



# RDFox Reasoning Workshop



The world's most performant knowledge graph and semantic reasoning engine.

# Requirements



## A. Get an RDFox License

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

## B. Download RDFox (& unzip)

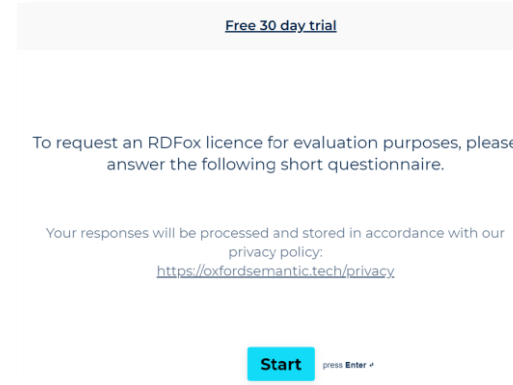
<https://www.oxfordsemantic.tech/downloads>

## C. Download the class materials from Github:

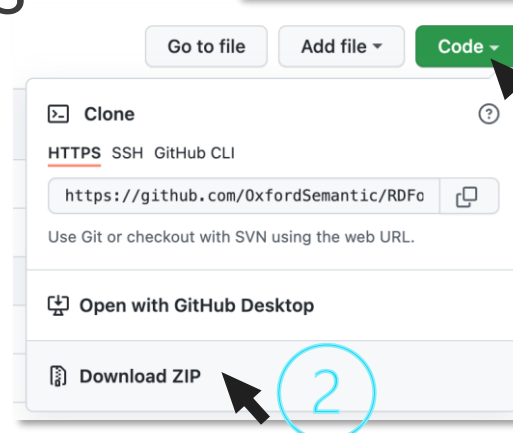
<https://github.com/OxfordSemantic/RDFoxWorkshop>

## D. *OPTIONAL* Get your IDE of choice ready (VS Code etc.)

<https://code.visualstudio.com/>



### Downloads





# You should have...



1

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



examples	✓	30/03/2022 11:23	File folder
include	✓	30/03/2022 11:23	File folder
lib	✓	30/03/2022 11:23	File folder
RDFox.lic	✓	30/03/2022 11:23	Text Document
RDFox	✓	30/03/2022 11:23	Application
readme	✓	30/03/2022 11:23	Text Document
version	✓	30/03/2022 11:23	Text Document

2

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



axioms	✓	24/05/2022 18:11	File folder
data	✓	24/05/2022 17:47	File folder
queries	✓	24/05/2022 17:47	File folder
RDFox-win64-x86_64-5.6	✓	26/05/2022 15:22	File folder
rules	✓	24/05/2022 15:05	File folder
.gitignore	✓	26/05/2022 15:23	Git Ignore Source ...
explorer	✓	24/05/2022 15:05	Text Document
RDFoxWorkshop-Reasoning	↻	10/05/2022 18:19	Chrome HTML Do...
RDFoxWorkshop-SPARQL	↻	10/05/2022 18:15	Chrome HTML Do...
README	✓	24/05/2022 15:05	Markdown Source...
start	✓	24/05/2022 15:05	Text Document
todo	✓	24/05/2022 15:05	Text Document

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



# Setting up RDFox



- We recommend using an IDE (e.g. VS Code)
- Open a terminal, navigate to the workshop folder (or open it in VS Code)  
`cd path/to/RDFoxWorkshop`
- From there run:
  - › MacOS ARM: `./RDFox-macOS-arm64-5.7/RDFox sandbox`
  - › MacOS INTEL: `./RDFox-macOS-x86_64-5.7/RDFox sandbox`
  - › Windows: `RDFox-win64-x86_64-5.7/RDFox.exe sandbox`
- The RDFox server should now be running.

```
Source code for RDFox v1.0 Copyright 2013 Oxford University Innovation Limited and subsequent improvements Copyright 2017-2021 by Oxford Semantic Technologies Limited.
```

```
This copy of RDFox is licensed for Developer use to Tom Vout (tom.vout@oxfordsemantic.tech) of OST until 07-Jun-2022 16:01:44
```

```
This system is equipped with 16.9 GB of RAM, and RDFox is configured to use at most 15.2 GB (89.9% of the total).
```

```
Currently, 2.8 GB (18.4% of the amount allocated to RDFox) appear to be available on the system.
```

```
Since RDFox is a RAM-based system, its performance can suffer when other running processes use a lot of memory.
```

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

```
> █
```



# Loading Data into RDFox

First we need to create a data store...

```
dstore create f1
```

Then to set it as active.

```
active f1
```

Then specify a prefix. We'll need it later.

```
prefix : <http://www.oxfordsemantic.tech/f1demo/>
```

Now import the race data up to the year 2020.

```
import data/upTo2020.ttl
```

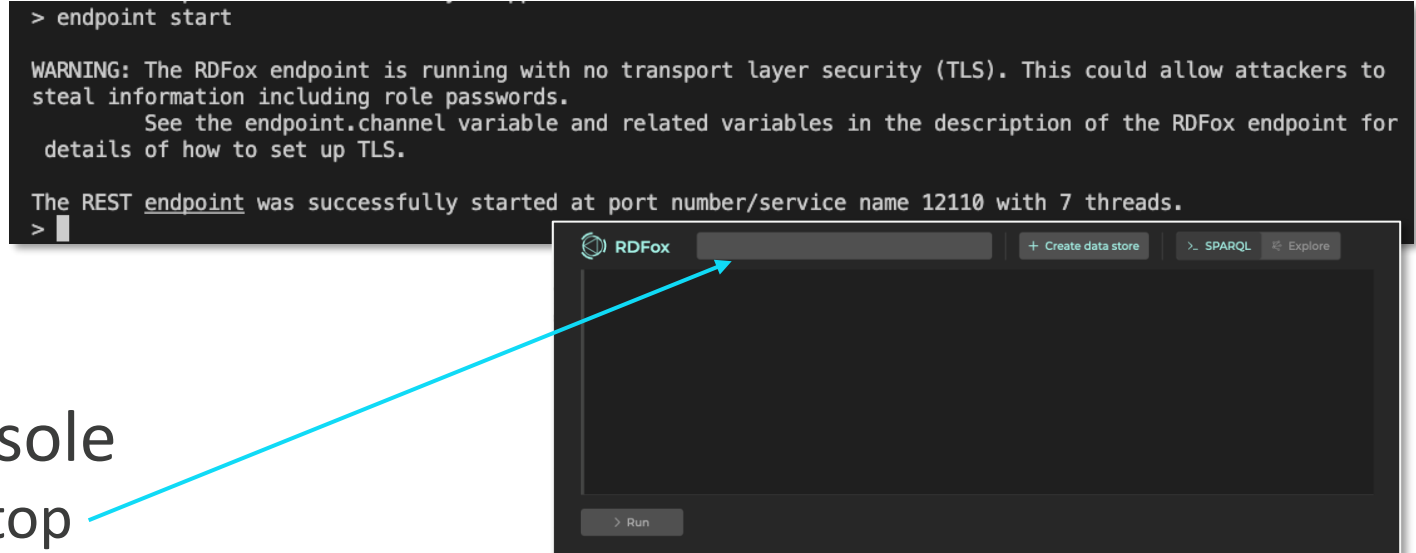


# Setting the output

## In the Web Console and the shell

To use the Web Console:


- In the terminal, run:  
`endpoint start`
- Open a browser and go to:  
[localhost:12110/console](http://localhost:12110/console)
- This will show an empty console
  - Select your data store at the top



To see the output in the shell:

