Model Deployment

Robert Clements

MSDS Program

University of San Francisco

What to Expect

- Goal: to learn about the different deployment patterns for ML models and the pros and cons of each.

- How: after learning about the different patterns, we will start practicing with an approach that will work easily for the final project using a combination of Metaflow, Seldon and Streamlit.

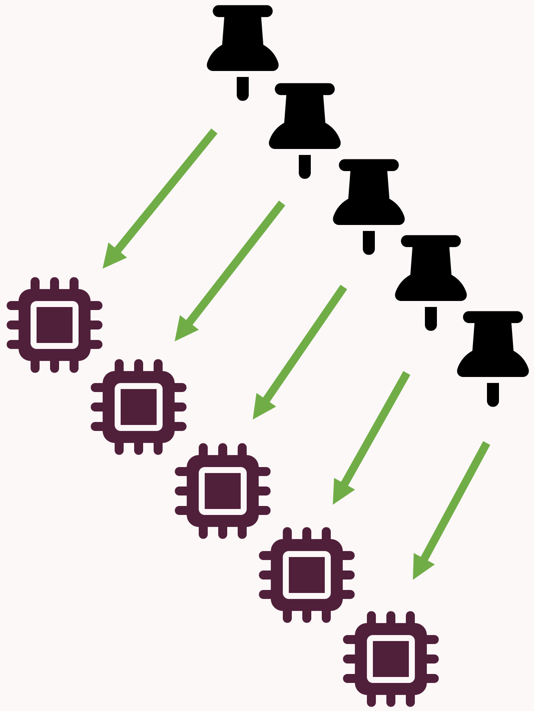
Making Predictions

- A model to generate a list of leads to investigate for potential fraudulent healthcare billing practices
- A recommender system for millions of products, or something to watch on Netflix
- A language model for extracting names of people and locations
- A model to match customers with drivers delivering their meals

Considerations

Scalability

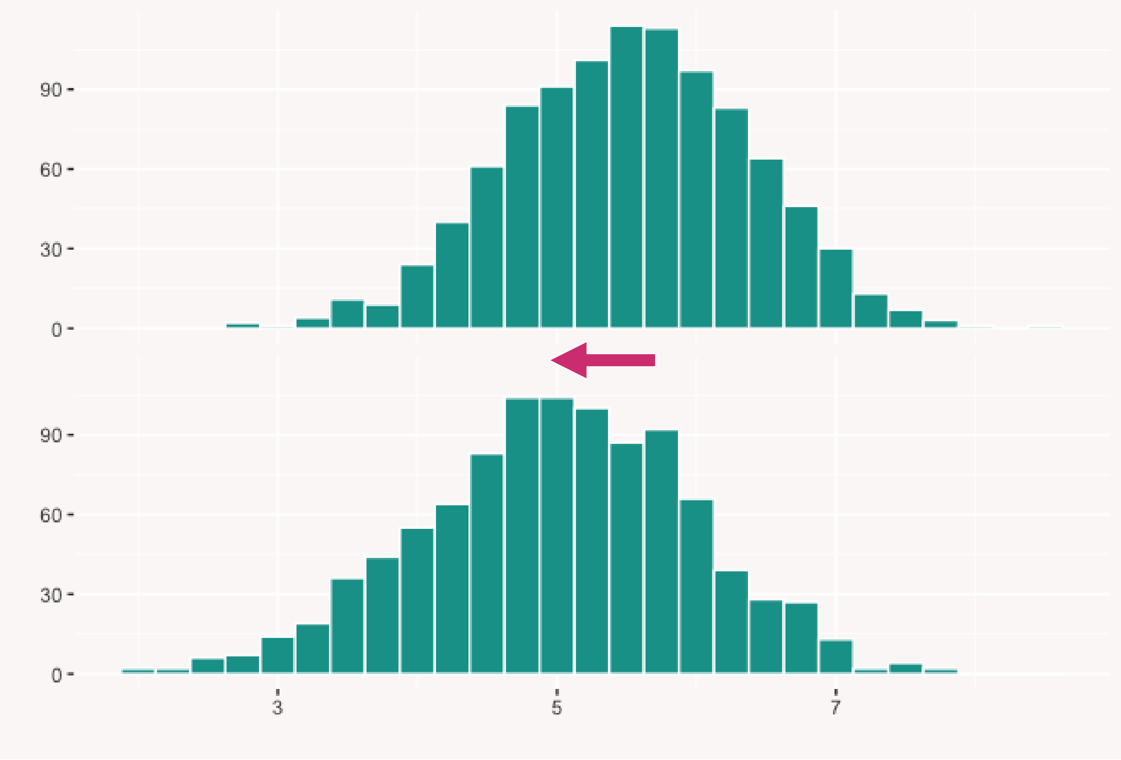
How do you go from making one prediction to making thousands or millions of predictions?



Considerations

Change

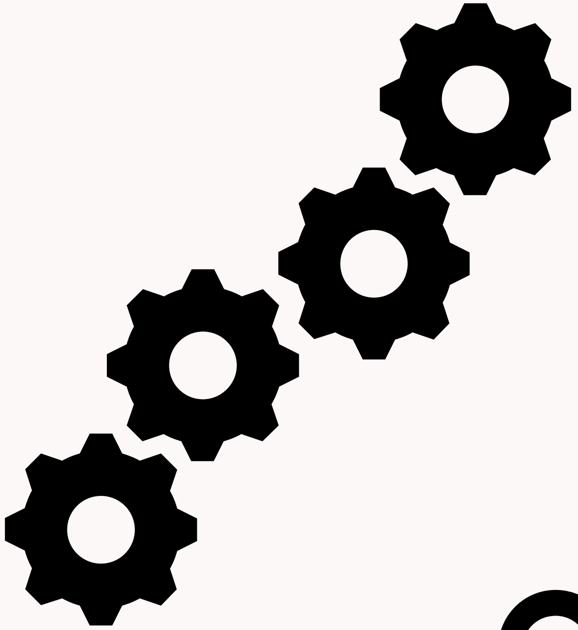
Data and code will change over time. When data changes, models may need to change.

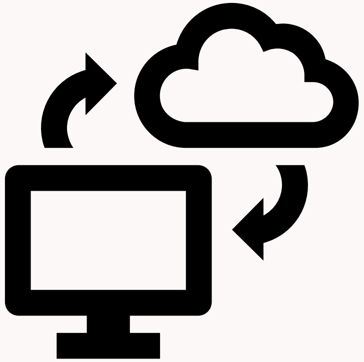


Considerations

Integrations

What do you do with your predictions after they are made? How do you take action on them? How can we “operationalize” models?

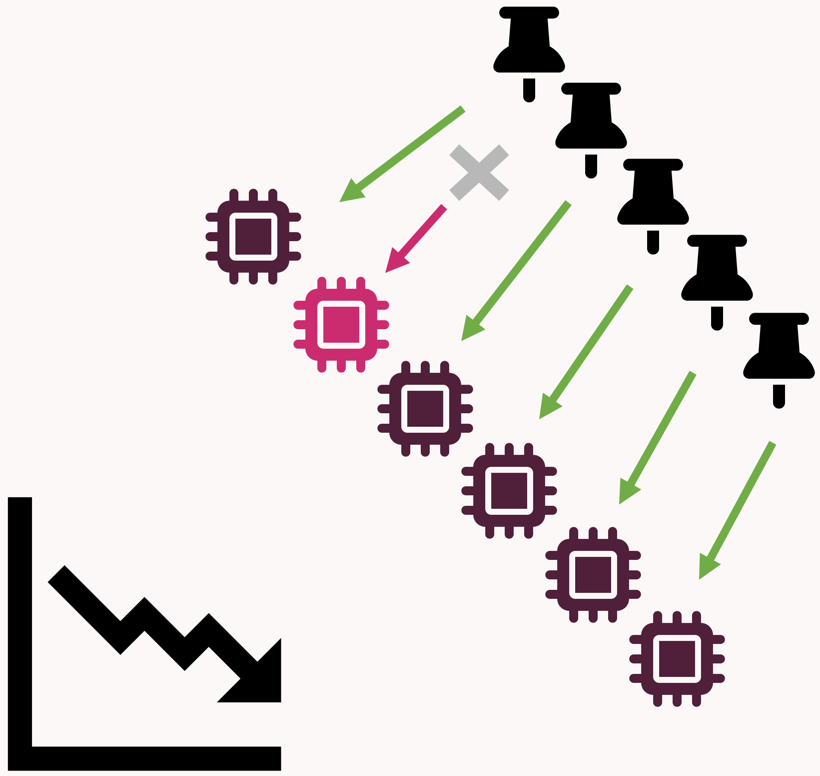




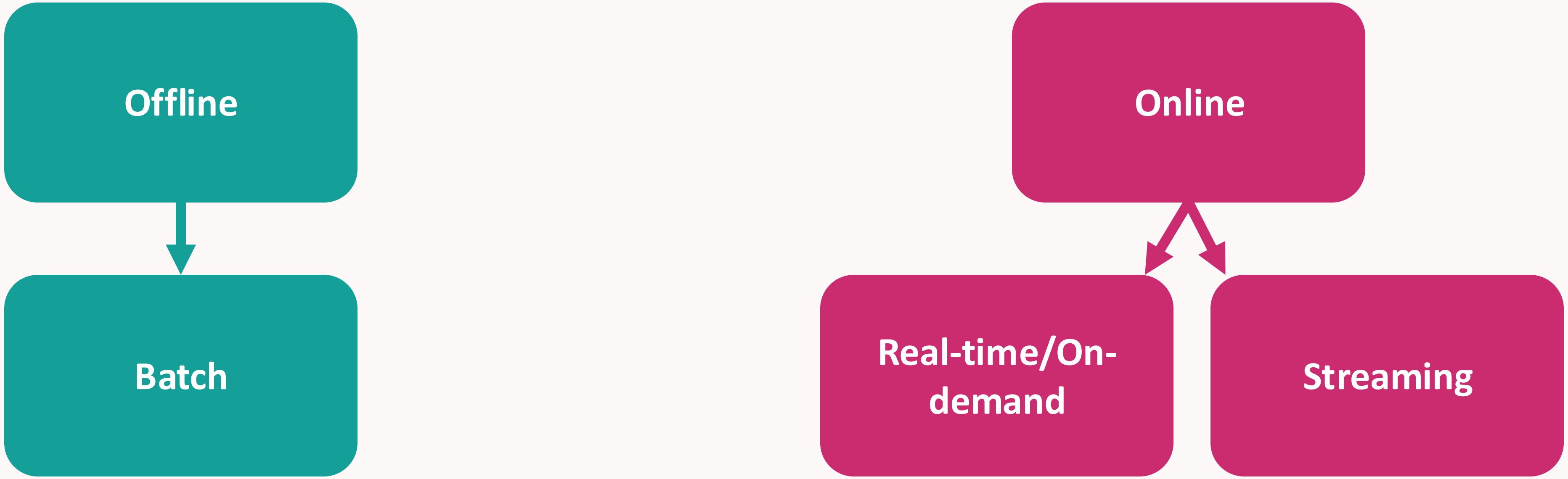
Considerations

Failures

ML systems are complex, providing many opportunities for failures. Models will also degrade and fail at some point.

