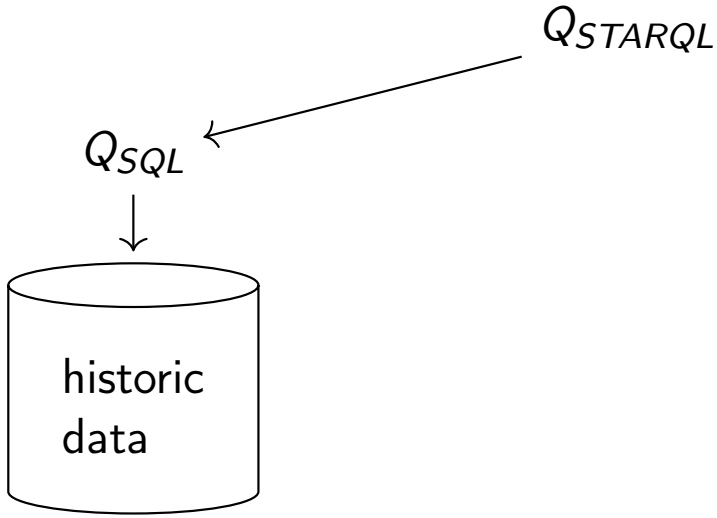
UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR INFORMATIONSSYSTEME

Simon Schiff, Özgür L. Özçep

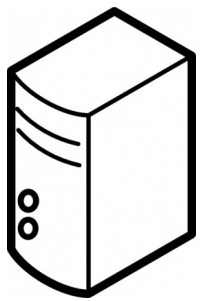
Exploiting Back-End APIs for Feasible Ontology-Based Stream Access

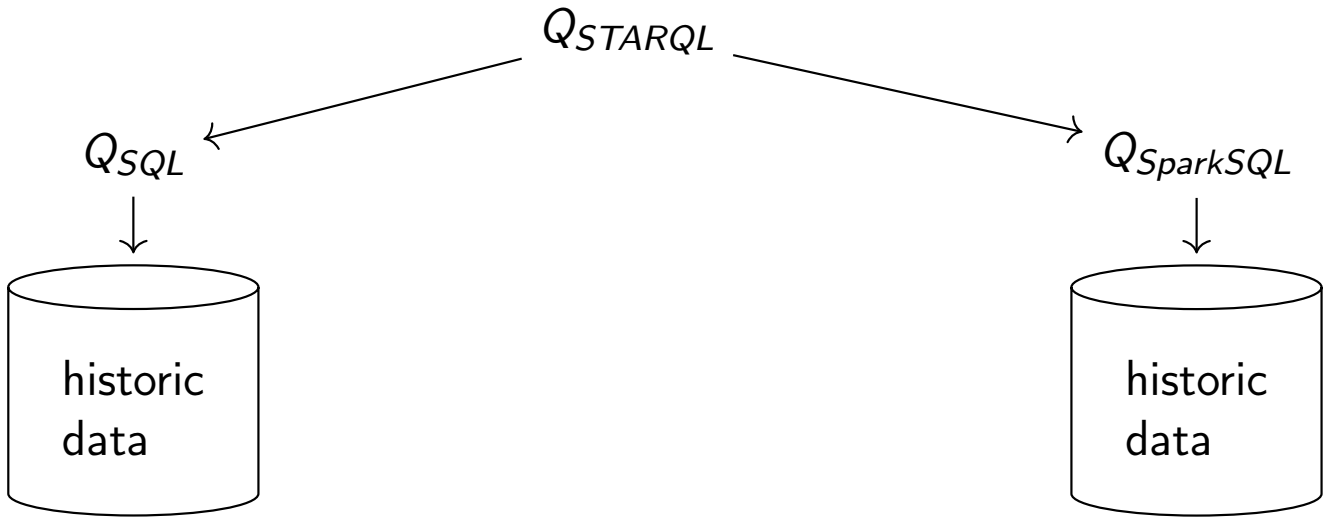
Fourth Stream Reasoning Workshop, Linköping, 17th April

Institute of Information Systems
University of Lübeck



OBDA on huge datasets
(w/o optimization)
$\Rightarrow$ Long processing times

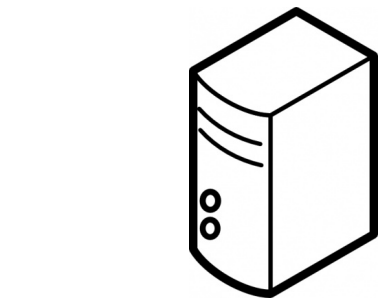


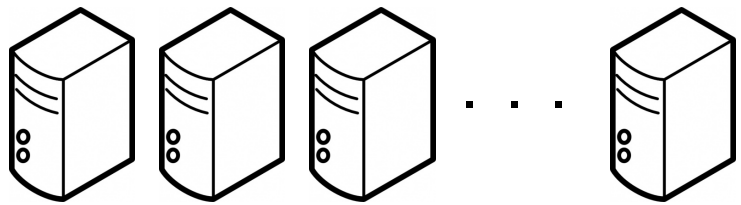


OBDA on huge datasets
(w/o optimization)
⇒ Long processing times

Idea

Processing times are reducible using additional hardware and parallelisation
⇒ Short processing times

