Continuous Query Engine to Detect Anomalous ATM Transactions

Fernando Martín Canfrán Daniel Benedí Amalia Duch
Edelmira Pasarella

Facultat d'Informàtica de Barcelona
Universitat Politècnica de Catalunya

October,2024

Overview

Motivation

- In general, today's applications

  $\Rightarrow$ Critical real time systems
  $\Rightarrow$ Results must be emitted as they are computed (incrementally)

- Data are in motion, continuously changing and (possibly) unbounded $\rightarrow$ stream data

  $\Rightarrow$ Evolving/Dynamic data sources

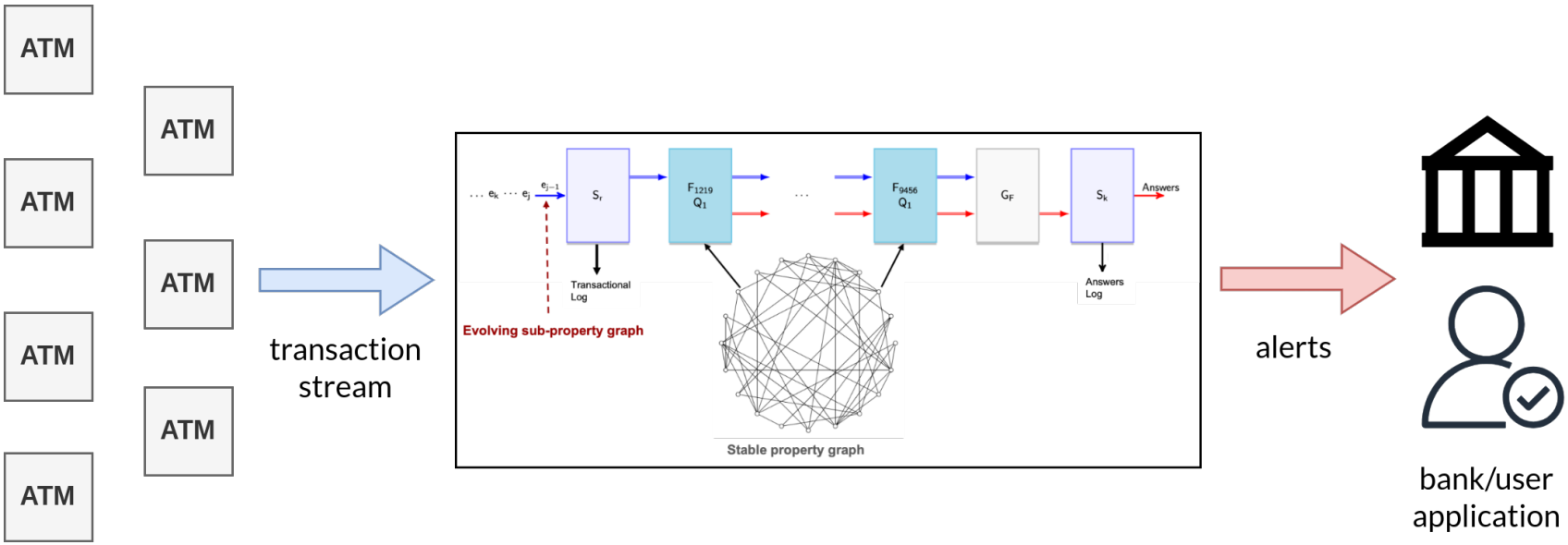
- Query evaluation

  $\Rightarrow$ Continuous queries
  $\Rightarrow$ Incremental/progressive query evaluation

Motivation

- Early detection of anomalous ATM transactions can prevent certain types of frauds
- This kind of tool can promote the design of policies of double authentication, benefiting both bank and users

Motivation


ATM
ATM
ATM
ATM
ATM
ATM
transaction stream
Transactional Log
Evolving sub-property graph
Answers
Answers Log
Stable property graph
alerts
bank/user application

Our Proposal

1. Data Model
2. Query Model
3. Continuous Query Evaluation (CQE)

   Stream Processing Approach - Dynamic Pipeline Computational Model

Our proposal:: The data model

- **Property graphs**: basic reference data model considered
- ***Continuous evolving database*** - data can be stable and volatile

⇒ **Continuosly evolving property data graph**:

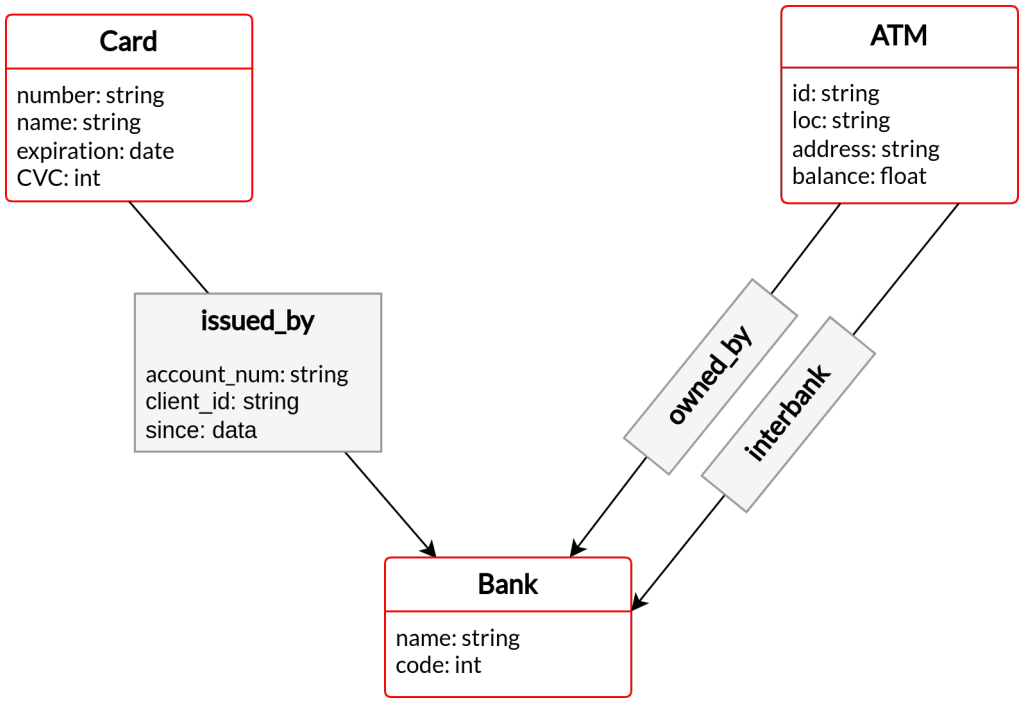
  ⇒ Stable PG: Central bank database - *persistent relations*

