CONFLUENT

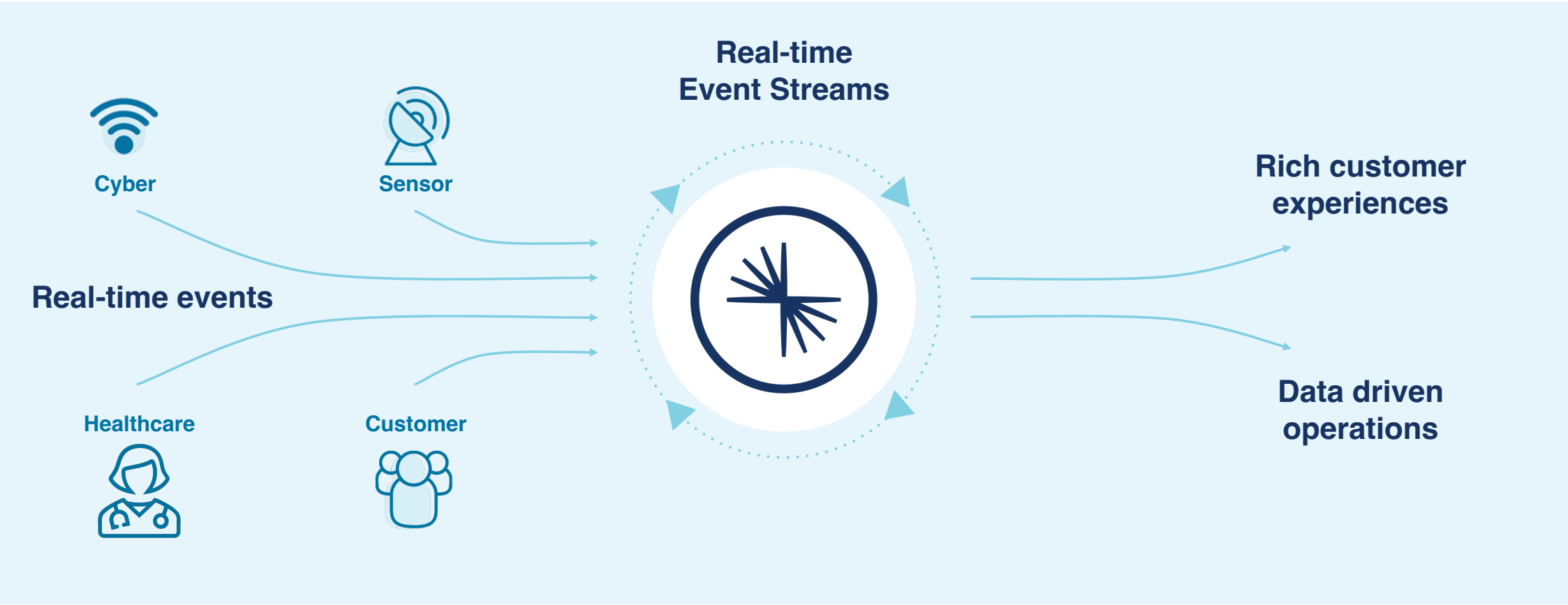
Adopting Data in Motion: The Modern-Event Driven Architecture

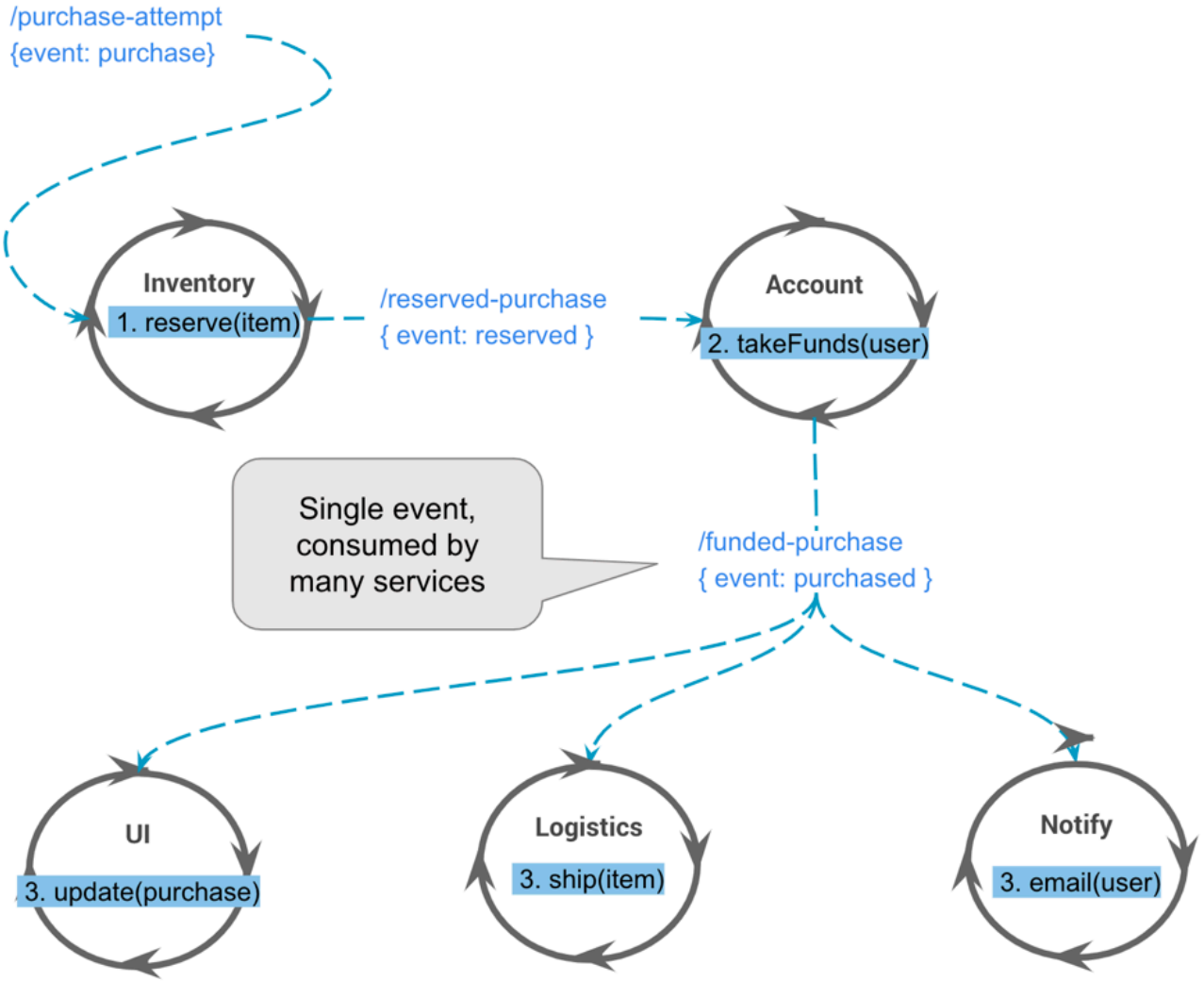
Alex Stuart
Advisory Solutions Engineer
astuart@confluent.io
ajfstuart

The Foundational Assumption of Every Database: Data at Rest

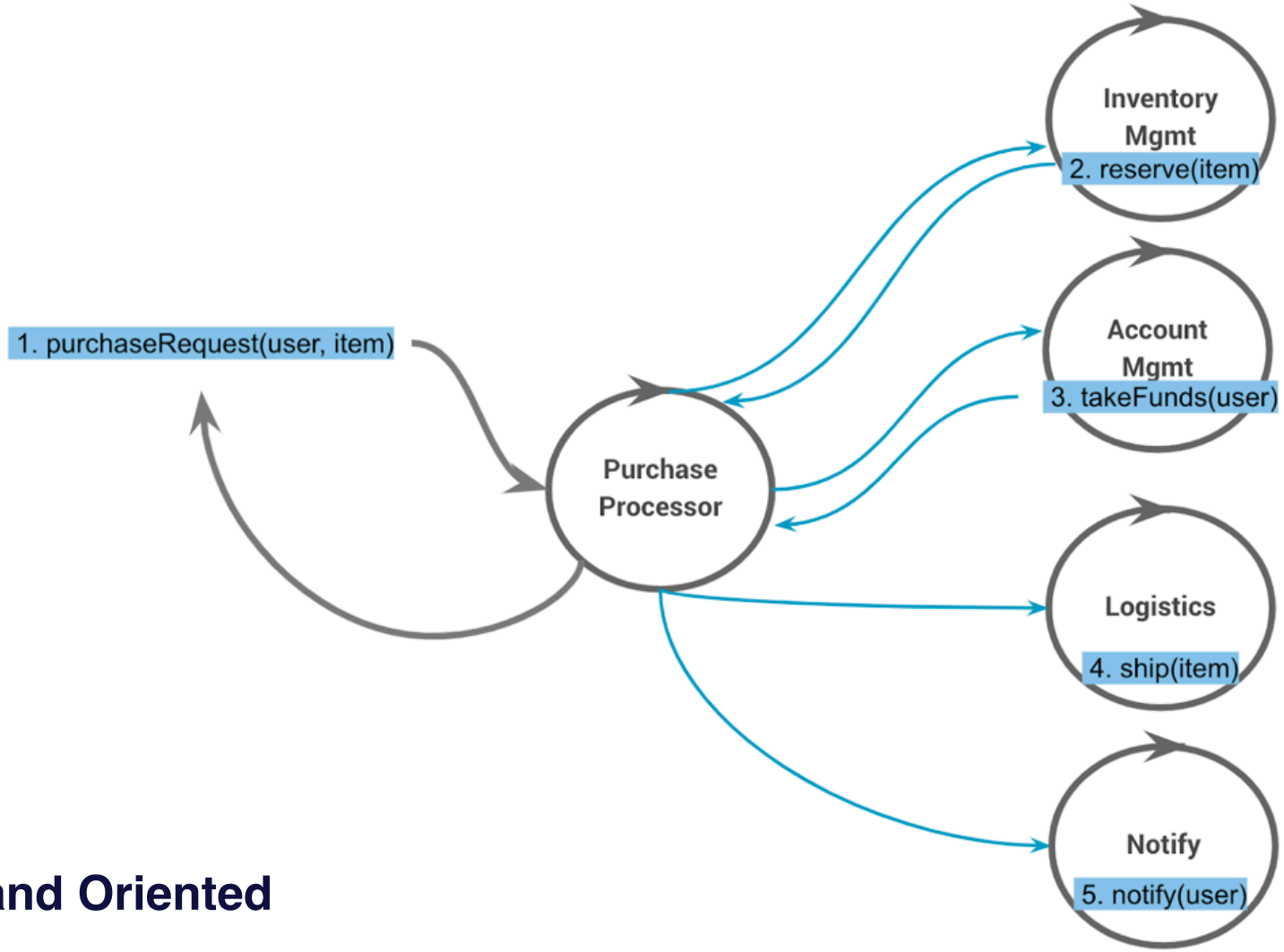
Data Is The New Engine of Business Success

