RULES FEST

ONTORULE

Where Ontologies Meet Business Rules

The Demonstrator Engines

Hugues Citeau

Senior Developer

IBM - French CAS

Outline

ONTORULE

OWL2 Ontologies

Production Rules over OWL2 Ontologies

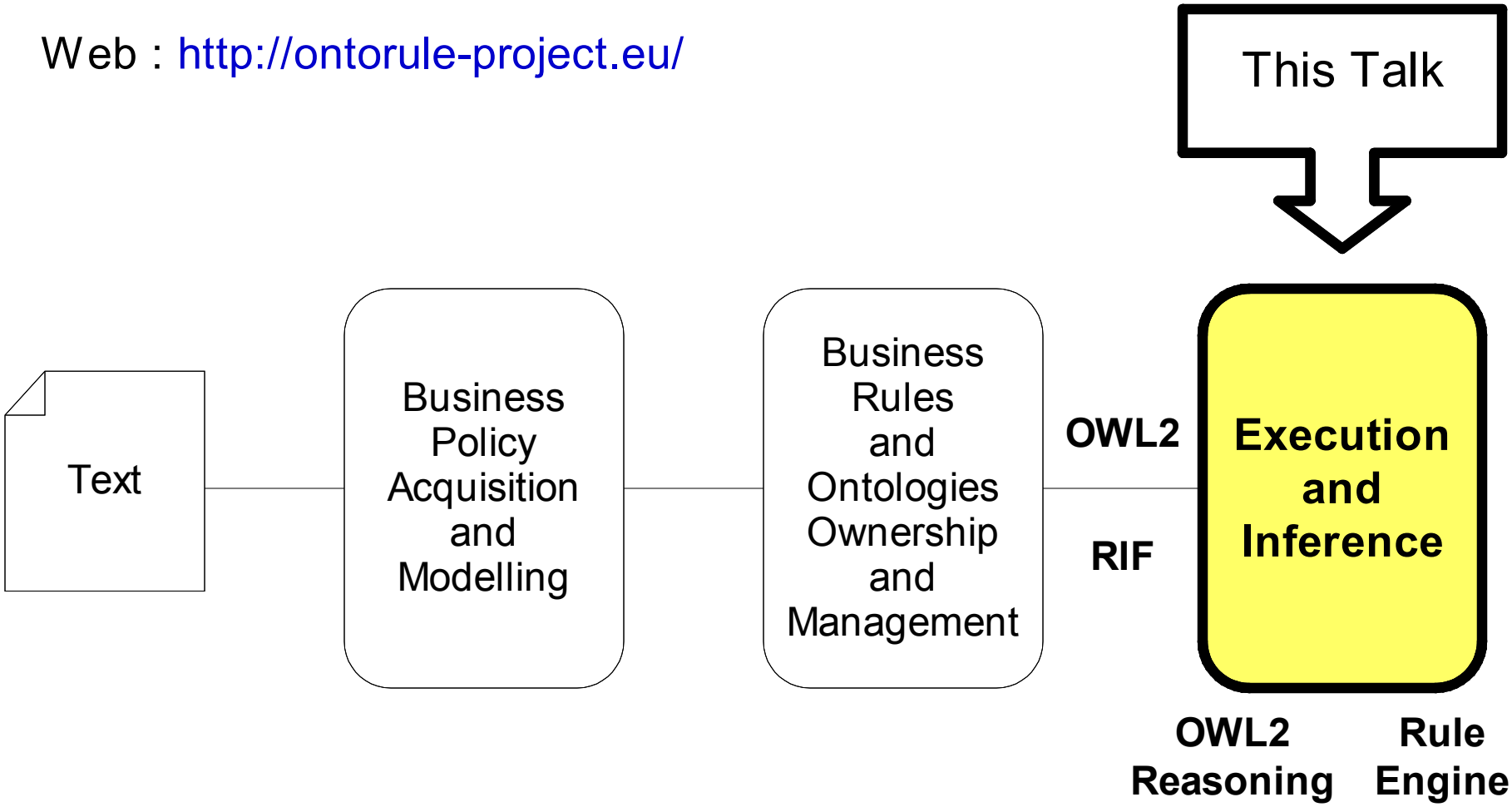
Weakly Coupled Approaches

OWL2RL

Tightly Coupled Approach

ONTORULE EEC Project

Web : http://ontorule-project.eu/



OWL2 Ontologies

Overview of OWL2 Ontologies

Rich Logical Knowledge Descriptions

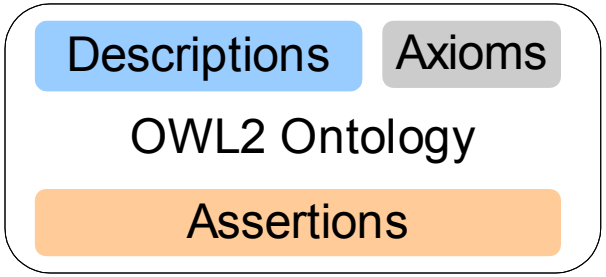
- Class Expressions
- Property Expressions
- Axioms

