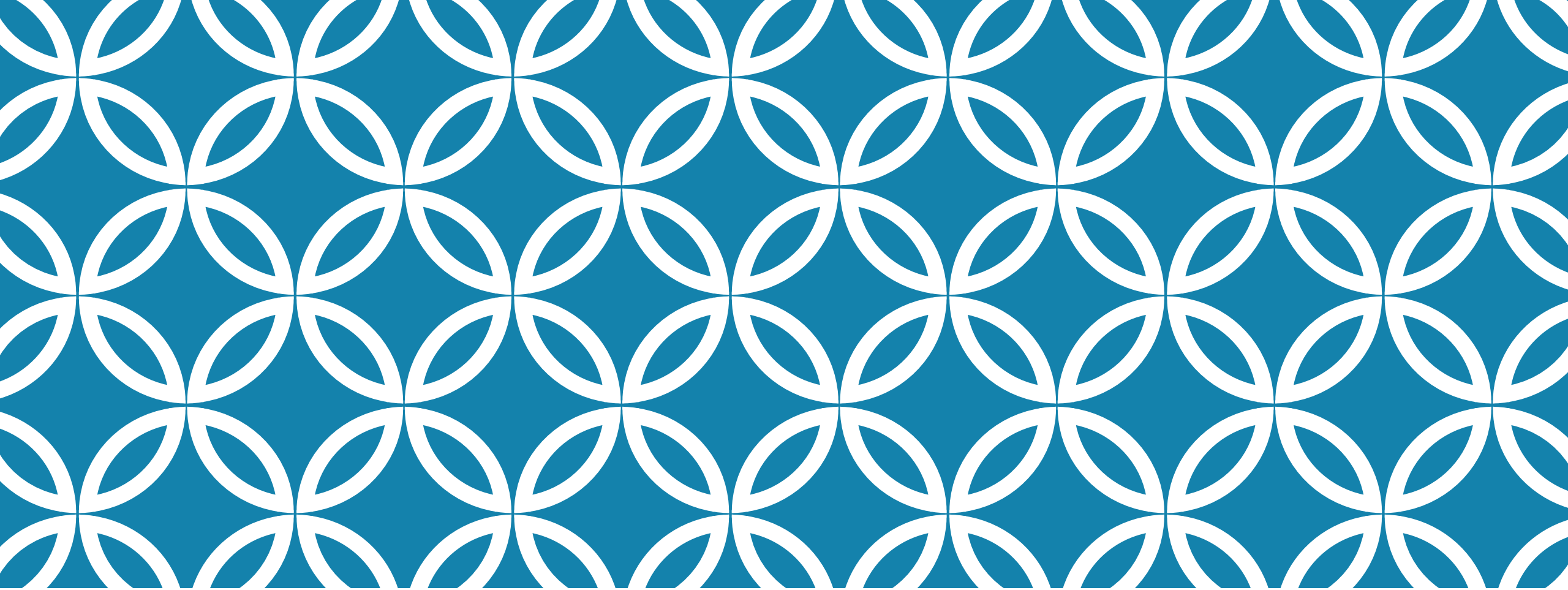
Domain experts: decision making about solution's goals, designing model features (based on non-ML baselines), evaluate usefulness and unacceptable behavior



PART II: DATAOPS FOR ML

PART II: DATAOPS FOR ML

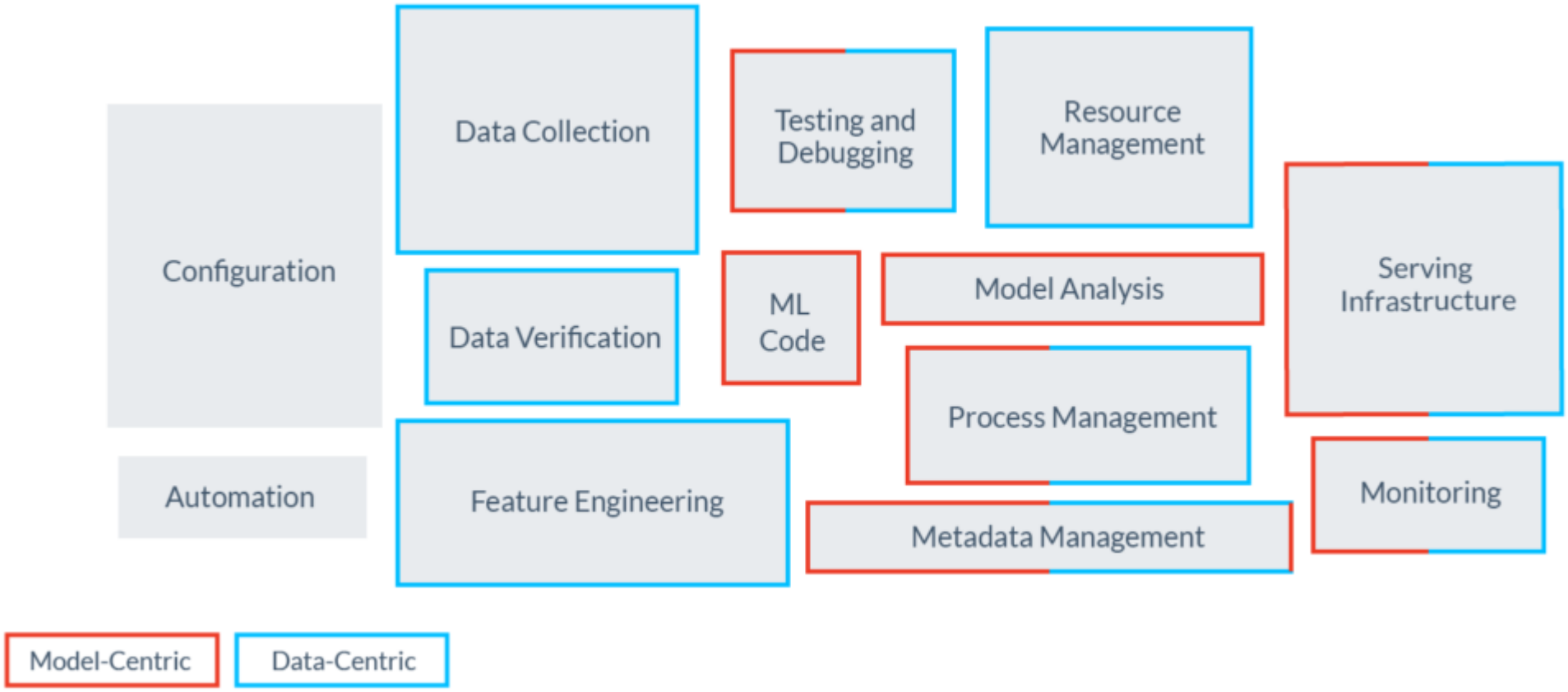




DATA CENTRIC ML

Part II: DataOps for ML

DATA CENTRIC ML



DATA CENTRIC ML

Industry focus: “data first, algorithm second” = more data of wide variety and high quality instead of trying to squeeze the maximum out of a learning algorithm

	Model Centric	Data Centric
Goals	Collect as much data as possible. The AI system improvement is mostly based on optimizing the model so that it can deal with the presence of noise. To overcome this, the data scientist tend to try the most recent state-of-the-art models and iterate until they find the best hyper-parameters to solve their problems.	Data consistency is key. More time invested in data quality tools, rather than in data volume. Multiple models perform well once the data is curated.
Approach	Historical data is more or less fixed after the initial data wrangling iterations. Model is improved iteratively through multiple experiments (MLOps).	The code/algorithm is more or less fixed. Data quality is iteratively improved, with periodical releases.

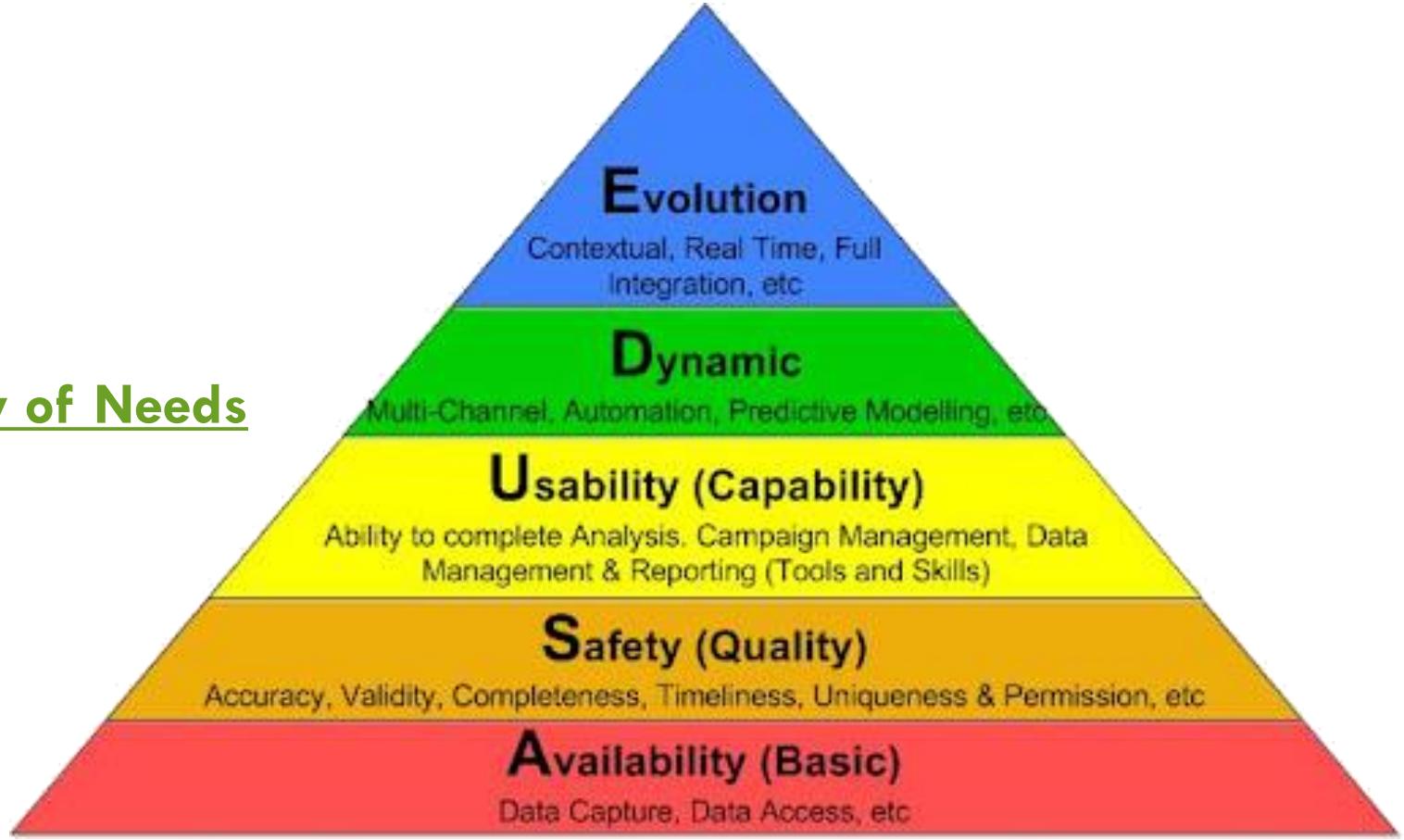
DATA CENTRIC ML

Main challenges

- **Create, maintain** and **reuse high-quality** data for training
- Costly **data access:**
  - When no self-service, data consumers overwhelm data producers
  - Costly process: unboxed response times, escalations
- **Inconsistency:** many teams might prepare similar datasets in different ways

DATA CENTRIC ML

Data Hierarchy of Needs



DATA CENTRIC ML

Data Sources (Storage Levels)

- Filesystems: text or binary, not versioned, can be erased or overwritten
- Distributed filesystems: such as NFS (Network File System) or HDFS; files are stored and accessed over multiple machines in the network
- Object storage: fundamental unit is an object (e.g., images, sound, or video files), versioned, built into an API service; examples: Amazon S3 and Google Cloud Storage (GCS)
- Databases/Data Warehouses: persistent, fast, and scalable storage of structured data
- Data lakes: an unstructured aggregation of data from multiple sources; data stored in its natural or raw format (blobs or files)
- Data meshes: delegate ownership, usage, and sharing of data, to data owners with the right business context
- Data streams: Intermediate storage hosting real-time events and messages

DATA CENTRIC ML

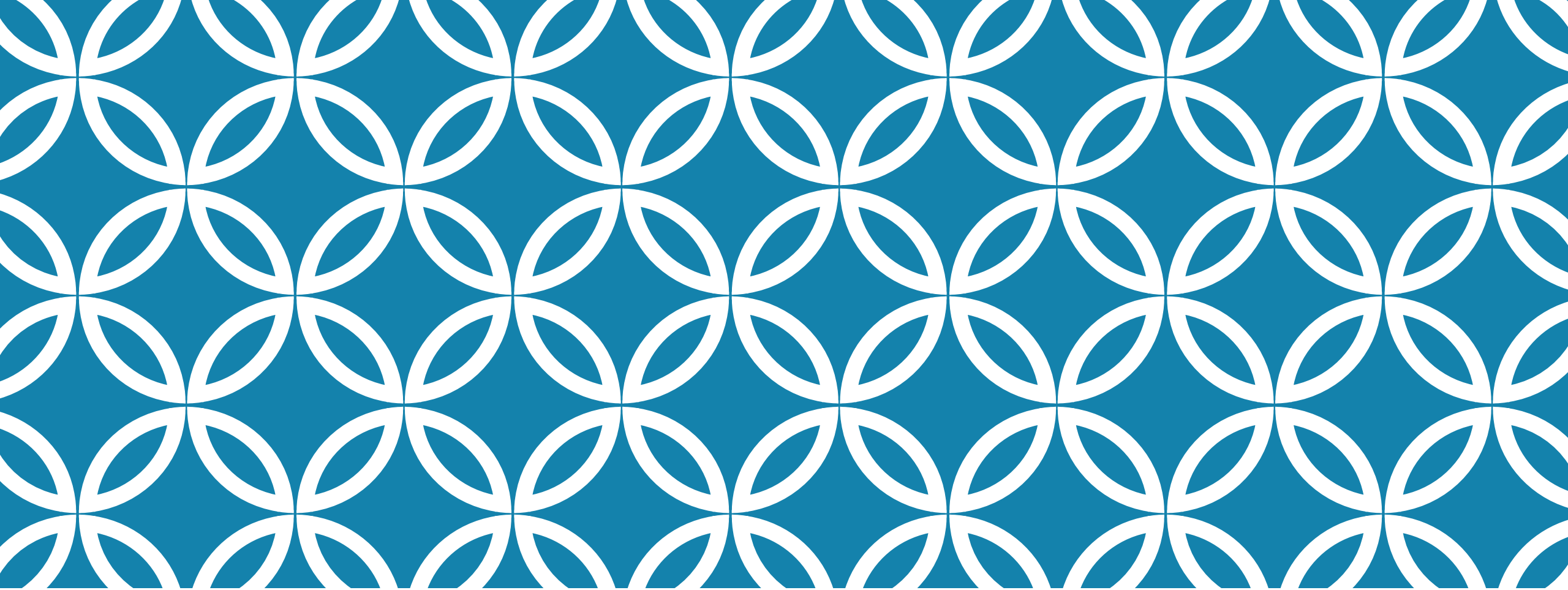
Data Mesh

Enables data consumers to:

- discovery data (centralized catalog)
- find out location and owners of data
- assess data quality and status (approval) for use in production
- understand how dataset relate to each other and how they are transformed

Google's Dataplex

Data mesh: enterprises can delegate ownership, usage, and sharing of data, to data owners who have the right business context



DATA QUALITY

Part II: DataOps for ML

DATA QUALITY

Data quality is one of the major factors affecting the performance of the model

Dimensions of data quality:

- accessibility
- relevance
- timeliness
- usability
- understandability
- reliability
- adequacy

DATA QUALITY

Usual problems with data quality:

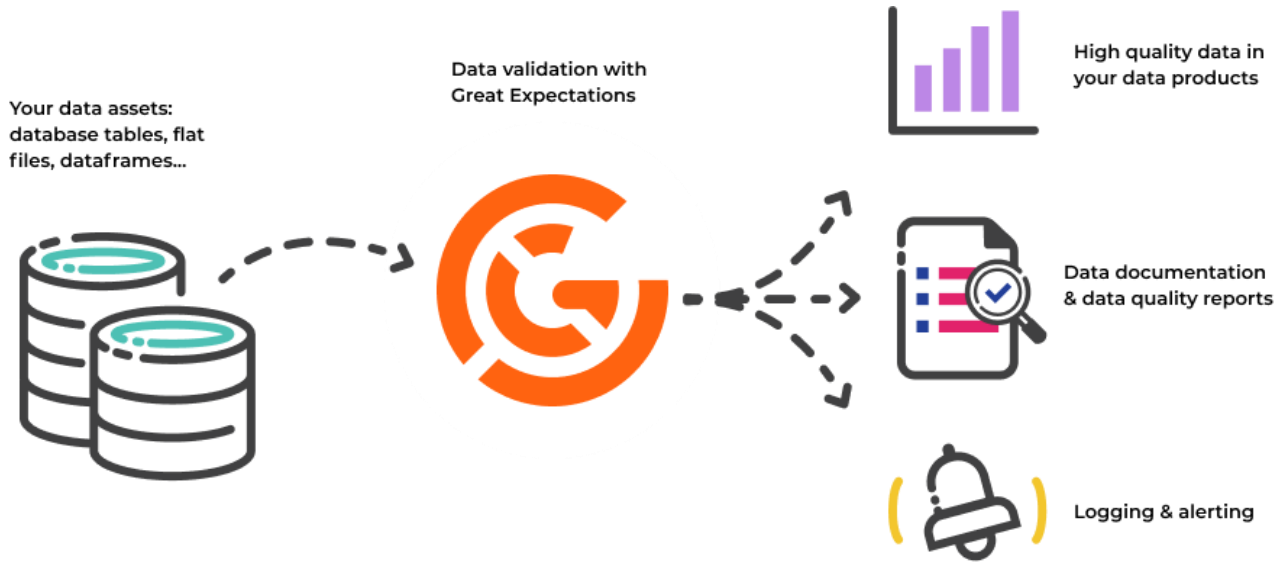
- Duplicates, noise, missing values (or “magic” numbers)
- Outliers and class imbalance (very uneven distribution of features and labels, resp.)
- Bias in the data: inconsistency with the phenomenon that data represents
- Outdated (predictive performance drops due to drifts in the data) and/or correlated data
- Target leakage: affecting several stages of the machine learning life cycle and invalidates the model

DATA QUALITY

Data Validation: Qualitative data meet consumer **expectations**

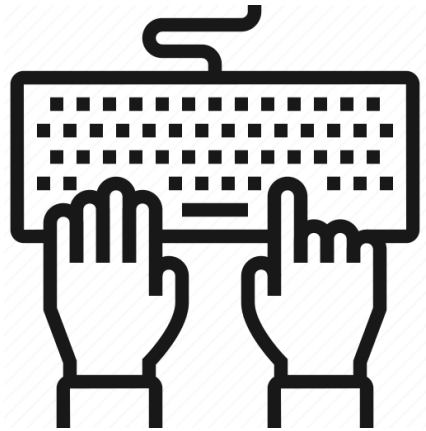
Great Expectations:

Add tests in ML pipeline to validate data against the expected schema and/or assumptions about values (e.g. they fall within expected ranges, or are not null)



Expectations are unit tests for data

DATA QUALITY



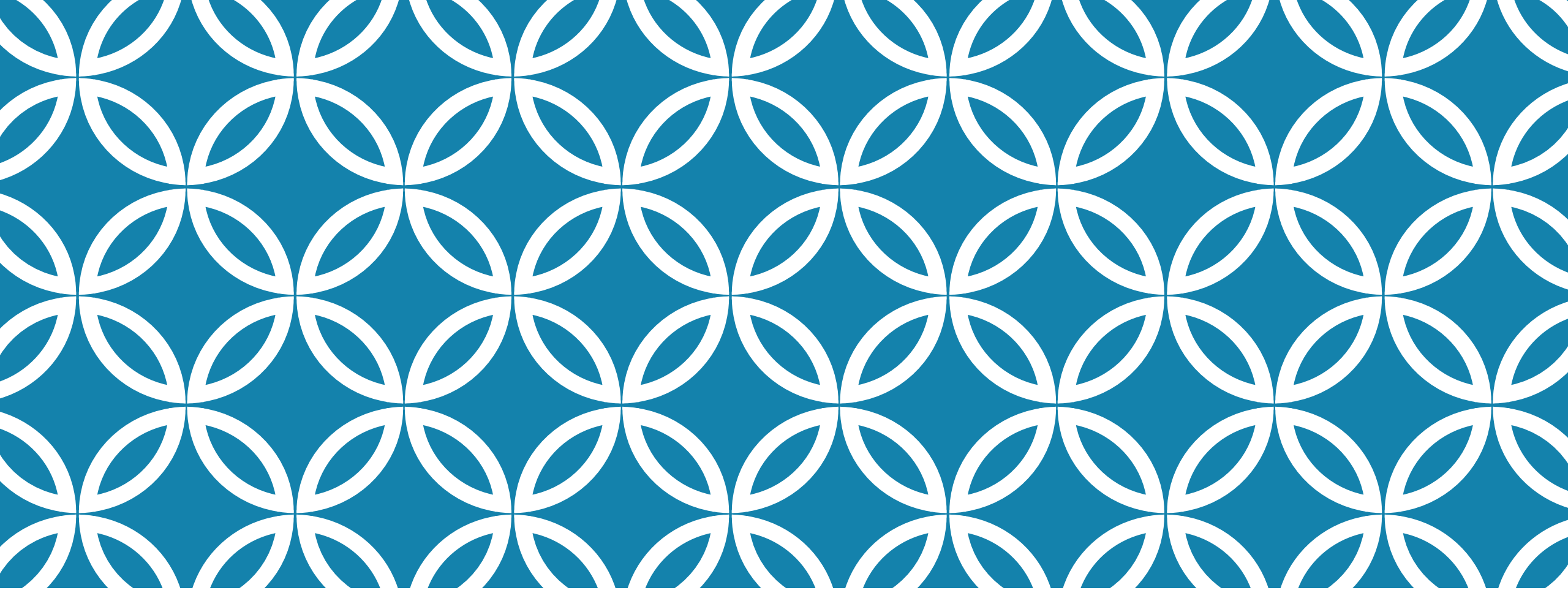
Exercise: Quickstart to Great Expectations

Test some more expectations from the use cases

Problems with Windows:

context.build_data_docs()

context.open_data_docs()



LABELING

Part II: DataOps for ML

LABELING

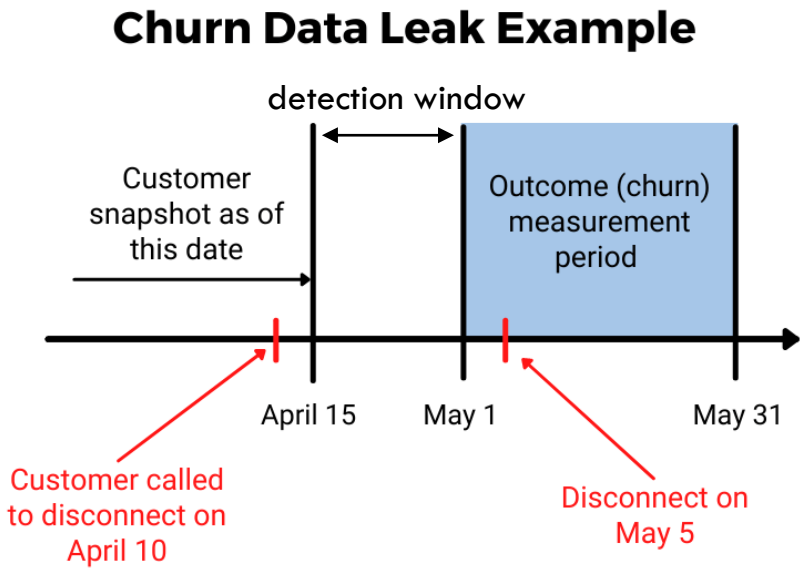
Labels acquisition:

- Availability/costs depending on the use case
- **Proxy labels**: pros and cons
- Harder for unstructured data: automate parts of the process (e.g., Label Studio)
- Challenges: need for domain expertise, risk of inconsistency, error proneness
- Labeling solution as part of the application (during **ingestion**) =>
  alerts, trigger model updates

LABELING

Caution when labels arrive with **delay**!

