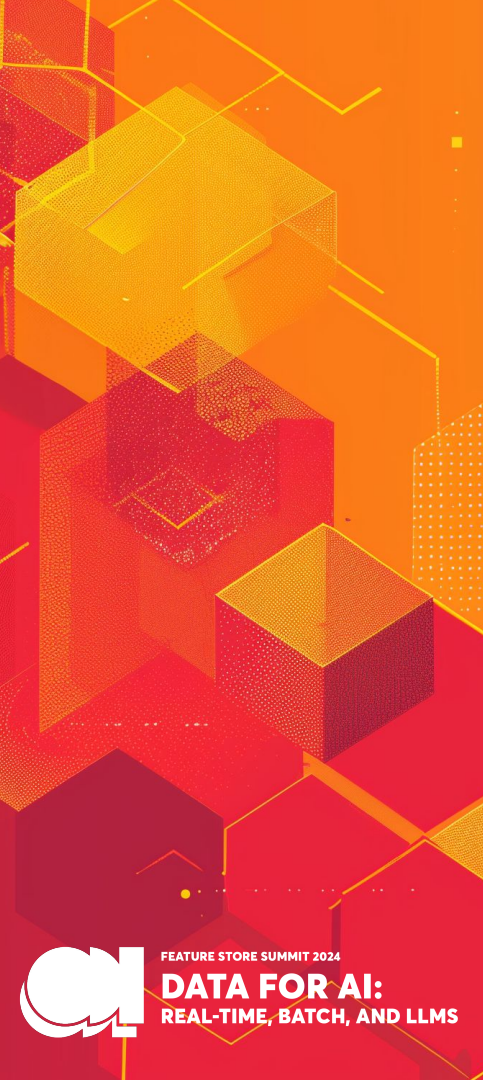


Shepherd: High-Scale, Low-Latency Machine Learning with Flink at Stripe

Caio Camatta, Software Engineer, Stripe
Divya Manohar, Software Engineer, Stripe

Organized by HOPSWORKS

Machine Learning at Stripe

ML algorithms are deployed across Stripe's product line, optimizing everything from backend processing to user interfaces.

One of the most popular use cases is fraud prevention.

- Blocking fraudulent transactions across all payment methods
- Stopping fraudsters from testing cards
- Detecting merchant fraud

Shepherd: Stripe's feature engineering platform

- In 2022, partnered with Airbnb to adopt Chronon.
- Designed Shepherd to accelerate feature development, which is crucial in adversarial spaces like fraud detection.
- Allows quickly feature iteration, backfill of historical data, automated deploys, and performance monitoring.
- Supports both batch and streaming features with strict latency and freshness guarantees.

Chronon: Defining a feature

Example Source definition

Amount of money charged on a credit card over the last day.

```
card_transactions_source = Source(
    table="my_table",
    topic="my_kafka_topic",
    events=ttypes.EventSource(
        query=Query(
            selects=select(
                card="card_id",
                amount_in_dollars="CAST(amount_cents/100.0 AS DOUBLE)",
            ),
            time_column="created",
        )
    )
)
```

Example Source definition

Amount of money charged on a credit card over the last day.

```
card_transactions_source = Source(
    table="my_table",
    topic="my_kafka_topic",
    events=ttypes.EventSource(
        query=Query(
            selects=select(
                card="card_id",
                amount_in_dollars="CAST(amount_cents/100.0 AS DOUBLE)",
            ),
            time_column="created",
        )
    )
)
```

Example Source definition

Amount of money charged on a credit card over the last day.

```
card_transactions_source = Source(
    table="my_table",
    topic="my_kafka_topic",
    events=ttypes.EventSource(
        query=Query(
            selects=select(
                card="card_id",
                amount_in_dollars="CAST(amount_cents/100.0 AS DOUBLE)",
            ),
            time_column="created",
        )
    )
)
```

