REAL TIME STREAM PROCESSING WITH

KSQL AND KAFKA

Changing Architectures

Kafka?

Stream Processing

KSQL

KSQL in Production

76% of Kafka code created by Confluent team

confluent

Changing Architectures

confluent

Events

A Sale

An Invoice

A Trade

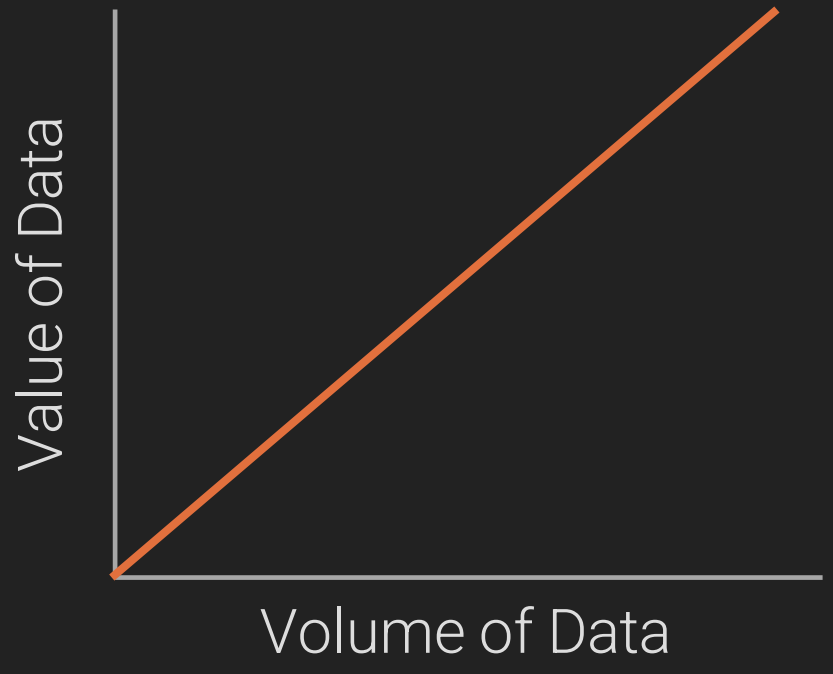
A Customer Experience


DATA

WE ARE CHALLENGING OLD ASSUMPTIONS...

