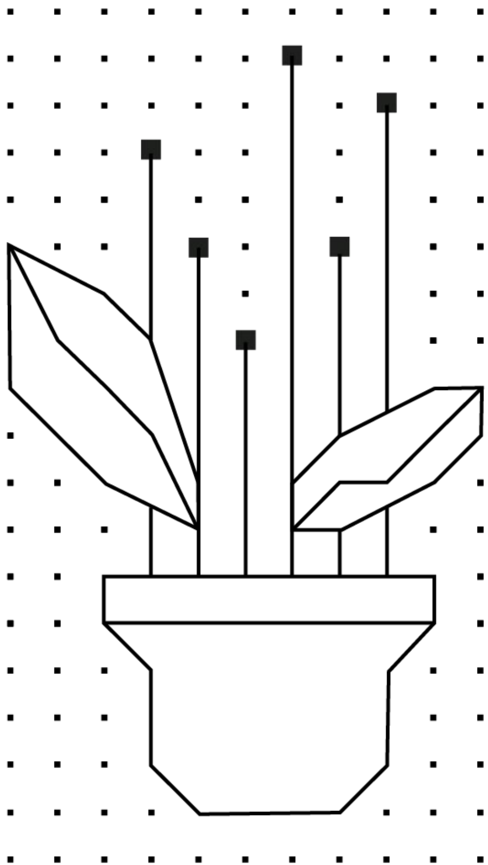
Invisible Interfaces

Considerations for Abstracting Complexities of a Real-time ML Platform

Zhenzhong Xu
Cofounder & CTO @ claypot.ai
July, 2023



The discovery of something invisible

The Invisible Interface

- Ubiquitous
- Easy and responsive
- Just works!

The endeavor to make things useful

Real-time Decisions

that powers your business

Fraud prevention Personalization Customer support Dynamic pricing/discounting
Trending products Risk Assessment Account Take Over
Ads
ETA Network analysis Sentiment analysis Object detection
…

The world is moving towards real-time

- Instacart: The Journey to Real-Time Machine Learning (2022)
    - Directly reduces **millions of fraud-related costs** annually.
- LinkedIn's Real-time Anti-abuse (2022)
    - LinkedIn moved from an offline pipeline (hours) to real-time pipeline (minutes), and saw **30% increase in bad actors caught online** and **21% improvement in fake account detection.**
- How WhatsApp catches and fights abuse (2022 | slides)
    - **A few 100ms delay can increase the spam by 20-30%**.
- How Pinterest Leverages Realtime User Actions in Recommendation to Boost Engagement (2022)
    - According to Pinterest, this "*has been one of our most impactful innovations recently,* ***increasing Home feed engagement by 11% while reducing Pinner hide volume by 10%.***"
- Airbnb: Real-time Personalization using Embeddings for Search Ranking (2018)
    - Moving from offline scoring to online scoring grows bookings by **+5.1%**

