Continuous Query Engine to Detect Anomalous Electronic Transactions Patterns Using Bank Cards

Fernando Martín Canfrán

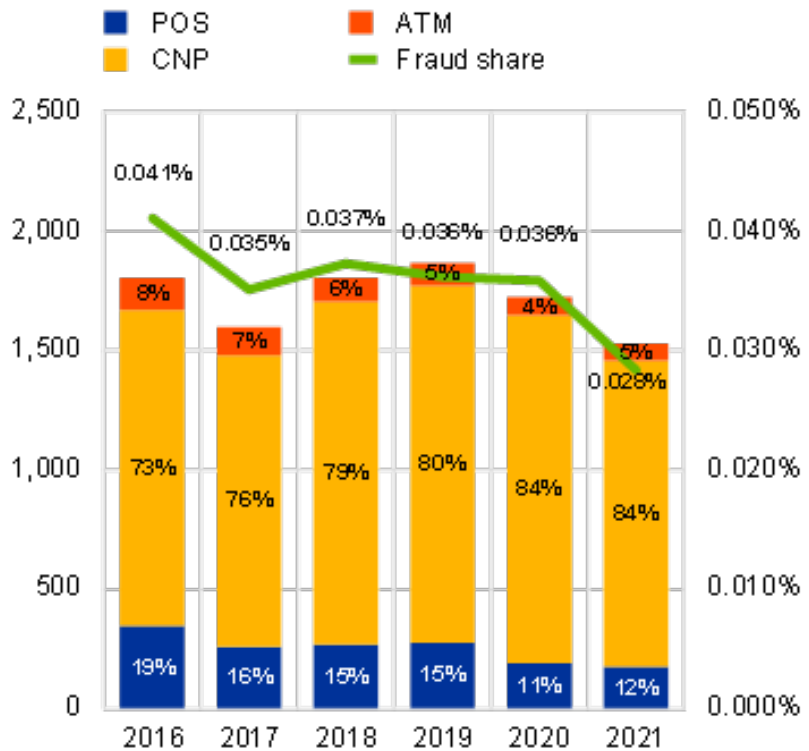
Supervisor: Edelmira Pasarella Sánchez
Co-Supervisor: Amalia Duch Brown

Facultat d'Informàtica de Barcelona
Universitat Politècnica de Catalunya

Overview

Motivation: Card Fraud Trends

- The total value of transactions using cards issued in SEPA (Single Euro Payments Area) amounted to €5.40 trillion in 2021, of which €1.53 billion (0.028%) was fraudulent[1].



- Value of ATM fraud $\sim$ €74M; POS terminals €177M (2021).

[1] Report on card fraud in 2020 and 2021, BCE [2]

Motivation: ATM Fraud Detection - State of the Art

- **Classical treatment**: by consulting log files because of the complaint of customers when detecting by themselves some weird movement in their accounts.

- **ML systems**: delayed detection based on prediction. Need training processes and big volume of data. Rahman et al. survey (2019) [4].

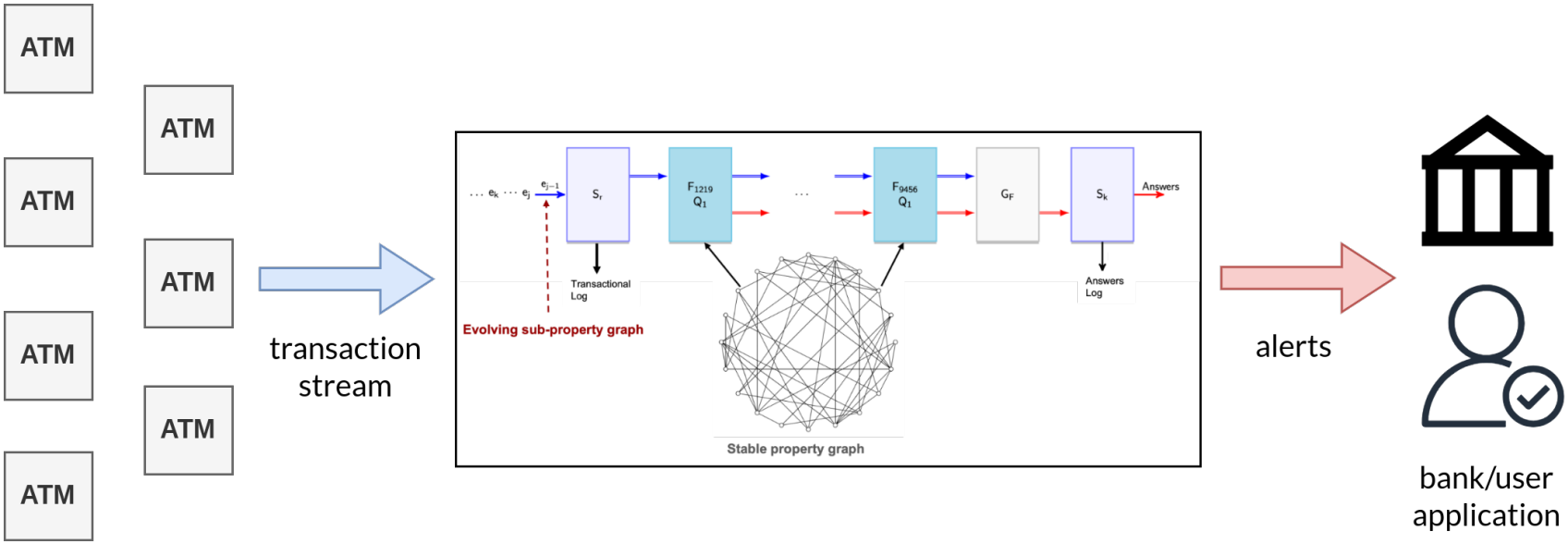
$\rightarrow$ To our knowledge, most of the works are based on artificial intelligence techniques. No clear characterization of the problem.

Contributions

- A general technique for addressing the problem of continuous query evaluation against an evolving graph database.
- A characterization of some possible fraud (graph) patterns.
- A continuous query engine (DP_{ATM}) to detect abnormal or suspicious ATM transactions in real-time, with 100% accuracy.
- A synthetic dataset bank and transaction stream generator tool (lack of real-world data).

Proposal

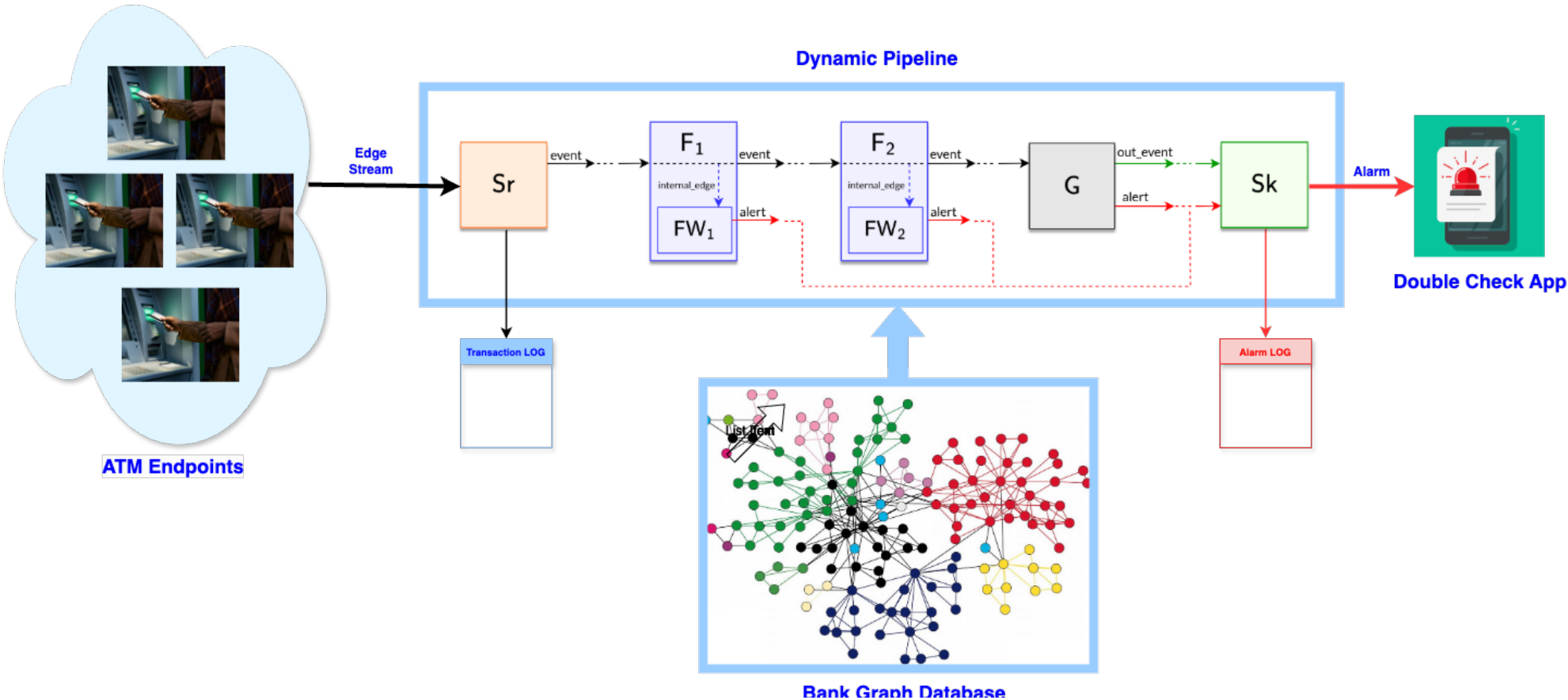
⇒ Continuous Query Engine for the detection of anomalous ATM card fraud patterns in real-time: DP_{ATM}.



