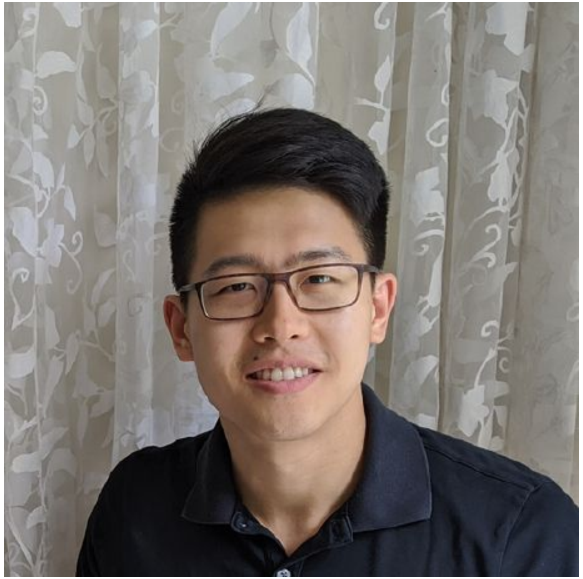
Building high performance recommender systems with feature stores

Data Council 2022



Danny Chiao

Software Engineer
Engineering Lead
Tecton / Feast (former lead at Google)

Agenda

- Background
  - Recommender systems intro
  - What is a feature store?
- Recommender systems challenges
- Optimizing performance
- Correctness in operational recommender systems
- Feast x RecSys

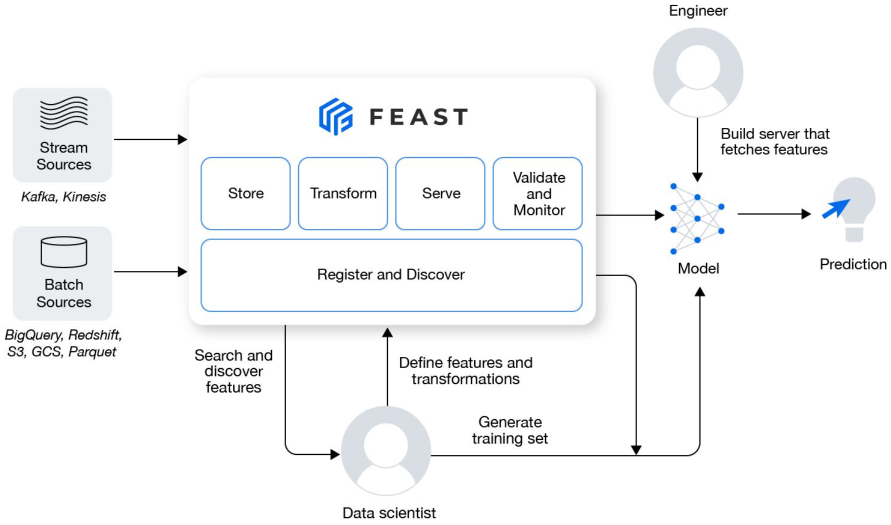
Background

Recommender systems

- **Use cases:** e-commerce, media streaming, social, ride-hailing, biomedical, etc
- **Who:** data scientists, data engineers, platform engineers
- **Trend:** Batch predictions → online predictions
- In practice, consists usually of two steps: candidate generation + reranking
- Long tail of operational challenges

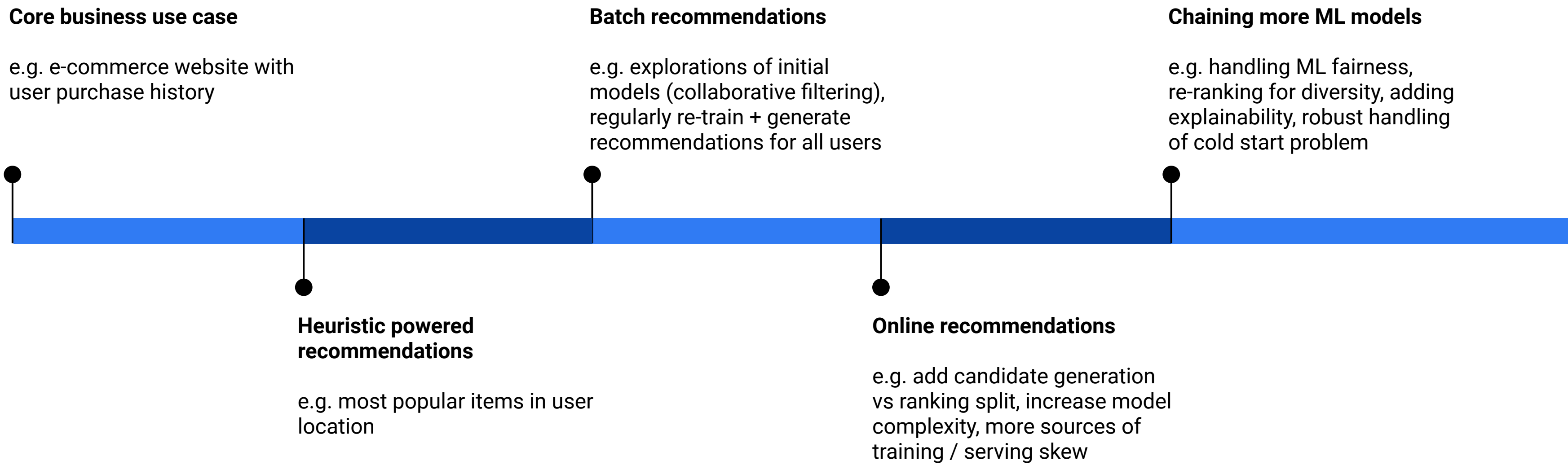
What is a feature store?

- Manages ingestion and storage of streaming and batch data
- Allows for standardized definitions of features and transformations
- Generates point-in-time correct features
- Ensures model performance by tracking, validating, and monitoring features



Recommender system challenges

A typical journey of building a recommender system



Core business use case

e.g. e-commerce website with user purchase history

Heuristic powered recommendations

e.g. most popular items in user location

Batch recommendations

e.g. explorations of initial models (collaborative filtering), regularly re-train + generate recommendations for all users

Online recommendations

e.g. add candidate generation vs ranking split, increase model complexity, more sources of training / serving skew

Chaining more ML models

e.g. handling ML fairness, re-ranking for diversity, adding explainability, robust handling of cold start problem

Challenges with recommender systems

Type of challenge	Examples
Operational	● Low latency batch retrieval of features ● Feature freshness
Feature engineering	● Access to request data at training time ● Supporting time-travel in model training
Data quality	● Mitigating training / serving skew or data drift ● Bad data pushes from stream sources
Organizational	● Data scientist vs engineers ● Multiple business objectives to optimize ● A/B tests to measure business metrics lift
Miscellaneous	● Cold start ● Privacy / GDPR

Operational challenges

Among other requirements, an online recommender system often:

- Needs fresh features (write heavy)
  - **Why?** e.g. user session activity
  - Different events update different features
- Needs low latency access to features for many entities (read heavy)
  - **Why?** e.g. for a given user, need to rank 100s to 1000s of items
    - Typically, the faster the recommendation, the more likely users accept them.
    - The less time spent on data, the more time the model can spend inferring.

Optimizing for the above can introduce significant data quality issues too.

Low-latency access to fresh features

Achieving lower latency

Generally, there is a need to have features available at low latency in serving via an online store. There are many challenges in building such a store though:

1. Balancing requirements (read vs write, cost, etc)
2. Complex + slow type conversions across different sources
3. Optimizing for batch retrieval

Challenges of building a low latency online store

Consideration

1. Balancing requirements
   a. **update features independently (e.g. from streams)**
   b. **reading features for a specific model quickly**
   c. enable feature re-use across models
   d. cost management

Example strategies

➜ Store features from an event together in both online store & offline store

➜ Store features for an entity for a specific model together (pruning unused columns)

User Features
user_id
country
age
last_viewed_item_category
ts_country
ts_age
ts_last_5_viewed_item_category

User Metadata Features
user_id
country
age
timestamp

User Session Features
user_id
last_viewed_item_category
last_transaction_amt
timestamp

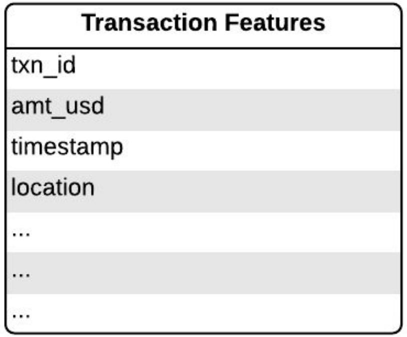
User Historical Features
user_id
28d_avg_transaction_amt
28d_top_item_category
timestamp

Embedding features
user_id
user_embedding
timestamp

Challenges of building a low latency online store

Consideration

1. Balancing requirements
   a. update features independently (e.g. from streams)
   b. reading features for a specific model quickly
   c. **enable feature re-use across models** ➜ Feature versioning
   d. **cost management** ➜ TTL entities (warning: multiple models)



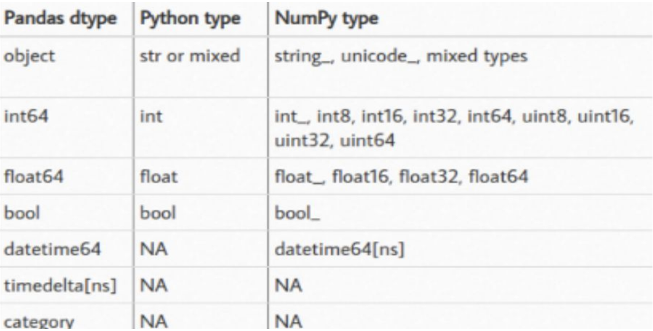
Transaction Features
txn_id
amt_usd
timestamp
location
...
...
...

Challenges of building a low latency online store

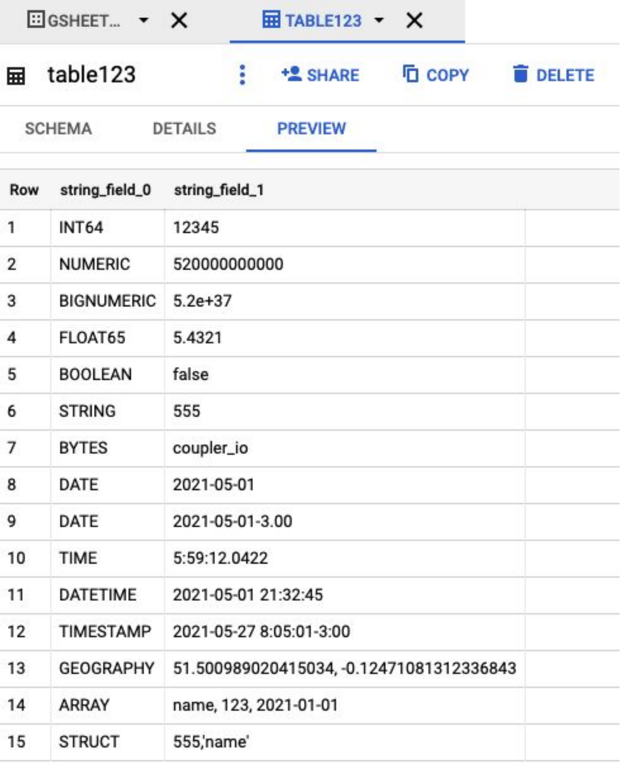
Consideration

2. Managing type conversions for online store
   a. Data source types and Pandas / Python types (in data scientist notebook)

APACHE ARROW



Pandas dtype	Python type	NumPy type
object	str or mixed	string_, unicode_, mixed types
int64	int	int_, int8, int16, int32, int64, uint8, uint16, uint32, uint64
float64	float	float_, float16, float32, float64
bool	bool	bool_
datetime64	NA	datetime64[ns]
timedelta[ns]	NA	NA
category	NA	NA



Row	string_field_0	string_field_1
1	INT64	12345
2	NUMERIC	520000000000
3	BIGNUMERIC	5.2e+37
4	FLOAT65	5.4321
5	BOOLEAN	false
6	STRING	555
7	BYTES	coupler_io
8	DATE	2021-05-01
9	DATE	2021-05-01-3.00
10	TIME	5:59:12.0422
11	DATETIME	2021-05-01 21:32:45
12	TIMESTAMP	2021-05-27 8:05:01-3:00
13	GEOGRAPHY	51.500989020415034, -0.12471081312336843
14	ARRAY	name, 123, 2021-01-01
15	STRUCT	555,'name'

Challenges of building a low latency online store

Consideration

3. Optimizing for batch retrieval
   a. Multiple types of entities in same request (e.g. user ids + item ids)
   b. Large batch sizes (i.e. number of entities to score in the sample request)
   c. Online store specific optimizations.

Example strategies

➜ De-duplication of requested entities

➜ **Co-locating entities**

➜ **Caching**

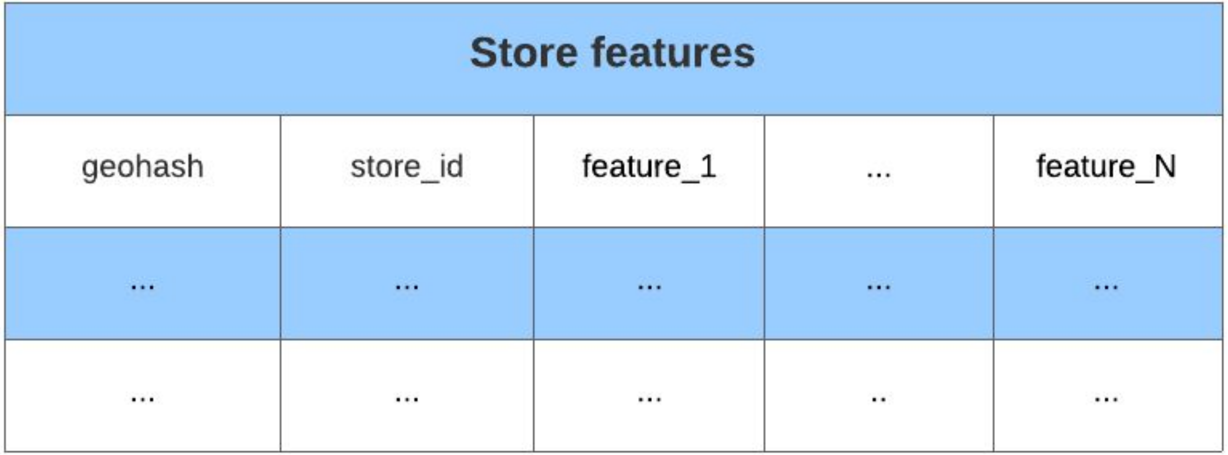
➜ **E.g. Redis pipelines & `mget` vs `hmget` vs `hgetall`**

➜ E.g. Different ways of bulk loading data into online store

Co-locating entities

Example: fetch features for all stores in a region

Also: Redis hmget vs hgetall

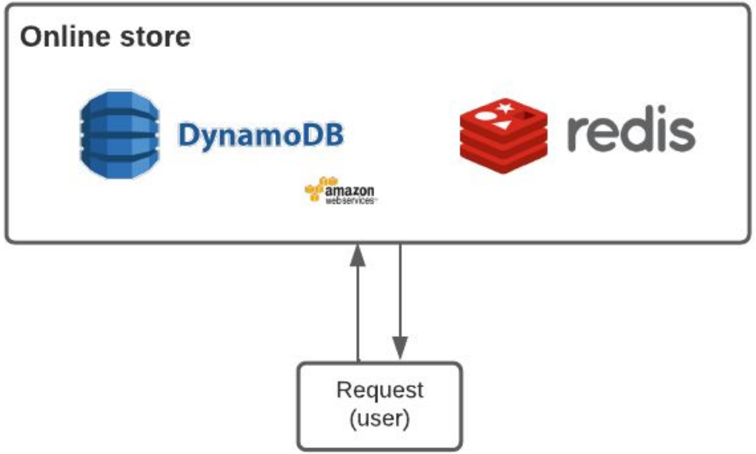


Store features

geohash	store_id	feature_1	...	feature_N
...	...	...	...	...
...	...	...	..	...

Caching

Caching (e.g. popular entities hit Redis)



Correctness

Handling bad data

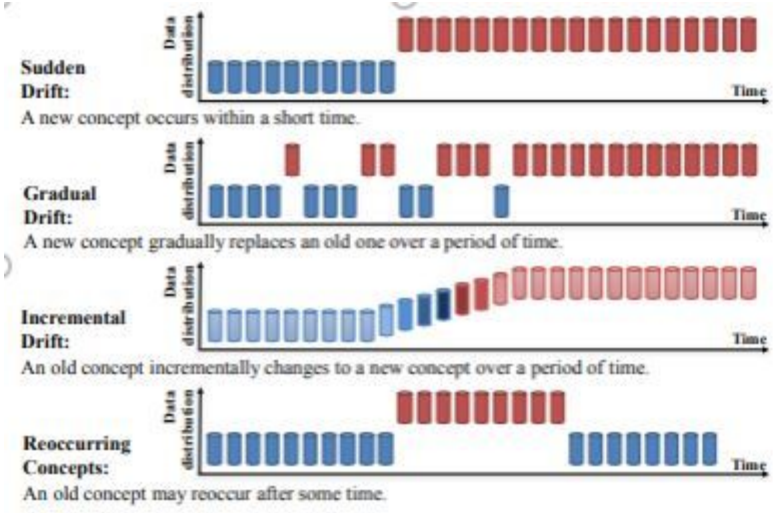
Many sources of bad data:

- E.g. upstream systems change, resulting in schema or feature distribution shifts (e.g. engineers change normalization logic)
- E.g. faulty feature transformation logic or messy data that has not been properly cleaned
- E.g. streams can publish bad data (or fail to publish data), leading to poor quality or missing data.

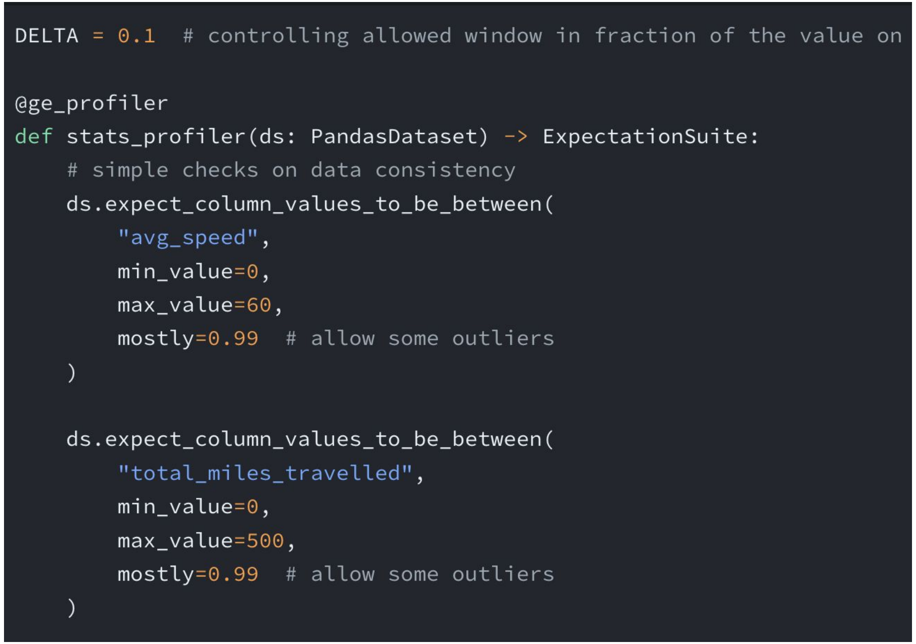
Feature stores may:

- Implement data quality monitoring
  - e.g. see Feast DQM and versioned datasets via `SavedDatasets`
  - e.g. Great Expectations integration
  - can easily go wrong with false alerts
- Distinguish between missing data & empty feature values in response.
- Fallback to old / default values or impute values for missing / faulty data.

Why using Great Expectations isn’t enough


Sudden Drift:
Data distribution
Time
A new concept occurs within a short time.
Gradual Drift:
Data distribution
Time
A new concept gradually replaces an old one over a period of time.
Incremental Drift:
Data distribution
Time
An old concept incrementally changes to a new concept over a period of time.
Reoccurring Concepts:
Data distribution
Time
An old concept may reoccur after some time.

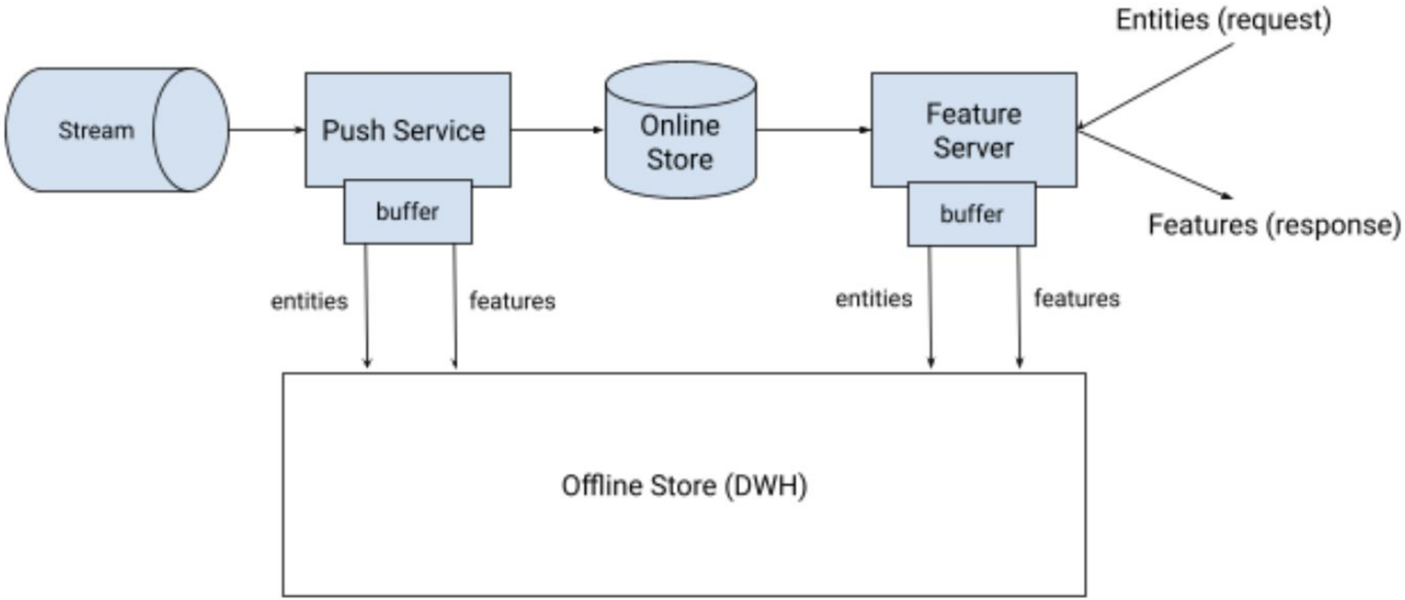
Source: https://arxiv.org/pdf/2004.05785.pdf



```
DELTA = 0.1  # controlling allowed window in fraction of the value on

@ge_profiler
def stats_profiler(ds: PandasDataset) -> ExpectationSuite:
    # simple checks on data consistency
    ds.expect_column_values_to_be_between(
        "avg_speed",
        min_value=0,
        max_value=60,
        mostly=0.99  # allow some outliers
    )

    ds.expect_column_values_to_be_between(
        "total_miles_travelled",
        min_value=0,
        max_value=500,
        mostly=0.99  # allow some outliers
    )
```

Source: [Feast 0.19 data quality monitoring tutorial]() (first milestone in RFC)

Handling bad data



Why a transformation library isn’t enough

- Data scientist vs engineers
  - Data scientists do their SQL transformations in DWH and Python transformations in notebooks
  - Engineers work in a different environment (e.g. Java / Go servers)
- What about Spark / Flink / Beam?
  - Some transformations need access to request data or need to execute at inference time
- Optimizing for fast model training iterations != optimizing for fast serving
  - E.g. Pandas is much slower with small number of rows (e.g. at serving time)
- Assumption that data is neatly organized by timestamp + available in the same fashion
  - E.g. request data that’s available in memory isn’t regularly snapshotted at inference time

Data pipeline delays

Consider a naive approach:

- At training time, generate features from offline store (data warehouse) using event timestamps
- At serving time, use an online store to serve features that loads data from batch periodically

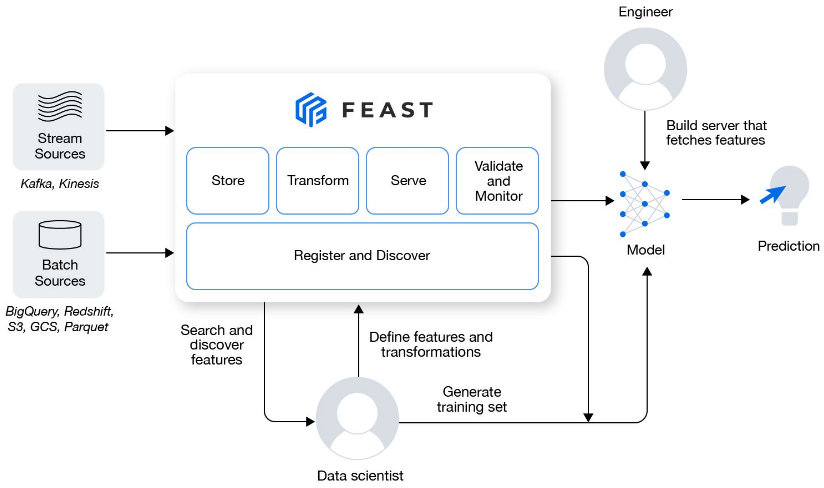
BUT: what if there's a delay in data pipelines to populate offline / online stores?

- A watermark + materialization (e.g. processing time) timestamps matters
  - e.g. incremental processing of offline data
    - e.g. reject items later than X (materialize from last end time - X until now)
  - e.g. monitor delay (e.g. most recent event_timestamp in data source or data source `created_timestamp` vs `event_timestamp`.)
- Stream based ingestion into online store

Feast x RecSys

Feast

- Feast is an open-source feature store that connects to GCP, AWS, Azure, Snowflake, Hive, Redis, Spark, etc
- Active community with 3k+ members on Slack and bi-weekly community calls
- Goal: to simplify & reduce overhead of managing features / data in ML systems



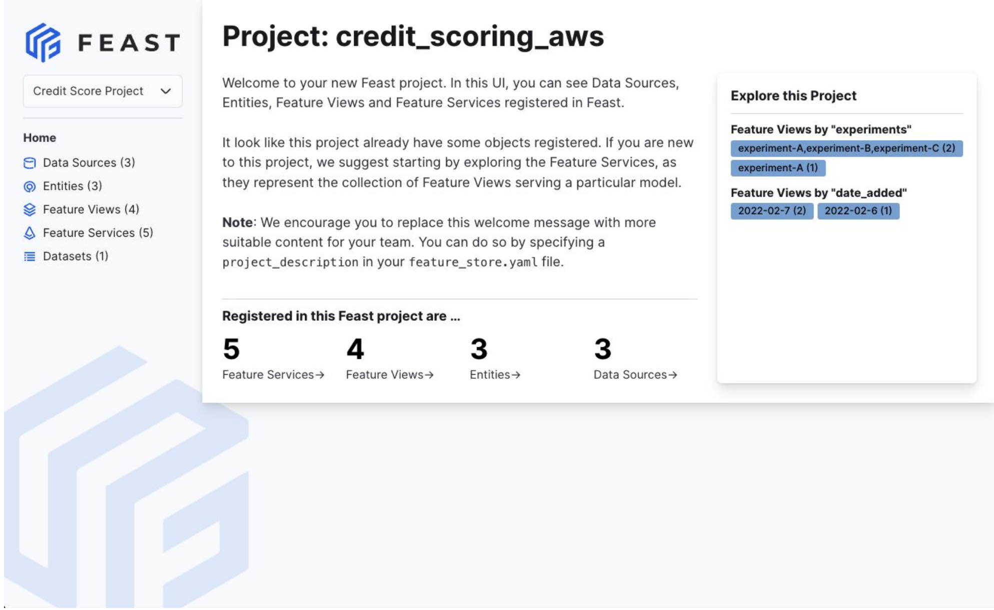
Teams running or contributing to Feast

tecton · gojek · shopify · salesforce · IBM · Google Cloud · adyen · Postmates

Robinhood · Porch · FARFETCH · zulily · agoda · SUMATRA · redislabs HOME OF REDIS

Feast

- Building a framework for managing:
  - Multiple data sources
  - Multiple types of entities
  - Sources of training / serving skew
- Enforcing best practices on storage within offline / online stores & batch / stream sources
- Encouraging best practices for reducing error across teams (e.g. feature reuse, `feast plan`)
- Reducing effort to author features (e.g. abstracting away point-in-time joins)
- Pluggable (e.g. custom offline / online stores)



Project: credit_scoring_aws

Welcome to your new Feast project. In this UI, you can see Data Sources, Entities, Feature Views and Feature Services registered in Feast.

It look like this project already have some objects registered. If you are new to this project, we suggest starting by exploring the Feature Services, as they represent the collection of Feature Views serving a particular model.

Note: We encourage you to replace this welcome message with more suitable content for your team. You can do so by specifying a `project_description` in your `feature_store.yaml` file.

Registered in this Feast project are ...

5 Feature Services→
4 Feature Views→
3 Entities→
3 Data Sources→

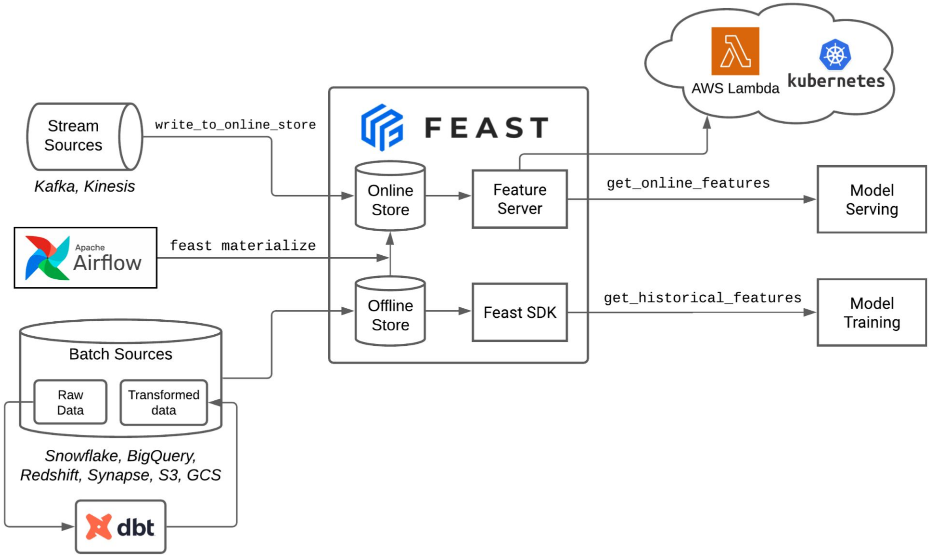
Explore this Project

Feature Views by "experiments"
experiment-A,experiment-B,experiment-C (2)
experiment-A (1)

Feature Views by "date_added"
2022-02-7 (2)
2022-02-6 (1)

Deploying feature stores

- Airflow for scheduled materialization of online features
  - Stream processors push data to online store directly
- Deployment of a feature server
  - Serverless (e.g. Using Feast’s [Lambda integration]())
  - Kubernetes (e.g. [Feast Serving]())
- Versioning models with feature service

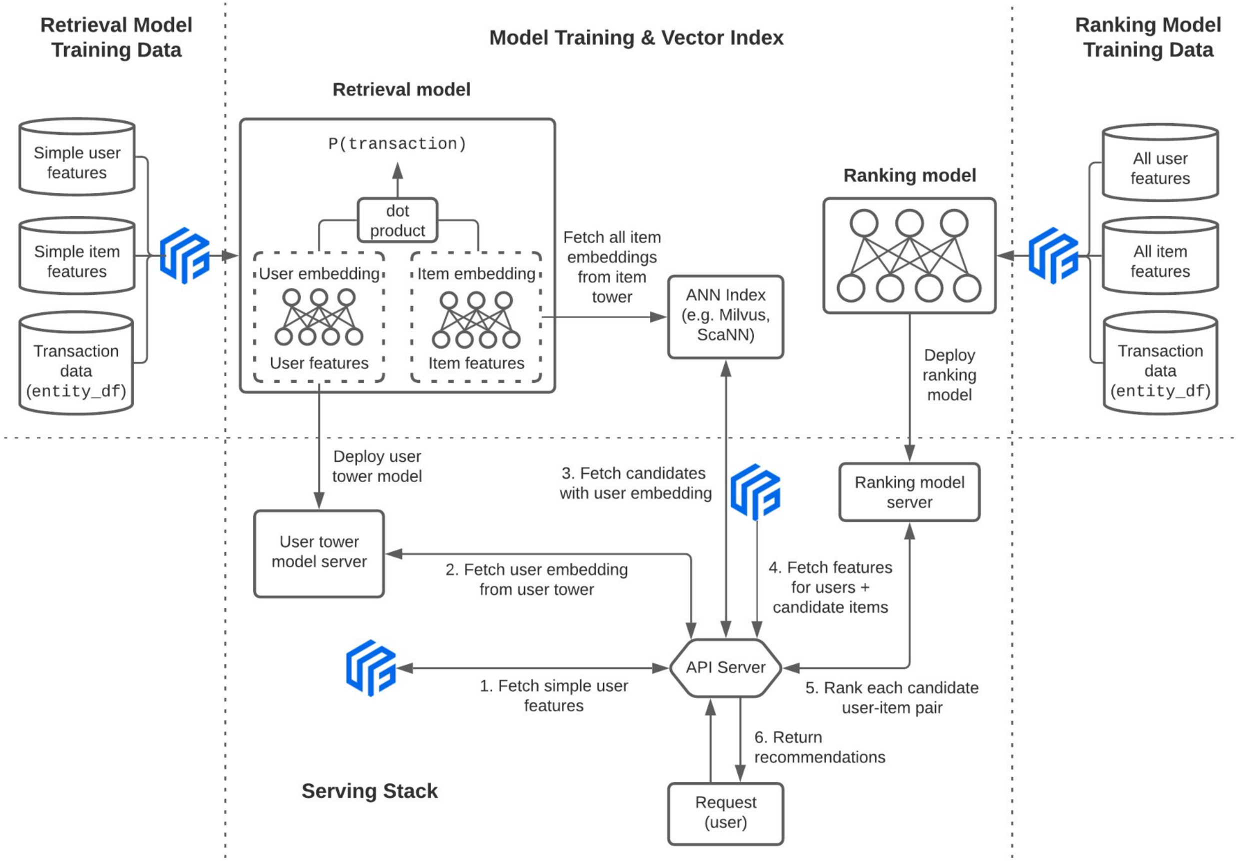


For batch recommender systems

Example: predict suggested items for users to purchase with matrix factorization

- V0: use only user ids and item ids with purchase patterns
  - User id + item id -> user embeddings + item ids -> dot product for purchase
  - Store embeddings in Feast + do lookups
- V1: Mitigate cold start problems
  - Use content-based model or other heuristic (e.g. bandit algorithms)
  - Use Feast to fetch other user features & item features as part of model to generate embeddings
- Key challenges solved by Feast
  - Collaboration + sharing of features / pipelines.
    - Future: Feast batch transformations
  - Lineage: knowing which models depend on which features
    - Future: Data drift (when to retrain model, or when other upstream signals are changing)

For online recommender systems


Retrieval Model Training Data
Simple user features
Simple item features
Transaction data (entity_df)
Model Training & Vector Index
Retrieval model
P(transaction)
dot product
User embedding
User features
Item embedding
Item features
Fetch all item embeddings from item tower
ANN Index (e.g. Milvus, ScaNN)
Ranking model
Deploy ranking model
Ranking Model Training Data
All user features
All item features
Transaction data (entity_df)
Deploy user tower model
User tower model server
3. Fetch candidates with user embedding
Ranking model server
2. Fetch user embedding from user tower
4. Fetch features for users + candidate items
1. Fetch simple user features
API Server
5. Rank each candidate user-item pair
6. Return recommendations
Serving Stack
Request (user)

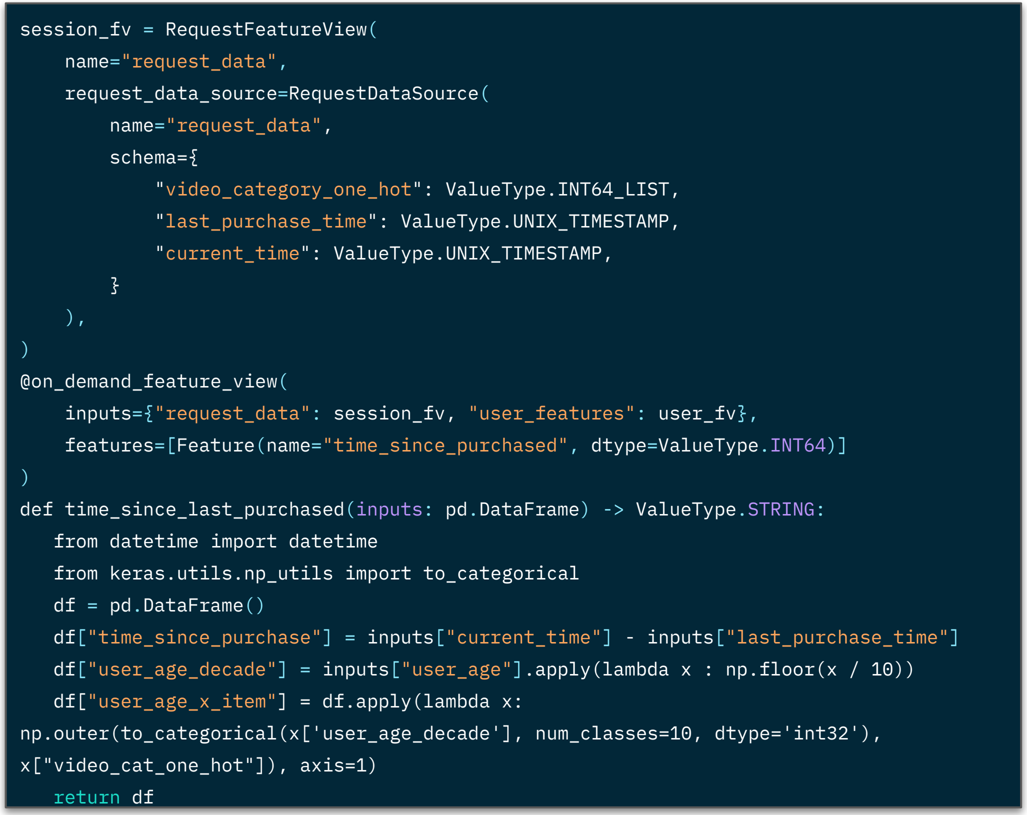
For online recommender systems

Example: predict videos to watch, predict next piece of clothing to buy given previous purchases, generate search results given a query

- Need access to data only available at request time (e.g. current time vs time of video, session data, etc)
  - Feast: `OnDemandFeatureView`
- Freshness of features can matter (e.g. data in session)
  - Feast: regular feature materialization & push based stream ingestion. Also: DQM for drift detection
- Low latency is important. Often, this means we have a candidate generation model + a ranking model.
  - Feast: simplifies fetching and monitoring features (+ versioning models with features)
- Environment difference between training environment (notebook) and serving environment (API server)
  - Feast: `OnDemandFeatureView` enables python logic from training to be re-used at serving
  - Feast: manages differences in type systems across data sources vs online stores vs feature schemas

Sample features

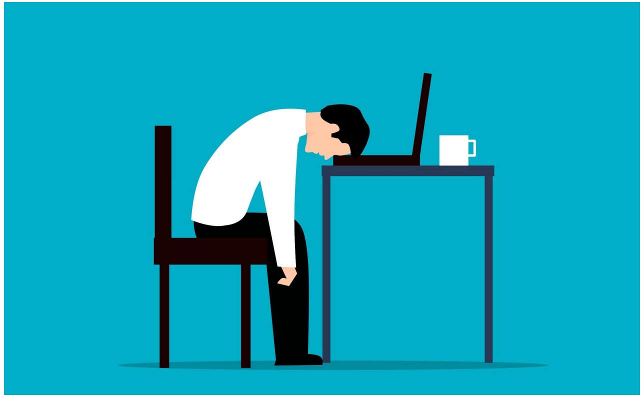
- Binning
- Feature crossing
  - (instead of e.g. using BQML's crossing functionality which can't be done at serving time)
- Time related features (e.g. time since last event, how recently a video was published)
- Batch features
  - User / item metadata
  - User / item history: `last_X_purchased_items`



```
session_fv = RequestFeatureView(
    name="request_data",
    request_data_source=RequestDataSource(
        name="request_data",
        schema={
            "video_category_one_hot": ValueType.INT64_LIST,
            "last_purchase_time": ValueType.UNIX_TIMESTAMP,
            "current_time": ValueType.UNIX_TIMESTAMP,
        }
    ),
)
@on_demand_feature_view(
    inputs={"request_data": session_fv, "user_features": user_fv},
    features=[Feature(name="time_since_purchased", dtype=ValueType.INT64)]
)
def time_since_last_purchased(inputs: pd.DataFrame) -> ValueType.STRING:
   from datetime import datetime
   from keras.utils.np_utils import to_categorical
   df = pd.DataFrame()
   df["time_since_purchase"] = inputs["current_time"] - inputs["last_purchase_time"]
   df["user_age_decade"] = inputs["user_age"].apply(lambda x : np.floor(x / 10))
   df["user_age_x_item"] = df.apply(lambda x:
np.outer(to_categorical(x['user_age_decade'], num_classes=10, dtype='int32'),
x["video_cat_one_hot"]), axis=1)
   return df
```

Takeaways

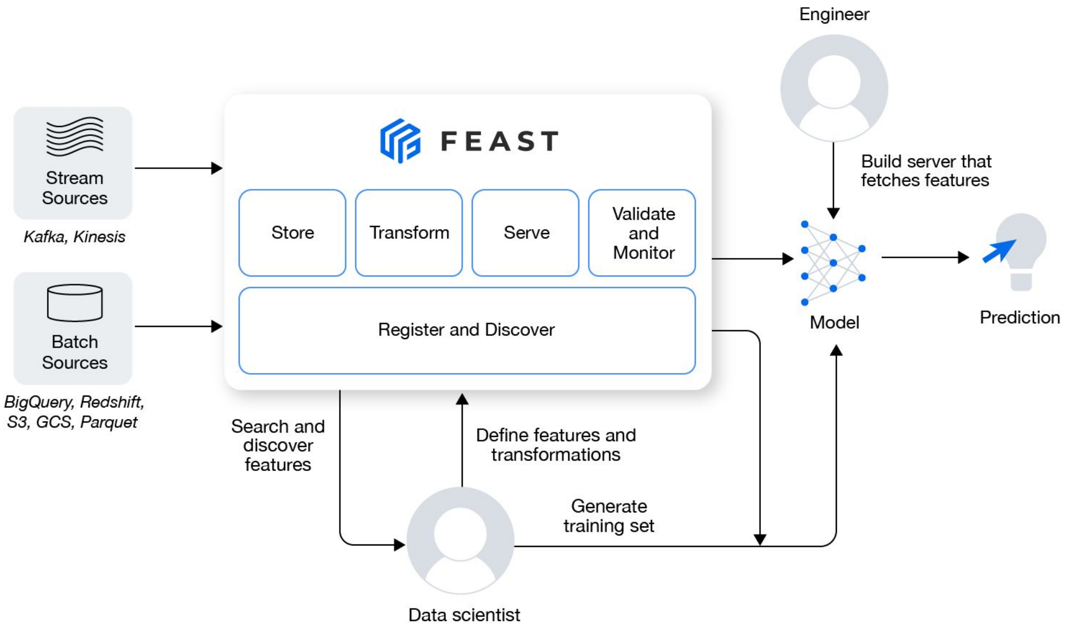
- Recommender systems can quickly balloon in complexity
    - E.g. low latency (read vs write), batch reads, correctness / bad data, type conversions
- Feature stores can enforce best practices / abstract some of this complexity away for both batch + online recommender systems
    - E.g. difference between offline / online store
    - E.g. lineage via feature repository + Web UI
- Consistent + performant on demand transformations are key to online recommender systems



Questions?

Useful resources

- https://feast.dev/
- https://github.com/feast-dev/feast
- https://slack.feast.dev/



Scaling AI/ML Workloads with Ray Ecosystem

Jules S. Damji, @2twitme

Lead Developer Advocate, Ray Team @ Anyscale

Data Council, Austin, TX March 23, 2022

anyscale

Overview

- **Why & What Ray & Ray Ecosystem**
- **Ray Architecture & Components**
- **Ray Core APIs**
- **Ray Native ML Libraries**
  - Ray Tune, XGBoost-Ray
- **Demo**
  - Scaling ML workloads
- **Q & A**

Why Ray?

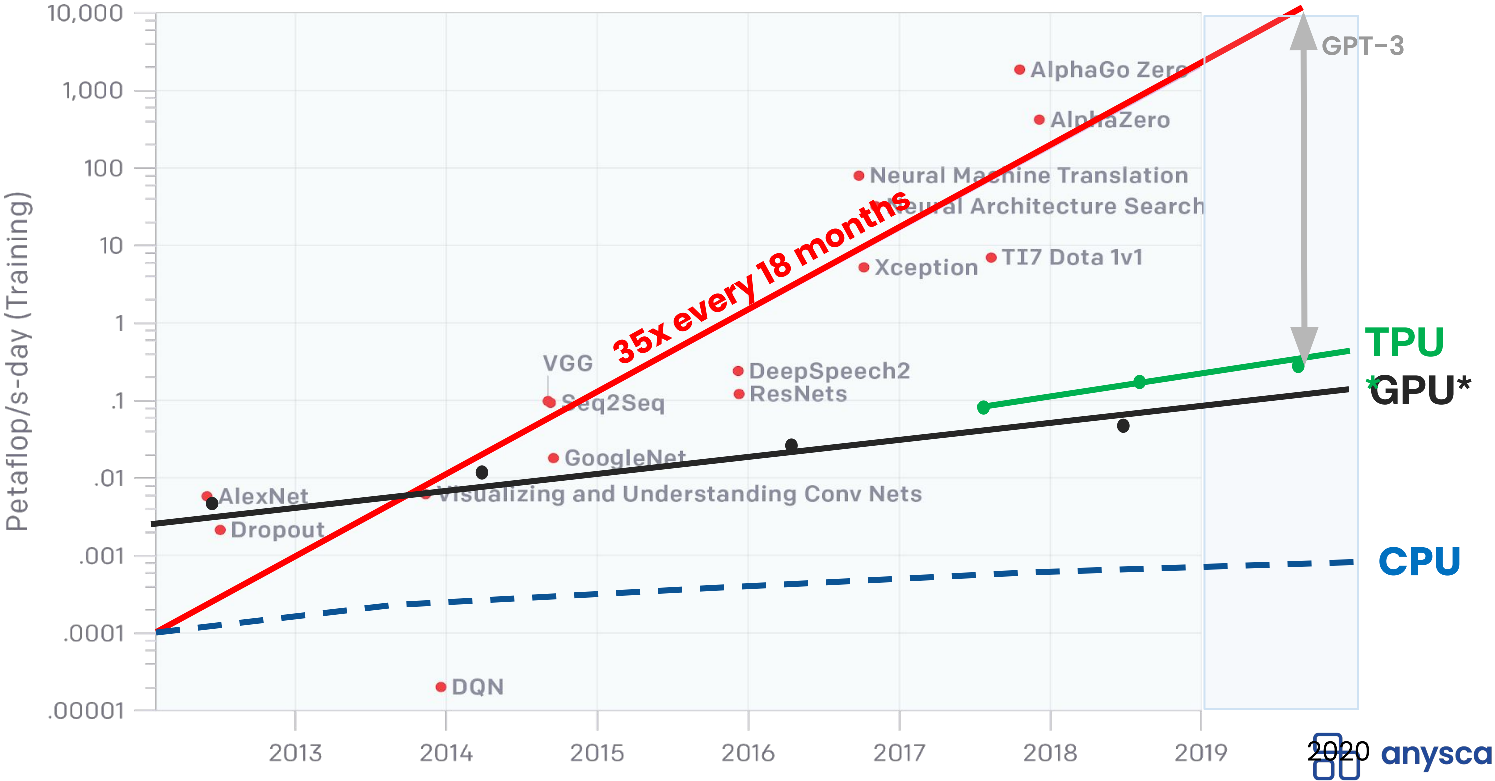
- **Machine learning is pervasive in every domain**
- Distributed machine learning is becoming a necessity
- Distributed systems is notoriously hard

Why Ray?

- Machine learning is pervasive in every domain
- **Distributed machine learning is becoming a necessity**
- Distributed systems is notoriously hard

anyscale

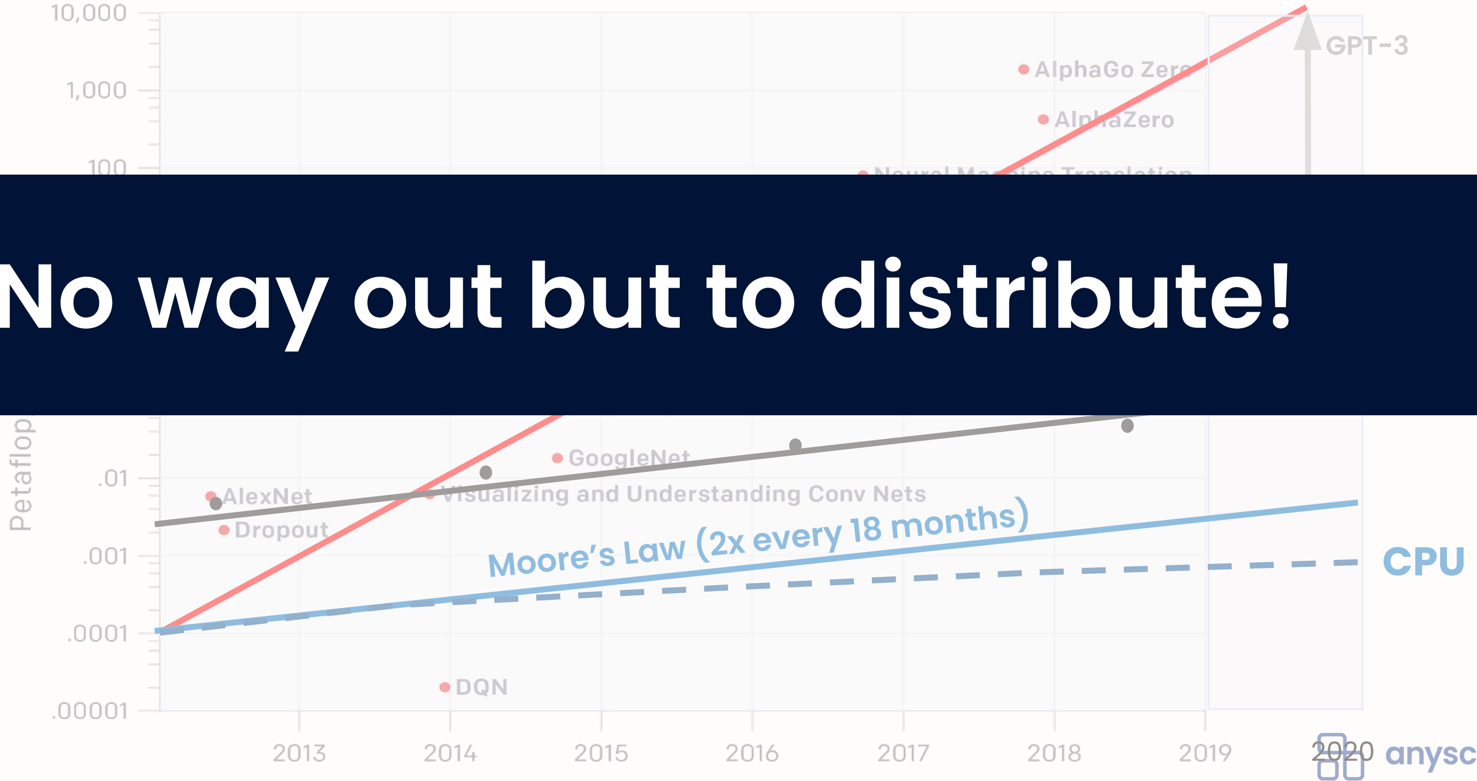
Specialized hardware is also not enough



https://openai.com/blog/ai-and-compute/

RAY anyscale

Specialized hardware is also not enough



No way out but to distribute!

https://openai.com/blog/ai-and-compute/

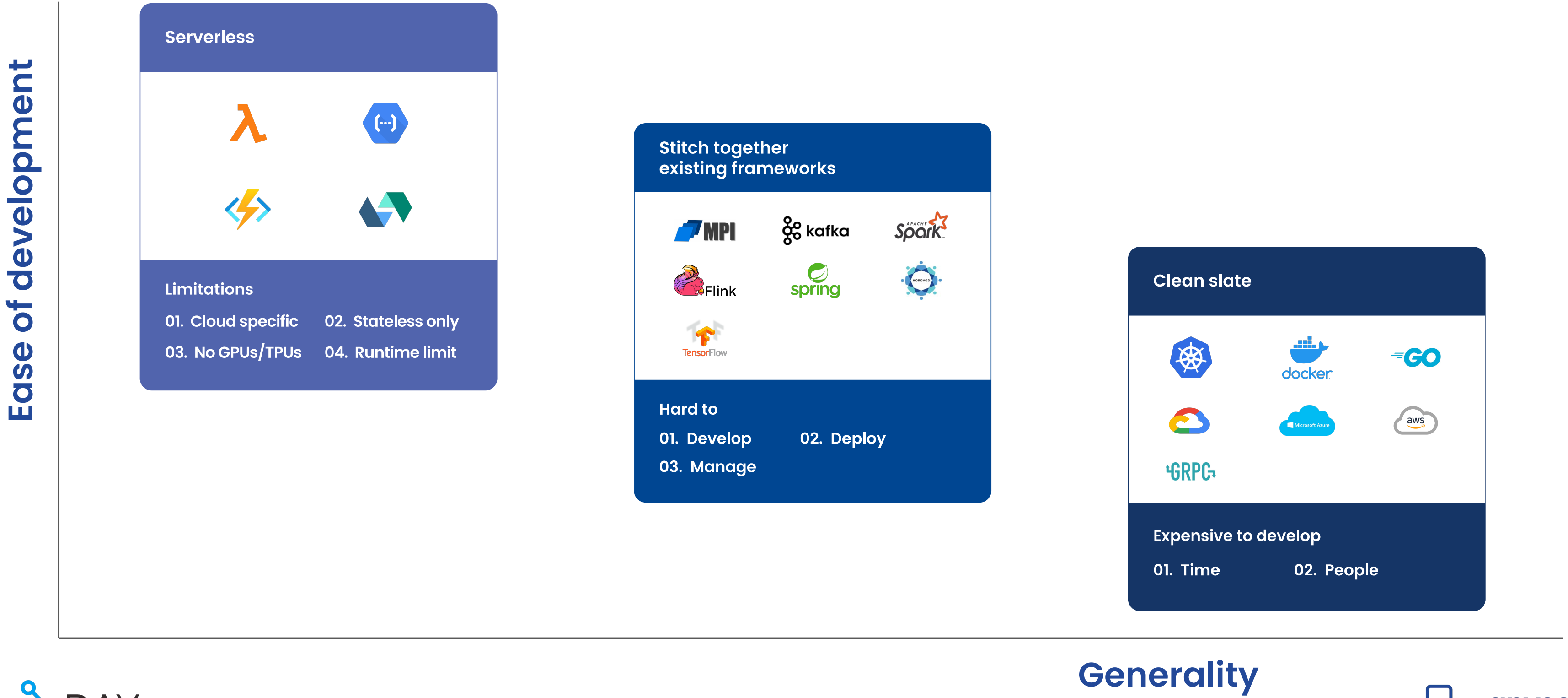
RAY

anyscale

Why Ray?

- Machine learning is pervasive in every domain
- Distributed machine learning is becoming a necessity
- **Distributed systems and programming are notoriously hard**

Existing solutions have may tradeoffs



Serverless

Limitations

01. Cloud specific
02. Stateless only
03. No GPUs/TPUs
04. Runtime limit

Stitch together existing frameworks

Hard to

01. Develop
02. Deploy
03. Manage

Clean slate

Expensive to develop

01. Time
02. People

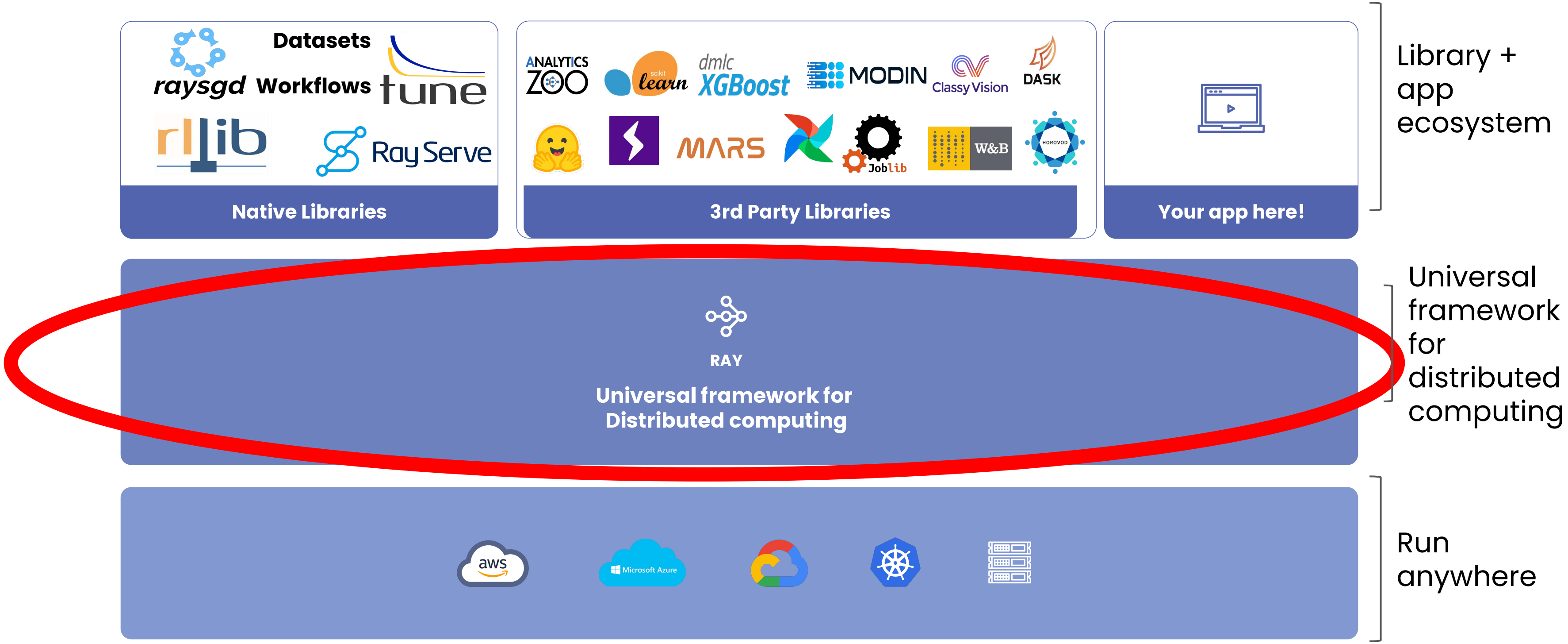
Why Ray?

- Machine learning is pervasive in every domain
- Distributed machine learning is becoming a necessity
- Distributed systems are notoriously hard

Ray's vision:

Make distributed computing accessible to every developer

The Ray Layered Cake and Ecosystem



Native Libraries

3rd Party Libraries

Your app here!

