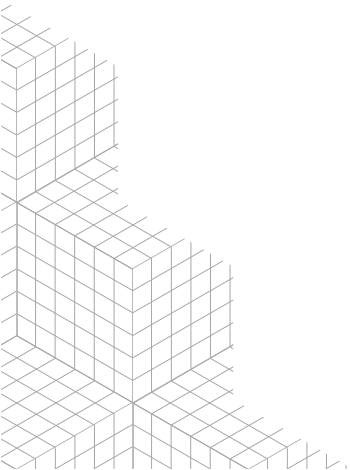
OBDA on Temporal and Streaming Data

Özgür L. Özçep, Ralf Möller
(TUHH)

RW2014, Athens



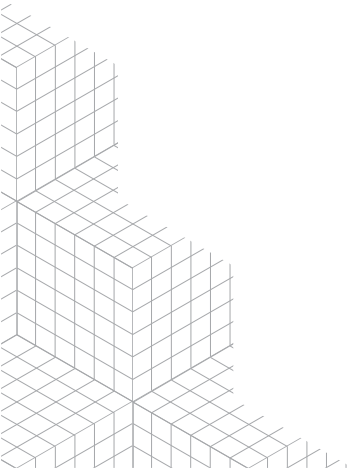
Optique

Tutorial Overview

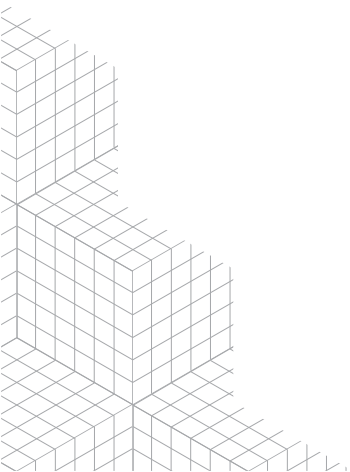
I Ontology-Based Data Access

II Temporalizing and Streamifying OBDA

III STARQL

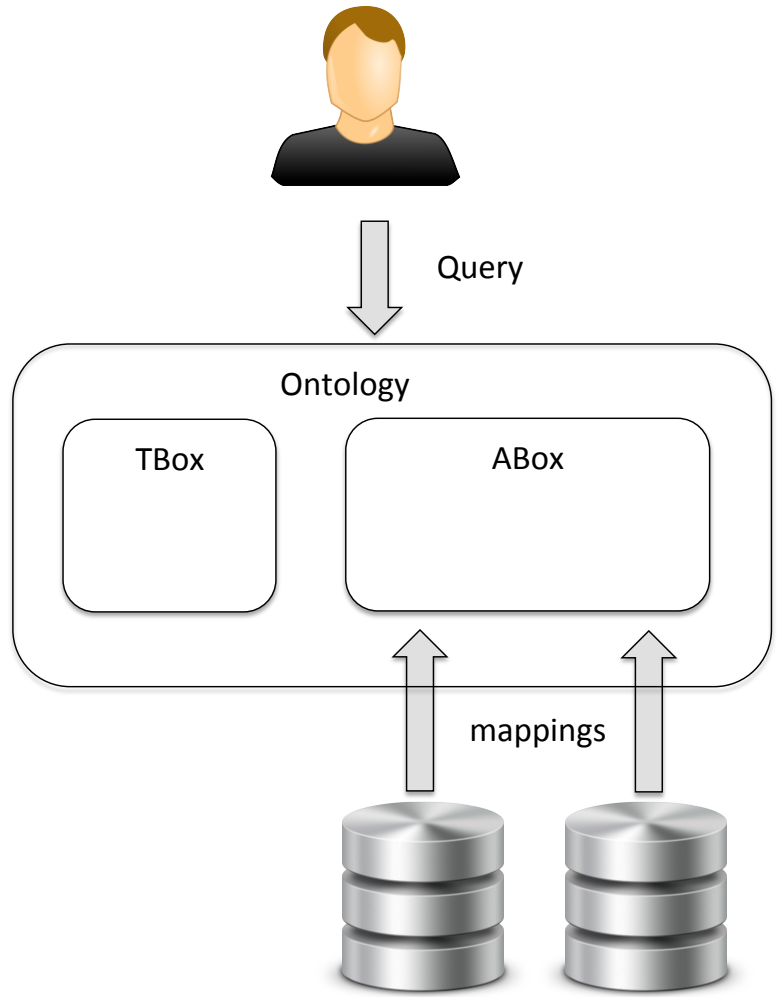


PART I: OBDA



Ontology-Based Data Access

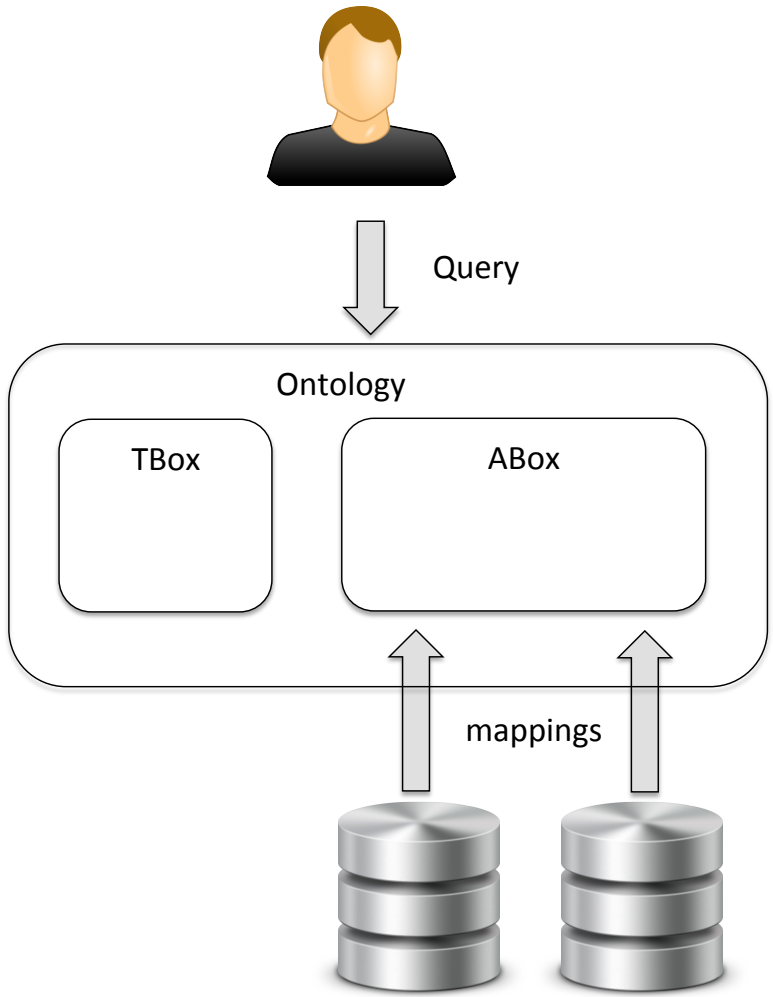
- Use ontologies as interface ...
- to access (here: query)
- data stored in some format ...
- using mappings



Ontology-Based Data Access

Optique

- Use ontologies as interface ...
- to access (here: query)
- data stored in some format ...
- using mappings



Ontologies

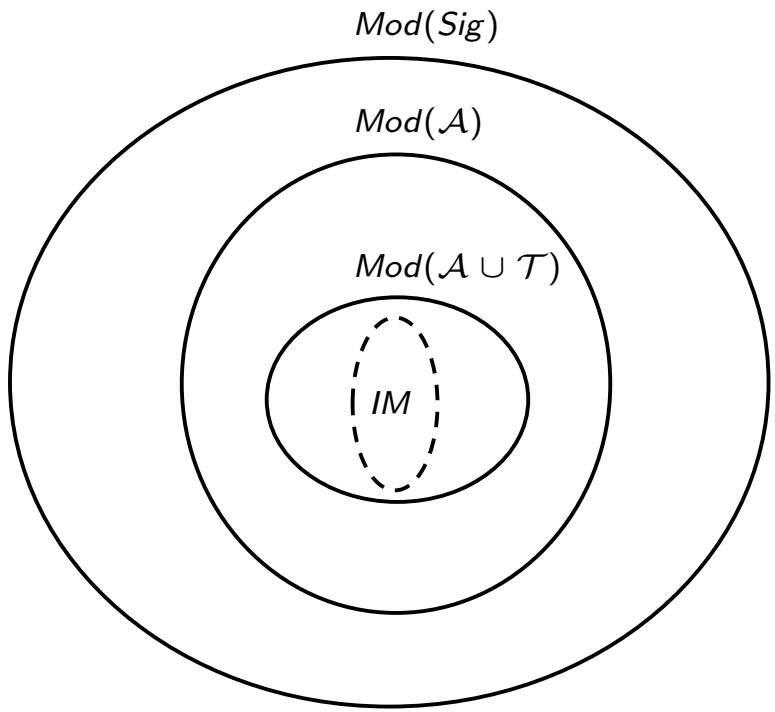
- Ontologies are structures of the form $\mathcal{O} = (Sig, \mathcal{T}, \mathcal{A})$
  - Signature: Non-logical vocabulary
    $Sig = Const_{Sig} \cup Conc_{Sig} \cup Role_{Sig}$
  - TBox $\mathcal{T}$: set of Sig-axioms in some DL logic to capture terminological knowledge
  - ABox $\mathcal{A}$: set of Sig-axioms in (same DL logic) to capture assertional/contingential knowledge
- Note: Sometimes only TBox termed ontology
- Semantics defined on the basis of Sig-interpretations $\mathcal{I}$
  - $\mathcal{I} \models Ax$ iff $\mathcal{I}$ makes all axioms in Ax true
  - $Mod(Ax) = \{\mathcal{I} \models Ax\}$

Ontologies

- Ontologies are structures of the form $\mathcal{O} = (Sig, \mathcal{T}, \mathcal{A})$
  - Signature: Non-logical vocabulary
    $Sig = Const_{Sig} \cup Conc_{Sig} \cup Role_{Sig}$
  - TBox $\mathcal{T}$: set of Sig-axioms in some DL logic to capture terminological knowledge
  - ABox $\mathcal{A}$: set of Sig-axioms in (same DL logic) to capture assertional/contingential knowledge
- Note: Sometimes only TBox termed ontology
- Semantics defined on the basis of Sig-interpretations $\mathcal{I}$
  - $\mathcal{I} \models Ax$ iff $\mathcal{I}$ makes all axioms in Ax true
  - $Mod(Ax) = \{\mathcal{I} \models Ax\}$

General Idea

- $\mathcal{A}$: Represents facts in domain of interest
- Open world assumption: $Mod(\mathcal{A})$ is not a singleton
- $\mathcal{T}$: Constrains $Mod(\mathcal{A})$ with intended Sig readings
- In most cases one has only approximations of intended models *IM*
- Realize inference services on the basis of the constrained interpretations



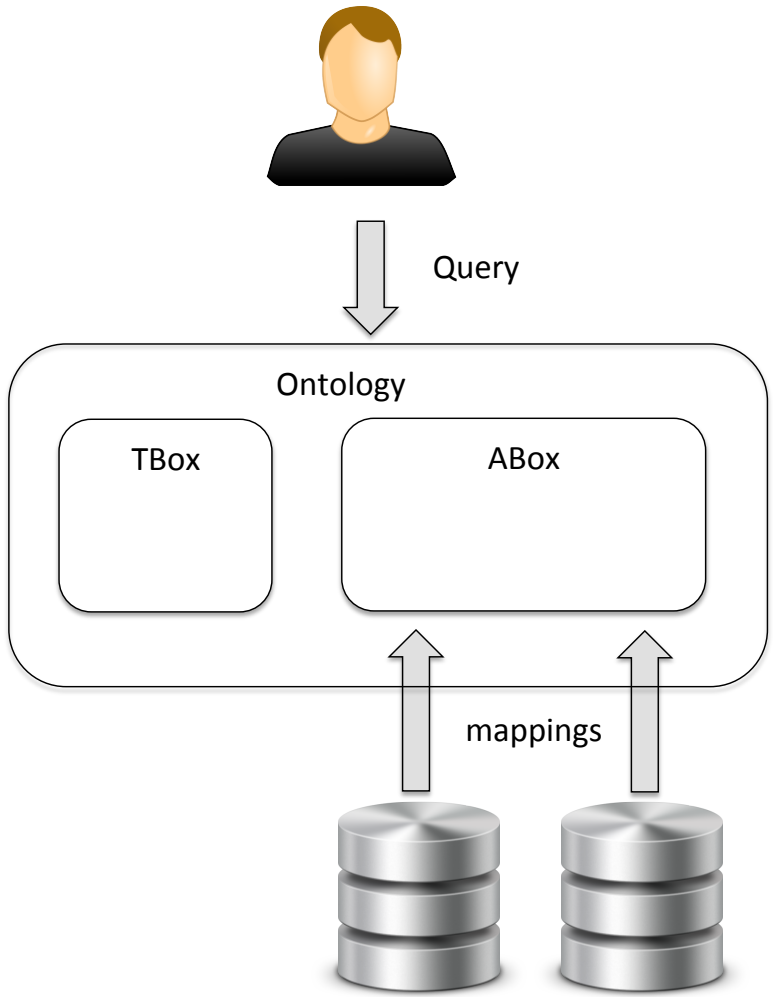
WARNING: A Misconception

- With ontologies one does not declare data structures
- ABox data in most cases show pattern of data structures
- One does not have to re-model patterns/constraints in the ABox data
  - Knowing "All A are B" in the ABox is different from stipulating $A \sqsubseteq B$
  - Add $A \sqsubseteq B$, if you need to handle this relation for objects not mentioned in the ABox
- Motto: Keep the TBox simple
- In so far: Ontology bootstrapping on backend data sources to be handled with care

Ontology-Based Data Access

Optique

- Use ontologies as interface ...
- to access (here: query)
- data stored in some format ...
- using mappings



Reasoning Services

- Different standard and nonstandard reasoning services exists
- May be reducible to each other
- Examples: consistency check, subsumption check, taxonomy calculations, ... most specific subsumer, most specific concept, matching, ...
- Focus on
  - Consistency checking: $Mod(\mathcal{A} \cup \mathcal{T}) \neq \emptyset$.
  - Query answering
- Next to ABox and TBox language query language QL over Sig is a relevant factor for OBDA
- Queries are formulas $\psi(\vec{x})$ in QL with open variables $\vec{x} = (x_1, \ldots, x_n)$ (distinguished variables).
- Certain query answering

$$cert(\psi(\vec{x}), \mathcal{T} \cup \mathcal{A}) \quad = \quad \{\vec{a} \in (Const_{Sig})^n \mid \mathcal{T} \cup \mathcal{A} \models \psi[\vec{x}/\vec{a}]\}$$

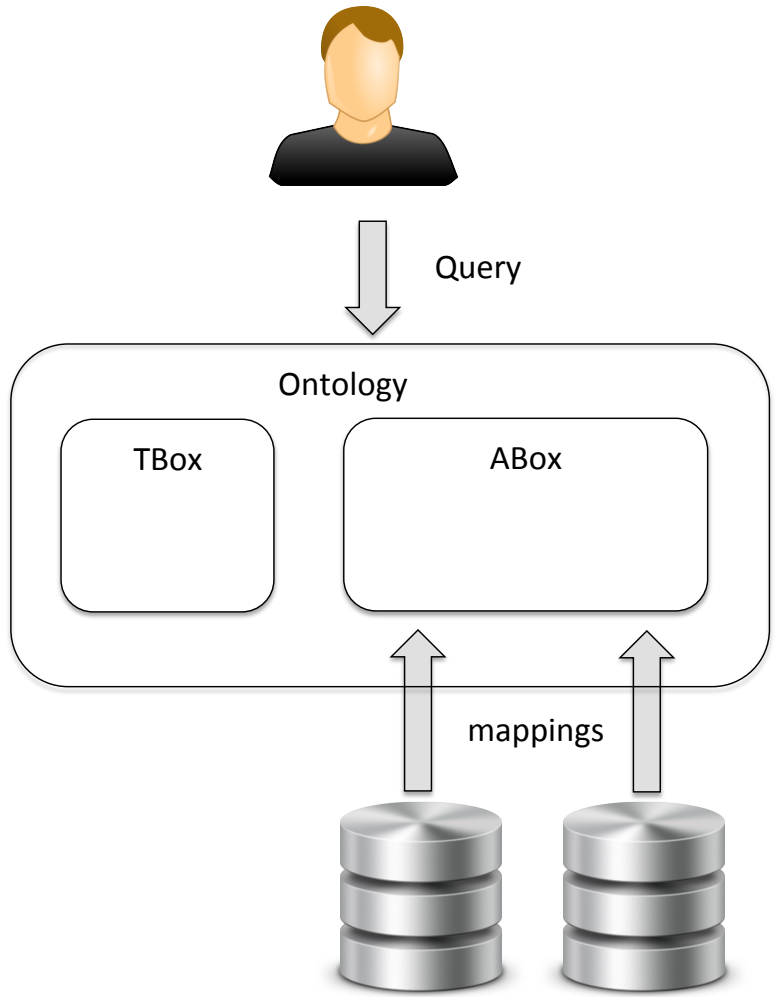
Reasoning Services

- Different standard and nonstandard reasoning services exists
- May be reducible to each other
- Examples: consistency check, subsumption check, taxonomy calculations, ... most specific subsumer, most specific concept, matching, ...
- Focus on
  - Consistency checking: $Mod(\mathcal{A} \cup \mathcal{T}) \neq \emptyset$.
  - Query answering
- Next to ABox and TBox language query language QL over Sig is a relevant factor for OBDA
- Queries are formulas $\psi(\vec{x})$ in QL with open variables $\vec{x} = (x_1, \ldots, x_n)$ (distinguished variables).
- Certain query answering

$$cert(\psi(\vec{x}), \mathcal{T} \cup \mathcal{A}) \quad = \quad \{\vec{a} \in (Const_{Sig})^n \mid \mathcal{T} \cup \mathcal{A} \models \psi[\vec{x}/\vec{a}]\}$$

