



# Web Stream Processing with RSP4J

**The Web Conf 2022**

**Pieter Bonte (BE🇧🇪) & Matteo Belcao (🇮🇹IT) & Marco Balduini (🇮🇹IT) & Emanuele Della Valle (🇮🇹IT)**

**26/04/2022**

# Agenda
## Processing

- RSP4J, YASPER, & CO
- RSP Internals
  - (and their mapping to the BG)
- Demo and Exercises

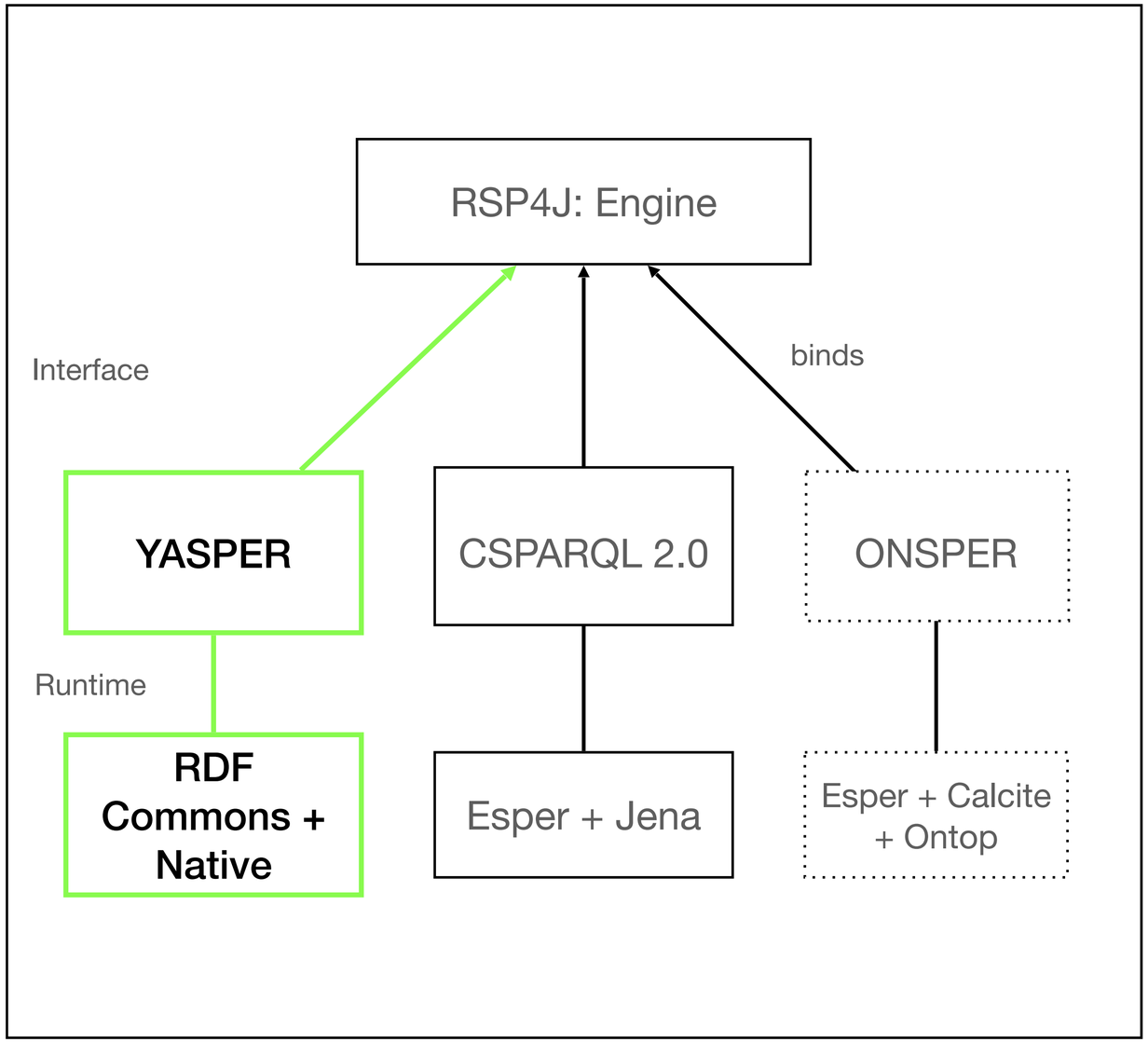
There will be exercises!

# You want to follow the exercises?

- Requirements:
  - Maven
  - Java 9+
- Clone the git repo: https://github.com/pbonte/WSP-TheWebConf2022Tutorial
- Open with your favourite Java IDE (e.g. Eclipse/IntelliJ)