RAY

Universal framework for Distributed computing

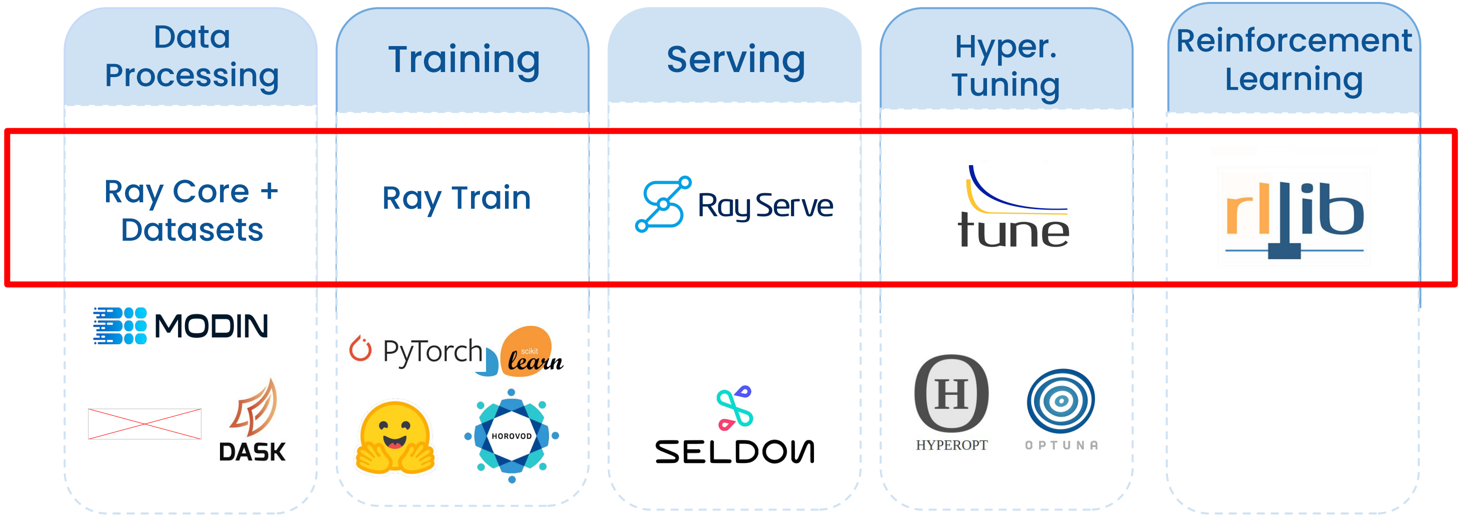
Library + app ecosystem

Universal framework for distributed computing

Run anywhere

Rich ecosystem for scaling ML workloads

Built-in **"batteries included"** libraries



Data Processing

Training

Serving

Hyper. Tuning

Reinforcement Learning

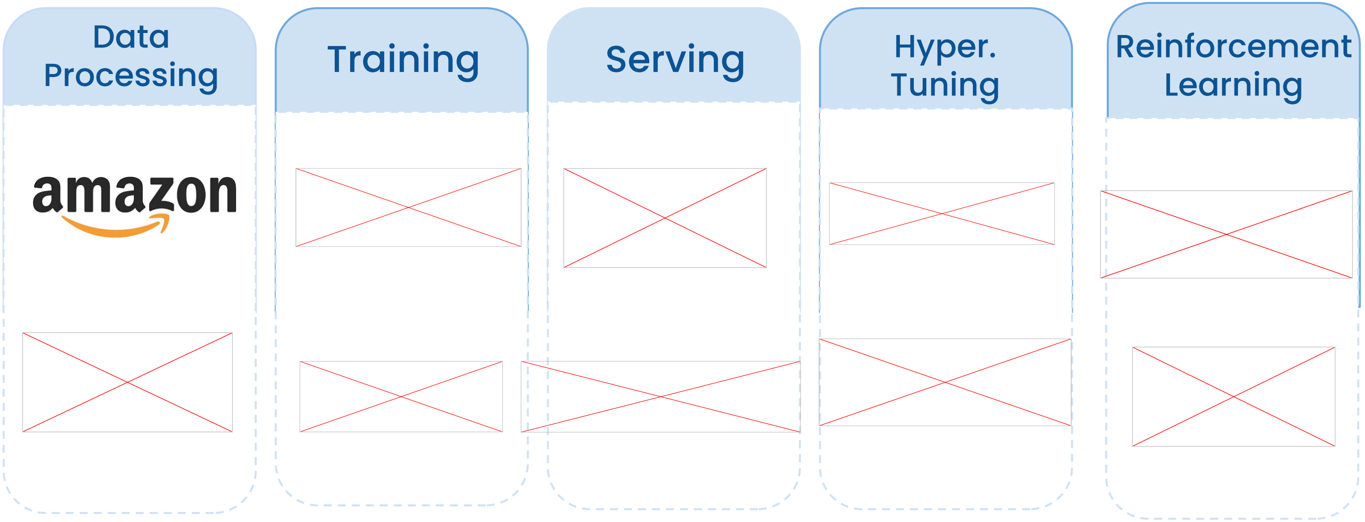
Ray Core + Datasets

Ray Train

Only use the libraries you need!

Companies scaling ML with Ray

Companies scaling ML with Ray



Data Processing

Training

Serving

Hyper. Tuning

Reinforcement Learning

- https://eng.uber.com/horovod-ray/
- https://www.anyscale.com/blog/wildlife-studios-serves-in-game-offers-3x-faster-at-1-10th-the-cost-with-ray
- https://www.ikigailabs.com/blog/how-ikigai-labs-serves-interactive-ai-workflows-at-scale-using-ray-serve

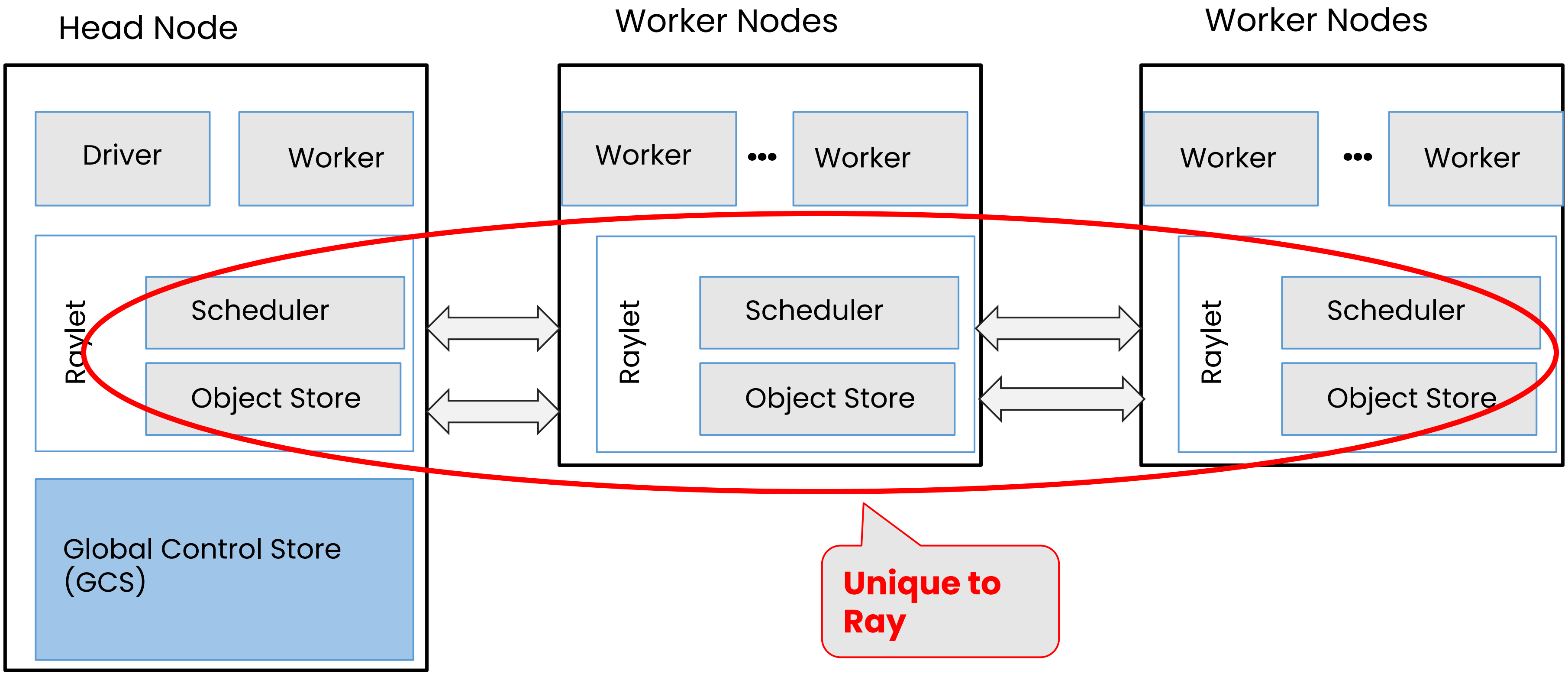
Ray’s approach for scaling ML

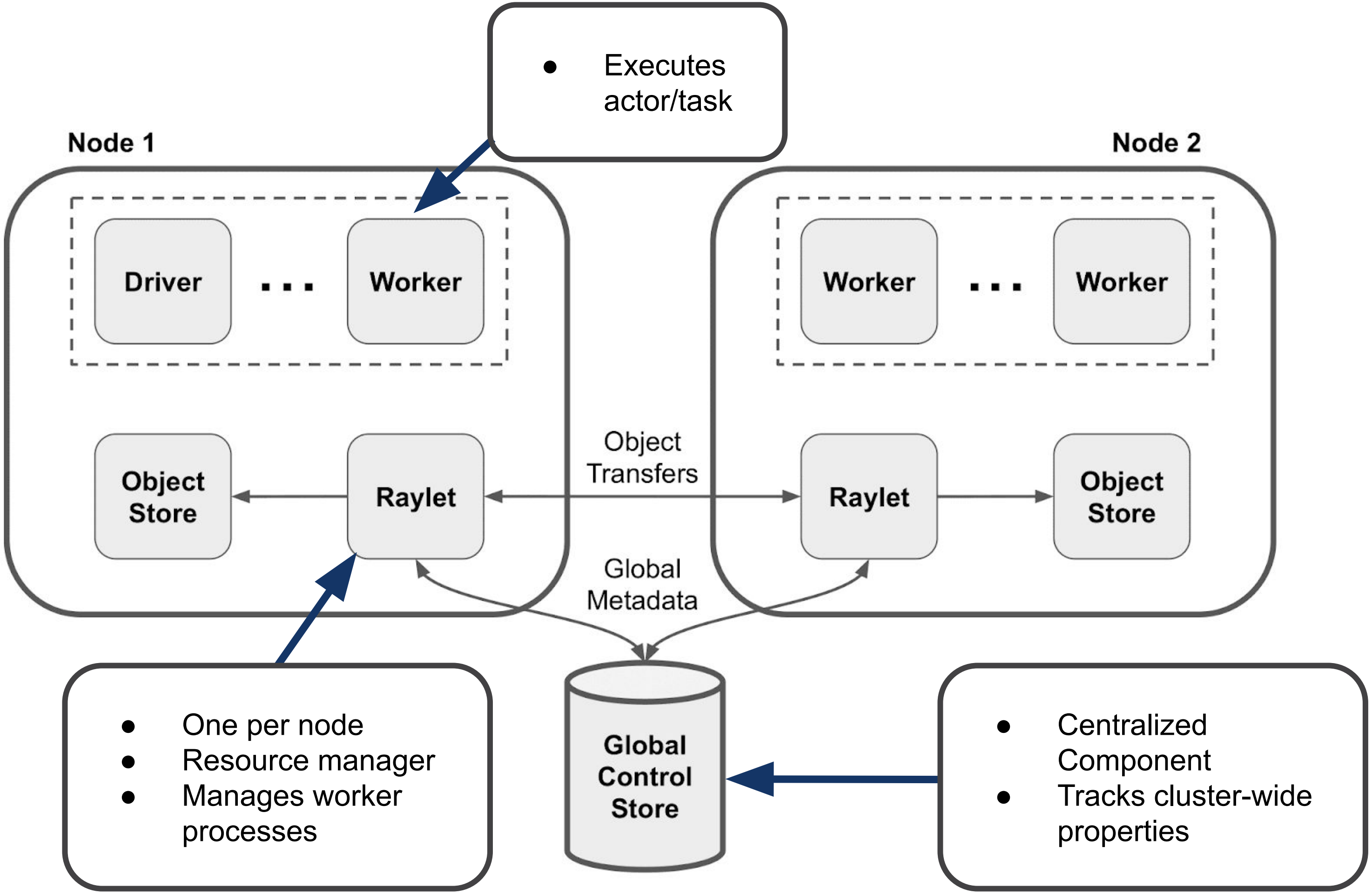
(Traditional, non-parallelized vanilla Python)
(with as few code changes as possible)
Ray-ified Python
Runs on
Runs on
CPU
CPU
CPU
CPU
GPU A
GPU B
GPU B
GPU B

Same Python code runs on laptop as infinite cloud!

Ray Architecture & Components

What does Ray Cluster Looks Like …





- Executes actor/task

- One per node
- Resource manager
- Manages worker processes

- Centralized Component
- Tracks cluster-wide properties

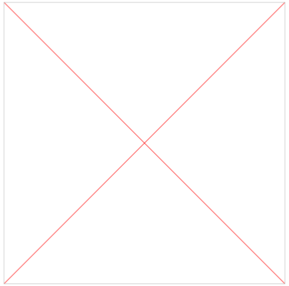
Ray Distributed Design Patterns & APIs

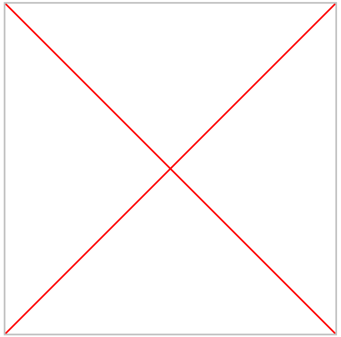
Ray Basic Design Patterns

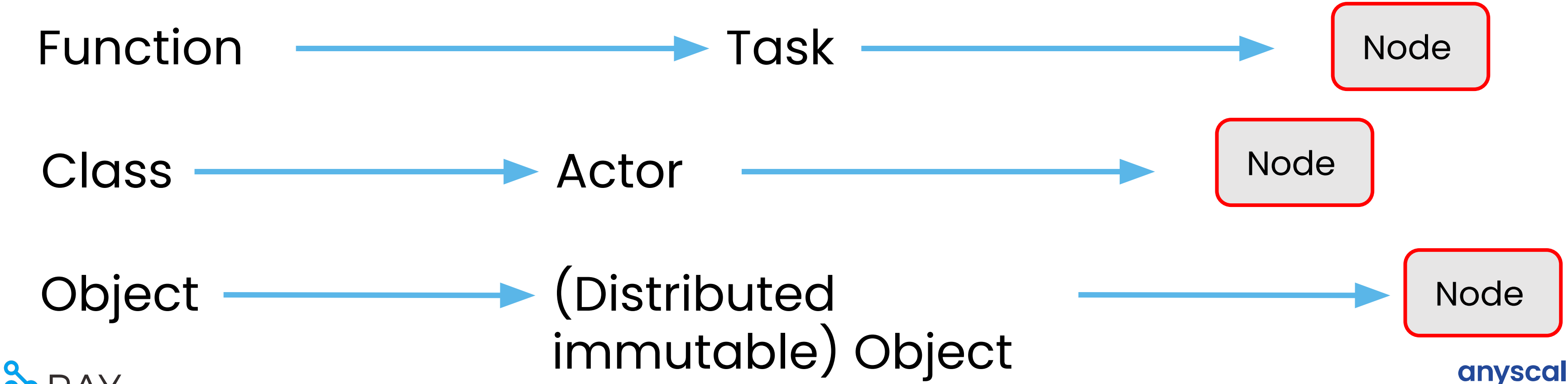
- Ray Parallel Tasks
  - Functions as stateless units of execution
  - Functions distributed across a clusters as tasks
- Ray Objects or Futures
  - Distributed (immutable) Object stored in cluster
  - Retrievable when available
  - Enable asynchronous execution of
- Ray Actors
  - Stateful service on a cluster
  - Message passing and maintains state

1. Patterns for Parallel Programming
2. Ray Design Patterns
3. Ray Distributed Library Integration Patterns

Python → Ray Basic Patterns







Function → Task → Node

Class → Actor → Node

Object → (Distributed immutable) Object → Node

Function → Task

```python
@ray.remote
def read_array(file):
    # read ndarray “a”
    # from “file”
    return a

@ray.remote
def add(a, b):
    return np.add(a, b)

id1 = read_array.remote(file1)
id2 = read_array.remote(file2)
id = add.remote(id1, id2)
sum = ray.get(id)
```

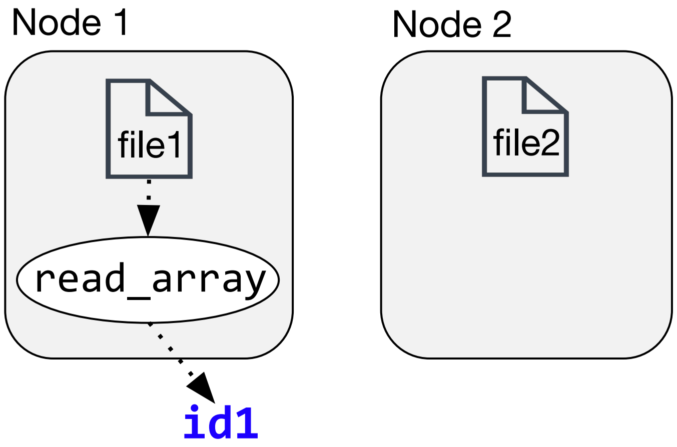
Class → Actor

```python
@ray.remote(num_gpus=1)
class Counter(object):
    def __init__(self):
        self.value = 0
    def inc(self):
        self.value += 1
        return self.value

c = Counter.remote()
id4 = c.inc.remote()
id5 = c.inc.remote()
```

Task API

Blue variables are ObjectRef IDs (similar to futures)

```
@ray.remote
def read_array(file):
    # read ndarray “a”
    # from “file”
    return a

@ray.remote
def add(a, b):
    return np.add(a, b)

id1 = read_array.remote(file1)
id2 = read_array.remote(file2)
id = add.remote(id1, id2)
sum = ray.get(id)
```

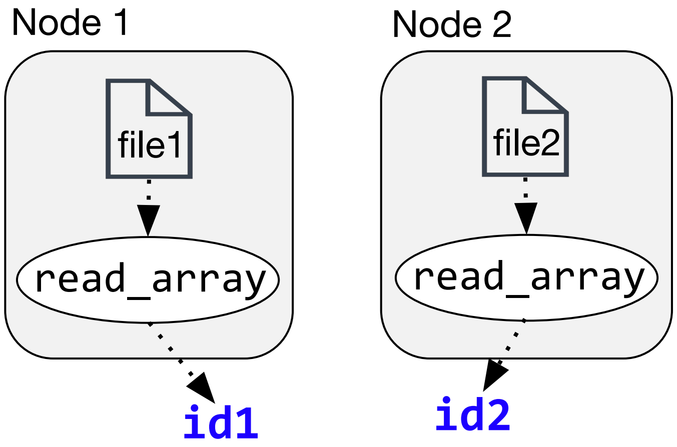


Return id1 (future) immediately, before **read_array()** finishes

Task API

Blue variables are Object IDs (similar to futures)

```
@ray.remote
def read_array(file):
    # read ndarray “a”
    # from “file”
    return a

@ray.remote
def add(a, b):
    return np.add(a, b)

id1 = read_array.remote(file1)
id2 = read_array.remote(file2)
id = add.remote(id1, id2)
sum = ray.get(id)
```

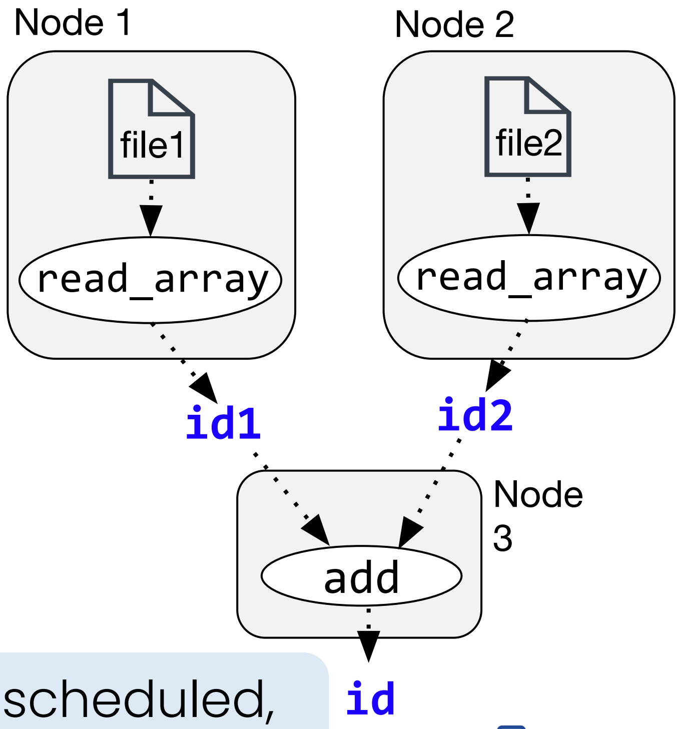


Dynamic task graph: build at runtime

Task API

Blue variables are Object IDs (similar to futures)

```
@ray.remote
def read_array(file):
    # read ndarray “a”
    # from “file”
    return a

@ray.remote
def add(a, b):
    return np.add(a, b)

id1 = read_array.remote(file1)
id2 = read_array.remote(file2)
id = add.remote(id1, id2)
sum = ray.get(id)
```



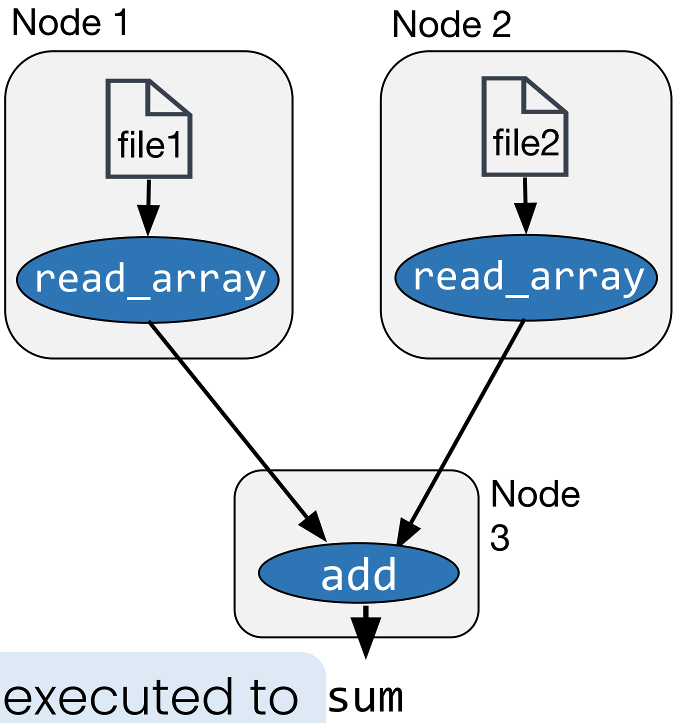
Every task scheduled, but not finished yet

anyscale

Task API

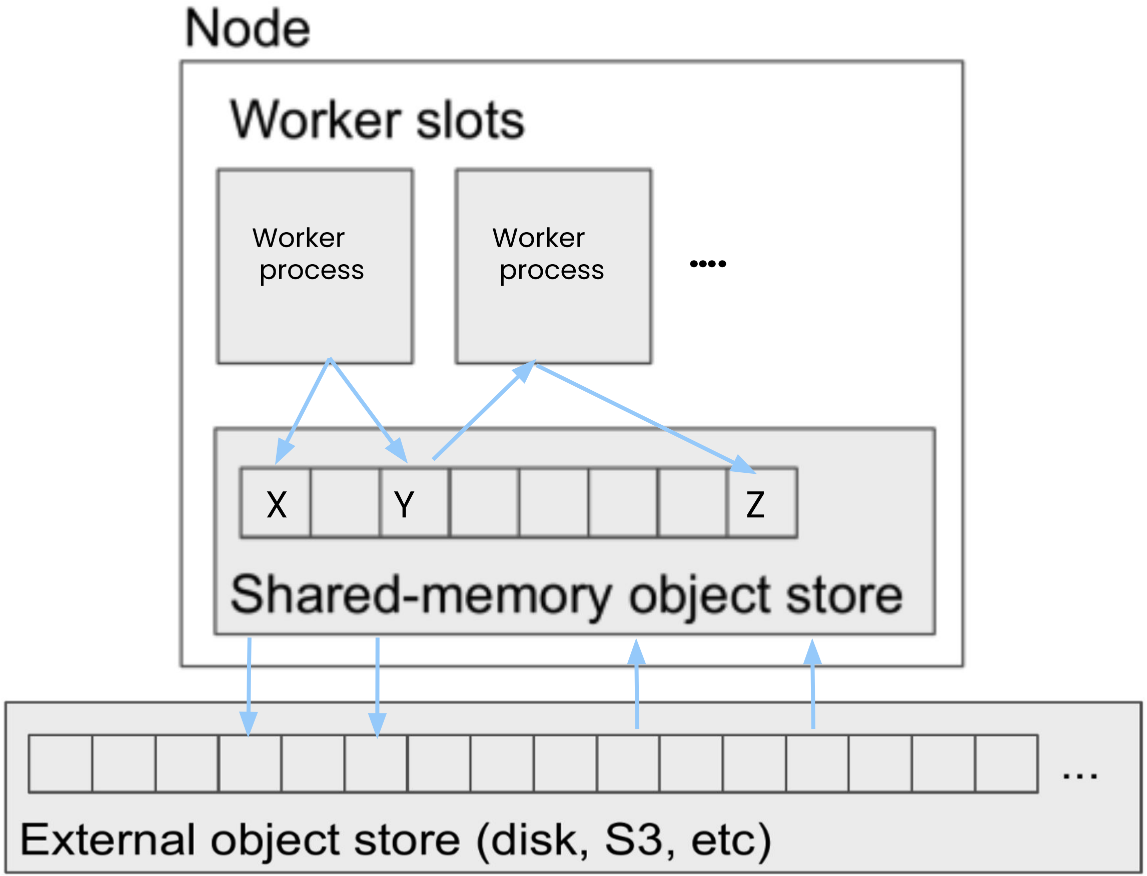
Blue variables are Object IDs (similar to futures)

```
@ray.remote
def read_array(file):
    # read ndarray “a”
    # from “file”
    return a

@ray.remote
def add(a, b):
    return np.add(a, b)

id1 = read_array.remote(file1)
id2 = read_array.remote(file2)
id = add.remote(id1, id2)
sum = ray.get(id)
```



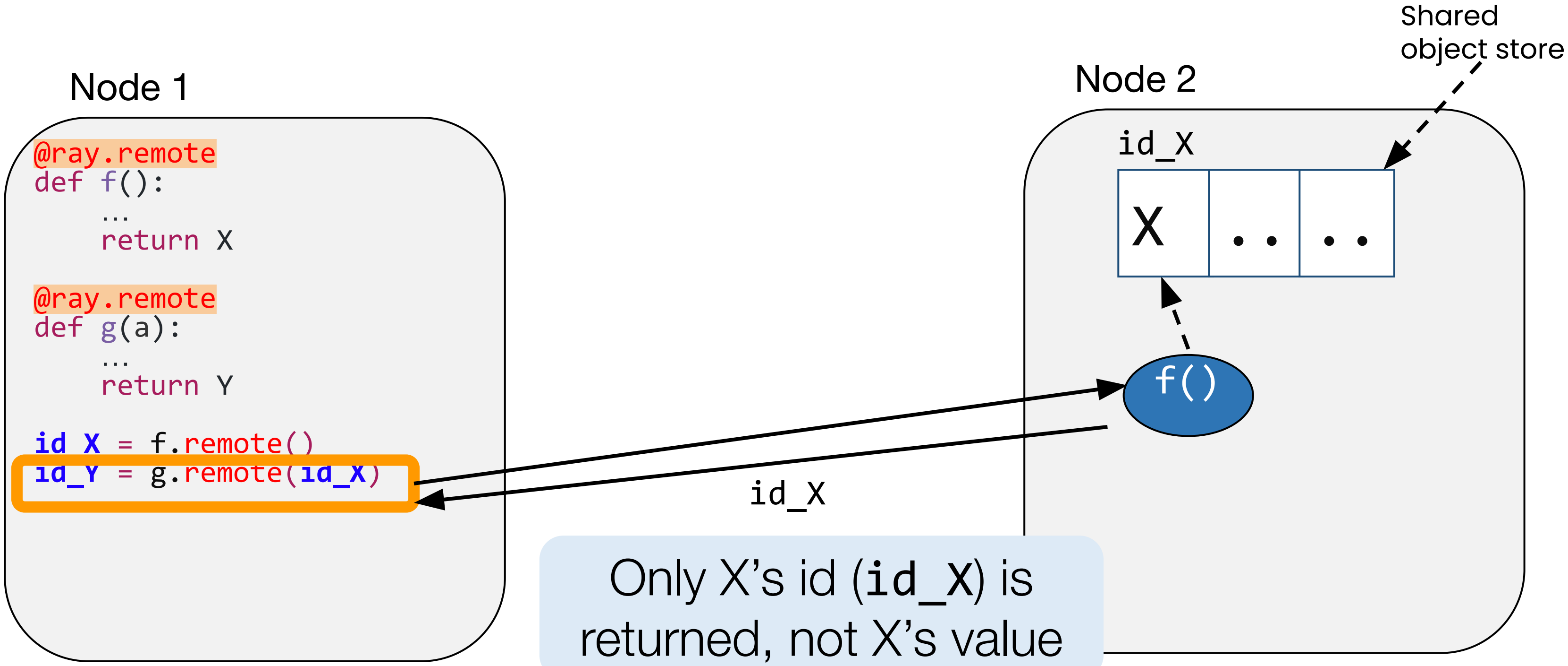
Task graph executed to compute sum

Distributed Immutable obiect store


Node
Worker slots
Worker process
Worker process
....
X
Y
Z
Shared-memory object store
...
External object store (disk, S3, etc)

Spill over to external storage

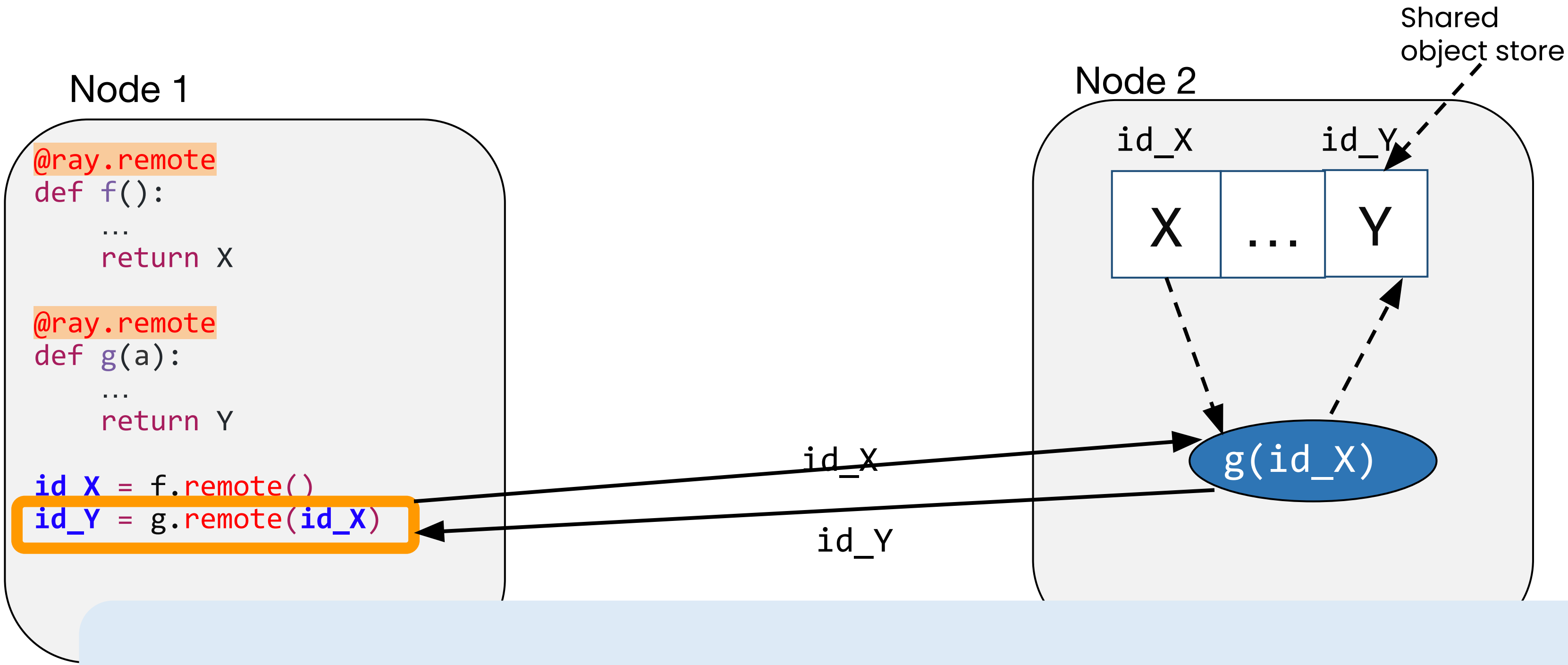
Distributed Immutable object store



```
@ray.remote
def f():
    ...
    return X

@ray.remote
def g(a):
    ...
    return Y

id_X = f.remote()
id_Y = g.remote(id_X)
```

Only X’s id (**id_X**) is returned, not X’s value

Distributed object store



```
@ray.remote
def f():
    ...
    return X

@ray.remote
def g(a):
    ...
    return Y

id_X = f.remote()
id_Y = g.remote(id_X)
```

`g(id_X)` is scheduled on same node, so X is never transferred

Ray Ecosystem

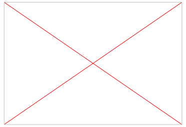
- Ray Tune
- XGBoost-Ray

Ray Tune

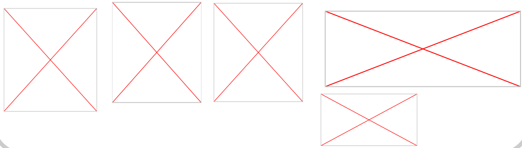
Ray Tune - For distributed HPO

- Efficient algorithms that enable running trials in parallel
- Effective orchestration of distributed trials
- Easy to use APIs
- Interoperable with Ray Train and Ray Datasets
- Saves cost (early stopping bad trials)

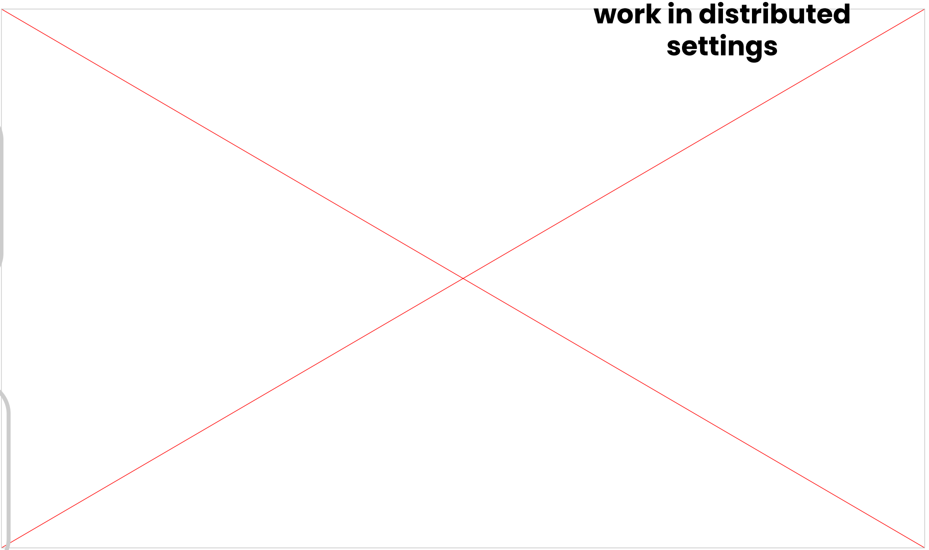
Cutting edge optimization algorithms

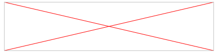


Compatible with ML ecosystem



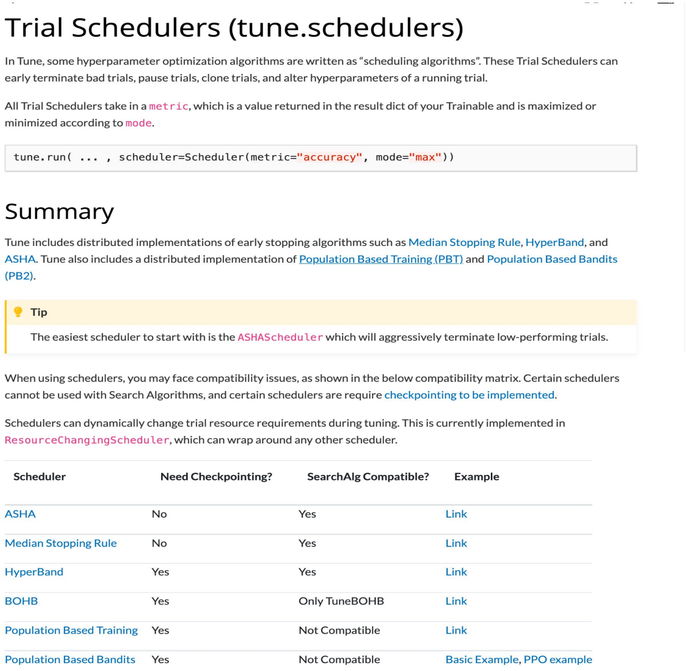
Minimal code changes to work in distributed settings





Ray

https://docs.ray.io/en/latest/tune/api_docs/suggestion.html #tune-search-alg



Trial Schedulers (tune.schedulers)

In Tune, some hyperparameter optimization algorithms are written as “scheduling algorithms”. These Trial Schedulers can early terminate bad trials, pause trials, clone trials, and alter hyperparameters of a running trial.

All Trial Schedulers take in a `metric`, which is a value returned in the result dict of your Trainable and is maximized or minimized according to `mode`.

```
tune.run( ... , scheduler=Scheduler(metric="accuracy", mode="max"))
```

Summary

Tune includes distributed implementations of early stopping algorithms such as Median Stopping Rule, HyperBand, and ASHA. Tune also includes a distributed implementation of Population Based Training (PBT) and Population Based Bandits (PB2).

Tip

The easiest scheduler to start with is the `ASHAScheduler` which will aggressively terminate low-performing trials.

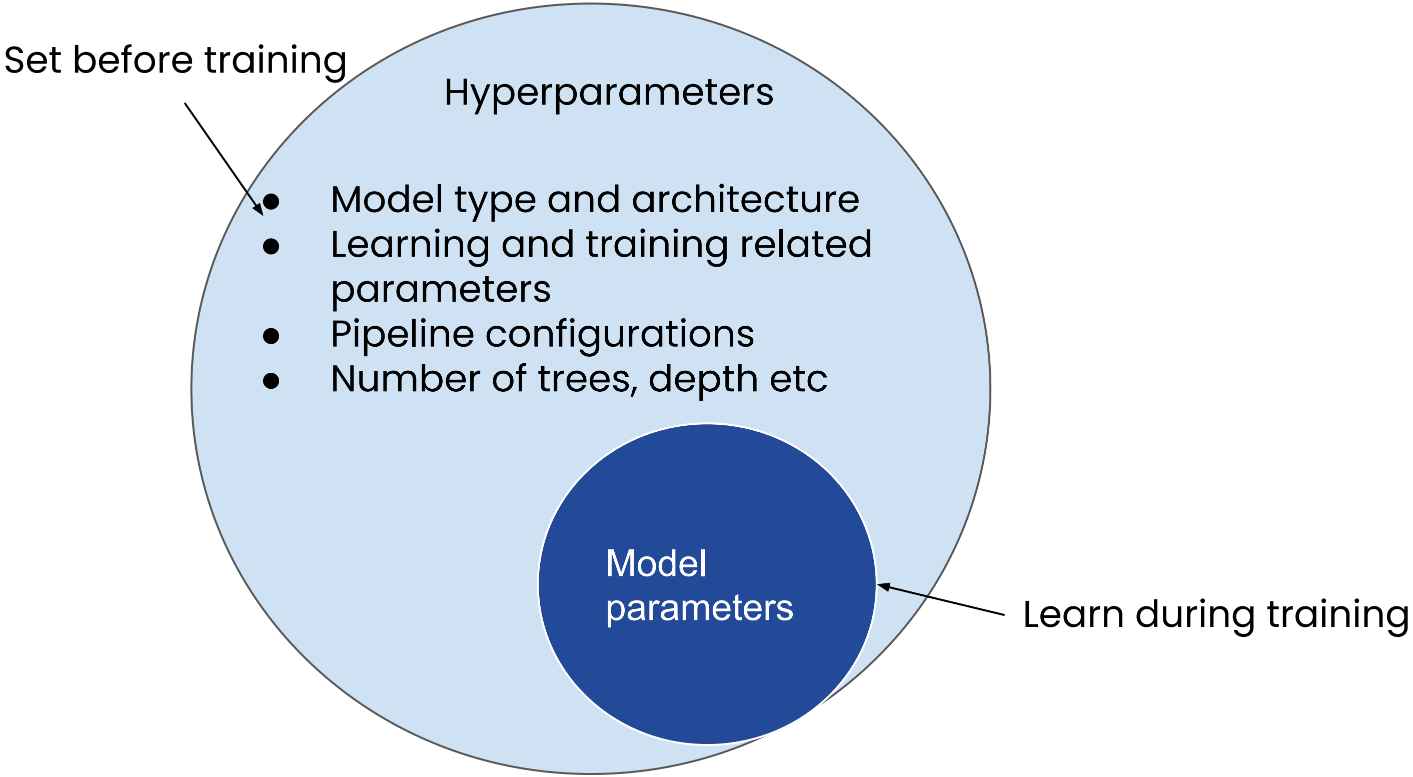
When using schedulers, you may face compatibility issues, as shown in the below compatibility matrix. Certain schedulers cannot be used with Search Algorithms, and certain schedulers are require checkpointing to be implemented.

Schedulers can dynamically change trial resource requirements during tuning. This is currently implemented in `ResourceChangingScheduler`, which can wrap around any other scheduler.

Scheduler	Need Checkpointing?	SearchAlg Compatible?	Example
ASHA	No	Yes	Link
Median Stopping Rule	No	Yes	Link
HyperBand	Yes	Yes	Link
BOHB	Yes	Only TuneBOHB	Link
Population Based Training	Yes	Not Compatible	Link
Population Based Bandits	Yes	Not Compatible	Basic Example, PPO example

https://docs.ray.io/en/latest/tune/api_docs/schedulers.html#tune-schedulers

Hyperparameters



Set before training

Hyperparameters

- Model type and architecture
- Learning and training related parameters
- Pipeline configurations
- Number of trees, depth etc

Model parameters

Learn during training

Hyperparameter tuning

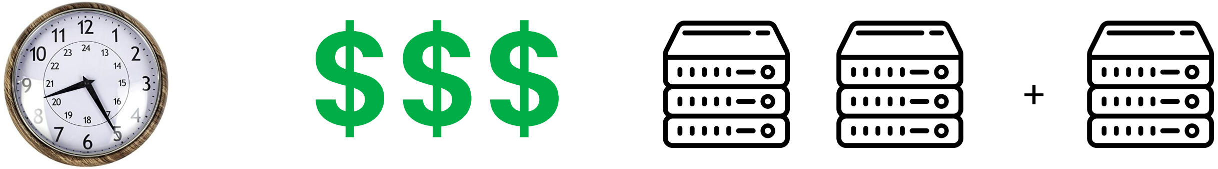
“choosing a set of optimal hyperparameters for a learning algorithm”

<u>Example</u>: what network structure is best for your binary classification problem?

How many layers? What kinds of layers? Learning rate schedule?
Every number here is a hyperparameter!

HPO Challenges at scale

- Time consuming and costly
- Use Resources (GPUs/CPUs) at lower costs
- Fault-tolerance and elasticity



Ray Tune - HPO algorithms

- Over 15+ algorithms natively provided or integrated
- Easy to swap out different algorithms with no code change

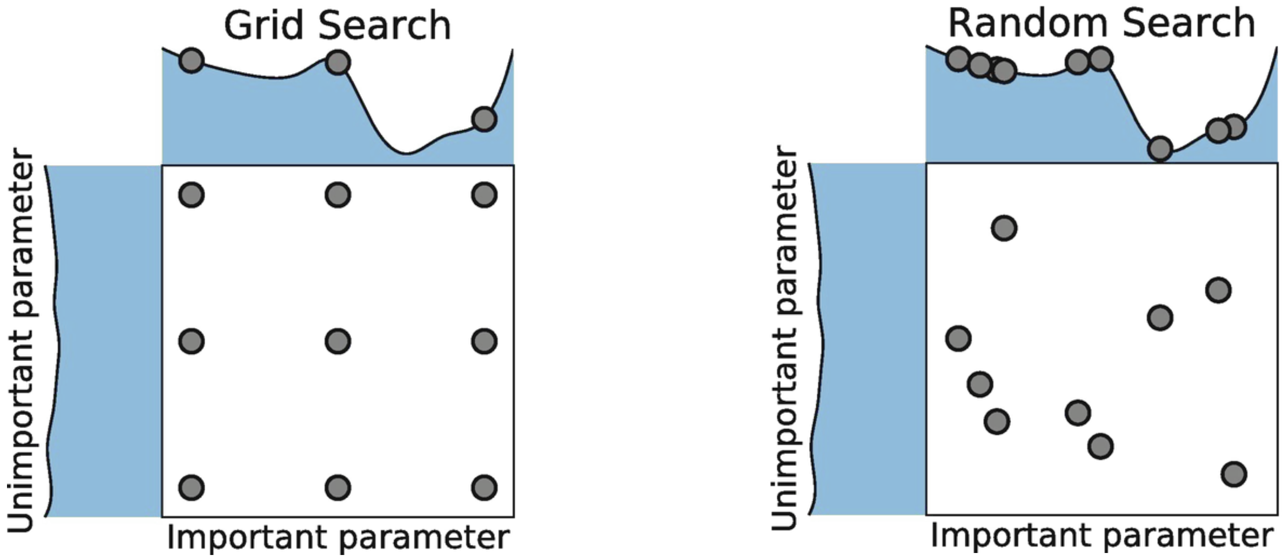
01 **Exhaustive Search**

02 **Bayesian Optimization**

03 **Advanced Scheduling**

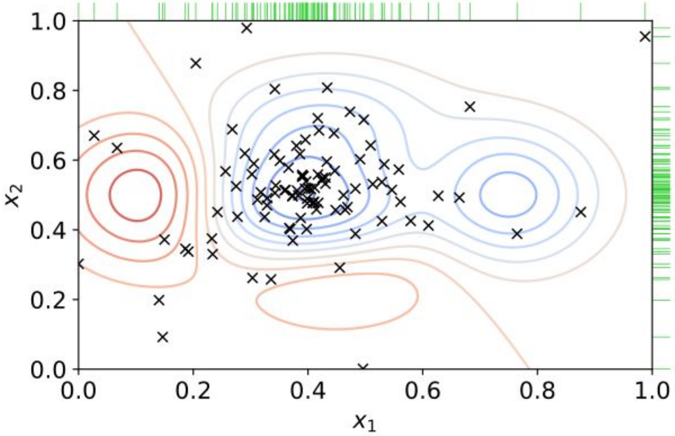
Exhaustive Search

- Easily parallelizable, easy to implement
- Inefficient, compute intensive



Bayesian optimization

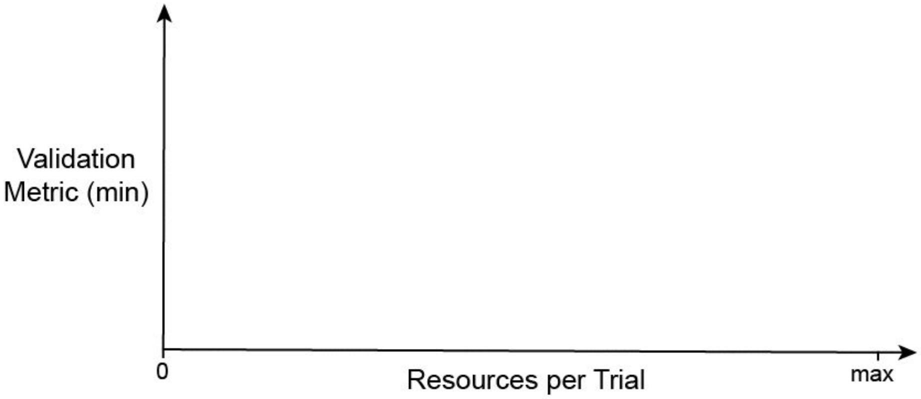
- Uses results from previous combinations (trials) to decide which trial to try next
- Inherently sequential
- Popular libraries:
  - hyperopt
  - Optuna
  - Scikit-optimize
  - Nevergrad



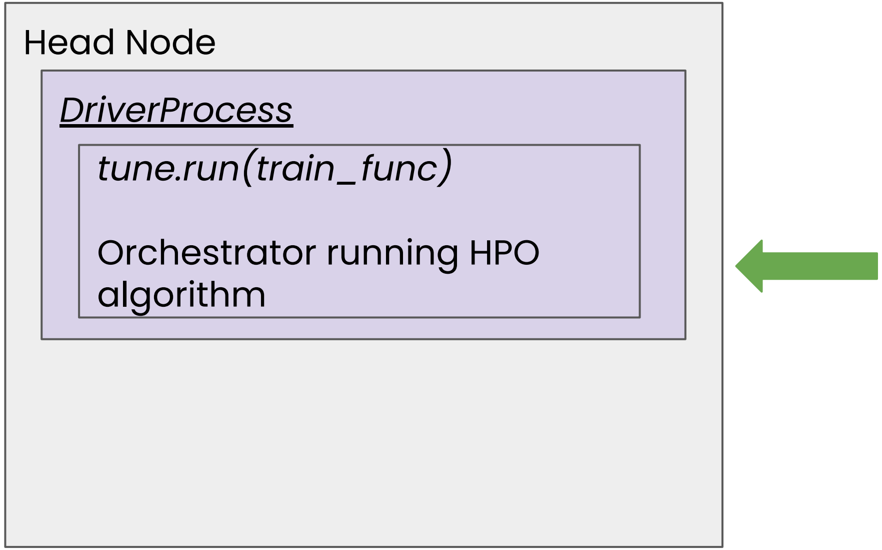
https://www.wikiwand.com/en/Hyperparameter_optimization

Advanced Scheduling - Early stopping

- Fan out parallel trials during the initial exploration phase
- Use intermediate results (epochs, trees, samples) to prune underperforming trials, saving time and computing resources
- Median stopping, ASHA/Hyperband
- Can be combined with Bayesian Optimization (BOHB)



Ray Tune - *distributed* HPO



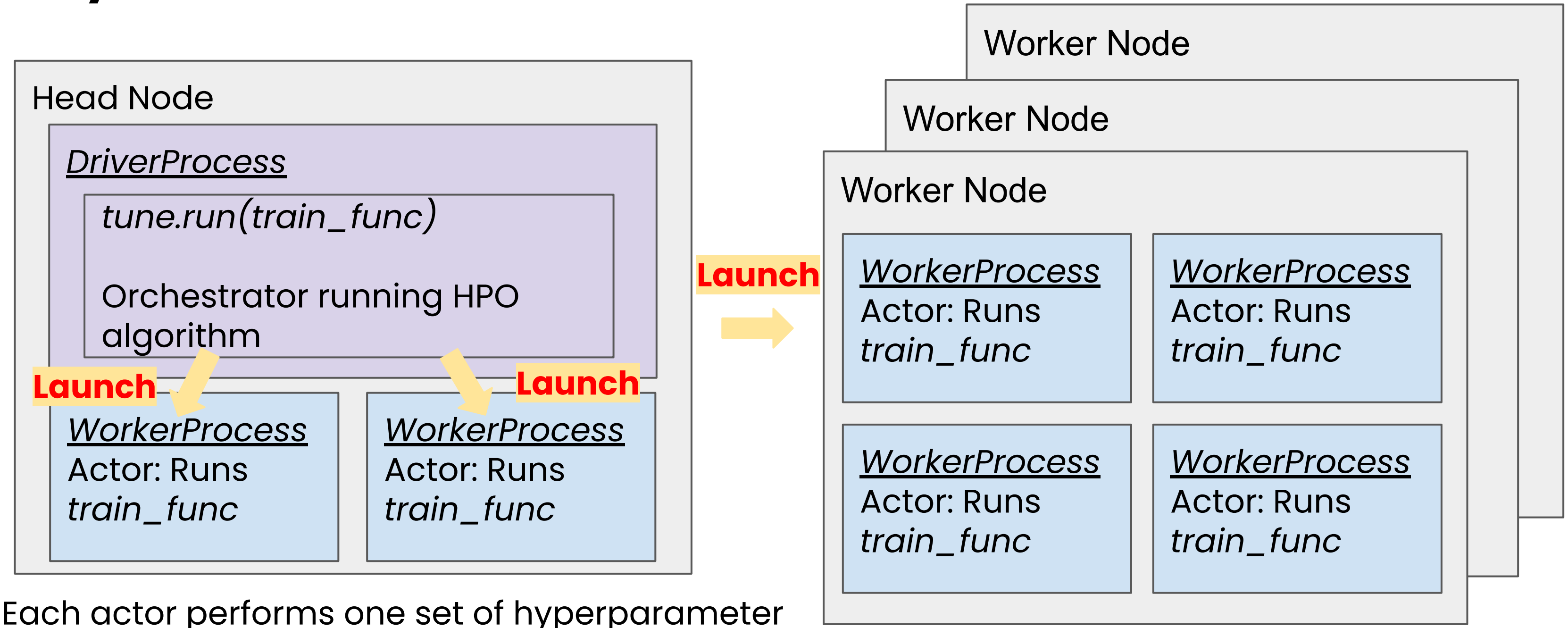
```
from ray import tune

def train_func(config):
    model = ConvNet(config)
    for i in range(epochs):
        current_loss = model.train()
        tune.report(loss=current_loss)

tune.run(
    train_func,
    config={“alpha”: tune.uniform(0.001,
0.1)},
    num_samples=100,
    scheduler=“asha”,
    search_alg=”optuna”)
```

Easily define your training function

Just use tun.run(..)

Easily specify hyperparameter ranges to search over

Ray Tune - *distributed* HPO



Head Node

DriverProcess

tune.run(train_func)

Orchestrator running HPO algorithm

Launch

Launch

WorkerProcess
Actor: Runs
train_func

WorkerProcess
Actor: Runs
train_func

Launch

Worker Node

Worker Node

Worker Node

WorkerProcess
Actor: Runs
train_func

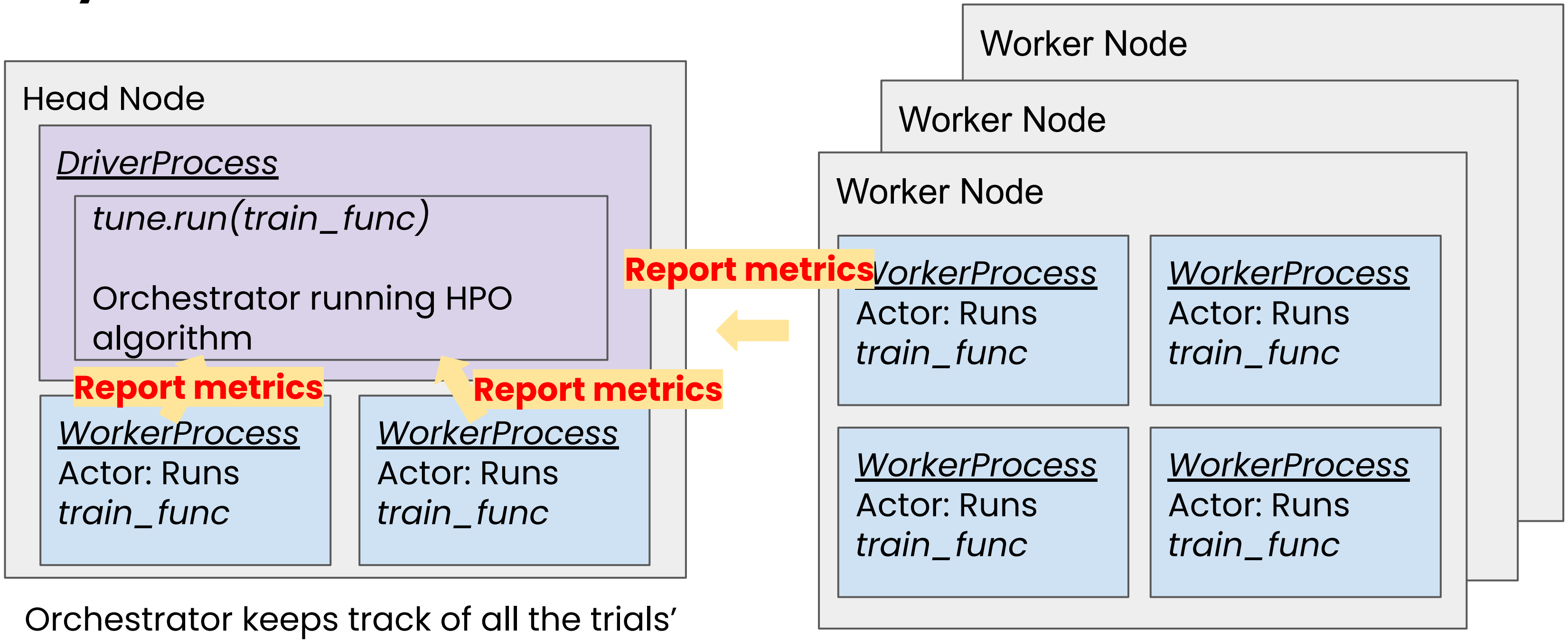
WorkerProcess
Actor: Runs
train_func

WorkerProcess
Actor: Runs
train_func

WorkerProcess
Actor: Runs
train_func

Each actor performs one set of hyperparameter combination evaluation (a trial)

Ray Tune - *distributed* HPO



Head Node

DriverProcess

tune.run(train_func)

Orchestrator running HPO algorithm

Report metrics

WorkerProcess
Actor: Runs *train_func*

Report metrics

WorkerProcess
Actor: Runs *train_func*

Report metrics

Worker Node

Worker Node

Worker Node

WorkerProcess
Actor: Runs *train_func*

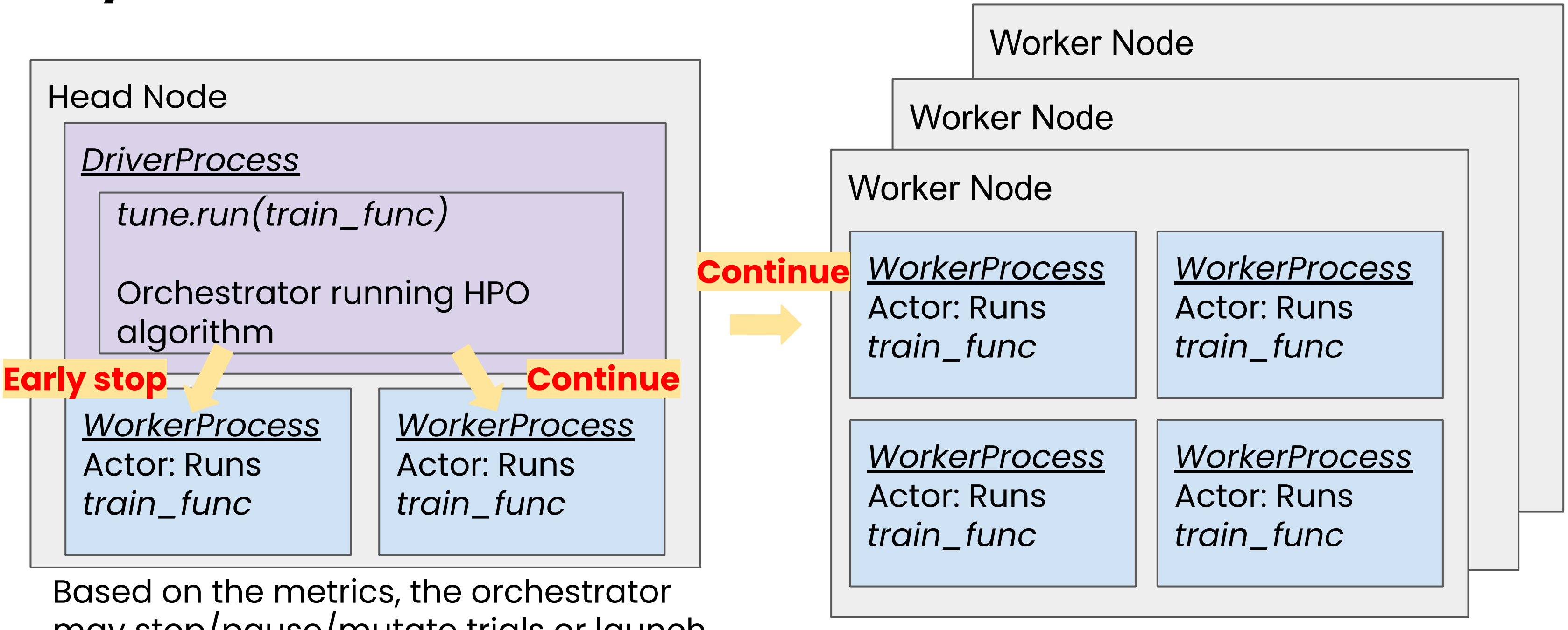
WorkerProcess
Actor: Runs *train_func*

WorkerProcess
Actor: Runs *train_func*

WorkerProcess
Actor: Runs *train_func*

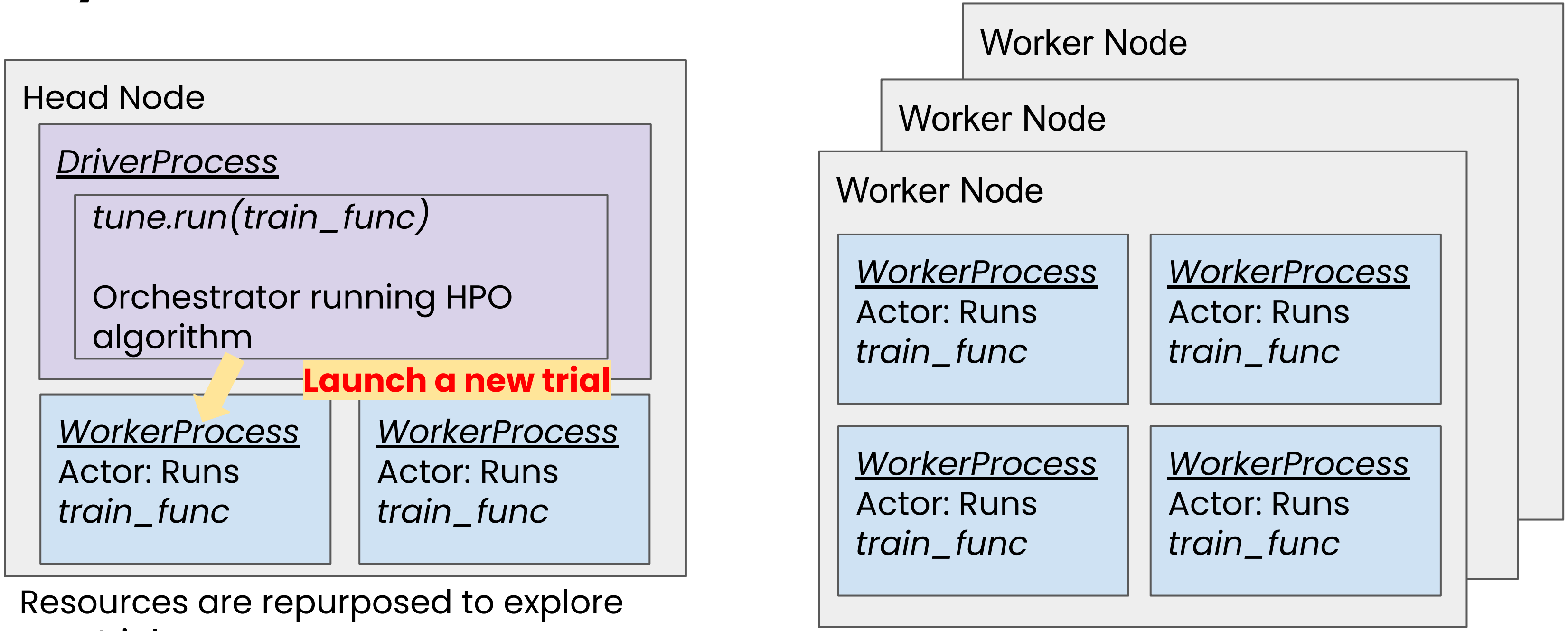
Orchestrator keeps track of all the trials' progress and metrics.

