Real-Time Anomaly Detection and Alerting in Financial Markets Using Stream Processing

Machine Learning Prague

👋 Hello

Tun Shwe

Data & AI Consultant

/in/tunshwe

Ben Gamble

Field CTO, Ververica

/in/bengamble7

Training Agenda

1. Stream processing
2. Financial use cases
3. PyFlink & Flink ML
4. The data
5. Run it

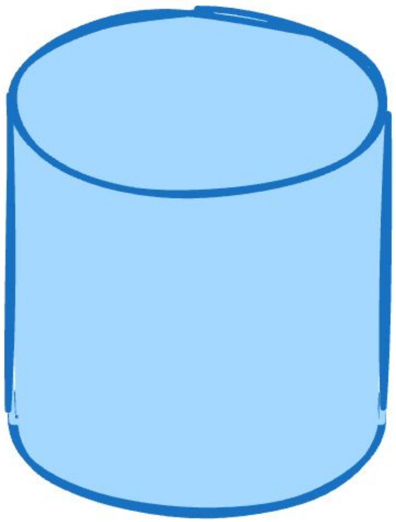
Workshop Requirements

🐳 Complete the Setup section

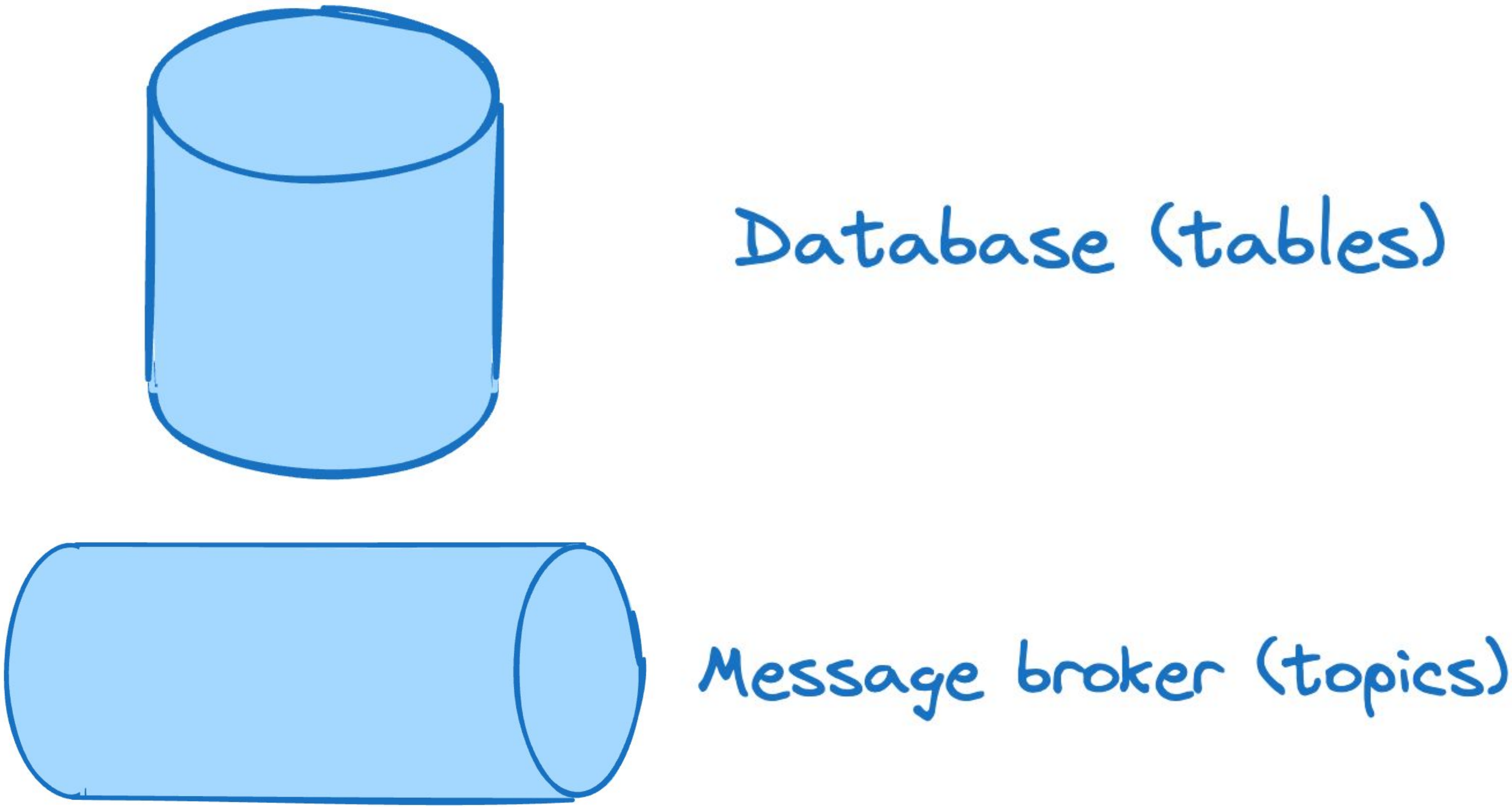
github.com/Ugbot/realtime_ML_Workshop

bit.ly/4cQayM0

🌊 Stream processing



Database (tables)


Database (tables)
Message broker (topics)

