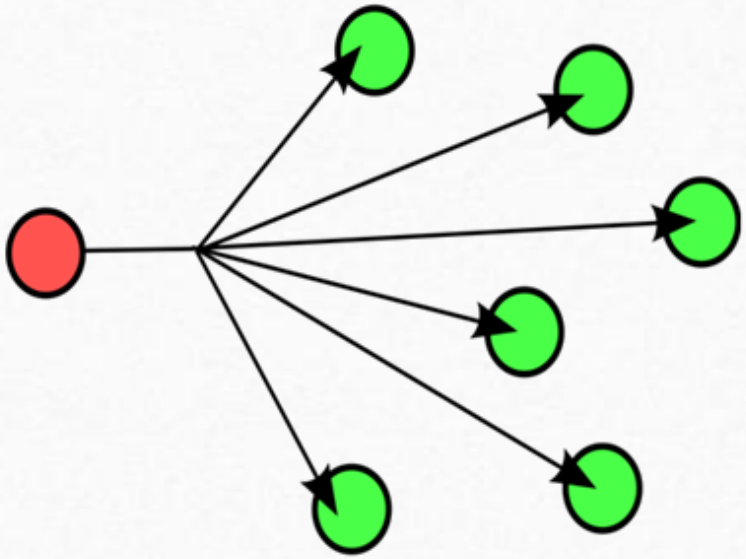
Il Consenso in Reti Asincrone

Gestione del fallimento dei processi

Il Sistema Broadcast



In telecomunicazioni per **broadcasting** si intende una tecnica di invio di informazioni da un sistema trasmittente ad un insieme di sistemi riceventi, collegati alla rete, non definito a priori.

Il Sistema Broadcast

La trasmissione broadcasting è **unidirezionale**. Le informazioni sono inviate dal trasmettitore ai ricevitori, **senza canale di ritorno** e **senza sicurezza** che le stesse riescano ad essere consegnate.

Infatti se un processo manda in broadcast un messaggio e incorre in un guasto, la primitiva di invio offerta dalla rete non garantisce che tutti i processi a cui il messaggio era destinato lo ricevano.

Quindi se dei guasti incorrono durante un broadcast, è plausibile che solo un sottoinsieme dei processi consegnino il messaggio precedentemente inviato.

Confronto Broadcast con Send/Receive

TEOREMA 21.1

Se **A** è un sistema di trasmissione asincrona con un canale di trasmissione broadcast affidabile, allora esiste un sistema di *invio/ricezione* **B** asincrono con canali di *invio/ricezione* FIFO affidabili con la stessa interfaccia utente di A e che "simula" A, come segue.

Per ogni esecuzione **α** di **B**, esiste un'esecuzione **α'** di **A** tale che valgano le seguenti condizioni:

- **α** e **α'** sono indistinguibili da U (dove ogni $U_i \in U$ che è l'insieme degli utenti)
- Per ogni *i*, un $stop_i$ si verifica in **α** esattamente se lo fa in **α'**.

Inoltre, se **α** è fair, allora **α'** è anche fair.

Il Problema del Consenso

Nel problema del consenso un gruppo di processi deve mettersi d'accordo su un valore comune.
Ogni processo parte con un suo valore iniziale (arbitrario) in input e ci si aspetta in output un valore comune a tutti.

Il consenso è definito in termini di due primitive **init(v)$_i$** e **decide(v)$_i$**, con $v \in V$ e $1 \leq i \leq n$.

Quando un processo esegue *init(v)$_i$*, significa che il processo propone il valore v. Similarmente quando un processo esegue *decide(v)$_i$* , significa che il processo decide il valore v.

Una sequenza di azioni *init$_i$* e *decide$_i$* viene definita **well-formed** se preso un qualsiasi prefisso di tale sequenza questo è della forma *init$_i$(v)*, *decide$_i$(v)*.

Proprietà del Consenso

- **Agreement**: Se un processo corretto decide *v*, allora tutti i processi corretti alla fine decidono *v*.
- **Termination**: Tutti i processi che non falliscono, prima o poi decidono.
- **Integrity**: Ogni processo decide al più una volta, e se decide per *v* allora qualche processo ha proposto *v*.
- **Validity**: Se tutti i processi corretti che propongono un valore, propongono *v*, allora tutti i processi corretti alla fine decidono *v*, che è l'unico valore decisionale possibile.

Proprietà del Consenso

- **Well-formedness:** In ogni esecuzione, e per ogni *i*, l'interazione tra U_i e A deve essere **well-formed** per *i*.

- **F-failure termination, $0 \leq f \leq n$**: In ogni esecuzione fair con eventi *init* su tutte le porte, se si verificano eventi di *stop* su un massimo di *f* porte, dopo un intervallo indeterminato ma finito di tempo, si verifica l'evento di *decide* sulle porte che non sono fallite.

Risultato di Impossibilità

TEOREMA 21.2

"Non esiste un algoritmo in un modello di broadcast asincrona con canale broadcast affidabile che risolva il problema del consenso e garantisca la terminazione anche con un solo processo fallito."

Risultato di Impossibilità

TEOREMA 21.2

L'impossibilità da parte dei processi corretti di determinare con accuratezza quali processi sono realmente guasti e quali invece solo estremamente lenti, è il cuore dell'impossibilità di raggiungere il consenso tra i processi in un sistema asincrono.

La prova si basa su:

- il teorema di impossibilità per il modello a memoria condivisa
- l'esistenza di una trasformazione da sistemi broadcast a sistemi asincroni con memoria condivisa.