Paradigm for Data in Motion: Event Streams

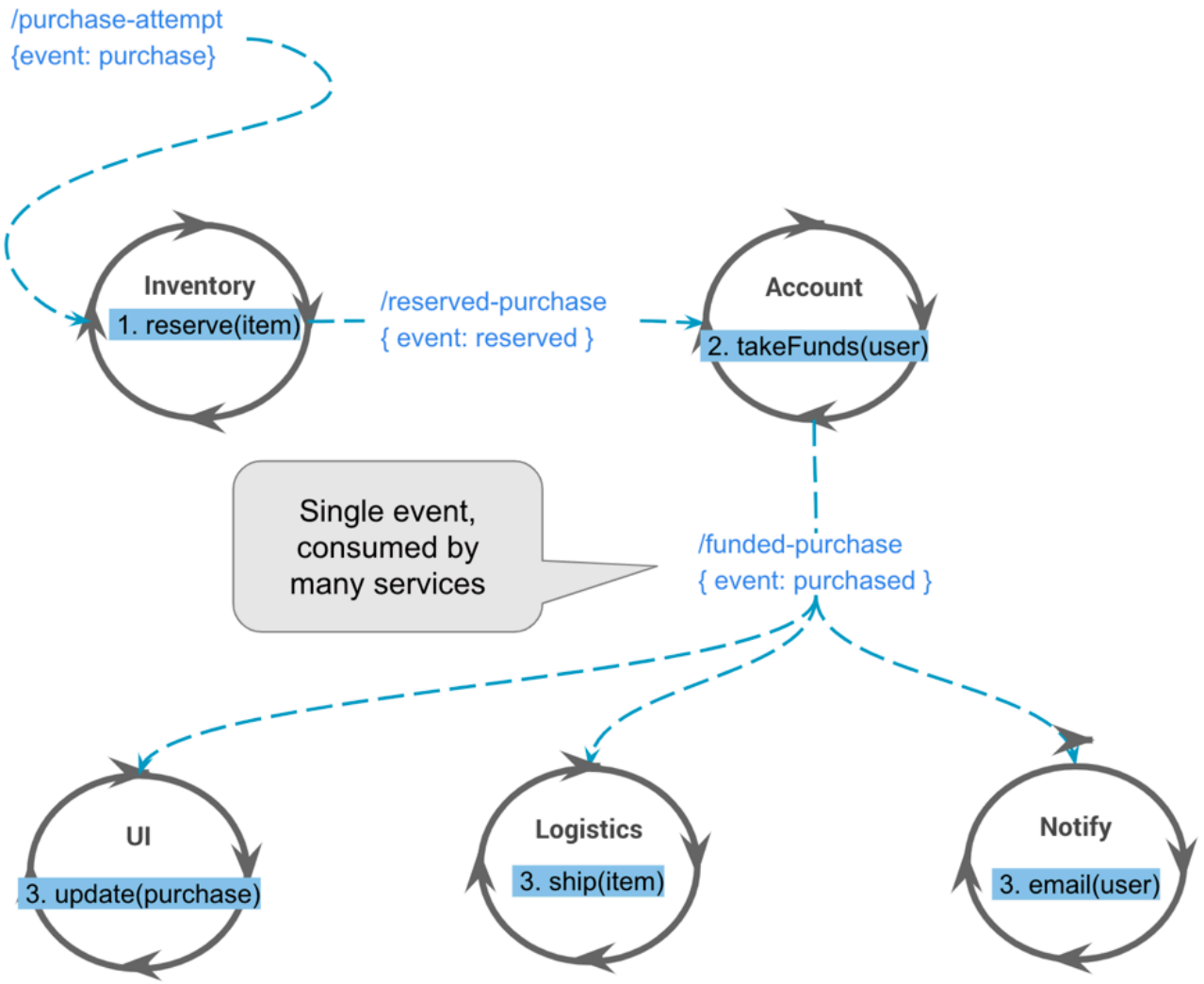




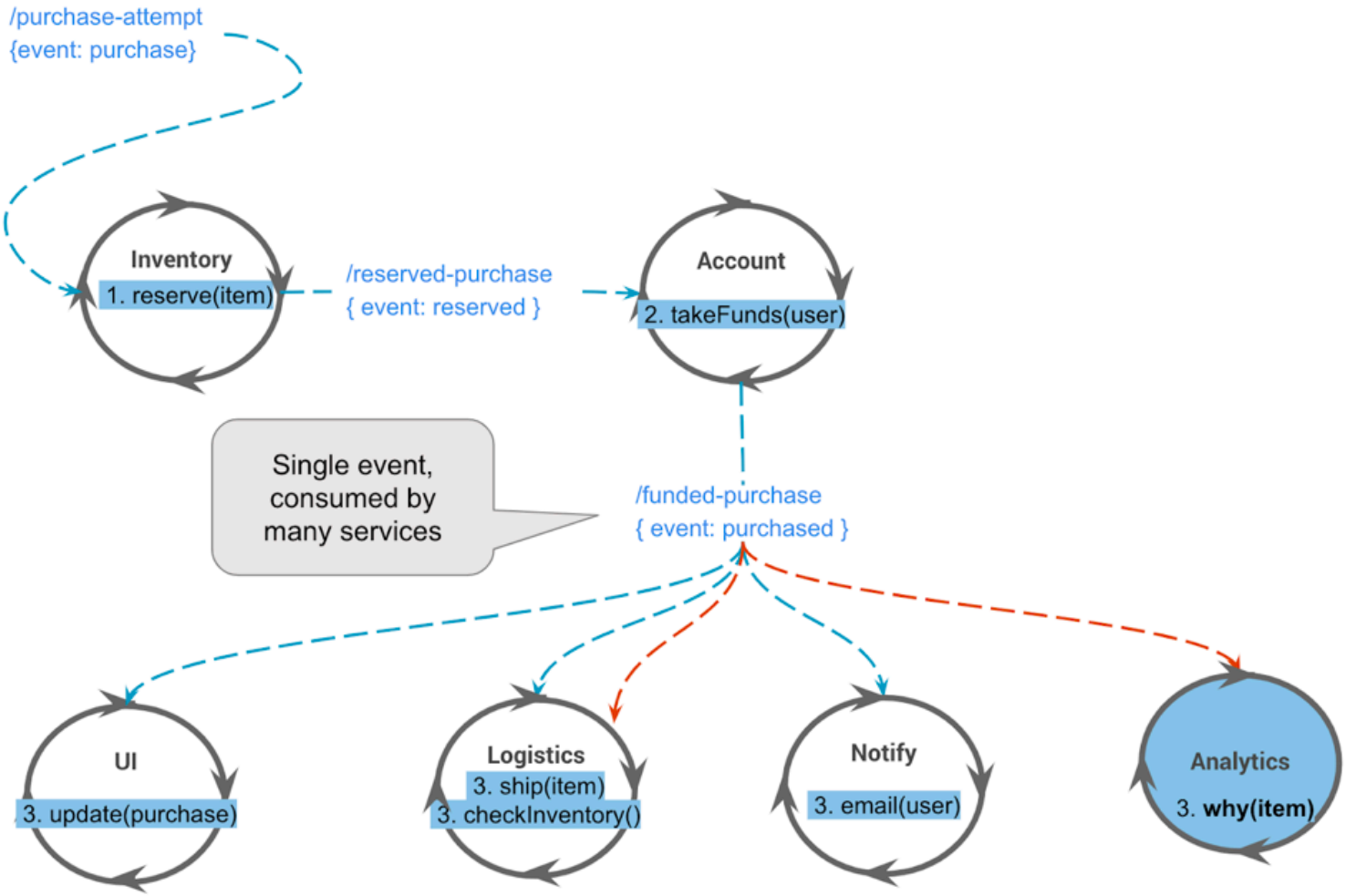
Basic Event Oriente



Event-Command Oriented



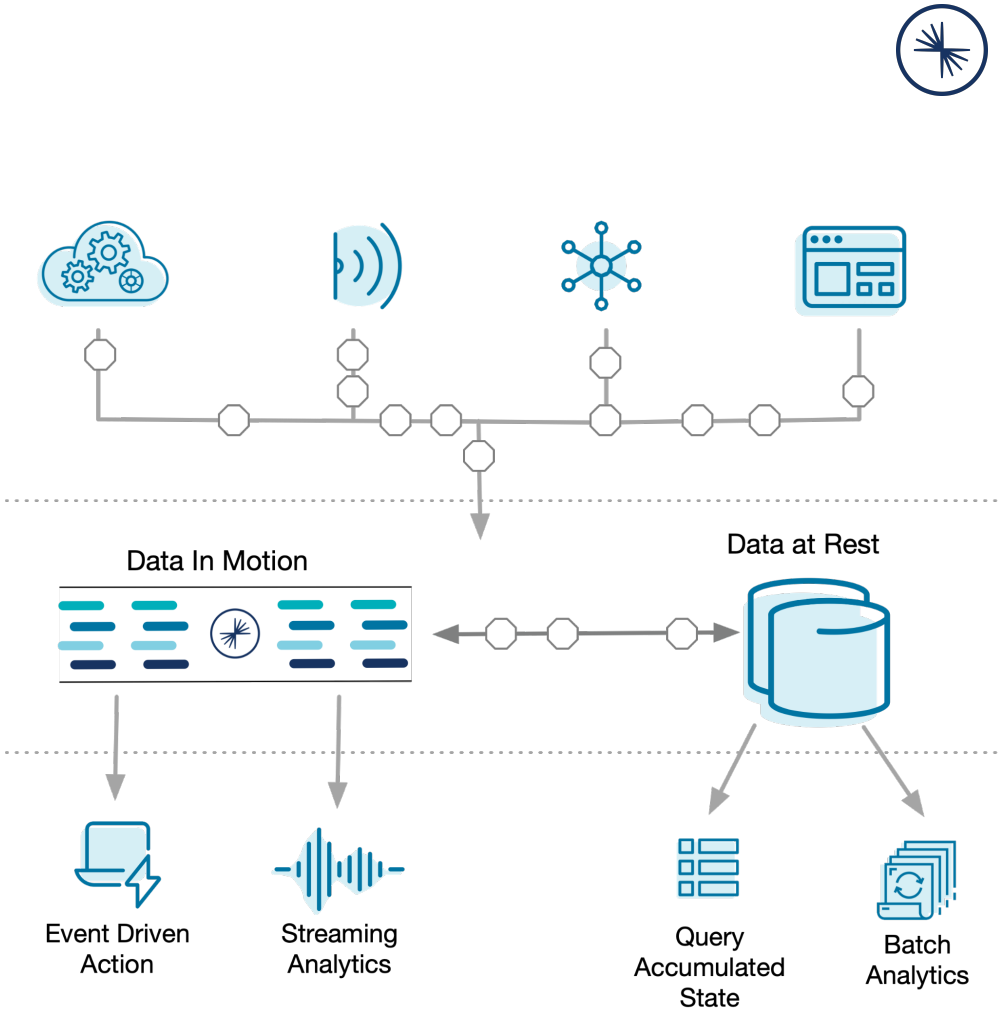
Event Oriented - Asynchronous

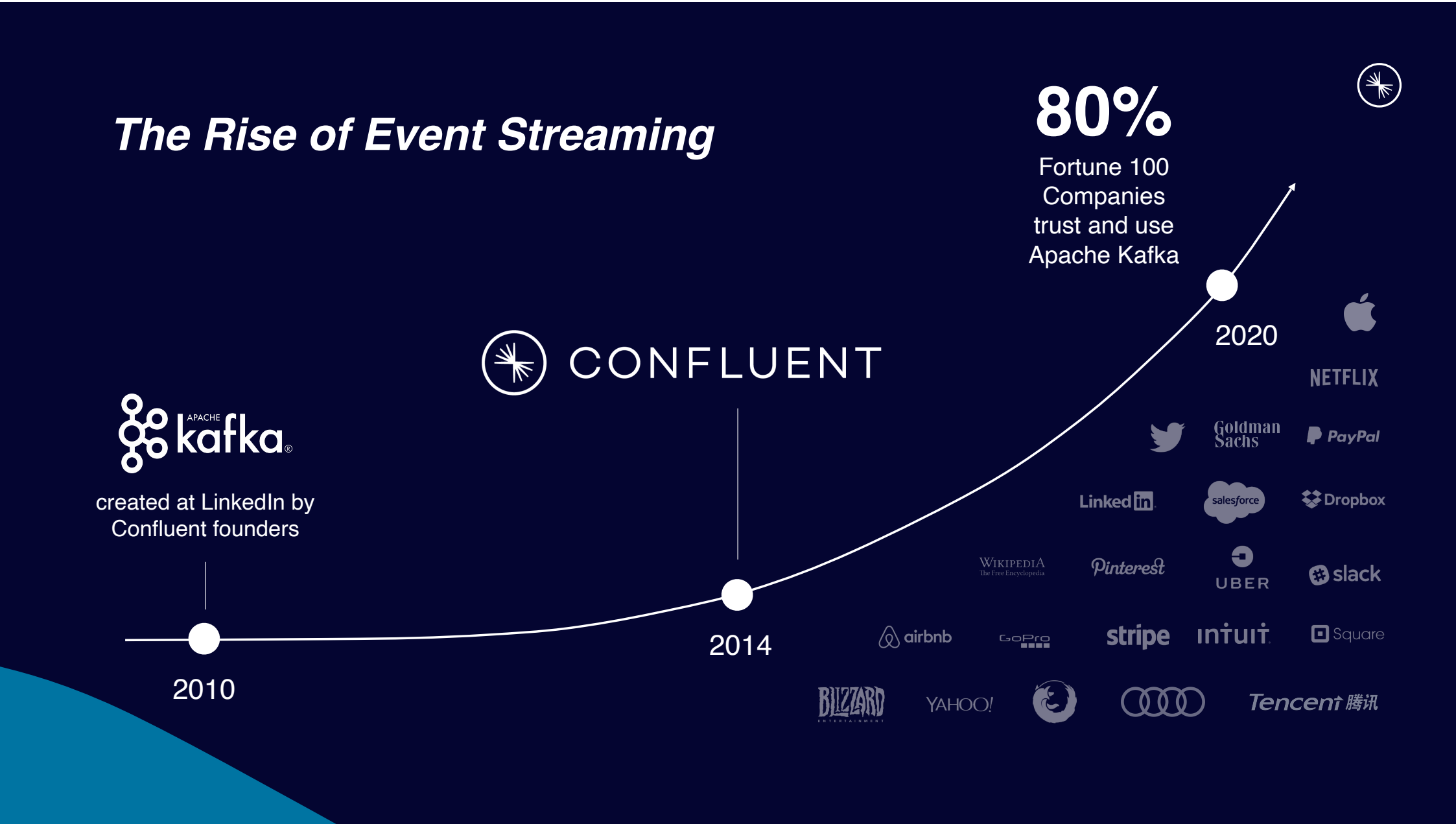


Event Oriented Decoupled

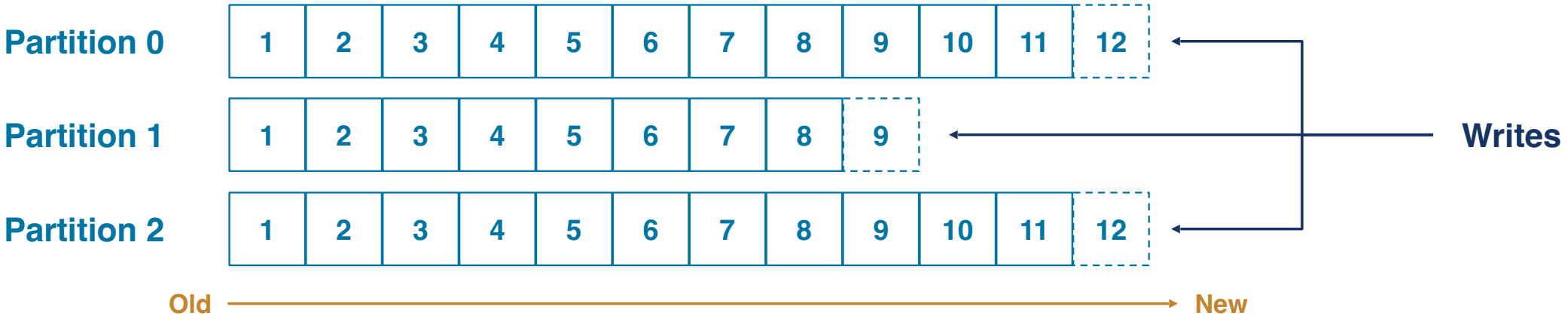
Modern Architecture

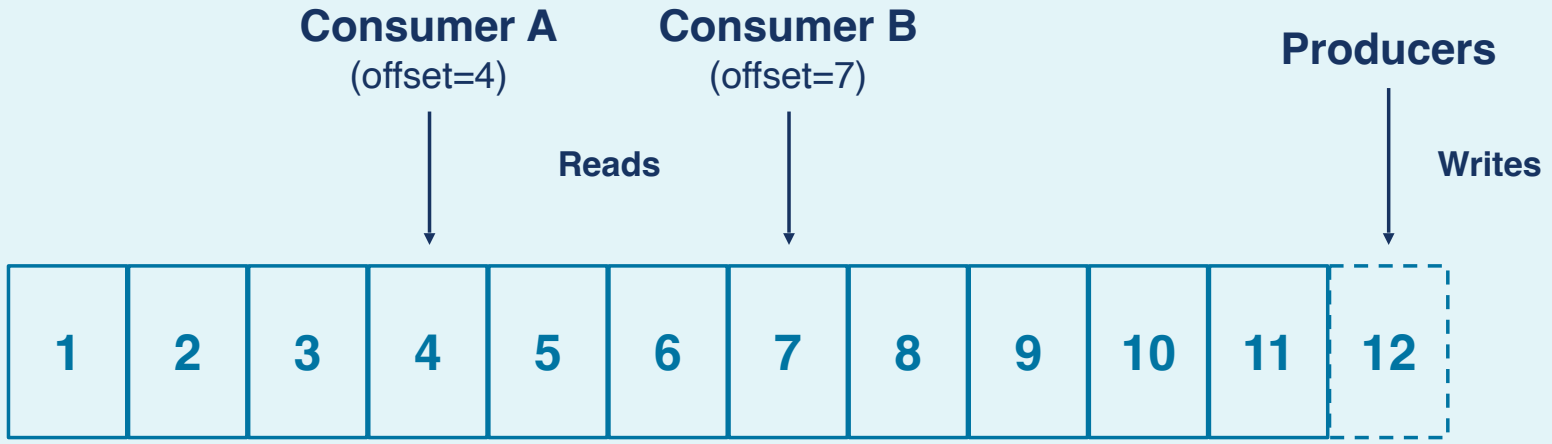
- Event Production
  - Everything is a source
  - Native Events
  - DB Transactions
- Events Flow Between
- Data in Motion
- Data at Rest




The Rise of Event Streaming
80%
Fortune 100 Companies trust and use Apache Kafka
APACHE kafka
created at LinkedIn by Confluent founders
CONFLUENT
2010
2014
2020
NETFLIX
Goldman Sachs
PayPal
LinkedIn
salesforce
Dropbox
WIKIPEDIA
The Free Encyclopedia
Pinterest
UBER
slack
airbnb
GoPro
stripe
INTUIT
Square
BLIZZARD ENTERTAINMENT
YAHOO!
Tencent 腾讯