`set output out`

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

You will need to use both of these through this tutorial 

# Objectives



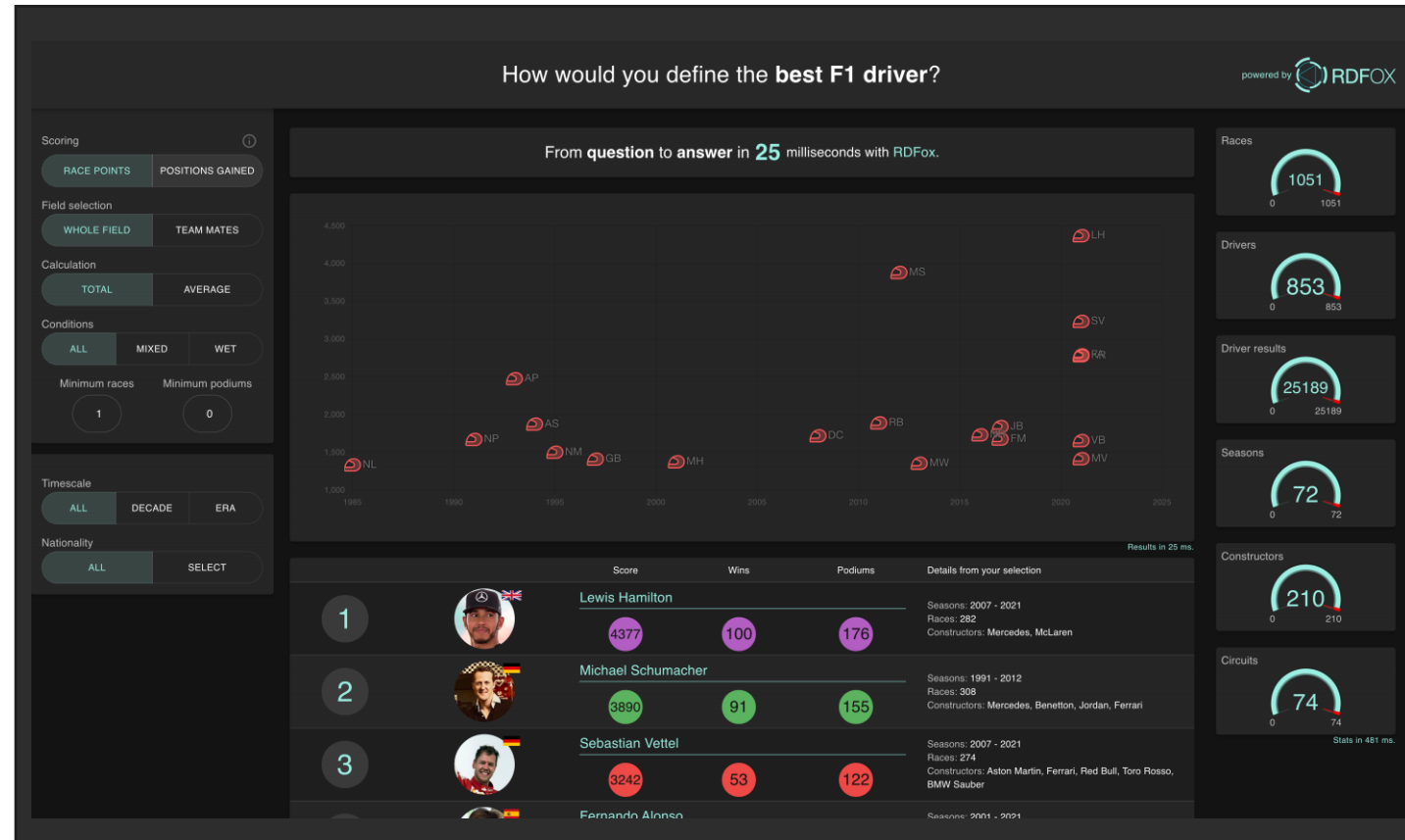
By the end of the class everyone will:

- 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

# Who is the Greatest Formula One Driver of All Time?



Controls  
and filters  
for the  
scoring  
system.



Try it for yourself!