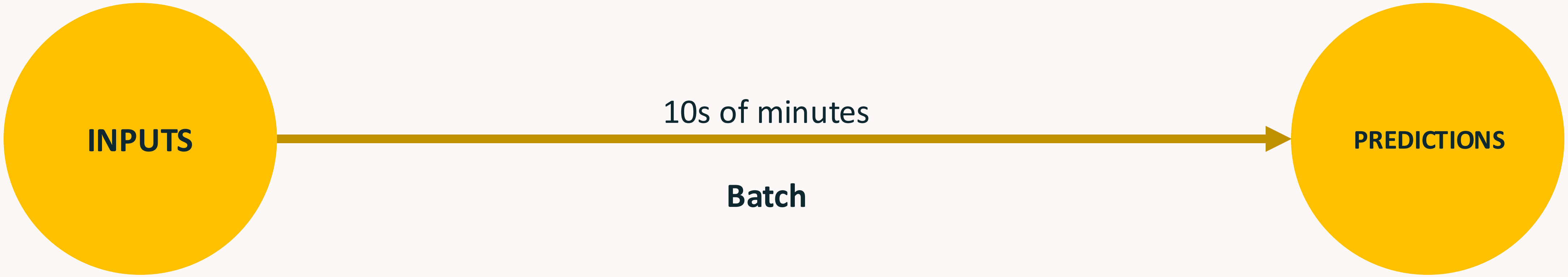


Two Primary Types



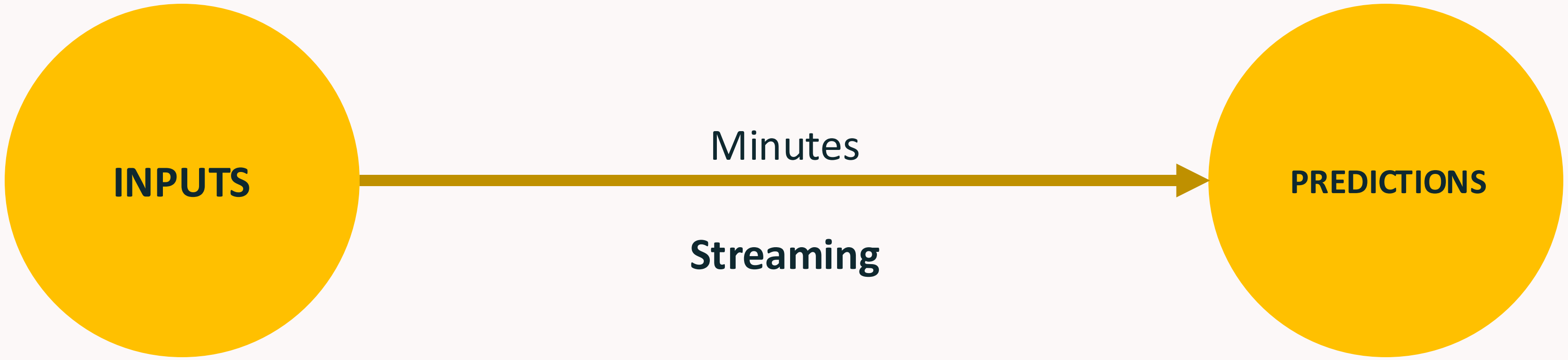
Batch, Streaming, Real-time (on-demand)

Given inputs, how quickly do we need predictions?



Batch, Streaming, Real-time (on-demand)

Given inputs, how quickly do we need predictions?

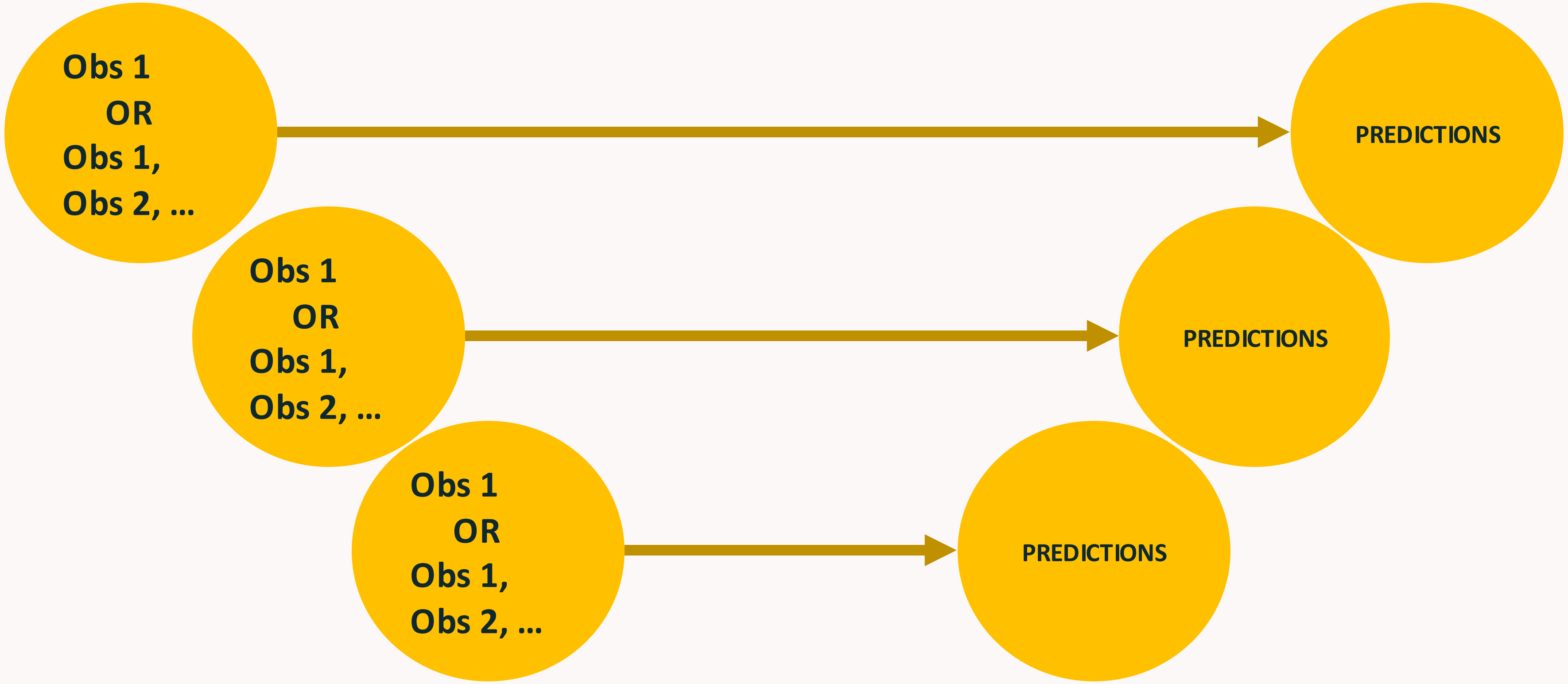


Batch, Streaming, Real-time (on-demand)

Given inputs, how quickly do we need predictions?

Batch, Streaming, Real-time (on-demand)

All can make predictions on batches of observations, or a single observation.



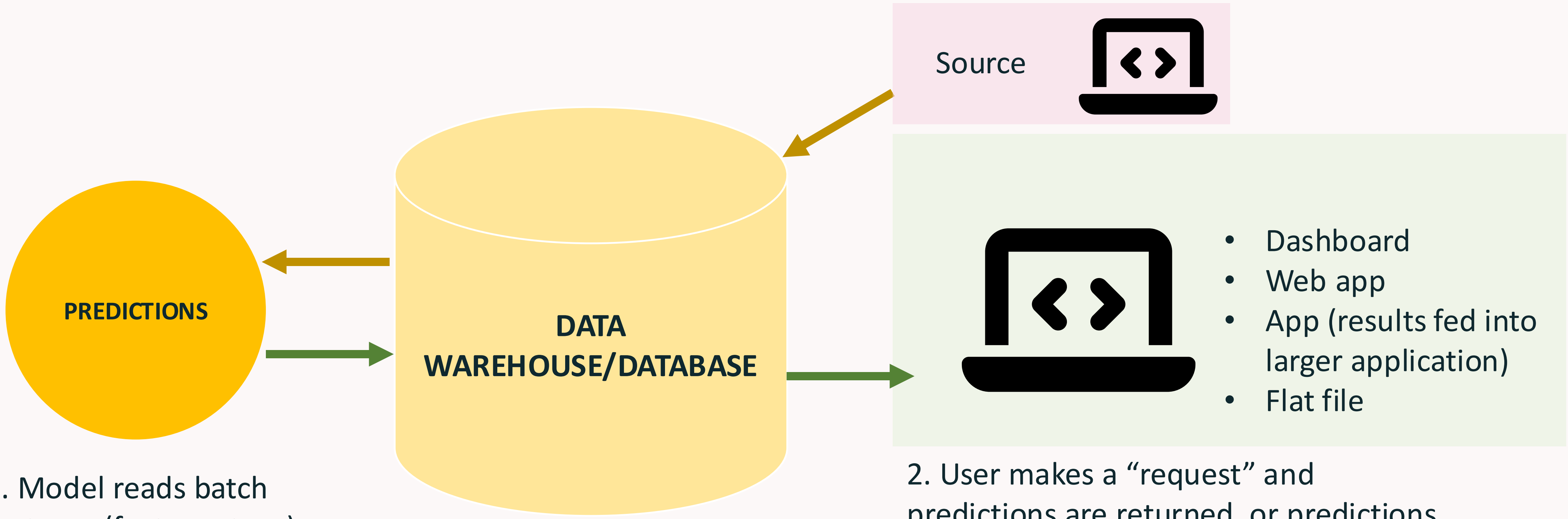
Canary and Shadow Mode Deployments

How should we roll out our new model?

We'll talk about this in the model monitoring section

Batch Predictions

How are predictions used?

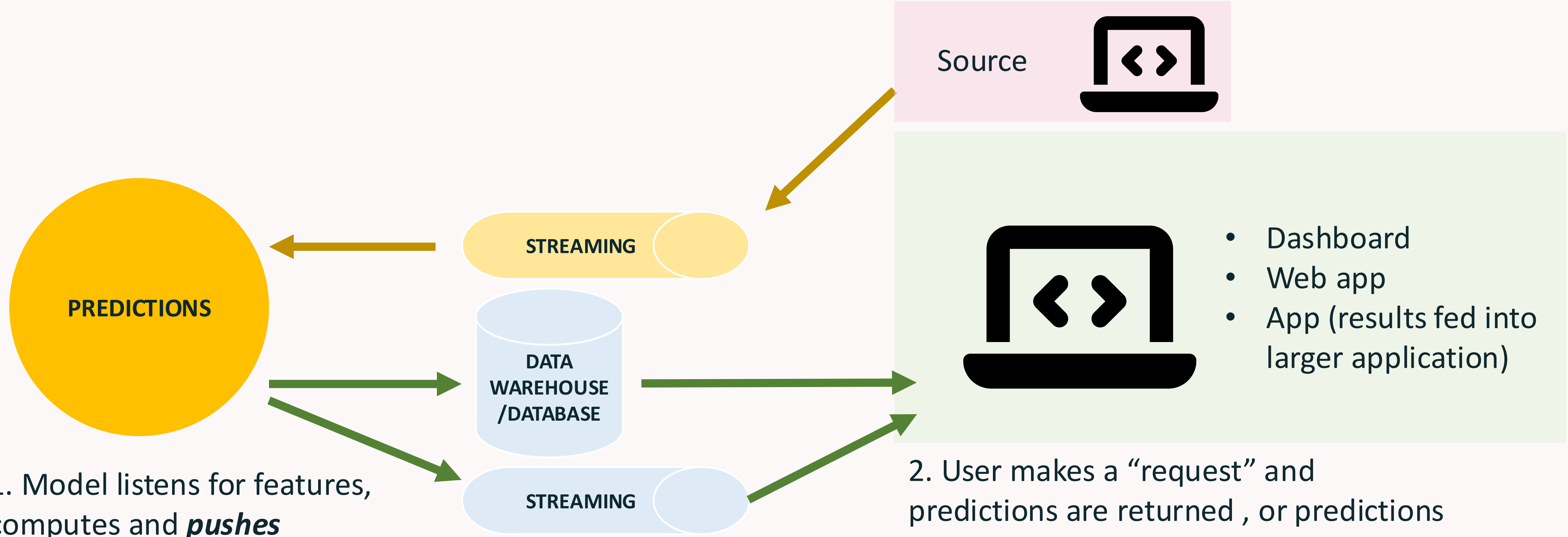


1. Model reads batch features (feature store), computes and ***pushes*** predictions.

2. User makes a “request” and predictions are returned, or predictions are pushed to user’s system. Predictions are needed > 10 minutes, at least, after inputs are known.

