

# It's a StreamER World

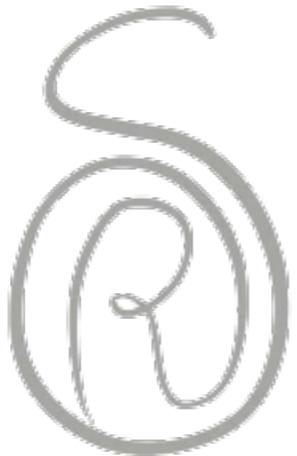
A Journey Through Processing Flows of Data

Paper We Love ✖ Kafka Meetup



**POLITECNICO**  
MILANO 1863





# Who I Am

Emanuele Della Valle

Professor @ Polimi

Inventor of Stream Reasoning

Kafka Enthusiast



POLITECNICO  
MILANO 1863



# Who I Am

Riccardo Tommasini  
PhD Candidate @ Polimi  
Thesis: Velocity on the Web  
Future Assistant Professor @  
University of Tartu, Estonia  
Enthusiast!



# Timeline

Precision not Recall

Models and Issues in Data Stream Systems 3

Department of Computer Science  
St. Olaf College  
Northfield, MN 55057  
<http://www.cs.stolaf.edu/~mowat/>

This survey paper will examine the trend for real resource reuse arising from a wide range of activities. In this respect, the survey will take the form of a broad-based review, for certain sectors in particular (e.g., wastewater reuse), more detailed coverage is provided. In addition to reviewing past trends in reuse systems and selected properties of reuse, the paper also reviews reuse system design principles, performance and examples in quarry processing, agricultural reuse.

## Answers

One of the most common applications for SPC is quality management, applications in which historical data is used to perform statistical analysis on current process status. Examples of such statistical process control include: process control, process reliability, process improvement, manufacturing control systems, and mining. In the data mining model, historical data is used to identify patterns, detect anomalies, predict future events, and support decision making. However, the confidence in all models, such as neural networks, depends on the quality of the training data. Therefore, it is important to obtain high-quality training data.

the system can be used to support the user and the user can respond to the system and specify the required applications. Furthermore, it is important that the application is developed in such a way that it can be used in various environments and for different purposes. In addition, the system should be able to handle different types of data and information, while maintaining consistency and accuracy.

**The 8 Requirements of Real-Time Stream Processing**

Requirement	Description	Notes
1. Data Continuity	Guaranteed delivery of data streams.	Not present in current systems.
2. Low Latency	Delivery of data within milliseconds.	Not present in current systems.
3. High Throughput	Delivery of data at high rates.	Not present in current systems.
4. Scalability	Ability to handle increasing amounts of data.	Present in current systems.
5. Durability	Ability to store data for later retrieval.	Present in current systems.
6. Consistency	Delivery of data in the same order as it was received.	Present in current systems.
7. Fault Tolerance	Ability to handle system failures without loss of data.	Present in current systems.
8. Resource Efficiency	Efficient use of system resources.	Present in current systems.

**ABSTRACT**  
THE EFFECTS OF EASIER BUDGET PROCESSES ON BUDGETARY AND FINANCIAL DECISIONS IN THE STATE GOVERNMENT OF KENYA ARE EXAMINED IN THIS STUDY. THE RESEARCH HYPOTHESIS IS THAT EASIER BUDGET PROCESSES LEAD TO BETTER BUDGETING AND FINANCIAL MANAGEMENT. THE STUDY CONSIDERS THE BUDGETARY AND FINANCIAL DECISIONS AS A WHOLE, AS WELL AS SEPARATELY. THE STUDY FINDS THAT EASIER BUDGET PROCESSES DO NOT HAVE A SIGNIFICANT EFFECT ON THE BUDGETARY DECISIONS. HOWEVER, EASIER BUDGET PROCESSES DO HAVE A SIGNIFICANT EFFECT ON FINANCIAL DECISIONS. THE STUDY FINDS THAT EASIER BUDGET PROCESSES ARE ASSOCIATED WITH BETTER BUDGETARY AND FINANCIAL DECISIONS. THE STUDY FINDS THAT EASIER BUDGET PROCESSES ARE ASSOCIATED WITH BETTER BUDGETARY AND FINANCIAL DECISIONS. THE STUDY FINDS THAT EASIER BUDGET PROCESSES ARE ASSOCIATED WITH BETTER BUDGETARY AND FINANCIAL DECISIONS.

Journal of Machine Learning Research 11 (2010) 1891-1908

Submitted 11/08; Published 6/10

## MIA: Massive Online Analysis

ADRIEN BENOIT

Goeff Holmes

Richard Kirkle

Bernhard Pfahringer

Department of Computer Science

University of Waikato

Hamilton, New Zealand

adrien@cs.waikato.ac.nz

geoff@cs.waikato.ac.nz

richard@cs.waikato.ac.nz

bernhard@cs.waikato.ac.nz

Editor: Michael West

### Abstract

Massive Online Analysis (MOA) is a software environment for implementing algorithms and running experiments for online learning data streams. MOA includes a collection of source code for learning from binary classification data streams as well as for regression. Targets are classification, tree-of-weights and without feature selection at the leaves. MOA supports incremental and batch learning as well as multi-class classification. MOA is released under the GNU General Public License.

**Keywords:** data streams, classification, ensemble methods, java, machine learning software

### 1. Introduction

Online computing is the study and practice of using computing resources efficiently, in order to process or generate a benefit or algorithmic advantage. In the data stream setting, data arrive at high speed, and an algorithm must process them under very strict constraints of space and time.

MOA is an open-source framework for dealing with massive evolving data streams. MOA is related to WEKA, the Waikato Environment for Knowledge Analysis, which is an award-winning open-source Java-based software implementation of a wide range of machine learning methods.

A data stream environment has different requirements from the traditional batch learning setting. The most significant are the following:

**Requirement 1** Process an example at a time, sequentially if only one (at most)

**Requirement 2** Use a limited amount of memory

**Requirement 3** Work in a limited amount of time

Dynamically Scaling Out-of-Order Data Processing

IEEE Xplore Digital Library

DOI 10.1109/TPDS.2018.2839053

Published online in IEEE Transactions on Parallel and Distributed Systems, December 2018, pp. 1–16.

© 2018 IEEE. Personal use of this material is permitted. However, permission to reprint or republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution must be obtained from IEEE.

For more information, please refer to our Terms & Conditions at [http://www.ieee.org/publications\\_standards/terms\\_conditions.html](http://www.ieee.org/publications_standards/terms_conditions.html).

Anneke Elisa<sup>TM</sup>, Stevens and Batch Processing in a Simic Engine

<b>Palo Alto</b> Palo Alto Unified School District	<b>Stephen Everett</b> Nielsen Worldwide	<b>Jeffrey Sardell</b> Kestrel Investors
<b>WTHA</b> WTHA 100% Student <a href="http://www.wtha.org">www.wtha.org</a>	<b>Uma Arora</b> <a href="http://www.umaa.com">www.umaa.com</a>	<b>TJ Horwitz</b> TJH Consulting LLC <a href="http://www.tjhconsulting.com">www.tjhconsulting.com</a>

**Apache Flink**<sup>2</sup> is an open-source system for processing streaming and batch data. Flink is built on the philosophy that most systems of data processing applications, including real-time ones, can benefit from a unified architecture for both batch and stream processing. This allows for a single system to support both types of analysis (batch and stream) and to reuse the same physical execution engines for both batch and stream processing. In this paper we present Flink's architecture and explain how its (somewhat diverse) set of use cases can be unified under a single execution model.

**2 Introduction**  
Data-stream processing (e.g., as exemplified by complex-event processing systems and static check) data processing systems have been around for decades. However, they have only recently become prominent in the context of real-time systems and applications. They were programmed using different programming models and APIs, and were used by different domains (e.g., indicated streaming systems such as Apache Storm, IBM InfoSphere Streams, Microsoft Stream, or StreamBase), resulting in different data formats and execution engines for Web, mobile, machine learning, big data, and Apache Flink. Today, data has analysis made up for the big three: batch (one-use case, one time, and state), while streaming data analysis needs specialized applications.

(b) Increasingly more data appears over time, but a homogeneous or today's large-scale data processing system is not able to handle it. Data streams are often unstructured and heterogeneous. They can be generated from, for example from web logs, application log, sensors, etc. as changes in application-state introduce discontinuous log entries. Rather than treating the data as discrete events, today's systems ignore the continuous and timely nature of data-generation. Instead, data is received in batches artificially. Each batch has data that is (e.g., a day or memory capacity); one new generation in a time step. This approach is used in many data management, machine learning, and data mining systems. The problem is that batches, however, do not support the continuous and timely nature of data-generation. Techniques such as the "batched stream" [23] approach break up streaming data into micro-batches and then process them sequentially. This approach is useful for memory-constrained, and a batch-on-the-fly for accurate results. All these approaches suffer from high latency (imposed by batches).

The writes of records to a increasing jobs are long—  
the continuously or more event streams, the application logic on producing derived output potentially writing compact

Strained Strumplings: Deformation Within Rock-Fill Assemblies in a Spur-and-Groove System	
Author:	Robert A. Hougham, Michael J. McPhee, and James R. Weller
Organization:	Geotechnical and Geoenvironmental Engineering Department, University of Alberta, Edmonton, Alberta, Canada T6G 2G8
Keywords:	spur-and-groove system, rockfill, strumpling, deformation, soil mechanics, geotechnical engineering
Abstract:	This paper presents results from a series of laboratory tests conducted on a model spur-and-groove system. The system was constructed using a coarse-grained rockfill material and a fine-grained backfill material. The tests were conducted to determine the effect of the system on the deformation characteristics of the rockfill material. The results show that the system has a significant effect on the deformation characteristics of the rockfill material. The system causes the rockfill material to deform more than it would if it were not part of the system. This is due to the fact that the system provides a restraint to the rockfill material, which causes it to deform more than it would if it were not restrained.
Introduction:	The spur-and-groove system is a type of rockfill system that is used in the construction of dams and other structures. The system consists of a coarse-grained rockfill material and a fine-grained backfill material. The system is designed to provide a restraint to the rockfill material, which causes it to deform more than it would if it were not restrained. This is due to the fact that the system provides a restraint to the rockfill material, which causes it to deform more than it would if it were not restrained.
Methodology:	The tests were conducted using a model spur-and-groove system. The system was constructed using a coarse-grained rockfill material and a fine-grained backfill material. The system was designed to provide a restraint to the rockfill material, which causes it to deform more than it would if it were not restrained. This is due to the fact that the system provides a restraint to the rockfill material, which causes it to deform more than it would if it were not restrained.
Results:	The results show that the system has a significant effect on the deformation characteristics of the rockfill material. The system causes the rockfill material to deform more than it would if it were not part of the system. This is due to the fact that the system provides a restraint to the rockfill material, which causes it to deform more than it would if it were not restrained.
Conclusion:	The results show that the system has a significant effect on the deformation characteristics of the rockfill material. The system causes the rockfill material to deform more than it would if it were not part of the system. This is due to the fact that the system provides a restraint to the rockfill material, which causes it to deform more than it would if it were not restrained.

#### **→ Slides of the Same C**

se dicono appartenenze e similità con un concetto più costitutivo quanto non

processing of high-level state transitions at least the state transition parameters provide some information about the system's future behavior. This can facilitate better automation. Furthermore, it is interesting to compare different fragments of the same system's history during diagnosis [11].

The Rev. Dr. J.  
J. C. Rau.  
1870-1871.

# The Origin

## Models and Issues in Data Stream Systems \*

Brian Babcock Shivnath Babu Mayur Datar Rajeev Motwani Jennifer Widom

Department of Computer Science  
Stanford University  
Stanford, CA 94305

{babcock, shivnath, datar, rajeev, widom}@cs.stanford.edu

### Abstract

In this overview paper we motivate the need for and research issues arising from a new model of data processing. In this model, data does not take the form of persistent relations, but rather arrives in multiple, continuous, rapid, time-varying *data streams*. In addition to reviewing past work relevant to data stream systems and current projects in the area, the paper explores topics in stream query languages, new requirements and challenges in query processing, and algorithmic issues.

## 1 Introduction



# The Vision

## The 8 Requirements of Real-Time Stream Processing

Michael Stonebraker

Computer Science and Artificial  
Intelligence Laboratory, M.I.T., and  
StreamBase Systems, Inc.

[stonebraker@csail.mit.edu](mailto:stonebraker@csail.mit.edu)

Uğur Çetintemel

Department of Computer Science,  
Brown University, and  
StreamBase Systems, Inc.

[ugur@cs.brown.edu](mailto:ugur@cs.brown.edu)

Stan Zdonik

Department of Computer Science,  
Brown University, and  
StreamBase Systems, Inc.

[sbz@cs.brown.edu](mailto:sbz@cs.brown.edu)

### ABSTRACT

Applications that require real-time processing of high-volume data streams are pushing the limits of traditional data processing infrastructures. These stream-based applications include market feed processing and electronic trading on Wall Street, network and infrastructure monitoring, fraud detection, and command and control in military environments. Furthermore, as the “sea change” caused by cheap micro-sensor technology takes hold, we expect to see everything of material significance on the planet get “sensor-tagged” and report its state or location in real time. This sensorization of the real world will lead to a “green field” of novel monitoring and control applications with high-volume and low-latency processing requirements.

Recently, several technologies have emerged—including off-the-shelf stream processing engines—specifically to address the challenges of processing high-volume, real-time data without requiring the use of custom code. At the same time, some existing software technologies, such as main memory DBMSs and rule engines, are also being “repurposed” by marketing departments to address these applications.

In this paper, we outline eight requirements that a system software should meet to excel at a variety of real-time stream processing applications. Our goal is to provide high-level guidance to information technologists so that they will know what to look for when evaluating alternative stream processing solutions. As such,

Similar requirements are present in monitoring computer networks for denial of service and other kinds of security attacks. Real-time fraud detection in diverse areas from financial services networks to cell phone networks exhibits similar characteristics. In time, process control and automation of industrial facilities, ranging from oil refineries to corn flakes factories, will also move to such “firehose” data volumes and sub-second latency requirements.

There is a “sea change” arising from the advances in micro-sensor technologies. Although RFID has gotten the most press recently, there are a variety of other technologies with various price points, capabilities, and footprints (e.g., mote [1] and Lojack [2]). Over time, this sea change will cause everything of material significance to be sensor-tagged to report its location and/or state in real time.

Military has been an early driver and adopter of wireless sensor network technologies. For example, the US Army has been investigating putting vital-signs monitors on all soldiers. In addition, there is already a GPS system in many military vehicles, but it is not connected yet into a closed-loop system. Using this technology, the army would like to monitor the position of all vehicles and determine, in real time, if they are off course.

Other sensor-based monitoring applications will come over time in non-military domains. Tagging will be applied to customers at amusement parks for ride management and prevention of lost





Arvind Arasu · Shivnath Babu · Jennifer Widom

## The CQL continuous query language: semantic foundations and query execution

Received: 7 June 2004 / Accepted: 22 November 2004 / Published online: 22 July 2005  
© Springer-Verlag 2005

**Abstract** *CQL*, a *continuous query language*, is supported by the STREAM prototype data stream management system (DSMS) at Stanford. CQL is an expressive SQL-based declarative language for registering continuous queries against streams and stored relations. We begin by presenting an abstract semantics that relies only on “black-box” mappings among streams and relations. From these mappings we define a precise and general interpretation for continuous queries. CQL is an instantiation of our abstract semantics using SQL to map from relations to relations, window specifications derived from SQL-99 to map from streams to relations, and three new operators to map from relations to streams. Most of the CQL language is operational in the STREAM system. We present the structure of CQL’s query execution plans as well as details of the most important components: operators, interoperator queues, synopses, and sharing of components among multiple operators and queries. Examples throughout the paper are drawn from the *Linear Road* benchmark recently proposed for DSMSs. We also curate a public repository of data stream applications that includes a wide variety of queries expressed in CQL. The relative ease of capturing these applications in CQL is

[2, 19, 20, 23, 28, 32]. However, these queries tend to be simple and primarily for illustration – a precise language semantics, particularly for more complex queries, often is left unclear. Furthermore, very little has been published to date covering execution details of general-purpose continuous queries. In this paper we present the *CQL* language and execution engine for general-purpose continuous queries over streams and stored relations. CQL (for *continuous query language*) is an instantiation of a precise abstract continuous semantics also presented in this paper, and CQL is implemented in the *STREAM* prototype data stream management system (DSMS) at Stanford.<sup>1</sup>

It may appear initially that defining a continuous query language over (relational) streams is not difficult: take a relational query language, replace references to relations with references to streams, register the query with the stream processor, and wait for answers to arrive. For simple monotonic queries over complete stream histories, indeed this approach is nearly sufficient. However, as queries get more complex – when we add aggregation, subqueries, windowing constructs, relations mixed with streams, etc. – the situation becomes much murkier. Consider the following simple query:

# Timeline

## Precision not Recall

### Models and Issues in Data Stream Systems \*

Brian Babcock Shivnath Babu Mayur Datar Rajeev Motwani Jennifer Widom  
Department of Computer Science  
Stanford University  
Stanford, CA 94305  
[babcock,shivnath,datar,rajeev,widom]@cs.stanford.edu

#### Abstract

In this overview paper we motivate the need for and research issues arising from a new model of data processing. In this model, data does not take the form of persistent relations, but rather arrives in multiple, continuous, rapid, time-varying *data streams*. In addition to reviewing past work relevant to data stream systems and current projects in the area, the paper explores topics in stream query languages, new requirements and challenges in query processing, and algorithmic issues.

#### 1 Introduction

Recently a new class of data-intensive applications has become widely recognized: applications in which the data is modeled best not as persistent relations, but rather as *transient data streams*. Examples of such applications include financial trading, network monitoring, security, telecommunications data management, sensor reading, and so on. In addition to reviewing past work relevant to data stream systems and current projects in the area, the paper explores topics in stream query languages, new requirements and challenges in query processing, and algorithmic issues.

In all of the applications cited above, it is not feasible to simply load the arriving data into a traditional database management system (DBMS) and operate on it there. Traditional DBMS's are not designed for rapid and continuous loading of individual items, and they do not directly support the *continuous queries* [84] that are typical of stream applications. Furthermore, it is recognized that both *approximation* [13] and *sampling* [10] are required for processing queries and performing other processing (e.g., data analysis and mining) over rapid data streams, while traditional DBMS's focus largely on the opposite goal of precise answers computed by static query plans.

In this paper we consider fundamental models and issues in developing a general-purpose *Data Stream Management System* (DSMS). We are developing such a system at Stanford [82], and we will touch on some of our own work in this paper. However, we also attempt to provide a general overview of the area, along with its related and current work. (Any glaring omissions are, naturally, our own fault.)

We begin in Section 2 by considering the data stream model and queries over streams. In this section we

\*Accepted: 22 November 2004 / Published online: 22 July 2005

© 2005 IEEE. Reprinted with permission from the IEEE.

1063-6923/05/0100-0121-10 \$25.00 © 2005 IEEE

PAPER

#### L continuous query language: semantic foundations by execution

Shivnath Babu - Jennifer Widom

Journal of Web Semantics 2005.6(1):121-142

121

ISSN 1570-1526

DOI 10.1016/j.websem.2005.01.001

Editorial

# Stream Processing + AI (Deductive)

Brian



In the  
data pro-  
multiple  
data struc-  
new req-

1 Introduct-

Recently a new  
the data is me-  
applications in  
and ap-  
data items and  
ings, and so o-  
and unbound-

In all of t-

real-time  
for rapid  
queries [84] d-

data and th-

goal of preci-

in this pa-

Management!

of our own wa-

with its related and current work. (Any glaring omissions are, naturally, our own fault.)