Ontology-Based Data Access

- Use ontologies as interface ...
- to access (here: query)
- data stored in some format ...
- using mappings

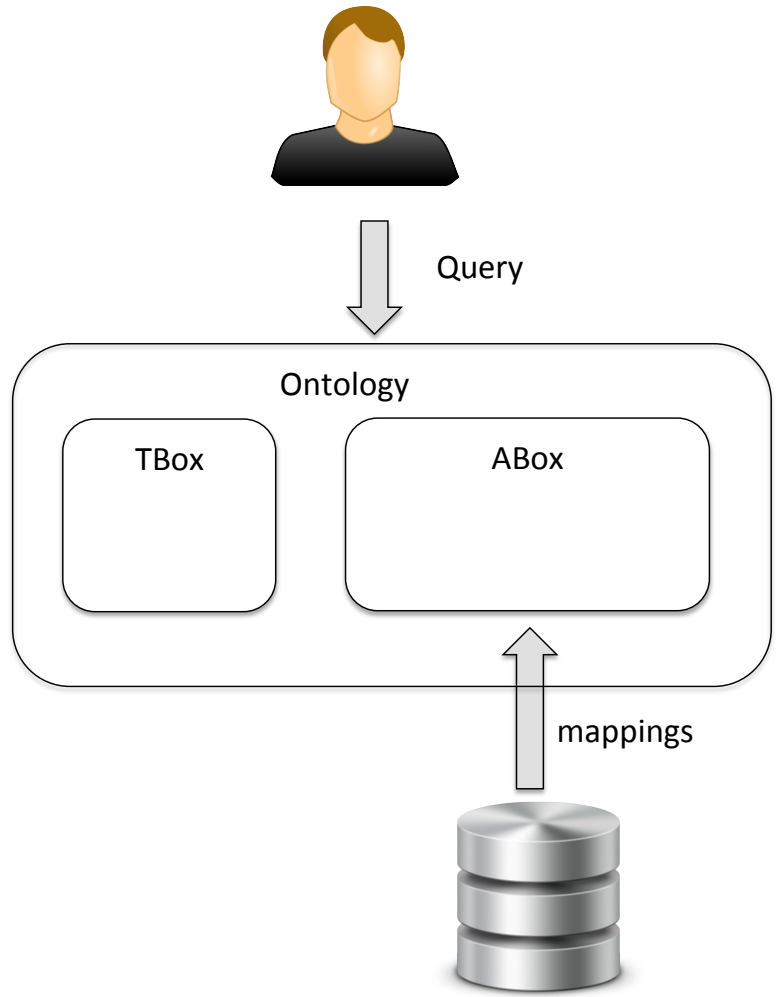


Backend Data Sources

- Classically: relational SQL DBs with static data
- Possible extensions: non-SQL DBs
  - datawarehouse repositories for statistical applications
  - pure logfiles
  - RDF repositories
- Non-static data
  - historical data (stored in temporal DB)
  - dynamic data coming in streams
- Originally intended for multiple DBs but ...

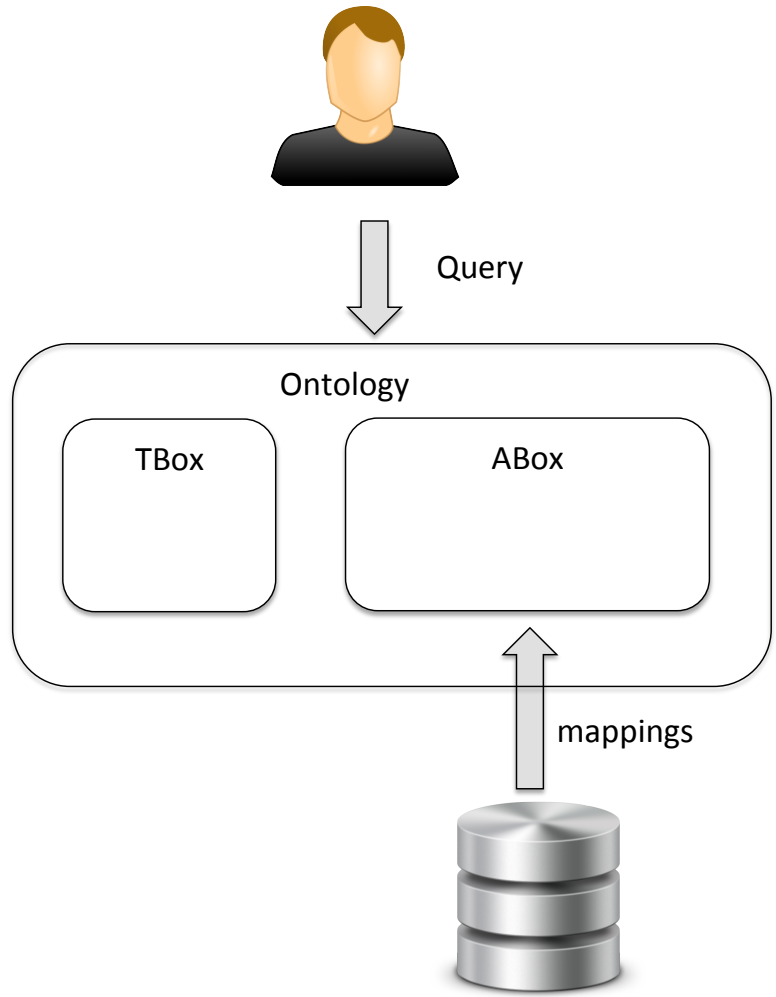
Federation

- ... we would have to deal with federation
- not trivial in classical OBDA ...
- because one has to integrate data from different DBs
- Ignore federation aspect: we have one DB but possibly many tables



Ontology-Based Data Access

- Use ontologies as interface ...
- to access (here: query)
- data stored in some format ...
- using mappings



Mappings

- Mappings have an important crucial role in OBDA
- Lift data to the ontology level
  - Data level: (nearly) close world
  - Ontology Level: open world

Schema of Mappings

$$m : \psi(\vec{f}(\vec{x})) \longleftarrow Q(\vec{x}, \vec{y})$$

- $\psi(\vec{f}(\vec{x}))$: Template (query) for generating ABox axioms
- $Q(\vec{x}, \vec{y})$: Query over the backend sources
- Function $\vec{f}$ translates backend instantiations of $\vec{x}$ to constants

- Mappings M over backend sources generates ABox $\mathcal{A}(M, DB)$.

Example Scenario: Measurements

Optique

- Example schema for measurement and event data in DB

```
SENSOR(SID, CID, Sname, TID, description)
SENSORTYPE(TID, Tname)
COMPONENT(CID, superCID, AID, Cname)
ASSEMBLY(AID, AName, ALocation)
MEASUREMENT(MID, MtimeStamp, SID, Mval)
MESSAGE(MesID, MesTimeStamp, MesAssemblyID,  catID, MesEventText)
CATEGORY(catID, catName)
```

- For mapping

  m: $Sens(x) \wedge name(x, y) \longleftarrow$

```
SELECT f(SID) as x,   Sname as y FROM SENSOR
```

- the row data in SENSOR table

SENSOR
(123, comp45, TC255, TempSens, 'A temperature sensor')

- generates facts

$Sens(f(123)), name(f(123), TempSens) \in \mathcal{A}(m, DB)$

Example Scenario: Measurements

- Example schema for measurement and event data in DB

```
SENSOR(SID, CID, Sname, TID, description)
SENSORTYPE(TID, Tname)
COMPONENT(CID, superCID, AID, Cname)
ASSEMBLY(AID, AName, ALocation)
MEASUREMENT(MID, MtimeStamp, SID, Mval)
MESSAGE(MesID, MesTimeStamp, MesAssemblyID,  catID, MesEventText)
CATEGORY(catID, catName)
```

- For mapping

  m: $Sens(x) \wedge name(x, y) \longleftarrow$

```
SELECT f(SID) as x,   Sname as y FROM SENSOR
```

- the row data in SENSOR table

```
SENSOR
(123, comp45, TC255, TempSens, 'A temperature sensor')
```

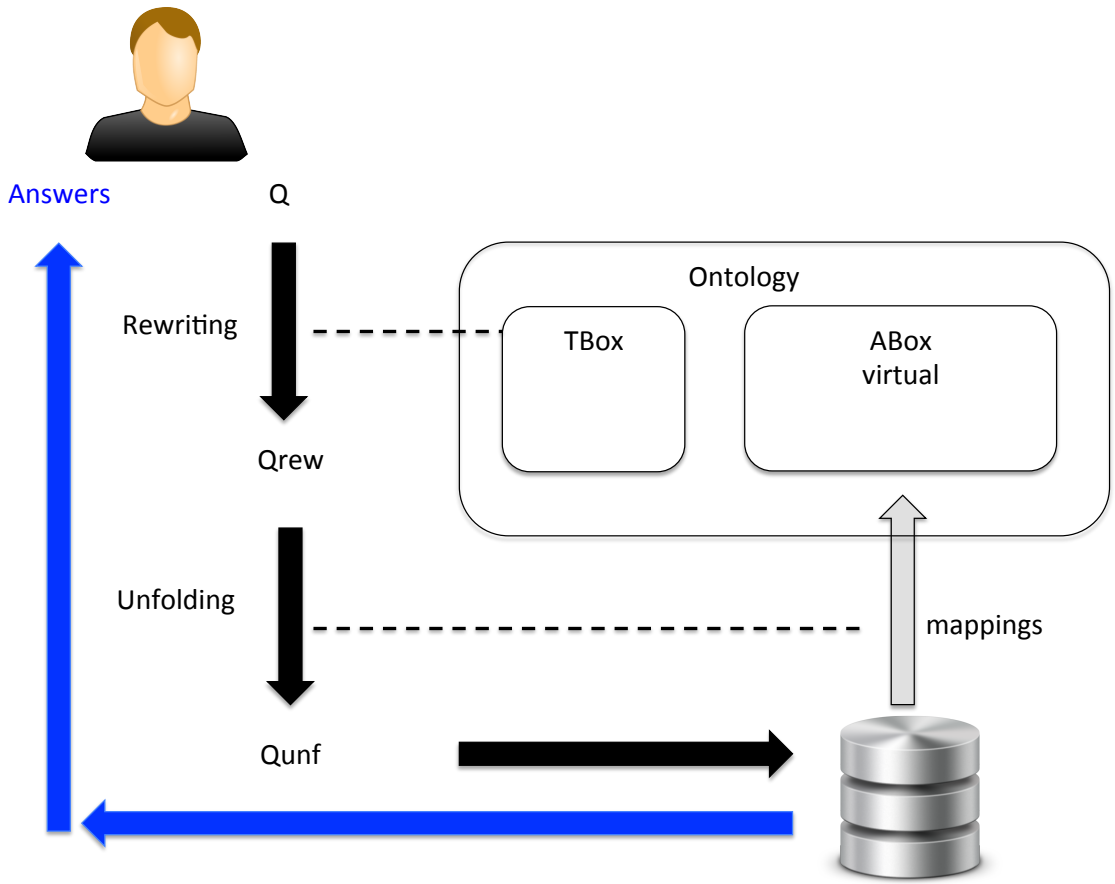
- generates facts

$Sens(f(123)), name(f(123), TempSens) \in \mathcal{A}(m, DB)$

OBDA in the Classical Sense

Optique

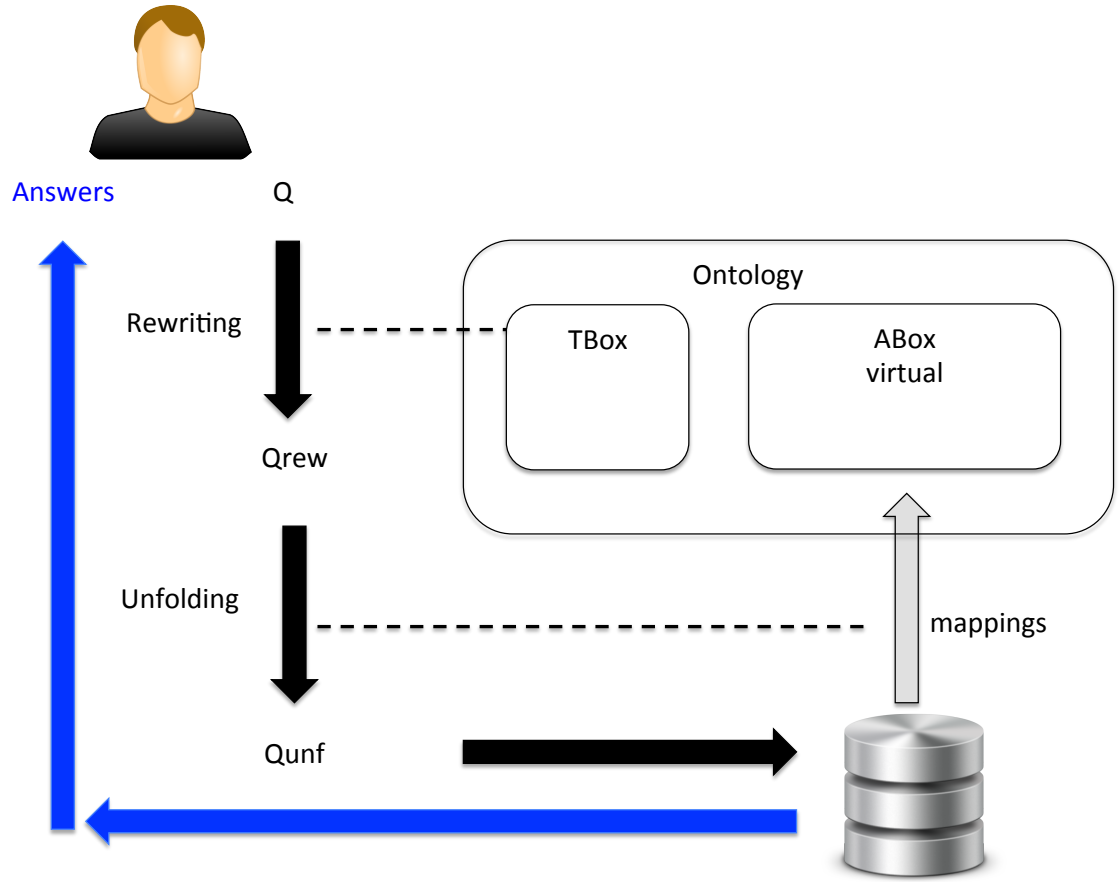
- Keep the data where they are because of large volume
- ABox is virtual (no materialization)



OBDA in the Classical Sense

Optique

- First-order logic (FOL) perfect rewriting + unfolding for realizing reasoning services



OBDA in the Classical Sense

- $\mathcal{T}$ language: Some logic of the DL-Lite family
- $\mathcal{A}$ language: assertions of the form $A(a), R(a, b)$
- QL : Unions of conjuctive queries (UCQs)
- Language of $Qrew$: safe FOL
- Allows for perfect rewriting (of consistency checking and) UCQ answering

$$cert(Q, (Sig, \mathcal{T}, \mathcal{A})) = cert(Qrew, \mathcal{A}) = ans(Qrew, DB(\mathcal{A}))$$

- and unfolding

$$cert(Q, (Sig, \mathcal{T}, \mathcal{A}(M, DB))) = ans(Qunf, DB)$$

- Note that query language over DB is relevant for possibility of unfolding

OBDA in the Classical Sense

Optique

- $\mathcal{T}$ language: Some logic of the DL-Lite family
- $\mathcal{A}$ language: assertions of the form $A(a), R(a, b)$
- QL : Unions of conjuctive queries (UCQs)
- Language of $Qrew$: safe FOL
- Allows for perfect rewriting (of consistency checking and) UCQ answering

$$cert(Q, (Sig, \mathcal{T}, \mathcal{A})) = cert(Qrew, \mathcal{A}) = ans(Qrew, DB(\mathcal{A}))$$

- and unfolding

$$cert(Q, (Sig, \mathcal{T}, \mathcal{A}(M, DB))) = ans(Qunf, DB)$$

- Note that query language over DB is relevant for possibility of unfolding

OBDA in the Classical Sense

- $\mathcal{T}$ language: Some logic of the DL-Lite family
- $\mathcal{A}$ language: assertions of the form $A(a), R(a, b)$
- QL : Unions of conjuctive queries (UCQs)
- Language of $Qrew$: safe FOL
- Allows for perfect rewriting (of consistency checking and) UCQ answering

$$cert(Q, (Sig, \mathcal{T}, \mathcal{A})) = cert(Qrew, \mathcal{A}) = ans(Qrew, DB(\mathcal{A}))$$

