RDFOX®
Oxford Semantic Technologies

# RDFox Reasoning Workshop

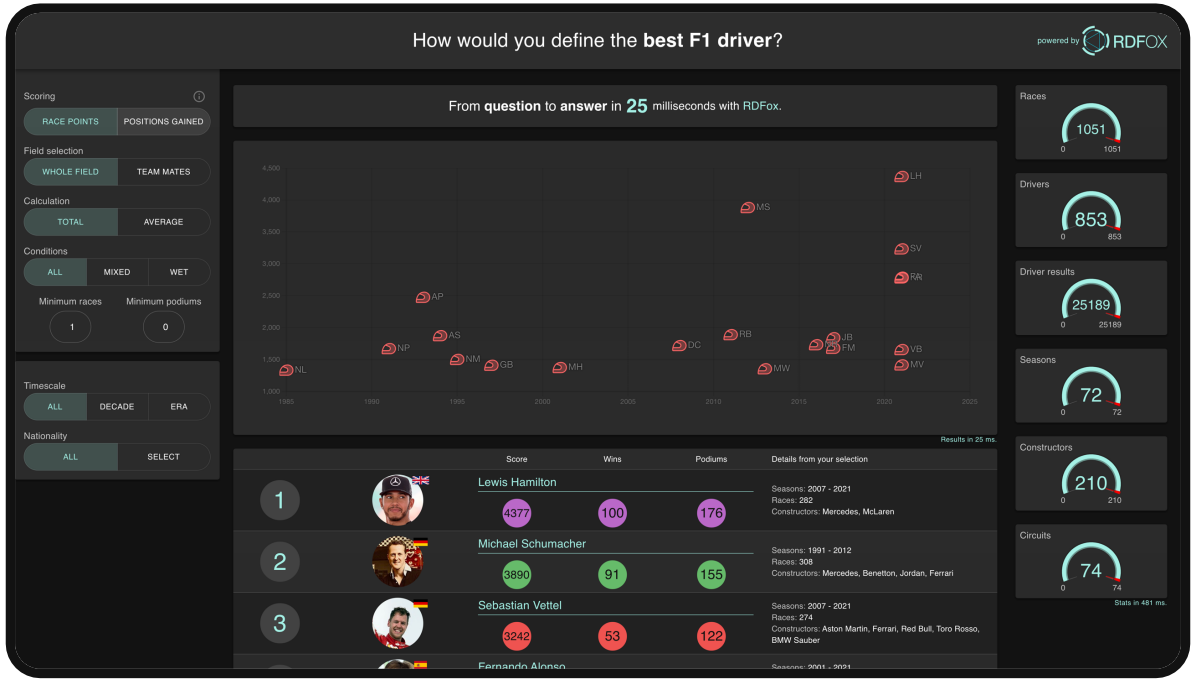

**Tom Vout**

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.rdfox.tech

# Objectives

## The Foundations

- What is Semantic Reasoning?
- What is RDFox?
- Setting up RDFox

## OWL Reasoning

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

## Datalog Reasoning

- Basic rules
- Filters
- Aggregates
- Negation
- Binds
- Incremental Reasoning
- Materialization vs. query rewriting
- RDFox in enterprise

Please feel free to ask questions at any time! Exercises are scattered throughout. Links are provided for reference.

https://docs.oxfordsemantic.tech/

# Objectives

## The Foundations

- What is Semantic Reasoning?
- What is RDFox?
- Setting up RDFox

## OWL Reasoning

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

## Datalog Reasoning

- Basic rules
- Filters
- Aggregates
- Negation
- Binds
- Incremental Reasoning
- Materialization vs. query rewriting
- RDFox in enterprise

# What is Semantic Reasoning?

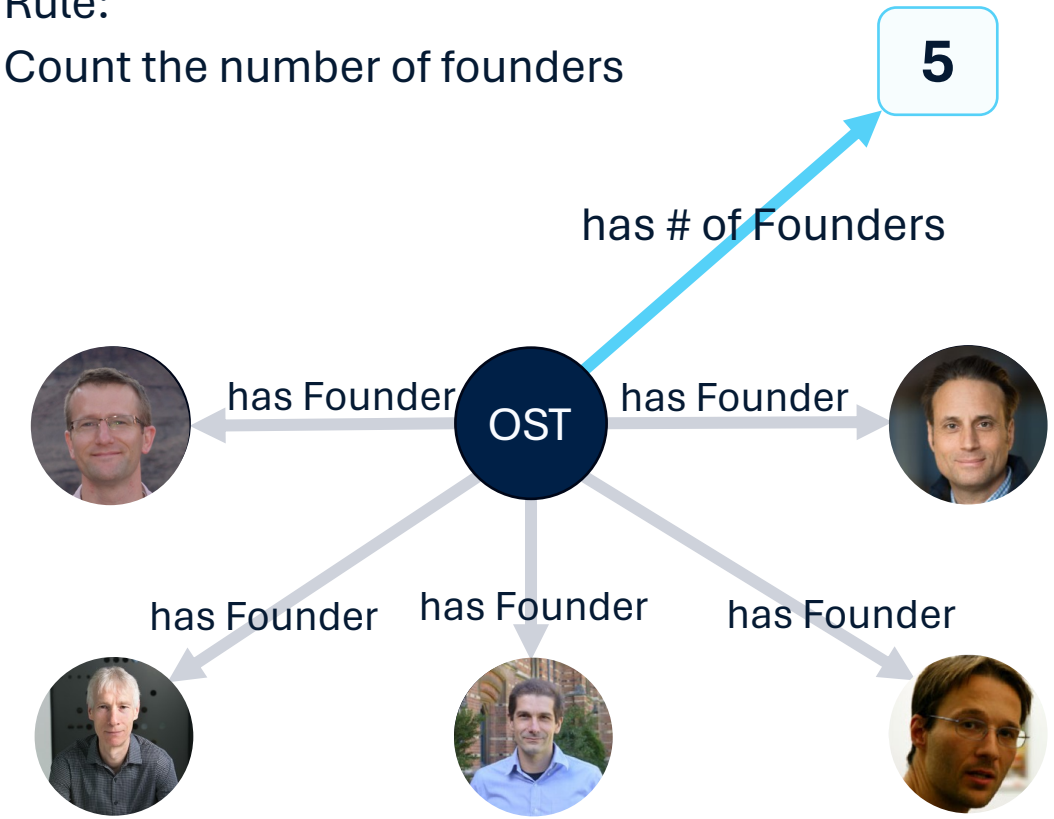
Semantic Reasoning (rules-based AI) enriches a Knowledge Graph by adding new information

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?

```
#Query 11
PREFIX : <http://www.oxfordsemantic.tech/f1demo/>

# Drivers with their win percentage, ordered by win percentage
SELECT ?forename ?surname ?raceCount ?raceWins ?percentage
WHERE {
# First get the drivers
?driver :driver_forename ?forename ;
        :driver_surname ?surname .

# Then get the race count for each driver with an innery query...
    {SELECT ?driver (COUNT(?race) AS ?raceCount)
    WHERE {
    ?result :result_driver ?driver ;
            :result_race ?race .
    }
    GROUP BY ?driver}

# ... and get the *win* count for each driver with another inner query.
    {SELECT ?driver (COUNT(?race) AS ?raceWins)
    WHERE {
    ?result :result_driver ?driver ;
            :result_race ?race ;
            :result_positionOrder 1 .
    }
    GROUP BY ?driver}

# Finally use the two aggregate variables to compute a percentage
# with the BIND keyword.
BIND(?raceWins/?raceCount AS ?percentage)

}
ORDER BY DESC(?percentage)
```

Fetched 108 answers in 0.011 s.