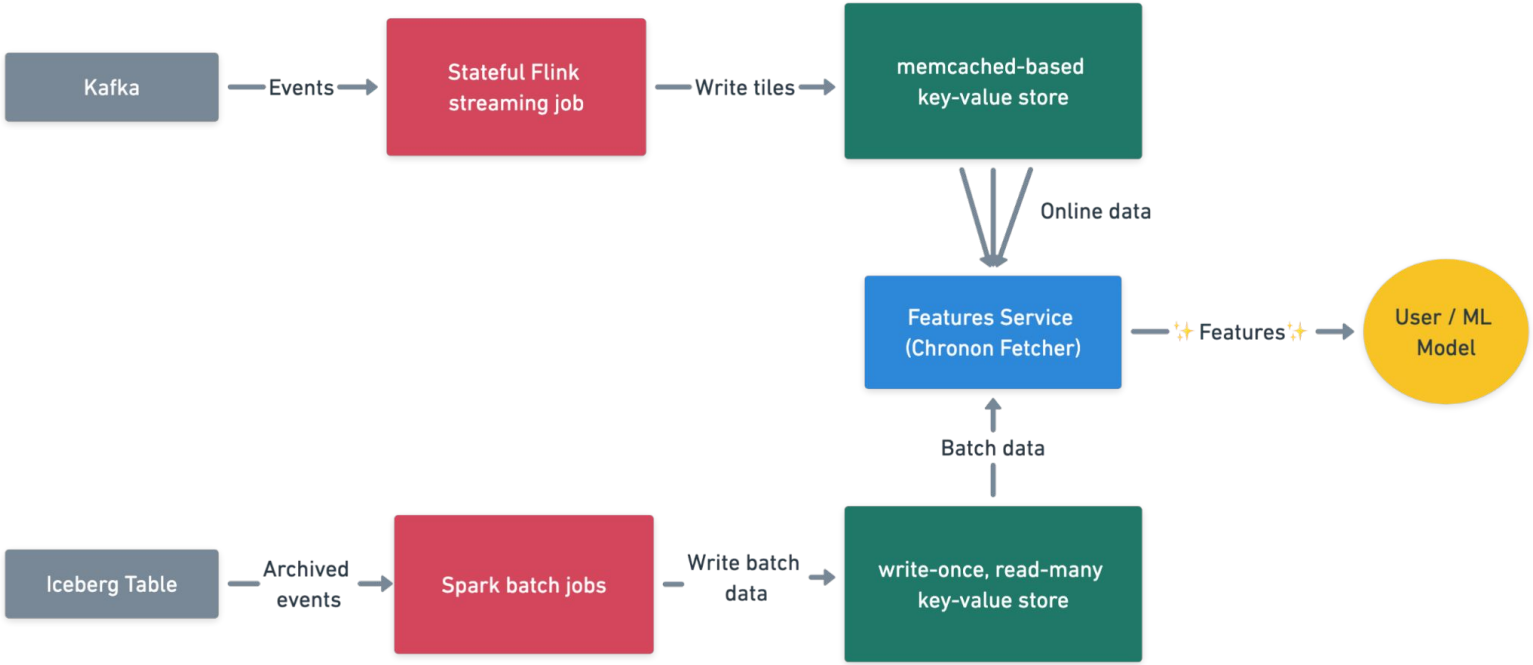
Spark SQL

Example GroupBy definition

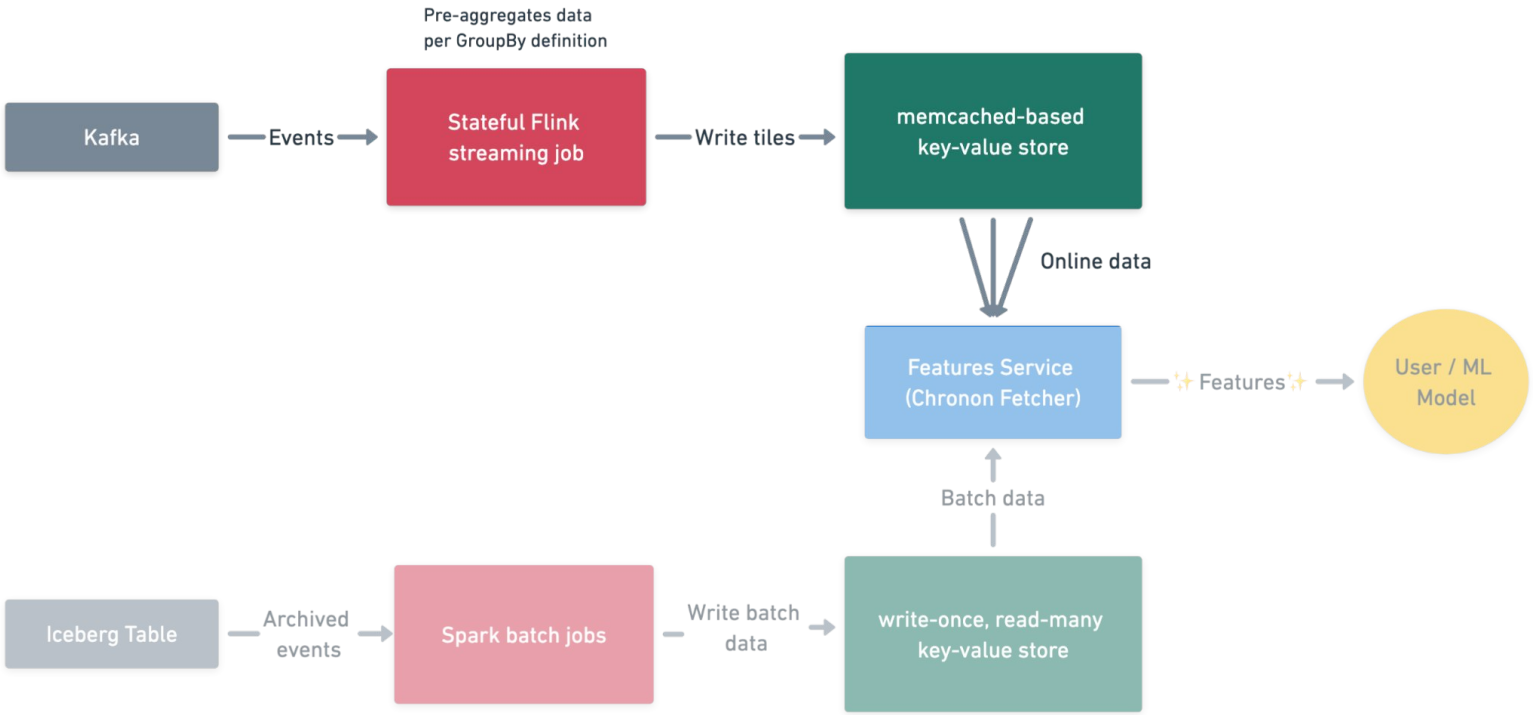
Amount of money charged on a credit card over the last day.

```
card_transactions_group_by = GroupBy(
    sources=card_transactions_source,
    keys=["card"],
    aggregations=[
        Aggregation(
            input_column="amount_in_dollars",
            operation=Operation.SUM,
            windows=[
                Window(length=1, timeUnit=TimeUnit.DAYS)
            ],
        ),
    ],
    accuracy=Accuracy.TEMPORAL, # streaming feature
)
```