Ray Tune - *distributed* HPO



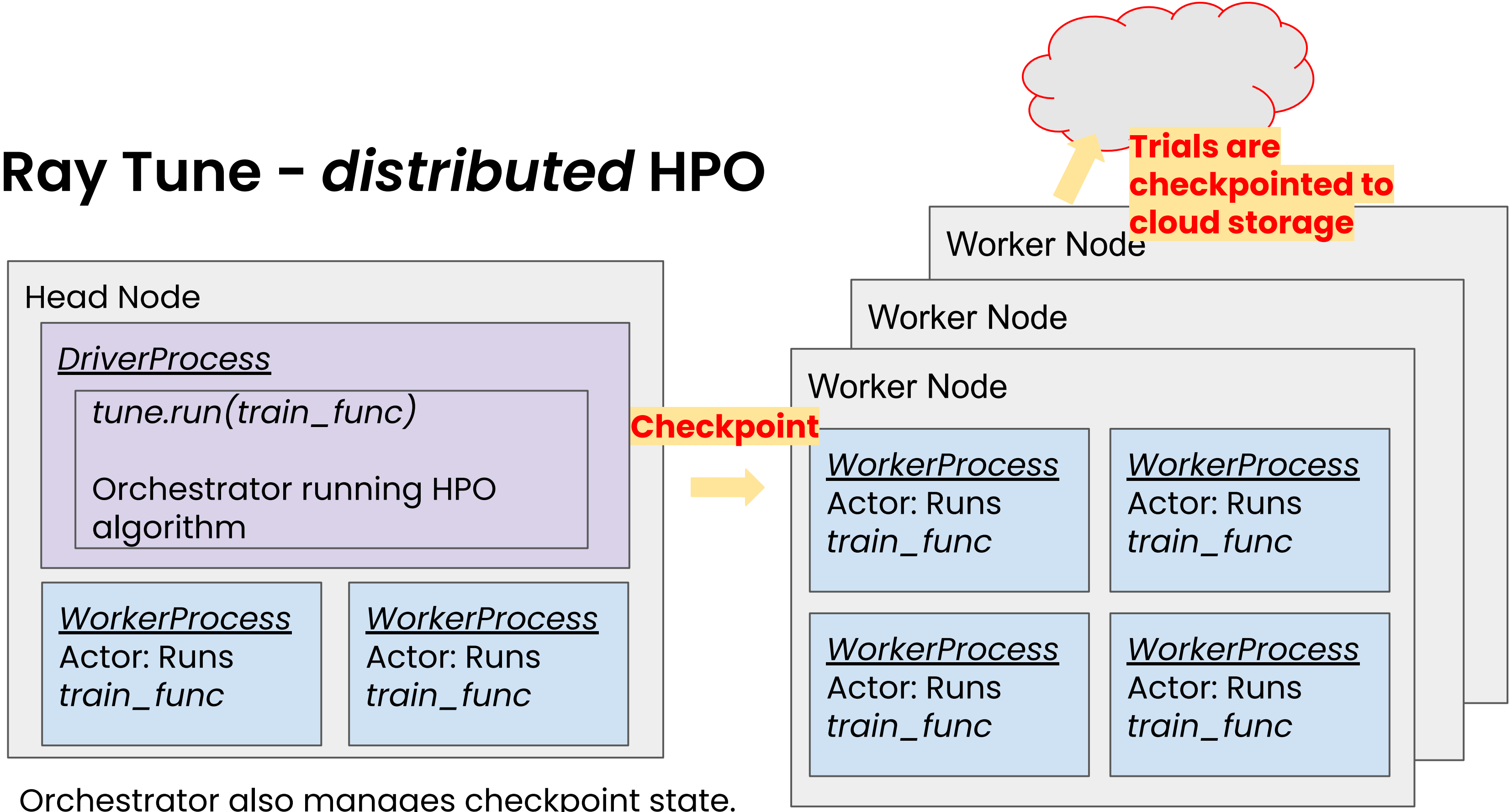
Based on the metrics, the orchestrator may stop/pause/mutate trials or launch new trials when resources are available.

Ray Tune - *distributed* HPO



Resources are repurposed to explore new trials.

Ray Tune - *distributed* HPO



Trials are checkpointed to cloud storage

Head Node

DriverProcess

tune.run(train_func)

Orchestrator running HPO algorithm

WorkerProcess
Actor: Runs *train_func*

WorkerProcess
Actor: Runs *train_func*

Checkpoint

Worker Node

Worker Node

Worker Node

WorkerProcess
Actor: Runs *train_func*

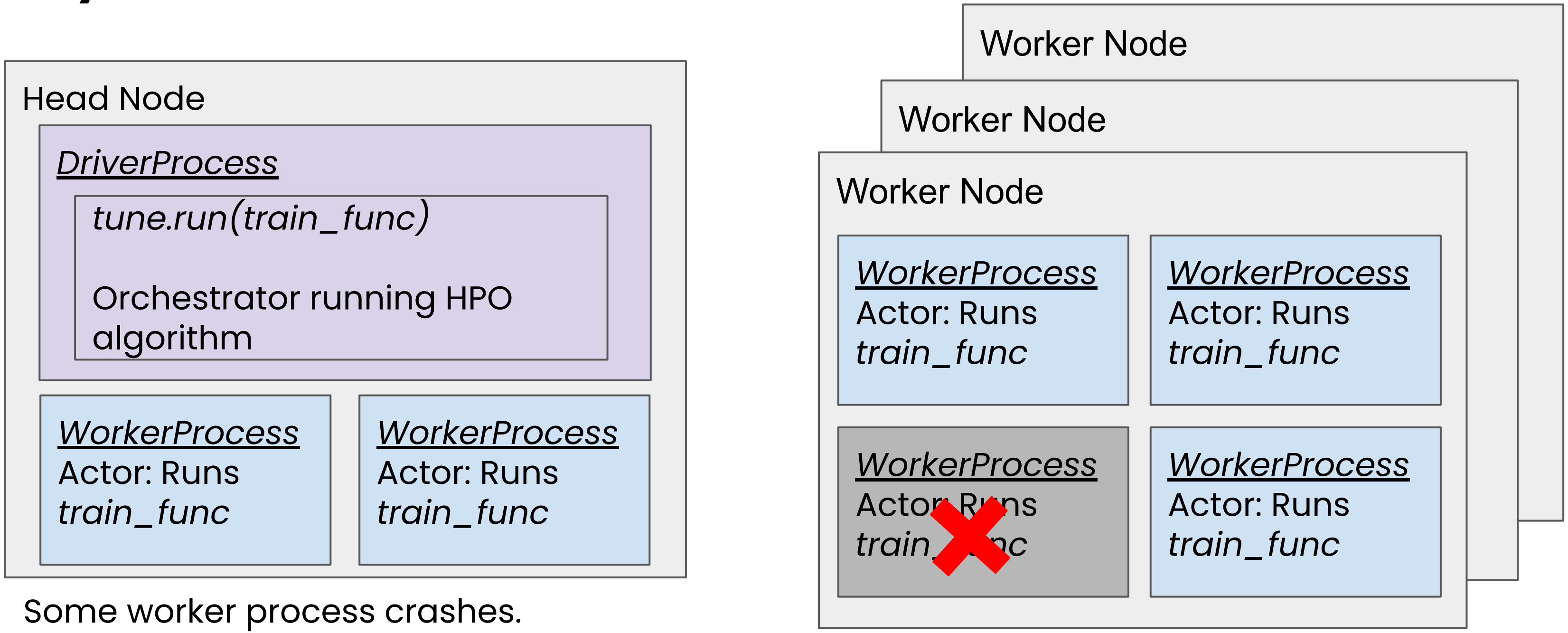
WorkerProcess
Actor: Runs *train_func*

WorkerProcess
Actor: Runs *train_func*

WorkerProcess
Actor: Runs *train_func*

Orchestrator also manages checkpoint state.

Ray Tune - *distributed* HPO



Head Node

DriverProcess

tune.run(train_func)

Orchestrator running HPO algorithm

WorkerProcess
Actor: Runs
train_func

WorkerProcess
Actor: Runs
train_func

Worker Node

Worker Node

Worker Node

WorkerProcess
Actor: Runs
train_func

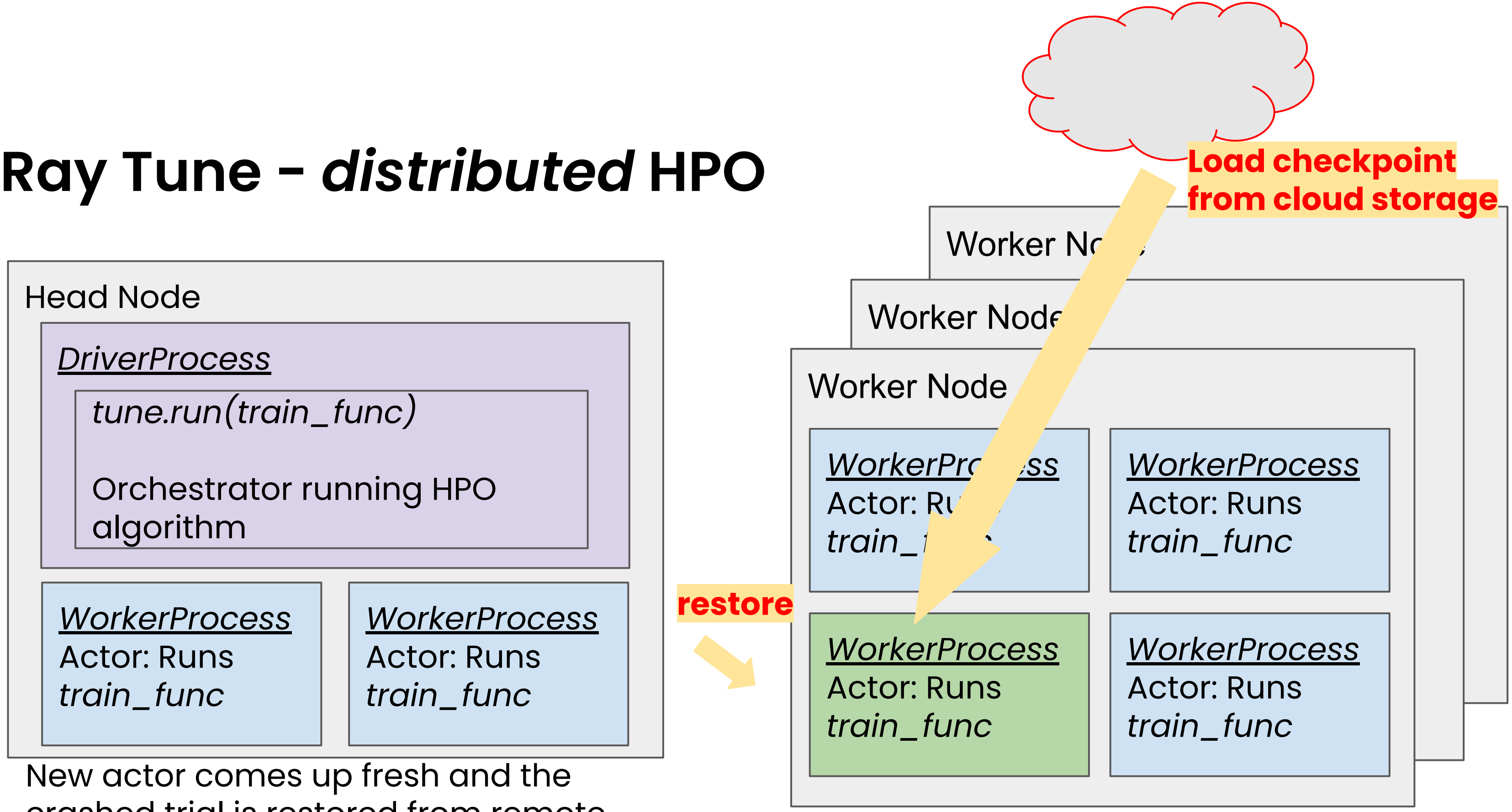
WorkerProcess
Actor: Runs
train_func

WorkerProcess
Actor: Runs
train_func

WorkerProcess
Actor: Runs
train_func

Some worker process crashes.

Ray Tune - *distributed* HPO



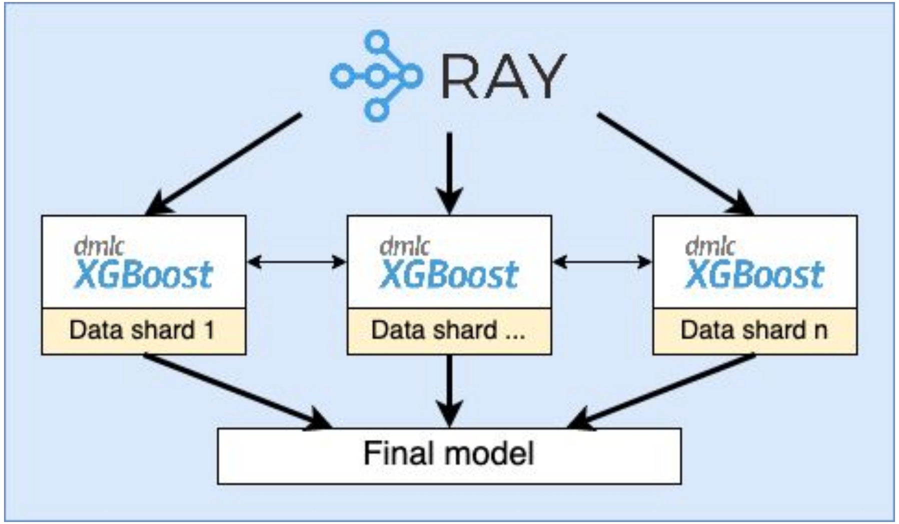
New actor comes up fresh and the crashed trial is restored from remote checkpoint.

XGBoost-Ray

- Design & Features

XGBoost-Ray

- Distributed XGBoost-Ray - Drop-in replacement for XGBoost
- Fault tolerance & Elastic training
- Integration with Ray Datasets and Ray Tune



- https://github.com/ray-project/xgboost_ray
- https://docs.ray.io/en/latest/xgboost-ray.html

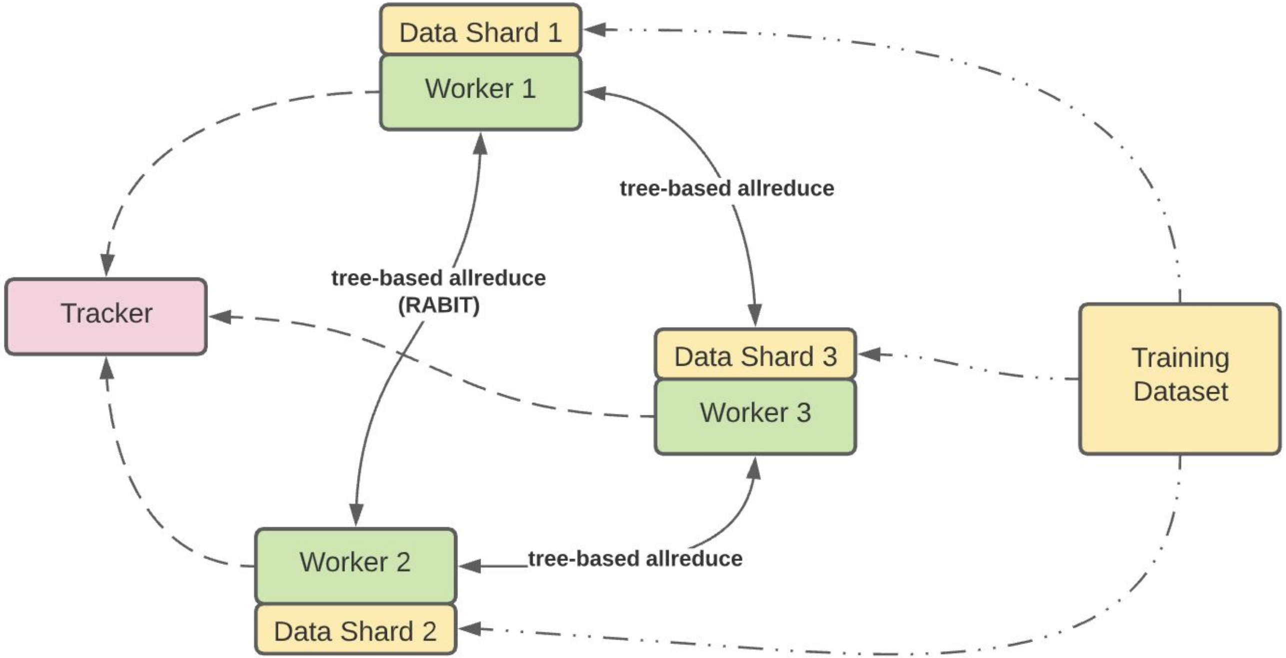
Motivation

- There are existing solutions for distributed XGBoost
  - E.g. Apache Spark, Dask, Kubernetes etc
- But most existing solutions have shortcomings:
  - Dynamic computation graphs
  - Fault tolerance handling
  - GPU support
  - Integration with hyperparameter tuning libraries

XGBoost-Ray

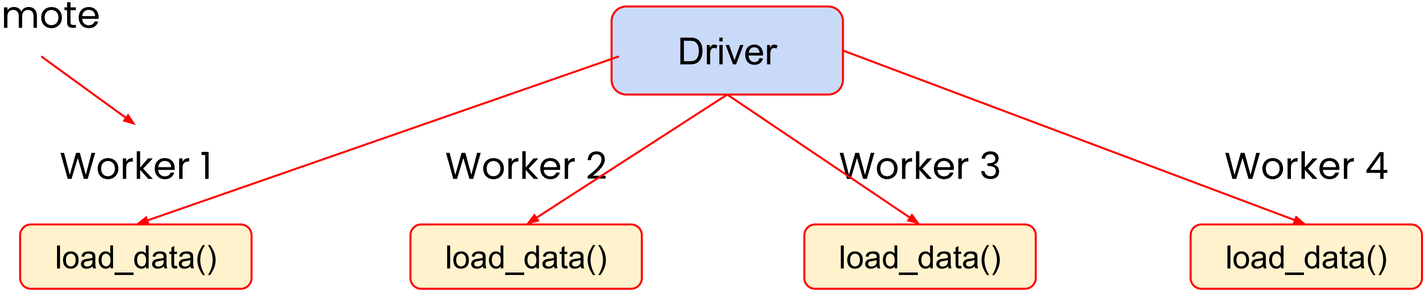
- Ray actors for stateful training workers
- Advanced fault tolerance mechanisms
- Full (multi) GPU support
- Locality-aware distributed data loading
- Integration with Ray Tune

Distributed XGBoost Architecture



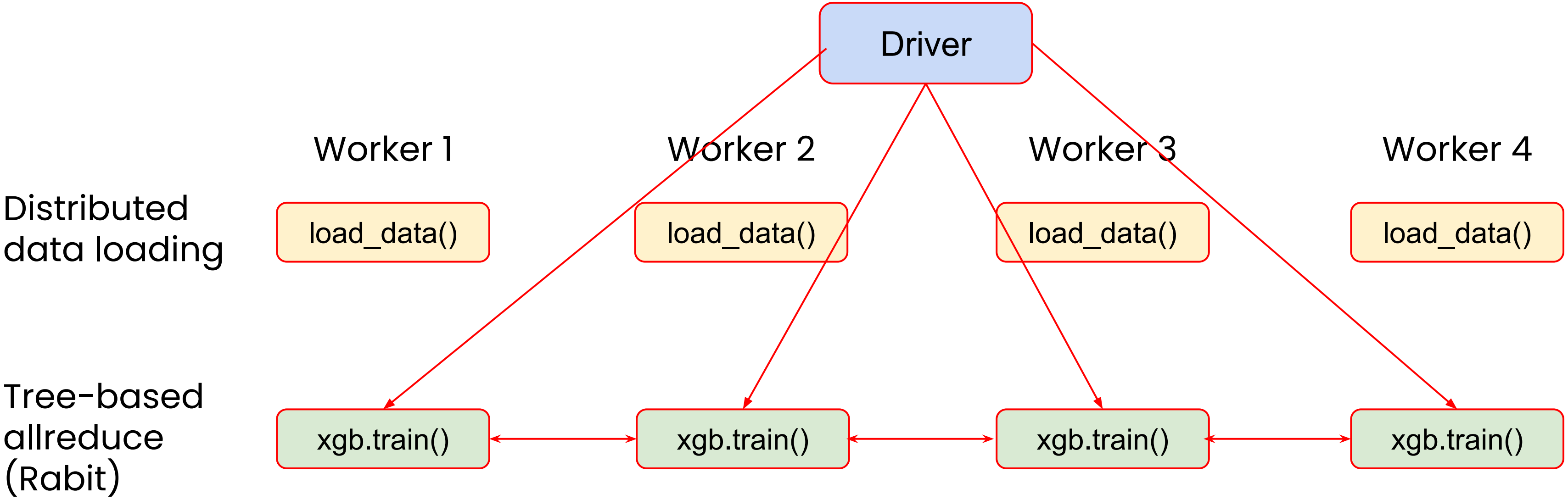
Architecture

@ray.remote
Actors



Distributed
data loading

Architecture



Driver

Worker 1

Worker 2

Worker 3

Worker 4

Distributed data loading

load_data()

load_data()

load_data()

load_data()

Tree-based allreduce (Rabit)

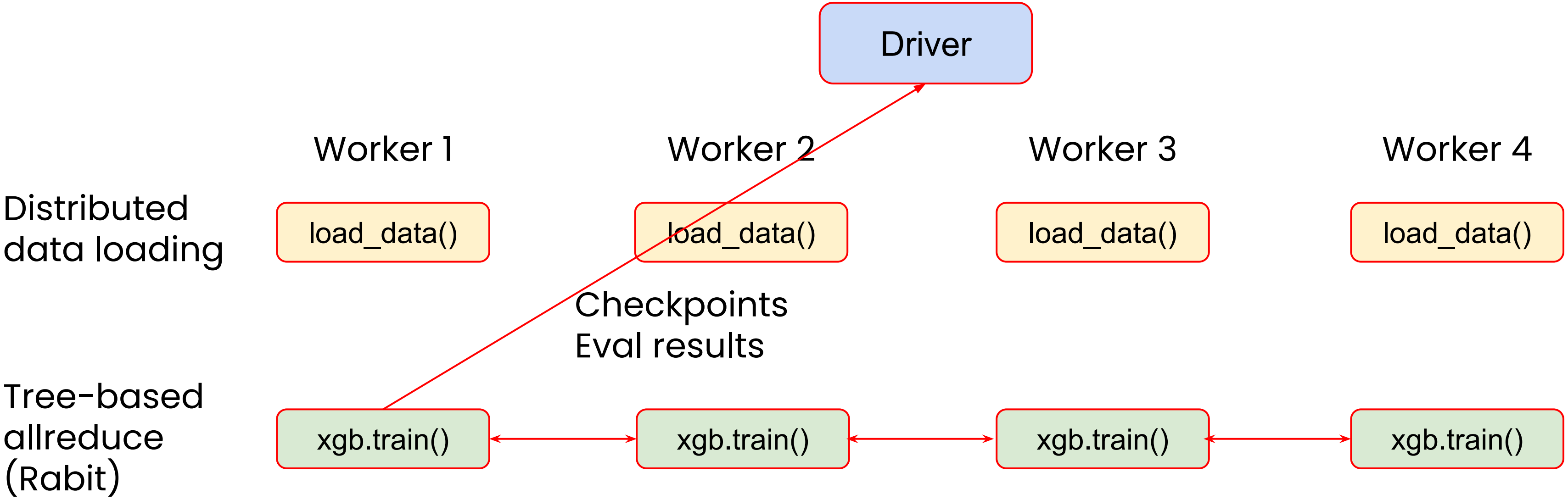
xgb.train()

xgb.train()

xgb.train()

xgb.train()

Architecture



Driver

Worker 1

Worker 2

Worker 3

Worker 4

Distributed data loading

load_data()

load_data()

load_data()

load_data()

Checkpoints
Eval results

Tree-based allreduce (Rabit)

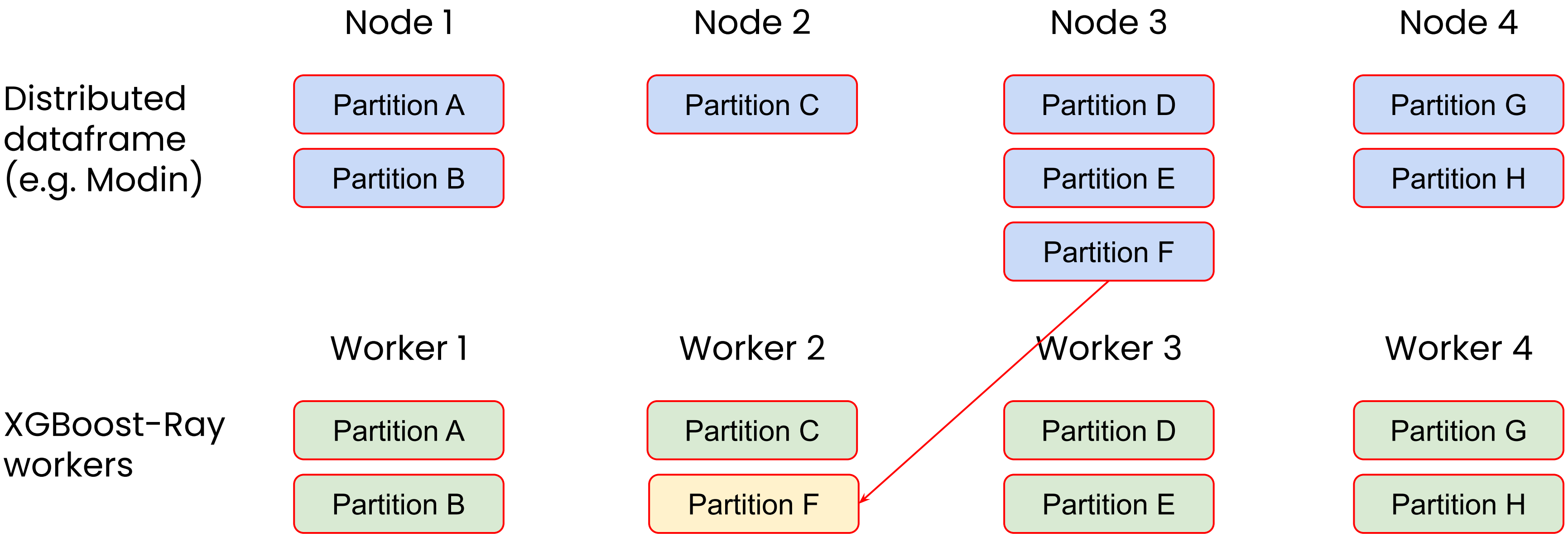
xgb.train()

xgb.train()

xgb.train()

xgb.train()

Distributed data loading



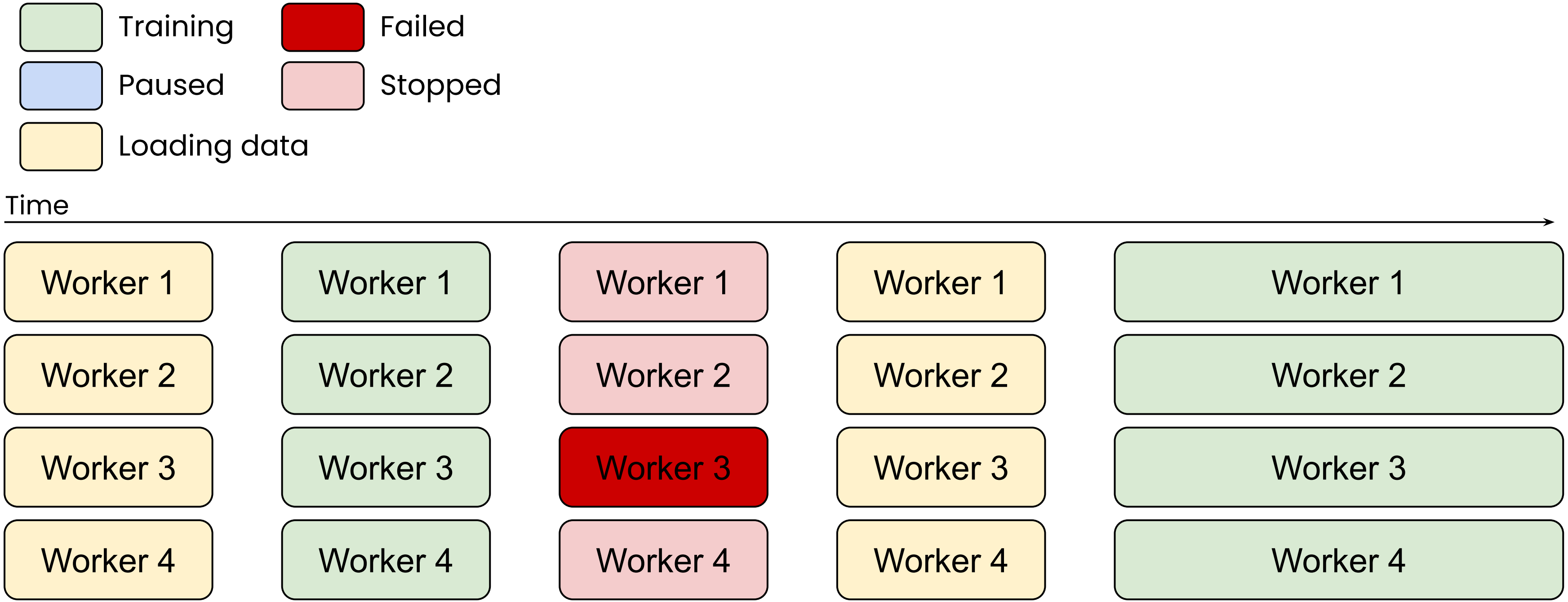
	Node 1	Node 2	Node 3	Node 4
Distributed dataframe (e.g. Modin)	Partition A Partition B	Partition C	Partition D Partition E Partition F	Partition G Partition H

	Worker 1	Worker 2	Worker 3	Worker 4
XGBoost-Ray workers	Partition A Partition B	Partition C Partition F	Partition D Partition E	Partition G Partition H

Fault tolerance strategies

- In distributed training, some worker nodes are bound to fail eventually
- **Default**: Simple (cold) restart from last checkpoint
- **Non-elastic** training (warm restart):
  Only failing worker restarts
- **Elastic training:** Continue training with fewer workers until failed actor is back

Fault tolerance: Simple (cold) restart



🟩 Training

🟦 Paused

🟨 Loading data

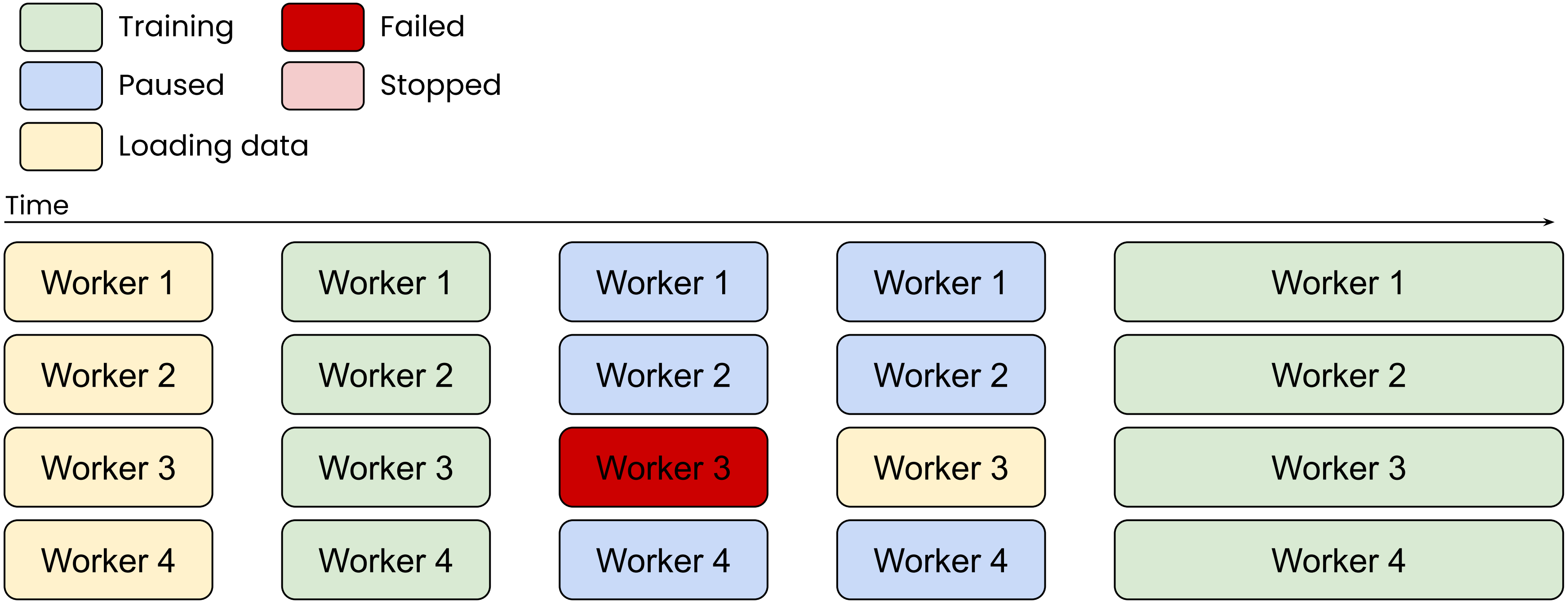
🟥 Failed

▨ Stopped

Time →

🟨 Worker 1	🟩 Worker 1	▨ Worker 1	🟨 Worker 1	🟩 Worker 1
🟨 Worker 2	🟩 Worker 2	▨ Worker 2	🟨 Worker 2	🟩 Worker 2
🟨 Worker 3	🟩 Worker 3	🟥 Worker 3	🟨 Worker 3	🟩 Worker 3
🟨 Worker 4	🟩 Worker 4	▨ Worker 4	🟨 Worker 4	🟩 Worker 4

Fault tolerance: Non-elastic training (warm restart)

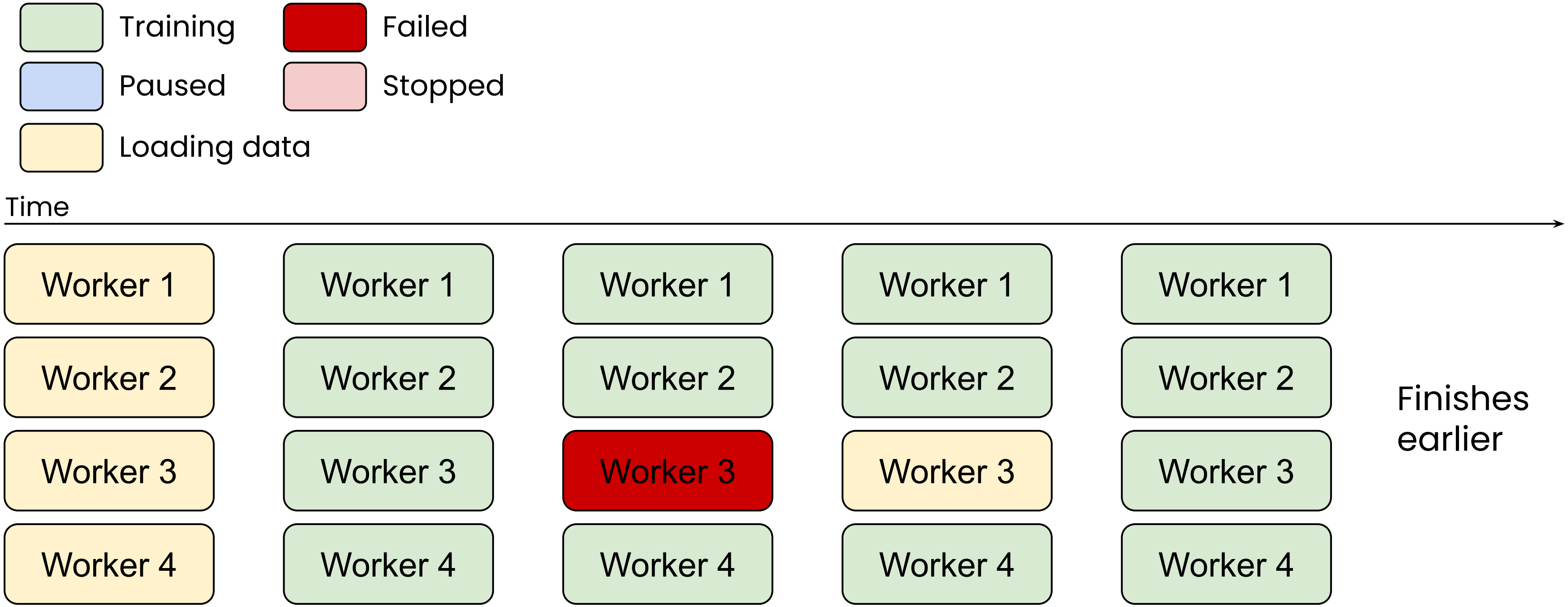


🟩 Training
🟥 Failed
🟦 Paused
▨ Stopped
🟨 Loading data

Time →

🟨 Worker 1	🟩 Worker 1	🟦 Worker 1	🟦 Worker 1	🟩 Worker 1
🟨 Worker 2	🟩 Worker 2	🟦 Worker 2	🟦 Worker 2	🟩 Worker 2
🟨 Worker 3	🟩 Worker 3	🟥 Worker 3	🟨 Worker 3	🟩 Worker 3
🟨 Worker 4	🟩 Worker 4	🟦 Worker 4	🟦 Worker 4	🟩 Worker 4

Fault tolerance: Elastic training



🟩 Training

🟥 Failed

🟦 Paused

▨ Stopped

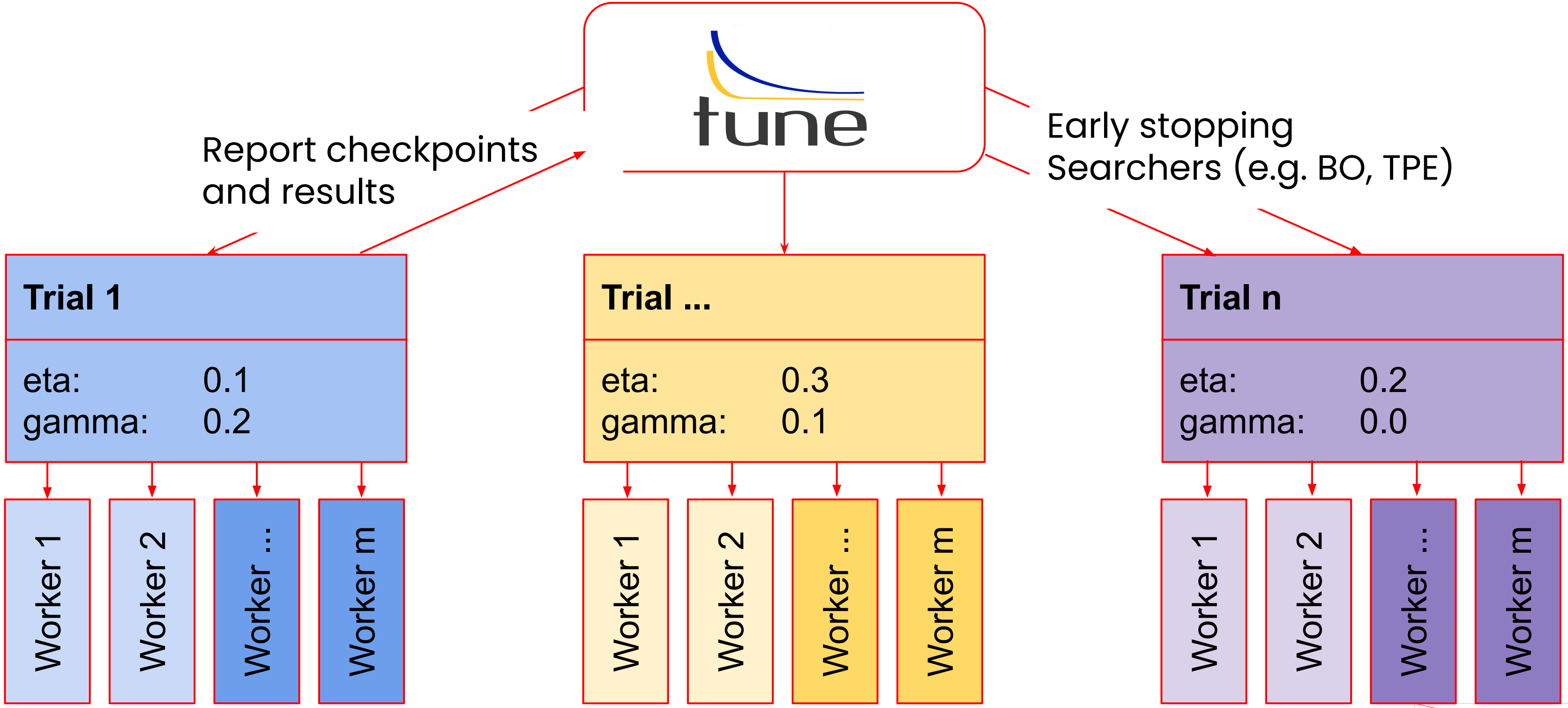
🟨 Loading data

Time →

🟨 Worker 1	🟩 Worker 1	🟩 Worker 1	🟩 Worker 1	🟩 Worker 1
🟨 Worker 2	🟩 Worker 2	🟩 Worker 2	🟩 Worker 2	🟩 Worker 2
🟨 Worker 3	🟩 Worker 3	🟥 Worker 3	🟨 Worker 3	🟩 Worker 3
🟨 Worker 4	🟩 Worker 4	🟩 Worker 4	🟩 Worker 4	🟩 Worker 4

Finishes earlier

Hyperparameter tuning



Simple API example

```
from sklearn.datasets import load_breast_cancer
from xgboost_ray import RayDMatrix, RayParams, train

train_x, train_y = load_breast_cancer(return_X_y=True)
train_set = RayDMatrix(train_x, train_y)

bst = train(
   {"objective": "binary:logistic"},
   train_set,
   ray_params=RayParams(num_actors=2)
)
bst.save_model("trained.xgb")
```

Takeaways

- Distributed computing is a necessity & norm
- Ray's vision: make distributed programming simple
  - Don't have to be distributed systems expert. Just use @ray.remote :)
- Scale your ML workloads with Ray Libraries

MARCH 29 - VIRTUAL - FREE

Production RL Summit

A reinforcement learning event for practitioners

Register: https://tinyurl.com/mr9rd32h

ORGANIZED BY anyscale

Sergey Levine

Ben Kasper

Sumitra Ganesh

J.P.Morgan

Adam Kelloway

DOW

Marc Weber

SIEMENS

Volkmar Sterzing

SIEMENS

MARCH 29 - VIRTUAL

Production RL Summit

A reinforcement learning event for practitioners

Register: https://tinyurl.com/mr9rd32h

~~$75~~ **$30**
Use code **DCRL2022**

ORGANIZED BY anyscale

HANDS-ON TUTORIAL

Contextual Bandits & RL with RLlib

Learn how to apply cutting edge RL in production with RLlib.

Tutorial covers:

- Brief overview of RL concepts.
- Train and tune contextual bandits and SlateQ algorithm
- Offline RL using cutting-edge algos
- Deploy RL models into a live service

~~$75~~ **$30** (use code **DCRL2022**)

Instructor:

Sven Mika, Lead maintainer, RLlib

AUGUST 23 & 24TH | SAN FRANCISCO

DON'T WAIT!

CFP closes
April 11th

Start learning Ray and contributing …

Getting Started: `pip install ray`

Documentation (docs.ray.io)
Quick start example, reference guides, etc

Join Ray Meetup
Revived in Jan 2022. Next meetup March 2nd.
Meetup each month and publish recording to the members
https://www.meetup.com/Bay-Area-Ray-Meetup/

Forums (discuss.ray.io)
Learn / share with broader Ray community, including core team

Ray Slack
Connect with the Ray team and community

Social Media (@raydistrtibuted, @anyscalecompute)
Follow us on Twitter and linkedIn

GitHub
Check out sources, file an issue, become a contributor, give us a ***Star*** *:)*
https://github.com/ray-project/ray

Thank you!

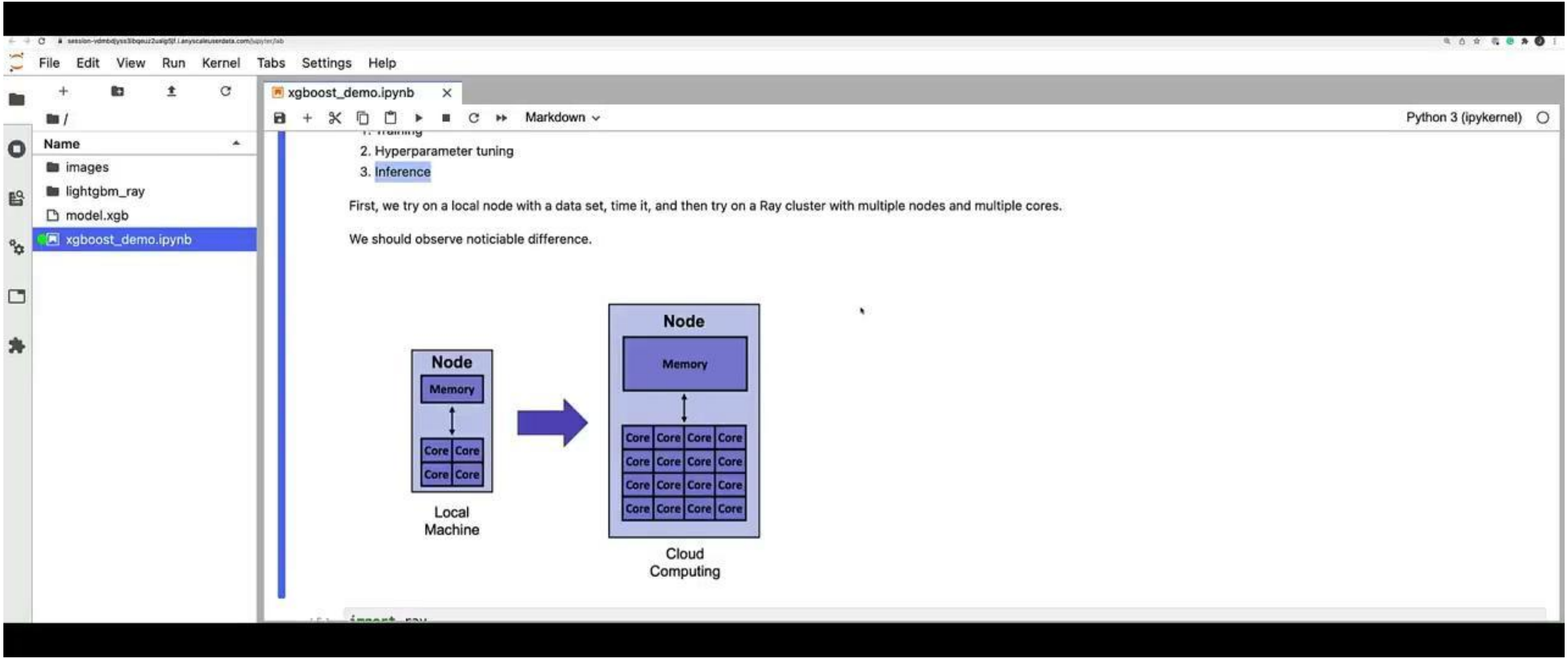
Let's stay in touch:

jules@anyscale.com
https://www.linkedin.com/in/dmatrix/

@2twitme

VIDEO

anyscale



1. Training
2. Hyperparameter tuning
3. Inference

First, we try on a local node with a data set, time it, and then try on a Ray cluster with multiple nodes and multiple cores.

We should observe noticiable difference.

```
import ray
```

Making Humans & Code GPU-Capable

Data Council Austin 2022

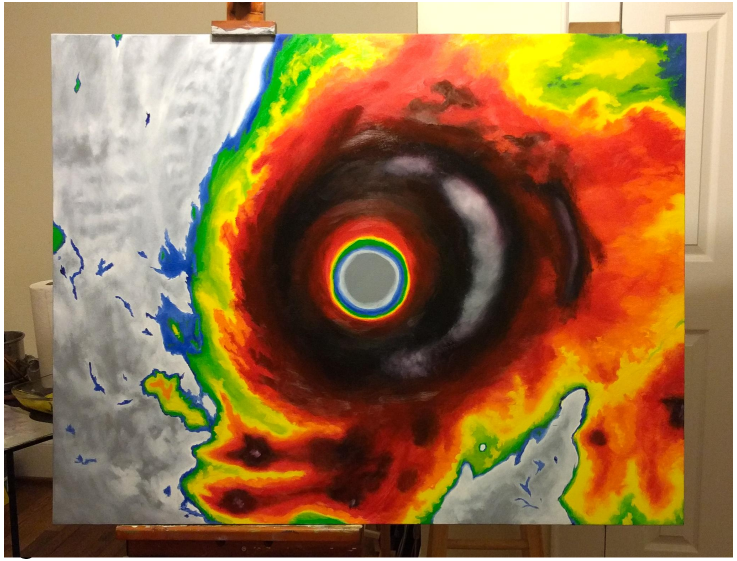
Emily May Curtin

Senior ML Ops Engineer, Mailchimp/Intuit

@emilymaycurtin

Howdy, I'm Emily

- 👽 ATLien (don't call it Hotlanta)
- ❌ #NotADataScientist
- 🎨 Oil painter by passion
- 💾 MLOps by day job (btw we're hiring!)
- ❤️ Big fan of Ryan Curtin



Our Goal:

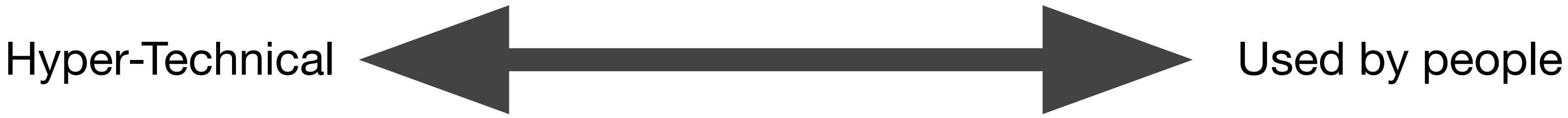
Help Data Scientists produce higher quality work faster

MLOps

is a hyper-technical field that is

all about *people*

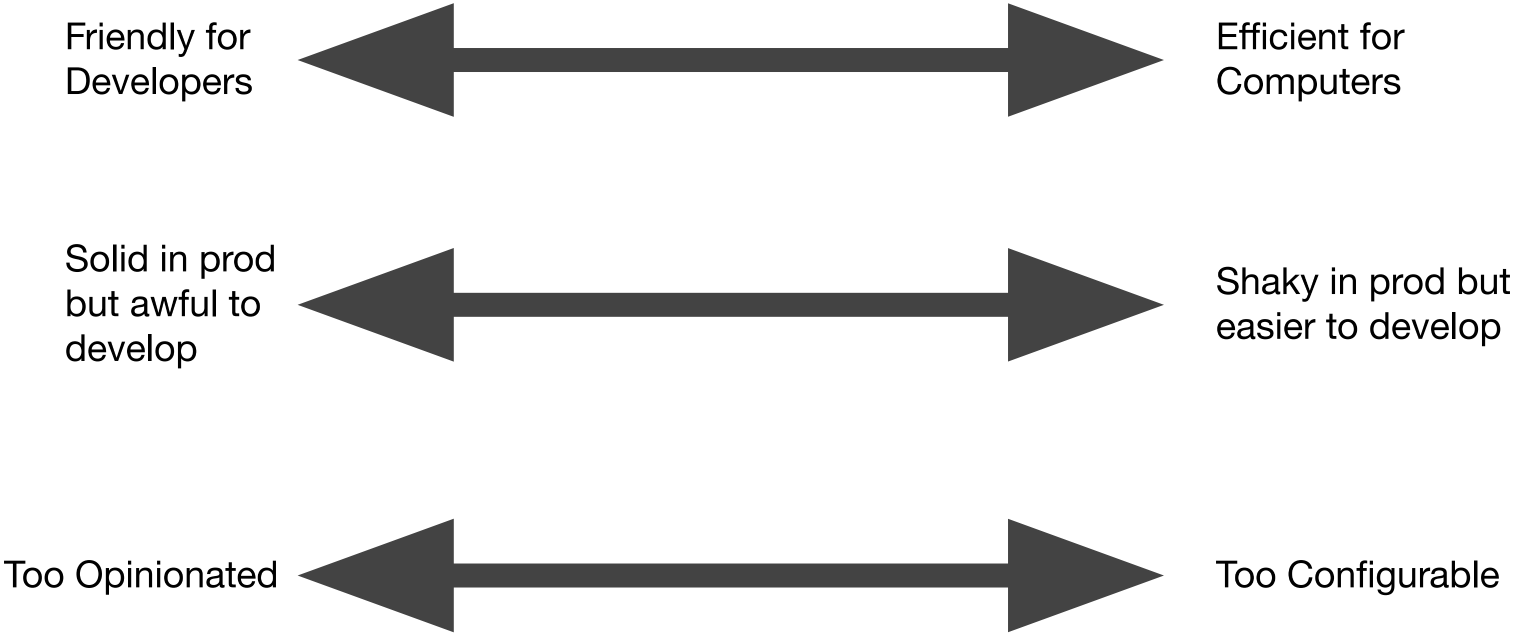
Inherent Design Tradeoff



Hyper-Technical

Used by people

Other Design Tradeoffs



Friendly for Developers

Efficient for Computers

Solid in prod but awful to develop

Shaky in prod but easier to develop

Too Opinionated

Too Configurable

Let's talk about ML stacks

2022

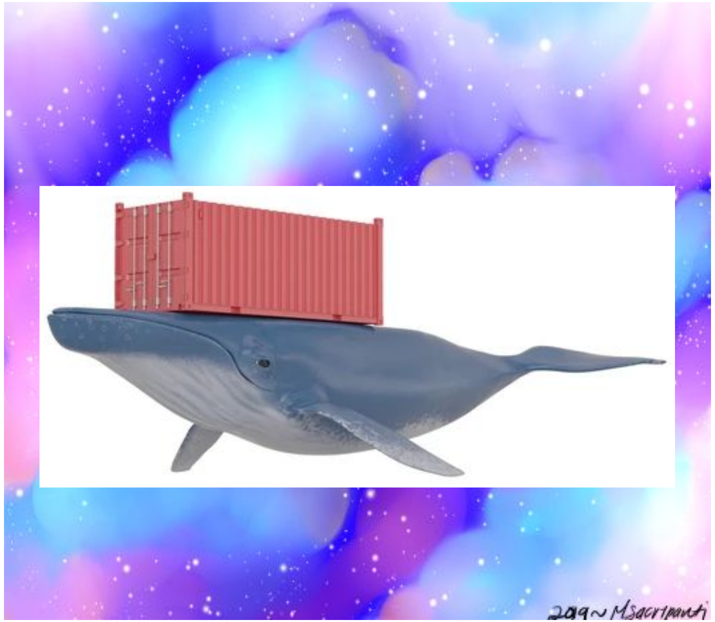
Typical ML Tech Stack

- Python
- Pytorch, HuggingFace, Tensorflow
- Docker
- Cloud infrastructure (we happen to use GCP)
- Kubernetes either directly or indirectly

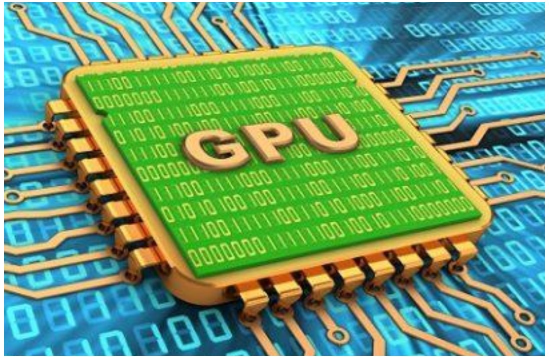
Benefits

- Good scalability, reproducibility
- Cloud infra good for spiky ML workloads (vs. more consistent, predictable web service)

But...