Proposal

1. Architecture of the System
2. Data Model
3. Definition of Anomalous Patterns of Transactions
4. Query Model
5. Continuous Query Engine - DP_{ATM}

Proposal: Architecture of the System



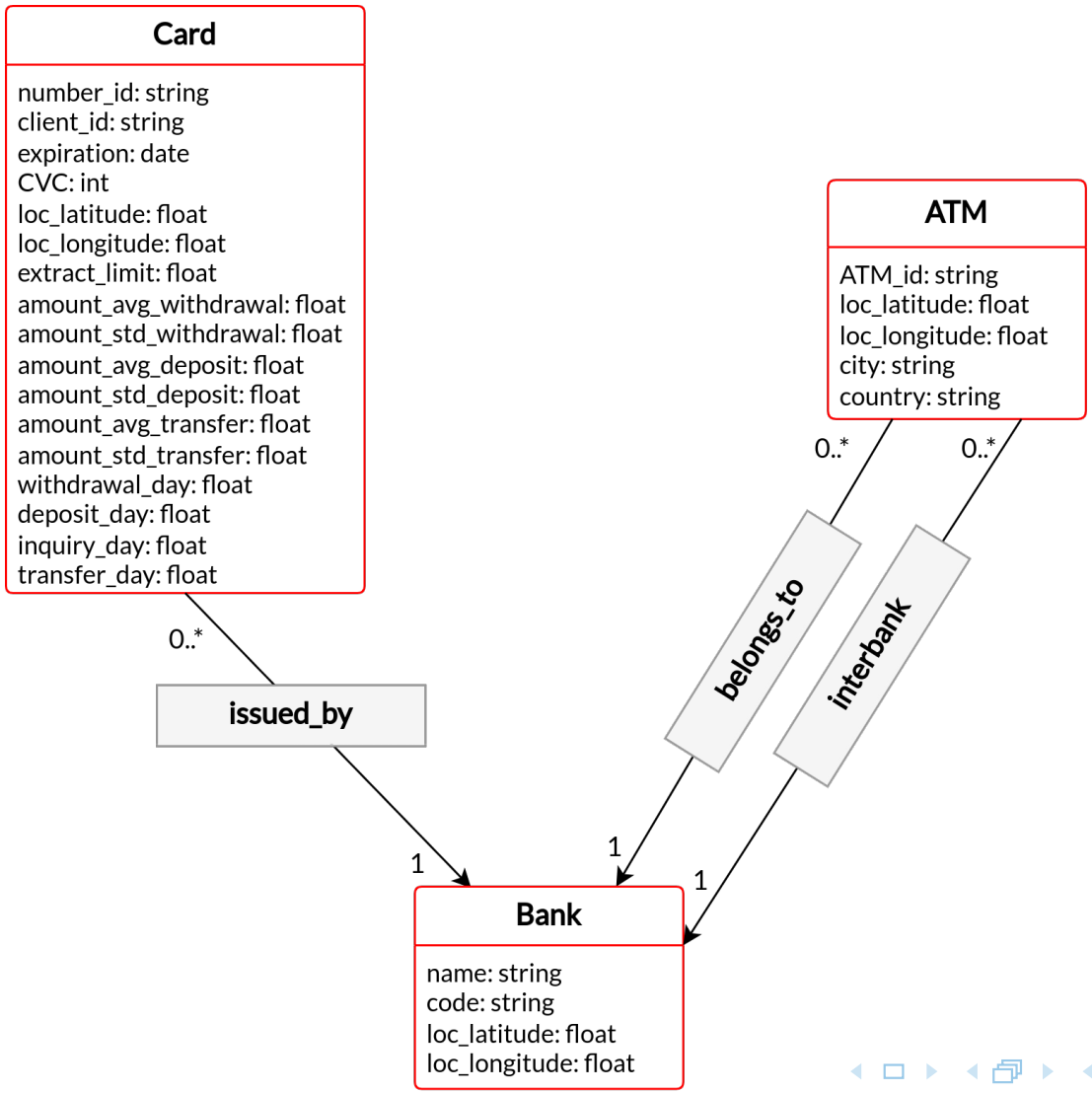
Proposal: Data Model

- Nature of our data: ATM transactions on a bank system.
  - $\Rightarrow$ Evolving/Dynamic data sources. Transactions occur continuously and are not (necessarily) bounded.
- ***Continuously evolving database*** - data can be stable and volatile.
  - $\Rightarrow$ Suitable data model: Graph Data Model - Property Graph (PG).
- **Continuously evolving property data graph**:
  - $\Rightarrow$ Stable PG: Central bank database - *persistent relations of standard graph databases.*
  - $\Rightarrow$ Volatile PG: Transactions - *non-persistent relations. Edges arriving in the data stream.*

Proposal: Data Model

⇒ Stable PG

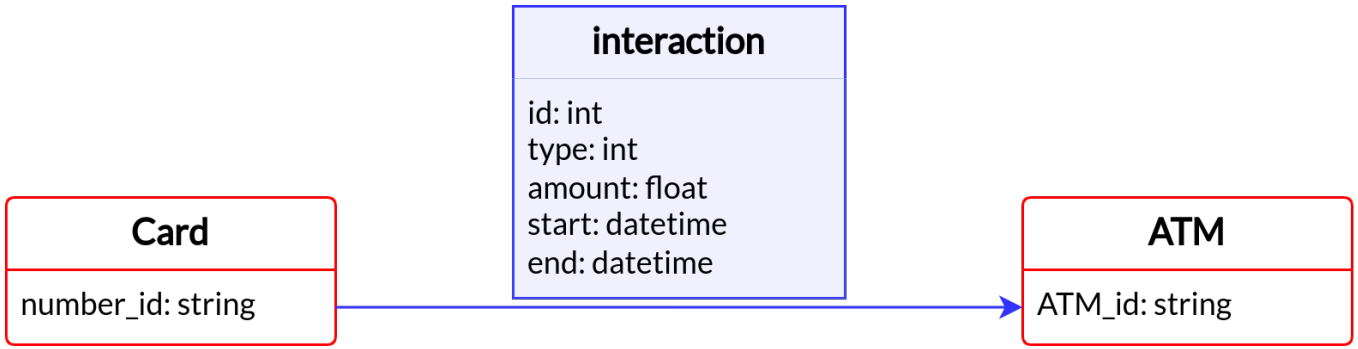
- Models the data a bank typically gathers on cards, ATMs...



Proposal: Data Model

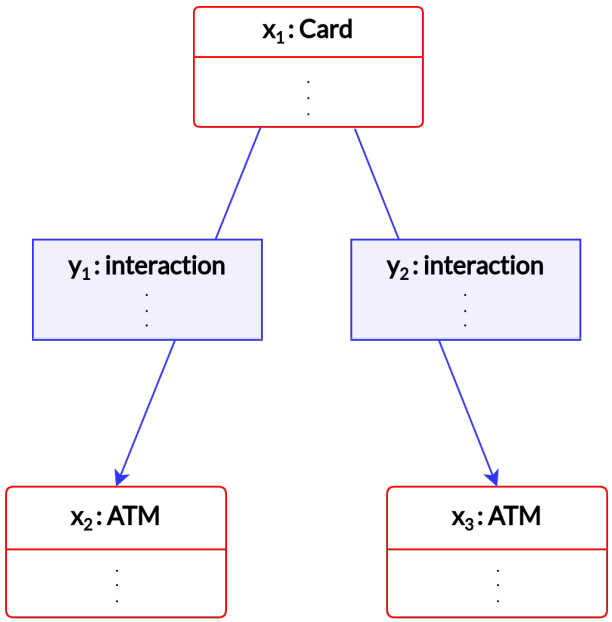
⇒ Volatile PG

- 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.



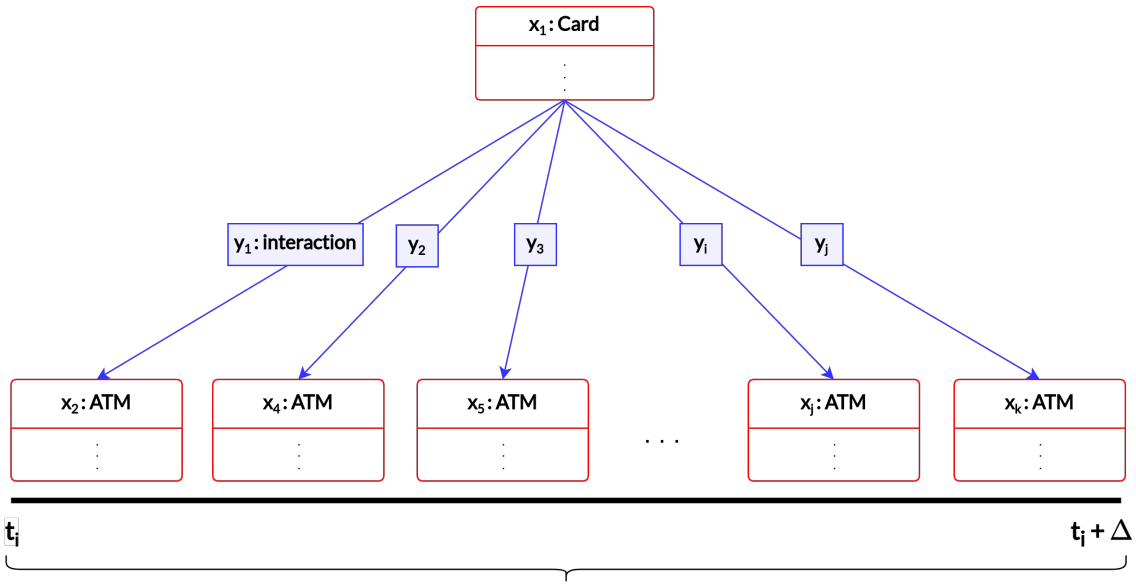
Proposal: Definition of Anomalous Patterns of Transactions