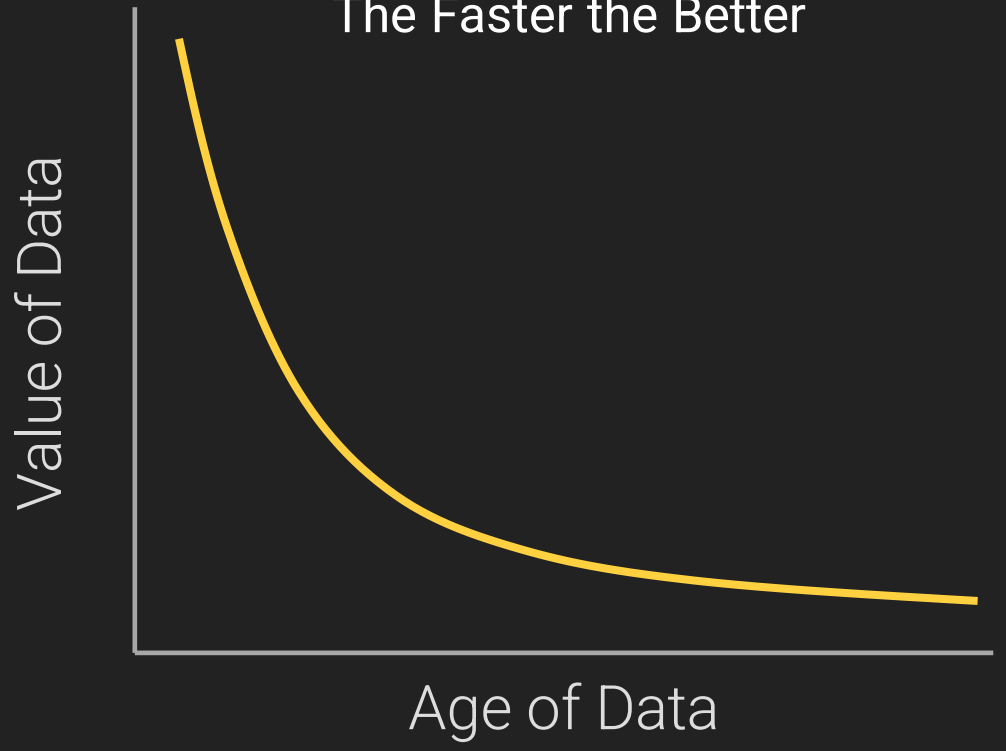
Big Data was

The More the Better



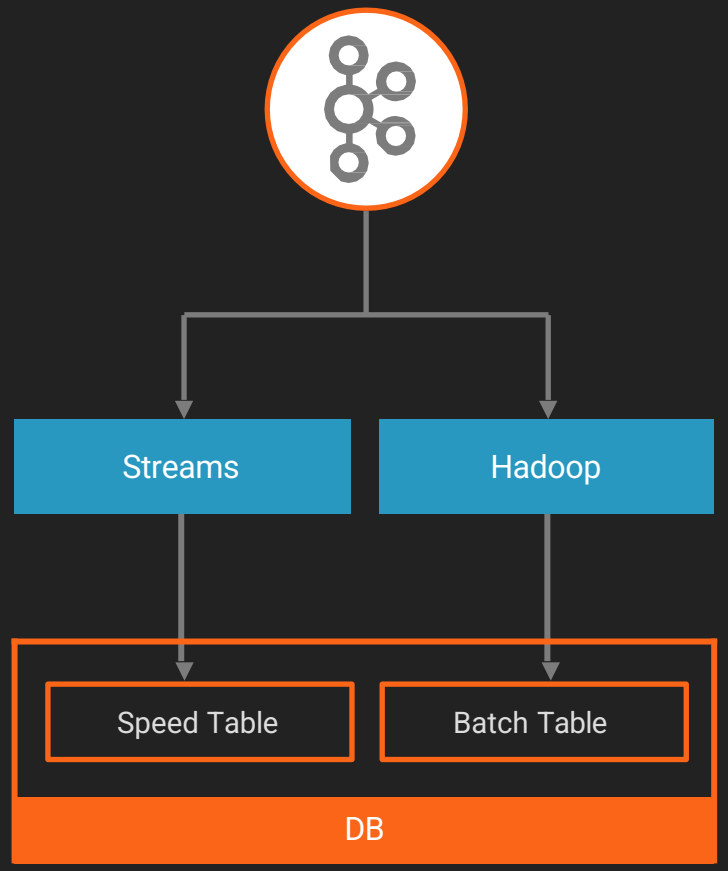
Stream Data is

The Faster the Better

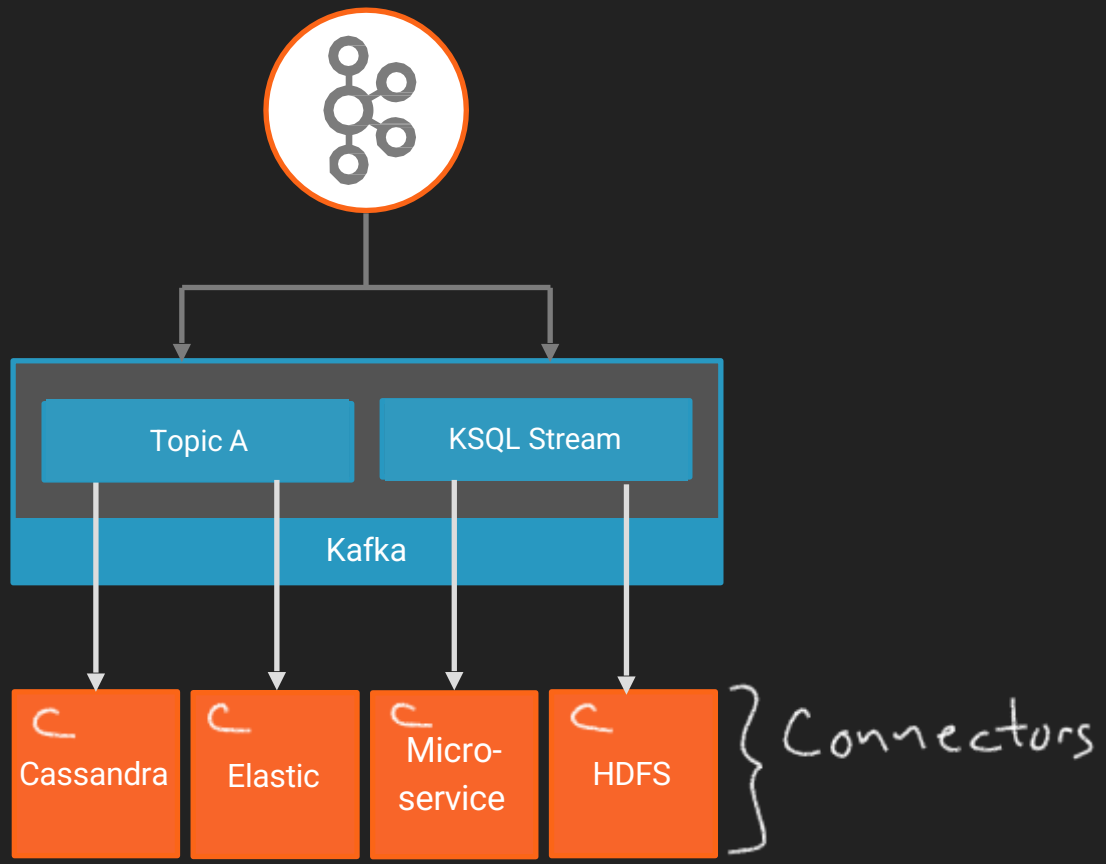


WE ARE CHALLENGING OLD ARCHITECTURES…

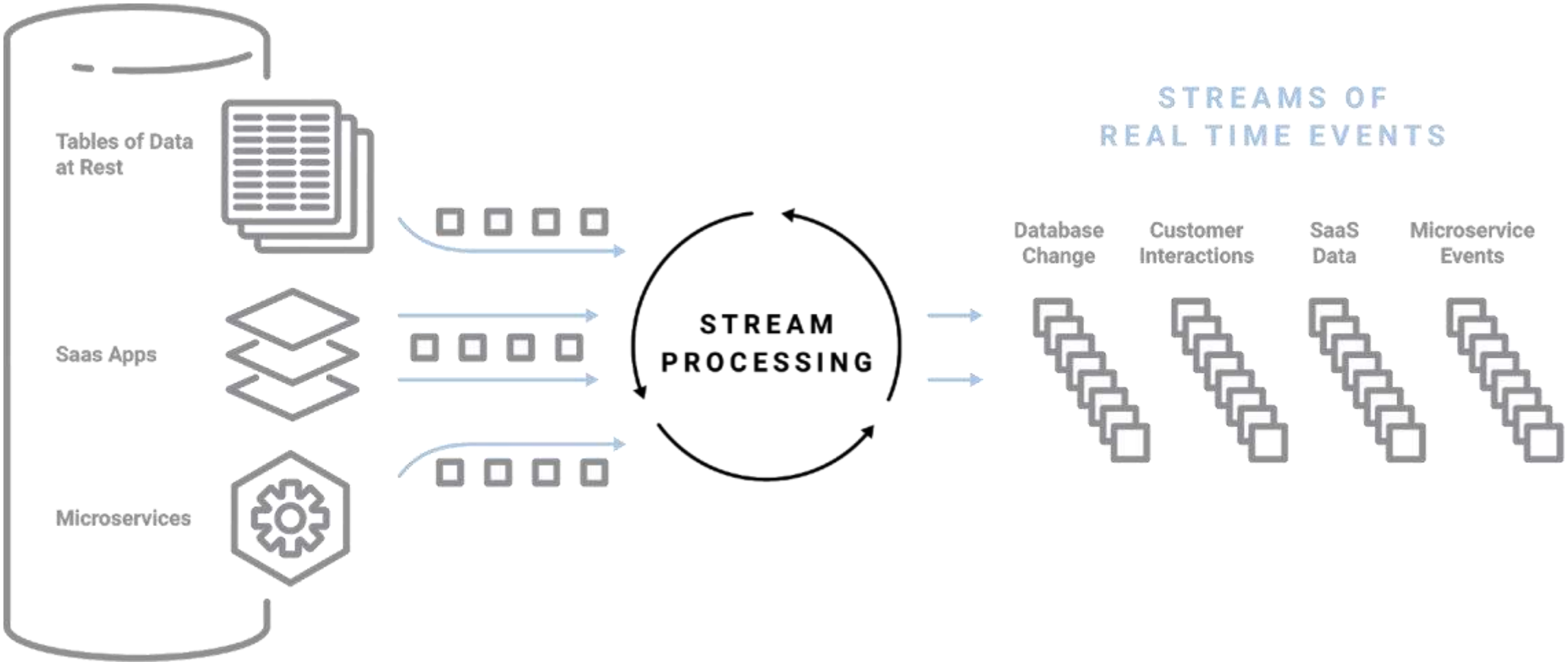
Lambda
Big **OR** Fast



Kappa
Big **AND** Fast



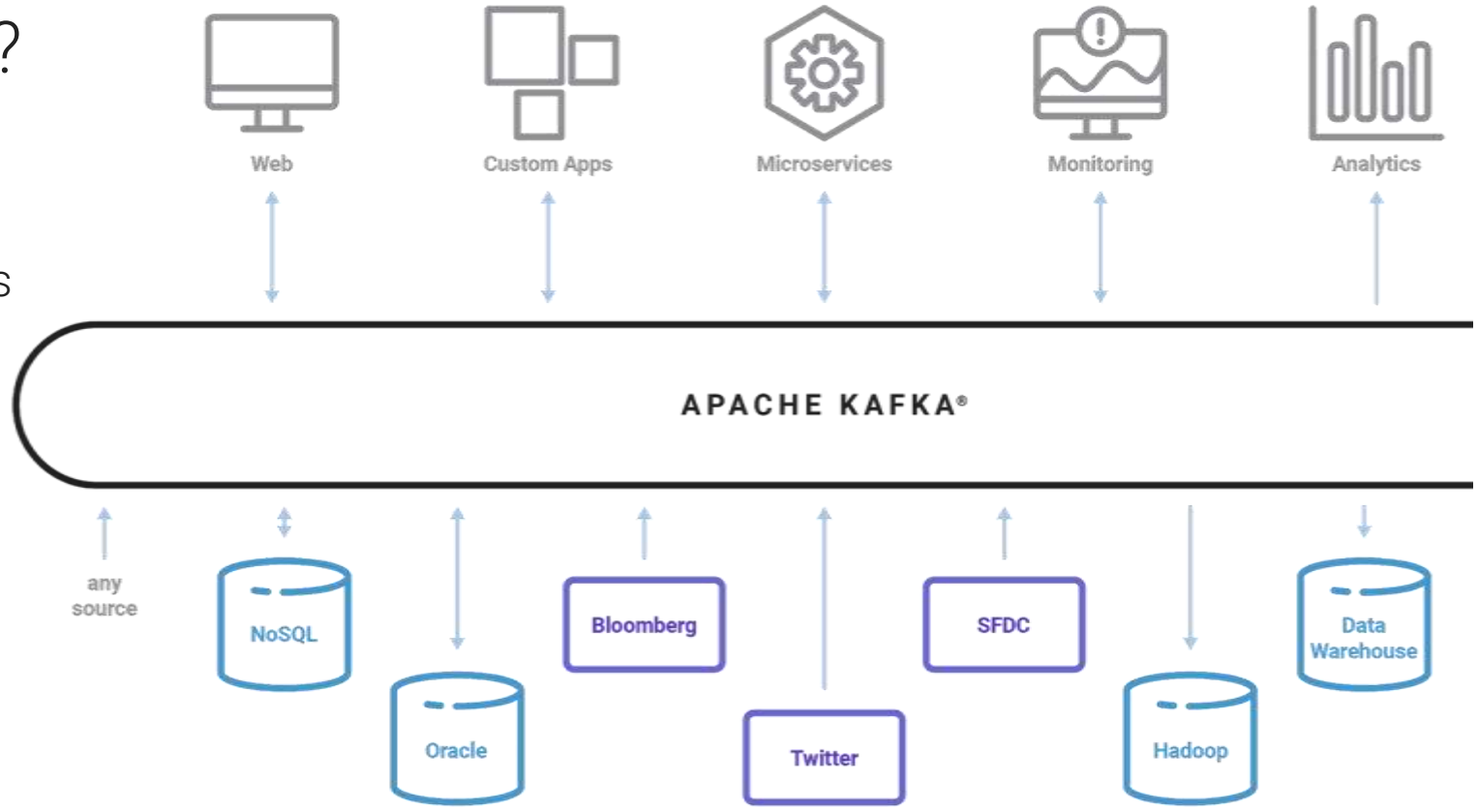
KAFKA: EVENT CENTRIC THINKING



AN EVENT-DRIVEN ENTERPRISE

What are the possibilities?

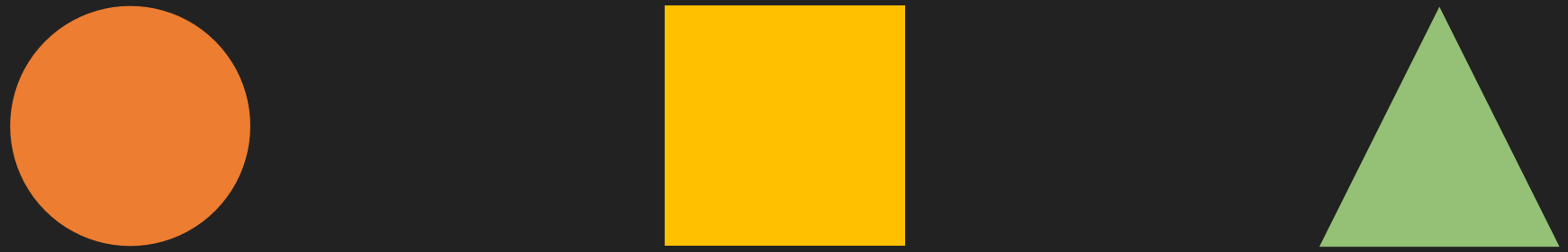
- Everything is an event
- Available instantly to all applications in a company
- Ability to query data as it arrives vs when it is too late
- Simplifying the data architecture by deploying a single platform



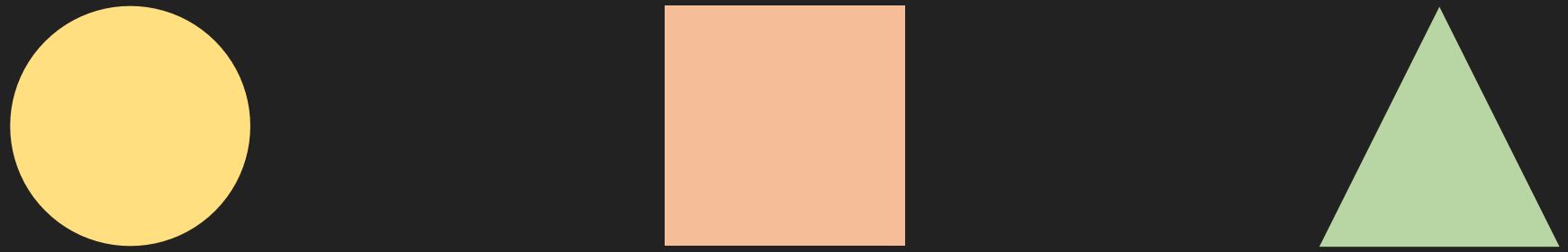

apache kafka

It’s a massively scalable distributed, fault tolerant, **publish & subscribe** key/value **datastore** with infinite data retention computing unbounded, **streaming data** in real time.

It’s a massively scalable distributed, fault tolerant, **publish & subscribe** key/value **datastore** with infinite data retention computing unbounded, **streaming data** in real time.



So, what is Kafka really?

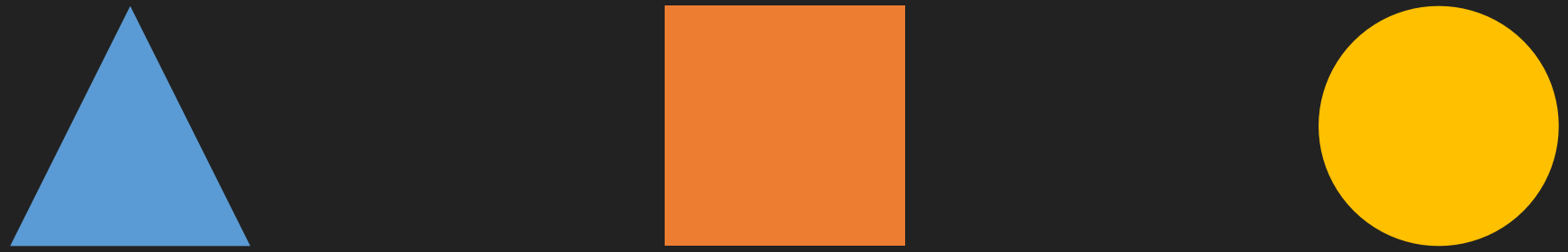


It’s made up of 3 key primitives

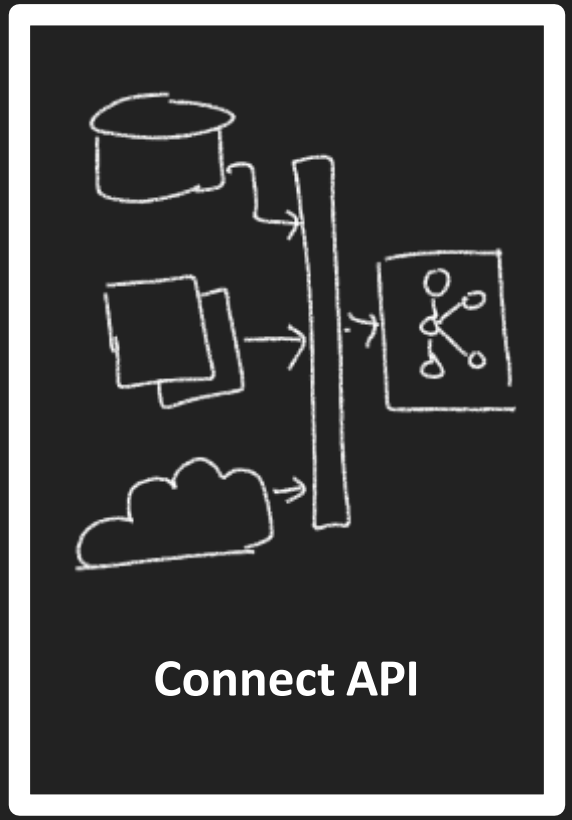
Publish &
Subscribe

Store

Process

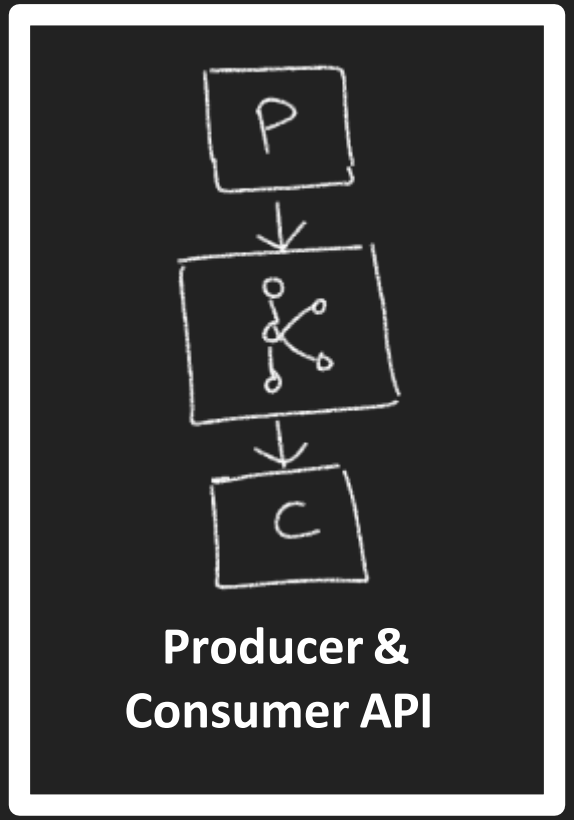


So, what is Kafka really?



