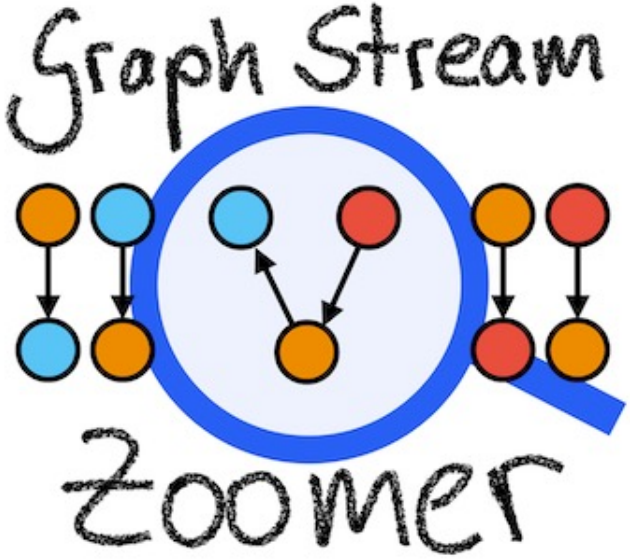
UNIVERSITÄT
LEIPZIG

ScaDS.AI
DRESDEN LEIPZIG

# Graph Stream Zoomer

## Distributed grouping of property graph streams




FOSDEM 2023 – Graph Devroom

https://github.com/dbs-leipzig/graph-stream-zoomer

**Christopher Rost**
rost@informatik.uni-leipzig.de
4rd February 2023

# About us

↑ **Prof. Dr. Erhard Rahm**
Head of database department
University of Leipzig


← **Christopher Rost**
PhD student since 2018

**Max Zimmer** →
Master student

**Elias Saalmann**
Alumnus - University of Leipzig

**Rana Noureddin**
Alumna - University of Leipzig

# What you should take away from this talk

- What is a property graph stream?
- Why should I group a graph stream?
- What is the “graph stream zoomer” and which grouping configuration leads to which results?
- What are the implementation challenges?
- How can I use the graph stream zoomer?

# Basics and motivation

- **What is an (event-) stream?**
- What is a graph stream?
- Why a graph stream?
- Why grouping of graph streams?

bike rental events

stock prices

online purchases

***event***
anything that happens at a clearly defined time and that can be specifically recorded

***event stream***
sequence of events ordered by time

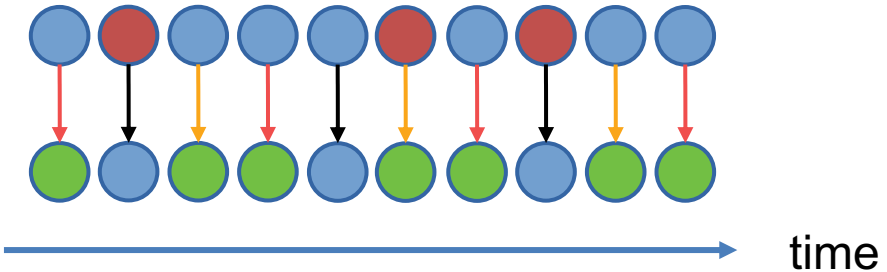
***event processing***
identify meaningful events and respond to them as quickly as possible



[image] https://www.nextbike.de/media/720/nb_0818_22_1.jpg

# Basics and motivation

- What is an (event-) stream?
- **What is a graph stream?**
- Why a graph stream?
- Why grouping of graph streams?




***graph stream***
event stream where an *event* is a graph element or update

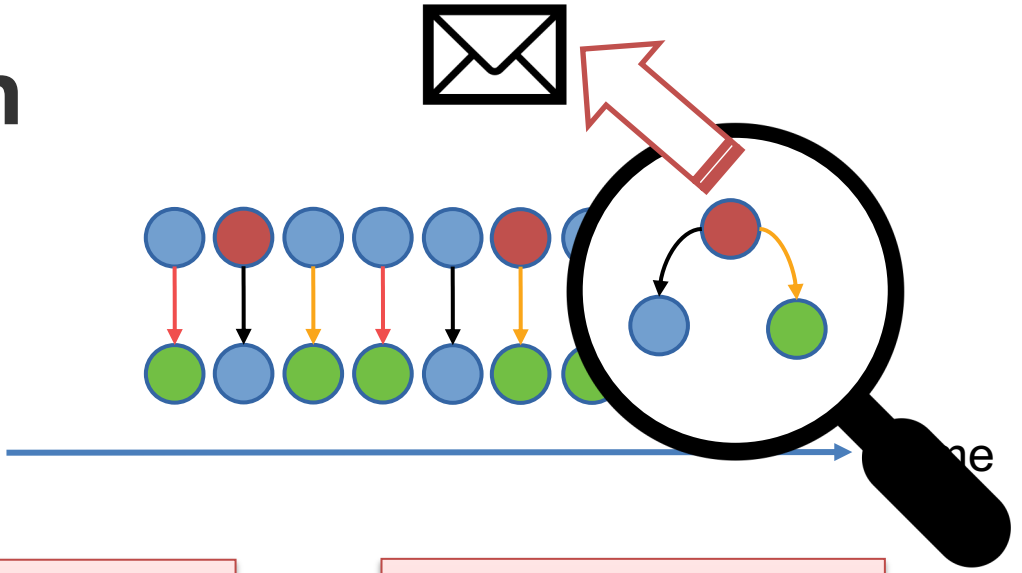
***graph element***
vertex, edge, triple

possibly labeled and attributed

***graph update***
modification of the graph structure and content, e.g., edge insertion/deletion

# Basics and motivation

- What is an (event-) stream?
- What is a graph stream?
- **Why a graph stream?**
- Why grouping of graph streams?



execution of graph analysis algorithms (e.g., PageRank) **concurrently** with graph updates

