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

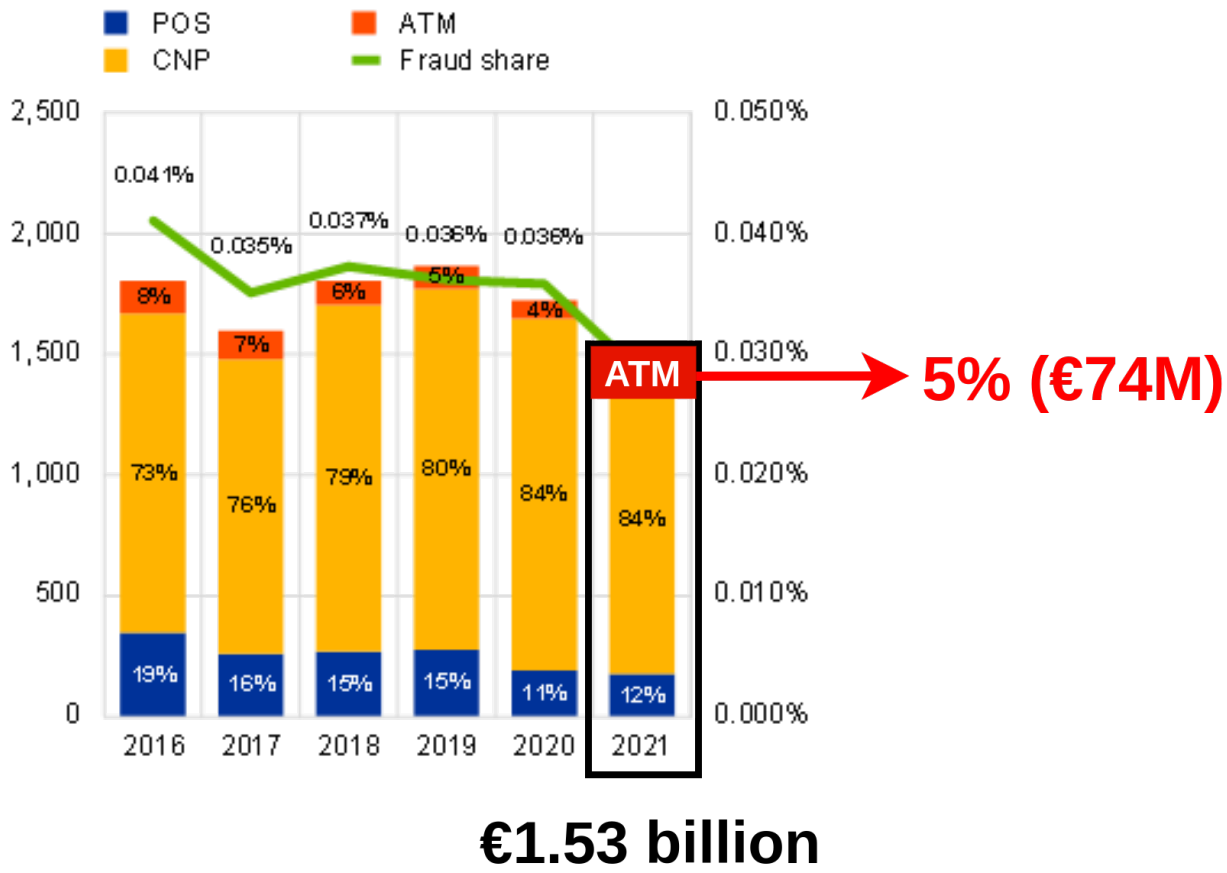
Fernando Martín Canfrán

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

Facultat d'Informàtica de Barcelona
Universitat Politècnica de Catalunya

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].



[1] Report on card fraud in 2020 and 2021 [**ECB2023**]

Motivation: ATM Fraud Detection - State of the Art

-  **Classical Treatment**:
  - ✗ **Delayed** annoying consulting of **log files** because of customers complain when they themselves detect some weird movement in their accounts.
-  **ML Systems**:
  - ✗ Detection based on **predictions**, not exact detection ($< 100\%$ accuracy).
  - ✗ Need **training** process and big volumes of **data**.
  - ✗ Quality of the predictions depends on the training data (possibly **biased**).

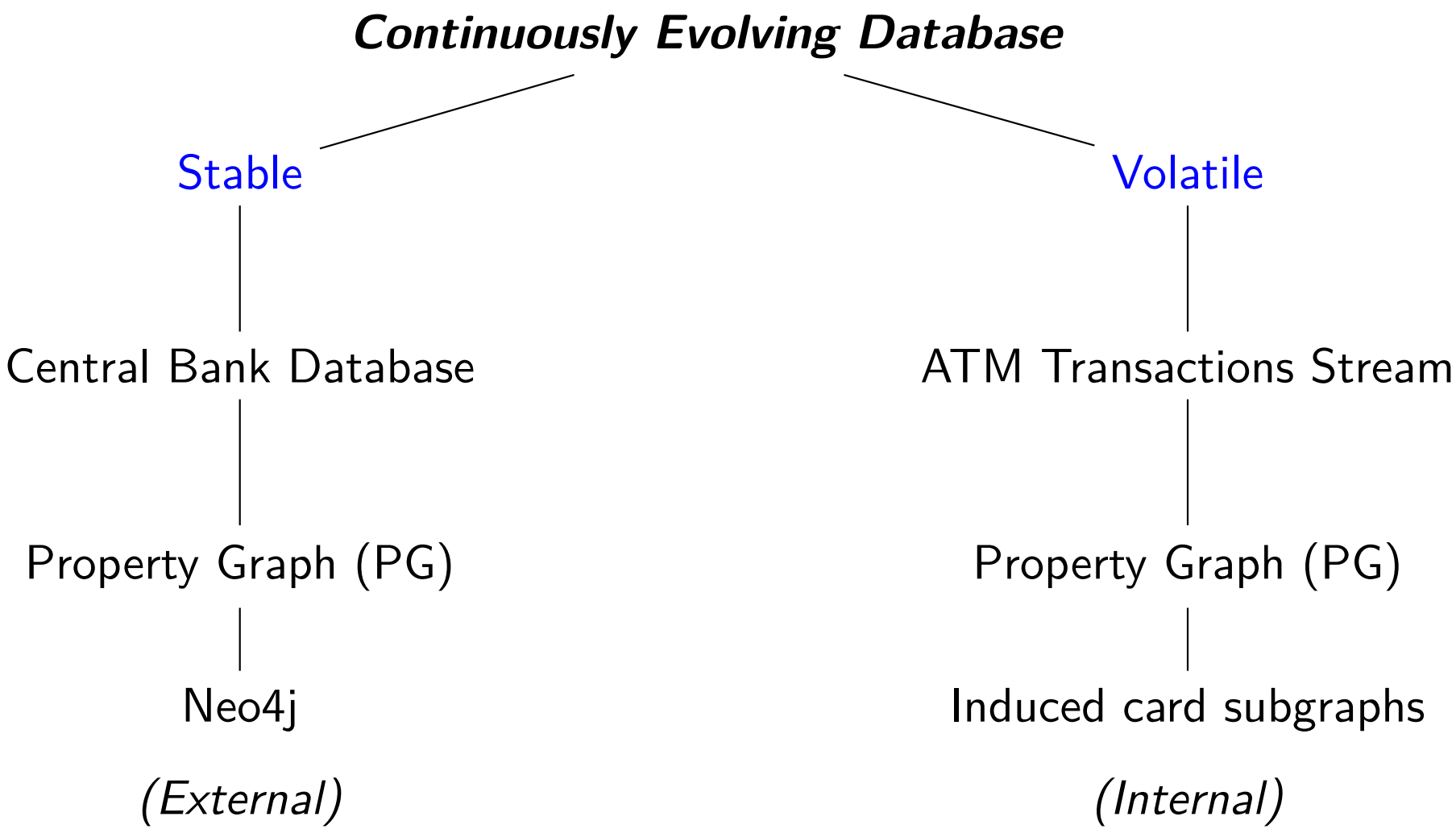
$\rightarrow$ To our knowledge, most of the works are based on artificial intelligence techniques. No clear characterization of the problem. [**Rahman2019**].