  ⇒ Volatile PG: Transactions - *non-persistent relations. Data stream*

Our proposal::The data model

⇒ Stable PG

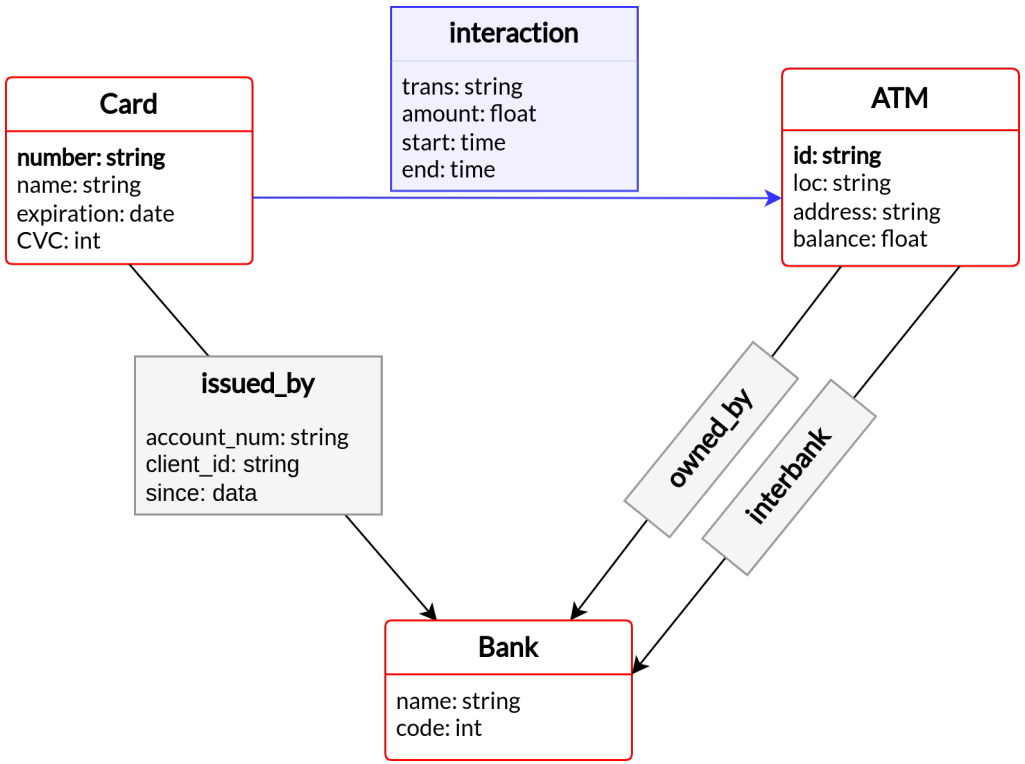
- Models the data a bank typically gathers on cards, ATMs...



Our proposal::The data model

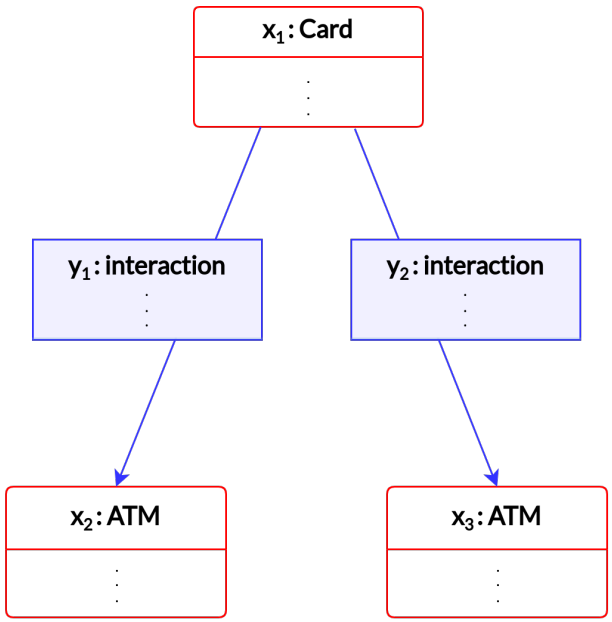
⇒ Volatile PG

- Volatile relations (transactions) are the edges arriving in data streams during a set time interval.
- Induce subgraphs that exist only while the relations are still valid.



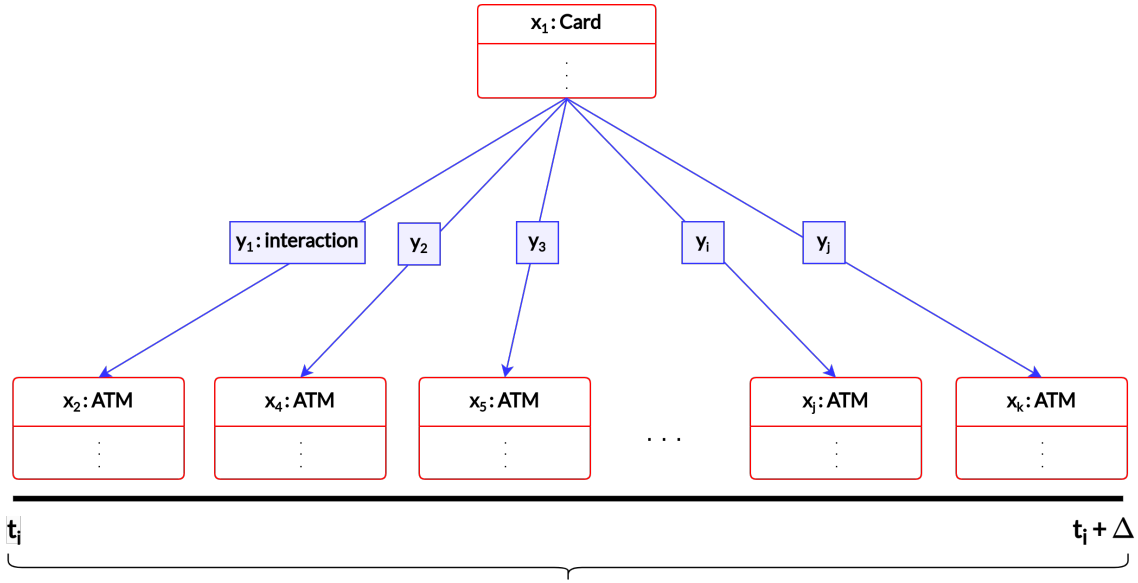
Our proposal:: The Query Model

- Continuous queries are characterized as (constrained) graph patterns



$x_2.id \neq x_3.id \wedge y_2.start - y_1.end < T_{min}(x_2.loc, x_3.loc)$

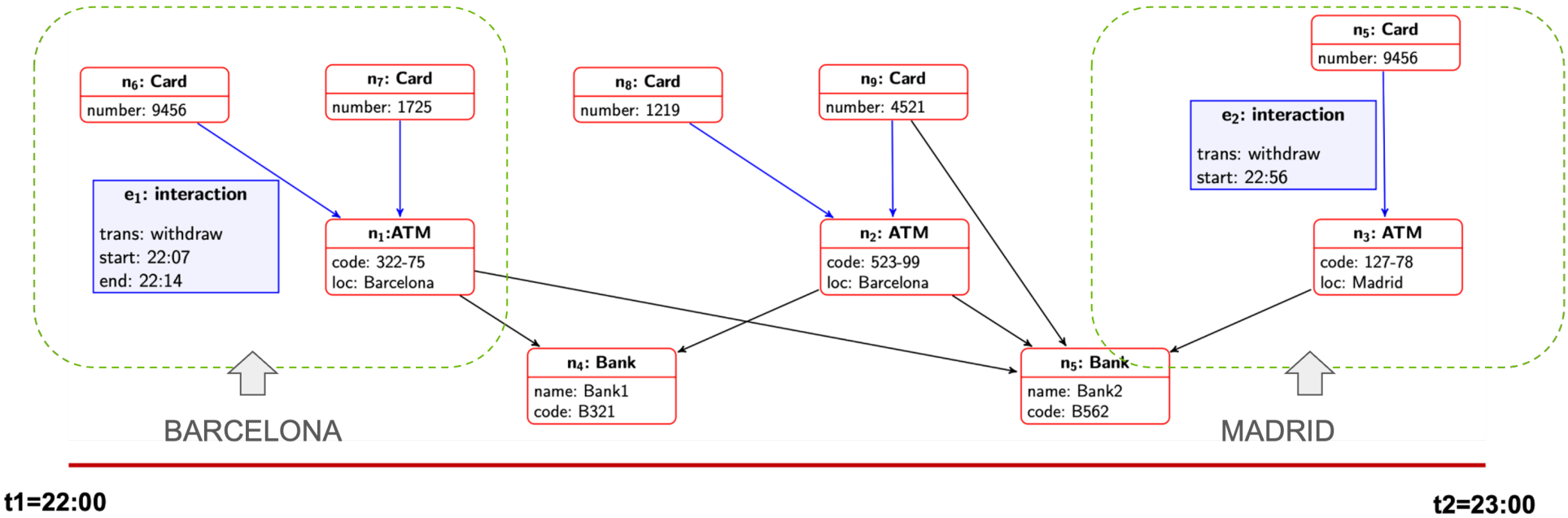
Card cloning



- ATMs close to each other (possibly in the same neighborhood/district/city)
- #(y:interaction) > fraud_threshold(t_i, $t_i + \Delta$)