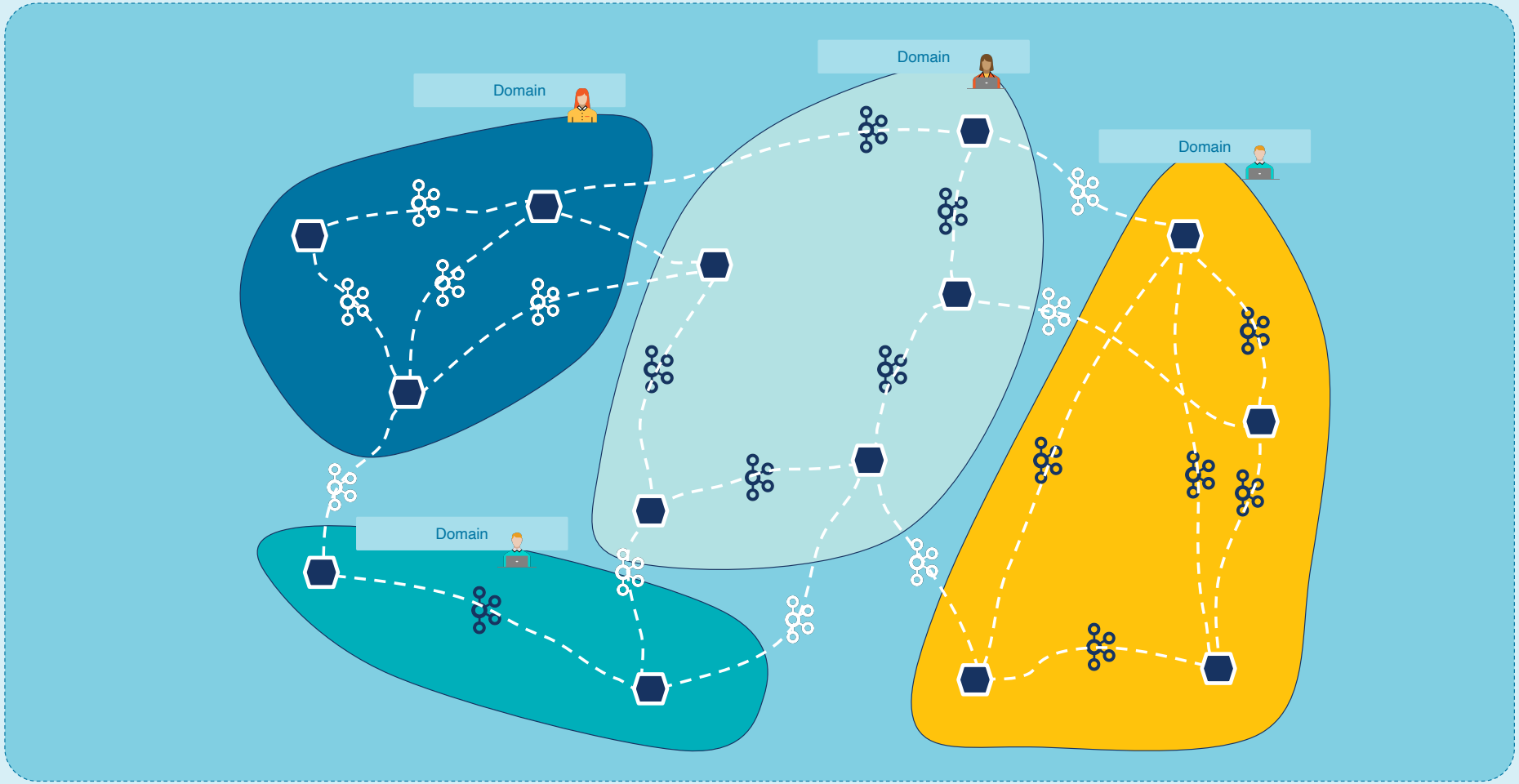
Anatomy of a Kafka Topic



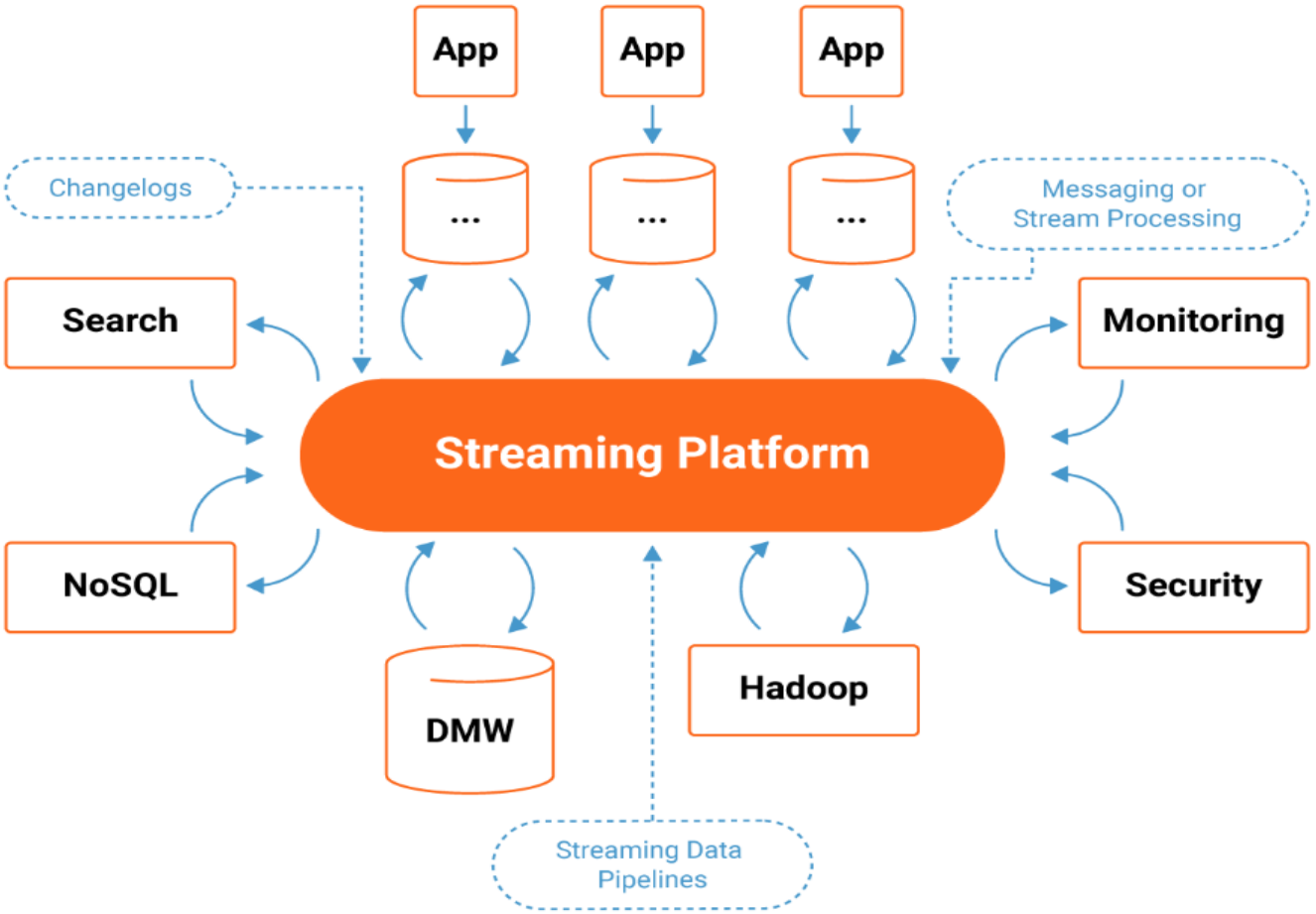


So what are the principles of a good EDA using Data in Motion?

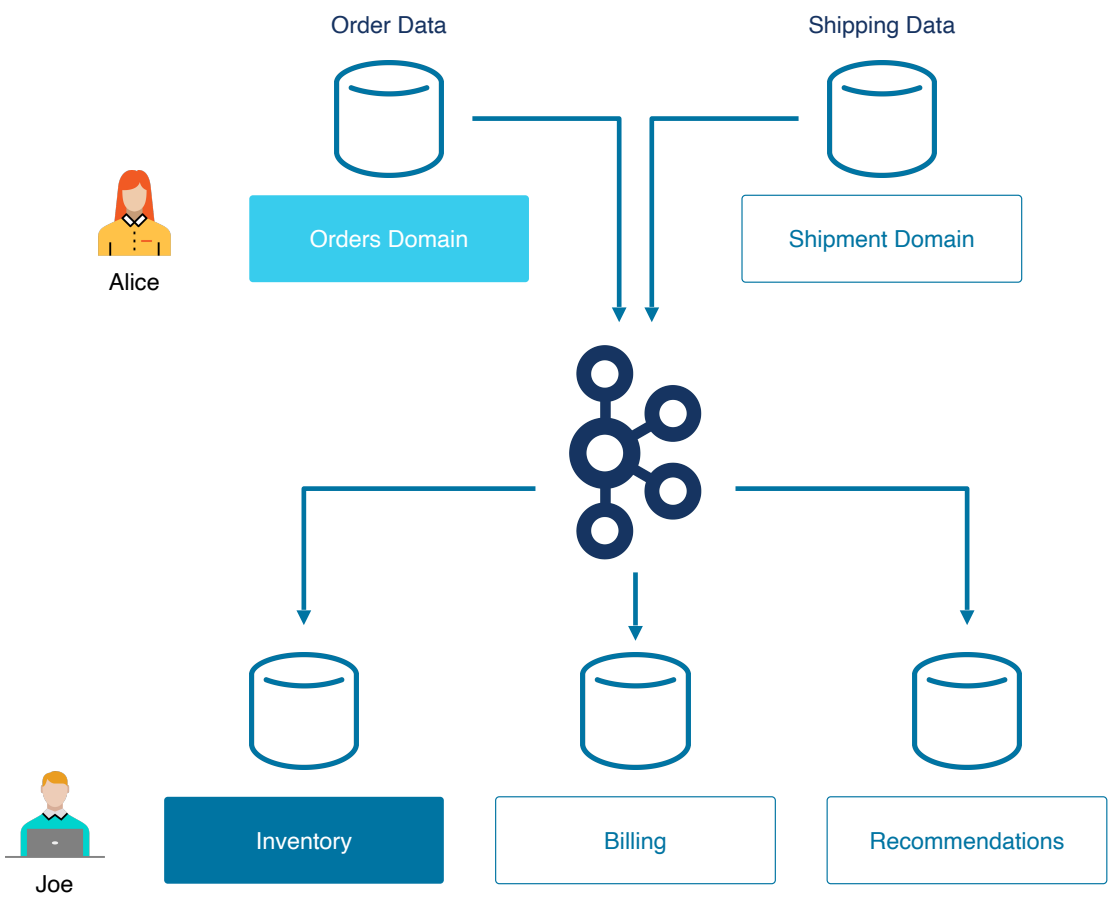
1.Treat Event Data as First Class Citizen



