RDFox Reasoning Workshop

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

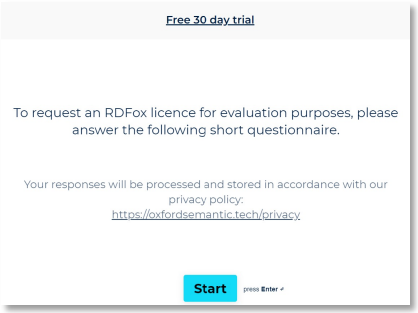
Requirements

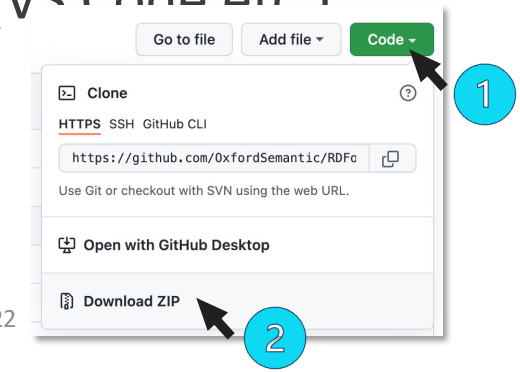
A. Get an RDFox License
https://www.oxfordsemantic.tech/tryrdfoxforfree

B. Download RDFox (& unzip)
https://www.oxfordsemantic.tech/downloads

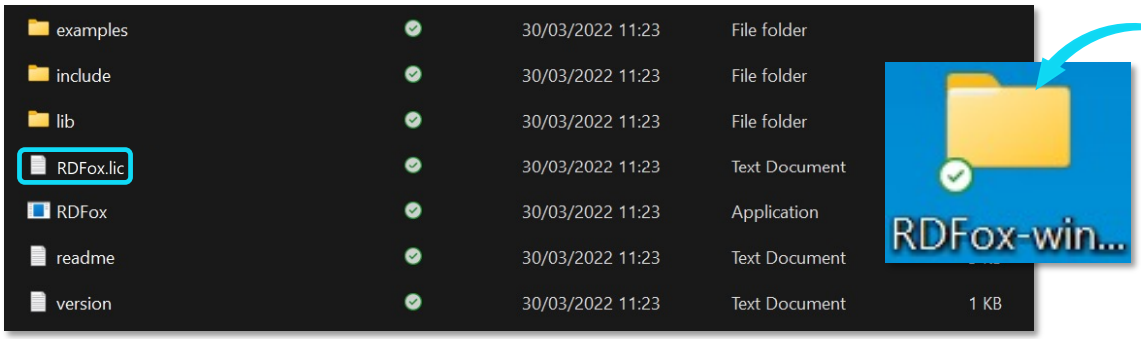
C. Download the class materials from Github:
git@github.com:OxfordSemantic/RDFoxWorkshop-Reasoning.git

D. OPTIONAL Get your IDE of choice ready (VS Code etc.)
https://code.visualstudio.com/





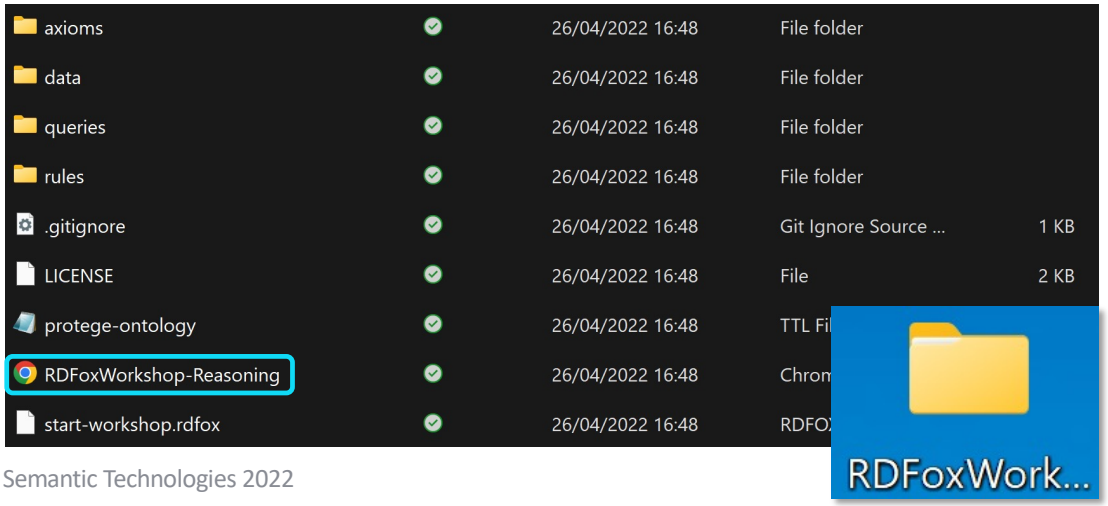
You should have…



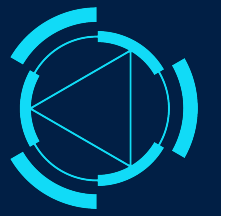
Make sure to put your license **INSIDE** the **RDFox folder**.