<http://f1.rdfox.tech>





# OWL reasoning

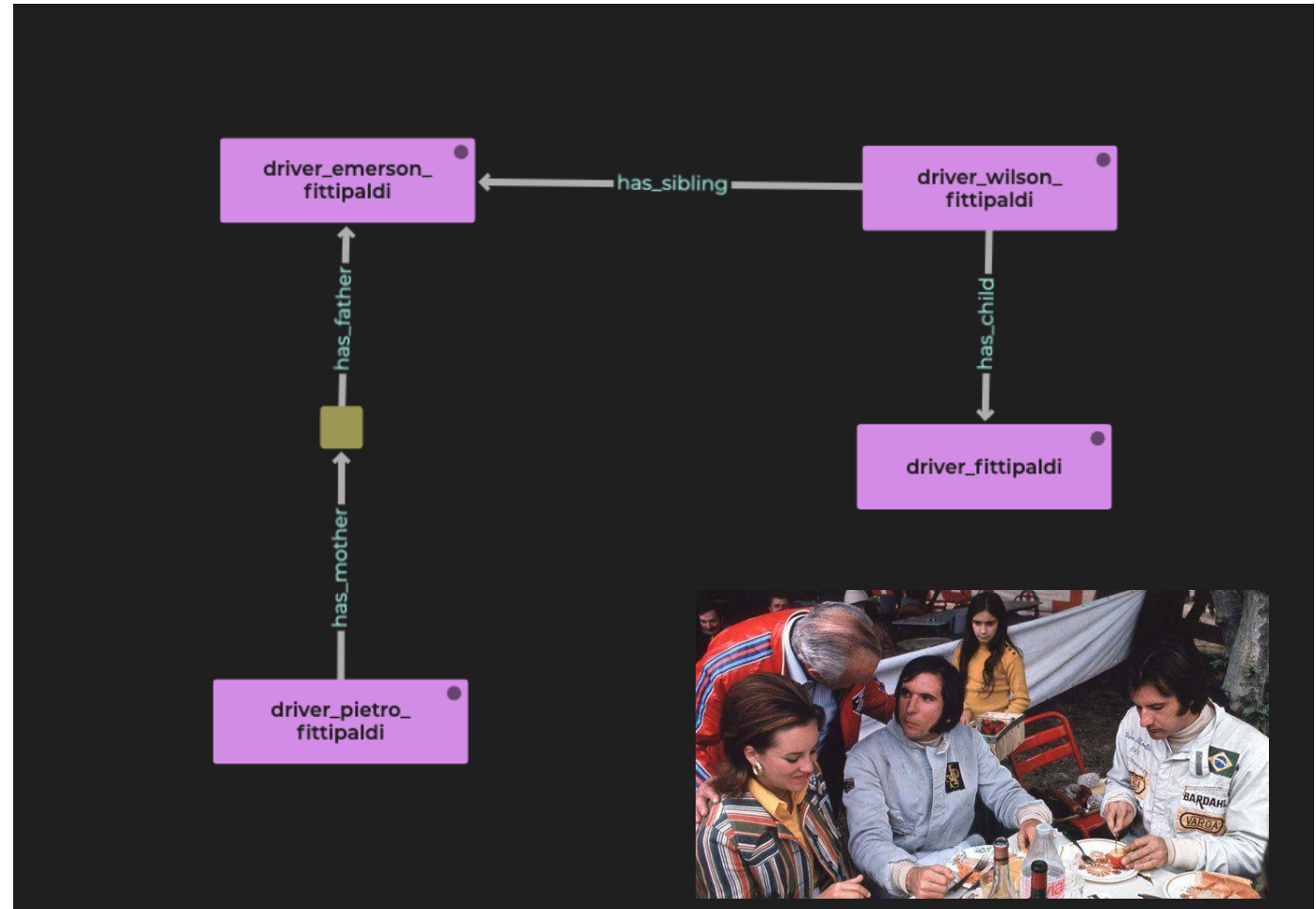
---

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

# Meet the Fittipaldis



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





# Web Ontology Language (OWL)

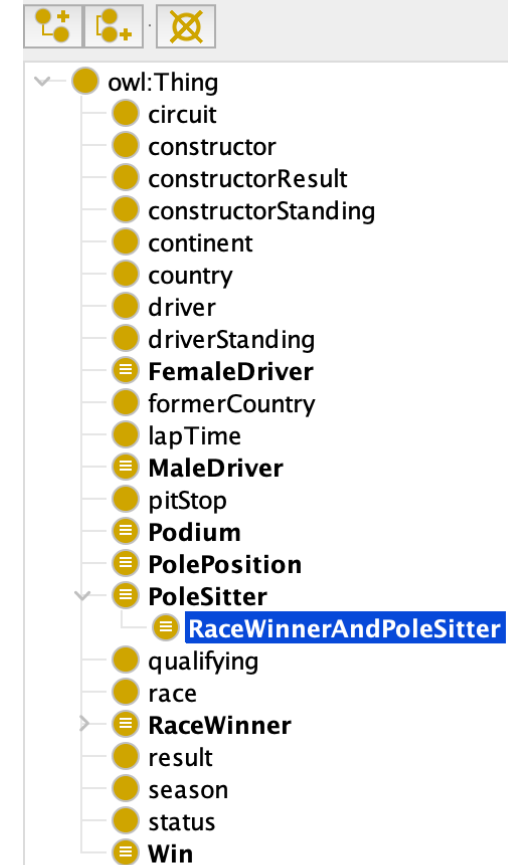
- standard created by W3C, RDFS supports the profile OWL 2 RL
- axioms are assertions about classes, properties or individuals
- ontology are collection of axioms
- ontologies are themselves graphs and can be written in many different syntaxes - we will use Turtle

# Protégé



- 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

## Class hierarchy: RaceWinnerAndPoleSitter





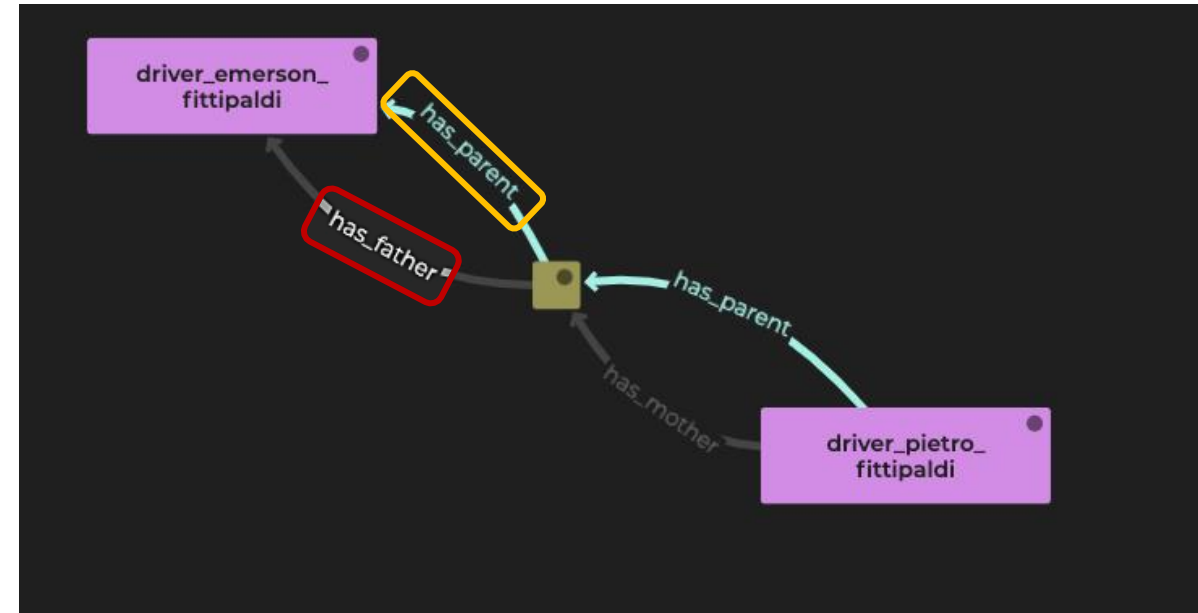
# Axiom Set 1

## Sub Properties

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

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

```
:has_parent a owl:ObjectProperty ;  
owl:inverseOf :has_child ;  
rdfs:subPropertyOf :has_relative .
```



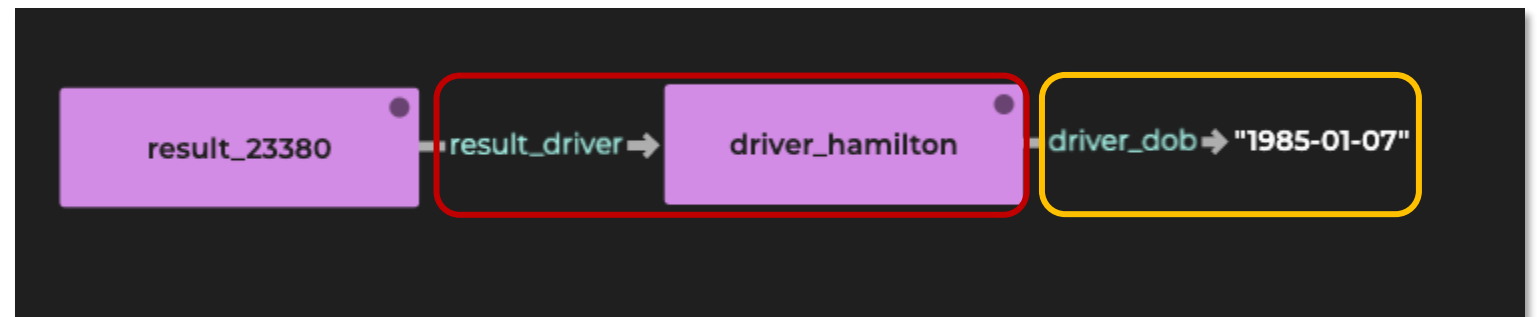


# Axiom Set 2

## Object & Datatype Properties

```
:result_driver rdf:type owl:ObjectProperty;  
rdfs:domain :result ;  
rdfs:range :driver ;  
owl:inverseOf :driver_result .
```

```
:driver_dob rdf:type owl:DatatypeProperty;  
rdfs:domain :driver ;  
rdfs:range xsd:string .
```

