Time Series Analysis

Using an Event Streaming Platform

Virtual Meetup - September 8-th 2020

Dr. Mirko Kämpf - Solution Architect @ *Confluent, Inc. - Team CEMEA*

confluent

Abstract

Time Series Analysis using an Event Streaming Platform

Advanced time series analysis (TSA) requires very special data preparation procedures to convert raw data into useful and compatible formats.

In this presentation you will see some typical processing patterns for time series based research, from simple statistics to reconstruction of correlation networks.

The first case is relevant for anomaly detection and to protect safety.

Reconstruction of graphs from time series data is a very useful technique to better understand complex systems like supply chains, material flows in factories, information flows within organizations, and especially in medical research.

With this motivation we will look at typical data aggregation patterns. We investigate how to apply analysis algorithms in the cloud. Finally we discuss a simple reference architecture for TSA on top of the Confluent Platform or Confluent cloud.

This presentation is about linking:

- Time-Series-Analysis (TSA)
- Network- or Graph-Analysis
- Complex Event Processing (CEP).

Confluent Platform

Research work ends often with nice charts, scientific papers, and conference talks.

But, many published results **can't** be reproduced -
often because the setup it is simply too complicated …

Question:

How can we integrate data streams,
experiments, and decision making better?

Why not using batch processing?

Study anatomy ...

- **Batch processing is fine:**
  - as long as your data doesn’t change.
  - in PoCs for method development in a lab.
  - for research in a fixed scope.

Why using Kafka?

- **Stream processing is better:**
  - for real time business in changing environments.
  - iterative (research) projects.
  - repeatable experiments on replayed data.

Study and influence
the living system ...

Content:

(1) Intro

Typical types of event

How to identify hidden events?

(2) The Challenge

(3) Approach

Time Series Analytics &
Network Analytics in Kafka

Create time series from events

Create graphs from time series pairs

(4) Demo

(5) Architecture & Building Blocks:

(6) Unified Domain Model

confluent

Events - 1

Business events

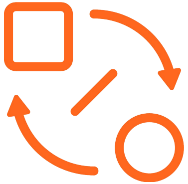
- transaction records
- discrete observation

How to handle events?

JUTS SIMPLE
OBSERVATION &
DATA CAPTURING

A Sale

An Invoice



A Trade

A Customer Experience

confluent

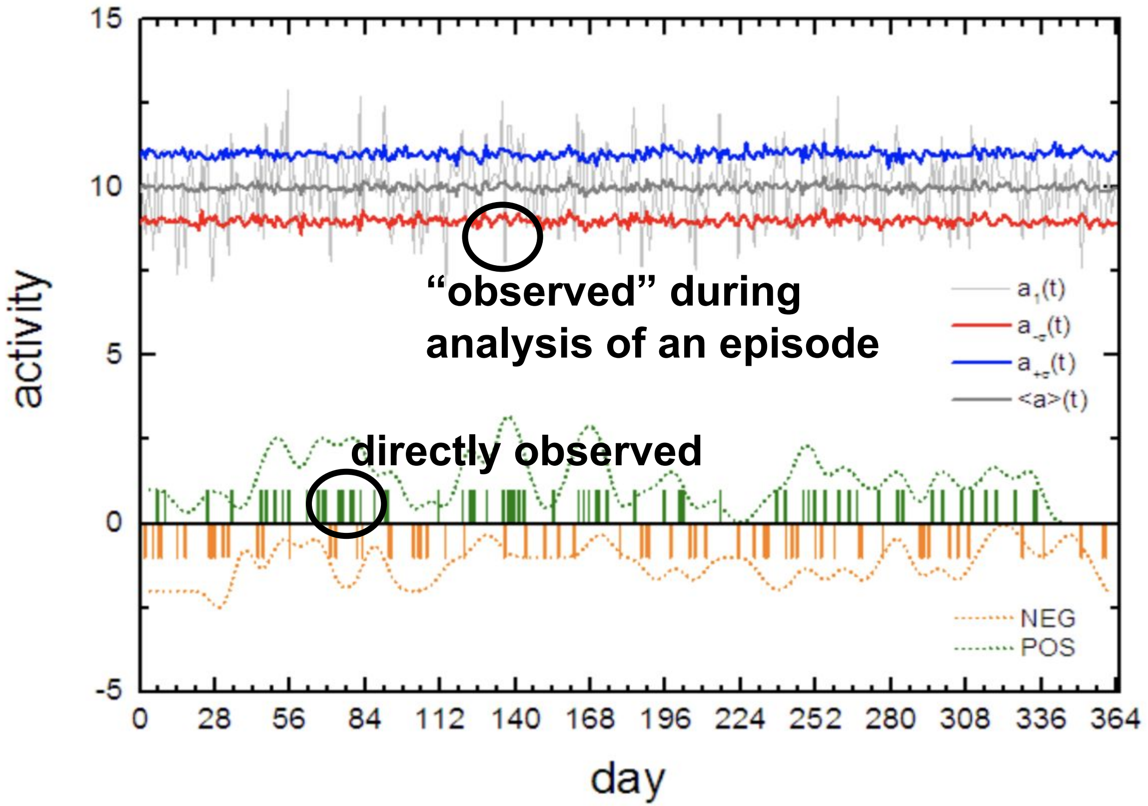
Events - 2

Well defined events

- in known context

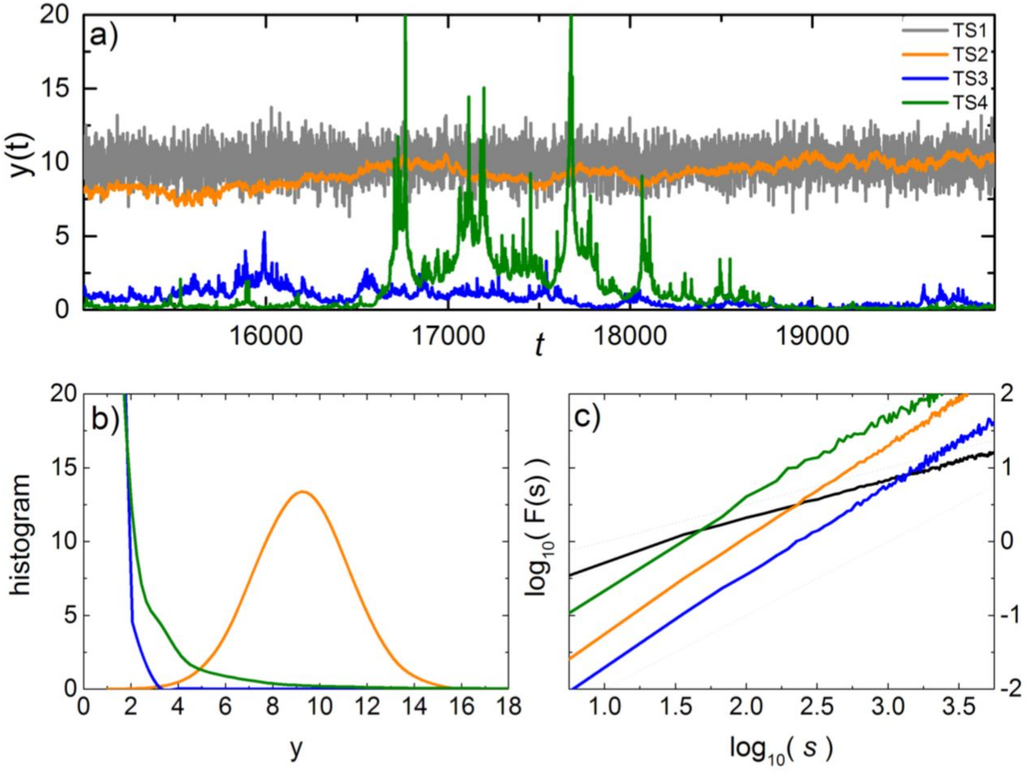
How to identify events?

