cube

Headless BI with streaming data

Artyom Keydunov
Co-founder and CEO at Cube

Pavel Tiunov
PhD, Co-founder and CTO at Cube

Igor Lukanin
Head of Developer Relations

Community Code of Conduct

We want to foster an open and welcoming environment where everyone feels they belong in the Cube community.

The full text of our Code of Conduct is available at github.com/cube-js/cube.js/blob/master/CODE_OF_CONDUCT.md

Any instances of inappropriate/unacceptable behavior can be reported to conduct@cube.dev

Quick notes

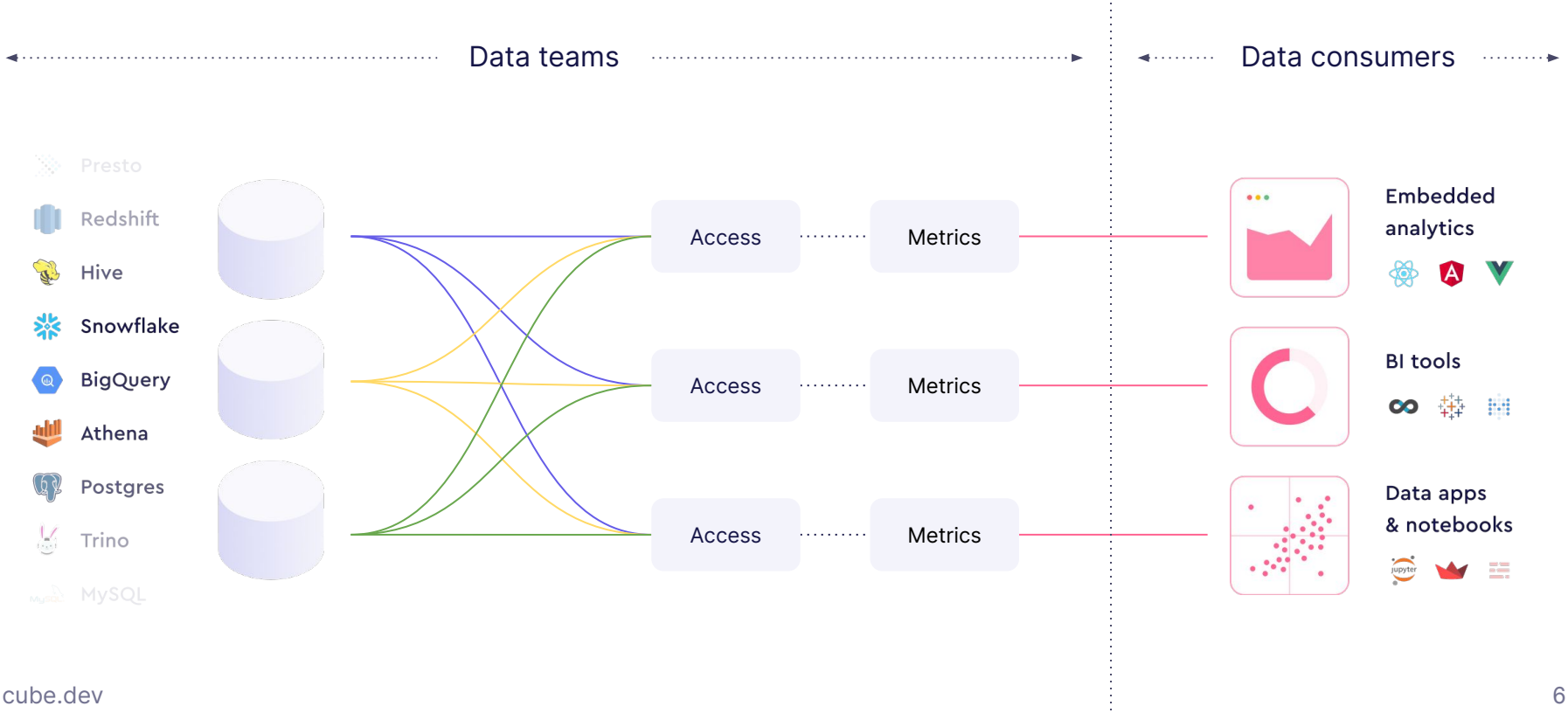
- If you have any questions during the workshop, please type them in the “Q&A” section (on Zoom & YouTube)
- We will be using Cube Cloud for “hands-on” demos
- Recording of the workshop will be posted on the Cube’s YouTube channel
- All attendees will receive a post-event survey and we’d appreciate your feedback to help us with future events