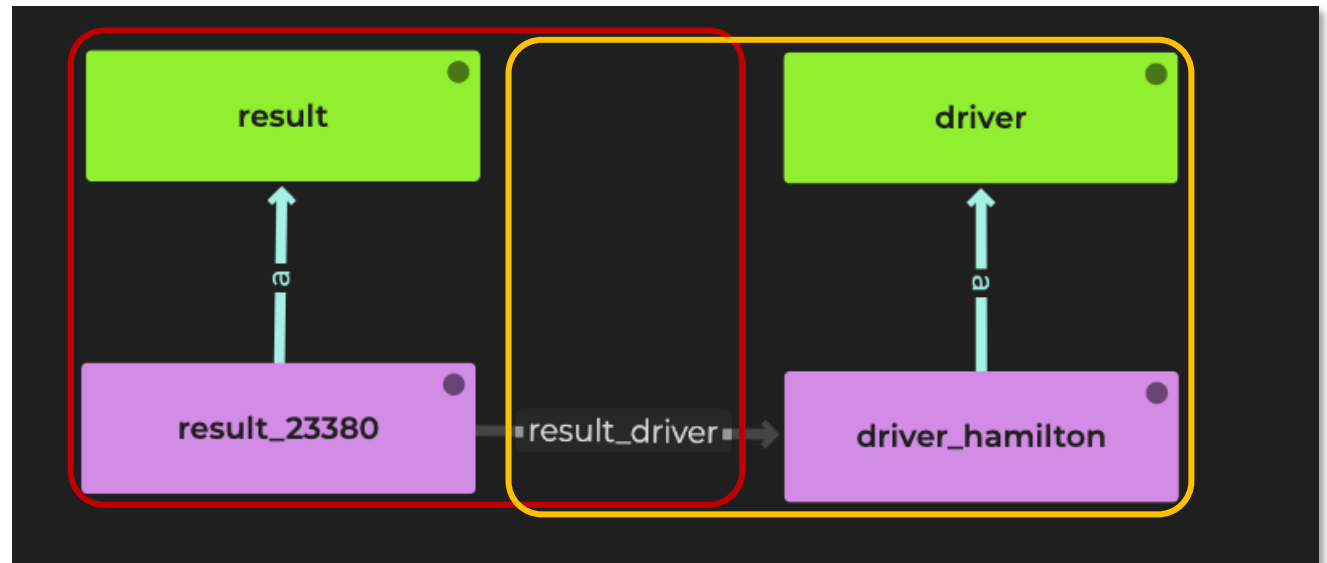


# Axiom Set 3

## Domain & Range



```
:result_driver rdf:type owl:ObjectProperty ;  
rdfs:domain :result ;  
rdfs:range :driver ;  
owl:inverseOf :driver_result .
```

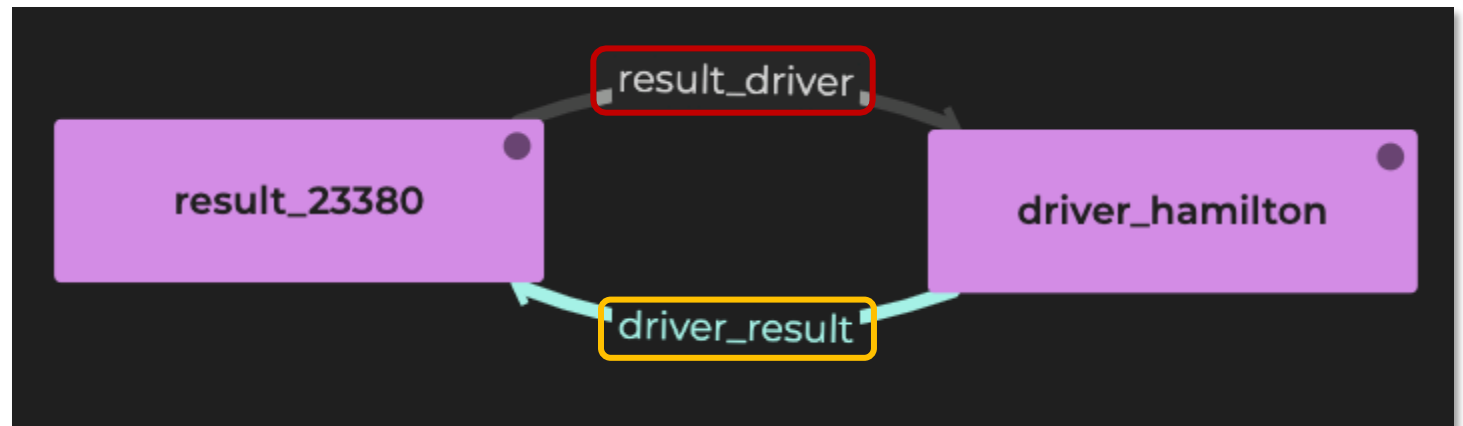


# Axiom Set 4

## Inverse Properties



```
:result_driver rdf:type owl:ObjectProperty ;  
  rdfs:domain :result ;  
  rdfs:range :driver ;  
  owl:inverseOf :driver_result .
```



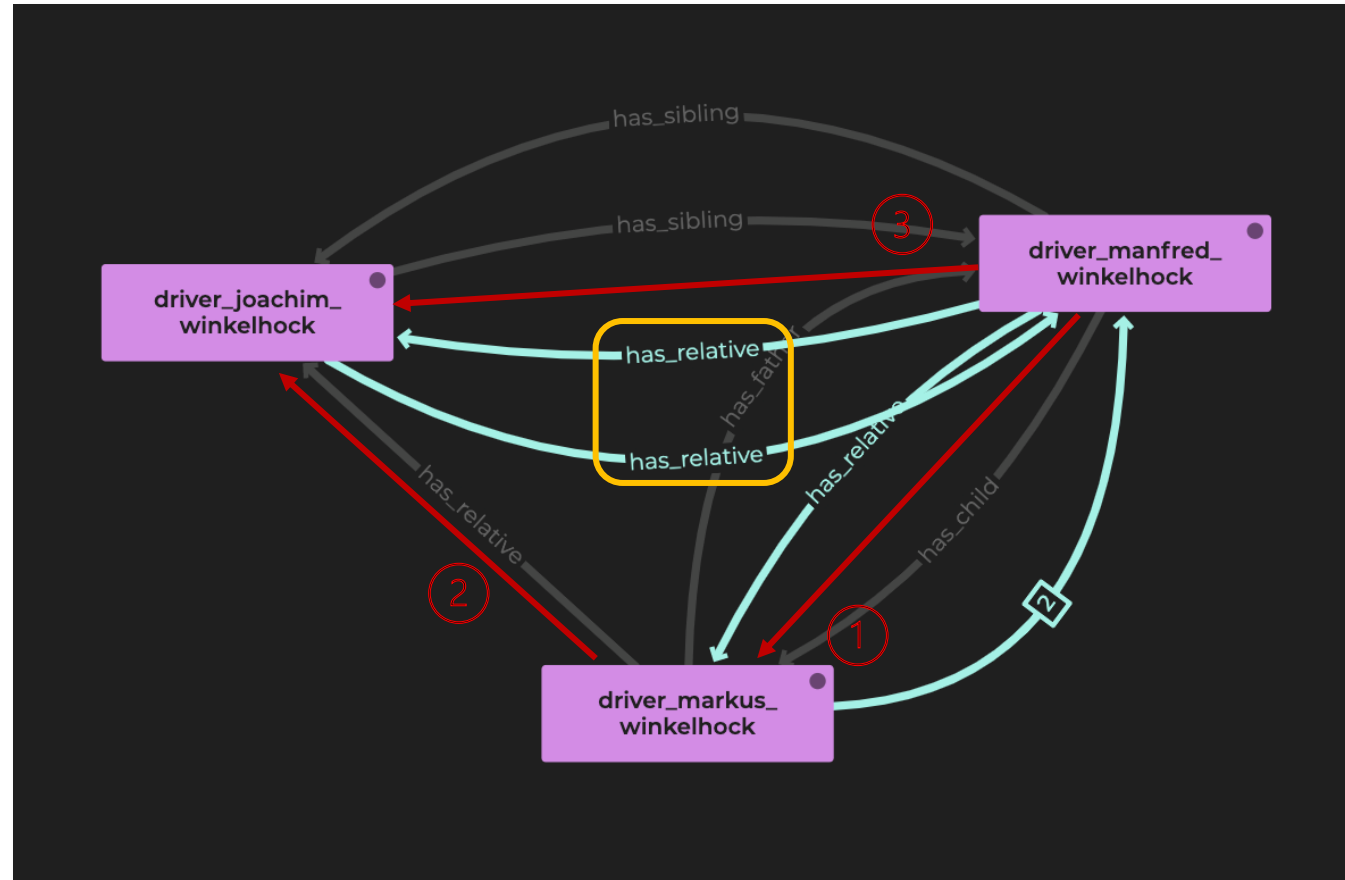


# Axiom Set 5

## Symmetry & Transitivity



:has\_relative a owl:ObjectProperty, owl:TransitiveProperty, owl:SymmetricProperty.