Sometimes: SIMPLE
Sometimes: DATA ANALYSIS



Univariate TSA: single episodes are processed

- Extreme values
- Patterns
- Distribution of values
- Fluctuation properties
- Long-term correlations (memory effects)



confluent

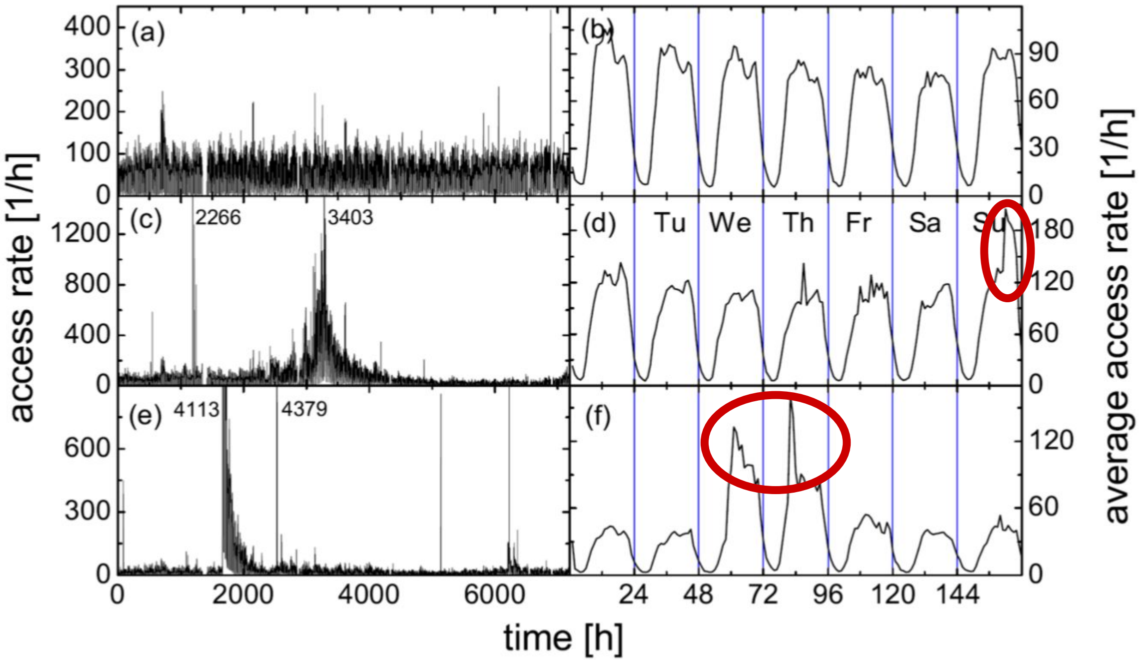
Events - 3

Extreme Events

- *"outliers"* in unknown context

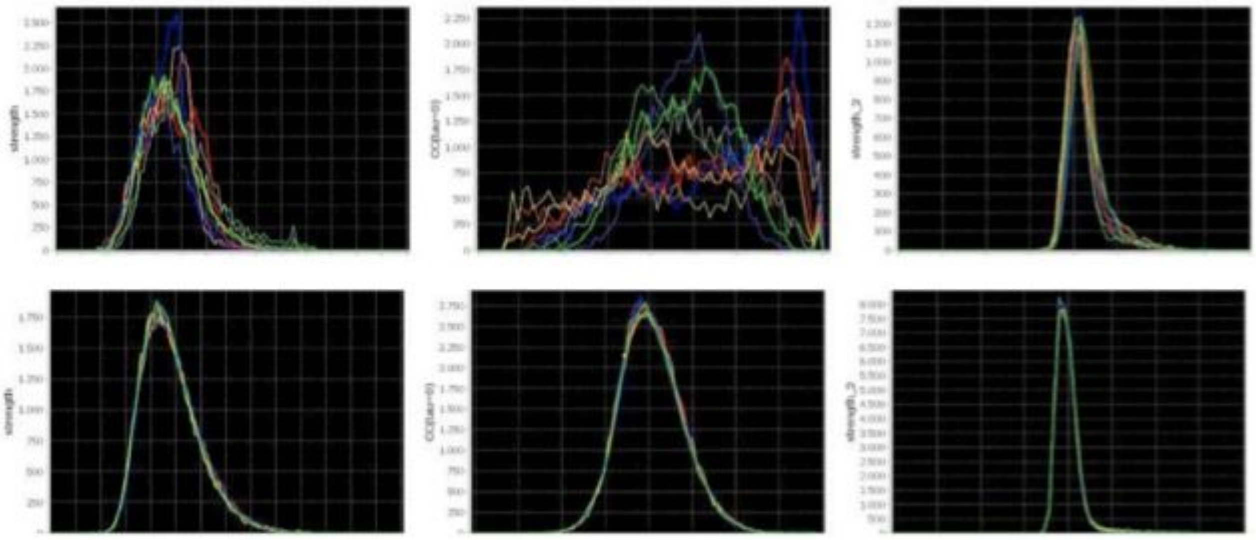
How to handle?

ADVANCED
DATA ANALYSIS (*& ML)*



Multivariate TSA: pairs / tuples of episodes are processed

- Comparison Similarity measures for link creation



Distribution of cross-correlation coefficients for pairs of access-rate time series of Wikipedia pages (top) compared to surrogat data (bottom) - 100 shuffled configurations are considered

Reality is Complex:

We should simplify a bit!

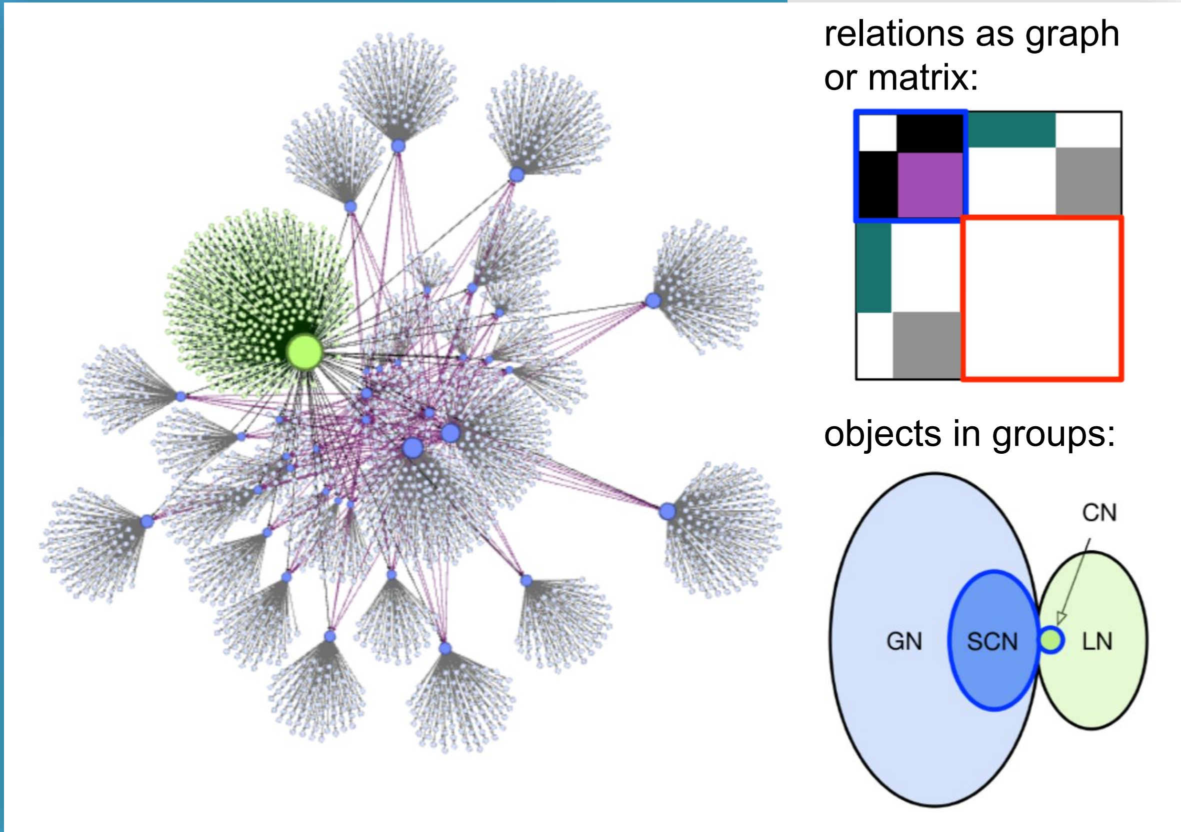
Simplification in our method can lead to isolation:

- DATA SILOS
- OPERATIONAL SILOS

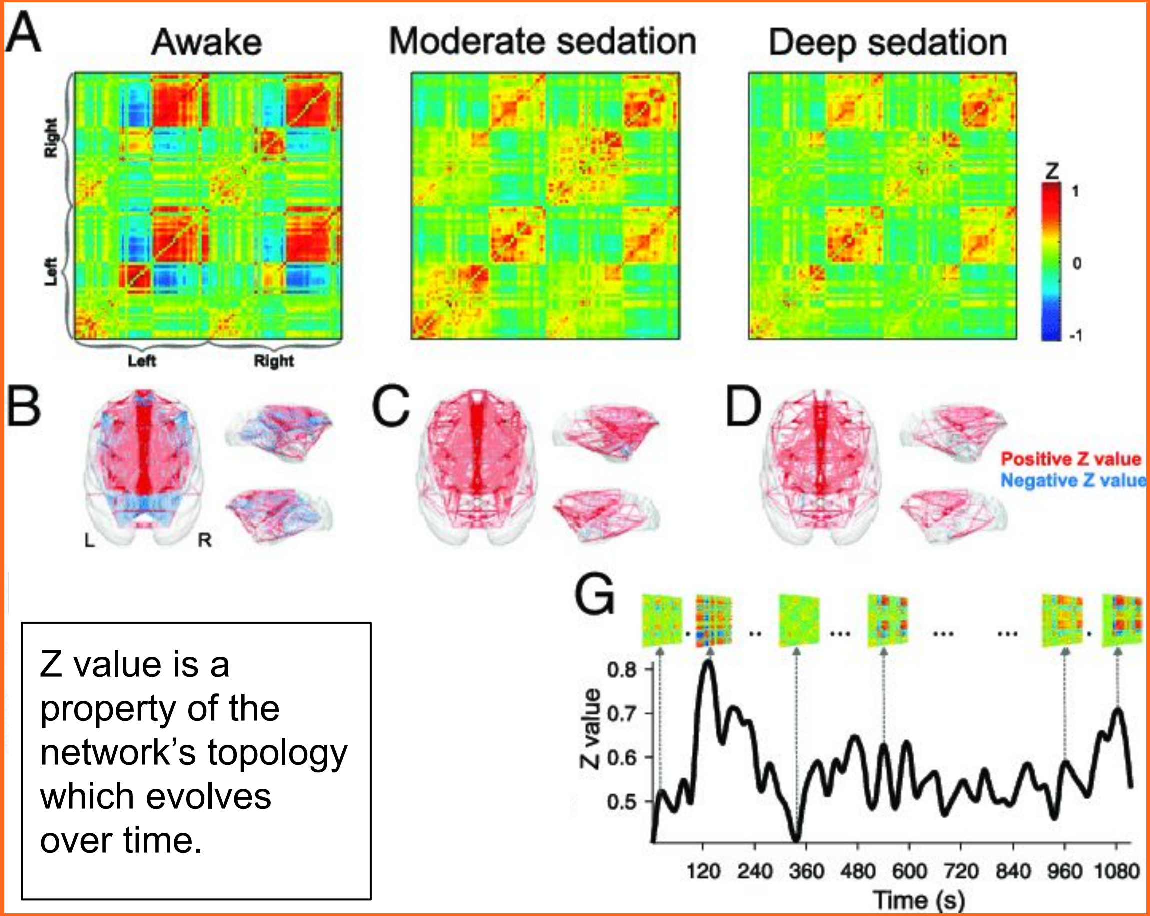
SOLUTION:

GRAPHS capture structure.

TIME SERIES capture properties over time (history).



confluent



Z value is a property of the network's topology which evolves over time.

WHAT?

WHY?

confluent

Recap:

What events are and how to process event-data can be misunderstood *or simply unclear.*

It all depends on our view and our goals!

IT Operations

- Server crash
- Cyber crime

Special procedures are established.

Business Events

- Big deal won
- Technical issue solved

The events which make people & the market happy :-)

Transactions (in business)

- orders placed
- products shipped
- bills paid

Event Driven Architecture

Extreme Events:

- Service slow down due to emerging bottlenecks
- Increased demand in a resource

Complex Event Analysis

The Challenge:

How can you combine **unbound data assets** and **scientific methods**?

- You bring data to a place where it can be processed easily,
  for example to a cloud system or into special purpose systems, such a event processing platform.

- You integrate important algorithms as early as possible in your processing pipeline to get results fast, using streaming processing capabilities.

Things can become complicated:

Complex Event Analysis

Integration Across Domains

Extraction of Hidden Event

METHODOLOGY

Complex Event Analysis

- time series analysis and ML reveal hidden events
- multi-stage processing is usually needed

ORGANIZATION & OPERATIONS

Integration Across Domains

- distributed event processing systems are used
- apps consume and produce events of different flavors
- event-types and data structures my change over time

Problems on ORGANIZATION level:

Many legacy systems can’t be integrated without additional expensive servers.
Often, this data is unreachable for externals.

Business data is managed by different teams using different technologies.

Data scientists use data in the cloud, and they all do really L❤️VE notebooks.
But often, they don’t use any automation.

TECHNOLOGY & SCIENCE

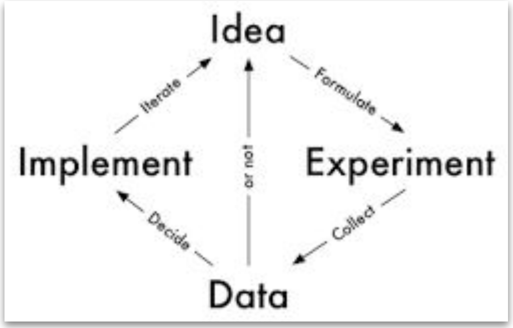
Extraction of Hidden Events

- requires **Applied** Data **Analysis** & Data **Science**
- embedding of **Complex Algorithms** in IT landscape
- integration of **GPU/HPC** and streaming data pipelines

Kafka and its Ecosystem …

- are considered to be middleware, managed by IT people:
  - researchers do not plan nor build their experiments around this technology yet.
- don't offer ML / AI components:
  - many people think, that a model has to be executed on an edge-device or in the cloud.

Yes, Apache Kafka can support agile (data)experiments!



- **Kafka APIs** give access to data (flows) in real time.
  - allows replay of experiments at a later point in time

- Kafka allows **variation of analysis without redoing a simulation or ingestion** by simply reusing persisted event-streams again.