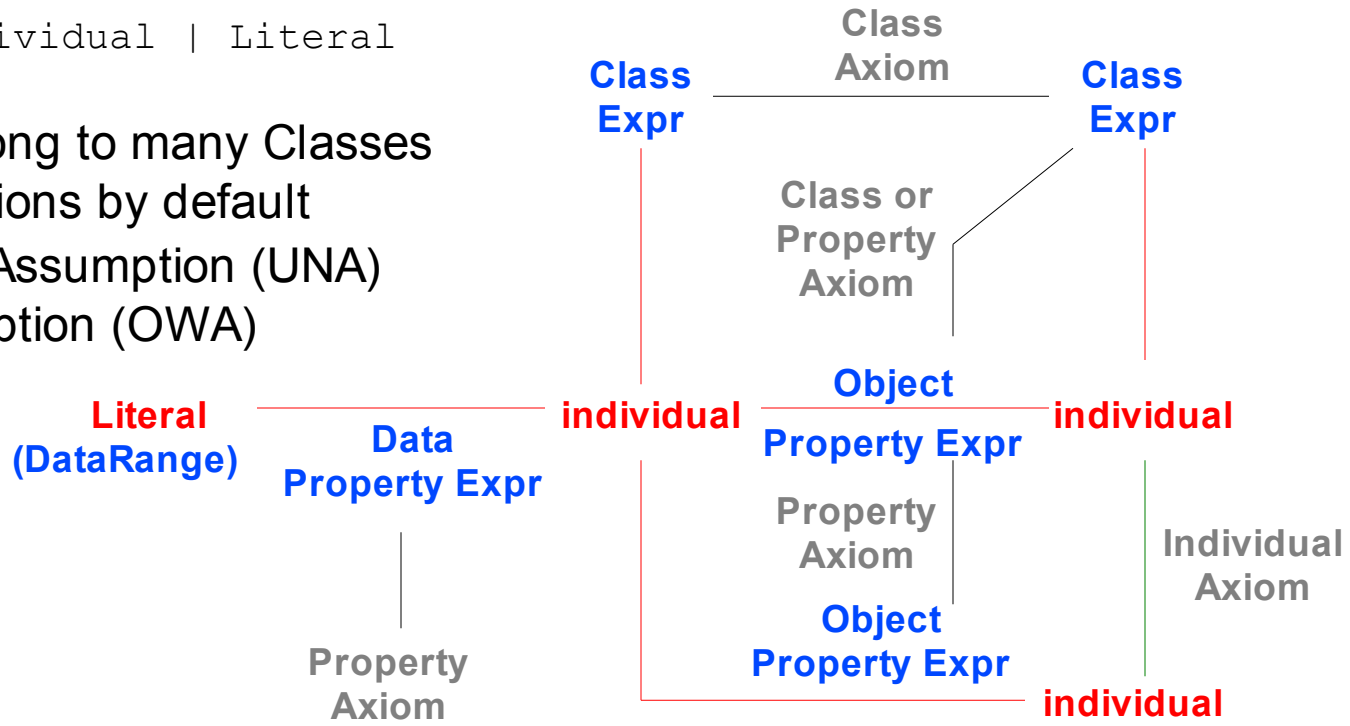
Simple Runtime Model

- Assertions about Constants

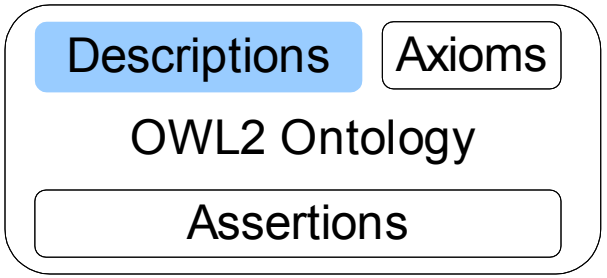
```
Constant ::= Individual | Literal
```

- Individuals may belong to many Classes
- Properties are Relations by default
- **Non** Unique Name Assumption (UNA)
- Open World Assumption (OWA)





OWL2 Descriptions



ClassExpr ::=

```
  Class Name
| oneOf(Individual ...)
| unionOf(ClassExpr,ClassExpr)
| intersectionOf(ClassExpr,ClassExpr)
| complementOf(ClassExpr)
| allValuesFrom(PropExpr,  ClassExpr
                          | DataRange)
| someValuesFrom(PropExpr,  ClassExpr
                           | DataRange)
| cardRestriction((<= | >= | =) nonNegInt,
                  PropExpr)
| hasValue(PropExpr,Constant)
```

PropExpr ::=

```
  Property Name
| propertyChain(PropExpr ...)
| inverseOf(PropExpr)
```

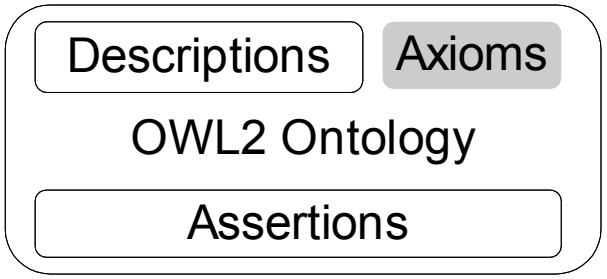
DataRange ::=

```
  Datatype Name
| oneOf(Literal ...)
| unionOf(DataRange,DataRange)
| intersectionOf(DataRange,DataRange)
| complementOf(DataRange)
| restriction(DataRange,
              ( ConstrainingFacet
                Literal ) ...)
```

ConstrainingFacet ::=

```
  Constraint Name
  // XSD: minValue,maxValue...
```

OWL2 Axioms



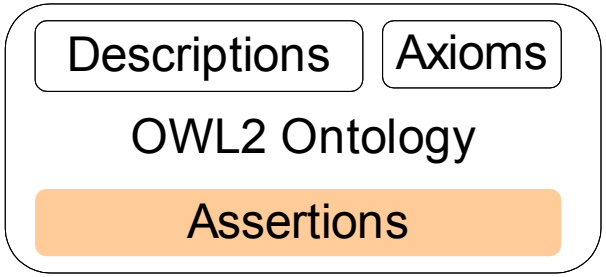
ClassAxiom ::=

```
  equivalentClasses(ClassExpr,ClassExpr)
| subClassOf(ClassExpr,ClassExpr)
| disjointClasses(ClassExpr ...)
| disjointUnion(ClassExpr,ClassExpr ...)
| hasKey(ClassExpr,PropExpr ...)
```

PropertyAxiom ::=

```
  equivalentProperties(PropExpr,PropExpr)
| subPropertyOf(PropExpr,PropExpr)
| domain(PropExpr,ClassExpr)
| range(PropExpr,  ClassExpr
                 | DataRange)
| reflexive(PropExpr)
| irreflexive(PropExpr)
| symmetric(PropExpr)
| asymmetric(PropExpr)
| transitive(PropExpr)
| functional(PropExpr)
| inverseFunctional(PropExpr)
```

OWL2 Assertions



Assertion ::=

```
  classAssertion(ClassExpr,Individual)
| positivePropertyAssertion(PropExpr,Individual,Constant)
| negativePropertyAssertion(PropExpr,Individual,Constant)

| sameIndividual(Individual ...)
| differentIndividuals(Individual,Individual ...)
```

The Simpsons in OWL2 (*in Tight syntax*)

```
// Descriptions and Axioms:
class Person;
class Girl << Person;
class Boy << Person;
class Girl <> Boy;
class Parent == some(child,>=1);
class OneChildParent == some(child,=1);
property child {
  domain Person;
  range Person;
};
property wife {
  domain Boy;
  range Girl;
};
property cash {
  functional;
  domain Person;
  range integer ! (>= 0);
};
```

```
// Assertions:
Girl(Maggie);
Girl(Lisa);
Boy(Bart);
Boy(Homer);
Boy(MrBurns);
Boy(Apu);
Parent(Apu);
child(Marge,Maggie);
child(Marge,Lisa);
child(Marge,Bart);
wife(Homer,Marge);
cash(Homer,12);
cash(MrBurns,3500000000);
```

Understanding Axioms

Derived Knowledge is built by recursive application of Axioms on direct and derived Knowledge

Example

The direct assertions do not initially say much about Marge

```
property wife {
  range Girl;
};
```

wife(Homer,Marge)

=>

Girl(Marge)

Understanding the Open World Assumption

Existence of Individuals is inferred according to the constraints accumulated along the reasoning path

Example

The direct assertions do not initially say much about Apu's children

```
class Parent == some(child,>=1);
```

```
Parent(Apu)
<=>
(some(child,>=1))(Apu)
=>
child(Apu,new1)  // there exists at least
                 // one child even if
                 // not directly mentioned
```

Understanding the Non Unique Name Assumption

Individuals with different identifiers are not different by default

Example

Bart fills an administrative form and asserts that Marge has only one child

```
class OneChildParent == some(child,=1);

child(Marge,Maggie);
child(Marge,Lisa);
child(Marge,Bart);
```

```
OneChildParent(Marge)
<=>
(some(child,=1))(Marge)
=>
sameIndividual(Maggie,Lisa,Bart)
```