Real-time Decisions

Exploration & Research

Model Architecture & Turning

Model Analysis & Selection

LLM Prompt Engineering

Data Fabric for Real-time AI

Data Infrastructure

Data Sources

Ingestion & Transport

Storage

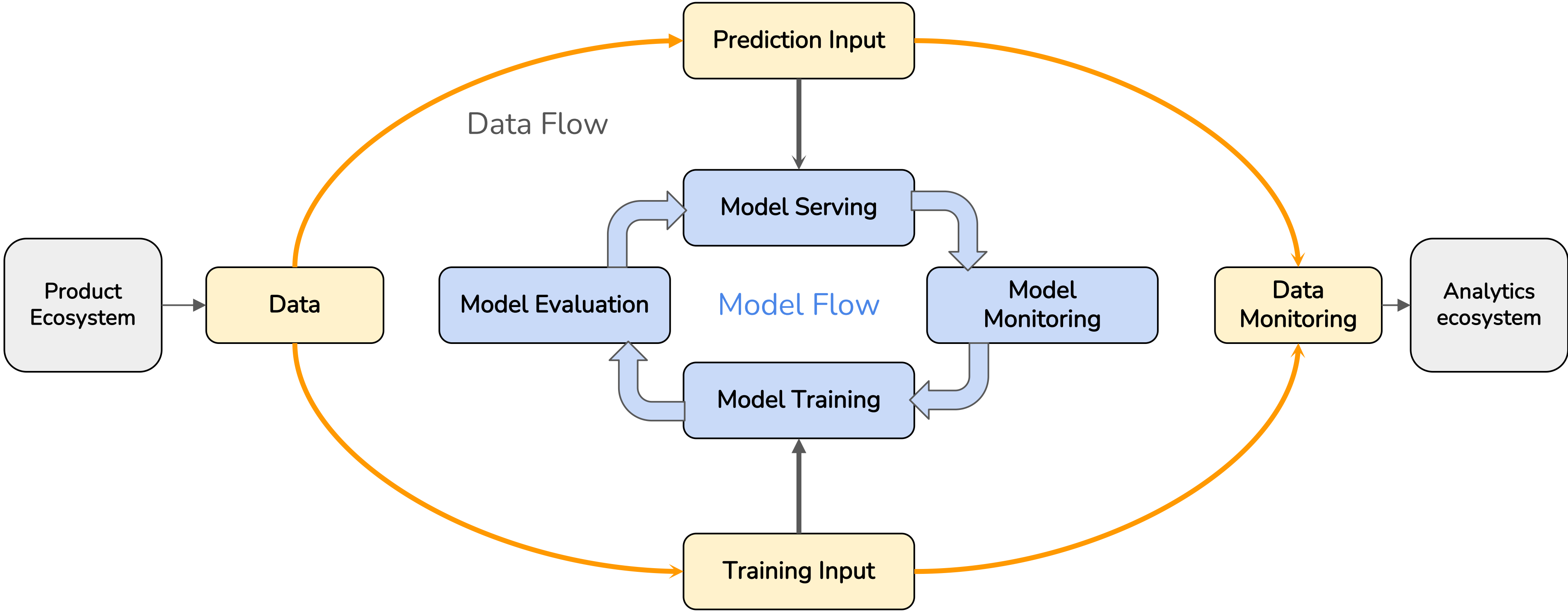
Query & Compute

Workflow Orchestration

Analytics / Visualization

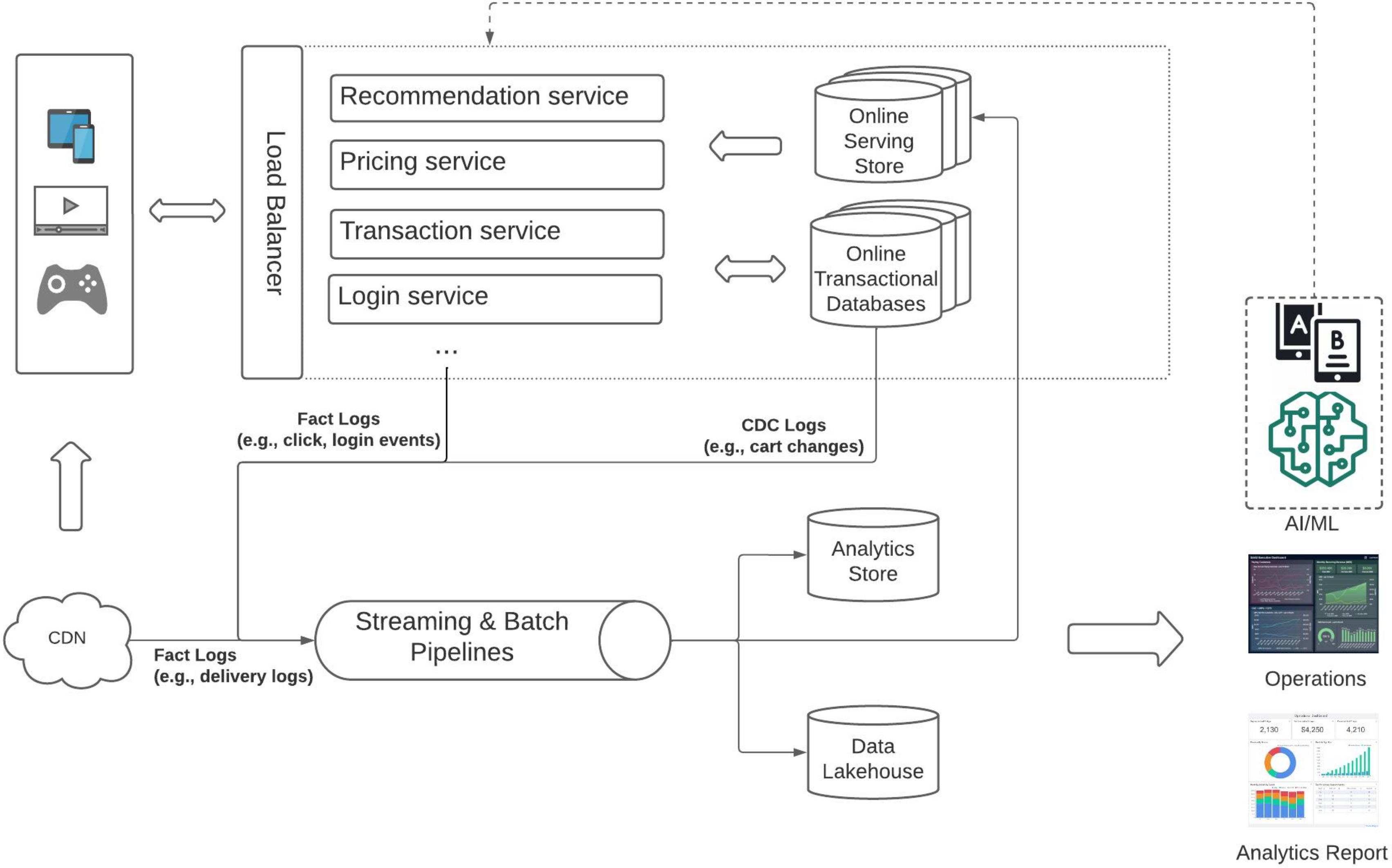
Multi-tenancy Isolation

Security & Governance


Prediction Input
Data Flow
Model Serving
Product Ecosystem
Data
Model Evaluation
Model Flow
Model Monitoring
Data Monitoring
Analytics ecosystem
Model Training
Training Input

The hard things towards real-time decisions

- Data silo and staleness
- Collaboration overhead
- Tech complexity



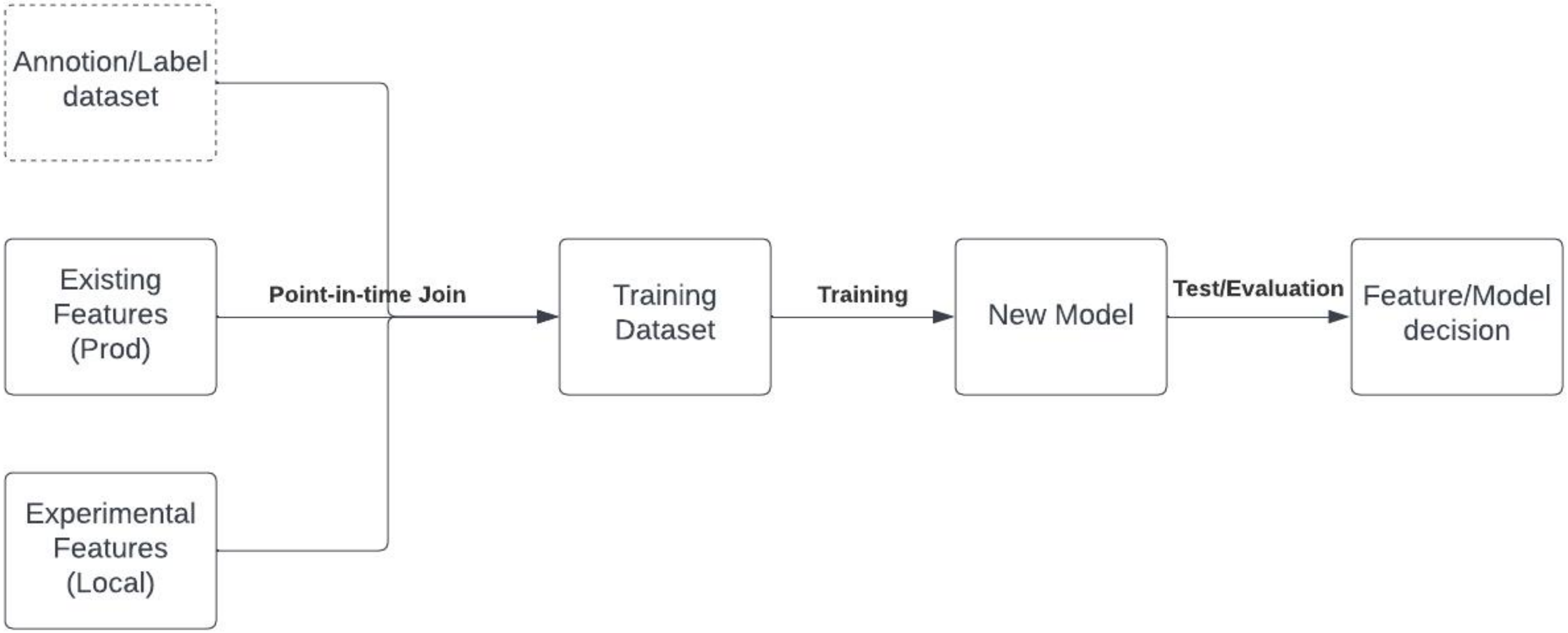
Load Balancer
Recommendation service
Pricing service
Transaction service
Login service
...
Online Serving Store
Online Transactional Databases
Fact Logs (e.g., click, login events)
CDC Logs (e.g., cart changes)
CDN
Fact Logs (e.g., delivery logs)
Streaming & Batch Pipelines
Analytics Store
Data Lakehouse
A
B
AI/ML
Operations
2,130
$4,250
4,210
Analytics Report

