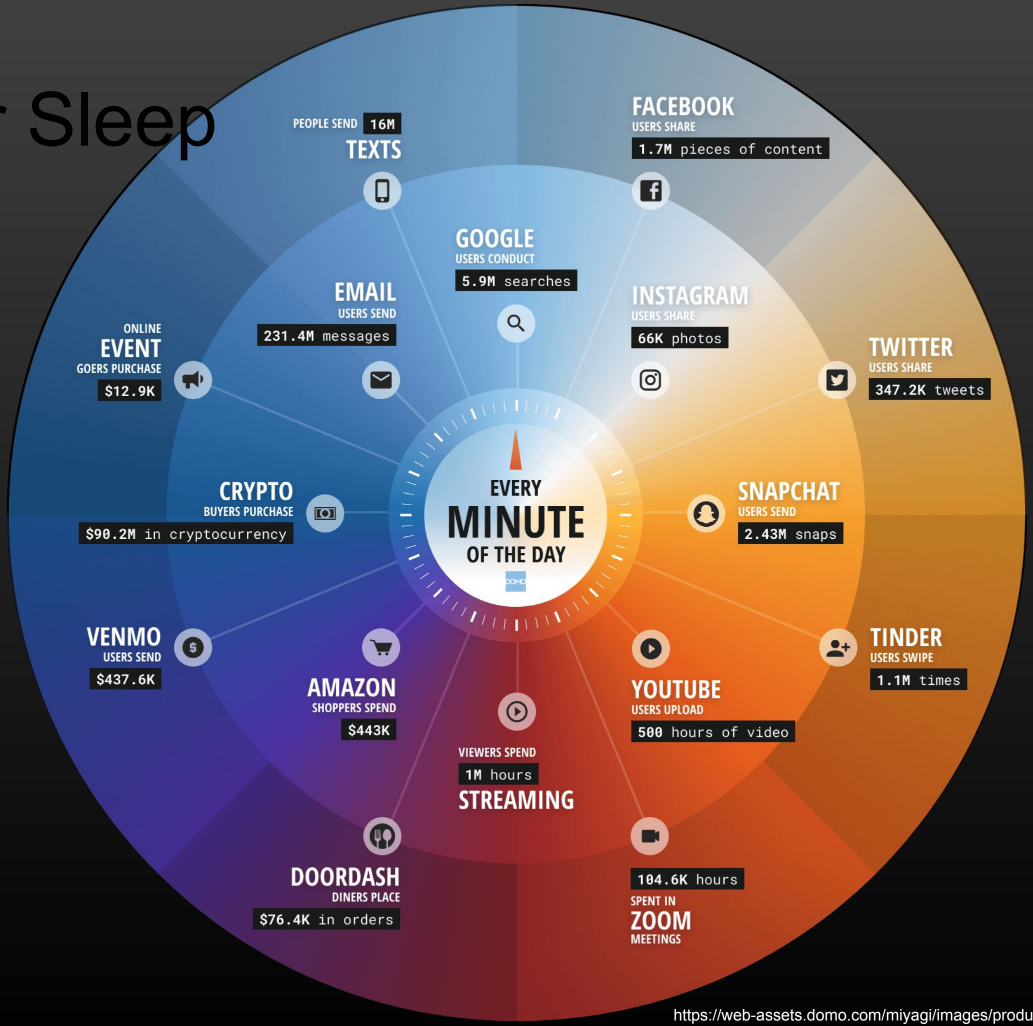
An Overview of Continuous Querying in (Modern) Data System

Riccardo Tommasini, INSA Lyon, CNRS Liris (France)
Angela Bonifati, Lyon 1 University, CNRS Liris, IUF (France)

Santiago Chile
SIGMOD PODS 2024

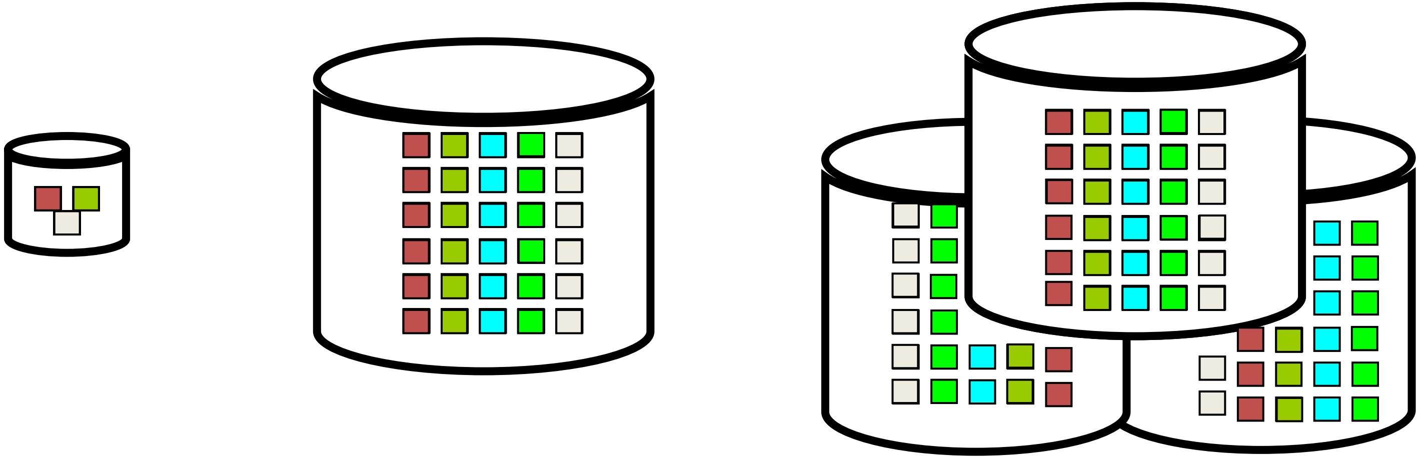
Slides at

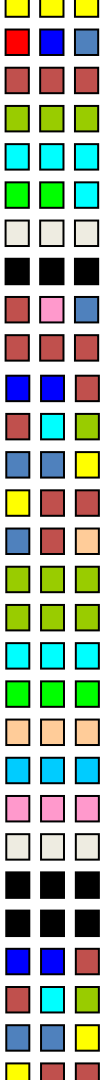
Data Never Sleep



https://web-assets.domo.com/miyagi/images/product/product-feature-22-data-never-sleeps-10.png

Data

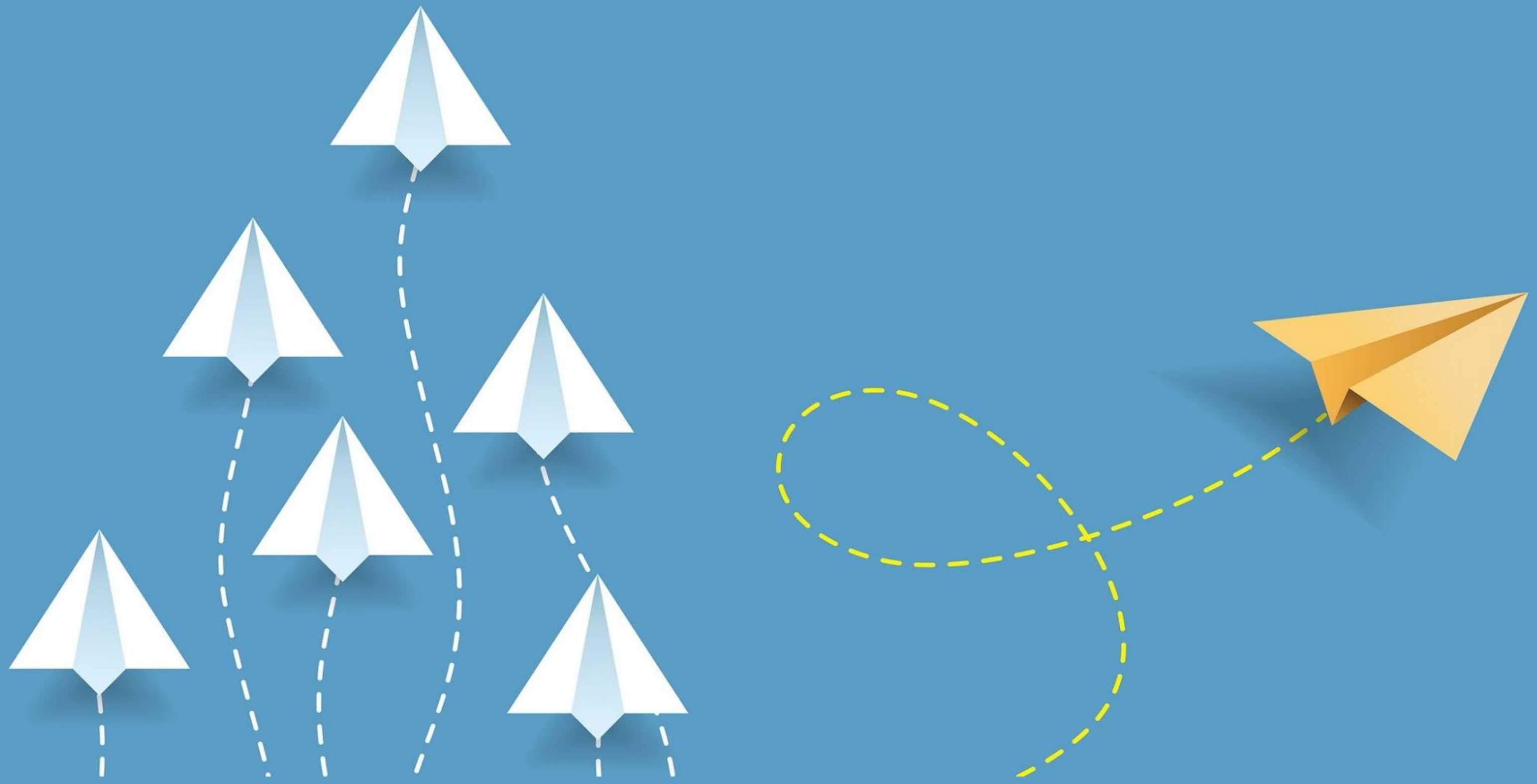




credits to Sara Robinson, Felipe Hoffa

The Burden of Unboundedness

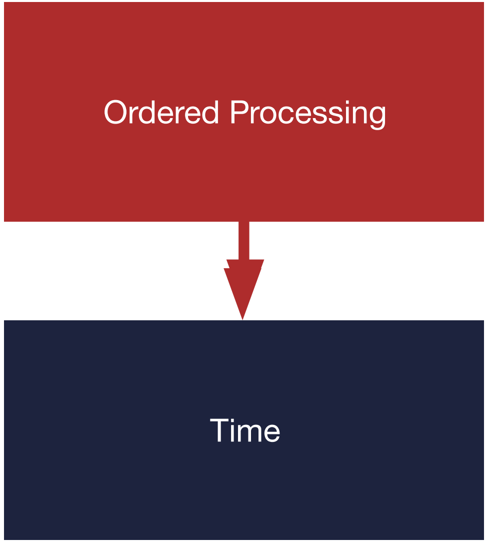
…requires a paradigm shift



Unboundedness

The Burden of

- A program expressed over an **infinite** input may **not terminate**
- Unless we can **reformulate** the notion of "**termination**"
- From an **infinite input** we may observe an **infinite output**
- We still need a way to **determine** what **part** of the **input** maps to the **output**

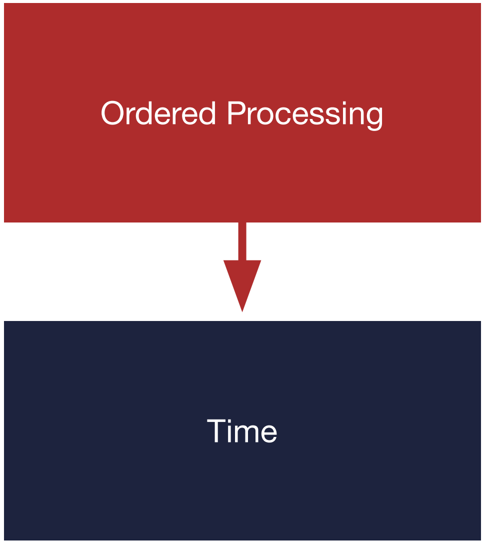


Babcock, Brian, et al. "Models and issues in data stream systems." Proceedings of the twenty-first ACM SIGMOD 2002.

Unboundedness

The Burden of

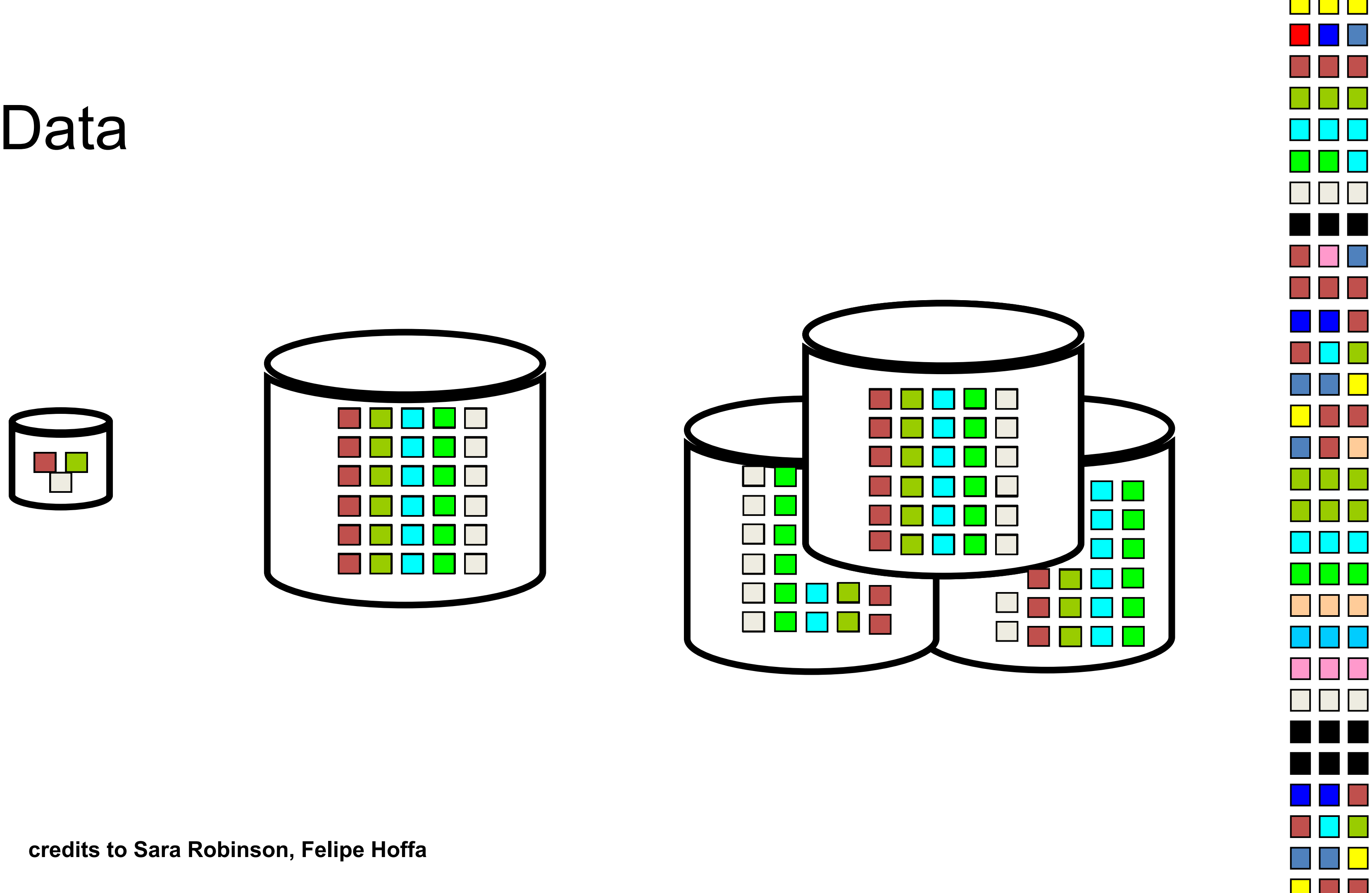
- An **infinite dataset** poses the problem of where to start **computing**
- **Recency*** is a form of **temporality** that enables also **reactivity**
- **Temporality**** may assume other forms
  - About Time (Temporal Data)
  - Through Time (Versioned Data)
  - In-Time (Streaming Data)



*Akidau, Tyler, et al. "The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing." PVLDB (2015).

** Polleres, Axel, et al. "How does knowledge evolve in open knowledge graphs?." *Transactions on Graph Data and Knowledge* 1.1 (2023): 11-1.

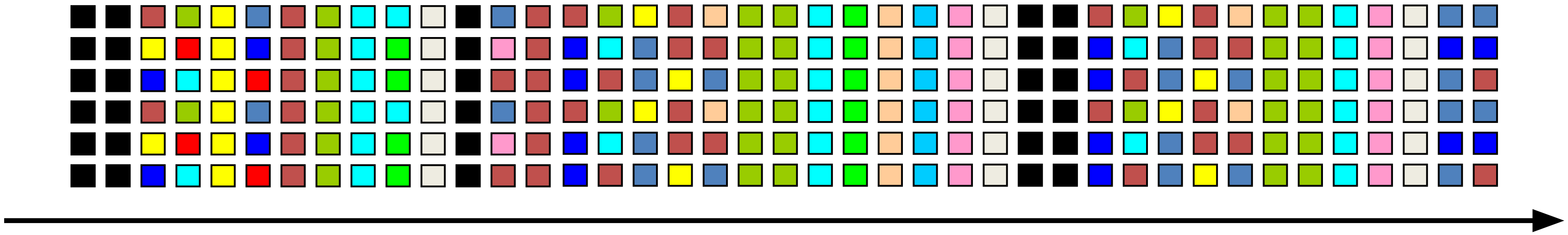
Data



credits to Sara Robinson, Felipe Hoffa

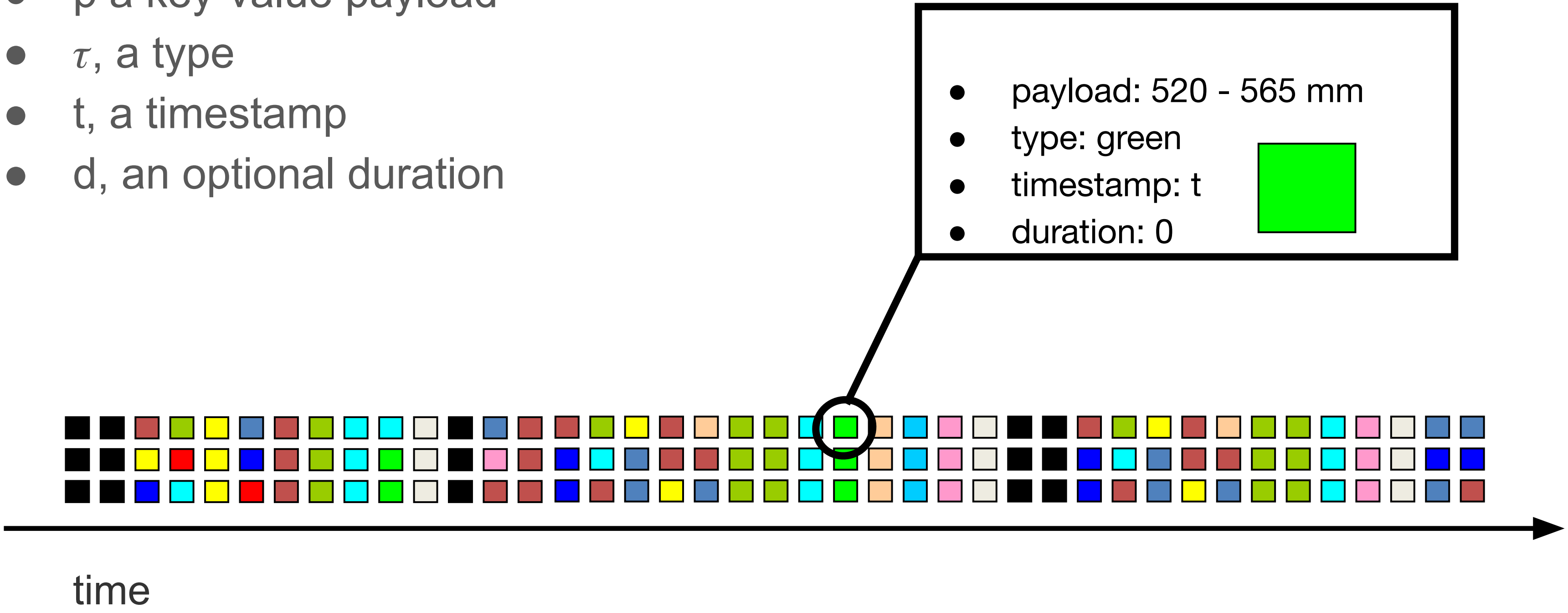
What is a Stream?

An unbounded partially ordered sequence of data points

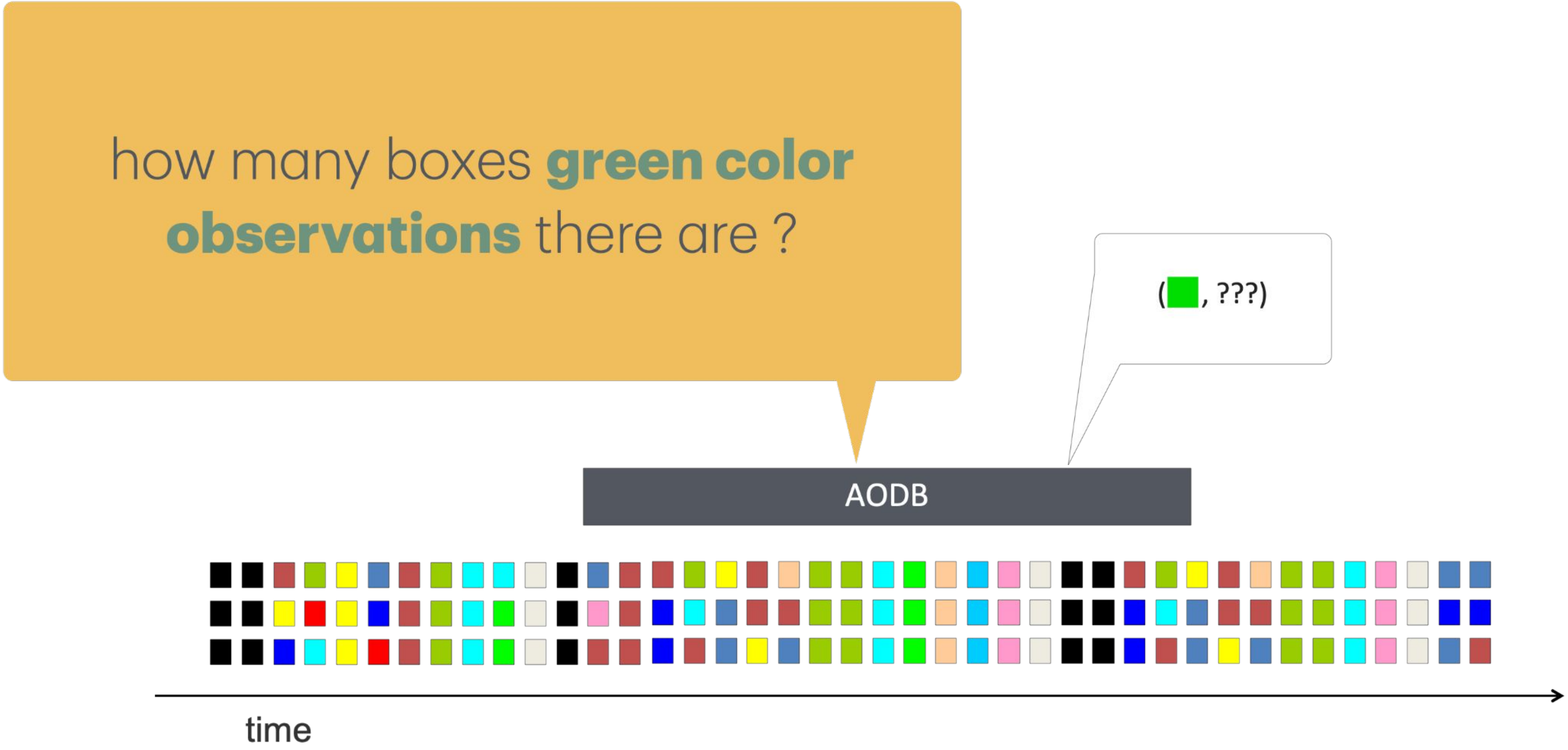


What is an Event?

