Machine Learning Systems Design

Deployment and Monitoring

Lecture 18: Model Serving

CE 40959 Spring 2023
Ali Zarezade
SharifMLSD.github.io

Agenda

1. Stream Serving
2. Batch vs Online Serving
3. Model Serving Considerations

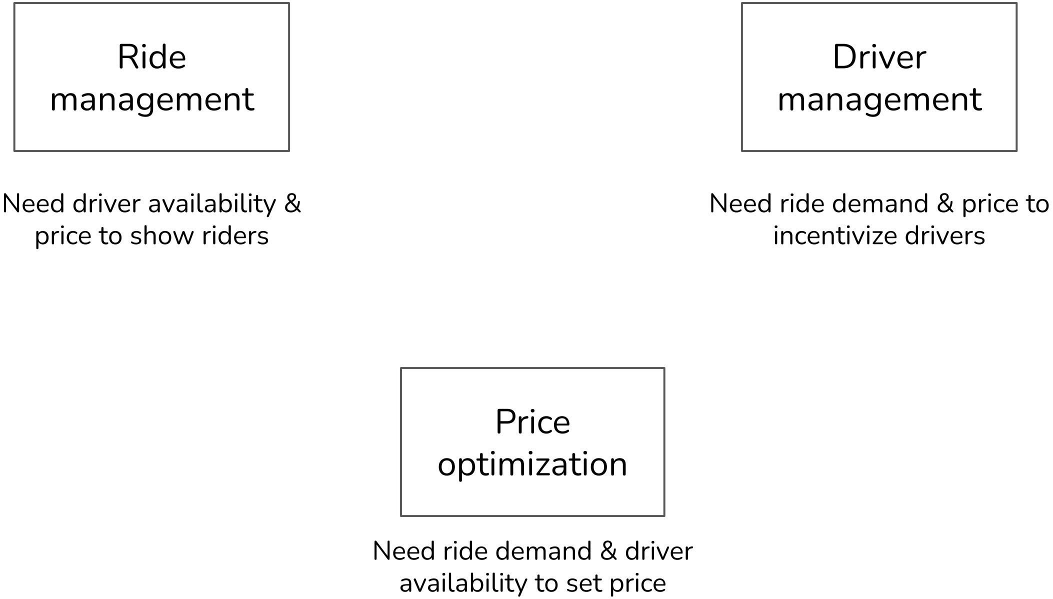
1. Stream Serving

Serving strategies

One fundamental decision you’ll have to make that will affect both your end users and developers working on your system is how it generates and serves its predictions to end users: online or batch.

- **Batch** prediction, which uses only batch features.
- **Online** prediction that uses only batch features (e.g., precomputed embeddings).
- Online prediction that uses both batch features and streaming features. This is also known as **streaming** prediction.

How to pass data between processes?



Ride management

Need driver availability & price to show riders

Driver management

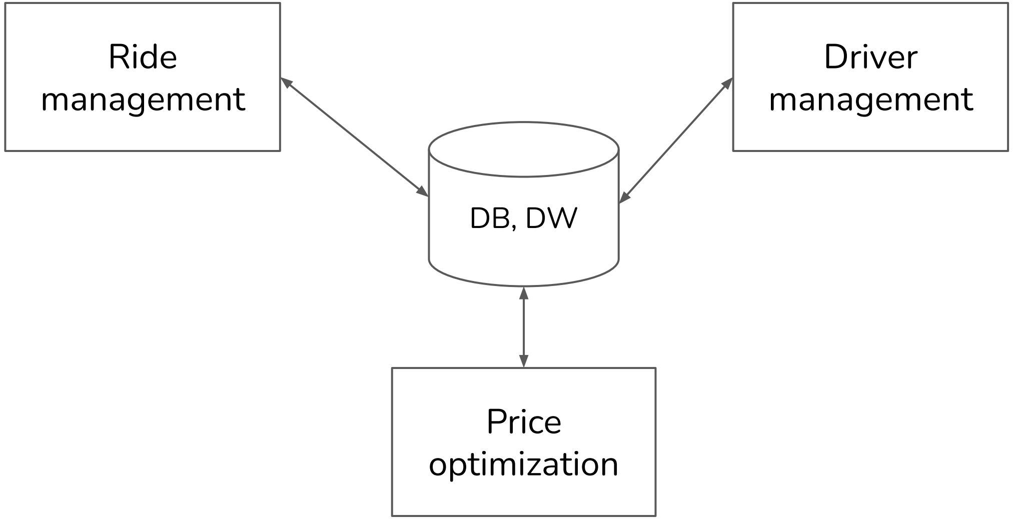
Need ride demand & price to incentivize drivers

Price optimization

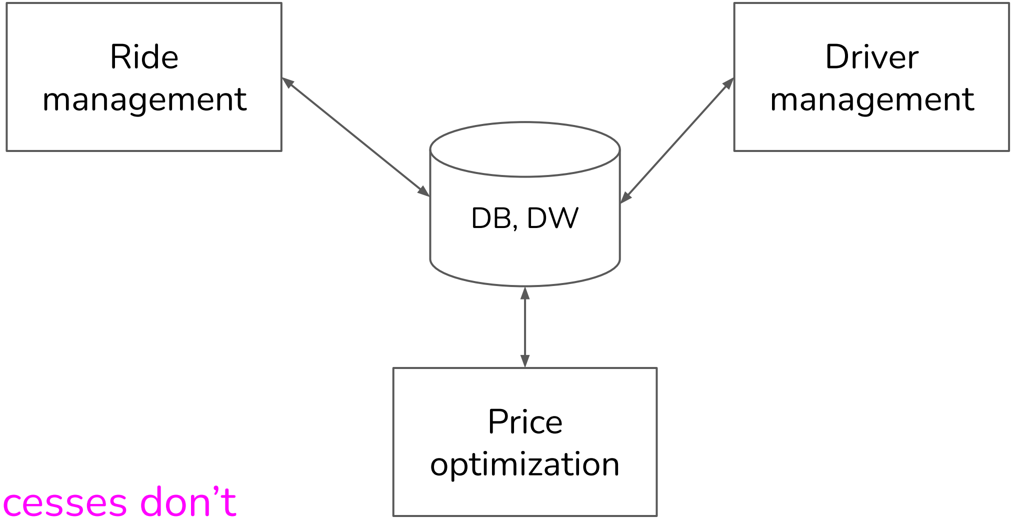
Need ride demand & driver availability to set price

