Production Rule Systems (Drools)

Rule-based systems

- Rules are the **main way** to express **knowledge** in many fileds of I.A.
- Most common rules are:
  - *logic programs* (eg.: Prolog)
  - *production rules* (eg.: Drools)
- They are similar, but realized **in a dual way**

Rule-based systems

- **Modus Ponens**:

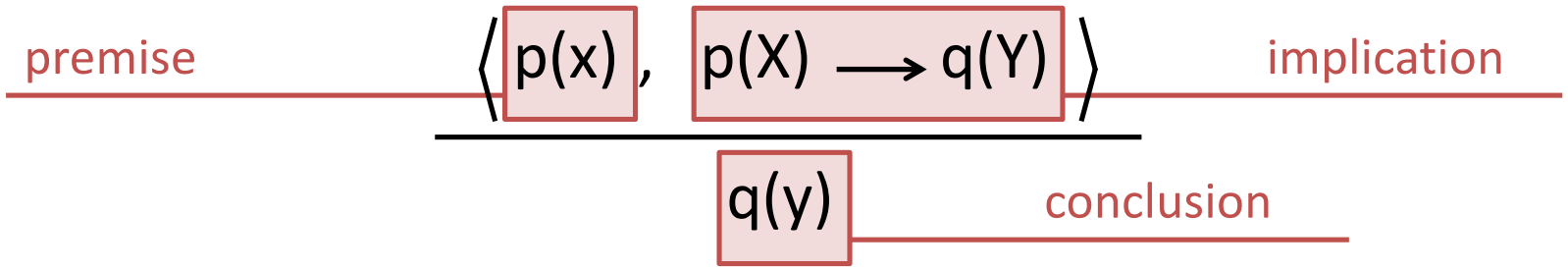
$$\frac{\langle\, p(x)\ ,\quad p(X) \longrightarrow q(Y)\ \rangle}{q(y)}$$

if it holds that p(X) *implies* q(Y) *and* p(x) *holds, then* q(y) *holds*

- **Es.:** If it rains, then the street is wet.
  Here it rains.
  Then, here the street is wet.

Rule-based systems

- Il **Modus Ponens**:



if it holds that p(X) *implies* q(Y) *and* p(x) *holds, then* q(y) *holds*

- **Es.:** If it rains, then the street is wet. — implication
  Here it rains. — premise
  Then, here the street is wet. — conclusion

Rule-based systems

Logic programs

- *Backward-chaining*
- From goal to facts, applying rules in a backward way
- Unification
- Backtracking

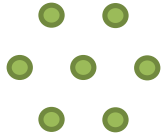
Production rules

- *Forward-chaining*
- The facts activate rules that generate new facts
- Pattern matching
- Parallelism

Rule-based systems

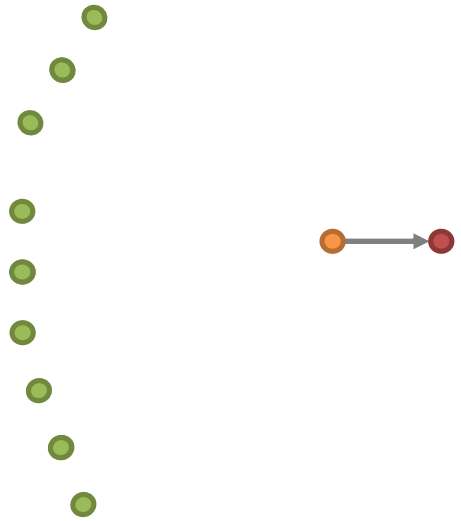
Logic program

Production rules

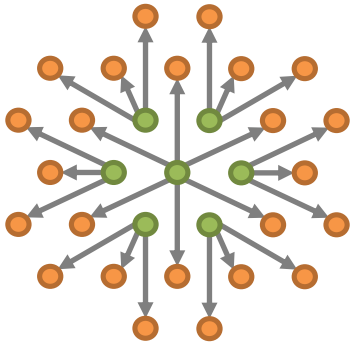


Rule-based systems

Logic program

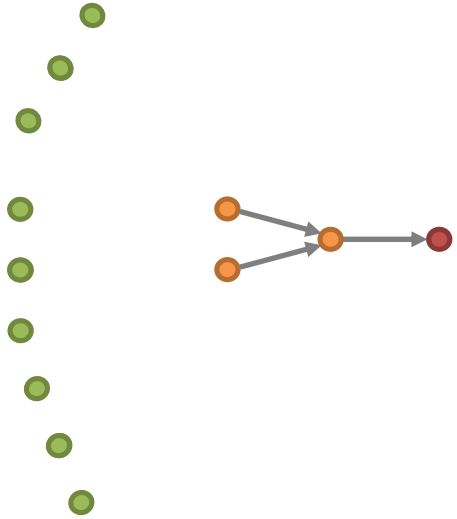


Production rules

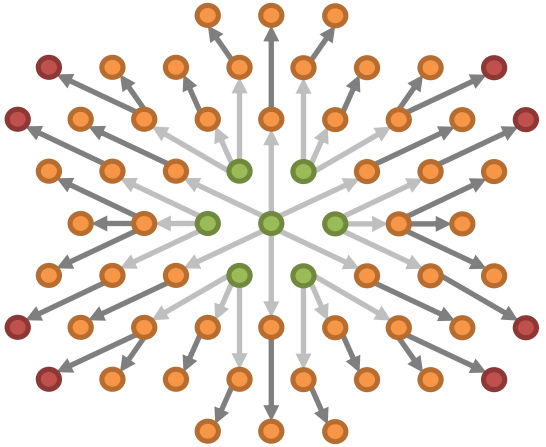


Rule-based systems

Logic program

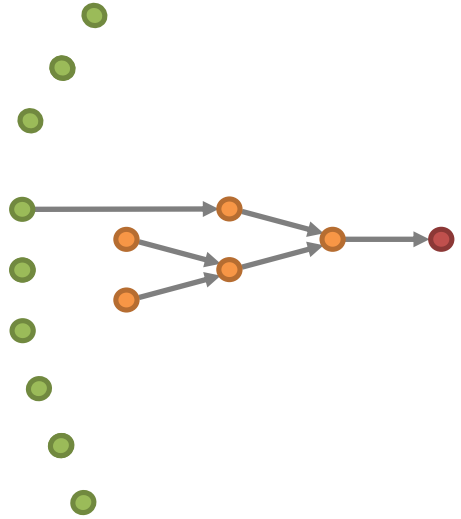


Production rules

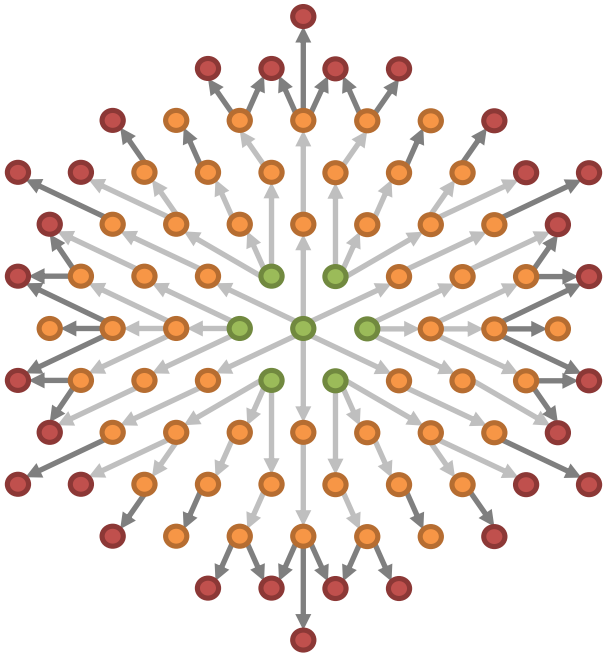


Rule-based systems

Logic program

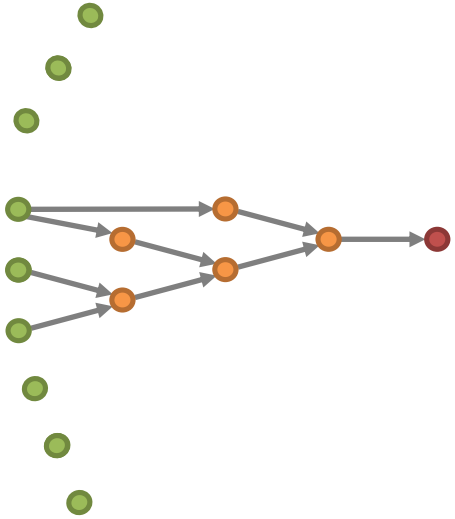


Production rules

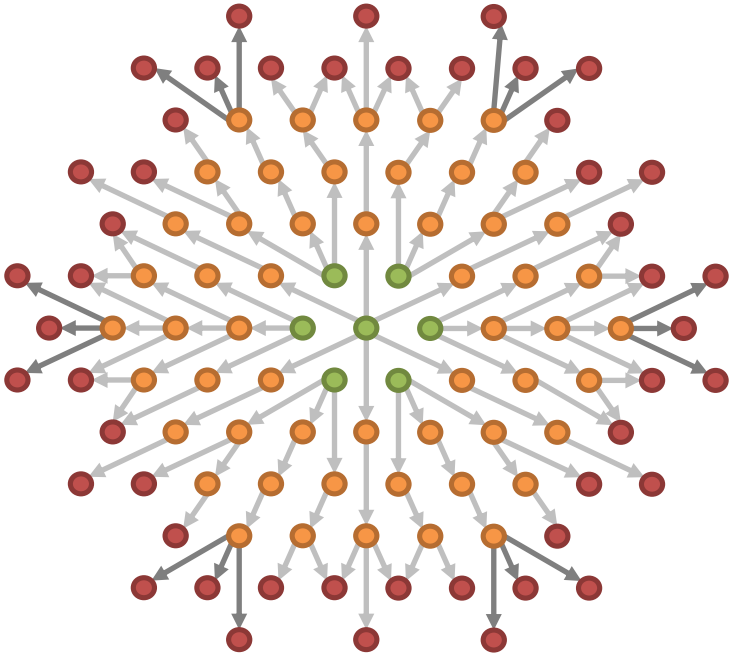


Rule-based systems

Logic program

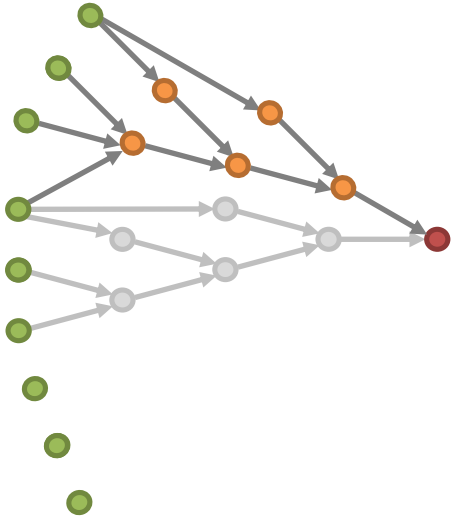


Production rules

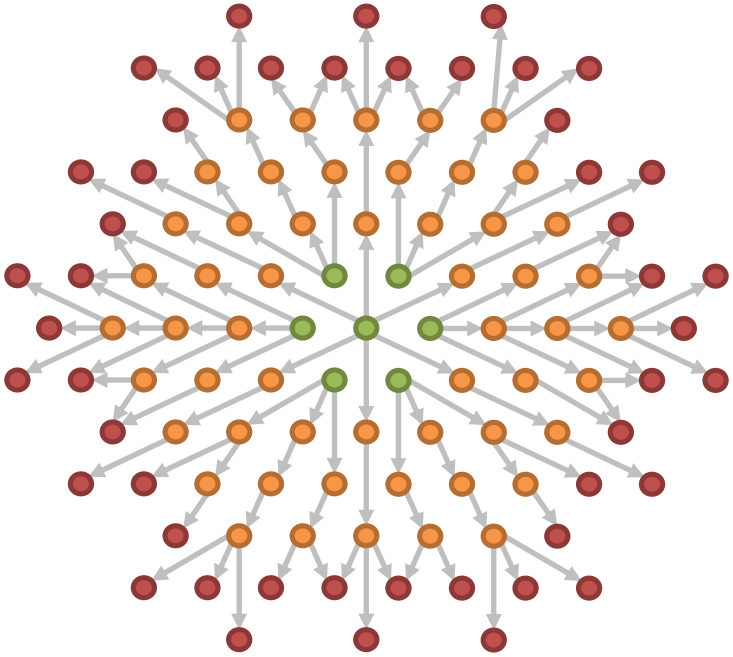


Rule-based systems

Logic program

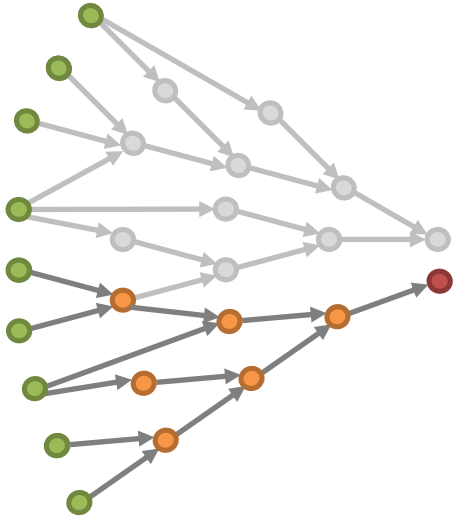


Production rules

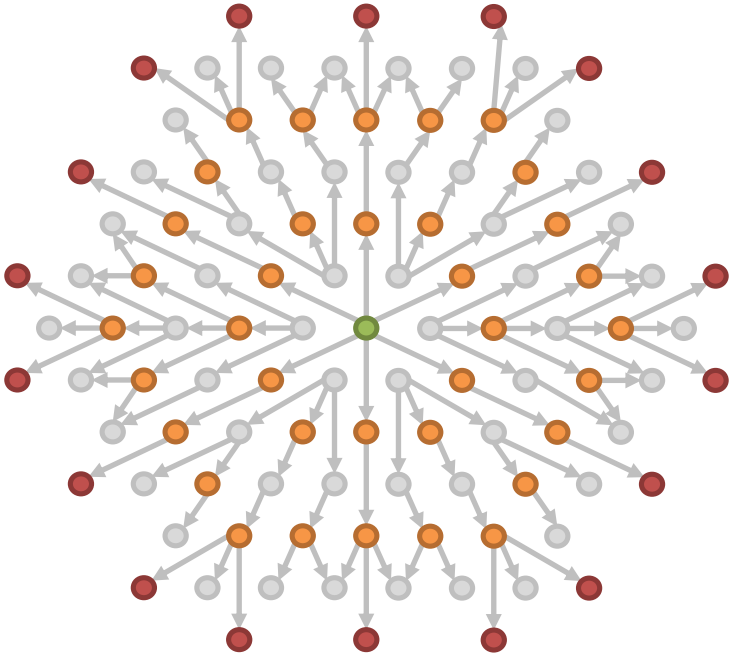


Rule-based systems

Logic program



Production rules



Production Rule Systems

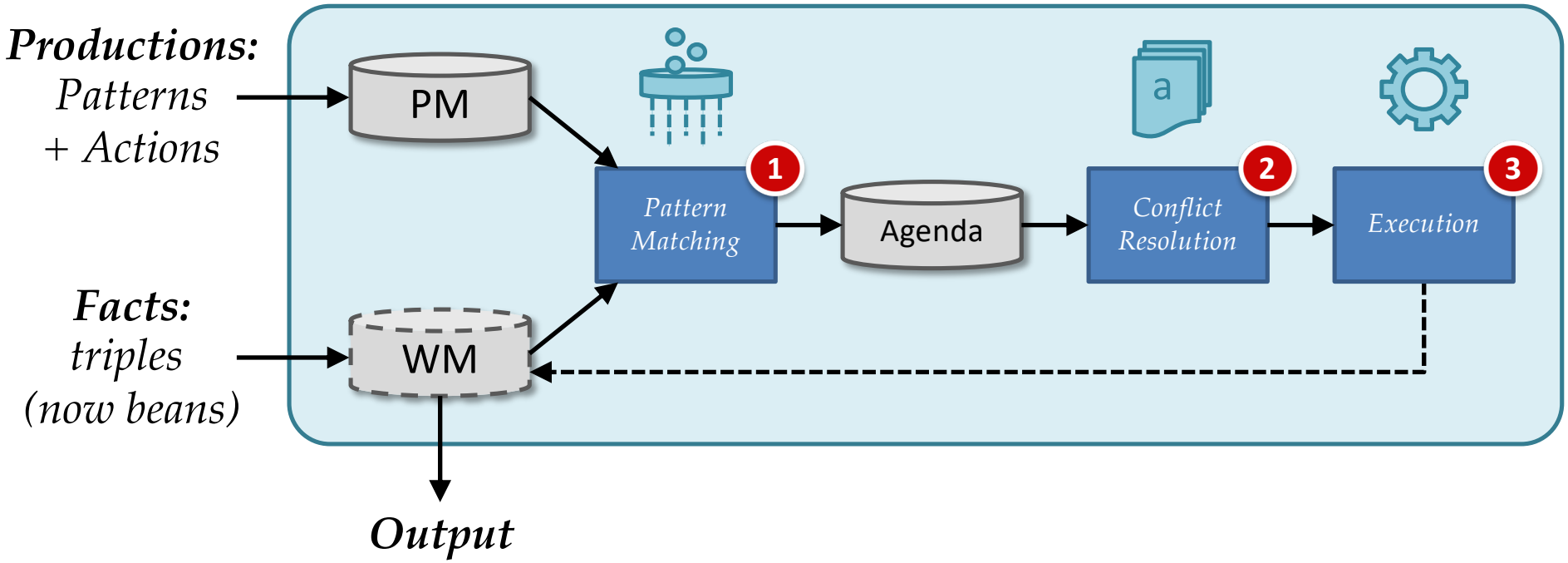
- **Production Rule Systems (PRS)**:
  - are **Rule Based Systems (RBS)**,
  - are based on the **Modus Ponens** principle,
  - rely on a **reactive/generative approach**

When a PRS is a right choice?