Lost-and-stolen card

Our proposal:: The Query Model



Card cloning example

Our proposal:: Continuous Queries Evaluation

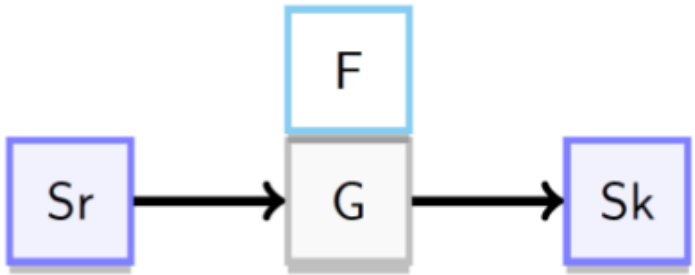
- Progressive query evaluation process. Based on:

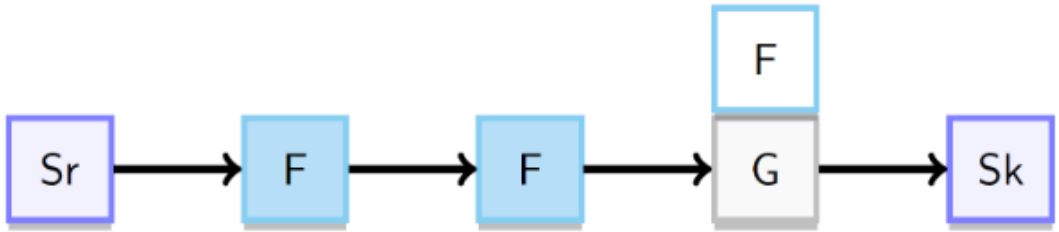
  Graph pattern matching + Satisfiability of its constraints

- Graph pattern matching in the volatile subgraph.
- Satisfiability of constraints over the properties: possibly querying the stable graph database (*retrieve some additional info*).
- Different kinds of anomalous patterns.
  - ⇒ *Card cloning* pattern
  - ⇒ Usage of (stolen) card many times over a period of time with small withdrawals - lost-and-stolen
    - Frequent/Very high expenses
    - Transactions in ATMs out/far of the usual/registered address of the cardholder

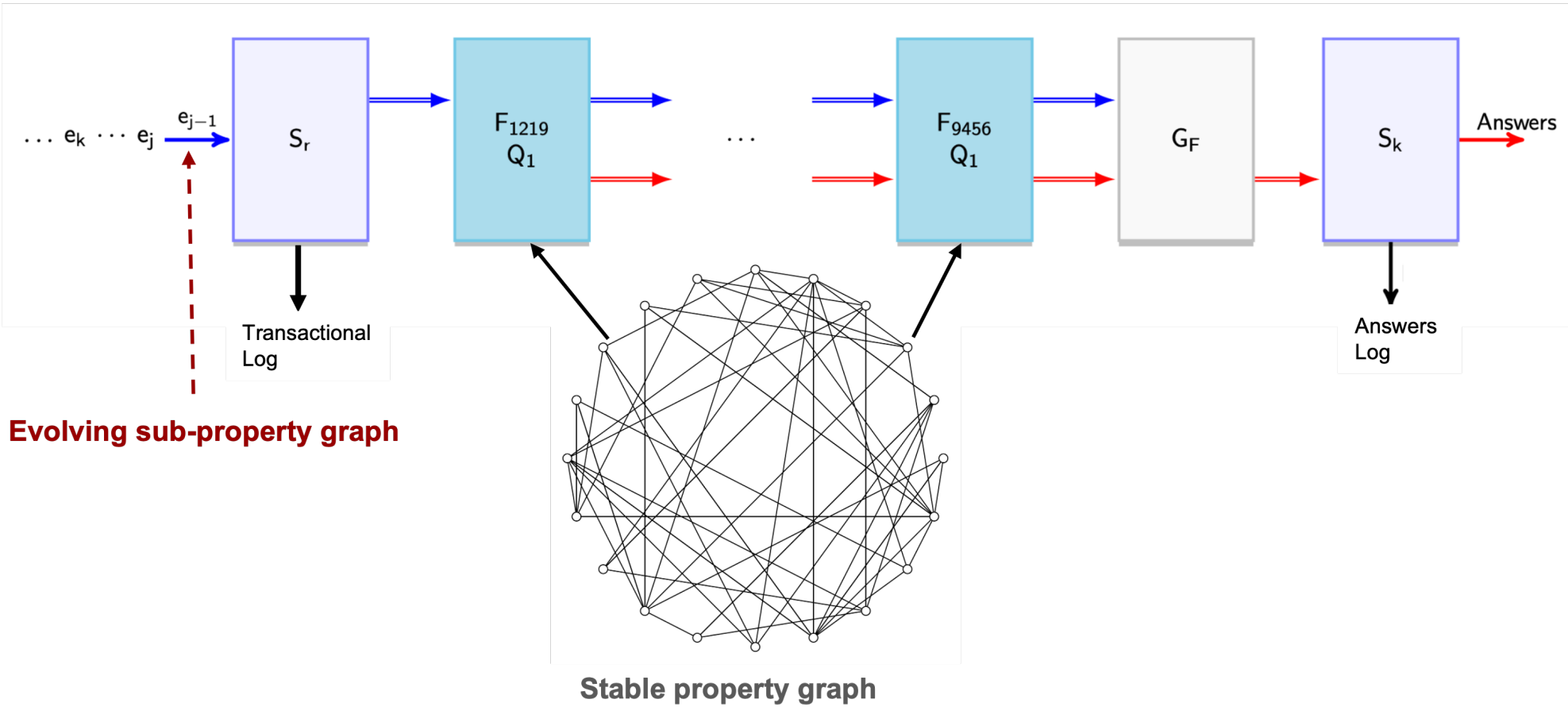
Our proposal:: Continuous Query Evaluation

- Continuous Query Evaluation Engine based on the **Dynamic Computational Approach** - [2].





Continuous Query Engine: DP_{CQE}



Continuous Query Engine: DP_{CQE} - Input Stream

Note: 2 edges per transaction - the *opening* edge and the *closing* edge.