- Event: time-based notification of a known fact defined by
- p a key-value payload
- τ, a type
- t, a timestamp
- d, an optional duration



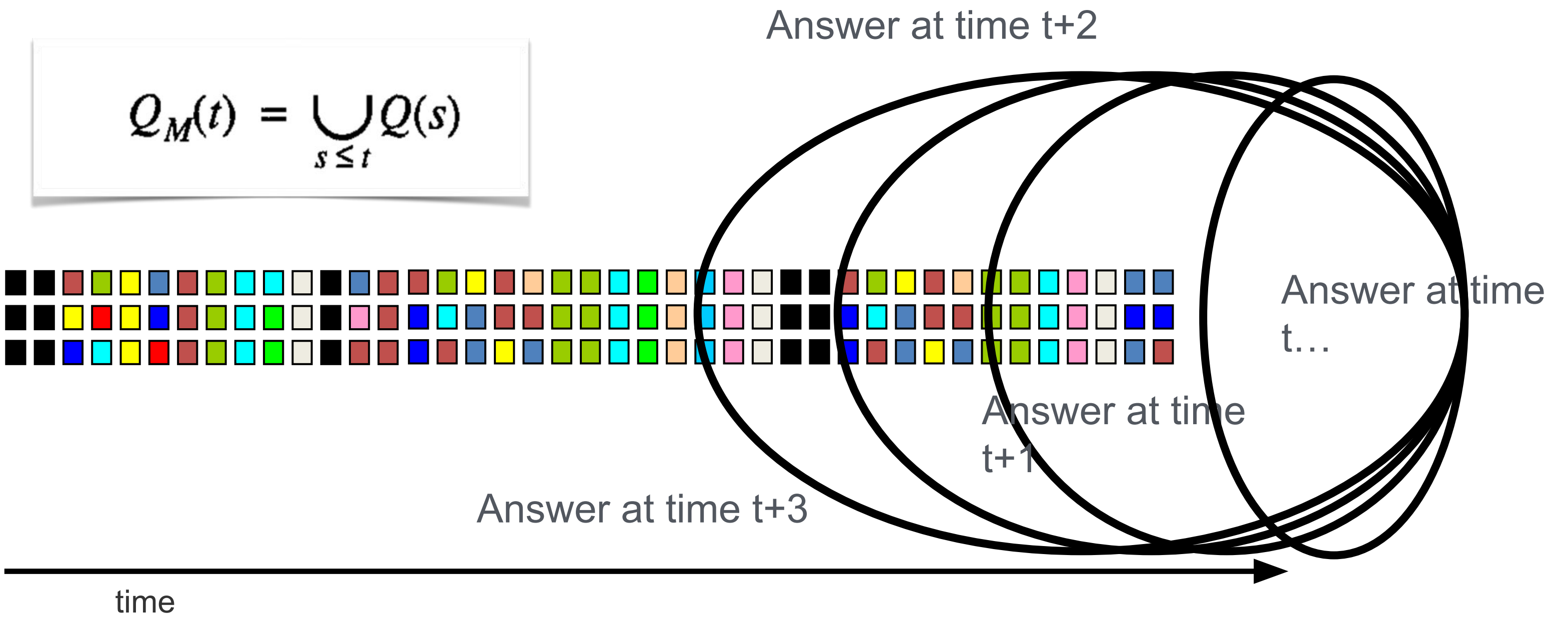
Continuous Queries on append only databases



Continuous Queries

on append only databases

$$Q_M(t) = \bigcup_{s \leq t} Q(s)$$

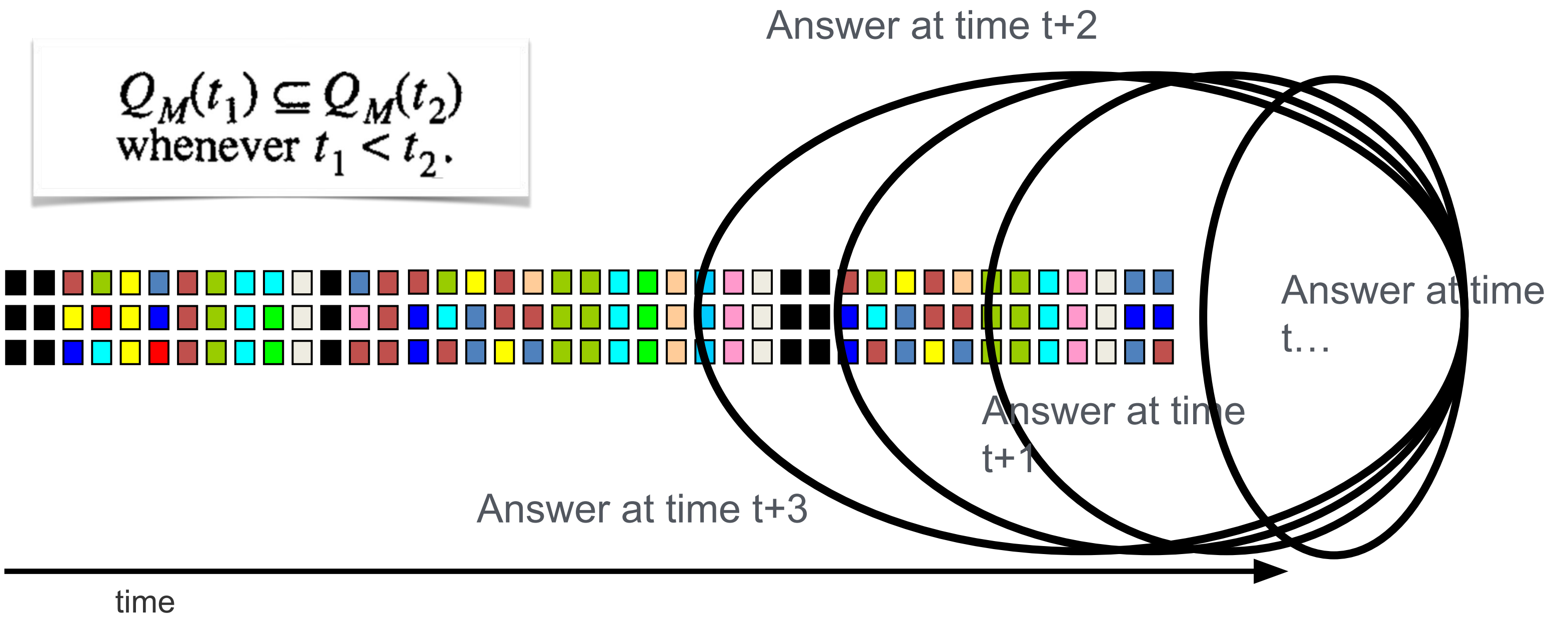


Terry, Douglas, et al. "Continuous queries over append-only databases." *Acm Sigmod Record* 21.2 (1992): 321-330.

Continuous Queries (Monotonic)

on append only databases

$$Q_M(t_1) \subseteq Q_M(t_2)$$
whenever $t_1 < t_2$.



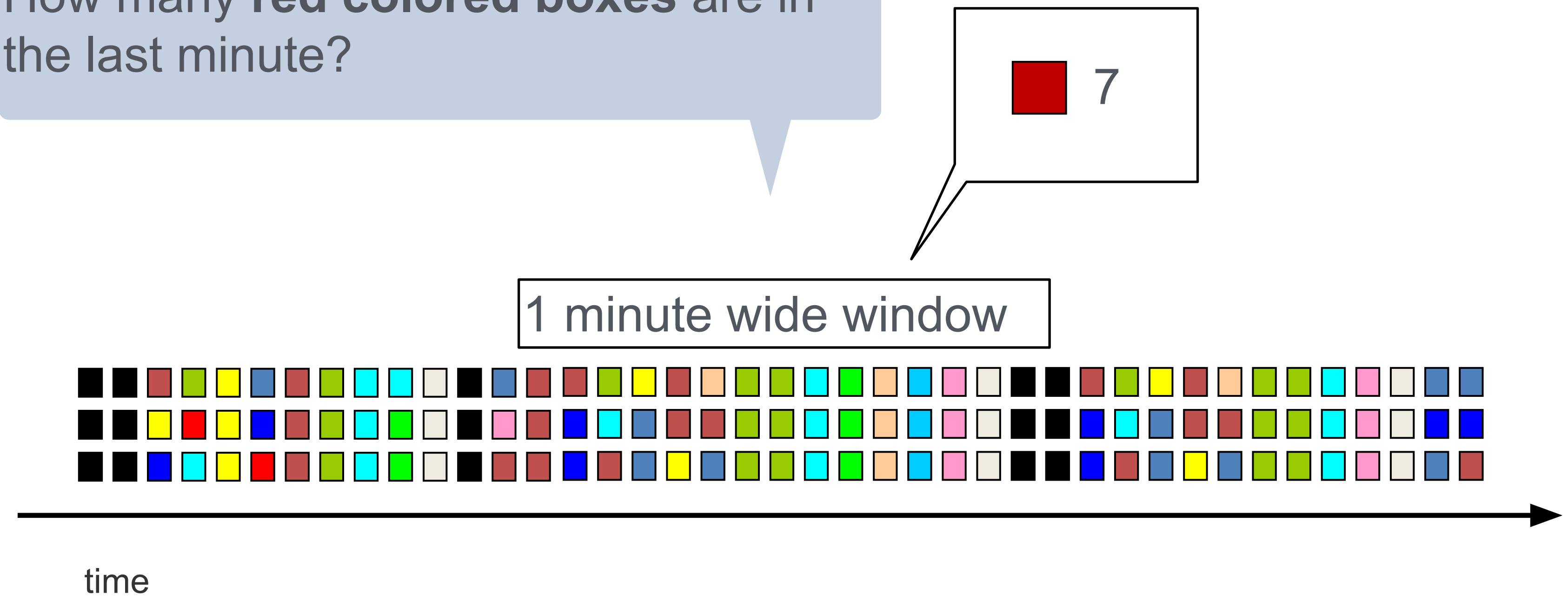
Terry, Douglas, et al. "Continuous queries over append-only databases." *Acm Sigmod Record* 21.2 (1992): 321-330.

Monotonicity is not enough!

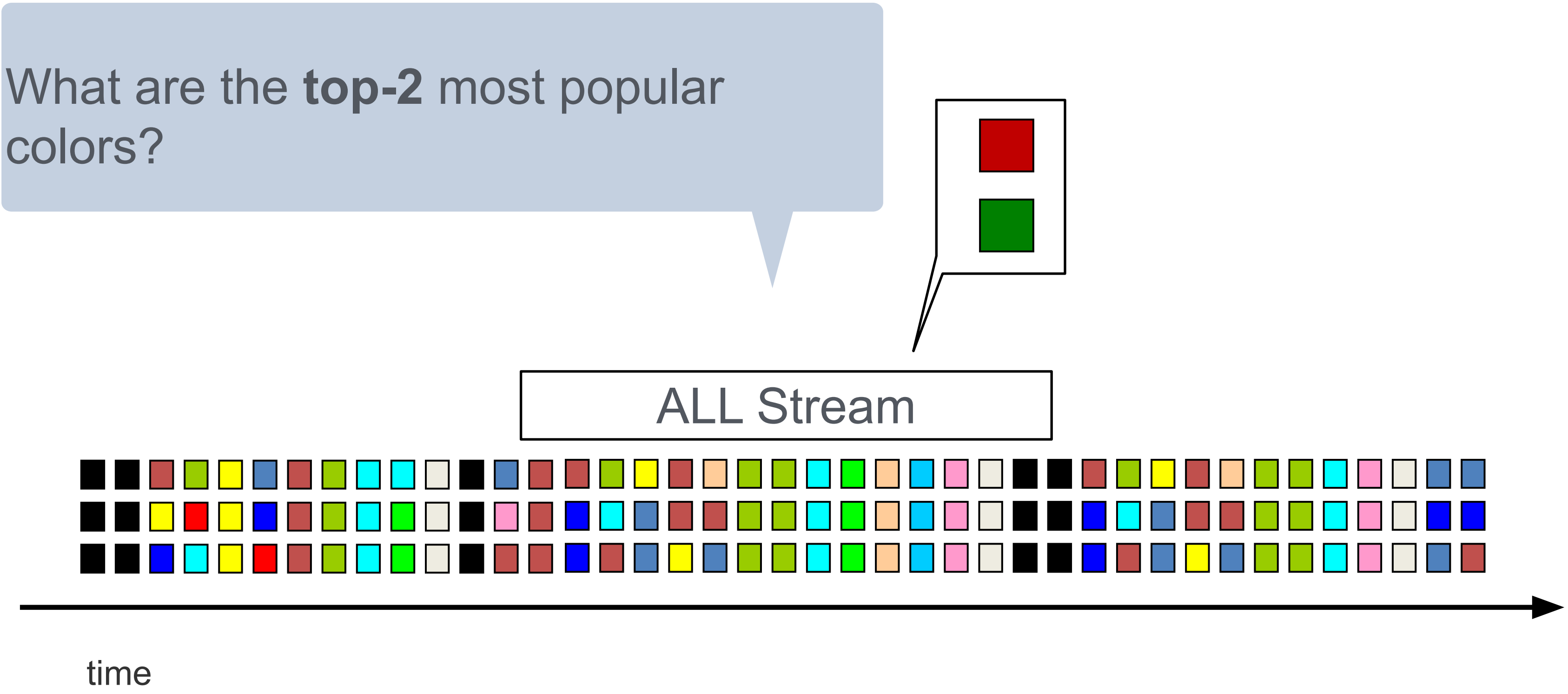
Can we achieve more?

Continuous Aggregation

How many **red colored boxes** are in the last minute?

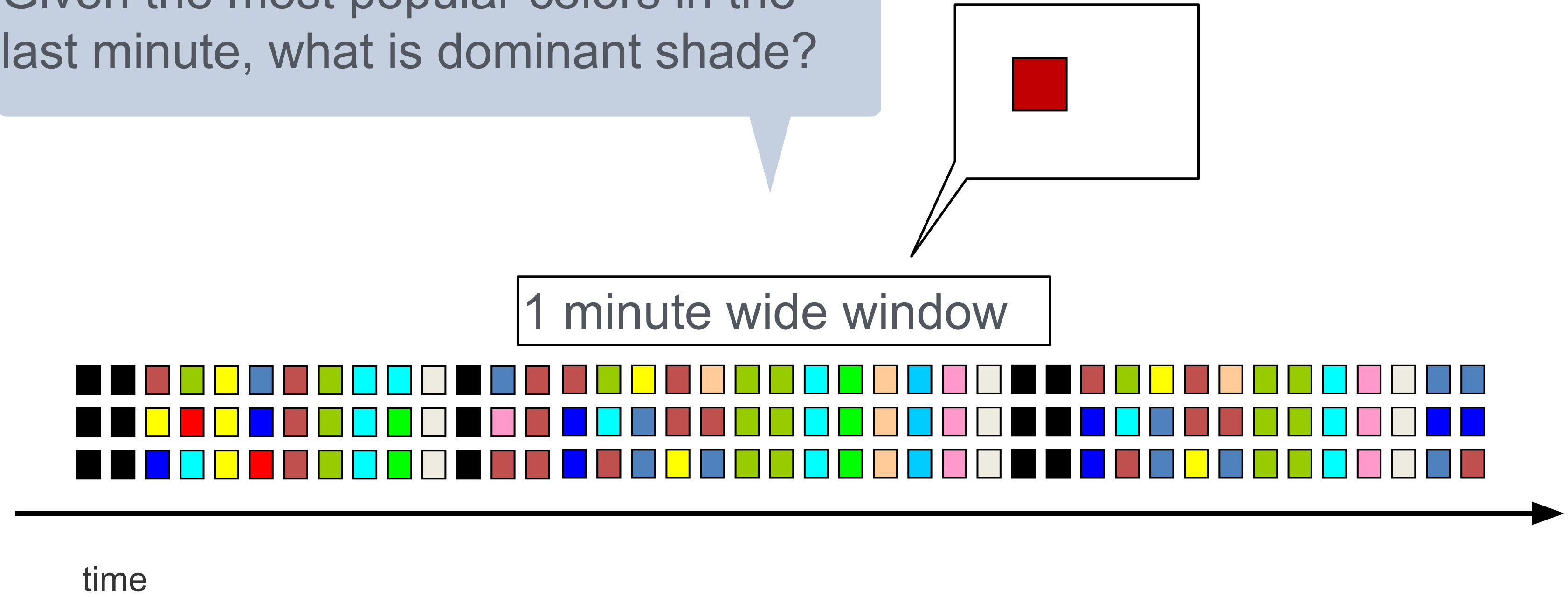


Top-K



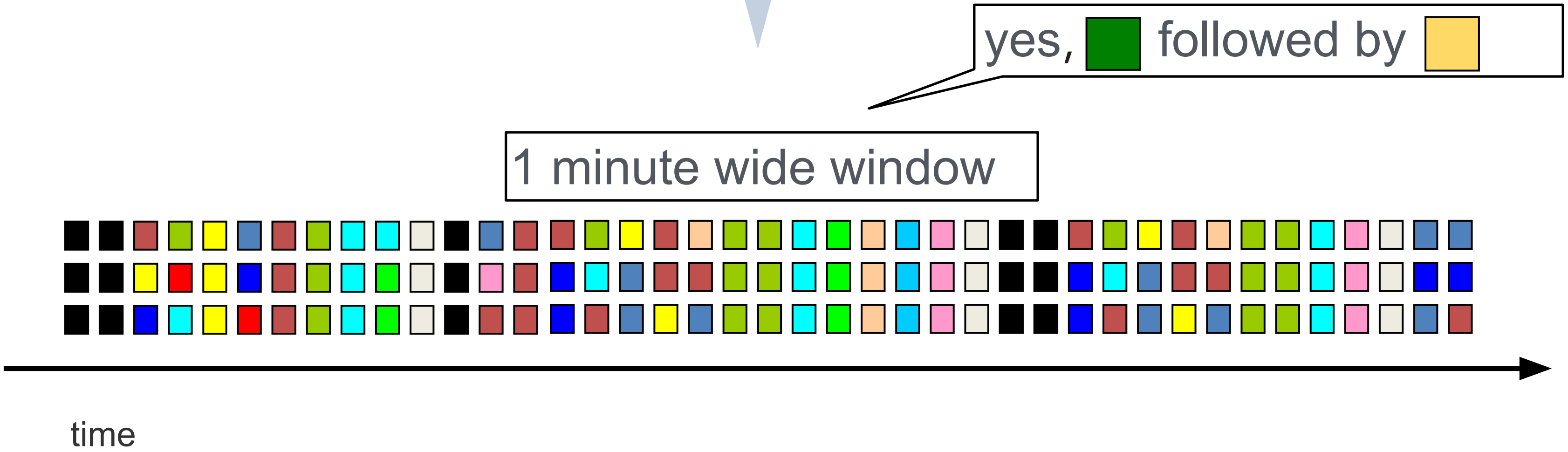
Skyline Continuous Queries

Given the most popular colors in the last minute, what is dominant shade?



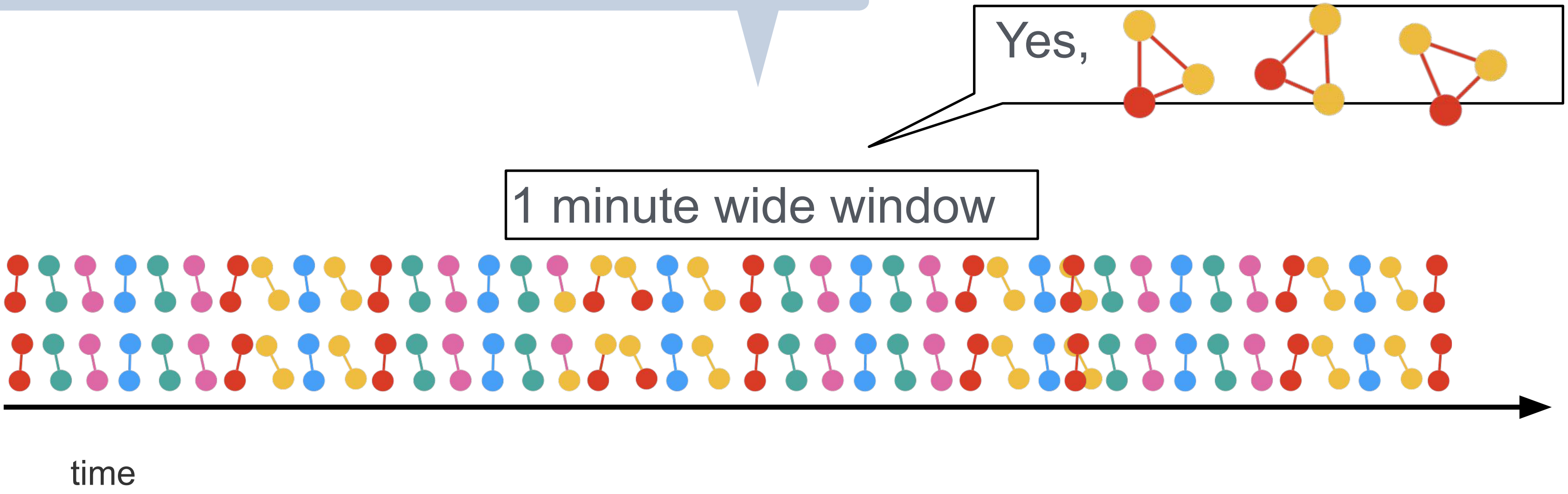
Complex Event Recognition

Is there a **primary cool** colour followed by a **secondary warm** one in the last minute?

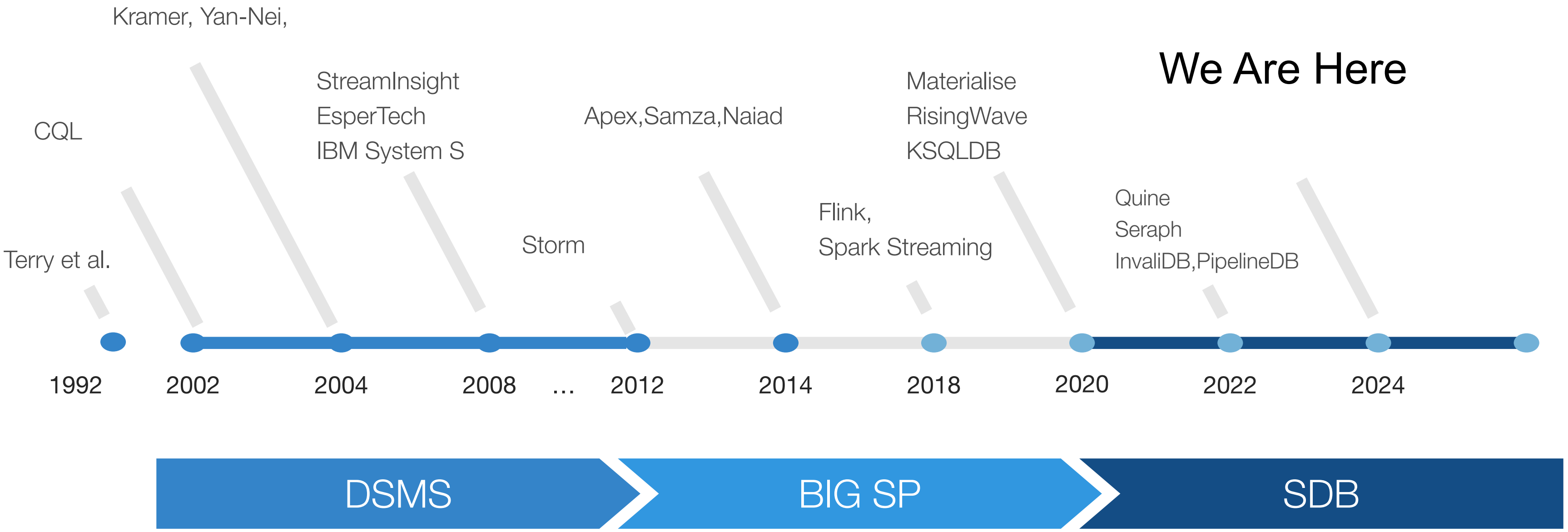


Graph Stream Processing

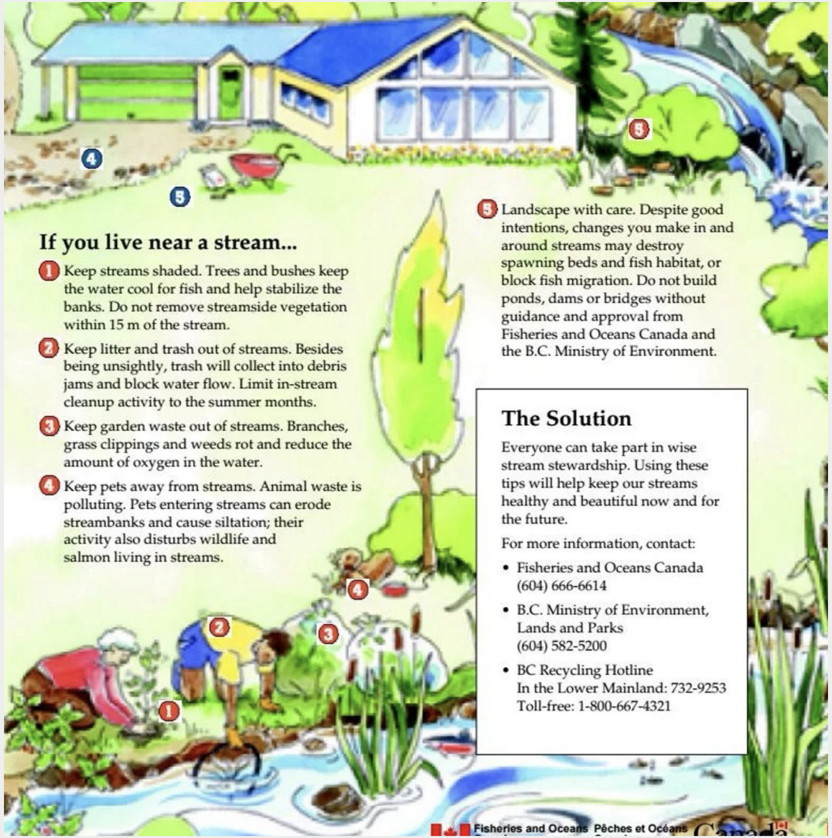
Are there any **triangles** with warm colours are in the last minute?