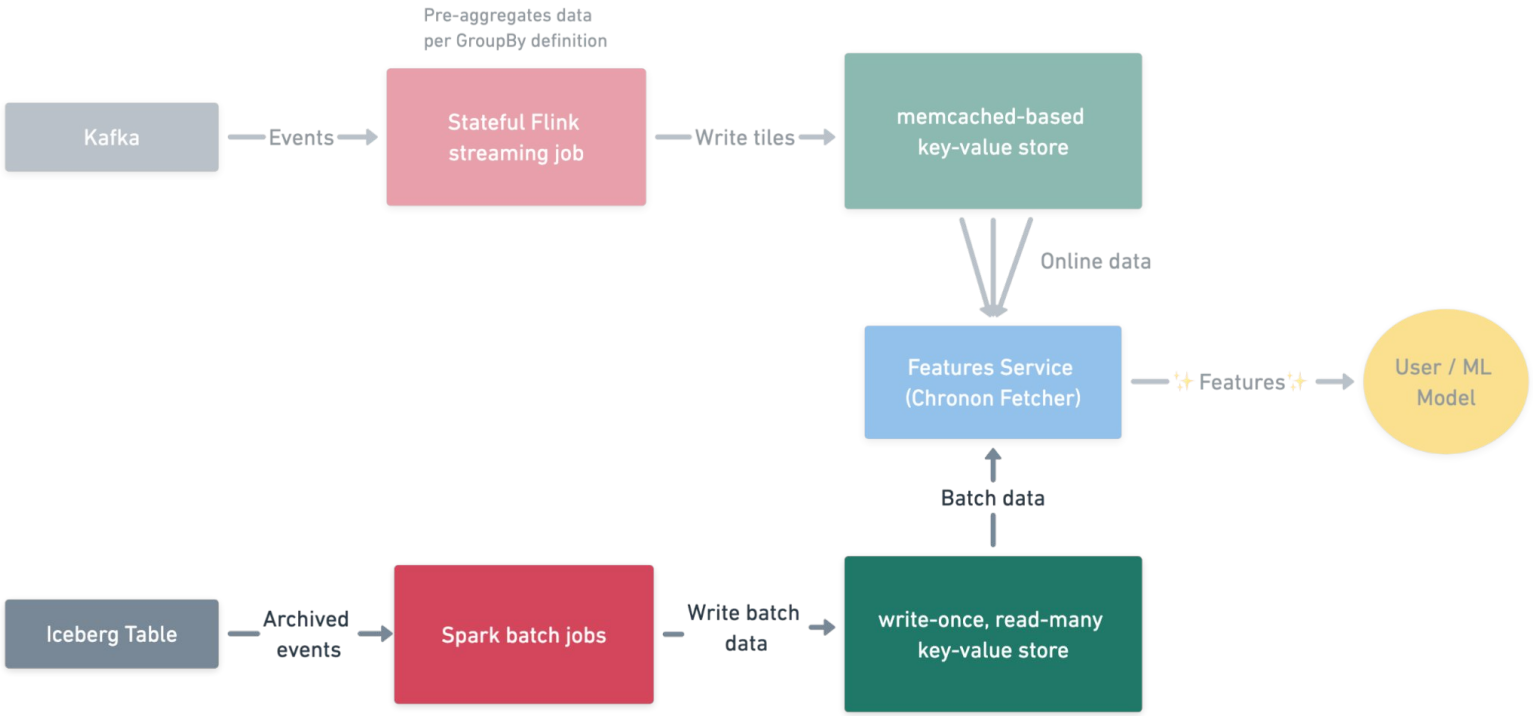
Shepherd Architecture



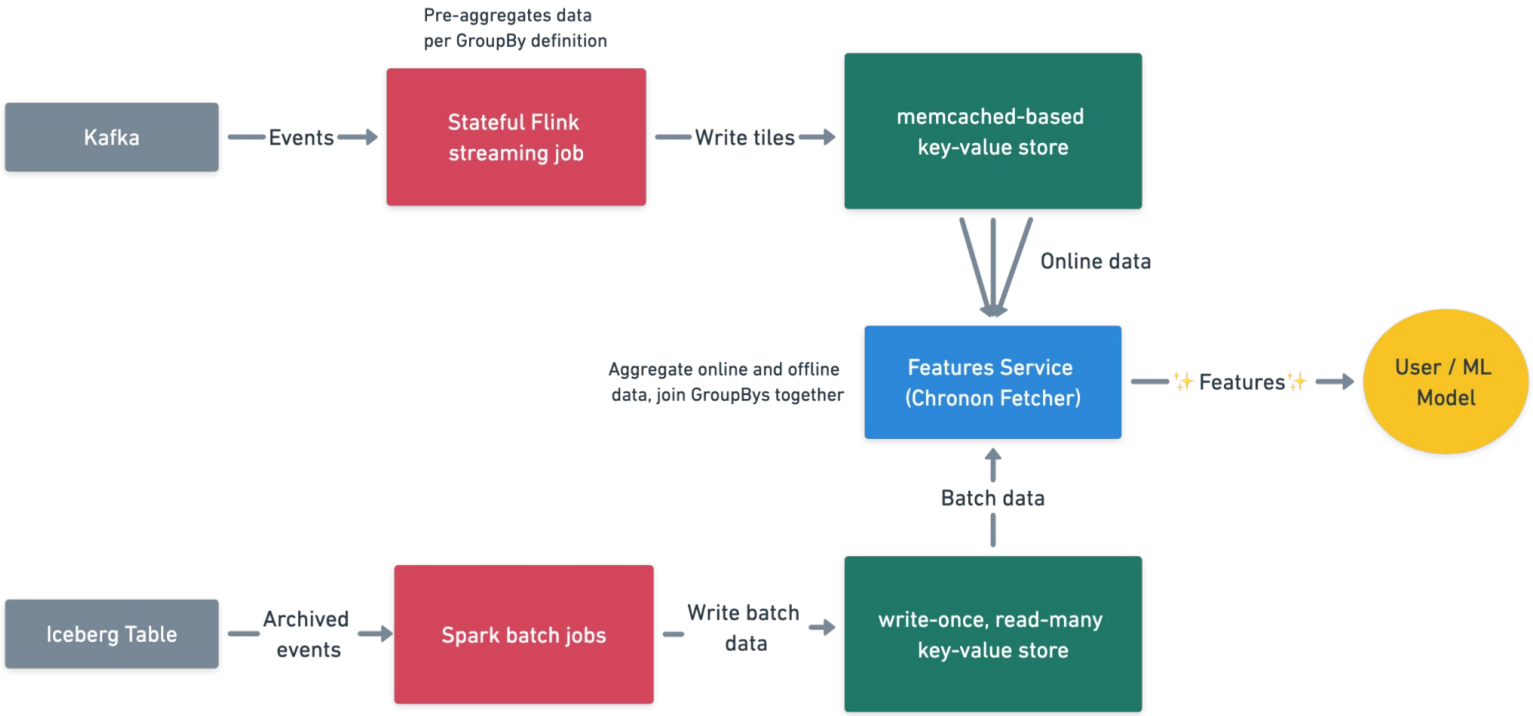
Shepherd Architecture



Shepherd Architecture



Shepherd Architecture



Correctness: Online-Offline Consistency

Spark on Flink

To guarantee values computed online (in Flink) and offline (in Spark) are identical, we run the same Chronon code in both platforms.