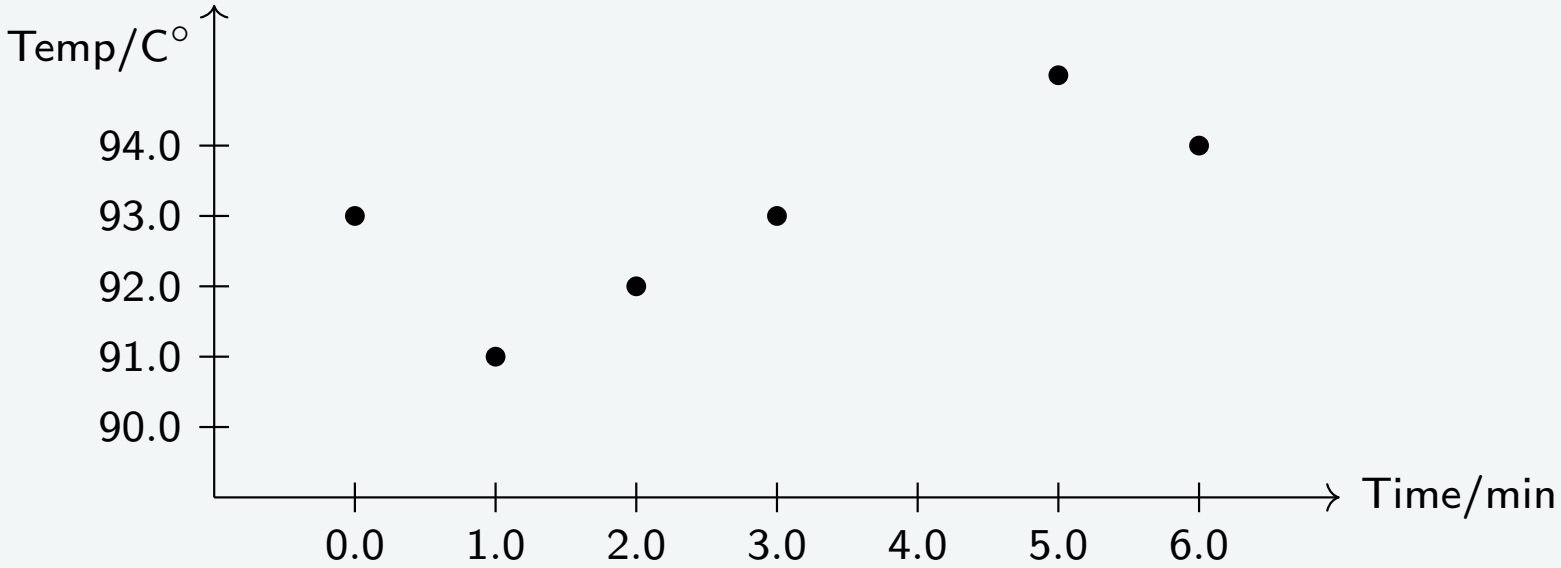




Horizontal scaling

STARQL Query Example

Measurements

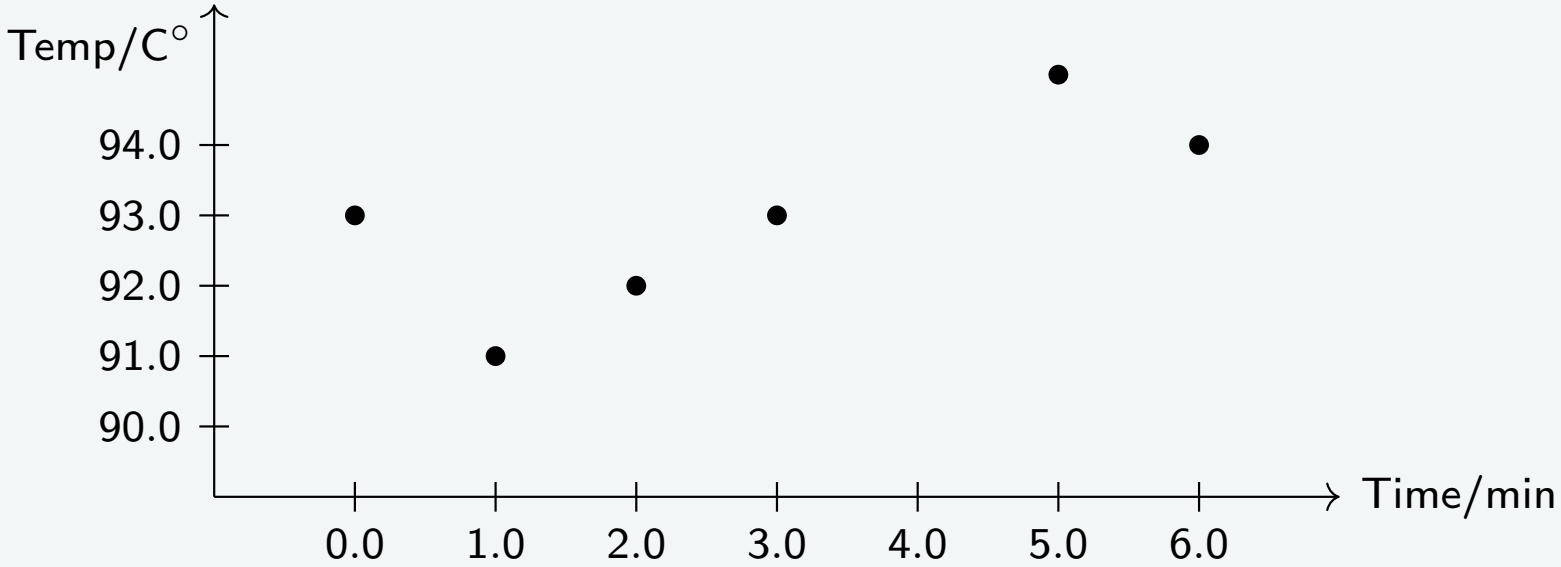


Information need for monotonicity

Tell every minute whether the temperature measured by a sensor increased monotonically in the last 5 minutes.

