Modeling Streaming Data for Processing with Apache Beam

GETTING STARTED WITH STREAM PROCESSING

Janani Ravi
CO-FOUNDER, LOONYCORN
www.loonycorn.com

Overview

Batch data and bounded datasets

Streaming data for unbounded datasets and real-time processing

Micro-batch processing and continuous processing

Lambda and Kappa architectures

Challenges in real-time stream processing

Prerequisites and Course Outline

Prerequisites

No prior experience of working with Streaming Data required

Experience programming in Java

Apache Maven for dependency management

Course Outline

Getting Started with Stream Processing

Introducing Apache Beam for Stream Processing

Perform Windowing Operations

Batch Processing and Stream Processing

Analysis of Deliveries

E-commerce site

How are they distributed across the country?

Are there routes that can be clubbed together?

How do courier companies compare?

Analysis of Deliveries

Generate periodic reports to improve delivery metrics

Analysis of Deliveries

Collect

Source and destination of packages

Courier company details

Analyze

Run jobs on different slices

Courier, metro areas, rural areas, warehouses

Extract

Trends in the form of visuals

Actionable insights

Analysis of Deliveries

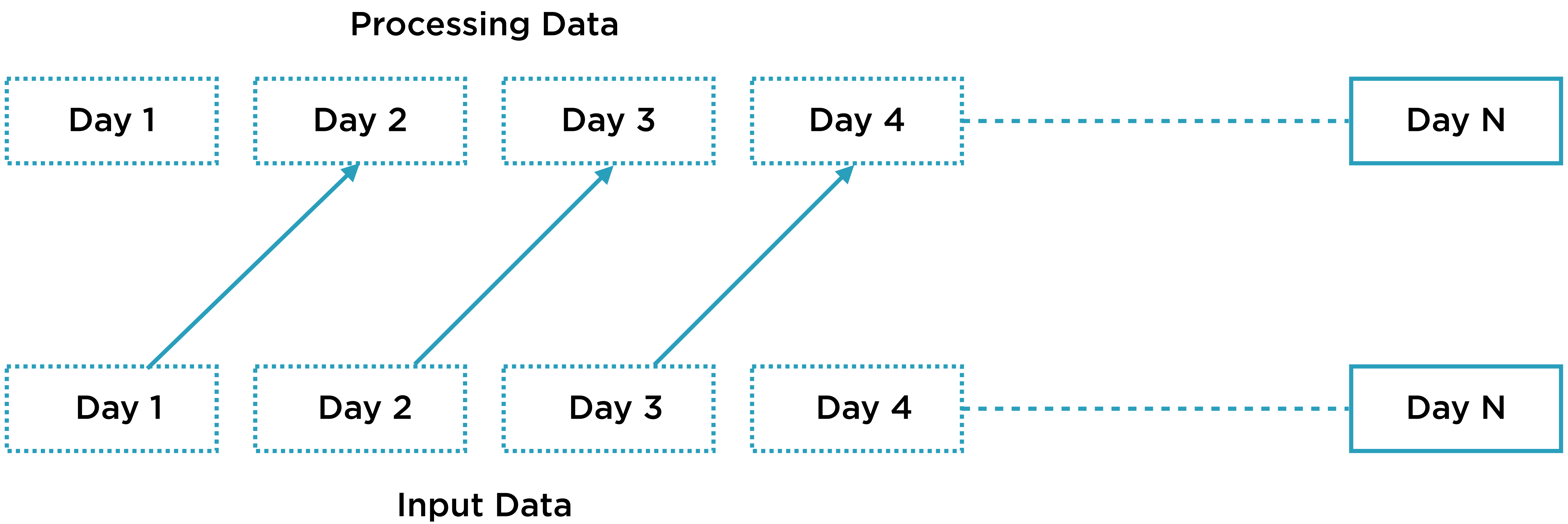
Bounded datasets: Finite unchanging datasets to analyze