CONFLUENT

Buf

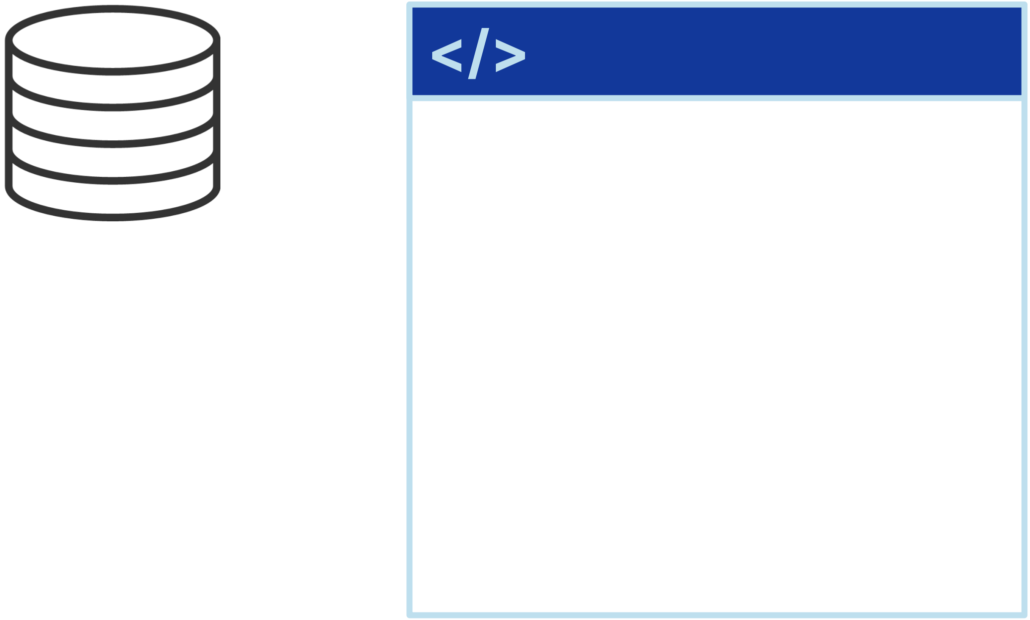
kafka

PULSAR

Batch

Process data at rest

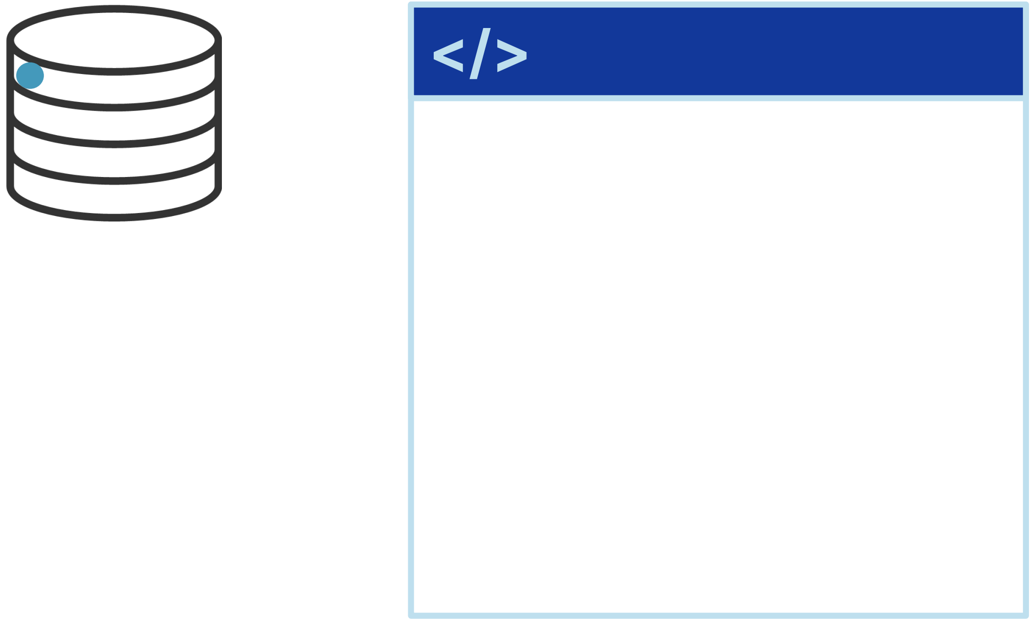
Data is collected over time into a database



Batch

Process data at rest

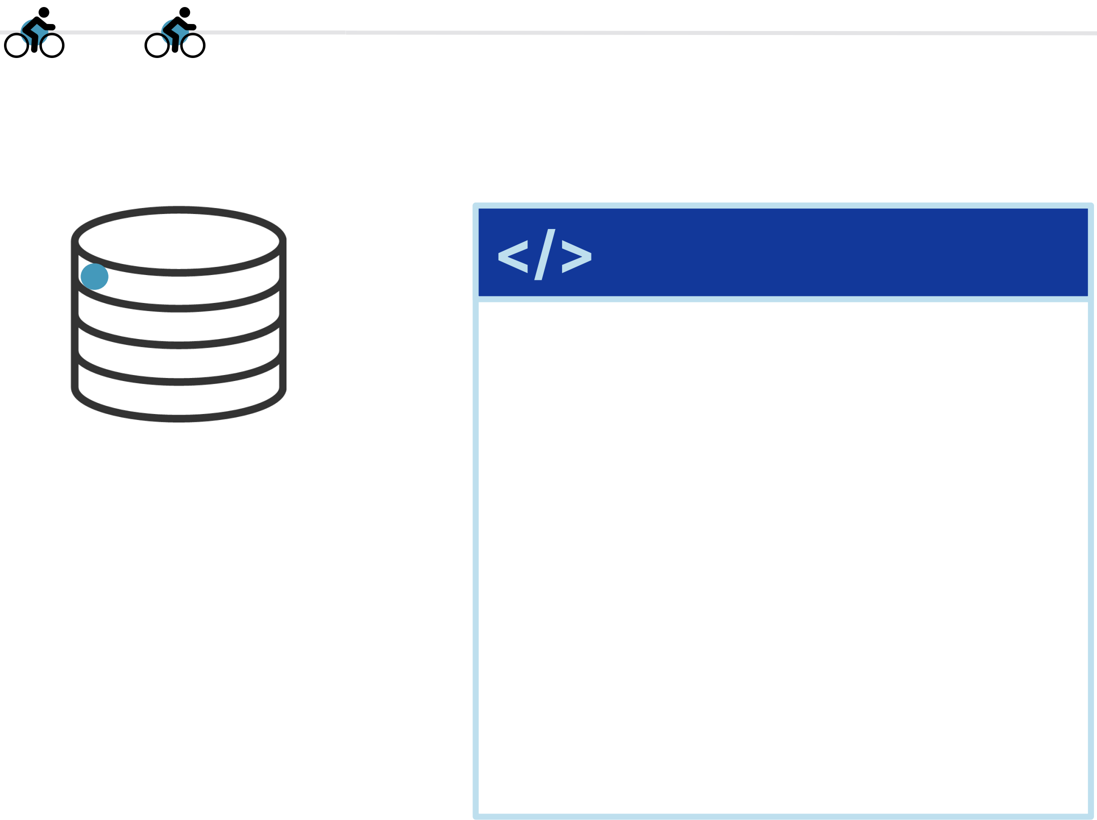
Data is collected over time into a database



Batch

Process data at rest

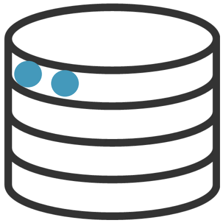
Data is collected over time into a database



Batch

Process data at rest

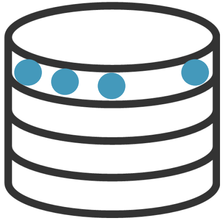
Data is collected over time into a database



Batch

Process data at rest

Data is collected over time into a database

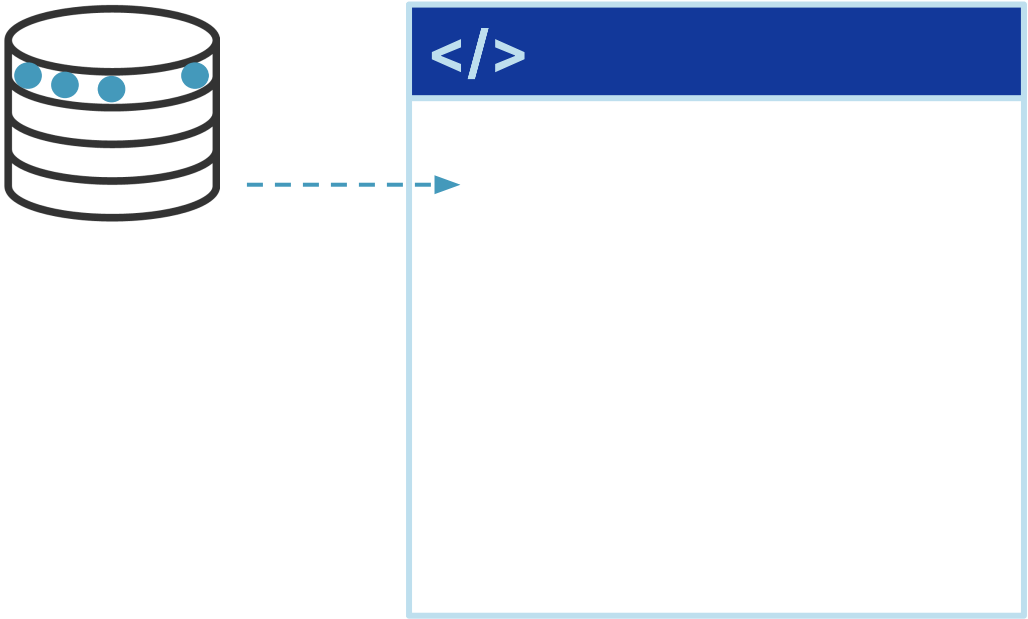


Batch

Process data at rest

Data is collected over time into a database

Bounded data is periodically scheduled to be loaded from the database into the processing system