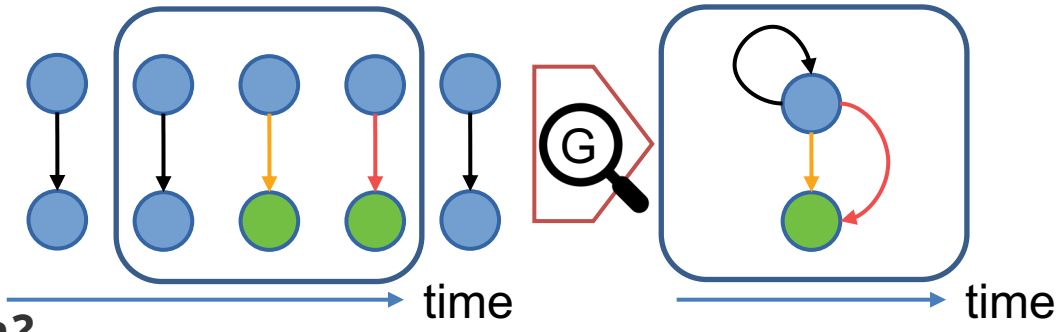
updates of analysis results with a low latency in (near) real time

***goal***

monitoring and/or notification/reactivity

# Basics and motivation

- What is an (event-) stream?
- What is a graph stream?
- Why a graph stream?
- **Why grouping a graph stream?**

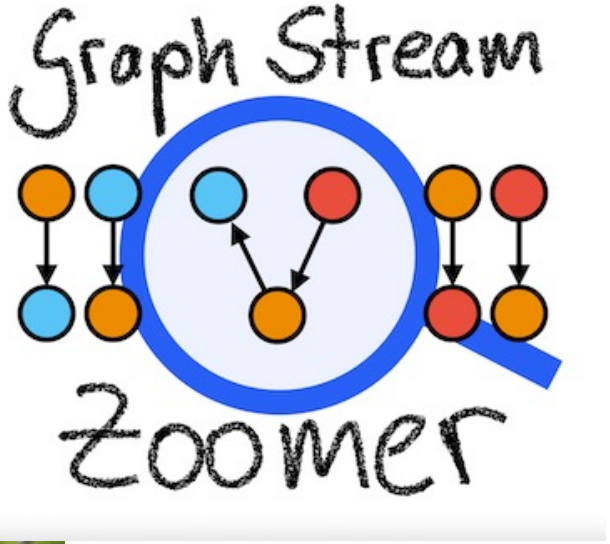



- graph streams may be heterogeneous and high frequent
- get overview and reduce complexity on different levels
- summarize graph elements/updates on similar characteristics
  - **time**, **structure**, **content** (label, properties)
  - via grouping key functions: f(v/e) → key
- get the grouping result “real-time” after graph update >>> again a graph stream

# Applications for graph stream grouping

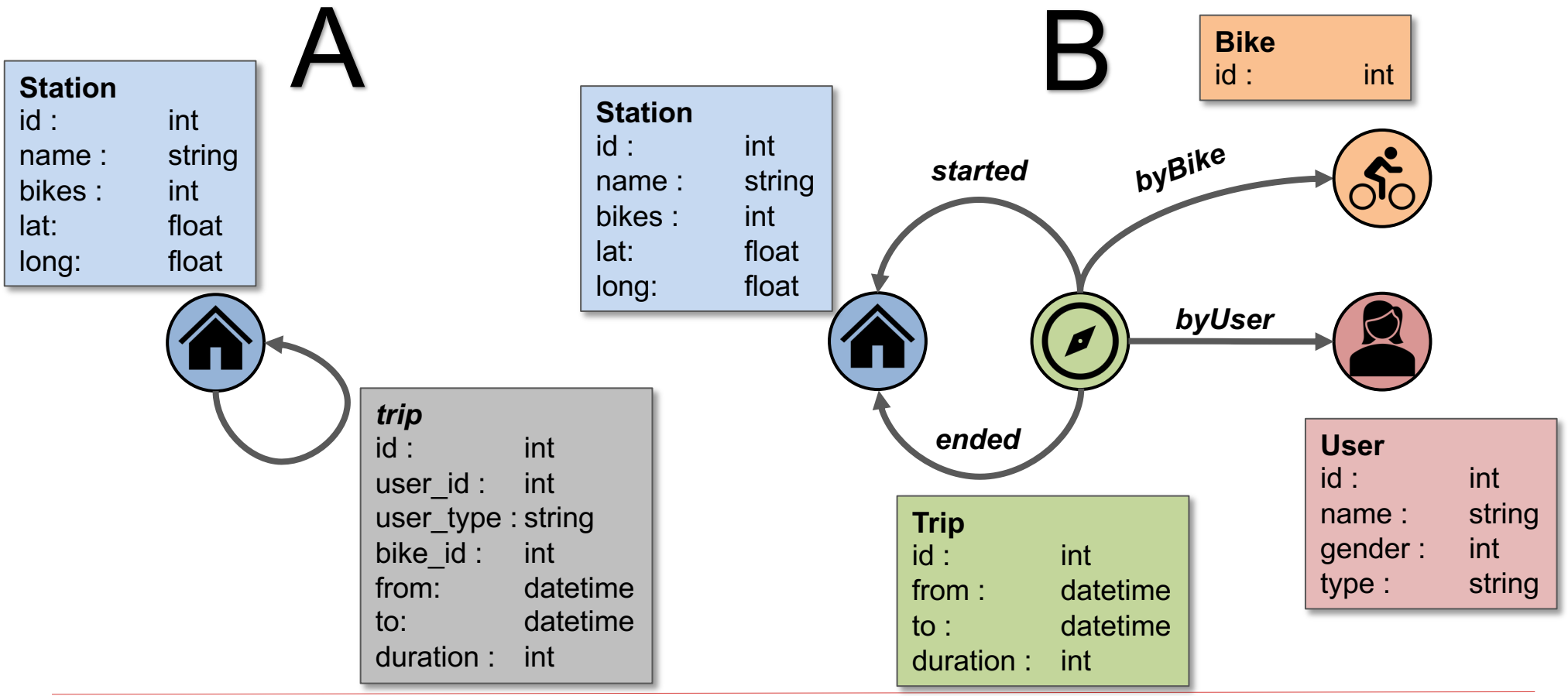
- **Pre-processing** for graph stream systems (ETL)
  - e.g., before PageRank, group graph stream on city attribute of users
- **Post-processing** after a applied graph stream analysis
  - e.g., after community algorithm, group elements with the same cluster id
- **Understanding** the graph stream (and its evolution)
  - Which vertex/edge types exist in the stream?
  - How frequent the different types arrive?
  - How vertices of different characteristics are connected with edges of certain characteristics?
- **Reveal hidden information** and get instantly notified
  - aggregation of attributes -> deeper insights
  - e.g., how an average value changes over time
  - notification by defining thresholds