- week, month, year

Batch processing: Runs for a specific time, completes, releases resources

- minutes, hours, days

Batch Processing



Batch Processing

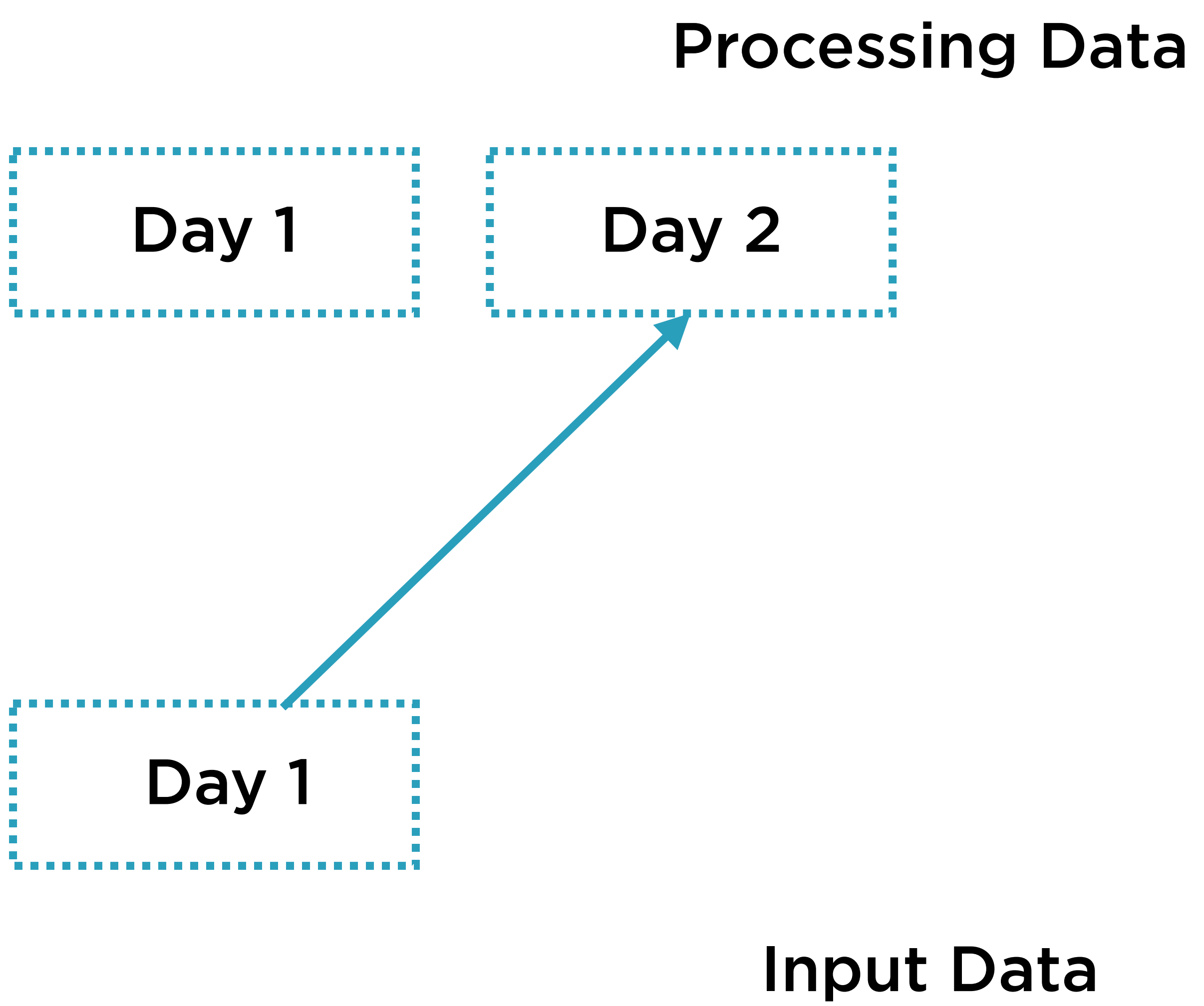
Processing Data

Day 1

Day 1

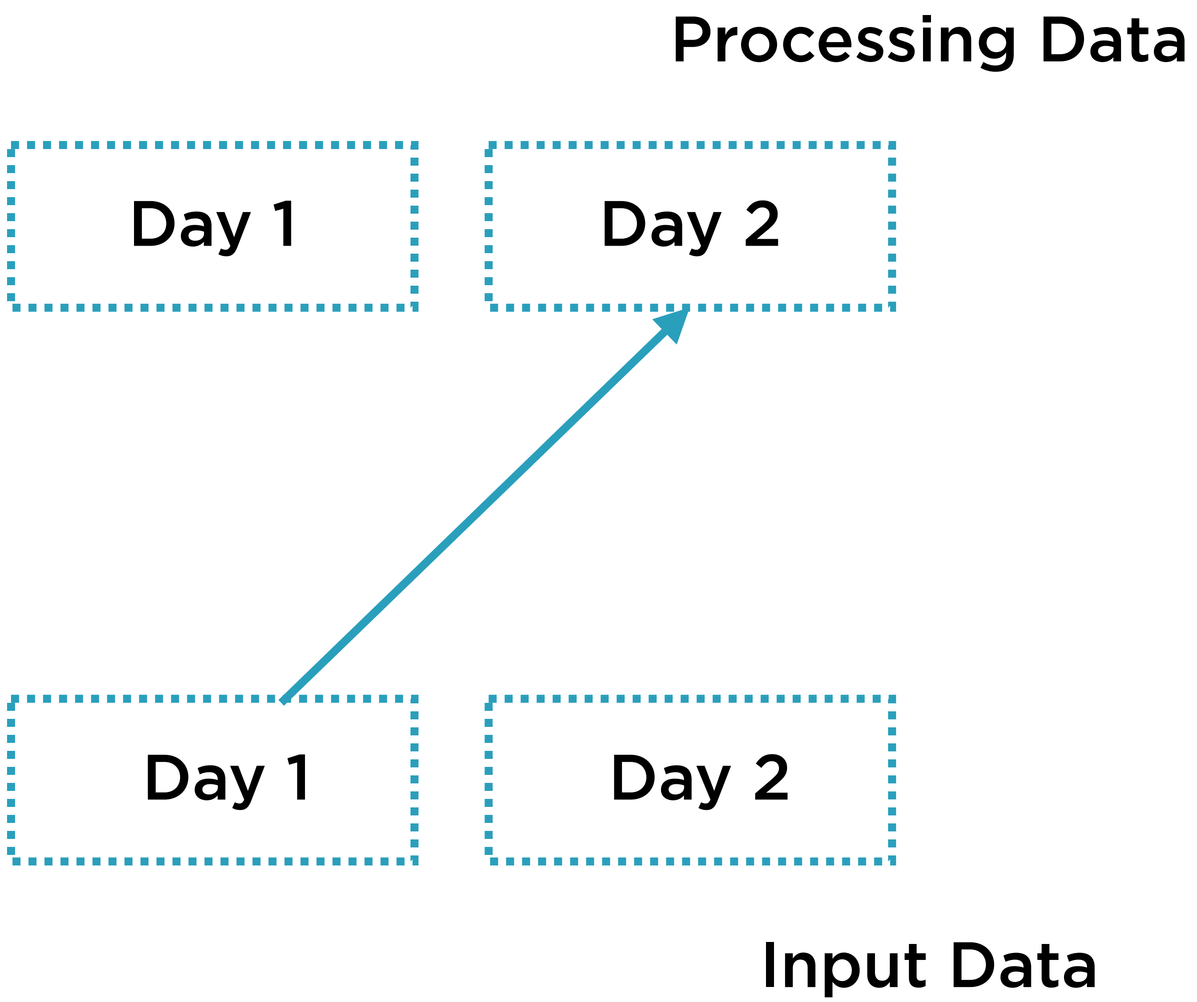
Input Data

Batch Processing



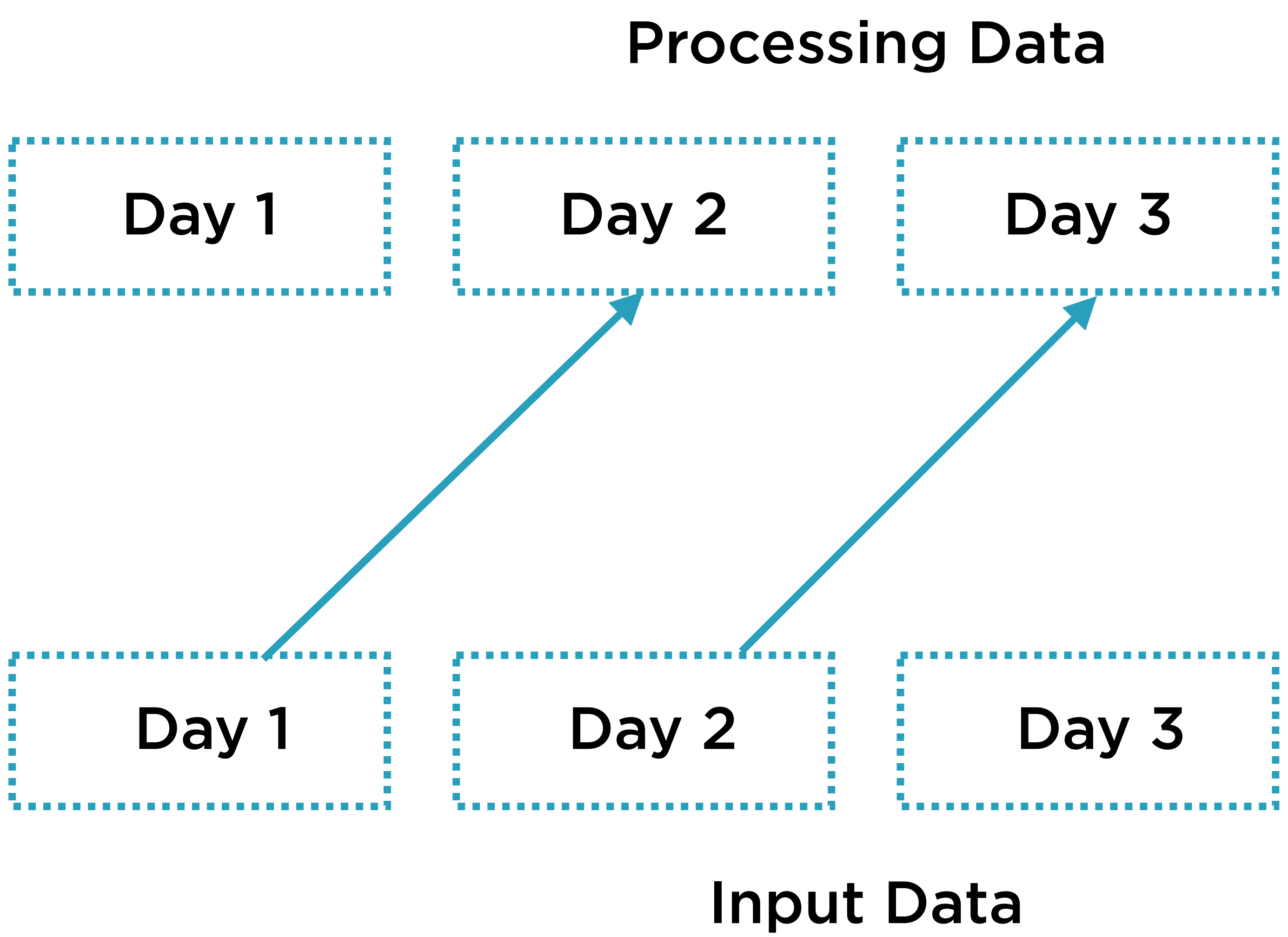
Stored data processed over a period of time

Batch Processing



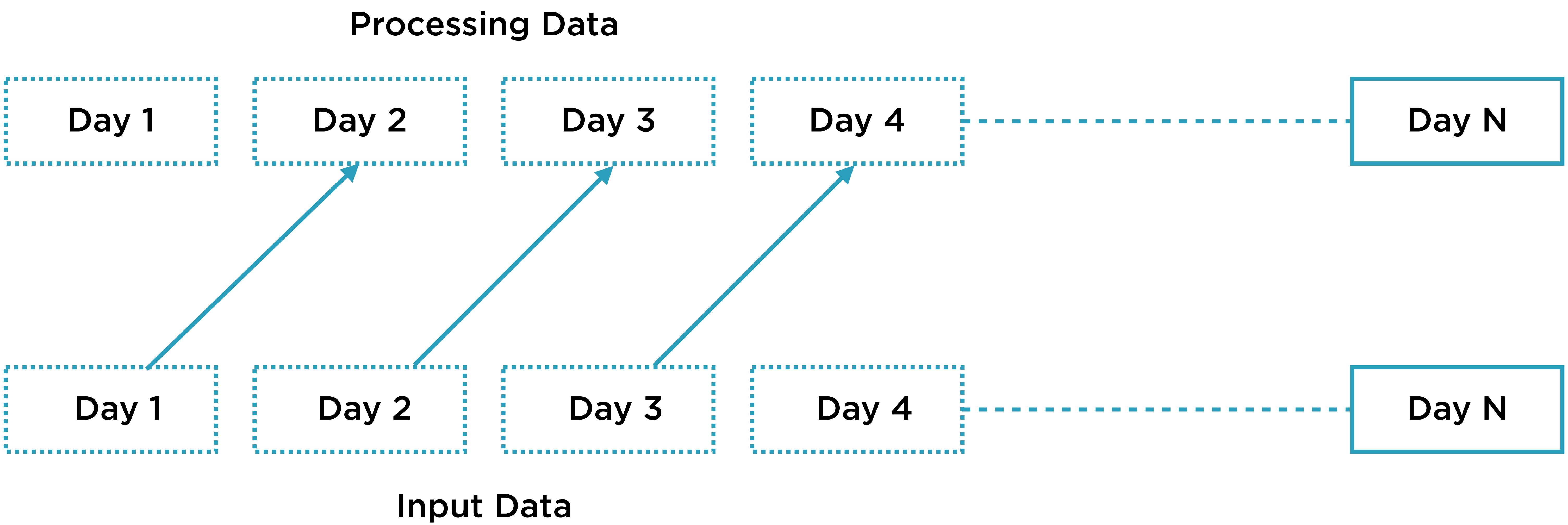
Stored data processed over a period of time

Batch Processing



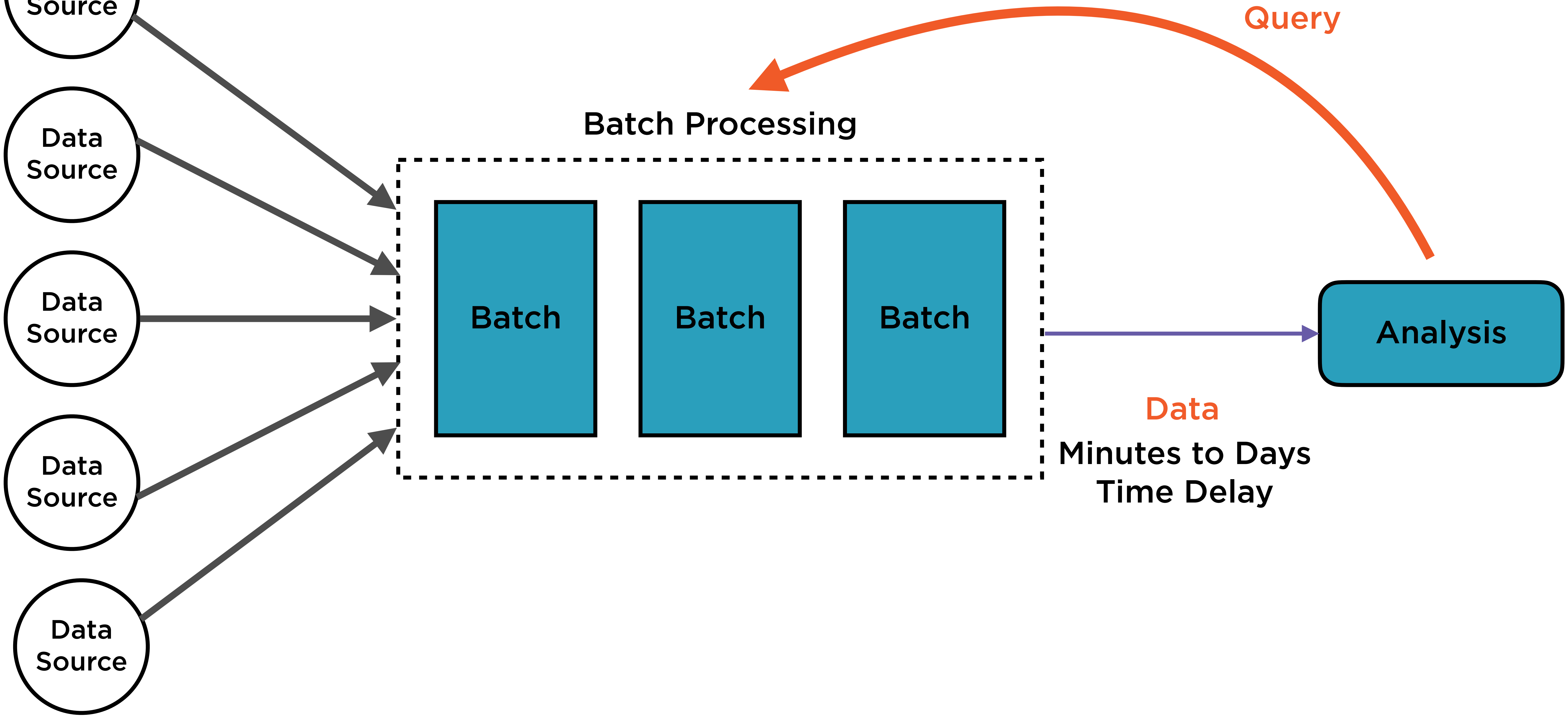
Stored data processed over a period of time