- and unfolding

$$cert(Q, (Sig, \mathcal{T}, \mathcal{A}(M, DB))) = ans(Qunf, DB)$$

- Note that query language over DB is relevant for possibility of unfolding

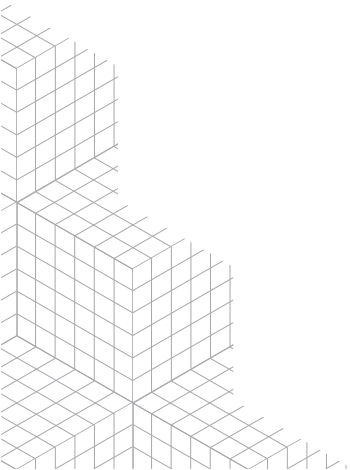
A Critical View on DL-Lite

- Rewriting may lead to exponential blow up of queries
  - but argue that this does not occur in practical cases
  - and use optimizations by looking at the ABox
- TBox language weak
  - Very strict use of functional roles
  - One has only qualified existentials on the right of GCIs
    $\Longrightarrow$ no expressive sufficient conditions
- UCQs not that expressive
  - But may be useful if UCQs are generated automatically

Extended OBDA

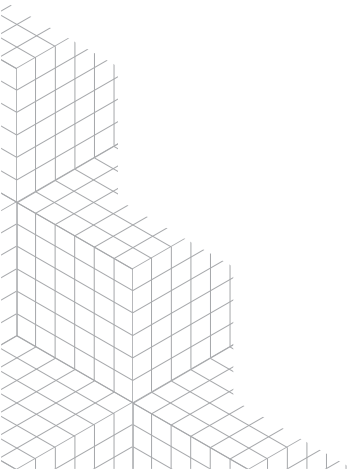
- Use more expressive TBox language
  - ABDEO (Accessing very big data using expressive ontologies)
  - Rewritability for UCQs not guaranteed
  - Materialize ABox and use ABox modularization to answer queries
- Use different (more expressive) QL
  - E.g. SPARQL instead of UCQ; but no full existentials in combination with DL-Lite
  - OWL2QL + SPARQL used in Optique platform
- Use different reasoning/rewriting paradigm
  - e.g. combined rewriting: First enhance ABox with TBox information and then rewrite
  - Streaming

PART II: Temporalized and Streamified OBDA



A Confession

- Ontology-Based Data Access on temporal and **Streaming** Data
- But: Streams are temporal streams and we talk about local temporal reasoning



Adding a Temporal Dimension to OBDA Optique

- Most conservative strategy: handle time as "ordinary" attribute *time*

$$\left\{ \begin{array}{l} meas(x) \wedge \\ val(x, y) \wedge \\ time(x, z) \end{array} \right\} \longleftarrow$$

```
SELECT f(MID) AS m, Mval AS y, MtimeStamp AS z
FROM MEASUREMENT
```

- Classical Mapping
- Pro: Minimal (no) adaptation
- Contra:
  - No control on "logic of time"
  - Need reification
    - sometimes necessary (because DLs provided only predicates up to arity 2)
    - but not that "natural"

Flow of Time

- Flow of time $(T, \leq_T)$ is a structure with a time domain T and a binary relation $\leq_T$ over it.
- Take points as primitives
- Different properties to describe different temporal aspects
- We consider non-branching (or: linear) time, i.e., $\leq_T$ is
  - reflexive: $\forall t \in T\colon\ t \leq_T t$
  - antisymmetric: $\forall t_1, t_2 \in T\colon\ (t_1 \leq t_2 \wedge t_2 \leq_T t_1) \Rightarrow t_1 = t_2$
  - transitive: $\forall t_1, t_2, t_3 \in T : (t_1 \leq_T t_2 \wedge t_2 \leq t_3) \Rightarrow t_1 \leq t_3$.
  - total: $\forall t_1, t_2 \in T\colon\ t_1 \leq t_2 \vee t_2 \leq t_1 \vee t_1 = t_2$.
- Existence of first or last element
- discreteness (Example: $T = \mathbb{N}$); also used for modeling state sequences;
- density (Example: $T = \mathbb{Q}$);
- continuity (Example: $T = \mathbb{R}$)

Flow of Time

- Flow of time $(T, \leq_T)$ is a structure with a time domain T and a binary relation $\leq_T$ over it.
- Take points as primitives
- Different properties to describe different temporal aspects
- We consider non-branching (or: linear) time, i.e., $\leq_T$ is
  - reflexive: $\forall t \in T\colon\ t \leq_T t$
  - antisymmetric: $\forall t_1, t_2 \in T\colon\ (t_1 \leq t_2 \wedge t_2 \leq_T t_1) \Rightarrow t_1 = t_2$
  - transitive: $\forall t_1, t_2, t_3 \in T : (t_1 \leq_T t_2 \wedge t_2 \leq t_3) \Rightarrow t_1 \leq t_3$.
  - total: $\forall t_1, t_2 \in T\colon\ t_1 \leq t_2 \vee t_2 \leq t_1 \vee t_1 = t_2$.
- Existence of first or last element
- discreteness (Example: $T = \mathbb{N}$); also used for modeling state sequences;
- density (Example: $T = \mathbb{Q}$);
- continuity (Example: $T = \mathbb{R}$)

Flow of Time

- Flow of time $(T, \leq_T)$ is a structure with a time domain T and a binary relation $\leq_T$ over it.
- Take points as primitives
- Different properties to describe different temporal aspects
- We consider non-branching (or: linear) time, i.e., $\leq_T$ is
  - reflexive: $\forall t \in T\colon\ t \leq_T t$
  - antisymmetric: $\forall t_1, t_2 \in T\colon\ (t_1 \leq t_2 \wedge t_2 \leq_T t_1) \Rightarrow t_1 = t_2$
  - transitive: $\forall t_1, t_2, t_3 \in T : (t_1 \leq_T t_2 \wedge t_2 \leq t_3) \Rightarrow t_1 \leq t_3$.
  - total: $\forall t_1, t_2 \in T\colon\ t_1 \leq t_2 \vee t_2 \leq t_1 \vee t_1 = t_2$.
- Existence of first or last element
- discreteness (Example: $T = \mathbb{N}$); also used for modeling state sequences;
- density (Example: $T = \mathbb{Q}$);
- continuity (Example: $T = \mathbb{R}$)

Flow of Time

- Flow of time $(T, \leq_T)$ is a structure with a time domain T and a binary relation $\leq_T$ over it.
- Take points as primitives
- Different properties to describe different temporal aspects
- We consider non-branching (or: linear) time, i.e., $\leq_T$ is
  - reflexive: $\forall t \in T\colon\ t \leq_T t$
  - antisymmetric: $\forall t_1, t_2 \in T\colon\ (t_1 \leq t_2 \wedge t_2 \leq_T t_1) \Rightarrow t_1 = t_2$
  - transitive: $\forall t_1, t_2, t_3 \in T : (t_1 \leq_T t_2 \wedge t_2 \leq t_3) \Rightarrow t_1 \leq t_3$.
  - total: $\forall t_1, t_2 \in T\colon\ t_1 \leq t_2 \vee t_2 \leq t_1 \vee t_1 = t_2$.
- Existence of first or last element
- discreteness (Example: $T = \mathbb{N}$); also used for modeling state sequences;
- density (Example: $T = \mathbb{Q}$);
- continuity (Example: $T = \mathbb{R}$)

Temporalized OBDA

- Semantics rests on family of interpretations $(\mathcal{I}_t)_{t \in T}$
- Temporal ABox $\tilde{A}$: Finite set of T-tagged ABox axioms
- $val(s_0, 90°)\langle 3s \rangle$ holds in $(\mathcal{I}_t)_{t \in T}$ iff $\mathcal{I}_{3s} \models val(s_0, 90°)$
  "sensor s_0 has value 90° at time point 3s"
- Sequence representation of temporal ABox $\tilde{\mathcal{A}}$
  - $(\mathcal{A}_t)_{t \in T'}$ (where T' are set of timestamps in T)
  - $\mathcal{A}_t = \{ax \mid ax\langle t \rangle \in \tilde{\mathcal{A}}\}$
- Rewriting w.r.t. temporally extended DLs

$$cert(Q, (Sig, \mathcal{T}, (\mathcal{A}_t)_{t \in T'}) = ans(Q_{rew}, (DB(\mathcal{A}_t))_{t \in T'})$$

Temporalized OBDA

- Different approaches based on modal (temporal) operators
- LTL (linear temporal logic) operators only in QL (Borgwardt et al. 13)

$$\begin{aligned} \mathit{Critical}(x) &= \exists y.\, \mathit{Turbine}(x) \wedge \mathit{showsMessage}(x, y) \wedge \\ &\quad \mathit{FailureMessage}(y) \\ Q(x) &= \bigcirc^{-1} \bigcirc^{-1} \bigcirc^{-1} (\Diamond(\mathit{Critical}(x) \wedge \bigcirc \Diamond \mathit{Critical}(x))) \end{aligned}$$

"turbine has been at least two times in a critical situation in the last three time units"

- CQ embedded into LTL template
- Special operators taking care of endpoints of state sequencing
- Non-safe
- Rewriting simple due to atemporal TBox

Temporalized OBDA

- LTL operators in TBox and T argument in QL (Artale et al. 13)

  TBox axiom : $\mathit{showsAnomaly} \sqsubseteq \Diamond \mathit{UnplanedShutDown}$

  "if turbine shows anomaly (now)

  then sometime in the future it will shut down"

  Query : $\exists t.3s \leq t \leq 6s \wedge \mathit{showsAnomaly}(x, t)$

- Can formulate rigidity assumptions
- Rewriting not trivial
- Both approaches do not deal with unfolding

Adding a Stream Dimension to OBDA

Definition (Temporal Stream)

A stream S is a sequence of timestamped objects $d\langle t\rangle$ over some domain D and flow of time $(T, \leq_T)$.

- Consider non-branching (or: linear) time, i.e., $\leq_T$ is
- We assume that there is no last element in T
- We do not restrict T further, so it may be
  - discrete or
  - dense or
  - continuous

Arrival Ordering

Definition (Temporal Stream)

A stream S is a sequence of timestamped objects $d\langle t\rangle$ over some domain D and flow of time $(T, \leq_T)$.

- Sequence fixed by arrival ordering fixed $<_{ar}$
- $<_{ar}$ is a strict total ordering on the elements of S
- Synchronuous streams: $\leq_T$ compatible with $<_{ar}$
- Compatibility: For all $d_1\langle t_1\rangle, d_2\langle t_2\rangle \in S$: If $d_1\langle t_1\rangle <_{ar} d_2\langle t_2\rangle$, then $t_1 \leq_T t_2$.
- Asynchronous streams: $\leq_T$ not necessarily compatible with $<_{ar}$
- Convention for the following
  - Consider only temporal streams
  - Consider only synchronous streams $\Longrightarrow$ neglect $<_{ar}$.
  - Represent streams as a potentially infinite multi-set (bag) of elements

Arrival Ordering

Definition (Temporal Stream)

A stream S is a sequence of timestamped objects $d\langle t\rangle$ over some domain D and flow of time $(T, \leq_T)$.

- Sequence fixed by arrival ordering fixed $<_{ar}$
- $<_{ar}$ is a strict total ordering on the elements of S
- Synchronuous streams: $\leq_T$ compatible with $<_{ar}$
- Compatibility: For all $d_1\langle t_1\rangle, d_2\langle t_2\rangle \in S$: If $d_1\langle t_1\rangle <_{ar} d_2\langle t_2\rangle$, then $t_1 \leq_T t_2$.
- Asynchronous streams: $\leq_T$ not necessarily compatible with $<_{ar}$
- Convention for the following
  - Consider only temporal streams
  - Consider only synchronous streams $\Longrightarrow$ neglect $<_{ar}$.
  - Represent streams as a potentially infinite multi-set (bag) of elements

Arrival Ordering

Definition (Temporal Stream)

A stream S is a sequence of timestamped objects $d\langle t\rangle$ over some domain D and flow of time $(T, \leq_T)$.

- Sequence fixed by arrival ordering fixed $<_{ar}$
- $<_{ar}$ is a strict total ordering on the elements of S
- Synchronuous streams: $\leq_T$ compatible with $<_{ar}$
- Compatibility: For all $d_1\langle t_1\rangle, d_2\langle t_2\rangle \in S$: If $d_1\langle t_1\rangle <_{ar} d_2\langle t_2\rangle$, then $t_1 \leq_T t_2$.
- Asynchronous streams: $\leq_T$ not necessarily compatible with $<_{ar}$
- Convention for the following
  - Consider only temporal streams
  - Consider only synchronous streams $\Longrightarrow$ neglect $<_{ar}$.
  - Represent streams as a potentially infinite multi-set (bag) of elements

Domain Objects for Streams

Definition (Temporal Stream)

A stream S is a sequence of timestamped objects (d, t) over some domain D and flow of time $(T, \leq_T)$.

In streamified OBDA we have to deal with different types of domains

- D is a set of typed tuples adhering to a relational schema
  - Streams at the backend sources
  - $S_{rel} = \{(s_1, 90°)\langle 1s\rangle, (s_2, 92°)\langle 2s\rangle, (s_1, 94°)\langle 3s\rangle, \dots\}$
  - Schema: hasSensorRelation(Sensor:string, temperature:integer)
- D is a set of untyped tuples (of the same arity)
  - Stream of tuples resulting as bindings for subqueries
- D is a set of assertions (RDF tuples).
  - $S_{rdf} = \{\ val(s_0, 90°)\langle 1s\rangle, val(s_2, 92°)\langle 2s\rangle,\ val(s_1, 94°)\langle 3s\rangle, \dots\ \}$
- D is a set of RDF graphs

Domain Objects for Streams

Definition (Temporal Stream)

A stream S is a sequence of timestamped objects (d, t) over some domain D and flow of time $(T, \leq_T)$.

In streamified OBDA we have to deal with different types of domains

- D is a set of typed tuples adhering to a relational schema
  - Streams at the backend sources
  - $S_{rel} = \{(s_1, 90°)\langle 1s\rangle, (s_2, 92°)\langle 2s\rangle, (s_1, 94°)\langle 3s\rangle, \dots\}$
  - Schema: hasSensorRelation(Sensor:string, temperature:integer)
- D is a set of untyped tuples (of the same arity)
  - Stream of tuples resulting as bindings for subqueries
- D is a set of assertions (RDF tuples).
  - $S_{rdf} = \{\ val(s_0, 90°)\langle 1s\rangle, val(s_2, 92°)\langle 2s\rangle,\ val(s_1, 94°)\langle 3s\rangle, \dots\ \}$
- D is a set of RDF graphs

Domain Objects for Streams

Definition (Temporal Stream)

A stream S is a sequence of timestamped objects (d, t) over some domain D and flow of time $(T, \leq_T)$.

In streamified OBDA we have to deal with different types of domains

- D is a set of typed tuples adhering to a relational schema
  - Streams at the backend sources
  - $S_{rel} = \{(s_1, 90°)\langle 1s\rangle, (s_2, 92°)\langle 2s\rangle, (s_1, 94°)\langle 3s\rangle, \dots\}$
  - Schema: hasSensorRelation(Sensor:string, temperature:integer)
- D is a set of untyped tuples (of the same arity)
  - Stream of tuples resulting as bindings for subqueries
- D is a set of assertions (RDF tuples).
  - $S_{rdf} = \{\ val(s_0, 90°)\langle 1s\rangle, val(s_2, 92°)\langle 2s\rangle,\ val(s_1, 94°)\langle 3s\rangle, \dots\ \}$
- D is a set of RDF graphs

Domain Objects for Streams

Definition (Temporal Stream)

A stream S is a sequence of timestamped objects (d, t) over some domain D and flow of time $(T, \leq_T)$.

In streamified OBDA we have to deal with different types of domains

- D is a set of typed tuples adhering to a relational schema
  - Streams at the backend sources
  - $S_{rel} = \{(s_1, 90°)\langle 1s\rangle, (s_2, 92°)\langle 2s\rangle, (s_1, 94°)\langle 3s\rangle, \dots\}$
  - Schema: hasSensorRelation(Sensor:string, temperature:integer)
- D is a set of untyped tuples (of the same arity)
  - Stream of tuples resulting as bindings for subqueries
- D is a set of assertions (RDF tuples).
  - $S_{rdf} = \{\ val(s_0, 90°)\langle 1s\rangle, val(s_2, 92°)\langle 2s\rangle,\ val(s_1, 94°)\langle 3s\rangle, \dots\ \}$
- D is a set of RDF graphs

Relational Stream Processing

- Stream stack
  1. Low-level sensor perspective (semantic sensor networks)
  2. Relational database perspective (data stream management)
  3. Ontology layer streams
- Different groups working on Data Stream Management Systems around 2004
  - Academic prototypes: STREAM and CQL (Stanford); TelgraphCQ (Berkeley) (extends PostGreSQL); Aurora/Borealis (Brandeis, Brown and MIT); PIPES from Marburg University
  - Commercial systems: StreamBase, Truviso (Satandalone), extensions of commercial DBMS (MySQL, PostgreSQL, DB2 etc.)
- Though well investigated and many similarities there is no streaming SQL standard

Window Operator

- Querying on streams is a continuous not a one-shot activity
- Fundamental idea to cope with infiniteness of streams
  - Blocking operators cannot be applied to the whole stream
  - consider finite updated window content
- Window classification
  - Direction of movement of the endpoints
    - Both endpoints fixed (needed for "historical" queries)
    - Both moving/sliding
    - One moving the other not
  - Window size
    - Temporal
    - Tuple-based
    - Partitioned window
    - Predicate window
  - Window update
    - tumbling
    - sampling
    - overlapping

Window Operator

- Querying on streams is a continuous not a one-shot activity
- Fundamental idea to cope with infiniteness of streams
  - Blocking operators cannot be applied to the whole stream
  - consider finite updated window content
- Window classification
  - Direction of movement of the endpoints
    - Both endpoints fixed (needed for "historical" queries)
    - Both moving/sliding
    - One moving the other not
  - Window size
    - Temporal
    - Tuple-based
    - Partitioned window
    - Predicate window
  - Window update
    - tumbling
    - sampling
    - overlapping

- Early relational stream query language extending SQL
- Assumes synchronous streams
- Special data structure next to streams: relations R
  - R maps times t to ordinary (instantaneous) relations $R(t)$
  - Motivation: Use of ordinary SQL operators on instantaneous relations
- Operators: Stream-to-relation, relation-to-relation, relation-stream
- Non-predictability condition for operators op:
  - If two inputs S_1 , S_2 are the same up to t, then $op(S_1)(t) = op(S_2)(t)$.