What we will discuss today

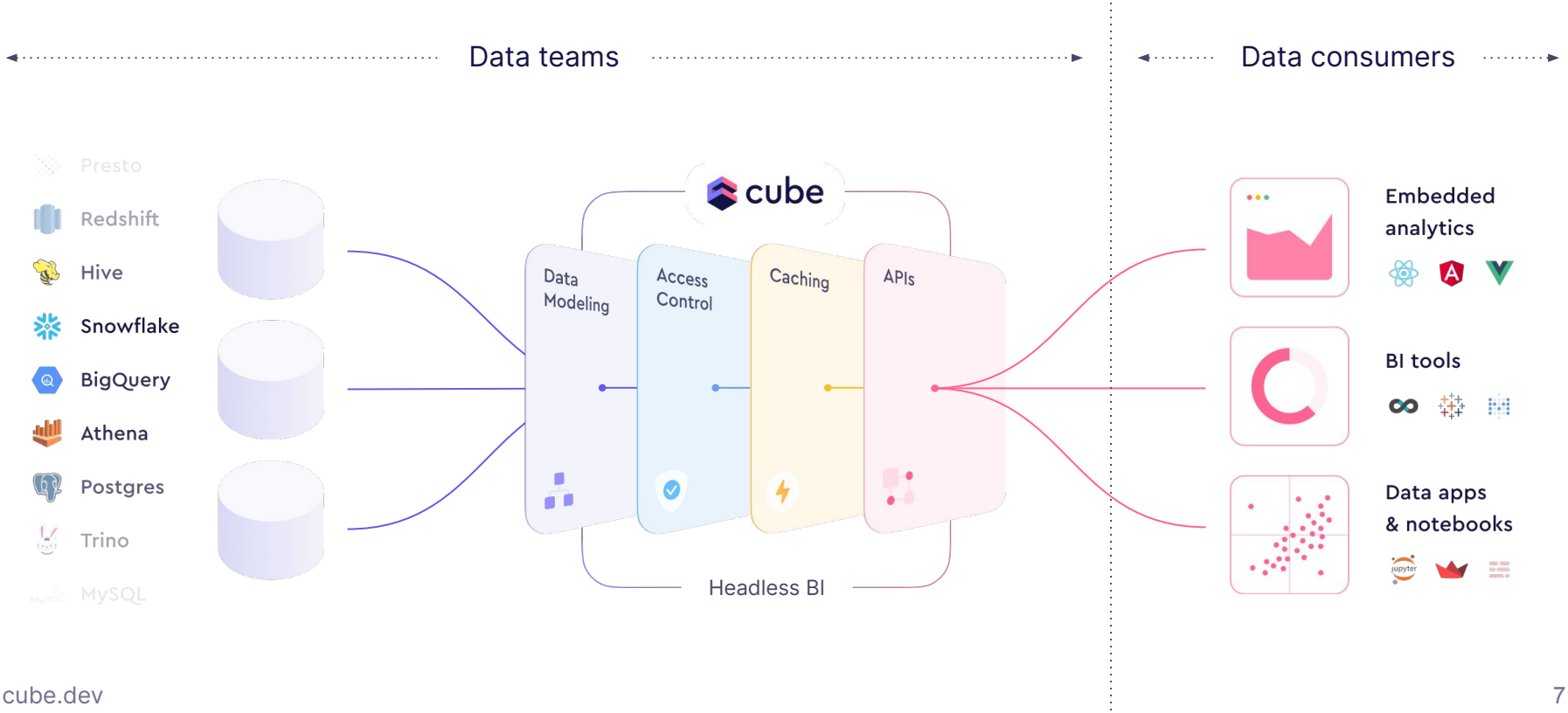
- What is headless BI
  - New feature just released 🚀
- Headless BI with streaming data
- ksqlDB integration — and Q & A
- Lambda pre-aggregations — and Q & A

Headless BI

Data teams have a many-to-many problem



Cube solves for consistency and accessibility



Cube — Headless BI

Cube helps to...

