



RDFox Reasoning Workshop

Tom Vout

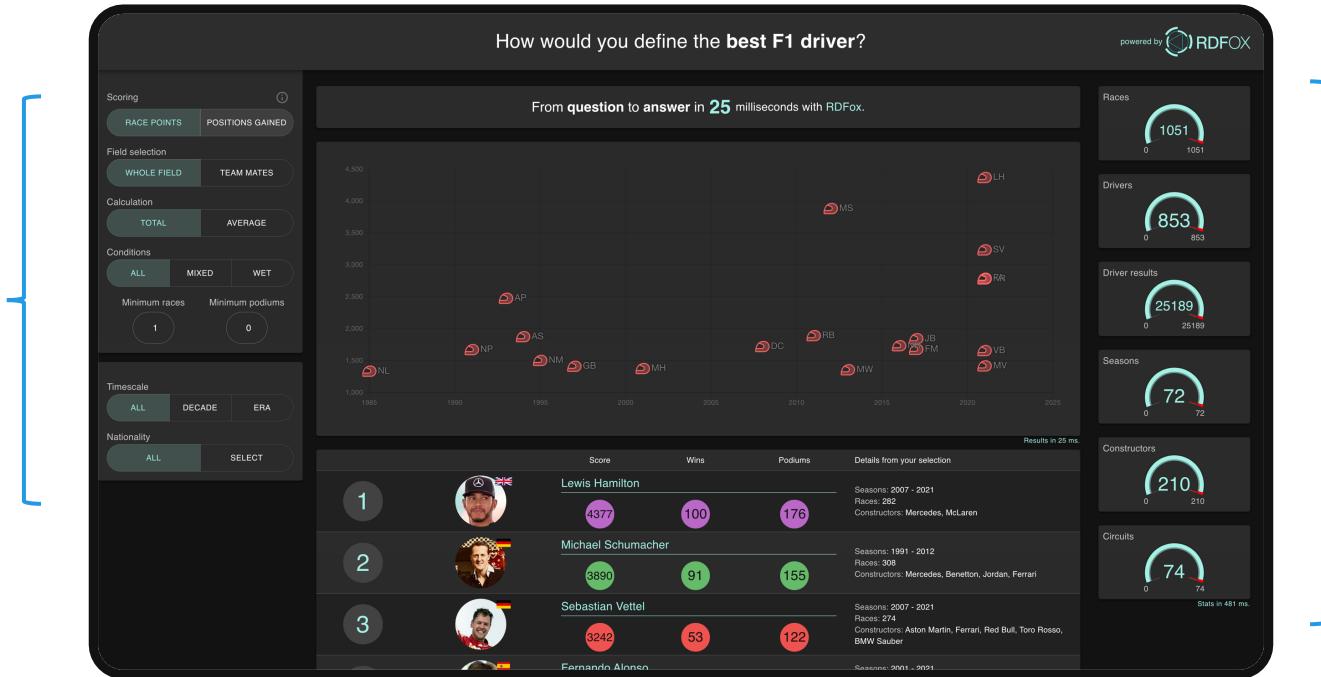
Technical Marketing Manager
Tom.vout@oxfordsemantic.tech



F1 Dashboard with Knowledge Graphs



Controls and filters for the scoring system.



Statistics about the data used to form the results.

This matches the filters.

Try it for yourself!

<http://f1.rdfbox.tech>

Objectives



By the end of the class everyone will:



Know how to write OWL axioms in Turtle syntax



Know how to add OWL axioms to RDFox datastores



Know how to write and import Datalog rules



Understand how RDFox performs reasoning



Know common uses of rules

What is Semantic Reasoning?



Semantic Reasoning (rules-based AI) enriches a Knowledge Graph by adding new information



This combines knowledge and data to answer more complex questions beyond queries alone



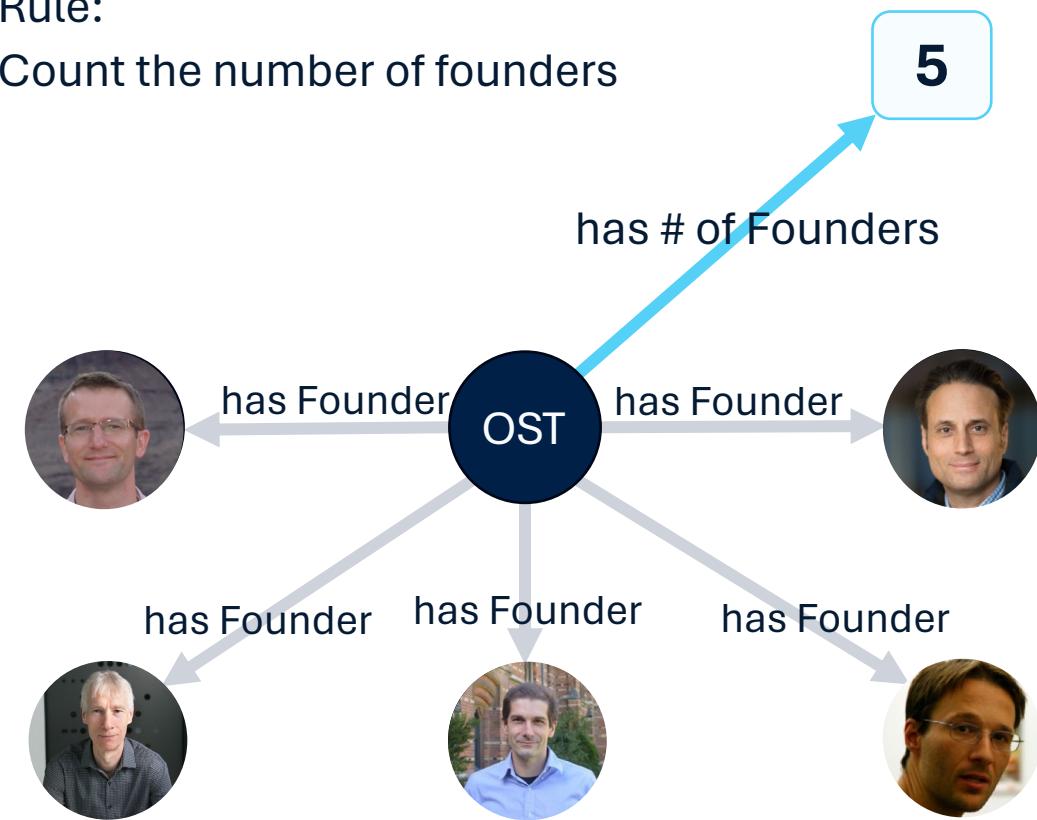
It drastically improves performance of queries by doing the hard work ahead of time



Incremental Reasoning updates automatically with new data

Rule:

Count the number of founders



What can Reasoning do for you?



```
1 #Query 11
2 PREFIX : <http://www.oxfordsemantic.tech/f1demo/>
3
4 # Drivers with their win percentage, ordered by win percentage
5 SELECT ?forename ?surname ?raceCount ?raceWins ?percentage
6 WHERE {
7 # First get the drivers
8 ?driver :driver_forename ?forename ;
9 :driver_surname ?surname .
10
11 # Then get the race count for each driver with an inner query...
12 {SELECT ?driver (COUNT(?race) AS ?raceCount)
13 WHERE {
14 ?result :result_driver ?driver ;
15 :result_race ?race .
16 }
17 GROUP BY ?driver}
18
19 # ... and get the *win* count for each driver with another inner query.
20 {SELECT ?driver (COUNT(?race) AS ?raceWins)
21 WHERE {
22 ?result :result_driver ?driver ;
23 :result_race ?race ;
24 :result_positionOrder 1 .
25 }
26 GROUP BY ?driver}
27
28 # Finally use the two aggregate variables to compute a percentage
29 # with the BIND keyword.
30 BIND(?raceWins/?raceCount AS ?percentage)
31
32 }
33 ORDER BY DESC(?percentage)
```

Fetched 108 answers in 0.011 s.

Datalog rules

```
1 #Query 19
2 PREFIX : <http://www.oxfordsemantic.tech/f1demo/>
3
4 SELECT ?forename ?surname ?percentage
5 WHERE {
6 ?driver :driver_forename ?forename ;
7 :driver_surname ?surname;
8 :hasWinPercentage ?percentage;
9 :hasRaceCount ?count .
10 }
11 ORDER BY DESC(?percentage)
```

Fetched 108 answers in 0.003 s.

What is RDFox®?