We begin in Section 2 by considering the data stream model and queries over streams. In this section we

pay

attention to

the

and

work.

2 Stream Model

and

queries

over

streams

and

stream

model

and

queries

over

streams

and

stream

model

and

queries

over

streams

and

stream

model

and

queries

over

streams

and

stream

model

and

queries

over

streams

and

stream

model

and

queries

over

streams

and

stream

model

and

queries

over

streams

and

stream

model

and

queries

over

streams

and

stream

model

and

queries

over

streams

and

stream

model

and

queries

over

streams

and

stream

model

and

queries

over

streams

and

stream

model

and

queries

over

streams

and

stream

model

and

queries

over

streams

and

stream

model

and

queries

over

streams

and

stream

model

and

queries

over

streams

and

stream

model

and

queries

over

streams

and

stream

model

and

queries

over

streams

and

stream

model

and

queries

over

streams

and

stream

model

and

queries

over

streams

and

stream

model

and

queries

over

streams

and

stream

model

and

queries

over

streams

and

stream

model

and

queries

over

streams

and

stream

model

and

queries

over

streams

and

stream

model

and

queries

over

streams

and

stream

model

and

queries

over

streams

and

stream

model

and

queries

over

streams

and

stream

model

and

queries

over

streams

and

stream

model

and

queries

over

streams

and

stream

model

and

queries

over

streams

and

stream

model

and

queries

over

streams

and

stream

model

and

queries

over

streams

and

stream

model

and

queries

over

streams

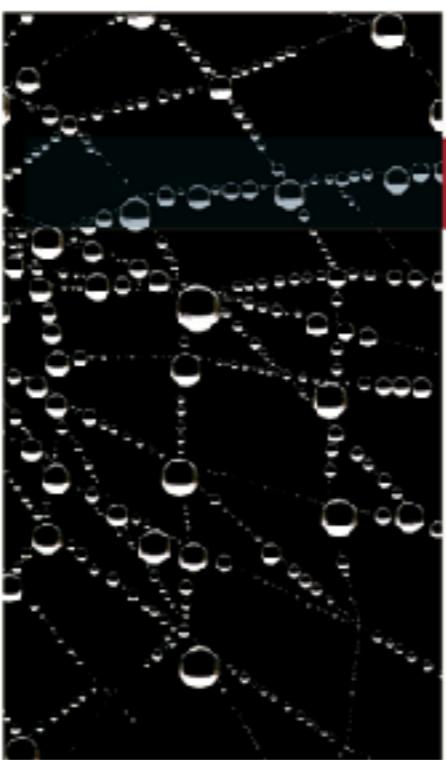
and

stream

model

</div

# The Title



## THE SEMANTIC WEB

Editor: Steffen Staab, University of Koblenz-Landau, staab@uni-koblenz.de

# It's a Streaming World! Reasoning upon Rapidly Changing Information

Emanuele Della Valle and Stefano Ceri, Politecnico di Milano

Frank van Harmelen, Vrije Universiteit Amsterdam

Dieter Fensel, University of Innsbruck

**W**ill there be a traffic jam on this highway? Can we reroute travelers on the basis of the forecast? By examining the clickstream from a given IP, can we discover shifts in interest of the person behind the computer? Which content on the news Web portal is attracting the most attention? Which navigation pattern would lead readers to other news related to that content? Do trends in medical records indicate any new disease spreading in a given part of the world? Where are all my

The state of the art in reasoning over changing worlds is based on temporal logic and belief revision; these are heavyweight tools, suitable for data that changes in low volumes at low frequency. Similarly, the problem of changing vocabularies and evolving ontologies has undergone thorough investigation, but here the standard practice relies on configuration management techniques taken from software engineering, such as vocabulary and ontology versioning. These are suitable for ontologies that change on a weekly or monthly basis, but not



# Timeline

## Precision not Recall

### Models and Issues in Data Stream Systems \*

Brian Babcock Shivnath Babu Mayur Datar Rajeev Motwani Jennifer Widom  
Department of Computer Science Stanford University  
Stanford, CA 94305 [babcock,shivnath,datar,rajeev,widom]@cs.stanford.edu

#### Abstract

In this overview paper we motivate the need for and research issues arising from a new model of data processing. In this model, data does not take the form of persistent relations, but rather arrives in multiple, continuous, rapid, time-varying *data streams*. In addition to reviewing past work relevant to data stream systems and current projects in the area, the paper explores topics in stream query languages, new requirements and challenges in query processing, and algorithmic issues.

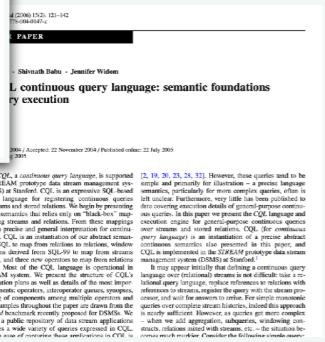
#### 1 Introduction

Recently a new class of data-intensive applications has become widely recognized: applications in which the data is modeled best not as persistent relations, but rather as *tensest data streams*. Examples of such applications include financial applications, network monitoring, security, telecommunications data management, and sensor networks. In these applications, the data is modeled as a stream model; individual data items may be relational tuples, e.g., network measurements, call records, web page visits, sensor readings, and so on. However, their continuous arrival in multiple, rapid, time-varying, possibly unpredictable and unbounded streams appears to yield some fundamentally new research problems.

In all of the applications cited above, it is not feasible to simply load the arriving data into a traditional database management system (DBMS) and operate on it there. Traditional DBMSs are not designed for rapid and continuous loading of individual items, and they do not directly support the *continuous queries* [84] that are typical of stream applications. Furthermore, it is recognized that both *approximation* [13] and *stream mining* [10] are needed for answering queries and performing other processing (e.g., data analysis and mining) over rapid data streams, while traditional DBMSs focus largely on the opposite goal of precise answers computed by stable query plans.

In this paper we consider fundamental models and issues in developing a general-purpose *Data Stream Management System* (DSMS). We are developing such a system at Stanford [82], and we will touch on some of our own work in this paper. However, we also attempt to provide a general overview of the area, along with its related and current work. (Any glaring omissions are, naturally, our own fault.)

We begin in Section 2 by considering the data stream model and queries over streams. In this section we



**Abstract**  
CQL: A continuous query language is proposed by the STREAM research group at Stanford. CQL is an expressive SQL-based declarative language for continuous queries over streams and stored relations. We begin by presenting an alternative semantics that relies only on “stateless” query processing. Then we propose a more expressive semantics that includes a “stateful” component. We define a precise and general interpretation for continuous queries over streams and stored relations. CQL is far more expressive than SQL, to map solutions to relations, while supporting complex temporal operators, such as windowing, joins, and three new operators to map them relationally. CQL is designed to be used in conjunction with the STREAM system. We present the structure of CQL’s query language, and the semantics of its components. The most important components are temporal operators, interpreted queries, complex, and stateless operators. Examples and benchmarks are drawn from the Linear Rulebook recently proposed for DSMSs. We also show a prototype implementation of CQL that includes a wide variety of queries expressed in CQL. The system is currently being developed at Stanford.

#### 2 Requirements

##### 2.1

##### 2.2

##### 2.3

##### 2.4

##### 2.5

##### 2.6

##### 2.7

##### 2.8

##### 2.9

##### 2.10

##### 2.11

##### 2.12

##### 2.13

##### 2.14

##### 2.15

##### 2.16

##### 2.17

##### 2.18

##### 2.19

##### 2.20

##### 2.21

##### 2.22

##### 2.23

##### 2.24

##### 2.25

##### 2.26

##### 2.27

##### 2.28

##### 2.29

##### 2.30

##### 2.31

##### 2.32

##### 2.33

##### 2.34

##### 2.35

##### 2.36

##### 2.37

##### 2.38

##### 2.39

##### 2.40

##### 2.41

##### 2.42

##### 2.43

##### 2.44

##### 2.45

##### 2.46

##### 2.47

##### 2.48

##### 2.49

##### 2.50

##### 2.51

##### 2.52

##### 2.53

##### 2.54

##### 2.55

##### 2.56

##### 2.57

##### 2.58

##### 2.59

##### 2.60

##### 2.61

##### 2.62

##### 2.63

##### 2.64

##### 2.65

##### 2.66

##### 2.67

##### 2.68

##### 2.69

##### 2.70

##### 2.71

##### 2.72

##### 2.73

##### 2.74

##### 2.75

##### 2.76

##### 2.77

##### 2.78

##### 2.79

##### 2.80

##### 2.81

##### 2.82

##### 2.83

##### 2.84

##### 2.85

##### 2.86

##### 2.87

##### 2.88

##### 2.89

##### 2.90

##### 2.91

##### 2.92

##### 2.93

##### 2.94

##### 2.95

##### 2.96

##### 2.97

##### 2.98

##### 2.99

##### 2.100

##### 2.101

##### 2.102

##### 2.103

##### 2.104

##### 2.105

##### 2.106

##### 2.107

##### 2.108

##### 2.109

##### 2.110

##### 2.111

##### 2.112

##### 2.113

##### 2.114

##### 2.115

##### 2.116

##### 2.117

##### 2.118

##### 2.119

##### 2.120

##### 2.121

##### 2.122

##### 2.123

##### 2.124

##### 2.125

##### 2.126

##### 2.127

##### 2.128

##### 2.129

##### 2.130

##### 2.131

##### 2.132

##### 2.133

##### 2.134

##### 2.135

##### 2.136

##### 2.137

##### 2.138

##### 2.139

##### 2.140

##### 2.141

##### 2.142

##### 2.143

##### 2.144

##### 2.145

##### 2.146

##### 2.147

##### 2.148

##### 2.149

##### 2.150

##### 2.151

##### 2.152

##### 2.153

##### 2.154

##### 2.155

##### 2.156

##### 2.157

##### 2.158

##### 2.159

##### 2.160

##### 2.161

##### 2.162

##### 2.163

##### 2.164

##### 2.165

##### 2.166

##### 2.167



## MOA: Massive Online Analysis

**Albert Bifet**

ABIFET@CS.WAIKATO.AC.NZ

**Geoff Holmes**

GEOFF@CS.WAIKATO.AC.NZ

**Richard Kirkby**

RKIRKBY@CS.WAIKATO.AC.NZ

**Bernhard Pfahringer**

BERNHARD@CS.WAIKATO.AC.NZ

*Department of Computer Science*

*University of Waikato*

*Hamilton, New Zealand*

**Editor:** Mikio Braun

### Abstract

Massive Online Analysis (MOA) is a software environment for implementing algorithms and running experiments for online learning from evolving data streams. MOA includes a collection of offline and online methods as well as tools for evaluation. In particular, it implements boosting, bagging, and Hoeffding Trees, all with and without Naïve Bayes classifiers at the leaves. MOA supports bi-directional interaction with WEKA, the Waikato Environment for Knowledge Analysis, and is released under the GNU GPL license.

**Keywords:** data streams, classification, ensemble methods, java, machine learning software

### 1. Introduction

*Green computing* is the study and practice of using computing resources efficiently. A main approach to green computing is based on algorithmic efficiency. In the data stream model, data arrive at high speed, and an algorithm must process them under very strict constraints of space and time.

MOA is an open-source framework for dealing with massive evolving data streams. MOA is related to WEKA, the Waikato Environment for Knowledge Analysis, which is an award-winning open-source workbench containing implementations of a wide range of batch machine learning





# Our Focus

## Streams and Tables: Two Sides of the Same Coin

Matthias J. Sax\*  
Confluent Inc.  
Palo Alto, USA  
matthias@confluent.io

Matthias Weidlich  
Humboldt-Universität zu Berlin  
Berlin, Germany  
matthias.weidlich@hu-berlin.de

Guozhang Wang  
Confluent Inc.  
Palo Alto, USA  
guozhang@confluent.io

Johann-Christoph Freytag  
Humboldt-Universität zu Berlin  
Berlin, Germany  
freytag@informatik.hu-berlin.de

### TRACT

processing has emerged as a paradigm for applications that require low-latency evaluation of operators on unbounded sequences of data. Defining the semantics of stream processing is challenging in the presence of distributed data sources. The physical and logical order of data in a stream may become inconsistent in such a setting. Existing models either neglect these inconsistencies or handle them by means of data buffering and reordering techniques, thereby compromising processing latency.

In this paper, we introduce the Dual Streaming Model, a model that distinguishes between physical and logical order in data streams. This model presents the result of an operator as a sequence of successive updates, which induces a duality of streams and tables. As such, it provides a natural way to cope with inconsistencies between the physical and logical order of data by processing data in a continuous manner, without explicitly buffering and reordering. We further discuss the trade-offs and challenges faced when implementing this model in terms of correctness, latency, and processing cost. A case study on Apache Kafka illustrates the effectiveness of our model in the light of real-world requirements.

### KEYWORDS

Stream Processing, Processing Model, Semantics

### ACM Reference Format:

Matthias J. Sax, Guozhang Wang, Matthias Weidlich, and Johann-Christoph Freytag. 2018. Streams and Tables: Two Sides of the Same Coin. In *International Workshop on Real-Time Business Intelligence and Analytics (BIRTE '18)*, August 27, 2018, Rio de Janeiro, Brazil. ACM, New York, NY, USA, 10 pages. <https://doi.org/10.1145/3242155>

## 1 INTRODUCTION

Stream processing has emerged as a paradigm to develop real-time applications. It builds on an evaluation of operators over unbounded sequences of data, enabling low-latency processing of large-scale data in a continuous manner [1]. As such, the stream processing paradigm turned out to be particularly suited to support the implementation of business logic in large organizations. It provides the backbone for communication between independent components of a large system, a.k.a. "microservices", through asynchronous message-passing [19].

\*Defining semantics for stream processing however

## The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing

Tyler Akidau, Robert Bradshaw, Craig Chambers, Slava Chernyak, Rafael J. Fernández-Moñezuma, Reuven Lax, Sam McVeety, Daniel Mills, Frances Perry, Eric Schmidt, Sam Whittle  
Google

(takida, robertwb, chambers, chernyak, fernand, relax, sgmc, millsd, tjp, clouce, samuelw)@google.com

### ABSTRACT

Unbounded, unordered, global-scale datasets are increasingly common in day-to-day business (e.g. Web logs, mobile usage statistics, and sensor networks). At the same time, consumers of these datasets have evolved sophisticated requirements, such as event-time ordering and windowing by features of the data themselves, in addition to an insatiable hunger for faster answers. Meanwhile, practicality dictates that one can never fully optimize along all dimensions of correctness, latency, and cost for these types of input. As a result, data processing practitioners are left with the quandary of how to reconcile the tensions between these seemingly competing prepositions, often resulting in disparate implementations and systems.

We propose that a fundamental shift of approach is necessary to deal with these evolved requirements in modern data processing. We as a field must stop trying to groove unbounded datasets into finite pools of information that eventually become complete, and instead live and breathe under the assumption that we will never know if or when we have

### 1. INTRODUCTION

Modern data processing is a complex and exciting field. From the scale enabled by MapReduce [18] and its successors (e.g. Hadoop [4], Pig [18], Hive [25], Spark [33]), to the vast body of work on streaming within the SQL community (e.g. query systems [1, 4, 15], windowing [22], data streams [24], time domains [28], semantic models [30]), to the more recent forays in low-latency processing such as Spark Streaming [34], MLlib, and Storm [6], modern consumers of data wield remarkable amounts of power in slicing and dicing massive-scale disorder into organized structures with far greater value. Yet, existing models and systems still fall short at a number of common use cases.

Consider an initial example: a streaming video provider wants to monetize their content by displaying video ads and billing advertisers for the amount of advertising watched. The platform supports online and offline views for content and ads. The video provider wants to know how much to bill each advertiser each day, as well as aggregate statistics about the videos and ads. In addition, they want to efficiently run

Arvind Arasu · Shivnath Babu · Jennifer Widom

## The CQL continuous query language: semantic foundations and query execution

Received: 7 June 2004 / Accepted: 22 November 2004 / Published online: 22 July 2005  
© Springer-Verlag 2005

**Abstract** *CQL*, a *continuous query language*, is supported by the STREAM prototype data stream management system (DSMS) at Stanford. CQL is an expressive SQL-based declarative language for registering continuous queries against streams and stored relations. We begin by presenting an abstract semantics that relies only on “black-box” mappings among streams and relations. From these mappings we define a precise and general interpretation for continuous queries. CQL is an instantiation of our abstract semantics using SQL to map from relations to relations, window specifications derived from SQL-99 to map from streams to relations, and three new operators to map from relations to streams. Most of the CQL language is operational in the STREAM system. We present the structure of CQL’s query execution plans as well as details of the most important components: operators, interoperator queues, synopses, and sharing of components among multiple operators and

[2, 19, 20, 23, 28, 32]. However, these queries tend to be simple and primarily for illustration – a precise language semantics, particularly for more complex queries, often is left unclear. Furthermore, very little has been published to date covering execution details of general-purpose continuous queries. In this paper we present the CQL language and execution engine for general-purpose continuous queries over streams and stored relations. CQL (for *continuous query language*) is an instantiation of a precise abstract continuous semantics also presented in this paper, and CQL is implemented in the STREAM prototype data stream management system (DSMS) at Stanford.<sup>1</sup>

It may appear initially that defining a continuous query language over (relational) streams is not difficult: take a relational query language, replace references to relations with references to streams, register the query with the stream processor, and wait for answers to arrive. For simple monotonic

# Is this too ambitious?



[https://www.google.com/url?  
sa=i&source=images&cd=&ved=2ahUKE  
wjaouS-  
uK3mAhVEqaQKHfdOA4MQjRx6BAgBE  
AQ&url=https%3A%2F%2Fmedium.com  
%2Fpersonal-growth-lab%2Fshould-  
you-set-realistic-or-highly-ambitious-  
goals-7cf400505444&psig=AOvVaw3xPk  
Wzwvy8SBXD9NS0o\\_Oe&ust=15761481  
63494865](https://www.google.com/url?sa=i&source=images&cd=&ved=2ahUKEwjaouS-uK3mAhVEqaQKHfdOA4MQjRx6BAgBEAQ&url=https%3A%2F%2Fmedium.com%2Fpersonal-growth-lab%2Fshould-you-set-realistic-or-highly-ambitious-goals-7cf400505444&psig=AOvVaw3xPkWzwvy8SBXD9NS0o_Oe&ust=1576148163494865)

# A (Query) Language Perspective

Arvind Arasu · Shivnath Babu · Jennifer Widom

## The CQL continuous query language: semantic foundations and query execution

Received: 7 June 2004 / Accepted: 22 November 2004 / Published online: 22 July 2005  
© Springer-Verlag 2005

**Abstract** *CQL*, a *continuous query language*, is supported by the STREAM prototype data stream management system (DSMS) at Stanford. CQL is an expressive SQL-based declarative language for registering continuous queries against streams and stored relations. We begin by presenting an abstract semantics that relies only on “black-box” mappings among streams and relations. From these mappings we define a precise and general interpretation for continuous queries. CQL is an instantiation of our abstract semantics using SQL to map from relations to relations, window specifications derived from SQL-99 to map from streams to relations, and three new operators to map from relations to streams. Most of the CQL language is operational in the STREAM system. We present the structure of CQL’s query execution plans as well as details of the most important components: operators, interceptor queues, synopses, and sharding of components among multiple operators and management system (DSMS) at Stanford.<sup>1</sup>

It addresses the problem of manipulating and managing data-streams. It stresses on the **operations** that are **necessary** to build **applications**. Some **Assumptions** are made on the underlying **system(s)**.

