Model Governance Tutorial

Strata Data, San Jose, 2020

Boris Lublinsky, Lightbend
Dean Wampler, Anyscale

boris.lublinsky@lightbend.com
dean@anyscale.io

Lightbend

anyscale

If you have not done so already, download the tutorial from GitHub

https://github.com/lightbend/ML-metadata-tutorial

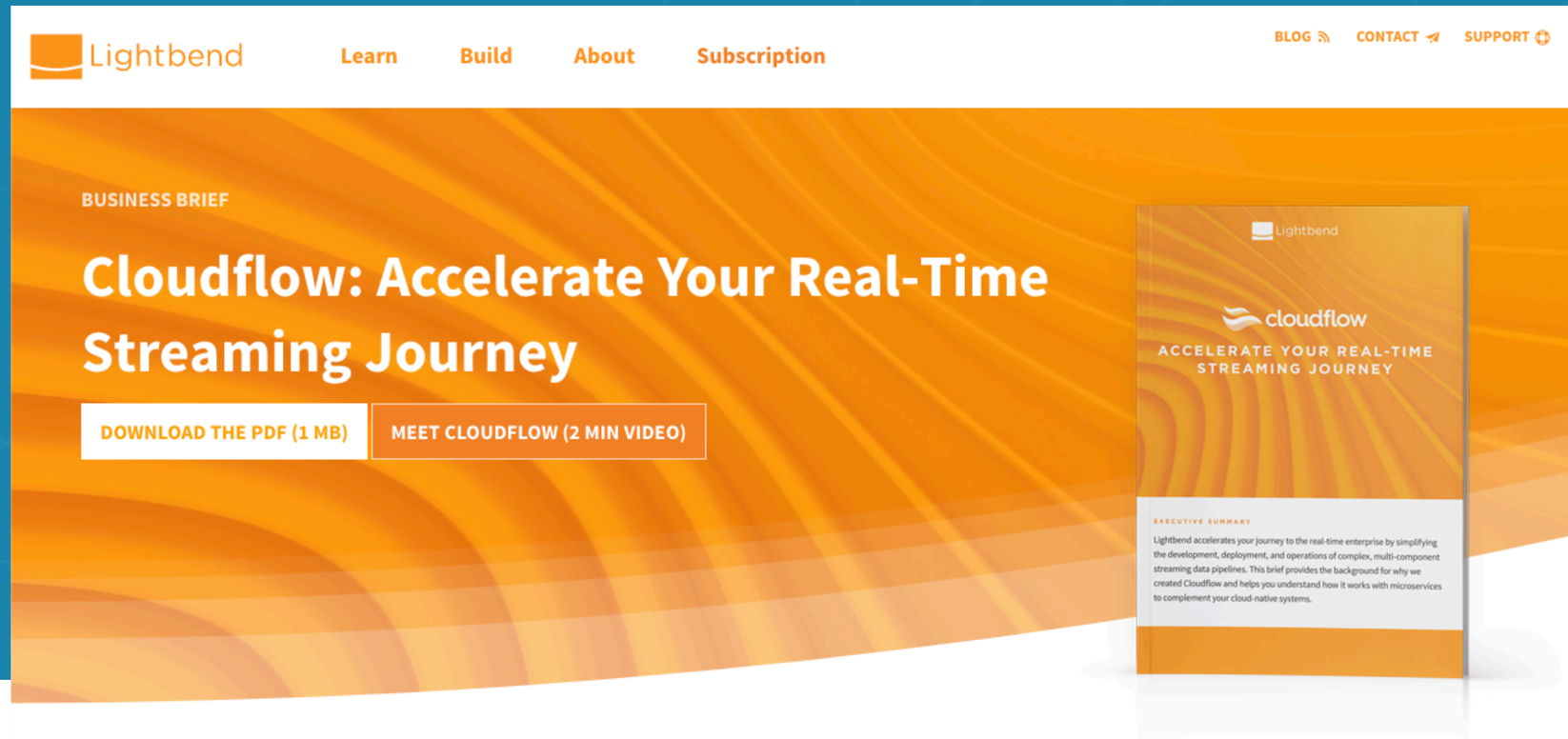
See the README for setup instructions.

These slides are in the presentation folder.

Introductions

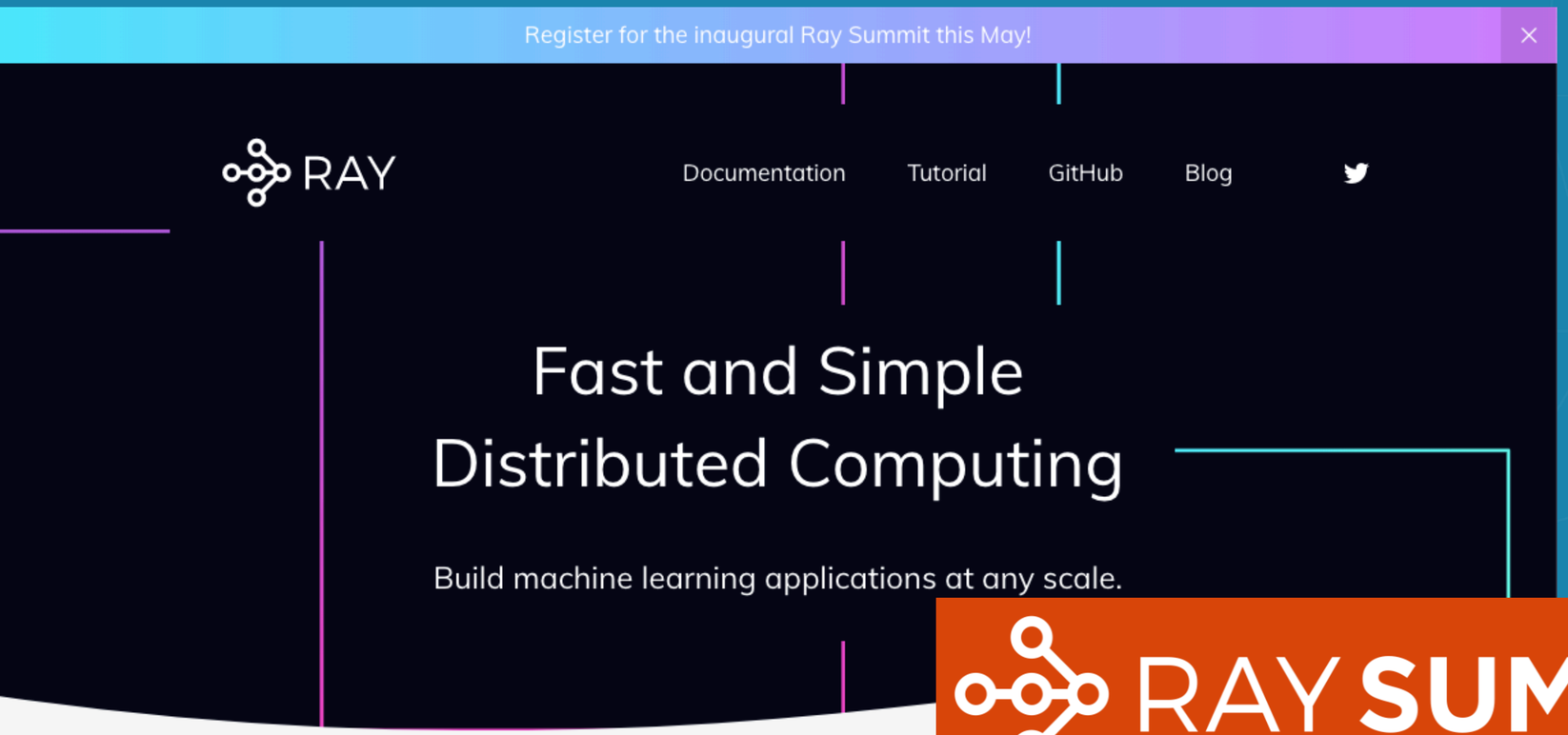
Boris Lublinsky

- Lightbend
- cloudflow.io



Dean Wampler

- Anyscale
- anyscale.io & ray.io

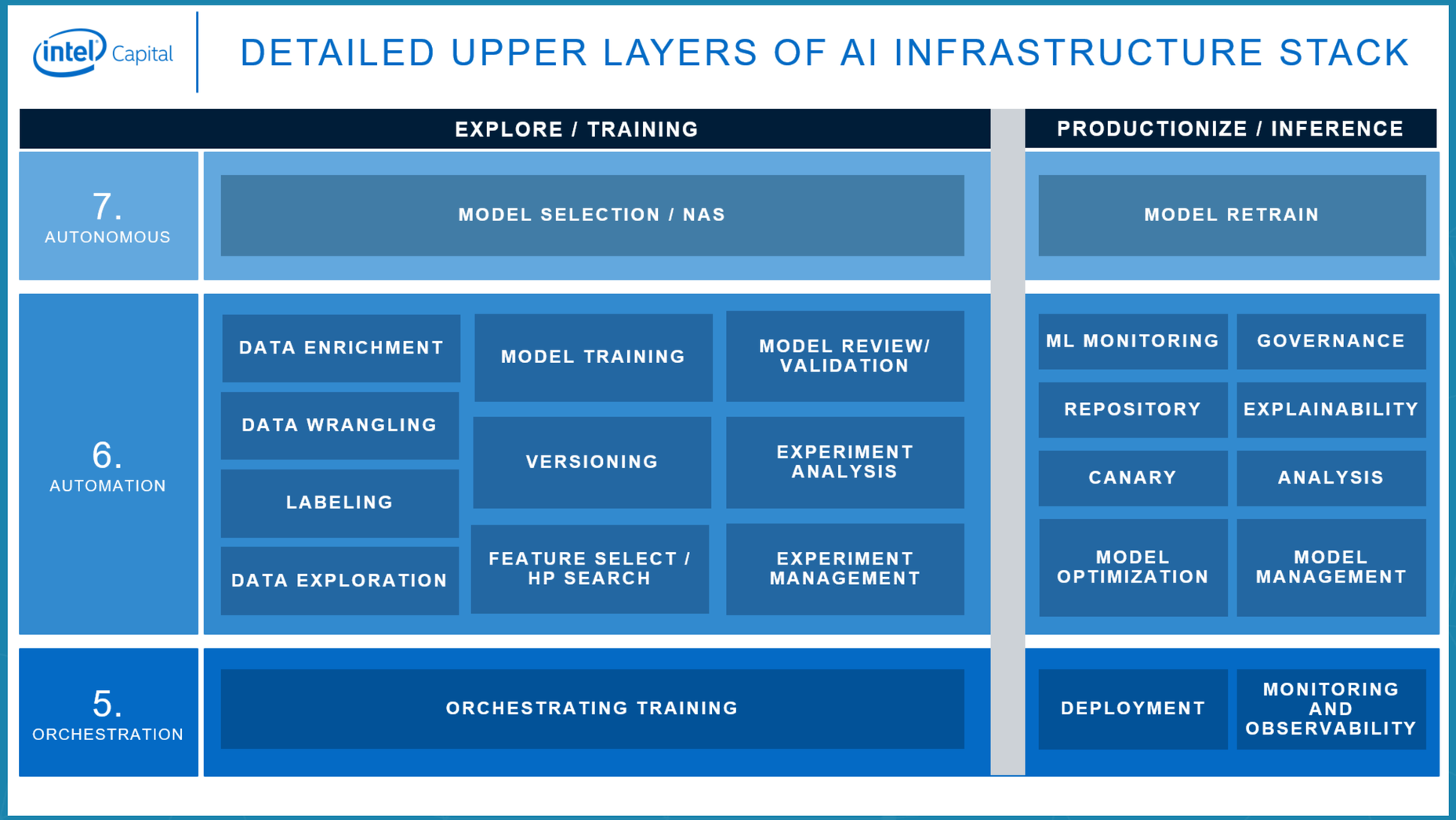


You?

- Data scientists? Data engineers? Other?
- Who is doing model training & serving now?
- Using the JVM? Python? Other language?
- Using Kubernetes/Kubeflow? MLflow? Other?

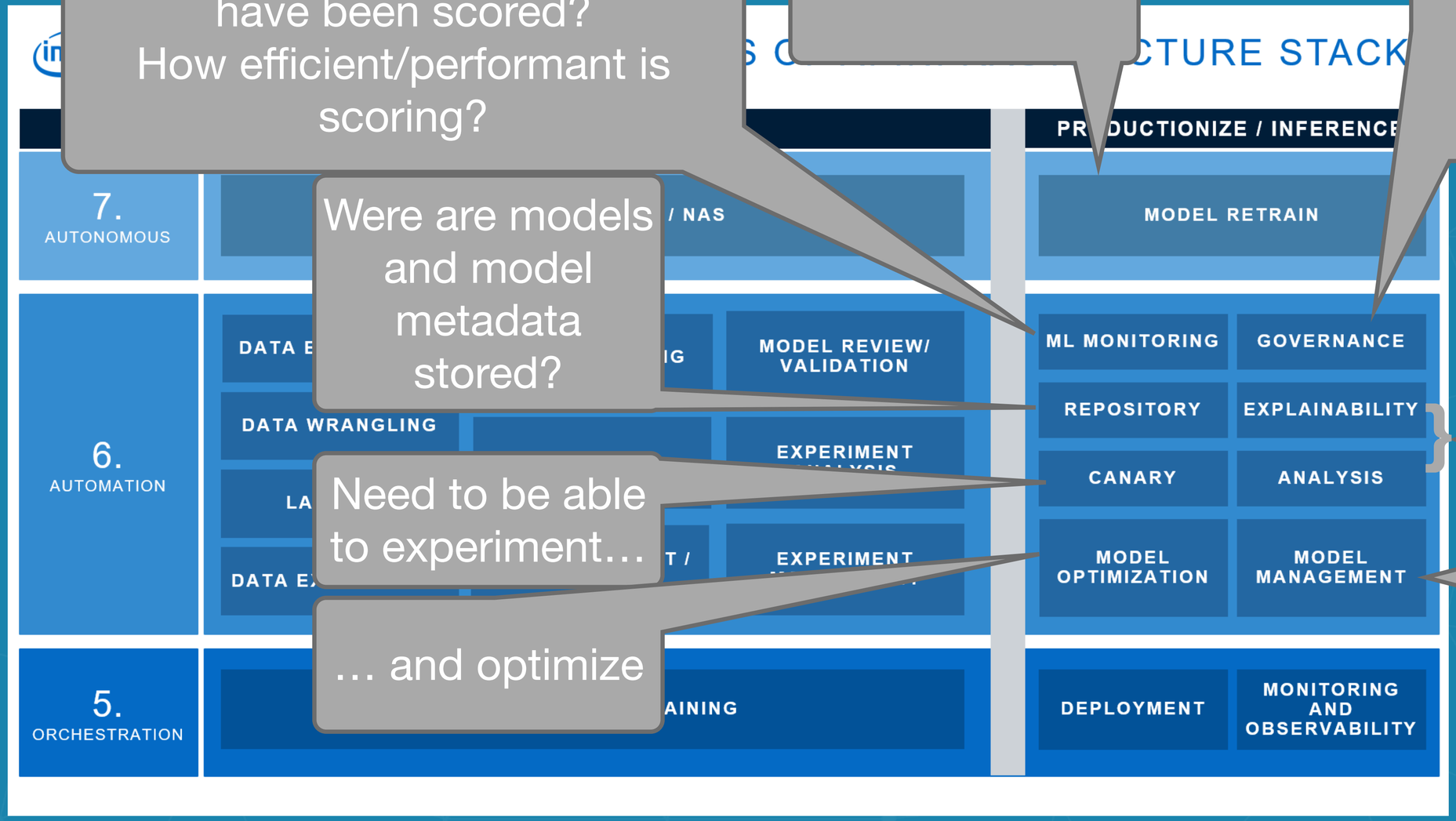
Model Serving in the Larger Context

The AI Stack



Source: https://www.intel.com/content/www/us/en/intel-capital/news/story.html?id=730

Th



How well is the model performing inference? How many records have been scored? How efficient/performant is scoring?

When do we need to retrain?

Model governance is our focus, but it depends on other parts, too…

Were are models and model metadata stored?

