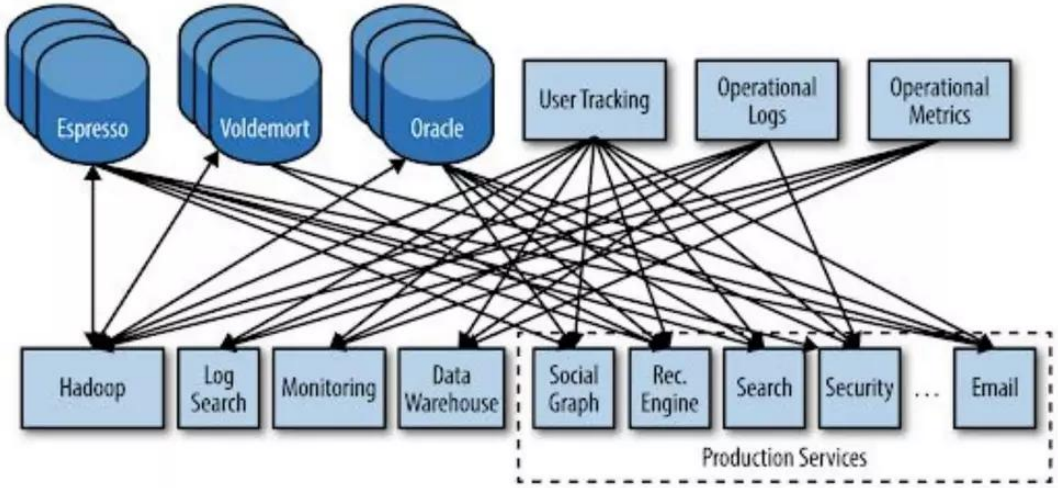
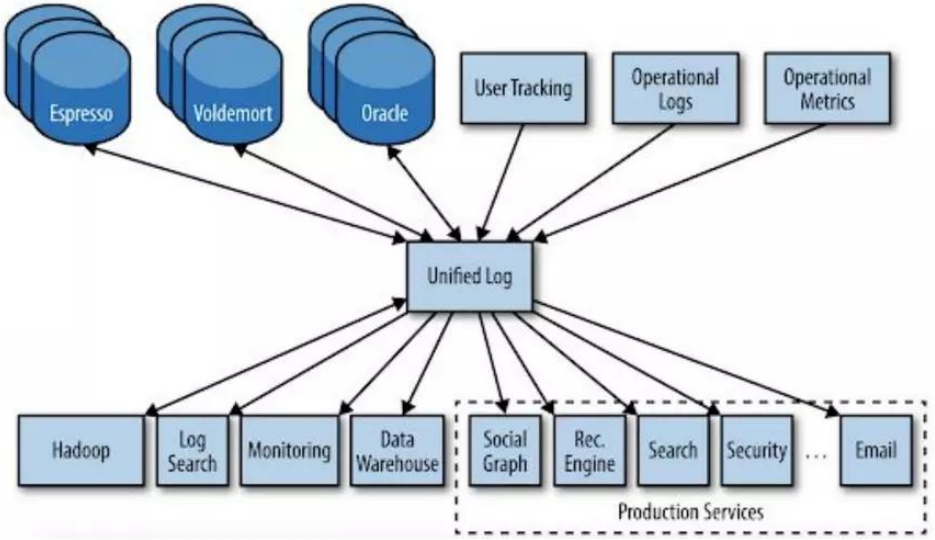
Kafka 102 - Applied

Alex Woolford, Systems engineer

why Kafka was created





Source: I ♥ logs, Jay Kreps, 2014

Cuatro Commas Club ,,,,

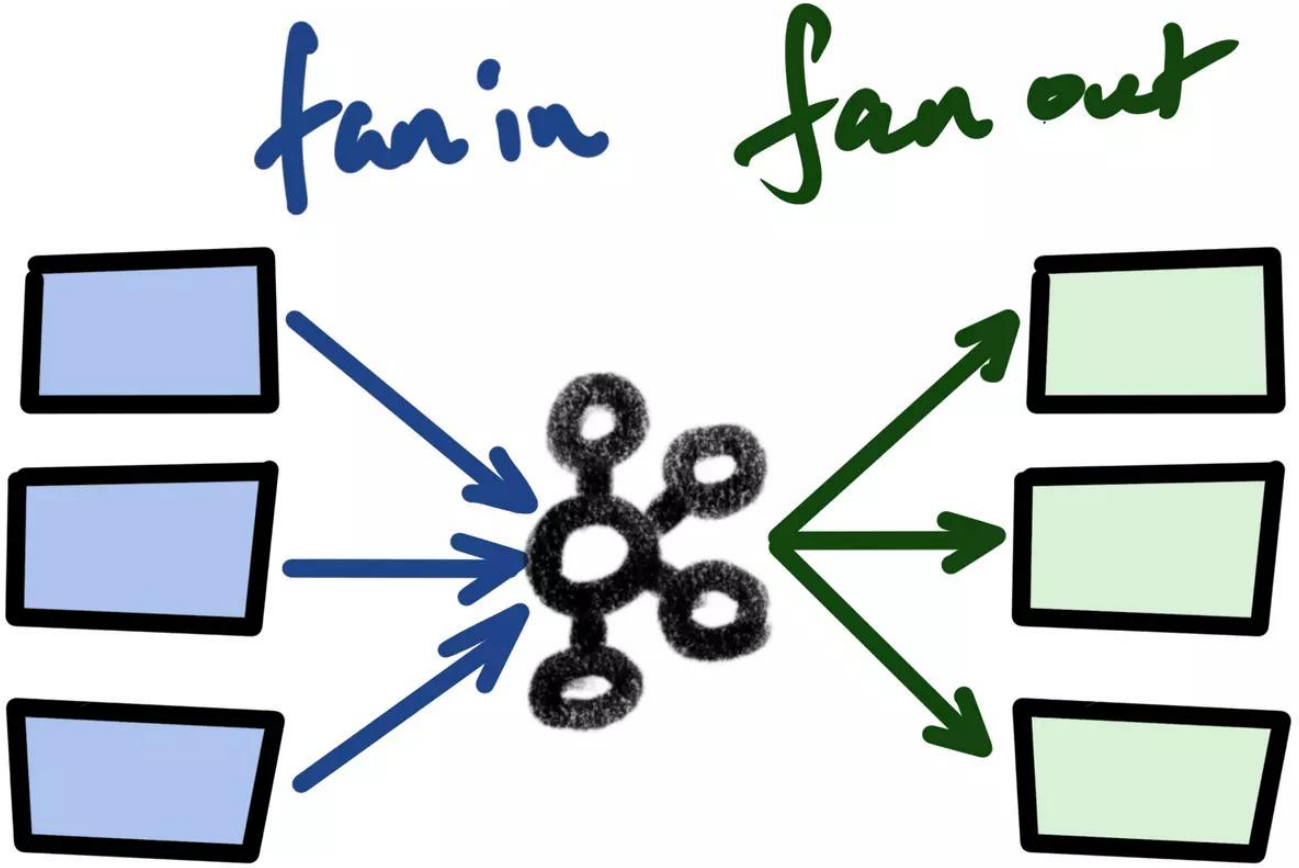
LinkedIn
4.5 trillion messages/day

Uber
1+ trillion messages/day

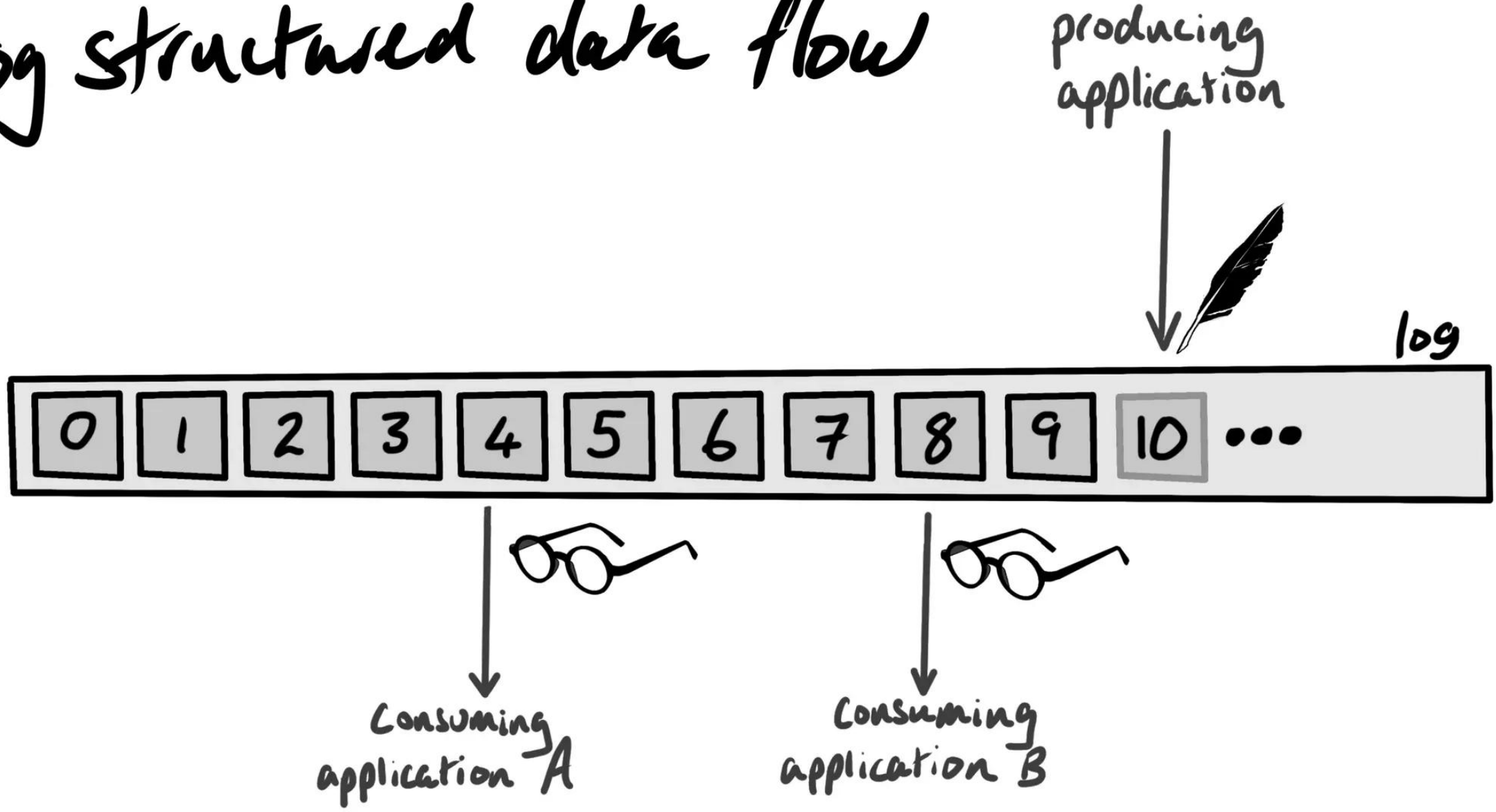
NETFLIX
2 trillion messages/day

Microsoft
3 trillion messages/day

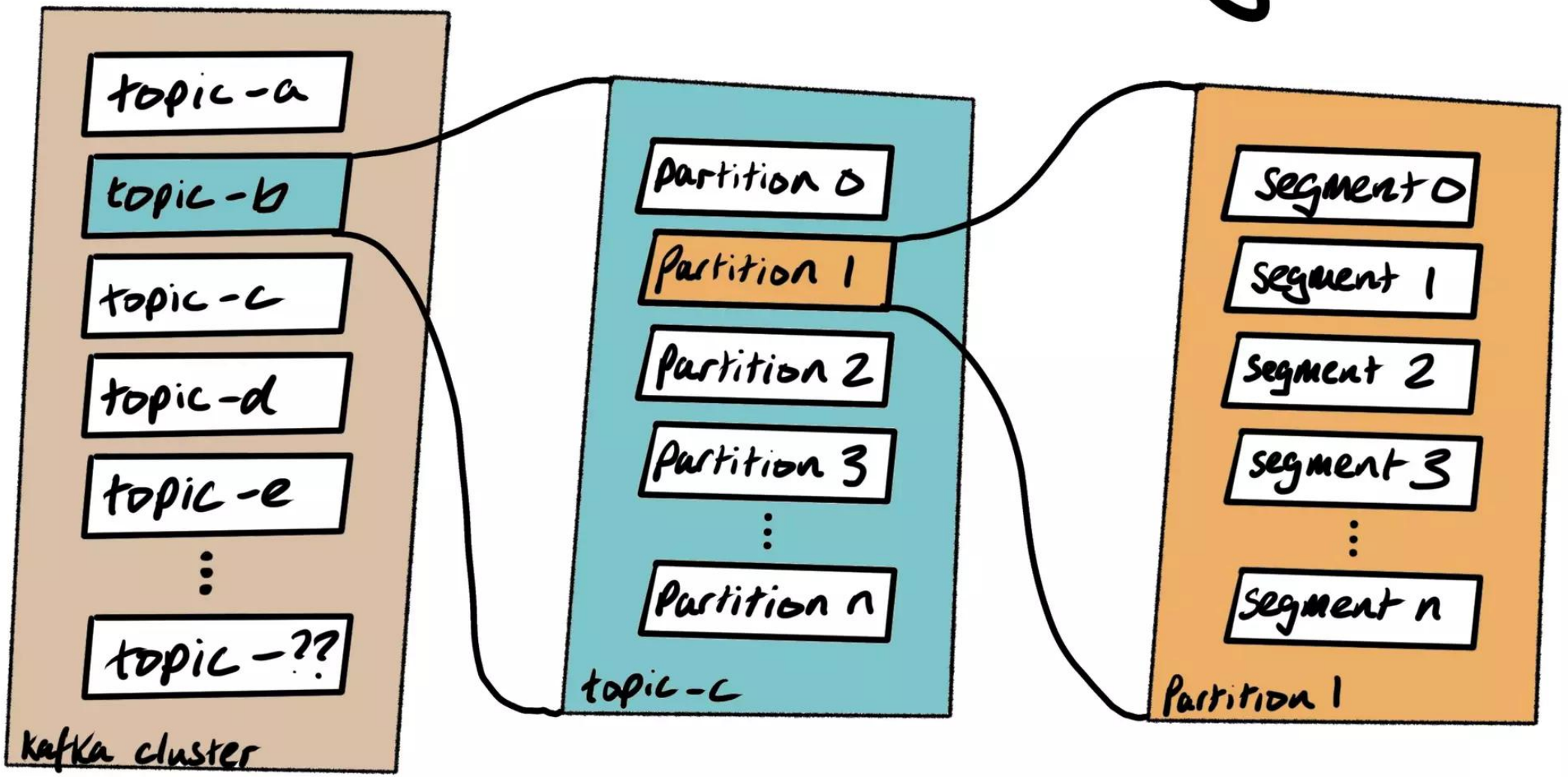
Kafka ♥'s fan out



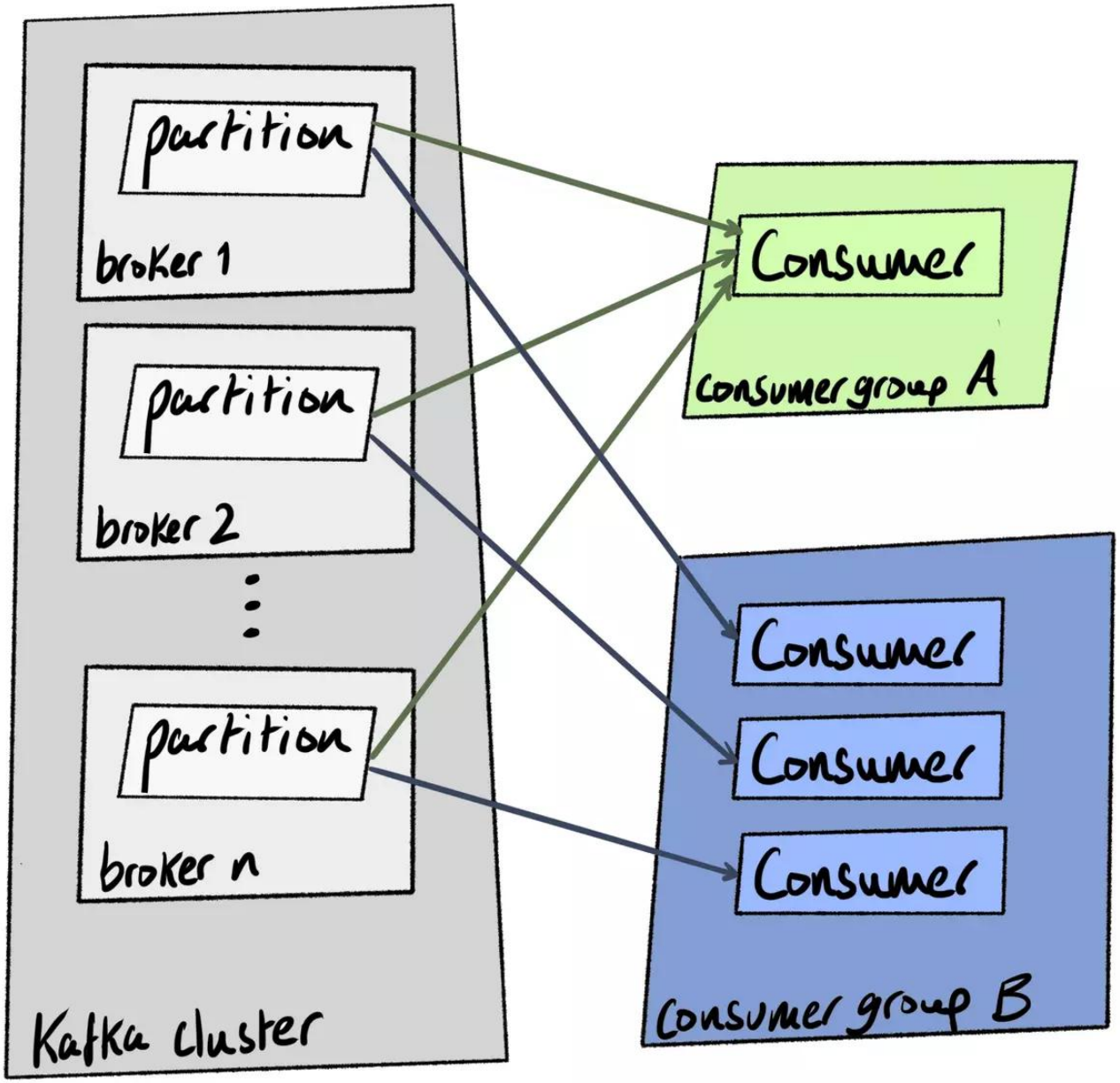
log structured data flow



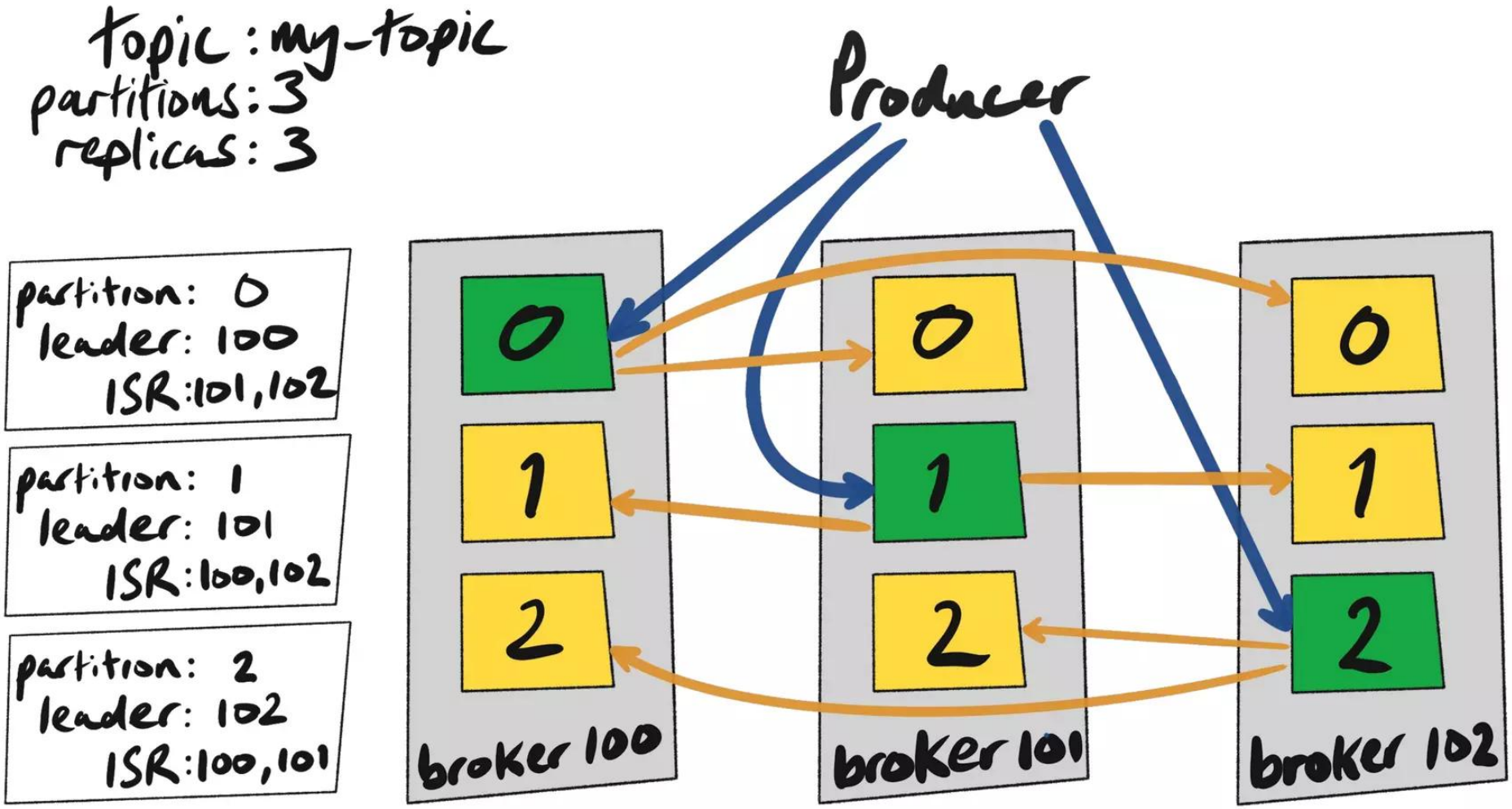
topics, partitions, and segments



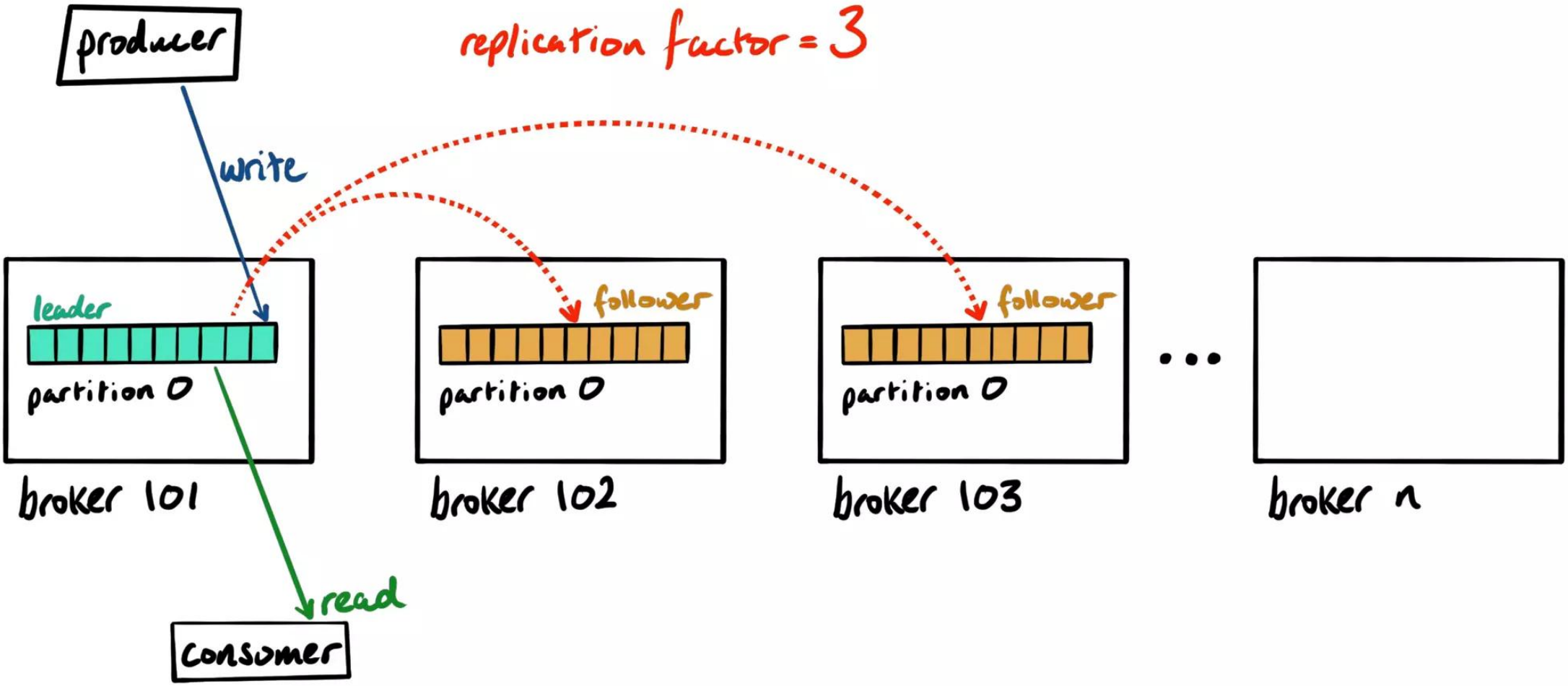
Consumer groups



replication and in-sync replicas (ISR)



replication factor = 3



partitioning

why partition?

