Impossibilità del consenso nel modello a rete asincrono: una soluzione randomizzata

Seminario 8

Daniele Liberatore, Alberto Mulone,
Matteo Palazzo, Stefano Porta

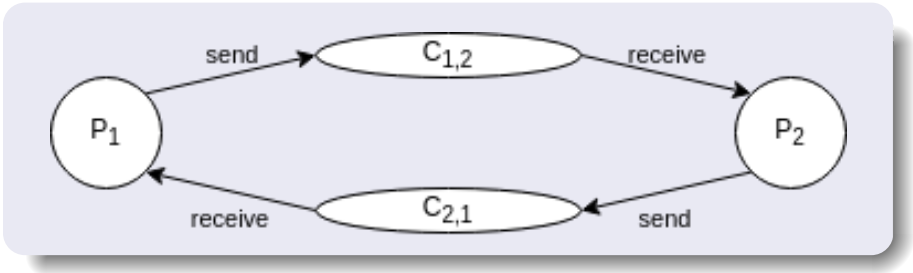
Università degli Studi di Torino

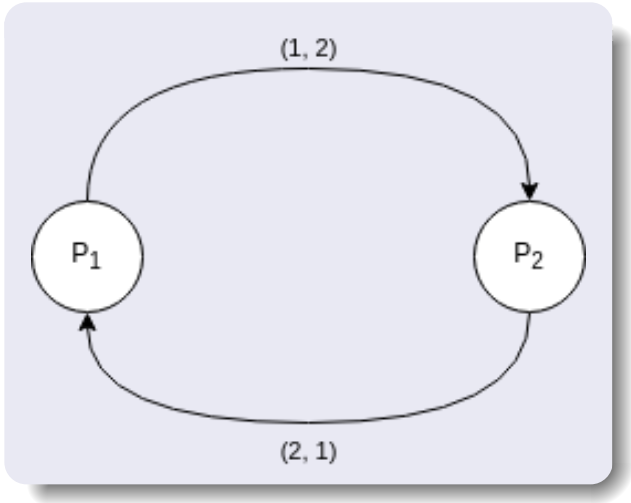
20 gennaio 2021

Overview

Modello a rete asincrono

- Un modello a rete di tipo *send/receive* è costituito da processi interconnessi tramite appositi canali.
- Per canale si intende una coda FIFO *affidabile* (i.e. i messaggi ivi contenuti sono ordinati e non duplicati).
- Un processo invia un messaggio tramite l'apposita azione di output `send`, mentre la lettura avviene tramite l'azione di input `receive`.
- Quando un singolo messaggio può essere inviato a molteplici destinatari si parla di modello a rete *broadcast*.





Fault-tolerance

Un processo può fallire semplicemente stoppandosi in qualsiasi momento della sua esecuzione. Definiamo questo tipo di fallimento **stopping failure**.

Un fallimento di questo tipo può essere modellato aggiungendo a ogni processo P_i un evento di input chiamato $stop_i$, questo causa il fallimento del processo cambiandone lo stato e disabilitando tutti i suoi task.

Un sistema A interfacciato verso gli utenti U è fault-tolerant rispetto a f fallimenti se rispetta la seguente condizione di liveness:

f-failure termination

Per ogni esecuzione del sistema composto $A \times U$ in cui occorrono degli eventi di stop su al più f porte, ogni invocazione su una porta non-fallita ha una risposta.

Overview

Da modello a rete al modello a memoria condivisa

Perché convertire un modello a rete in uno a memoria condivisa?

- I sistemi a memoria condivisa sono di più facile comprensione e offrono una maggiore espressività.
- Esistono numerosi algoritmi già sviluppati per i modelli a memoria condivisa.
- Il modello a rete simula ed eredita le caratteristiche proprie di un sistema a memoria condivisa.

I-simulazione

Un sistema B **simula** un sistema A se il suo comportamento è indistinguibile per un insieme di utenti esterni U.

I-simulazione

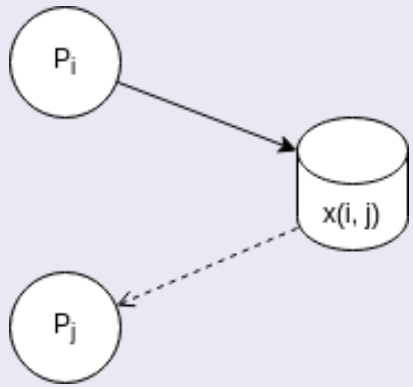
Un sistema B composto da n processi P_i con $1 \leq i \leq n$, è una I-simulazione (dove I indica un determinato insieme di porte) di A se per ogni esecuzione α di B e per ogni collezione di utenti U_i, esiste una esecuzione α' di A con gli stessi utenti per cui:

- α e α' sono indistinguibili agli utenti U.
- per ogni i un evento $stop_i$ occorre in α se e solo se occorre in α'

SimpleSRSim - 1

- Supponiamo di voler creare un sistema asincrono a memoria condivisa B che ne simuli uno a rete A.
- Per ogni arco di A che collega due processi P_i e P_j, nel sistema B si crea una variabile condivisa $x(i,j)$ modificabile solamente da P_i e leggibile esclusivamente da P_j.
- Ogni variabile condivisa del tipo $x(i,j)$ contiene una propria coda di messaggi, utile per l'emulazione delle operazioni di base di un sistema a rete.

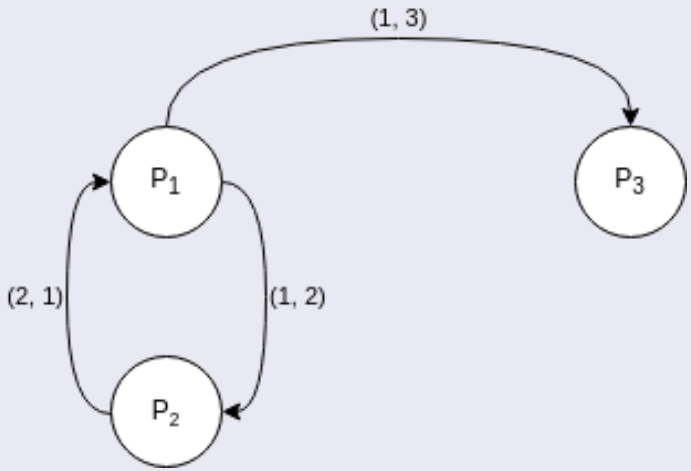
Variabile condivisa single-reader/single-writer



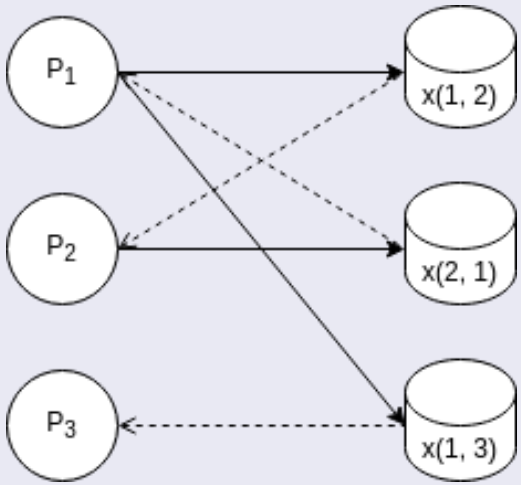
SimpleSRSim - 2

- L'invio del messaggio m dal processo i al processo j viene emulato aggiungendo il messaggio m al fondo della coda contenuta in $x(i, j)$.
- Un generico processo i verifica ciclicamente la presenza di eventuali nuovi messaggi ricevuti da un qualche processo j esaminando il contenuto della variabile $x(j, i)$.
- L'elaborazione di un generico messaggio m rimane invariata.

