The CQL Continuous Query Language: Semantic Foundations and Query Execution

Arvind Arasu and Shivnath Babu and Jennifer Widom

University of Stanford

presented by Weichen Wang
2016.10.26

Outline

- Data Streams Overview
- The Linear Road: A Benchmark
- The Formal Model
  - Design Goals
  - Streams, Relations and Operators
  - Semantics for Continuous Query
- Continuous Query Language (CQL)
  - Specification
  - Implementation: STREAM
- Future Works

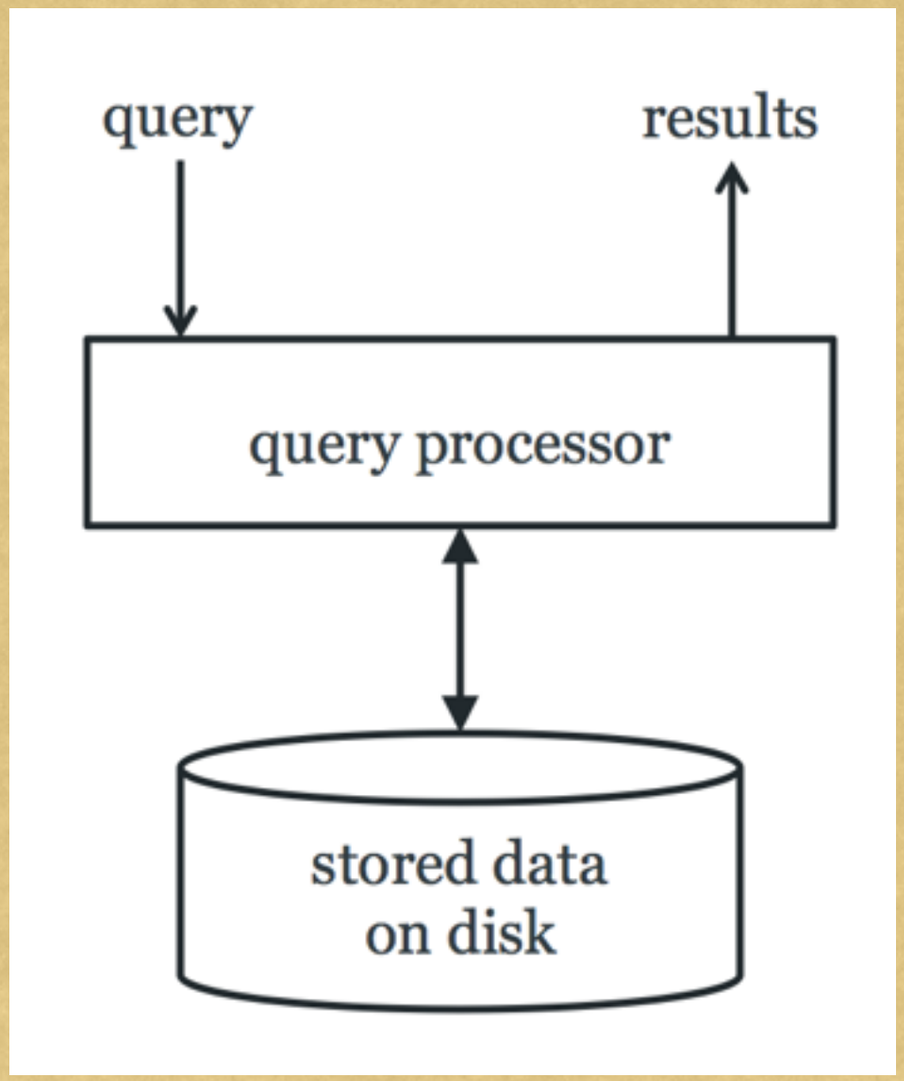
Data Streams

- Continuous streams of data elements (may be unbounded)
  - Telecommunications
  - Sensor networks (road network, weather stations)
  - Transaction logs
  - Financial applications
  - Router traffic
  - Tweets