- The problem is **too complex** for traditional coding approaches: rules provide a more abstract view, preventing fragile implementations
- The problem is **not fully known**
- **Flexibility**, when system logic changes often over time
- Domain knowledge readily available

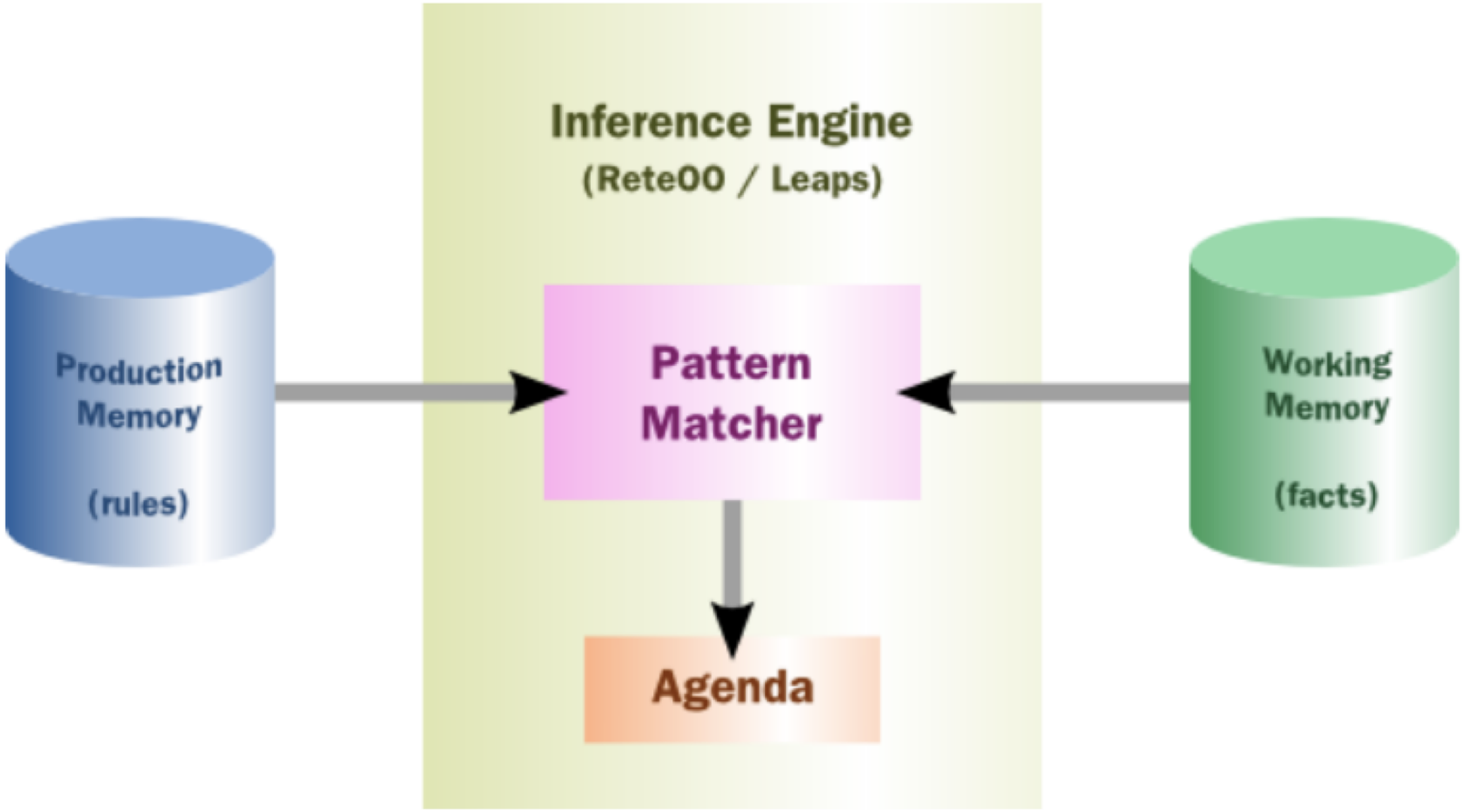
Production Rule Systems

Architecture and working schema



Production Rule Systems

Architecture and working schema

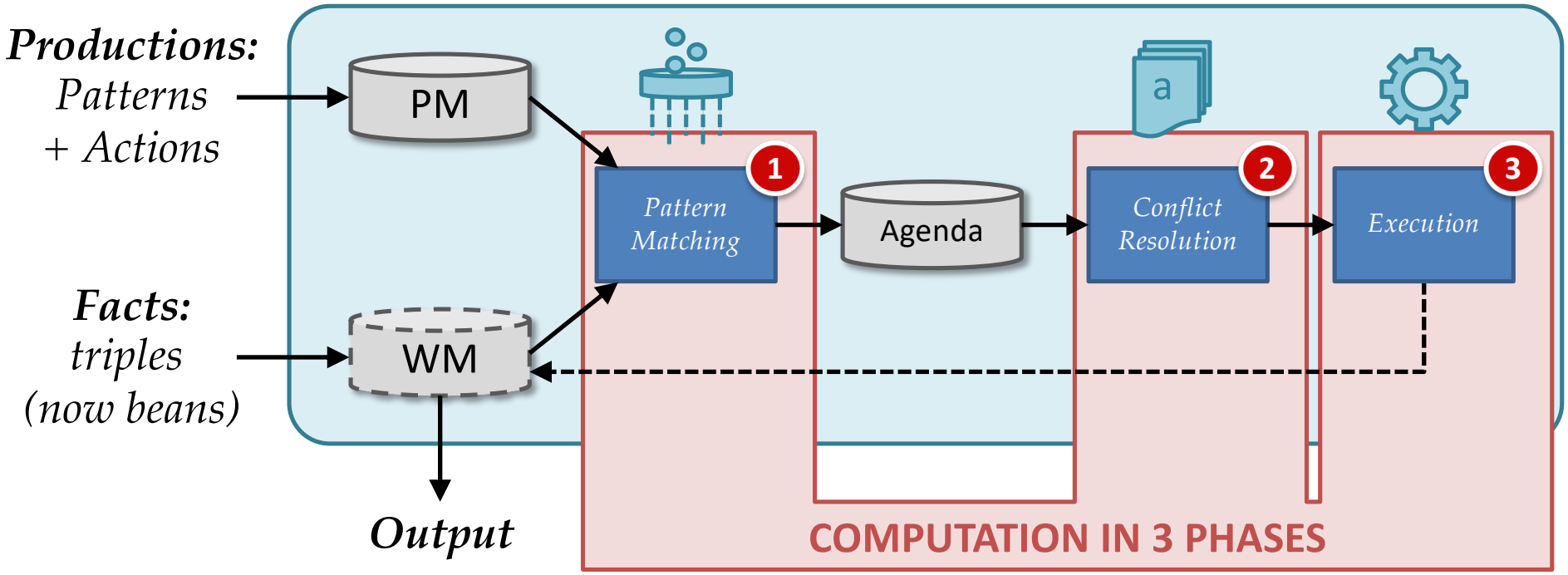


Production Rule Systems

- Rules are stored in the **Production Memory (PM)**
- Facts are stored in the **Working Memory (WM)**, where they can be changed or retracted
- **Inference engine** applies to data in the WM the rules in in the PM to deduce new information
- The **Agenda** deals with the execution order in case of conflicts, using conflict resolution strategies

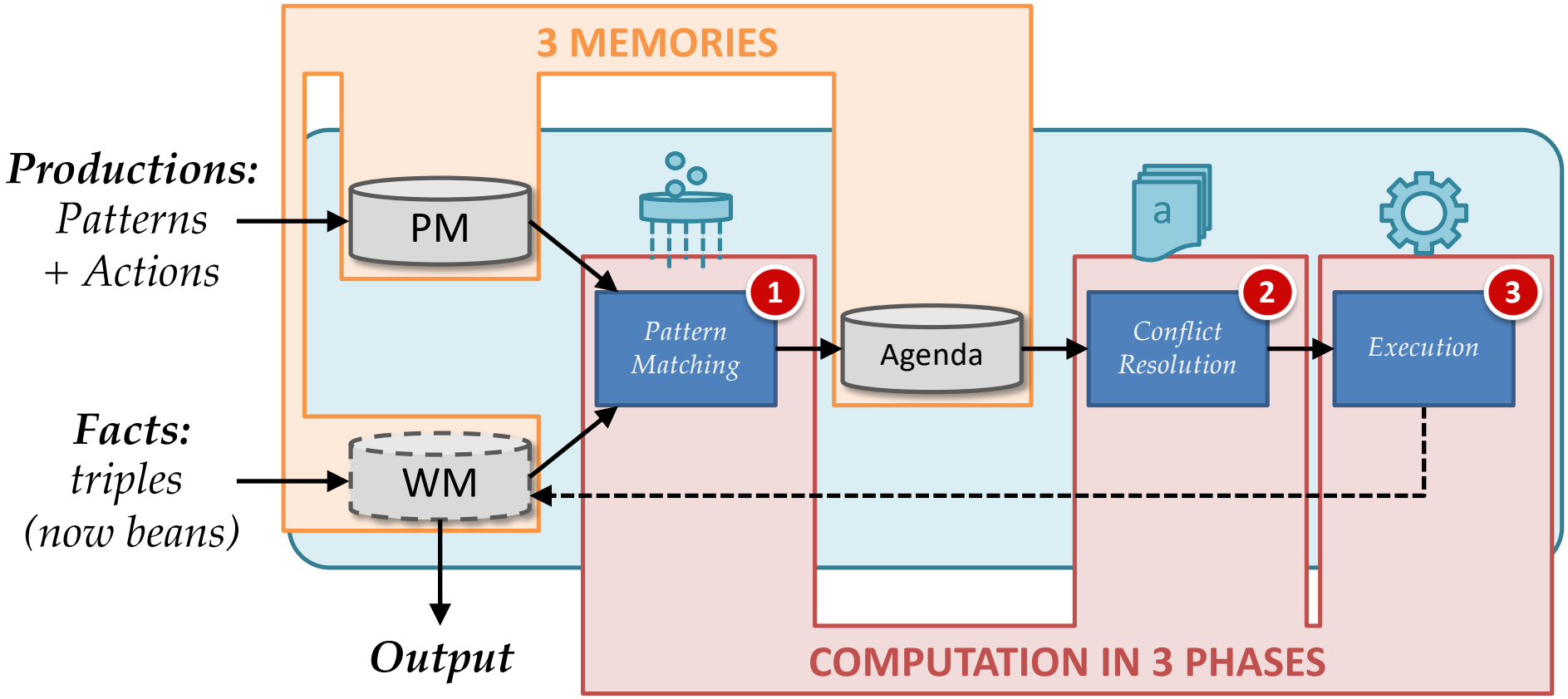
Production Rule Systems

Architecture and working schema



Production Rule Systems

Architecture and working schema



JBOSS DROOLS

JBoss Drools

- **Alternative**
  - OPS5, CLIPS, Jess, ILOG, Jrules, BizTalk, …
- **Reference:**
  - JBoss Drools (http://www.jboss.org/drools)
- **Why?**
  - Open source, Java-based, integrated with Eclipse
- **Integrated Platform**

 *Expert* (rule engine)

 *Fusion* (event processing)

 *Guvnor* (rule repo)

 *jBPM* (workflow)

 *Planner* (constraints)

JBoss Drools

- **Alternative**
  - OPS5, CLIPS, Jess, ILOG, Jrules, BizTalk, …
- **Sistema di riferimento**
  - JBoss Drools (http://www.jboss.org/drools)
- **Perchè?**
  - Open source, Java-based, integrato con Eclipse
- **Parte di una piattaforma integrata**


Expert (rule engine)


Fusion (event processing)


Guvnor (rule repo)


jBPM (workflow)


Planner (constraints)

JBoss Drools

- Syntax of Drools language: **rules**

```
rule "ID_regola"                    /* IMPLICATION */
// attributes
when                                    /* premise*/
  // pattern(s)
then                                   /* conclusion*/
  // logical actions
  // side effects
end
```

JBoss Drools

- Syntax of Drools language: **rules**

```
rule ”Delete Francescos"    /* IMPLICATION*/
salience 5
when                                  /* premise*/
  $p: Person ( name=="Francesco" )
then                                /* conclusion*/
  retract($p);
  System.out.println($p);
end
```

PATTERN MATCHING:
RETE ALGORITHM

Pattern matching: RETE algorithm

EXAMPLE 1

Pattern matching: RETE algorithm

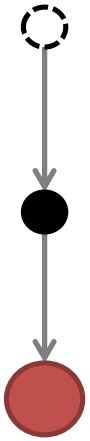
```
rule ”Find Francescos"
when
  $p: Person( name=="Francesco" )
then
  System.out.println($p);
end
```

Root node

Entry Point nodes

Pattern matching: RETE algorithm

```
rule "Find Francescos"
when
  $p: Person( name=="Francesco" )
then
  System.out.println($p);
end
```



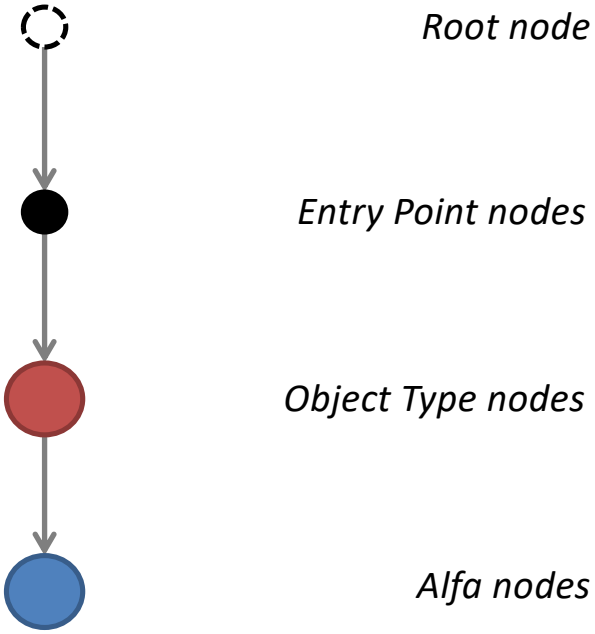
Root node

Entry Point nodes

Object Type nodes

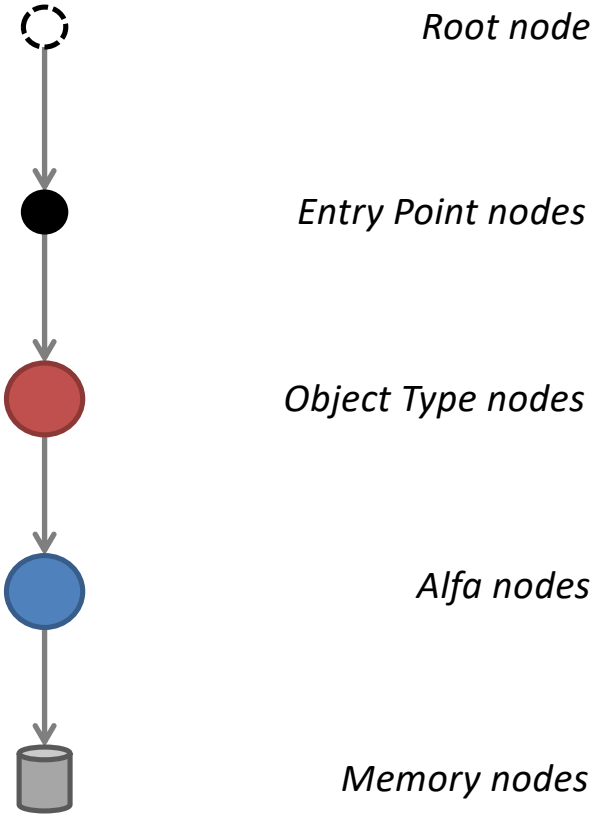
Pattern matching: RETE algorithm

```
rule "Find Francescos"
when
  $p: Person( name=="Francesco" )
then
  System.out.println($p);
end
```



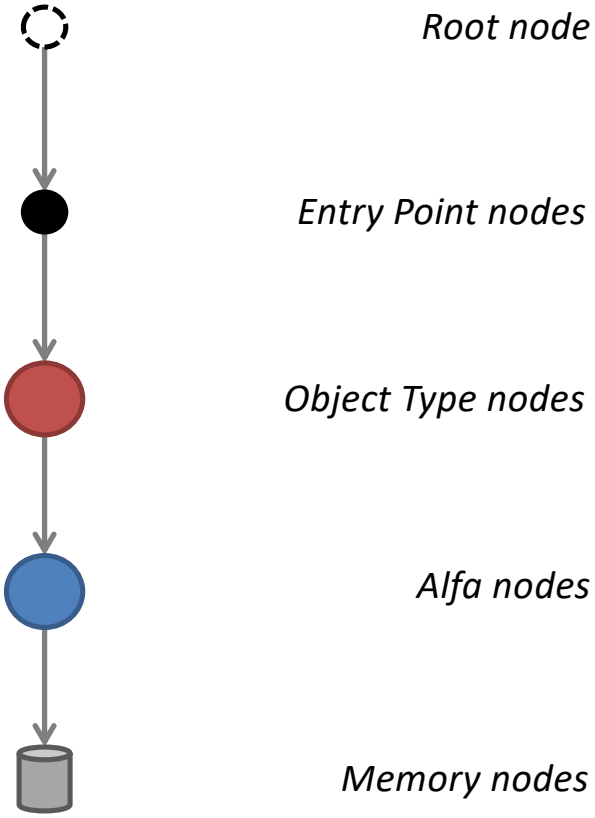
Pattern matching: RETE algorithm

```
rule "Find Francescos"
when
  $p: Person( name=="Francesco" )
then
  System.out.println($p);
end
```



Pattern matching: RETE algorithm

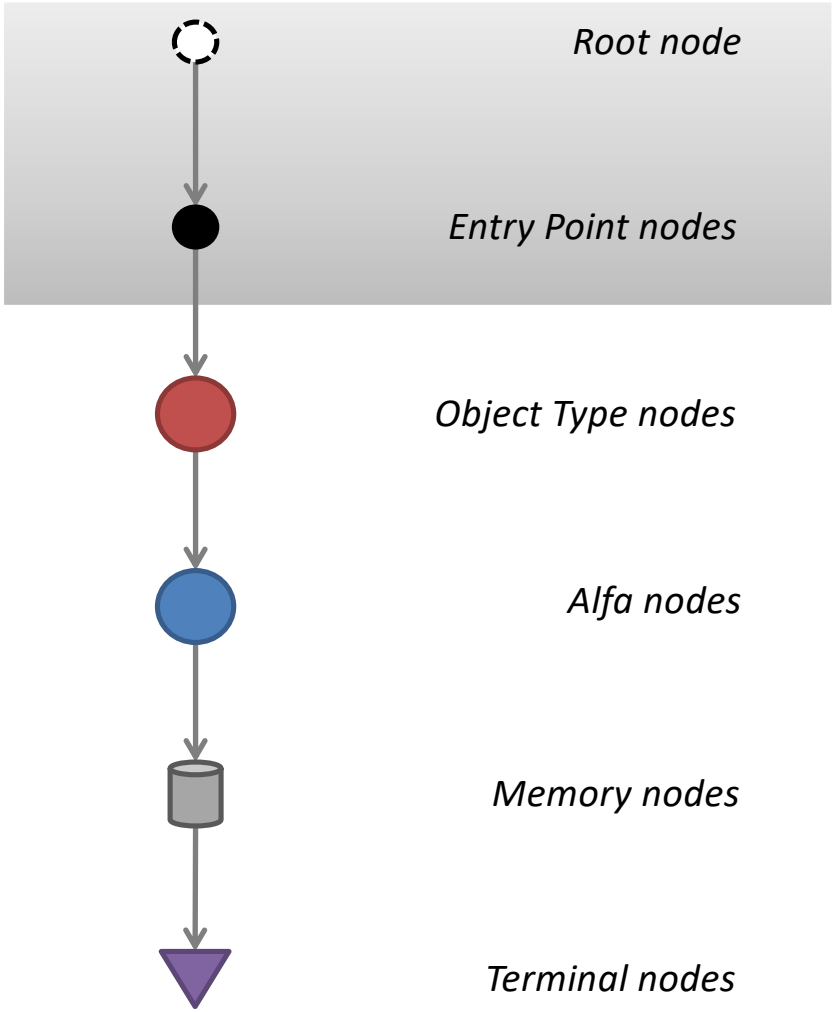
```
rule "Find Francescos"
when
  $p: Person( name=="Francesco" )
then
  System.out.println($p);
end
```



NB: facts in a (Alfa) Memory Node match with a simple pattern!