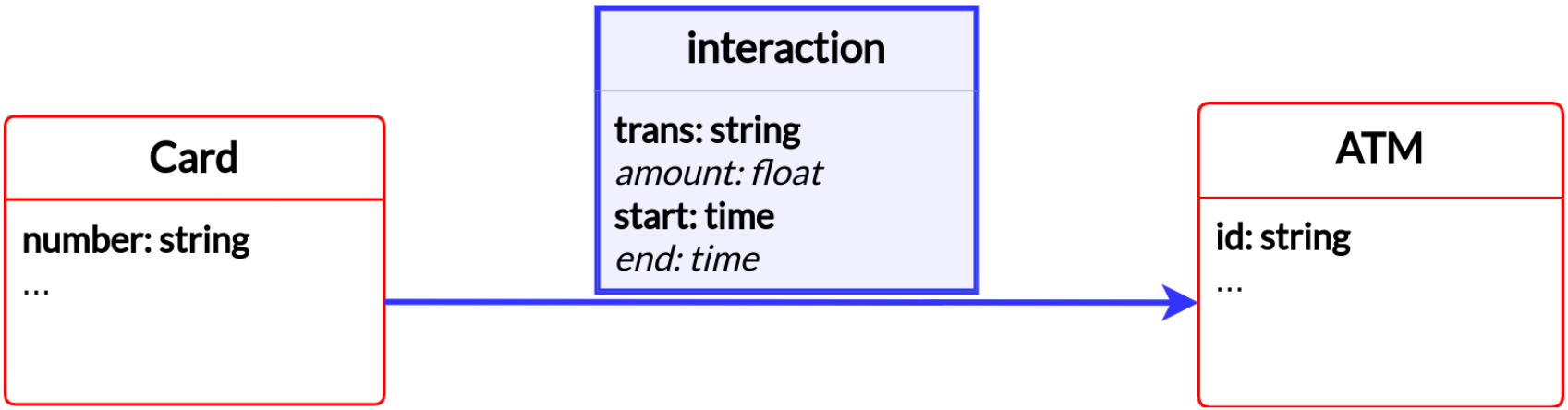


Figure: Opening edge

Continuous Query Engine: DP_{CQE} - Input Stream

Note: 2 edges per transaction - the *opening* edge and the *closing* edge.

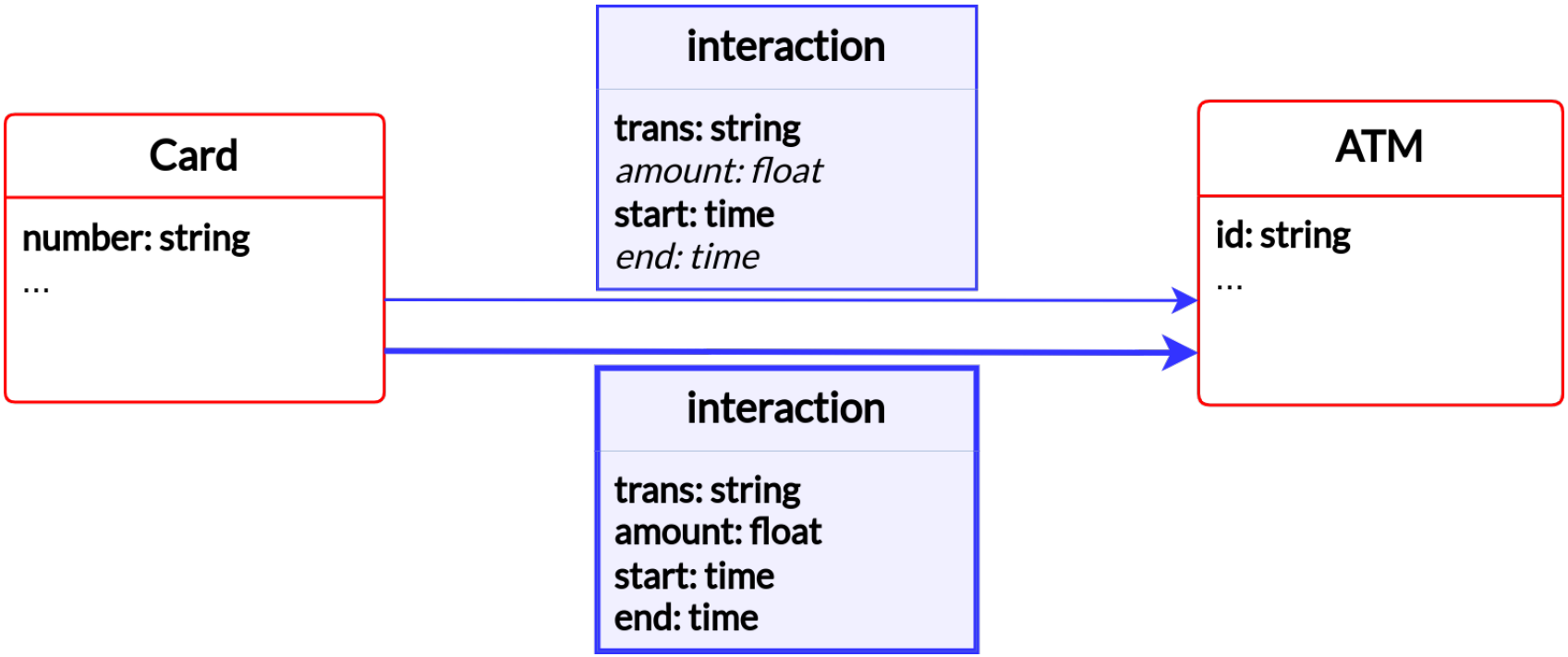
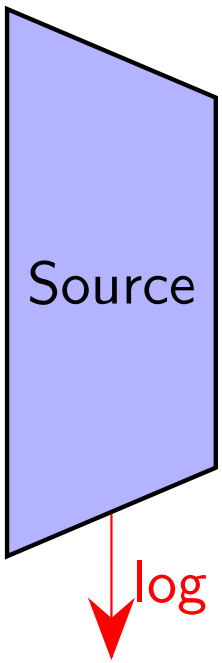


Figure: Closing edge

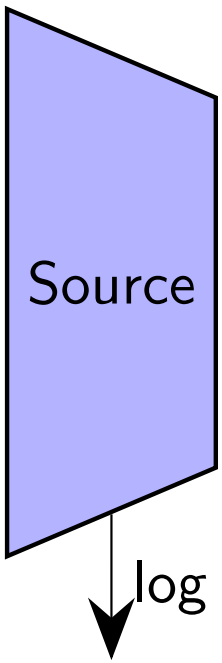
Continuous Query Engine: DP_{CQE} - Stages

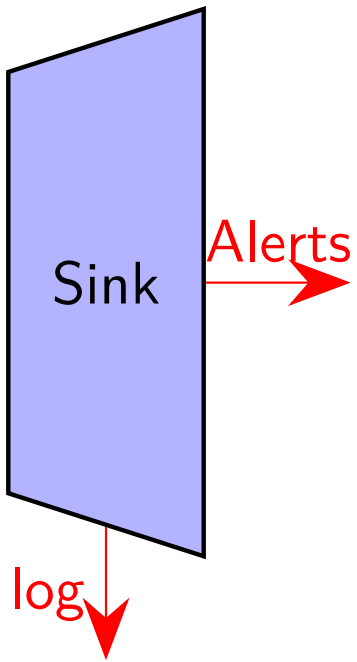
- **Source:** Manages the in-connection with the outside: streaming input / file reading... & general transactions log.