A simple ride-sharing microservice

Data passing through databases

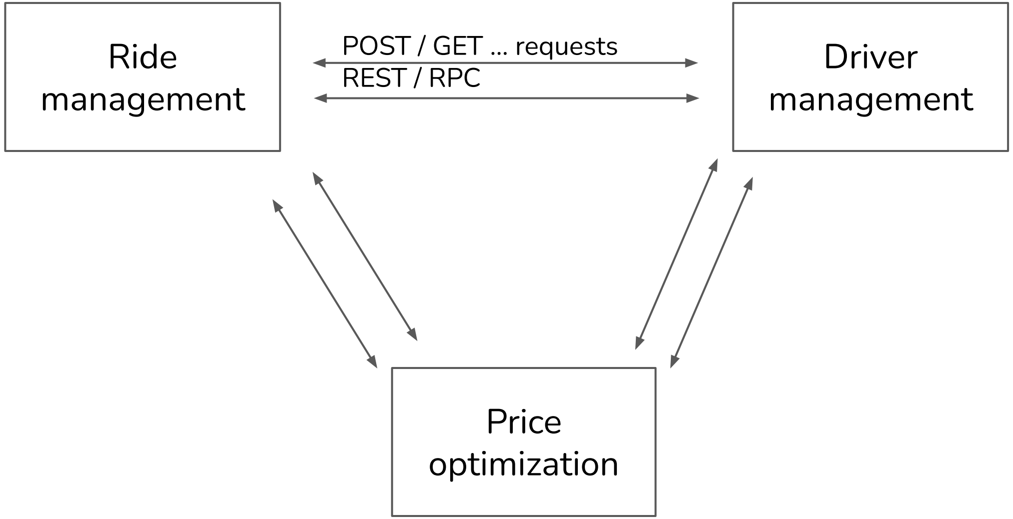


Data passing through databases



1. What if processes don’t share database access?
2. Read & write from databases can be slow

Data passing through services



Ride management

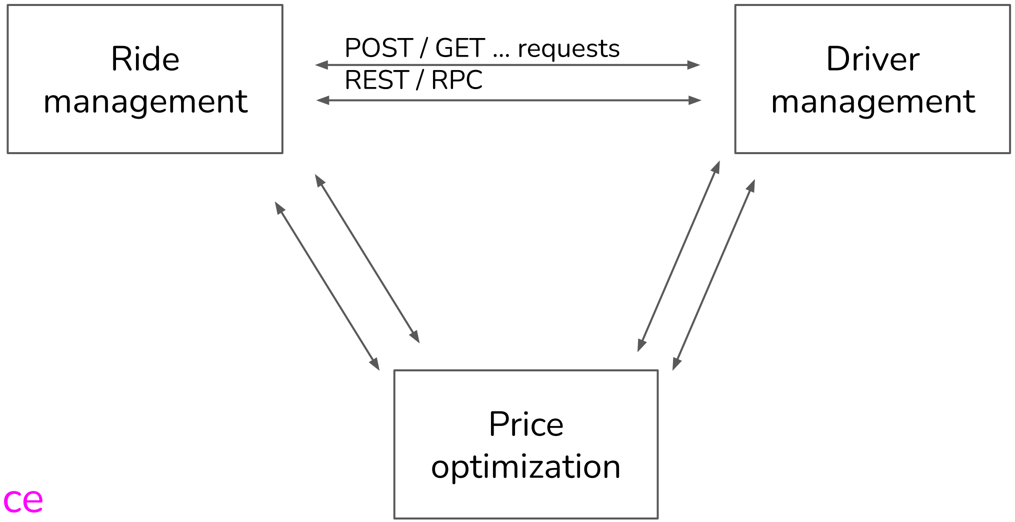
POST / GET ... requests

REST / RPC

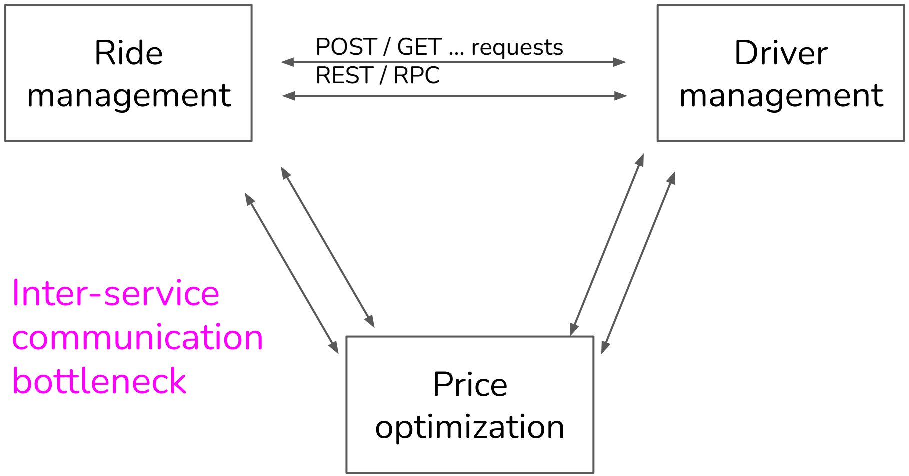
Driver management

Price optimization

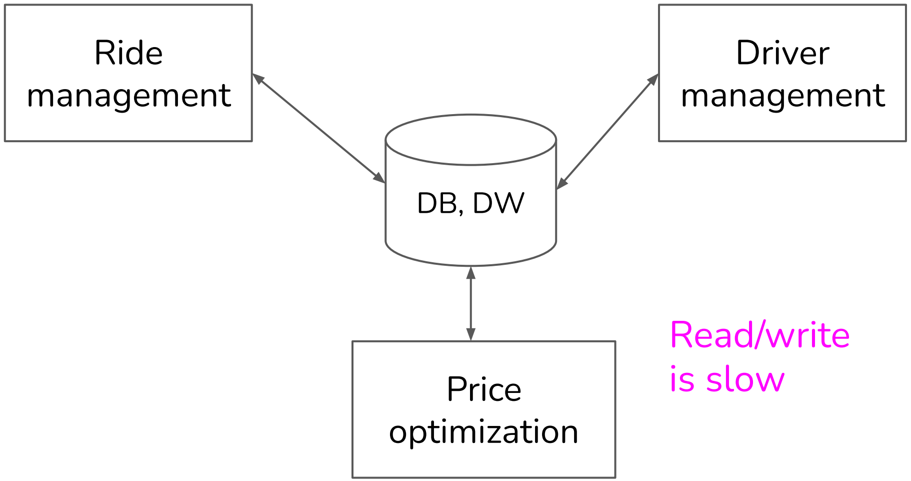
Data passing through services



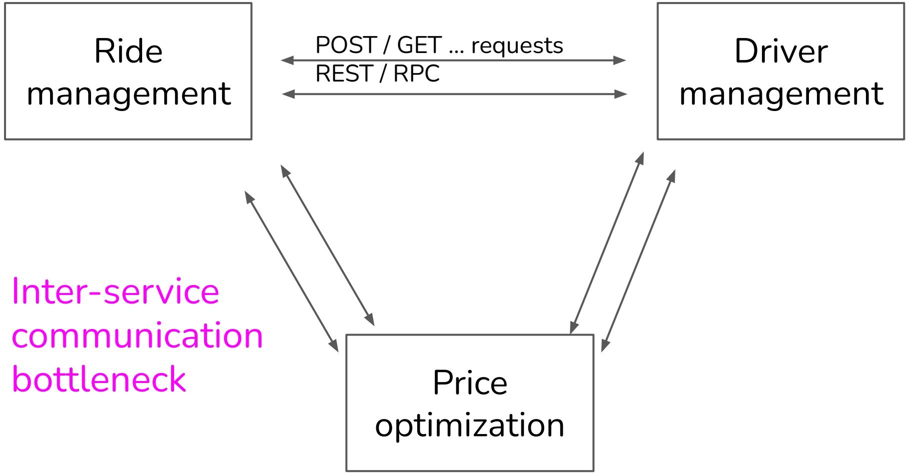
Inter-service communication can blow up



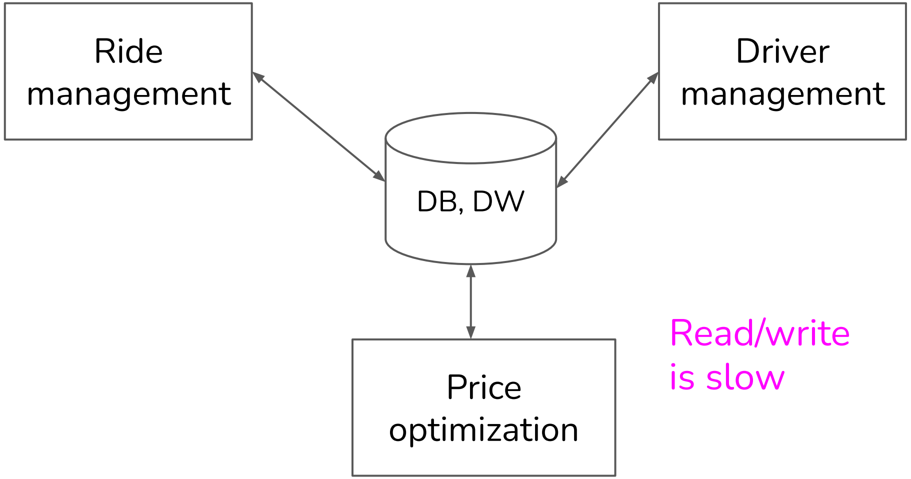
Inter-service communication bottleneck



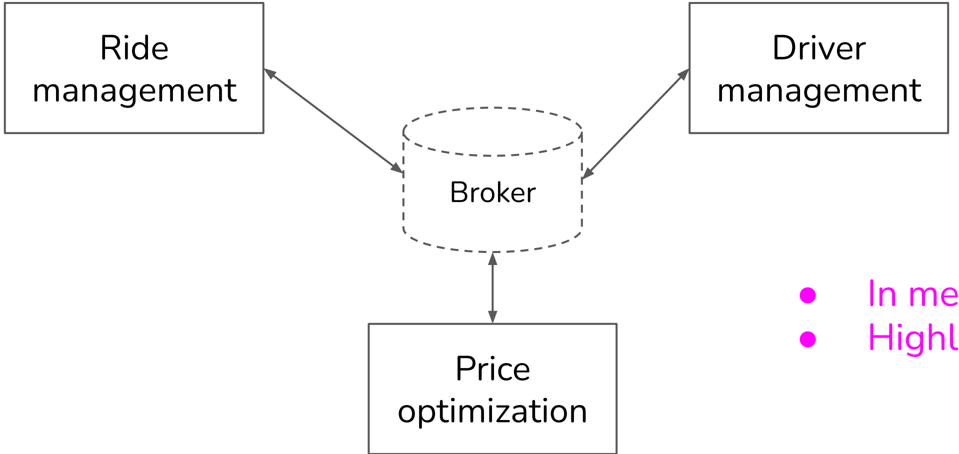
Read/write is slow