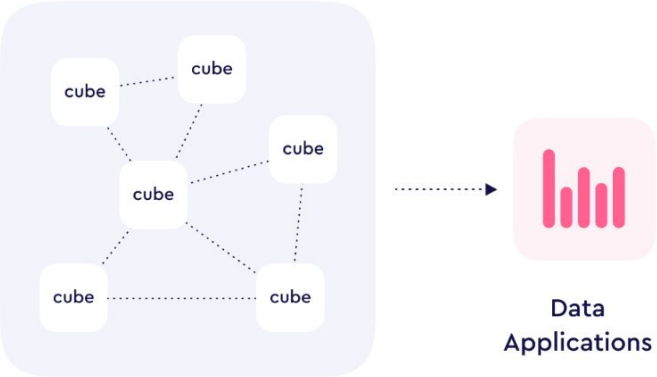
- consume data from modern data stores,
- organize it into consistent definitions, and
- deliver it to every application.

Cube's mission is making data consistent and accessible.

In-depth blog post: cube.dev/blog/headless-bi

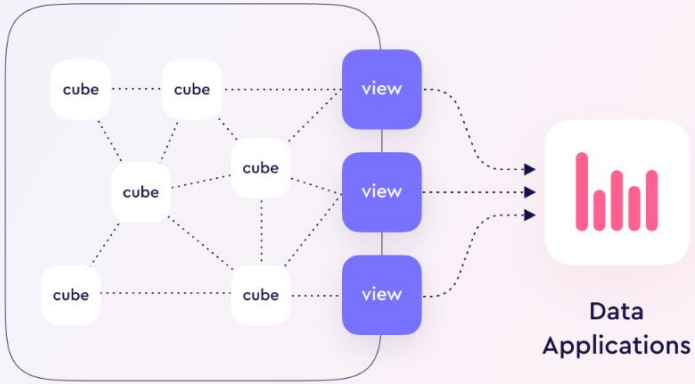
Just released: Views

Before



How to define metrics? How to control join path?
How to manage visibility of cubes
to different teams/users?

Now



Views are the place to define metrics, control ambiguousjoin paths, and manage governance and accessibility for data consumers.