**Datalog rules**

```
#Query 19
PREFIX : <http://www.oxfordsemantic.tech/f1demo/>

SELECT ?forename ?surname ?percentage
WHERE {
    ?driver :driver_forename ?forename;
        :driver_surname ?surname;
        :hasWinPercentage ?percentage;
        :hasRaceCount ?count .
}
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.oxfordsemantic.tech/free-trial

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

https://www.oxfordsemantic.tech/downloads

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

https://github.com/OxfordSemantic/RDFoxWorkshop

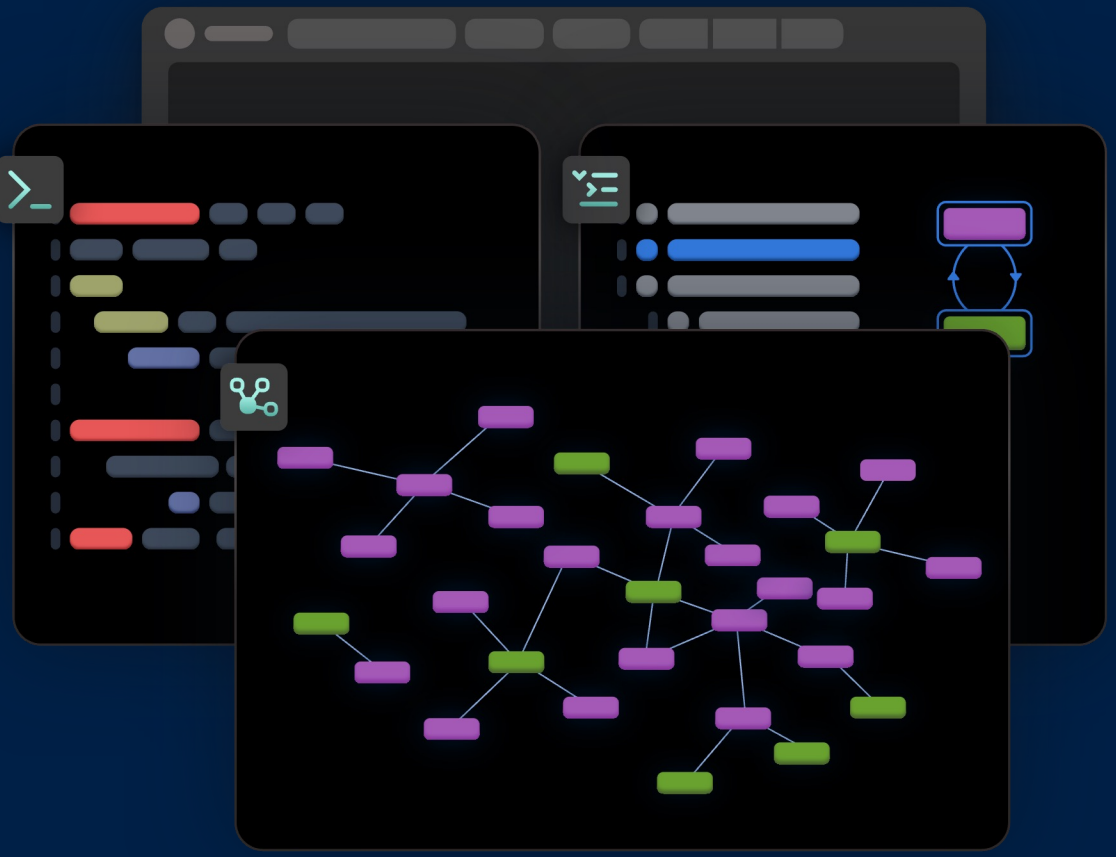
## D.
## Get your IDE of choice ready

**(VS Code recommended)**

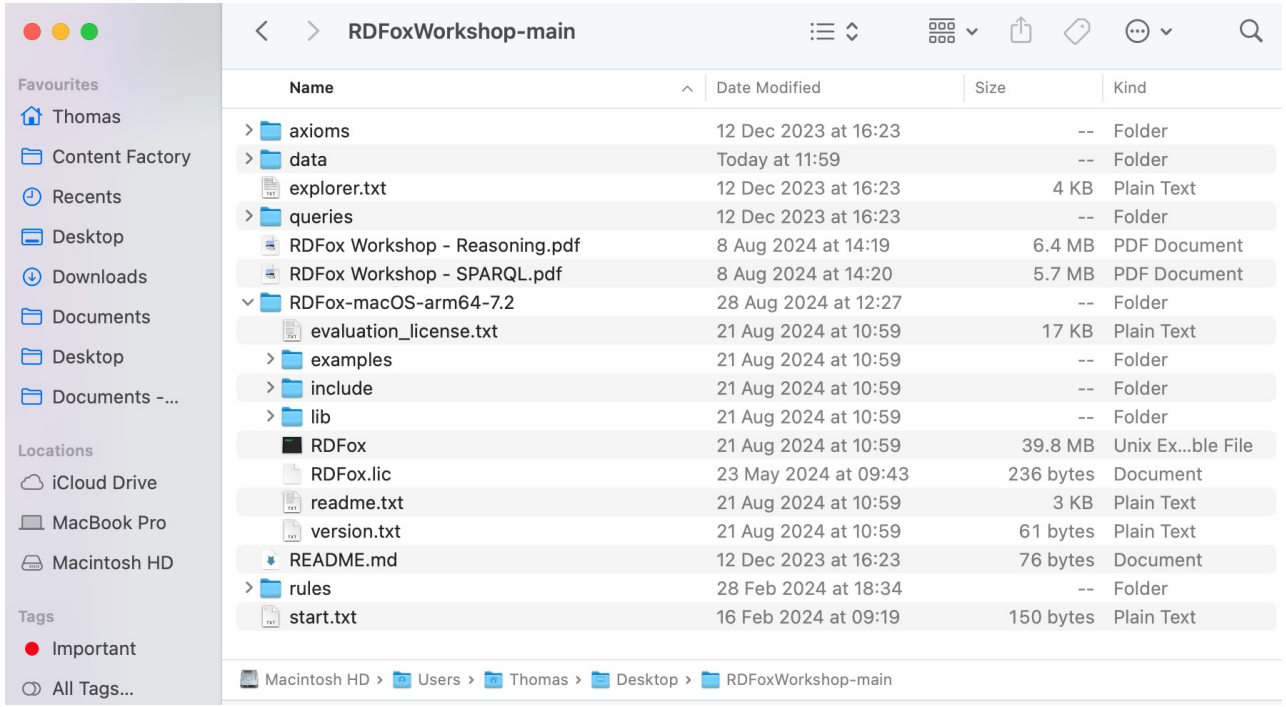
https://code.visualstudio.com

# Setting up RDFox

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



# Organizing your files




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

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

   **Your workshop folder will be called 'RDFoxWorkshop' if you cloned it (rather than downloading it)**

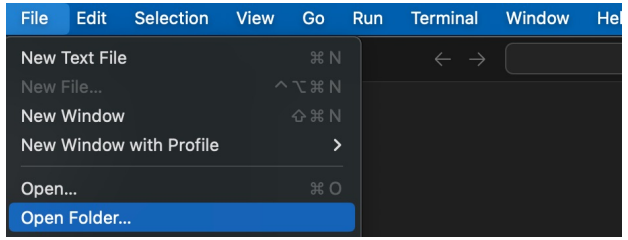
2. Then drop your **RDFox license** INSIDE the **RDFox folder**

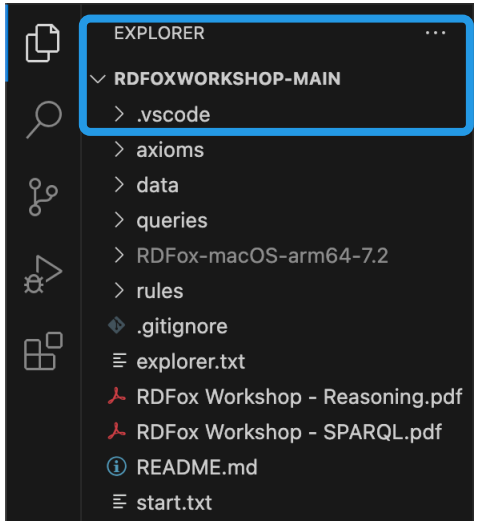
# Opening the terminal in VS Code

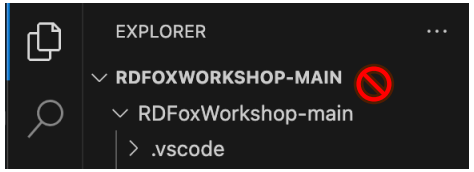
1. Open VS Code

2. Open the workshop folder with
**File > Open Folder**

Select the folder directly containing the ‘data’, ‘rules’, ‘queries’ folders etc.

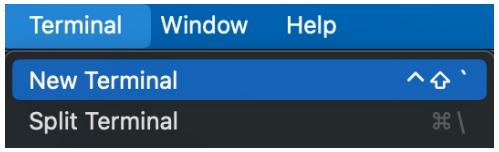


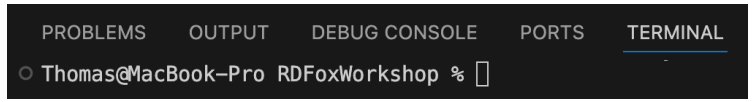




If the folder name appears twice, repeat step 2 and open the inner-most folder

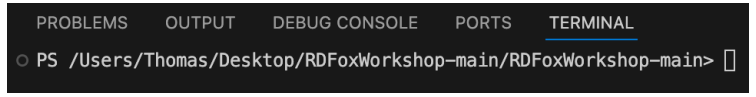
3. Open a new terminal with
**Terminal > New Terminal**




› E.g. MacOS (zsh)




› E.g. Windows (PowerShell)




This path is unique to you

# Starting RDFox

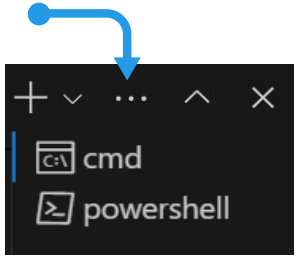
1. In the terminal, from the ‘RDFox Workshop’ working directory,
run the relevant command for the version of RDFox you downloaded:

**You will need to edit the command if you are not using RDFox v7.2 or are running on LINUX**

- MacOS ARM
```
./RDFox-macOS-arm64-7.2/RDFox sandbox
```
- MacOS INTEL
```
./RDFox-macOS-x86_64-7.2/RDFox sandbox
```

**The Windows command depends on the shell used – shown right of the terminal window**

- Windows (CMD)
```
RDFox-win64-x86_64-7.2\RDFox.exe sandbox
```
- Windows (PowerShell)
```
& RDFox-win64-x86_64-7.2\RDFox sandbox
```




2. The RDFox server should now be running!

**If not, see next slide for common fixes**

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

# Common fixes

## File path needs quotes and/or backslash

All systems