Inter-service communication bottleneck

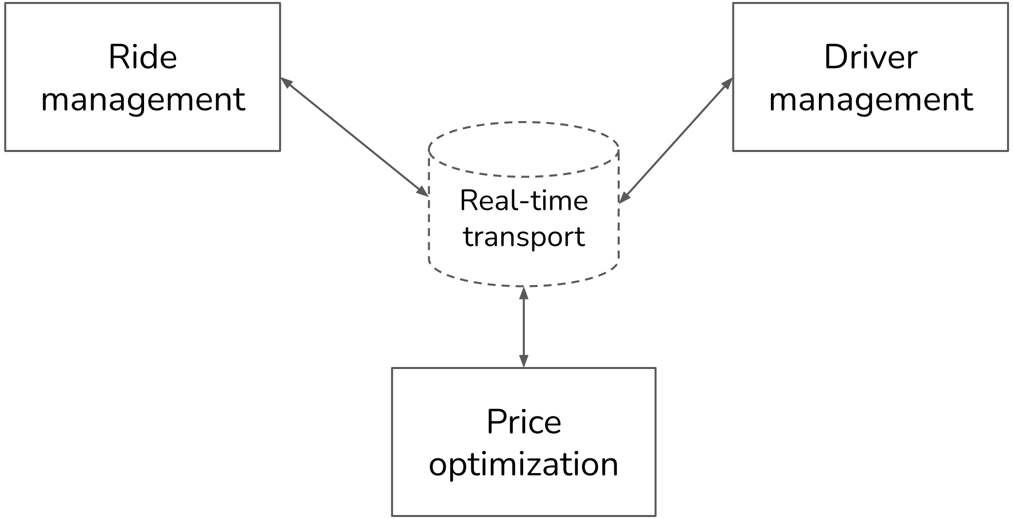


Read/write is slow



- In memory
- Highly distributed

Data passing through brokers



Need for speed: ride-sharing example

To detect whether a transaction is fraud, need features from:

- this transaction
- user’s recent transactions (e.g. 7 days)
- credit card recent transactions
- recent in-app frauds
- and so on.

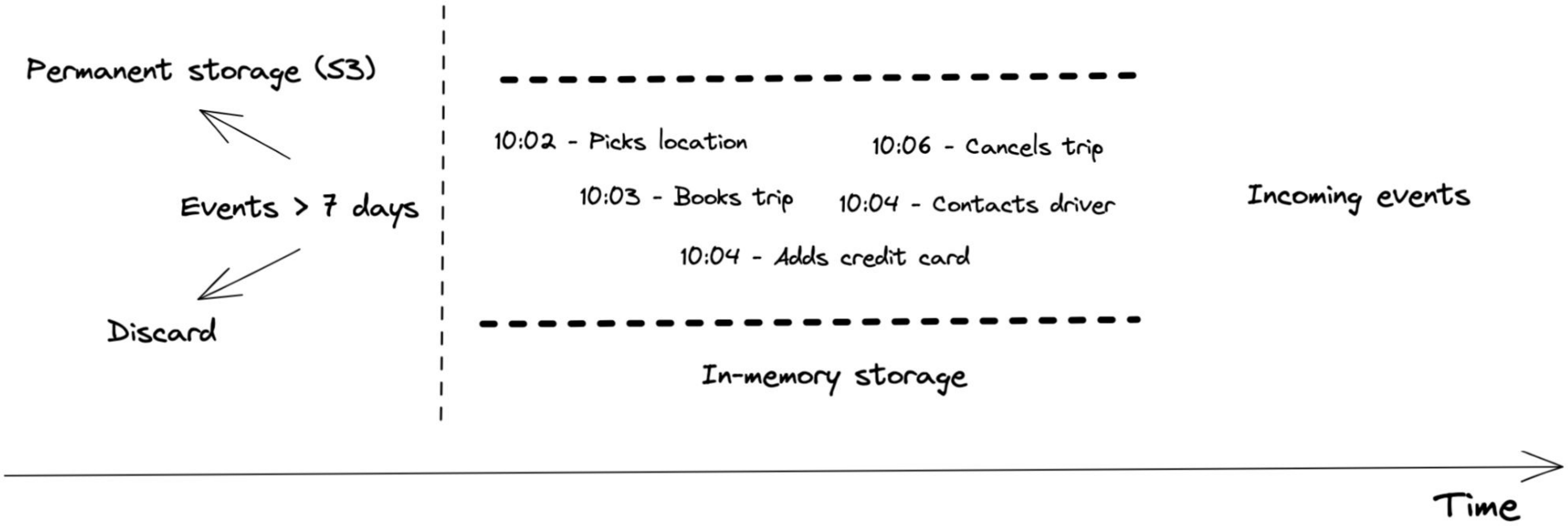
Need for speed: ride-sharing example

To detect whether a transaction is fraud, need features from:

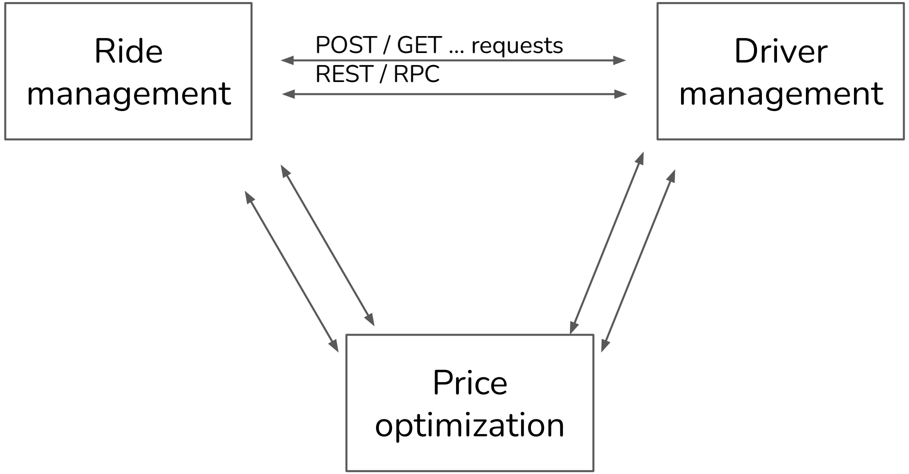
- this transaction
- user's recent transactions (e.g. 7 days)
- credit card recent transactions
- recent in-app frauds
- and so on.

How to quickly access these features?