Views in Cube’s data modeling layer

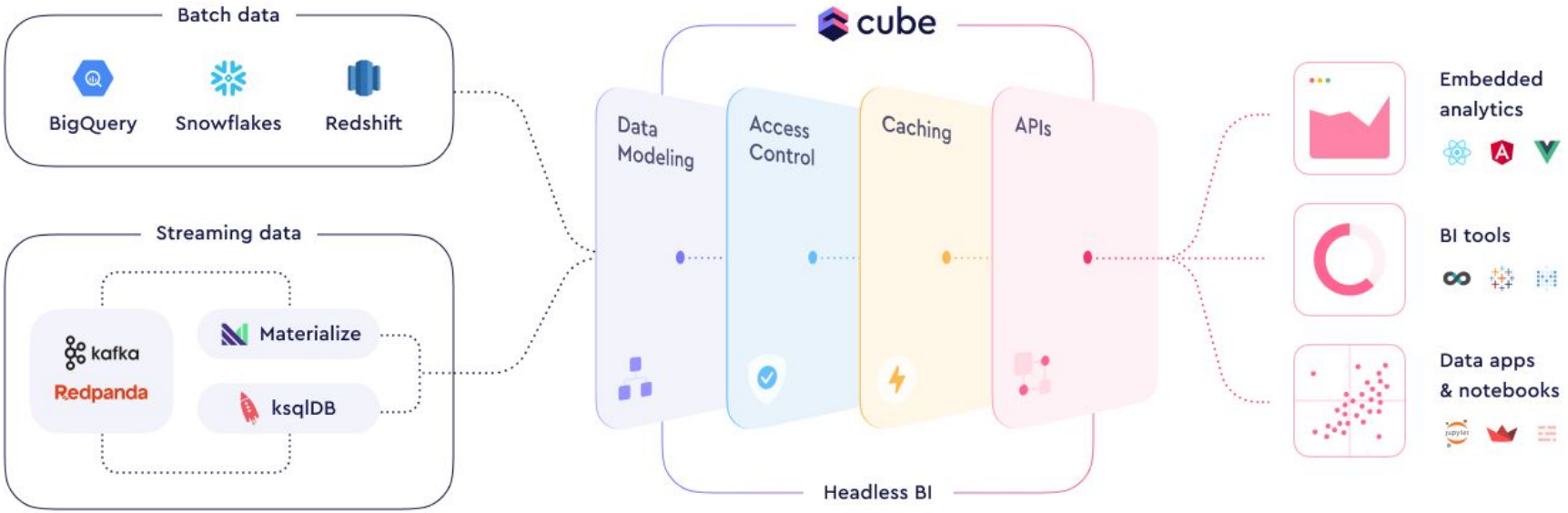
In-depth blog post: cube.dev/blog/introducing-views

Documentation: cube.dev/docs/schema/reference/view

We’ll host a workshop on Views on November 2. Please RSVP at cube.dev/events/using-cube-views-for-metrics-management

Headless BI with streaming data

Run Cube on top of SQL streaming engines



SQL streaming engines

Supported:

- ksqlDB — works with Apache Kafka and Redpanda

Work-in-progress:

- Materialize
- Flink SQL
- Spark Streaming

Let us know if you'd like to use Cube with these or other streaming engines: cube.dev/contact

ksqlDB integration

What is ksqlDB?

Streaming database, purpose-built for stream processing applications.

- Works with Apache Kafka
  - Also, with Redpanda: redpanda.com/blog/ksqldb-materialized-cache
- Allows to turn continuous *event streams* into *materialized views* (tables) and access them with SQL

How to work with ksqlDB (1/2)

1. Connect ksqlDB to the event streaming platform
2. Create event streams over topics:

```
CREATE STREAM geoEvents (...) WITH (
  KAFKA_TOPIC = 'my-events'
);
```

3. If needed, continuously transform and reshape them:

```
CREATE STREAM locations AS
  SELECT ...
  FROM geoEvents
EMIT CHANGES;
```

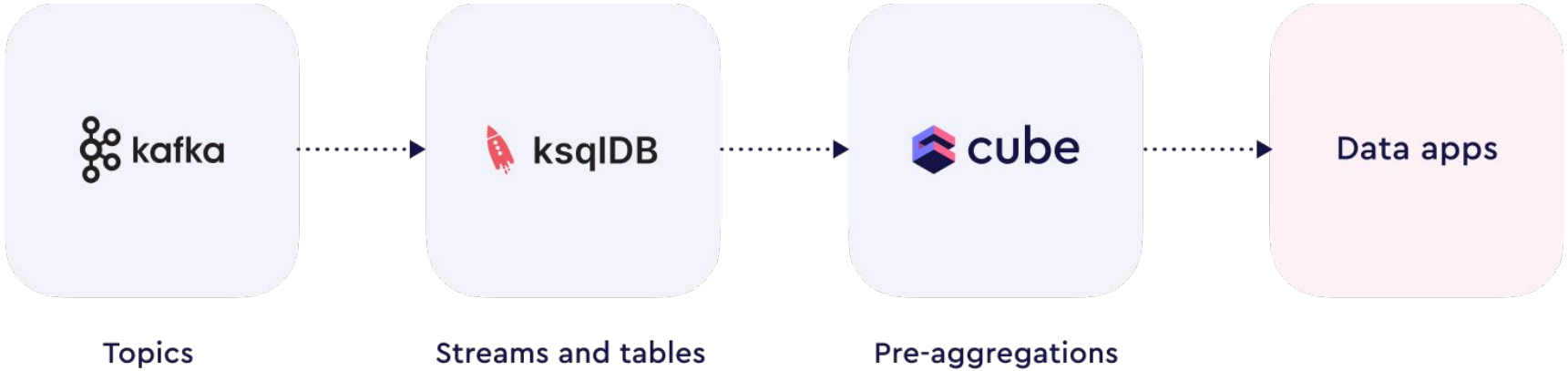
How to work with ksqlDB (2/2)