Batch

Process data at rest

Data is collected over time into a database

Bounded data is periodically scheduled to be loaded from the database into the processing system

</>

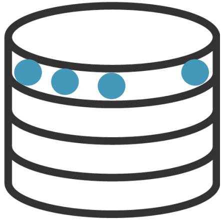
t	Gx	Gy	Gz
t_1	0.1	1.0	0.2
t_2	0.2	1.1	0.1
t_3	0.1	0.9	0.1
t_n	0.3	1.0	0

Batch

Process data at rest

Data is collected over time into a database

Bounded data is periodically scheduled to be loaded from the database into the processing system



</>

t	Gx	Gy	Gz	GT
t_1	0.1	1.0	0.2	1.3
t_2	0.2	1.1	0.1	1.4
t_3	0.1	0.9	0.1	1.1
t_n	0.3	1.0	0	1.3

Batch

Process data at rest

Data is collected over time into a database

Bounded data is periodically scheduled to be loaded from the database into the processing system

</>

t	Gx	Gy	Gz	GT	ΔG
t_1	0.1	1.0	0.2	1.3	
t_2	0.2	1.1	0.1	1.4	0.1
t_3	0.1	0.9	0.1	1.1	-0.3
t_n	0.3	1.0	0	1.3	$1.3 - GT_{n-1}$

Batch

Process data at rest

Data is collected over time into a database

Bounded data is periodically scheduled to be loaded from the database into the processing system

- Computation on all historical data (stateless)
- Results are not in real time

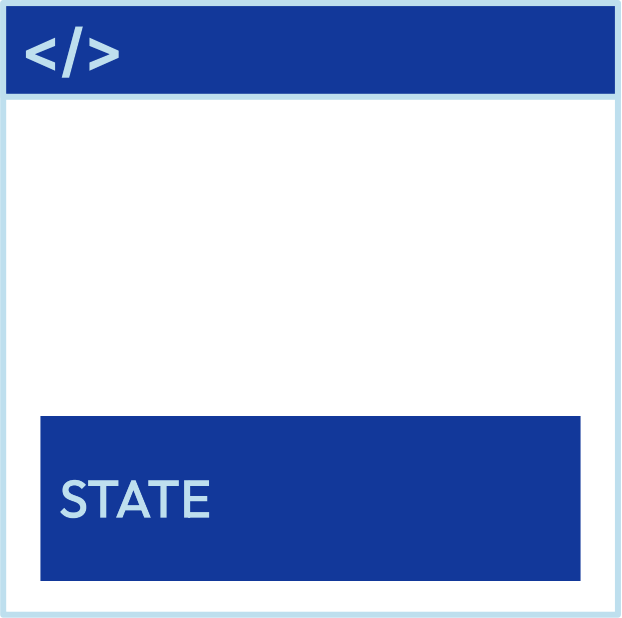
</>

t	Gx	Gy	Gz	GT	ΔG
t_1	0.1	1.0	0.2	1.3	
t_2	0.2	1.1	0.1	1.4	0.1
t_3	0.1	0.9	0.1	1.1	-0.3
t_n	0.3	1.0	0	1.3	$1.3 - GT_{n-1}$

Stream

Process data in motion

Data is collected over time into a broker/transport (Kafka topic)

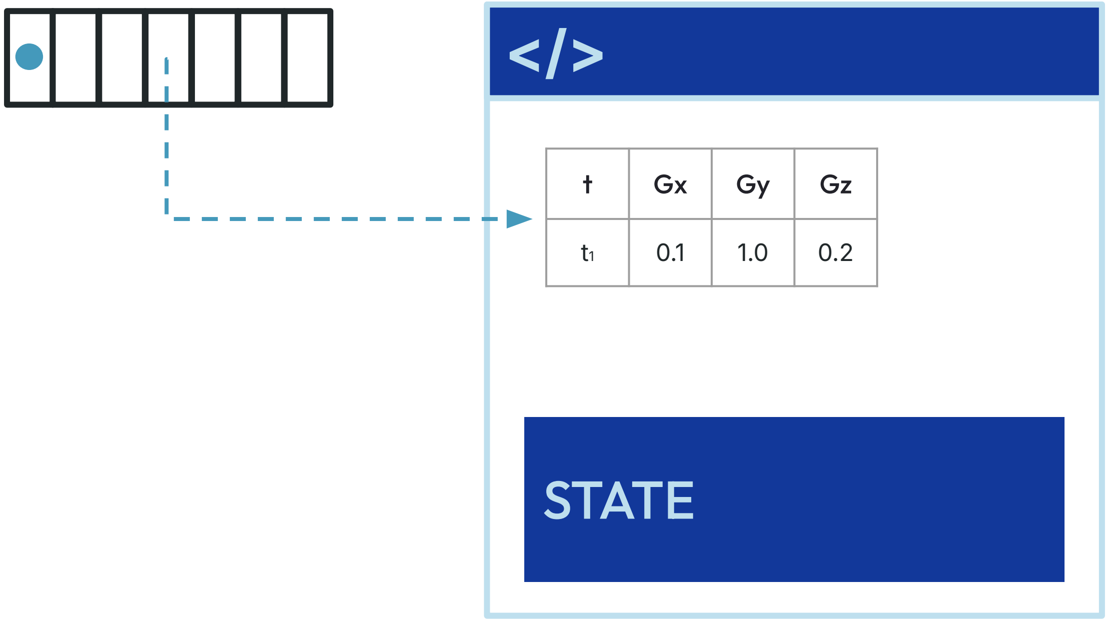


Stream

Process data in motion

Data is collected over time into a broker/transport (Kafka topic)

Unbounded data is continuously consumed by the processing system as soon as new data is published to the topic