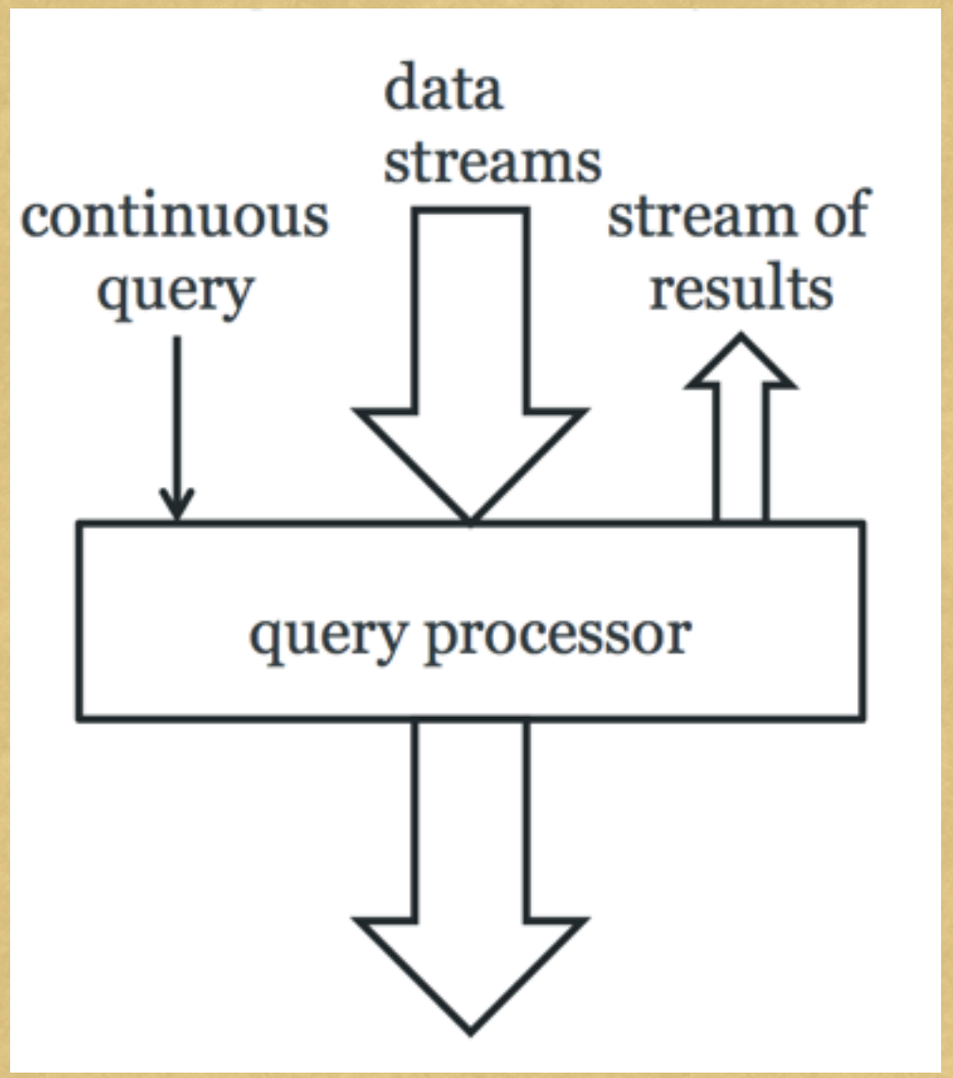
- DSMS = Data Stream Management System

Data Stream Management System (DSMS)

DBMS



DSMS



Data Stream Management System (DSMS)

DBMS	DSMS
➤ Persistent relations	➤ Transient streams (and persistent relations)
➤ One-time query	➤ Continuous queries
➤ Random access	➤ Sequential access
➤ Only current state matters	➤ Historical data matters