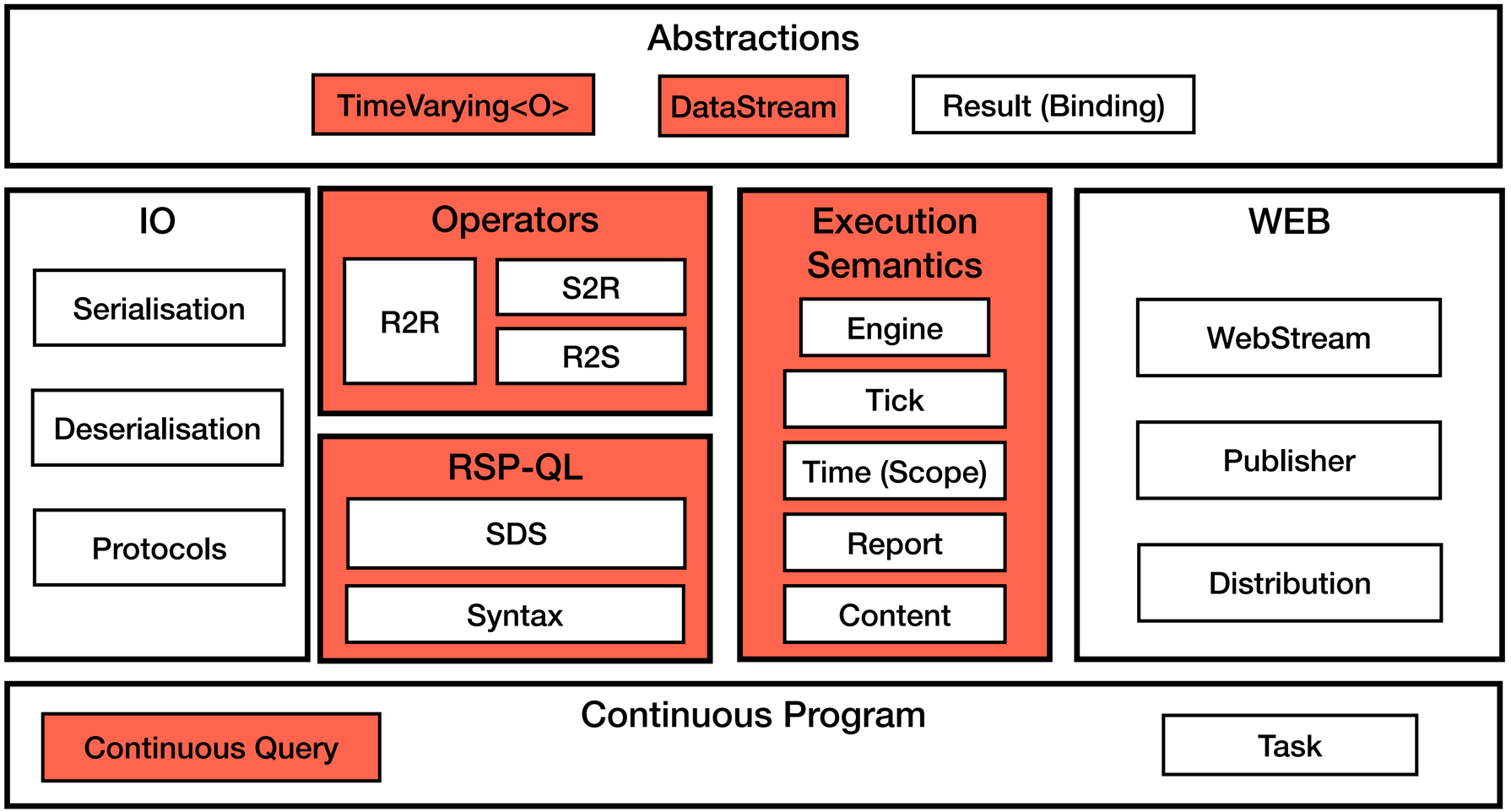
# RSP4J

- Allows controlling the RSP Engine, e.g., stream ingestion, query registration, result stream.
- Can reproduce the execution semantics of common RSP engines
- Includes some bindings, YASPER, CSPARQL 2.0, ONSPER*



# RSP4J

## 30.000ft View: Architecture



# RSP4J

## Abstractions

- Data Streams<T>
  - Generalizes the idea of data stream using Java Generics
- Time-Varying<T>
  - Generalizes the idea of time-varying relation/graph using Java generics
- Result
  - Encapsulate the return type of a continuous computation

# YASPER

## Abstractions

- Data Streams<Graph>
  - Implements Data Stream to stream commons RDF graphs
- Time-Varying<Graph>
  - Generalises the idea of time-varying relation/graph using Java generics
- Result
  - Encapsulate the return type of a continuous computation

```
1 RDF rdf = RDFUtils.getInstance();
2 String prefix = "http://www.w3.org/1999/02/22-rdf-syntax-ns#";
3 IRI a = rdf.createIRI(prefix + "type");
4
5 RDFStream stream = new RDFStream("stream1");
6
7 Graph g1 = rdf.createGraph();
8
9 g1.add(rdf.createTriple(
10     rdf.createIRI("a1"), //subject
11     a, //predicate
12     rdf.createIRI("Red"))); //object
13
14 stream.put(g1, 1000); //Stream in
15
16 Graph g2 = rdf.createGraph();
17
18 g2.add(rdf.createTriple(
19     rdf.createIRI("b1"), //subject
20     a, //predicate
21     rdf.createIRI("Blue"))); //object
22
23 stream.put(g2, 2000); //Stream in
```

# YASPER

## Abstractions

- Data Streams<Graph>
  - Implements Data Stream to stream commons RDF graphs
- Time-Varying<Graph>
  - Generalizes the idea of time-varying relation/graph using Java generics
- Result
  - Encapsulate the return type of a continuous computation

```java
public class TimeVaryingGraph implements TimeVarying<Graph> {

    private final StreamToRelationOp<?, Graph> op;
    private final IRI name;
    private E graph;

    @Override
    public void materialize(long ts) {
        graph = op.content(ts).coalesce();
    }

    @Override
    public Graph get() {
        return graph;
    }

[...]

}

```

# YASPER

## Abstractions

- Data Streams<Graph>
  - Implements Data Stream to stream commons RDF graphs
- Time-Varying<Graph>
  - Generalizes the idea of time-varying relation/graph using Java generics
- Result
  - Encapsulate the return type of a continuous computation

```
1  public interface Binding {
2
3      Set<Var> variables();
4
5      RDFTerm value(Var v);
6
7      boolean compatible(Binding b);
8
9  [...]
10 }
```

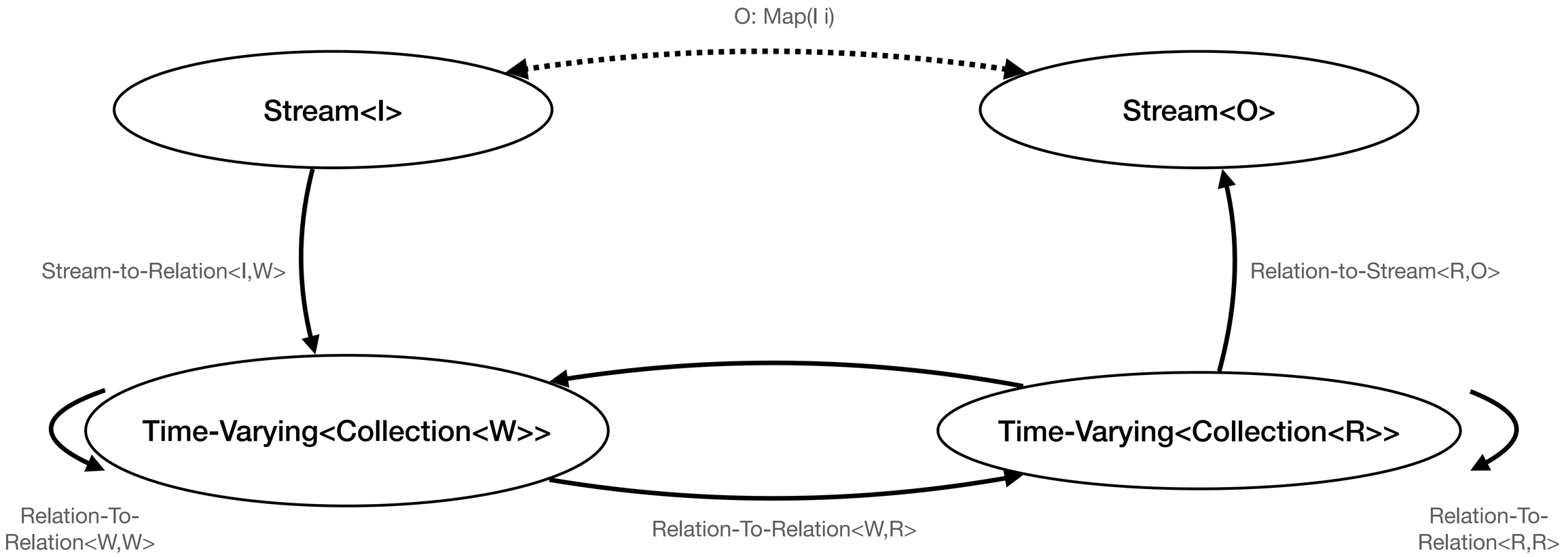
# Continuous Program

- A continuous program evaluates a set of (stateful)  functions (aka Tasks) on an input data stream, then it populates an output stream with the computation.
- In RSP4J, a continuous program is **described by four data types**, i.e.
  - **I**, i.e., the type of the data items in the **input data stream**
  - **O**, i.e., the Type of the data items in the **output data stream**
  - **W**, and **R**, which respectively describe the **intermediate** data **types**, right **after windowing** and after **the application of an R2R** Operator.