Claypot

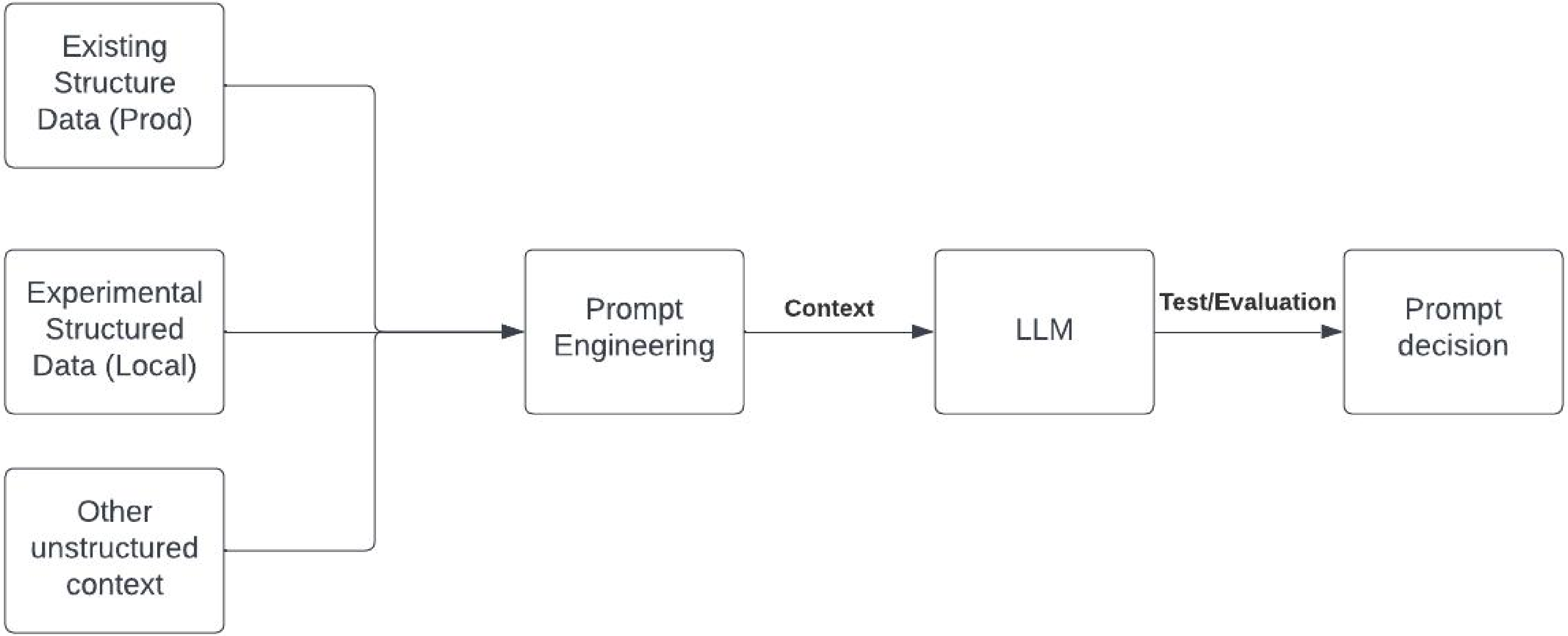
Challenge 1: From Experimentation to Production

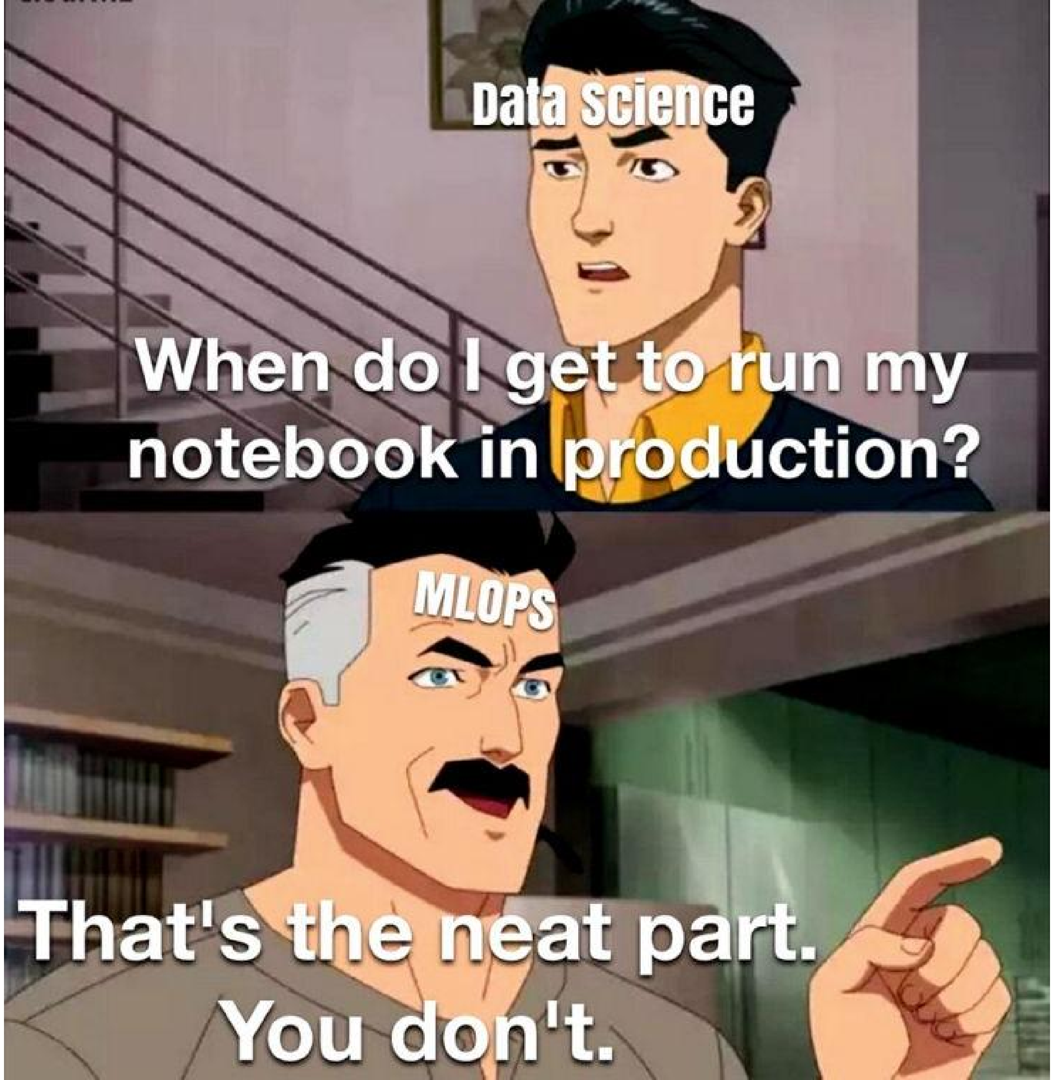
- Slow prototyping
- Local vs. remote execution
- Divergent language & runtime