Data-Centric Approach to Computing

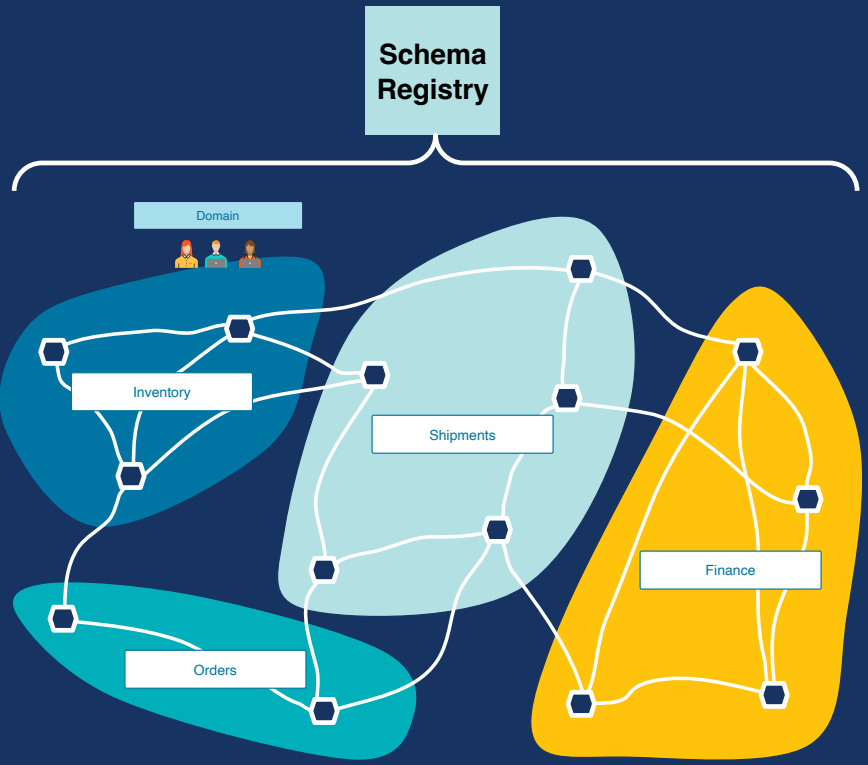


Practical example



1. Joe in Inventory has a problem with Order data.
2. Inventory items are going negative, because of bad Order data.
3. He could fix the data up locally in the Inventory domain, and get on with his job.
4. Or, better, he contacts Alice in Orders and get it fixed at the source. This is more reliable as Joe doesn't fully understand the Orders process.
5. Ergo, Alice needs be an responsible & responsive "Data Owner", so everyone benefits from the fix to Joe's problem.

2. Make Schemas the Contract for Event Streams



Confluent Schema Registry:

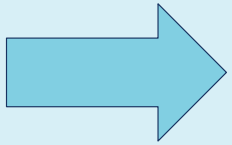
- Supports:
  - Avro
  - Protobuf
  - JSON Schema
- Can be used with Event Streams and other technologies

Schema Evolution - Compatible Changes

- Schemas will evolve!
- Think about backwards and forwards compatibility
- Compatible changes include (for example):
  - Adding new fields
  - Using default values (improves compatibility rules)
  - Removing fields with defaults

Schema Evolution

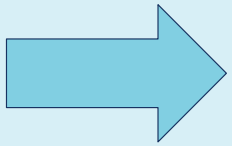
```
message User {
    int32 id = 1;
    string first_name = 2;
    string last_name = 3;
}
```



Adding fields

```
message User {
    int32 id = 1;
    string first_name = 2;
    string last_name = 3;
    string country_code = 4;
}
```

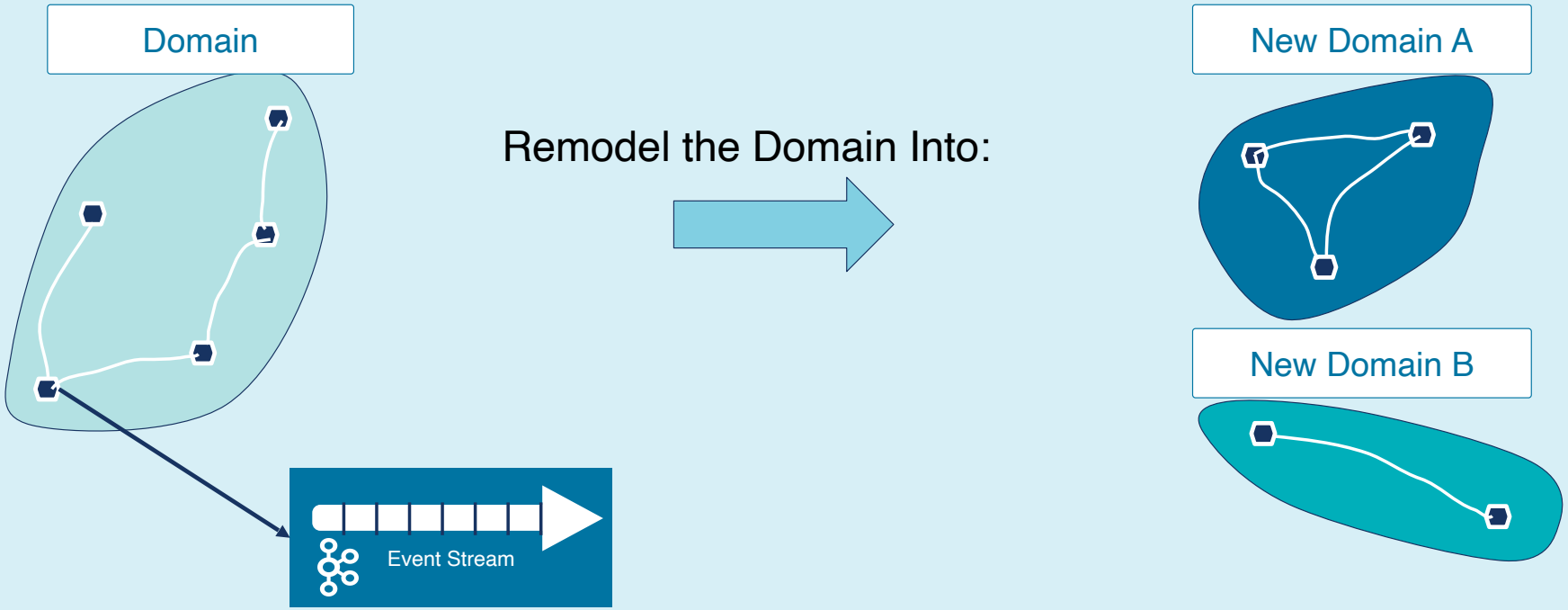
```
message User {
    int32 id = 1;
    string first_name = 2;
    string last_name = 3;
}
```