Motivation: ATM Fraud Detection

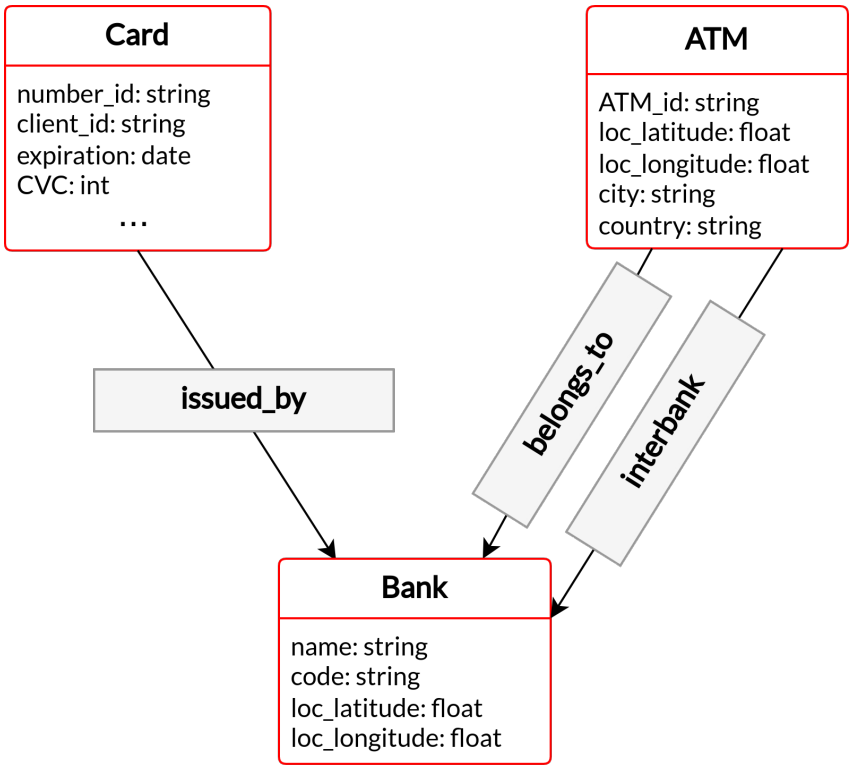
We propose a system to overcome the previous problems:

- ✓ A **exact** detection of the fraud. **100% accuracy**.
- ✓ In **real-time**.
- ✓ With a **clear characterization** of some fraud patterns.
- ✓ A synthetic **bank dataset** and **transaction stream generator tool** (lack of real-world data).

Proposal: Data Model

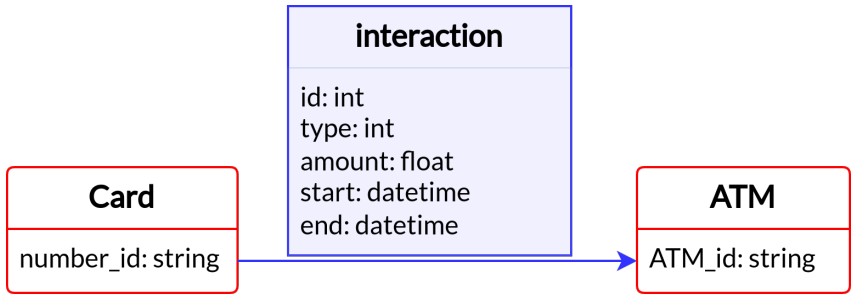


Proposal: Data Model



Stable PG

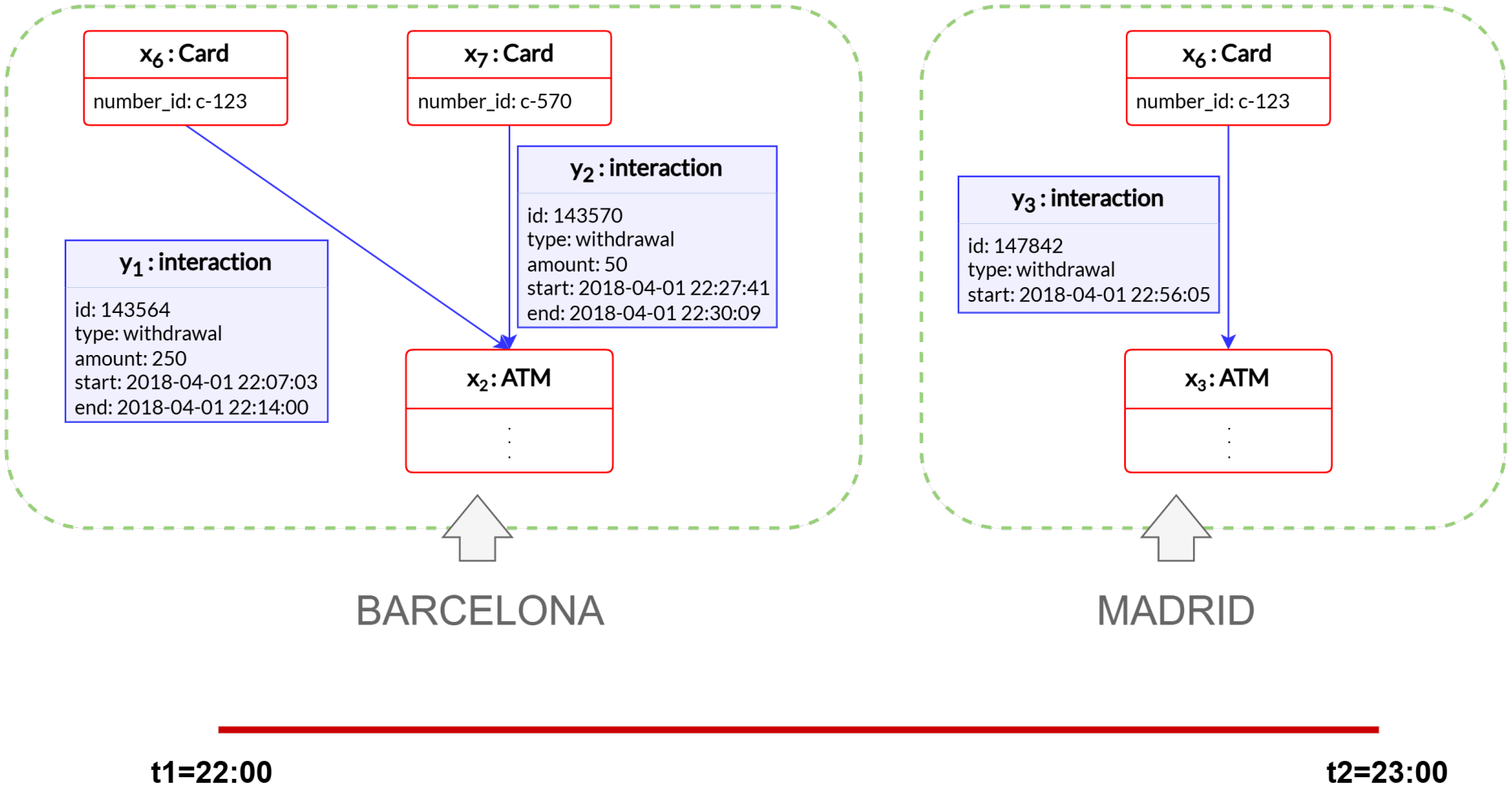
Central bank database (*persistent relations of standard graph dbs).*



Volatile PG

Transactions (*non-persistent relations. Edges in the data stream).* ***Induce subgraphs used to perform query evaluation***

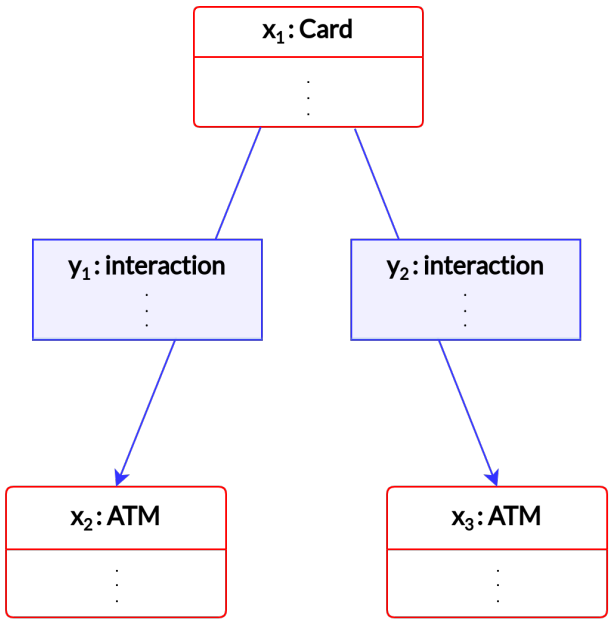
Proposal: Definition of Anomalous Patterns of Transactions



Card cloning characterization - an example

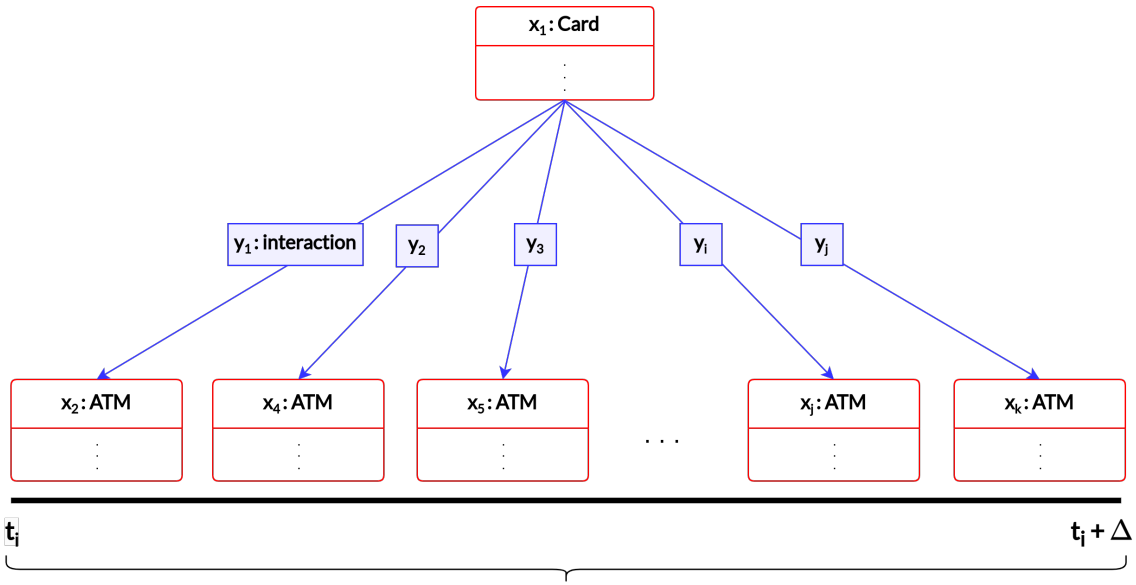
Proposal: Definition of Anomalous Patterns of Transactions