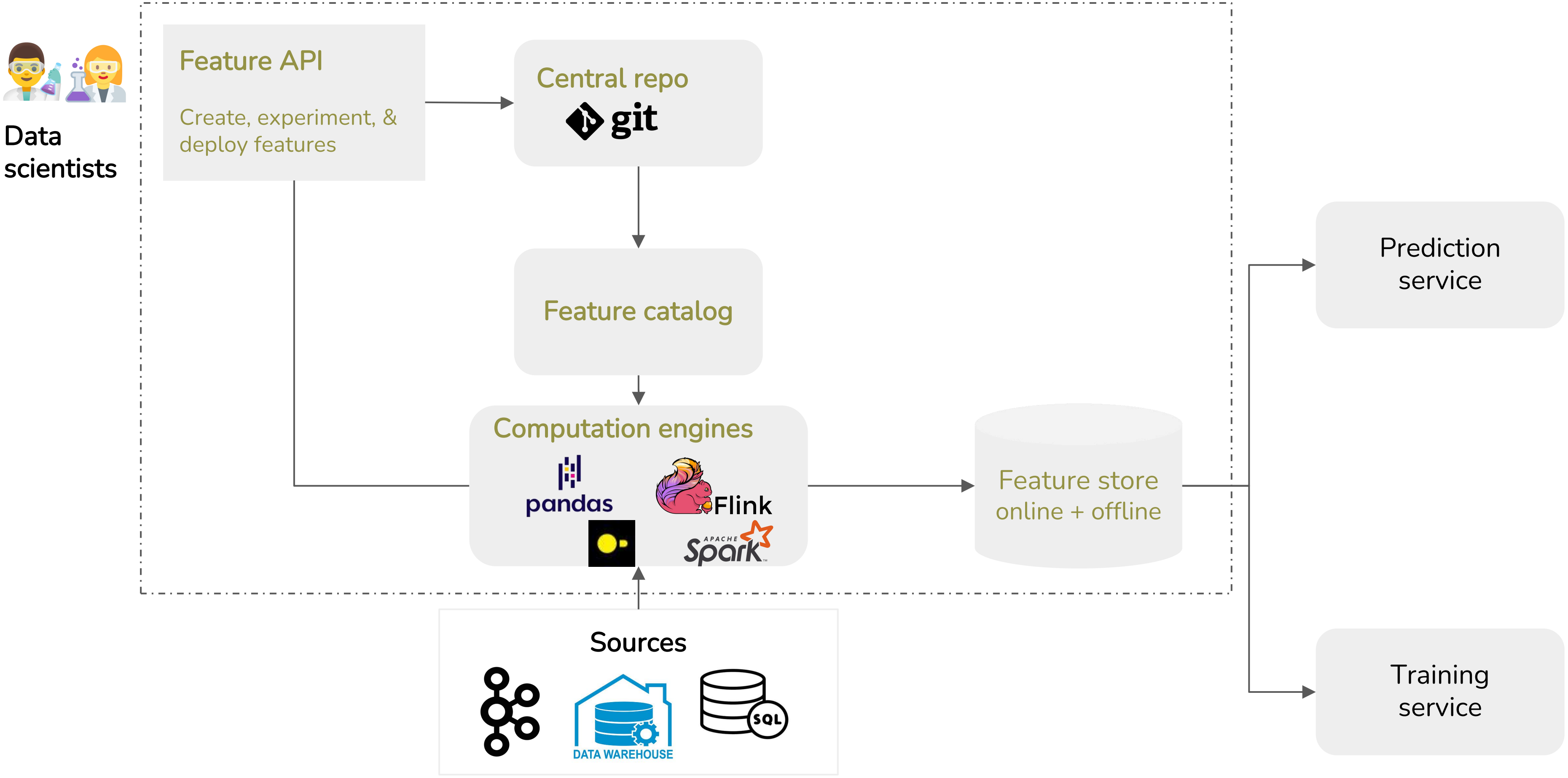
Local Experimentation with Traditional Models



Local Experimentation with LLMs




Data science
When do I get to run my notebook in production?
MLOPS
That's the neat part. You don't.


Data scientists
Feature API
Create, experiment, & deploy features
Central repo
git
Feature catalog
Computation engines
pandas
Flink
APACHE Spark
Feature store online + offline
Prediction service
Training service
Sources
DATA WAREHOUSE
SQL

Need an invisible interface to plug into compute ecosystems

pandas

DuckDB

dask

RAPIDS

jupyter

APACHE ARROW

Flink

RisingWave

RAY

APACHE Spark

Materialize

trino

Local/Single Machine

Remote/Distributed

Declare features with familiar APIs

```
@transformation
def average_transaction_amount_by_merchant(
    tx: Transactions,
    wspec: WindowSpec):

return tx.groupby(["cc_num", "merchant"])["amt"].window(wspec).mean()
```

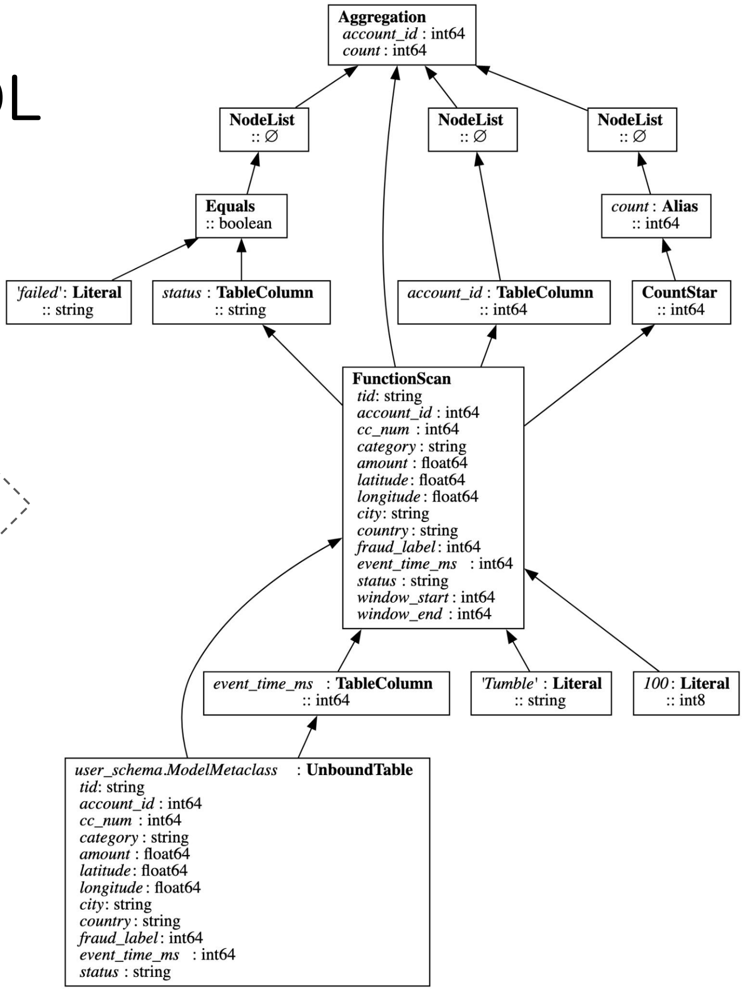
Data Science Friendly: Python <> SQL

```python
@transformation
def transaction_count(tx: Transactions, wspec: WindowSpec):
  return tx[tx.status == "failed"].groupby("account_id").window(wspec).count()
```

Relational Expression

Workload Compiler / Optimizer

Deployment



Same code can run on different computation engines

```python
@transformation
def transaction_count(tx: Transactions, wspec: WindowSpec):
  return tx[tx.status == "failed"].groupby("account_id").window(wspec).count()
```

Intermediate Representation

Relational Expression

Compile into a relational expression (RE), which is SQL equivalent

Workload Compiler/Optimizer

Compile & optimize RE into the computation engine
(e.g., Panda, DuckDb, Flink, Spark) best suited for the job