Modello a rete



Modello a memoria condivisa



SimpleBCastSim - 1

- In questo caso, per ogni P_i con $1 \leq i \leq n$, il sistema B ha una variabile condivisa $x(i)$ a singolo scrittore/multipli lettori.
  La variabile è scrivibile da P_i e leggibile da tutti i processi (P_i incluso), e contiene una coda di messaggi, inizialmente vuota.

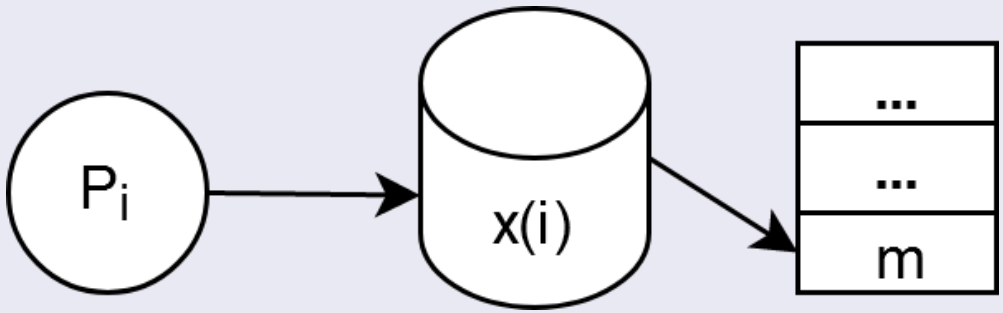
SimpleBCastSim - 1

- In questo caso, per ogni P_i con $1 \leq i \leq n$, il sistema B ha una variabile condivisa $x(i)$ a singolo scrittore/multipli lettori.
  La variabile è scrivibile da P_i e leggibile da tutti i processi (P_i incluso), e contiene una coda di messaggi, inizialmente vuota.
- Come con SimpleSRSim, un processo P_i di B simula direttamente lo stesso i-esimo processo di A.

SimpleBCastSim - 2

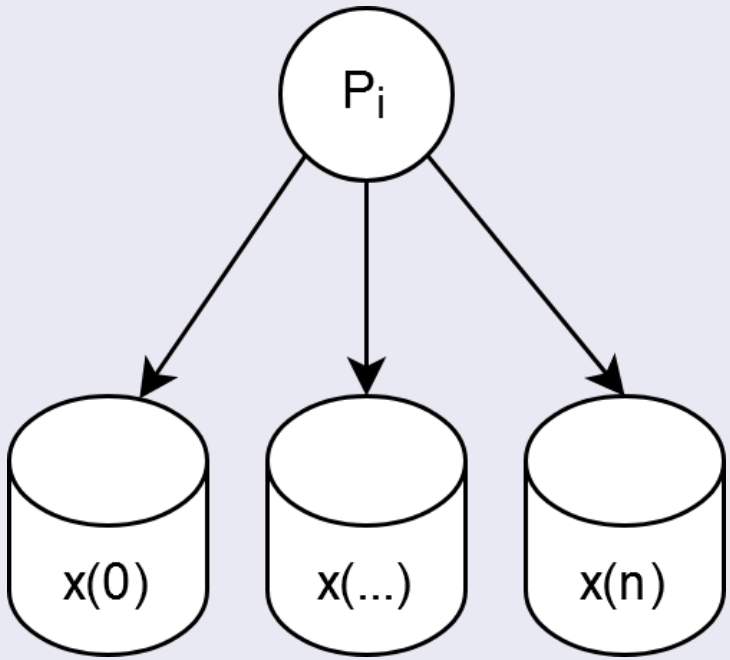
Broadcast

Per simulare un'azione $bcast(m)_i$ di P_i, il processo i di A aggiunge il messaggio m alla fine della coda presente nella variabile $x(i)$.



Ricezione

Il processo *i* periodicamente controlla tutte le variabili *x(j)* (inclusa *x(i)*), per verificare se sono presenti dei nuovi messaggi. Se effettivamente ve ne sono, li gestisce allo stesso modo di P_i.



Equivalenza tra modello Send/Receive e Broadcast

Teorema

Dato un generico sistema asincrono a rete broadcast A esiste un sistema asincrono send / receive B che è una I-simulazione di A.

Data questa equivalenza da ora in poi supporremo di lavorare in sistemi asincroni a rete broadcast, in quanto permettono la stesura di algoritmi più intuitivi e più facilmente dimostrabili.

Problema del consenso

Un insieme di n utenti U_i interagiscono con un sistema di n processi P_i attraverso un certo modello di comunicazione, ogni utente U_i inizializza con un certo valore v il processo P_i attraverso l'azione $init(v)_i$.

I processi P_i interagiranno tra di loro per compiere una scelta unanime di un certo valore v da comunicare agli utenti U_i attraverso l'azione $decide(v)_i$.

Impossibilità del consenso nel modello a rete

Il modello a rete eredita dal modello a memoria condivisa l'impossibilità del consenso.

Teorema

Dato un qualsiasi sistema asincrono a rete composto da un numero $n \geq 2$ di processi, non esiste un algoritmo che risolva il problema del consenso e garantisca la *1-failure termination*.

Dimostrazione

- Supponiamo per **assurdo** che esista un algoritmo `A` in grado di risolvere il problema del consenso in un sistema asincrono a rete broadcast e che garantisca la 1-failure termination.
- Per quanto dimostrato è possibile ottenere un algoritmo B per un sistema a modello a memoria condivisa che è un n-simulazione di `A`.
- Pertanto `B` risolverebbe il problema del consenso garantendo la 1-failure termination. Questo però contraddice il teorema dell'impossibilità del consenso nel modello a memoria condivisa.