4. Create ever-updating materialized views:

```
CREATE TABLE activePromotions AS
  SELECT rideId, qualifyPromotion(kmToDst) AS promotion
  FROM locations
  GROUP BY rideId
EMIT CHANGES;
```

5. Access streaming data with *pull queries* or *push queries*
   - Pull query: `SELECT ... FROM ...;`
   - Push query: `SELECT ... FROM ... EMIT CHANGES;`

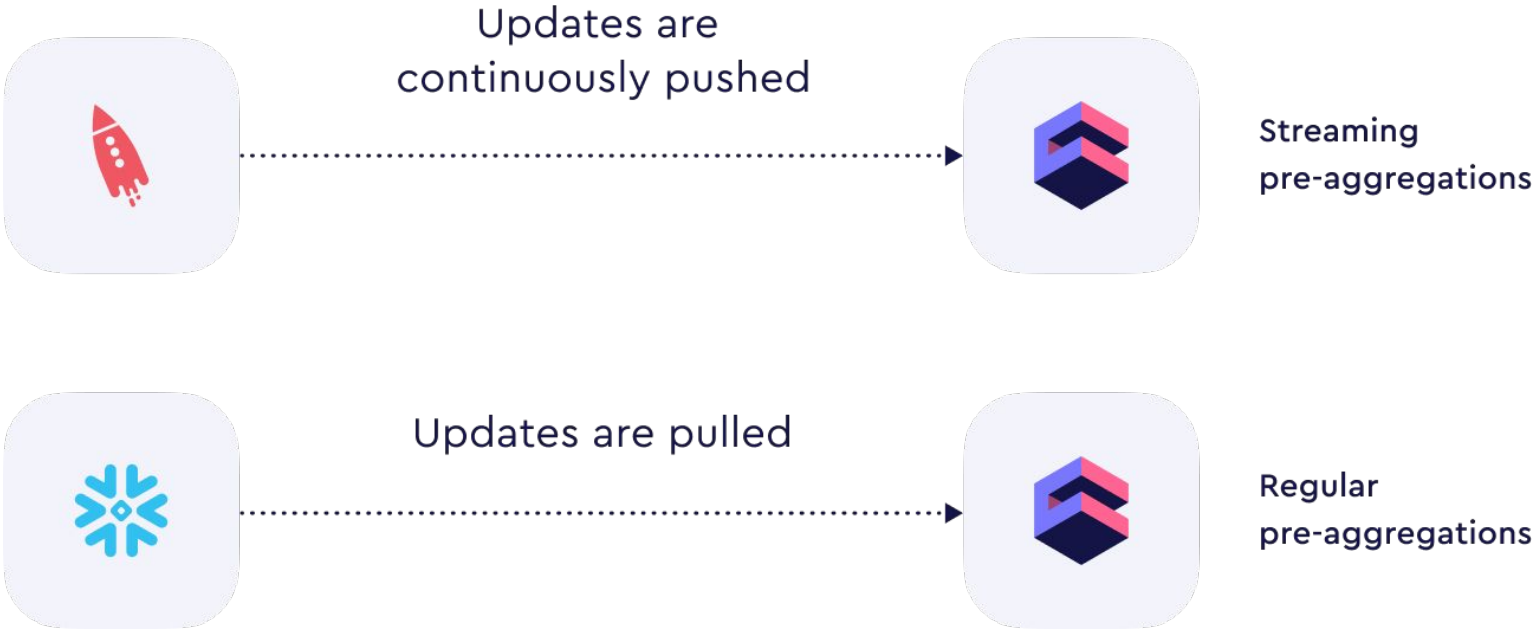
How Cube works with ksqlDB



How Cube works with ksqlDB

1. Connect using the `ksql` database driver: cube.dev/docs/config/databases/ksqldb
2. Define a cube with a SQL query
   - Cube will add `EMIT CHANGES` for you
3. Define a *streaming* pre-aggregation ⚡
   - Cube will issue push queries to ksqlDB and subscribe to updates
   - Cube will keep the pre-aggregation up-to-date in the background
   - Cube will serve queries from the pre-aggregation

Streaming pre-aggregations



Streaming pre-aggregations

- Streaming pre-aggregations use push queries