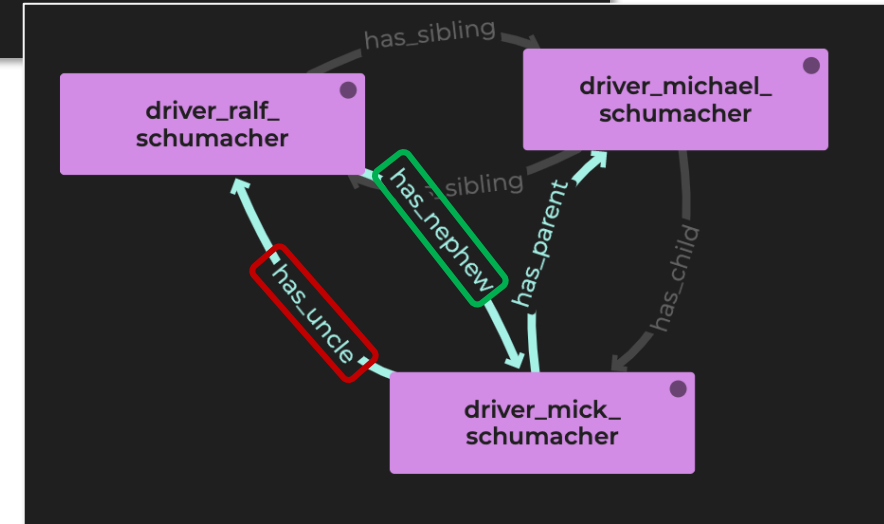
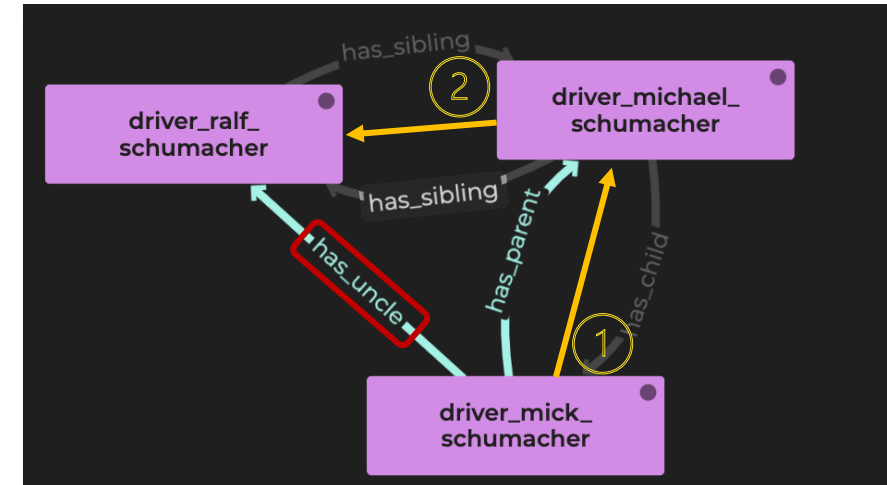


# Axiom Set 6

## Property Chains



```
:has_uncle a owl:ObjectProperty ;  
  rdfs:subPropertyOf :has_relative ;  
  owl:propertyChainAxiom (  
    :has_parent  
    :has_sibling  
  );  
owl:inverseOf :has_nephew.
```

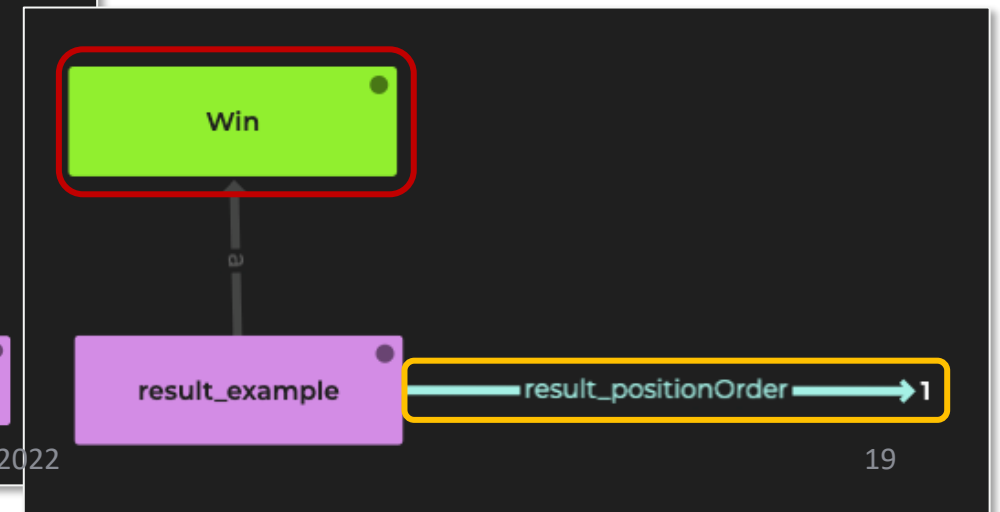
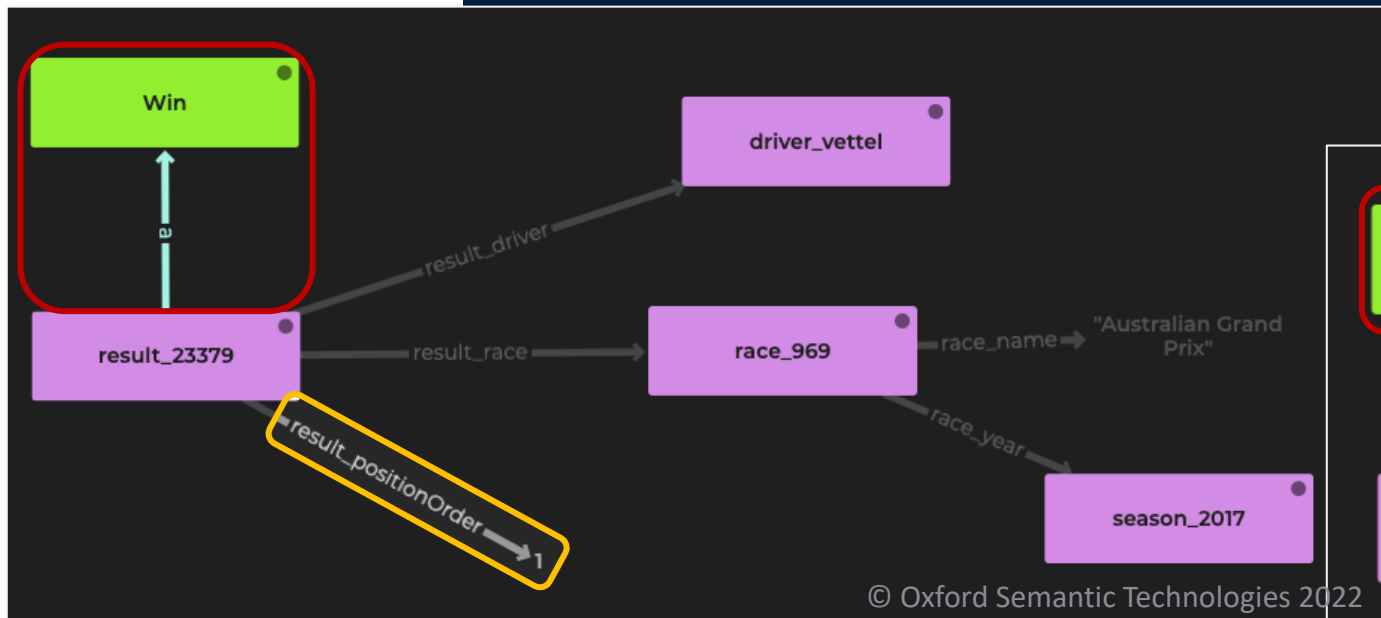