Streaming Predictions

How are predictions used?

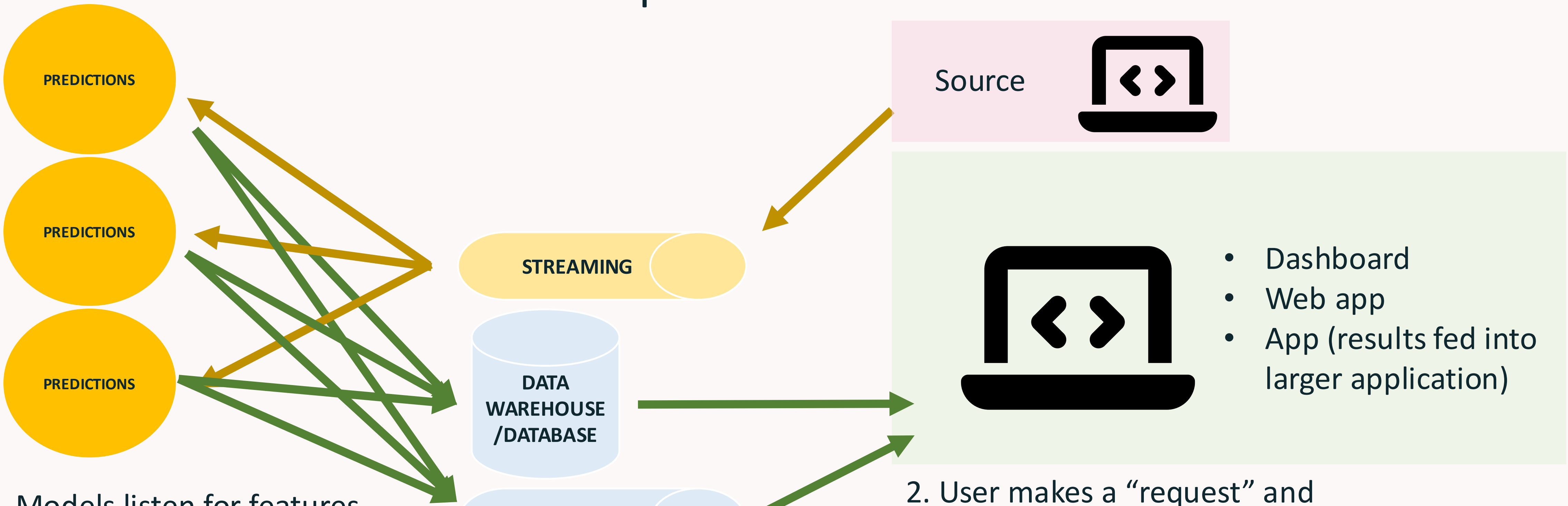


1. Model listens for features, computes and ***pushes*** predictions.

2. User makes a “request” and predictions are returned , or predictions are pushed to user’s system. Predictions are needed < 10 minutes, at most, after inputs are known.

Streaming Predictions

How are predictions used?

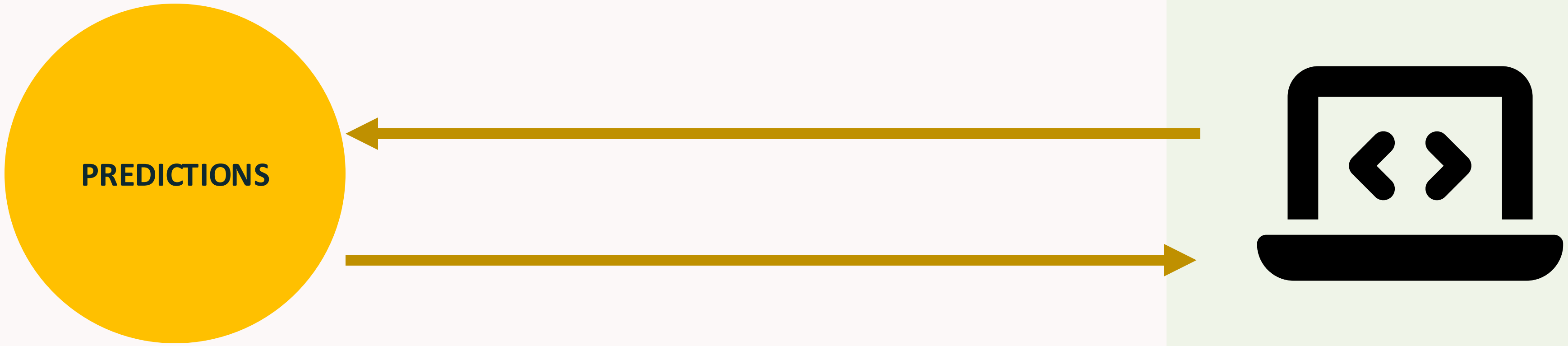


1. Models listen for features, computes and ***pushes*** predictions.

2. User makes a "request" and predictions are returned , or predictions are pushed to user's system. Predictions are needed < 10 minutes, at most, after inputs are known.

Real-time Predictions

How are predictions used?

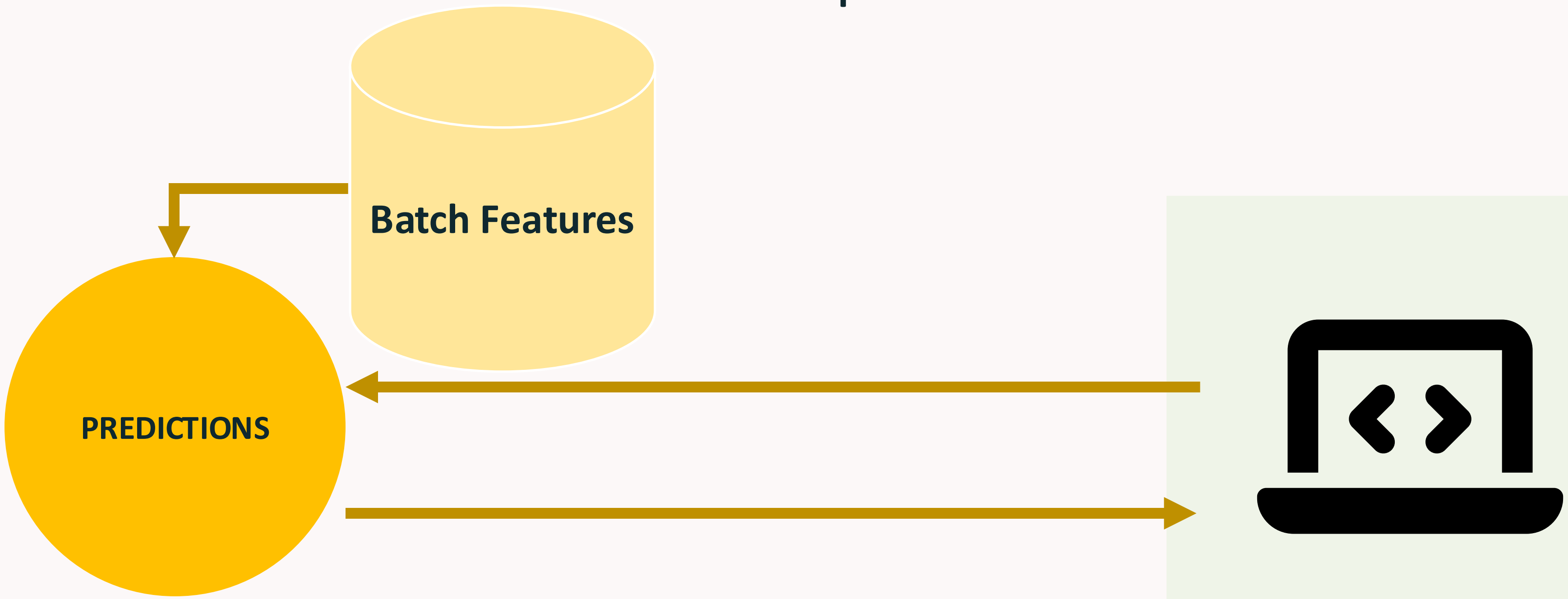


- Dashboard
- Web app
- App (results fed into larger application)

1. User makes a “request”, inputs are processed and predictions are ***pulled***. Predictions are needed within seconds/ms after inputs are known.

Real-time Predictions w/Batch Features

How are predictions used?

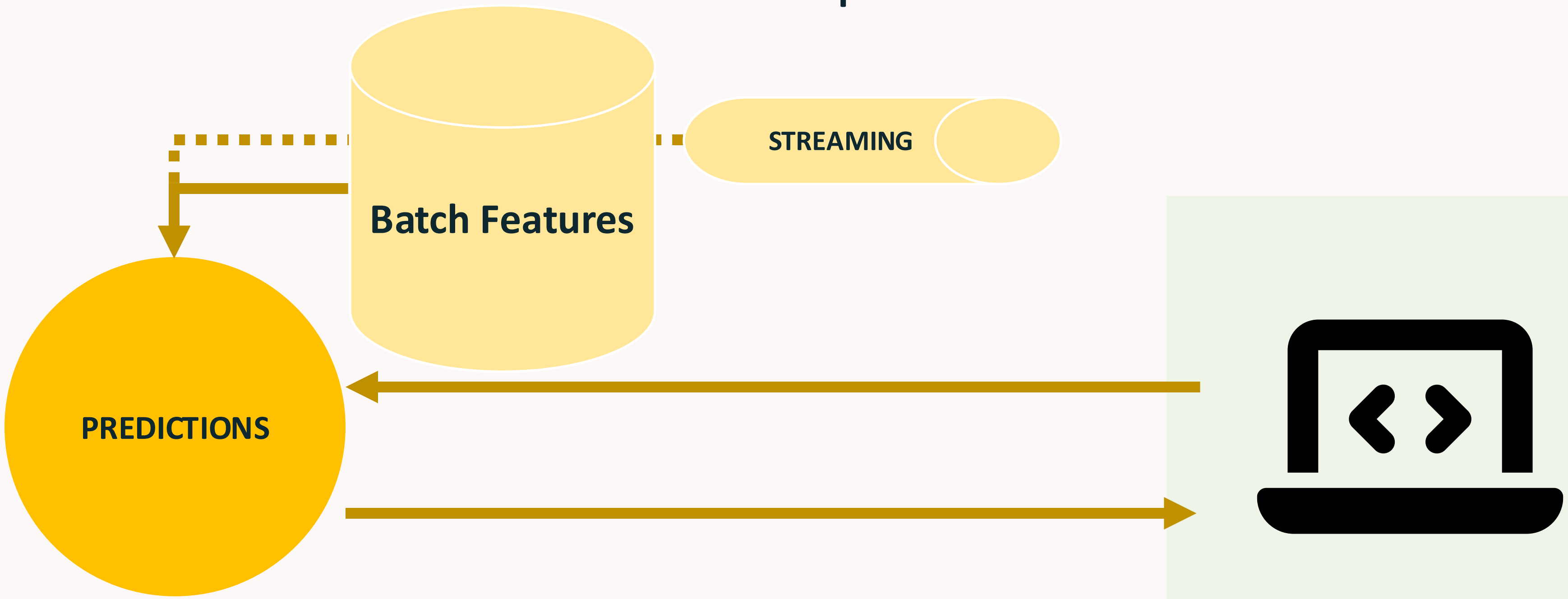


- Dashboard
- Web app
- App (results fed into larger application)

1. User makes a “request”, inputs are processed, batch features are loaded, and predictions are ***pulled***. Predictions are needed within seconds/ms after inputs are known.