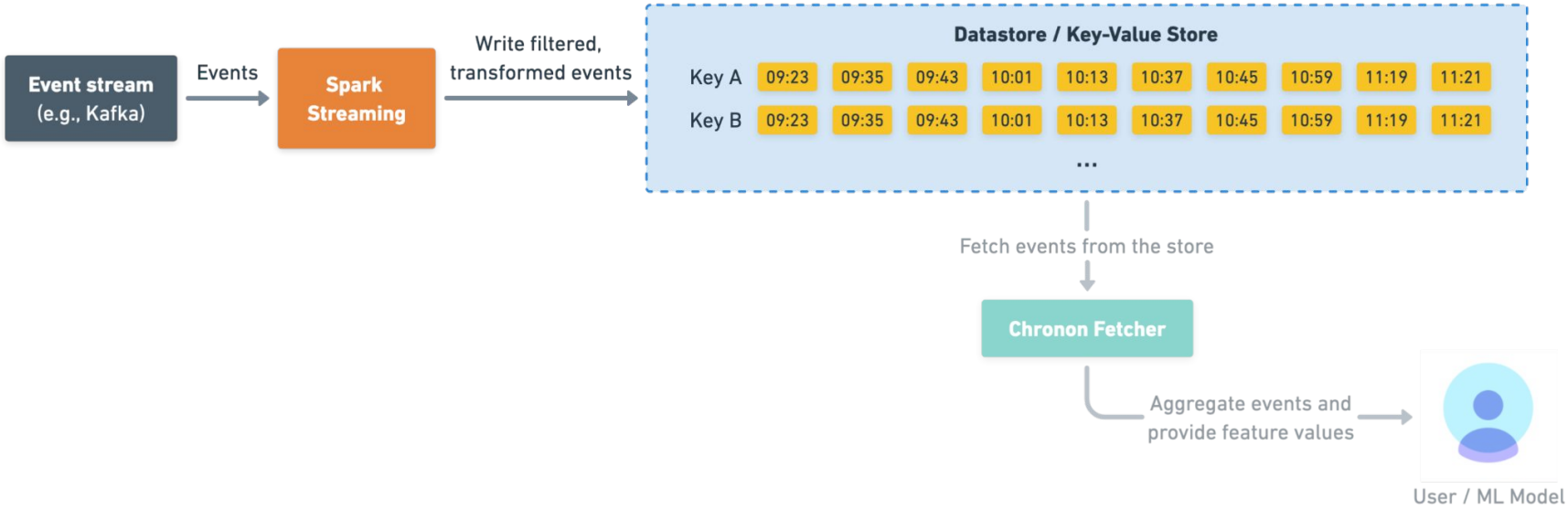
Because Chronon supports Spark SQL, we need to run Spark inside of Flink. We achieve this by directly utilizing Catalyst, Spark's query optimizer.

Our custom CatalystUtil[1] class extracts optimized execution plans from Spark DataFrames and creates transformation functions that can be applied in Flink at low latency.

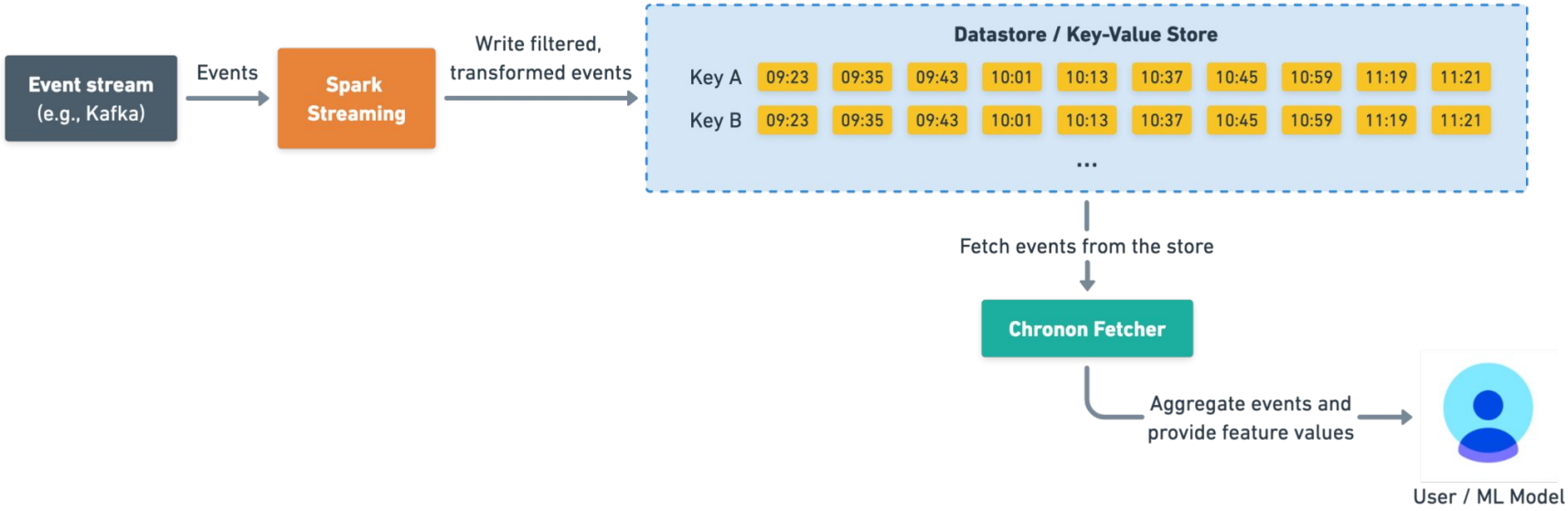
[1] See *CatalystUtil.scala* in github.com/airbnb/chronon if you are curious.