Need to be able to experiment…

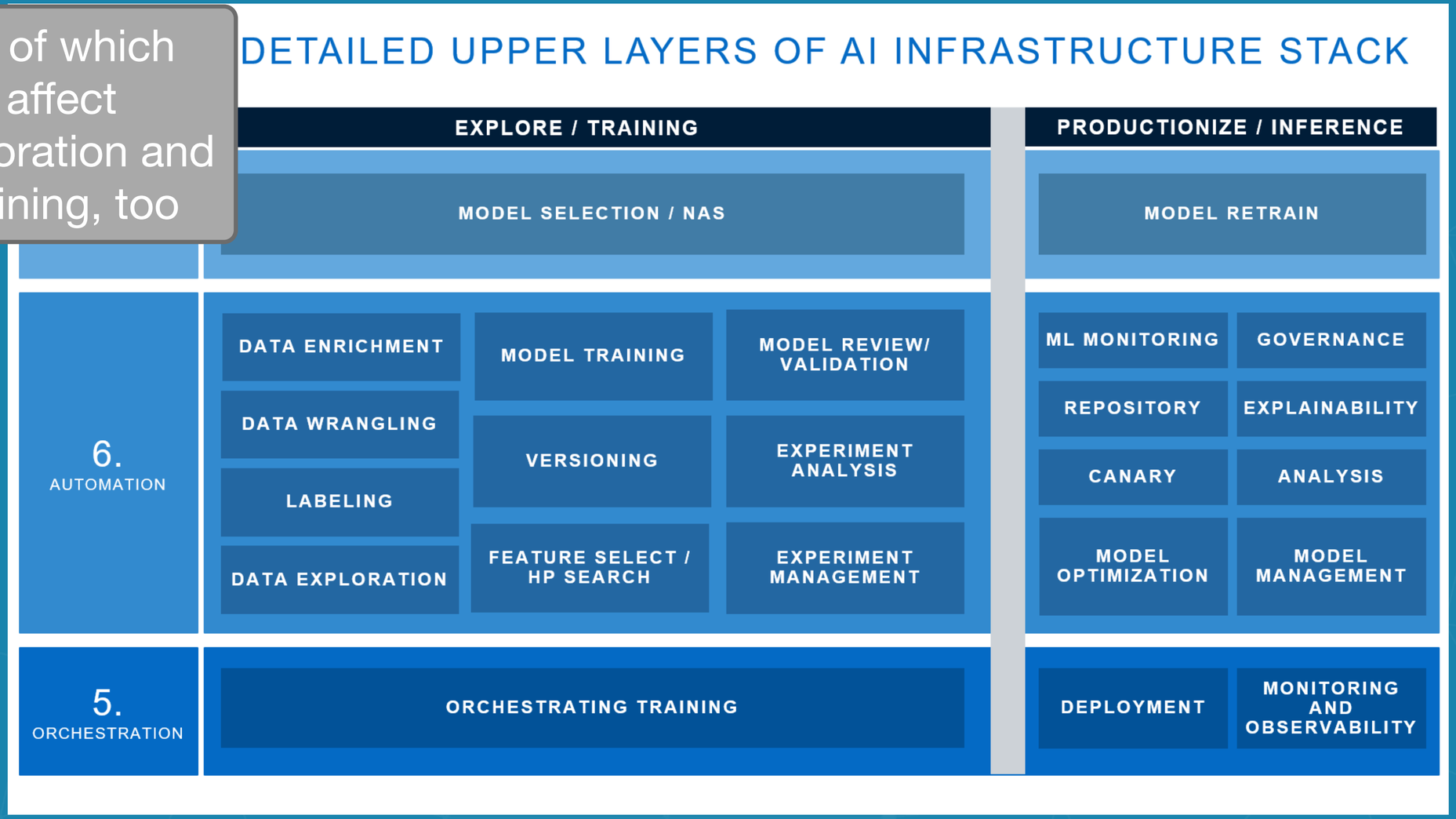
… and optimize

Governance supports these

How are models managed?

The AI Stack

All of which affect exploration and training, too



The AI Stack

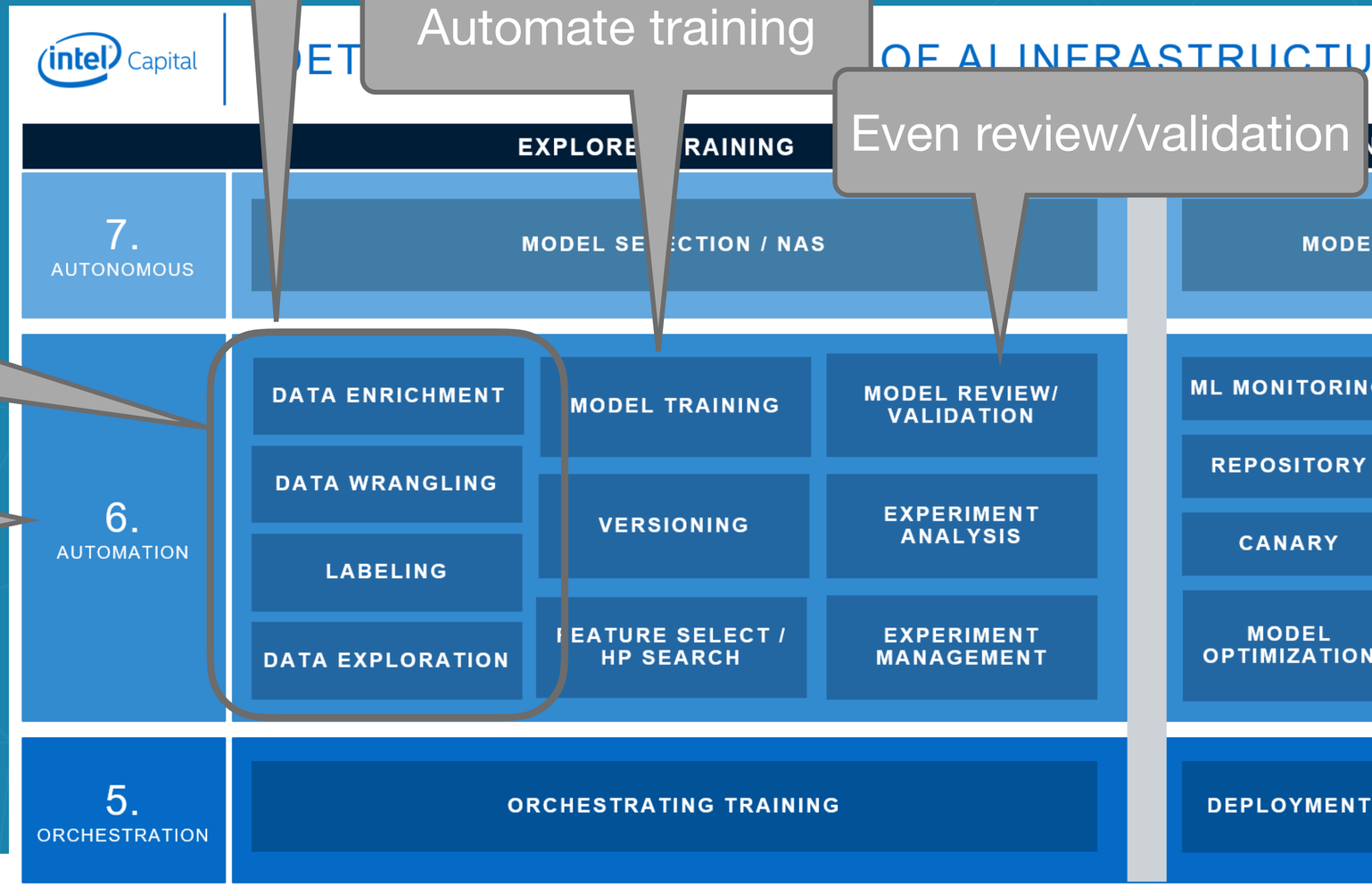
Automate model selection (e.g., hyper parameter tuning and AutoML)

Automate training

Even review/validation

Data governance requires controlled access

Automation necessary for repeatability, traceability, control, security, …



Cloudy with High Chance of DBMS: A 10-year Prediction for Enterprise-Grade ML

Ashvin Agrawal, Rony Chatterjee, Carlo Curino, Avrilia Floratou, Neha Gowdal, Matteo Interlandi, Alekh Jindal, Konstantinos Karanasos, Subru Krishnan, Brian Kroth, Jyoti Leeka, Kwanghyun Park, Hiren Patel, Olga Poppe, Fotis Psallidas, Raghu Ramakrishnan, Abhishek Roy, Karla Saur, Rathijit Sen, Markus Weimer, Travis Wright, Yiwen Zhu

name.surname@microsoft.com

Microsoft

ABSTRACT

Machine learning (ML) has proven itself in high-value web applications such as search ranking and is emerging as a powerful tool in a much broader range of enterprise scenarios including voice recognition and conversational understanding for customer support, autotuning for videoconferencing, intelligent feedback loops in large-scale sysops, manufacturing and autonomous vehicle management, complex financial predictions, just to name a few.

Meanwhile, as the value of data is increasingly recognized and monetized, concerns about securing valuable data and risks to individual privacy have been growing. Consequently, rigorous data management has emerged as a key requirement in enterprise settings.

How will these trends (ML growing popularity, and stricter data

of thousands of small teams will build millions[2] of ML-infused applications—most just moderately remunerative, but with huge collective value.

When it comes to leveraging ML in *enterprise* applications, especially in regulated environments, the level of scrutiny for data handling, model fairness, user privacy, and debuggability will be substantially higher than in the first wave of ML applications. Consider the healthcare domain: ML models may be trained on sensitive medical data, and make predictions that determine patient treatments—copying CSV files on a laptop and maximizing *average* model accuracy just doesn't cut it! We refer to this new class of applications as *Enterprise Grade Machine Learning (EGML)*.

In this paper, we speculate on how ML and database systems

Link: 10th Annual Conference on Innovative Data Systems Research (CIDR '20) Jan 12-15, 2020, Amsterdam, Netherlands.
Summarized in: https://blog.acolyer.org/2020/02/19/ten-year-egml-predictions/

Enterprise-Grade ML (EGML)

- How will growth of ML and data governance concerns intersect?
- How can small orgs build millions of small ML-infused apps over the next decade?
- Without the ML expertise of the big companies?

Enterprise-Grade ML (EGML)

- What these enterprises want from ML:
  - *“automate it, and don’t get me sued”*

Enterprise-Grade ML (EGML)

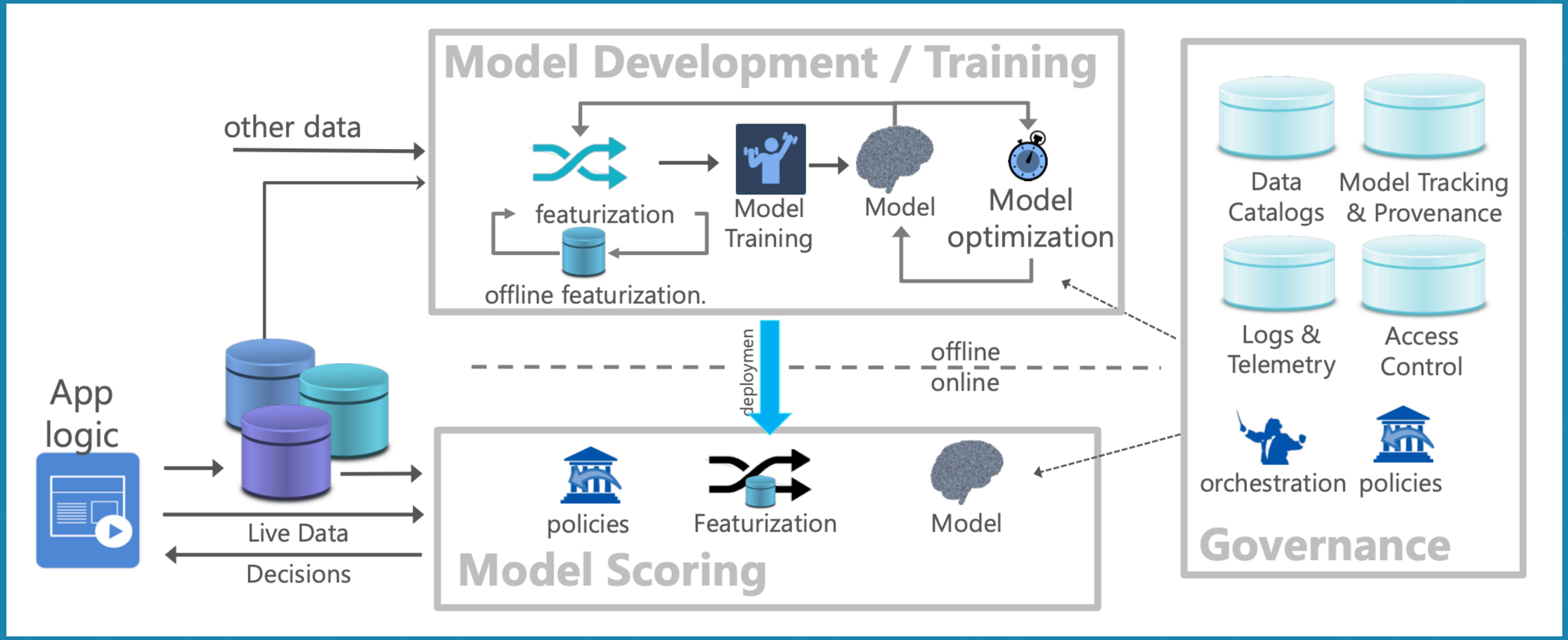
- EGML must satisfy:
  - Privacy regulations
  - Integration with legacy DBMS systems

Will ML be assimilated by the DBMS?

Enterprise-Grade ML (EGML)

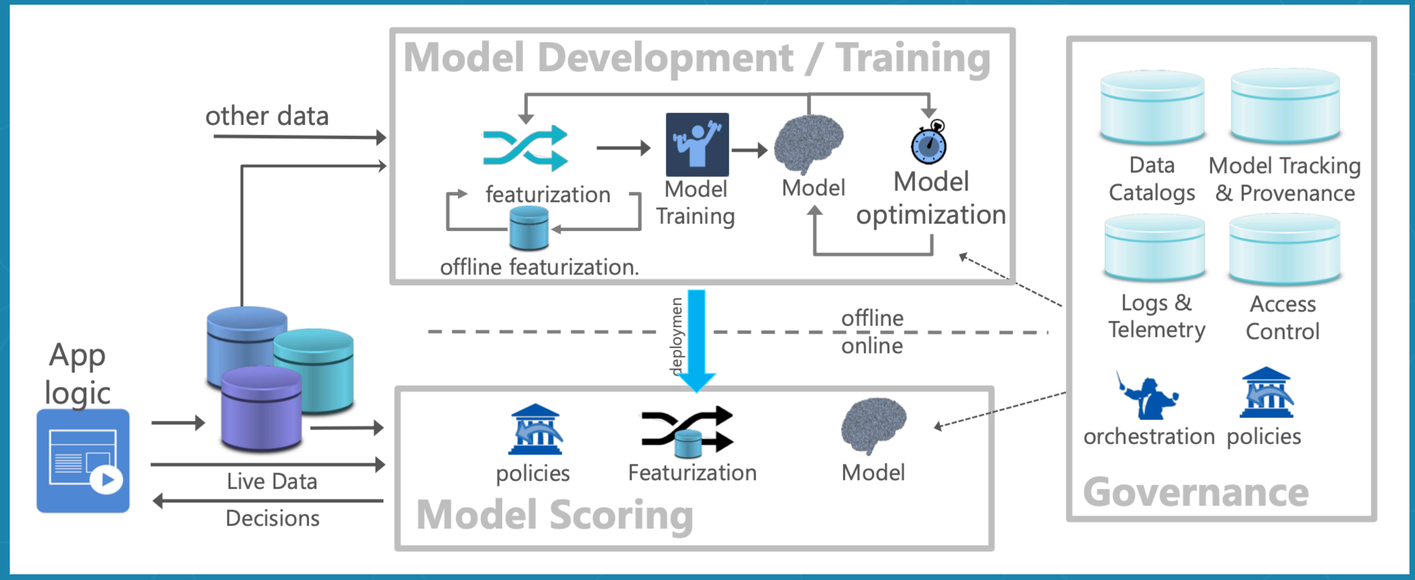
- Three areas:
  - Model development and training
  - Model scoring
  - Model management and governance

“Flock” Reference Architecture



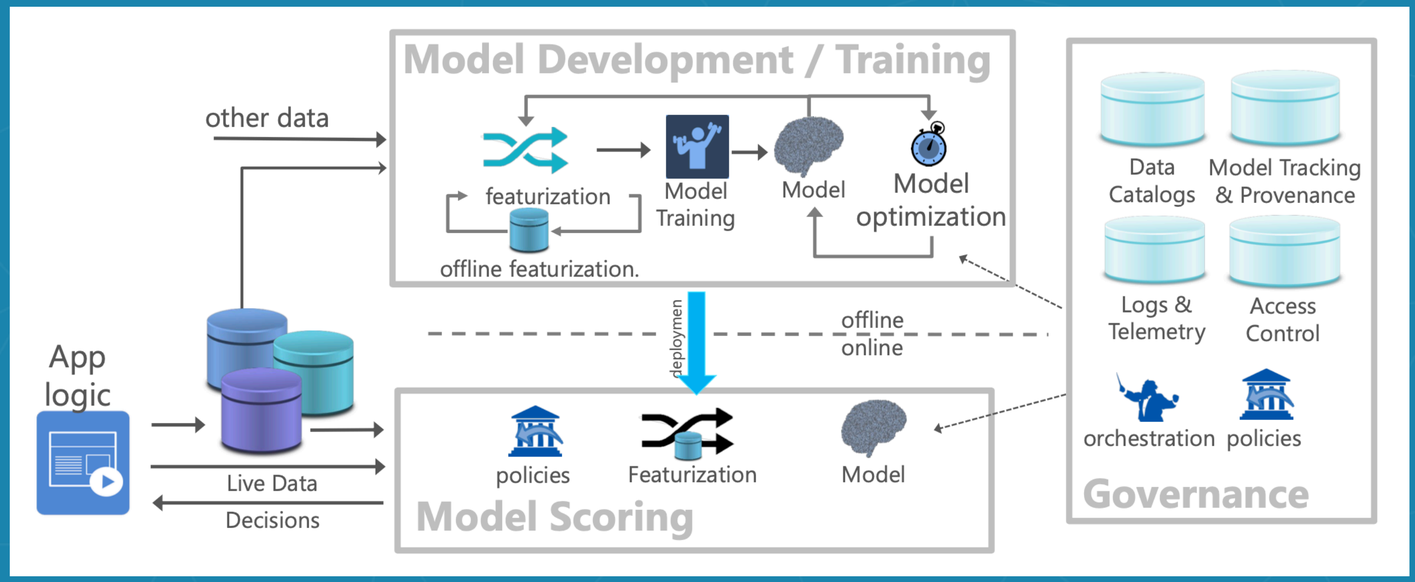
“Flock” Reference Architecture

- First, *models are data* (as we’ll explore)
- ML pipelines are optimizable data flows



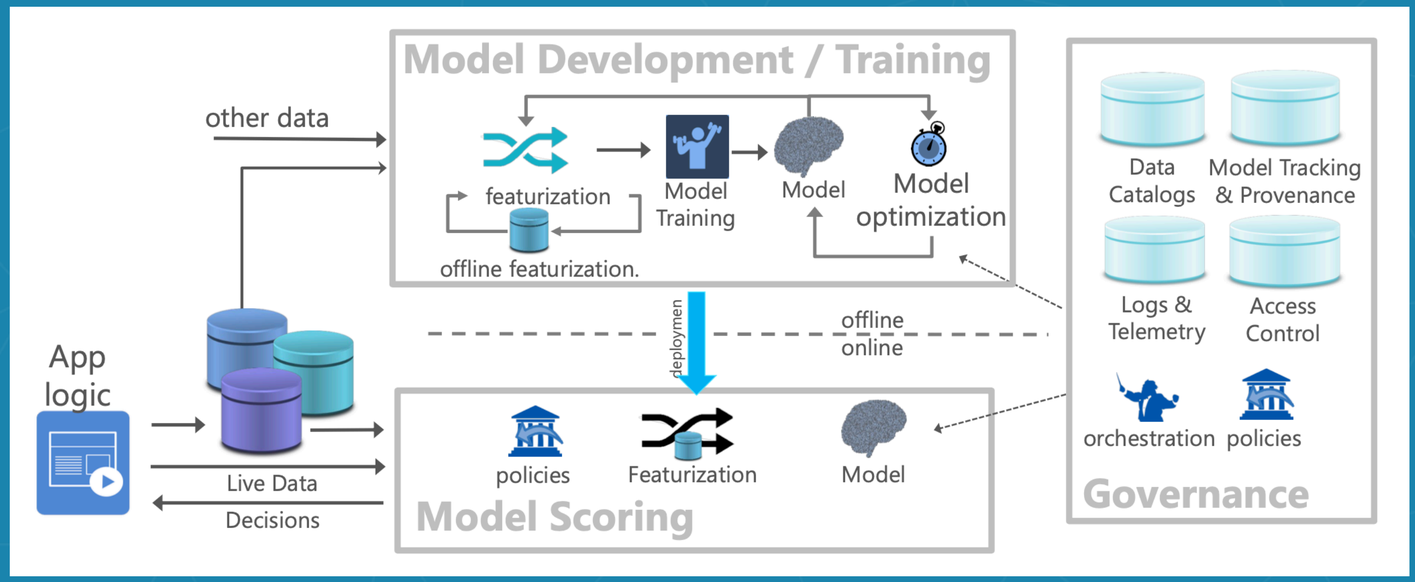
“Flock” Reference Architecture

- Train offline in an environment optimal for training
- Serve online…
  - In the DBMS?