# A new hybrid hope perspective

It addresses the problems in between. How do we map the abstraction of an high-level language to the underlying system?

# The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing

Tyler Axidau, Robert Bradshaw, Craig Chambers, Slava Chernyak, Rafael J. Fernández-Macías, Reuven Lax, Sam McVea, Daniel Mills, Frances Perry, Eric Schmidt, Sam Whittle  
Google

{takidau, robertwb, chambers, chernyak, rfernand, relax, sgmc, millsd, tjp, clouce, samuelwi@google.com

## ABSTRACT

**ABSTRACT.** Unbiased, unordered, global-scale datasets are increasingly common in day-to-day business (e.g. Web logs, mobile usage statistics, and sensor networks). At the same time, consumers of these datasets have evolved sophisticated requirements, such as event-time ordering and windowing by features of the data themselves, in addition to an inevitable hunger for faster answers. Meanwhile, practitioners dictate that one can never fully optimize along all dimensions of correctness, latency, and cost for these types of input. As a result, data processing practitioners are left with the quandary of how to reconcile the tensions between these seemingly competing propositions, often resulting in disparate implementations and systems.

We propose that a fundamental shift of approach is necessary to deal with the evolving requirements in modern data processing. We see as a field need to “try” to get useful datasets into finite pools of information that gradually become complete, one instead live and breathe under the representation that will serve known life or when we have to monetize their content by displaying video ads and billing advertisers for the amount of advertising watched. The platform supports online and offline views for content and ads. The video provider wants to know how much is bill each advertiser individually, as well as aggregate statistics about the videos and ads. In addition, they want to efficiently run

# A System Perspective

to address the problem of dealing with **unbounded data**. Discuss the systems' **primitives** that are necessary to guarantee **low-latency** and **fault-tolerance in** presence of \*uncertainty\*, e.g., late arrivals.

## Streams and Tables: Two Sides of the Same Coin

Matthias J. Sax<sup>\*</sup>  
Confluent Inc.  
Palo Alto, USA  
*matthias@confluent.io*

Guozhang Wang  
Confluent Inc.  
Palo Alto, USA  
[guozhang@confluent.io](mailto:guozhang@confluent.io)

**Matthias Weidlich**  
Humboldt-Universität zu Berlin  
Berlin, Germany  
[matthias.weidlich@hu-berlin.de](mailto:matthias.weidlich@hu-berlin.de)

Johann-Christoph Freytag  
Humboldt-Universität zu Berlin  
Berlin, Germany  
[freytag@informatik.hu-berlin.de](mailto:freytag@informatik.hu-berlin.de)

## KEYWORDS

Stream Processing, Processing Model, Semantics  
**ACM Reference Format:**  
Matthias J. Sax, Guanhang Wang, Matthias Weidlich, and Johan Christoph Freytag. 2018. Streams and Tables: Two Sides of the Same Coin. In *International Workshop on Real-Time Business Intelligence and Analytics (BERTE '18)*, August 27, 2018, Rio de Janeiro, Brazil. ACM, New York, NY, USA, 10 pages. <https://doi.org/10.1145/3242153>

TRACT

a processing has emerged as a paradigm for applications that require low-latency evaluation of operators on bounded sequences of data. Defining the semantics of stream processing is challenging in the presence of distributed data sources. The physical and logical order of data items may become inconsistent in such a setting. Existing models either neglect these inconsistencies or handle them by means of data buffering and reordering techniques, thus compromising processing latency.

In this paper, we introduce the Dual Streaming Model about physical and logical order in data streams. This model presents the result of an operator as a sum of successive updates, which induces a duality of streams. As such, it provides a natural way to cope with inconsistencies between the physical and logical order of data items in a continuous manner, without explicit buffering and reordering. We further discuss the trade-offs and challenges faced when implementing this model in terms of correctness, latency, and processing cost. A case study on Apache Kafka illustrates the effectiveness of our model in the light of real-world requirements.

1 INTRODUCTION

Stream processing has emerged as a paradigm to develop real-time applications. It builds on an evaluation of operators over unbounded sequences of data, enabling low-latency processing of large-scale data in a continuous manner [1]. As such, the stream processing paradigm turned out to be particularly suited to support the implementation of business logic in large organizations. It provides the backbone for communication between independent components of large systems, a.k.a. "microservices", through asynchronous message-passing [19].

Arvind Arasu · Shivnath Babu · Jennifer Widom

## The CQL continuous query language and query execution

CQL

- Exploit lesson learnt in RDBMS theory.

Received: 7 June 2004 / Accepted: 22 November 2004 / Published online: 10 January 2006  
© Springer 2006

- simple, clear, yet powerful language.
- efficiently combine streams and relations.

- low-latency is paramount.
- take data distribution and ordering into account.
- avoid implicit operator semantics.

# Goals

Matthias J. Sax\*

Confluent Inc.  
Palo Alto, USA

matthias@confluent.io

Matthias Weidlich  
DSM

Humboldt-Universität zu Berlin  
Berlin, Germany  
matthias.weidlich@hu-berlin.de

Tyler Akidau, Robert Bradshaw  
Rafael J. Fernández-Macézuma  
Frances Perry, Lin Guo

DFM

akidau, robertwb, e

relax, sgmc, millsdf, fjp

# The Dataflow Model: A Perspective on Correctness, Latency, and Unbounded, Out-of-order Data

## ABSTRACT

- Correctness first with sessions support.

- Latency requirements vs resource cost should dictate system choice.

We propose that a fundamental shift of approach is necessary to deal with these evolved requirements in modeling data processing. We find that trying to groom bounded datasets into finite pools of information that eventually become stale and breathe unproductively under the assumption that we will never know if or when we have seen all the data is not sustainable. Late, old data may be retracted, and the only way to make this problem tractable is via principled abstractions that allow the practitioner the choice of appropriate tradeoffs along the axes of interest: correctness, latency, and cost.

- Single processing model for bounded and unbounded data.

# Streams and Tables: Time

Arvind Arasu · Shivnath Babu · Jennifer Widom

The CQL continuous query language and query execution

CQL

Matthias J. Sax\*

Confluent Inc.  
Palo Alto, USA

matthias@confluent.io

DSM

Matthias Weidlich  
Humboldt-Universität zu Berlin  
Berlin, Germany

matthias.weidlich@hu-berlin.de

Tyler Akidau, Robert Bradshaw

Rafael J. Fernández-Macías

Frances Perry, I

DFM

{takidau, robertwb, c  
relax, sgmc, mills, fjp

- DSMS Clock Time, i.e., the timestamp assigned by the DSMS.
- Source Time, i.e., the timestamp assigned by the source.
- Source Heartbeat are used to declare no element with lower timestamp will be sent.
- Logical Order induced by the timestamp assigned by the data source.
- Physical Order induced by the timestamp assigned by the system at ingestion.

The Dataflow Model: A P  
Correctness, Latency,  
Unbounded, Out-of-

**ABSTRACT**  
Unbounded, unordered, global-scale datasets are incr  
ingly common in data mining, log processing, usage statistics, and sensor networks). At the same time, consumers have increasingly stringent requirements, such as event-time ordering and windowing features of the data themselves, in addition to an insatiable hunger for faster answers. Meanwhile, practicality dictates that one can no longer afford to ignore the trade-offs between these seemingly competing propositions, often resulting in disparate im

**• Processing Time** is the time at which an event is observed during processing.

We propose that a fundamental shift of approach is necessary to deal with these evolved requirements in modern data processing. We as a field must stop trying to groom both data and time, the two basic dimensions of time that eventually become complete, and instead live and breathe unstructured time. When we have seen all of our data, only that new data will arrive, old data may be discarded. To make this problem tractable is via principled abstractions that allow the partitioning of the choices of appropriate tradeoffs along the axes of interest: correctness, latency, and cost.

**• Watermark**, i.e., global progress metrics that tracks the skewness between ET and PT.

# Watermark

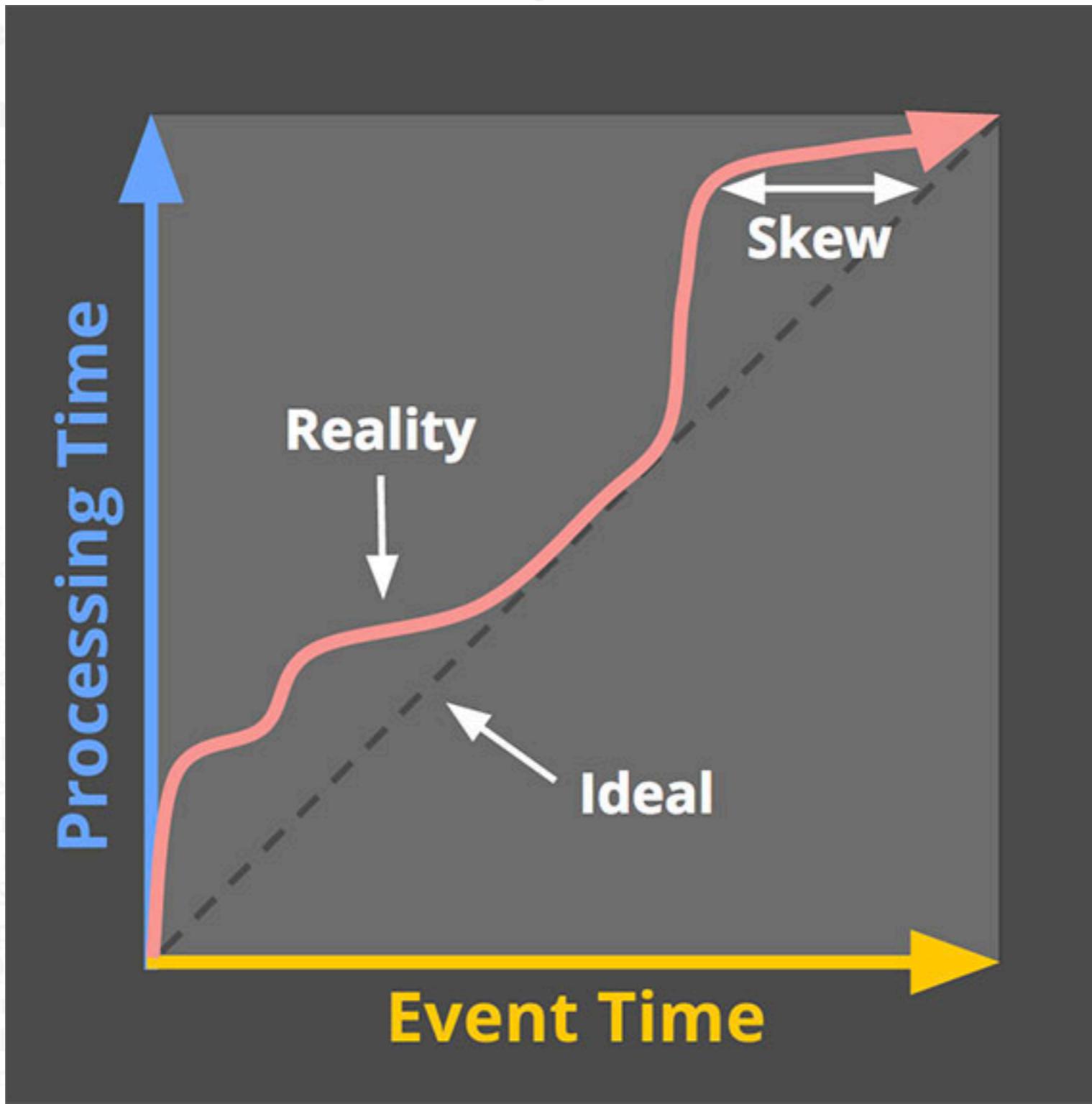
Matthias J. Sax\*

Arvind Arasu · Shivnath B

The CQL continuous query language and query execu

Received: 7 June 2004 / Accepted:  
© Springer-Verlag 2005

**Abstract** *CQL*, a *continuous query language*, is introduced by the STREAM prototype system (DSMS) at Stanford. CQL is a declarative language for reasoning against streams and stored relations. It provides an abstract semantics that relies on the notion of watermarks among streams and relations. We define a precise and general semantics for continuous queries. CQL is an instantiation of the semantics using SQL to map from continuous specifications derived from SQL to relations, and three new operators to map from relations to streams. Most of the CQL features have been implemented in the STREAM system. We present the basic concepts of query execution plans as well as the most important components: operators, intermediate relations, and sharing of components among multiple executions. We discuss correctness, latency, and processing cost. In case of interest, correctness is based on Apache Kafka illustrates the effectiveness of the model in the light of real-world requirements.



tyler Akidau, Robert Bradshaw, Daniel J. Fernández-Moctezuma, Frances Perry, I

{takidau, robertwb, dfernandez, frances, relax, sgmc, mills, fjp}

red, global-scale datasets are increasingly common in today's business (e.g. Web logs, mobile devices, sensor networks). At the same time, these datasets have evolved sophisticated requirements for event-time ordering and windowing, often forcing themselves, in addition to an insatiable desire for answers. Meanwhile, practicality dictats that we must optimize along all dimensions of performance and cost for these types of input. As a result, practitioners are left with the quandary of balancing the tensions between these seemingly opposing goals, often resulting in disparate implementation choices.

A fundamental shift of approach is required to meet these evolved requirements in modern systems. As a field must stop trying to groom data to finite pools of information that are static, discrete, and instead live and breathe under us, we will never know if or when we have processed all data. The only thing we can know is that new data will arrive, old data will leave, and the only way to make this problem tractable is to find principled abstractions that allow the practitioner to make appropriate tradeoffs along the axes of correctness, latency, and cost.

# Abstractions

Matthias J. Sax\*

Confluent Inc.

Palo Alto, USA

matthias@confluent.io

CQL

Matthias Weidlich  
DSM

Humboldt-Universität zu Berlin

Berlin, Germany

matthias.weidlich@hu-berlin.de

The Dataflow Model: A P  
Correctness, Latency,  
Unbounded, Out-

Arvind Arasu · Shivnath Babu · Jennifer Widom

Tyler Akidau, Robert Bradsh  
Rafael J. Fernández-Macías

Frances Perry, I

The CQL continuous query la  
and query execution

DFM

akidau, robertwb, c

relax, sgmc, millsdf, fjp

Received: 7 June 2004 / Accepted: 22 November 2004 / Published online: 22 November 2004  
© Springer-Verlag 2005

## • Streams

## • (Time-Varying) Relations\*

## • Streams

## • Change-log Streams

## • Records Streams

## • Tables

## ABSTRACT

Unbounded, unordered, global-scale datasets are increasingly common in day-to-day business (e.g. Web logs, mobile usage statistics, and sensor networks). At the same time, consumers of these datasets have evolved sophisticated requirements, such as event-time ordering and windowing features of the data themselves, in addition to an insatiable hunger for faster answers. Meanwhile, practicality dictates that one must trade off correctness, latency, and cost along all dimensions of these types of input. As a result, data processing systems are left with the quandary of how to reconcile the tensions between these seemingly competing requirements, resulting in disparate implementations and systems.

## • PCollections (bounded and unbounded)

We propose that a fundamental shift of approach is necessary to deal with these evolved requirements in modern data processing. We as a field must stop trying to groom bounded datasets into finite pools of information that eventually become complete, and instead live and breathe under the assumption that we will never know if or when we have seen all of our data, only that new data will arrive, old data may be retracted, and the only way to make this problem tractable is via principled abstractions that allow the practitioner the choice of appropriate tradeoffs along the axes of interest: correctness, latency, and cost.

ABSTRACT

Unbounded, unsorted, global-scale datasets are increasingly common in day-to-day business (e.g. Web logs, mobile usage statistics, and sensor networks). At the same time, consumers of these datasets have increasingly stringent requirements, such as event-time ordering and windowing by fixed or sliding time intervals. This has created a growing hunger for faster answers. Meanwhile, practicality dictates that one can never fully optimize along all dimensions of correctness, latency, and cost. As a result, the result of a data processing practitioner is left with the quandary of how to best trade off these three metrics. In this paper, we propose that a fundamental shift of approach is necessary to deal with these evolved requirements in modern data processing systems. We introduce the Dataflow Model, which bounds datasets into finite pools of information that eventually become complete, and instead are used to guide code generation that will never know if or when we have

# CQL in 3 Slides

A Stream **S** is a possibly infinite **multi-set** of elements  $\langle s, t \rangle$  where  $s$  is a tuple belonging to the schema of  $S$  and  $t$  is a timestamp.

Relation **R** is a set of tuples  $(d_1, d_2, \dots, d_n)$ , where each element  $d_i$  is a member of  $D_i$ , a **data domain**<sup>1</sup>.

<sup>1</sup> a Data Domain refers to all the values which a data element may contain.

# CQL in ✘ 4 Slides

Received: 7 June 2004 / Accepted: 22 November 2004 / Published online: 22 July 2005  
© Springer-Verlag 2005

**Abstract** CQL, a continuous query language, is supported by the STREAM prototype data stream management system (DSMS). It extends SQL with an expressive SQL-like declarative language for registering continuous queries against streams and stored relations. We begin by presenting an abstract syntax that includes only a “black-box” mapping among streams and relations. From this abstract syntax we define a precise and general interpretation for continuous queries. CQL is an instantiation of an abstract semantics for SQL-like continuous relations. The CQL query language specifications derived from SQL 99 to map from streams to relations, and to map from relations to streams. Most of the CQL language is implemented in the STREAM prototype data stream management system. In this paper, we present the structure of CQL’s query execution plans as well as details of the most important components: operators, temporal operators, synopses, and sharing of components among multiple operators and

A Stream **S** is a possibly infinite **multi-set** of elements  $\langle s, t \rangle$  where  $s$  is a tuple belonging to the schema of  $S$  and  $t$  is a timestamp.

~~Relation **R** is a set of tuples  $(d_1, d_2, \dots, d_n)$ , where each element  $d_i$  is a member of  $D_i$ , a data domain<sup>1</sup>.~~

<sup>1</sup> a Data Domain refers to all the values which a data element may contain.

# Ok, CQL in ~~4~~ 5 Slides

Received 11 June 2004 / Accepted 22 November 2004 / Published online: 22 July 2005  
© 2005 Springer Science+Business Media, Inc.  
Abstract CQL, the continuous query language, is developed for the STREAM system. CQL is a query language based on SQL, particularly designed for stream management systems. It is a declarative language for defining queries against streams and stored relations. We begin by defining the basic concepts of streams and stored relations. We then present the CQL language and its semantics. Finally, we illustrate how CQL can be used to express continuous queries over streams and stored relations. CQL (for continuous query language) is an instantiation of a more abstract query language over (relational) streams. In this paper, we present the CQL language and its semantics. We also present the implementation of CQL in the STREAM prototype data stream processor.

A Stream **S** is a possibly infinite **multi-set** of elements  $\langle s, t \rangle$  where  $s$  is a tuple belonging to the schema of  $S$  and  $t$  is a timestamp.

Relation **R** is a mapping from each time instant in  $T$  to a finite but unbounded bag of tuples belonging to the schema of  $R$ .