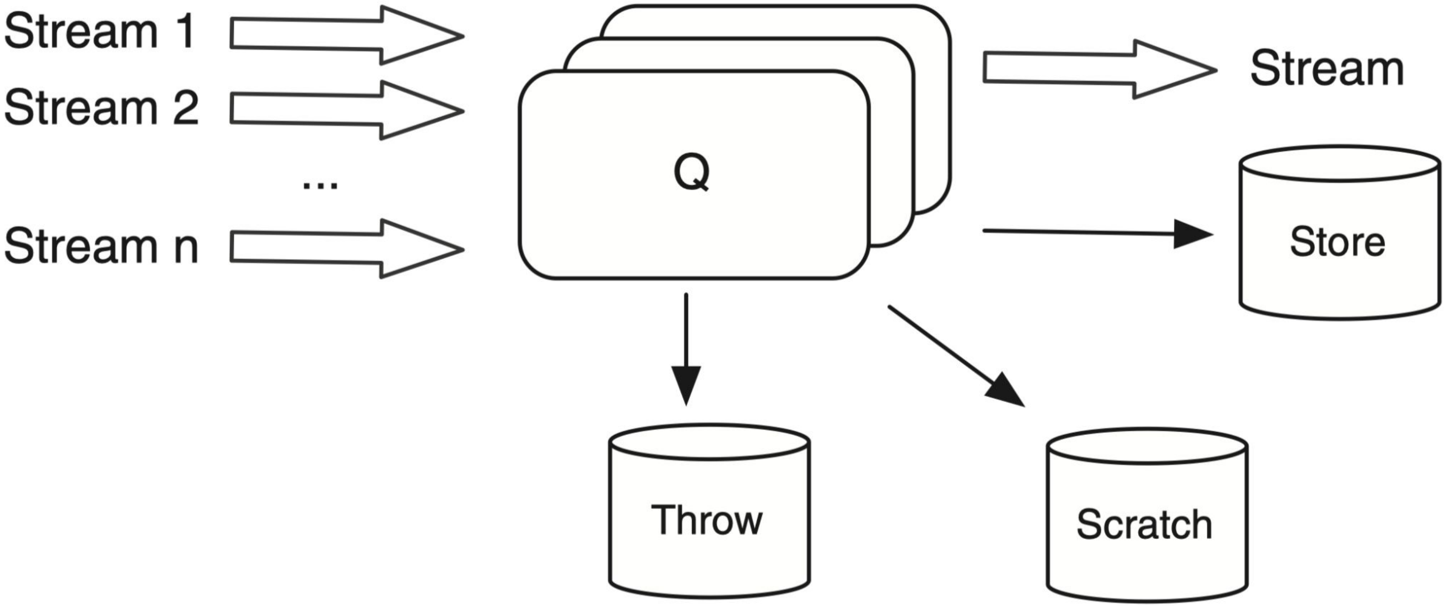
Historical Notes



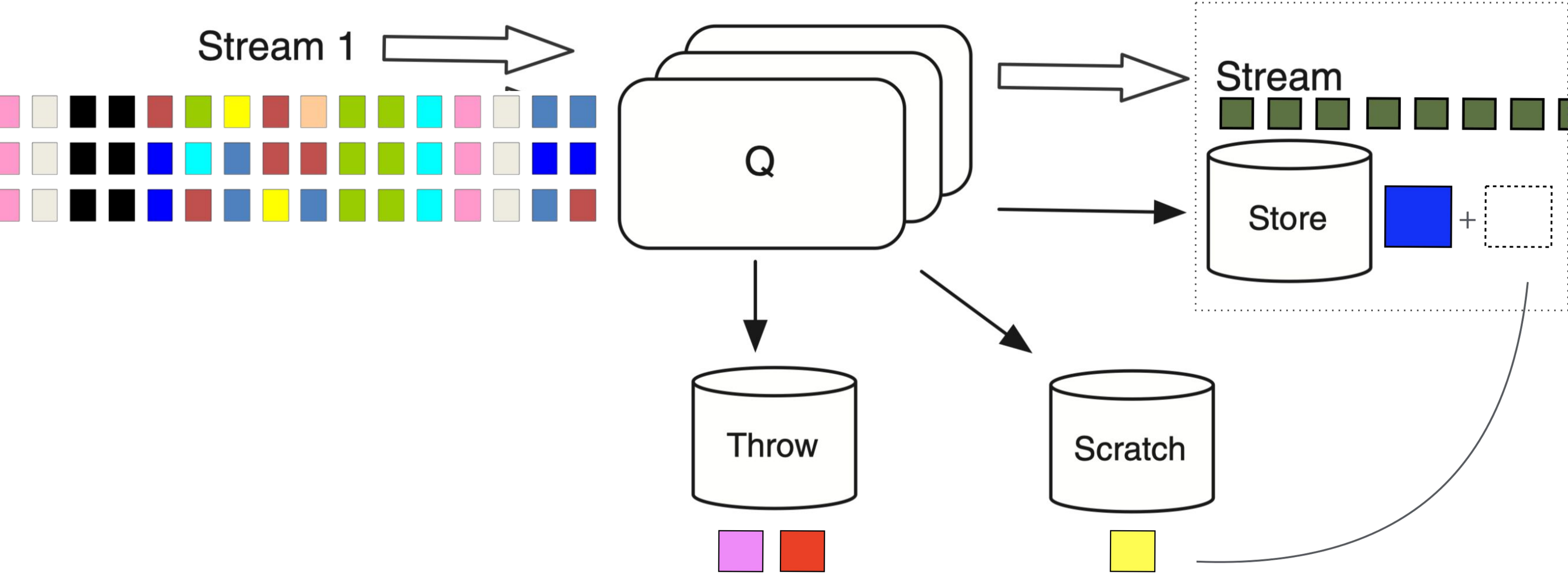
Data Stream Management Systems



CQ@DBMS



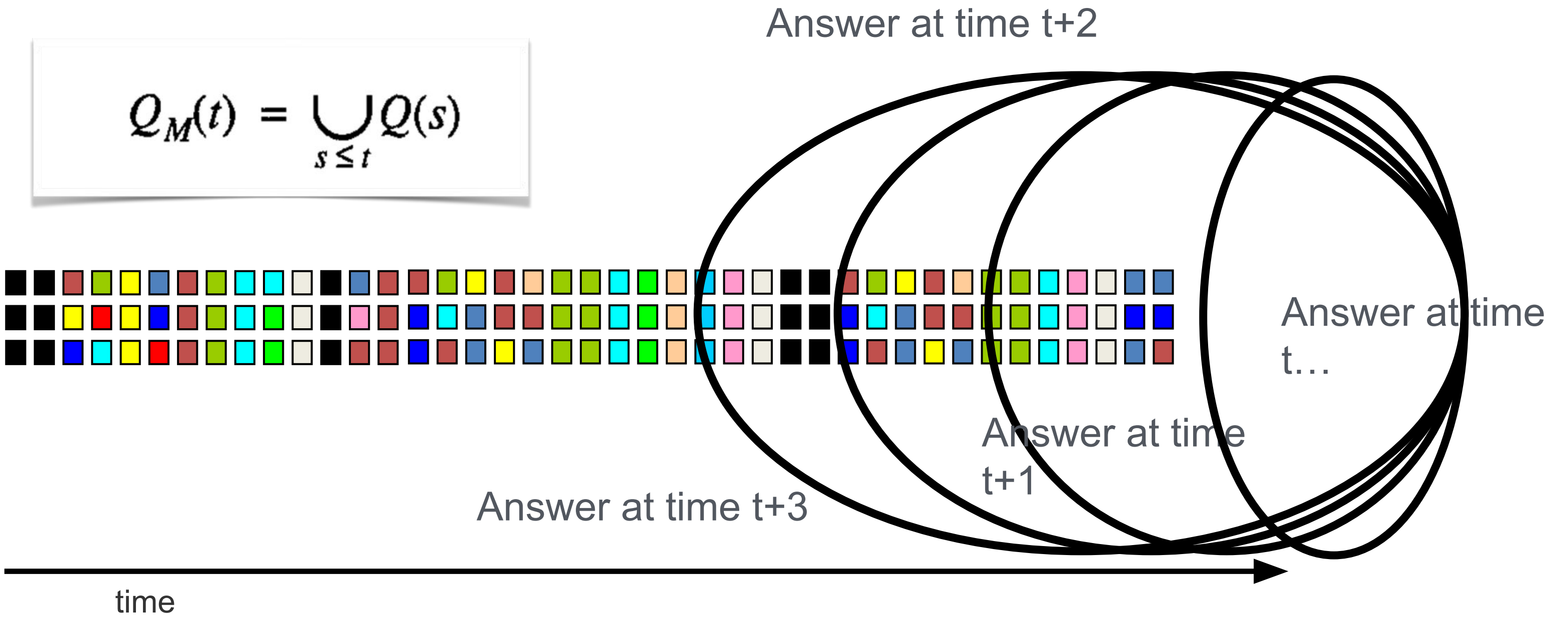
CQ@DBMS



Continuous Queries

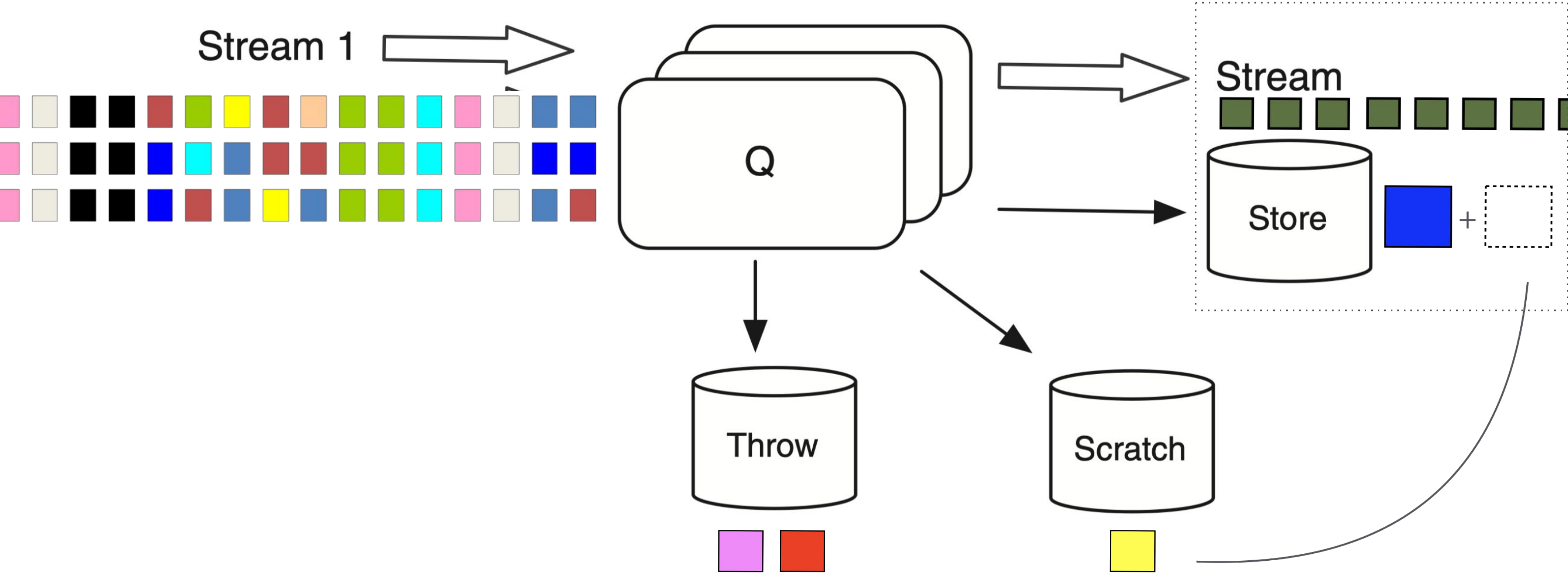
on append only databases

$$Q_M(t) = \bigcup_{s \leq t} Q(s)$$



Terry, Douglas, et al. "Continuous queries over append-only databases." *Acm Sigmod Record* 21.2 (1992): 321-330.

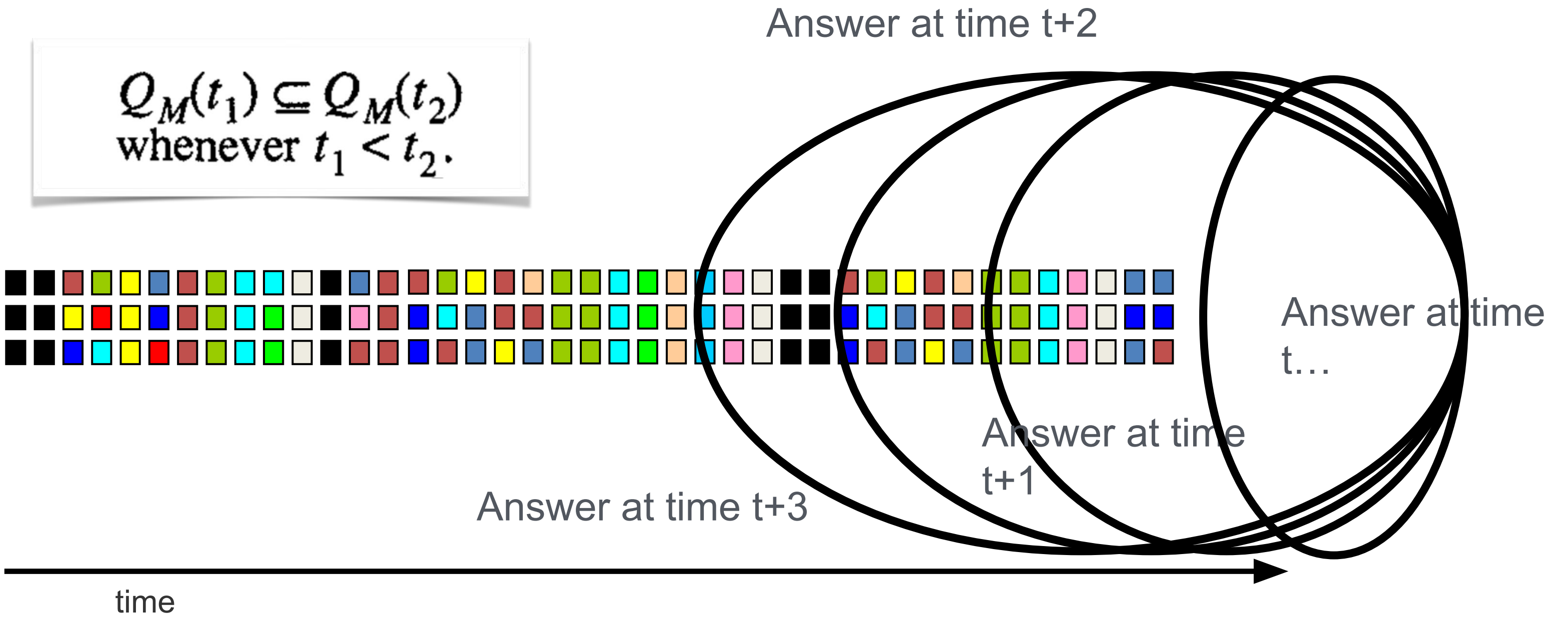
CQ@DBMS



Continuous Queries (Monotonic)

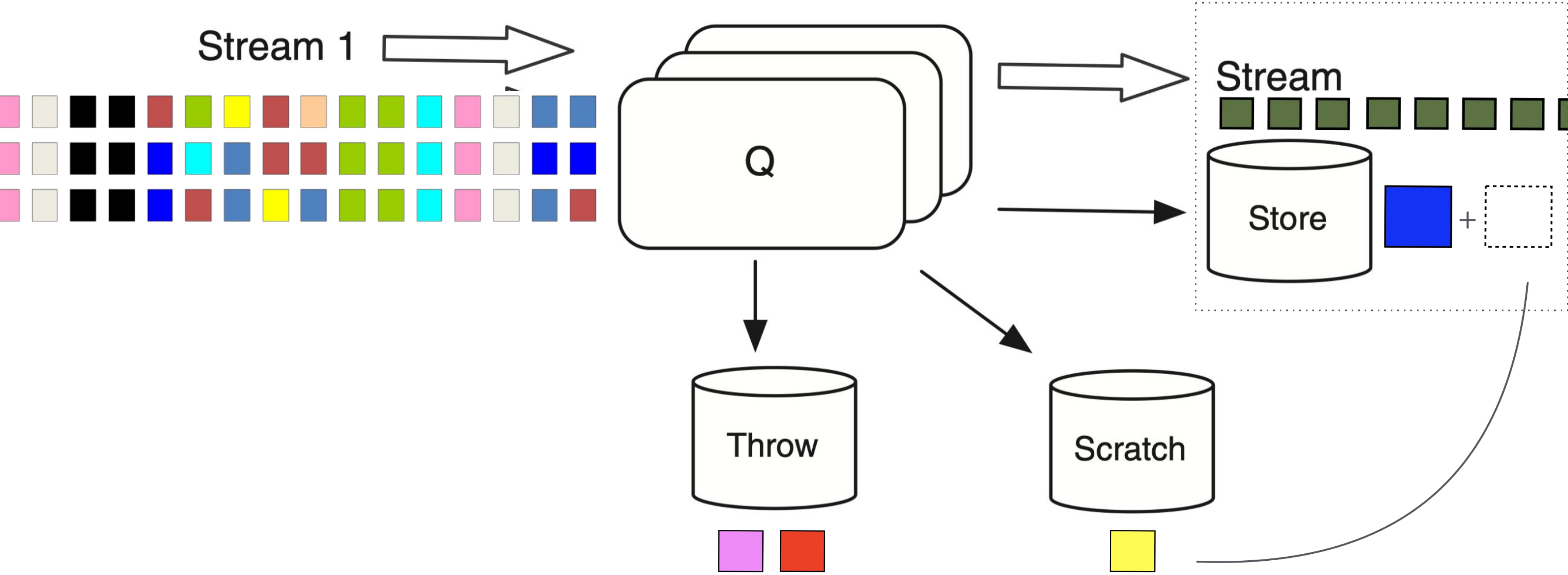
on append only databases

$$Q_M(t_1) \subseteq Q_M(t_2)$$
whenever $t_1 < t_2$.



Terry, Douglas, et al. "Continuous queries over append-only databases." *Acm Sigmod Record* 21.2 (1992): 321-330.

CQ@DBMS



Monotonicity Explained

- **Monotonic** queries produce an **append-only** output stream and therefore do not incur deletions from their answer set.
    - A query is monotonic if for two instances of the database S1 and S2 such that S1⊆ S2 then Q(S1) ⊆ Q(S2), where Q(Si) denotes the set of tuples that satisfy Q when applied to the instance Si.
- Only stateless **operators** over infinite streams (projection, selection, time-wise union, and distributive aggregates) can give rise to **monotonic** queries.
- Hence, non **monotonicity** is caused by so called **blocking** operators, i.e., operators that need to see the whole stream to report

Barbará, Daniel. "The characterization of continuous queries." International Journal of Cooperative Information Systems 8.04 (1999): 295-323.

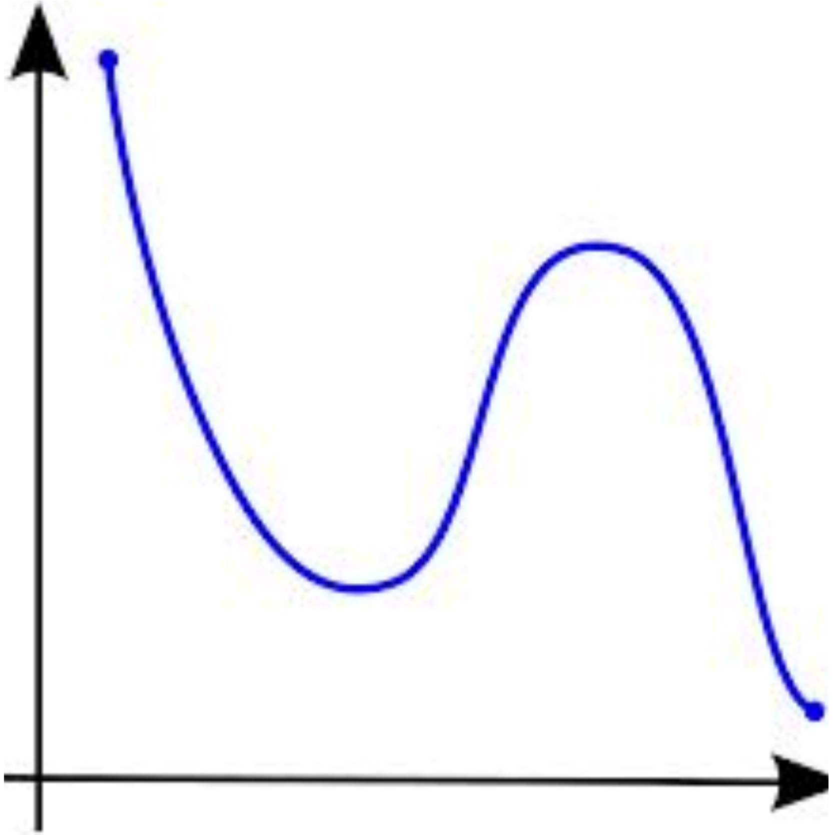
Non-monotonicity

Causes of

The concept has taken many names

- stateless/stateful **functions**
- blocking/non blocking **operators**
- event-level/stream-level **semantics**

All notions share the intuition of "memory", how far do I have to know to answer?



Interval Strategy

- If there are no deletions in the database, the answer to a continuous, non-monotonic query Q can be approximated as Q = P – N
- P(kτ) - P((k - 1)τ) at every interval [(k-1)τ, kτ)
- In the worst case, this approximation gives a superset of set of data items in the right hand side of the equation below

$$Q(k\tau) - Q((k-1)\tau) = \bigcup_{t_0 \leq s \leq k\tau} Q(s) - \bigcup_{t_0 \leq s \leq (k-1)\tau} Q(s) .$$

Barbará, Daniel. "The characterization of continuous queries." International Journal of Cooperative Information Systems 8.04 (1999): 295-323.