Batch Processing



Stored data processed over a period of time

Batch Processing



Tracking of Deliveries

E-commerce site

Real-time location of delivery agents

Real-time order status updates

Real-time inventory tracking

Tracking of Deliveries

Continuously monitor data to ensure deliveries are flowing smoothly

Tracking of Deliveries

Monitor

Constantly listen for updates

GPS coordinates, status information, inventory changes

Process

As entities flow in process them in micro-batches

Whole stream, predetermined window

Extract

Plot real-time graphs

Track on a map

Tracking of Deliveries

Unbounded datasets: **Infinite datasets which are added to continuously**

- streaming data

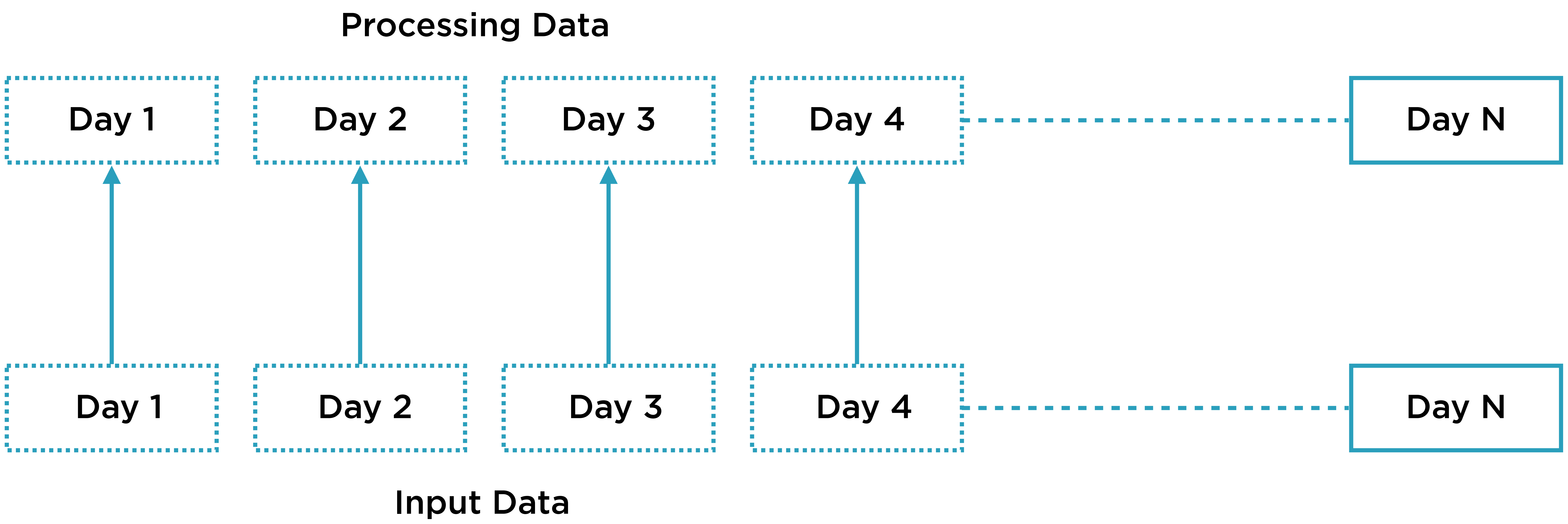
Continuous processing: **Runs constantly as long as data is received**

- stream processing

Bounded datasets are processed in **batches**

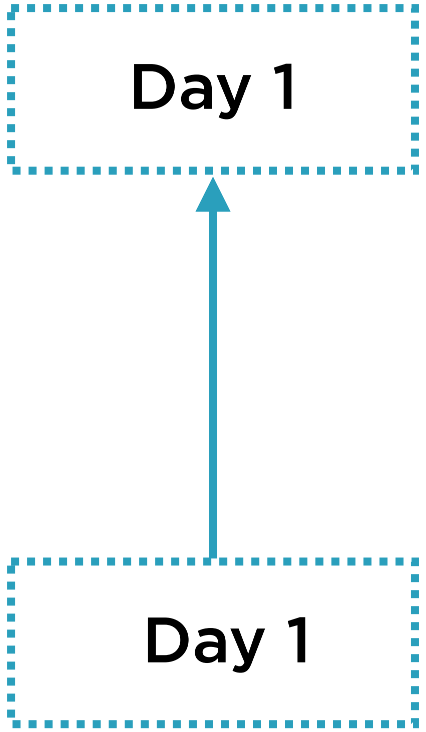
Unbounded datasets are processed as **streams**