- load balancing
- semantic partitioning

out of the box behavior:

- with key: hash(key) % number of partitions
- no key: round robin

or write your own custom partioner

(you might do this if you have a composite key)

Kafka configs

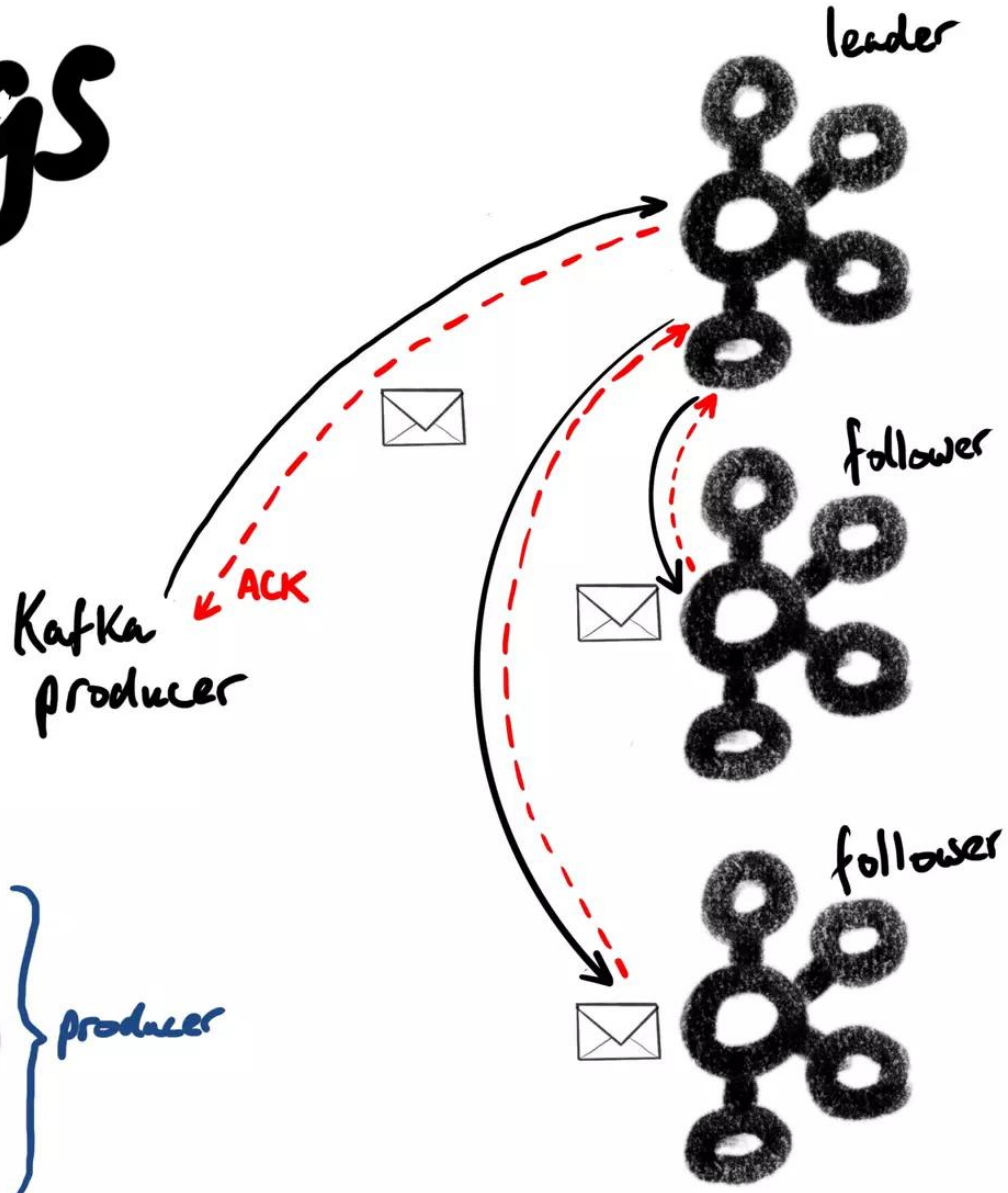
Typical configs:

- replication.factor : 3
- min.insync.replicas : 2

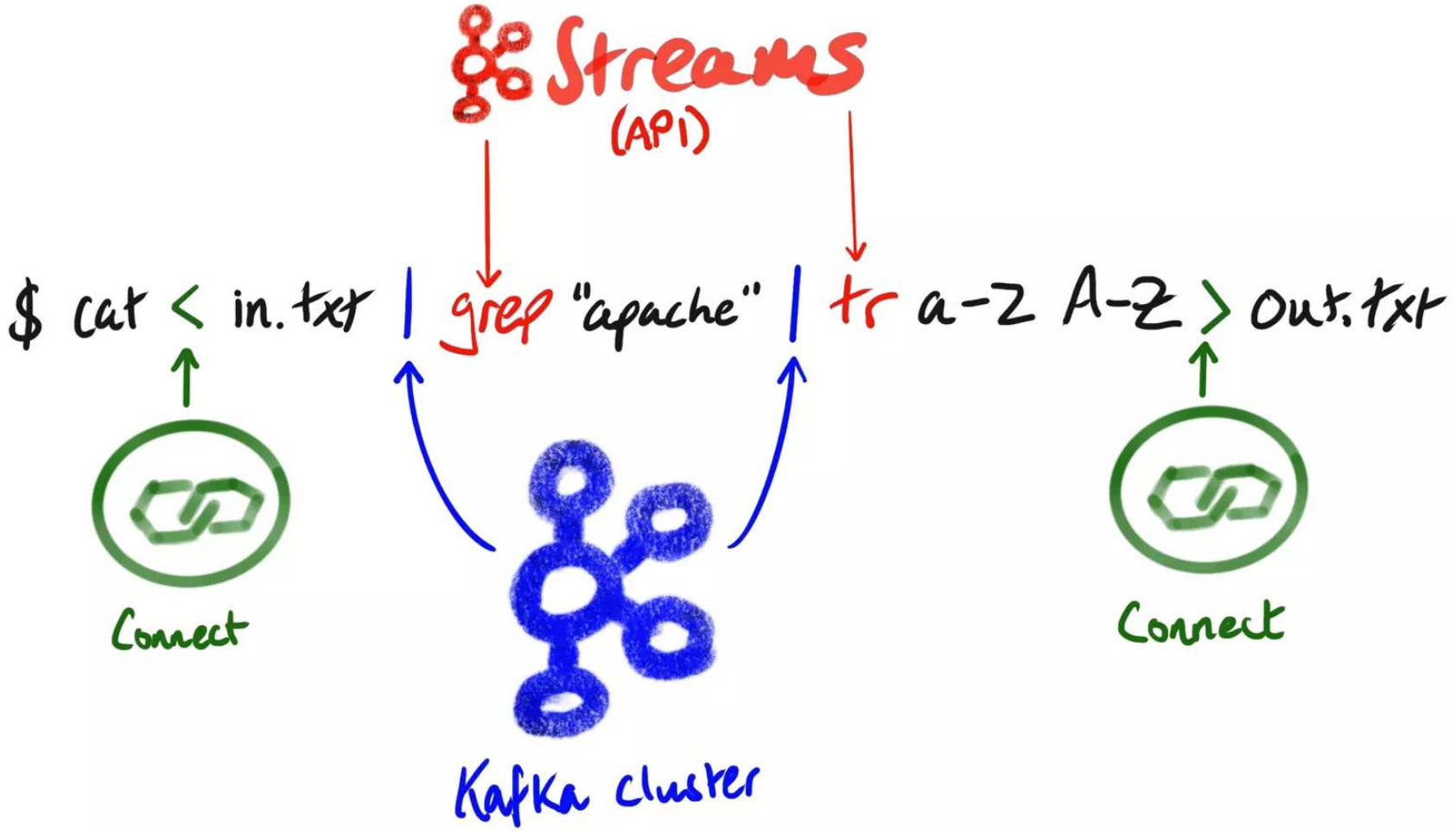
} topic (brokers)

- acks: 0 (fire and forget)
- 1 (data loss unlikely)
- all (no data loss)

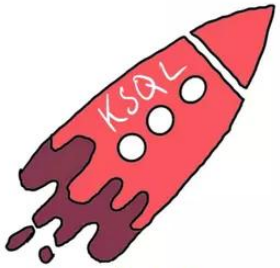
} producer



Kafka as a Linux command



ksqlDB vs Kafka Streams vs producer/consumer



is built on

Streams

is built on

producer/
consumer

ease of use

flexibility

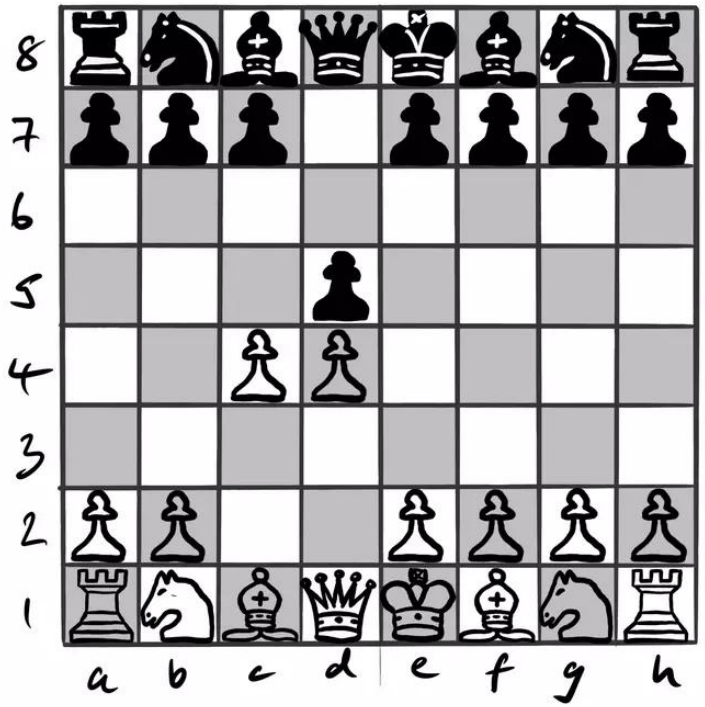
CREATE STREAM, CREATE TABLE,
SELECT, JOIN, GROUP BY, SUM, WHERE ...

KStream, KTable,
filter(), map(), flatMap(),
join(), aggregate(), ...

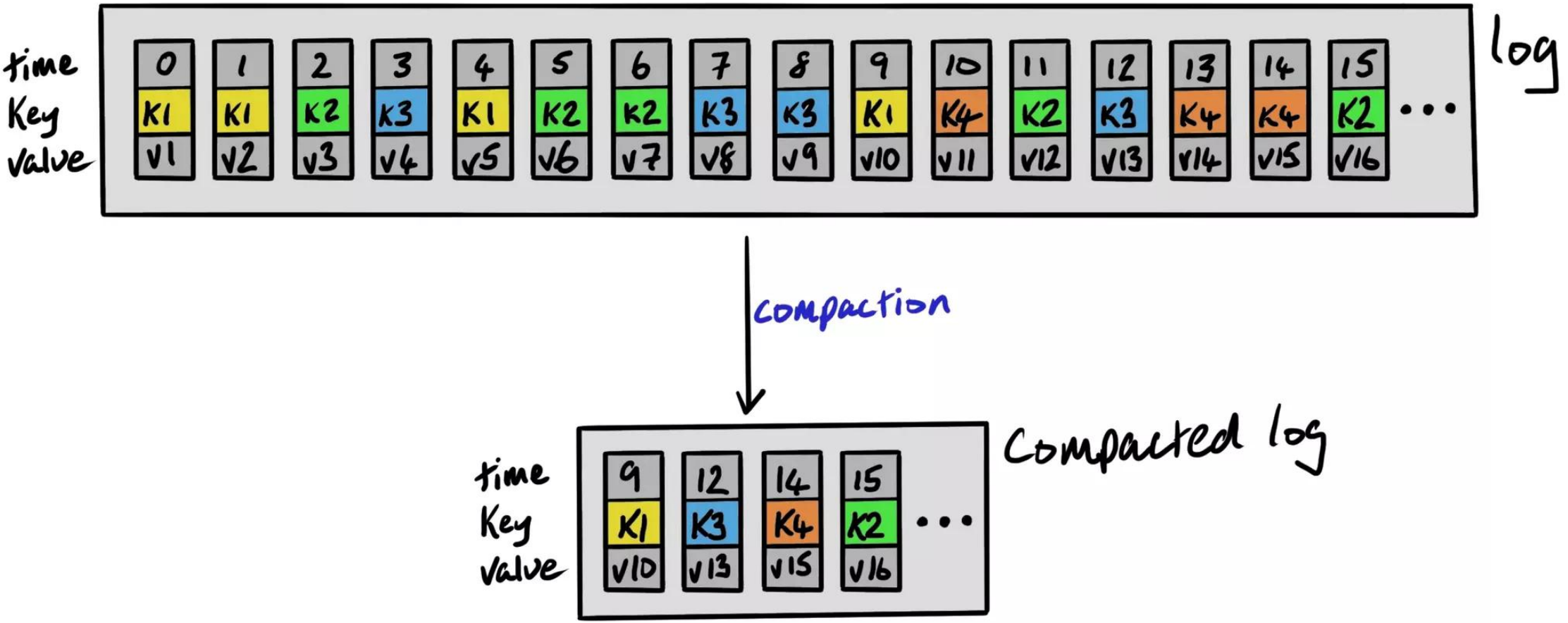
subscribe(), poll(), send(),
flush(), beginTransaction(), ...

streams and tables

1. move white Queens pawn forward two squares
2. move black Queens pawn forward two squares
3. move white Queen-side bishop pawn forward two squares



Compacted topic



Stateless Operations

Branch
Filter
FilterNot
Flatmap
FlatmapValues
Foreach
GroupByKey
GroupBy
Map
MapValues
Peek
Print
SelectKey
toStream

* marks stream for re-partitioning
* always causes re-partitioning
* requires repartitioning if stream is marked for it

Stateful Operations

Aggregations

Joins

Windows

Custom (PAPI)

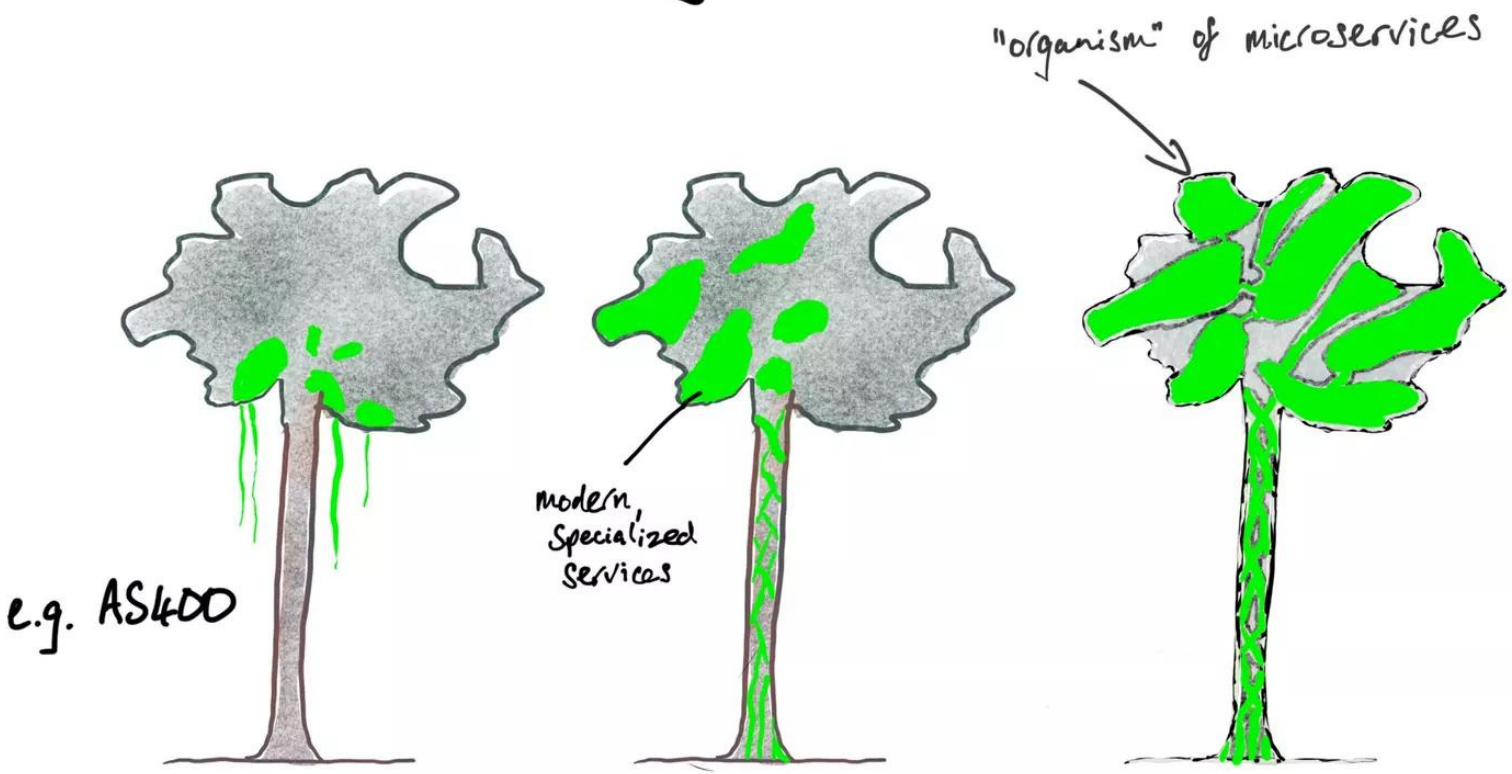
Event-time processing

Stream time:

- ingest time: when the record is written to a partition
- event time: when the event occurred 

processing time: when the event was processed by the streaming app. 

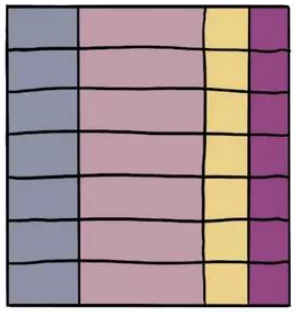
the strangler pattern



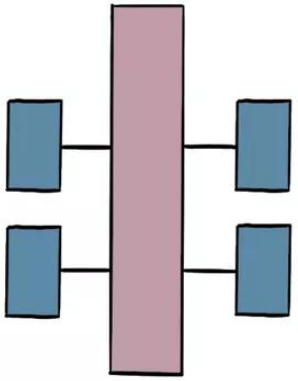
data access patterns

SQL

Ye olde RDBMS

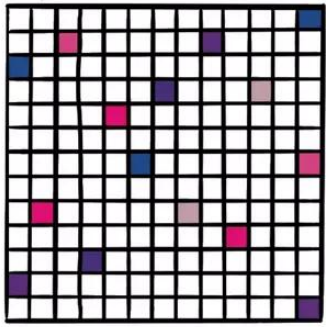


OLAP

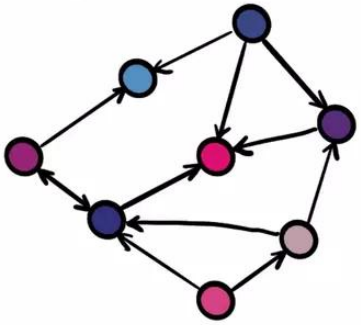


NoSQL

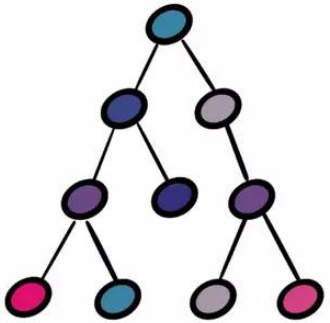
Column-store



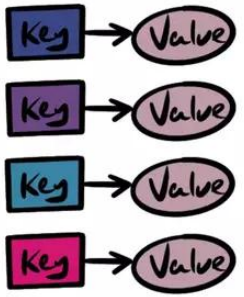
Graph

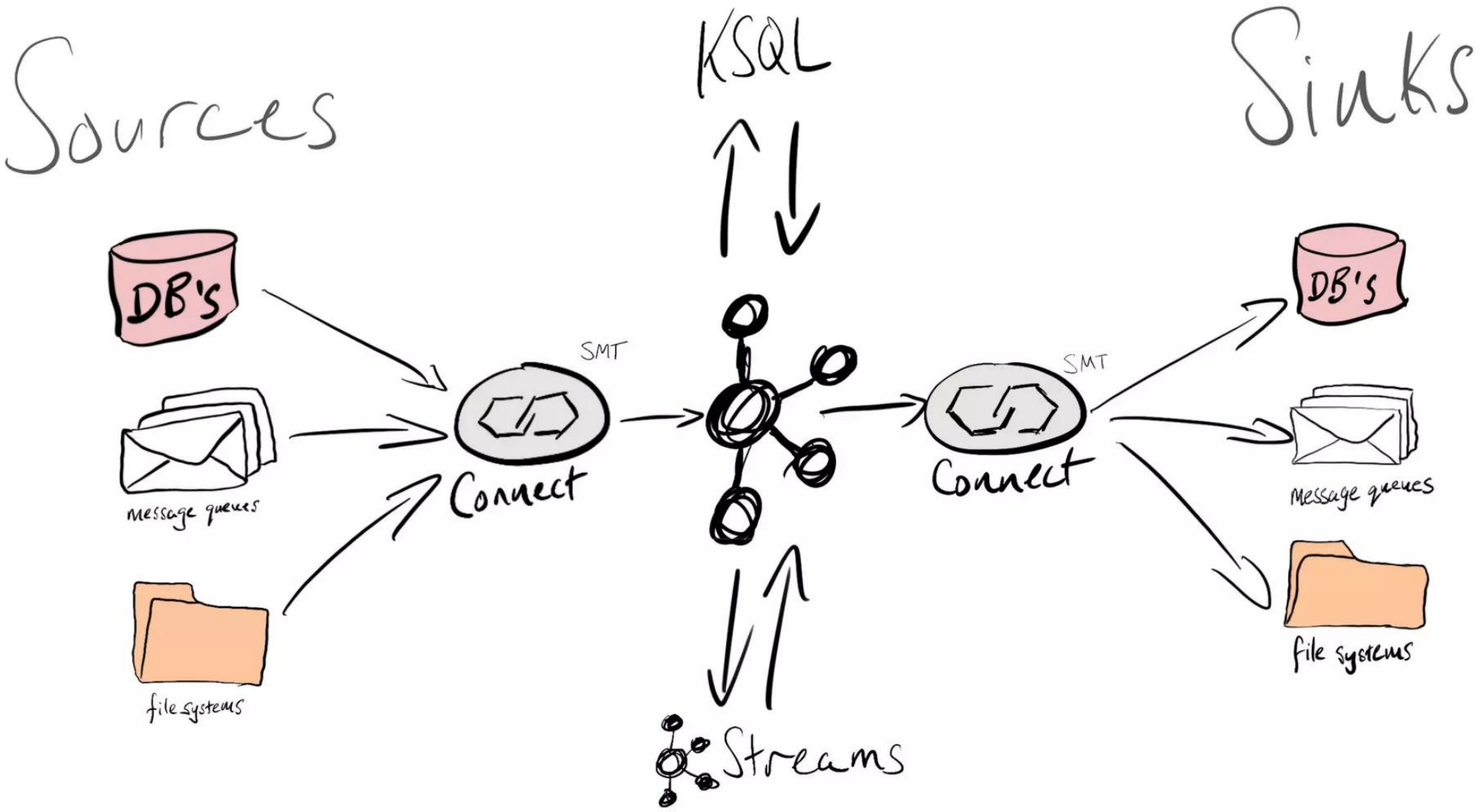


Document



Key/Value




Sources
KSQL
Sinks
DB's
message queues
file systems
SMT
Connect
Streams
SMT
Connect
DB's
message queues
file systems

Kafka Connect

Memory/cache

hazelcast JET

GridGain

Columnar DB's

DATASTAX

cassandra

APACHE HBASE

APACHE PHOENIX

MAPR-DB

Search/document DB's

splunk> Solr

Couchbase

Graph DB's

Storage

hadoop

timeseries DB's

druid

Wow! Over 100 connectors!

change data capture (CDC)

ORACLE GOLDEN GATE

ATTUNITY

HVR

SQDATA

SQL

Java JDBC

VERTICA

YugaByte DB

NETEZZA

ROCKSET

messaging

TIBCO

JMS

solace.