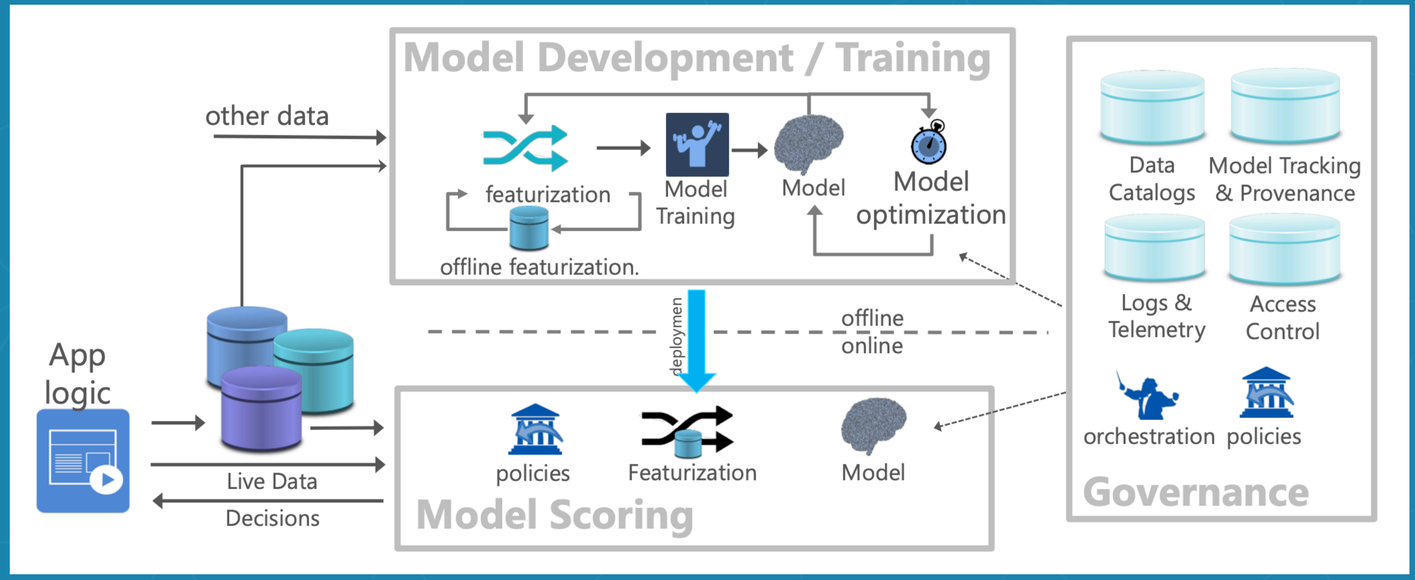
“Flock” Reference Architecture

- The DBMS is best for
  - data governance and management practices
  - Low latency…



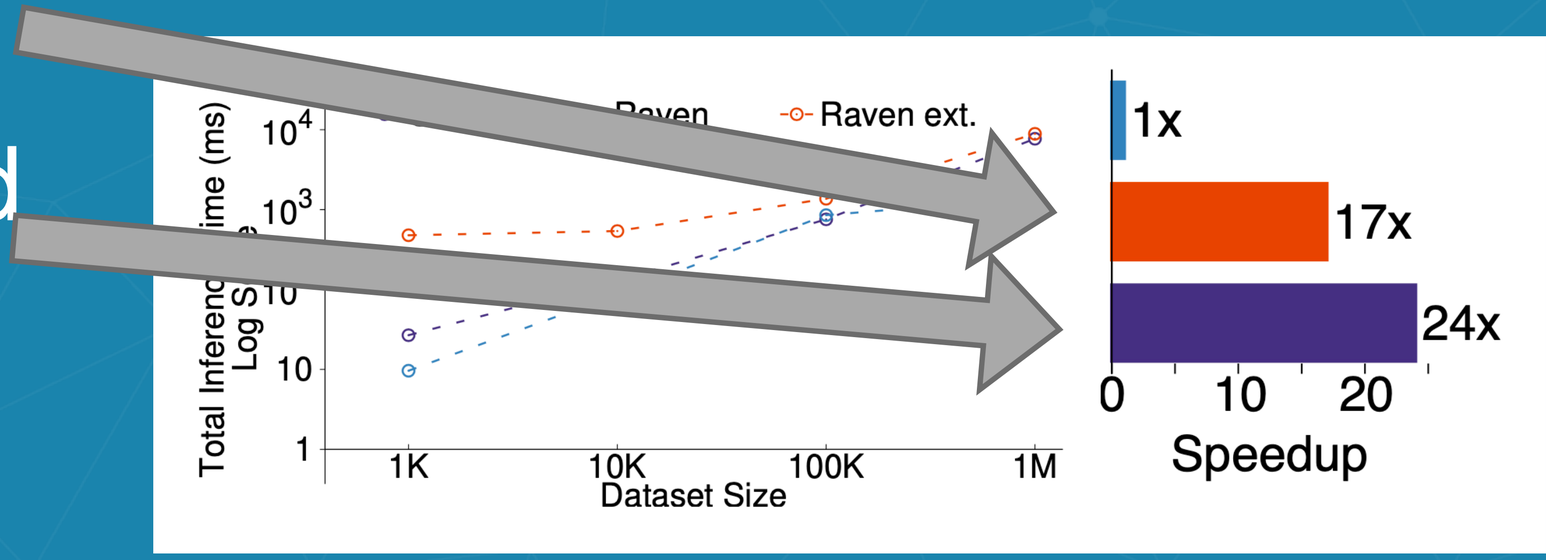
“Flock” Reference Architecture

- Calling an inference service is expensive.
- Use a stored procedure instead?



“Flock” Reference Architecture

- They integrated ONNX runtime into SQL Server:
  - Model inlining
  - Predicate-based model pruning



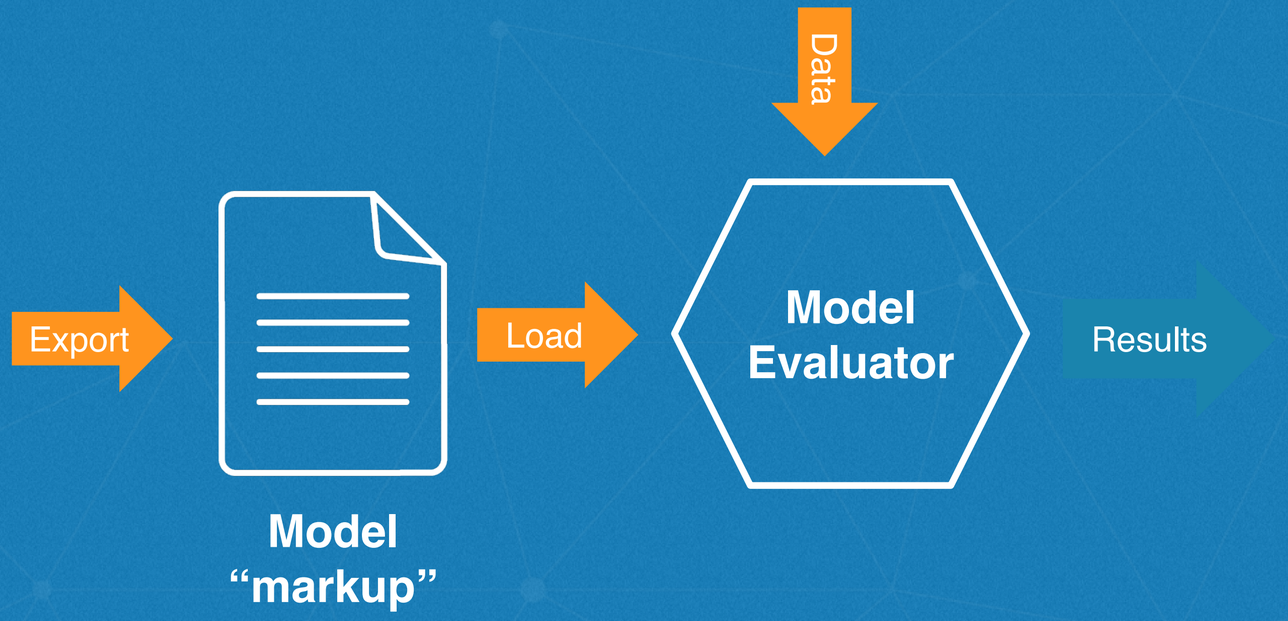
Models Are Data

Models Are Data?

- A machine learning model essentially consists of a given algorithm + weights + hyper parameters.
- Weights and parameters are data, not code, while the algorithm of any particular model is a reference to a static library.
- From this perspective, treating models as data is a more natural fit than creating packaging semantics around weights + hyper parameters.

Source: https://towardsdatascience.com/deploying-machine-learning-models-as-data-not-code-omega-ml-8825a0ae530a

Models As Data - One Way to Do It



Standards:

PMML Predictive Model Markup Language

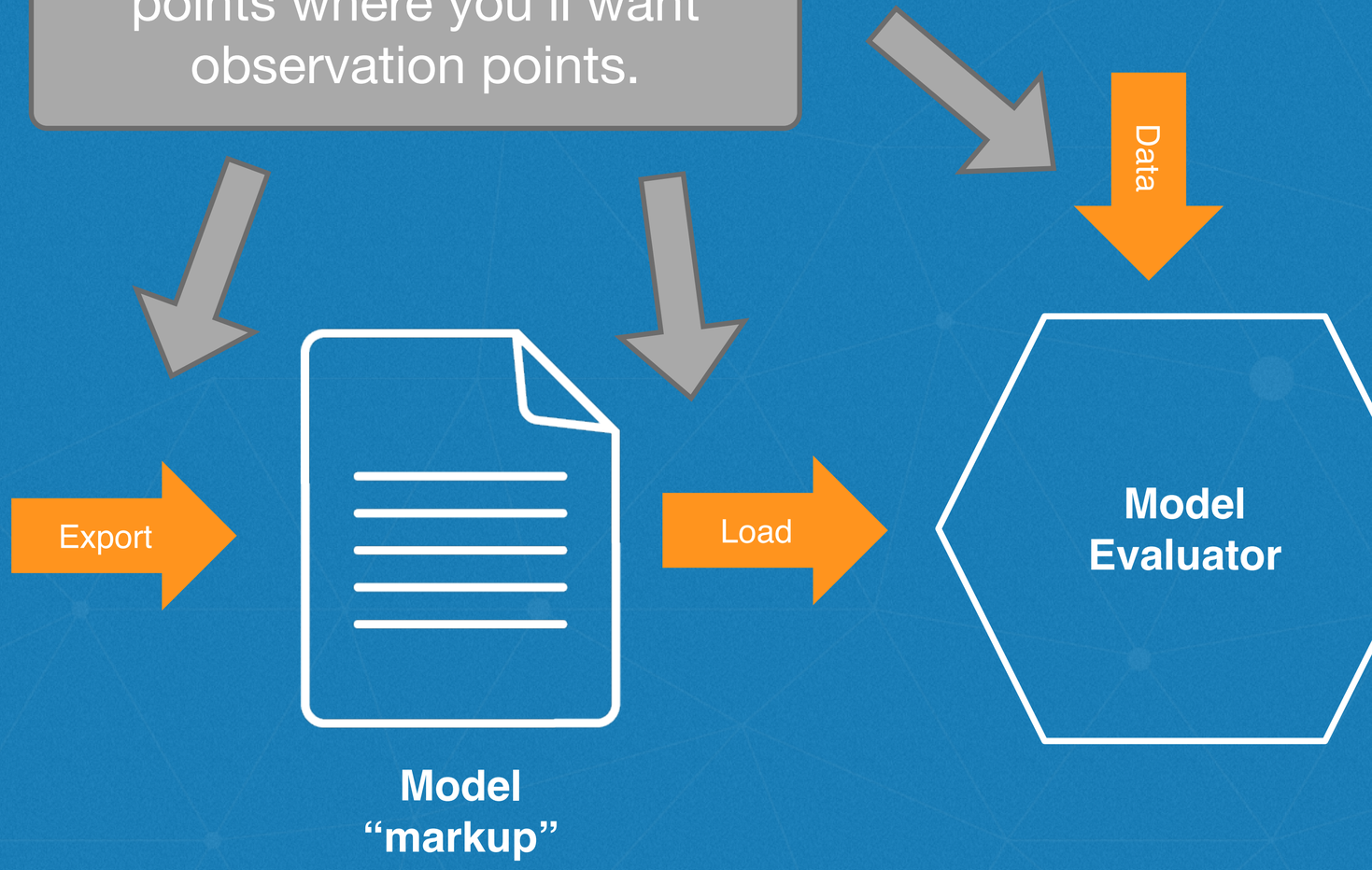
Portable Format for Analytics **(PFA)**

ONNX

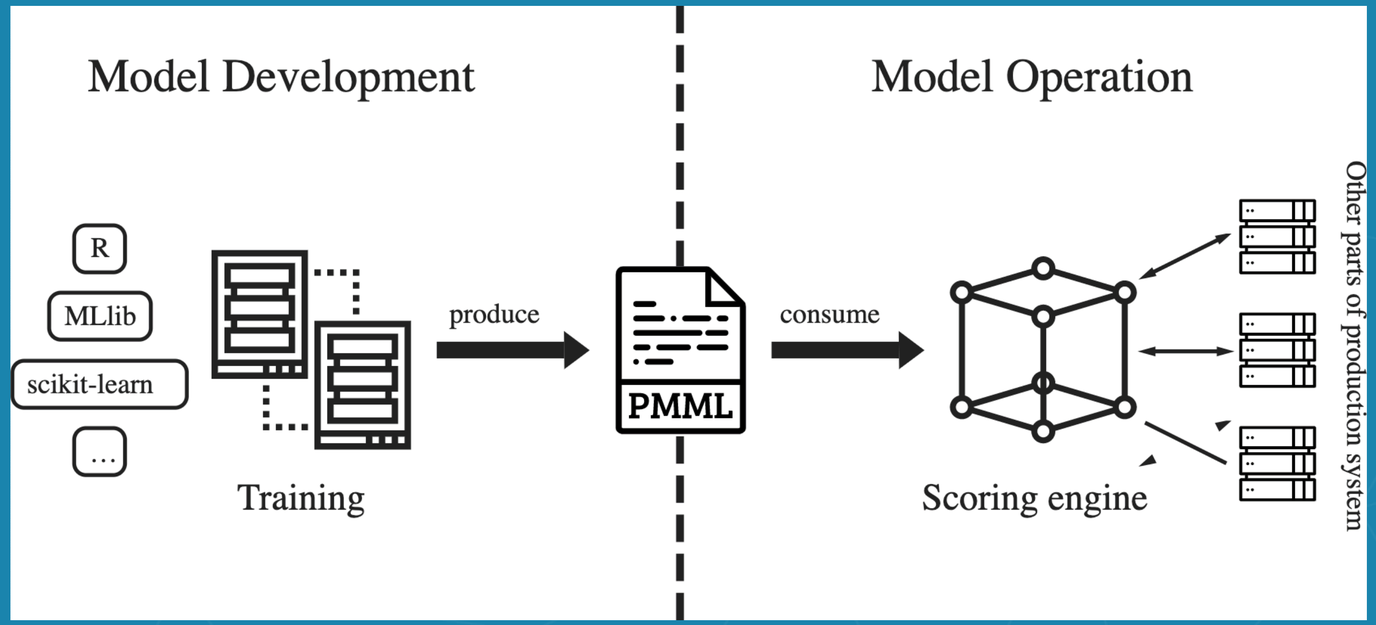
TensorFlow

- When exported?
- When loaded?
- Id of model instance
- Hyper parameters
- Parameters
- Training data set
- …

Since model governance requires traceability, here are points where you'll want observation points.