Possibili Soluzioni al Problema del Consenso

- Failure Detectors
- Randomized Algorithm
- K-Agreement
- Approximate Agreement
- Paxos

Failure Detector

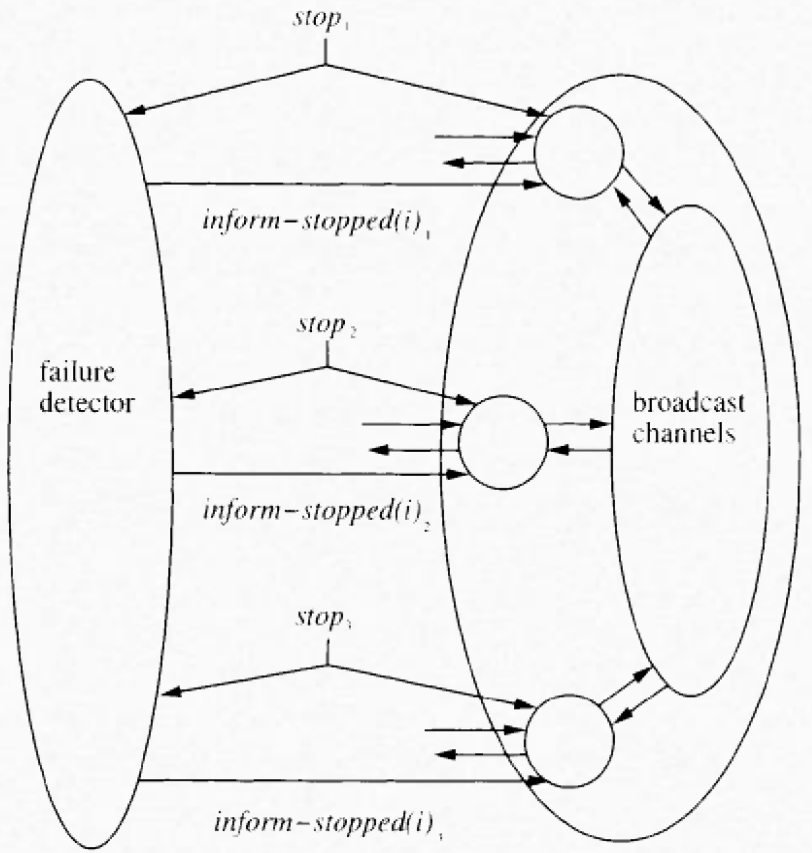
Il Failure Detector è un modulo che provvede a segnalare ai processi in una rete asincrona gli eventuali fallimenti avvenuti fino a quel momento.

In tal modo è possibile distinguere i sistemi falliti da quelli lenti.

Tale funzione prende il nome di *inform-stopped(j)* $_i$ (per ogni coppia i e j con i ≠ j) reperibile nell'accesso ad un modulo locale FD_i.

Esistono diversi tipi di *failures detectors* in base alle informazioni di arresto:

- *Completa*: capacità di identificare i processi guasti presenti nel sistema
- *Corretta*: capacità di identificare i processi corretti nel sistema e quindi non sospettare di guasto



Algoritmo di Randomizzazione

Il problema del consenso può essere risolto in una rete asincrona **randomizzata**. Questo modello è più forte del normale modello di rete asincrona, perché consente ai processi di effettuare **scelte casuali** durante l'elaborazione.

Le condizioni di correttezza sono leggermente più deboli rispetto alle condizioni nel modello di rete asincrona ordinario: tutte le condizioni sono garantite tranne la condizione di **terminazione** che ora è **probabilistica**.

Tutti i processori corretti decideranno entro un tempo *t,* dopo l'arrivo di tutti gli input, con probabilità di almeno *p(t)*, dove *p* è una particolare funzione monotona non decrescente illimitata. Questo implica l'eventuale terminazione con probabilità 1.

Algoritmo di Ben-Or

L'algoritmo vale con

- $n > 3f$
- $V = \{0, 1\}$

Teorema 21.7

L'algoritmo di Ben-Or garantisce la **well-formedness**, l'**agreement** e la **validity.**

Inoltre garantisce, con probabilità 1, che tutti i processi "corretti" effettueranno l'evento *decide*(v).

```
1  procedure BenOr(v) //v is the init(v) of process p
2
3      x ← v
4      s ← 1
5
6      while true do
7
8          send ("first", s, v) to all processes //where v is its current value of x
9
10         wait for messages of the form ("first", s, *) from n − f processes //"*" can
           be 0 or 1
11         if received all ("first", s, v) with the same v
12              y=v
13         else y=null
14
15         send ("second", s, v) to all processes //where v is its current value of y
16
17         wait for messages of the form ("second", s, *) from n − f processes //"*"
           can be 0, 1 or null
18
19         if all messages propose the same value v!=null then
20             x=v and decide(v)
21
22         else if n−2f messages propose the same value v!=null then // the assumption
           that n>3f implies that there can't be two different such values v
23             x=v
24
25         else choose v randomly, with P r[v = 0] = P r[v = 1] = 1/2
26
27         s ← s + 1 // s is the current phase number
```

Validità e Accordo di Ben-Or

Validity: supponiamo che tutti i processi facciano *init(v)* con il medesimo valore *v*. Tutti i messaggi inviati nel primo e nel secondo round della prima fase avranno la forma rispettivamente di ("first", s, v) e ("second", s , v) con il medesimo valore *v*, che quindi sarà l'unico valore inviato.

Agreement: supponiamo che P_i esegua una *decide(v)* alla fase s e non ci siano altri processi che decidano prima di lui; questo implica che P_i avrà ricevuto *n-f* ("second", s, v) messaggi. Pertanto qualsiasi altro processo P_j che completerà la fase s avrà sicuramente ricevuto almeno *n-2f* ("second", s, v) poichè potrebbero esserci al massimo f fallimenti. P_j non potrà eseguire una decide con valori differenti da v e setterà $x=v$.

Questo è vero per tutti i processi Pj che completano la fase s, quindi tutti i processi che inizieranno la fase s+1 inizieranno con $x=v$, pertanto tutti i processi corretti decideranno v alla fine dello stage s+1.

Terminazione in Ben-Or

Termination (Lemma)

Per ogni $s \geq 0$, tutti i processi corretti decideranno entro s+1 fasi, con probabilità $\geq \mathbf{1-(1-½^n)^s}$

Dimostrazione

Il caso con $\boldsymbol{s=0}$ è banale, perché la probabilità che tutti i processi corretti decidano entro la prima fase ha una probabilità certamente maggiore di 0.

Consideriamo il caso in cui $\boldsymbol{s \geq 1}$.

La prova della tesi si basa sulla dimostrazione che in ogni fase $s > 0$, la probabilità, che tutti i processi che non falliscono scelgano, alla fine della fase s, lo stesso valore di x, è uguale a $\mathbf{½^n}$.