STARQL Query Example

Measurements

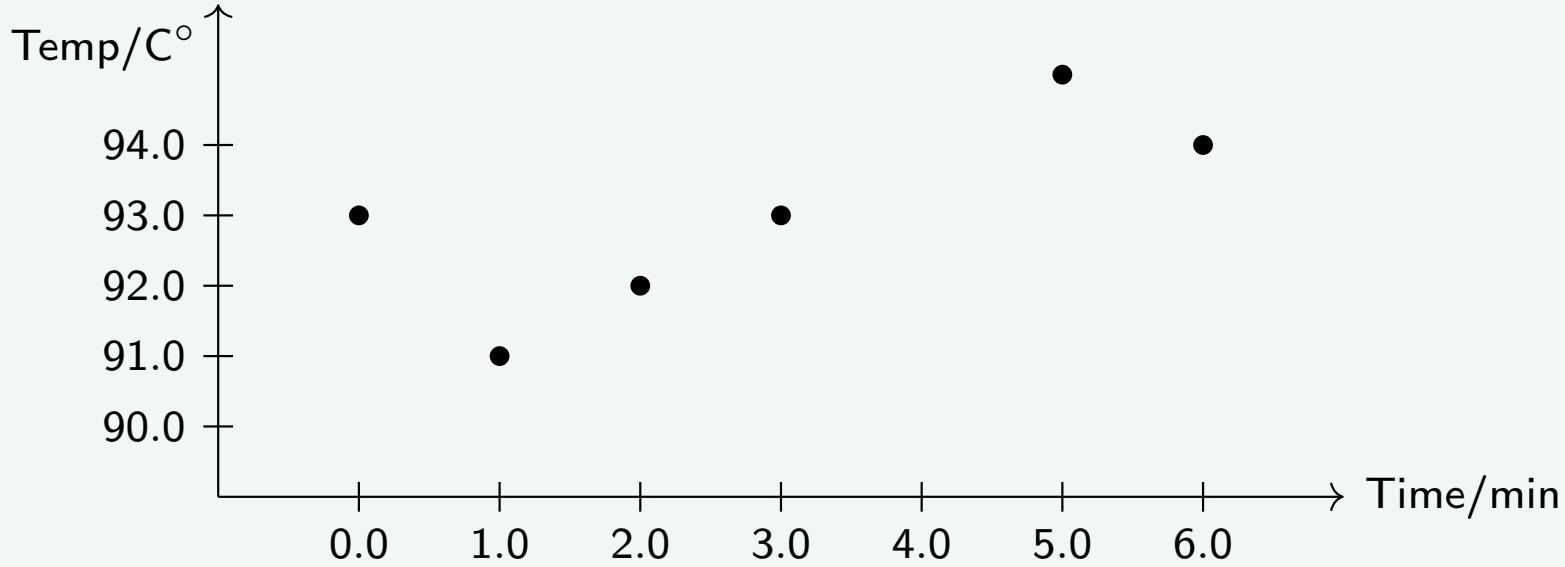


STARQL Representation of monotonicity

```
SELECT x
FROM measurements [NOW - PT5M, NOW] -> PT1M
WHERE Sensor(x)
HAVING FORALL t_i, t_j, y1, y2
 IF hasVal(x,y1)<t_i> AND hasVal(x,y2)<t_j>
 AND t_i < t_j THEN y1 <= y2
```