- Continuous queries are characterized as (constrained) graph patterns



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

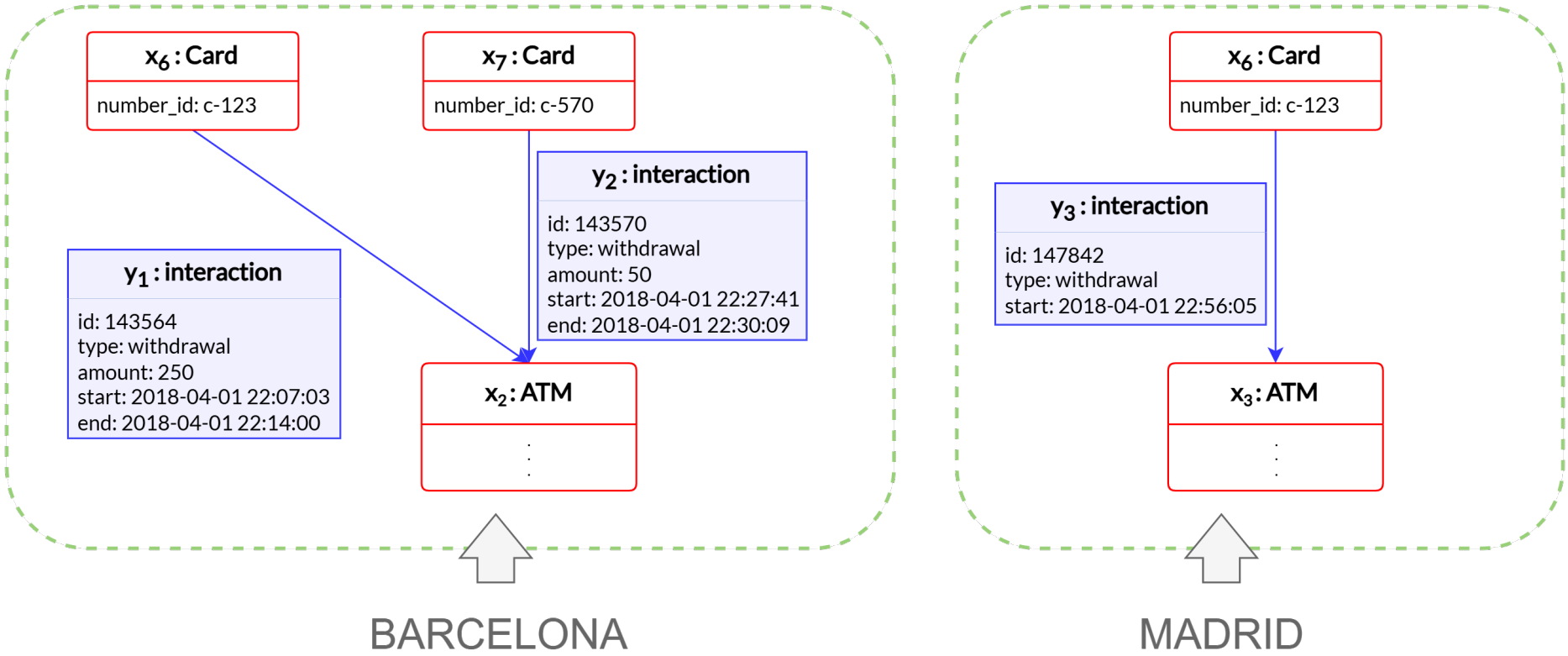
Card cloning



- 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

Proposal: Definition of Anomalous Patterns of Transactions



Card cloning characterization - an example

Proposal: Query Model

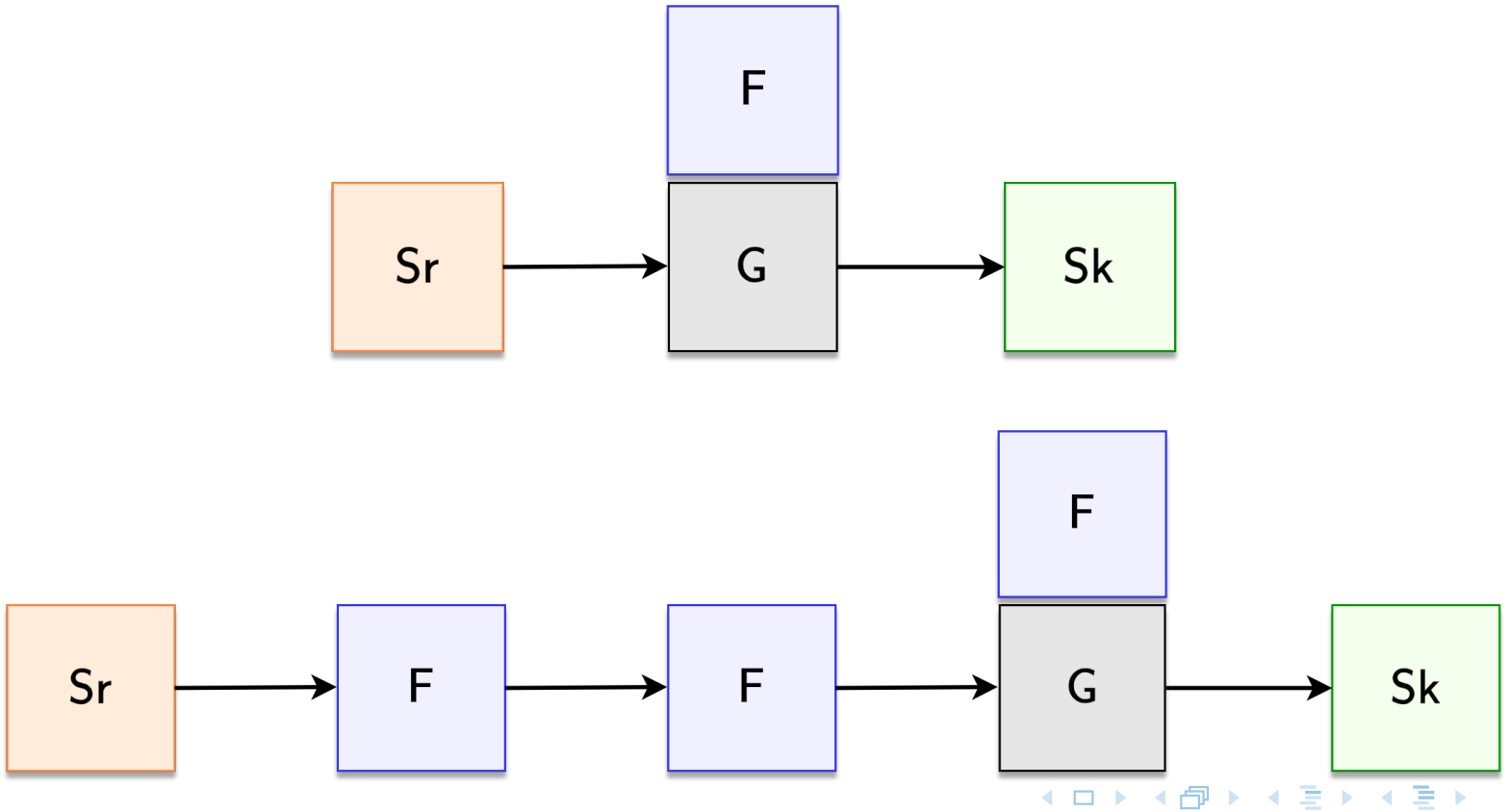
- *Continuous query model*: Fixed queries evaluated over data streams.
- Progressive query evaluation process. Based on:

  Graph pattern matching + Satisfiability of constraints

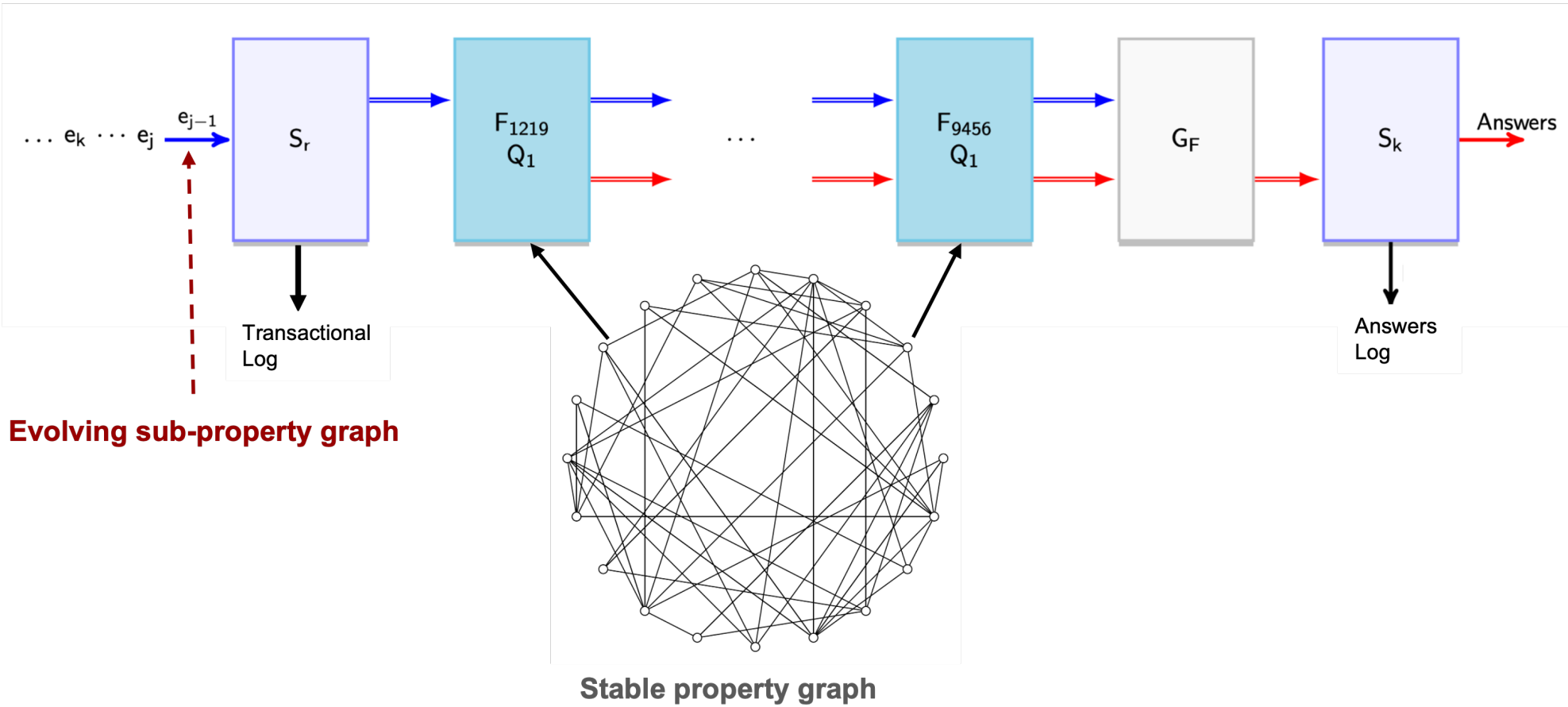
- Graph pattern matching in the volatile subgraph.
- Satisfiability of constraints over the properties: possibly querying the stable graph database (*retrieve some additional info*).
- Different kinds of anomalous patterns:
  - *Card cloning* characterization
  - *Lost-and-stolen* card characterization
  - Anomalous location usage
  - Anomalous number of operations
  - Anomalous high expenses

Proposal: Query Model

- Continuous Query Evaluation Engine based on the **Dynamic Computational Approach** - Pasarella [3]. Stream processing computational model.
- Useful to address the problem of evaluating continuous queries over *continuously evolving PGs* (Royo-Sales [5])



