Welcome to Stream Processing with Apache Kafka

UDACITY

Welcome Message

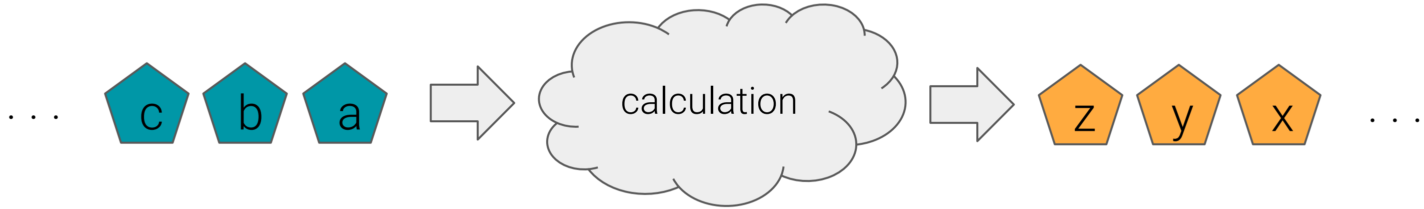
Stream Processing with Apache Kafka

Understanding Stream Processing

What is Stream Processing?

A **stream** is a potentially unbounded sequence of data

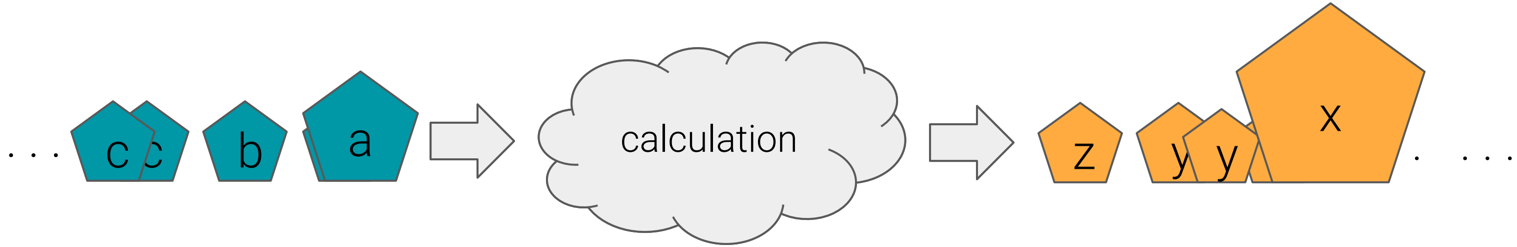
Stream Processing is the act of performing continual calculations on a stream



What is Stream Processing?

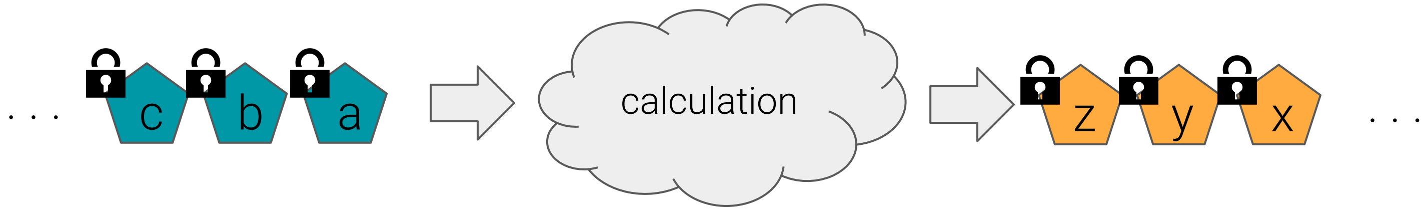
Data may be produced at an **even** or **constant** rate

Data may also be produced **unevenly** and in **different shapes and sizes**



What is Stream Processing?

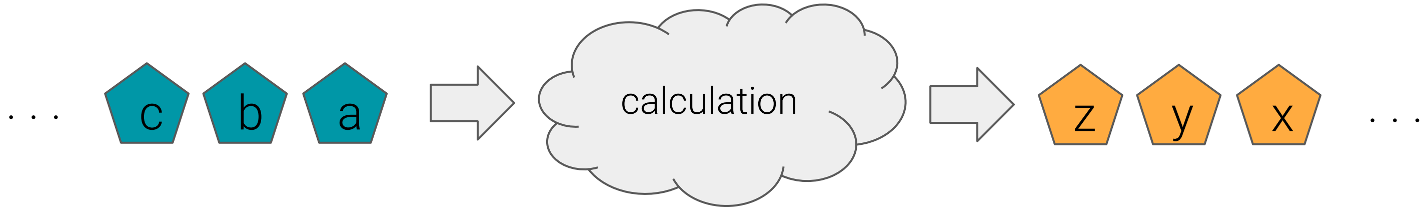
Data streams are made up of **immutable data.** Data cannot be changed once in the stream.



What is Stream Processing?

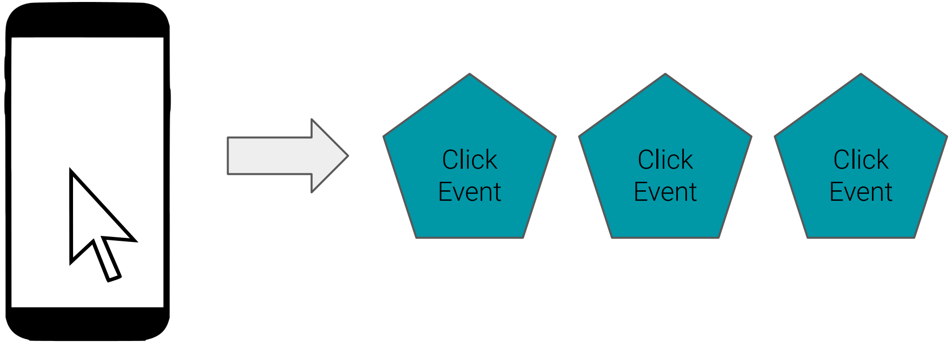
Data records in streams is typically **small**, usually **less than 1MB**

Data **throughput** may range from **one record** per second up to **many thousands** per second



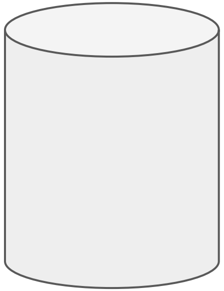
What is an event?

An **event** is an **immutable fact** regarding something that occurred within our system



What is an event?

SQL Databases are built to **store the state** of an application **at that point in time**



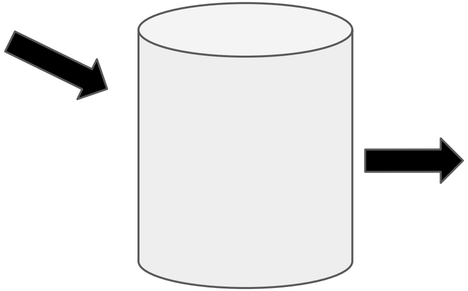
user	
id	favorite_color
123	**“brown”**

What is an event?