PMML



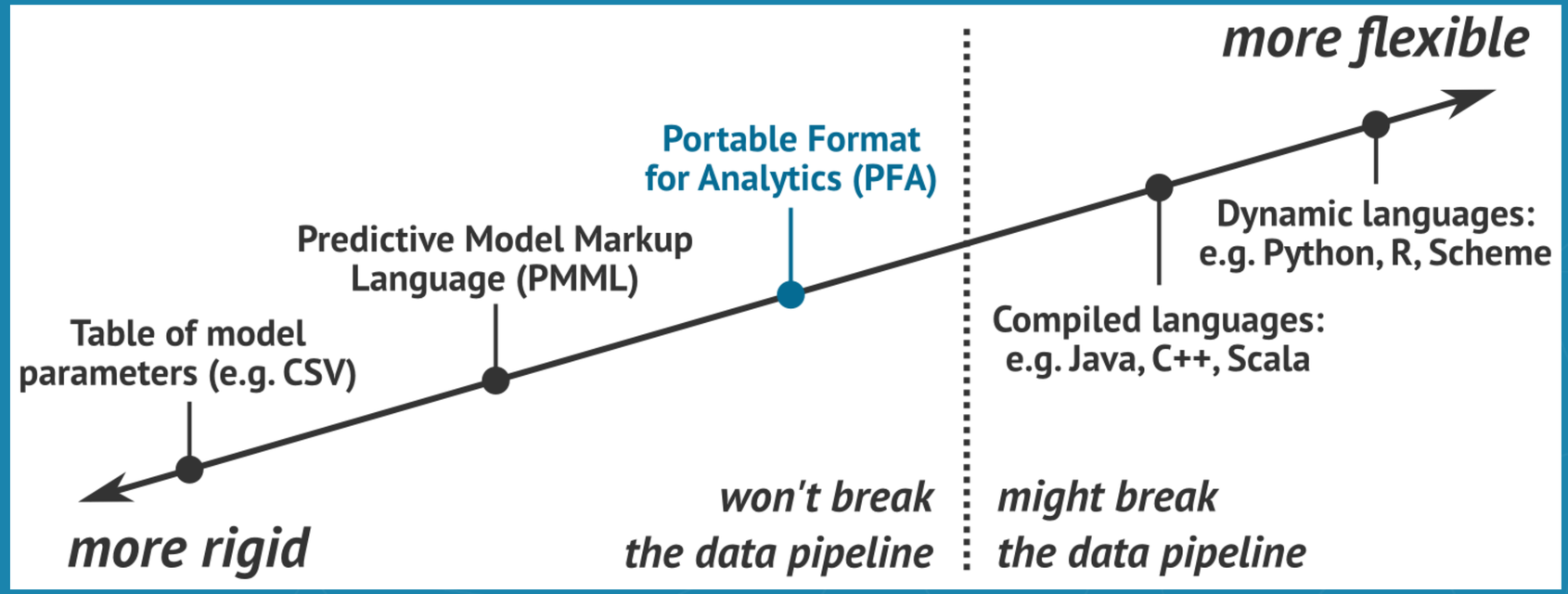
Predictive Model Markup Language (PMML) is an XML-based language that enables the definition and sharing of predictive models between applications.

Implementations for:

Java (JPMML), R, Python Scikit-Learn, Spark here and here, …

Source: https://www.wismutlabs.com/blog/agile-data-science-2/

PFA



Portable Format for Analytics (PFA) is an emerging standard for statistical models and data transformation engines. PFA combines the ease of portability across systems with algorithmic flexibility: models, pre-processing, and post-processing are all functions that can be arbitrarily composed, chained, or built into complex workflows.

Implementations for:

- Java (Hadrian), R (Aurelius), Python (Titus), Spark (Aardpfark), …

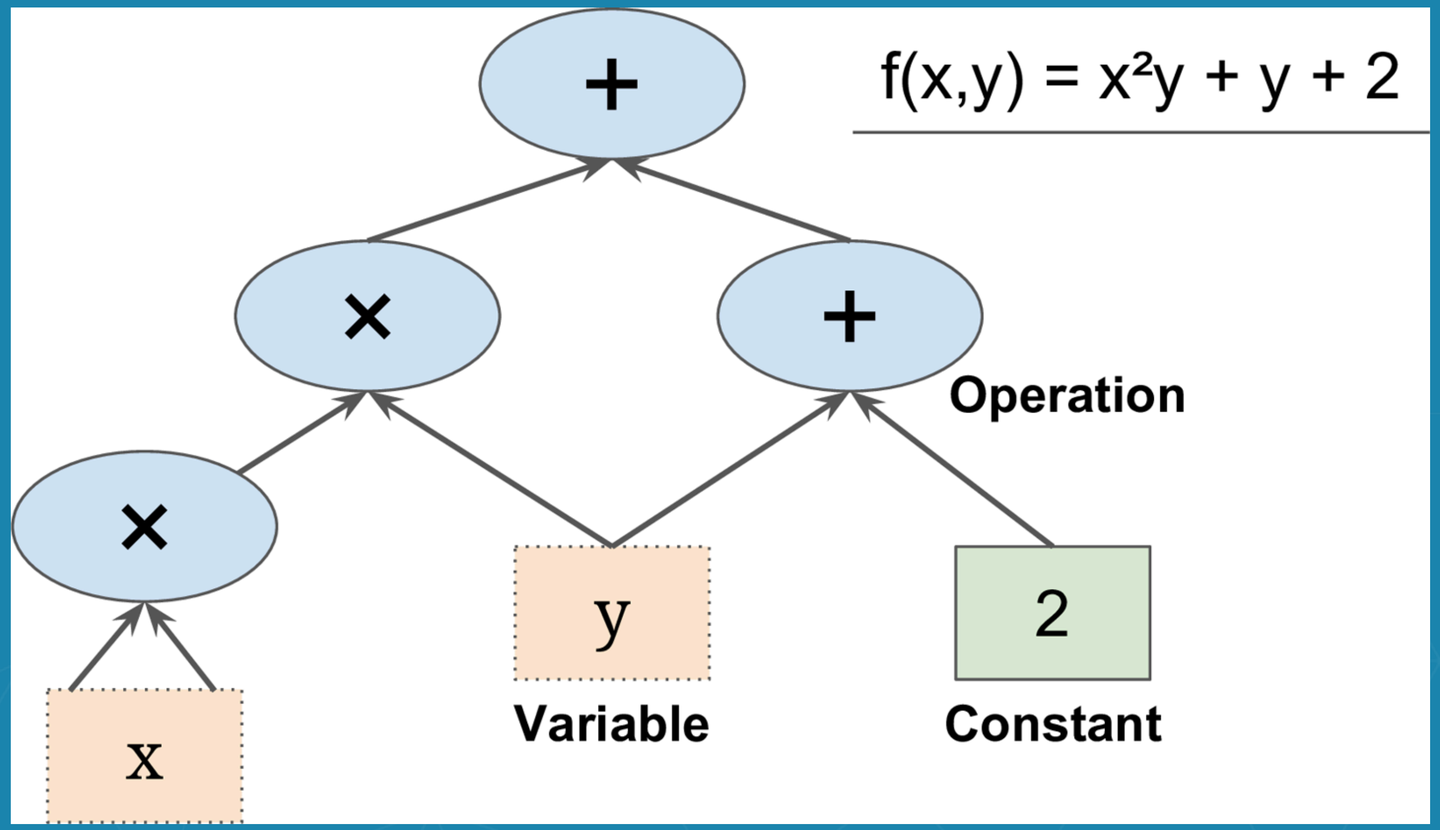
Source: http://dmg.org/pfa/docs/motivation/

ONNX

Open Neural Networks Exchange (ONNX) is an open standard format of machine learning models to offer interoperability between various AI frameworks. Led by Facebook, Microsoft, and AWS.

Source:https://azure.microsoft.com/en-us/blog/onnx-runtime-for-inferencing-machine-learning-models-now-in-preview/

TensorFlow



- A TensorFlow model is represented as a computational graph of Tensors.
- Tensors are defined as multilinear functions which consist of various vector variables. (i.e., matrices of more than two dimensions)
- TensorFlow supports exporting graphs in the form of binary protocol buffers

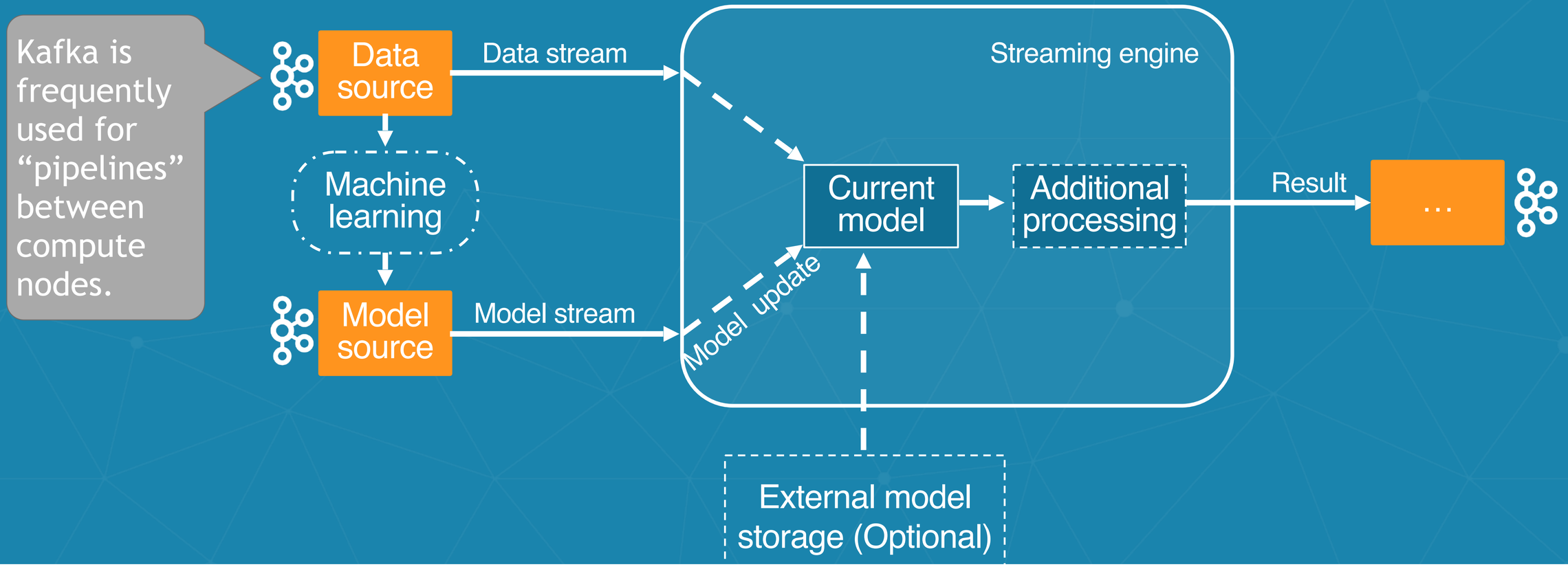
Source: https://learning.oreilly.com/library/view/hands-on-machine-learning/9781491962282/ch09.html

Considerations for Model as Data

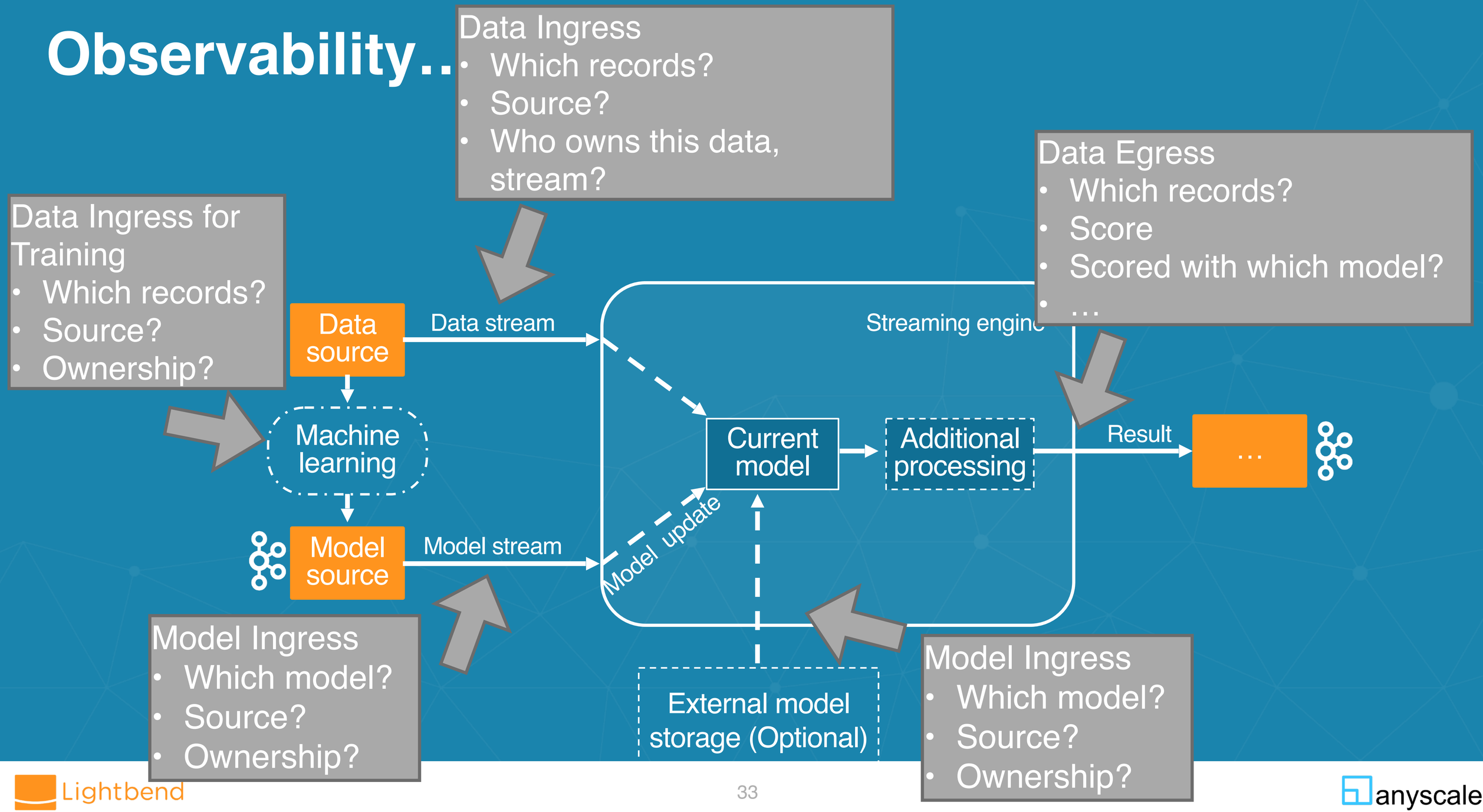
- For *training* and *serving* (*inferencing*) environments are different:
  - Your *training* tools must support *exporting* with PMML, PFA, etc.
  - Your *serving* tools must support the same format for *import*
  - Both ends must support the desired model types
    - E.g., random forests, neural networks, etc.?
  - Does the *serving* implementation faithfully reproduce the results of your *training* environment?

Model Serving for Models as Data

Dynamically Controlled Stream: a streaming system supporting model updates without interruption of execution.



Observability…



Observa

Even if you use an “all-in-one” system, like TensorFlow + TensorFlow serving, you’ll want to observe similar transition events.

Tools for Model Governance

Tools for Data Governance

- Every organization needs customization
- Many organizations want governance capabilities with a minimum amount of programming
  - Hence, there are lot of commercial vendors…
  - https://towardsdatascience.com/top-5-data-governance-framework-tools-to-look-out-for-8d753ab314de

Tools for Data Governance

- We can’t cover all the commercial tools
  - (You should really look at them yourself…)
- Instead, we’ll show how several general-purpose, open-source tools can be used to provide some of the functionality you might want.

Exercise 1: Model Serving

Introducing Cloudflow

Cloudflow boosts development productivity by allowing you to focus only on core business logic. All boilerplate – serialization, port configuration, operational parameters, data durability between processing staging – is handled for you.

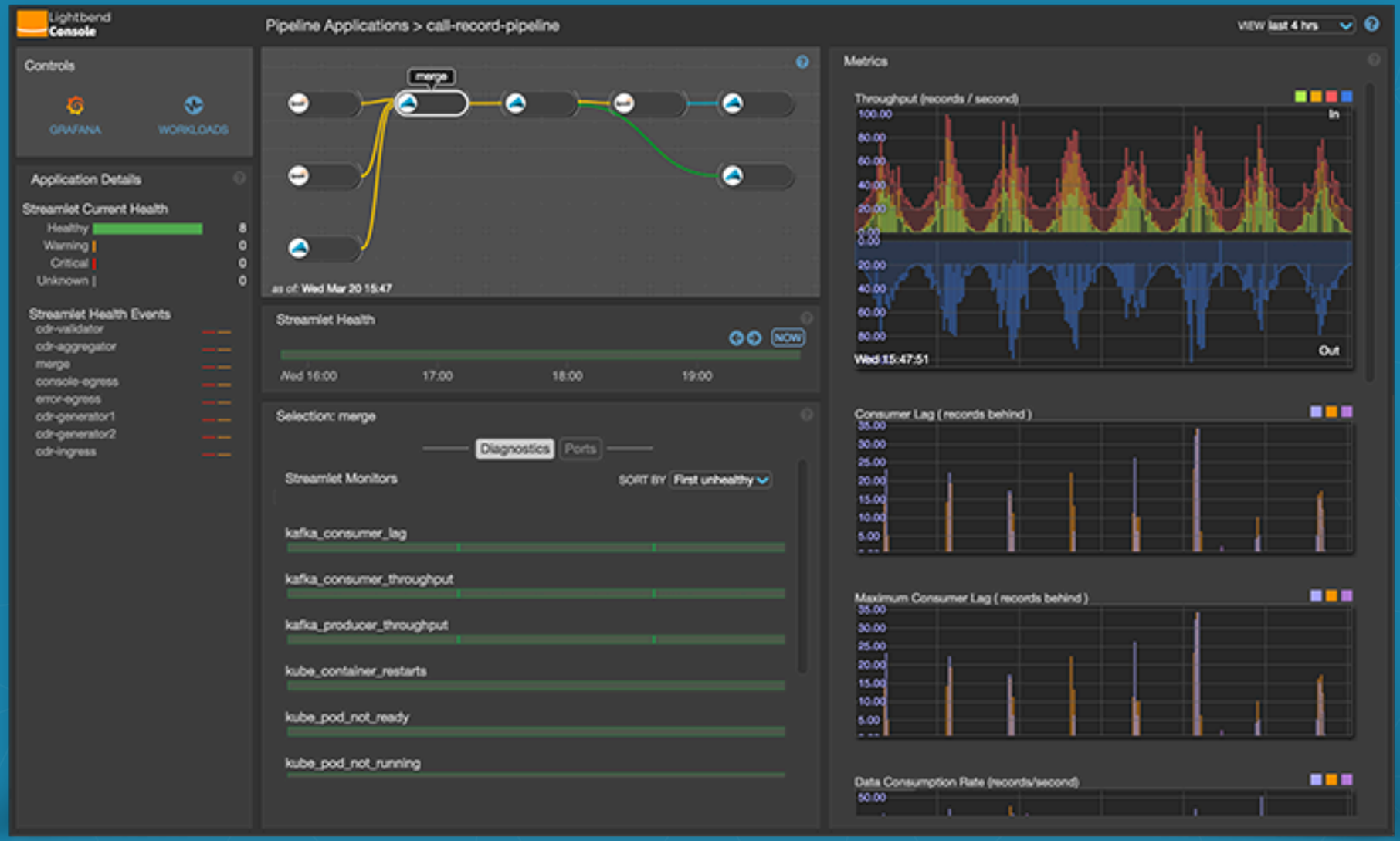
Supported runtimes include Akka Streams, Flink and Spark structured streaming.

