Sistemi a Regole di Produzione

Stefano Bragaglia

Fondamenti di Intelligenza Artificiale M

4 Giugno 2013

Sommario

1. Sistemi a regole
2. Sistemi a regole di produzione
   - *JBoss Drools*
   - *Un caso d'uso*
3. Pattern matching: l'algoritmo RETE
4. Risoluzione dei conflitti & Esecuzione
5. Riferimenti
6. Informazioni

Fondamenti di Intelligenza Artificiale M

SISTEMI A REGOLE

Sistemi a Regole

- Le regole sono il **principale modo** di esprimere la **conoscenza** in molti campi dell'I.A.
- I tipi di regole più comuni sono:
  - i *programmi logici* (es.: Prolog)
  - le *regole di produzione* (es.: Drools)
- Entrambi i tipi sono **molto comuni**, basate su **principi simili** ma realizzate **in modo duale**

Sistemi a Regole

- Il **Modus Ponens**, anche detto *Principio di disgiunzione*, *Affermazione dell'antecedente* o *Ragionamento diretto*, prevede che:

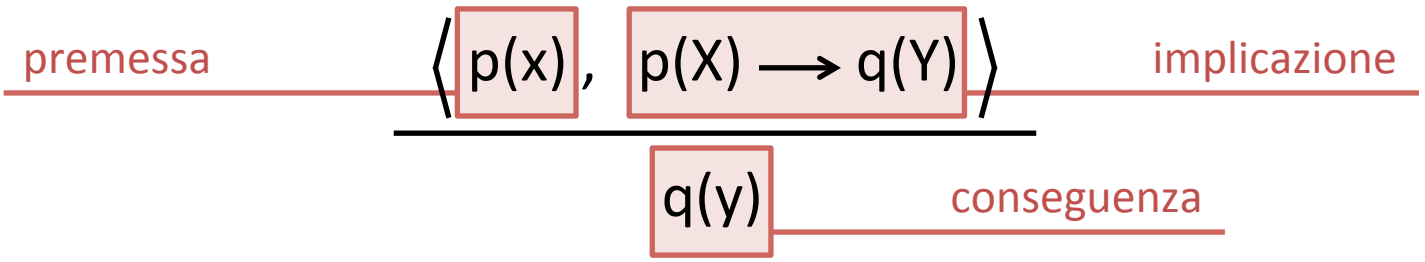
$$\frac{\langle\, p(x)\,,\quad p(X) \longrightarrow q(Y) \,\rangle}{q(y)}$$

  se è vero che p(X) *implica* q(Y) *e* p(x) *è vero, allora* q(y) *è ugualmente vero*

- **Es.:** Se piove, allora la strada si bagna.
  Qui piove.
  Dunque questa strada è bagnata.

Sistemi a Regole

- Il **Modus Ponens**, anche detto *Principio di disgiunzione*, *Affermazione dell'antecedente* o *Ragionamento diretto*, prevede che:



se è vero che p(X) *implica* q(Y) *e* p(x) *è vero, allora* q(y) *è ugualmente vero*

- **Es.:** Se piove, allora la strada si bagna. — implicazione
  Qui piove. — premessa
  Dunque questa strada è bagnata. — conseguenza

Sistemi a Regole

Programmi logici

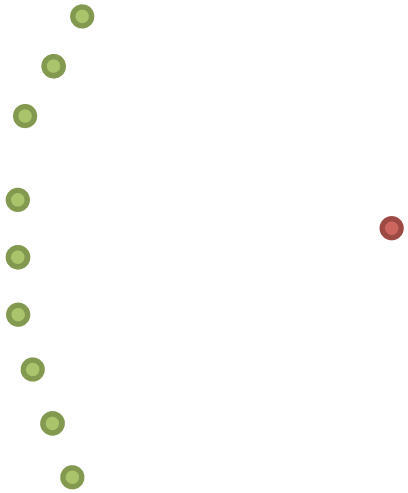
- *Backward-chaining*
- Dal goal ai fatti, applicando le regole all'indietro
- Generalmente conservativi
- Unificazione
- Backtracking

Regole di produzione

- *Forward-chaining*
- I fatti attivano le regole che generano nuovi fatti
- Potenzialmente distruttive
- Pattern matching
- Parallelismo

Sistemi a Regole

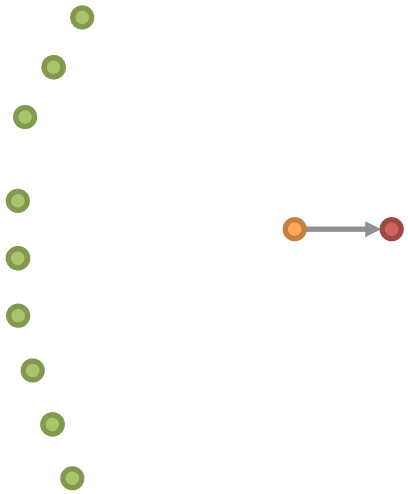
Programmi logici



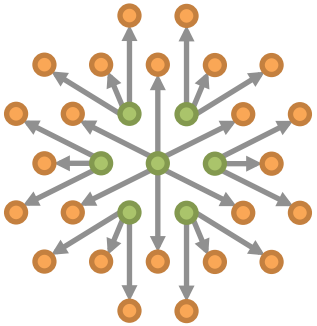
Regole di produzione

Sistemi a Regole

Programmi logici

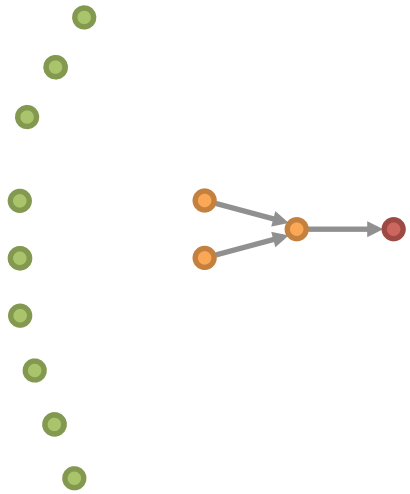


Regole di produzione

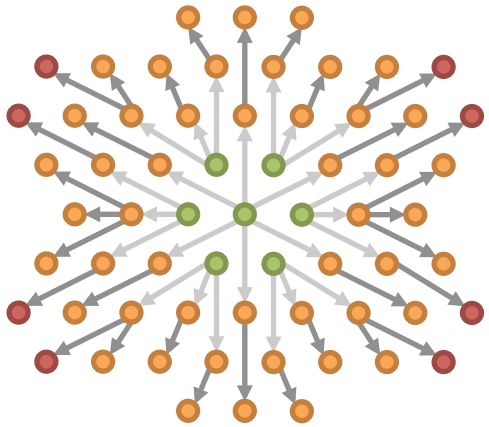


Sistemi a Regole

Programmi logici

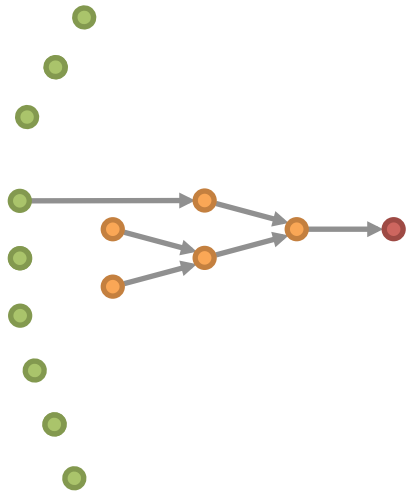


Regole di produzione

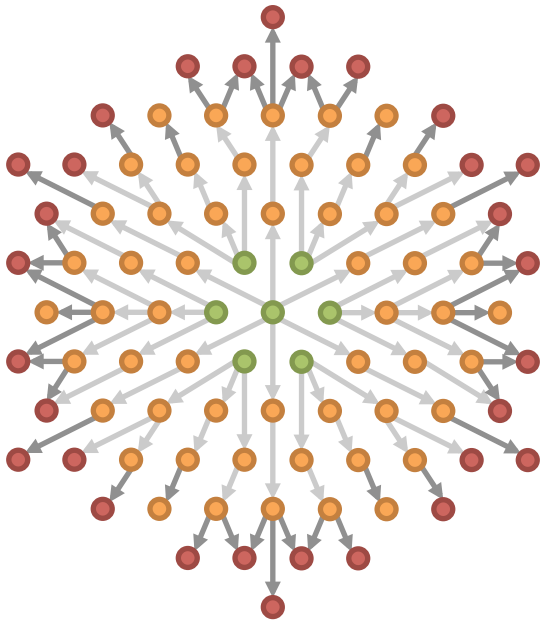


Sistemi a Regole

Programmi logici

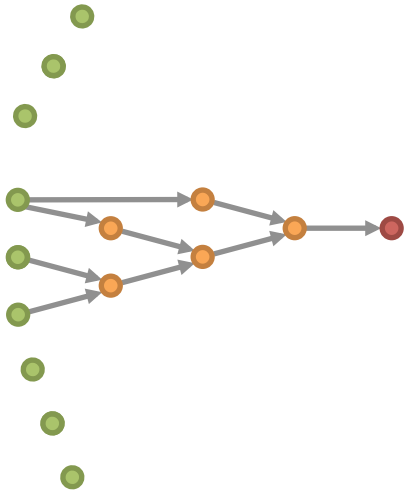


Regole di produzione

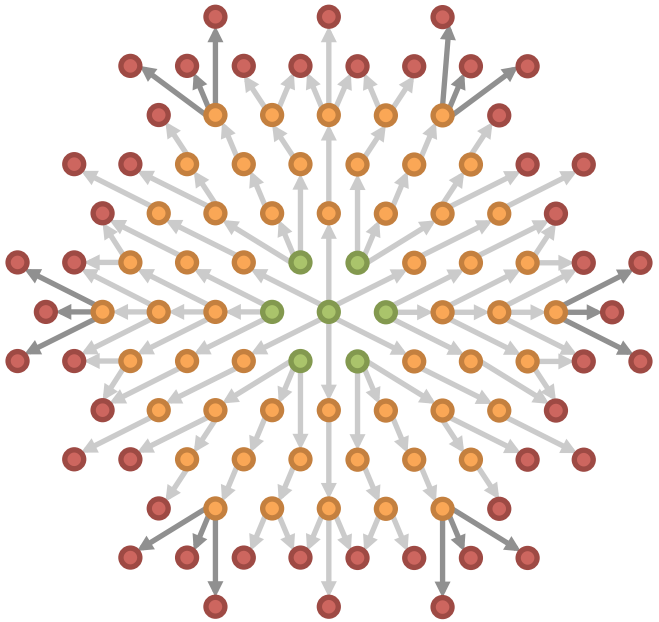


Sistemi a Regole

Programmi logici

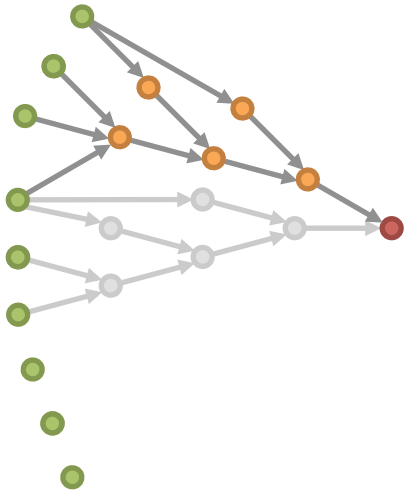


Regole di produzione

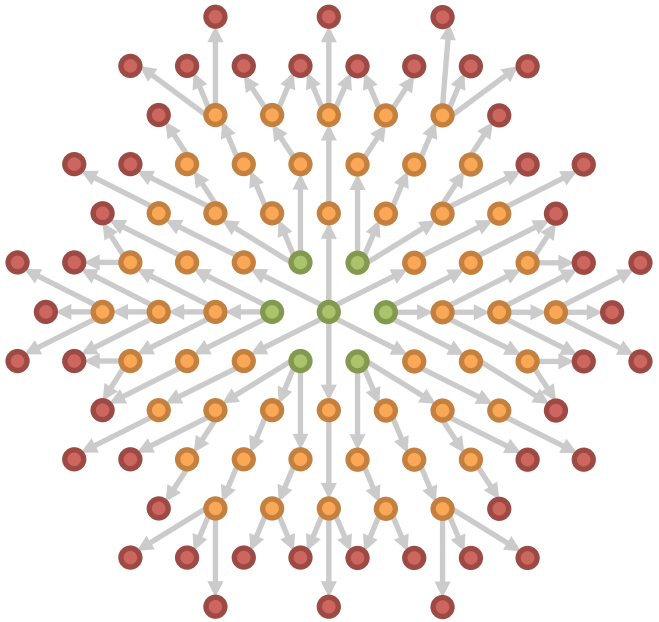


Sistemi a Regole

Programmi logici

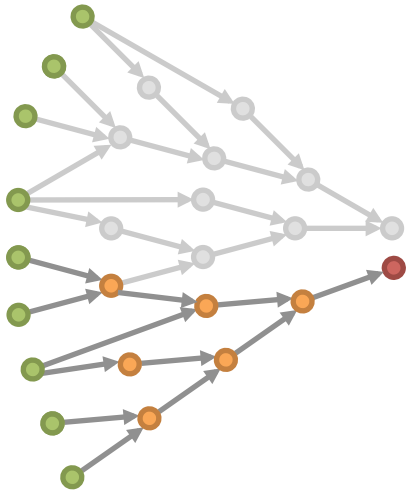


Regole di produzione

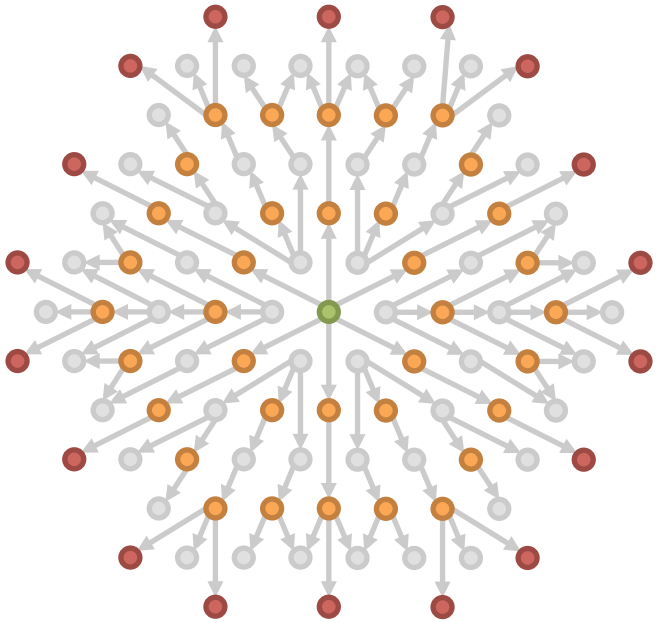


Sistemi a Regole

Programmi logici



Regole di produzione



Fondamenti di Intelligenza Artificiale M

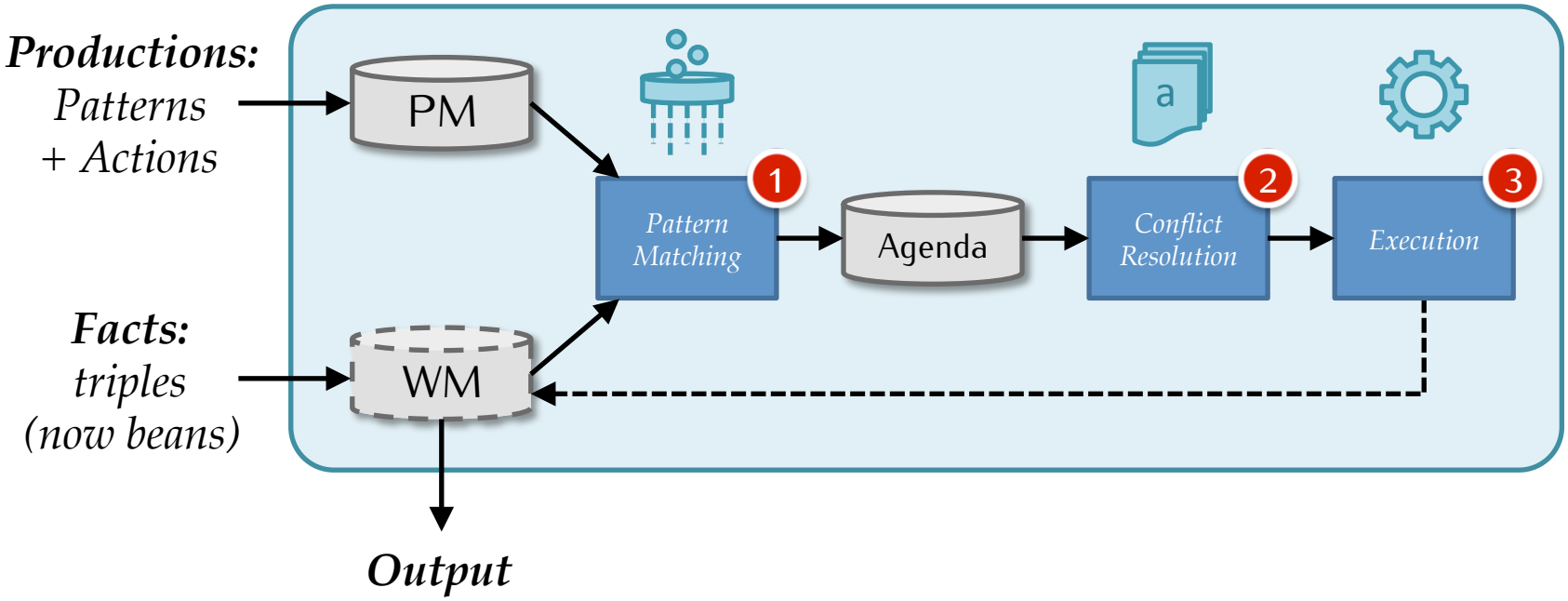
SISTEMI A REGOLE DI PRODUZIONE

Sistemi a Regole di Produzione

- I *sistemi a regole di produzione*, o **Production Rule Systems (PRS)**:
  - sono *sistemi a regole*, **Rule Based Systems (RBS)**,
  - basati sul principio deduttivo del **Modus Ponens**,
  - che adottano un **approccio reattivo/generativo**.