Do you think those are tables you’re querying?

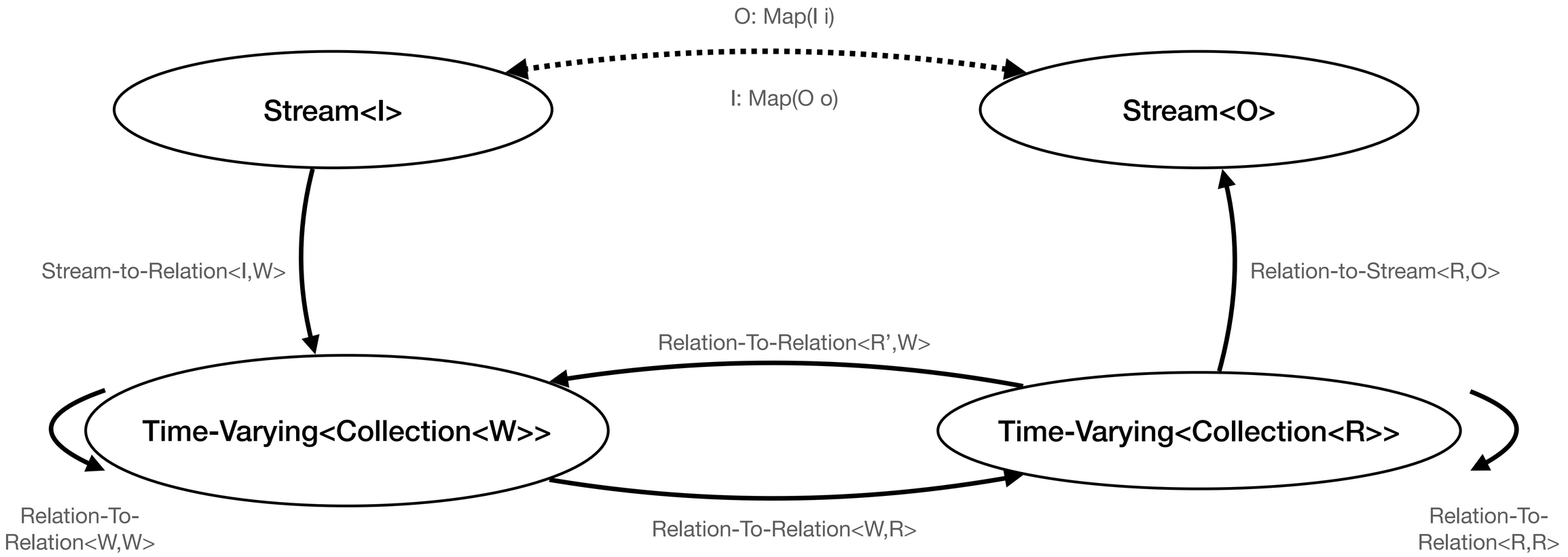
# RSP4J

## Generics



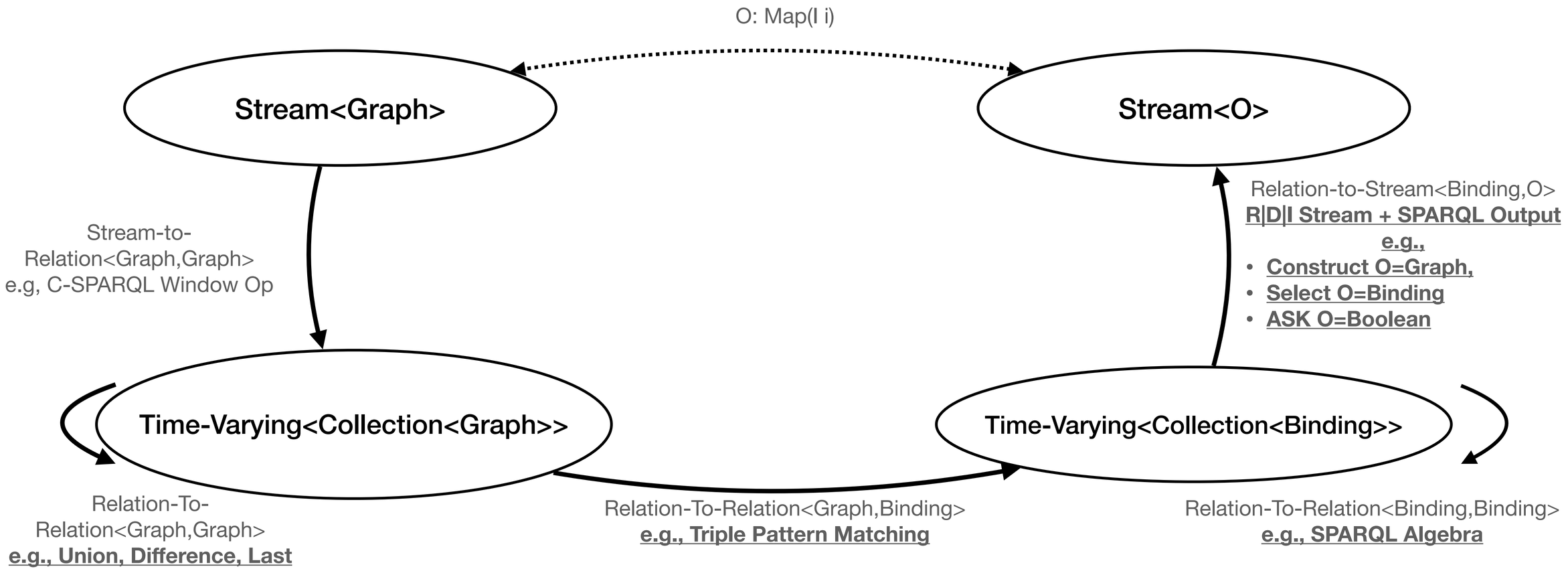