Local execution for simplified testing.

Automated deployment to the cloud - just specify your data flow via simple blueprint file and deploy your multi-stage pipeline with one command. All ports and topics are created for you Automatically surface HTTP service endpoints.

Monitoring and management - unique custom UI provides visibility into pipelines so you can pinpoint and troubleshoot problems quickly, spot latency and performance bottlenecks, and leverage custom metrics to relate technical issues to business performance.

Introducing Cloudflow



Exercise - Implement inference for models as data

- Let’s start by walking through the project code…

Exercise - Implement inference for models as data

- See the project README for model serving instructions:
  - “Model Serving with Cloudflow”

Data vs. Model Governance

Data vs. Model Governance

- So, let's start with data governance (DG):
  - Why DG?
  - The goals and benefits of DG
- Then discuss enabling technologies and what's unique about model governance.

Data Governance

- Data governance (DG) is the overall management of the availability, usability, integrity and security of data used in an enterprise.

Source: https://searchdatamanagement.techtarget.com/definition/data-governance

Data Governance - Business Benefits (1/3)

- Ensures data is consistent and trustworthy.
- This is critical as more organizations rely on data analytics to make business decisions, optimize operations, create new products and services, improve profitability and minimize risks.

Source: https://searchdatamanagement.techtarget.com/definition/data-governance

Data Governance - Business Benefits (2/3)

- Data is a core asset for success.
- A goal of Digital Transformation is better exploitation of data.
- Governance is a critical part, but it must fit your organization and your future objectives and business models.

Source: https://profisee.com/data-governance-what-why-how-who/

Data Governance - Business Benefits (3/3)

- Your DG framework must:
  - Define and control data standards needed
  - Delegate the required roles and responsibilities within your organization
  - Fit the business ecosystem where your company operates.

Source: https://profisee.com/data-governance-what-why-how-who/

Without Effective Data Governance

- Data inconsistencies in different systems across an organization might not get resolved.
- Regulatory compliance initiatives might fail:
  - e.g., compliance with new data privacy and protection laws.

Source: https://searchdatamanagement.techtarget.com/definition/data-governance

Data Governance Goals and Benefits (1/3)

- Break down data silos in an organization, which develop when individual business units deploy separate data systems without centralized coordination or an enterprise data architecture.

Source: https://searchdatamanagement.techtarget.com/definition/data-governance

Data Governance Goals and Benefits (2/3)

- Harmonization of data in those systems through a collaborative process, with stakeholders from the various business units participating.

Source: https://searchdatamanagement.techtarget.com/definition/data-governance

Data Governance Goals and Benefits (3/3)

- Ensure that data is used properly:
  - To avoid introducing data errors.
  - To block misuse of personal information and other sensitive data.
  - To meet regularity requirements.

Source: https://searchdatamanagement.techtarget.com/definition/data-governance

Avoiding ML Failures

Failure Modes in Machine Learning

Ram Shankar Siva Kumar*, David O'Brien#, Kendra Albert^, Salome Viljoen#, Jeffrey Snover*

ram.shankar@microsoft.com; jsnover@microsoft.com
*Microsoft

dobrien@cyber.harvard.edu; sviljoen@cyber.harvard.edu
#Berkman Klein Center for Internet and Society at Harvard University

kalbert@law.harvard.edu
^Harvard Law School

Introduction & Background

In the last two years, more than 200 papers have been written on how machine learning (ML) can fail because of adversarial attacks on the algorithms and data; this number balloons if we were to incorporate papers covering non-adversarial failure modes. The spate of papers has made it difficult for ML practitioners, let alone engineers, lawyers, and policymakers, to keep up with the attacks against and defenses of ML systems. However, as these systems become more pervasive, the need to understand how they fail, whether by the hand of an adversary or due to the inherent design of a system, will only become more pressing. The purpose of this document is to jointly tabulate both of these failure modes in a single place.

- *Intentional failures* where the failure is caused by an active adversary attempting to subvert the system to attain her goals – either to misclassify the result, infer private training data, or to steal the underlying algorithm.
- *Unintentional failures* where the failure is because an ML system produces a formally correct but completely unsafe outcome

We would like to point out that there are other taxonomies and frameworks that individually highlight intentional failure modes[1,2] and unintentional failure modes.[3,4] Our classification brings the two separate failure modes together in one place and addresses the following needs:

1) The need to equip software developers, security incident responders, lawyers, and policy makers with a common vernacular to talk about this problem. After developing the initial version of the taxonomy last year, we worked with security and ML teams across Microsoft, 23 external partners, standards organization, and governments to understand how stakeholders would use our framework. Based on this usability study and stakeholder feedback, we iterated on the framework.
   Result: When presented with an ML failure mode, we frequently observed that software developers and lawyers mentally mapped the ML failure modes to traditional software attacks like data exfiltration. So, throughout the paper, we attempt to highlight how machine learning failure modes are

Source: https://docs.microsoft.com/en-us/security/failure-modes-in-machine-learning

Intentional Failures

- Perturbation attack - modifying the query
- Poisoning attack - corrupt the training method or data
- Model inversion - recover feature information from a model
- …

Intentional Failures

- Record recovery - reconstruct sensitive data from the model
- Model stealing - reconstruct the model itself from query results
- Corrupt the ML pipeline
- … and others

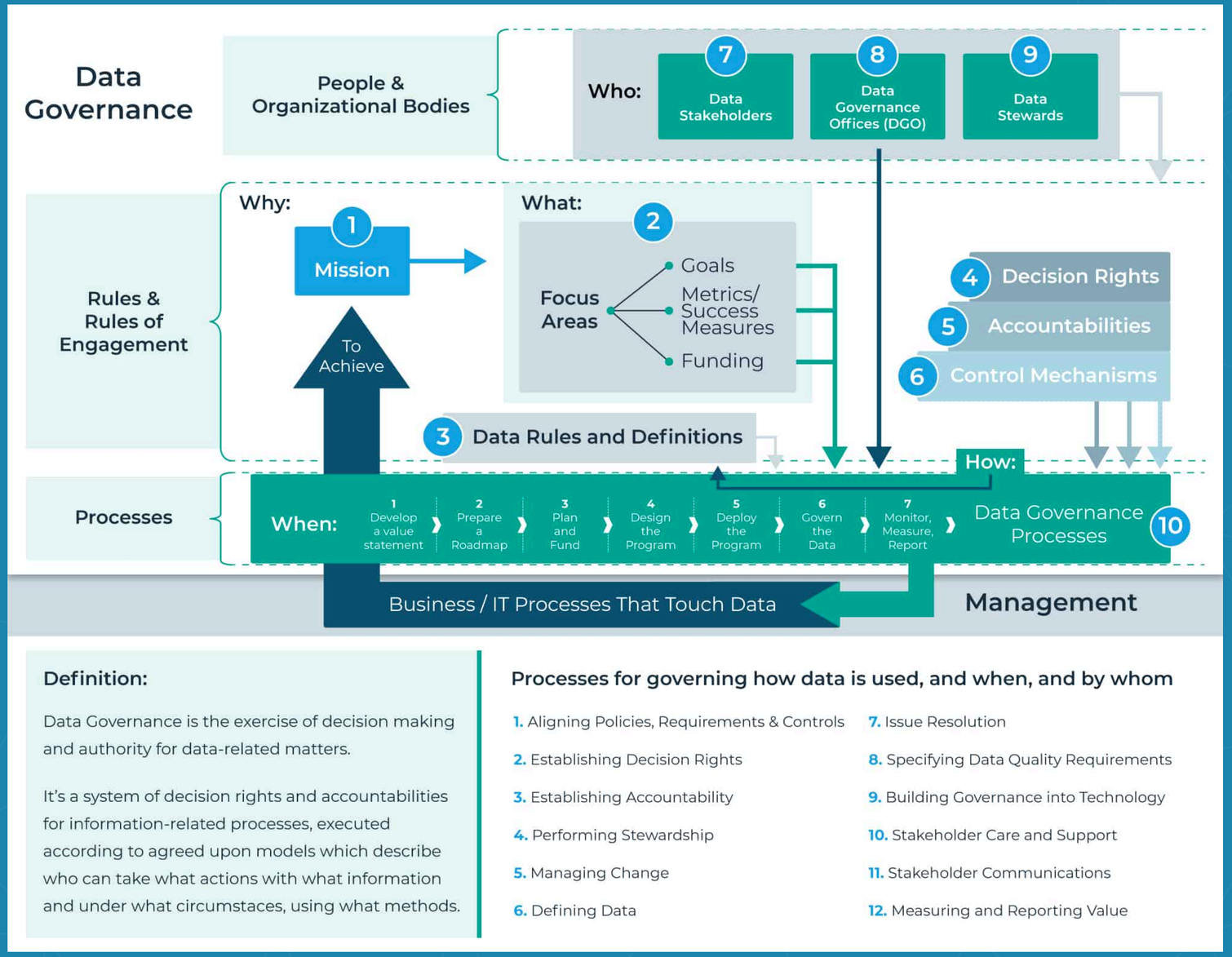
Unintentional Failures

- Side effects - Unintended effects of system behavior driven by ML decisions
- Reward hacking (RL) - trained reward model doesn't match real rewards
- Concept drift
- Adversarial examples
- …

A Data Governance Framework

Source: https://www.dama.org/sites/default/files/download/DAMA-DMBOK2-Framework-V2-20140317-FINAL.pdf

Recap, in a Diagram



Definition:

Data Governance is the exercise of decision making and authority for data-related matters.

It's a system of decision rights and accountabilities for information-related processes, executed according to agreed upon models which describe who can take what actions with what information and under what circumstaces, using what methods.

Processes for governing how data is used, and when, and by whom

1. Aligning Policies, Requirements & Controls
2. Establishing Decision Rights
3. Establishing Accountability
4. Performing Stewardship
5. Managing Change
6. Defining Data
7. Issue Resolution
8. Specifying Data Quality Requirements
9. Building Governance into Technology
10. Stakeholder Care and Support
11. Stakeholder Communications
12. Measuring and Reporting Value

Source: https://profisee.com/data-governance-what-why-how-who/

Enabling Technology - Metadata

Metadata: Foundation of Data Governance

- "*Metadata* is a **set of data** that describes and gives information about other **data**."
- Three main components of metadata:
  - Application Context
  - Behavior
  - Change

Source: https://eng.lyft.com/amundsen-lyfts-data-discovery-metadata-engine-62d27254fbb9

Metadata: Application Context

- Information needed by humans or applications to operate. This includes:
  - What data exists
  - Description, semantics, “tags” associated with that data

Source: https://eng.lyft.com/amundsen-lyfts-data-discovery-metadata-engine-62d27254fbb9

Metadata: Behavior

- Information about *lifecycle:*
  - How the data is created and used over time.
  - Provenance and lineage.
  - Ownership and access privileges.
  - Common usage patterns.
  - Log of people or processes that have touched the data.
  - Frequent users.

Source: https://eng.lyft.com/amundsen-lyfts-data-discovery-metadata-engine-62d27254fbb9

Metadata: Change

- Information about how the data changes over time.
  - How does the schema evolve?
  - What processes are used in this evolution?

Source: https://eng.lyft.com/amundsen-lyfts-data-discovery-metadata-engine-62d27254fbb9

Model Metadata

- In addition to data governance…

Model Metadata (1/2)

- In addition to data governance…
- “Static” metadata per model
  - Kind (e.g., logistic regression, CNN, …)
  - Hyper Parameters (e.g., structure of NN)
  - Parameters (e.g., weights in NN)
  - Data set used for training
  - When trained, how trained, …

Model Metadata (2/2)

- In addition to data governance…
- “Dynamic” metadata per model
  - When deployed
  - Used to score which records
  - Performance statistics
    - Time to score
    - Confidence level

Metadata Repositories

- Several Open Source Examples:
  - LinkedIn WhereHows
  - Lyft Amundsen
  - Apache Atlas.
- Let's look at each one…

Source: https://eng.lyft.com/amundsen-lyfts-data-discovery-metadata-engine-62d27254fbb9

LinkedIn WhereHows (1/2)

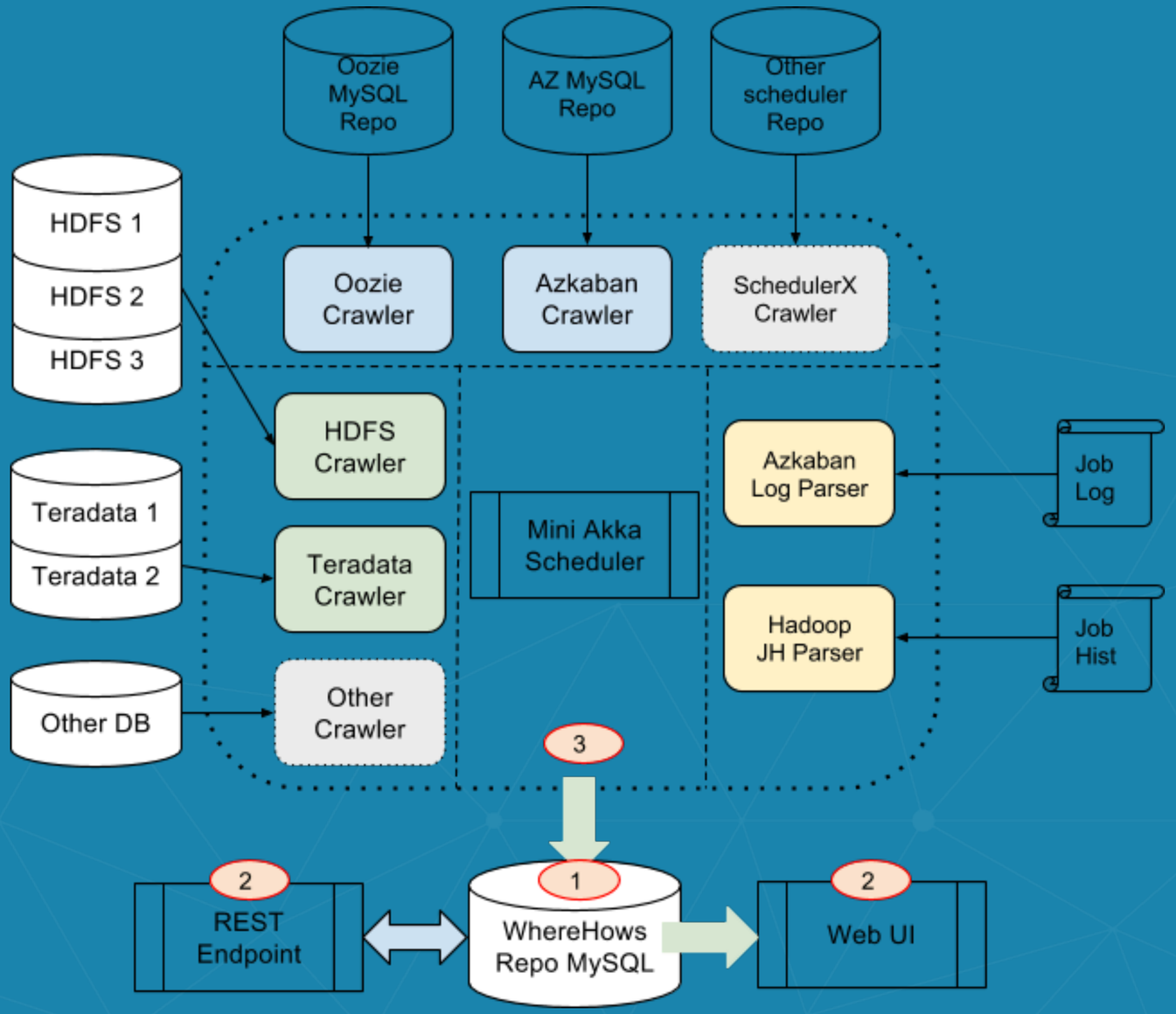
- Central metadata repository to capture metadata across diverse systems.
  - Surfaces it through a one platform for simple data and flow discovery.
  - Track data and process:
    - Lineage
    - Ownership
    - Evolution history
  - One place for schema discovery.