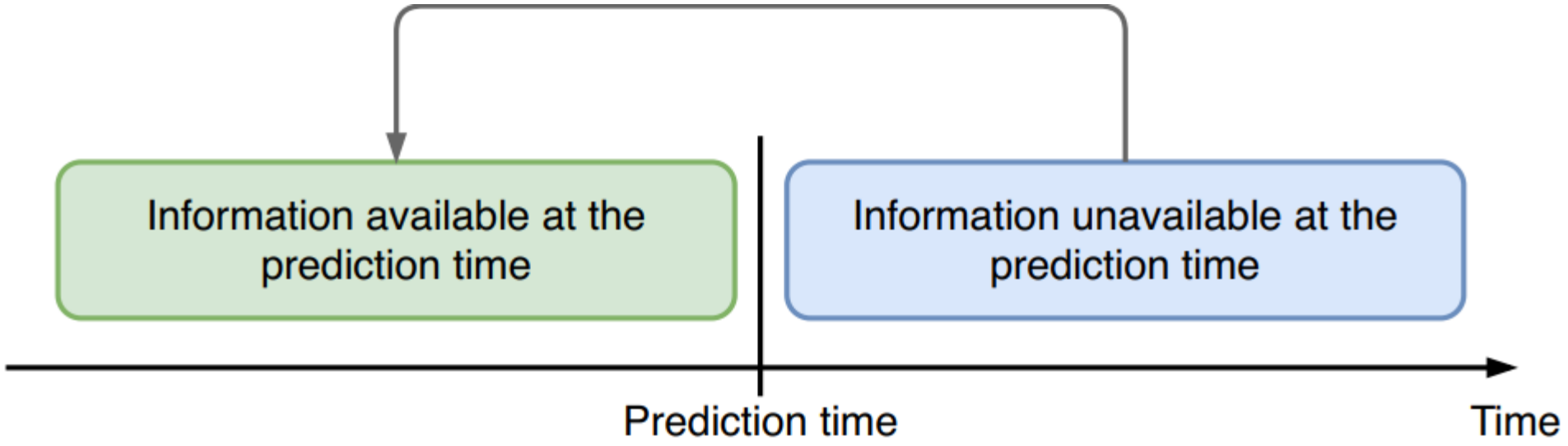
Example: churn => features shifted according to required detection window



Source:zyabkina.com

LABELING

Data (Target) leakage: erroneous use of information that relates to the target and is not available during training; model performance becomes overly optimistic



LABELING

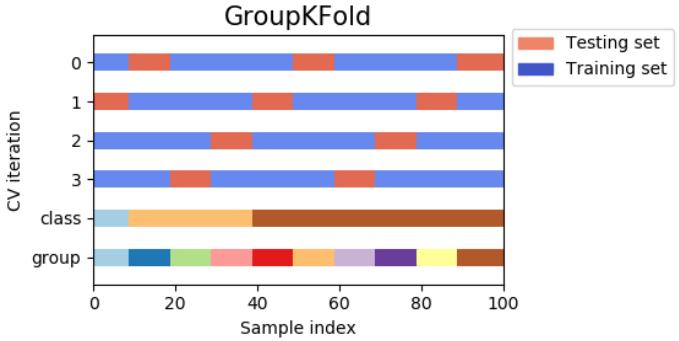
Causes of data leakage

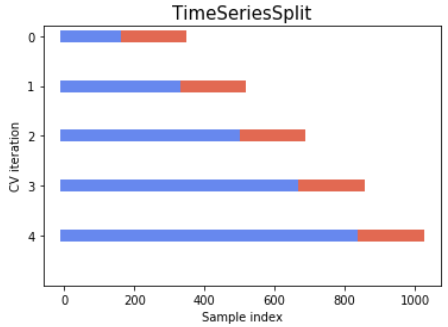
- Target is a function of a feature
  - Duplicated target (with different name/data type) or transformed target value (e.g., predicting “GDP” based on “GDP per capita” and “Population”)
- Feature hides the target
  - Feature that includes the target (e.g., predicting “Customer Gender” based on “Customer Segment” with values “M18-25” or “F25-35”)
- Feature from the future
  - Feature with values that are unavailable during prediction (e.g., predicting “Customer Will Pay Loan Back” during loan application when using a feature “Number of Late Payment Reminders”, which in production is always 0; predicting house price based on real-estate broker commission)
- Leaky feature transformations
  - Normalization or imputation that use the entire dataset before training-test split
- Leakage during partitioning
  - Training-Test split when same entity incorrectly participates both sets (e.g., face recognition for new users when photos of same users are both in training and test sets; or time-series forecasting without split based on time)

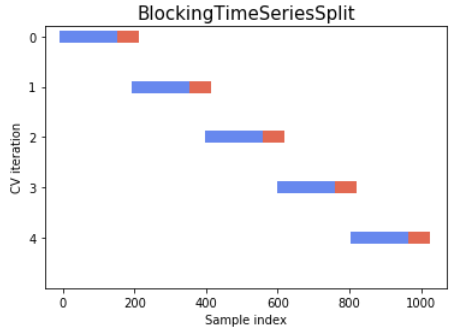
LABELING

How to detect/avoid data leakage:

- Red flag when model's performance is (nearly) perfect
- Red flag when some feature-importance scores are very high
- Understand business context to know which features are available during prediction
- Use pipelines for feature transformations
- Use Grouped Cross Validation or Time-Series Split when needed

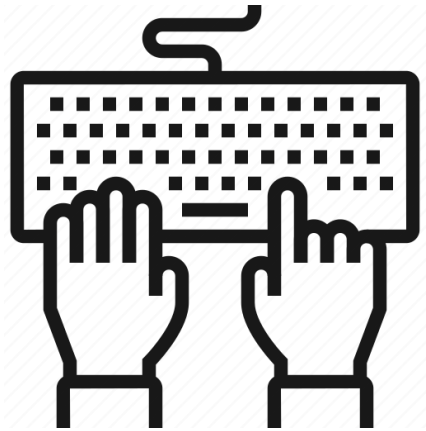


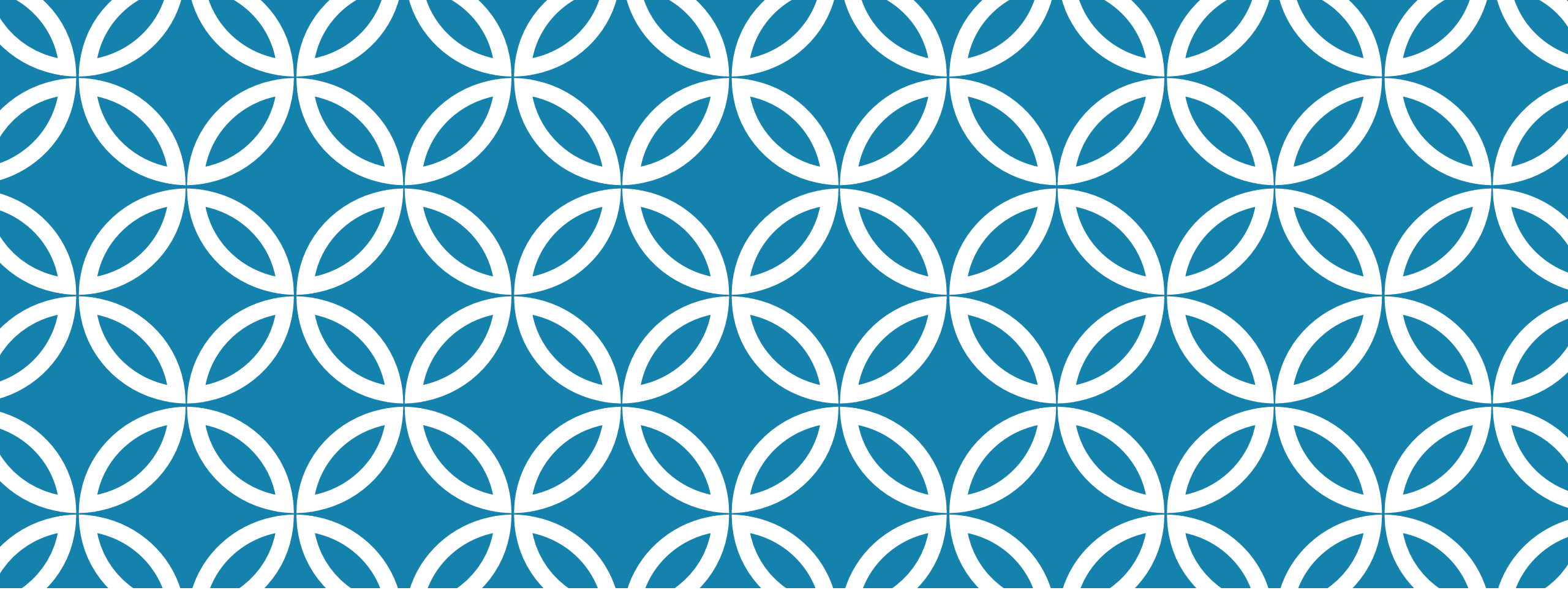




LABELING

Optional exercise: Detect Data Leakage





DATA VERSIONING AND LINEAGE

Part II: DataOps for ML

DATA VERSIONING AND LINEAGE

The need: enable access to the data used in training

Why? reproduce and **explain** model behavior; address **regulatory compliance**

Data Lineage: save and manage metadata after each model build (sources, transformations/code, configurations/parameters, dependencies, etc.)

Data Versioning: save/track the data used for each model build

DATA VERSIONING AND LINEAGE

Data Versioning Levels

Level 0: data is **unversioned**

Level 1: data is versioned as a **snapshot** at training time

Level 2: both data and code are **versioned** as one asset

Level 3: using or building a **specialized data versioning** solution (interoperate with code versioning software, such as Git)

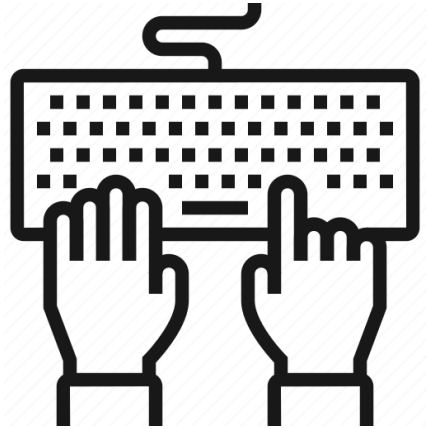
DATA VERSIONING AND LINEAGE

Data-versioning system: DVC

- Works like Git but for large file-based datasets and model artifacts
- DVC is technically not a version control system by itself!
- It manipulates .dvc files, whose contents define the data file versions.
- Git is already used to version your code, and now it can also version your data alongside it.

DATA VERSIONING AND LINEAGE

Explore the DVC „Get Started“



DATA VERSIONING AND LINEAGE

Dynamic data needs to be repeatedly filtered, cleaned, and transformed => DVC can track data **pipelines** by creating stages

Stages:

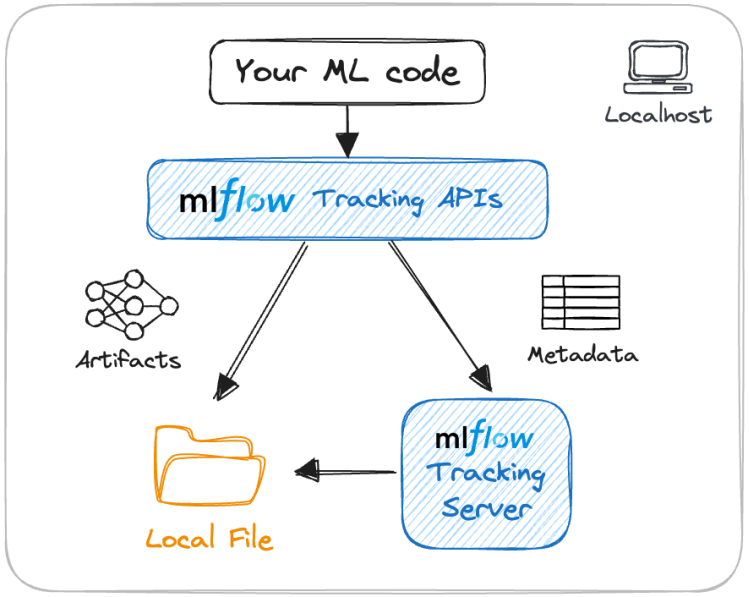
- processing steps (usually scripts/code tracked with Git) combined in form of a pipeline
- allow connecting code to its corresponding data input and output
- specify which parameters (-p), dependencies (-d) and outputs (-o) to an executable (python)

```
dvc stage add -n featurize \
              -p featurize.max_features,featurize.ngrams \
              -d src/featurization.py -d data/prepared \
              -o data/features \
              python src/featurization.py data/prepared data/features
```

DATA VERSIONING AND LINEAGE

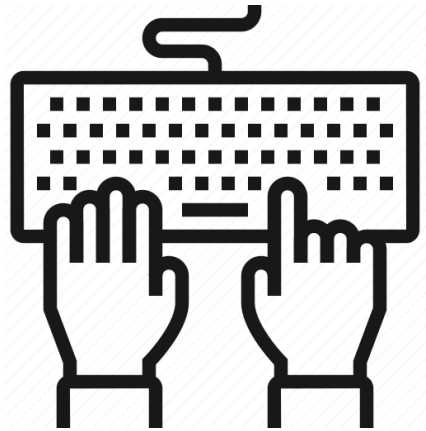
Data-versioning subsystem: MLflow Data Tracking

- MLflow is an open source platform for managing the end-to-end ML lifecycle
- Core component: **MLflow Tracking** (API and UI) for logging machine learning runs, their inputs and outputs
- Not a complete data versioning solution (no data lineage or deduplication)
- Enables logging and indexing the data outputs of every run along with the source code, parameters, and some execution details
- Can be manually integrated with other tools like DVC



DATA VERSIONING AND LINEAGE

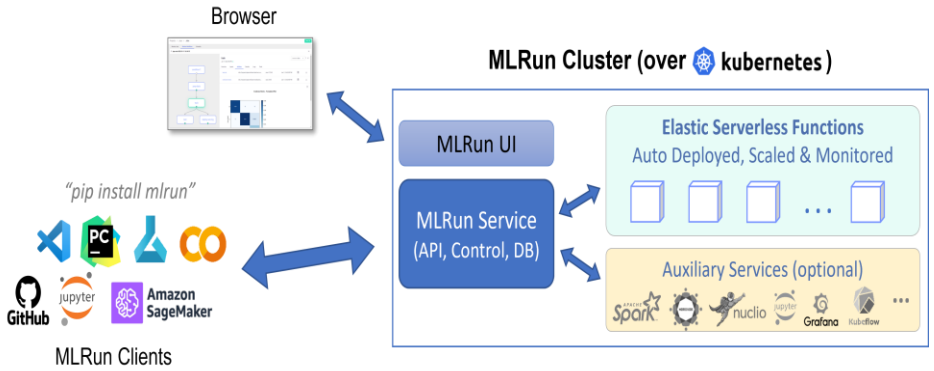
Explore the MLflow Dataset Tracking Tutorial



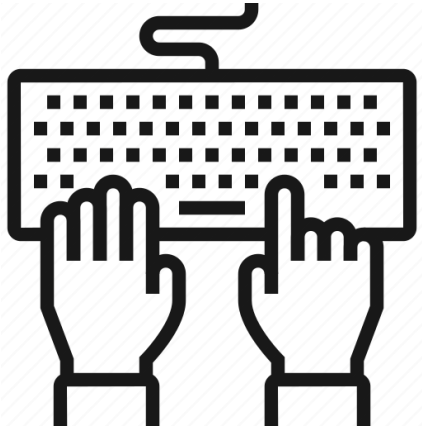
DATA VERSIONING AND LINEAGE

Data-versioning subsystem: MLRun

- MLRun is an open source MLOps **orchestration** framework with multiple sub-components for the complete ML lifecycle.
- Supports a variety of data objects (data stores, files, streams, feature sets, feature vectors), each with unique metadata, actions, and viewers
- Every object has a unique version ID, tags, user-defined labels, and relations to other objects in the project
- MLRun data objects and artifacts automatically get metadata, (who produced them, when, etc.), which data sources were used, schema, statistics, preview, etc.
- Deduplication and versioning are supported



DATA VERSIONING AND LINEAGE



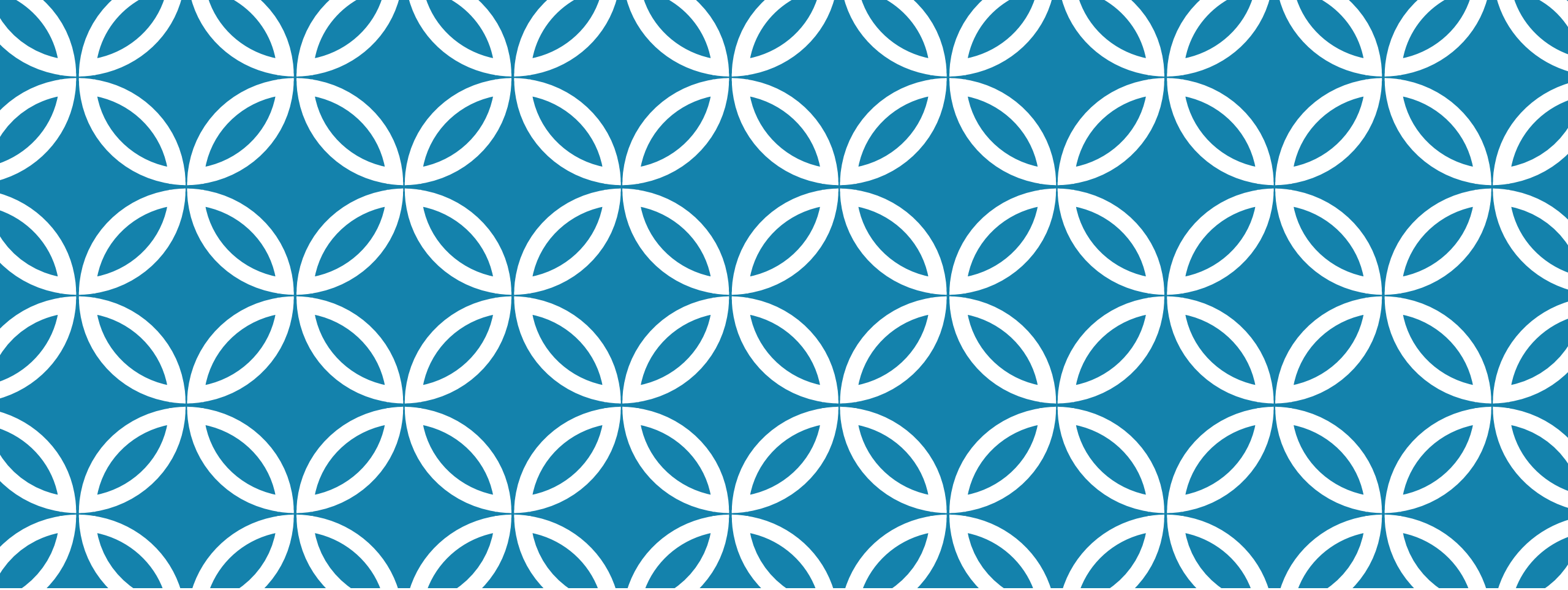
Explore the MLRun Quick start tutorial

(without deployment)

DATA VERSIONING AND LINEAGE

Other data-versioning systems:

- Pachyderm: Pachyderm is a data pipeline and versioning tool built on a containerized infrastructure
- Amazon SageMaker ML Lineage Tracking
- Azure ML datasets



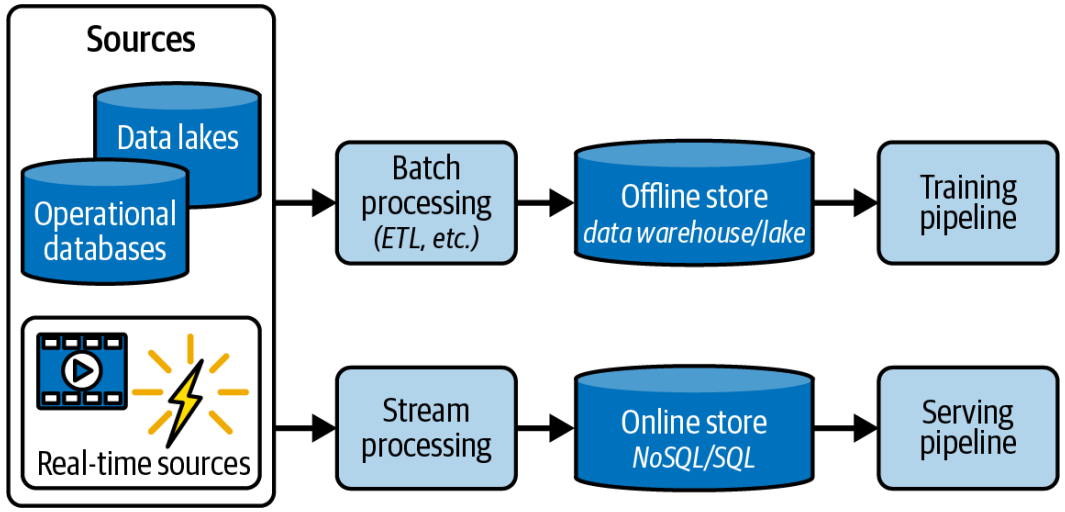
FEATURE STORE

Part II: DataOps for ML

FEATURE STORE

Two (automated) data pipelines:

- **training** pipeline (batch)
- **serving** pipeline (streaming)



Challenges

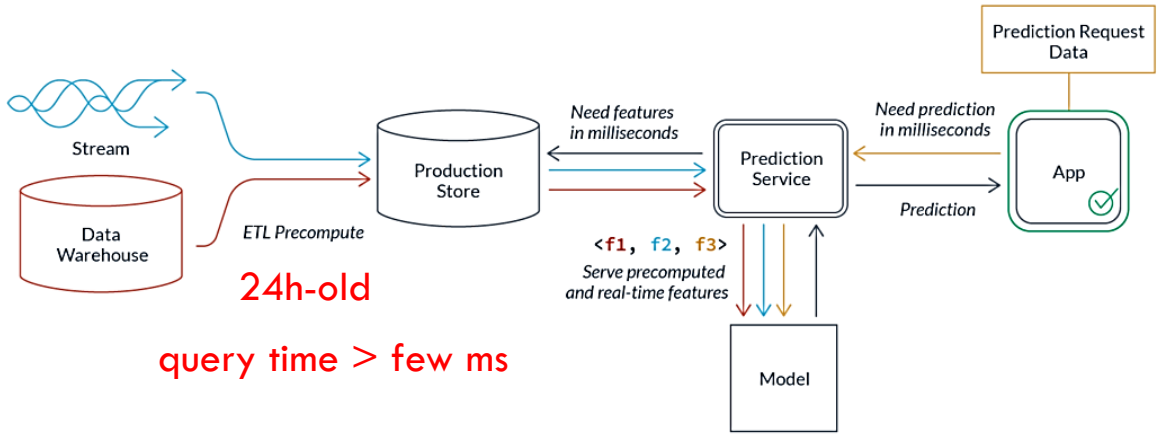
- **Training-serving skew**: discrepancies between serving and training data
- **Efficiency**: real-time features (e.g., total number of purchases in the last hour) need to be calculated fast

FEATURE STORE

Batch vs. real-time features: different data sources with different freshness

Retrieving multiple features from different sources, times, and with different indexes can be a complex analytics task

	Data Warehouse *(e.g. Snowflake)*	Transactional *(e.g. MySQL)*	Streams *(e.g. Kafka)*	Prediction Request Data
Data Quantity	Complete History	Recent History	Near Real-time	Real-time
Data Freshness	Low *(hours / days)*	Medium *(milliseconds – seconds)*	Medium *(milliseconds – seconds)*	High



FEATURE STORE

More challenges:

- Feature development **duplicated** for every new project
- Lack of simple **access** to production-ready features at scale
- Disjointed or nonexistent model and feature **monitoring**

Need for a feature store:

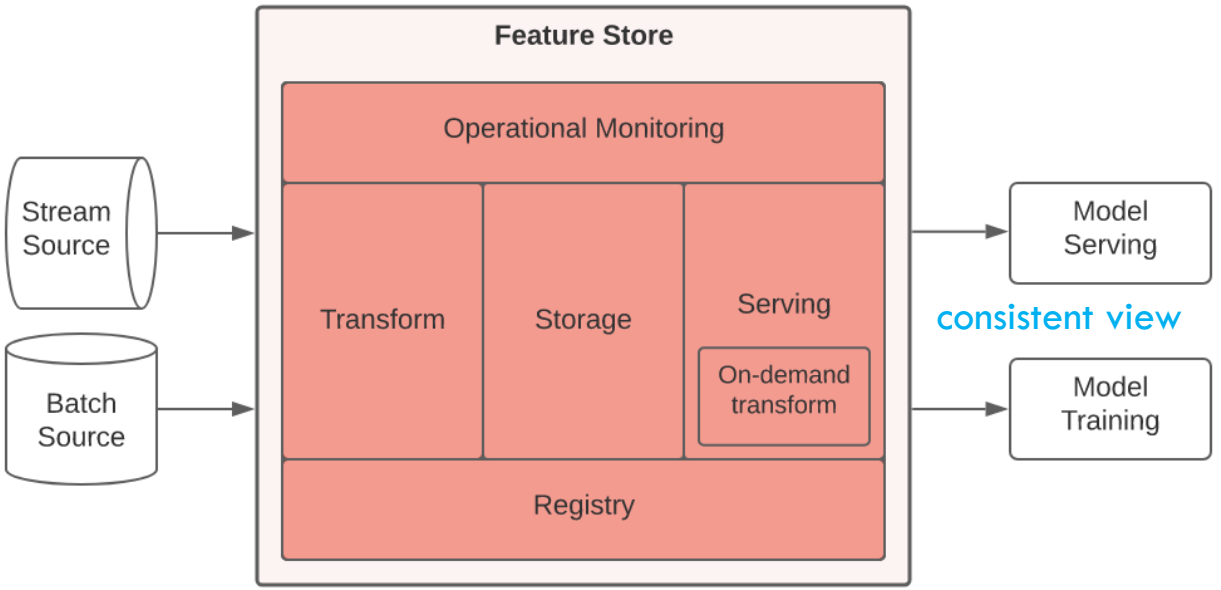
- **Sharing** all available features (+metadata) across the organization
- Accessing a **catalog** to easily find features when starting a project
- Incorporating **data transformation** and storing features ready to be used for training serving