SQL Databases are built to **store the state** of an application **at that point in time**

1

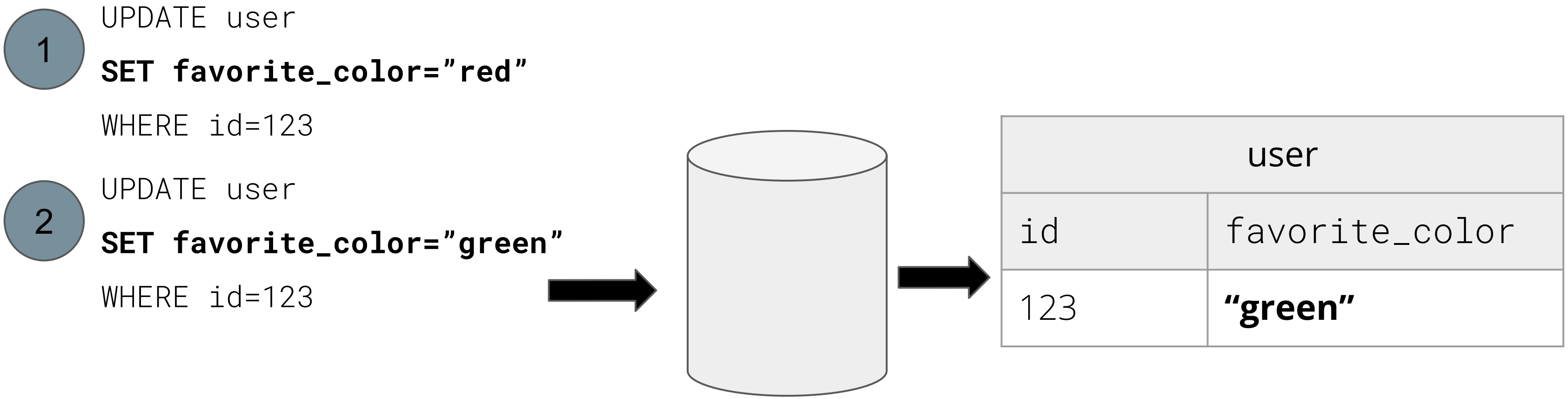
```
UPDATE user
SET favorite_color=”red”
WHERE id=123
```



user	
id	favorite_color
123	**“red”**

What is an event?

SQL Databases are built to **store the state** of an application **at that point in time**



1

```
UPDATE user
SET favorite_color=”red”
WHERE id=123
```

2

```
UPDATE user
SET favorite_color=”green”
WHERE id=123
```

user	
id	favorite_color
123	**“green”**

What is an event?

SQL Databases are built to **store the state** of an application **at that point in time**

1

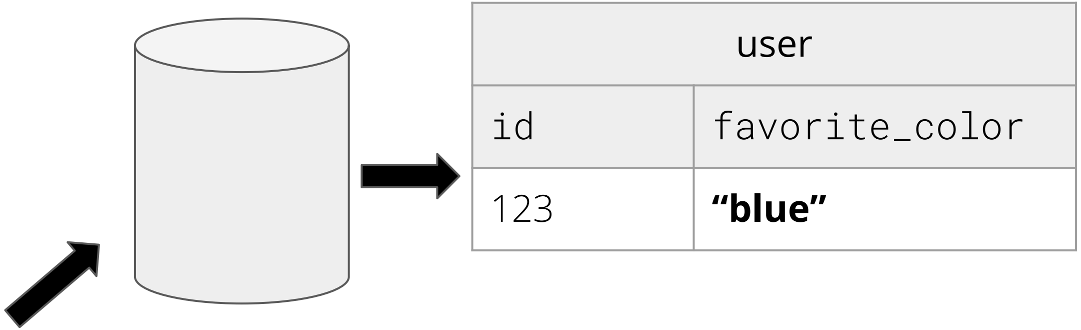
```
UPDATE user
SET favorite_color=”red”
WHERE id=123
```

2

```
UPDATE user
SET favorite_color=”green”
WHERE id=123
```

3

```
UPDATE user
SET favorite_color=”blue”
WHERE id=123
```

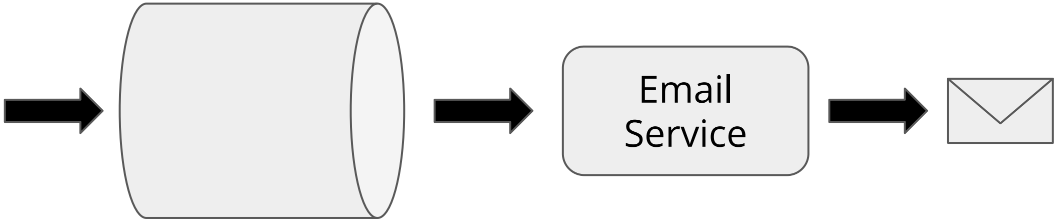


user	
id	favorite_color
123	**“blue”**

What is an event?

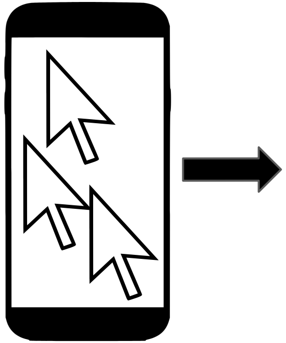
Traditional **Message Queues** are used to **communicate commands** to **perform an action**

```
{
  “command”: “send_email”,
  “email”: “bob@email.com”
}
```



What is an event?

Evented systems **react** to the facts communicated to them. The communication is **indirect** and the events they utilize are usually not **specifically targeted** to any one system.



```
{
  “action”: “click”,
  “element”: “search”
}
```

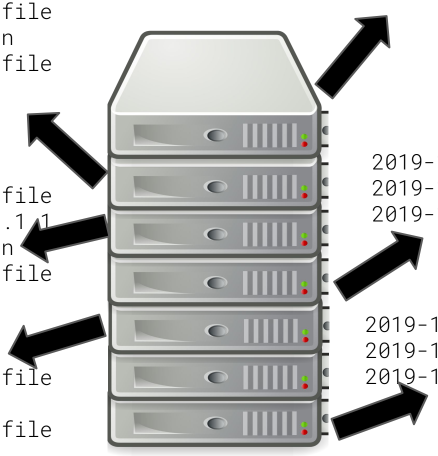
```
{
  “action”: “click”,
  “element”: “view”
}
```

```
{
  “action”: “click”,
  “element”: “checkout”
}
```

Examples of Stream Processing

Example: Log Analysis

Logs are hard to process in batch systems due to the speed and size of data



```
2019-10-08 file.go:23 exception | failed opening file
2019-10-08 ormd.go:11 warn      | long transaction
2019-10-08 file.go:70 exception | failed opening file
```

```
2019-10-08 file.go:23 exception | failed opening file
2019-10-08 teld.go:20 info      | dialing 192.168.1.1
2019-10-08 ormd.go:11 warn      | long transaction
2019-10-08 file.go:70 exception | failed opening file
2019-10-08 webp.go:99 info      | opening socket
```

```
2019-10-08 file.go:23 exception | failed opening file
2019-10-08 webp.go:99 info      | opening socket
2019-10-08 file.go:70 exception | failed opening file
```

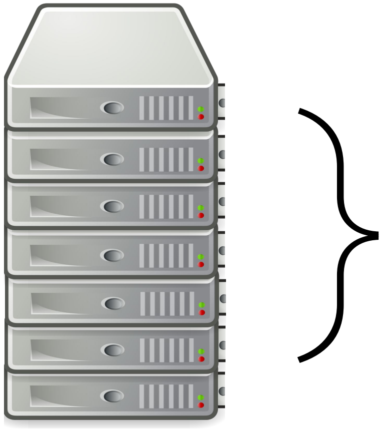
```
2019-10-08 file.go:23 exception | failed opening file
2019-10-08 teld.go:20 info      | dialing 192.168.1.1
2019-10-08 ormd.go:11 warn      | long transaction
2019-10-08 file.go:70 exception | failed opening file
2019-10-08 webp.go:99 info      | opening socket
```

```
2019-10-08 file.go:23 exception | failed opening file
2019-10-08 webp.go:99 info      | opening socket
2019-10-08 file.go:70 exception | failed opening file
```

```
2019-10-08 file.go:23 exception | failed opening file
2019-10-08 ormd.go:11 warn      | long transaction
2019-10-08 file.go:70 exception | failed opening file
```

Example: Log Analysis

Companies push **log data** as **events** into a **data stream** to perform stream processing.