Continuous Query Engine: DP_{ATM}



DP_{ATM} - Input Stream

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

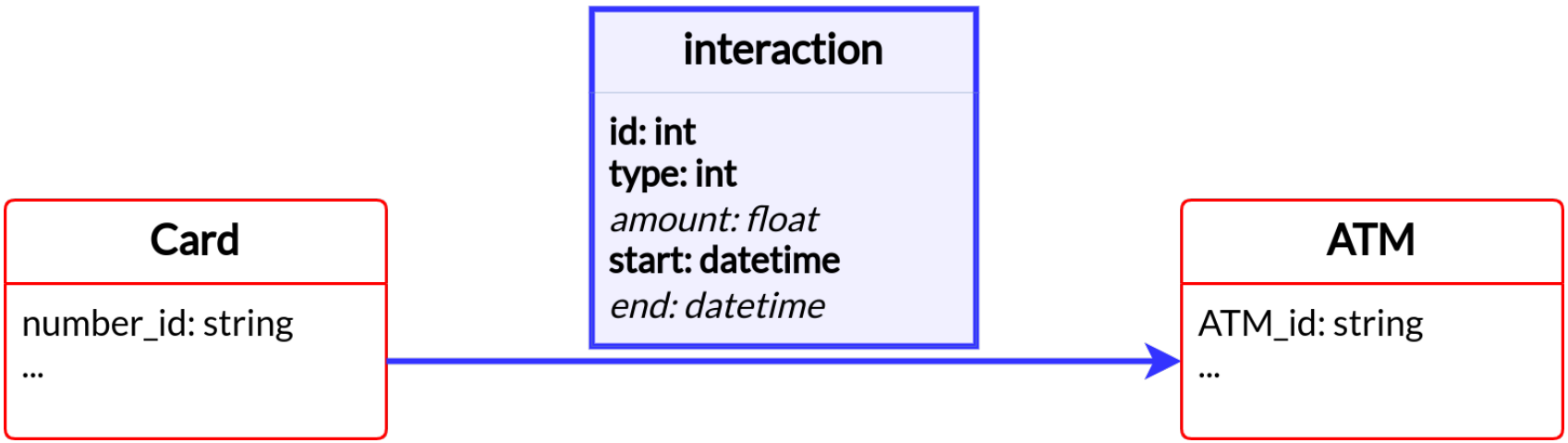


Figure: Opening edge

DP_{ATM} - Input Stream

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

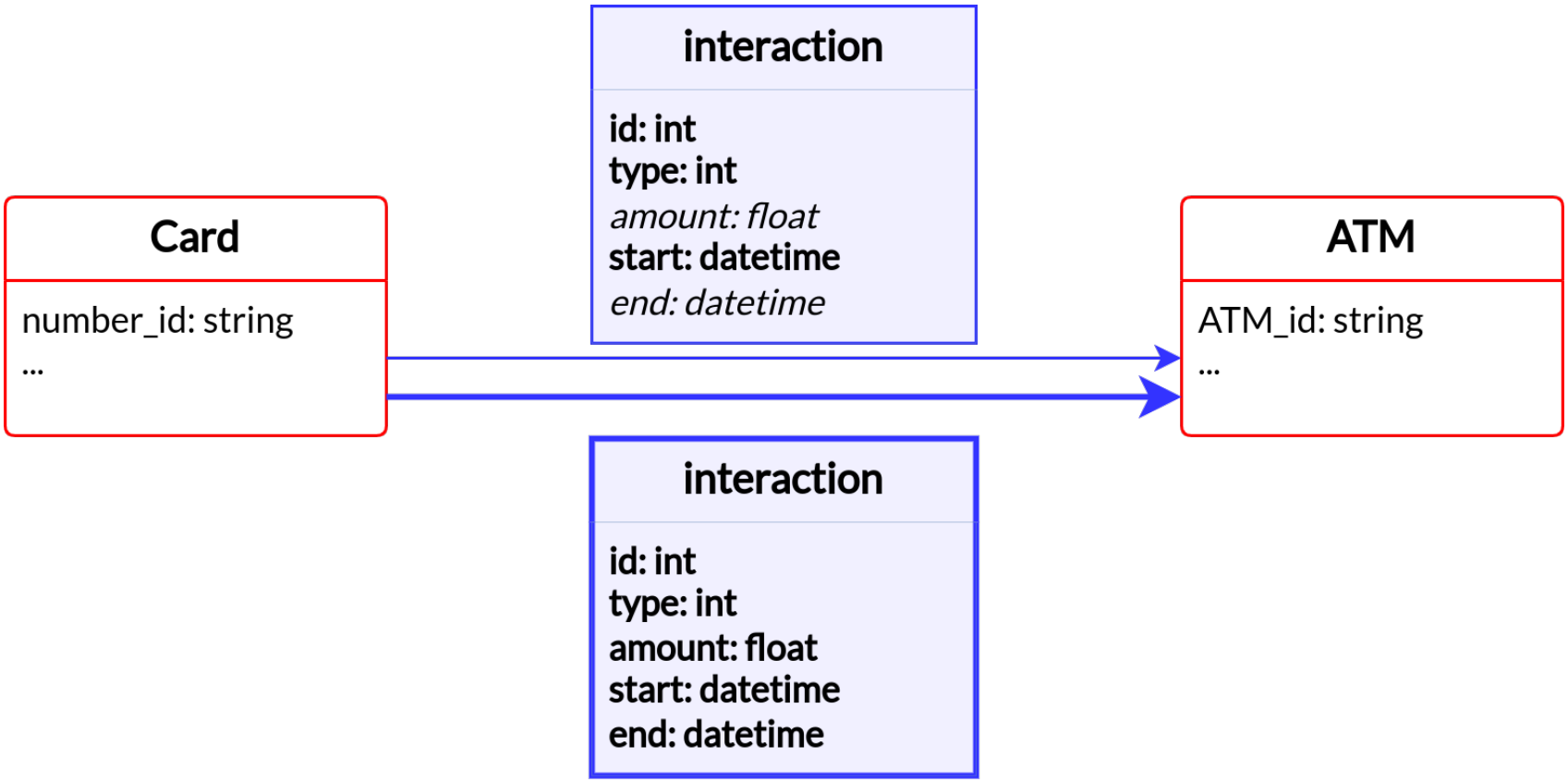
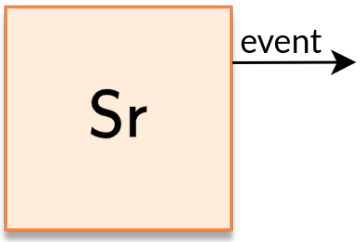


Figure: Closing edge

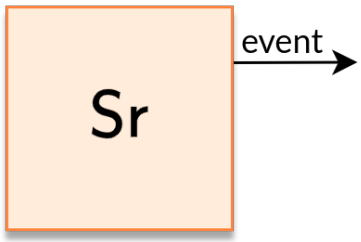
DP_{ATM} - Stages

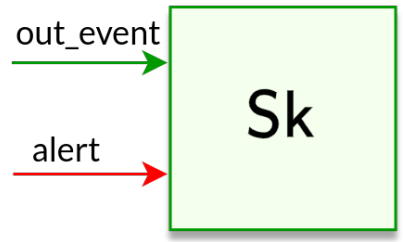
- **Source:** Manages the in-connection with the outside: streaming input / file reading... & general transactions log.



DP_{ATM} - Stages

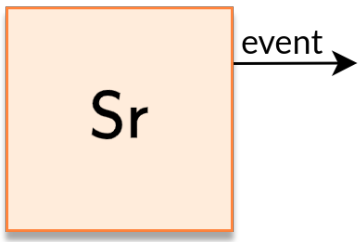
- **Source:** Manages the in-connection with the outside: streaming input / file reading... & general transactions log.
- **Sink:** Manages the out-connection. Outside answers/alerts emission & alert log.

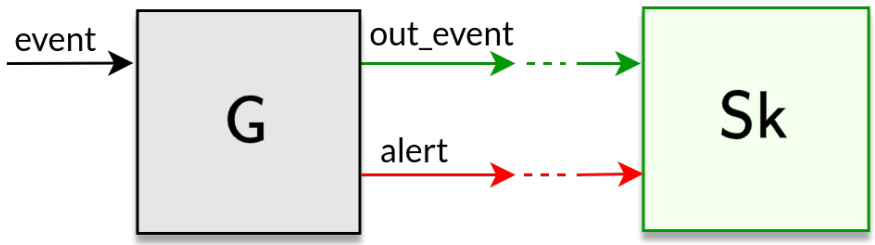




DP_{ATM} - Stages

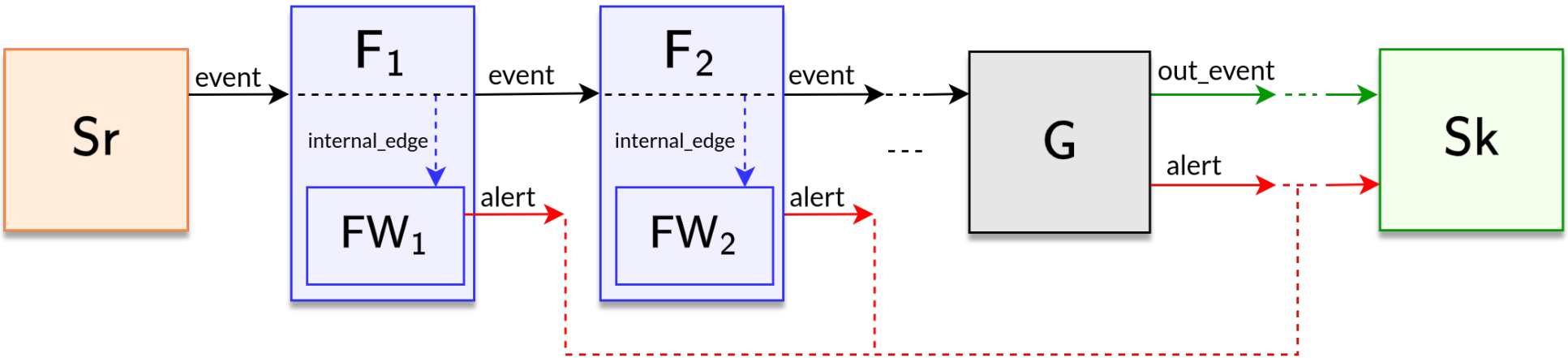
- **Source:** Manages the in-connection with the outside: streaming input / file reading... & general transactions log.
- **Sink:** Manages the out-connection. Outside answers/alerts emission & alert log.
- **Generator:** Generation of new filters.