Source: https://engineering.linkedin.com/blog/2016/03/open-sourcing-wherehows--a-data-discovery-and-lineage-portal

LinkedIn WhereHows (2/2)

- Collects the following types of metadata:
  - **Catalog information**: schemas, datasets physical location, timestamp of create/modify, ownership, etc.
  - **Operational metadata**: jobs, flows, and execution information.
  - **Lineage metadata**: connects jobs and datasets.

Source: https://engineering.linkedin.com/blog/2016/03/open-sourcing-wherehows--a-data-discovery-and-lineage-portal



Source:https://engineering.linkedin.com/blog/2016/03/open-sourcing-wherehows--a-data-discovery-and-lineage-portal

Lyft Amudsen (1/2)

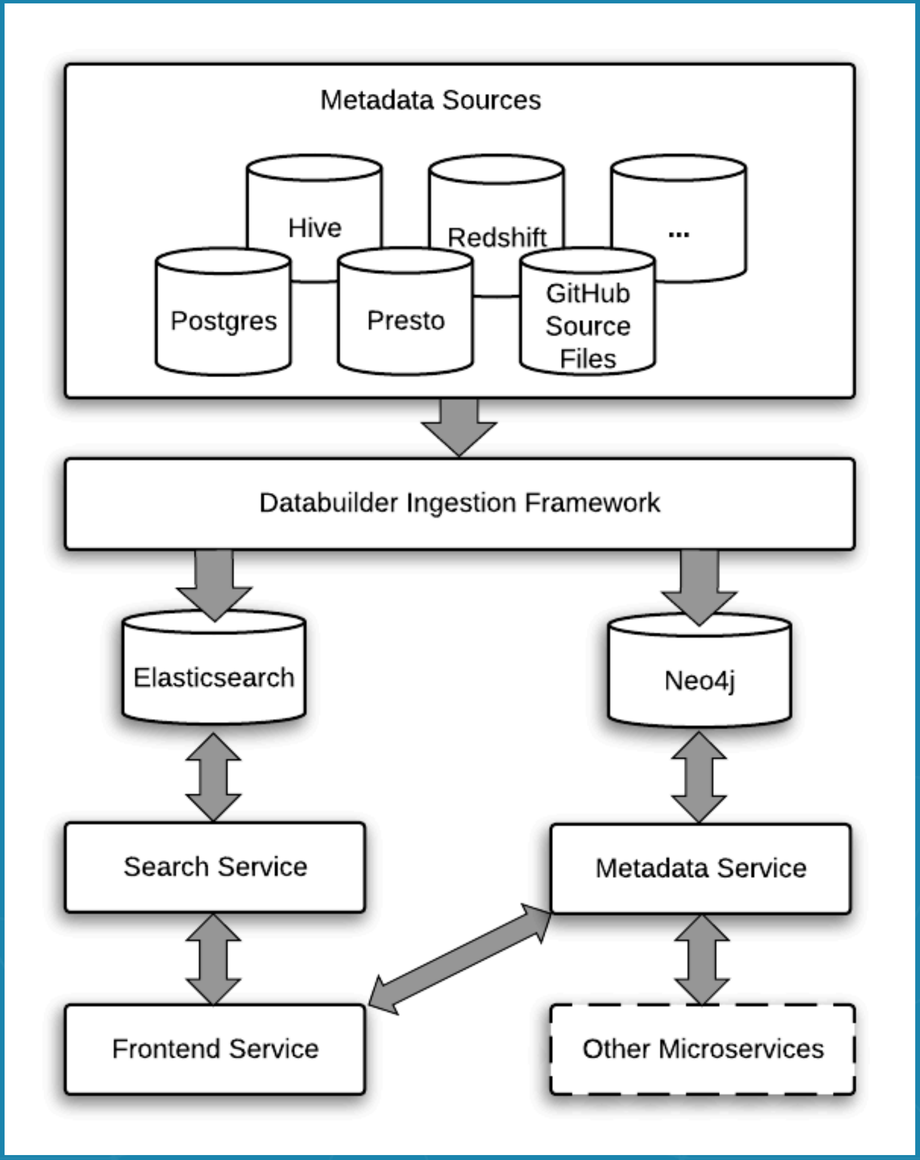
- **Data Stores** - tables, schemas, documents of structured data stores like Hive, Presto, MySQL, unstructured data stores, like S3, Google Cloud Storage, etc.
- **Dashboards/reports** - saved queries, reports and dashboards in BI/reporting tools like Tableau, Looker, Apache Superset, etc.
- **Events/Schemas** - Events and schemas stored in schema registries or tools like Segment.

Source:https://eng.lyft.com/amundsen-lyfts-data-discovery-metadata-engine-62d27254fbb9

Lyft Amudsen (2/2)

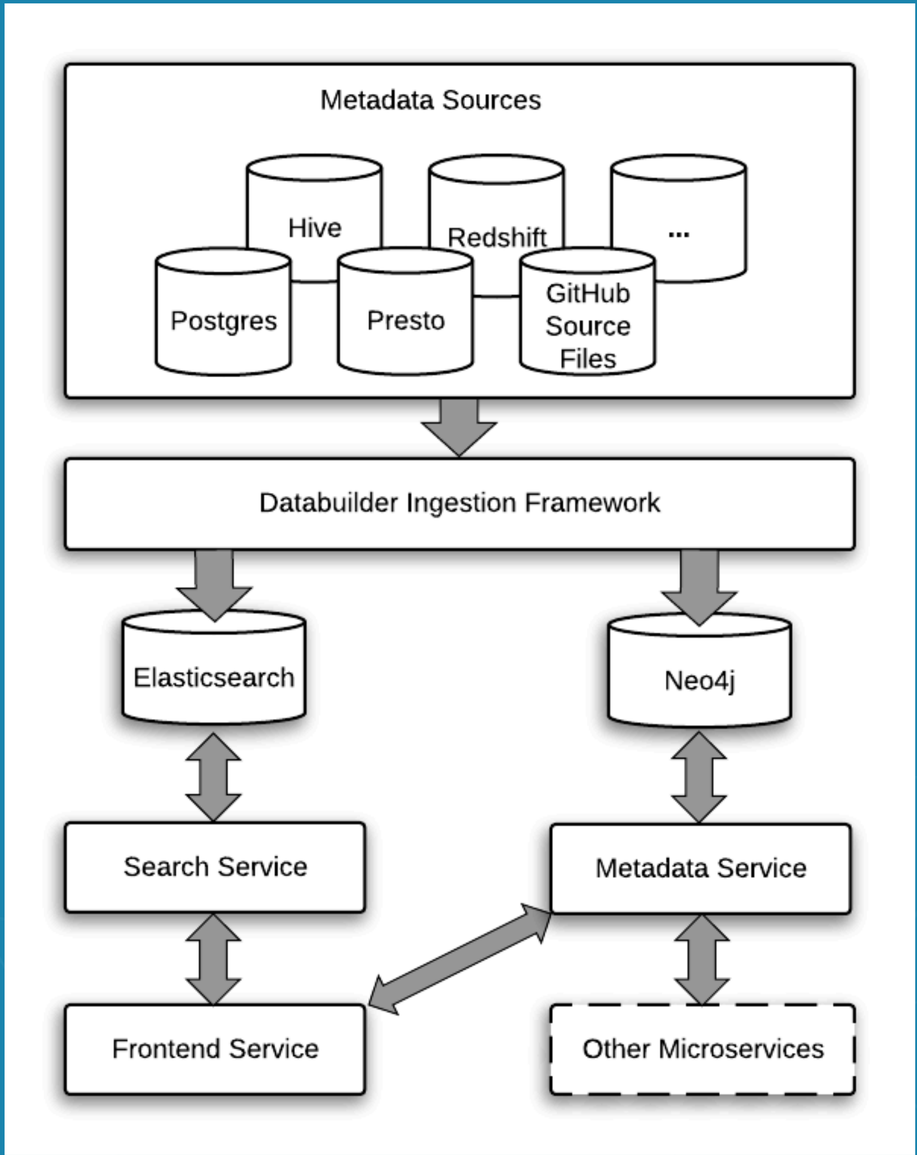
- **Streams** - Streams/topics in Apache Kafka, AWS Kinesis, etc.
- **Processing** - ETL jobs, ML workflows, streaming jobs, etc.
- **People** - name, team, title, data resources frequently used, data resources bookmarked, etc.

Source:https://eng.lyft.com/amundsen-lyfts-data-discovery-metadata-engine-62d27254fbb9



- Metadata services handles metadata requests from the UI, other micro services.
- Persistent layer is Neo4j, but alternatives can be used.
- Search service is backed by Elasticsearch, handles requests from UI.
- Search engine uses ElasticSearch, but alternatives can be used.

Source:https://eng.lyft.com/open-sourcing-amundsen-a-data-discovery-and-metadata-platform-2282bb436234



- Front-end service hosts Amundsen's web application.
- Databuilder is a generic data ingestion framework which extracts metadata from various sources.
- Common is a library repo which holds common code shared among all micro services in Amundsen.

Source:https://eng.lyft.com/open-sourcing-amundsen-a-data-discovery-and-metadata-platform-2282bb436234

Apache Atlas (1/2)

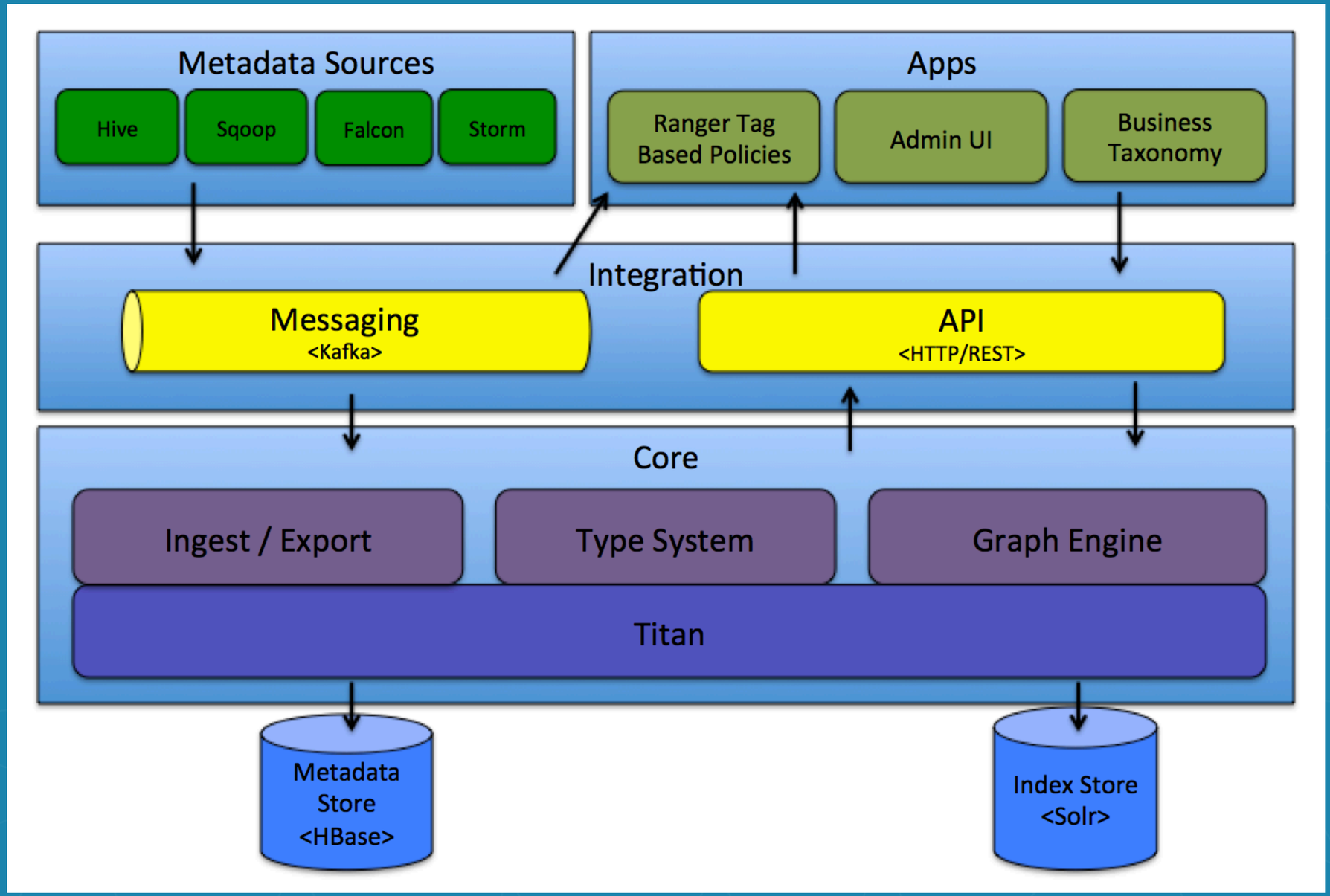
- A scalable and extensible set of core foundational governance services.
  - Enables enterprises to effectively and efficiently meet their compliance requirements
  - Hadoop focused.
  - Allows integration with the whole enterprise data ecosystem.

Source:https://blog.knoldus.com/apache-atlas/

Apache Atlas (2/2)

- Can define new metadata types
- Facilitates easy exchange of metadata by enabling any metadata consumer to share a common metadata store.
- Can dynamically create classifications (e.g., PII, EXPIRES_ON, DATA_QUALITY, SENSITIVE).
  - Classifications can include attributes (e.g., expiry_date attribute in EXPIRES_ON classification).
- An Intuitive UI to view the lineage of data as it moves through various processes, plus a REST APIs to access and update lineage.

Source:https://blog.knoldus.com/apache-atlas/


Metadata Sources
Hive
Sqoop
Falcon
Storm
Apps
Ranger Tag Based Policies
Admin UI
Business Taxonomy
Integration
Messaging
<Kafka>
API
<HTTP/REST>
Core
Ingest / Export
Type System
Graph Engine
Titan
Metadata Store
<HBase>
Index Store
<Solr>

We'll explore an Apache Atlas example later today.

What Makes Model Metadata Unique?

What makes models special

- **Model quality** - with software you can write a limited set of tests that will test each of your things. Here you're suddenly moving from a yes-no answer to this probabilistic
- **Model correctness** - because the models usage results are probabalistic, one need model explanation to reason about its correctness

Source:https://qconsf.com/sf2019/presentation/mls-hidden-tasks-checklist-developers-when-build

Special Requirements for Models

- Reproducibility
- Data vs. Concept Drift
- Model Explainability and Interpretability

- Let’s look at each one…

Reproducibility in ML Is Critical

- Without it,
  - Claimed improvements from changing one parameter could actually be due to hidden sources of randomness.
  - Variations when rerunning failed jobs or prior experiments can’t be eliminated.
  - Essential for fault tolerance and iterative refinement of models.

Source: https://determined.ai/blog/reproducibility-in-ml/

Reproducibility in ML Is Critical

- Without it,
  - As more sophisticated models and real-time data streams push us towards distributed training, CPU/GPU clusters, reliable model construction becomes even harder.
  - It's difficult to debug problems, understand model behavior, etc.

Source: https://determined.ai/blog/reproducibility-in-ml/

Nondeterminism

- One way in which ML models are different from most other software artifacts is the inherent *nondeterminism* of model training.
- Let's look at causes…

Source: https://determined.ai/blog/reproducibility-in-ml/

Cause: Random Initialization of weights

- When training, it's common to set the initial model weight (i.e., parameter) values by sampling from a particular distribution, with random being popular.
- This improves the speed of convergence compared to initializing all weights to zeros, but it means:

  No two training runs will return the same model!!

Source: https://determined.ai/blog/reproducibility-in-ml/

Cause: Shuffling/Sampling of Data Sets

- Training, testing, and cross-validation data sets are sampled from the full data set available for training.
- Often the training set is shuffled.

Source: https://determined.ai/blog/reproducibility-in-ml/

Cause: Noisy Hidden Layers

- Certain NN architectures include layers with inherent randomness during training. Dropout layers, for example, exclude the contribution of a particular input node with probability p. While this may help prevent overfitting, it means the same input sample will produce different layer activations on any given iteration.

Source: https://determined.ai/blog/reproducibility-in-ml/

Cause: Switching Libraries, Versions

- Behaviors change change:
  - Between ML library versions
  - When migrating a model from one framework to another
- For example, Tensorflow warns its users to "rely on approximate accuracy, not on the specific bits computed" across versions. Keras will exhibit different behaviors if swapping between Theano and Tensorflow backends without taking appropriate steps.

Source: https://determined.ai/blog/reproducibility-in-ml/

Cause: Nondeterministic GPU Floating Point Calculations

- Some functions in cuDNN, the Nvidia Deep NN library for GPUs, do not guarantee reproducibility across runs by default, including several convolutional operations.
- Also, reproducibility is not guaranteed across different GPU architectures unless these operations are forcibly disabled by your ML library.

Source: https://determined.ai/blog/reproducibility-in-ml/

Cause: CPU Multithreading

- For CPU training, TensorFlow by default configures thread pools with one thread per CPU core to parallelize computation. This parallelization happens both within execution of certain individual ops (intra_op_parallelism) as well as between graph operations deemed independent (inter_op_parallelism). While this speeds up training, the existing implementation introduces non-determinism.

Source: https://determined.ai/blog/reproducibility-in-ml/

Approaches to reproducibility

Source of variation	Solution
Inconsistent hyperparameters	Metadata saved for each experiment
Changes in Model architecture	Model architecture saved for each experiment
Random initialization of layer weight	Code changes
Shuffling of the data sets	Code changes
Noisy hidden layers	Code changes
Changes in ML frameworks	Containerization of execution
Non Determenistic GPU floating point calculation	Requires changes in ML frameworks
CPU multi threading	Some frameworks (TF) support control

Source: https://determined.ai/blog/reproducibility-in-ml/

Data vs. Concept Drift

- Data *characteristics* change over time. The *interpretation* of data also evolves.
- Both cause model performance to degrade over time, since they are trained on old data.