CQL Windows

- Window operators are stream-to-relation operators
- CQL knows tuple-based, partition based and time-based windows
- R = S [Range wr Slide sl]
  - with slide parameter sl and range wr
  - $t_{start} = \lfloor t/sl \rfloor \cdot sl$
  - $t_{end} = max\{t_{start} - wr, 0\}$

$$R(t) = \begin{cases} \emptyset & \text{if } t < sl \\ \{s \mid s\langle t' \rangle \in S \text{ and } t_{end} \leq t' \leq t_{start}\} & \text{else} \end{cases}$$

- Standard slide = 1: [RANGE wr]
- Left end fixed: [Range UNBOUND]
- Width 0: [NOW]

CQL Windows

- Window operators are stream-to-relation operators
- CQL knows tuple-based, partition based and time-based windows
- R = S [Range wr Slide sl]
  - with slide parameter sl and range wr
  - $t_{start} = \lfloor t/sl \rfloor \cdot sl$
  - $t_{end} = max\{t_{start} - wr, 0\}$

$$R(t) = \begin{cases} \emptyset & \text{if } t < sl \\ \{s \mid s\langle t' \rangle \in S \text{ and } t_{end} \leq t' \leq t_{start}\} & \text{else} \end{cases}$$

- Standard slide = 1: [RANGE wr]
- Left end fixed: [Range UNBOUND]
- Width 0: [NOW]

CQL Windows

- Window operators are stream-to-relation operators
- CQL knows tuple-based, partition based and time-based windows
- R = S [Range wr Slide sl]
  - with slide parameter sl and range wr
  - $t_{start} = \lfloor t/sl \rfloor \cdot sl$
  - $t_{end} = max\{t_{start} - wr, 0\}$

$$R(t) = \begin{cases} \emptyset & \text{if } t < sl \\ \{s \mid s\langle t' \rangle \in S \text{ and } t_{end} \leq t' \leq t_{start}\} & \text{else} \end{cases}$$

- Standard slide = 1: [RANGE wr]
- Left end fixed: [Range UNBOUND]
- Width 0: [NOW]

Sliding Window Example in CQL

Optique

- Flow of time $(\mathbb{N}, \leq)$
- Input stream

$$S = \{(s_0, 90^\circ)\langle 0\rangle, (s_1, 94^\circ)\langle 0\rangle, (s_0, 91^\circ)\langle 1\rangle, (s_0, 92^\circ)\langle 2\rangle, (s_0, 93^\circ)\langle 3\rangle, (s_0, 95^\circ)\langle 5\rangle, (s_0, 94^\circ)\langle 6\rangle ....\}$$

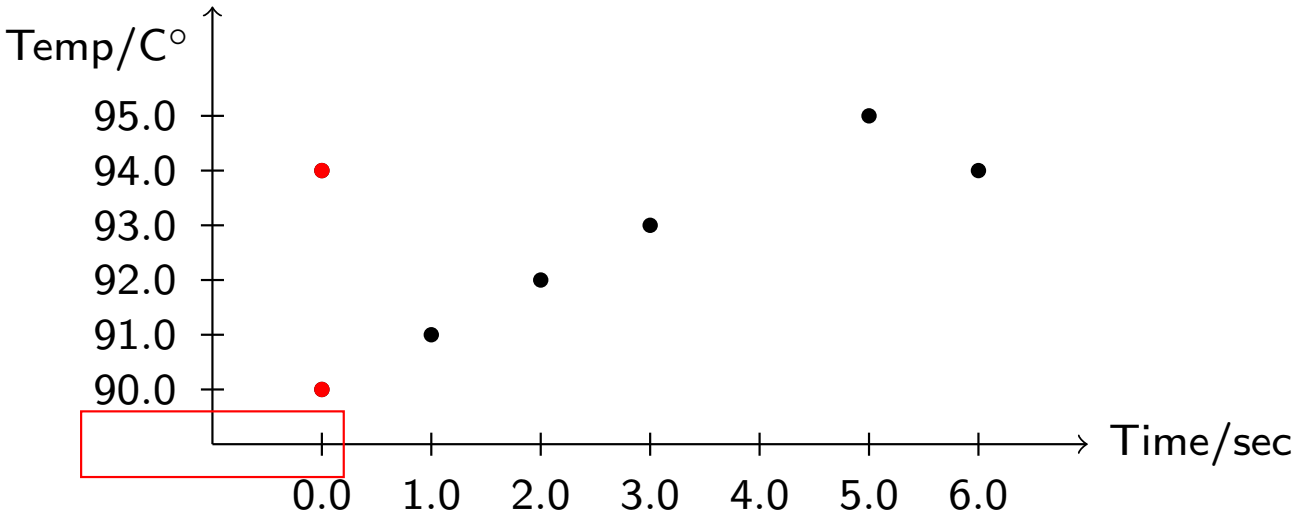
- Output relation $R =$ `S [Range 2 Slide 1]`

t :	0	1	2	3	4	5	6
$R(t)$:	$\{(s_0, 90), (s_1, 94)\}$	$\{(s_0, 90), (s_1, 94), (s_0, 91)\}$	$\{(s_0, 90), (s_1, 94), (s_0, 91), (s_0, 92)\}$	$\{(s_0, 91), (s_0, 92), (s_0, 93)\}$	$\{(s_0, 92), (s_0, 93)\}$	$\{(s_0, 92), (s_0, 93), (s_0, 95)\}$	$\{(s_0, 93), (s_0, 95), (s_0, 94)\}$

Sliding Window Example in CQL

Optique

$S = \{(s_0, 90)\langle 0\rangle, (s_1, 94)\langle 0\rangle, (s_0, 91)\langle 1\rangle, (s_0, 92)\langle 2\rangle, (s_0, 93)\langle 3\rangle, (s_0, 95)\langle 5\rangle, (s_0, 94)\langle 6\rangle\}$



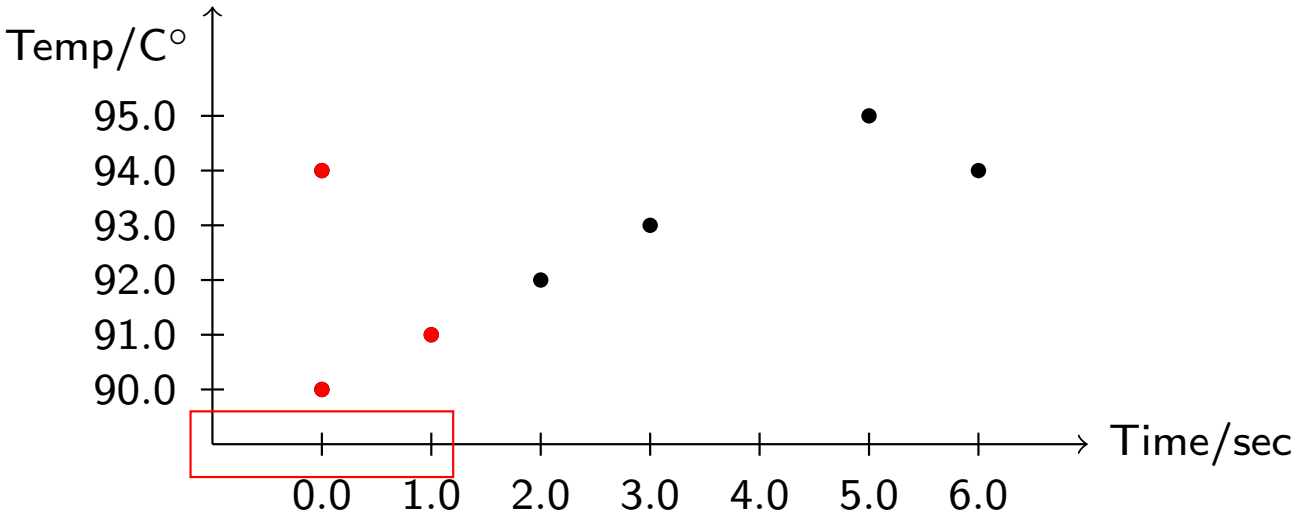
Output relation $R =$ `S [Range 2 Slide 1]`

t :	0	1	2	3	4	5	6
$R(t)$:	$\{(s_0, 90), (s_1, 94)\}$	$\{(s_0, 90), (s_1, 94), (s_0, 91)\}$	$\{(s_0, 90), (s_1, 94), (s_0, 91), (s_0, 92)\}$	$\{(s_0, 91), (s_0, 92), (s_0, 93)\}$	$\{(s_0, 92), (s_0, 93)\}$	$\{(s_0, 93), (s_0, 95)\}$	$\{(s_0, 95), (s_0, 94)\}$

Sliding Window Example in CQL

Optique

$S = \{(s_0, 90)\langle 0\rangle, (s_1, 94)\langle 0\rangle, (s_0, 91)\langle 1\rangle, (s_0, 92)\langle 2\rangle, (s_0, 93)\langle 3\rangle, (s_0, 95)\langle 5\rangle, (s_0, 94)\langle 6\rangle\}$



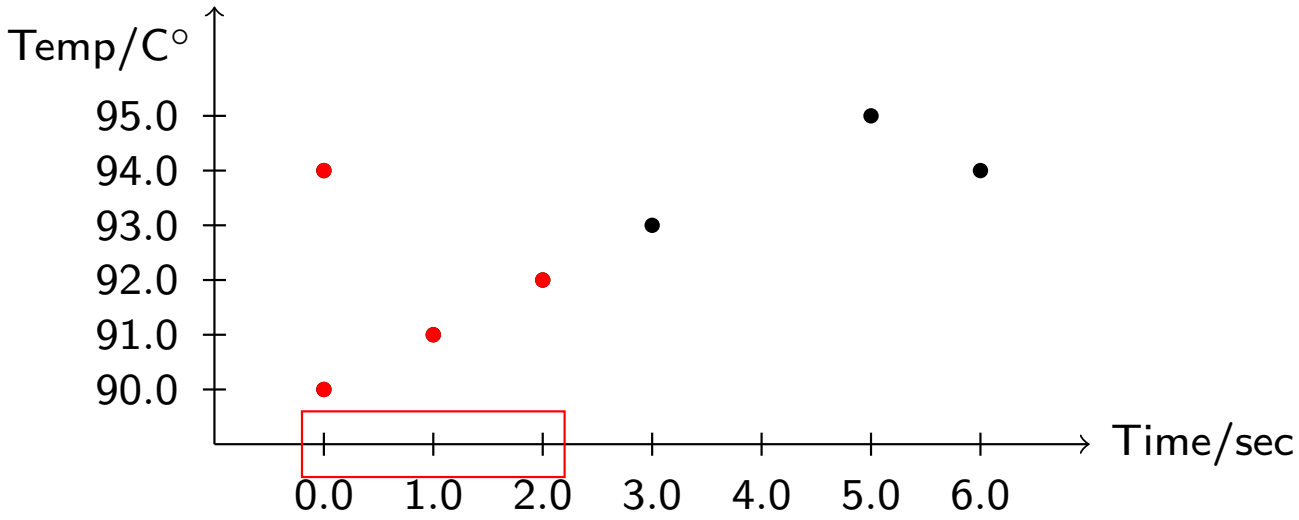
Output relation $R =$ `S [Range 2 Slide 1]`

t :	0	1	2	3	4	5	6
$R(t)$:	$\{(s_0, 90), (s_1, 94)\}$	$\{(s_0, 90), (s_1, 94), (s_0, 91)\}$	$\{(s_0, 90), (s_1, 94), (s_0, 91), (s_0, 92)\}$	$\{(s_0, 91), (s_0, 92), (s_0, 93)\}$	$\{(s_0, 92), (s_0, 93)\}$	$\{(s_0, 93), (s_0, 95)\}$	$\{(s_0, 95), (s_0, 94)\}$

Sliding Window Example in CQL

Optique

$S = \{(s_0, 90)\langle 0\rangle, (s_1, 94)\langle 0\rangle, (s_0, 91)\langle 1\rangle, (s_0, 92)\langle 2\rangle, (s_0, 93)\langle 3\rangle, (s_0, 95)\langle 5\rangle, (s_0, 94)\langle 6\rangle\}$



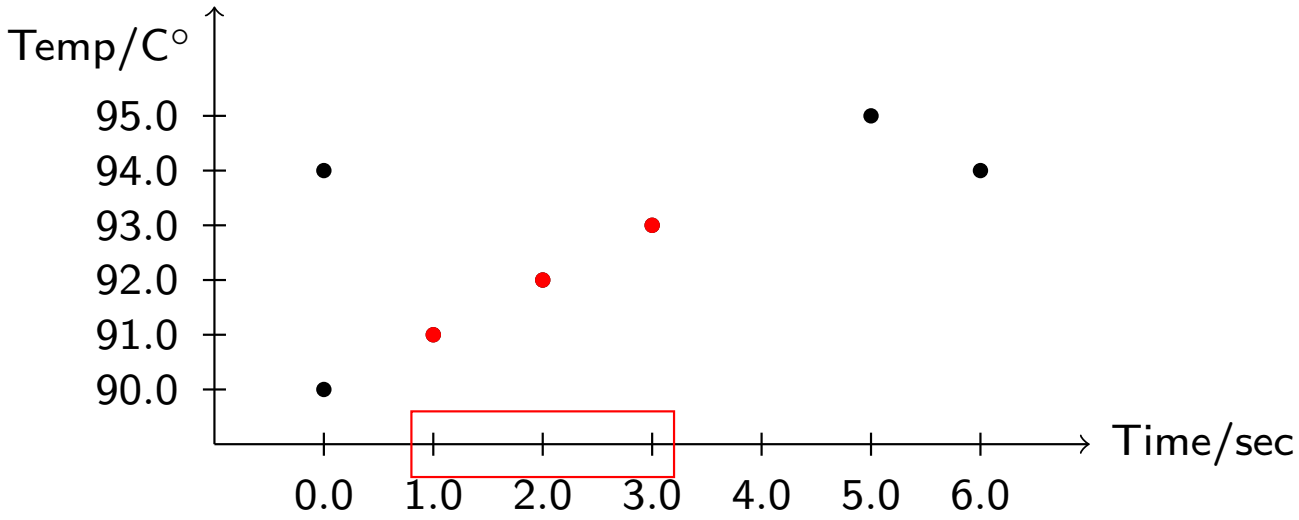
Output relation $R =$ `S [Range 2 Slide 1]`

t :	0	1	2	3	4	5	6
$R(t)$:	$\{(s_0, 90), (s_1, 94)\}$	$\{(s_0, 90), (s_1, 94), (s_0, 91)\}$	$\{(s_0, 90), (s_1, 94), (s_0, 91), (s_0, 92)\}$	$\{(s_0, 91), (s_0, 92), (s_0, 93)\}$	$\{(s_0, 92), (s_0, 93)\}$	$\{(s_0, 93), (s_0, 95)\}$	$\{(s_0, 95), (s_0, 94)\}$

Sliding Window Example in CQL

Optique

$S = \{(s_0, 90)\langle 0\rangle, (s_1, 94)\langle 0\rangle, (s_0, 91)\langle 1\rangle, (s_0, 92)\langle 2\rangle, (s_0, 93)\langle 3\rangle, (s_0, 95)\langle 5\rangle, (s_0, 94)\langle 6\rangle\}$



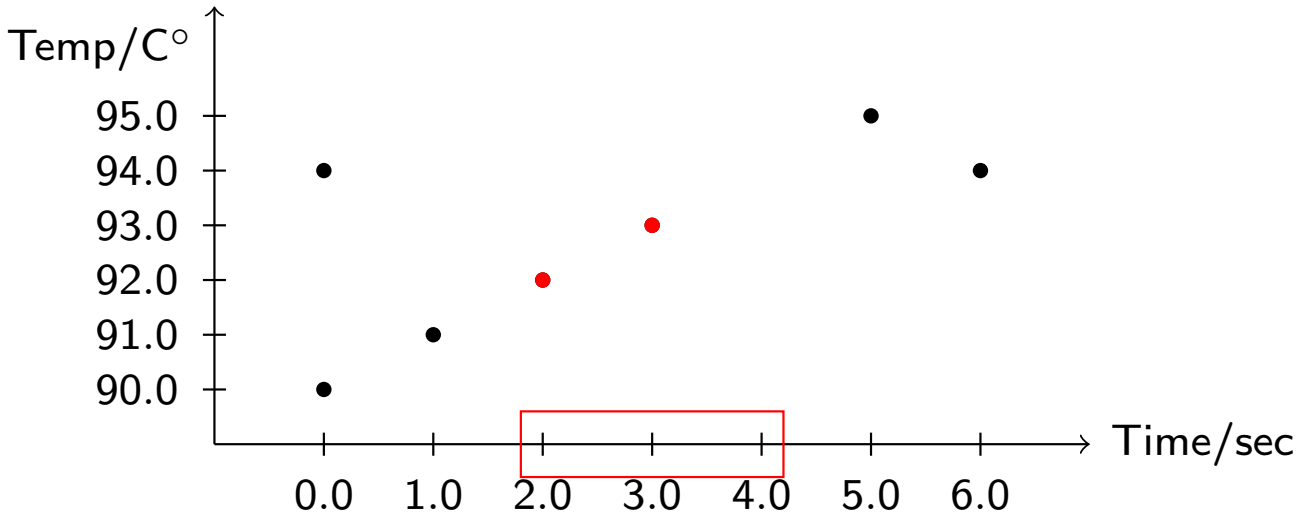
Output relation $R =$ `S [Range 2 Slide 1]`

t :	0	1	2	3	4	5	6
$R(t)$:	$\{(s_0, 90), (s_1, 94)\}$	$\{(s_0, 90), (s_1, 94), (s_0, 91)\}$	$\{(s_0, 90), (s_1, 94), (s_0, 91), (s_0, 92)\}$	$\{(s_0, 91), (s_0, 92), (s_0, 93)\}$	$\{(s_0, 92), (s_0, 93)\}$	$\{(s_0, 93), (s_0, 95)\}$	$\{(s_0, 95), (s_0, 94)\}$