Overview

Una soluzione randomizzata

Poichè problema del consenso è di vitale importanze in diversi settori, come ad esempio la gestione di transazioni in database distribuiti; è stato necessario sviluppare degli espedienti:

- Indebolimento dei requisiti di correttezza
- Rafforzamento del modello attraverso:
  - **Utilizzo della randomizzazione**
  - Failure detection
  - Consenso su un insieme di valori
  - Consenso parziale

Requisiti di correttezza

Un sistema A risolve il problema del consenso se per ogni collezione di utenti U garantisce:

- **Well-formedness**: ogni interazione tra il sistema ed utenti U_i è ben formata:
  - Non contiene azioni ripetute di *init*,
  - Non contiene azioni ripetute di *decide*,
  - Ogni *decide* è preceduto un *init*.
- **Agreement**: Tutti i valori di decisione sono uguali.
- **Validity**: Se tutte le azioni di *init* danno in input lo stesso valore v, allora l'unico valore che può essere deciso è v.
- **Failure-free termination**: In ogni esecuzione failure-free in cui un *init* è stata fatta su ciascuna porta, un evento *decide* viene fatto su tutte le porte.

Requisiti di terminazione

Un sistema A composto da n processi P_i è *fault-tolerant*, rispetto a f $(0 \leq f \leq n)$ fallimenti, se soddisfa la seguente proprietà di terminazione:

f-failure termination

In ogni esecuzione fair in cui degli eventi di *init* occorrono su tutte le porte, se ci sono al più f eventi di *stop*, allora un evento di *decide* occorre su tutte le porte non-fallite.

Algoritmo di BenOr

- L'algoritmo di BenOr funziona con $n > 3f$ processi e con l'insieme di valori di scelta $V = \{0, 1\}$.
- Ogni P_i esegue una serie infinita di **stage** numerati in maniera incrementale e ognuna di essa è divisa in due **round**.
- I processi nei vari stage si scambiano messaggi i quali contengono:
  - un tag che può essere o R (Report) o P (Propose);
  - un numero che indica lo stage corrente, indicato con s;
  - un valore definito nel round indicato con v.
- L'algoritmo continua la propria esecuzione anche dopo aver effettuato una $decide(v)_i$.

Algoritmo di BenOr - Inizializzazione

- Ogni processo P_i ha due variabili locali x e y inizializzate a *null*.
- All'occorrenza di un input $init(v)_i$ sul processo P_i viene assegnato ad x il valore v ($v \in V$).
- La numerazione degli stage inizia con il valore 0 e viene incrementato all'inizio di ogni round 1.

Algoritmo di BenOr - Round 1

- P_i invia in broadcast agli altri processi un messaggio della forma (R, s, x).
- Successivamente attende di ricevere messaggi (R, s, *) da n - f processi[1].
- Se tutti i messaggi ricevuti hanno lo stesso valore v allora salva nella variabile y il valore v.
- altrimenti salva in y il valore null.

[1] Il valore indicato da * può essere 0 o 1

Algoritmo di BenOr - Round 2

- P_i invia in broadcast agli altri processi un messaggio della forma (P, s, y).
- Successivamente attende di ricevere messaggi (P, s, *) da n - f processi[2].
- Se tutti i messaggi ricevuti hanno lo stesso valore v e v diverso da null allora salva in x il valore v ed esegue l'azione $decide(v)_i$ (se non è già stato fatto in uno stage precedente).
- altrimenti Se almeno $n - 2f$ messaggi ricevuti hanno lo stesso valore v e v diverso da null allora salva in x il valore v.
- altrimenti viene assegnato ad x un valore scelto casualmente[3] dall'insieme V.

[2] Il valore indicato da * può essere 0, 1 o null

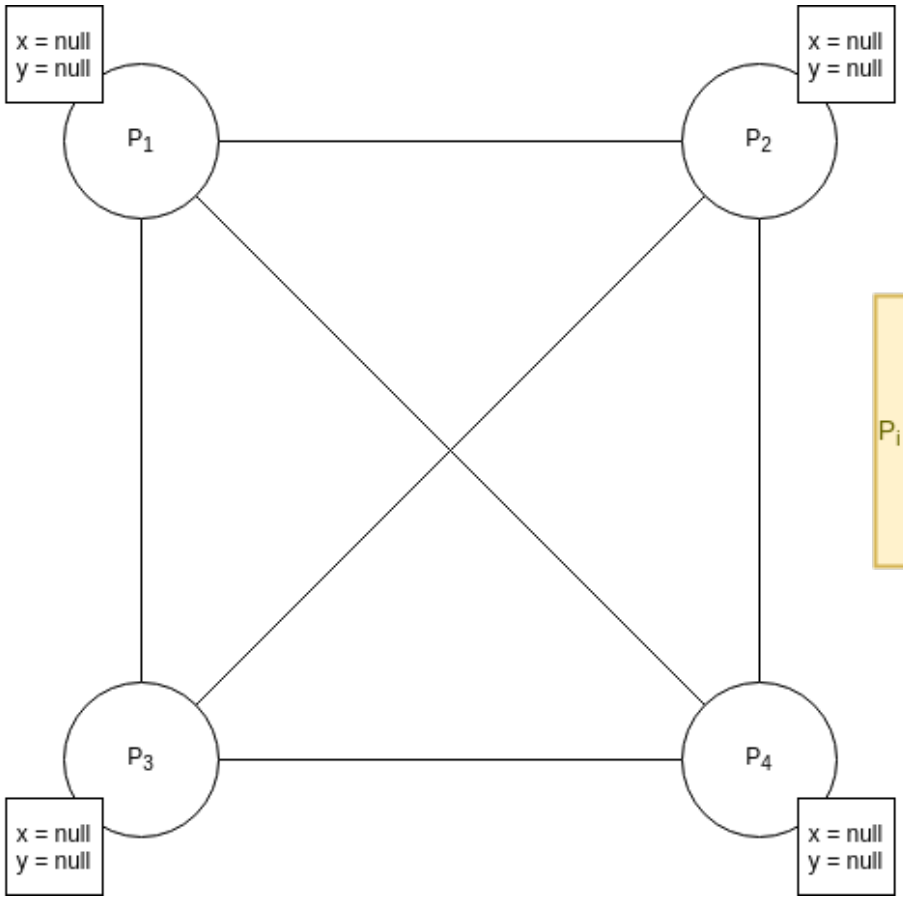
[3] Quest'azione rende probabilistico l'intero algoritmo.

Considerazioni

Teorema

L'algoritmo di BenOr garantisce le proprietà di well-formdness, validity e agreement. Inoltre viene garantita con probabilità 1 che tutti i processi non-falliti prima o poi decidono.