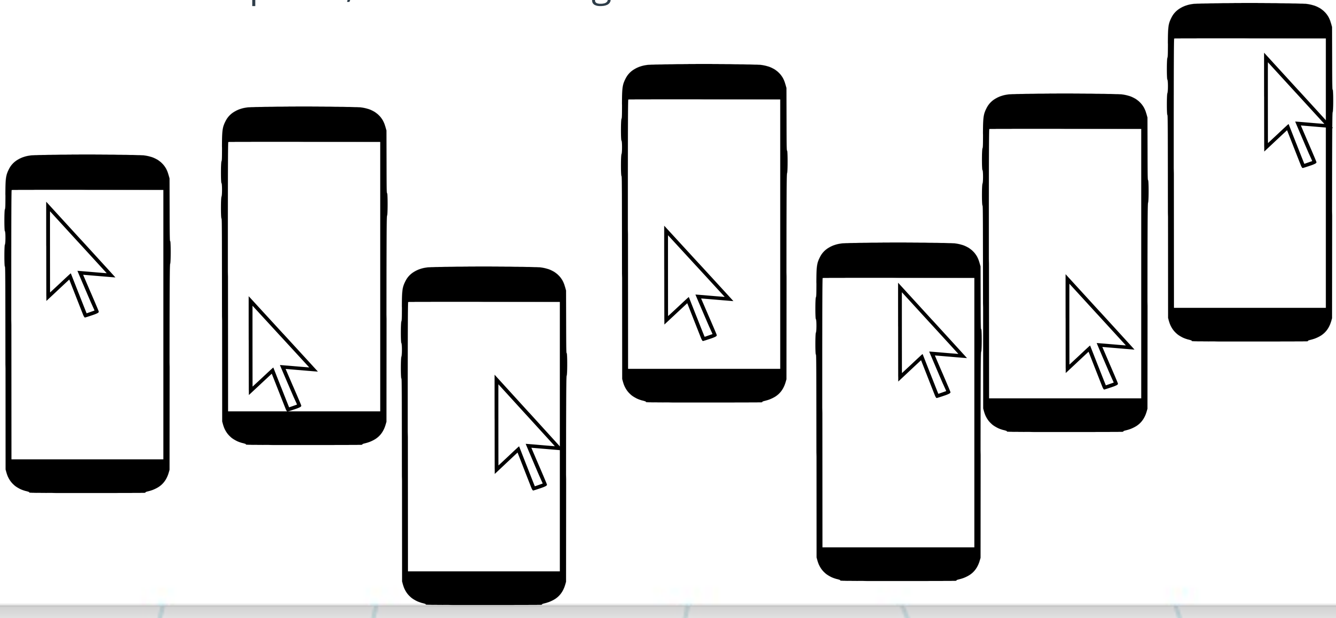


```
2019-10-08 file.go:23 exception | failed opening file
2019-10-08 teld.go:20 info      | dialing 192.168.1.1
2019-10-08 file.go:23 exception | failed opening file
2019-10-08 webp.go:99 info      | opening socket
2019-10-08 ormd.go:11 warn      | long transaction
2019-10-08 ormd.go:11 warn      | long transaction
2019-10-08 teld.go:20 info      | dialing 10.0.0.1
2019-10-08 teld.go:20 info      | dialing 172.168.0.1
```

Example: Web Analytics

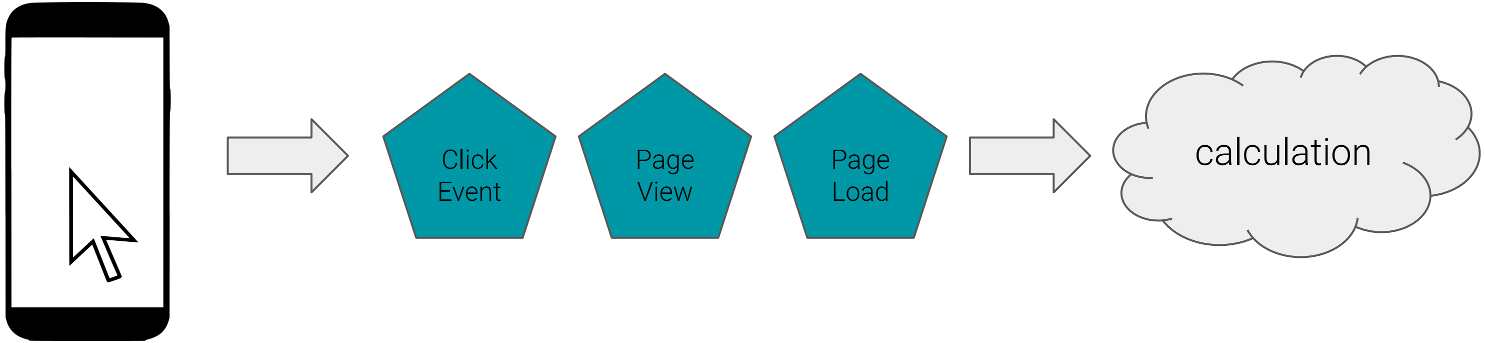
Modern web applications track user actions such as **clicks** and **page views**.

These actions add up fast, overwhelming traditional data stores.



Example: Web Analytics

Stream processing allows companies to process data **as it's generated** and not hours after the fact as is common with batch processing.

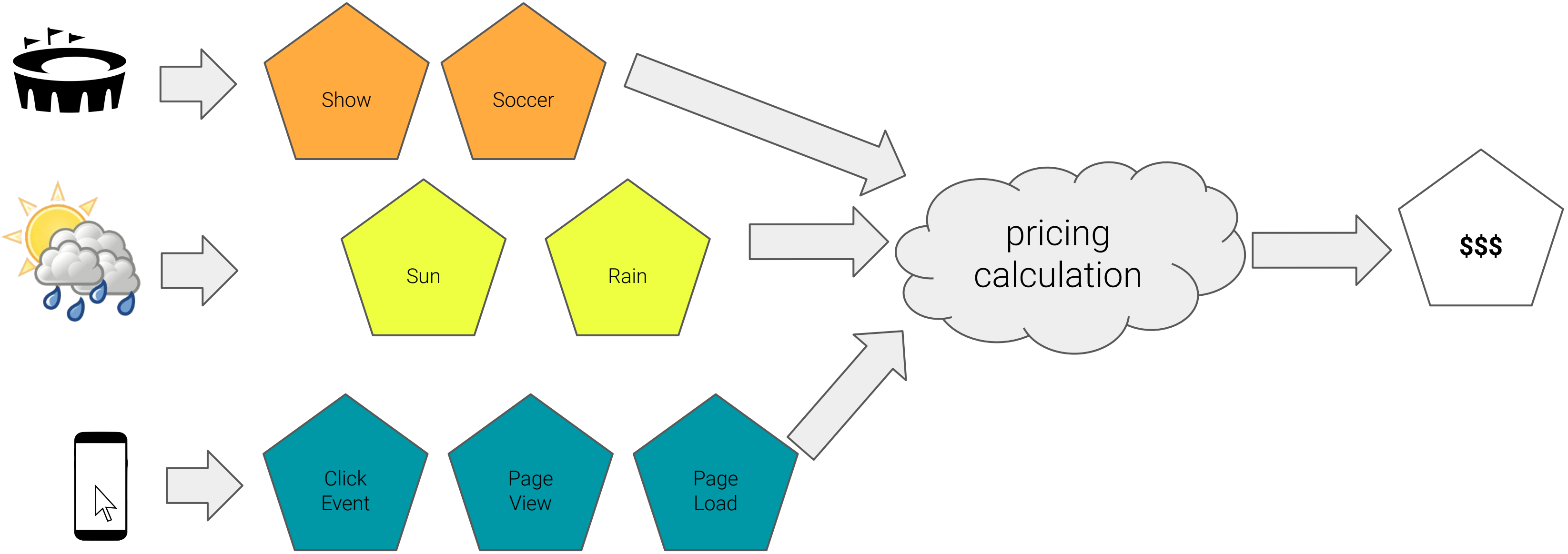


Example: Real-Time Pricing

Real Time Pricing adjusts to **environmental** factors and **instantaneous demand**