Understanding Inconsistencies

It is easy to introduce inconsistencies
- OWL2 Reasoning is complex
- OWL2 Ontologies contain many redundancies

But it is impossible to reason on an inconsistent Ontology

Inconsistencies must be Statically Detected or Automatically Fixed

Example
- *After a few drinks at Moe's, Homer and MrBurns get married*

```
property wife {
  range Girl;
};

Boy(MrBurns)

Boy <> Girl
```

```
wife(Homer,MrBurns)
=>
Girl(MrBurns)
=>
(Girl && Boy)(MrBurns)
=>
Nothing(MrBurns)
```

Understanding the limits of OWL2

Only predicates, arity <= 2 are supported

```
BoyProfile(b,10,"Bart",Sk8)
```

```
(Boy && has(age,10) && has(name,"Bart") && has(vehicle,Sk8))(b)
```

Restriction values can only be constants, not variables

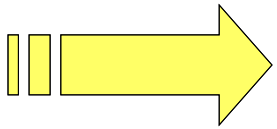
```
some(child,<= ?x) ! Boy
```

Error

```
some(cash) ! integer!(>= ?y)
```

Error

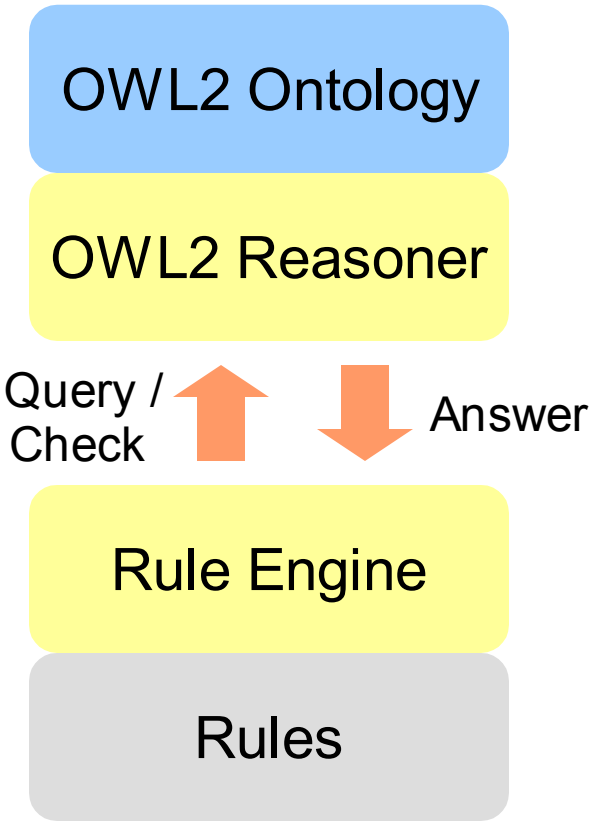
That's why Rules with more powerful conditions have to be added to Ontologies



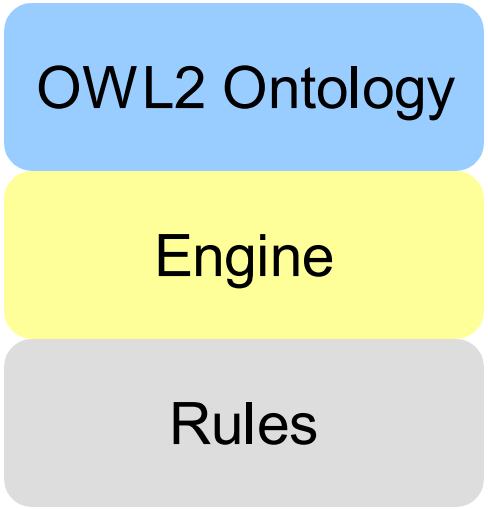
Production Rules over OWL2 Ontologies

Coupling Strategies

Weak Coupling



Tight Coupling



Common Coupling Assumptions

The OWL2 Ontology is initially consistent

The Production Rules do not modify the Descriptions of the OWL2 Ontology (the T-Box)

The Production Rules
- match the direct and the derived Assertions of the OWL2 Ontology
- update the direct Assertions of the OWL2 Ontology (the A-Box is the Working Memory

Updating Assertions is a Challenge

- OWL2 is designed for deductive Queries on read-only Ontologies
- When compared to Queries and Logical Rules, Production Rules are providing a mean to update the OWL2 Ontology in the Actions as part of the Engine Cycle
- But the updated OWL2 Ontology must remain consistent
- When the full descriptive power of OWL2 is used, it is very difficult for a Human to find out which Assertions will be inconsistent after an Update

OWL2 reasoning is complex, it is mandatory to provide Tools
- to detect and explain inconsistent Updates at compile time
- to automatically fix inconsistent Updates at runtime

Introduce an *Inconsistency Fixing Strategy*

Wanted and *Unwanted* Inconsistencies

Wanted = Runtime Fixing

```
property cash {
    functional;
    domain Person;
    range integer;
};

production rule Debit100 {
    when {
        cash(?x,?c);
    }
    then {
        insert cash(?x,?c - 100);
    }
}
```

Unwanted = Static Verification
(do the best you can here)
or Runtime Error

```
property cash {
    functional;
    domain Person;
    range integer!(>= 0);
};

production rule InitCash {
    when {
        Person(?x);
    }
    then {
        insert cash(?x,-100);
    }
}
```

Static Type Checking in Actions vs Dynamic Classification

```
class Good;
class Bad <> Good;
```