- 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



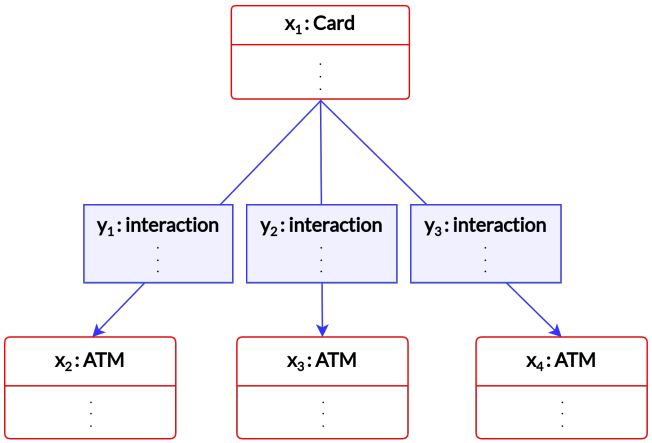
- ATMs close to each other (possibly in the same neighborhood/district/city)
- #(y:interaction) > fraud_threshold(t_i, $t_i + \Delta$)

Lost-and-stolen card

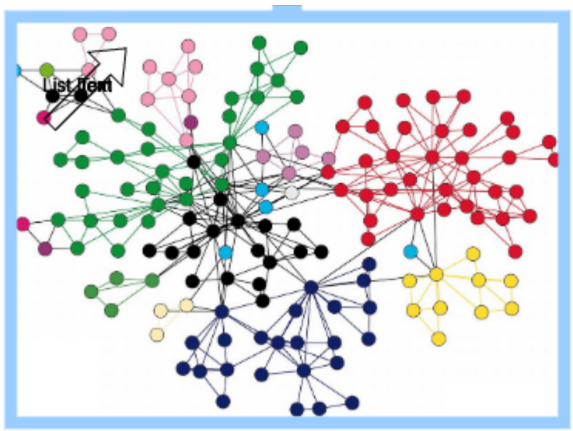
Proposal: Query Model

- ***Continuous query model***: Fixed queries evaluated over data streams.
- Progressive query evaluation process. Based on:
  - Graph pattern matching in the volatile subgraph.

    +
  - Satisfiability of constraints over the properties: possibly querying the stable graph database (*retrieve some additional info*).



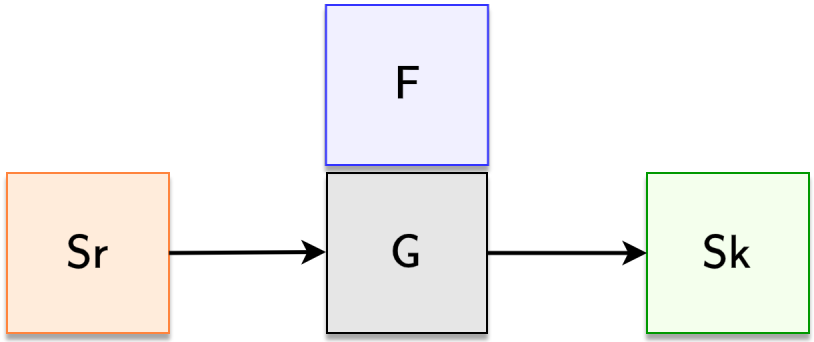
Volatile subgraph

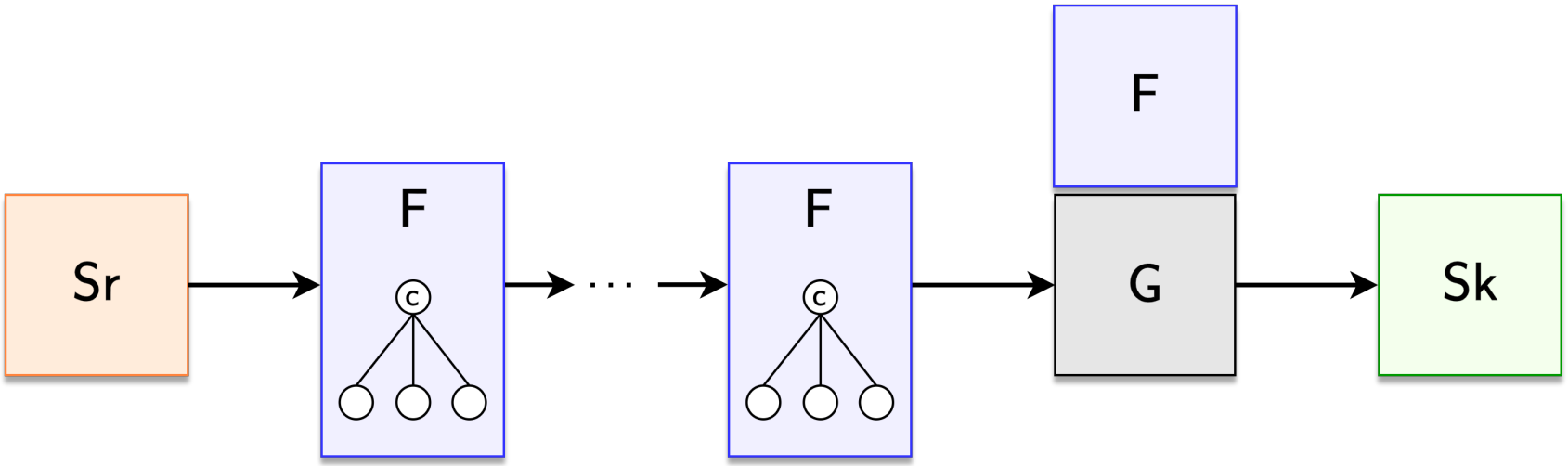


Stable gdb

Proposal: Continuous Query Engine - DP_{ATM}

- Stream processing computational model. Dynamic Pipeline.
- Execution of tasks (query evaluations) concurrently/in-parallel.

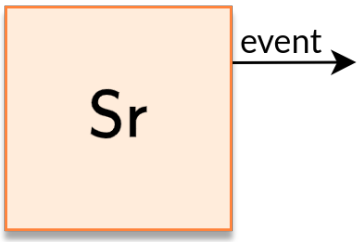




Based on **Dynamic Computational Approach** - [**Pasarella2024**].

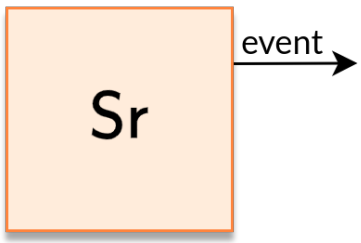
Continuous Query Engine - DP_{ATM}: Stages

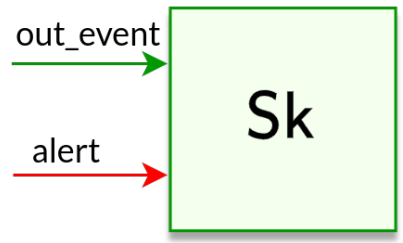
- **Source:** Manages the in-connection with the outside: streaming input / file reading... & general transactions log.



Continuous Query Engine - DP_{ATM}: Stages

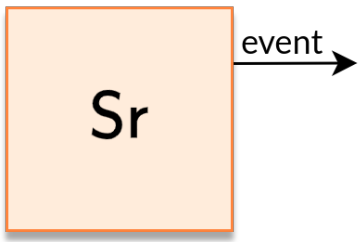
- **Source:** Manages the in-connection with the outside: streaming input / file reading... & general transactions log.
- **Sink:** Manages the out-connection. Outside answers/alerts emission & alert log.