- **Kafka Streams** and **KSQL** allow data processing in place:
  - this allows faster iterations, for example because plausibility checks can be done in place
  - the streaming API gives freedom for extension of core logic
  - DSL and KSQL will sav you a lot of implementation time

confluent

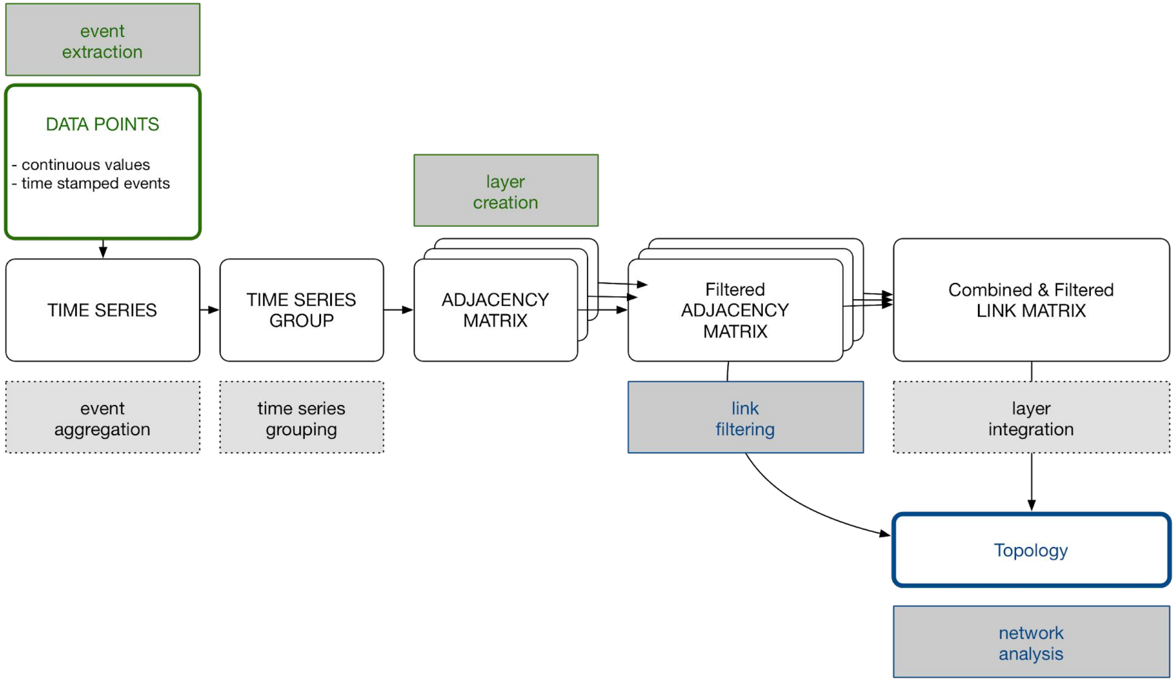
Why not building something great using the right tools ???

ADVANCED
TIME SERIES ANALYSIS &
NETWORK ANALYSIS

… how does it work?

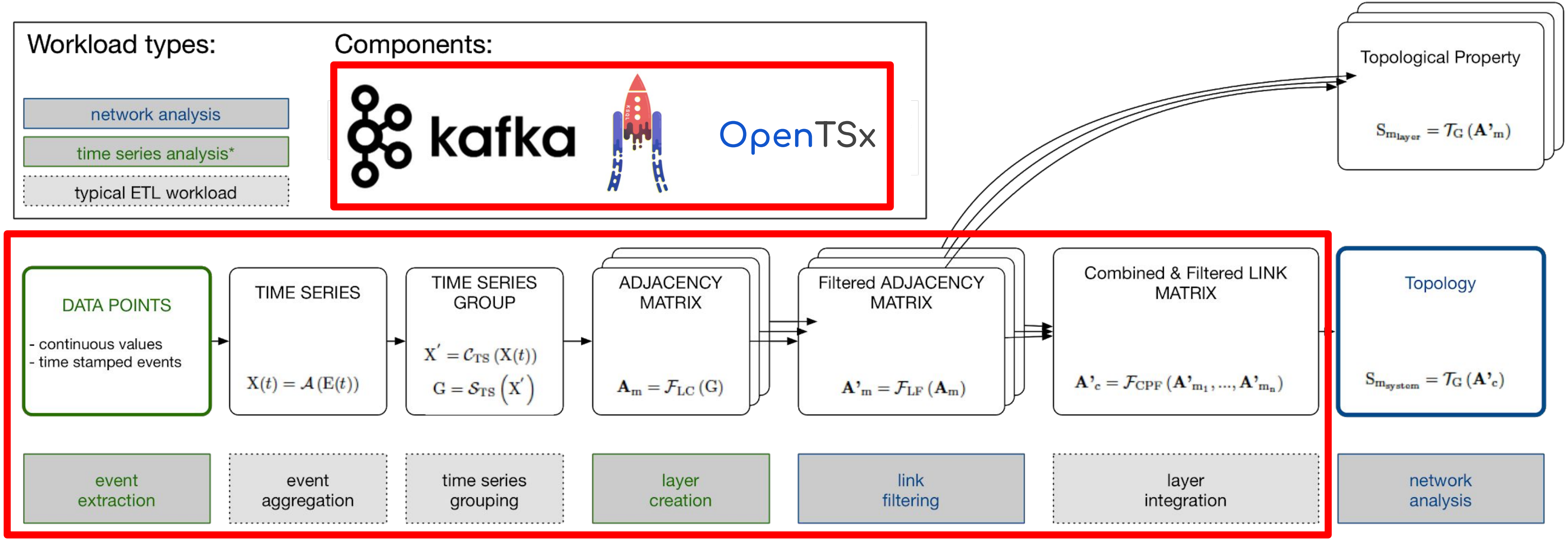
Network Reconstruction & Topology Analysis:

The Approach



Network Reconstruction & Topology Analysis:

A Standardized Event Processing Pipeline

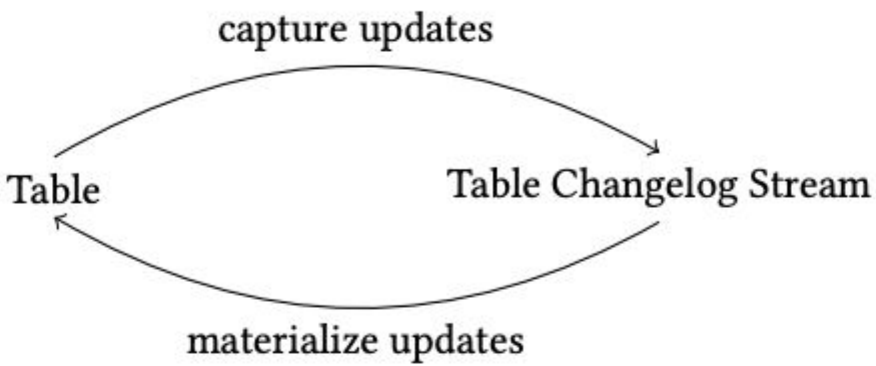


ADVANCED
TIME SERIES ANALYSIS &
NETWORK ANALYSIS

... and how does this fit into Kafka?