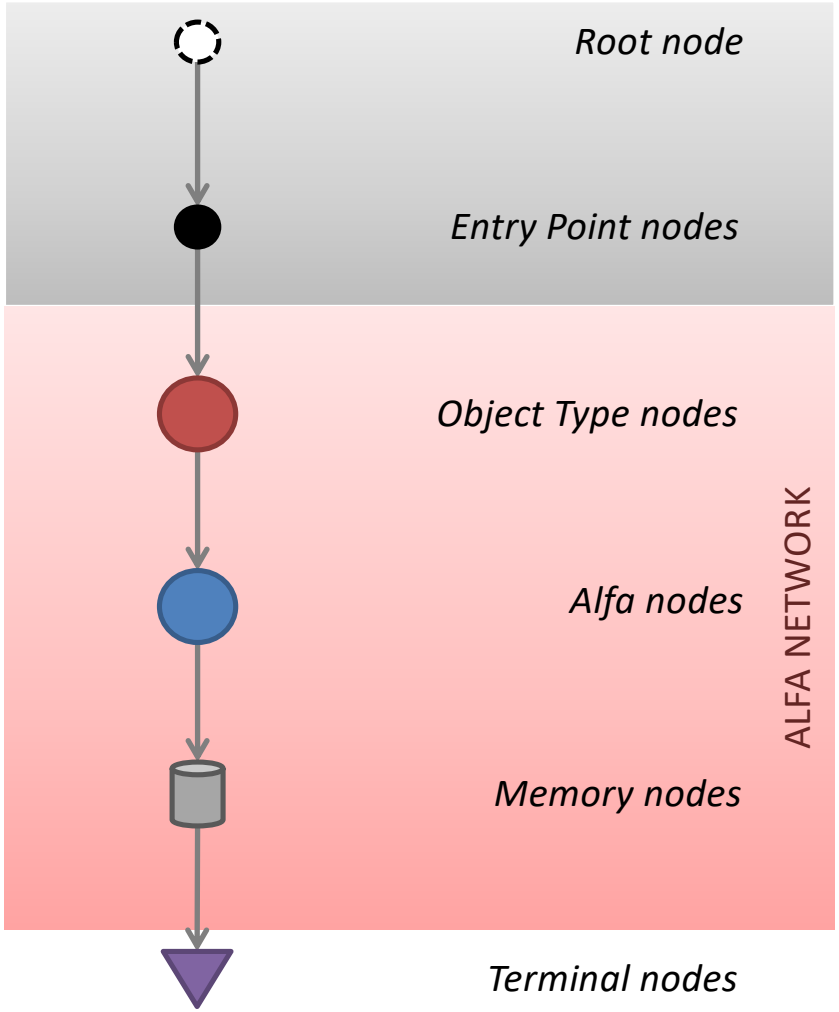
Pattern matching: RETE algorithm

```
rule "Find Francescos"
when
  $p: Person( name=="Francesco" )
then
  System.out.println($p);
end
```



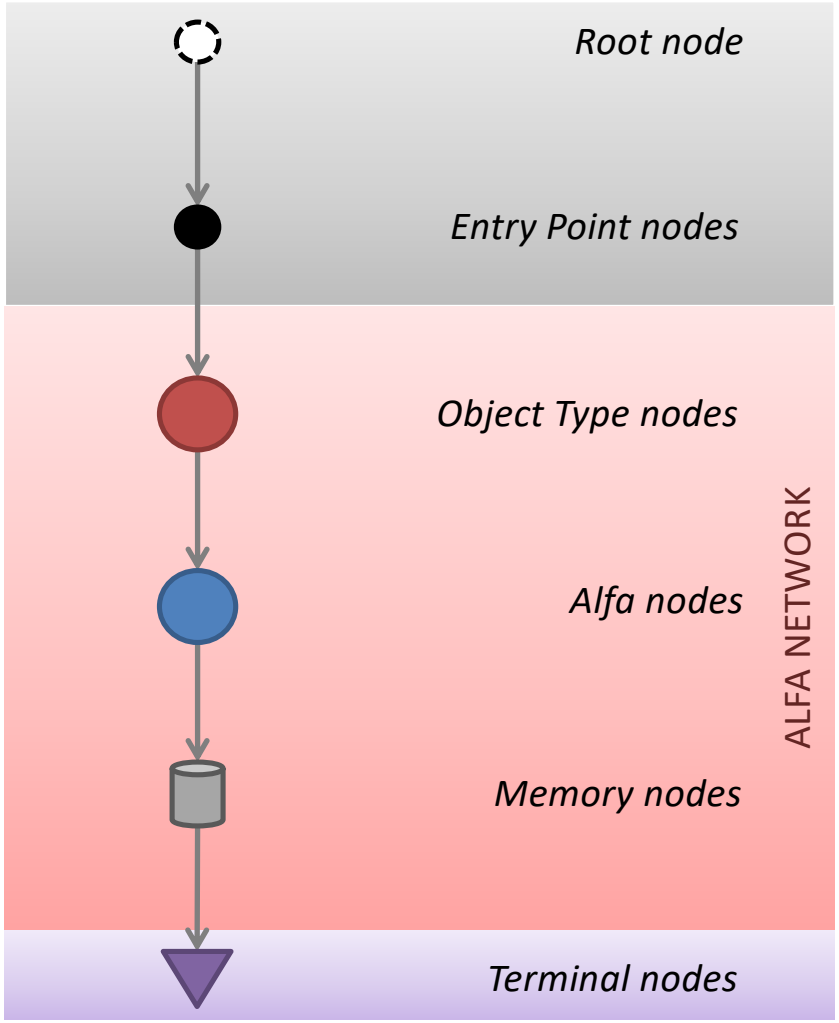
Pattern matching: RETE algorithm

```
rule "Find Francescos"
when
  $p: Person( name=="Francesco" )
then
  System.out.println($p);
end
```



Pattern matching: RETE algorithm

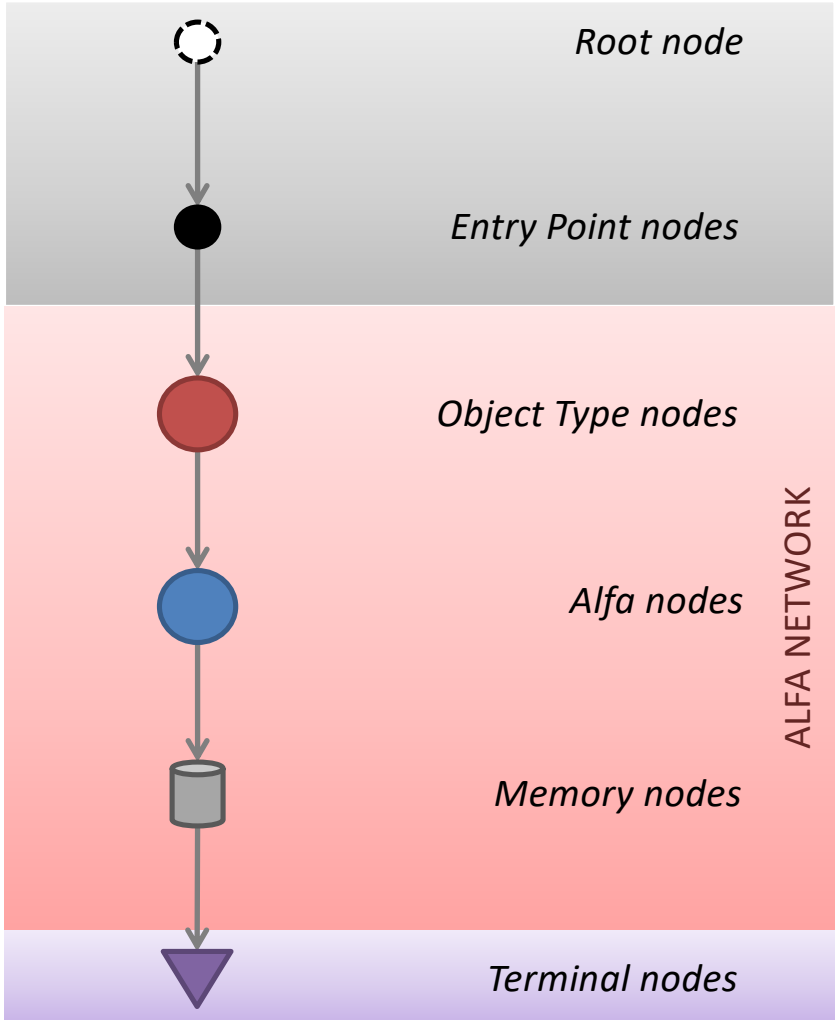
```
rule "Find Francescos"
when
  $p: Person( name=="Francesco" )
then
  System.out.println($p);
end
```



Pattern matching: RETE algorithm

```
rule "Find Francescos"
when
  $p: Person( name=="Francesco" )
then
  System.out.println($p);
end
```


```
_
```

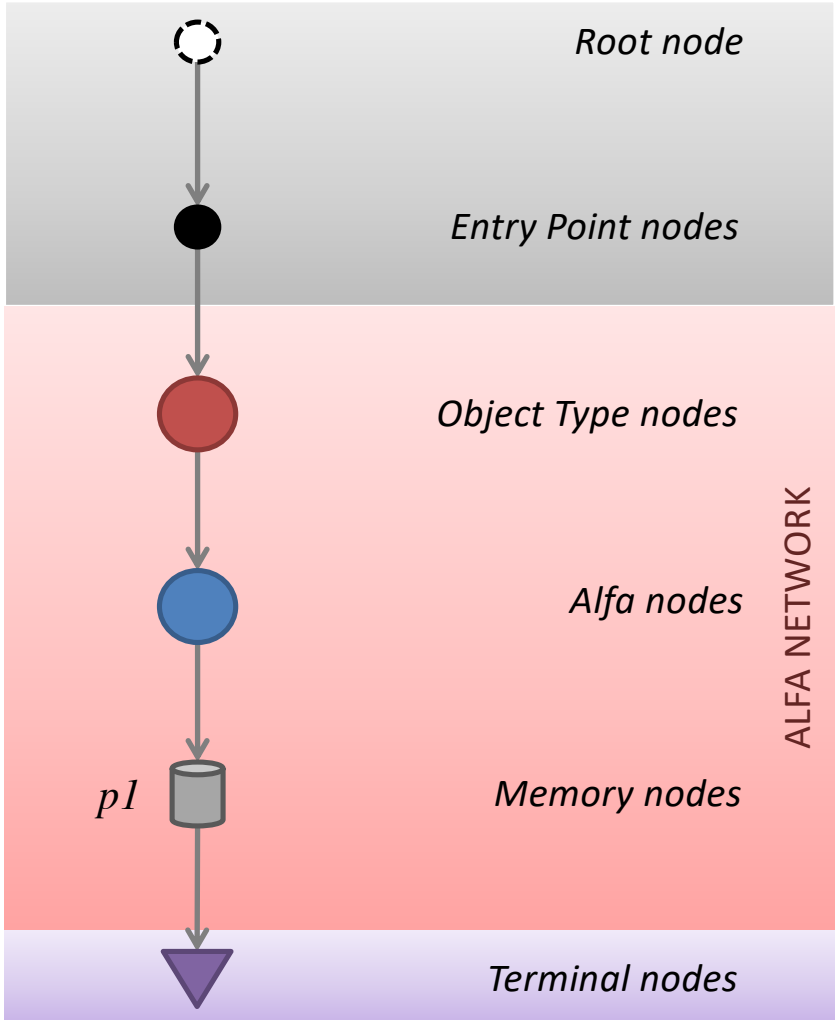


Pattern matching: RETE algorithm

```
rule "Find Francescos"
when
  $p: Person( name=="Francesco" )
then
  System.out.println($p);
end
```

p1: Person("Francesco", null)


```
Person[Francesco, <null>]
_
```

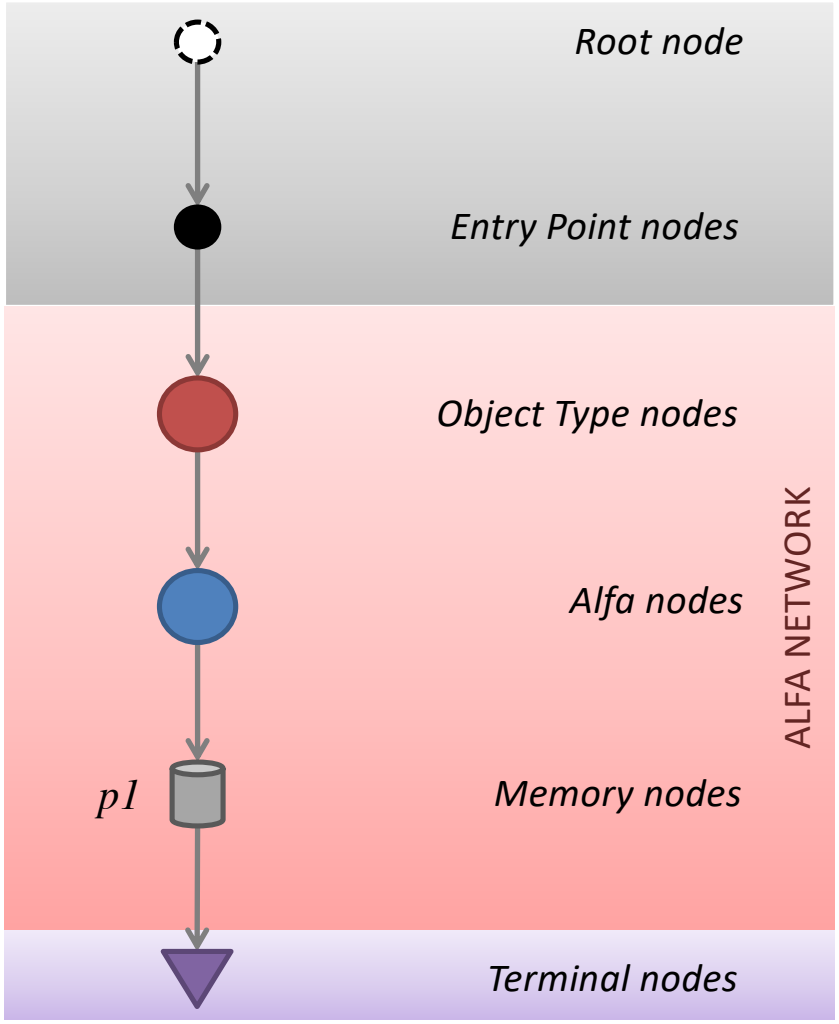


Pattern matching: RETE algorithm

```
rule "Find Francescos"
when
  $p: Person( name=="Francesco" )
then
  System.out.println($p);
end
```

p1: Person("Francesco", null)
a1: Address("Via Po 2", 40068, "San Lazzaro")

```
Person[Francesco, <null>]
_
```

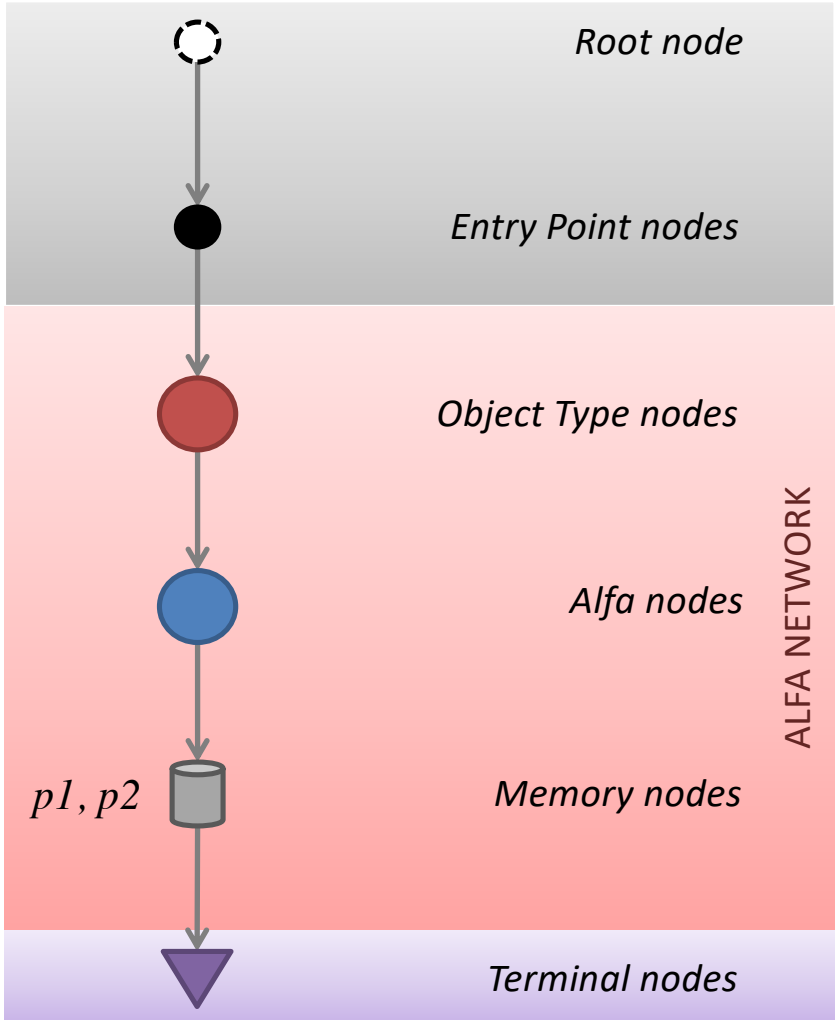


Pattern matching: RETE algorithm

```
rule "Find Francescos"
when
  $p: Person( name=="Francesco" )
then
  System.out.println($p);
end
```

p1: Person("Francesco", null)
a1: Address("Via Po 2", 40068, "San Lazzaro")
p2: Person("Francesco", a1)

```
Person[Francesco, <null>]
Person[Francesco, Address[Via Po 2, 40068, San Lazzaro]]
_
```

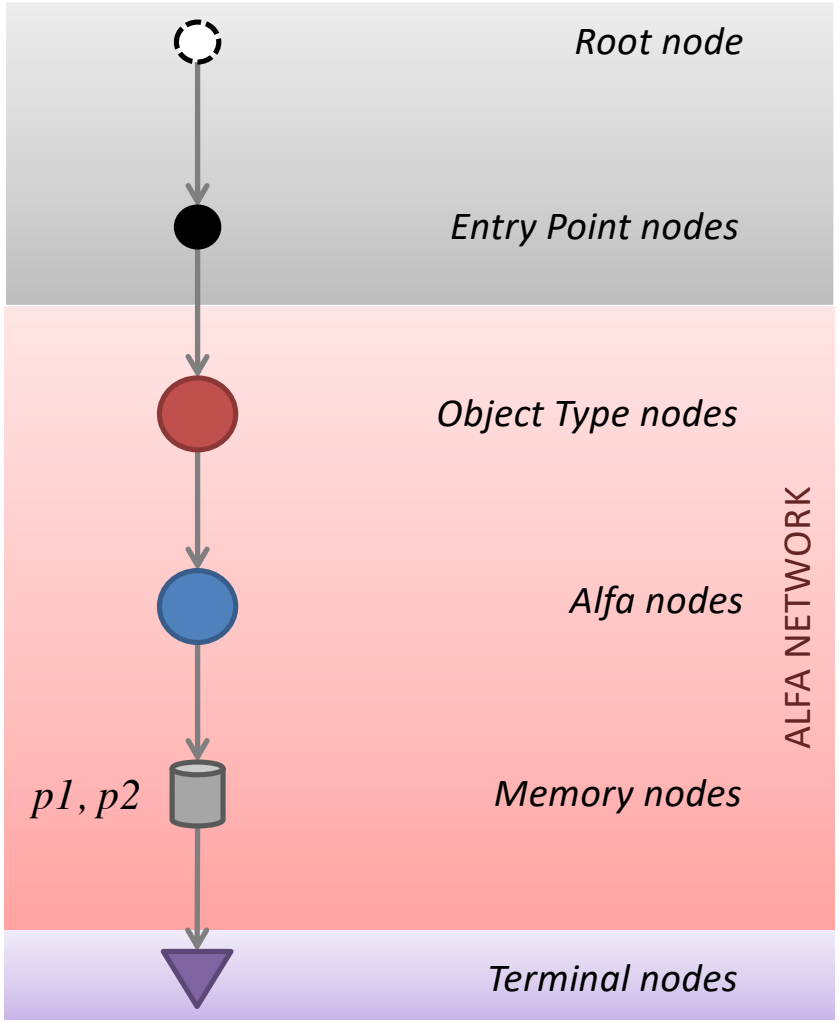


Pattern matching: RETE algorithm

```
rule "Find Francescos"
when
  $p: Person( name=="Francesco" )
then
  System.out.println($p);
end
```

p1: Person("Francesco", null)
a1: Address("Via Po 2", 40068, "San Lazzaro")
p2: Person("Francesco", a1)
p3: Person("Giacomo", a1)

```
Person[Francesco, <null>]
Person[Francesco, Address[Via Po 2, 40068, San Lazzaro]]
_
```