STARQL Query Example

Measurements

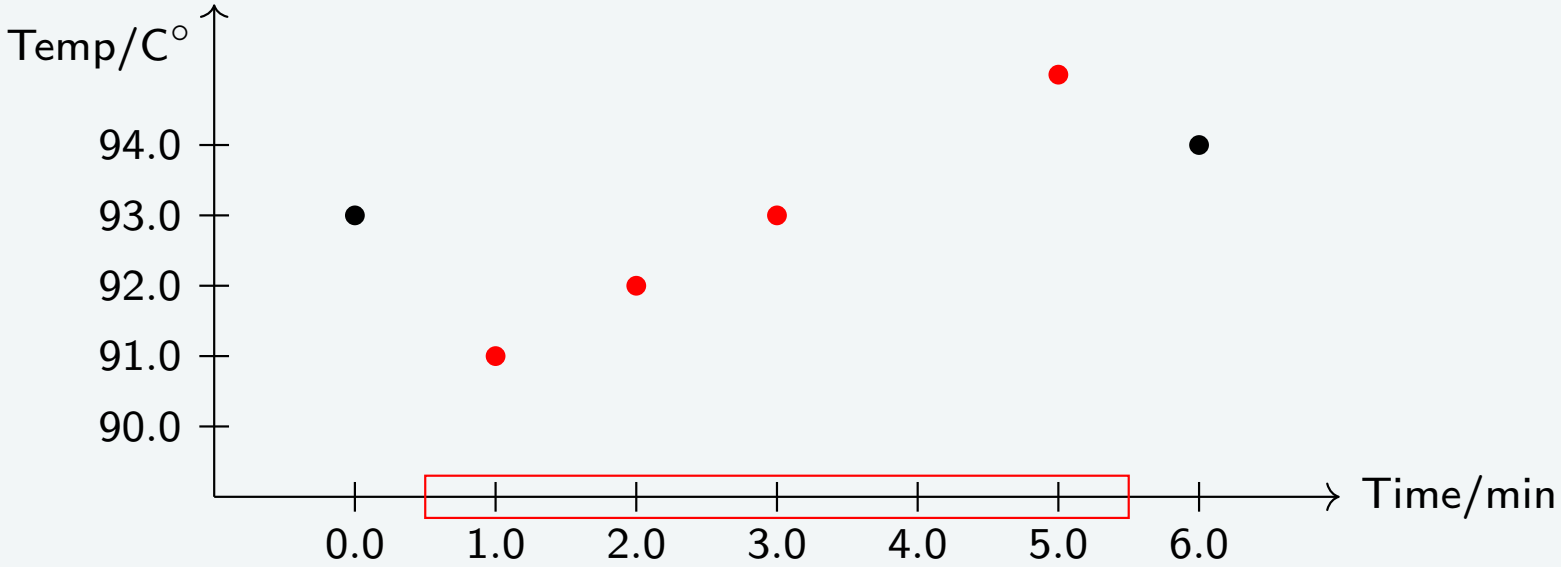


STARQL Representation of monotonicity

```
SELECT x
FROM measurements [NOW - PT5M, NOW] -> PT1M
WHERE Sensor(x)
HAVING FORALL t_i, t_j, y1, y2
 IF hasVal(x,y1)<t_i> AND hasVal(x,y2)<t_j>
 AND t_i < t_j THEN y1 <= y2
```

STARQL Query Example

Measurements

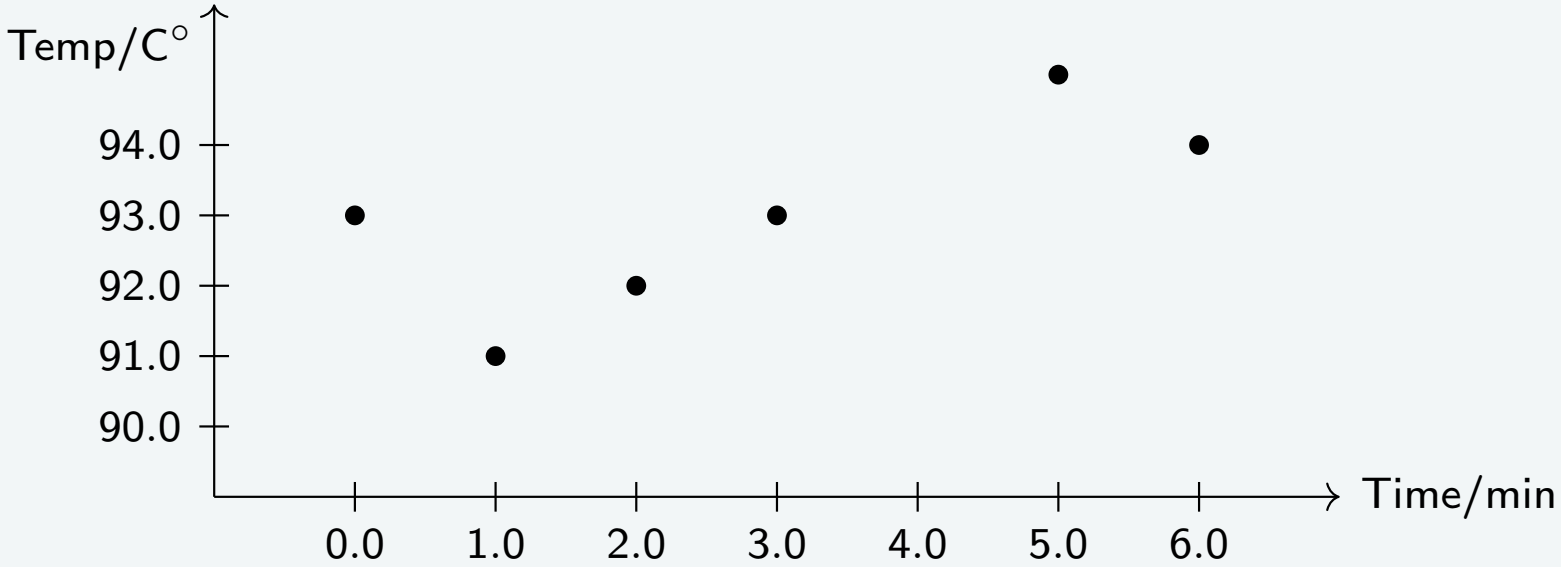


STARQL Representation of monotonicity

```
SELECT x
FROM measurements [NOW - PT5M, NOW] -> PT1M
WHERE Sensor(x)
HAVING FORALL t_i, t_j, y1, y2
 IF hasVal(x,y1)<t_i> AND hasVal(x,y2)<t_j>
 AND t_i < t_j THEN y1 <= y2
```