FEATURE STORE

Common feature store architecture

compute and monitor metrics (availability, utilization, staleness)

orchestrate data transformations, configured by definitions in feature registry and producing fresh feature values



consistent view

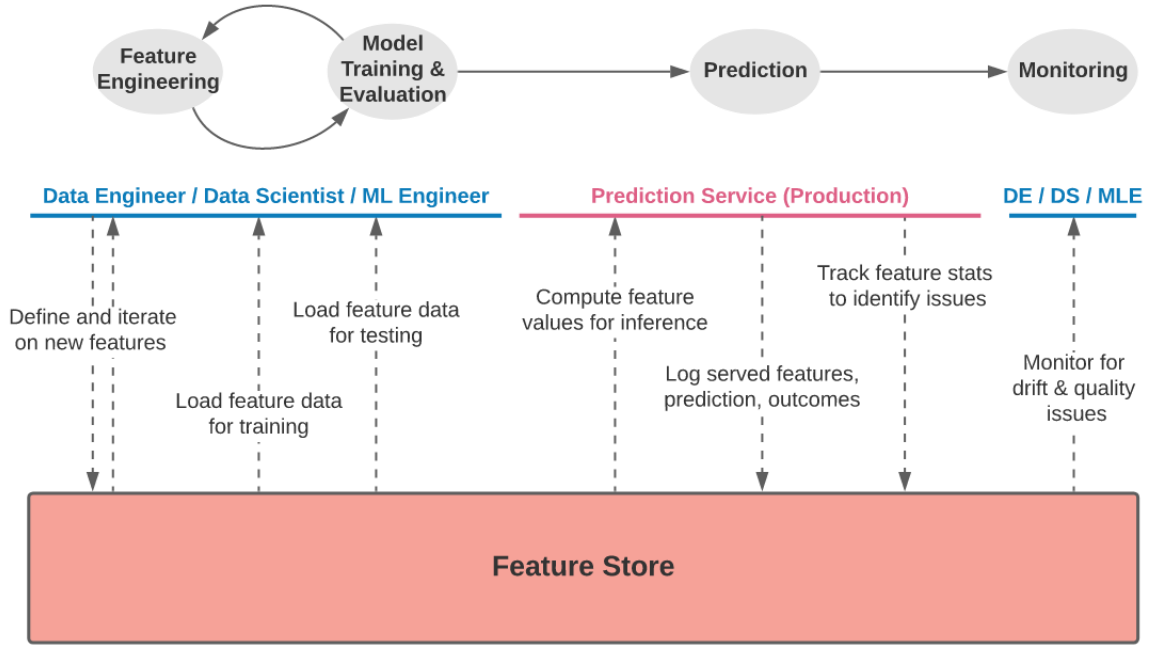
online (key-val) vs. offline (db) storage

common catalog to explore, develop, collaborate on

FEATURE STORE

Feature-store functionalities

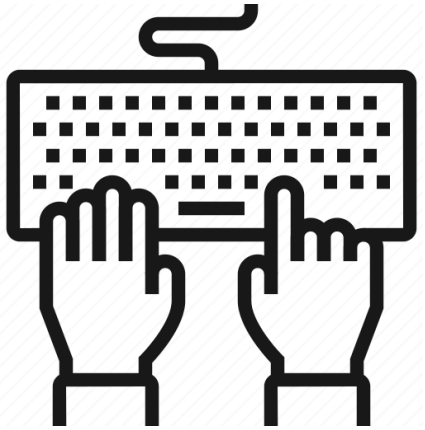
- Feature computation: compute once, save, and reuse; *single point of truth*
- Feature consistency: unify the logic for both batch features and streaming features; avoid training/serving-skew
- Feature management: sharing features among different models/apps; feature discovery (a.k.a. *feature catalog*)
- Feature governance: security and versioning



FEATURE STORE

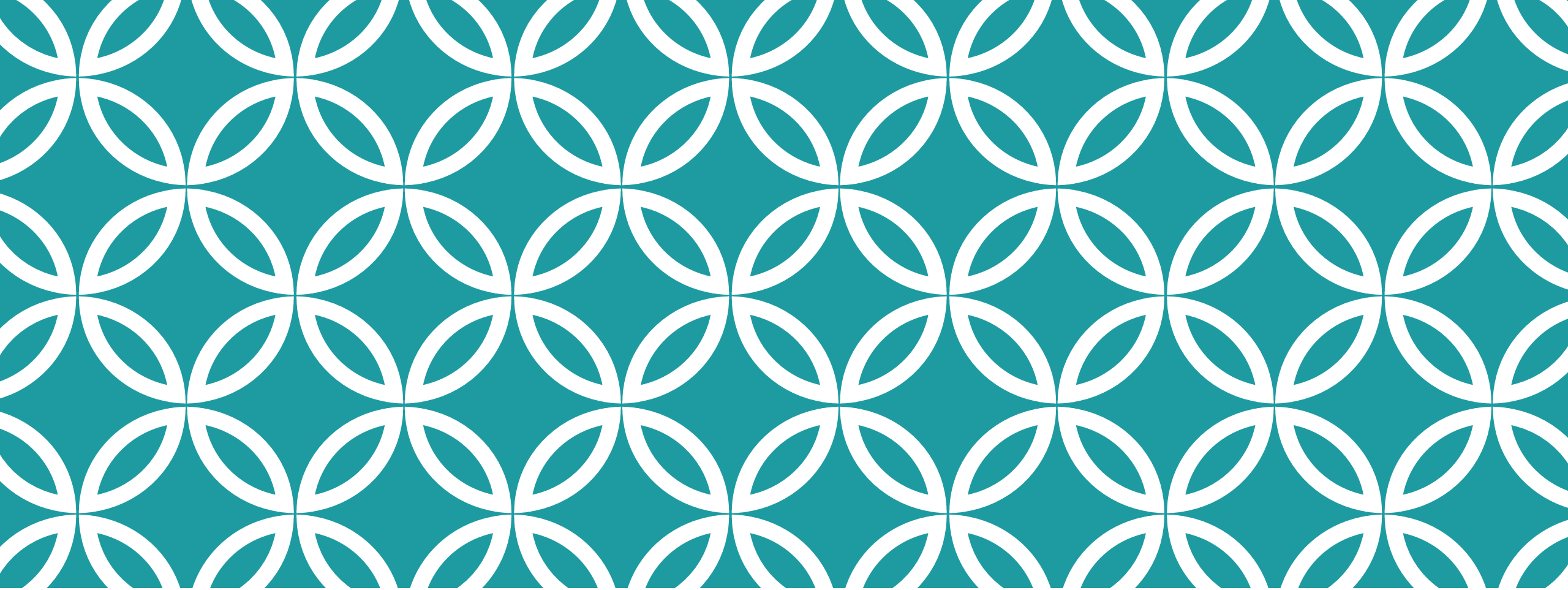
Note: Some feature stores are focused on cataloging and don't automate the process of ingestion and online or offline transformation => make sure you properly evaluate before selecting a solution

Feast: when you already have transformation pipelines and need a storage and serving layer to use them in production



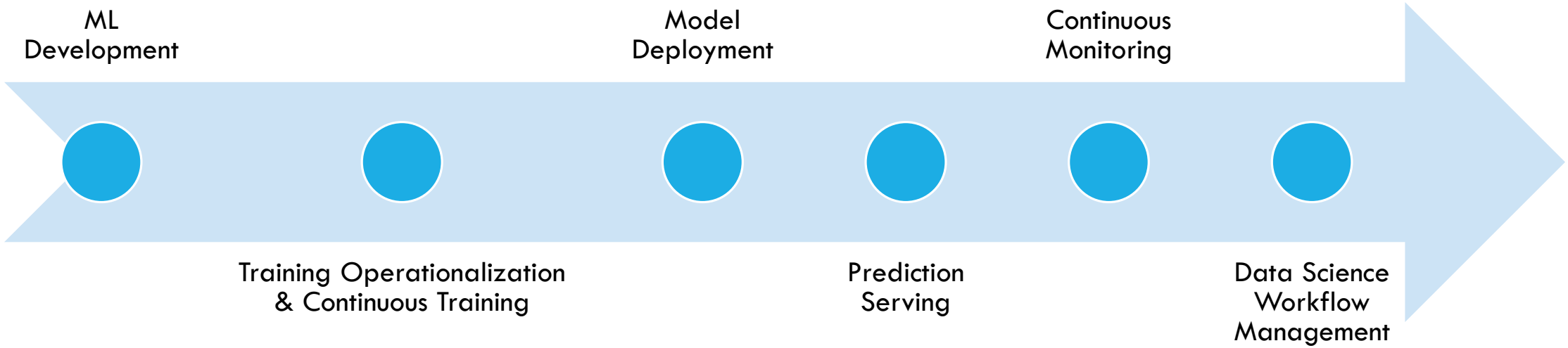
Explore the Feast Quickstart

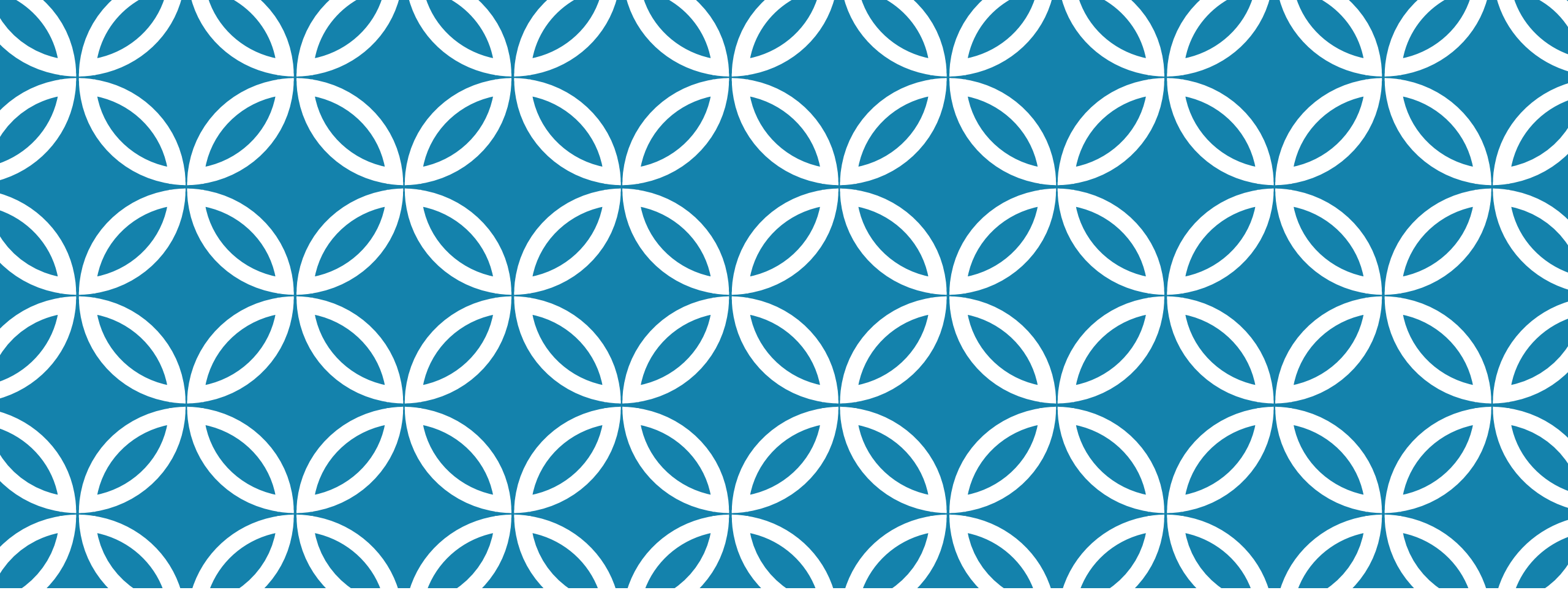
(until including Step 3d)



PART III: MLOPS

PART III: MLOPS





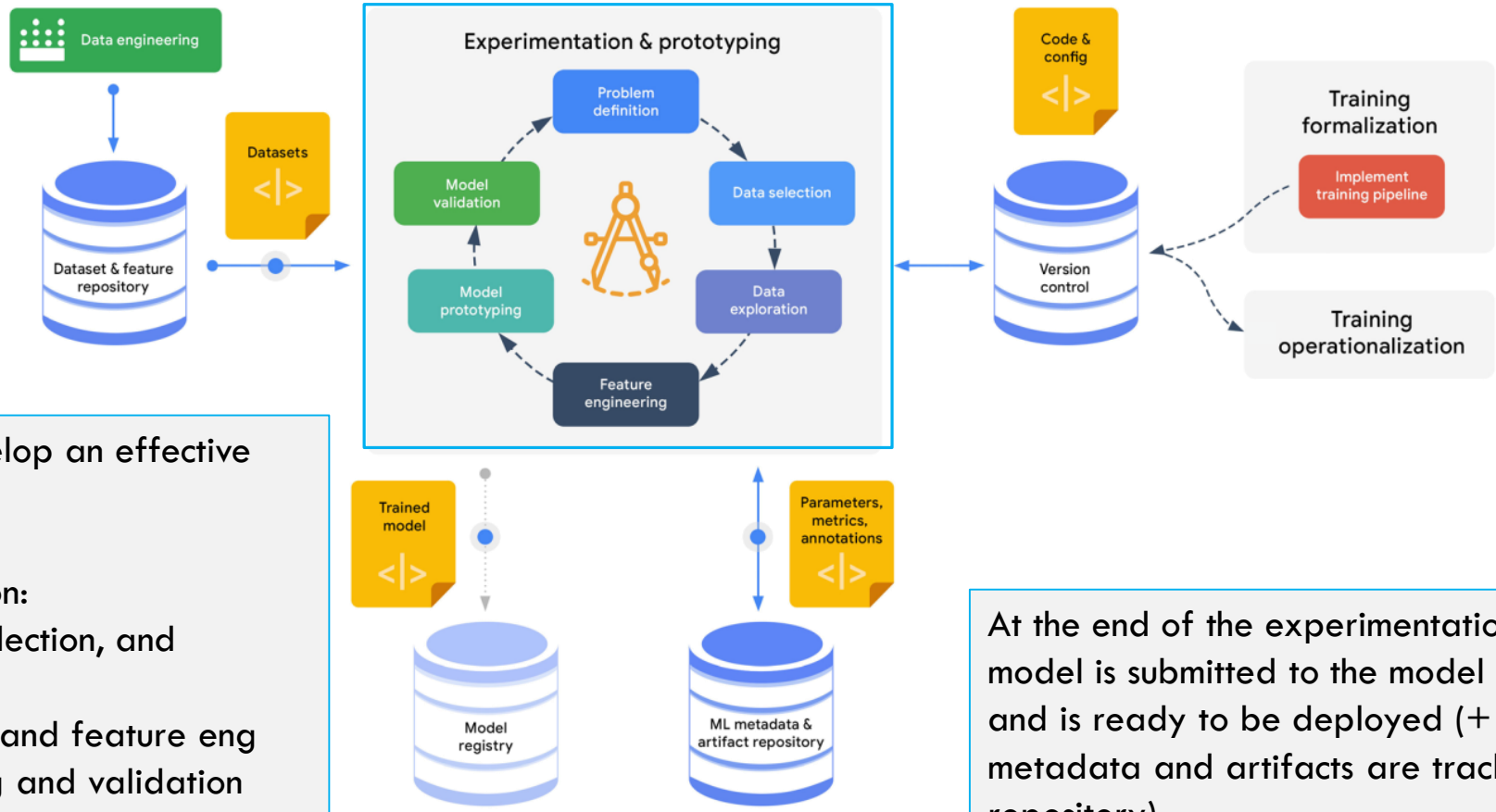
ML DEVELOPMENT

Part III: MLOps

ML DEVELOPMENT

ML development process

Key success aspects:
- experiment tracking (configurations)
- reproducibility
- collaboration



Experimentation: develop an effective prototype model

During experimentation:
- Data discovery, selection, and exploration
- Data preparation and feature eng
- Model prototyping and validation

At the end of the experimentation, the model is submitted to the model registry and is ready to be deployed (+ all metadata and artifacts are tracked in a repository)

ML DEVELOPMENT

Challenges during experimentation:

- Difficult to keep track of experiments
- Difficult to reproduce code
- No standard to package (and deploy) models
- No central store to store/manage models

ML DEVELOPMENT

Requirements: running, tracking, and comparing ML jobs

- **Running ML jobs**: data preprocessing, model training, hyperparameter tuning, and testing, computational resource allocation and pipeline automation
- **Tracking ML jobs**: logging of metrics, version control, results visualization, reproducibility, collaboration and compliance with industry standards
- **Comparing ML jobs**: performance evaluation, hyperparameter comparison, analyzing resource usage, cost analysis, and interpretability analysis, selecting best models

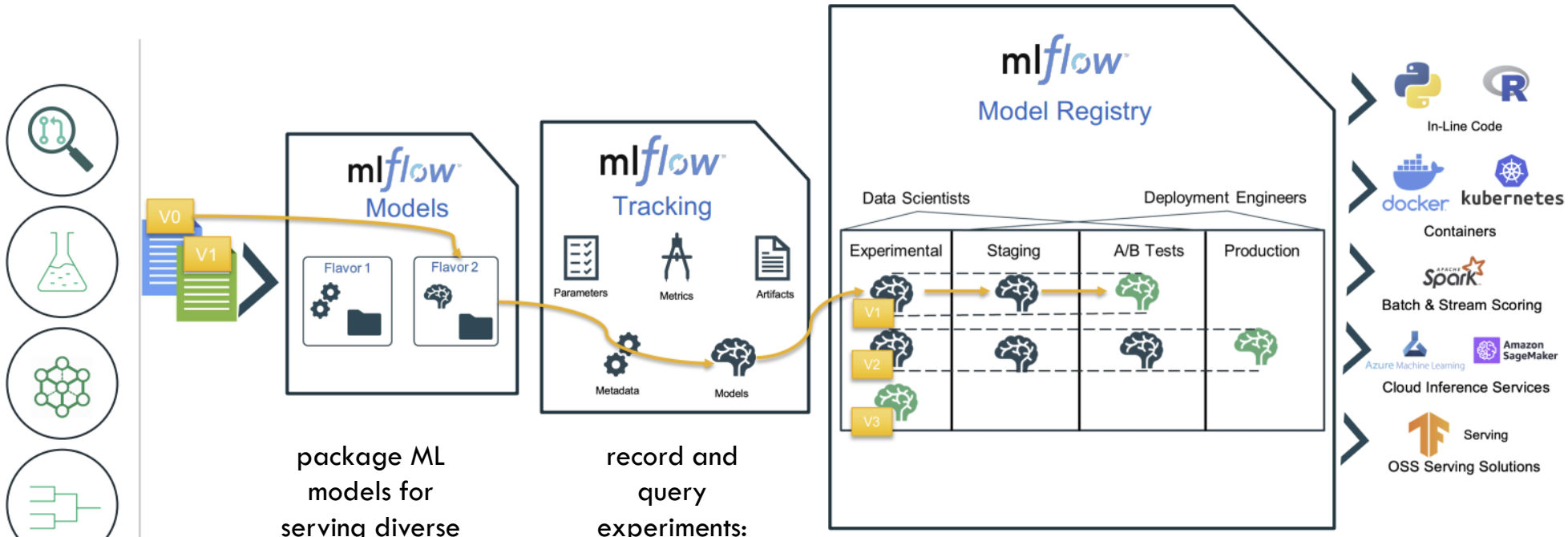
ML DEVELOPMENT

Tracking ML Jobs (Experiment Tracking)

- Systematically tracking and documenting different experimental setups, as well as the configurations, code, parameters, data inputs and outputs, logs, returned metrics, and various artifacts (datasets, models, charts, and etc.)
- **ML Governance**: supports traceability and explainability of models
- Solutions: TensorBoard (TensorFlow), MLflow, Weights & Biases, Comet, ClearML

ML DEVELOPMENT

MLflow: open source platform (Databricks) to manage the ML lifecycle, including experimentation, reproducibility, deployment, and a **central model registry**

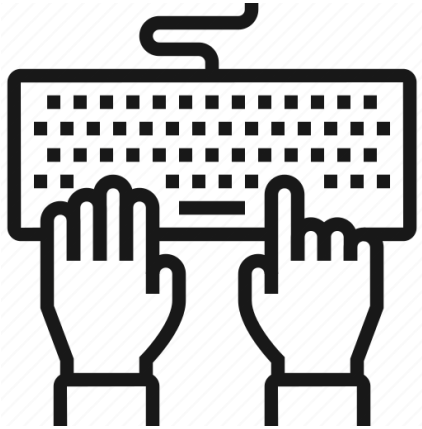


package ML models for serving diverse environments

record and query experiments: code, data, config, results

store, annotate, discover, and manage models in a central repository

ML DEVELOPMENT



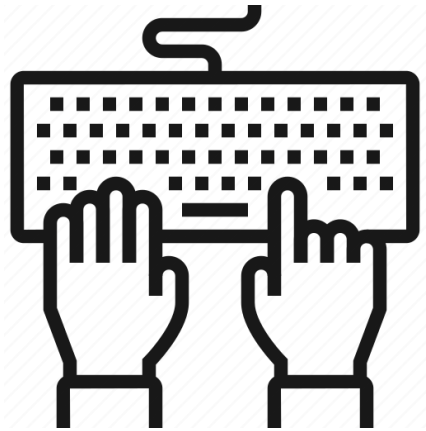
Explore MLflow Experiment Tracking Quickstart

Additional info on:

Search Runs Programmatically

MLflow Model Registry

ML DEVELOPMENT



Auto-Logging

- automatically capture key metrics, parameters, and metadata during machine learning processes **without manual** calls to the experiment tracking/logging

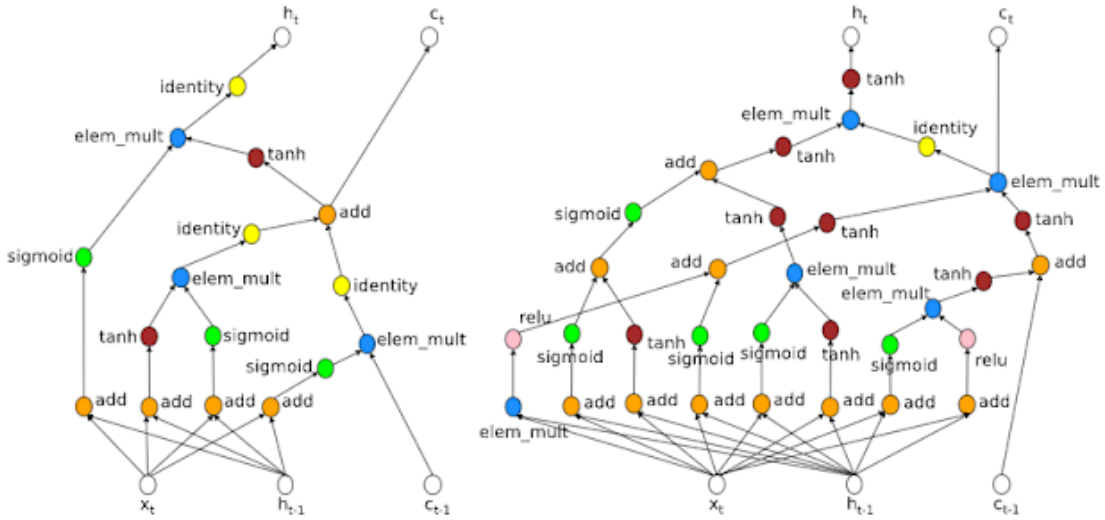
Explore Automatic Logging with MLflow Tracking

ML DEVELOPMENT

AutoML

- Automates model building: discovers architectures that build the best model
- Automates data prep, hyper-param tuning
- Easy-to-use, low-code user experience
- Computationally intensive

Simpler architecture on the left was designed by a human and the more complicated architecture on the right was designed by a neural net

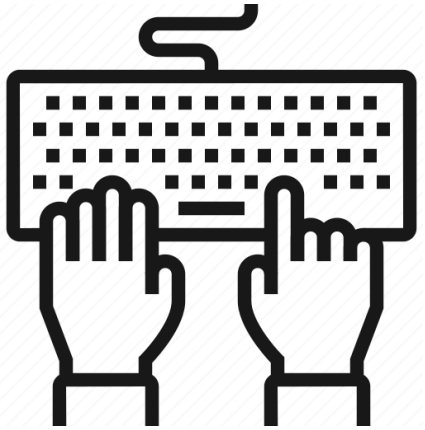


"What may happen is that AutoML and generative AI are combined to create sophisticated ML systems that require very little manual human interaction."