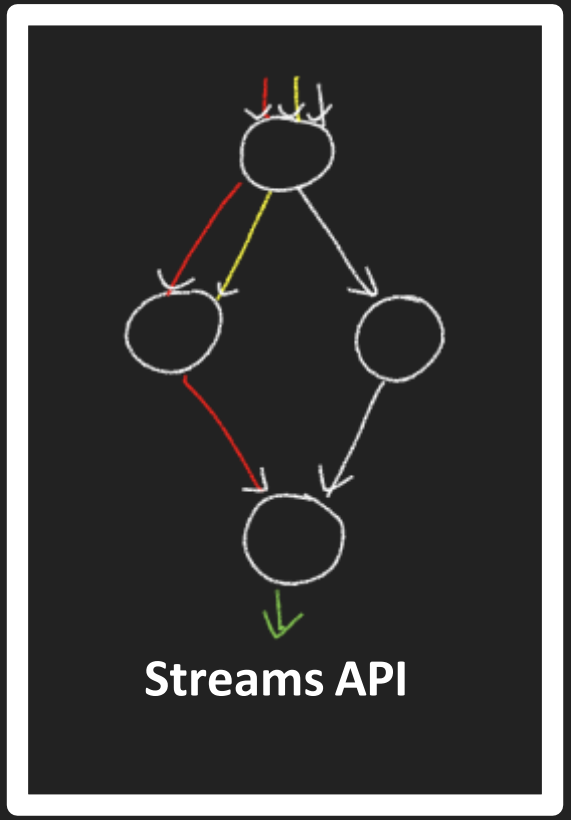
Connect API

Reliable and scalable integration of Kafka with other systems – no coding required.



Producer & Consumer API

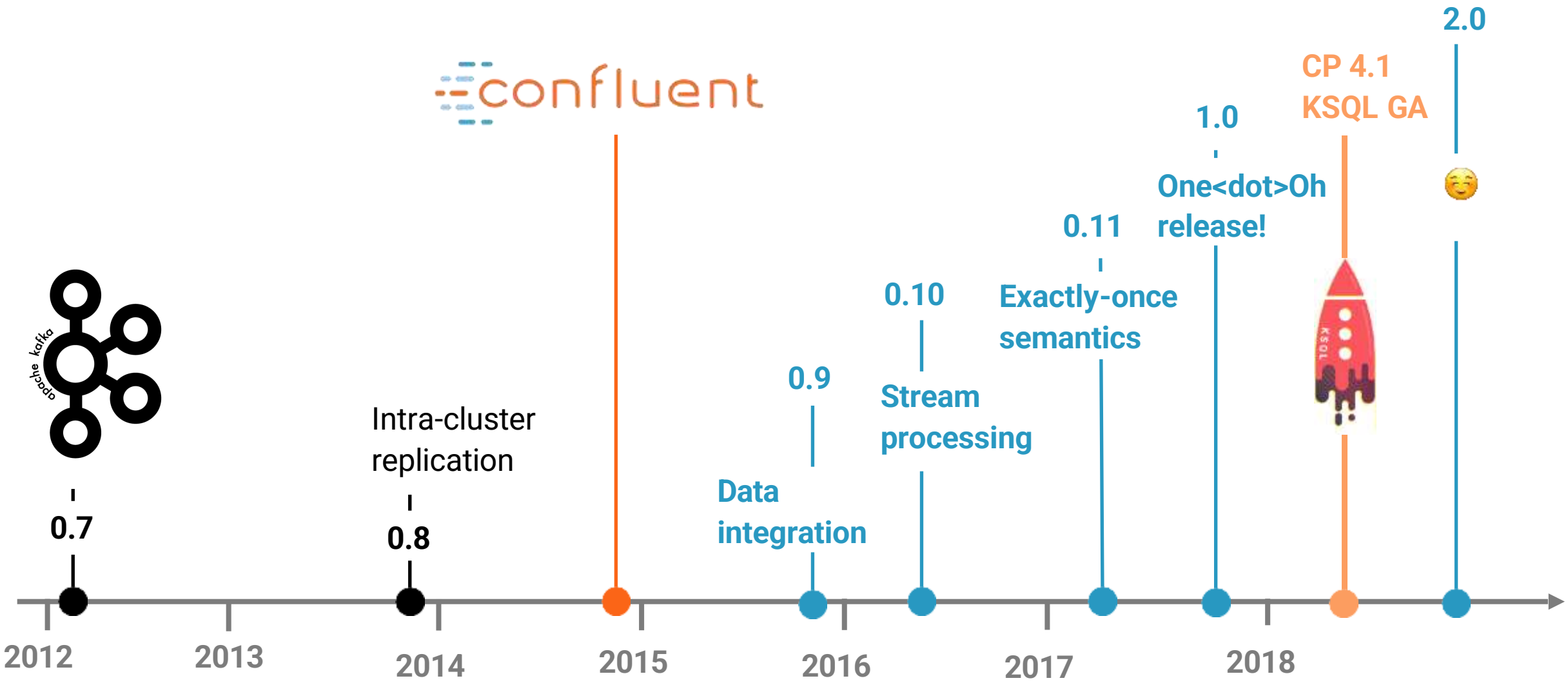
Open-source client libraries for numerous languages. Direct integration with your systems.

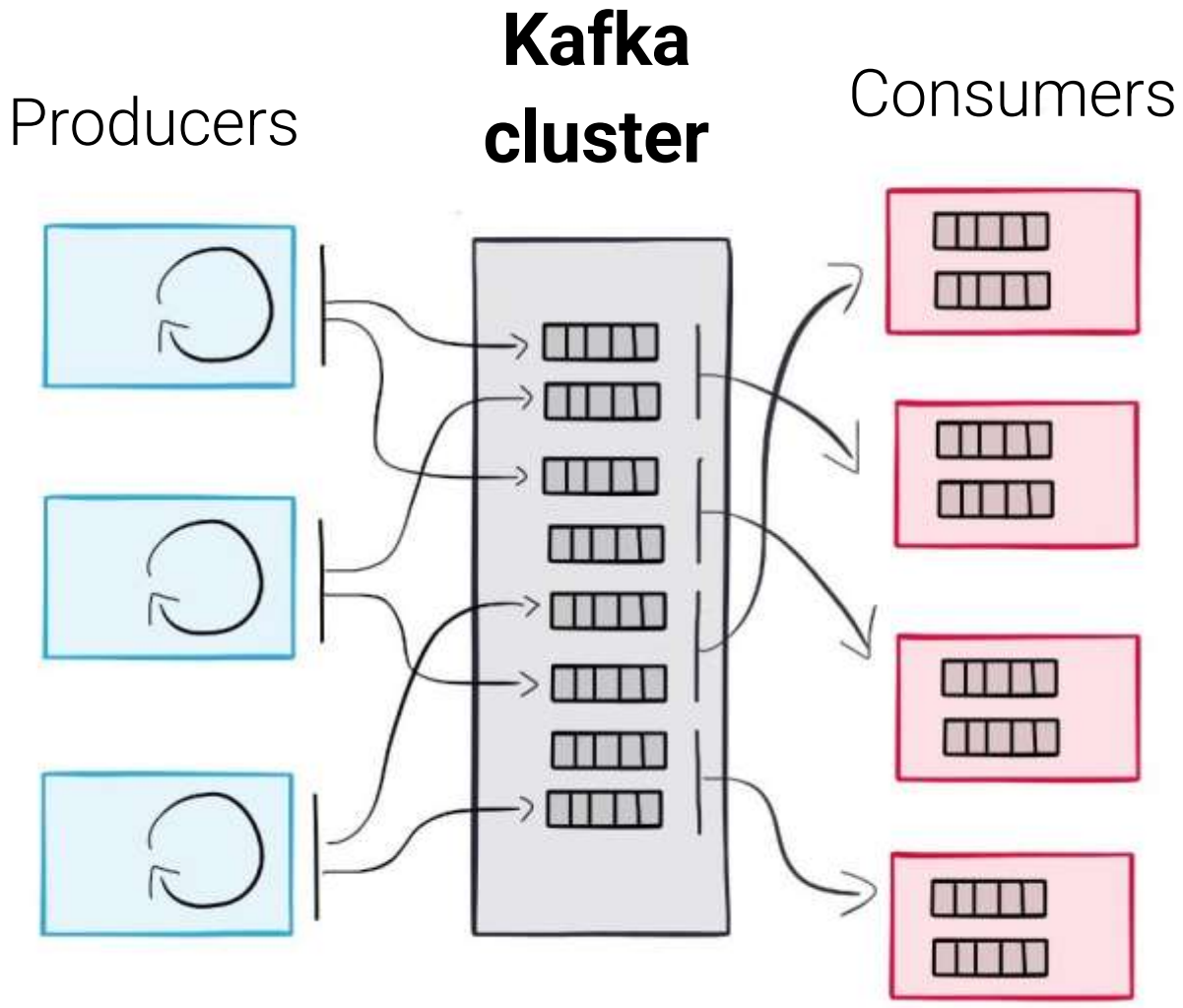


Streams API

Low-level and DSL, create applications & microservices to process your data in real-time