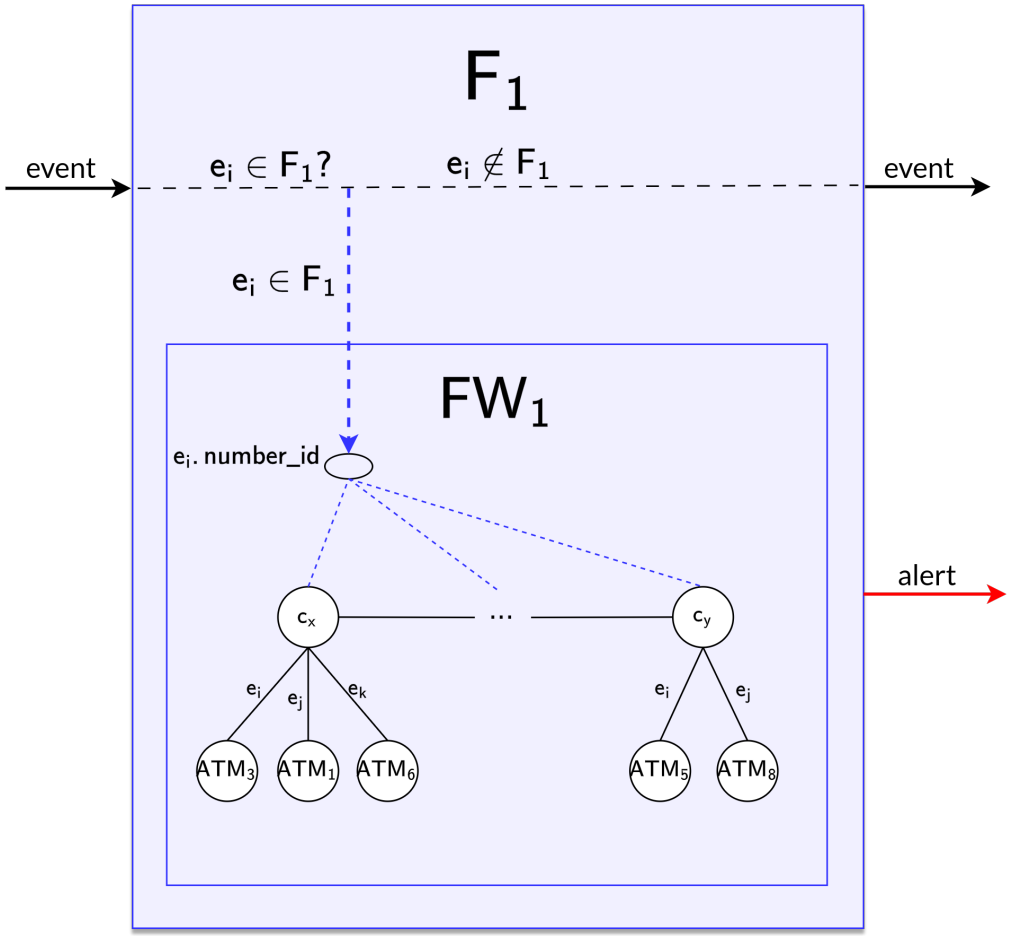
DP_{ATM} - Stages

- **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 *maxFilterSize* cards.



DP_{ATM} - Filter Stage

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



DP_{ATM} - Filter Stage

- Card cloning fraud pattern algorithmic evaluation

Algorithm `checkFraud`(S_c, e_{new})

Require: S_c is a non-empty subgraph of interaction edges of card c, e_{new} is the `Edge` related to the new incoming opening interaction `EdgeStart` of card c

1: $e_{last} \leftarrow S_c[|S_c| - 1]$ ▷ Retrieve last edge from S_c
2: **if** e_{last}.`Tx_end` is empty **then**
3: `LOG: Warning: A tx (`e_{new}`) starts before the previous (`e_{last}`) ends!` **return**
4: **end if**
5: **if** e_{last}.`ATM_id` $\neq$ e_{new}.`ATM_id` **then**
6: `t_min` $\leftarrow$ obtainTmin(e_{last}, e_{new})
7: `t_diff` $\leftarrow$ e_{new}.`Tx_start` $-$ e_{last}.`Tx_end`
8: **if** `t_diff` $<$ `t_min` **then**
9: emitAlert(e_{last}, e_{new})
10: **end if**
11: **end if**

Experiments

- Datasets
- Design of Experiments
- Results

Experiments: Datasets

- **No public real bank dataset found**. Confidential and private nature of bank data.
- **Synthetic bank dataset generation tool**: (i) Bank database and (ii) streams of transactions generators.
  - $\Rightarrow$ Customisable data generation.
  - $\Rightarrow$ Based on a previously developed synthetic bank database *Wisabi Bank Dataset* (publicly available). Used as a reference for the geographical distribution of ATM locations and card/client transaction *behavior*.

Experiments: Datasets

- **(i) Bank database generator**. `bankDataGenerator.py`: creates a dataset of n ATMs and m Cards.
- **(ii) Transaction stream generator**. `txGenerator.py`: parametrizable, with `ratio` $\in [0, 1]$ of anomalous tx..
  - 1. *Regular* transactions: avoiding creation of fraud scenarios. Based on gathered customer *behavior* from *Wisabi DS*.
  - 2. Injection of transactions that create anomalous scenarios: taylored injection depending on the ATM fraud considered.

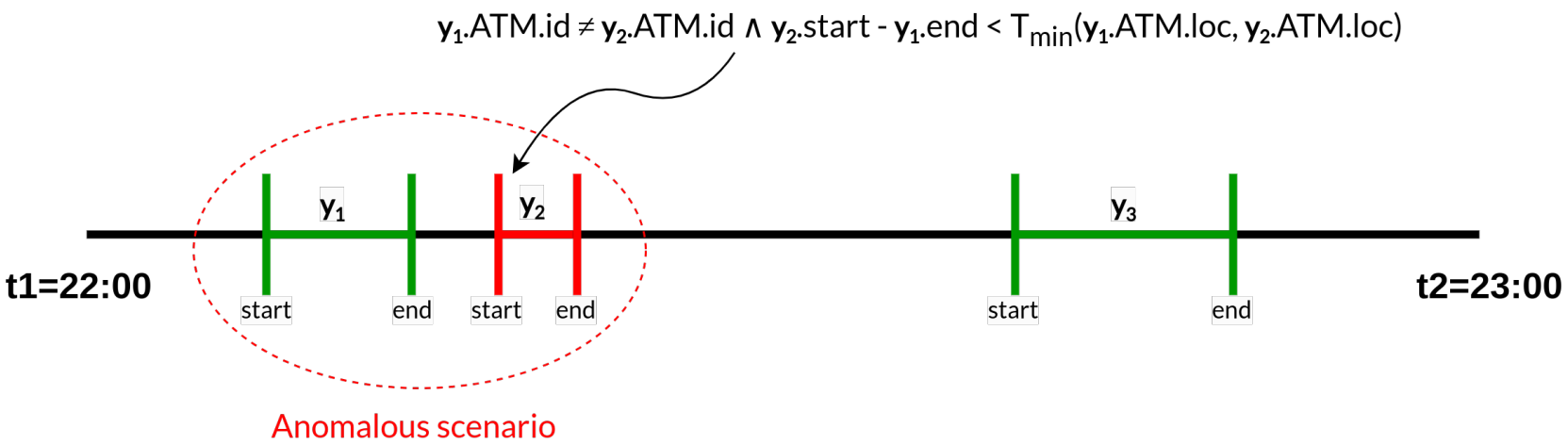


Figure: Creation of anomalous scenario I - Card cloning

Experiments: Design of Experiments

- **E0: Evaluation in a real-world stress scenario**.
  - → Impractical simulation due to large time needed.
  - → Meaningless insights due to insufficient stress.

  Therefore, two options were considered:

  (i) Scaling of the transaction stream to smaller-sized time intervals streams.

  (ii) Consider larger bank database system.

- **E1: Evaluation in a high-load stress scenario** - highest possible transaction stream frequency so to identify the system limits.

Experiments: Considered Datasets

- Bank databases g:

Name	$\lvert$**Card**$\rvert$	$\lvert$**ATM**$\rvert$	$\lvert$**ATM**$_{\textbf{Internal}}\rvert$	$\lvert$**ATM**$_{\textbf{External}}\rvert$
GDB_A	2000	50	40	10
GDB_B	500000	1000	900	100

- Synthetic streams $s(k, p)$, for each g:

Bank DB (g)	$\lvert$**Days**$\rvert$ (k)	**Anomalous Ratio** (p)	**Stream Size** $\lvert$**Tx**$\rvert$
GDB_A	30	0.02 (2%)	39959
GDB_A	60	0.02 (2%)	80744
GDB_A	120	0.02 (2%)	160750
GDB_B	7	0.03 (3%)	2428286
GDB_B	15	0.03 (3%)	4856573

Experiments: E1 - Evaluation in a high-load stress scenario

Objectives:

- Comparison to sequential baseline.
- Analysis of the number of filters configuration.
- Analysis of the behavior and suitability of the DP_{ATM} as a real-time engine.

Experimental variations:

We performed experiments $\Sigma = (g, s(k, p), f, c)$ with the following combinations of number of filters and cores

- Number of filters f:
  - $g = \text{GDB}_{\text{A}}$ $f = 1, 2, 5, 10, 20, 40, 100, 200, 500, 1000, 2000$.
  - $g = \text{GDB}_{\text{B}}$ $f = 1, 5, 10, 100, 250, 500, 1000, 2000, 5000, 10000$.
- Number of cores: $c = 1, 2, 4, 8, 16$.

