# Logical Foundations of Continuous Query Languages for Data Streams

**Carlo Zaniolo**
**Computer Science Department**
**UCLA**
zaniolo@cs.ucla.edu

September 2012

# Data Streams

- Unbounded, rapid, time-varying streams of data elements, continuous flowing on the internet and broad-band
- **Data Stream Management Systems** (DSMS) are designed to process them continuously, with immediate response to new arriving tuples.
- Typical applications involve database like queries. Many adaptations of SQL proposed for continuous queries.
- Continuous (i.e., persistent) queries on transient data, are very different from the transient queries on persistent data creating difficult issues needing better formal models.
  - **blocking queries must be disallowed on data streams**
  - **Previous formal treatments have have focused on streams without time-stamps and proved that for queries:** blocking = non-monotonic !

# The Renaissance of Datalog

- Many DSMS projects were developed during Datalog's Dark Ages, …
- The time has come to revisit data stream query languages with the insights and formal tools provided by logic--surprising results:
  - **Negation is a simpler problem here than in Datalog or Prolog,**
  - **Datalog with minor adjustments becomes a powerful and natural language for data streams.**
- These results hold directly on time-stamped data streams.

# Outline

- Analysis and Design of Logic-based languages for Data streams
  - One time-stamped Data Stream
  - Closed World Assumption (CWA) for data streams.
  - Several time-stamped data streams and the synchronization problem,
  - Streamlog, vs. Datalog and Prolog.

# Time-Stamped Data Streams

A. **Input tuples enter operators in time-stamp order,**
B. **Output of query operators must also be ordered.**

*A stream of messages (ground facts):* **msg(Time, MsgCode)**

*Repeated occurrences of a "red" alarm:*

**repeated(T, X) ← msg(T, X), msg(T0, X), T0 < T .**

**? repeated(T, red)**

*When 'red alarm' occurs at time* **T** *event , an output tuple is produced if the red alarm had also occurred earlier, i.e. at time* **T0 < T.**

# The Importance of Order

*For repeated occurrence of code 'red' we write:* **? repeated(T, red)**

*This is OK:* **repeated(T, X) ← msg(T, X), msg(T0, X), T0 <T.**

*This is not OK:* **repeated(T0, X) ← msg(T, X), msg(T0, X), T0 < T.**

*Thus the* **T0** *event comes first and then when the* **T** *event occurs, an output tuple is produced at once.*

*An immediate response produces out-of-order outputs. Input ($t_1$ a) … ($t_2$ b), … ($t_3$ b), … ($t_4$ a) produces ($t_2$ b) , ($t_1$ a)*

*of course, we do not want wait until we can output tuples in the right order, this would produce a blocking behavior.*

# Progressively Closed World Assumption (PCWA) for Data Streams

- PCWA for a single data stream revises the standard CWA of deductive databases with the provision that the world knowledge is expanding according to the timestamps of the arriving data stream tuples.
- CWA: Once the **p** is not entailed by the given set of facts and Horn rules, then **¬p** can be safely assumed.
- PCWA: Once a **streamfact(T, . . .)** is observed in the input stream, the PCWA allows us to assume **¬streamfact(T0, . . .)** provided that **T0 < T** , and **streamfact(T0, . . .)** is not entailed by the *fact base* augmented with the stream facts having timestamp **< T.**

# Negated Goals

- First occurrence of code red: $?\mathtt{first}(\mathtt{T}, \mathtt{red})$

$$\mathtt{first}(\mathtt{T}, \mathtt{X}) \leftarrow \quad \mathtt{msg}(\mathtt{T}, \mathtt{X}), \neg \mathtt{previous}(\mathtt{T}, \mathtt{X}).$$
$$\mathtt{previous}(\mathtt{T}, \mathtt{X}) \leftarrow \quad \mathtt{msg}(\mathtt{T0}, \mathtt{X}), \mathtt{T0} < \mathtt{T}.$$

This query uses negation on events that, according to their timestamps, are past events. The query can be answered in the present: it is non-blocking.