# Axiom Set 7

## Class Equivalence - hasValue

```
:Win a owl:Class ;  
  owl:equivalentClass [  
    a owl:Restriction ;  
    owl:onProperty :result_positionOrder ;  
    owl:hasValue "1"^^xsd:integer  
  ] .
```

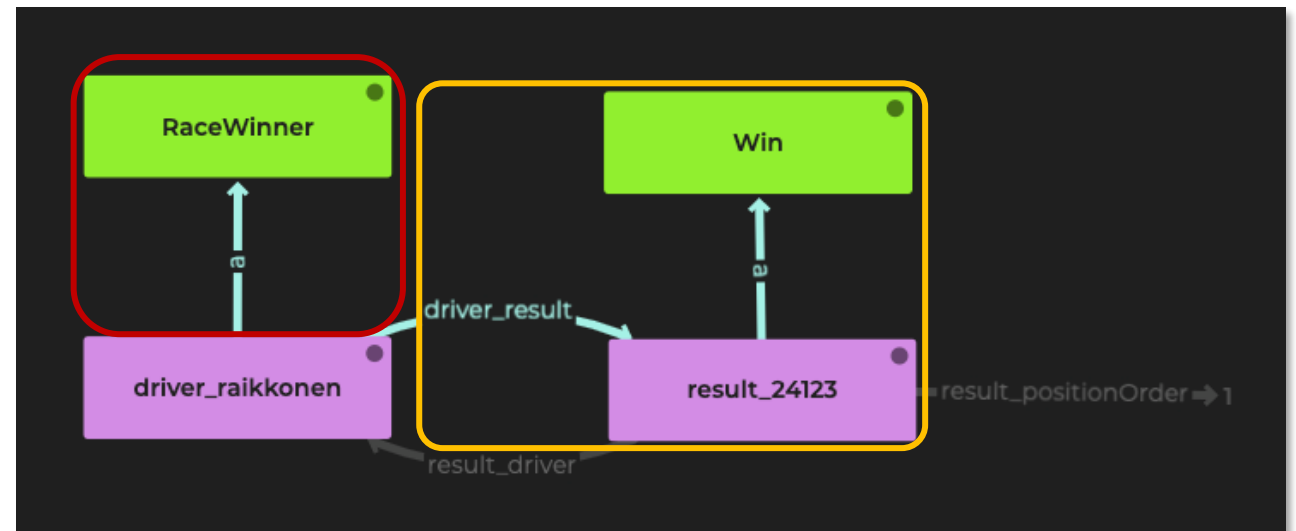




# Axiom Set 8

## Class Equivalence - someValuesFrom

```
:RaceWinner a owl:Class ;  
  owl:equivalentClass [  
    a owl:Restriction ;  
    owl:onProperty :driver_result ;  
    owl:someValuesFrom :Win  
  ] .
```