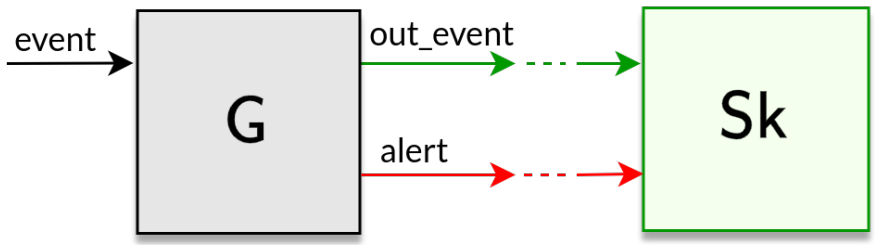




Continuous Query Engine - DP_{ATM}: Stages

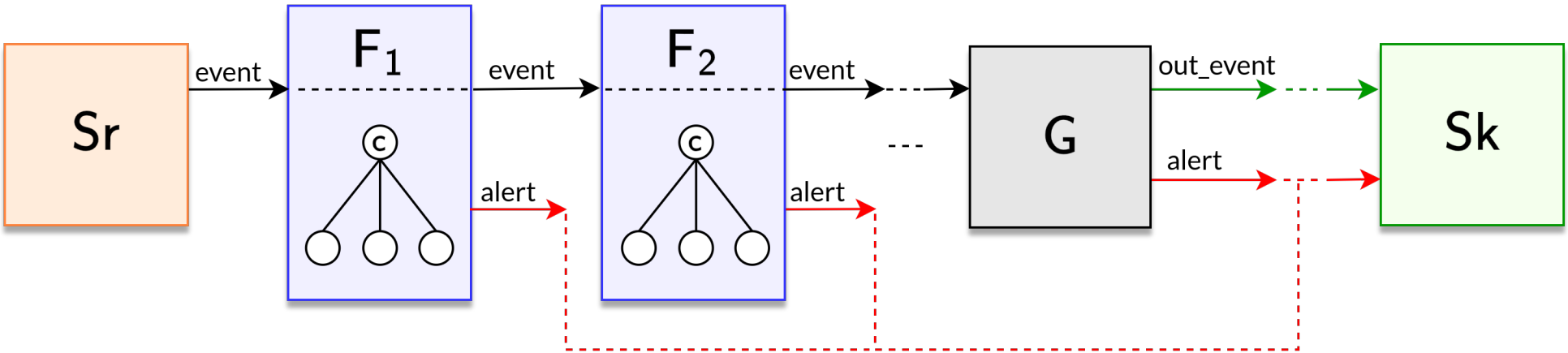
- **Source:** Manages the in-connection with the outside: streaming input / file reading... & general transactions log.
- **Sink:** Manages the out-connection. Outside answers/alerts emission & alert log.
- **Generator:** Generation of new filters.





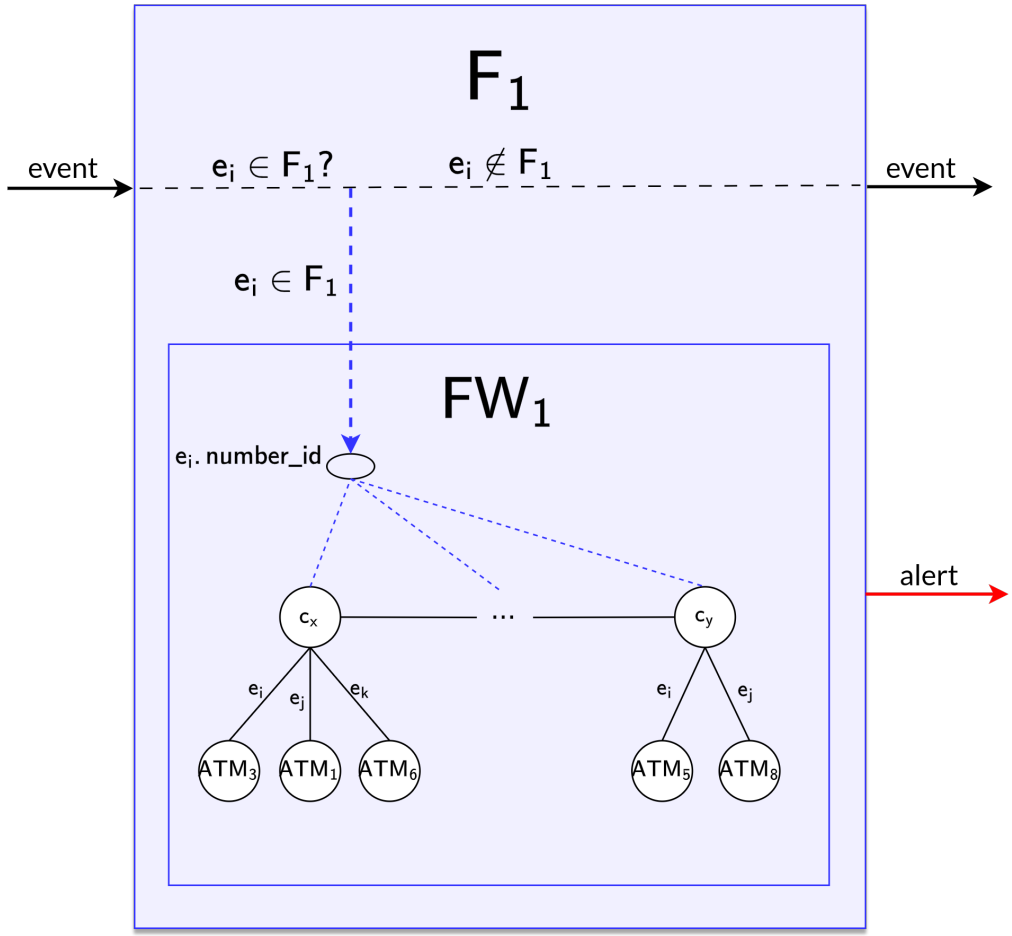
Continuous Query Engine - DP_{ATM}: Stages

- **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.

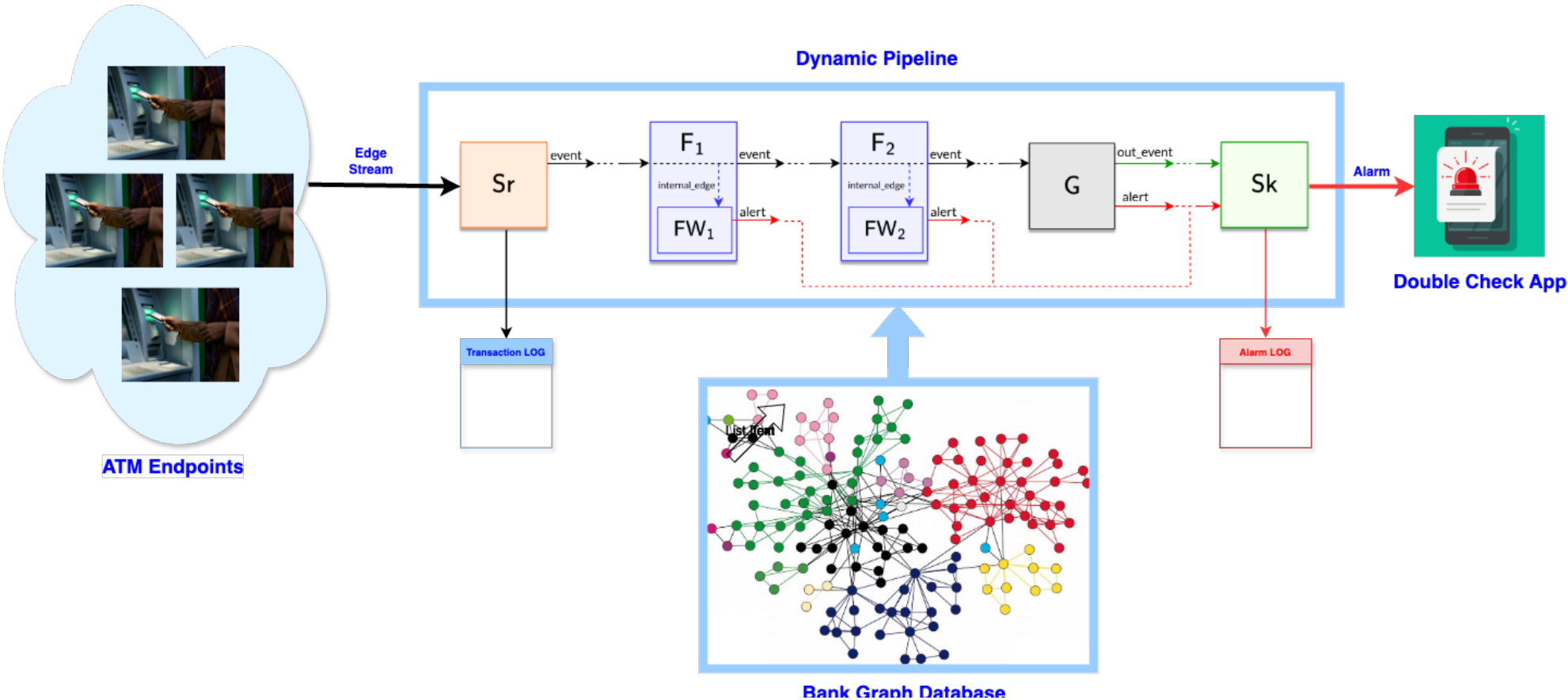


Continuous Query Engine - DP_{ATM}: Filter Stage

- Stores the **induced card subgraph(s)** of the incoming belonging edges.
- Evaluation of continuous query pattern(s). Each has a **connection** session with the **stable gdb**.
- Emission of **alerts** in case of matching a query pattern.
- Filter *worker* FW to avoid bottlenecks - *in parallel*.

