Streaming Queries Over Streaming Data
S. Chandrasekaran and M. Franklin. In Proc. Int'l Conf. on Very Large Data Bases (VLDB'02). 2002, pp. 203-214.

Typical analysis of streaming data includes monitoring or extracting data from sources such as telephone call records, stock market data, and sports statistics. Databases for these types of info are generally implemented as continuous query (CQ) systems where data is streamed over a fixed query. The main downfall of these systems is that they cannot query data that arrived before the time the query was established, in other words, a query on a CQ system cannot retrieve past data. In contrast, a query on a traditional database management system (DBMS) does not apply to future data from a data stream. This is where PSoup is introduced as a solution that can support queries on streaming data from both before or after the time of the query specification.

PSoup takes two inputs: the streaming data sources, as well as queries from the client(s). PSoup then builds relevant data into table representations called Data State Modules (Data SteMs), and queries into a single Query State Module (Query SteM). PSoup stores results into a table called the Results Structure, in which the columns represent queries, the rows represent data. This table is filled with Boolean values of whether each query matches the given data. When a *new query* is entered into PSoup, an entry is added to the Query SteM, and the entry is probed (evaluated) against Data SteMs. This result is added as a *column* to the Results Structure. When *new data* is entered to PSoup, the Data SteMs are updated, and the Query SteM is probed and the results are inserted as a *row* to the Results Structure.

PSoup is implemented in Java, and consists of a query processor called the PSoup Eddy, which builds off of relevant database literature. Performance of PSoup is compared with NoMat, a CQ system that stores the data stream given a time window, but does materialize the results into a results structure. Test results consistently show PSoup outperforming NoMat, in query response-time by roughly two orders of magnitude when running queries on large amount of tuples.

The authors did a great job in conceptualizing the idea of PSoup. The paper instills PSoup as a CQ system with the data storing capabilities of a traditional DBMS. Figures do a good job of illustrating the idea of a "temporary databases" called SteM's, as a novel approach to materializing only the data that matters, rather than a traditional DBMS which would materialize all the data. The figures in this paper give a clear process of what happens in PSoup when either new queries or new data are given. It is quite eloquent that the paper provides a single figure that can explain PSoup's mechanism for being able to apply new queries to old data, and new data to old queries. PSoup's abilities are also explained in an intuitive manner, first the paper explains a selection query over a single data stream, and then builds on that to explain a selection query over multiple data streams.

It would have been beneficial to see a figure comparing memory storage space needed for the queries for each database implementation. The authors could have probed the Java Virtual Machine for memory usage as the amount of tuples increased, which would illustrate how much more memory PSoup uses compared to NoMat. The more interesting question is how PSoup would scale under an infinite or very large number of data streams. This idea is briefly discussed in the conclusion of the paper, but it is left for future work. Materializing data from infinite data streams would lead to memory issues, which is the same reason DBMSs are not used for data streams. The answer to this question would be the true test for PSoup. If it can be built to handle a very large number of data streams on limited memory, then PSoup would become very practical. This paper is recommended as an early preview of an implementation to handle streaming queries on both past and future data, but does not yet address scaling issues with very large amounts of data streams.