Sliding Window Example in CQL

Optique

$S = \{(s_0, 90)\langle 0\rangle, (s_1, 94)\langle 0\rangle, (s_0, 91)\langle 1\rangle, (s_0, 92)\langle 2\rangle, (s_0, 93)\langle 3\rangle, (s_0, 95)\langle 5\rangle, (s_0, 94)\langle 6\rangle\}$



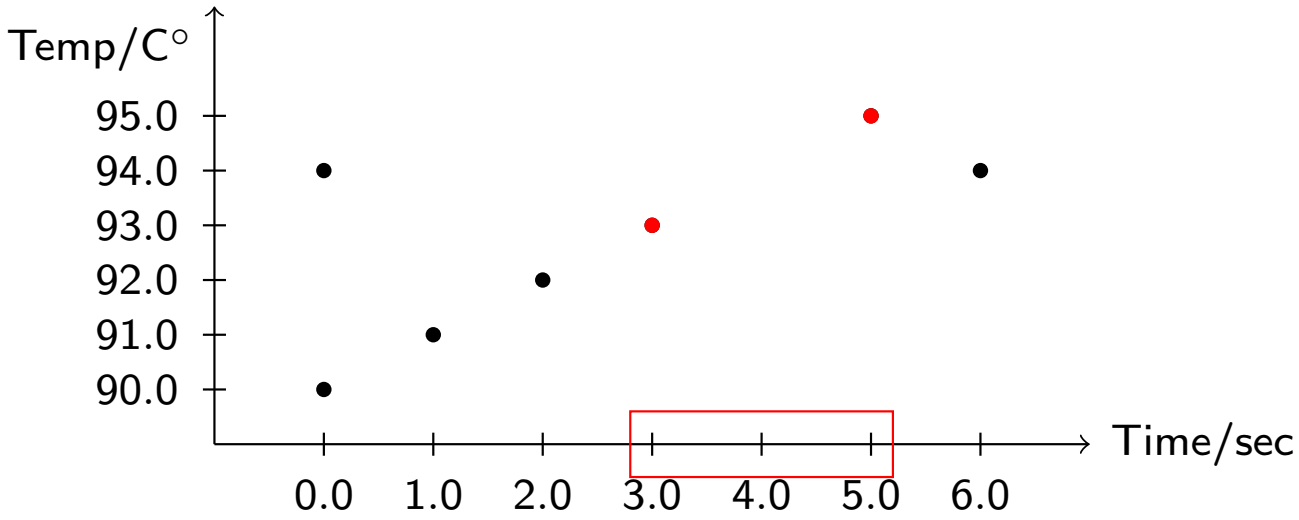
Output relation $R =$ `S [Range 2 Slide 1]`

t :	0	1	2	3	4	5	6
$R(t)$:	$\{(s_0, 90), (s_1, 94)\}$	$\{(s_0, 90), (s_1, 94), (s_0, 91)\}$	$\{(s_0, 90), (s_1, 94), (s_0, 91), (s_0, 92)\}$	$\{(s_0, 91), (s_0, 92), (s_0, 93)\}$	$\{(s_0, 92), (s_0, 93)\}$	$\{(s_0, 93), (s_0, 95)\}$	$\{(s_0, 95), (s_0, 94)\}$

Sliding Window Example in CQL

Optique

$$S = \{(s_0, 90)\langle 0\rangle, (s_1, 94)\langle 0\rangle, (s_0, 91)\langle 1\rangle, (s_0, 92)\langle 2\rangle, (s_0, 93)\langle 3\rangle, (s_0, 95)\langle 5\rangle, (s_0, 94)\langle 6\rangle\}$$



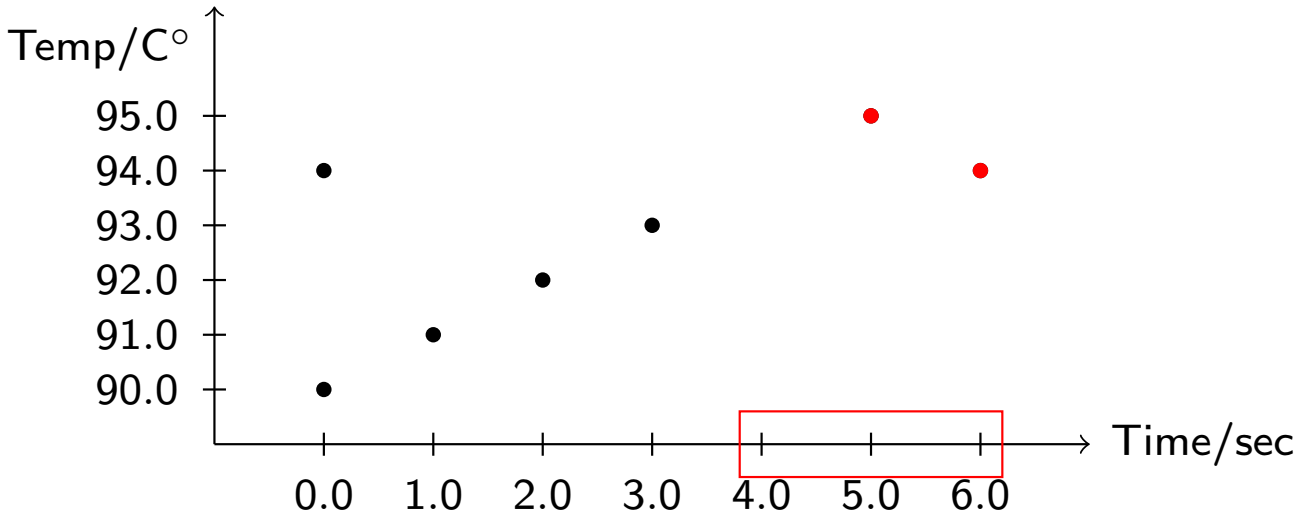
Output relation $R =$ `S [Range 2 Slide 1]`

t :	0	1	2	3	4	5	6
$R(t)$:	$\{(s_0, 90), (s_1, 94)\}$	$\{(s_0, 90), (s_1, 94), (s_0, 91)\}$	$\{(s_0, 90), (s_1, 94), (s_0, 91), (s_0, 92)\}$	$\{(s_0, 91), (s_0, 92), (s_0, 93)\}$	$\{(s_0, 92), (s_0, 93)\}$	$\{(s_0, 93), (s_0, 95)\}$	$\{(s_0, 95), (s_0, 94)\}$

Sliding Window Example in CQL

Optique

$$S = \{(s_0, 90)\langle 0\rangle, (s_1, 94)\langle 0\rangle, (s_0, 91)\langle 1\rangle, (s_0, 92)\langle 2\rangle, (s_0, 93)\langle 3\rangle, (s_0, 95)\langle 5\rangle, (s_0, 94)\langle 6\rangle\}$$



Output relation $R =$ `S [Range 2 Slide 1]`

t :	0	1	2	3	4	5	6
$R(t)$:	$\{(s_0, 90), (s_1, 94)\}$	$\{(s_0, 90), (s_1, 94), (s_0, 91)\}$	$\{(s_0, 90), (s_1, 94), (s_0, 91), (s_0, 92)\}$	$\{(s_0, 91), (s_0, 92), (s_0, 93)\}$	$\{(s_0, 92), (s_0, 93)\}$	$\{(s_0, 93), (s_0, 95)\}$	$\{(s_0, 95), (s_0, 94)\}$

Relation vs. Stream

$$S = \{(s_0, 90^\circ)\langle 0\rangle, (s_1, 94^\circ)\langle 0\rangle, (s_0, 91^\circ)\langle 1\rangle, (s_0, 92^\circ)\langle 2\rangle, (s_0, 93^\circ)\langle 3\rangle, (s_0, 95^\circ)\langle 5\rangle, (s_0, 94^\circ)\langle 6\rangle ....\}$$

- Output relation $R =$ `S [Range 2 Slide 1]`

t :	0	1	2	3	4	5	6
$R(t)$:	$\{(s_0, 90), (s_1, 94)\}$	$\{(s_0, 90), (s_1, 94), (s_0, 91)\}$	$\{(s_0, 90), (s_1, 94), (s_0, 91), (s_0, 92)\}$	$\{(s_0, 91), (s_0, 92), (s_0, 93)\}$	$\{(s_0, 92), (s_0, 93)\}$	$\{(s_0, 92), (s_0, 93), (s_0, 95)\}$	$\{(s_0, 93), (s_0, 95), (s_0, 94)\}$

- Note that there are also entries for second 4
- Note that timestamps are lost in the bags
- Slides are local to streams and may be different over different streams

Relation-To-Stream Operators

- Output stream of input relation R:

$$\begin{aligned} \mathit{Istream}(R) &= \bigcup_{t \in T} (R(t) \setminus R(t-1)) \times \{t\} \\ &\quad \text{stream of inserted elements} \\ \mathit{Dstream}(R) &= \bigcup_{t \in T} (R(t-1) \setminus R(t)) \times \{t\} \\ &\quad \text{stream of deleted elements} \\ \mathit{Rstream}(R) &= \bigcup_{t \in T} R(t) \times \{t\} \\ &\quad \text{stream of all elements} \end{aligned}$$

- In CQL *IStream* and *DStream* are syntactic sugar

Sensor Measurement CQL Example

Optique

```
SELECT Rstream(m.sensorID)
FROM Msmt[Range 1] as m, Events[Range 2] as e
WHERE m.val > 30 AND
      e.category = Alarm AND
      m.sensorID = e.sensorID
```

- Stream join realized by join of window contents
- Output is a stream

Non-discrete Time Flows

- Taken literally, CQL window definitions work only for discrete flows of times
- Time flow: $(T, \leq) = (\mathbb{R}, \leq)$
- Input stream: $S = \{i\langle i \rangle \mid i \in \mathbb{N}\}$
- *RStream*(*S*[*RANGE* 1 *SLIDE* 1]) is "stream" with cardinality of $\mathbb{R}$
- "Solution" in CQL hidden in stream engine layer
- Heartbeat with smallest possible time granularity

Streamified OBDA

- Nearly ontology layer stream processing
  - CEP (Complex event processing)
  - EP-SPARQL/ETALIS, T-REX/ TESLA, Commonsens/ESPER
- RDF-ontology layer stream processing
  - C-SPARQL (della Valle et al. 09), CQELS
- Classical OBDA stream processing
  - SPARQL_{Stream} (Calbimonte et al. 12)
- All approaches rely on CQL window semantics
- extend SPARQL or use some derivative of it
- Treat timestamped RDF triples but use reification

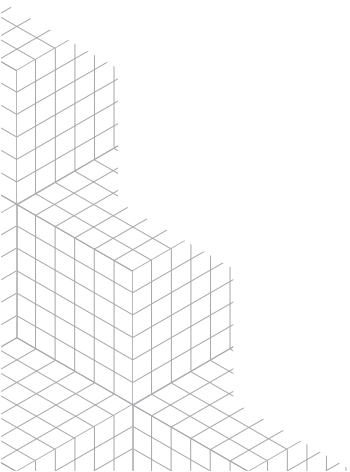
Example of Reified Handling


```
SELECT ?windspeed ?tidespeed
FROM NAMED STREAM <http://swiss-experiment.ch/data#WannengratSensors.srdf>
[NOW-10 MINUTES TO NOW-0 MINUTES]
WHERE
  ?WaveObs a ssn:Observation;
           ssn:observationResult ?windspeed;
           ssn:observedProperty sweetSpeed:WindSpeed.
  ?TideObs a ssn:Observation;
           ssn:observationResult ?tidespeed;
           ssn:observedProperty sweetSpeed:TideSpeed.
  FILTER (?tidespeed<?windspeed)
```

SRBench (Zhang et al. 2012)

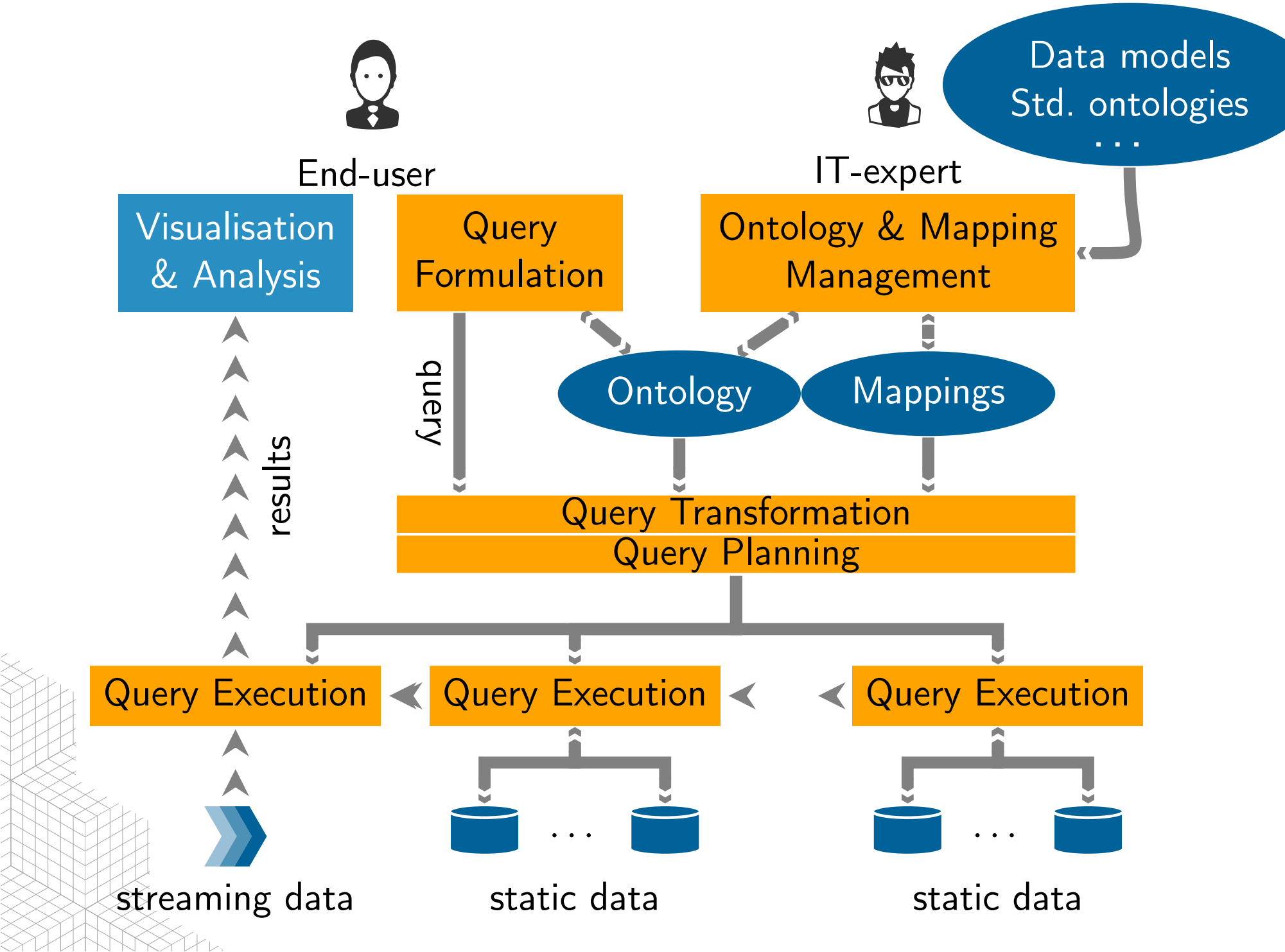
- Benchmark for RDF/SPARQL Stream Engines
- Contains data from LinkedSensorData, GeoNames, DBPedia
- Mainly queries for functionality tests, with eye on SPARQL 1.1. functionalities
- Example Query (to test basic pattern matching)
  Q1. Get the rainfall observed once in an hour.
- Tested on CQELS, SPARQL$_{Stream}$ and C-SPARQL
- Test results (for engine versions as of 2012)
  - Basic SPARQL features supported
  - SPARQL 1.1 features (property paths) rather not supported
  - Only C-SPARQL supports reasoning (on RDFS level)
    (tested subsumption and sameAs)
  - Combined treatment of static data plus streaming data only for CQELS and C-SPARQL

PART III: STARQL



OPTIQUE

- EU 7th framework program
- Two big data use cases from industrial partners
  - STATOIL SAS: Querying data on wellbore related DBs
  - SIEMENS: Querying sensor and event data from (gas) turbines
- Optique platform: OBDA + User Friendliness + Scalable Rewriting + Elastic Clouds + Real-Time Processing
- For more information: http://www.optique-project.eu/


Data models
Std. ontologies
...
End-user
IT-expert
Visualisation & Analysis
Query Formulation
Ontology & Mapping Management
query
results
Ontology
Mappings
Query Transformation
Query Planning
Query Execution
Query Execution
Query Execution
...
...
streaming data
static data
static data

The query framework STARQL

- Started development within OPTIQUE
- Uses non-reified approach
- Use local temporal reasoning on finite state sequences
- Has framework character: embed different condition languages
- Convention for the following
  - Use logical ABox/TBox notation for RDF assertions (also in streams.), i.e.,
    - `{s0 rdf:type TempSensor}` written as TempSens(s0).
    - `{s0 val 90}` written as val(s0,90)
  - Use SPARQL notation within STARQL queries.