MQTT.ORG

IBM MQ

HUMIO

ACTIVEMQ

PUSH TECHNOLOGY

other Stuff

salesforce

servicenow

PRIVITAR VENAFI

StreamSets

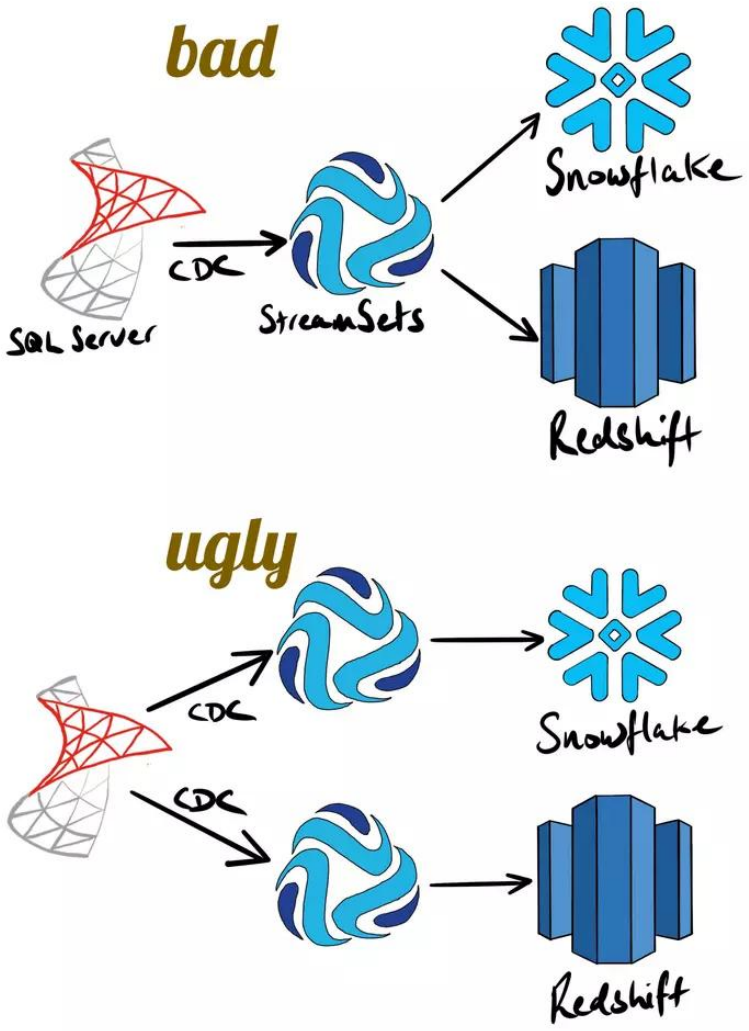
GPU

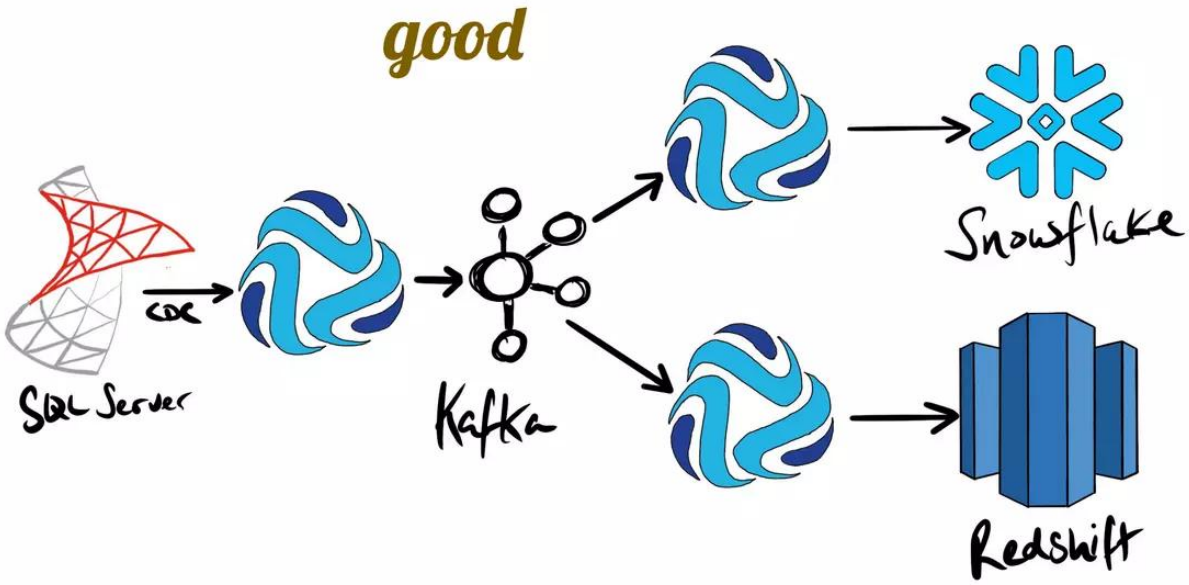
kinetica

omni·sci

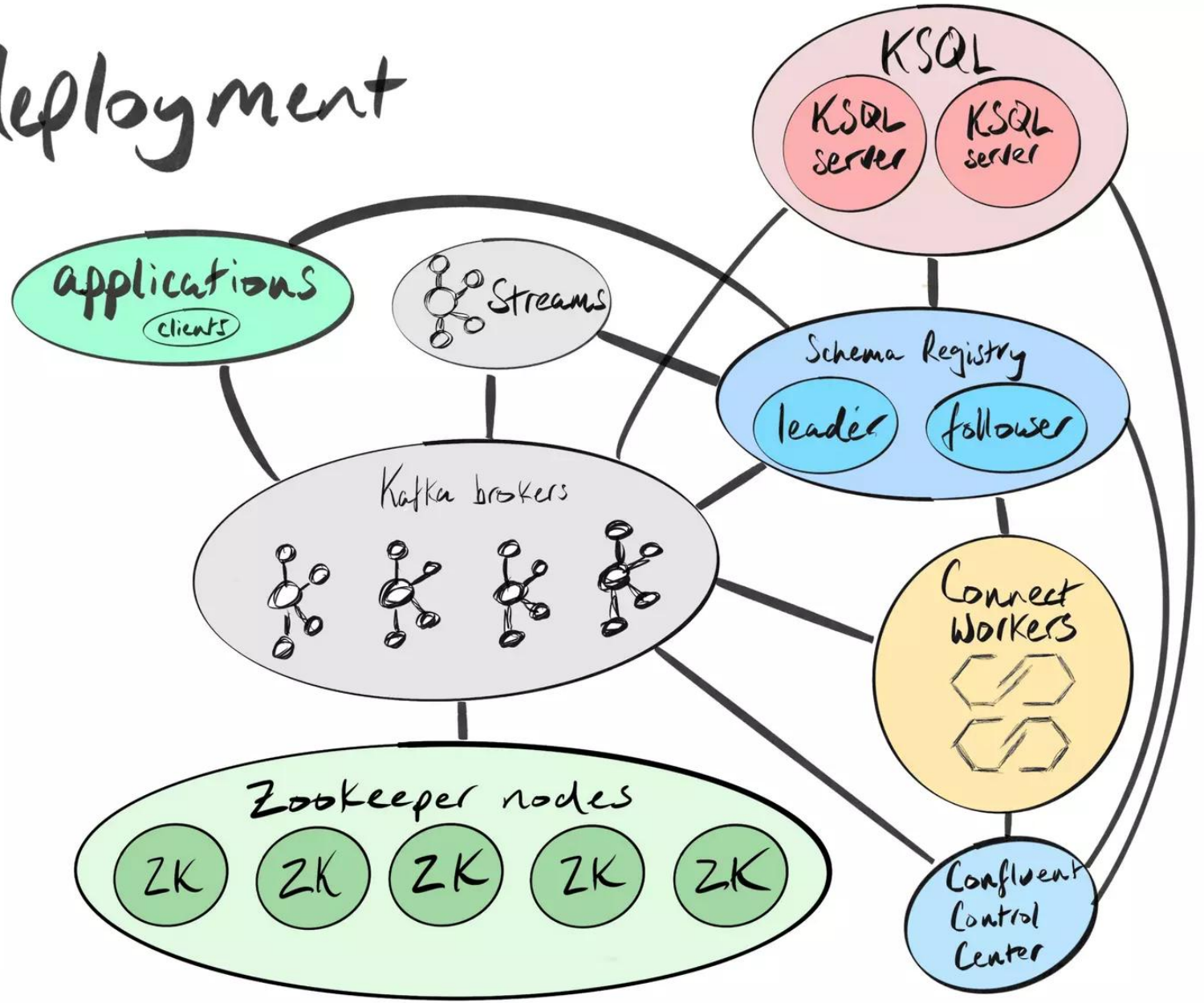
confluent.io/hub

StreamSets: don't tightly couple pipelines

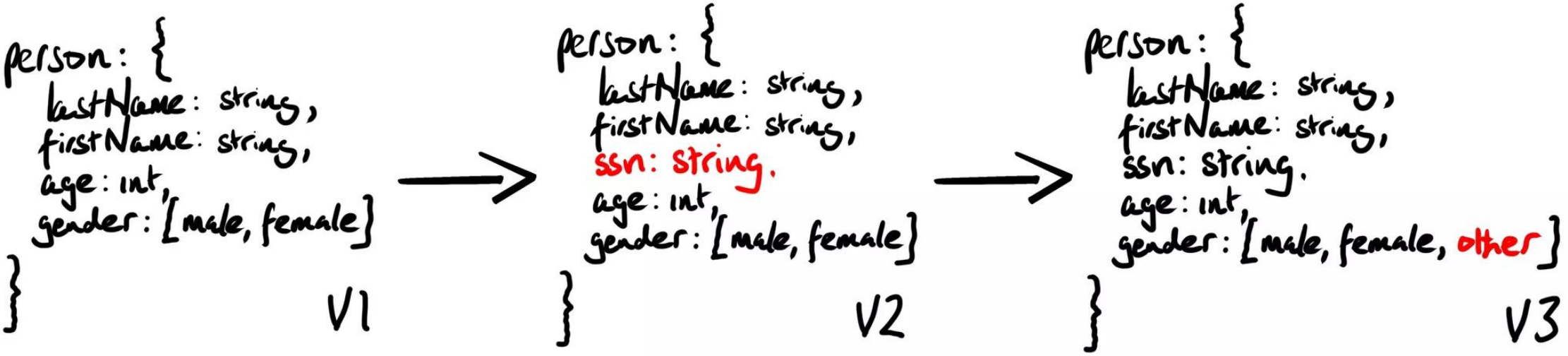




a typical deployment



Schema Evolution



Pushtopic events from Salesforce

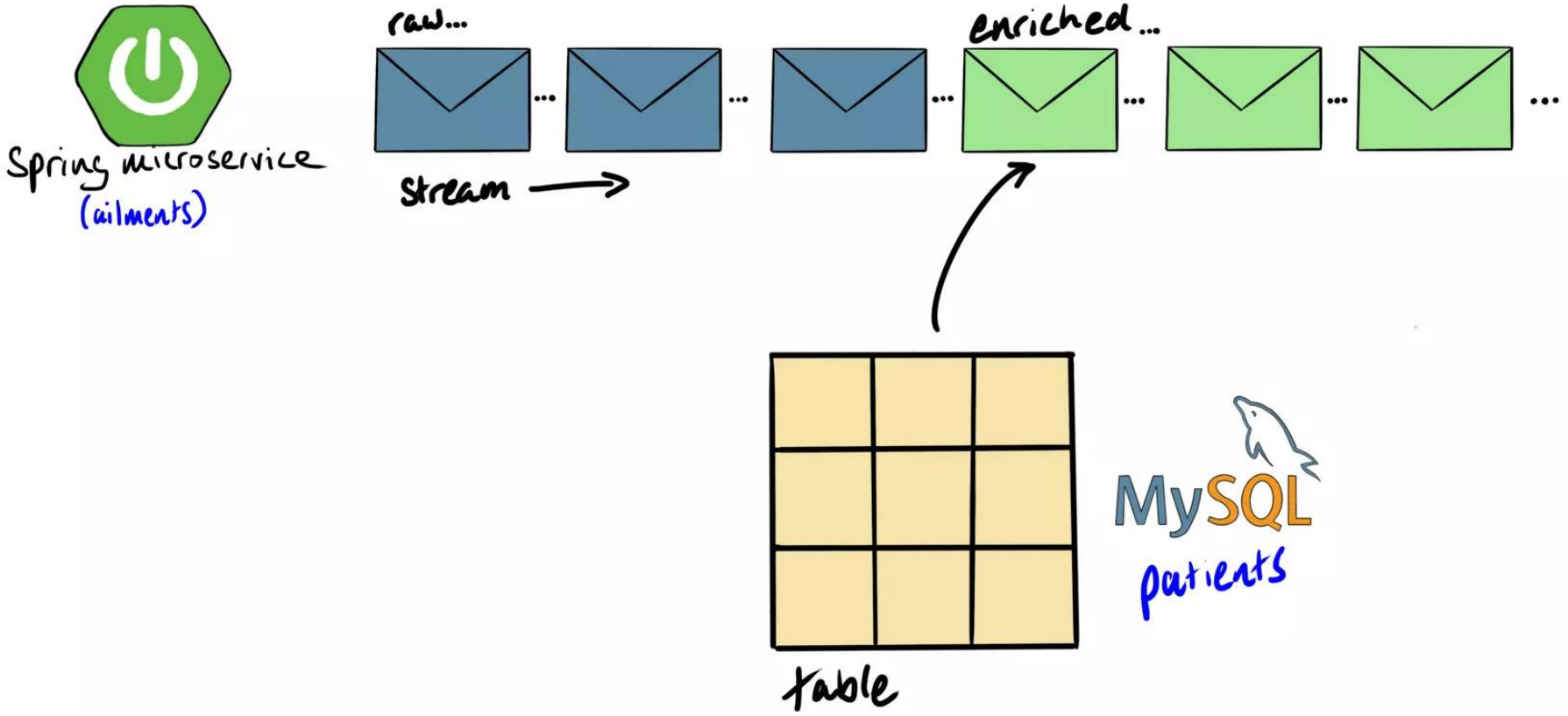
create pushtopic

```
PushTopic pushTopic = new PushTopic();
pushTopic.Name = 'ContactUpdates';
pushTopic.Query = 'SELECT Id, [...] FROM Contact';
pushTopic.ApiVersion = 50.0;
pushTopic.NotifyForOperationCreate = true;
pushTopic.NotifyForOperationUpdate = true;
pushTopic.NotifyForOperationUndelete = true;
pushTopic.NotifyForOperationDelete = true;
pushTopic.NotifyForFields = 'All';
insert pushTopic;
```

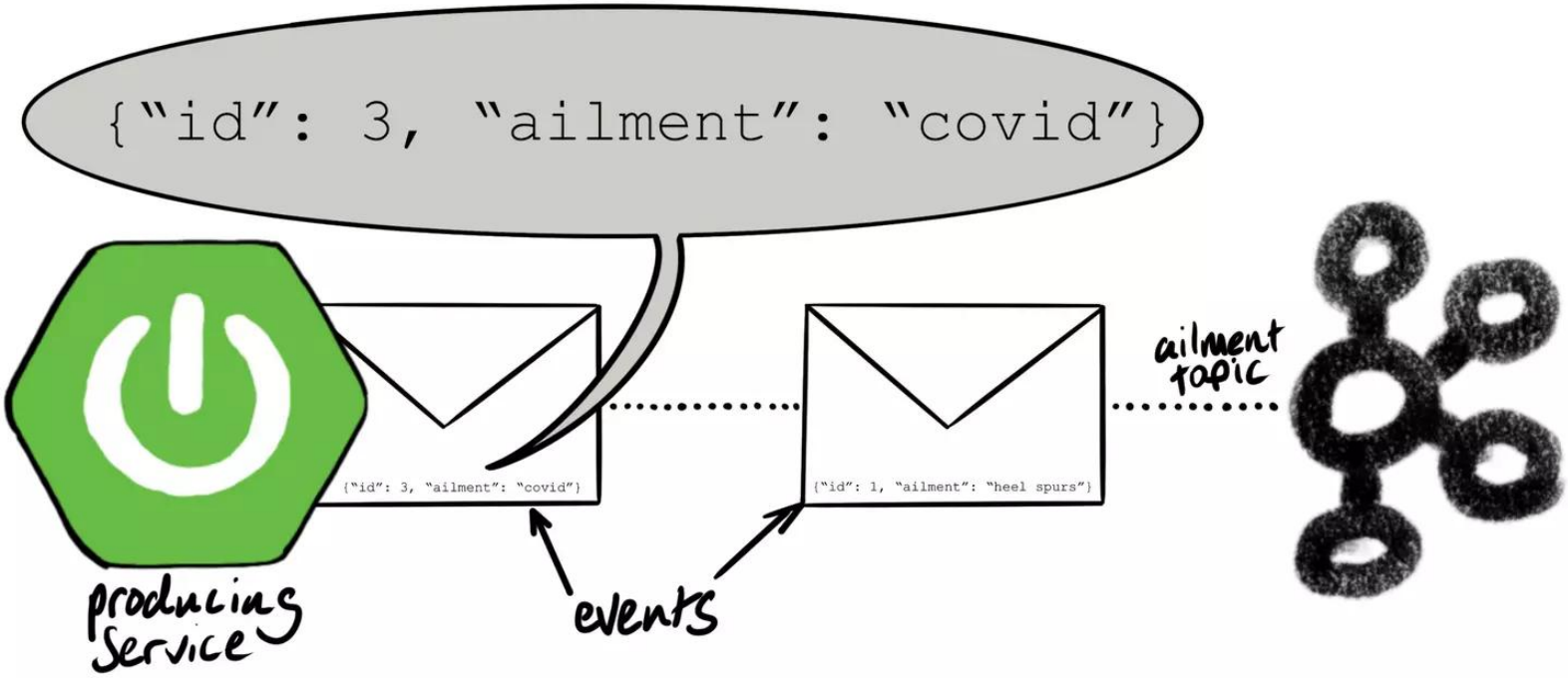
create connector

```
http PUT cp01.woolford.io:8083/connectors/sfdc-contact-updates/config <<< '
{
    "confluent.topic.bootstrap.servers": "cp01.woolford.io:9092,cp02...",
    "confluent.topic.replication.factor": "1",
    "connector.class":
"io.confluent.salesforce.SalesforcePushTopicSourceConnector",
    "kafka.topic": "sfdc-contact-updates",
    "name": "sfdc-contact-updates",
    "salesforce.consumer.key": "********",
    "salesforce.consumer.secret": "********",
    "salesforce.initial.start": "all",
    "salesforce.object": "Contact",
    "salesforce.password": "cxI3o3R$****",
    "salesforce.password.token": "iggX9UEPizeaizvpg02Vj****",
    "salesforce.push.topic.name": "ContactUpdates",
    "salesforce.username": "alex@woolford.io",
    "tasks.max": "1"
}'
```

stream/table join



the ailments

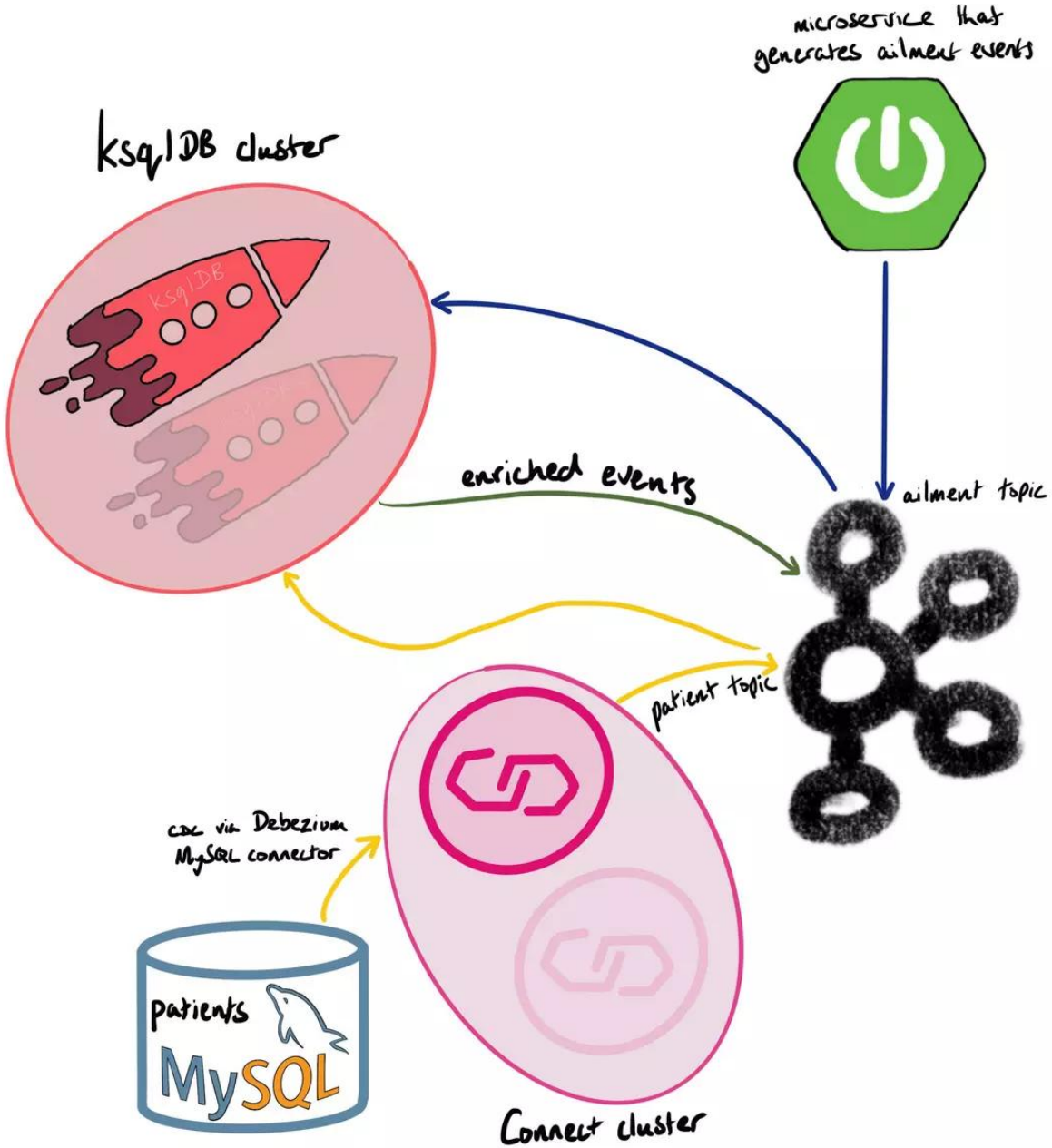


The patients.

```
mysql> use healthcare;

Database changed
mysql> select * from patient;
+----+-----------+------------+--------------+------------+
| id | firstname | lastname   | enroll_start | enroll_end |
+----+-----------+------------+--------------+------------+
|  1 | Donald    | Trump      | 2019-04-14   | 2020-04-13 |
|  2 | Vladimir  | Putin      | 2019-05-20   | 2020-05-19 |
|  3 | Boris     | Johnson    | 2019-01-24   | 2020-01-24 |
|  4 | Mark      | Zuckerberg | 2019-03-03   | 2020-03-02 |
|  5 | Benjamin  | Netanyahu  | 2019-08-24   | 2020-08-23 |
|  6 | Jair      | Bolsonaro  | 2019-09-19   | 2020-09-18 |
|  7 | Farty     | McFartface | 2019-08-23   | 2020-08-22 |
+----+-----------+------------+--------------+------------+
7 rows in set (0.00 sec)
```

Change data capture from MySQL with the Debezium connector

```
http PUT http://cp01.woolford.io:8083/connectors/mysql-cdc-healthcare/config  <<< '
{
    "name": "mysql-cdc-healthcare",
    "connector.class": "io.debezium.connector.mysql.MySqlConnector",
    ...
    "include.schema.changes": "false",
    "snapshot.mode": "when_needed",
    "table.whitelist": "healthcare.patient",
    "transforms": "unwrap,HoistField",
    "transforms.unwrap.type": "io.debezium.transforms.ExtractNewRecordState",
    "transforms.HoistField.type": "org.apache.kafka.connect.transforms.ExtractField$Key",
    "transforms.HoistField.field": "id"
}'
```


microservice that generates ailment events
ksqlDB cluster
enriched events
ailment topic
patient topic
CDC via Debezium MySQL connector
patients
MySQL
Connect cluster

create the stream and table in ksqlDB

```
CREATE TABLE PATIENT (
  id STRING,
  firstname VARCHAR,
  lastname VARCHAR,
  enroll_start BIGINT,
  enroll_end BIGINT
) WITH (
  kafka_topic='deepthought.healthcare.patient',
  value_format='JSON',
  KEY='id'
);
```

```
CREATE STREAM AILMENT (
  id STRING,
  ailment VARCHAR
) WITH (
  kafka_topic='ailment',
  value_format='JSON',
  KEY='id'
);
```

streaming join in ksqlDB

```
select
  ailment.id as id,
  ailment,
  firstname,
  lastname,
  case
    when ailment.rowtime > enroll_start * 86400 * 1000
      and ailment.rowtime <= enroll_end * 86400 * 1000
    then true
    else false
  end as covered
from ailment inner join patient
on ailment.id = patient.id
emit changes;
```

ksqlDB

Typical ksqlDB use cases:

- streaming ETL
- real-time monitoring and analytics
- data exploration and discovery
- anomaly detection
- personalization
- sensor and IoT
- customer 360

events from video

KSQL user defined function

Enrichment with MaxMind's geoIP

```
select getgeoforip(ip) from mystream emit changes;
```

github.com/alexwoolford/ksql-udf-geoip

enrichment with EXIF from an image

NYC released 173 million anonymized taxi journeys

... which was easily de-anonymized thanks to EXIF

Bradley Cooper (Click to Explore)

```
SELECT D.dropoff_latitude, D.dropoff_longitude, F.total_amount, F.tip_amount
FROM tripData AS D, tripFare AS F
WHERE D.hack_license = F.hack_license AND D.pickup_datetime = F.pickup_datetime
  AND pickup_datetime > "2013-07-08 19:33:00" AND pickup_datetime < "2013-07-08 19:37:00"
  AND pickup_latitude > 40.719 AND pickup_latitude < 40.7204
  AND pickup_longitude > -74.0106 AND pickup_longitude < -74.01;
```

```
from exif import Image

with open('IMG_20181103_200537.jpg', 'rb') as
image_file:
  my_image = Image(image_file)
```

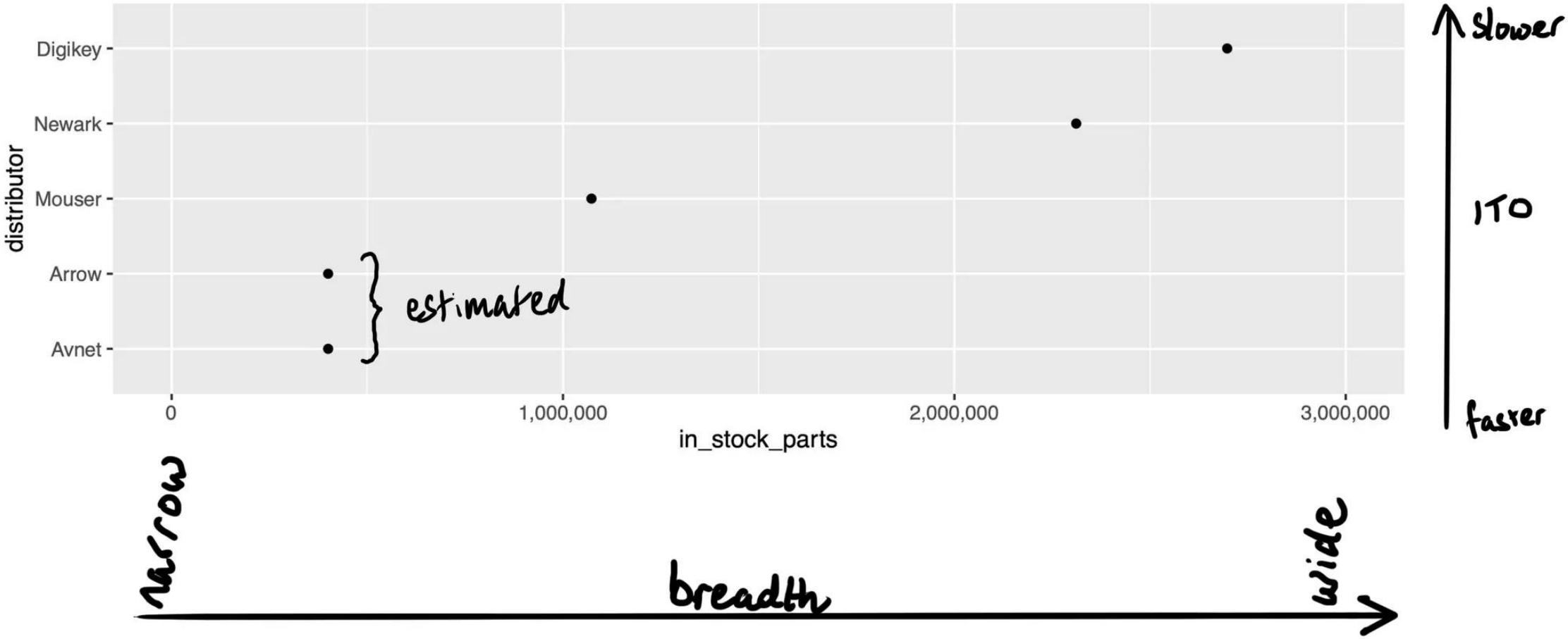
lat/lon
timestamp
camera settings
copyright

Data-driven inventory expansion

The challenge: add SKU's without decreasing inventory turns

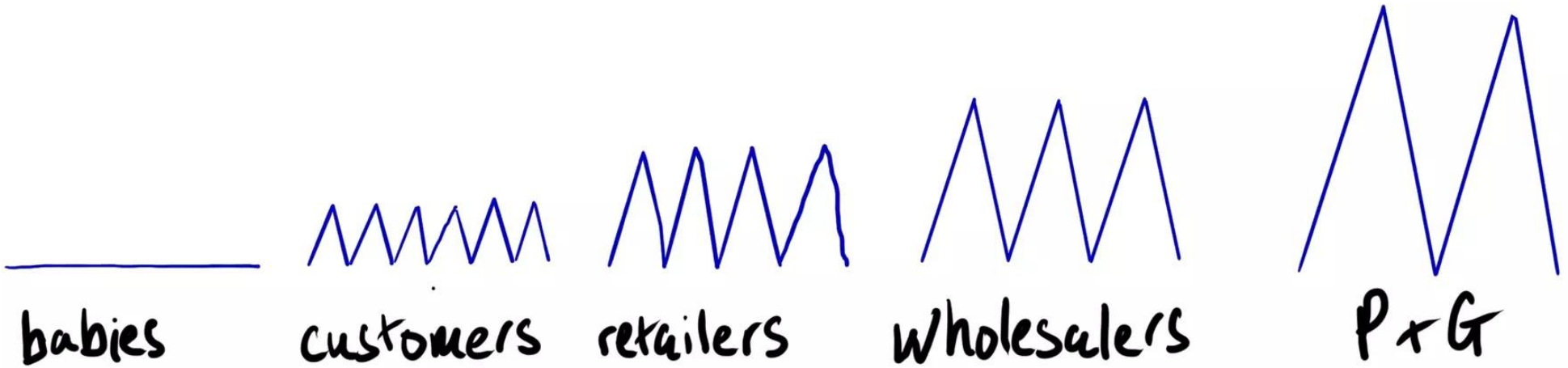
$$\text{inventory turnover (ITO)} = \frac{\text{cost of goods sold}}{\text{average inventory}}$$

more SKU's == slower inventory turns



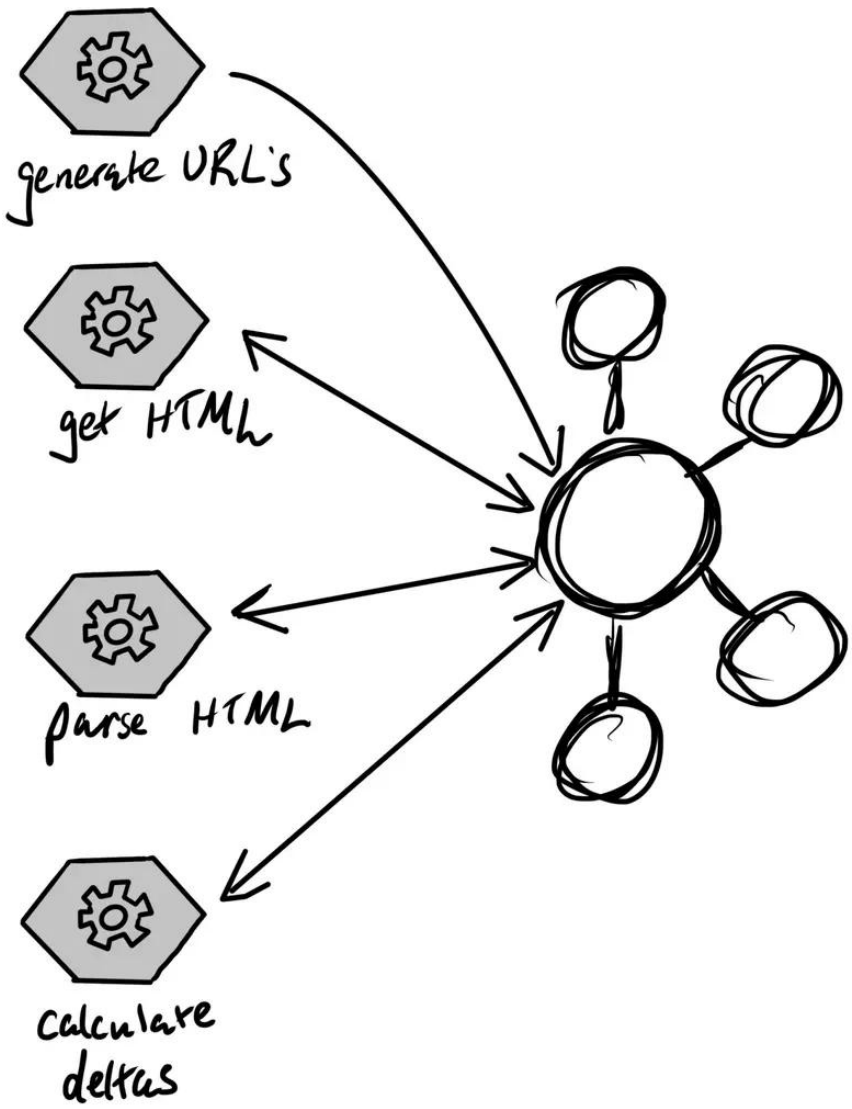
Information distortion: the bullwhip effect

order variability up the Pampers supply chain



source: Lee, Padmanabhan and Whang, 1997

Supply chain visibility



products/en/development-boards-kits-programmers/evaluation-and-demonstration-boards-and-

Image	[redacted] Part Number	Manufacturer Part Number	Manufacturer	Description	Quantity Available	Unit Price USD	Minimum Quantity
	1568-1209-ND	BOB-12009	SparkFun Electronics	LOGIC LEVEL CONVERTER - BI-DIREC	1,638 - Immediate	$2.95000	1
	1568-1279-ND	BOB-09118	SparkFun Electronics	SPARKFUN OPTO-ISOLATOR BREAKOUT	279 - Immediate	$4.95000	1
	497-10828-ND	EVAL6235PD	STMicroelectronics	EVAL BOARD FOR THE L6235PD	0	Active	1 Non-Stock
	1568-1193-ND	BOB-11189	SparkFun Electronics	TRANSCEIVER BREAKOUT - MAX3232	1,387 - Immediate	$5.95000	1
	1460-1201-ND	TMC2208 SILENTSTEPSTICK	Trinamic Motion Control GmbH	TMC2208 STEPPER DRIVER BOARD	1,188 - Immediate	$7.32000	1
	768-1317-ND	LC234X	FTDI, Future Technology Devices International Ltd	MOD USB UART DEV FT234XD	284 - Immediate	$7.78000	1

Audio Products - 222 New Products

- Accessories (659 items)
- Alarms, Buzzers, and Sirens (5607 items)
- Amplifiers (23 items)
- Buzzer Elements, Piezo Benders (143 items)
- Guitar Parts, Accessories (213 items)
- Microphones (1767 items)
- Speakers (2608 items)
- Vacuum Tubes (1408 items)

Battery Products - 344 New Products

- Accessories (274 items)
- Batteries Non-Rechargeable (Primary) (698 items)
- Batteries Rechargeable (Secondary) (837 items)
- Battery Chargers (415 items)
- Battery Holders, Clips, Contacts (1704 items)
- Battery Packs (2832 items)
- Cigarette Lighter Assemblies (69 items)

Predictable URL's

the # of pages, for each category, can be determined by the # of parts.