<sup>1</sup> a Data Domain refers to all the values which a data element may contain.

Received: 7 June 2004 / Accepted: 22 November 2004 / Published online: 22 July 2005  
© Springer-Verlag 2005

**Abstract** CQL, a continuous query language, is supported by the STREAM prototype data stream management system (DSMS). It extends SQL with an expressive SQL-like declarative language for registering continuous queries against streams and stored relations. We begin by presenting an abstract model of tuples only via “black-box” mappings among streams and relations. From these primitives we define a precise and general interpretation for continuous queries. CQL is an instantiation of our abstract semantics using SQL-like from relations and SQL-like query specifications derived from SQL 99 to map from streams to relations, and from relations to relations. We present the CQL language in the STREAM prototype data stream management system. Most of the CQL language is operational. CQL is implemented in the STREAM prototype data stream management system.

It may appear initially that defining a continuous query language over (relational) streams is not difficult: take a relation, add timestamp columns, and then add references to streams, register the query with the stream processor, and wait for answers to arrive. For simple monodic

# CQL in 5 Slides

A Stream **S** is a possibly infinite **multi-set** of elements  $\langle s, t \rangle$  where  $s$  is a tuple belonging to the schema of  $S$  and  $t$  is a timestamp.

Relation **R** is a mapping from each time instant in  $T$  to a finite but unbounded bag of tuples belonging to the schema of  $R$ .

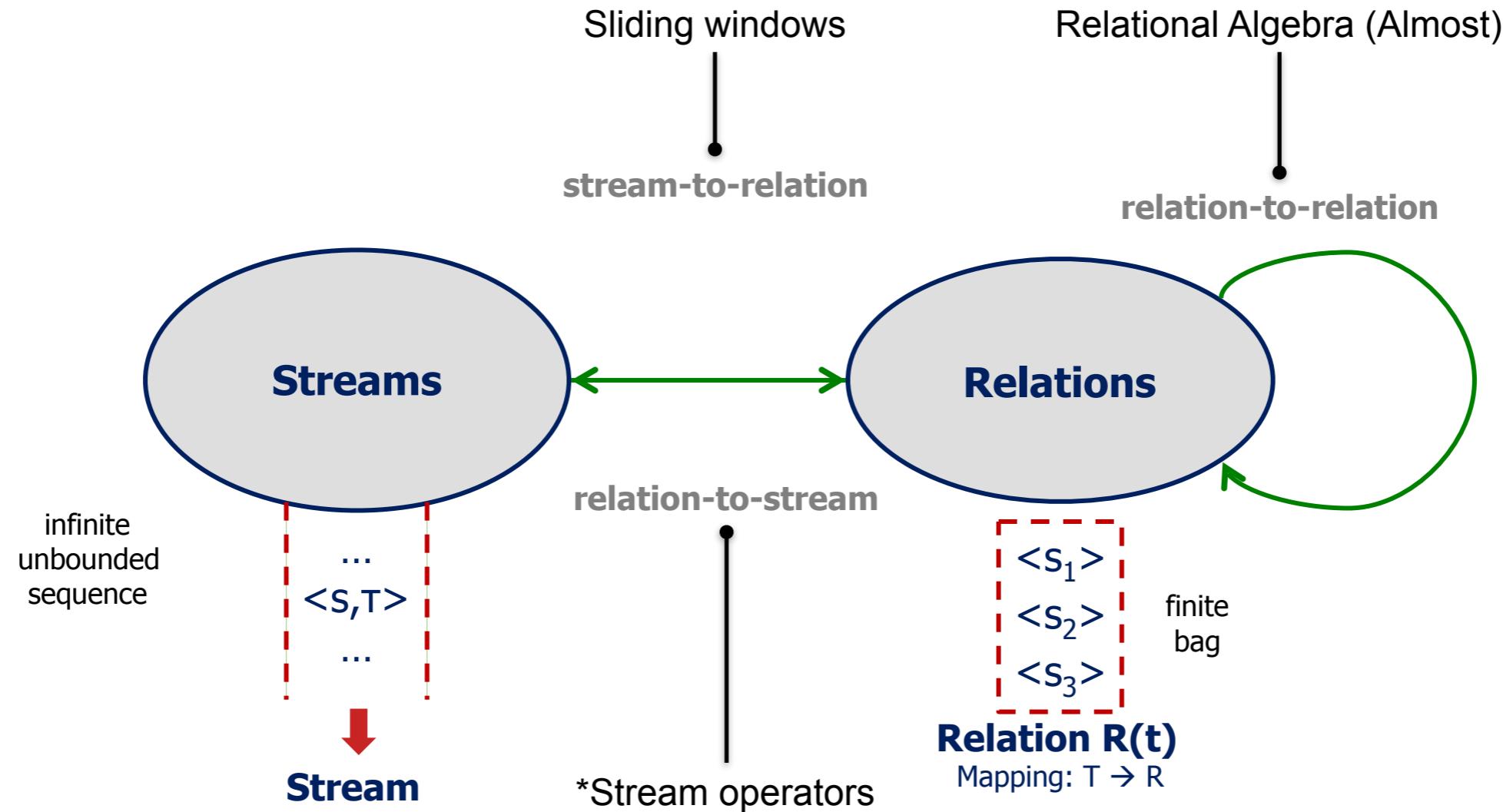
<sup>1</sup> a Data Domain refers to all the values which a data element may contain.

# CQL in 5 Slides

Arvind Arasu · Shivnath Babu · Jennifer Widom  
The CQL continuous query language: semantic foundations  
and query execution

Received: 7 June 2004 / Accepted: 22 November 2004 / Published online: 22 July 2005  
© Springer-Verlag 2005

**Abstract CQL**, a continuous query language, is supported by the STREAM prototype data management system (DSMS) at Stanford. CQL is an expressive SQL-based language for specifying continuous queries over streams and relations. It is built upon an abstract semantics that relies only on “black-box” mapping functions between stream and relation domains. In this paper, we define a precise and general interpretation for continuations in CQL. As an instance of our abstract semantics, we show how CQL’s continuous relations can be mapped to SQL queries derived from SQL 99 to map from streams to relations. We also show how CQL’s continuous relations map to streams. Most of the CQL language is operational in the STREAM system. We present the structure of CQL’s query language, its semantics, and its operational interpretation. The language supports operators, interpretation queries, and sharing of components among multiple query engines.



Received: 7 June 2004 / Accepted: 22 November 2004 / Published online: 22 July 2005  
© Springer-Verlag 2005

**Abstract** CQL, a continuous query language, is supported by the STREAM prototype data stream management system (DSMS). It extends SQL with an expressive SQL-like declarative language for registering continuous queries against streams and stored relations. We begin by presenting an abstract syntax that includes only a “black-box” mapping among streams and relations. From this abstract syntax we define a precise and general interpretation for continuous queries. CQL is an instantiation of our abstract semantics for SQL. It maps relations and streams to SQL-like specifications derived from SQL 99 to map from streams to relations, and to map relations to streams. This mapping is used to implement CQL. Most of the CQL language is implemented in the STREAM system. We present the structure of CQL’s query execution plans as well as details of the most important components: operators, function symbols, type synopses, and sharing of components among multiple operators and

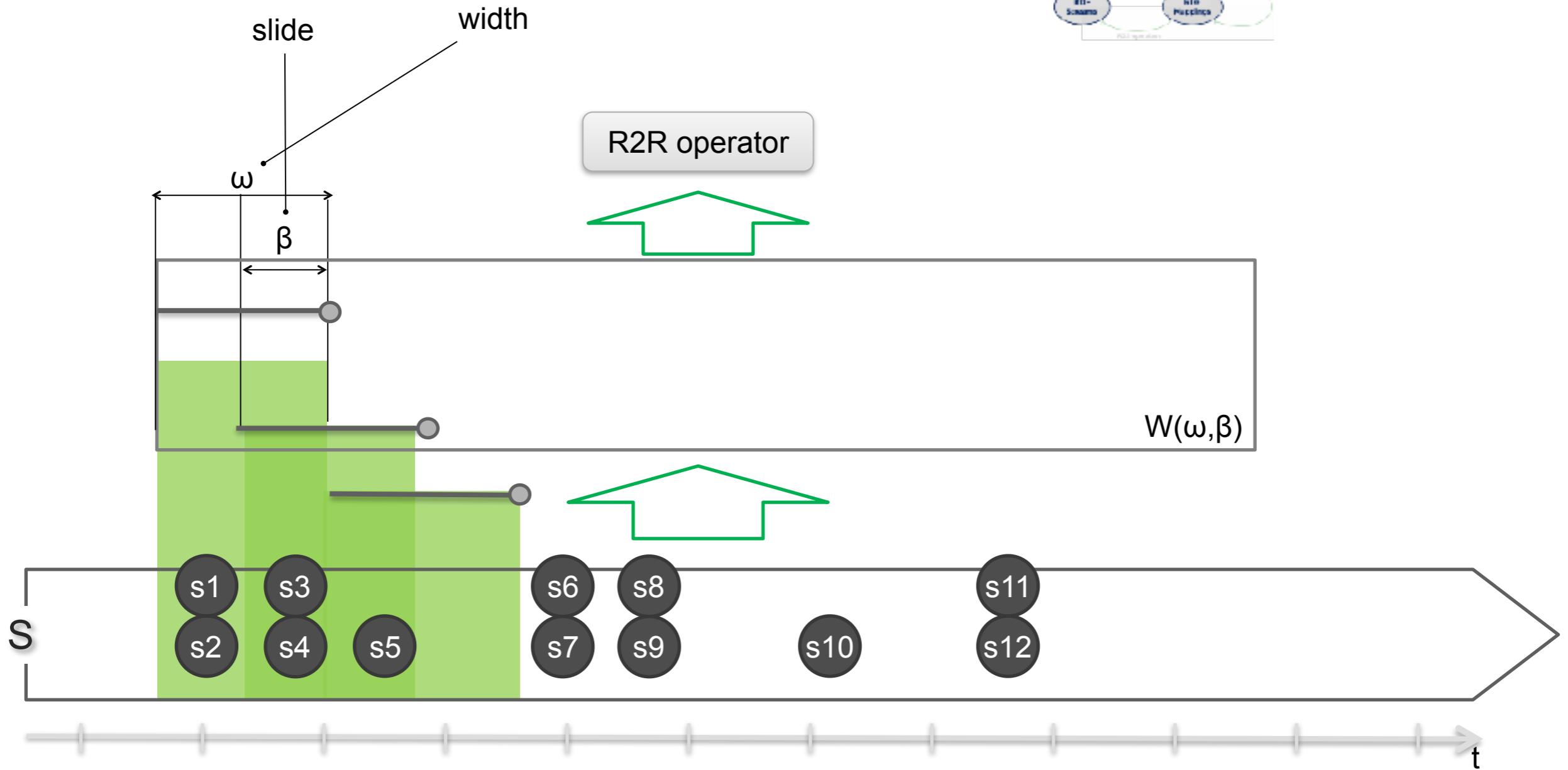
# CQL in ~~X~~ 6 Slides

## Stream-to-Relation Operators:

- **Sliding Window:**  
FROM S [ RANGE 5 Minutes]
- **Parametric Sliding Windows:**  
FROM S [ RANGE 5 Minutes Slide 1 Min]
- **Partitioned Windows:**  
FROM S [PARTITIONED BY A<sub>1..n</sub> ROW m]

<sup>1</sup> a Data Domain refers to all the values which a data element may contain.

# CQL in 6 Slides



Received: 7 June 2004 / Accepted: 22 November 2004 / Published online: 22 July 2005  
© Springer-Verlag 2005

**Abstract** CQL, a continuous query language, is supported by the STREAM prototype data stream management system (DSMS). It extends SQL with an expressive SQL-like declarative language for registering continuous queries against streams and stored relations. We begin by presenting an abstract query calculus only “black-box” mapping among streams and relations. From there, we proceed to define a precise and general interpretation for continuous queries. CQL is an instantiation of our abstract semantics using SQL-like from relations and SQL-like query specifications derived from SQL 99 to map from streams to relations, and vice versa. We present the presentation layer of the CQL language, which is implemented in the STREAM system. We present the structure of CQL’s query execution plans as well as details of the most common non-compositional operators, including joins, synapses, and sharding of components among multiple operators and

[2, 19, 20, 23, 28, 32]. However, these queries tend to be simple and precise for illustration – a precise language is needed, particularly for more complex queries, often is left unclear. Furthermore, very little has been published to date covering execution details of general-purpose continuous query systems. In this paper, we present the CQL language and its semantics, and we present the presentation layer of CQL, which is implemented in the STREAM prototype data stream management system. We also present the execution plan of CQL, which is implemented in the STREAM prototype data stream management system.

In my opinion initially that defining a continuous query language over (relational) streams is not difficult: take a relation and a timestamp, and then add a timestamp to every row with references to streams, register the query with the stream processor, and wait for answers to arrive. For simple monomic

Received: 7 June 2004 / Accepted: 22 November 2004 / Published online: 22 July 2005  
© Springer-Verlag 2005

# CQL in 6 Slides

Abstract CQL, a continuous query language, is supported by the STREAM prototype data stream management system (DSMS) at Stanford. CQL is an expressive SQL-like declarative language for registering continuous queries against streams and stored relations. We begin by presenting an abstract syntax of CQL, and a “black-box” mapping among streams and relations. From these primitives we define a precise and general interpretation for continuous queries. CQL is an instantiation of an abstract semantics called SQL\*. It maps relations and query specifications derived from SQL 99 to map from streams to relations, and from relations to stored relations. Most of the CQL language is operational in the STREAM system. We present the structure of CQL’s query execution plans as well as details of the most common operators: operators for reading streams, windowing, and sharing of components among multiple operators and

## Relation-to-Stream Operators:

- **Rstream:** streams out all data in the last step
- **Istream:** streams out data in the last step that wasn’t on the previous step, i.e. streams out what is new
- **Dstream:** streams out data in the previous step that isn’t in the last step, i.e. streams out what is old

<sup>1</sup> a Data Domain refers to all the values which a data element may contain.

# DFM in X<42\* Slides

## Reasoning about time

- **What** results are being computed
- **Where** in event time are being computed
- **When** in processing time are being materialised
- **How** “earlier results” relate to “later refinements”

\*overestimation

# DFM in X Slides

## (WHAT) Processing Model

A Stream is represented as a possibly unbounded collection of key-value pairs, i.e., a PCollection<K,V>.

PTransforms are PCollections-to-PCollections operations.

# DFM in X Slides

## (WHAT) Processing Model

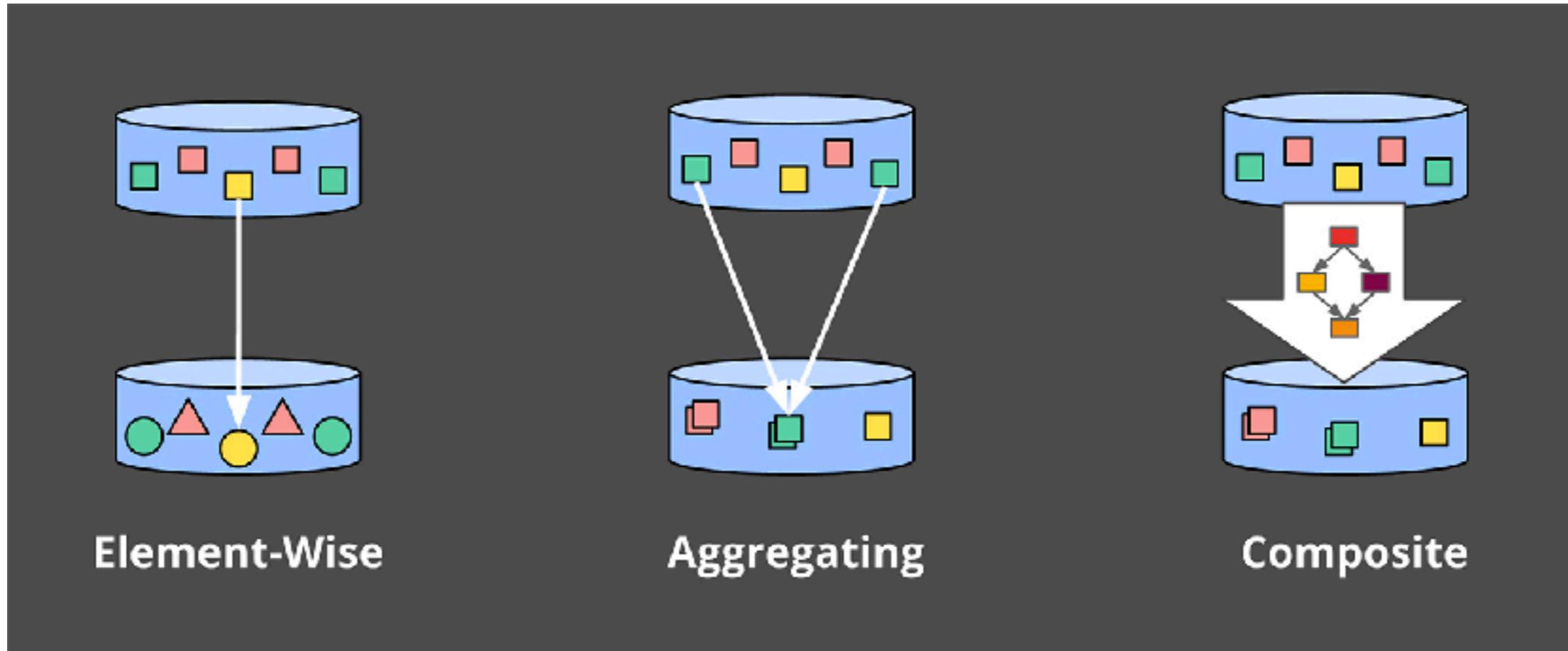
Arvind Arasu · Shivnath Babu · Jennifer Widom  
The CQL continuous query language: semantic foundations  
and query execution

Received: 7 June 2004 / Accepted: 22 November 2004 / Published online: 22 July 2005  
© Springer-Verlag 2005

**Abstract** CQL, a *continuous query language*, is supported by the STREAM prototype data stream management system (DSMS) at Stanford. CQL is an expressive SQL-based domain-specific language for expressing continuous queries against streams and stored relations. We begin by presenting an abstract semantics that relies only on the “block-and-maybe-split” paradigm of continuous queries. Then we present a precise and general interpretation for continuous queries. In an instantiation of our abstract semantics using SQL, we show how relations and windows specifications derived from SQL 99 to map from streams to relations. Finally, we show how to map these relations to streams. Most of the CQL language operations are implemented in the STREAM system. We present the structure of CQL’s query execution system, which includes support for query rewriting, operator scheduling, query optimization, and query execution. The CQL language supports operators for reading from streams, registering the query with the stream processor, and sharing of components among multiple operators and

[2, 19, 20, 23, 28, 32]. However, these queries tend to be simple and primarily for illustration – a precise language semantics, particularly for more complex queries, does not yet exist. Nevertheless, the work has been presented to date covering execution details of general-purpose continuous queries. In this paper we present the CQL language and semantics, particularly for continuous queries over streams and stored relations. CQL (for *continuous query language*) is an instantiation of a precise abstract semantics, also presented here. Finally, we show how CQL is implemented in the STREAM prototype data stream management system.