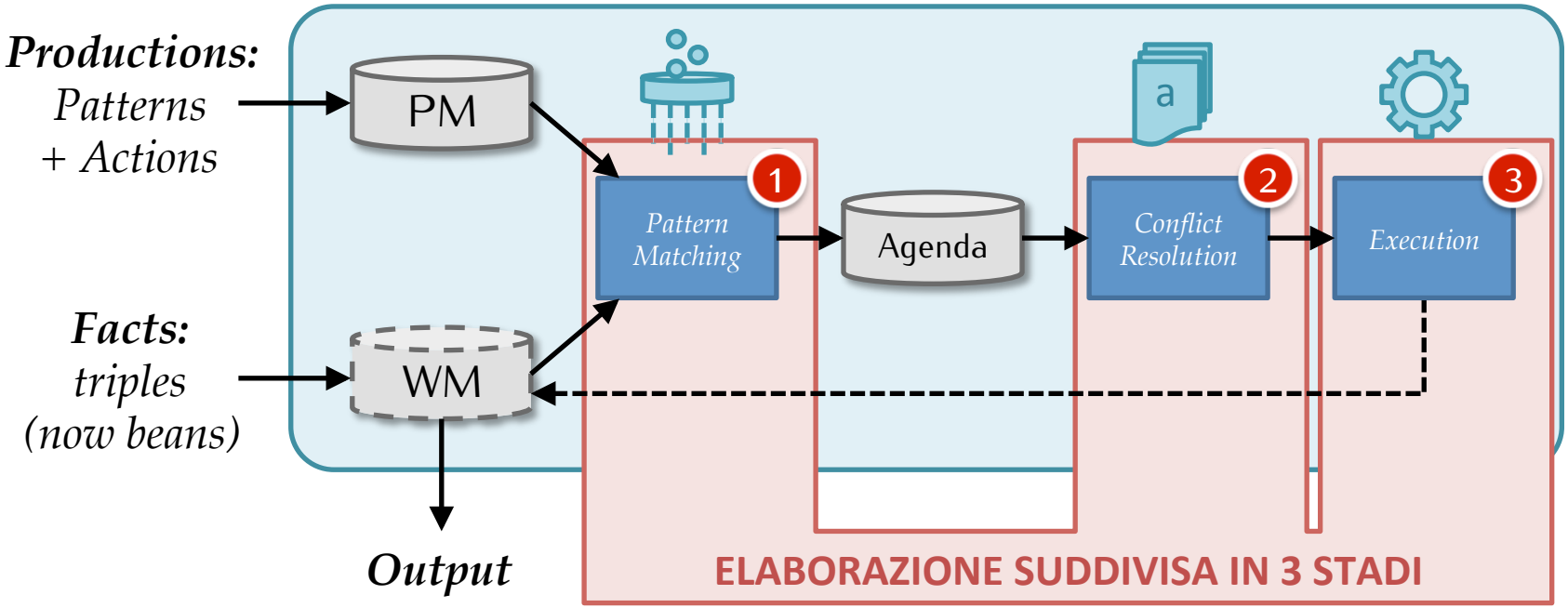
Sistemi a Regole di Produzione

Schema architetturale e principio di funzionamento



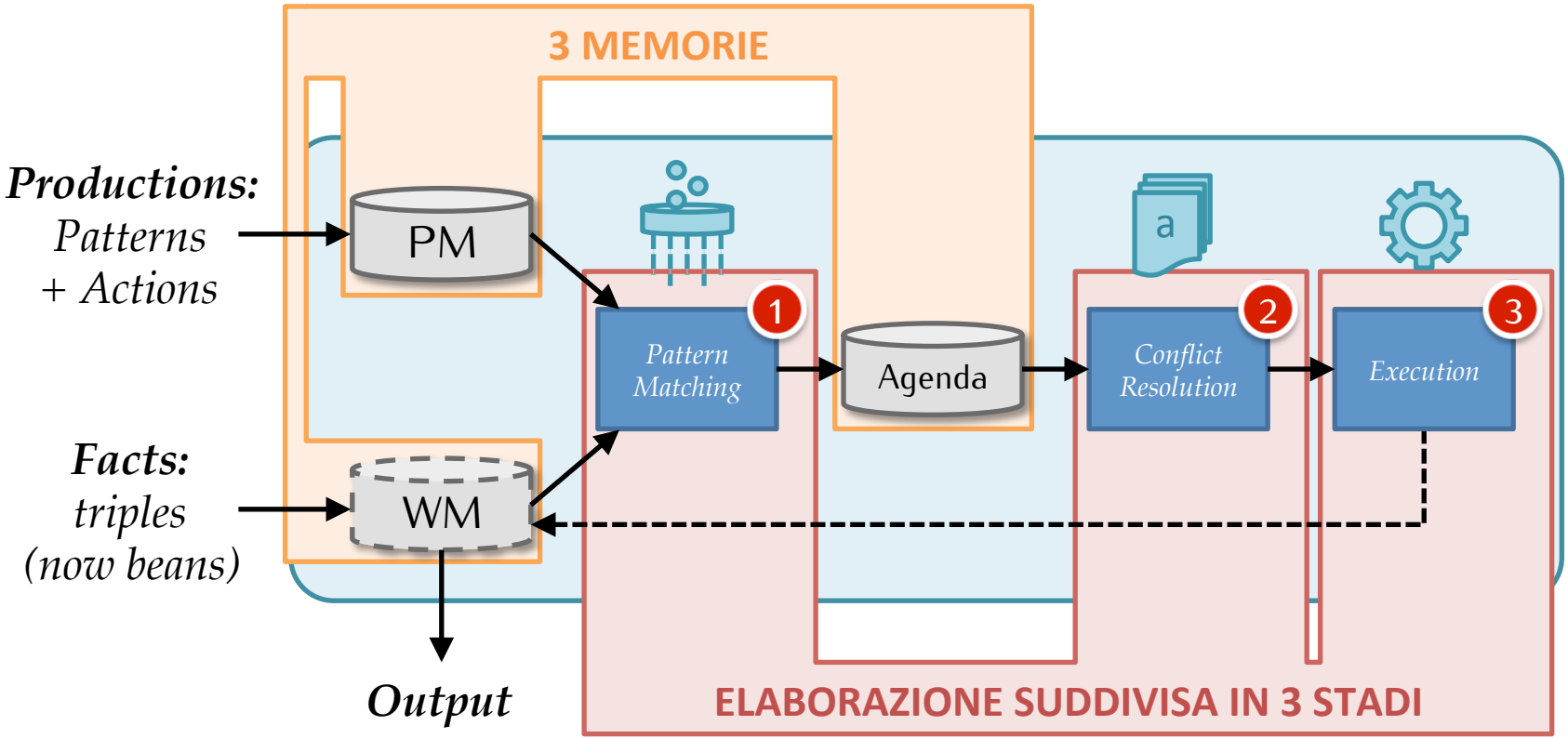
Sistemi a Regole di Produzione

Schema architetturale e principio di funzionamento



Sistemi a Regole di Produzione

Schema architetturale e principio di funzionamento



Fondamenti di Intelligenza Artificiale M

JBOSS DROOLS

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

- **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

- Sintassi del linguaggio Drools: **regole**

```
rule "ID_regola"                    /* IMPLICAZIONE */
// attributi
when                                   /* premessa */
  // pattern (composito)
then                                  /* conseguenza */
  // azioni logiche
  // effetti collaterali
end
```

JBoss Drools

- Sintassi del linguaggio Drools: **regole**

```
rule "Cancella gli Stefano"   /* IMPLICAZIONE */
salience 5
when                                  /* premessa */
  $p: Person ( name == "Stefano" )
then                                /* conseguenza */
  retract($p);
  System.out.println($p);
end
```

JBoss Drools

- Sintassi del linguaggio Drools: **queries**

```
query "ID_query"                          /* premessa */
  // pattern (composito)
end
```

JBoss Drools

- Sintassi del linguaggio Drools: **queries**

```
query "Trova gli Stefano"          /* premessa */
  $p: Person ( name == "Stefano" )
end
```

JBoss Drools

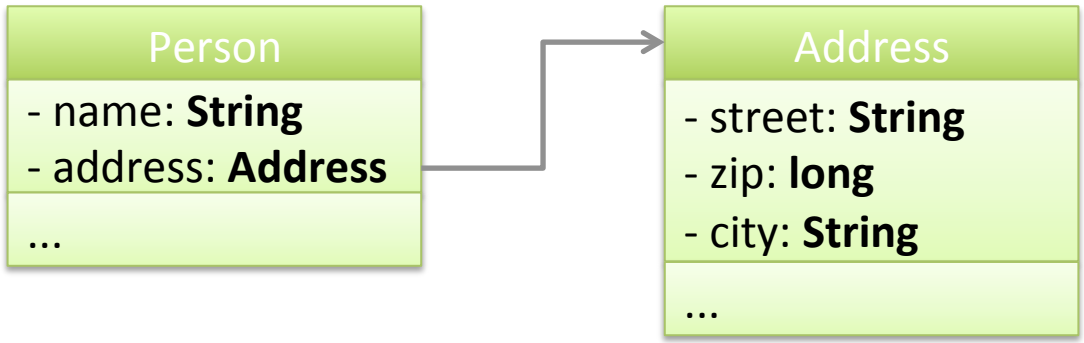
- Sintassi del linguaggio Drools: **oggetti**

```
declare ID_Class                    /* dichiarazione */
  // dichiarazione di campo
  // dichiarazione di campo
end
```

JBoss Drools

- Sintassi del linguaggio Drools: **oggetti**

```
declare Person                          /* dichiarazione */
   name: String
   address: Address = new Address(...)
end
```



JBoss Drools

- Sintassi del linguaggio Drools: **eventi**

```
declare ID_Event                    /* dichiarazione */
  // annotazioni
  // annotazioni
  // dichiarazioni di campo
  // dichiarazioni di campo
end
```

JBoss Drools

- Sintassi del linguaggio Drools: **eventi**

```
declare Alarm                          /* dichiarazione */
  @role( event )
  @timestamp( time )
  message: String
  time: long
end
```

JBoss Drools

- Sintassi del linguaggio Drools:
  - **Operatori relazionali:** == != >= > <= <
  - **Operatori logici:** && , ||
  - **Negazione:** not
  - **Operatori funzionali:** min, max, count, accumulate
  - **Dot notation:** name == "…" o $p.getName().equals("…")

JBoss Drools

- **Caso d'uso:** sistema marcatempo aziendale
  - Ogni dipendente ha un badge con RFID
  - Quando passa attraverso un gate, la WM riceve un evento "passed" (marcato temporalmente) attraverso l'entry-point corrispondente al dipendente
    - Filtrando il primo e l'ultimo evento "passed" di ogni giorno per ogni dipendente, si determina quante ore ha lavorato
    - Se non ci sono eventi "passed" per un dato dipendente in un dato giorno e non è in vacanza/malato, allora è assente
    - Accumulando il numero di ore lavorate in un mese da un dipendente si può calcolare in proporzione la sua busta paga
    - Sapendo che un dipendente è assegnato a un progetto, l'azienda può calcolare i mesi/uomo dedicati a quel progetto

Fondamenti di Intelligenza Artificiale M

PATTERN MATCHING: L'ALGORITMO RETE

Pattern Matching: l'algoritmo RETE

1A REGOLA DI ESEMPIO

Pattern matching: l'algoritmo RETE

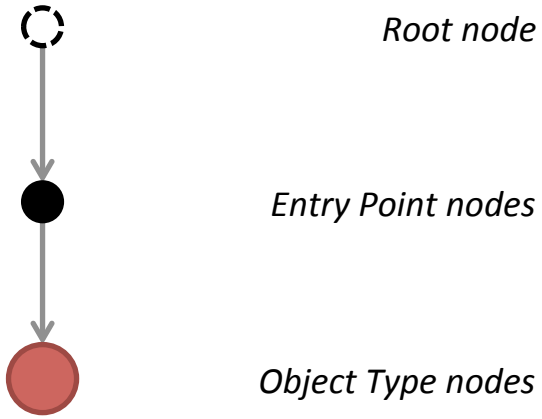
```
rule "Trova gli Stefano"
when
  $p: Person( name == "Stefano" )
then
  System.out.println($p);
end
```

Root node

Entry Point nodes

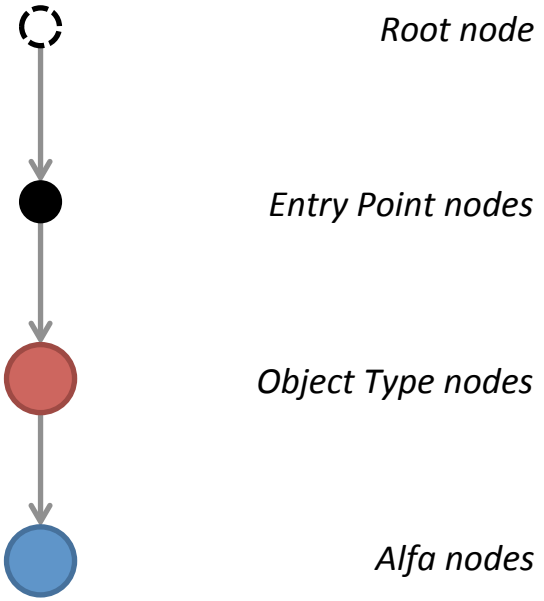
Pattern matching: l'algoritmo RETE

```
rule "Trova gli Stefano"
when
  $p: Person( name == "Stefano" )
then
  System.out.println($p);
end
```



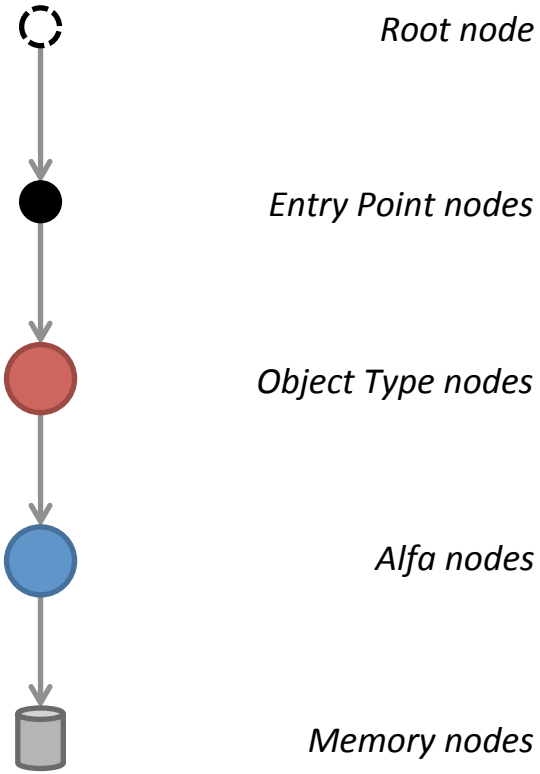
Pattern matching: l'algoritmo RETE

```
rule "Trova gli Stefano"
when
  $p: Person( name == "Stefano" )
then
  System.out.println($p);
end
```



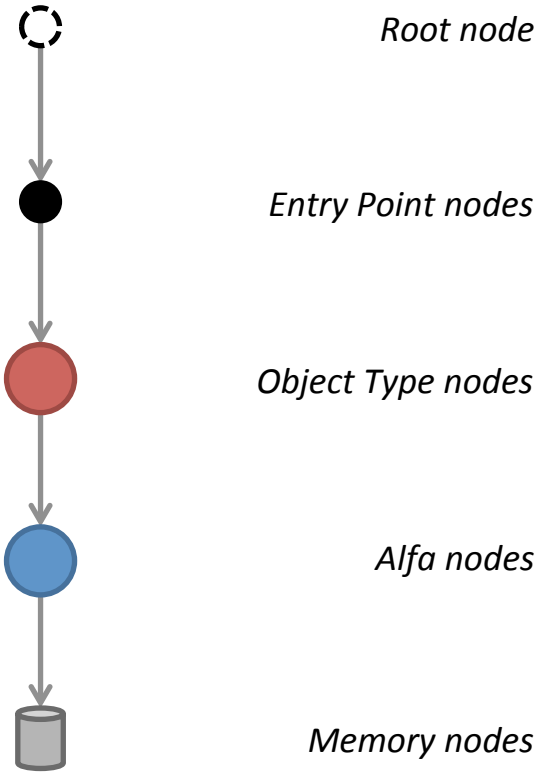
Pattern matching: l'algoritmo RETE

```
rule "Trova gli Stefano"
when
  $p: Person( name == "Stefano" )
then
  System.out.println($p);
end
```



Pattern matching: l'algoritmo RETE

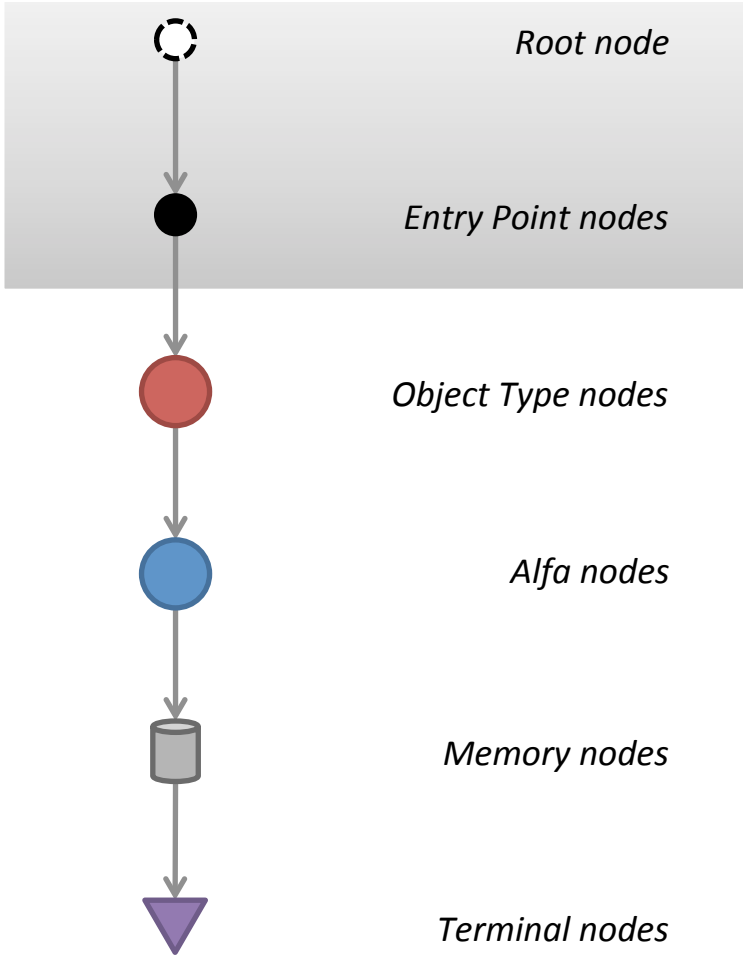
```
rule "Trova gli Stefano"
when
  $p: Person( name == "Stefano" )
then
  System.out.println($p);
end
```



NB: I fatti contenuti in un (Alfa) Memory Node fanno match con un pattern semplice!