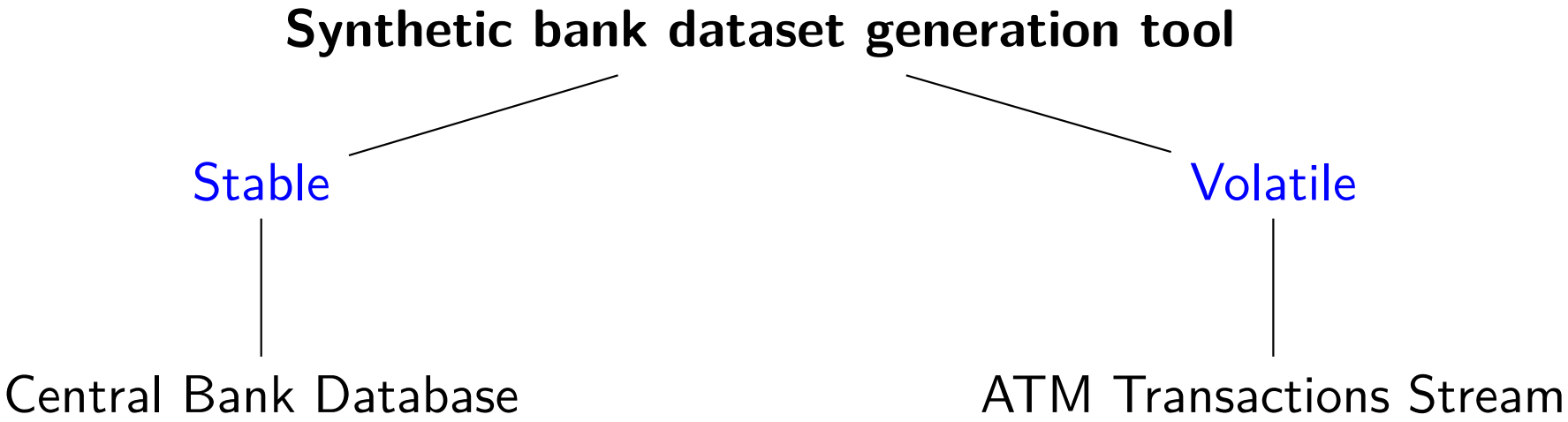


Continuous Query Engine - DP_{ATM}: Architecture



Experiments: Datasets

- No public real bank dataset found.



- Customisable data generation.
- Based on *Wisabi Bank Dataset* (publicly available). Used as a reference for the **geographical** distribution of **ATM** locations and **card/client** transaction ***behavior***.
- Left as a public available tool.[2]

[2] `https://github.com/FCanfran/ATM-DP/tree/main/gdb`

Experiments: Datasets

- **Bank database generator**. `bankDataGenerator.py`: creates a dataset of n ATMs and m Cards.
- **Transaction stream generator**. `txGenerator.py`: parametrizable, with `ratio` $\in [0, 1]$ of anomalous tx..
  - *Regular* transactions: avoiding creation of fraud scenarios.
  - Injection of transactions that create anomalous scenarios.

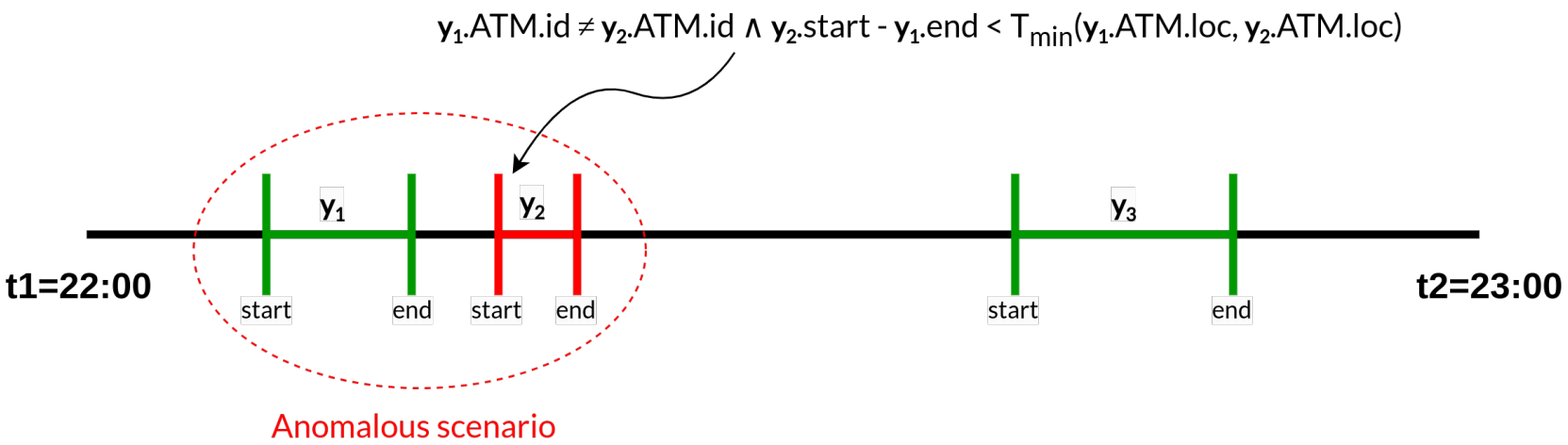


Figure: Creation of anomalous scenario I - Card cloning

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	**|Card|**	**|ATM|**
$\text{GDB}_\text{A} \approx$ *Caixabank* small office	2,000	50
$\mathbf{GDB_B} \approx$ *Deutsche Bank Spain*	500,000	1,000

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

Bank DB (g)	**|Days|** (k)	**Anomalous Ratio** (p)	**|Anom. Tx|**	**Stream Size |Tx|**
GDB_A	30	0.02 (2%)	451	39,959
GDB_A	60	0.02 (2%)	1,739	80,744
GDB_A	120	0.02 (2%)	2,994	160,750
GDB_B	7	0.03 (3%)	26,480	2,428,286
GDB_B	15	0.03 (3%)	50,653	4,856,573

$\sim 4 \times 10^4$

$\sim 5 \times 10^6$

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

Experimental variations. $\Sigma = (g, s(k, p), f, c)$
with these combinations on:

- 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: $c = 1, 2, 4, 8, 16$.

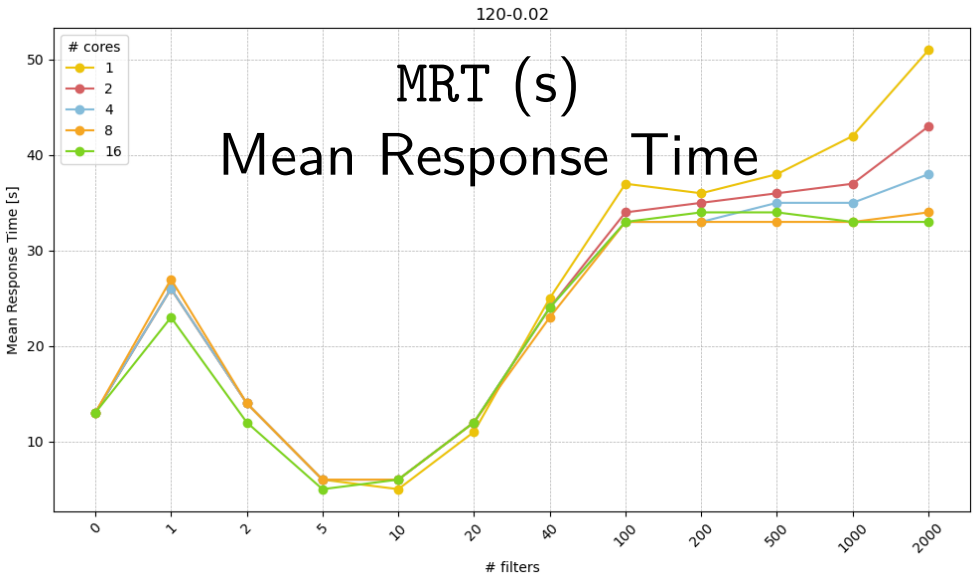
$\Rightarrow$ E.g. $\Sigma(\text{GDB}_\text{A}, s(30, 0.02), f, c)$

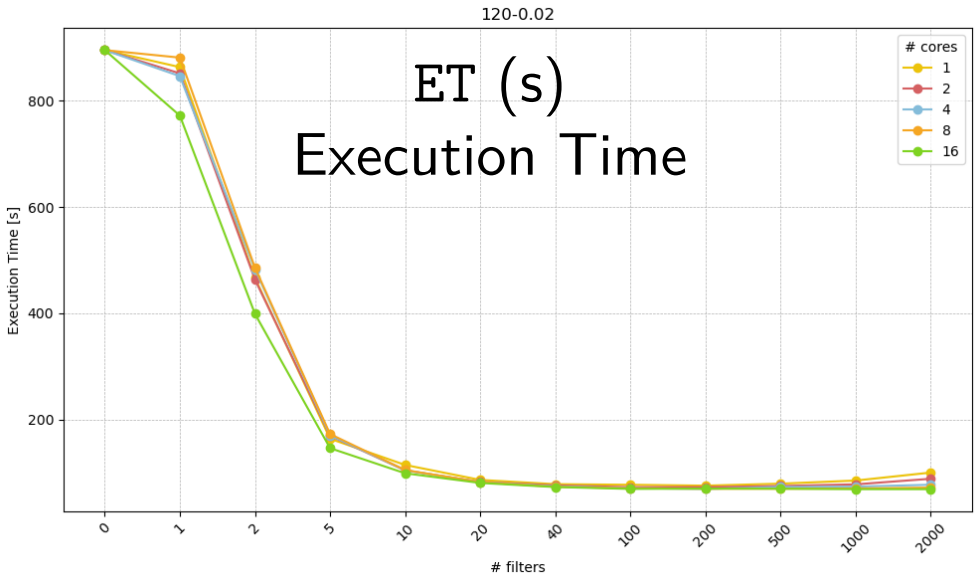
Results consideration: All fraud pattern **`checks` as results** for experimental purposes. Only `alerts` as results in production.