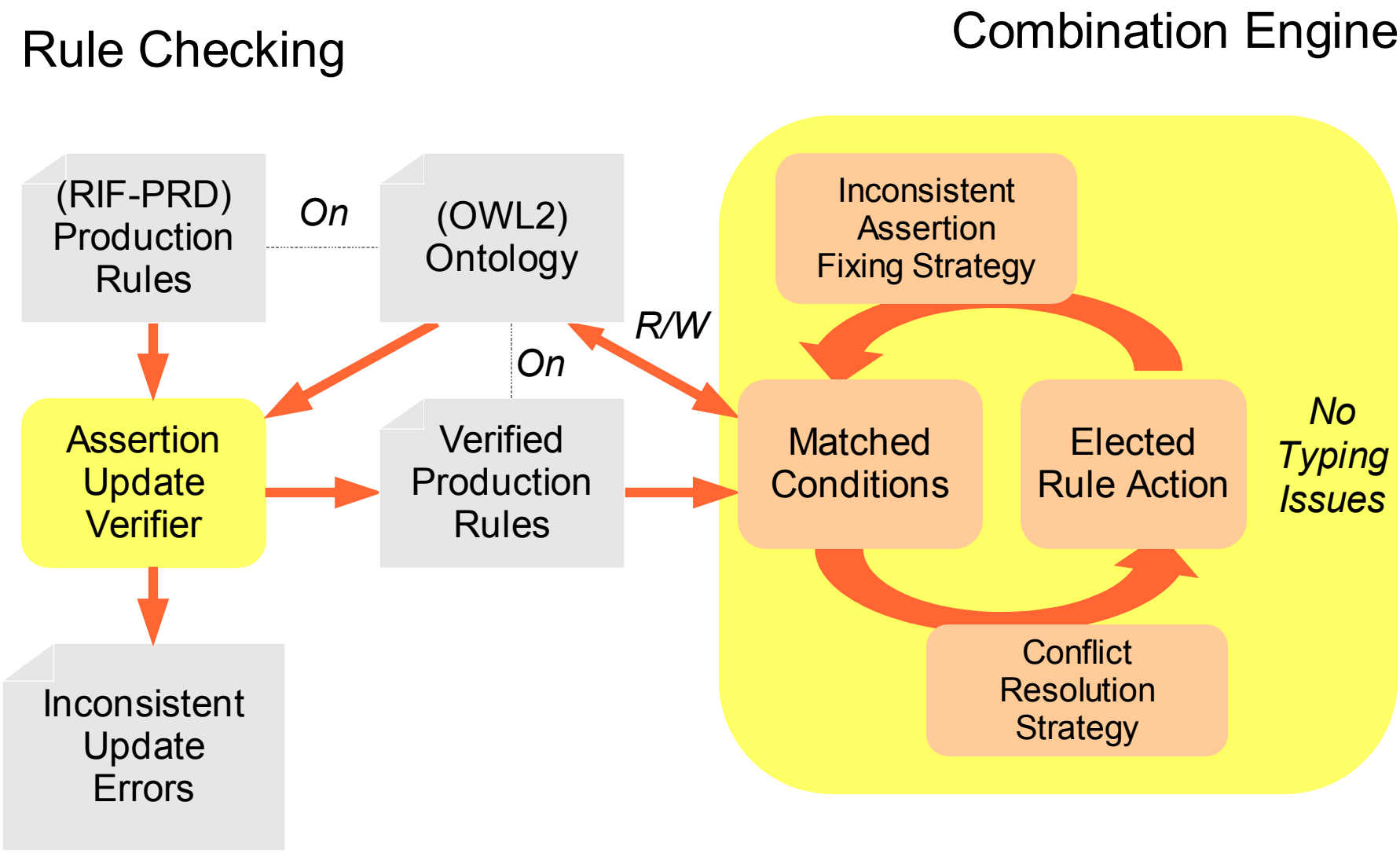
```
function f(Good x);
```

```
production rule R {
  when {
     x : Good();
  }
  then {
    Good y = x;
    insert x as Bad;  // Assuming Good(x) has
                      // automatically been removed
    // Past this point y is not Good anymore !
    f(y);
  }
}
```

Possible Solutions

- Drop static type checking of Action Variables
  - Only `Thing` (*the Top* Class) is allowed to statically type Individuals
  - (*Note that no issue here with the* `DataRange` *for Data*)
- Put all Assertion modification Actions as the last Actions

Drawing an Objective



ONTORULE Demonstrator #1

Weak Coupling
between
Legacy Object-Oriented Production Rules
and
OWL2 Ontologies

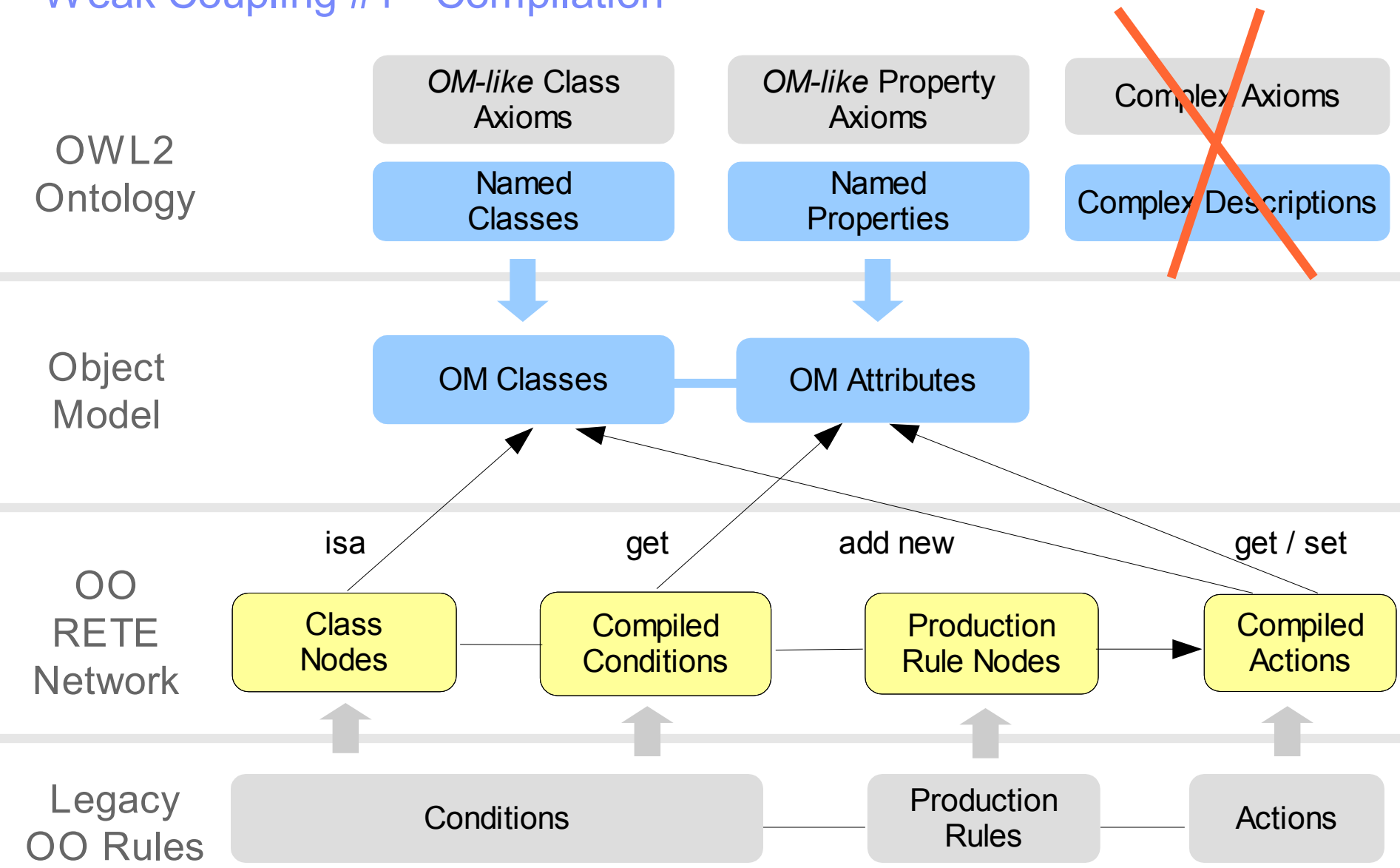
(*Missing the Train*)