# RSP4J

## Generics



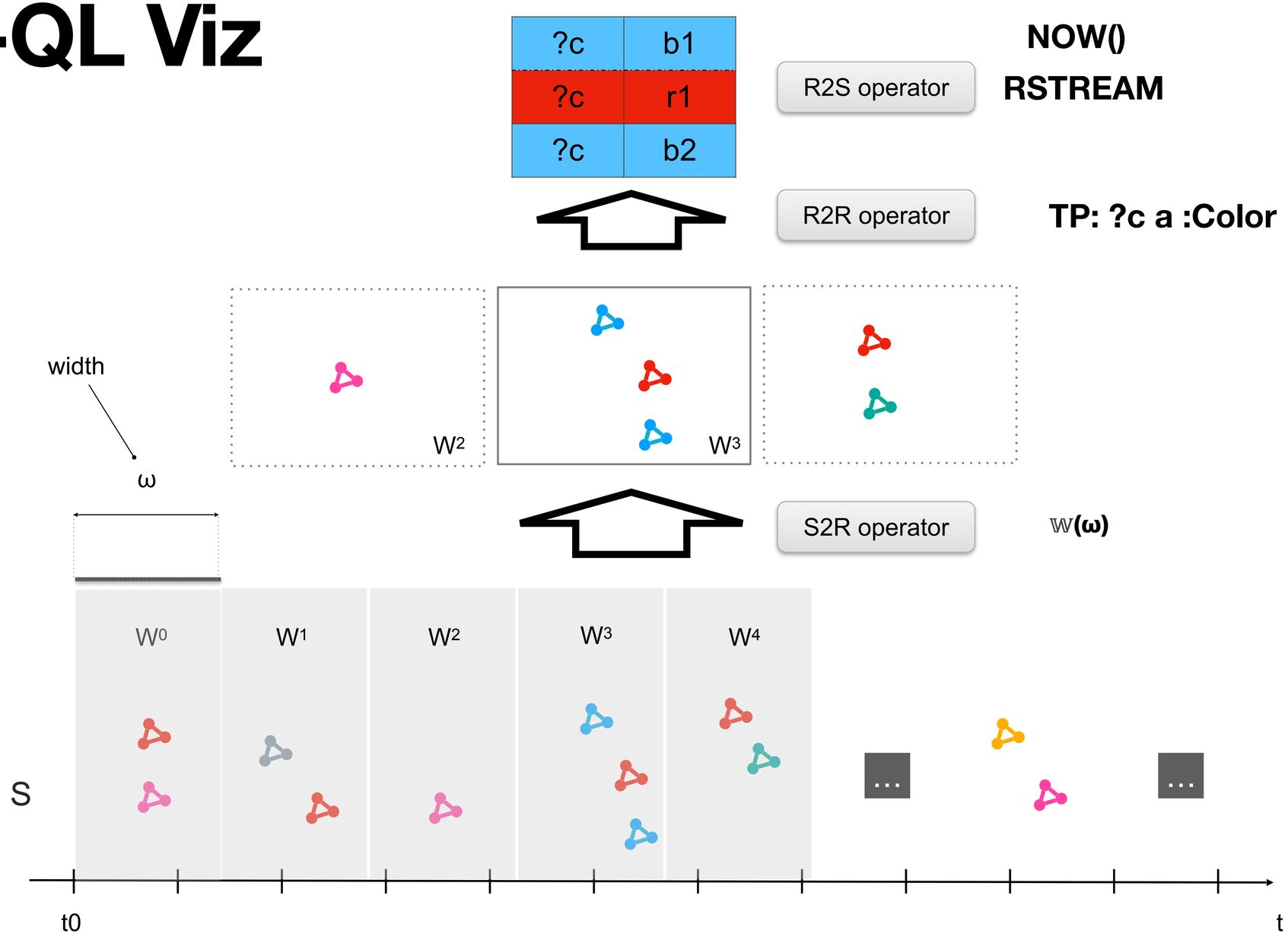
# RSP4J

## RSP-QL Configuration



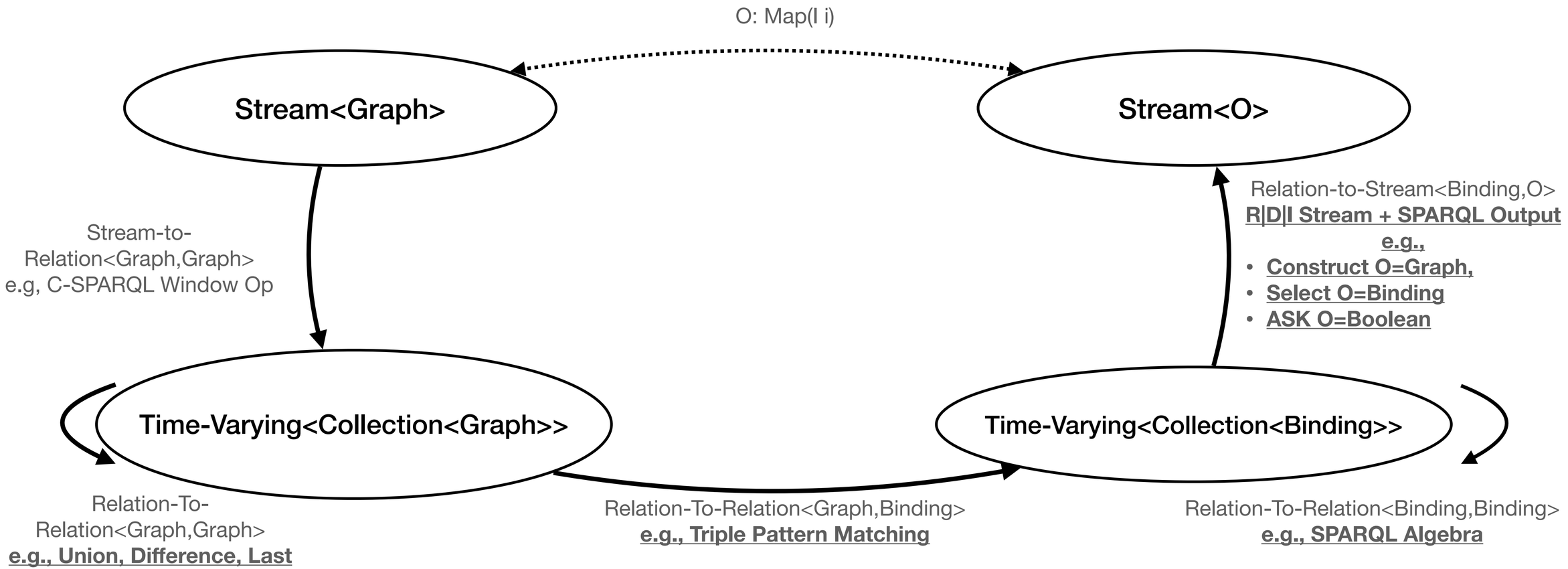
# RSP-QL Viz

## S2R



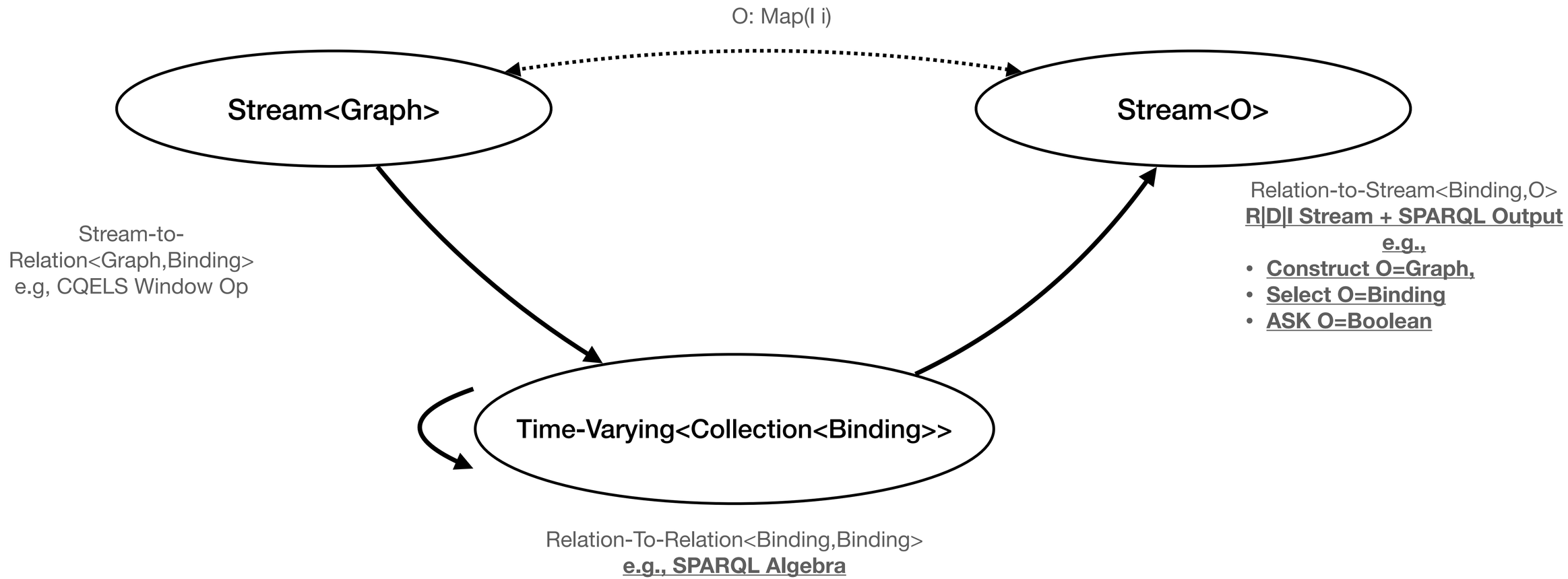