- Last occurrence of code red: $?\mathtt{last}(\mathtt{T}, \mathtt{red})$

$$\mathtt{last}(\mathtt{T}, \mathtt{Z}) \leftarrow \quad \mathtt{msg}(\mathtt{T}, \mathtt{Z}), \neg \mathtt{next}(\mathtt{T}, \mathtt{Z}).$$
$$\mathtt{next}(\mathtt{T}, \mathtt{Z}) \leftarrow \quad \mathtt{msg}(\mathtt{T1}, \mathtt{Z}), \mathtt{T1} > \mathtt{T}.$$

We do not know if the current red is the last one until we have seen the all stream. Obviously, a **blocking** query. **Thus negation can cause blocking but not always. We must understand when.**

# Sequentiality of Rules & Predicates

*A Sequential rule. The TS of the goals are less or equal than that of the head.*

repeated(T, X) ← msg(T, X), msg(T0, X), T0 < T.

Sequentiality is required for all goals.

**Strict** sequentiality required for negated goals:

$$\begin{aligned} \texttt{first}(\texttt{T},\texttt{X}) &\leftarrow \quad \texttt{msg}(\texttt{T},\texttt{X}), \neg\texttt{previous}(\texttt{T},\texttt{X}). \\ \texttt{previous}(\texttt{T},\texttt{X}) &\leftarrow \quad \texttt{msg}(\texttt{T0},\texttt{X}), \texttt{T0} < \texttt{T}. \end{aligned}$$

*A strictly sequential rule: time-stamp in the head is > than that of every goal. A predicate is strictly sequential when all the rules defining it are strictly sequential.*

# Stratification in Datalog

minpath(X, Y, D) ← path(X, Y, D), ¬shorter(X, Y, D).

shorter(X, Z, D) ← path(X, Z, D1), D1 < D.

path(X, Y, D) ← arc(X, Y, D).

path(X,Z,D) ← path(X,Y,D1), path(Y,Z,D2), D =D1+D2,

- Inefficient computation, since non-minimal paths are eliminated at the end of the recursive iteration, rather than as-soon-as generated.

- More general kinds of stratifications can solve this problem. E.g., XY-stratification, or Statelog, that are based on the introduction of an additional temporal argument—a complication for the users.

- But in Streamlog the temporal argument is already there!!!!!!

# Shortest Path in Streamlog

$$\mathtt{path}(T,X,Y,D) \leftarrow \quad \mathtt{arc}(T,X,Y,D), \neg\mathtt{shorter}(T,X,Y,D).$$
$$\mathtt{shorter}(T,X,Y,D) \leftarrow \mathtt{path}(T',X,Y,D'), T' < T, D' \leq D.$$

$$\begin{aligned}
\mathtt{path}(T,X,Z,Ds) \leftarrow\ & \mathtt{path}(T,X,Y,D), \mathtt{path}(T',Y,Z,D'),\\
& T' < T,\ Ds = D{+}D'.\\
\mathtt{path}(T,X,Z,Ds) \leftarrow\ & \mathtt{path}(T',X,Y,D'), \mathtt{path}(T,Y,Z,D),\\
& T' < T,\ Ds = D'{+}D.\\
\mathtt{path}(T,X,Z,Ds) \leftarrow\ & \mathtt{path}(T',X,Y,D'), \mathtt{path}(T,Y,Z,D),\\
& T' = T,\ Ds = D'{+}D.
\end{aligned}$$

- Arriving arcs are check against previous paths $T' < T,$
- now $\neg\mathtt{shorter}(T,X,Y,D)$ can be added in the last three rules too
- The last three rules can be condensed into one:

$$\begin{aligned}
\mathtt{path}(T3,X,Z,Ds) \leftarrow\ & \mathtt{path}(T2,X,Y,D), \mathtt{path}(T1,Y,Z,D'),\\
& \mathtt{lgr}(T1,T2,T3)\ Ds = D{+}D'.
\end{aligned}$$

# Bistate Version of a Program

1. Rename all the predicates in the body whose temporal argument is less than that of the head by the suffix **_old**

$$\mathtt{path}(T,X,Y,D) \leftarrow \quad \mathtt{arc}(T,X,Y,D), \neg\mathtt{shorter}(T,X,Y,D).$$