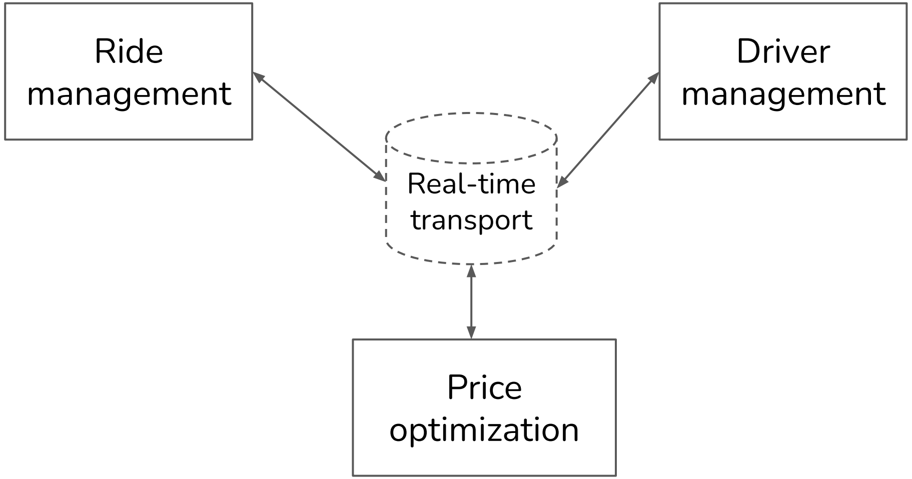
Real-time transport



Request-driven

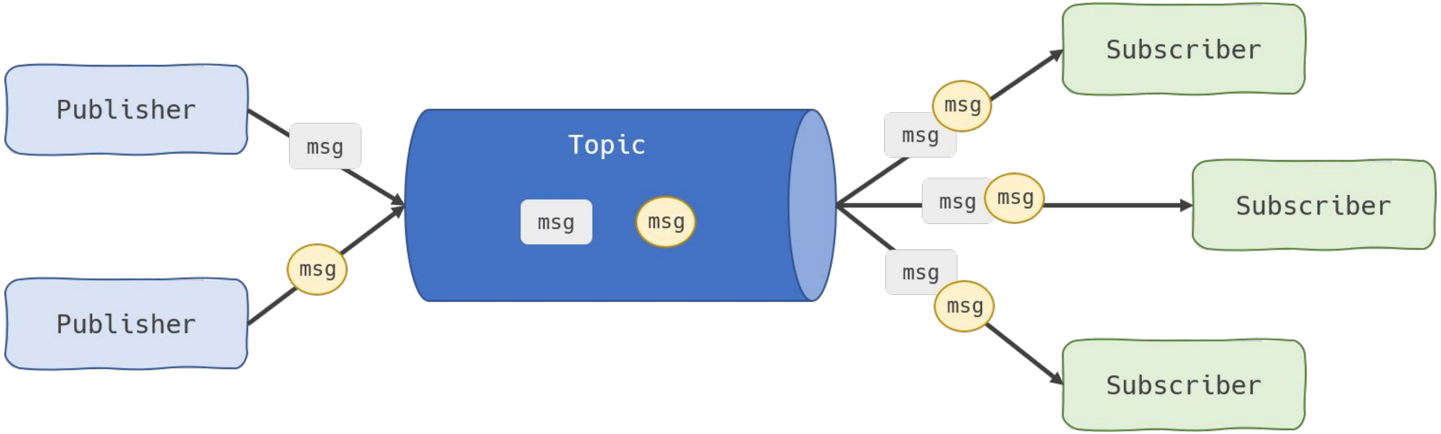


Event-driven



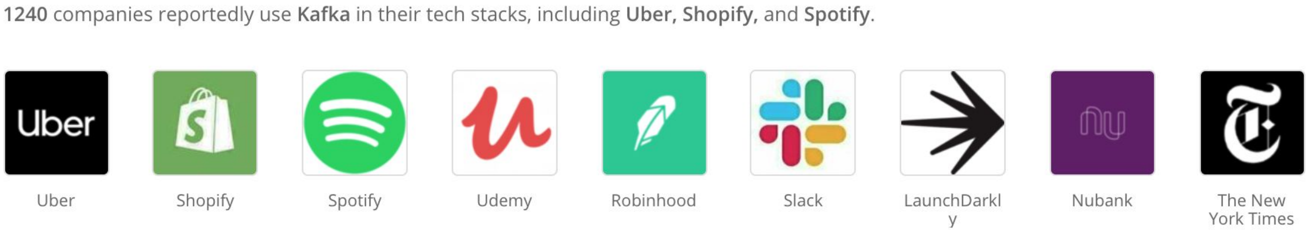
Real-time transport: pubsub

- Any service can publish to a stream [producer]
- Any service can subscribe to a stream to get info they need [consumer]



Real-time transport: pubsub, message queue, etc.

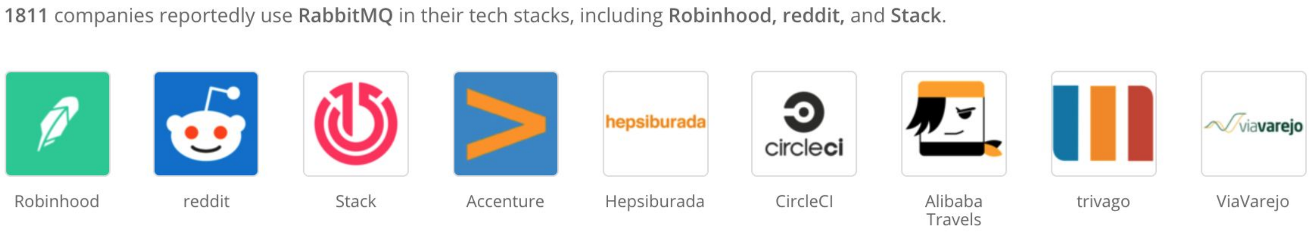
APACHE kafka®



1240 companies reportedly use **Kafka** in their tech stacks, including **Uber, Shopify,** and **Spotify**.

Uber Shopify Spotify Udemy Robinhood Slack LaunchDarkly Nubank The New York Times

RabbitMQ

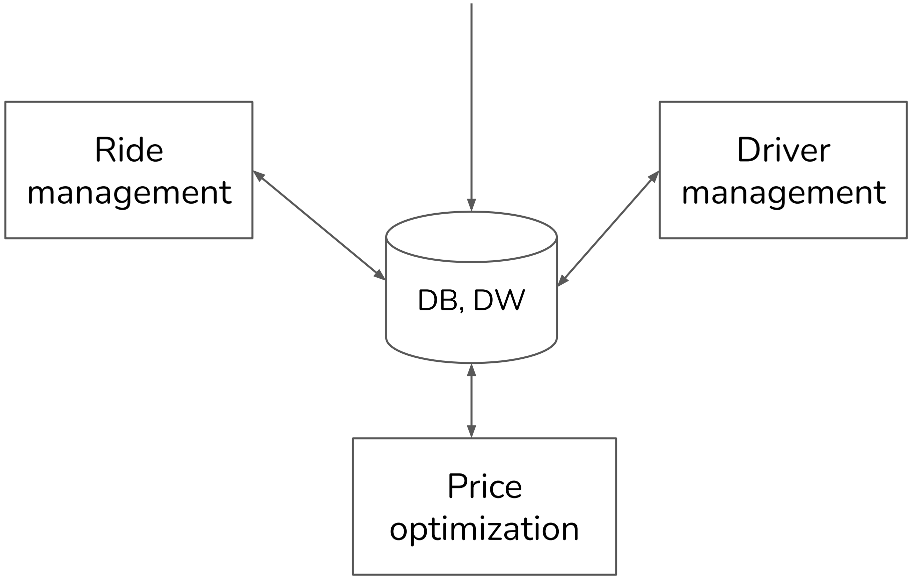


1811 companies reportedly use **RabbitMQ** in their tech stacks, including **Robinhood, reddit,** and **Stack**.