It may appear initially that defining a continuous query language over (relational) streams is not difficult: take a relation over a stream and project it. However, this is not so simple, as references to streams, register the query with the stream processor, and wait for answers to arrive. For simple monoton-



# DFM in X Slides

## (WHERE) Windowing Model

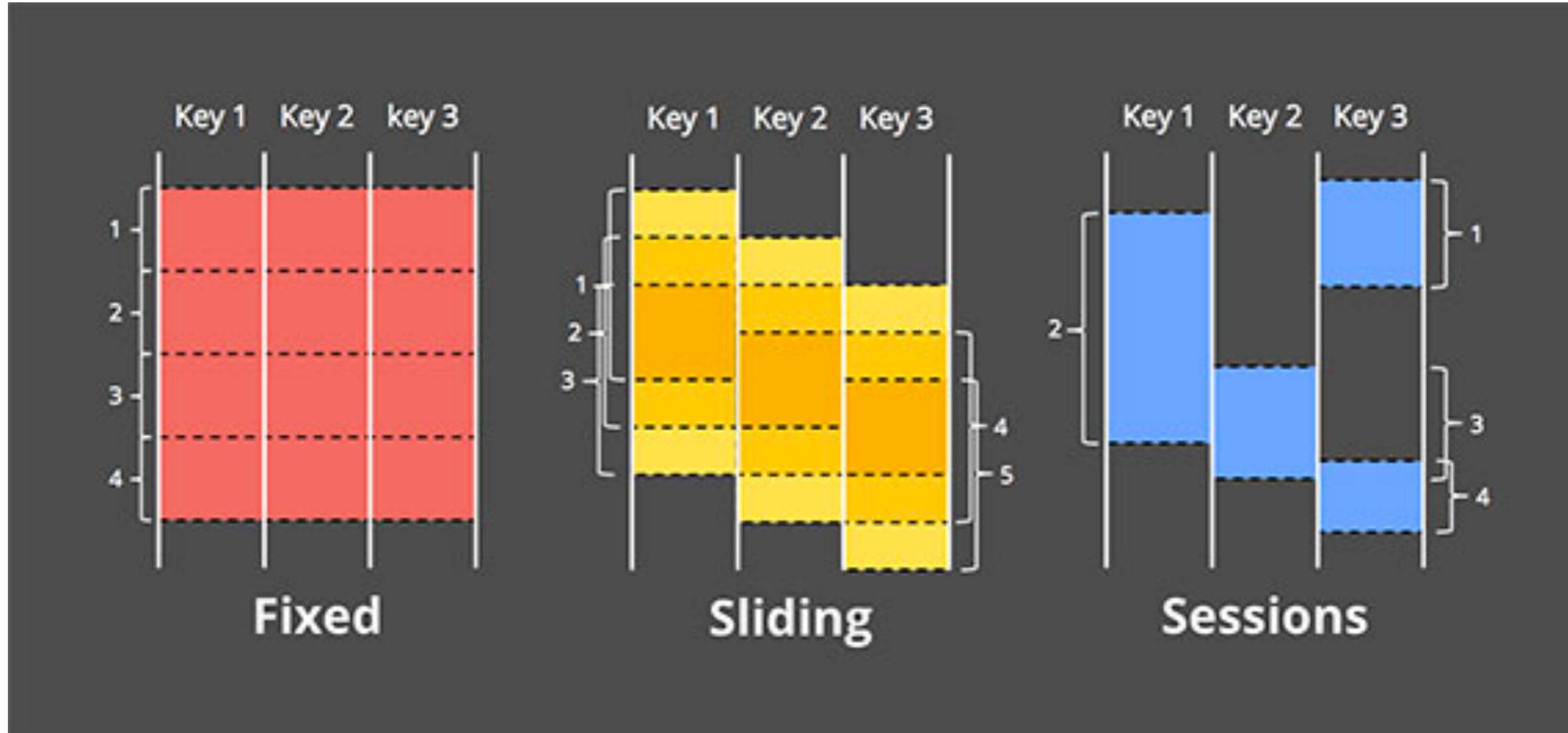
Received: 7 June 2004 / Accepted: 22 November 2004 / Published online: 22 July 2005  
© Springer-Verlag 2005  
Arvind Arasu · Shivnath Babu · Jennifer Widom  
The CQL continuous query language: semantic foundations and query execution

Received: 7 June 2004 / Accepted: 22 November 2004 / Published online: 22 July 2005  
© Springer-Verlag 2005

Abstract CQL, a continuous query language, is supported by the STREAM prototype data stream management system (DSMS) at Stanford. CQL is an expressive SQL-based domain-specific language for expressing continuous queries against streams and stored relations. We begin by presenting an abstract semantics that relies only on the “block-and-marry” paradigm for stream processing. Then we present concrete semantics, particularly for more complex queries, that is more amenable to implementation. Finally, we have provided to date covering execution details of general-purpose continuous queries. In this paper we present the CQL language and semantics, and also provide a brief overview of the STREAM system.

[2, 19, 20, 23, 28, 32]. However, these queries tend to be simple and primarily for illustration – a precise language semantics, particularly for more complex queries, does not yet exist. In this paper we present the CQL language and semantics, and also provide a brief overview of the STREAM system.

It may appear initially that defining a continuous query language over (relational) streams is not difficult: take a relation and map it to a stream. However, there are many issues with references to streams, register the query with the stream processor, and wait for answers to arrive. For simple monotonous



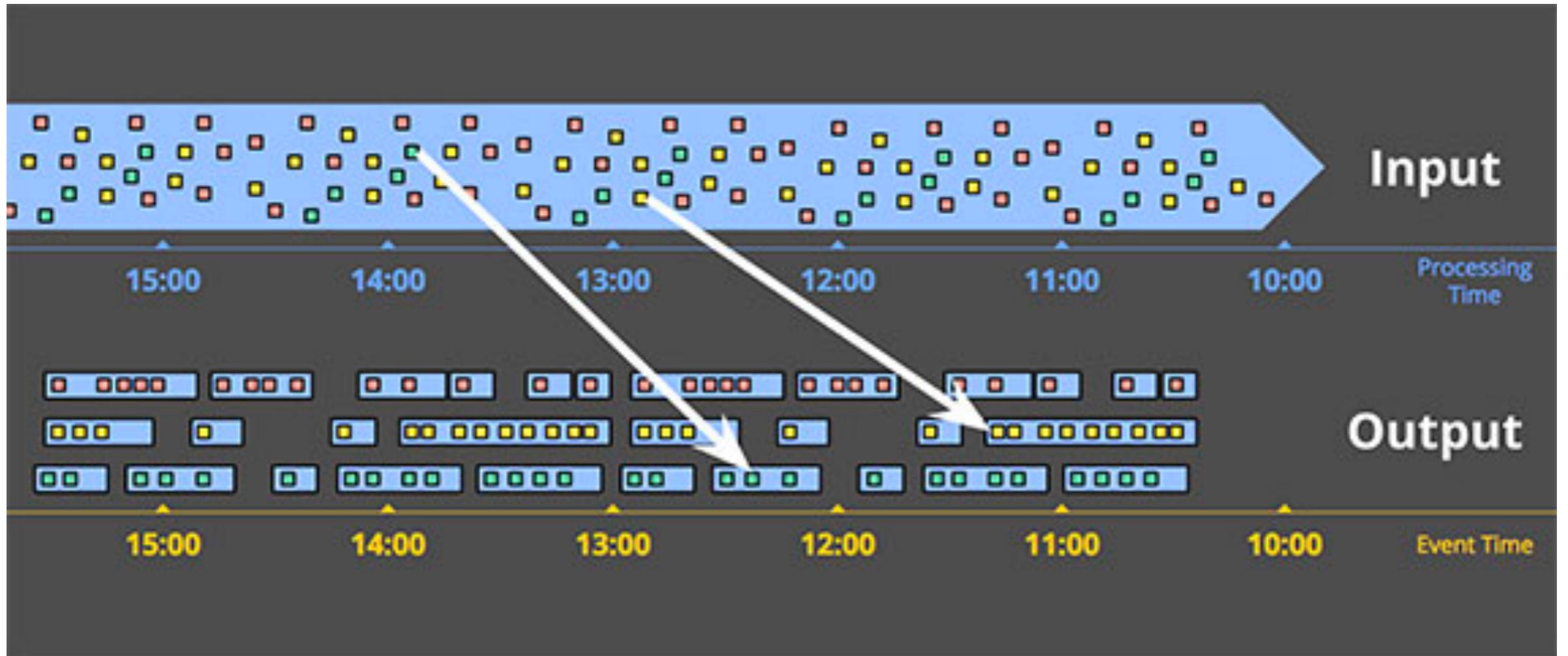
# DFM in X Slides

## (WHERE) Windowing Model

Received: 7 June 2004 / Accepted: 22 November 2004 / Published online: 22 July 2005  
© Springer-Verlag 2005  
Arvind Arasu · Shivnath Babu · Jennifer Widom  
The CQL continuous query language: semantic foundations and query execution

Received: 7 June 2004 / Accepted: 22 November 2004 / Published online: 22 July 2005  
© Springer-Verlag 2005

Abstract CQL, a continuous query language, is supported by the STREAM prototype data stream management system (DSMS) at Stanford. CQL is an expressive SQL-based domain-specific language for expressing continuous queries against streams and stored relations. We begin by presenting an abstract semantics that relies only on the “block-and-maybe-split” paradigm of continuous queries. Following this, we define a precise and general interpretation for continuous queries. In an instantiation of our abstract semantics, we use SQL’s notion of relations to map from window specifications derived from SQL 99 to map from streams to relations. This allows us to map any relation in streams. Most of the CQL language operations relate to streams. More details of the CQL language operations in the STREAM system. We present the structure of CQL’s query execution system, which includes components such as references to streams, register the query with the stream processor, and wait for answers to arrive. For simple monotonous [1, 19, 20, 23, 28, 32]. However, these queries tend to be simple and primarily for illustration – a precise language semantics, particularly for more complex queries, does not yet exist. Nevertheless, this paper has been prepared to date covering execution details of general-purpose continuous queries. In this paper we present the CQL language and semantics, particularly for WHERE clauses. Finally, we discuss how CQL’s semantics also provides for WHERE and GROUP BY clauses over streams and stored relations. CQL (for continuous query language) is an instantiation of a precise abstract query language with an interpretation of a precise abstract query language. It is also a query language with an interpretation of a precise abstract query language.





## **Disclaimer**

# DFM in X Slides

## (WHERE) Windowing Model

# The CQL continuous query language: semantic foundations and query execution

Received: 7 June 2004 / Accepted: 22 November 2004 / Published online: 22 July 2005  
© Springer-Verlag 2005

Abstract CQL, a continuous query language, is supported by the STREAM prototype data stream management system. Data streams are represented as an expression of the declarative language, containing continuous queries against streams and stored relations. We begin by presenting an abstract view of the language, followed by a discussion of mappings among streams and relations. From these mappings, we define a precise and general interpretation for continuous queries. Finally, we show how to map continuous queries written using SQL to map from relations, where windows specified from SQL to map from streams to relations. The CQL language is operational in STREAM. Most of the CQL language is operational in the STREAM system. The present structure of CQL's query language is similar to that of SQL, with support for query, set and composite operators, interrupter queues, synapses, and component operators among multiple operators and relations.

- The DFM windowing model requires extended primitives than CQL's one.
  - Window Assignment, i.e., each element is assigned to a corresponding window.
  - Window Merge:
    - **Drop Timestamp:** only the window interval is relevant here on
    - GroupBy Key: to enable parallel execution
    - Window Merge (the merge logic depends on the window strategy)
    - GroupByWindow: to ensure elements are processed in sequence window-wise.
  - ExpandToElements: assigned a valid timestamp to the elements.

# DFM in X Slides

## (WHEN) Triggering Model

Arvind Arasu · Shivnath Babu · Jennifer Widom  
The CQL continuous query language: semantic foundations  
and query execution

Received: 7 June 2004 / Accepted: 22 November 2004 / Published online: 22 July 2005  
© Springer-Verlag 2005

**Abstract** CQL, a *continuous query language*, is supported by the STREAM prototype data stream management system (DSMS) at Stanford. CQL is an expressive SQL-based domain-specific language for expressing continuous queries against streams and stored relations. We begin by presenting an abstract semantics that relies only on the “block-and-marry” paradigm of continuous query evaluation. Following this, we define a precise and general interpretation for continuous queries. CQL is an instance of our abstract semantics. We also show how CQL can map directly to window specifications derived from SQL 99 to map from streams to relations. Finally, we show how to map from relations to streams. Most of the CQL language operations are implemented in the STREAM system. We present the structure of CQL’s query execution system, which includes components such as query compilers, operators, interpreter queues, synopses, and sharing of components among multiple operators and

[2, 19, 20, 23, 28, 32]. However, these queries tend to be simple and primarily for illustration – a precise language semantics, particularly for more complex queries, does not yet exist. Nevertheless, the work has been performed to date covering execution details of general-purpose continuous queries. In this paper we present the CQL language and its semantics. We also present the CQL query execution system, which also provides a precise interpretation, and CQL is implemented in the STREAM prototype data stream management system.

It may appear initially that defining a continuous query language over (relational) streams is not difficult: take a relation and apply it to a stream. However, this is not the case. To correctly handle continuous queries, one must take into account references to streams, register the query with the stream processor, and wait for answers to arrive. For simple monoton-

- In order to build unaligned (apply across subset of the data) event-time windows DFM is forced to decouple the reporting of the window content.
- To do so, DFM introduces an orthogonal model that allows to signal when a window is ready to be processed.

# DFM in X Slides

## (WHEN) Triggering Model

Arvind Arasu · Shivnath Babu · Jennifer Widom  
The CQL continuous query language: semantic foundations  
and query execution

Received: 7 June 2004 / Accepted: 22 November 2004 / Published online: 22 July 2005  
© Springer-Verlag 2005

**Abstract** CQL, a *continuous query language*, is supported by the STREAM prototype data stream management system (DSMS) at Stanford. CQL is an expressive SQL-based domain-specific language for expressing continuous queries against streams and stored relations. We begin by presenting an abstract semantics that relies only on “block-and-marry” parse trees and a simple notion of parallelism. Then we show how to define a precise and general interpretation for continuous queries. CQL is an instance of our abstract semantics. We also show how to map from CQL to monolithic window specifications derived from SQL 99 to map from streams to relations. Finally, we show how to map from relations to streams. Most of the CQL language operations relate to streams. It may appear initially that defining a continuous query language over (relational) streams is not difficult: take a relation and map it to a stream. However, this is not the case. In particular, one needs to be able to query streams, update them with references to streams, register the query with the stream processor, and wait for answers to arrive. For simple monotonic

- In order to build unaligned (apply across subset of the data) event-time windows DFM is forced to decouple the reporting of the window content.
- To do so, DFM introduces an orthogonal model that allows to signal when a window is ready to be processed.
- Triggers solve this issue allowing DFM users to write an arbitrary logic to signal the completion of a window.



Disclaimer

# DFM in X Slides

## (HOW) Triggering Model (part 2)

Received: 7 June 2004 / Accepted: 22 November 2004 / Published online: 22 July 2005  
© Springer-Verlag 2005  
Arvind Arasu · Shivnath Babu · Jennifer Widom  
The CQL continuous query language: semantic foundations and query execution

Received: 7 June 2004 / Accepted: 22 November 2004 / Published online: 22 July 2005  
© Springer-Verlag 2005

Abstract CQL, a *continuous query language*, is supported by the STREAM prototype data stream management system (DSMS) at Stanford. CQL is an expressive SQL-based domain-specific language for expressing continuous queries against streams and stored relations. We begin by presenting an abstract semantics that relies only on the “block-and-marry” paradigm of continuous queries. From this semantics we define a precise and general interpretation for continuous queries. CQL is an instantiation of our abstract semantics that uses SQL-like stored relations and window specifications derived from SQL 99 to map from streams to relations. Most of the CQL language operators map directly to streams. Many of the CQL language operators in the STREAM system. We present the structure of CQL’s query execution system, which includes support for fault-tolerant components: operator queues, synopses, and sharing of components among multiple operators and

[2, 19, 20, 23, 28, 32]. However, these queries tend to be simple and primarily for illustration – a precise language semantics, particularly for more complex queries, does not yet exist.

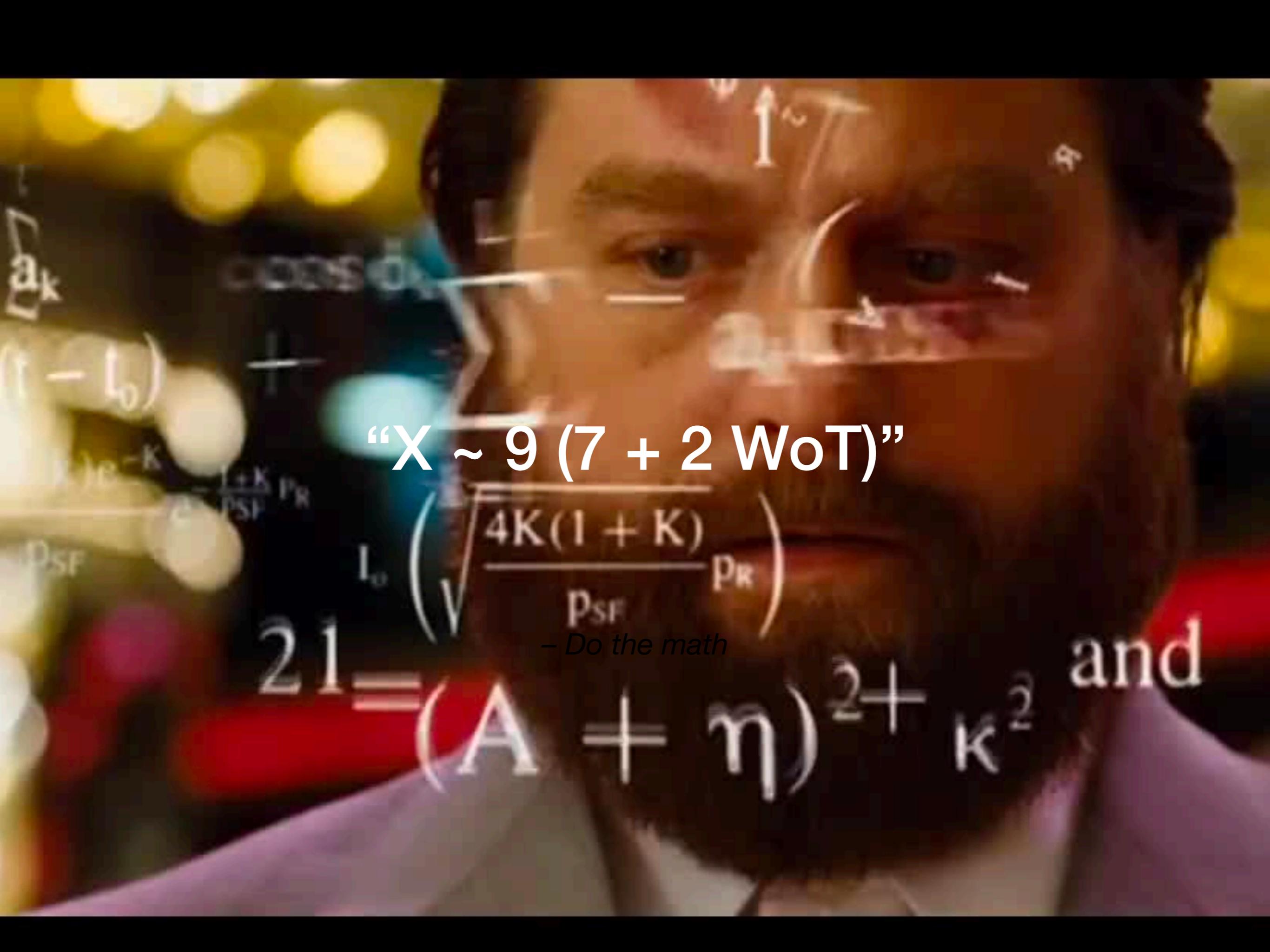
In this paper we present the CQL language and semantics, and also present a precise interpretation of CQL queries over streams and stored relations. CQL (for *continuous query language*) is an instantiation of a precise abstract query language with an interpretation of a precise abstract query language, and also provides fault tolerance, and

CQL is implemented in the STREAM prototype data stream management system.