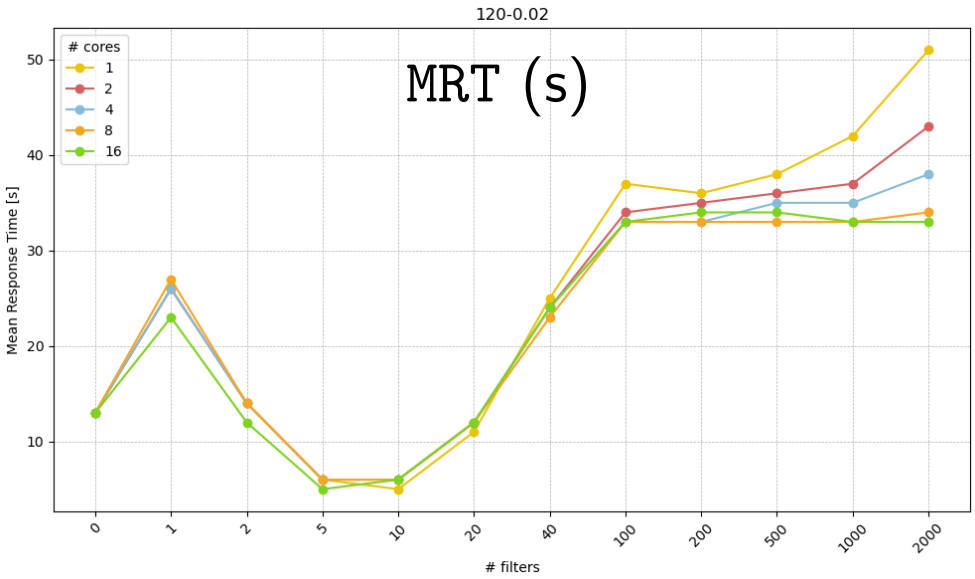
Experiments: E1 - Evaluation in a high-load stress scenario

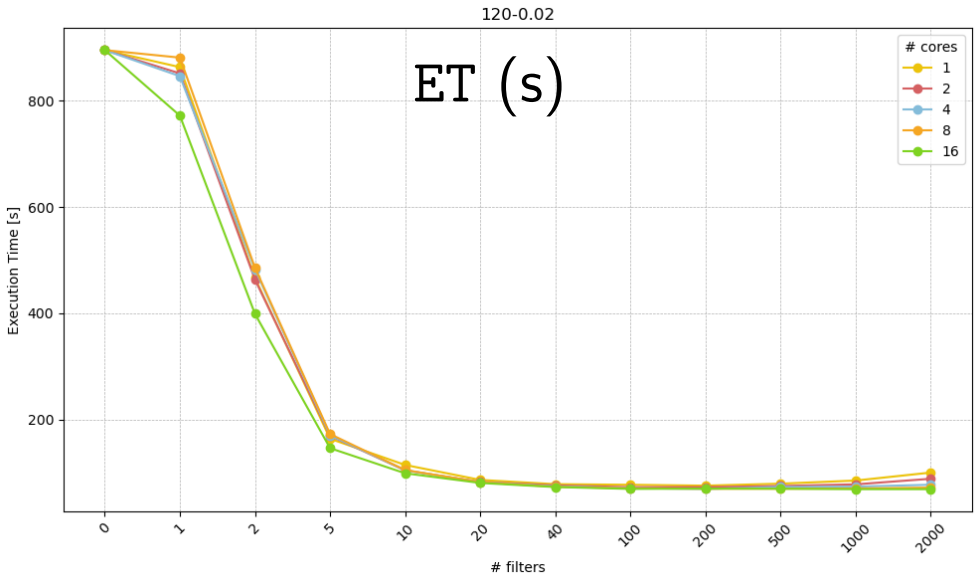
Metrics:

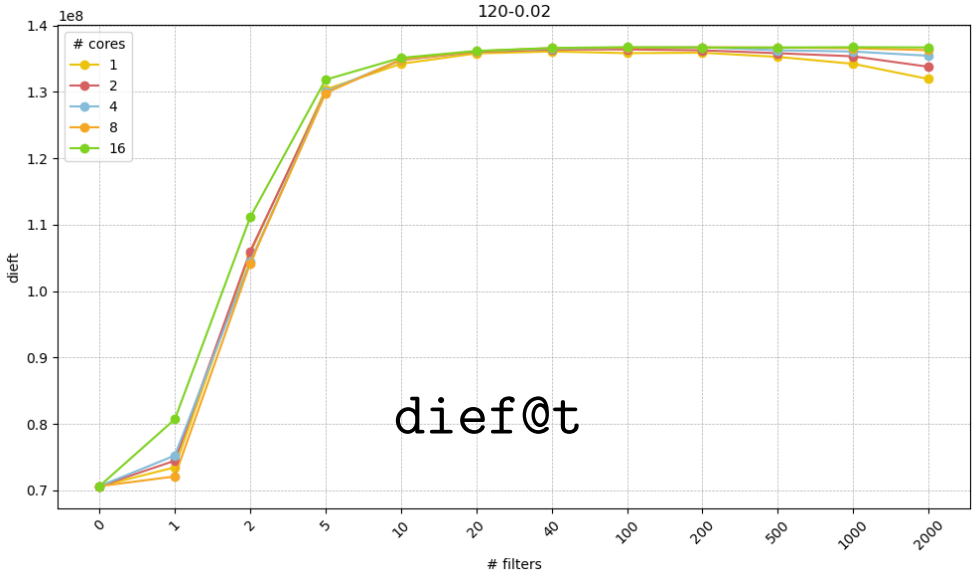
As a real-time system, we evaluated the performance of the DP_{ATM} in terms of:

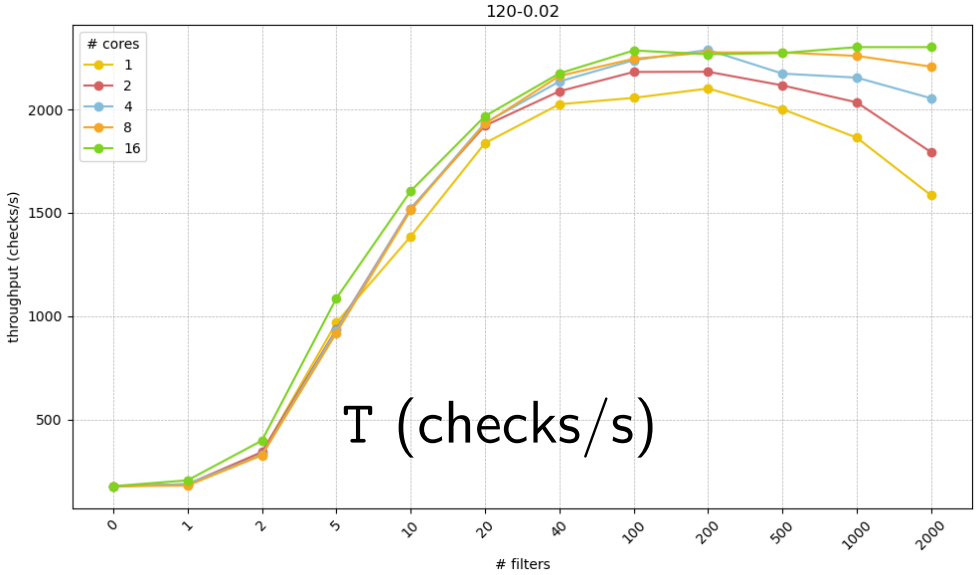
- Response Time (`RT`) and Mean Response Time (`MRT`).
- Execution Time (`ET`).
- Throughput (`T`) (results emitted per second) and Interactions per second (`interactions/s`).
- `dief@t`: evaluation of the continuous emission of results. Acosta, Vidal et al.[1].