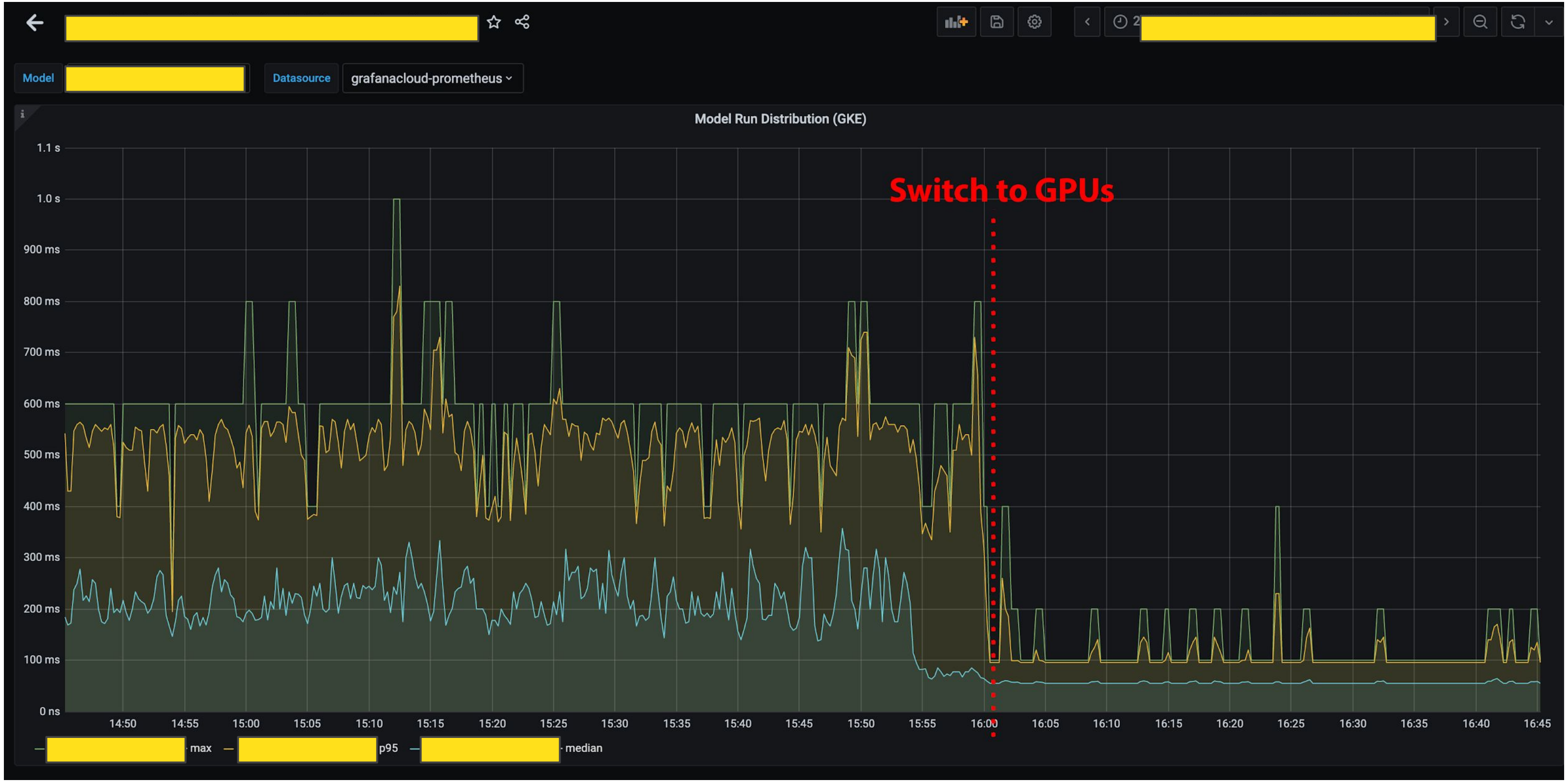
?????????



Let’s talk about GPUs

GPU
GPU
GPU
ANIMATED LASERS
GEFORCE
GEFORCE
HDMI
VGA
DVI

GPUs Can Be *Really* Awesome



[redacted]

Model [redacted] Datasource grafanacloud-prometheus

Model Run Distribution (GKE)

Switch to GPUs

[redacted] max [redacted] p95 [redacted] median

GPUs . . .

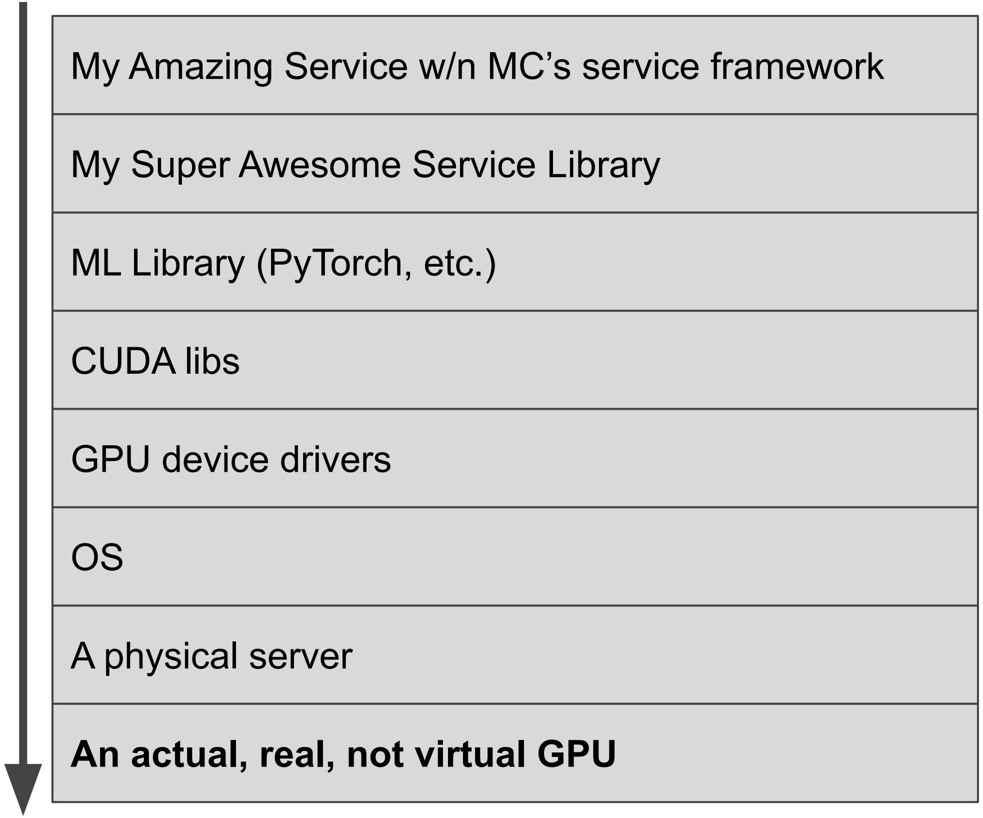
- Are optional hardware peripherals
- Require special drivers
- Rely on system buses for I/O

GPUs . . . Are Printers

GPUs . . . Are Printers

That are very good at linear algebra

Call Stack on a plain server



My Amazing Service w/n MC’s service framework

My Super Awesome Service Library

ML Library (PyTorch, etc.)

CUDA libs

GPU device drivers

OS

A physical server

An actual, real, not virtual GPU

Call Stack on a plain server

My Amazing Service w/n MC’s service framework

My Super Awesome Service Library

ML Library (PyTorch, Tensorflow, XGBoost, etc.)

CUDA libs

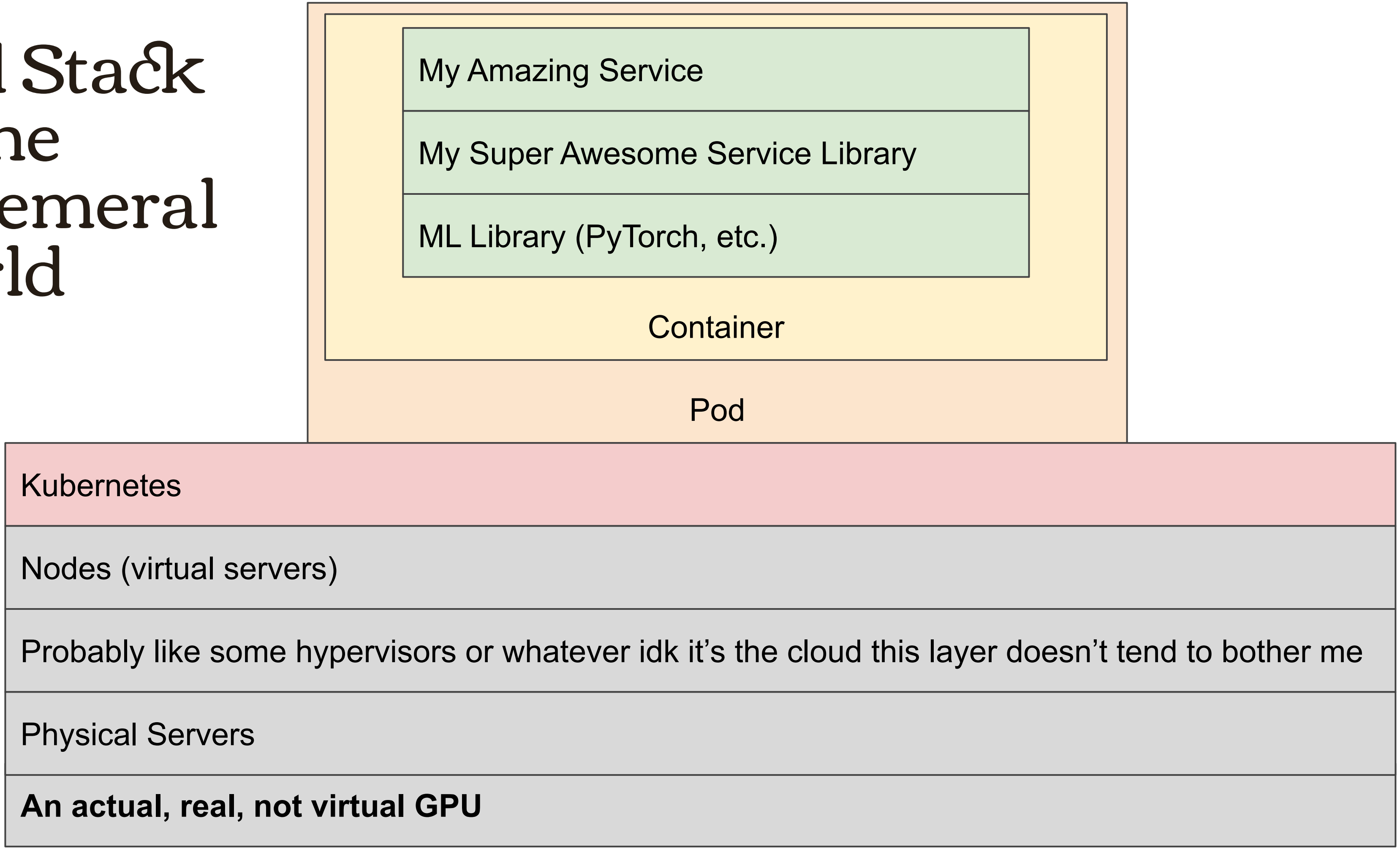
GPU device drivers

OS

A physical server

An actual, real, not virtual

Call Stack in the ephemeral world



My Amazing Service

My Super Awesome Service Library

ML Library (PyTorch, etc.)

Container

Pod

Kubernetes

Nodes (virtual servers)

Probably like some hypervisors or whatever idk it’s the cloud this layer doesn’t tend to bother me

Physical Servers

An actual, real, not virtual GPU

What you need to talk to a GPU

- GPU
- Drivers
    - `nvidia.ko` - Kernel mode GPU driver
    - `libcuda.so` - User mode GPU driver (aka low-level API)
- CUDA Toolkit
    - `libcudart.so` - Runtime API (aka high-level API)
    - `cuBLAS`, `cuRAND`, `cuSOLVER`, and other toolkit libs

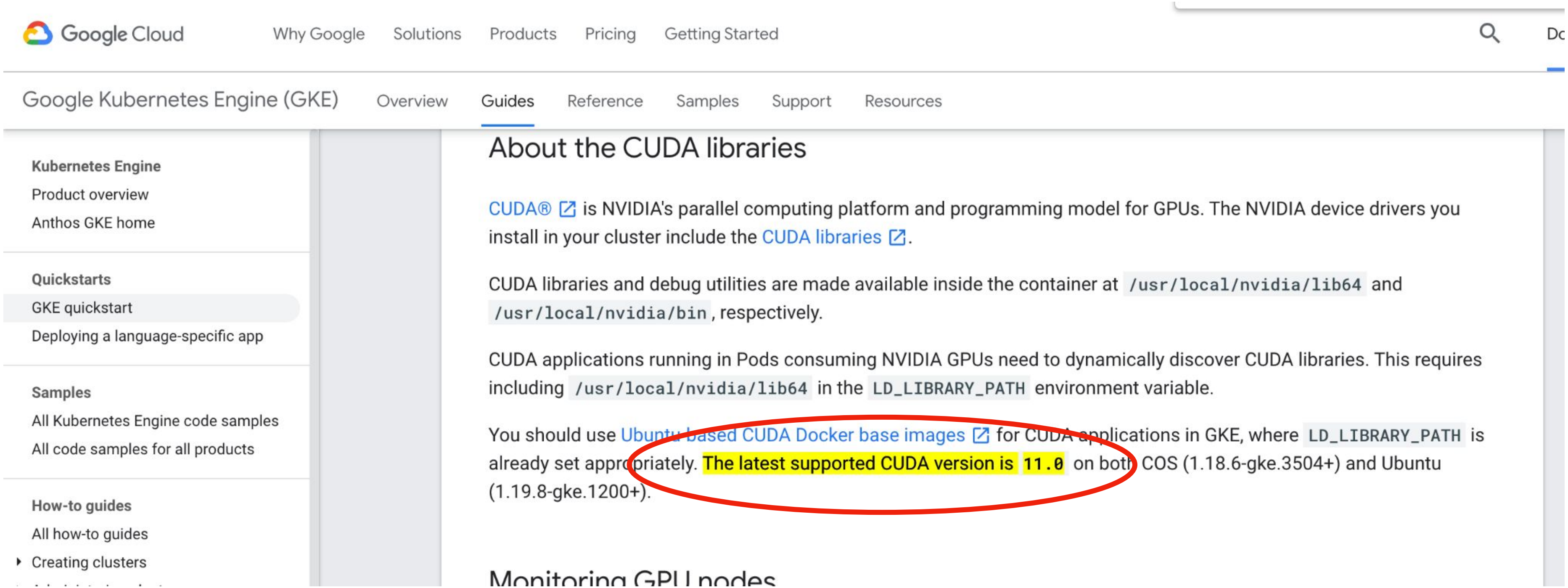
GPUs and Device Drivers

2022

These come from your k8s service provider, GKE in my case

GKE Provides

- Configurable GPUs and GPU pools
- DaemonSet for device drivers



About the CUDA libraries

CUDA® is NVIDIA's parallel computing platform and programming model for GPUs. The NVIDIA device drivers you install in your cluster include the CUDA libraries.

CUDA libraries and debug utilities are made available inside the container at `/usr/local/nvidia/lib64` and `/usr/local/nvidia/bin`, respectively.

CUDA applications running in Pods consuming NVIDIA GPUs need to dynamically discover CUDA libraries. This requires including `/usr/local/nvidia/lib64` in the `LD_LIBRARY_PATH` environment variable.

You should use Ubuntu-based CUDA Docker base images for CUDA applications in GKE, where `LD_LIBRARY_PATH` is already set appropriately. The latest supported CUDA version is `11.0` on both COS (1.18.6-gke.3504+) and Ubuntu (1.19.8-gke.1200+).

Monitoring GPU nodes

Various CUDA APIs and other libs

- Some Python ML Libs ship with binaries in the wheels
    - Dependent on Python package manager (pip, anaconda, etc)
    - Usually does not include `libcuda.so`
- Might be made available via your device driver Daemonset
    - Set `LD_LIBRARY_PATH` to access
    - Usually only API binaries, not other toolkit libs
- Might have to DIY via base container or custom install step
- Might have to combine all of the above

Matching CUDA Versions Matters

- CUDA version supported by your ML library of choice
- CUDA version in your base docker image
- CUDA version available on your k8s nodes, exposed through Daemonset

Matching CUDA Versions Matters*

Matching CUDA Versions Matters*

*Sometimes. Depending. Maybe not.

Matching CUDA Versions Matters*

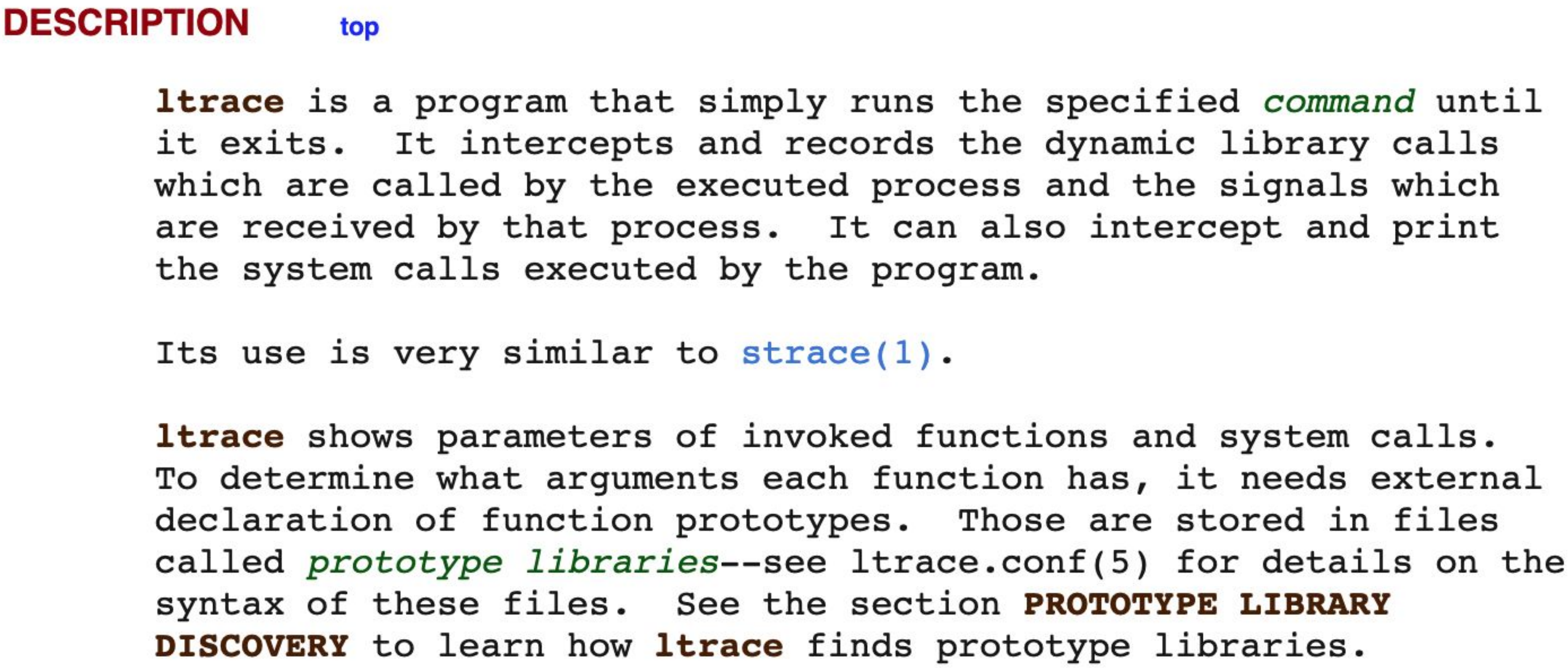
*Sometimes. Depending. Maybe not.

YMMV depending on your library

- PyTorch does a lot of stuff to support 10.x and 11.x
- Tensorflow is very picky about everything

CUDA has complex forward and backward compatibility scenarios

ltrace and strace rock

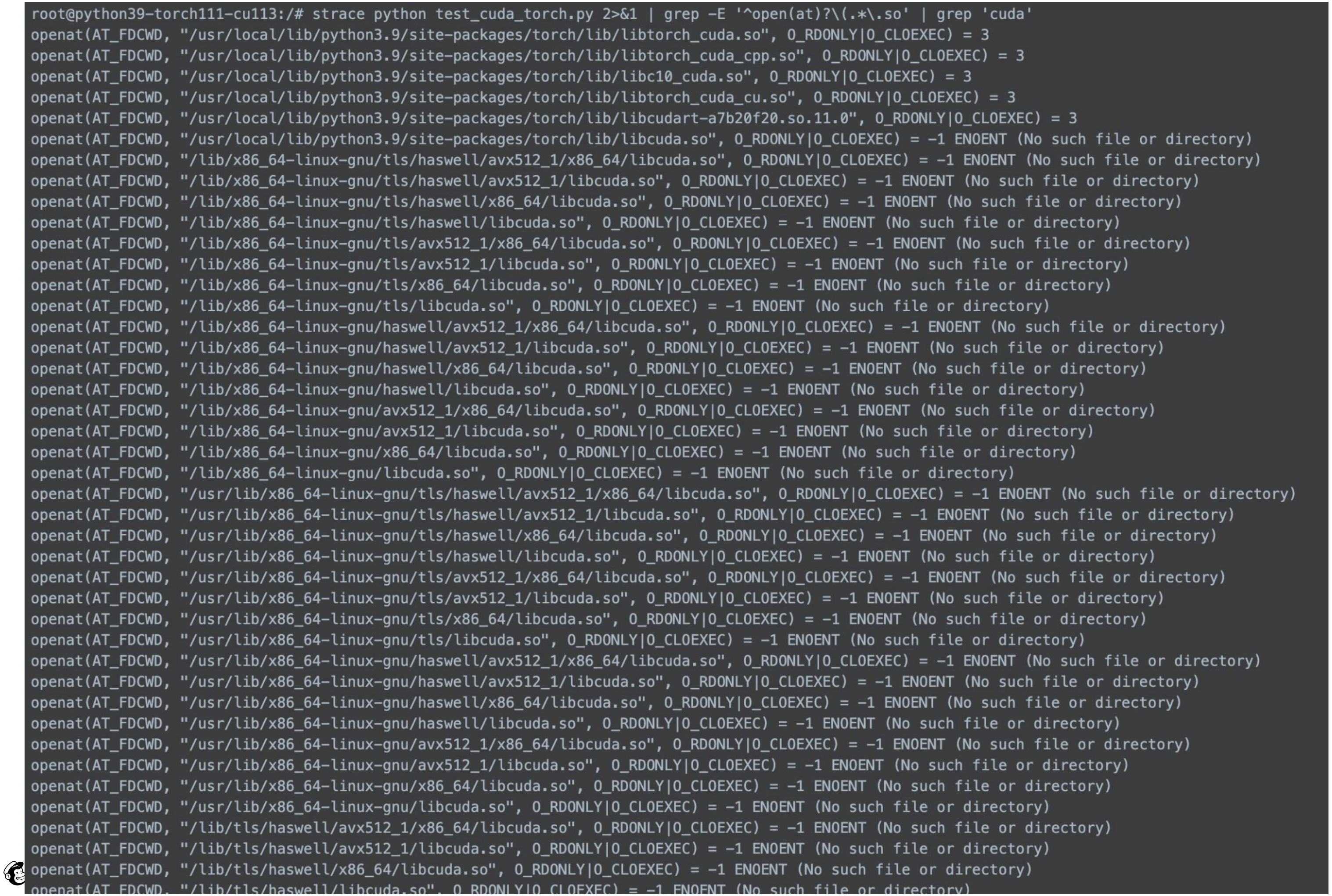


DESCRIPTION top

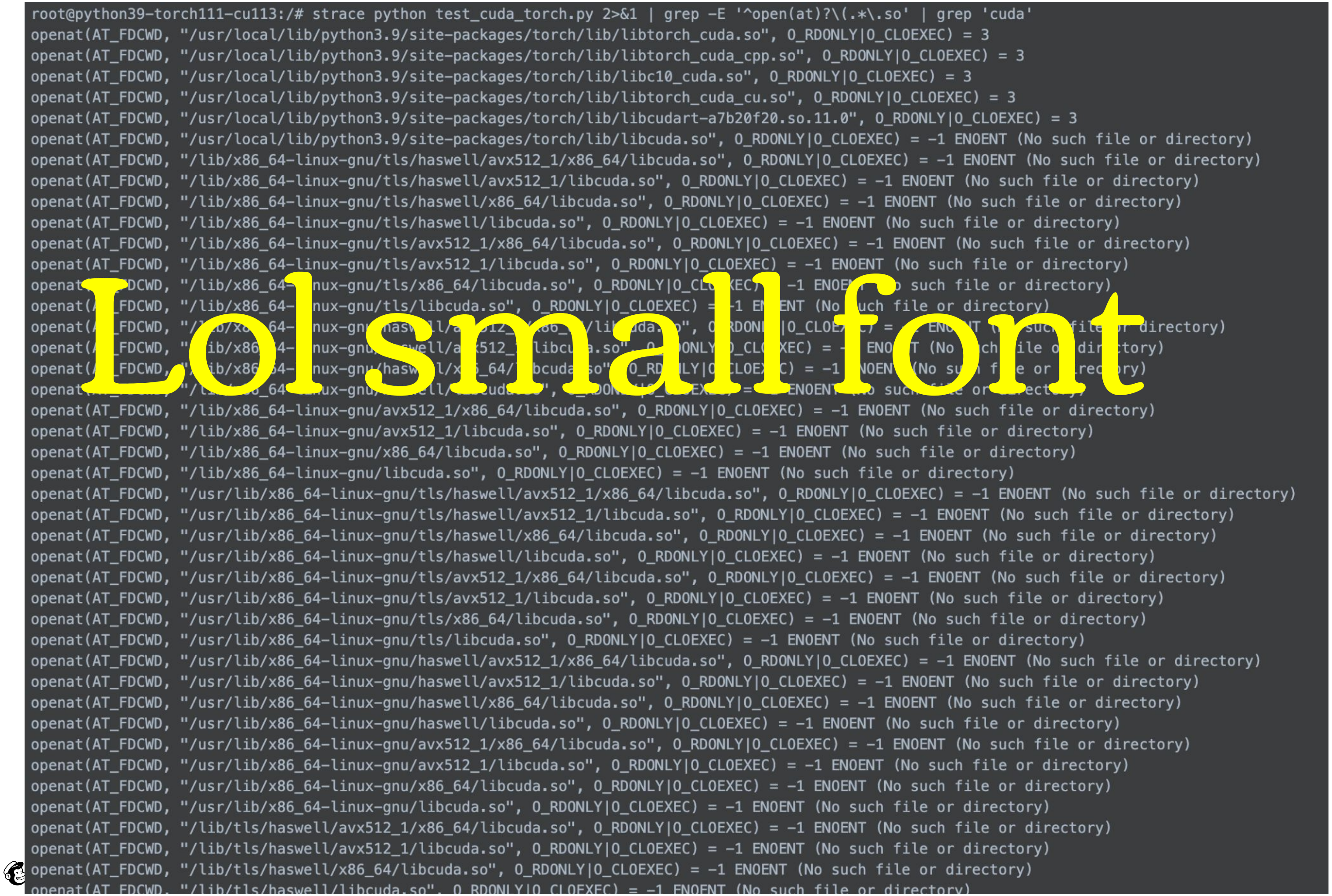
ltrace is a program that simply runs the specified *command* until it exits. It intercepts and records the dynamic library calls which are called by the executed process and the signals which are received by that process. It can also intercept and print the system calls executed by the program.

Its use is very similar to strace(1).

ltrace shows parameters of invoked functions and system calls. To determine what arguments each function has, it needs external declaration of function prototypes. Those are stored in files called *prototype libraries*--see ltrace.conf(5) for details on the syntax of these files. See the section **PROTOTYPE LIBRARY DISCOVERY** to learn how **ltrace** finds prototype libraries.

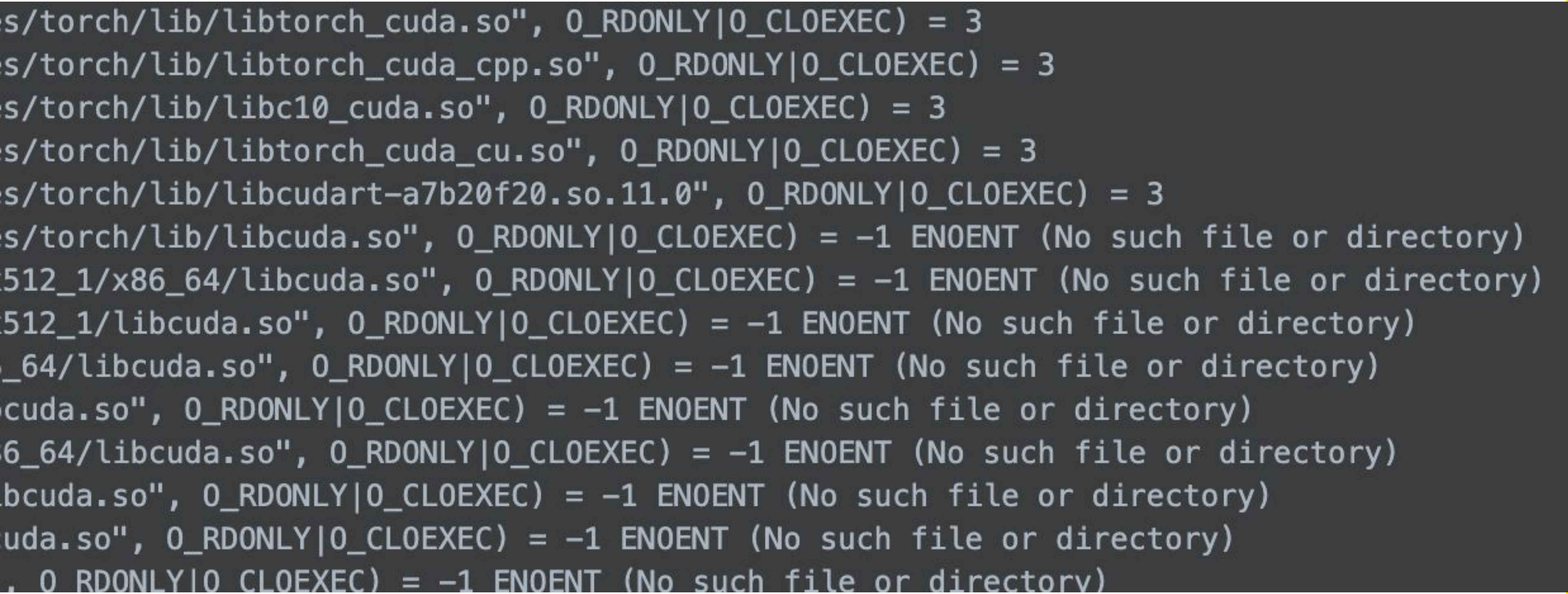


```
root@python39-torch111-cu113:/# strace python test_cuda_torch.py 2>&1 | grep -E '^open(at)?\(.*\.so' | grep 'cuda'
openat(AT_FDCWD, "/usr/local/lib/python3.9/site-packages/torch/lib/libtorch_cuda.so", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.9/site-packages/torch/lib/libtorch_cuda_cpp.so", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.9/site-packages/torch/lib/libc10_cuda.so", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.9/site-packages/torch/lib/libtorch_cuda_cu.so", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.9/site-packages/torch/lib/libcudart-a7b20f20.so.11.0", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.9/site-packages/torch/lib/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/tls/haswell/avx512_1/x86_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/tls/haswell/avx512_1/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/tls/haswell/x86_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/tls/haswell/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/tls/avx512_1/x86_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/tls/avx512_1/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/tls/x86_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/tls/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/haswell/avx512_1/x86_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/haswell/avx512_1/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/haswell/x86_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/haswell/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/avx512_1/x86_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/avx512_1/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/x86_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/lib/x86_64-linux-gnu/tls/haswell/avx512_1/x86_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/lib/x86_64-linux-gnu/tls/haswell/avx512_1/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/lib/x86_64-linux-gnu/tls/haswell/x86_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/lib/x86_64-linux-gnu/tls/haswell/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/lib/x86_64-linux-gnu/tls/avx512_1/x86_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/lib/x86_64-linux-gnu/tls/avx512_1/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/lib/x86_64-linux-gnu/tls/x86_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/lib/x86_64-linux-gnu/tls/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/lib/x86_64-linux-gnu/haswell/avx512_1/x86_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/lib/x86_64-linux-gnu/haswell/avx512_1/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/lib/x86_64-linux-gnu/haswell/x86_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/lib/x86_64-linux-gnu/haswell/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/lib/x86_64-linux-gnu/avx512_1/x86_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/lib/x86_64-linux-gnu/avx512_1/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/lib/x86_64-linux-gnu/x86_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/lib/x86_64-linux-gnu/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/tls/haswell/avx512_1/x86_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/tls/haswell/avx512_1/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/tls/haswell/x86_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/tls/haswell/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
```



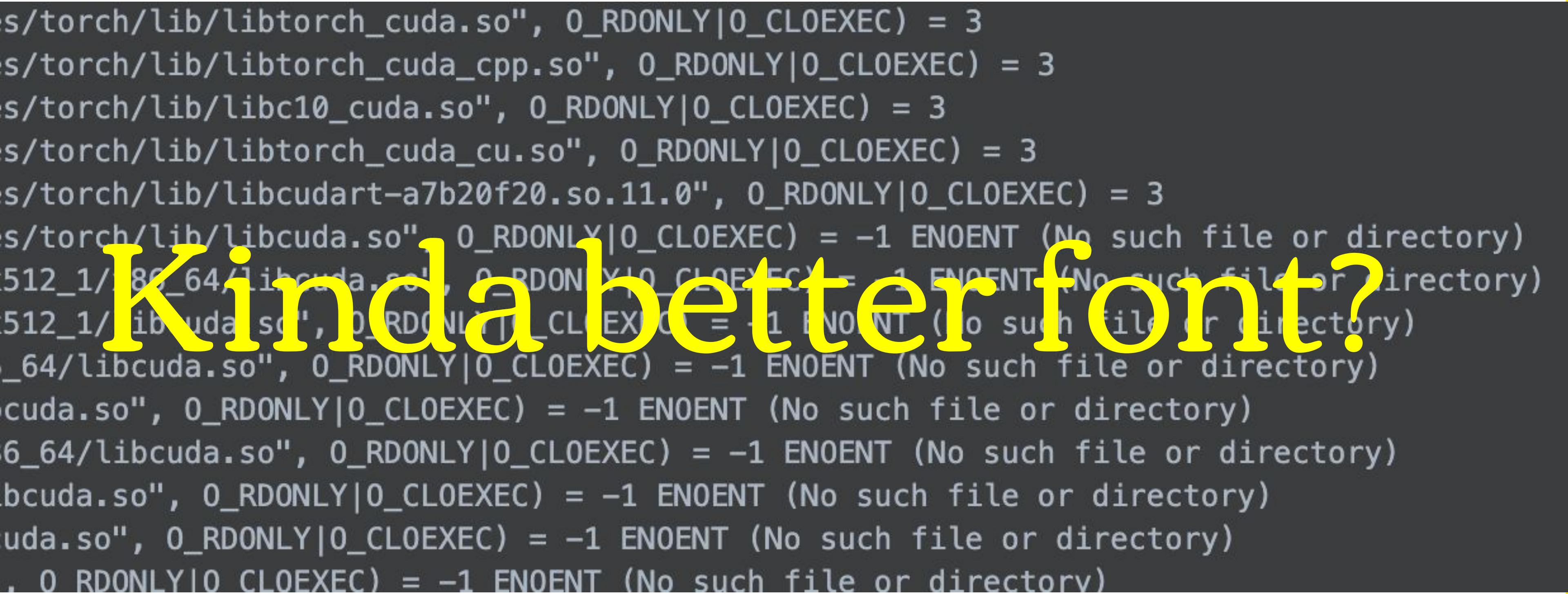
```
root@python39-torch111-cu113:/# strace python test_cuda_torch.py 2>&1 | grep -E '^open(at)?\(.*\.so' | grep 'cuda'
openat(AT_FDCWD, "/usr/local/lib/python3.9/site-packages/torch/lib/libtorch_cuda.so", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.9/site-packages/torch/lib/libtorch_cuda_cpp.so", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.9/site-packages/torch/lib/libc10_cuda.so", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.9/site-packages/torch/lib/libtorch_cuda_cu.so", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.9/site-packages/torch/lib/libcudart-a7b20f20.so.11.0", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.9/site-packages/torch/lib/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/tls/haswell/avx512_1/x86_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/tls/haswell/avx512_1/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/tls/haswell/x86_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/tls/haswell/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/tls/avx512_1/x86_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/tls/avx512_1/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/tls/x86_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/tls/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/haswell/avx512_1/x86_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/haswell/avx512_1/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/haswell/x86_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/haswell/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/avx512_1/x86_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/avx512_1/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/x86_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/lib/x86_64-linux-gnu/tls/haswell/avx512_1/x86_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/lib/x86_64-linux-gnu/tls/haswell/avx512_1/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/lib/x86_64-linux-gnu/tls/haswell/x86_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/lib/x86_64-linux-gnu/tls/haswell/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/lib/x86_64-linux-gnu/tls/avx512_1/x86_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/lib/x86_64-linux-gnu/tls/avx512_1/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/lib/x86_64-linux-gnu/tls/x86_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/lib/x86_64-linux-gnu/tls/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/lib/x86_64-linux-gnu/haswell/avx512_1/x86_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/lib/x86_64-linux-gnu/haswell/avx512_1/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/lib/x86_64-linux-gnu/haswell/x86_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/lib/x86_64-linux-gnu/haswell/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/lib/x86_64-linux-gnu/avx512_1/x86_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/lib/x86_64-linux-gnu/avx512_1/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/lib/x86_64-linux-gnu/x86_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/lib/x86_64-linux-gnu/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/tls/haswell/avx512_1/x86_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/tls/haswell/avx512_1/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/tls/haswell/x86_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/tls/haswell/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
```

Lol small font

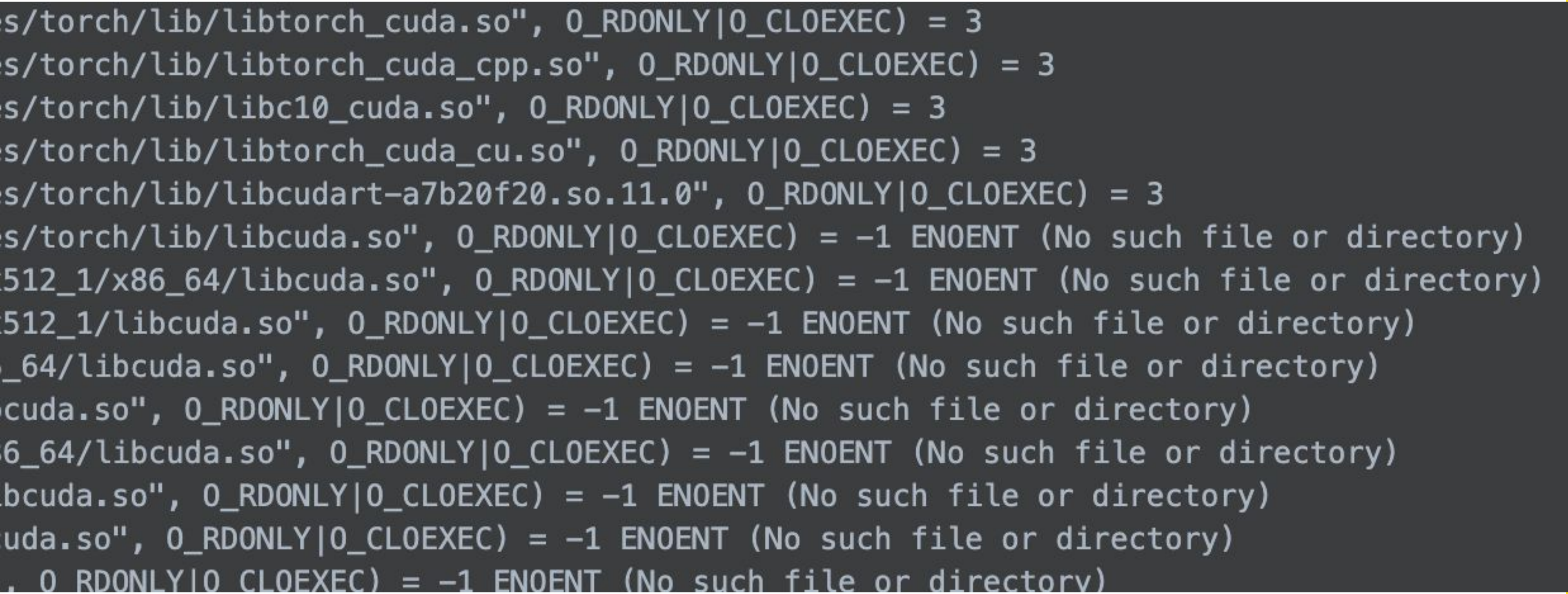


```
s/torch/lib/libtorch_cuda.so", O_RDONLY|O_CLOEXEC) = 3
s/torch/lib/libtorch_cuda_cpp.so", O_RDONLY|O_CLOEXEC) = 3
s/torch/lib/libc10_cuda.so", O_RDONLY|O_CLOEXEC) = 3
s/torch/lib/libtorch_cuda_cu.so", O_RDONLY|O_CLOEXEC) = 3
s/torch/lib/libcudart-a7b20f20.so.11.0", O_RDONLY|O_CLOEXEC) = 3
s/torch/lib/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
512_1/x86_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
512_1/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
cuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
6_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
bcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
uda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
, O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
```

Kinda better font?



```
s/torch/lib/libtorch_cuda.so", O_RDONLY|O_CLOEXEC) = 3
s/torch/lib/libtorch_cuda_cpp.so", O_RDONLY|O_CLOEXEC) = 3
s/torch/lib/libc10_cuda.so", O_RDONLY|O_CLOEXEC) = 3
s/torch/lib/libtorch_cuda_cu.so", O_RDONLY|O_CLOEXEC) = 3
s/torch/lib/libcudart-a7b20f20.so.11.0", O_RDONLY|O_CLOEXEC) = 3
s/torch/lib/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
512_1/x86_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
512_1/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
cuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
6_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
bcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
uda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
, O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
```

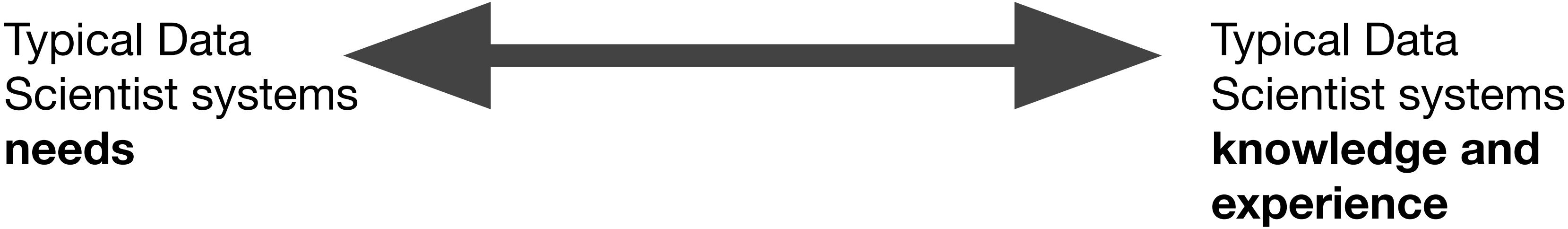


```
s/torch/lib/libtorch_cuda.so", O_RDONLY|O_CLOEXEC) = 3
s/torch/lib/libtorch_cuda_cpp.so", O_RDONLY|O_CLOEXEC) = 3
s/torch/lib/libc10_cuda.so", O_RDONLY|O_CLOEXEC) = 3
s/torch/lib/libtorch_cuda_cu.so", O_RDONLY|O_CLOEXEC) = 3
s/torch/lib/libcudart-a7b20f20.so.11.0", O_RDONLY|O_CLOEXEC) = 3
s/torch/lib/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
512_1/x86_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
512_1/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
cuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
6_64/libcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
bcuda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
uda.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
, O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
```

MLOps

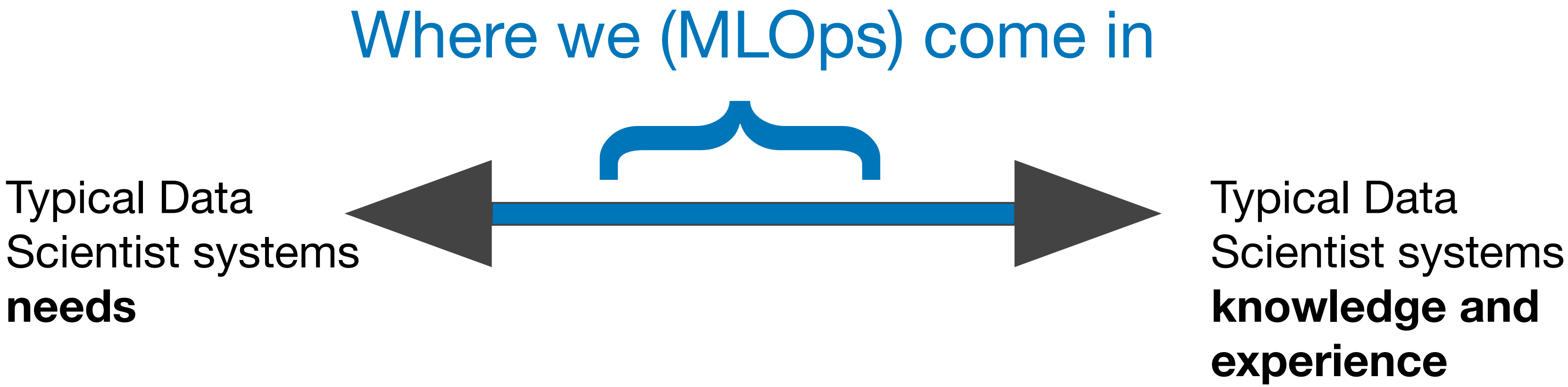
is a hyper-technical field that is

all about *people*



Typical Data Scientist systems **needs**

Typical Data Scientist systems **knowledge and experience**



Where we (MLOps) come in

Typical Data Scientist systems **needs**

Typical Data Scientist systems **knowledge and experience**

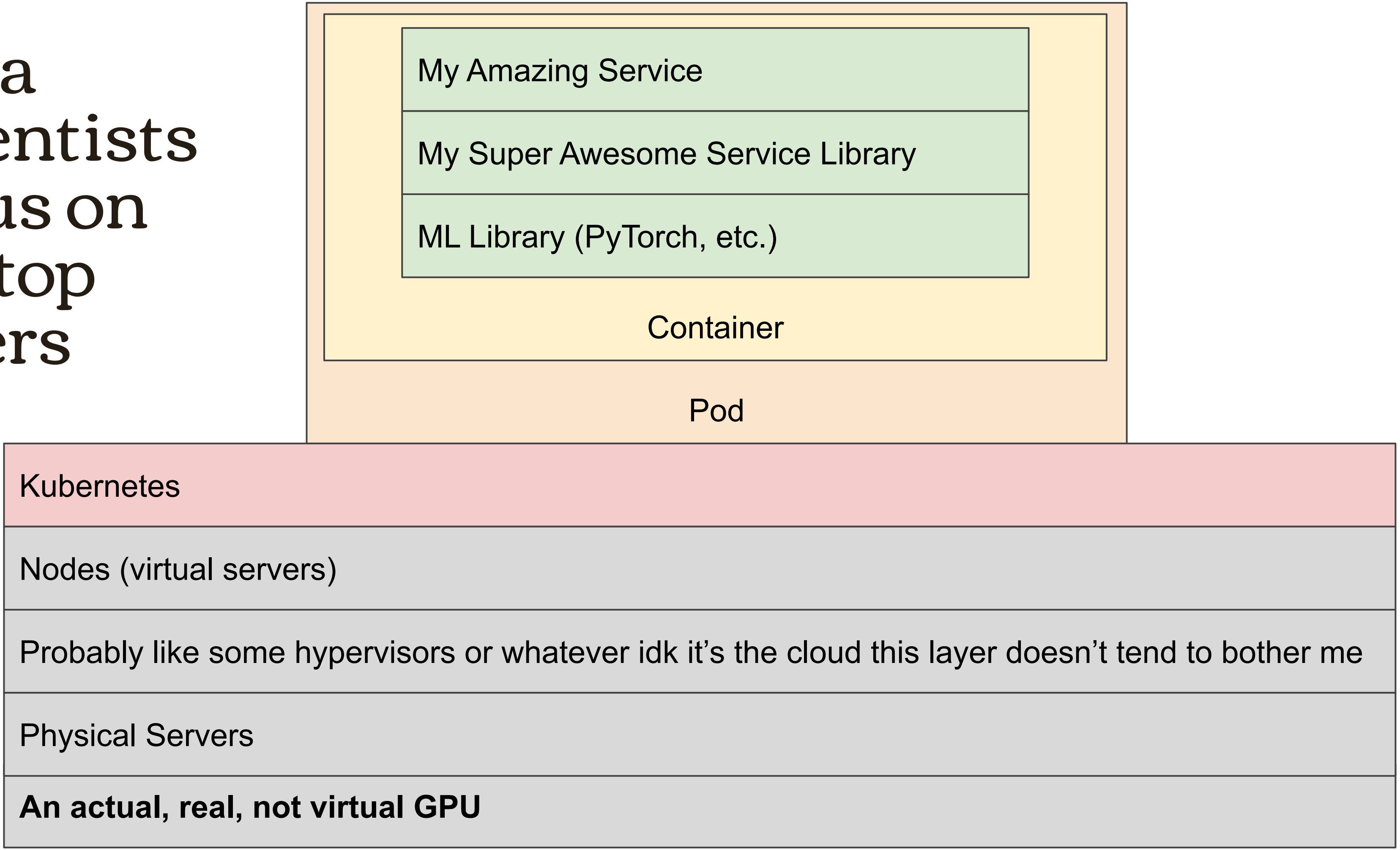
Systems Abstraction

Providing a good enough encapsulation of the system so Data Scientists can focus on the application layers.

It’s really hard.

Most MLOps systems are *full* of leaky abstractions.

Data Scientists focus on the top layers



My Amazing Service

My Super Awesome Service Library

ML Library (PyTorch, etc.)

Container

Pod

Kubernetes

Nodes (virtual servers)

Probably like some hypervisors or whatever idk it’s the cloud this layer doesn’t tend to bother me

Physical Servers

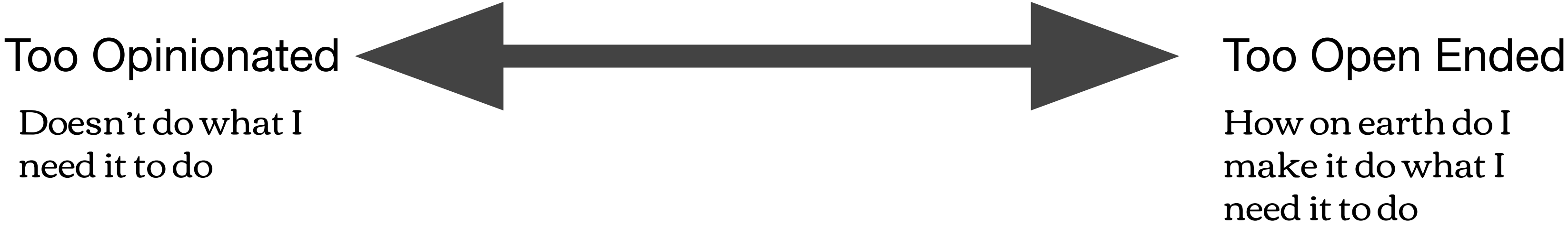
An actual, real, not virtual GPU

Design Tradeoffs

Too Opinionated

Too Open Ended

Design Tradeoffs



Too Opinionated

Doesn’t do what I need it to do

Too Open Ended

How on earth do I make it do what I need it to do

To enable high tech,
go low tech

GPUs for ML

… via repo templating

@lowcost_cosplay

Repo templating is not cool. And it works.

Repo Templating

- Provide a good enough, general enough base for the majority
- Includes
    - Base container to encapsulate the runtime environment
    - Places to integrate custom Python code
    - Basic run scripts for applications
    - Basic CI/CD stuff (ex: Jenkinsfile)
- GPU capability built in via base container(s)

Challenges

- Is your base container general enough? Will it match prod?
- Differences between libraries, batch jobs, live services, etc.
- How do children of a template get updates from the parent?
- How do we provide general GPU capability to everything using the template(s)?

Some Hard-Won Wisdom

- One template per project type (library, batch job, etc.) with shared base containers.

- Allow massive flexibility in ML lib choice within your language

- One base container is probably not good enough. Have curated options. (ex: tensorflow breaks everything)

- Design for the 90% cases, don’t generalize the other 10%

In Conclusion

- MLOps is a super technical role that’s **all about people**
- `strace` is your friend
- Repo templating is your friend
- Be uncool to do cool stuff

@emilymaycurtin

Thank you.

The Modern Stack for ML Infrastructure

Ville Tuulos

outerbounds

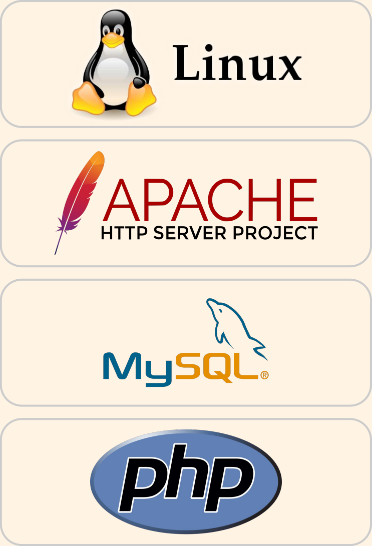
The modern stack?

🤔

The stack?