# RSP4J

## RSP-QL Configuration



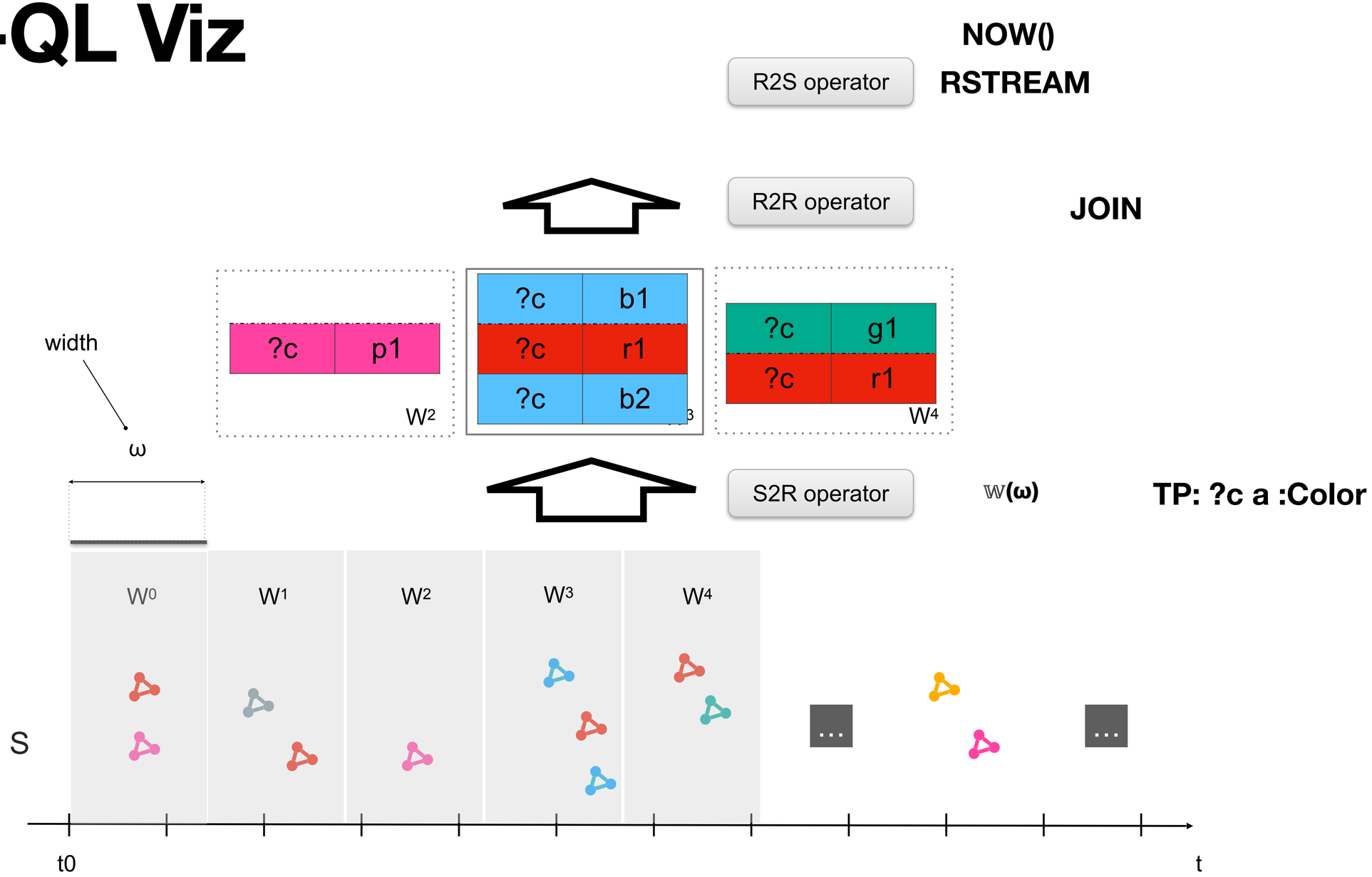
# RSP4J

## RSP-QL Configuration



# RSP-QL Viz

## S2R

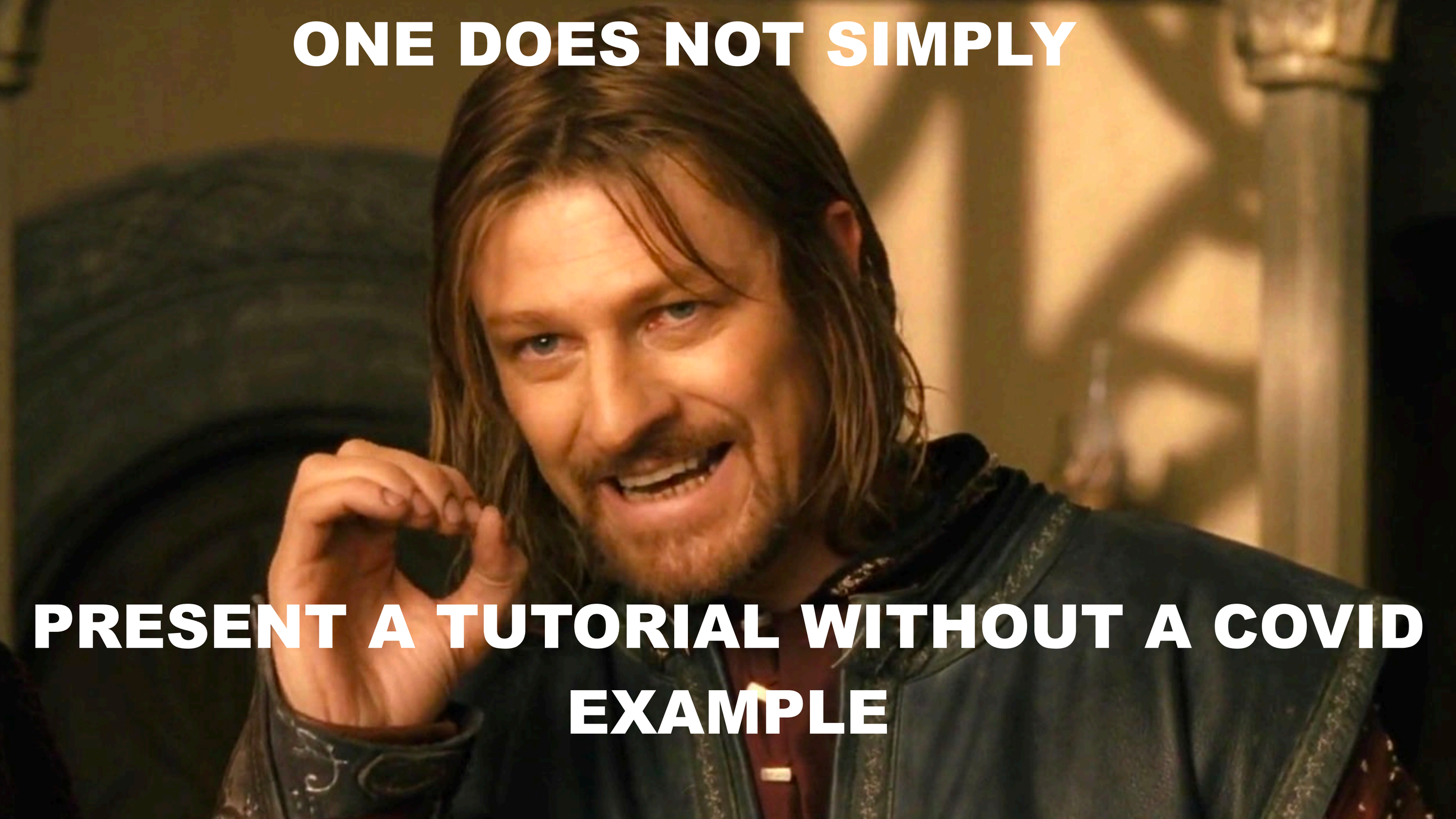