```
command not found
is not recognised
permission denied
is a directory
```

Problem: Some terminals handle slashes and special characters differently
Fix: Surround the file path (excluding ‘sandbox’) with quotation marks ”” and, on Windows, replace forward-slashes ‘/’ with back-slashes ‘\’

## Unzipping creates a nested folder

Predominantly Windows

```
The system cannot find the path specified.
The term is not recognized
```

Problem: Unzipping a folder sometimes puts the contents in a folder of its own name
Fix: Use the inner ‘RDFoxWorkshop’ as the working directory (open VS code at the inner folder)

## Check your folder structure

All systems

Ensure your folder structure follows RDFoxWorkshop > RDFox-macOS-arm64-7.2 > RDFox.lic

## Check your working directory and spelling

All systems

Ensure your terminal is running from the RDFoxWorkshop folder and your commands are correct



If you are still encountering issues, please raise your hand

# 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:

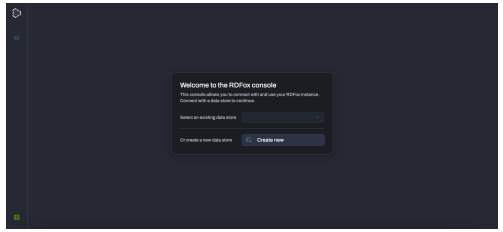
1. In the terminal, run: `endpoint start`

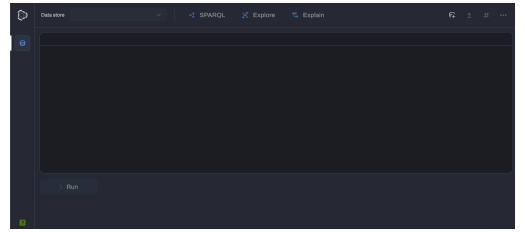
```
> endpoint start
The REST endpoint was successfully started at port number/service name 12110 with 8 threads.
WARNING: The RDFox endpoint is running with no transport layer security (TLS). This could allow attackers to steal
         information including role passwords or authorization tokens. See the endpoint.channel variable and related
         variables in the description of the RDFox endpoint for details of how to set up TLS.