ML DEVELOPMENT

Ludwig: a low-code framework (by Uber) for building custom AI models and other deep neural networks

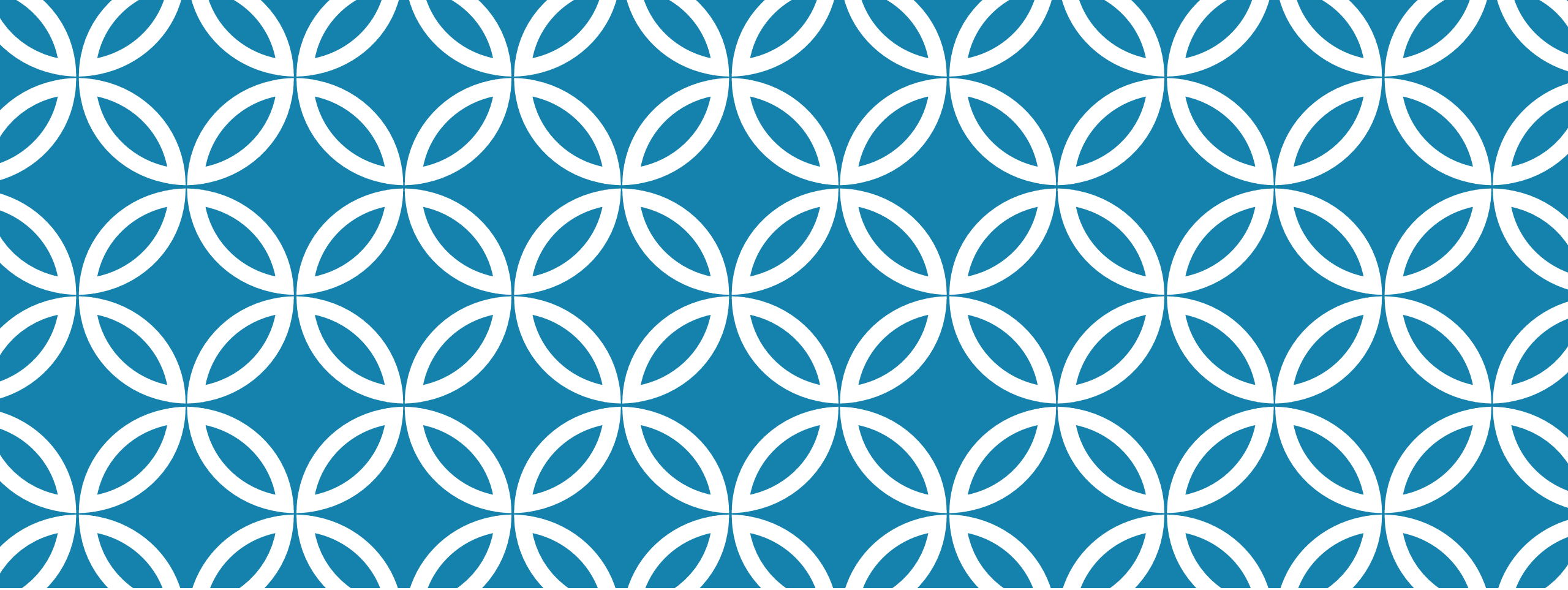
Explore the Tabular Data Classification (Ludwig Python API)



Additional AutoML Solutions:

Datarobot, Google Cloud AutoML, Azure Machine Learning Studio AutoML, SageMaker Autopilot

Auto-sklearn, Auto-Keras, Tree-based Pipeline Optimization Tool (TPOT), MLBox, AutoGluon, AutoWEKA



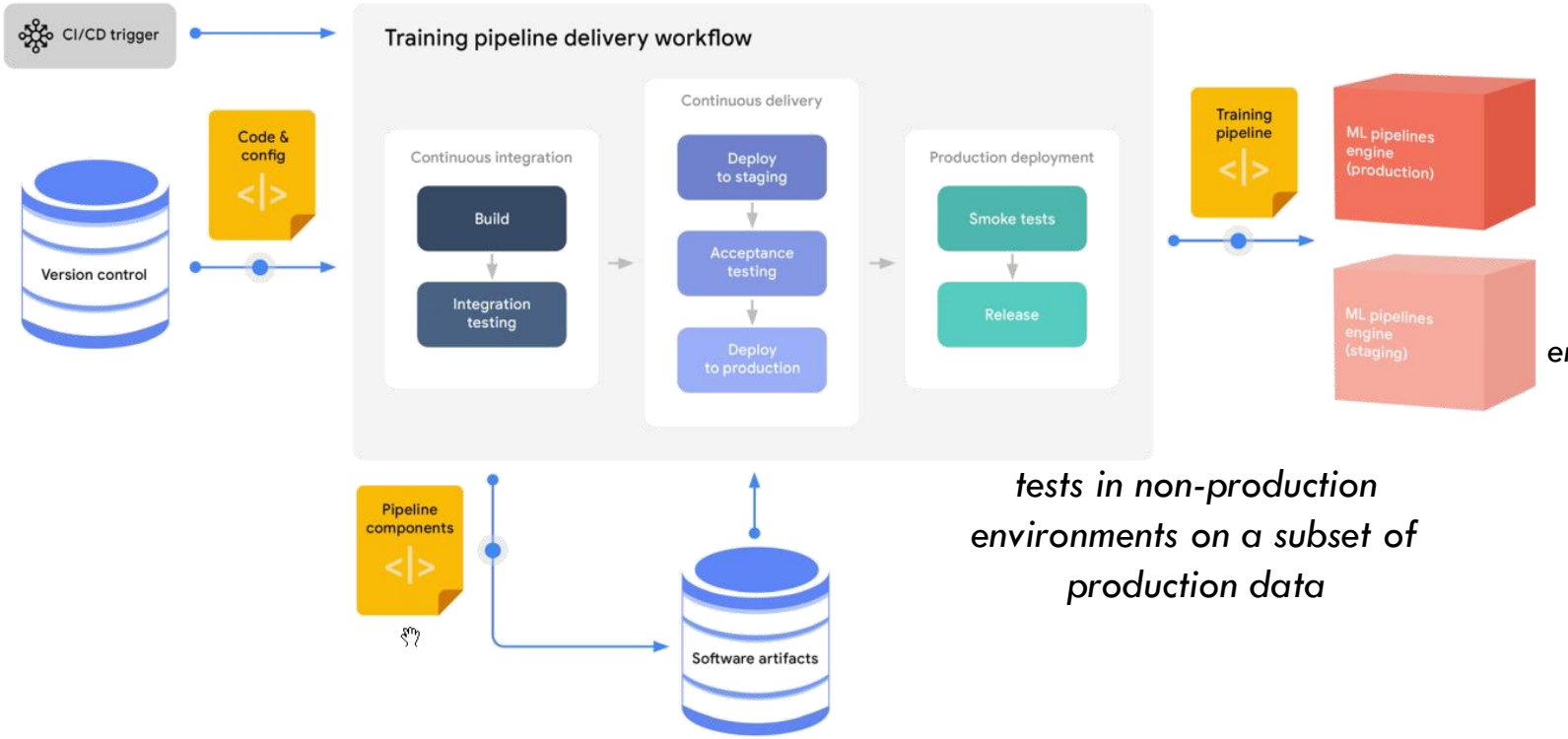
CONTINUOUS TRAINING

Part III: MLOps

CONTINUOUS TRAINING

ML models retrained on a regular basis (e.g., new data, code changes, drifts, etc.)

Training operationalization: the process of building and testing a repeatable ML training pipeline and then deploying it to a target execution environment

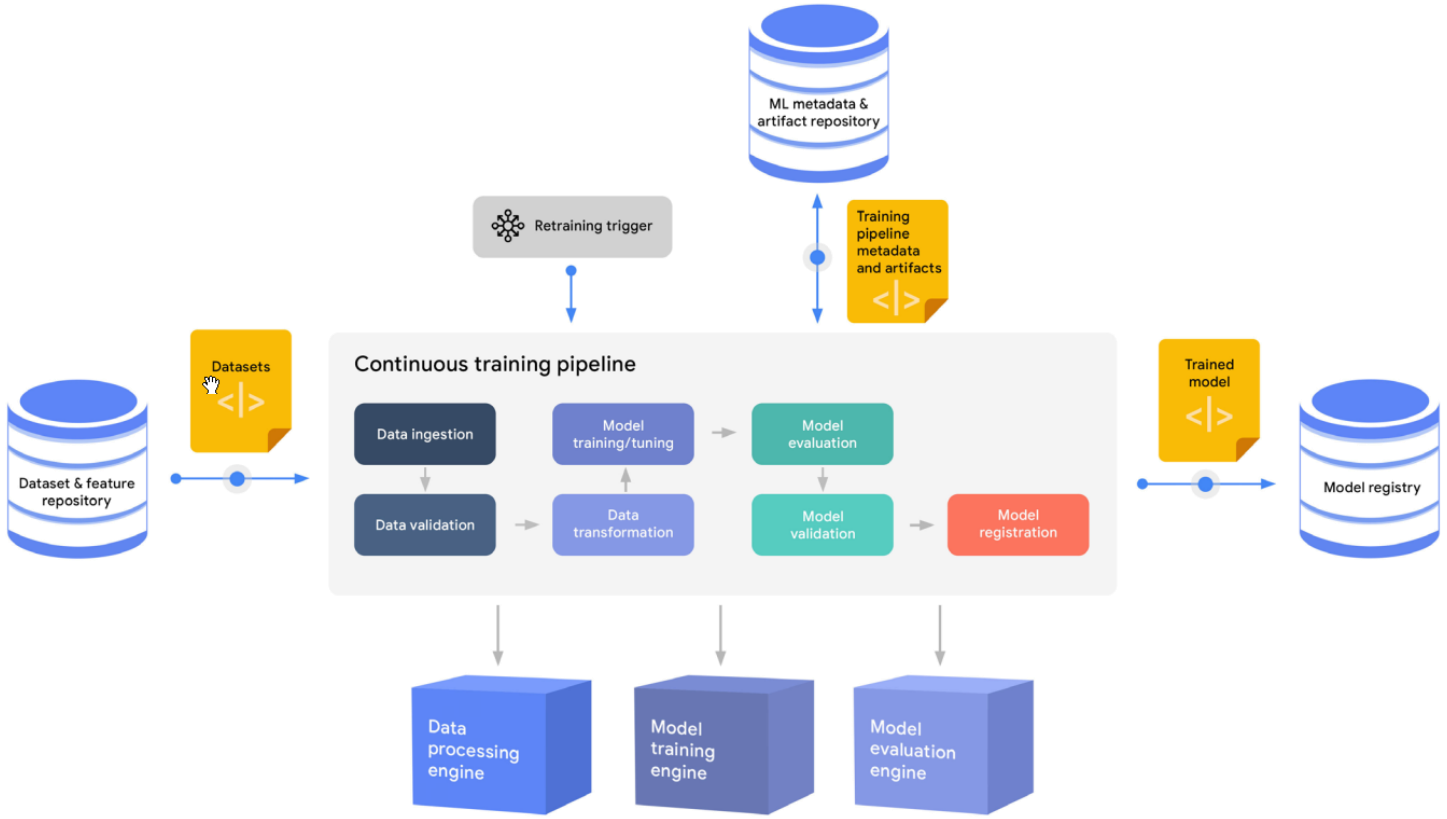


At least one testing environment before production

tests in non-production environments on a subset of production data

CONTINUOUS TRAINING

Continuous training process: automating training pipelines



CONTINUOUS TRAINING

Questions	Answers
When should a model be retrained?	– Periodic training – Performance-based trigger – Trigger based on Data changes – Retraining on demand
How much data is needed for retraining?	– Fixed window – Dynamic window – Representative Subsample selection
What should be retrained?	– Continual learning vs Transfer learning – Offline(batch) Vs Online(Incremental)

https://neptune.ai/blog/retraining-model-during-deployment-continuous-training-continuous-testing

CONTINUOUS TRAINING

Re-training frequency depends on use-case, business value, and costs

Scheduled runs: based on configured jobs (e.g., cron-like)

Event-driven runs: e.g., new data becomes available above a certain threshold, or when model decay is detected by the continuous monitoring

Ad hoc runs: based on manual invocation

CONTINUOUS TRAINING

- Runs of an automated pipeline are unattended
- Data processing and training procedures may become sub-optimal or invalid
- **Data validation** and **model validation** are **gatekeepers** for continuous training
- Validated model **candidates** become part of the **model registry**

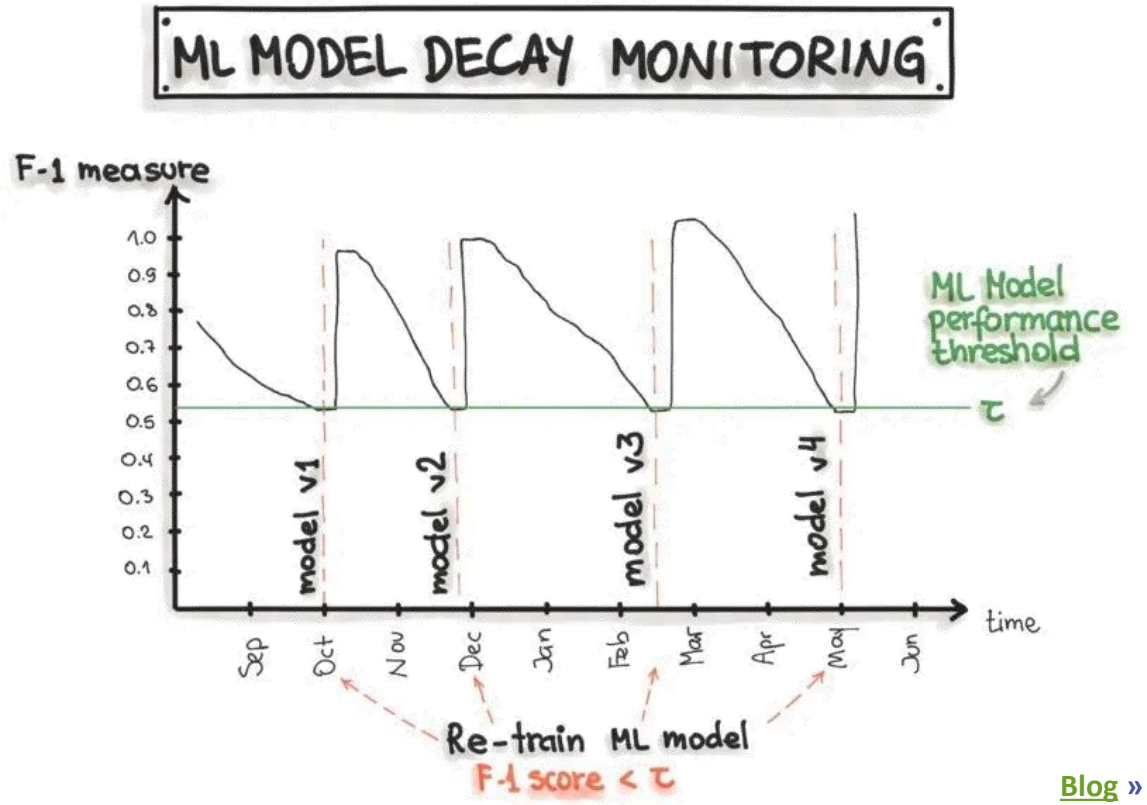
CONTINUOUS TRAINING

How Often to Update Your Models?

- Value of **data freshness**: estimating performance after updating
  - Simulation: training on the data from different time windows in the past and evaluating it on the data from today to see how the performance changes
- **Data iteration** versus **model iteration**:
  - trade-off between updating the model with new data vs. the model with new features, architecture, and hyperparameters
- **Tests in production**: detect changing data distribution and update the model

CONTINUOUS TRAINING

Drawback: it may take time until new groundtruth becomes available

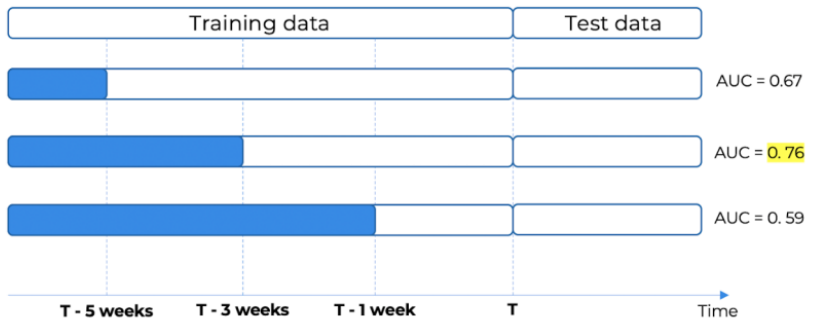


Blog » MLOps

CONTINUOUS TRAINING

How much data is needed for retraining?

- **Fixed window size:**
  - selects a constant period of historical data (usually, starting from most recent)
  - avoids the problem of ever-increasing training data
  - window size trade-off: too small (underfitting) vs. too large (training time and noise/irrelevance)
  - problematic when data changes patterns very dynamically
- **Dynamic window size:**
  - window size becomes a tunable hyperparameter (increased training time)
- **Representative subsample selection:**
  - select training data similar to production data
  - exclude data indicating drift



CONTINUOUS TRAINING

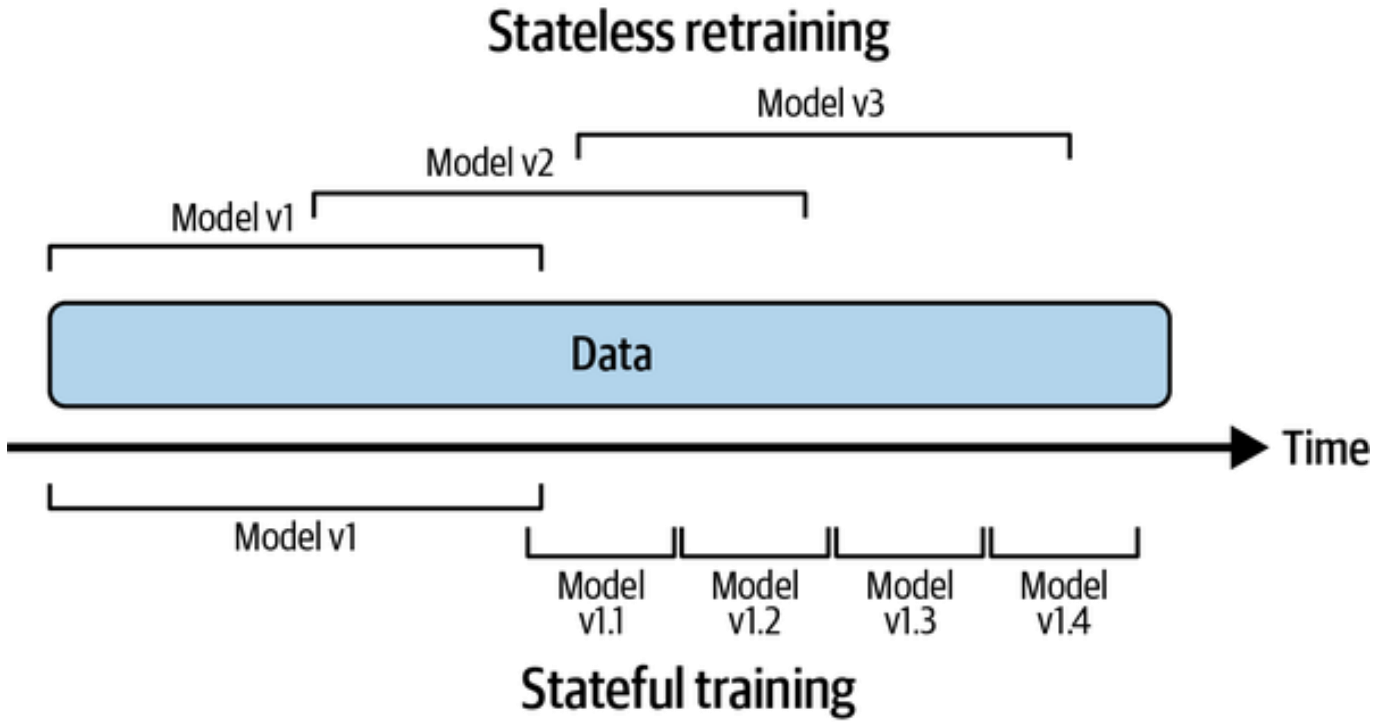
What should be retrained?

- **Offline (batch) learning:**
  - re-builds each time a new model from scratch with the whole (newly selected) training set
  - after deployment, the model does not learn until the next re-training
  - simple but ignores previous knowledge from past trainings and requires large training time
- **Online (incremental) learning:**
  - retrains the model incrementally by passing data sequentially as they come in
  - cost-effective (out-of-core learning) for streaming data with dynamic rhythm
  - retains and updates past knowledge
  - only some algorithms support incremental training, trade-off for the size of mini-batch
  - may become hard to reduce the impact of older irrelevant training instances
  - need to adapt to drifts

CONTINUOUS TRAINING

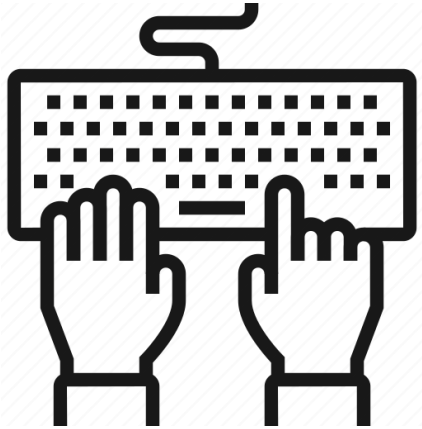
Stateless vs. Stateful Training

the model is trained from scratch each time



the model continues training on new data

CONTINUOUS TRAINING



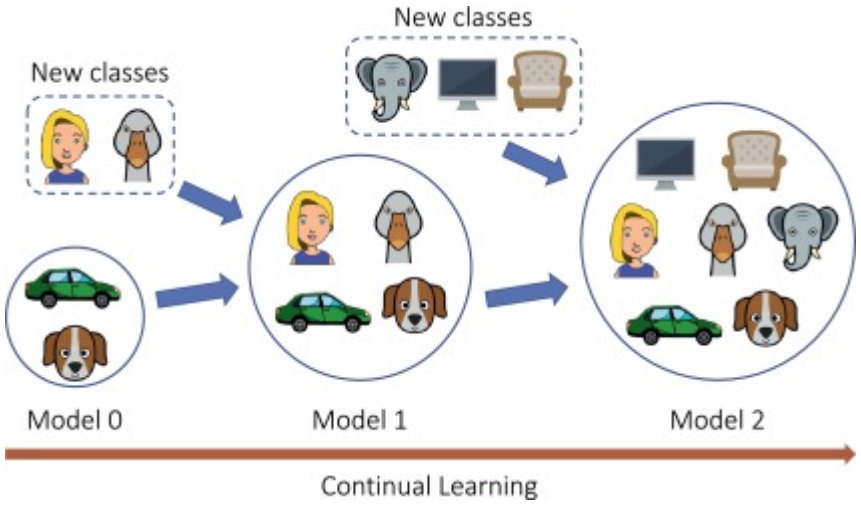
Explore the quickstart river

Optional: explore the impact of concept drift

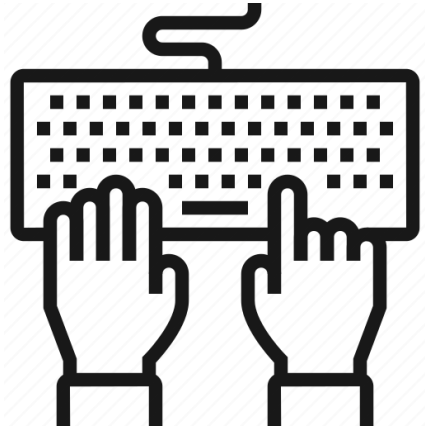
CONTINUOUS TRAINING

Special topic: Continual Learning

- model learns continually from a stream of data
- single model on **changing data distributions**
- challenge: avoid catastrophic forgetting, tendency to forget previously learned information upon learning new information

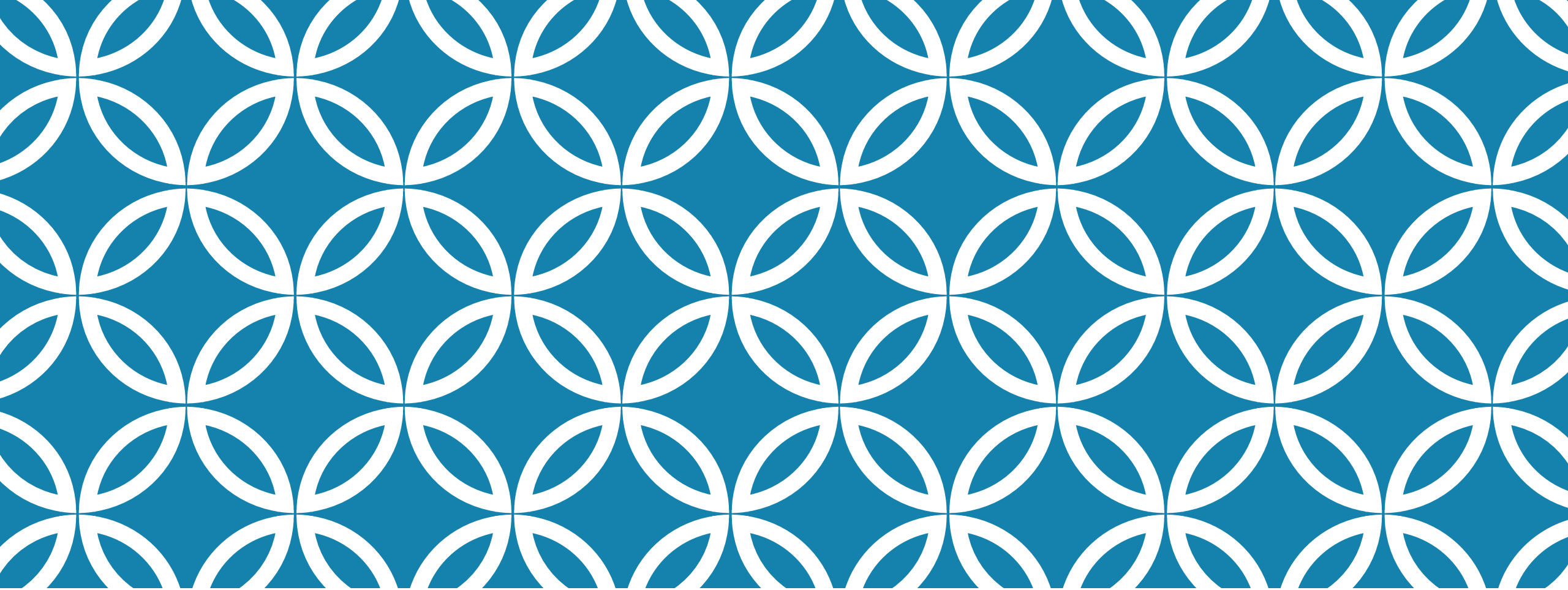


CONTINUOUS TRAINING



Continuum: Explore **Quick Example**

(Also available as Colab Notebook)