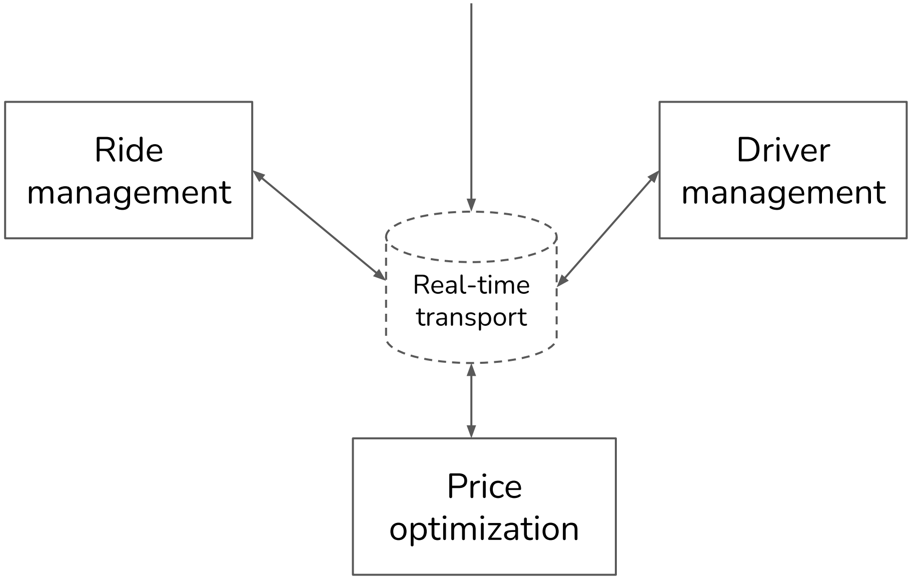
Robinhood reddit Stack Accenture Hepsiburada CircleCI Alibaba Travels trivago ViaVarejo

Batch processing vs. stream processing

Batch processing



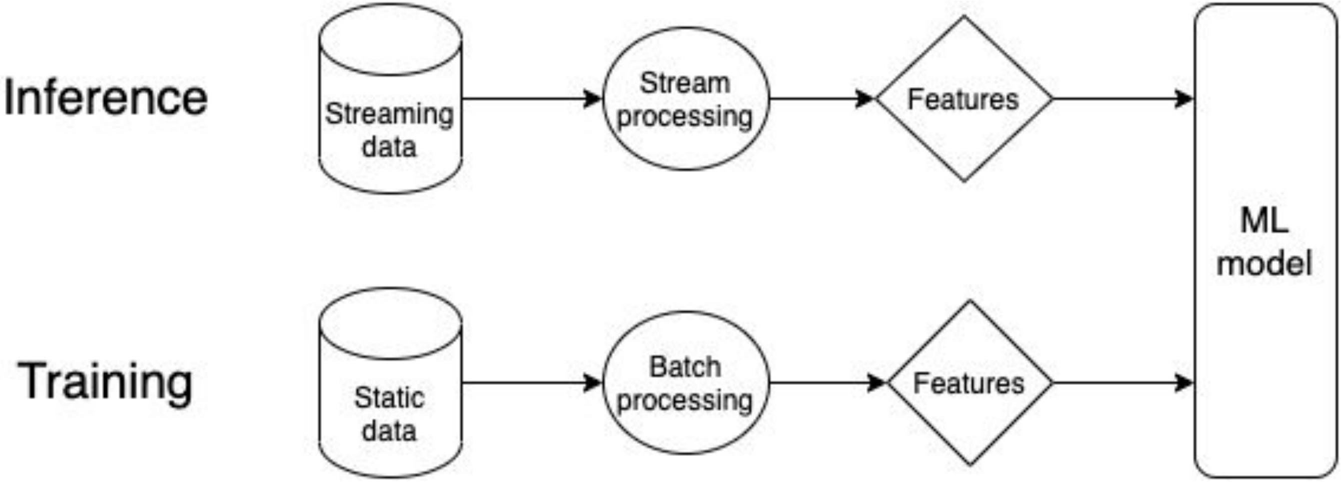
Stream processing



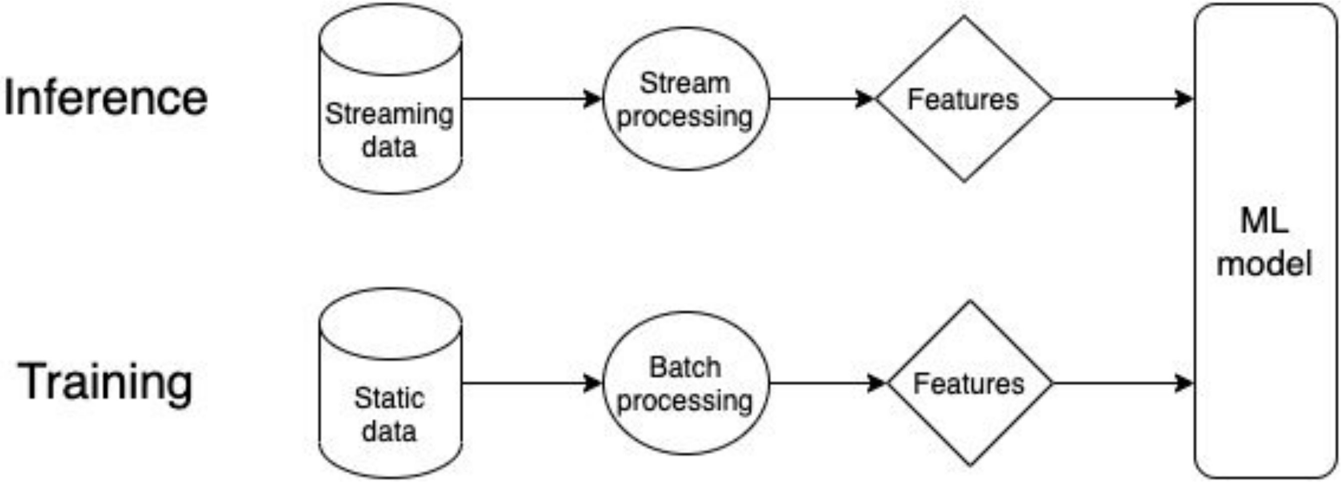
Batch processing vs. stream processing

Historical data	**Streaming data**
Databases, data warehouses	Kafka, Kinesis, Pulsar, etc.
Batch features: ● age, gender, job, city, income ● when account was created	Dynamic features ● locations in the last 10 minutes ● recent activities
Bounded: know when a job finishes	Unbounded: never finish
Processing kicked of periodically, in batch ● e.g. MapReduce, Spark	Processing can be kicked off as events arrive ● e.g. Flink, Samza, Spark Streaming