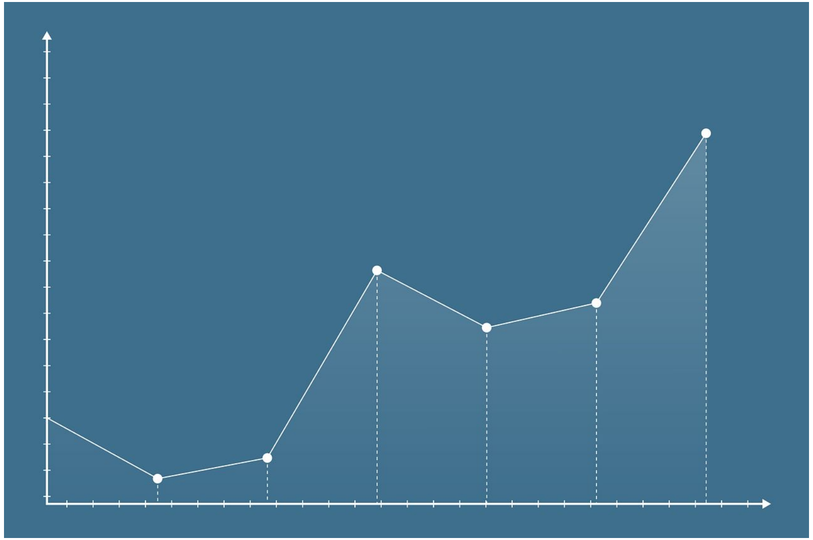
Example: Real-Time Pricing

Real Time Pricing adjusts to **environmental** factors and **instantaneous demand**



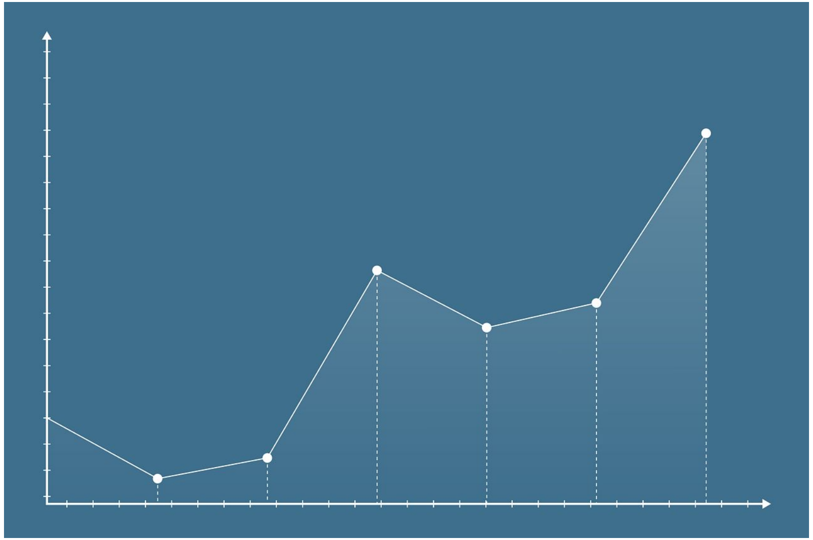
Example: Financial Analysis

Prices for **stocks fluctuate rapidly**, creating huge amounts of data



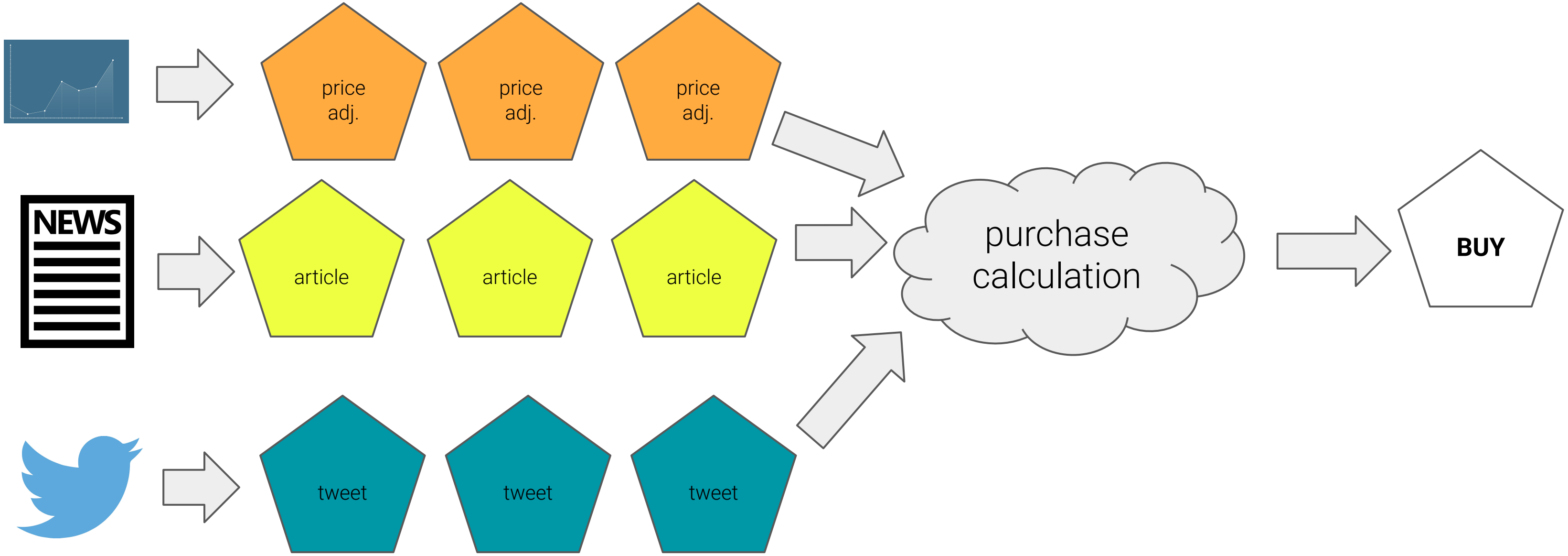
Example: Financial Analysis

Data from many other sources, such as news articles and Twitter influence purchasing



Example: Financial Analysis

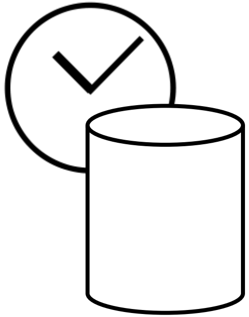
Stream Processing coalesces these data streams into one real-time decision to buy or sell



Differences Between Stream and Batch Processing

x: Contrasting Stream and Batch Processing

Batch Processing



- Scheduled analysis of related groups of data
- As up to date as the last scheduled run
- May run for long periods of time
- Often involve mutable data stores
- May access all historical data

Stream Processing

- Real-time analysis of data as it is produced
- Runs as soon as event is produced
- As up to date as the last event generated
- Often involves immutable data stores
- Often uses recently produced, windowed data

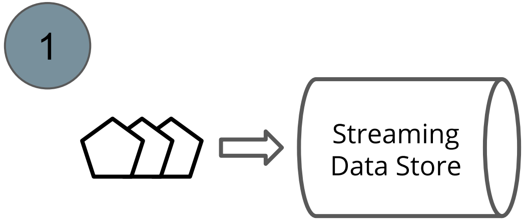
Batch vs Stream Processing

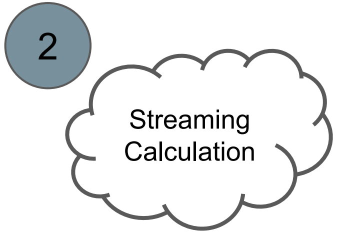
The differences are **generalizations, not hard rules.**

Most data engineering teams use both **batch** and **stream processing together** to achieve their goals.

Components of a Stream Processing Solution

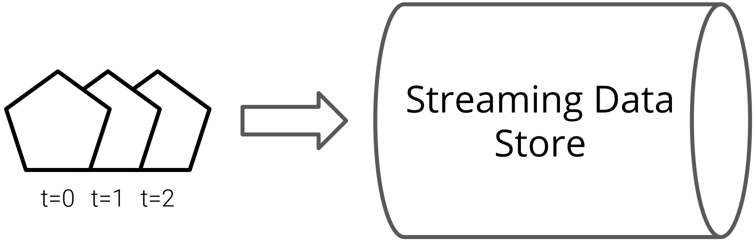
Stream Processing applications consist of a **streaming data store** and the **streaming calculation(s)**.