The Linear Road

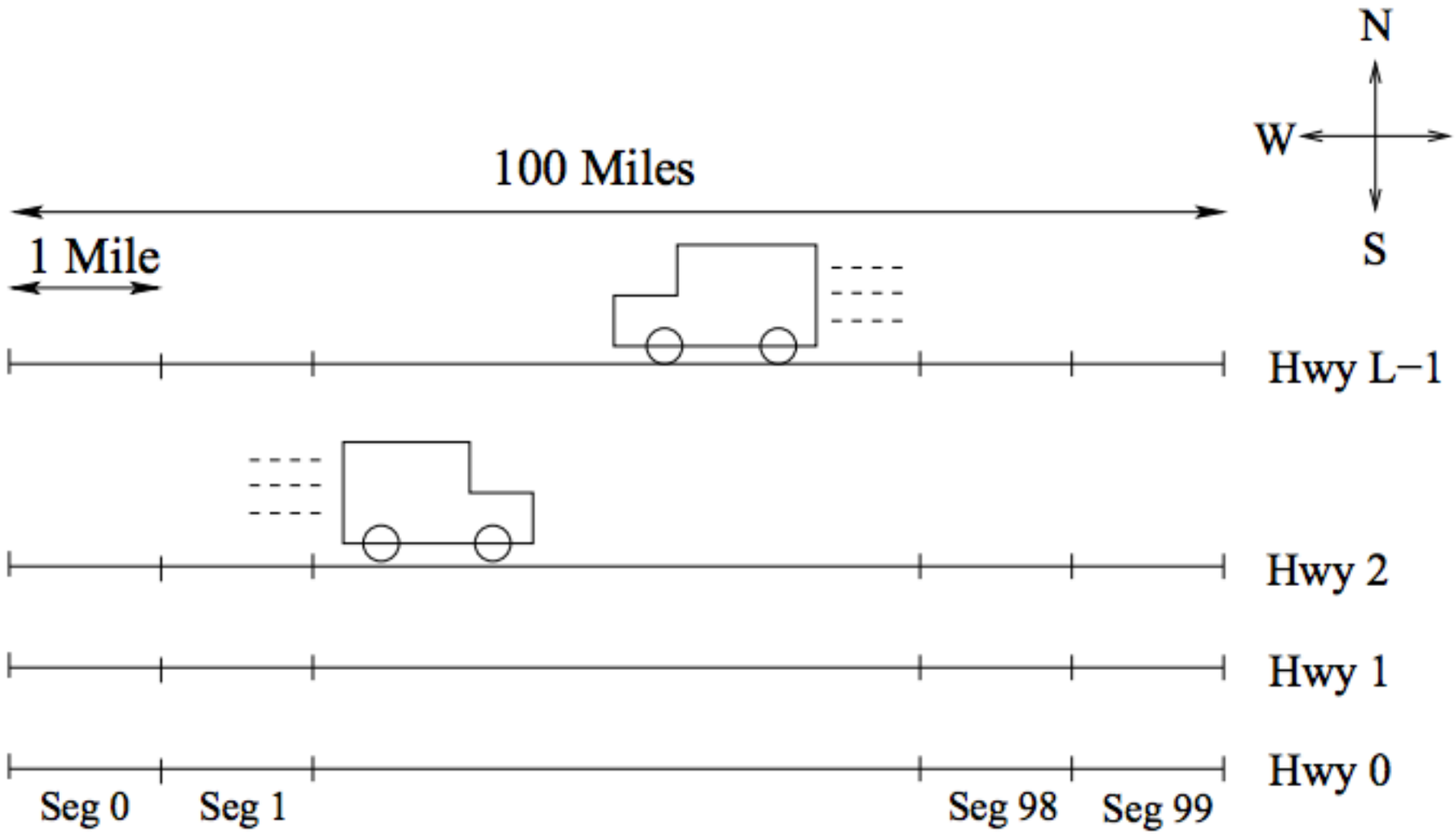


Figure 1: The Linear Road highway system

The Linear Road: A Benchmark

- a hypothetical road traffic management application for DSMS
- adaptive, real-time computation of vehicle tolls based on traffic conditions.
  - $toll = basetoll \times (numVehicles - 150)^2$, *if there is congestion.*
- The simplified *Linear Road* application in this paper has:
  - A single ***input stream***: the stream of positions and speeds of vehicles
  - A single ***continuous query*** computing the tolls
  - A single ***output stream*** of the tolls

The Formal Model: Streams

Definition: A *stream* $\boldsymbol{S}$ is a (possibly infinite) bag (multiset) of *elements* $\langle \boldsymbol{s}, \boldsymbol{\tau} \rangle$, where s is a tuple belonging to the schema of $\boldsymbol{S}$ and $\boldsymbol{\tau} \in \boldsymbol{T}$ is the *timestamp* of the element.

- *Base stream*: a source data stream that arrives at the DSMS
- *Derived stream*: an intermediate stream produced by query operators
- *Tuple of a stream*: the data (non-timestamp) portion of a stream

- $\boldsymbol{S}$ *up to* $\boldsymbol{\tau}$: the bag of elements in stream $\boldsymbol{S}$ with timestamps $\leq \boldsymbol{\tau}$
  - i.e., $\{\langle \boldsymbol{s}, \boldsymbol{\tau'} \rangle \in \boldsymbol{S} : \boldsymbol{\tau'} \leq \boldsymbol{\tau}\}$.
- $\boldsymbol{S}$ *at* $\boldsymbol{\tau}$: the bag of elements in stream $\boldsymbol{S}$ with timestamps $= \boldsymbol{\tau}$
  - i.e., $\{\langle \boldsymbol{s}, \boldsymbol{\tau'} \rangle \in \boldsymbol{S} : \boldsymbol{\tau'} = \boldsymbol{\tau}\}$.

Example: In the *Linear Road* application, there is just one *base stream* containing vehicle speed-position measurements, with schema:

- PosSpeedStr(vehicleId, speed, xPos, dir, hwy)

The Formal Model: Relations

Definition: A *relation* R is a mapping from T to a finite but unbounded bag of tuples belonging to the schema of R.

- *Relation* : a time-varying bag of tuples
- *Instantaneous relation*: a relation in the traditional bag-of-tuples sense
  - R is a *relation* $\Rightarrow$ $R(\tau)$ is an *instantaneous relation*.
- *Base relation:* an input relation from source data
- *Derived relation*: a relation produced by query operators

Example: In the *Linear Road* application, the toll for a congested segment depends on the current number of vehicles in the segment, which can be represented in a *derived relation*:

- SegVolRel(segNo, dir, hwy, numVehicles)

The Formal Model: Operators

- *Stream-to-relation operator*: produce a *relation* from a *stream*
- *Relation-to-relation operator*: produce a *relation* from one or more other *relations*
- *Relation-to-stream operator*: produce a *stream* from a *relation*