The query framework STARQL

- Started development within OPTIQUE
- Uses non-reified approach
- Use local temporal reasoning on finite state sequences
- Has framework character: embed different condition languages
- Convention for the following
  - Use logical ABox/TBox notation for RDF assertions (also in streams.), i.e.,
    - `{s0 rdf:type TempSensor}` written as TempSens(s0).
    - `{s0 val 90}` written as val(s0,90)
  - Use SPARQL notation within STARQL queries.

Structure of STARQL queries

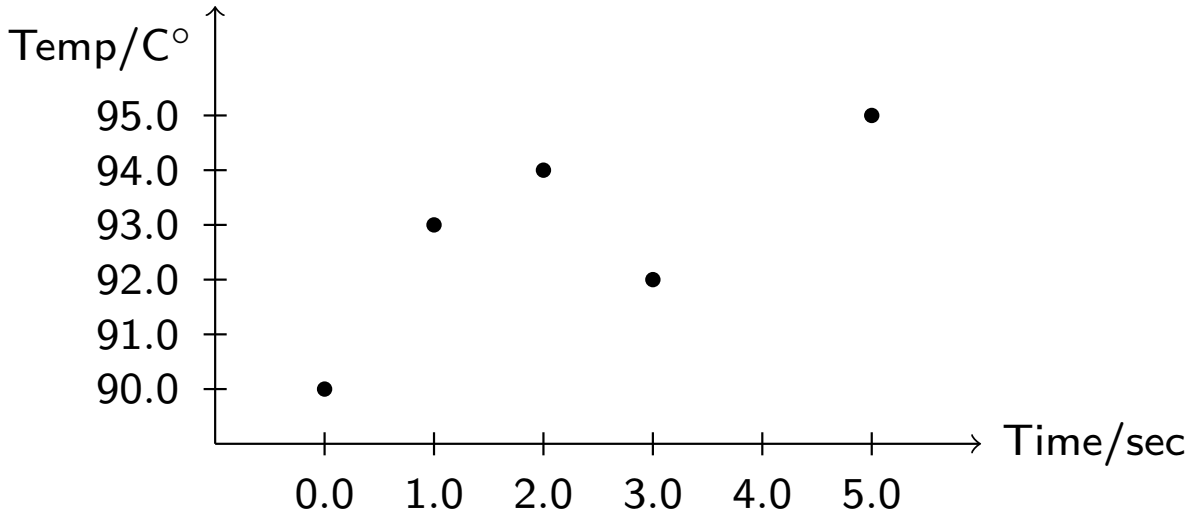
STARQL Query Template

```
CREATE STREAM                    (initializes new stream)
CREATE PULSE                          (create pulse for
                                          output times)
SELECT/                        (specifies output format)
CONSTRUCT
FROM                     (specifies the input streams)
USING                     (specifies the static input)
WHERE                   (selection w.r.t. static data)
SEQUENCE BY                          (sequencing strategy)
HAVING                (FOL template for local temporal
                                    reasoning on states)
```

A Basic STARQL Example

Input: Stream S_{Msmt} of measurement assertions.

$$
\begin{aligned}
S_{Msmt} \quad = \quad \{ \; & val(s_0, 90^\circ C)\langle 0s \rangle, \\
& val(s_0, 93^\circ C)\langle 1s \rangle, \\
& val(s_0, 94^\circ C)\langle 2s \rangle, \\
& val(s_0, 92^\circ C)\langle 3s \rangle, \\
& val(s_0, 95^\circ C)\langle 5s \rangle \\
& \ldots \}
\end{aligned}
$$



Information Needs in STARQL

Information Need for Monotonicity (IN-Mon)

Tell every 1s whether the temperature in sensor $s0$ increased monotonically in the last 2s.

STARQL Representation (STARQL-Mon)

```
CREATE STREAM S_out_1 AS
PULSE START = 0s, FREQUENCY = 1s
CONSTRUCT {s0 rdf:type RecMonInc}<NOW>
FROM S_Msmt [NOW-2s, NOW]->1s
SEQUENCE BY StdSeq AS SEQ1
HAVING FORALL i < j IN SEQ1,?x,?y:
       IF {s0 val ?x}<i>  AND {s0  val ?y}<j>
       THEN ?x <= ?y
```

Components of STARQL

Information Need for Monotonicity (IN-Mon)

Tell every 1s whether the temperature in sensor *s*0 increased monotonically in the last 2s in stream S_Msmt.

```
CREATE STREAM S_out_1 AS
PULSE START = 0s, FREQUENCY = 1s
CONSTRUCT {s0 rdf:type RecMonInc}<NOW>
FROM S_Msmt [NOW-2s, NOW]->1s
SEQUENCE BY StdSeq AS SEQ1
HAVING FORALL i < j IN SEQ1,?x,?y:
       IF {s0 :val ?x}<i>  AND {s0 :val ?y}<j>
       THEN ?x <= ?y
```

- Creates stream named S_{out_1}
- Can be referenced under this name within another query

Components of STARQL

Information Need for Monotonicity (IN-Mon)

Tell every 1s whether the temperature in sensor *s*0 increased monotonically in the last 2s in stream S_Msmt.

```
CREATE STREAM S_out_1 AS
PULSE START = 0s, FREQUENCY = 1s
CONSTRUCT {s0 rdf:type RecMonInc}<NOW>
FROM S_Msmt [NOW-2s, NOW]->1s
SEQUENCE BY StdSeq AS SEQ1
HAVING FORALL i < j IN SEQ1,?x,?y:
       IF {s0 :val ?x}<i>  AND {s0 :val ?y}<j>
       THEN ?x <= ?y
```

- CONSTRUCT is a SPARQL like constructor
- Alternatively, if one is interested only in the bindings one uses SELECT

Output Format

$$
\begin{aligned}
S_{Msmt} = \{\ & val(s_0, 90^\circ C)\langle 0s\rangle, \\
& val(s_0, 93^\circ C)\langle 1s\rangle, \\
& val(s_0, 94^\circ C)\langle 2s\rangle, \\
& val(s_0, 92^\circ C)\langle 3s\rangle, \\
& \\
& val(s_0, 95^\circ C)\langle 5s\rangle \\
& \dots \}
\end{aligned}
\qquad
\begin{aligned}
S_{out_1} = \{\ & RecMonInc(s_0)\langle 0s\rangle, \\
& RecMonInc(s_0)\langle 1s\rangle, \\
& RecMonInc(s_0)\langle 2s\rangle, \\
& \\
& \\
& RecMonInc(s_0)\langle 5s\rangle \\
& \dots \}
\end{aligned}
$$

Components of STARQL

Information Need for Monotonicity (IN-Mon)

Tell every 1s whether the temperature in sensor *s*0 increased monotonically in the last 2s in stream S_Msmt.

```
CREATE STREAM S_out_1 AS
PULSE START = 0s, FREQUENCY = 1s
CONSTRUCT {s0 rdf:type RecMonInc}<NOW>
FROM S_Msmt [NOW-2s, NOW]-> 1s
SEQUENCE BY StdSeq AS SEQ1
HAVING FORALL i < j IN SEQ1,?x,?y:
       IF {s0 :val ?x}<i>  AND {s0 :val ?y}<j>
       THEN ?x <= ?y
```

- Pulse fixes output times (bindings of NOW variable)
- Needed also for synchronization of streams

Components of STARQL

Information Need for Monotonicity (IN-Mon)

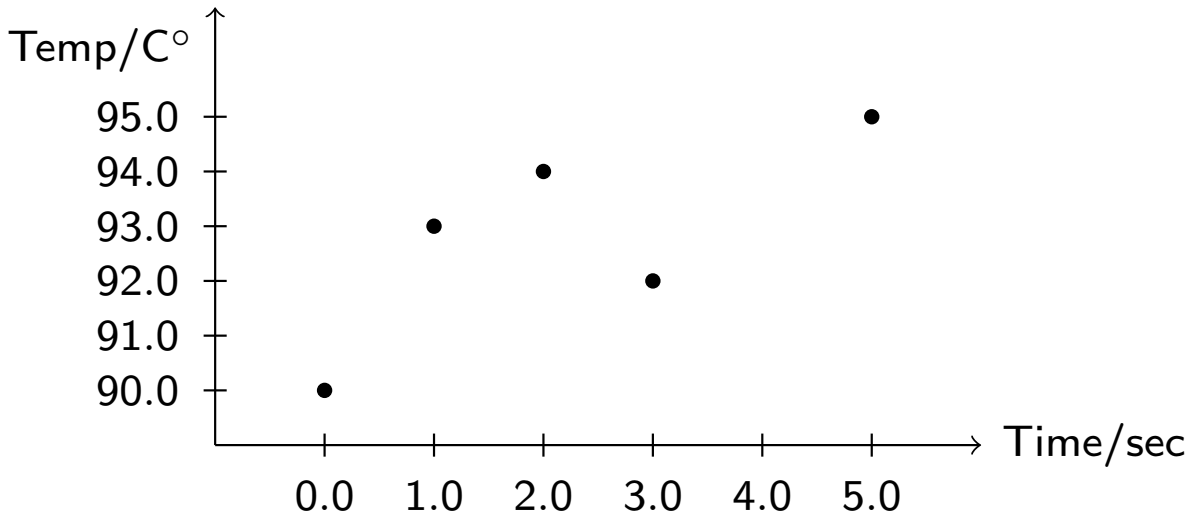
Tell every 1s whether the temperature in sensor *s*0 increased monotonically in the last 2s in stream S_Msmt.

```
CREATE STREAM S_out_1 AS
PULSE START = 0s, FREQUENCY = 1s
CONSTRUCT {s0 rdf:type RecMonInc}<NOW>
FROM S_Msmt [NOW-2s, NOW]-> 1s
SEQUENCE BY StdSeq AS SEQ1
HAVING FORALL i < j IN SEQ1,?x,?y:
       IF {s0 :val ?x}<i>  AND {s0 :val ?y}<j>
       THEN ?x <= ?y
```

- Window specification with window interval and slide parameter
- Applied to input stream S_Msmt of timestamped RDF assertions (= RDF quadruples)

Window Semantics

- `S_Msmt [NOW-2s,NOW]->1s`: stream of temporal ABoxes
- Sliding movement as in CQL but with timestamp preservation



Window sliding every second

Time	Window contents
$0s$	$val(s_0, 90°)\langle 0s\rangle$
$1s$	$val(s_0, 90°)\langle 0s\rangle$, $val(s_0, 93°)\langle 1s\rangle$
$2s$	$val(s_0, 90°)\langle 0s\rangle$, $val(s_0, 93°)\langle 1s\rangle$, $val(s_0, 94°)\langle 2s\rangle$
$3s$	$val(s_0, 93°)\langle 1s\rangle$, $val(s_0, 94°)\langle 2s\rangle$, $val(s_0, 92°)\langle 3s\rangle$
$4s$	$val(s_0, 94°)\langle 2s\rangle$, $val(s_0, 92°)\langle 3s\rangle$
$5s$	$val(s_0, 92°)\langle 3s\rangle$, $val(s_0, 95°)\langle 5s\rangle$

Components of STARQL

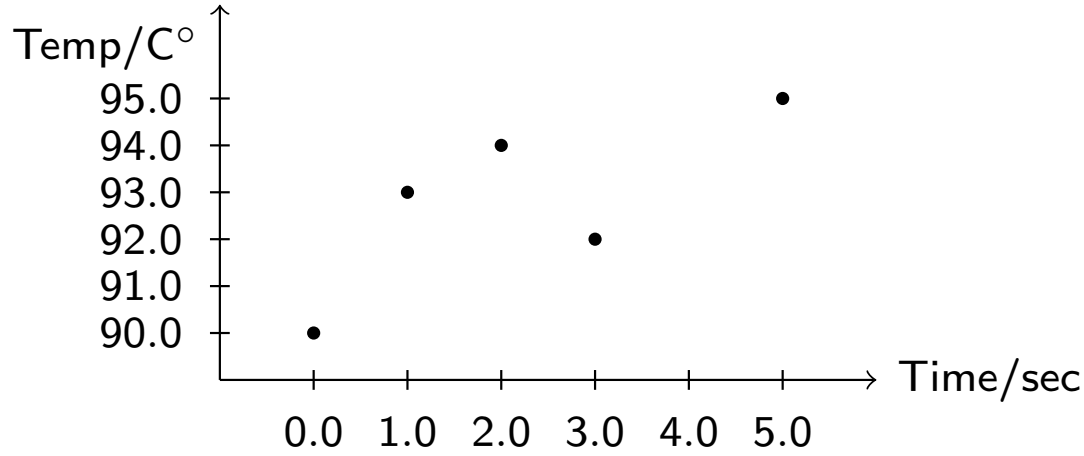
Information Need for Monotonicity (IN-Mon)

Tell every 1s whether the temperature in sensor *s*0 increased monotonically in the last 2s in stream S_Msmt.

```
CREATE STREAM S_out_1 AS
PULSE START = 0s, FREQUENCY = 1s
CONSTRUCT {s0 rdf:type RecMonInc}<NOW>
FROM S_Msmt [NOW-2s, NOW] -> 1s
SEQUENCE BY StdSeq AS SEQ1
HAVING FORALL i < j IN SEQ1,?x,?y:
       IF {s0 :val ?x}<i>  AND {s0 :val ?y}<j>
       THEN ?x <= ?y
```

- Generate every 1 second a sequence of states referred to by variables i, j
- States are annotated with ABoxes (RDF repositories)
- StdSeq = Standard Sequencing

The Sequence View



Time	Window contents before sequencing
0s	$val(s_0, 90°)\langle 0s\rangle$
1s	$val(s_0, 90°)\langle 0s\rangle$, $val(s_0, 93°)\langle 1s\rangle$
2s	$val(s_0, 90°)\langle 0s\rangle$, $val(s_0, 93°)\langle 1s\rangle$, $val(s_0, 94°)\langle 2s\rangle$
3s	$val(s_0, 93°)\langle 1s\rangle$, $val(s_0, 94°)\langle 2s\rangle$, $val(s_0, 92°)\langle 3s\rangle$
4s	$val(s_0, 94°)\langle 2s\rangle$, $val(s_0, 92°)\langle 3s\rangle$
5s	$val(s_0, 92°)\langle 3s\rangle$, $val(s_0, 95°)\langle 5s\rangle$

Time	Window contents after standard sequencing	$SEQ1$
0s	$\{val(s_0, 90°)\}\langle 0\rangle$	$\{0\}$
1s	$\{val(s_0, 90°)\}\langle 0\rangle$, $\{val(s_0, 93°)\}\langle 1\rangle$	$\{0, 1\}$
2s	$\{val(s_0, 90°)\}\langle 0\rangle$, $\{val(s_0, 93°)\}\langle 1\rangle$, $\{val(s_0, 94°)\}\langle 2\rangle$	$\{0, 1, 2\}$
3s	$\{val(s_0, 93°)\}\langle 0\rangle$, $\{val(s_0, 94°)\}\langle 1\rangle$, $\{val(s_0, 92°)\}\langle 2\rangle$	$\{0, 1, 2\}$
4s	$\{val(s_0, 94°)\}\langle 0\rangle$, $\{val(s_0, 92°)\}\langle 1\rangle$	$\{0, 1\}$
5s	$\{val(s_0, 92°)\}\langle 0\rangle$, $\{val(s_0, 95°)\}\langle 1\rangle$	{0,1}

Timestamps to Sequences

Optique

Time	Window contents before sequencing
...	...
$5s$	$val(s_0, 92°)\langle 3s\rangle$, $val(s_0, 95°)\langle 5s\rangle$

Time	Window contents after standard sequencing	$SEQ1$
...	...	
5s	$\{val(s_0, 92°)\}\langle 0\rangle$, $\{val(s_0, 95°)\}\langle 1\rangle$	{0,1}

- Timestamped assertions are grouped to ABoxes with state index
- Information on timestamps and on their distance gets lost
- The index set SEQ may be different at every time point NOW
- One may think of SEQ as a dynamic relation giving for every time point the set of states
- For unfolding: Additionally SEQ may contain for every state also the corresponding timestamp.

Timestamps to Sequences

Time	Window contents before sequencing	
...	...	
5s	$val(s_0, 92°)\langle 3s\rangle$, $val(s_0, 95°)\langle 5s\rangle$	
Time	Window contents after standard sequencing	$SEQ1$
...	...	
5s	$\{val(s_0, 92°)\}\langle 0\rangle$, $\{val(s_0, 95°)\}\langle 1\rangle$	{0,1}

- Timestamped assertions are grouped to ABoxes with state index
- Information on timestamps and on their distance gets lost
- The index set SEQ may be different at every time point NOW
- One may think of SEQ as a dynamic relation giving for every time point the set of states
- For unfolding: Additionally SEQ may contain for every state also the corresponding timestamp.