🤔

The Evolution of Web Stacks



LAMP (1998)

The Evolution of Web Stacks

LAMP (1998)

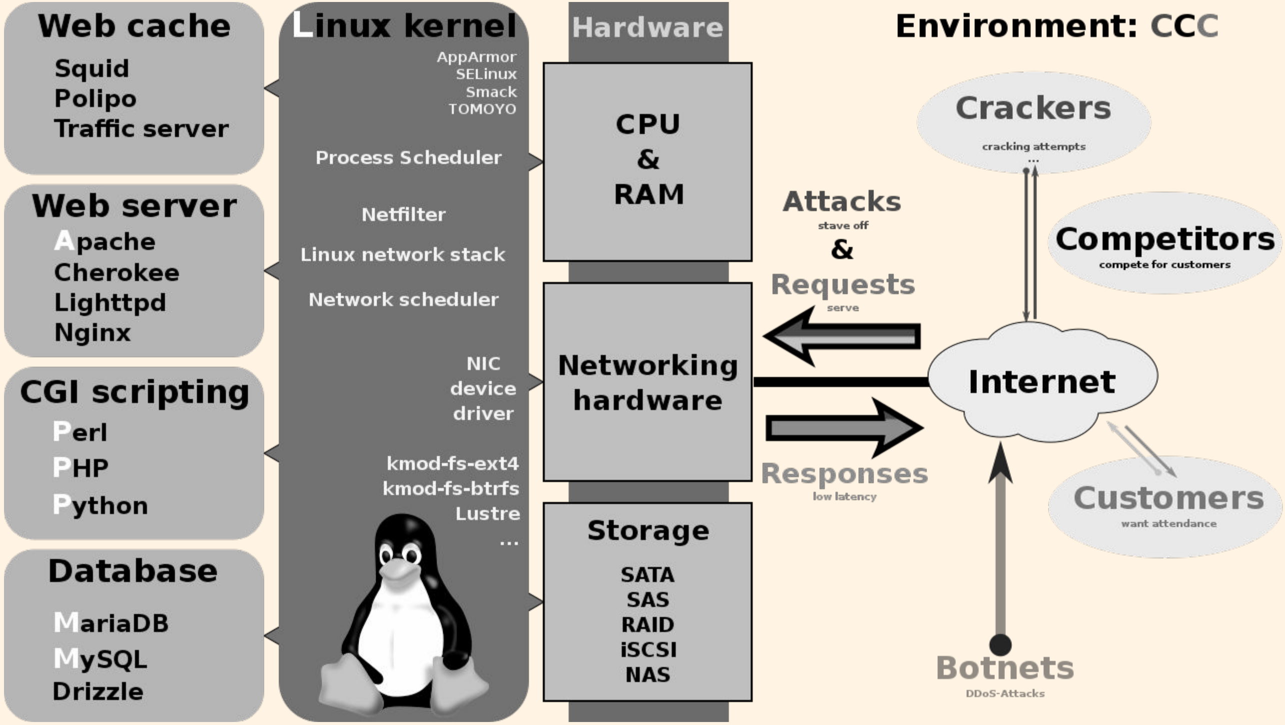
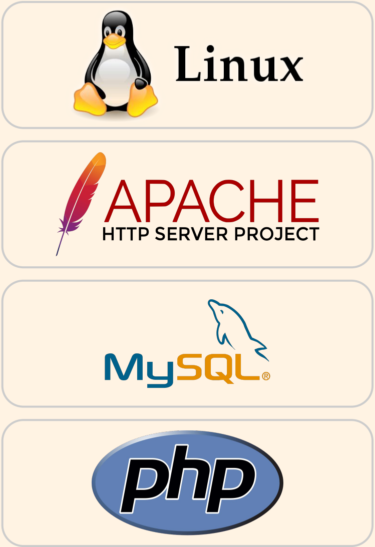
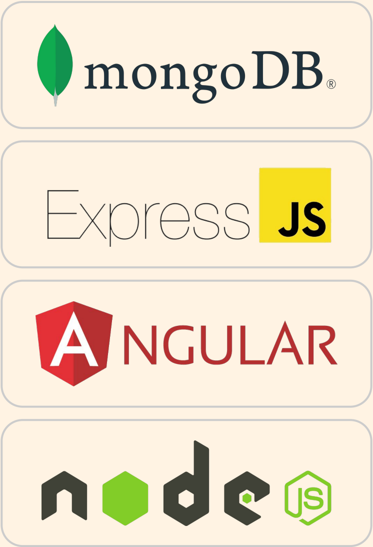


Figure by Shmuel Csaba Otto Traian / Wikipedia

The Evolution of Web Stacks

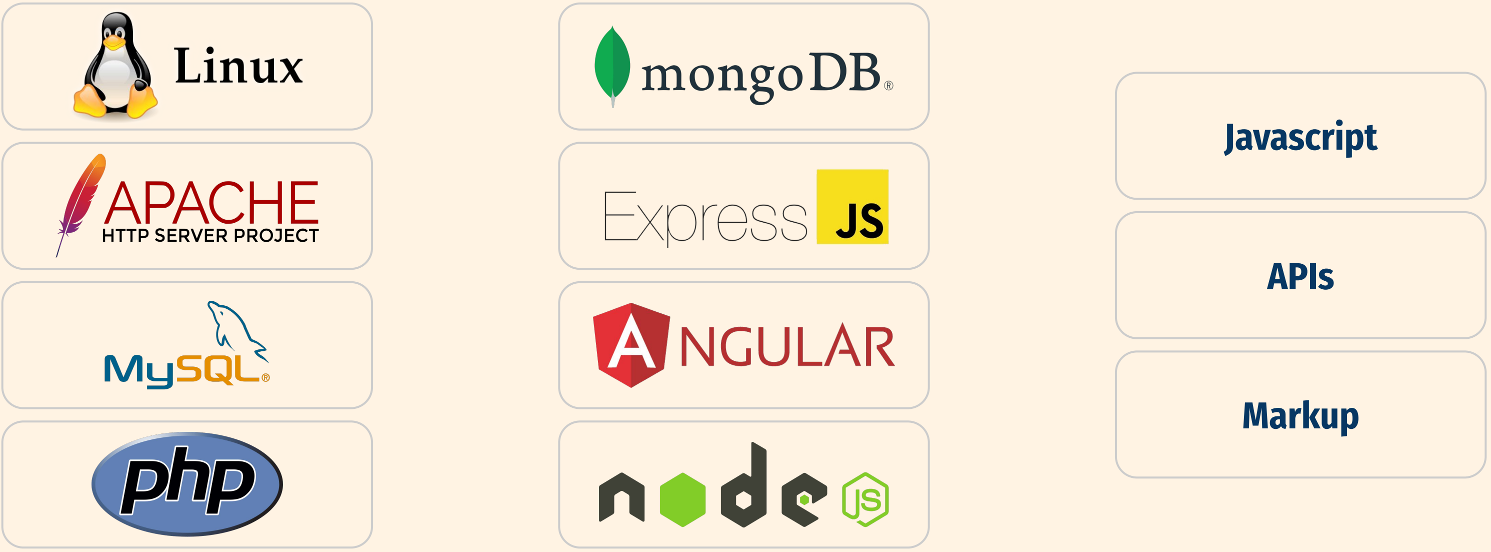


LAMP (1998)



MEAN (2013)

The Evolution of Web Stacks



Linux

APACHE HTTP SERVER PROJECT

MySQL

php

LAMP (1998)

mongoDB

Express JS

ANGULAR

node JS

MEAN (2013)

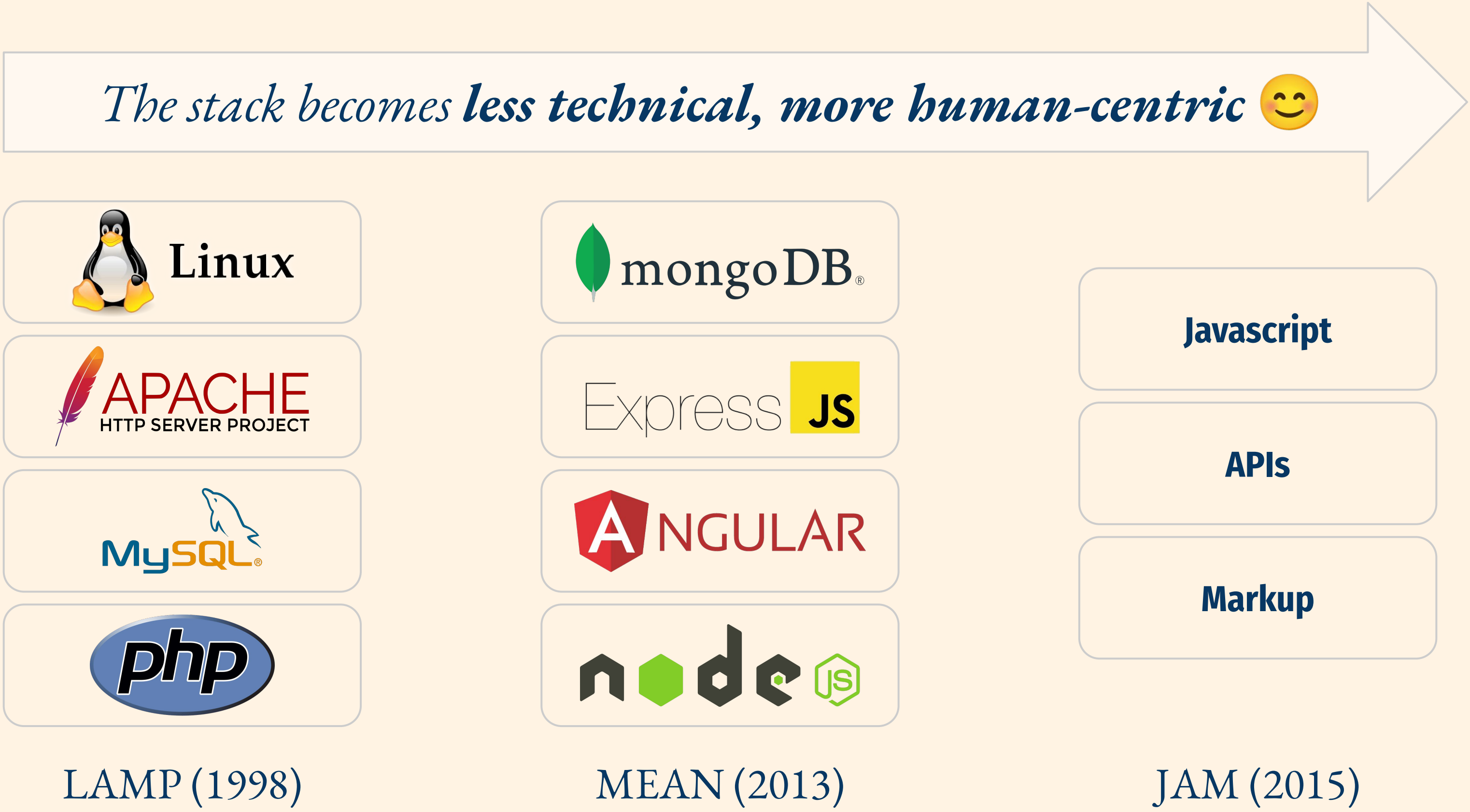
Javascript

APIs

Markup

JAM (2015)

The Evolution of Web Stacks



*The stack becomes **less technical, more human-centric*** 😊

Linux

APACHE HTTP SERVER PROJECT

MySQL

php

LAMP (1998)

mongoDB

Express JS

ANGULAR

node JS

MEAN (2013)

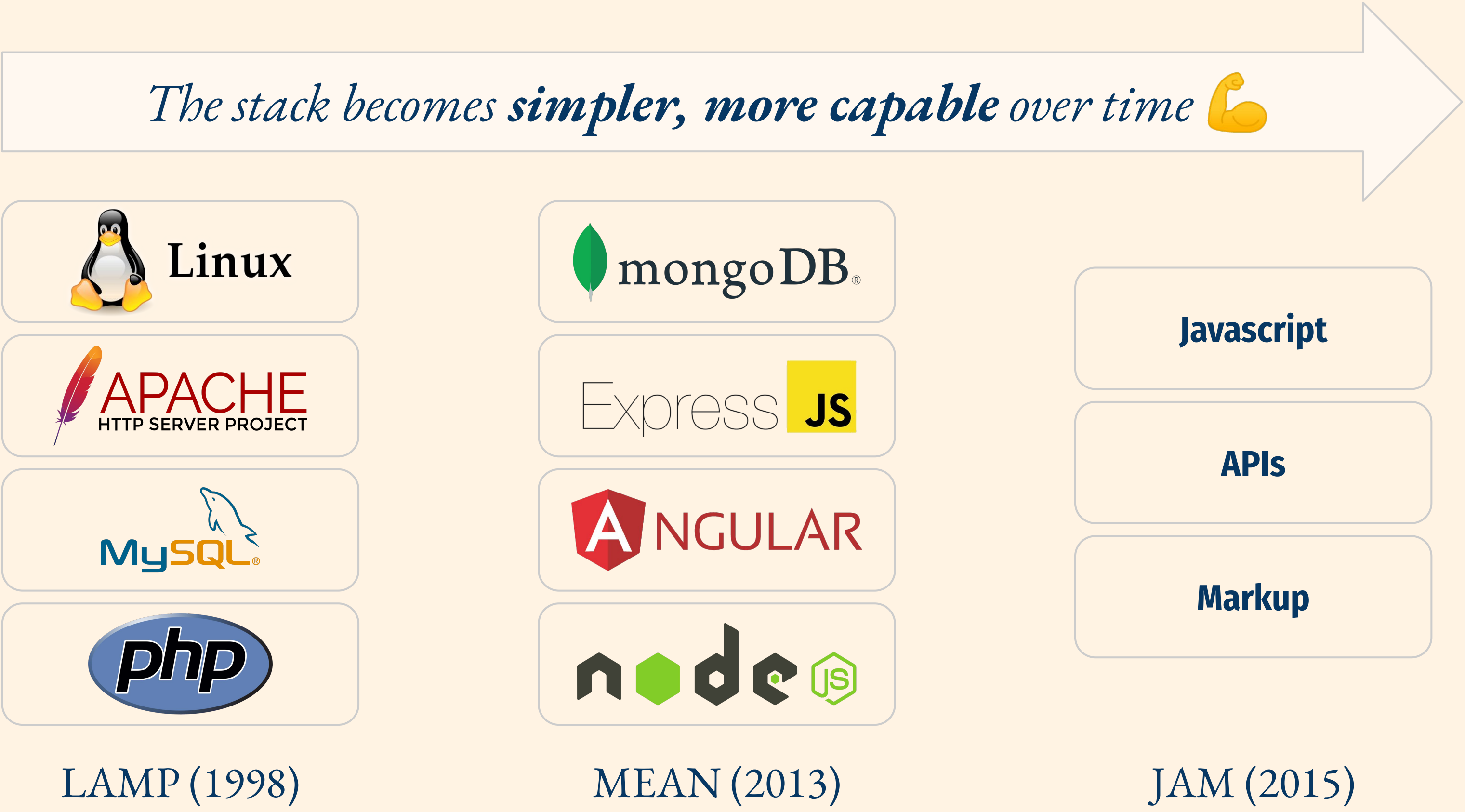
Javascript

APIs

Markup

JAM (2015)

The Evolution of Web Stacks



The stack becomes ***simpler, more capable*** *over time* 💪

Linux

APACHE HTTP SERVER PROJECT

MySQL

php

LAMP (1998)

mongoDB

Express JS

ANGULAR

node JS

MEAN (2013)

Javascript

APIs

Markup

JAM (2015)

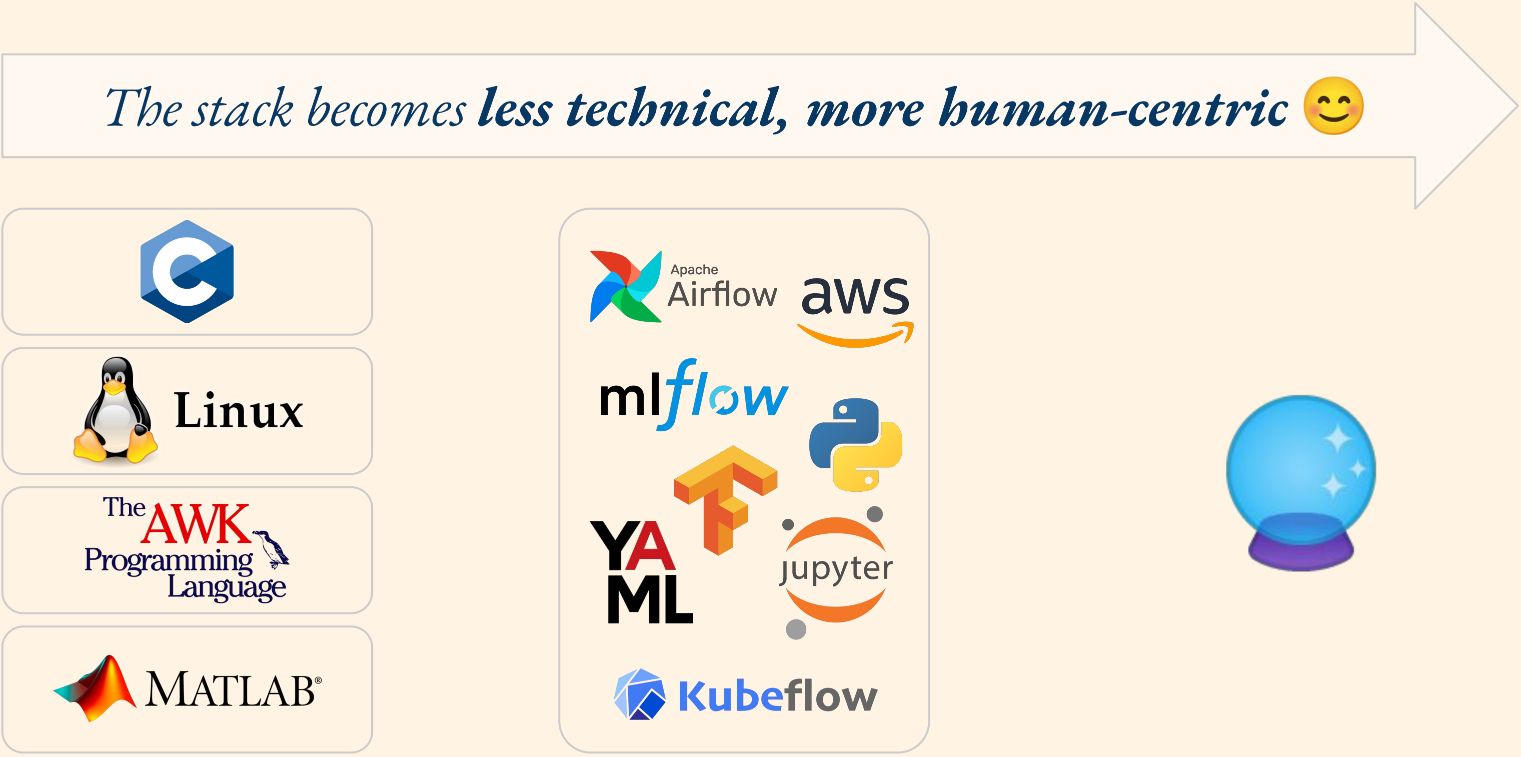
The stack for ML infrastructure will become

simpler, more capable 💪

&

more human-centric 😊

The Evolution of ML Stack



The stack becomes ***less technical, more human-centric*** 😊

CLAM (1998)

MLOps (2018)

Future?

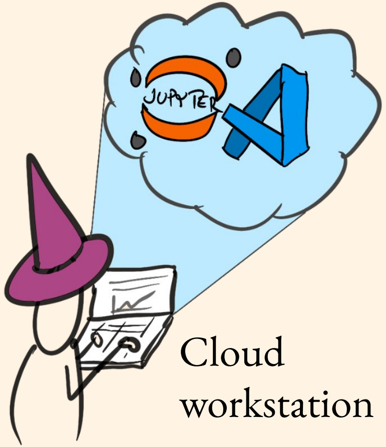
Let’s design

a modern ML stack

from the ground up

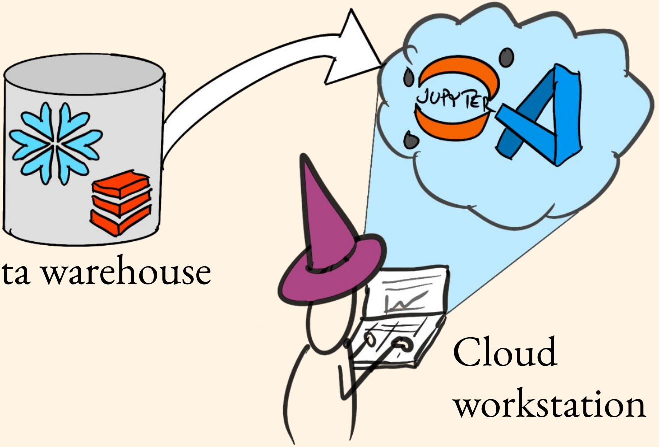
Here's a data scientist

A modern data scientist uses a cloud workstation



Cloud
workstation

Data flows seamlessly from the data warehouse to the workstation

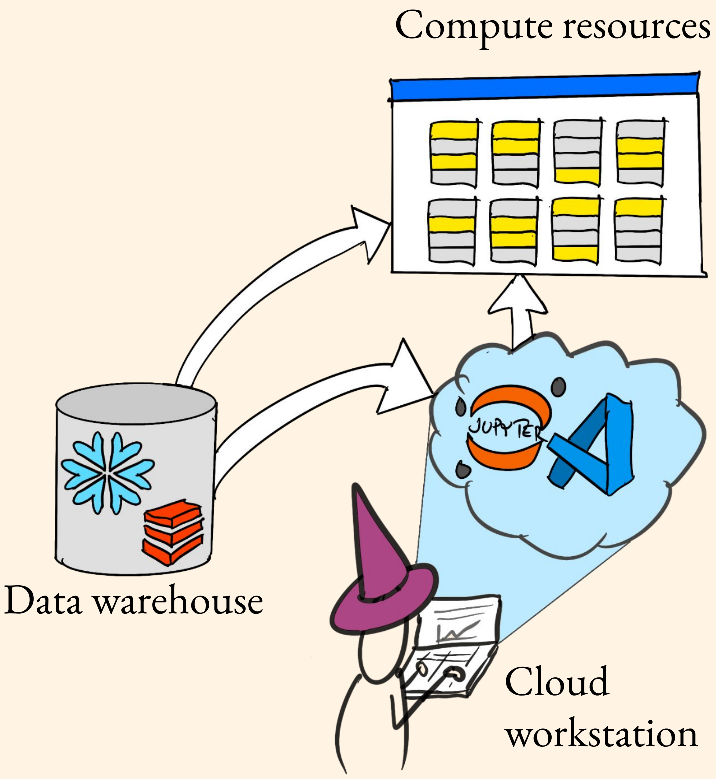


Data warehouse

Cloud workstation

Data

Experiments run at scale on a cloud-based compute cluster



Compute resources

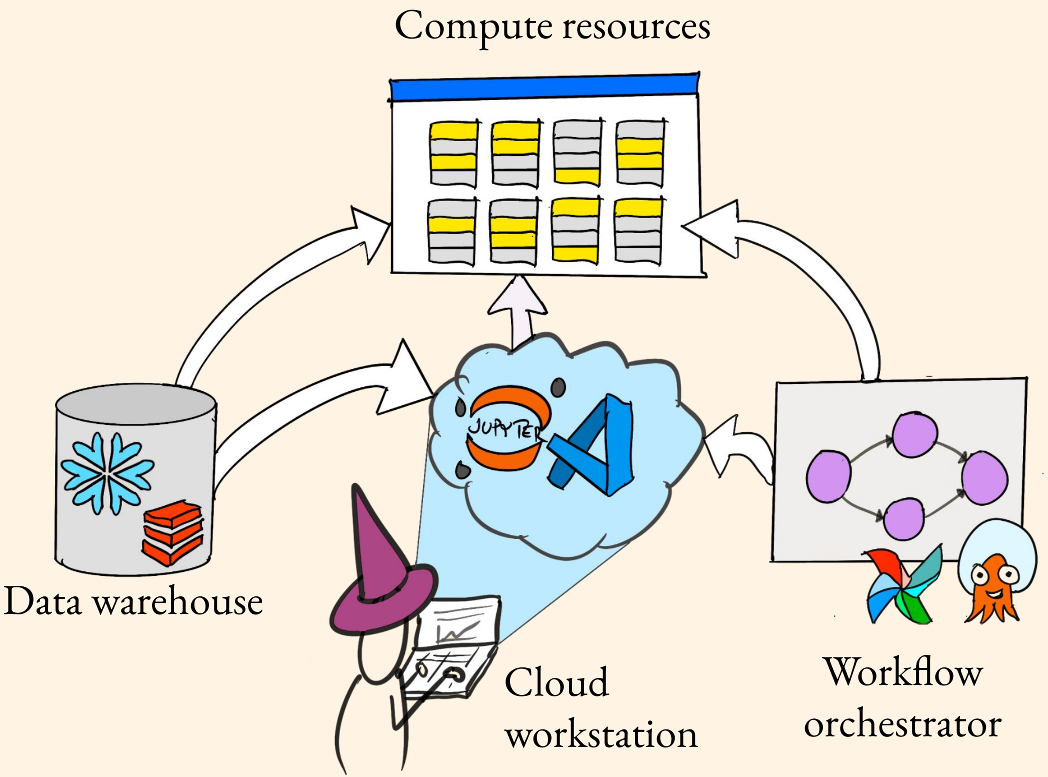
Data warehouse

Cloud workstation

Compute

Data

Complete workflows are developed and tested locally

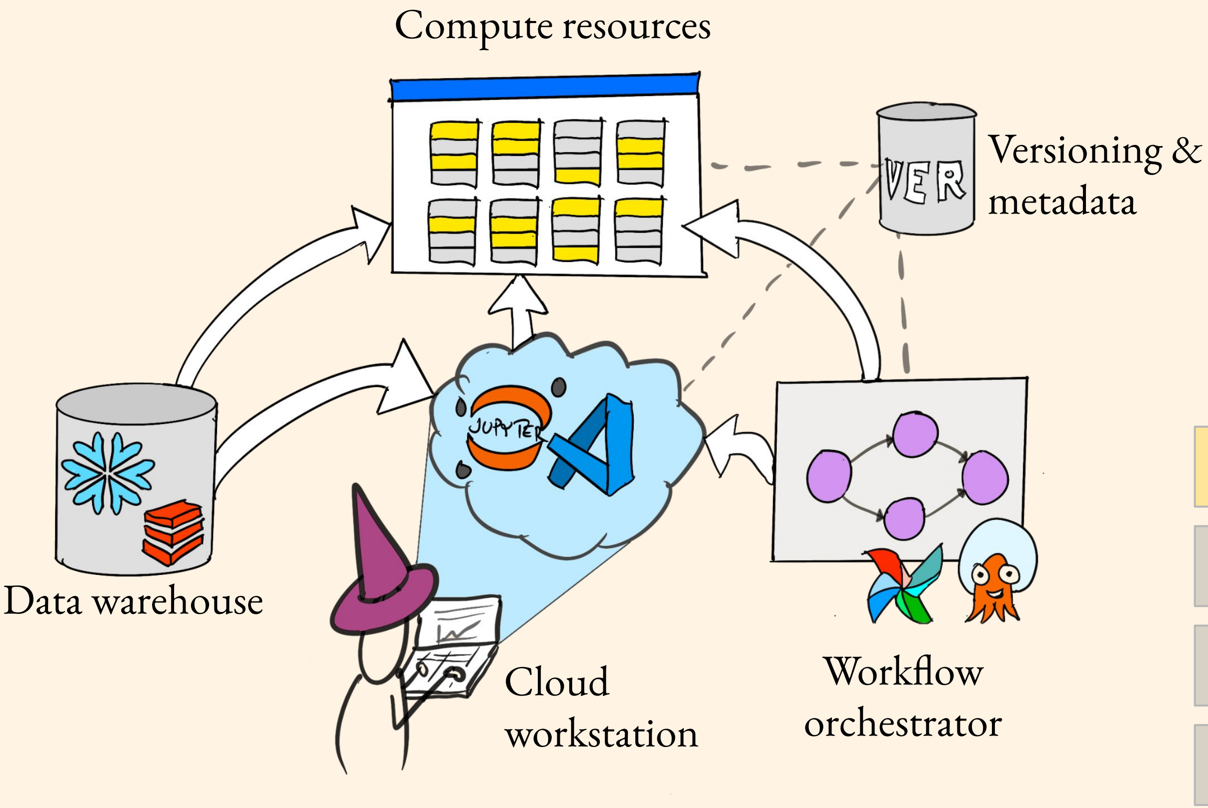


Orchestration

Compute

Data

Code, models, logs, and metrics gets stored and versioned automatically



Compute resources

Versioning & metadata

Data warehouse

Cloud workstation

Workflow orchestrator

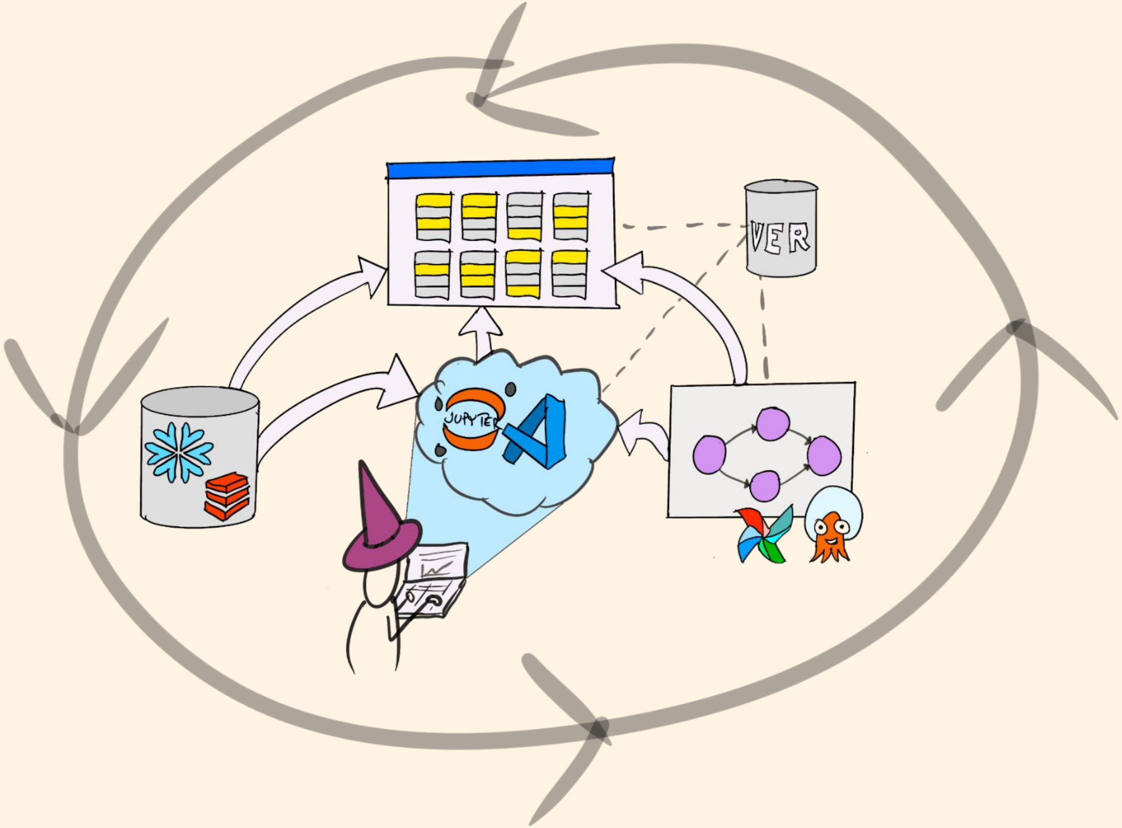
Versioning

Orchestration

Compute

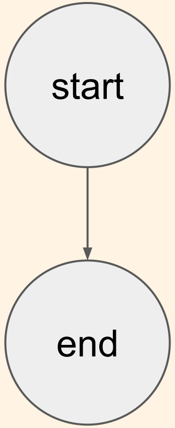
Data

Data Scientist can develop, test, and iterate on projects rapidly

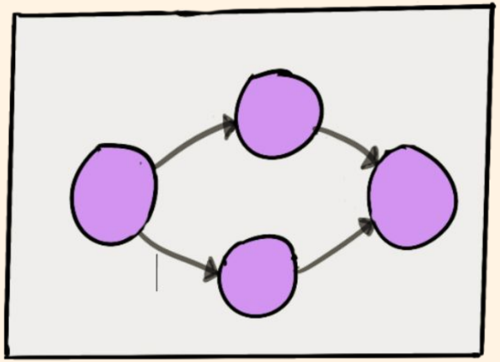


Example

Define workflows with a human-friendly syntax

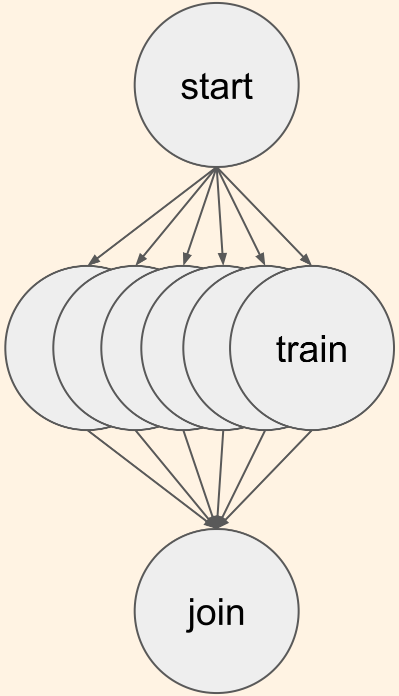


```
class MyFlow(FlowSpec):

    @step
    def start(self):
        import pandas as pd
        pd.DataFrame(big_one)
        self.next(self.end)

    @step
    def end(self):
        pass
```

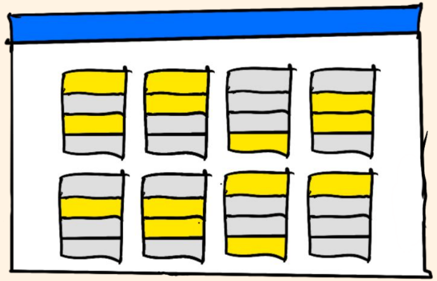


```
# python myflow.py run
```

Experiments run at scale on a cloud-based compute cluster

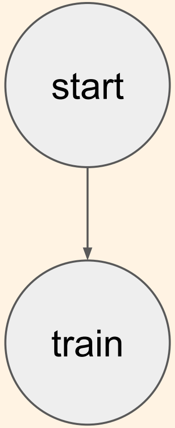


```
@step
def start(self):
    self.params = list(range(100))
    self.next(self.train, foreach='params')

@resources(memory=128000)
@step
def train(self):
    self.model = train(...)
    self.next(self.join)

@step
def join(self, inputs):
    ...
```



```
# python myflow.py run —with kubernetes
```

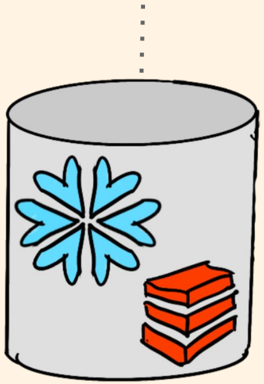
Everything gets versioned automatically



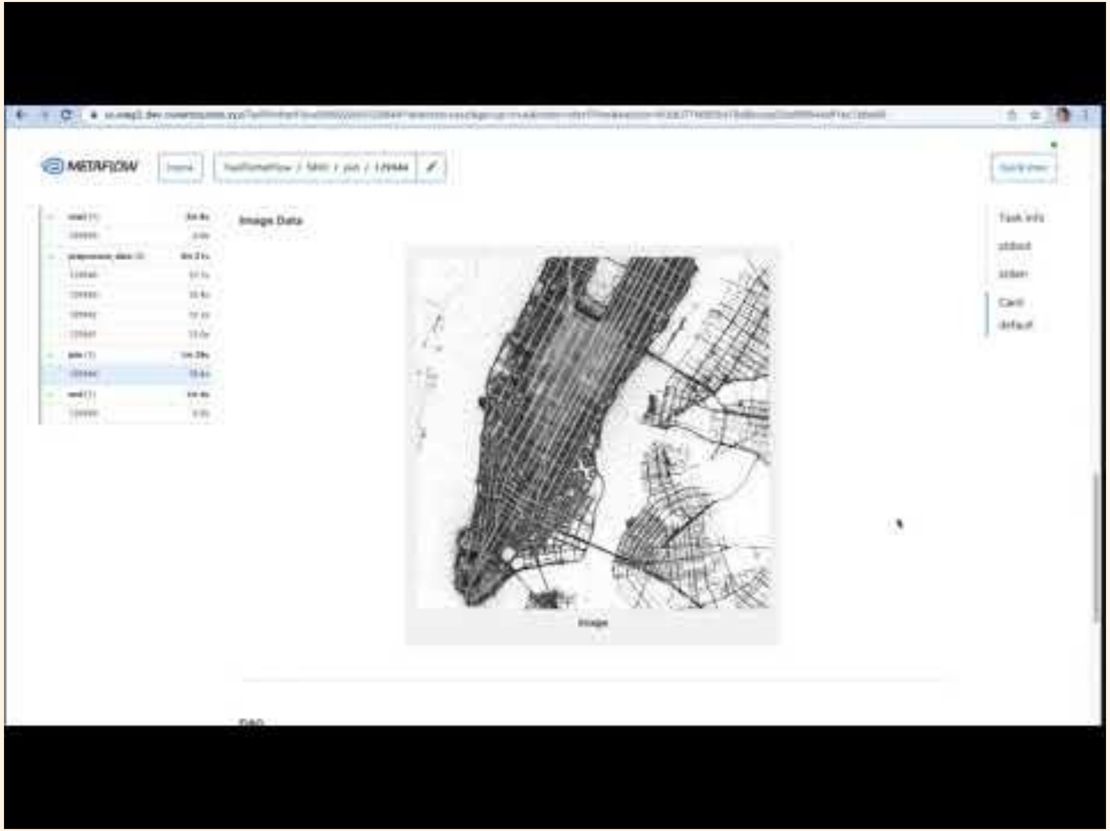
```
class MyFlow(FlowSpec):

    @step
    def start(self):
        self.alpha = 0.5
        self.next(self.train)

    @step
    def train(self):
        self.model = train_model(self.alpha)
```

Comes with tools for fast data access

```
class QueryFlow(FlowSpec):
    @step
    def query(self):
        self.ctas = "CREATE TABLE %s AS %s" % (self.table, self.sql)
        query = wr.athena.start_query_execution(self.ctas)
        output = wr.athena.wait_query(query)
        loc = output['ResultConfiguration']['OutputLocation']
        with metaflow.S3() as s3:
            results = [obj.url for obj in s3.list_recursive([loc])
```



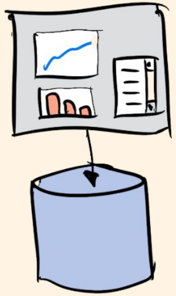
Data Scientist can develop, test, and iterate on projects rapidly



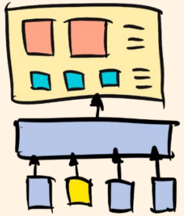
From prototype to Production

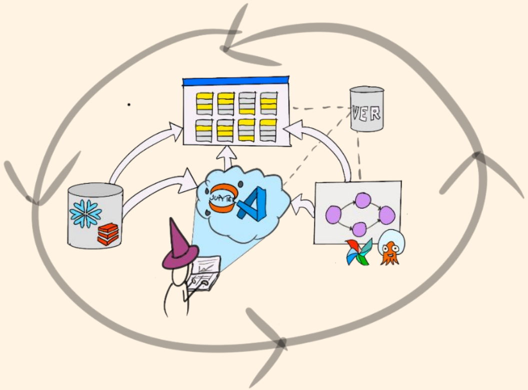
Real-world ML comes in many shapes and sizes

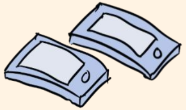
Decision-support systems



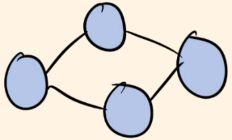
Product features







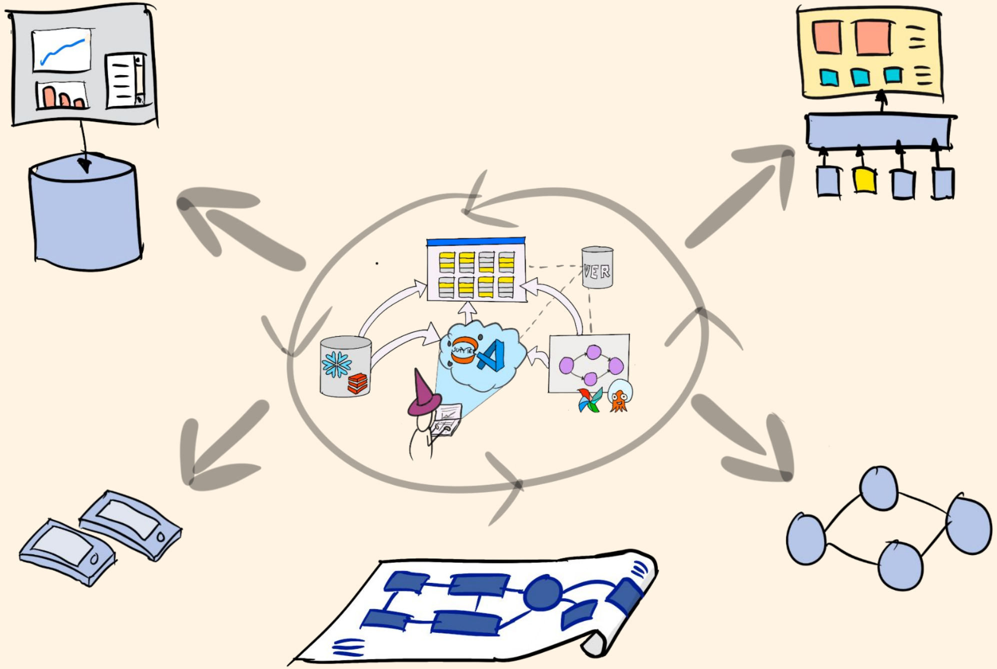
On-device ML



Data enrichment

There is not a single *production* but many

Provide architectural blueprints to support various deployment patterns



Architecture

Versioning

Orchestration

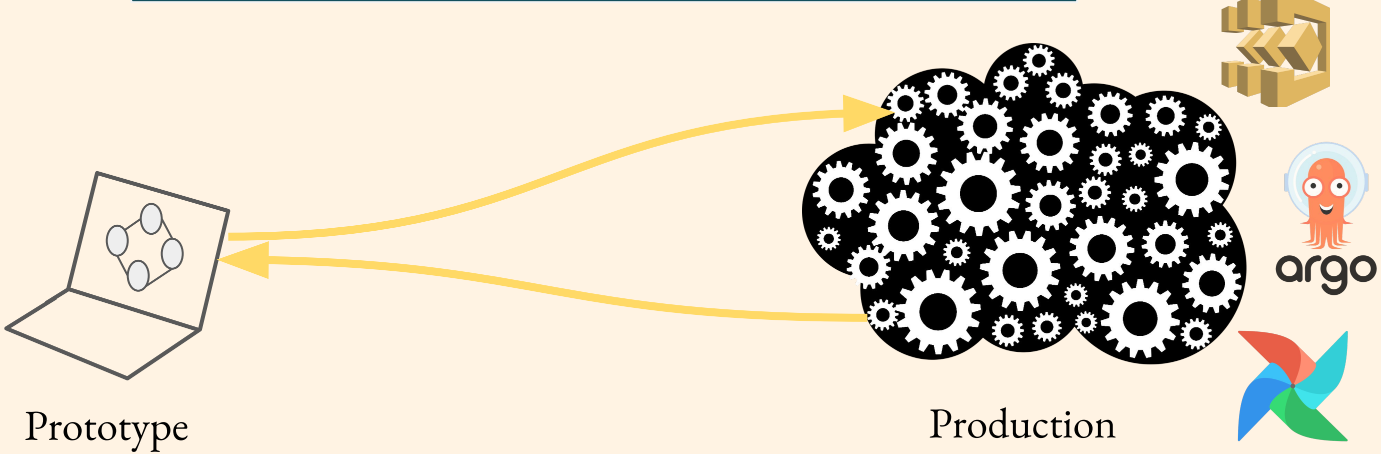
Compute

Data

Metaflow Example

Single-click deployment (and back)

```
# python myflow.py step-functions create
```

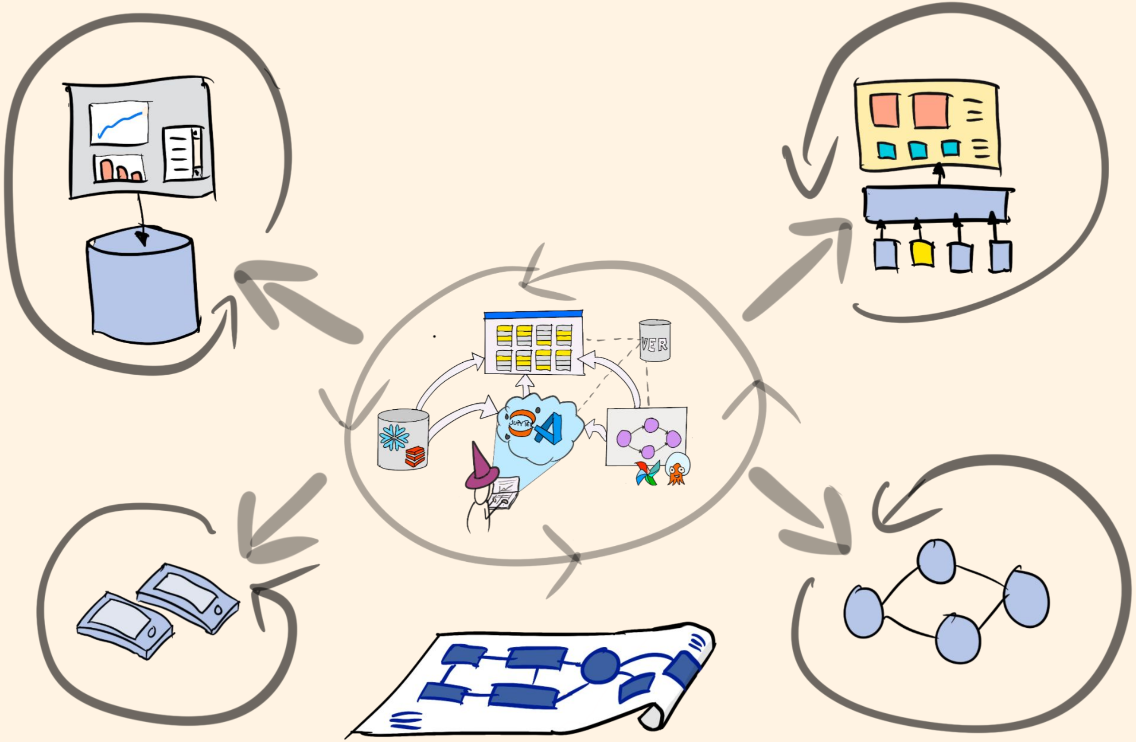


Prototype

Production

```
# python myflow.py resume --origin-run-id sfn-199874
```

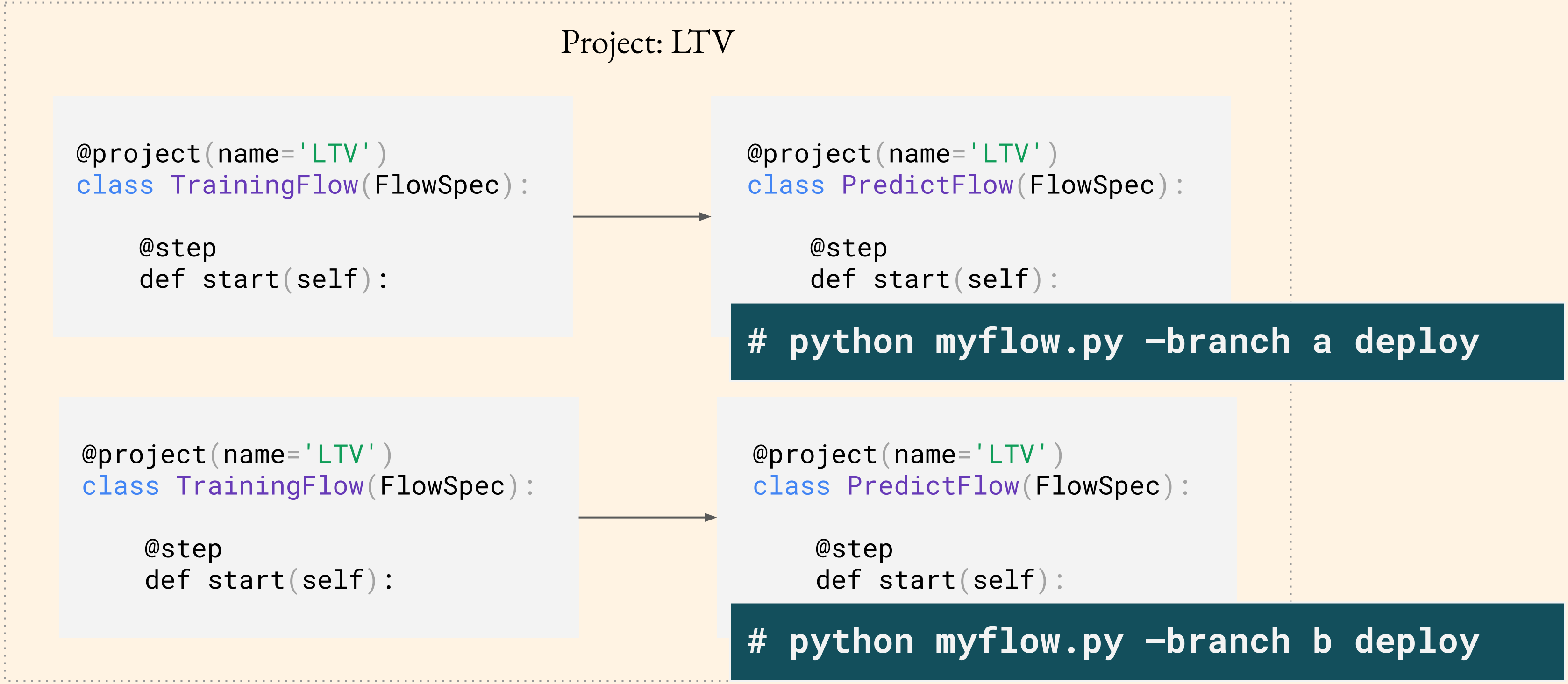
Continuous deployment, continuous experimentation



- Operations
- Architecture
- Versioning
- Orchestration
- Compute
- Data

Metaflow example

Deploy parallel models for A/B testing



Project: LTV

```
@project(name='LTV')
class TrainingFlow(FlowSpec):

    @step
    def start(self):
```

```
@project(name='LTV')
class PredictFlow(FlowSpec):

    @step
    def start(self):
```

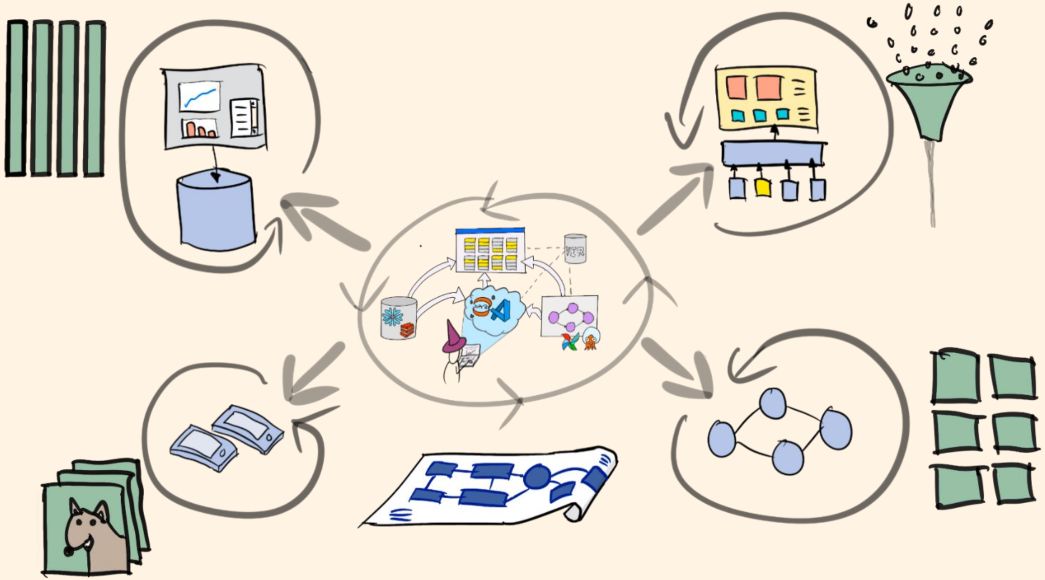
```
# python myflow.py –branch a deploy
```

```
@project(name='LTV')
class TrainingFlow(FlowSpec):

    @step
    def start(self):
```

```
@project(name='LTV')
class PredictFlow(FlowSpec):

    @step
    def start(self):
```

```
# python myflow.py –branch b deploy
```

Data scientists can experiment with features flexibly…



Relational data

Streaming events

Images

Semi-structured data

Features

Operations

Architecture

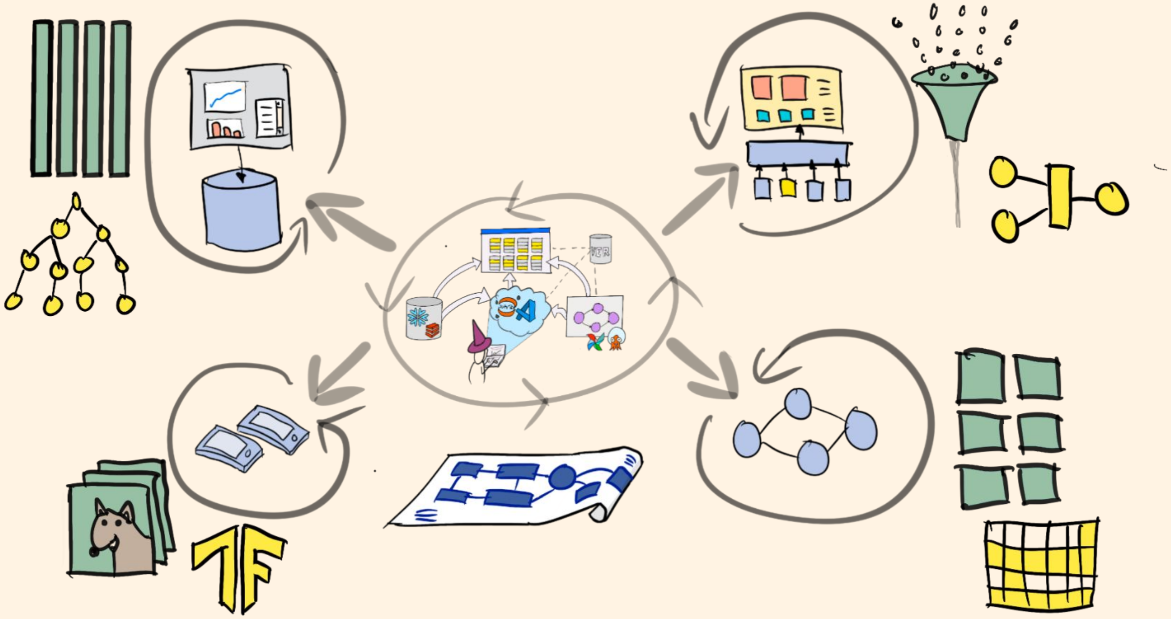
Versioning

Orchestration

Compute

Data

…as well as iterate on various modeling approaches…



Decisions trees

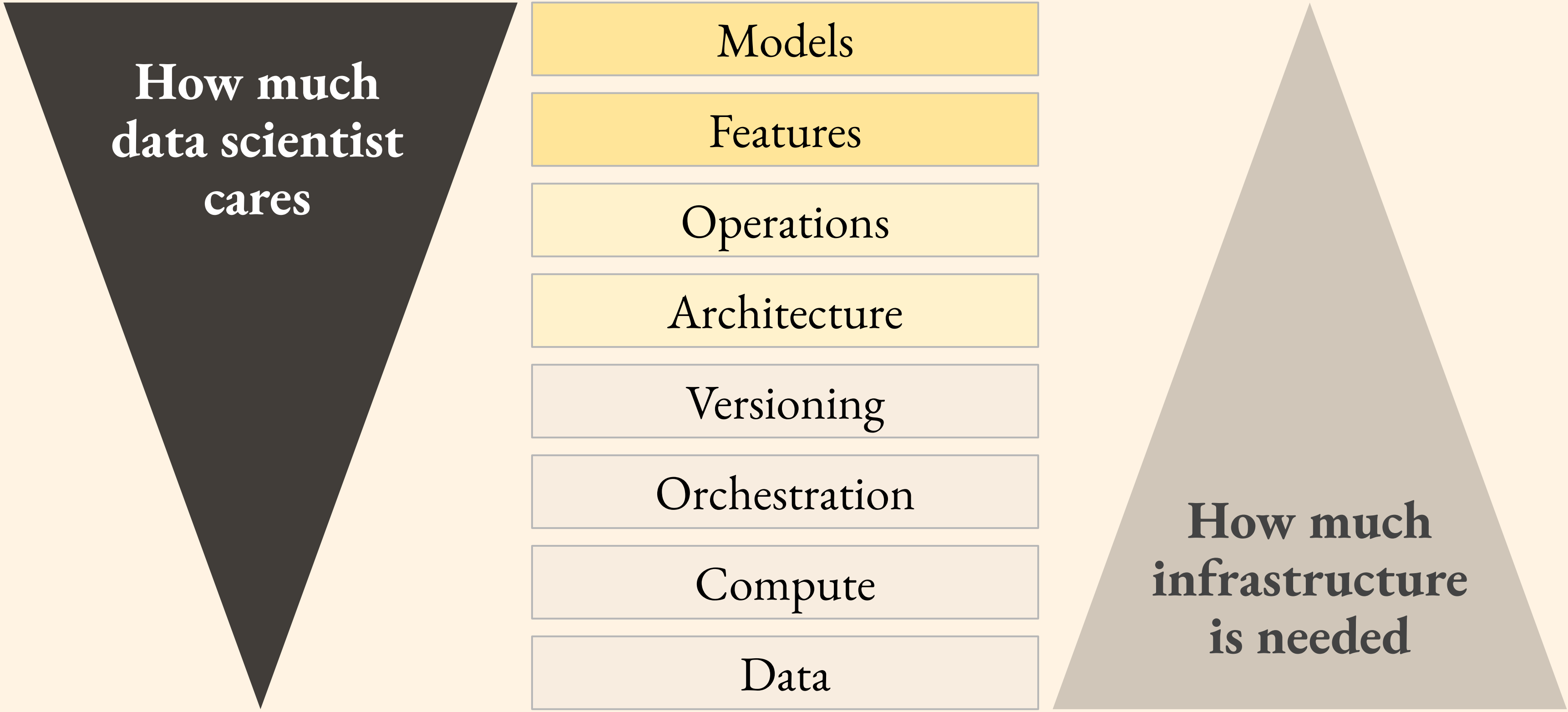
Model Ensembles

Deep neural networks

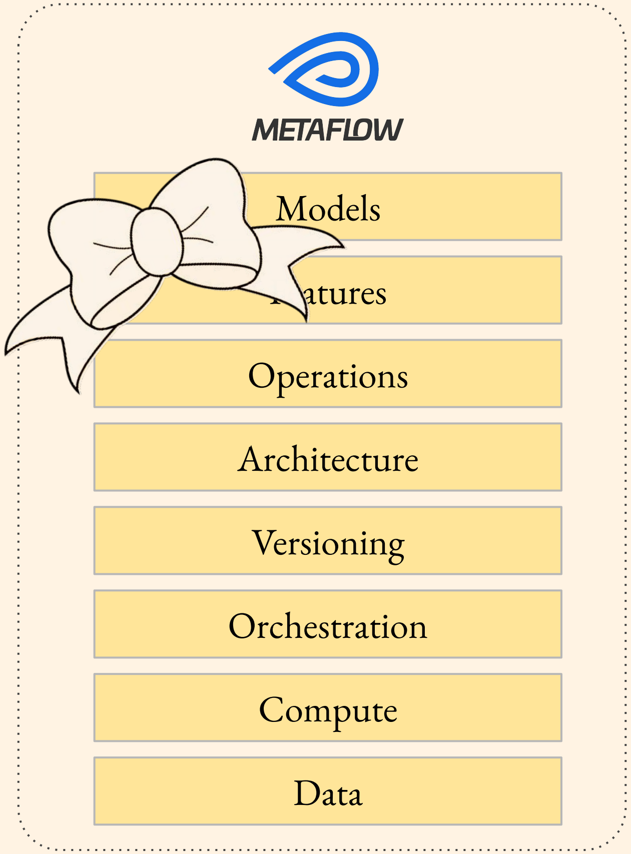
Embeddings

- **Models**
- Features
- Operations
- Architecture
- Versioning
- Orchestration
- Compute
- Data

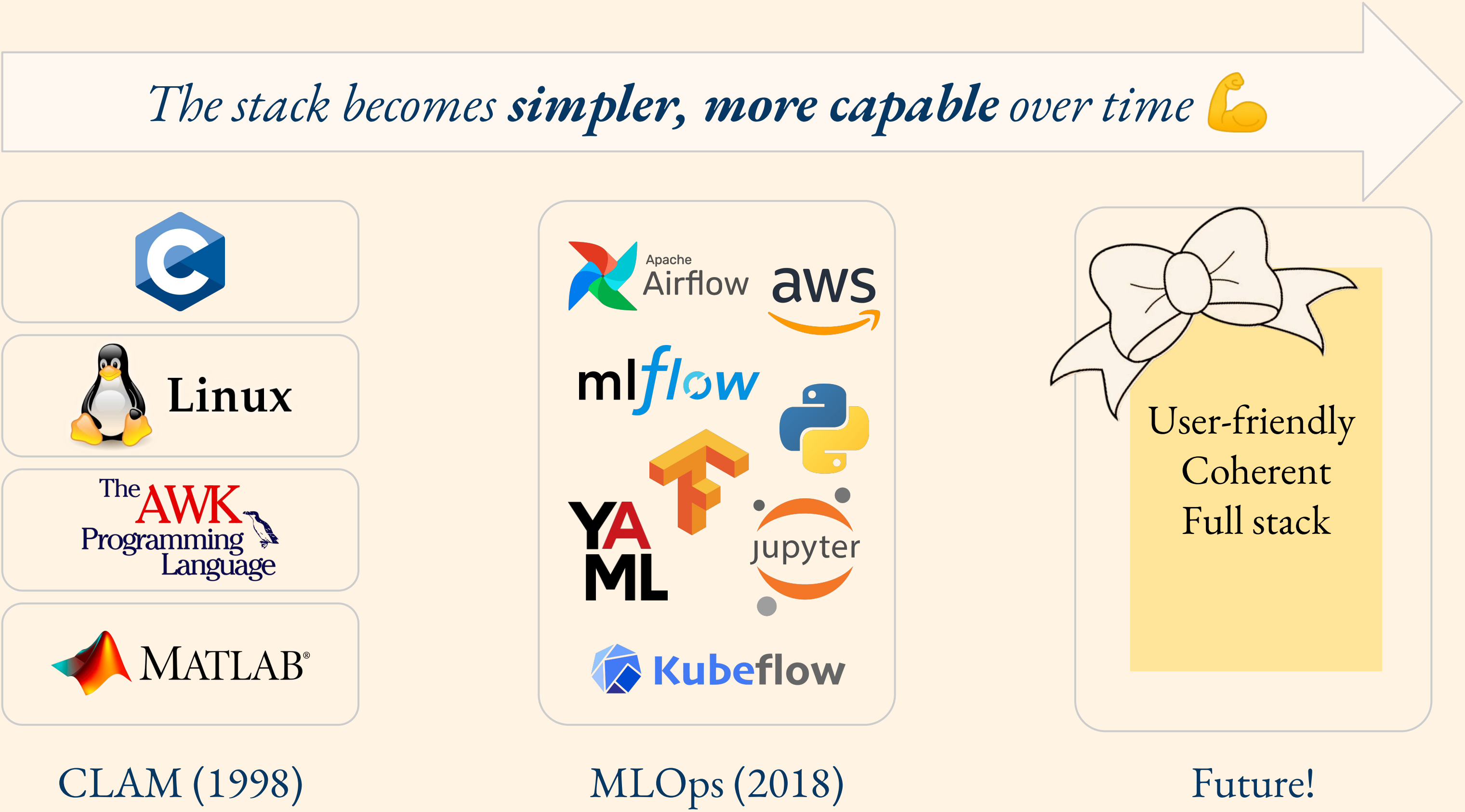
because that’s what data scientists are mostly supposed to do!



The full stack as a single, coherent, user-friendly package



The Evolution of ML Stack



*The stack becomes **simpler, more capable** over time* 💪

User-friendly
Coherent
Full stack

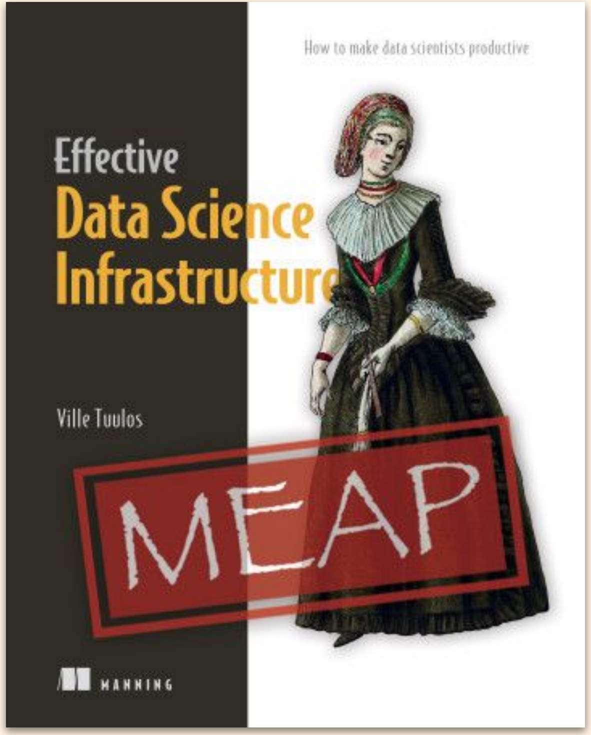
CLAM (1998)

MLOps (2018)

Future!