One model, two pipelines

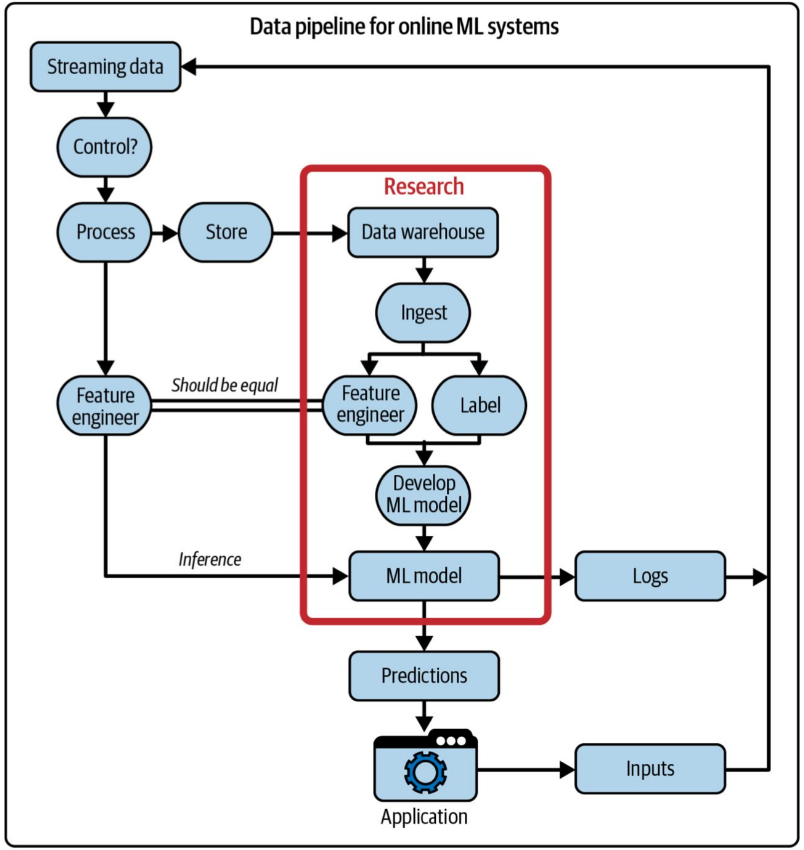


One model, two pipelines



⚠️⚠️ A common source of errors in production ⚠️⚠️

Research vs industry



Stream & batch processing

- Batch is a special case of streaming

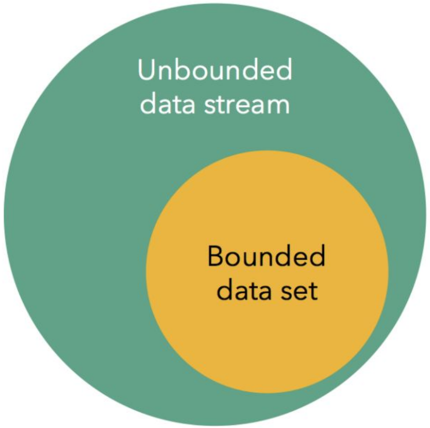
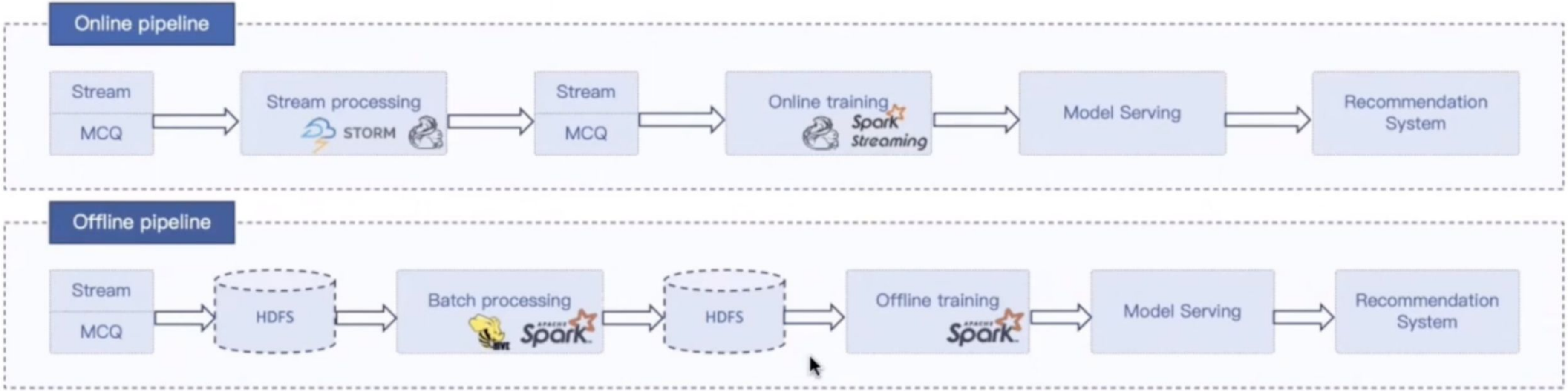
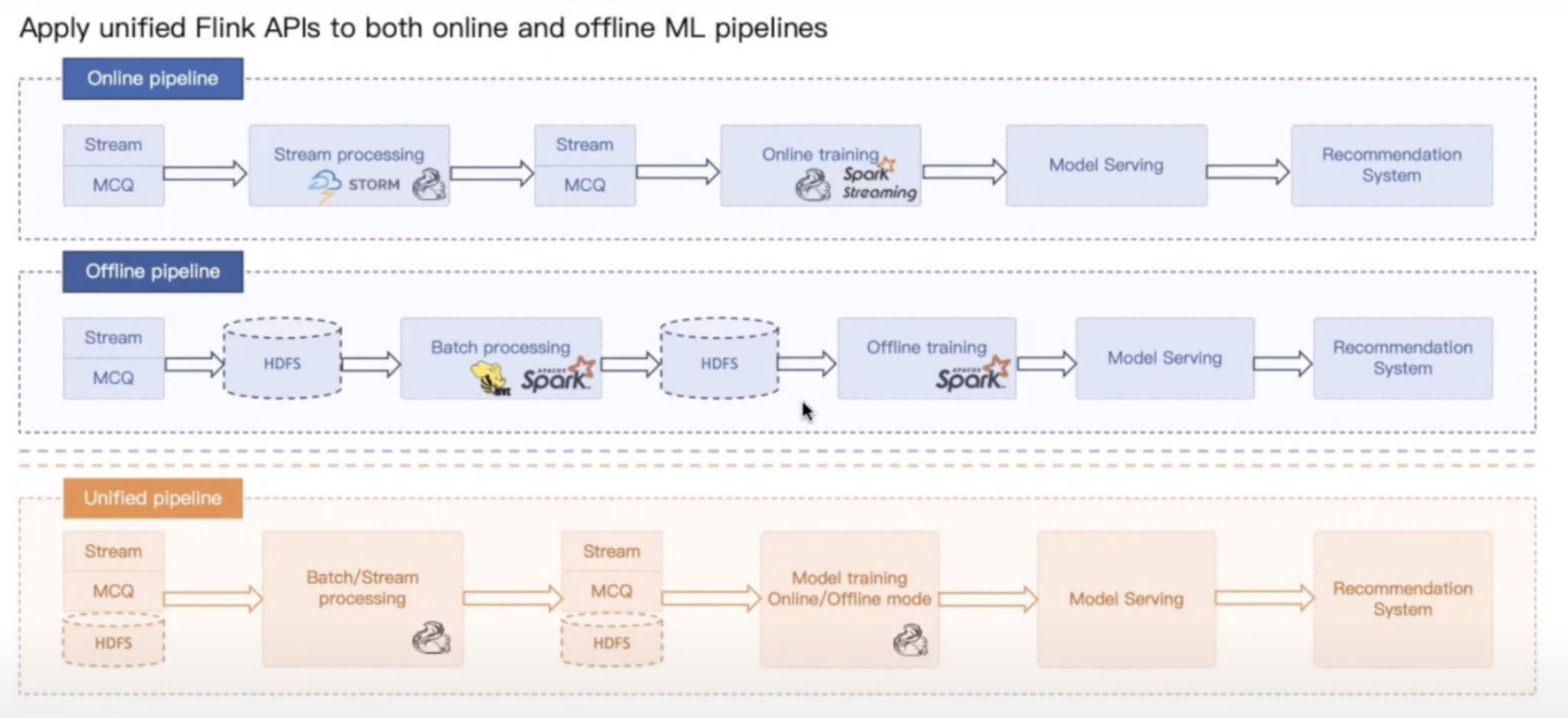


Image by Kostas Tzoumas (Ververica)

One model, two pipelines: example



Machine learning with Flink in Weibo (Qian Yu, QCon 2019)



Apply unified Flink APIs to both online and offline ML pipelines

Online pipeline

Stream / MCQ → Stream processing (STORM) → Stream / MCQ → Online training (Spark Streaming) → Model Serving → Recommendation System

Offline pipeline

Stream / MCQ → HDFS → Batch processing (Spark) → HDFS → Offline training (Spark) → Model Serving → Recommendation System

Unified pipeline

Stream / MCQ / HDFS → Batch/Stream processing → Stream / MCQ / HDFS → Model training Online/Offline mode → Model Serving → Recommendation System

Machine learning with Flink in Weibo (Qian Yu, QCon 2019)

Barriers to stream processing

1. Companies don’t see the benefits of streaming
   - Systems not at scale
   - Batch predictions work fine
   - Online predictions would work better but they don’t know that

Barriers to stream processing

1. Companies don’t see the benefits of streaming
2. High initial investment on infrastructure
3. Mental shift
4. Python incompatibility

2. Batch vs Online Prediction

Separation: causes of many MLOps problems

- Development environment vs. production environment
- Batch pipeline vs. streaming pipeline
- Development vs. monitoring

Batch prediction vs. online prediction

- Batch prediction
  - Generate predictions periodically, before requests arrive
  - Predictions are stored (e.g. SQL tables, CSV files) and retrieved when requests arrive
- Online prediction
  - Generate predictions after requests arrive
  - Predictions are returned as responses

⚠️ Misnomer ⚠️

- Both can do one or more samples (batch) at a time
- If you do compute on the cloud, then both are technically "online" - over the Internet

Batch prediction vs. online prediction

- Batch prediction
  - Generate predictions periodically before requests arrive
  - Predictions are stored (e.g. SQL tables) and retrieved when requests arrive
  - Asynch
- Online prediction
  - Generate predictions after requests arrive
  - Predictions are returned as responses
  - Sync when using requests like REST / RPC
    - HTTP prediction
  - Async [with low latency) with real-time transports like Kafka / Kinesis
    - Streaming prediction