It may appear initially that defining a continuous query language over (relational) streams is not difficult: take a relation and add a timestamp column. However, this is not the case. In addition, it is important to note that when defining a continuous query language, one must consider how to handle updates to streams, register the query with the stream processor, and wait for answers to arrive. For simple monotonous

In addition to the triggering semantics, DFM introduces different refinements models to deal with late arrivals

- Discarding, i.e., upon triggering, window contents are discarded and later results bear no relation to previous results.
- Accumulating, i.e., upon triggering, window contents are left intact in a persistent state and later results will become a refinement to previous results
- Accumulating & Retracting, i.e, it extends the accumulating semantics with retraction of the previous value.



"X ~ 9 (7 + 2 WoT)"

$$I_0 \left( \sqrt{\frac{4K(I+K)}{P_{SF}}} \right)$$

- Do the math

$$21 = (A + m)^2 + K^2 \text{ and}$$

# Was this too ambitious?



[https://www.google.com/url?  
sa=i&source=images&cd=&ved=2ahUKE  
wjaouS-  
uK3mAhVEqaQKHfdOA4MQjRx6BAgBE  
AQ&url=https%3A%2F%2Fmedium.com  
%2Fpersonal-growth-lab%2Fshould-  
you-set-realistic-or-highly-ambitious-  
goals-7cf400505444&psig=AOvVaw3xPk  
Wzvvy8SBXD9NS0o\\_Oe&ust=15761481  
63494865](https://www.google.com/url?sa=i&source=images&cd=&ved=2ahUKEwjaouS-uK3mAhVEqaQKHfdOA4MQjRx6BAgBEAQ&url=https%3A%2F%2Fmedium.com%2Fpersonal-growth-lab%2Fshould-you-set-realistic-or-highly-ambitious-goals-7cf400505444&psig=AOvVaw3xPkWzvvy8SBXD9NS0o_Oe&ust=1576148163494865)

zzzz

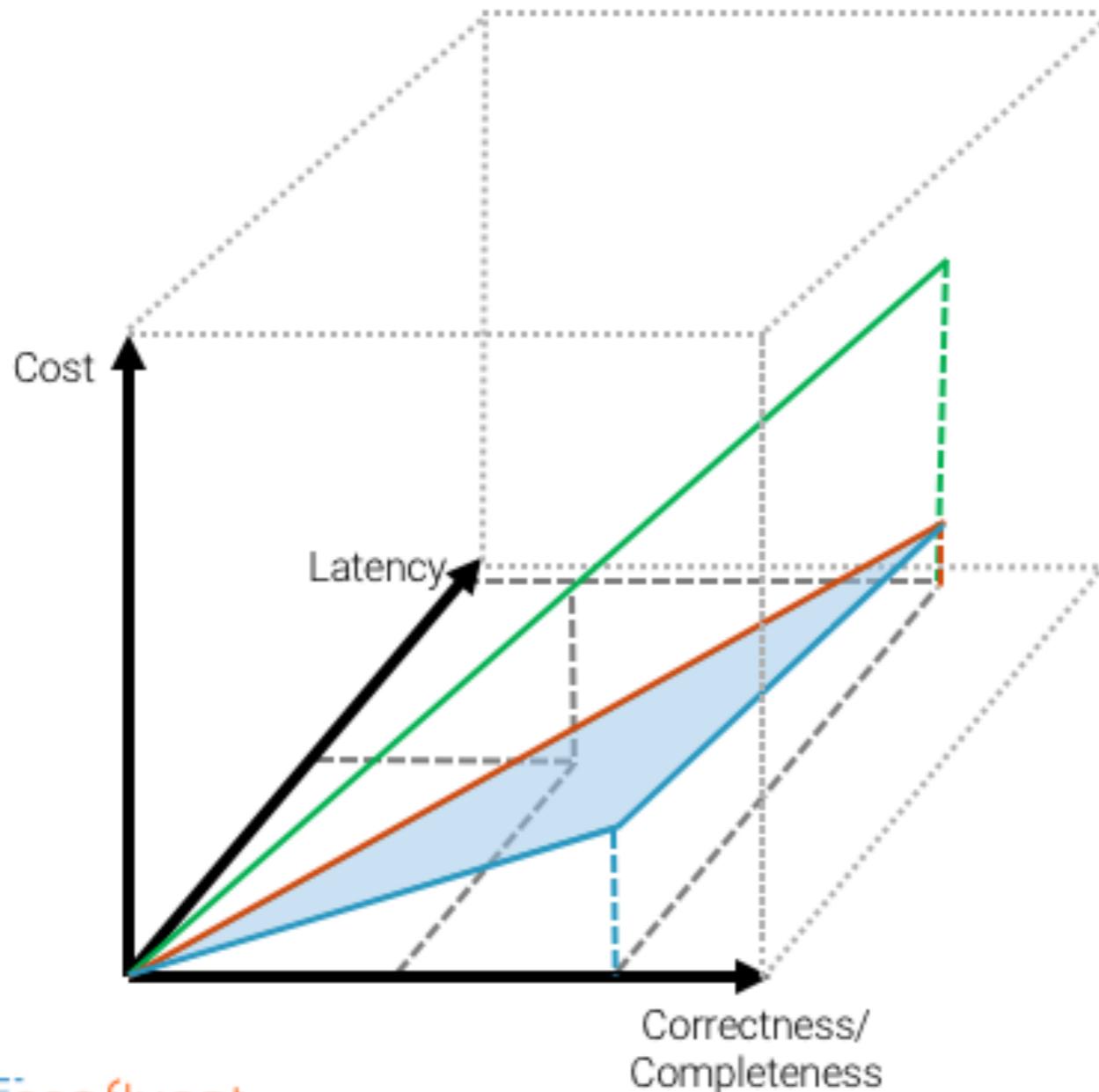




I'M NOT DONE

# Dual Stream Model

## The design space



confluent

### Streams and Tables: Two Sides of the Same Coin

Matthias J. Sax<sup>\*</sup>  
Confluent Inc.  
Palo Alto, USA  
matthias@confluent.io

Guozhang Wang<sup>\*</sup>  
Confluent Inc.  
Palo Alto, USA  
guozhang@confluent.io

Matthias Weidlich  
Humboldt-Universität zu Berlin  
Berlin, Germany  
matthias.weidlich@hu-berlin.de

Johann-Christoph Freytag  
Humboldt-Universität zu Berlin  
Berlin, Germany  
freytag@informatik.hu-berlin.de

**KEYWORDS**  
Stream Processing, Processing Model, Semantics

**ACM Reference Format:**  
Matthias J. Sax, Guozhang Wang, Matthias Weidlich, and Johann-Christoph Freytag, 2018. Streams and Tables: Two Sides of the Same Coin. In *International Workshop on Real-Time Business Intelligence and Analytics (BIRTE '18)*, August 27, 2018, Rio de Janeiro, Brazil. ACM, New York, NY, USA, 10 pages. <https://doi.org/10.1145/3242132/2155>

### 1 INTRODUCTION

Stream processing has emerged as a paradigm to develop real-time applications. It builds on an evaluation of operators over unbounded sequences of data, enabling low-latent processing of large-scale data in a continuous manner [1]. As such, the stream processing paradigm turned out to be particularly suited to support the implementation of business logic in large organizations. It provides the backbone for communication between independent components operating asynchronously.

### Buffering and Reordering

- Ref: CQL<sup>1</sup>, Trill<sup>2</sup>

### Punctuations/Watermarks

- Ref: Li et al.<sup>3</sup>, Krishnamurthy et al.<sup>4</sup>

### Dual Streaming Model

- Continuous updates/changelogs
- Decouple latency from correctness
- Trade-off latency and cost
- Trade-off cost and completeness (retention time)

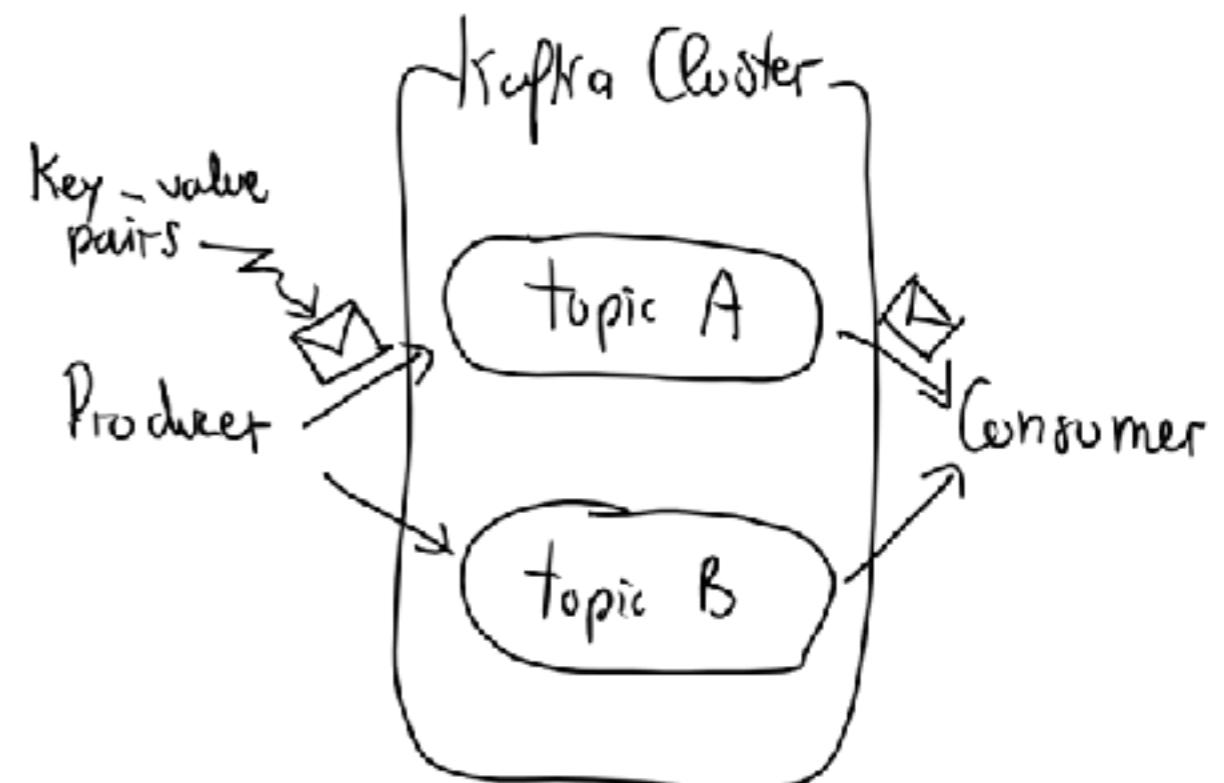
A medium shot of a young man with short brown hair, wearing a purple zip-up hoodie over a white t-shirt. He is looking upwards and slightly to his left with a contemplative expression. His right hand is resting against his chin, supporting his head. The background is dark and out of focus.

Should I Take a Step back?

T  
I

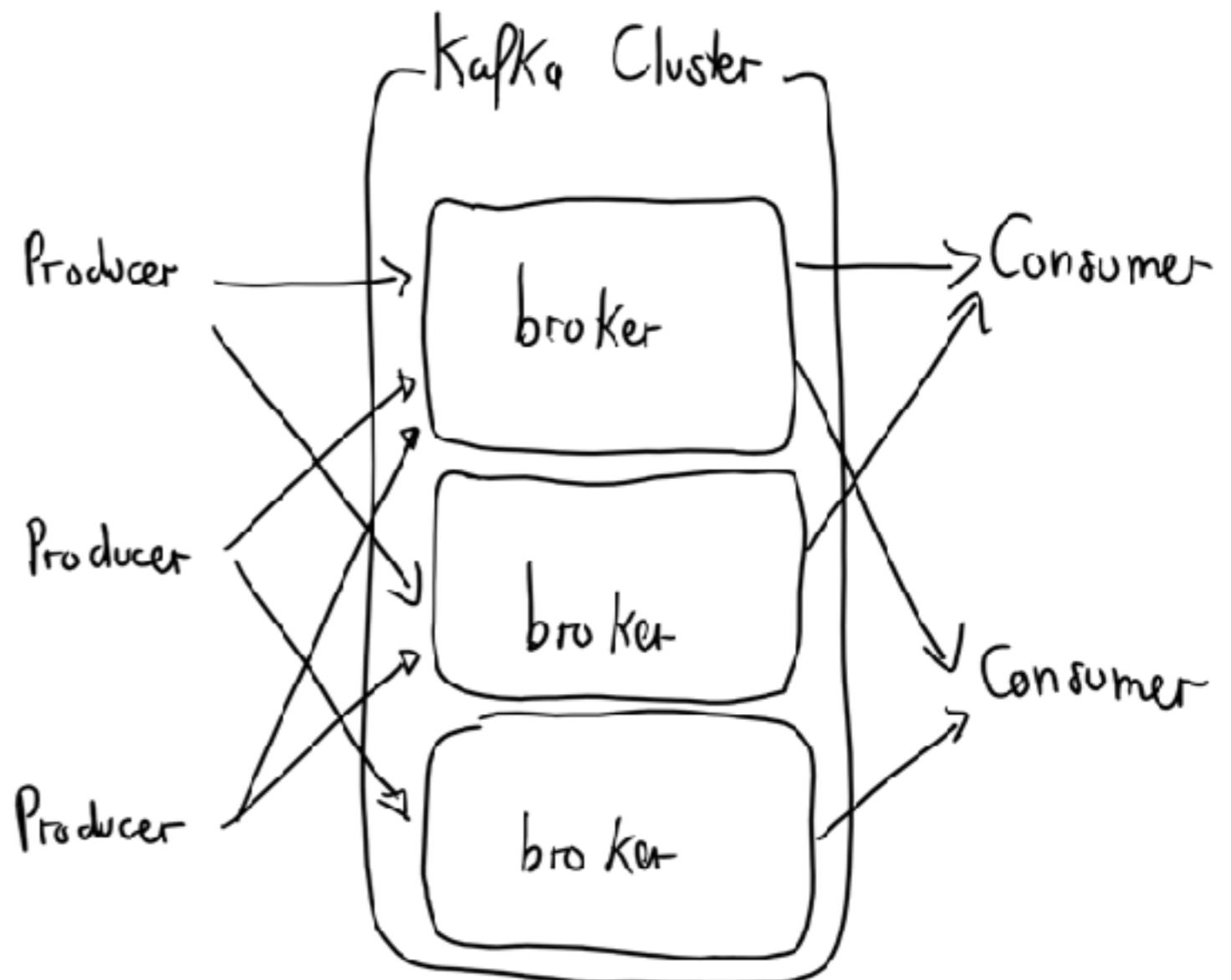
# A Conceptual View of Kafka

- **Producers** send messages on topics
- **Consumers** read messages from topics
- **Messages** are key-value pairs
- **Topics** are streams of messages
- Kafka cluster manages topics