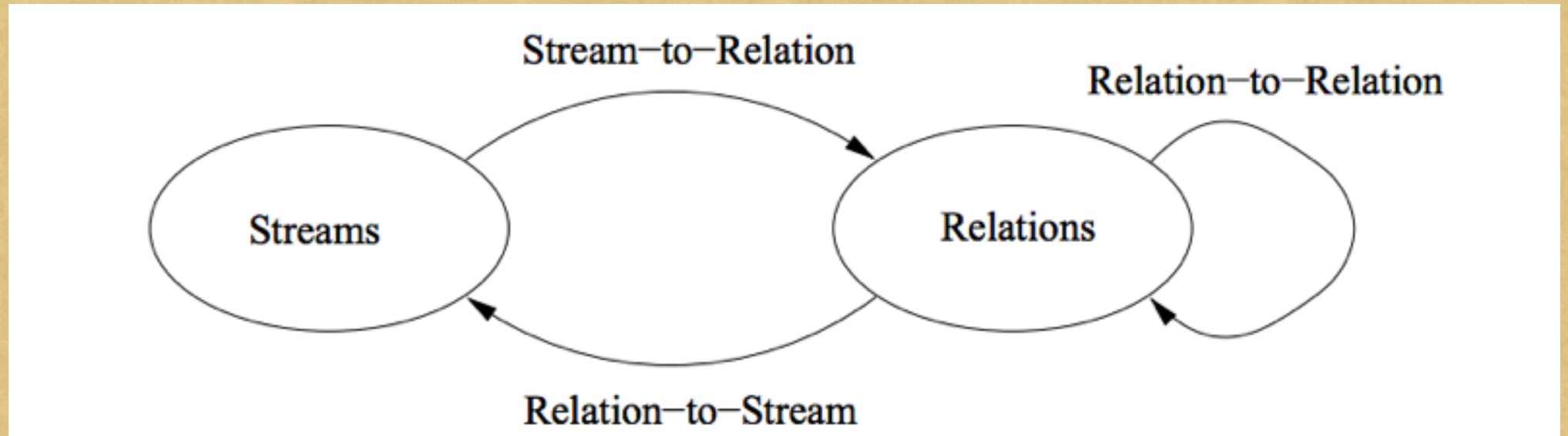


Figure 2: Operator classes and mappings used in abstract semantics

- Stream-to-stream operators are absent

The Formal Model: Continuous Semantics

- $\boldsymbol{Q}$: a query of type-consistent composition of operators.
- $\boldsymbol{S_1 \ldots S_n}$: input streams to the innermost operators of $\boldsymbol{Q}$. ($n \geq 0$)
- $\boldsymbol{R_1 \ldots R_m}$: input relations to the innermost operators of $\boldsymbol{Q}$. ($m \geq 0$)

Definition: The result of *continuous query* $\boldsymbol{Q}$ *at time* $\boldsymbol{\tau}$ is the result of $\boldsymbol{Q}$ once all inputs up to $\boldsymbol{\tau}$ are "available". There are two cases:

- Case 1: The outermost operator in $\boldsymbol{Q}$ is *relation-to-stream*, producing a stream $\boldsymbol{S}$. The result of $\boldsymbol{Q}$ at time $\boldsymbol{\tau}$ is $\boldsymbol{S}$ up to $\boldsymbol{\tau}$, produced by recursively applying the operators comprising $\boldsymbol{Q}$ to streams $\boldsymbol{S_1 \ldots S_n}$ up to $\boldsymbol{\tau}$ and relations $\boldsymbol{R_1 \ldots R_m}$ up to $\boldsymbol{\tau}$.
- Case 2: The outermost operator in $\boldsymbol{Q}$ is *stream-to-relation* or *relation-to-relation*, producing a relation $\boldsymbol{R}$. The result of $\boldsymbol{Q}$ at time $\boldsymbol{\tau}$ is $\boldsymbol{R(\tau)}$, produced by recursively applying the operators comprising $\boldsymbol{Q}$ to streams $\boldsymbol{S_1 \ldots S_n}$ up to $\boldsymbol{\tau}$ and relations $\boldsymbol{R_1 \ldots R_m}$ up to $\boldsymbol{\tau}$.

Example: $Q(S_1, R_1, R_2) = O_1(O_2(S_1), O_3(R_1, R_2))$

The Formal Model: Time Advances

Time "advances" to τ from $\tau - 1$ when all inputs up to $\tau - 1$ have been processed. Assumptions:

- streams arrive in timestamp order
- relations are updated in timestamp order
- no timestamp "skew" across streams or relations

Example: In the *Linear Road* application, the sequence of operators producing derived relation SegVolRel conceptually produces, at every time instant τ , the instantaneous relation SegVolRel(τ) containing the current number of vehicles in each segment.

- SegVolRel(τ) cannot be produced until all elements on input stream SegVolRel(segNo, dir, hwy, numVehicles) with timestamp $\leq \tau$ have been received.

CQL: Design Goals and Strategies

Design Goals

- To exploit well-understood relational semantics.
- To keep queries simple to write and easy to understand.

General Design Strategy

- Support a large number of relation-to-relation operators, with a small set of stream-to-relation and relation-to-stream operators
- Reuse the formal foundations and huge body of implementation techniques for relation-to-relation languages such as relational algebra and SQL

CQL: Stream-to-Relation Operators

All stream-to-relation operators in CQL are based on a *sliding window*.

- A *sliding window* of a stream is a window that at any point of time contains a historical snapshot of a finite portion of a stream.
- *Time-based* sliding window:
  $\mathbf{R}_T(\boldsymbol{\tau}) = \{\mathbf{s} \mid \langle \mathbf{s}, \boldsymbol{\tau}' \rangle \in \mathbf{S} \wedge (\max\{\boldsymbol{\tau} - \mathbf{T}, 0\} \leq \boldsymbol{\tau}' \leq \boldsymbol{\tau})\}$
  Example:"PosSpeedStr [Range 10 Seconds]" is a time-based sliding window of 10 seconds over input stream PosSpeedStr.
- *Tuple-based* sliding window:
  $\mathbf{R}_N(\boldsymbol{\tau}) = \{\mathbf{s} \mid \langle \mathbf{s}, \boldsymbol{\tau}' \rangle \in \mathbf{S} \wedge (\boldsymbol{\tau}'$ is the largest N timestamps$\leq \boldsymbol{\tau})\}$
- *Partitioned* sliding window:
  $\mathbf{R}_{N,A}(\boldsymbol{\tau}) = \{\mathbf{s} \mid \langle \mathbf{s}, \boldsymbol{\tau}' \rangle \in \mathbf{S} \wedge (\boldsymbol{\tau}'$ is the largest N timestamps$\leq \boldsymbol{\tau}$ for all $\langle \mathbf{s}'', \boldsymbol{\tau}'' \rangle \in \mathbf{S}$ s.t. $\mathbf{A}(\mathbf{s}'') = \mathbf{A}(\mathbf{s}))\}$. **A** is a partition function.