2. $\mathtt{shorter}(T,X,Y,D) \leftarrow \mathtt{path}(T',X,Y,D'), T' < T, D' \leq D.$

$$\begin{aligned}
\mathtt{path}(T,X,Z,Ds) &\leftarrow \mathtt{path}(T,X,Y,D), \mathtt{path}(T',Y,Z,D'),\\
&\quad T' < T,\ Ds = D{+}D'.\\
\mathtt{path}(T,X,Z,Ds) &\leftarrow \mathtt{path}(T',X,Y,D'), \mathtt{path}(T,Y,Z,D),\\
&\quad T' < T,\ Ds = D'{+}D.\\
\mathtt{path}(T,X,Z,Ds) &\leftarrow \mathtt{path}(T',X,Y,D'), \mathtt{path}(T,Y,Z,D),\\
&\quad T' = T,\ Ds = D'{+}D.
\end{aligned}$$

The bistate version of the program is stratified: e.g.

- **old_path** and **shorter** at lower stratum and
- **path** at stratum next stratum.

Thus, the original program is locally stratified in the same way.

# Semantics: formal and Operational

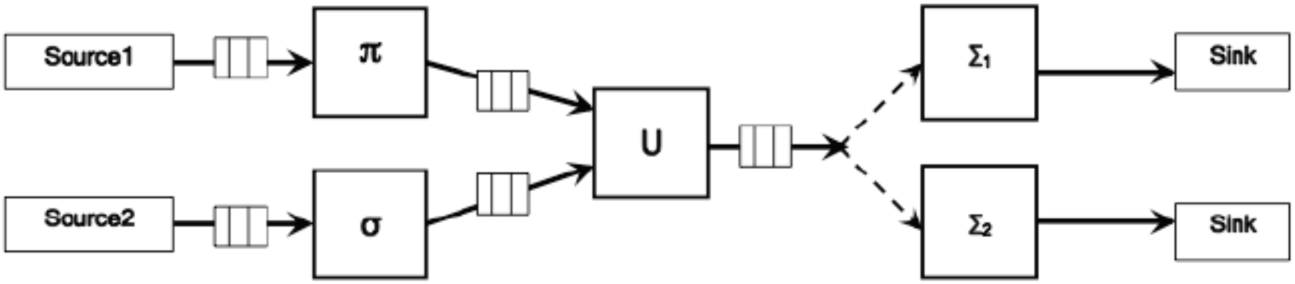
Theorem 1: if the bistate version of the program is stratified then the original program is locally stratified.

Theorem 2: if the original program is strictly sequential then its bistate version is stratified.

Perfect Model of a strictly sequential program is simple to compute:

For each new arriving data stream fact
begin
if the fact has a timestamp larger than that of the previous one, then update the old_ tables;
compute the implications of the new fact according to the stratified bistate version of the program.
end