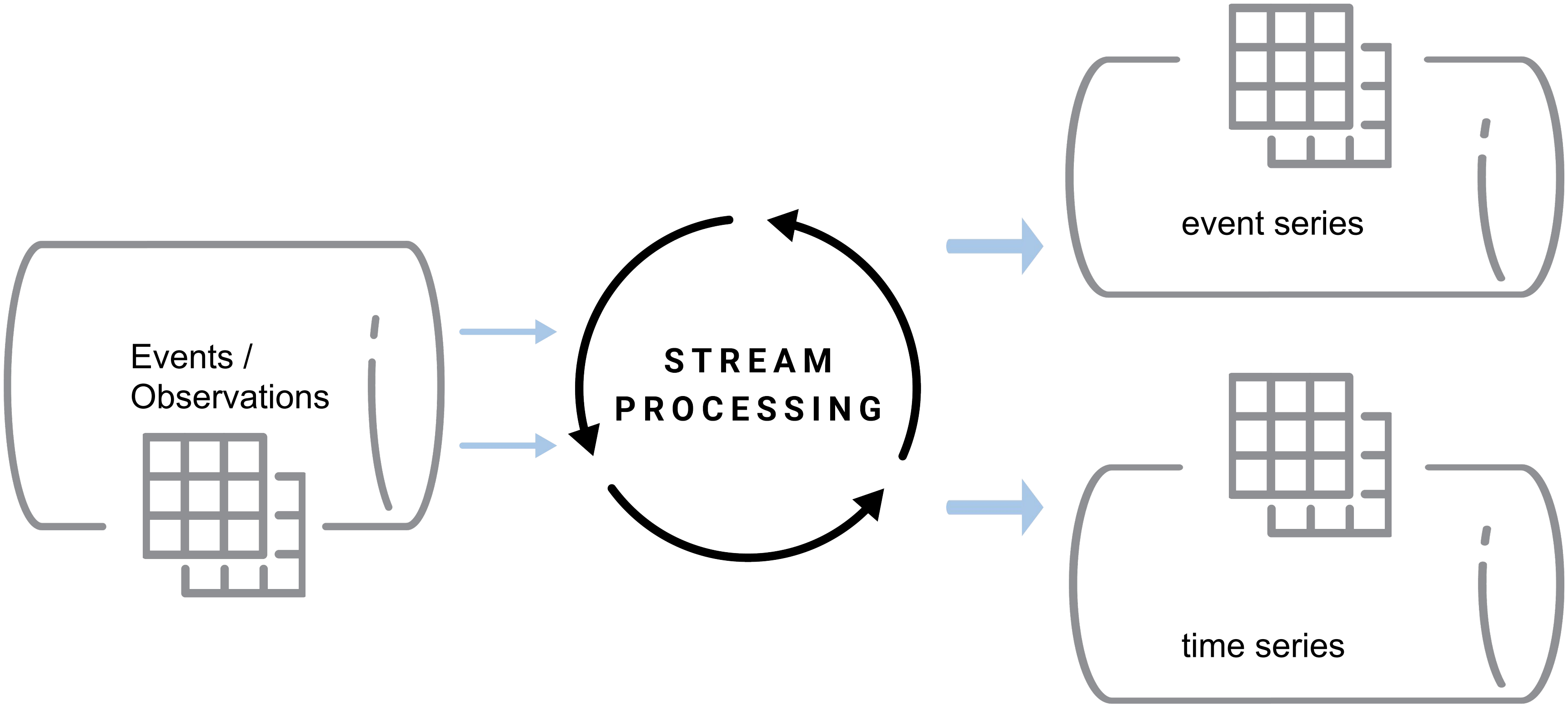
The following slides will show how TSA concepts and Kafka concepts fit together.

Table Stream Duality



BIRTE '18, August 27, 2018, Rio de Janeiro, Brazil M. J. Sax, G. Wang, M. Weidlich, J.-C. Freytag

Create Time Series from Event Streams:
By Aggregation, Grouping, and Sorting



From Table of Events to - Time Series

Table 1: Operators with their input and output types

Operator	1st Input	2nd Input	Output
`filter`, `mapValue`	KStream		KStream
	KTable		KTable
`map`, `flatMap`	KStream		KStream
`groupBy` $\longrightarrow$ `agg`	KStream		KTable
	KTable		KTable
`groupBy` + `windowBy` $\longrightarrow$ `agg`	KStream		KTable
`inner-/left-/outer-join`	KStream	KStream	KStream
`inner-/left-/outer-join`	KTable	KTable	KTable

BIRTE '18, August 27, 2018, Rio de Janeiro, Brazil

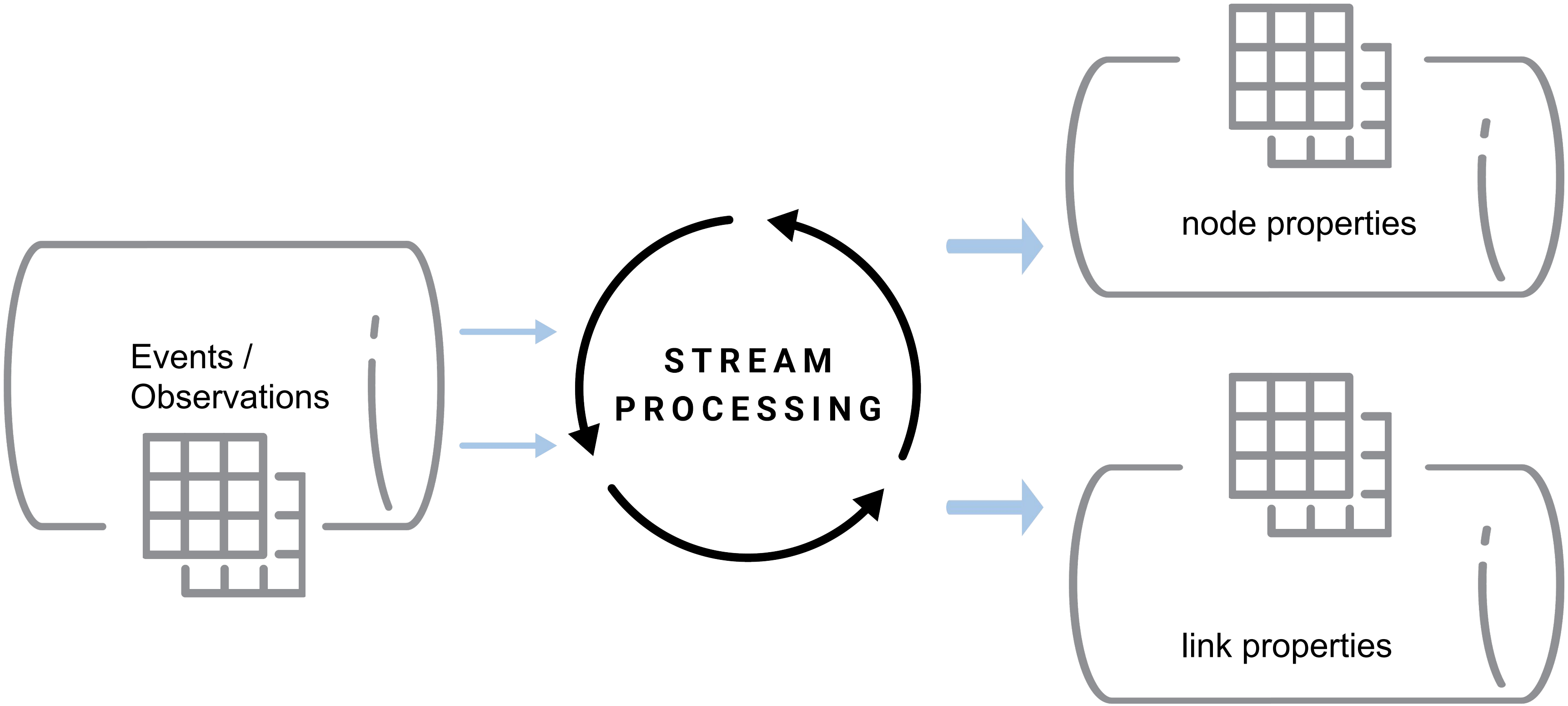
M. J. Sax, G. Wang, M. Weidlich, J.-C. Freytag