- Cube subscribes to updates and keeps the pre-aggregation up-to-date
  - No need to define `refreshKey`
- Whether pre-aggregation is streaming or not is defined by the data source
  - Currently, only ksqlDB is supported

Documentation: cube.dev/docs/caching/using-pre-aggregations #streaming-pre-aggregations

Demo: ksql-demo.netlify.app

Source code on GitHub:
github.com/cube-js/cube.js/tree/master/examples/ksql

Demo

Streaming roadmap

- ksqlDB
  - Partitions sealing
  - Offset selection
  - Replay from offset
  - Rollup validations and optimizations (disallow rollups without measures, treat `rollup` with primary key as `originalSql`, etc.)
- Materialize
  - Support streaming pre-aggregations in Materialize

Q & A

Lambda pre-aggregations

Lambda architecture

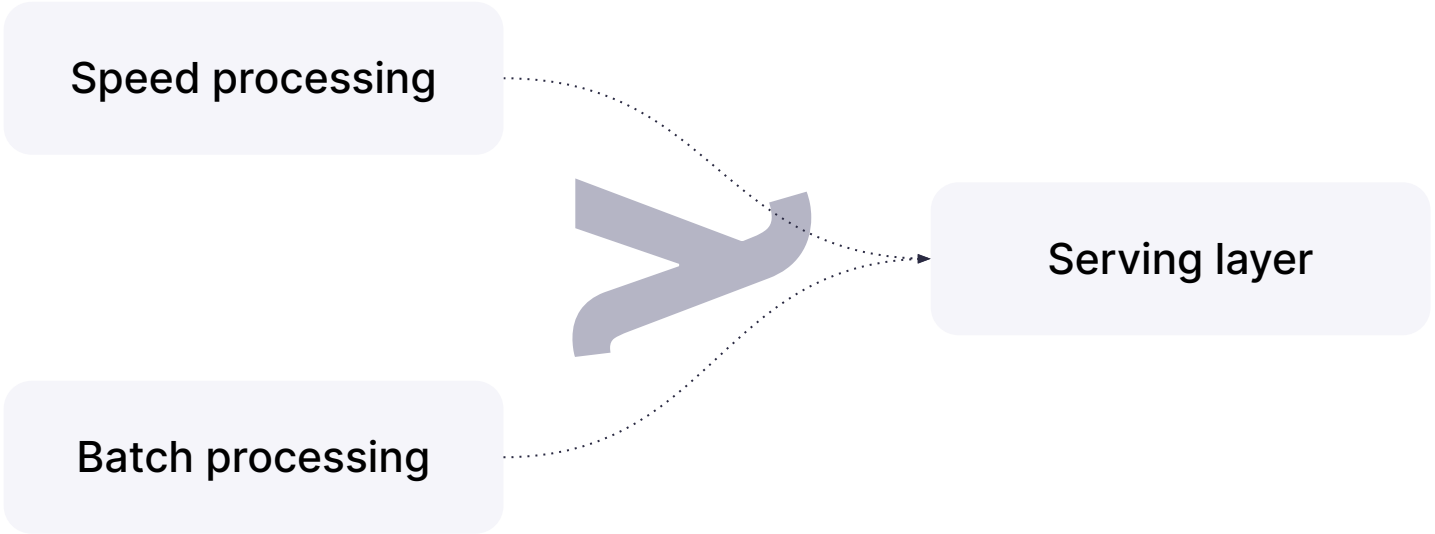
Lambda architecture is a data-processing architecture that is taking advantage of both batch and stream-processing methods.