Fixed-Structure Rewriting

- A rewriting Q = P - N is fixed structure if the following conditions are true:
- P is monotonic
- Interval strategy holds while
  - P holding within k1 and Inf, N holding within k2,k3, with k1 < k2

$$P(k\tau) - P((k-1)\tau) = \bigcup_{t_0 \le s \le k\tau} Q(s) - \bigcup_{t_0 \le s \le (k-1)\tau} Q(s) \,.$$

Barbará, Daniel. "The characterization of continuous queries." International Journal of Cooperative Information Systems 8.04 (1999): 295-323.

Admitting Deletions (append-only dbs)

- A query Q is deletion sensitive iff for S1 and S2 such that S2 ⊆ Si, then there exists an item D1 ∈ S1, D1 ∈ S2 such that one of the following is true:

- D1 ∈ Q(S1)(t) AND D1 ∉ Q(S2)(t)
- D1 ∉ Q(Si)(t) AND D1 ∈ Q(S2)(t)

The semantics evolves as

$$P(k\tau) - P((k-1)\tau) = \bigcup_{t_0 \le s \le k\tau} Q(s) - \bigcup_{t_0 \le s \le (k-1)\tau} Q(s) - D_\tau^k .$$

Barbará, Daniel. "The characterization of continuous queries." International Journal of Cooperative Information Systems (1999)

Fix the Size of the Answer

Fixing the Size of the Answer

Idea: summarize the characteristics of a stream reducing the memory footprint

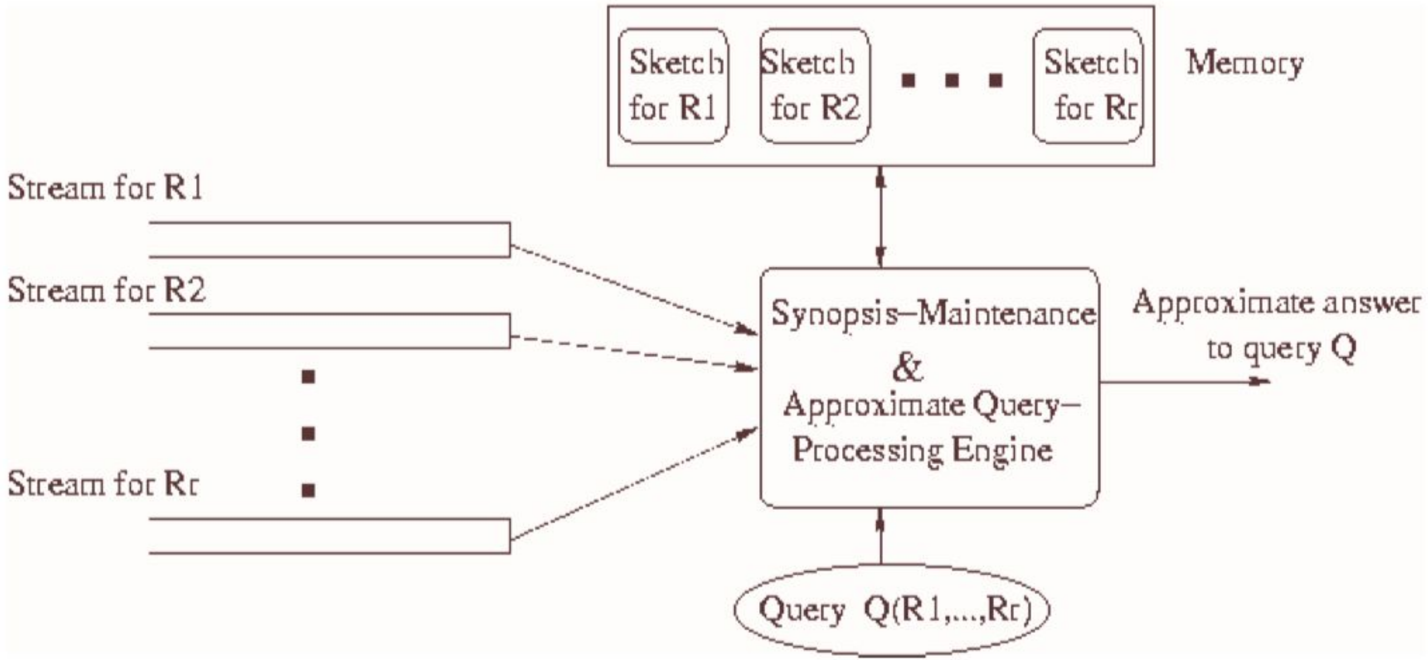
A **histograms** summarize a dataset by grouping the data values into buckets and compute for each bucket a set of summary statistics

Wavelet transform the data to represent the most significant features in a frequency domain

Sketches, data structures or algorithms that provide approximate answers to given queries.

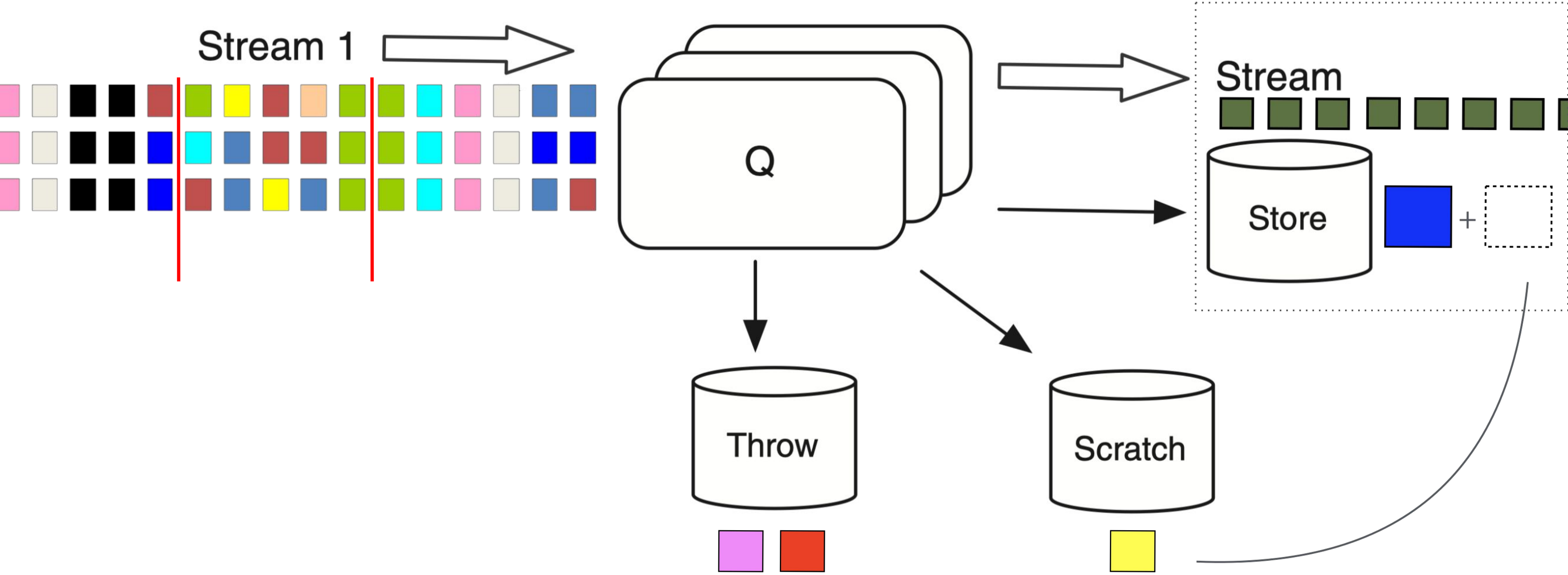
Babcock, Brian, et al. "Models and issues in data stream systems." *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 2002.

Synopsis-Based DSMS



Modify the Input Stream

CQ+Pubctuations@DBMS



Punctuations

- A punctuation is a **predicate** on stream elements that **must** evaluate to **false** for **every element** following the punctuation (Boolean functions).

- In the **original** papers, they are presented as a simple **grammar**
    - [*,+,value,[ ], range]
    - A tuple matches the punctuation if each of its attributes matches the corresponding pattern

- A punctuated stream is a data stream that contains additional information describing a (possibly empty) subset of data over the domain of the stream

Punctuations Correctness

- A **punctuated stream** S is **grammatical** if for all i, for all j > i, if the punctuation $p \in S[i]$ □ and the tuple $t \in S[i \rightarrow j]$, t does not match p.

- **Safety**: That is, we never emit output unless we can be sure it will not conflict with any later input.

- **Completeness**: we always emit an output if it will necessarily be generated by the relational operator under any additional input, including no input.

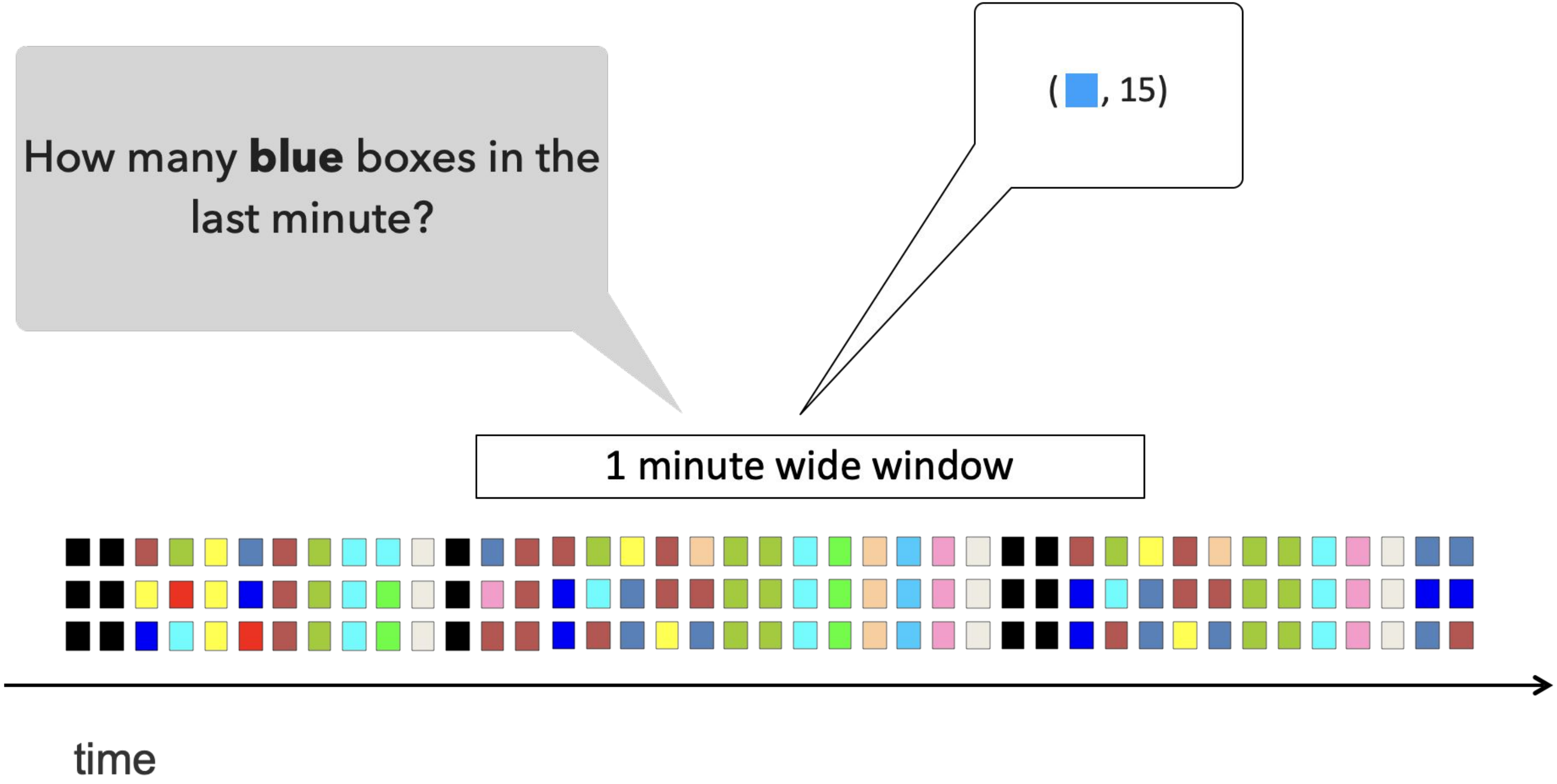
Advantages/Caveats of Punctuations

- Simple to implement
- All **result** data items for a query will eventually be output.
- **Cleanses**. Every data item that resides in the state for any operator in the query **will** eventually be removed.

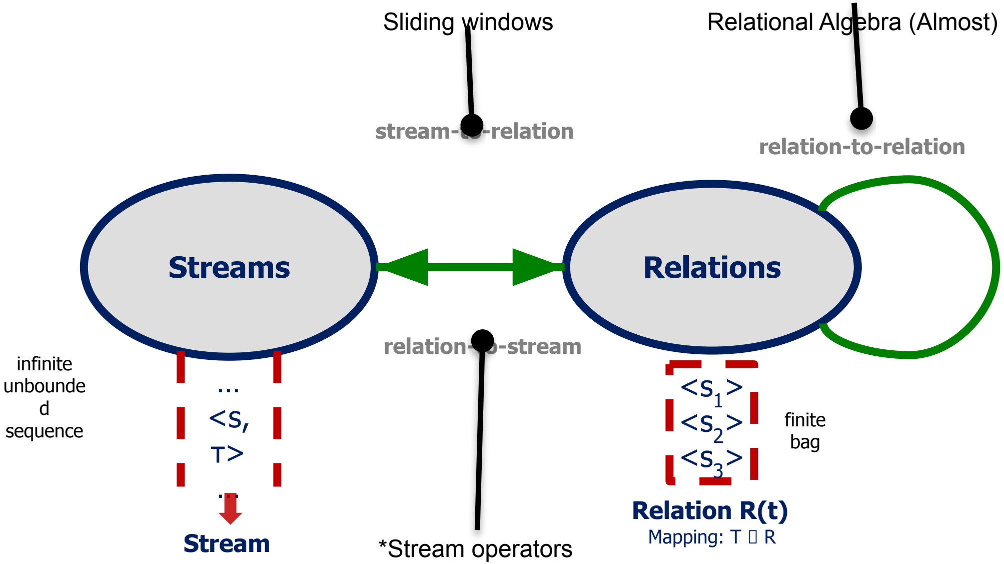
- **Who is going to emit the punctuation?**
  - **Sources?**
  - **Other operators?**
- **Different data models have different punctuation semantics**
- **We still do not know what queries benefits from a given punctuation**

Modify the Output Stream a.k.a. Modify the Query

Window-Based Continuous Querying

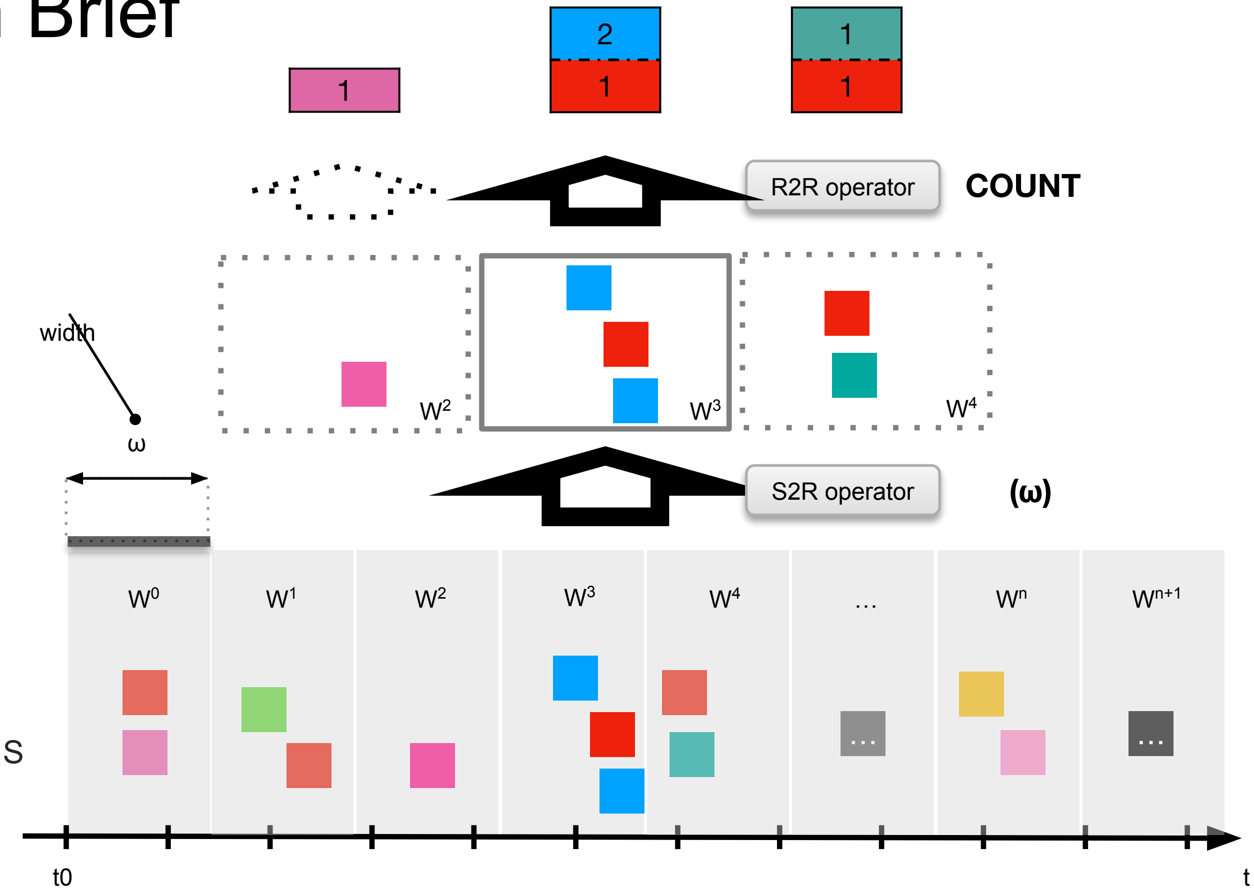


Continuous Query Language



Arasu, Arvind, Shivnath Babu, and Jennifer Widom. "The CQL continuous query language: semantic foundations and query execution." The VLDB Journal 15 (2006):.

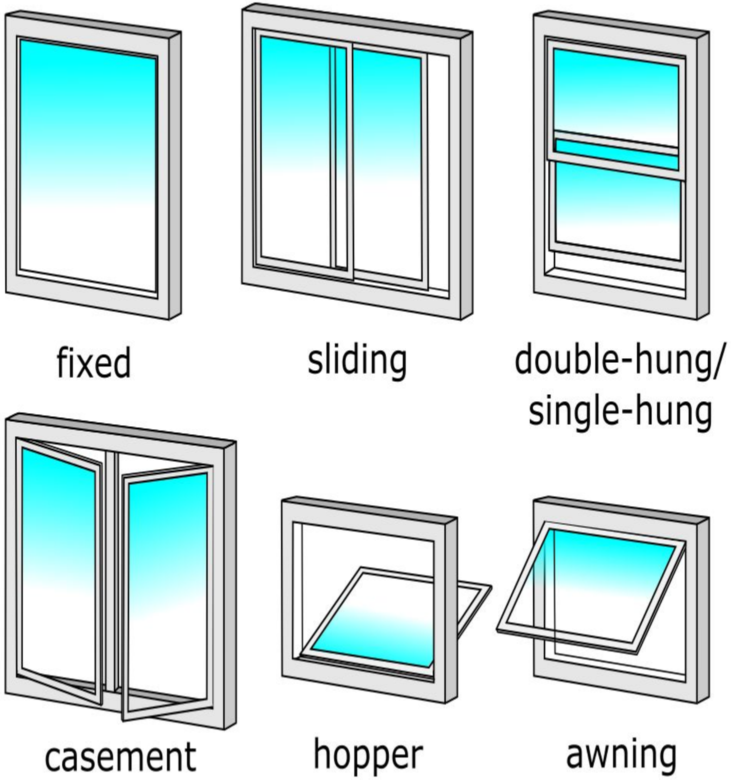
CQL In Brief



1
2
1
1
1
R2R operator
COUNT
W2
W3
W4
width
ω
S2R operator
(ω)
W0
W1
W2
W3
W4
…
Wn
Wn+1
S
…
…
t0
t

Types of Windows

Window Types	Parameters
Sliding	width
Hopping	Width, slide
Tumbling	Width == slides
Session	Inactivity
…	…



Verwiebe, Juliane, et al. "Survey of window types for aggregation in stream processing systems." *The VLDB Journal* 32.5 (2023): 985-1011.

Windows and Monotonicity

- **Weakest non-monotonic** queries do not store state and do not reorder incoming tuples during processing; tuples are either dropped or appended to the output stream immediately.
  - Projection and selection over a single sliding window are weakest non-monotonic
- **Weak non-monotonic** have the property that the expiration time of each result tuple can be determined without generating negative tuples on the output stream.
  - join, duplicate elimination, and groupby.
- **Strict non-monotonic** queries have the property that at least some of their results expire at unpredictable times.
  - Negation over two windows is one example.

Golab, Lukasz, and M. Tamer Özsu. "Update-pattern-aware modeling and processing of continuous queries." *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. 2005.