Table Stream Duality ⇒ Time Series and Graphs

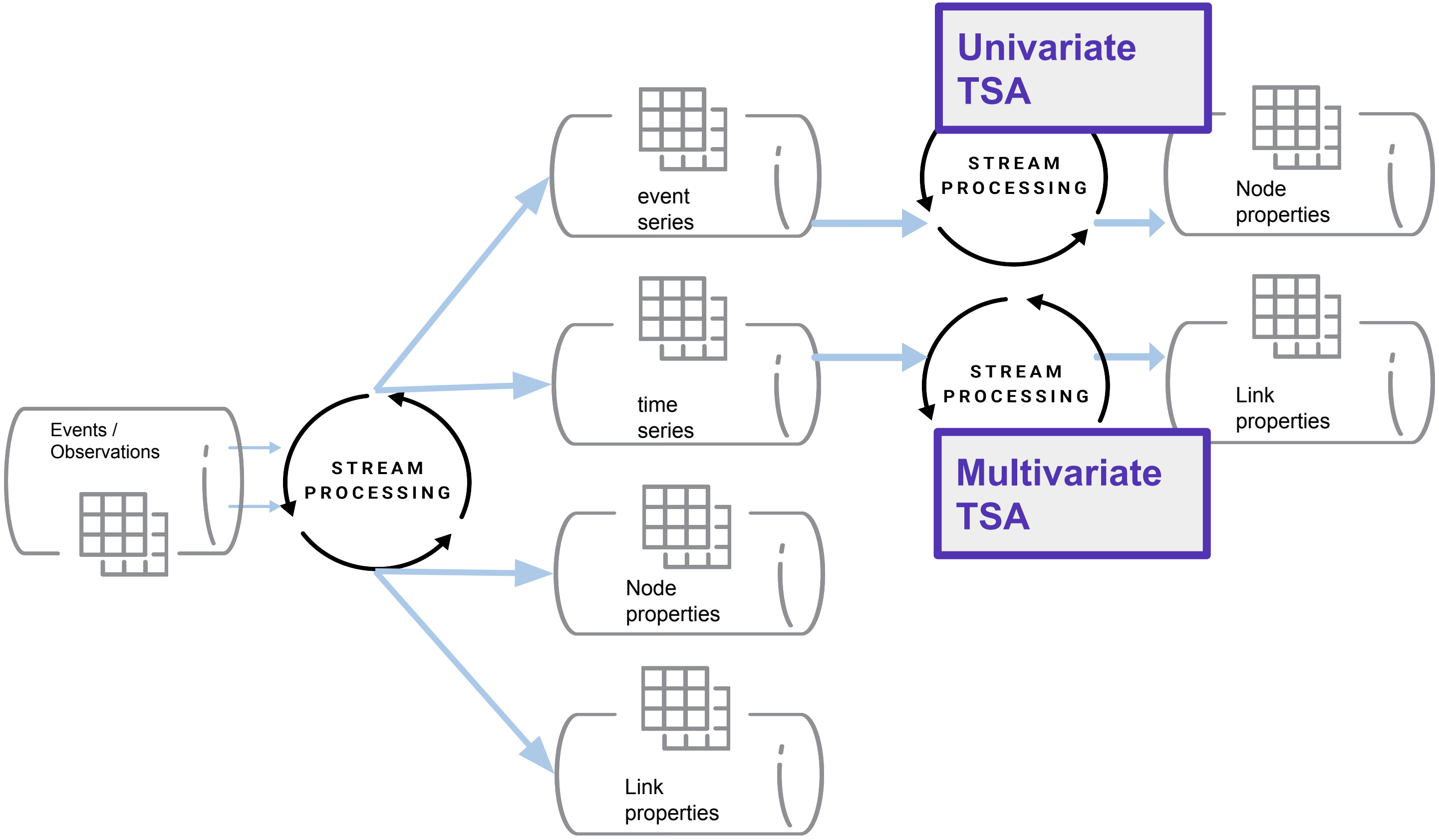
A ***time series*** is a table of **ordered observations** in a fixed context.

A ***graph*** can be seen as a table of node- and link-properties - stored in **two tables**.

Create Networks from Event Streams:
By Aggregation, Grouping, and Sorting



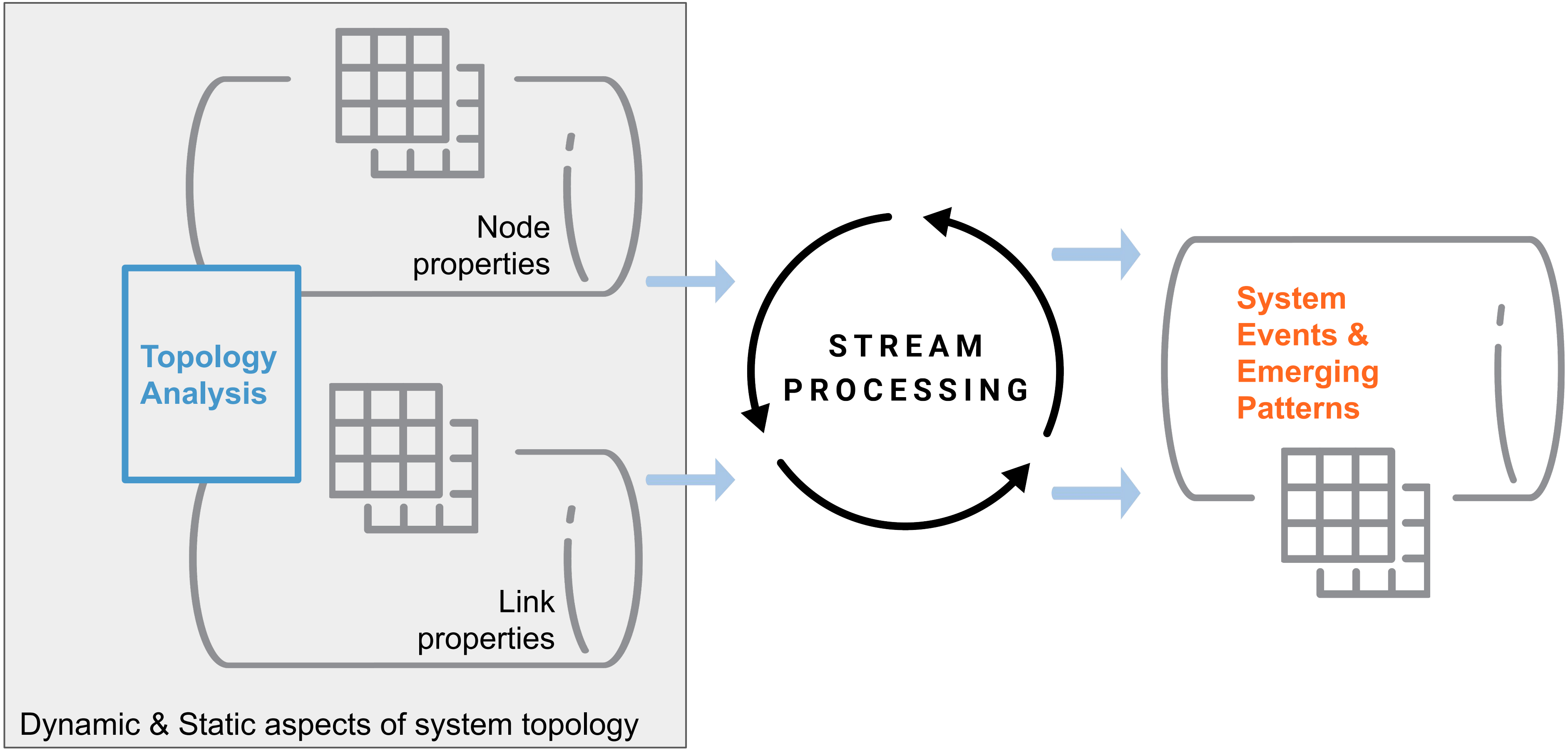
Multi Layer Stream Processing:
TSA to Reveal Hidden System Properties



Dynamic aspects of system topology

Static aspects of system topology

Complex Event Processing: For Complex Systems



Use the Table-Network Analogy: Kafka Graphs

large durable graphs:

Persisted in Kafka topic

Creating Graphs

A graph in Kafka Graphs is represented by two tables from Kafka Streams, one for vertices and one for edges. The vertex table is comprised of an ID and a vertex value, while the edge table is comprised of a source ID, target ID, and edge value.