STARQL Query Example

Measurements

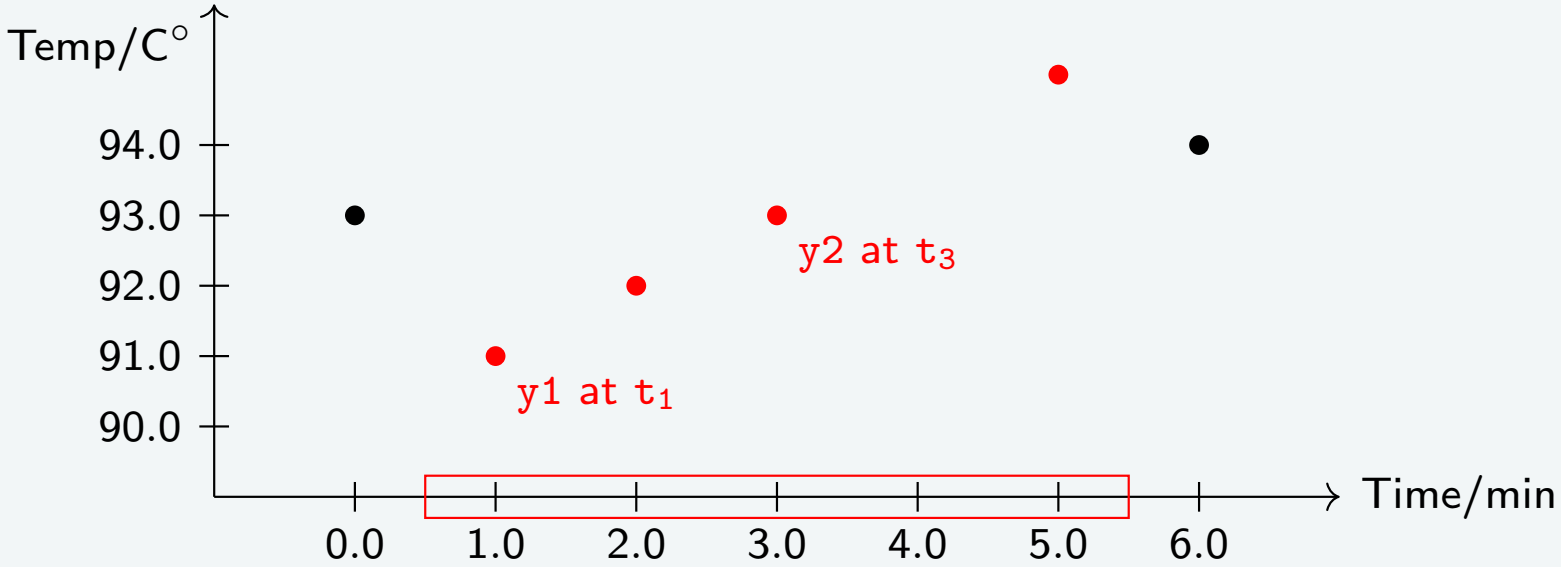


STARQL Representation of monotonicity

```
SELECT x
FROM measurements [NOW - PT5M, NOW] -> PT1M
WHERE Sensor(x)
HAVING FORALL t_i, t_j, y1, y2
 IF hasVal(x,y1)<t_i> AND hasVal(x,y2)<t_j>
 AND t_i < t_j THEN y1 <= y2
```

STARQL Query Example

Measurements



STARQL Representation of monotonicity

```
SELECT x
FROM measurements [NOW - PT5M, NOW] -> PT1M
WHERE Sensor(x)
HAVING FORALL t_i, t_j, y1, y2
 IF hasVal(x,y1)<t_i> AND hasVal(x,y2)<t_j>
 AND t_i < t_j THEN y1 <= y2
```

STARQL Query Example

STARQL Representation of monotonicity

```
SELECT x
FROM measurements [NOW - PT5M, NOW] -> PT1M
WHERE Sensor(x)
HAVING FORALL t_i, t_j, y1, y2
 IF hasVal(x,y1)<t_i> AND hasVal(x,y2)<t_j>
 AND t_i < t_j THEN y1 <= y2
```

The FOL template language is domain independent[1]:

STARQL HAVING clause can be unfolded into languages such as SQL.

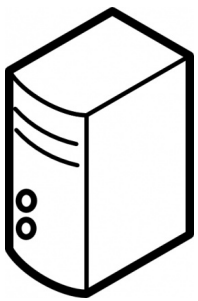
$\Rightarrow$ Process historic (e.g. timestamped datasets)

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. Foundations of Databases: The Logical Level. Addison-Wesley Longman Publishing Co., Inc., 1995.