Batch prediction vs. online prediction

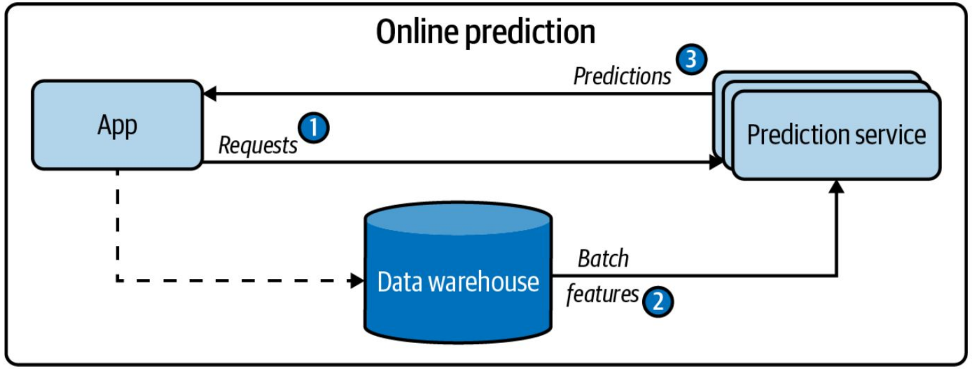
- Batch prediction
  - Generate predictions periodically before requests arrive
  - Predictions are stored (e.g. SQL tables) and retrieved when requests arrive
  - Asynch
- Online prediction
  - Generate predictions after requests arrive
  - Predictions are returned as responses
  - Sync when using requests like REST / RPC
    - HTTP prediction
  - Async [with low latency) with real-time transports like Kafka / Kinesis
    - Streaming prediction

Offered by major cloud providers

Still challenging

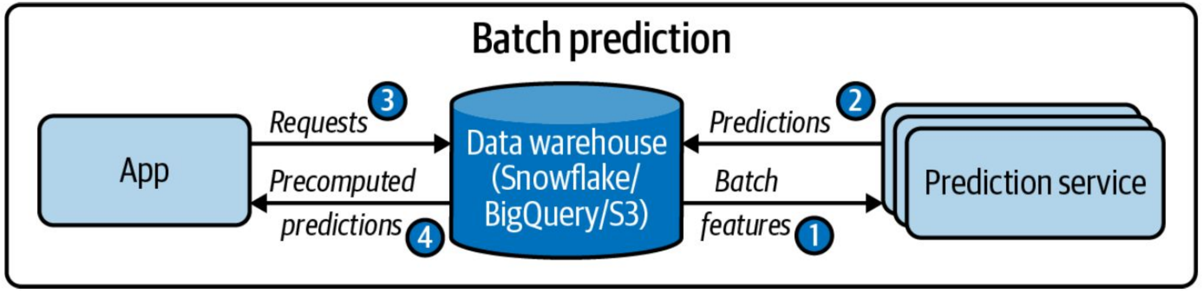
Online prediction

- Predictions are generated and returned as soon as requests for these predictions arrive (on-demand prediction).
- Traditionally requests are sent to the prediction service via RESTful APIs
- It is also known as synchronous prediction (predictions are generated in synchronization with requests) when prediction requests are sent via HTTP requests.

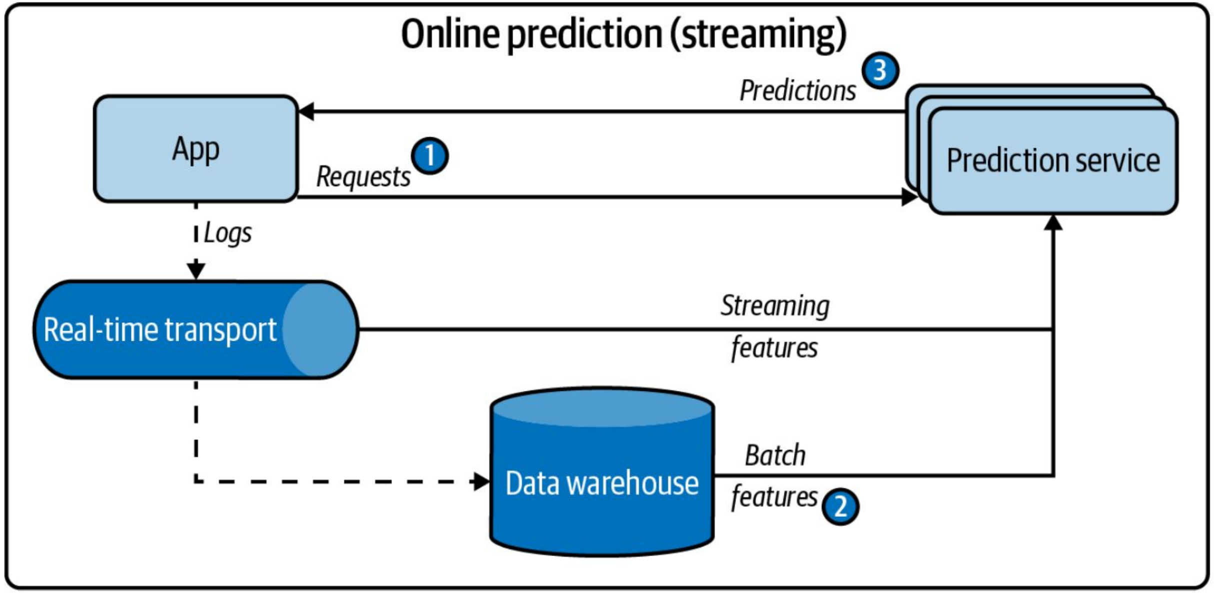


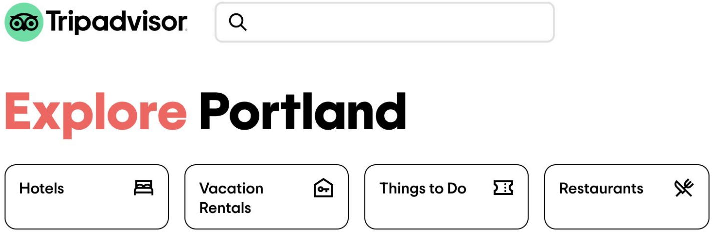
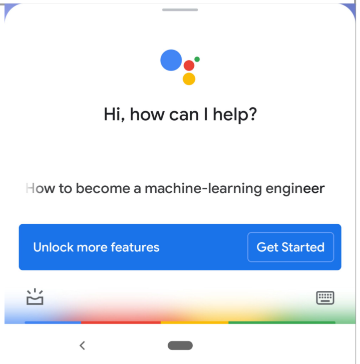
Batch prediction

- Predictions are generated periodically or whenever triggered.
- The predictions are stored somewhere, such as in SQL tables or an in-memory database, and retrieved as needed.
- Also known as asynchronous prediction: predictions are generated asynchronously with requests.



Streaming prediction



	Batch prediction (async)	Online prediction (generally sync)
Frequency	Periodical (e.g. every 4 hours)	As soon as requests come
Useful for	Processing accumulated data when you don't need immediate results (e.g. recommendation systems)	When predictions are needed as soon as data sample is generated (e.g. fraud detection)
Optimized	High throughput	Low latency
Input space	Finite: need to know how many predictions to generate	Can be infinite
Examples	● TripAdvisor hotel ranking ● Netflix recommendations  Tripadvisor Explore Portland Hotels Vacation Rentals Things to Do Restaurants	● Google Assistant speech recognition ● Twitter feed  Hi, how can I help? How to become a machine-learning engineer

Hybrid: batch & online prediction

- Online prediction is default, but common queries are precomputed and stored
- DOORDASH
  - Restaurant recommendations use batch predictions
  - Within each restaurant, item recommendations use online predictions
- NETFLIX
  - Title recommendations use batch predictions
  - Row orders use online predictions

3. Model Serving Considerations

Security and correctness

The runtime is responsible for authenticating the user identity, and authorizing their requests. Things to check are:

- whether a specific user has authorized access to the models they want to run
- whether the names and the values of parameters passed correspond to the model’s specification
- whether those parameters and their values are currently available to the user

Ease of deployment

- The model should be updated with minimal effort and without affecting the entire application.
- Different deployment methods require different update strategies:
    - Web service: replace model file and restart service
    - Virtual machine or container: replace instances with new image
    - Model streaming: stream new version of model and related components
- Model streaming requires stateful application that changes state when new version is received.