Stream Processing



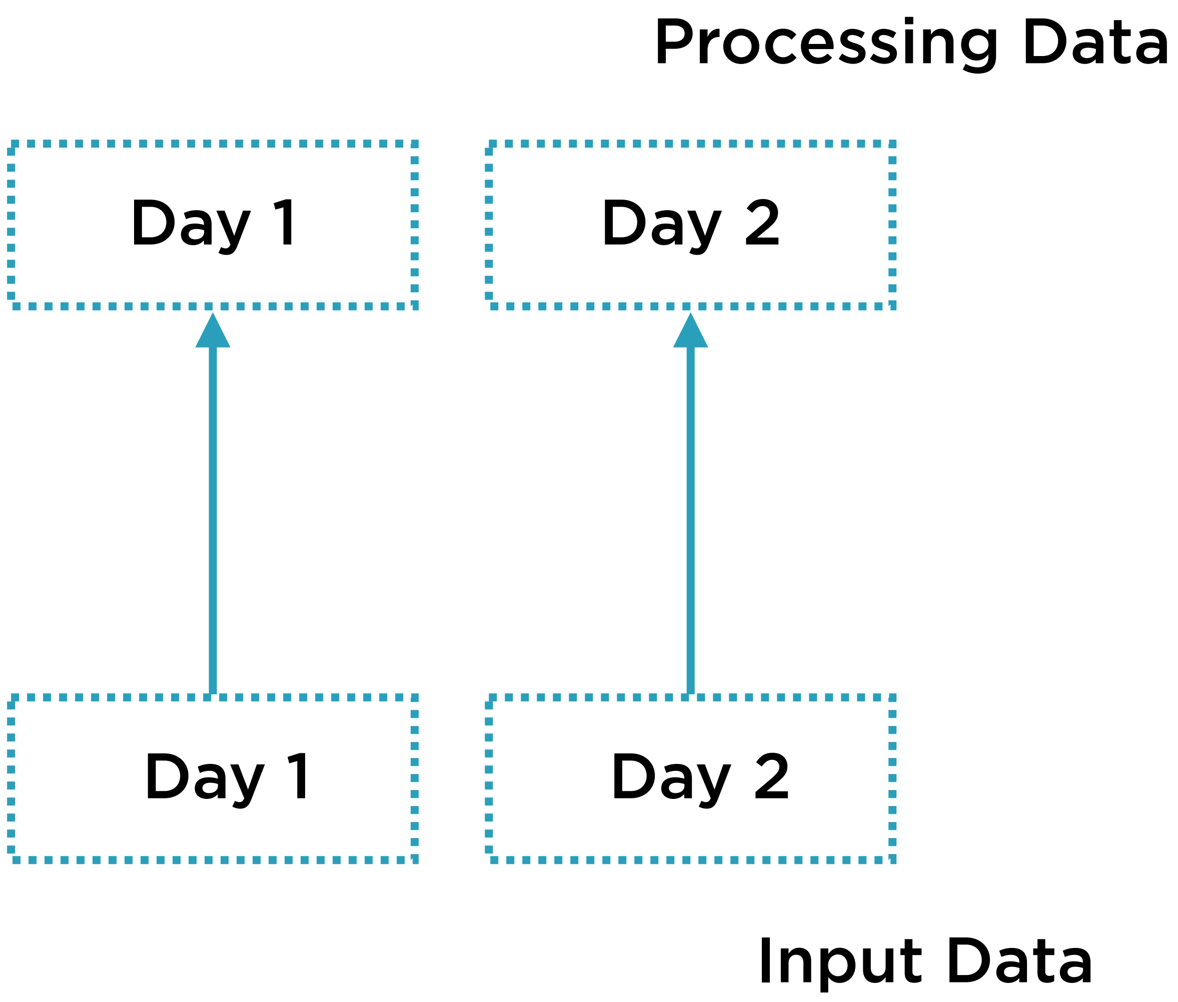
Stream Processing

Processing Data

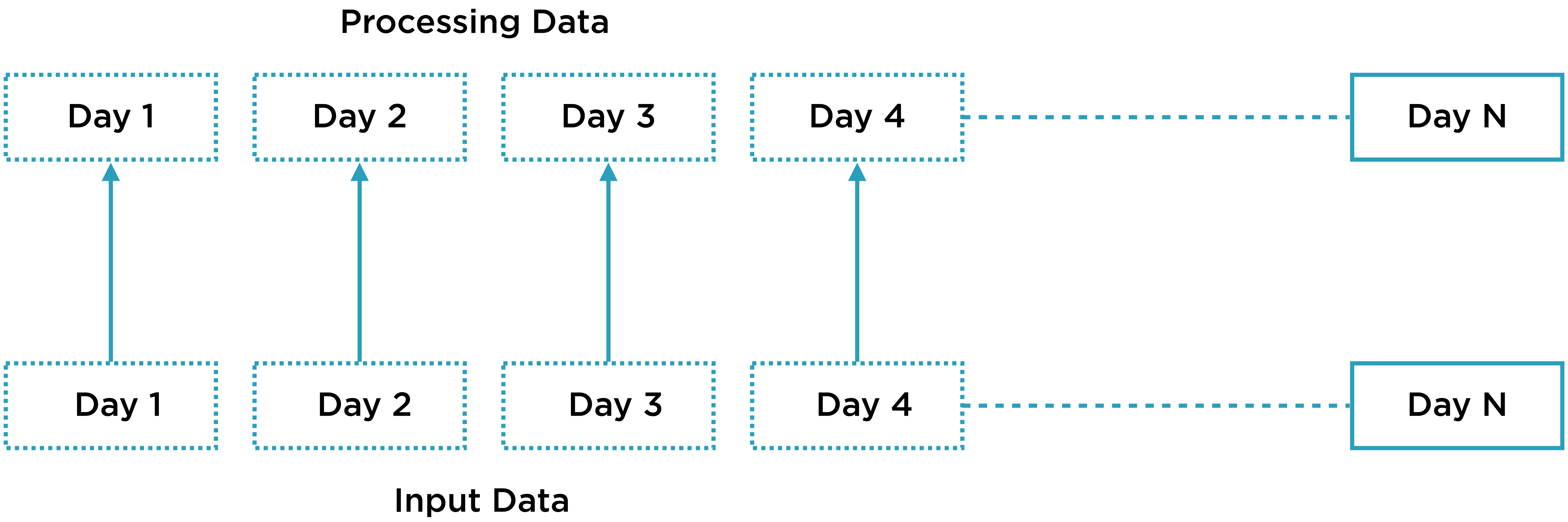


Input Data

Stream Processing

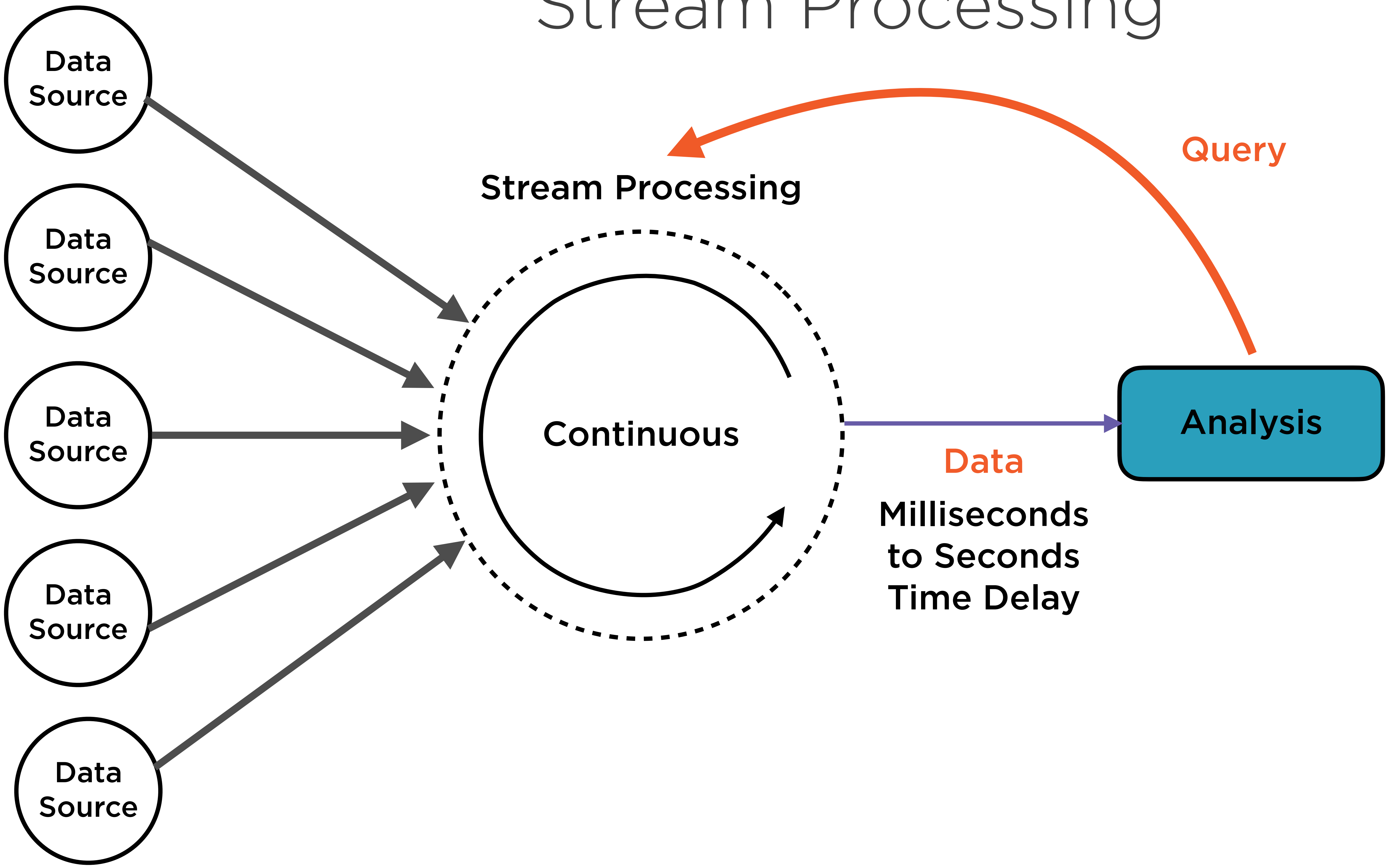


Stream Processing



Input data is processed with no time lag

Stream Processing



Batch vs. Stream Processing

Batch	Stream
Bounded, finite datasets	Unbounded, infinite datasets
Slow pipeline from data ingestion to analysis	Processing immediate, as data is received
Latency in minutes, hours considered acceptable	Latency usually must be in seconds, milliseconds
Periodic updates as jobs complete	Continuous updates as jobs run constantly

Batch vs. Stream Processing

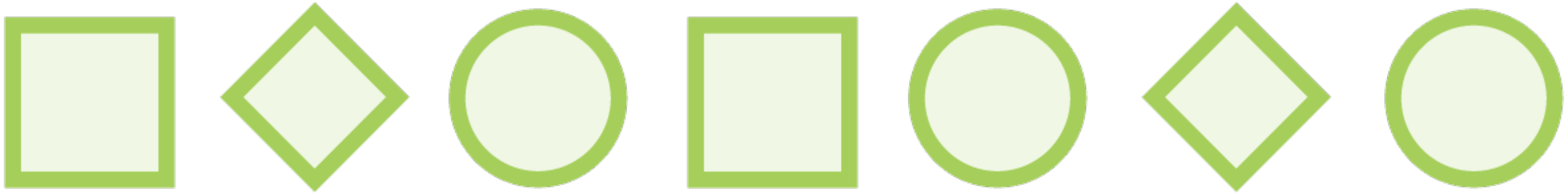
Batch	Stream
Order of data received unimportant	Order important, out of order arrival tracked
Single global state of the world at any point in time	No global state, only history of events received
Processing code “knows” all data	Processing code does not know what lies ahead

Batch vs. Stream Processing

Batch	Stream
Payroll processing	Fraud detection
No latency threshold	Latency important
All employee data available before processing begins	New data keeps coming - need to detect fraud quickly without slowing down legitimate transactions

Stream Processing

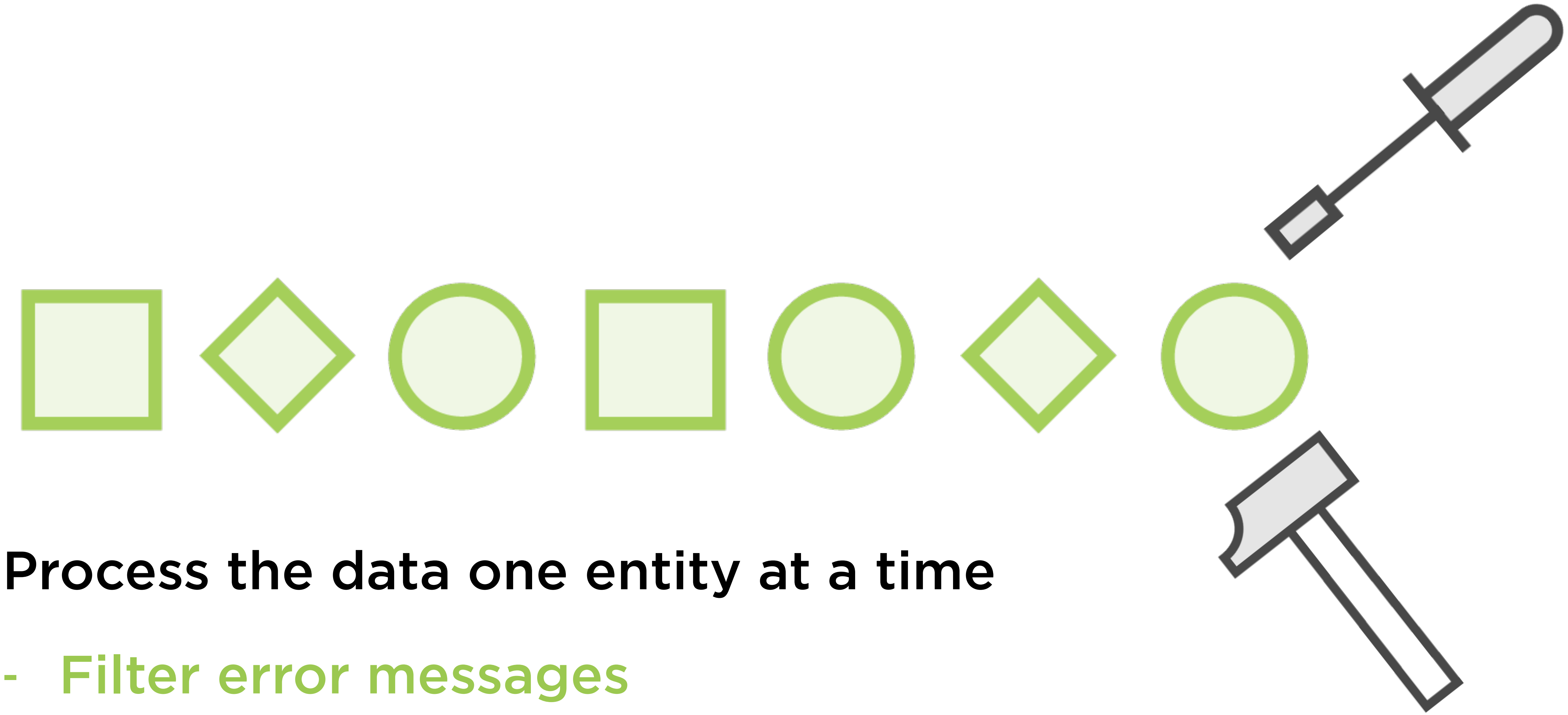
Stream Processing



Data is received as a stream

- Log messages
- Tweets
- Climate sensor data

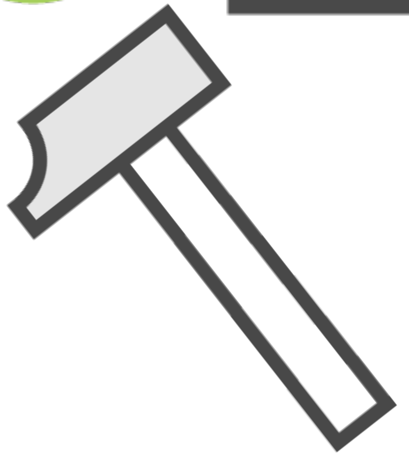
Stream Processing



Process the data one entity at a time

- Filter error messages
- Find references to the latest movies
- Track weather patterns