# RSP4J

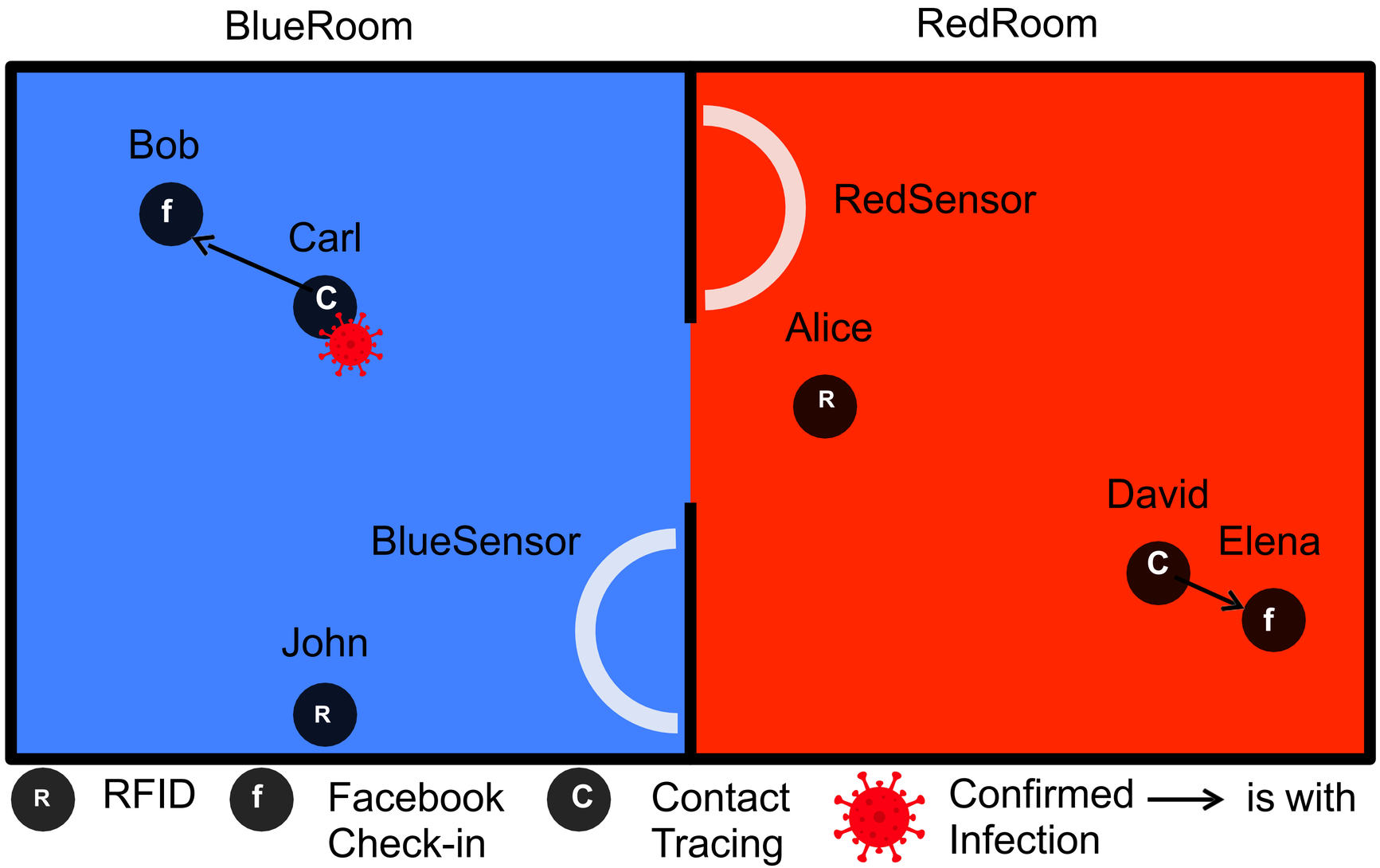
## 10.000ft View:  Querying

- Definition in RSP-QL syntax
- Creation of task from query
- Creation of ContinuousProgram from task