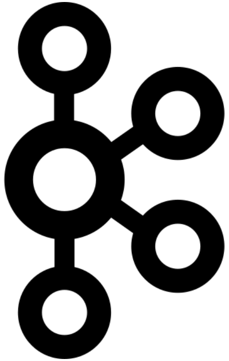
Components of a Stream Processing Solution

Streaming data stores **hold all of the immutable event data** in a system. These data stores guarantee that **data is stored in the order it was produced**.



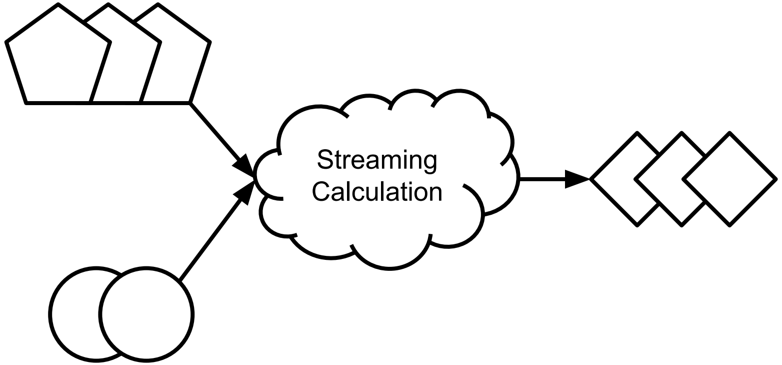
Components of a Stream Processing Solution

Apache **Kafka** and Apache **Cassandra** are examples of streaming data stores.



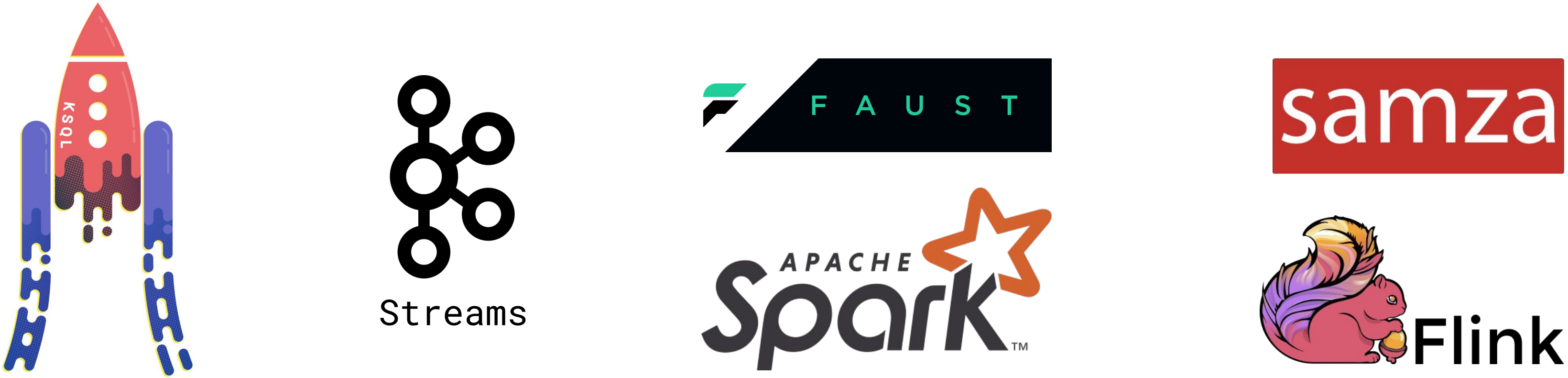
Components of a Stream Processing Solution

Stream processing applications are **downstream** of the data store and perform calculations on one or more data streams, such as **aggregations**, **joins**, and **filtering**. These calculations **produce new data events.**



Components of a Stream Processing Solution

Popular Stream Processing tools, all of which share common features, include **Confluent KSQL, Kafka Streams, Faust, Spark Structured Streaming, Apache Flink, and Samza**.



Stream Processing Benefits

Stream Processing Benefits

- **Faster calculations** on windowed data

Stream Processing Benefits

- **Faster calculations** on windowed data
- More **scalable** due to distributed nature of data stores

Stream Processing Benefits

- **Faster calculations** on windowed data
- More **scalable** due to distributed nature of data stores
- Decouples how data is ***used*** from how it is ***produced***

Stream Processing Benefits

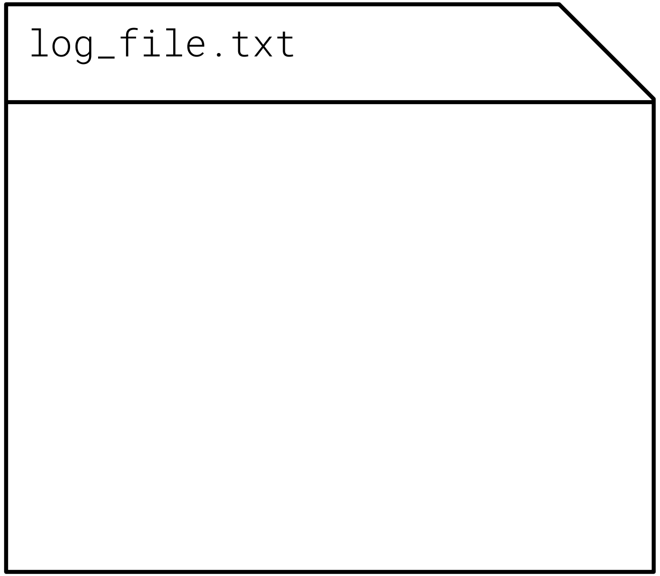
- **Faster calculations** on windowed data
- More **scalable** due to distributed nature of data stores
- Decouples how data is ***used*** from how it is ***produced***
- Immutable data provides pipeline **repeatability**

Append-Only Logs

Append-Only Logs

Append-Only Logs

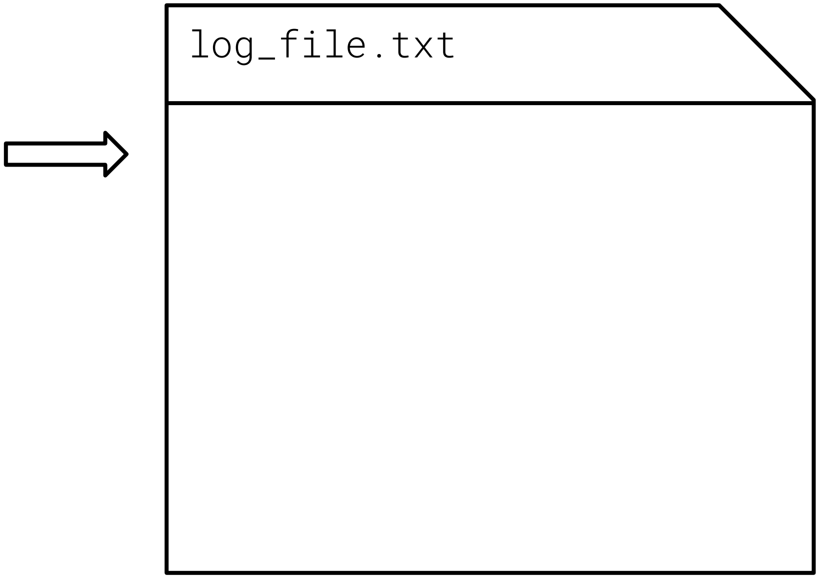
Append-only logs are files which **append events to the end as they arrive**