Feature Freshness: Streaming Job Architecture

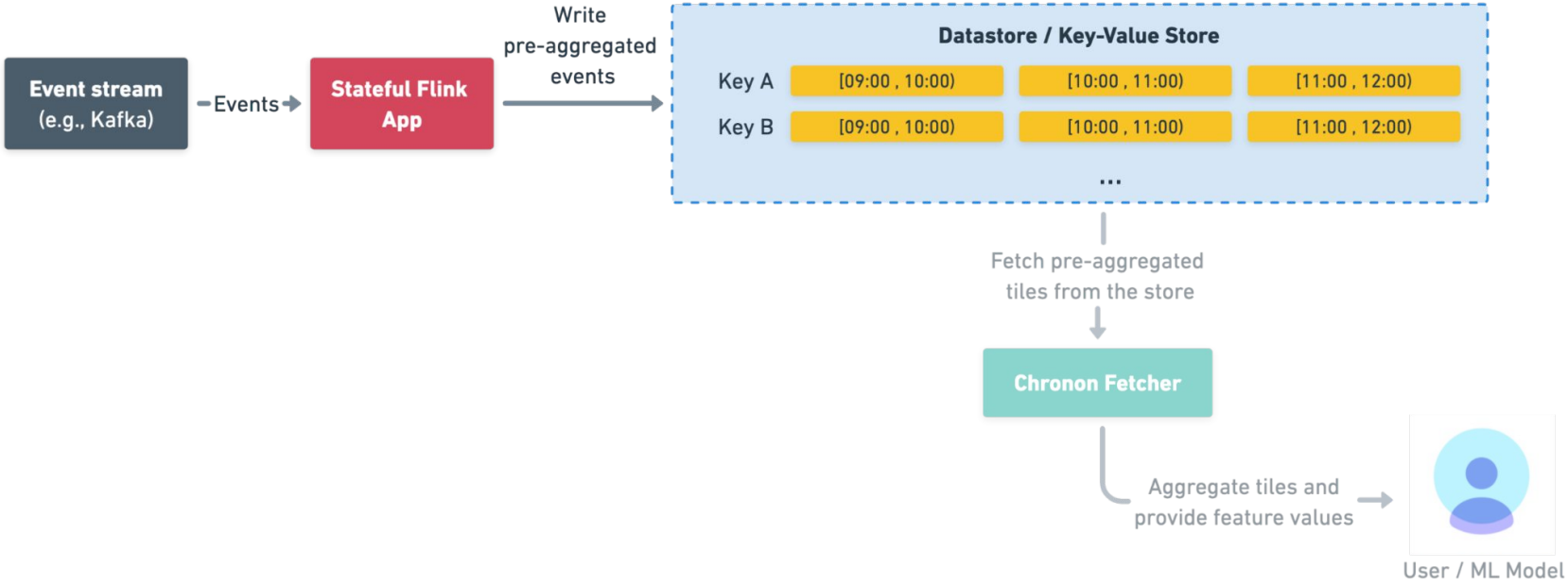
Default Chronon Online Architecture



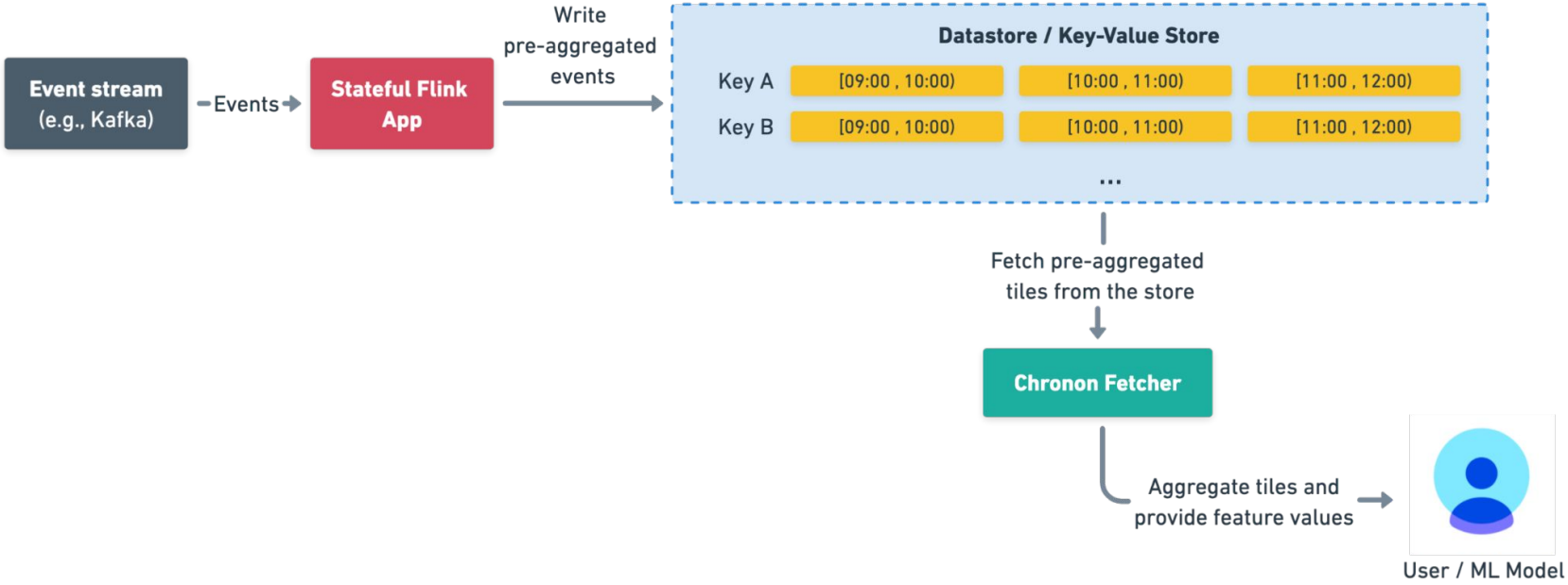
Default Chronon Online Architecture



Stripe's Tiled Flink Architecture



Stripe's Tiled Flink Architecture



Tiled Architecture: Feature Serving Example

In this architecture, the Flink job processes the Source and pre-aggregates based on the GroupBy definition. For example, if the following events arrive in Kafka

```
00:14 -> [“card_A”, 5.99]
00:45 -> [“card_A”, 30.00]
00:59 -> [“card_B”, 60.00]
01:12 -> [“card_A”, 40.00]
01:33 -> [“card_C”, 2.00]
01:34 -> [“card_C”, 34.00]
```

Tiled Architecture: Feature Serving Example

In this architecture, the Flink job processes the Source and pre-aggregates based on the GroupBy definition. For example, if the following events arrive in Kafka

```
00:14 -> [“card_A”, 5.99]
00:45 -> [“card_A”, 30.00]
00:59 -> [“card_B”, 60.00]
01:12 -> [“card_A”, 40.00]
01:33 -> [“card_C”, 2.00]
01:34 -> [“card_C”, 34.00]
```

Flink would pre-aggregate the `SUM` and store the following data in the KV store:

```
[00:00, 01:00), “card_A” -> [35.99]
[00:00, 01:00), “card_B” -> [60.00]
[01:00, 02:00), “card_A” -> [40.00]
[01:00, 02:00), “card_C” -> [36.00]
```

Then, when serving feature values, the Fetcher will gather and merge the relevant tiles.