Real-time Predictions w/Batch + Streaming Features

How are predictions used?



- Dashboard
- Web app
- App (results fed into larger application)
- Flat file

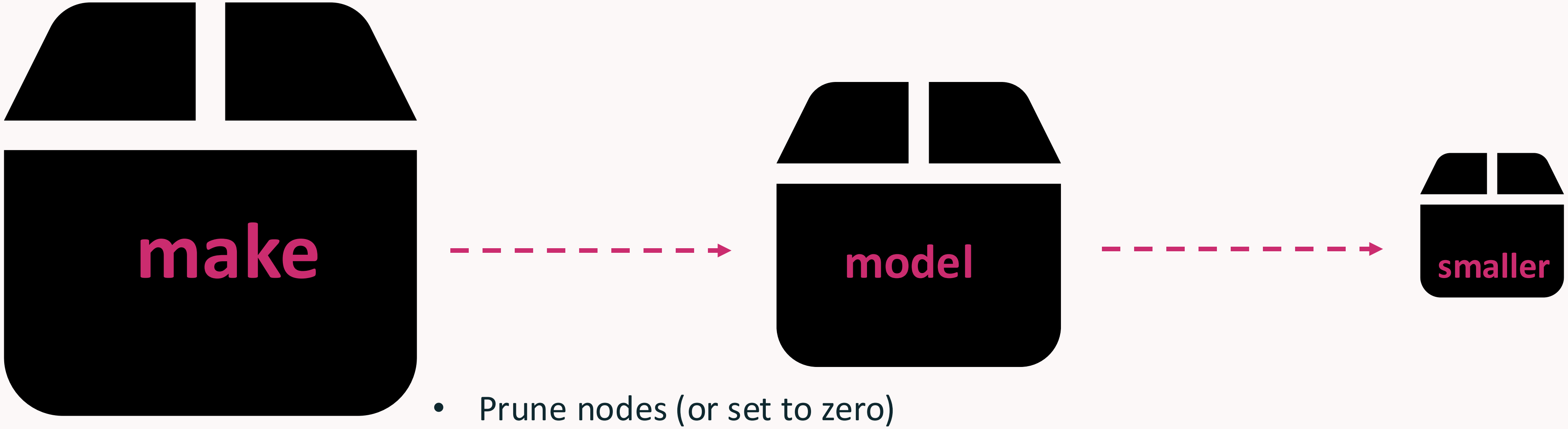
1. User makes a “request”, inputs are processed, batch and streaming features are loaded, and predictions are ***pulled***. Predictions are needed within seconds/ms after inputs are known.

Terminology

- Some references will refer to real-time processing as streaming
- Some references will refer to real-time processing as on-demand or online
- Some references will refer to batch and streaming processing as offline
- Some references will refer to batch processing as asynchronous and online processing as synchronous
- Some references will not differentiate batch processing from streaming processing, and instead will only differentiate batch and real-time/on-demand/online
- Features can be both batch (pre-computed) and streaming (computed from streaming data)
- It is all very confusing…

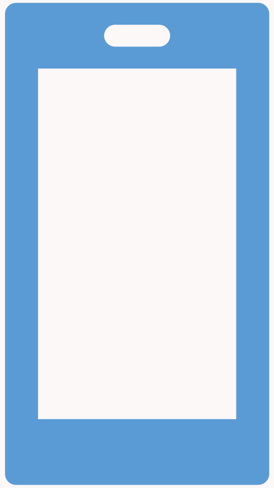
Getting to Real-time Predictions

When near-real-time is not good enough:



- Prune nodes (or set to zero)
- Train smaller model based off larger model (e.g. DistilBERT)
- Quantization (e.g. represent floats using 16 bits instead of 32)
- Caching (cache most common predictions using functools)
- This presentation from Roblox used many of these methods

Cloud and Edge ML

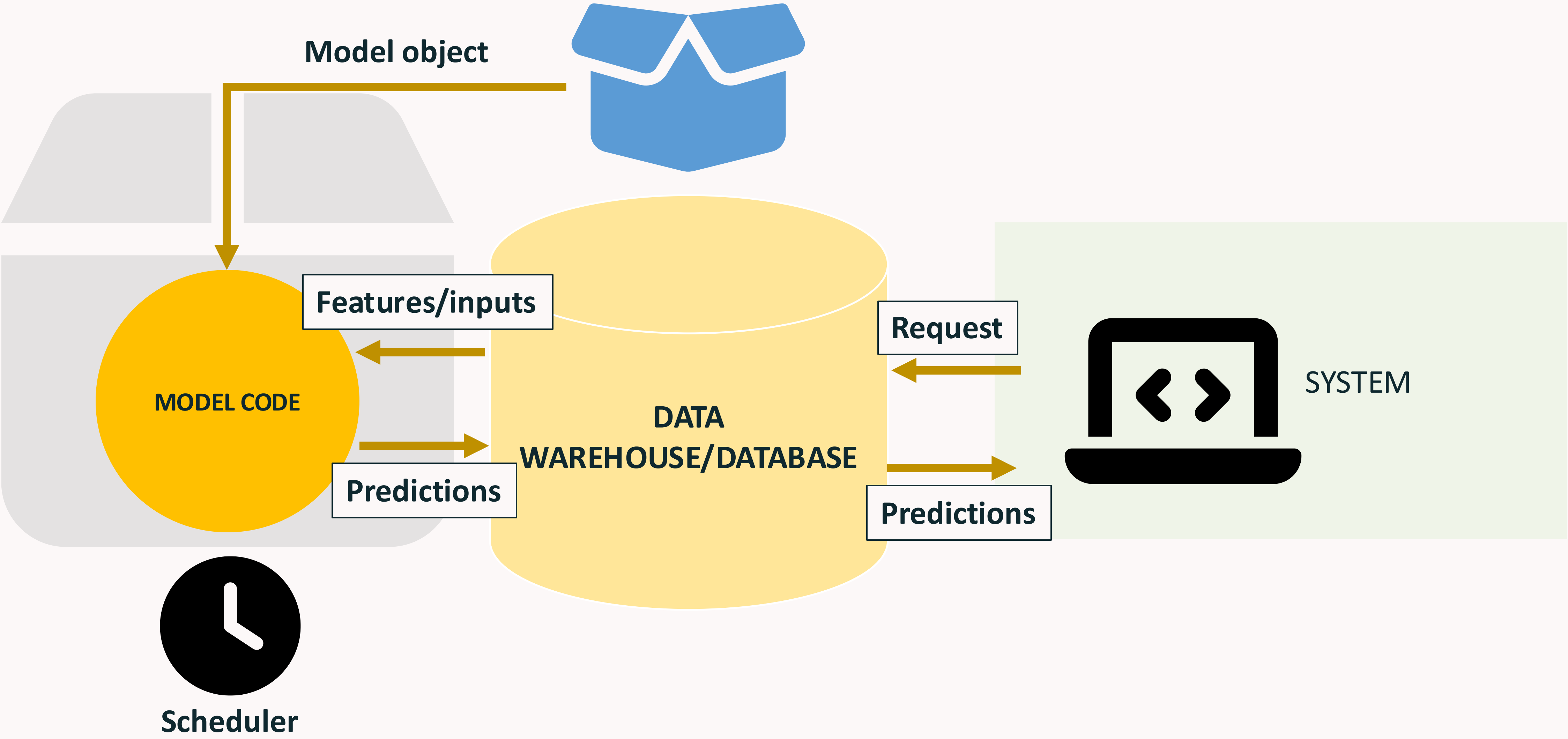


- Push compute to device
- Can run where internet is unstable
- Network latency a non-issue (no data transfer)
- Data is (more) secure
- Limited by resources on device

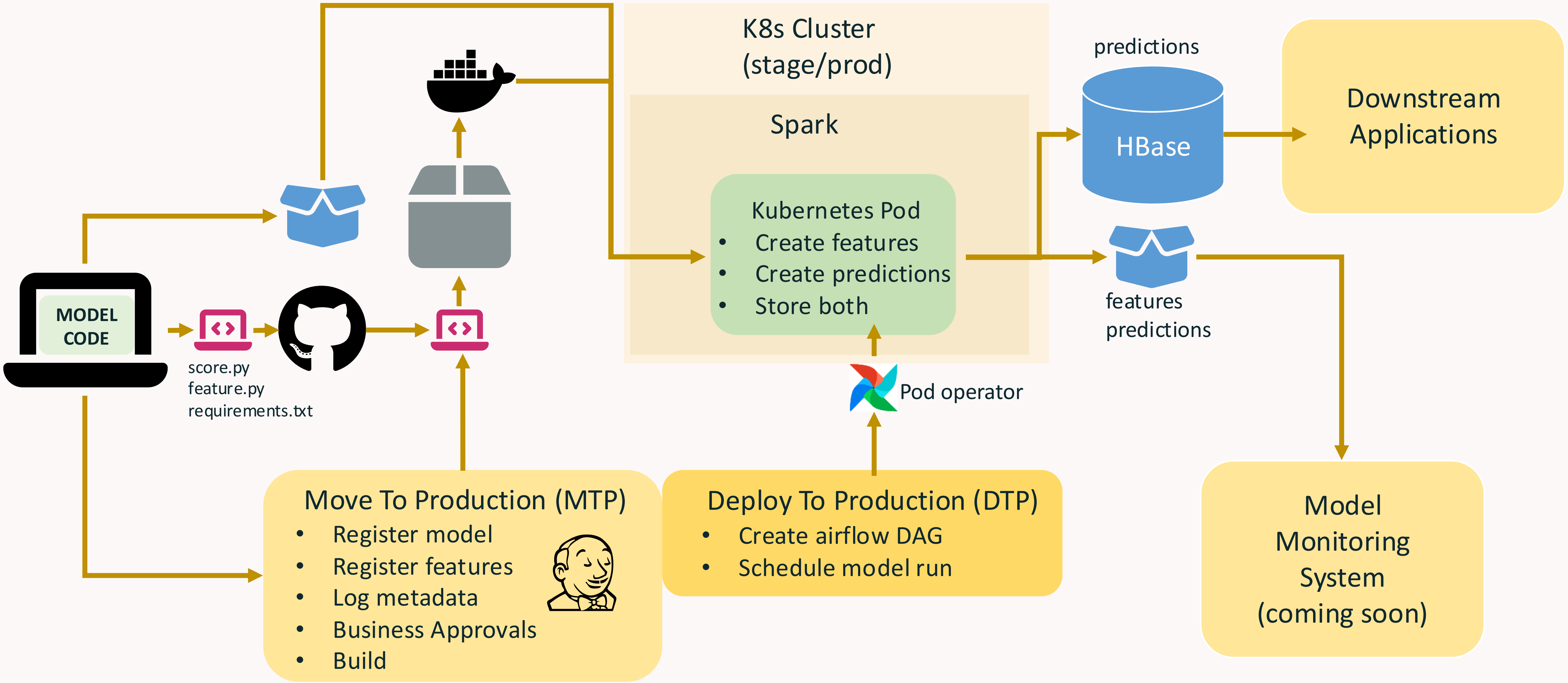
- No need for on-prem infrastructure
- Scalable
- Heavy costs
- Data can be breached