Open the class PDF so you can follow along!

You will need to follow the links provided.



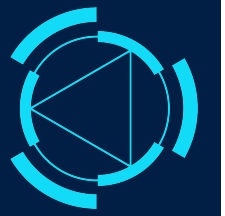
Objectives



By the end of the class everyone will:

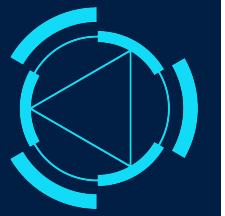
- know how to write OWL axioms in Turtle syntax
- know how to add OWL axioms to RDFox datastores
- know how to write and import Datalog rules
- understand how RDFox performs reasoning
- know common uses of rules

Workshop Structure



- The repository contains an RDFox script file “start-workshop.rdfox” with all the commands you need – copy and paste them one by one as we go
- Each axiom/rule has an associated query (mentioned on the slide). Run the query before and after reasoning to see the difference!
- Some examples are exercises where you will need to fill in the gaps
- Answers to the exercises have “completed” in their names

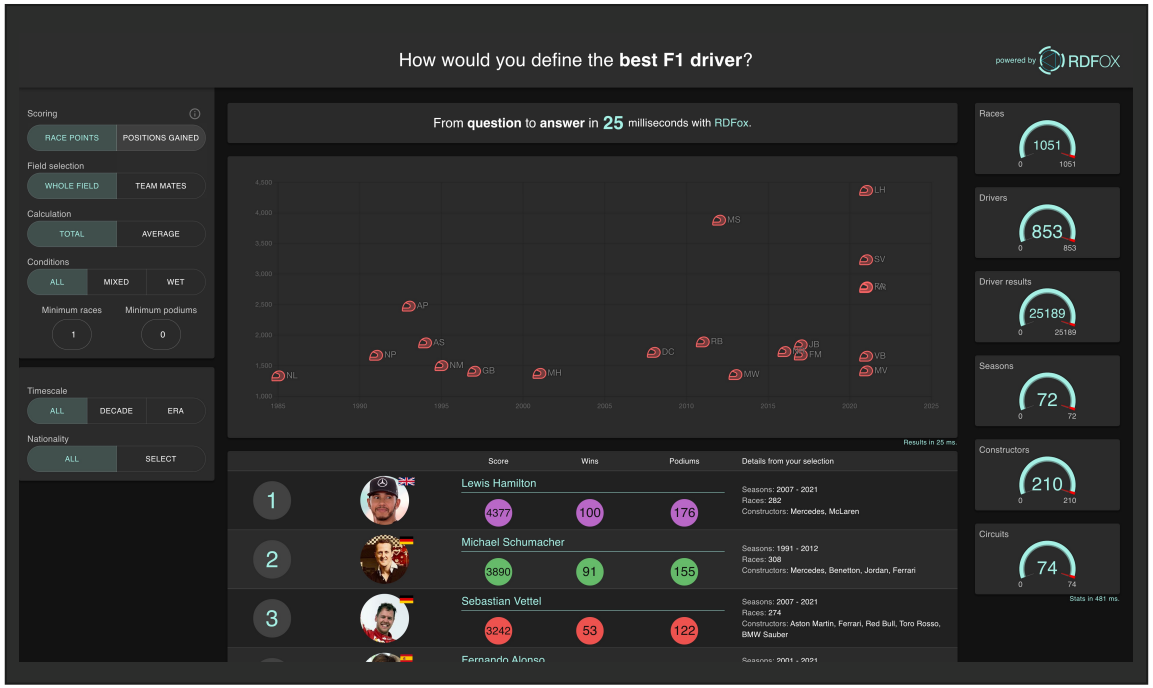
Workshop Structure



- I will talk about a concept, what it is, and why it's relevant
- I will go through a set of commands to show you this
- You are encouraged to follow along
- If there is an exercise, I will give you 1-2 minutes to try this, then I will go through it

Who is the Greatest Formula One Driver of All Time?

Controls and filters for the scoring system.



Statistics about the data used to form the results.

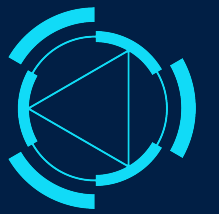
This matches the filters.

http://f1.rdfox.tech

Try it for yourself!