Source: https://machinelearningmastery.com/gentle-introduction-concept-drift-machine-learning/

Data vs. Concept Drift

- Data Drift:
  - The data itself is changing over time, e.g., a feature's distribution changes, new discrete values (labels) appear, etc.
  - Model performance will be poor on new data that is very different from the data used for training.
  - But old data (usually) doesn't need rescoring.

Source: https://machinelearningmastery.com/gentle-introduction-concept-drift-machine-learning/

Data vs. Concept Drift

- Concept Drift:
  - Our *interpretation* of the data has changed.
    - e.g., words change meaning over time.
  - Often, the relationship between inputs and output variables is unknown and hidden.
  - Rescoring of past data may be required.

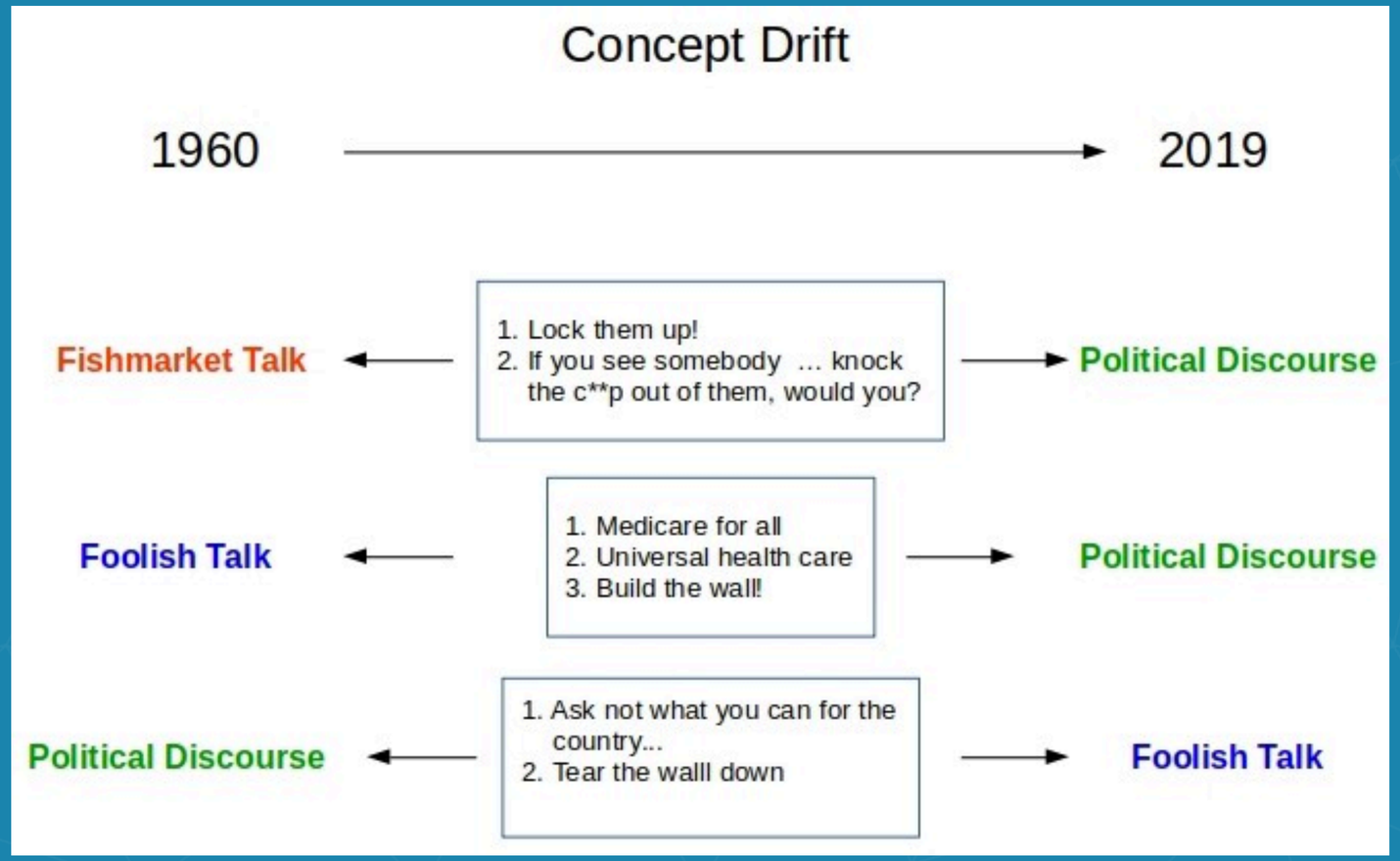
Source: https://machinelearningmastery.com/gentle-introduction-concept-drift-machine-learning/

Concept Drift

- For example, in classification, labels will shift. The decision boundary for new data diverges from the boundary of a model built from earlier data and labels.
- Measuring what happens when scoring randomly sampled new data can be used to detect the drift, allowing us to decide when to retrain the model, which may be expensive.
- Concept drift arises when our interpretation of the data changes over time even while the data may not have.

Source: https://towardsdatascience.com/concept-drift-and-model-decay-in-machine-learning-a98a809ea8d4

Concept Drift Example



Source: https://towardsdatascience.com/concept-drift-and-model-decay-in-machine-learning-a98a809ea8d4

How to Measure Drift?

- Track quality metrics, for example:
  - F1 score for binary classification
  - Confidence metrics

Source: https://machinelearningmastery.com/gentle-introduction-concept-drift-machine-learning/

How to Address Drift?

- Baseline: assume that the data does not change. Then compare performance/cost tradeoff with update techniques.
- Techniques:
  - Periodically update a model incrementally with more recent data vs. retaining from scratch.
  - Discount older data with a weighting mechanism, so retrained model better represents the current situation.
  - Have a suite of models, measure performance and switch to the best one periodically.

Source: https://machinelearningmastery.com/gentle-introduction-concept-drift-machine-learning/

Explainability and Interpretability (1/3)

- **Interpretability**: The degree to which a human can consistently predict a model's result.
- **Explainability:** The degree to which a human understands the root causes of a decision produced by a model.

Sources:
https://www.kdnuggets.com/2018/12/machine-learning-explainability-interpretability-ai.html
https://christophm.github.io/interpretable-ml-book/interpretability.html

Explainability and Interpretability (2/3)

- So, interpretability doesn't require deep knowledge of the "black box", but I should know what will happen if some "knob is turned". It's forward looking, predictive.
- Explainability is backwards and internal looking; can I tell you exactly why a model returned the result it returned for a given datum? It's often the harder challenge.

Sources:
https://www.kdnuggets.com/2018/12/machine-learning-explainability-interpretability-ai.html
https://christophm.github.io/interpretable-ml-book/interpretability.html

Explainability and Interpretability (3/3)

- Interpretability is essential for comprehending why certain decisions or predictions will be made.
- This is essential in many regulatory contexts (medical diagnosis, financial transactions, anything where anti-discrimination is required…)
- Hence, a more interpretable model is often **preferred** over another model, even if the latter is more “powerful”.

Source: https://christophm.github.io/interpretable-ml-book/interpretability.html

Model's Interpretability Techniques (1/2)

- **Feature summary statistics**: Many interpretation methods provide summary statistics for each feature. Some methods return a single number per feature, such as feature importance.
- **Feature summary visualization**: Visualizing feature summary statistics can be faster for spotting "interesting" behaviors.

Source: https://christophm.github.io/interpretable-ml-book/interpretability.html

Model's Interpretability Techniques (2/2)

- **Model internals (e.g. learned weights)**: Understanding the impact of internals is required for explainable models. Examples are the weights in linear models or the learned tree structure.
- **Data point**: This category includes all methods that return data points (already existent or newly created) to make a model interpretable. One method is called counterfactual explanations. To explain the prediction of a data instance, the method finds a similar data point by changing some of the features for which the predicted outcome changes in a relevant way.

Source: https://christophm.github.io/interpretable-ml-book/interpretability.html

Beyond Interpretability (1/2)

- **Fairness:** Ensuring that predictions are unbiased and do not implicitly or explicitly discriminate against protected groups. An explainable model can tell you why it has decided to reject a certain person's loan application. It becomes easier for a human to judge whether the decision is based on a learned demographic bias (e.g. racial).
- **Privacy:** Ensuring that sensitive information in the data is protected.

Source: https://christophm.github.io/interpretable-ml-book/interpretability-importance.html

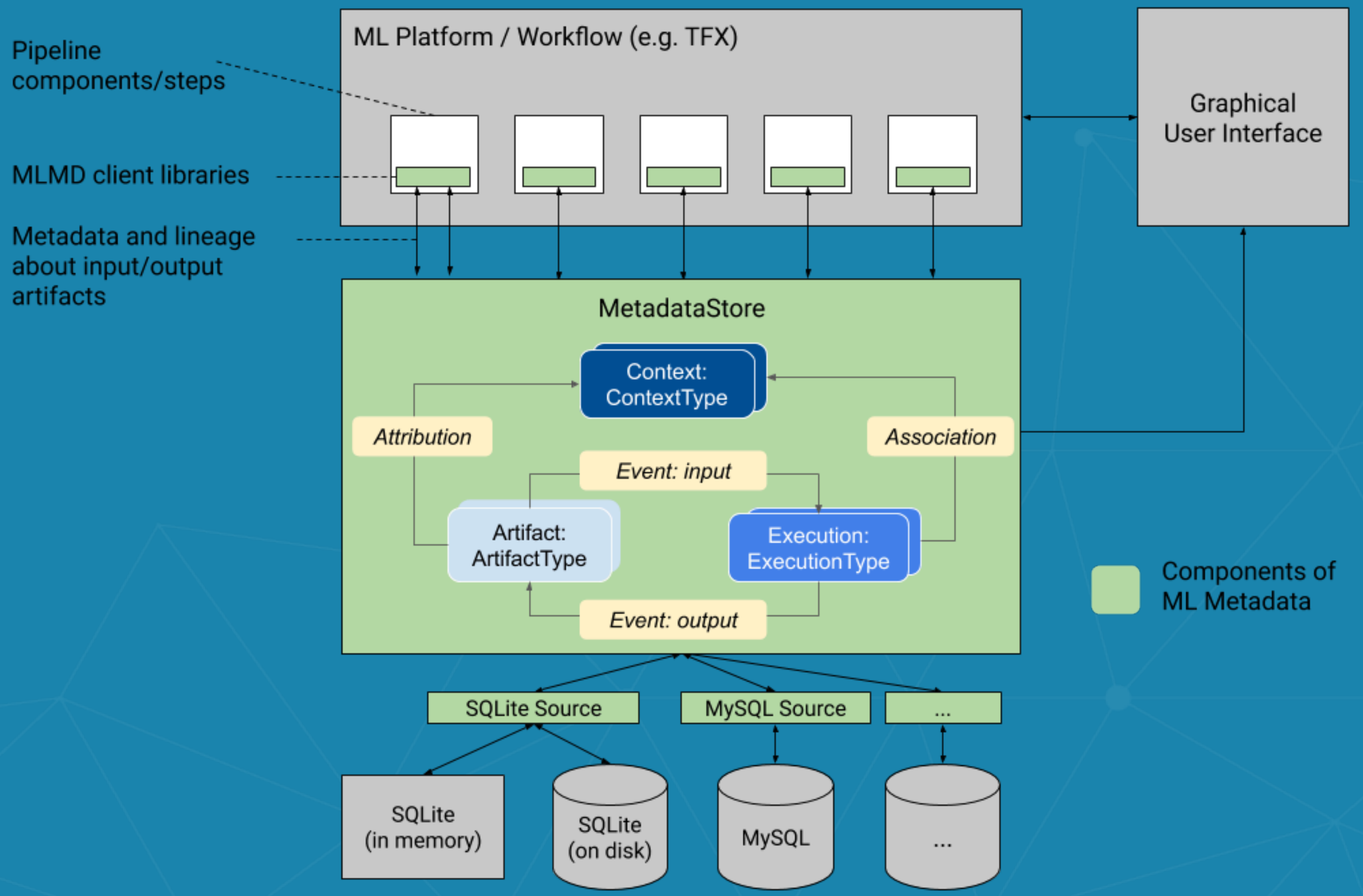
Beyond Interpretability (2/2)

- **Reliability or Robustness:** Ensuring that small changes in the input do not lead to large changes in the prediction.
- **Causality:** Check that only causal relationships are picked up.
- **Trust:** It is easier for humans to trust a system that explains its decisions compared to a black box.

https://github.com/google/ml-metadata/blob/master/g3doc/get_started.md

Source: https://christophm.github.io/interpretable-ml-book/interpretability-importance.html

Google ML metadata



Source: https://github.com/google/ml-metadata/blob/master/g3doc/get_started.md

MLMD functionality (1/2)

- **List** all Artifacts of a specific type. Example: all Models that have been trained.
- **Load** two Artifacts of the same type for comparison**.** Example: compare results from two experiments.
- **Show a DAG** of all related executions and their input and output artifacts of a context**.** Example: visualize the workflow of an experiment for debugging and discovery.
- **Recurse back** through all events to see how an artifact was created. Examples: see what data went into a model; enforce data retention plans.

Source: https://github.com/google/ml-metadata/blob/master/g3doc/get_started.md

MLMD functionality (2/2)

- **Identify all artifacts** that were created using a given artifact. Examples: see all Models trained from a specific dataset; mark models based upon bad data.
- **Determine** if an execution has been run on the same inputs before. Example: determine whether a component/step has already completed the same work and the previous output can just be reused.
- **Record and query** context of workflow runs. Examples: track the owner and changelist used for a workflow run; group the lineage by experiments; manage artifacts by projects.