A Brief History of Apache Kafka and Confluent

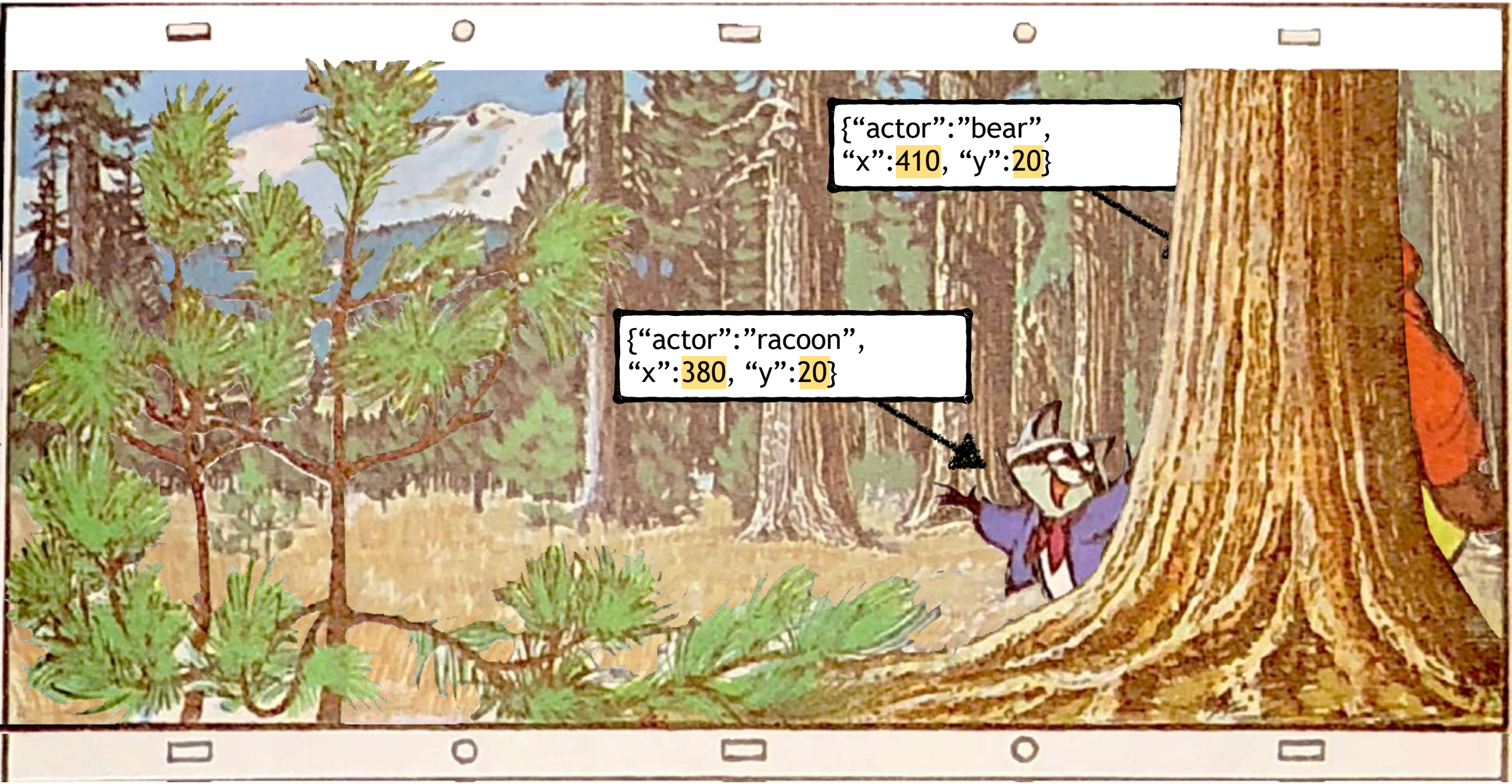



Producers
Kafka cluster
Consumers

confluent

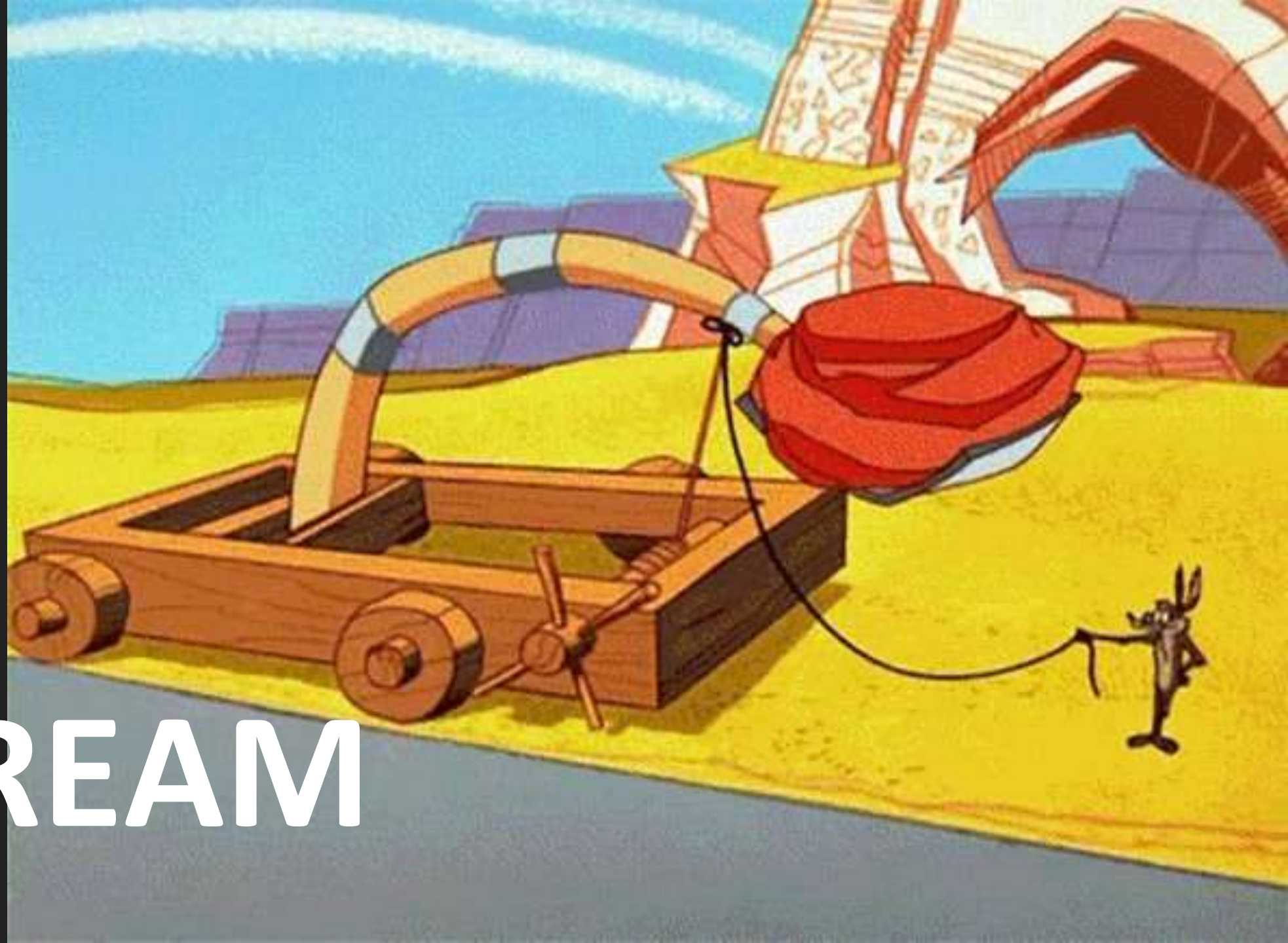
So, what exactly is a stream?

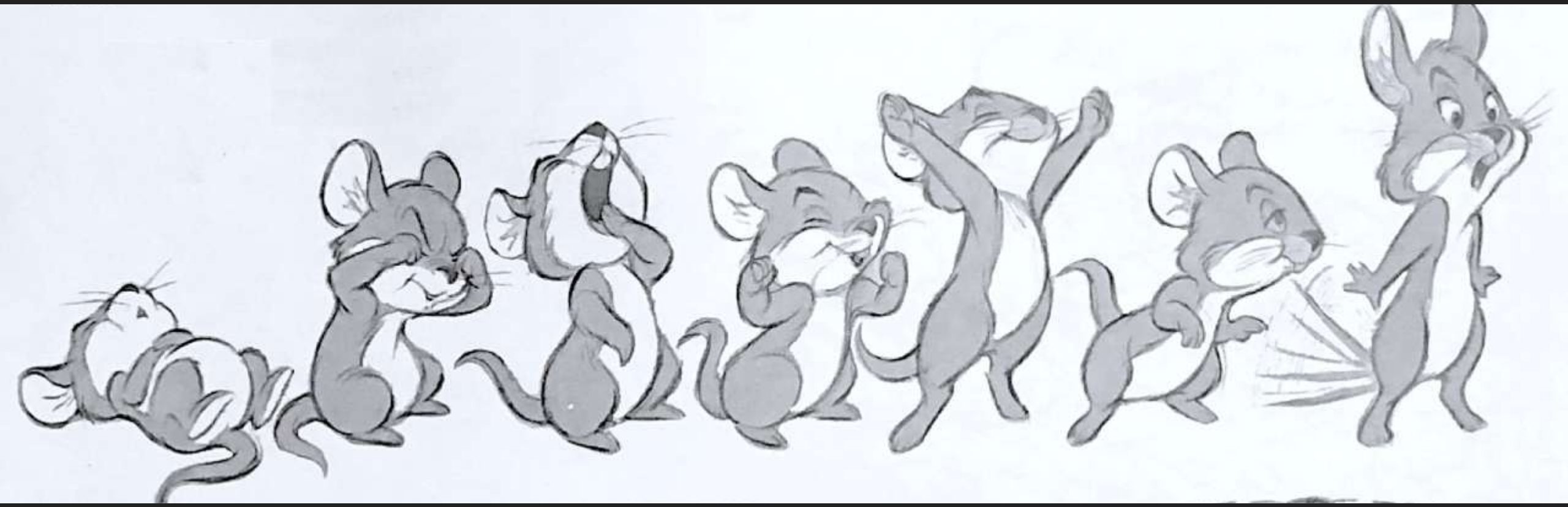
1. TOPIC


{“actor”:”bear”,
“x”:410, “y”:20}
{“actor”:”racoon”,
“x”:380, “y”:20}


{“actor”:”bear”,
“x”:380, “y”:22}
{“actor”:”racoon”,
“x”:350, “y”:22}

2.STREAM





3.TABLE

T KITTEN

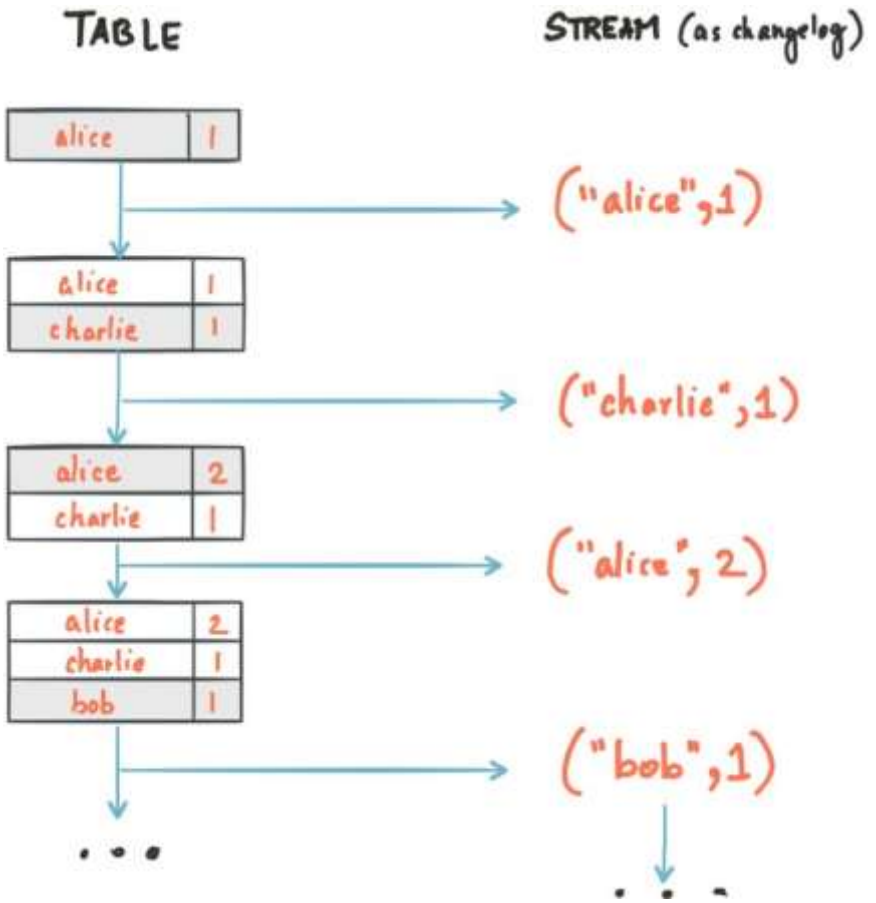
	AUDIO
OPPER	SONG - MUSIC
gs to off- d Duckling nd spirit.	GRASSHOPPER SINGS: (bouncing) All I do.... ...is have FUN..
SSHOPPER TEN	SONG - MUSIC
ten look ho becomes ous as he	GRASSHOPPER SINGS: (the blues) ...But I'm hun-gry.. ..some time..like now.
ing look ner.	
TO)	
KLING & EAD OF	SAD MUSIC

PRODUCTION "LOST KITTEN" #34 | ANIMATOR | SCENE 3 | SHEET 1

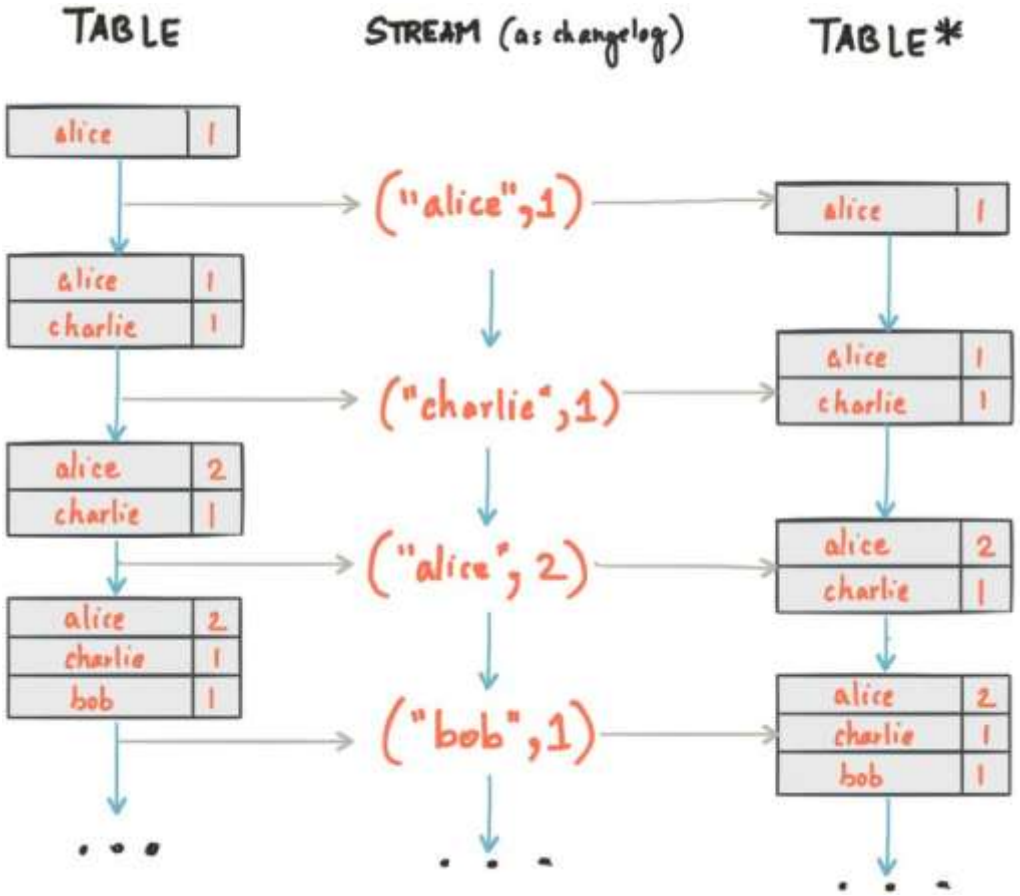
SCENE DESCRIPTION GRASSHOPPER SINGS SONG