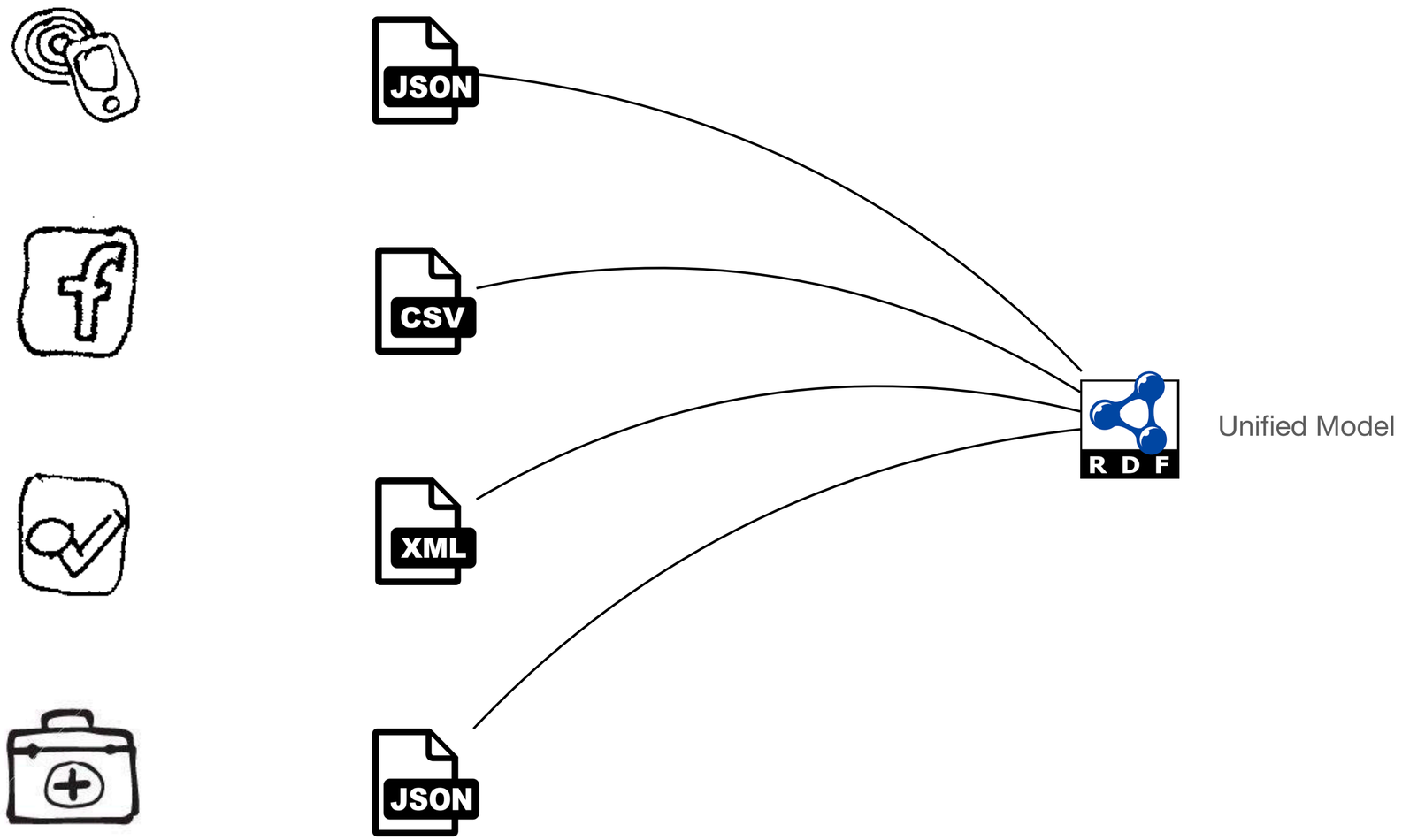
```
ContinuousQuery<Graph, Graph, Binding, Binding> query =
  TPQueryFactory.parse(
  """REGISTER RSTREAM <http://out/stream> AS
     SELECT (COUNT(?s) AS ?count)
     FROM NAMED WINDOW <w1> ON <http://test/stream> [RANGE PT1S STEP PT1S]
     WHERE {
       WINDOW <w1> { ?s ?p ?o .}
     }""");

Task<Graph, Graph, Binding, Binding> t =
        new QueryTask.QueryTaskBuilder()
           .fromQuery(query)
           .build();

ContinuousProgram<Graph, Graph, Binding, Binding> cp =
  new ContinuousProgram.ContinuousProgramBuilder()
           .in(inputStream)
           .addTask(t)
           .out(query.getOutputStream())
           .build();
    // Add the Consumer to the stream
query.getOutputStream().addConsumer((el, ts) ->
                                    System.out.println(el + " @ " + ts));
```


ONE DOES NOT SIMPLY
PRESENT A TUTORIAL WITHOUT A COVID EXAMPLE

# Covid Scenario



# Realistic example data in the streams



# Simplified example data in the streams

| Sensor | Room | Person | Time-stamp |
|---|---|---|---|
| RedSensor | RedRoom | Alice | $T_1$ |
| … | … | … | … |

| Person | ChecksIn | Time-stamp |
|---|---|---|
| Bob | BlueRoom | $T_2$ |
| … | … | … |

| Person | With | Time-stamp |
|---|---|---|
| Carl | Bob | $T_2$ |
| … | … | … |

| Person | Result | Time-stamp |
|---|---|---|
| Carl | Positive | $T_3$ |
| … | … | … |

# Simplified example data in the streams

| Sensor | Room | Person | Time-stamp |
|---|---|---|---|
| RedSensor | RedRoom | Alice | $T_1$ |
| … | … | … | … |

```
:observationX a :RFIDObservation .
:observationX :who :Alice .
:observationX :where :RedRoom .
:Alice :isIn :RedRoom .
```

| Person | ChecksIn | Time-stamp |
|---|---|---|
| Bob | BlueRoom | $T_2$ |
| … | … | … |

```
:postY a :FacebookPost .
:postY :who :Bob .
:postY :where :BlueRoom .
:Bob :isIn :BlueRoom .
```

| Person | With | Time-stamp |
|---|---|---|
| Carl | Bob | $T_2$ |
| … | … | … |

```
:postZ a :ContactTracingPost .
:postZ :who :Carl.
:Carl :isWith :Bob .
```

| Person | Result | Time-stamp |
|---|---|---|
| Carl | Positive | $T_3$ |
| … | … | … |

```
:postQ a :TestResultPost.
:postQ :who :Carl .
:postQ :hasResult :positive
```

# Covid Scenario Insights

- Who is with who in which room?

  R f C

  e.g. *Dave, Elena, RedRoom*

- Who is infected through close contact?