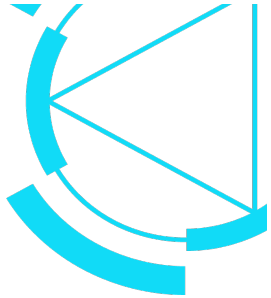
© Oxford Semantic Technologies 2022

Setting up RDFox



- We recommend using an IDE (e.g. VS Code)
- Open a terminal, navigate to the workshop folder
  `cd <path_to_workshop_folder>`
  and from there run
  `<path_to_RDFox>/RDFox sandbox`
  (Mac) or
  `<path_to_RDFox>\RDFox.exe sandbox`
  (Windows)
- RDFox server should be running now:

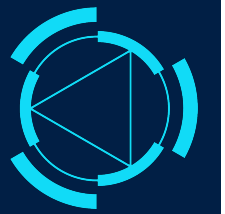
```
Source code for RDFox v1.0 Copyright 2013 Oxford University Innovation Limited and subsequent improvements Copyright 2017-2021 by Oxford Semantic Techn
ologies Limited.
This copy of RDFox is licensed for Developer use to Tom Vout (tom.vout@oxfordsemantic.tech) of OST until 07-Jun-2022 16:01:44
This 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'.
>
```



OWL reasoning

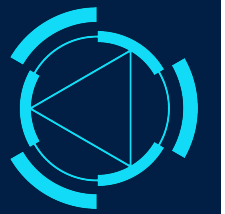
- ❑ Intro
- ❑ Subclasses
- ❑ Property types and domain/range
- ❑ Fixing mistakes
- ❑ Restrictions
- ❑ Open vs Closed World
- ❑ Property Chain
- ❑ A more complex example

Web Ontology Language (OWL)



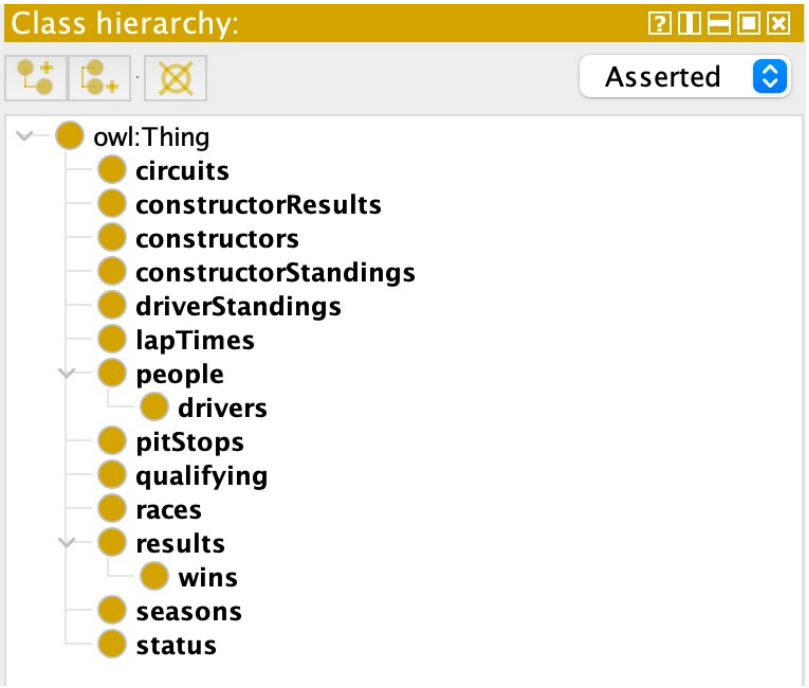
- standard created by W3C, RDFox supports the profile OWL 2 RL
- axioms = assertions about classes, properties or individuals
- ontology = collection of axioms
- ontologies are themselves graphs and can be written in many different syntaxes - we will use Turtle
- internally, RDFox translates axioms into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

Protégé

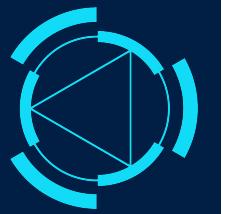


Protégé

- useful program for managing ontologies
- free and open source
- before writing an ontology, it is often better to have some data first



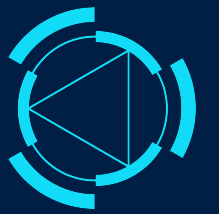
Sample data



- we import real F1 race data from file “upTo2020.ttl”
- we also import sample data suited to showing how OWL axioms work (file “sample.ttl”)

```
:drivers\/123456789 # no class
    :drivers_forename "John";
    :drivers_surname "Smith" .

:results\/123456789 a :results;
    :results_driver :drivers\/123456789, :drivers\/1; # two drivers
    :drivers_forename "John"; # wrong - this is a driver property
    # no constructor
    :results_race :races\/123456789;
    :results_positionOrder "1"^^xsd:integer .
```

Loading Data into RDFox



First we need to create a data store…

```
dstore create f1
```

Then to set it as active.

```
active f1
```

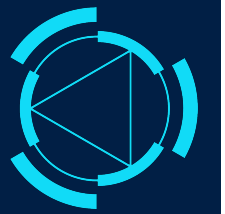
Now import the race data up to the year 2020.