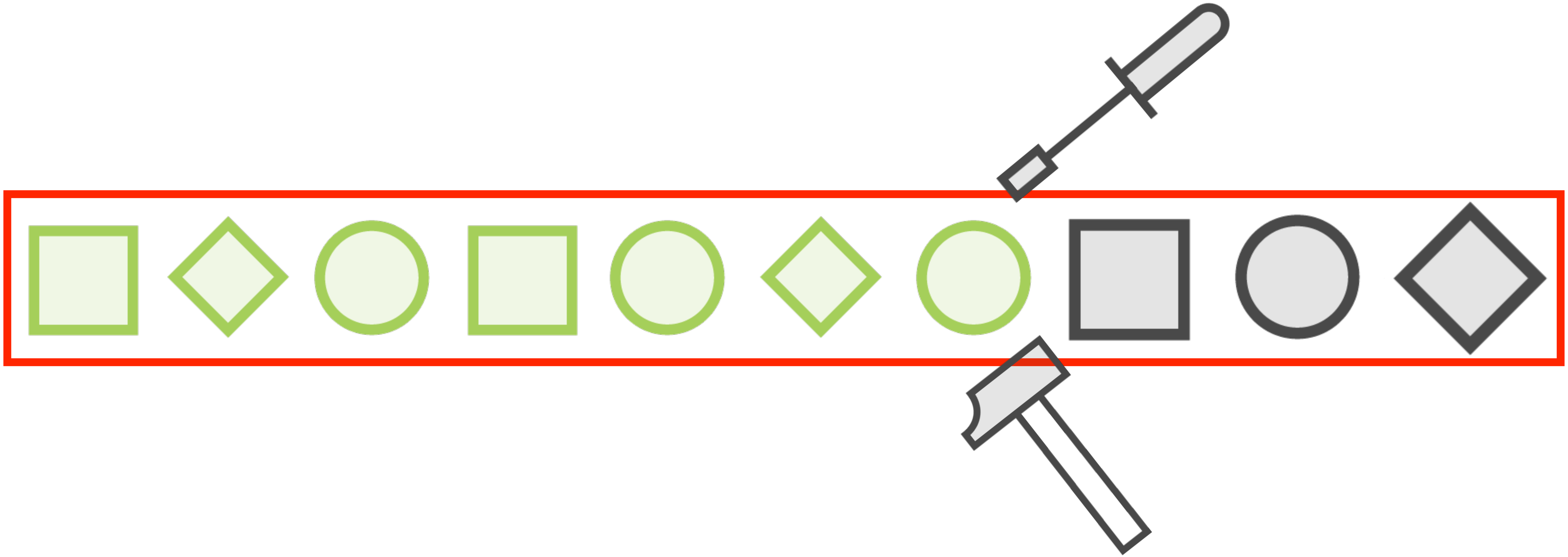
Stream Processing



Store, display, act on filtered messages

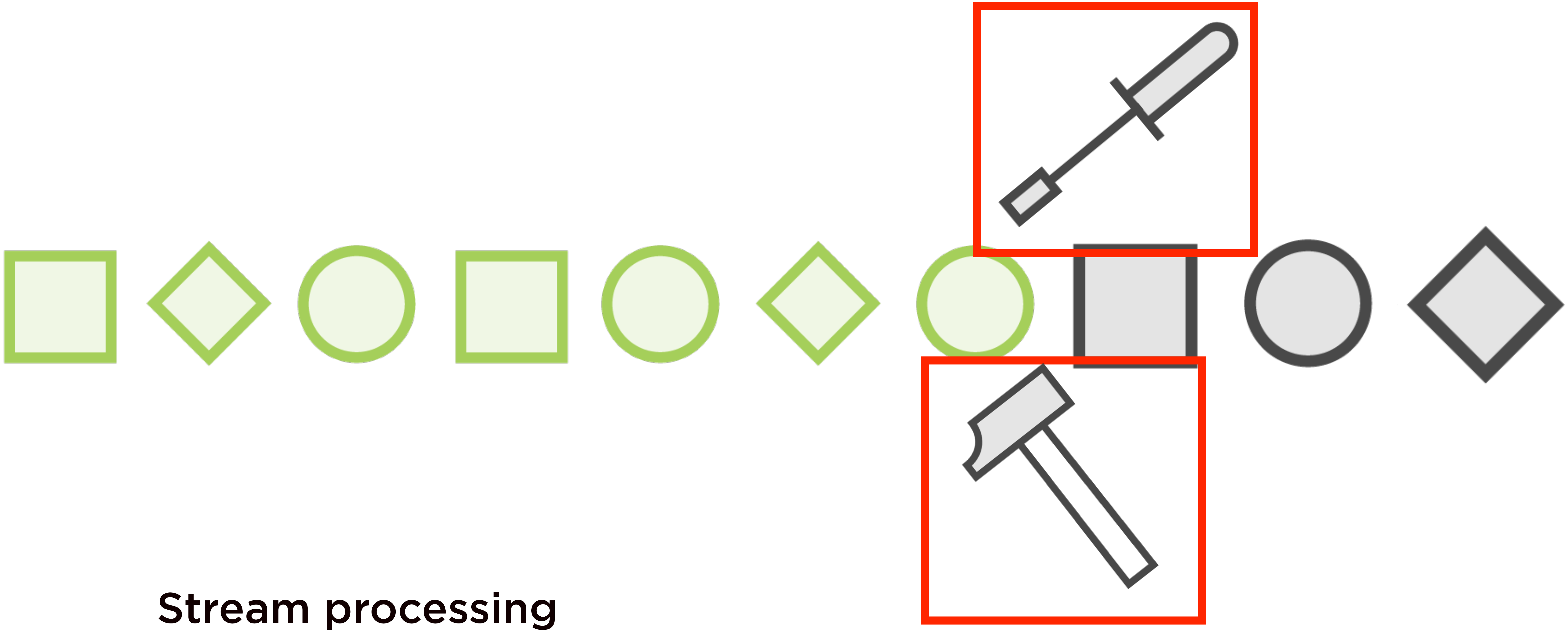
- Trigger an alert
- Show trending graphs
- Warn of sudden squalls