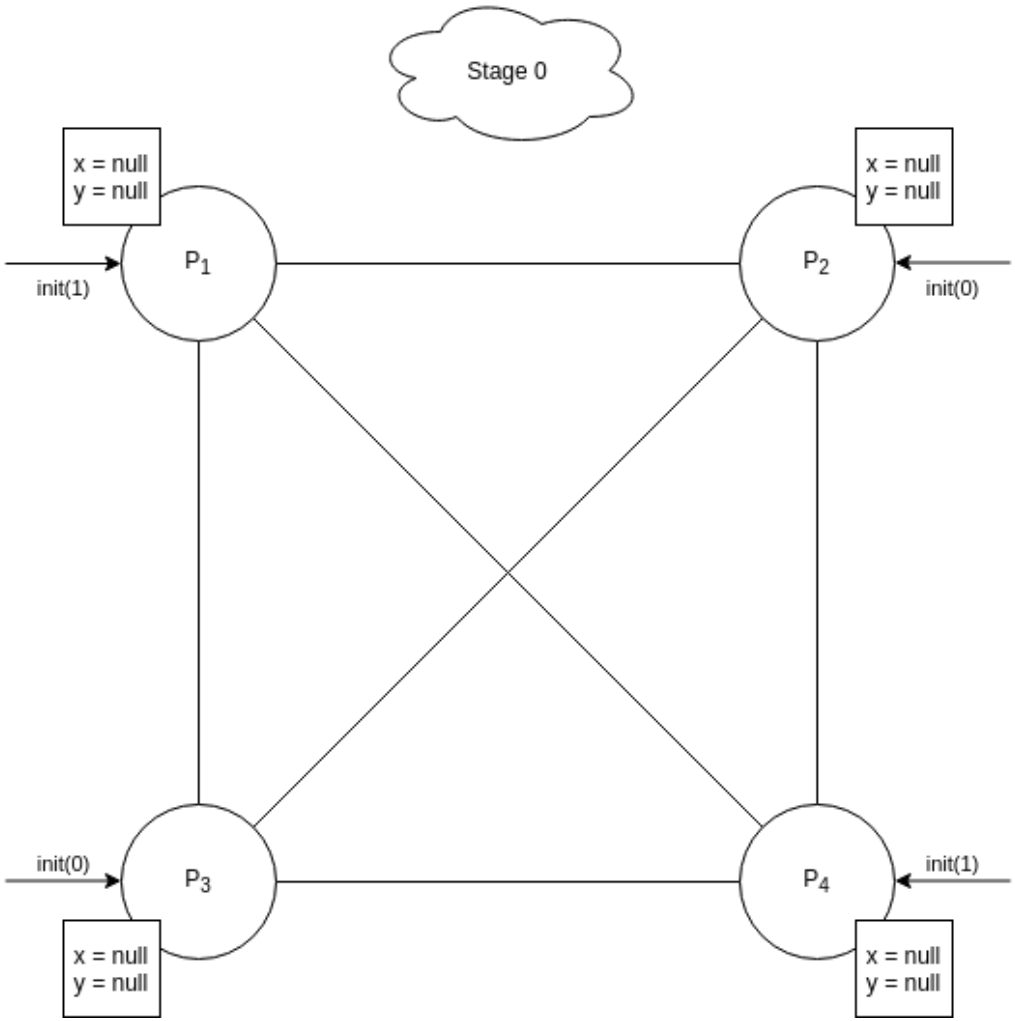
Simulazione - Configurazione iniziale



Rete broadcast asincrona con 4 processi di cui uno che avrà dei malfunzionamenti

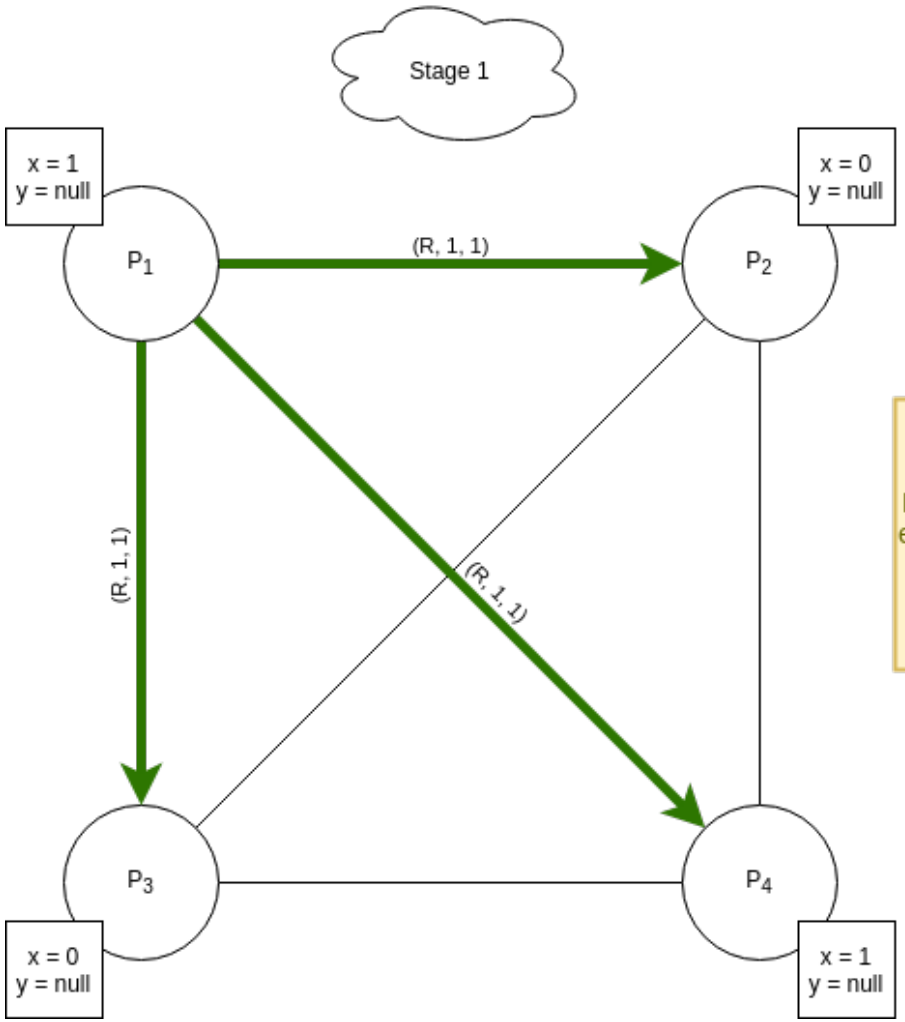
Inizialmente tutti i processi P_i pongono le proprie variabili x e y a null.