Legacy Object-Oriented Production Rules

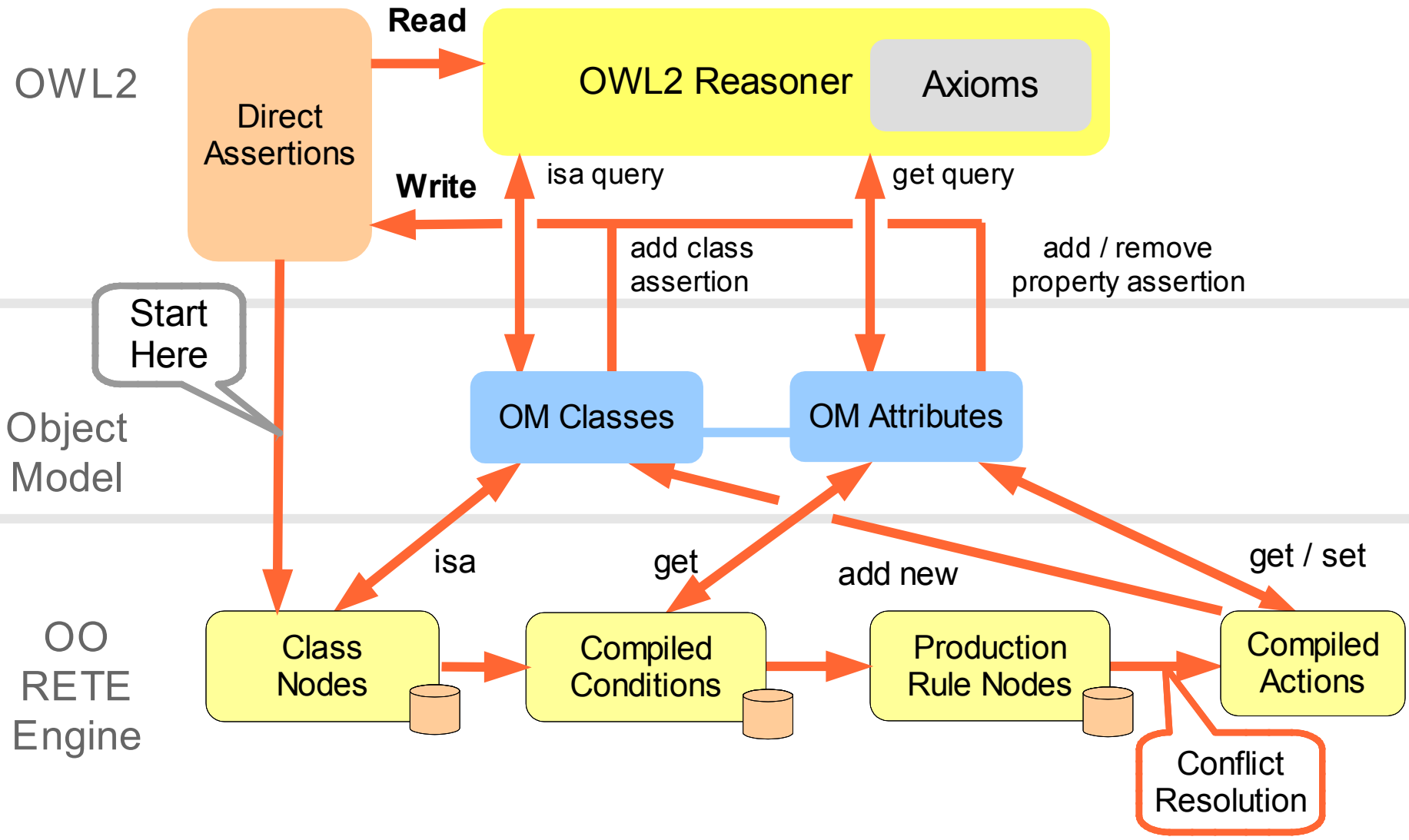
```
class Person {
  int age;
  Collection<Suggestion> suggestions;
}

class Girl extends Person {
  Person child;
}

class Boy extends Person {
}

class Suggestion {
  Suggestion(String text);
}
```

```
production rule Sk8Suggestion {
  when {
    g : Girl ( child c );
    b : Boy ( age <= 12 ) from c;
  }
  then {
    Suggestion s = new Suggestion
      ("Ask for a new skateboard");

    insert s;
    b.suggestions.add(s);
    update g;
  }
}
```

Weak Coupling #1 - Compilation



Weak Coupling #1 - Execution



Weak Coupling #1 – How far from the Objective ?

Reasoning Capabilities

Depend on the plugged OWL2 Reasoner

Rule Language

+ Compliant with legacy authoring and maintenance Tools
- No complex OWL2 Class Expressions
- No complex OWL2 Property Expressions
- No update of Individual Classes
- No update of Individual identity Assertions