Low-Latency Feature Serving

The Chronon Fetcher (Simplified)

For example, at 01:30 AM, to compute our previously-defined feature for `card_A`, the service would fetch and merge the following data.

From the batch key-value store:

```
[01:00 -1d, 00:00), “card_A” -> [200.00]
```

The Chronon Fetcher (Simplified)

For example, at 01:30 AM, to compute our previously-defined feature for `card_A`, the service would fetch and merge the following data.

From the batch key-value store:

```
[01:00 -1d, 00:00), “card_A” -> [200.00]
```

From the streaming key-value store:

```
[00:00, 01:00), “card_A” -> [35.99]
[01:00, 02:00), “card_A” -> [40.00]
```

The Chronon Fetcher (Simplified)

For example, at 01:30 AM, to compute our previously-defined feature for `card_A`, the service would fetch and merge the following data.

From the batch key-value store:

```
[01:00 -1d, 00:00), “card_A” -> [200.00]
```

From the streaming key-value store:

```
[00:00, 01:00), “card_A” -> [35.99]
[01:00, 02:00), “card_A” -> [40.00]
```

Final feature vector computed: `[275.99]`

The Shepherd Control Plane

Shepherd Onboarding Challenges

- Flink application provisioning
- Creation of a key-value store for streaming data
- Batch dataset registration for each Feature Group
- Shepherd API tier creation per use case for feature serving isolation

Number of open Jira tickets per day to register batch datasets

Storage Pools

Storage Pools contain information required to automatically provision Shepherd infrastructure, and are assigned to feature groups. This abstraction promotes feature serving isolation.

Storage Pool assignment to a feature group

```
card_transactions_group_by = GroupBy(
    sources=card_transactions_source,
    keys=["card"],
    aggregations=[
        Aggregation(
            input_column="amount_in_dollars",
            operation=Operation.SUM,
            windows=[
                Window(length=1, timeUnit=TimeUnit.DAYS)
            ],
        ),
    ],
    accuracy=Accuracy.TEMPORAL,
    online=True,
    storage_pool="cardtesting", # pool assignment
)
```

Storage Pools

Storage Pools contain information required to automatically provision Shepherd infrastructure, and are assigned to feature groups. This abstraction promotes feature serving isolation.

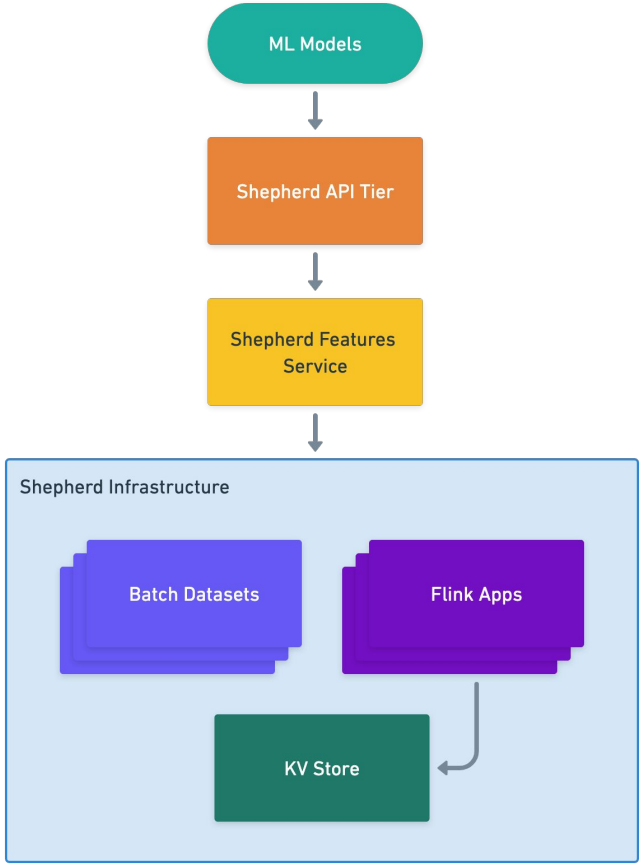
Storage Pool assignment to a feature group

```
card_transactions_group_by = GroupBy(
    sources=card_transactions_source,
    keys=["card"],
    aggregations=[
        Aggregation(
            input_column="amount_in_dollars",
            operation=Operation.SUM,
            windows=[
                Window(length=1, timeUnit=TimeUnit.DAYS)
            ],
        ),
    ],
    accuracy=Accuracy.TEMPORAL,
    online=True,
    storage_pool="cardtesting", # pool assignment
)
```