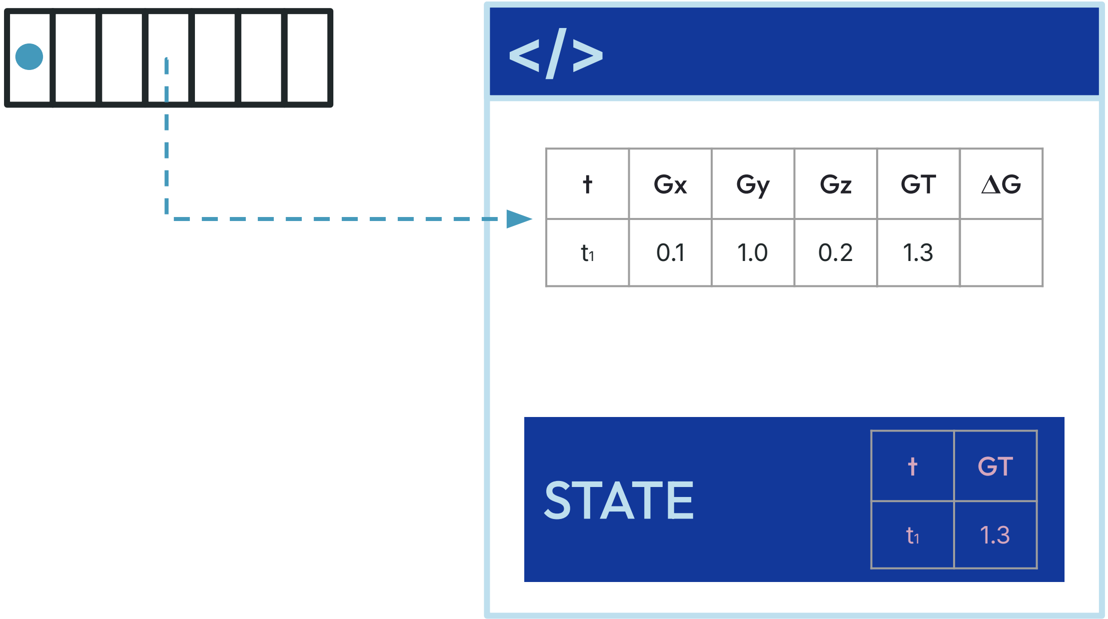
t	Gx	Gy	Gz
t_1	0.1	1.0	0.2

Stream

Process data in motion

Data is collected over time into a broker/transport (Kafka topic)

Unbounded data is continuously consumed by the processing system as soon as new data is published to the topic

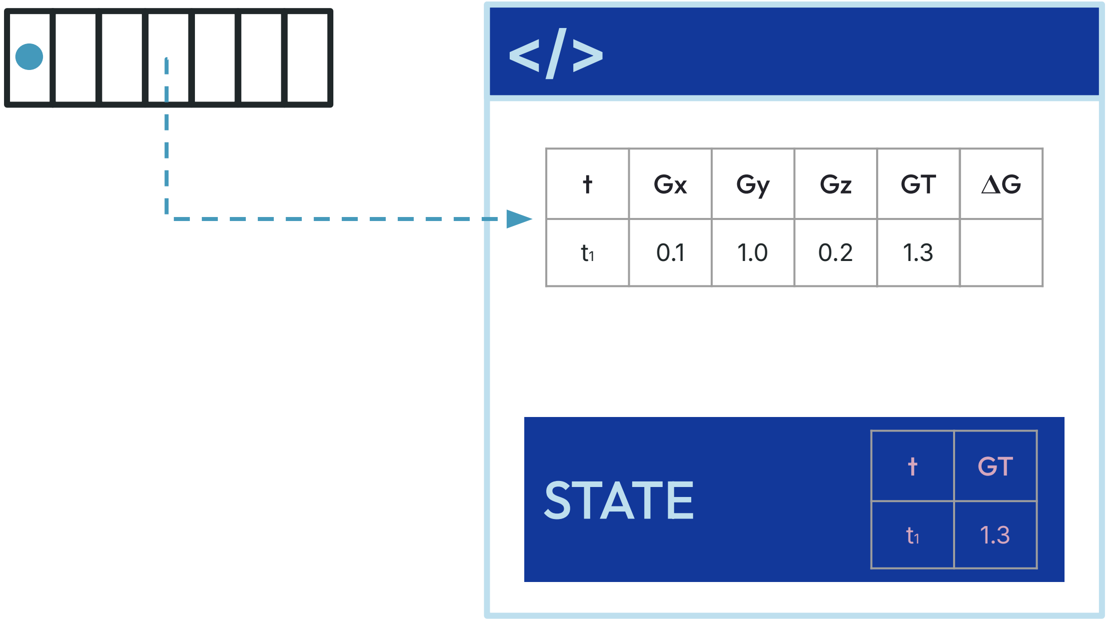


Stream

Process data in motion

Data is collected over time into a broker/transport (Kafka topic)

Unbounded data is continuously consumed by the processing system as soon as new data is published to the topic

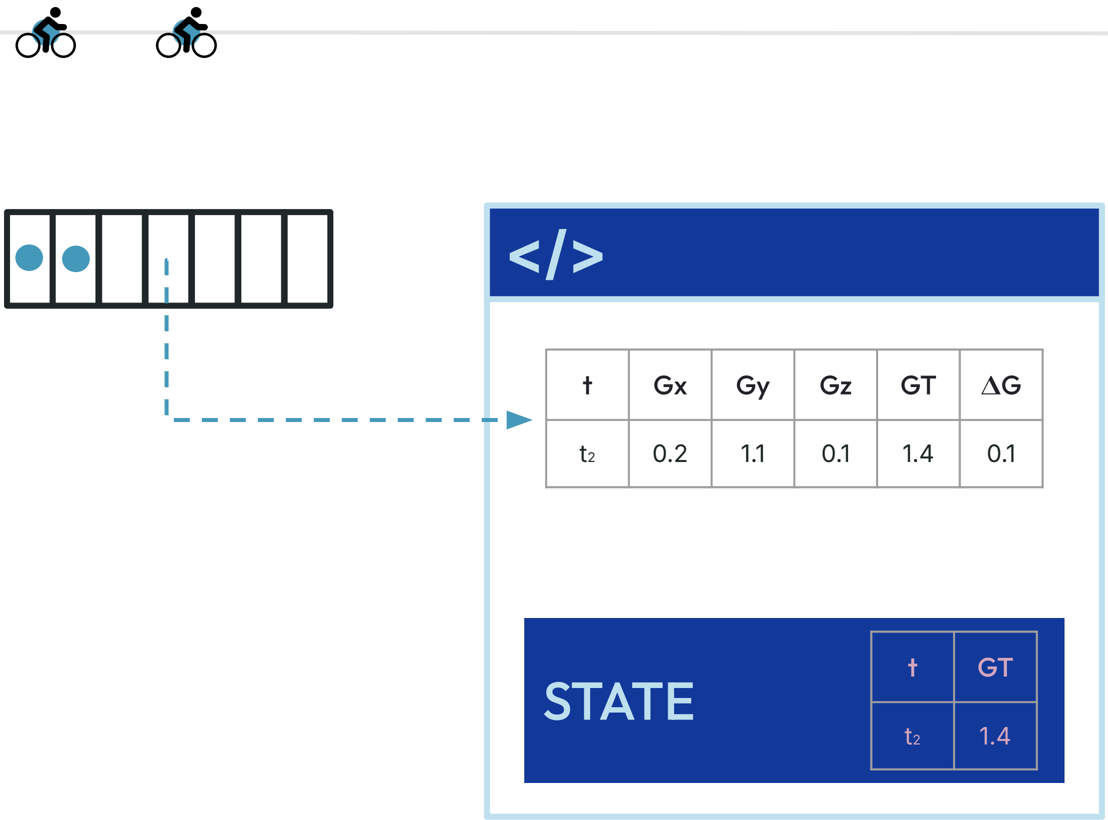


Stream

Process data in motion

Data is collected over time into a broker/transport (Kafka topic)