Pattern matching: RETE algorithm

EXAMPLE 2

Pattern matching: RETE algorithm

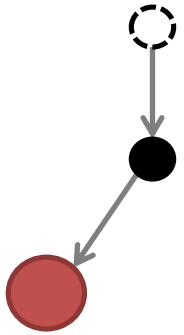
```
rule "Find Francescos and addresses"
when
  $a: Address()
  $p: Person( name=="Francesco" )
then
  System.out.println($p+"/"+$a+" ");
end
```

Root node

Entry Point nodes

Pattern matching: RETE algorithm

```
rule "Find Francescos and addresses"
when
  $a: Address()
  $p: Person( name=="Francesco" )
then
  System.out.println($p+"/"+$a+" ");
end
```



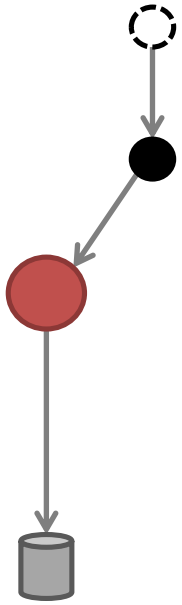
Root node

Entry Point nodes

Object Type nodes

Pattern matching: RETE algorithm

```
rule "Find Francescos and addresses"
when
  $a: Address()
  $p: Person( name=="Francesco" )
then
  System.out.println($p+"/"+$a+" ");
end
```



Root node

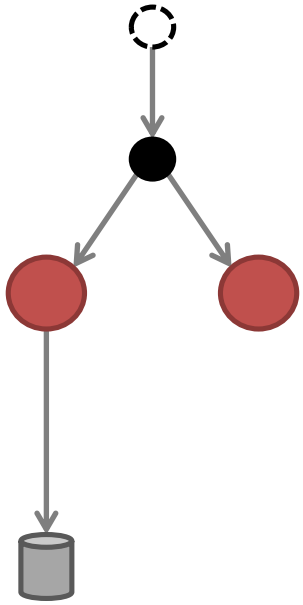
Entry Point nodes

Object Type nodes

Memory nodes

Pattern matching: RETE algorithm

```
rule "Find Francescos and addresses"
when
  $a: Address()
  $p: Person( name=="Francesco" )
then
  System.out.println($p+"/"+$a+" ");
end
```



Root node

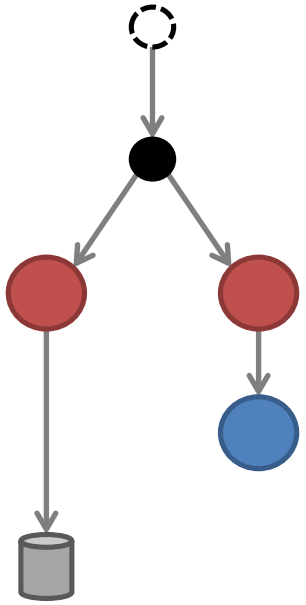
Entry Point nodes

Object Type nodes

Memory nodes

Pattern matching: RETE algorithm

```
rule "Find Francescos and addresses"
when
  $a: Address()
  $p: Person( name=="Francesco" )
then
  System.out.println($p+"/"+$a+" ");
end
```



Root node

Entry Point nodes

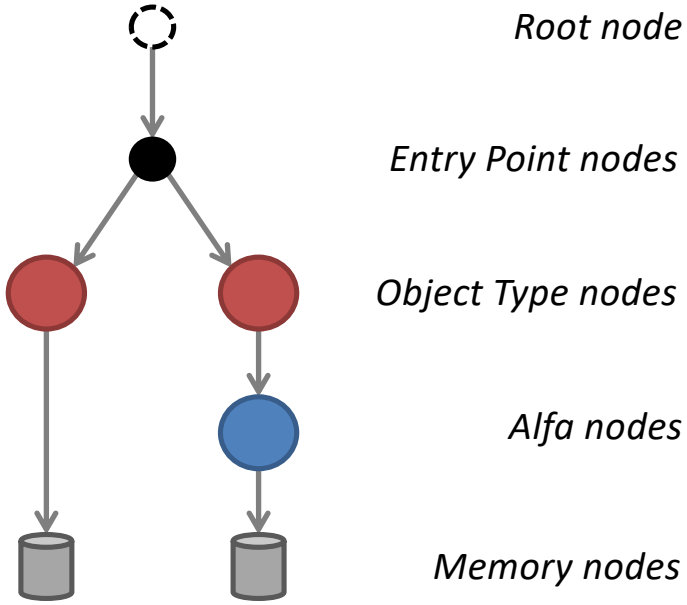
Object Type nodes

Alfa nodes

Memory nodes

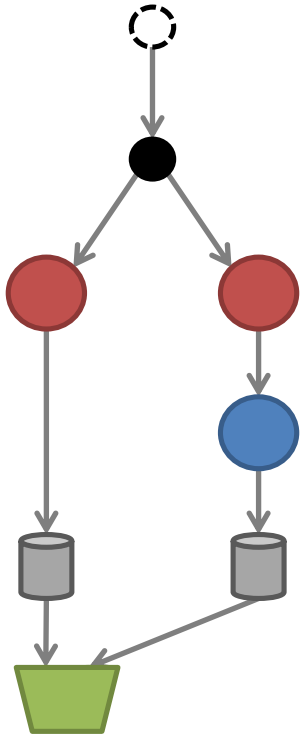
Pattern matching: RETE algorithm

```
rule "Find Francescos and addresses"
when
  $a: Address()
  $p: Person( name=="Francesco" )
then
  System.out.println($p+"/"+$a+" ");
end
```



Pattern matching: RETE algorithm

```
rule "Find Francescos and addresses"
when
  $a: Address()
  $p: Person( name=="Francesco" )
then
  System.out.println($p+"/"+$a+" ");
end
```



Root node

Entry Point nodes

Object Type nodes

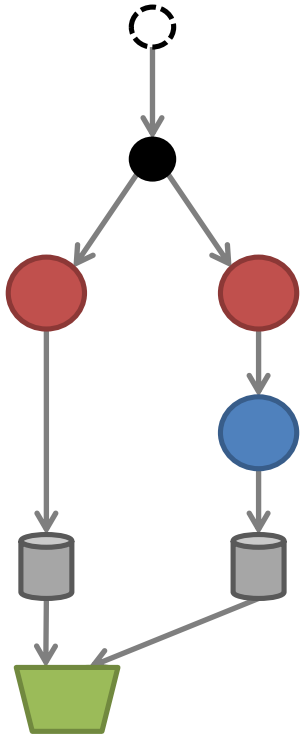
Alfa nodes

Memory nodes

Beta nodes

Pattern matching: RETE algorithm

```
rule "Find Francescos and addresses"
when
  $a: Address()
  $p: Person( name=="Francesco" )
then
  System.out.println($p+"/"+$a+" ");
end
```



Root node

Entry Point nodes

Object Type nodes

Alfa nodes

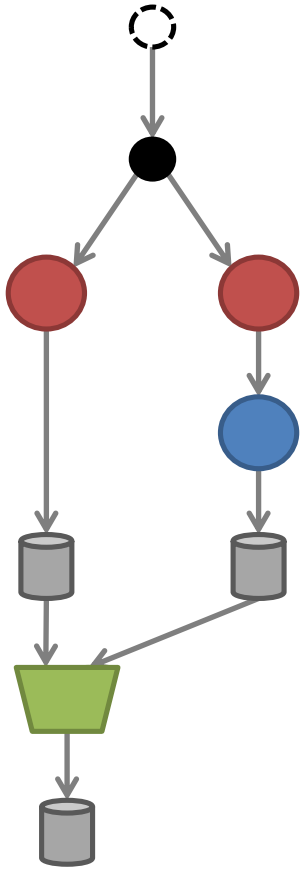
Memory nodes

Beta nodes

NB: Beta Nodes make cartesian product of objects filtered by Alfa father!

Pattern matching: RETE algorithm

```
rule "Find Francescos and addresses"
when
  $a: Address()
  $p: Person( name=="Francesco" )
then
  System.out.println($p+"/"+$a+" ");
end
```



Root node

Entry Point nodes

Object Type nodes

Alfa nodes

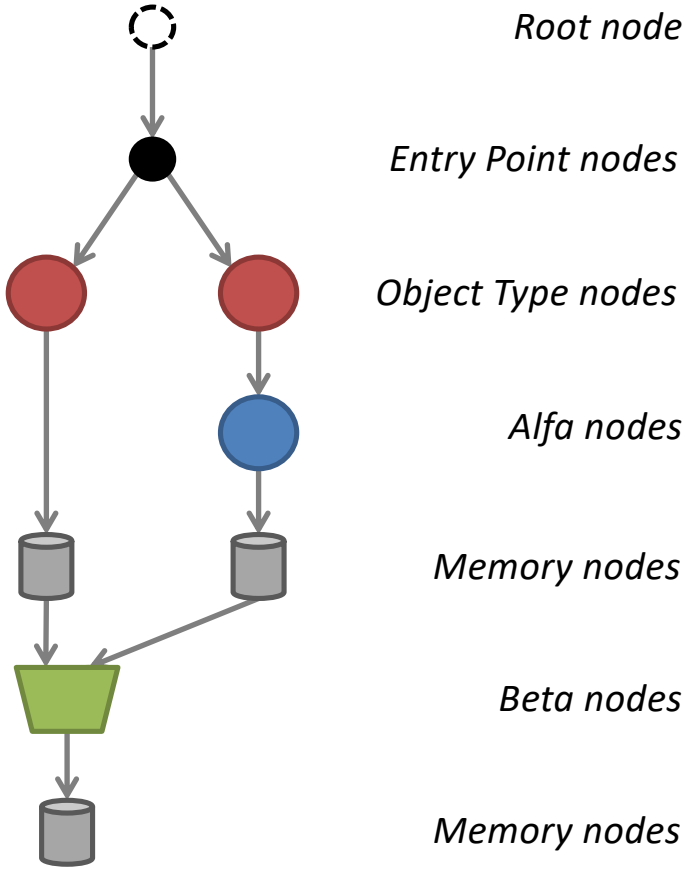
Memory nodes

Beta nodes

Memory nodes

Pattern matching: RETE algorithm

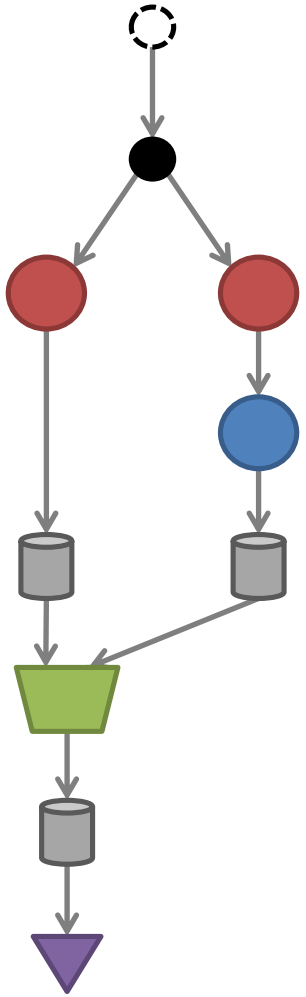
```
rule "Find Francescos and addresses"
when
  $a: Address()
  $p: Person( name=="Francesco" )
then
  System.out.println($p+"/"+$a+" ");
end
```



NB: tuple in a(Beta) Memory Node match with a composite pattern!

Pattern matching: RETE algorithm

```
rule "Find Francescos and addresses"
when
  $a: Address()
  $p: Person( name=="Francesco" )
then
  System.out.println($p+"/"+$a+" ");
end
```



Root node

Entry Point nodes

Object Type nodes

Alfa nodes

Memory nodes

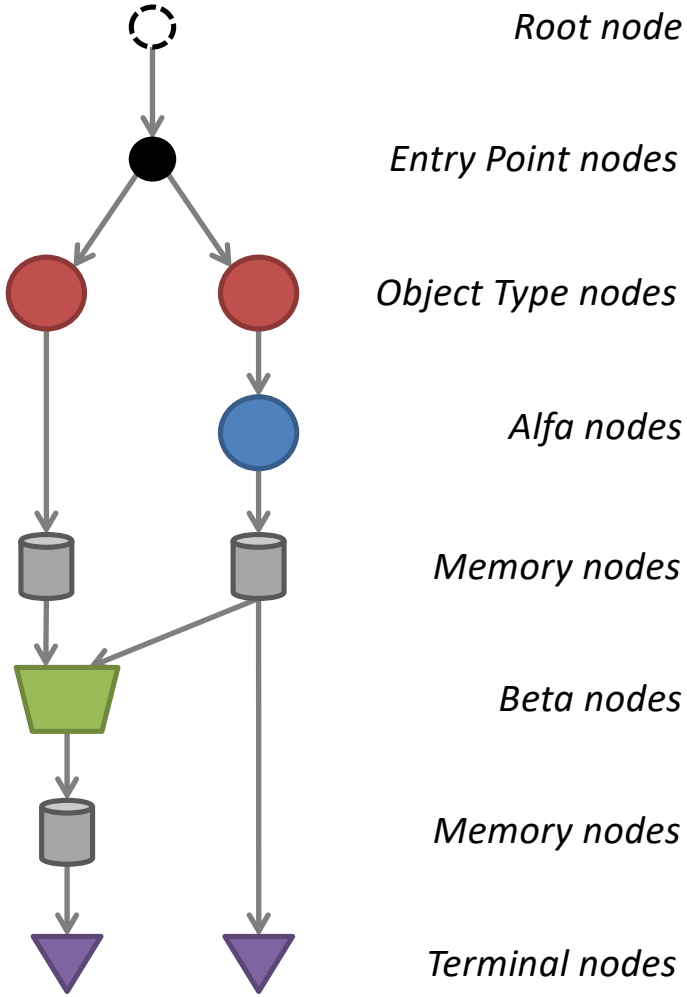
Beta nodes

Memory nodes

Terminal nodes

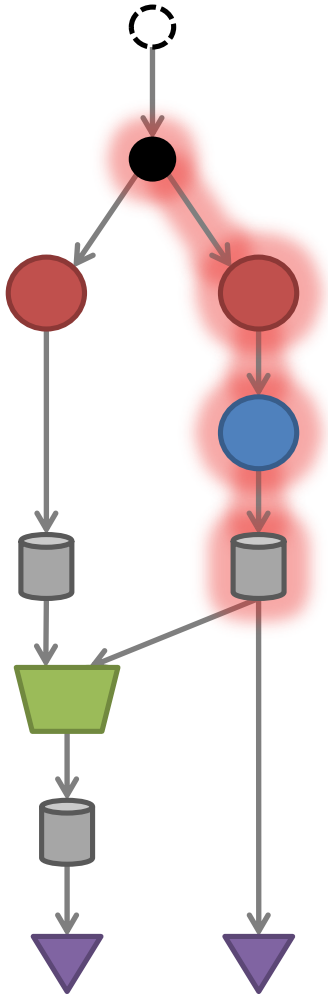
Pattern matching: RETE algorithm

```
rule "Find Francescos and addresses"
when
  $a: Address()
  $p: Person( name=="Francesco" )
then
  System.out.println($p+"/"+$a+" ");
end
```



Pattern matching: RETE algorithm

```
rule "Find Francescos and addresses"
when
  $a: Address()
  $p: Person( name=="Francesco" )
then
  System.out.println($p+"/"+$a+" ");
end
```



Root node

Entry Point nodes

Object Type nodes

Alfa nodes

Memory nodes

Beta nodes

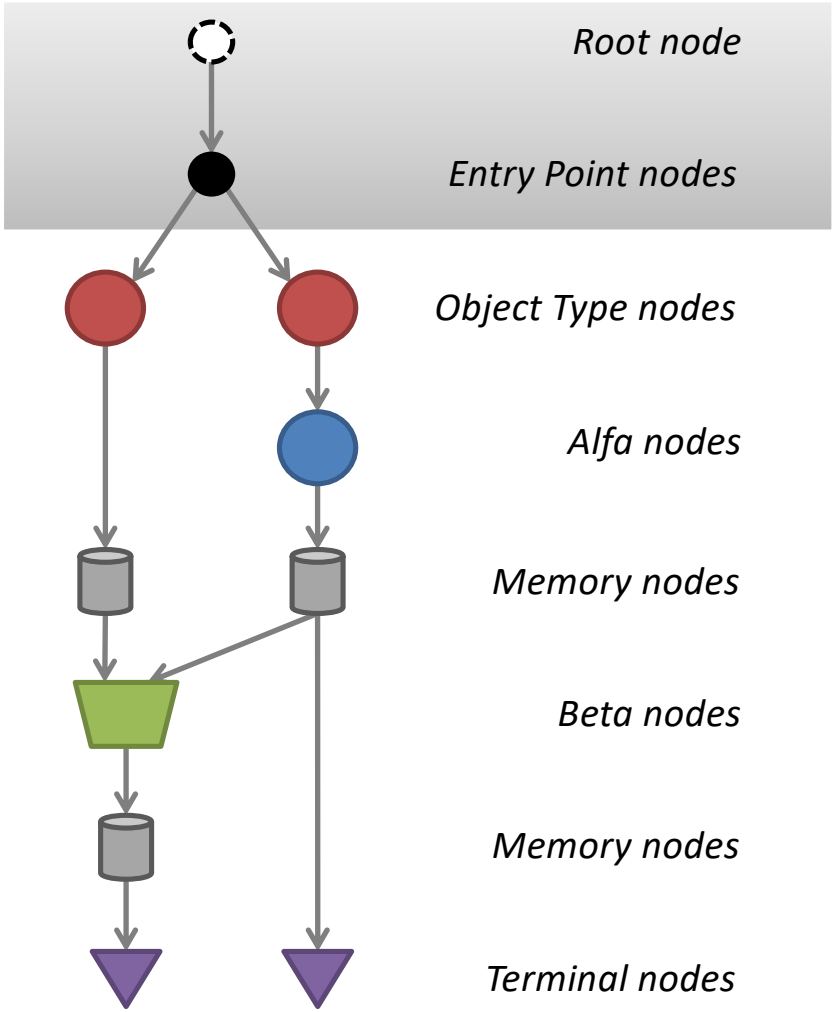
Memory nodes

Terminal nodes

NB: nodes of the previous rules are shared!