UNIVERSITÄT
LEIPZIG

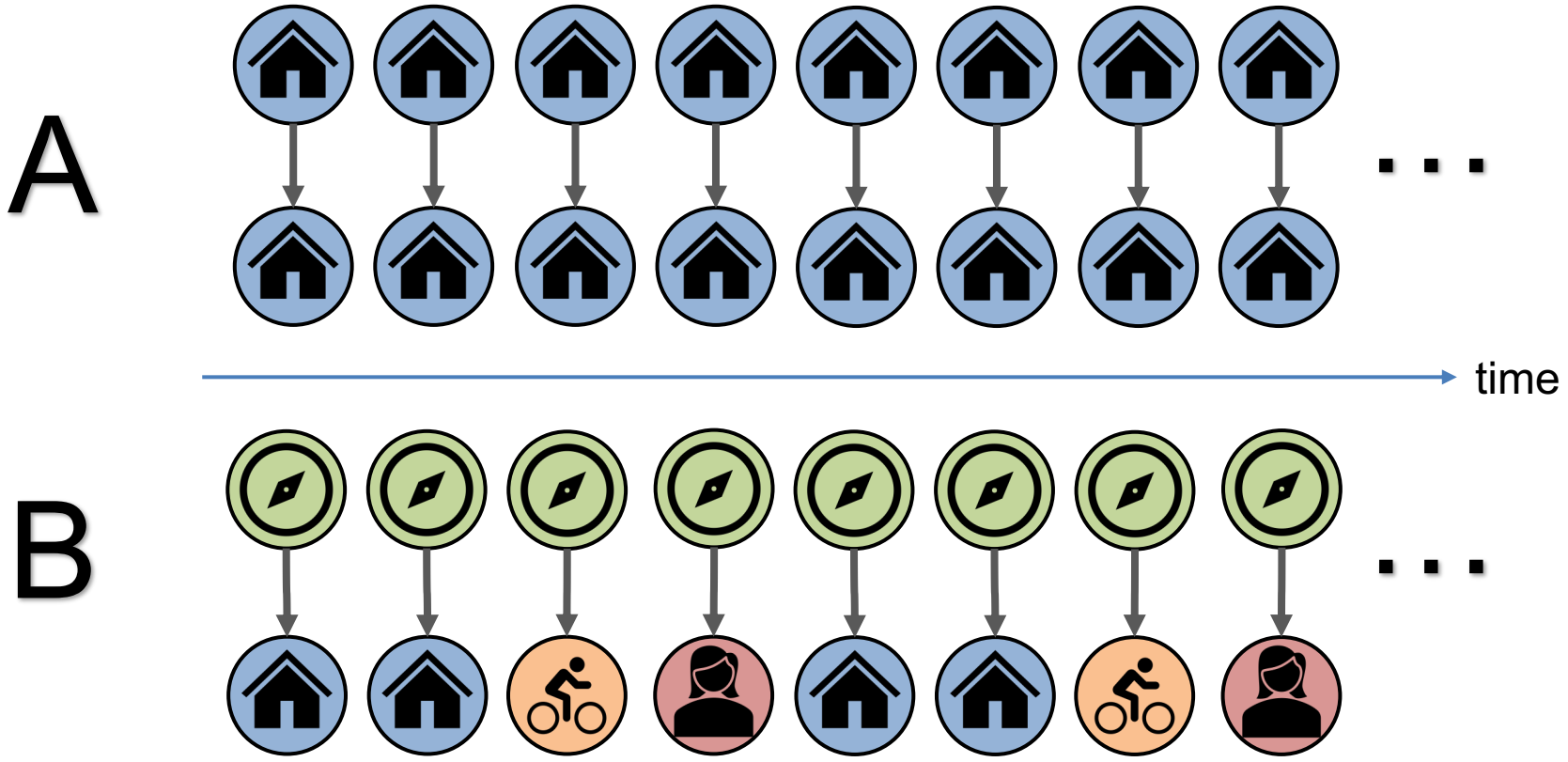



# by example

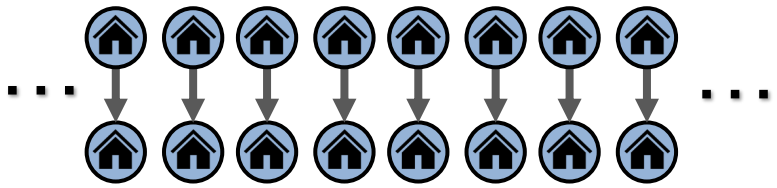
# Running example - Graph Schemas



# Running example - Graph Stream



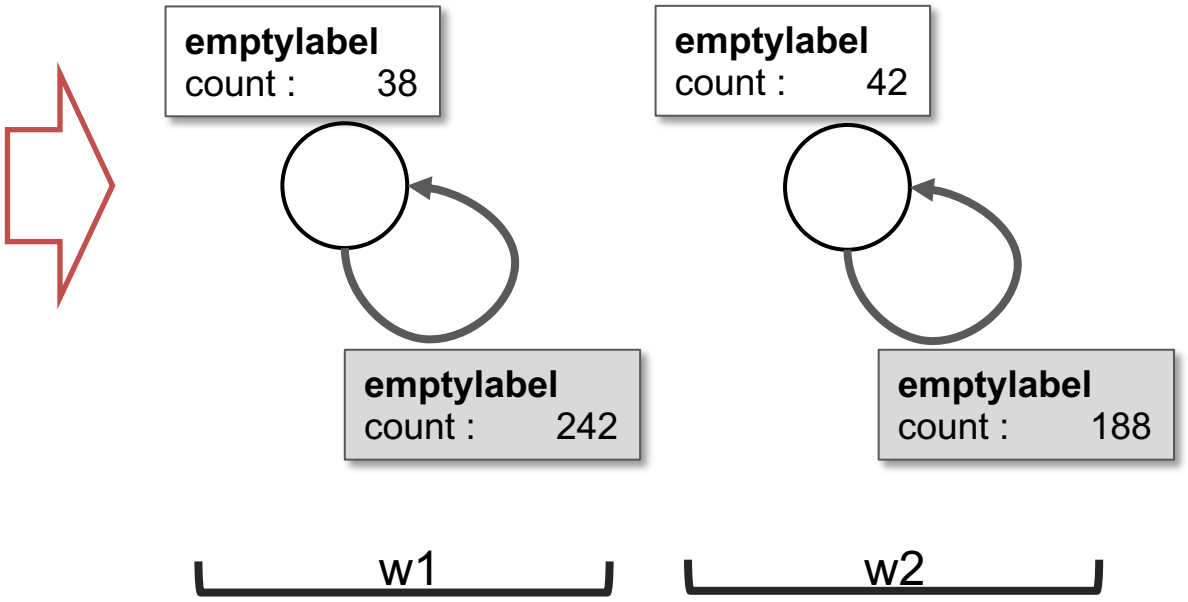
# Example 1 – Zoomed out (Schema A)



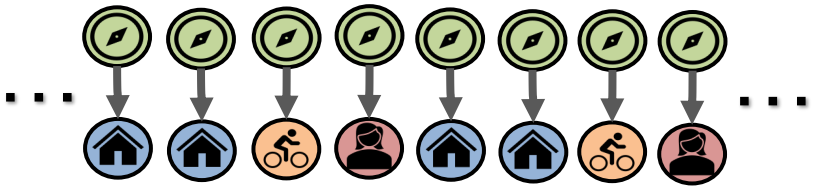
*Input stream*

***Grouping config***

| | |
|---|---|
| W: | 10m |
| VGK: | (v → 1) |