Shameless plug: New book!

Effective Data Science Infrastructure



Thank you

Curious to learn more about **open-source Metaflow**?
Join 1000+ data scientists and engineers at

http://slack.outerbounds.co

outerbounds

Get Ready for ML!

Level Up Your Data Lake

With Delta and lakeFS

Data Council – Austin

March 2022

Speakers

Adi Polak

Vice President of Developer Experience | Treeverse

Adi is an open-source technologist who believes in communities and is passionate about building a better world through open collaboration. As Vice President of Developer Experience at Treeverse, Adi helps build lakeFS, git-like interface for the data lakehouse. In her work, she brings her vast industry research and engineering experience to bear in educating and helping teams design, architect, and build cost-effective data systems and machine learning pipelines that emphasize scalability, expertise, and business goals.

Adi is a frequent worldwide presenter and the author of O'Reilly's upcoming book, "Machine Learning With Apache Spark." Adi is also a proud Beacon for Databricks! Previously, she was a senior manager for Azure at Microsoft, where she focused on building advanced analytics systems and modern architectures.

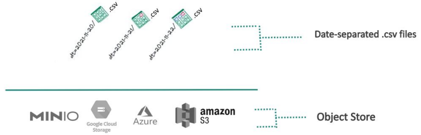
Paul Singman

Developer Advocate | Treeverse

Paul is a developer advocate for the lakeFS project, after several years on the analytics team at Equinox Fitness. His goal is to democratize big data analytics through explaining data architectures that are both user-friendly and cost-effective. He's spoken at various conferences and meetups, including the Postgres Conference NYC and AWS re:Invent. When not working you can find him drinking tea and playing golf

Narrative Flow

Level 0: Basic Data Lake

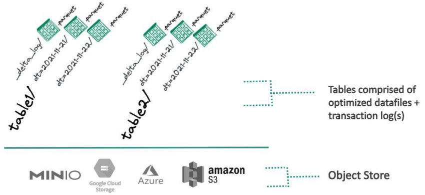


Date-separated .csv files

Object Store

Narrative Flow

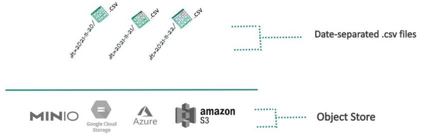
Level 1: Table-Format Enhanced



Tables comprised of optimized datafiles + transaction log(s)

Object Store

Level 0: Basic Data Lake

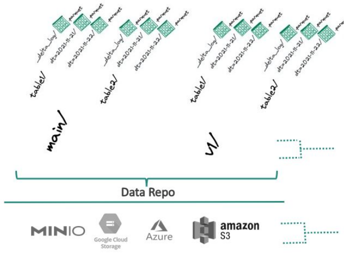


Date-separated .csv files

Object Store

Narrative Flow

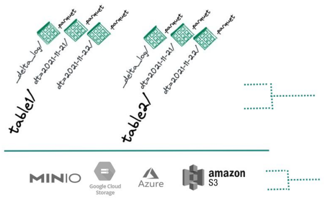
Level 2: Full Data Version Control



Branches of tables within a repository

Object Store

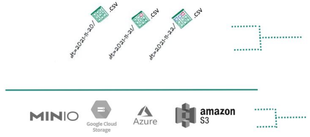
Level 1: Table-Format Enhanced



Tables comprised of optimized datafiles + transaction log(s)

Object Store

Level 0: Basic Data Lake



Date-separated .csv files

Object Store

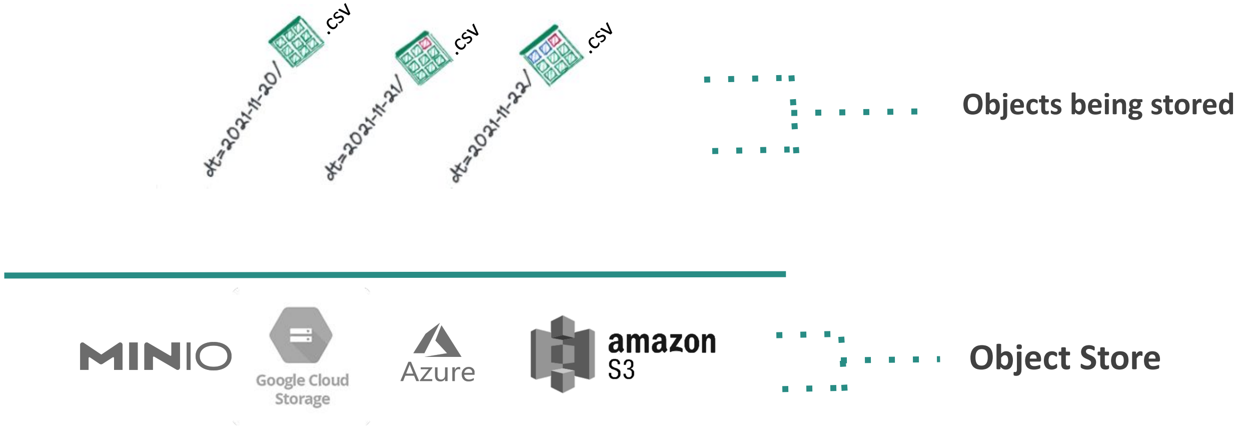
L0: Basic Data Lake

L0: Basic Data Lake

MINIO
Google Cloud Storage
Azure
amazon S3

Object Store

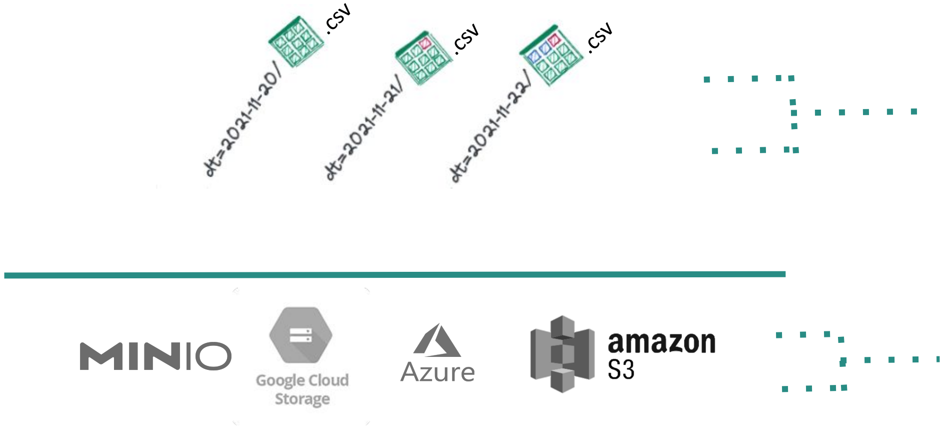
L0: Basic Data Lake



Objects being stored

Object Store

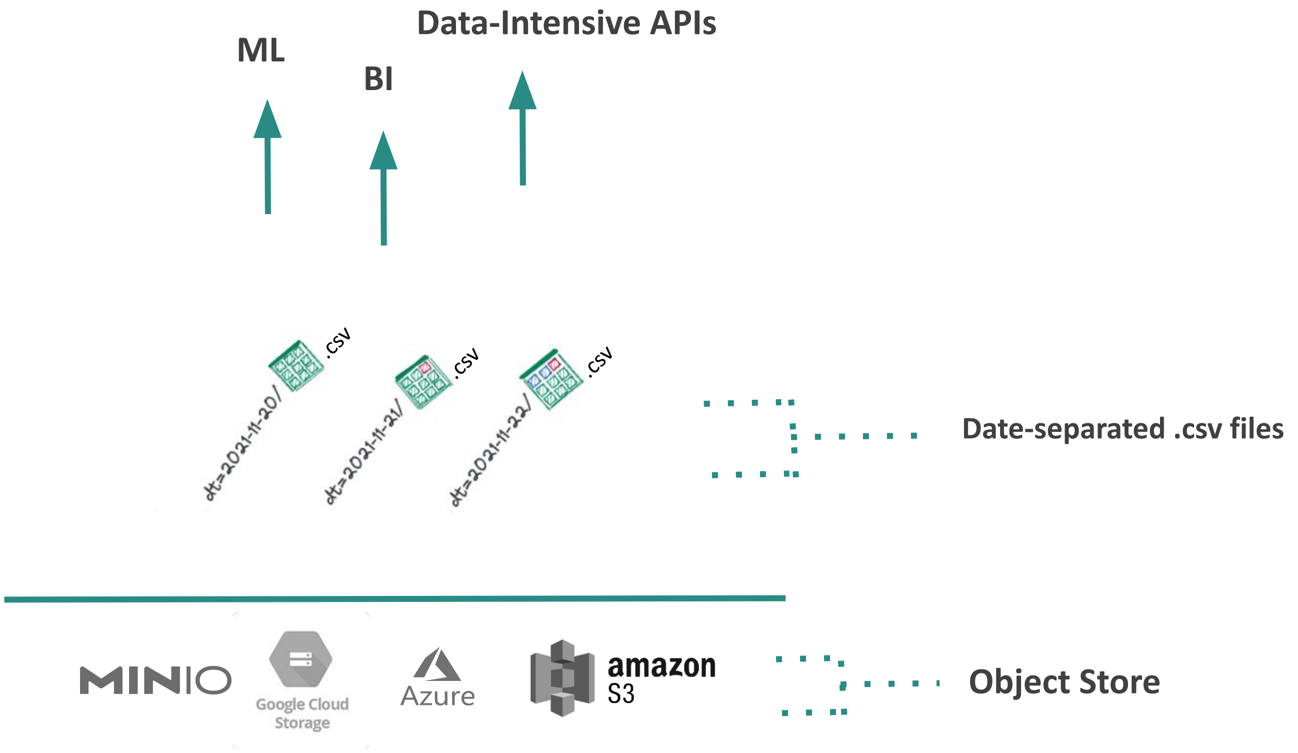
L0: Basic Data Lake



Date-separated .csv files

Object Store

L0: Basic Data Lake



Date-separated .csv files

Object Store

Why Object Storage?

Why Object Storage?

MINIO Google Cloud Storage Azure amazon S3 are **awesome** in terms of

Why Object Storage?

MINIO Google Cloud Storage Azure amazon S3 are **awesome** in terms of

- Performance
- Cost
- Developer Experience
- Connectivity

Why Object Storage?

MINIO Google Cloud Storage Azure amazon S3 are **awesome** in terms of

- **Performance**
- Cost
- Developer Experience
- Connectivity

- Achieve **3.5k** PUT requests per second **per prefix**
- **5.5k** GET requests per second **per prefix**
- Auto-scales to this limit automatically and overall capacity is limitless
- "something like 11 '9's of availability"

Why Object Storage?

MINIO Google Cloud Storage Azure amazon S3 are **awesome** in terms of

- Performance
- **Cost**
- Developer Experience
- Connectivity

- **Storage**: $.023 per GB vs $.10 for RDS or $.12 for EBS
- **Network:**
  - $5 per milllion PUT, $.40 per million GET requests,
  - $0 transfer data in, $.09 per GB for data transfer out
- ~5-8x times cheaper than block storage

Why Object Storage?

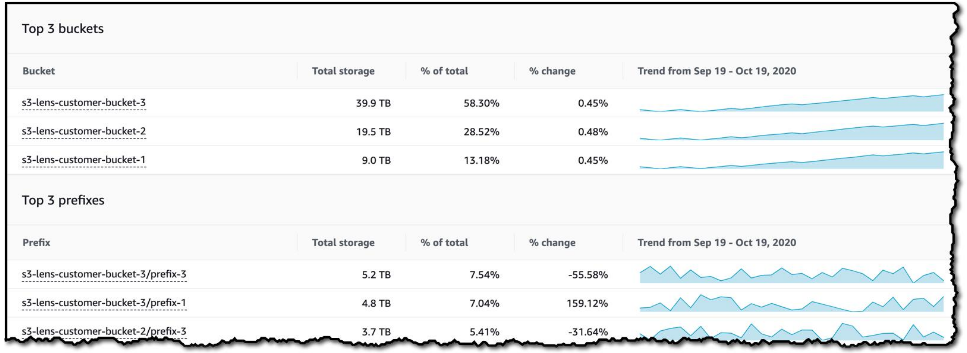
MINIO Google Cloud Storage Azure amazon S3 are **awesome** in terms of

- Performance
- Cost
- **Developer Experience**
- Connectivity

- Mature client SDKs
- Strong Consistency (2020)
- AWS Storage Lens (2020)
- Feature-rich (events, permissions, inventories, replication...)

Why Object Storage?

MINIO Google Cloud Storage Azure amazon S3 are **awesome** in terms of



Top 3 buckets

Bucket	Total storage	% of total	% change	Trend from Sep 19 - Oct 19, 2020
s3-lens-customer-bucket-3	39.9 TB	58.30%	0.45%	
s3-lens-customer-bucket-2	19.5 TB	28.52%	0.48%	
s3-lens-customer-bucket-1	9.0 TB	13.18%	0.45%	

Top 3 prefixes

Prefix	Total storage	% of total	% change	Trend from Sep 19 - Oct 19, 2020
s3-lens-customer-bucket-3/prefix-3	5.2 TB	7.54%	-55.58%	
s3-lens-customer-bucket-3/prefix-1	4.8 TB	7.04%	159.12%	
s3-lens-customer-bucket-2/prefix-3	3.7 TB	5.41%	-31.64%	

- Mature client SDKs
- Strong Consistency (2020)
- **AWS Storage Lens (2020)**
- Feature-rich (events, permissions, inventories, replication...)

Why Object Storage?

MINIO Google Cloud Storage Azure amazon S3 are **awesome** in terms of

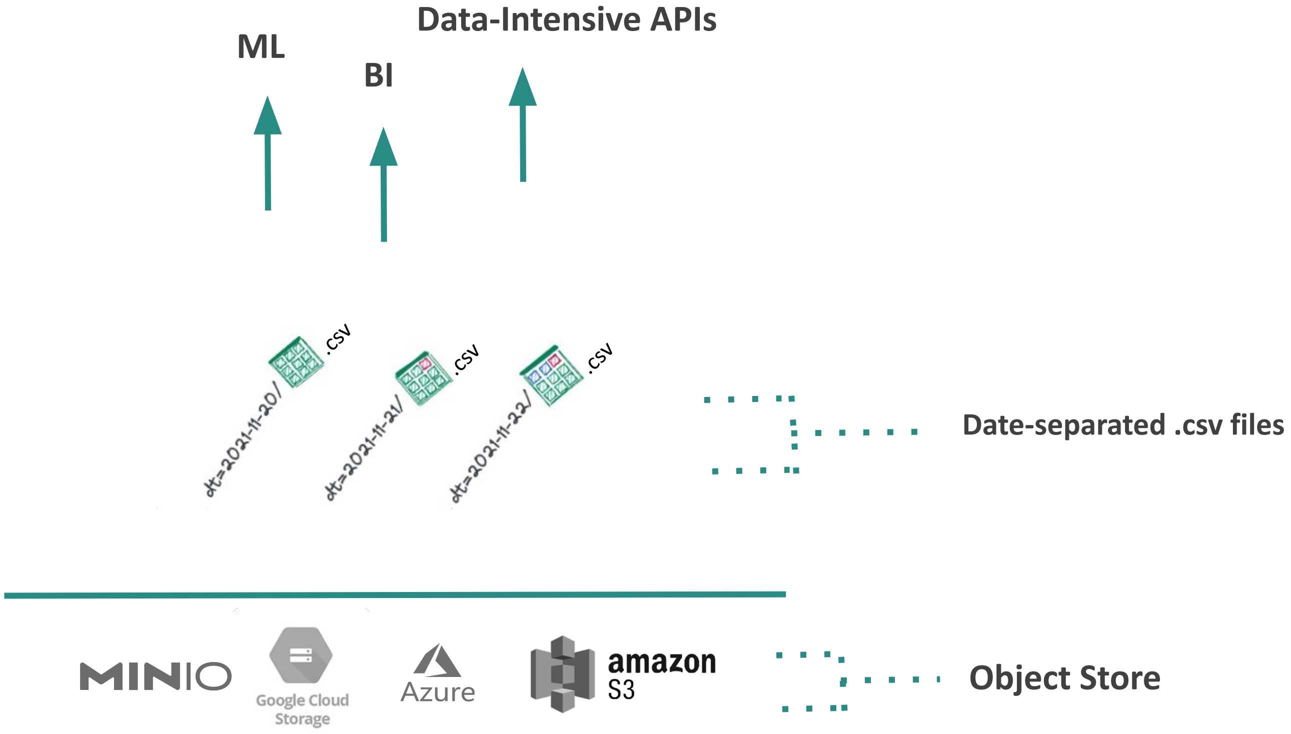
- Performance
- Cost
- Developer Experience
- **Connectivity**

Why Object Storage?

MINIO Google Cloud Storage Azure amazon S3 are **awesome** in terms of

- Performance
- Cost
- Developer Experience
- **Connectivity**

L0: Basic Data Lake



ML

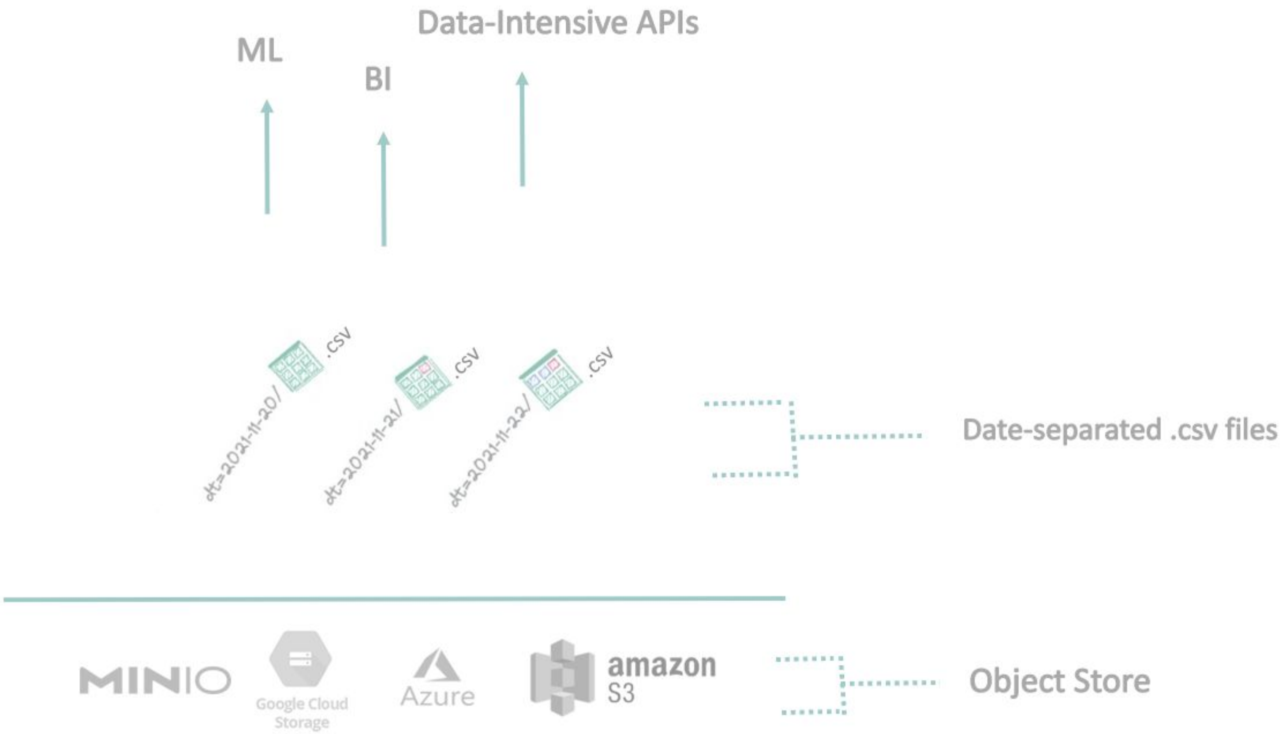
BI

Data-Intensive APIs

Date-separated .csv files

Object Store

L0: Basic Data Lake



Now let's make object store-specific improvements

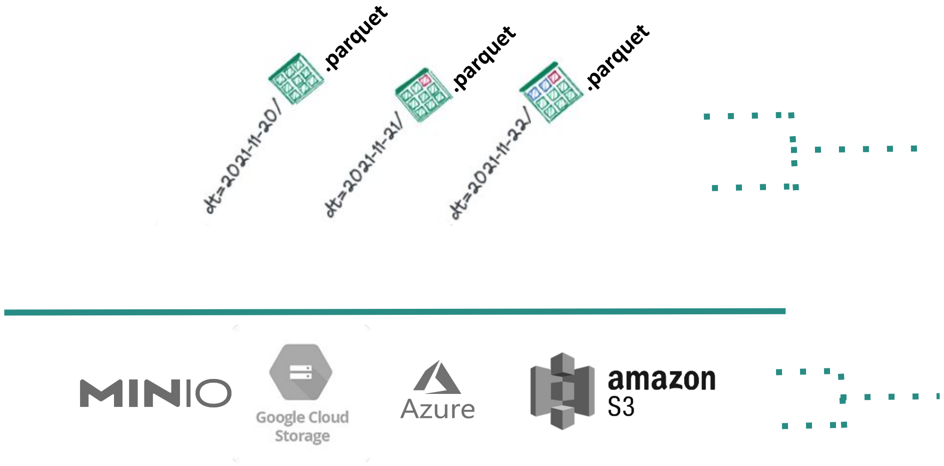
L0.5: Parquet File Format

.parquet

.parquet

.parquet

L0.5: Parquet File Format



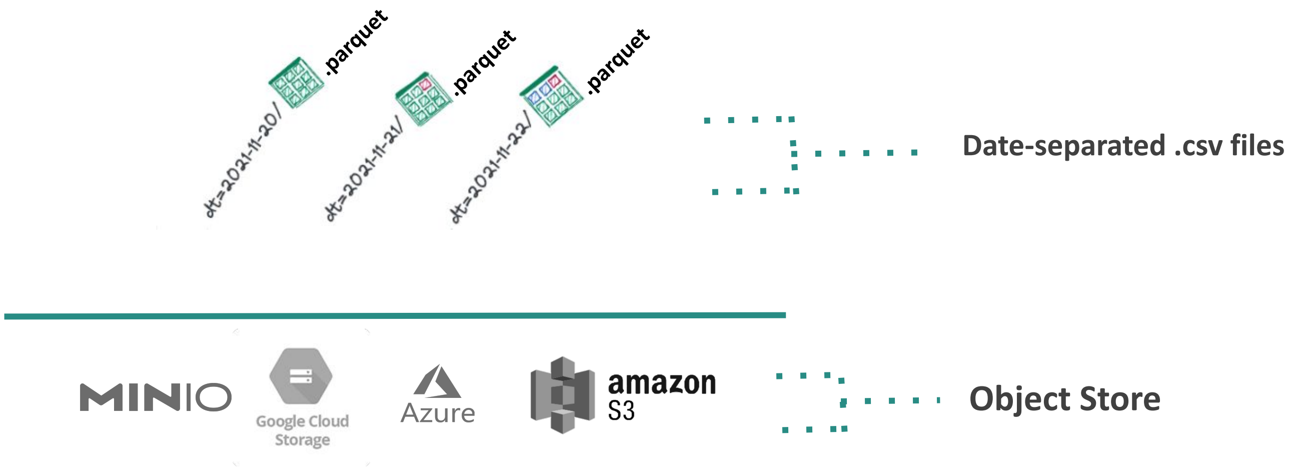
Date-separated .csv files

Object Store

L0.5: Parquet File Format

Benefits of parquet:

1. Columnar
2. Compressible
3. Complex



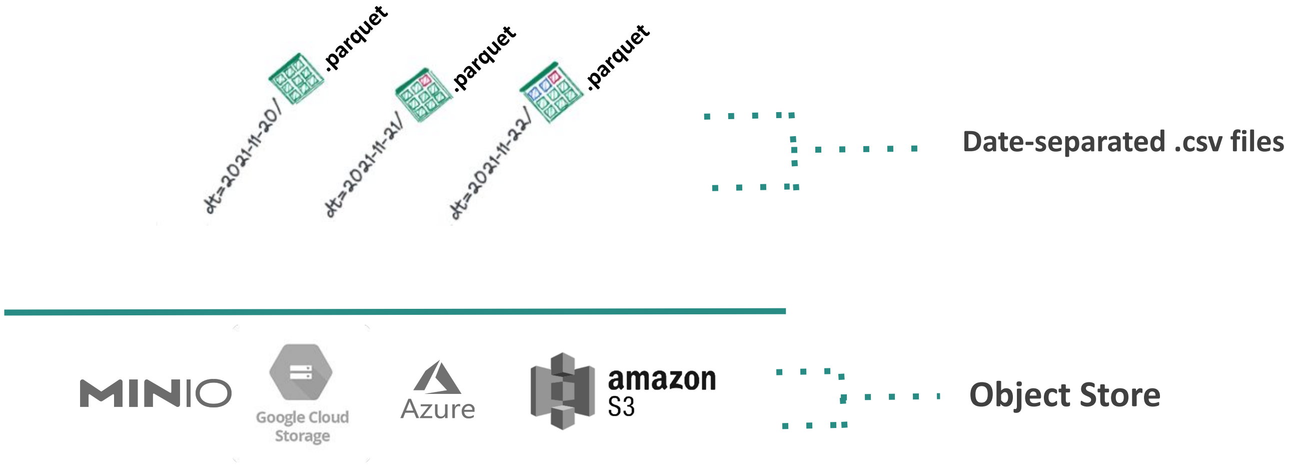
Date-separated .csv files

Object Store

L0.5: Parquet File Format

Challenges with parquet:

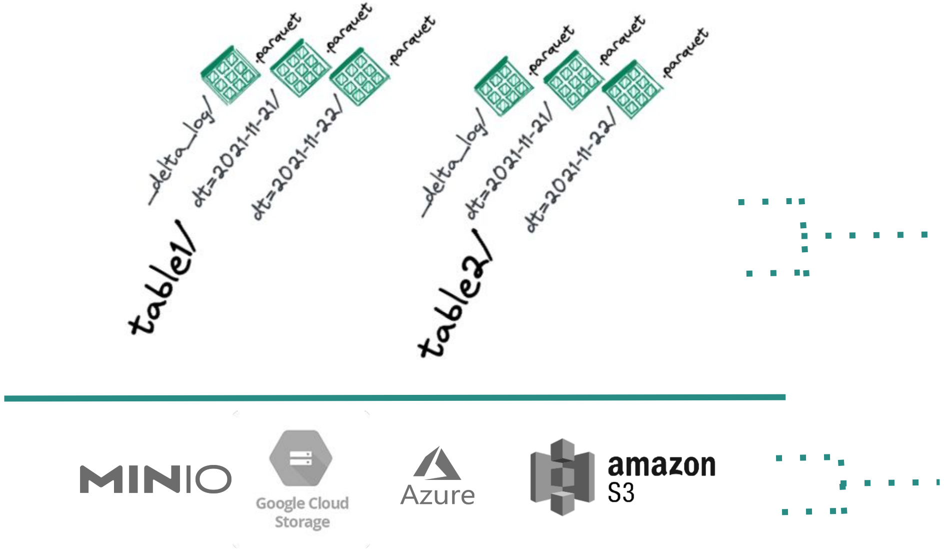
1. Operates at the object level



Date-separated .csv files

Object Store

L1: Modern Table Formats



Tables comprised of optimized datafiles + transaction log(s)

Object Store

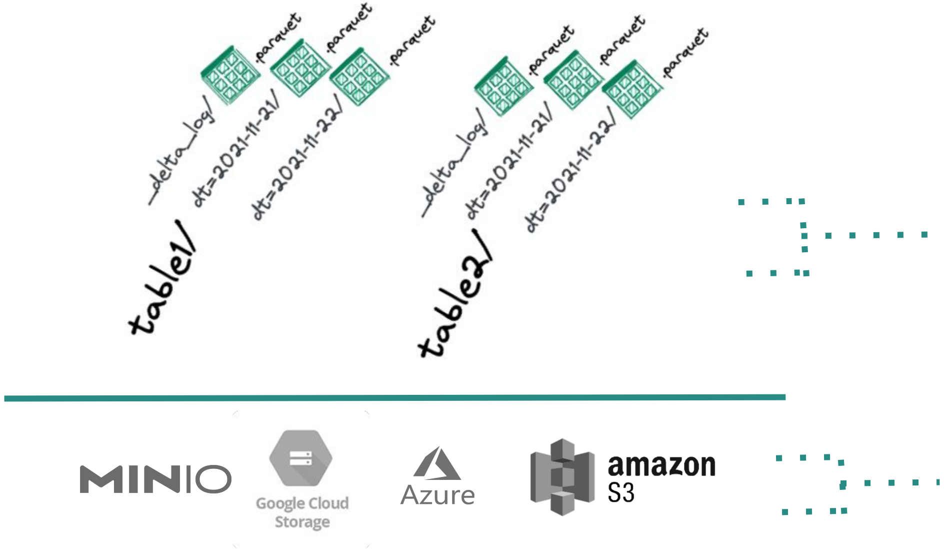
L1: Modern Table Formats

New Operations at the table level

- Define schema
- Traverse versions
- Upsert atomically

Implementations:

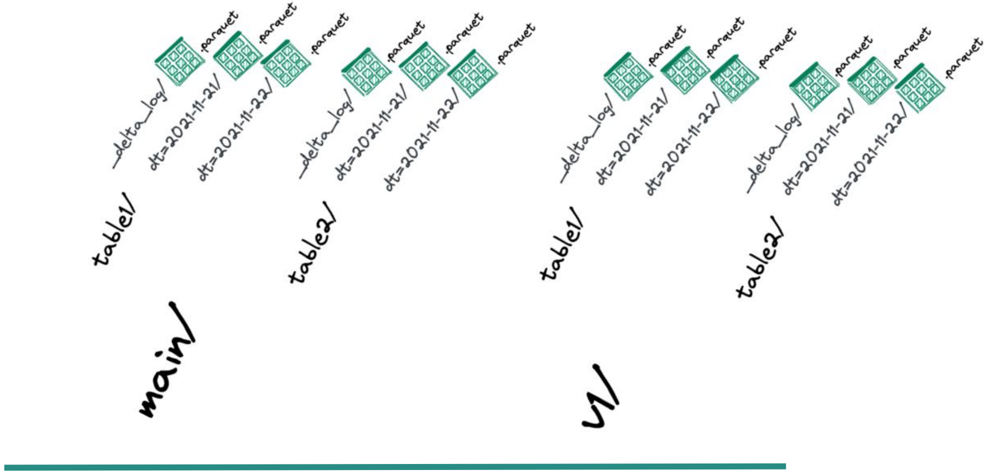
- Apache Hudi
- Apache Iceberg
- Delta Lake



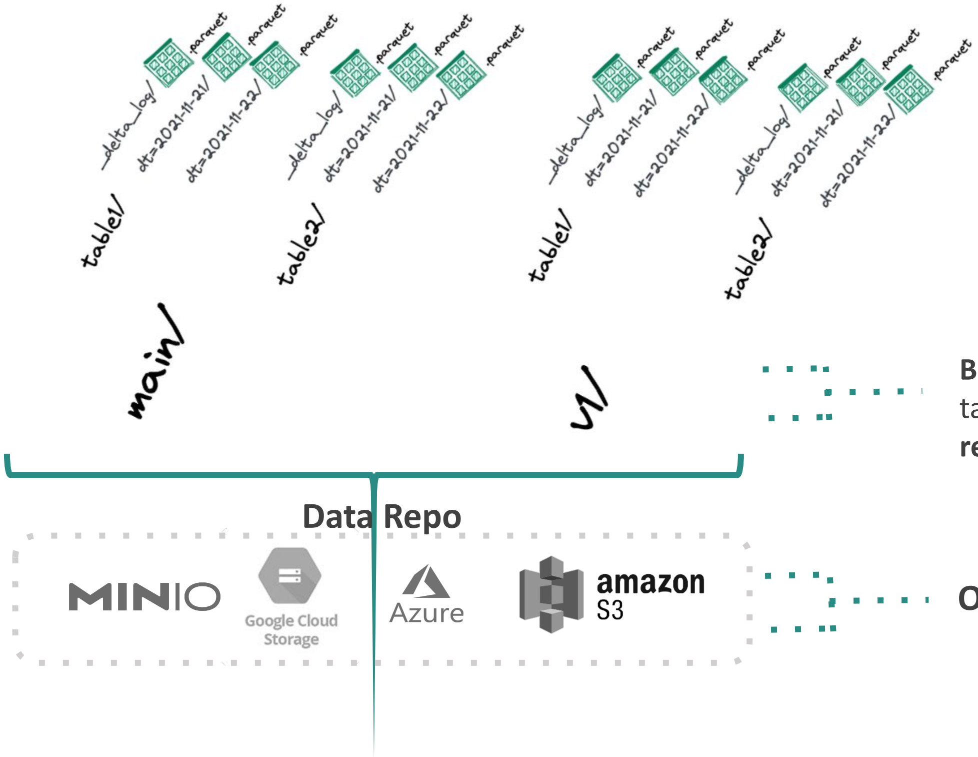
Tables comprised of optimized datafiles + transaction log(s)

Object Store

L2: Data Version Control



L2: Data Version Control



Branches of tables within a **repository**

Object Store

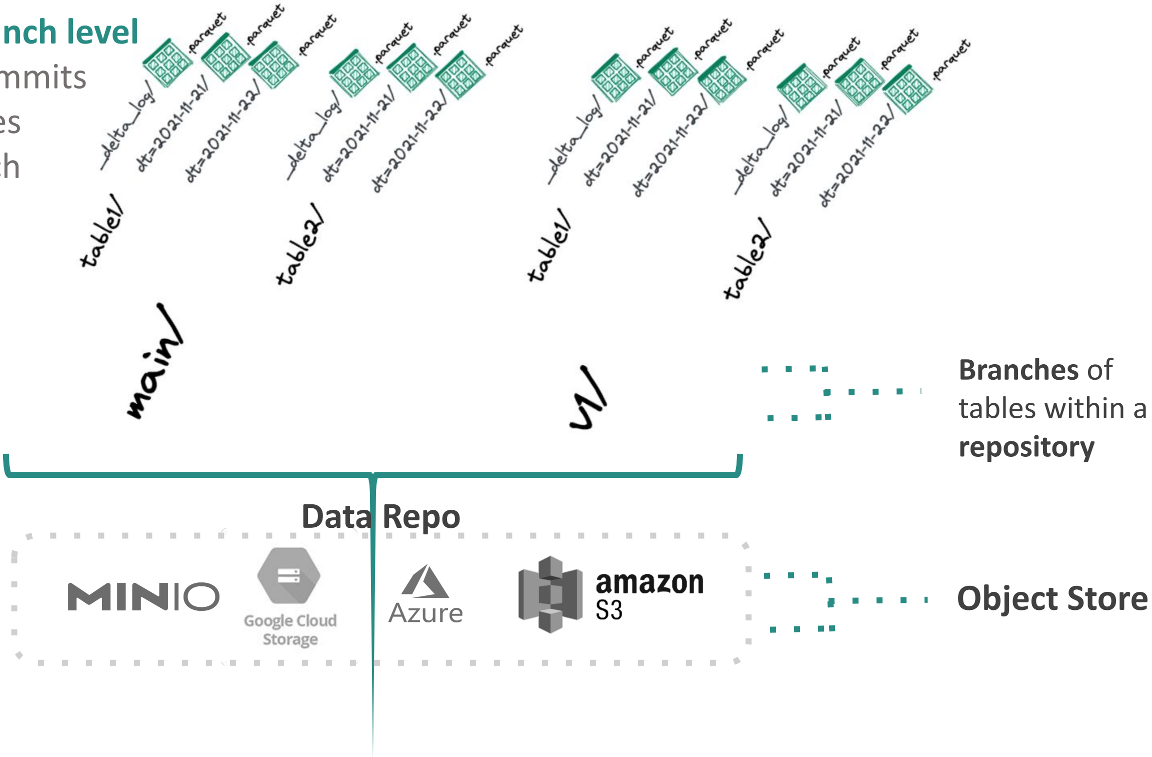
L2: Data Version Control

New Operations at the branch level

- Traverse among commits
- Merge two branches
- Create a new branch
- Take a commit

Implementations:

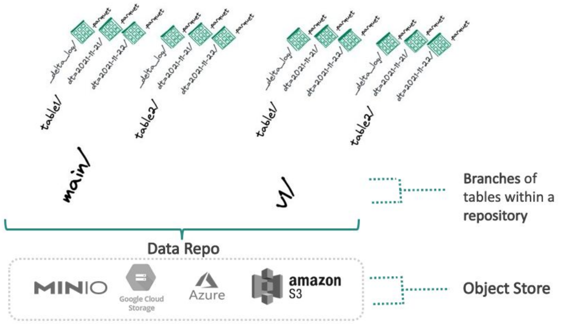
- **lakeFS**
- **Proj Nessie**



Branches of tables within a **repository**

Object Store

L2: Data Version Control Applications



Branches of tables within a repository

Data Repo

Object Store

New Operations at the branch level

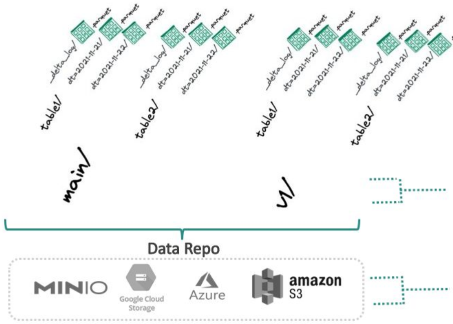
Traverse among commits

Merge two branches

Create a new branch

Take a commit

L2: Data Version Control Applications

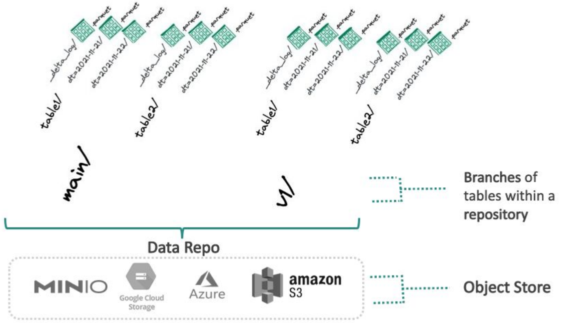


Branches of tables within a repository

Object Store

New Operations at the branch level	lakeFS CLI Example
Traverse among commits	`$ lakectl revert main^1`
Merge two branches	`$ lakectl merge my-branch-main`
Create a new branch	`$ lakectl branch create my-branch`
Take a commit	`$ lakectl commit –m "new commit" my-branch`

L2: Data Version Control Applications



Branches of tables within a repository

Data Repo

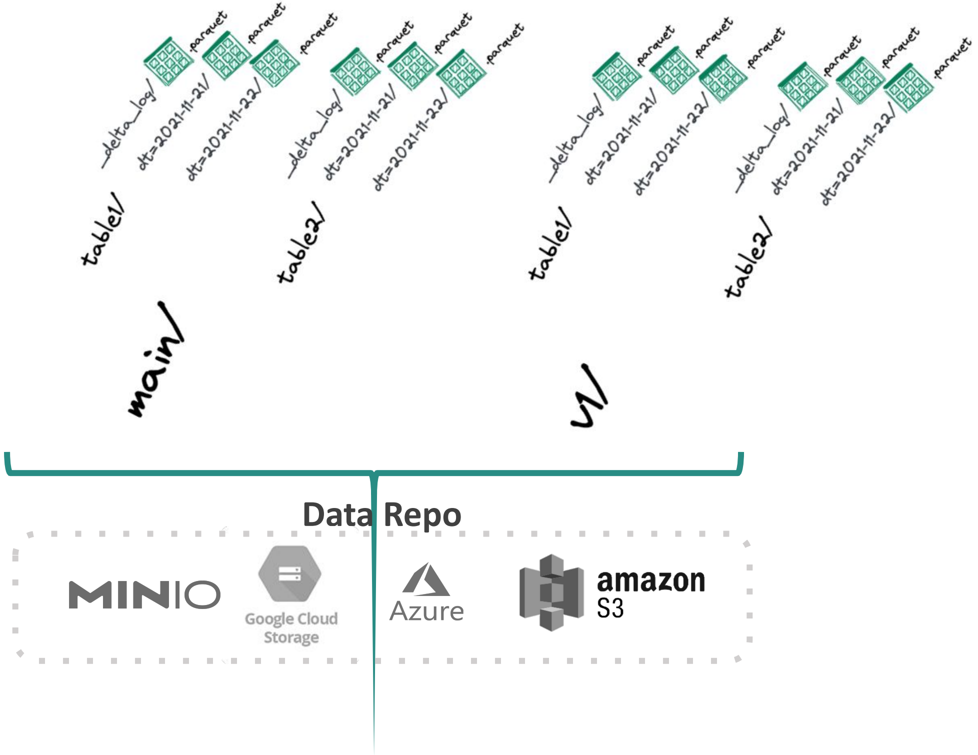
Object Store

New Operations at the branch level	lakeFS CLI Example	Useful for...
Traverse among commits	$ lakectl revert main^1	Instant recovery from issues
Merge two branches	$ lakectl merge my-branch-main	Atomic updates (cross-coll)
Create a new branch	$ lakectl branch create my-branch	Dev Environment creation
Take a commit	$ lakectl commit –m "new commit" my-branch	Reproducing ML experiments

Leveling Up Data Lake Takeaways

Stop operating at the **file level**

Start operating at the **table** and **repository level**



lakeFS.io

THANK YOU!

Type-safe Machine Learning Orchestration with Flyte and Pandera

D Data Council Austin 2022

Niels Bantilan, ML Engineer @ Union.ai

03/23/2022

Type-safety is a critical feature of *orchestration tools* that deal with *data* and *machine learning*

Types define the *set of values* that data can take, but they also define the *domain of operations* that we can perform on that data.

integers ∈ { 1, 2, -1, 5, 1000, ... }

strings ∈ { “a”, “xyz”, “hello”, “foobar”, ...}

✅ 1 + 1 → 2

❌ 1 + “a” → *undefined*

✅ mean([1, 2, 3]) → 2

❌ mean([“a”, “b”, “a”, “c”]) → *undefined*

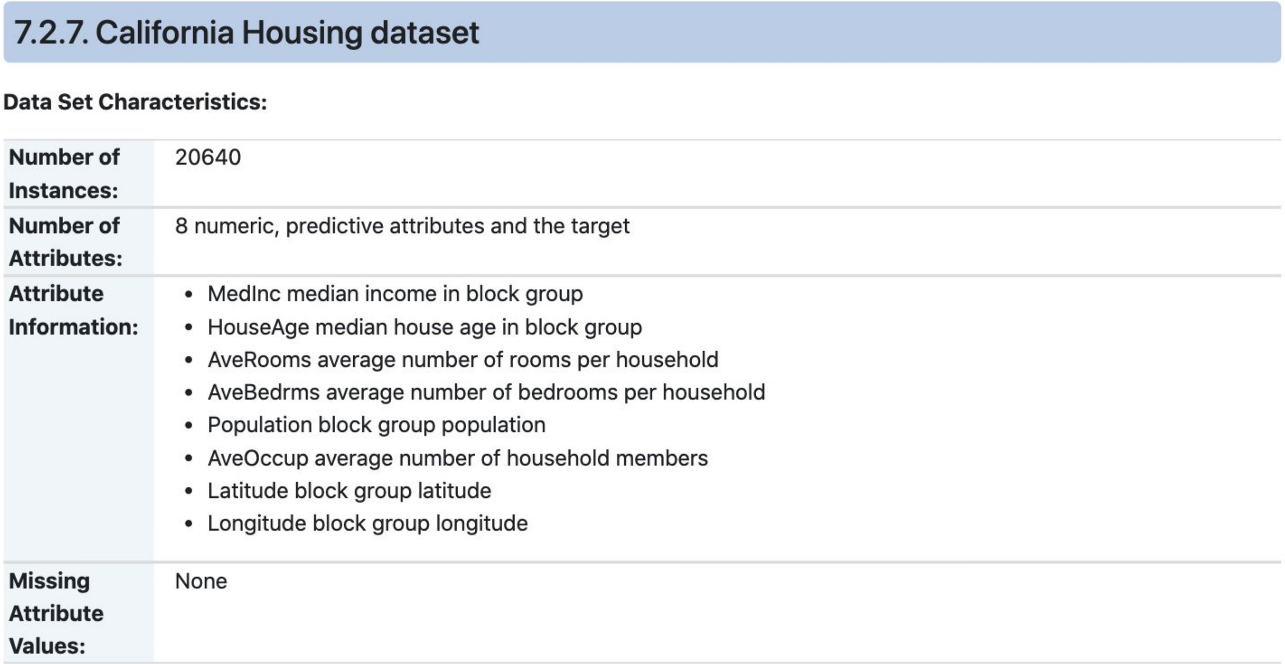
Types can be simple:

```
int, float, str
```

Or more complex:

```
list[int]
dict[str, float]
dict[str, list[float]]
```

Let’s talk about housing 🏡



7.2.7. California Housing dataset

Data Set Characteristics:

Number of Instances:	20640
Number of Attributes:	8 numeric, predictive attributes and the target
Attribute Information:	• MedInc median income in block group • HouseAge median house age in block group • AveRooms average number of rooms per household • AveBedrms average number of bedrooms per household • Population block group population • AveOccup average number of household members • Latitude block group latitude • Longitude block group longitude
Missing Attribute Values:	None

Source: https://scikit-learn.org/stable/datasets/real_world.html#california-housing-dataset

Let’s talk about housing 🏡

```
pandas.DataFrame({
    'Latitude': [37.88, …],
    'Longitude': [-122.23, …],
    'AveBedrms': [1.0238, …],
    'AveOccup': [2.5555, …],
    'AveRooms': [6.9841, …],
    'HouseAge': [41.0, …],
    'MedInc': [8.3252, …],
    'Population': [322.0, …],
    'MedHouseVal': [4.526, …],
})
```

float

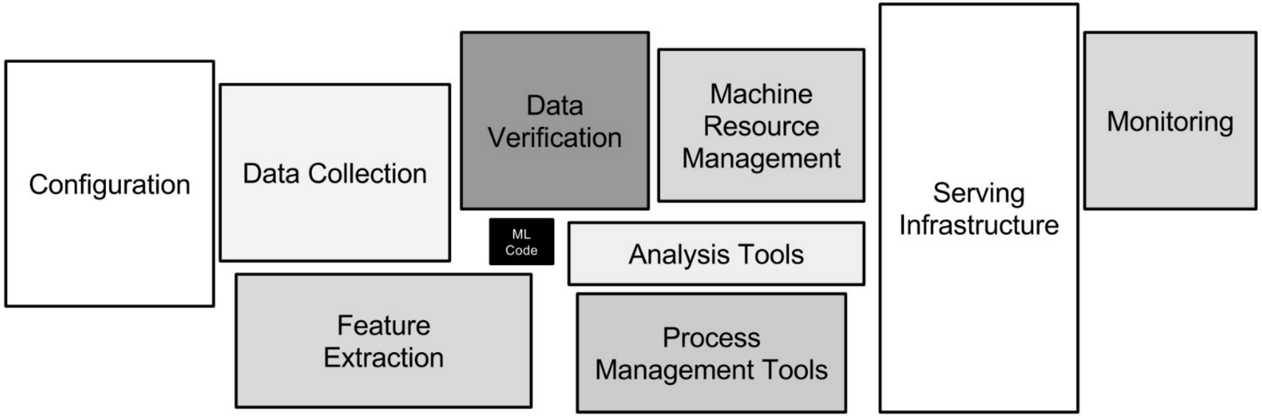
positive float

Source: https://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html

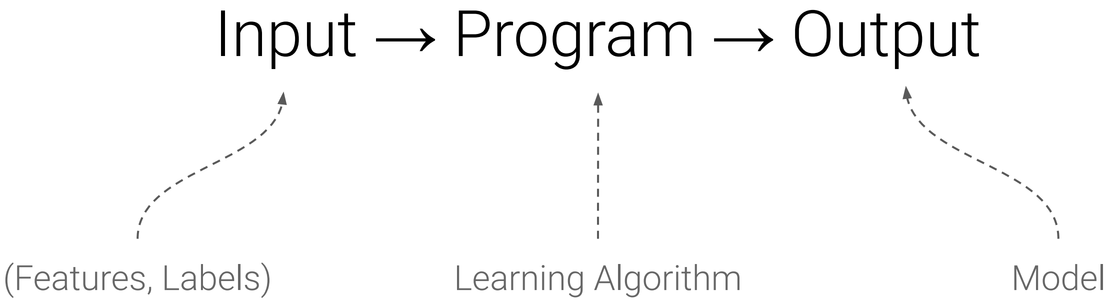
Enforcing and **maintaining** data quality is challenging

Production machine learning has a *complexity* problem

How do I know if these components are compatible?



source: https://proceedings.neurips.cc/paper/2015/file/86df7dcfd896fcaf2674f757a2463eba-Paper.pdf

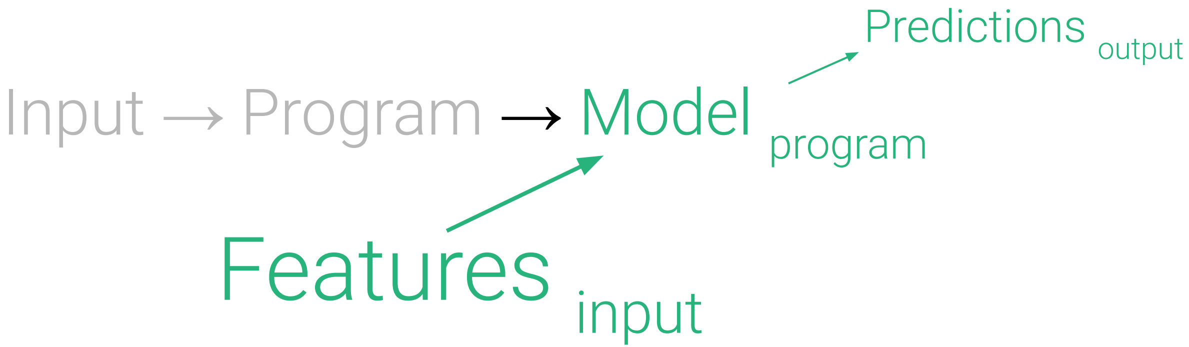


Input → Program → Output

(Features, Labels)

Learning Algorithm

Model



Input → Program → Model program

Predictions output

Features input

Strongly-typed interfaces unlock static analysis capabilities that push many potential errors from the *runtime context* into the *compile-time context*.

Reliability

Readability: as a human being 👫 or machine 🤖, I can tell what a component needs as input and what it produces as output.

Reproducibility: when a component fails 💥 at its input/output boundaries, I can be more confident that I can reproduce the error 🐞.

Efficiency

Caching: if I want to determine whether I should hit the cache 🎒 or re-compute 🔁 the result of a component, I can first check for changes in a function’s type signature before checking actual input values.

Parallelization: before I try to concurrently apply functions to a collection of inputs 🛍️🛍️🛍️, I can be confident that the elements in the collection are of the correct type.

Auditability

Debugging: When a pipeline execution fails 💥, I can pinpoint the cause of the error quickly and understand how to address it.

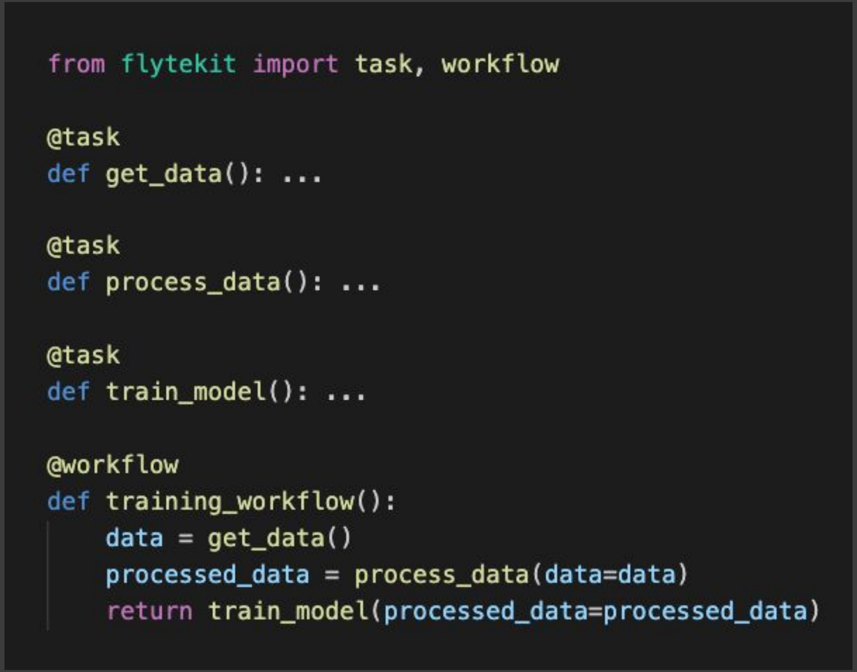
Data Lineage: I can understand how some downstream artifact 📦 came to be by looking at the upstream processes 🏭 that produced it.

Flyte is a *data*- and *machine-learning*-aware **orchestration tool** with **type-safety** built into multiple layers of the software stack.

Flyte

Easily Compose Workflows 🔀 using Tasks as Building Blocks 🧱

```
pip install flytekit
```



```python
from flytekit import task, workflow

@task
def get_data(): ...

@task
def process_data(): ...

@task
def train_model(): ...

@workflow
def training_workflow():
    data = get_data()
    processed_data = process_data(data=data)
    return train_model(processed_data=processed_data)
```

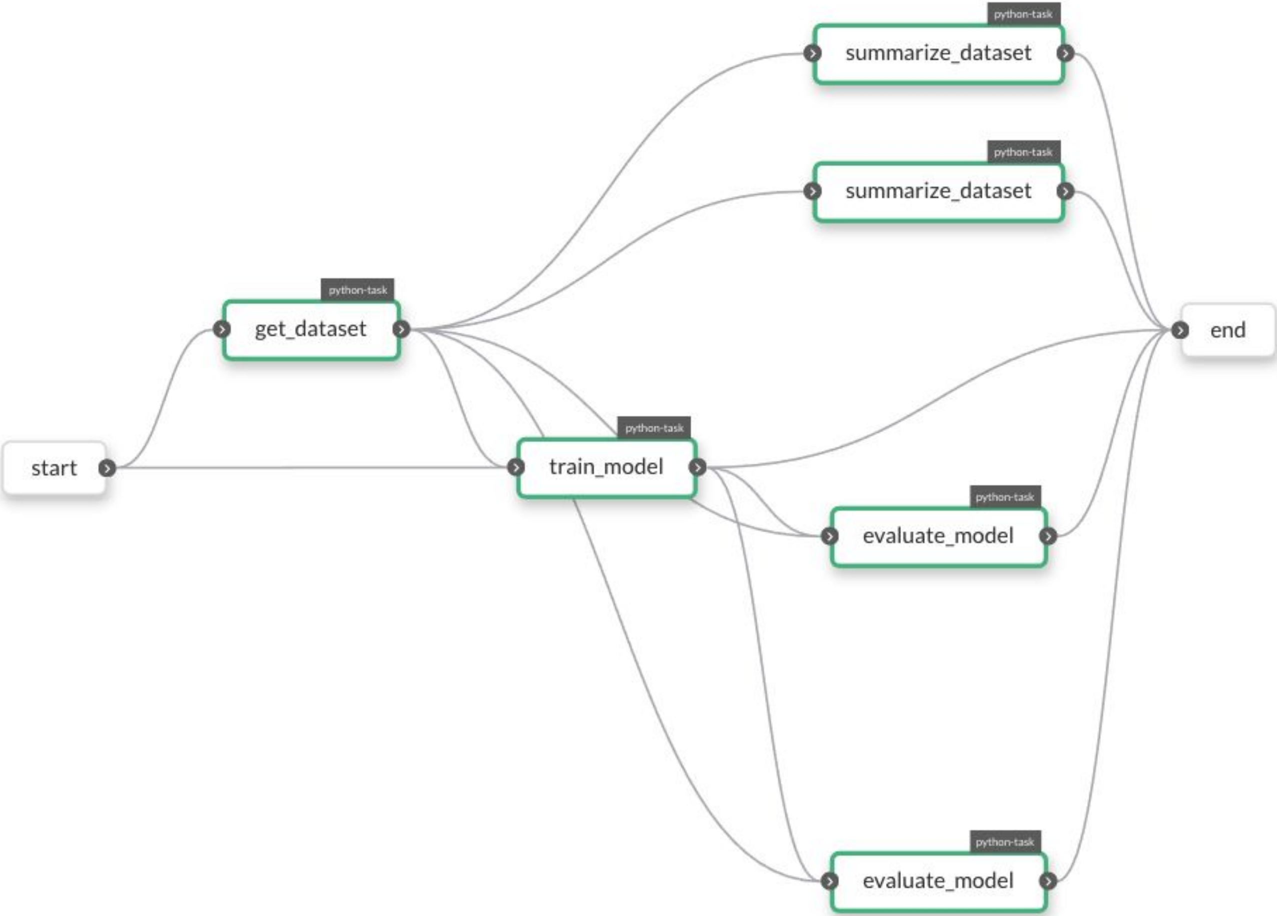
California House Price Regression

```
pandas.DataFrame({
    'Latitude': [37.88, …],
    'Longitude': [-122.23, …],
    'AveBedrms': [1.0238, …],
    'AveOccup': [2.5555, …],
    'AveRooms': [6.9841, …],
    'HouseAge': [41.0, …],
    'MedInc': [8.3252, …],
    'Population': [322.0, …],
    'MedHouseVal': [4.526, …],
})
```

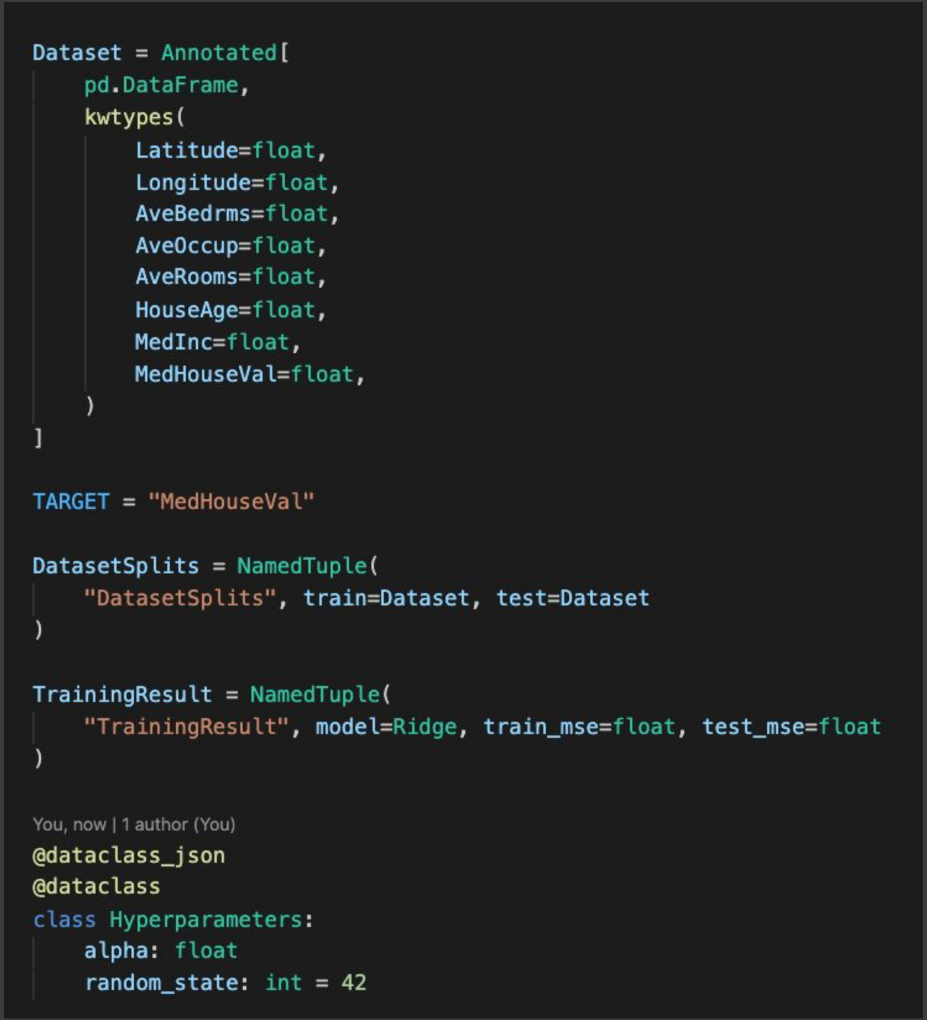
features

target

Pipeline Overview

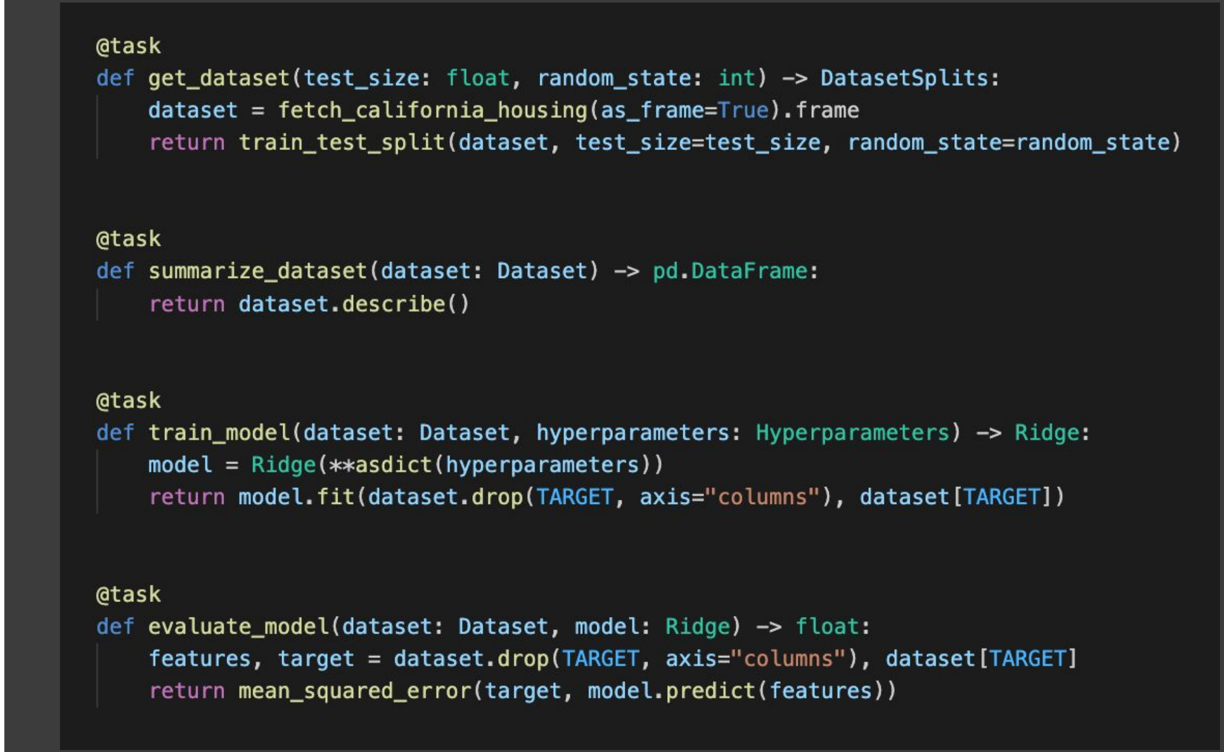


What Types are We Going to Use?