Removing fields
(eg: PII)

```
message User {
    int32 id = 1;
}
```

3. Plan for Breaking Changes and Failures



Collaborative Effort - Coordinate and Plan

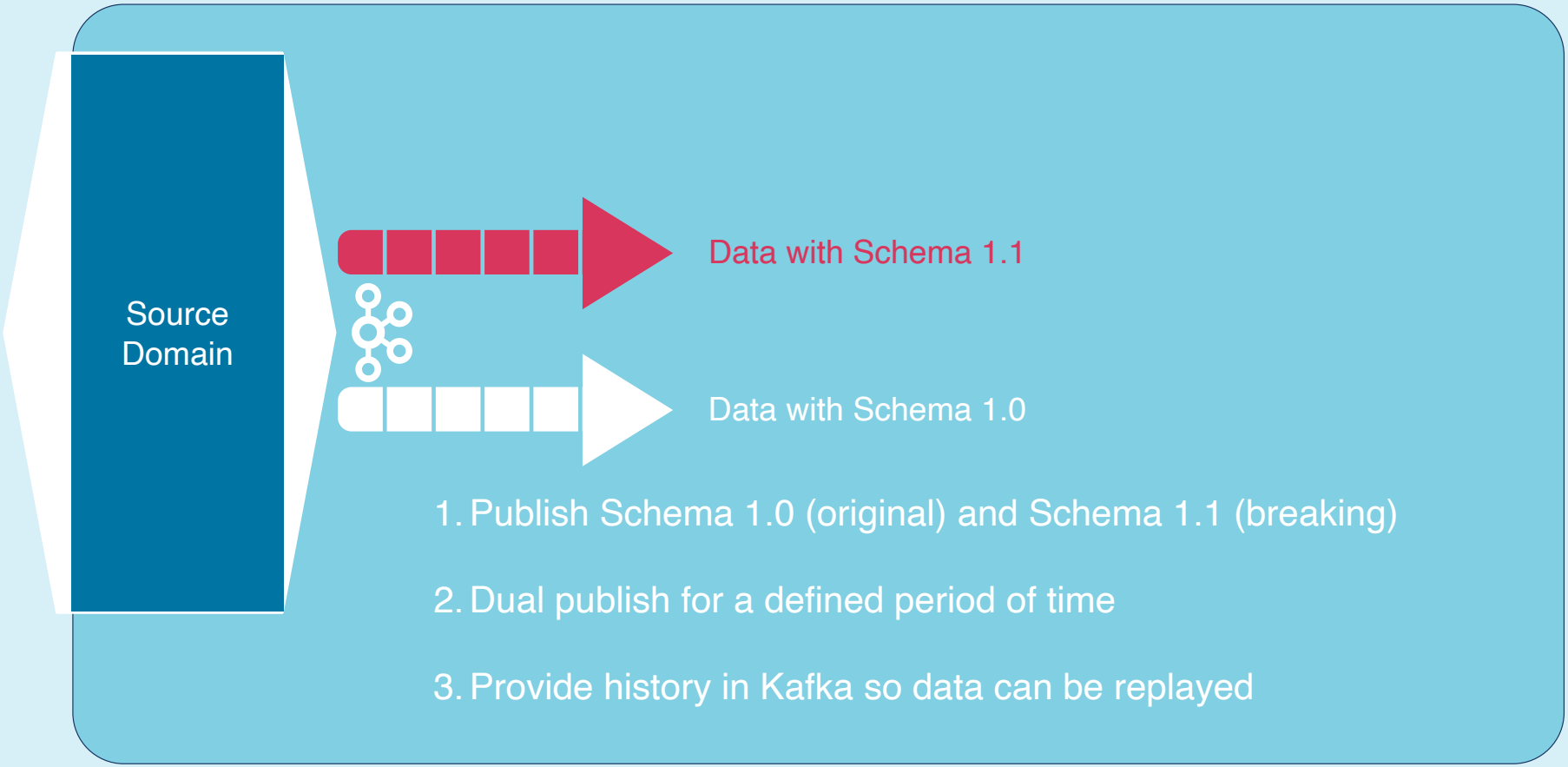
Know your event-stream consumers

- Who is affected?
- What are their needs?

Communicate with them

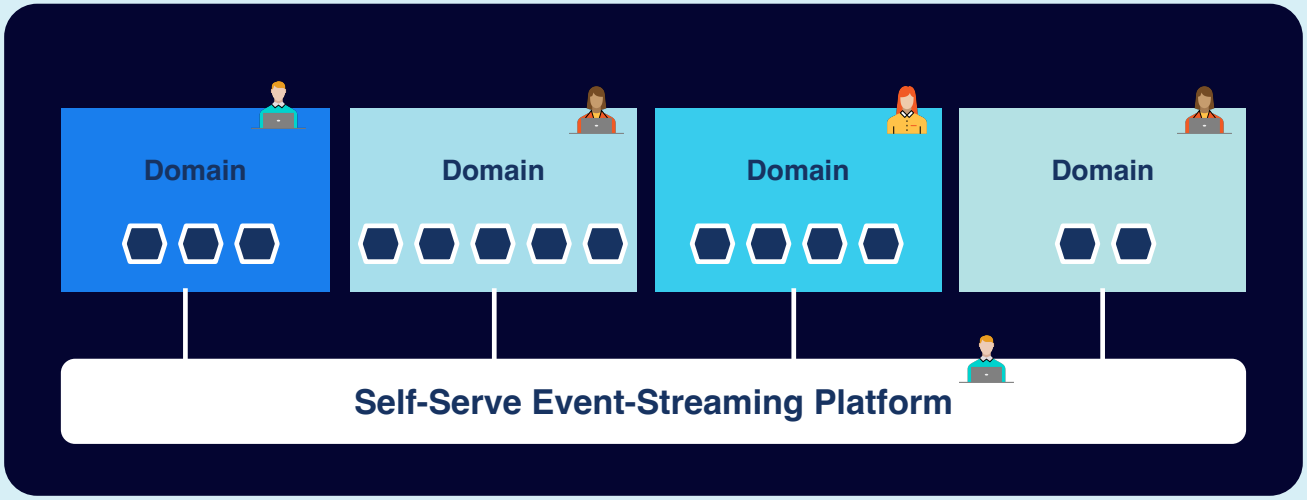
- Let them know ahead of time of upcoming changes
- Discuss how to fulfil their needs, such as:
  - Migrating old data to new format
  - Minimizing downtime

Incompatible changes require a rolling upgrade window



4. Create and Empower a Governing Body

- Standards of Interoperability, Policies, and Support
- The minimum requirements for data in the org? (SLAs on changes, uptime, data, etc.)
- How schema changes are proposed, approved and communicated



Decisions can Include:

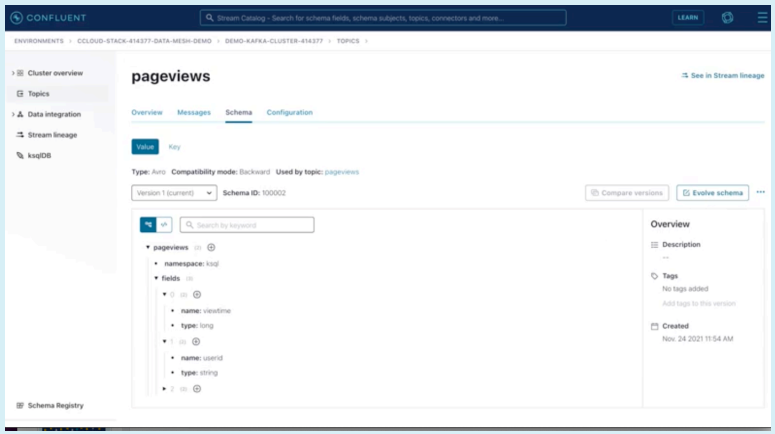
- What schema format do we use?
- How do we monitor our applications?
- How do we impose Access Controls?
- How do we adhere to our data policies?
- Etc: What else are the main data concerns of your business?

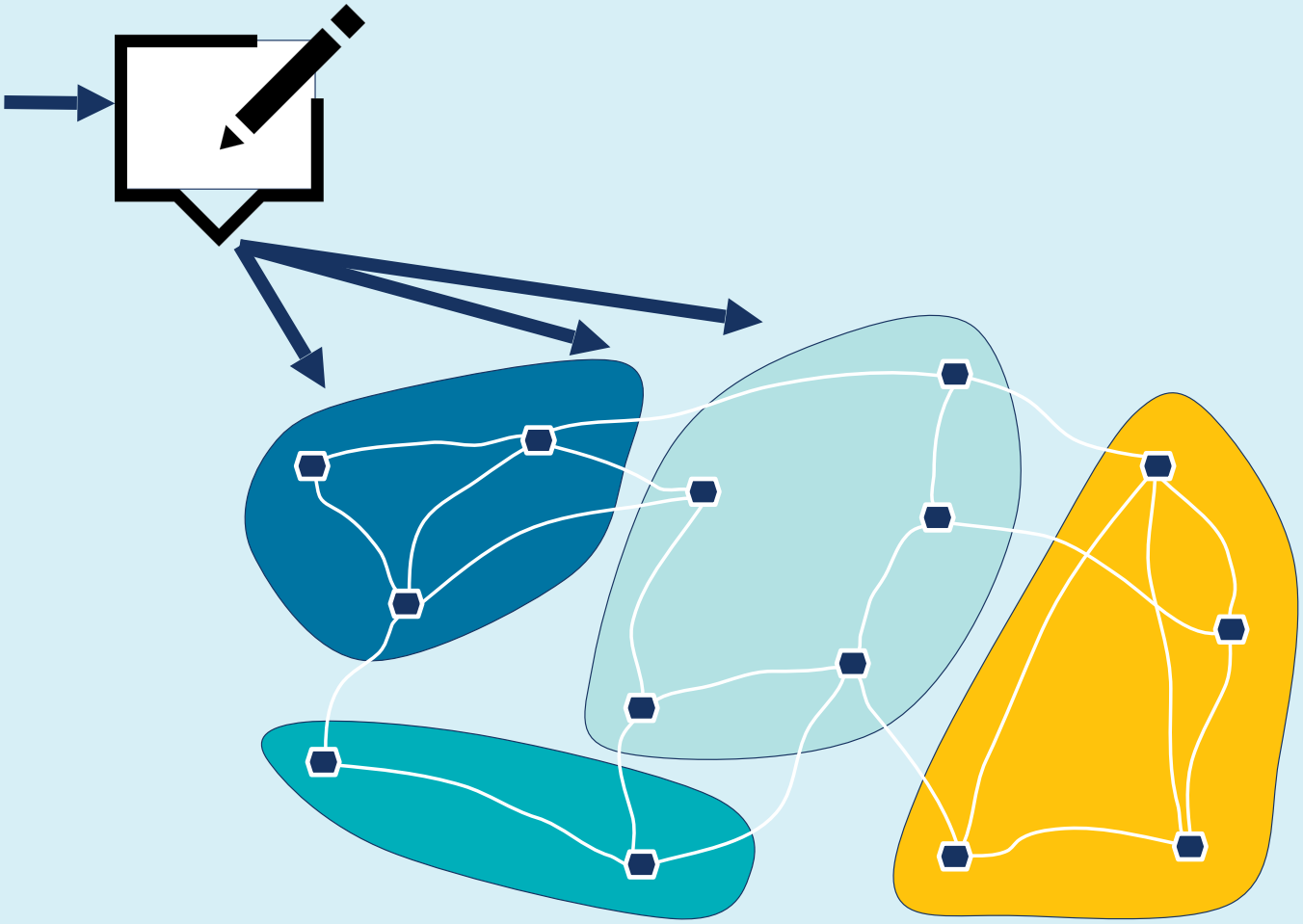
5. Provide Self-Service Functionality with First Class Support

- **Global standards and streamlined support**
  - Eg: New applications must be in containers
  - Eg: New event-driven applications must use Confluent Cloud