Storage Pool configuration

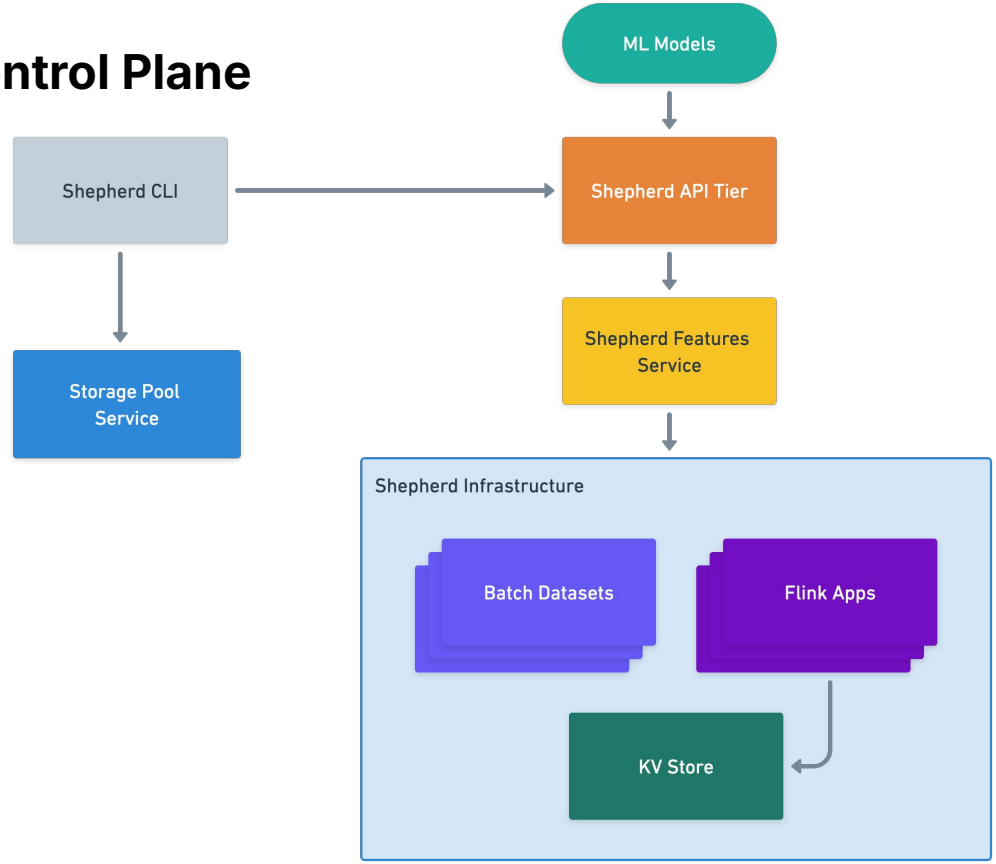
```
metadata:
  name: cardtesting
  project: shepherd
spec:
  environments:
  - production
  expected_peak_rps: 1
  feature_groups:
  - name: card_transactions_group_by
    online: true
  priority: TEST
  regions:
  - northwest
  slack_channel: '#cardtesting-alerts'
```

Onboarding with the Shepherd Control Plane



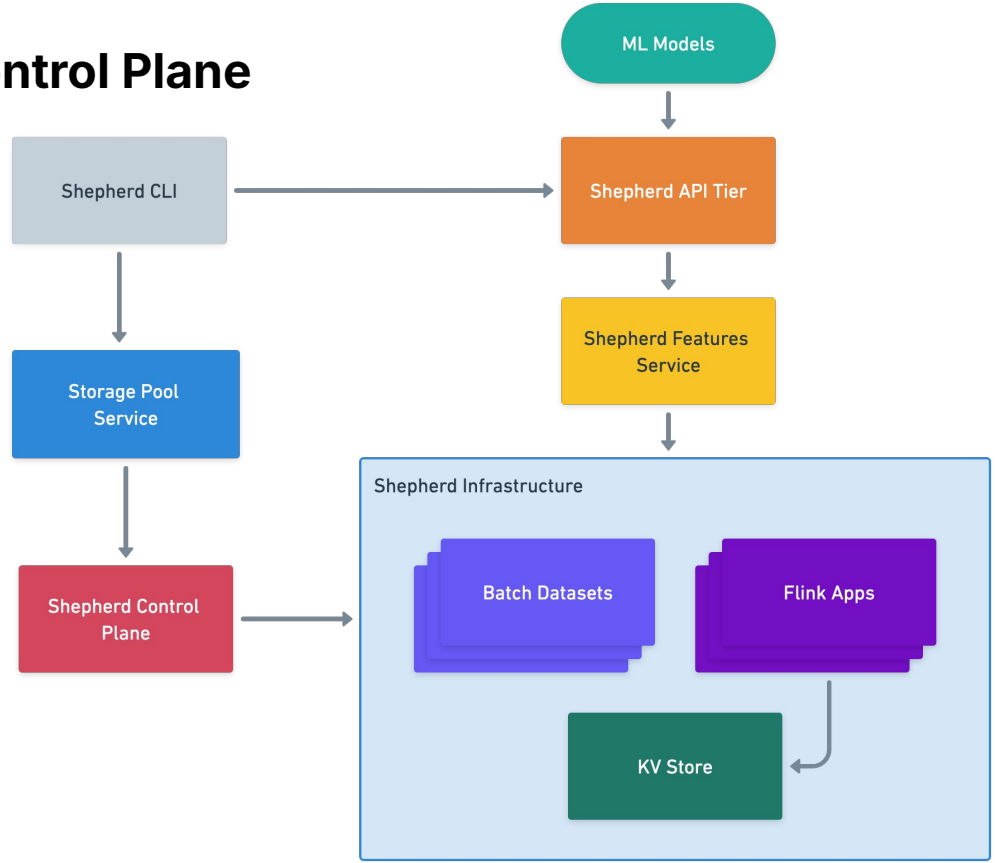
Onboarding with the Shepherd Control Plane

- User CLI tooling to auto-generate
  - Storage Pool configurations
  - Deployment management
  - New Shepherd API tier