Stream Processing



Streaming data

Stream Processing



Stream processing

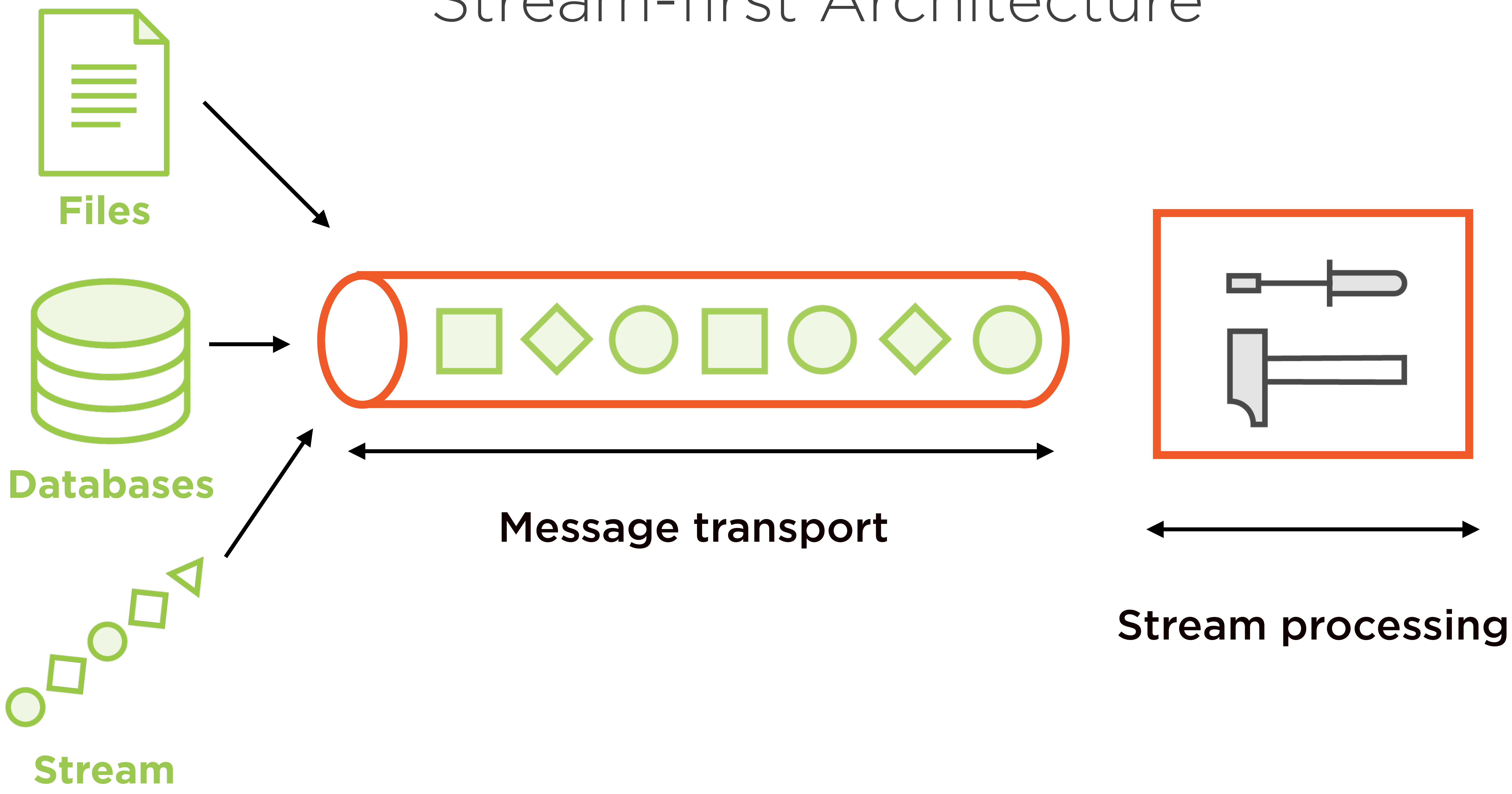
Traditional Systems

Files

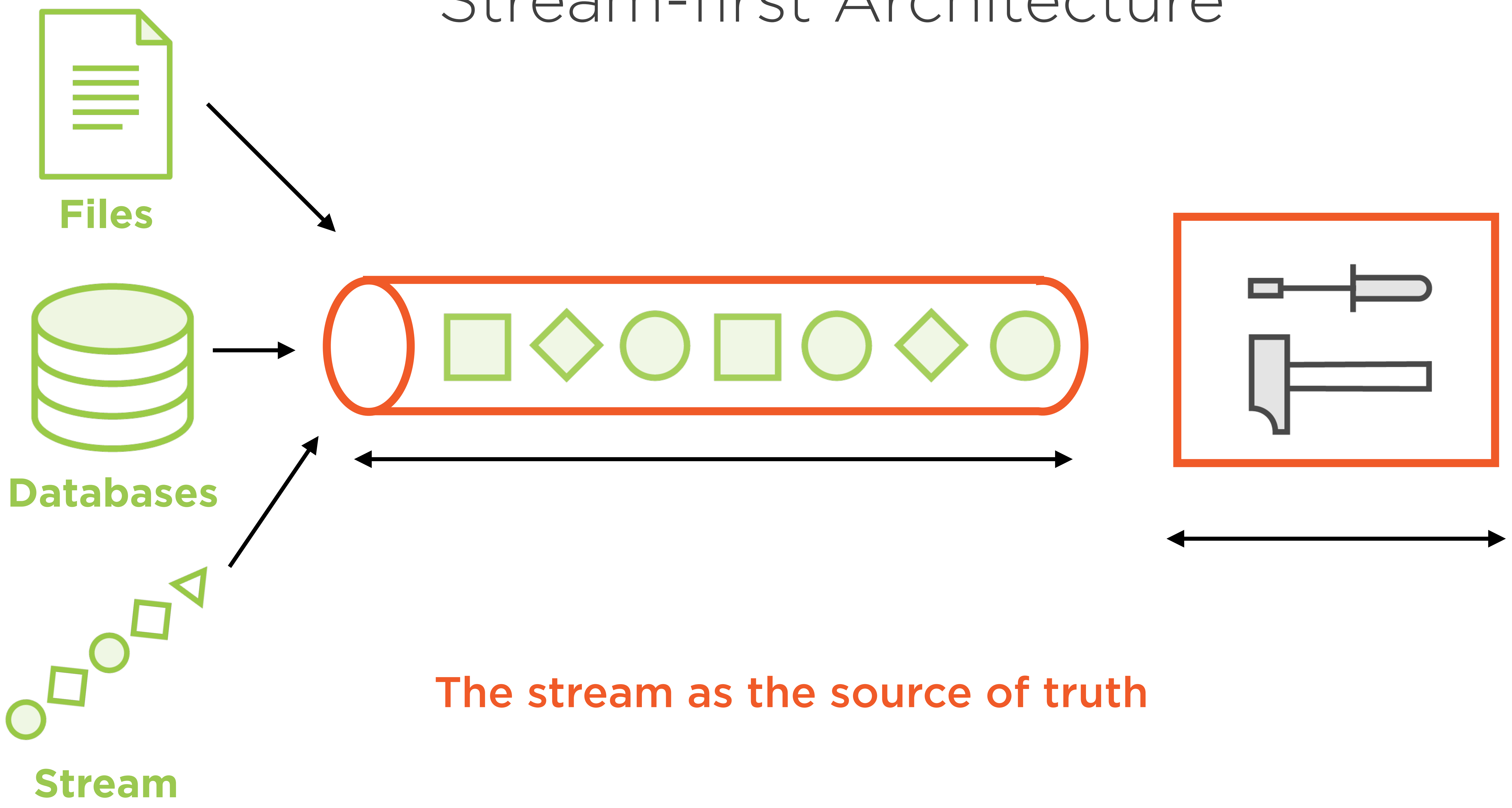
Databases

Reliable storage as the source of truth

Stream-first Architecture



Stream-first Architecture



The stream as the source of truth

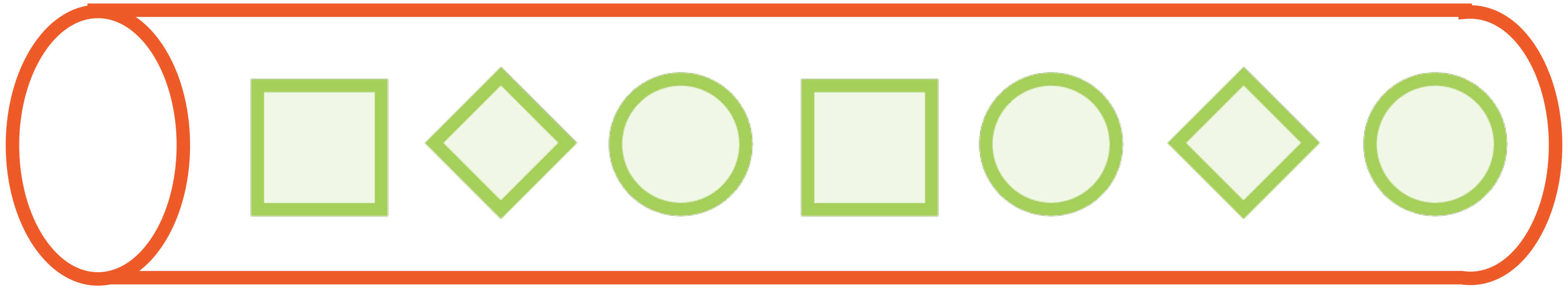
Message Transport

Buffer for event data

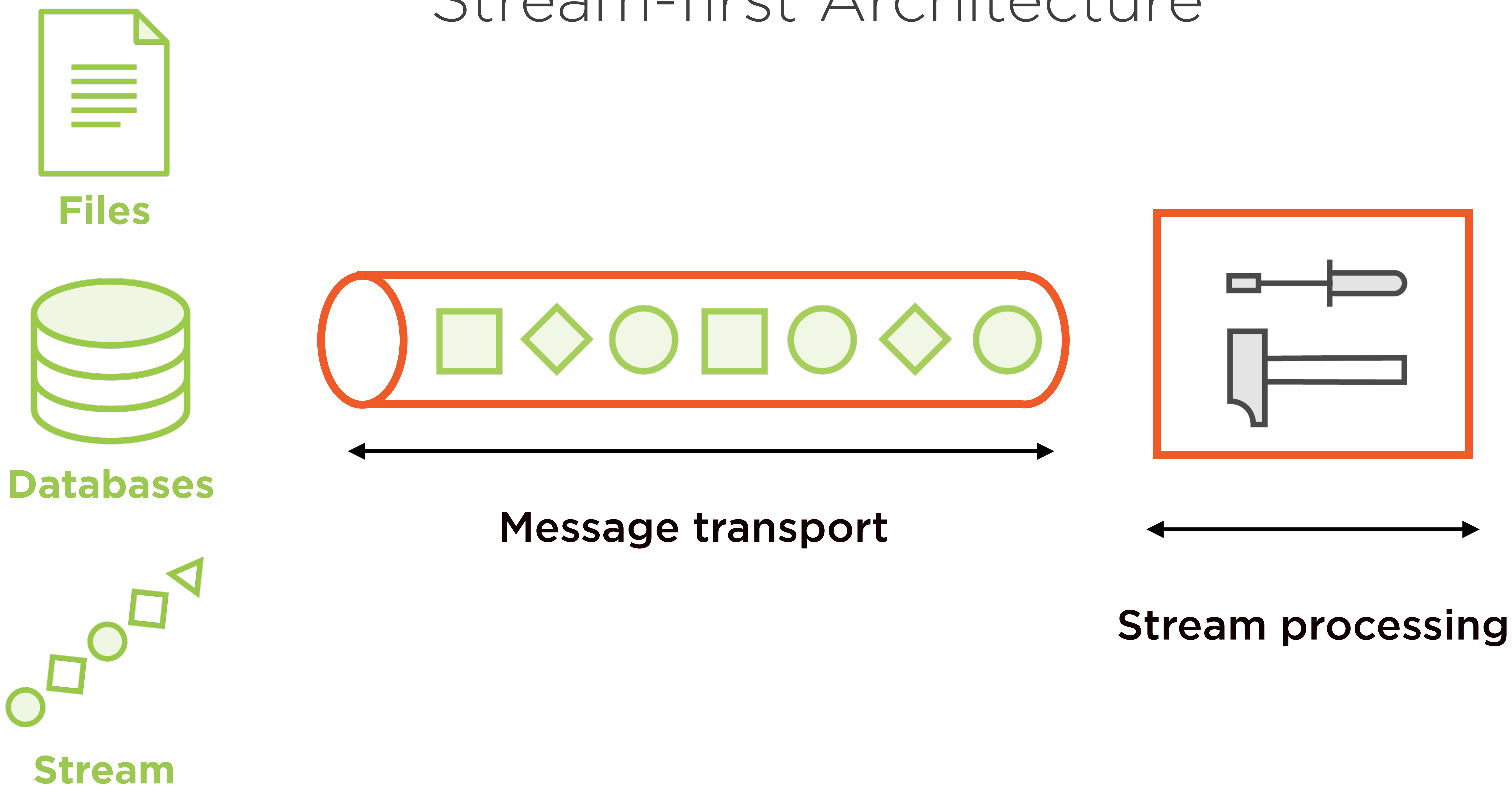
Performant and persistent

Decoupling multiple sources from processing

Kafka, MapR streams



Stream-first Architecture



Stream Processing

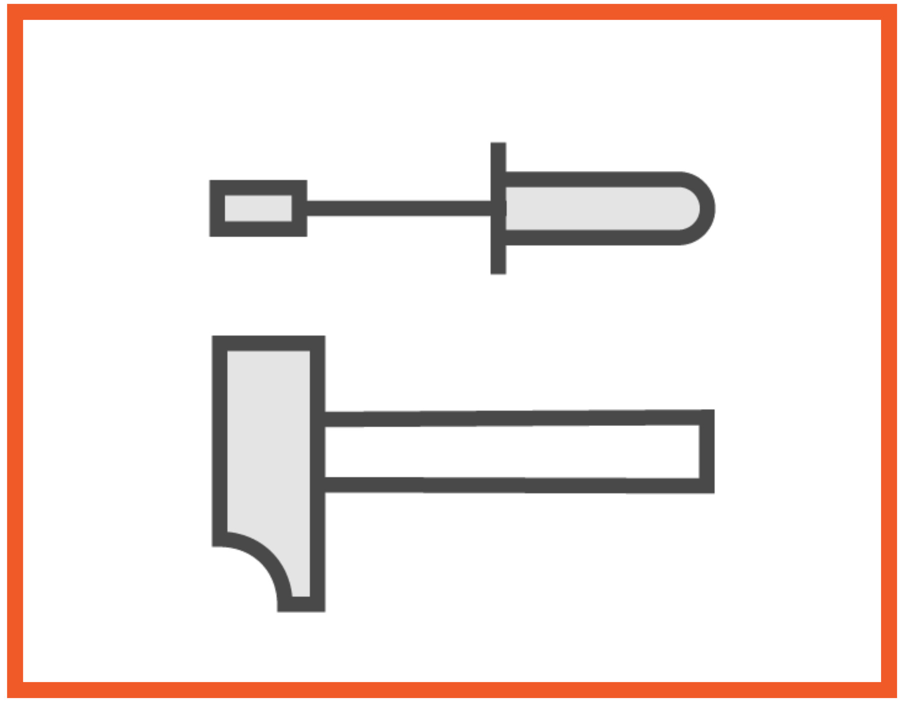
High throughput, low latency

Fault tolerance with low overhead

Manage out of order events

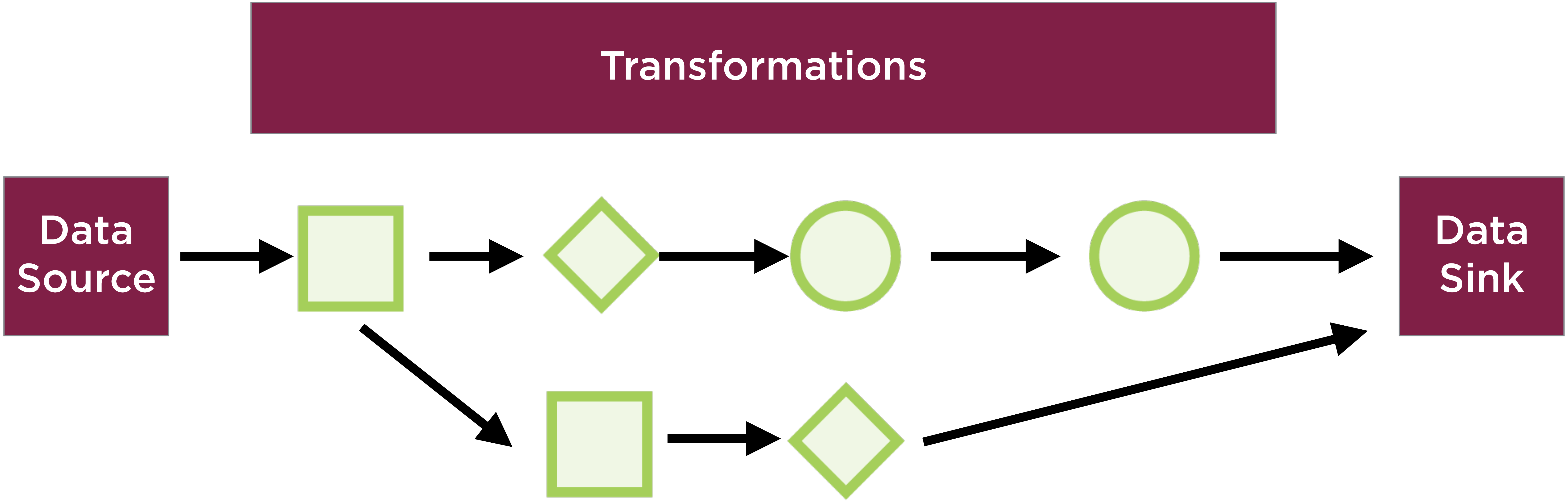
Easy to use, maintainable

Replay streams



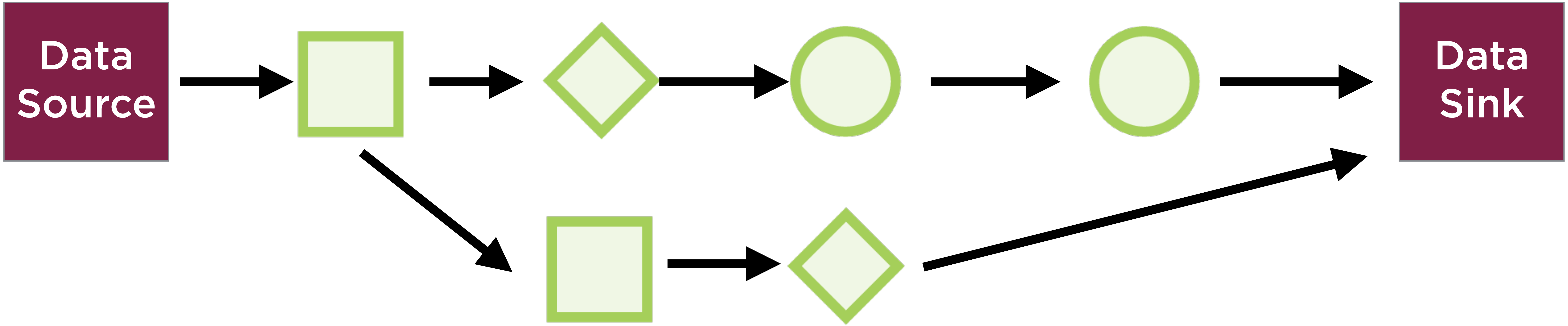
Stream Processing Model

Stream Processing Model



Transformations

A directed-acyclic graph



Stream Processing Models

Stream Processing Models

Stream Processing Models

Stream processing does not necessarily mean continuous real-time processing

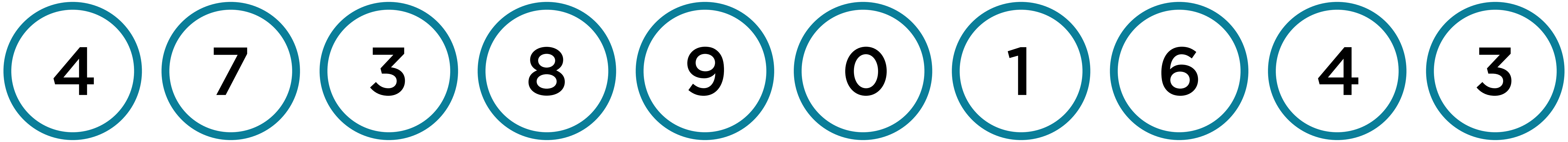
Micro-batch Processing

Run transformations on smaller accumulations of data

Collect say less than one minute of data

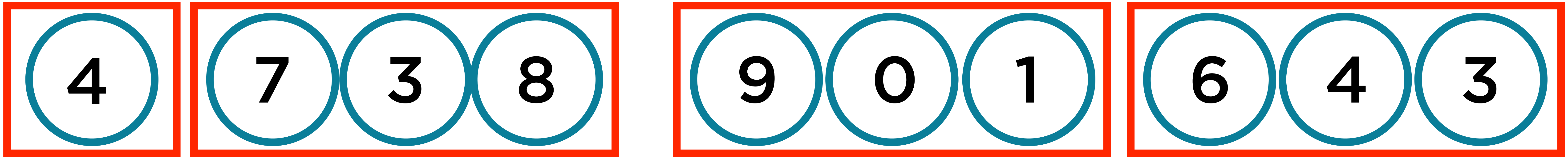
Process this micro-batch in near real-time

Micro-batch Processing



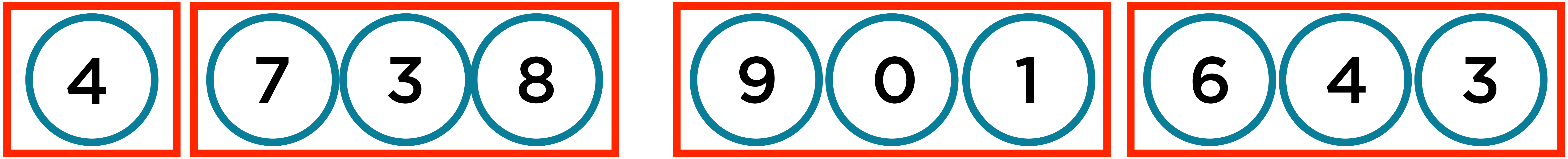
A stream of integers

Micro-batch Processing



Grouped into batches

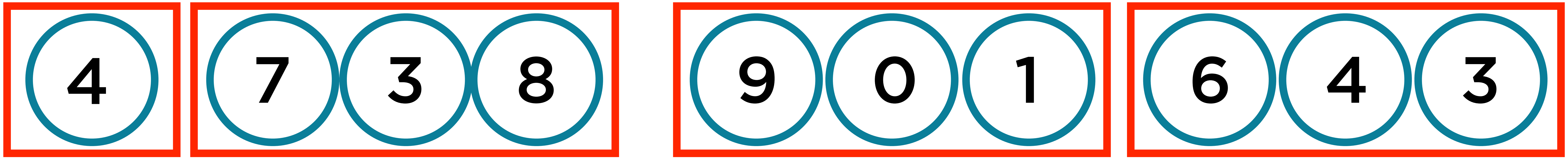