Unbounded data is continuously consumed by the processing system as soon as new data is published to the topic

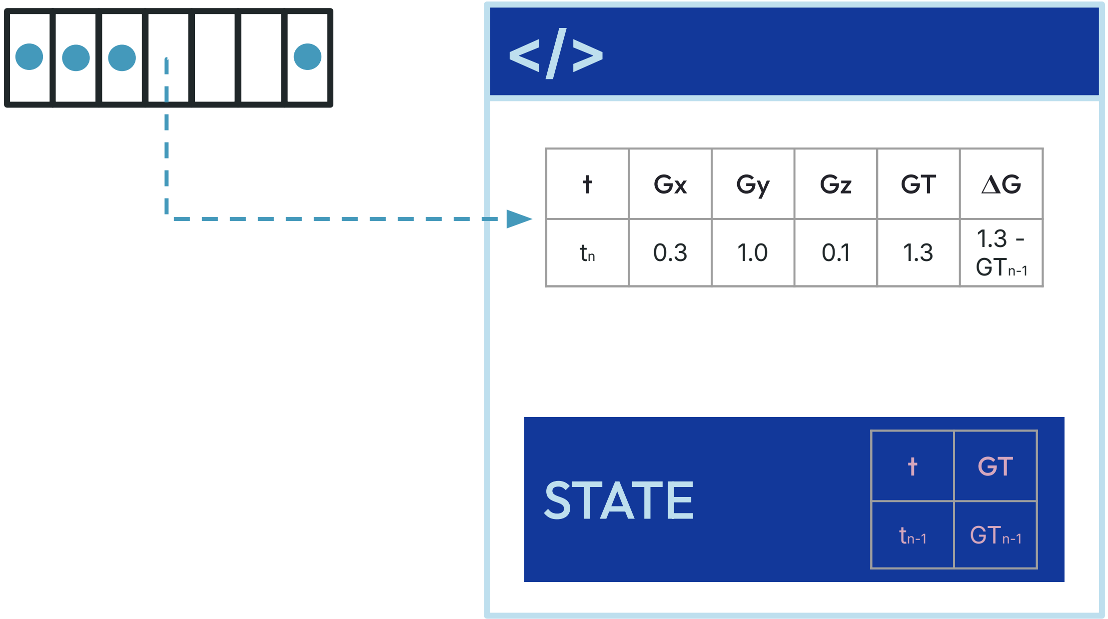


Stream

Process data in motion

Data is collected over time into a broker/transport (Kafka topic)

Unbounded data is continuously consumed by the processing system as soon as new data is published to the topic



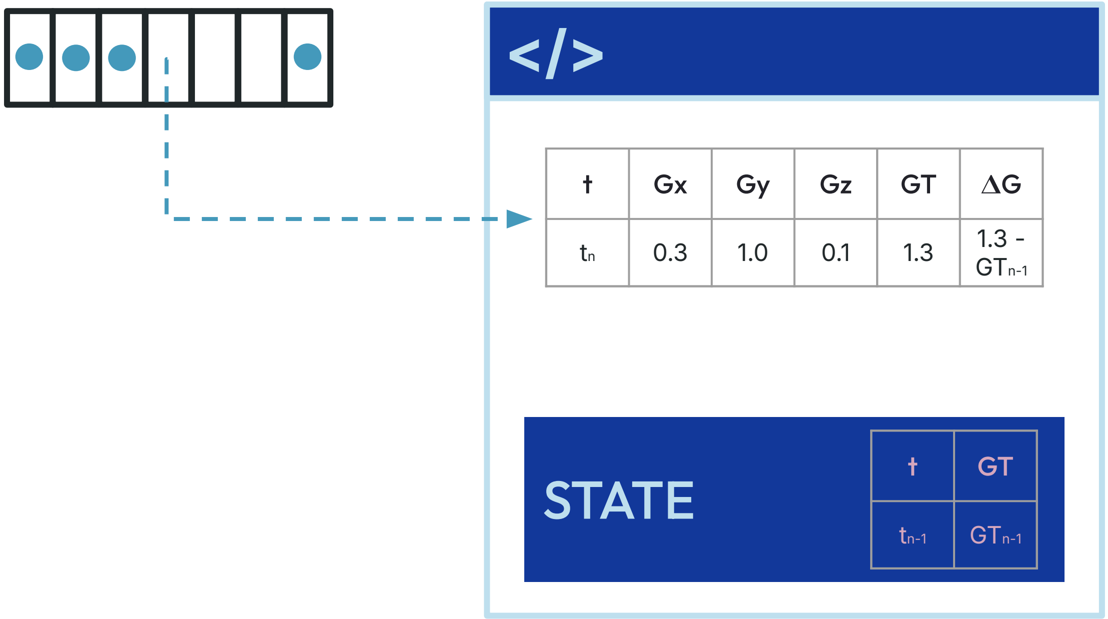
Stream

Process data in motion

Data is collected over time into a broker/transport (Kafka topic)

Unbounded data is continuously consumed by the processing system as soon as new data is published to the topic

- Computation on each event. Requires state for historical data (stateful)
- Real-time results



t	Gx	Gy	Gz	GT	ΔG
t_n	0.3	1.0	0.1	1.3	1.3 - GT_{n-1}

t	GT
t_{n-1}	GT_{n-1}

Batch or stream?

Batch or stream? Both!

Batch and Stream together

- Batch and stream are complementary (**one does not replace the other**)
- Batch for historical context
- Stream for current context
- Most teams try to fit streaming into a batch system
- Much easier to fit batch into a streaming system, where **events are the foundational unit**

Why is Kafka fast?

Sequential I/O

- Kafka is an append-only log
- Stored data is organised as contiguous blocks of memory
- Modern drives and SSDs are optimised for sequential I/O
- Contrast that with databases that are optimised for random I/O

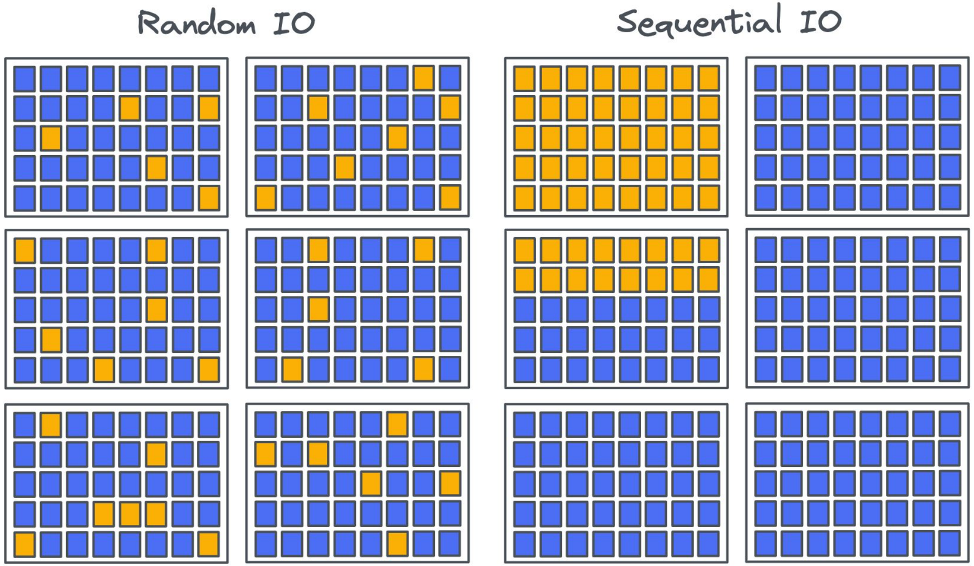


Image: Jack Vanlightly
https://jack-vanlightly.com/blog/2023/5/9/is-sequential-io-dead-in-the-era-of-the-nvme-drive

Why is Kafka fast?