Batch Deployments

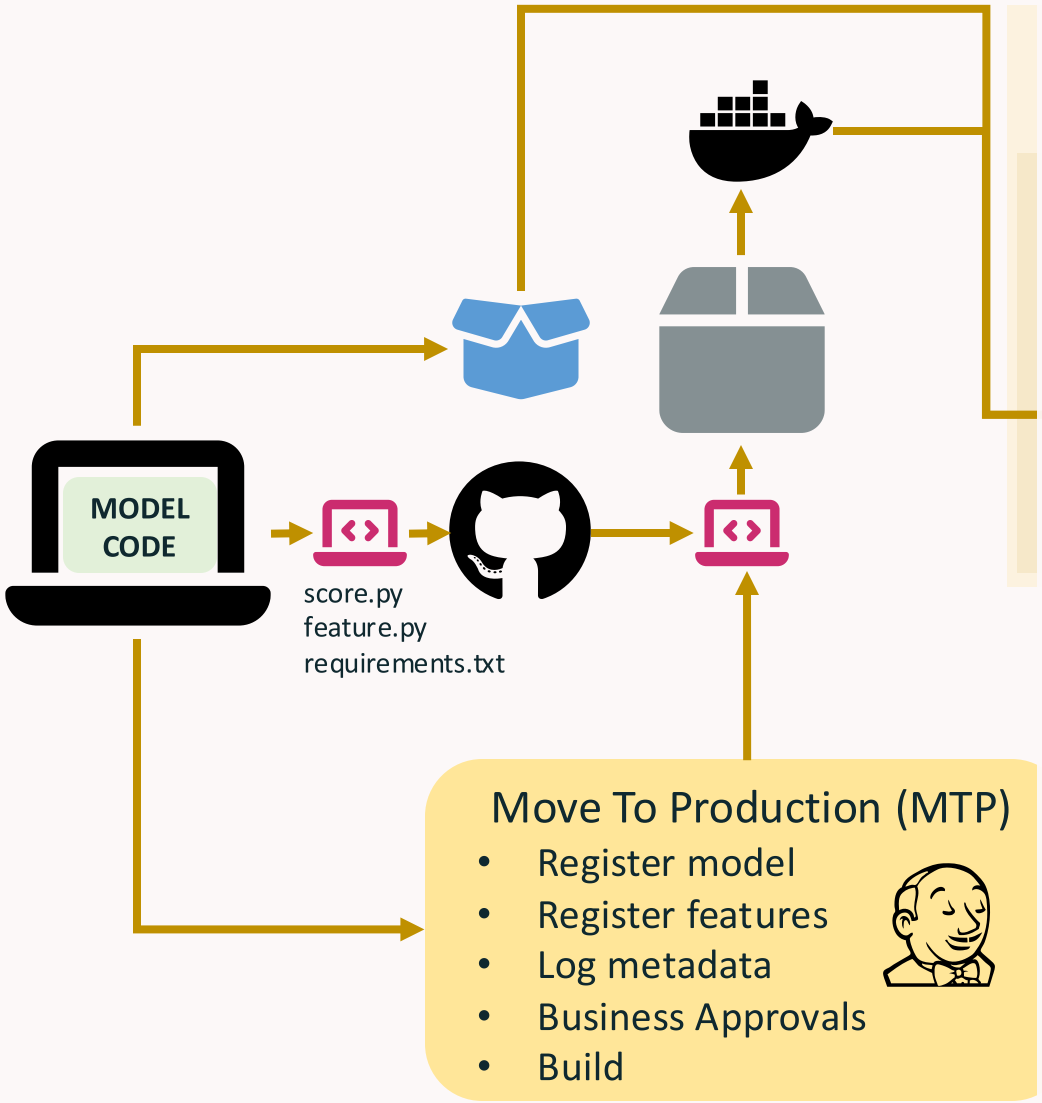
Generic Batch Architecture



Sample Batch Architecture



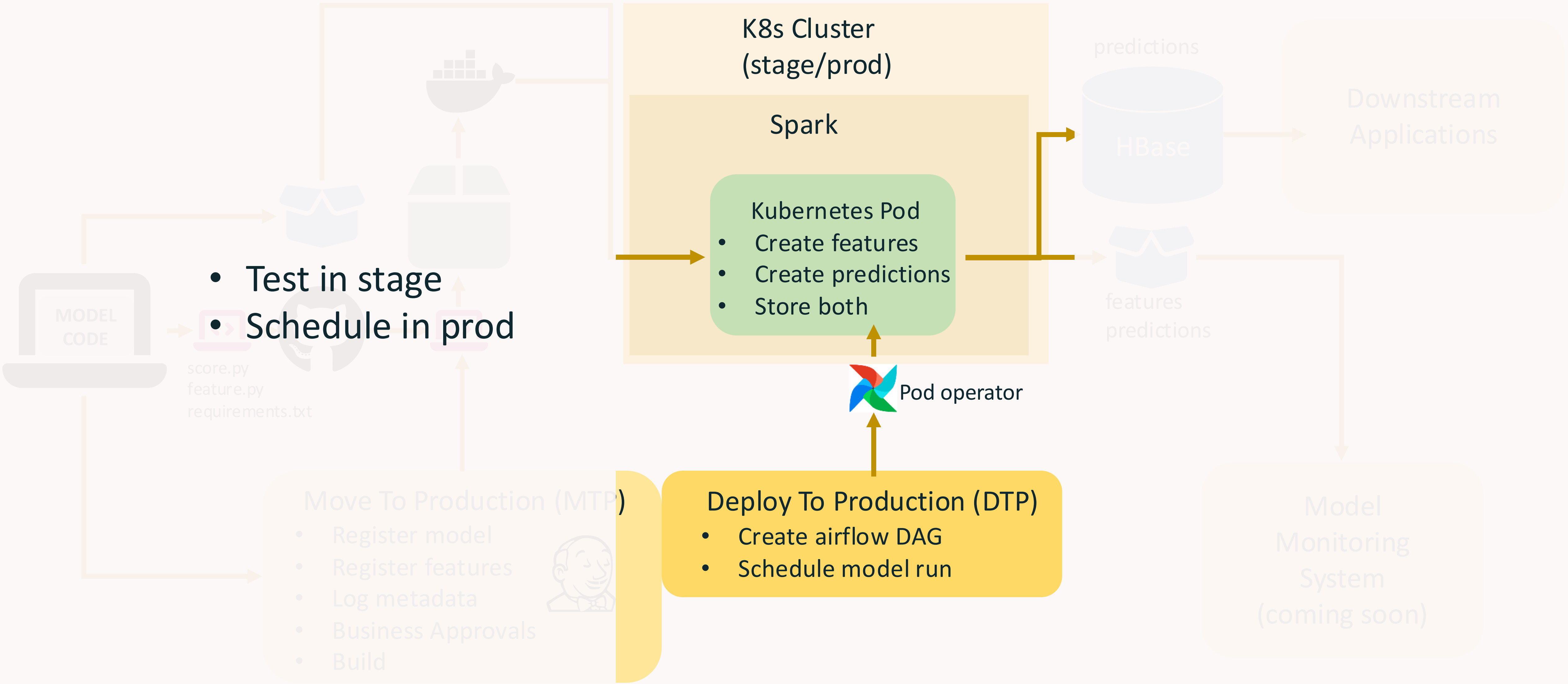
Sample Batch Architecture



- Run experiments
- Write prediction pipeline
- Register artifacts
  - Model
  - Metadata
  - Pipeline