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

And finally import our error-filled sample data.

```
import data/sample.ttl
```

Axiom set 1
Subclasses



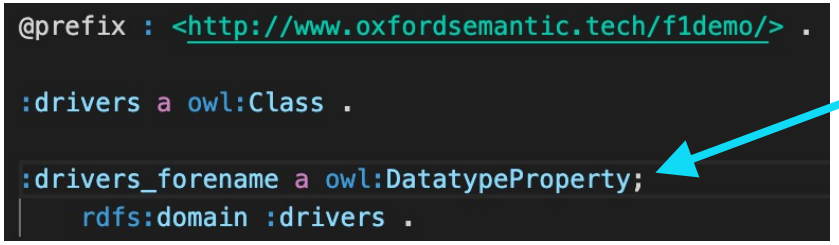
```
@prefix : <http://www.oxfordsemantic.tech/f1demo/> .

:drivers a owl:Class .
:people a owl:Class .

:drivers rdfs:subClassOf :people .
```

this is a completely new class that was not previously present in the data

- declares that `:drivers` and :people are classes
- says that `:drivers` is a subclass of `:people`
- effect: all instances of `:drivers` also become instances of `:people` (check by running query 1 – q1.rq)

Axiom set 2

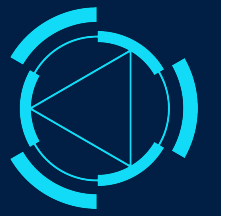
Property types and domain/range



OWL defines each property as either a "data" or "object" property

- says the domain of the property `:drivers_forename` is the class `:drivers`
- effect: `:drivers\/123456789` and `:results\/123456789` from sample both become instances of `:drivers` (check by running query 2 – q2.rq)

Fixing Mistakes



What should I do if I make a mistake?

```
importaxioms :axioms –
```

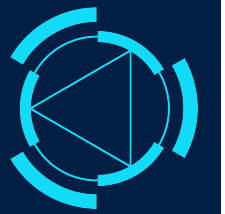
```
update ! clear graph :axioms
```

```
import > :axioms axioms/a1.ttl axioms/a2.ttl …
```

```
importaxioms :axioms
```

Axiom set 3 - exercise

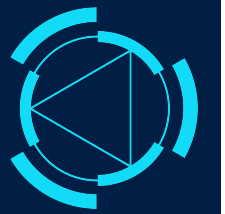
Property types and domain/range



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

___ a owl:Class .
___ a owl:Class .

:results_race a owl:ObjectProperty ;
    rdfs:domain ___ ;
    rdfs:range ___ .
```

- establishes the domain and range of the `:results_race` property
- effect: `:races\/123456789` from the sample becomes an instance of `:races` (check by running query 3 – q3.rq)

Axiom set 4
Restrictions

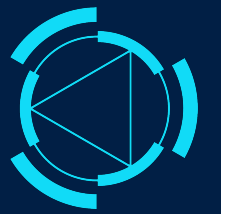


```
@prefix : <http://www.oxfordsemantic.tech/f1demo/> .

:results_driver a owl:ObjectProperty .

:results rdfs:subClassOf [
    a owl:Restriction ;
    owl:onProperty :results_driver ;
    owl:maxCardinality 1
] .
```

- says that every node of class `:results` has at most one value of `:results_driver`
- effect: property `owl:sameAs` now connects `:driver\/123456789` and `:driver\/1`, as well as every driver who ever raced to themselves (check by running query 4 – q4.rq)

Axiom set 5 - exercise

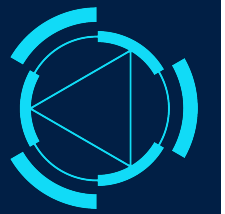
Restrictions



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

:results_constructor a ___ .

___ rdfs:subClassOf [
    a ___ ;
    ___ :results_constructor ;
    owl:minCardinality ___
] .
```

- says that every node of class `:results` has at least one value of `:results_constructor`
- effect: none (Why? Because of the open world assumption of OWL)

Axiom set 6
Property chains



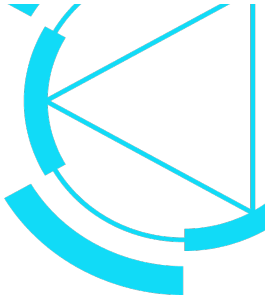
```
@prefix : <http://www.oxfordsemantic.tech/f1demo/> .

:results_driver a owl:ObjectProperty .

:hasRacedIn owl:propertyChainAxiom (
    [owl:inverseOf :results_driver]
    :results_race
) .
```

- defines a new property `:hasRacedIn` as a combination of `:results_driver` (inverted) and `:results_race`
- effect: property `:hasRacedIn` now connects drivers and their races (check by running query 6 – q6.rq)

Axiom set 7
A more complex example

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

:qualifying a owl:Class .

:qualifying_position a owl:DatatypeProperty .