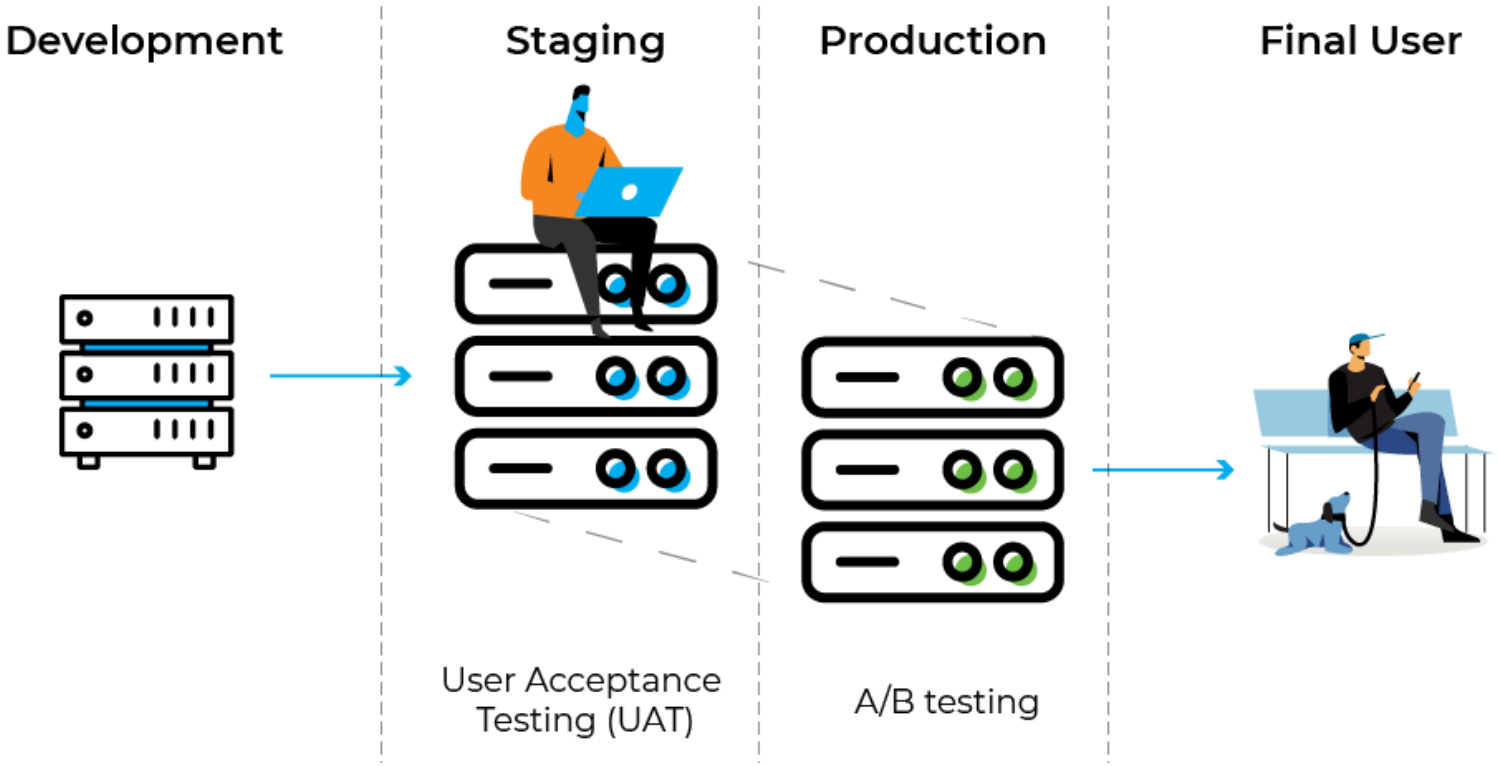


MODEL DEPLOYMENT

Part III: MLOps

MODEL DEPLOYMENT

Making an ML model running and **accessible**



Challenges:

available to many users

latency of milliseconds

99% uptime

information paths are set

seamlessly deploying updates/fixes

X 100's of models

X multiple updates per day/week

MODEL DEPLOYMENT

Deployment responsibility

- models exported and handed off to another team to deploy them vs.
- models deployed by the same people who developed them (main reason for MLOps)

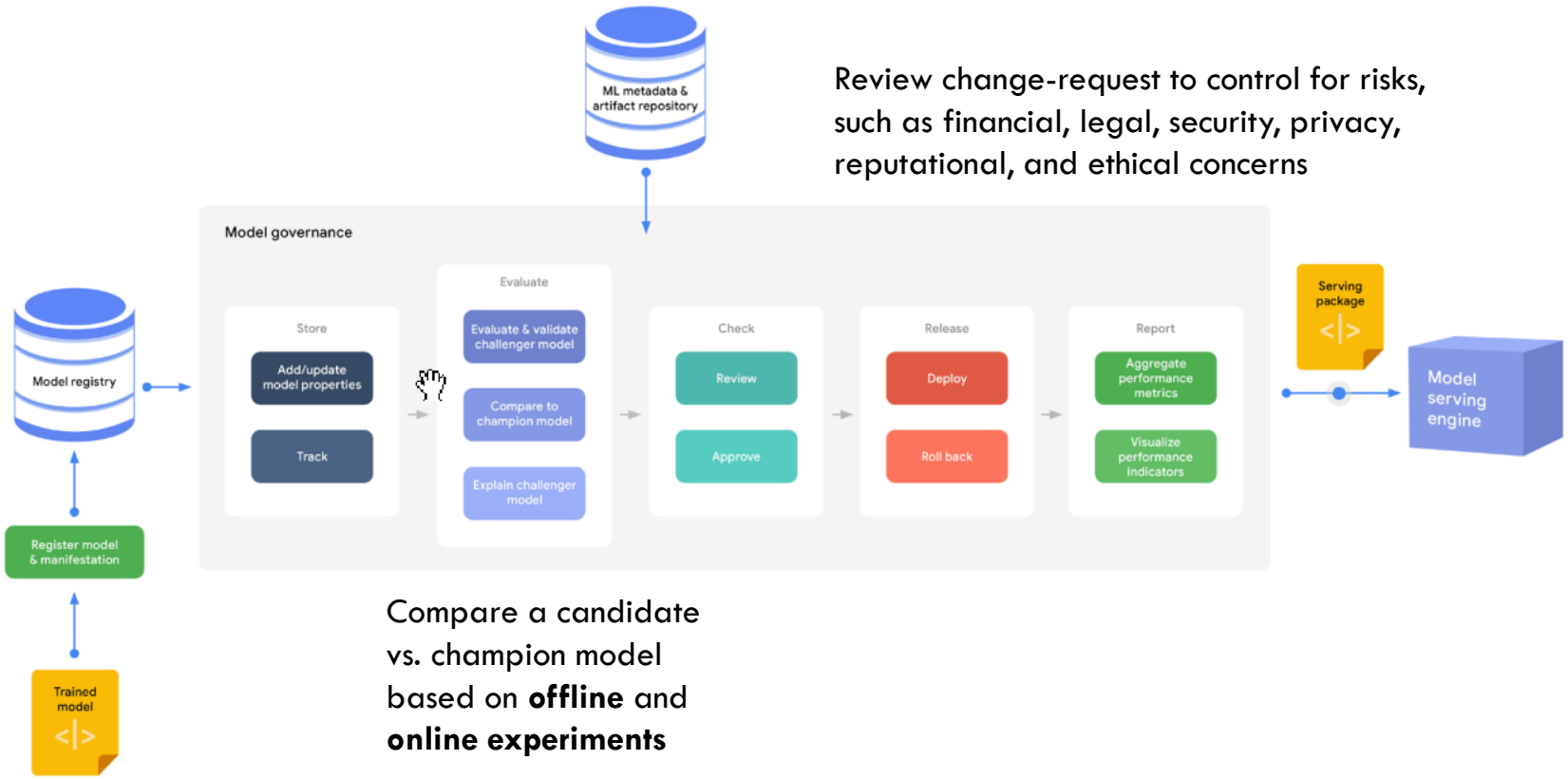
Why no separation of responsibility?

- high overhead in communications
- slow update times
- hard to support

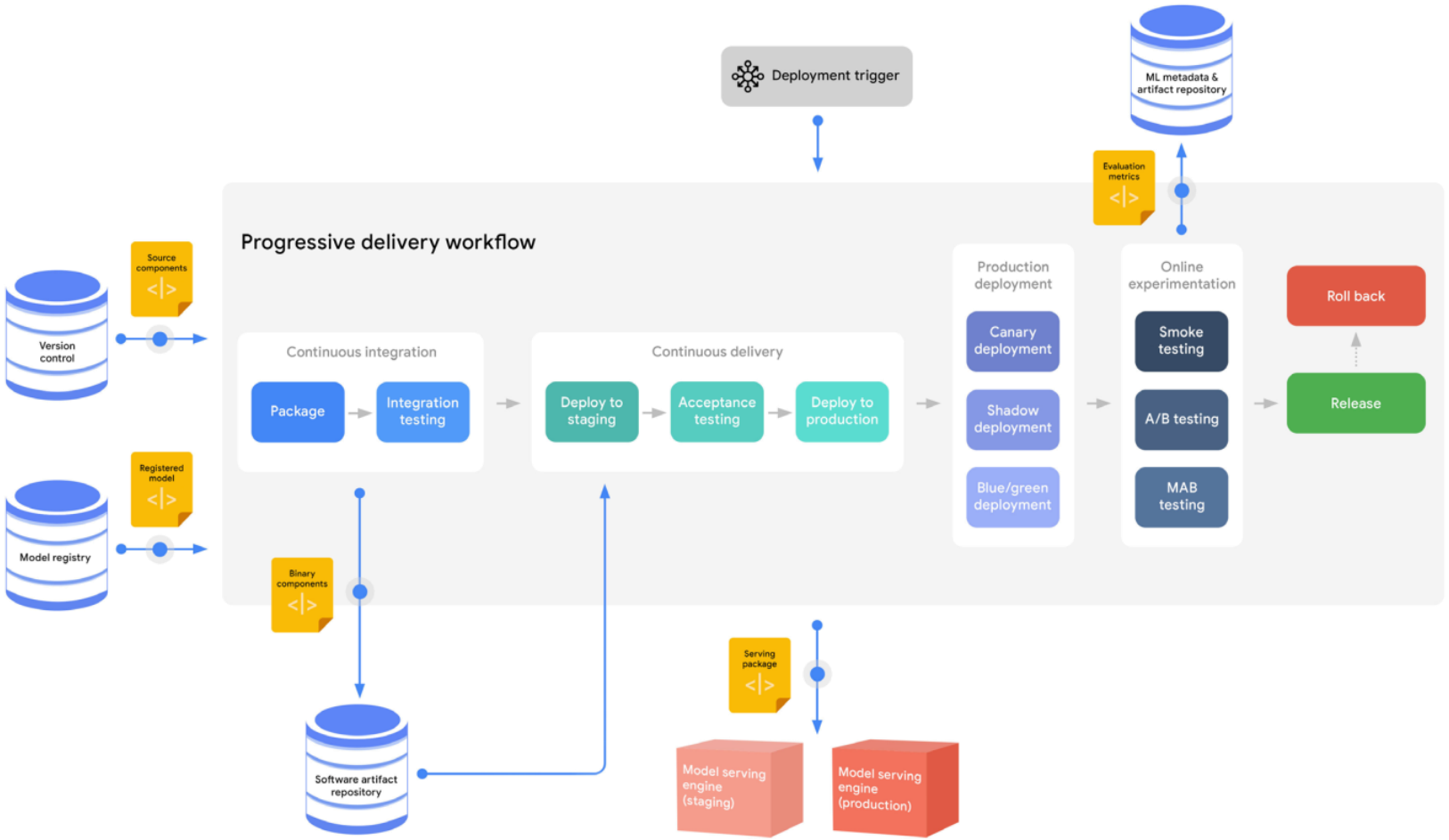
MODEL DEPLOYMENT

Model Governance

- registering, reviewing, validating, and approving models for deployment



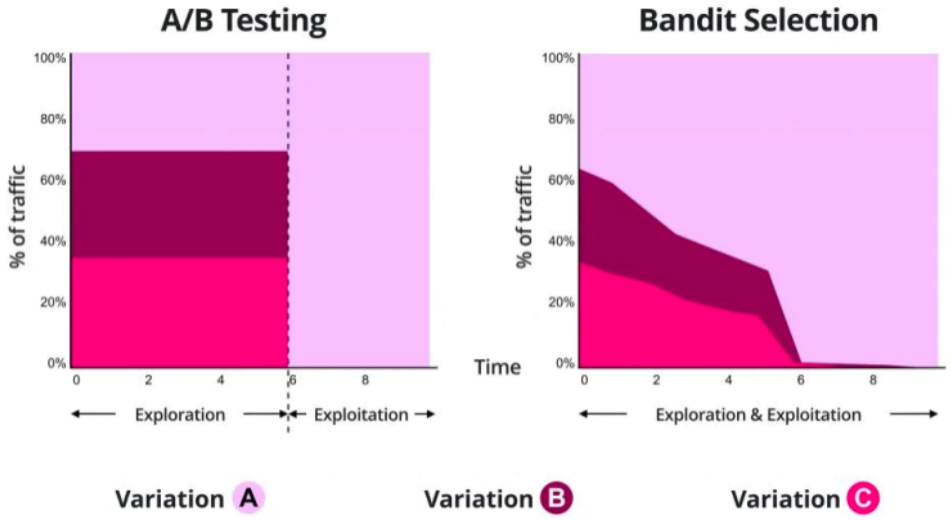
MODEL DEPLOYMENT



MODEL DEPLOYMENT

Online experimentation methods

- deciding whether a candidate model should replace the production version
- A/B testing and multi-armed bandits (MAB)
- Interleaving experiments: users see both predictions



MODEL DEPLOYMENT

Deployment strategies

- **Recreate:** Version A is terminated then version B is rolled out
- **Blue/Green:** Version B released alongside version A, traffic is switched to version B
- **Shadow:** Version B receives real-world traffic alongside version A but doesn't impact the response (offline experimentation)
- **Canary:** Version B is released to a subset of users, then proceed to a full rollout
- **Ramped** (aka rolling-update): Version B is slowly rolled out and replacing version A

MODEL DEPLOYMENT

Progressive deployment strategies

- **Canary** and **ramped** (rolling-update) support online experimentation
- Candidate model does not immediately replace the production version, it **runs in parallel**
- Only a **fraction of users** are exposed to candidate model

Ensure affinity: the same user is always routed to the same model

MODEL DEPLOYMENT

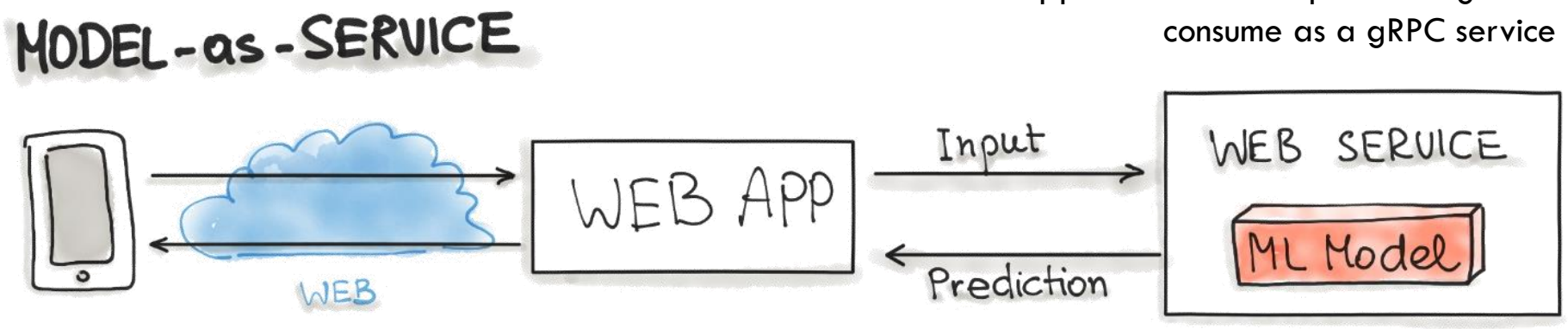
Model Deployment Patterns

- Package and deploy models as microservices (containers)
- Access them through an API
- Allow for independent scaling of the model (add/remove containers)

MODEL DEPLOYMENT

Deployment pattern: Model-as-Service

Wraps the model within a dedicated web service that applications can request through a REST API or consume as a gRPC service



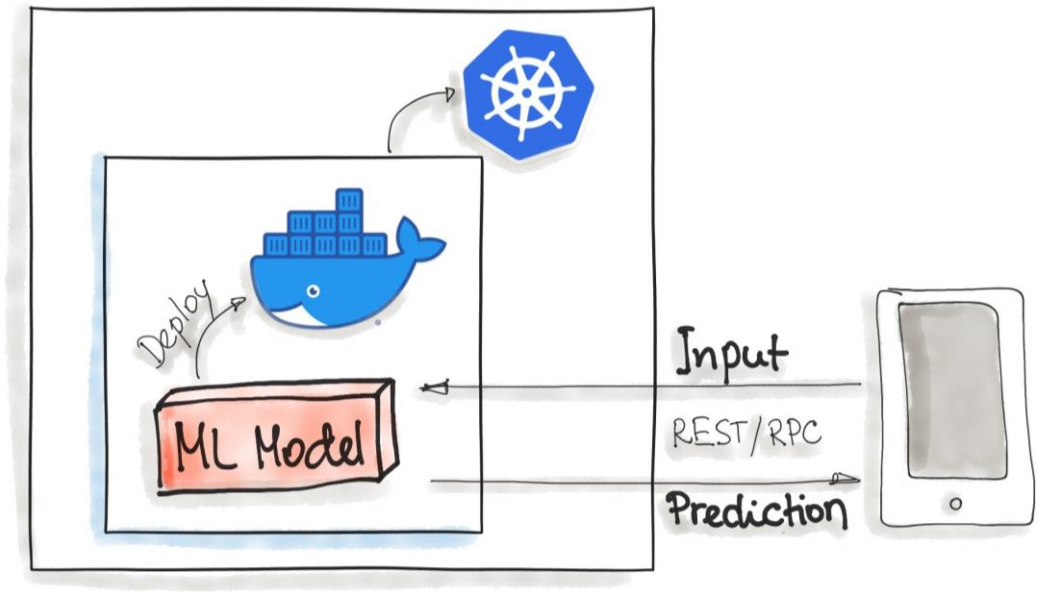
MODEL DEPLOYMENT

Wrapping trained models as deployable services: Docker Containers

Kubernetes (or AWS Fargate) for container orchestration

Deploy a container that wraps an ML model inference code

Docker as de-facto standard for on-premise, cloud, or hybrid deployments



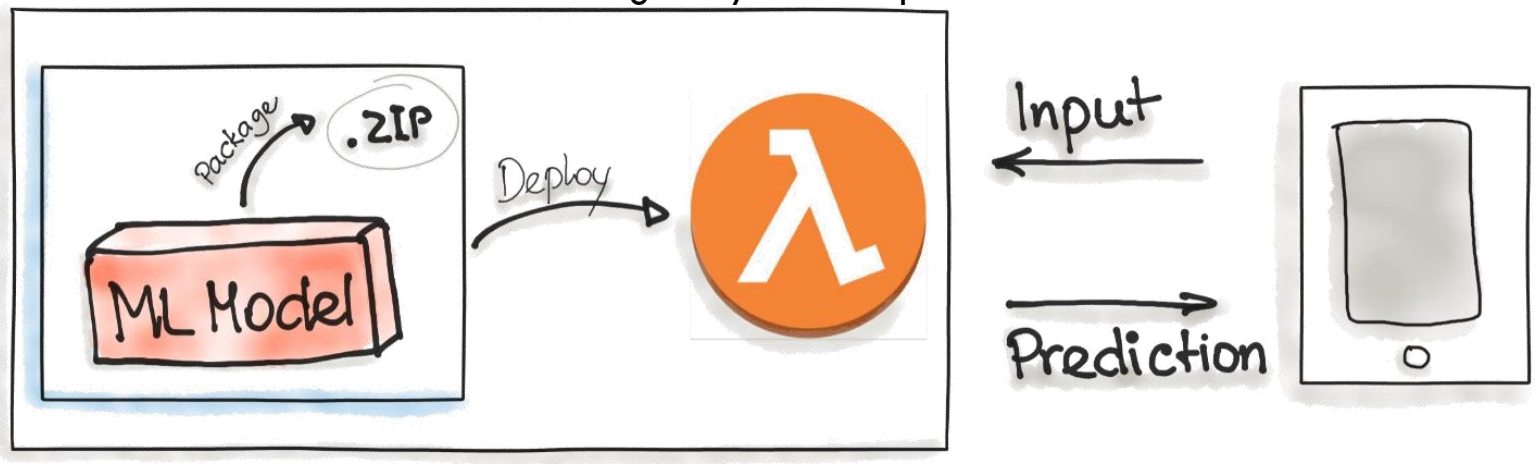
Prediction available through a REST API (e.g. Flask or FastAPI)

MODEL DEPLOYMENT

Wrapping trained models as deployable services: Serverless Functions

application code and dependencies are packaged into .zip files, with a single entry point function

function then managed by a cloud provider



possible constraints such as the size of the deployed artifact, RAM and availability of accelerators.

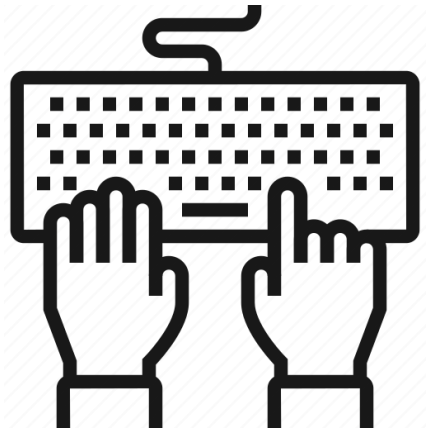
Examples: Amazon AWS Sagemaker, Google Cloud AI Platform, Azure Machine Learning Studio, and IBM Watson Machine Learning

MODEL DEPLOYMENT

Deployment services

- Getting help in pushing models and dependencies to production and exposing models as endpoints
- Major cloud services: AWS SageMaker, GCP Vertex AI, Azure ML, Alibaba Machine Learning Studio
- Other Tools: MLflow Models, Seldon, Fast API
- Important criterion: support for test in production

MODEL DEPLOYMENT



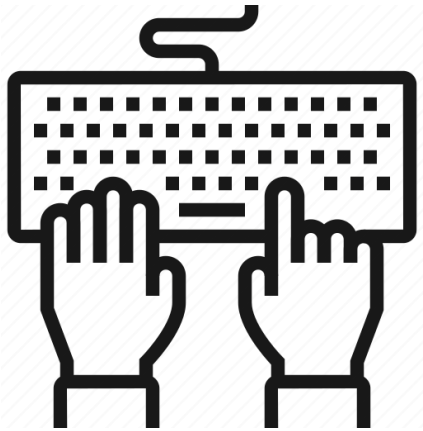
MLflow packages models and dependencies and create either:

- a virtual environment for **local deployment** or
- a **Docker container** image containing everything needed to run your model

MLflow launches an inference server with **REST endpoints** (e.g., with Flask or Seldon)

Explore Deploy a Model Locally

MODEL DEPLOYMENT



FastAPI is a high-performance web framework for building APIs with Python based on standard Python type hints.

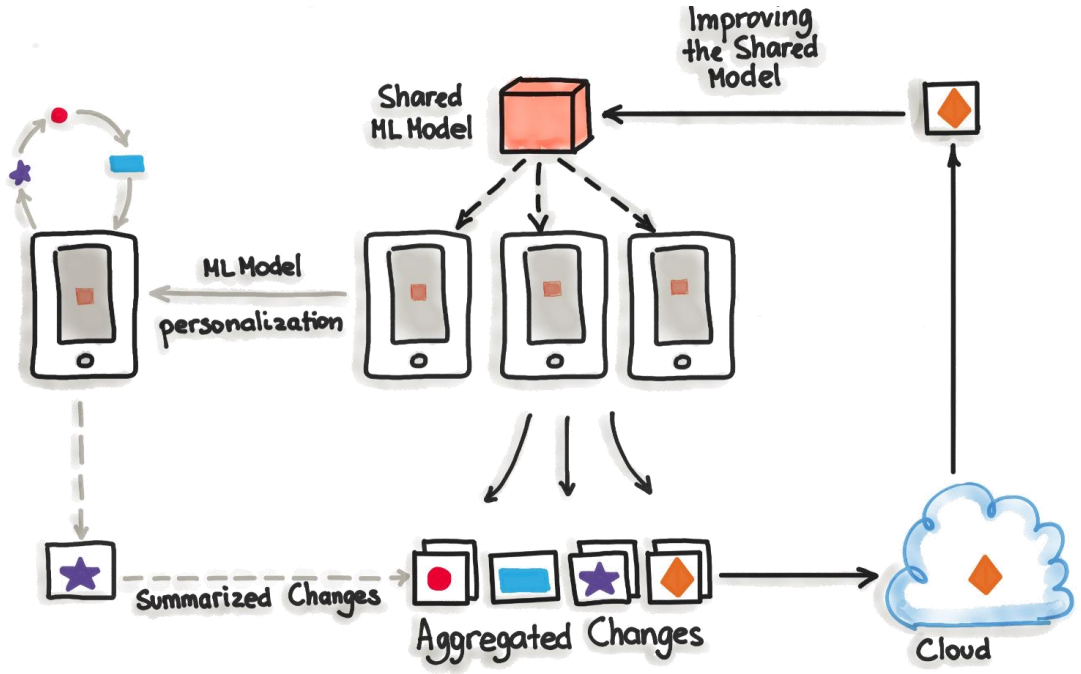
It can be used to deploy a model:

Explore First Steps and try to deploy a model of your choice as API.