STARQL Query Example

STARQL Representation of monotonicity

```
SELECT x
FROM measurements [NOW - PT5M, NOW] -> PT1M
WHERE Sensor(x)
HAVING FORALL t_i, t_j, y1, y2
 IF hasVal(x,y1)<t_i> AND hasVal(x,y2)<t_j>
 AND t_i < t_j THEN y1 <= y2
```



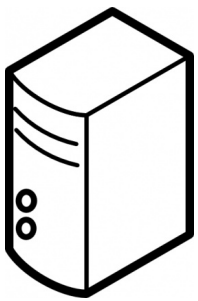
No function exists for executing the unfolded query per window!

- First idea: Create table with window intervals and join with historic dataset.
- Too slow?

STARQL Query Example

STARQL Representation of monotonicity

```
SELECT x
FROM measurements [NOW - PT5M, NOW] -> PT1M
WHERE Sensor(x)
HAVING FORALL t_i, t_j, y1, y2
 IF hasVal(x,y1)<t_i> AND hasVal(x,y2)<t_j>
 AND t_i < t_j THEN y1 <= y2
```



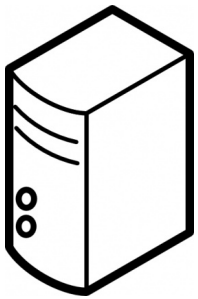
No function exists for executing the unfolded query per window!

- Second idea: Create a function for executing the unfolded query per window using PL/pgSQL.
- Sufficient?

STARQL Query Example

STARQL Representation of monotonicity

```
SELECT x
FROM measurements [NOW - PT5M, NOW] -> PT1M
WHERE Sensor(x)
HAVING FORALL t_i, t_j, y1, y2
 IF hasVal(x,y1)<t_i> AND hasVal(x,y2)<t_j>
 AND t_i < t_j THEN y1 <= y2
```



No function exists for executing the unfolded query per window!

- Second idea: Create a function for executing the unfolded query per window using PL/pgSQL.
- Sufficient?
- 

STARQL Query Example

```
1  CREATE TYPE window_state AS (memory measurements[],
       wid bigint, start timestamp, stop timestamp, pulse
        timestamp);
2
3  CREATE TABLE measurements_data AS SELECT NULL::bigint
       AS wid, NULL::timestamp AS timestamp, NULL::
       integer AS sensor, NULL::numeric(12,3) AS value
       WHERE false;
4
5  CREATE OR REPLACE FUNCTION moving_window(source text,
       pulse interval, range interval, slide interval)
       RETURNS SETOF measurements_data AS $$
6  DECLARE
7   win window_state;
8
9   cnt bigint;
10
11  line_cursor refcursor;
12  line measurements;
13 BEGIN
14  OPEN line_cursor FOR EXECUTE 'SELECT * FROM '  format
        ('%1$s', source)  ' ORDER BY timestamp ASC';
15  FETCH line_cursor INTO line;
16
17  win.start := line.timestamp;
```

STARQL Query Example

```
18  win.stop := line.timestamp + range;
19  win.pulse := line.timestamp;
20
21  WHILE line.timestamp < win.stop LOOP
22   win.memory := array_append(win.memory, line);
23   FETCH line_cursor INTO line;
24  END LOOP;
25
26  win.wid := 0;
27  RETURN QUERY SELECT win.wid, (unnest(win.memory::
        measurements[])).*;
28
29  win.pulse := win.pulse + pulse;
30  WHILE line.timestamp IS NOT NULL LOOP
31   IF win.pulse < win.stop AND win.pulse < win.start +
         slide THEN
32    win.wid := win.wid + 1;
33    RETURN QUERY SELECT win.wid, (unnest(win.memory::
          measurements[])).*;
34    win.pulse := win.pulse + pulse;
35   ELSIF win.pulse >= win.stop AND win.pulse < win.
         start + slide THEN
36    win.pulse := win.pulse + pulse;
37
38    win.start := win.start + slide;
```

STARQL Query Example

```
39      win.stop := win.stop + slide;
40      WHILE line.timestamp < win.stop LOOP
41       win.memory := array_append(win.memory, line);
42       FETCH line_cursor INTO line;
43      END LOOP;
44      cnt := 1;
45      FOR i IN coalesce(array_lower(win.memory, 1), 1) ..
             coalesce(array_upper(win.memory, 1), 1) LOOP
46       IF win.memory[i].timestamp < win.start THEN
47        cnt := cnt + 1;
48       ELSE
49        EXIT;
50       END IF;
51      END LOOP;
52      win.memory := win.memory[cnt:];
53     ELSIF win.pulse >= win.start + slide THEN
54      win.start := win.start + slide;
55      win.stop := win.stop + slide;
56      WHILE line.timestamp < win.stop LOOP
57       win.memory := array_append(win.memory, line);
58       FETCH line_cursor INTO line;
59      END LOOP;
60      cnt := 1;
```