Continuous Query Engine: DP_{CQE} - Stages

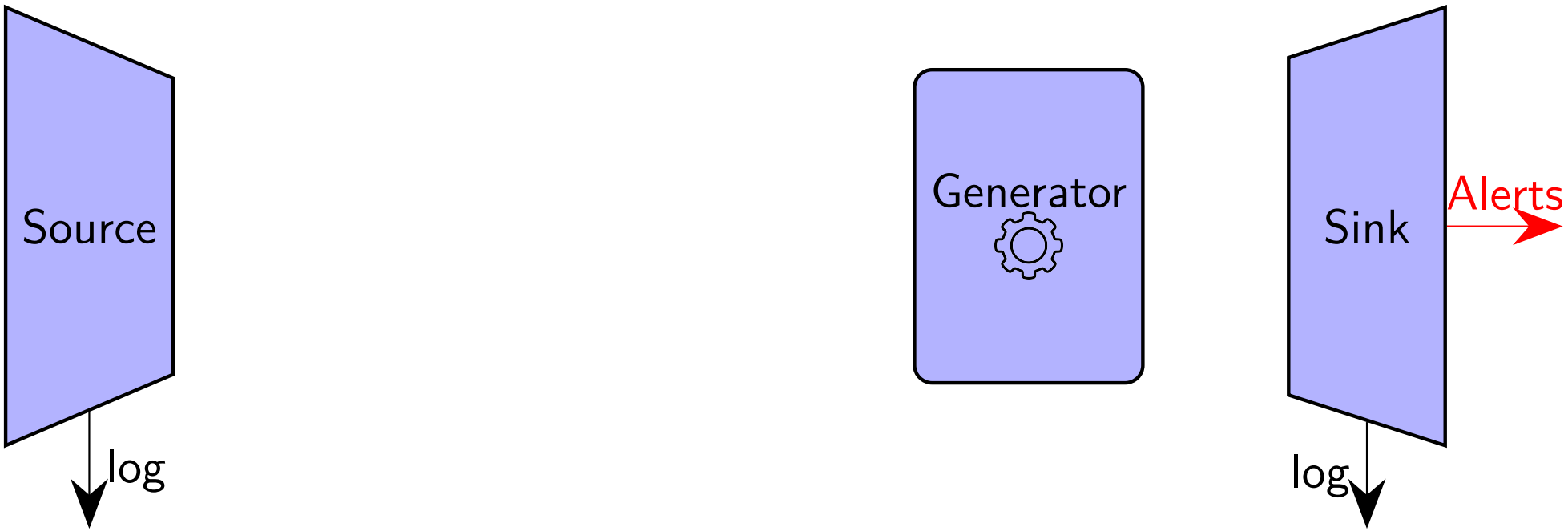
- **Source:** Manages the in-connection with the outside: streaming input / file reading... & general transactions log.
- **Sink:** Manages the out-connection. Outside answers/alerts emission & alert log.





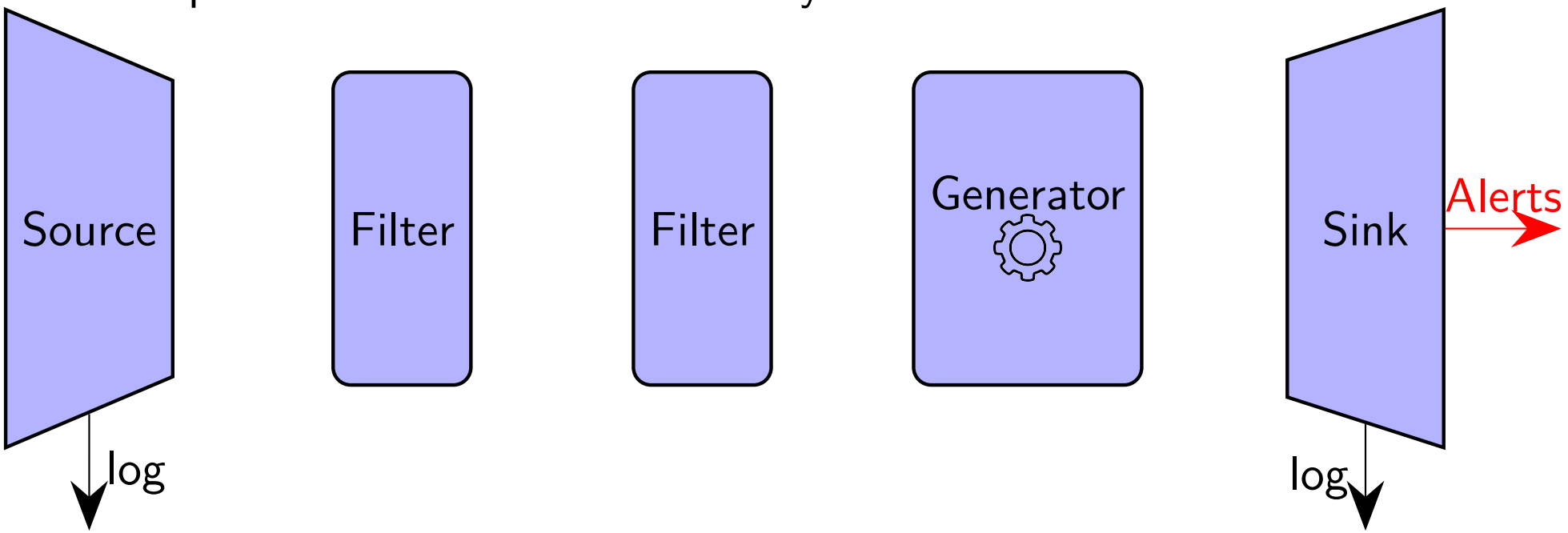
Continuous Query Engine: DP_{CQE} - Stages

- **Source:** Manages the in-connection with the outside: streaming input / file reading... & general transactions log.
- **Sink:** Manages the out-connection. Outside answers/alerts emission & alert log.
- **Generator:** Generation of new filters.



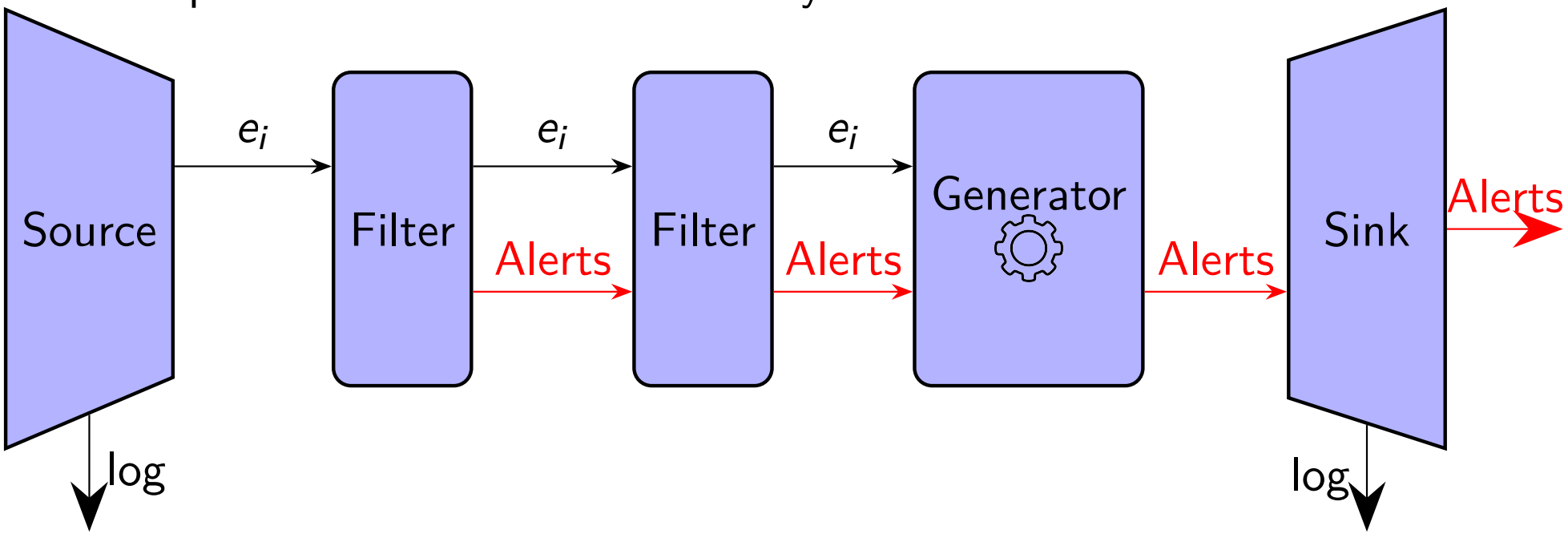
Continuous Query Engine: DP_{CQE} - Stages

- **Source:** Manages the in-connection with the outside: streaming input / file reading... & general transactions log.
- **Sink:** Manages the out-connection. Outside answers/alerts emission & alert log.
- **Generator:** Generation of new filters.
- **Filter:** (Stateful) stage. Anomalous detection tracking process of at least 1 card entity.