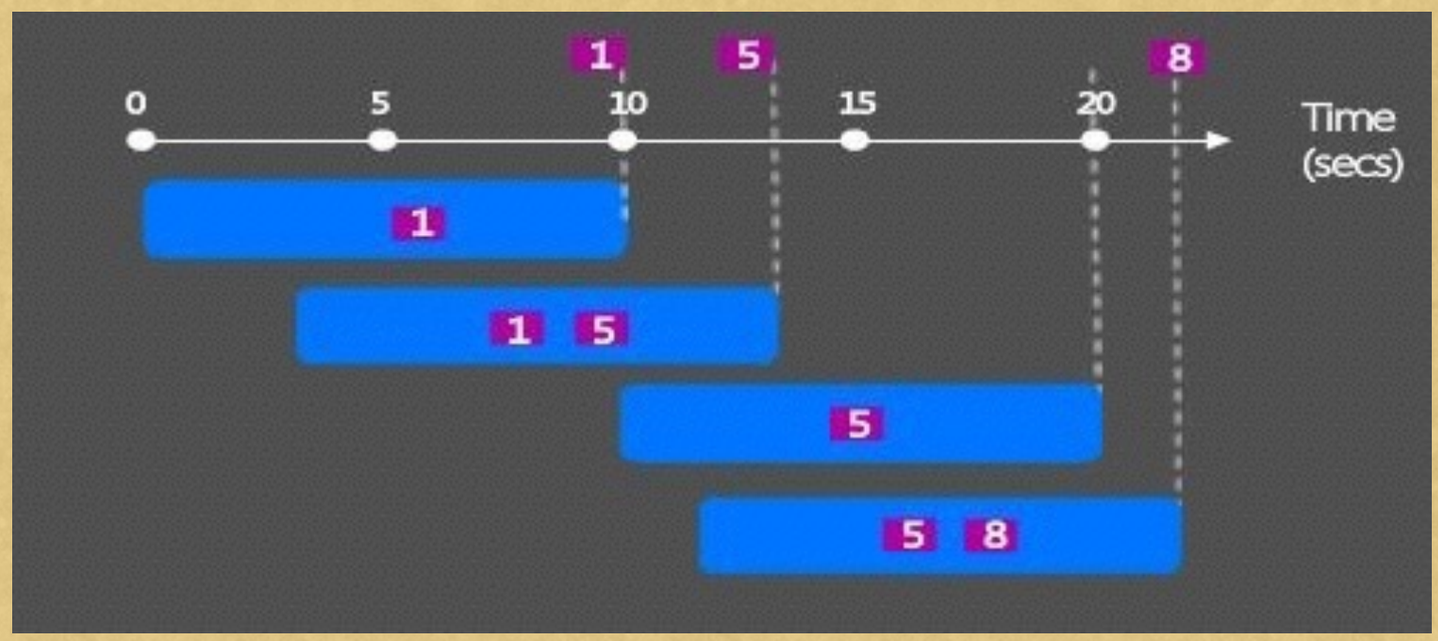
CQL: Relation-to-Relation Operators

All relation-to-relation operators in CQL are derived from traditional relational queries expressed in SQL.

- Anywhere a traditional relation is referenced in a SQL query, a relation can be referenced in CQL.

Example: for the Linear Road application, consider this CQL query:

Select Distinct vehicleId
From PosSpeedStr [Range 10 Seconds]



CQL: Relation-to-Stream Operators

CQL has three relation-to-stream operators:

- *Istream* (for "insert stream"):

$$\texttt{Istream}(R) = \bigcup_{\tau \geq 0} ((R(\tau) - R(\tau - 1)) \times \{\tau\})$$

- *Dstream* (for "delete stream"):

$$\texttt{Dstream}(R) = \bigcup_{\tau > 0} ((R(\tau - 1) - R(\tau)) \times \{\tau\})$$

- *Rstream* (for "relation stream"):

$$\texttt{Rstream}(R) = \bigcup_{\tau \geq 0} (R(\tau) \times \{\tau\})$$

CQL: Syntactic Shortcuts and Defaults

- Default Windows (Unbounded)
- Default Relation-to-Stream Operators (Istream)

Example:

```
Select Istream(*)                                    Select *
From PosSpeedStr [Range Unbounded]                   From PosSpeedStr
Where speed > 65                                     Where speed > 65
```