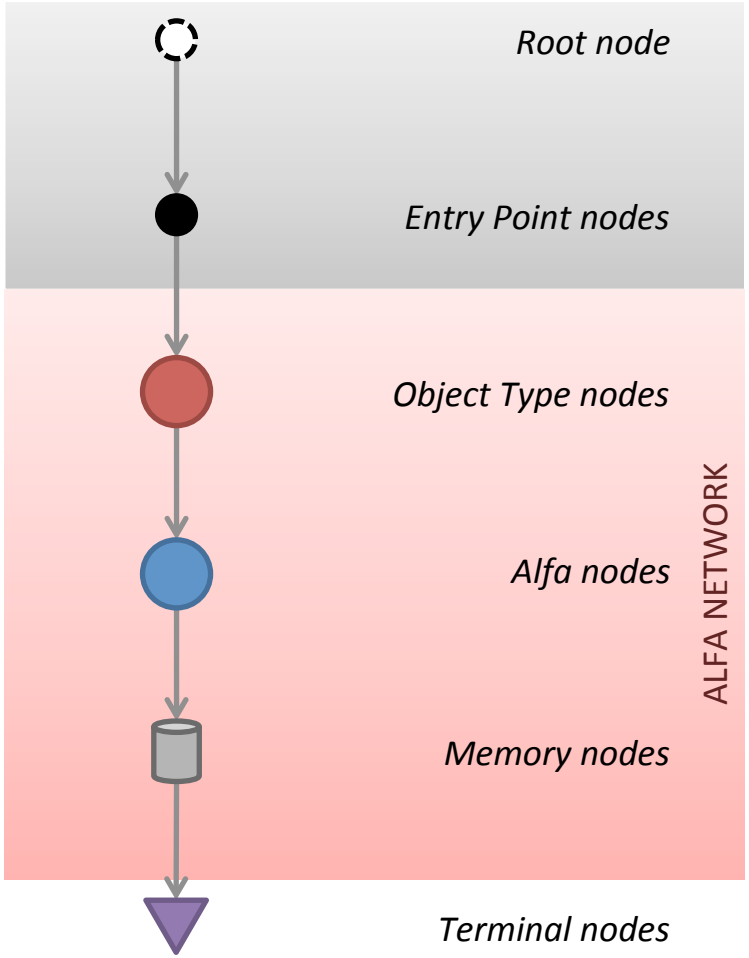
Pattern matching: l'algoritmo RETE

```
rule "Trova gli Stefano"
when
  $p: Person( name == "Stefano" )
then
  System.out.println($p);
end
```



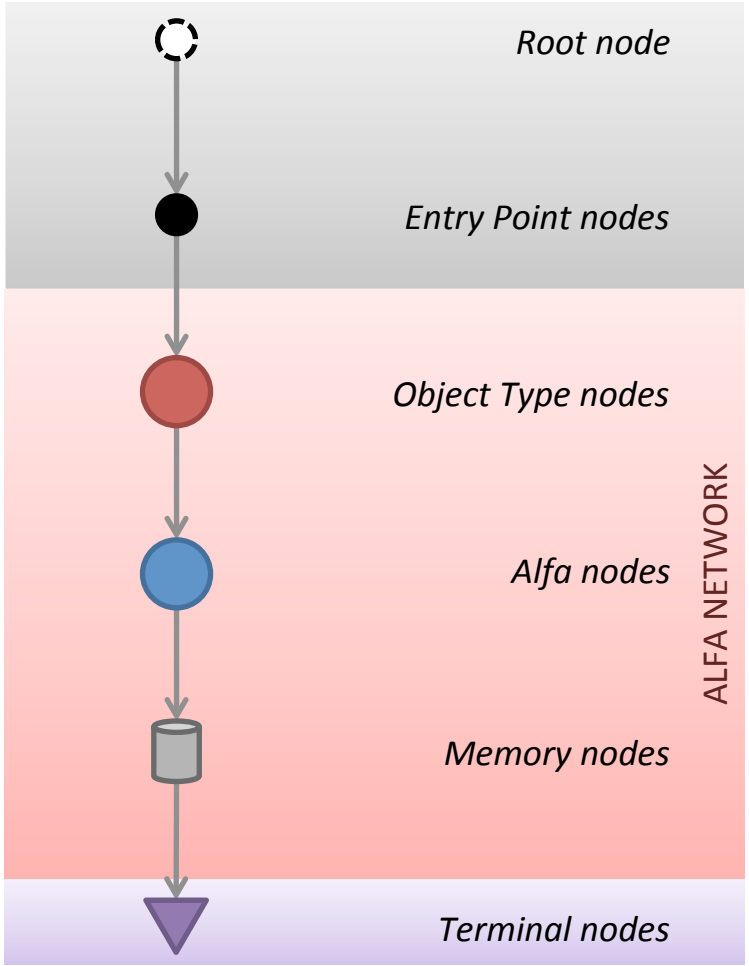
Pattern matching: l'algoritmo RETE

```
rule "Trova gli Stefano"
when
  $p: Person( name == "Stefano" )
then
  System.out.println($p);
end
```



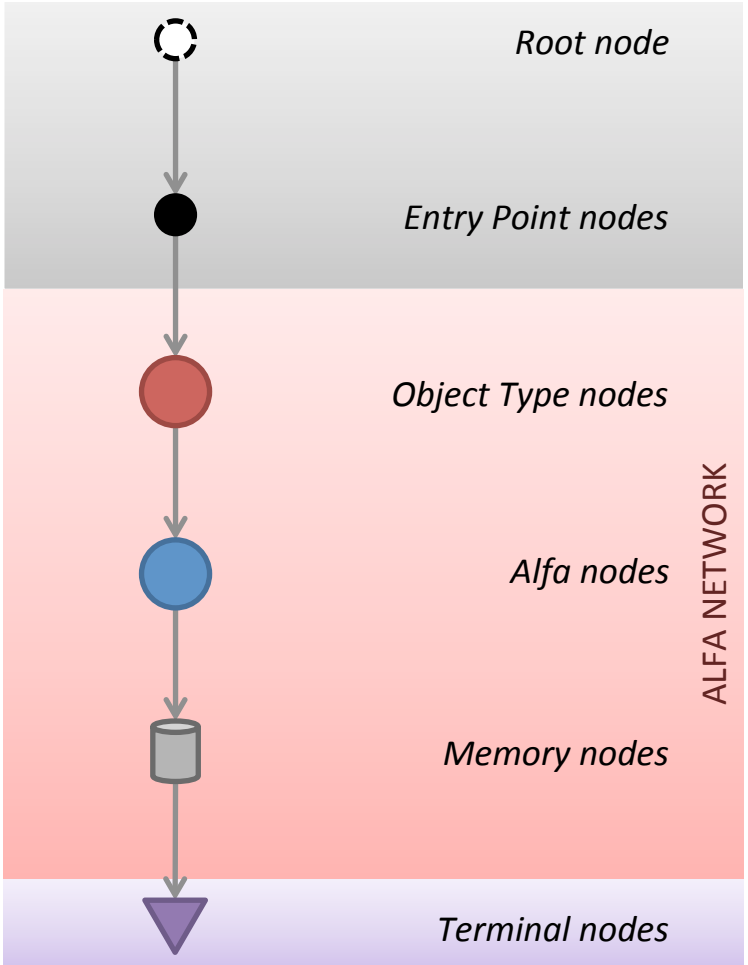
Pattern matching: l'algoritmo RETE

```
rule "Trova gli Stefano"
when
  $p: Person( name == "Stefano" )
then
  System.out.println($p);
end
```



Pattern matching: l'algoritmo RETE

```
rule "Trova gli Stefano"
when
  $p: Person( name == "Stefano" )
then
  System.out.println($p);
end
```

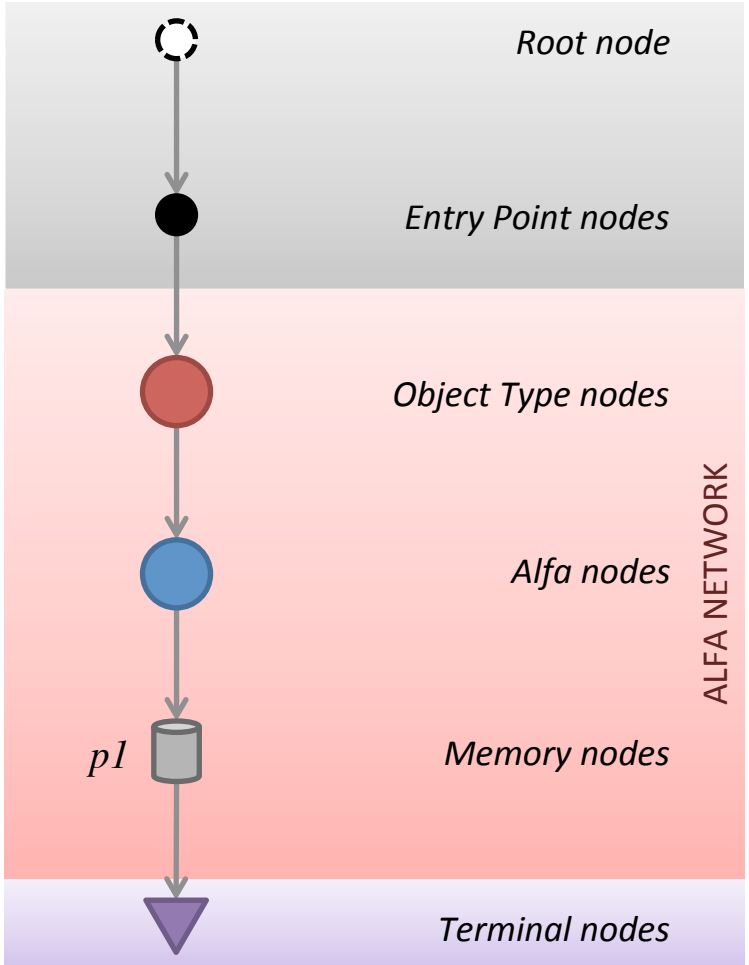


Pattern matching: l'algoritmo RETE

```
rule "Trova gli Stefano"
when
  $p: Person( name == "Stefano" )
then
  System.out.println($p);
end
```

p1: Person("Stefano", null)

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

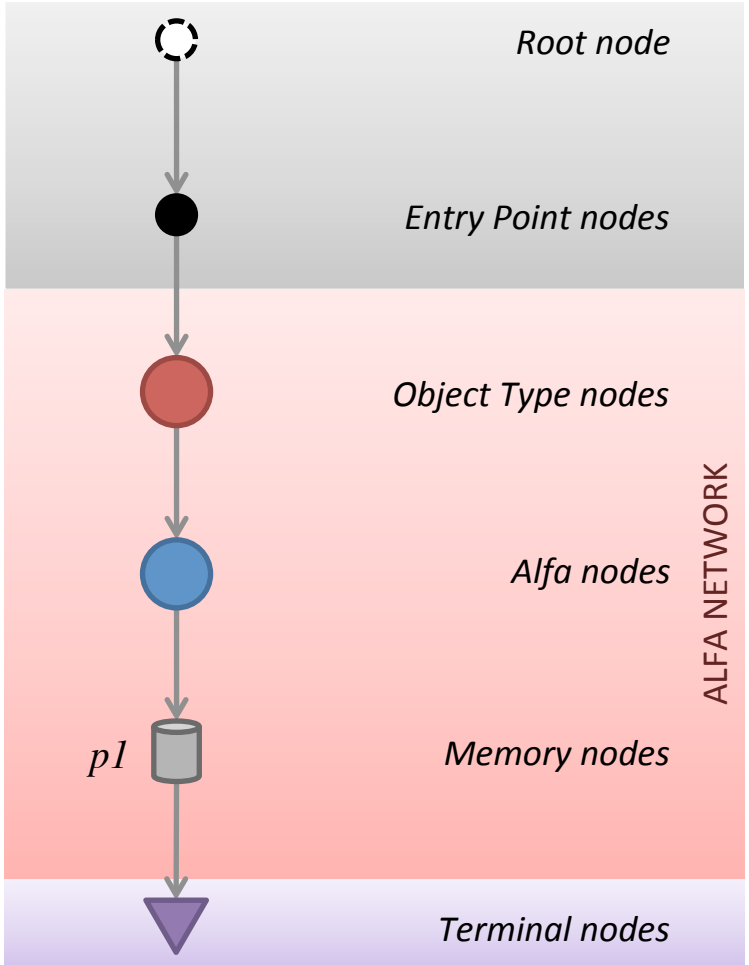


Pattern matching: l'algoritmo RETE

```
rule "Trova gli Stefano"
when
  $p: Person( name == "Stefano" )
then
  System.out.println($p);
end
```

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

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

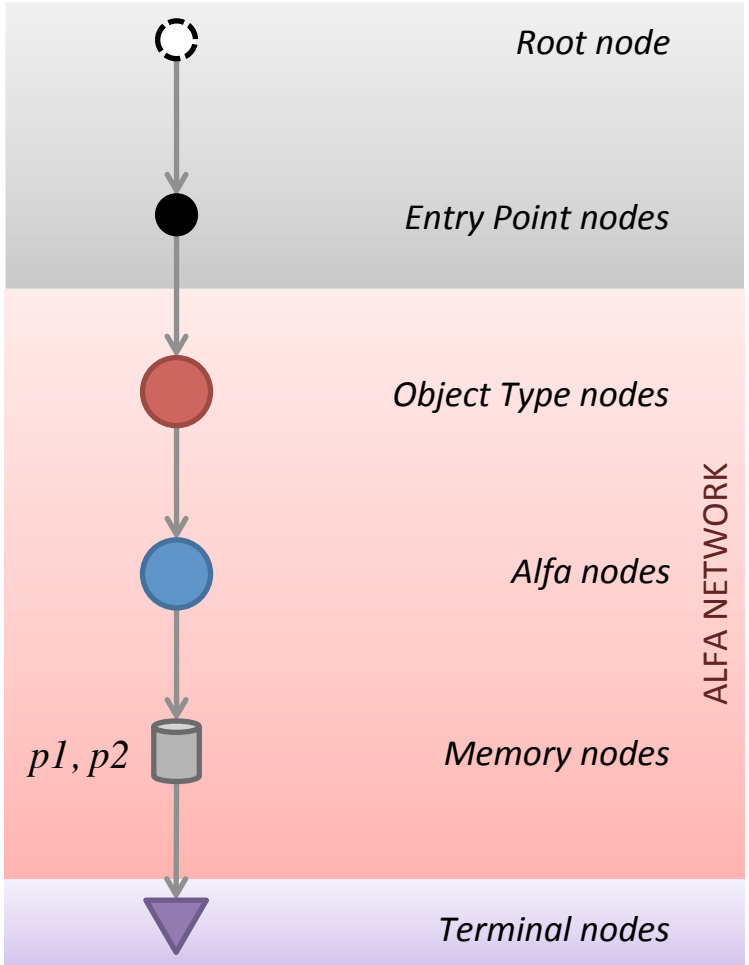


Pattern matching: l'algoritmo RETE

```
rule "Trova gli Stefano"
when
  $p: Person( name == "Stefano" )
then
  System.out.println($p);
end
```

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

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

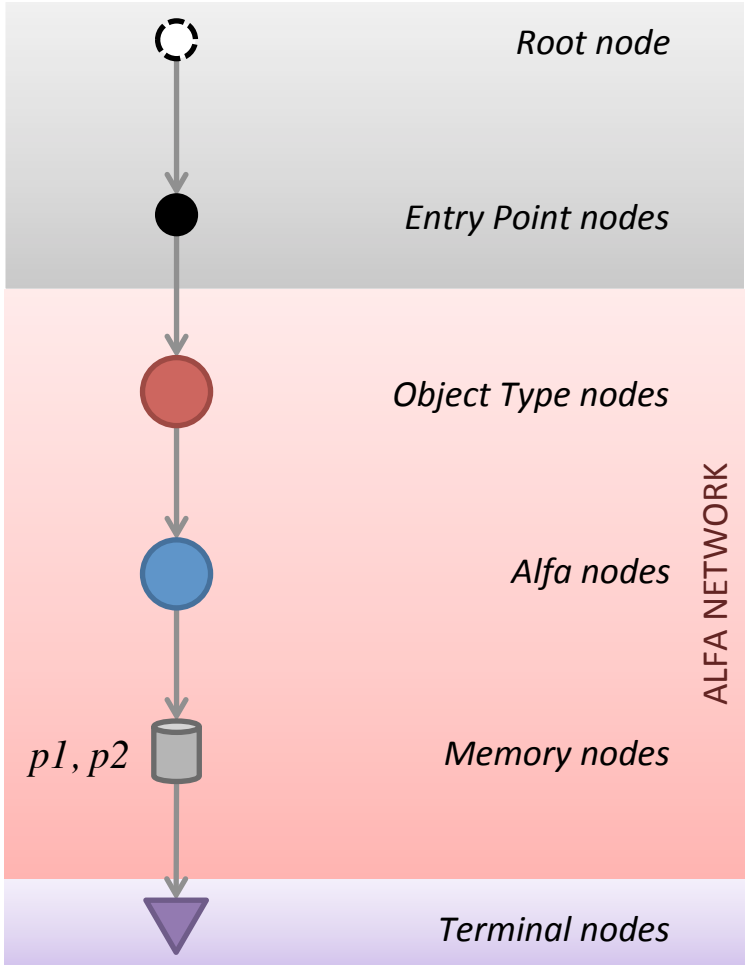


Pattern matching: l'algoritmo RETE

```
rule "Trova gli Stefano"
when
  $p: Person( name == "Stefano" )
then
  System.out.println($p);
end
```