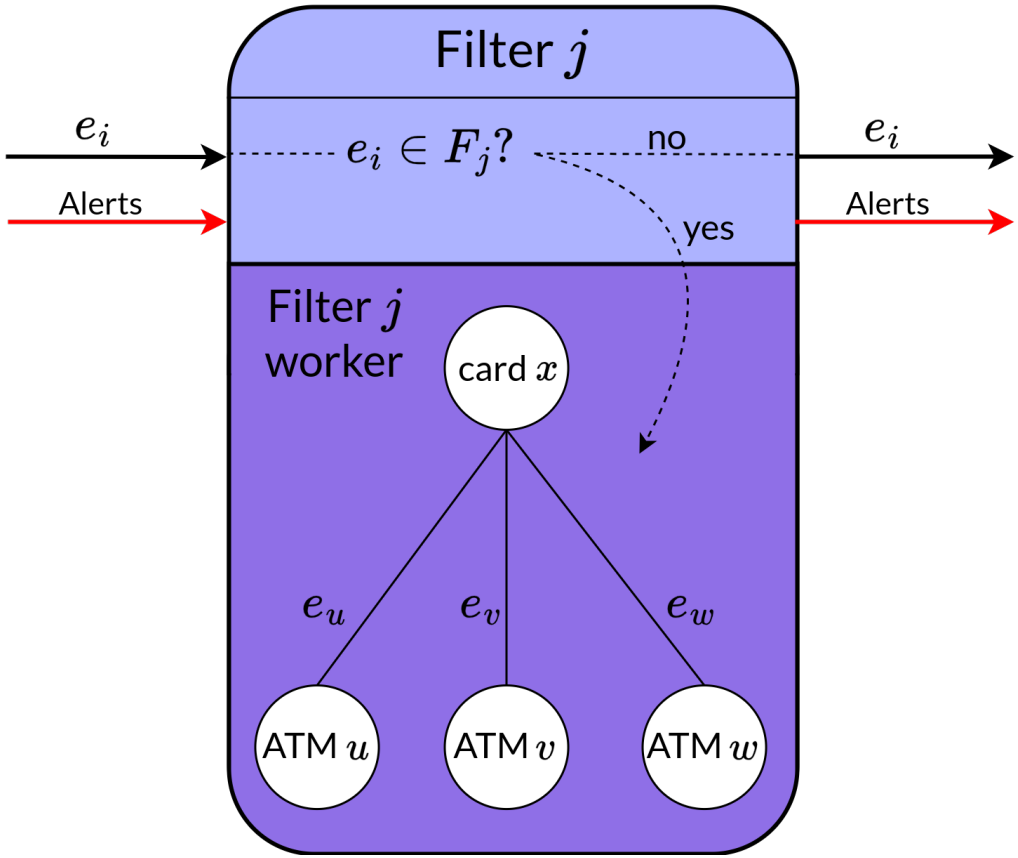
Continuous Query Engine: DP_{CQE} - Stages

- **Source:** Manages the in-connection with the outside: streaming input / file reading... & general transactions log.
- **Sink:** Manages the out-connection. Outside answers/alerts emission & alert log.
- **Generator:** Generation of new filters.
- **Filter:** (Stateful) stage. Anomalous detection tracking process of at least 1 card entity.



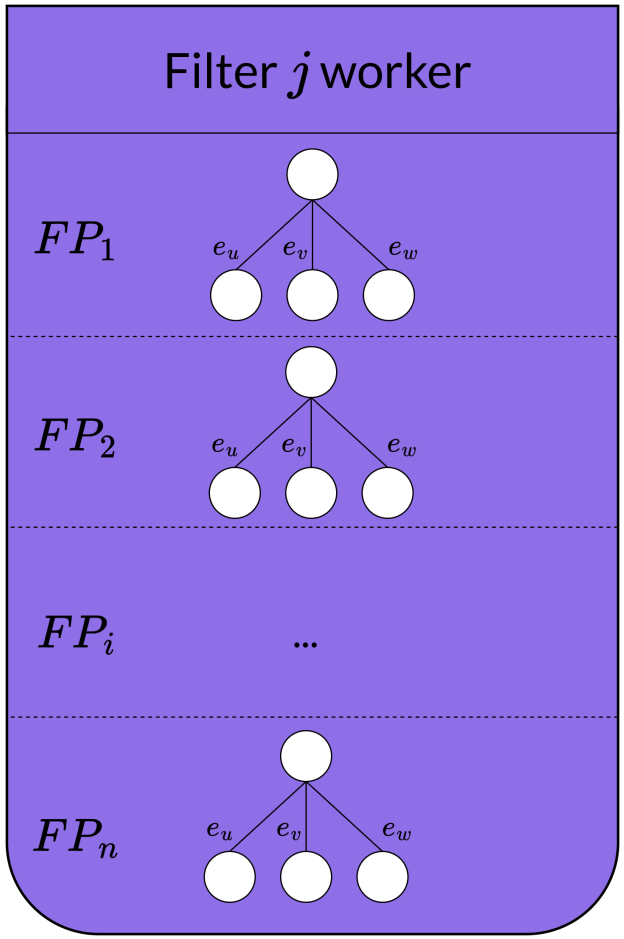
Continuous Query Engine: DP_{CQE} - Filter Stage

- Stores the induced subgraph(s) of the incoming belonging edges.
- Evaluation of (many possible) continuous query pattern(s).
- Emission of alerts in case of matching a query pattern.
- Filter *worker* to avoid bottlenecks - *in parallel*.



Continuous Query Engine: DP_{CQE} - Filter Stage

- **Filter *worker***.
- Simultaneously (many possible) different continuous query patterns: $FP_1, FP_2, ..., FP_n$.



Experimental Design

- Property Graph Generation
- Transaction Generation
- Experiments

Experimental Design:: Property Graph Generation

- **No public real bank dataset found**. Confidential and private nature of bank data.
- **Synthetic PG bank dataset generation tool**:
  - ⇒ Customisable data generation.
  - ⇒ Neo4j Graph Database generation.
  - ⇒ Based on a previously developed synthetic bank database: *Wisabi Bank Dataset*[1].

[1] https://www.kaggle.com/datasets/obinnaiheanachor/wisabi-bank-dataset

Experimental Design:: Interaction Generation

- 1. Generate *regular* transactions: assuring that do not create fraud scenarios.
- 2. Introduce transactions to create anomalous scenarios: taylored injection depending on the anomalous ATM scenario to be tested against.