Zero Copy principle

- Refers to the copying of data between kernel and application representations
- Consumers read topic data directly from the log file using direct memory access (DMA)
- Doesn't apply when encryption/SSL/TLS is used

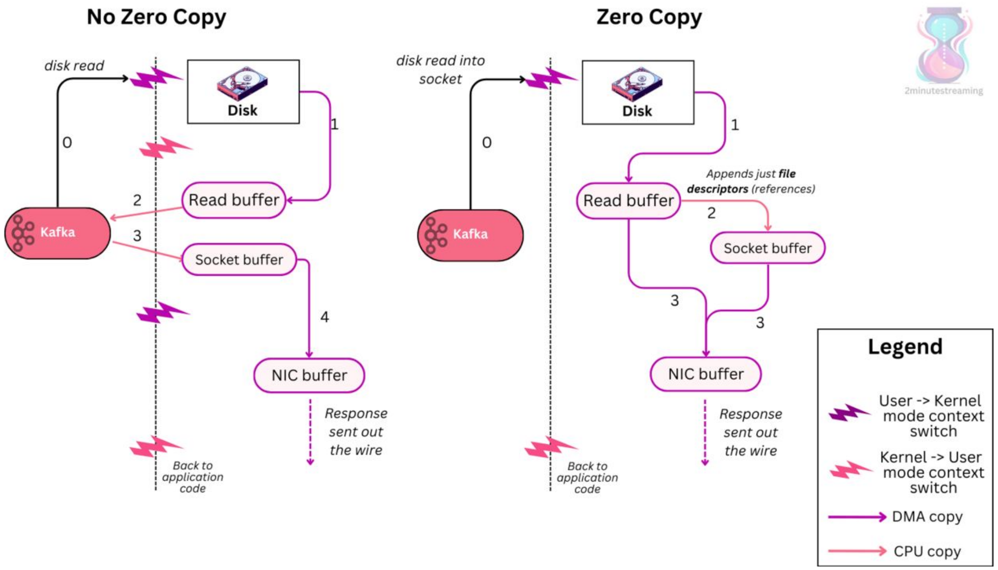


Image: Stanislav Kozlovski
https://2minutestreaming.beehiiv.com/p/apache-kafka-zero-copy-operating-system-optimization

Streaming data use cases

Retail, e-commerce and customer service 🛍️

- Real-time personalisation and recommendations
- Inventory management

Manufacturing, energy and IIoT

- Condition monitoring and operational efficiency
- Predictive maintenance

Financial services

- Fraud detection
- Algorithmic trading and risk assessment

Streaming data use cases

Retail, e-commerce and customer service

- Real-time personalisation and recommendations
- Inventory management

Manufacturing, energy and IIoT 🏭

- Condition monitoring and operational efficiency
- Predictive maintenance

Financial services

- Fraud detection
- Algorithmic trading and risk assessment

Streaming data use cases

Retail, e-commerce and customer service

- Real-time personalisation and recommendations
- Inventory management

Manufacturing, energy and IIoT

- Condition monitoring and operational efficiency
- Predictive maintenance

Financial services 💹

- Fraud detection
- Algorithmic trading and risk assessment

💹 Financial use cases

Types of anomaly detection

- 2 types of anomaly detection
- Outliers: Individual data points that deviate significantly from the general data distribution
- Drift: A gradual or sudden change in the underlying data distribution over time
- Outlier detection focuses on individual anomalies
- Drift detection focuses on broader changes

Popular real-time stock market indicators

- Relative Strength Index (RSI)
- Moving Average Convergence Divergence (MACD)
- Bollinger Bands

Relative Strength Index (RSI)

- Momentum oscillator measuring speed and change of price movements
- Identifies overbought or oversold stock
- Traders can determine when to buy and when to sell

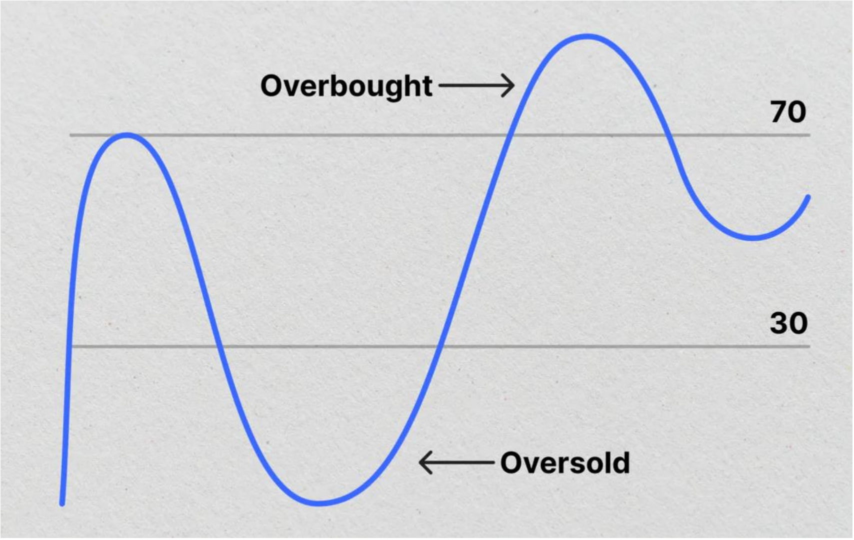


Image: https://liquidity-provider.com/articles/what-is-rsi-how-to-use-rsi-in-trading/

Moving Average Convergence Divergence (MACD)

- Shows relationship between 2 moving avg (MACD line, signal line, histogram)
- Identifies changes in direction, momentum and duration of a stock
- Traders can determine when to enter long or short positions

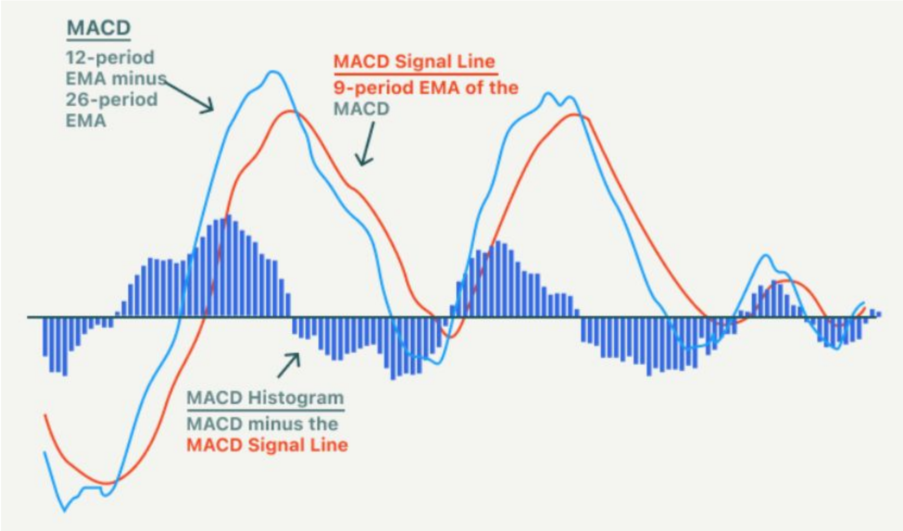


Image: https://www.sahmcapital.com/education/article/moving-average-convergence-divergence-macd-explained