E.g. ceiling(5,607/500) = 12

lots of parts on each page

https://www.digikey.com/products/en/audio-products/alarms-buzzers-and-sirens/157?&quantity=0&pageSize=500&page=1

Show in-stock items only

easy pagination

Although it's common practice, web scraping is ethically dubious and can lead to legal problem (e.g. trespass to chattels).

Always spread the requests across a pool of residential IP addresses and go slow, so it doesn't look like a denial of service attack.

Selenium browser automation

html

url

Spring Boot microservice reads from URL topic, gets the HTML, and writes that back to another topic

use Selenium when the pages have to execute Javascript to display the content

Chrome browser

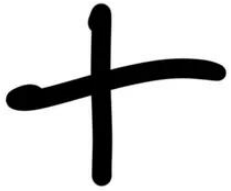
Temporarily (or permanently) storing the HTML gives a lot of flexibility. For example, it's possible to re-parse information from the HTML if the page format changes, or if there's some useful information that wasn't captured.

Compare Parts		Image	Digi-Key Part Number	Manufacturer Part Number	Manufacturer	Description	Quantity Available	Unit Price USD	Minimum Quantity	Packaging	Series	Part Status
☐			102-1153-ND	CEM-1203(42)	CUI Devices	AUDIO MAGNETIC XDCR 3-5V TH	57,928 - Immediate	$0.74000	1	Tray	CEM	Active
☐			102-1286-ND	CSQG703BP	CUI Devices	AUDIO MAGNETIC XDCR 2-4V TH	31,760 - Immediate	$0.83000	1	Tray	CSQ	Active
☐			102-1156-ND	CEM-1212S	CUI Devices	AUDIO MAGNETIC XDCR 6-16V TH	40,718 - Immediate	$0.86000	1	Tray	CEM	Active
☐			102-1458-ND	CST-931AP	CUI Devices	AUDIO MAGNETIC XDCR 2-4V TH	69,146 - Immediate	$1.06000	1	Tray	CST	Active
☐			102-1199-2-ND	CT-1205-SMT-TR	CUI Devices	AUDIO MAGNETIC XDCR 3-8V SMD	36,000 - Immediate	$1.18678	450	Tape & Reel (TR) Alternate Packaging	CT	Active

{"timestamp":1598492553789,"dpn":"DM330028-2-ND","mpn":"DM330028-2","mfg":"Microchip Technology","qoh":3}
{"timestamp":1598492553789,"dpn":"296-28530-ND","mpn":"MSP-EXP430FR5739","mfg":"Texas Instruments","qoh":166}
{"timestamp":1598492553789,"dpn":"497-STM32H7B3I-DK-ND","mpn":"STM32H7B3I-DK","mfg":"STMicroelectronics","qoh":1}
{"timestamp":1598492553789,"dpn":"TMDSLCDK6748-ND","mpn":"TMDSLCDK6748","mfg":"Texas Instruments","qoh":10}
{"timestamp":1598492553789,"dpn":"497-12073-ND","mpn":"STM8SVLDISCOVERY","mfg":"STMicroelectronics","qoh":5}
{"timestamp":1598492553789,"dpn":"497-13930-ND","mpn":"STM32L100C-DISCO","mfg":"STMicroelectronics","qoh":12}
{"timestamp":1598492553789,"dpn":"428-4367-ND","mpn":"CY8CKIT-146","mfg":"Cypress Semiconductor Corp","qoh":20}
{"timestamp":1598492553789,"dpn":"428-4510-ND","mpn":"CY8CKIT-147","mfg":"Cypress Semiconductor Corp","qoh":10}
{"timestamp":1598492553789,"dpn":"497-10660-ND","mpn":"STM8L-DISCOVERY","mfg":"STMicroelectronics","qoh":19}
{"timestamp":1598492553789,"dpn":"497-18395-ND","mpn":"NUCLEO-G070RB","mfg":"STMicroelectronics","qoh":22}
{"timestamp":1598492553789,"dpn":"497-15095-ND","mpn":"NUCLEO-F070RB","mfg":"STMicroelectronics","qoh":1}
{"timestamp":1598492553789,"dpn":"497-15979-ND","mpn":"NUCLEO-F031K6","mfg":"STMicroelectronics","qoh":10}
{"timestamp":1598492553789,"dpn":"497-16284-ND","mpn":"NUCLEO-L011K4","mfg":"STMicroelectronics","qoh":16}
{"timestamp":1598492553789,"dpn":"497-18155-ND","mpn":"NUCLEO-8L152R8","mfg":"STMicroelectronics","qoh":16}
{"timestamp":1598492553789,"dpn":"497-18239-ND","mpn":"NUCLEO-L412KB","mfg":"STMicroelectronics","qoh":20}

Calculate the deltas

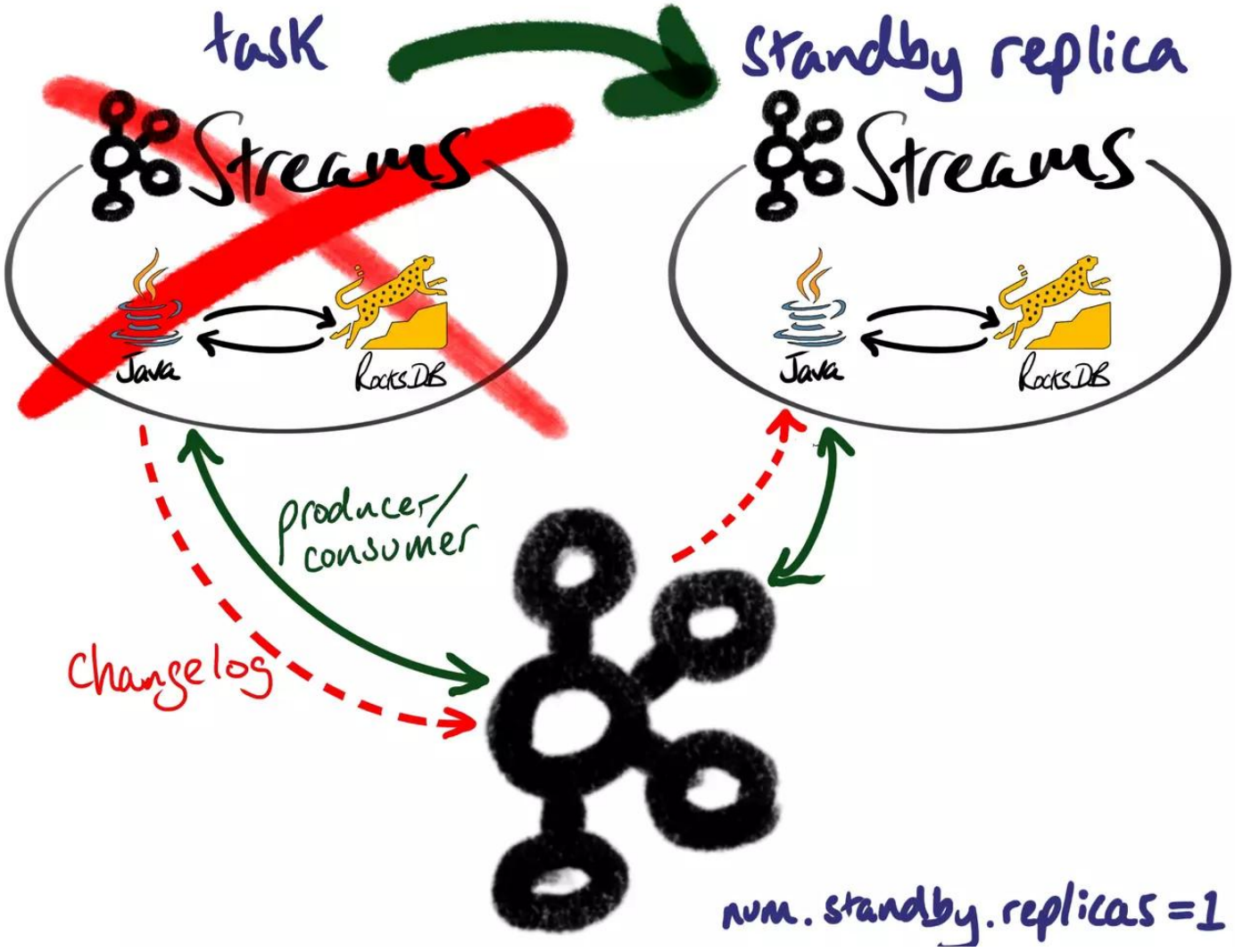
{"timestamp":1598492567811,"dpn":"1568-DEV-16996-ND","mpn":"DEV-16996","mfg":"SparkFun Electronics","qoh":196}



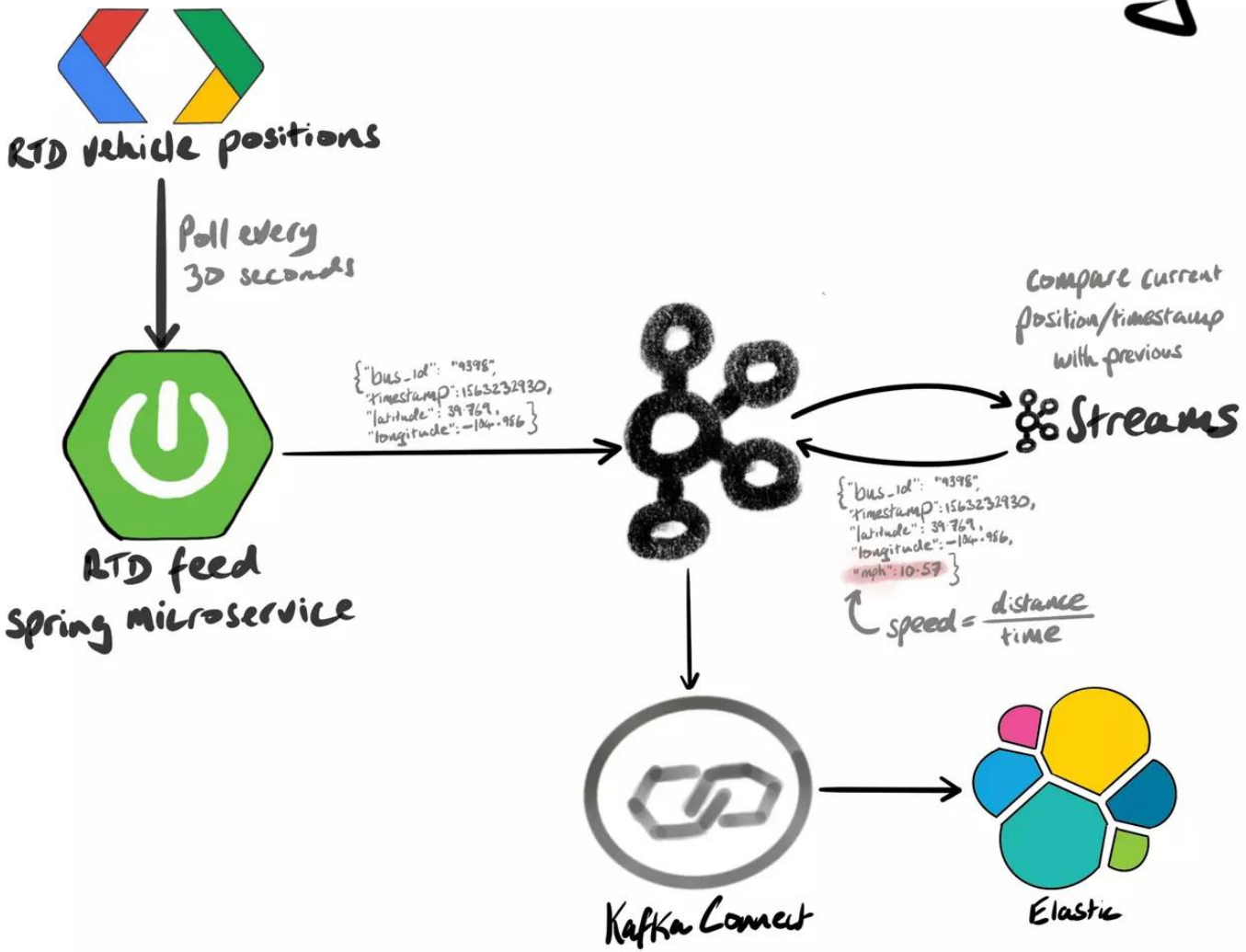
{"timestamp":1598492654204,"dpn":"1568-DEV-16996-ND","mpn":"DEV-16996","mfg":"SparkFun Electronics","qoh":194}

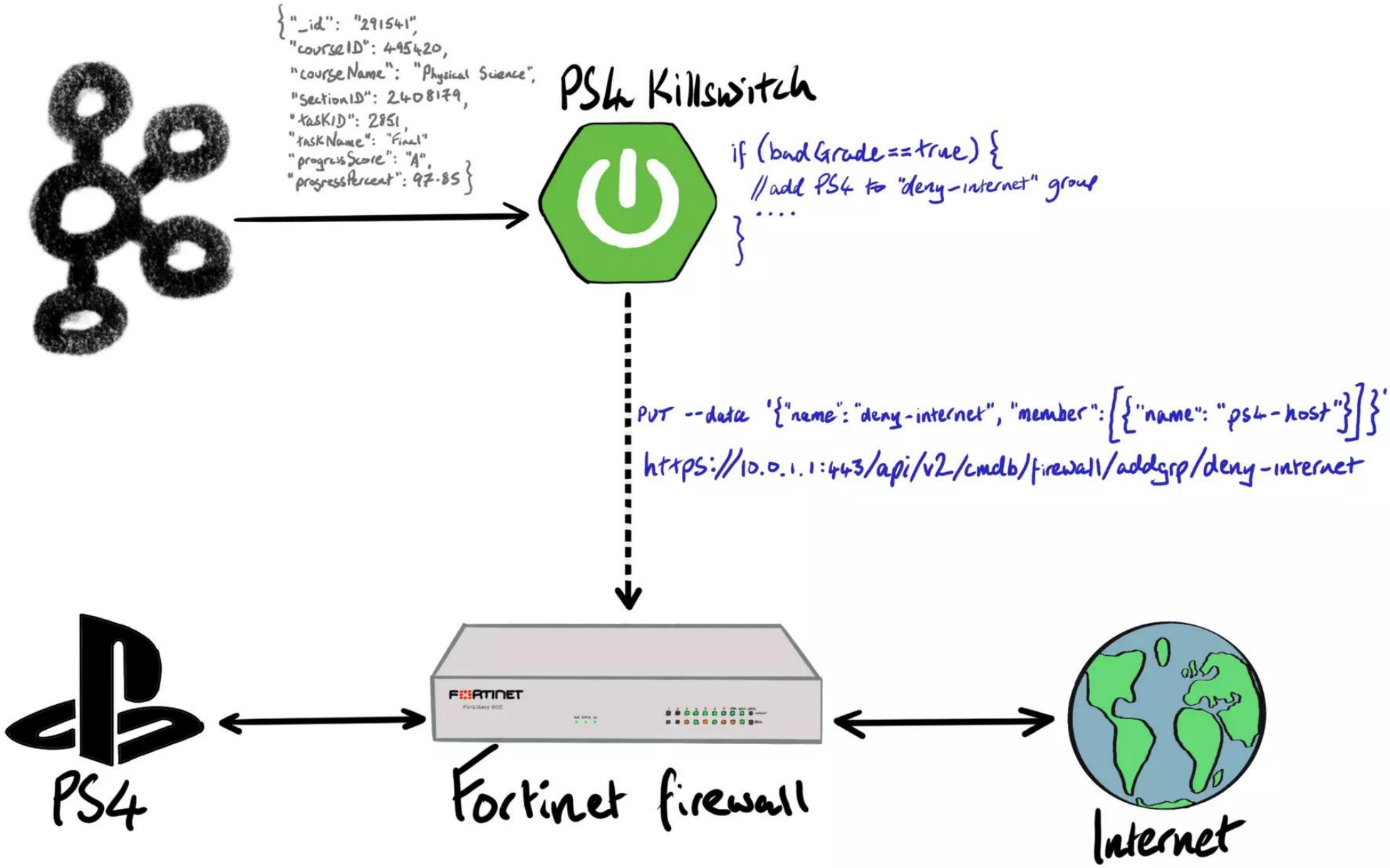
{"timestamp":1598492654204,"dpn":"1568-DEV-16996-ND","mpn":"DEV-16996","mfg":"SparkFun Electronics","delta":2}

Kafka Streams' state store



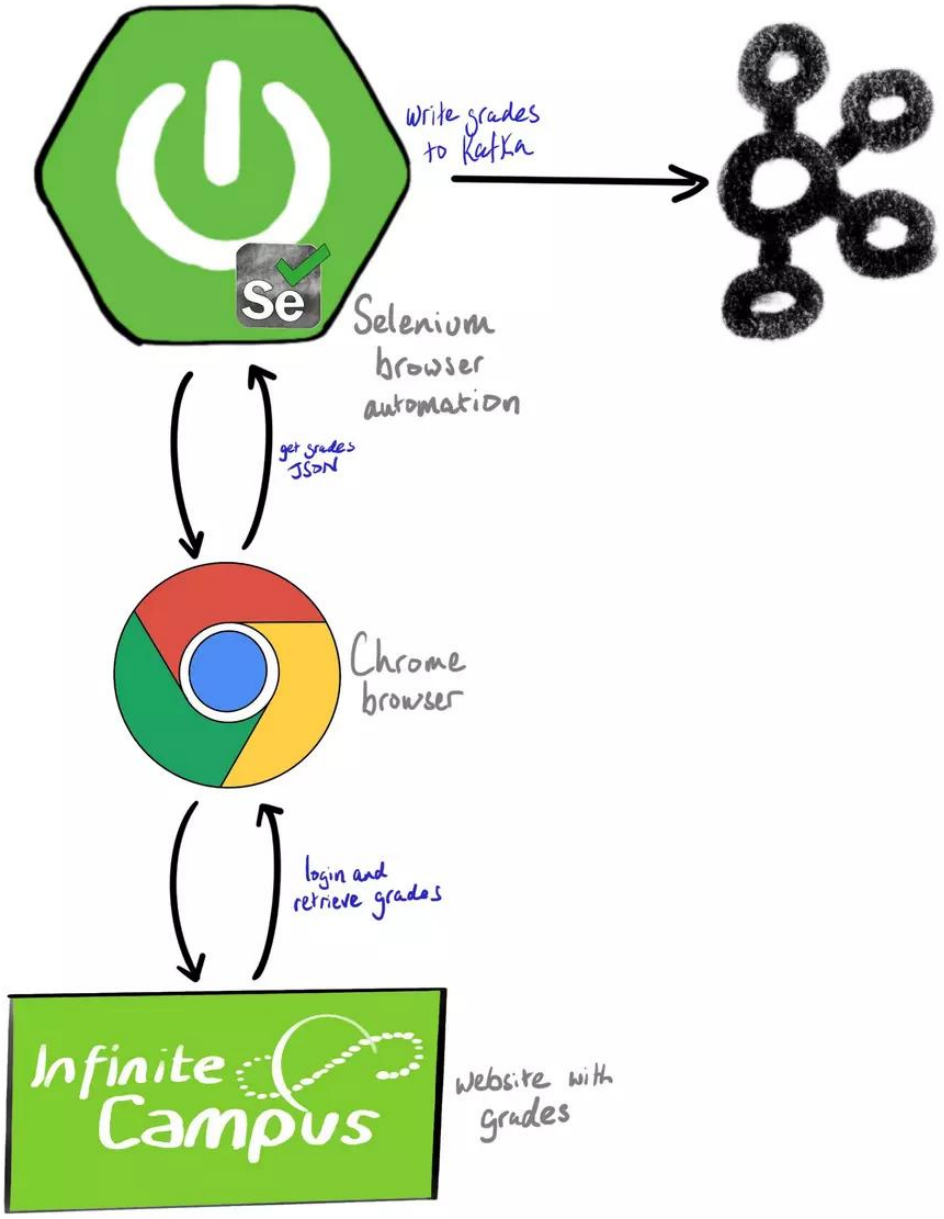
how fast are RTD buses going?




{"_id": "291541",
"courseID": 495420,
"courseName": "Physical Science",
"sectionID": 2408179,
"taskID": 2851,
"taskName": "Final"
"progressScore": "A",
"progressPercent": 97.85}
PS4 Killswitch
if (badGrade==true) {
//add PS4 to "deny-internet" group
....
}
PUT --data '{"name":"deny-internet", "member":[{"name": "ps4-host"}]}'
https://10.0.1.1:443/api/v2/cmdb/firewall/addgrp/deny-internet
FORTINET
PS4
Fortinet firewall
Internet

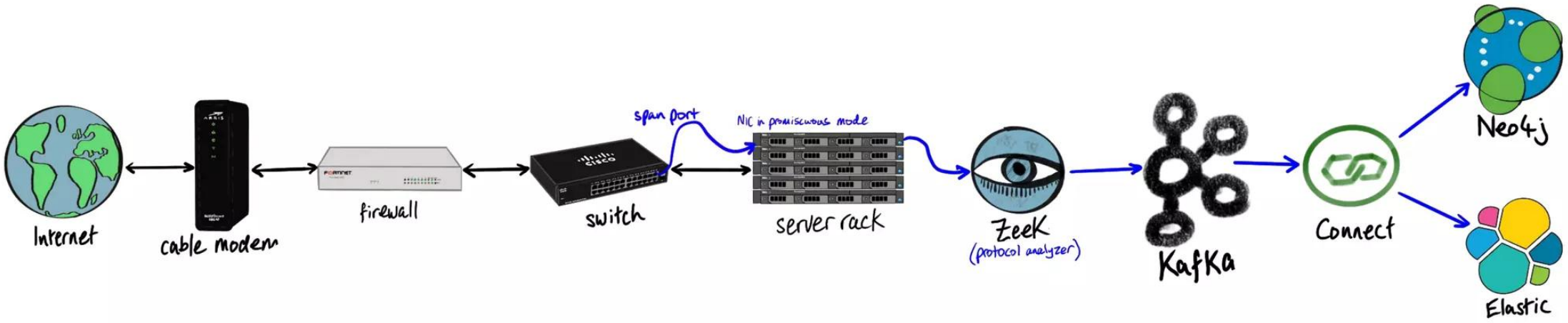
Bad grades?

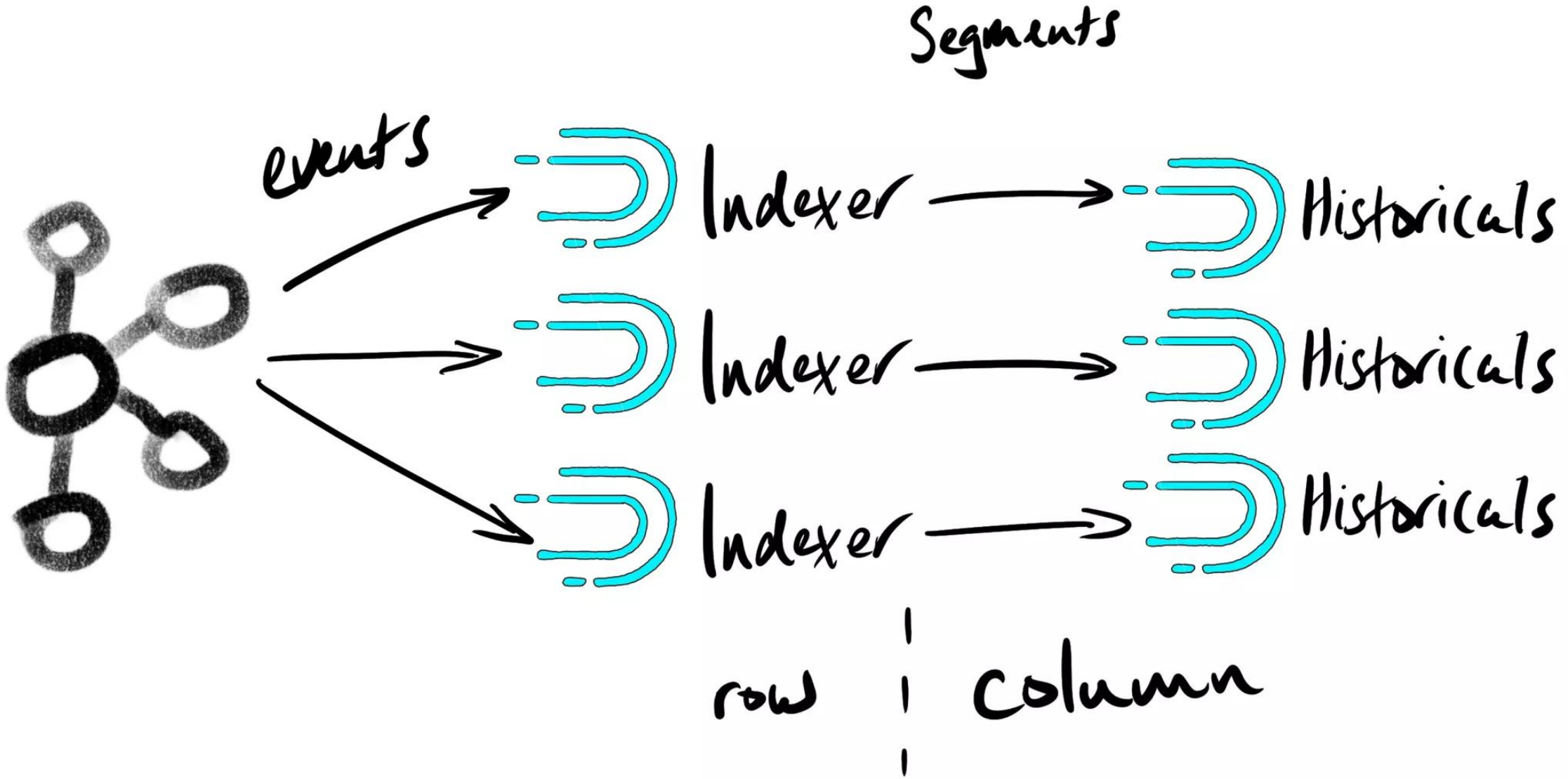
No PS4.