Simulazione - Stage 0 (Inizializzazione)



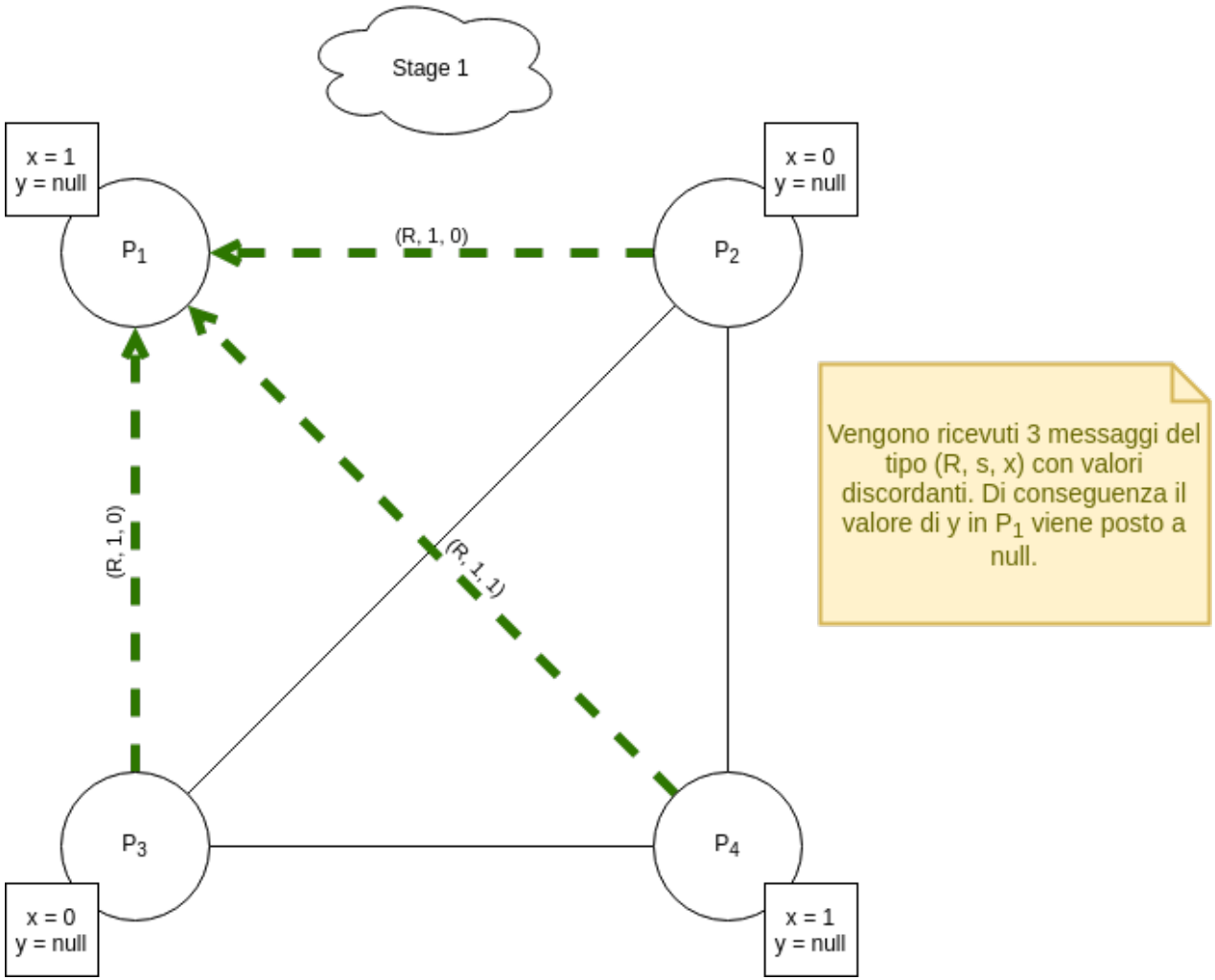
A ogni processo P_i è richiesto di inizializzare il valore della propria variabile *x* tramite le apposite azioni *init(v)*.

Simulazione - Stage 1 (Report 1/2)