| EGK: | (e → 1) |
| VAgg: | count() |
| EAgg: | count() |



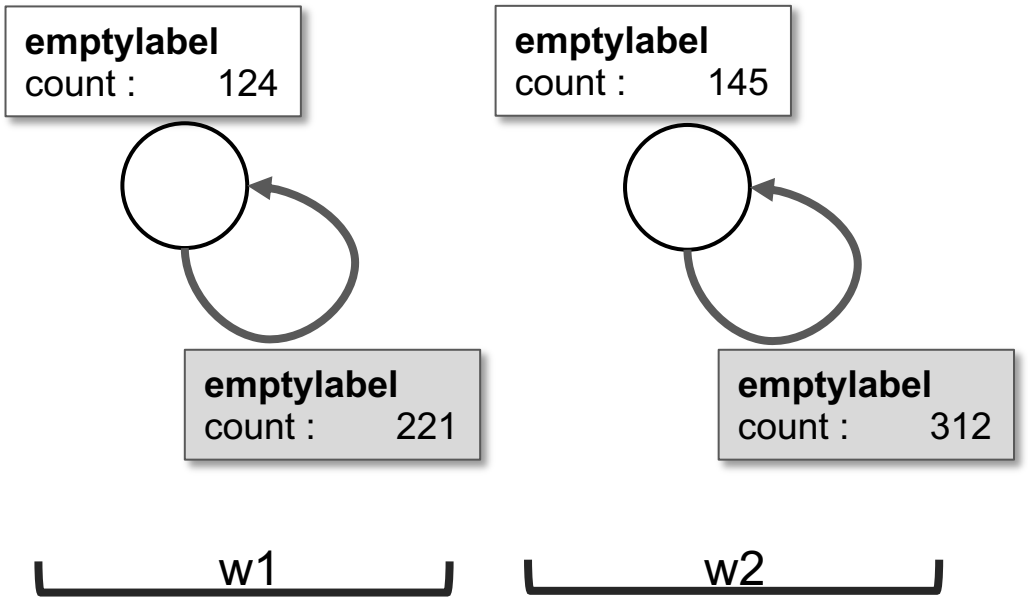
# Example 1 – Zoomed out (Schema B)



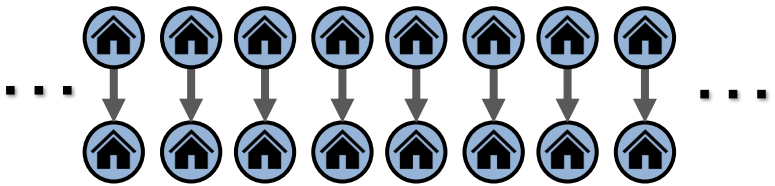
*Input stream*

**Grouping config**

| | |
|---|---|
| W: | 10m |
| VGK: | (v → 1) |
| EGK: | (e → 1) |
| VAgg: | count() |
| EAgg: | count() |



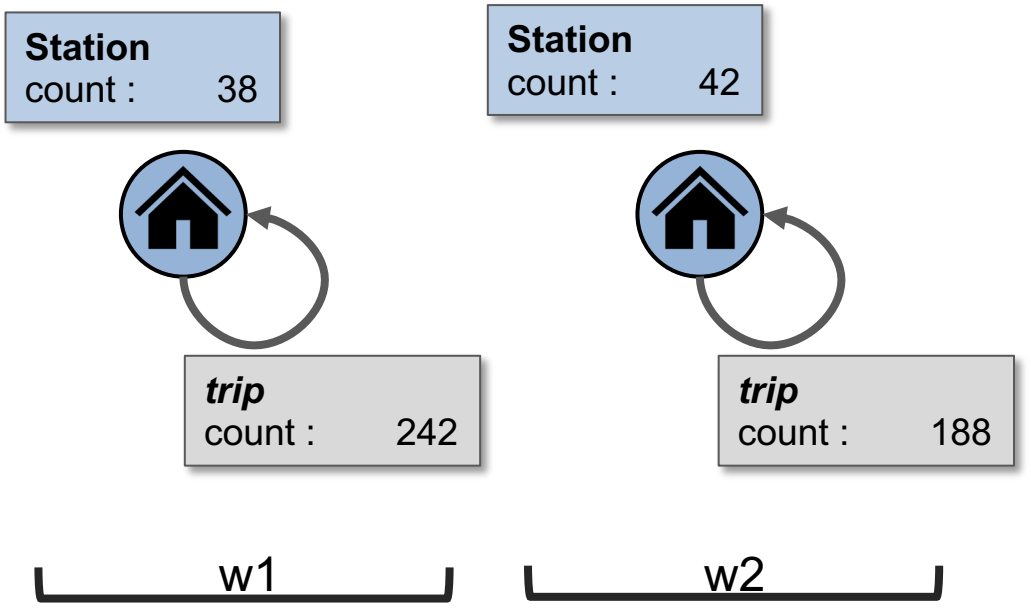
# Example 2 – Graph Stream Schema (A)