Real-time network traffic analysis

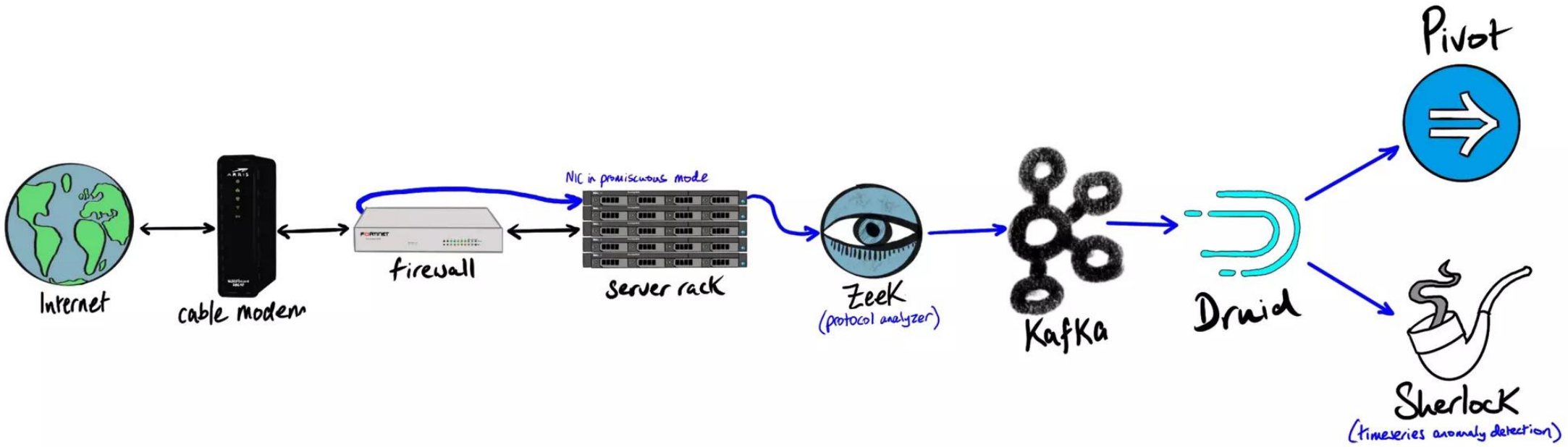
...with ZeeK, Kafka, Neo4j, and Elastic



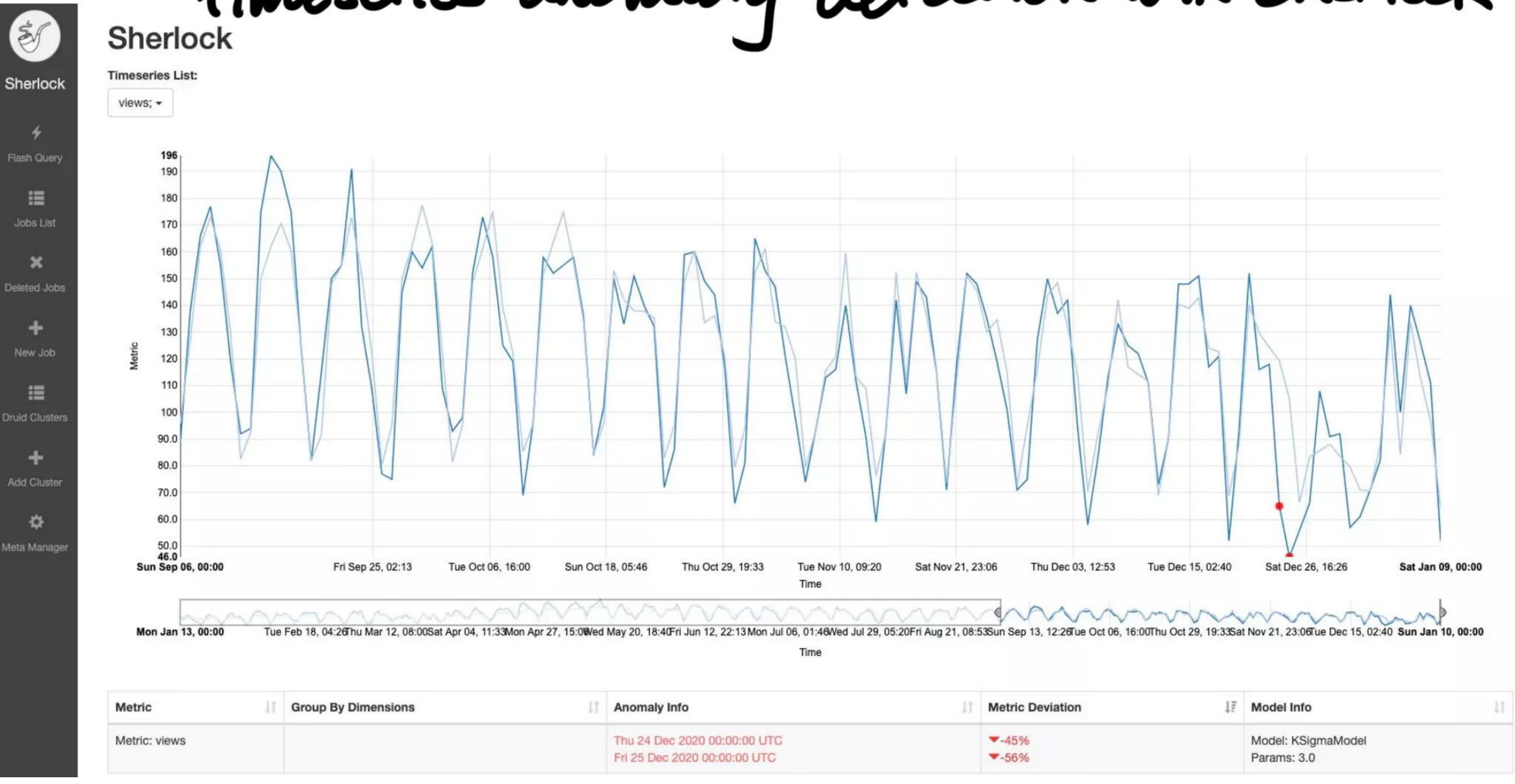

Segments
events
Indexer
Indexer
Indexer
Historicals
Historicals
Historicals
row
column

Real-time network traffic analysis

... with Zeek, Kafka, Druid, and Pivot

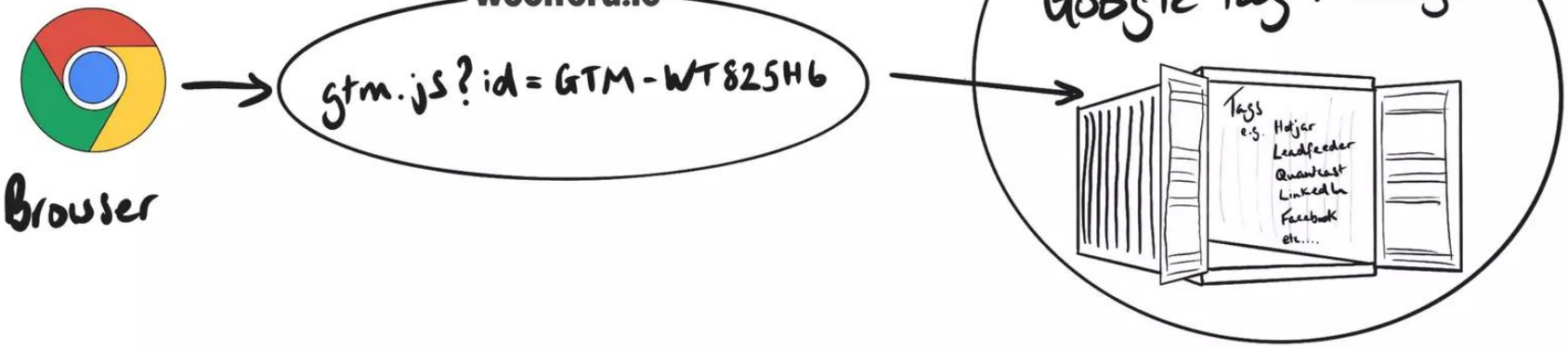


timeseries anomaly detection with Sherlock

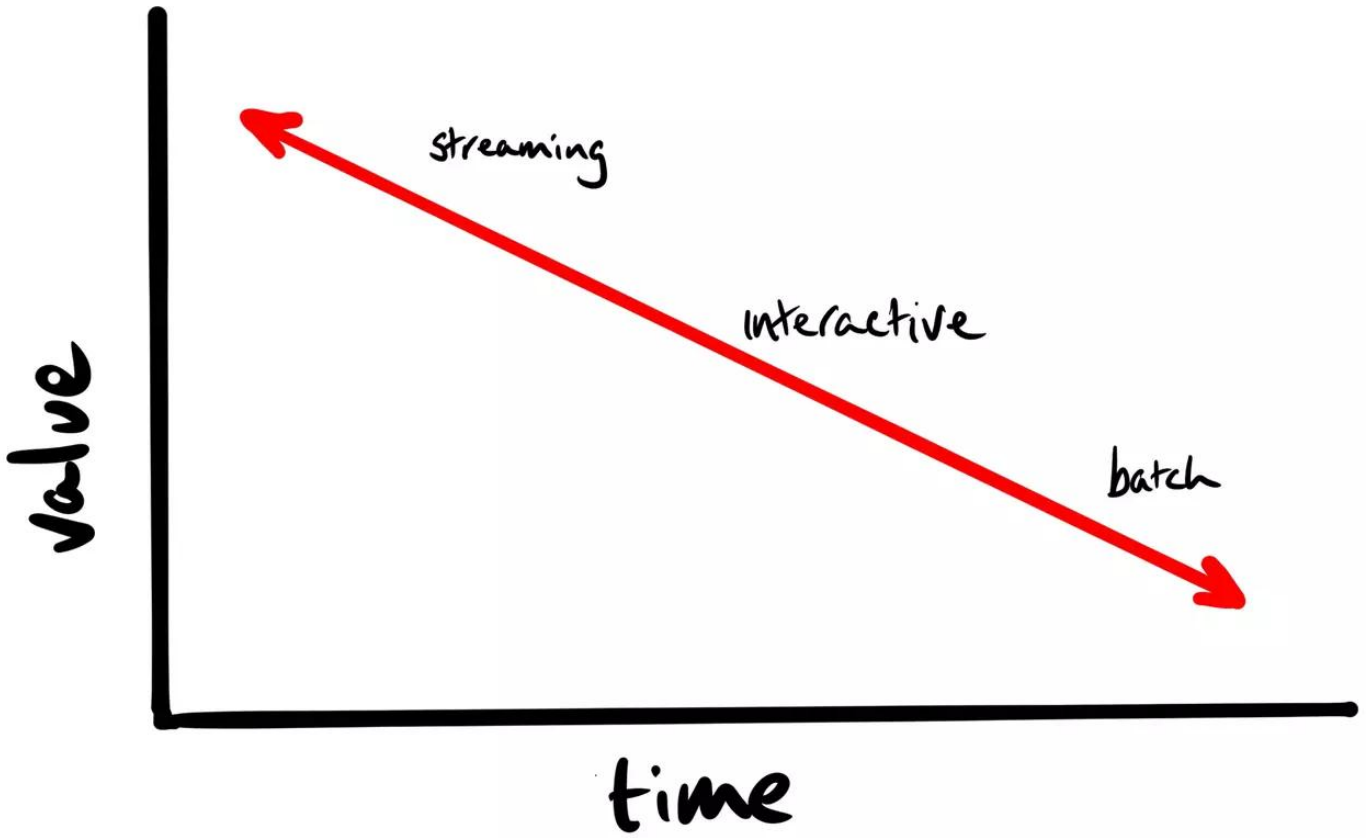


Metric	Group By Dimensions	Anomaly Info	Metric Deviation	Model Info
Metric: views		Thu 24 Dec 2020 00:00:00 UTC Fri 25 Dec 2020 00:00:00 UTC	▼-45% ▼-56%	Model: KSigmaModel Params: 3.0

Javascript tags

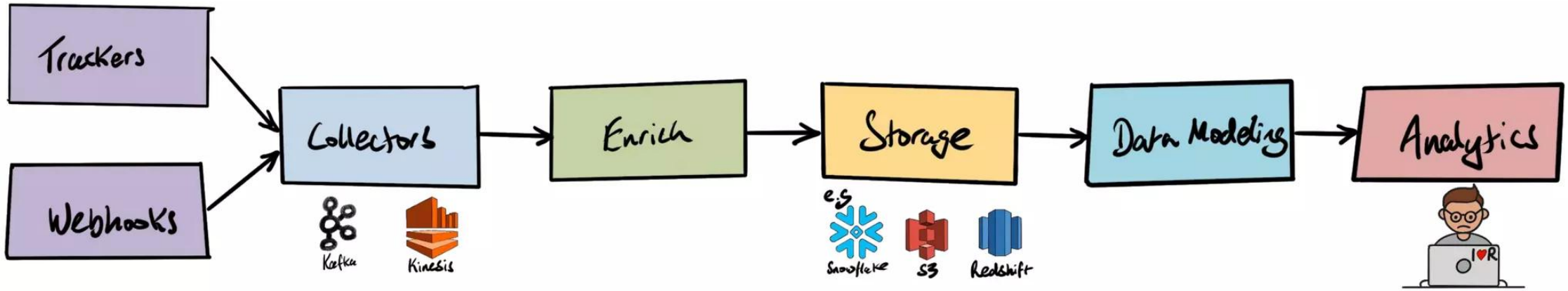


time-value of data

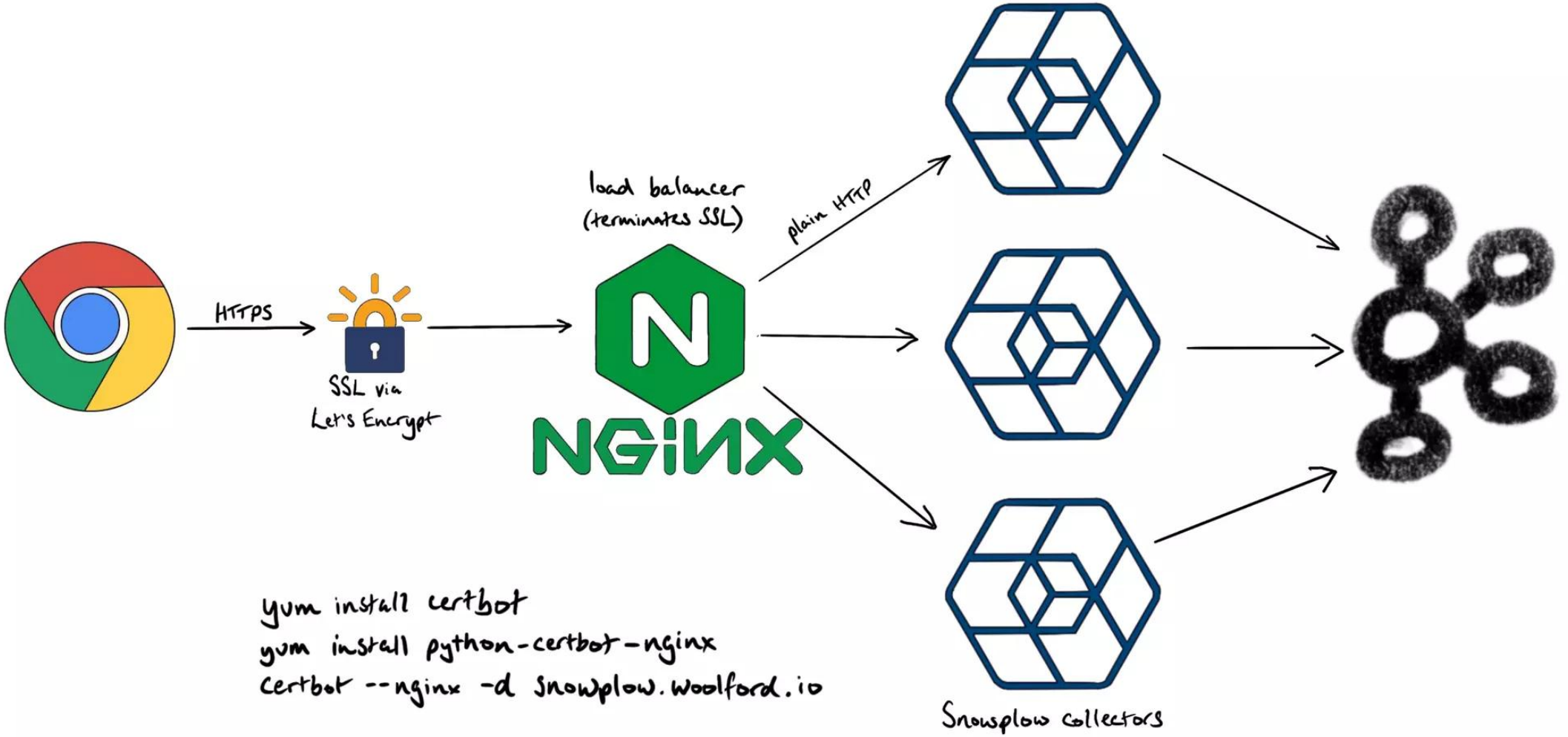


Source: Mike Franklin, UC Berkeley

Snowplow Analytics



SSL with Let's Encrypt; terminated by Nginx




```
yum install certbot
yum install python-certbot-nginx
certbot --nginx -d snowplow.woolford.io
```

Visualizing Kafka Streams topologies

...

`final Topology = builder.build;`

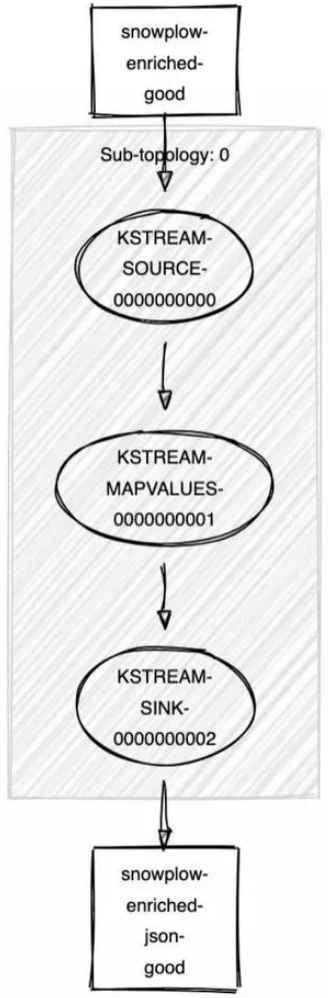
↓

paste the output from `topology.describe()` into

↓

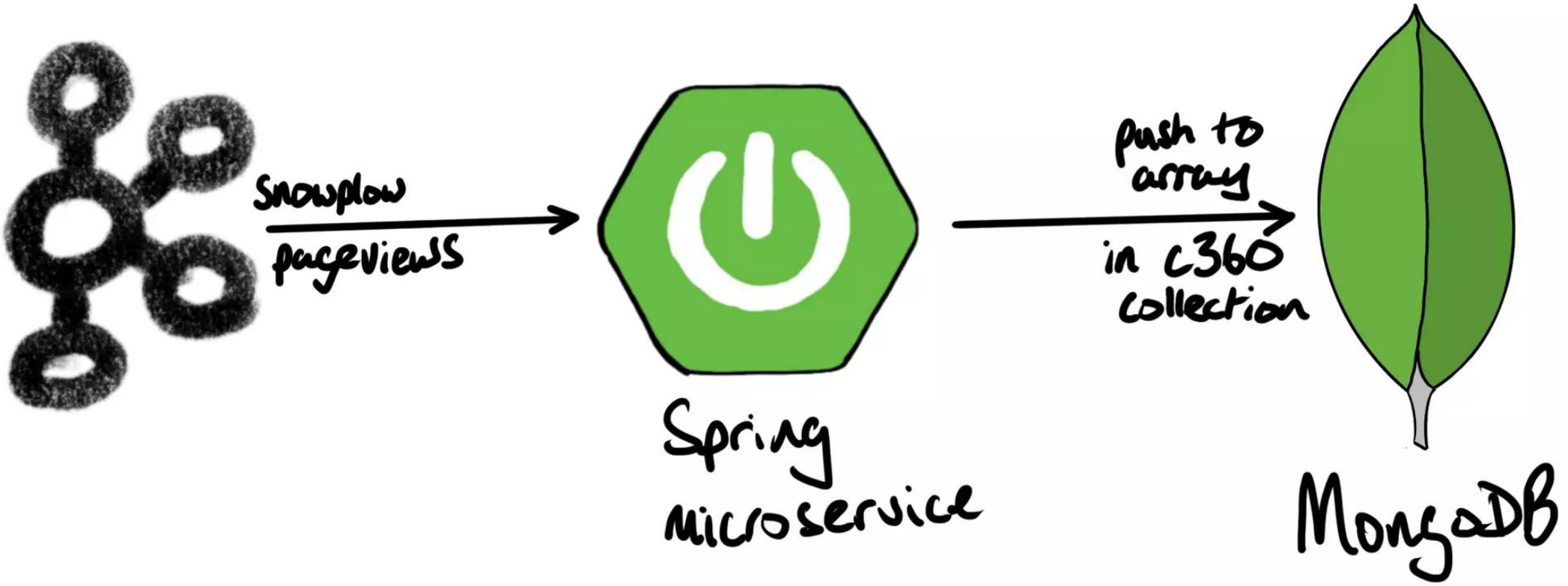
zz85.github.io/kafka-streams-viz

```
Sub-topology: 0
 Source: KSTREAM-SOURCE-0000000000 (topics: [snowplow-enriched-good])
   --> KSTREAM-MAPVALUES-0000000001
 Processor: KSTREAM-MAPVALUES-0000000001 (stores: [])
   --> KSTREAM-SINK-0000000002
   <-- KSTREAM-SOURCE-0000000000
 Sink: KSTREAM-SINK-0000000002 (topic: snowplow-enriched-json-good)
   <-- KSTREAM-MAPVALUES-0000000001
```