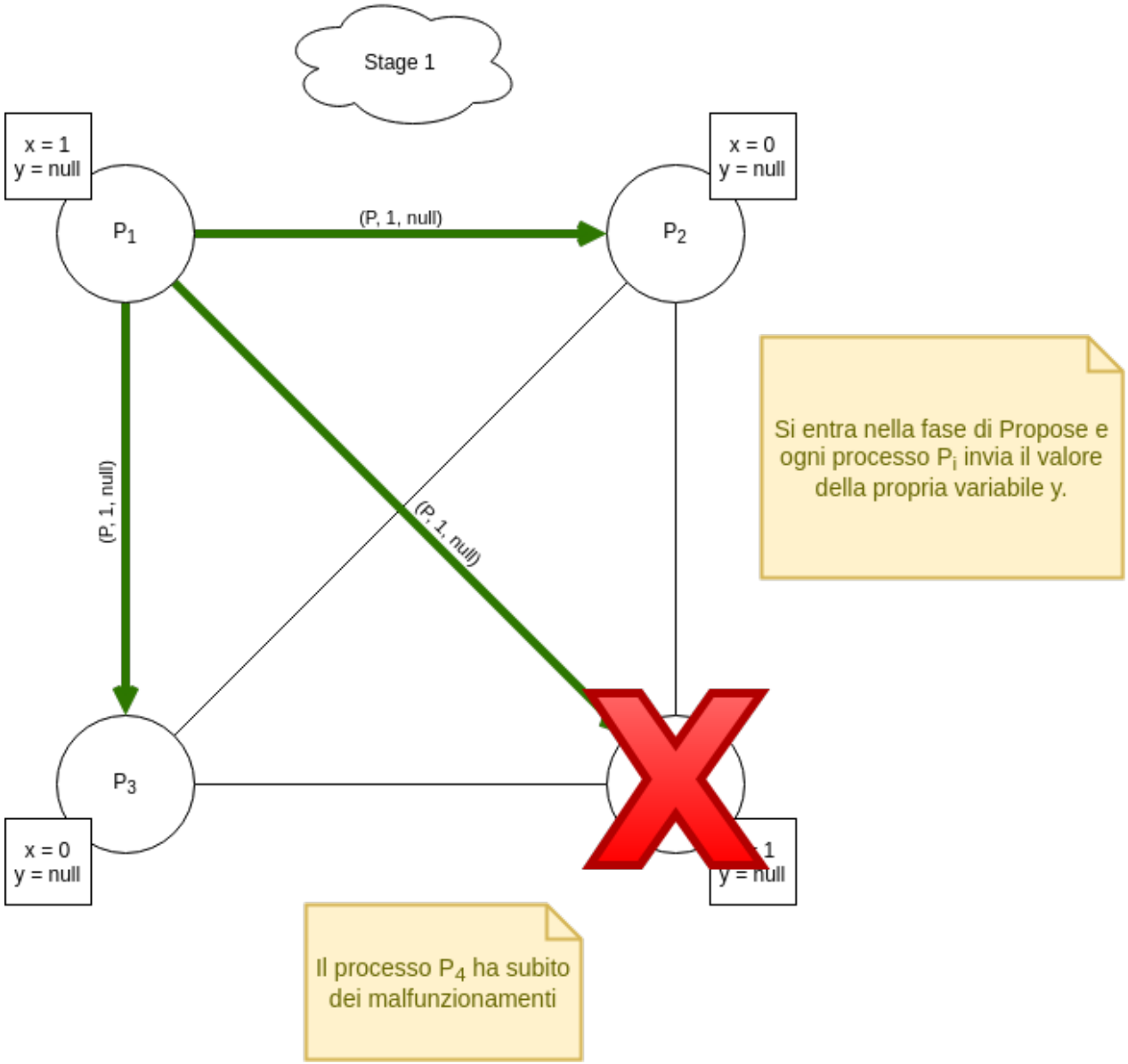


L'invio di tuple del tipo (R, s, x) è eseguito in modo analogo da tutti i processi P_i.

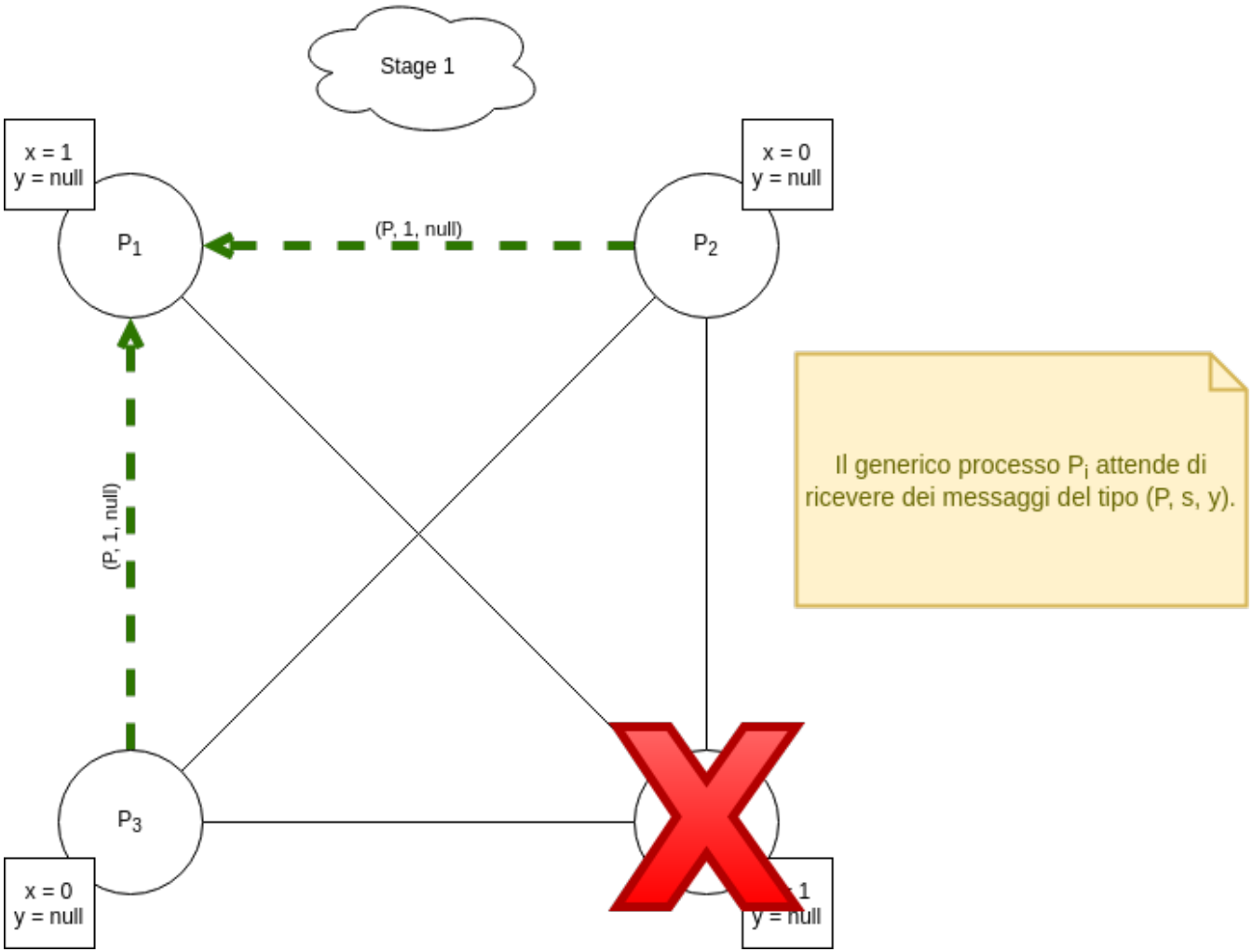
Simulazione - Stage 1 (Report 2/2)