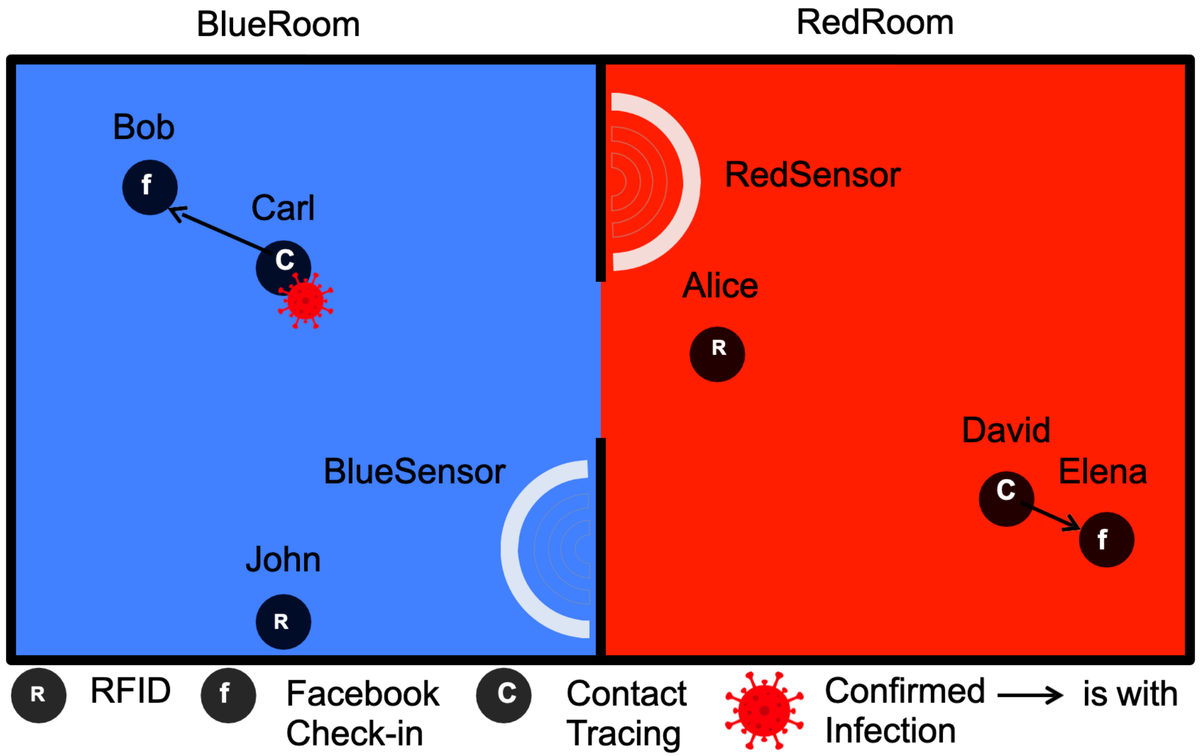
  R f C

  e.g. *Bob, Carl*

- Who is in a room where there is an infected person?

  R f C

  *e.g. BlueRoom, John, Bob, Carl*




Demo Time!

# Are you following? Let's check!

[source http://origin-www.yoursingapore.com/content/traveller/zh/browse/see-and-do/hands-on/_jcr_content/flash/image.img.png ]

- Open QueryProcessingAssignment (in org.streamreasoning.rsp4j.bigdata2021.processing.assignment)

- Define a query that extracts who might be infected through close contact in the last 10 minutes, with test results valid for 24 hours.

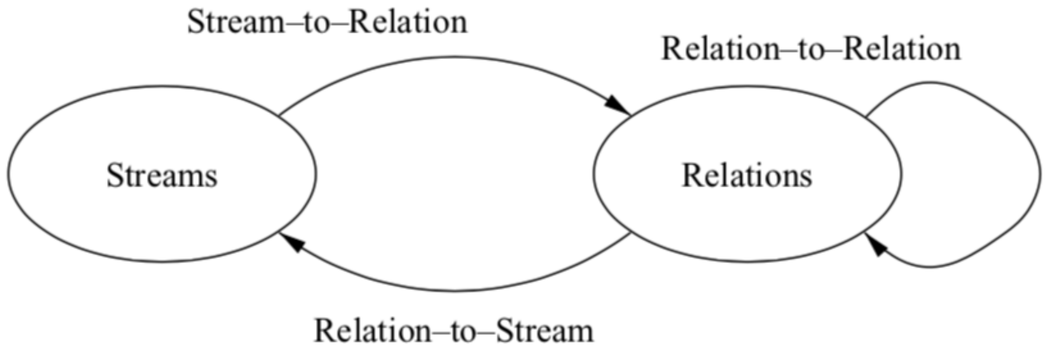
- Check the example (QueryProcessingExample) to get you started

- Git repo: https://github.com/pbonte/rsp4j-bigdata2021

# RSP4J

## 10.000ft **View: Operators**

- Generalize CQL Style of operators families
- Includes already canonical examples of S2R
  - CQELS's Window Operator
  - CSPARQL's Window Operator
- Includes custom implementation of R2R
  - BGP
  - TP
- Includes generic implementation of R2S
  - RStream, IStream, DStream



# RSP4J

## Operators

- Customize Tasks using TaskBuilder
- Specify operators (S2R, R2R, R2S)
- Optional aggregation
- Creation of ContinuousProgram from Task

```
1 Task<Graph, Graph, Binding, Binding> t =
2     new Task.TaskBuilder()
3             .addS2R("stream1", build, "w1")
4             .addR2R("w1", r2r)
5             .addR2S("out", new Rstream<Binding, Binding>())
6             .aggregate("w1","COUNT","s","count")
7             .build();
8  ContinuousProgram<Graph, Graph, Binding, Binding> cp =
9     new ContinuousProgram.ContinuousProgramBuilder()
10            .in(inputStream)
11            .addTask(t)
12            .out(outStream)
13            .build();
14
15 outStream.addConsumer((el, ts) ->
16                     System.out.println(el + " @ " + ts));
```


Demo Time!

# Are you following? Let's check!

- Open AbstractionAssignment
(in org.streamreasoning.rsp4j.bigdata2021.processing.assignment)

- Follow the instructions and define the different operators in the Task

- Check the example (AbstractionExample) to get you started

[source http://origin-www.yoursingapore.com/content/traveller/zh/browse/see-and-do/hands-on/_jcr_content/flash/image.img.png ]


Demo Time!

# Are you following? Let's check!

- Open CustomR2RAssignment
(in org.streamreasoning.rsp4j.bigdata2021.processing.assignment)

- Follow the instructions and create your own R2R operator

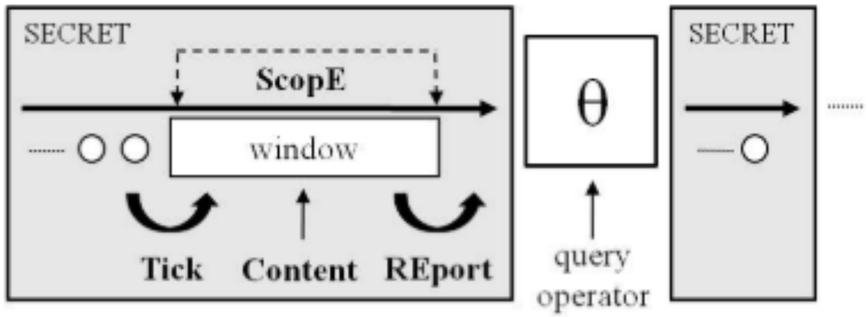
- Check the example (CustomR2RExample) to get you started

[source http://origin-www.yoursingapore.com/content/traveller/zh/browse/see-and-do/hands-on/_jcr_content/flash/image.img.png ]

# RSP4J

## 10.000ft **View: Execution Semantics**

- Generalise SECRET primitives
- Includes SECRETS Tick Enumeration
- Allows definition of custom reporting politicise
- Examples are used within
  - CQELS's Window Operator
  - CSPARQL's Window Operator





# Web Stream Processing with RSP4J

**The Web Conf 2022**

**Pieter Bonte (BE🇧🇪) & Matteo Belcao (🇮🇹IT) & Marco Balduini (🇮🇹IT) & Emanuele Della Valle (🇮🇹IT)**

**26/04/2022**