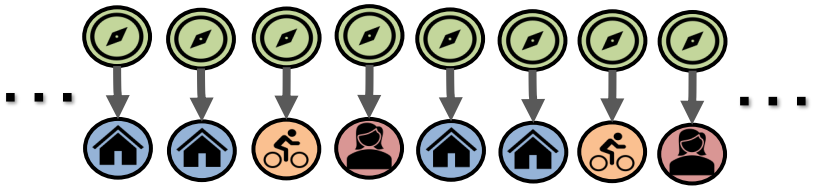
*Input stream*

**Grouping config**

| | |
|---|---|
| W: | 10m |
| VGK: | (v → label(v)) |
| EGK: | (e → label(e)) |
| VAgg: | count() |
| EAgg: | count() |



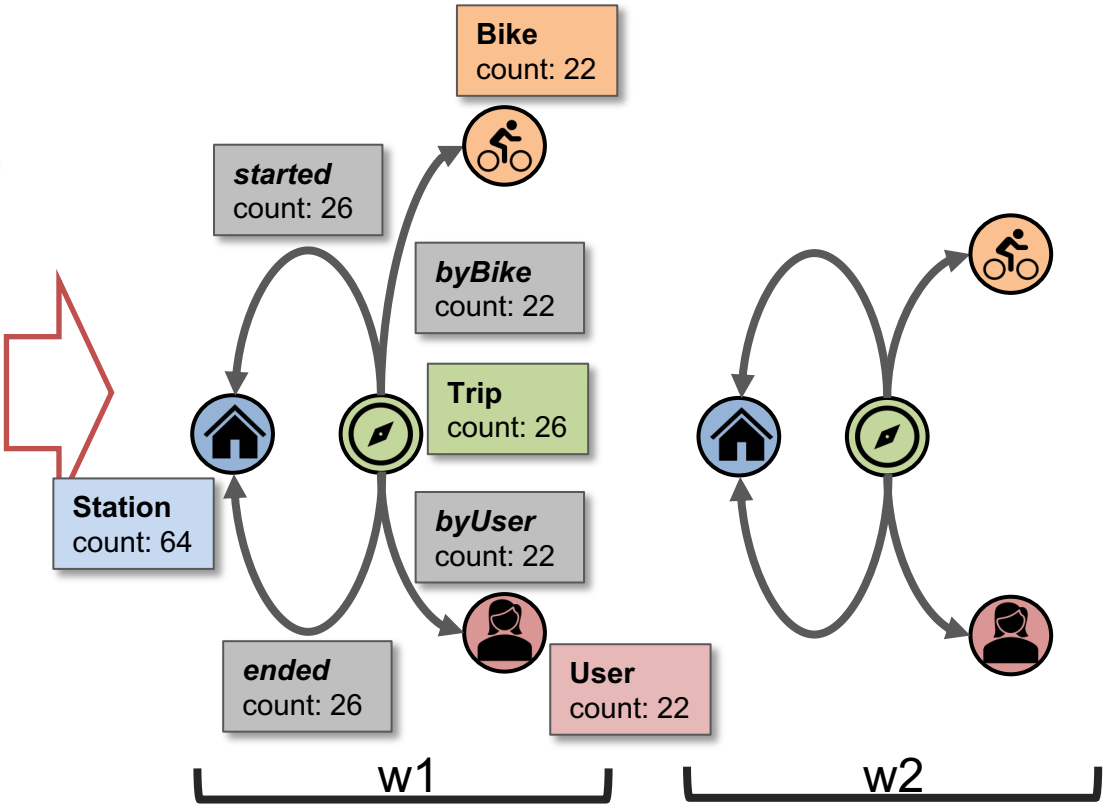
# Example 2 – Graph Stream Schema (B)



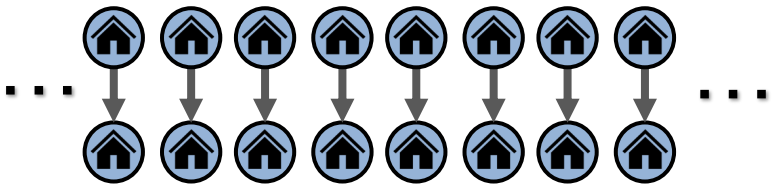
*Input stream*

**Grouping config**

| | |
|---|---|
| W: | 10m |
| VGK: | (v → label(v)) |
| EGK: | (e → label(e)) |
| VAgg: | count() |
| EAgg: | count() |