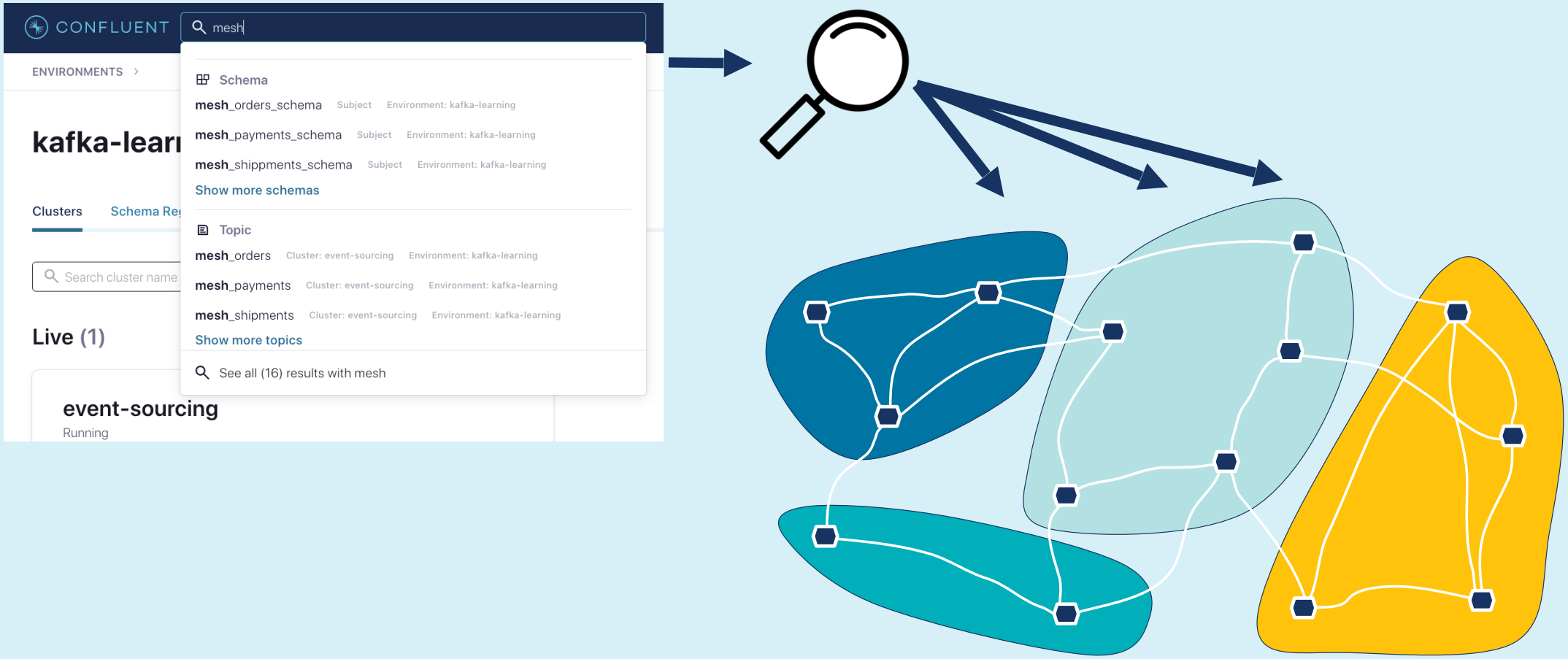
- **“Paved Roads” - Make it Easy to Use**
  - Make it easy to build applications using event streams, in the languages and frameworks of your choice.
  - Proof-of-concept applications and pilot projects can lead the way
  - Focus on making it easy to get things done: reduce toil and overhead.

6. Make Data Easy to Discover and Use

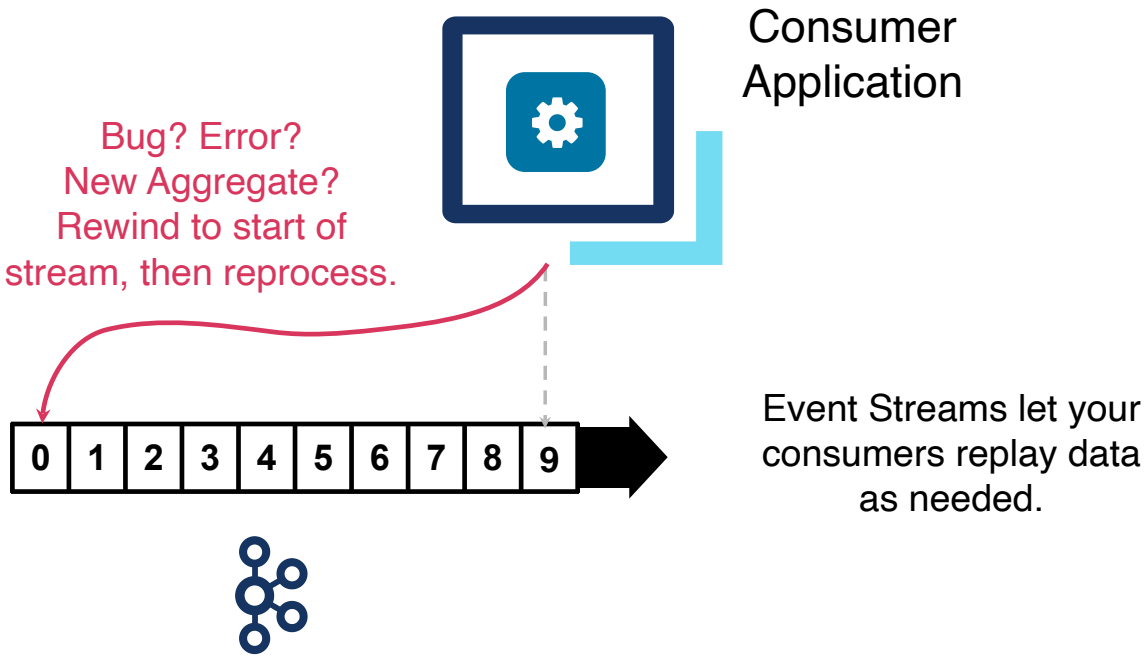




Search through Events, Streams, and Tags



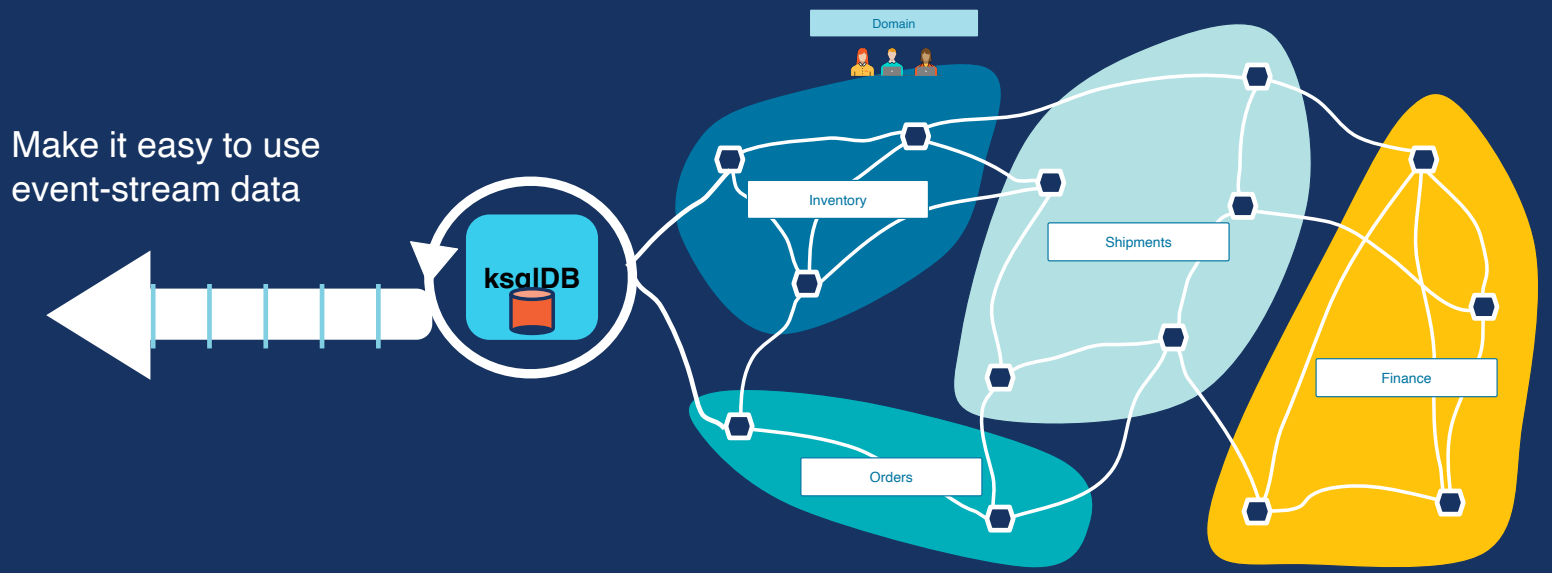
Rely on Event Streams for Historical Data



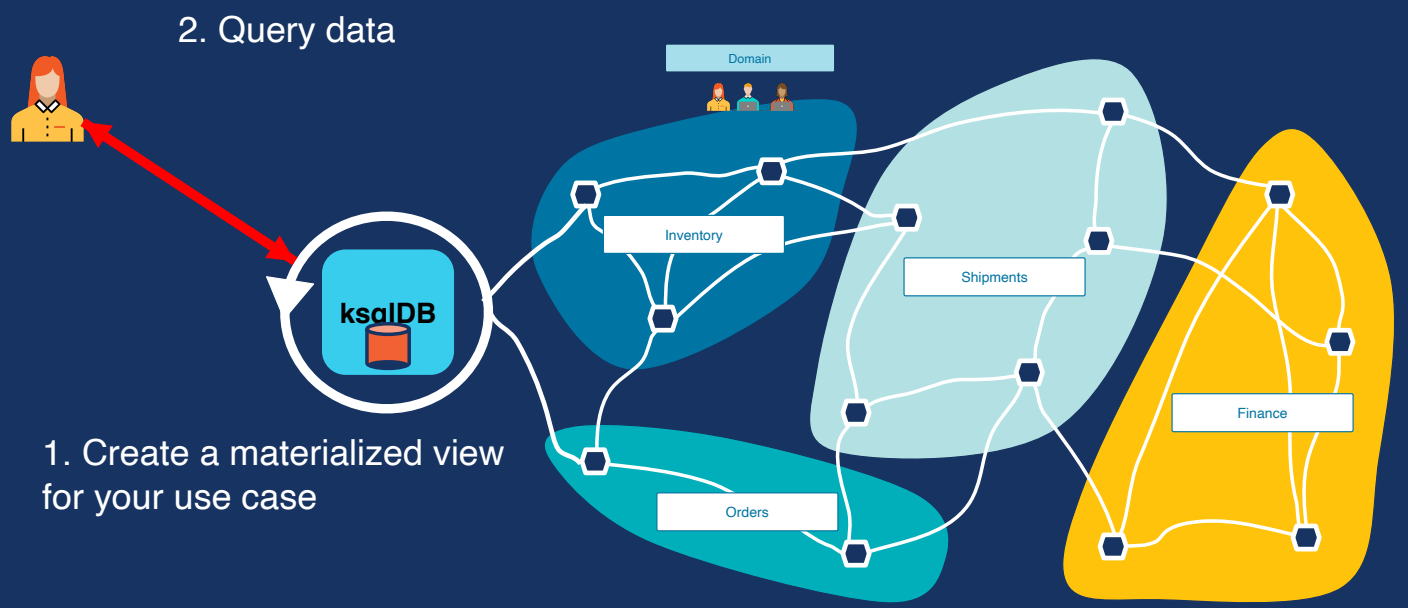
Store Event Data For As Long As Necessary