ACTION	DIALOGUE	DIAL NO.	4	3	2	1	BG	CAMERA INSTRUCTION
	I	8 1			BLK	1		CAM. AT 7 FIELD CENTER
GRASSHOPPER RAISES HANDS	O	2						
	L	3				2		
		4						
		5				3		
	I	6						
		7				4		
		8				5		
		9				6		
INTO *		9 0				7		
	DOO	1				8		
	(EW)	2						
GESTURE		3				9		
		4						
		5				10		
6		6						
		7				11		
		8						
GESTURE		9			12A	12		
		10 0						
	IS	1			12B			
		2						
		3			12C			
		4						
	HAVE	5			12D			
		6						
		7			12E			
INTO		8						

Changelog stream – immutable events



Rebuild original table



Stream Processing

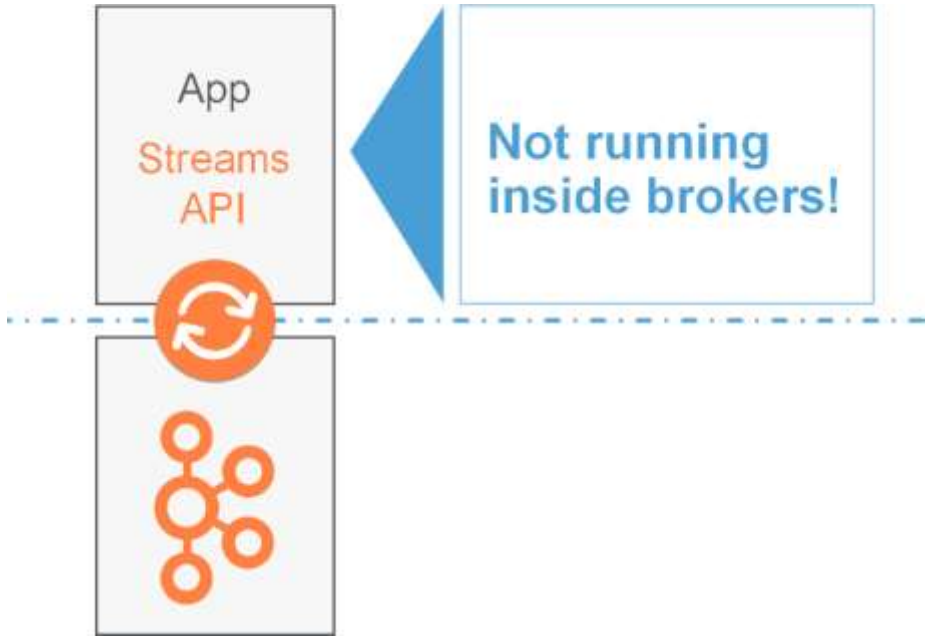
confluent

confluent

Standard App
No need to create a separate cluster
Highly scaleable, elastic, fault tolerant

Stream processing

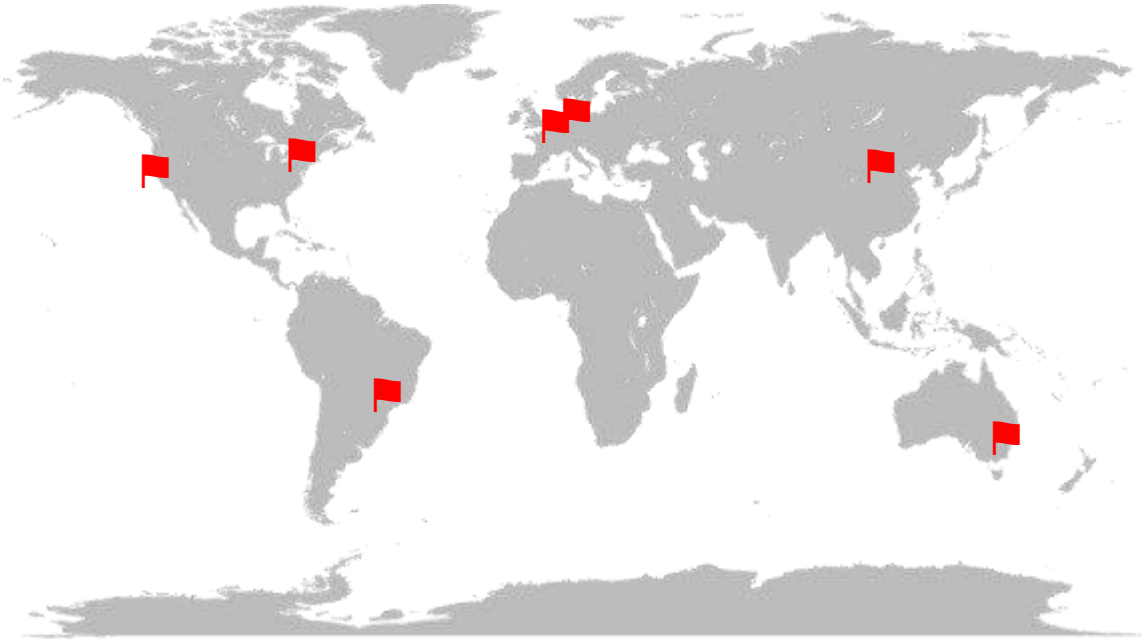
Lives inside your application

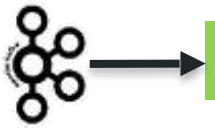


Same data, but different use cases

Use case 1: Frequent traveler status?

"Alice has been to SFO, NYC, Rio, Sydney, Beijing, Paris, and finally Berlin."

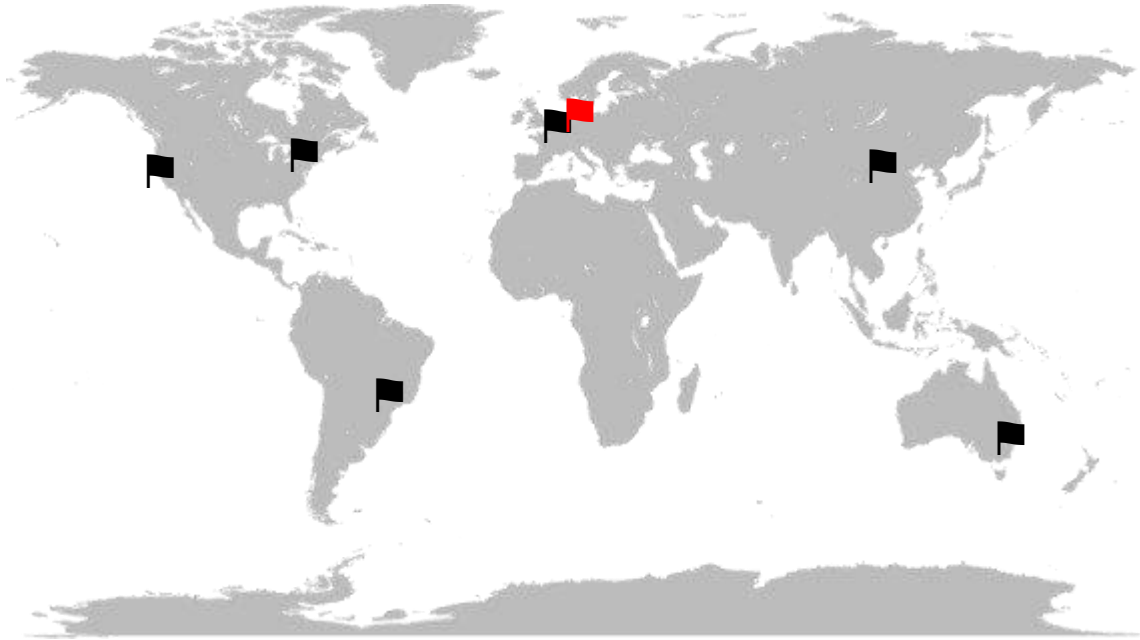


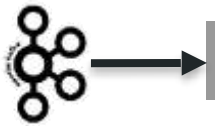


KStream

Use case 2: Current location?

"Alice is in ~~SFO~~, ~~NYC~~, ~~Rio~~, ~~Sydney~~, ~~Beijing~~, ~~Paris,~~ Berlin right now."





KTable

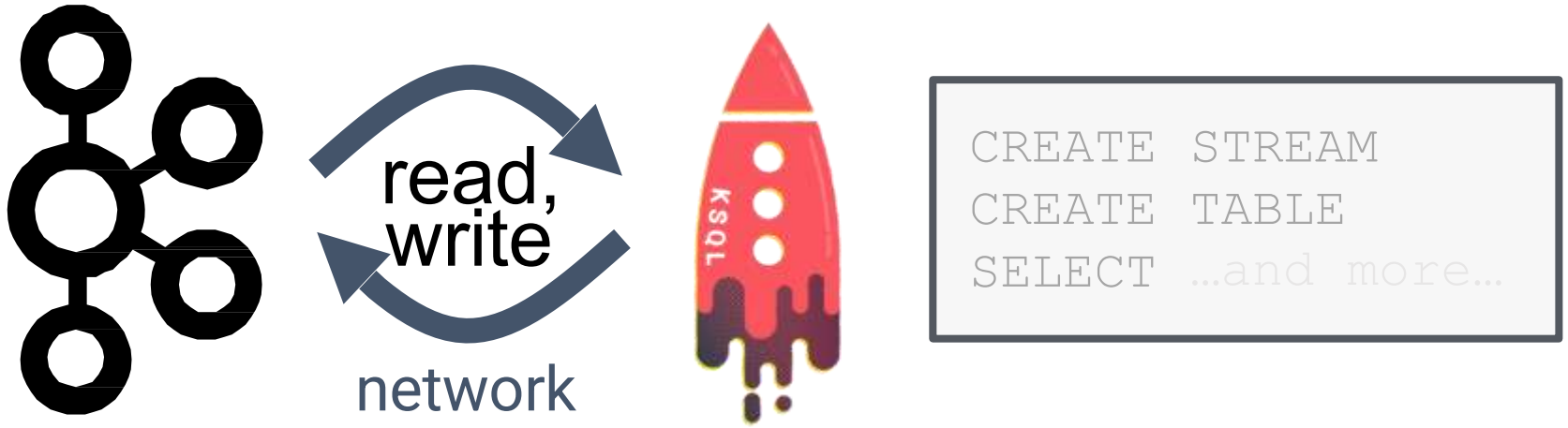
KSQL

confluent

KSQL — get started fast with Stream Processing

Kafka
(data)

KSQL
(processing)



All you need is Kafka – no complex deployments of bespoke systems for stream processing!

KSQL Concepts

- No need for source code deployment
  - Zero, none at all, not even one tiny file
- All the Kafka Streams capabilities out-of-the-box
  - Exactly Once Semantics
  - Windowing
  - Event-time aggregation
  - Late-arriving data
  - Distributed, fault-tolerant, scalable, ...

KSQL — SELECT statement syntax

```
SELECT `select_expr` [, ...]
FROM `from_item` [, ...]
[ WINDOW `window_expression` ]
[ WHERE `condition` ]
[ GROUP BY `grouping expression` ]
[ HAVING `having_expression` ]
[ LIMIT n ]
```

where *`from_item`* is one of the following:

```
stream_or_table_name [ [ AS ] alias]
from_item LEFT JOIN from_item ON join_condition
```

what are some KSQL use cases?