RDFox is a Knowledge Graph database
and Semantic Reasoning Engine



**Incremental
Semantic
Reasoning**



**Speed &
Scalability**



On Device



**Oxford
University**

Requirements



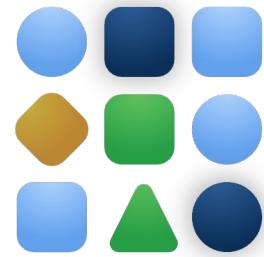
**A.
Get an RDFox
License**

<https://www.oxfordsemantictech.tech/free-trial>



**B.
Download
RDFox (& unzip)**

<https://www.oxfordsemantictech.tech/downloads>



**C.
Download the
class materials
from Github**

<https://github.com/OxfordSemantic/RDFoxWorkshop>



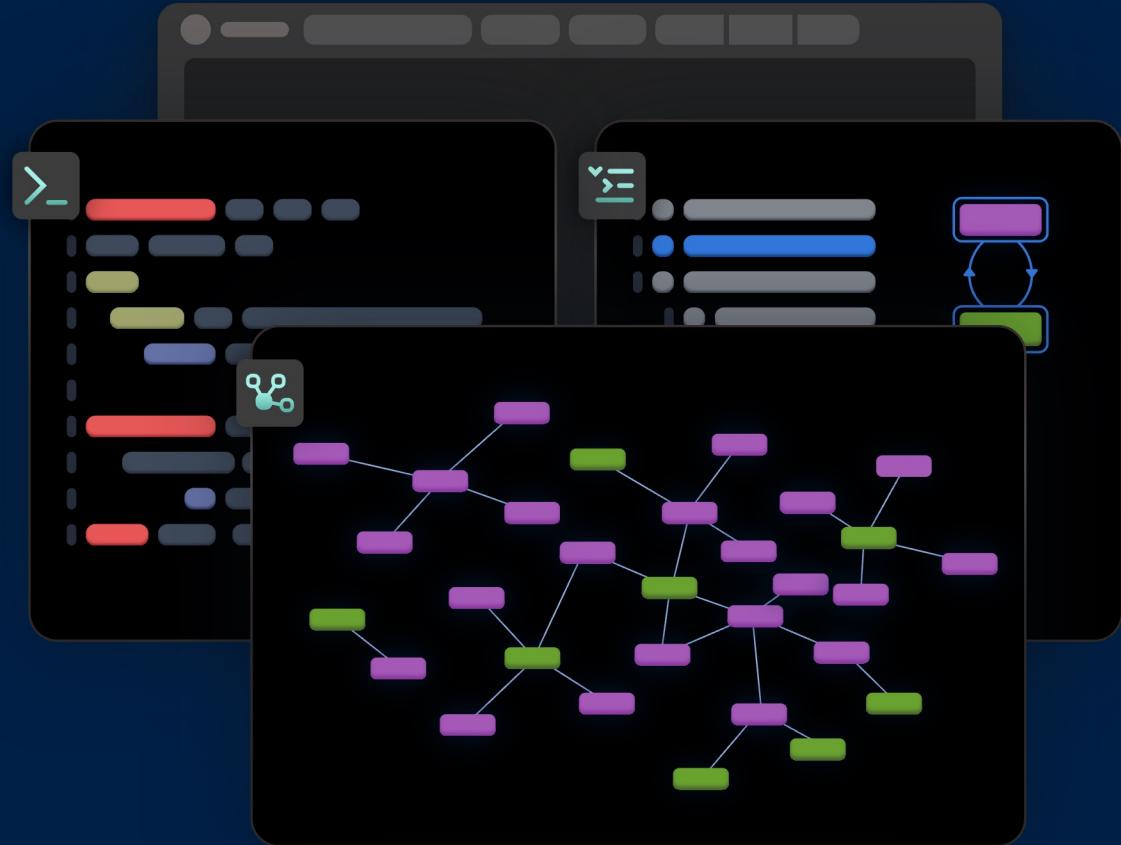
**D.
Get your IDE of
choice ready
(VS Code recommended)**

<https://code.visualstudio.com>

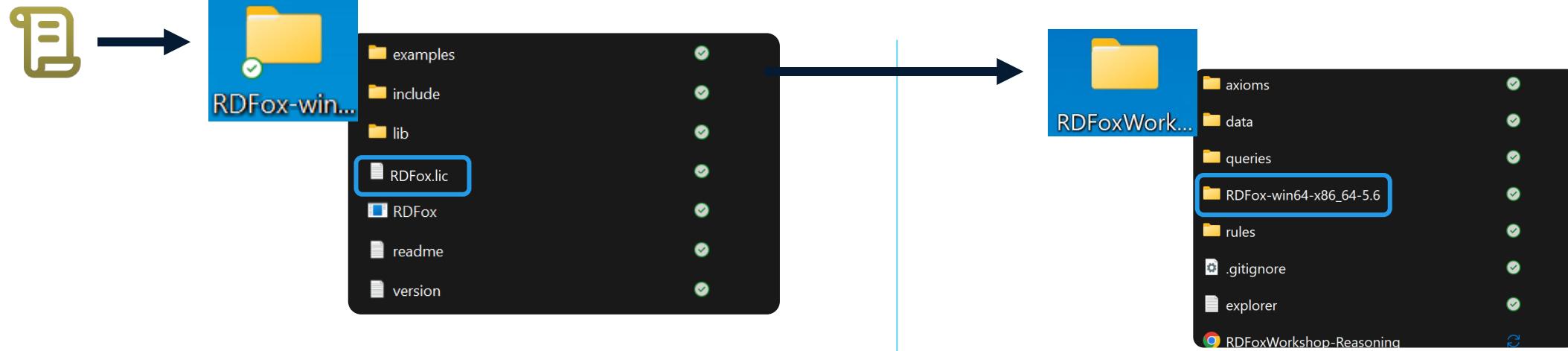


Setting up RDFox

- Start RDFox
- Create a data store
- Load the data



You should have...



1. Make sure to put your **license** **INSIDE** the **RDFox** folder

Without this setup you will need to use different file paths in the commands we provide.

2. Then drop your **RDFox** folder **INSIDE** the **Workshop** folder



Setting up RDFox

- 1 We recommend using an IDE (e.g.VS Code)
- 2 Open a terminal, navigate to the workshop folder (or open it in VS Code)

```
cd <path_to_workshop_folder>
```

- 3 From there run:
 - › MacOS ARM `./RDFox-macOS-arm64-7.1/RDFox sandbox`
 - › MacOS INTEL `./RDFox-macOS-x86_64-7.1/RDFox sandbox`
 - › Windows `RDFox-win64-x86_64-7.1/RDFox.exe sandbox`

- 4 The RDFox server should be running

```
A new server connection was opened as role 'guest' and stored with name 'sc1'.
```



Loading Data into RDFox

- 1 First we need to create a data store...

```
dstore create f1
```

-
- 2 Then to set it as active.

```
active f1
```

Everyone, include those who have just completed the SPARQL intro, need to do this step.

-
- 3 Now import the race data up to the year 2020

```
import +p data/upTo2020.ttl
```



Setting the output

In the Web Console and the shell

To see the output in the shell:

1

```
set output out
```

```
> set output out  
output = "out"  
> █
```

You will only need to use the shell in this tutorial.

To use the Web Console:

1

In the terminal, run:

```
endpoint start
```

```
> endpoint start  
WARNING: The RDFox endpoint is running with no transport layer security (TLS). This could allow attackers to  
steal information including role passwords.  
See the endpoint.channel variable and related variables in the description of the RDFox endpoint for  
details of how to set up TLS.  
The REST endpoint was successfully started at port number/service name 12110 with 7 threads.  
> █
```

2

Open a browser and go to:

localhost:12110/console

3

This will show an empty console

Select your data store at the top





OWL Reasoning

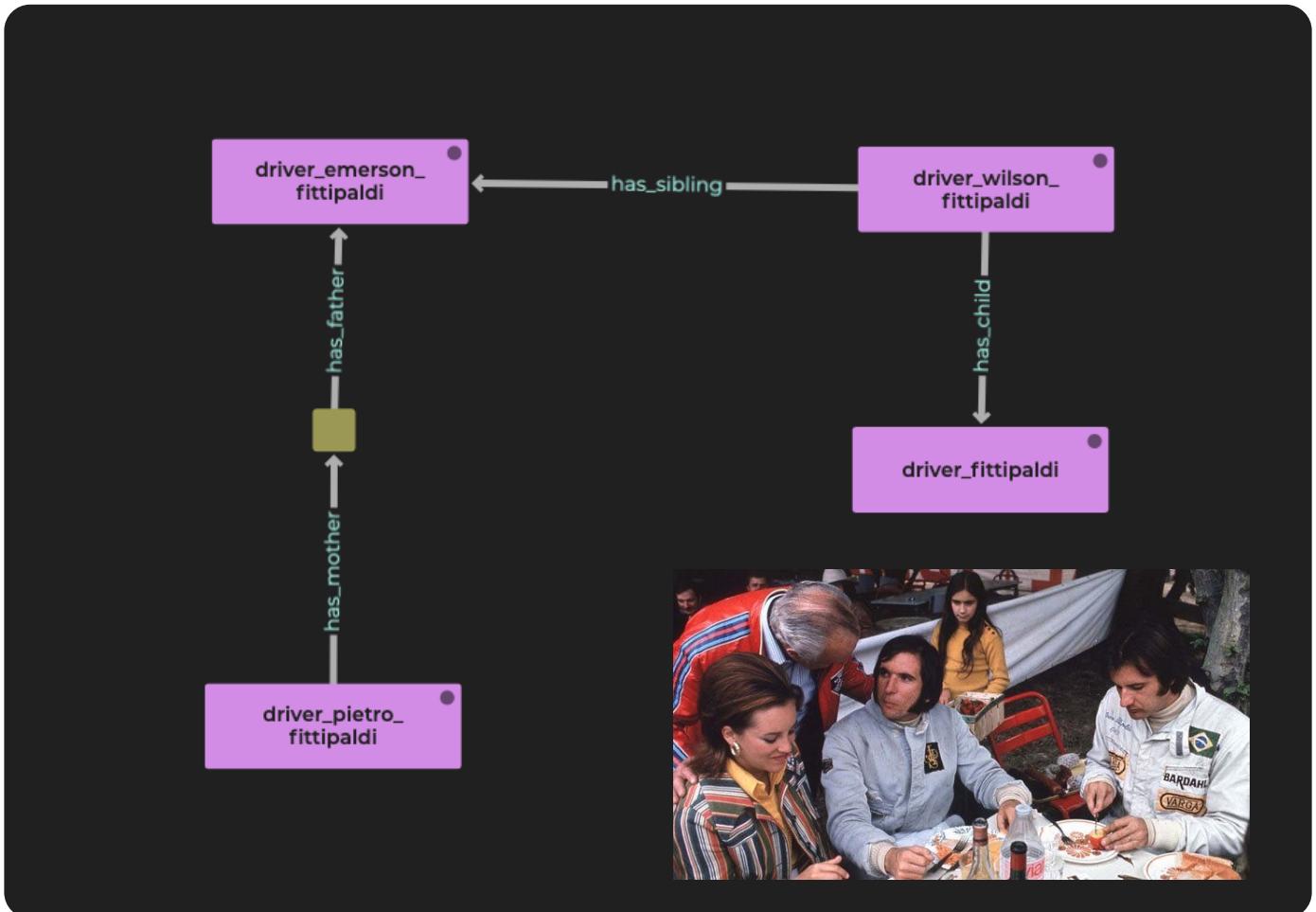
- Object & Datatype Properties
- Sub Properties
- Domain & Range
- Inverse Properties
- Symmetry & Transitivity
- Property Chains
- Class Equivalence
- Unions
- Class Intersections





Meet the Fittipaldi

- Data is often irregular
- Is there something that can help us make it regular?
- As it happens, there is...



Web Ontology Language (OWL)



- **Standard created by W3C**, RDFox supports the profile OWL 2 RL
- Axioms are assertions about classes, properties or individuals
- Ontology are collection of axioms
- Ontologies are themselves graphs and can be written in many different syntaxes – we will use Turtle



- Useful program for managing ontologies
- Free and open source
- Before writing an ontology, it is often better to have some data first and to consider the queries you want to facilitate





Axiom Set 1

Object & Datatype Properties

Predicates have two different types.

The **subject** of any predicate is always a **node**.

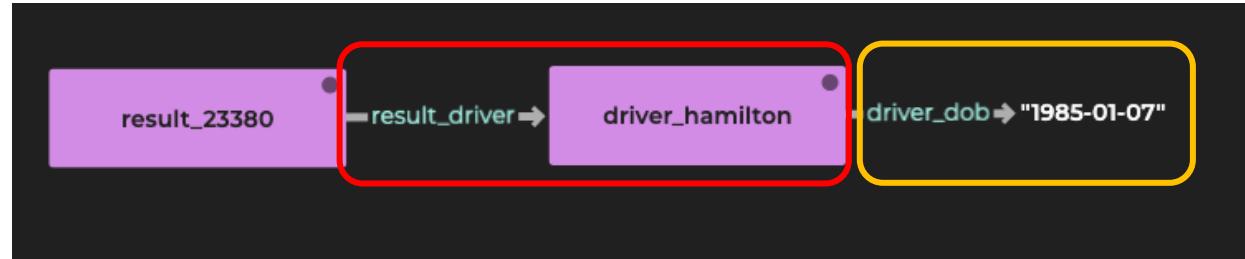
```
:result_driver rdf:type owl:ObjectProperty . ←  
:driver_dob rdf:type owl:DatatypeProperty . ←
```

Predicates that are **Object Properties** always connect a node to another node.

Predicates that are **Datatype Properties** always connect a node to a literal.

rdf:type is longhand for the '**a**' keyword we've seen in SPARQL .

Here, we're providing information about **predicates**.





Axiom Set 2

Domain & Range

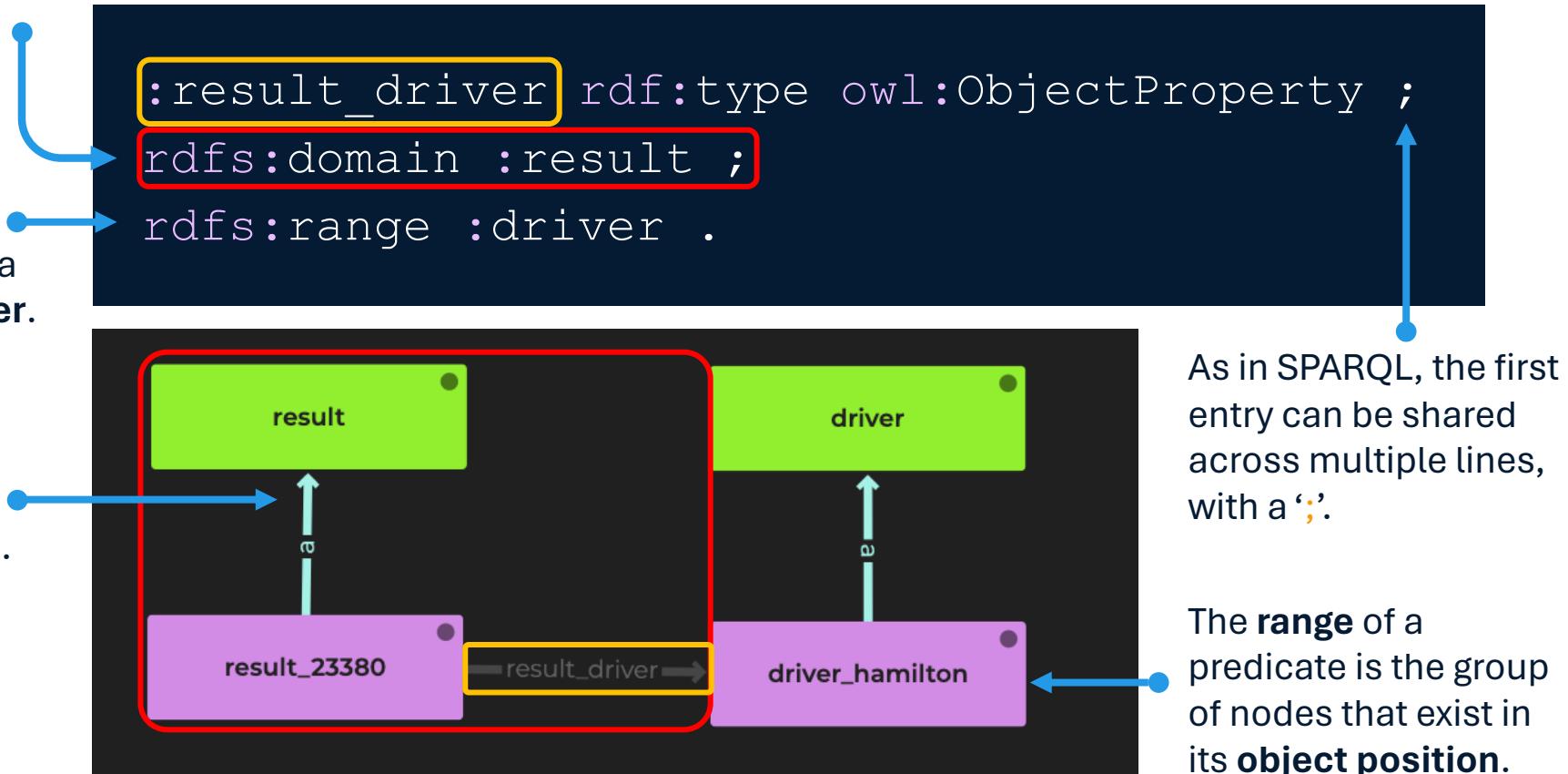
We can infer information about the subject and object nodes of a given relationship.

The **domain** of a predicate is the group of nodes that exist in its **subject position**.

By design, ‘**result_driver**’ will always connect an instance of a **Result** to an instance of a **Driver**.