> 
```

2. Open a browser and go to: `localhost:12110/console`

3. This will show an empty console



# Objectives

## The Foundations

- ✓ What is Semantic Reasoning?
- ✓ What is RDFox?
- ✓ Setting up RDFox

## OWL Reasoning

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

## Datalog Reasoning

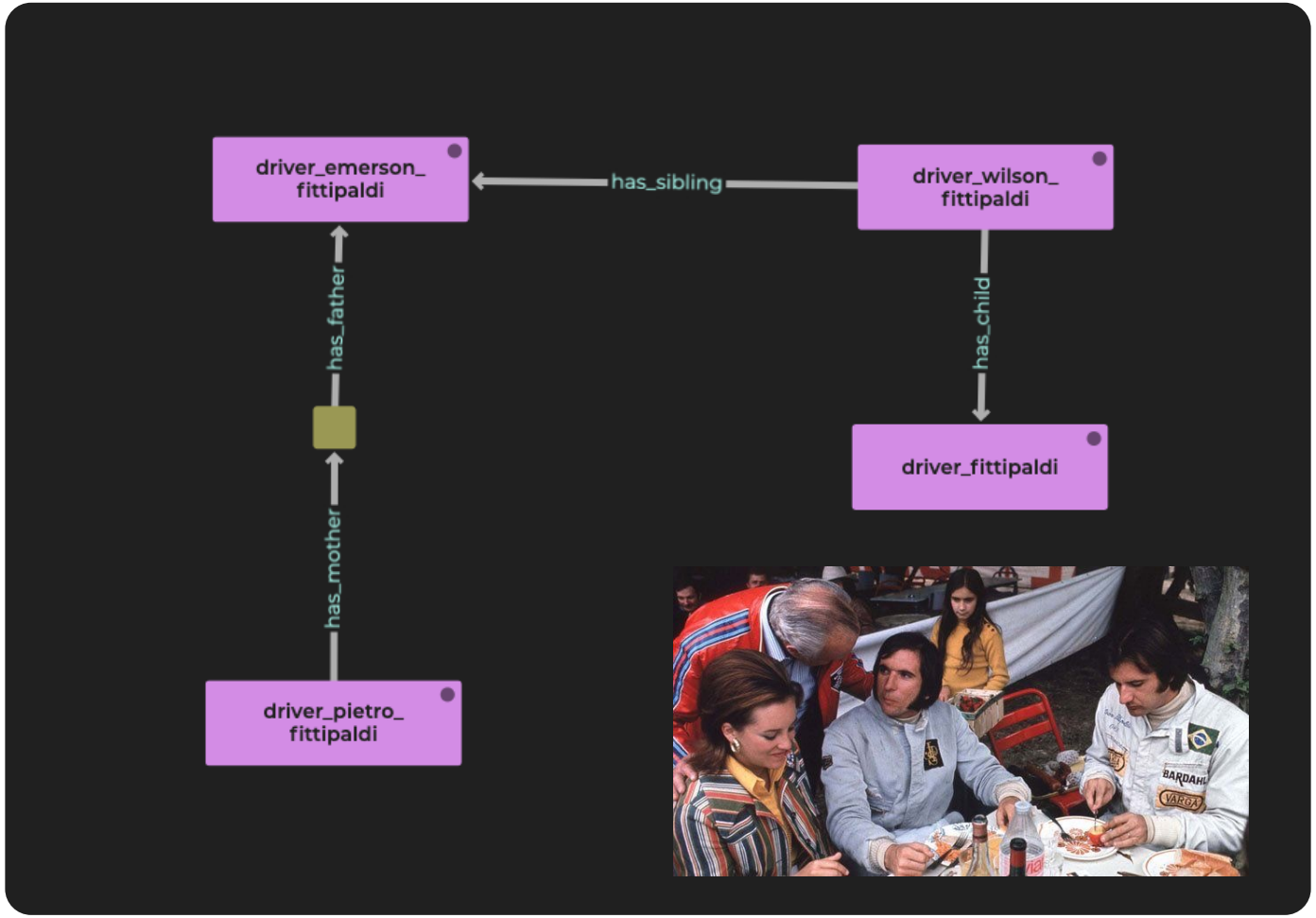
- ○ Basic rules
- ○ Filters
- ○ Aggregates
- ○ Negation
- ○ Binds
- ○ Incremental Reasoning
- ○ Materialization vs. query rewriting
- ○ RDFox in enterprise

# OWL Reasoning

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

# Meet the Fittipaldis

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

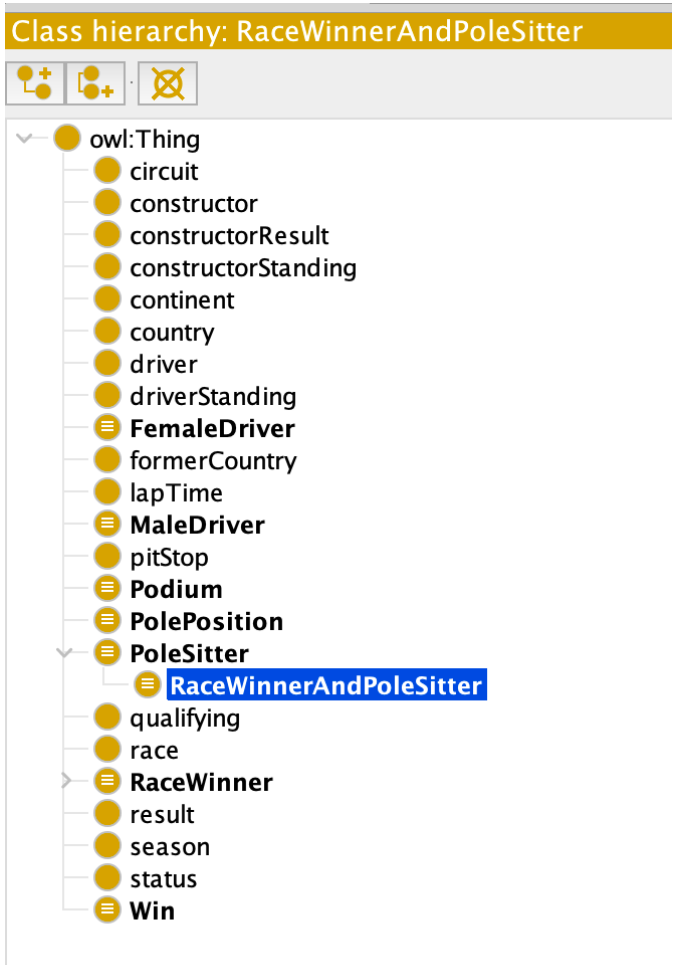


# Web Ontology Language (OWL)

- **Standard created by W3C**, 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

# 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



# Axiom Set 1

## Object & Datatype Properties

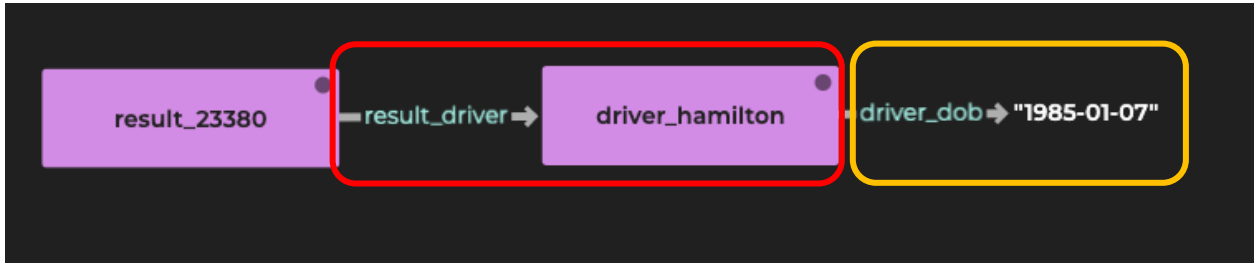
Predicates have two different types.

```
:result_driver rdf:type owl:ObjectProperty .