Micro-batch Processing



If the batches are small enough...

Close to real-time processing

Micro-batch Processing



Exactly once semantics

Replay micro-batches

Latency-throughput trade-off based on batch sizes

Batch Processing for Streams

Latency, freshness of data are not considerations

Complex analytical operations

Joins on relational data

- Data might be in a data warehouse, need not be in an RDBMS

Continuous Stream Processing for Streams

Latency and freshness of data are most important considerations

Rate of arrival is high

- Latency in seconds/milliseconds only possible with continuous processing

Micro-batch Processing for Streams

Latency and freshness of data are important

but

Real-time processing is overkill

Rate of arrival is low/moderate

- Latency in seconds/milliseconds less important
- Acceptable latency possible with micro-batches

Stream Processing Architectures

Stream Processing Architectures

Distinct Batch Layer and Stream Layer

Unified Batch and Stream Layers

The difference between these architectures depend on how you treat batch as well as stream data

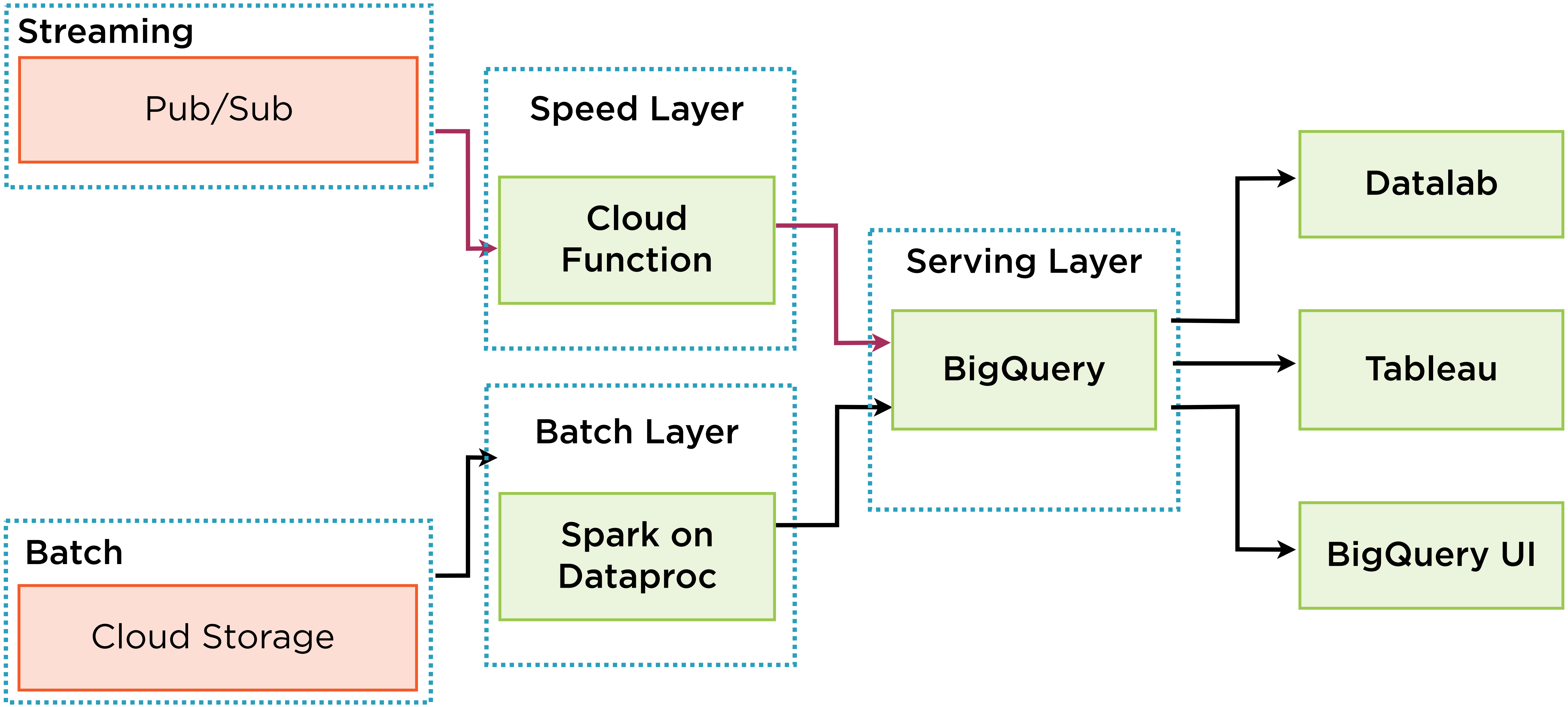
Lambda Architecture

Run a streaming system along with a batch system

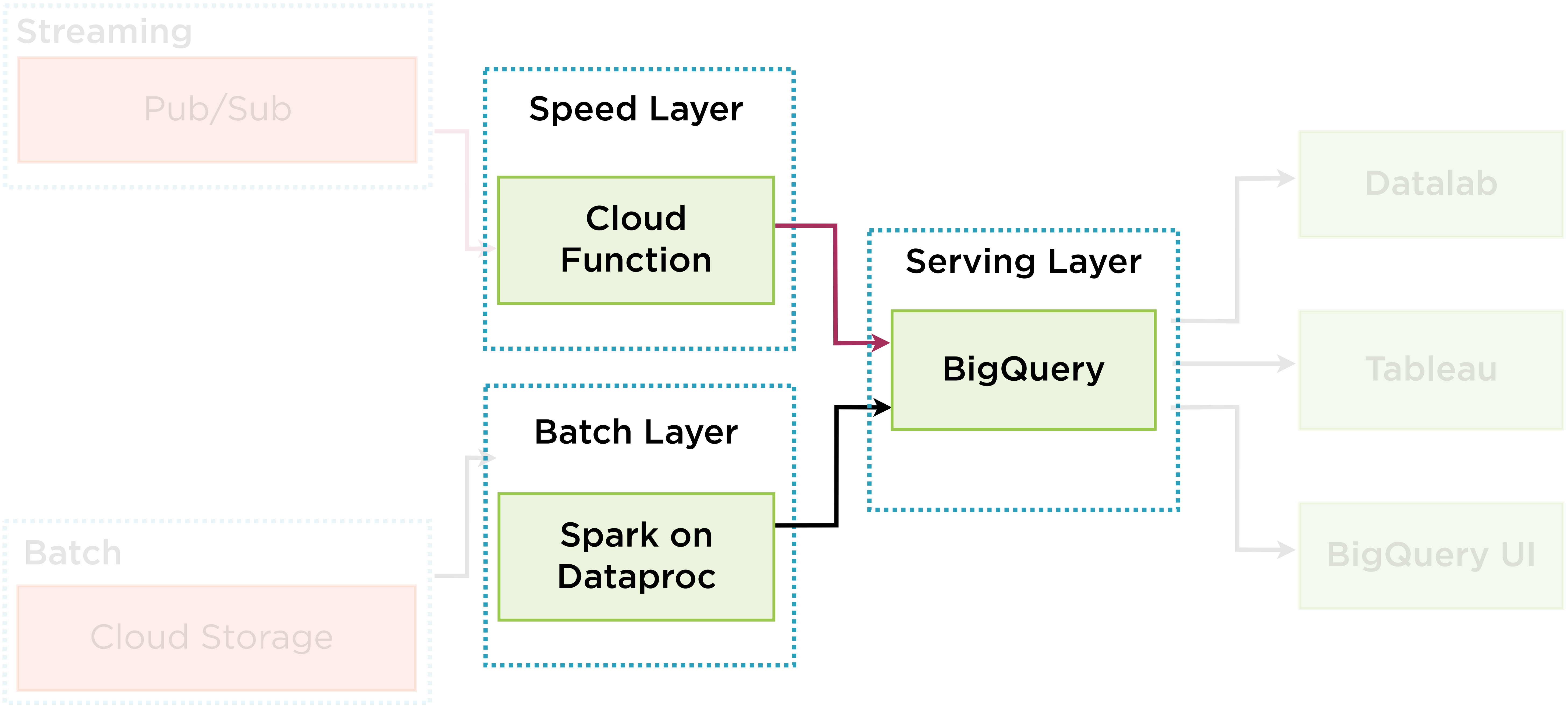
Streaming systems gives you low-latency but approximate results

Batch system ensures correctness, with higher-latency

Lambda Architecture



Lambda Architecture



Hybrid approach to batch and near real-time processing

Why Lambda?