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

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



Pattern Matching: l'algoritmo RETE

2A REGOLA DI ESEMPIO

Pattern matching: l'algoritmo RETE

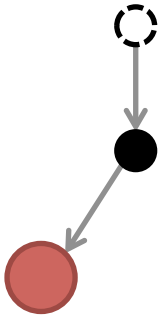
```
rule "Trova gli Stefano e indirizzi"
when
  $a: Address()
  $p: Person( name == "Stefano" )
then
  System.out.println($p+"/"+$a+" ");
end
```

Root node

Entry Point nodes

Pattern matching: l'algoritmo RETE

```
rule "Trova gli Stefano e indirizzi"
when
  $a: Address()
  $p: Person( name == "Stefano" )
then
  System.out.println($p+"/"+$a+" ");
end
```



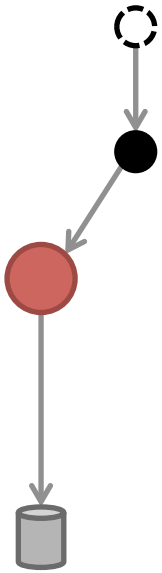
Root node

Entry Point nodes

Object Type nodes

Pattern matching: l'algoritmo RETE

```
rule "Trova gli Stefano e indirizzi"
when
  $a: Address()
  $p: Person( name == "Stefano" )
then
  System.out.println($p+"/"+$a+" ");
end
```



Root node

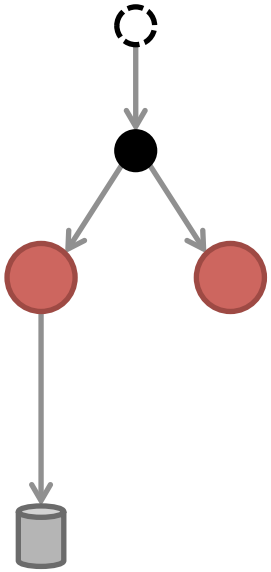
Entry Point nodes

Object Type nodes

Memory nodes

Pattern matching: l'algoritmo RETE

```
rule "Trova gli Stefano e indirizzi"
when
  $a: Address()
  $p: Person( name == "Stefano" )
then
  System.out.println($p+"/"+$a+" ");
end
```



Root node

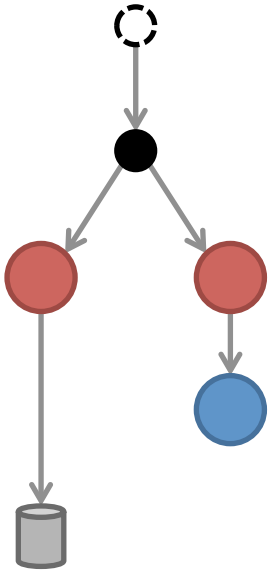
Entry Point nodes

Object Type nodes

Memory nodes

Pattern matching: l'algoritmo RETE

```
rule "Trova gli Stefano e indirizzi"
when
  $a: Address()
  $p: Person( name == "Stefano" )
then
  System.out.println($p+"/"+$a+" ");
end
```



Root node

Entry Point nodes

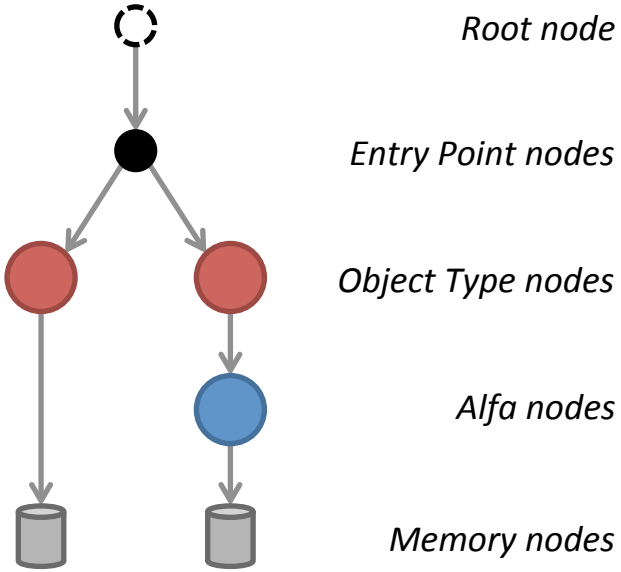
Object Type nodes

Alfa nodes

Memory nodes

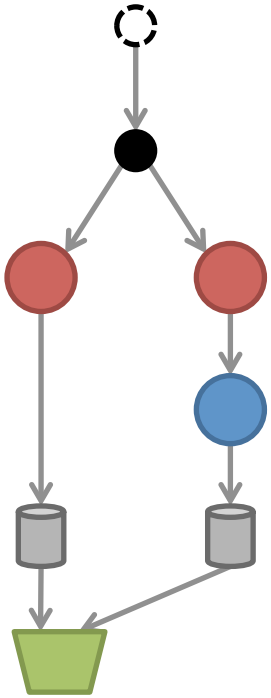
Pattern matching: l'algoritmo RETE

```
rule "Trova gli Stefano e indirizzi"
when
  $a: Address()
  $p: Person( name == "Stefano" )
then
  System.out.println($p+"/"+$a+" ");
end
```



Pattern matching: l'algoritmo RETE

```
rule "Trova gli Stefano e indirizzi"
when
  $a: Address()
  $p: Person( name == "Stefano" )
then
  System.out.println($p+"/"+$a+" ");
end
```



Root node

Entry Point nodes

Object Type nodes

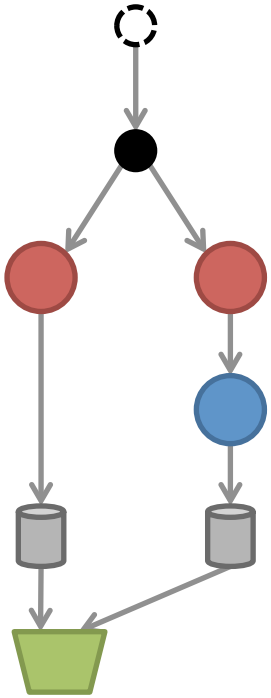
Alfa nodes

Memory nodes

Beta nodes

Pattern matching: l'algoritmo RETE

```
rule "Trova gli Stefano e indirizzi"
when
  $a: Address()
  $p: Person( name == "Stefano" )
then
  System.out.println($p+"/"+$a+" ");
end
```



Root node

Entry Point nodes

Object Type nodes

Alfa nodes

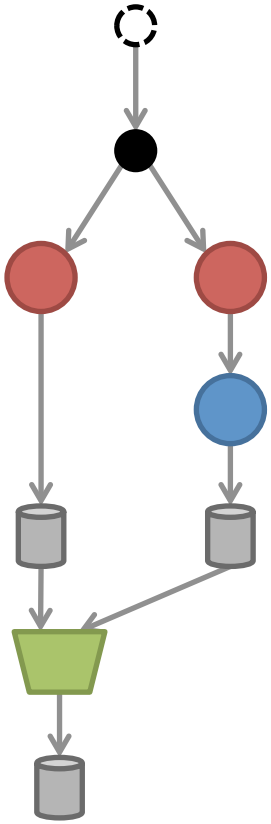
Memory nodes

Beta nodes

NB: I Beta Nodes fanno il prodotto cartesiano degli oggetti filtrati dagli Alfa padre!

Pattern matching: l'algoritmo RETE

```
rule "Trova gli Stefano e indirizzi"
when
  $a: Address()
  $p: Person( name == "Stefano" )
then
  System.out.println($p+"/"+$a+" ");
end
```



Root node

Entry Point nodes

Object Type nodes

Alfa nodes

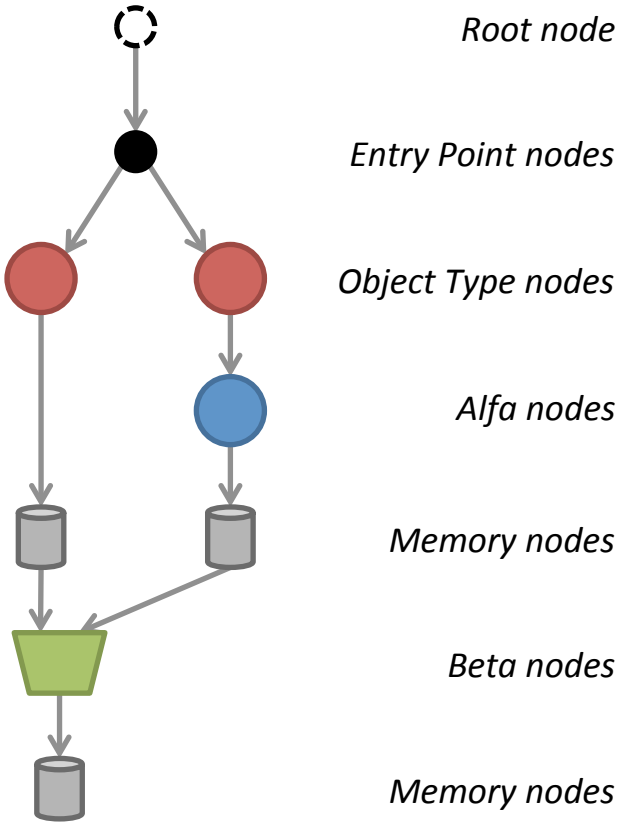
Memory nodes

Beta nodes

Memory nodes

Pattern matching: l'algoritmo RETE

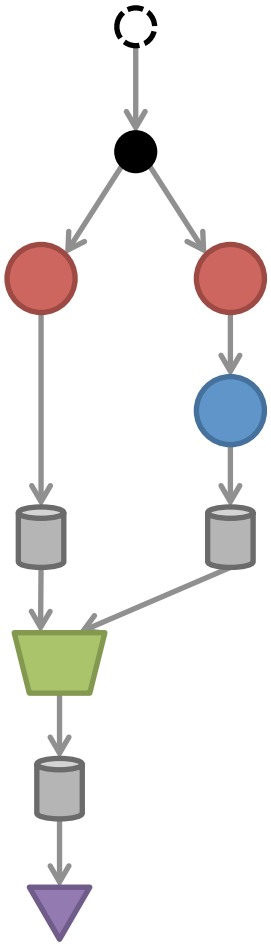
```
rule "Trova gli Stefano e indirizzi"
when
  $a: Address()
  $p: Person( name == "Stefano" )
then
  System.out.println($p+"/"+$a+" ");
end
```



NB: Le tuple contenute in un (Beta) Memory Node fanno match con un pattern composito!

Pattern matching: l'algoritmo RETE

```
rule "Trova gli Stefano e indirizzi"
when
  $a: Address()
  $p: Person( name == "Stefano" )
then
  System.out.println($p+"/"+$a+" ");
end
```



Root node

Entry Point nodes

Object Type nodes

Alfa nodes

Memory nodes

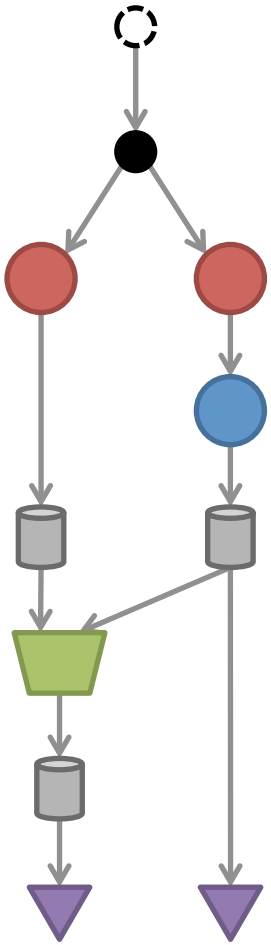
Beta nodes

Memory nodes

Terminal nodes

Pattern matching: l'algoritmo RETE

```
rule "Trova gli Stefano e indirizzi"
when
  $a: Address()
  $p: Person( name == "Stefano" )
then
  System.out.println($p+"/"+$a+" ");
end
```



Root node

Entry Point nodes

Object Type nodes

Alfa nodes

Memory nodes

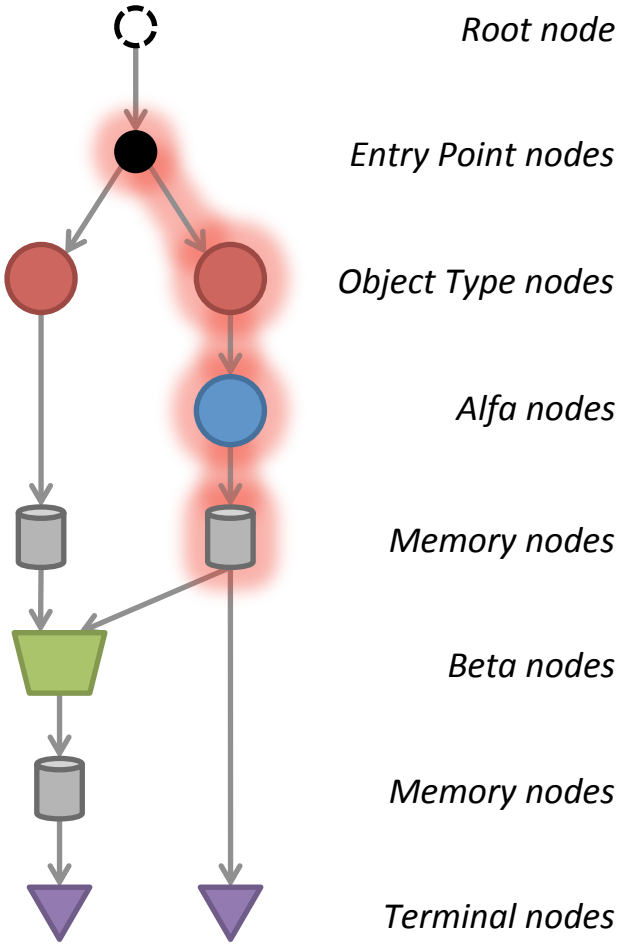
Beta nodes

Memory nodes

Terminal nodes

Pattern matching: l'algoritmo RETE

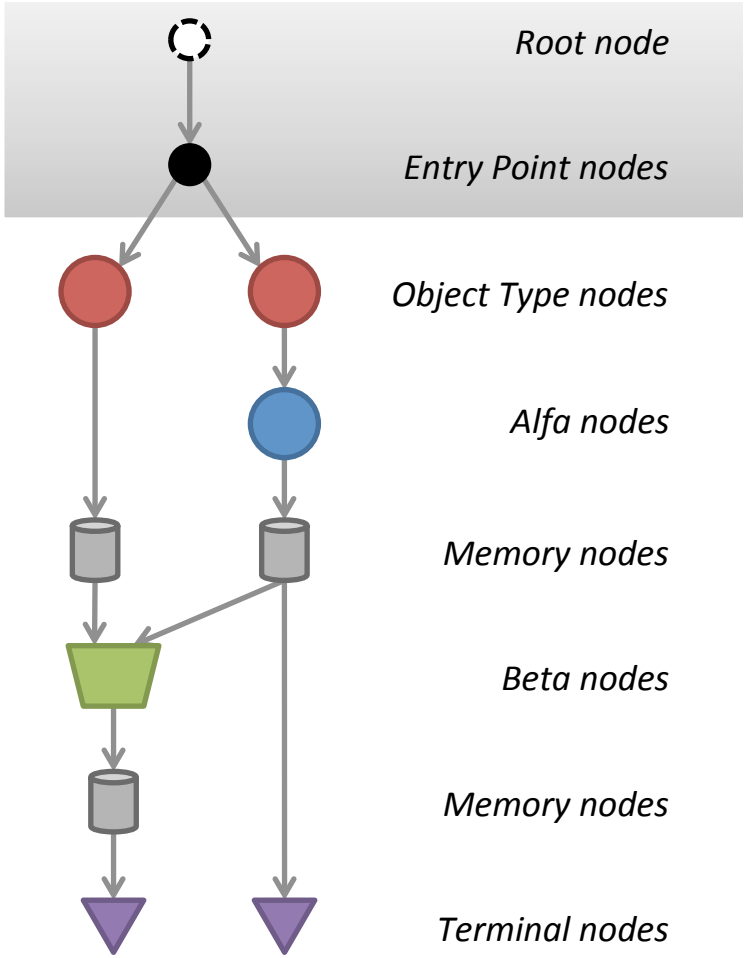
```
rule "Trova gli Stefano e indirizzi"
when
  $a: Address()
  $p: Person( name == "Stefano" )
then
  System.out.println($p+"/"+$a+" ");
end
```



NB: Introduco la regola precedente: I nodi della RETE vengono condivisi quando possibile!