- Store all the data you need, for as long as you need it.
- Cheap disk! Compaction!
- Confluent Cloud's Infinite Storage
  - Fully transparent tiered-storage

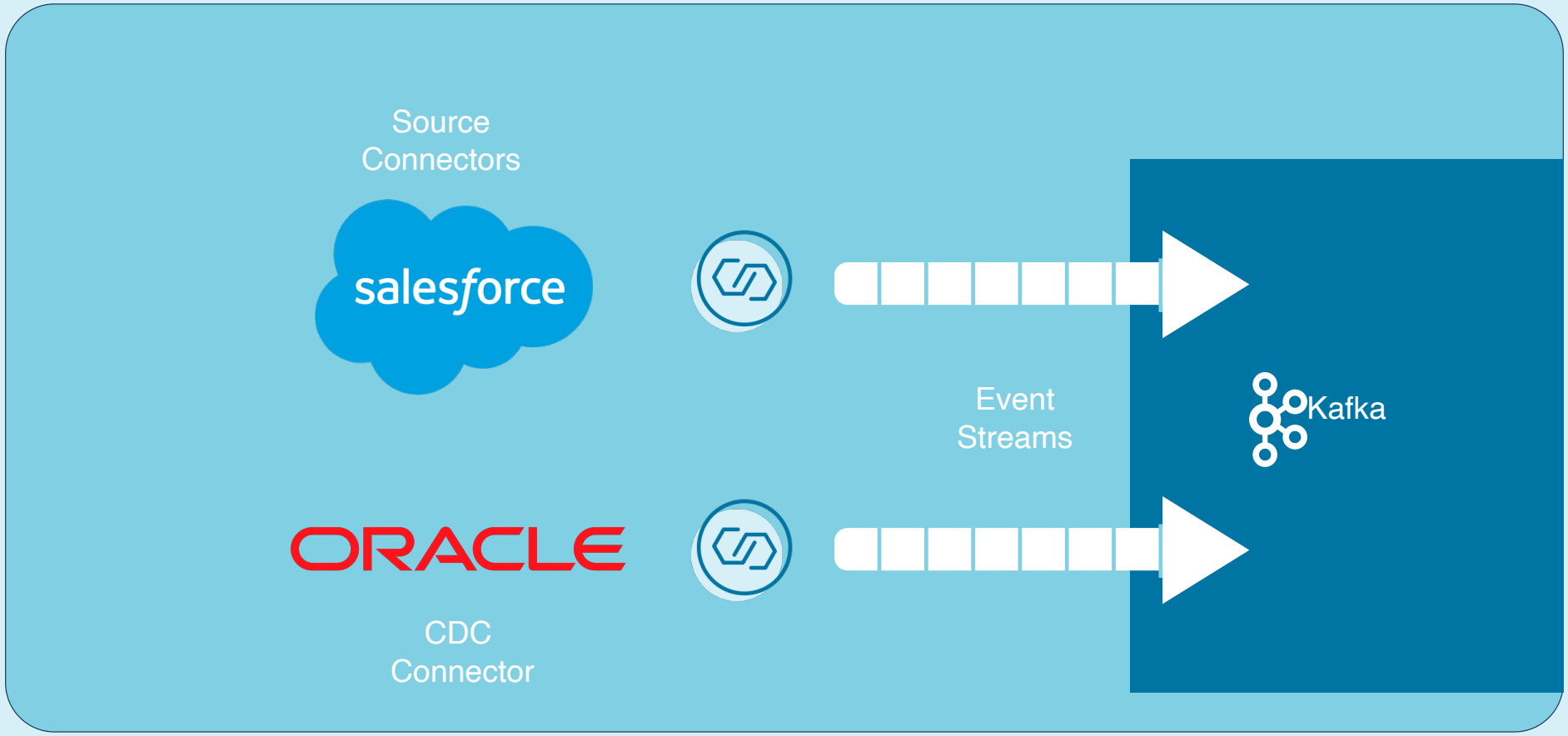
Self-Service ksqlDB:
Process and Transform Event Streams



Self-Service ksqlDB:
Query data however you need it



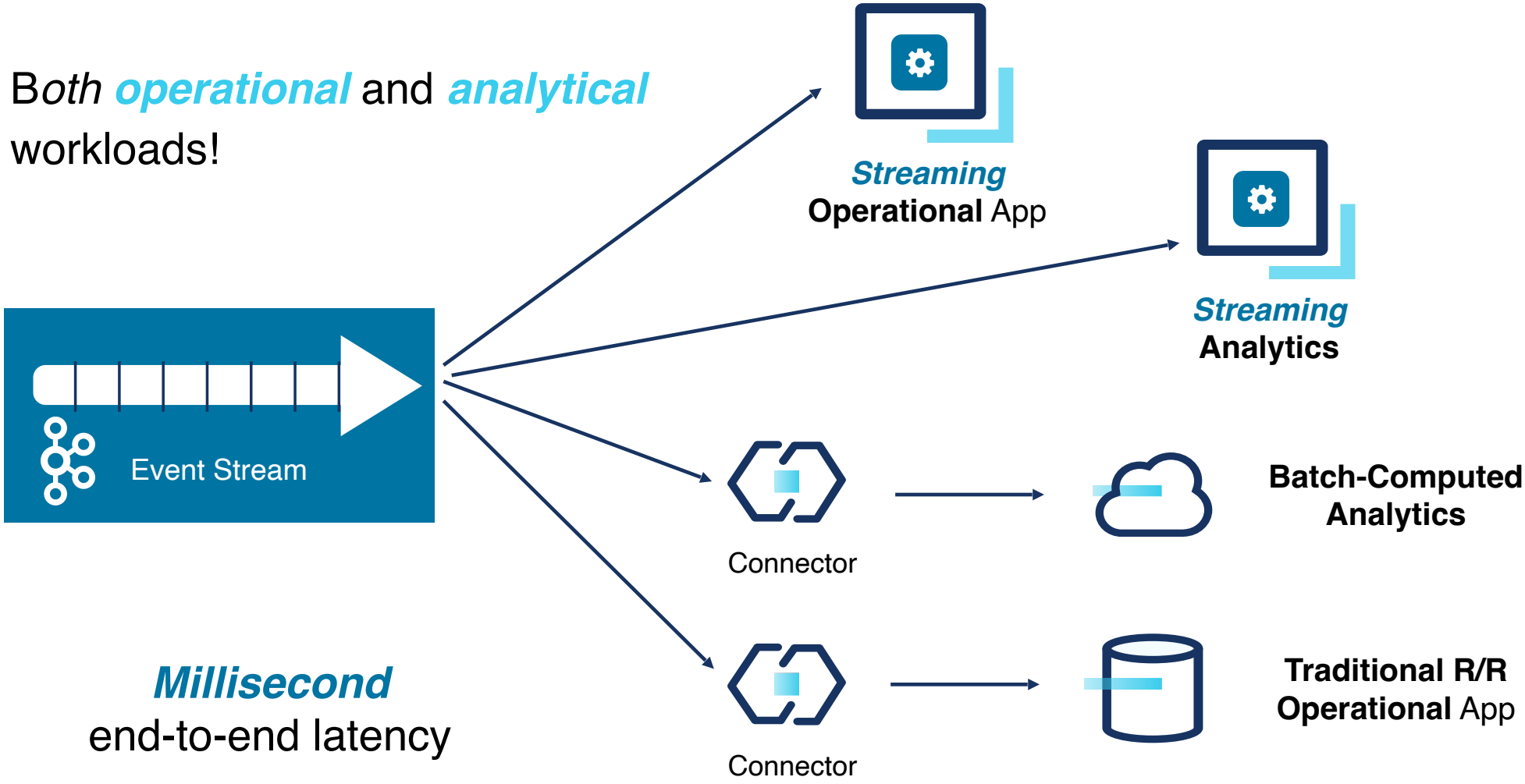
7. Rely on Connectors to Bridge Streams and Batch



https://www.confluent.io/hub/

Event Streams Power Realtime & Batch Processing

B*oth* ***operational*** and ***analytical*** workloads!



Millisecond
end-to-end latency

Principles for Building Event-Driven Architectures

1. Treat Data as a First-Class Citizen

So that it can be relied upon and reused by all users. Dedicated owners for all key datasets in the organization.

2. Make Schemas the Contract for Event Streams

Schema evolution. Confluent Schema Registry.

3. Plan for Breaking Changes and Failures

Some changes require remodeling existing streams. Plan for breakages.

4. Create and Empower a Governing Body

Standard. How to validate, publish, change, find, and use event streams.

5. Provide Self-Service Functionality

Compute and processing frameworks. Connectors. Create, build, and manage your own the data you publish for others.

6. Make Data Easy to Discover and Use

Search for schemas and event data. Preview event streams. Request access to data. Data lineage views.

7. Rely on Connectors to Bridge Streams and Batch

Connectors for sourcing and sinking data. Data is essential for getting EDAs started.

Thanks!
Check out *developer.confluent.io*

Alex Stuart

ajfstuart