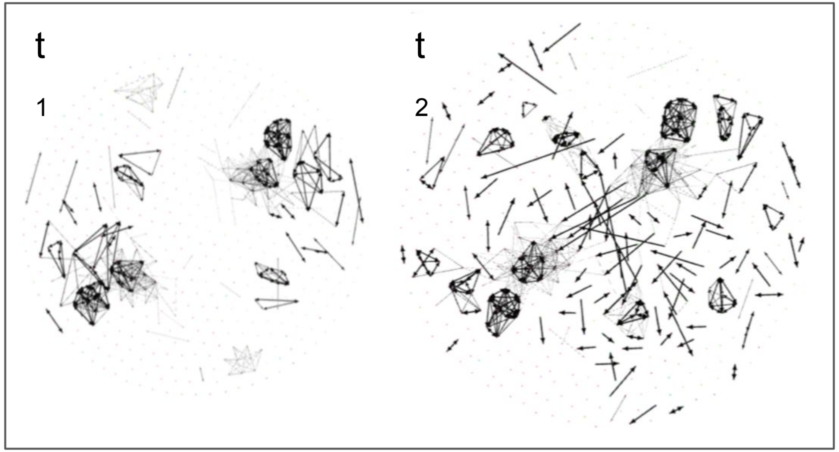
```
KTable<Long, Long> vertices = ...
KTable<Edge<Long>, Long> edges = ...
KGraph<Long, Long, Long> graph = new KGraph<>(
    vertices,
    edges,
    GraphSerialized.with(Serdes.Long(), Serdes.Long(), Serdes.Long())
);
```

https://github.com/rayokota/kafka-graphs

Sliding Windows: Define the Temporal Graphs

In some use cases, we don't want to keep the node and link data in topics:

- nodes aren't always linked
- changes are very fast
- focus on activation patterns,
  rather than on underlying structure



It is fine to calculate the correlation links
and the topology metrics on the fly,
just for a given time window.

Back to Streams

Demo ...

TSAExample1 › main()

```java
public static void main(String[] args) throws Exception {

  md = MessageDigest.getInstance("MD5");

  StreamsConfig streamsConfig = new StreamsConfig( props );
  StreamsBuilder builder = new StreamsBuilder();

  Serde<String> stringSerde1 = Serdes.String();
  Serde<String> stringSerde2 = Serdes.String();
  final Map<String, String> serdeConfig = Collections.singletonMap("schema.registry.url",
          props.getProperty( "schema.registry.url" ));

  final Serde<Event> valueSpecificAvroSerde = new SpecificAvroSerde<>();
  valueSpecificAvroSerde.configure(serdeConfig, isKey: false); // `false` for record values

  KStream<String, Event> my_events = builder.stream( topic: "OpenTSx_Events", Consumed.with( stringSerde1, valueSpecificAvroSerde));

  my_events = my_events.mapValues( (v) -> getCompactedEvent(v) );
  my_events.to( topic: "EVENT_DATA_TRAFO_" + EXPERIMENT_TAG, Produced.with( stringSerde2, valueSpecificAvroSerde) ) ;
  my_events.print(Printed.<~>toSysOut().withLabel("[EVENT_DATA]"));

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

  KafkaStreams kafkaStreams = new KafkaStreams(topology, streamsConfig);
  kafkaStreams.setUncaughtExceptionHandler((t, e) -> { LOG.error("had exception ", e); });

  LOG.info("Starting event processing example { experiment_duration=" + experiment_duration + " ms}.");
  kafkaStreams.cleanUp();
  kafkaStreams.start();

  if ( experiment_duration > 0 ) { Thread.sleep(experiment_duration); }

  LOG.info("Shutting down the event processing example application now.");
  kafkaStreams.close();

}

private static void dumpTopology(Topology topology) throws IOException {
  FileWriter fw = new FileWriter( fileName: props.getProperty(StreamsConfig.APPLICATION_ID_CONFIG) + "_topology.dat");
  fw.write( topology.describe().toString() );
  fw.flush();
  fw.close();
}
```

The project will provide reusable streaming apps so that developers implement just their specific logic.

```java
package org.opentsx.udf;

import io.confluent.ksql.function.udf.Udf;
import io.confluent.ksql.function.udf.UdfDescription;
import io.confluent.ksql.function.udf.UdfParameter;

/**
 * select mdExtract( TSTART, TEND ) as LENGTH FROM EPISODES_STREAM EMIT CHANGES;
 */
@UdfDescription(name        = "mdExtract",
                description = "Extracts metadata from single episods.",
                author      = "Mirko Kämpf",
                version     = "3.0.1" )
public class EpisodesMetadataExtractor {

    @Udf(description = "Calculates length of an episode (in ms).")
    public double calc_length(
            @UdfParameter(value = "TSTART", description = "START time") final long TSTART,
            @UdfParameter(value = "TEND", description = "END time") final long TEND
    )
    {
        return  (double)(TEND - TSTART);
    }

}
```

```java
/**
 * Simple threshold filter
 *
 * @param k - message key (is ignored in this example)
 * @param e - message value (our event)
 *
 * @return boolean value for filtering
 */
private static boolean getCriticalEvents(String k, Event e) {

  if ( e.getValue() > 3.0 ) return true;
  else return false;

}
```

```java
/**
 * getSimpleStats from TSO
 * -----------------------
 *
 * @param k - key (we extract the uuid which is part of the key and put it also into our result map.
 * @param v - the result map, calculated by the OpenTSx library. Simple descriptive statistics in this case.
 *
 * @return Hashtable with results encoded as JSON.
 */
private static String getSimpleStats(String k, EpisodesRecord v) {

  TimeSeriesObject tso = TimeSeriesObject.getFromEpisode( v );

  Hashtable<String,String> stats = tso.getStatisticData( new Hashtable<String,String>() );

  Map<String, String> retMap = new Gson().fromJson(
          k, new TypeToken<HashMap<String, String>>() {}.getType()
  );

  stats.put( "mkey" , retMap.get("uuid") );

  return gson.toJson( stats );

}
```

Finally, we can compose a complex flow ...