Linear Road in CQL

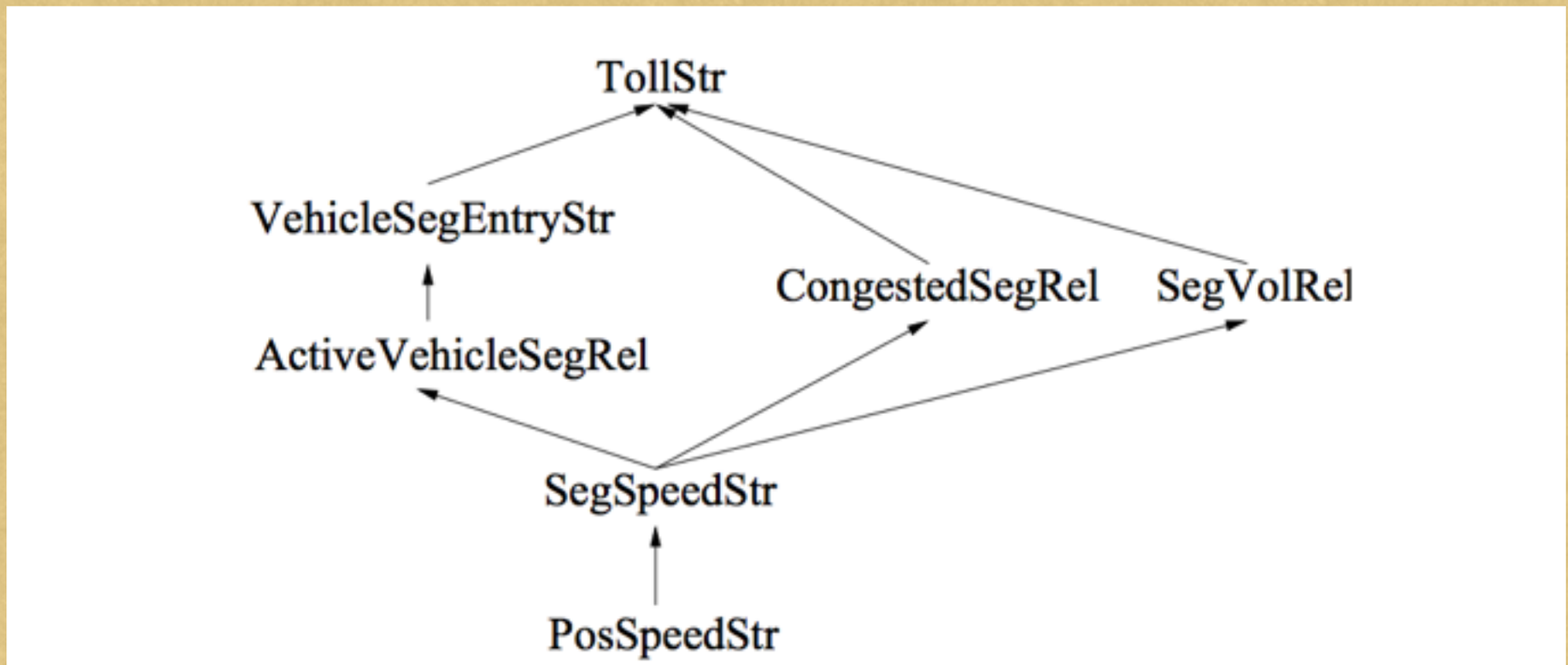


Figure 3: Derived relations and streams for Linear Road queries

The only input stream:

- **PosSpeedStr**(vehicleId, speed, xPos, dir, hwy)

The expected output stream:

- **TollStr**(vehicleId, toll)

Linear Road in CQL

- **SegSpeedStr**(vehicleId, speed, segNo, dir, hwy):
  Select vehicleId, speed, xPos/1760 as segNo, dir, why
  From PosSpeedStr
- **ActiveVehicleSegRel**(vehicleId, segNo, dir, hwy):
  Select Distinct L.vehicleId, L.segNo, L.dir, L.hwy
  From SegSpeedStr [Range 30 Seconds] as A,
  SegSpeedStr [Partition by vehicleId Rows 1] as L
  Where A.vehicleId = L.vehicleId
- **VehicleSegEntryStr**(vehicleId, segNo, dir, hwy):
  Select Istream(*) From ActiveVehicleSegRel
- **CongestedSegRel**(segNo, dir, hwy):
  Select segNo, dir, hwy
  From SegSpeedStr [Range 5 Minutes]
  Group By segNo, dir, hwy
  Having Avg(speed) < 40

Linear Road in CQL

- **SegVolRel**(segNo,dir,hwy,numVehicles):
  Select segNo, dir, hwy, count(vehicleId) as numVehicles
  From ActiveVehicleSegRel
  Group By segNo, dir, why

- **TollStr**(vehicleId, toll):
  Select Rstream(E.vehicleId,
  basetoll*(V.numVehicles-150)*(V.numVehicles-150) as toll)
  From VehicleSegEntryStr [Now] as E, CongestedSegRel as C,
  SegVolRel as V
  Where E.segNo = C.segNo and C.segNo = V.segNo and
  E.dir = C.dir and C.dir = V.dir and
  E.hwy = C.hwy and C.hwy = V.hwy

STREAM: A CQL IMPLEMENTATION

Goals for query execution plans

- Modularity and extensibility with general interface for operators and synopsis structures
- Efficient execution model that captures the combination of streams and relations
- Easy experimentation with different strategies for crucial system components, including operator scheduling, overflowing state to disk, sharing state and computation among multiple continuous queries, etc.