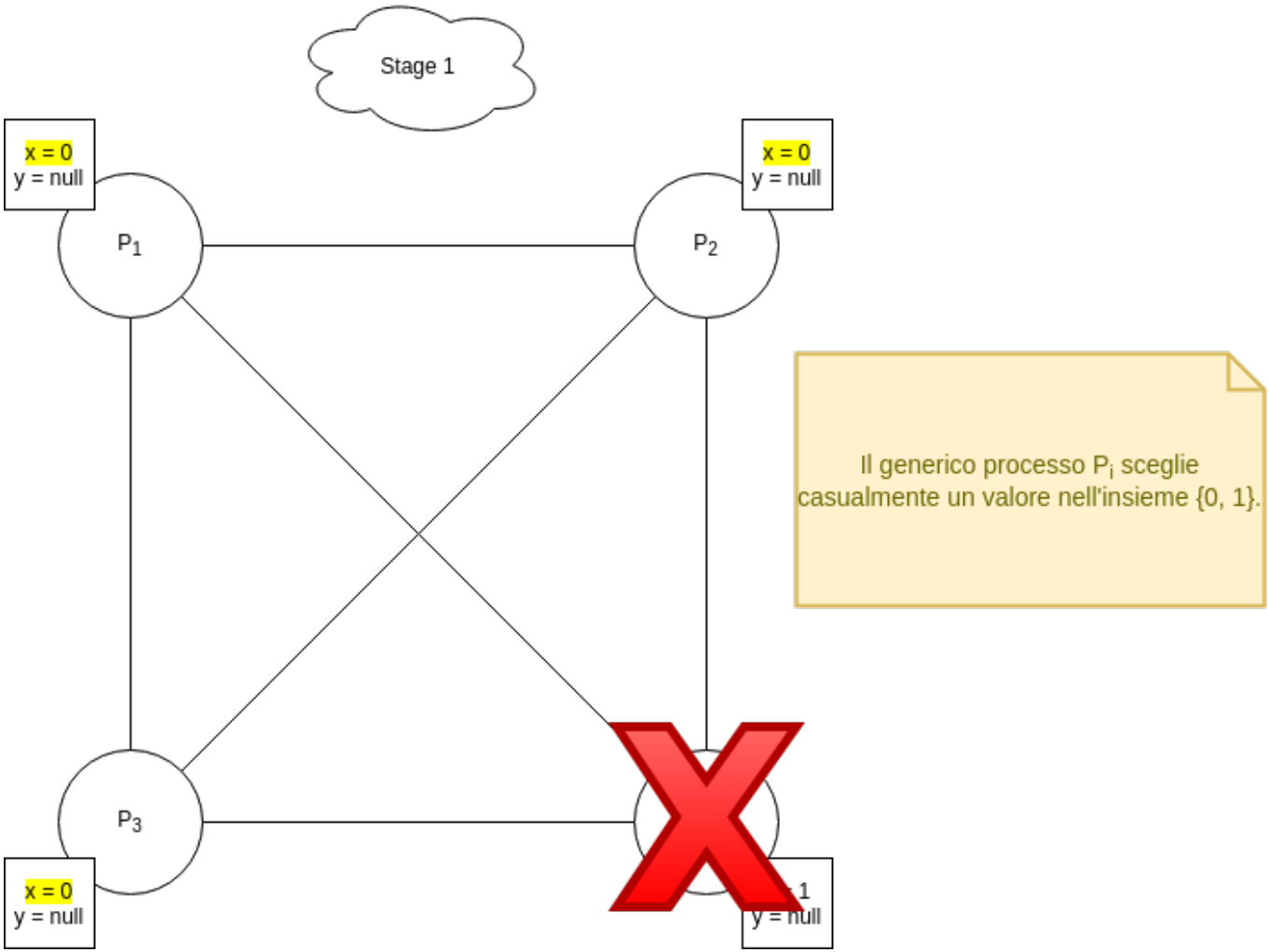
Simulazione - Stage 1 (Propose 1/2)



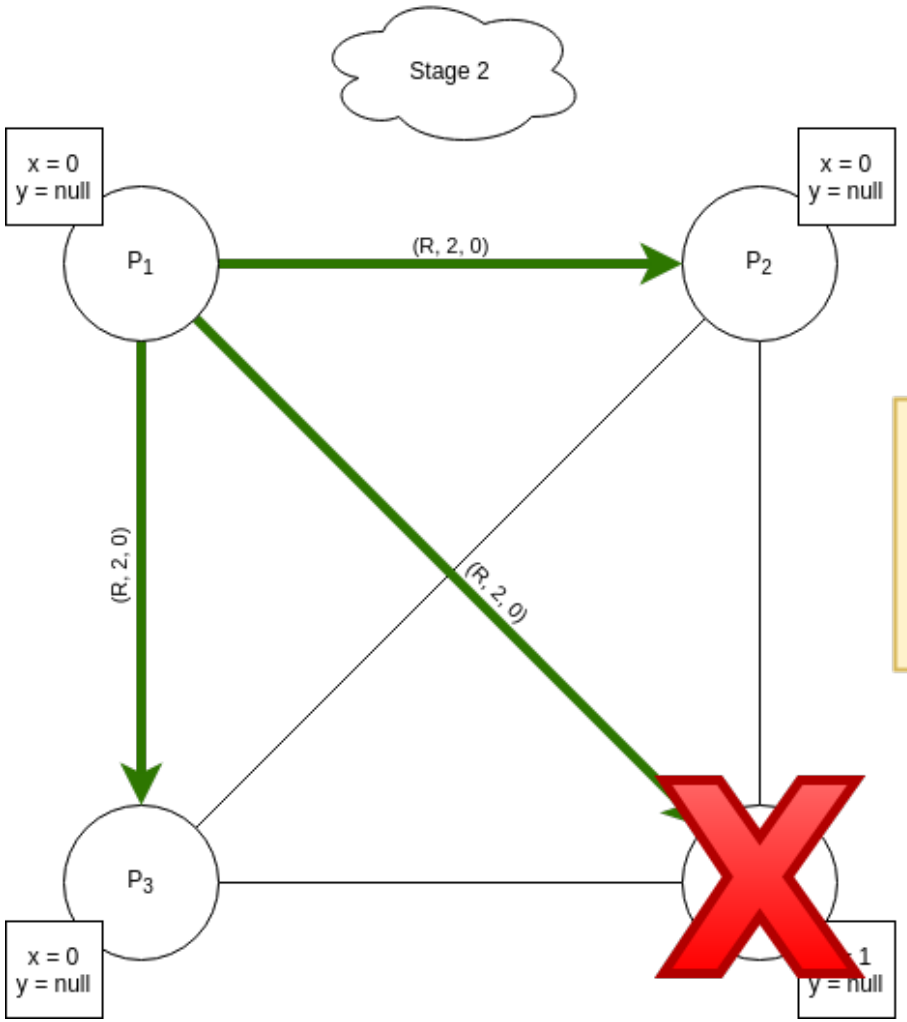
Simulazione - Stage 1 (Propose 2/2)



Simulazione - Stage 1 (Scelta casuale)