Deployment

Spin up and manage computation jobs

Claypot

Solution 1: Relational Expression based Compilation

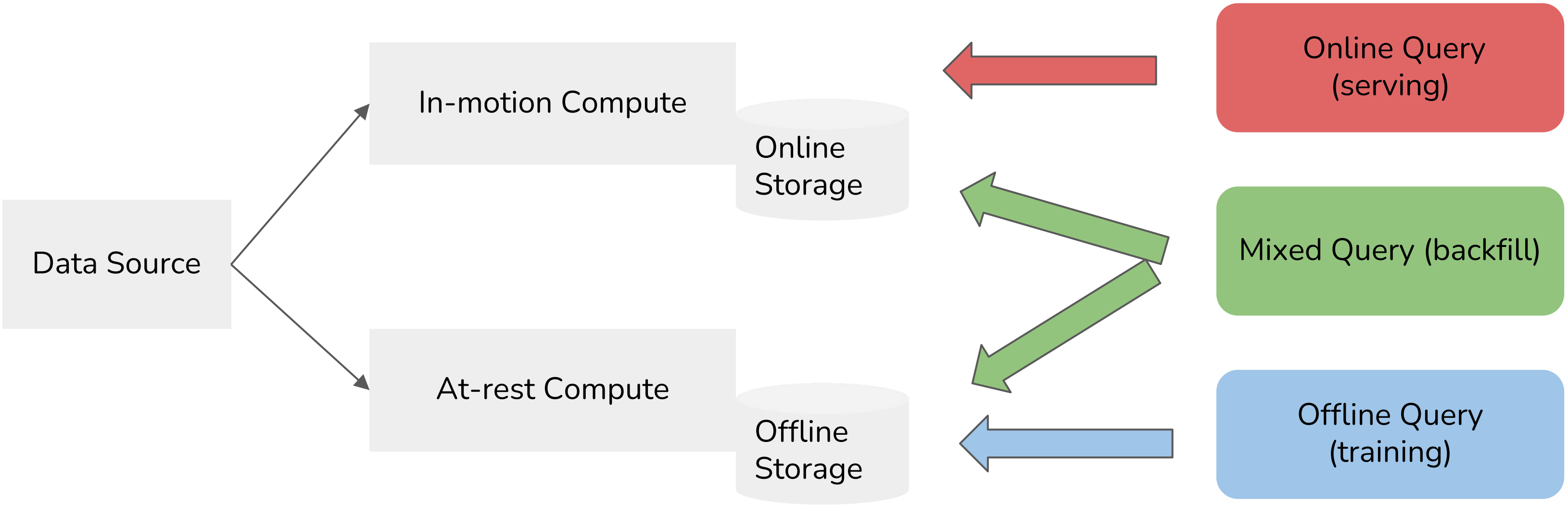
- Unified yet familiar API
- Pluggable to many compute engines
- Minimize human error
- Prototype in minutes

Claypot

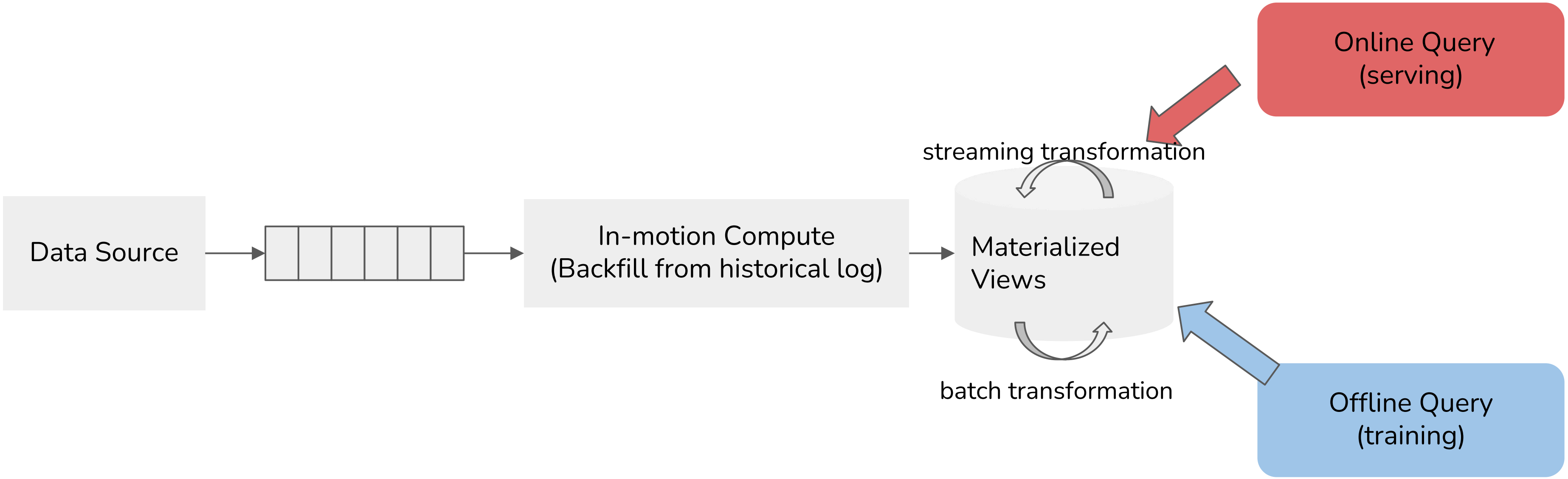
Challenge 2: Streaming and Batch Divided