Bollinger Bands

- Simple moving average and 2 standard deviations above and below
- Identifies market volatility
- Traders can sell when price reaches the upper band and buy at the lower band

Image: https://www.truedata.in/blog/bollinger-bands-all-you-need-to-know/

🐿️ PyFlink & Flink ML

The history of Apache Flink

- Started in 2010
- Joined Apache in 2014
- Designed to give real-time processing some better semantics

Why is Apache Flink special?

- Stream first
    - Though it came from the Dataset API, it now Supports The table API
    - Newest FLIPs are all about near real time

- Proven at scale
    - TikTok runs on a Flink stack for online inference
    - Past Flink Forward talks into this space

So it's like Apache Spark!

Apache Flink *is* like Apache Spark

- Its got a similar set of APIs and interfaces!
- Can run ML in Java and Python
- Works well with a Jupyter Notebook
- Similar types of integrations
- Can do large scale batch processing

But not really!

- Everything is a stream in Flink
- It’s really setup to run models and preprocess data
- It's not as rich an ecosystem
- It’s way lower latency
- Training a model is rarely the whole story

So it's like Quix/Faust/Bytewax!

No, not really!

- Flink has its own ecosystem
- Java OR Python*
- No Kafka dependency
- Stronger guarantees
- Task management built in
- Scaling built in
- Proven at scale

So what can it do?

So what can it do?

- Text Feature Extraction
  - **Tokenizer**: Splits text into words or tokens. This is a common preprocessing step for text data
  - **StopWordsRemover**: Removes common words (stop words) from text data, which can improve model performance
  - **NGram**: Extracts n-grams from text data, which can capture word sequences that are important for understanding the meaning of the text
  - **CountVectorizer**: Converts text data into a numerical representation, where each feature represents the frequency of a term in the document
  - **TF-IDF (Term Frequency-Inverse Document Frequency)**: Calculates the importance of a term in a document relative to the corpus. This is a common technique for weighting terms in text data

So what can it do?

- Classification
    - **Linear SVC**: A support vector machine classifier that can be utilized for collaborative filtering by framing the recommendation problem as a classification task
    - **Logistic Regression**: A statistical model that can be employed for collaborative filtering by modeling the probability of a user rating an item as a logistic function
    - **Naive Bayes**: A probabilistic classifier that can be used for collaborative filtering by assuming conditional independence of features (user-item ratings) given the target variable (recommendation)

So what can it do?

- Categorical Feature Encoding
  - **OneHotEncoder**: Converts categorical features into a binary vector, where each element represents the presence or absence of a category. This is useful for algorithms that expect numerical input, such as linear regression or decision trees
  - **StringIndexer**: Encodes categorical features into numerical labels, often a preprocessing step before applying other transformations. This is useful for algorithms that require numerical input but don't require one-hot encoding

So what can it do?

- Numerical Features as well
  - **MinMaxScaler**: Scales numerical features to a specified range, typically between 0 and 1. This is useful for algorithms that are sensitive to the scale of features, such as k-means clustering or support vector machines
  - **StandardScaler**: Standardizes numerical features to have a mean of 0 and a standard deviation of 1. This is useful for algorithms that assume normally distributed features, such as linear regression or logistic regression

An AI pipeline

1. Start with a stream of text
2. Vectorise it
3. Classify the vector via clustering
4. Find the top topic over time

So what about AI?

So what about AI?

Yes!

So what about AI?

- KNN Vector search
- Clustering
- Local state
- Remote lookup

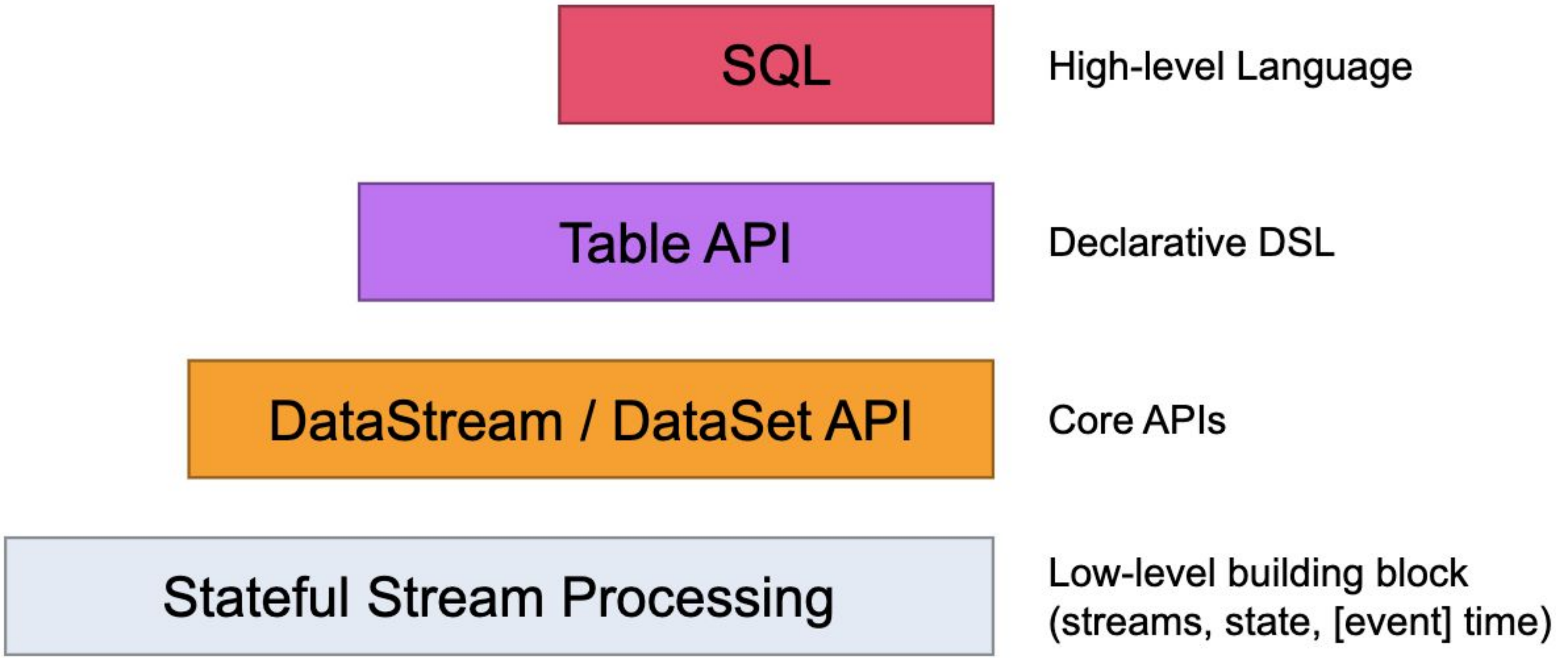
Example time!