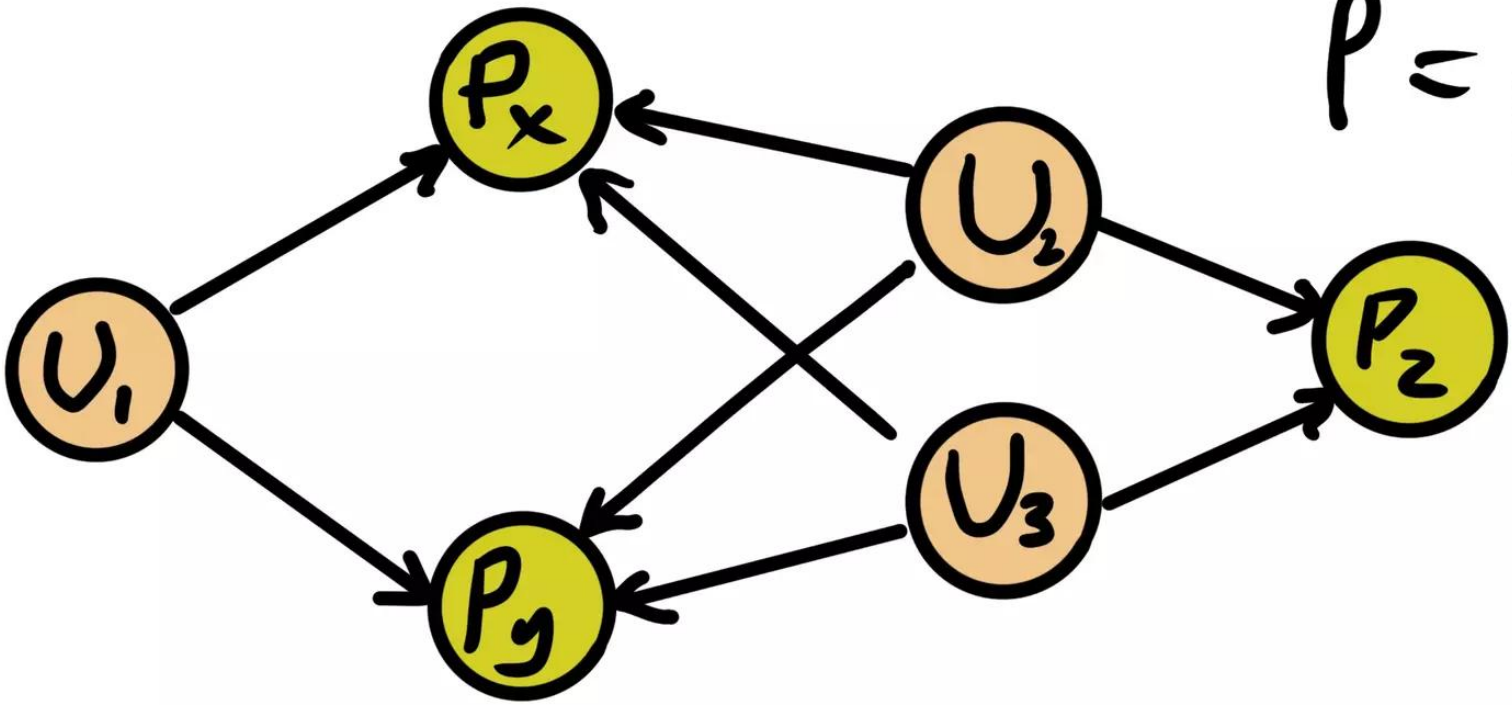
Kafka Streams Topology

build profiles



Graph-based recommender

U = user
P = page

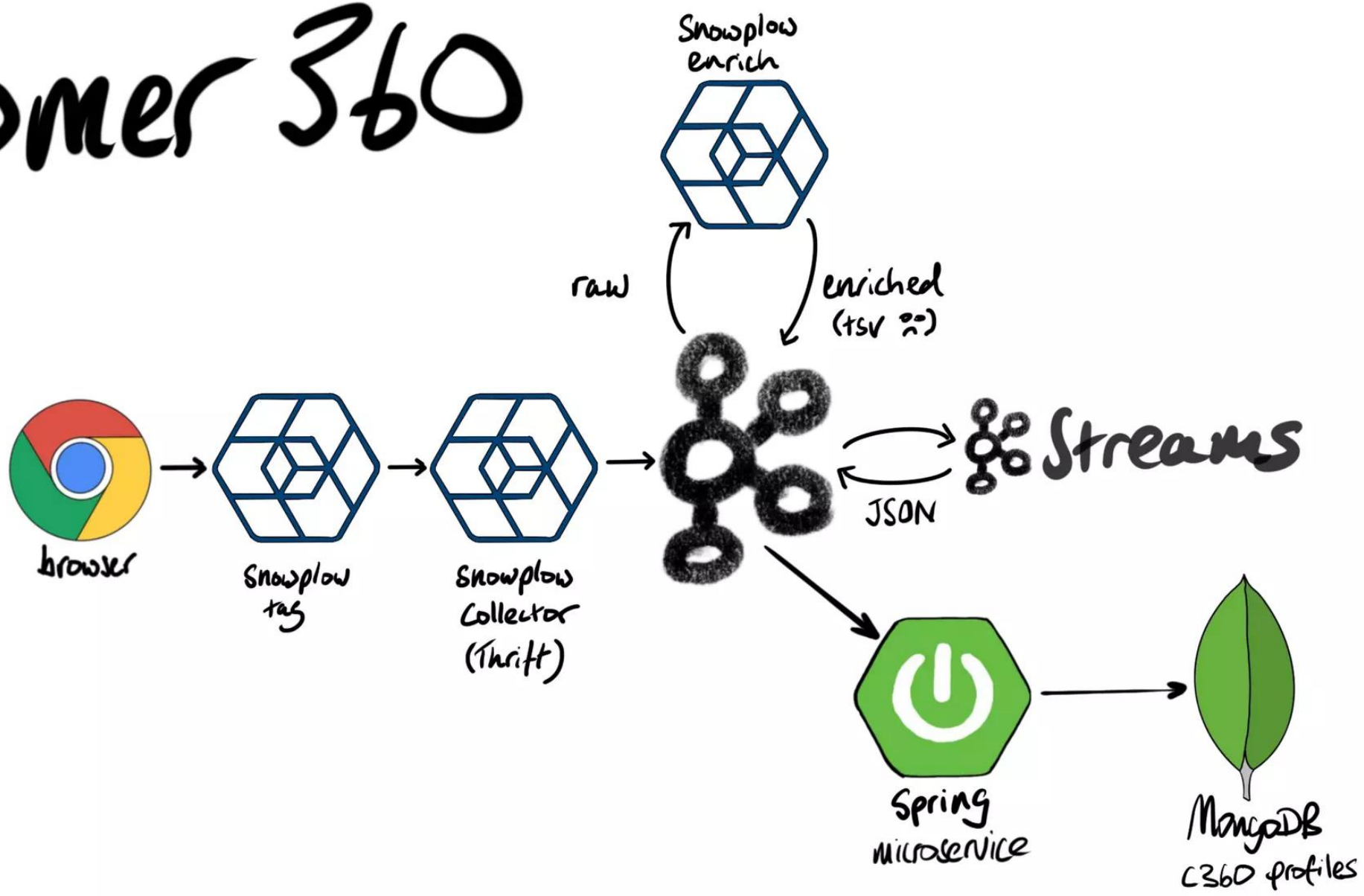


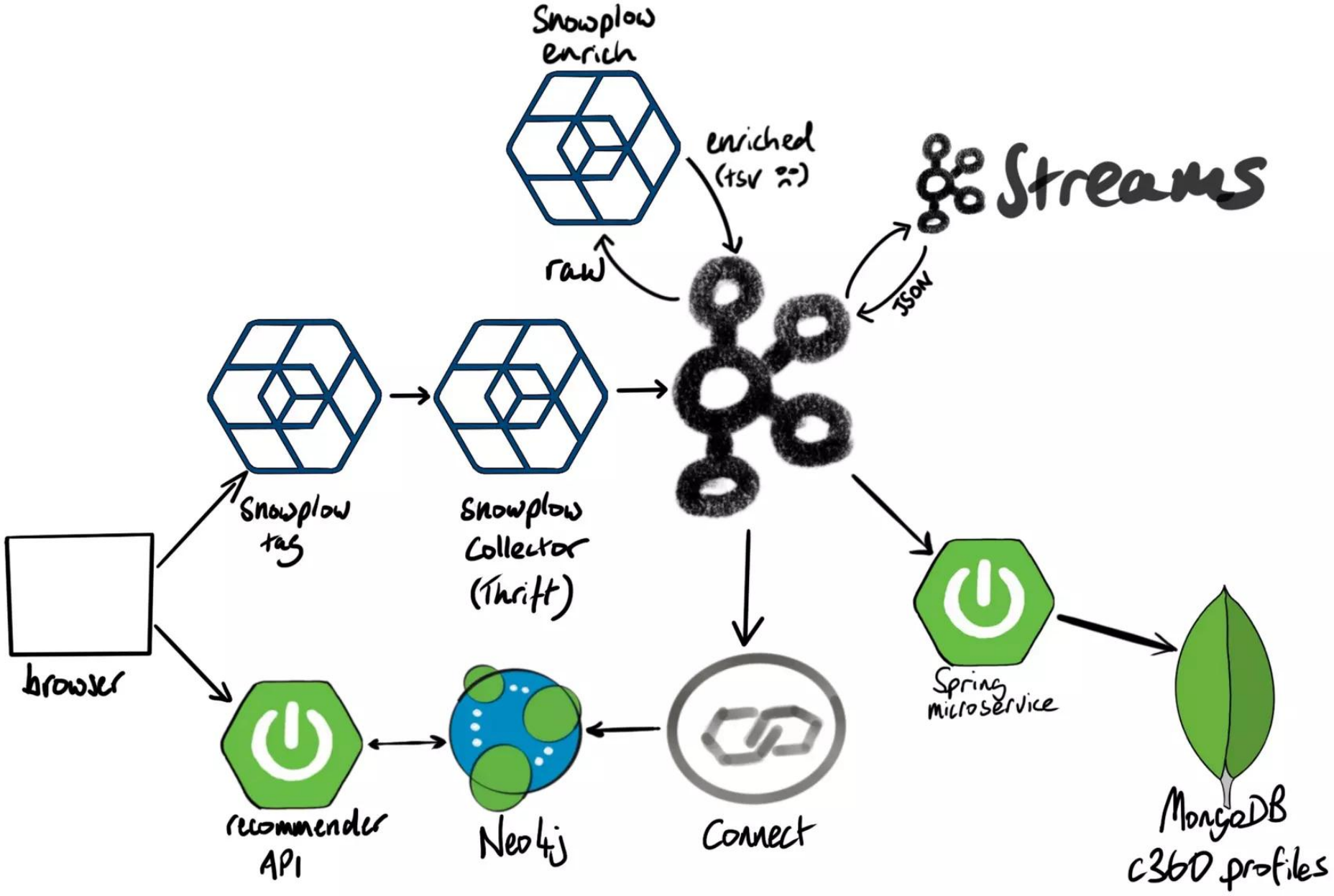
Cypher statement to build the graph

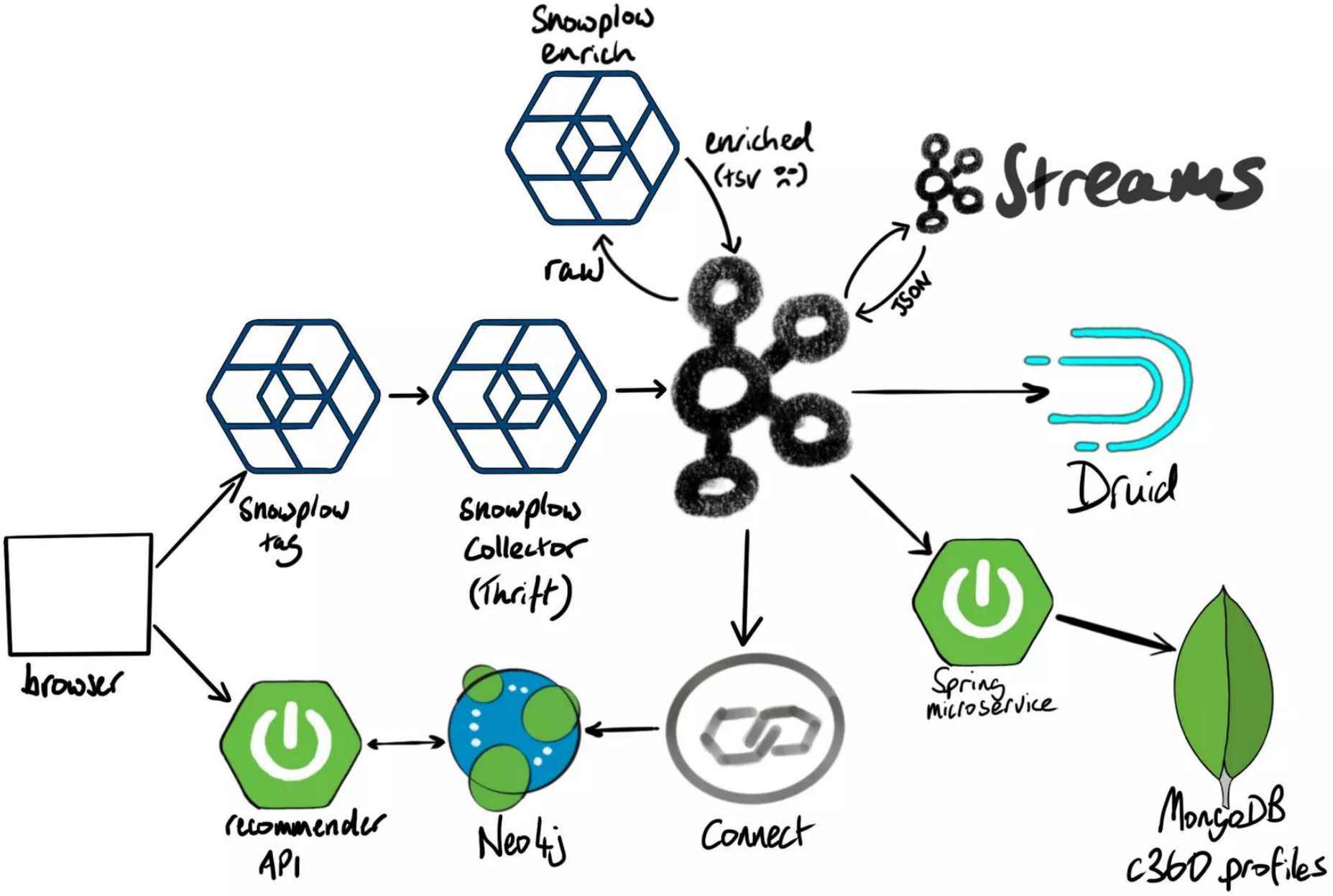
```
merge (network_userid:network_userid {id: event.network_userid})
merge (page_url:page_url {id: event.page_url})
merge (network_userid)-[:VIEWED]->(page_url)
```

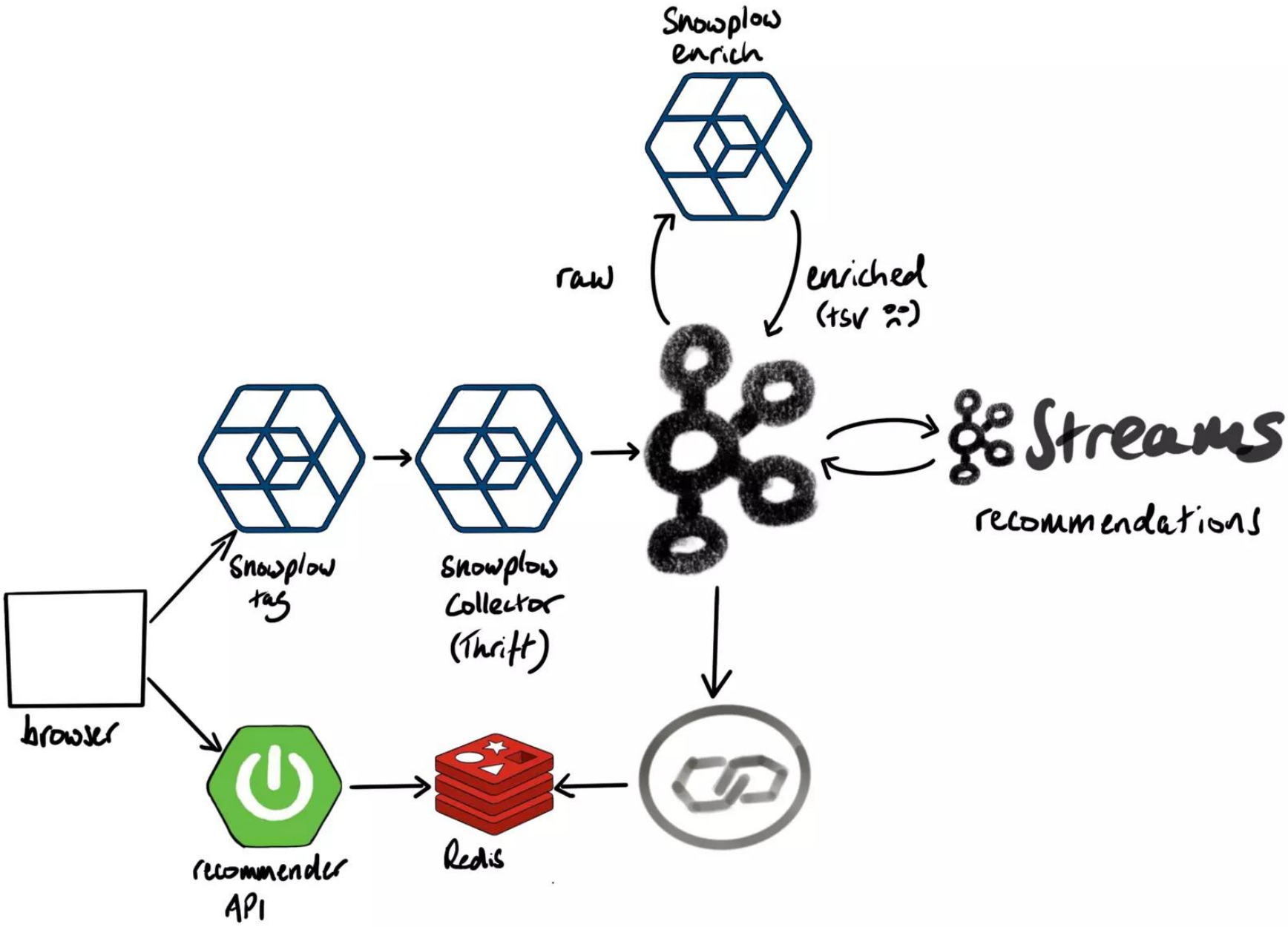
1. get or create user
2. get or create page
3. create relationship

Customer 360

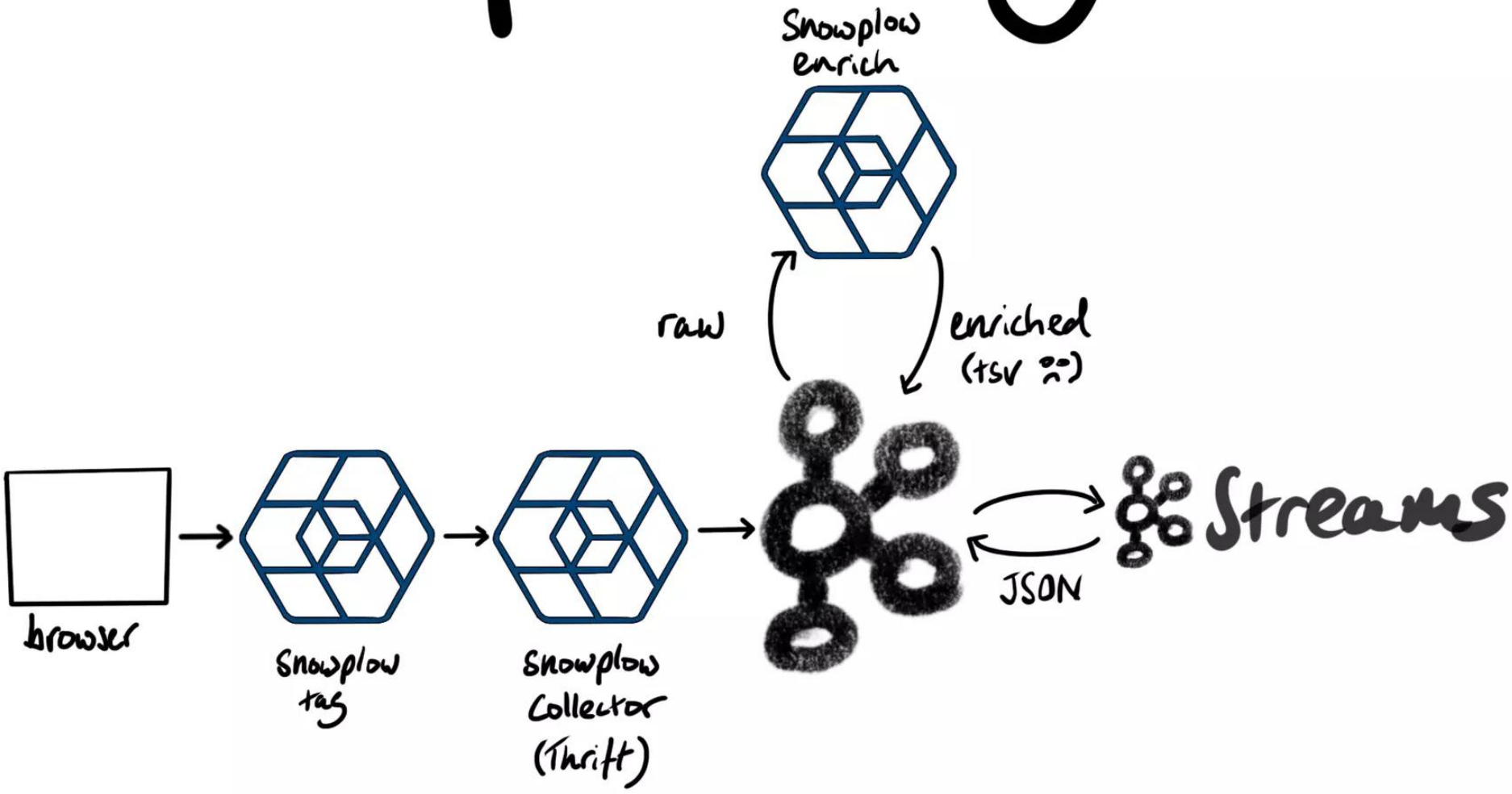



Snowplow enrich
enriched (tsv :-/)
Streams
raw
JSON
Snowplow tag
Snowplow Collector (Thrift)
browser
Spring microservice
recommender API
Neo4j
Connect
MongoDB
C360 profiles

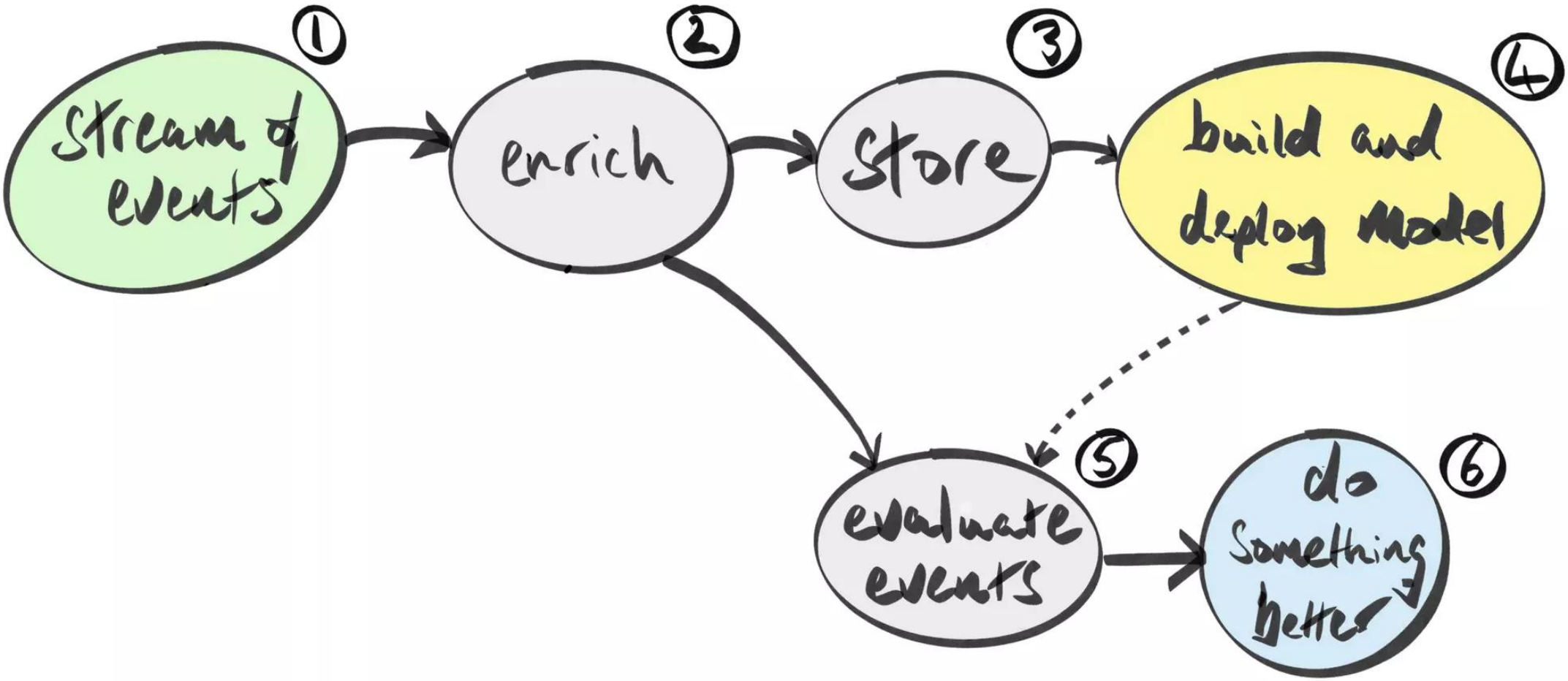

Snowplow enrich
enriched (tsv ☹)
raw
Streams
JSON
Druid
Snowplow tag
Snowplow Collector (Thrift)
browser
Spring microservice
MongoDB c360 profiles
recommender API
Neo4j
Connect

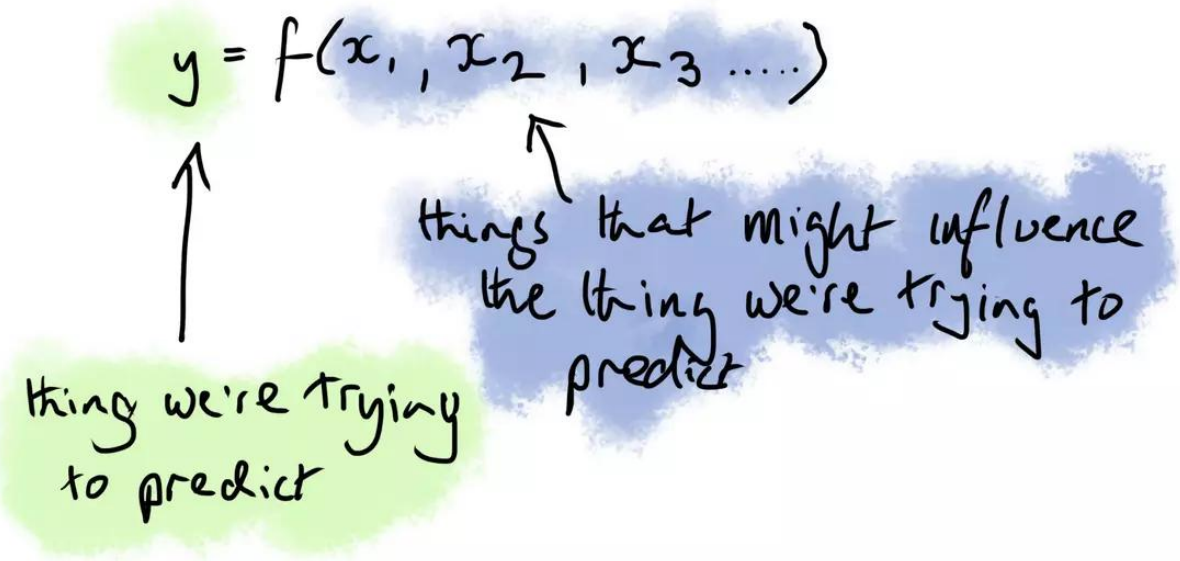

Snowplow
enrich
raw
enriched
(tsv 😞)
Streams
recommendations
Snowplow
tag
Snowplow
Collector
(Thrift)
browser
recommender
API
Redis

Snowplow ingest



putting AI into streams



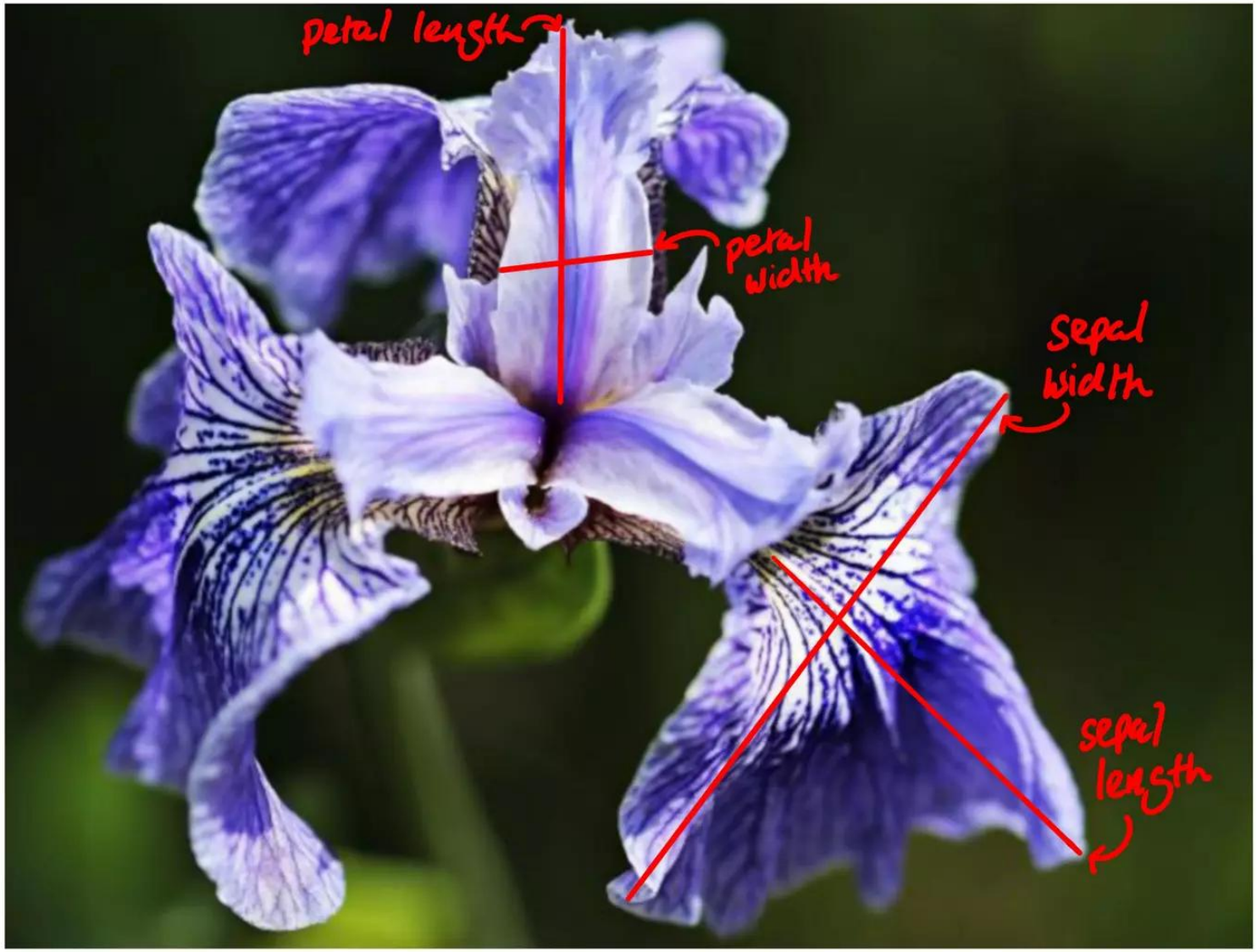


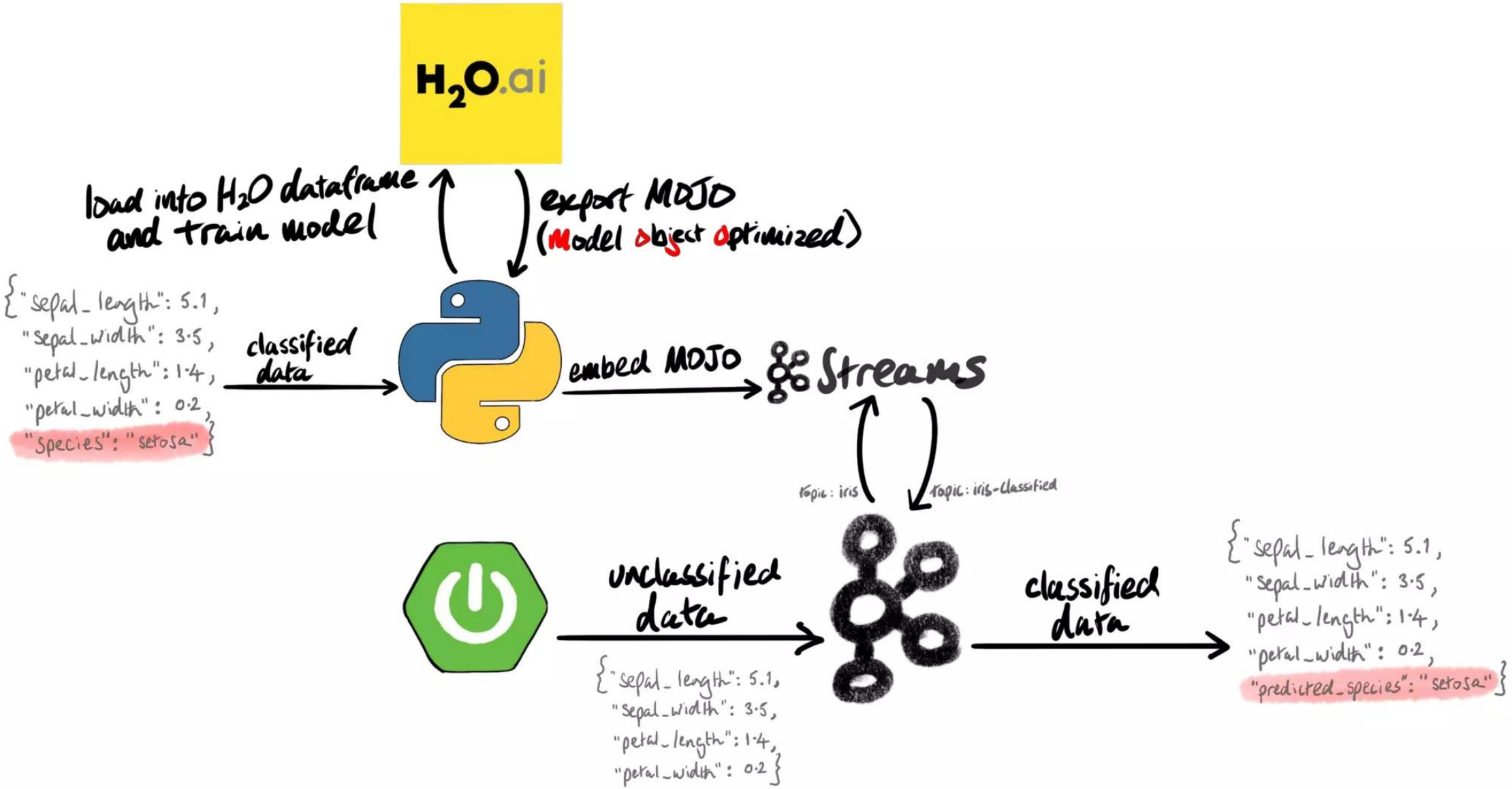
Step 1) create lots of candidate models.