Terminazione in Ben-Or

Consideriamo un frammento finito α in cui un processo corretto P_i riceva n - f (“first”, s, *) messaggi

Se almeno f +1 messaggi contengono il valore *v* definiremo *v* come valore “buono”.

Sono possibili due casi:

1) il valore "buono" è unico
2) esistono due valori "buoni"

Terminazione in Ben-Or

CASO CON UN SOLO VALORE "BUONO"

Se nei messaggi ricevuti è presente un solo valore "buono", allora ogni messaggio inviato ("second", s, v) in ogni estensione di α contiene o v o *null*.

In questo caso ci sono alcuni processi che deterministicamente impostano x a v.

I restanti processi devono scegliere il valore a random.

La probabilità che essi scelgano lo stesso valore v è $\geq \frac{1}{2^n}$.

Terminazione in Ben-Or

CASO CON DUE VALORI "BUONI"

In questo caso, i processi corretti riceveranno sia il valore 0 che il valore 1 all'interno del messaggio "first", pertanto nel Round 2 il messaggio inviato nella forma ("second", s, v) avrà il valore *v* uguale a *null*.

Poichè il valore di *v* è null, il valore di *x*, per ogni processo, verrà scelto in modo casuale.

La probabilità che tutti i processi facciano la stessa scelta casuale, concordando sullo stesso valore *v* è $\geq \frac{1}{2^n}$.

Terminazione in Ben-Or

CONCLUSIONE

Abbiamo visto che in entrambi i casi la probabilità che venga scelta v è $\geq \frac{1}{2^n}$.

Quindi la probabilità che non si sia deciso un unico valore nella fase s è:

$$\mathbf{pr(\text{nessuna scelta unica in s}) \leq 1-\frac{1}{2^n}}$$

Dal momento che le scelte nelle varie fasi sono indipendenti tra loro avremo che:

$$\mathbf{pr(\text{nessuna scelta unica nelle prime s fasi}) \leq (1-\frac{1}{2^n})^s}$$

Quindi la probabilità che tutti i processi corretti decidano lo stesso valore v in una fase $\mathbf{s' \leq s}$ è:

$$\mathbf{pr(\text{unico valore in una fase s}' \leq \text{s}) \leq 1-(1-\frac{1}{2^n})^s} .$$

Algoritmo di Ben-Or

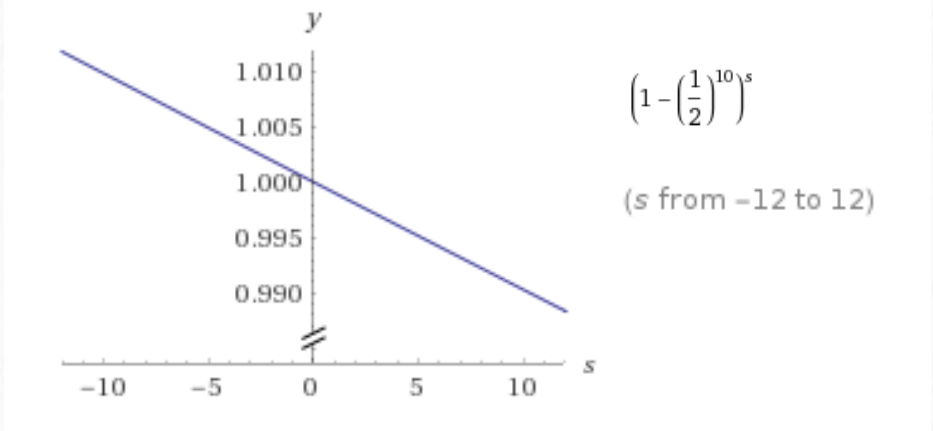
TEOREMA 21.7

L'algoritmo di Ben-Or garantisce la **well-formedness**, l'**agreement** e la **validity.**

Inoltre garantisce, con probabilità **1**, che tutti i processi "corretti" effettueranno l'evento *decide(v)*.

Questo deriva dal fatto che all'aumentare del numero di fasi ***s***, $\mathbf{(1-½^n)^s} \rightarrow 0$

pertanto $\mathbf{1-(1-½^n)^s} \rightarrow 1$, ottenendo così pr=1.



Complessità dell'Algoritmo di Ben-Or

Ogni processo corretto completa ogni fase in $O(s(d+l))$ dopo l'ultima *init(v)*.

Dove

- *s* rappresenta una fase
- *d* rappresenta il limite massimo di tempo per il trasferimento del messaggio tra processi
- *l* rappresenta il limite massimo di tempo per svolgere le attività del processo

K-Agreement

Il problema del k-agreement è una variazione del problema del consenso che nelle reti asincrone può essere risolto con un numero limitato di fallimenti ($f < k$).

L'**Agreement** risulta essere più debole rispetto al consenso, poiché permette k valori decisionali invece di uno solo.

La **Validity** risulta essere più forte di quella del consenso su un solo valore, poiché ci possono essere k valori decisionali e sicuramente quello che riceve è un valore dato in input da un altro processo.

TrivialKAgreement algorithm:

I P1, P2, … Pk processi trasmettono il loro iniziale valore. Ogni processo Pi decide in merito al primo valore che riceve.

K-Agreement

SVANTAGGI

Teorema 21.9: l'algoritmo sopra scritto risolve il k-problema di consenso e garantisce f-fallimenti terminali, se $f < k$.

Il problema del K-agreement non può essere risolto se il numero di fallimenti $f \geq k$.

Teorema 21.10: Il problema dei k-agreement non si può risolvere con k-fallimenti totali nel modello di trasmissione asincrona.

Approximate Agreement

Invece di dover essere esattamente d'accordo, come nel problema del consenso, viene richiesto che i processi siano d'accordo con una tolleranza positiva ***e***.

Agreement: in qualsiasi esecuzione, i due valori di decisione sono nell'intorno di ***e*** rispetto agli altri valori.
Validity: in qualsiasi esecuzione, qualsiasi valore di decisione rientra nell'intervallo dei valori iniziali V (insieme di numeri reali mandabili all'interno dei messaggi)

Teorema 21.12 Il problema approssimativo del consenso non è risolvibile con f-failure termination nel modello di trasmissione asincrono se $n < 2f$.