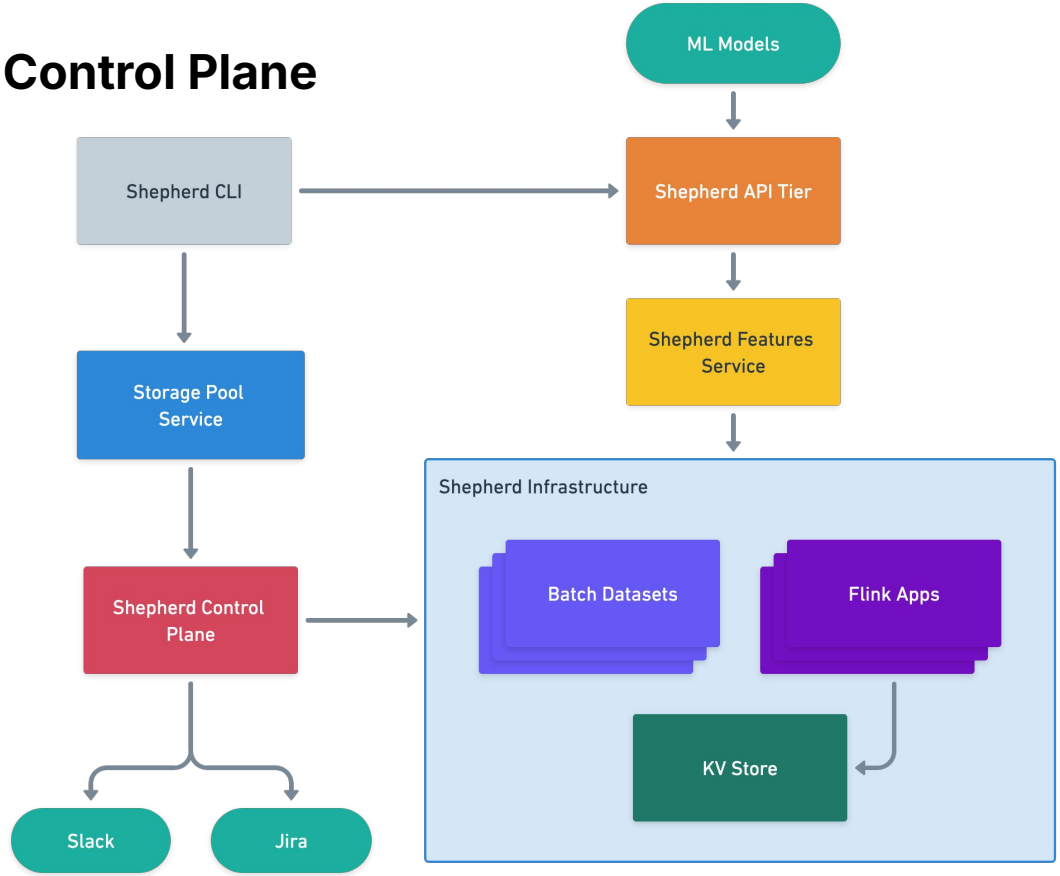
Onboarding with the Shepherd Control Plane

- User CLI tooling to auto-generate
  - Storage Pool configurations
  - Deployment management
  - New Shepherd API tier
- Daily pool deployments
- Single-tenant Flink apps



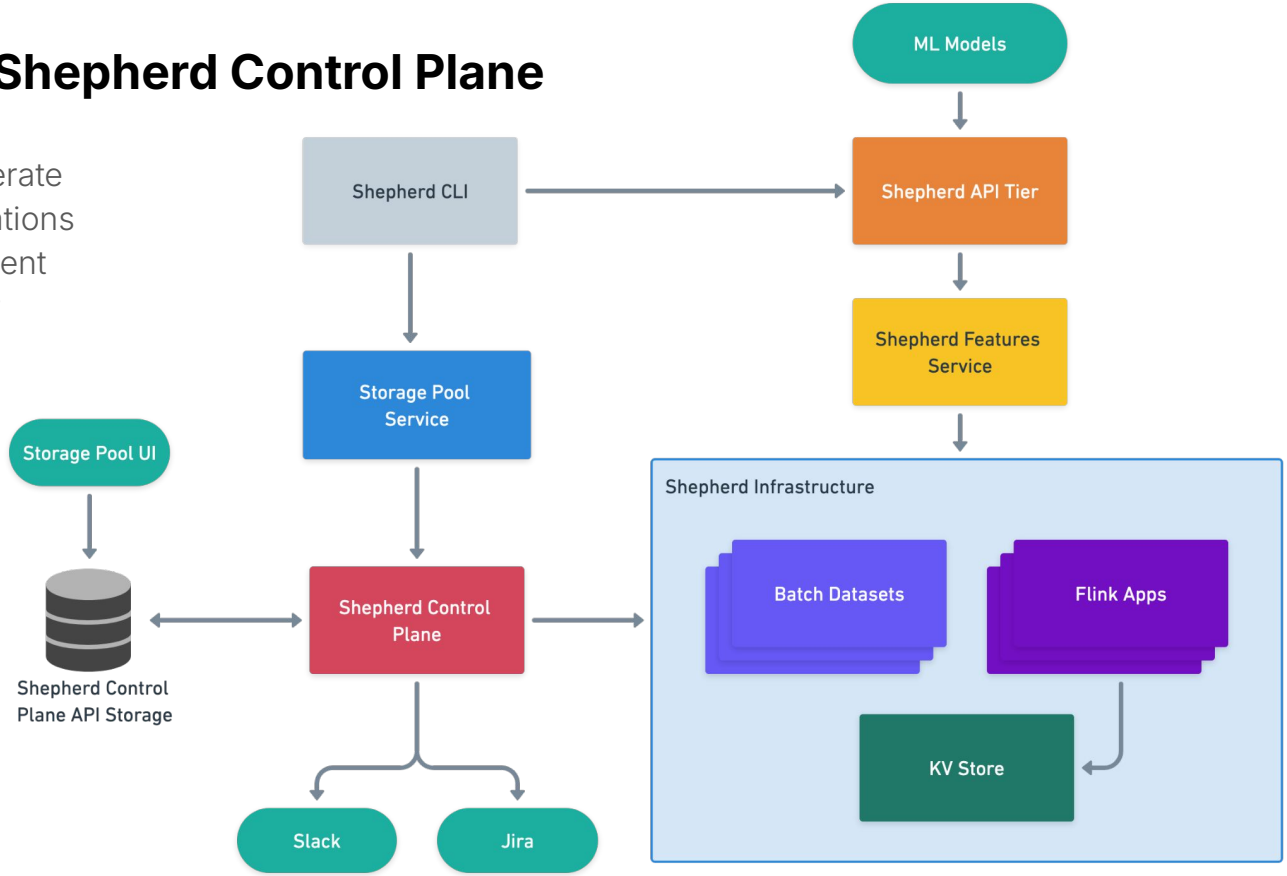
Onboarding with the Shepherd Control Plane

- User CLI tooling to auto-generate
  - Storage Pool configurations
  - Deployment management
  - New Shepherd API tier
- Daily pool deployments
- Single-tenant Flink apps
- Notifications



Onboarding with the Shepherd Control Plane

- User CLI tooling to auto-generate
  - Storage Pool configurations
  - Deployment management
  - New Shepherd API tier
- Daily pool deployments
- Single-tenant Flink apps
- Notifications
- Observability



Impact and Looking Forward

- Accelerated feature development for improved model performance
- High Availability via automated cross-region replication
- Growth in fraud loss savings
- Reduced toil for ML infrastructure teams

Thank you!

chronon.ai

stripe.com/jobs

caiocamatta@stripe.com
divyamanohar@stripe.com