Axioms do not validate the data, only add new information.

RDFox supports data validation with SHACL and Datalog but it is beyond the scope of this workshop. Get in to learn more.





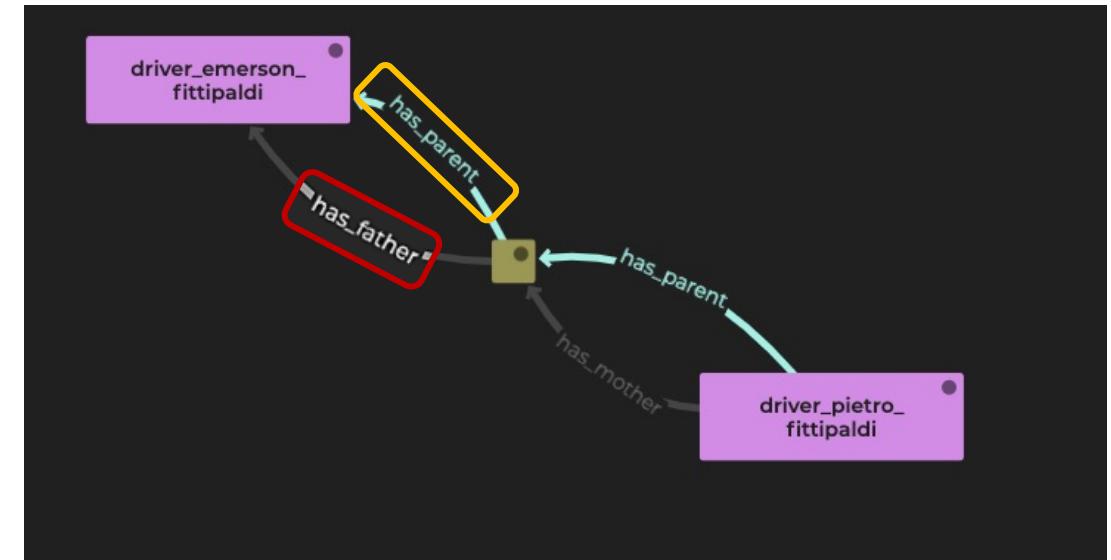
Axiom Set 3

Sub Properties

‘Father’ and ‘mother’ relationships are sub-properties of ‘parent’.

```
:has_father a owl:ObjectProperty ;  
rdfs:subPropertyOf :has_parent .  
  
:has_mother a owl:ObjectProperty ;  
rdfs:subPropertyOf :has_parent .
```

‘:has_parent’ will be added between all nodes that share the ‘:has_father’ relationship.



Multiple axioms can refer to the same properties.

‘:has_parent’ will also be inferred wherever nodes share ‘:has_mother’.



Axiom Set 4

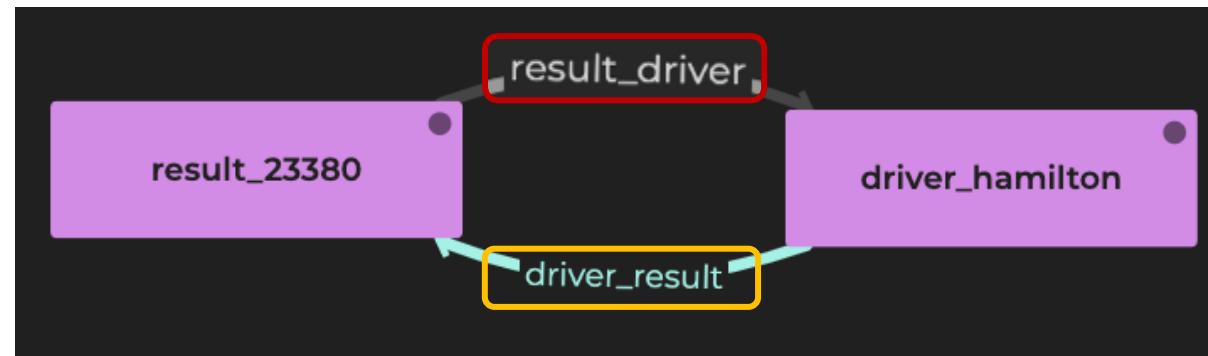
Inverse Properties

result_driver is the opposite of driver_result.

```
:result_driver rdf:type owl:ObjectProperty ;  
owl:inverseOf :driver_result .
```

‘:driver_result’ will be inferred between nodes that share the ‘:result_driver’ relationship in the opposite direction, **swapping around the subject and object nodes.**

Only **Object Properties** can have **inverse** relationships as the subject of a triple must always be a node.





Axiom Set 5

Symmetry & Transitivity

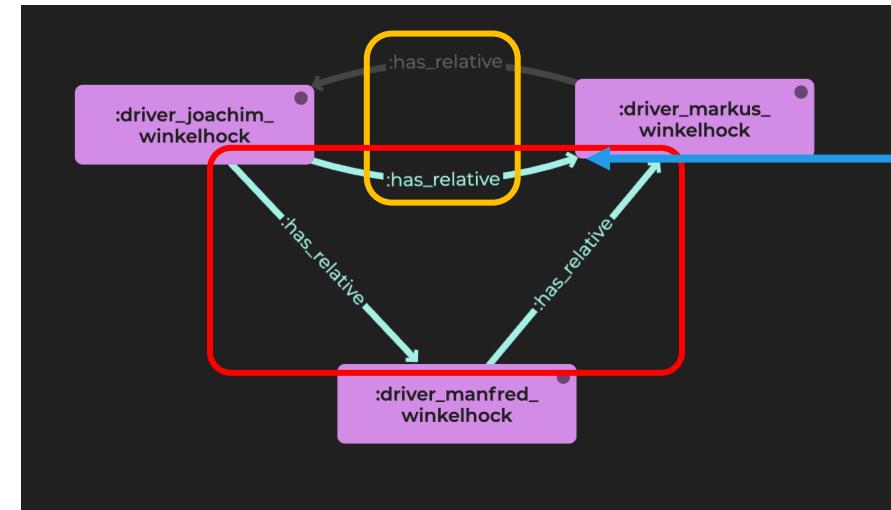
Family relatives share the same relationships in many ways.

```
:has_relative a owl:ObjectProperty ,  
owl:TransitiveProperty ,  
owl:SymmetricProperty
```

Transitive Properties are inferred between nodes that are **indirectly connected via a chain** of that property.

Symmetric Properties are inferred between nodes that are **indirectly connected via a chain** of that property.

As in SPARQL, the first and second entries can be shared across multiple lines, with a ‘,’.



The same fact can be derived in multiple ways.

RDFox will handle duplicates and only include one.

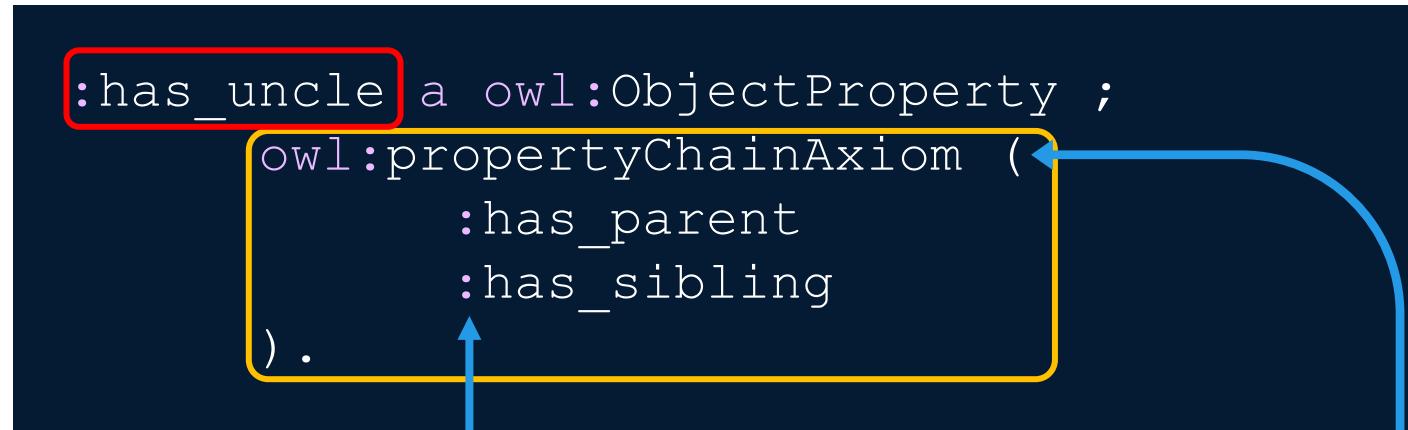
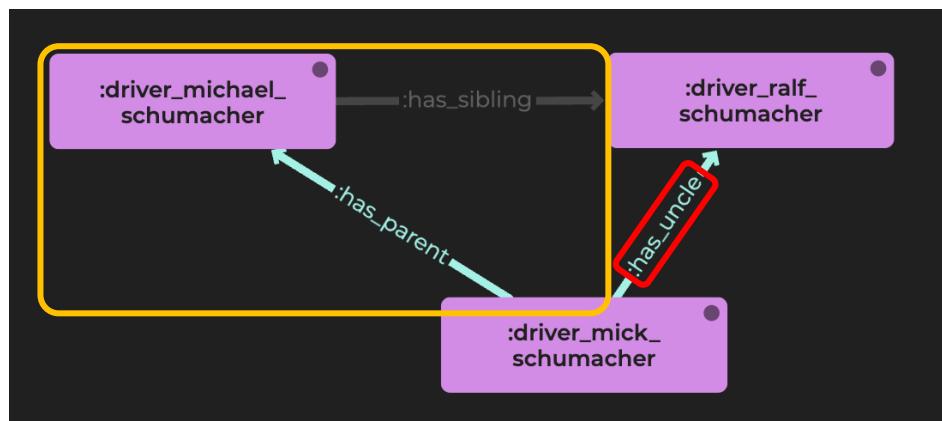


Axiom Set 6

Property Chains

An Uncle is a parent's sibling ...in this male-only dataset.