It describes a system consisting of three layers:

- batch processing layer,
- speed (or real-time) processing layer,
- and a serving layer for responding to queries.

With lambda architecture, you can balance *throughput* and *latency*.

Lambda architecture

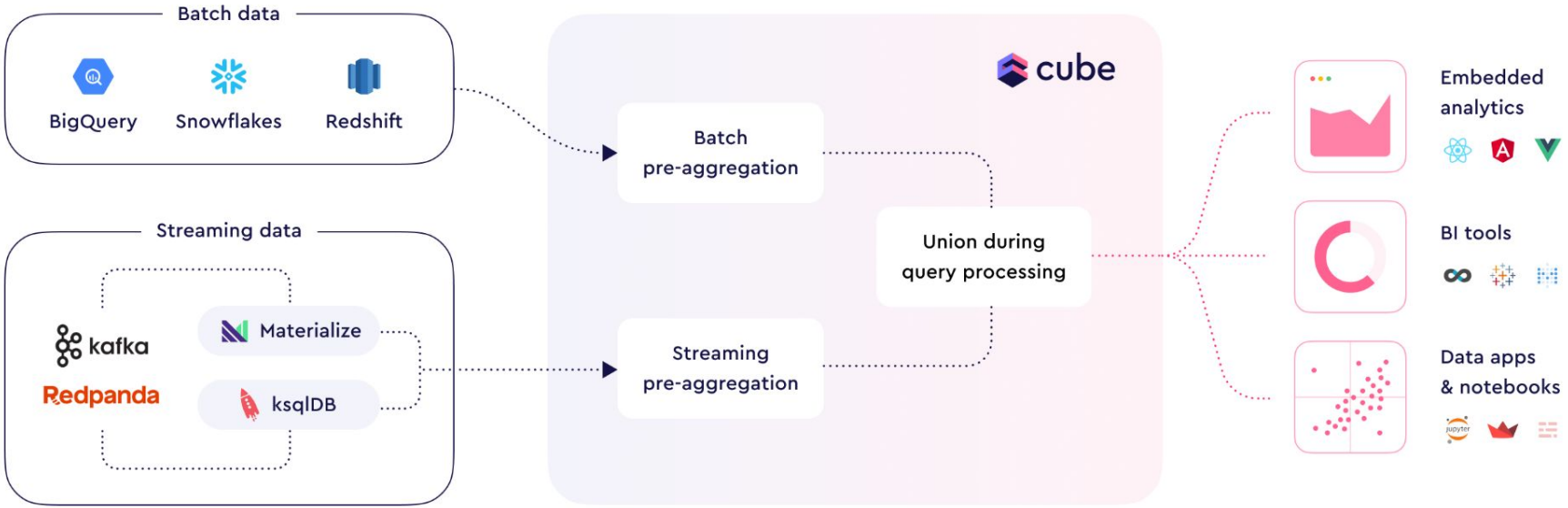


Lambda pre-aggregations

- Lambda pre-aggregations follow the lambda architecture design
- They union batch data and real-time data
  - Batch layer: regular *pre-aggregations*
  - Speed layer: *streaming pre-aggregations* or *source data*