E1 - Comparison to Seq. Baseline: $\Sigma(\text{GDB}_A, s(120, 0.02), f, c)$



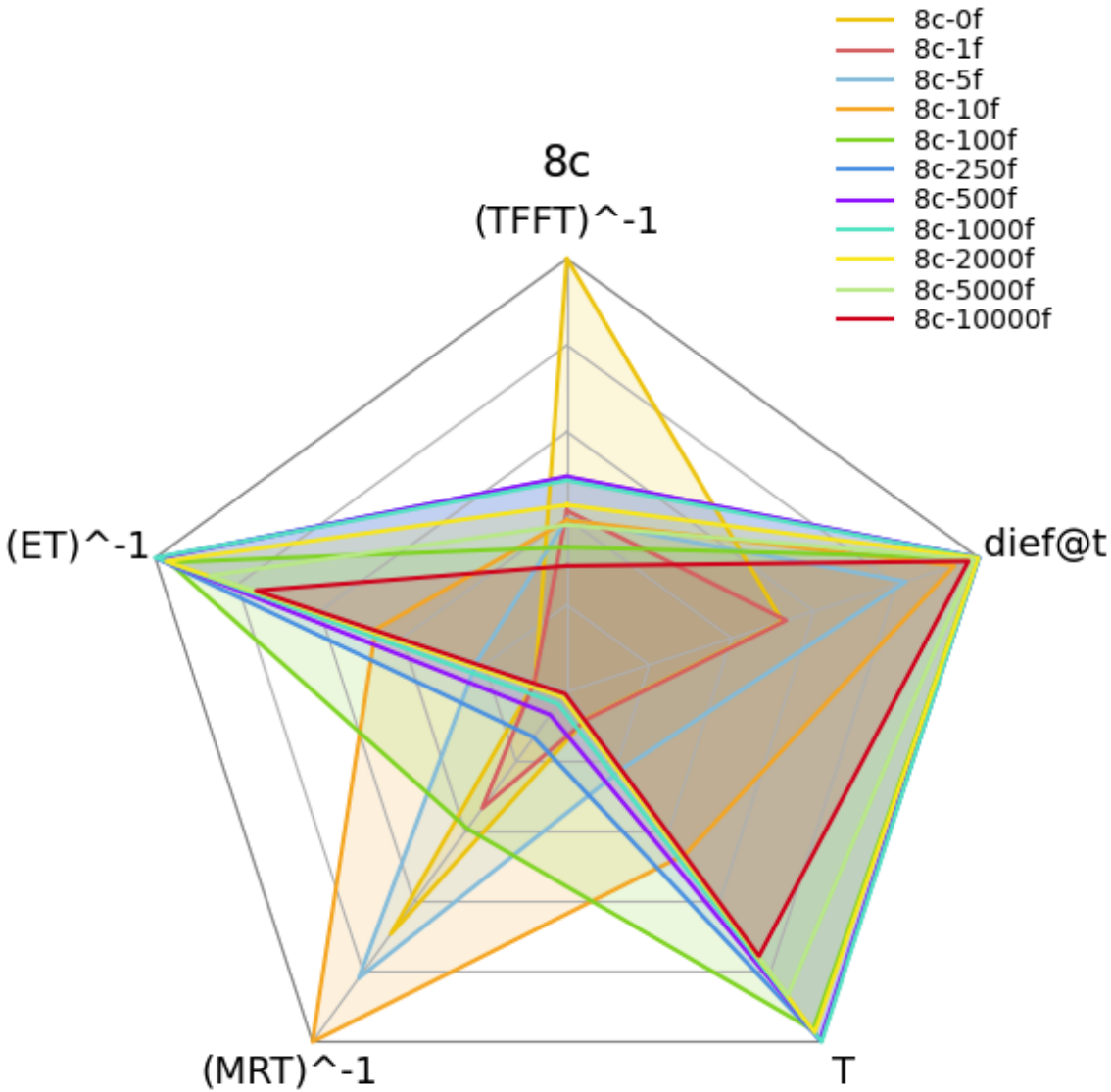






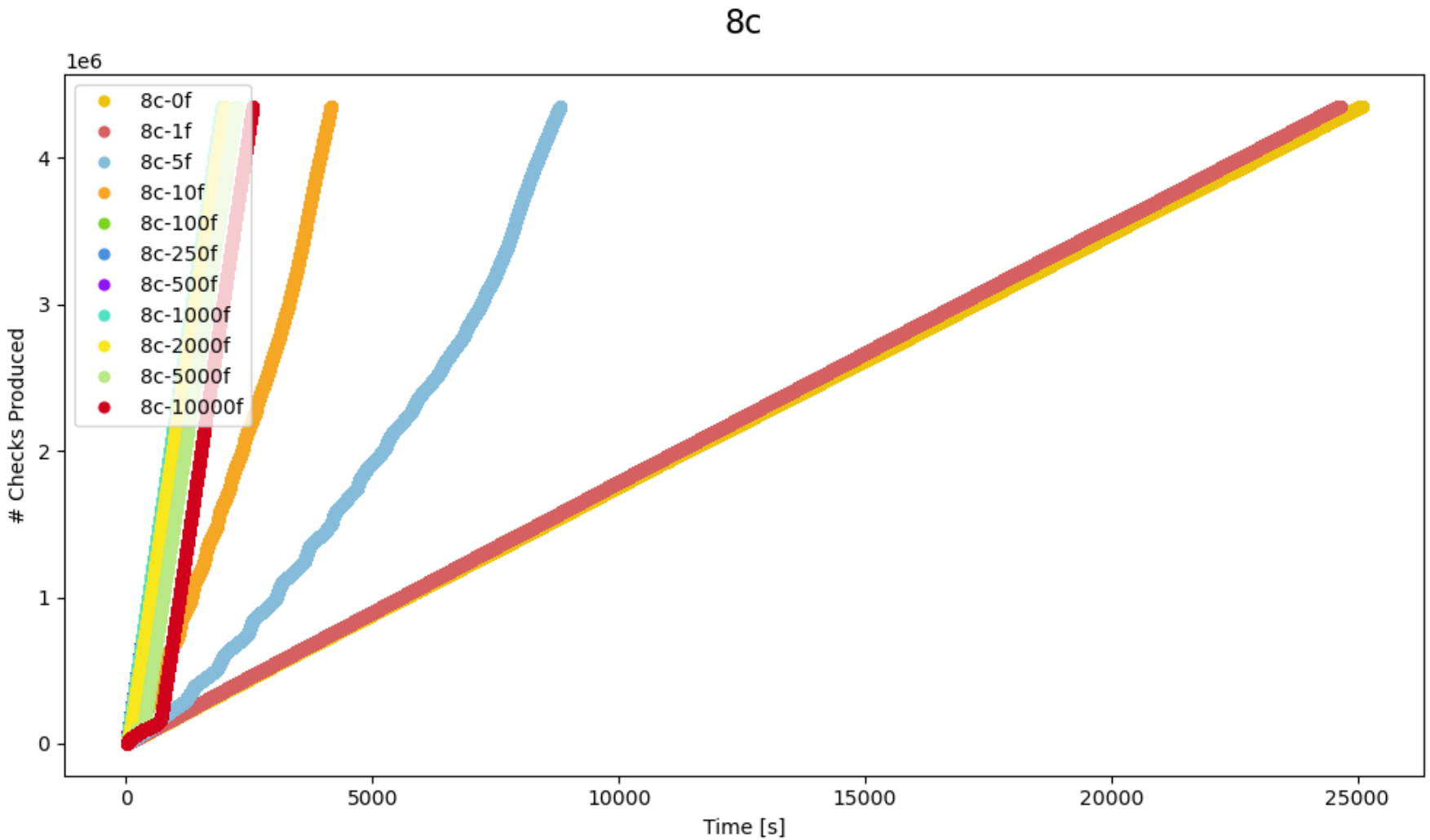
E1 - Num. Filters Configuration: $\Sigma(\text{GDB}_B, s(15, 0.03), f, 8)$

- Best `MRT` for $f = 5$-10.
- Best `ET`, `dief@t` for higher f's.



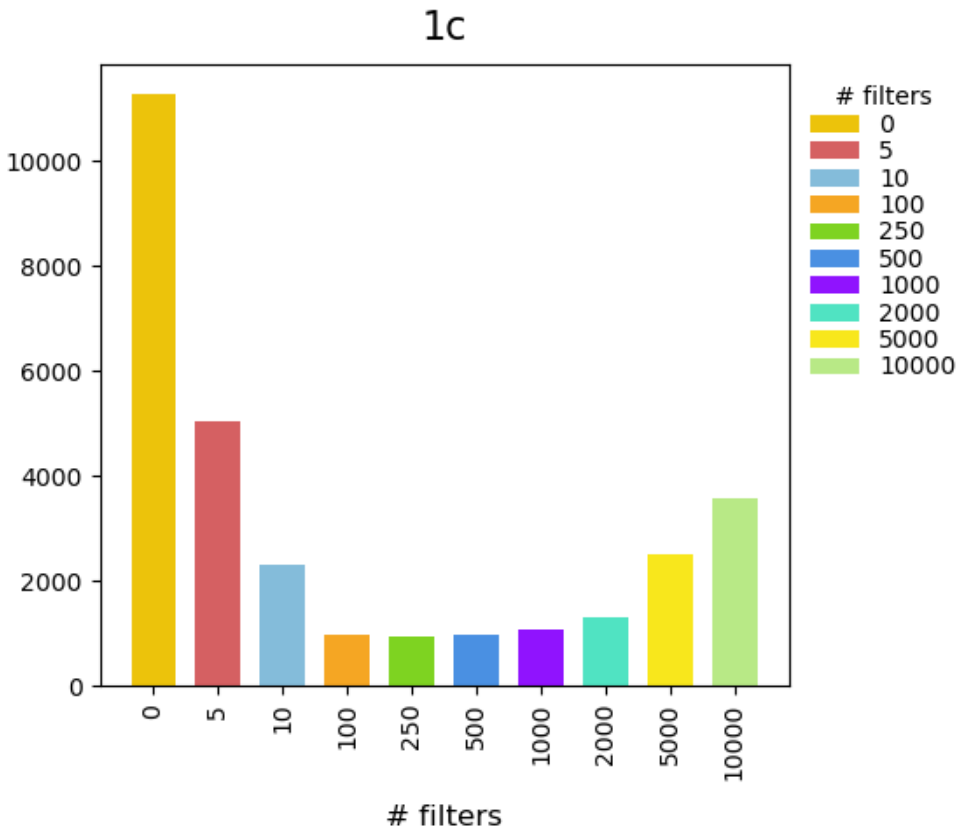
E1 - Num. Filters Configuration: $\Sigma(\text{GDB}_B, s(15, 0.03), f, 8)$

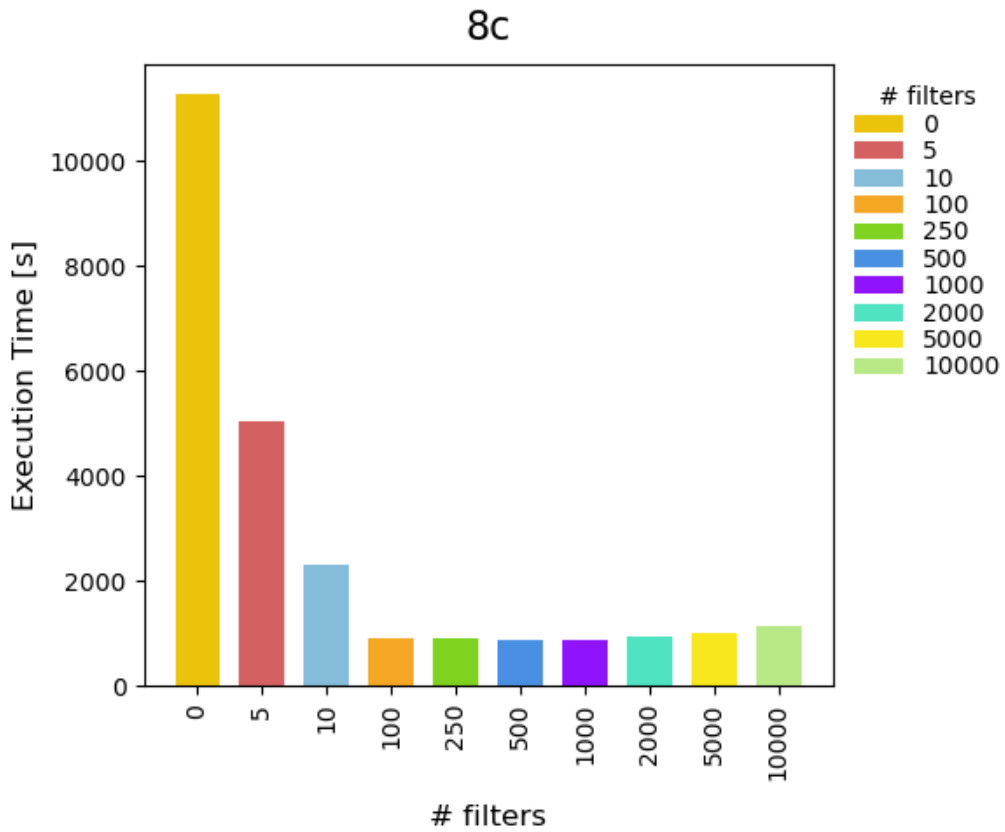
- Check results trace for $\Sigma(\text{GDB}_B, s(15, 0.03), f, 8)$.
- Higher number of filters outperform in continuous behavior.



E1 - Num. Filters Configuration: $\Sigma(GDB_B, s(7, 0.03), f, c)$

- Degradation of the behavior of the configurations with high number of filters f for low number of cores c executions.
- Possible causes: (i) `goroutines` overhead; (ii) Neo4j multiple parallel connections overhead; (iii) Sink (Sk) stage bottleneck.