KSQL — Data exploration

An easy way to inspect data in Kafka

```
SHOW TOPICS;

PRINT 'my-topic' FROM BEGINNING;
```

```
SELECT page, user_id, status, bytes
FROM clickstream
WHERE user_agent LIKE 'Mozilla/5.0%';
```

KSQL — Data enrichment

Join data from a variety of sources to see the full picture

```
CREATE STREAM enriched_payments AS
  SELECT payment_id, u.country, total
  FROM payments_stream p
  LEFT JOIN users_table u
         ON p.user_id = u.user_id;
```

Stream-table join

KSQL — Streaming ETL

Filter, cleanse, process data while it is moving

```
CREATE STREAM clicks_from_vip_users AS
  SELECT user_id, u.country, page, action
  FROM clickstream c
  LEFT JOIN users u ON c.user_id = u.user_id
  WHERE u.level ='Platinum';
```

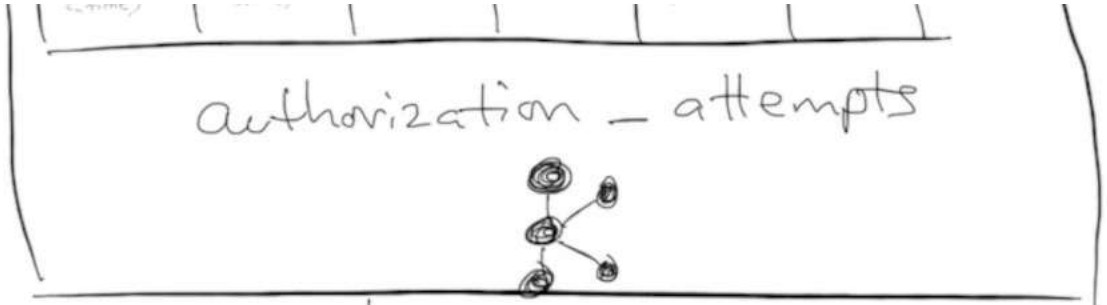
KSQL — Anomaly Detection

Aggregate data to identify patterns or anomalies in real-time

```
CREATE TABLE possible_fraud AS
  SELECT card_number, COUNT(*)
  FROM authorization_attempts
  WINDOW TUMBLING (SIZE 5 MINUTE)
  GROUP BY card_number
  HAVING COUNT(*) > 3;
```

Aggregate data

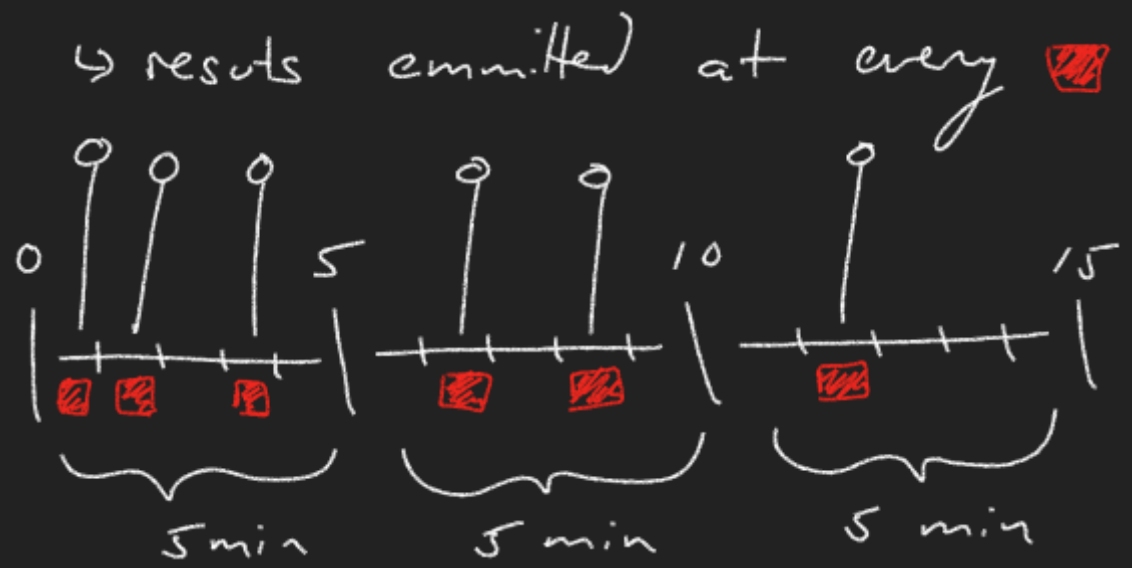
… per 5 min windows




```
CREATE STREAM possible_fraud AS
SELECT card_number, count(*)
FROM authorization_attempts
WINDOW TUMBLING (SIZE 5 MINUTE)
GROUP BY card_number
HAVING count(*) > 3;
```

STREAMING

TIME



TUMBLING

HOPPING

SESSION

KSQL — Real time monitoring

Derive insights from events (IoT, sensors, etc.) and turn them into actions

```
CREATE TABLE failing_vehicles AS
  SELECT vehicle, COUNT(*)
  FROM vehicle_monitoring_stream
  WINDOW TUMBLING (SIZE 1 MINUTE)
  WHERE event_type = 'ERROR'
  GROUP BY vehicle
  HAVING COUNT(*) >= 3;
```

KSQL — Data transformation

Quickly make derivations of existing data in Kafka

```
CREATE STREAM clicks_by_user_id
  WITH (PARTITIONS=6,
        TIMESTAMP='view_time'
        VALUE_FORMAT='JSON') AS
  SELECT * FROM clickstream
  PARTITION BY user_id;
```

Convert data to JSON

Re-key the data

KSQL — Stream to Stream JOINs

Example: Detect late orders by matching every SHIPMENTS row with ORDERS rows that are within a 2-hour window.

```
CREATE STREAM late_orders AS
  SELECT o.orderid, o.itemid FROM orders o
  FULL OUTER JOIN shipments s WITHIN 2 HOURS
  ON s.orderid = o.orderid WHERE s.orderid IS NULL;
```

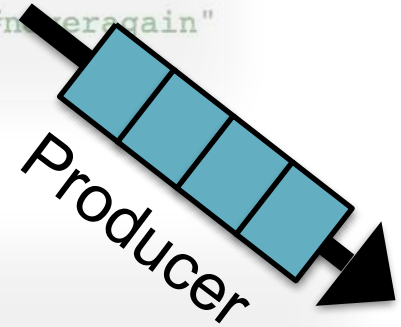
INSERT INTO statement for Streams

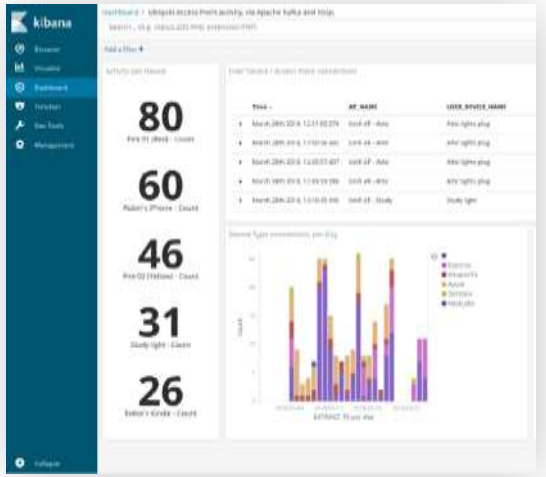
Example: Compute daily sales per item across online and offline stores

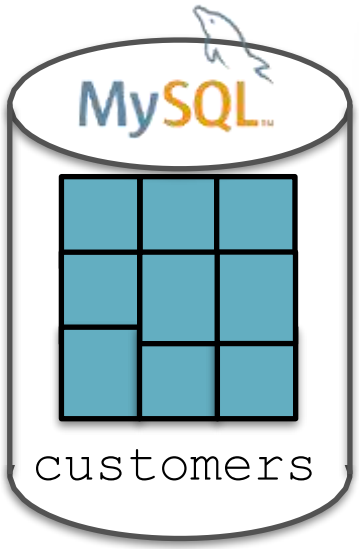
```
CREATE STREAM sales_online  (itemId BIGINT, price INTEGER, shipmentId BIGINT) WITH (...);
CREATE STREAM sales_offline (itemId BIGINT, price INTEGER, storeId    BIGINT) WITH (...);
CREATE STREAM all_sales  (itemId BIGINT, price INTEGER) WITH (...);

-- Merge the streams into `all_sales`
INSERT INTO all_sales SELECT itemId, price FROM sales_online;
INSERT INTO all_sales SELECT itemId, price FROM sales_offline;

CREATE TABLE daily_sales_per_item AS
  SELECT itemId, SUM(price) FROM all_sales
  WINDOW TUMBLING (SIZE 1 DAY) GROUP BY itemId;
```

KSQL — Demo

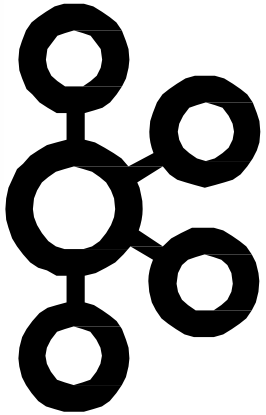
KSQL processes **table** changes in real-time