:driver_dob rdf:type owl:DatatypeProperty .
```

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

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




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

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

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

# 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**.

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


As in SPARQL, the first entry can be shared across multiple lines, with a ';'.

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.

The **range** of a predicate is the group of nodes that exist in its **object position**.

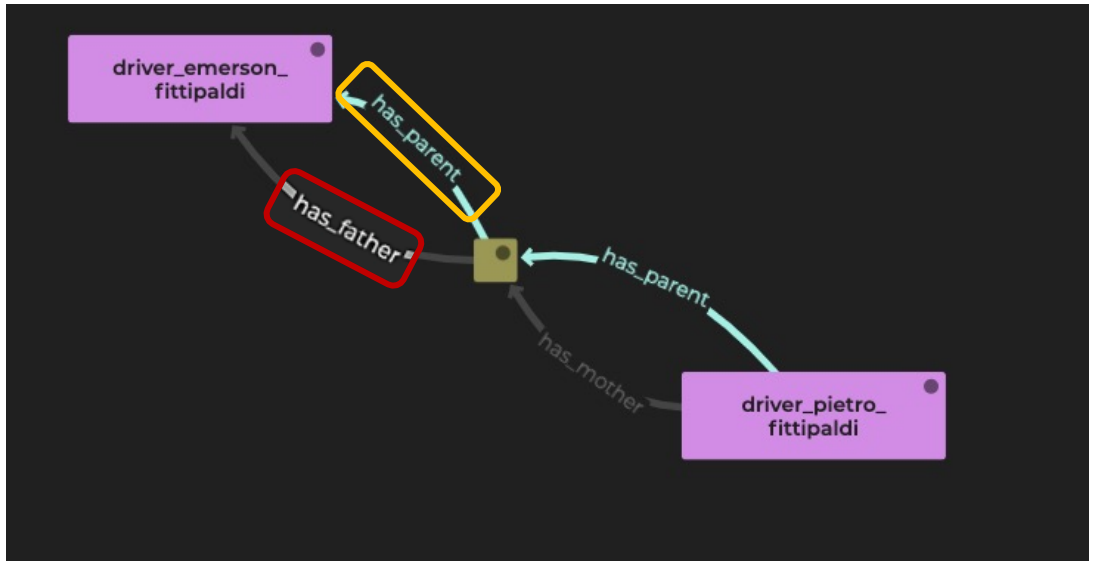
# Axiom Set 3

## 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

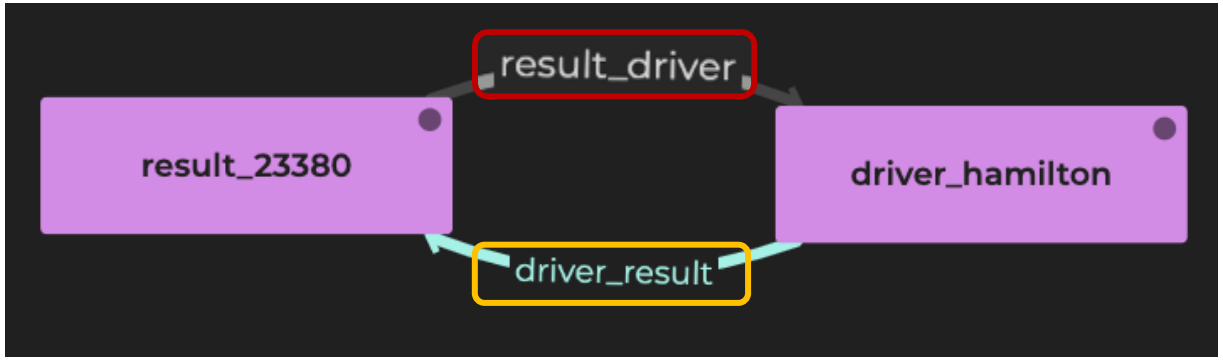
## Inverse Properties

result_driver is the opposite of driver_result.

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

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

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



# Axiom Set 5

## Symmetry & Transitivity

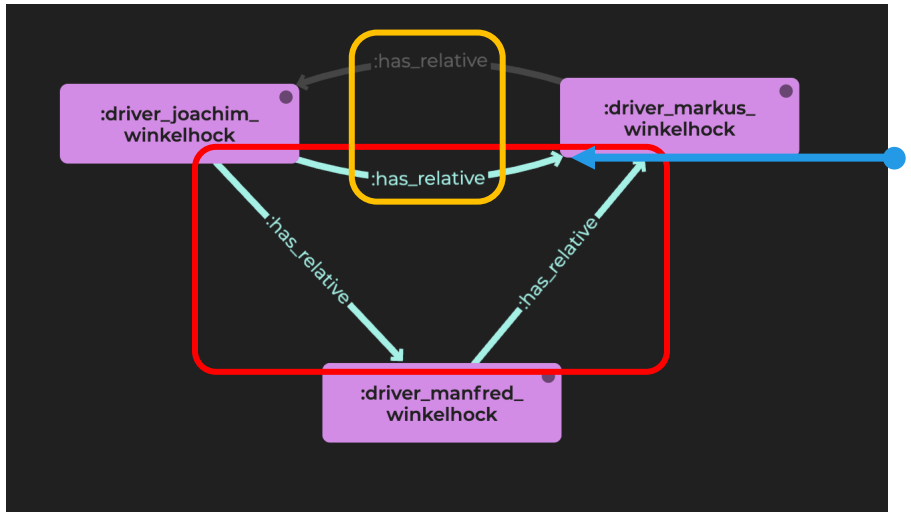
Family relatives share the same relationships in many ways.

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

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

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

**Symmetric Properties** are inferred in **both directions** between a pair of nodes.




The same fact can be derived in multiple ways.

RDFox will handle duplicates and only include one.

Some edges have been hidden from this diagram for clarity.

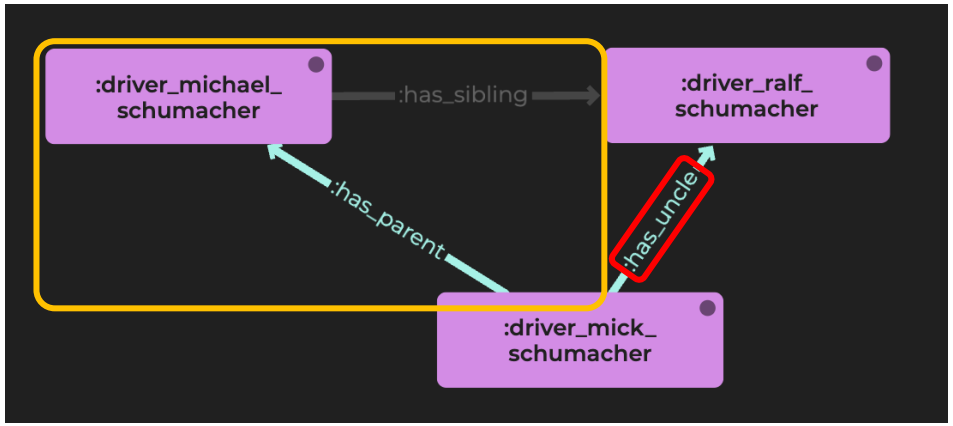
# Axiom Set 6

## 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.