# Multiple Streams: Unions

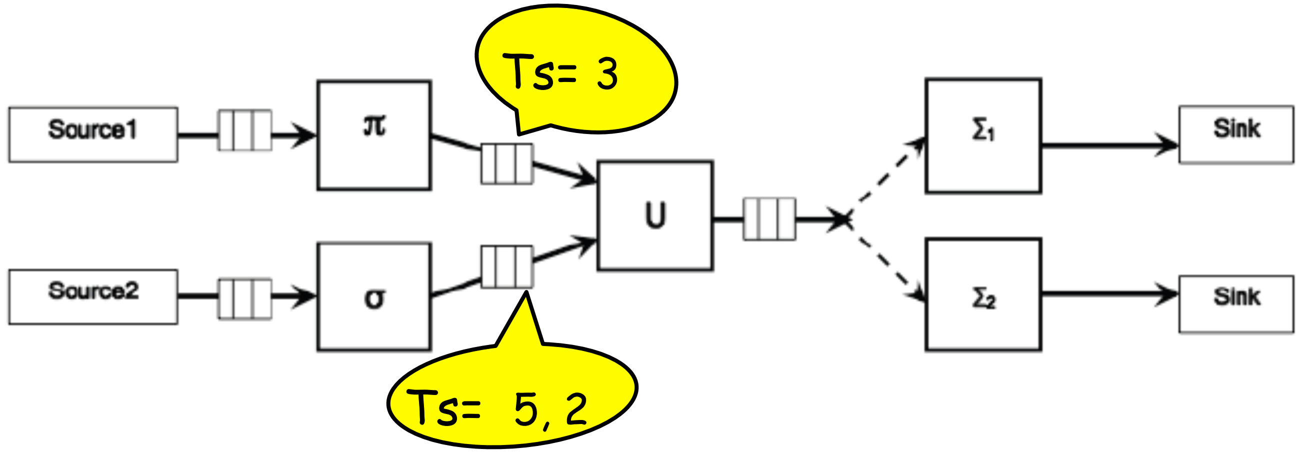



msg(T, S) ← sensr1(T, S).

msg(T, S) ← sensr2(T, S).

- ***On stored data, multiple rules simply define disjunction.***
- ***But on data streams there is also a time-stamp order constraint.***

# Multiple Streams: Unions

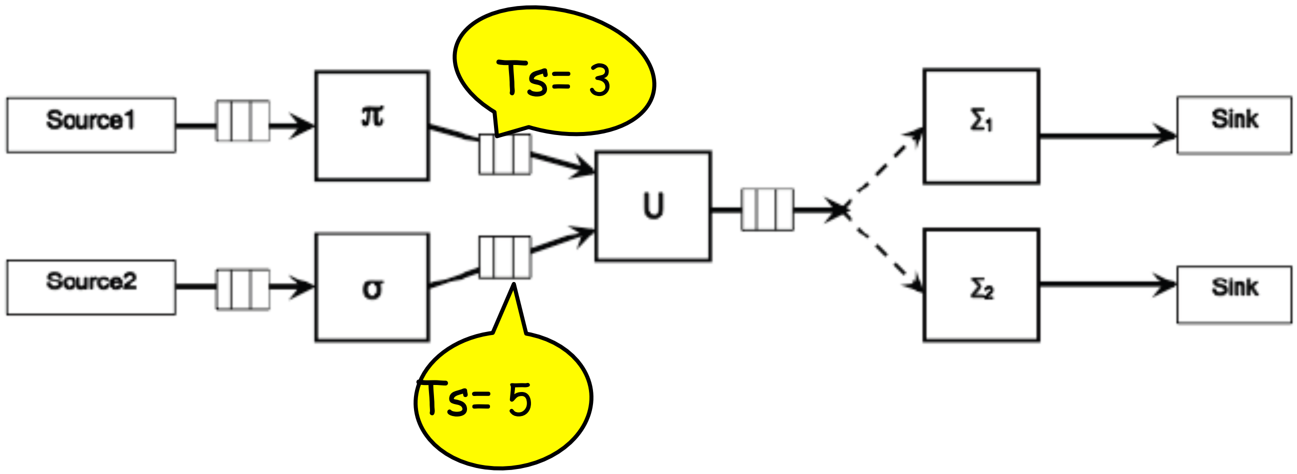



msg(T, S) ← sensr1(T, S).

msg(T, S) ← sensr2(T, S).

When both input buffers have tuples, simply take a tuple that has a mininimal timestamp.