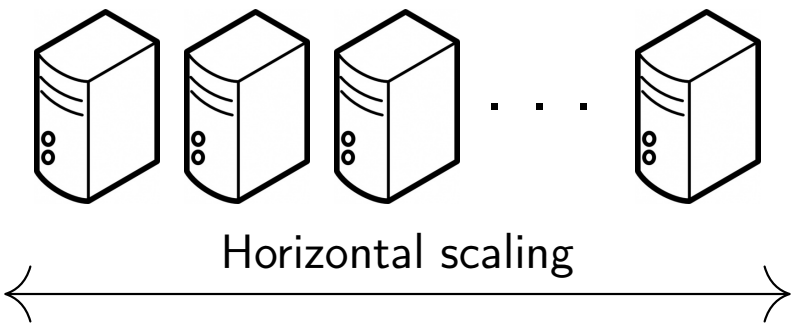
STARQL Query Example

```
61      FOR i IN coalesce(array_lower(win.memory, 1), 1) ..
            coalesce(array_upper(win.memory, 1), 1) LOOP
62       IF win.memory[i].timestamp < win.start THEN
63        cnt := cnt + 1;
64       ELSE
65        EXIT;
66       END IF;
67      END LOOP;
68      win.memory := win.memory[cnt:];
69     END IF;
70    END LOOP;
71    win.wid := win.wid + 1;
72    RETURN QUERY SELECT win.wid, (unnest(win.memory::
          measurements[])).*;
73    CLOSE line_cursor;
74  END
75  $$ language plpgsql;
```

STARQL Query Example

STARQL Representation of monotonicity

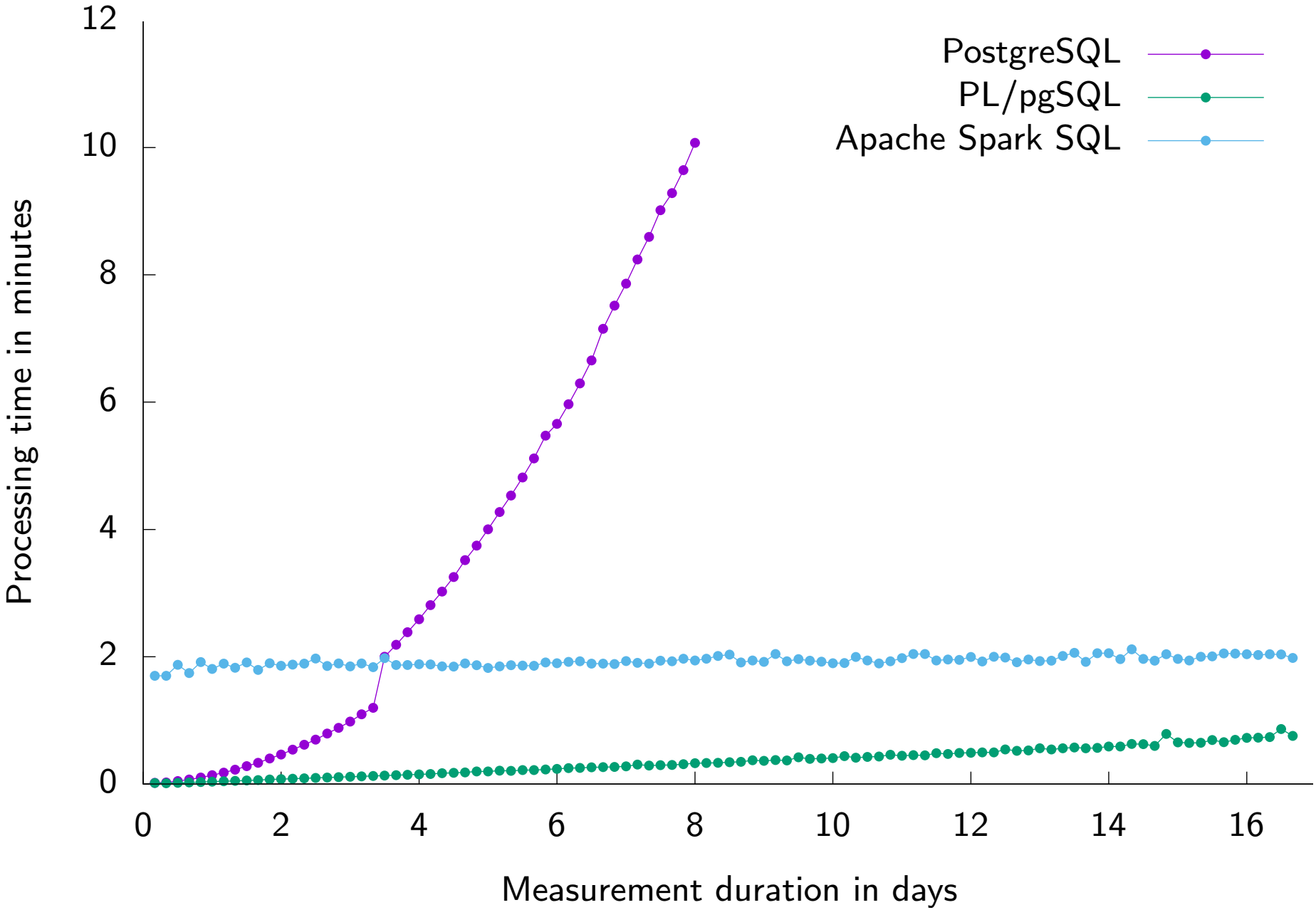
```
SELECT x
FROM measurements [NOW - PT5M, NOW] -> PT1M
WHERE Sensor(x)
HAVING FORALL t_i, t_j, y1, y2
 IF hasVal(x,y1)<t_i> AND hasVal(x,y2)<t_j>
 AND t_i < t_j THEN y1 <= y2
```