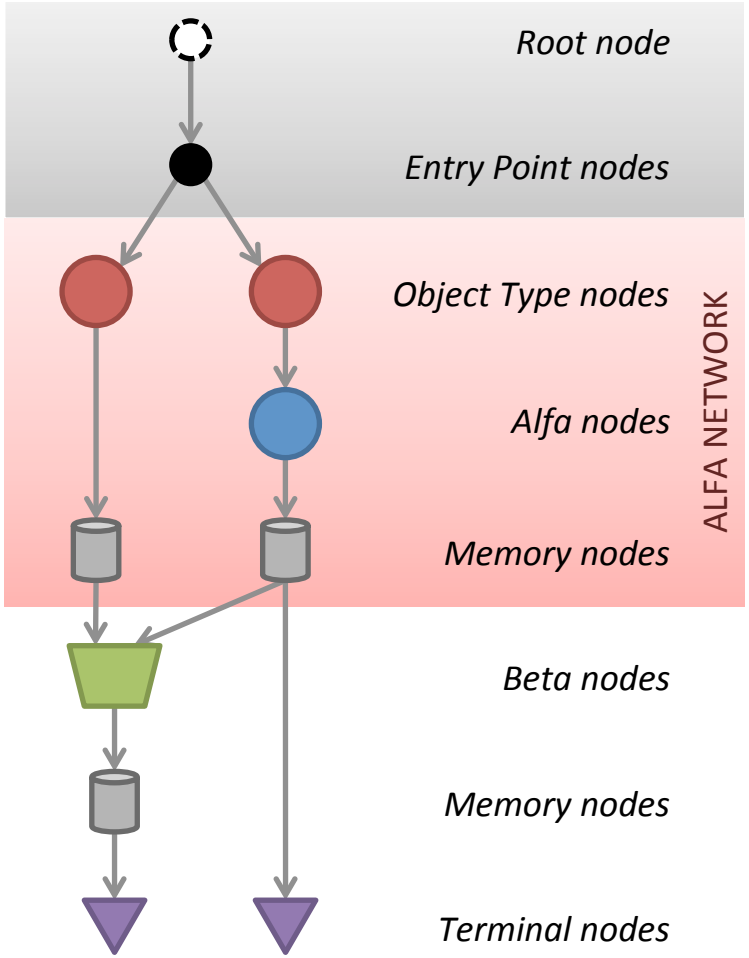
Pattern matching: l'algoritmo RETE

```
rule "Trova gli Stefano e indirizzi"
when
  $a: Address()
  $p: Person( name == "Stefano" )
then
  System.out.println($p+"/"+$a+" ");
end
```



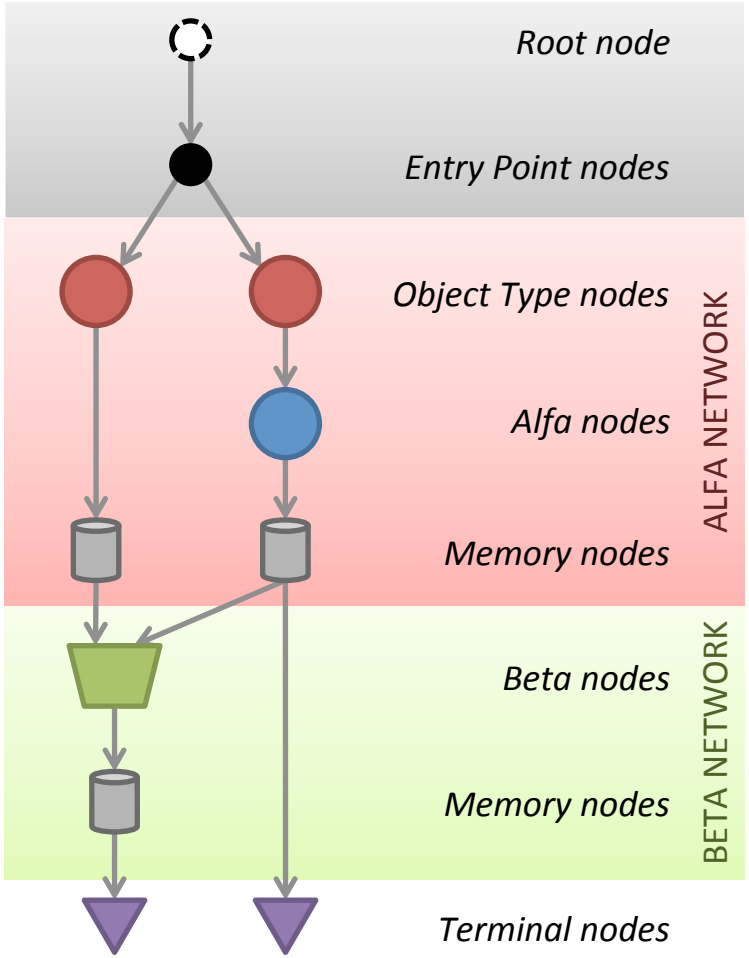
Pattern matching: l'algoritmo RETE

```
rule "Trova gli Stefano e indirizzi"
when
  $a: Address()
  $p: Person( name == "Stefano" )
then
  System.out.println($p+"/"+$a+" ");
end
```



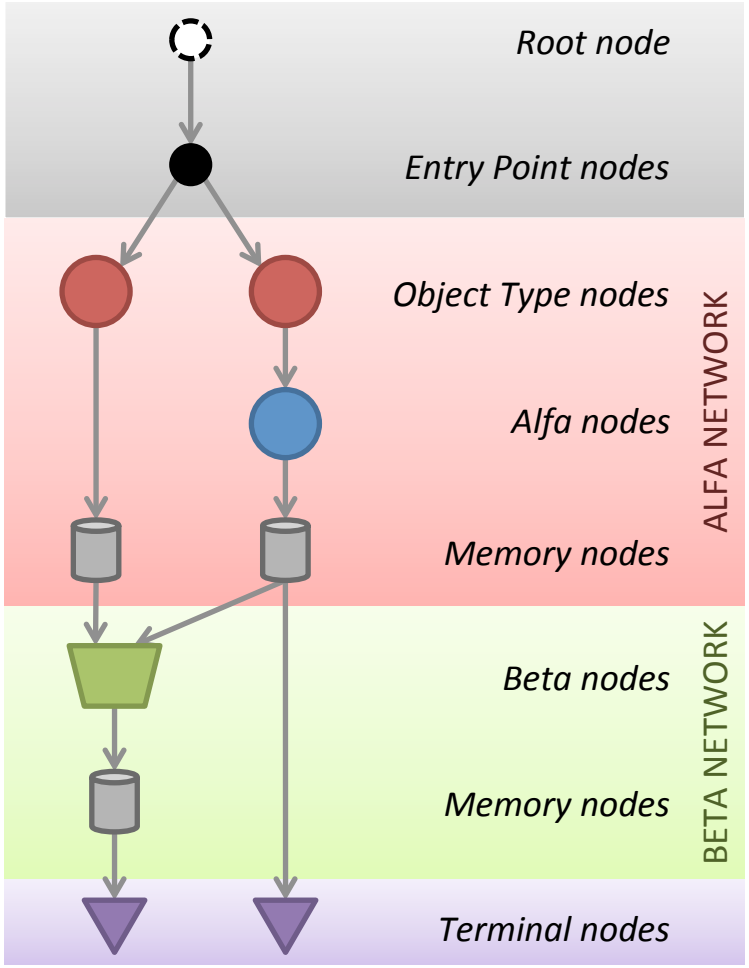
Pattern matching: l'algoritmo RETE

```
rule "Trova gli Stefano e indirizzi"
when
  $a: Address()
  $p: Person( name == "Stefano" )
then
  System.out.println($p+"/"+$a+" ");
end
```



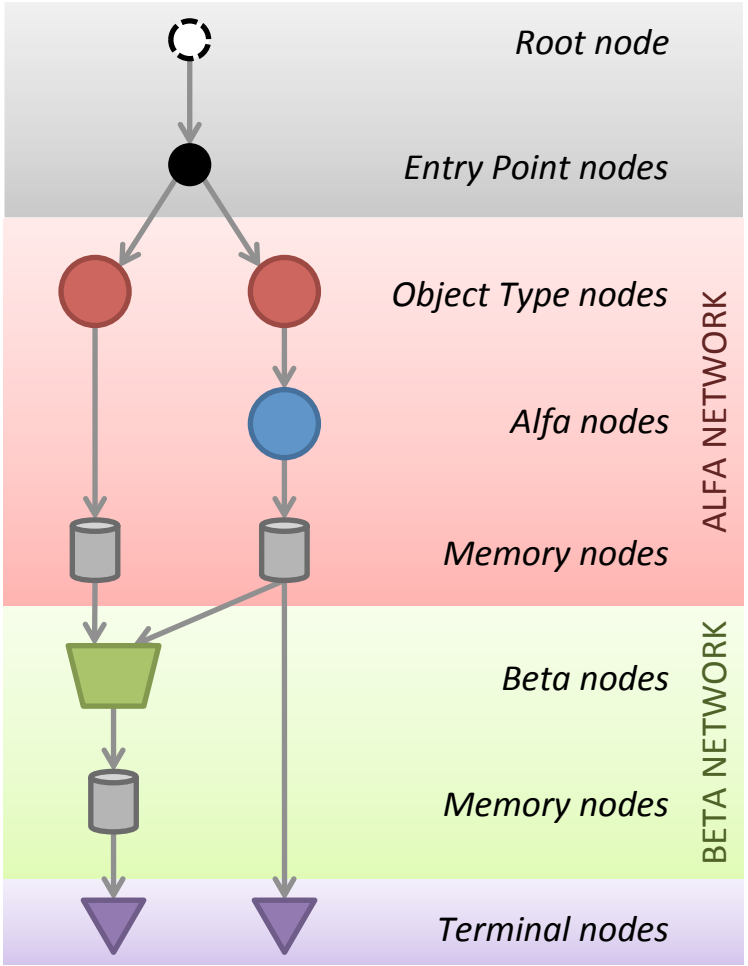
Pattern matching: l'algoritmo RETE

```
rule "Trova gli Stefano e indirizzi"
when
  $a: Address()
  $p: Person( name == "Stefano" )
then
  System.out.println($p+"/"+$a+" ");
end
```



Pattern matching: l'algoritmo RETE

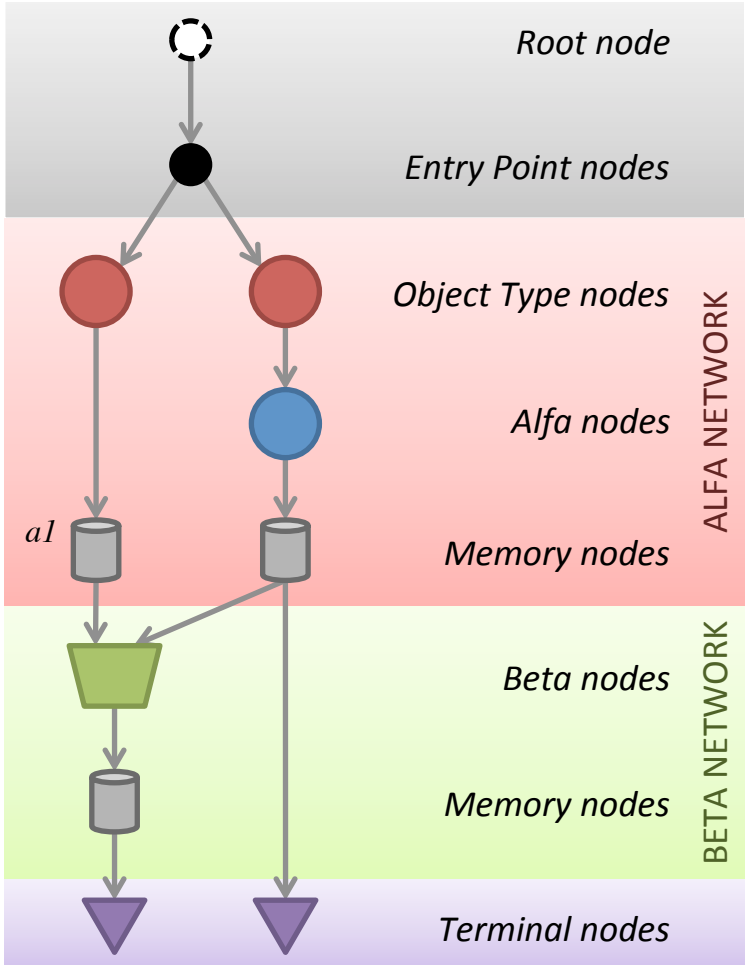
```
rule "Trova gli Stefano e indirizzi"
when
  $a: Address()
  $p: Person( name == "Stefano" )
then
  System.out.println($p+"/"+$a+" ");
end
```



Pattern matching: l'algoritmo RETE

```
rule "Trova gli Stefano e indirizzi"
when
  $a: Address()
  $p: Person( name == "Stefano" )
then
  System.out.println($p+"/"+$a+" ");
end
```

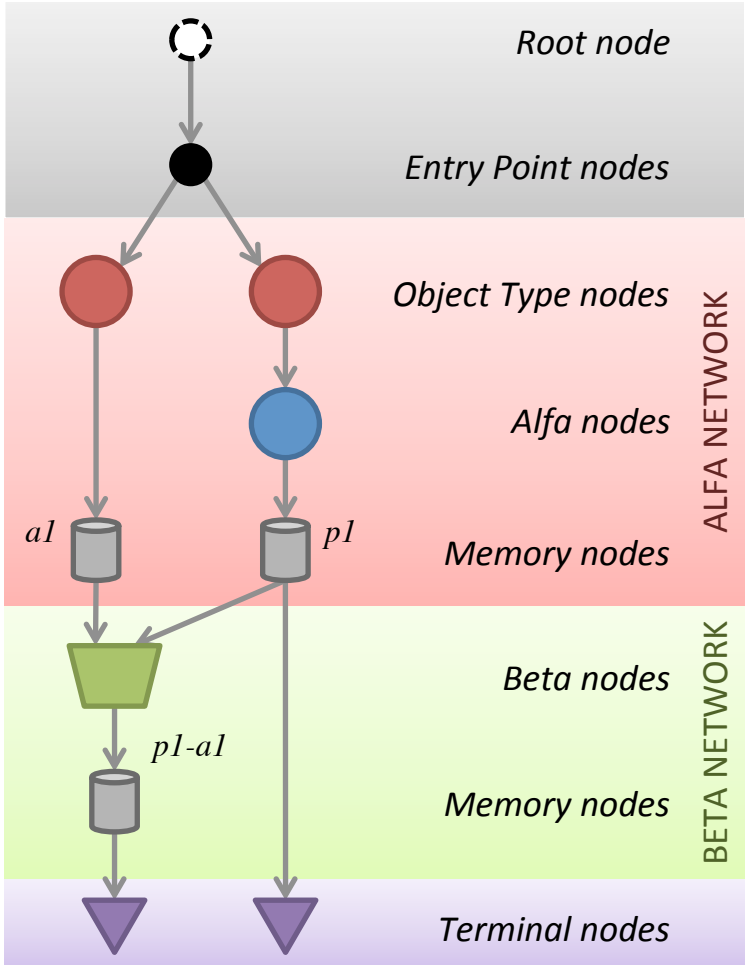
a1: Address("Via Po 2", 40068, "San Lazzaro")



Pattern matching: l'algoritmo RETE

```
rule "Trova gli Stefano e indirizzi"
when
  $a: Address()
  $p: Person( name == "Stefano" )
then
  System.out.println($p+"/"+$a+" ");
end
```

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



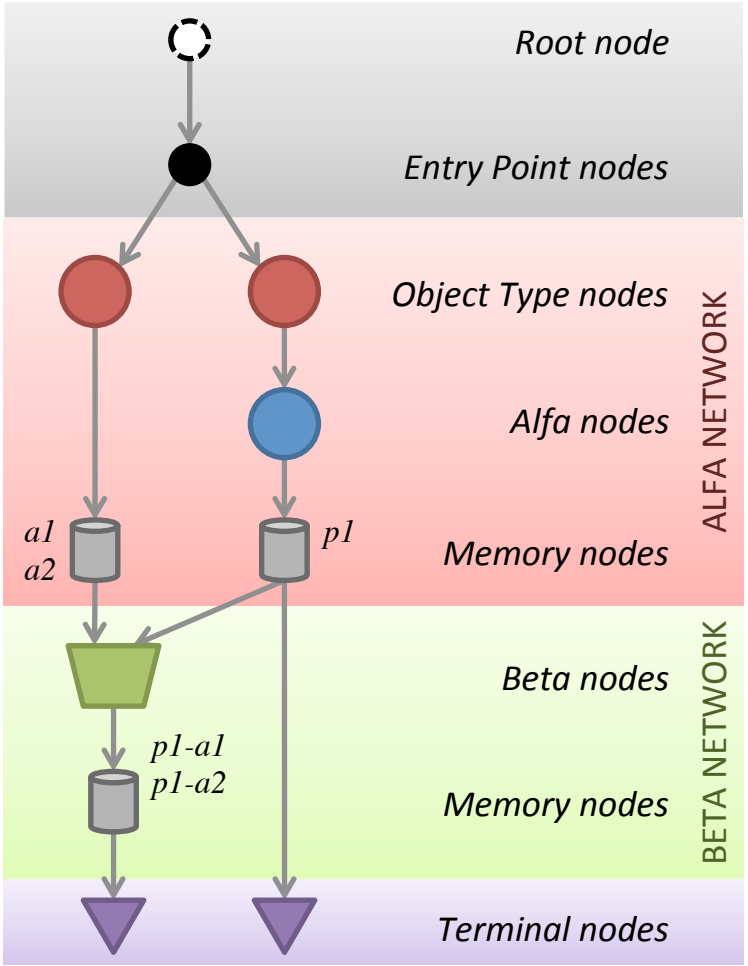

```
Person[p1, -]/Address[a1]   _
```

NB: Stampa semplificata e output prima regola omesso!