:polePositionQualifying a owl:Class ;
    owl:equivalentClass [
        a owl:Class ;
        owl:intersectionOf (
            :qualifying
            [
                a owl:Restriction ;
                owl:onProperty :qualifying_position ;
                owl:hasValue "1"^^xsd:integer
            ]
        )
    ] .
```

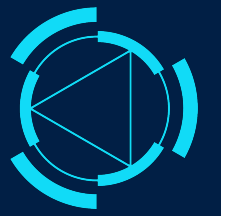
- says that the `:polePositionQualifying` class is the intersection of the :qualifying class and nodes that have 1 as the value of their `:qualifying_position` property
- effect: some `:qualifying` instances also become instances of `:polePositionQualifying` (check by running query 7 – q7.rq)



Datalog Reasoning with RDFox

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

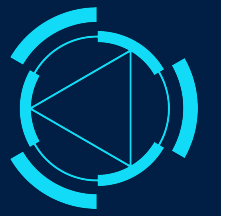
Datalog Rules



[*fact A*] :- [*fact B*]

“A is true whenever B is true”

Rule 1
Basic rules



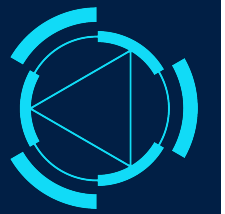
```
[?driver, :hasRacedIn, ?race] :-
    [?result, :results_race, ?race],
    [?result, :results_driver, ?driver] .
```

This rule will add a direct link between drivers and the races they raced in. Notice this has the same effect as axiom 6, but expressed more clearly

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 8 – q8.rq

Rule 2
Filters

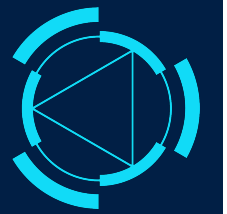


```
[?driver, :hasPodiumInRace, ?race] :-
    [?result, :results_race, ?race],
    [?result, :results_driver, ?driver],
    [?result, :results_positionOrder, ?positionOrder],
    FILTER(?positionOrder < 4) .
```

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

Check for inferred triples with query 9 – q9.rq

Rule 3
Aggregates



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

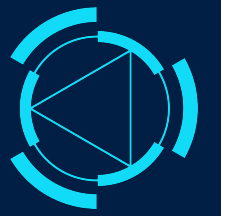
This rule adds a count of races a driver has entered.

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.

Check for inferred triples with query 10– q10.rq

Rule 4 - exercise

Aggregates



```
[?___, :hasRaceWinCount, ?___] :-
    AGGREGATE(
        [?___, :results_driver, ?___],
        [?___, :results_positionOrder, 1]
        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.

Check for inferred triples with query 11 – q11.rq

Rule 5
Negation

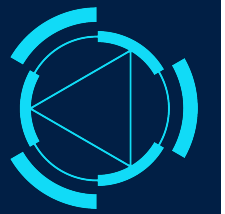
```
[?driver, a, :DriverWithoutPodiums] :-
    [?driver, a, :drivers],
    NOT EXISTS ?race IN (
        [?driver, :hasPodiumInRace, ?race]) .
```

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).

Check for inferred triples with query 12 – q12.rq

Negation as Failure



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

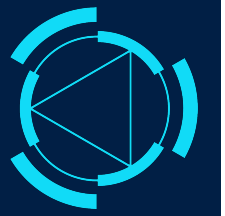
Using Negation as Failure leads to 'non-monotonic reasoning'- the engine may need to retract some triples instead of just adding them. This adds another dimension to an already complicated problem of reasoning planning, but RDFox handles all of that automatically.

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 "q12b.rq", then import the data from file "2021-22.ttl" and try again

Rule 6 - exercise

Binds

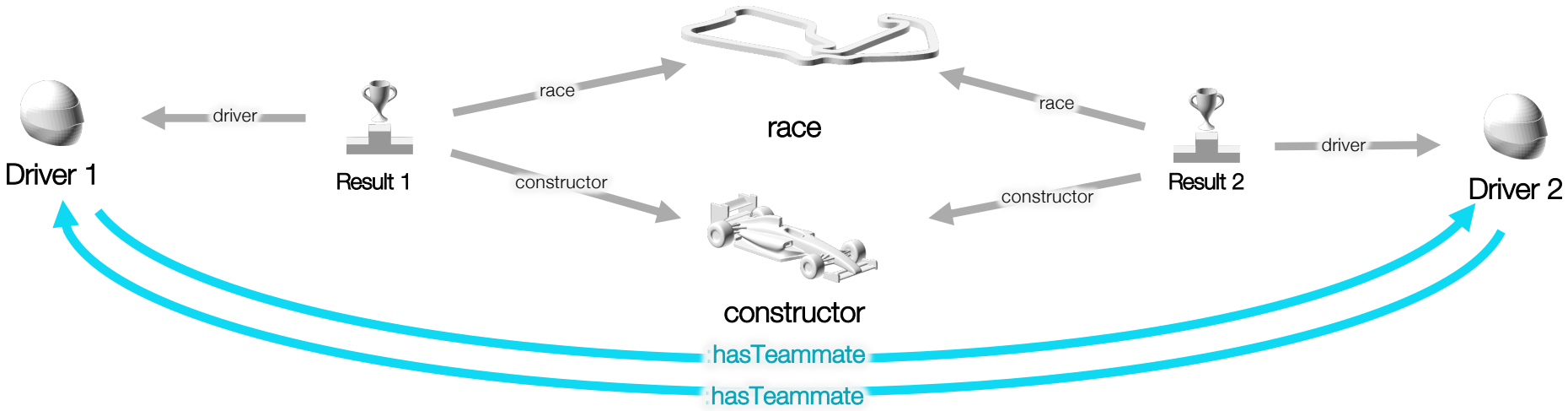


```
[?driver, :hasWinPercentage, ?percentage] :-
    [?___, :hasRaceCount, ?___],
    [?___, :hasRaceWinCount, ?___],
    BIND(?raceWinCount/?raceCount AS ?percentage) .
```