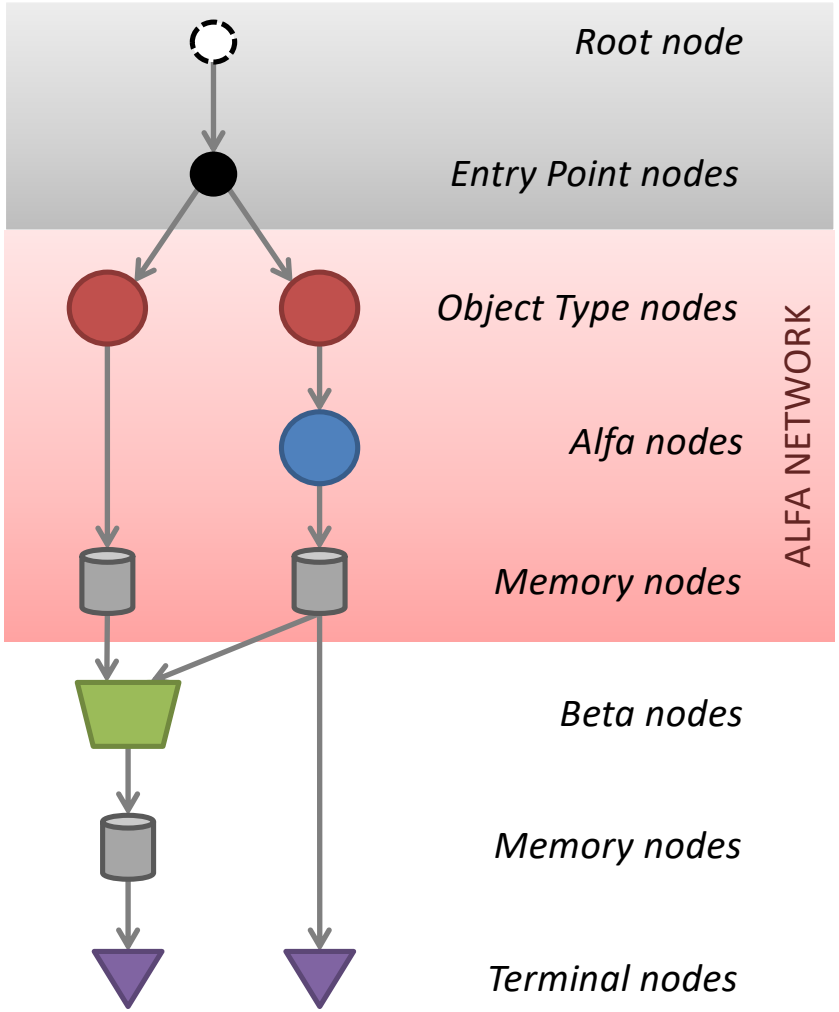
Pattern matching: RETE algorithm

```
rule "Find Francescos and addresses"
when
  $a: Address()
  $p: Person( name=="Francesco" )
then
  System.out.println($p+"/"+$a+" ");
end
```



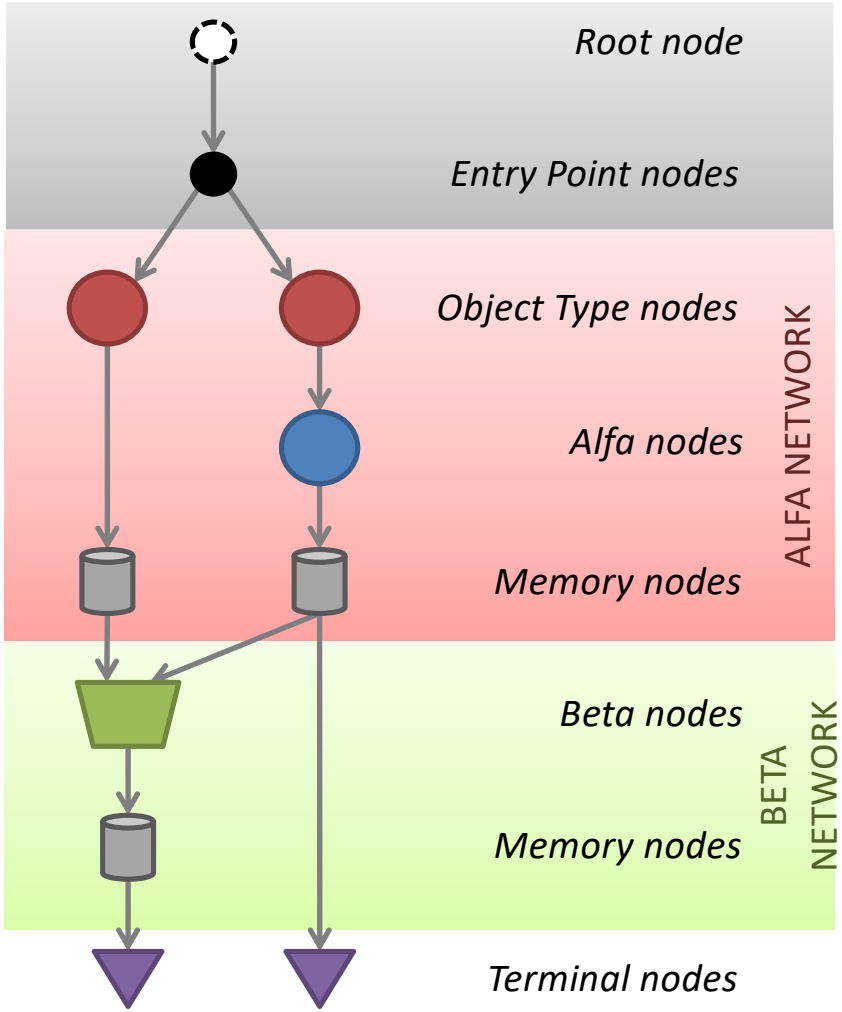
Pattern matching: RETE algorithm

```
rule "Find Francescos and addresses"
when
  $a: Address()
  $p: Person( name=="Francesco" )
then
  System.out.println($p+"/"+$a+" ");
end
```



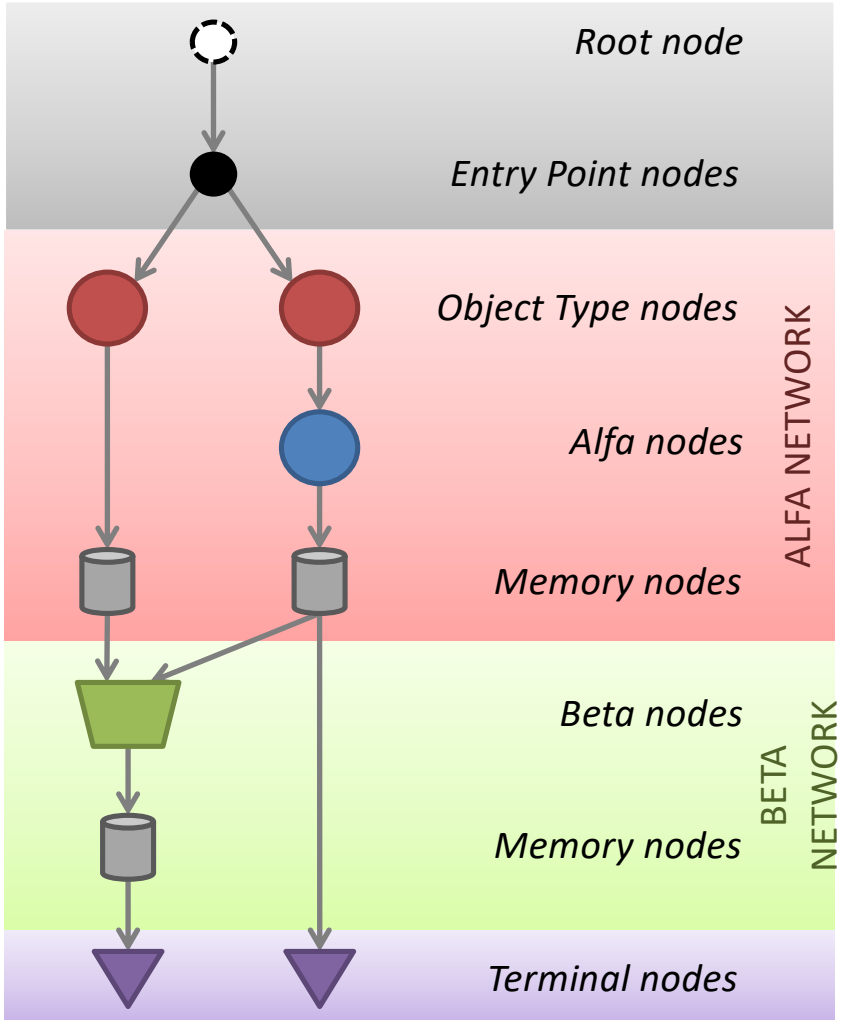
Pattern matching: RETE algorithm

```
rule "Find Francescos and addresses"
when
  $a: Address()
  $p: Person( name=="Francesco" )
then
  System.out.println($p+"/"+$a+" ");
end
```



Pattern matching: RETE algorithm

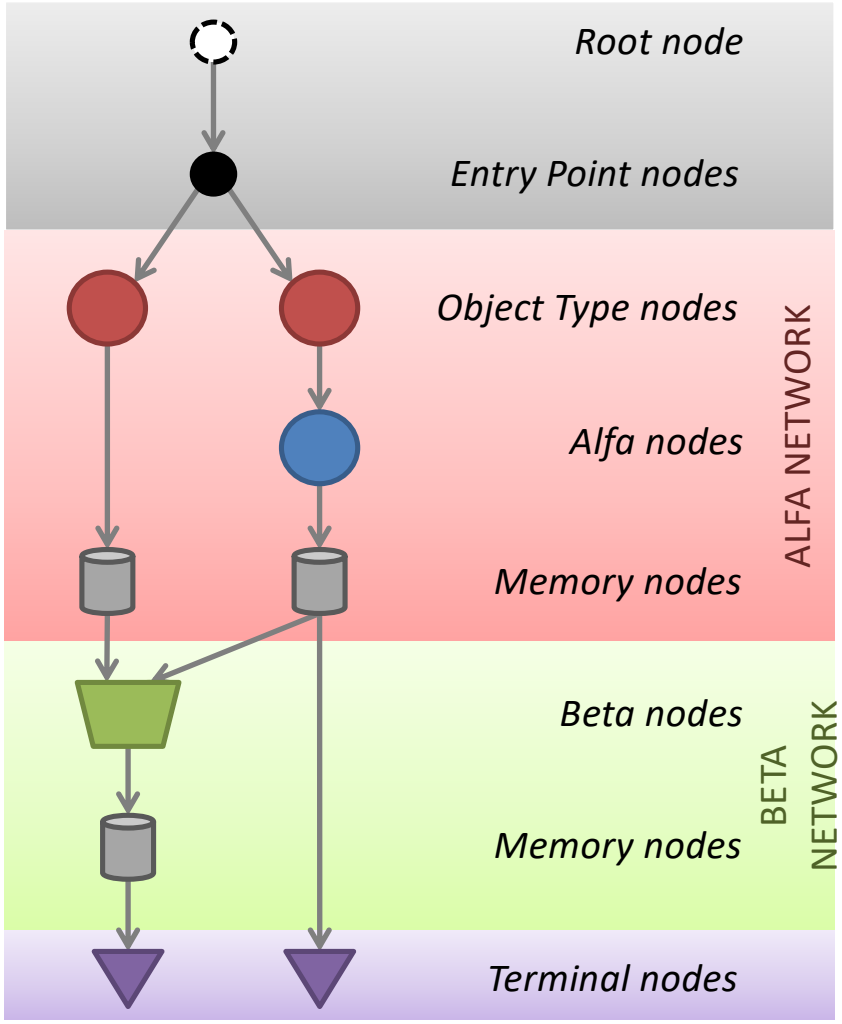
```
rule "Find Francescos and addresses"
when
  $a: Address()
  $p: Person( name=="Francesco" )
then
  System.out.println($p+"/"+$a+" ");
end
```



Pattern matching: RETE algorithm

```
rule "Find Francescos and addresses"
when
  $a: Address()
  $p: Person( name=="Francesco" )
then
  System.out.println($p+"/"+$a+" ");
end
```

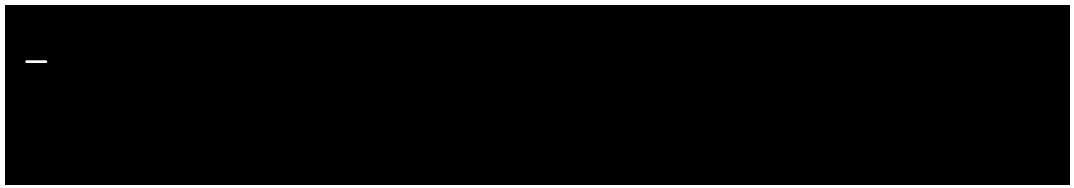

```
_
```

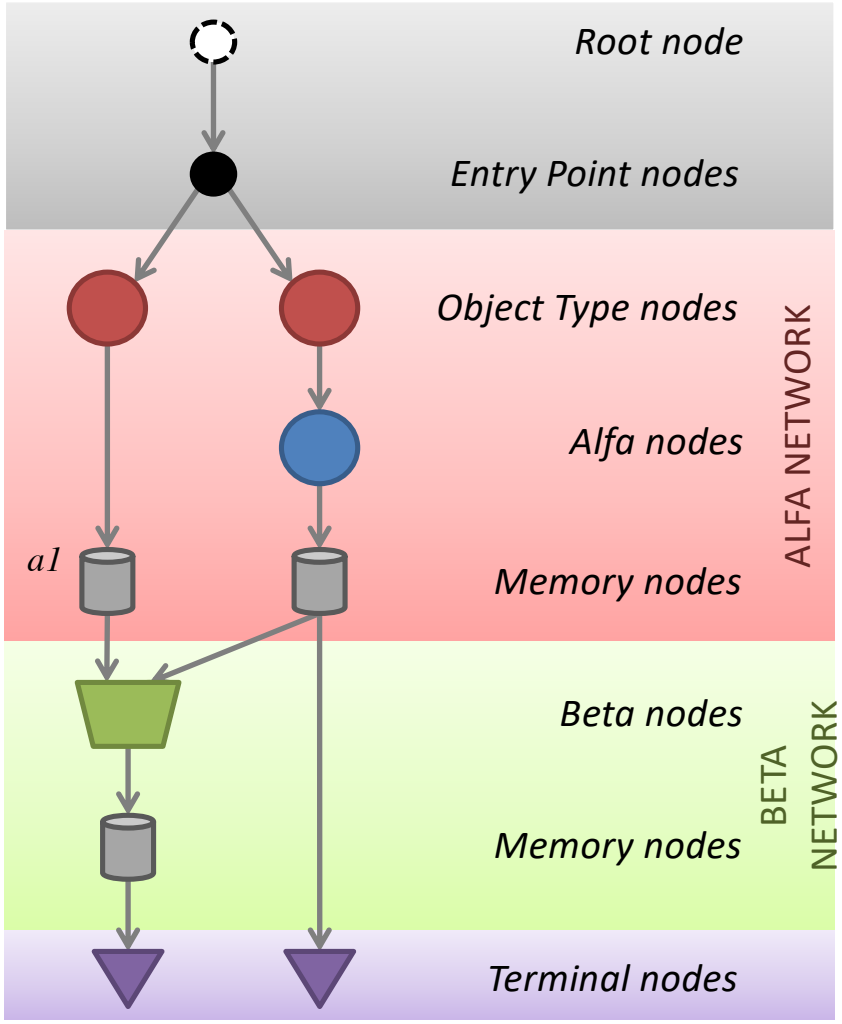


Pattern matching: RETE algorithm

```
rule "Find Francescos and addresses"
when
  $a: Address()
  $p: Person( name=="Francesco" )
then
  System.out.println($p+"/"+$a+" ");
end
```

a1: Address("Via Po 2", 40068, "San Lazzaro")





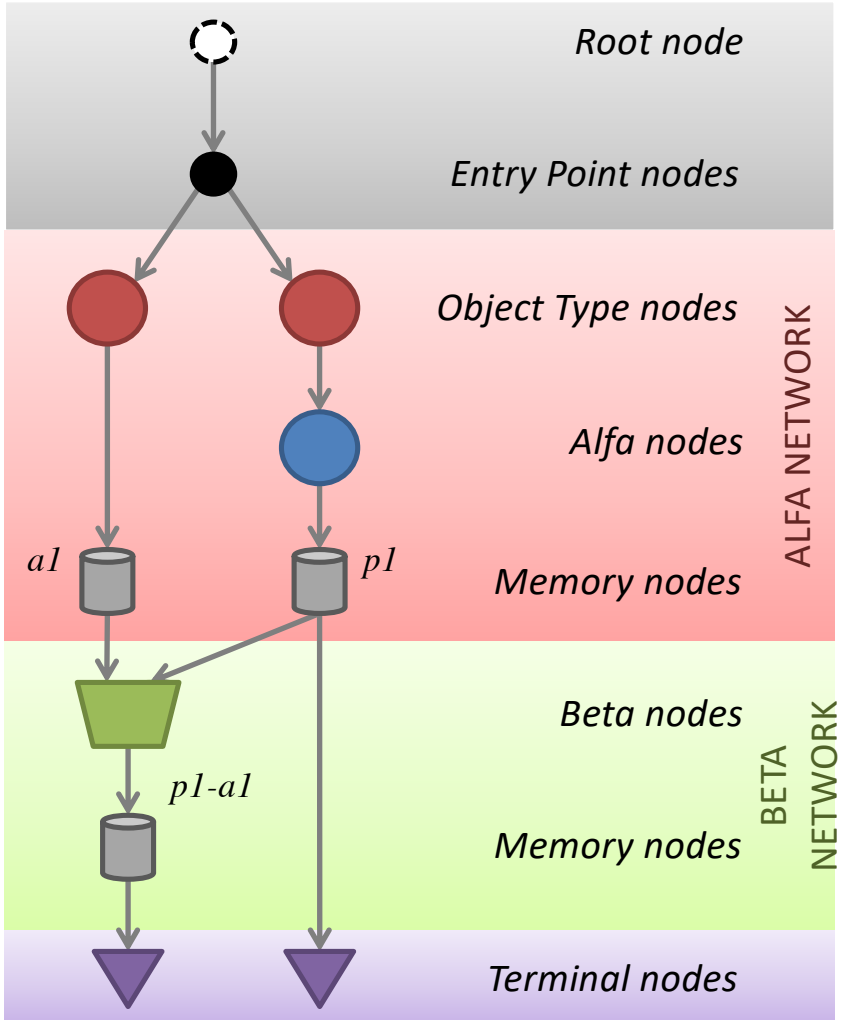
Pattern matching: RETE algorithm

```
rule "Find Francescos and addresses"
when
  $a: Address()
  $p: Person( name=="Francesco" )
then
  System.out.println($p+"/"+$a+" ");
end
```

a1: Address("Via Po 2", 40068, "San Lazzaro")

p1: Person("Francesco", null)


```
Person[p1, -]/Address[a1]  _
```

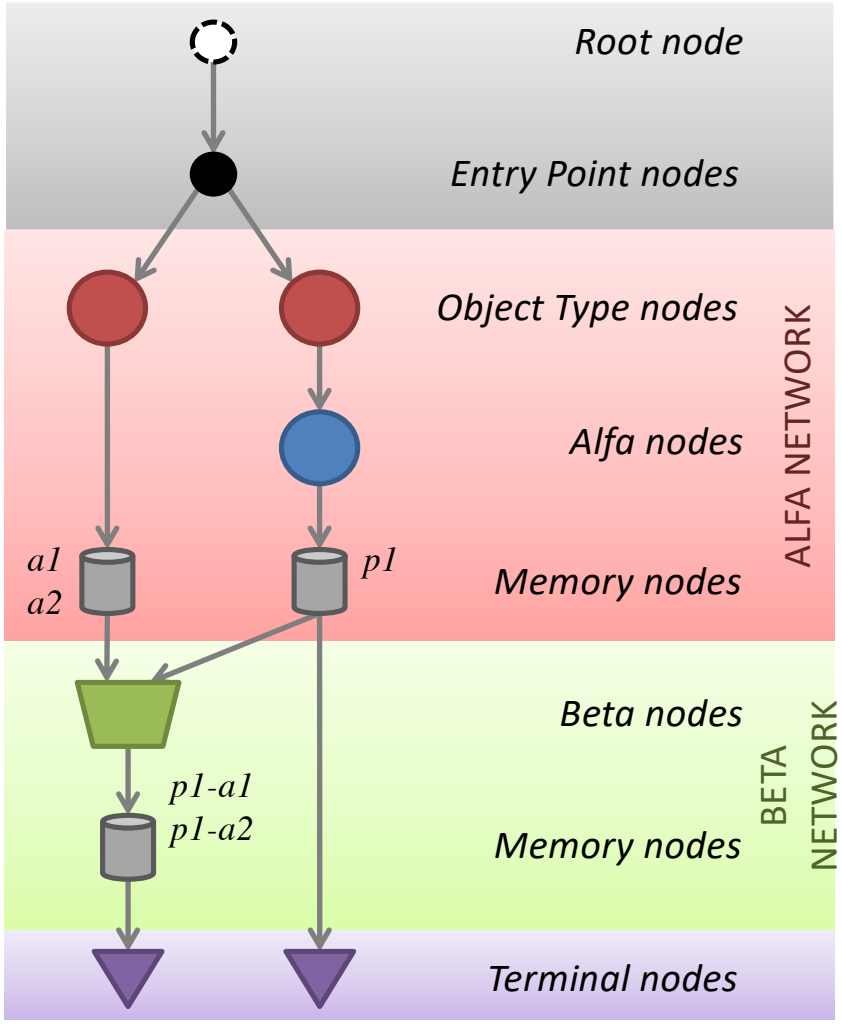


Pattern matching: RETE algorithm

```
rule "Find Francescos and addresses"
when
    $a: Address()
    $p: Person( name=="Francesco" )
then
    System.out.println($p+"/"+$a+" ");
end
```

a1: Address("Via Po 2", 40068, "San Lazzaro")
p1: Person("Francesco", null)
a2: Address("Via Roma 5", 40128, "Bologna")

```
Person[p1, -]/Address[a1]    Person[p1, -]/Address[a2]
_
```