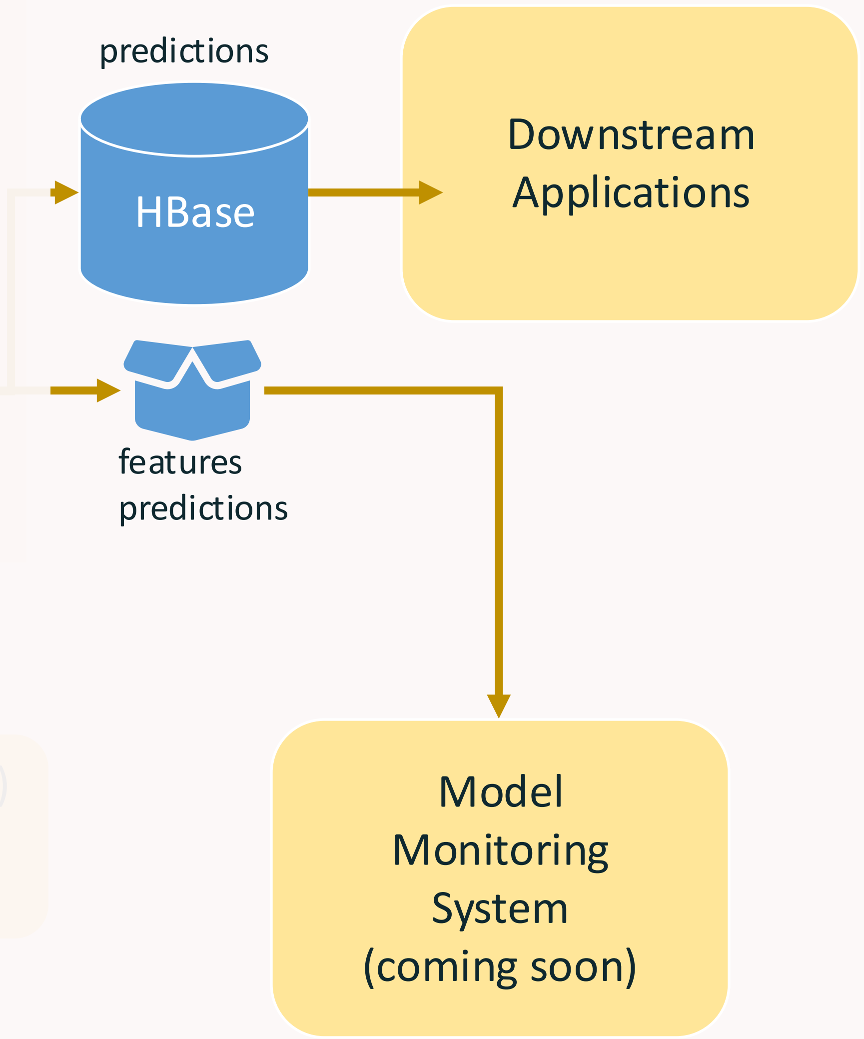
Sample Batch Architecture

- Test in stage
- Schedule in prod

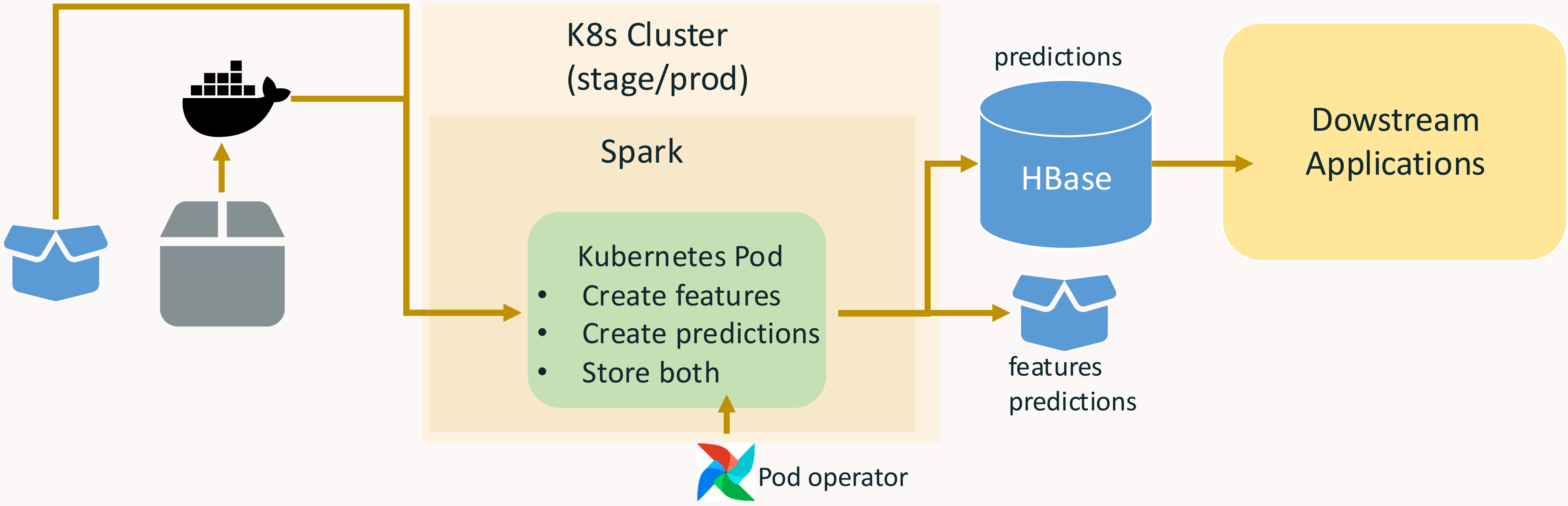


Sample Batch Architecture

- Push scores to downstream applications for Operations
- Push features/scores to "monitoring" system



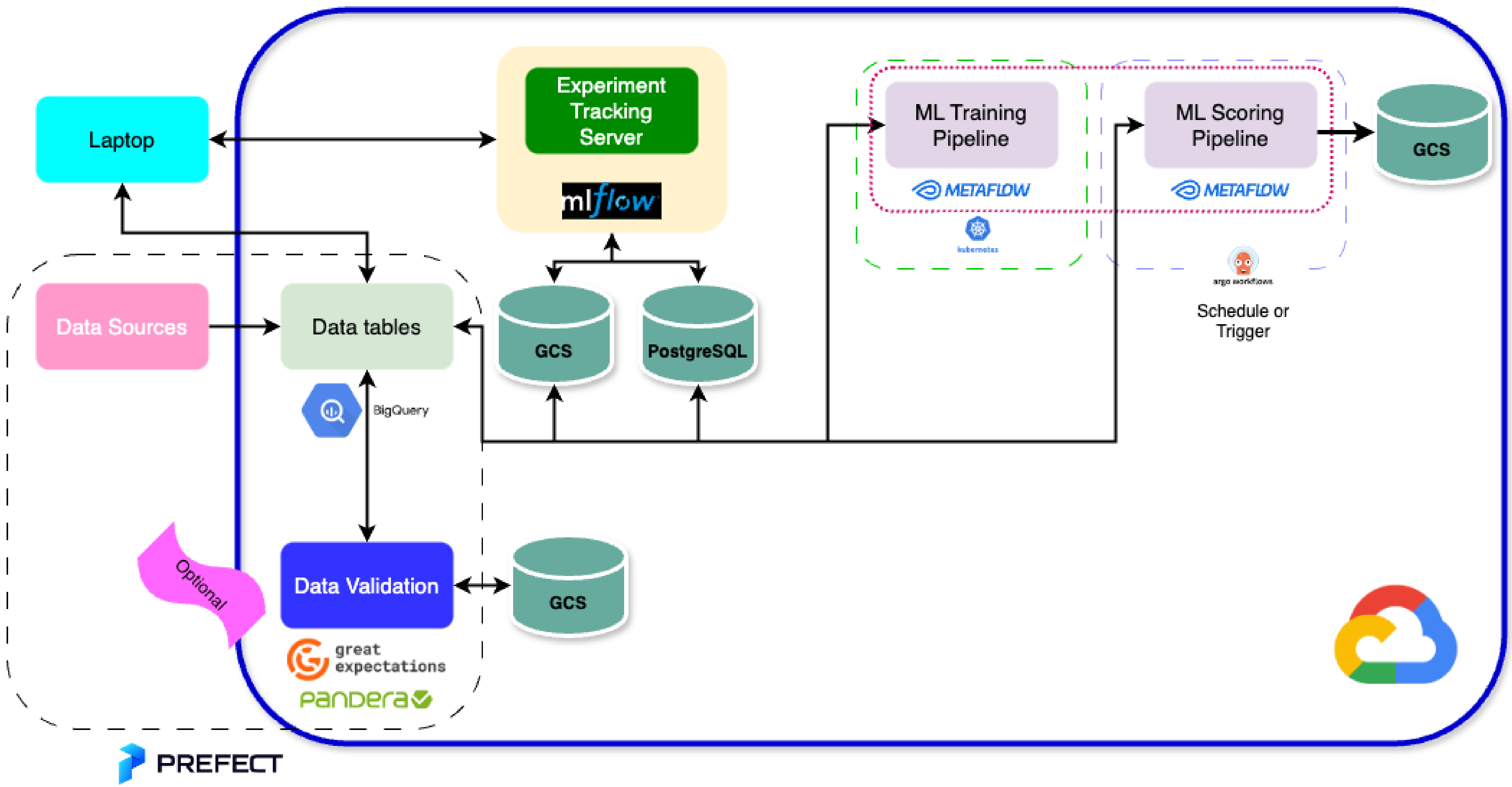
Sample Batch Architecture



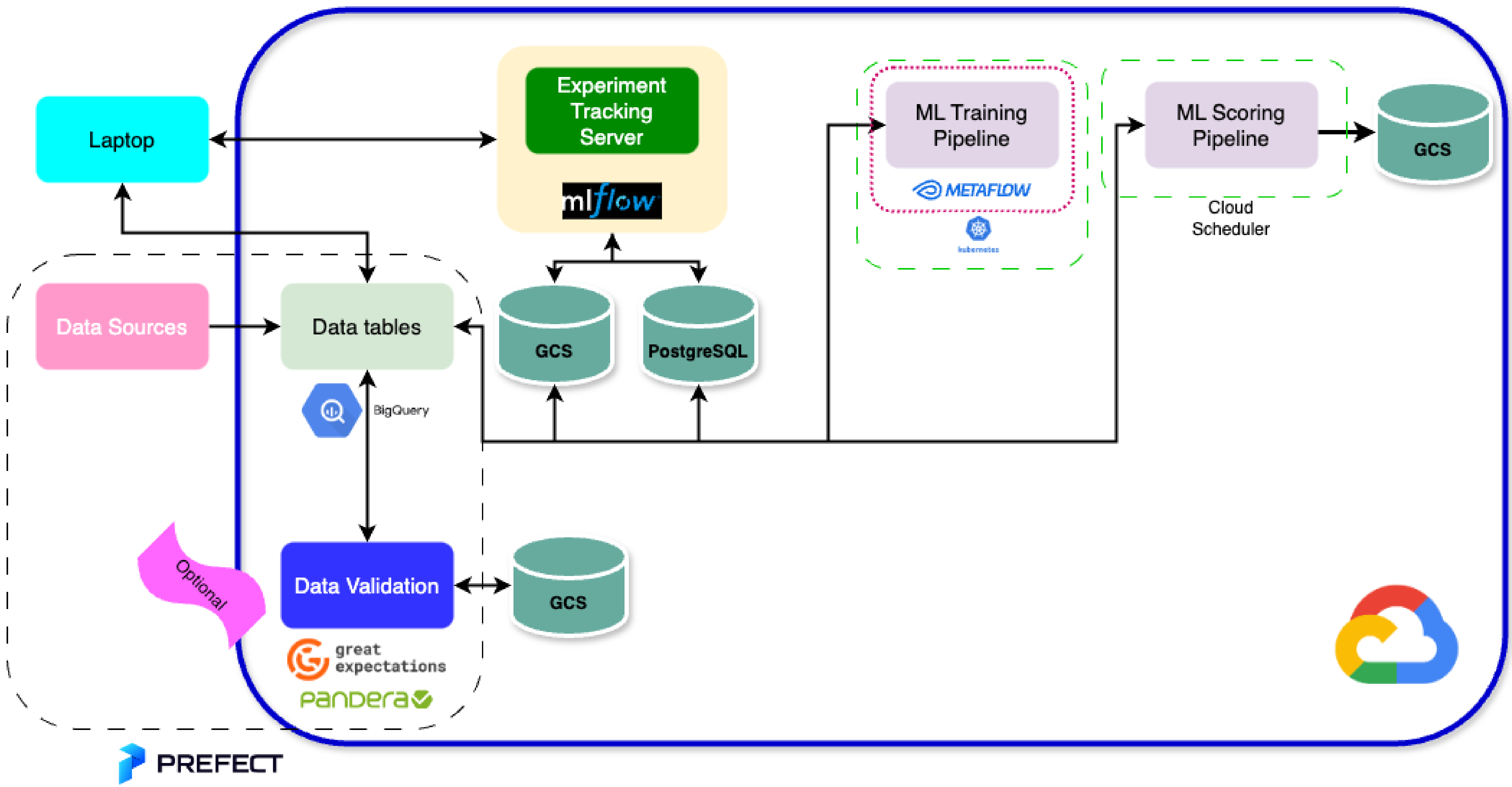
Batch Deployment Process – Your Project

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

@conda_base(libraries={'numpy':'1.23.5', 'scikit-
learn':'1.2.2'}, python='3.9.16')
@schedule(hourly=True)
class ClassifierPredictFlow(FlowSpec):

    @step
    def start(self):
        #import data

    @step
    def import_model(self):
        # import model

    @step
    def scoring(self):
        # get predictions

    @step
    def end(self):
        #store predictions


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

- Can just use Metaflow (or alternative), scheduled in Argo Worklows (or alternative)
  - At scheduled time:
    - Ingest data (or features) from data store
      - Need separate workflows for making scoring data available
    - Process data
    - Load model from model registry
    - Score data
    - Do any post-processing necessary
    - Output results to data store for downstream processing