STREAM: Internal Representation

- Both streams and relations are represented as sequences of tagged tuples.
  - Sequences are append-only.
  - Sequences are always in nondecreasing order by timestamp.
- Operators are connected with queues.
  - A queue connects its input operator to its output operator
  - Synopses are used to store intermediate states of an operator
    - All tuple data are stored in synopses and is not duplicated
    - Some synopses simply point to data in other synopses.

STREAM: Query Plans

Example:

Q1: Select B, max(A)

From S1 [Rows 50,000]

Group By B

Q2: Select Istream(*)

From S1 [Rows 40,000],
S2 [Range 600 Seconds]

Where S1.A = S2.A

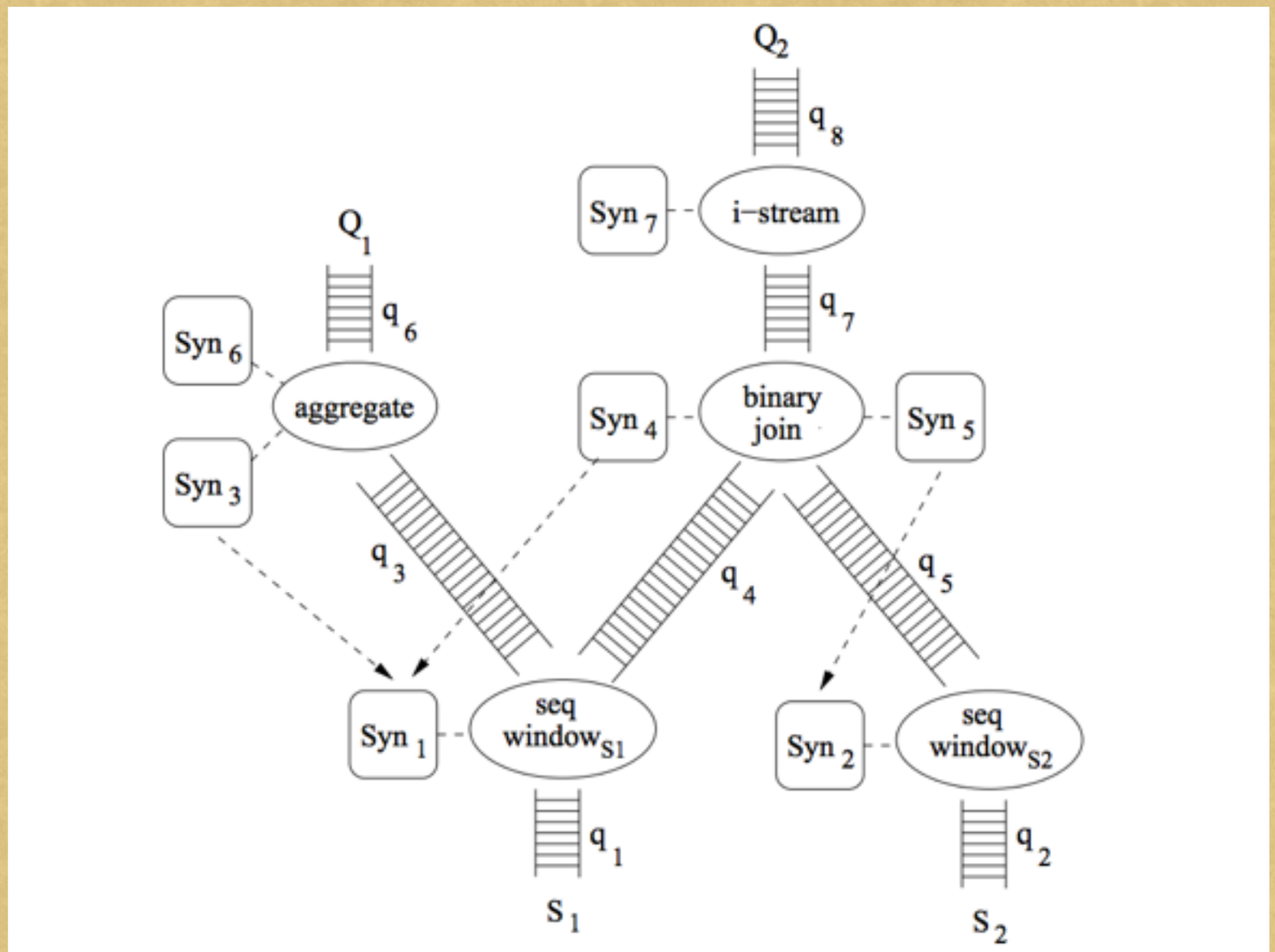


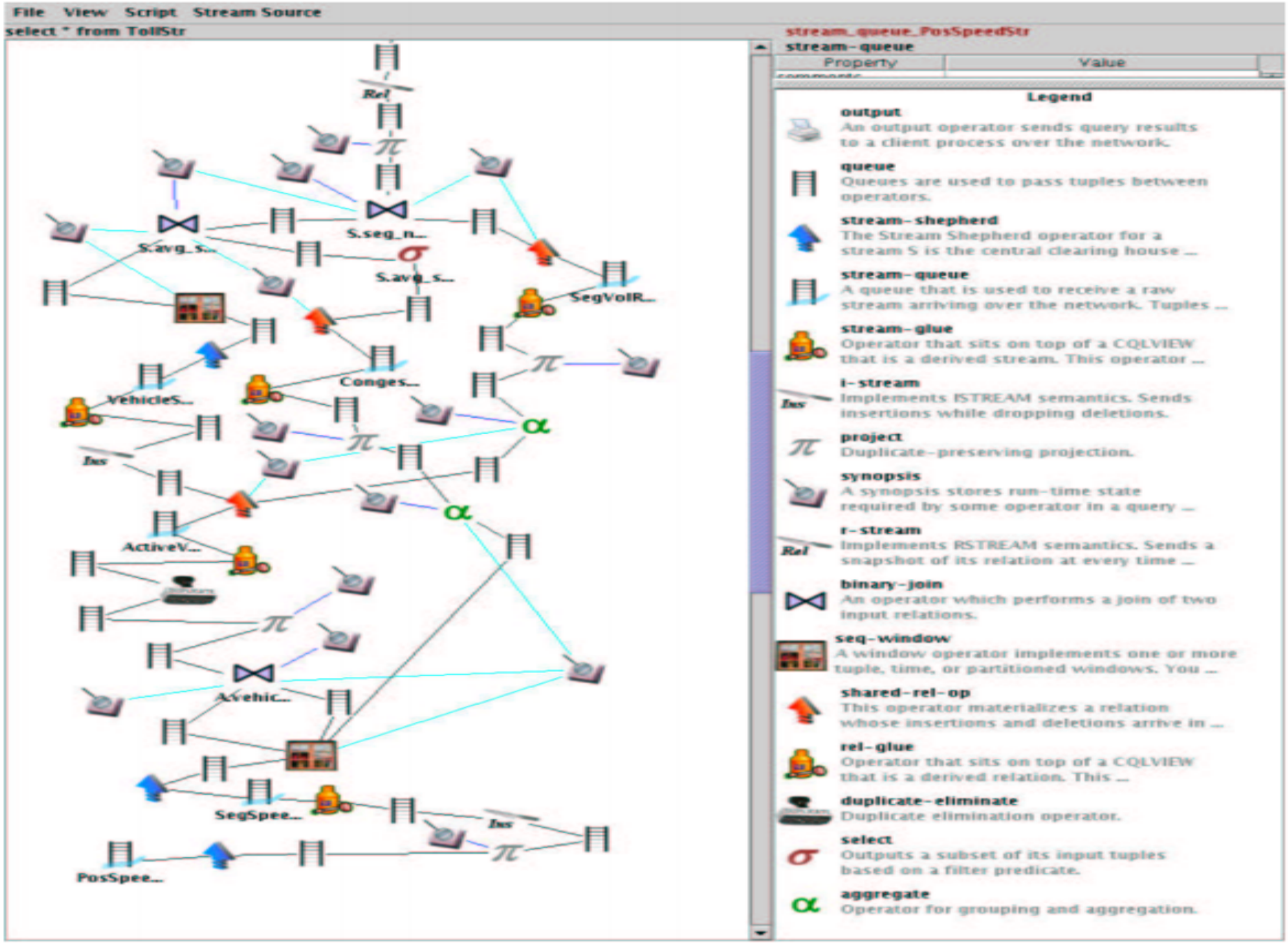
Figure 4: STREAM continuous query plan for two queries

STREAM: Query Operators

Name	Operator Type	Description
`seq-window`	stream-to-relation	Implements time-based, tuple-based, and partitioned windows
`select`	relation-to-relation	Filters tuples based on predicate(s)
`project`	relation-to-relation	Duplicate-preserving projection
`binary-join`	relation-to-relation	Joins two input relations
`mjoin`	relation-to-relation	Multiway join from [VNB03]
`union`	relation-to-relation	Bag union
`except`	relation-to-relation	Bag difference
`intersect`	relation-to-relation	Bag intersection
`antisemijoin`	relation-to-relation	Antisemijoin of two input relations
`aggregate`	relation-to-relation	Performs grouping and aggregation