```
:has_uncle a owl:ObjectProperty ;
        owl:propertyChainAxiom (
                :has_parent
                :has_sibling
        ).
```




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**.

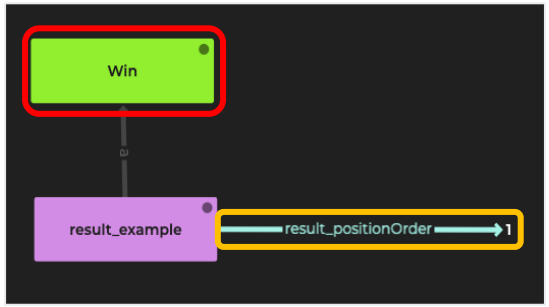
# Axiom Set 7

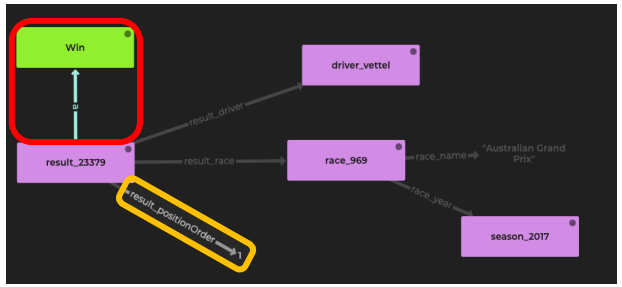
## Class Equivalence

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.




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

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

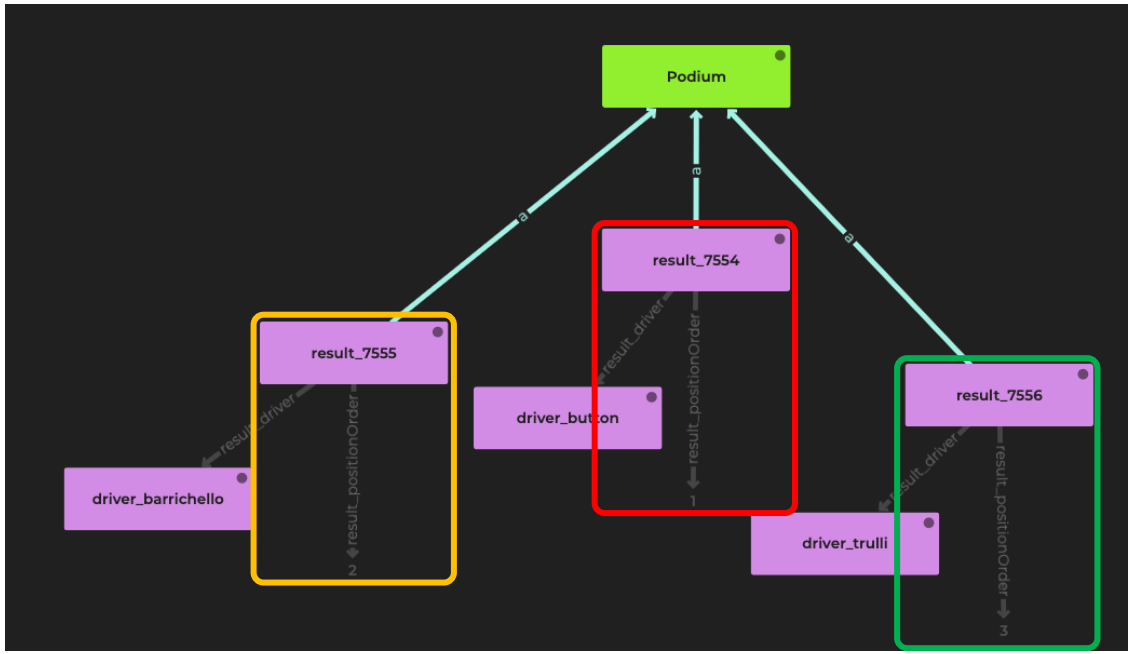
## Unions

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
      ]
    )
  ] .
```

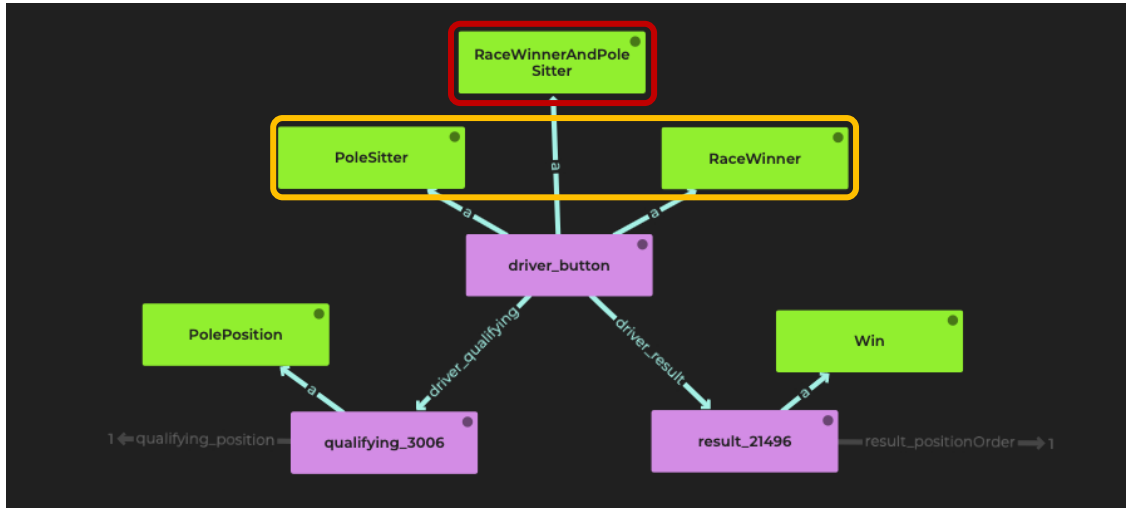
# Axiom Set 9

## 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
```

```
importaxioms :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 out data store:

```
importaxioms :myAxioms -
```

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

# Objectives

## The Foundations

- ✓ What is Semantic Reasoning?
- ✓ What is RDFox?
- ✓ Setting up RDFox

## OWL Reasoning

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

## Datalog Reasoning

- ○ Basic rules
- ○ Filters
- ○ Aggregates
- ○ Negation
- ○ Binds
- ○ Incremental Reasoning
- ○ Materialization vs. query rewriting
- ○ RDFox in enterprise

# Datalog Reasoning with RDFox

- Basic rules
- Filters
- Aggregates
- Negation
- Binds
- Incremental reasoning
- Materialization vs. query rewriting
- RDFox in enterprise