MODEL DEPLOYMENT

Deploying pattern: Federated Learning

Multiple entities **collaboratively** train a model while ensuring that their data remain **decentralized** (**privacy**)



MODEL DEPLOYMENT

Deployment on edge devices

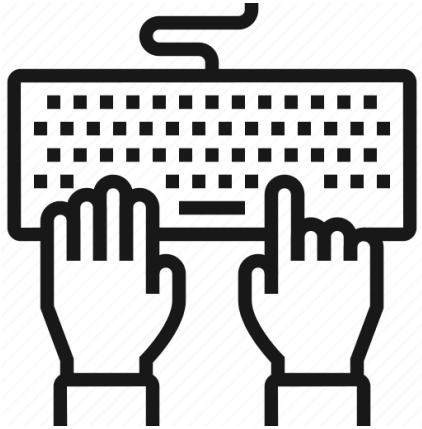
- browsers, phones, laptops, smartwatches, cars, security cameras, robots, embedded devices, FPGAs (field programmable gate arrays), and ASICs (application-specific integrated circuits)
- edge app run where cloud computing cannot (unreliable, high costs, privacy)
- edge devices require **energy efficiency** and have constrained resources (RAM, etc.)

MODEL DEPLOYMENT

Deployment on edge devices

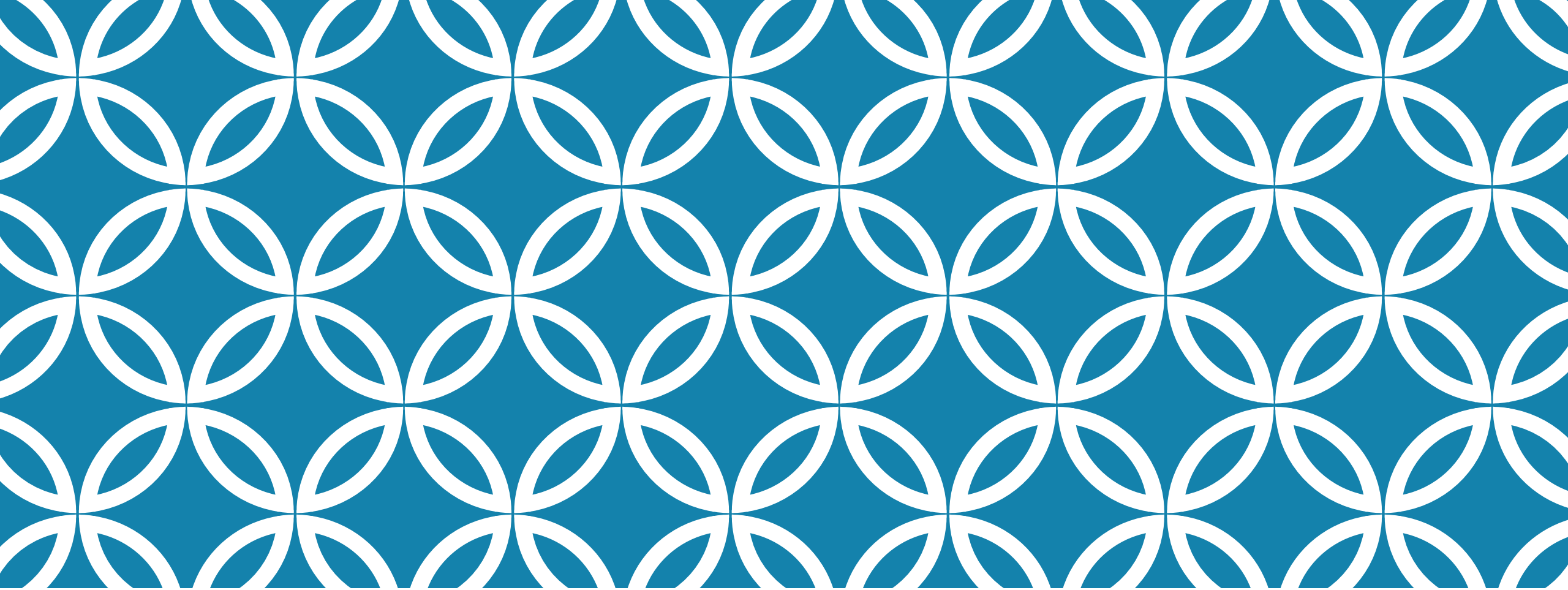
- Model Optimization: automatically performance improvement of ML frameworks through graph simplifications & high-level optimizations for accelerators (e.g., Grappler for TF)
- ML in Browsers: deploy ML on any device that supports browsers (tablet, smartphone, etc.), by compiling ML models into JavaScript (e.g., TensorFlow.js)
  - WebAssembly (WASM) backend

MODEL DEPLOYMENT



Optional Exercise: Deployment for On-Device Training with TensorFlow Lite

Optional Exercise: deployment with TensorFlow.js — Handwritten digit recognition with CNNs

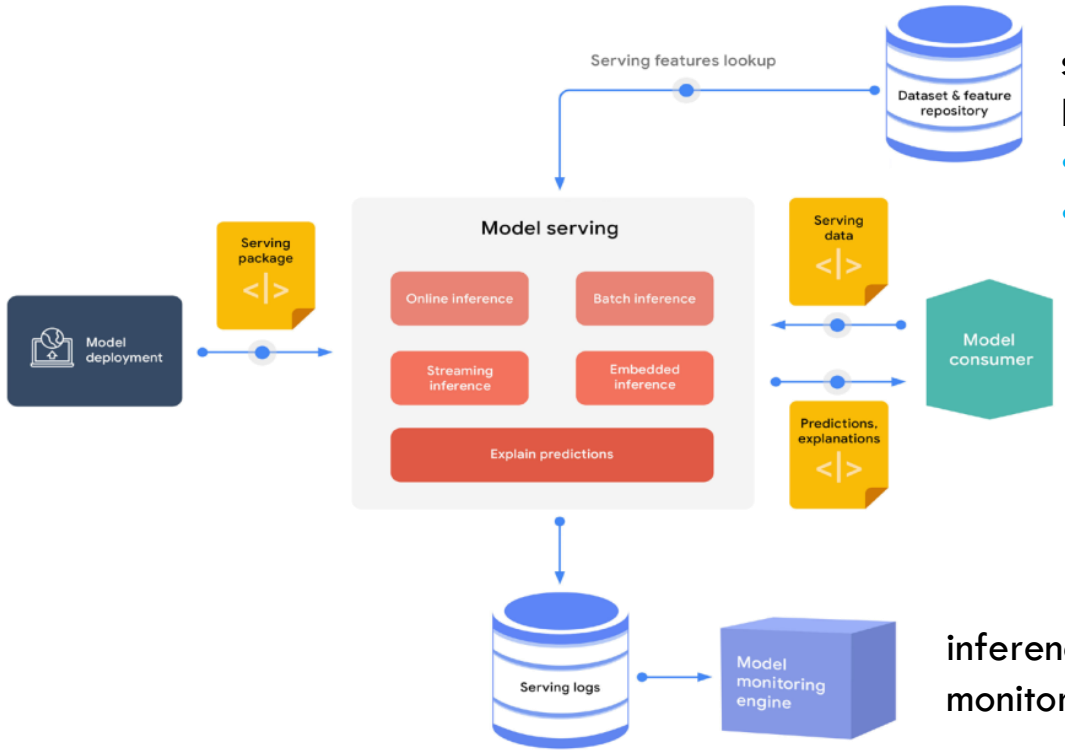


PREDICTION SERVING

Part III: MLOps

PREDICTION SERVING

Prediction serving process: a deployed model serves prediction requests



serving engine gathers user context and looks up feature values in feature store:

- batch features based on historical data
- online features based on online data

inference logs stored for continuous monitoring and analysis

PREDICTION SERVING

Considerations about predictions based on RESTful API:

- **validate** the **request**: verify permissions, etc.
- gather the **context**: information in request to fetch features from feature store
- apply the model to fetched features and generate prediction
- **prediction validation**: verifying that prediction makes sense and react otherwise (see next)
- prediction **presentation**: transform prediction into an interpretable form
- optional: prediction **explanation** in form of e.g., feature attributions
- **log** prediction and context (see: **eval store**)

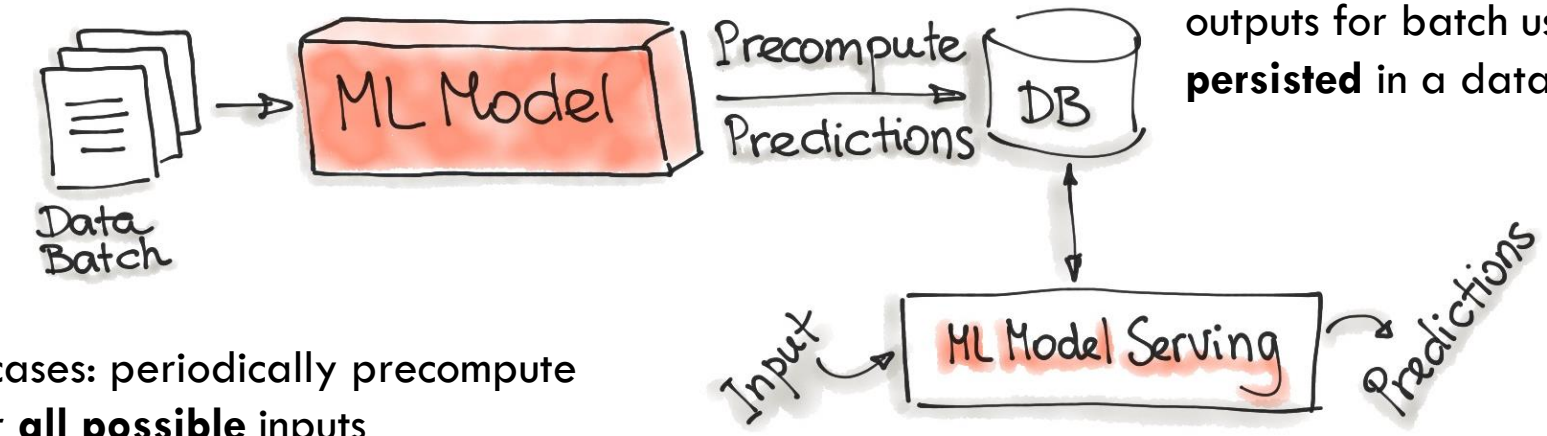
PREDICTION SERVING

Options for dealing with prediction errors during serving

- allow users to **report** prediction errors: log enables fast model debugging
- additional **logging** of user **interactions**:
  - less interactions after errors?
  - which predictions (e.g., recommendations) users tend to ignore?
- block "outlier" predictions, keep **humans in the loop** (send alerts)
- prediction validation based on **confidence**
- develop **fallback predictions** (esp. for costly type of errors)

PREDICTION SERVING

Batch Serving: bulk data scoring for large quantities of input data



outputs for batch usually **persisted** in a database

Example use-cases: periodically precompute predictions for **all possible** inputs

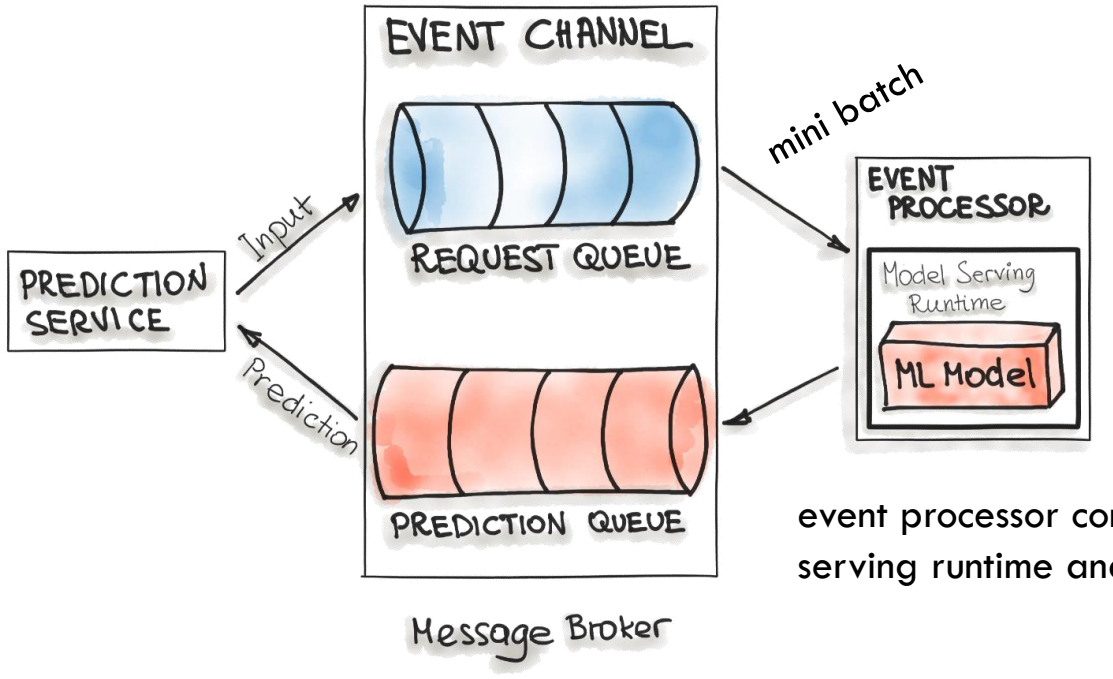
- generate weekly recommendations to all users
- extract named entities from documents of a search engine

serve by **querying** the database to get predictions

PREDICTION SERVING

On-Demand Streaming Serving: near real-time streaming predictions

message-broker (event-driven) architecture



event processor contains the model serving runtime and the ML model

PREDICTION SERVING

Summary/Comparison

Batch serving

- Optimizes throughput
- Asynchronous
- "All possible inputs" may be intractable

On-demand serving

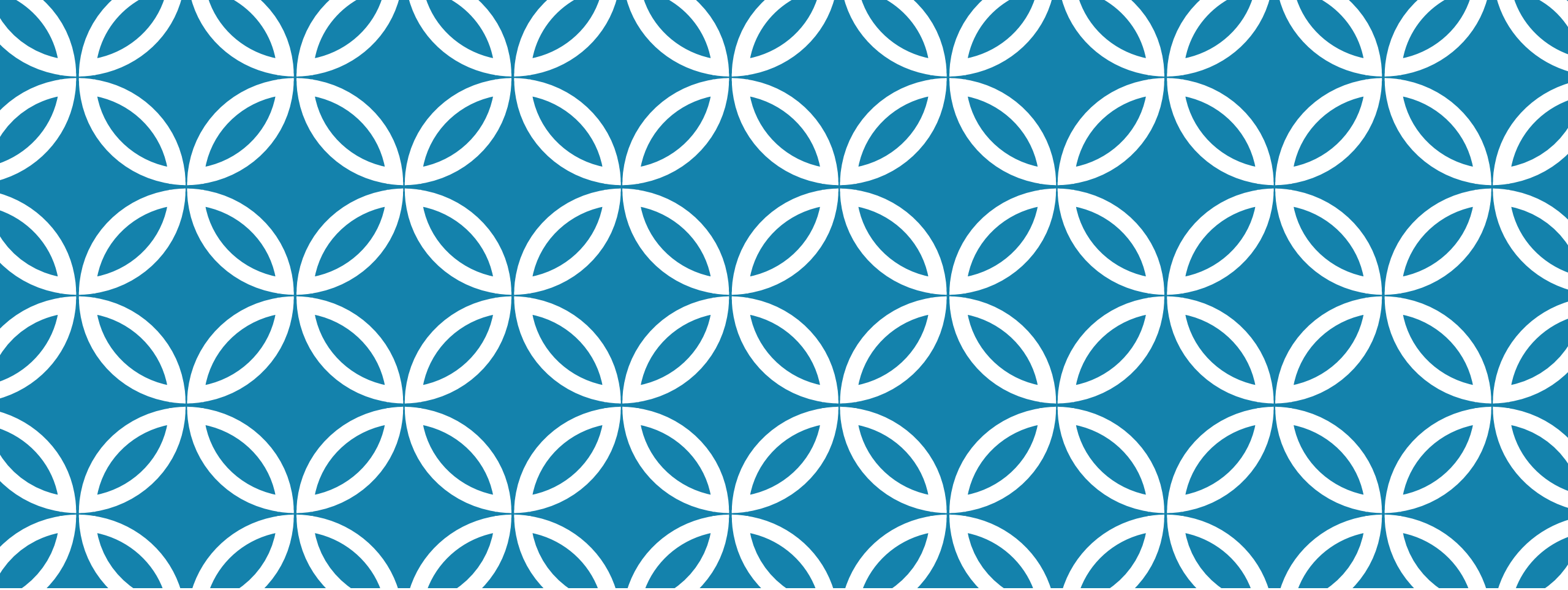
- Optimizes latency
- Synchronous
- Adaption to changing context
- No resources wasted for unnecessary predictions

Hybrids are possible

PREDICTION SERVING

Additional topic: side-effects to consider

- **User-fatigue**: when user experience is excessively interrupted by predictions (when users are unaware of ML components)
- **"Creep factor"**: when user perceives model's predictive capacity as too high and intrusive w.r.t. their privacy



CONTINUOUS MONITORING

Part III: MLOps

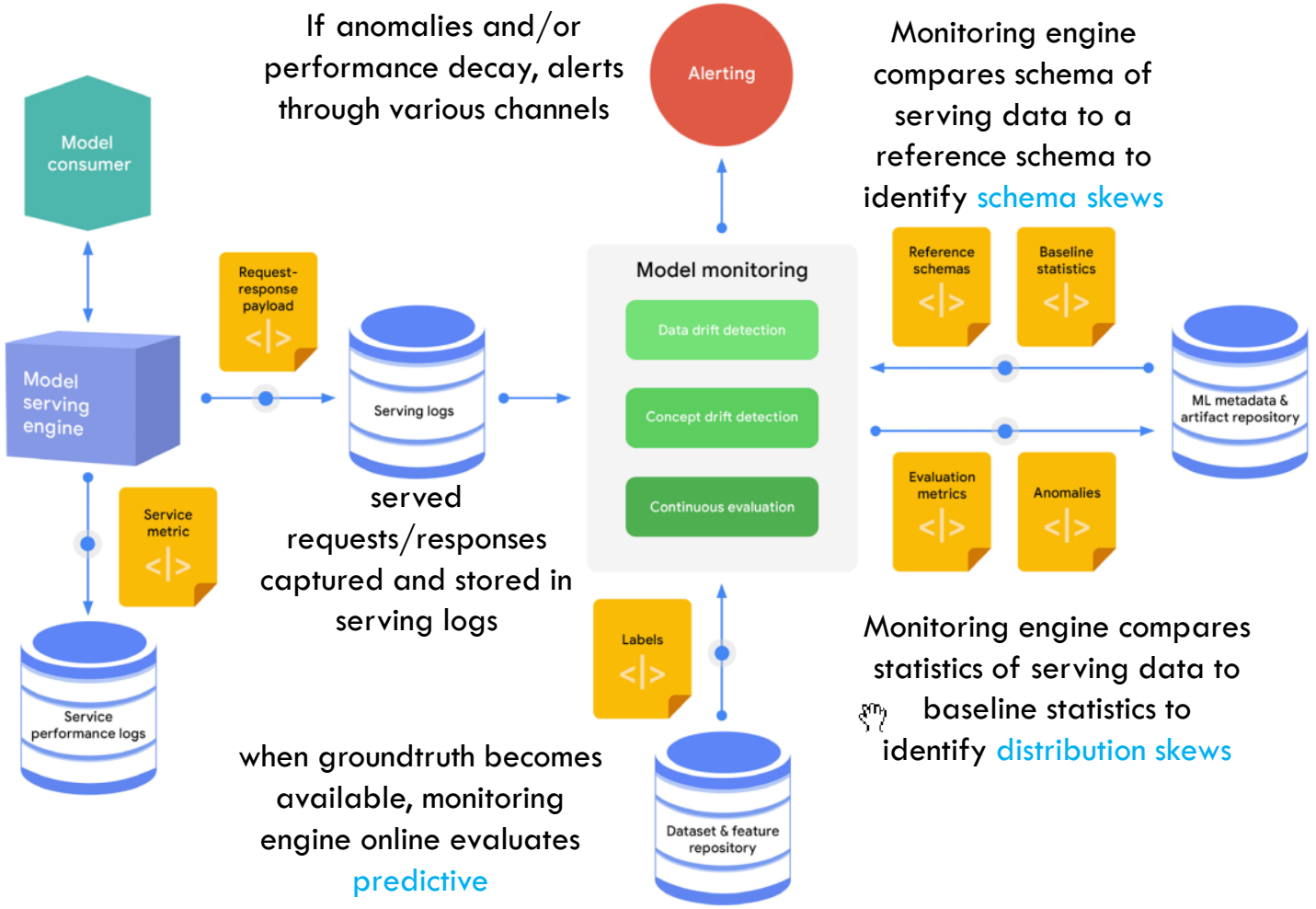
CONTINUOUS MONITORING

The process of monitoring the **effectiveness** and **efficiency** of a model in production

- Verify proactively (early warnings) that performance remains **acceptable**
  - Example: COVID-19 pandemic and model disruption
- What can go wrong with a model proved to work well in development?
  - Failures of operational expectations: service availability, latency, throughput
  - ML-specific failures: decay in business-relevant performance metrics – "**ML fails silently**"

Measure the **Business Impact**:
✓ define **success metrics** aligned with business objectives and
✓ establish a **baseline** according to cost-benefit analysis

CONTINUOUS MONITORING



If anomalies and/or performance decay, alerts through various channels

Monitoring engine compares schema of serving data to a reference schema to identify schema skews

served requests/responses captured and stored in serving logs

predictive effectiveness
+
resource utilization, inference latency, throughput

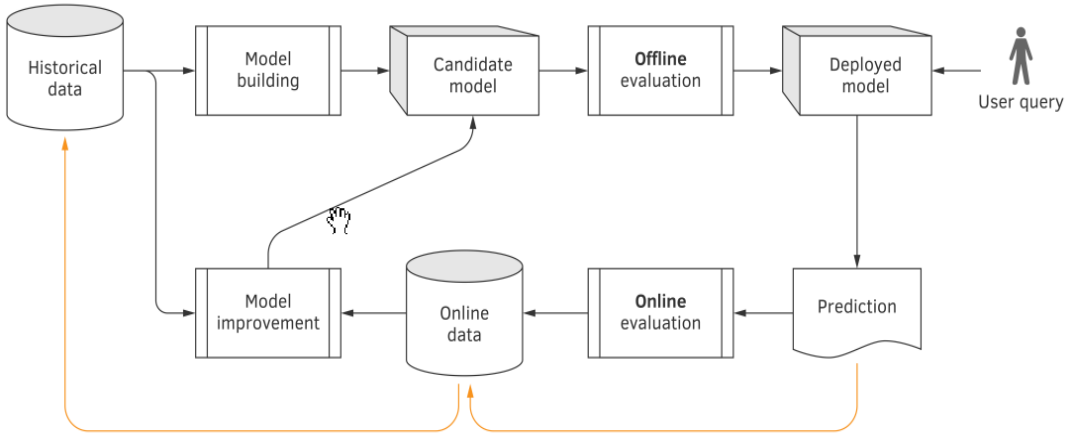
Monitoring engine compares statistics of serving data to baseline statistics to identify distribution skews

when groundtruth becomes available, monitoring engine online evaluates predictive effectiveness

CONTINUOUS MONITORING

Why both offline and online evaluation?