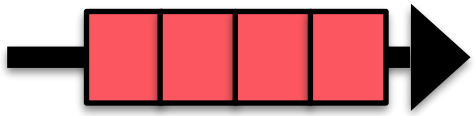






Kafka Connect streams data in



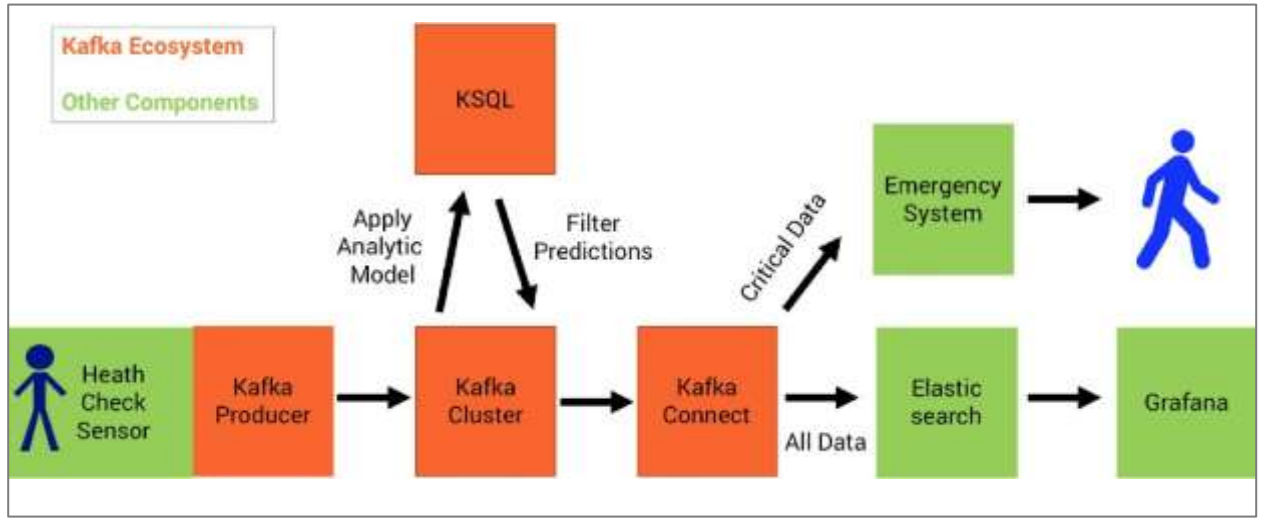


Kafka Connect streams data out

elastic

```
kafka_1             | [2018-09-03 04:18:51,227] INFO [Log partition=_confluent-ksql-confluent_rmoff_01query_CSAS_RATINGS_WITH_
CUSTOMER_DATA_2-KSTREAM-MAPVALUES-0000000011-repartition-0, dir=/var/lib/kafka/data] Incrementing log start offset to 675959 (
kafka.log.Log)
kafka_1             | [2018-09-03 04:18:51,229] INFO Cleared earliest 0 entries from epoch cache based on passed offset 675959
 leaving 1 in EpochFile for partition _confluent-ksql-confluent_rmoff_01query_CSAS_RATINGS_WITH_CUSTOMER_DATA_2-KSTREAM-MAPVAL
UES-0000000011-repartition-0 (kafka.server.epoch.LeaderEpochFileCache)
datagen-ratings_1   | 39681 --> ([ 39681 | 13 | 3 | 7387 | 1535948331371 | 'iOS' | 'thank you for the most friendly, helpful e
xperience today at your new lounge' ])
datagen-ratings_1   | 39682 --> ([ 39682 | 10 | 2 | 7771 | 1535948331632 | 'web' | 'worst. flight. ever. #neveragain' ])
datagen-ratings_1   | 39683 --> ([ 39683 | 0 | 3 | 9054 | 1535948331703 | 'web' | 'more peanuts please' ])
datagen-ratings_1   | 39684 --> ([ 39684 | 14 | 1 | 4301 | 1535948331748 | 'web' | 'worst. flight. ever. #neveragain' ])
datagen-ratings_1   | 39685 --> ([ 39685 | 4 | 2 | 490 | 1535948331786 | 'iOS-test' | 'worst. flight. ever. #neveragain' ])
datagen-ratings_1   | 39686 --> ([ 39686 | 16 | 2 | 9395 | 1535948331789 | 'iOS-test' | 'airport refurb looks great, will fly
outta here more!' ])
datagen-ratings_1   | 39687 --> ([ 39687 | 2 | 4 | 6316 | 1535948332026 | 'iOS-test' | 'meh' ])
datagen-ratings_1   | 39688 --> ([ 39688 | 10 | 2 | 6012 | 1535948332084 | 'ios' | '(expletive deleted)' ])
datagen-ratings_1   | 39689 --> ([ 39689 | 13 | 4 | 2813 | 1535948332251 | 'iOS-test' | 'meh' ])
datagen-ratings_1   | 39690 --> ([ 39690 | 8 | 1 | 7071 | 1535948332342 | 'ios' | 'your team here rocks!' ])
datagen-ratings_1   | 39691 --> ([ 39691 | 12 | 4 | 3368 | 1535948332466 | 'iOS' | 'meh' ])
datagen-ratings_1   | 39692 --> ([ 39692 | 10 | 3 | 6749 | 1535948332843 | 'iOS' | 'thank you for the most friendly, helpful e
xperience today at your new lounge' ])
kafka_1             | [2018-09-03 04:18:53,253] INFO [Log partition=_confluent-ksql-confluent_rmoff_01query_CSAS_RATINGS_WITH_
CUSTOMER_DATA_2-KSTREAM-MAPVALUES-0000000011-repartition-0, dir=/var/lib/kafka/data] Incrementing log start offset to 675972 (
kafka.log.Log)
kafka_1             | [2018-09-03 04:18:53,255] INFO Cleared earliest 0 entries from epoch cache based on passed offset 675972
 leaving 1 in EpochFile for partition _confluent-ksql-confluent_rmoff_01query_CSAS_RATINGS_WITH_CUSTOMER_DATA_2-KSTREAM-MAPVAL
UES-0000000011-repartition-0 (kafka.server.epoch.LeaderEpochFileCache)
datagen-ratings_1   | 39693 --> ([ 39693 | 18 | 2 | 2096 | 1535948333324 | 'iOS-test' | '(expletive deleted)' ])
datagen-ratings_1   | 39694 --> ([ 39694 | 9 | 2 | 8230 | 1535948333785 | 'iOS' | 'thank you for the most friendly, helpful ex
perience today at your new lounge' ])
```

KSQL — Deep Learning for IoT Sensor Analytics



KSQL UDF using an analytic model under the hood

→ Write once, use in any KSQL statement

```
SELECT event_id
  anomaly(SENSORINPUT)
  FROM health_sensor;
```

User Defined Function

KSQL — User Defined Function (UDF)


```java
@Override
public Object evaluate(Object... args) {
  if (args.length != 1) {
    throw new KsqlFunctionException("Anomaly udf should have one input argument.");
  }
  try {
    return applyAnalyticModel(args[0]);

  } catch (Exception e) {
    throw new KsqlFunctionException("Model Inference failed. Please check the logs.");
  }
}

private Object applyAnalyticModel(Object object) throws Exception {

  GenModel rawModel;
  rawModel = (hex.genmodel.GenModel) Class.forName(modelClassName).newInstance();
  EasyPredictModelWrapper model = new EasyPredictModelWrapper(rawModel);

  // Prepare input sensor data to be in correct data format for the autoencoder model (double[]):
      String inputWithHash = (String) object;
      String[] inputStringArray = inputWithHash.split("#");
      double[] doubleValues = Arrays.stream(inputStringArray)
          .mapToDouble(Double::parseDouble)
          .toArray();

  RowData row = new RowData();
  int j = 0;
  for (String colName : rawModel.getNames()) {
    row.put(colName, doubleValues[j]);
    j++;
  }

  AutoEncoderModelPrediction p = model.predictAutoEncoder(row);
```

Putting KSQL into Production

confluent

DEPLOYING KSQL

CLI

REST

CODE

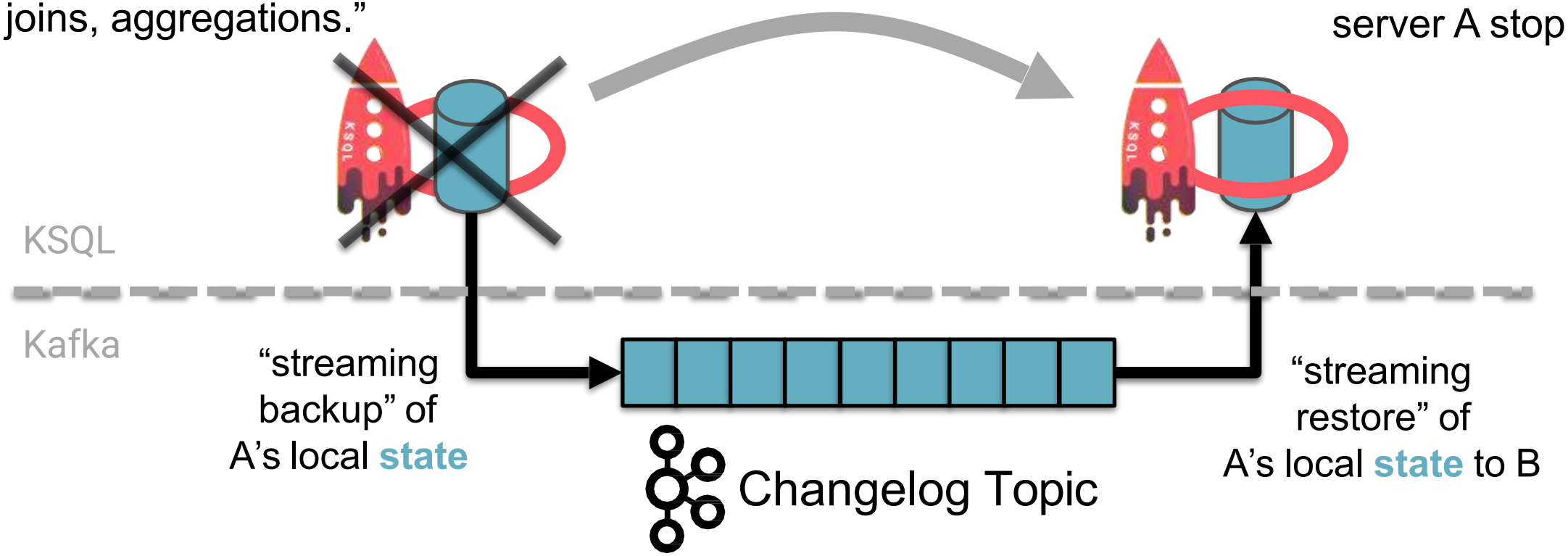
Fault-Tolerance, powered by Kafka

A key challenge of distributed stream processing is fault-tolerant state.

Server A:
"I do stateful stream processing, like tables, joins, aggregations."

State is automatically migrated in case of server failure

Server B:
"I restore the state and continue processing where server A stopped."

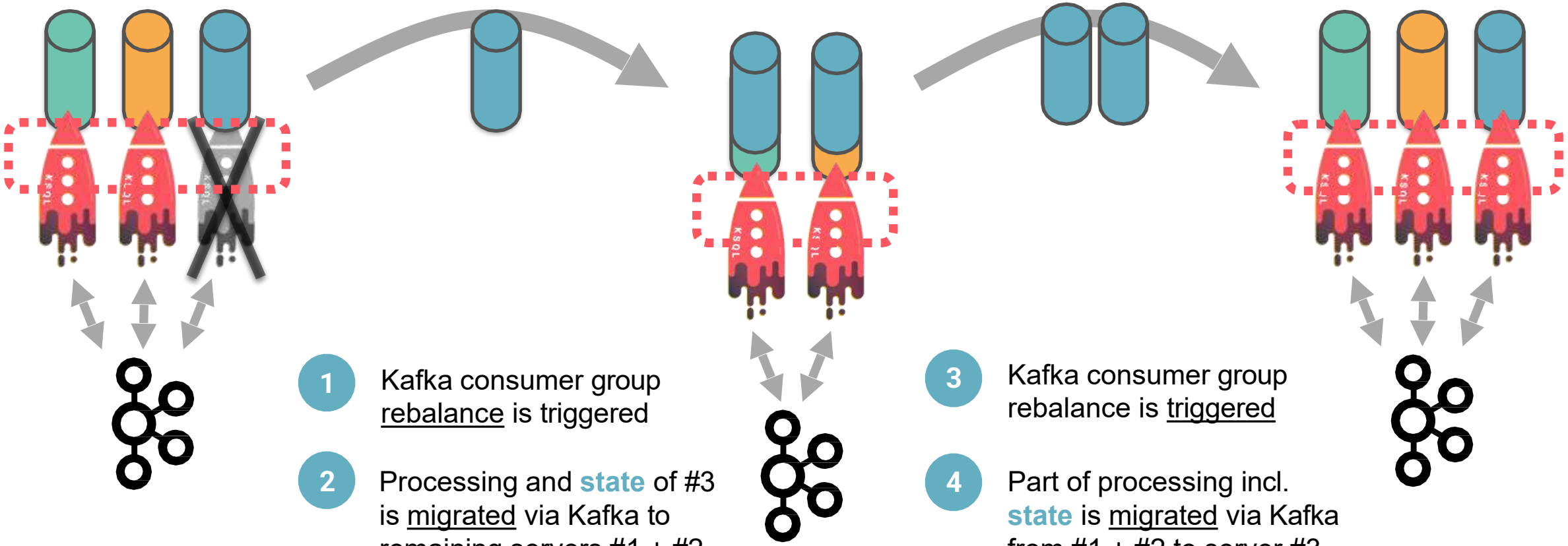


Fault-Tolerance, powered by Kafka

Processing fails over automatically, without data loss or miscomputation.

#3 died so #1 and #2 take over

#3 is back so the work is split again



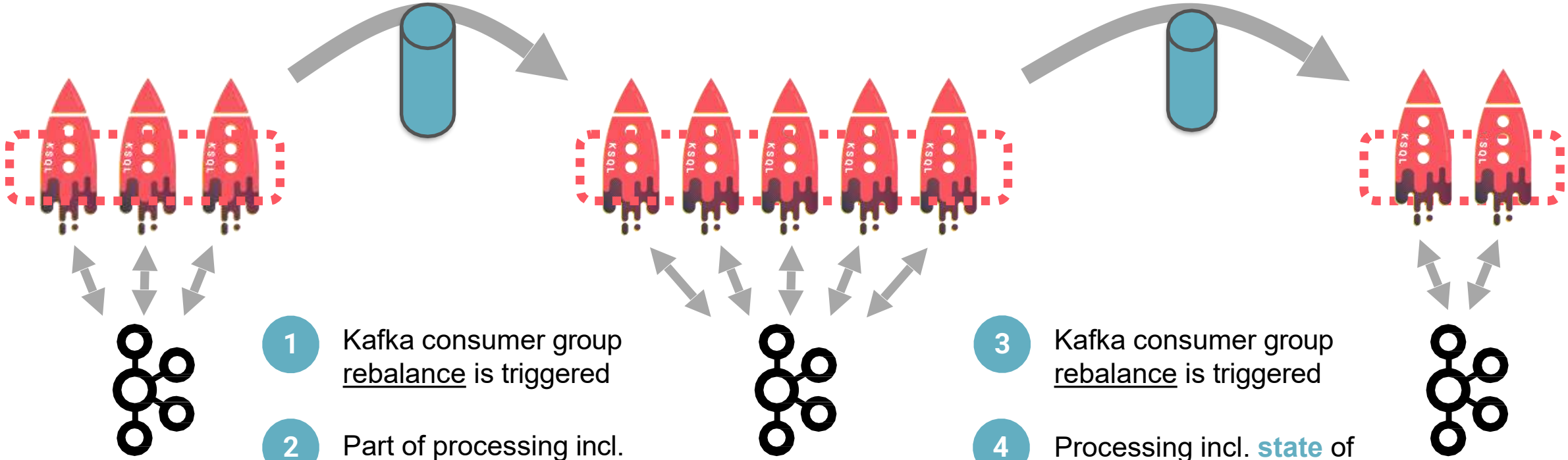
1. Kafka consumer group rebalance is triggered
2. Processing and **state** of #3 is migrated via Kafka to remaining servers #1 + #2
3. Kafka consumer group rebalance is triggered
4. Part of processing incl. **state** is migrated via Kafka from #1 + #2 to server #3