Pattern matching: l'algoritmo RETE

```
rule "Trova gli Stefano e indirizzi"
when
  $a: Address()
  $p: Person( name == "Stefano" )
then
  System.out.println($p+"/"+$a+" ");
end
```

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



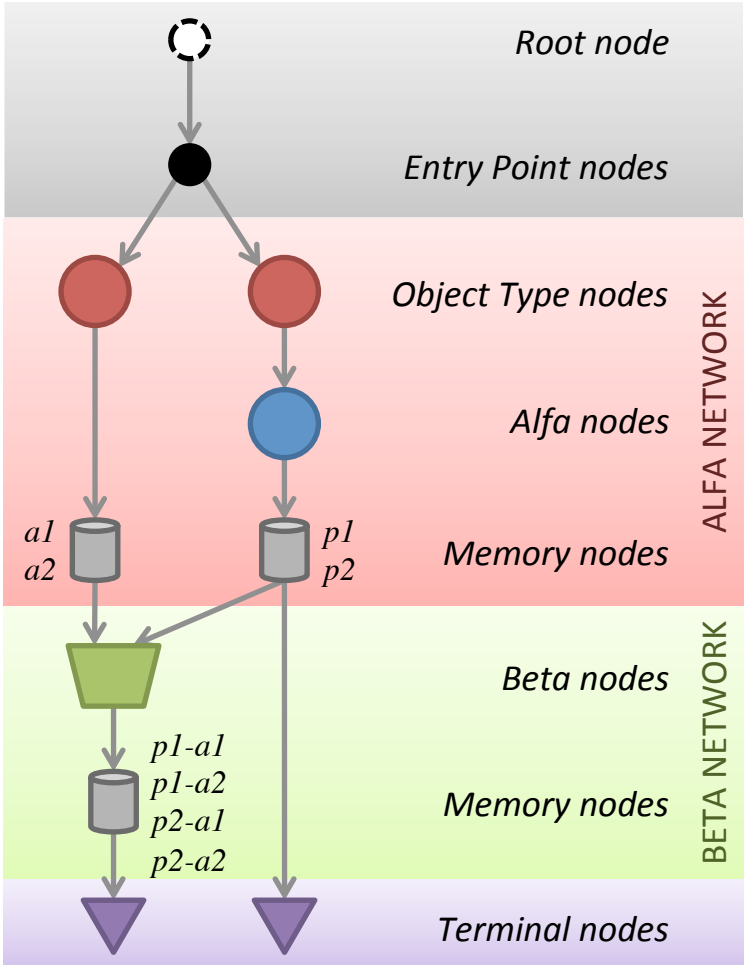

```
Person[p1, -]/Address[a1]   Person[p1, -]/Address[a2]
_
```

NB: Stampa semplificata e output prima regola omesso!

Pattern matching: l'algoritmo RETE

```
rule "Trova gli Stefano e indirizzi"
when
  $a: Address()
  $p: Person( name == "Stefano" )
then
  System.out.println($p+"/"+$a+" ");
end
```

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




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

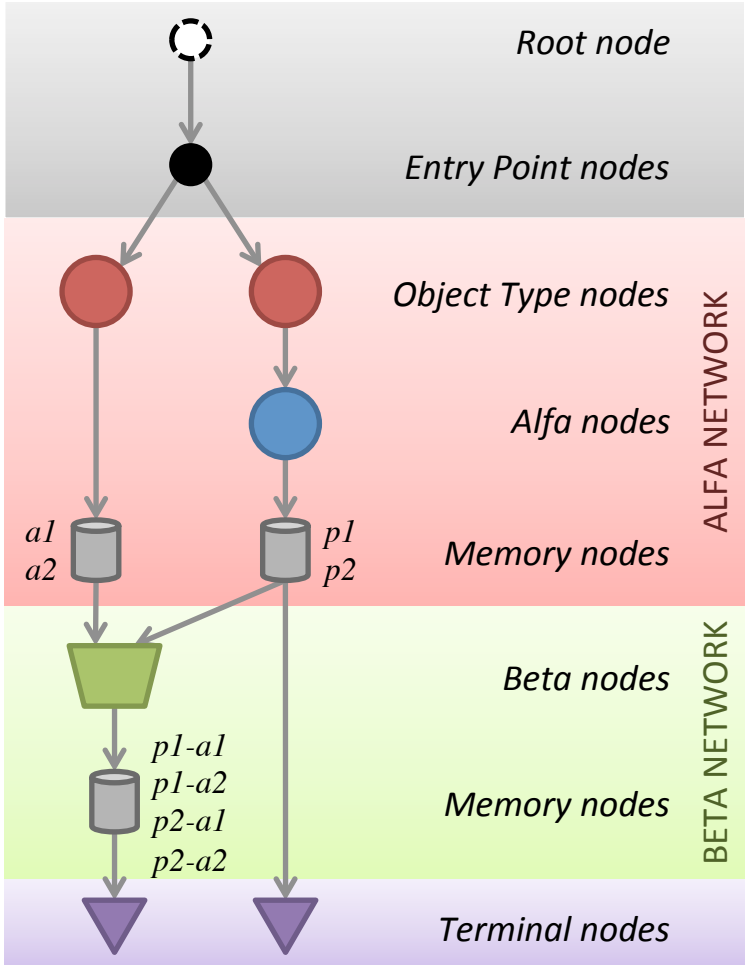
NB: Stampa semplificata e output prima regola omesso!

Pattern matching: l'algoritmo RETE

```
rule "Trova gli Stefano e indirizzi"
when
    $a: Address()
    $p: Person( name == "Stefano" )
then
    System.out.println($p+"/"+$a+" ");
end
```

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

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



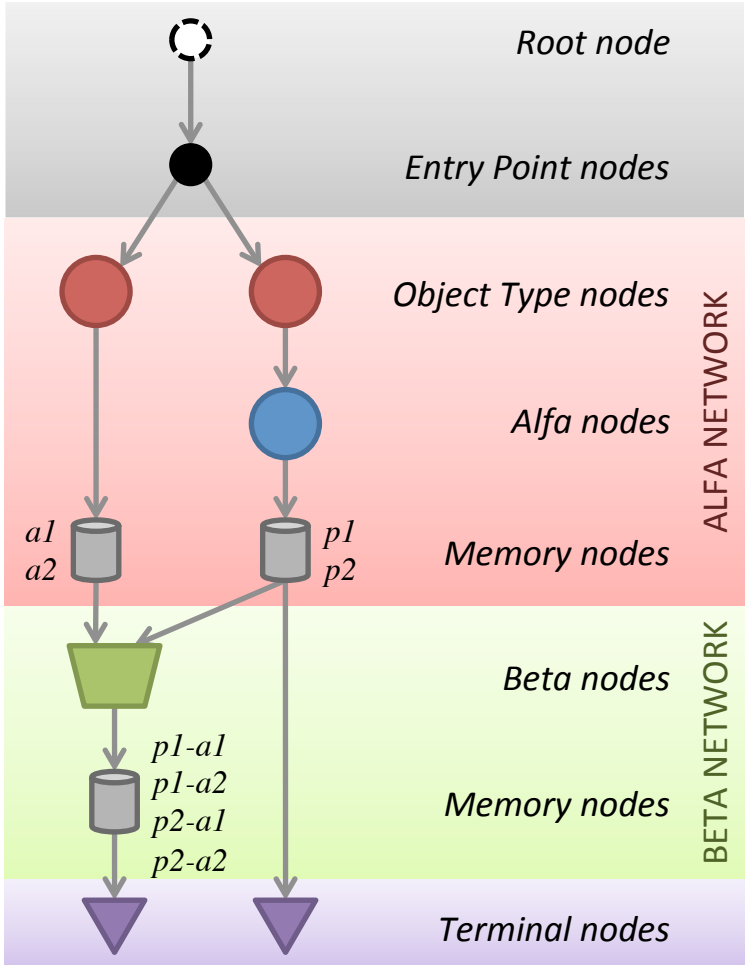
NB: Stampa semplificata e output prima regola omesso!

Pattern matching: l'algoritmo RETE

```
rule "Trova gli Stefano e indirizzi"
when
  $a: Address()
  $p: Person( name == "Stefano" )
then
  System.out.println($p+"/"+$a+" ");
end
```

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

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



NB: Stampa semplificata e output prima regola omesso!

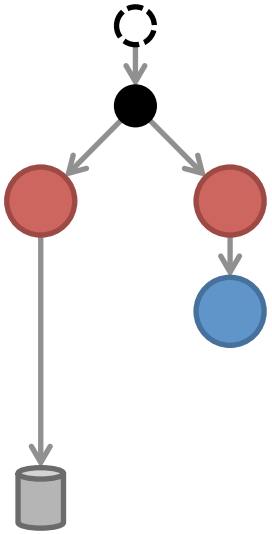
NB: WM = insieme delle memories (distribuita); oggetti ripetuti (ricondante): spazio vs tempo.

Pattern Matching: l'algoritmo RETE

3A REGOLA DI ESEMPIO

Pattern matching: l'algoritmo RETE

```
rule "Trova Stefano col suo indirizzo"
when
  $a: Address()
  $p: Person( name == "Stefano",
              address == $a )
then
  System.out.println($p);
end
```



Root node

Entry Point nodes

Object Type nodes

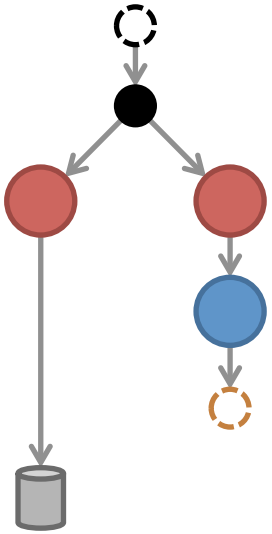
Alfa nodes

Dummy nodes

Memory nodes

Pattern matching: l'algoritmo RETE

```
rule "Trova Stefano col suo indirizzo"
when
  $a: Address()
  $p: Person( name == "Stefano",
              address == $a )
then
  System.out.println($p);
end
```



Root node

Entry Point nodes

Object Type nodes

Alfa nodes

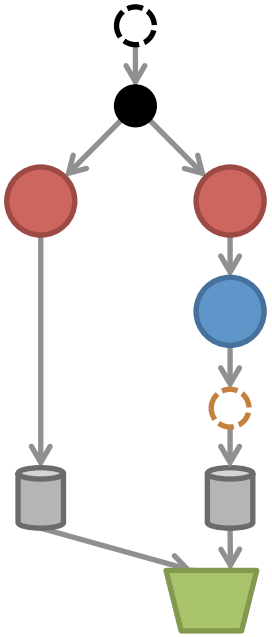
Dummy nodes

Memory nodes

NB: Questo Alfa node contiene un riferimento incrociato che non si può ancora risolvere.

Pattern matching: l'algoritmo RETE

```
rule "Trova Stefano col suo indirizzo"
when
  $a: Address()
  $p: Person( name == "Stefano",
              address == $a )
then
  System.out.println($p);
end
```



Root node

Entry Point nodes

Object Type nodes

Alfa nodes

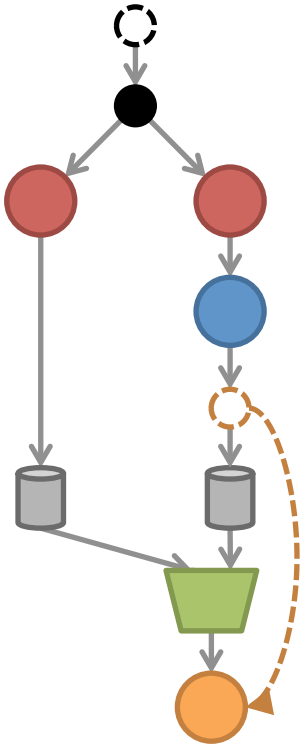
Dummy nodes

Memory nodes

Beta nodes

Pattern matching: l'algoritmo RETE

```
rule "Trova Stefano col suo indirizzo"
when
    $a: Address()
    $p: Person( name == "Stefano",
                address == $a )
then
    System.out.println($p);
end
```



Root node

Entry Point nodes

Object Type nodes

Alfa nodes

Dummy nodes

Memory nodes

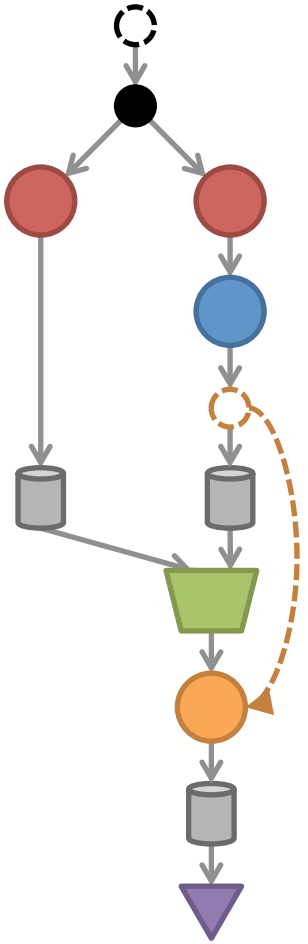
Beta nodes

Delayed nodes

NB: Il precedente Alfa node è inserito qui perchè può risolvere il riferimento incrociato.

Pattern matching: l'algoritmo RETE

```
rule "Trova Stefano col suo indirizzo"
when
    $a: Address()
    $p: Person( name == "Stefano",
                address == $a )
then
    System.out.println($p);
end
```



Root node

Entry Point nodes

Object Type nodes

Alfa nodes

Dummy nodes

Memory nodes

Beta nodes

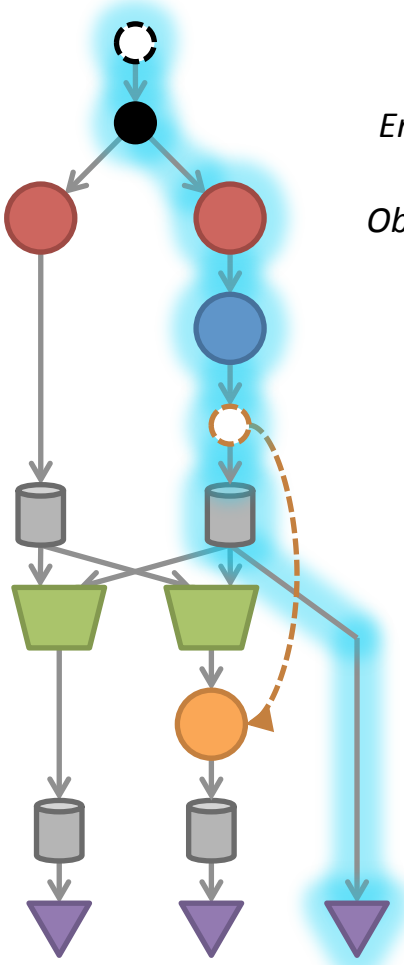
Delayed nodes

Memory nodes

Terminal nodes

Pattern matching: l'algoritmo RETE

```
rule "Trova Stefano col suo indirizzo"
when
    $a: Address()
    $p: Person( name == "Stefano",
                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

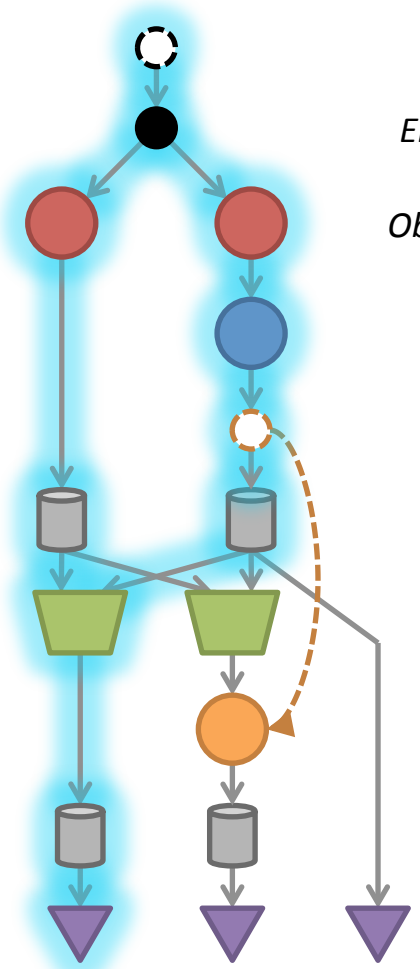
Memory nodes

Terminal nodes

NB: I nodi della RETE vengono condivisi quando possibile! Questa è la prima regola…

Pattern matching: l'algoritmo RETE

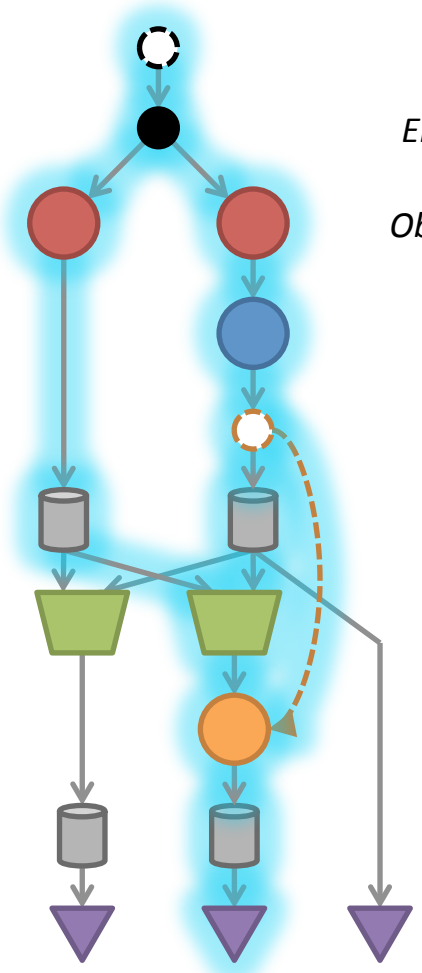
```
rule "Trova Stefano col suo indirizzo"
when
  $a: Address()
  $p: Person( name == "Stefano",
              address == $a )
then
  System.out.println($p);
end
```



NB: I nodi della RETE vengono condivisi quando possibile! Questa è la seconda…

Pattern matching: l'algoritmo RETE

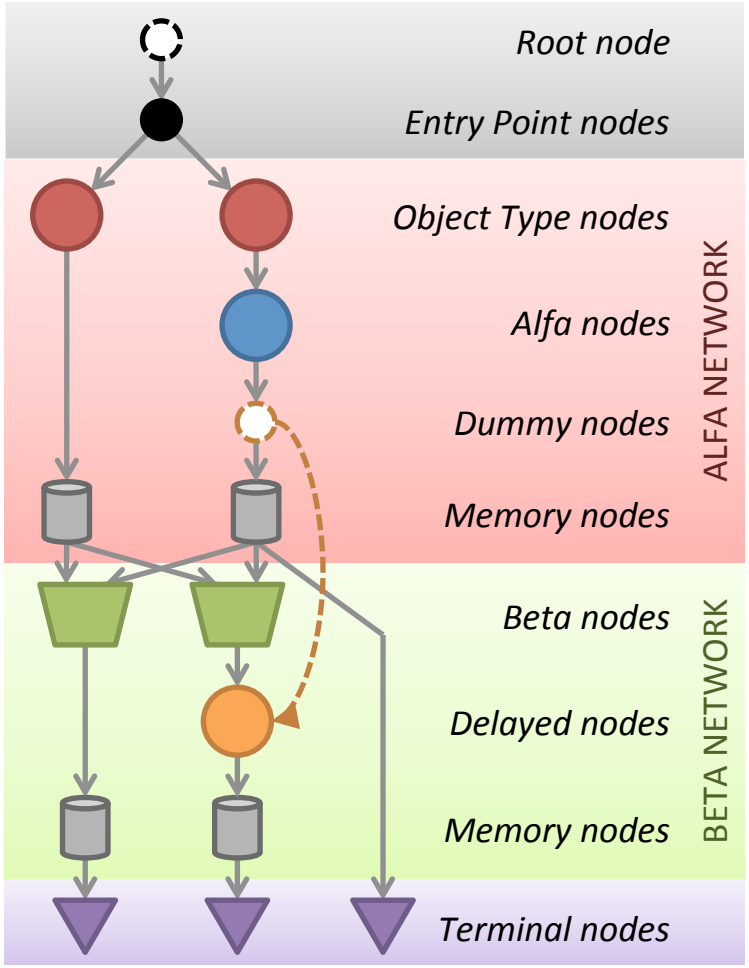
```
rule "Trova Stefano col suo indirizzo"
when
    $a: Address()
    $p: Person( name == "Stefano",
                address == $a )
then
    System.out.println($p);
end
```



NB: I nodi della RETE vengono condivisi quando possibile! Questa è l'ultima regola!

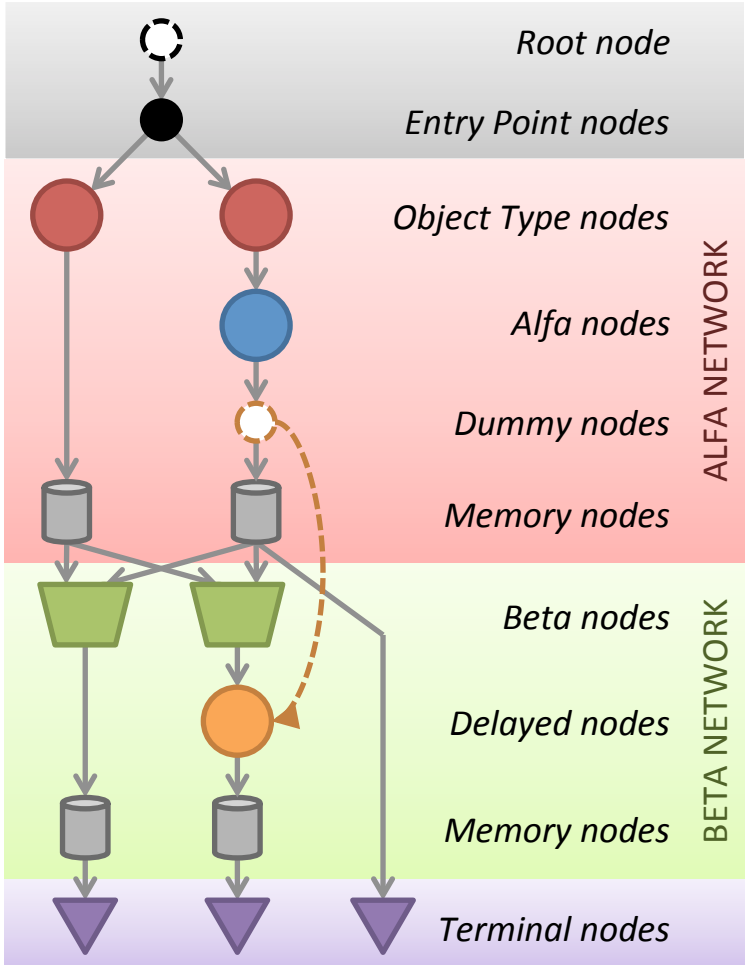
Pattern matching: l'algoritmo RETE