- offline evaluation: **after training**
  - how well the right features, learning algorithm, model, and hyperparameters have been selected
  - based on **historical data**: test/training data are similar
  - estimation of post-deployment performance
- online evaluation: in production
  - how well is the model serving its purpose ***now*** (latency, predicting effectiveness, customer satisfaction) – **not possible** with offline evaluation
  - based on **current data**: detect data distribution shifts
  - ongoing monitoring of post-deployment performance
  - establish an **eval store**: log every generated prediction, its **context** and its assessment (groundtruth may come delayed!)



closing the loop: online data integrated in the offline (historical) data repository

CONTINUOUS MONITORING

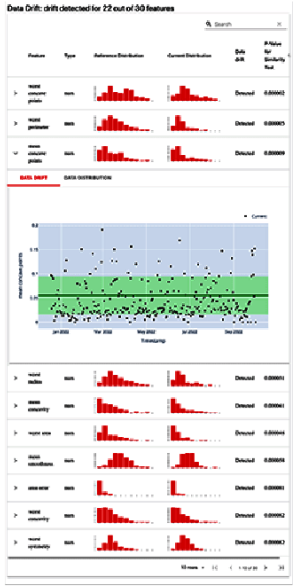
Summary of Monitoring Toolbox (Observability)

- **Logs**
  - need for scalability: number of events grows fast (log management such as Splunk, etc.)
  - distributed tracing: use unique process ID to track over various microservices/parts of pipelines
- **Dashboards**
  - visualization of monitoring (usually supported by observability systems)
  - avoid **cluttering**
- **Alerts**
  - alert policy and **information paths** (who should be contacted, when and how)
  - descriptive (necessary context) and actionable alerts (clear what is to do)
  - avoid "**alert fatigue**": monitoring team stops paying attention to too frequent alerts

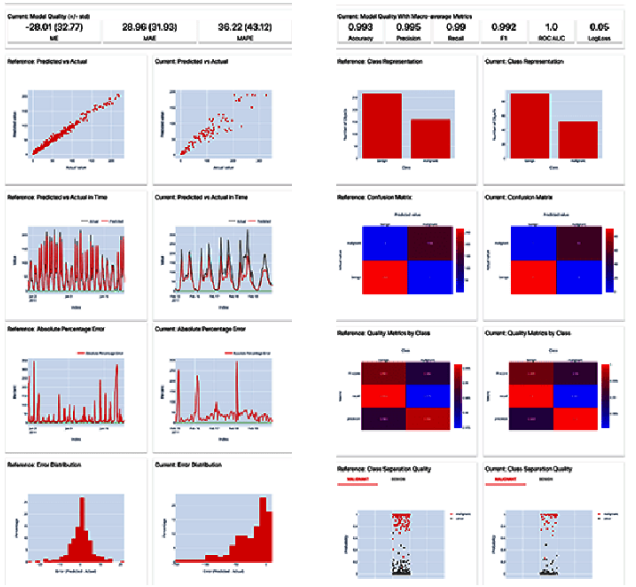
CONTINUOUS MONITORING

- Monitoring typically in **dashboards**
- Monitoring solutions should be mechanisms that enable definition of **conditions, thresholds,** and **actions**

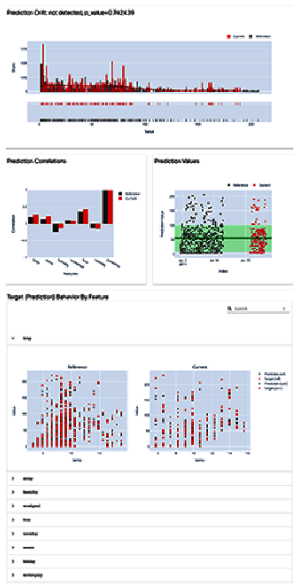
Data drift



Model performance



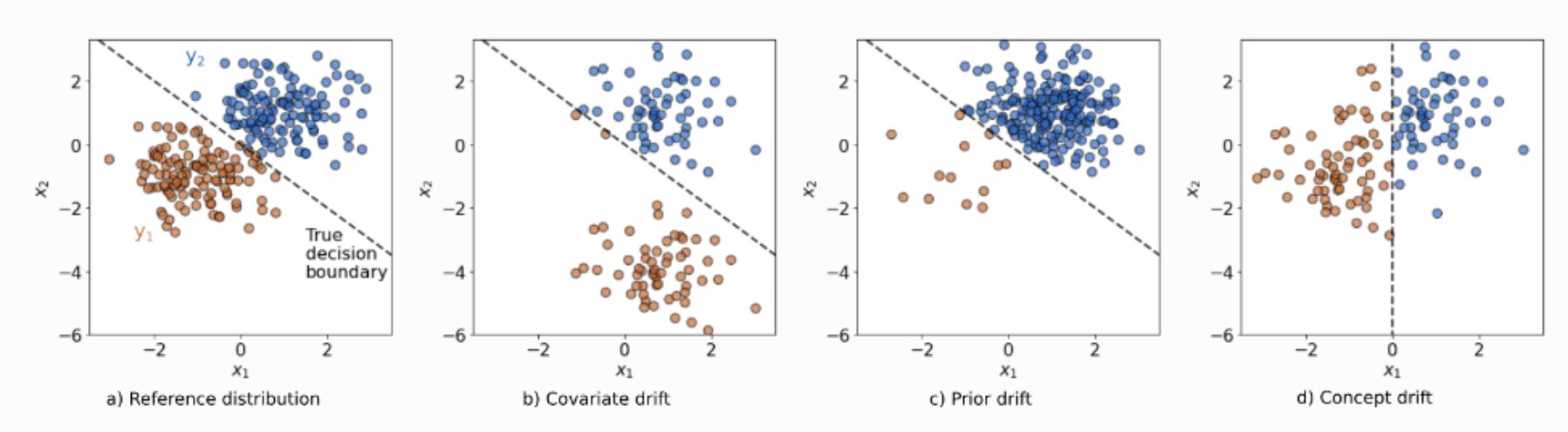
Target drift



CONTINUOUS MONITORING

Data distribution shifts

- difference between the source (training) and target (production) data distributions
- 3 types of data distribution shifts based on $P(X, Y) = P(Y|X)\,P(X) = P(X|Y)\,P(Y)$
  - covariate shift: P(X) changes but P(Y | X) remains same
  - prior (label) shift: P(Y) changes but P(X | Y) remains same
  - concept drift: P(Y | X) changes but P(X) remains same



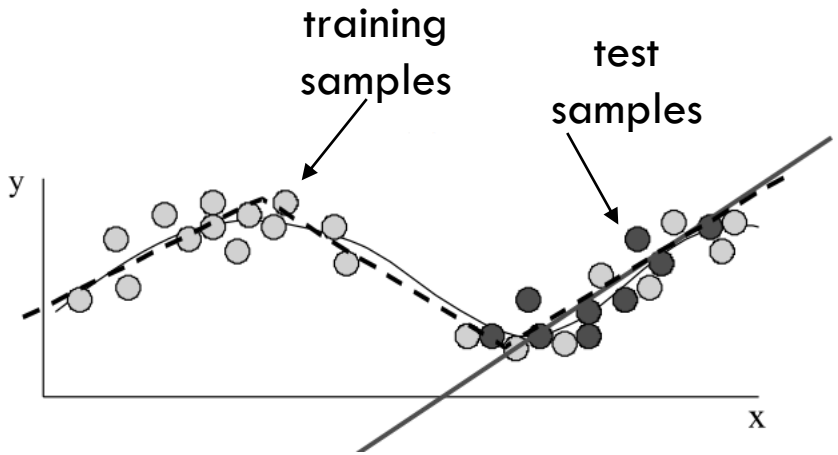
CONTINUOUS MONITORING

Covariate shift

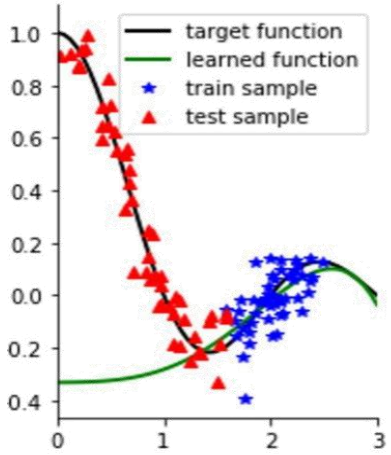
- Difference in P(X) between source and target distributions due to actual real-world changes or selection bias in the training set
- Changes in P(X) have no effect on P(Y | X): outcome is same, independent of P(X)
- Modeling implications
  - when target distribution is represented within source distribution => computational costs (more complex model)

vs.

  - otherwise => model can be severely impacted by the **sparsity** in the corresponding part of target distribution



vs.



CONTINUOUS MONITORING

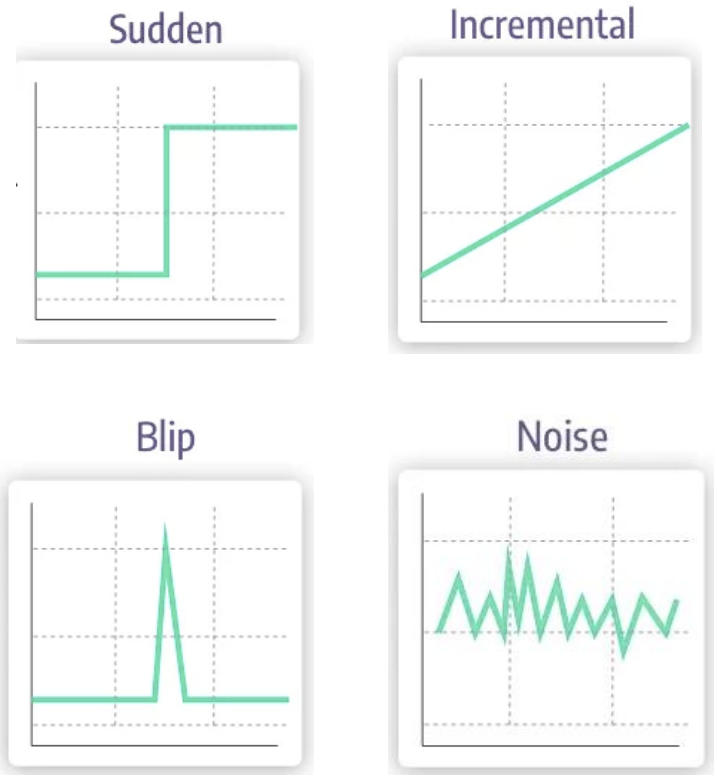
Covariate shift (cont.)

- Detection
  - simple check of summary statistics (min, max, quantiles)
  - suitable statistical tests to compare distributions of individual features: parametric vs. non-parametric
  - **meta-model** for binary classification between “Source” and “Target” data
  - understand the reason: actual changes vs. errors in data pipeline
- Compensation
  - re-train
    - stateless (from scratch) vs. stateful training, importance weighting to target (e.g., newer) data
  - exclude (if possible) features that are subject to covariate drift
  - learn domain-invariant representations

CONTINUOUS MONITORING

Concept drift

- change in relationship between inputs and label: "same input, different output"
- typically, consequence of *unforeseen* circumstances
- Detection
  - performance degradation (+ in conjunction with other shifts)
- Compensation
  - re-training, evtl. with re-weighting newer/older data
  - update models with new features to reflect drifts



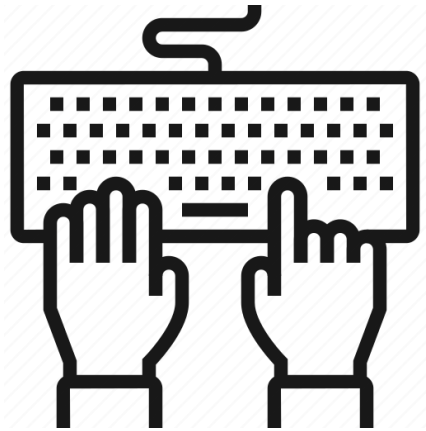
CONTINUOUS MONITORING

Drift Detection

Detector	Tabular	Image	Time Series	Text	Categorical Features	Online	Feature Level
Kolmogorov-Smirnov	✓	✓		✓	✓		✓
Cramér-von Mises	✓	✓				✓	✓
Fisher's Exact Test	✓				✓	✓	✓
Maximum Mean Discrepancy (MMD)	✓	✓		✓	✓	✓	
Learned Kernel MMD	✓	✓		✓	✓		
Context-aware MMD	✓	✓	✓	✓	✓		
Least-Squares Density Difference	✓	✓		✓	✓	✓	
Chi-Squared	✓				✓		✓
Mixed-type tabular data	✓				✓		✓
Classifier	✓	✓	✓	✓	✓		
Spot-the-diff	✓	✓	✓	✓	✓		✓
Classifier Uncertainty	✓	✓	✓	✓	✓		
Regressor Uncertainty	✓	✓	✓	✓	✓		

CONTINUOUS MONITORING

Explore one of the examples in Alibi

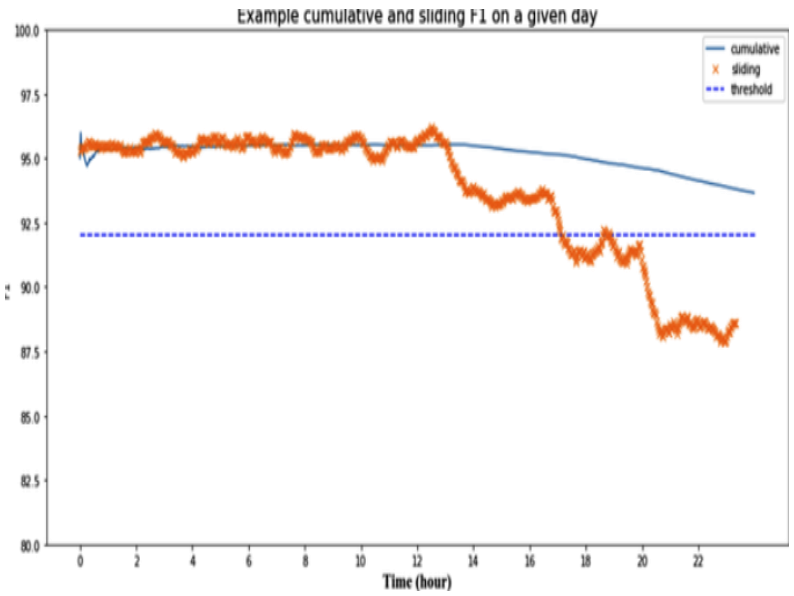


CONTINUOUS MONITORING

What and ***how*** to monitor:

- Monitor directly business KPI(s) w.r.t. time
  - predefined thresholds
  - length of sliding window: trade-off

- When **delayed groundtruth** in production:
  - monitor **shift** in **predicted** labels (proxy for covariate shift)
  - monitor covariate shift in P(X)
  - advanced: black box shift estimation



CONTINUOUS MONITORING

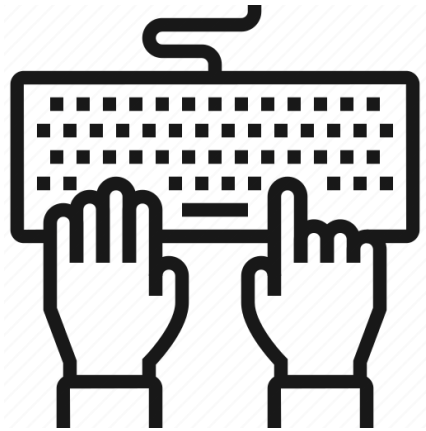
Focused errors

- Monitor separately errors for important special cases: data **not well-represented** in training data with **asymmetric impact**

Sliced metrics

- **Slice**: a segment of the data in which a specific attribute has a certain value
- Monitor performance separately per slice (caution: complex decision-making!)
- Connection to model **bias** and issues related **fairness**

CONTINUOUS MONITORING



Evidently: an open-source Python library and Cloud platform that helps evaluate, test, and monitor data and ML-powered systems.

Explore Evidently Tutorial

See also: OSS Quickstart - Data and ML monitoring

CONTINUOUS MONITORING

Additional considerations:

- **integration test**: use an **end-to-end set** that defines model inputs and outputs that must **always** work to detect schema drifts
- **numerical stability**: detect NaNs or Inf in feature values to detect source changes
- monitor **suspicious usage**: hourly/daily model servings, compare to corresponding values from previous day/week
- reaction to prevent **model abuse**:
  - pay per request
  - progressively longer pauses before responding to requests
  - develop user-reputation scores (e.g., review helpfulness)

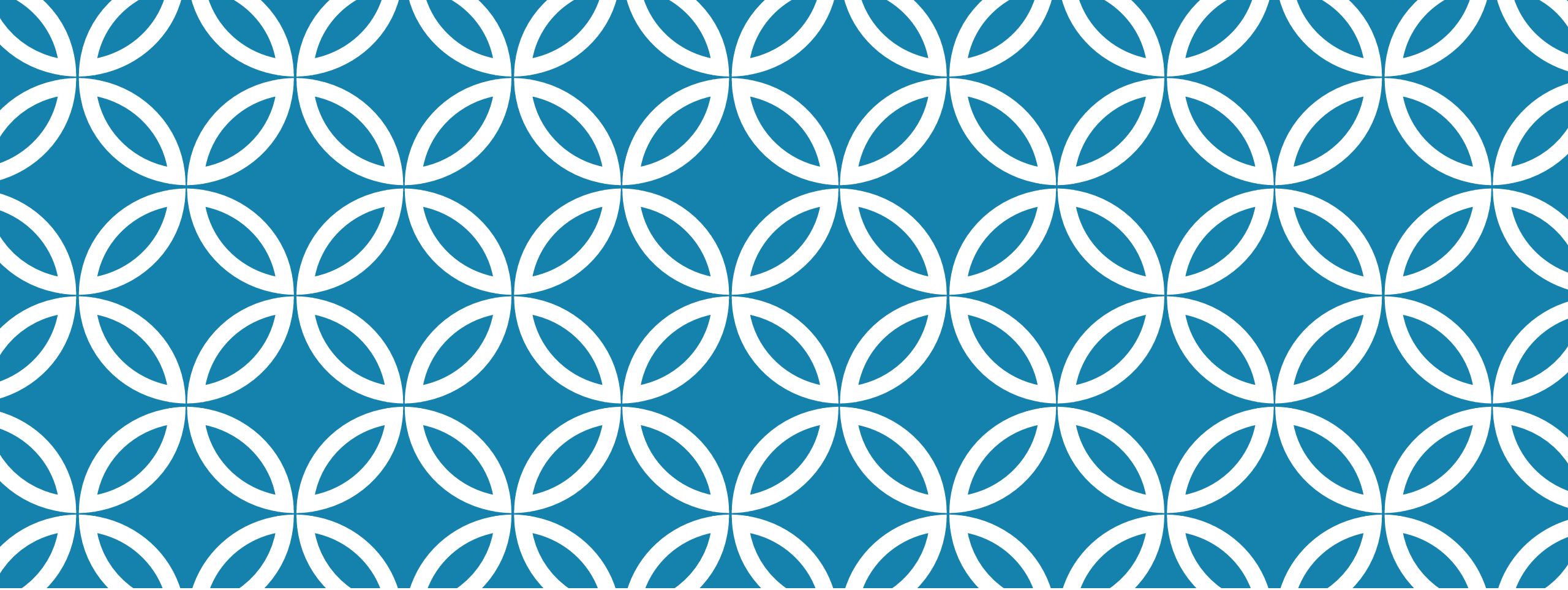
CONTINUOUS MONITORING

Special topic: Degenerate Feedback Loops

- When predictions of a deployed model negatively affect the model itself
  - a.k.a. “exposure bias”, “popularity bias”, “filter bubbles”, “echo chambers”

Monitoring to detect degenerate feedback loops

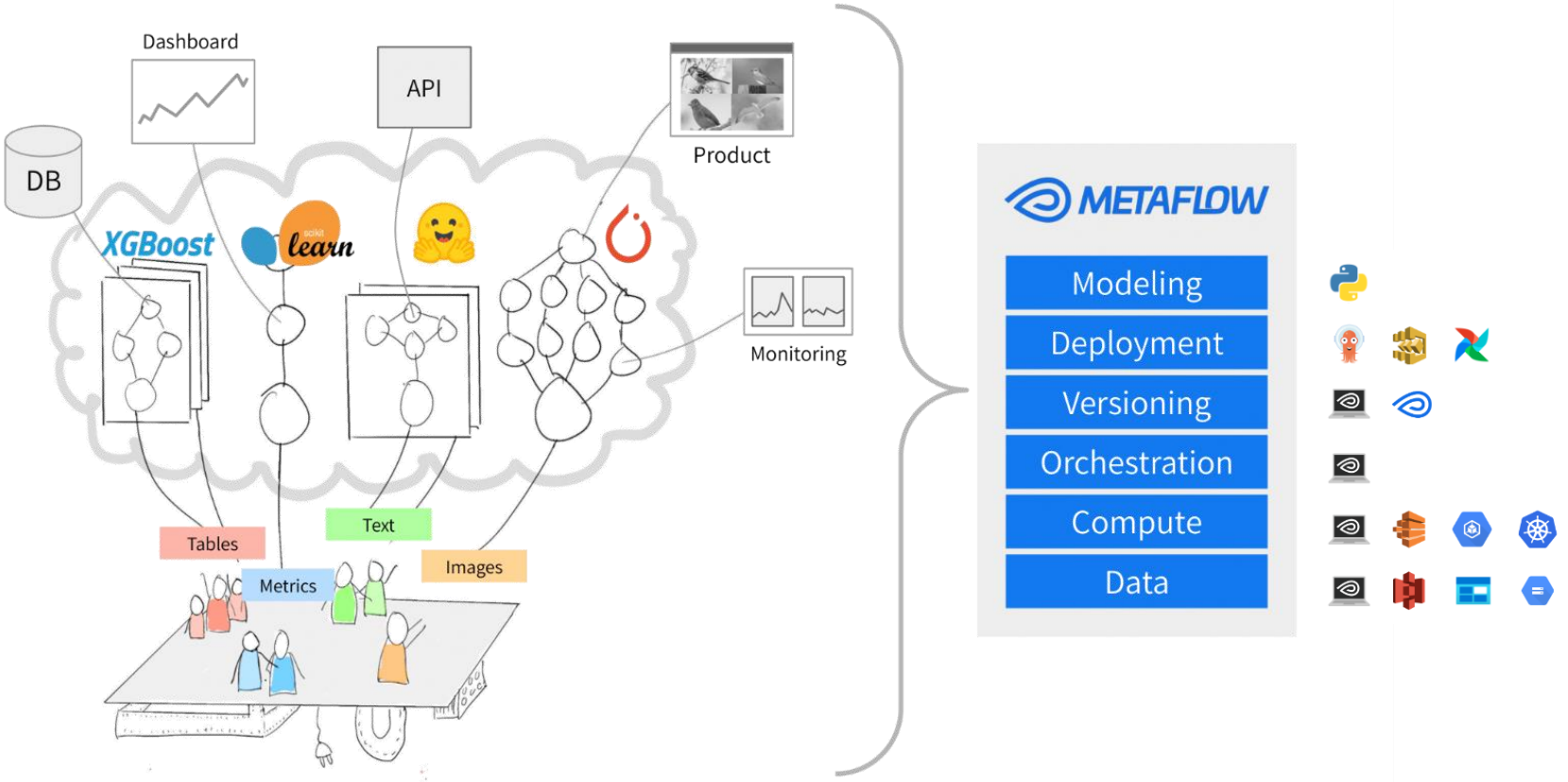
- Example: **diversity score** in the predictions (especially in recommeder systems)
- Define measures for correcting degenerate feedback loops



DATA SCIENCE WORKFLOW MANAGEMENT

Part III: MLOps

DATA SCIENCE WORKFLOW MANAGEMENT



What we need: infrastructure/technology that enables MLOps processes

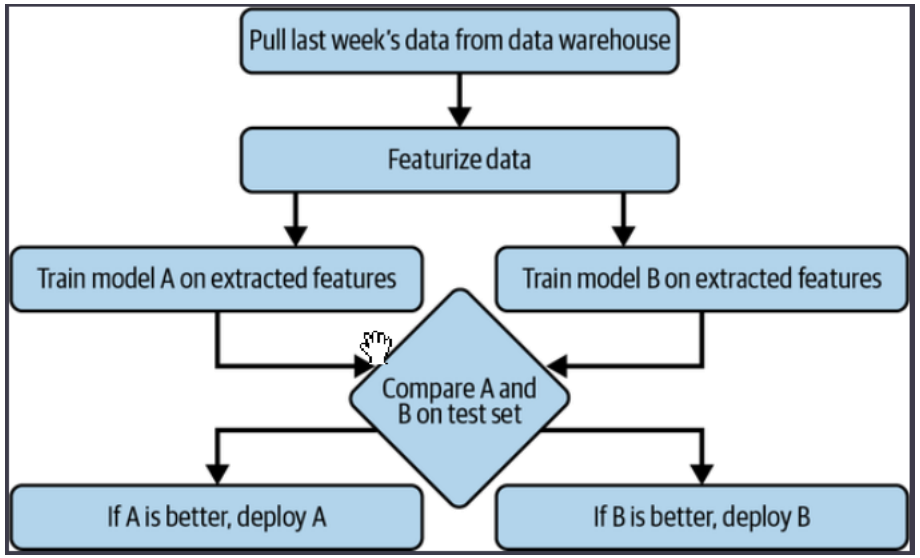
DATA SCIENCE WORKFLOW MANAGEMENT

ML workflows have:

- repetitiveness
- dependencies

(Conditional) Dependencies as complex relationships in workflows:

- actions depending on data validation, online experiments, etc.
- represented as DAGs



DATA SCIENCE WORKFLOW MANAGEMENT

Schedulers

- **when** to run jobs and **what** resources are needed to run those jobs
- for periodical jobs
- job-type abstractions: DAGs, priority queues, user-level quotas (max resources per user)

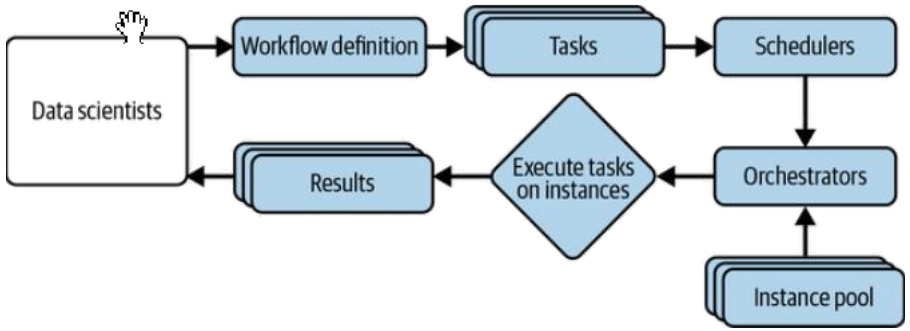
Orchestrators

- **where** to get resources to run the jobs, etc.
  - "provisioning": add more resources to handle the workload
- for services where you have a long-running server that responds to requests
- lower level abstractions: machines, instances, clusters, service-level grouping, replication, etc.