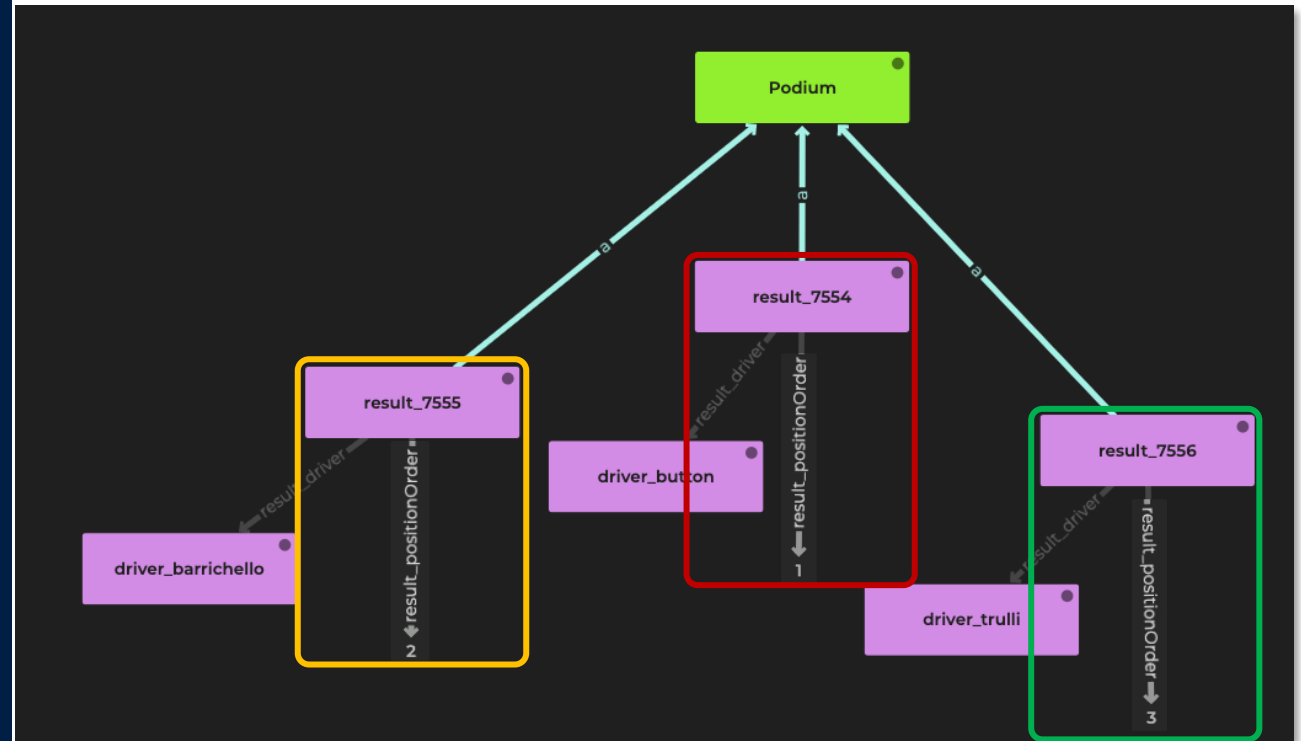


# Axiom Set 9

## Unions



```
:Podium 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  
    ]  
  )  
].
```

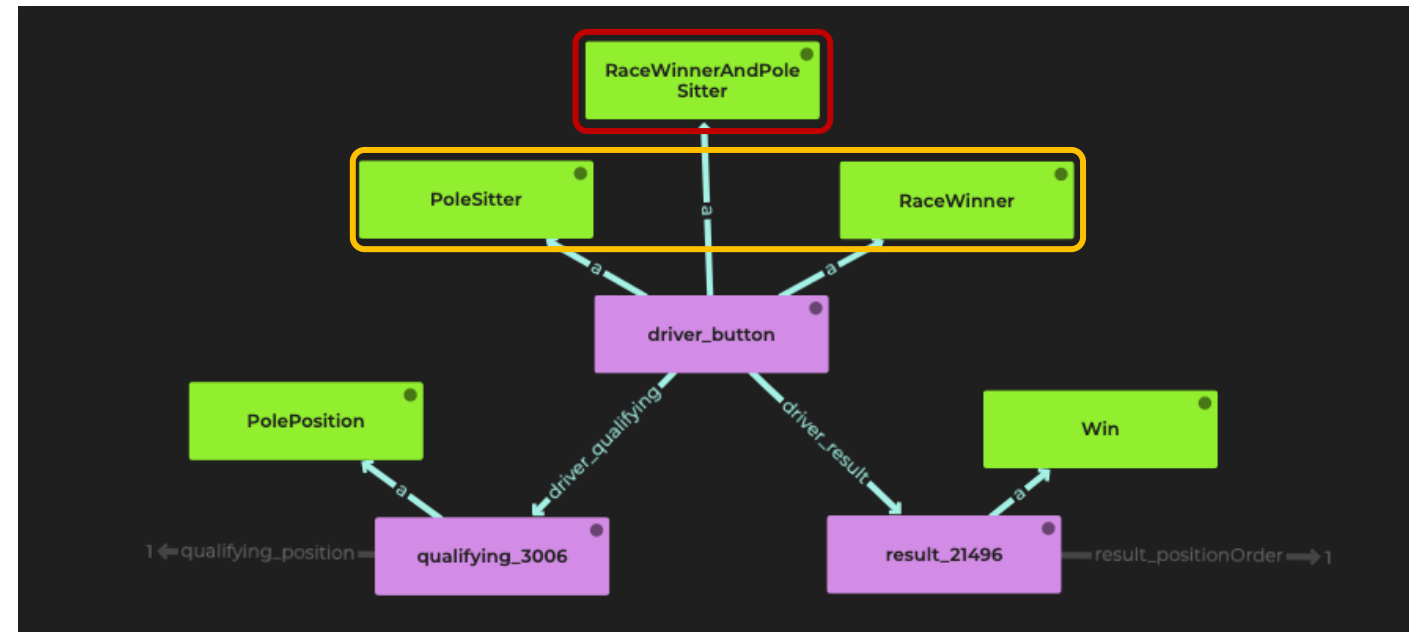




# Axiom Set 10

## Class Intersections

```
: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 warnings 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.





# Fixing Mistakes

**What should I do if I make a mistake?**

```
import axioms :myAxioms -
```

```
update ! clear graph :myAxioms
```

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