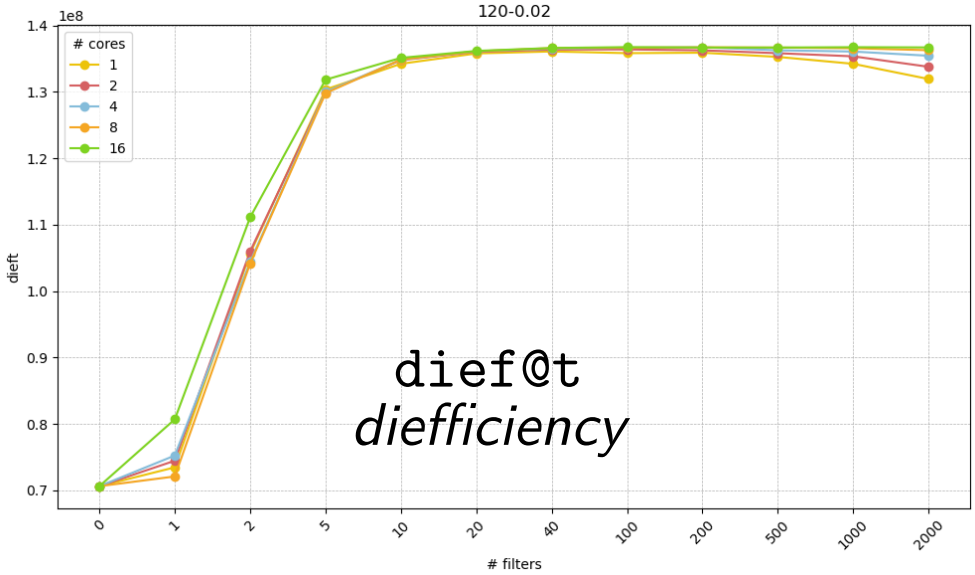
Objectives:

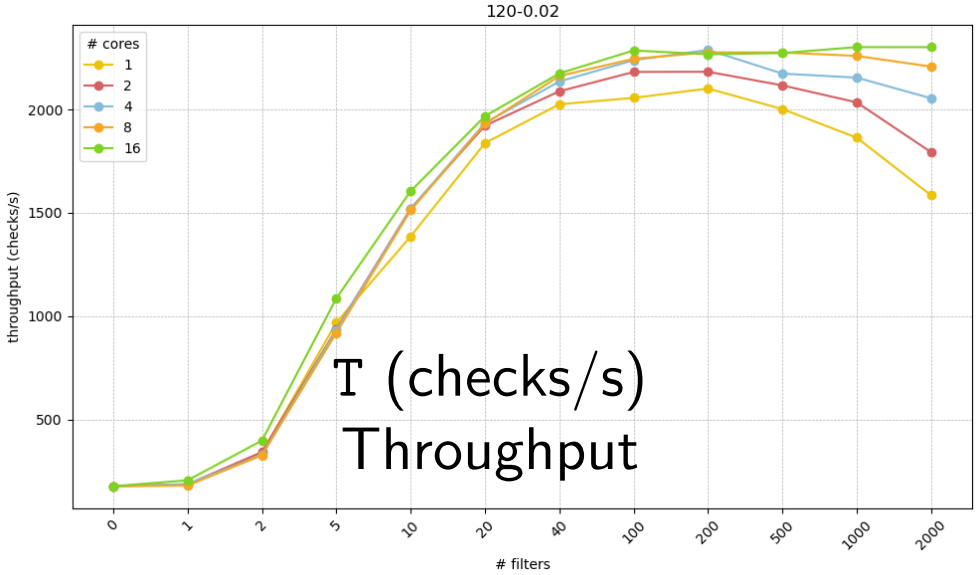
- Comparison to **sequential** baseline.
- Analysis of the **number of filters** configuration.
- Analysis of the behavior and **suitability** of the DP_ATM as a real-time engine.