Advantages/Caveats of Windows

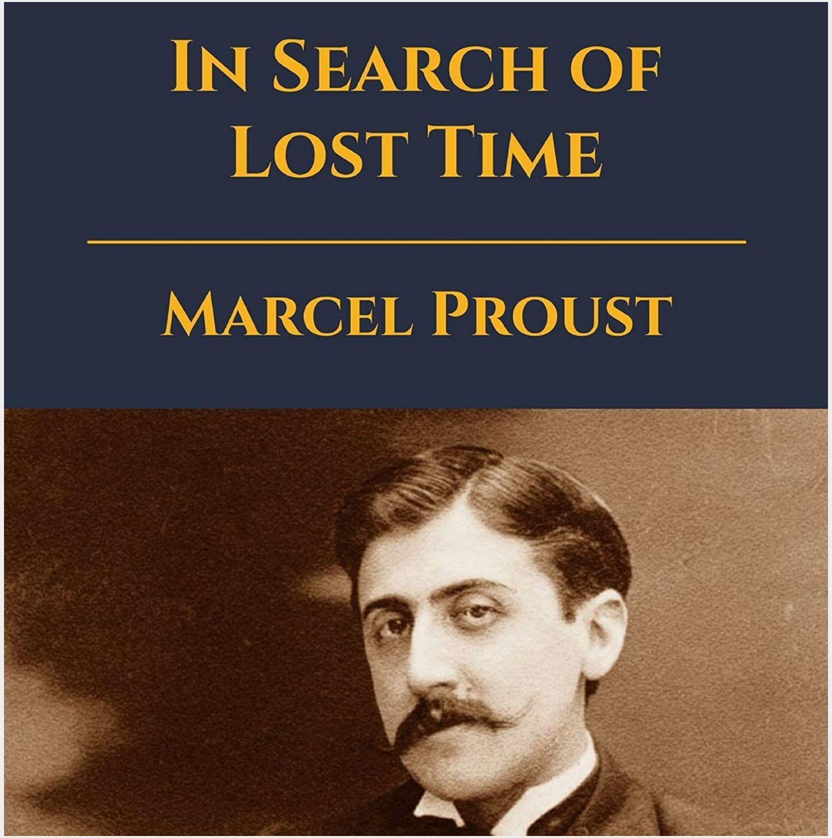
- SP over sliding Windows are very easy to optimise
- Advantages for parallelised computations
- Enable efficient aggregation and possibly synopsis [paper]

- Users need to know the data semantics
  - Which may be hard given their non-monotonic behaviour
- Out-of-order processing requires sophisticated strategies
- Currently are system-dependent, and harm query portability

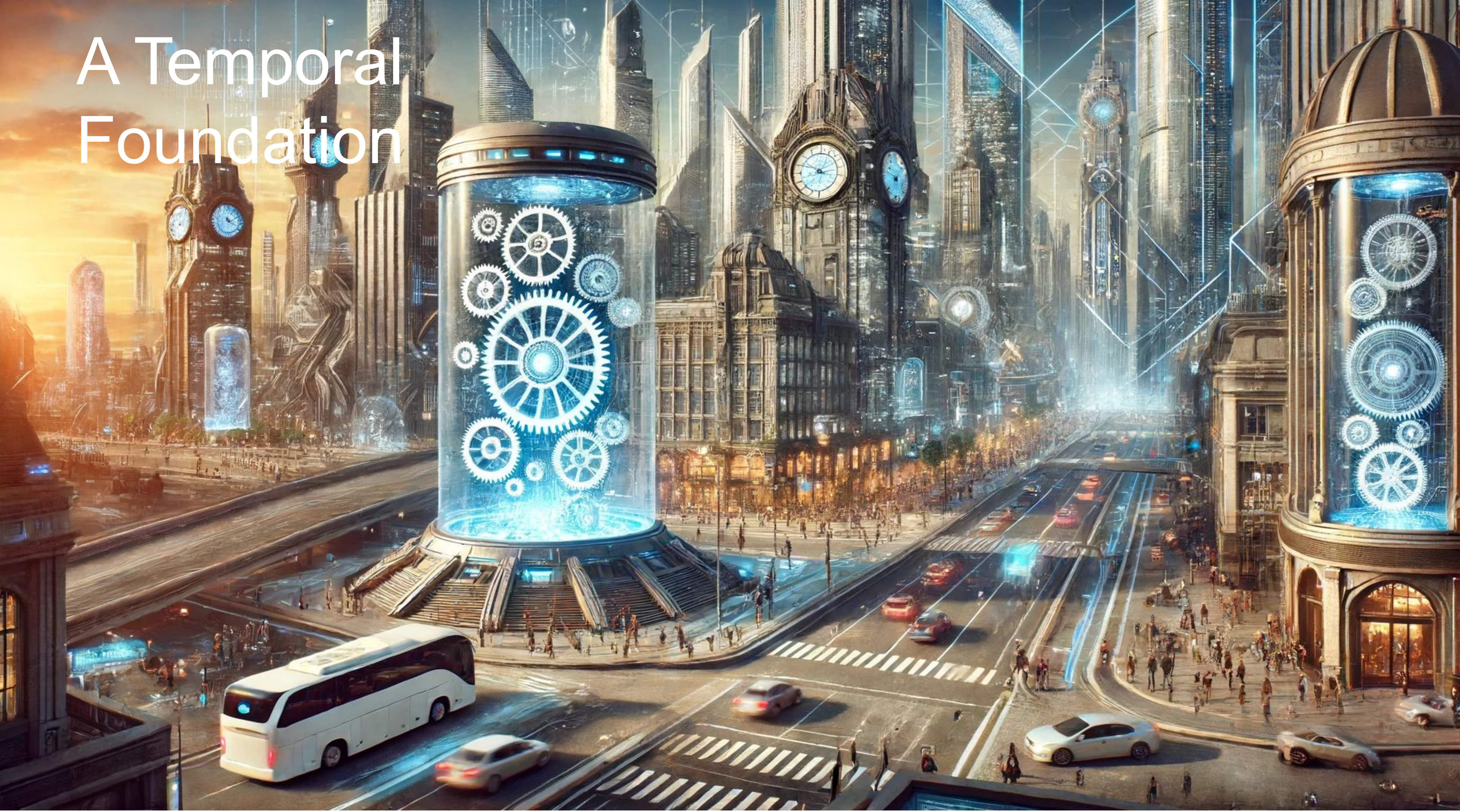
Golab, Lukasz, and M. Tamer Özsu. "Update-pattern-aware modeling and processing of continuous queries." *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. 2005.

Correctness

In Search of



A Temporal Foundation



Data Model

Physical Stream (PS) is infinite sequence of tuples (e, [ts, te)) with the same schema. Two elements i, j, are value-equivalent iff ei==ej

Logical Stream (LS) is a possibly infinite multiset of triples (e, t, n) composed of a record $e \in \Omega$, a point in time $t \in T$, and a multiplicity $n \in N$.

Physical to Logical (**planning**): For each tuple (e, [ts, te)) ∈ LS, we split the associated time interval into points of time at finest time granularity.

Logical to Physical (**execution**)
- Map each logical stream element (e,t,n) into a physical element (e,[t,t+1))
- Coalesce value-equivalent elements that are close to each other (maximal validity)

Krämer, Jürgen, and Bernhard Seeger. A temporal foundation for continuous queries over data streams. Univ., 2004.

Snapshot Reducibility

$$
\begin{tikzcd}
S_1, \ldots, S_n \arrow[r, "\tau_t"] \arrow[d, "\mathrm{Op}_T"'] & R_1, \ldots, R_n \arrow[d, "\mathrm{Op}"] \\
S'_1, \ldots, S'_m \arrow[r, "\tau_t"] & R'_1, \ldots, R'_m
\end{tikzcd}
$$

Krämer, Jürgen, and Bernhard Seeger. A temporal foundation for continuous queries over data streams. Univ., 2004.

Snapshot Reducibility

For a given logical stream LS and a specified point in time t, the timeslice operation returns a non-temporal multiset of all records in LS that are valid at time instant t

(Snapshot-Reducibility) A **logical** stream operator **opT** is snapshot-**reducible** to its non-temporal counterpart **op** over multisets, if for any point in time $t \in T$ and for all logical input streams $LS\_1, . . . , LS\_n$ $n \in Sl$,

Operators over logical streams

Kramer et al introduce the following operations on logical stream: filter(σ), map (μ), Cartesian product (×), duplicate elimination (δ), difference (−), group (γ), aggregation (α), union (∪) and window (ω).

Unfortunately, windows, unions, and are again non-window reducible…

Are we condemned to observe this kinds of results…?

The course of querying streams

Several people tries to overcome the issue of “memory”...

Yan-Nei et al showed us that if we use **sequences as our basic data model**, non-monotonic queries become so dominant that only basic project/select operations can be expressed as continuous queries.

The non-blocking subset of relational algebra (NB-RA) and SQL (NB-SQL) are not NB-complete i.e., it cannot express every monotonic set function

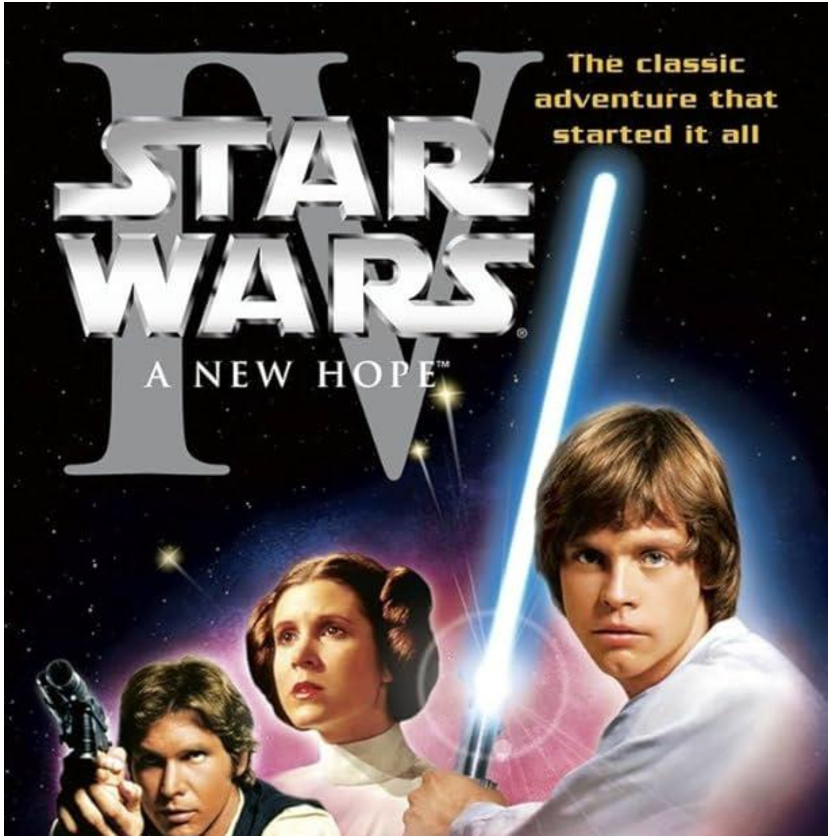
Law, Yan-Nei, Haixun Wang, and Carlo Zaniolo. "Relational languages and data models for continuous queries on sequences and data streams." *ACM Transactions on Database Systems (TODS)* 36.2 (2011): 1-32.

A New Hope

User defined Aggregates

A query language that supports non-blocking UDAs and set union can express all monotonic set functions on data streams.

While **UDAs** makes the query language NB-complete they also make it **turing complete** on classical tables



Law, Yan-Nei, Haixun Wang, and Carlo Zaniolo. "Relational languages and data models for continuous queries on sequences and data streams." *ACM Transactions on Database Systems (TODS)* 36.2 (2011): 1-32.

Time and Approximation

However, when moving to timestamped data stream (ordered set), we are losing again the monotonicity of binary operators (.e.g, join).

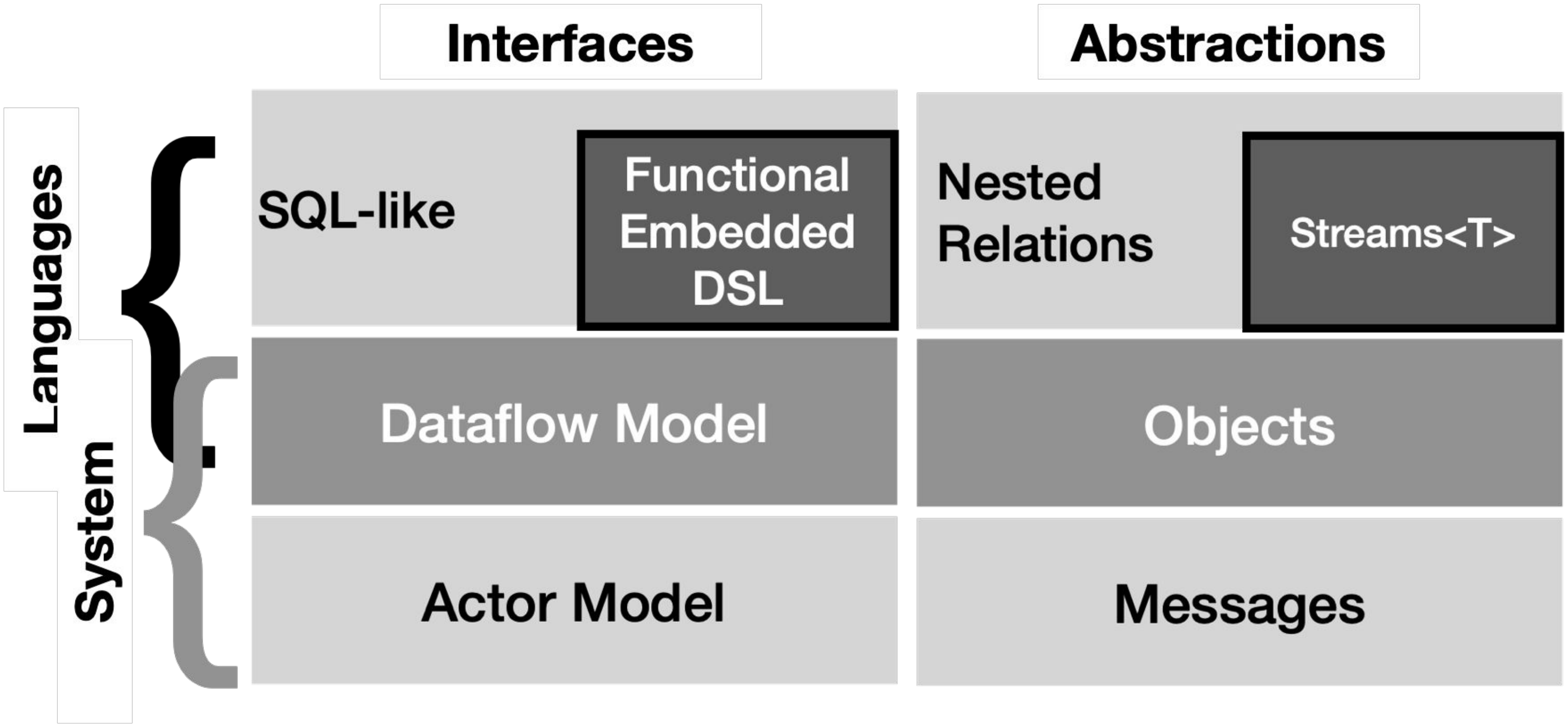
- **Can at least have a monotonic approximation?**
    - Yes, if we consider the recent sub-portion of the data stream: t-operations (union, product, difference) only look at the data up to t
- Focusing on **streams that have no delay** (not true in practice):
    - a query language that has UDA and a version of union that is **t-approximated** is NB-complete.

Law, Yan-Nei, Haixun Wang, and Carlo Zaniolo. "Relational languages and data models for continuous queries on sequences and data streams." *ACM Transactions on Database Systems (TODS)* 36.2 (2011): 1-32.

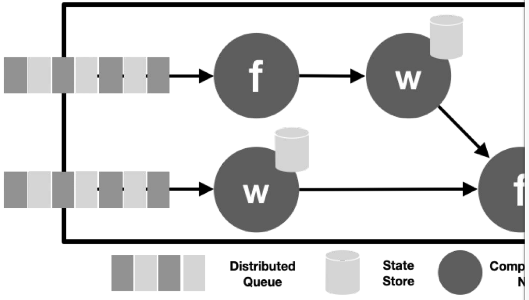
Streaming Systems

For Big Data

57


Interfaces
Abstractions
Languages
System
SQL-like
Functional Embedded DSL
Nested Relations
Streams<T>
Dataflow Model
Objects
Actor Model
Messages

Continuous Queries

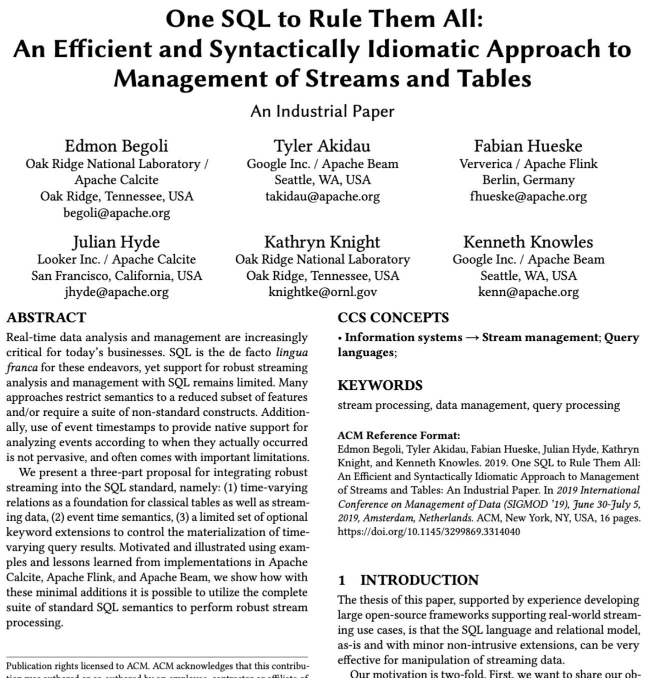



```
DataStream<T> input = ...;

// tumbling event-time windows
input
    .keyBy(<key selector>)
    .window(TumblingEventTimeWindows.of(Time.seconds(5)))
    .<windowed transformation>(<window function>);

// tumbling processing-time windows
input
    .keyBy(<key selector>)
    .window(TumblingProcessingTimeWindows.of(Time.seconds(5
    .<windowed transformation>(<window function>);

// daily tumbling event-time windows offset by -8 hours.
input
    .keyBy(<key selector>)
    .window(TumblingEventTimeWindows.of(Time.days(1), Time.hours(-8)))
    .<windowed transformation>(<window function>);
```

```
SELECT nation, COUNT(*) ,
HOP_START(..)
HOP_END(...)FROM
pageviewsGROUP BY
HOP(rowtime, INTERVAL 1H,
INTERVAL 1M), nation
```

SQL-Like Languages

- New trend is hiding the complexity of the processing behind SQL
- Alternative design debates on how to extend the languages
  - WINDOW Clauses
  - CEP Operations
  - Extended GroupBy
  - Report Controlling



One SQL to Rule Them All: An Efficient and Syntactically Idiomatic Approach to Management of Streams and Tables

An Industrial Paper

Edmon Begoli
Oak Ridge National Laboratory / Apache Calcite
Oak Ridge, Tennessee, USA
begoli@apache.org

Tyler Akidau
Google Inc. / Apache Beam
Seattle, WA, USA
takidau@apache.org

Fabian Hueske
Ververica / Apache Flink
Berlin, Germany
fhueske@apache.org

Julian Hyde
Looker Inc. / Apache Calcite
San Francisco, California, USA
jhyde@apache.org

Kathryn Knight
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA
knightke@ornl.gov

Kenneth Knowles
Google Inc. / Apache Beam
Seattle, WA, USA
kenn@apache.org

ABSTRACT

Real-time data analysis and management are increasingly critical for today's businesses. SQL is the de facto *lingua franca* for these endeavors, yet support for robust streaming analysis and management with SQL remains limited. Many approaches restrict semantics to a reduced subset of features and/or require a suite of non-standard constructs. Additionally, use of event timestamps to provide native support for analyzing events according to when they actually occurred is not pervasive, and often comes with important limitations.

We present a three-part proposal for integrating robust streaming into the SQL standard, namely: (1) time-varying relations as a foundation for classical tables as well as streaming data, (2) event time semantics, (3) a limited set of optional keyword extensions to control the materialization of time-varying query results. Motivated and illustrated using examples and lessons learned from implementations in Apache Calcite, Apache Flink, and Apache Beam, we show how with these minimal additions it is possible to utilize the complete suite of standard SQL semantics to perform robust stream processing.

CCS CONCEPTS

• **Information systems → Stream management**; **Query languages**;

KEYWORDS

stream processing, data management, query processing

ACM Reference Format:

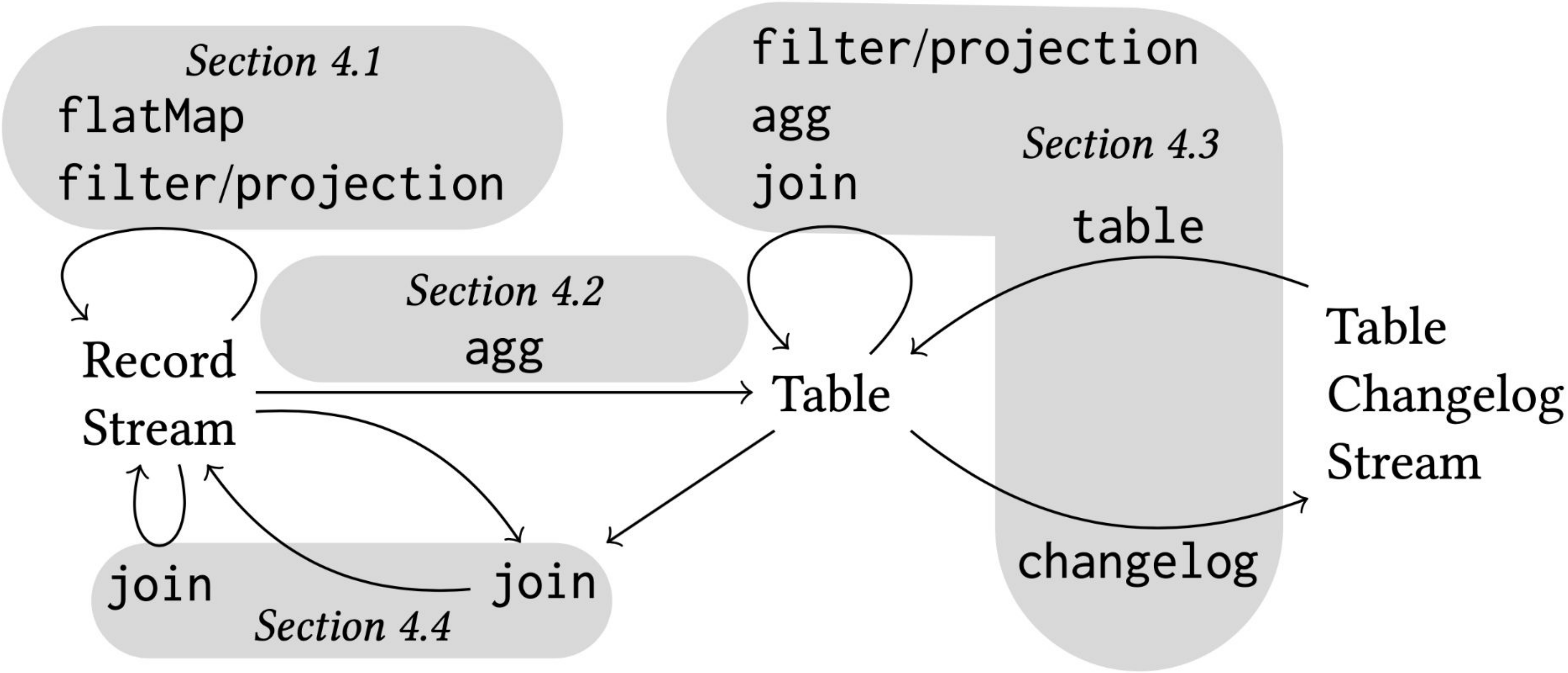
Edmon Begoli, Tyler Akidau, Fabian Hueske, Julian Hyde, Kathryn Knight, and Kenneth Knowles. 2019. One SQL to Rule Them All: An Efficient and Syntactically Idiomatic Approach to Management of Streams and Tables: An Industrial Paper. In *2019 International Conference on Management of Data (SIGMOD '19), June 30-July 5, 2019, Amsterdam, Netherlands.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3299869.3314040

1 INTRODUCTION

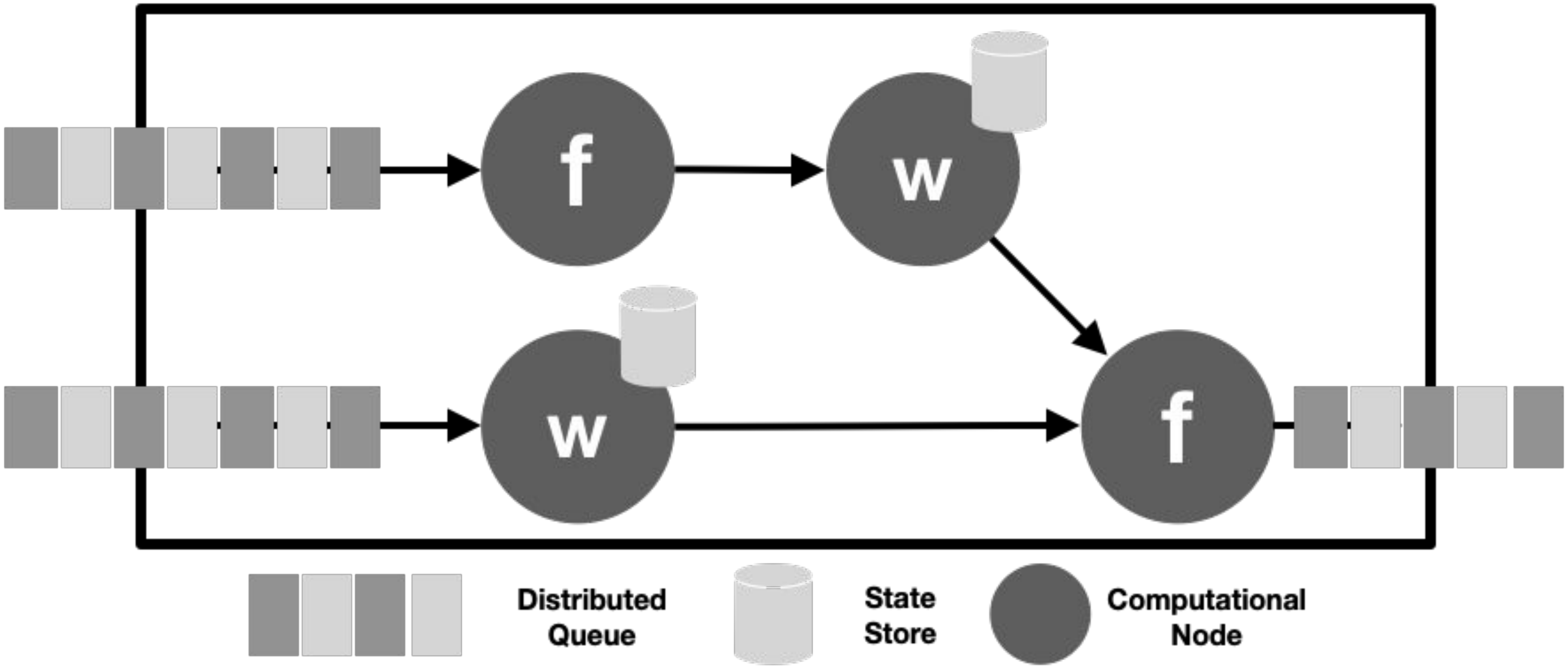
The thesis of this paper, supported by experience developing large open-source frameworks supporting real-world streaming use cases, is that the SQL language and relational model, as-is and with minor non-intrusive extensions, can be very effective for manipulation of streaming data.