```
import axioms :myAxioms
```

# Datalog Reasoning with RDFS

- ❑ 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 **FILTER**s 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 RDFS 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/q18.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 “2021-22.ttl” and try again  
`evaluate queries/q18_1.rq`

# Rule 6 - exercise

## Binds



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

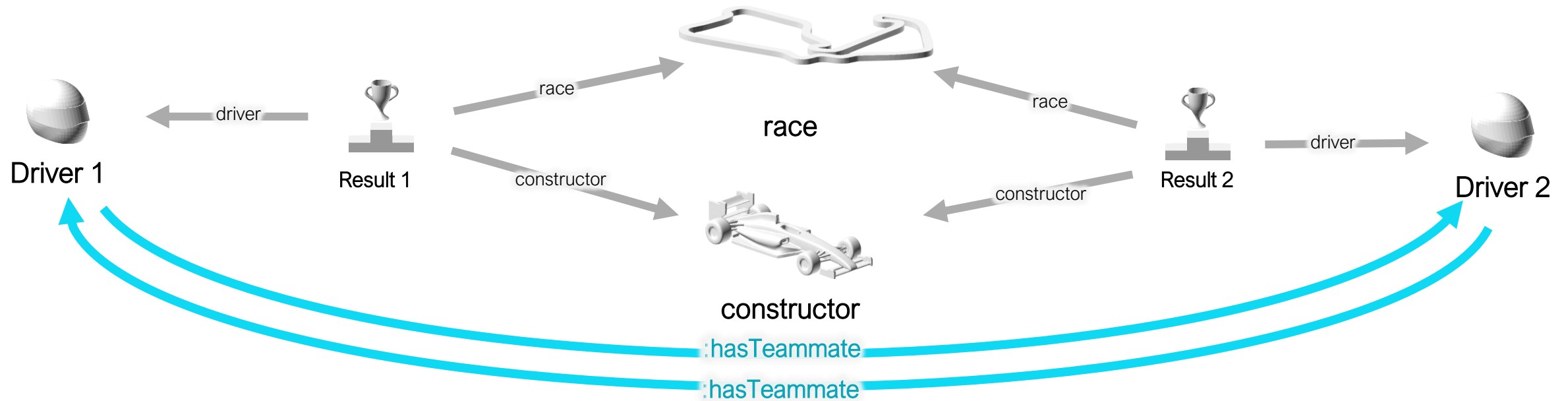
With **BIND** we can use various mathematical functions, string manipulation, regular expression matching, conditional binds, hashing and IRI creation.

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

Check for inferred triples with query 19 – q19.rq

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

# Bonus Exercise

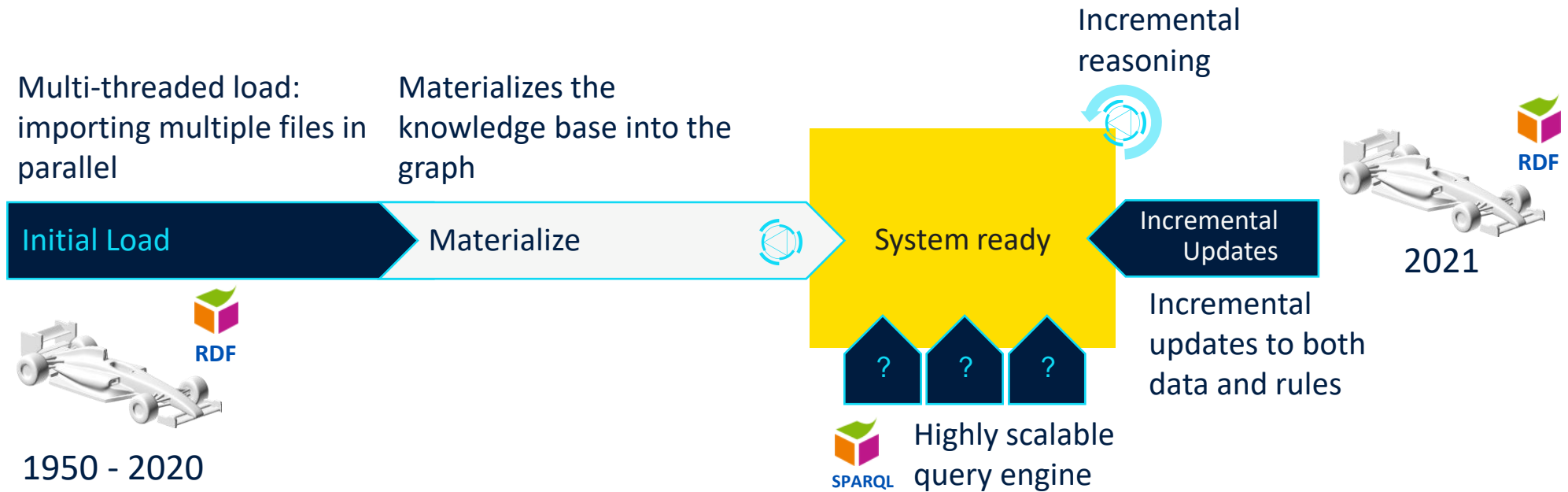


- Write and import a rule to find out if two drivers have been teammates (use the predicate **:hasTeammate**).



# RDFox: How does it work?

# Incremental Reasoning











# Incremental Reasoning

- One of the key strengths of RDFS.
- Data can be imported at any time, and RDFS 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

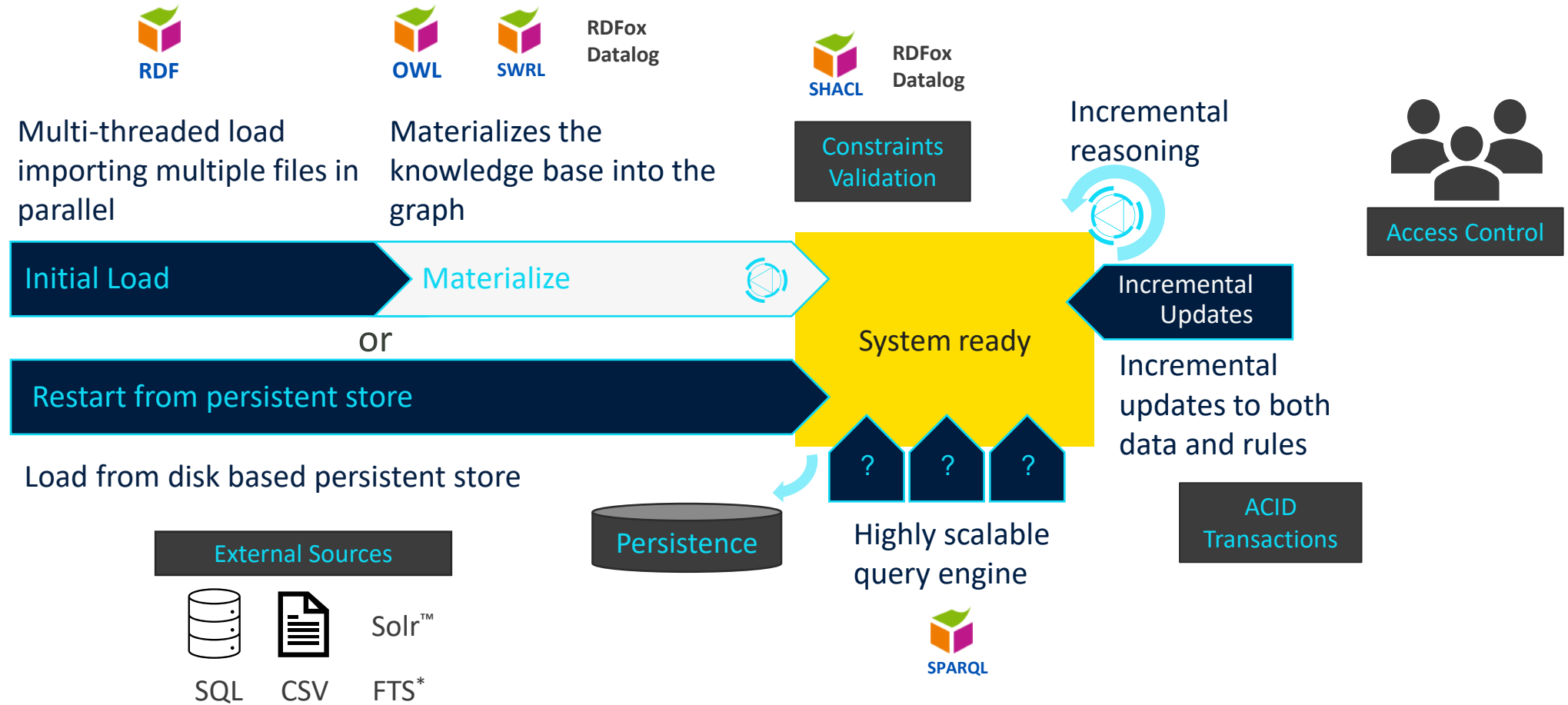


Feature	Example	RDFox	Materialisation Competitors	Query Rewriting Competitors
 Ontological reasoning	“List all financial instruments / derivative instruments / futures.” [class inferencing]	✓	✓*	✓**
 Graph analytics	“What is the nearest electrical switch for a circuit?” [negation + recursion]	✓		
 Data 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

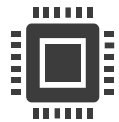
\*\* orders of magnitude slower than RDFox

# How does it work?





# Deployment Options



Edge and mobile devices



Personal machines, on-premises to cloud



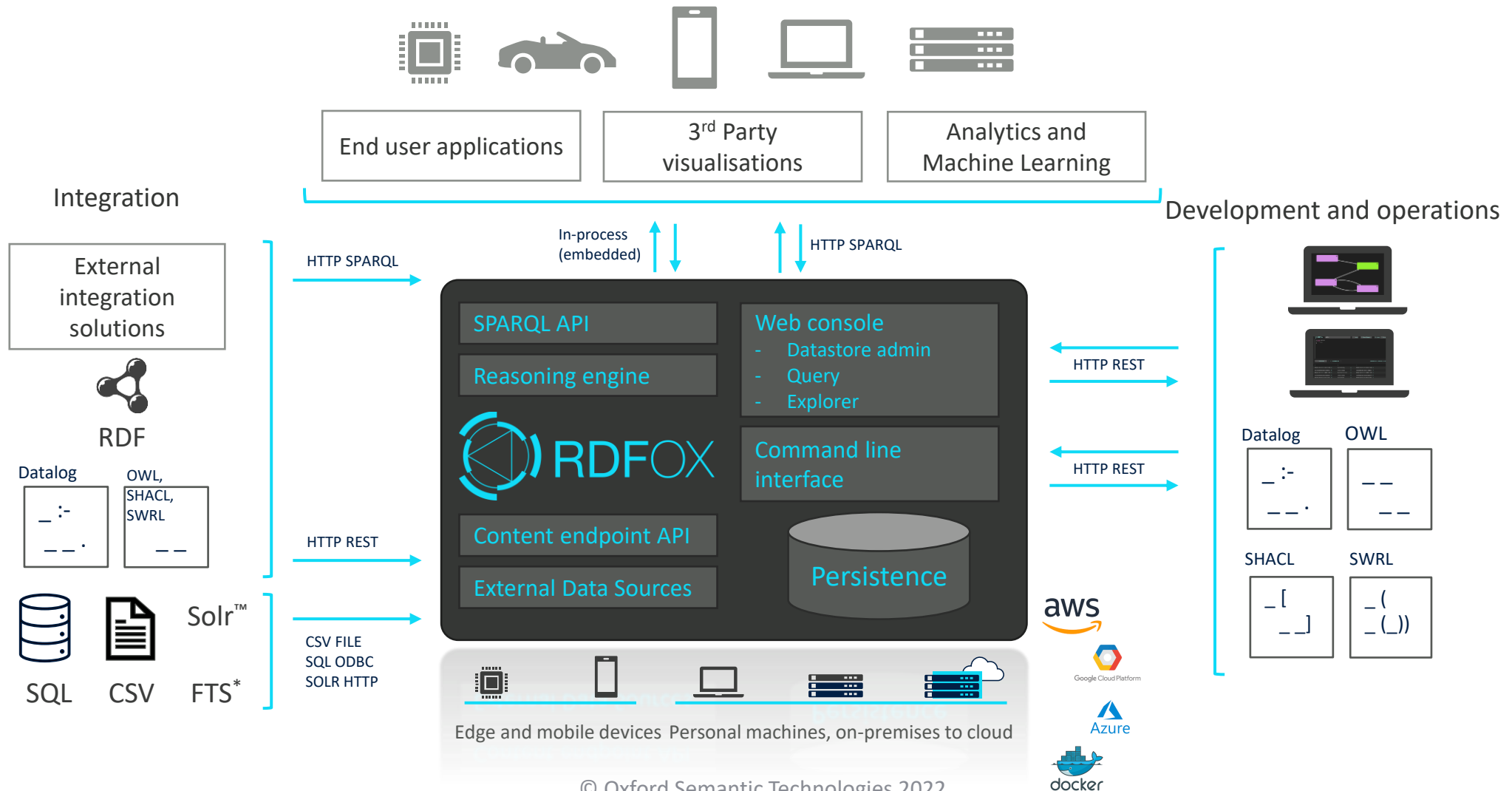
Google Cloud Platform



docker

# Architecture

## RDfox system context diagram





# 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>

# Further resources



Our website

<https://www.oxfordsemantic.tech>

Request an evaluation license

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

Read the documentation

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

Our blog

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