Why at all Bother with State Sequences? Optique

- Building microcosm for LTL like temporal reasoning on states
- But note
  - Temporal logic frameworks presuppose state sequences
  - In contrast, sequence construction is part of STARQL query
- Can, if needed, regain information by timestamp function on states
- With state approach one can handle non-standard sequencing techniques
  - for advanced machine learning techniques
  - in order to realize pre-processing: Filter out inconsistent ABoxes
  - in order to realize pre-processing: Roughen time granularity

Non-Standard Sequencing

- Use arbitrary congruence $\sim$ on time domain for sequencing
- Example: $x \sim y$ iff $\lfloor x/2 \rfloor = \lfloor y/2 \rfloor$ for all $x, y \in T = \mathbb{N}$.

Time	Window contents before sequencing
$0s$	$val(s_0, 90°)\langle 0s \rangle$
$1s$	$val(s_0, 90°)\langle 0s \rangle$, $val(s_0, 93°)\langle 1s \rangle$
$2s$	$val(s_0, 90°)\langle 0s \rangle$, $val(s_0, 93°)\langle 1s \rangle$, $val(s_0, 94°)\langle 2s \rangle$
$3s$	$val(s_0, 93°)\langle 1s \rangle$, $val(s_0, 94°)\langle 2s \rangle$, $val(s_0, 92°)\langle 3s \rangle$
$4s$	$val(s_0, 94°)\langle 2s \rangle$, $val(s_0, 92°)\langle 3s \rangle$
$5s$	$val(s_0, 92°)\langle 3s \rangle$, $val(s_0, 95°)\langle 5s \rangle$
Time	**Window contents after $\sim$ sequencing**
$0s$	$\{val(s_0, 90°)\}\langle 0 \rangle$
$1s$	$\{val(s_0, 90°), val(s_0, 93°)\}\langle 0 \rangle$
$2s$	$\{val(s_0, 90°), val(s_0, 93°)\}\langle 0 \rangle$, $\{val(s_0, 94°)\}\langle 1 \rangle$
$3s$	$\{val(s_0, 93°)\}\langle 0 \rangle$, $\{val(s_0, 94°), val(s_0, 92°)\}\langle 1 \rangle$
$4s$	$\{val(s_0, 94°), val(s_0, 92°)\}\langle 0 \rangle$
$5s$	$\{val(s_0, 92°)\}\langle 0 \rangle$, $\{val(s_0, 95°)\}\langle 1 \rangle$

Components of STARQL

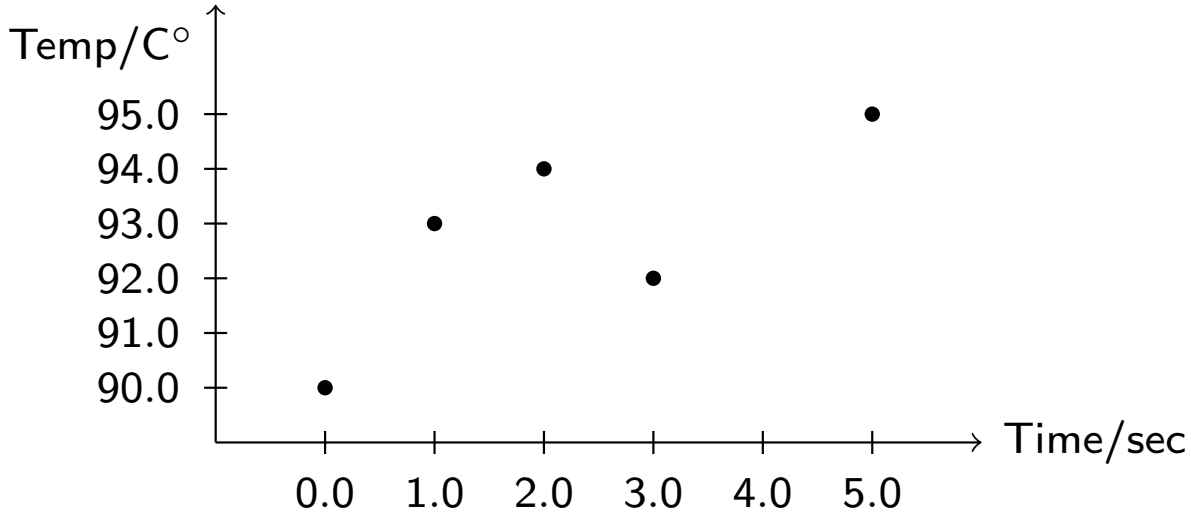
Information Need for Monotonicity (IN-Mon)

Tell every 1s whether the temperature in sensor *s0* increased monotonically in the last 2s in stream S_Msmt.

```
CREATE STREAM S_out_1 AS
PULSE START = 0s, FREQUENCY = 1s
CONSTRUCT {s0 rdf:type RecMonInc}<NOW>
FROM S_Msmt [NOW-2s, NOW] -> 1s
SEQUENCE BY StdSeq AS SEQ1
HAVING FORALL i < j IN SEQ1,?x,?y:
       IF {s0 :val ?x}<i>  AND {s0 :val ?y}<j>
       THEN ?x <= ?y
```

- First order condition over states with special "atoms"
- Informal epistemic semantics of {s0 :val ?x}<i>:
  it is (provably) the case that in state i s0 has value ?x.

Testing the Conditions



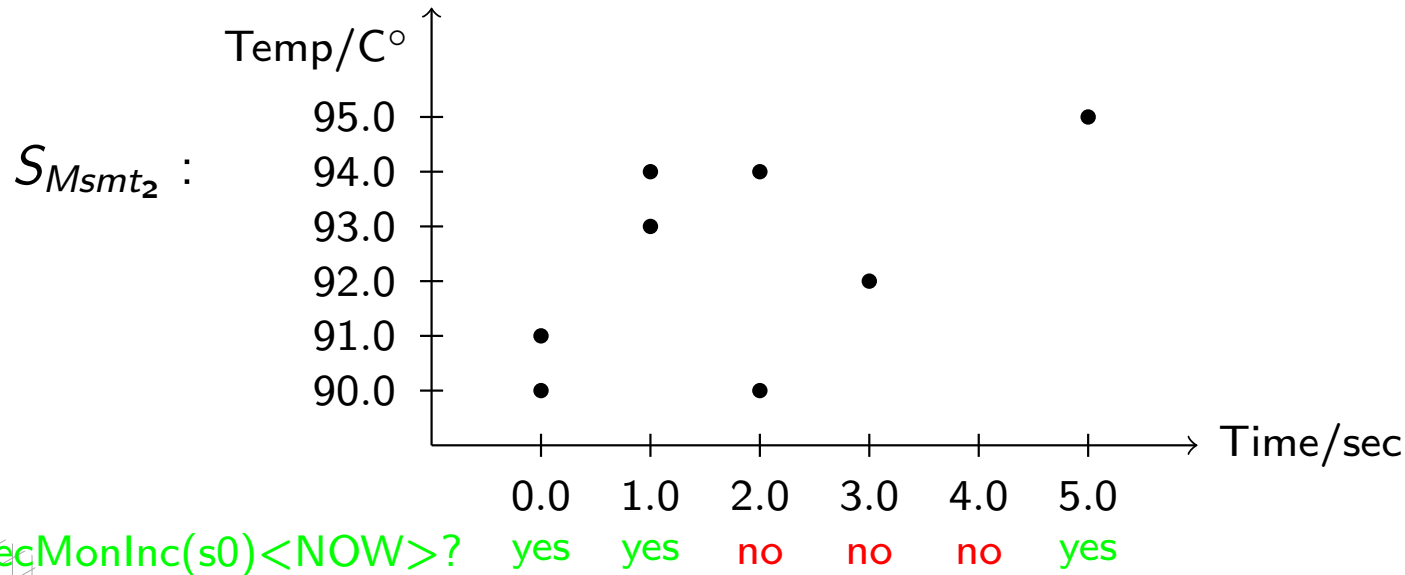
$$S_{Msmt} = \{\ val(s_0, 90^\circ C)\langle 0s\rangle,\ val(s_0, 93^\circ C)\langle 1s\rangle,\ val(s_0, 94^\circ C)\langle 2s\rangle,\ val(s_0, 92^\circ C)\langle 3s\rangle,\ val(s_0, 95^\circ C)\langle 5s\rangle\ \ldots\ \}$$

$$S_{out_1} = \{\ RecMonInc(s_0)\langle 0s\rangle,\ RecMonInc(s_0)\langle 1s\rangle,\ RecMonInc(s_0)\langle 2s\rangle,\ RecMonInc(s_0)\langle 5s\rangle\ \ldots\ \}$$

Intricacies of the Monotonicity Condition

Information Need for Monotonicity (IN-Mon)

```
...
HAVING FORALL i < j IN SEQ1,?x,?y:
       IF {s0 :val ?x}<i>  AND {s0 :val ?y}<j>
       THEN ?x <= ?y
```



Expressive Strength of HAVING Clauses

Optique

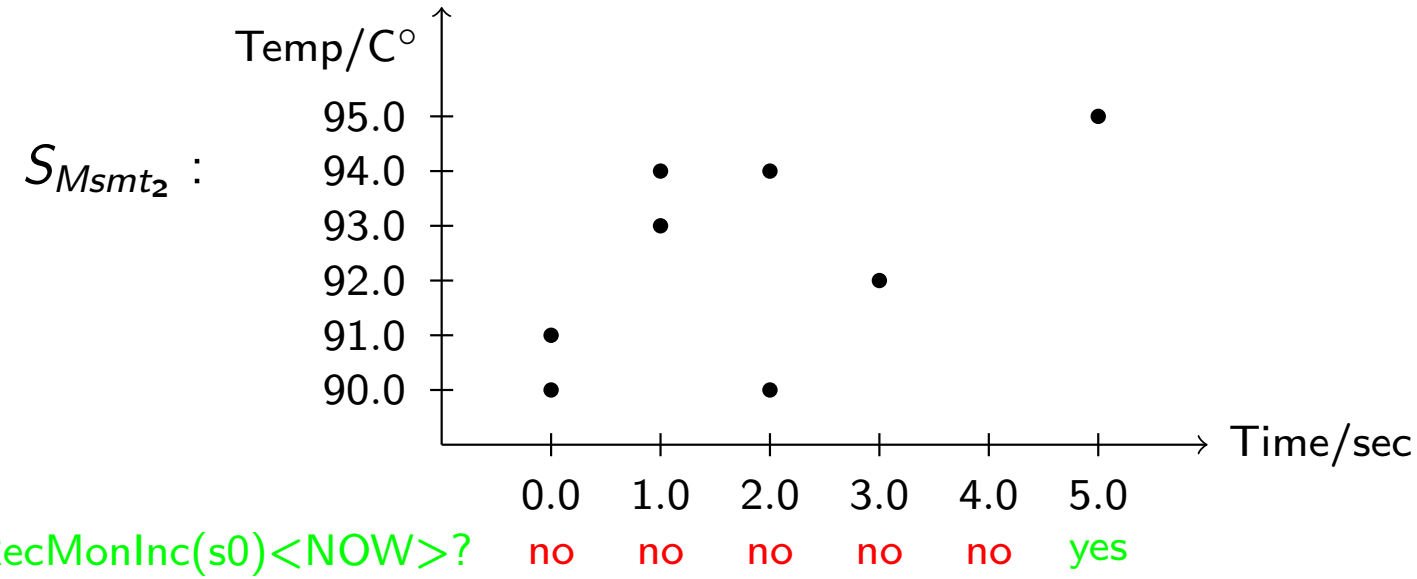
Information Need for Monotonicity (IN-Mon)

Tell every 1s whether the temperature in sensor *s0* increased monotonically in the last 2s in stream S_Msmt and whether the value is functional.

```
CREATE STREAM S_out_1 AS
CONSTRUCT {s0 rdf:type RecMonInc}<NOW>
FROM S_Msmt [NOW-2s, NOW] -> 1s
SEQUENCE BY StdSeq AS SEQ1
HAVING FORALL i <= j IN SEQ1,?x,?y:
       IF {s0 :val ?x}<i>  AND {s0 :val ?y}<j>
       THEN ?x <= ?y
```

- Check for $i = j$ means checking of "functionality" of *val* in ABox $i = j$
- then "monotonicity" in the usual sense on non-empty bindings (which must be unique at every time point)

Monotonicity Variant



Pulse Declarations

- STARQL uses window operator as in CQL
- but mitigates CQLs "problems" with continuous time flows on the query language level
- STARQL uses pulse declaration for well-defined output stream with `CREATE PULSE`
- Pulse synchronizes multiples streams
- Pulse defines output times

Operational Semantics

Example template for operational semantics

```
CREATE STREAM S_out
CREATE PULSE START = st, FREQUENCY = fr
...
FROM S_MSMt [NOW-wr, NOW] -> sl
...
```

- Pulse time vs. stream time
- Pulse time t_{pulse} regular according to FREQUENCY
  $t_{pulse} = st \longrightarrow st + fr \longrightarrow st + 2fr \longrightarrow \ldots$
- Stream time t_{str} determined by trace of endpoint of sliding window
- Stream time jumping/sliding

How Streaming Time Evolves I

Example template for operational semantics

```
CREATE STREAM S_out
CREATE PULSE START = st, FREQUENCY = fr
...
FROM S_MSMt [NOW-wr, NOW] -> sl
...
```

- Evolvement of t_{str}:

$$t_{str} \xrightarrow{\text{IF } t_{str} + m \times sl \leq t_{pulse} \text{ (for } m \in \mathbb{N} \text{ maximal)}} t_{str} + m \times sl$$

- Window contents at t_{pulse}:
  $\{ax\langle t\rangle \in S_{Msmt} \mid t_{str} - wr \leq t \leq t_{str}\}$
- Always $t_{str} \leq t_{pulse}$.

How Streaming Time Evolves II

Instantiation of example template

```
CREATE STREAM S_out
CREATE PULSE START = 0s, FREQUENCY = 2s
...
FROM S_MSMt [NOW-3s, NOW] -> 3s
...
```

$$t_{pulse}: \quad 0s \rightarrow 2s \rightarrow 4s \rightarrow 6s \rightarrow 8s \rightarrow 10s \rightarrow 12s \rightarrow$$

$$t_{str}: \quad 0s \rightarrow 0s \rightarrow 3s \rightarrow 6s \rightarrow 6s \rightarrow 9s \rightarrow 12s \rightarrow$$

Multiple Streams

Example for multiple streams

```
CREAT STREAM Sout AS
PULSE START = 0s, FREQUENCY = 2s
CONSTRUCT  ?sens rdf:type RecentMonInc <NOW>
FROM   S_Msmt_1 0s<-[NOW-3s, NOW]->3s,
       S_Msmt_2 0s<-[NOW-3s, NOW]->2s
SEQUENCE BY StdSeq AS SEQ
HAVING (...)
```

$$t_{pulse}: \quad 0s \rightarrow 2s \rightarrow 4s \rightarrow 6s \rightarrow 8s \rightarrow 10s \rightarrow 12s \rightarrow$$

$$t_{S_{Msmt_1}}: \quad 0s \rightarrow 0s \rightarrow 3s \rightarrow 6s \rightarrow 6s \rightarrow 9s \rightarrow 12s \rightarrow$$

$$t_{S_{Msmt_2}}: \quad 0s \rightarrow 2s \rightarrow 4s \rightarrow 6s \rightarrow 8s \rightarrow 10s \rightarrow 12s \rightarrow$$

Reasoning w.r.t. TBox and Static ABox

Optique

Extended Information Need (IN-Emon)

Tell every 1s whether the temperature in all temperature sensors increased monotonically in the last 2s in stream S_Msmt

```
CREATE STREAM S_out_2 AS
PULSE START = 0s, FREQUENCY = 1s
SELECT  { ?s rdf:type RecentMonInc }<NOW>
FROM S_Msmt [NOW-2s, NOW]->1s
USING STATIC ABOX <http://Astatic>,
      TBOX <http://TBox>
WHERE { ?s rdf:type TempSens }
SEQUENCE BY StdSeq AS SEQ
HAVING
 FORALL i < j IN SEQ,?x,?y:
 IF ({ ?s val ?x }<i>  AND { ?s  val ?y }<j>)
 THEN ?x <= ?y
```

Reasoning

- TBox $\mathcal{T}$
  - No temporal constructors
  - Example: $\mathit{BurnerTipTempSensor} \sqsubseteq \mathit{TempSens}$
    "At every time point: a burner tip temperature sensor is a temperature sensor"
- Static ABox $\mathcal{A}_{st}$
  - Assertions assumed not to change in time
  - i.e., to hold at every time point
  - Example: $\mathit{BurnerTipTempSens}(s0)$, $\mathit{hasComponent}(\mathit{turb}, s1)$

WHERE Clause

Extended Information Need (IN-Emon)

```
...
USING STATIC ABOX <http://Astatic>,
      TBOX <http://TBox>
WHERE { ?s rdf:type TempSens }
SEQUENCE BY StdSeq AS SEQ
HAVING
 FORALL i < j IN SEQ,?x,?y:
 IF { ?s val ?x }<i>  AND { ?s  val ?y }<j>
 THEN ?x <= ?y
```

- Answering WHERE clause by certain answer semantics
  - $\psi_{WHERE}(?s) = TempSens(?s)$
  - $cert(\psi_{WHERE}, \mathcal{T} \cup \mathcal{A}_{st})$
  - Example: Captures also BurnerTipTempSensors
- Gives preselection of constants for instantiation in HAVING clause