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



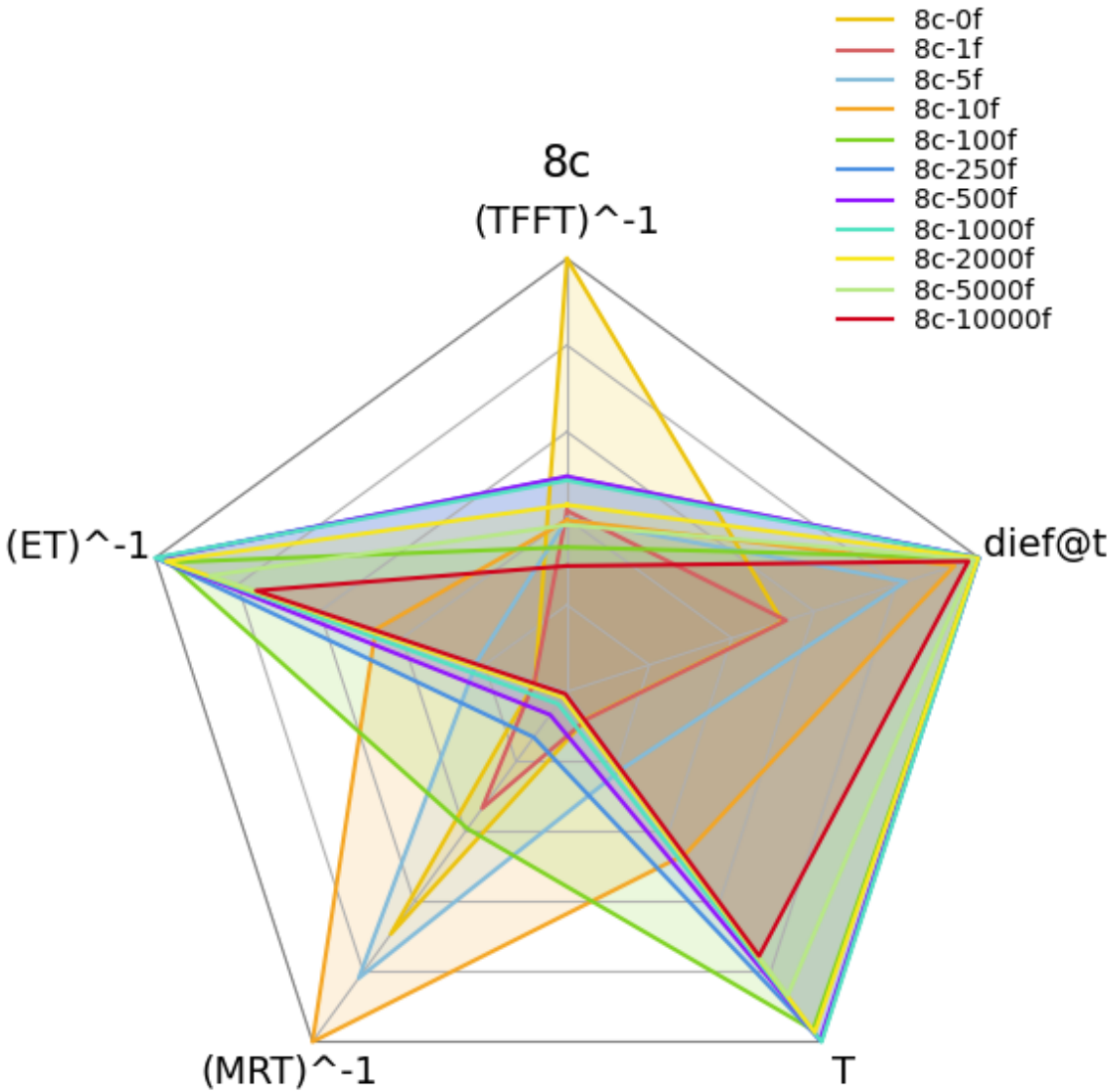






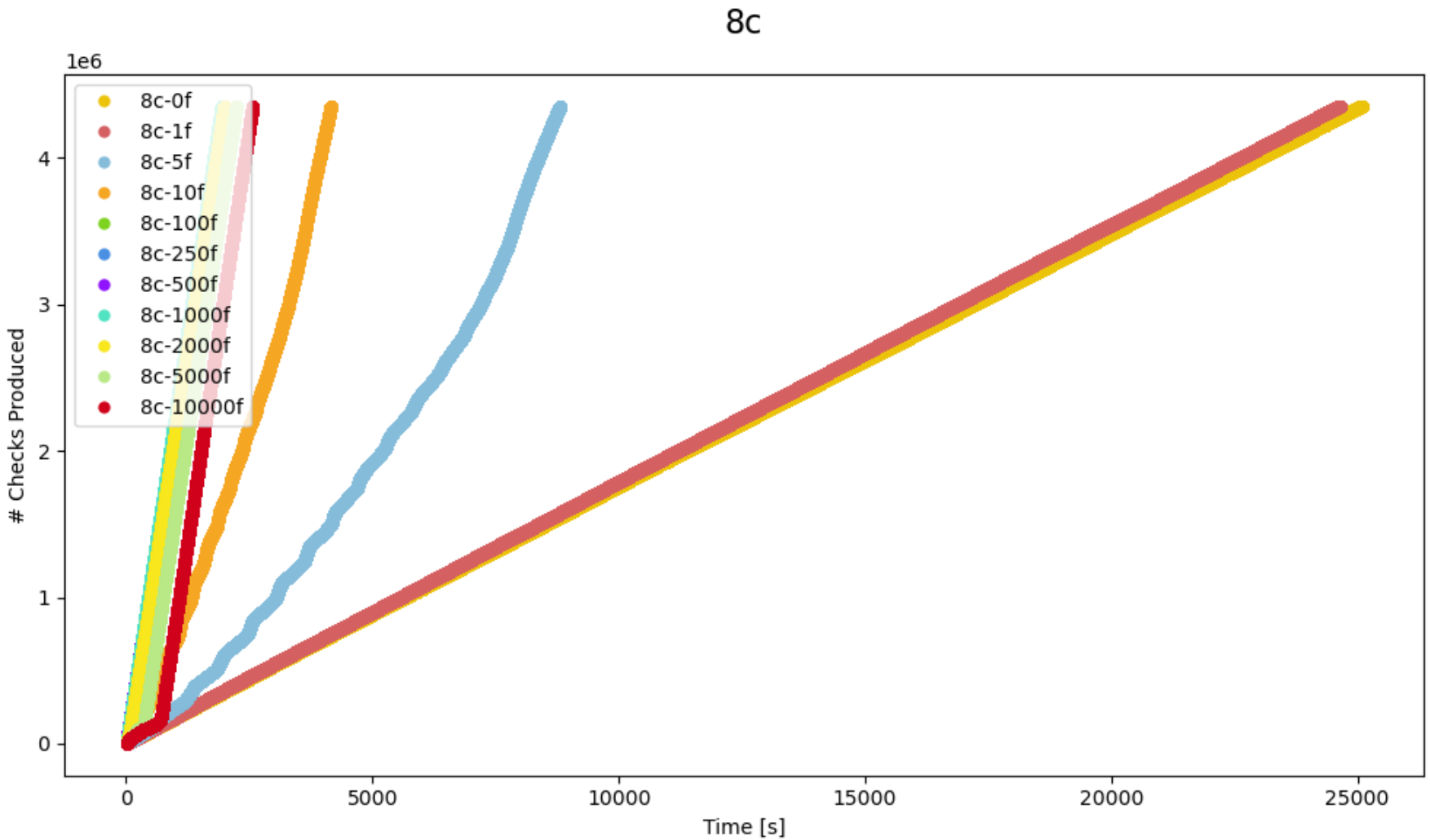
E1 - Num. Filters Configuration: $\Sigma(\text{GDB}_B, s(15, 0.03), f, 8)$

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



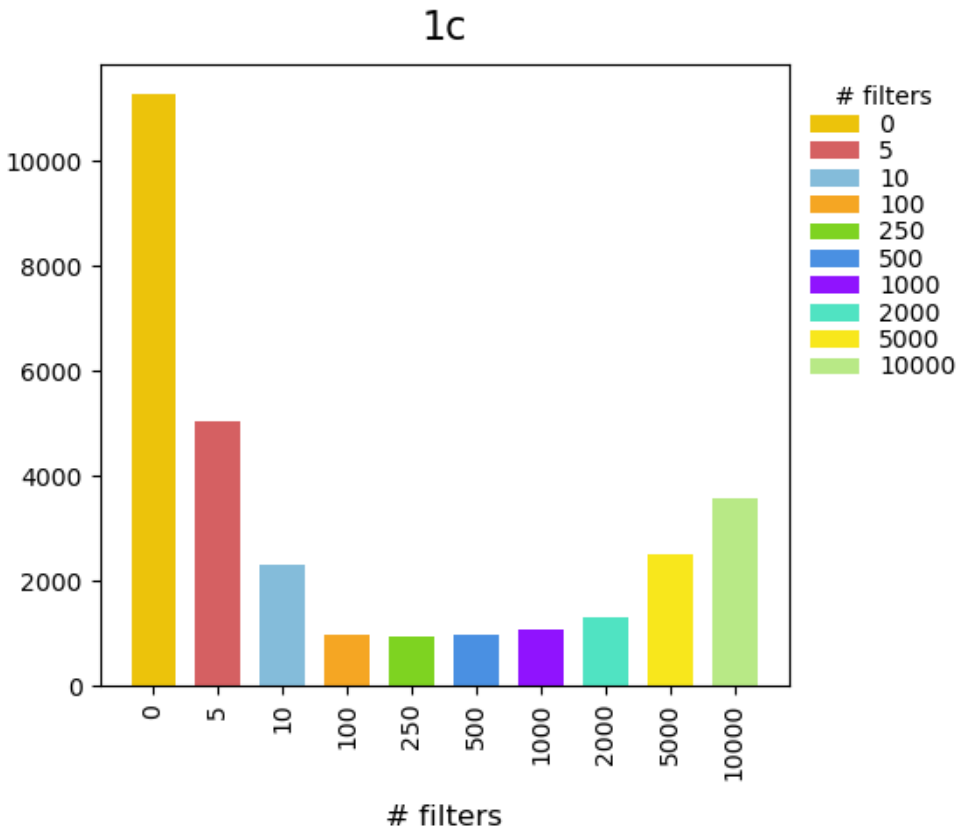
E1 - Num. Filters Configuration: $\Sigma(\text{GDB}_B, s(15, 0.03), f, 8)$

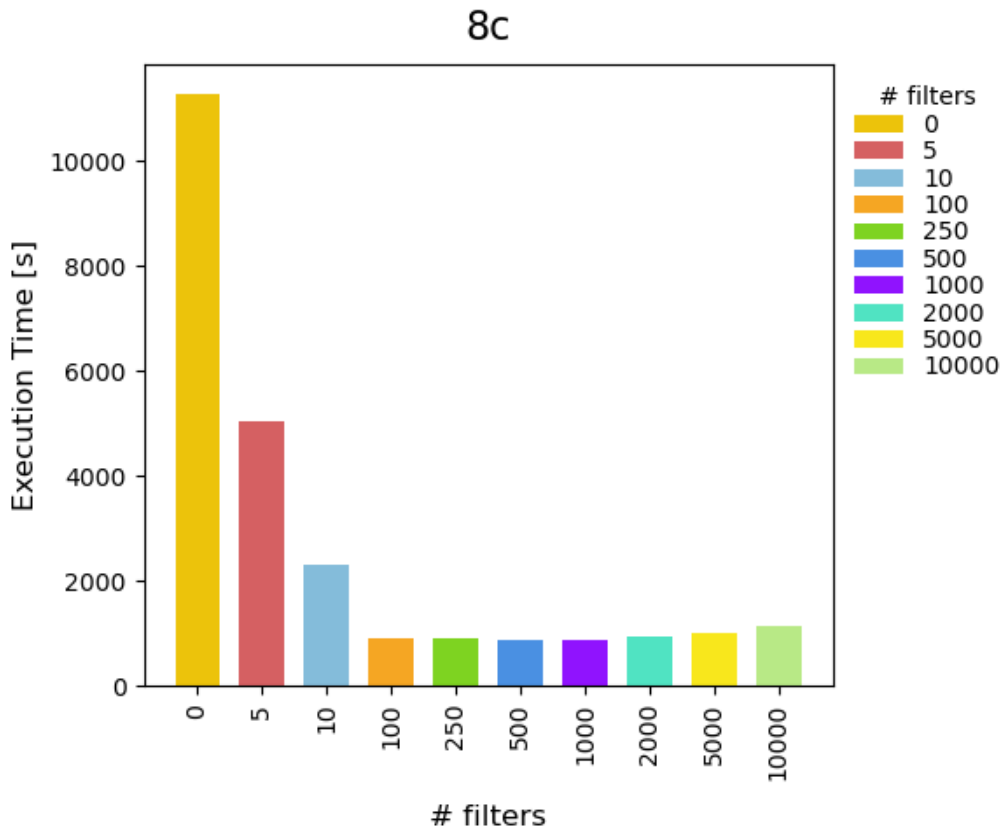
- Check results trace for $\Sigma(\text{GDB}_B, s(15, 0.03), f, 8)$.
- **Higher** number of **filters** outperform in **continuous**



E1 - Num. Filters Configuration: $\Sigma(\text{GDB}_B, s(7, 0.03), f, c)$

- **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)