- Evolving architecture
- Difficult to backfill
- Train-predict inconsistencies

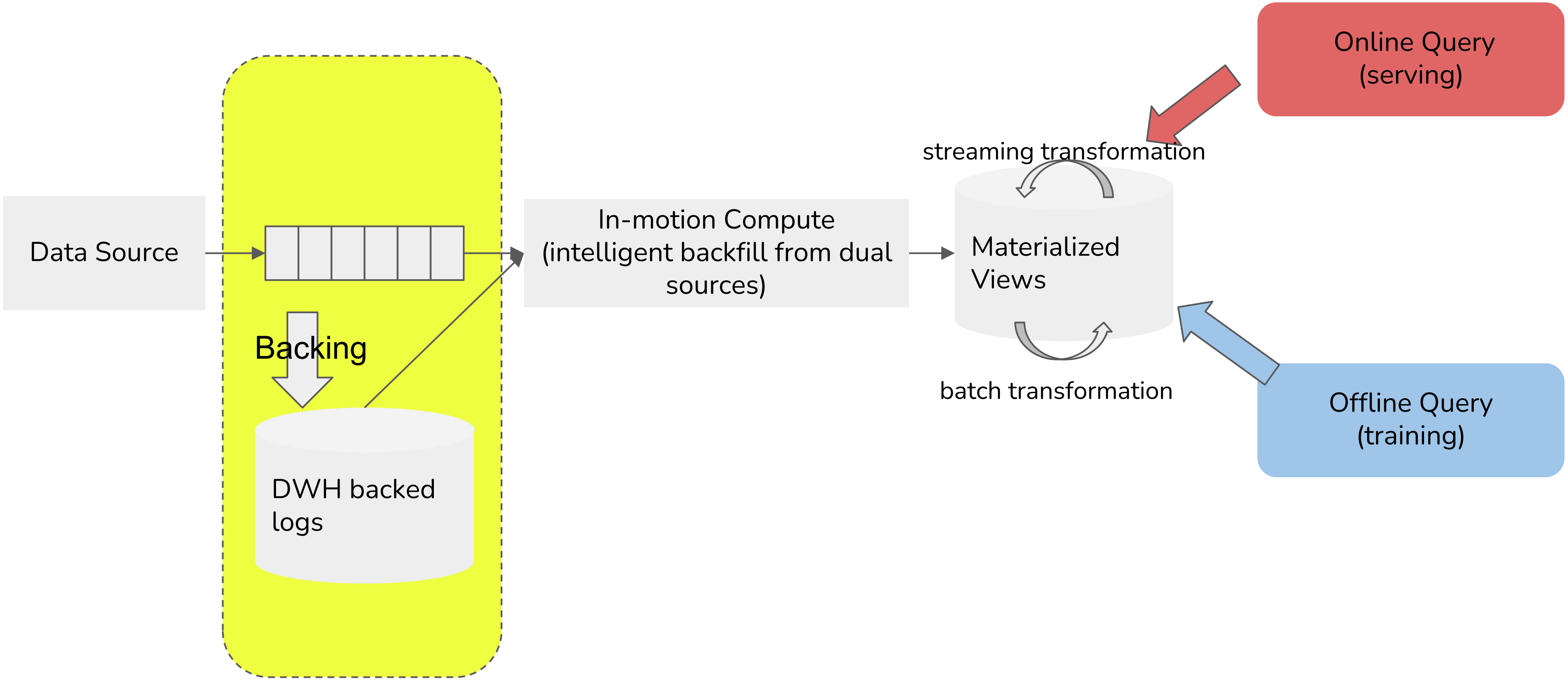
Lambda Architecture



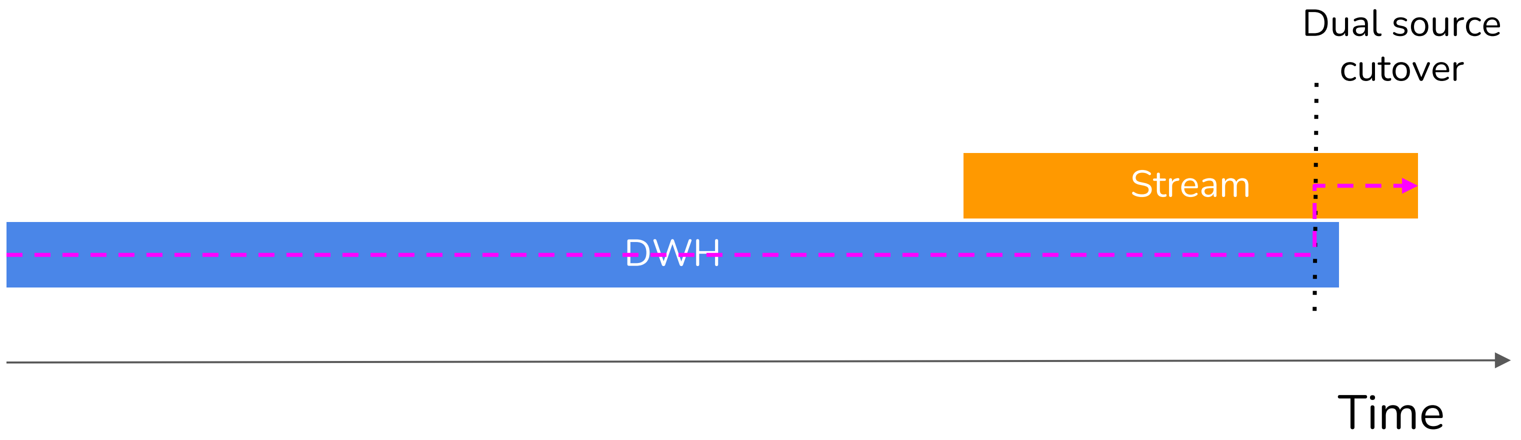
Kappa (Streaming) Architecture



Unified Architecture



Batch and streaming source unified to simplify backfill

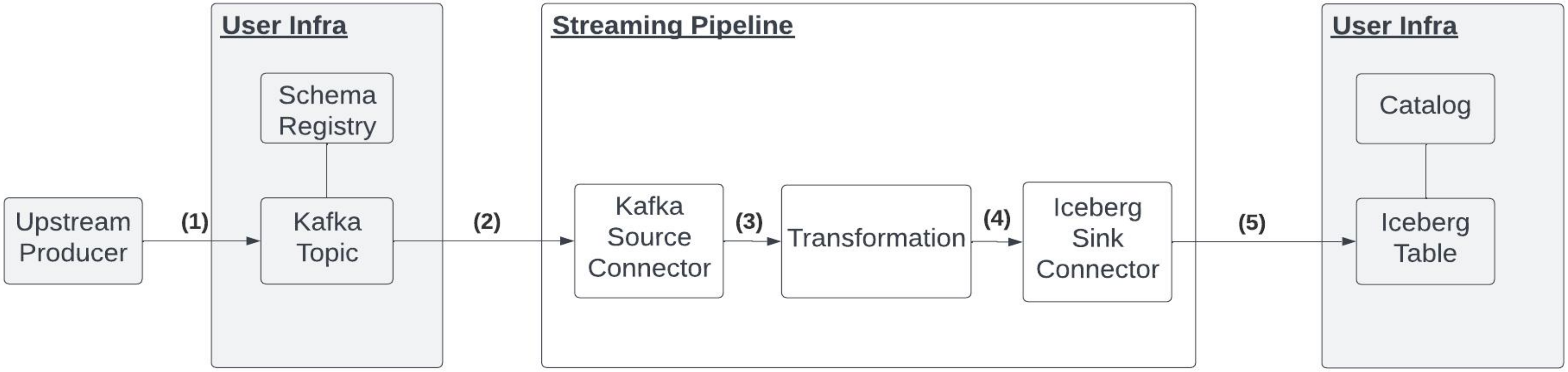


Need an invisible interface to plug into storage ecosystems

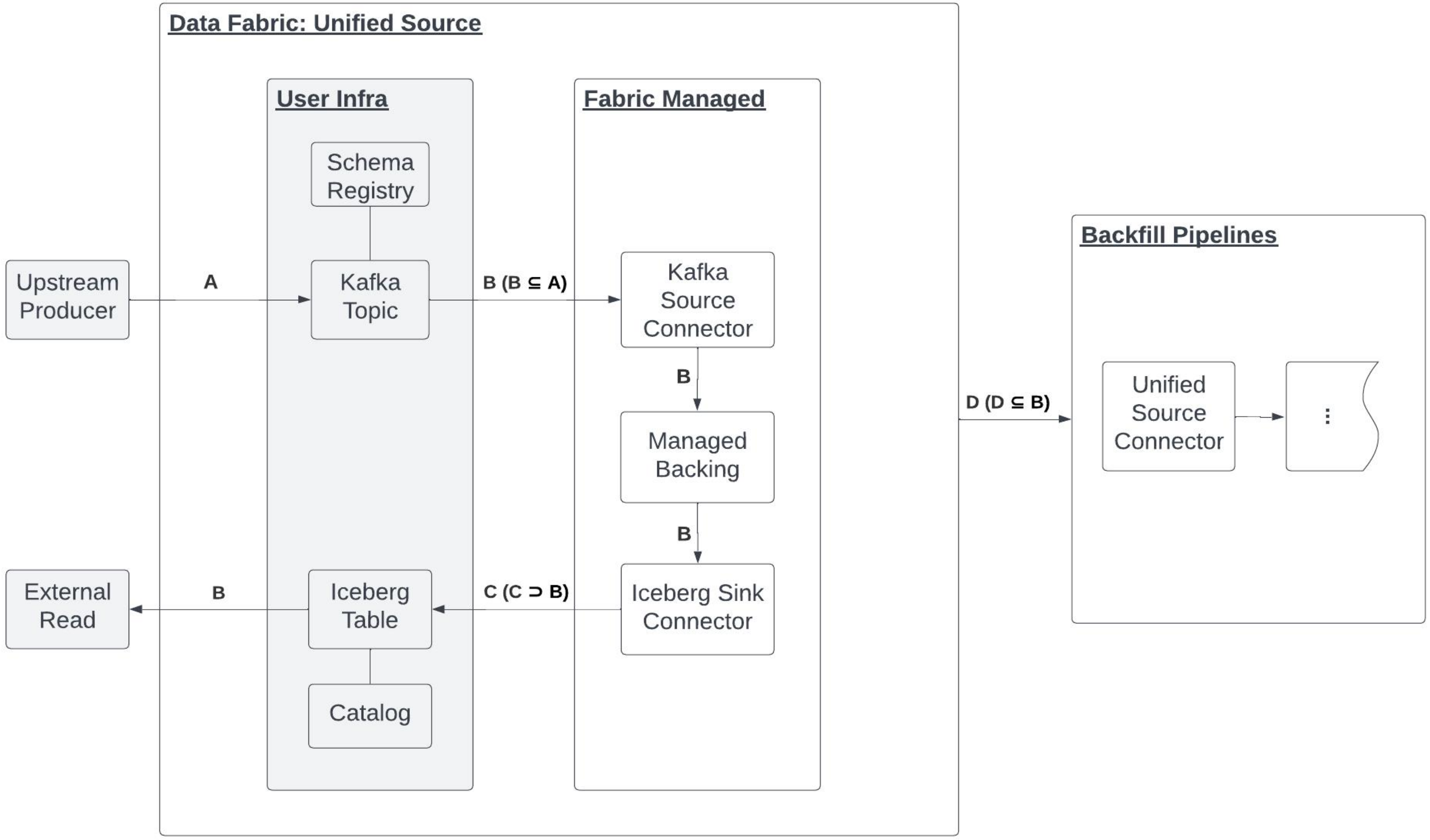
Streaming Leaning

Batch Leaning

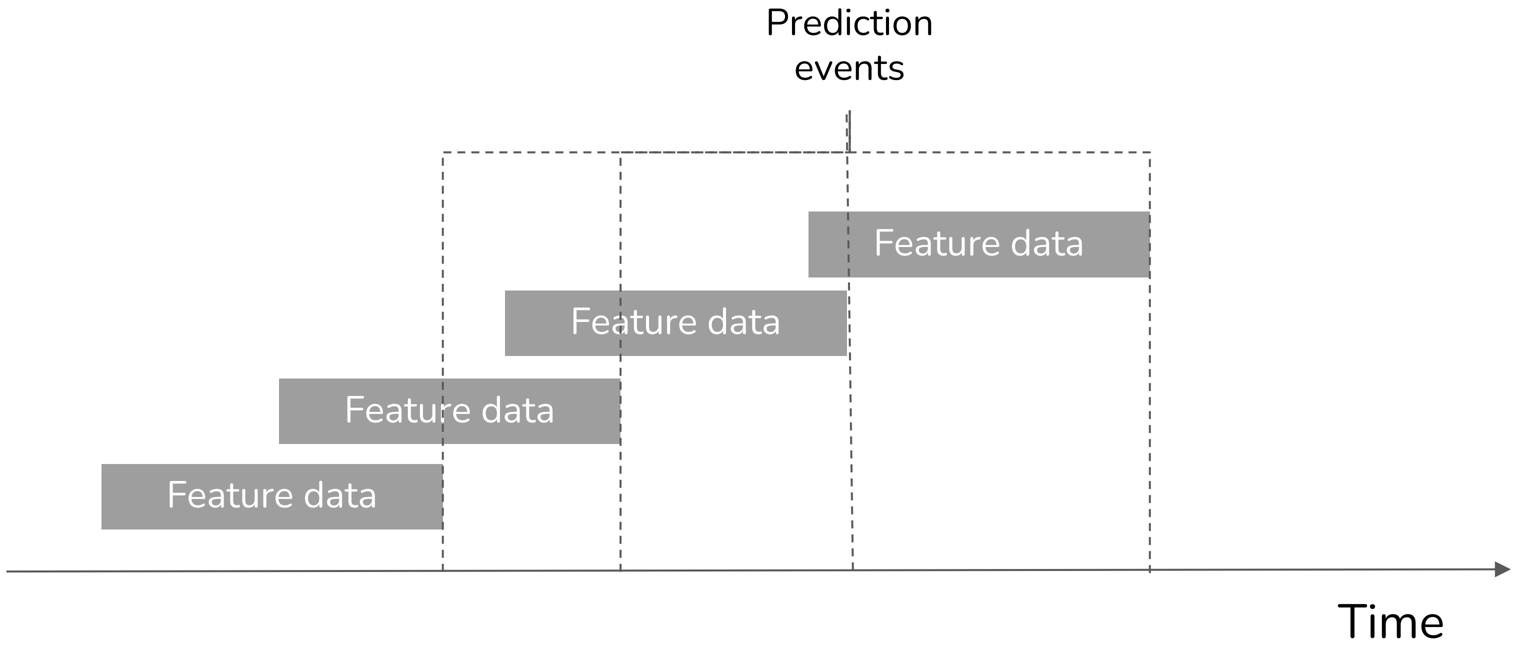
Data Fabric for a Streaming Pipeline



Data Fabric for a Unified Backfill Pipeline



Training dataset backfill requires point-in-time correctness



Point-in-time joins to generate training data