`duplicate-eliminate`	relation-to-relation	Performs duplicate elimination
`i-stream`	relation-to-stream	Implements `Istream` semantics
`d-stream`	relation-to-stream	Implements `Dstream` semantics
`r-stream`	relation-to-stream	Implements `Rstream` semantics
`stream-shepherd`	system operator	Handles input streams arriving over the network
`stream-sample`	system operator	Samples specified fraction of tuples
`stream-glue`	system operator	Adapter for merging a stream-producing view into a plan
`rel-glue`	system operator	Adapter for merging a relation-producing view into a plan
`shared-rel-op`	system operator	Materializes a relation for sharing
`output`	system operator	Sends results to remote clients

Table 1: Operators in STREAM query plans

STREAM: Query Plans



Query Optimization and Future Works

Hard-coded heuristics:

- Push selections below joins.
- Use indexes for synopses on join and aggregate operators.
- Share synopses within query plans whenever possible.

techniques

Future Works:

- one-time and dynamic cost-based optimization of CQL queries
  - leverage techniques on tradition relational systems
- coarser-grained adaptive query optimization techniques
  - Monitor streams and system behavior
  - reconfigure query plans and reconfigure query plans over time

Thank you!

Questions?

Reference
1. http://www.edshare.soton.ac.uk/14234/1/15_-_Data_Streams.pptx
2. http://ilpubs.stanford.edu:8090/758/1/2003-67.pdf
3. http://www.desertislesql.com/wordpress1/wp-content/uploads/2015/11/SlidingWindows.jpg