Paxos - Parlamento Part-Time

- Parlamento part-time
- Ciascun membro aveva un registro in cui annotare tutti i decreti approvati
- I registri di due parlamentari non dovevano contenere informazioni contraddittorie
- I decreti approvati erano numerati (in ordine crescente)
- I parlamentari, così come i messaggeri, potevano entrare ed uscire dal parlamento a piacere
- I messaggeri potevano anche uscire prima di consegnare un messaggio affidatogli, "*magari per un viaggio di sei mesi*" o "*andar via per sempre e il messaggio non veniva consegnato*"
- Quando in Camera, parlamentari e messaggeri erano dediti al lavoro ed eseguivano prontamente i loro compiti
- C'era molto rispetto e fiducia tanto che si tendeva a far passare ogni decreto proposto

Per garantire progresso occorreva che almeno la metà dei legislatori fosse in Camera per un tempo sufficientemente lungo.

Confronto tra Storia e Algoritmo

Parlamento	→	Sistema Distribuito
Parlamentare	→	Processo
Uscita/Entrata dalla Camera	→	Fallimento per Crash/Recovery
Registro	→	Memoria Stabile
Messaggeri	→	Comunicazione asincrona tra i processi

Descrizione

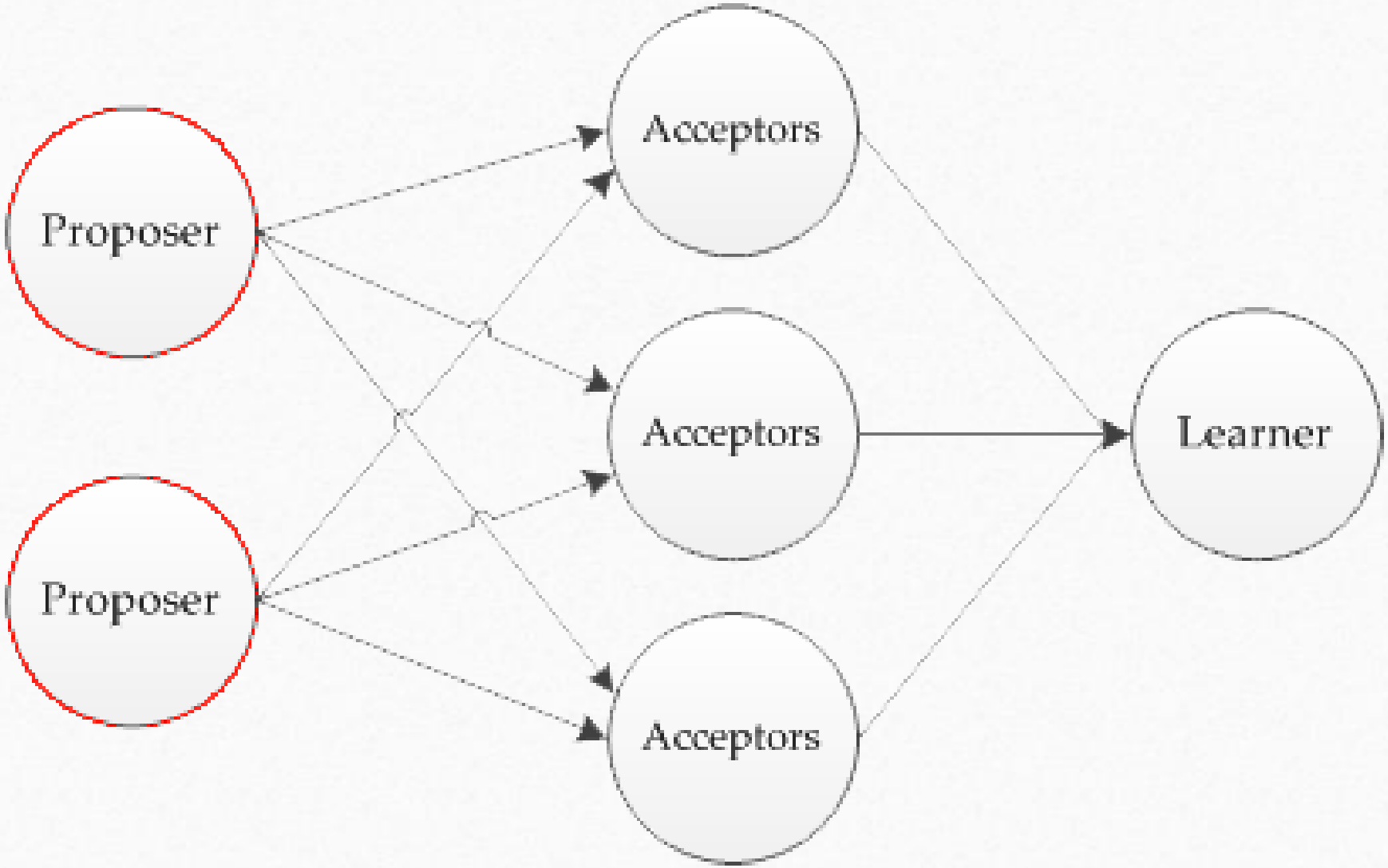
- Un algoritmo distribuito per risolvere il problema del consenso
- Garantisce le proprietà di
  - Liveness
  - Safety
- Non garantisce la proprietà di Terminazione
- Si compone da nodi che possono avere uno o tre ruoli
- Garantisce che i nodi sceglieranno sempre e solo un singolo valore

Proprietà

- Garantisce la proprietà di **liveness**
  - Uno tra i valori proposti viene prima o poi scelto.
  - Se un valore viene scelto, ogni processo prima o poi saprà di tale scelta.