Step 2) choose the best one.

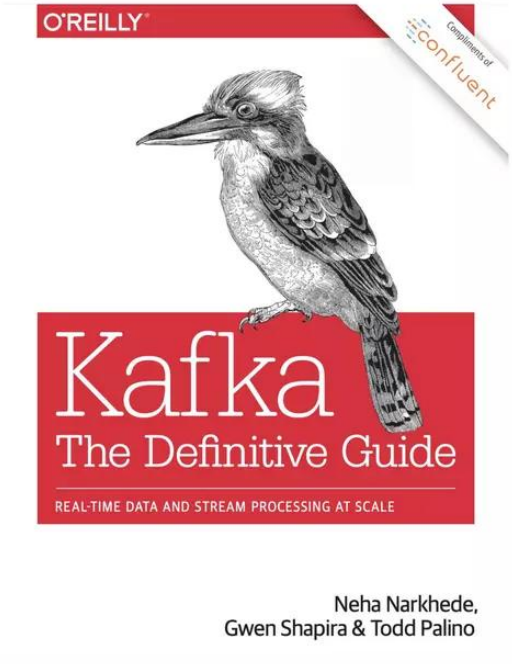
Step 3) stick it in the stream.

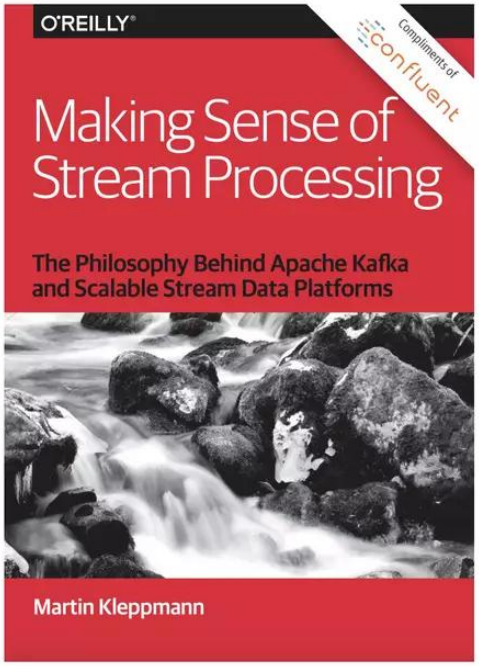
The iris dataset

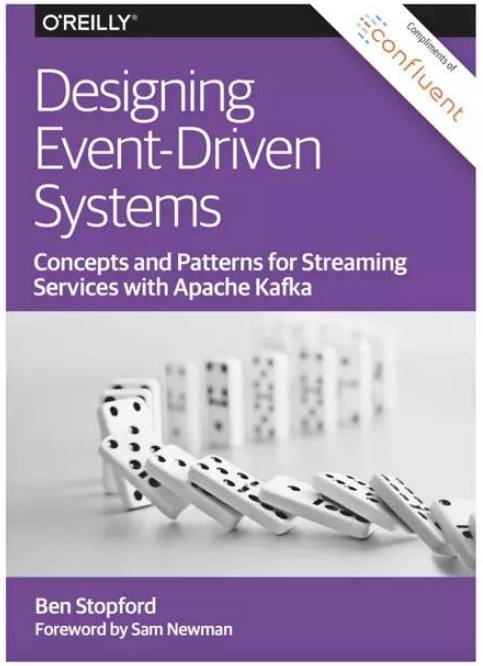


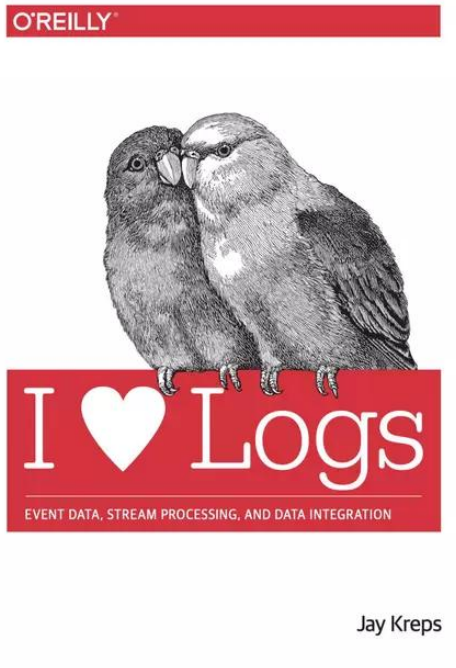

H2O.ai
load into H2O dataframe and train model
export MOJO (model object optimized)
{"sepal_length": 5.1,
"sepal_width": 3.5,
"petal_length": 1.4,
"petal_width": 0.2,
"species": "setosa"}
classified data
embed MOJO
Streams
topic: iris
topic: iris-classified
unclassified data
{"sepal_length": 5.1,
"sepal_width": 3.5,
"petal_length": 1.4,
"petal_width": 0.2}
classified data
{"sepal_length": 5.1,
"sepal_width": 3.5,
"petal_length": 1.4,
"petal_width": 0.2,
"predicted_species": "setosa"}

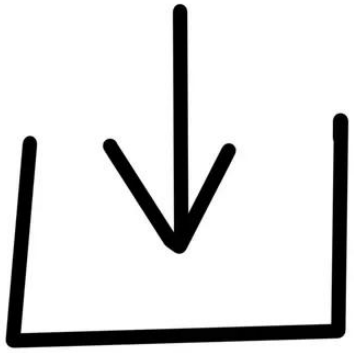
confluent.io/apache-kafka-stream-processing-book-bundle



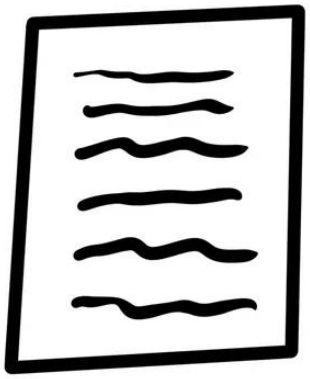




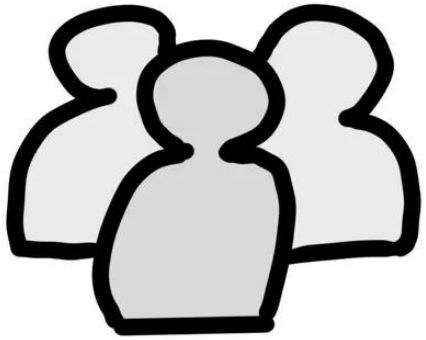




cnfl.io/download



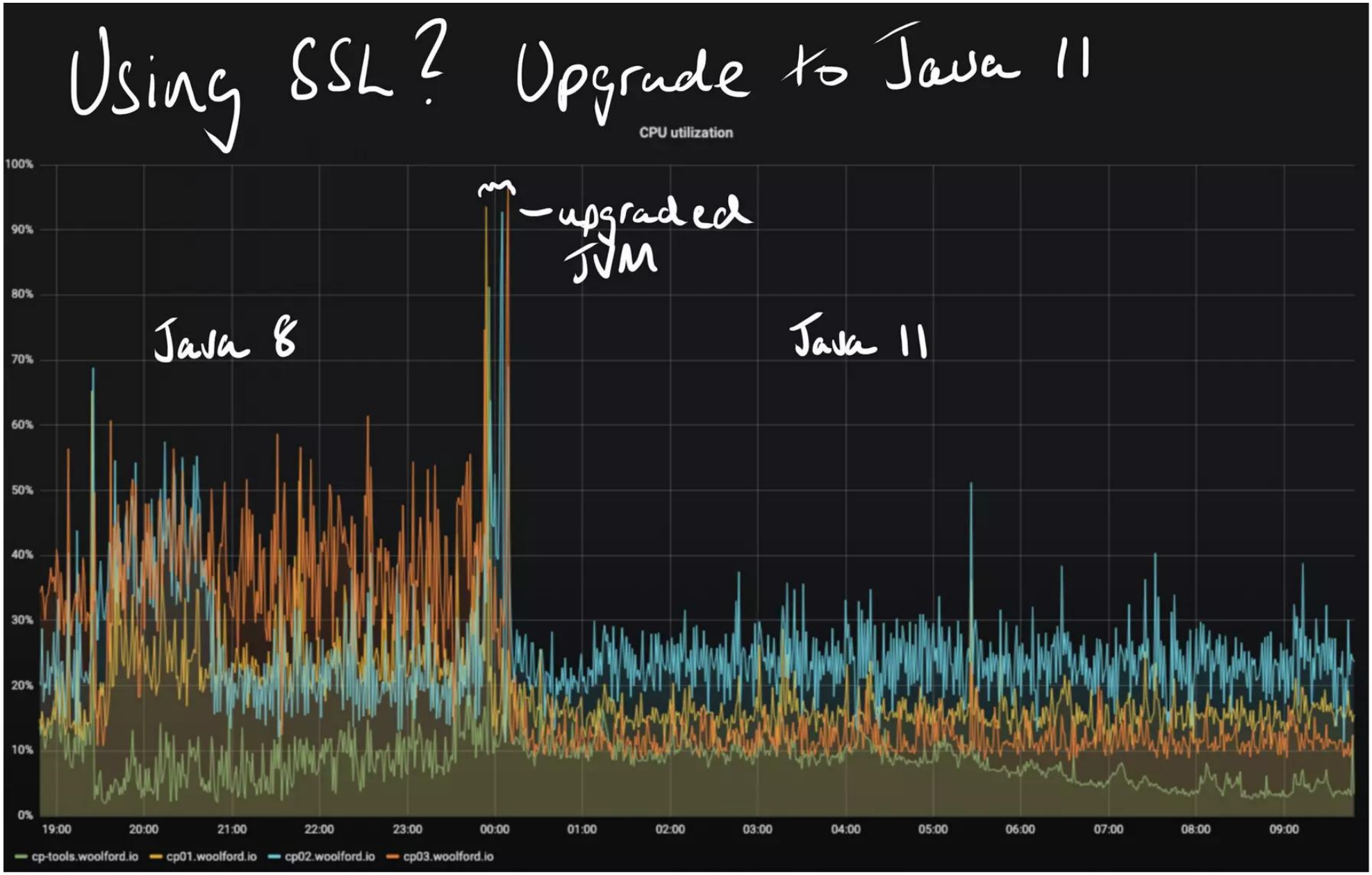
cnfl.io/blog



cnfl.io/meetups

LinkedIn: alexwoolford

Twitter: @bigsgus_data


Using SSL? Upgrade to Java 11
CPU utilization
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
upgraded JVM
Java 8
Java 11
19:00
20:00
21:00
22:00
23:00
00:00
01:00
02:00
03:00
04:00
05:00
06:00
07:00
08:00
09:00
cp-tools.woolford.io
cp01.woolford.io
cp02.woolford.io
cp03.woolford.io

Avro and Schema Registry

strict ordering in Kafka

exactly once

transactions

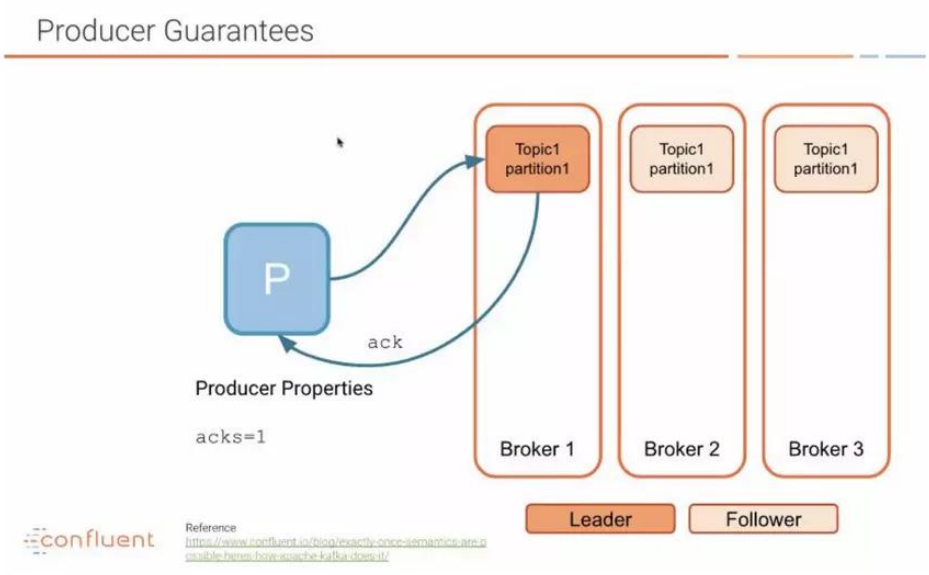
monitoring with Prometheus, Datadog, Burrow, Lightbend Lag Exporter

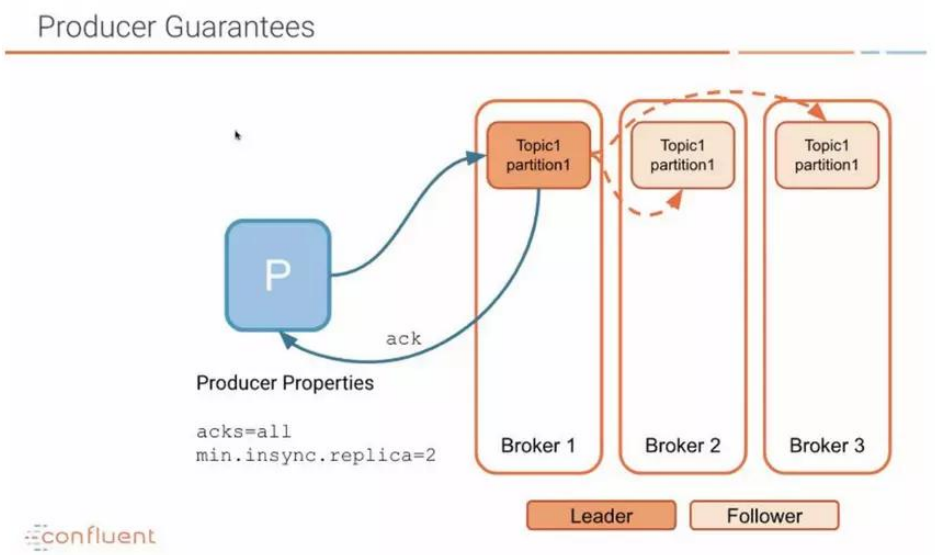
tailing log files with FileBeat

consumer offsets

zero copy, sequential reads, and the page cache

producer request configuration properties





consumer request configuration properties

bridge to cloud example

OSQuery

Persistent stores, punctuate, grace period, suppress

timeseries analysis

How to run KSQL

CLI
REST
C3
headless?

Terminology

- **Stream** - unbounded, continuously updating data set
- **Stream partition** - ordered, replayable, and fault-tolerant sequence of immutable data records
- **Data record** - key-value pair
- **Topology** - computational logic of the data processing (typically expressed by a DAG)
- **Stream processor** - node in the topology (i.e. processing step). It can be declarative (DSL) or imperative (PAPI)
- **KStream** - abstraction of a record stream
- **KTable** - abstraction of a changelog stream
- **KGlobalTable** - KTable with the data populated from all partitions

Agenda

- Developing Streams Apps
  - Streams and Tables
  - Topology
  - Serdes
  - Joins and windows
  - Error handling
  - Suppress
  - Exposing interactive queries
  - Testing
  - Tricks
- How Streams works
  - Threading model
  - State store
  - Key configuration
  - Processing guarantees
- Running Streams Apps
  - Monitoring
  - Security

DSL vs PAPI

https://docs.confluent.io/current/streams/developer-guide/dsl-api.html

https://docs.confluent.io/current/streams/developer-guide/processor-api.html

DSL API

- Preferred option for most of the use cases
- 9 out of 10 developers pick the DSL

```
KStream<String, String> textLines = ...;

KStream<String, Long> wordCounts = textLines
    // Split each text line, by whitespace, into words.  The text lines are the record
    // values, i.e. you can ignore whatever data is in the record keys and thus invoke
    // `flatMapValues` instead of the more generic `flatMap`.
    .flatMapValues(value -> Arrays.asList(value.toLowerCase().split("\\W+")))
    // Group the stream by word to ensure the key of the record is the word.
    .groupBy((key, word) -> word)
    // Count the occurrences of each word (record key).
    //
    // This will change the stream type from `KGroupedStream<String, String>` to
    // `KTable<String, Long>` (word -> count).
    .count()
    // Convert the `KTable<String, Long>` into a `KStream<String, Long>`.
    .toStream();
```

Processor API

- Same logic as previous code
- Full flexibility but much more manual work
- DSL and Processor API can be mixed (i.e. use DSL whenever possible and Processor for exceptions)

```java
public class WordCountProcessor implements Processor<String, String> {

  private ProcessorContext context;
  private KeyValueStore<String, Long> kvStore;

  @Override
  @SuppressWarnings("unchecked")
  public void init(ProcessorContext context) {
      // keep the processor context locally because we need it in punctuate() and commit()
      this.context = context;

      // retrieve the key-value store named "Counts"
      kvStore = (KeyValueStore) context.getStateStore("Counts");

      // schedule a punctuate() method every second based on event-time
      this.context.schedule(Duration.ofSeconds(1), PunctuationType.STREAM_TIME, (timestamp) -> {
          KeyValueIterator<String, Long> iter = this.kvStore.all();
          while (iter.hasNext()) {
              KeyValue<String, Long> entry = iter.next();
              context.forward(entry.key, entry.value.toString());
          }
          iter.close();

          // commit the current processing progress
          context.commit();
      });
  }

  @Override
  public void process(String dummy, String line) {
      String[] words = line.toLowerCase(Locale.getDefault()).split(" ");

      for (String word : words) {
          Integer oldValue = this.kvStore.get(word);

          if (oldValue == null) {
              this.kvStore.put(word, 1);
          } else {
              this.kvStore.put(word, oldValue + 1);
          }
      }
  }

  @Override
  public void close() {
      // nothing to do
  }

}
```

Should I use DSL or the Processor API ?

“DSL looks awesome but I feel limited, I need more control.”

“**PAPI** looks simpler, powerful, and I have more control on things”

“With great power comes great responsibility !”

Don’t need to go one way or the other we can mix both the **DSL** and **PAPI**.

Before using **PAPI**, start thinking if there is no other way to implement it.

Most probably you are not thinking in a Event Streaming way yet !

Naming the topology

No naming

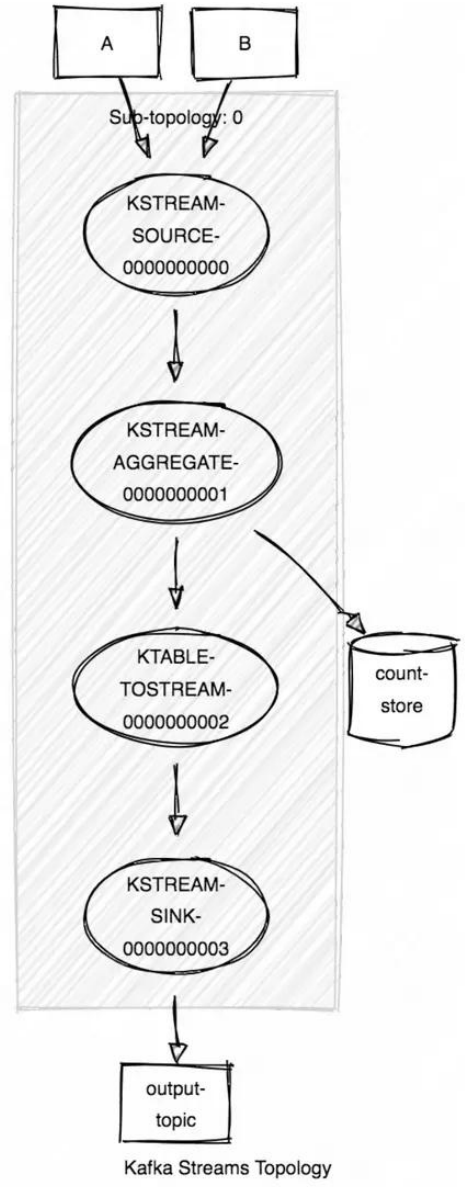
```
KStream<String,String> stream =
builder.stream("input");

stream.filter((k,v) -> !v.equals("invalid_txn"))

      .mapValues((v) -> v.substring(0,5))

      .to("output")
```

With naming

```
KStream<String,String> stream =

builder.stream("input",
Consumed.as("Customer_transactions_input_topic"));

stream.filter((k,v) -> !v.equals("invalid_txn"),
Named.as("filter_out_invalid_txns"))

      .mapValues((v) -> v.substring(0,5),
Named.as("Map_values_to_first_6_characters"))

      .to("output",
Produced.as("Mapped_transactions_output_topic"));
```

Visualize the topology!

https://zz85.github.io/kafka-streams-viz/



Kafka Streams Topology

Map vs MapValues

Map vs Map Values

```
final StreamsBuilder builder = new StreamsBuilder();
final KTable<Long, Long> count = builder
        .stream(inputTopic, Consumed.with(Serdes.Long(), Serdes.String()))
        .map((key, val) -> new KeyValue<>(key, "new value for " + val))
        .groupByKey()
        .count();

count
        .toStream()
        .to(outputTopic, Produced.with(Serdes.Long(), Serdes.Long()));

Topology topology = builder.build();
```

Here, even though the key is unchanged, which will trigger a repartition because as Kafka Streams will flag the operation as if the key was modified.

Map vs Map Values

```
final StreamsBuilder builder = new StreamsBuilder();
final KTable<Long, Long> count = builder
        .stream(inputTopic, Consumed.with(Serdes.Long(), Serdes.String()))
        .mapValues(val -> "new value for " + val)
        .groupByKey()
        .count();

count
        .toStream()
        .to(outputTopic, Produced.with(Serdes.Long(), Serdes.Long()));

Topology topology = builder.build();
```

Now that we replaced the topology step with a mapValues, this won't trigger a repartition, leading to better performance.

Requirements

- Join on message keys
  - In Confluent 5.4 there is the support non-key joining on KTable (KIP-213)
- Co-partitioned
- Streams to Streams are time windowed

In Confluent 5.5 we introduce CoGroup (KIP-150) !