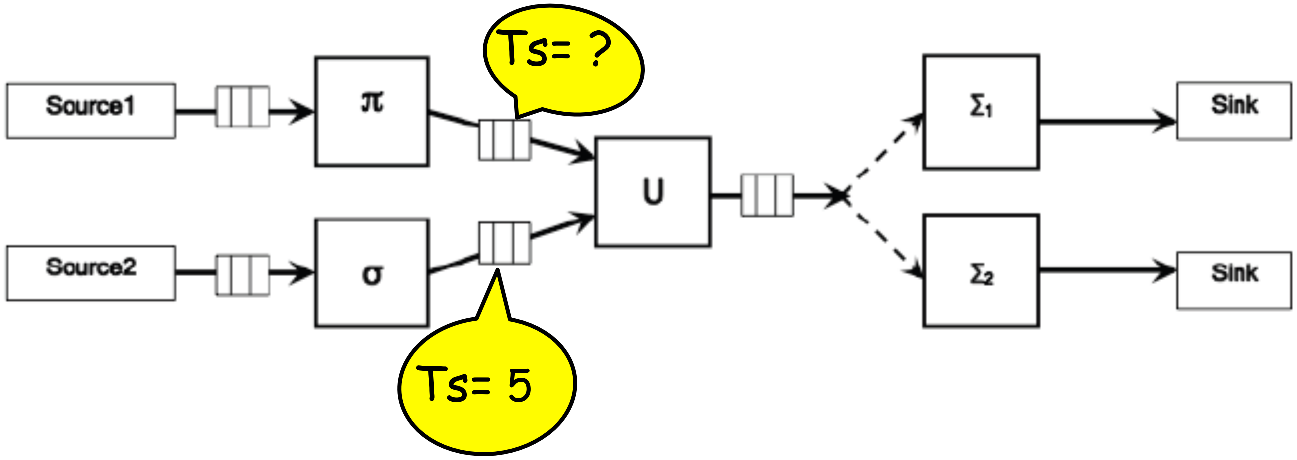
# Multiple Streams: Unions




msg(T, S) ← sensr1(T, S).

msg(T, S) ← sensr2(T, S).

# Multiple Streams: Unions




msg(T, S) ← sensr1(T, S).

msg(T, S) ← sensr2(T, S).

- In order to perform a correct sort-merge, when one of the imput buffer is empty , we must wait until a new tuple arrives.
- This strategy can cause long waits, and stop working when one streams stops.
- System-added punctuation tuples can be used to addres this problem.

# Multiple Streams and Synchronization

**A. The union of two streams:**

$$\mathtt{msg}(\mathtt{T1},\mathtt{S1}) \leftarrow \mathtt{sensr1}(\mathtt{T1},\mathtt{S1}).$$
$$\mathtt{msg}(\mathtt{T2},\mathtt{S2}) \leftarrow \mathtt{sensr2}(\mathtt{T2},\mathtt{S2}).$$

**B. Sort-Merge of two streams:**

$$\mathtt{msg}(\mathtt{T1},\mathtt{S1}) \leftarrow \mathtt{sensr1}(\mathtt{T1},\mathtt{S1}), \mathtt{sensr2}(\mathtt{T2},\_), \mathtt{T2} \geq \mathtt{T1}.$$
$$\mathtt{msg}(\mathtt{T2},\mathtt{S2}) \leftarrow \mathtt{sensr2}(\mathtt{T2},\mathtt{S2}), \mathtt{sensr1}(\mathtt{T1},\_), \mathtt{T1} \geq \mathtt{T2}.$$