# 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 on 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) .
```

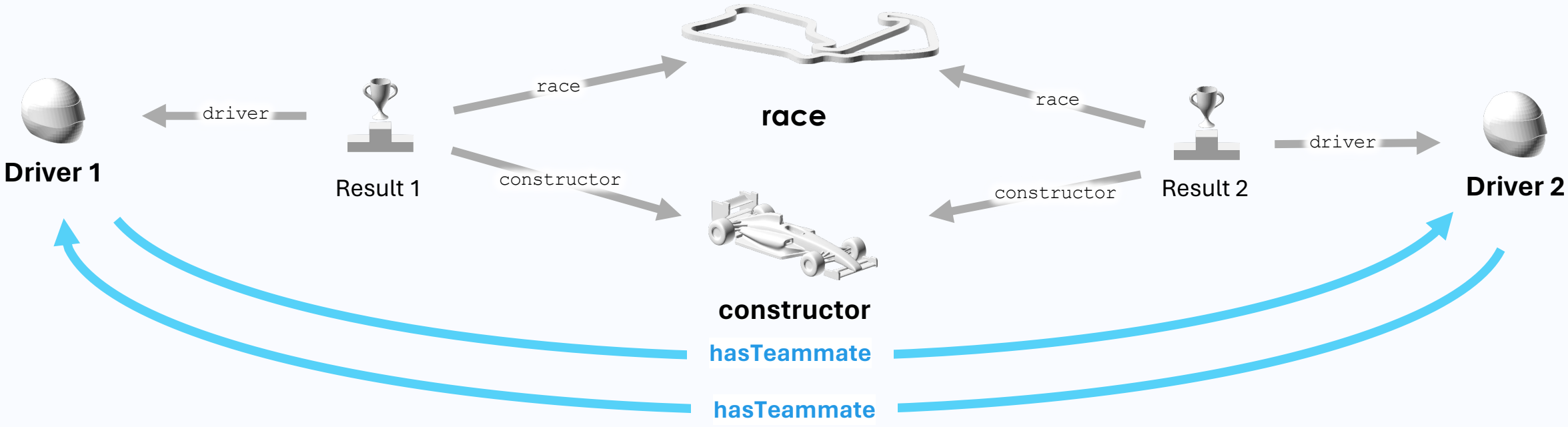
- With BIND we can use various mathematical functions, string manipulation, regular expression matching, conditional binds, hashing and IRI creation.

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

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

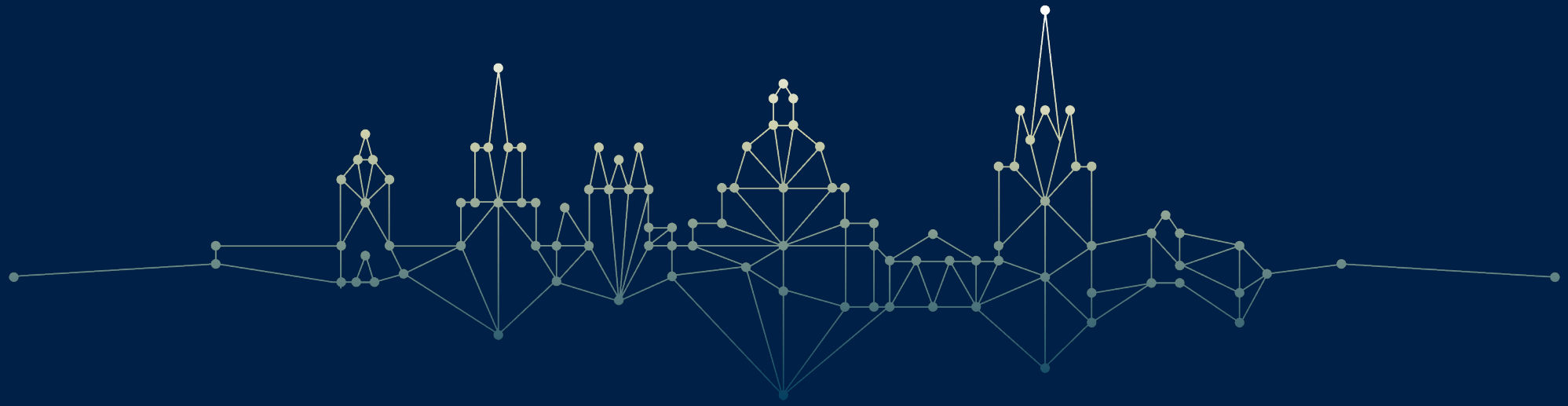
```
evaluate queries/q19.rq
```

# Bonus Exercise



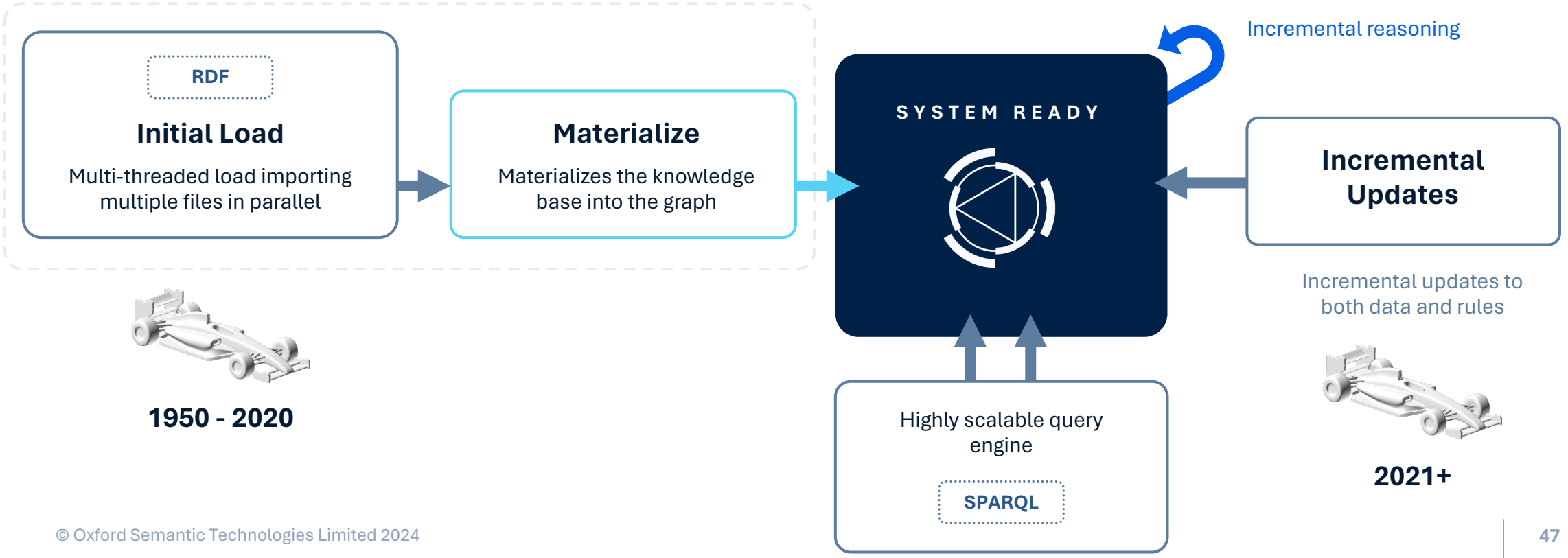

Write and import a rule to find out if two drivers have been teammates (use the predicate **:hasTeammate**).

# RDFox: How does it work?



# Incremental Reasoning

- 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.

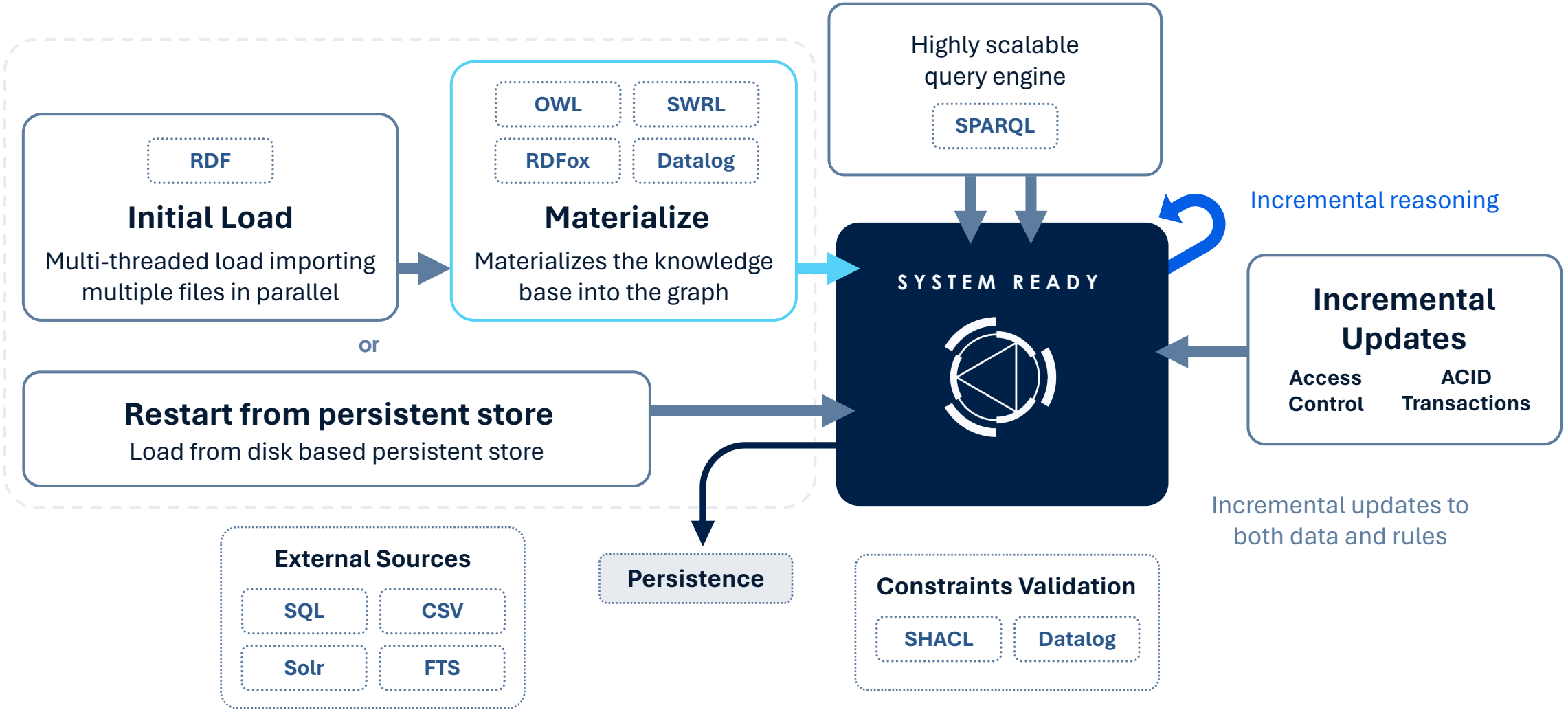


# RDFox vs. other solutions

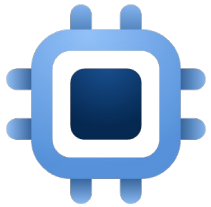
| Feature | Example | RDFox | Materialisation Competitors | Query Rewriting Competitors |
|---|---|---|---|---|