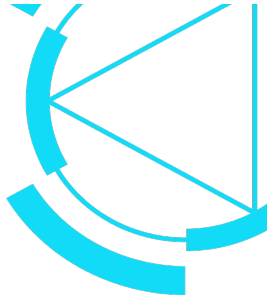
With `BIND` we can use various mathematical functions, string manipulation, regular expression matching, conditional binds, hashing and IRI creation.

Check for inferred triples with query 13 – q13.rq

Bonus Exercise

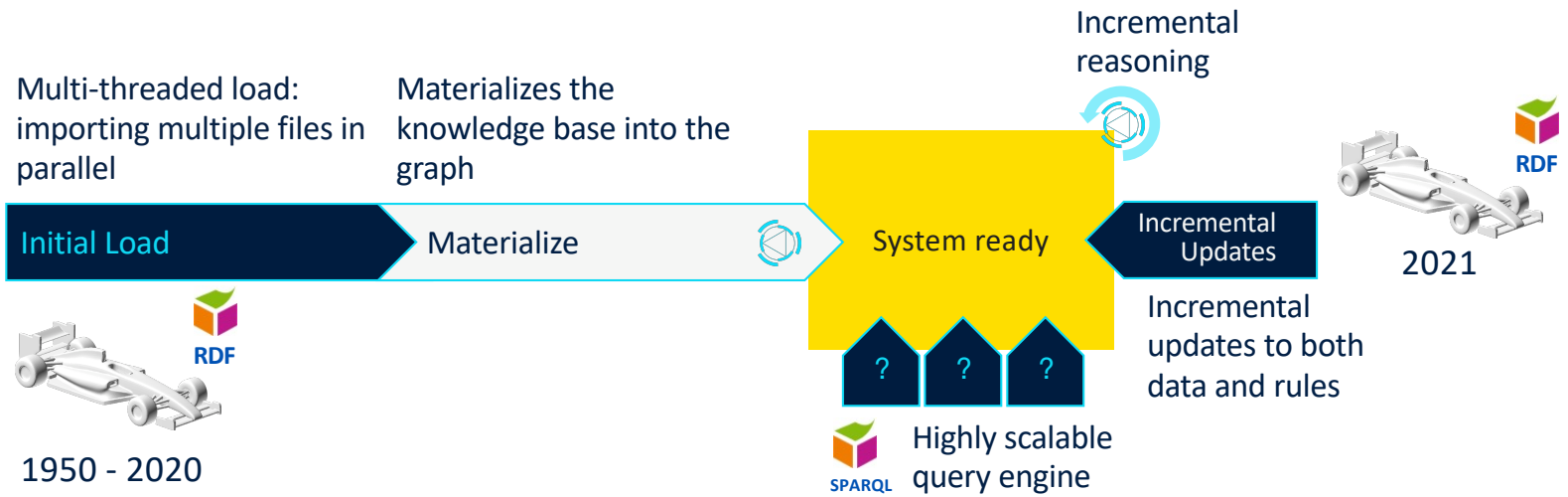


- Write and import a rule to find out if two drivers have been teammates (use the predicate `:hasTeammate`).

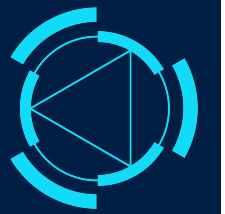


RDFox: How does it work?