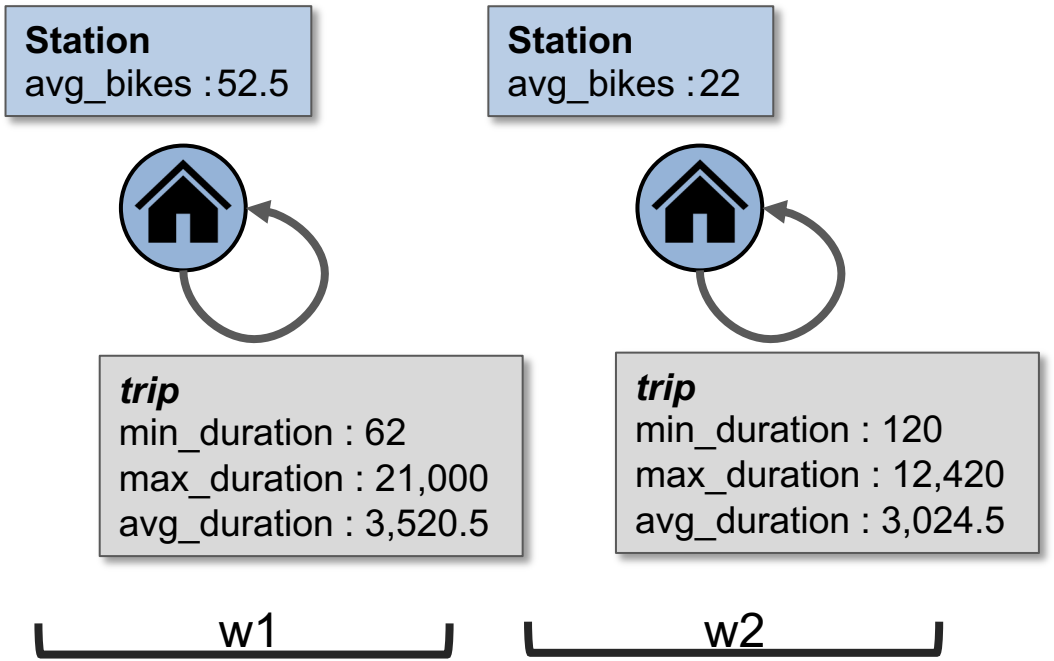
# Example 3 – Schema with aggregates (A)



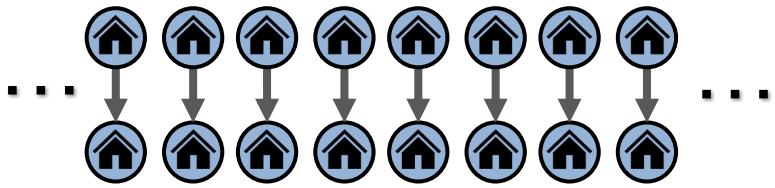
*Input stream*

**Grouping config**

| | |
|---|---|
| W: | 10m |
| VGK: | (v → label(v)) |
| EGK: | (e → label(e)) |
| VAgg: | avg(v.bikes) |
| EAgg: | min(e.duration) |
| | max(e.duration) |
| | avg(e.duration) |



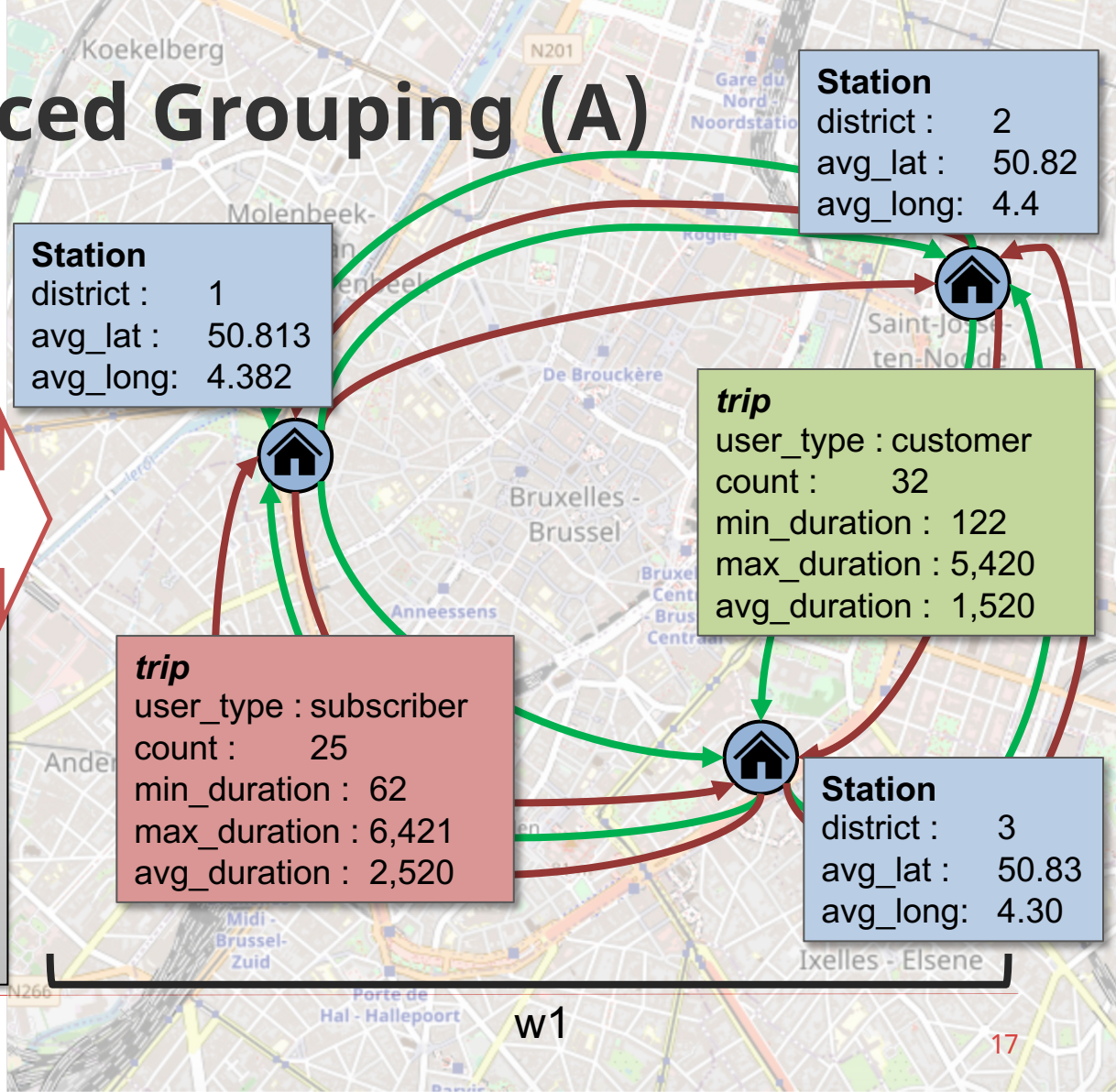
# Example 4 – Advanced Grouping (A)