```
rule "Trova Stefano col suo indirizzo"
when
    $a: Address()
    $p: Person( name == "Stefano",
                address == $a )
then
    System.out.println($p);
end
```



Pattern matching: l'algoritmo RETE

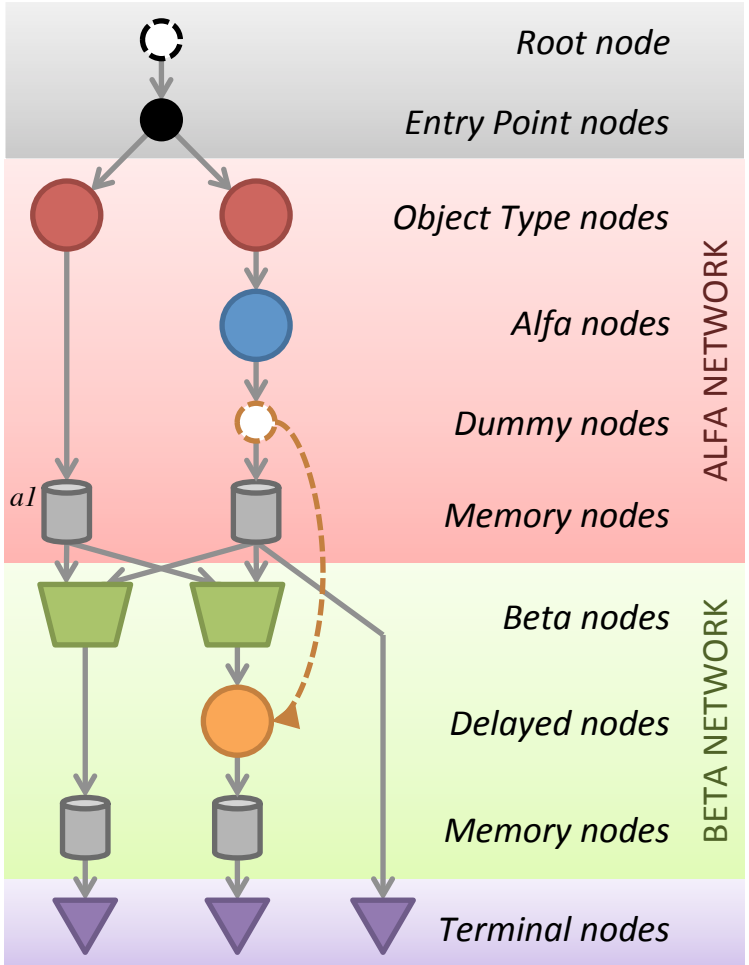
```
rule "Trova Stefano col suo indirizzo"
when
  $a: Address()
  $p: Person( name == "Stefano",
              address == $a )
then
  System.out.println($p);
end
```



Pattern matching: l'algoritmo RETE

```
rule "Trova Stefano col suo indirizzo"
when
    $a: Address()
    $p: Person( name == "Stefano",
                address == $a )
then
    System.out.println($p);
end
```

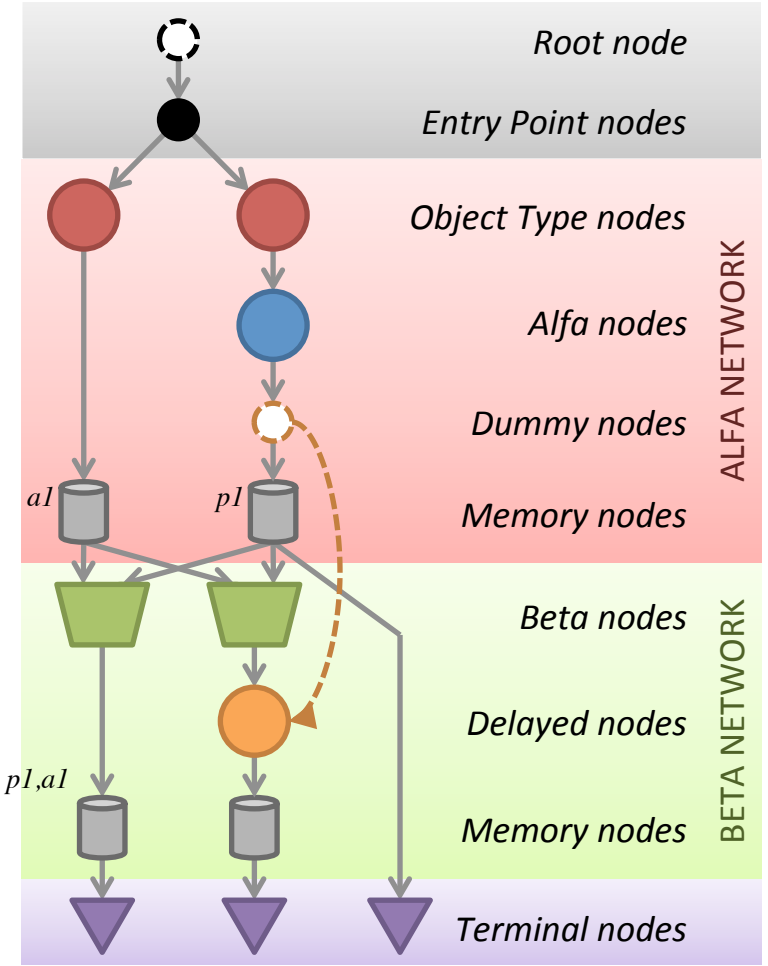
a1: Address("Via Po 2", 40068, "San Lazzaro")



Pattern matching: l'algoritmo RETE

```
rule "Trova Stefano col suo indirizzo"
when
    $a: Address()
    $p: Person( name == "Stefano",
                address == $a )
then
    System.out.println($p);
end
```

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

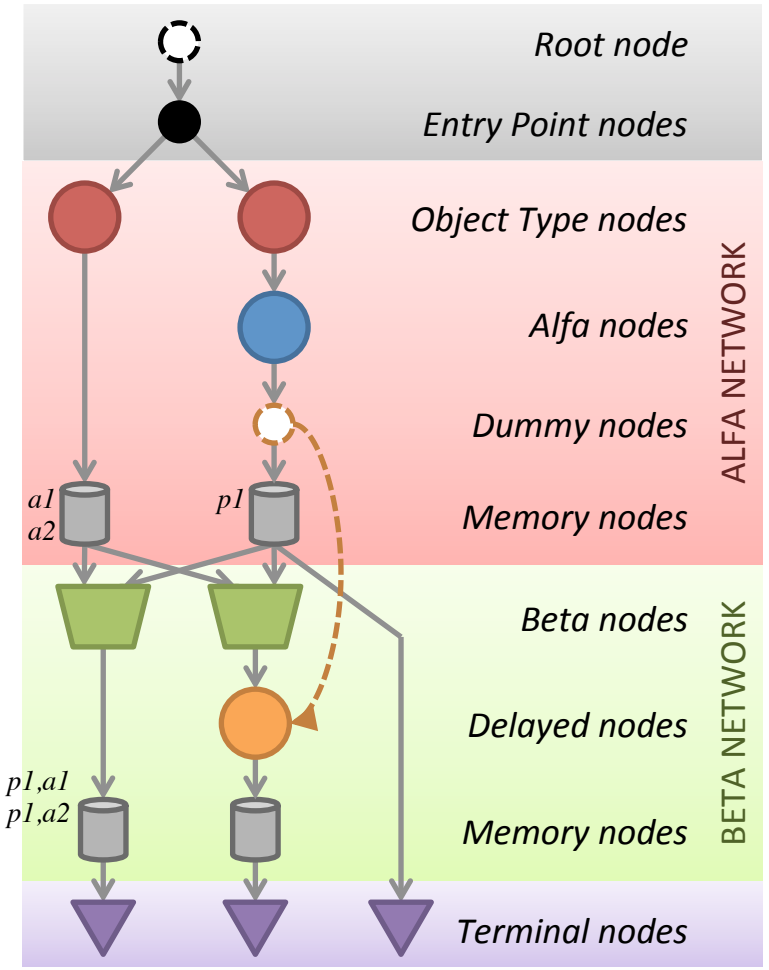


NB: L'output delle prime due regole è stato omesso!

Pattern matching: l'algoritmo RETE

```
rule "Trova Stefano col suo indirizzo"
when
    $a: Address()
    $p: Person( name == "Stefano",
                address == $a )
then
    System.out.println($p);
end
```

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

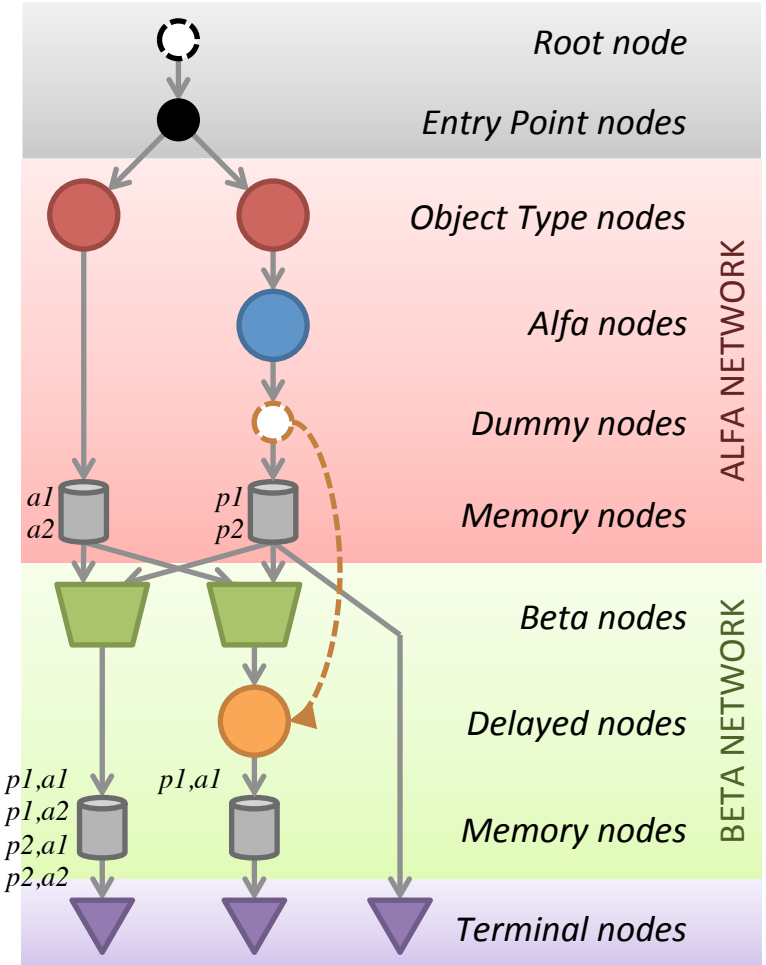


NB: L'output delle prime due regole è stato omesso!

Pattern matching: l'algoritmo RETE

```
rule "Trova Stefano col suo indirizzo"
when
    $a: Address()
    $p: Person( name == "Stefano",
                address == $a )
then
    System.out.println($p);
end
```

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




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

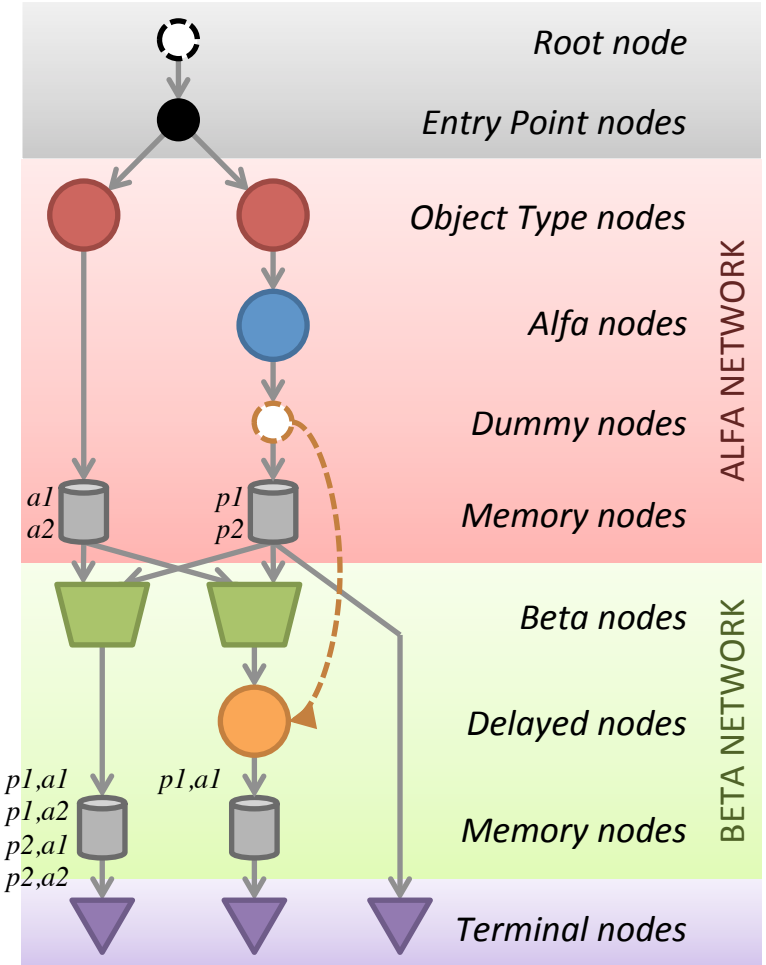
NB: L'output delle prime due regole è stato omesso!

Pattern matching: l'algoritmo RETE