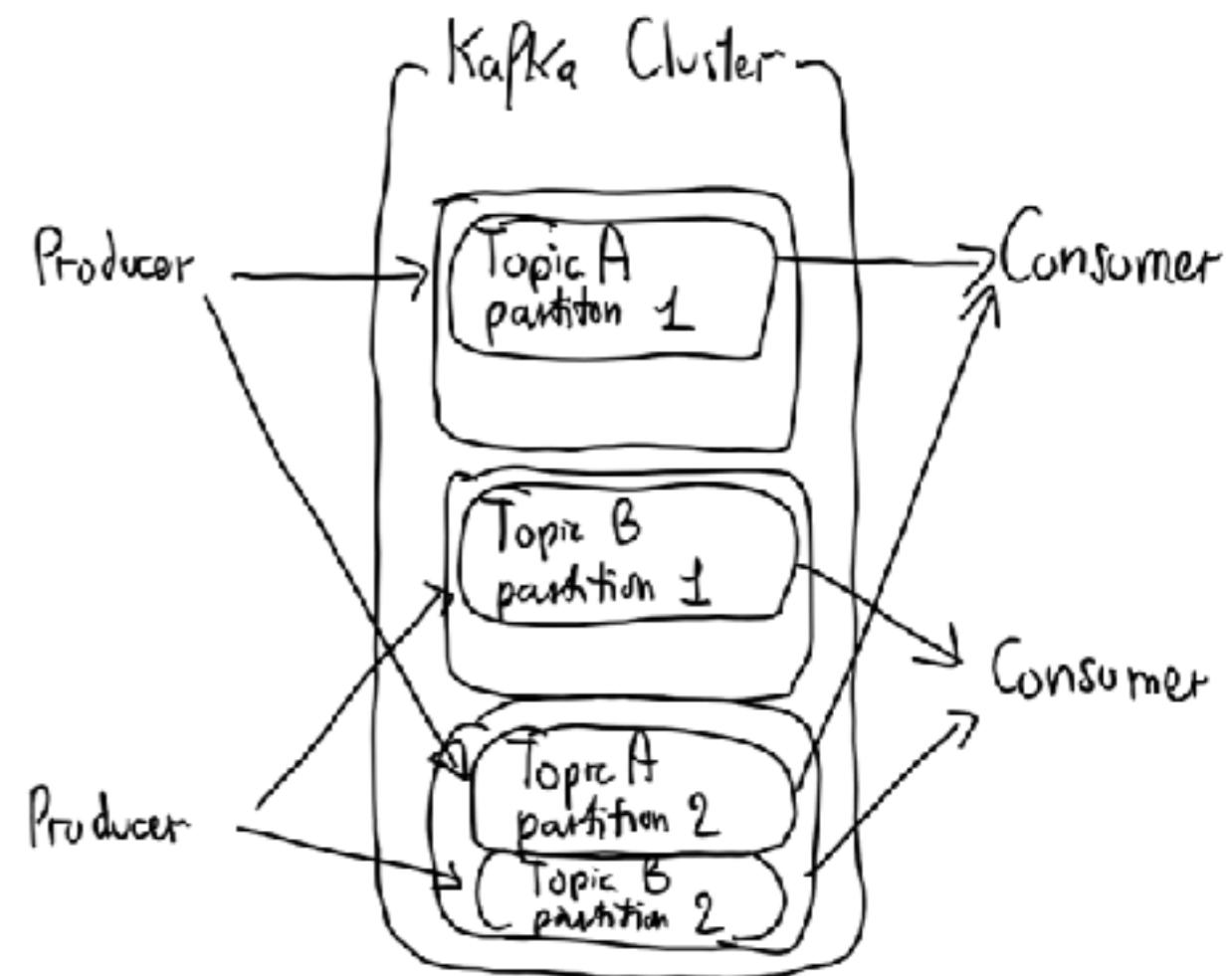
# A Logical View of Kafka

- **Brokers** are the main storage and messaging components of the Kafka cluster



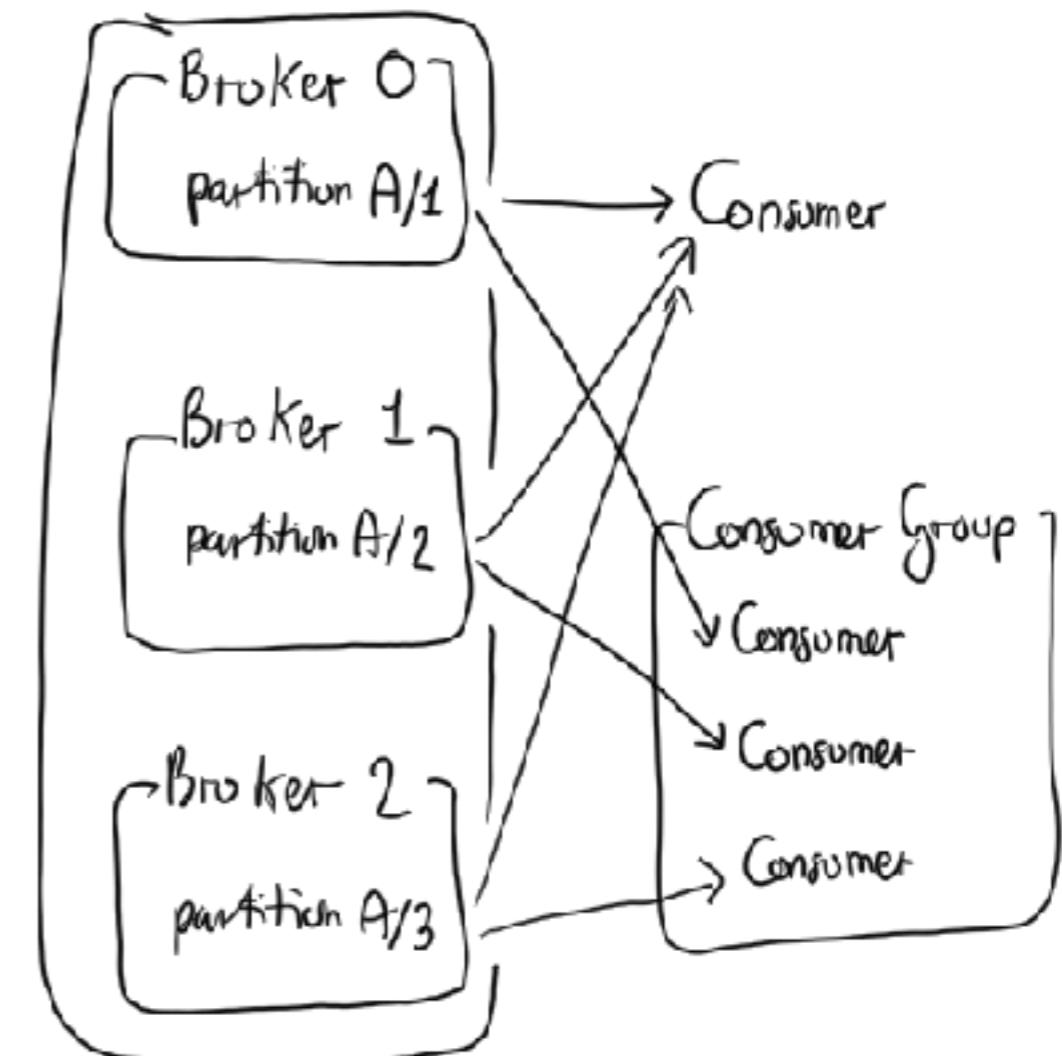
# Reconciling the two views of Kafka

- Topics are partitioned across brokers
- Producers shard messages over the partitions of a certain topic
- Typically, the message key determines which Partition a message is assigned to



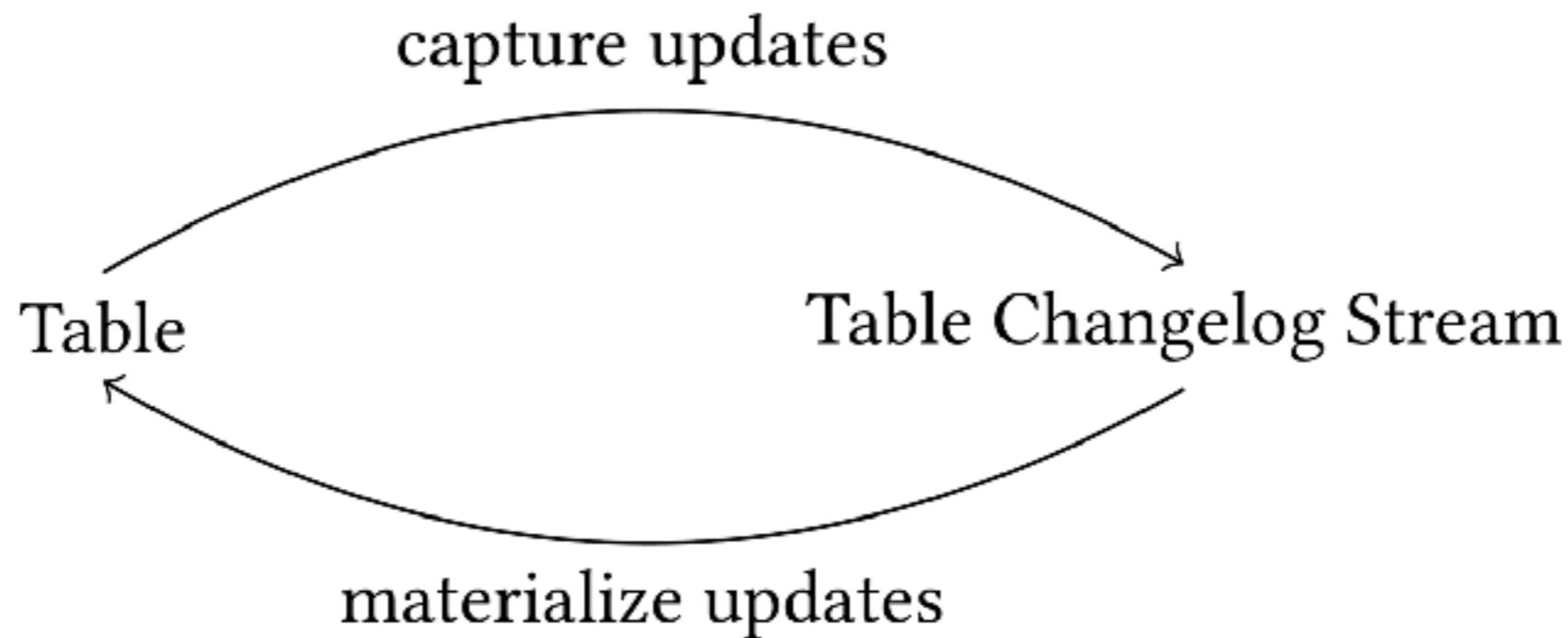
# Topic partitioning invites distributed consumption

- Different Consumers can read data from the same Topic
  - By default, each Consumer will receive all the messages in the Topic
- Multiple Consumers can be combined into a Consumer Group
  - Consumer Groups provide scaling capabilities
  - Each Consumer is assigned a subset of Partitions for consumption



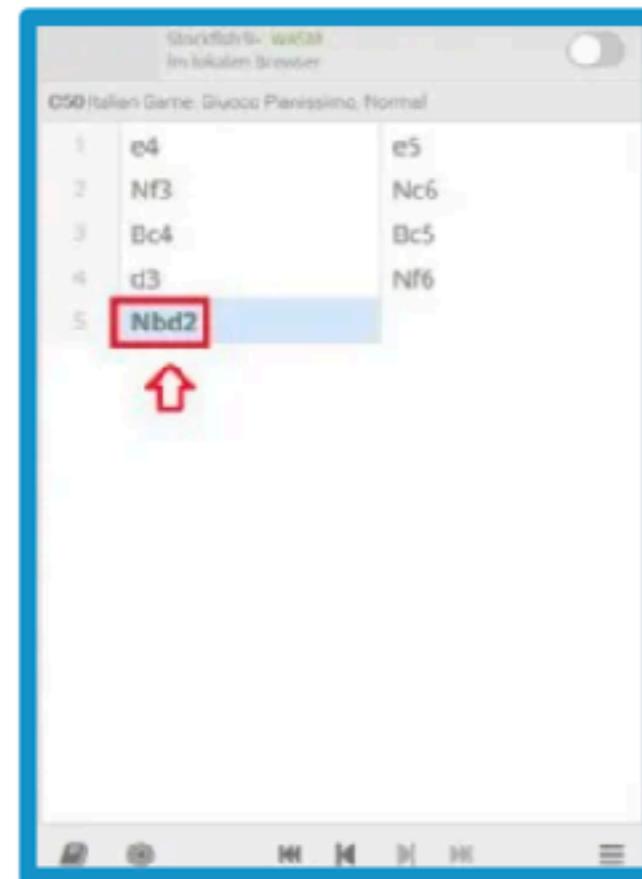
# Dual Stream Model

The intuition



# Dual Stream Model

## The intuition



Streams record History  
“The sequence of moves.”

Tables represent State  
“The state of the board at last move.”

<https://www.michael-noll.com/blog/2018/04/05/of-stream-and-tables-in-kafka-and-stream-processing-part1/>

Matthias J. Sax  
Confluent Inc.  
Palo Alto, USA  
matthias@confluent.io

Guozhang Wang  
Confluent Inc.  
Palo Alto, USA  
guozhang@confluent.io

Johann-Christoph Freytag  
Humboldt-Universität zu Berlin  
Berlin, Germany  
johannes.weidlich@hu-berlin.de

KEYWORDS  
Stream Processing, Processing Model, Semantics

ACM Reference Format:  
Matthias J. Sax, Guozhang Wang, Matthias Weidlich, and Johann-Christoph Freytag. 2018. Streams and Tables: Two Sides of the Same Coin. In *International Workshop on Real-Time Business Intelligence and Analytics (BIRTE '18)*, August 27, 2018, Rio de Janeiro, Brazil. ACM, New York, NY, USA, 10 pages. <https://doi.org/10.1145/3242155>

### 1 INTRODUCTION

Stream processing has emerged as a paradigm to develop real-time applications. It builds on an evaluation of operators over unbounded sequences of data, enabling low-latency processing of large-scale data in a continuous manner [1]. As such, the stream processing paradigm turned out to be particularly suited to support the implementation of business logic in large organizations. It provides the backbone for communication between independent components of a large system, a.k.a. ‘microservices’, through asynchronous message-passing [19].

in the light of real-world requirements

# Dual Stream Model

## The intuition

Streams and Tables: Two Sides of the Same Coin

Matthias J. Sax<sup>\*</sup>  
Confluent Inc.  
Palo Alto, USA  
matthias@confluent.io

Guozhang Wang  
Confluent Inc.  
Palo Alto, USA  
guozhang@confluent.io

Matthias Weidlich  
Humboldt-Universität zu Berlin  
Berlin, Germany  
matthias.weidlich@hu-berlin.de

Johann-Christoph Freytag  
Humboldt-Universität zu Berlin  
Berlin, Germany  
freytag@informatik.hu-berlin.de

KEYWORDS  
Stream Processing, Processing Model, Semantics

ACM Reference Format:  
Matthias J. Sax, Guozhang Wang, Matthias Weidlich, and Johann-Christoph Freytag, 2018. Streams and Tables: Two Sides of the Same Coin. In *International Workshop on Real-Time Business Intelligence and Analytics (BIRTE '18), August 27, 2018, Rio de Janeiro, Brazil*. ACM, New York, NY, USA, 10 pages. <https://doi.org/10.1145/3242132/2155>

ABSTRACT  
Stream processing has emerged as a paradigm to develop real-time applications. It builds on an evaluation of operators over unbounded sequences of data, enabling low-latency processing of large-scale data in a continuous manner [1]. As such, the stream processing paradigm turned out to be particularly suited to support the implementation of business logic in large organizations. It provides the backbone for communication between independent components of a large system, a.k.a. ‘microservices’, through asynchronous message-passing [19].

*\*Author responsible for stream processing framework*

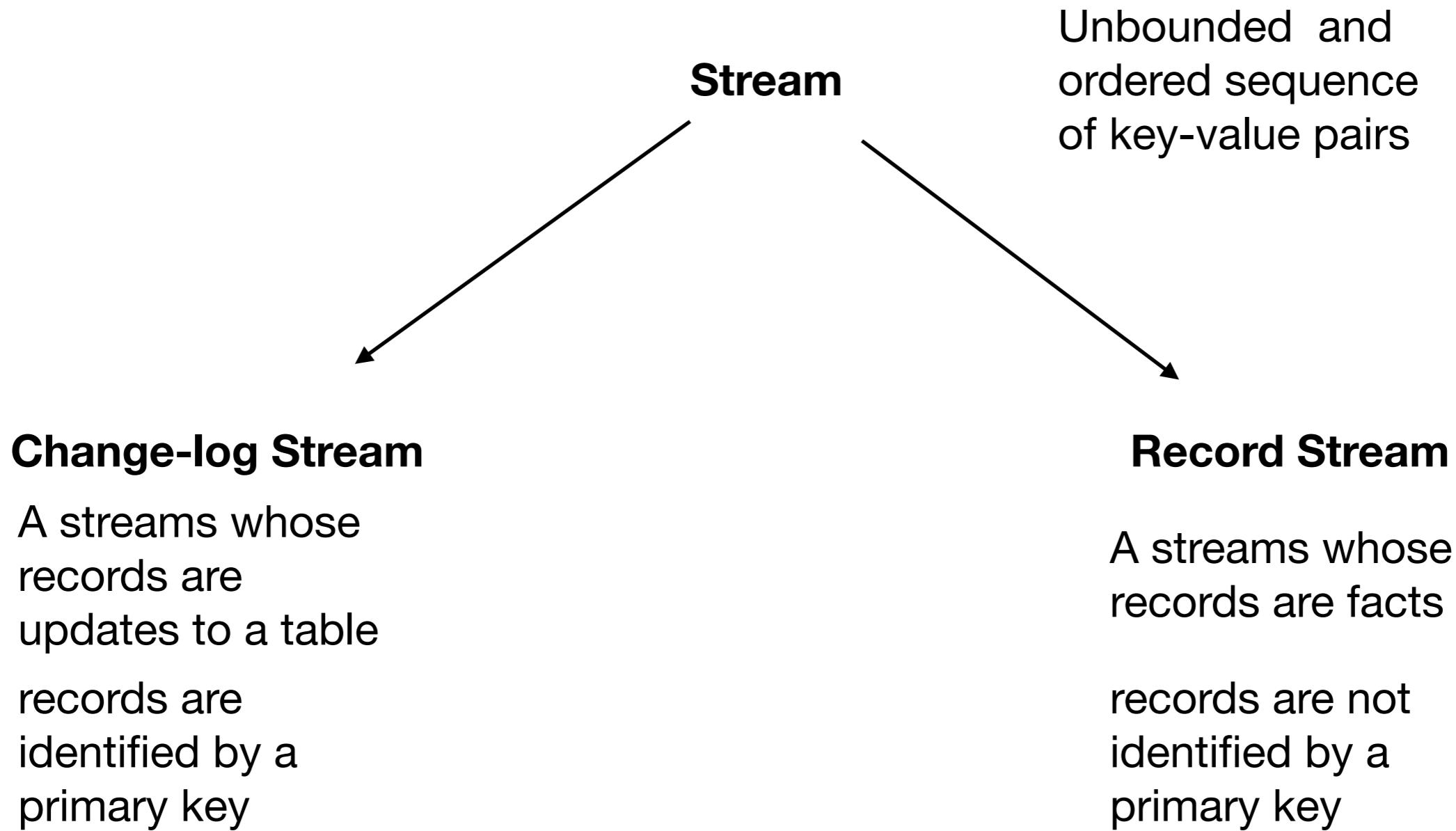
### Table



<https://www.michael-noll.com/blog/2018/04/05/of-stream-and-tables-in-kafka-and-stream-processing-part1/>

# Dual Stream Model

## The Truth about Streams



### Streams and Tables: Two Sides of the Same Coin

Matthias J. Sax<sup>\*</sup>  
Confluent Inc.  
Palo Alto, USA  
matthias@confluent.io

Guozhang Wang  
Confluent Inc.  
Palo Alto, USA  
guozhang@confluent.io

Matthias Weidlich  
Humboldt-Universität zu Berlin  
Berlin, Germany  
matthias.weidlich@hu-berlin.de

Johann-Christoph Freytag  
Humboldt-Universität zu Berlin  
Berlin, Germany  
freytag@informatik.hu-berlin.de

### KEYWORDS

Stream Processing, Processing Model, Semantics

ACM Reference Format:  
Matthias J. Sax, Guozhang Wang, Matthias Weidlich, and Johann-Christoph Freytag, 2018. Streams and Tables: Two Sides of the Same Coin. In *International Workshop on Real-Time Business Intelligence and Analytics (BIRTE '18)*, August 27, 2018, Rio de Janeiro, Brazil. ACM, New York, NY, USA, 10 pages. <https://doi.org/10.1145/3242132/2155>

### 1 INTRODUCTION

Stream processing has emerged as a paradigm to develop real-time applications. It builds on an evaluation of operators over unbounded sequences of data, enabling low-latency processing of large-scale data in a continuous manner [1]. As such, the stream processing paradigm turned out to be particularly suited to support the implementation of business logic in large organizations. It provides the backbone for communication between independent components of a large system, a.k.a. ‘microservices’, through asynchronous message-passing [19].

†Authors contributing equally to this work.