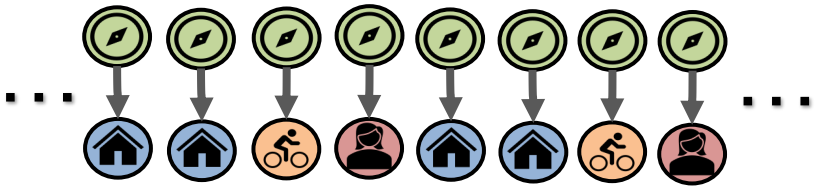
***Input stream***

***Grouping config***

| | |
|---|---|
| W: | 10m |
| VGK: | (v → label(v)) |
| | (v → getDistrict(v.lat,v.long)) |
| EGK: | (e → label(e)) |
| | (e → e.user_type) |
| VAgg: | avg(v.lat),avg(v.long) |
| EAgg: | count() |
| | min(e.duration) |
| | max(e.duration) |
| | avg(e.duration) |



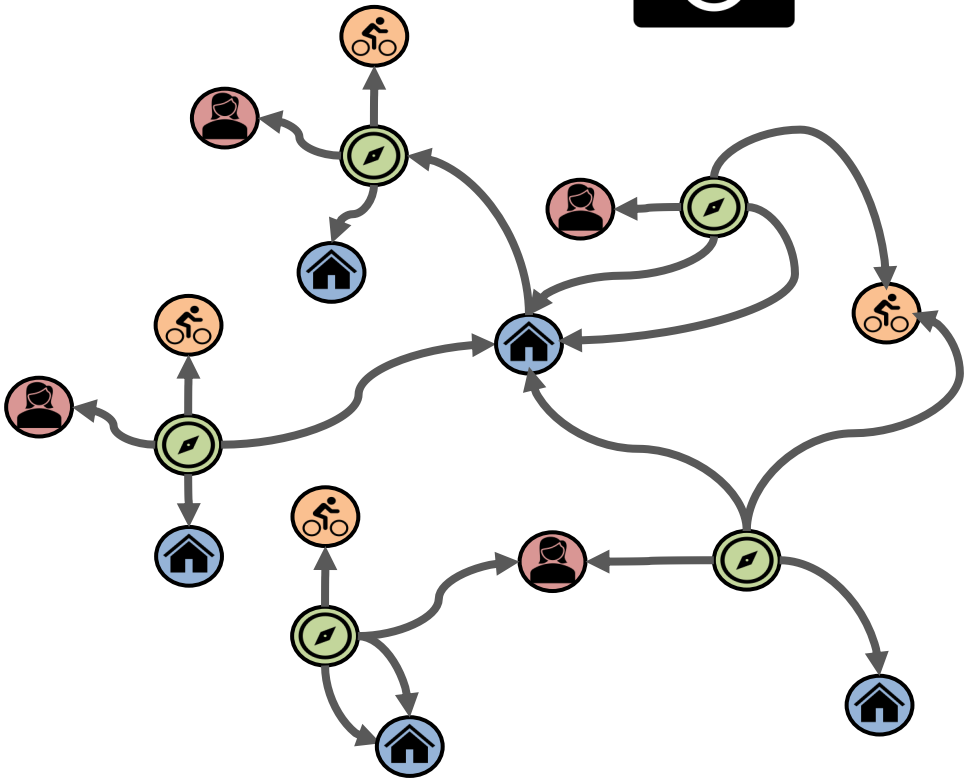
# Example 5 – Zoomed In (B)



*Input stream*

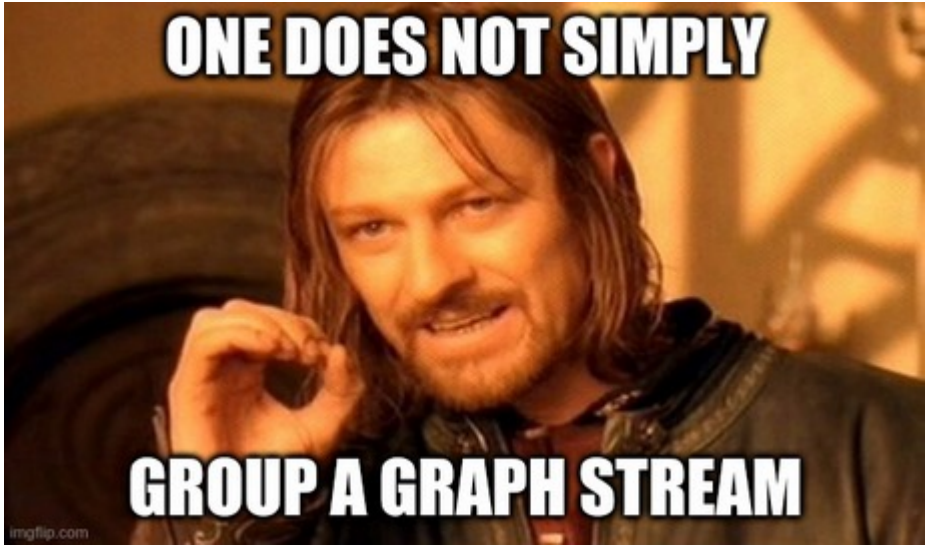
**Grouping config**

| | |
|---|---|
| W: | 60m |
| VGK: | (v → label(v)) |
| | (v → v.id) |
| EGK: | (e → label(e)) |
| | (e → e.id) |
| VAgg: | - |
| EAgg: | - |