Append-Only Logs

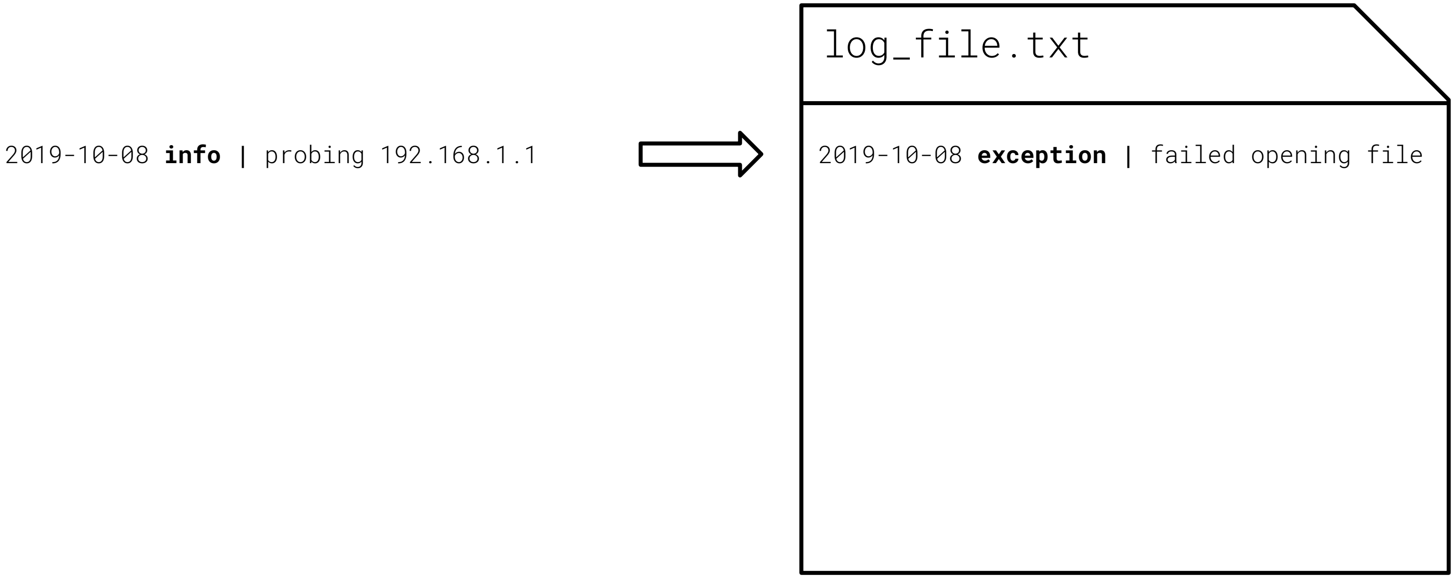
Append-only logs are files which **append events to the end as they arrive**

```
2019-10-08 exception | failed opening file
```



Append-Only Logs

Append-only logs are files which **append events to the end as they arrive**



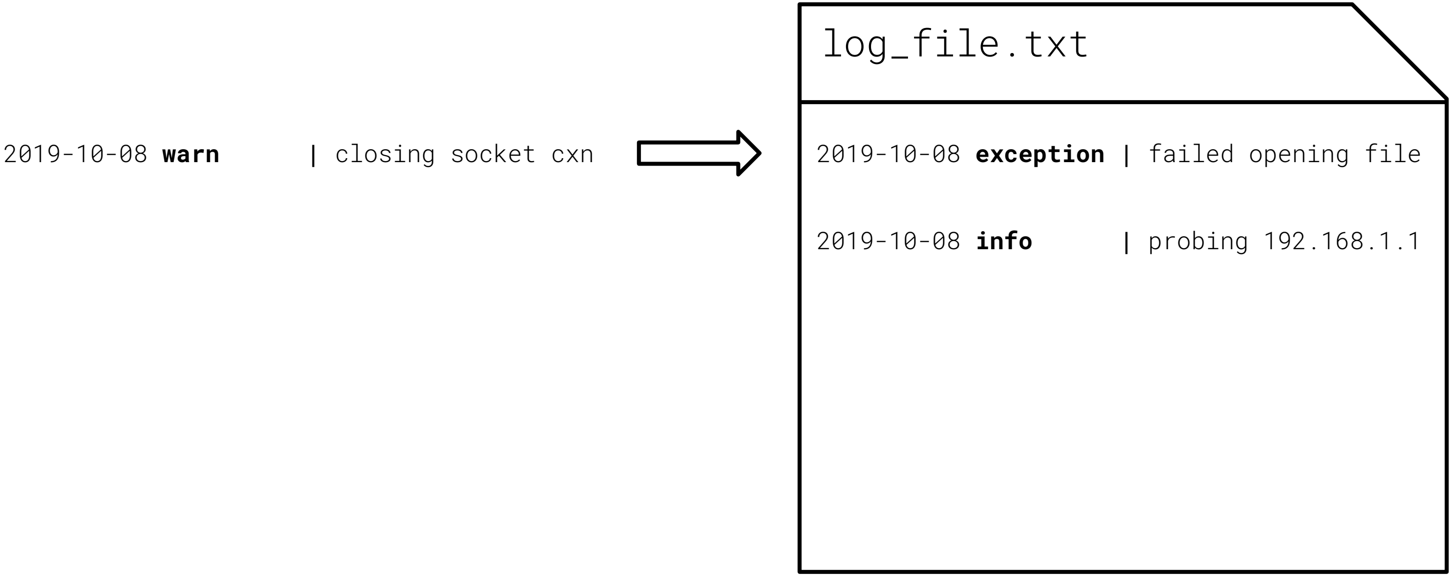
```
2019-10-08 info | probing 192.168.1.1
```

log_file.txt

```
2019-10-08 exception | failed opening file
```

Append-Only Logs

Append-only logs are files which **append events to the end as they arrive**

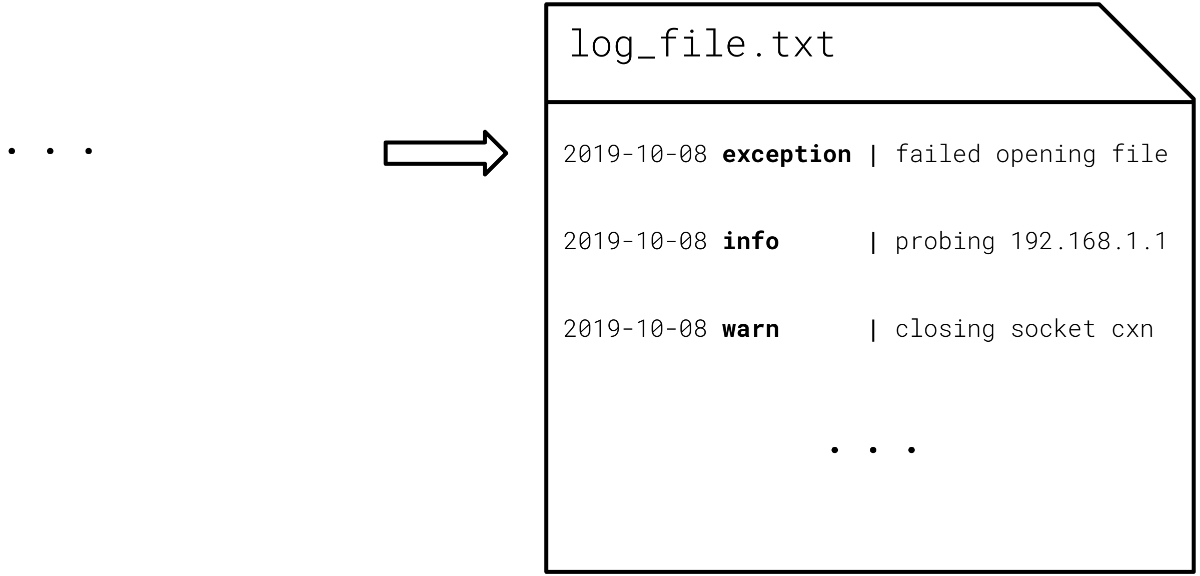


```
2019-10-08 warn      | closing socket cxn
```

```
2019-10-08 exception | failed opening file

2019-10-08 info      | probing 192.168.1.1
```