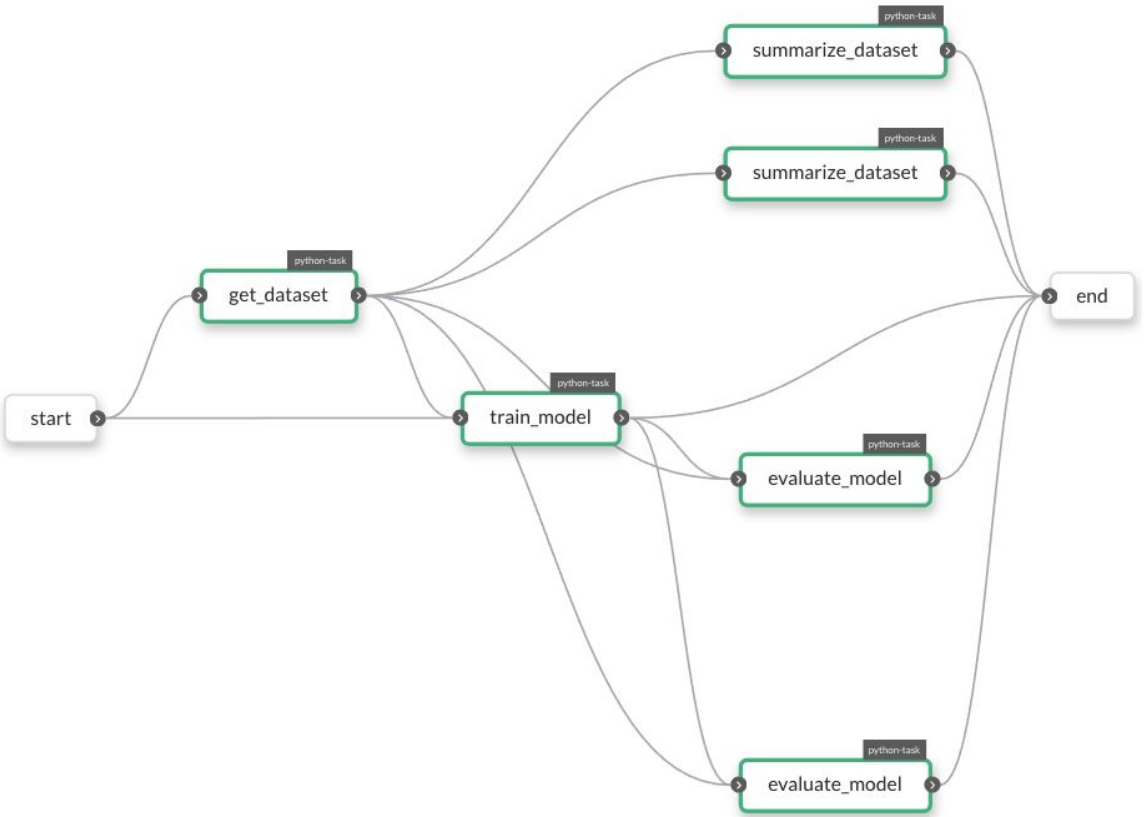
```
Dataset = Annotated[
    pd.DataFrame,
    kwtypes(
        Latitude=float,
        Longitude=float,
        AveBedrms=float,
        AveOccup=float,
        AveRooms=float,
        HouseAge=float,
        MedInc=float,
        MedHouseVal=float,
    )
]

TARGET = "MedHouseVal"

DatasetSplits = NamedTuple(
    "DatasetSplits", train=Dataset, test=Dataset
)

TrainingResult = NamedTuple(
    "TrainingResult", model=Ridge, train_mse=float, test_mse=float
)

You, now | 1 author (You)
@dataclass_json
@dataclass
class Hyperparameters:
    alpha: float
    random_state: int = 42
```

Tasks are 📦 Containerized Units of Work 🛠️ with a Transparent Interface

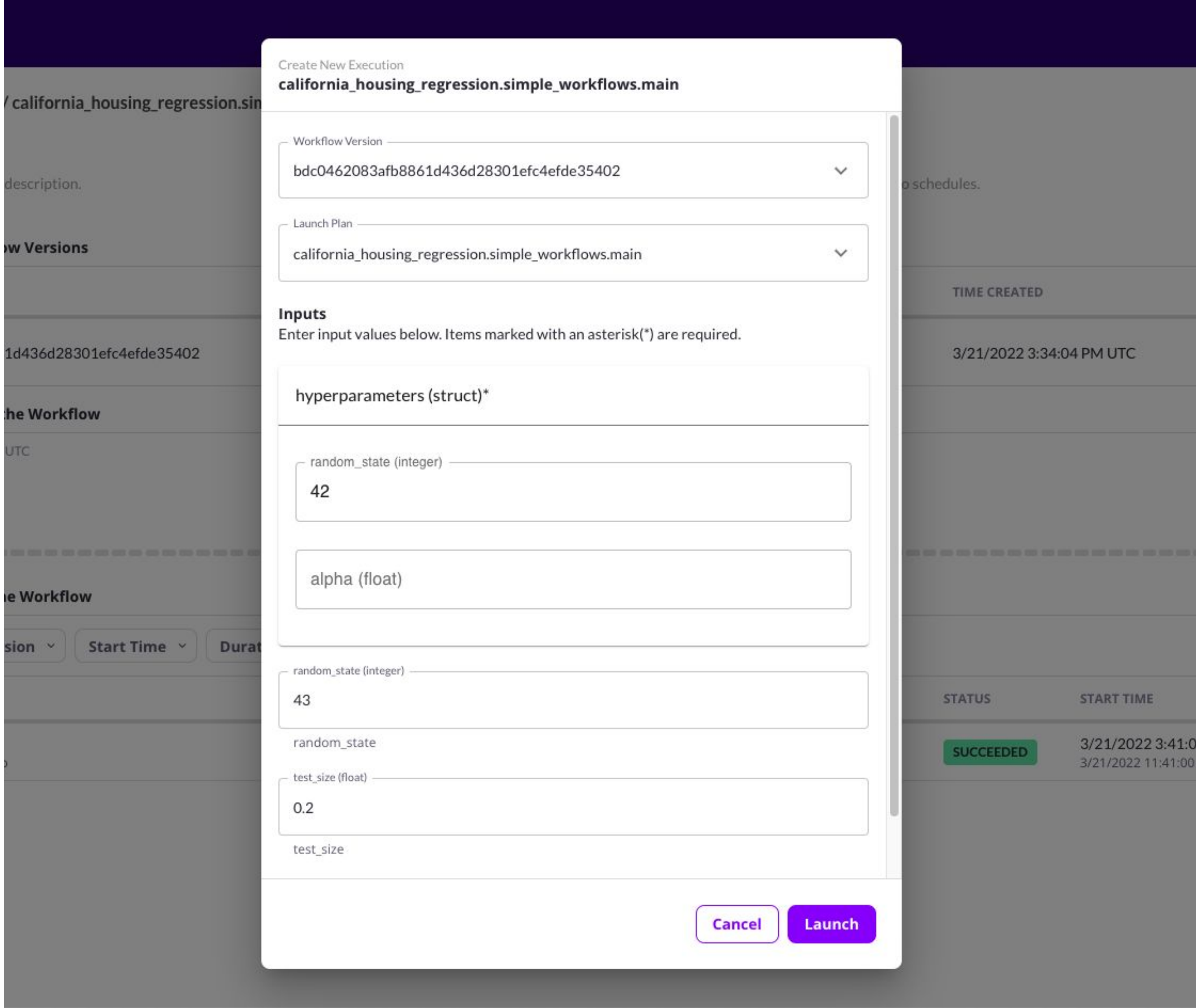


```python
@task
def get_dataset(test_size: float, random_state: int) -> DatasetSplits:
    dataset = fetch_california_housing(as_frame=True).frame
    return train_test_split(dataset, test_size=test_size, random_state=random_state)


@task
def summarize_dataset(dataset: Dataset) -> pd.DataFrame:
    return dataset.describe()


@task
def train_model(dataset: Dataset, hyperparameters: Hyperparameters) -> Ridge:
    model = Ridge(**asdict(hyperparameters))
    return model.fit(dataset.drop(TARGET, axis="columns"), dataset[TARGET])


@task
def evaluate_model(dataset: Dataset, model: Ridge) -> float:
    features, target = dataset.drop(TARGET, axis="columns"), dataset[TARGET]
    return mean_squared_error(target, model.predict(features))
```

Workflows are Dynamic DAGs that Compose Tasks Together to do Something Useful 🏗️



Auto-generate Strongly Typed Launch Forms 📝



Create New Execution

california_housing_regression.simple_workflows.main

Workflow Version: bdc0462083afb8861d436d28301efc4efde35402

Launch Plan: california_housing_regression.simple_workflows.main

Inputs

Enter input values below. Items marked with an asterisk(*) are required.

hyperparameters (struct)*

random_state (integer): 42

alpha (float)

random_state (integer): 43

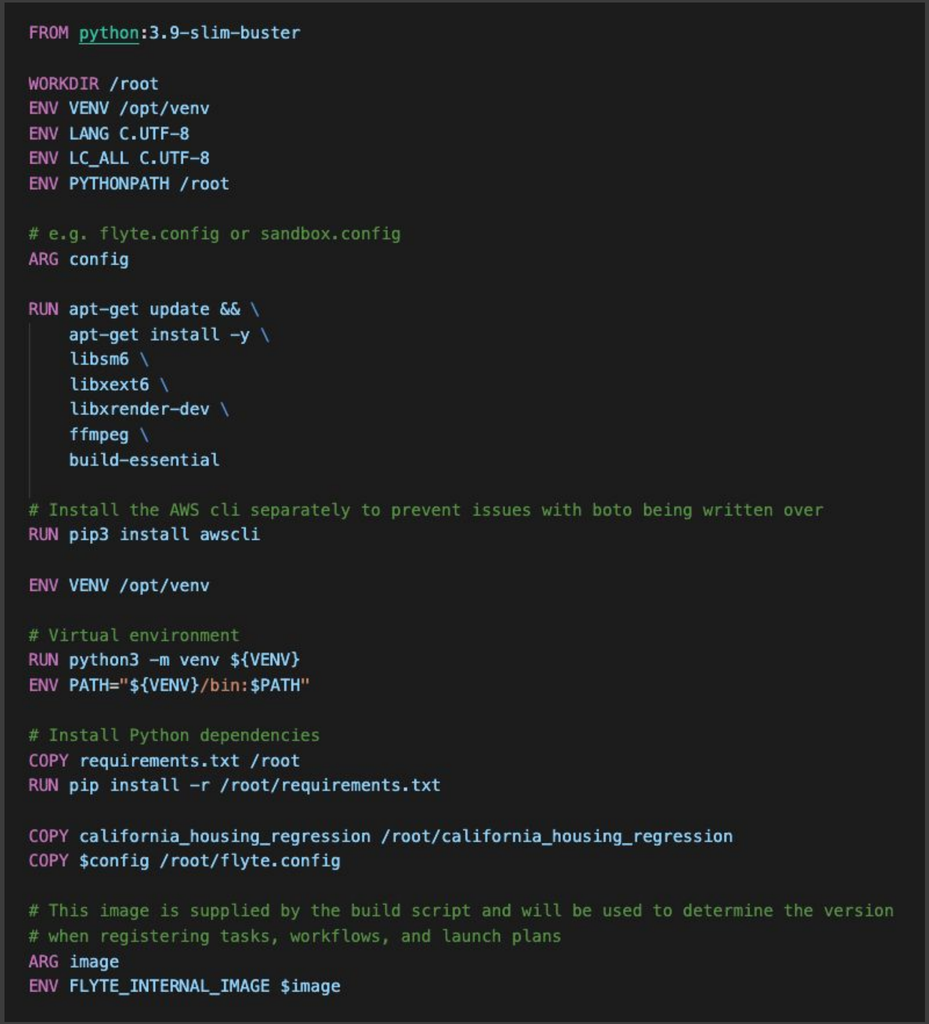
random_state

test_size (float): 0.2

test_size

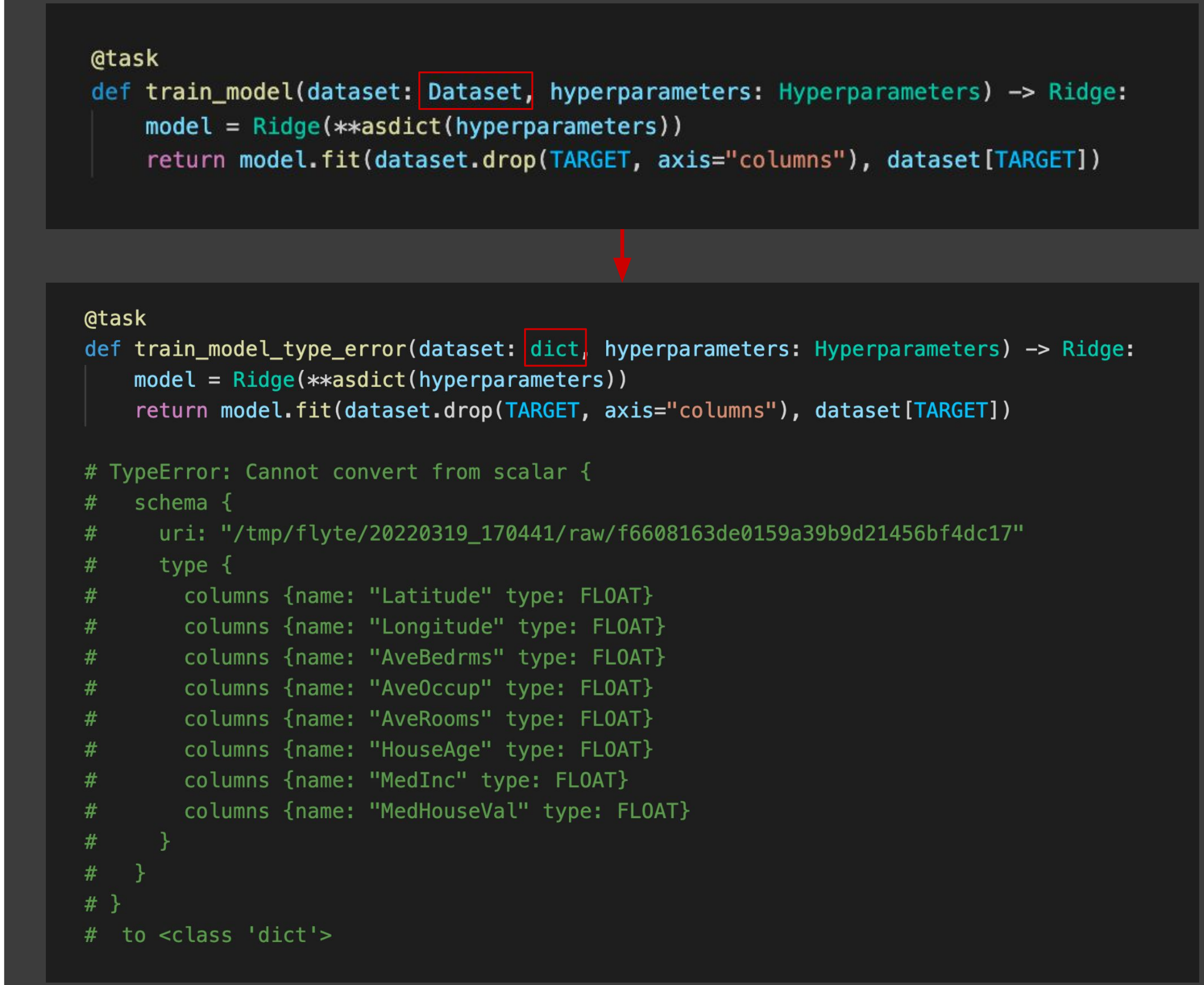
Docker 🐳 Guarantees Reproducibility

…as long as tasks are idempotent



```
FROM python:3.9-slim-buster

WORKDIR /root
ENV VENV /opt/venv
ENV LANG C.UTF-8
ENV LC_ALL C.UTF-8
ENV PYTHONPATH /root

# e.g. flyte.config or sandbox.config
ARG config

RUN apt-get update && \
    apt-get install -y \
    libsm6 \
    libxext6 \
    libxrender-dev \
    ffmpeg \
    build-essential

# Install the AWS cli separately to prevent issues with boto being written over
RUN pip3 install awscli

ENV VENV /opt/venv

# Virtual environment
RUN python3 -m venv ${VENV}
ENV PATH="${VENV}/bin:$PATH"

# Install Python dependencies
COPY requirements.txt /root
RUN pip install -r /root/requirements.txt

COPY california_housing_regression /root/california_housing_regression
COPY $config /root/flyte.config

# This image is supplied by the build script and will be used to determine the version
# when registering tasks, workflows, and launch plans
ARG image
ENV FLYTE_INTERNAL_IMAGE $image
```

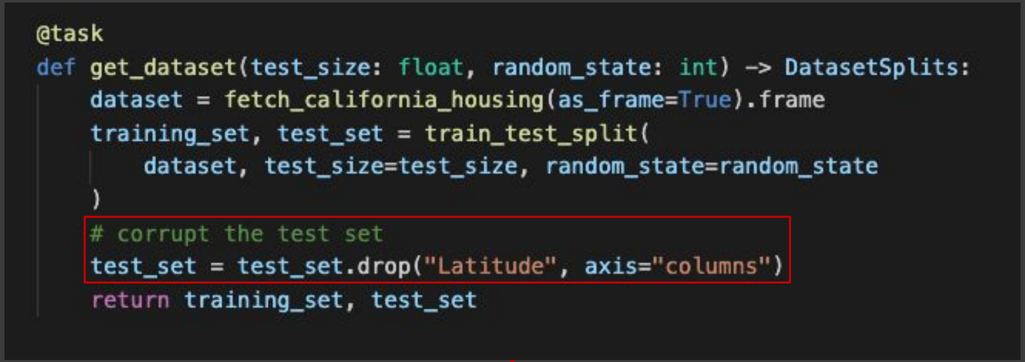
Flyte Statically Analyzes 🔍 the DAG to catch Type Errors



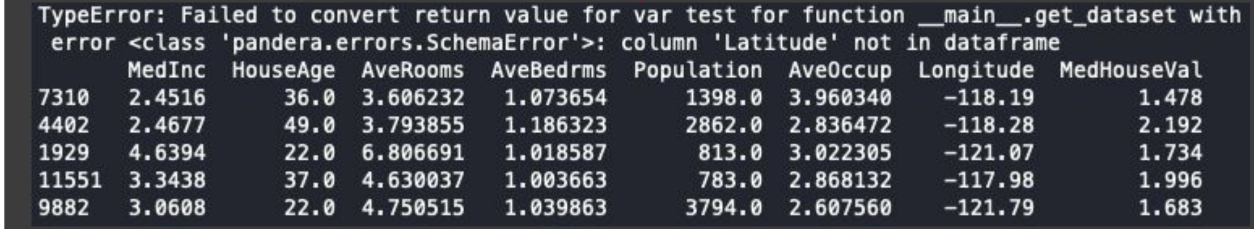
```
@task
def train_model(dataset: Dataset, hyperparameters: Hyperparameters) -> Ridge:
    model = Ridge(**asdict(hyperparameters))
    return model.fit(dataset.drop(TARGET, axis="columns"), dataset[TARGET])
```

```
@task
def train_model_type_error(dataset: dict, hyperparameters: Hyperparameters) -> Ridge:
    model = Ridge(**asdict(hyperparameters))
    return model.fit(dataset.drop(TARGET, axis="columns"), dataset[TARGET])

# TypeError: Cannot convert from scalar {
#   schema {
#     uri: "/tmp/flyte/20220319_170441/raw/f6608163de0159a39b9d21456bf4dc17"
#     type {
#       columns {name: "Latitude" type: FLOAT}
#       columns {name: "Longitude" type: FLOAT}
#       columns {name: "AveBedrms" type: FLOAT}
#       columns {name: "AveOccup" type: FLOAT}
#       columns {name: "AveRooms" type: FLOAT}
#       columns {name: "HouseAge" type: FLOAT}
#       columns {name: "MedInc" type: FLOAT}
#       columns {name: "MedHouseVal" type: FLOAT}
#     }
#   }
# }
#  to <class 'dict'>
```

Catch Value Errors 🐞 When Testing Locally

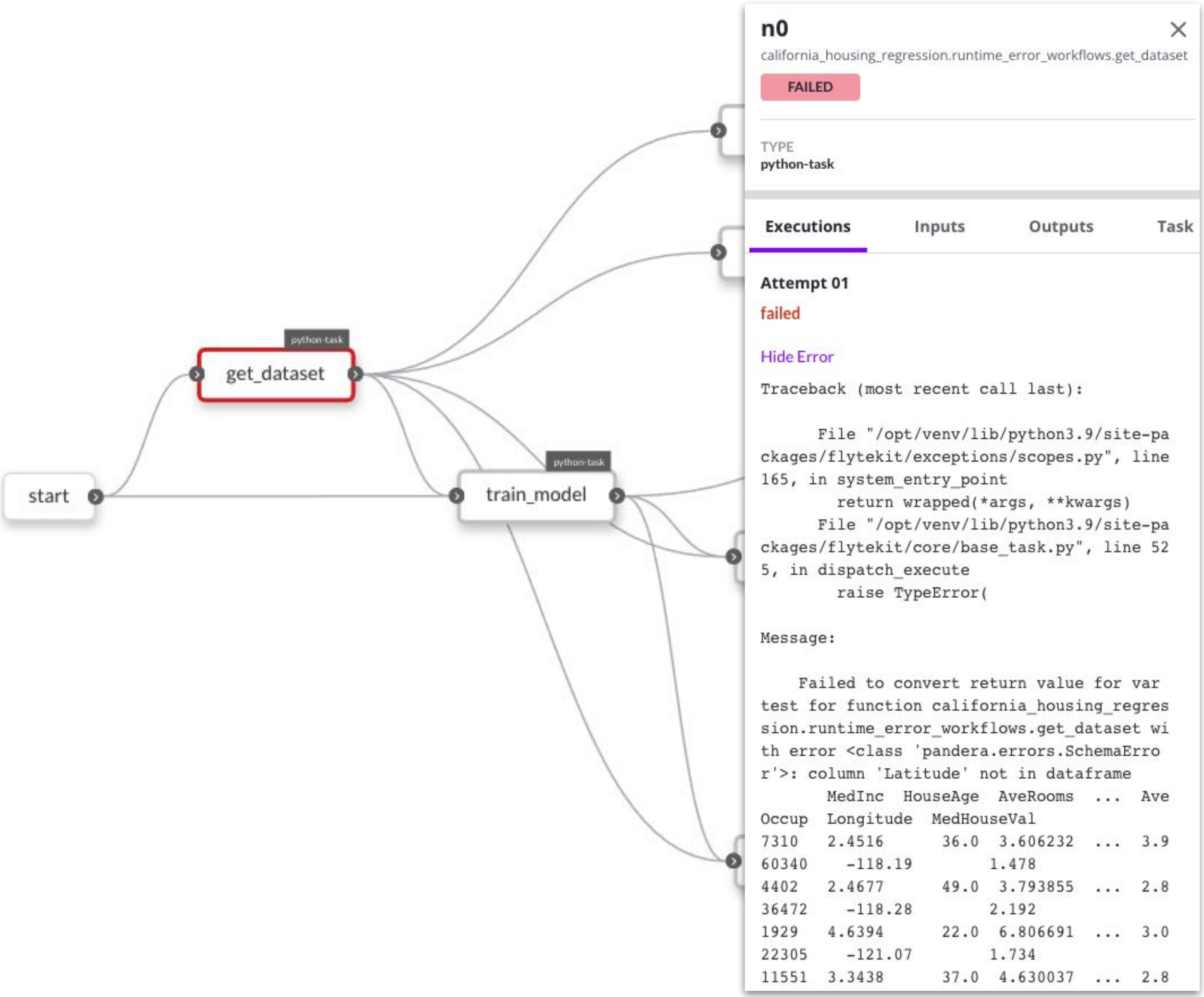


```python
@task
def get_dataset(test_size: float, random_state: int) -> DatasetSplits:
    dataset = fetch_california_housing(as_frame=True).frame
    training_set, test_set = train_test_split(
        dataset, test_size=test_size, random_state=random_state
    )
    # corrupt the test set
    test_set = test_set.drop("Latitude", axis="columns")
    return training_set, test_set
```



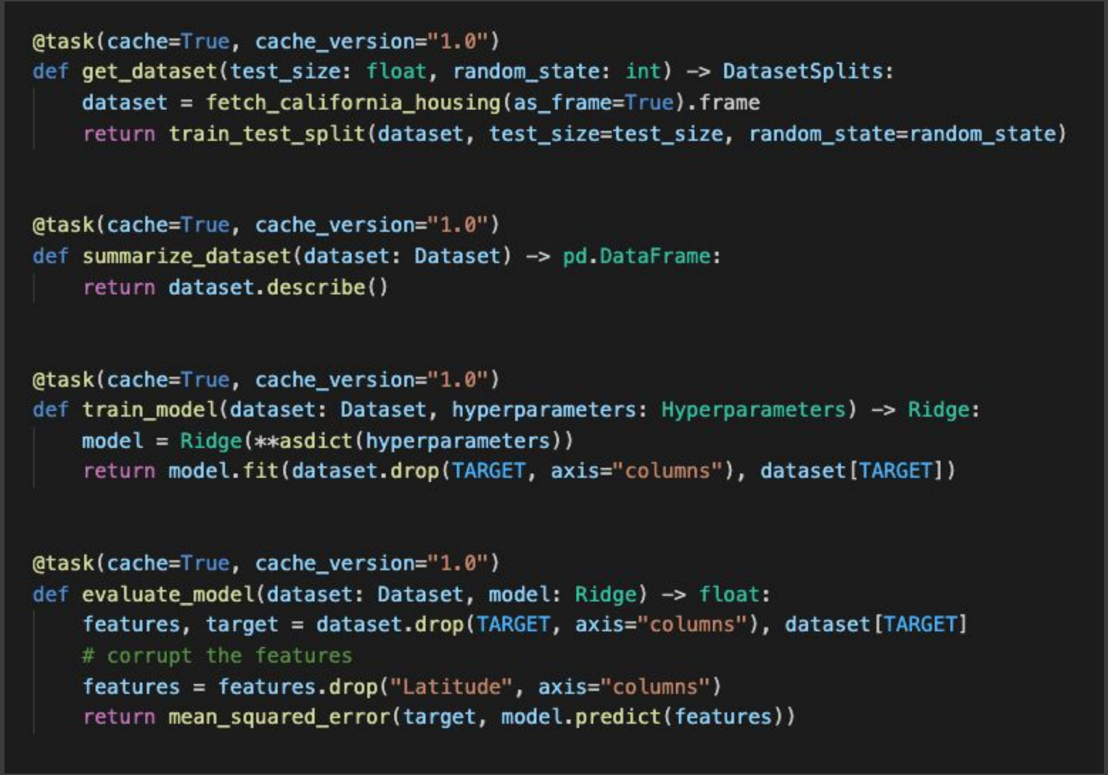
```
TypeError: Failed to convert return value for var test for function __main__.get_dataset with
 error <class 'pandera.errors.SchemaError'>: column 'Latitude' not in dataframe
       MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup  Longitude  MedHouseVal
7310   2.4516      36.0  3.606232   1.073654      1398.0  3.960340    -118.19        1.478
4402   2.4677      49.0  3.793855   1.186323      2862.0  2.836472    -118.28        2.192
1929   4.6394      22.0  6.806691   1.018587       813.0  3.022305    -121.07        1.734
11551  3.3438      37.0  4.630037   1.003663       783.0  2.868132    -117.98        1.996
9882   3.0608      22.0  4.750515   1.039863      3794.0  2.607560    -121.79        1.683
```

Know Where your Pipeline Blew Up 🧨



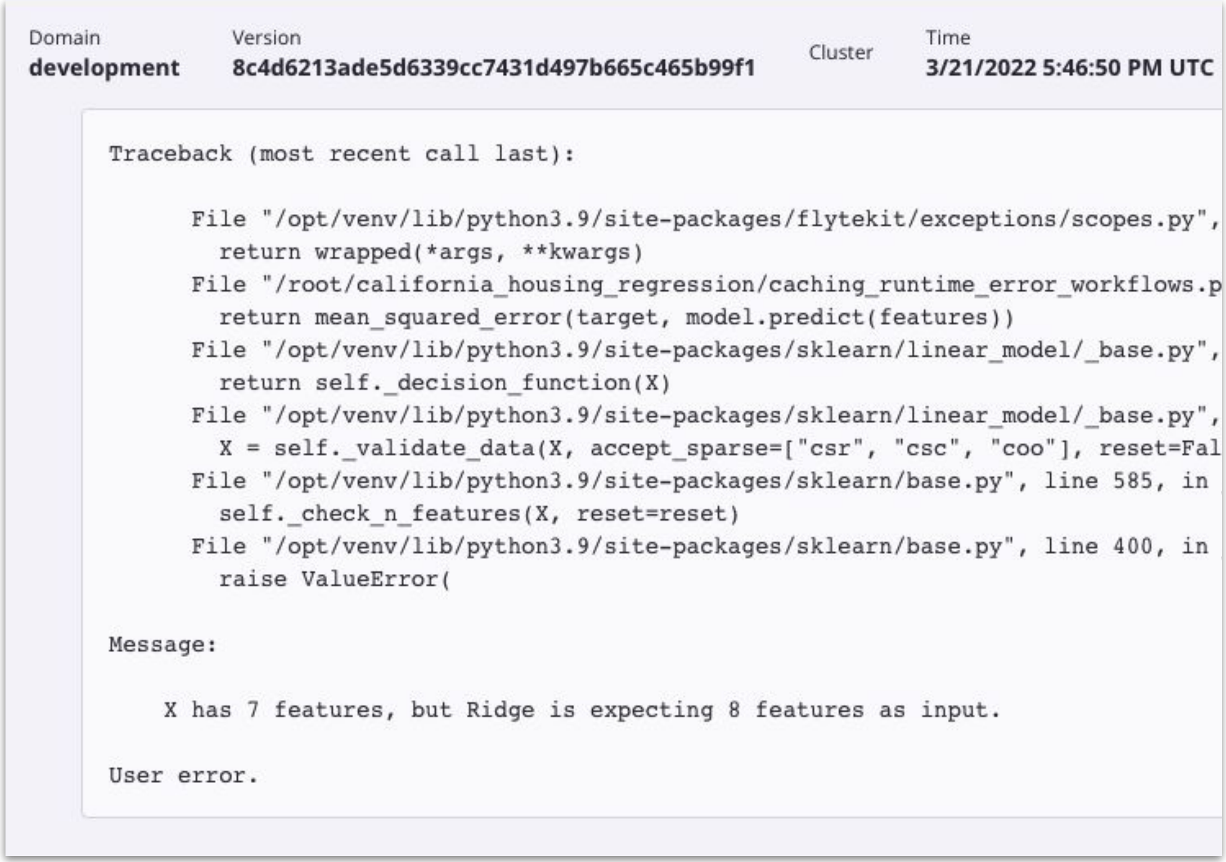
```
Traceback (most recent call last):

      File "/opt/venv/lib/python3.9/site-pa
ckages/flytekit/exceptions/scopes.py", line
165, in system_entry_point
        return wrapped(*args, **kwargs)
      File "/opt/venv/lib/python3.9/site-pa
ckages/flytekit/core/base_task.py", line 52
5, in dispatch_execute
        raise TypeError(

Message:

    Failed to convert return value for var
test for function california_housing_regres
sion.runtime_error_workflows.get_dataset wi
th error <class 'pandera.errors.SchemaErro
r'>: column 'Latitude' not in dataframe
       MedInc  HouseAge  AveRooms  ...  Ave
Occup  Longitude  MedHouseVal
7310   2.4516      36.0  3.606232  ...  3.9
60340    -118.19        1.478
4402   2.4677      49.0  3.793855  ...  2.8
36472    -118.28        2.192
1929   4.6394      22.0  6.806691  ...  3.0
22305    -121.07        1.734
11551  3.3438      37.0  4.630037  ...  2.8
```

Cache the Outputs of a Task 🎒



```python
@task(cache=True, cache_version="1.0")
def get_dataset(test_size: float, random_state: int) -> DatasetSplits:
    dataset = fetch_california_housing(as_frame=True).frame
    return train_test_split(dataset, test_size=test_size, random_state=random_state)


@task(cache=True, cache_version="1.0")
def summarize_dataset(dataset: Dataset) -> pd.DataFrame:
    return dataset.describe()


@task(cache=True, cache_version="1.0")
def train_model(dataset: Dataset, hyperparameters: Hyperparameters) -> Ridge:
    model = Ridge(**asdict(hyperparameters))
    return model.fit(dataset.drop(TARGET, axis="columns"), dataset[TARGET])


@task(cache=True, cache_version="1.0")
def evaluate_model(dataset: Dataset, model: Ridge) -> float:
    features, target = dataset.drop(TARGET, axis="columns"), dataset[TARGET]
    # corrupt the features
    features = features.drop("Latitude", axis="columns")
    return mean_squared_error(target, model.predict(features))
```

Errors at the End of a Long-running Training Pipeline got you Down 😓?



Domain
development

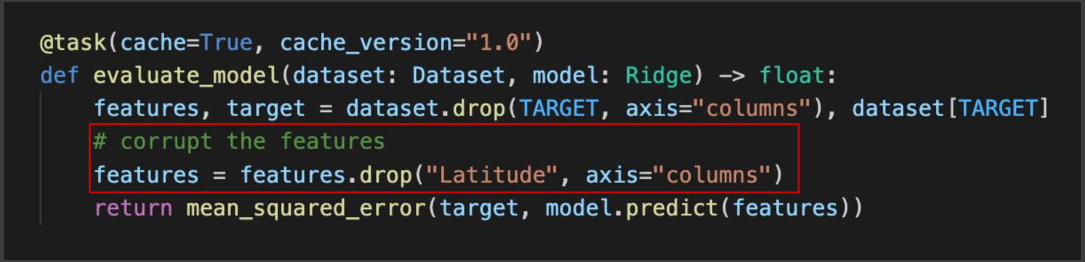
Version
8c4d6213ade5d6339cc7431d497b665c465b99f1

Cluster

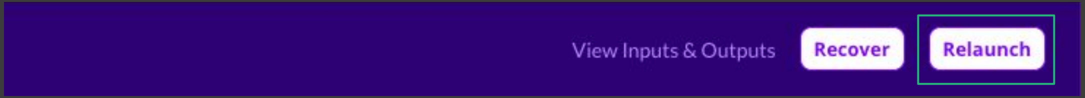
Time
3/21/2022 5:46:50 PM UTC

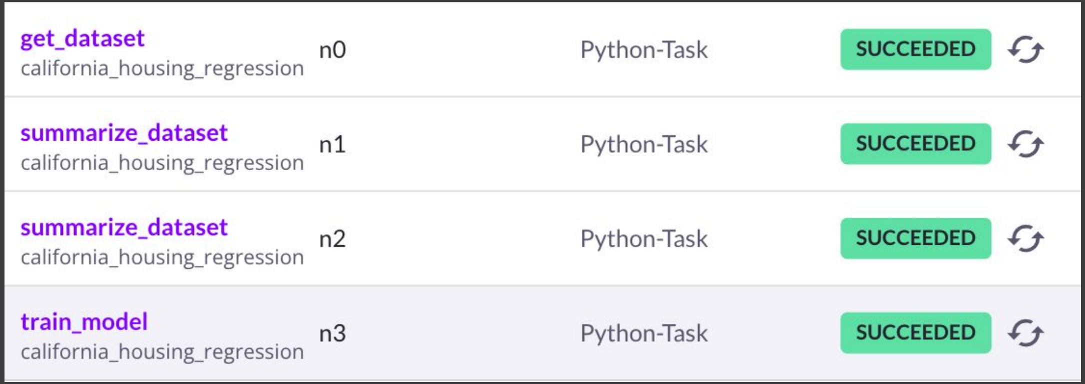
```
Traceback (most recent call last):

        File "/opt/venv/lib/python3.9/site-packages/flytekit/exceptions/scopes.py",
          return wrapped(*args, **kwargs)
        File "/root/california_housing_regression/caching_runtime_error_workflows.p
          return mean_squared_error(target, model.predict(features))
        File "/opt/venv/lib/python3.9/site-packages/sklearn/linear_model/_base.py",
          return self._decision_function(X)
        File "/opt/venv/lib/python3.9/site-packages/sklearn/linear_model/_base.py",
          X = self._validate_data(X, accept_sparse=["csr", "csc", "coo"], reset=Fal
        File "/opt/venv/lib/python3.9/site-packages/sklearn/base.py", line 585, in
          self._check_n_features(X, reset=reset)
        File "/opt/venv/lib/python3.9/site-packages/sklearn/base.py", line 400, in
          raise ValueError(

Message:

    X has 7 features, but Ridge is expecting 8 features as input.

User error.
```

Don’t Re-compute, Hit the Cache! 🤜 🎒



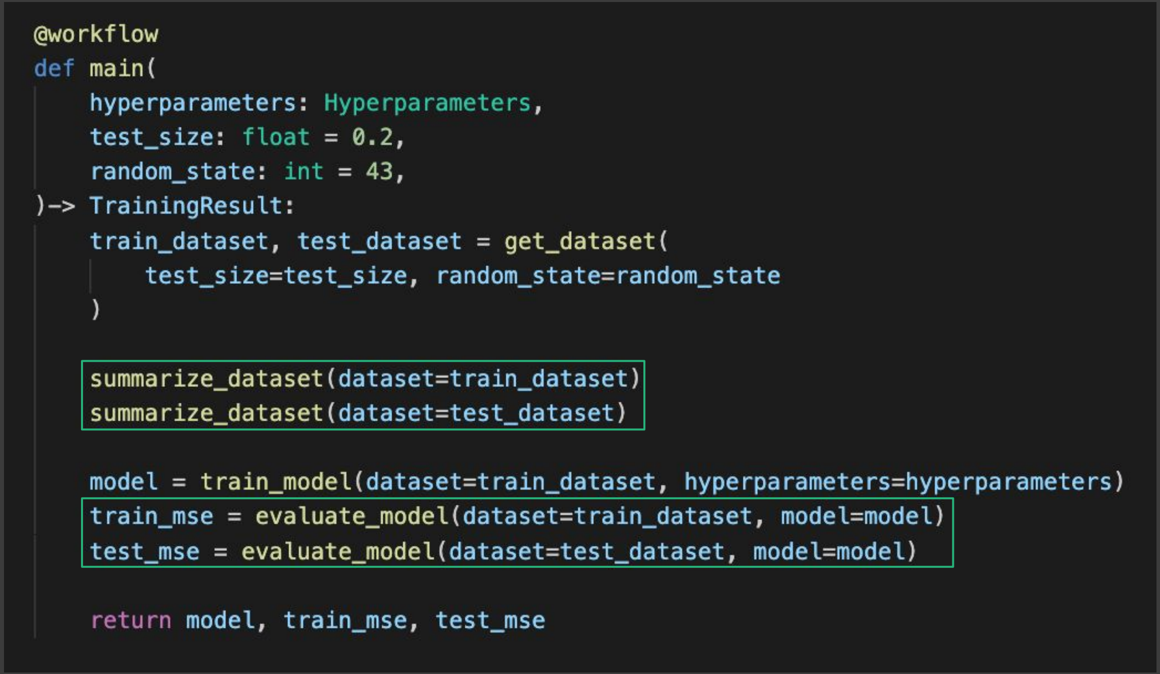
```python
@task(cache=True, cache_version="1.0")
def evaluate_model(dataset: Dataset, model: Ridge) -> float:
    features, target = dataset.drop(TARGET, axis="columns"), dataset[TARGET]
    # corrupt the features
    features = features.drop("Latitude", axis="columns")
    return mean_squared_error(target, model.predict(features))
```





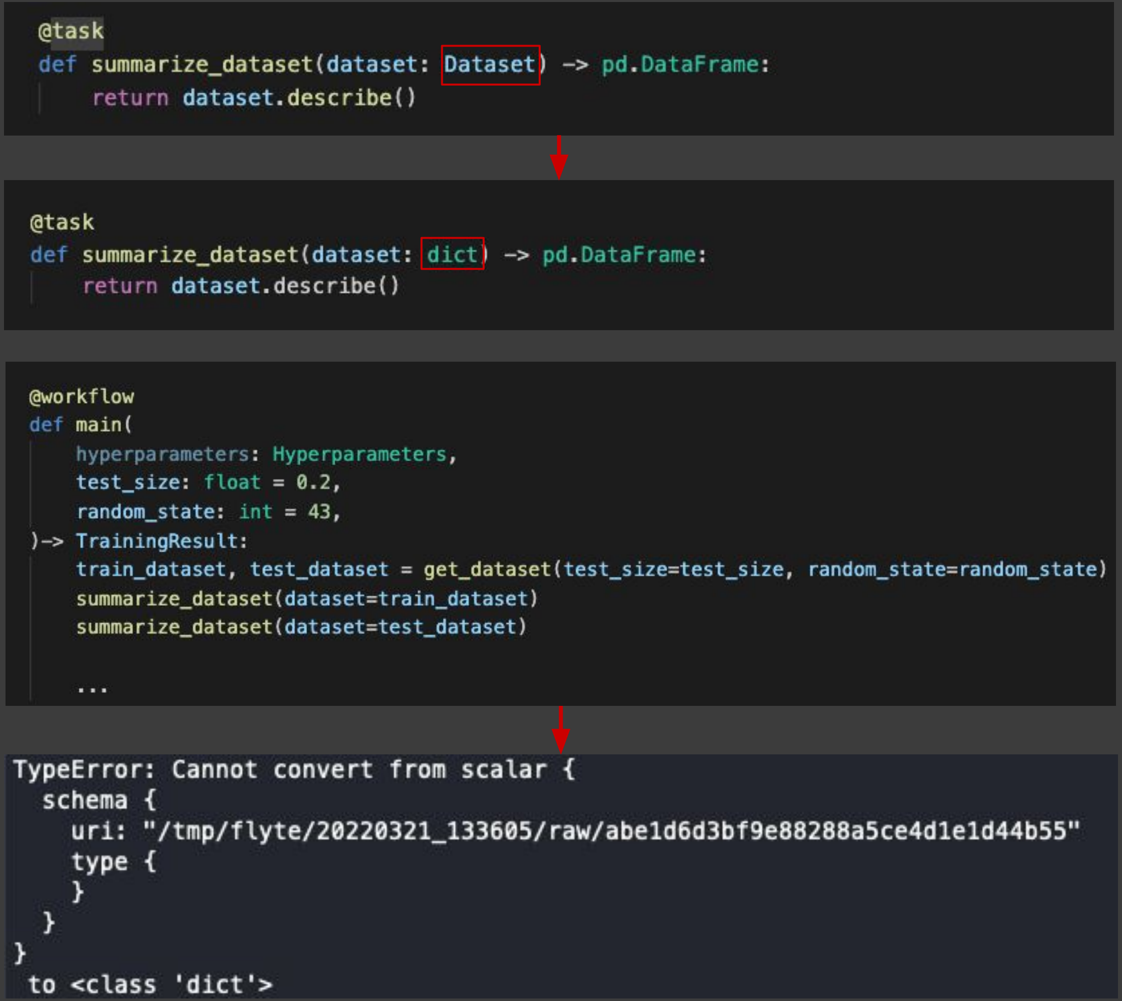
get_dataset california_housing_regression	n0	Python-Task	SUCCEEDED
summarize_dataset california_housing_regression	n1	Python-Task	SUCCEEDED
summarize_dataset california_housing_regression	n2	Python-Task	SUCCEEDED
train_model california_housing_regression	n3	Python-Task	SUCCEEDED

Workflows Execute Tasks with Built-in Parallelism 🔀



```
@workflow
def main(
    hyperparameters: Hyperparameters,
    test_size: float = 0.2,
    random_state: int = 43,
)-> TrainingResult:
    train_dataset, test_dataset = get_dataset(
        test_size=test_size, random_state=random_state
    )

    summarize_dataset(dataset=train_dataset)
    summarize_dataset(dataset=test_dataset)

    model = train_model(dataset=train_dataset, hyperparameters=hyperparameters)
    train_mse = evaluate_model(dataset=train_dataset, model=model)
    test_mse = evaluate_model(dataset=test_dataset, model=model)

    return model, train_mse, test_mse
```

Static Type Checking 🔍 Applies to Parallelized Invocations of a Task



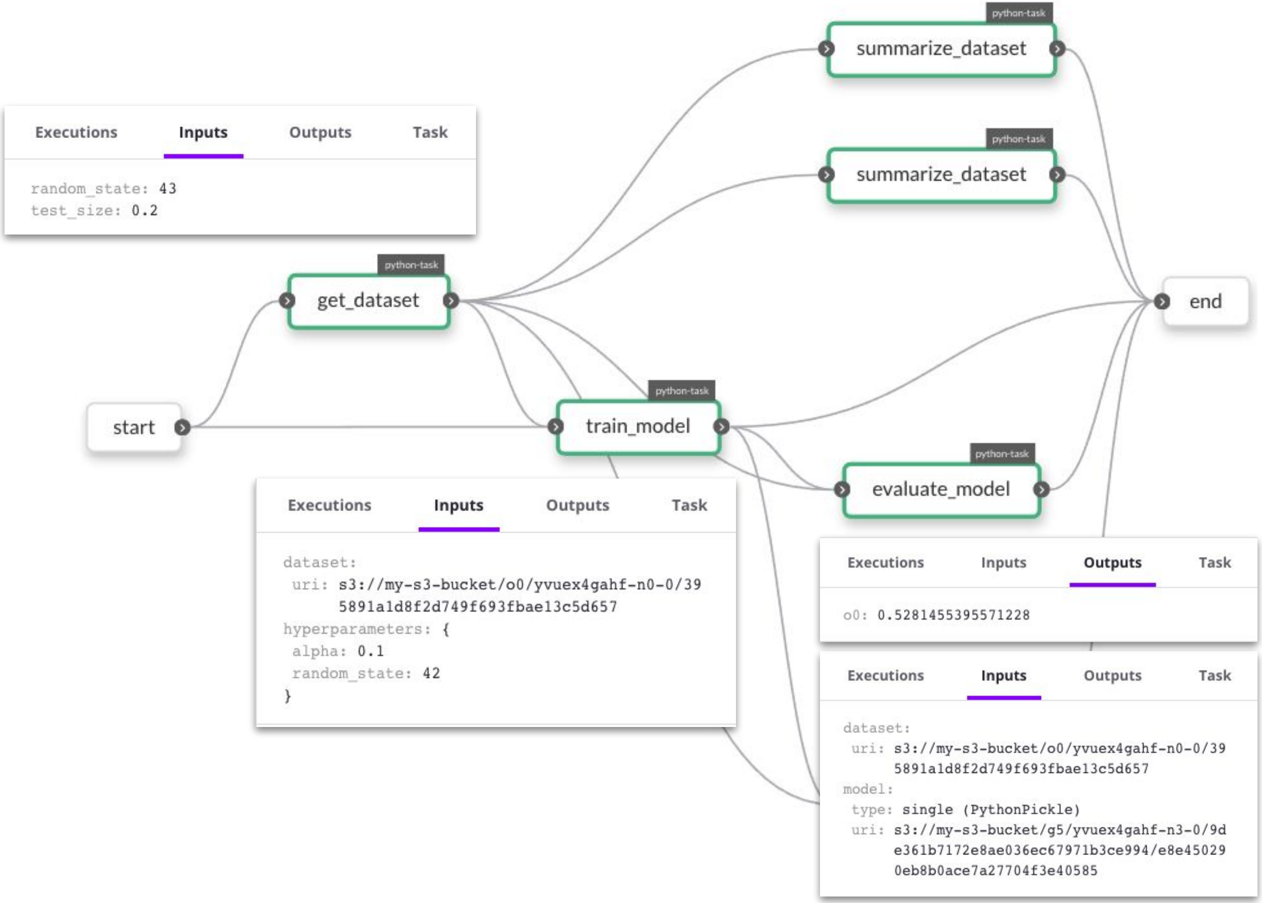
```python
@task
def summarize_dataset(dataset: Dataset) -> pd.DataFrame:
    return dataset.describe()
```

```python
@task
def summarize_dataset(dataset: dict) -> pd.DataFrame:
    return dataset.describe()
```

```python
@workflow
def main(
    hyperparameters: Hyperparameters,
    test_size: float = 0.2,
    random_state: int = 43,
)-> TrainingResult:
    train_dataset, test_dataset = get_dataset(test_size=test_size, random_state=random_state)
    summarize_dataset(dataset=train_dataset)
    summarize_dataset(dataset=test_dataset)

    ...
```

```
TypeError: Cannot convert from scalar {
  schema {
    uri: "/tmp/flyte/20220321_133605/raw/abe1d6d3bf9e88288a5ce4d1e1d44b55"
    type {
    }
  }
}
 to <class 'dict'>
```

Trace Model Artifacts to the Data and Downstream Processes that Produced it



```
random_state: 43
test_size: 0.2
```

```
dataset:
 uri: s3://my-s3-bucket/o0/yvuex4gahf-n0-0/39
      5891a1d8f2d749f693fbae13c5d657
hyperparameters: {
 alpha: 0.1
 random_state: 42
}
```

```
o0: 0.5281455395571228
```

```
dataset:
 uri: s3://my-s3-bucket/o0/yvuex4gahf-n0-0/39
      5891a1d8f2d749f693fbae13c5d657
model:
 type: single (PythonPickle)
 uri: s3://my-s3-bucket/g5/yvuex4gahf-n3-0/9d
      e361b7172e8ae036ec67971b3ce994/e8e45029
      0eb8b0ace7a27704f3e40585
```

But wait, what about **data types** for *machine learning*?

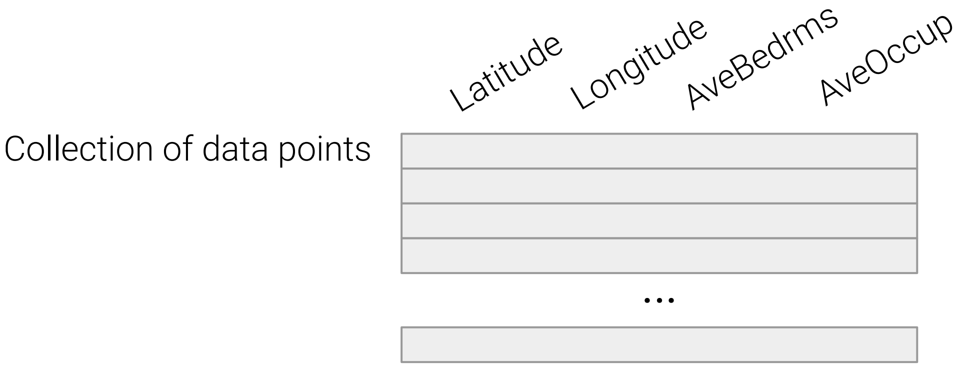
Pandera is a **statistical typing** and **data testing** library for *dataframes*, providing tools for defining *complex data types* and *unit testing* your pipelines with them.

Statistical Typing: Specifying the properties of collections of data points

Latitude
Longitude
AveBedrms
AveOccup
Single data point

- Primitive data types
- Value range
- Allowable values
- Regex string match
- Nullability

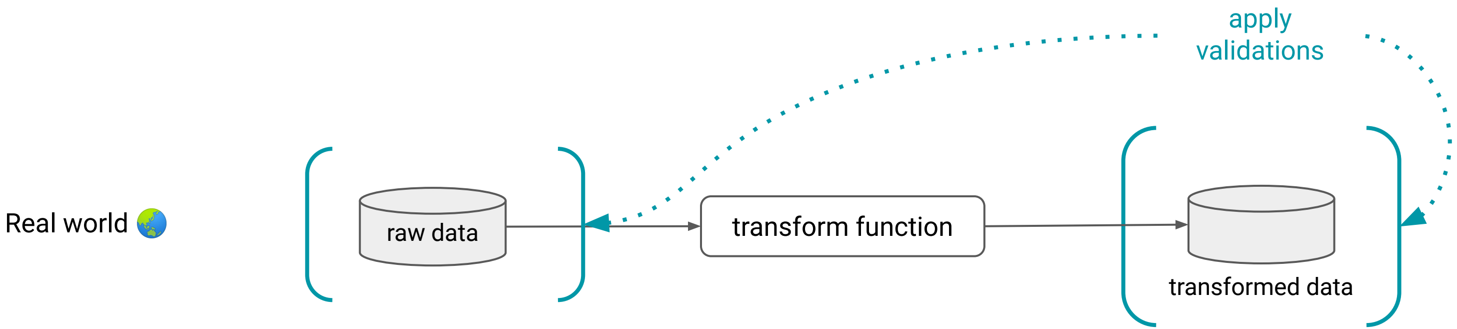
Statistical Typing: Specifying the properties of collections of data points



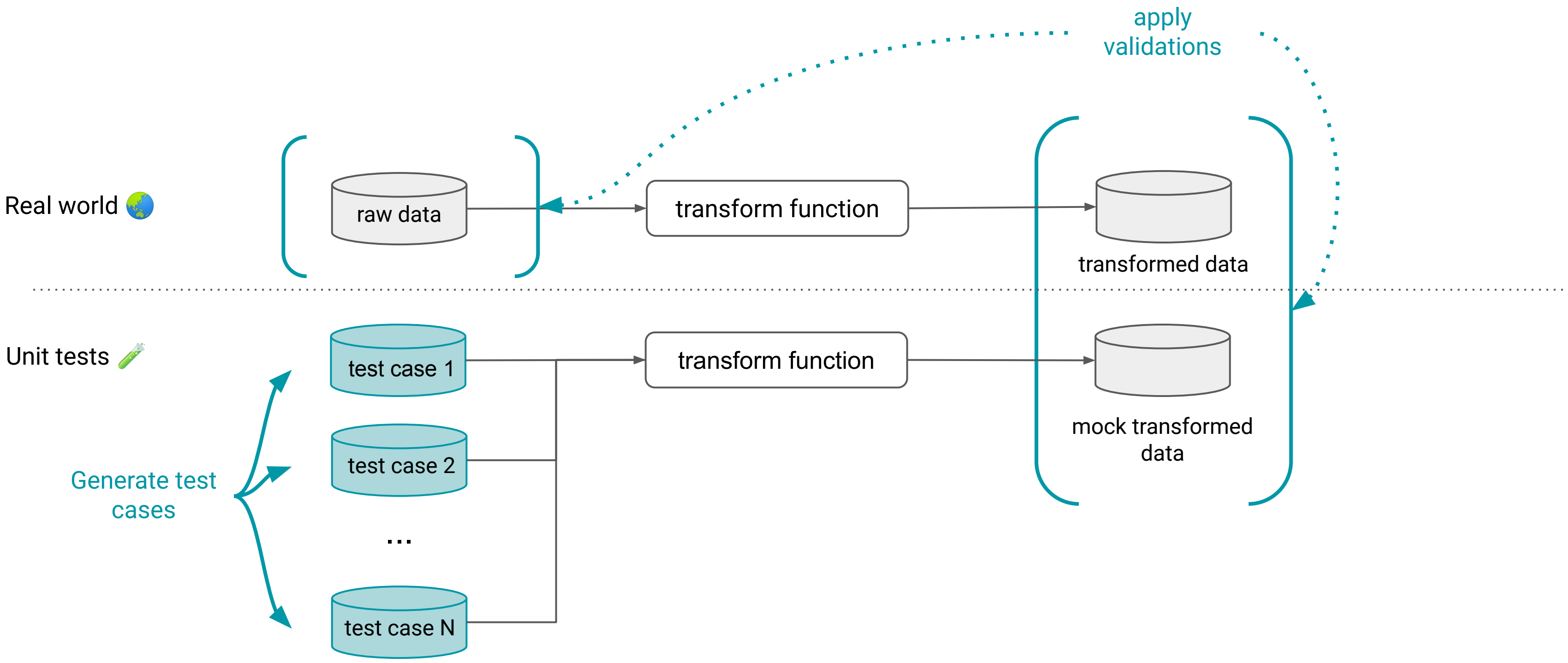
- Apply atomic checks at scale
- Uniqueness
- Monotonicity
- Mean, median, standard deviation
- Statistical distributions
- Fractional checks, e.g. “90% of data points are not null”

Statistical properties, by definition, can only be verified at *runtime*, but we can also define *functions* that use **statistical type annotations** that verify valid operations on those types.