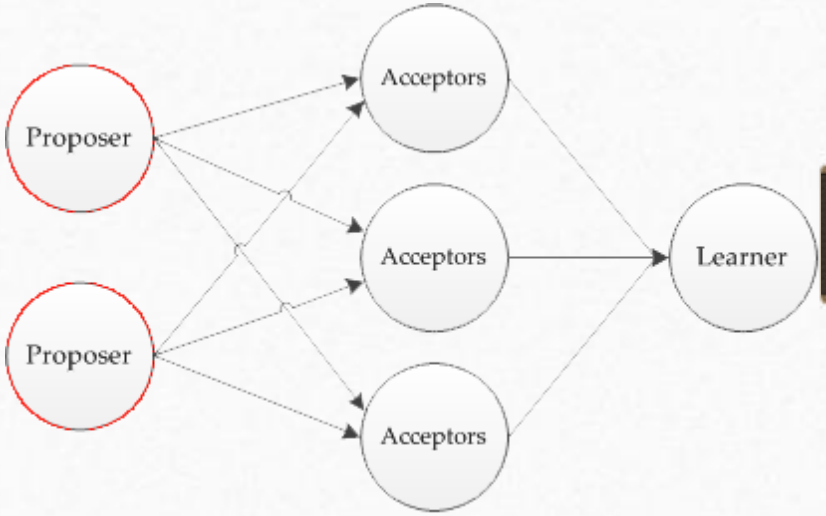
- Garantisce la proprietà di **safety**
  - Può essere scelto un unico valore tra quelli proposti
  - Un processo non saprà mai che un valore è stato scelto a meno che esso non sia stato effettivamente scelto.

- Non garantisce la proprietà di **terminazione**
  - Non garantisce che un valore finale verrà scelto se la maggioranza dei nodi non sono disponibili.


Proposer
Proposer
Acceptors
Acceptors
Acceptors
Learner

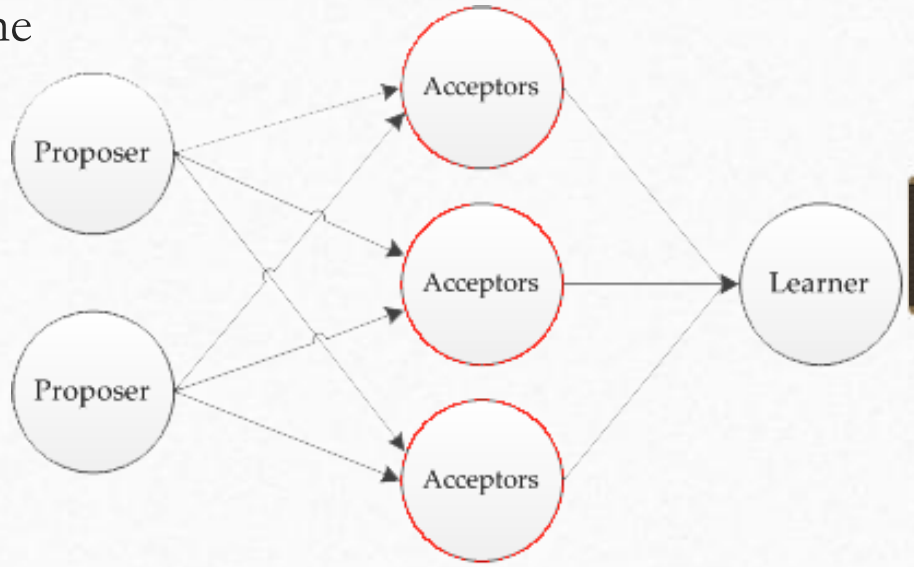
Ruoli dei Nodi - Proposer

- **Proposer**: propone un valore sul quale vuole che ci si metta d'accordo e lo invia a tutti gli acceptor
- un proposer deve conoscere:
  - il valore della proposta con numero seriale più alto minore di n accettato da ogni acceptor in una maggioranza
  - il valore della proposta con numero seriale più alto minore di n che sarà accettato da ogni acceptor in una maggioranza
- il processo *proposer* invia una **propose,** un messaggio con numero seriale *n'*.
- il processo *proposer* invia una **accept,** chiedendo che la sua richiesta sia accettata



Ruoli dei Nodi - Acceptor

- **Acceptor**: decide se accettare il valore, può ricevere più proposte ognuna da un proposer diverso e dopo la decisione trasmette la sua decisione ai learners
- Gli acceptor possono ricevere due tipi di messaggi: *prepare* e *accept*.
- Un acceptor può sempre rispondere ad una richiesta di tipo *prepare*.
- Un acceptor può rispondere ad una richiesta di tipo *accept*, accettando la proposta, solo se non ha promesso a qualche processo di non farlo.
- Un acceptor può accettare una proposta con numero seriale n se solo se non ha risposto ad una richiesta *prepare* il cui numero è maggiore di n.