Append-Only Logs

Append-only logs are files which **append events to the end as they arrive**



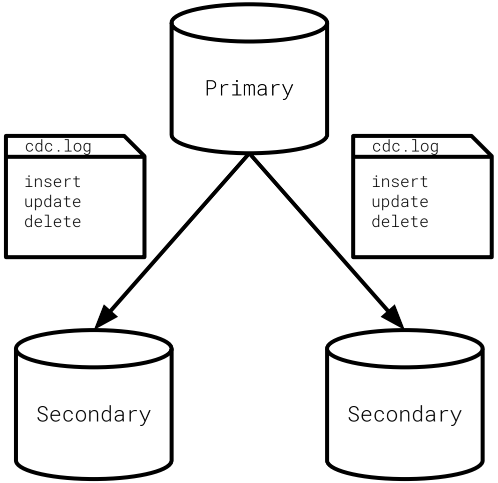
log_file.txt

```
2019-10-08 exception | failed opening file
2019-10-08 info      | probing 192.168.1.1
2019-10-08 warn      | closing socket cxn
```

Append-Only Logs in SQL Databases

Append-Only Logs in SQL Databases

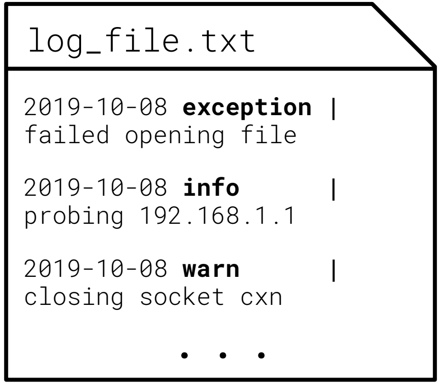
SQL Databases **use append-only logs** to communicate and synchronize changes in a process known as **Change Data Capture (CDC)**.



Log-structured storage

Log-structured storage

Characteristics

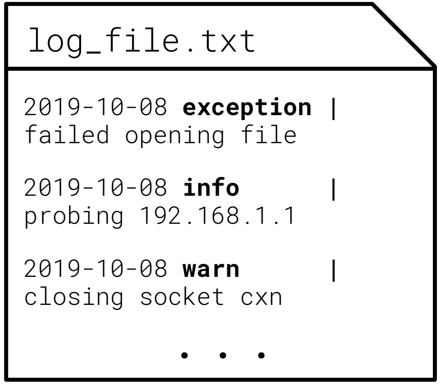


```
log_file.txt

2019-10-08 exception |
failed opening file

2019-10-08 info      |
probing 192.168.1.1

2019-10-08 warn      |
closing socket cxn

          . . .
```

Log-structured storage

Characteristics

- Consist of many **append-only logs** on disk



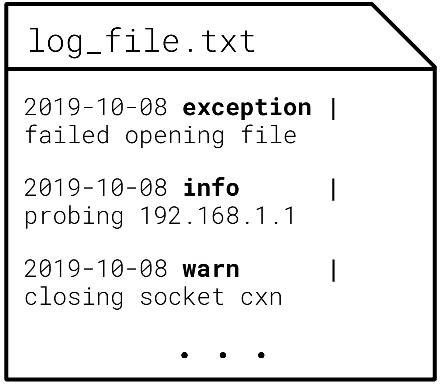
log_file.txt

```
2019-10-08 exception |
failed opening file

2019-10-08 info      |
probing 192.168.1.1

2019-10-08 warn      |
closing socket cxn

           . . .
```

Log-structured storage

Characteristics

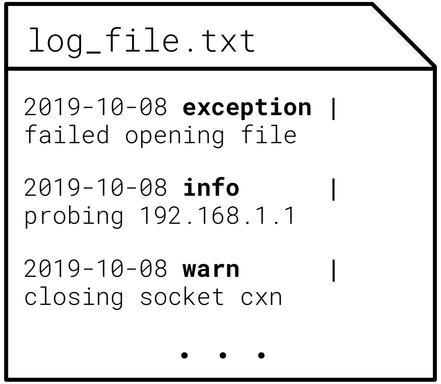
- Consist of many **append-only logs** on disk
- Files are periodically **merged**, or **joined together** into one file



```
log_file.txt

2019-10-08 exception |
failed opening file

2019-10-08 info      |
probing 192.168.1.1

2019-10-08 warn      |
closing socket cxn

        . . .
```

Log-structured storage

Characteristics



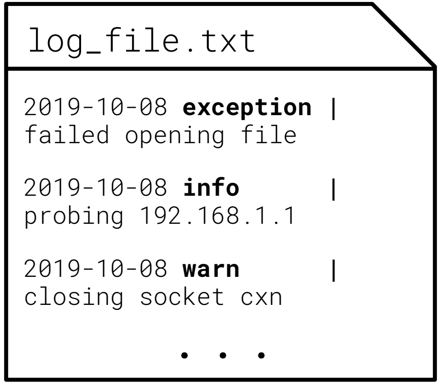
log_file.txt

```
2019-10-08 exception |
failed opening file

2019-10-08 info      |
probing 192.168.1.1

2019-10-08 warn      |
closing socket cxn

          . . .
```

- Consist of many **append-only logs** on disk
- Files are periodically **merged**, or **joined together** into one file
- Files are periodically **compacted**, where one or more files is **deleted,** typically **based on age**

Log-structured storage

Characteristics



```
log_file.txt

2019-10-08 exception |
failed opening file

2019-10-08 info      |
probing 192.168.1.1

2019-10-08 warn      |
closing socket cxn

          . . .
```

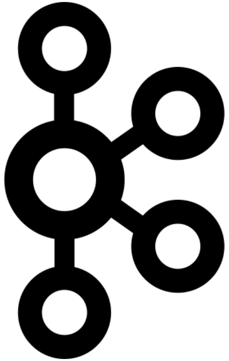
- Consist of many **append-only logs** on disk
- Files are periodically **merged**, or **joined together** into one file
- Files are periodically **compacted**, where one or more files is **deleted,** typically **based on age**
- Use many log files, instead of just one, which **increases speed** and reduces I/O **bottlenecks**

Examples of Log-Structured Storage

Cassandra & HBase

- Cassandra and HBase provide **SQL-like** interfaces
- Use **append-only, log-structured streams**
- Look and act like traditional SQL database to end user
- Clusters may consist of thousands of distributed nodes
- Popular for **batch workloads**

Examples of Log-Structured Storage

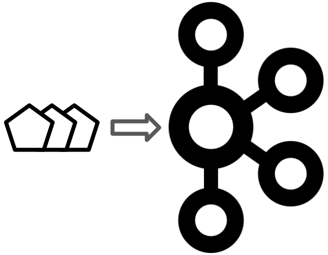


- Apache **Kafka** is a **message queue** based on log-structured, append-only storage
- Scales to thousands of distributed nodes
- Popular for **Stream Processing**

Apache Kafka as a Stream Processing Tool

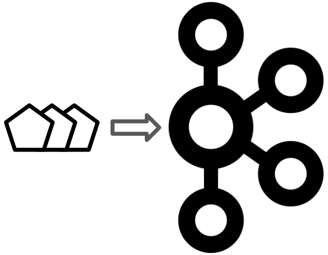
Apache Kafka as a Stream Processing Tool

- Kafka **stores events,** not actions or jobs

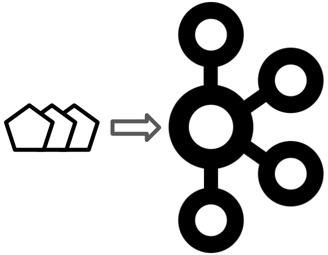


Apache Kafka as a Stream Processing Tool

- Kafka **stores events,** not actions or jobs
- Data is **distributed** to multiple nodes **by default**

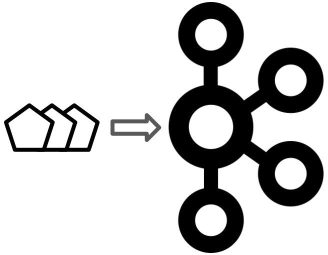


Apache Kafka as a Stream Processing Tool



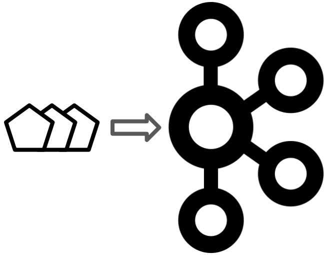
- Kafka **stores events,** not actions or jobs
- Data is **distributed** to multiple nodes **by default**
- **Highly scalable** with **strict** data **ordering guarantees**