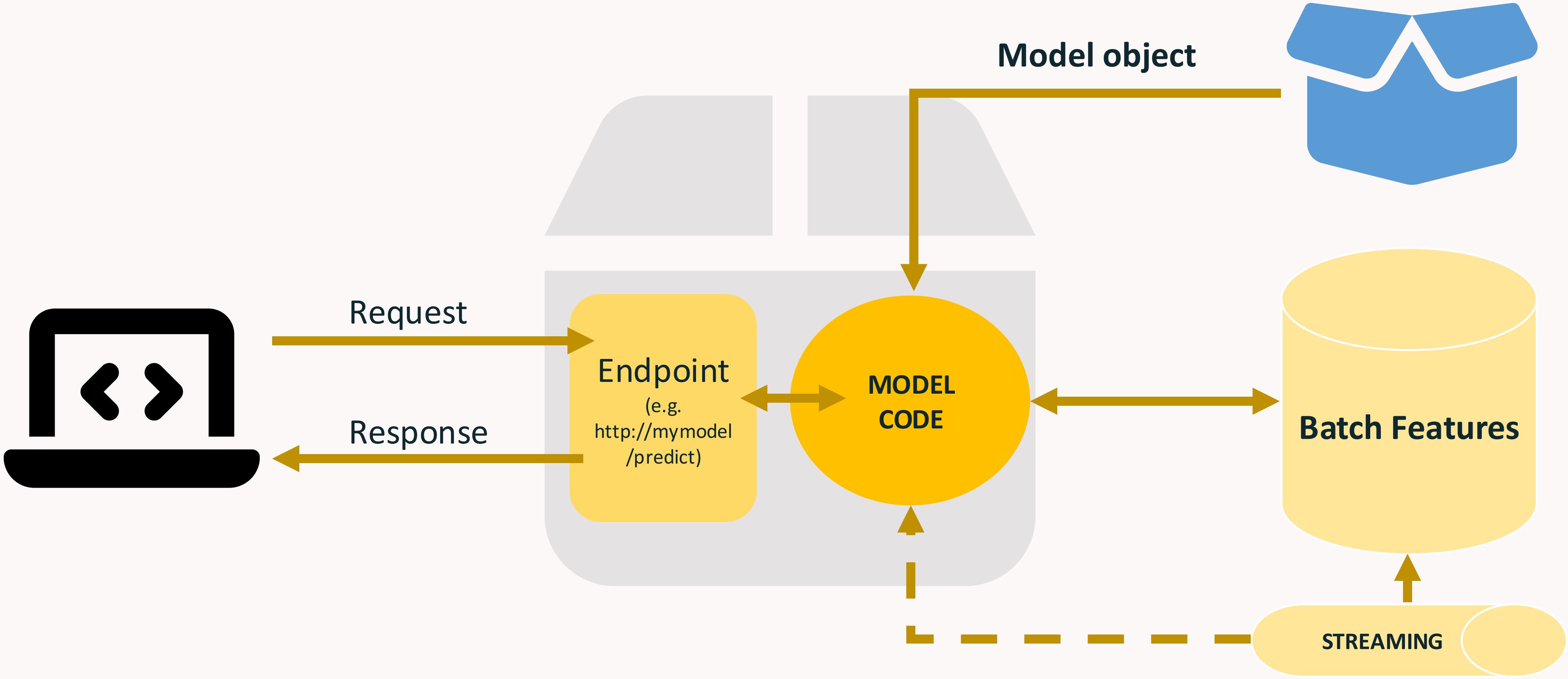


Laptop
Experiment Tracking Server
mlflow
ML Training Pipeline
METAFLOW
kubernetes
ML Scoring Pipeline
METAFLOW
GCS
argo workflows
Schedule or Trigger
Data Sources
Data tables
GCS
PostgreSQL
BigQuery
Optional
Data Validation
GCS
great expectations
pandera
PREFECT

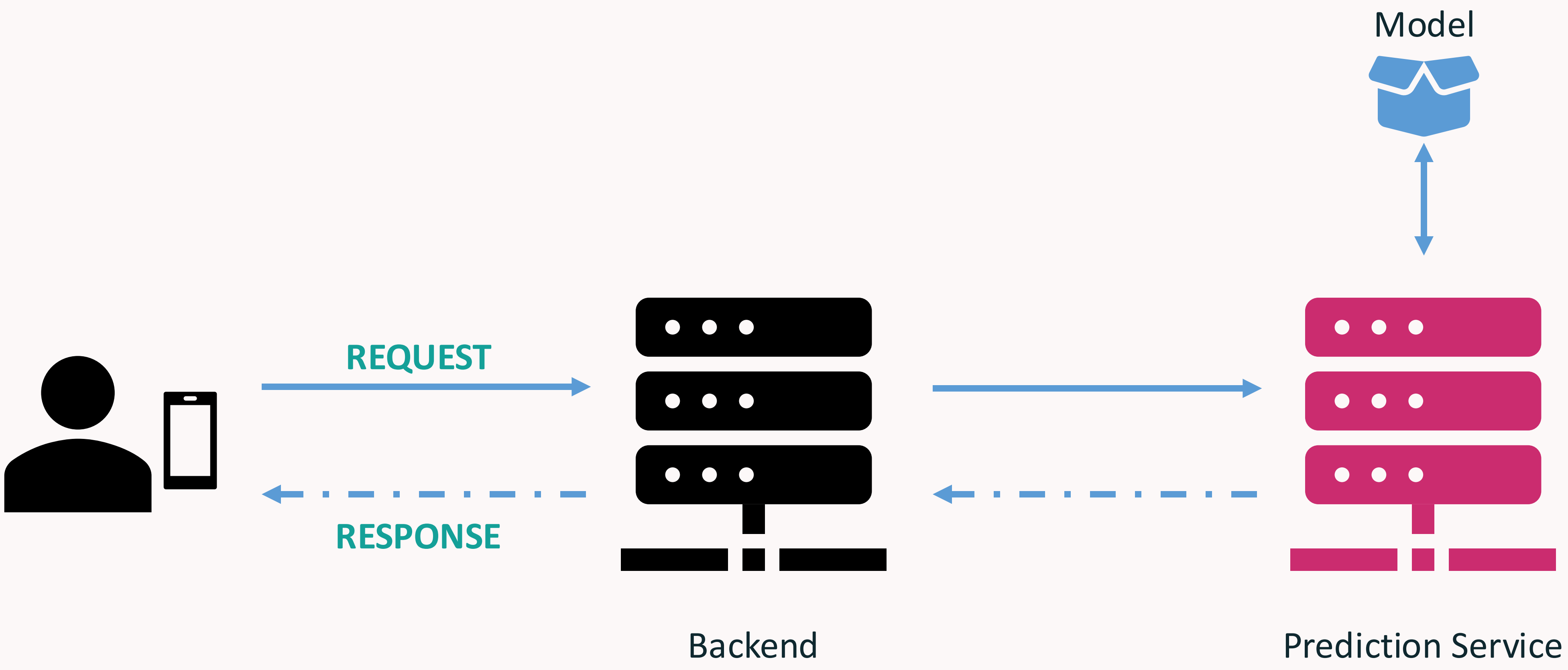

Laptop
Experiment Tracking Server
mlflow
ML Training Pipeline
METAFLOW
kubernetes
ML Scoring Pipeline
GCS
Cloud Scheduler
Data Sources
Data tables
GCS
PostgreSQL
BigQuery
Optional
Data Validation
GCS
great expectations
pandera
PREFECT

Real-time Deployments

Generic Real-time Architecture



Online Web Service



Real-time Deployment Process

- REST (REpresentational State Transfer)
    - Uses JSON
    - HTTP 1.1 (textual)
    - Request-Response

- gRPC (high performance Remote Procedure Call)
    - Uses protocal buffers (protobuf - more compressed than JSON)
    - HTTP/2.0
    - Bi-directional streaming capabilities

Uniform Resource Identifier (URI)

`http://127.0.0.1:8000/predict/model=a&shadow=true`

scheme

host or domain

port or endpoint

path to resource

query

Create Read Update and Delete (CRUD)

http://127.0.0.1:8000/predict/model=a&shadow=true

Four common methods to execute on the **resource**

- **GET** (Read): retrieve data from a resource.
  - Read-only.
  - Idempotent – multiple requests get the same resource.
- **POST** (Create): sends data to create a new resource.
  - Not idempotent – multiple requests duplicate the resource.
- PUT (Update): create/update a resource. Idempotent.
- DELETE (Delete): delete a resource. Idempotent.

Create Read Update and Delete (CRUD) for ML

http://127.0.0.1:8000/predict/model=a&shadow=true

Four common methods to execute on the **resource**

- **GET** (Read): retrieve model health check.
  - Read-only.
  - Idempotent – multiple requests get the same resource.
- **POST** (Create): sends data to request new predictions.
  - Not idempotent – multiple requests duplicate the resource.

Curl to Send API Requests

Method

Headers: info about event, such as format sent and received

```
curl -X 'POST' \
'http://127.0.0.1:8000/predict' \
-H 'accept: application/json' \
-H 'Content-Type: application/json' \
-d '{"reddit_comment": "Useless comment, you should flag it for removal"}'
```

Data: data to send. Also referred to as the body of the request to send.

FastAPI to Create the API

- Easy to use
- Performant
- Data validation via pydantic library
- Autogenerated docs

```
from fastapi import FastAPI
import uvicorn

app = FastAPI(
     title="Reddit Comment Classifier",
     description="Classify Reddit comments as either
1 = Remove or 0 = Do Not Remove.",
     version="0.1",
)

# Defining path operation for root endpoint
@app.get('/')
def main():
     return {'message': 'This is a model for
classifying Reddit comments'}

# Defining path operation for /name endpoint
@app.get('/{name}')
def hello_name(name : str):
     return {'message': f'Hello {name}'}
```

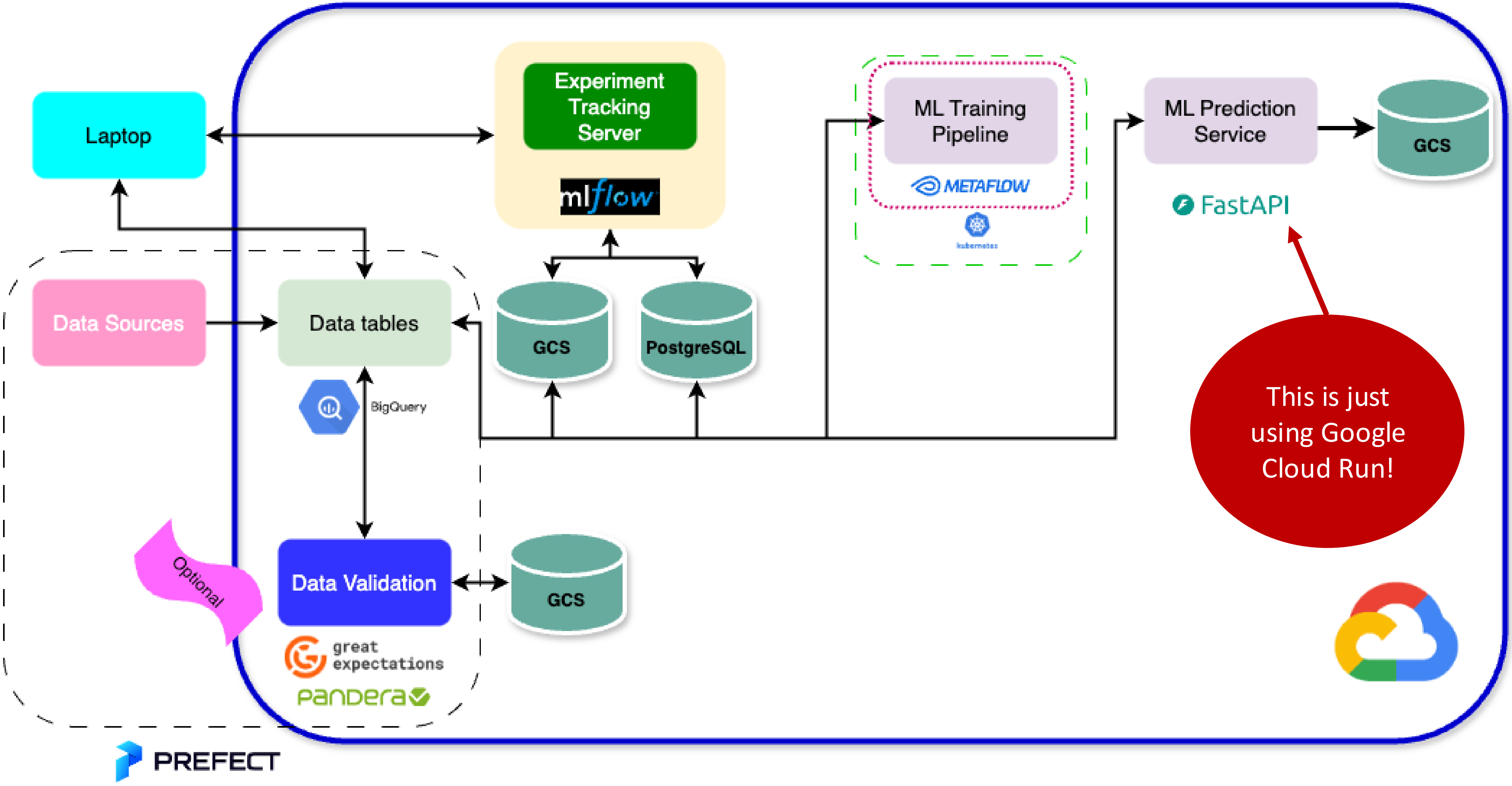
Launch API Using uvicorn and Make Requests