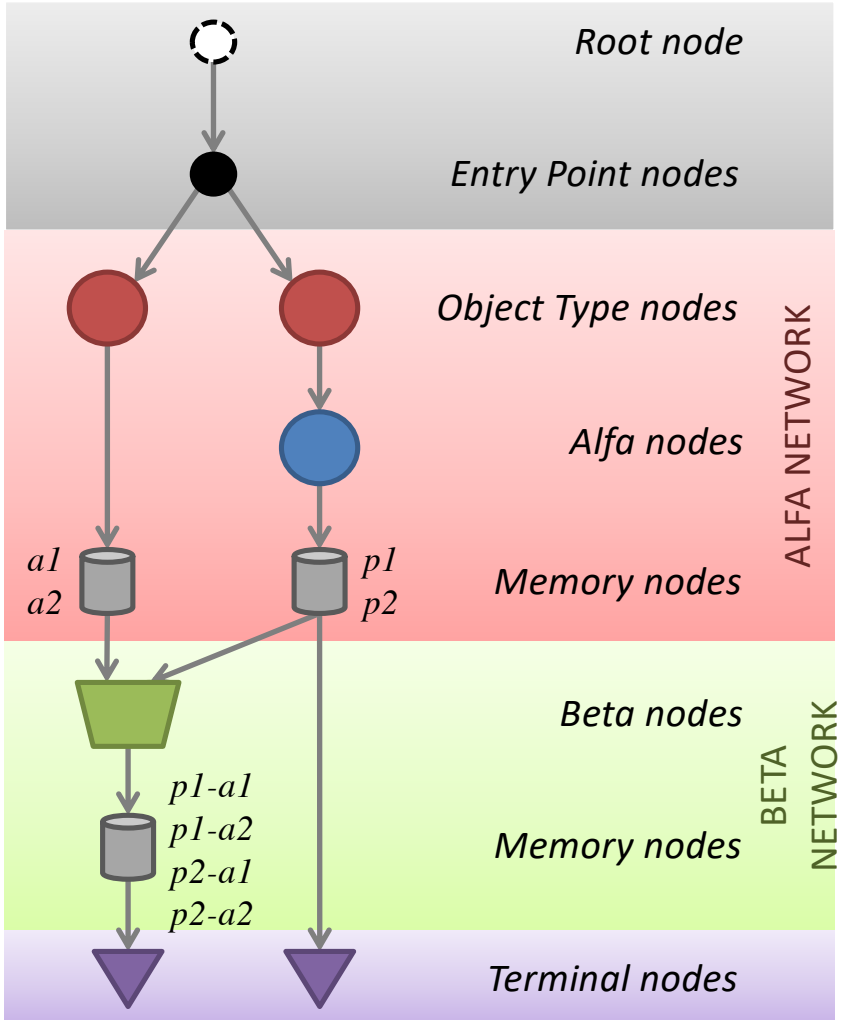
Pattern matching: RETE algorithm

```
rule "Find Francescos and addresses"
when
    $a: Address()
    $p: Person( name=="Francesco" )
then
    System.out.println($p+"/"+$a+" ");
end
```

WM

a1: Address("Via Po 2", 40068, "San Lazzaro")
p1: Person("Francesco", null)
a2: Address("Via Roma 5", 40128, "Bologna")
p2: Person("Francesco", a1)

```
Person[p1, -]/Address[a1]   Person[p1, -]/Address[a2]
Person[p2, -]/Address[a1]   Person[p2, a1]/Address[a2]
_
```



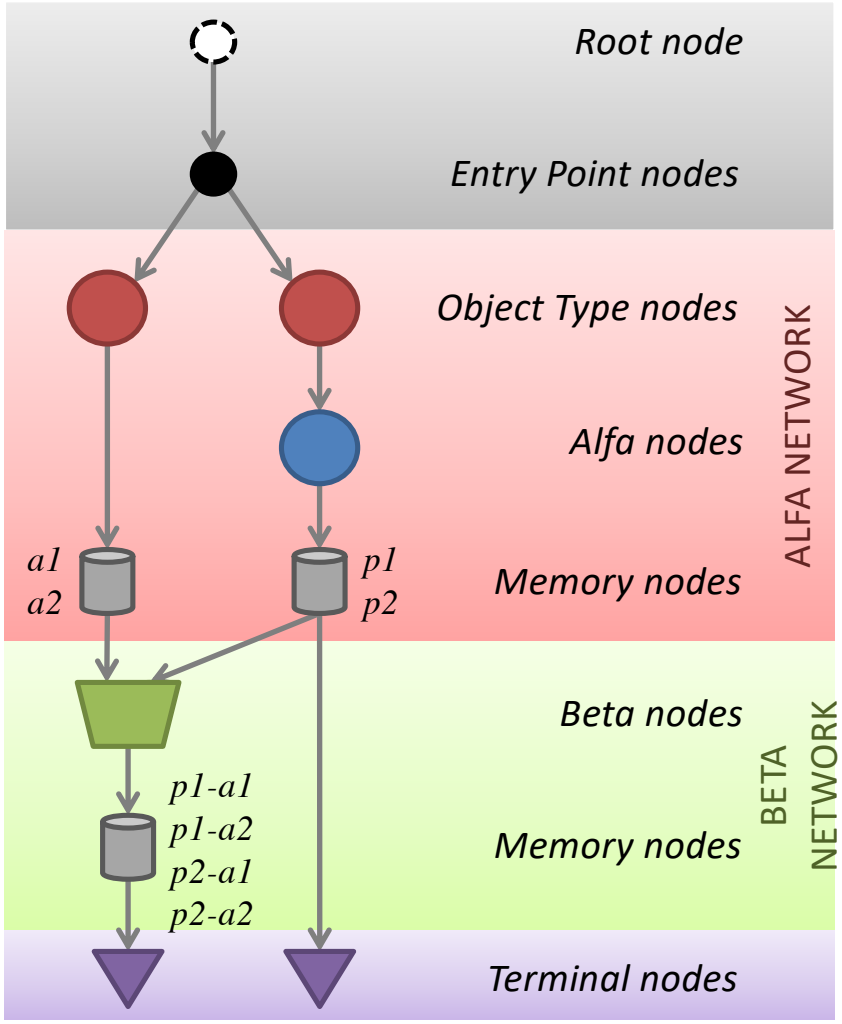
Pattern matching: RETE algorithm

```
rule "Find Francescos and addresses"
when
    $a: Address()
    $p: Person( name=="Francesco" )
then
  System.out.println($p+"/"+$a+" ");
end
```

WM

a1: Address("Via Po 2", 40068, "San Lazzaro")
p1: Person("Francesco", null)
a2: Address("Via Roma 5", 40128, "Bologna")
p2: Person("Francesco", a1)
p3: Person("Giacomo", a1)

```
Person[p1, -]/Address[a1]   Person[p1, -]/Address[a2]
Person[p2, -]/Address[a1]   Person[p2, a1]/Address[a2]
_
```

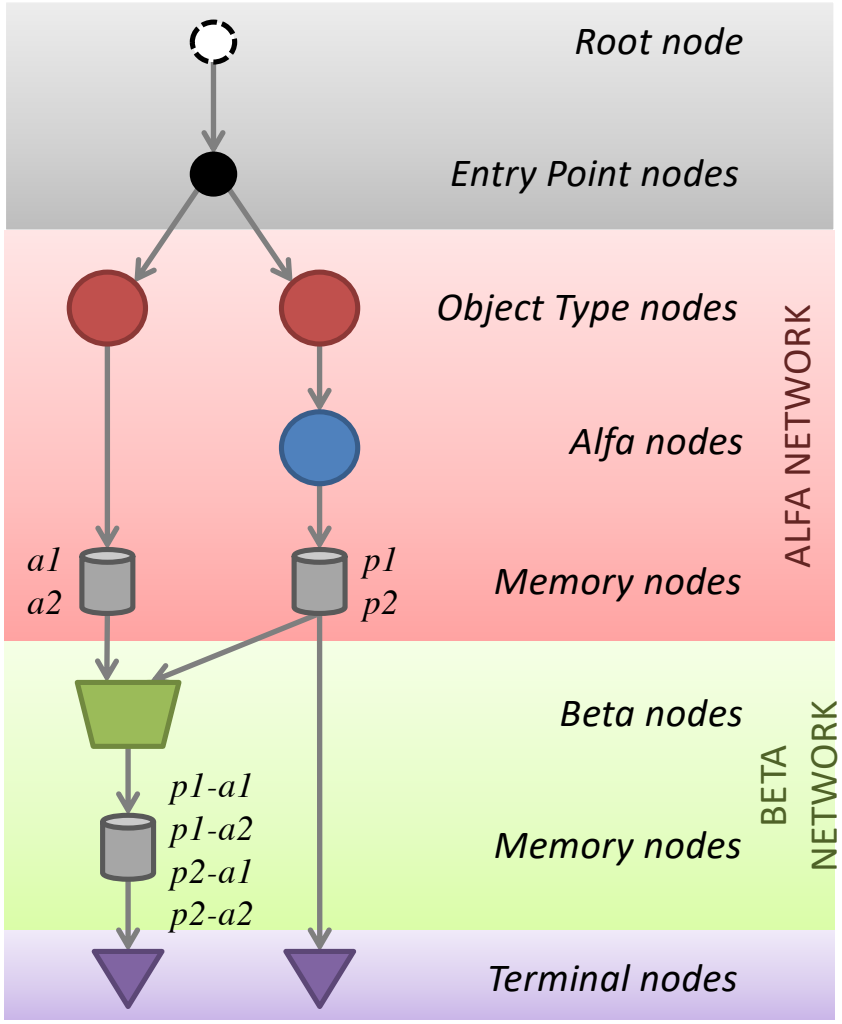


Pattern matching: RETE algorithm

```
rule "Find Francescos and addresses"
when
  $a: Address()
  $p: Person( name=="Francesco" )
then
  System.out.println($p+"/"+$a+" ");
end
```

a1: Address("Via Po 2", 40068, "San Lazzaro")
p1: Person("Francesco", null)
a2: Address("Via Roma 5", 40128, "Bologna")
p2: Person("Francesco", a1)
p3: Person("Giacomo", a1)

```
Person[p1, -]/Address[a1]   Person[p1, -]/Address[a2]
Person[p2, -]/Address[a1]   Person[p2, a1]/Address[a2]
_
```

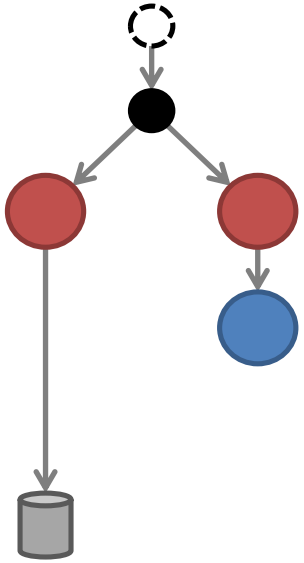


Pattern matching: RETE algorithm

EXAMPLE 3

Pattern matching: RETE algorithm

```
rule ”Find Francesco with its address"
when
  $a: Address()
  $p: Person( name=="Francesco",
              address == $a )
then
  System.out.println($p);
end
```



Root node

Entry Point nodes

Object Type nodes

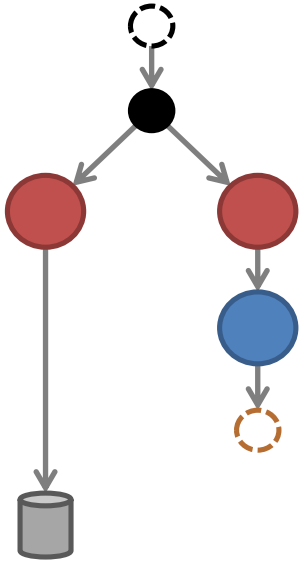
Alfa nodes

Dummy nodes

Memory nodes

Pattern matching: RETE algorithm

```
rule ”Find Francesco with its address"
when
  $a: Address()
  $p: Person( name=="Francesco",
              address == $a )
then
  System.out.println($p);
end
```



Root node

Entry Point nodes

Object Type nodes

Alfa nodes

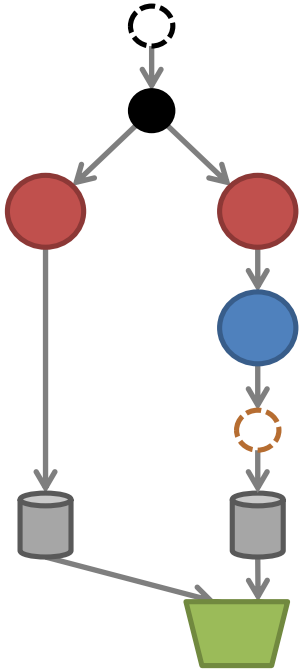
Dummy nodes

Memory nodes

NB: this Alfa node contains a cross reference that cannot be resolved.

Pattern matching: RETE algorithm

```
rule "Find Francesco with its address"
when
  $a: Address()
  $p: Person( name=="Francesco",
              address == $a )
then
  System.out.println($p);
end
```



Root node

Entry Point nodes

Object Type nodes

Alfa nodes

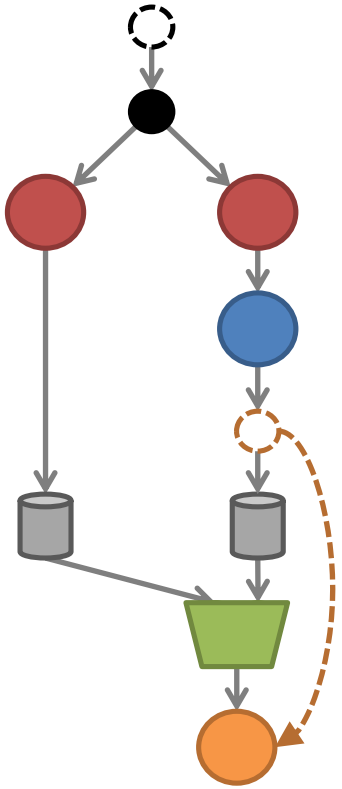
Dummy nodes

Memory nodes

Beta nodes

Pattern matching: RETE algorithm

```
rule "Find Francesco with its address"
when
  $a: Address()
  $p: Person( name=="Francesco",
              address == $a )
then
  System.out.println($p);
end
```



Root node

Entry Point nodes

Object Type nodes

Alfa nodes

Dummy nodes

Memory nodes

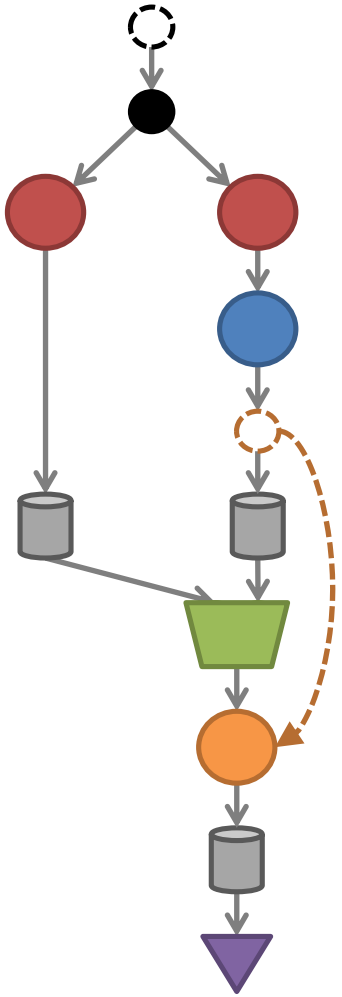
Beta nodes

Delayed nodes

NB: the previous Alfa node is inserted here because it can resolve the cross reference.

Pattern matching: RETE algorithm

```
rule "Find Francesco with its address"
when
    $a: Address()
    $p: Person( name=="Francesco",
                address == $a )
then
    System.out.println($p);
end
```



Root node

Entry Point nodes

Object Type nodes

Alfa nodes

Dummy nodes

Memory nodes

Beta nodes

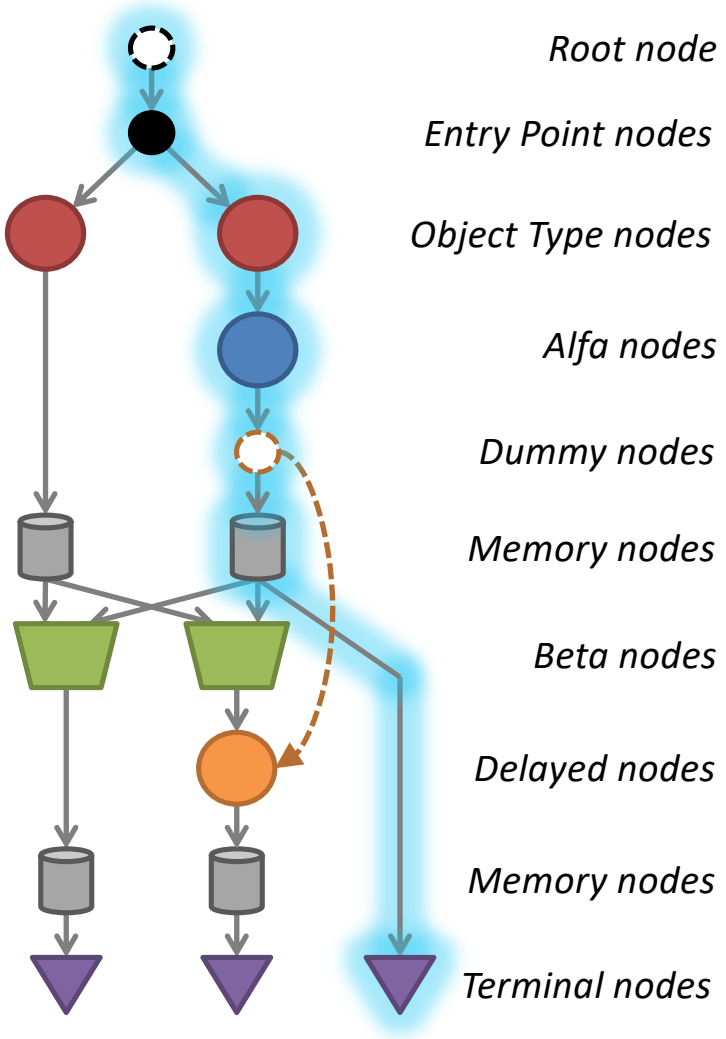
Delayed nodes

Memory nodes

Terminal nodes

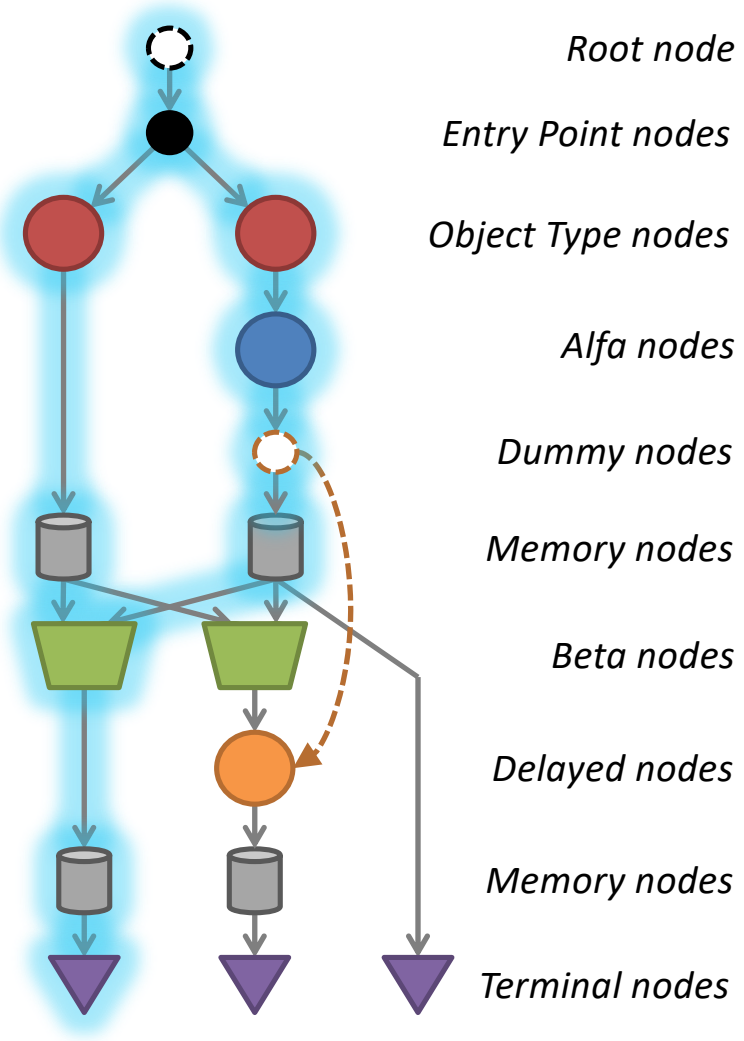
Pattern matching: RETE algorithm

```
rule "Find Francesco with its address"
when
    $a: Address()
    $p: Person( name=="Francesco",
                address == $a )
then
    System.out.println($p);
end
```