Source: https://github.com/google/ml-metadata/blob/master/g3doc/get_started.md

ML Metadata

1) Register ArtifactTypes

2) Register ExecutionTypes

3) Create DataSet Artifact

4) Create Execution for Trainer

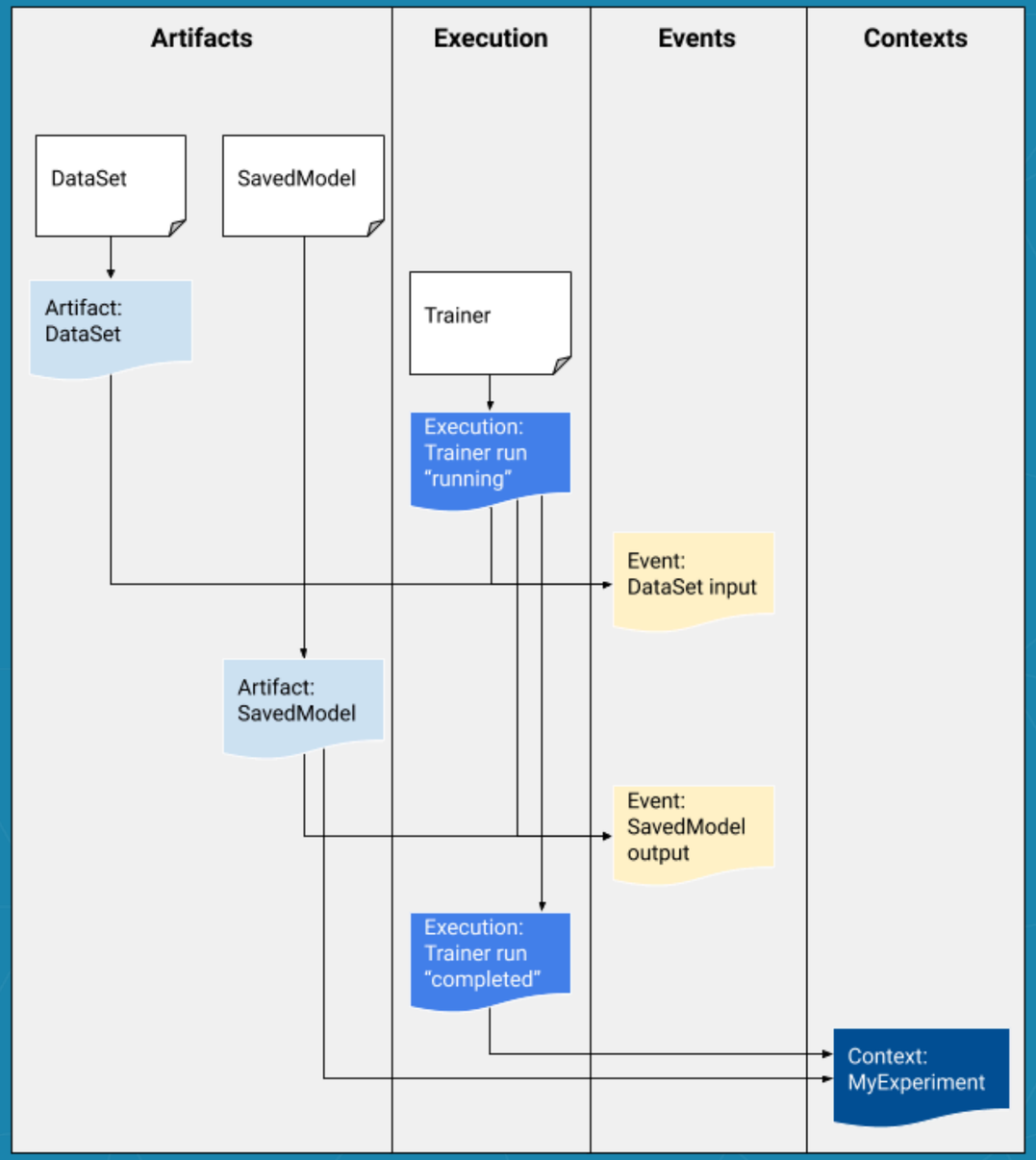
5) Read DataSet and record input event

6) Train Model and Create SavedModel Artifact

7) Write SavedModel and record output event

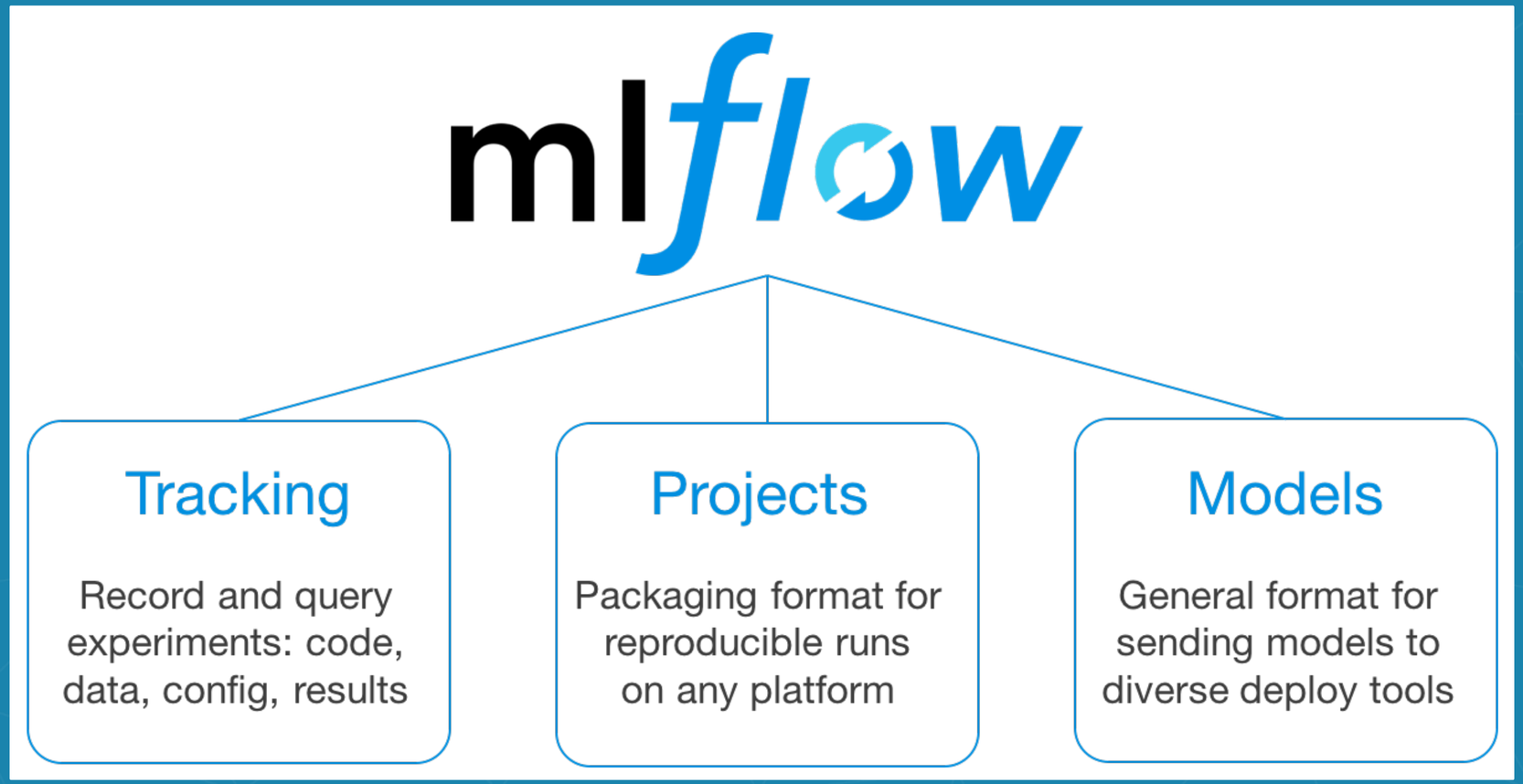
8) Mark Execution completed

9) Annotate the experiment with a Context



Source: https://github.com/google/ml-metadata/blob/master/g3doc/get_started.md

MLflow


mlflow
Tracking
Record and query experiments: code, data, config, results
Projects
Packaging format for reproducible runs on any platform
Models
General format for sending models to diverse deploy tools

What is MLflow?

- **MLflow Tracking** - An API and UI for logging parameters, code versions, metrics and output files when running your machine learning code to later visualize them. You invoke this API to track parameters, metrics, and artifacts.
- **MLflow Projects** - A standard format for packaging reusable data science code. Each project is a directory with code or a Git repository, and uses a descriptor file to specify its dependencies and how to run the code. A MLflow Project is defined by a YAML file.
- **MLflow Models** - A convention for packaging machine learning models in multiple formats called “flavors”. MLflow offers tools to help you deploy different model flavors. Each MLflow model is saved as a directory containing arbitrary files and a descriptor file that lists the flavors it can be used in.

MLflow Tracking (1/2)

- Organized around the concept of *runs*, which are executions of some data science code.
- Each run records the following information:
  - **Code Version.** Git commit hash used for the run, if it was run from an MLflow Project.
  - **Start & End Time.** Start and end time of the run.
  - **Source.** Name of the file to launch the run, or the project name and entry point for the run if run from an MLflow Project
  - …

MLflow Tracking (2/2)

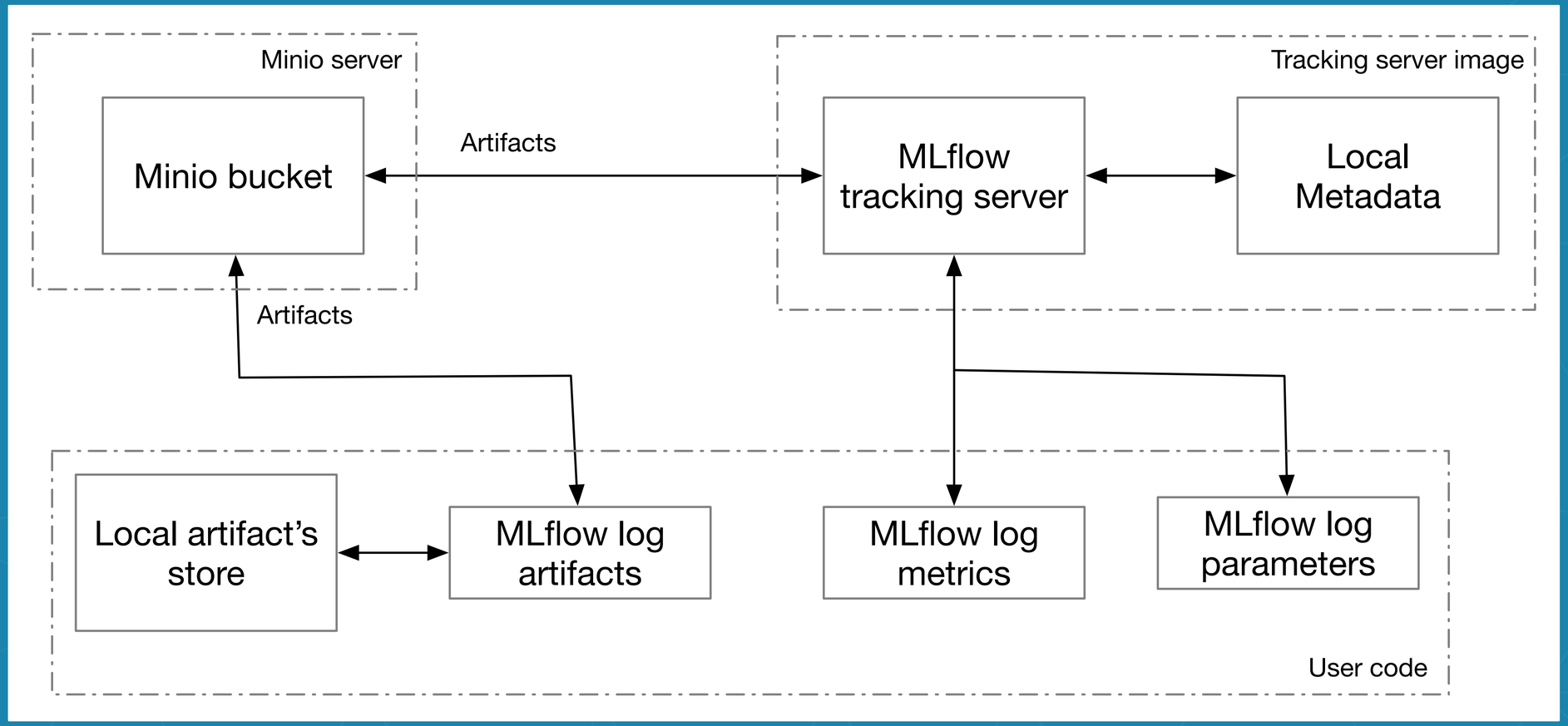
- Each run records the following information:
- …
- **Parameters.** Key-value input parameters of your choice. Both keys and values are strings.
- **Metrics.** Key-value metrics, where the value is numeric. Each metric can be updated throughout the course of the run (for example, to track how your model's loss function is converging), and MLflow records and lets you visualize the metric's full history.
- **Artifacts.** Output files in any format. For example, you can record images (for example, PNGs), models (for example, a pickled scikit-learn model), and data files (for example, a Parquet file) as artifacts.

Exercise 2:
Using MLflow to Capture and View Model Training Metadata

Exercise - Capturing Metadata Using MLflow Tracking

- See the project README for installation instructions.
  - For example, Python3 and Pip are required.
  - Install MLflow using pip: `pip3 install mlflow`
  - Load and run python code to produce tracking results.
  - Start MLflow UI - mlflow server --host 0.0.0.0 —port 5000
  - Go to http://localhost:5000 to look at a the tracking UI

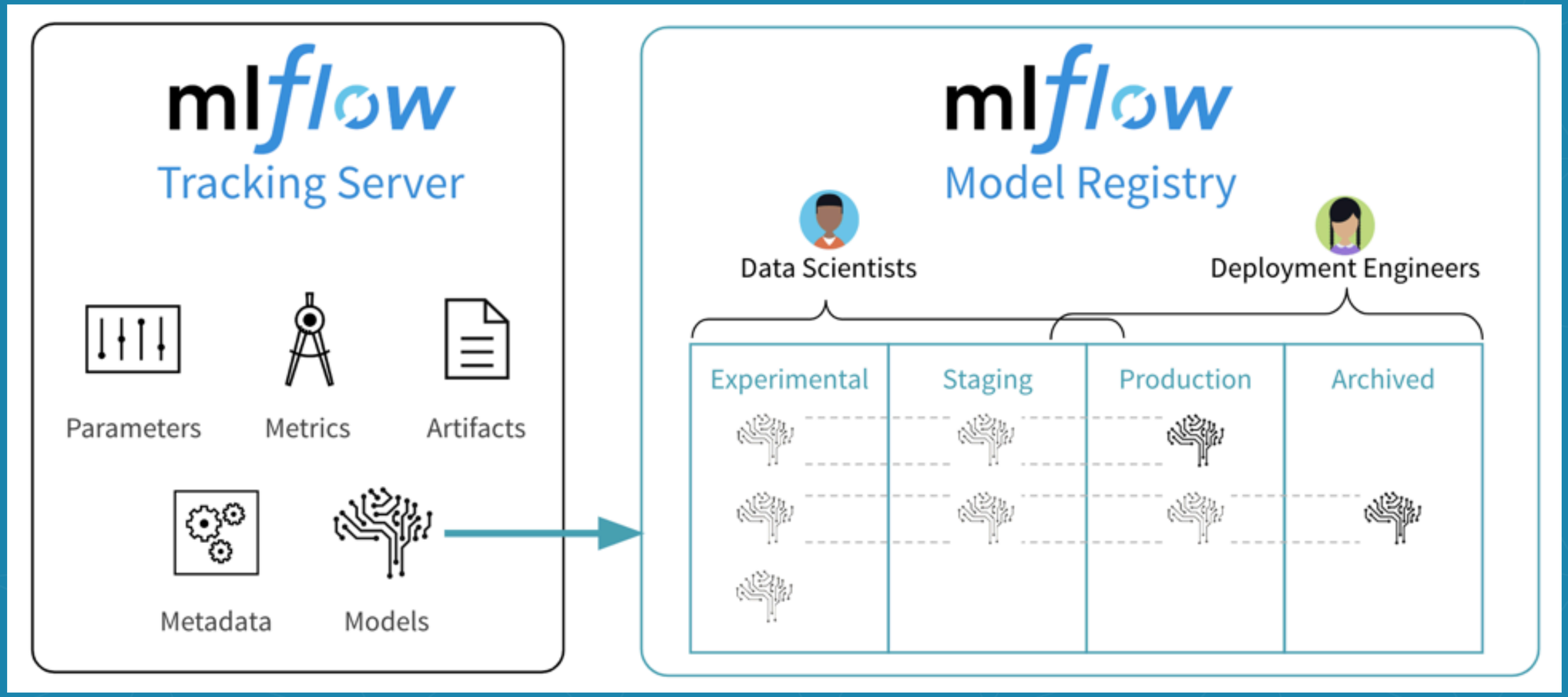
MLflow Tracking - Clustered Deployment



Model Metadata Stores

- Custom systems for model management are emerging:
  - Hortonworks Registry contains model registry plugin, that was never complete, but can be used as a starting point.
  - MLflow Model Registry introduced by DataBricks in 2019

MLflow Model Registry

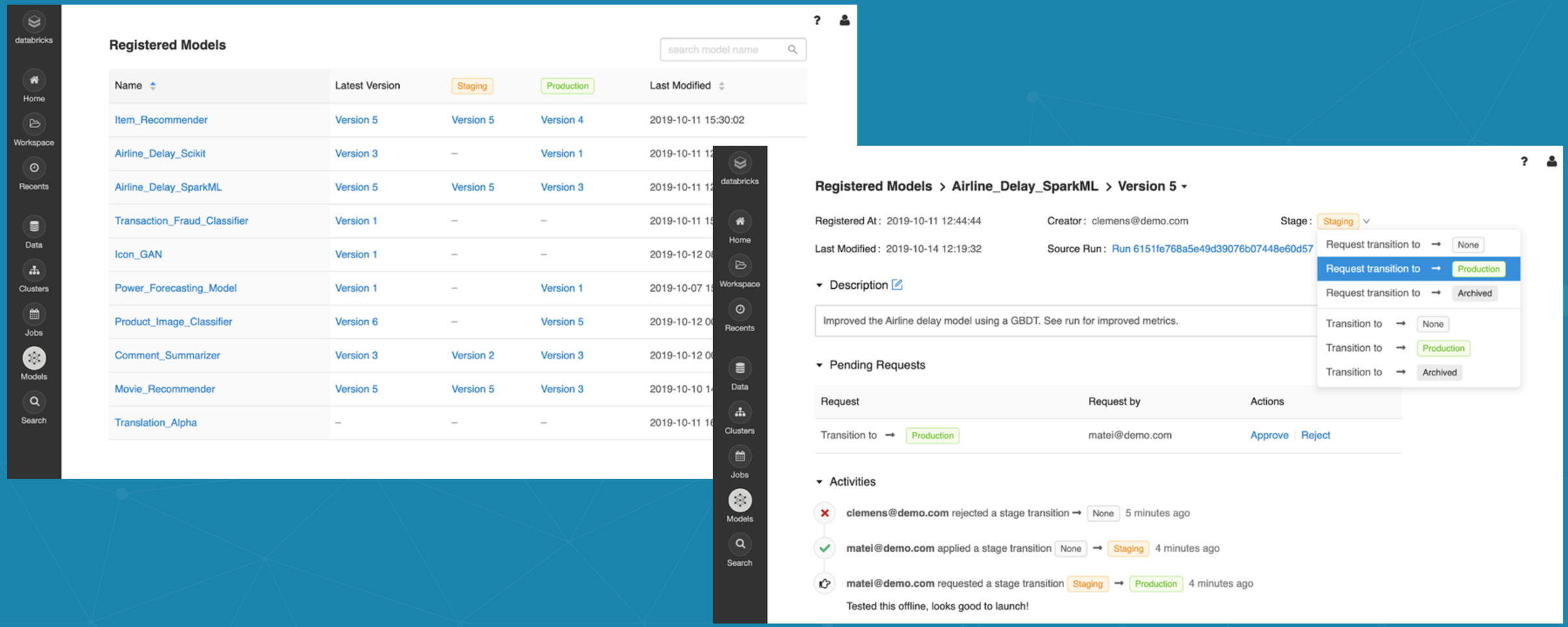


Source: https://databricks.com/blog/2019/10/17/introducing-the-mlflow-model-registry.html

MLflow model Registry

- Tightly connected to the MLflow tracking server. As a result it is linked to data dependencies, and inherent non-deterministic characteristics of statistical modeling.
- Provides nice collaboration capabilities.
- Enables different workflow processes, enabling handoffs between experimentation, testing, and production deployments.

MLflow Model Registry UI



Source: https://databricks.com/blog/2019/10/17/introducing-the-mlflow-model-registry.html

Exercise 3: Creating a Model Registry Using Apache Atlas

Building model registry leveraging Atlas

- The role of model registry is to capture information about deployed model servers describing their deployment, input and output messages, etc.
- Important Atlas capabilities that are relevant to registry creation include the abilities, to create new types, lineage support, etc.
- There already are precedents for using Atlas as registry, for example Avro schema registry and Model Registry

Atlas base types

- **Referenceable**: This type represents all entities that can be searched for using a unique attribute called qualifiedName.
- **Asset**: This type extends Referenceable and adds attributes like name, description and owner.
- **Infrastructure**: This type extends Asset and typically can be used to be a common super type for infrastructural metadata objects like clusters, hosts etc.
- **DataSet**: This type extends Referenceable. Conceptually, it can be used to represent an type that stores data.
- **Process**: This type extends Asset. Conceptually, it can be used to represent any data transformation operation. Process type has two specific attributes, inputs and outputs of type arrays of DataSet entities. Thus an instance of a Process type can use these inputs and outputs to capture how the lineage of a DataSet evolves.

Source: https://atlas.apache.org/#/TypeSystem

Exercise - Creating a Model Registry Using Apache Atlas

- See the project README for the Atlas instructions.
  - While they explain how to build and run Atlas, use the Docker implementation!

Questions?

Boris Lublinsky, boris@lightbend.com
Dean Wampler, anyscale.io, @deanwampler