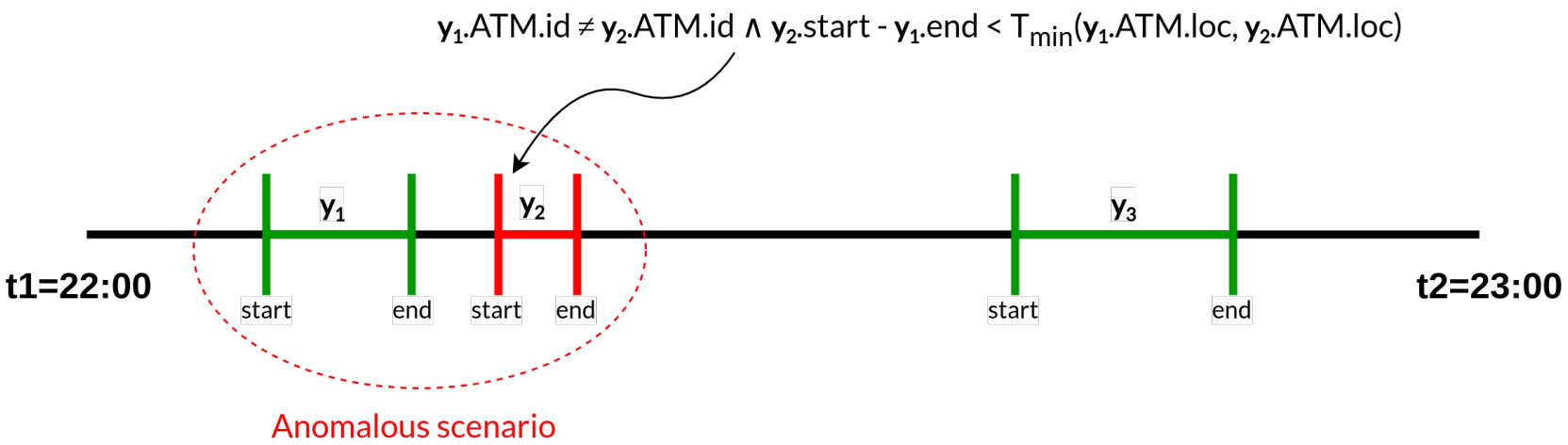


Figure: Creation of anomalous scenario - type 1

Experimental Design:: Interaction Generation

Considerations:

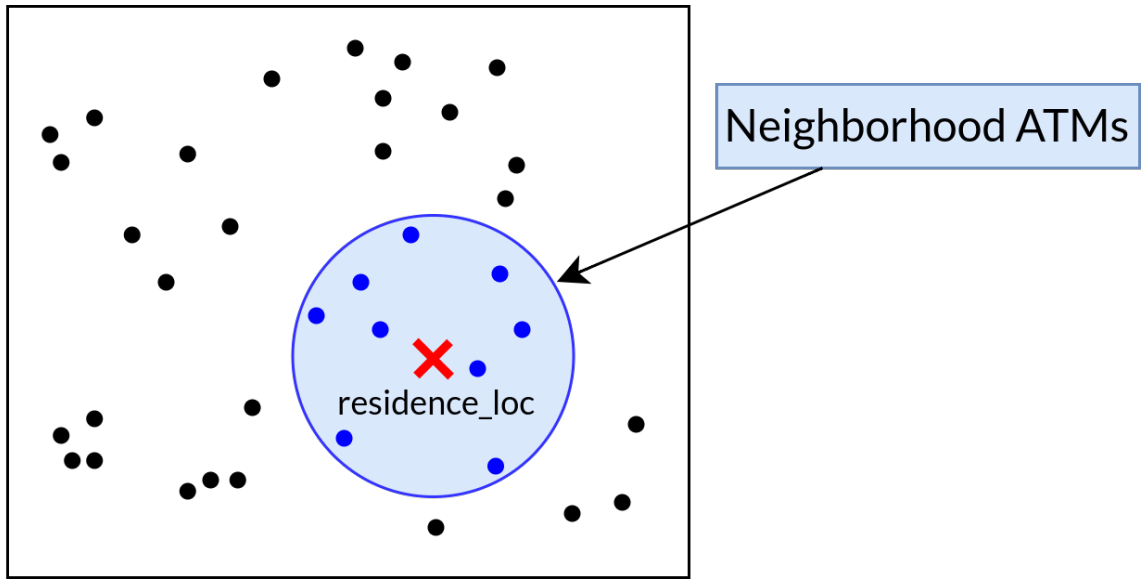
- Particular ATM usage frequency
- ATM selection for the generated transactions of each card
- Distribution of the generated transactions on the decided considered time
-
  - ATM - Pre-selection of closed card ATM subset
  - Time - Uniform distribution
  - Time - Poisson process distribution
  - Random walks
  - ...

Experimental Design:: Regular transactions

Idea: For each card, generate a set of *regular* transactions for a d number of days (starting on a selected *start_date*).

- Limiting the regular transactions of the client to `Neighborhood`. (ATM $\in$ `Neighborhood`).

$$\texttt{Neighborhood} = \{\texttt{ATM} \,|\texttt{dist(ATM, residence\_loc)} \leq \texttt{max\_distance}\}$$



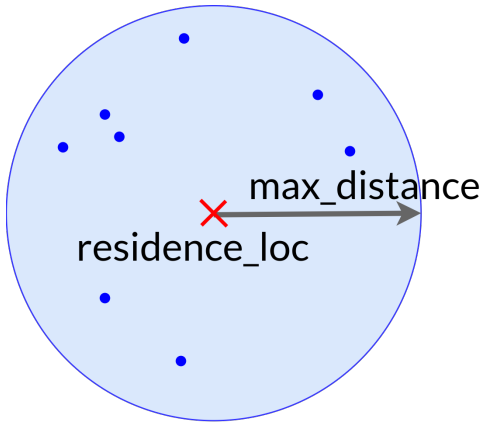
Experimental Design:: Regular transactions

Set a minimum time distance t_{min} between any two consecutive generated transactions of the client.

$$t_{min} = \frac{2 * \texttt{max\_distance}}{\texttt{max\_speed}}$$

- **max_distance**: maximum distance from residence_loc to an ATM $\in$ *Neighborhood*

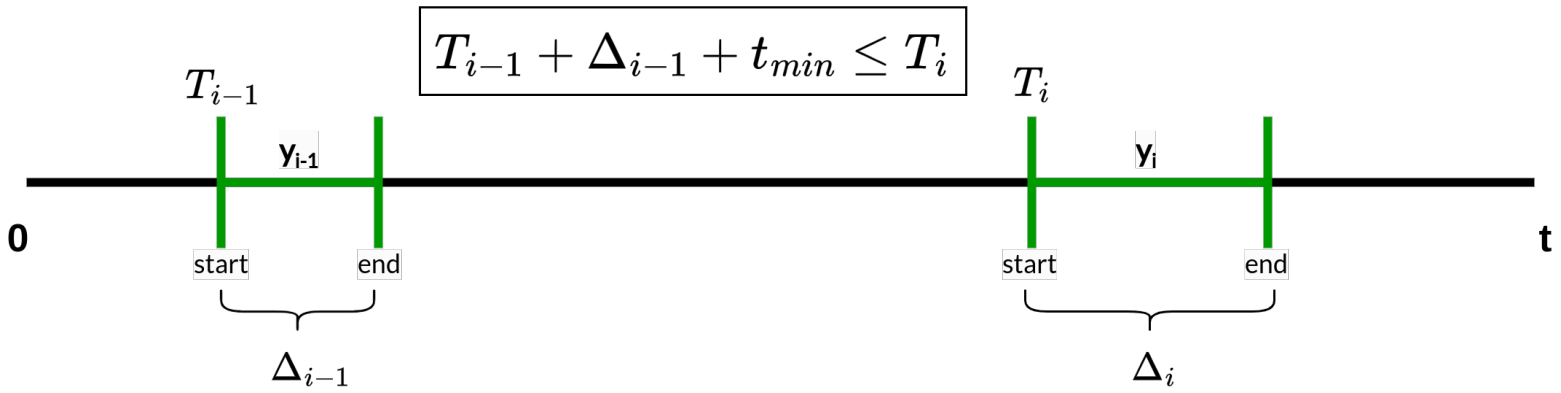
  ($\texttt{Neighborhood} = \{\texttt{ATM} \mid \texttt{dist(ATM, residence\_loc)} \leq \texttt{max\_distance}\}$).
- **max_speed**: maximum speed at which it is possible to travel (by any possible means of transport) between any pair of ATMs $\in$ `Neighborhood`.