Static Update Verification

Not implemented

Consistency Checking can be delegated to OWL2 Reasoner

Static Type Checking in Actions

- Can be broken, updating Attributes can change the Classes of Individuals

Inconsistency Fixing Strategy

Experiments using OWL2 Reasoner to identify inconsistent Assertions going on

ONTORULE Demonstrator #2

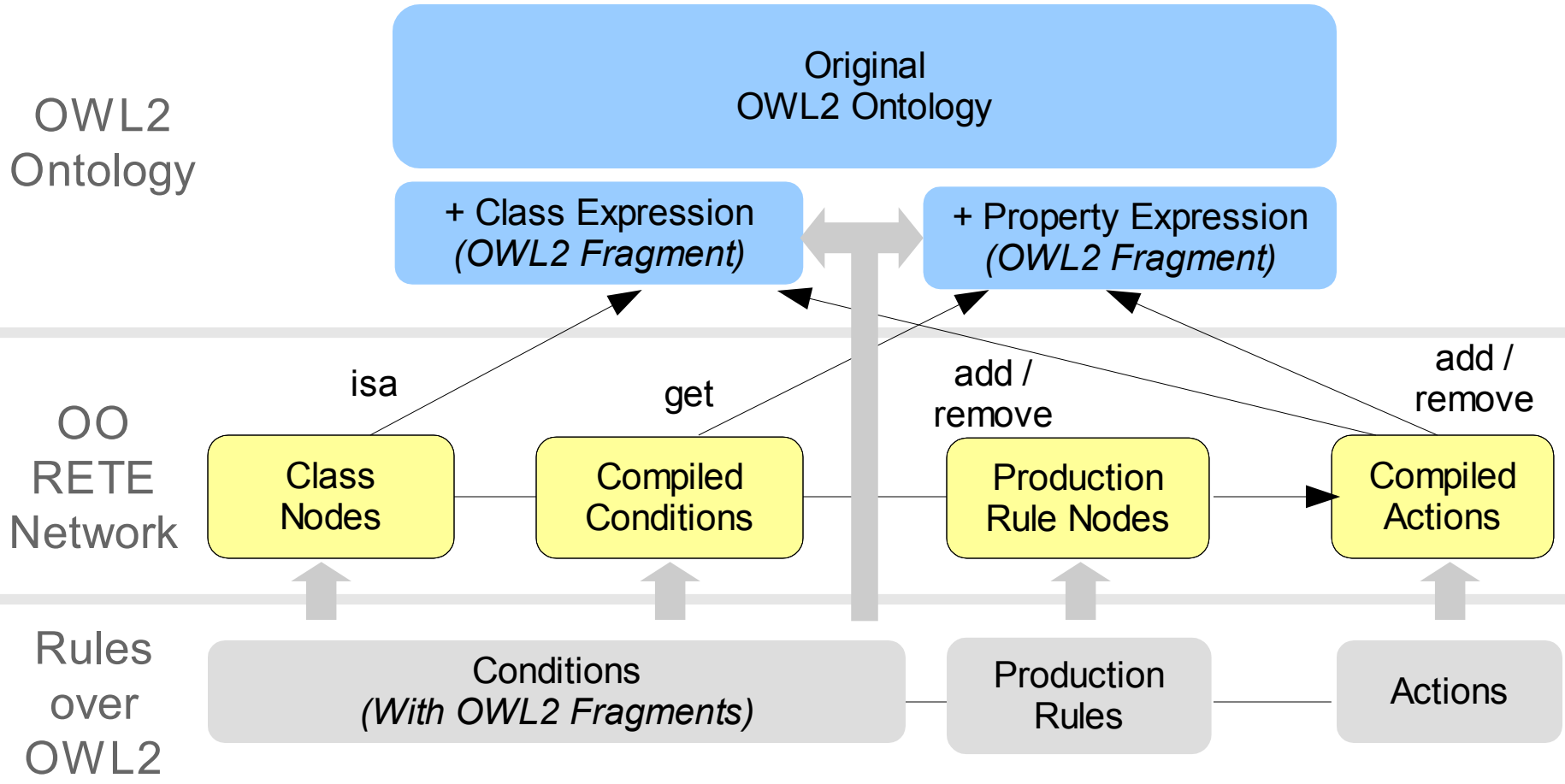
Weak Coupling
between
Production Rules
and
OWL2 Ontologies

(*Taking the wrong Train)*

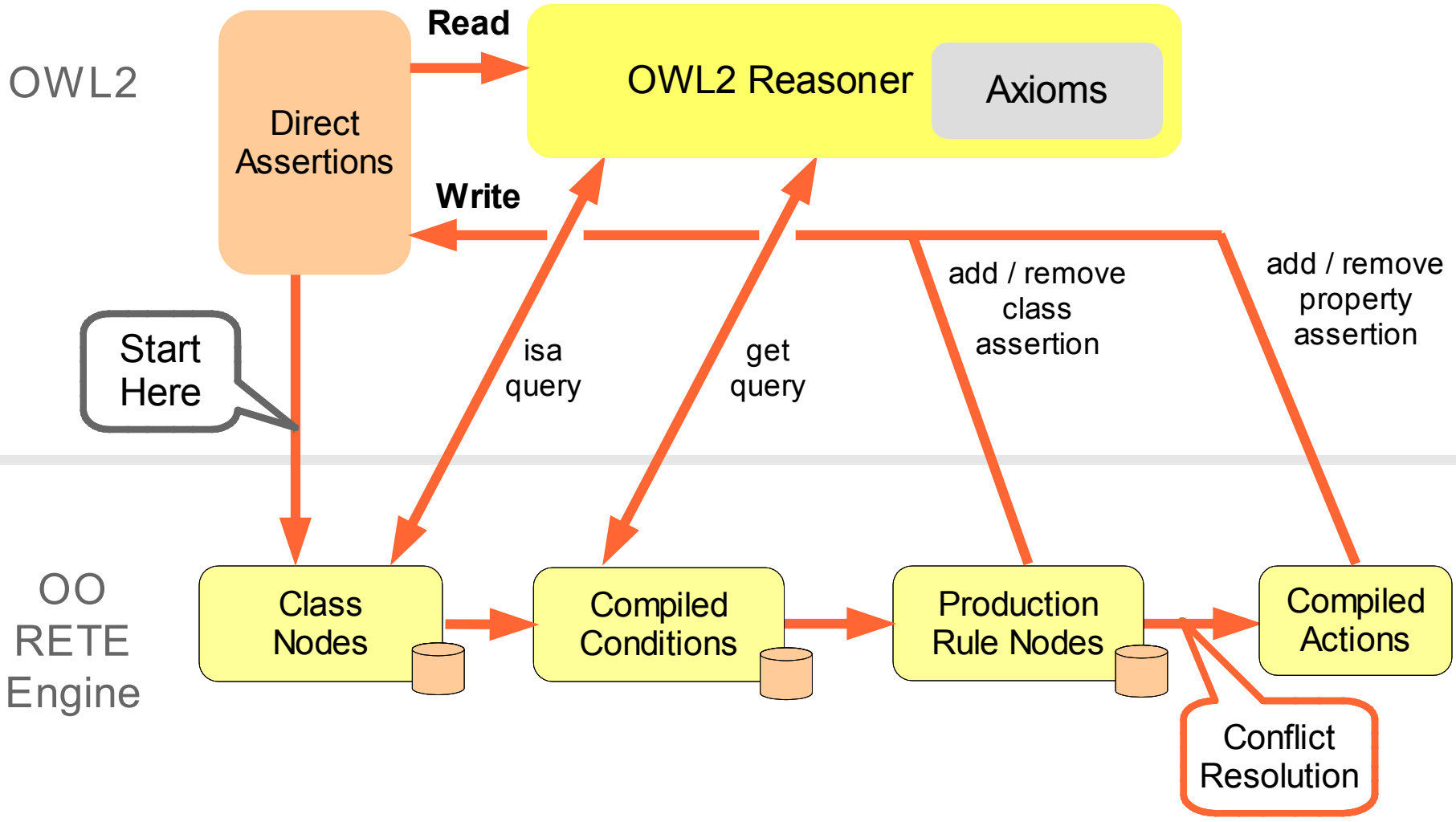
Production Rules with OWL2 Fragments

```
production rule Sk8Suggestion {
  when {
    this g isa Girl {
      child {  // can match all the objects (b) related to the current subject (g)
        this b isa (Boy && some(age,<= 12)) {  // can match class expressions
          - fatherOf f;  // can match property expressions (- means inverse here)
        } }; }
  }
  then {
    Suggestion s = new Suggestion ("Ask for a new skateboard");

    insert b as TrackedCustomer;  // can update classes of individual
    insert b suggestions s;  // can incrementally update relation
  }
}
```

Weak Coupling #2 - Compilation



Weak Coupling #2 - Execution



Weak Coupling #2 – How far from the Objective ?