It can be useful to directly connect nodes that are connected indirectly by other nodes and a chain of properties.



The predicates in a **Property Chain** are given as lists using '**()**'.

- If **Alice** has a **parent Bob**, and **Bob** has a **sibling Charlie**, then it will be inferred that **Charlie** is **Alice's Uncle**.



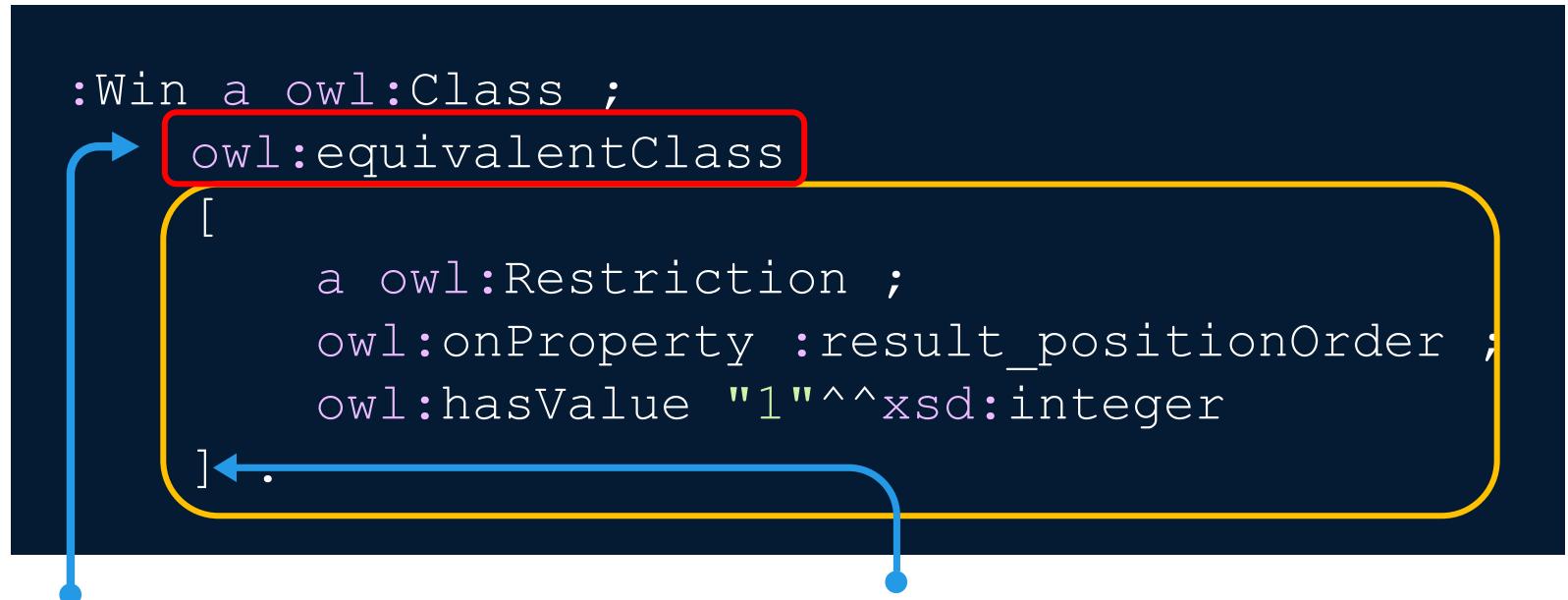
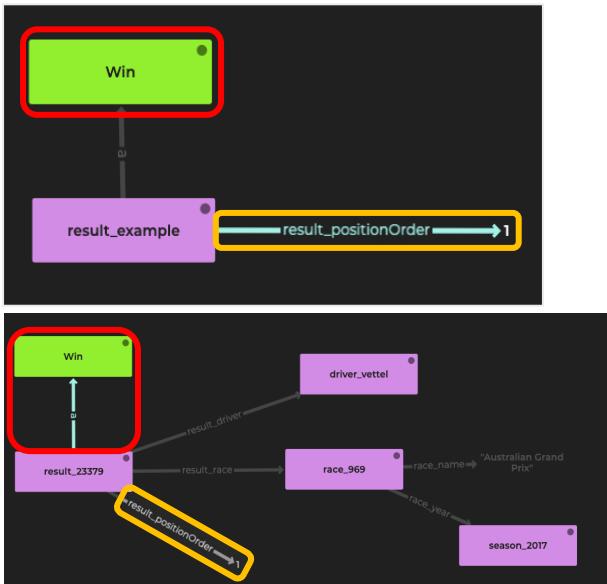
Axiom Set 7

Class Equivalence

Ensuring every first-place finish is classed as a win.

Classes are often groups of things that share common properties.

Having those **properties** is **equivalent to** being in that **class**, and vice versa.



For any class, an **equivalent class** can either be **another class** or a **node that has certain properties**.

Using square brackets '[]', we define a blank node that represents all things with a **restriction** that it has the property '**:result_positionOrder**' with a value of **1**.

Axiom Set 8



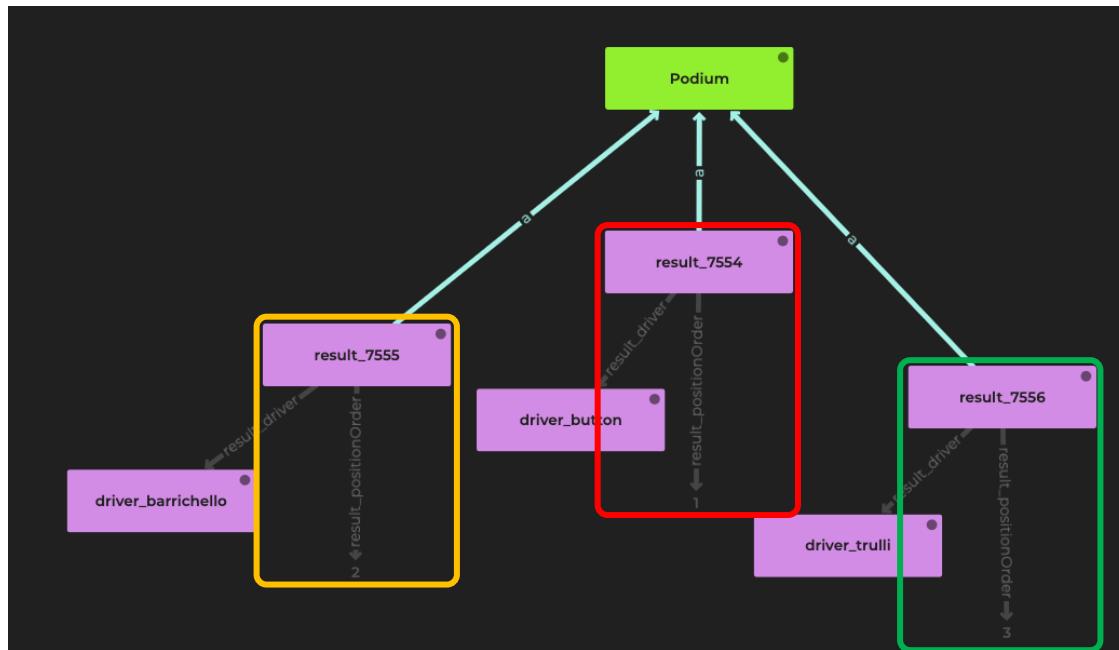
Unions

Accepting one thing OR another.

By using **owl:unionOf** we can accept several patterns.

In F1, a 'Podium' means finishing **1st**, **2nd**, or **3rd**.