Given a spine (entity keys + timestamp + label), join features to generate training data

spine_df

inference_ts	tid	cc_num	user_id	is_fraud
21:30	0122	2	1	0
21:40	0298	4	1	0
21:55	7539	6	3	1

cc_num_tx_max_1h

ts	cc_num	tx_max_1h
9:20	2	…
10:24	2	…
20:00	4	…

user_unique_id_30d

ts	user_id	unique_ip_30d
6:00	1	…
6:00	3	…
6:00	5	…

```
train_df = pitc_join_features(
    spine_df,
    features=[
      "tx_max_1h",
      "user_unique_ip_30d",
    ],
)
```

inference_ts	tid	cc_num	user_id	is_fraud	tx_max_1h	user_unique_ip_30d
21:30	0122	2	1	1	…	…
21:40	0298	4	1	1	…	…
21:55	7539	6	3	3	…	…

Claypot

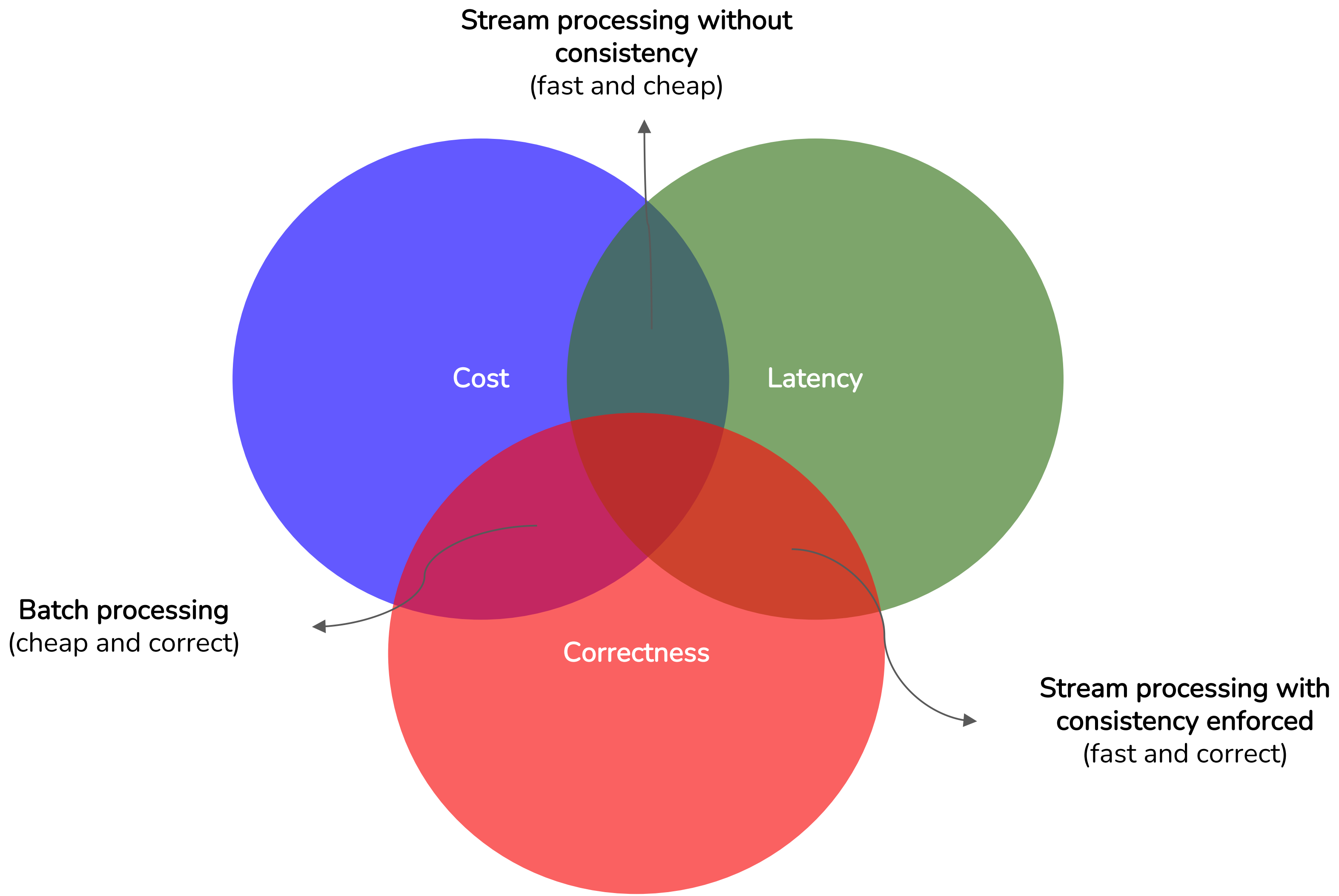
Solution 2: Abstract streaming and batch data storage