Our motivation is two-fold. First, we want to share our ob-

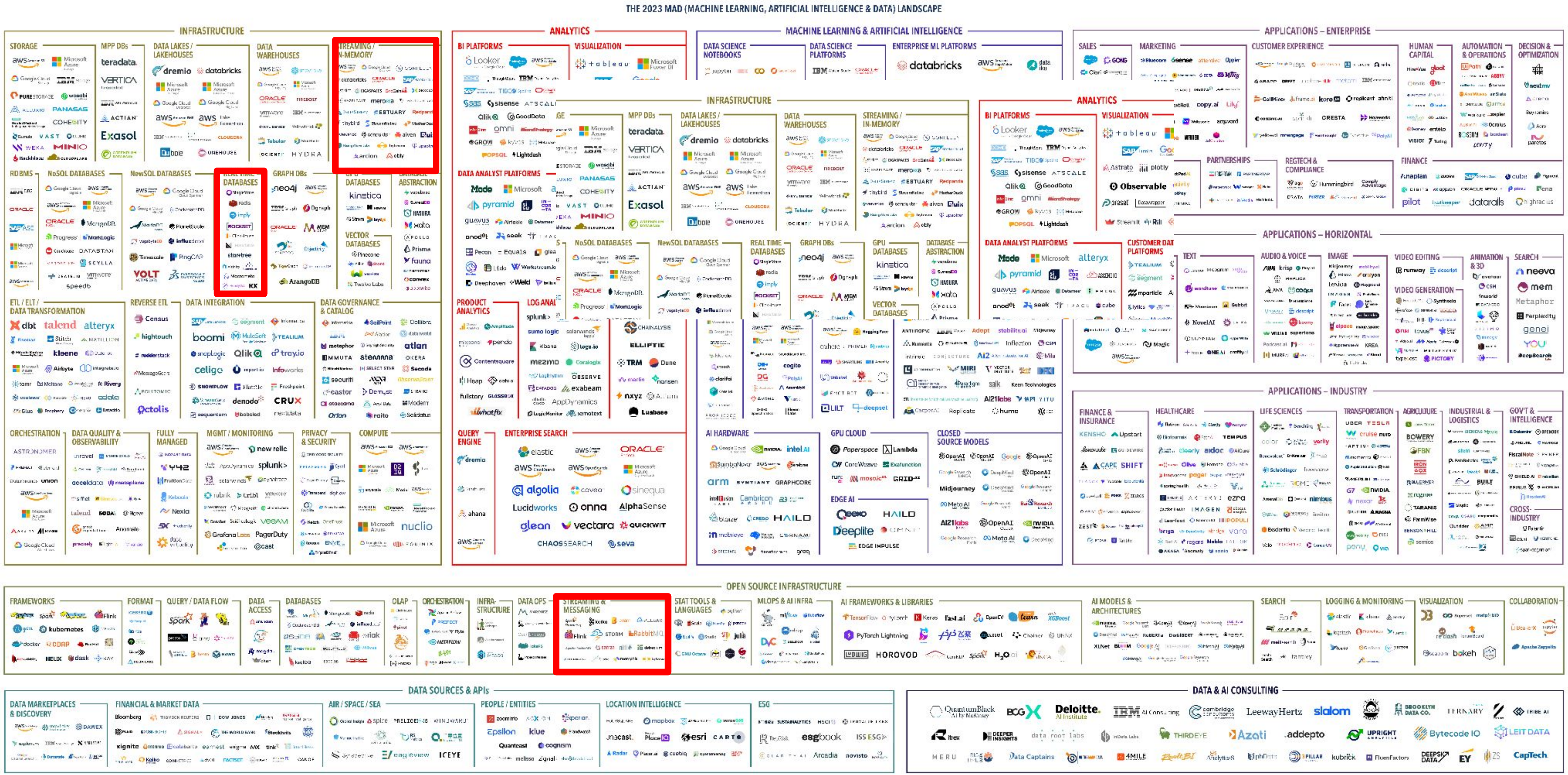
Functional DSL

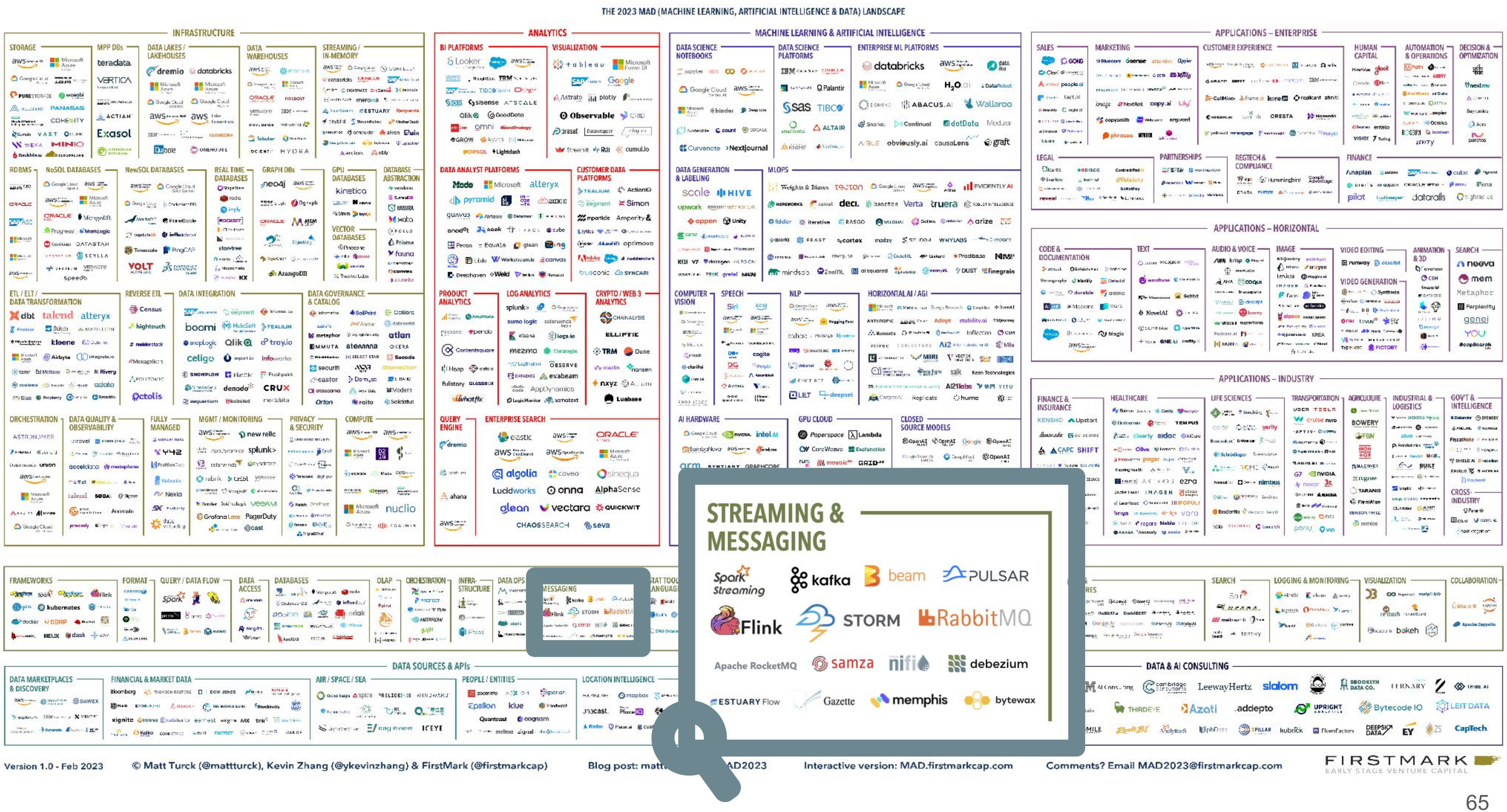


Operator Topology

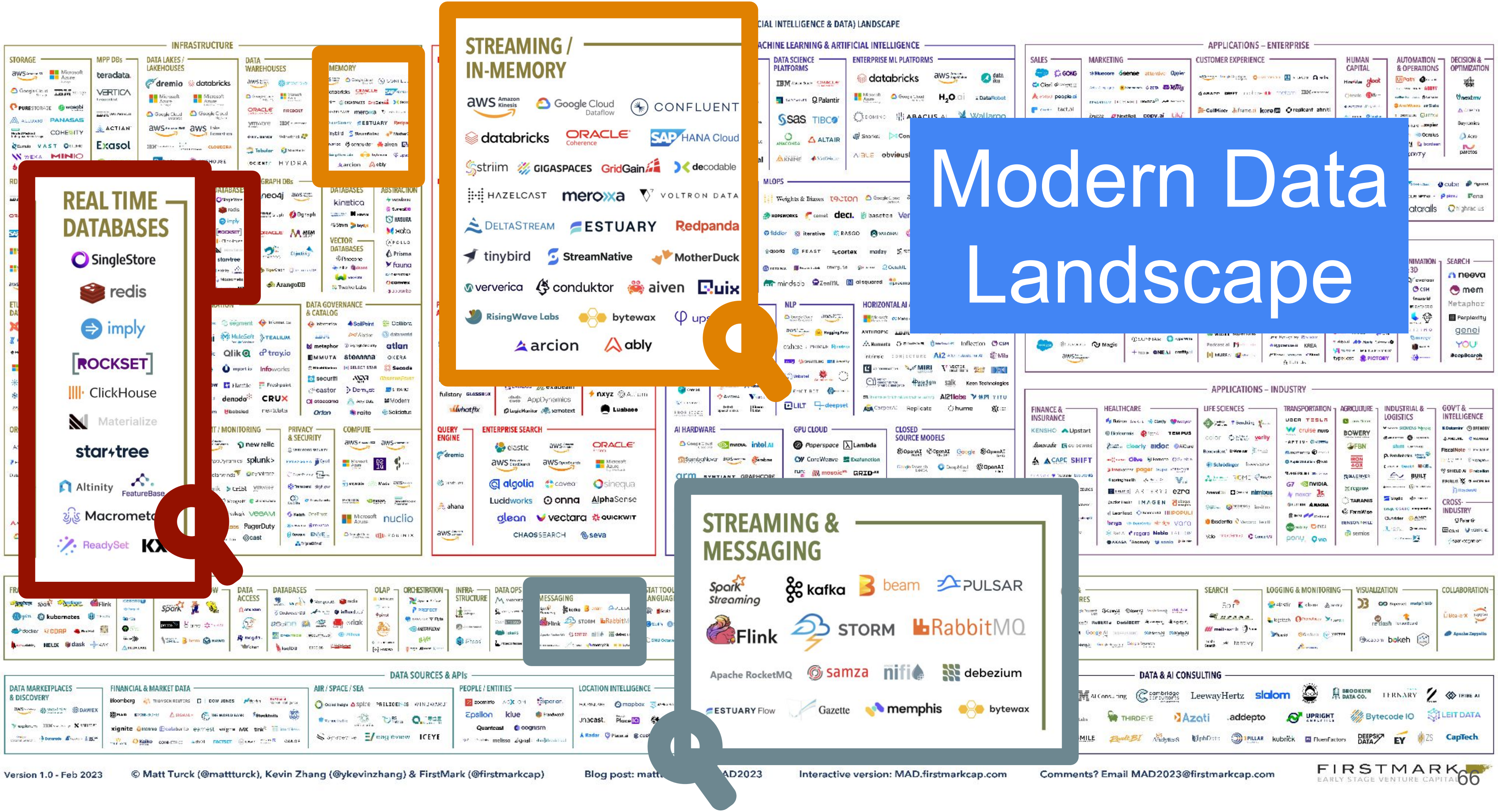


Continuous Queries in the Modern Data Landscape


THE 2023 MAD (MACHINE LEARNING, ARTIFICIAL INTELLIGENCE & DATA) LANDSCAPE
INFRASTRUCTURE
ANALYTICS
MACHINE LEARNING & ARTIFICIAL INTELLIGENCE
APPLICATIONS – ENTERPRISE
APPLICATIONS – HORIZONTAL
APPLICATIONS – INDUSTRY
OPEN SOURCE INFRASTRUCTURE
DATA SOURCES & APIs
DATA & AI CONSULTING


THE 2023 MAD (MACHINE LEARNING, ARTIFICIAL INTELLIGENCE & DATA) LANDSCAPE
INFRASTRUCTURE
ANALYTICS
MACHINE LEARNING & ARTIFICIAL INTELLIGENCE
APPLICATIONS – ENTERPRISE
APPLICATIONS – HORIZONTAL
APPLICATIONS – INDUSTRY
DATA SOURCES & APIs
DATA & AI CONSULTING
STREAMING & MESSAGING
Spark Streaming
kafka
beam
PULSAR
Flink
STORM
RabbitMQ
Apache RocketMQ
samza
nifi
debezium
ESTUARY Flow
Gazette
memphis
bytewax
Version 1.0 - Feb 2023
© Matt Turck (@mattturck), Kevin Zhang (@ykevinzhang) & FirstMark (@firstmarkcap)
Interactive version: MAD.firstmarkcap.com
Comments? Email MAD2023@firstmarkcap.com
FIRSTMARK
EARLY STAGE VENTURE CAPITAL

Modern Data Landscape



Modern Landscape

- Real-Time Databases
  - focus on OLAP
  - long time series
- Streaming Databases
  - in-database windowing
  - continuous computation

Incremental View Maintenance

a long standing problem

- Beyond conjunctive queries, there are studies on IVM for intersection joins, Datalog, Differential Datalog, and DBSP (best paper VLDB 2023)
- The maintenance of complex analytics over evolving databases, which includes linear algebra computation, collection programming, and in-database machine learning.

DELTASTREAM

MATERIALIZE

Feldera

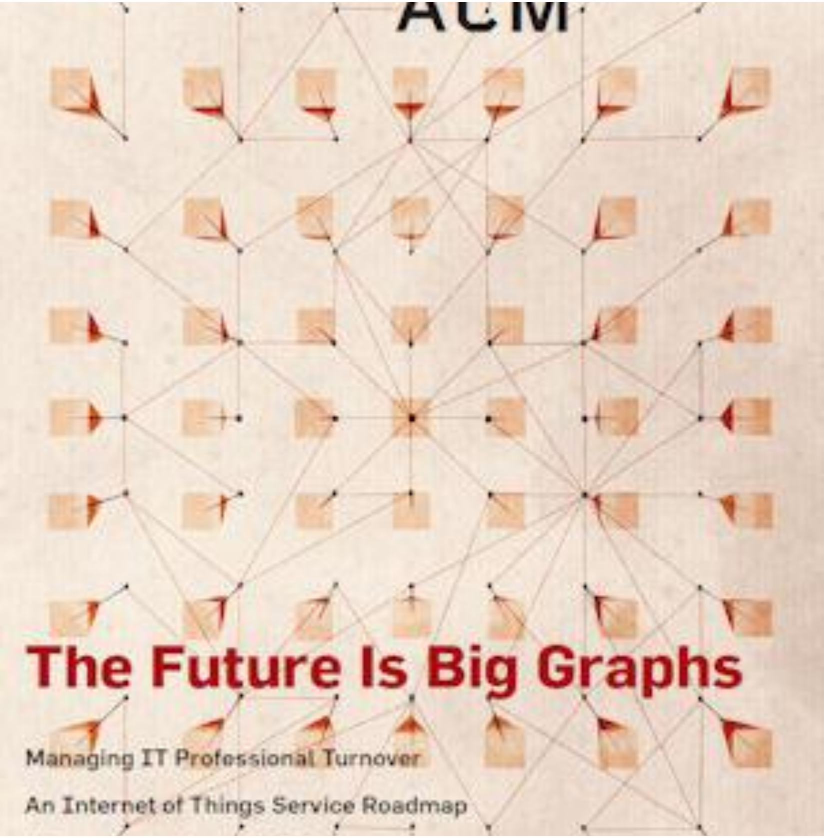
Olteanu, Dan. "Recent Increments in Incremental View Maintenance." *arXiv preprint arXiv:2404.17679* (2024).

Streaming Graphs

The Future is Big Graphs

In 2021 already

- A Community Vision of the role of graph in the future years
- Streaming Graph processing is core
  - Complex Query Execution
  - Incremental Graph Analytics
  - Temporal Dynamic Graphs



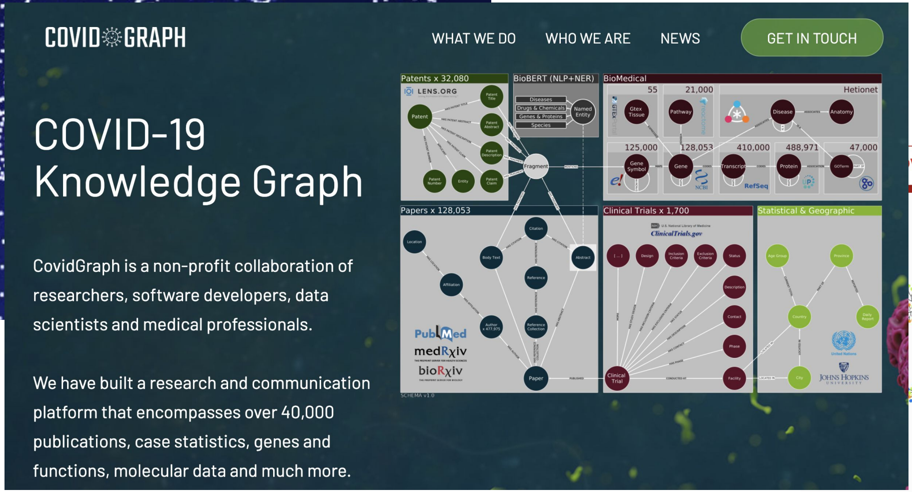
Challenges for Next-generation Graph Processing Systems

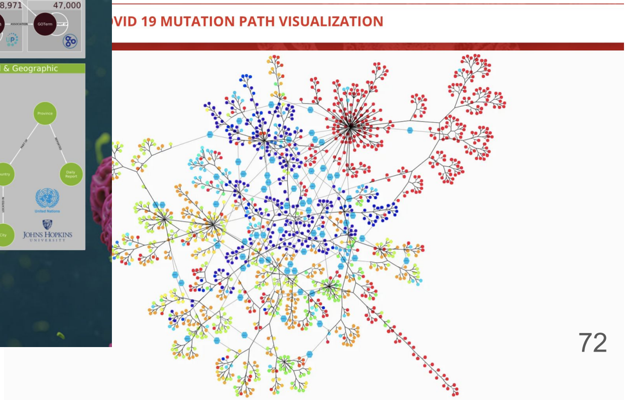
- Ch1. A lattice of graph data models and graph algebras
- Ch2. Complex data management ecosystems
- Ch3. Performance and benchmarking

S. Sakr, A. Bonifati, H. Voigt, A. Iosup et al. “The Future is Big Graphs: A Community View on Graph Processing Systems” Commun. ACM 64(9): 62-71 (2021)

Graphs are ubiquitous across diverse applications

- Several killer applications exist, e.g. financial, logistic, scientific, fraud detection, cybersecurity, supply chain management etc.





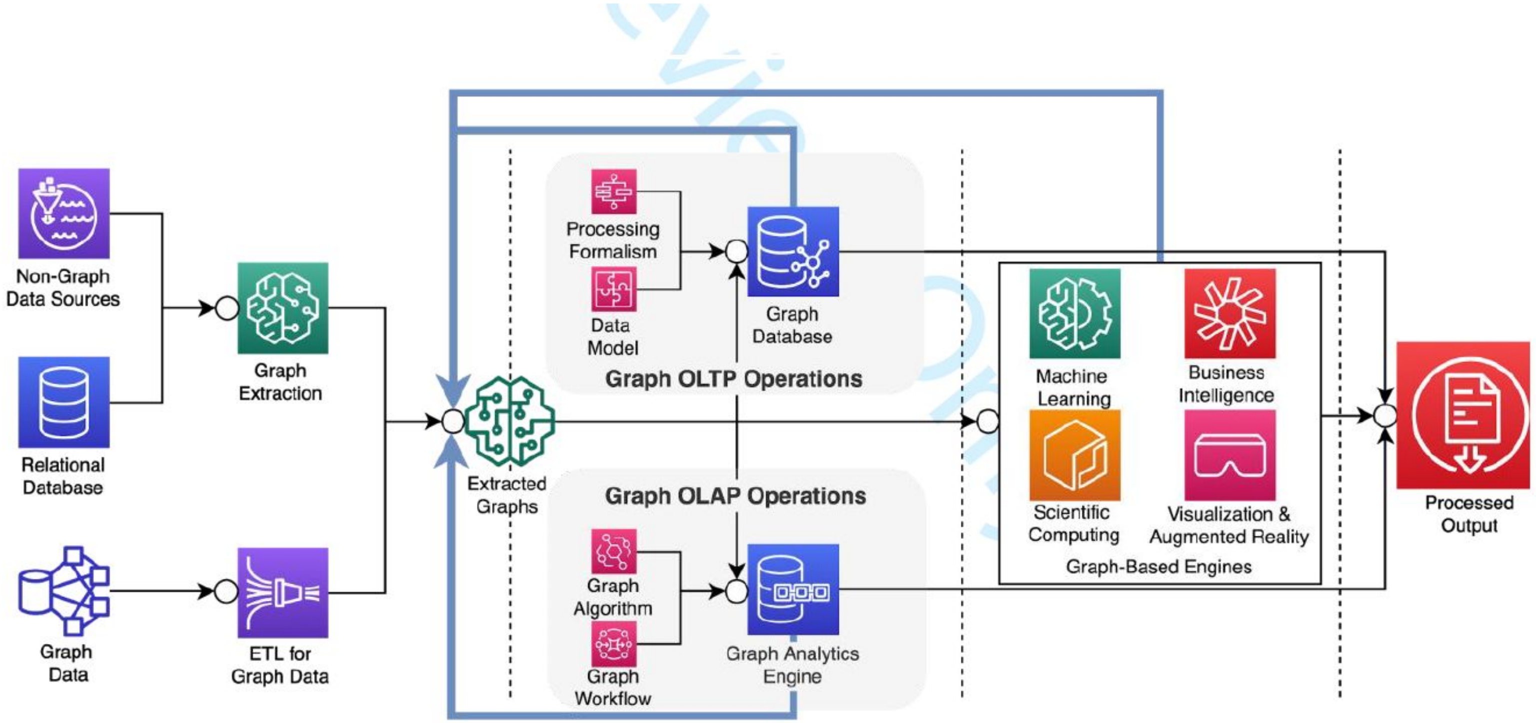
Ch1. Expressivity of the graphs/queries

- Dependence on the chosen data model
- How do humans conceptualize graphs?
- The interoperability issues (due to multiple heterogeneous data sources) are to be taken into account
- A data model lattice to navigate across data models, balancing understandability and expressive power
- A new algebra for the variety of graph workloads

A. Bonifati, G. Fletcher, H. Voigt, N. Yakovets "Querying Graphs" Synthesis Lectures on Data Management. Springer 2018

73

Ch2. A complex data management ecosystem



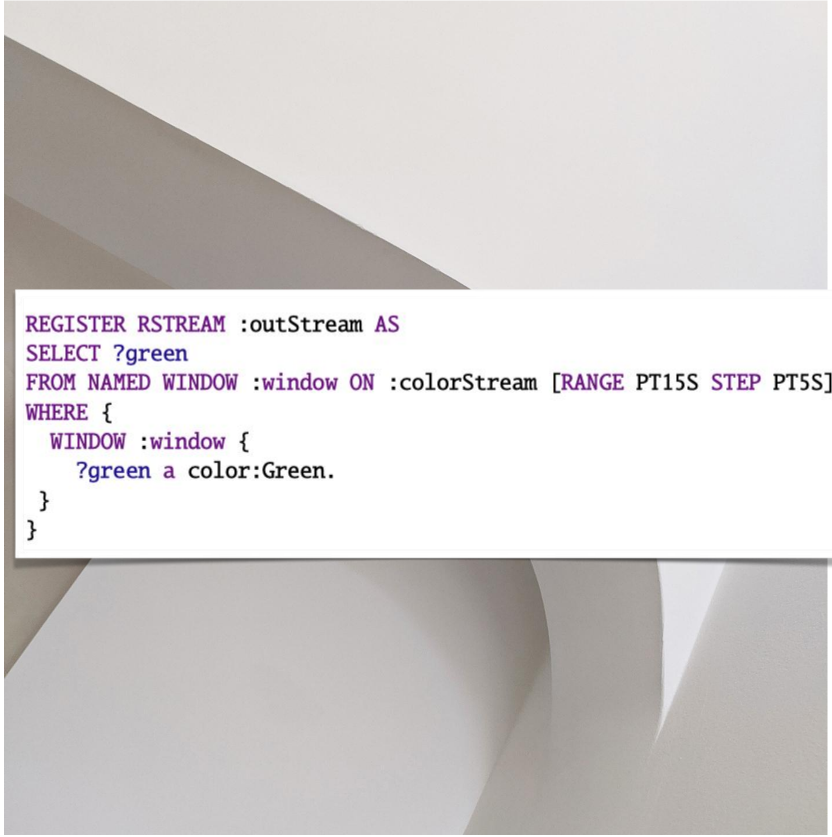
Ch3. Performance and benchmarking

- The need for new, reproducible experimental methodologies to facilitate quick yet meaningful performance-testing?
- How to define more faithful metrics for executing a graph algorithm, query, program, or workflow?
- How to generate workloads with combined operations, covering temporal, spatial, and streaming aspects?
- How to benchmark pipelines including machine learning and simulation?