Incremental Reasoning

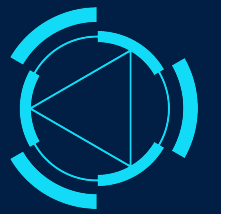


Incremental Reasoning



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

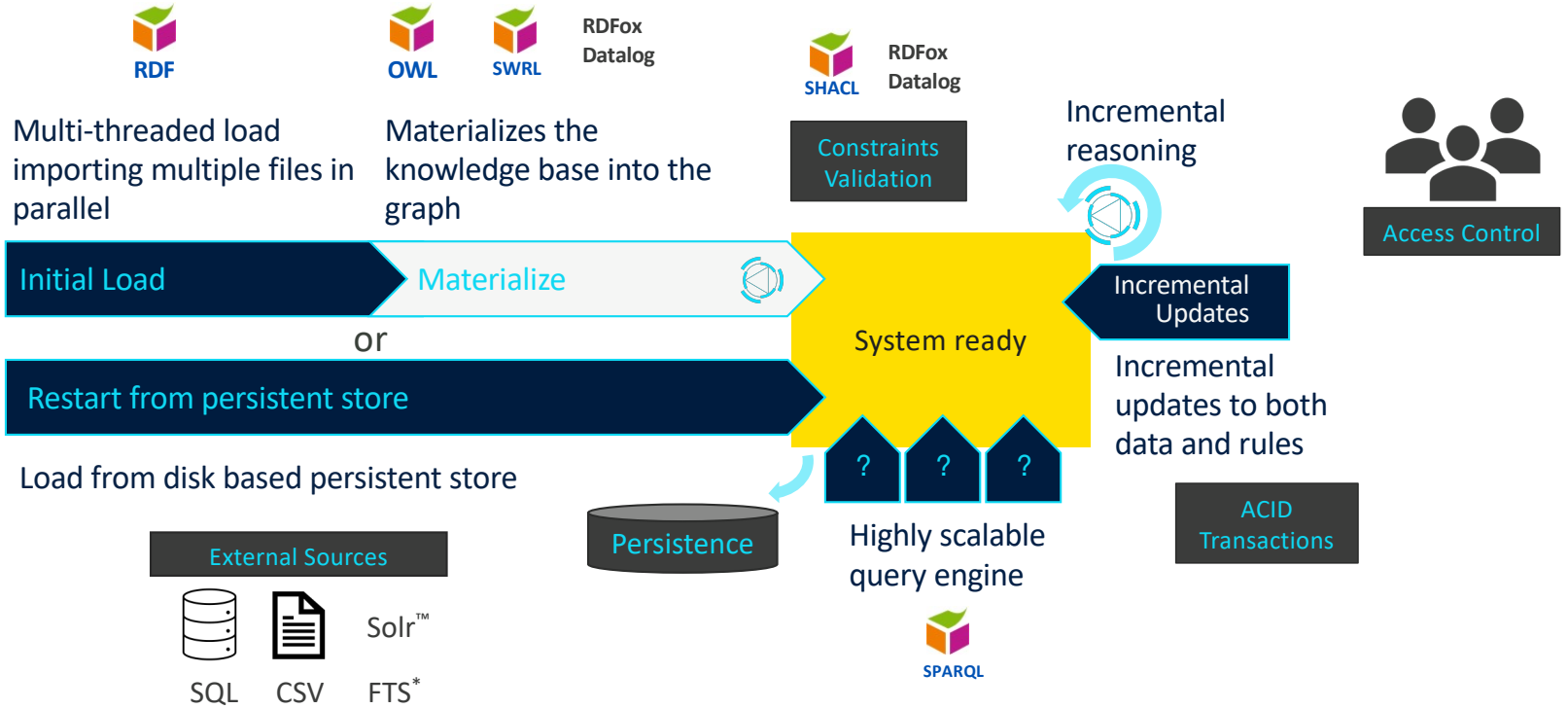
RDFox vs. Other Solutions



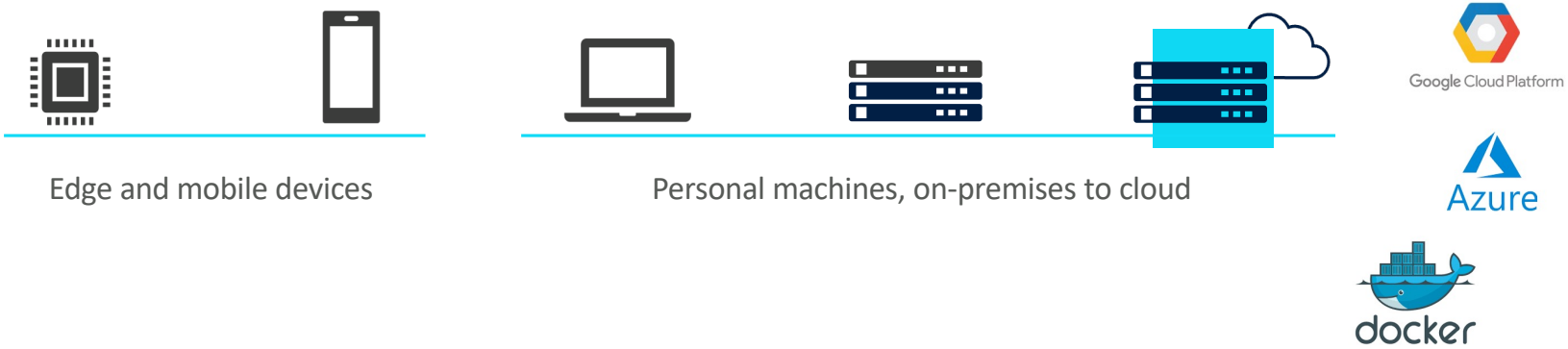
	Feature	Example	RDFox	Materialisation Competitors	Query Rewriting Competitors
	Ontological reasoning	"List all financial instruments / derivative instruments / futures." [class inferencing]	✓	✓*	✓**
	Graph analytics	"What is the nearest electrical switch for a circuit?" [negation + recursion]	✓		
	Data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