We can combine both **lists '()'** and **blank nodes '[]'**.



```
:Podium a owl:Class ;
  owl:equivalentClass [
    a owl:Class ;
    owl:unionOf (
      [
        a owl:Restriction ;
        owl:onProperty :result_positionOrder ;
        owl:hasValue "1"^^xsd:integer
      ]
      [
        a owl:Restriction ;
        owl:onProperty :result_positionOrder ;
        owl:hasValue "2"^^xsd:integer
      ]
      [
        a owl:Restriction ;
        owl:onProperty :result_positionOrder ;
        owl:hasValue "3"^^xsd:integer
      ]
    )
  ] .
```

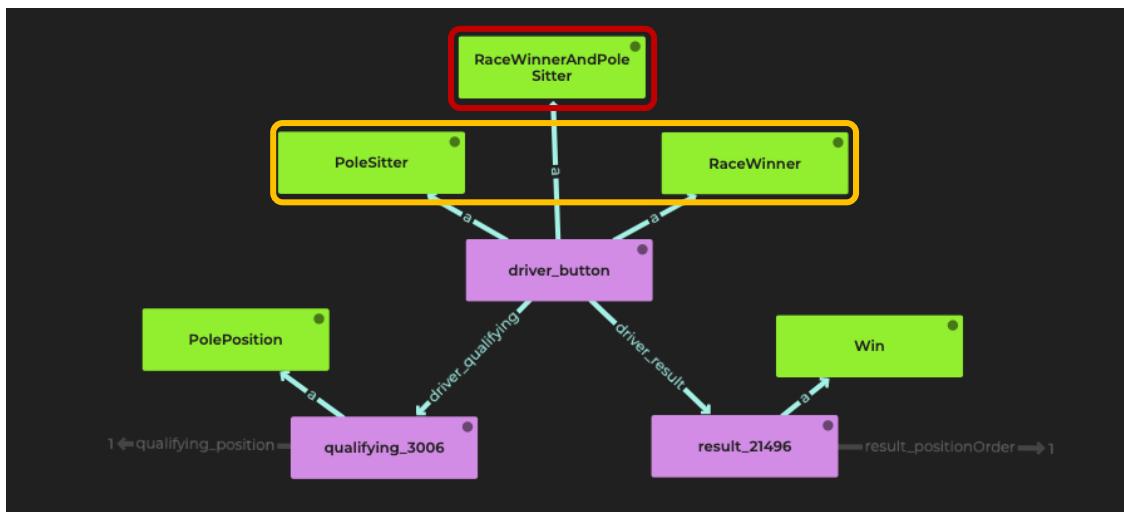


Axiom Set 9

Class Intersections

Combining one thing AND another.

Everything inside **owl:intersectionOf**
must be included in the matched pattern.



```
:RaceWinnerAndPoleSitter
a owl:Class ;
owl:equivalentClass [
a owl:Class ;
owl:intersectionOf (
:RaceWinner
:PoleSitter
)
].
```

Parsing the Ontology



Now we just need to import the axioms into a named graph, and then parse the graph for axioms:

```
import > :myAxioms axioms/axioms.ttl
```

```
import axioms :myAxioms
```

OWL 2 RL Limitations



- The RL fragment of OWL has some limitations
- It does not support *existentials*
- This is why we see some warning when parsing the axioms
- When asserting that a result is a podium, we have no way of knowing whether it's a P1, P2, or P3.
- Similarly, when declaring a race winner, we have no way of knowing which result of theirs is a win.



Removing Axioms

We will replace these axioms with Datalog rules, so we first need to remove them from our data store:

```
import axiom :myAxioms -
```

```
update ! Clear graph :myAxioms
```

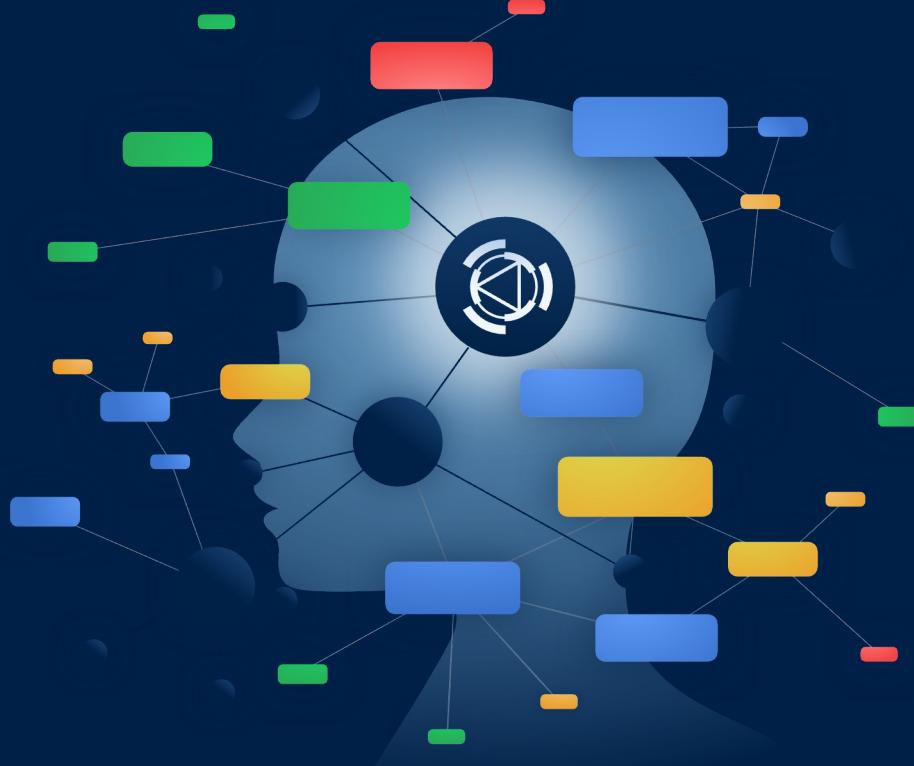
```
import > :myAxioms axioms/axioms.ttl
```

```
import axiom :myAxioms
```



Datalog Reasoning with RDFox

- Basic rules
- Filters
- Aggregates
- Negation
- Binds
- Incremental reasoning



Datalog Rules



[fact A] :- [fact B].

“A is true whenever B is true”



Rule 1

Basic rules

```
[?driver, :hasRacedIn, ?race] :-  
    [?result, :result_race, ?race],  
    [?result, :result_driver, ?driver] .
```

- This rule will add a direct link between drivers and the races they raced in. Notice this has the same effect as axiom 4, but expressed more clearly.

```
import rules/r1.dlog
```

- The same could seemingly be achieved with an equivalent SPARQL update, but rules offer several advantages over write queries.
- Check for inferred triples with query 14 – q14.rq

```
evaluate queries/q14.rq
```



Rule 2

Filters

```
[?driver, :hasPodiumInRace, ?race] :-  
    [?result, :result_race, ?race],  
    [?result, :result_driver, ?driver],  
    [?result, :result_positionOrder,  
     ?positionOrder],  
    FILTER(?positionOrder < 4) .
```

- We can use **FILTERs** in rules too, in this case to find the races in which a driver reached the podium.

```
import rules/r2.dlog
```

- Check for inferred triples with query 15 – q15.rq

```
evaluate queries/q15.rq
```

Rule 3



Aggregates

```
[?driver, :hasRaceCount, ?raceCount] :-  
    AGGREGATE (  
        [?driver, :hasRacedIn, ?race]  
    ON ?driver  
    BIND COUNT(?race) AS ?raceCount  
    ) .
```

- This rule adds a count of races a driver has entered.
- Notice we use the previously inferred **:hasRacedIn** property. Rules can be composed together, and RDFox will automatically find the order in which to run them.

```
import rules/r3.dlog
```

- Check for inferred triples with query 16 – q16.rq

```
evaluate queries/q16.rq
```



Rule 4 - Exercise

Aggregates

```
[?____, :hasRaceWinCount, ?____] :-  
    AGGREGATE (  
        [?____, :result_driver, ?____],  
        [?____, :result_positionOrder, 1] # Make sure the driver actually won  
        ON ?driver  
        BIND COUNT(?____) AS ?____  
    ) .
```

- This rule adds a count of races a driver has *won* (provided they won at least 1). We can put more than one atom *inside* the AGGREGATE statement.

```
import rules/r4.dlog
```

- Check for inferred triples with query 17 – q17.rq

```
evaluate queries/q17.rq
```

Rule 5



Negation

```
[?driver, a, :DriverWithoutPodiums] :-  
    [?driver, a, :driver],  
    NOT EXISTS ?race IN (  
        [?driver, :hasPodiumInRace, ?race]  
    ) .
```

- This rule tells us that if a driver does not have a race where they were on the podium, then they are a **:DriverWithoutPodiums**.
- So, we are looking for something that is **not** in the data (i.e. no race where the driver was on the podium).

```
import rules/r5.dlog
```

- Check for inferred triples with query 18 – q18.rq

```
evaluate queries/q16.rq
```

Negation as Failure



Negation as failure is negation understood as the *absence* of a triple.