**C. Synchronized union of two streams:**

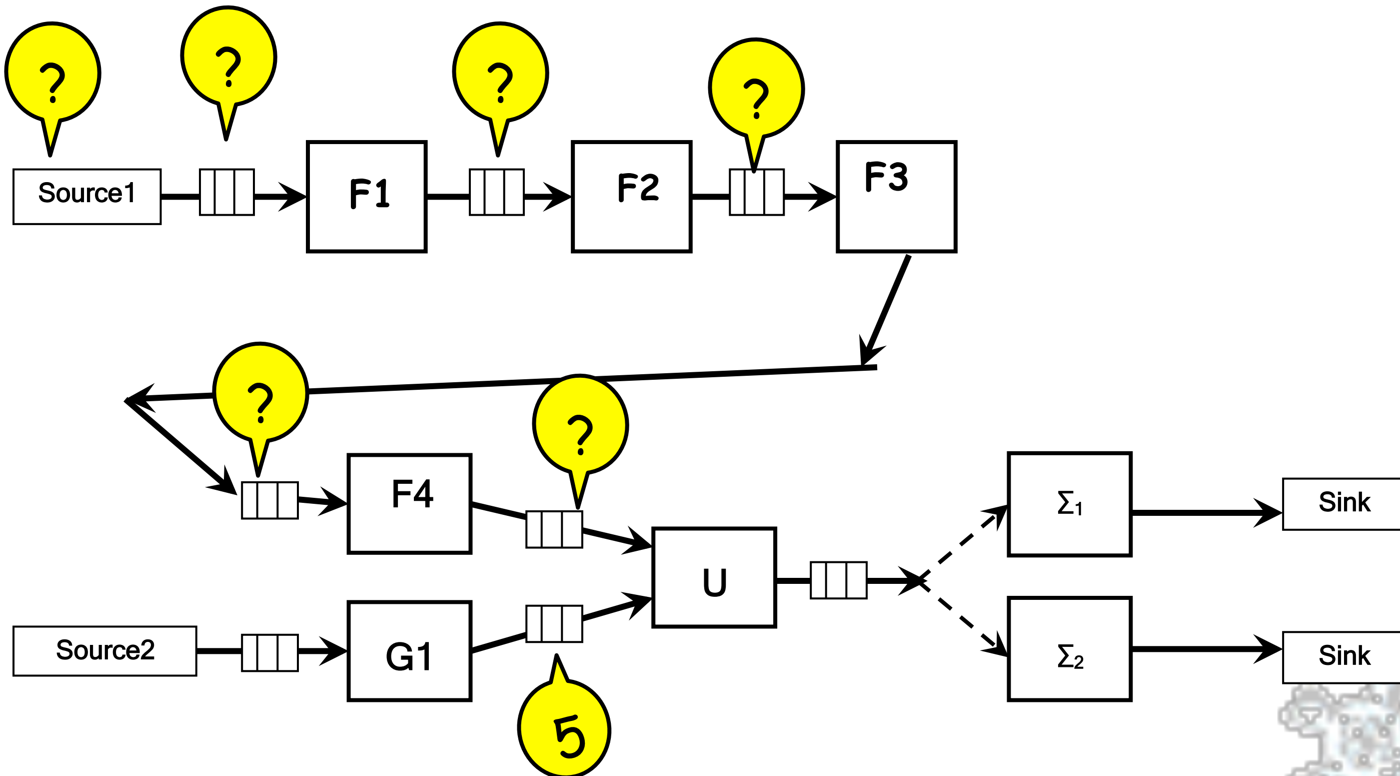
$$\mathtt{msg}(\mathtt{T1},\mathtt{S1}) \leftarrow \mathtt{sensr1}(\mathtt{T1},\mathtt{S1}), \neg \mathtt{missing2}(\mathtt{T1}).$$
$$\mathtt{msg}(\mathtt{T2},\mathtt{S2}) \leftarrow \mathtt{sensr2}(\mathtt{T2},\mathtt{S2}), \neg \mathtt{missing1}(\mathtt{T2}).$$

$$\mathtt{missing2}(\mathtt{T1}) \leftarrow \mathtt{sensr2}(\mathtt{T2},\mathtt{S}), \mathtt{T2} < \mathtt{T1}.$$
$$\mathtt{missing1}(\mathtt{T2}) \leftarrow \mathtt{sensr1}(\mathtt{T1},\mathtt{S}), \mathtt{T1} < \mathtt{T2}.$$

A: what users write.
B: the partially blocking way in which it is often treated now.
C: the proper characterization using negation.

From correct semantics to better implementation:

# Backtracking on Idle Branches



# Minimizing Idle Waiting in Implementation

- Generation of punctuation tuples (carrying enabling time stamps ETS) to unblock idle waiting union operators.
- At regular intervals or, on demand, via backtracking.