Reasoning Capabilities

Depend on the plugged OWL2 Reasoner

Rule Language

- Not Compliant anymore with legacy Tools
+ Complex OWL2 Class Expressions
+ Complex OWL2 Property Expressions
+ Update of Individual Classes
+ Update of Individual identity Assertions

Static Update Verification

Still not implemented

Static Type Checking in Actions

- Can still be broken, updating Properties can change the Classes of Individuals

Runtime Inconsistency Fixing Strategy

The most recent Assertion is the right one

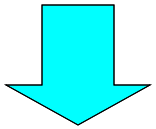
But only the direct Assertions are fixed

ONTORULE Demonstrator #3

Tight Coupling between Production Rules and OWL2 Ontologies

(*Taking one Train)*

Tight Coupling - Design Choice



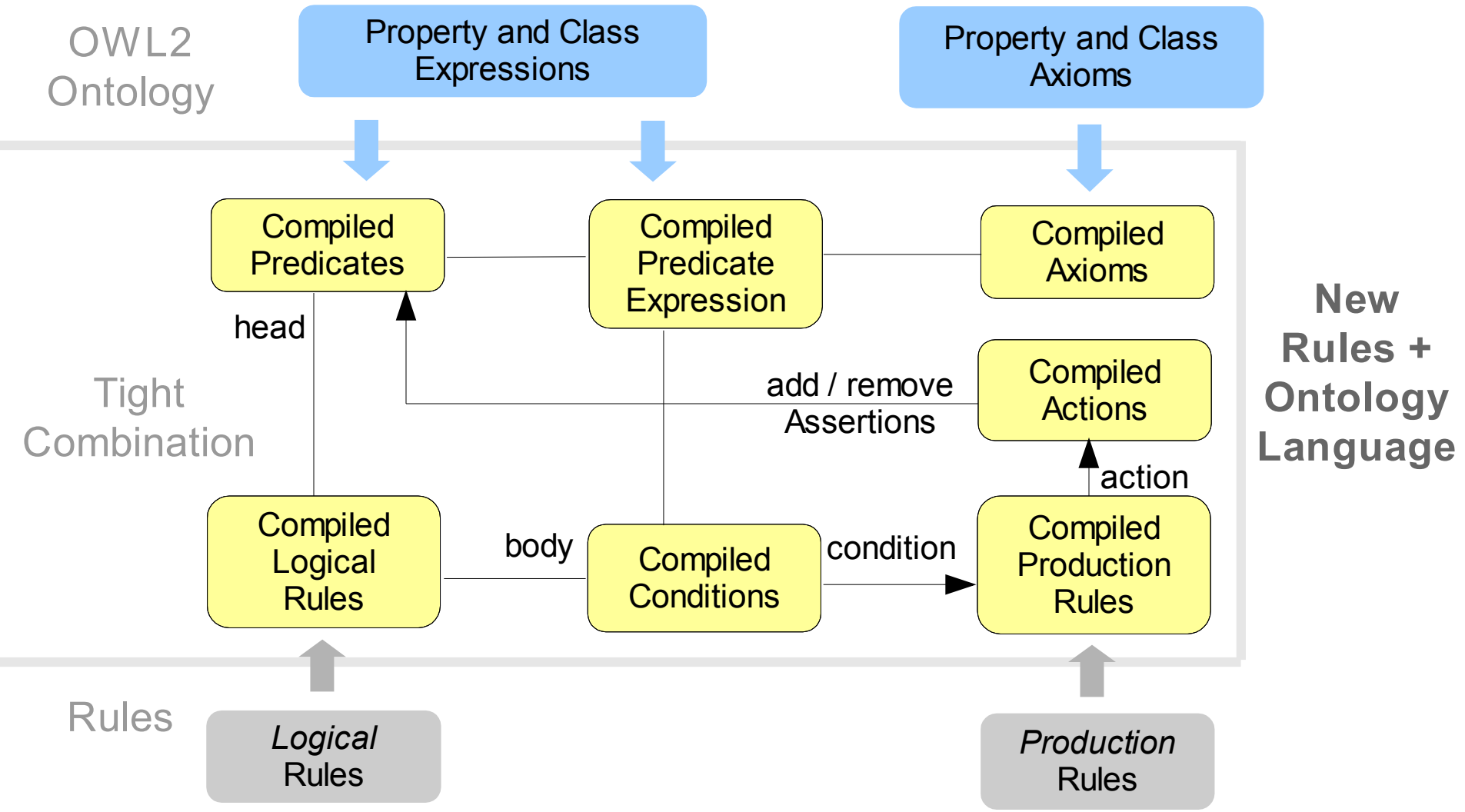
	Logic for Reasoning and Production Rule Emulation (*Theory*)	Production Rule Technology extended for Reasoning (*Practice*)
Impact on Legacy	Big bang	Significant Extension
OWL2 Profile	Any	+/- OWL2RL
Working Memory	Ontology Assertions (*OWA*)	Ontology Assertions (*CWA*)
Identity	Non UNA	UNA
Propagation / Valuation	Backward / Constraints,Unification not needed	Forward, limited Backward / few Constraints
Choice Point (*Search*)	Yes	No
Negation	Logical / Negative Assertions	NAF / No Negative Assertions
Recursion	Yes	Yes
Incrementality	?	Yes
Ontology Update in Engine Cycle	No satisfying answer yet	Yes