Using Negation as Failure leads to ‘non-monotonic reasoning’- the engine may need to retract some triples instead of just adding them. This adds another dimension to an already complicated problem of reasoning planning, but RDFox handles all of that automatically.

- Example** What if a driver without any podiums so *far* gets a podium next year? In 2021, for instance, George Russell got his first podium. In our data, which is accurate up to and including 2020, George Russell would be a `:DriverWithoutPodiums`. But if we add data from 2021, we will need to retract this fact.
- Exercise** Run the query “q18_1.rq”, then import the data from file “2021-23.ttl” and try again
`evaluate queries/q18_1.rq`



Rule 6 - Exercise

Binds

```
[?driver, :hasWinPercentage, ?percentage] :-  
    [?_____, :hasRaceCount, ?_____] ,  
    [?_____, :hasRaceWinCount, ?_____] ,  
    BIND (?raceWinCount/?raceCount AS ?percentage) .
```

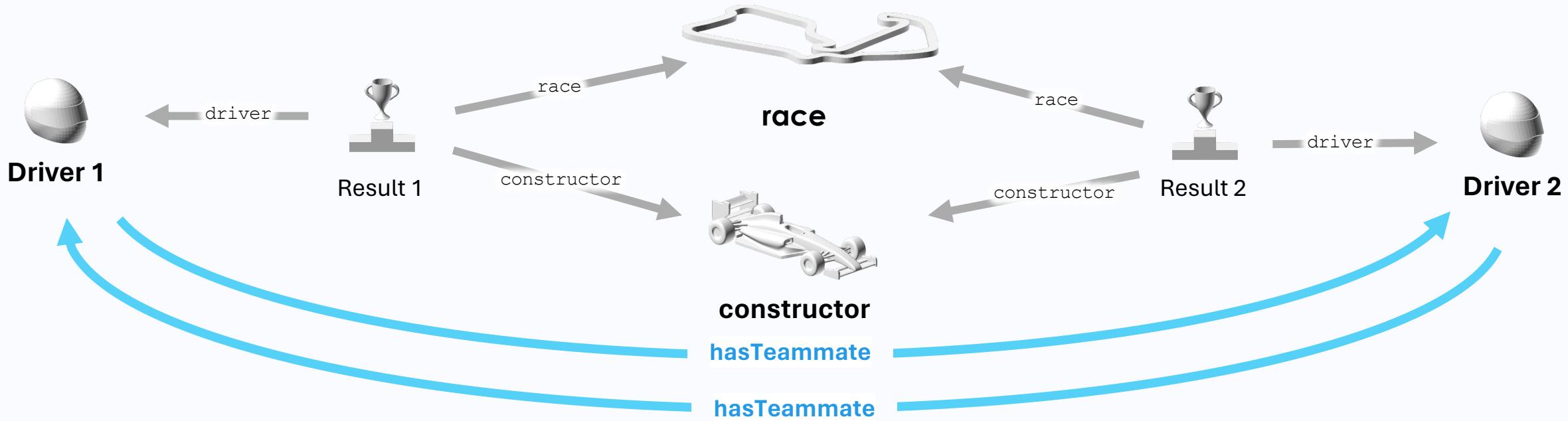
- With **BIND** we can use various mathematical functions, string manipulation, regular expression matching, conditional binds, hashing and IRI creation.

```
import rules/r6.dlog
```

- Check for inferred triples with query 19 – q19.rq

```
evaluate queries/q19.rq
```

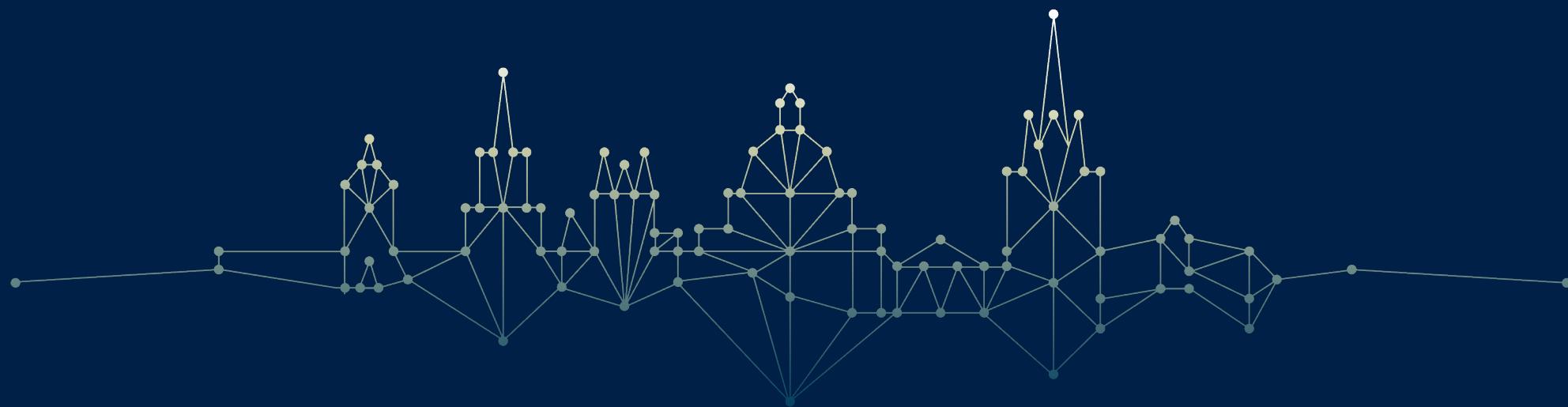
Bonus Exercise



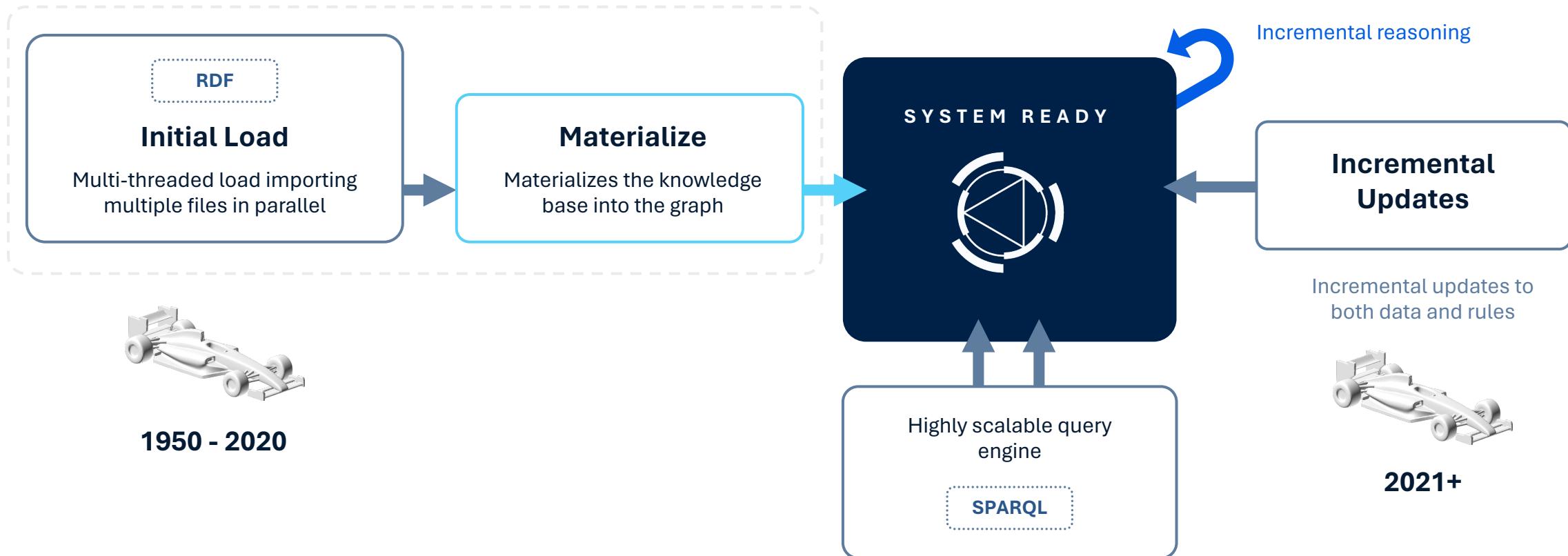
Write and import a rule to find out if two drivers have been teammates
(use the predicate **:hasTeammate**).



RDFox: How does it work?



Incremental Reasoning



Incremental Reasoning



- One of the key strengths of RDFox.
- Data can be imported at any time, and RDFox will automatically compute the necessary inferences.
- This all happens incrementally, without the need to reboot.
- Incremental reasoning open the door to many novel use cases which were previously impractical.