Elasticity and Scalability, powered by Kafka

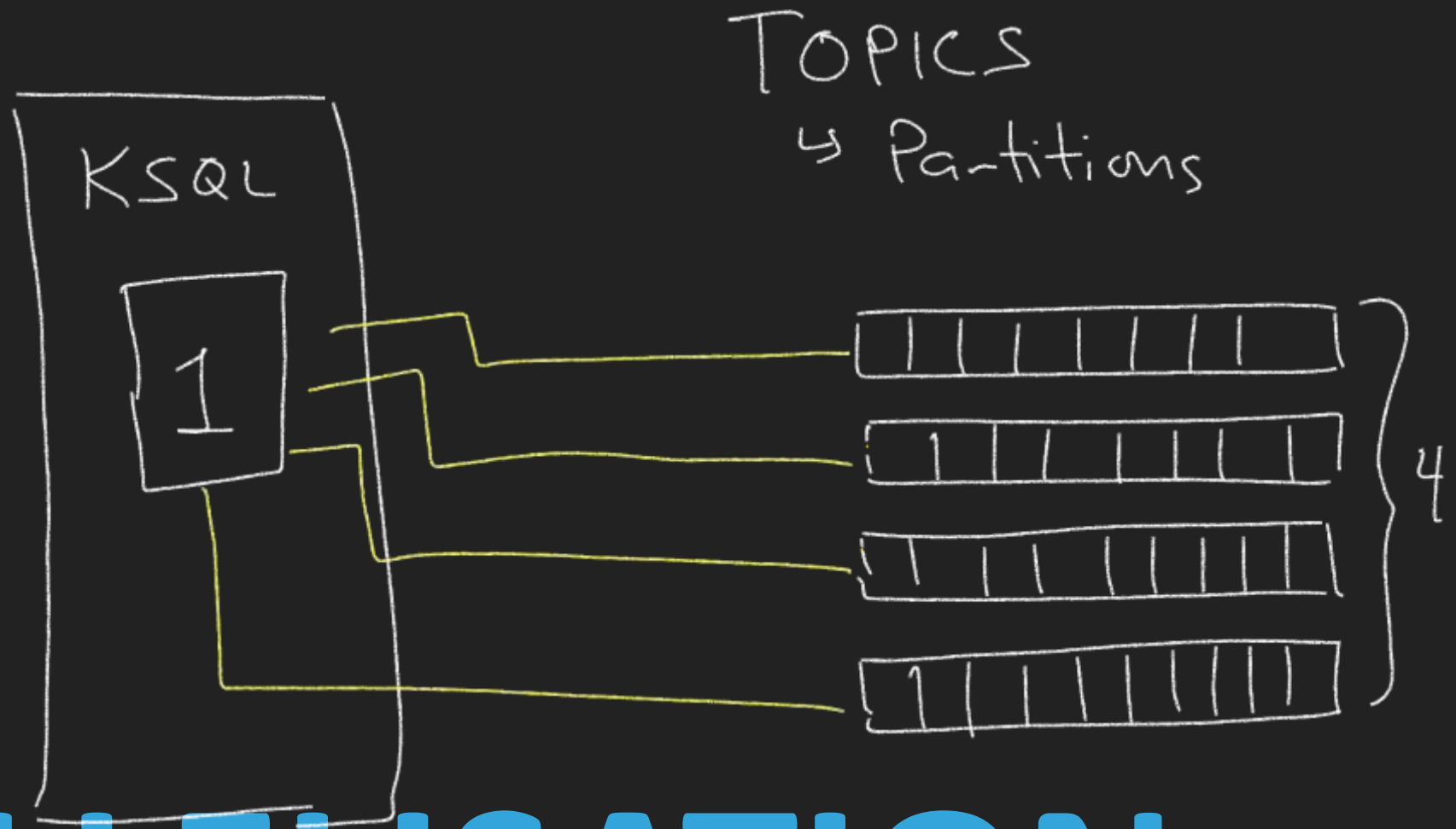
You can add, remove, restart servers in KSQL clusters **during live operations**.

"We need more processing power!"

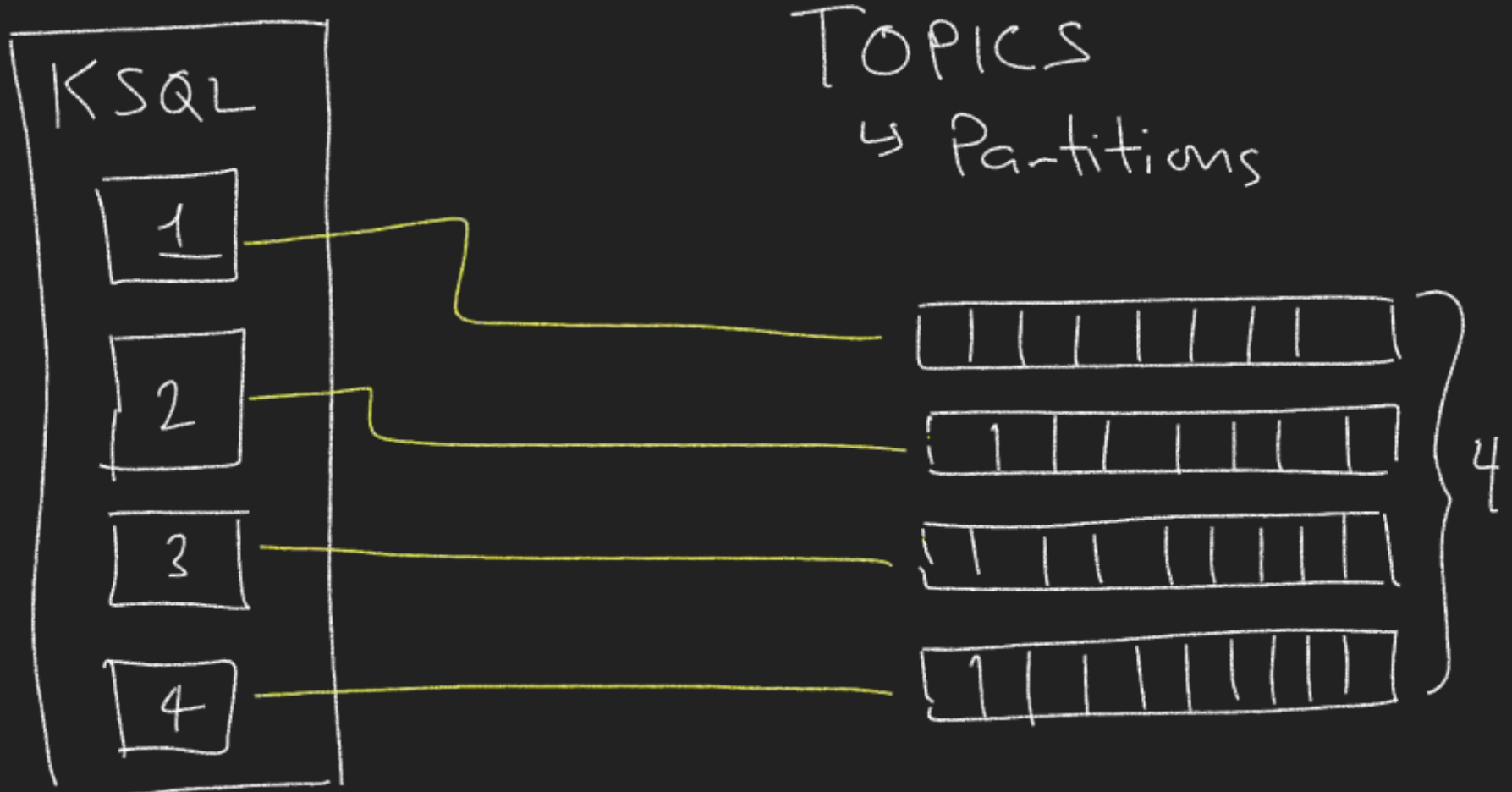
"Ok, we can scale down again."



1. Kafka consumer group rebalance is triggered
2. Part of processing incl. **state** is migrated via Kafka to additional server processes
3. Kafka consumer group rebalance is triggered
4. Processing incl. **state** of stopped servers is migrated via Kafka to remaining servers



PARALLELISATION



PARALLELISATION

KSQL

is the

Streaming SQL Engine

for

Apache Kafka

confluent

Resources and Next Steps

- Try the demo on GitHub :)
- Check out the code
- Play with the examples

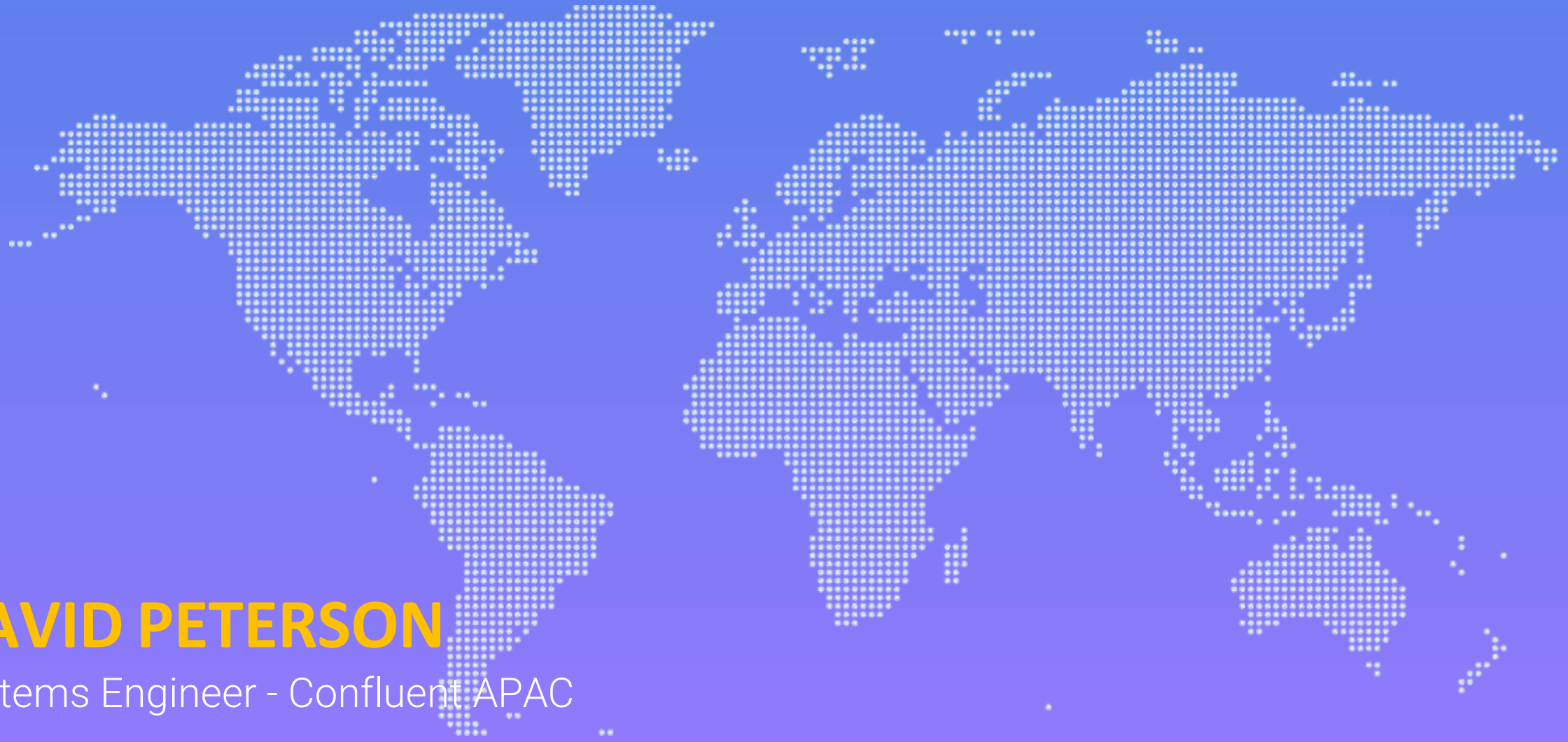
https://github.com/confluentinc/demo-scene

Download Confluent Open Source: https://www.confluent.io/download/

Chat with us: https://slackpass.io/confluentcommunity #ksql

confluent

The World's Best Streaming Platform — Everywhere



DAVID PETERSON

Systems Engineer - Confluent APAC

@davidseth