- Unified streaming & batch source
- Unified online & offline feature stores
- Pluggable to most storage technologies

Claypot

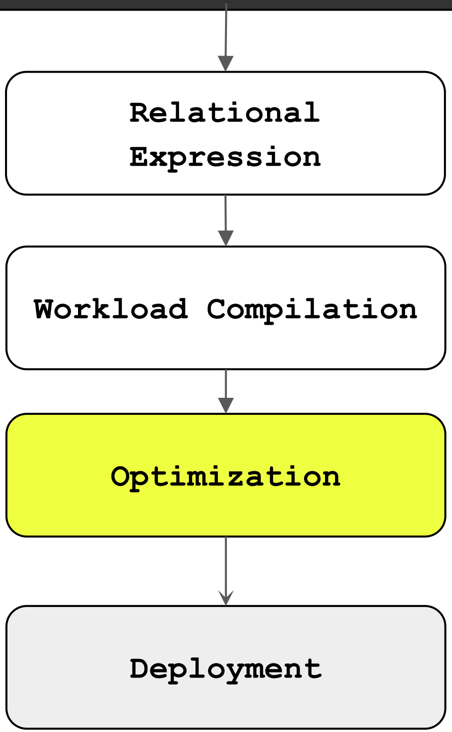
Challenge 3: It should just work!

- Cost, latency, correctness surprises!
- Lack optimizations knobs

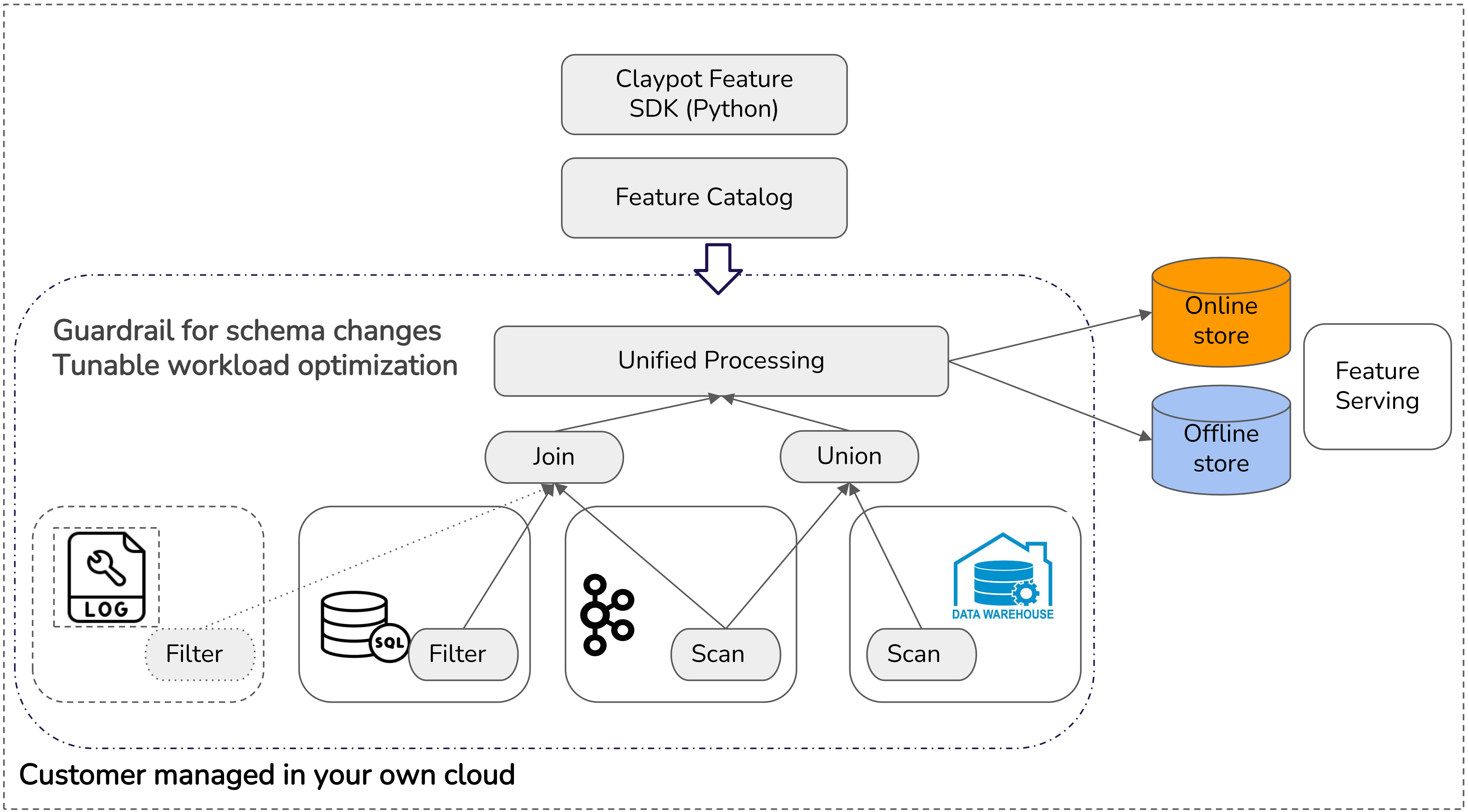

Stream processing without consistency
(fast and cheap)
Cost
Latency
Correctness
Batch processing
(cheap and correct)
Stream processing with consistency enforced
(fast and correct)

Optimization

```python
@transformation
def transaction_count(tx: Transactions, wspec: WindowSpec):
  return tx[tx.status == "failed"].groupby("account_id").window(wspec).count()
```



Various intelligent optimization can be done to make appropriate tradeoff across storage and compute systems.



Claypot Feature SDK (Python)
Feature Catalog
Guardrail for schema changes
Tunable workload optimization
Unified Processing
Online store
Offline store
Feature Serving
Join
Union
LOG
Filter
SQL
Filter
Scan
DATA WAREHOUSE
Scan
Customer managed in your own cloud

Claypot

Solution 3: Optimization knobs

- Abstract optimization complexity
- User controls with high level knobs
- Trust, no surprises!

Claypot

Make invisible interface possible!

- Ubiquitous
- Easy and responsive
- Just works!

https://zhenzhongxu.com/
zhenzhong@claypot.ai

the invisible interface