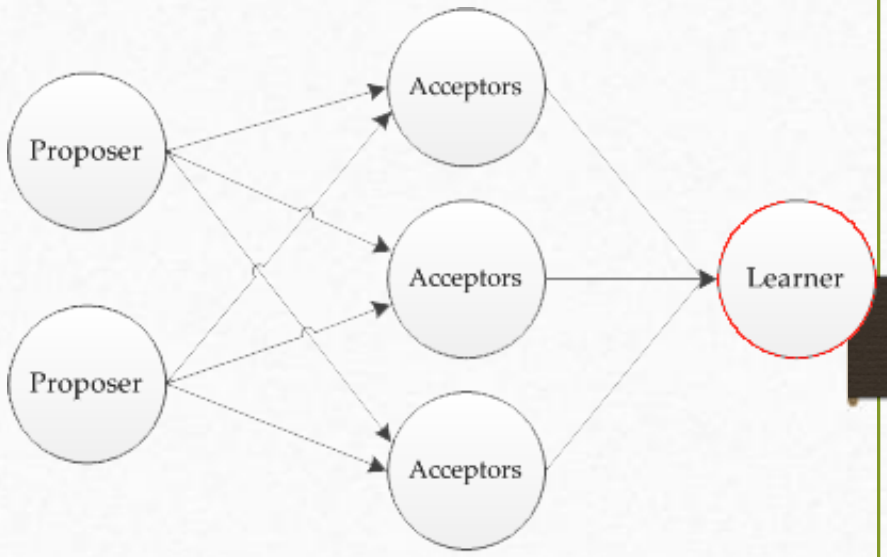
Ruoli dei Nodi - Learner

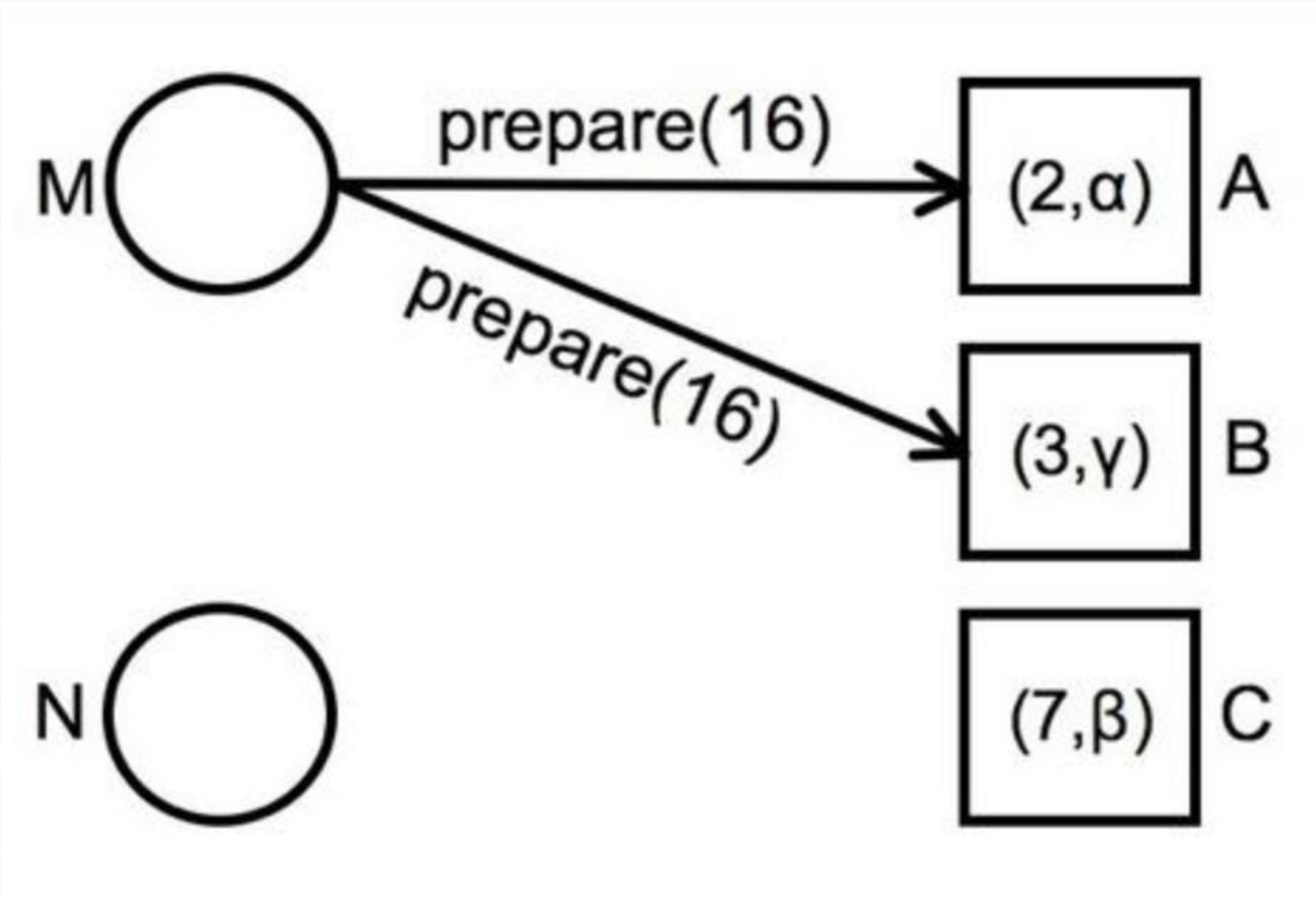
Learner: determinano se un valore è stato accettato.

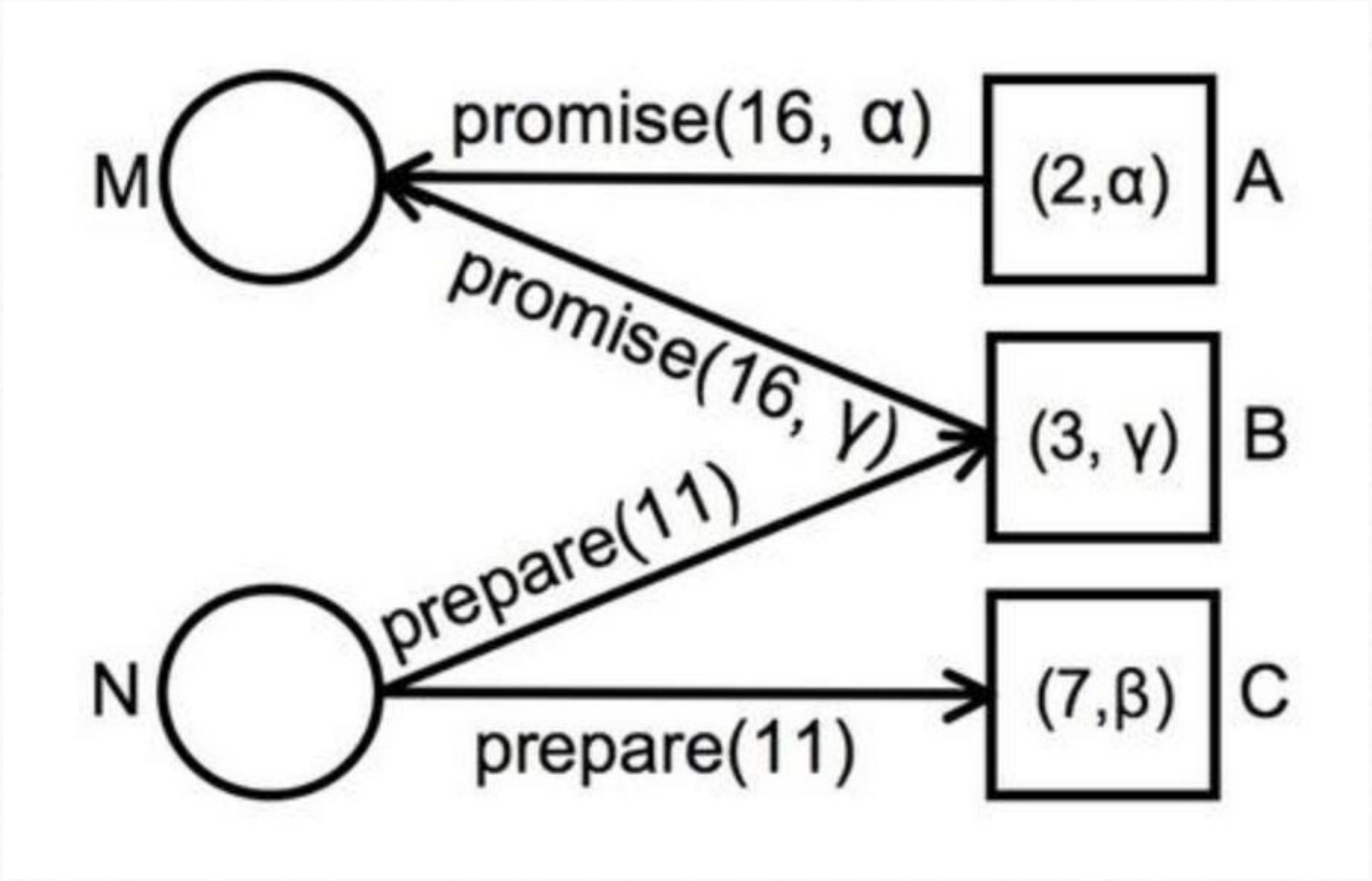
Vengono informati dagli Acceptors di ogni accettazione avvenuta con successo.