Dell’Aglio et al

RSP-QL

- Input data: RDF Triples (s,p,o)
- Semantics is based on denotation
  - extends SPARQL with window functions (outside algebraic structure)
  - derived from CQL
- output: time-annotated binding
  - or graphs



```
REGISTER RSTREAM :outStream AS
SELECT ?green
FROM NAMED WINDOW :window ON :colorStream [RANGE PT15S STEP PT5S]
WHERE {
  WINDOW :window {
    ?green a color:Green.
 }
}
```

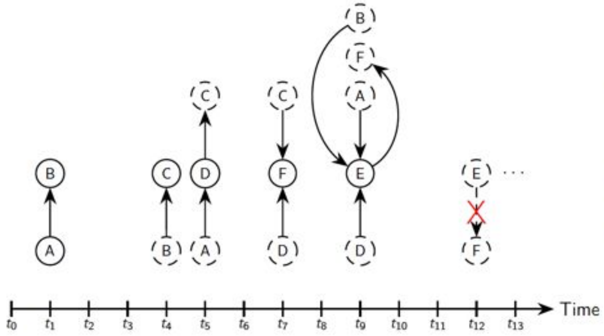
Streaming Graphs

Towards Stream Graph Processing Systems

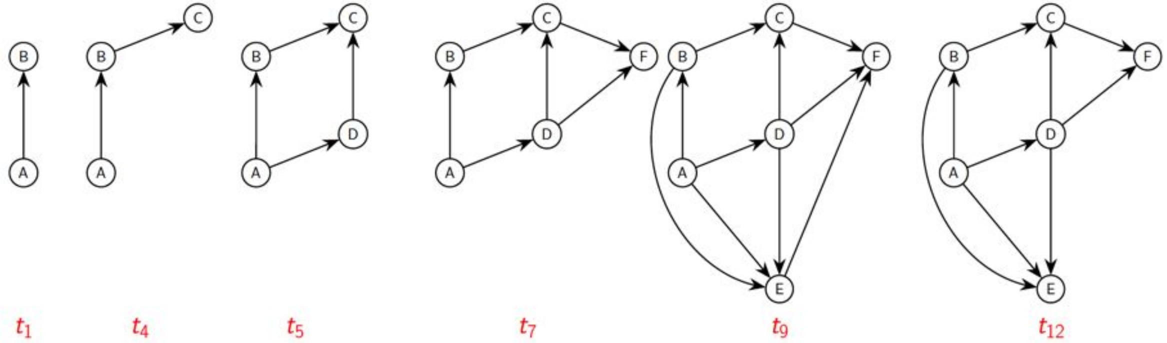
- Dynamic graphs are graphs that can accommodate updates (insertions, deletions, changes) and allow querying on the new/old state
- Streaming graphs are graphs that are unbound as new data arrives at high-speed.
- Current systems and libraries (Gelly/Apache Flink) focus on aggregates/projections
- However, more complex query processing operators taking into account recursion, path-oriented semantics etc. need to be investigated
- Graph processing systems are also inherently dynamic and need to respond to all these challenges

Streaming Graphs

Building the underlying graph one edge at a time

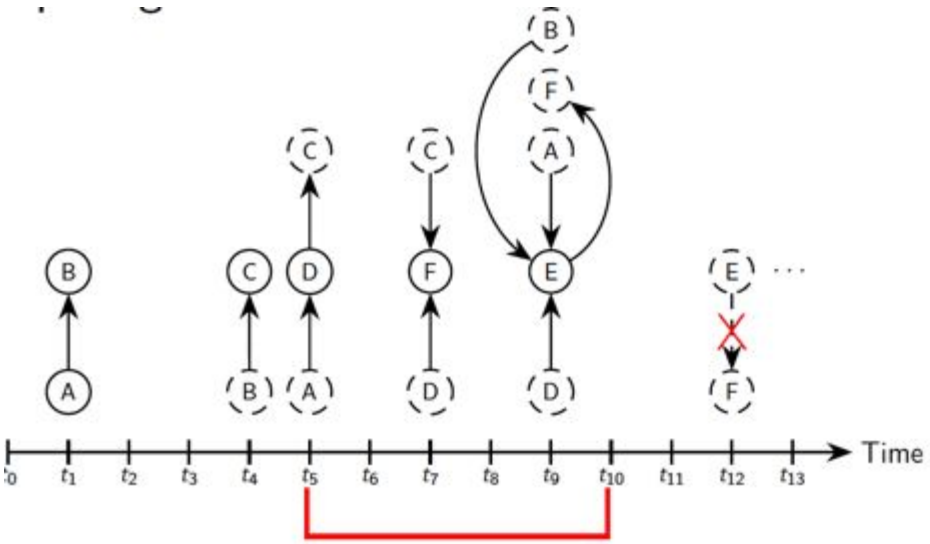


- Combines two difficult problems: streaming+graphs
- Unbounded $\Rightarrow$ don't see entire graph
- Streaming rates can be very high



Streaming graph models

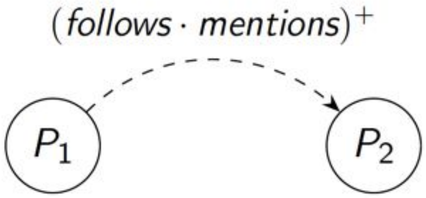
- Window-based semantics (use window to batch edges)
- Continuous semantics (edges are batched as they come)
- Complex vs. Simple operations

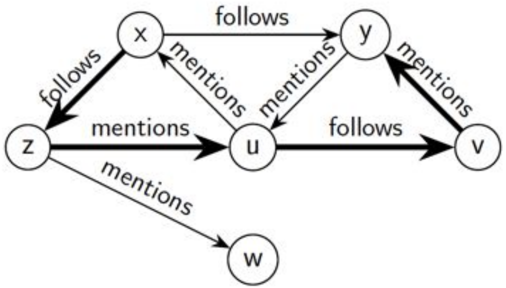


Streaming Regular Path Queries (RPQs)

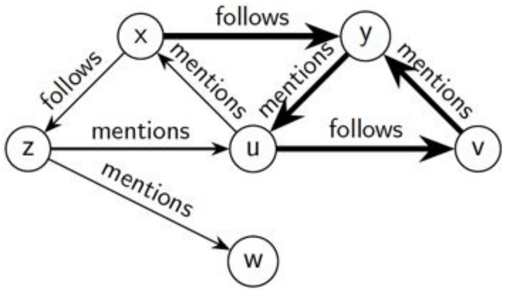
Reflecting the different semantics of graph queries

$$Q_1 = (follows \cdot mentions)^+$$





Simple paths

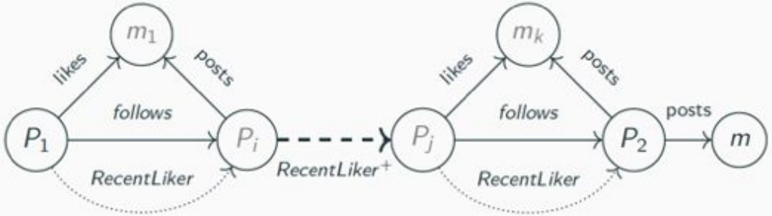


Arbitrary paths

Anil Pacaci et al.: Regular Path Query Evaluation on Streaming Graphs. SIGMOD Conference 2020: 1415-1430

Towards a Streaming Graph Query Processor

Based on LDBC SNB Interactive Query 7:



G-CORE representation

```
PATH RL = (x)-[:follows]->(y),
          (x)-[:likes]->(m1)<-[:posts]-(y)
CONSTRUCT (p1) -[:notify]-> (m)
MATCH (p1)-/ <~RL+> /->(p2),
      (p2)-[:posts]->(m)
ON ldbc_stream WINDOW(24 hours)
```

Datalog program:

$$RL(u_1, u_2) \leftarrow l(u_1, m_1), f(u_1, u2), p(u_2, m_1)$$

$$Answer(u, m) \leftarrow RL^{+}(u, u_2), p(u_2, m)$$

Streaming Graph Algebra

A common foundation for streaming graph query engines

G-CORE Query:

```
PATH RL = (x)-[:follows]->(y),
          (x)-[:likes]->(m1)<-[:posts]-(y)
CONSTRUCT (p1) -[:notify]-> (m)
MATCH (p1)-/ <~RL+> /->(p2),
      (p2)-[:posts]->(m)
ON ldbc_stream WINDOW(24 hours)
```

SGA Expression

$$S_l = \sigma_{l=likes}(\mathcal{W}^{24}(S))$$

$$S_f = \sigma_{l=follows}(\mathcal{W}^{24}(S))$$

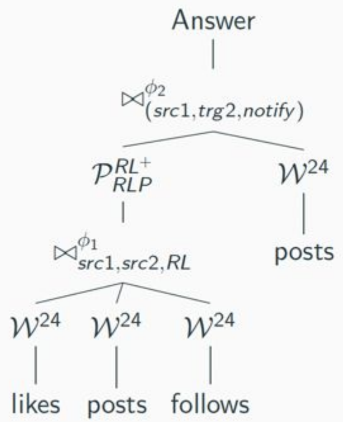
$$S_p = \sigma_{l=posts}(\mathcal{W}^{24}(S))$$

$$S_{RecentLiker} = \bowtie_{\phi}^{src1,src3,RecentLiker}(S_{likes}, S_{follows}, S_{posts})$$

$$S_{Related} = \mathcal{P}_{RecentLiker^+}^{Notify}(S_{RecentLiker})$$

$$Answer = \bowtie_{\phi}^{src1,trg2,Notify}(S_{Related}, S_p)$$

Logical query plan



A. Pacaci, A. Bonifati, T. Ozsu. Evaluating Complex Queries on Streaming Graphs. In IEEE ICDE 2022

Streaming graphs in graph query languages

- Input stream of edges (tuples)
- Snapshot graph is the query focus
- Query models is based on non-recursive Datalog + Kleene Star
- Semantics derived from snapshot reducibility (over graphs)
- Output: a graph (path) detected (as in standard query languages, such as GQL and SQL/PGQ)

G-CORE Query:

```
PATH RL = (x)-[:follows]->(y),
          (x)-[:likes]->(m1)<-[:posts]-(y)
CONSTRUCT (p1) -[:notify]-> (m)
MATCH (p1)-/ <~RL+> /->(p2),
      (p2)-[:posts]->(m)
ON ldbc_stream WINDOW(24 hours)
```

How can we continuously process large graph streams as soon as they are discovered?

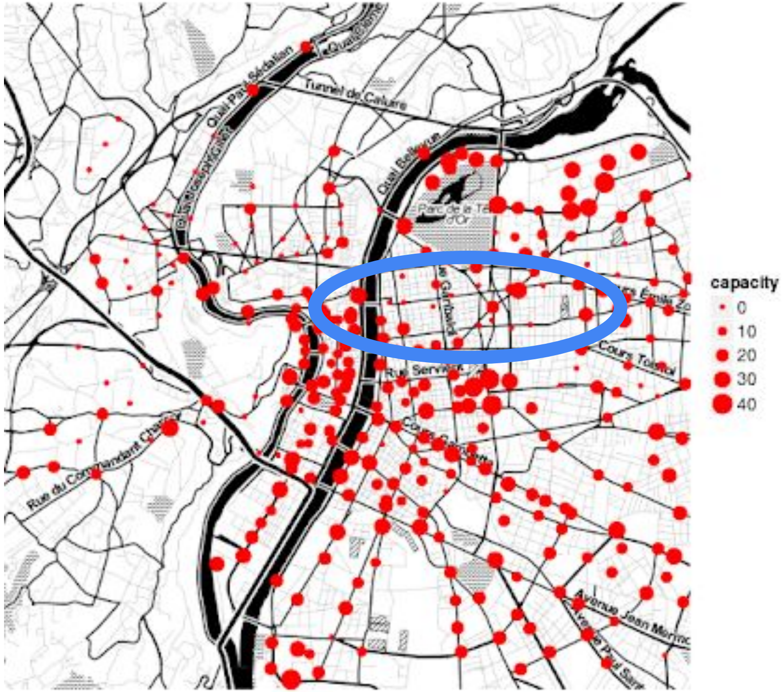
Can we design a declarative language that enables continuous graph querying?

Research Question

Research Goal

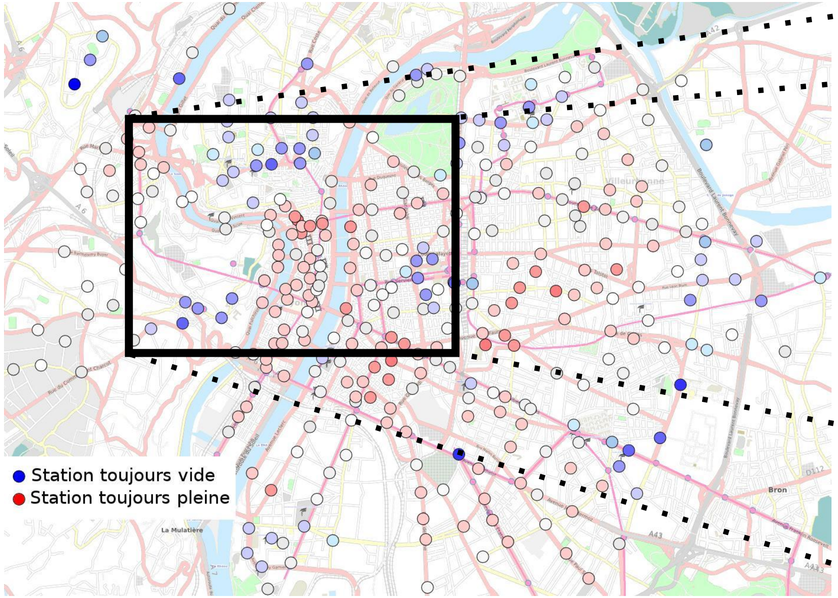
~~Running~~ Riding Example

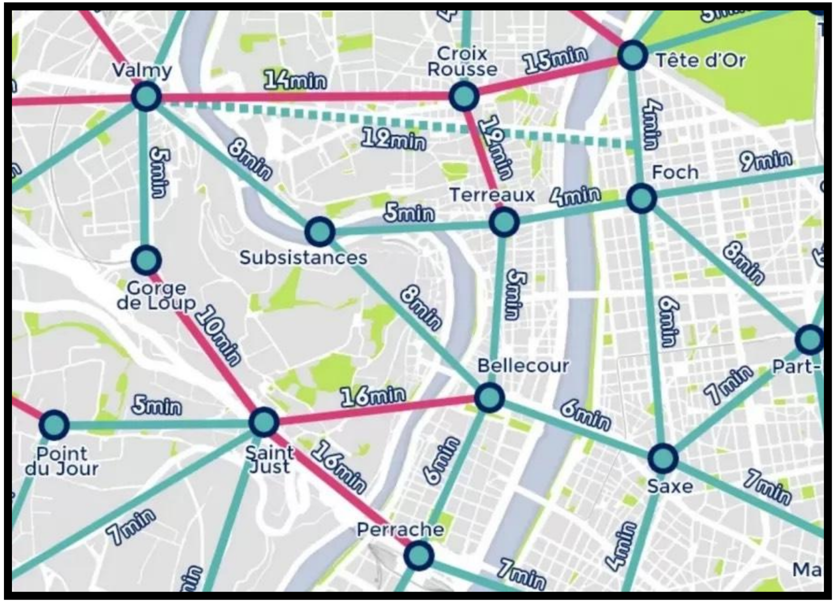
Detecting Free Riders in Smart Biking



~~Running~~ Riding Example

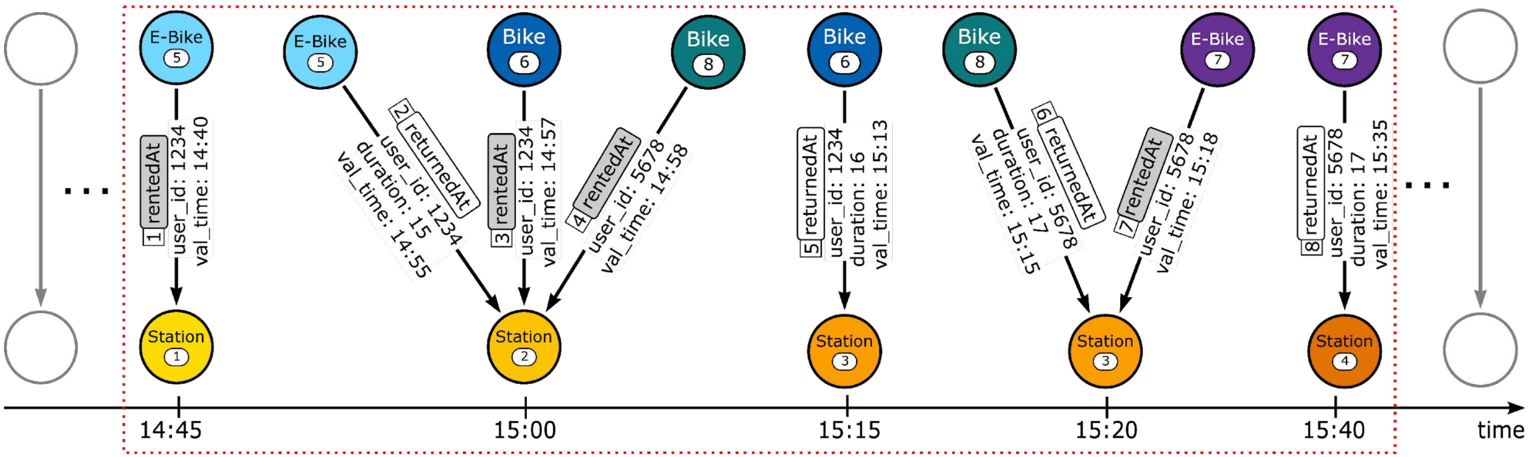
Detecting Free Riders in Smart Biking





~~Running~~ Riding Example

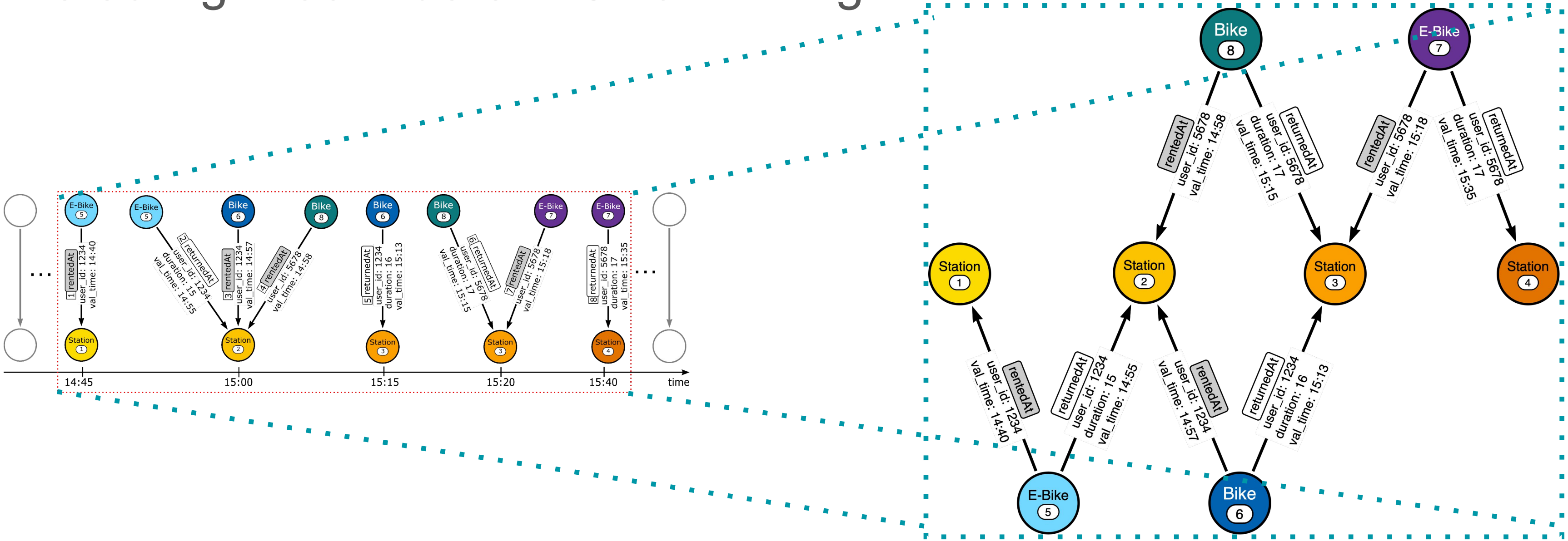
Detecting Free Riders in Smart Biking



What users have used the free period for subsequent rentals in the last hour?

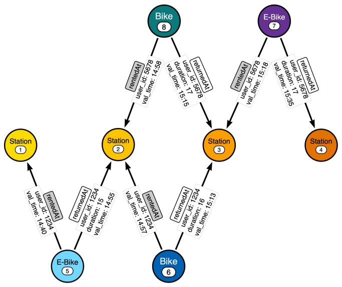
~~Running~~ Riding Example

Detecting Free Riders in Smart Biking



What users have used the free period for subsequent rentals in the last hour?