Frameworks make separate batch and stream architecture choices

Because stream-first architecture offer poor performance for pure batch data

Optimizations for batch data are bolted-on, rather than built-in

Problems with Lambda Architectures

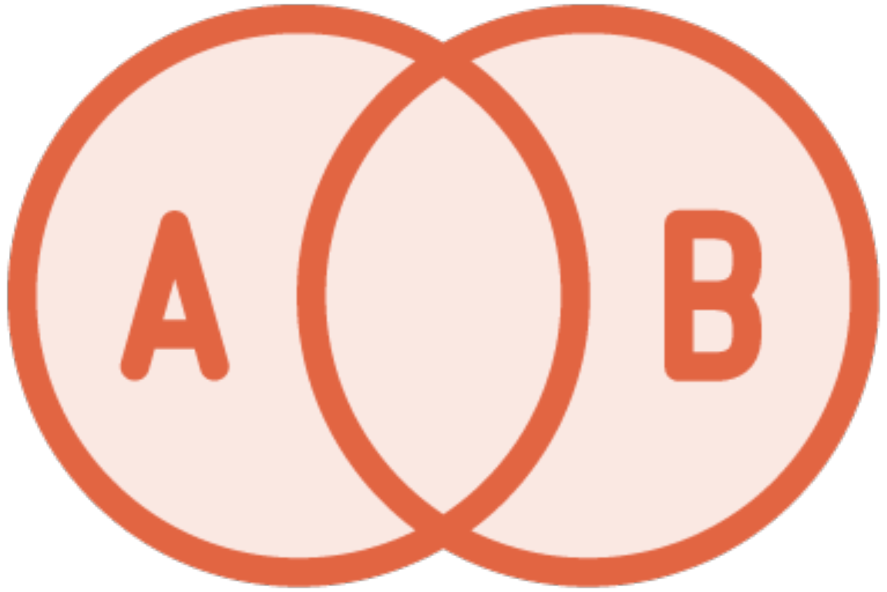
Batch and Stream layers perform the same computation, twice

- Batch computation is perfectly correct, but high latency
- Stream computation is low-latency, but often only approximately correct

Can lead to serious issues

- e.g. Training-serving skew in ML

Kappa Architecture



Kappa architectures tightly integrate batch and streaming

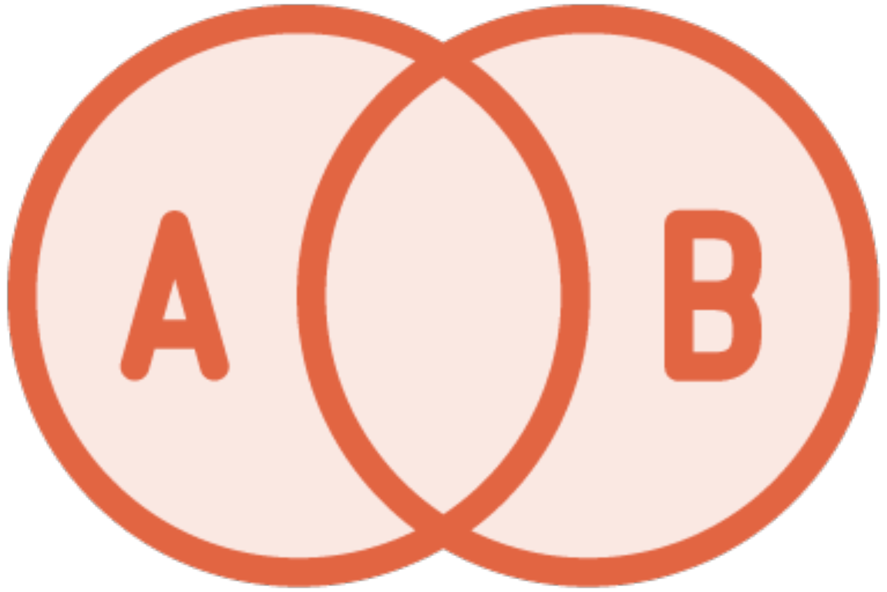
No separate code

Batch code simply fed through the streaming layer

In theory, eliminate training-serving skew

In practice, fragile and can be needlessly complex

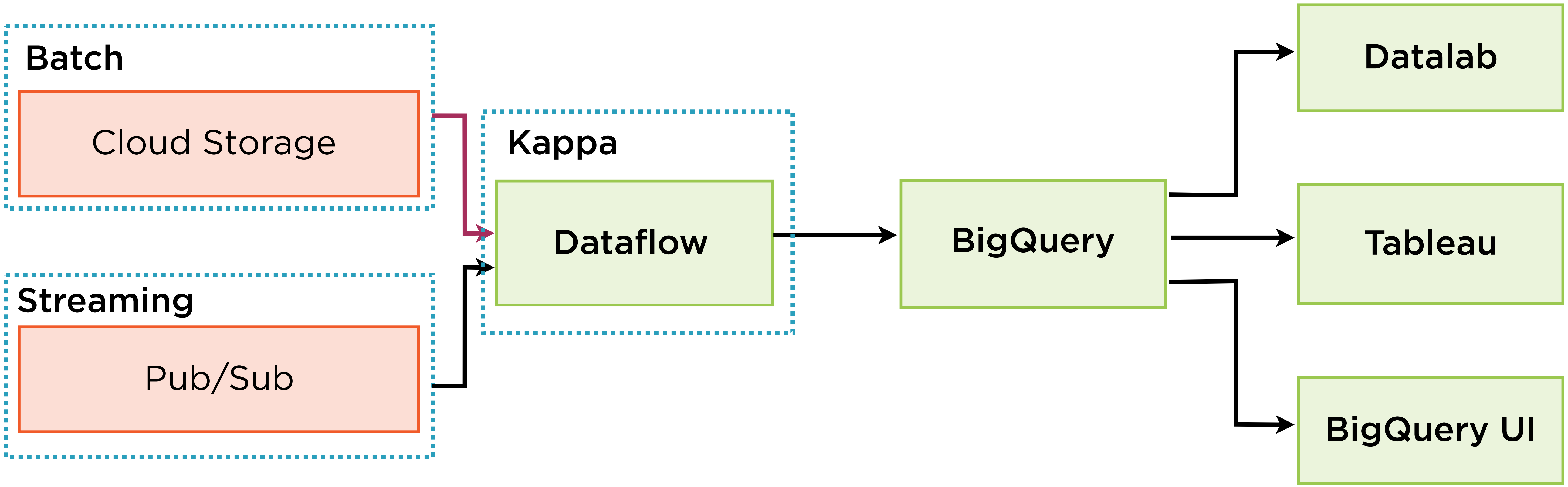
Kappa Architecture



"Batch is a special case of stream"

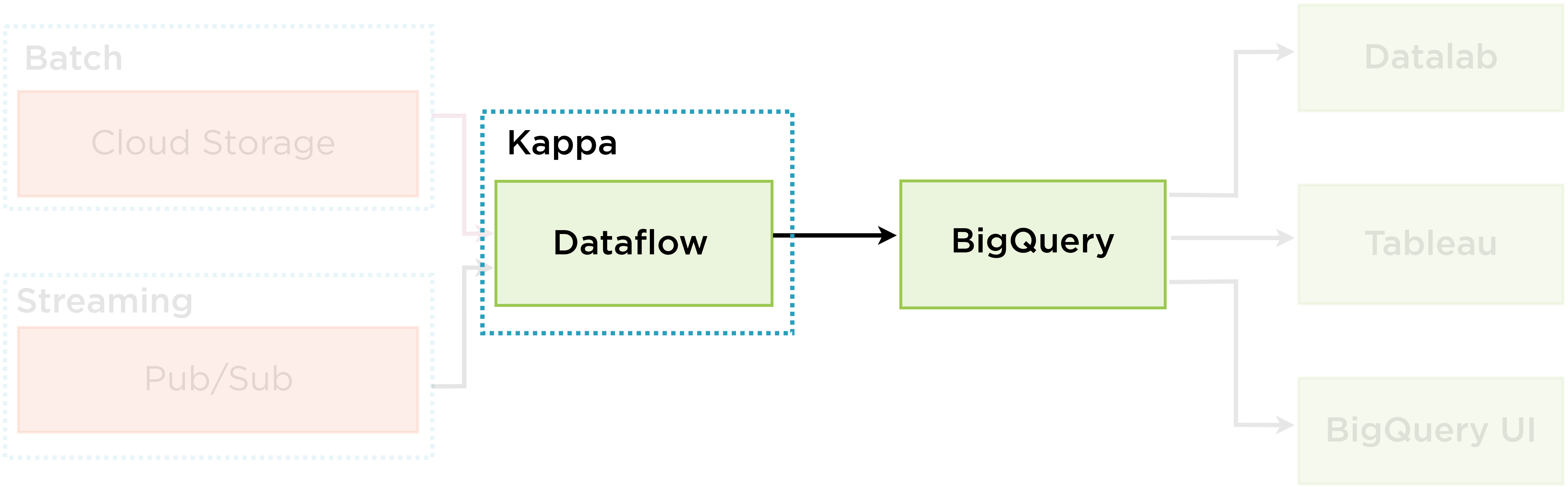
Well-designed streaming systems offer a superset of batch functionality

Kappa Architecture



Process batch and streaming data using the same code

Kappa Architecture



Process batch and streaming data using the same code

Stream Processing APIs

Unified Batch and Stream APIs are becoming more popular - but a Unified API can still rely on any of the architectures under the hood

Challenges of Stream Processing

Challenges of Stream Processing

Latency bounds

Dealing with late, out-of-order data

Guaranteeing reliability

- “Exactly-once, ordered processing”

Security

- Encryption
- Authentication
- MITM attacks

Challenges of Stream Processing

Dimensions of scaling

- Number of senders
- Number of receivers
- Number of messages
- Organizing messages (e.g. topics)
- Size of messages

Summary

Batch data and bounded datasets

Streaming data for unbounded datasets and real-time processing

Micro-batch processing and continuous processing

Lambda and Kappa architectures

Challenges in real-time stream processing

Up Next:

Introducing Apache Beam for Stream Processing