DATA SCIENCE WORKFLOW MANAGEMENT

Workflow management tools

- allow to specify workflows as DAGs
- workflows defined using either code (Python) or configuration files (YAML)
- workflow management tools usually come with some scheduler
- connected to an orchestrator, they allocate resources to run the workflow



DATA SCIENCE WORKFLOW MANAGEMENT

Workflow management tools: Airflow

- earliest (developed at Airbnb in 2014)
- “configuration as code”
- monolithic
- not easily parameterized
- static

```python
from datetime import datetime, timedelta

from airflow import DAG
from airflow.operators.bash import BashOperator
from airflow.providers.docker.operators.docker import DockerOperator

dag = DAG(
    'docker_sample',
    default_args={'retries': 1},
    schedule_interval=timedelta(minutes=10),
    start_date=datetime(2021, 1, 1),
    catchup=False,
)

t1 = BashOperator(task_id='print_date', bash_command='date', dag=dag)
t2 = BashOperator(task_id='sleep', bash_command='sleep 5', retries=3, dag=dag)
t3 = DockerOperator(
    docker_url='tcp://localhost:2375',  # Set your docker URL
    command='/bin/sleep 30',
    image='centos:latest',
    network_mode='bridge',
    task_id='docker_op_tester',
    dag=dag,
)

t4 = BashOperator(
    task_id='print_hello',
    bash_command='echo "hello world!!!"',
    dag=dag
)

t1 >> t2
t1 >> t3
t3 >> t4
```

DATA SCIENCE WORKFLOW MANAGEMENT

Workflow management tools: Kubeflow

- abstracts away infrastructure boilerplate code („infrastructure abstraction tools“)
- access to prod from local notebooks
- uses Kubeflow Pipelines for scheduling and K8s for orchestration
- fully parameterized and dynamic
- Kubeflow boilerplate: still need to write a Docker and a YAML file

Dockerfile for the component train

```
ARG BASE_IMAGE_TAG=1.12.0-py3
FROM tensorflow/tensorflow:$BASE_IMAGE_TAG
RUN python3 -m pip install keras
COPY ./src /pipelines/component/src
```

Load specs of different components

```
preprocess_operation = kfp.components.load_component_from_url(
    'https://storage.googleapis.com/{}/components/preprocess/component.yaml'.format(BUCKET))
help(preprocess_operation)

train_operation = kfp.components.load_component_from_url(
    'https://storage.googleapis.com/{}/components/train/component.yaml'.format(BUCKET))
help(train_operation)

ai_platform_deploy_operation = comp.load_component_from_url(
    "https://storage.googleapis.com/{}/components/deploy/component.yaml".format(BUCKET))
help(ai_platform_deploy_operation)
```

Spec for the component train

```
name: train
description: Trains the NER Bi-LSTM.
inputs:
- {name: Input x URI, type: GCSPath}
- {name: Input y URI, type: GCSPath}
- {name: Input job dir URI, type: GCSPath}
- {name: Input tags, type: Integer}
- {name: Input words, type: Integer}
- {name: Input dropout }
- {name: Output model URI template, type: GCSPath}
outputs:
  - name: Output model URI
    type: GCSPath
implementation:
  container:
    image: gcr.io/<PROJECT-ID>/kubeflow/ner/train:latest
    command: [
      python3, /pipelines/component/src/train.py,
      --input-x-path,            {inputValue: Input x URI},
      --input-job-dir,           {inputValue: Input job dir URI},
      --input-y-path,            {inputValue: Input y URI},
      --input-tags,              {inputValue: Input tags},
      --input-words,             {inputValue: Input words},
      --input-dropout,           {inputValue: Input dropout},
      --output-model-path,       {inputValue: Output model URI template},

      --output-model-path-file,  {outputPath: Output model URI},

    ]
```

Create the workflow in Python

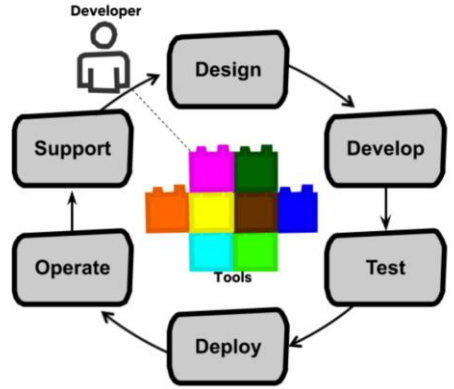
```
@dsl.pipeline(
  name='Named Entity Recognition Pipeline',
  description='Performs preprocessing, training and deployment.'
)
def pipeline():

    preprocess_task = preprocess_operation(
        input_1_uri='gs://kubeflow-examples-data/named_entity_recognition_dataset/ner.csv,
        output_y_uri_template="gs://{}/{{workflow.uid}}/preprocess/y/data".format(BUCKET),
        output_x_uri_template="gs://{}/{{workflow.uid}}/preprocess/x/data".format(BUCKET),
        output_preprocessing_state_uri_template="gs://{}/{{workflow.uid}}/model".format(BUCKET)
    ).apply(kfp.gcp.use_gcp_secret('user-gcp-sa'))


    train_task = train_operation(
        input_x_uri=preprocess_task.outputs['output-x-uri'],
        input_y_uri=preprocess_task.outputs['output-y-uri'],
        input_job_dir_uri="gs://{}/{{workflow.uid}}/job".format(BUCKET),
        input_tags=preprocess_task.outputs['output-tags'],
        input_words=preprocess_task.outputs['output-words'],
        input_dropout=0.1,
        output_model_uri_template="gs://{}/{{workflow.uid}}/model".format(BUCKET)
    ).apply(kfp.gcp.use_gcp_secret('user-gcp-sa'))


    deploy_task = ai_platform_deploy_operation(
        model_path= train_task.output,
        model_name="named_entity_recognition_kubeflow",
        model_region="us-central1",
        model_version="version1",
        model_runtime_version="1.13",
        model_prediction_class="model_prediction.CustomModelPrediction",
        model_python_version="3.5",
        model_package_uris="gs://{}/routine/custom_prediction_routine-0.2.tar.gz".format(BUCKET)
    ).apply(kfp.gcp.use_gcp_secret('user-gcp-sa'))
```

DATA SCIENCE WORKFLOW MANAGEMENT

Infrastructure requires a very different set of skills from data science

Success of full-stack data scientist relies on the tools: abstract the data scientists from complexities of containerization, distributed processing, automatic failover, etc.

1. Specialists build tools to automate their parts of the pipeline
2. Data scientists leverage these tools to own the project end-to-end



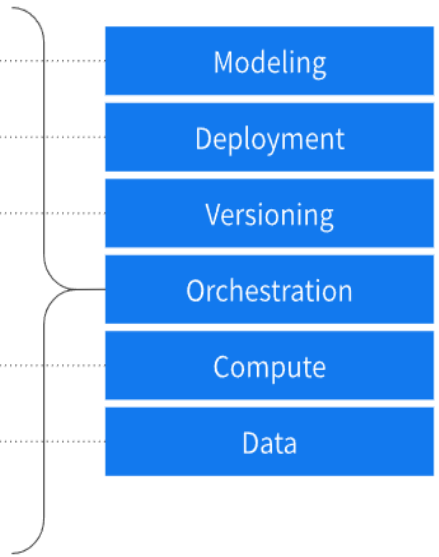
DATA SCIENCE WORKFLOW MANAGEMENT

Workflow management tools: Metaflow

Metaflow is a human-friendly Python library that makes it straightforward to develop, deploy, and operate various kinds of data-intensive applications, in particular those involving data science, ML, and AI.

Metaflow was originally developed at Netflix to boost the productivity of data scientists who work on a wide variety of projects, from classical statistics to state-of-the-art deep learning.

```
from metaflow import FlowSpec, step, conda_base,\
                     kubernetes, schedule

@conda_base(libraries={'scikit-learn': '1.1.2'})
@schedule(daily=True)
class HelloFlow(FlowSpec):
    @step
    def start(self):
        self.x = 1
        self.next(self.end)

    @kubernetes(memory=64000)
    @step
    def end(self):
        self.x += 1
        print("Hello world! The value of x is", self.x)

if __name__ == '__main__':
    HelloFlow()
```



DATA SCIENCE WORKFLOW MANAGEMENT

Views of a Meta(work)flow

Conceptually

- Metaflow workflows are DAGs
- Each node in DAG is processing step in the workflow
- Metaflow executes the idiomatic python code in each step of the workflow **as-is** in separate containers packaged with its corresponding dependencies

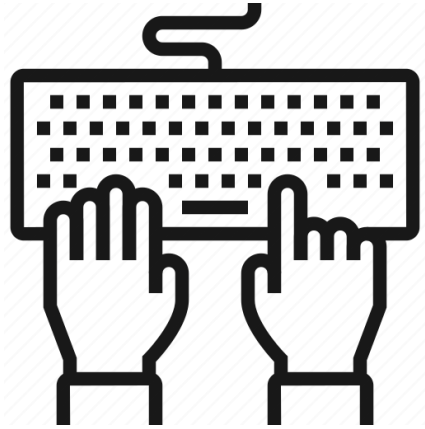
Programmatically

- Each flow is a standard python class
- Inherits from Metaflow's FlowSpec Class
- Each function that represents a step is tagged with a @step decorator
- Each step function should end by pointing to its successor step function. You can do this with self.next(self.function_name_here)
- Implements the start and end function.

The Structure of Metaflow Code

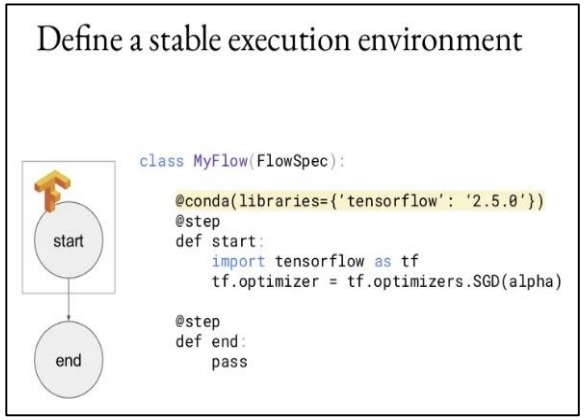
DATA SCIENCE WORKFLOW MANAGEMENT

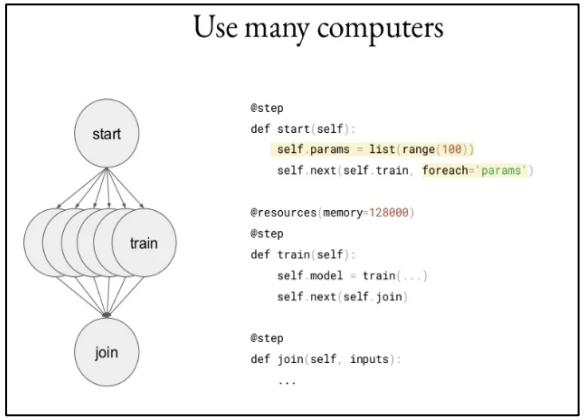
Explore the “Hello World” Metaflow tutorial

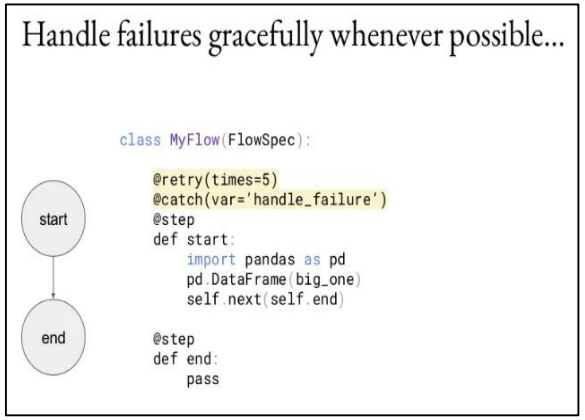


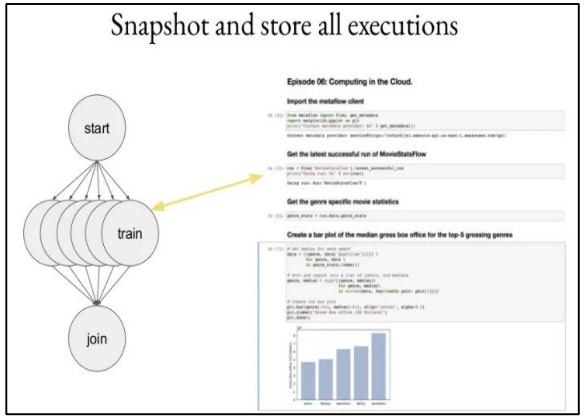
DATA SCIENCE WORKFLOW MANAGEMENT

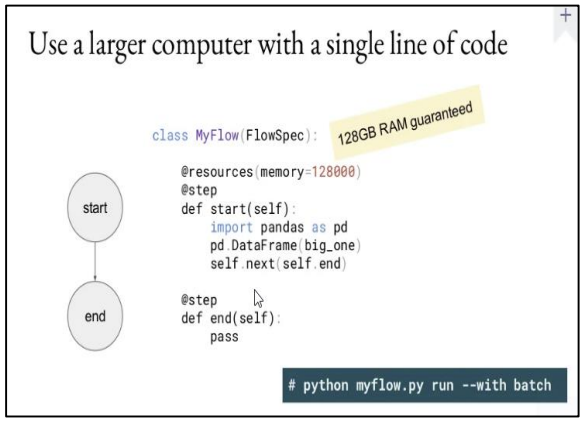
Metaflow presentation

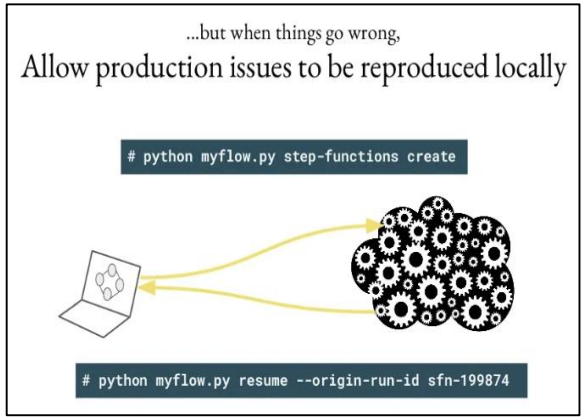












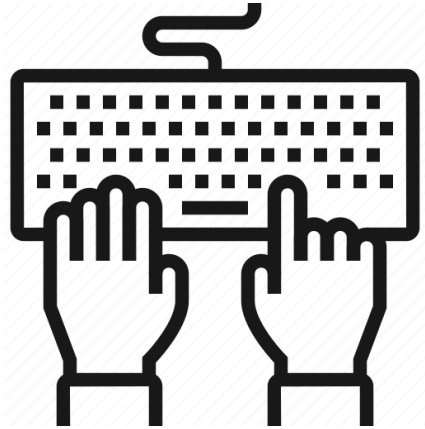
DATA SCIENCE WORKFLOW MANAGEMENT

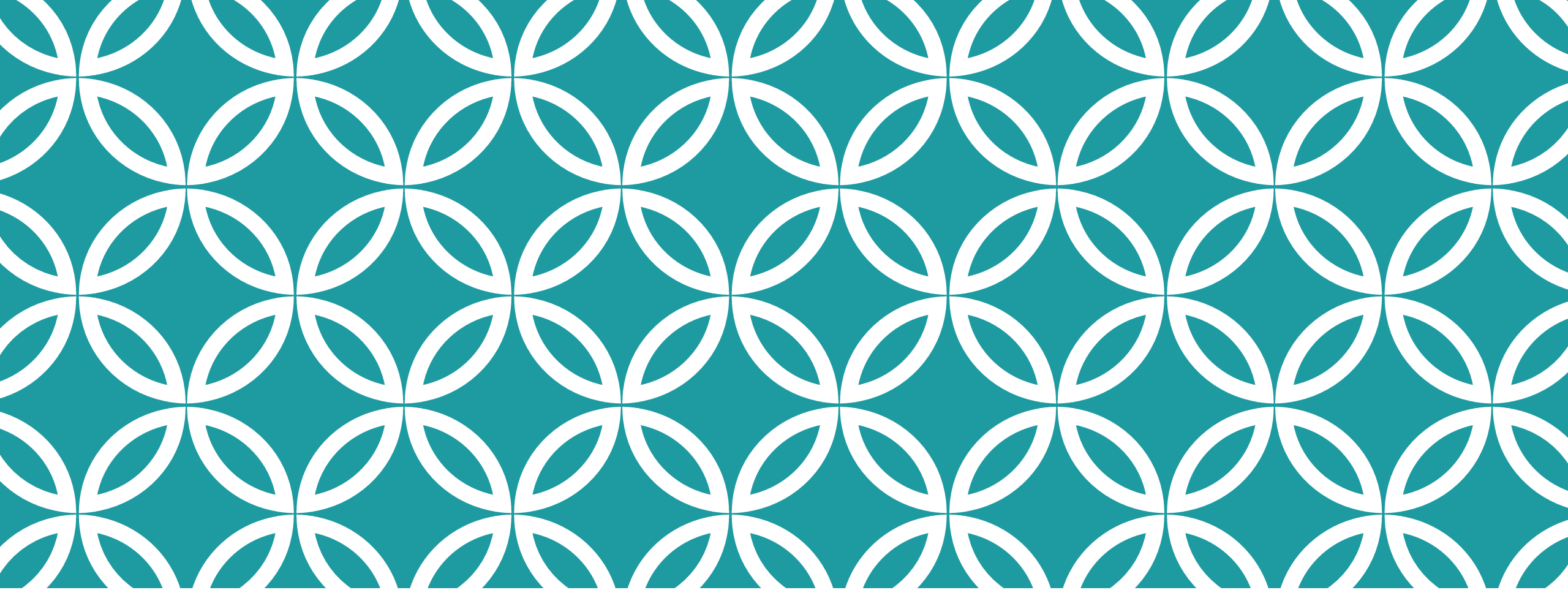
Scenarios that Metaflow makes possible

- Collaboration: You want to help another data scientist debug an error. You wish you could pull up his failed run state in your laptop as-is and restart the run after fixit it.
- Restarting a run: A run failed. You wish you could restart the workflow from where it failed/stopped.
- Hybrid runs: You want to run one step of your workflow locally (maybe the data load step since the dataset is in your downloads folder) but want to run another compute-intensive step (the model training) on the cloud.
  - python3 sample_flow.py run --with kubernetes
  - @kubernetes(memory=60000, cpu=8)
- Inspecting run metadata: Three data scientists have been tuning the hyperparameters to get better accuracy on the same model. Now you want to analyze all their training runs and pick the best performing hyperparameter set.
- Multiple versions of the same package: You wish you could use multiple versions of the sklearn library in your project?

DATA SCIENCE WORKFLOW MANAGEMENT

Optional: Explore rest Metaflow tutorials





CONCLUSION & OUTLOOK

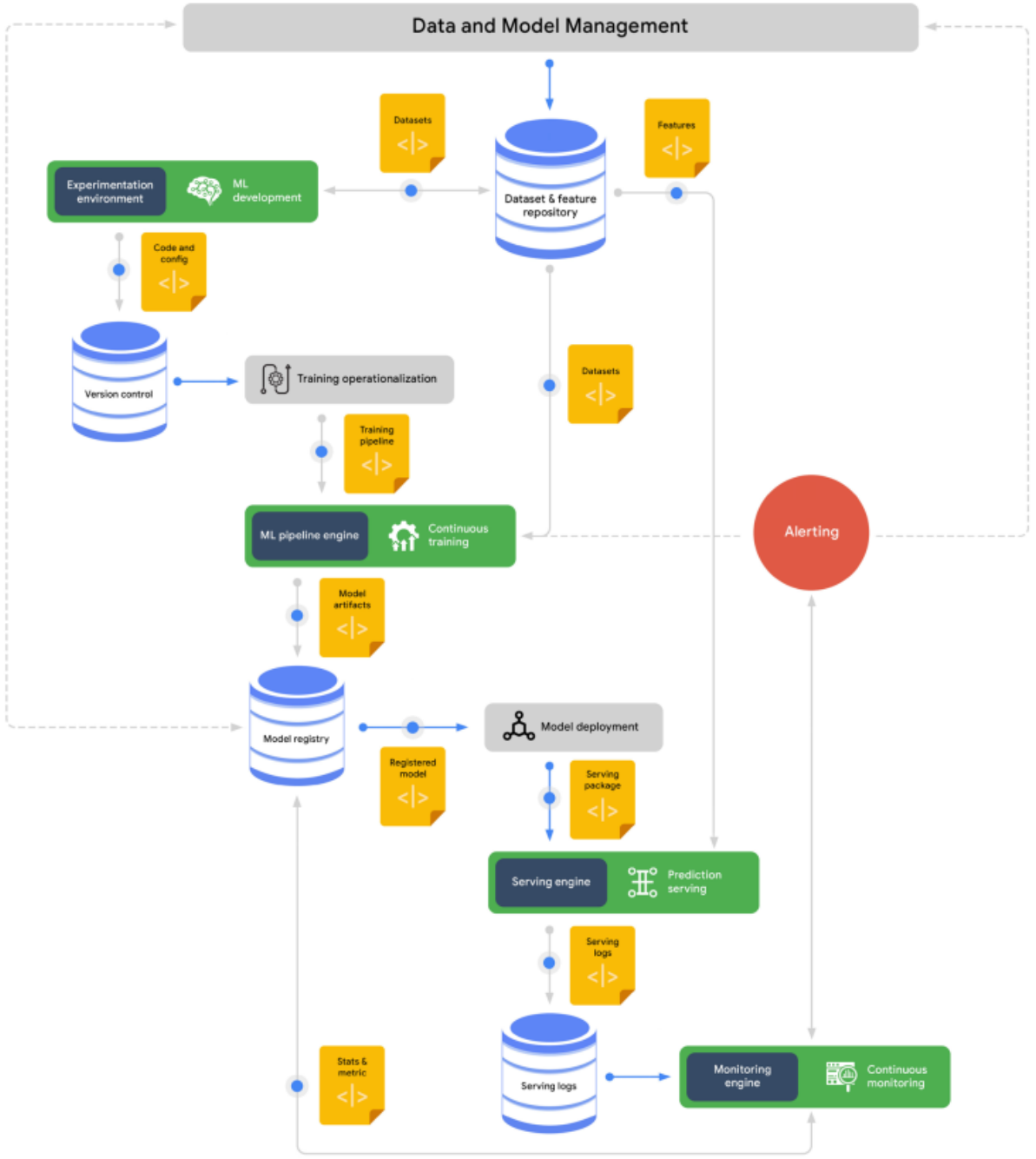
PUTTING IT ALL TOGETHER

Delivering **business value** through ML:

- building an integrated ML system that operates **continuously**
- **adapt** to changes in the dynamics of the business environment
- collect, process, and manage ML **artifacts** (datasets, features, etc.)
- train and evaluate models **at scale**
- **serve** the model for predictions
- **monitor** the model performance in production

Advantages:

- reduced time to market, reliability, performance, scalability, and security



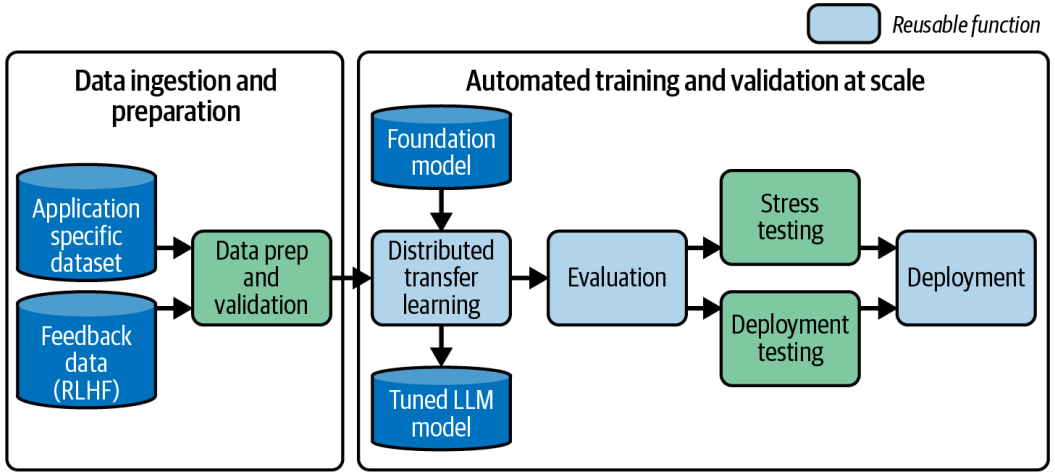
OUTLOOK

LLMOps: deployment, monitoring and maintenance of large language models

- **Computational resources**: fine-tuning requires access to specialized hardware
- **Efficient Inference**: challenging due to model size (compression/distillation)
- **Integrating human feedback**: approaches for reinforcement learning from human feedback
- **Custom performance metrics**: e.g., bilingual evaluation understudy (BLEU) and Recall-Oriented Understudy for Gisting Evaluation (ROUGE)
- **Prompt engineering**: critical for reliable responses (reduce hallucination and prompt hacking)
- **Complicated LLM chains/pipelines**: calls to external systems (vector databases or web)

OUTLOOK

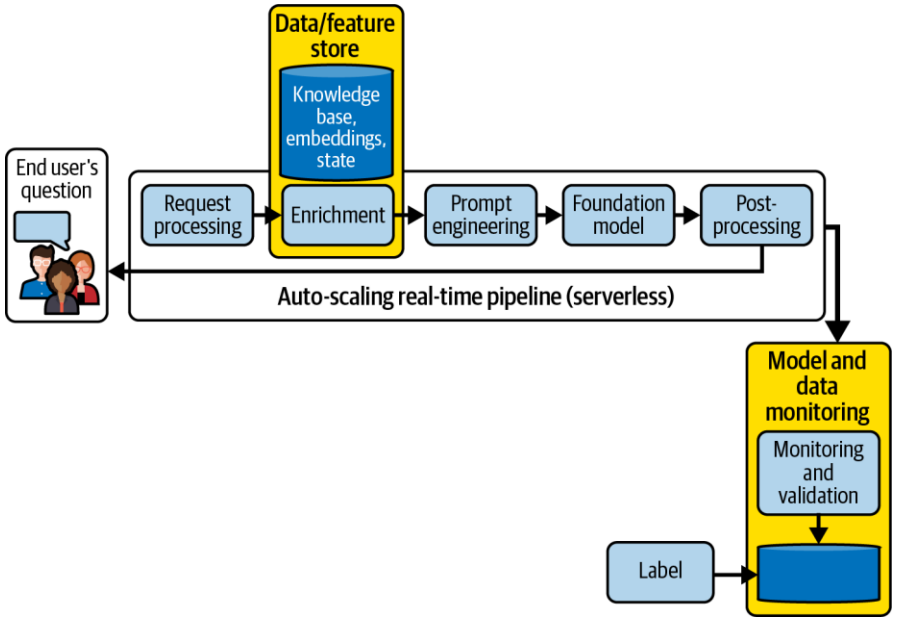
LLM model tuning pipeline



Additional evaluation criteria:

- Toxicity detection and filtering
- Bias detection
- Privacy protection

LLM real-time serving pipeline



Drifts for unstructured data:
Track the change of distribution of embeddings