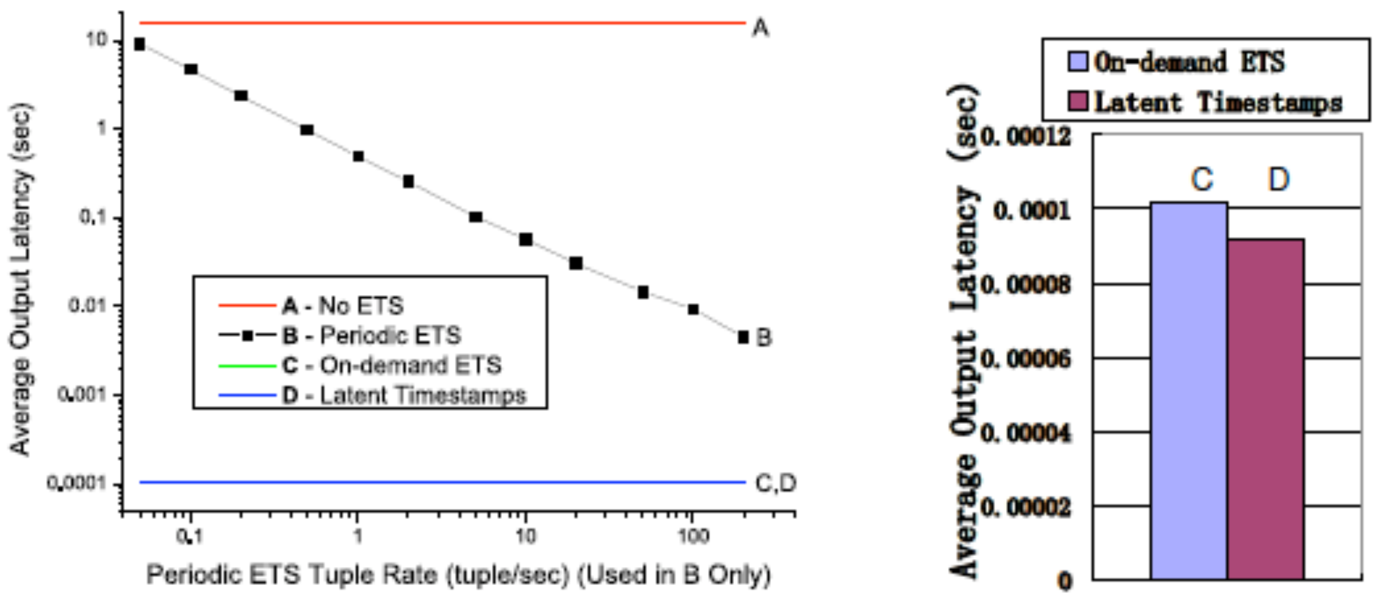



Fig. 2. Average Output Latency

Latent: same as no timestamp

# Conclusion

- Non-monotonic reasoning for data streams can be supported quite naturally and efficiently using simple extensions of Datalog.
- We introduced rigorous logical foundations for continuous query languages.
- These are practical solutions that significantly enhance the expressive power of continuous query languages.
- Streamlog extends Datalog but also benefits from Prolog.
- Current work: data streams without timestamps, and beyond strictly sequential.
- Future directions: a unified language for stored data and data streams: SAUL (Scalable Analytics Unification Language).

# Conclusion

Exciting progress in overcoming disabilities suffered by DSMS query languages in the dark age of our field.

# Thank you!

# References


1. B. Babcock, S. Babu, M. Datar, R. Motawani, and J. Widom. Models and issues in data stream systems.In PODS, 2002.
2. Yijian Bai, Hetal Thakkar, Haixun Wang, Chang Luo, and Carlo Zaniolo. A data stream language andsystem designed for power and extensibility. In CIKM, 2006
3. Yijian Bai, Hetal Thakkar, Haixun Wang, and Carlo Zaniolo. Timestamp management and query execution models in data stream management systems. IEEE Internet Computing, 12(6):13{21, 2008.
4. Yuri Gurevich, Dirk Leinders, and Jan Van den Bussche. A theory of stream queriesDatabase Programming Languages. DBPL 2007.
5. Yan-Nei Law, Haixun Wang, and Carlo Zaniolo. Data models and query language for data streams. In VLDB 2004.
6. Barzan Mozafari, Kai Zeng, and Carlo Zaniolo. From regular expressions to nested words: Unifying languages and query execution forrelational and xml sequences. In VLDB 2010.
7. P.Tucker, D. Maier, and T.Sheard. Applying punctuation schemes to queries over continuous data streams. IEEE Data Engineering Bulletin,26(1):33{40, 2003.
8. Arcot Rajasekar, Jorge Lobo, Jack Minker. Weak generalized closed world assumption. J. Autom. Reasoning, 5(3), 1989.
9. Raymond Reiter. Deductive question-answering on relational data bases. In Herve Gallaire and Jack Minker, editors, Logic and Data Bases, Symposium on Logic and Data Bases, Toulouse, 1977.
10. Hetal Thakkar, Nikolay Laptev, Hamid Mousavi,Barzan Mozafari, Vincenzo Russo, and Carlo Zaniolo.Smm: a data stream management system for knowledge discovery. In ICDE, page 1, 2011.