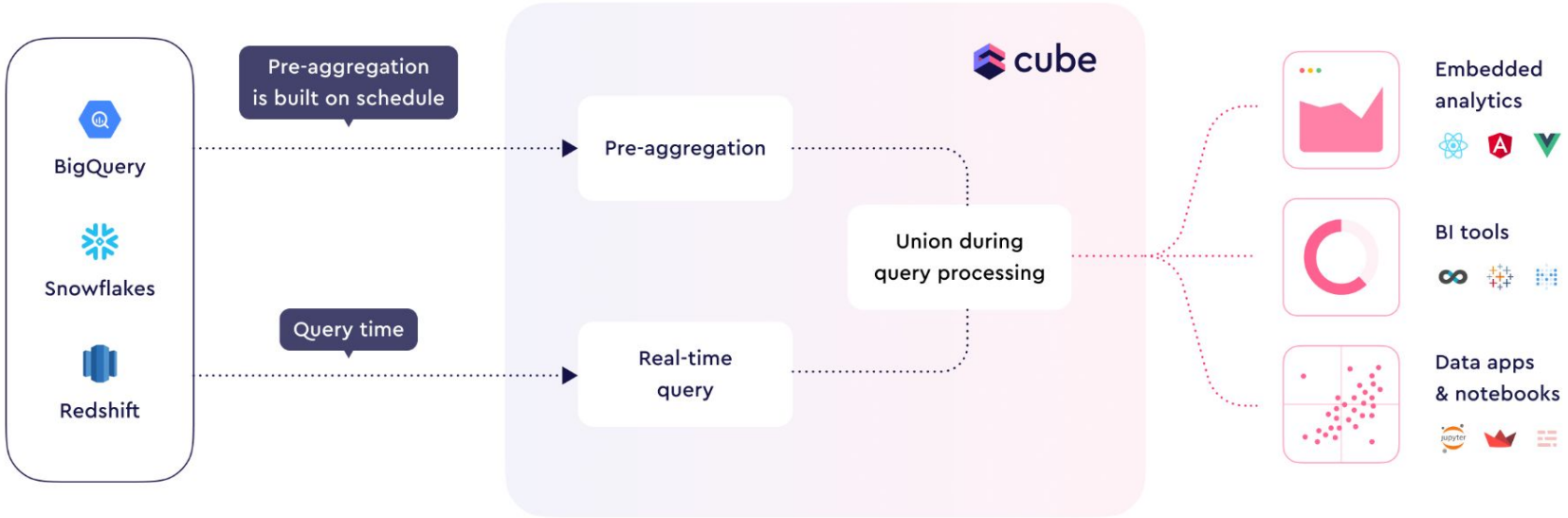
Docs: cube.dev/docs/caching/pre-aggregations/lambda-pre-aggregations

Batch and streaming data



```
batchStreamingLambda: {
  type: `rollupLambda`,
  rollups: [Users.batch, streaming]
},
batch: {
  type: `rollup`,
  measures: [Users.count],
  dimensions: [Users.name],
  timeDimension: Users.createdAt,
  granularity: `day`,
  buildRangeStart: {
    sql: `SELECT '2020-01-01'`
  },
  buildRangeEnd: {
    sql: `SELECT '2022-05-30'`
  }
},
streaming: {
  type: `rollup`,
  measures: [StreamingUsers.count],
  dimensions: [StreamingUsers.name],
  timeDimension: StreamingUsers.createdAt,
  granularity: `day`
}
```

Batch and source data



```
lambda: {
  type: `rollupLambda`,
  unionWithSourceData: true,
  rollups: [Users.batch]
},
batch: {
  measures: [Users.count],
  dimensions: [Users.name],
  timeDimension: Users.createdAt,
  granularity: `day`,
  buildRangeStart: {
    sql: `SELECT '2020-01-01'`
  },
  buildRangeEnd: {
    sql: `SELECT '2022-05-30'`
  }
}
```

Demo

Q & A

Learn more

Blog:

- cube.dev/blog/headless-bi
- cube.dev/blog/headless-bi-with-streaming-data

Documentation:

- cube.dev/docs/config/databases/ksqldb
- cube.dev/docs/caching/using-pre-aggregations#streaming-pre-agg...
- cube.dev/docs/caching/pre-aggregations/lambda-pre-aggregations

Slack community of 6500+ data engineers:

slack.cube.dev


cube