Apache Kafka as a Stream Processing Tool



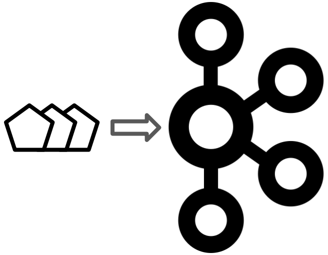
- Kafka **stores events,** not actions or jobs
- Data is **distributed** to multiple nodes **by default**
- **Highly scalable** with **strict** data **ordering guarantees**
- **Fault-tolerant** to node loss due to distributed nature

Apache Kafka as a Stream Processing Tool



- Kafka **stores events,** not actions or jobs
- Data is **distributed** to multiple nodes **by default**
- **Highly scalable** with **strict** data **ordering guarantees**
- **Fault-tolerant** to node loss due to distributed nature
- Created at **LinkedIn**, now maintained by **Confluent** as an open source product

Apache Kafka as a Stream Processing Tool



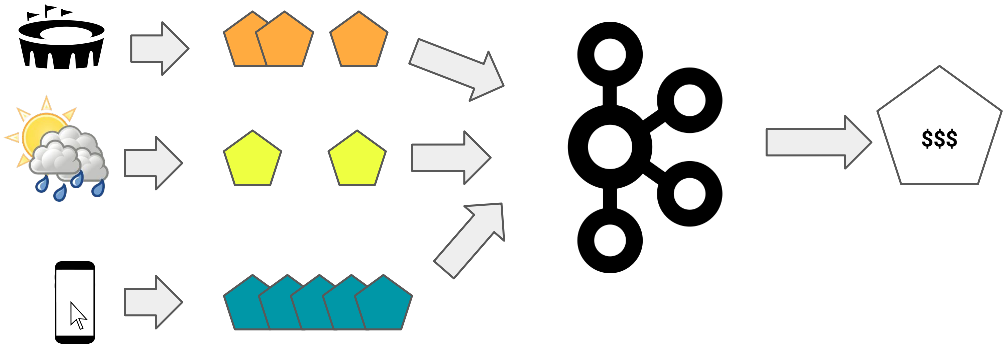
- Kafka **stores events,** not actions or jobs
- Data is **distributed** to multiple nodes **by default**
- **Highly scalable** with **strict** data **ordering guarantees**
- **Fault-tolerant** to node loss due to distributed nature
- Created at **LinkedIn**, now maintained by **Confluent** as an open source product
- Streaming data store for **Flink, Spark,** and **Samza**

Industry Use-Cases

Uber, Netflix, and Segment

Industry Use-Cases

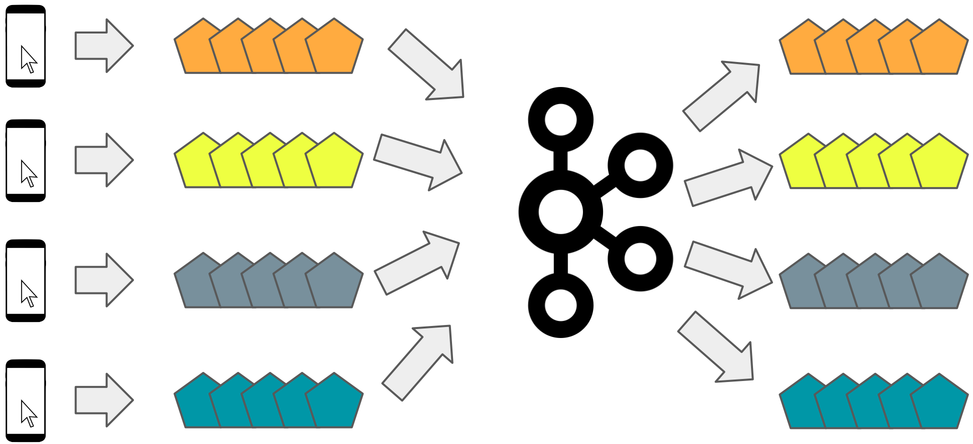
Uber uses Kafka extensively in their **real-time pricing** pipeline.



Industry Use-Cases

Segment uses Kafka as their **routing backbone** for analytics events

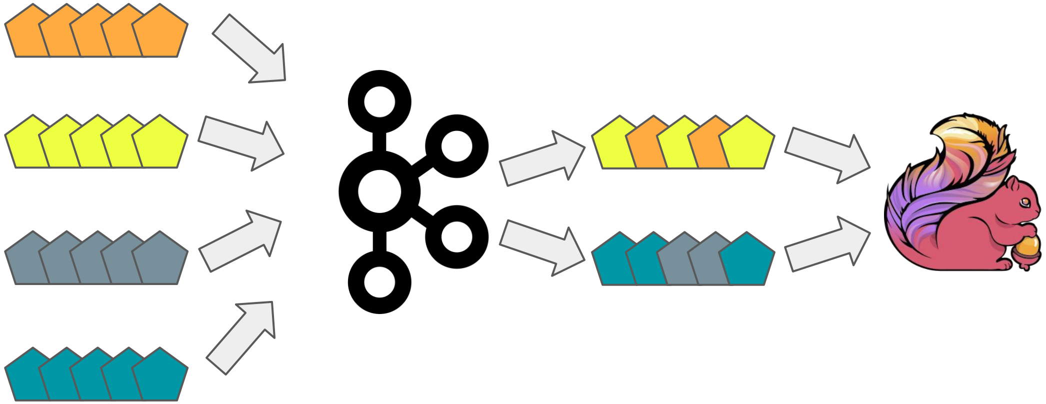
Segment



Industry Use-Cases

As part of Netflix's **Keystone**, Kafka handles billions of events a day

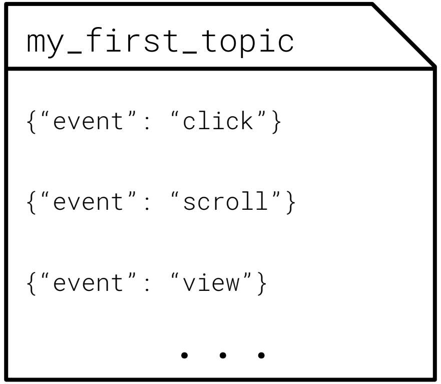
NETFLIX



Kafka in Action

Topics, Consumers, and Producers

Kafka Topics



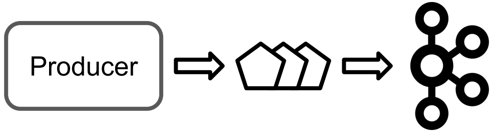
my_first_topic

```
{“event”: “click”}

{“event”: “scroll”}

{“event”: “view”}

. . .
```

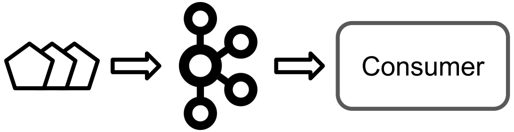
- Kafka Topics are roughly analogous to SQL Tables
- Topics **organize related events**
- Topics are **not queryable** like SQL tables
- Topics consist of append-only logs
- All **data** in a topic is in **key-value form**
- Many configuration options and considerations

Kafka Producers

- Kafka **Producers** send data to a topic
- Producers are often built with a client library
- Events may be sent one at a time, or in batches
- Many configuration options and considerations



Kafka Consumers



- **Consumers** retrieve events from Kafka Topic(s)
- May consume one or more topics at a time
- Keep track of what events have already been seen
- Many configuration options and considerations

Demo: Using Kafka CLI (Command Line Interface) tools to Produce and Consume

Demo: Using Python to Produce and Consume Events