Experimental Design:: Regular transactions

Generate `num_tx` transactions for a selected period of time `t`.
Distribution following a Poisson process distribution along $[0, t]$.

- $\lambda =$ `avg_tx` on a day for the client.
- `t = 24h`.
- Inter-arrival times are distributed following an exponential distribution: $T \sim \text{Exp}(\lambda)$ with the constraint that: $T_{i-1} + \Delta_{max} + t_{min} \leq T_i, \forall T_i$
  - T_i: starting time of the i-th transaction.
  - Δ_{max}: considered maximum duration of a transaction.
  - t_{min}: minimum calculated time distance between any 2 consecutive transactions of the client.



Experimental Design:: Anomalous scenarios

- Injection of transactions that produce *anomalous* scenarios.
- E.g. for the considered fraud scenario.

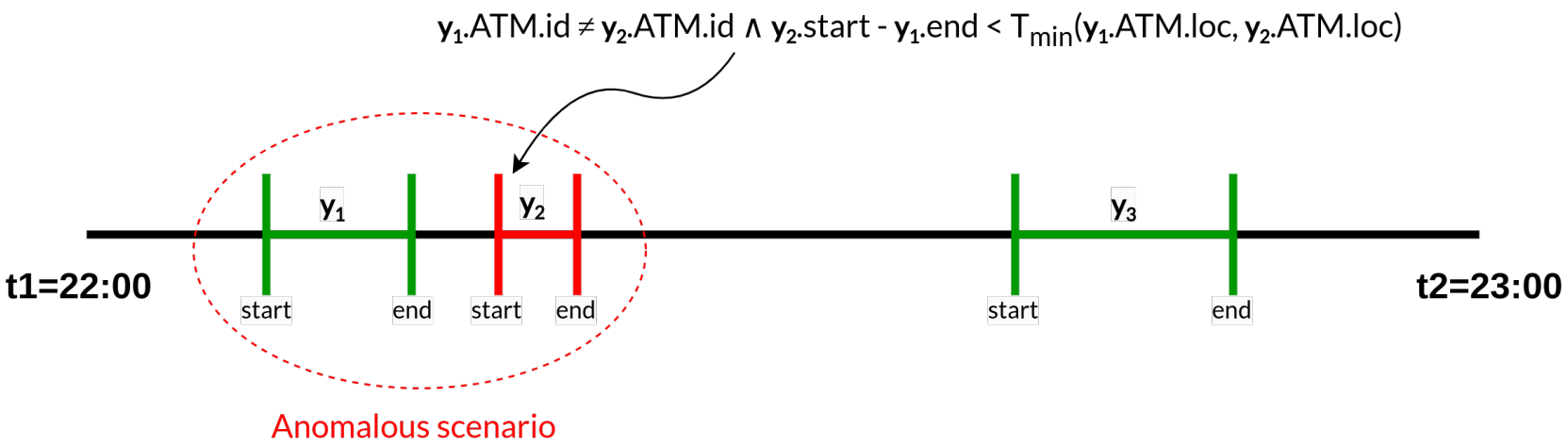
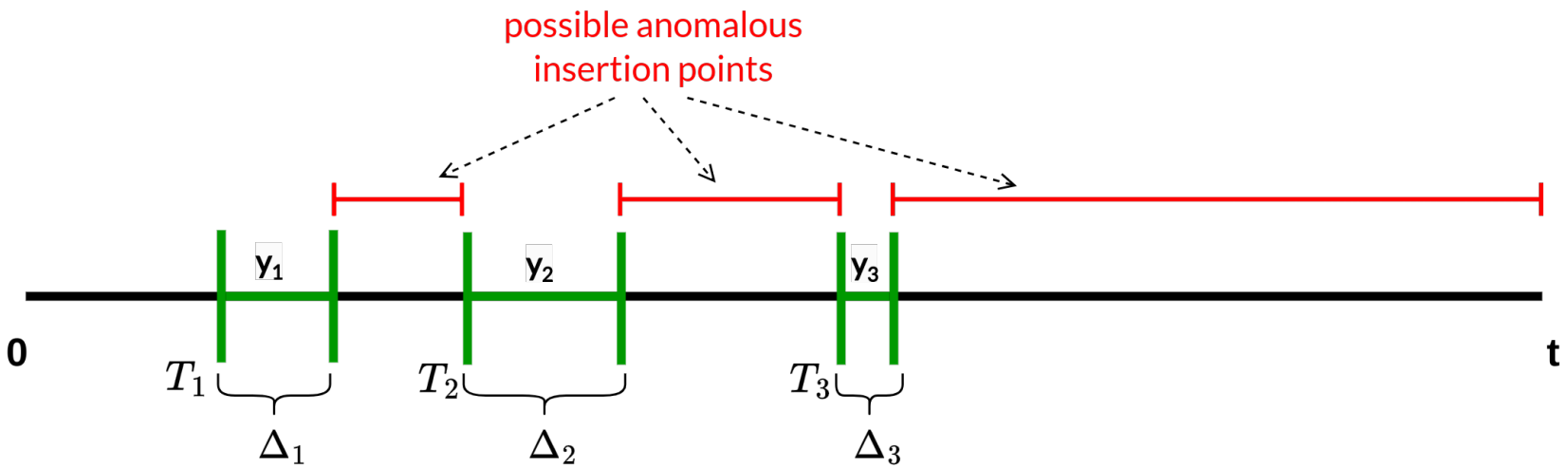


Figure: Creation of anomalous scenario - type 1

Experimental Design:: Anomalous scenarios (I)

- `ratio` $\in [0, 1]$ s.t. `ratio` $*$ `num_tx` is the number of anomalous scenarios created for a card in the considered time interval $[0, t]$.
- No overlapping of the transaction introduced a with the regular ones $i, i+1$: $T_i + \Delta_i < T_a < T_a + \Delta_a < T_{i+1}$



Final Remarks

- Single temporal window support. One-time based sliding window.
- *Golang* implementation. Suitable for thread management (concurrency).
- Stream input provided by file reading. Adaptable to be obtained through *Apache Kafka* or other handling real-time data feeds systems.
- Extensible beyond ATM transactions to online card transactions - *PoS (Point-of-Sale)*, *CNP (Card-Not-Present)* frauds.

Ongoing & Future Work

- Refine the preliminary implementation.
- Experimental study. *Diefficiency* metrics to measure the progressive delivery of results. [1]
- Characterization of different transactions anomalies of interest.
- Testing by size in the number of transactions and anomalies detected.
- Establish the window policy.

Thank you!