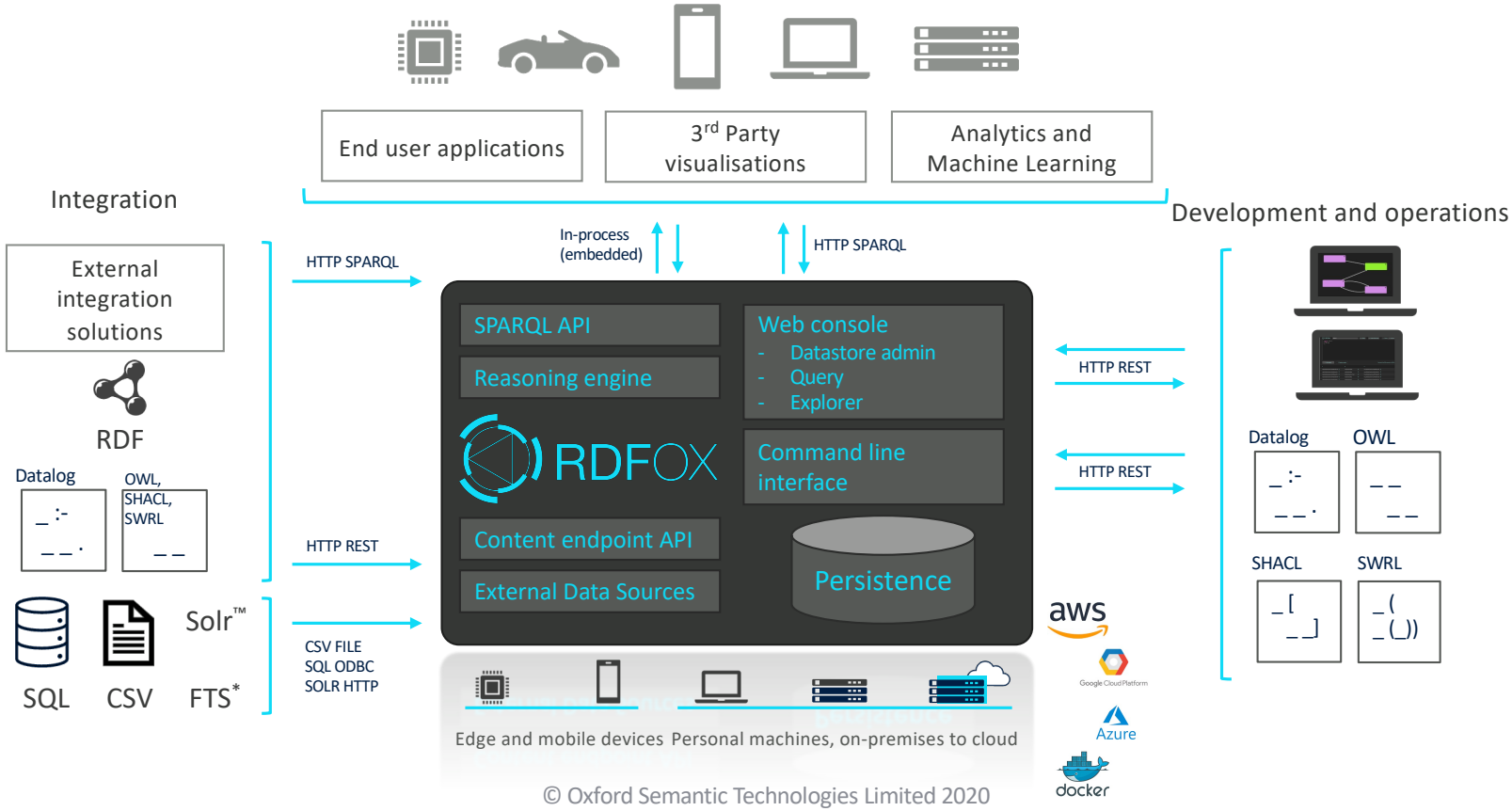


Deployment Options

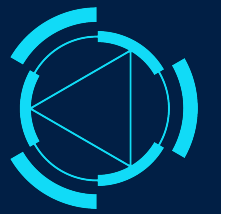


Architecture

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/fygpYBrS