- uvicorn: minimal server/application interface
  - Good for development, prototyping

```
uvicorn <filename>:app --reload
```

- Use either curl or python with requests library to make requests

```
import requests

comment =
{'reddit_comment':'Testing a
comment.'}

url =
'http://127.0.0.1:8000/predict'
response = requests.post(url,
json=comment)
print(response.json())
```

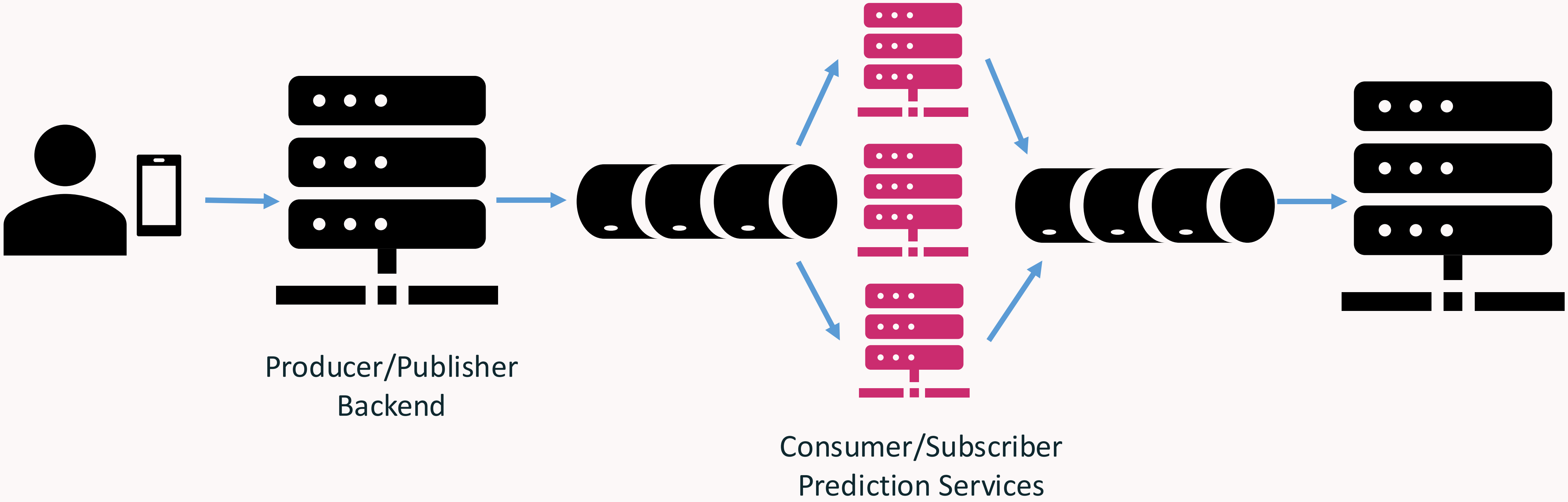


Laptop
Experiment Tracking Server
mlflow
ML Training Pipeline
METAFLOW
kubernetes
ML Prediction Service
GCS
FastAPI
Data Sources
Data tables
GCS
PostgreSQL
BigQuery
This is just using Google Cloud Run!
Optional
Data Validation
GCS
great expectations
pandera
PREFECT

Real-time Deployment Demo

Streaming Deployment

Online ”Streaming”



Options

- Kafka
- AWS Kinesis
- GCP Pub/Sub

Web Apps

Simple Web Apps

Simple Web Apps

Streamlit

- **Goal**: to create a simple prototype UI to quickly get users interacting with your model
- Streamlit is a simple to use python library for creating “data apps”
  - No knowledge of HTML, CSS, or JS necessary
  - Can deploy on the Streamlit Community Cloud, Heroku, Google App Engine, Azure, AWS, and Kubernetes

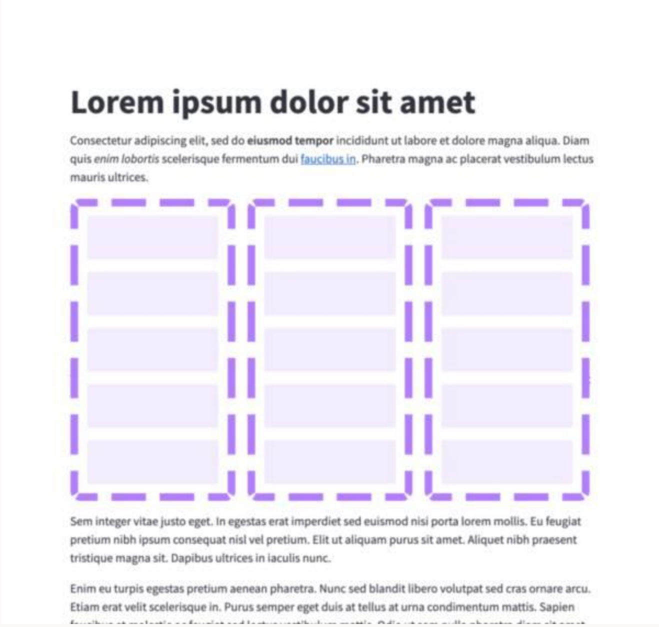
Considerations

- Streamlit is for prototyping, not production
- By default, each change made in the app (e.g. clicking a checkbox) causes the entire script to be rerun top to bottom
  - This can be alleviated using things like `@st.session_state`, `@st.cache_data` or `@st.cache_resource`

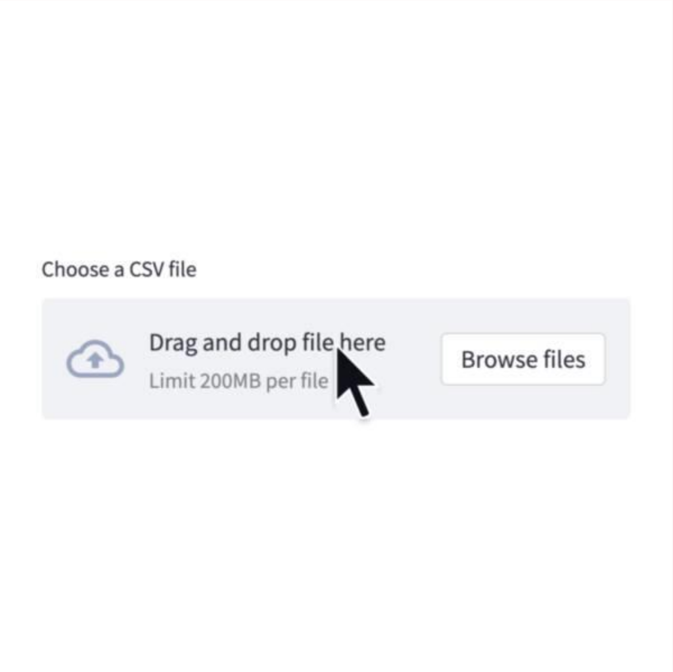
Prototyping

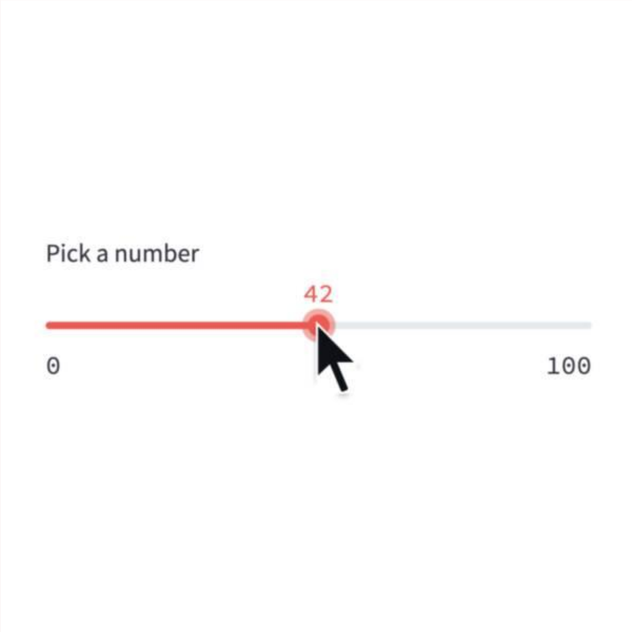
- Take a step back, consider the persona, before beginning to code
- Don’t overdo it for a prototype, a simple functional design will work fine
- What does your application really need?
  - Does it need multiple pages and tabs (lots of clicks)?
  - Does it really need all of these charts (useless information and clutter)?
  - Keep it simple, smartypants.
- Remember your color theory, data viz methods, etc.
  - Good resource here https://clauswilke.com/dataviz/index.html

Prototyping – UI Layout

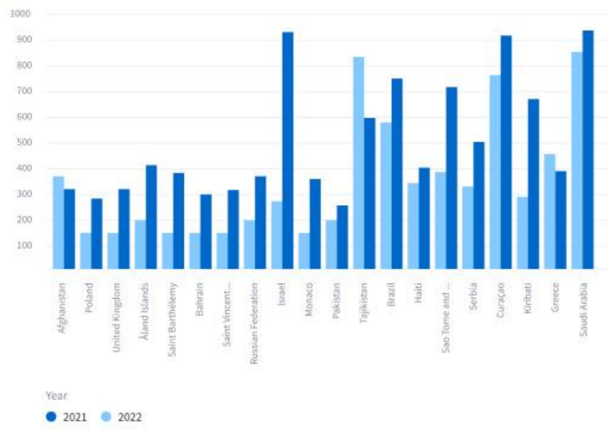


Prototyping – UI Inputs





Prototyping – UI Outputs



	A	B	C	D	E
0	1.000000	1.329212	nan	-0.316280	-0.990810
1	2.000000	-1.070816	-1.438713	0.564417	0.295722
2	3.000000	-1.626404	0.219565	0.678805	1.889273
3	4.000000	0.961538	0.104011	-0.481165	0.850229
4	5.000000	1.453425	1.057737	0.165562	0.515018

$$B' = -\nabla \times E$$
$$E' = \nabla \times B - 4\pi j$$

$$G_{\mu\nu} + \Lambda g_{\mu\nu} = \frac{8\pi G}{c^4} T_{\mu\nu}$$

$\LaTeX$

Streamlit Demo