A Tight Sample - *Customer Relationship Management*

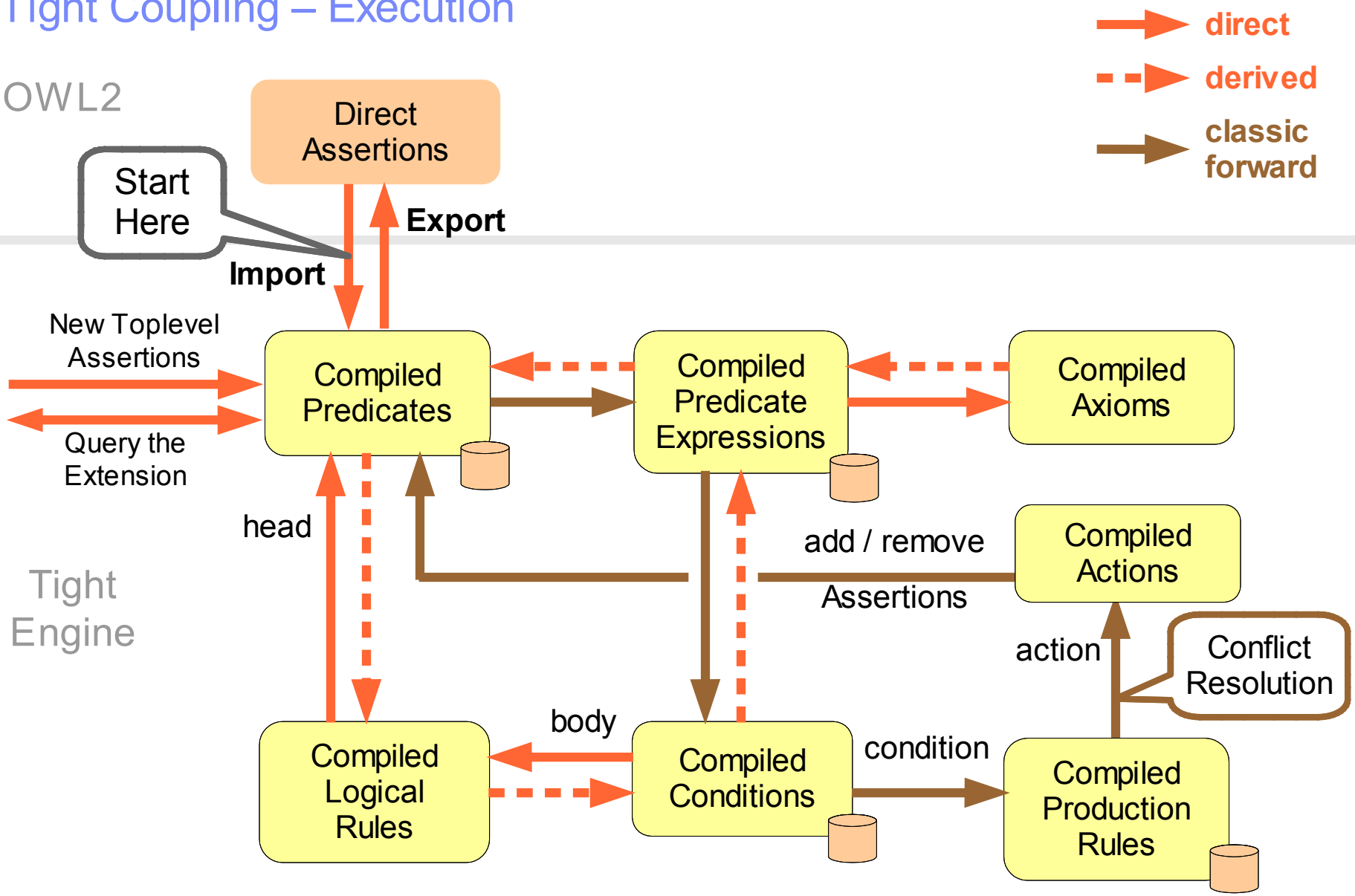
```
class Person;
class Nice;
class GoodCustomer;
class Gift;
class AvailableGift;

class Rich == some(cash) ! integer!(>= 10000);
class VeryRich == some(cash) ! integer!(>= 1000000);

property cash {
   functional;
   domain Person;
   range integer;
};

property givenTo {
   inverse functional;
   functional;
   domain Gift;
   range Person;
};
```

```
logical rule {
   when {  Gift(?g);
           not givenTo(?g,?y);
   }
   then {
     AvailableGift(?g);
   }
}

production rule IdentifyGoodCustomer {
   when {  (VeryRich || (Rich && Nice))(?x);
   }
   then {
     insert GoodCustomer(?x);
   }
}

production rule GiveGift {
   when { GoodCustomer(?x);
          AvailableGift(?g);
   }
   then {
     insert givenTo(?g,?x);
   }
}
```

Tight Coupling - Compilation



Tight Coupling – Execution



Tight Coupling – Getting closer to the Objective

Rule Language

+ OWL2 Constructs are first class citizens of the Language
+ Compatible with OWL2 ad RIF-PRD Standards
- No dedicated authoring Tool yet

Static Update Verification

+ Verified Assertion updates will never introduce an inconsistency that cannot be automatically fixed at runtime

Static Type Checking in Actions

+ Cannot be broken

  Only `Thing` (*the Top Class*) is allowed to type Action Variables

Runtime Inconsistency Fixing Strategy

The most recent Assertion is the right one

+ Now both direct and derived Assertions are fixed

However

Tight is an extension of the Production Rule Technology, not a Logic Engine

- Only a subset of OWL2 is supported (+/- OWL2RL, CWA,UNA)

  Compile time Error Messages are available to warn about Limitations

ONTORULE Experiment

Implementing OWL2RL

Understanding OWL2RL

- The subset of OWL2 Reasoning that can be implemented with a forward chaining Rule Engine

- The OWL2RL Ruleset

  http://www.w3.org/TR/owl-profiles/
  #Reasoning_in_OWL_2_RL_and_RDF_Graphs_using_Rules

  Dedicated to OWL2 Ontologies represented as RDF triples (`T`)
  - `c(i)` becomes `T(i,rdf:type,c)`
  - `p(i,j)` becomes `T(i,p,j)`
  - Classes like `c` and Properties like `p` are also (*higher order*) Individuals and described with RDF triples

  - The execution of the OWL2RL Rules completes the original RDF Graph with all the Knowledge that can be derived from the supported subset of OWL2 Reasoning (*rdf:type, owl:sameAs, ...)*

- Application Rules are simply added to OWL2RL Rules as lower priority Rules

Some OWL2RL Rules (*in Tight Syntax*)

```
production rule prp_dom {
    when {
      T(?p,rdfs:`domain`,?c);
      T(?x,?p,?y);
    }
    then {
      insert T(?x,rdf:type,?c);
    }
}

production rule prp_rng {
    when {
      T(?p,rdfs:`range`,?c);
      T(?x,?p,?y);
    }
    then {
      insert T(?y,rdf:type,?c);
    }
}
```

```
production rule prp_fp {
    when {
      T(?p,rdf:type,owl:FunctionalProperty);
      T(?x,?p,?y1);
      T(?x,?p,?y2);
    }
    then {
      insert T(?y1,owl:sameAs,?y2);
    }
}

production rule scm_sco {
    when {
      T(?c1,rdfs:subClassOf,?c2);
      T(?c2,rdfs:subClassOf,?c3);
    }
    then {
      insert T(?c1,rdfs:subClassOf,?c3);
    }
}
```

OWL2RL – Feedback

PROS

- A subset of OWL2 Reasoning can be implemented with an off-the-shelf forward chaining Rule Engine (Logic or Production)
- Highly comprehensive Ruleset for the Individual side

CONS

OWL2RL

- Restrictions on supported OWL2 constructs are hard to understand
- When implemented with Production Rules : Once all the Rules have fired, it is hard to distinguish between useful (*long-term*) direct knowledge and derived (*temporary*) Knowledge

Ruleset

- An advanced Rule Engine is required

  Some complex OWL2RL Rules require open List matching, have variable length Conditions or need conjunction in the head
- Almost no OWL2RL Rules are provided for reasoning on Datatypes and built-in Constraint Operators : be ready to add quite a few ...

Conclusion

Conclusion

- OWL2 is **not** an Object Model
  It is a Logic, Knowledge is available as Facts, not as complex Terms (Objects)

- The Weakly Coupled Approaches are limited
  Not incremental Query interface of the OWL2 Reasoner
  Performance of the OWL2 Reasoner

- The Tightly Coupled Approach is promising but putting together Production Rules and OWL2 Ontology requires tradeoffs

  When a Production Rule Technology is extended (*Tight*)
  Update is here but only a subset of OWL2 Logic is here

  When a (Constraint) Logic Programming Technology is used (*no demonstrator*)
  Full OWL2 Logic is here but Update is unlikely to be here

Thank you !

Questions and Answers