```
rule "Trova Stefano col suo indirizzo"
when
    $a: Address()
    $p: Person( name == "Stefano",
                address == $a )
then
    System.out.println($p);
end
```

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

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

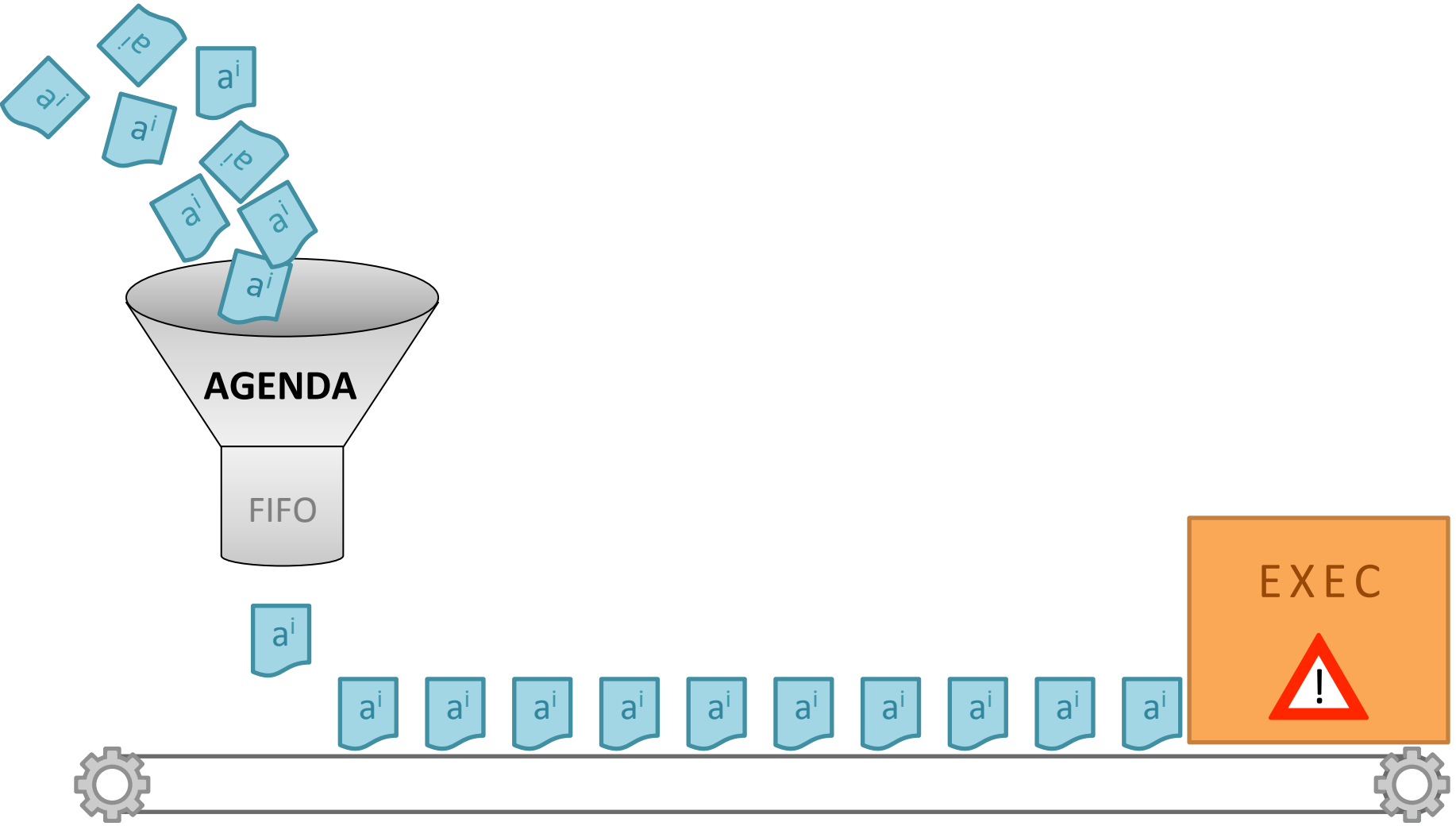


NB: L'output delle prime due regole è stato omesso!

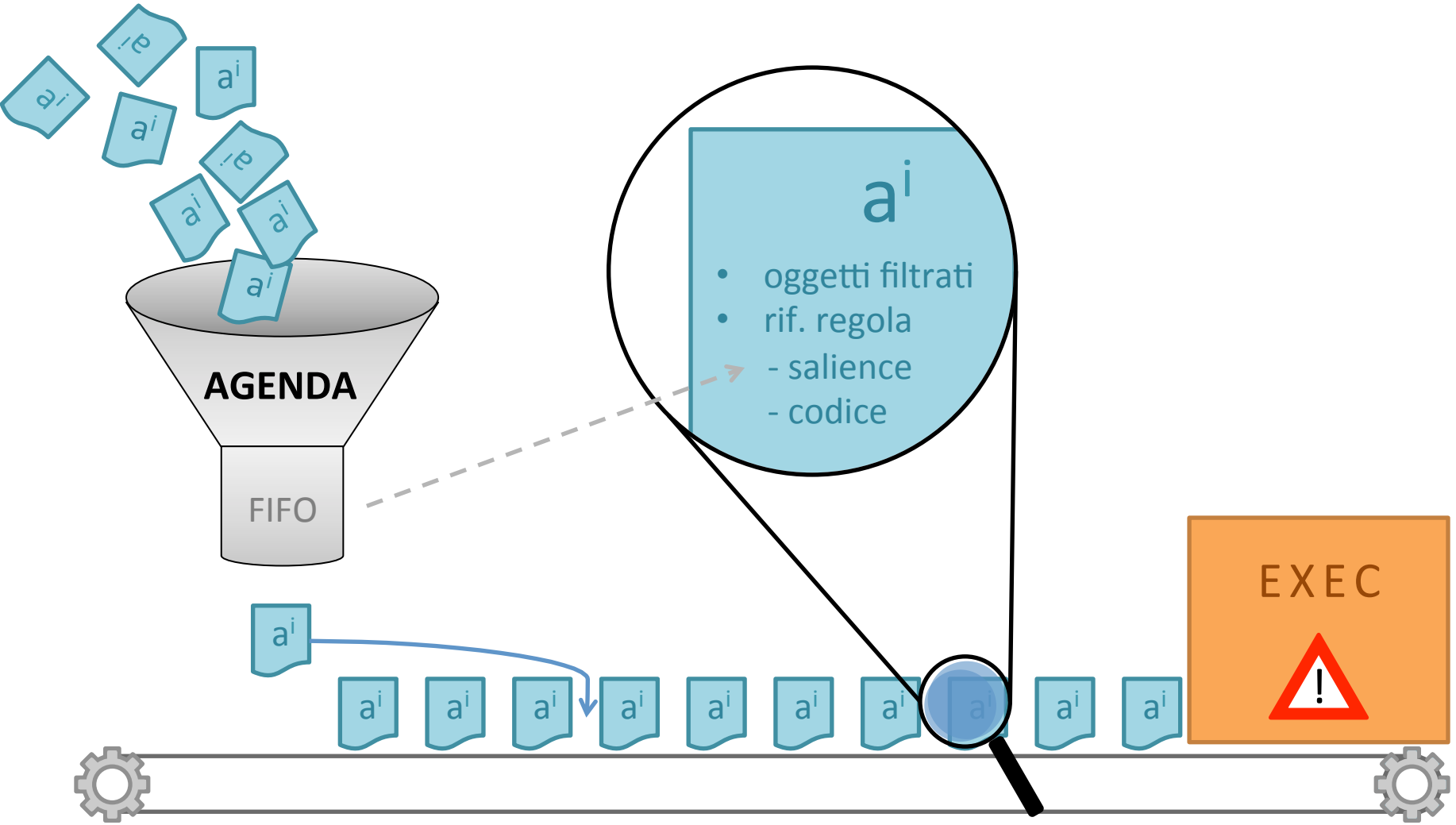
Fondamenti di Intelligenza Artificiale M

RISOLUZIONE DEI CONFLITTI ED ESECUZIONE

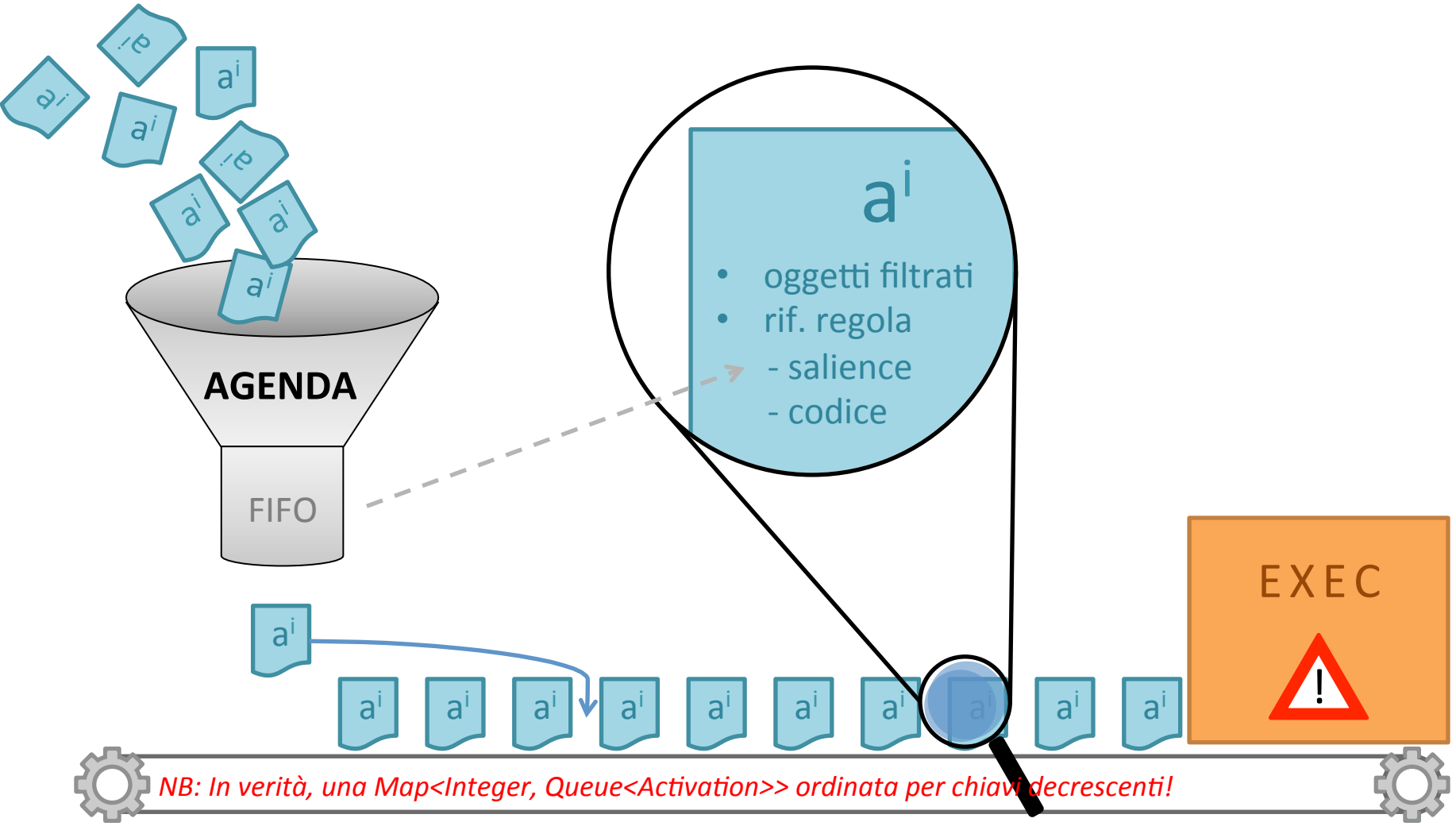
Risoluzione dei Conflitti & Esecuzione



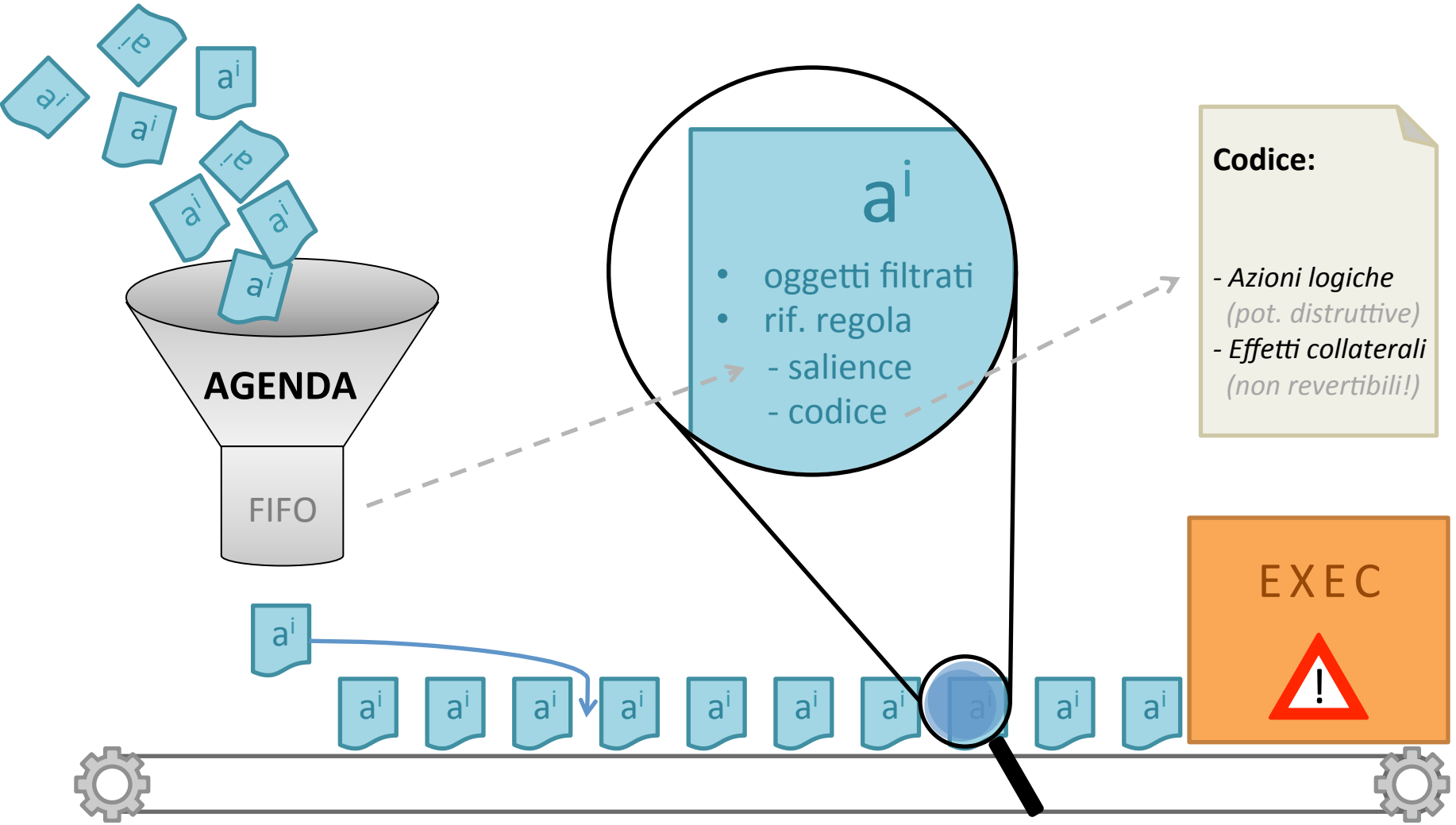
Risoluzione dei Conflitti & Esecuzione



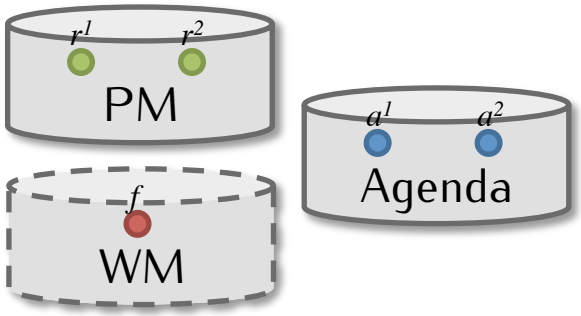
Risoluzione dei Conflitti & Esecuzione



Risoluzione dei Conflitti & Esecuzione



Risoluzione dei Conflitti & Esecuzione

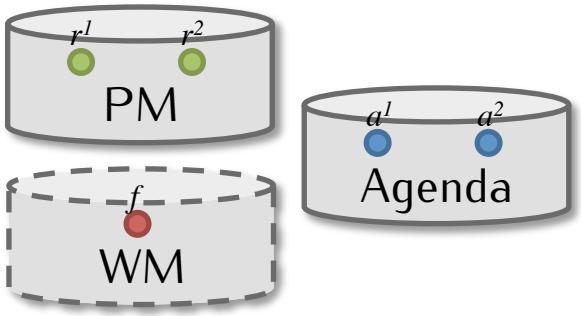



```
rule "r1"
when
  F()
then
  assert(new G());
end
```

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

?!?

Risoluzione dei Conflitti & Esecuzione

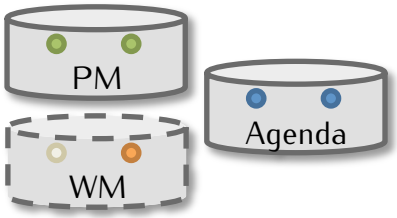



```
rule "r1"
when
  F()
then
  assert(new G());
end
```

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

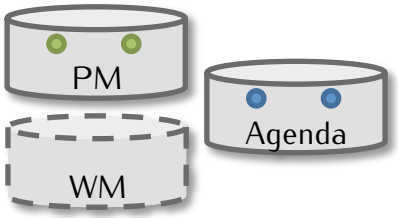
?!?

$a^1 < a^2$



Prima inserisco G,
poi elimino F.

$a^2 < a^1$



Prima elimino F,
a1 non sussiste più,
G mai asserito.

$r^1 < r^2$

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

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

Stabilisco un ordine
di precedenza tra
r1 e r2 (fisso).

Fondamenti di Intelligenza Artificiale M

RIFERIMENTI

Riferimenti

- **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

Fondamenti di Intelligenza Artificiale M

INFORMAZIONI

Informazioni

- Domani esercitazione in laboratorio
- Possibilità di svolgere attività progettuali o tesi
  - *Drools, Event Calculus, Expectations, MS-Kinect, Android SDK, SOA/Cloud, …*
- Per domande, dubbi, richieste:

stefano.bragaglia@unibo.it