| **Ontological reasoning** | List all financial instruments / derivative instruments / futures. **(Class inferencing)** | ✓ | ✓ * | ✓ ** |
| **Graph analytics** | What is the nearest electrical 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 valudation** | 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

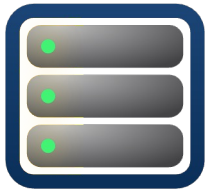
# RDFox : an enterprise solution



# Deployment Options



## Mobile & edge devices



## Personal machines on premises to cloud

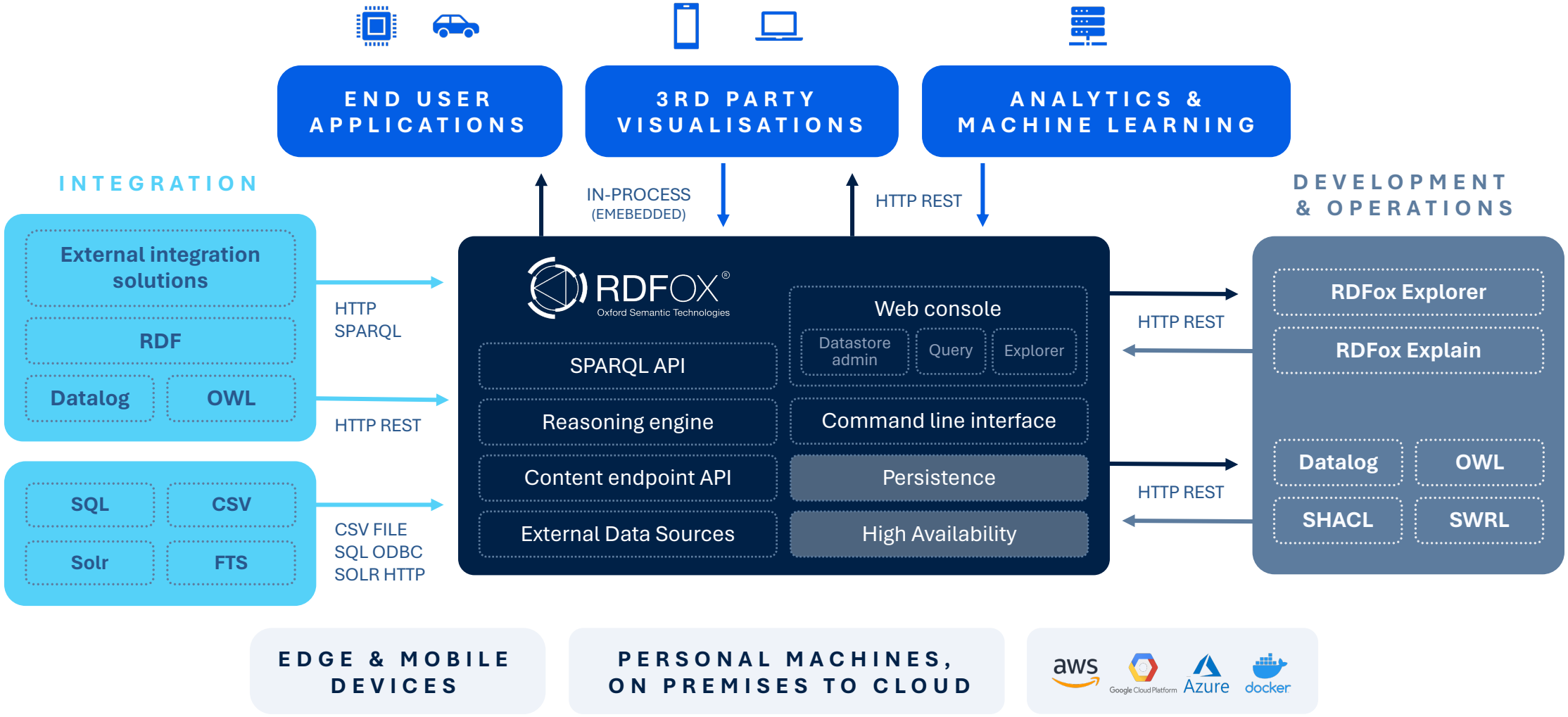
aws

Google Cloud Platform

Azure

docker

# RDFox Architecture



# Objectives

## The Foundations

- ✓ What is Semantic Reasoning?
- ✓ What is RDFox?
- ✓ Setting up RDFox

## OWL Reasoning

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

## Datalog Reasoning

- ✓ Basic rules
- ✓ Filters
- ✓ Aggregates
- ✓ Negation
- ✓ Binds
- ✓ Incremental Reasoning
- ✓ Materialization Vs. query rewriting
- ✓ RDFox in enterprise

# 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