E1 - Achieved Performance (RT and MRT)

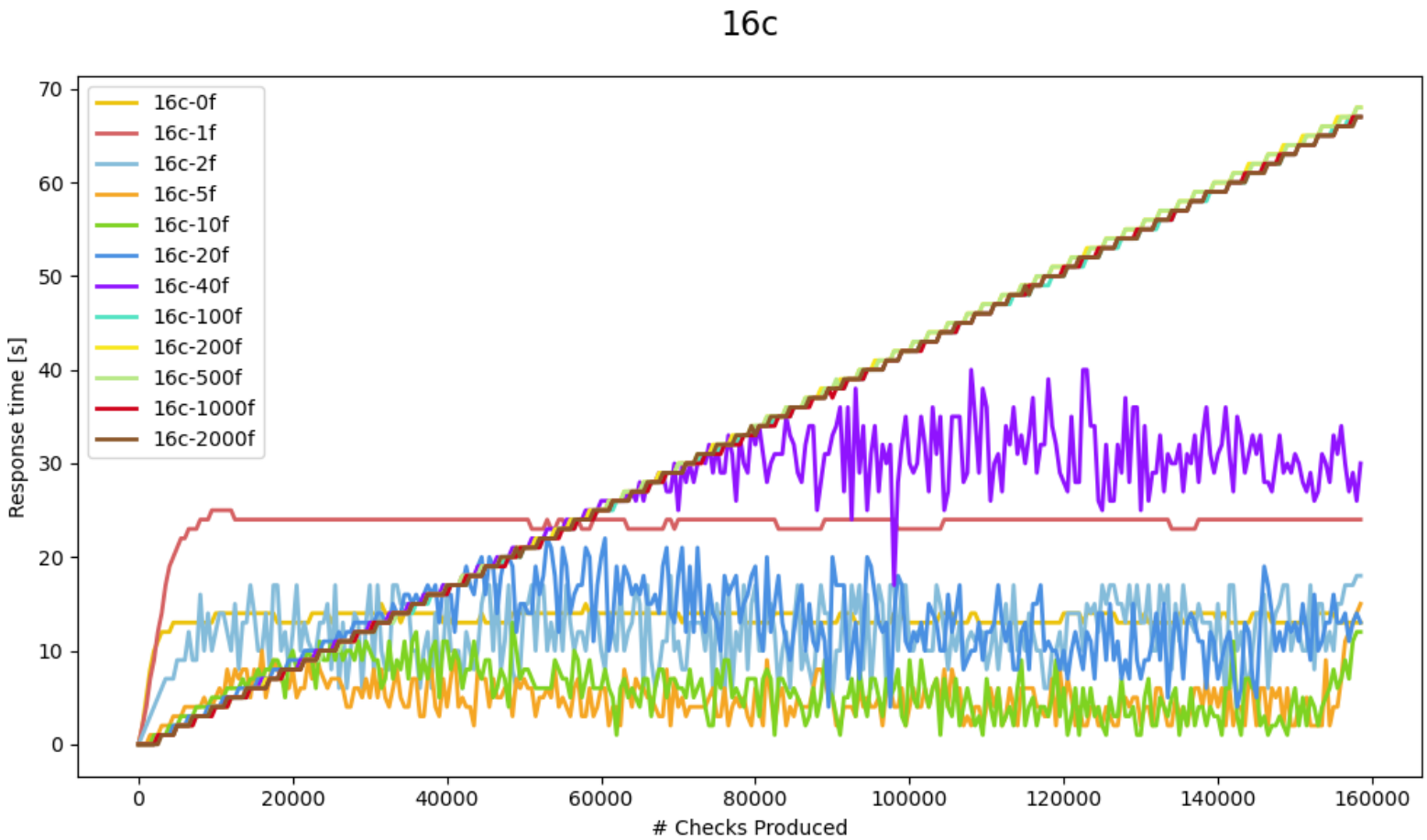
- Achieved low and constant RT values.
- For the largest tested streams for each g the MRT achieved is around 5-6 s for $\Sigma(\text{GDB}_\text{A}, s(120, 0.02), f, c)$ and around 10s for $\Sigma(\text{GDB}_\text{B}, s(15, 0.03), f, c)$.

MRT (s) for $\Sigma(\text{GDB}_\text{A}, s(120, 0.02), f, c)$					
	Number of Cores				
# Filters	**1**	**2**	**4**	**8**	**16**
0	13	13	13	13	13
1	26	26	26	27	23
2	14	14	14	14	12
5	6	6	6	6	5
10	5	6	6	6	6
20	11	12	12	12	12
40	25	24	24	23	24
100	37	34	33	33	33
200	36	35	33	33	34
500	38	36	35	33	34
1000	42	37	35	33	33
2000	51	43	38	34	33

MRT (s) for $\Sigma(\text{GDB}_\text{B}, s(15, 0.03), f, c)$					
	Number of Cores				
# Filters	**1**	**2**	**4**	**8**	**16**
0	13	13	13	13	13
5	11	10	11	11	9
10	11	9	10	9	7
100	33	40	29	23	37
250	124	80	91	69	90
500	221	161	134	136	138
1000	502	301	267	256	263
2000	937	702	618	562	554
5000	2830	1732	1422	1124	1060
10000	4957	2535	2025	1410	1249

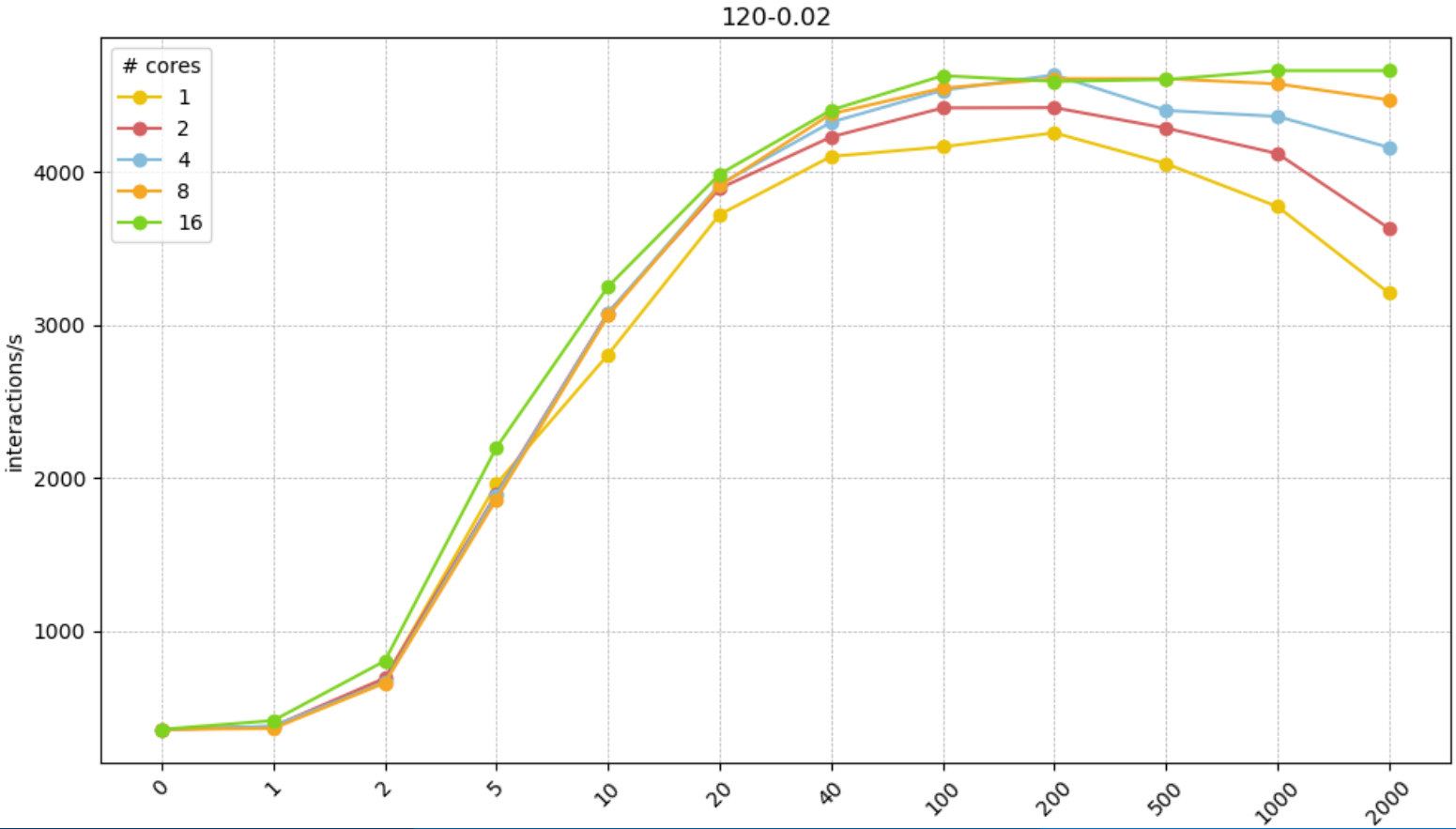
E1 - Achieved Performance (RT and MRT)

- Achieved low and constant RT values.
- E.g. Check results trace for $\Sigma(\text{GDB}_\text{A}, s(120, 0.02), f, 16)$.



E1 - Achieved Performance (`interactions/s`)

- Achieved large transaction processing rates:
  - $\Sigma(\text{GDB}_\text{A}, s(120, 0.02), f, c)$: 5,000 `interactions/s` (2,500 transactions/s). 216,000,000 per day.
  - $\Sigma(\text{GDB}_\text{B}, s(15, 0.03), f, 16)$: 2,500 `interactions/s` (1,250 transactions/s). 108,000,000 per day.



Conclusions & Future Work

$\rightarrow$ (For small number of filters configurations (5-10)) DP_{ATM} proved to be a effective real-time system with 100% accuracy with low and constant response time RT (less than 11s in all cases), with a stream processing capacity speed that suggest that the DP_{ATM} can be extrapolated to real big bank systems ($>$ 100M of transactions per day).

Future work:

- Investigate/Improve the response time performance for the combinations with large number of filters.
- Include more types of ATM card frauds or even other kind of frauds (POS or CNP).
- Study the problem of window management.
- Experiment with larger banking data graphs.