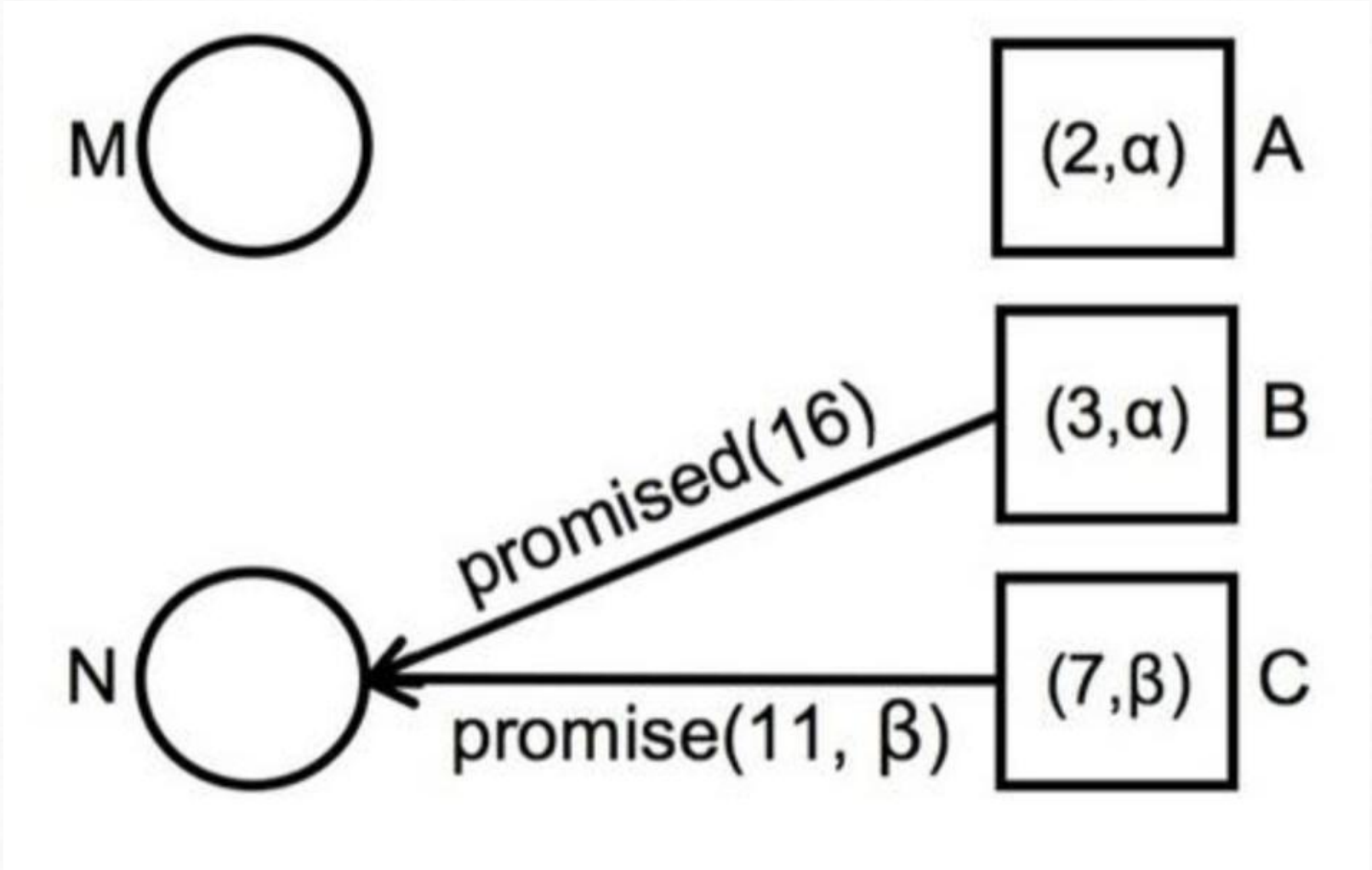
Durante la fase dell'apprendimento gli acceptor possono adottare diverse strategie:

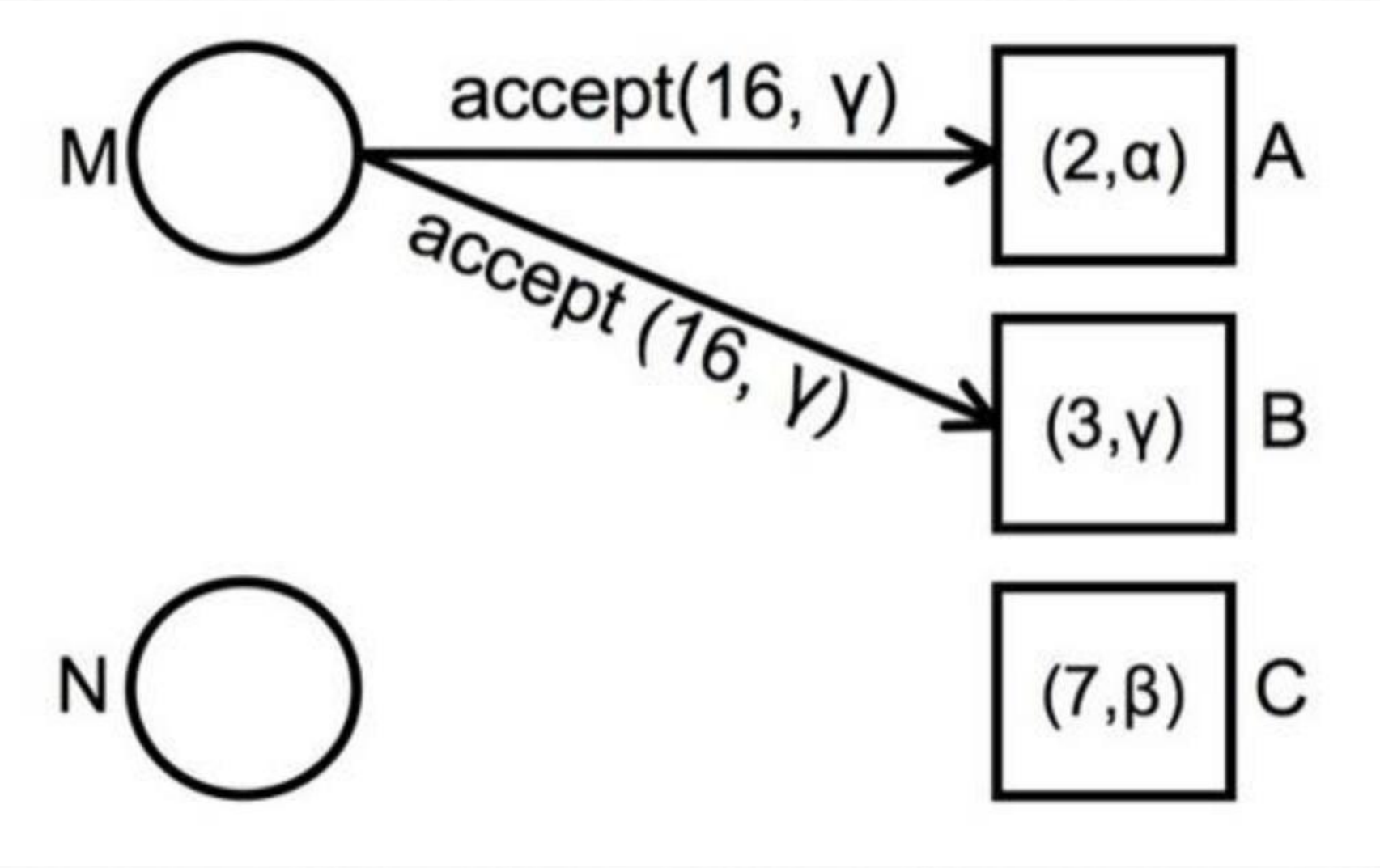
1. Ogni *acceptor* informa tutti i *learner* circa l'accettazione di un valore.
2. Gli *acceptor* informano un solo *learner distinto*, che poi informa tutti gli altri *learner*.
3. Gli *acceptor* informano un sottoinsieme di *learner* che poi provvedono ad informare gli altri *learner*.

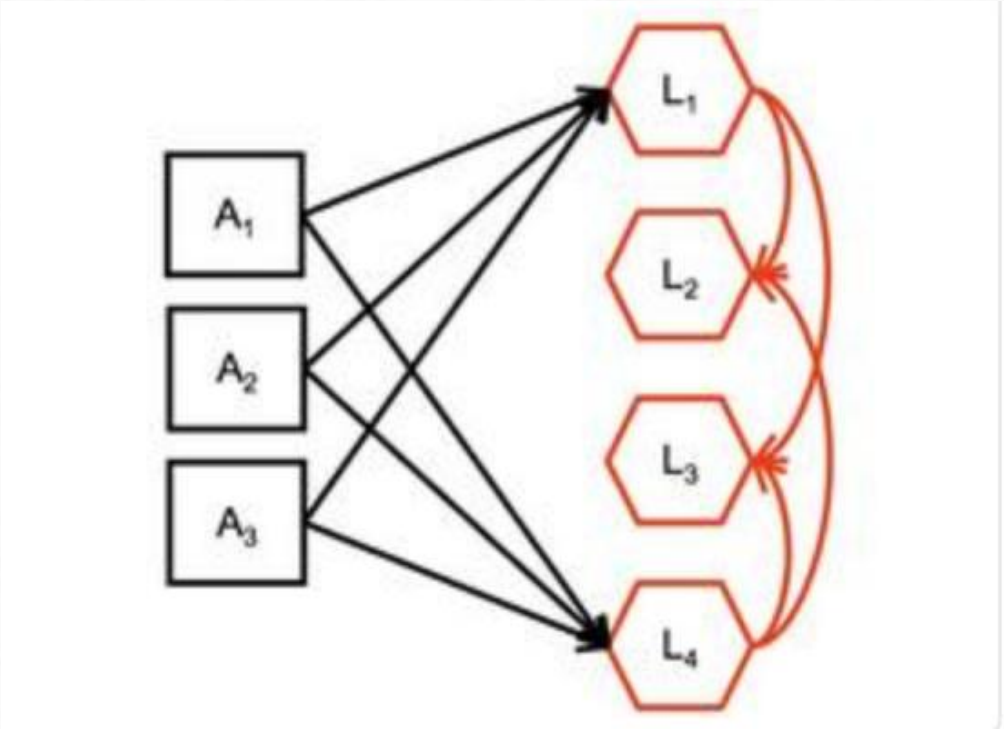



M
prepare(16)
(2,α)
A
prepare(16)
(3,γ)
B
N
(7,β)
C


M
promise(16, α)
(2,α)
A
promise(16, γ)
(3, γ)
B
prepare(11)
N
(7,β)
C
prepare(11)


M
N
(2,α)
A
(3,α)
B
(7,β)
C
promised(16)
promise(11, β)


M
accept(16, γ)
accept (16, γ)
(2,α)
A
(3,γ)
B
N
(7,β)
C


A1
A2
A3
L1
L2
L3
L4

GRAZIE PER L'ATTENZIONE

Arnone Marzia

Falco Erica

Cucinelli Gabriele

Martinero Gabriele