An AI + ML pipeline!

An AI + ML pipeline

1. LangChain4j
2. Vector search
3. ????
4. Magic

Flink APIs


SQL
High-level Language
Table API
Declarative DSL
DataStream / DataSet API
Core APIs
Stateful Stream Processing
Low-level building block
(streams, state, [event] time)

Anatomy of a Flink Cluster

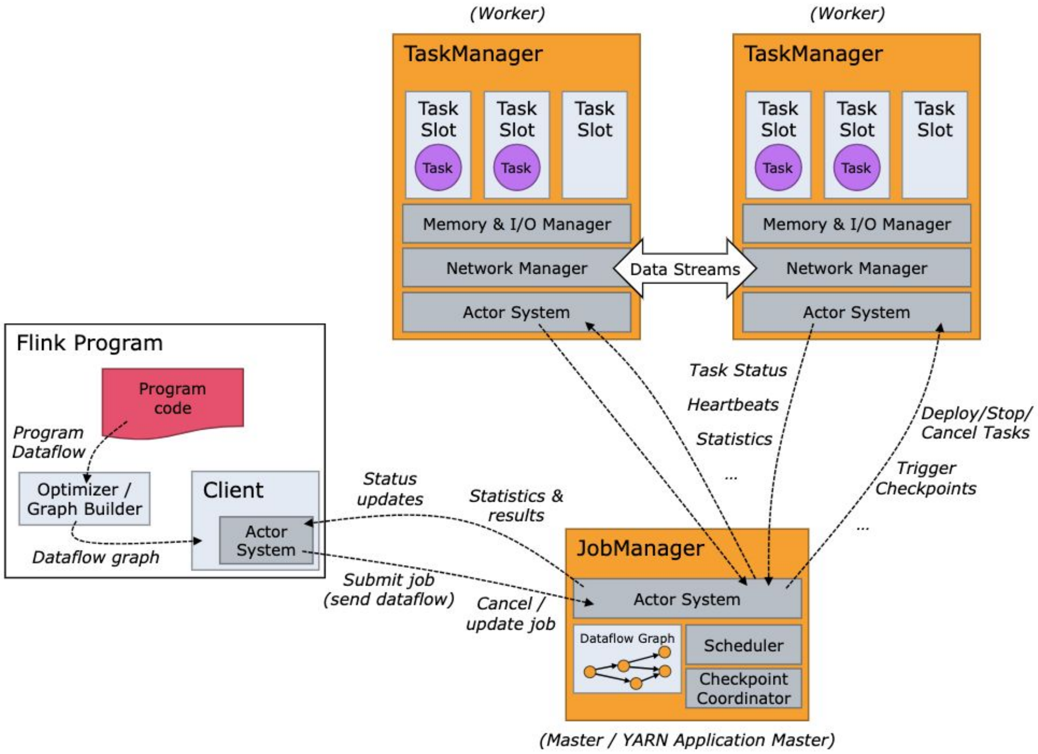


Image: Flink docs
https://nightlies.apache.org/flink/flink-docs-release-1.20/docs/concepts/flink-architecture/

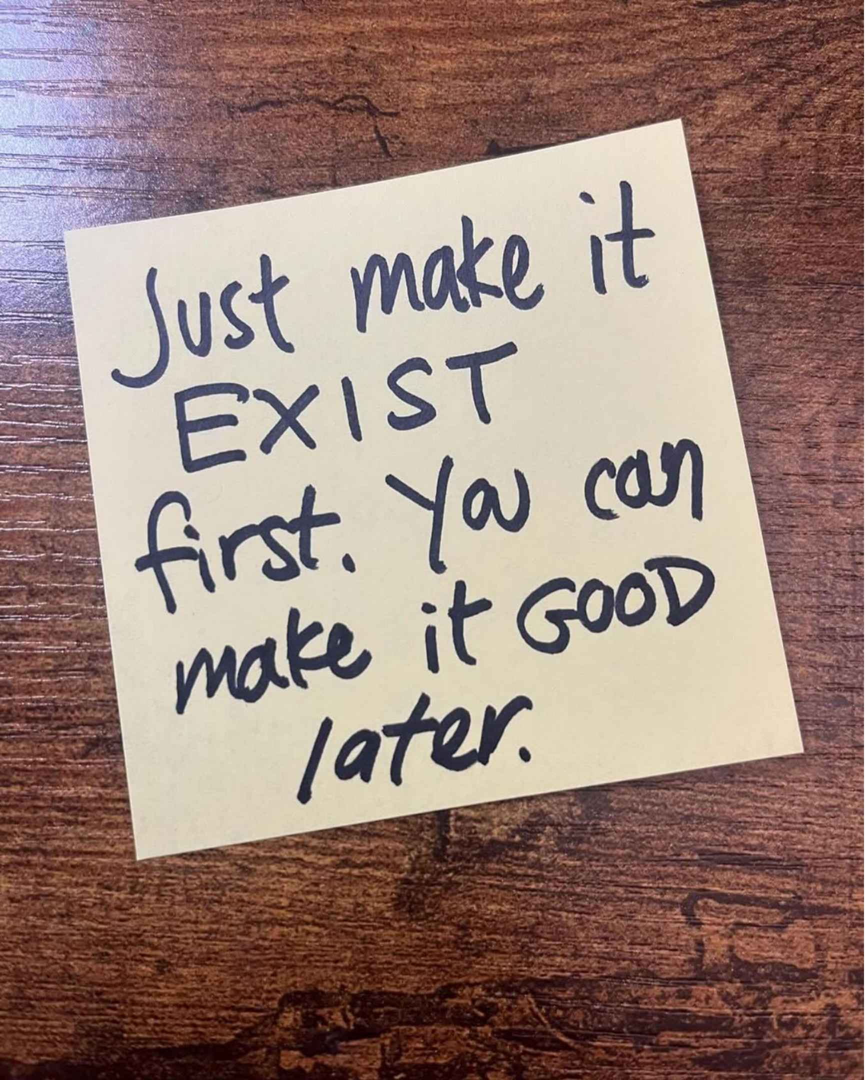
💾 The data

Coinbase data


```
{
    "channel": "ticker",
    "client_id": "",
    "timestamp": "2025-04-27T10:40:21.685011397Z",
    "sequence_num": 75,
    "events": [
        {
            "type": "update",
            "Tickers": []
        }
    ]
}
```

```
{
    "type": "ticker",
      "product_id": "ETH-USD",
      "price": "1807.7",
      "volume_24_h": "71145.59684201",
      "low_24_h": "1779.84",
      "high_24_h": "1857.85",
      "low_52_w": "1383.26",
      "high_52_w": "4109",
      "price_percent_chg_24_h": "-0.1265200360223",
      "best_bid": "1808.05",
      "best_ask": "1808.06",
      "best_bid_quantity": "0.9082549",
      "best_ask_quantity": "0.04469599"
}
```

▶️ Run it



Thank you!