KIP-150: Add Cogroup to the DSL

- In the past, aggregating multiple streams into one could be complicated and error-prone.
- It generally requires you to group + aggregate all of the streams into tables, then make multiple outer join calls.
- Using the new co-group operator will:
  - Clean up the syntax of your programs
  - Reduce the number of state store invocations
  - Increase performance

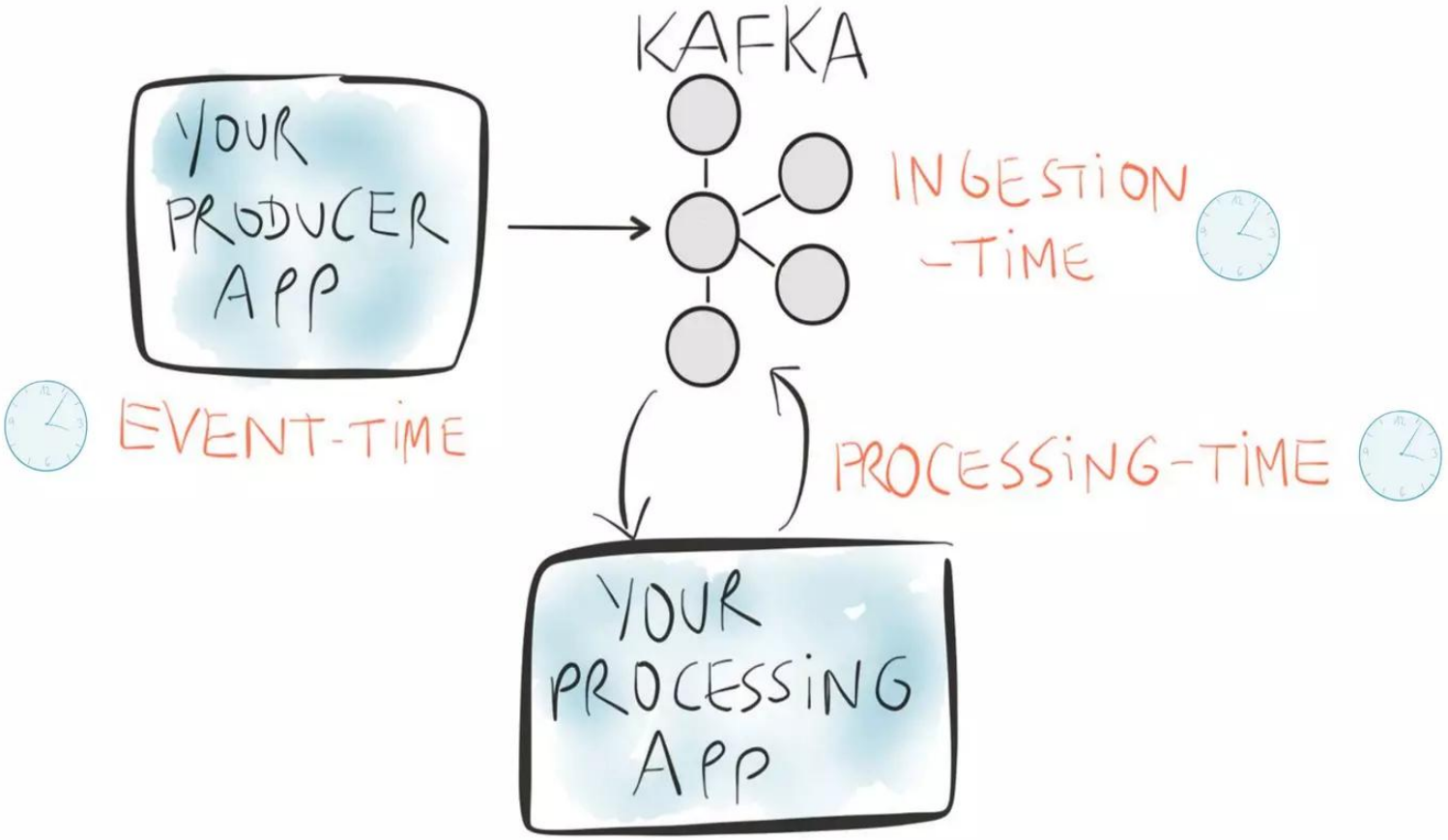
```
KGroupedStream<K, V1> grouped1 =
  builder.stream("topic1").groupByKey();

KGroupedStream<K, V2> grouped2 =
  builder.stream("topic2").groupByKey();

KGroupedStream<K, V3> grouped3 =
  builder.stream("topic3").groupByKey();


KTable<K, CG> cogrouped =
  grouped1
    .cogroup(aggregator1)
    .cogroup(grouped2, aggregator2)
    .cogroup(grouped3, aggregator3)
    .aggregate(initializer1, materialized1);
```

CP 5.5 / AK 2.5

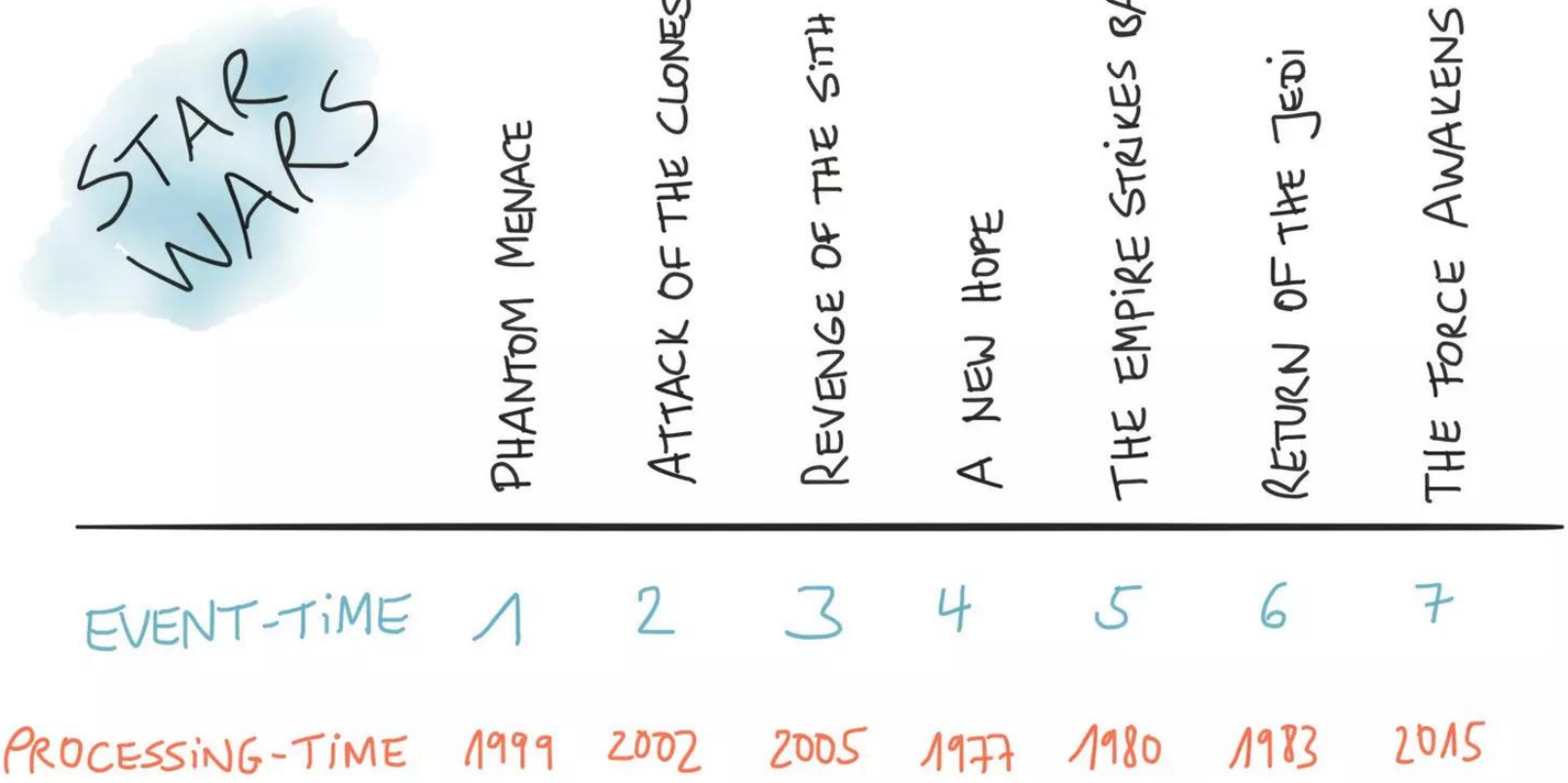
Time time time time !

https://docs.confluent.io/current/streams/concepts.html#time

Time


KAFKA
YOUR PRODUCER APP
INGESTION -TIME
EVENT-TIME
PROCESSING-TIME
YOUR PROCESSING APP

Time

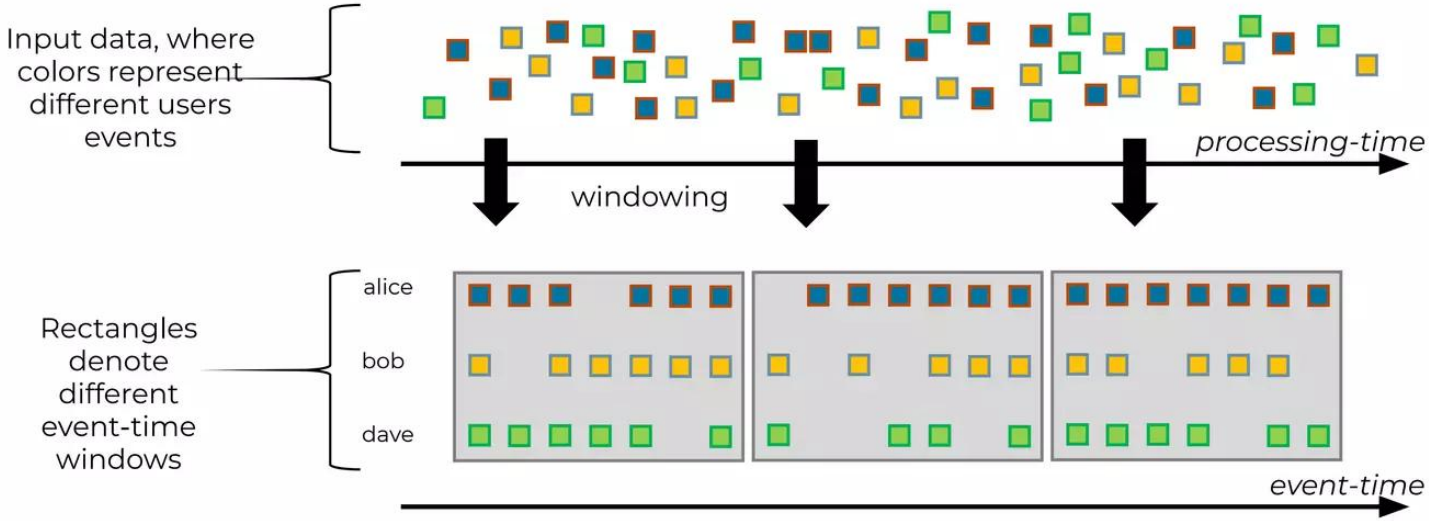

STAR WARS
PHANTOM MENACE
ATTACK OF THE CLONES
REVENGE OF THE SITH
A NEW HOPE
THE EMPIRE STRIKES BACK
RETURN OF THE JEDI
THE FORCE AWAKENS
EVENT-TIME
1
2
3
4
5
6
7
PROCESSING-TIME
1999
2002
2005
1977
1980
1983
2015

Windowing

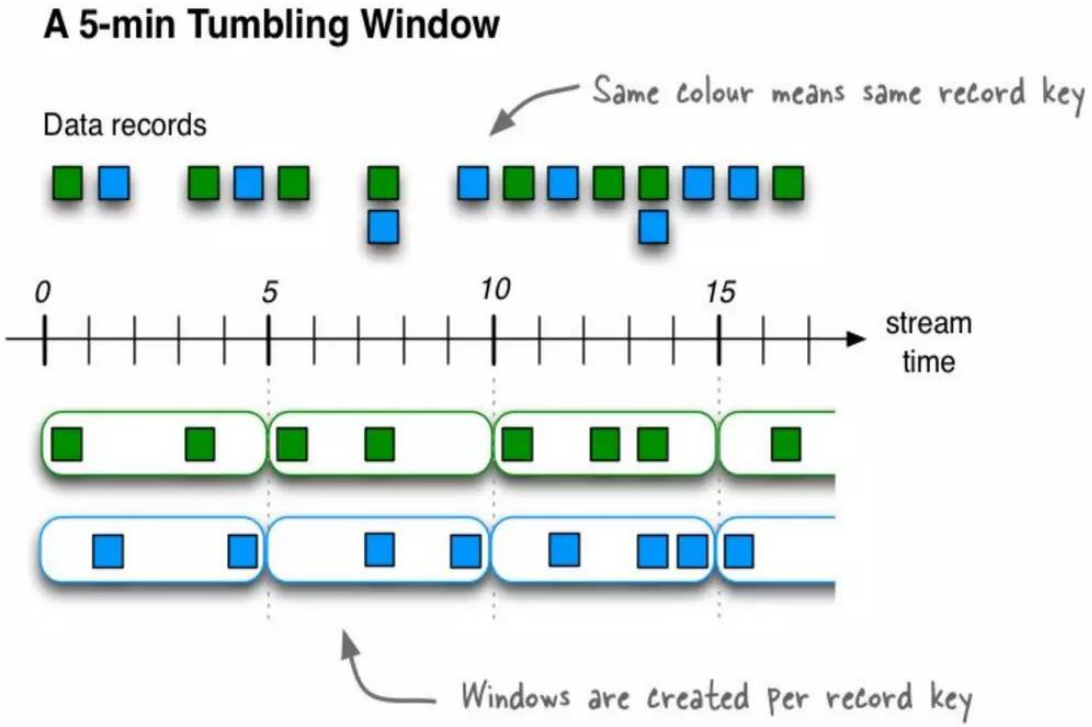
https://docs.confluent.io/current/streams/concepts.html#windowing

Windowing

- **To group related events in a stream**
- Use case examples:
  - Time-based analysis of ad impressions (”number of ads clicked in the past hour”)
  - Monitoring statistics of telemetry data (“1min/5min/15min averages”)
  - Analyzing user browsing sessions on a news site

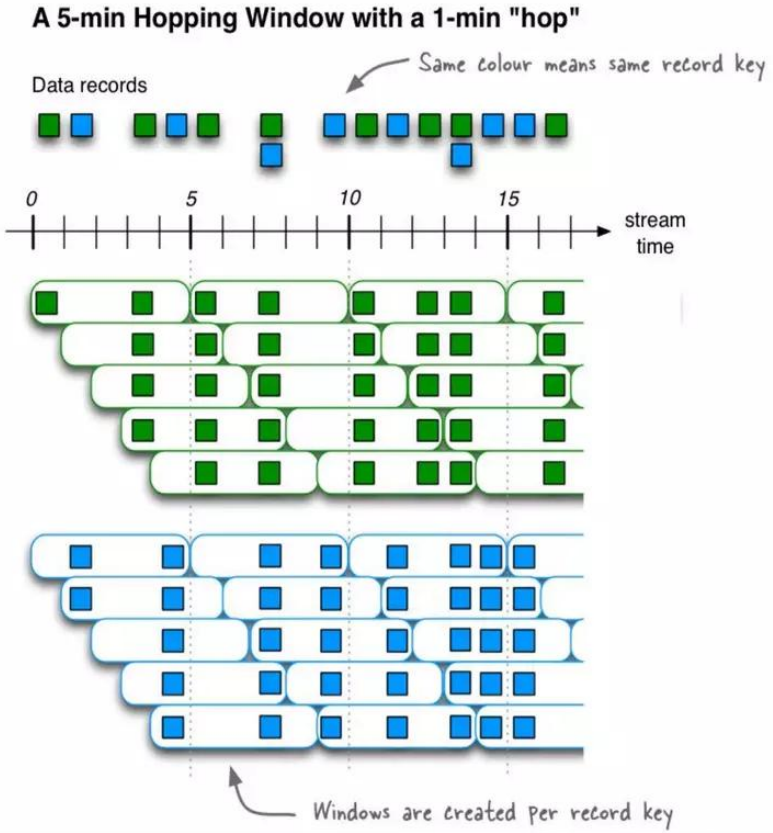


Tumbling window



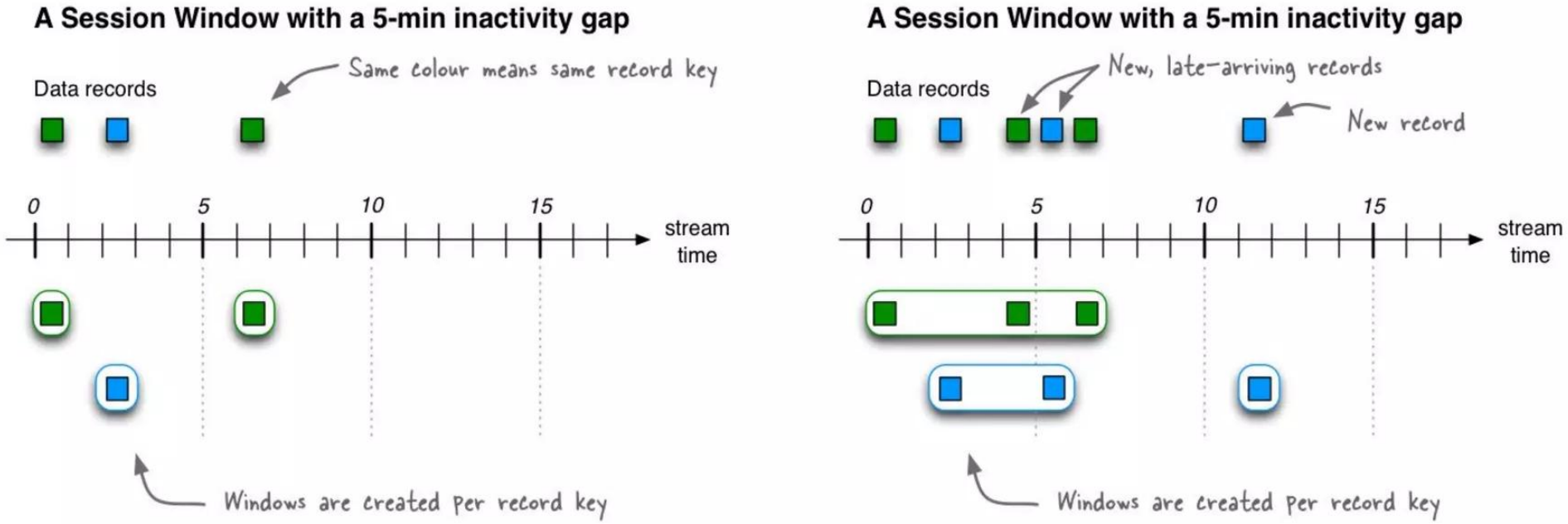

```
// Key (String) is user ID, value (Avro record) is the page view event for that user.
// Such a data stream is often called a "clickstream".
KStream<String, GenericRecord> pageViews = ...;

// Count page views per window, per user, with tumbling windows of size 5 minutes
KTable<Windowed<String>, Long> windowedPageViewCounts = pageViews
    .groupByKey(Grouped.with(Serdes.String(), genericAvroSerde))
    .windowedBy(TimeWindows.of(Duration.ofMinutes(5)))
    .count();
```

Hopping window




```
// Key (String) is user ID, value (Avro record) is the page view event for that user.
// Such a data stream is often called a "clickstream".
KStream<String, GenericRecord> pageViews = ...;

// Count page views per window, per user, with hopping windows of size 5 minutes that advance every 1 minute
KTable<Windowed<String>, Long> windowedPageViewCounts = pageViews
    .groupByKey(Grouped.with(Serdes.String(), genericAvroSerde))
    .windowedBy(TimeWindows.of(Duration.ofMinutes(5).advanceBy(Duration.ofMinutes(1))))
    .count()
```

Session window




```
// Key (String) is user ID, value (Avro record) is the page view event for that user.
// Such a data stream is often called a "clickstream".
KStream<String, GenericRecord> pageViews = ...;

// Count page views per session, per user, with session windows that have an inactivity gap of 5 minutes
KTable<Windowed<String>, Long> sessionizedPageViewCounts = pageViews
    .groupByKey(Grouped.with(Serdes.String(), genericAvroSerde))
    .windowedBy(SessionWindows.with(Duration.ofMinutes(5)))
    .count();
```

Kafka Streams: Write standard Java apps and microservices to process your data in real-time

- No separate processing cluster required
- Develop on Mac, Linux, Windows
- Deploy to containers, VMs, bare metal, cloud
- Powered by Kafka: elastic, scalable, distributed, battle-tested
- Perfect for small, medium, large use cases
- Fully integrated with Kafka security
- Exactly-once processing semantics
- Part of Apache Kafka

```
KStream<User, PageViewEvent> pageViews = builder.stream("pageviews-topic");
KTable<Windowed<User>, Long> viewsPerUserSession = pageViews
    .groupByKey()
    .count(SessionWindows.with(TimeUnit.MINUTES.toMillis(5)), "session-views");
```

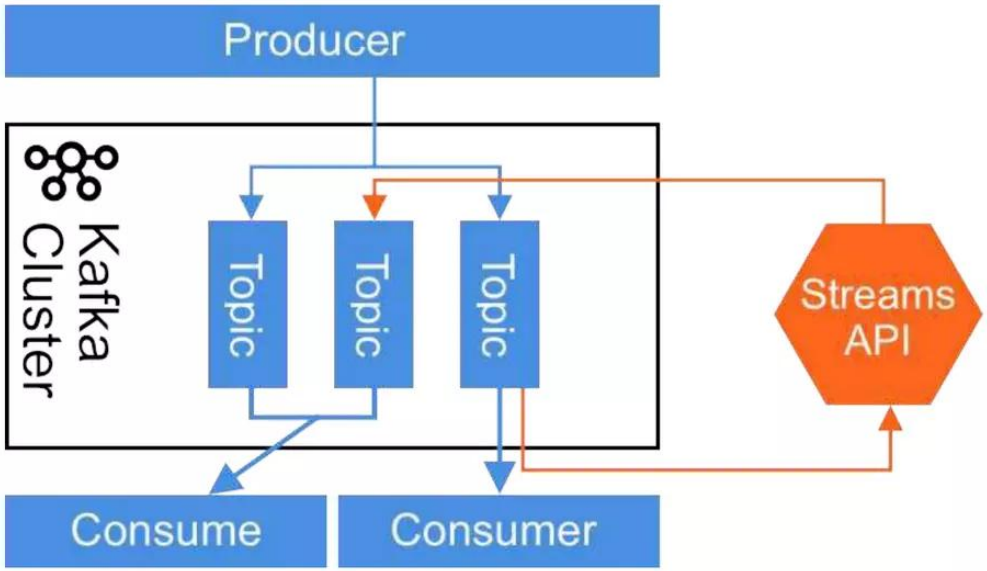
https://docs.confluent.io/current/streams/

Kafka Streams

Transform Data with Real-Time Applications

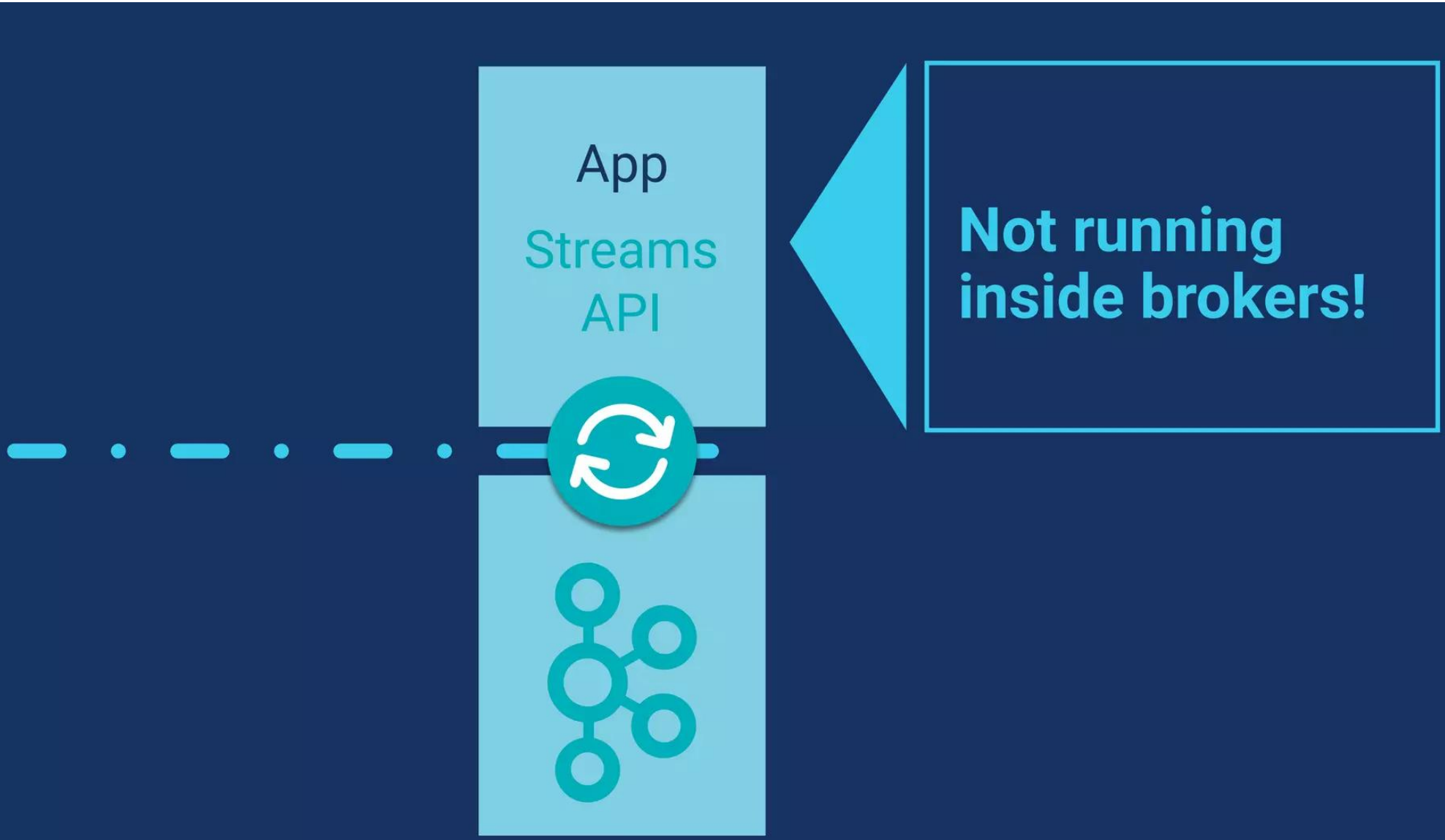
Overview

- Write standard Java applications
- No separate processing cluster required
- Exactly-once processing semantics
- Elastic, highly scalable, fault-tolerant
- Fully integrated with Kafka security



Example Use Cases

- Microservices
- Continuous queries
- Continuous transformations


App
Streams API
Not running inside brokers!

Things Kafka Streams Does

Runs everywhere

Clustering done for you

Exactly once processing

Event time processing

Integrated database

Joins, windowing, aggregation

S/M/L/XL/XXL/XXXL sizes

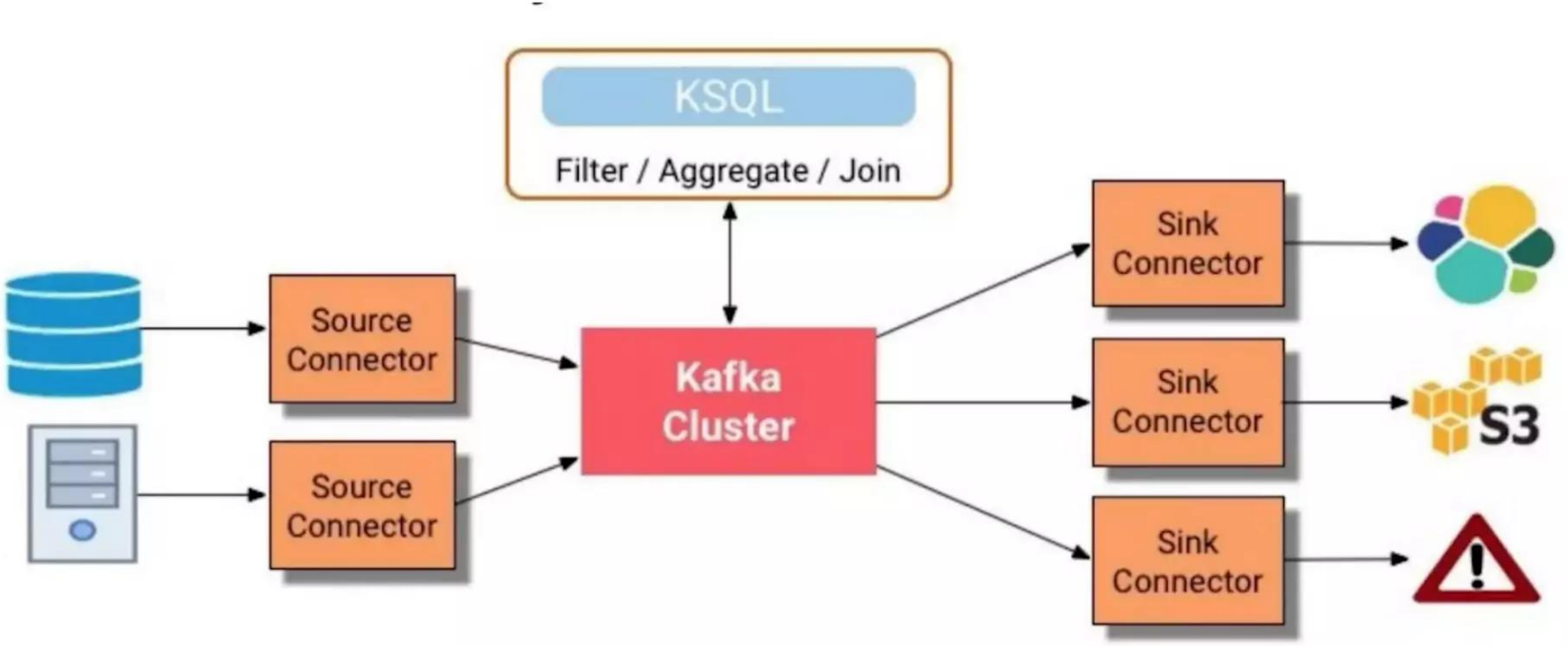
each consumer has its own position

offsets are stored in Kafka topic __consumer_offsets

Kafka Streams

- Java Library
- no out-of-the-box runnable application
- Build Jars to deploy
- No interactivity

KSQL Server


KSQL
Filter / Aggregate / Join
Source Connector
Source Connector
Kafka Cluster
Sink Connector
Sink Connector
Sink Connector
S3

Types of KSQL statement - p1

Create Data Sources:

- CREATE TABLE …
- CREATE STREAM …

- SHOW TABLES; SHOW STREAMS;

Transforming, persistent queries

Types of KSQL statement - p2

Metadata management:

- SHOW STREAMS / TABLES / QUERIES
- DESCRIBE STREAM / TABLE
- EXPLAIN QUERY
- DROP STREAM / TABLE
- TERMINATE
- PRINT

User Defined Functions

- Called inline in KSQL
- Types are: UDF, UDAF, UDTF
- Implemented in Java & deployed as JARs to ksql-server

Producer

Allows an application to publish a stream of records to one or more Kafka topics.

Consumer

Allows an application to subscribe to one or more topics and process the stream of records produced to them.

Streams

Allows an application to act as a stream processor, consuming an input stream from one or more topics and producing an output stream to one or more output topics, effectively transforming the input streams to output streams.

Connector

Allows building and running reusable producers or consumers that connect Kafka topics to existing applications or data systems. For example, a connector to a relational database might capture every change to a table.


confluent