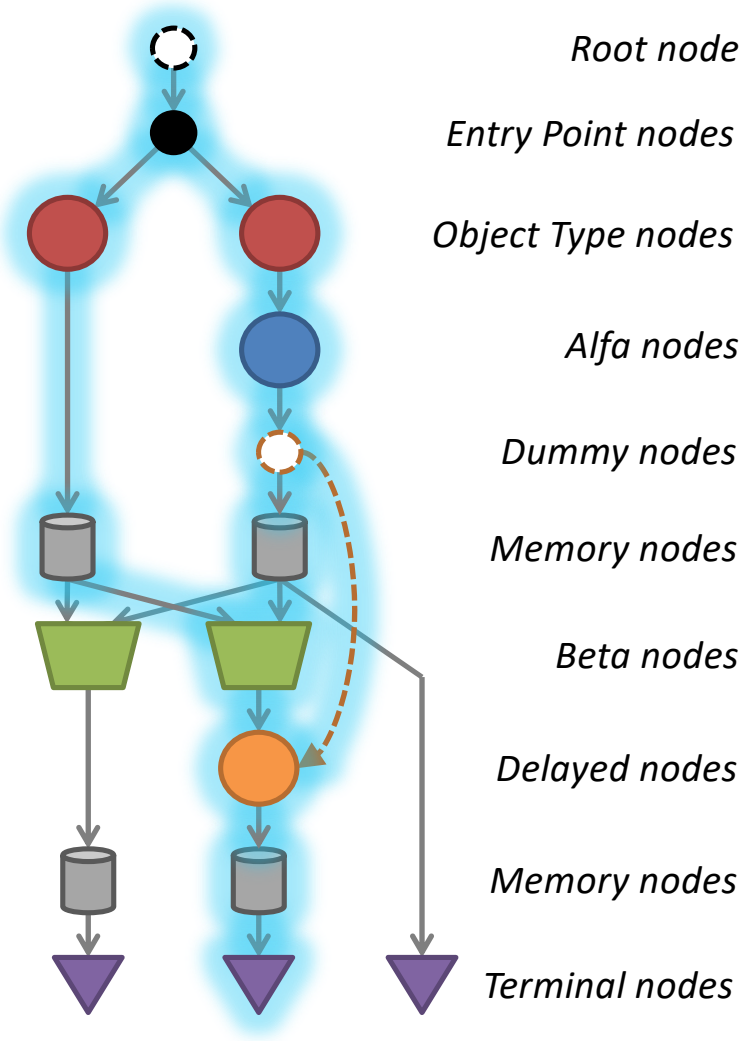
Pattern matching: RETE algorithm

```
rule "Find Francesco with its address"
when
    $a: Address()
    $p: Person( name=="Francesco",
                address == $a )
then
    System.out.println($p);
end
```



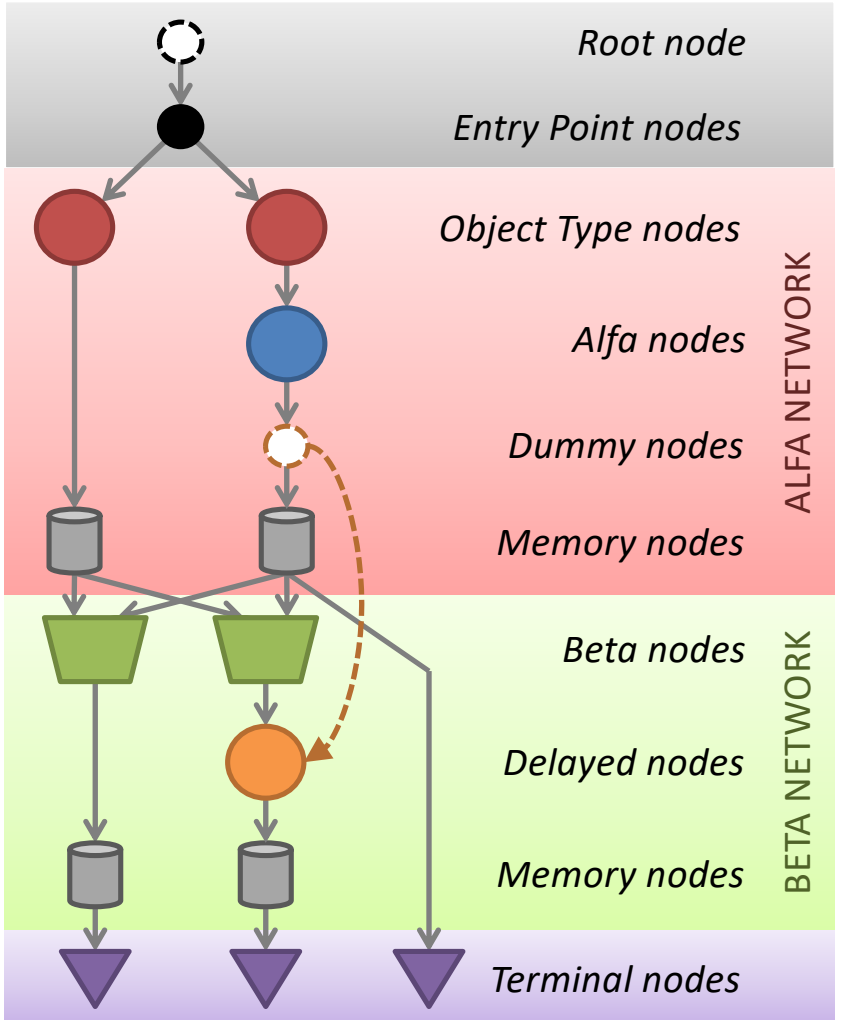
Pattern matching: RETE algorithm

```
rule "Find Francesco with its address"
when
    $a: Address()
    $p: Person( name=="Francesco",
                address == $a )
then
    System.out.println($p);
end
```



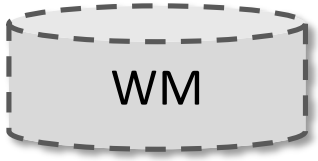
Pattern matching: RETE algorithm

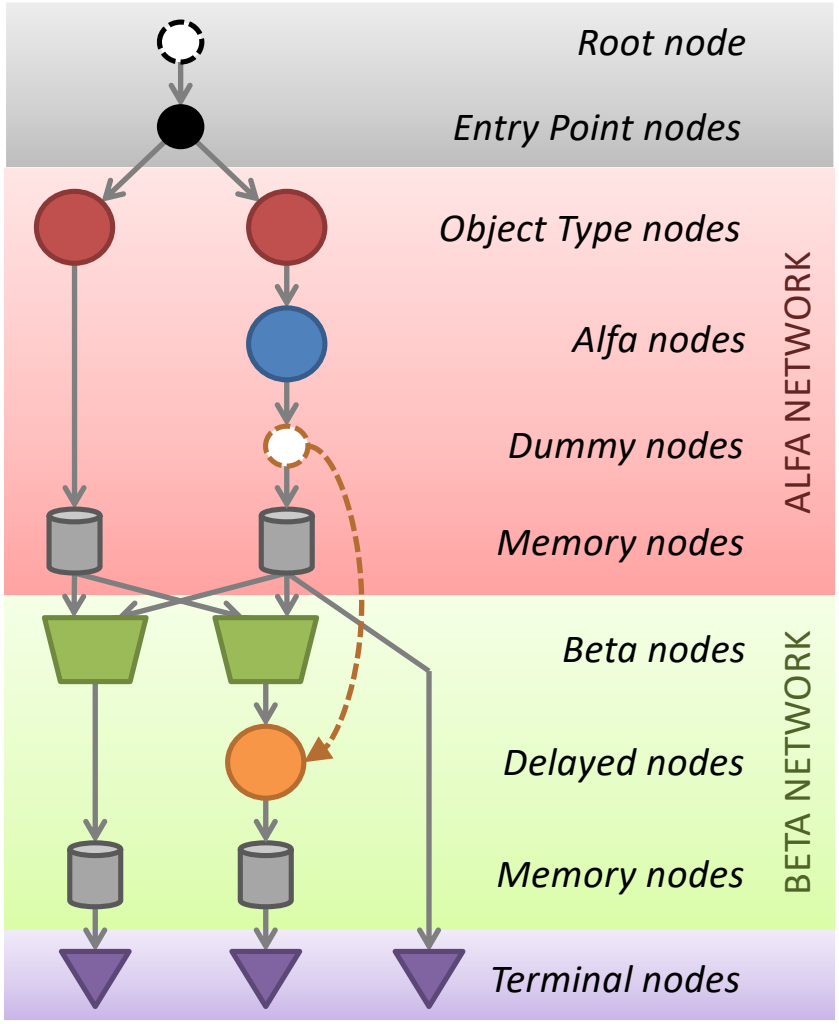
```
rule "Find Francesco with its address"
when
    $a: Address()
    $p: Person( name=="Francesco",
                address == $a )
then
  System.out.println($p);
end
```



Pattern matching: RETE algorithm

```
rule "Find Francesco with its address"
when
    $a: Address()
    $p: Person( name=="Francesco",
                address == $a )
then
  System.out.println($p);
end
```

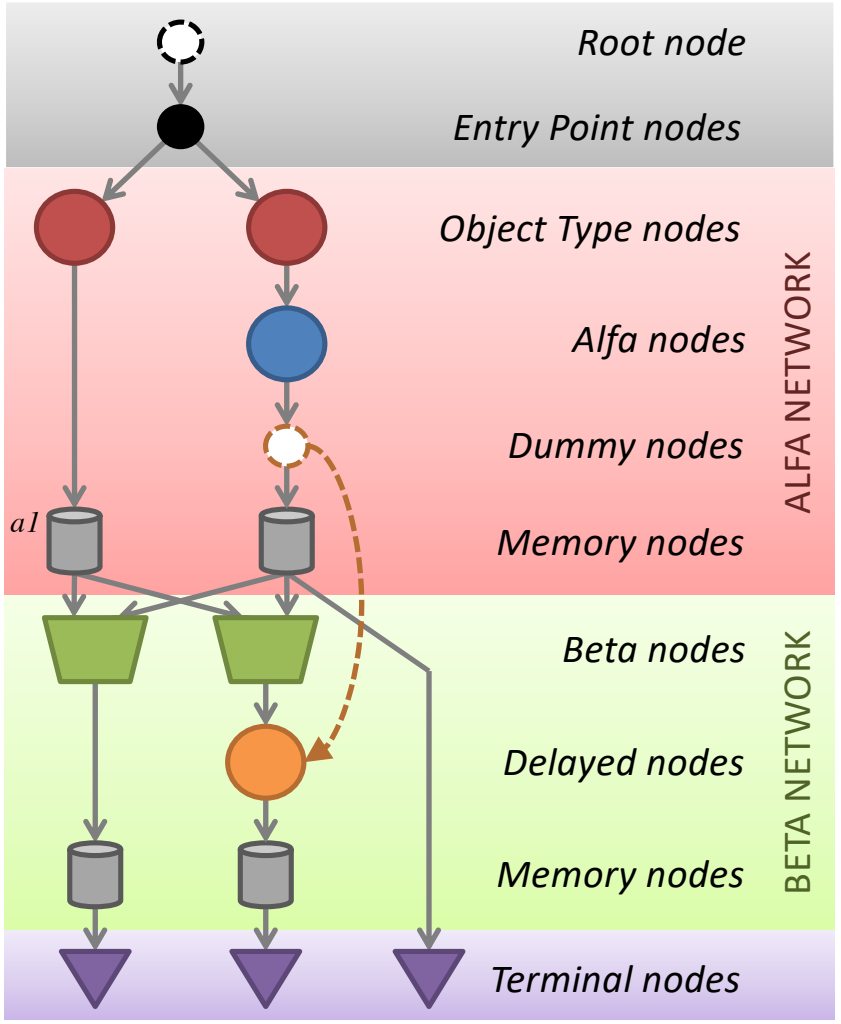




Pattern matching: RETE algorithm

```
rule "Find Francesco with its address"
when
    $a: Address()
    $p: Person( name=="Francesco",
                address == $a )
then
    System.out.println($p);
end
```

a1: Address("Via Po 2", 40068, "San Lazzaro")

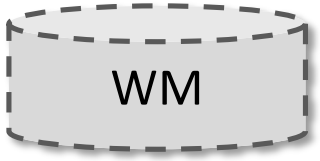


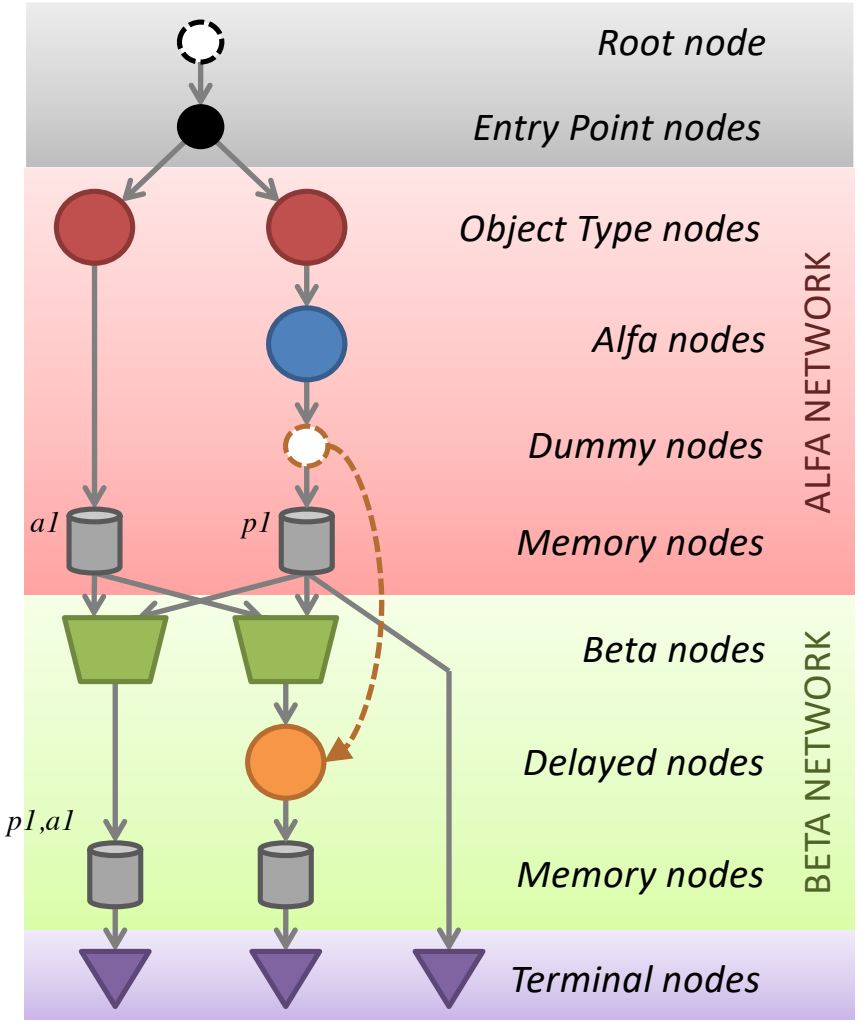
Pattern matching: RETE algorithm

```
rule "Find Francesco with its address"
when
    $a: Address()
    $p: Person( name=="Francesco",
                address == $a )
then
    System.out.println($p);
end
```

a1: Address("Via Po 2", 40068, "San Lazzaro")

p1: Person("Francesco", null)

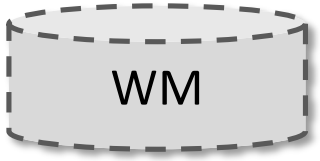


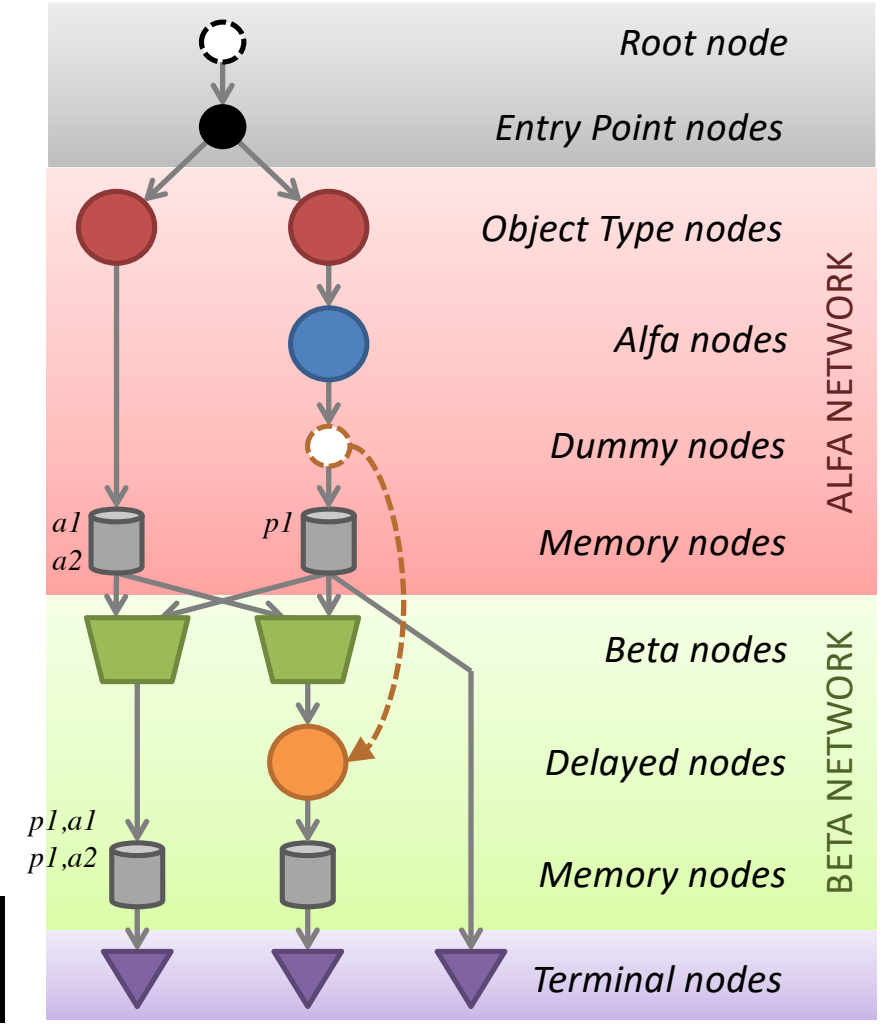


Pattern matching: RETE algorithm

```
rule "Find Francesco with its address"
when
    $a: Address()
    $p: Person( name=="Francesco",
                address == $a )
then
    System.out.println($p);
end
```

a1: Address("Via Po 2", 40068, "San Lazzaro")
p1: Person("Francesco", null)
a2: Address("Via Roma 5", 40128, "Bologna")