KSQLDB >

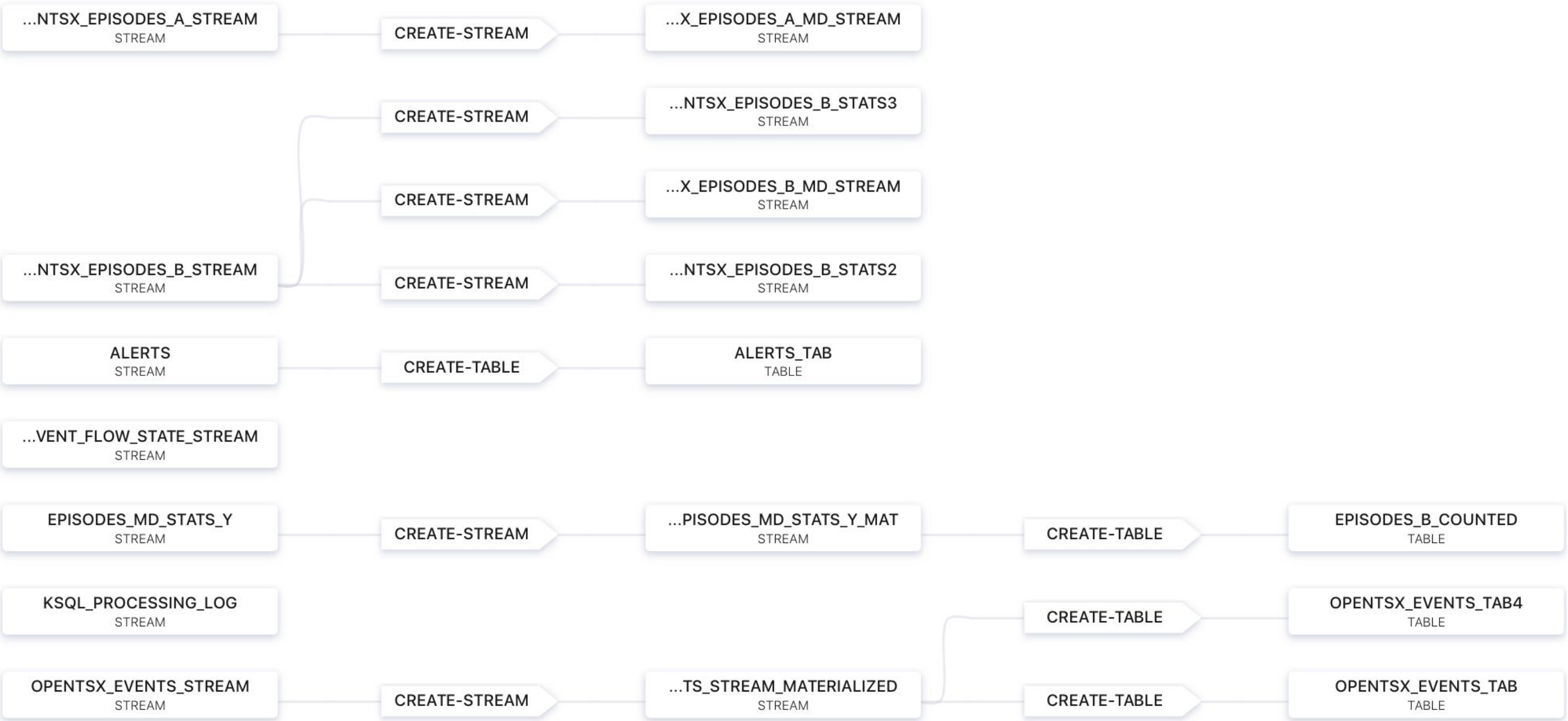
ksqlDB

Editor | Flow | Streams | Tables | Running queries

Search

```
ksql> select * from alerts_tab where URI='http://www.w3.org/TR/2004/metrics-owl-20200210/deviceID#0815a';

+----------------------------------------------------------+----------------------------------------------------------+----------------------------------------------------------+------
|URI                                                       |WINDOWSTART                                               |WINDOWEND                                                 |Z
+----------------------------------------------------------+----------------------------------------------------------+----------------------------------------------------------+------
|http://www.w3.org/TR/2004/metrics-owl-20200210/deviceID|1599496830000                                               |1599496845000                                             |234
|#0815a                                                    |                                                          |                                                          |
|http://www.w3.org/TR/2004/metrics-owl-20200210/deviceID|1599496845000                                               |1599496860000                                             |9677
|#0815a                                                    |                                                          |                                                          |
|http://www.w3.org/TR/2004/metrics-owl-20200210/deviceID|1599496860000                                               |1599496875000                                             |11705
|#0815a                                                    |                                                          |                                                          |
```



Demo: OpenTSx

Generate some events and some episodes.

Apply some time series processing procedures on:

- a stream of episodes.
- a stream of events

Complex procedures are composed from a set of fundamental building blocks.

Visualize the flows and dependencies.

Let’s Remember ...

We use 4 building blocks …

Domain specific logic is implemented in small reusable components:

UDFs:

Domain Driven Design

Source Connectors integrate sources …

Kafka Connect

Legacy and Future Systems

Data flows are no longer transient.
The event log acts as single source of truth.

Data Assets

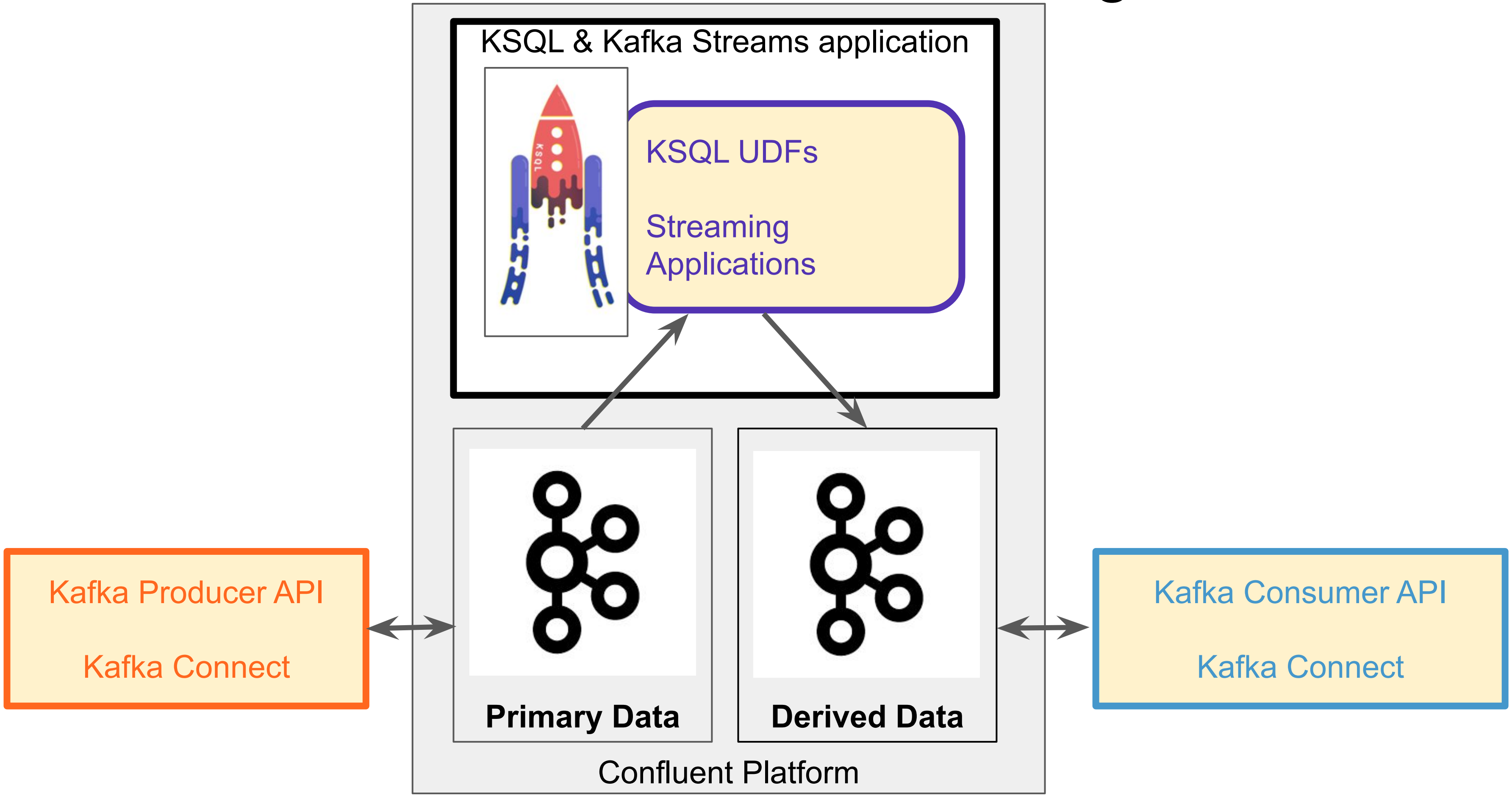
Paradigm Shift in Data Management

Sink Connectors integrate external targets …

Kafka Connect

Special Purpose Systems

… for our Time Series Processing Platform



Summary:

Because Kafka is a scalable & extensible platform it fits well for complex event processing in any industry on premise and in the cloud.

Kafka ecosystem provides extension points for any kind of domain specific or custom functionality - from advanced analytics to real time data enrichment.

Complex solutions are composed from a few fundamental building blocks:

What to do next?

(A) Identify **relevant main flows** and **processing patterns** in your project.

(B) Identify or implement source / sink **connectors** and establish 1st flow.

(C) Implement custom transformations as **Kafka independent components**.

(D) Integrate the processing topology as Kafka Streams application:

  (a) Do you apply standard transformations and joins (for enrichment)?

  (b) Is a special treatment required (advanced analysis)?

  (c) Do you need special hardware / external services (AI/ML for classification)?

(E) Share your connectors and UDFs with the growing Kafka community.

(F) Iterate, add more flows and more topologies to your environment.

END
Thank you !

mirko@confluent.io

@semanpix