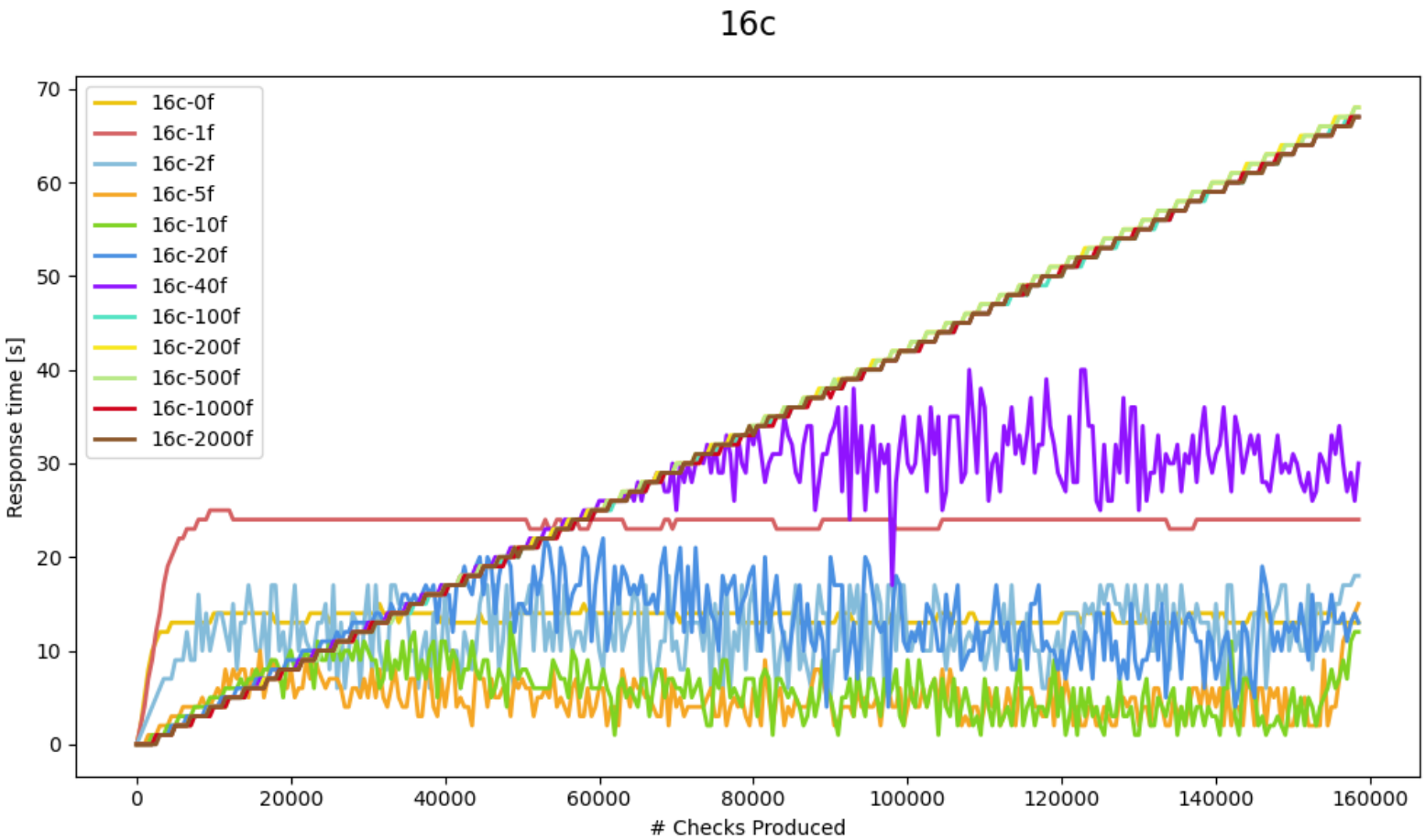
- Achieved **low** and **constant** RT (Response Time) values.
- For the **largest tested streams** for each g the MRT achieved is around 5-6 s for $\Sigma(\text{GDB}_\text{A}, s(120, 0.02), f, c)$ and around 10s for $\Sigma(\text{GDB}_\text{B}, s(15, 0.03), f, c)$.

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

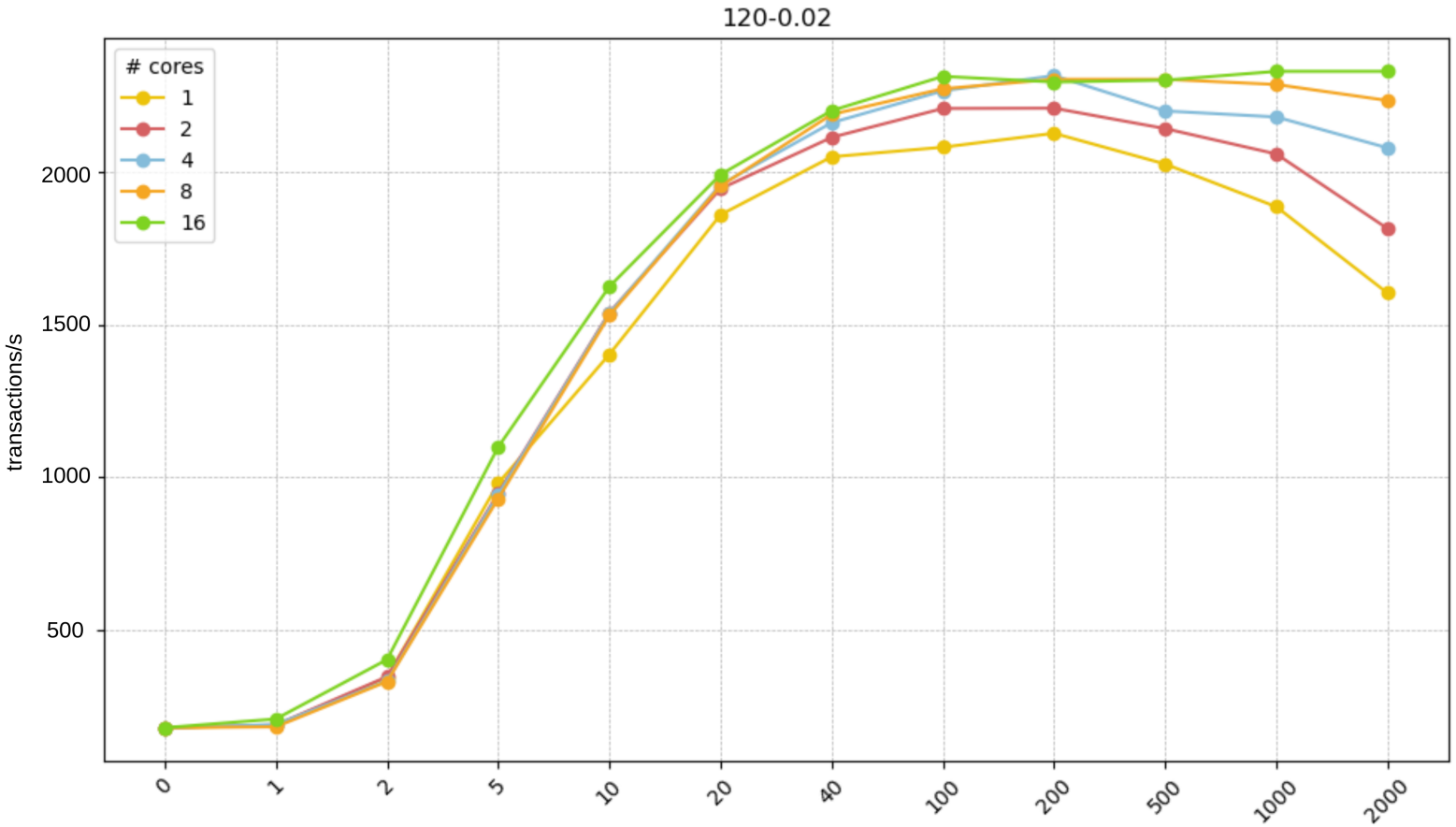
E1 - Achieved Performance (RT and MRT)

- Achieved **low** and **constant** RT (Response Time) values.
- E.g. Check results trace for $\Sigma(\mathrm{GDB}_A, s(120, 0.02), f, 16)$.



E1 - Achieved Performance (transactions/s)

- Achieved **large** transaction **processing rates**:
  - $\Sigma(\text{GDB}_\text{A}, s(120, 0.02), f, c)$: 2,500 transactions/s. 216,000,000 per day $\approx$ 200M.
  - $\Sigma(\text{GDB}_\text{B}, s(15, 0.03), f, 16)$: 1,250 transactions/s. 108,000,000 per day. $\approx$ 100M.



Conclusions

DP_{ATM} proved to be a effective real-time system with:

- **100% accuracy**.
- **Low** and **constant** response time `RT` (less than 11s in all cases).
- A stream **processing capacity speed** that suggest that the DP_{ATM} can be extrapolated to **real big bank systems** ($>$ 100M of transactions per day).

$\rightarrow$ Working version of DP_{ATM} publicly available on Github[3]

[3] `https://github.com/FCanfran/ATM-DP`

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):
  - *Card cloning* characterization
  - *Lost-and-stolen* card characterization
  - Anomalous location usage
  - Anomalous number of operations
  - Anomalous high expenses
- Study the problem of **window management**.
- Experiment with **larger** banking data **graphs**.

$\rightarrow$ Partial results of this work were reported and presented in the Alberto Mendelzon Workshop 2024.