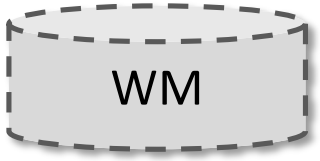




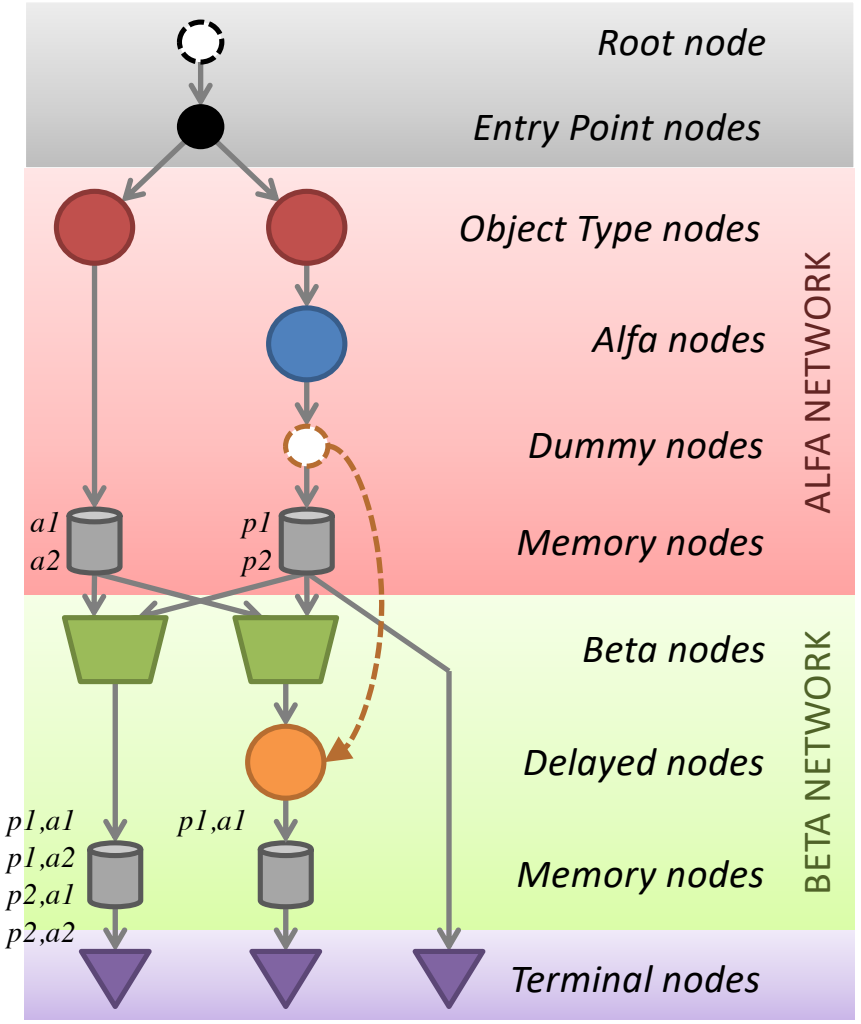
Pattern matching: RETE algorithm

```
rule "Find Francesco with its address"
when
    $a: Address()
    $p: Person( name=="Francesco",
                address == $a )
then
    System.out.println($p);
end
```

a1: Address("Via Po 2", 40068, "San Lazzaro")
p1: Person("Francesco", null)
a2: Address("Via Roma 5", 40128, "Bologna")
p2: Person("Francesco", a1)

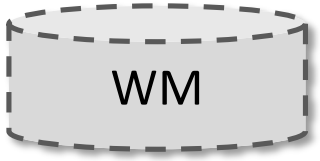



```
Person[Francesco, Address[Via Po 2, 40068, San Lazzaro]]
_
```



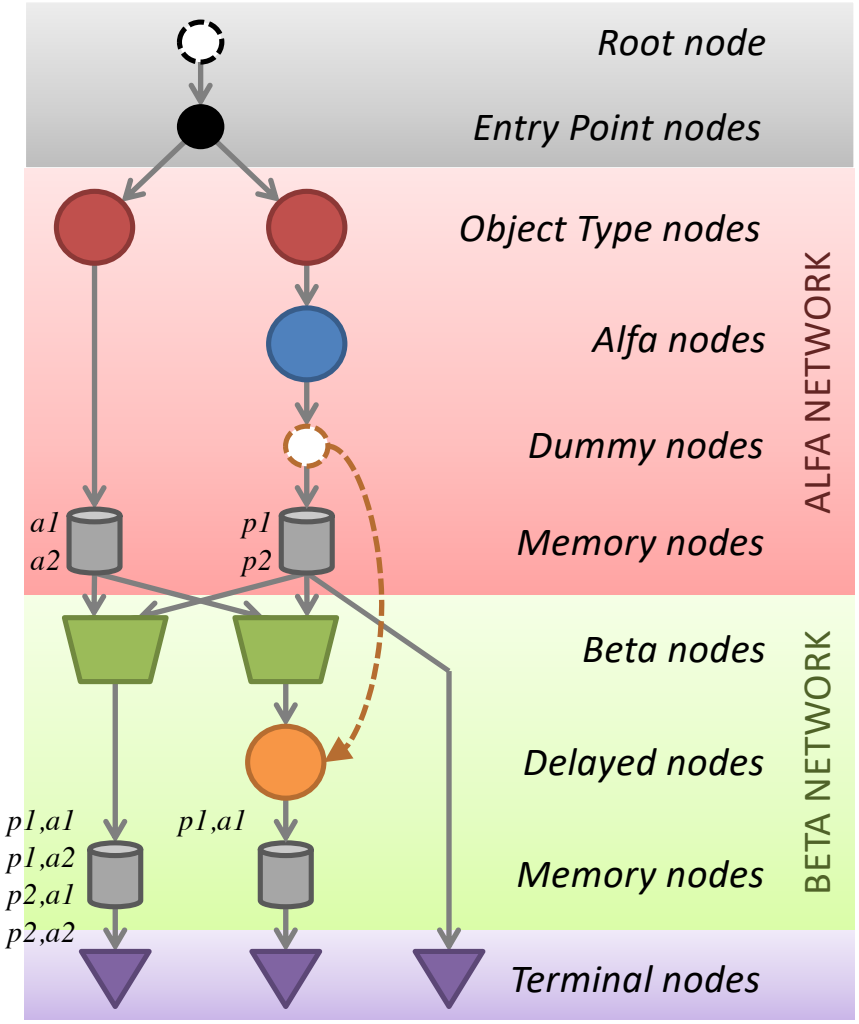
Pattern matching: RETE algorithm

```
rule "Find Francesco with its address"
when
    $a: Address()
    $p: Person( name=="Francesco",
                address == $a )
then
    System.out.println($p);
end
```



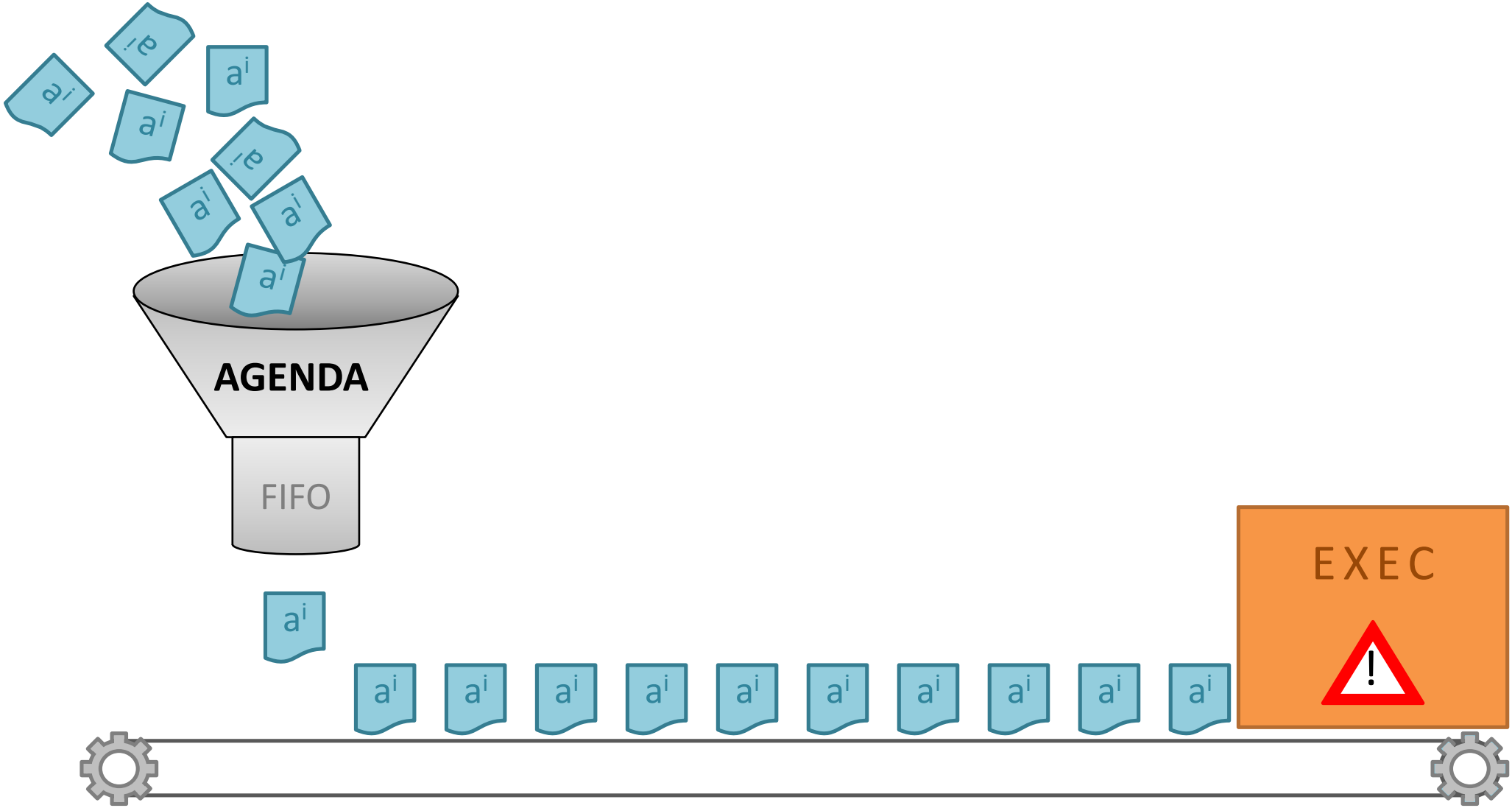
a1: Address("Via Po 2", 40068, "San Lazzaro")
p1: Person("Francesco", null)
a2: Address("Via Roma 5", 40128, "Bologna")
p2: Person("Francesco", a1)
p3: Person("Giacomo", a1)

```
Person[Francesco, Address[Via Po 2, 40068, San Lazzaro]]
_
```

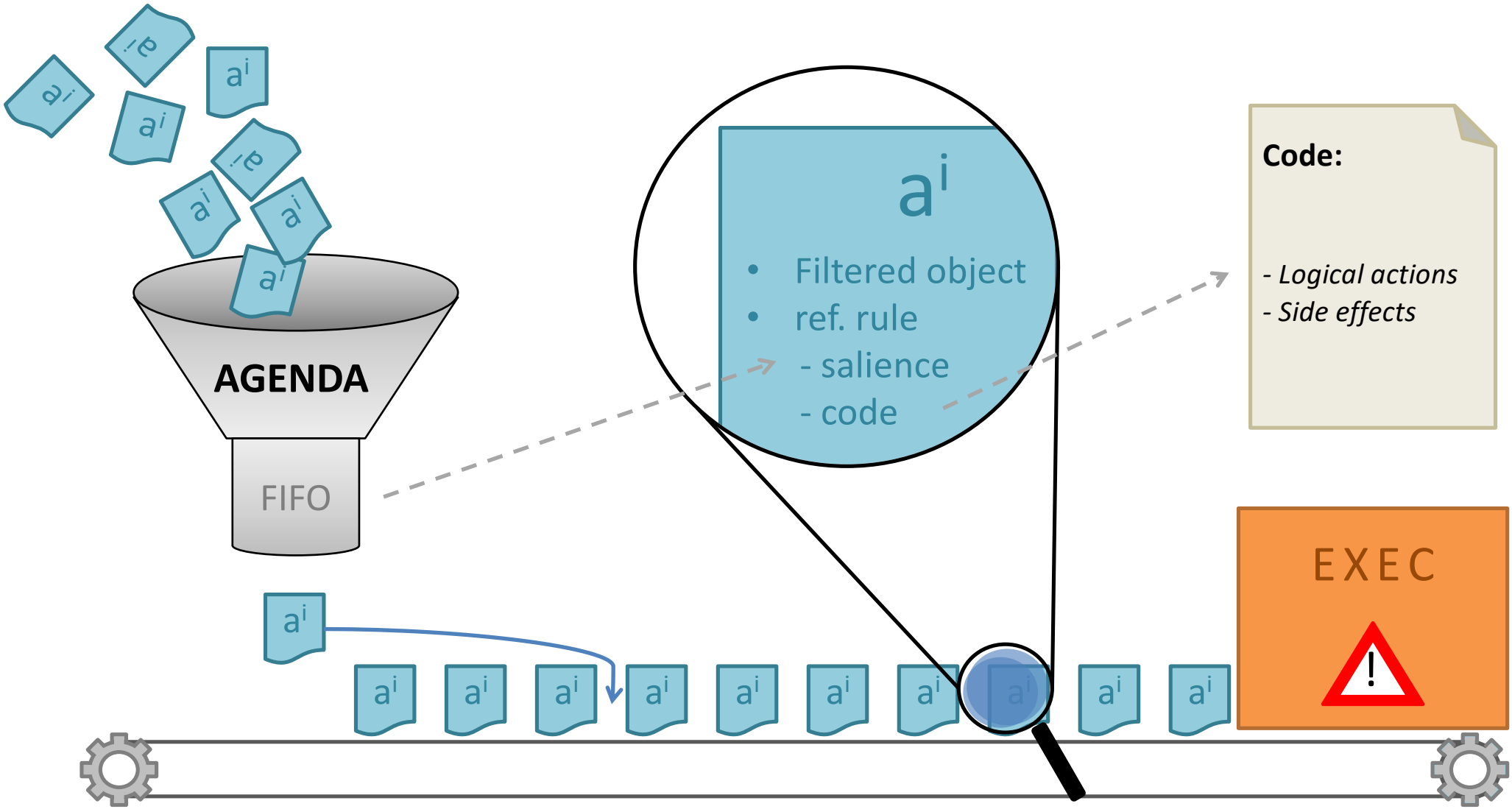


CONFLICT RESOLUTION AND EXECUTION

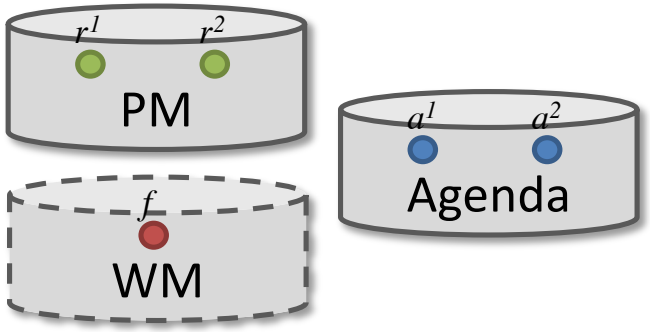
Conflict resolution & Execution



Conflict resolution & Execution



Conflict resolution & Execution

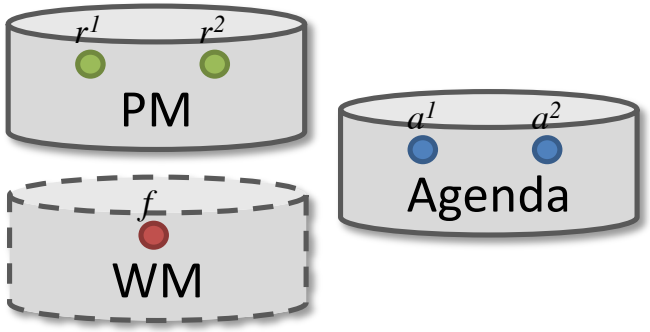



```
rule "r1"
when
  F()
then
  assert(new G());
end
```

```
rule "r2"
when
  $f: F()
then
  retract($f);
end
```

?!?

Conflict resolution & Execution

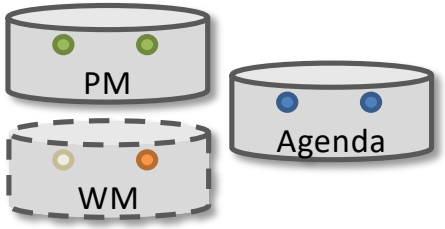



```
rule "r1"
when
  F()
then
  assert(new G());
end
```

```
rule "r2"
when
  $f: F()
then
  retract($f);
end
```

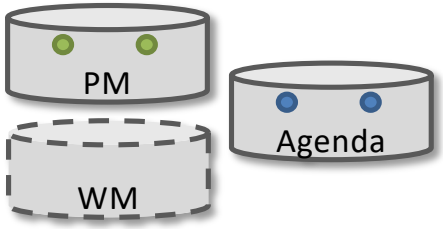
?!?

$a^1 < a^2$



First insert G,
then retract F.

$a^2 < a^1$



First retract F,
a1 cannot be applied,
G never inserted.

$r^1 < r^2$

```
rule "r1"
salience 10
...
```

```
rule "r2"
salience 5
...
```

Establish a precedence
fixed order between
r1 and r2.

References

- **Charles L. Forgy,** *"RETE: A Fast Algorithm for the Many Patter/Many Object Match Problem"*, Artificial Intelligence, 19, pp. 17-37, 1982
- **R.B. Doorenbos,** *"Production Matching for Large Learning Systems"*, Ph.D. Thesis, 1995
- **Schmit, Struhmer and Stojanovic,** *"Blending Complex Event Processing with the RETE algorithm"*, in Proceedings of iCEP2008, 2008
- http://en.wikipedia.org/wiki/Rete_algorithm
- http://en.wikipedia.org/wiki/Complex_event_processing

ADDING MORE CONSTRAINTS IN FACTS

In

- This allows to check if an attribute is in a list of values

```
rule ”The cashFlow can be a  credit or a debit "
when

  $cash : CashFlow(type in ( CashFlow.DEBIT,CashFlow.CREDIT) )

then
  System.out.println("The cashFlow is a credit or a debit");
end
```

Nested Accessor

- This allows to add a constraint to an attribute class without the need to add the linked object to the session

```
rule "Accessor"
when

  $cash : PrivateAccount( owner.name =="Héron" )

then
  System.out.println("Account is owned by Héron");
end
```

And/or

- It is possible to do constraints on attribute like in java.

```
rule "infixAnd"
when
  ( $c1 : Customer ( country=="GB") and  PrivateAccount(  owner==$c1))
   or
  ( $c1 : Customer (country=="US") and PrivateAccount(  owner==$c1))

then
  System.out.println("Person lives in GB or US");
end
```

Not

- This allows to test if no fact of a type is in the session.

```
rule "no customer"
when
  not Customer(  )

then
  System.out.println("No customer");
end
```

Exists

- On the contrary of previous syntax, this allows to test if there at least one fact type is in the session

```
rule "Exists"
when
  exists Customer(  )

then
  System.out.println("Account exists");
end
```

ForAll

- We would like to verify that every cashflow instance is linked to an Account instance

```
rule "ForAll"
when
forall ( Account( $no : accountNo  )
         CashFlow( accountNo  == $no) )
then
  System.out.println("All cashflows are related to an Account");
end
```