Matthias J. Sax<sup>\*</sup>  
Confluent Inc.  
Palo Alto, USA  
matthias@confluent.io

Guozhang Wang  
Confluent Inc.  
Palo Alto, USA  
guozhang@confluent.io

Johann-Christoph Freytag  
Humboldt-Universität zu Berlin  
Berlin, Germany  
johann-christoph.freytag@informatik.hu-berlin.de

**KEYWORDS**  
Stream Processing, Processing Model, Semantics

**ACM Reference Format:**  
Matthias J. Sax, Guozhang Wang, Matthias Weidlich, and Johann-Christoph Freytag, 2018. Streams and Tables: Two Sides of the Same Coin. In *International Workshop on Real-Time Business Intelligence and Analytics (BIRTE '18)*, August 27, 2018, Rio de Janeiro, Brazil. ACM, New York, NY, USA, 10 pages. <https://doi.org/10.1145/3242152>

# Dual Stream Model

## The Truth about Tables

A table is a collection of table versions; one version for each point in time using the timestamp as a version number.

$$T(1) = \{T_5 = \{\langle A, 7.2 \rangle\}\}$$

$$T(2) = \{T_5 = \{\langle A, 7.2 \rangle\}, T_6 = \{\langle B, 14.7 \rangle\}\}$$

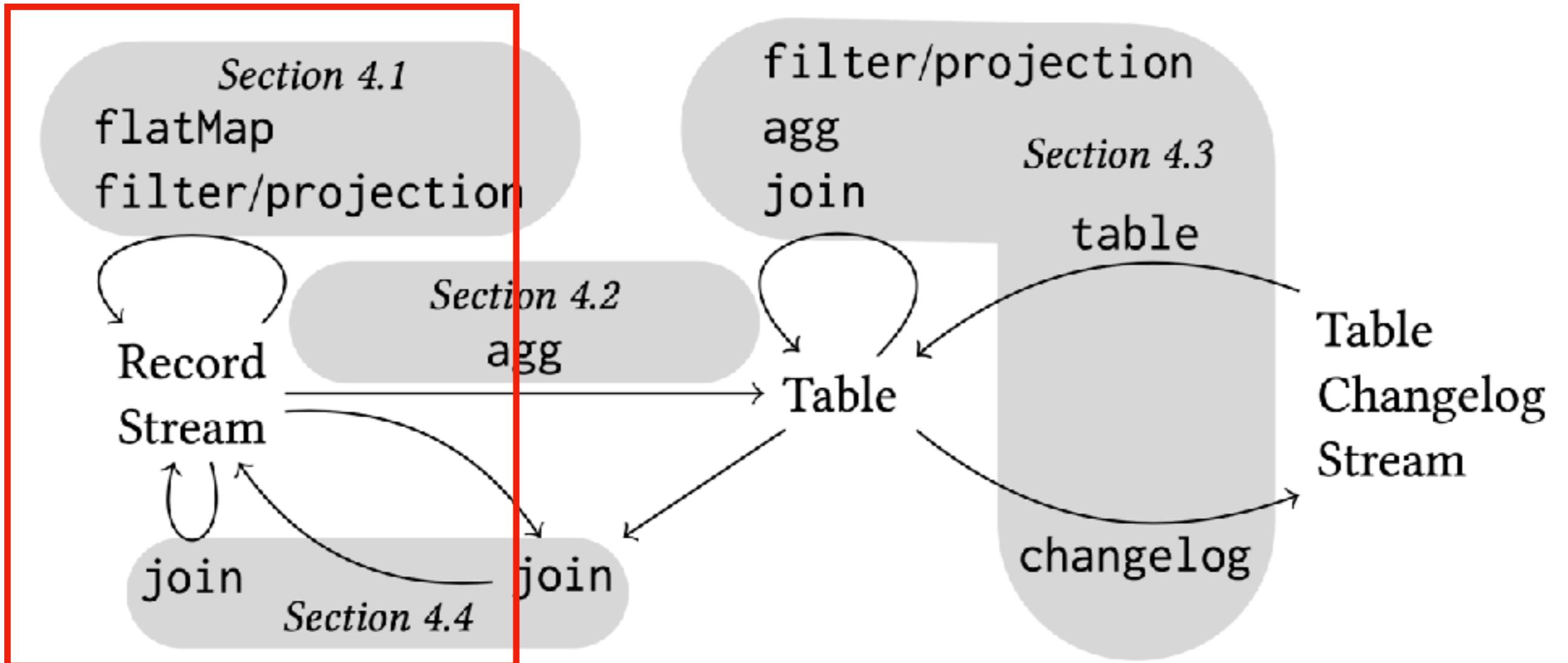
$$T(3) = \{T_5 = \{\langle A, 7.2 \rangle\}, T_6 = \{\langle A, 8.9 \rangle, \langle B, 14.7 \rangle\}\}$$

$$T(4) = \{T_3 = \{\langle B, 12.1 \rangle\}, T_5 = \{\langle A, 7.2 \rangle, \langle B, 12.1 \rangle\}, T_6 = \{\langle A, 8.9 \rangle, \langle B, 14.7 \rangle\}\}$$

$$T(5) = \{T_3 = \{\langle B, 12.1 \rangle\}, T_5 = \{\langle A, 7.2 \rangle, \langle B, 12.1 \rangle\}, T_6 = \{\langle A, 8.9 \rangle, \langle B, 14.7 \rangle\}, T_8 = \{\langle A, 8.9 \rangle, \langle B, 16.7 \rangle\}\}$$

# Dual Stream Model

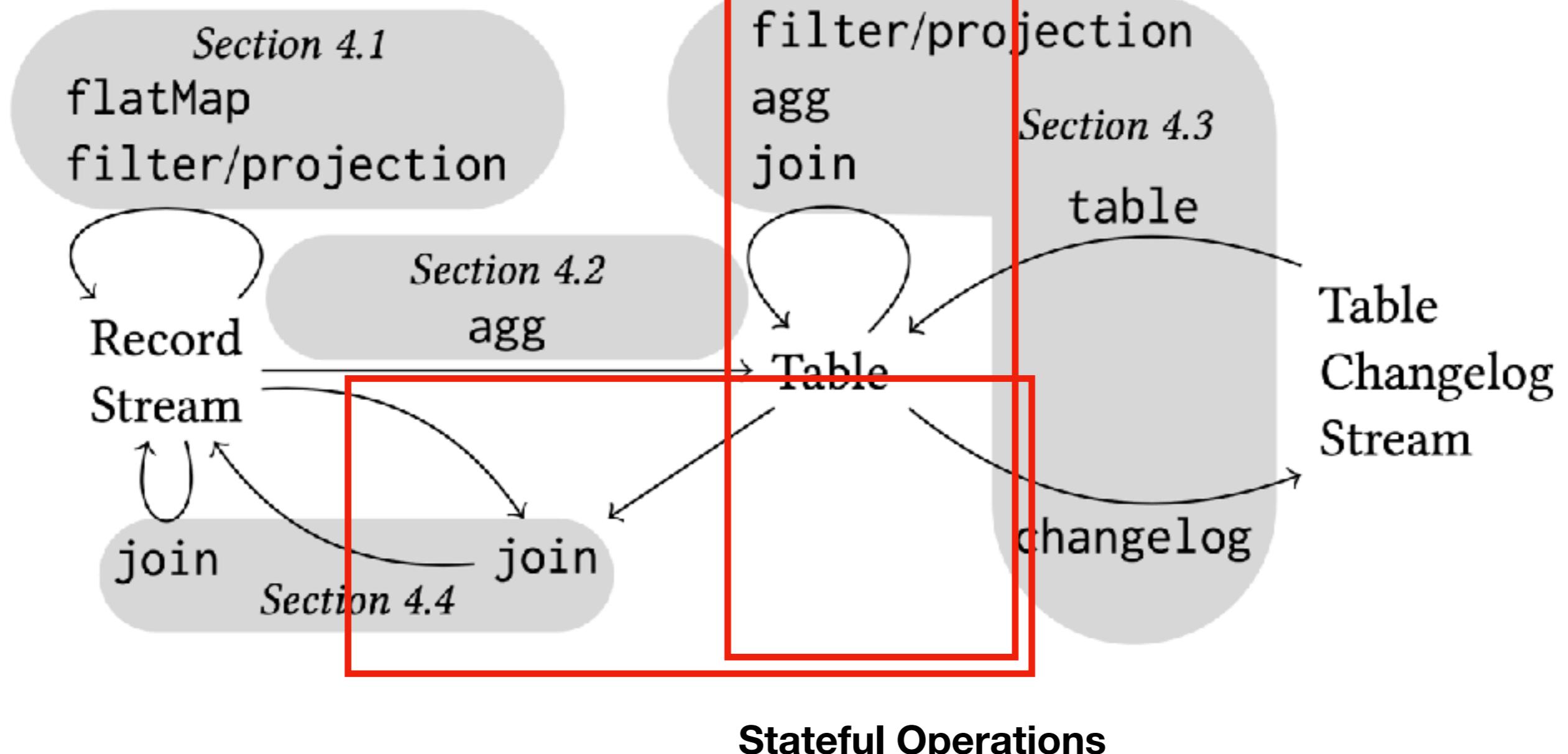
## The Truth about Operations



**Stateless Operations**

# Dual Stream Model

## The Truth about Operations



Matthias J. Sax<sup>\*</sup>  
Confluent Inc.  
Palo Alto, USA  
matthias@confluent.io

Guozhang Wang  
Confluent Inc.  
Palo Alto, USA  
guozhang@confluent.io

Matthias Weidlich  
Humboldt-Universität zu Berlin  
Berlin, Germany  
matthias.weidlich@hu-berlin.de

Johann-Christoph Freytag  
Humboldt-Universität zu Berlin  
Berlin, Germany  
freytag@informatik.hu-berlin.de

#### KEYWORDS

Stream Processing, Processing Model, Semantics  
ACM Reference Format:  
Matthias J. Sax, Guozhang Wang, Matthias Weidlich, and Johann-Christoph Freytag, 2018. Streams and Tables: Two Sides of the Same Coin. In *International Workshop on Real-Time Business Intelligence and Analytics (BIRTE '18)*, August 27, 2018, Rio de Janeiro, Brazil. ACM, New York, NY, USA, 10 pages. <https://doi.org/10.1145/3242132/2155>

# Dual Stream Model

## The Complete Picture

**Stream**  
(changelog)

**Table**

**Stream**

alice	Paris
-------	-------

older data → newer data

<https://www.michael-noll.com/blog/2018/04/05/of-stream-and-tables-in-kafka-and-stream-processing-part1/>

Matthias J. Sax<sup>\*</sup>  
Confluent Inc.  
Palo Alto, USA  
matthias@confluent.io

Guozhang Wang  
Confluent Inc.  
Palo Alto, USA  
guozhang@confluent.io

Matthias Weidlich  
Humboldt-Universität zu Berlin  
Berlin, Germany  
matthias.weidlich@hu-berlin.de

Johann-Christoph Freytag  
Humboldt-Universität zu Berlin  
Berlin, Germany  
freytag@informatik.hu-berlin.de

## KEYWORDS

Stream Processing, Processing Model, Semantics  
ACM Reference Format:  
Matthias J. Sax, Guozhang Wang, Matthias Weidlich, and Johann-Christoph Freytag, 2018. Streams and Tables: Two Sides of the Same Coin. In *International Workshop on Real-Time Business Intelligence and Analytics (BIRTE '18)*, August 27, 2018, Rio de Janeiro, Brazil. ACM, New York, NY, USA, 10 pages. <https://doi.org/10.1145/3242132/2155>

# Dual Stream Model

## Result Correctness vs Runtime Cost

**TRACTABILITY** has emerged as a paradigm in applications that require low-latency evaluation of operators on unbounded sequences of data. Defining the semantics of stream processing is challenging due to the presence of delayed data sources. The physical and logical order of data stream may become inconsistent in such a setting. Experiments have shown that it is possible to handle such inconsistencies by means of data buffering and reordering techniques, thereby compromising processing latency.

In this paper, we introduce the Dual Streaming Model to reason about physical and logical order in data stream processing. This model presents the result of an operator as a sequence of successive updates, which induces a duality of transformations. As such, it provides a natural way to cope with inconsistencies between the physical and logical order of data in a continuous manner, without explicitly defining them or reordering. We further discuss the trade-offs and challenges faced when implementing this model in terms of correctness, latency, and processing cost. A case study on Apache Kafka illustrates the effectiveness of our approach in the light of real-world requirements.

**1 INTRODUCTION**  
Stream processing has emerged as a paradigm to develop real-time applications. It builds on an evaluation of operators over unbounded sequences of data, enabling low-latency processing of large-scale data in a continuous manner [1]. As such, the stream processing paradigm turned out to be particularly suited to support the implementation of business logic in large organizations. It provides the backbone for communication between independent components of a large system, a.k.a. ‘microservices’, through asynchronous message-passing [19].

†Authors contributed equally to this work.

The DSM simplifies the **reasoning** about the **transformations**, but does not solve the **unboundedness** problem.

We still need infinite memory for processing an infinite stream.

DSM introduces the retention time to make the trade-off explicit.

Matthias J. Sax  
Confluent Inc.  
Palo Alto, USA  
matthias@confluent.io

Guozhang Wang  
Confluent Inc.  
Palo Alto, USA  
guozhang@confluent.io

Matthias Weidlich  
Humboldt-Universität zu Berlin  
Berlin, Germany  
matthias.weidlich@hu-berlin.de

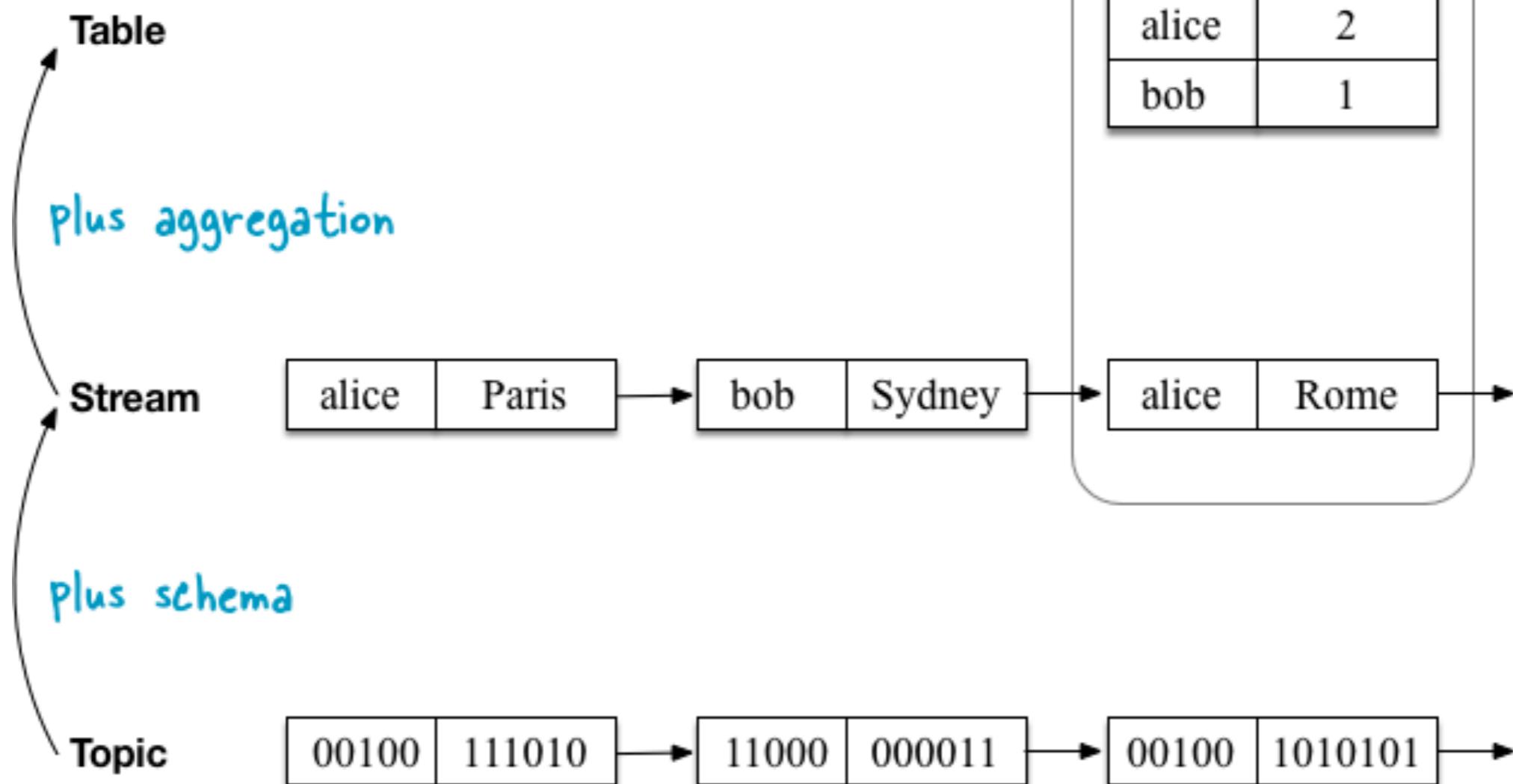
Johann-Christoph Freytag  
Humboldt-Universität zu Berlin  
Berlin, Germany  
freytag@informatik.hu-berlin.de

**KEYWORDS**  
Stream Processing, Processing Model, Semantics

**ACM Reference Format:**  
Matthias J. Sax, Guozhang Wang, Matthias Weidlich, and Johann-Christoph Freytag, 2018. Streams and Tables: Two Sides of the Same Coin. In *International Workshop on Real-Time Business Intelligence and Analytics (BIRTE '18), August 27, 2018, Rio de Janeiro, Brazil*. ACM, New York, NY, USA, 10 pages. <https://doi.org/10.1145/3242152>

# Mapping to Kafka

## Minimal



<https://www.michael-noll.com/blog/2018/04/05/of-stream-and-tables-in-kafka-and-stream-processing-part1/>

Matthias J. Sax<sup>\*</sup>  
Confluent Inc.  
Palo Alto, USA  
matthias@confluent.io

Guozhang Wang  
Confluent Inc.  
Palo Alto, USA  
guozhang@confluent.io

Matthias Weidlich  
Humboldt-Universität zu Berlin  
Berlin, Germany  
matthias.weidlich@hu-berlin.de

Johann-Christoph Freytag  
Humboldt-Universität zu Berlin  
Berlin, Germany  
freytag@informatik.hu-berlin.de

KEYWORDS  
Stream Processing, Processing Model, Semantics

ACM Reference Format:  
Matthias J. Sax, Guozhang Wang, Matthias Weidlich, and Johann-Christoph Freytag, 2018. Streams and Tables: Two Sides of the Same Coin. In *International Workshop on Real-Time Business Intelligence and Analytics (BIRTE '18), August 27, 2018, Rio de Janeiro, Brazil*. ACM, New York, NY, USA, 10 pages. <https://doi.org/10.1145/3242155>

1 INTRODUCTION  
Stream processing has emerged as a paradigm for applications that require low-latency processing of unbounded sequences of data. Using the semantics of stream processing is challenging due to the presence of distributed data sources. The physical and logical order of data streams can be inconsistent in such a setting. Examples include managing dependencies or handle by means of data buffering and reordering techniques, thereby compromising processing latency.

In this paper, we introduce the Dual Streaming Model to reason about physical and logical order in data stream processing. This model presents the result of an operator as a sequence of successive updates, which induces a duality of streams and tables. As such, it provides a natural way to cope with inconsistencies between the physical and logical ordering of data in a continuous manner, without explicitly specifying and reordering. We further discuss the trade-offs and challenges faced when implementing this model in terms of correctness, latency, and processing cost. A case study on Apache Kafka illustrates the effectiveness of our approach in the light of real-world requirements.

# Mapping to Kafka

## Minimal

Concept	Partitioned	Unbounded	Ordering	Mutable	Unique key	Schema
Topic	Yes	Yes	Yes	No	No	No (raw)
Stream	Yes	Yes	Yes	No	No	Yes
Table	Yes	Yes	No	Yes	Yes	Yes

Concept	Kafka	KSQL	Java	Scala	Python
Topic	-	-	List/Stream	List/Stream[ (Array[Byte],	[ ]
Stream	KStream	STREAM	List/Stream	List/Stream[ (K, V) ]	[ ]
Table	KTable	TABLE	HashMap	mutable.Map[ K, V ]	{ }

<https://www.michael-noll.com/blog/2018/04/05/of-stream-and-tables-in-kafka-and-stream-processing-part1/>

# Streaming SQL

- New trend is hiding the complexity of the processing behind SQL
- Like CQL, but even simpler.
- Visit [streaminglands.io](https://streaminglands.io)

## 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

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

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

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

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

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

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

**KEYWORDS**  
stream processing, data management, query processing

**ACM Reference Format:**  
Edmon Begoli, Tyler Alcidau, 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 29, 2019, Amsterdam, Netherlands. ACM, New York, NY, USA, 16 pages. <https://doi.org/10.1145/3290600.3314145>

### 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 modeling, 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 al-

Thanks!  
Questions?