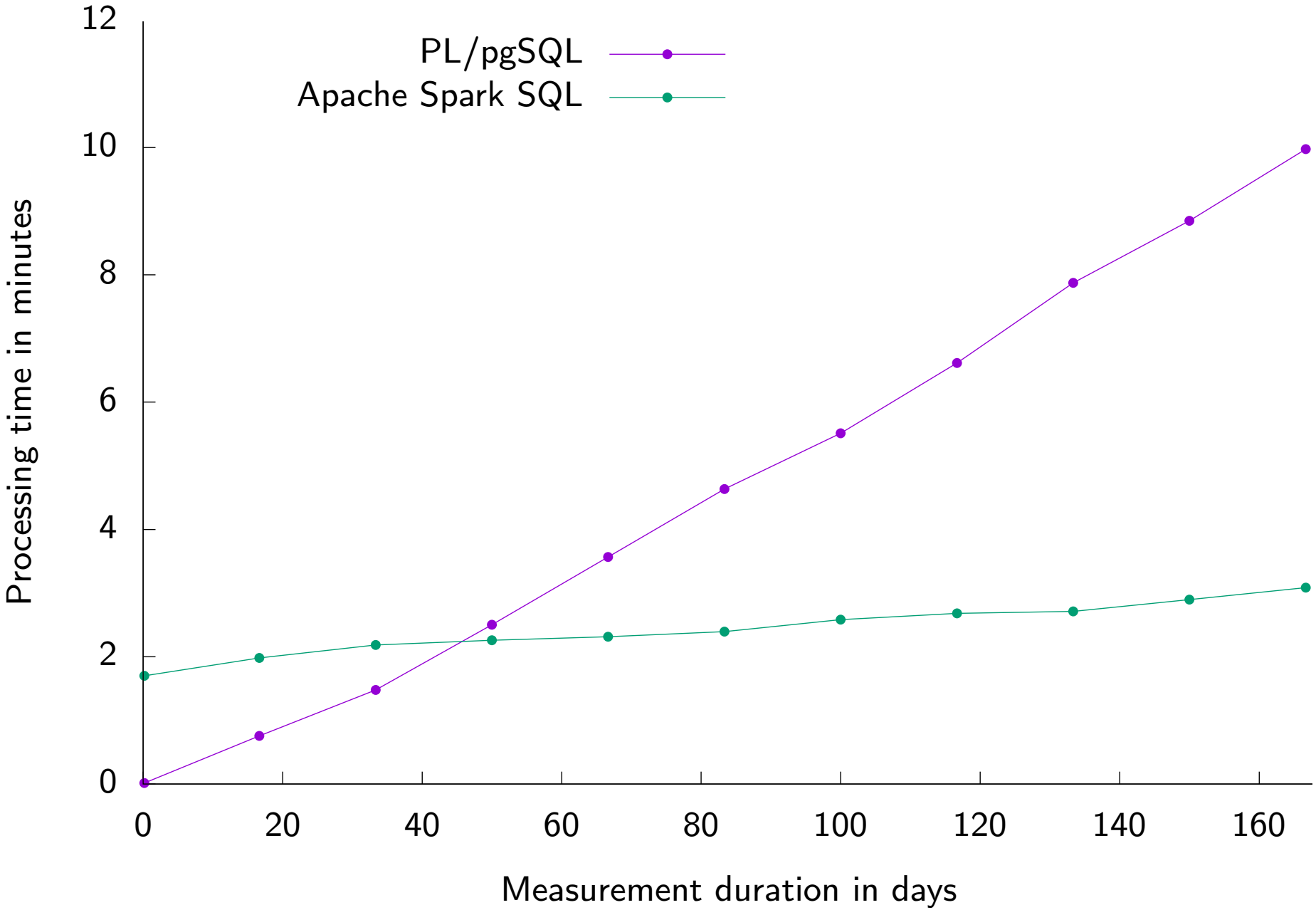
Function exists for executing the unfolded query per window!

- Apache Spark SQL scales horizontally and vertically.
- Scalable?

Processing times using different back ends



Processing times using different back ends



Conclusion and Future Work

Conclusion:

- Window function can be realized by using PL/pgSQL
- Speed gain by using Apache Spark SQL[1]
- Complexity hidden by STARQL

Future Work:

- Incremental stream processing (Not possible for every STARQL query)

[1] Simon Schiff, Özgür L. Özçep, and Ralf Möller. "Ontology-based Data Access to Big Data". In: Open Journal of Databases (OJDB) 6 (1 2018). Postproceeding of Hidest'18, pp. 21–32.

Processing times using distributed Apache Spark SQL