Data Testing: Validating not only real data…



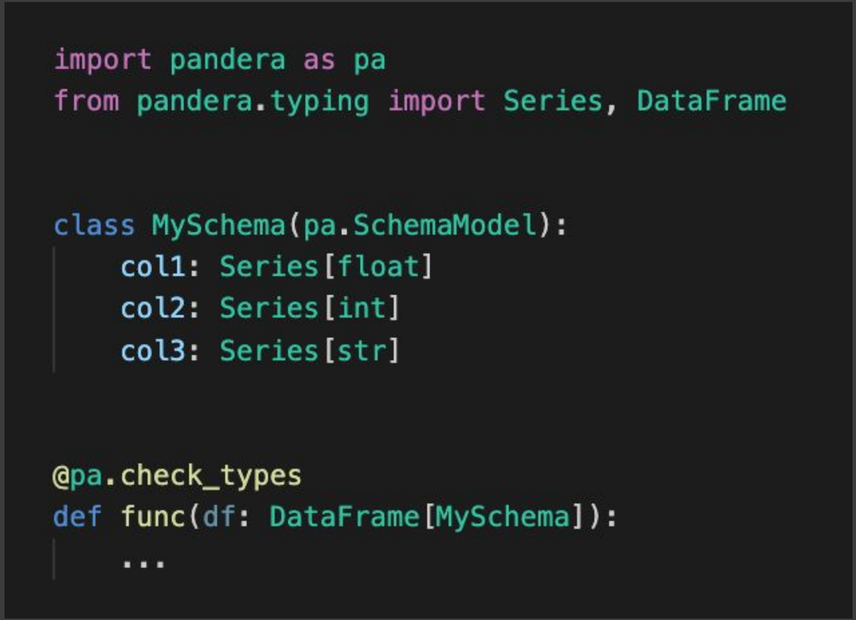
… but also the functions that produce them



Pandera

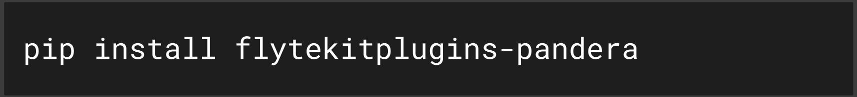
Define Statistical Types for your DataFrame-like Objects 📊🖼️

```
pip install pandera
```

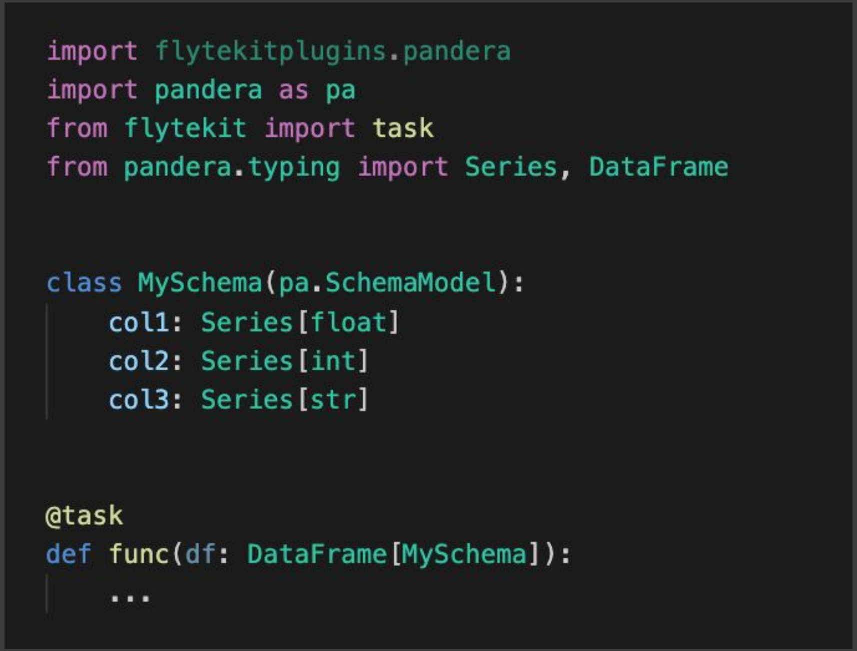


```
import pandera as pa
from pandera.typing import Series, DataFrame


class MySchema(pa.SchemaModel):
    col1: Series[float]
    col2: Series[int]
    col3: Series[str]


@pa.check_types
def func(df: DataFrame[MySchema]):
    ...
```

Pandera and **Flyte** Play Well Together 🤝

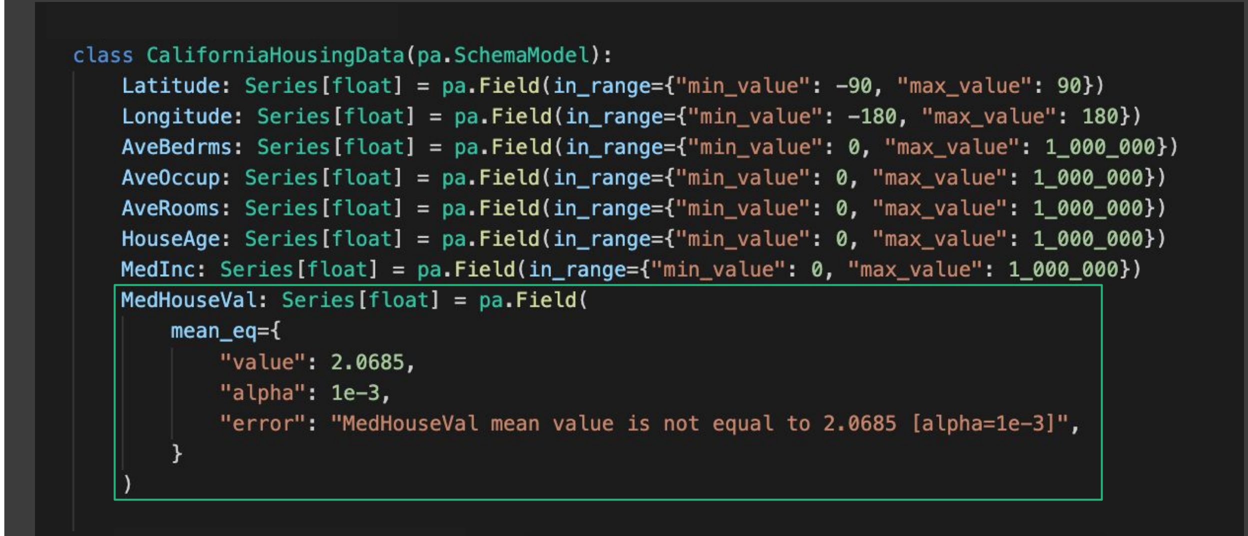


```
pip install flytekitplugins-pandera
```

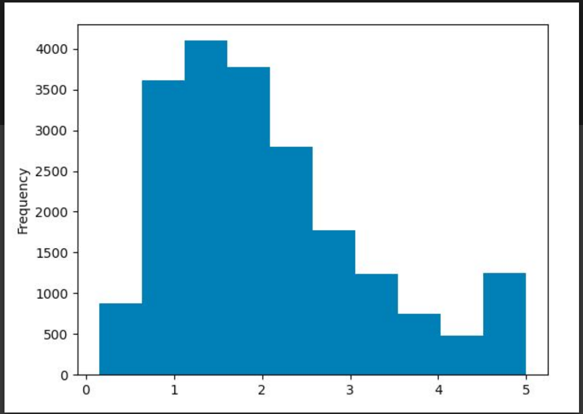


```python
import flytekitplugins.pandera
import pandera as pa
from flytekit import task
from pandera.typing import Series, DataFrame


class MySchema(pa.SchemaModel):
    col1: Series[float]
    col2: Series[int]
    col3: Series[str]


@task
def func(df: DataFrame[MySchema]):
    ...
```

Defining a Statistical Type for California Housing Dataset 🏡



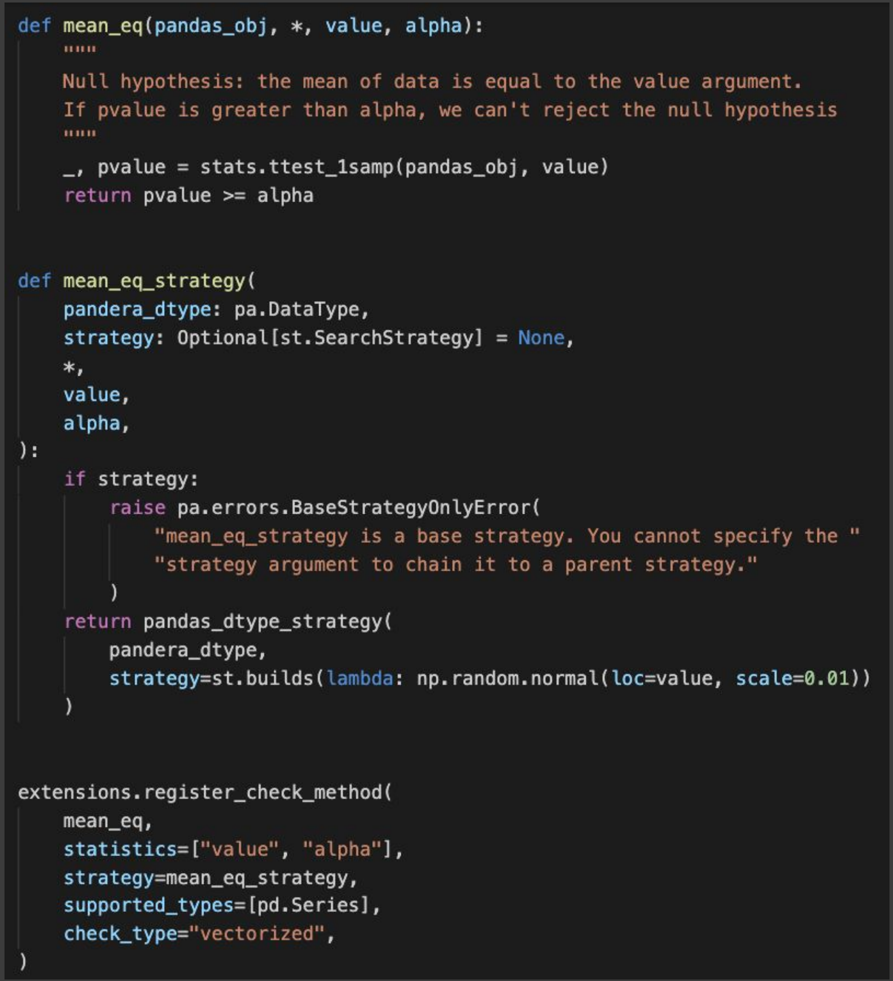
```
class CaliforniaHousingData(pa.SchemaModel):
    Latitude: Series[float] = pa.Field(in_range={"min_value": -90, "max_value": 90})
    Longitude: Series[float] = pa.Field(in_range={"min_value": -180, "max_value": 180})
    AveBedrms: Series[float] = pa.Field(in_range={"min_value": 0, "max_value": 1_000_000})
    AveOccup: Series[float] = pa.Field(in_range={"min_value": 0, "max_value": 1_000_000})
    AveRooms: Series[float] = pa.Field(in_range={"min_value": 0, "max_value": 1_000_000})
    HouseAge: Series[float] = pa.Field(in_range={"min_value": 0, "max_value": 1_000_000})
    MedInc: Series[float] = pa.Field(in_range={"min_value": 0, "max_value": 1_000_000})
    MedHouseVal: Series[float] = pa.Field(
        mean_eq={
            "value": 2.0685,
            "alpha": 1e-3,
            "error": "MedHouseVal mean value is not equal to 2.0685 [alpha=1e-3]",
        }
    )

    class Config:
        coerce = True
```



log(MedHouseVal)

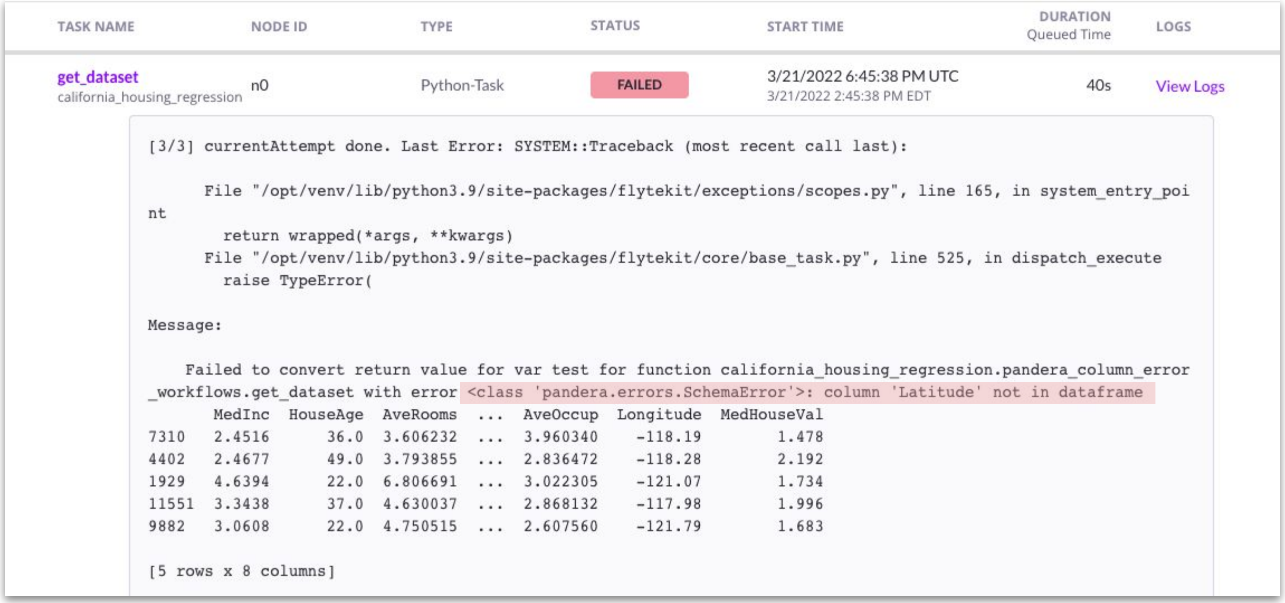
Custom Checks are Just…

🎉 Functions 🎉



```python
def mean_eq(pandas_obj, *, value, alpha):
    """
    Null hypothesis: the mean of data is equal to the value argument.
    If pvalue is greater than alpha, we can't reject the null hypothesis
    """
    _, pvalue = stats.ttest_1samp(pandas_obj, value)
    return pvalue >= alpha


def mean_eq_strategy(
    pandera_dtype: pa.DataType,
    strategy: Optional[st.SearchStrategy] = None,
    *,
    value,
    alpha,
):
    if strategy:
        raise pa.errors.BaseStrategyOnlyError(
            "mean_eq_strategy is a base strategy. You cannot specify the "
            "strategy argument to chain it to a parent strategy."
        )
    return pandas_dtype_strategy(
        pandera_dtype,
        strategy=st.builds(lambda: np.random.normal(loc=value, scale=0.01))
    )


extensions.register_check_method(
    mean_eq,
    statistics=["value", "alpha"],
    strategy=mean_eq_strategy,
    supported_types=[pd.Series],
    check_type="vectorized",
)
```

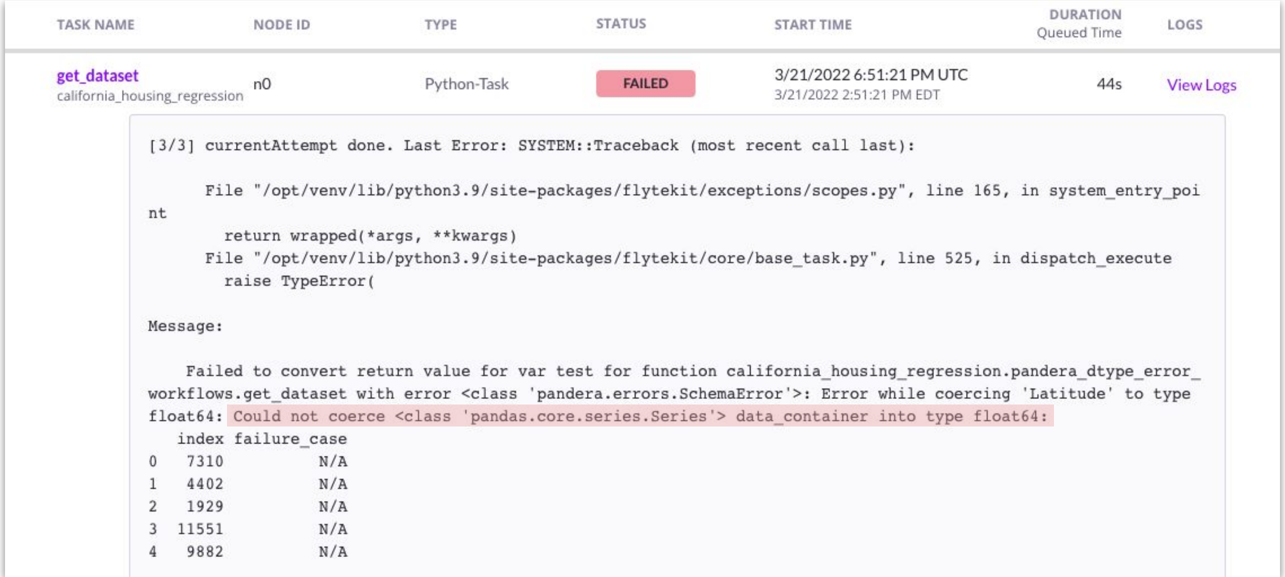
Know When Your Data Has Missing Columns 🏛



TASK NAME	NODE ID	TYPE	STATUS	START TIME	DURATION Queued Time	LOGS
get_dataset california_housing_regression	n0	Python-Task	FAILED	3/21/2022 6:45:38 PM UTC 3/21/2022 2:45:38 PM EDT	40s	View Logs

```
[3/3] currentAttempt done. Last Error: SYSTEM::Traceback (most recent call last):

      File "/opt/venv/lib/python3.9/site-packages/flytekit/exceptions/scopes.py", line 165, in system_entry_poi
nt
        return wrapped(*args, **kwargs)
      File "/opt/venv/lib/python3.9/site-packages/flytekit/core/base_task.py", line 525, in dispatch_execute
        raise TypeError(

Message:

    Failed to convert return value for var test for function california_housing_regression.pandera_column_error
_workflows.get_dataset with error <class 'pandera.errors.SchemaError'>: column 'Latitude' not in dataframe
       MedInc  HouseAge  AveRooms  ...  AveOccup  Longitude  MedHouseVal
7310   2.4516      36.0  3.606232  ...  3.960340    -118.19        1.478
4402   2.4677      49.0  3.793855  ...  2.836472    -118.28        2.192
1929   4.6394      22.0  6.806691  ...  3.022305    -121.07        1.734
11551  3.3438      37.0  4.630037  ...  2.868132    -117.98        1.996
9882   3.0608      22.0  4.750515  ...  2.607560    -121.79        1.683

[5 rows x 8 columns]
```

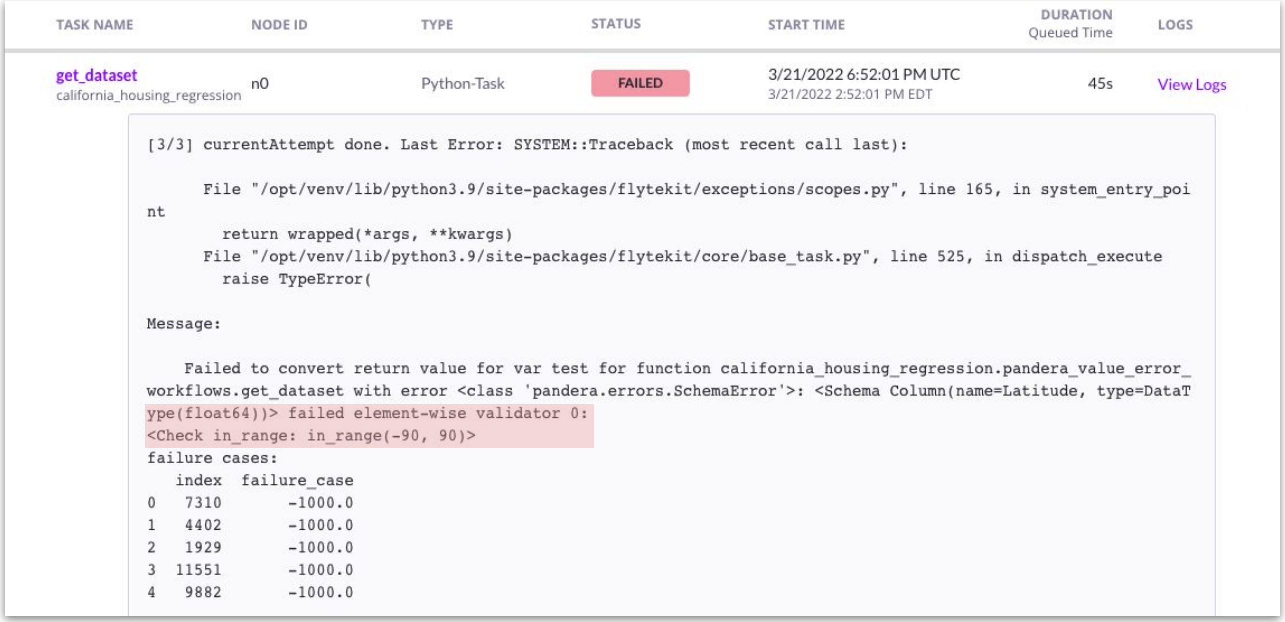
Know When Your Data Has the Wrong Type ⌨️



TASK NAME	NODE ID	TYPE	STATUS	START TIME	DURATION Queued Time	LOGS
get_dataset california_housing_regression	n0	Python-Task	FAILED	3/21/2022 6:51:21 PM UTC 3/21/2022 2:51:21 PM EDT	44s	View Logs

```
[3/3] currentAttempt done. Last Error: SYSTEM::Traceback (most recent call last):

      File "/opt/venv/lib/python3.9/site-packages/flytekit/exceptions/scopes.py", line 165, in system_entry_poi
nt
        return wrapped(*args, **kwargs)
      File "/opt/venv/lib/python3.9/site-packages/flytekit/core/base_task.py", line 525, in dispatch_execute
        raise TypeError(

Message:

    Failed to convert return value for var test for function california_housing_regression.pandera_dtype_error_
workflows.get_dataset with error <class 'pandera.errors.SchemaError'>: Error while coercing 'Latitude' to type
float64: Could not coerce <class 'pandas.core.series.Series'> data_container into type float64:
   index failure_case
0   7310          N/A
1   4402          N/A
2   1929          N/A
3  11551          N/A
4   9882          N/A
```

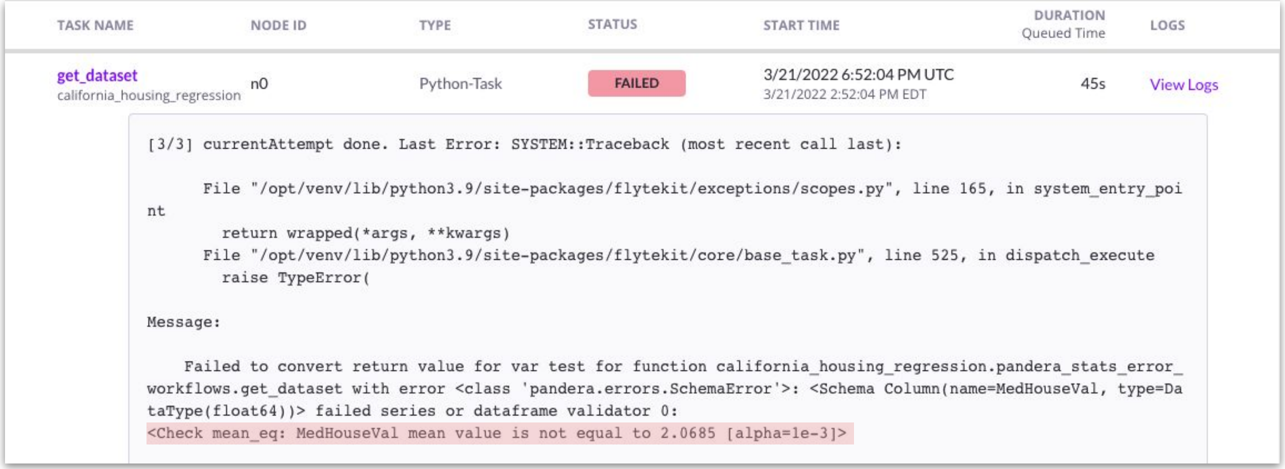
Know When Your Data Has the Wrong Values 💵



TASK NAME	NODE ID	TYPE	STATUS	START TIME	DURATION Queued Time	LOGS
get_dataset california_housing_regression	n0	Python-Task	FAILED	3/21/2022 6:52:01 PM UTC 3/21/2022 2:52:01 PM EDT	45s	View Logs

```
[3/3] currentAttempt done. Last Error: SYSTEM::Traceback (most recent call last):

      File "/opt/venv/lib/python3.9/site-packages/flytekit/exceptions/scopes.py", line 165, in system_entry_poi
nt
        return wrapped(*args, **kwargs)
      File "/opt/venv/lib/python3.9/site-packages/flytekit/core/base_task.py", line 525, in dispatch_execute
        raise TypeError(

Message:

    Failed to convert return value for var test for function california_housing_regression.pandera_value_error_
workflows.get_dataset with error <class 'pandera.errors.SchemaError'>: <Schema Column(name=Latitude, type=DataT
ype(float64))> failed element-wise validator 0:
<Check in_range: in_range(-90, 90)>
failure cases:
   index  failure_case
0   7310       -1000.0
1   4402       -1000.0
2   1929       -1000.0
3  11551       -1000.0
4   9882       -1000.0
```

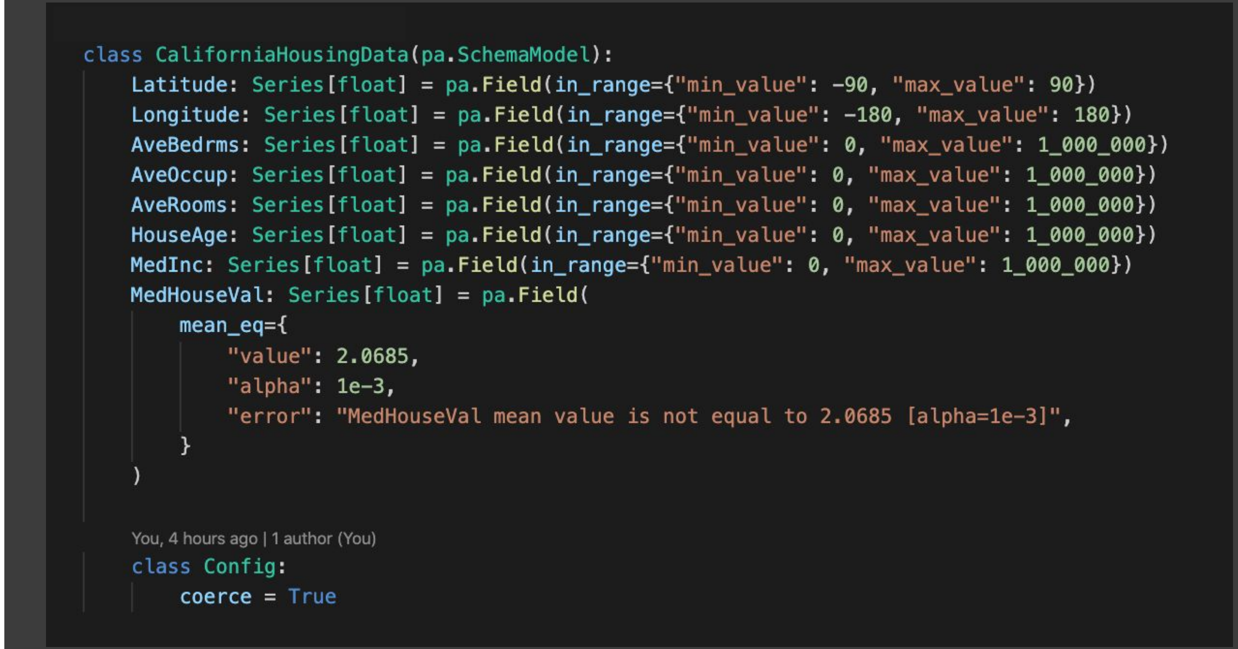
Know When Your Data Has the Wrong Statistical Distribution 📊



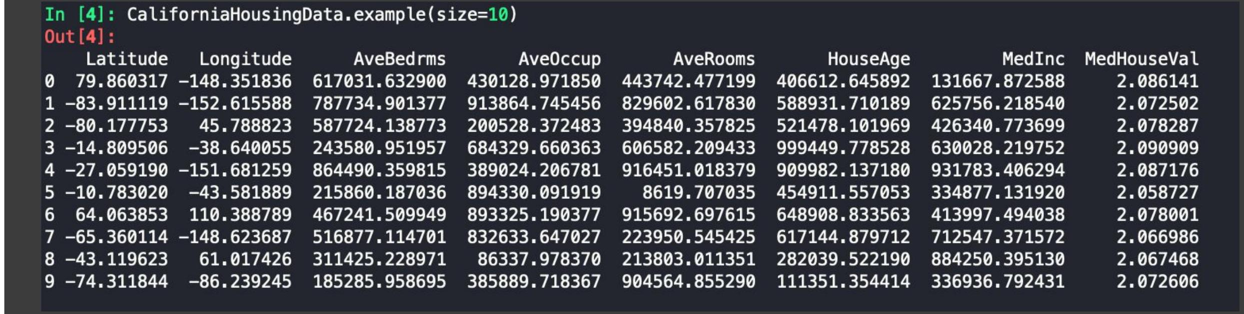
TASK NAME	NODE ID	TYPE	STATUS	START TIME	DURATION Queued Time	LOGS
get_dataset california_housing_regression	n0	Python-Task	FAILED	3/21/2022 6:52:04 PM UTC 3/21/2022 2:52:04 PM EDT	45s	View Logs

```
[3/3] currentAttempt done. Last Error: SYSTEM::Traceback (most recent call last):

      File "/opt/venv/lib/python3.9/site-packages/flytekit/exceptions/scopes.py", line 165, in system_entry_poi
nt
        return wrapped(*args, **kwargs)
      File "/opt/venv/lib/python3.9/site-packages/flytekit/core/base_task.py", line 525, in dispatch_execute
        raise TypeError(

Message:

    Failed to convert return value for var test for function california_housing_regression.pandera_stats_error_
workflows.get_dataset with error <class 'pandera.errors.SchemaError'>: <Schema Column(name=MedHouseVal, type=Da
taType(float64))> failed series or dataframe validator 0:
<Check mean_eq: MedHouseVal mean value is not equal to 2.0685 [alpha=1e-3]>
```

Synthesize Valid Data Under Your Schema's Constraints 🤯

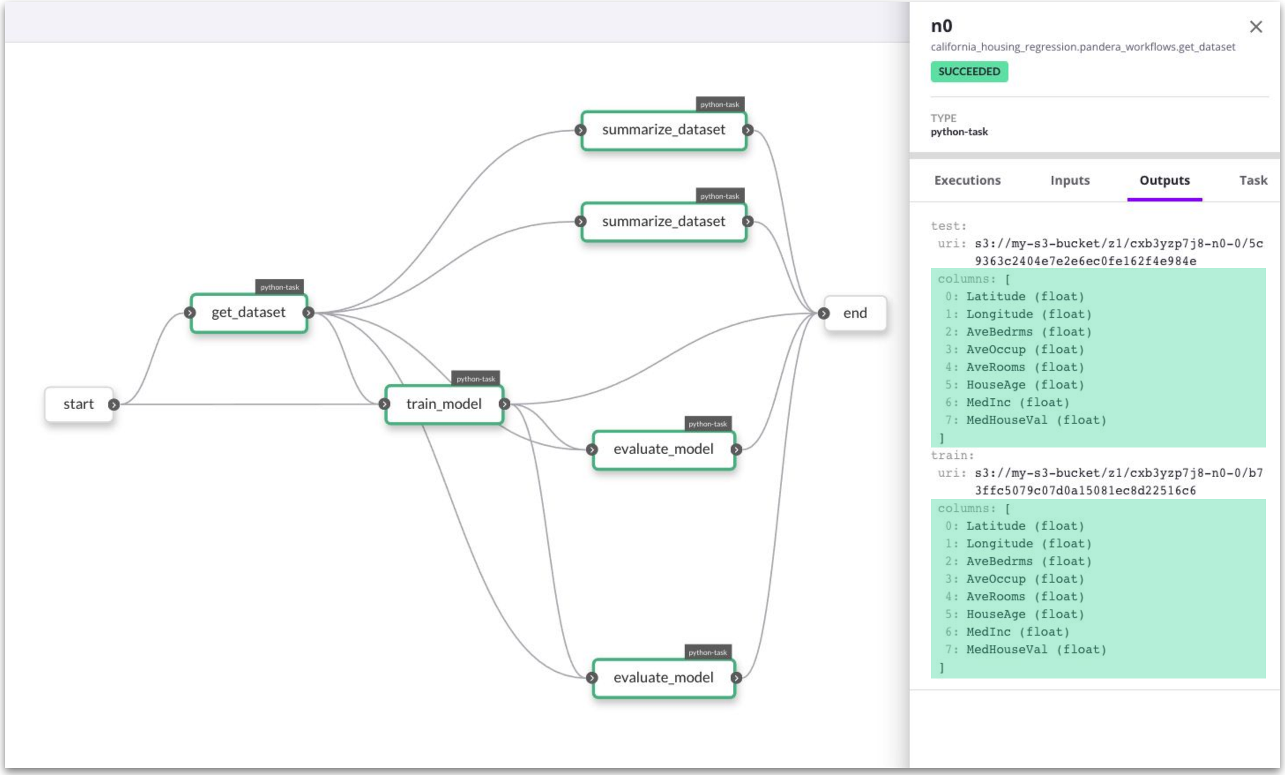


```
class CaliforniaHousingData(pa.SchemaModel):
    Latitude: Series[float] = pa.Field(in_range={"min_value": -90, "max_value": 90})
    Longitude: Series[float] = pa.Field(in_range={"min_value": -180, "max_value": 180})
    AveBedrms: Series[float] = pa.Field(in_range={"min_value": 0, "max_value": 1_000_000})
    AveOccup: Series[float] = pa.Field(in_range={"min_value": 0, "max_value": 1_000_000})
    AveRooms: Series[float] = pa.Field(in_range={"min_value": 0, "max_value": 1_000_000})
    HouseAge: Series[float] = pa.Field(in_range={"min_value": 0, "max_value": 1_000_000})
    MedInc: Series[float] = pa.Field(in_range={"min_value": 0, "max_value": 1_000_000})
    MedHouseVal: Series[float] = pa.Field(
        mean_eq={
            "value": 2.0685,
            "alpha": 1e-3,
            "error": "MedHouseVal mean value is not equal to 2.0685 [alpha=1e-3]",
        }
    )

    You, 4 hours ago | 1 author (You)
    class Config:
        coerce = True
```



```
In [4]: CaliforniaHousingData.example(size=10)
Out[4]:
    Latitude   Longitude      AveBedrms       AveOccup       AveRooms       HouseAge         MedInc  MedHouseVal
0  79.860317 -148.351836  617031.632900  430128.971850  443742.477199  406612.645892  131667.872588     2.086141
1 -83.911119 -152.615588  787734.901377  913864.745456  829602.617830  588931.710189  625756.218540     2.072502
2 -80.177753   45.788823  587724.138773  200528.372483  394840.357825  521478.101969  426340.773699     2.078287
3 -14.809506  -38.640055  243580.951957  684329.660363  606582.209433  999449.778528  630028.219752     2.090909
4 -27.059190 -151.681259  864490.359815  389024.206781  916451.018379  909982.137180  931783.406294     2.087176
5 -10.783020  -43.581889  215860.187036  894330.091919    8619.707035  454911.557053  334877.131920     2.058727
6  64.063853  110.388789  467241.509949  893325.190377  915692.697615  648908.833563  413997.494038     2.078001
7 -65.360114 -148.623687  516877.114701  832633.647027  223950.545425  617144.879712  712547.371572     2.066986
8 -43.119623   61.017426  311425.228971   86337.978370  213803.011351  282039.522190  884250.395130     2.067468
9 -74.311844  -86.239245  185285.958695  385889.718367  904564.855290  111351.354414  336936.792431     2.072606
```

Test Your Data...



n0

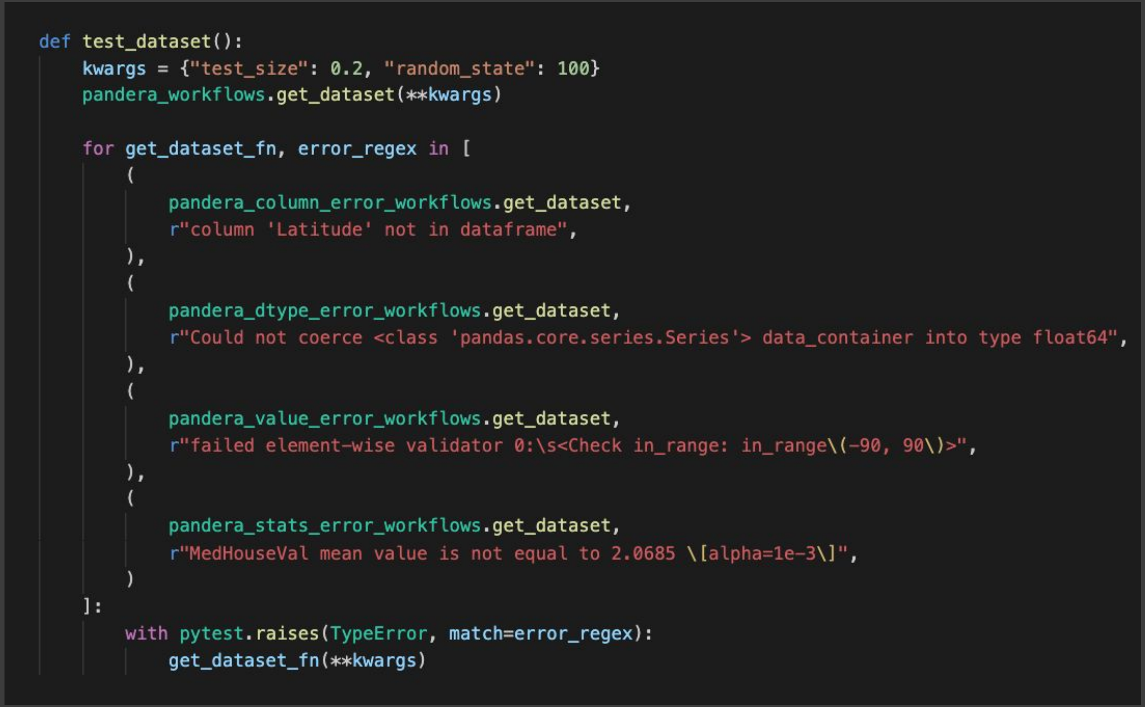
california_housing_regression.pandera_workflows.get_dataset

SUCCEEDED

TYPE
python-task

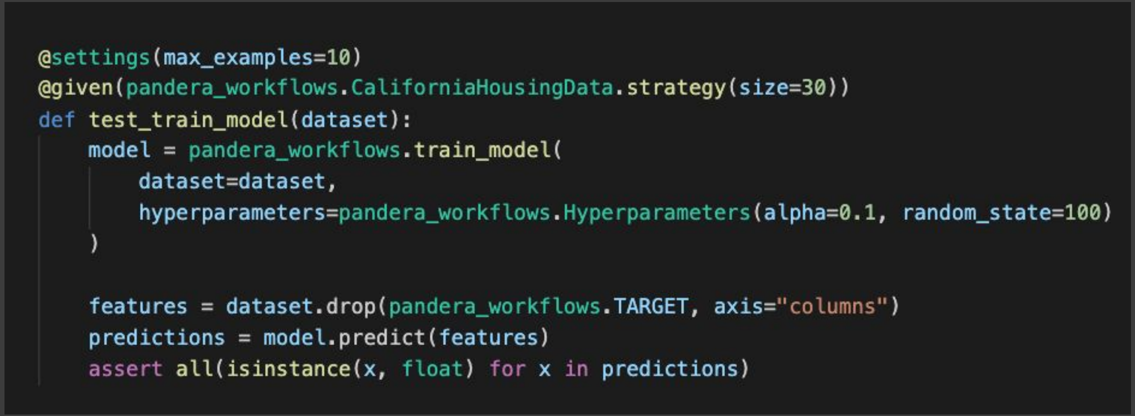
```
test:
 uri: s3://my-s3-bucket/z1/cxb3yzp7j8-n0-0/5c
      9363c2404e7e2e6ec0fe162f4e984e
 columns: [
  0: Latitude (float)
  1: Longitude (float)
  2: AveBedrms (float)
  3: AveOccup (float)
  4: AveRooms (float)
  5: HouseAge (float)
  6: MedInc (float)
  7: MedHouseVal (float)
 ]
train:
 uri: s3://my-s3-bucket/z1/cxb3yzp7j8-n0-0/b7
      3ffc5079c07d0a15081ec8d22516c6
 columns: [
  0: Latitude (float)
  1: Longitude (float)
  2: AveBedrms (float)
  3: AveOccup (float)
  4: AveRooms (float)
  5: HouseAge (float)
  6: MedInc (float)
  7: MedHouseVal (float)
 ]
```

… the Functions That Produce Them…



```python
def test_dataset():
    kwargs = {"test_size": 0.2, "random_state": 100}
    pandera_workflows.get_dataset(**kwargs)

    for get_dataset_fn, error_regex in [
        (
            pandera_column_error_workflows.get_dataset,
            r"column 'Latitude' not in dataframe",
        ),
        (
            pandera_dtype_error_workflows.get_dataset,
            r"Could not coerce <class 'pandas.core.series.Series'> data_container into type float64",
        ),
        (
            pandera_value_error_workflows.get_dataset,
            r"failed element-wise validator 0:\s<Check in_range: in_range\(-90, 90\)>",
        ),
        (
            pandera_stats_error_workflows.get_dataset,
            r"MedHouseVal mean value is not equal to 2.0685 \[alpha=1e-3\]",
        )
    ]:
        with pytest.raises(TypeError, match=error_regex):
            get_dataset_fn(**kwargs)
```

… and the Artifacts They Help Create.



```python
@settings(max_examples=10)
@given(pandera_workflows.CaliforniaHousingData.strategy(size=30))
def test_train_model(dataset):
    model = pandera_workflows.train_model(
        dataset=dataset,
        hyperparameters=pandera_workflows.Hyperparameters(alpha=0.1, random_state=100)
    )

    features = dataset.drop(pandera_workflows.TARGET, axis="columns")
    predictions = model.predict(features)
    assert all(isinstance(x, float) for x in predictions)
```

Takeaway 1

Flyte is an *orchestration* and *distributed execution* platform where **type-safety** is deeply integrated with other features, which together provide strong *reliability*, *efficiency*, and *auditability* guarantees.

Takeaway 2

With **Pandera**, you can ensure the *quality of data* flowing through your machine learning pipelines *and the correctness of those pipelines themselves* by expressing **statistical types** *directly in your codebase*.

Takeaway 3

With **Flyte** and **Pandera** combined, you can **build**, **deploy**, and **scale** these ML pipelines while enjoying the guarantee that, when things go wrong, you'll know where exactly the error occurred to help you fix it.

Flyte Roadmap

Flyte Decks: A Customizable Reporting API for your Pipeline Artifacts

ML-awareness: Intra-task model checkpointing, data labeling.

Serving Integrations: support for model serving, low latency batch workflows, model monitoring.

Pandera Roadmap

Extensibility: support for *xarray*, *jsonschema*, *pyarrow,* and more!

User Experience: more built-in checks, statistical hypothesis checks

Interoperability: tighter integrations with the python ecosystem, e.g. *fastapi*, *pydantic*, *pytest*

Where do I learn more?

Flyte

website: www.flyte.org
docs: docs.flyte.org
repo: github.com/flyteorg/flyte

Pandera

docs: pandera.readthedocs.io
repo: github.com/pandera-dev/pandera

Contact

email: niels@union.ai
twitter: @cosmicbboy
linkedin: linkedin.com/in/nbantilan

SODA

On the importance of using a data quality framework to monitor your data.

Don’t Let Your Models Decay!

Bastien Boutonnet,
Lead Data Scientist

Bastien joined Soda last year and before that he was at TripActions and Travelbird, once he decided that the Postdoctoral Fellow wasn't half the fun ;-). He's a die hard dbt fan, DJ, and French person living in Amsterdam.

https://www.bastienboutonnet.com

Zillow: A Cautionary Tale

$500m

Zillow **overestimated** the value of the houses it purchased in Q3 and Q4 of 2021 **by over $500m**

~25%

Q3 **losses of $304m**, leading into a **~25%** workforce reduction

Coincided with a **strong change in housing market conditions** which causes housing prices to fall

Strong evidence that their models were trained on the, then “old” situation which indicated growing prices, which caused their **unattended models to work under a different “assumption”** or concept

Did they do everything badly?

No!

- Their models were rigorously tested during development
- Their models were released to production gradually and KPIs were closely monitored.
- When those were deemed satisfactory, humans derived decisions to aggressively expand their purchasing programme.

Any good Data Science team would do that. It's their job and part of deploying to prod.

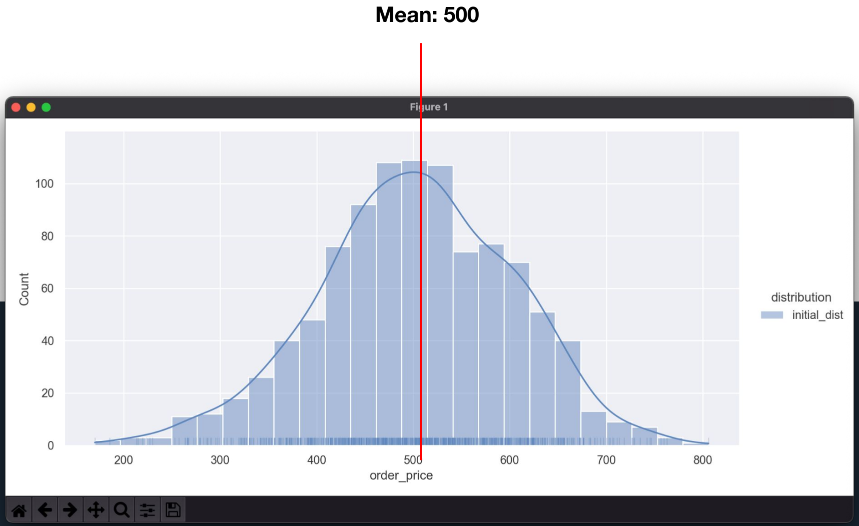
Could it have been avoided?

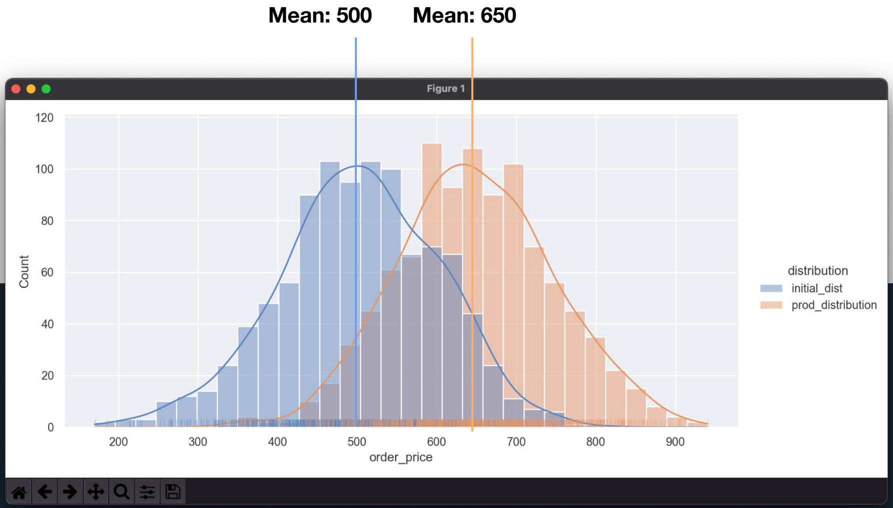
Yes!

- Some phenomena in nature are likely to change, and can do so drastically. When it comes to pricing, that's definitely true.
- This is commonly referred to as "data drift" and it can be detected by:
  - Tracking and alerting on drift.
  - Tracking and altering on accuracy.

Any good Data Science team is aware of that, but data quality management is not their job or core product.

A Detour Into Data Drift





So what is data drift?

When the distribution of one or more of your input features has changed between, for example training time and deployment.

How do you detect it?

01

“Freeze” a reference distribution

02

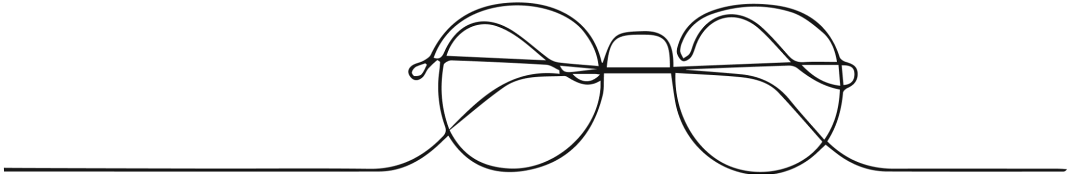
Compare distribution at time t+n and reference distribution

Simple right? 😉

On the Importance of A Data Quality Framework, Whichever It Is

Why data quality monitoring is “hard”.

- Simply put: you have to write a bunch more code
- Choose your methods from a sometimes large pool
- Orchestrating the checks
- Make it reusable
- Maintain and extend
- The list goes on…

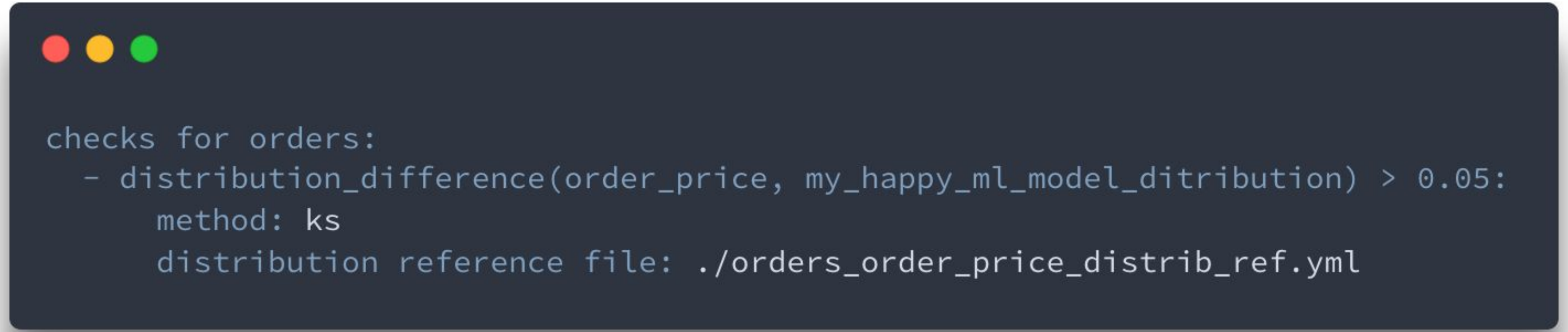


Data quality should not add time to release.

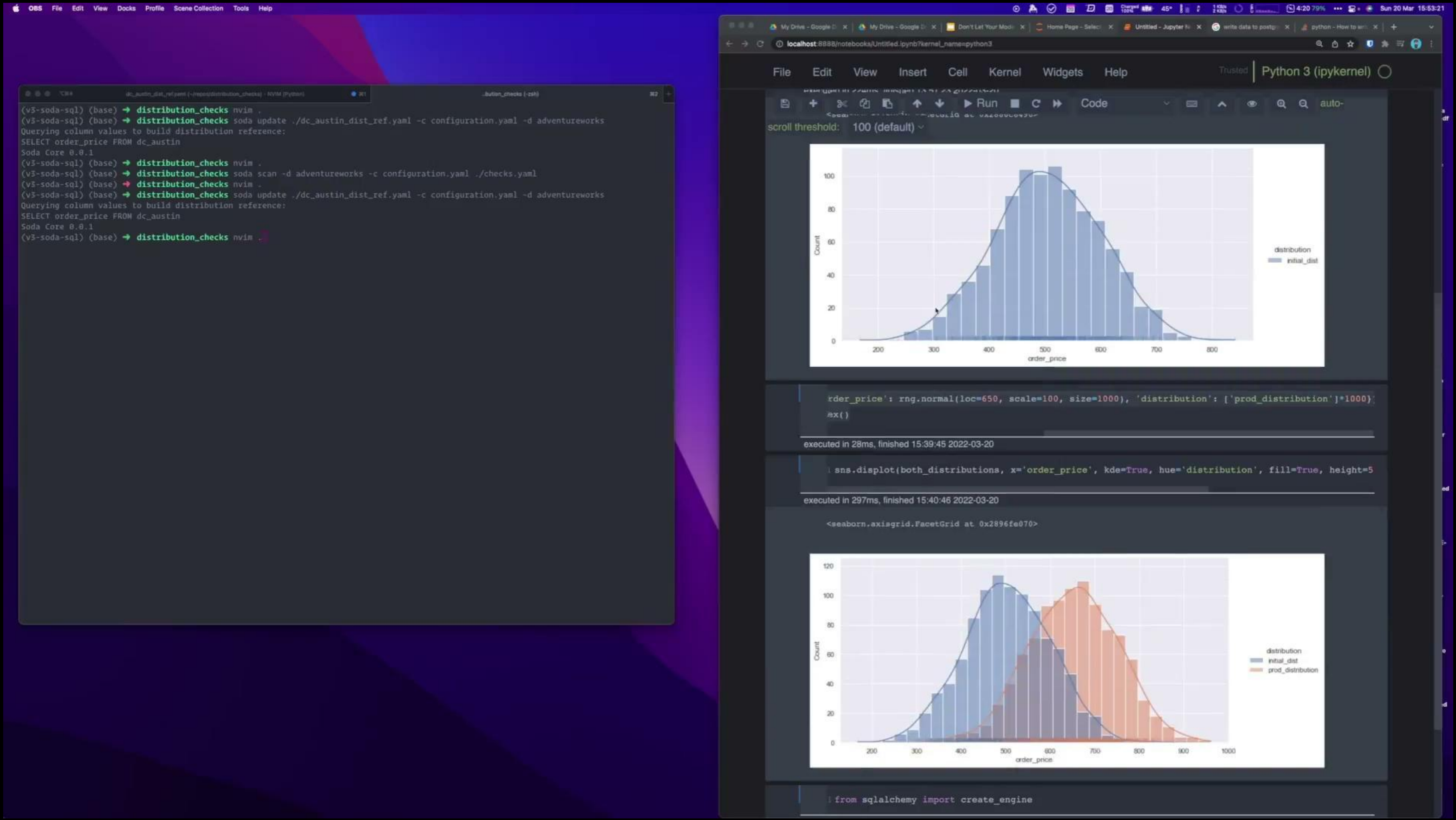
- Developing ML automation takes time and resources
- Data quality monitoring, isn’t an internal data team’s core product.
- Implementing data quality monitoring can easily increase the scope of any data product’s feature set with no direct value add.
- It often ends up “on the backlog”

Wouldn’t it Be Nice If…

Wouldn’t it be nice if you could to the following:



```
checks for orders:
  - distribution_difference(order_price, my_happy_ml_model_ditribution) > 0.05:
      method: ks
      distribution reference file: ./orders_order_price_distrib_ref.yml
```


OBS File Edit View Docks Profile Scene Collection Tools Help
Sun 20 Mar 15:53:21
localhost:8888/notebooks/Untitled.ipynb?kernel_name=python3
File Edit View Insert Cell Kernel Widgets Help
Python 3 (ipykernel)
Run
Code
scroll threshold: 100 (default)
Count
order_price
distribution
initial_dist
prod_distribution

```
(v3-soda-sql) (base) ➜ distribution_checks nvim .
(v3-soda-sql) (base) ➜ distribution_checks soda update ./dc_austin_dist_ref.yaml -c configuration.yaml -d adventureworks
Querying column values to build distribution reference:
SELECT order_price FROM dc_austin
Soda Core 0.0.1
(v3-soda-sql) (base) ➜ distribution_checks nvim .
(v3-soda-sql) (base) ➜ distribution_checks soda scan -d adventureworks -c configuration.yaml ./checks.yaml
(v3-soda-sql) (base) ➜ distribution_checks nvim .
(v3-soda-sql) (base) ➜ distribution_checks soda update ./dc_austin_dist_ref.yaml -c configuration.yaml -d adventureworks
Querying column values to build distribution reference:
SELECT order_price FROM dc_austin
Soda Core 0.0.1
(v3-soda-sql) (base) ➜ distribution_checks nvim .
```

```
rder_price': rng.normal(loc=650, scale=100, size=1000), 'distribution': ['prod_distribution']*1000)
ex()
```

executed in 28ms, finished 15:39:45 2022-03-20

```
sns.displot(both_distributions, x='order_price', kde=True, hue='distribution', fill=True, height=5
```

executed in 297ms, finished 15:40:46 2022-03-20

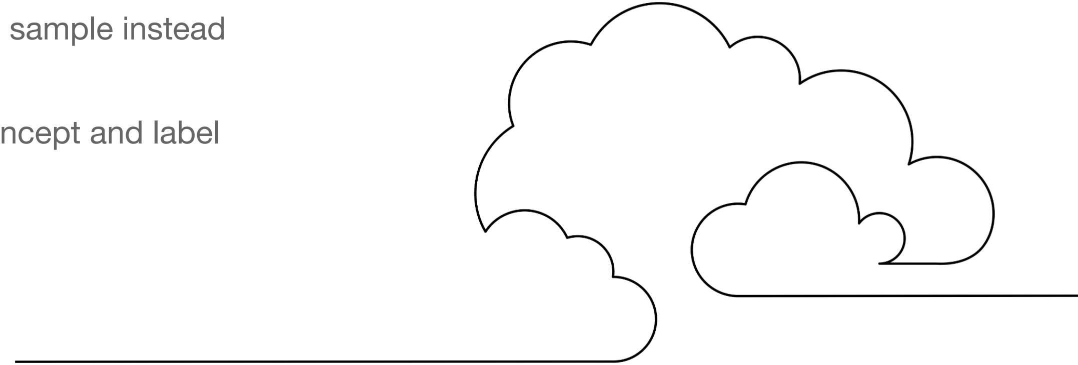
```
<seaborn.axisgrid.FacetGrid at 0x2896fe070>
```

```
from sqlalchemy import create_engine
```

What’s Next?

Why stop there?

- Connect to **Soda Cloud** (to avoid inconvenience of experimental file-based experimental feature)
- Rich visualisation in Cloud/and OSS
- More user control over algos + more algos to choose from
- Entirely data based solution (store reference sample instead of object in cloud/s3)
- Bespoke drift wrappers (monitor for both concept and label drift over one or several datasets)

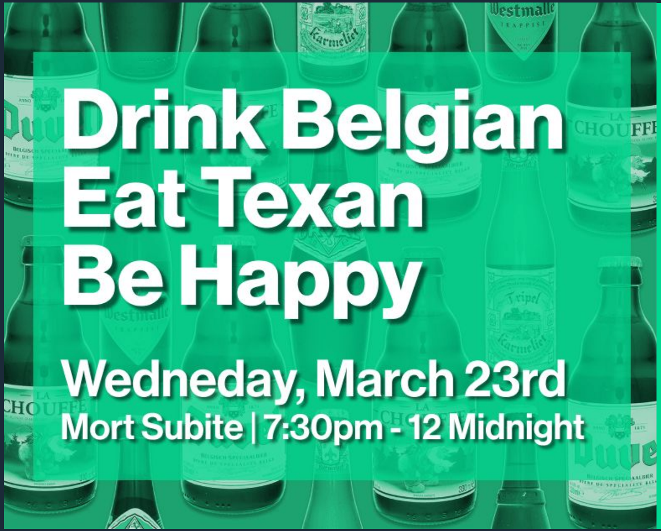


Give it a try!

- docs.soda.io
- `pip install soda-core-[datasource_type] soda-core-scientific`
- https://github.com/sodadata/soda-core

Hit me up, I’ll be around!

- Bastien Boutonnet
  (find me on the socials)
- bastien@soda.io
- www.bastienboutonnet.com



Drink Belgian
Eat Texan
Be Happy

Wedneday, March 23rd
Mort Subite | 7:30pm - 12 Midnight

Say Hello While in Austin

- **We’re at Booth 23**
- Watch a Soda product demo
- Join our Happy Hour
- Get good swag

Thank You!