RDFox vs. Other Solutions

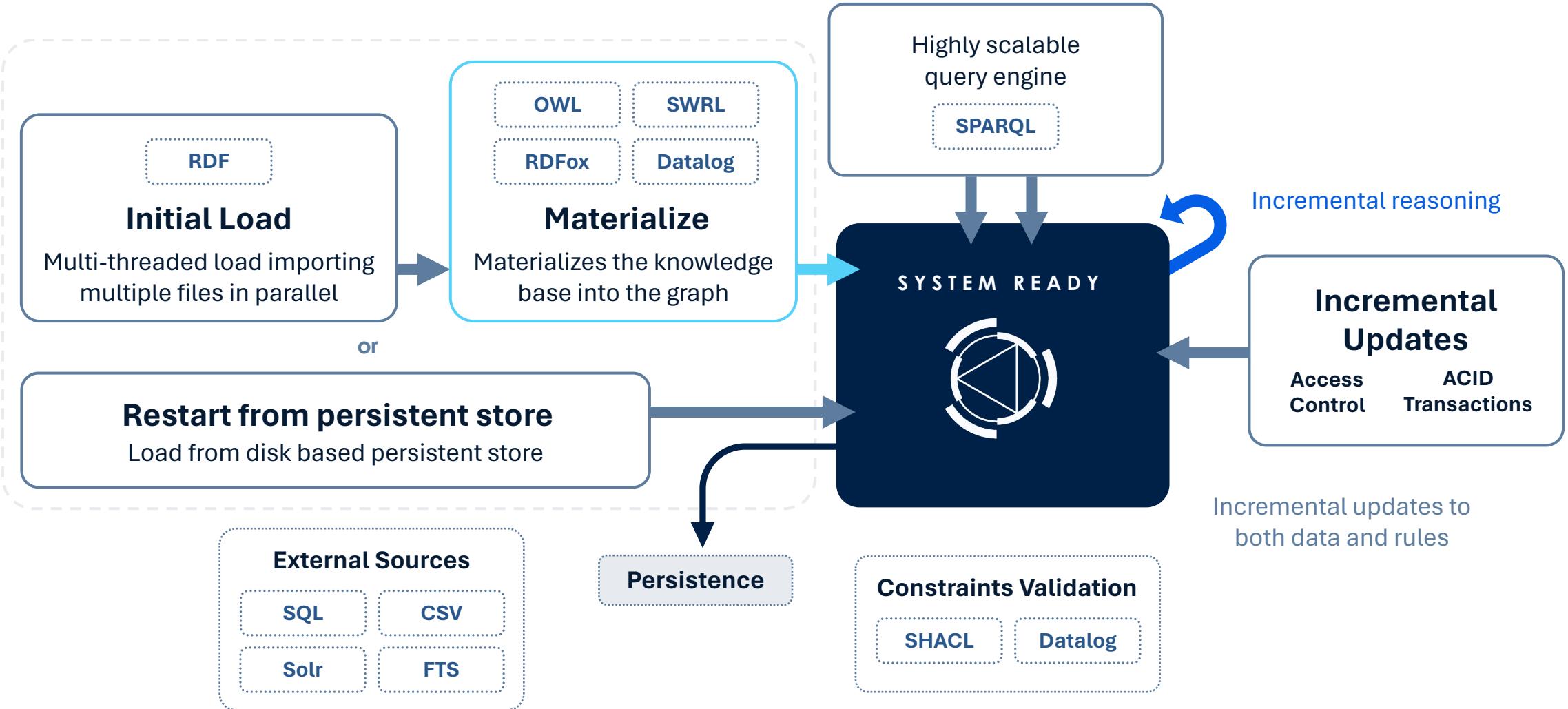


Feature	Example	RDFox	Materialisation Competitors	Query Rewriting Competitors
Ontological reasoning	List all financial instruments / derivative instruments / futures. (Class inferencing)	✓	✓ *	✓ **
Graph analytics	What is the nearest electrical switch for a circuit? (Negation & Recursion)	✓		
Data analytics	What is the total volume and value of trades? (Aggregation & Arithmetic)	✓		
Local constraints validation	No trades over a limit & no traders without trades! (Filters & Negation)	✓	Partial (SHACL)	
Global constraint validation	Every component must be connected to a power source! (Negation & Recursion)	✓		
All of the above	What is the total value of assets owned through holdings?	✓		

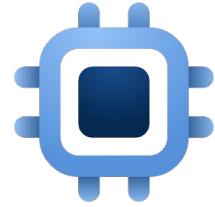
* many times slower than RDFox

** orders of magnitude slower than RDFox

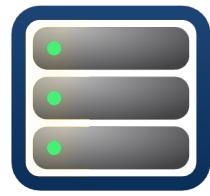
RDFox : How it works



Deployment Options



**Mobile & edge
devices**



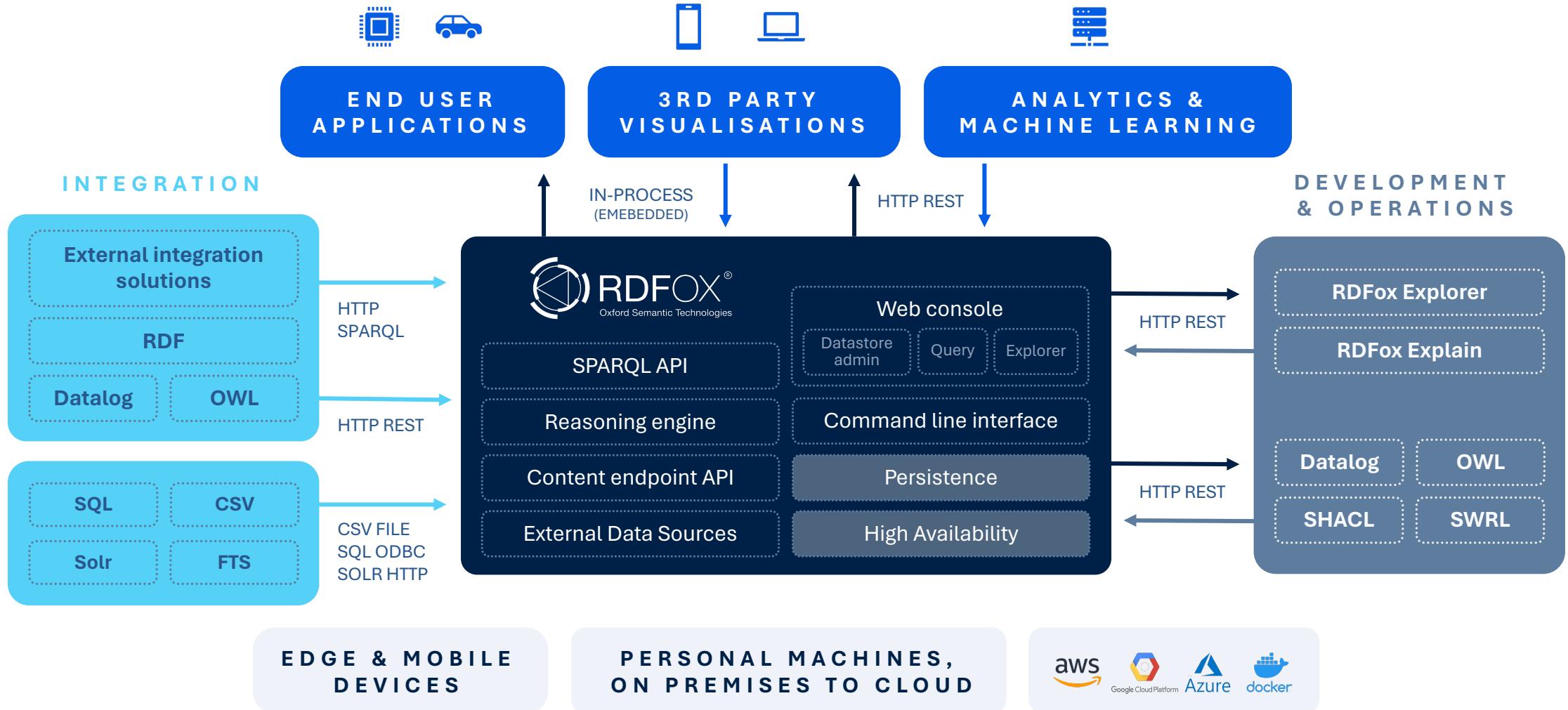
**Personal
machines on
premises to cloud**



Google Cloud Platform



RDFox Architecture



Your final exercise...



Click on the link below to start filling out our quick survey.
It won't take you more than 3 minutes!

<https://oxfordsemantictech.typeform.com/to/WuuNcFPm>

Additional information



Visit our website

<https://www.oxfordsemantic.tech>



Our blog

<https://www.oxfordsemantic.tech/blog>



Read our documentation

<https://docs.oxfordsemantic.tech/index.html>



Request an evaluation license

<https://www.oxfordsemantic.tech/free-trial>