~~Running~~ Riding Example



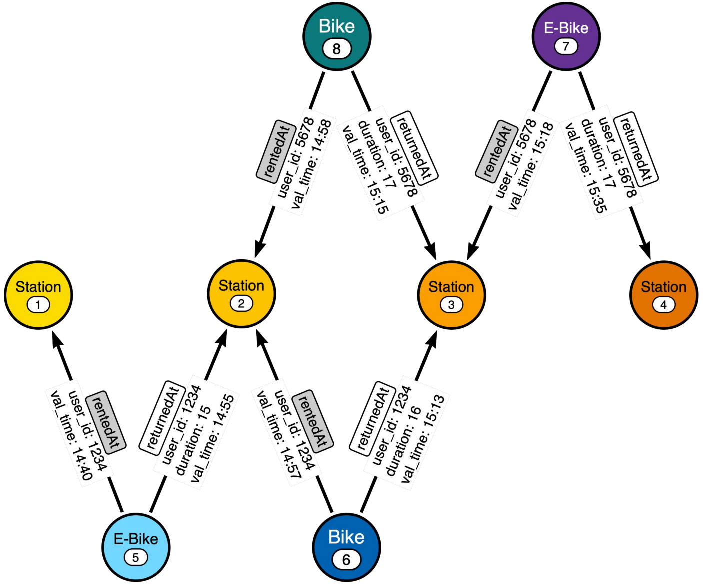
Detecting Free Riders in Smart Biking

What users have used the free period for subsequent rentals in the last hour?

```
WITH     datetime() - duration('PT60M') AS win_start, datetime() AS win_end
MATCH (:Bike)-[r:rentedAt]->(s:Station),
          q = (b)-[:returnedAt|rentedAt*3..]-(o:Station)
WITH r, s, q, relationships(q) AS rels,
        [n IN nodes(q) WHERE 'Station' IN labels(n) | n.id] AS hops
WHERE ALL(e IN rels WHERE win_start <= e.val_time <= win_end
               AND e.user_id = r.user_id AND e.val_time > r.val_time
               AND (e.duration IS NULL OR e.duration < 20) )
RETURN r.user_id, s.id, r.val_time, hops
```

~~Running~~ Riding Example

Detecting Free Riders in Smart Biking




```
WITH datetime() - duration('PT60M') AS win_start, datetime() AS win_end
MATCH (:Bike)-[r:rentedAt]->(s:Station), q = (b)-[:returnedAt|rentedAt*3..]-
(o:Station)
WITH r, s, q, relationships(q) AS rels, [n IN nodes(q) WHERE 'Station' IN
labels(n) | n.id] AS hops
WHERE ALL(e IN rels WHERE win_start <= e.val_time <= win_end
            AND e.user_id = r.user_id AND e.val_time > r.val_time
            AND (e.duration IS NULL OR e.duration < 20) )
RETURN r.user_id, s.id, r.val_time, hops
```

r.user_id	s.id	r.val_time	hops
1234	1	14:40	[2,3]
5678	2	14:58	[3,4]

~~Running~~ Riding Example

Detecting Free Riders in Smart Biking

r.user_id	s.id	r.val_time	hops
1234	1	14:40	[2,3]
5678	2	14:58	[3,4]

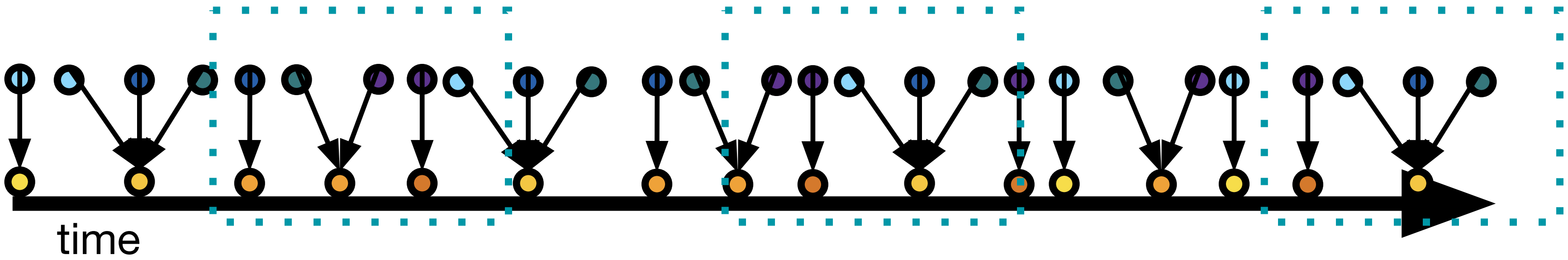
```
WITH datetime() - duration('PT60M') AS win_start, datetime() AS win_end
MATCH (:Bike)-[r:rentedAt]->(s:Station), q = (b)-[:returnedAt|rentedAt*3..]-
(o:Station)
WITH r, s, q, relationships(q) AS rels, [n IN nodes(q) WHERE 'Station' IN
labels(n) | n.id] AS hops
WHERE ALL(e IN rels WHERE win_start <= e.val_time <= win_end
          AND e.user_id = r.user_id AND e.val_time > r.val_time
          AND (e.duration IS NULL OR e.duration < 20) )
RETURN r.user_id, s.id, r.val_time, hops
```


r.user_id	s.id	r.val_time	hops
1234	1	14:40	[2,3]
5678	2	14:58	[3,4]

```
WITH datetime() - duration('PT60M') AS win_start, datetime() AS win_end
MATCH (:Bike)-[r:rentedAt]->(s:Station), q = (b)-[:returnedAt|rentedAt*3..]-
(o:Station)
WITH r, s, q, relationships(q) AS rels, [n IN nodes(q) WHERE 'Station' IN
labels(n) | n.id] AS hops
WHERE ALL(e IN rels WHERE win_start <= e.val_time <= win_end
          AND e.user_id = r.user_id AND e.val_time > r.val_time
          AND (e.duration IS NULL OR e.duration < 20) )
RETURN r.user_id, s.id, r.val_time, hops
```

r.user_id	s.id	r.val_time	hops
1234	1	14:40	[2,3]
5678	2	14:58	[3,4]

```
WITH datetime() - duration('PT60M') AS win_start, datetime() AS win_end
MATCH (:Bike)-[r:rentedAt]->(s:Station), q = (b)-[:returnedAt|rentedAt*3..]-
(o:Station)
WITH r, s, q, relationships(q) AS rels, [n IN nodes(q) WHERE 'Station' IN
labels(n) | n.id] AS hops
WHERE ALL(e IN rels WHERE win_start <= e.val_time <= win_end
          AND e.user_id = r.user_id AND e.val_time > r.val_time
          AND (e.duration IS NULL OR e.duration < 20) )
RETURN r.user_id, s.id, r.val_time, hops
```

Using Cypher for Streaming

Pros

- Declarative
- Interactive
- Intuitive
- Standard-ISH*

Cons

- Temporality is reduced to a selection
    - Verbose
    - Unoptimised
- “Now” = User Time
    - Not Reactive
- Results Reporting:
    - Now + Latency


You don't really know ~~someone~~ your data until you ~~fight~~ stream them

- **Declarative Semantics**. Seraph allows systems portability and optimisations, as well as adoption.
- **Continuous evaluation.** Seraph's operators allow the repeated evaluation over time, i.e., choosing a time interval and a sequence to evaluate the query.
- **Result emitting**. Seraph's operators allow controlling the report of results, i.e., what is part of the result and when it will be ready to be emitted.
- **Preserving expressiveness**. Seraph preserves openCypher's expressiveness

Christopher Rost et al. Seraph: Continuous Queries on Property Graph Streams. EDBT 2024: 234-247

Seraph's Syntax

Before

```
WITH datetime() - duration('PT60M') AS
win_start, datetime() AS win_end
MATCH (:Bike)-[r:rentedAt]->(s:Station), q =
(b)-[:returnedAt|rentedAt*3..]-(o:Station)
WITH r, s, q, relationships(q) AS rels, [n IN
nodes(q) WHERE 'Station' IN labels(n) |
n.id] AS hops
WHERE ALL(e IN rels WHERE win_start
<= e.val_time <= win_end
                    AND e.user_id = r.user_id
AND e.val_time > r.val_time
                    AND (e.duration IS NULL
OR e.duration < 20) )
RETURN r.user_id, s.id, r.val_time, hops
```

After

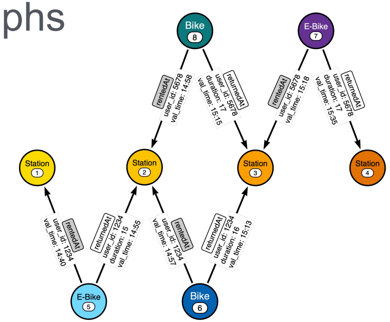
```
REGISTER QUERY student_trick
STARTING AT 2022-10-14T14:45 {

   MATCH (:Bike)-[r:rentedAt]->(s:Station),
   q = (b)-[:returnedAt|rentedAt*3..]-  (o:Station)
   WITHIN PT1H
   WITH r, s, q, relationships(q) AS rels,
   [n IN nodes(q) WHERE 'Station' IN labels(n) |
n.id] AS hops
   WHERE ALL(e IN rels WHERE e.user_id =
r.user_id AND e.
   val_time > r.val_time AND e.duration < 20)
   EMIT r.user_id, s.id, r.val_time, hops
   ON ENTERING EVERY PT5M
   }
```

Seraph's Data Model

After

Before

- Property Graphs



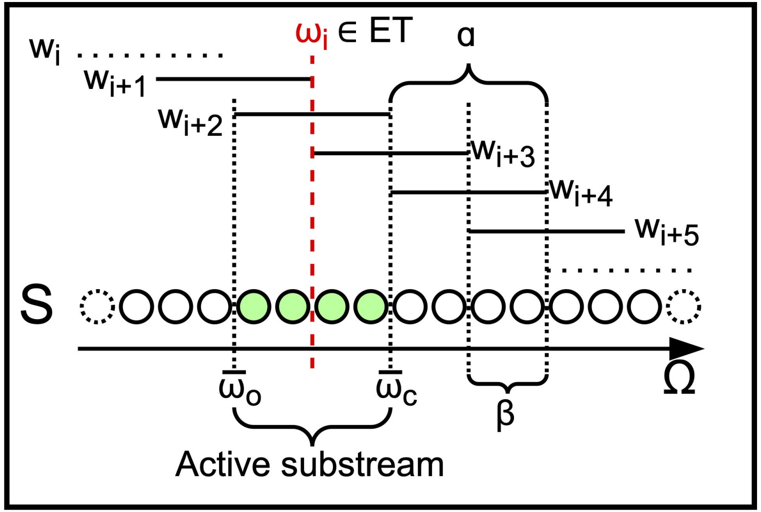
- Tables

r.user_id	s.id	r.val_time	hops
1234	1	14:40	[2,3]
5678	2	14:58	[3,4]

- Property Graph Stream
  - unbounded ordered sequence of pairs (G,ω) where G is a PG and ω a timestamp
- Snapshot Graph
  - Union of all the PGs within a finite sub-portion of a PGStream
- Time-annotated Tables
  - Extend Tables with temporal Bound
- Time-varying Table ($\Psi:\Omega\rightarrow T$)
  - a functional extension of the relational model to incorporate the time semantics

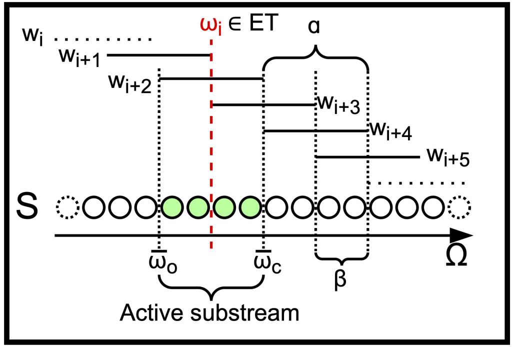
Seraph's Data Model

- Time-varying Table (Ψ:Ω→T)
  - Consistency, i.e., Ψ always identifies a time-annotated table
  - Chronologicality, i.e., Ψ always identifies the time-annotated table with the earliest (minimal) opening timestamp.
  - Monotonicity, i.e., Ψ always identifies subsequent time-annotated tables for subsequent time instants.



Seraph's Semantics

$$
\begin{aligned}
[\![\texttt{RETURN} *]\!]_{\widetilde{G}}(\Psi, \omega) &= \Psi(\omega) \text{ where } \omega \in [\omega_o, \omega_c) \text{, and } \omega_o, \omega_c \text{ are the time annotations of } \Psi(\omega) \\
[\![\texttt{EMIT} *]\!]_{\widetilde{G}}(\Psi, \omega) &= \forall \omega_e \in ET\ [\![\texttt{RETURN} *]\!]_{\widetilde{G}}(\Psi, \omega_e) \text{ a proposal} \\
[\![\texttt{EMIT} * \texttt{ ON ENTERING}]\!]_{\widetilde{G}}(\Psi, \omega) &= [\![\texttt{EMIT} *]\!]_{\widetilde{G}}(\Psi, \omega) \text{, where } \Psi = \{\mu \mid \mu \in \Psi(\omega) \setminus \Psi(\omega - 1)\} \\
[\![\texttt{EMIT} * \texttt{ ON EXIT}]\!]_{\widetilde{G}}(\Psi, \omega) &= [\![\texttt{EMIT} *]\!]_{\widetilde{G}}(\Psi, \omega) \text{, where } \Psi = \{\mu \mid \mu \in \Psi(\omega - 1) \setminus \Psi(\omega)\} \\
[\![\texttt{EMIT} * \texttt{ SNAPSHOT}]\!]_{\widetilde{G}}(\Psi, \omega) &= [\![\texttt{EMIT} *]\!]_{\widetilde{G}}(\Psi, \omega) \text{, where } \Psi = \{\mu \mid \mu \in \Psi(\omega)\} \\
[\![\texttt{WITH} *]\!]_{\widetilde{G}}(\Psi, \omega) &= \Psi(\omega) \text{ if } \Psi(\omega) \text{ has at least one field} \\
[\![\texttt{WITH} \text{ ret } \texttt{WHERE} \text{ expr}]\!]_{\widetilde{G}}(\Psi, \omega) &= \Psi(\omega) \text{ if } \Psi(\omega) \text{ has at least one field} \\
[\![\texttt{STARTING AT } \omega_0 \texttt{ MATCH } \pi \texttt{ WITHIN } \alpha \texttt{ EVERY } \beta]\!]_S &= [\![\texttt{MATCH } \pi]\!]_S^{W(\omega_0, \alpha, \beta)}(\Psi, \omega) \\
&\widehat{=} [\![\texttt{MATCH } \pi]\!]_{W(\omega_0, \alpha, \beta)(S)}(\Psi, \omega) \\
&\widehat{=} [\![\texttt{MATCH } \pi]\!]_{\widetilde{S}_{\omega_c}^{\omega_o}(\omega)}(\Psi, \omega) \\
&\widehat{=} [\![\texttt{MATCH } \pi]\!]_{\widetilde{G}_{\overline{w}}}(\Psi, \omega) \\
&= \biguplus_{\mu \in \Psi(\omega)} \{\mu \cdot \mu' \mid \mu' \in \overline{match}(\pi, \widetilde{G}, \mu)\}
\end{aligned}
$$

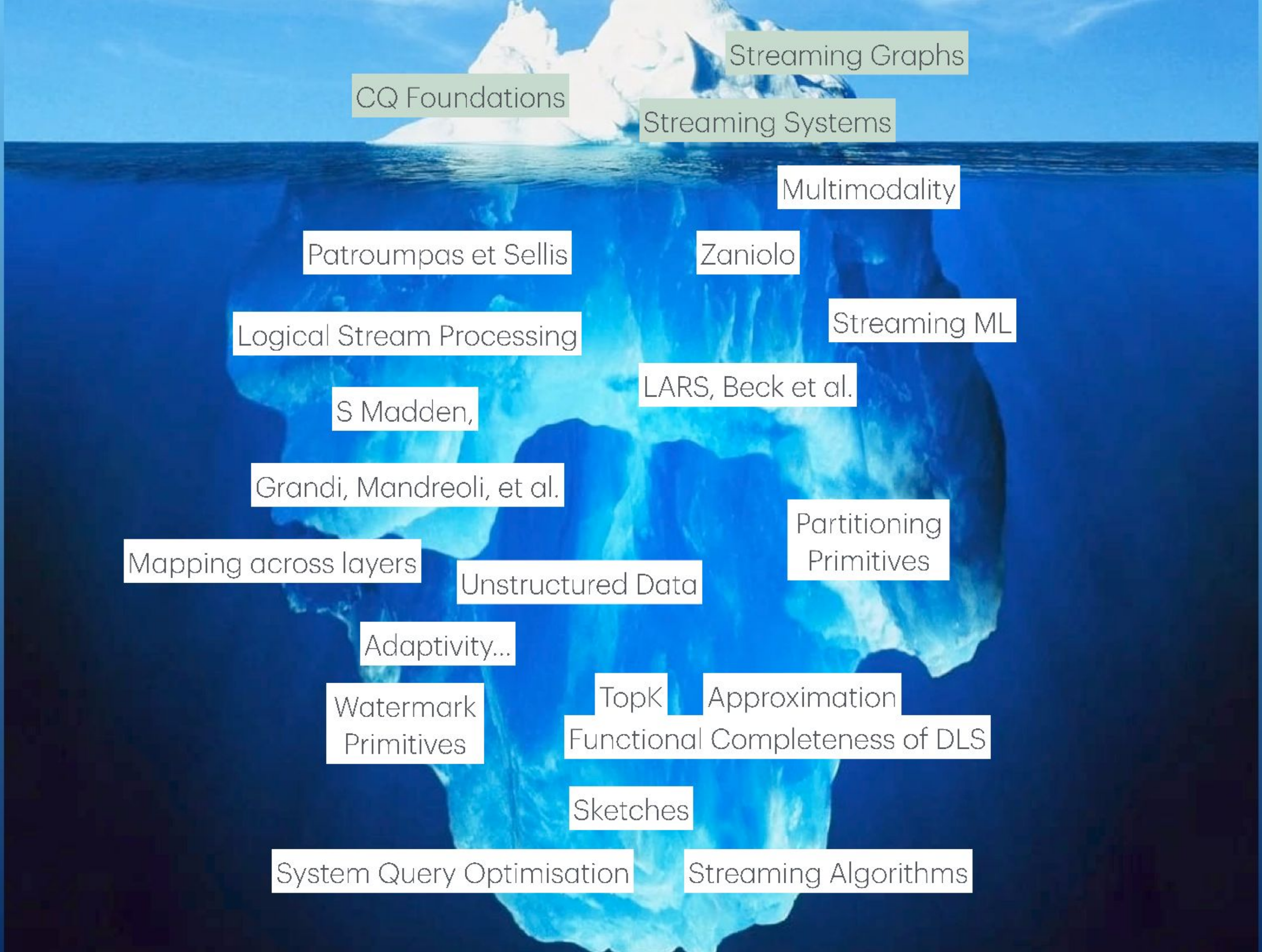


Graph Streaming

is in its infancy

- Need of considering standard graph query languages
- Need of adapting graph query semantics (trail, shortest path etc.)
- Need to make it efficient
- Related problems: quality-aware streaming, fairness-aware streaming

You don’t really know ~~someone~~ your graph until you ~~fight~~ stream them



Streaming Graphs
CQ Foundations
Streaming Systems
Multimodality
Patroumpas et Sellis
Zaniolo
Streaming ML
Logical Stream Processing
LARS, Beck et al.
S Madden,
Grandi, Mandreoli, et al.
Partitioning Primitives
Mapping across layers
Unstructured Data
Adaptivity...
Watermark Primitives
TopK
Approximation
Functional Completeness of DLS
Sketches
System Query Optimisation
Streaming Algorithms

Linguistic Maturity

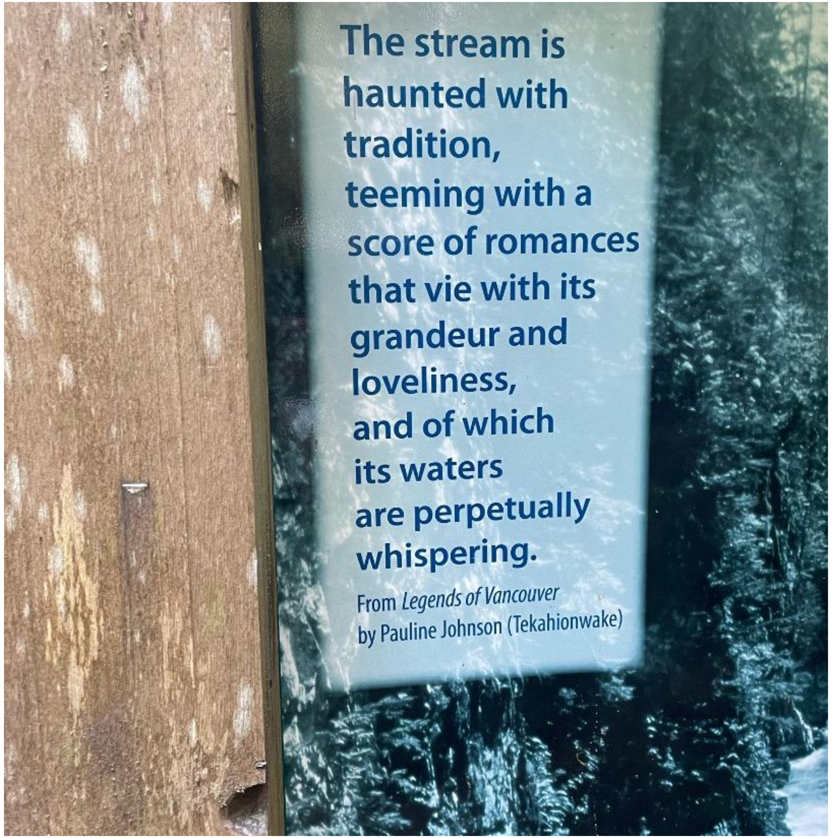
- what are the the fundamental abstractions to enable continuous queries ?
- In the first “Stream Processing Era”, several foundational languages have been presented
  - Terry et al, CQL, Kramer et al. but the list continues.
- Are the language design principles shared?

Query Portability

what are the the fundamental abstractions to enable continuous queries ?

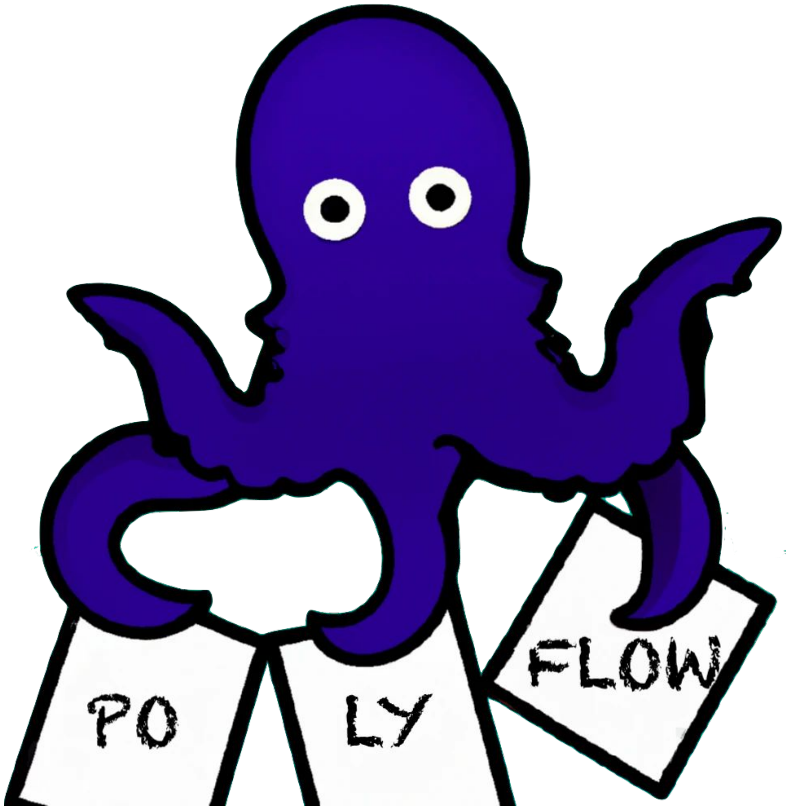
Surprisingly(?), it is not that easy to port continuous queries from a system to another

Intermediate representation like River, DBSP, Brooklet, Arc, go in such direction



Data Complexity

- Users are demanding more and more sophisticated view over data
- In the while data may seems simple, but information needs push for challenging this assumption
- How far can we push such challenge? to what data models SP generalise?
- Can we extend into unstructured multimodality?



Observations

Linguistic Maturity

- “windows” aside, what are the the fundamental abstractions to enable continuous queries ?
- There are evidence of industrial adoption and standardization seems possible
- Despite the variety of language design proposal, such languages are still “domain-specific”

Query Portability

- Streaming System internals remain largely custom, hindering query portability
- Can we dissect the modern and established approaches to uniform them?
- Intermediate representation like River, DBSP, Brooklet, Arc, go in such direction

Data Complexity

- Users are demanding more and more complex view over data
- How far can we push such challenge? to what data models SP generalise?
- How far can we push such challenge? to what data models SP generalise?

Q&A
An Overview of Continuous Querying in (Modern) Data System

Riccardo Tommasini, INSA Lyon, CNRS Liris (France)
Angela Bonifati, Lyon 1 University, CNRS Liris, IUF (France)

Santiago Chile

SIGMOD PODS 2024