Semantics of HAVING Clause

Extended Information Need (IN-Emon)

```
...
HAVING
 FORALL i < j IN SEQ,?x,?y:
 IF { ?s val ?x }<i>  AND { ?s  val ?y }<j>
 THEN ?x <= ?y
```

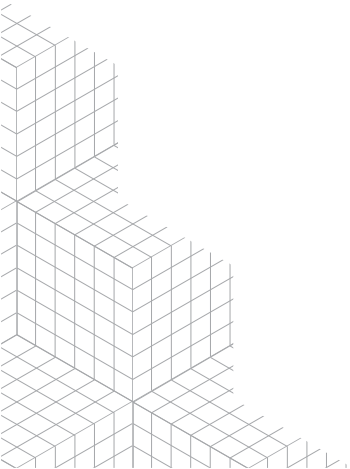
- In original STARQL semantics $\langle i \rangle$ is interpreted as epistemic operator
- Motivated by framework approach
- $val(?s, ?x)\langle i \rangle$ holds if it is provably the case in ith ABox that $val(?x, ?y)$
- Note the different uses of $\langle \cdot \rangle$
- $cert(val(?s, ?x), \mathcal{A}_i \cup \mathcal{T} \cup \mathcal{A}_{st})$

Rewritability of HAVING Clauses

- Rewritability of HAVING clause becomes almost trivial for epistemic semantics
  - One perfectly rewrites embedded queries in state indexed atoms w.r.t. $\mathcal{T}$
  - Resulting HAVING clause can be formulated in FOL with $<, +$
- Example
  - HAVING clause
    `...EXISTS i {?s val ?x} <i> ...`
  - TBox axiom: $\mathit{tempVal} \sqsubseteq \mathit{val} \in \mathcal{T}$
  - Rewritten HAVING clause
    `... EXISTS i ({?s val ?x} UNION {?s tempVal ?x})<i>`
    ...
- Works only for $\mathcal{T}$ without temporal operators

Rewritability of HAVING Clauses

- Non-epistemic semantics of $\langle i \rangle$
  - Read $\langle i \rangle$ not as operator but as state-index attachment
  - $val(s, x)\langle i \rangle$ read as $val(s, x, i)$
- Same rewriting as for epistemic semantics works for some fragment of HAVING clauses
  - No negation
  - No FORALL over domain variables



Inconsistency Handling

- Want to express that at every time point a sensor has at most one value
- Non-reified view with classical TBox $\mathcal{T}$: $(\mathit{func}\ \mathit{val}) \in \mathcal{T}$
- No home-made inconsistencies in STARQL window semantics
  - Window operator conserves timestamps

Time	Window contents before sequencing
...	...
$5s$	$\mathit{val}(s_0, 92°)\langle 3s\rangle,\ \mathit{val}(s_0, 95°)\langle 5s\rangle$

  - Otherwise we could have: $\mathit{val}(s_0, 90), \mathit{val}(s_0, 91)$
  - This was the reason to change CQL window semantics to STARQL window semantic
- In reified view no similar problem with window semantics
- But more difficult to express functionality
  "There are no two measurements having the same sensor but different times"

Inconsistency Handling

- Want to express that at every time point a sensor has at most one value
- Non-reified view with classical TBox $\mathcal{T}$: $(\mathit{func}\ \mathit{val}) \in \mathcal{T}$
- No home-made inconsistencies in STARQL window semantics
  - Window operator conserves timestamps

Time	Window contents before sequencing
...	...
$5s$	$\mathit{val}(s_0, 92°)\langle 3s\rangle,\ \ \mathit{val}(s_0, 95°)\langle 5s\rangle$

  - Otherwise we could have: $\mathit{val}(s_0, 90), \mathit{val}(s_0, 91)$
  - This was the reason to change CQL window semantics to STARQL window semantic
- In reified view no similar problem with window semantics
- But more difficult to express functionality
  "There are no two measurements having the same sensor but different times"

Non-Standard Sequencing Again

- $S_{Msmt} = \{\ldots val(s0, 90)\langle 3s\rangle, val(s0, 95)\langle 3s\rangle \ldots\}$
- With standard sequencing leads to an ABox not consistent with (*func val*)
- Can test for inconsistencies by FOL query (Consistency is FOL rewritable for DL-Lite)
- How to handle inconsistent ABoxes?
  1. Use repair semantics (not classical certain answer semantics) (perhaps in the future)
  2. Use non-standard sequencing eliminating non-consistent ABoxes

Detecting Inter-Temporal Inconsistencies Optique

- Remember: No temporal operators in $\mathcal{T}$

$$\exists tempVal \sqsubseteq TempSens,$$
$$\exists pressVal \sqsubseteq PressSens$$
$$TempSens \sqsubseteq \neg PressSens$$

- $S_{Msmt} = \{\ldots tempVal(s0, 90)\langle 3s\rangle, pressVal(s0, 70)\langle 4s\rangle \ldots\}$
- Intuitively: Information regarding $s0$ not consistent
- Not detected if s_0 not classified in static ABox
- Reasoning: Cannot express rigidity on sensor concepts

Querying Historical Data

- Different approaches to handle historical data in STARQL
  1. Put slide = 0 and fix window ends
  2. Stream historical data according to timestamps

Example solution 1

Return all sensor values of a specific sensor $s0$ within a specific time interval $[0s, 60s]$

```
CREATE GRAPH Solution-One AS
CONSTRUCT   { s0 :val ?x }
FROM S_Msmt[0s, 60s]->0s
USING  STATIC ABOX <http://ABox>,
       TBOX <http://TBox>
SEQUENCE BY StdSeq AS SEQ
HAVING EXISTS i  { s0 :val ?x } <i>
```

Querying Historical Data

Example solution 2

Return all sensor values of a specific sensor $s0$ within a specific time interval $[0s, 60s]$

```
CREATE STREAM Solution-TWO AS
CREATE PULSE AS
       START = 0s, FREQUENCY = 1s, END = 60s
CONSTRUCT   { s0 :val ?x }<NOW>
FROM S_Msmt[NOW, NOW]->1s
USING  STATIC ABOX <http://ABox>,
       TBOX <http://TBox>
SEQUENCE BY StdSeq AS SEQ
HAVING EXISTS i  { s0 :val ?x } <i>
```

Window Semantics Again

- No difference whether S_Msmt is real-time stream or streamed historical data
- Due to $t_{str} \leq t_{pulse}$
- Assume otherwise ($t_{str} > t_{pulse}$)
  - Historical query: window may contain future elements from $[t_{pulse}, t_{str}]$
  - Stream query: window cannot contain future elements from $[t_{pulse}, t_{str}]$

Mapping Temporal and Streaming Data

Optique

- Mapping historical data

$m1 : val(x, y)\langle z \rangle \longleftarrow$

```
SELECT f(SID) AS x, Mval AS y, MtimeStamp AS z
FROM MEASUREMENT-TABLE
```

  - $\mathcal{A}(m1, DB)$ is a temporal ABox
  - where MEASUREMENT-TABLE in DB
- Mapping streams

$m2 : val(x, y)\langle z \rangle \longleftarrow$

```
SELECT Rstream(f(SID) AS x, Mval AS y,
                MtimeStamp AS z)
FROM MEASUREMENT-REL-STREAM
```

  - $\mathcal{A}(m2, Str - DB)$ is a stream of timestamped ABox assertions
  - where MEASUREMENT-REL-STREAM in Str-DB
- Convention: Ontology layer streams are specified by one mapping

Mapping Temporal and Streaming Data

Optique

- Mapping historical data

$m1 : val(x, y)\langle z\rangle \longleftarrow$

```
SELECT f(SID) AS x, Mval AS y, MtimeStamp AS z
FROM MEASUREMENT-TABLE
```

  - $\mathcal{A}(m1, DB)$ is a temporal ABox
  - where `MEASUREMENT-TABLE` in DB
- Mapping streams

$m2 : val(x, y)\langle z\rangle \longleftarrow$

```
SELECT Rstream(f(SID) AS x, Mval AS y,
                MtimeStamp AS z)
FROM MEASUREMENT-REL-STREAM
```

  - $\mathcal{A}(m2, Str - DB)$ is a stream of timestamped ABox assertions
  - where `MEASUREMENT-REL-STREAM` in Str-DB
- Convention: Ontology layer streams are specified by one mapping

Challenges of Unfolding

- `HAVING` clause language refers to state tagged not time stamped assertions
- Solution
  - Use simple sequencing mechanisms such as standard sequencing
  - Keep track of time window processing by stream of SEQ-entries
- `HAVING` clause language uses domain calculus, CQL tuple calculus
- Solution: Use safety mechanisms by adornments for variables to guarantee domain independence
- CQL looses timestamps in window contents
- Solution: Assume Stream-To-Stream Operator duplicating timestamps as time attributes: $d\langle t\rangle \mapsto (d,t)\langle t\rangle$

Challenges of Unfolding

- `HAVING` clause language refers to state tagged not time stamped assertions
- Solution
  - Use simple sequencing mechanisms such as standard sequencing
  - Keep track of time window processing by stream of SEQ-entries
- `HAVING` clause language uses domain calculus, CQL tuple calculus
- Solution: Use safety mechanisms by adornments for variables to guarantee domain independence
- CQL looses timestamps in window contents
- Solution: Assume Stream-To-Stream Operator duplicating timestamps as time attributes: $d\langle t\rangle \mapsto (d,t)\langle t\rangle$

Challenges of Unfolding

- `HAVING` clause language refers to state tagged not time stamped assertions
- Solution
  - Use simple sequencing mechanisms such as standard sequencing
  - Keep track of time window processing by stream of SEQ-entries
- `HAVING` clause language uses domain calculus, CQL tuple calculus
- Solution: Use safety mechanisms by adornments for variables to guarantee domain independence
- CQL looses timestamps in window contents
- Solution: Assume Stream-To-Stream Operator duplicating timestamps as time attributes: $d\langle t\rangle \mapsto (d,t)\langle t\rangle$

Safety Mechanism

- `HAVING` clause $?y > 3$ is not safe: Infinite binding set for $?y$
- $val(s_0, ?y)\langle i \rangle \wedge (?y > 3)$ is safe
- Adornments for variables ensure not only finiteness but domain independence
- Domain independence
  - Query answer depends only on the interpretations of the predicates mentioned in the query or the DB but not the domain
  - A query ϕ is *domain independent* iff for all interpretations $\mathcal{I}, \mathcal{J}$ such that $\mathcal{I}$ is a substructure of $\mathcal{J}$: $ans(\phi, \mathcal{I}) = ans(\phi, \mathcal{J})$.

Safety Mechanism

- `HAVING` clause $?y > 3$ is not safe: Infinite binding set for $?y$
- $val(s_0, ?y)\langle i \rangle \wedge (?y > 3)$ is safe
- Adornments for variables ensure not only finiteness but domain independence
- Domain independence
  - Query answer depends only on the interpretations of the predicates mentioned in the query or the DB but not the domain
  - A query ϕ is *domain independent* iff for all interpretations $\mathcal{I}, \mathcal{J}$ such that $\mathcal{I}$ is a substructure of $\mathcal{J}$: $ans(\phi, \mathcal{I}) = ans(\phi, \mathcal{J})$.

Domain Independence

- Counterexample
  - $\phi(x, y) = A(x) \vee B(y)$
  - $\mathcal{I} = (\{\alpha\}, (\cdot)^{\mathcal{I}})$, $\mathcal{J} = (\{\alpha, \beta\}, (\cdot)^{\mathcal{J}})$
  - $A^{\mathcal{I}} = A^{\mathcal{J}} = \{\alpha\}$
  - $B^{\mathcal{I}} = B^{\mathcal{J}} = \emptyset$:
  - $(\alpha, \beta) \in \mathit{ans}(\phi, \mathcal{J})$ but $(\alpha, \beta) \notin \mathit{ans}(\phi, \mathcal{I})$.
- Arbitrary use of disjunction has strange consequences for answering on DB
  - $\psi(?x, ?y) = \mathit{TempSens}(?x) \vee \mathit{PressureSens}(?y)$
  - gives finite set of bindings but is not domain independent

DB: TempSens	PressureSens
a_1	b_1

  - $\mathit{ans}(\psi(?x, ?y), DB) = \{(a_1, b_1), (a_1, a_1), (b_1, b_1)\}$

Adornments

- Safety conditions by variable adornments in $\{+, -, --, \emptyset\}$
  - x^{+}: x is safe variable
  - x^{-}: x is non-safe variable (but may become safe by negation)
  - x^{--}: x is non-safe variable
  - $x^{\emptyset}$: x does not occur in other formula
- Allowed adornment combinations fixed by table
- Grammar rules have form

$$hCl(\vec{z}^{\vec{g}^1 \vee \vec{g}^2}) \longrightarrow hCl(\vec{z}^{\vec{g}^1}) \texttt{ OR } hCl(\vec{z}^{\vec{g}^2})$$

g_1	g_2	$g_1 \vee g_2$	$\ldots$
$--$	$--$	$--$	$\ldots$
$--$	$-$	$-$	
$--$	$+$	$--$	
$--$	$\emptyset$	$--$	
$-$	$--$	$-$	
$-$	$-$	$-$	
$-$	$+$	$-$	
$-$	$\emptyset$	$-$	
$+$	$--$	$--$	
$+$	$-$	$-$	
$+$	$+$	$+$	
$+$	$\emptyset$	$--$	
$\emptyset$	$--$	$--$	
$\emptyset$	$-$	$-$	
$\emptyset$	$+$	$--$	
$\emptyset$	$\emptyset$	$\emptyset$	

Adornments

- Grammar rules have form

$$hCl(\vec{z}^{\vec{g}^1 \vee \vec{g}^2}) \longrightarrow hCl(\vec{z}^{\vec{g}^1}) \texttt{ OR } hCl(\vec{z}^{\vec{g}^2})$$

- Example

$$F(x_1^{--}, x_2^{+}, x_3^{-}) \longrightarrow$$
$$F_1(x_1^{--}, x_2^{+}, x_3^{-}) \texttt{ OR } F_2(x_1^{+}, x_2^{+}, x_3^{\emptyset})$$

g_1	g_2	$g_1 \vee g_2$	$\ldots$
$--$	$--$	$--$	$\ldots$
$--$	$-$	$-$	
$--$	$+$	$--$	
$--$	$\emptyset$	$--$	
$-$	$--$	$-$	
$-$	$-$	$-$	
$-$	$+$	$-$	
$-$	$\emptyset$	$-$	
$+$	$--$	$--$	
$+$	$-$	$-$	
$+$	$+$	$+$	
$+$	$\emptyset$	$--$	
$\emptyset$	$--$	$--$	
$\emptyset$	$-$	$-$	
$\emptyset$	$+$	$--$	
$\emptyset$	$\emptyset$	$\emptyset$	

Safety in Monotonicity Condition

Extended Information Need (IN-Emon)

```
HAVING
 FORALL i < j IN SEQ,?x,?y:
 IF { ?s val ?x }<i>  AND { ?s  val ?y }<j>
 THEN ?x <= ?y
```

- Unsafe variables in `?x <= ?y` ...
- ... are bound by antecedens of all quantifier

Transformation into SQL

- Safety mechanism guarantees: `HAVING` clauses transformable into formulas in safe range normal form (SRNF)
- Folklore theorem: SRNF is domain independent
- Transformation into SQL
  - Rewrite `FORALL` with `NOT EXISTS NOT`
  - Push `NOT` inwards (stopping at `EXISTS`) ...

Transformation Example

```
FORALL i < j IN SEQ,?x,?y:
IF { ?s val ?x }<i>  AND { ?s  val ?y }<j>
THEN ?x <= ?y

;; ==== transformed to  ====>

 NOT EXISTS i,j in SEQ, x,y:
 i < j AND { ?s val ?x }<i>  AND { ?s  val ?y }<j>
 AND x > y
```

Unfolded Monotonicity Query

Optique

```
CREATE STREAM S_out_1 AS
PULSE START = 0s, FREQUENCY = 1s
CONSTRUCT {s0 rdf:type RecMonInc}<NOW>
FROM S_Msmt [NOW-2s, NOW]->1s
SEQUENCE BY StdSeq AS SEQ1
HAVING FORALL i < j IN SEQ1,?x,?y:
       IF {s0 val ?x}<i>  AND {s0  val ?y}<j>
       THEN ?x <= ?y
```

```
CREATE VIEW windowRel as
SELECT *  FROM REL-STREAM-MEASUREMENT[RANGE 2s Slide 1s];

SELECT  Rstream(' s0 rdf:Type RecMonInc '||'<'||timestamp||'>')
FROM    windowRel
WHERE   windowRel.SID = 'TC255'  AND
        NOT EXISTS (
             SELECT * FROM
             (SELECT timestamp as i, value as x FROM windowRel),
             (SELECT timestamp as j, value as y FROM windowRel)
             WHERE   i < j AND x > y );
```

STARQL prototype(s)

- Optique prototype
  - Uses stream extended version of ADP
    - Highly distributable DBMS
    - Extend SQL-Lite with window operators (as in CQL)
  - Mapping handling using ontop and hardcoded timestamp hook mechanism
  - Handles multiple streams
  - Nested queries
- Prolog prototype
  - Translates STARQL queries into safe non-recursive datalog with negation
  - Uses mappings to SQL
  - Stream handling to be implemented
- LISP prototype
  - ABDEO approach
  - Uses materialization

Take-Home Message

Research on temporalizing and streamifying OBDA is a new exciting research area that still calls for robust engines allowing to handle OBDA in a broader sense.