w1

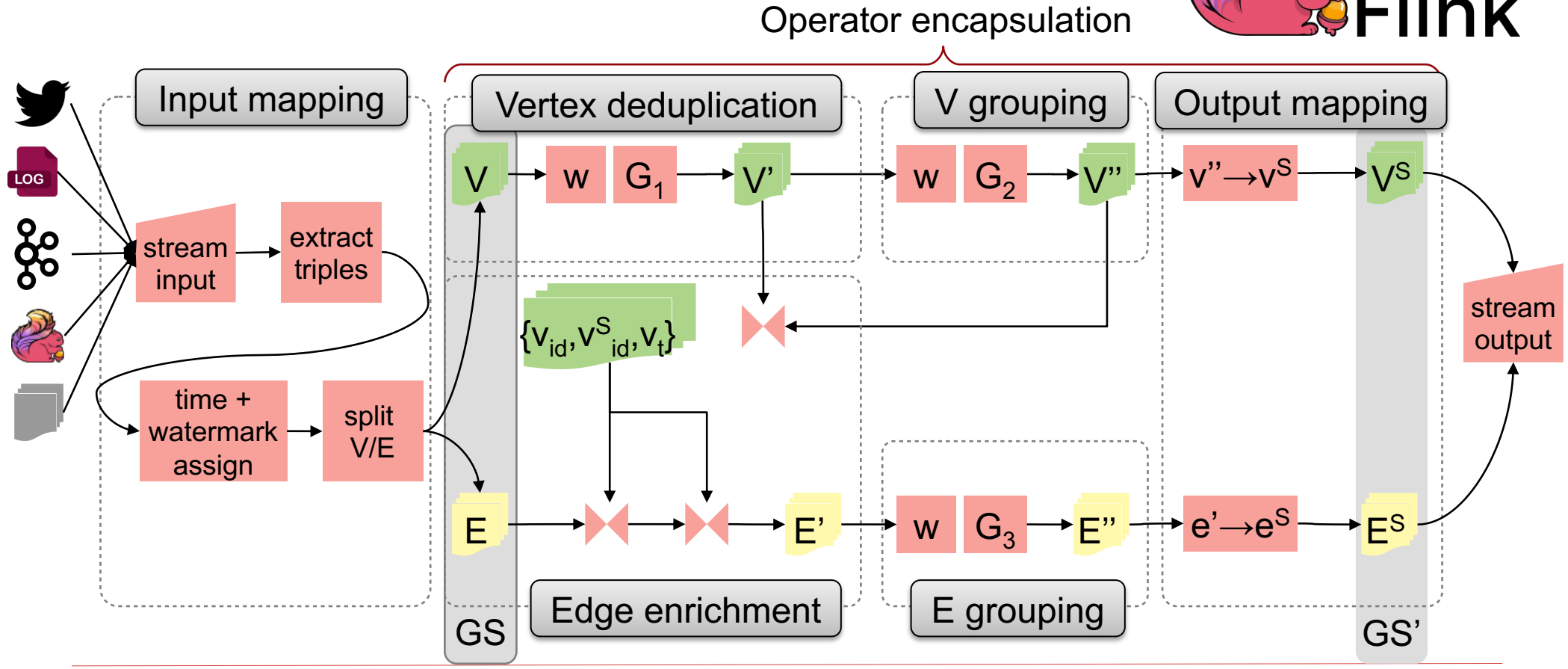
# Implementation challenges



# Implementation challenges

- Find a optimal **graph representation** in the streaming model
  - triple stream, vertex- and edge streams, adjacency lists/arrays
- Ensure **chronological order** after each step in the processing pipeline
  - use watermarks to prevent out-of-order events
- Ensure **scalability** of the pipeline parts (low communication overhead)
- Ensure a **finite** and minimized internal **state** of each processing step
  - e.g., join needs temporal predicate to clean up state
- Low latency / high throughput / high scalability (scale in/out)

# Grouping algorithm overview



# Exemplified operator call

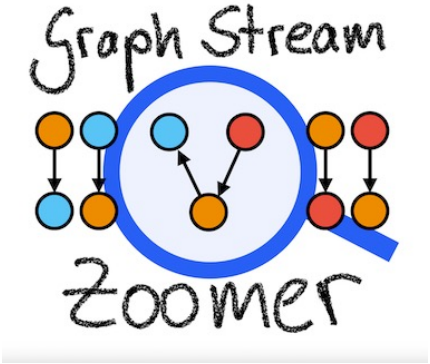
```
// Init the stream environment
final StreamExecEnvironment env = StreamExecEnvironment.createLocalEnv();
// Create the triple stream from a csv file
DataStream<StreamTriple> citiBikeStream = createInputFromCsv(env);
// Init the StreamGraph - our internal representation of a graph stream
StreamGraph sg = StreamGraph.fromFlinkStream(citiBikeStream, new Config(env));
// Configure and build the grouping operator
GraphStreamGrouping groupingOperator = new TableGroupingBase.GroupingBuilder()
  .setWindowSize(15, WindowConfig.TimeUnit.DAYS)
  .addVertexGroupingKey(":label")
  .addEdgeGroupingKey(":label")
  .addVertexAggregateFunction(new Count())
  .addEdgeAggregateFunction(new AvgProperty("tripduration")).build();

// Execute the grouping and overwrite the input stream with the grouping result
streamGraph = groupingOperator.execute(streamGraph);
// Print the result stream to console
streamGraph.printVertices();
// Trigger the workflow execution
env.execute();
```

UNIVERSITÄT
LEIPZIG

# Current state and future work

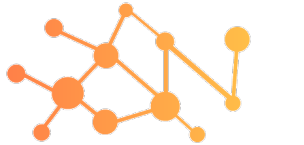
- prototypical implementation using Apache Flink's Table API at 90%
- bug at the Flink planner not fixed yet -> workaround via SQL API
  - https://issues.apache.org/jira/browse/FLINK-22530
- evaluation planned
  - latency, throughput, scalability, different grouping setups
  - on real-world and synthetic graph streams
- user-defined key and aggregate functions

UNIVERSITÄT
LEIPZIG





*That's all folks!*



**Temporal**
Graph Explorer

Graph Stream Zoomer >> https://github.com/dbs-leipzig/graph-stream-zoomer
Gradoop >> https://github.com/dbs-leipzig/gradoop
Temporal Graph Explorer >> https://github.com/dbs-leipzig/temporal_graph_explorer