Guarantees of model validity

- A runtime should ensure the model, the feature extractor, and other components are valid and in sync.
- A model should be deployed with four assets: an end-to-end set, a confidence test set, a performance metric, and its acceptable range.
- A model should not be served (or stopped if running) if:
  - any end-to-end test example is scored incorrectly, or
  - the performance metric on the confidence test set is out of range.

Ease of recovery

- A runtime should enable easy recovery from errors by rolling back to previous versions.
- The recovery process should be similar and simple as the deployment process, except using the previous working version instead of the new model.

Avoidance of training/serving skew

- Training/Serving Skew: when features used for training and production are different
- Causes: using two different codebases for feature extraction (e.g. for efficiency or compatibility reasons)
- Consequences: suboptimal or incorrect model performance
- Solutions:
  - Use the same feature extraction code for both training and production
  - Wrap the feature extraction object into a separate web service
  - Log feature values generated in production and use them for training

Avoidance of hidden feedback loops

- Hidden Feedback Loops: when model output influences model input
- Causes: using model output as feature for another model or for the same model
- Consequences: skewed data and biased learning
- Solutions:
  - Avoid circular dependencies between models
  - Use held-out examples that are not affected by the model output
  - Show all held-out examples to the user and use their feedback for training

Being ready for errors

Errors are inevitable in any software. In machine-learning-based software, errors are an integral part of the solution: no model is perfect. Because we cannot fix all errors, the only option is to embrace them.

Embracing errors means designing the software system in such a way that when an error happens, the system continues operating normally.

Being ready for errors

We must accept and embrace three “cannots”:

- We cannot always explain why an error happened
- We cannot reliably predict when it will happen
- We cannot always know how to fix a specific error

Dealing with errors

To deal with errors, you should:

- Have a strategy that minimizes the impact of the system mistakes on users.
- Limit the user's exposure to the model and monitor the user's engagement with the system.
- Be very careful in critical scenarios where the system acts on the user's behalf.

Dealing with errors

Some techniques that you can use are:

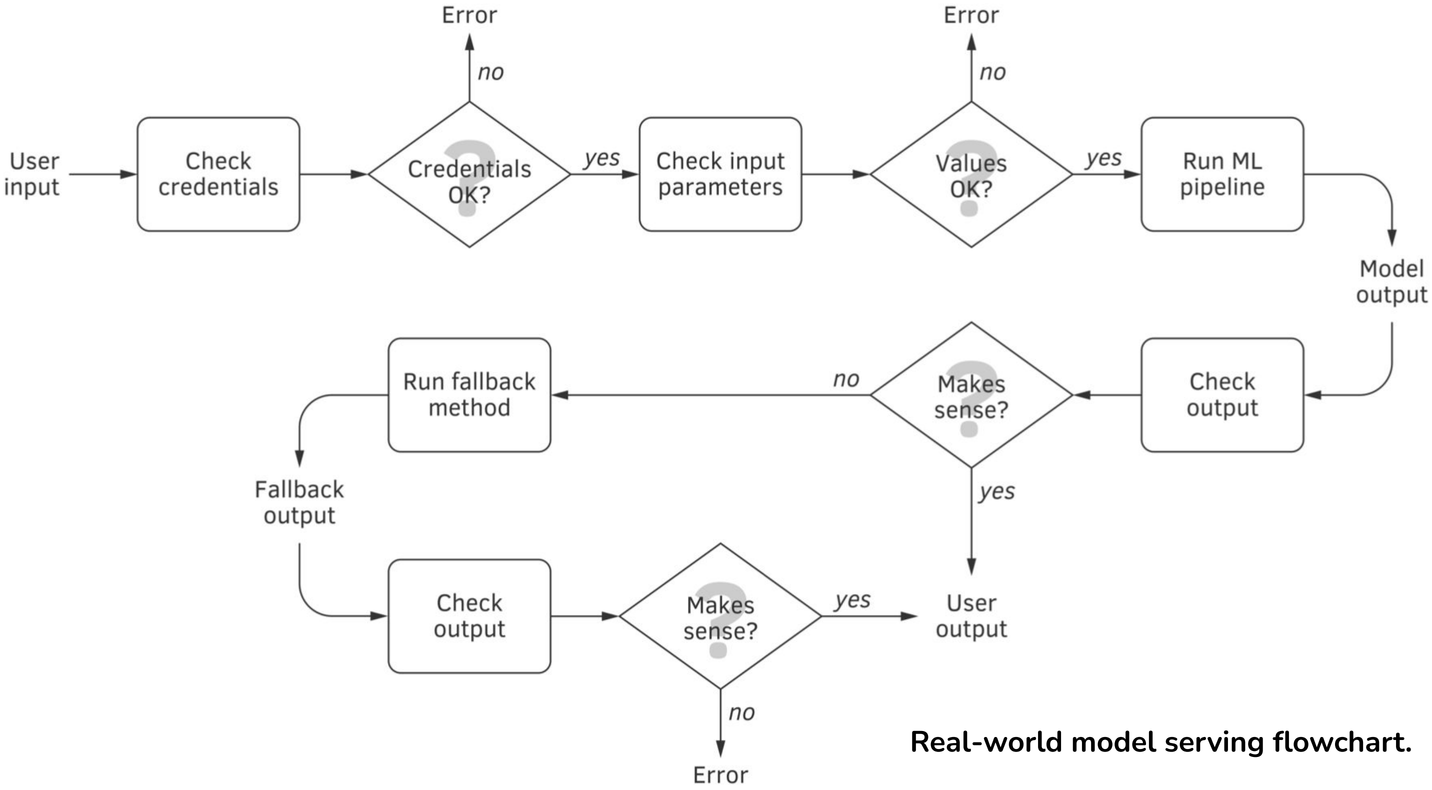
- Saying "I don't know" instead of giving a random answer.
- Calculating the cost of the error and hiding it if it is too high.
- Training a second model to detect when the first is likely to make an error.
- Optimizing the model for the type of error that you want to avoid.
- Presenting multiple options when the confidence is low.
- Measuring the number of errors and the user's tolerance level.

Dealing with errors

Some techniques that you can use are:

- Allowing the user to report errors and explaining how they will be fixed.
- Logging and analyzing suspicious interactions offline.
- Giving the user an option to undo an action recommended by the system.
- Sending an alert and putting an action on hold if the model predicts something unreasonable.
- Implementing a fallback strategy using a simpler model or a heuristic if the model prediction is rejected.
- Validating the output of the fallback strategy and sending an error message if it is also rejected.

Dealing with errors



Real-world model serving flowchart.

Being ready for and dealing with change

- Machine learning systems can change over time due to various factors, such as concept drift or data imbalance.
- These changes can affect the system's quality and predictions, and the user's perception and satisfaction.
- Users may have different preferences and interests, and may get used to certain results or behaviors.
- To avoid user frustration or confusion, you should:
  - Educate the user about the changes and what to expect from the new model.
  - Give the user time to adapt to the new model.
  - Use a gradual or parallel approach to transition from the old model to the new model.

Being ready for and dealing with human nature

- **Avoid Confusion**: The system should be easy to use and understand for the user, without expecting them to have any knowledge of machine learning and AI, and without showing them unexpected errors.
- **Manage Expectations**: Some users may have unrealistic expectations of machine learning systems because of advertisements that show them as “intelligent” or because of their previous experience with similar systems that seemed “very intelligent” to them.
- **Gain Trust**: Some users may distrust machine learning systems because of their past experience with them and may try to test your system’s abilities with simple queries or commands. You should make sure that your system can handle such tests and gain the user’s confidence quickly.

Being ready for and dealing with human nature

- **Manage User Fatigue**: User fatigue can occur when the system interrupts the user too often or makes inappropriate decisions on their behalf. You should balance the level of automation and user control and use a model to detect sensitive information that should not be shared without user consent.
- **Beware of the Creep Factor**: Creep factor is when the user feels uneasy about the system's ability to predict their personal details. You should avoid making the system seem too intrusive or authoritative.

Machine Learning Systems Design

Deployment and Monitoring

Next Lecture: Model Serving

CE 40959 Spring 2023
Ali Zarezade
SharifMLSD.github.io