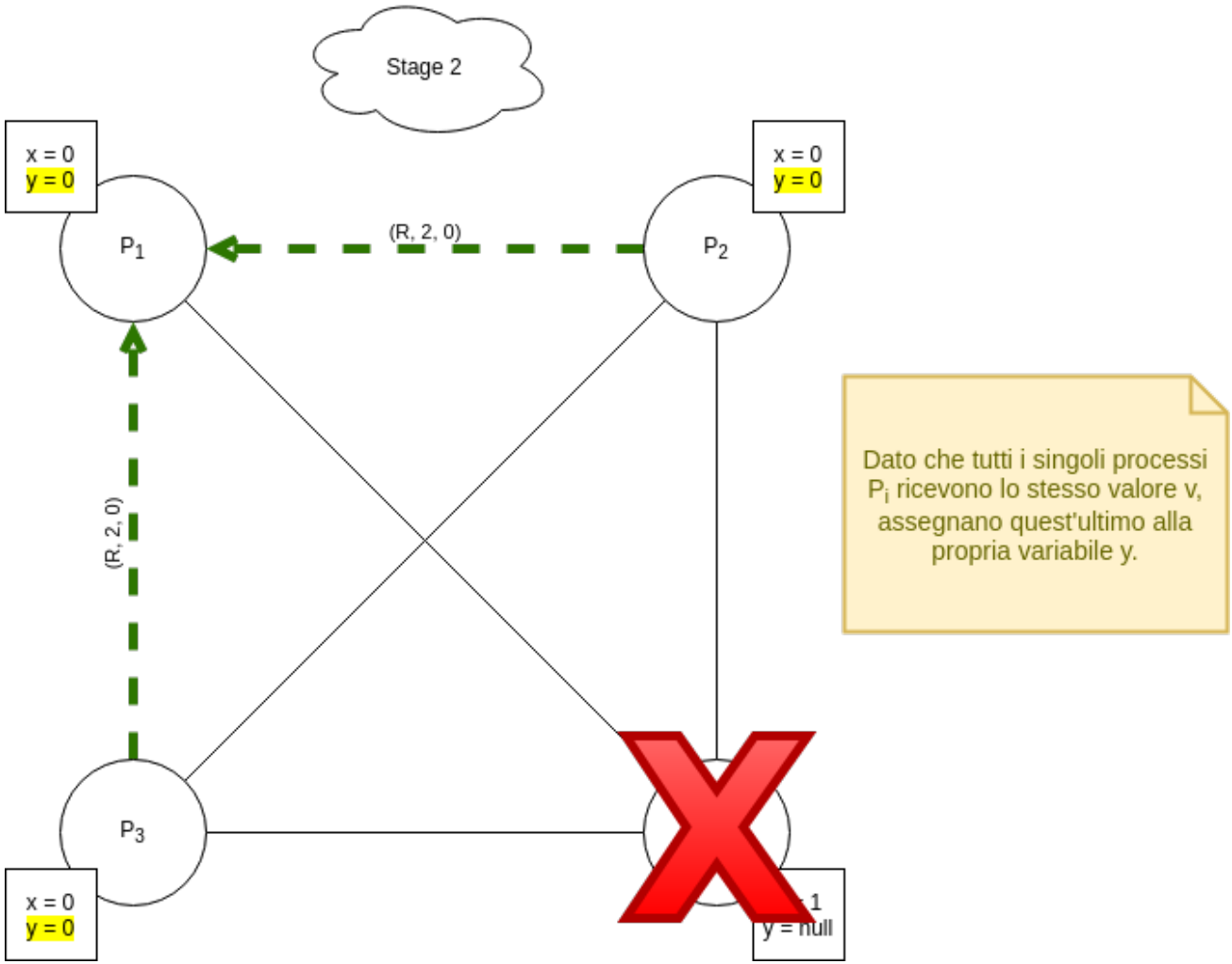


Simulazione - Stage 2 (Report 1/2)

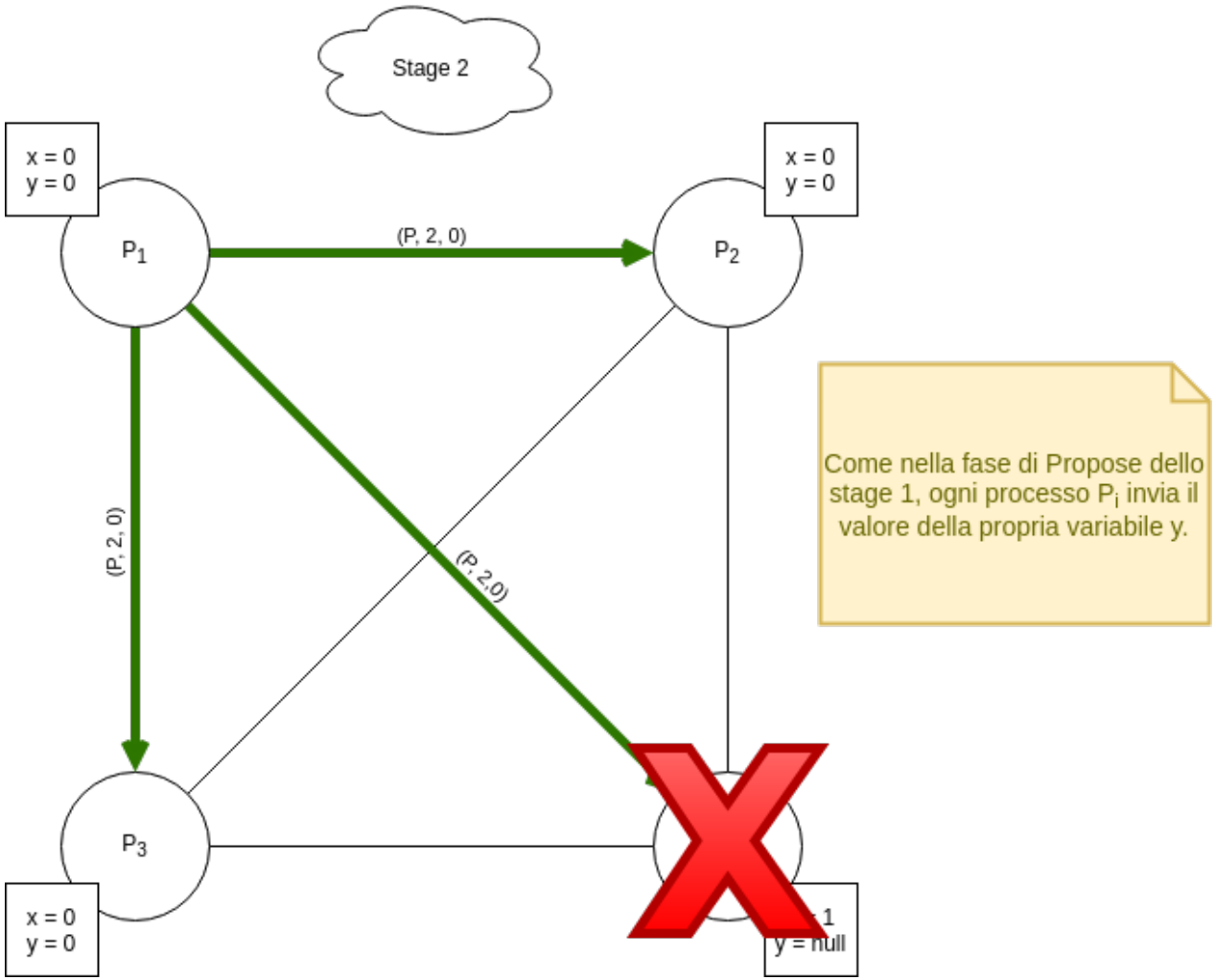


Analogamente a ciò che è accaduto nella fase di Report dello stage 1, vengono inviate tuple del tipo (R, s, x) da tutti i processi P_i.

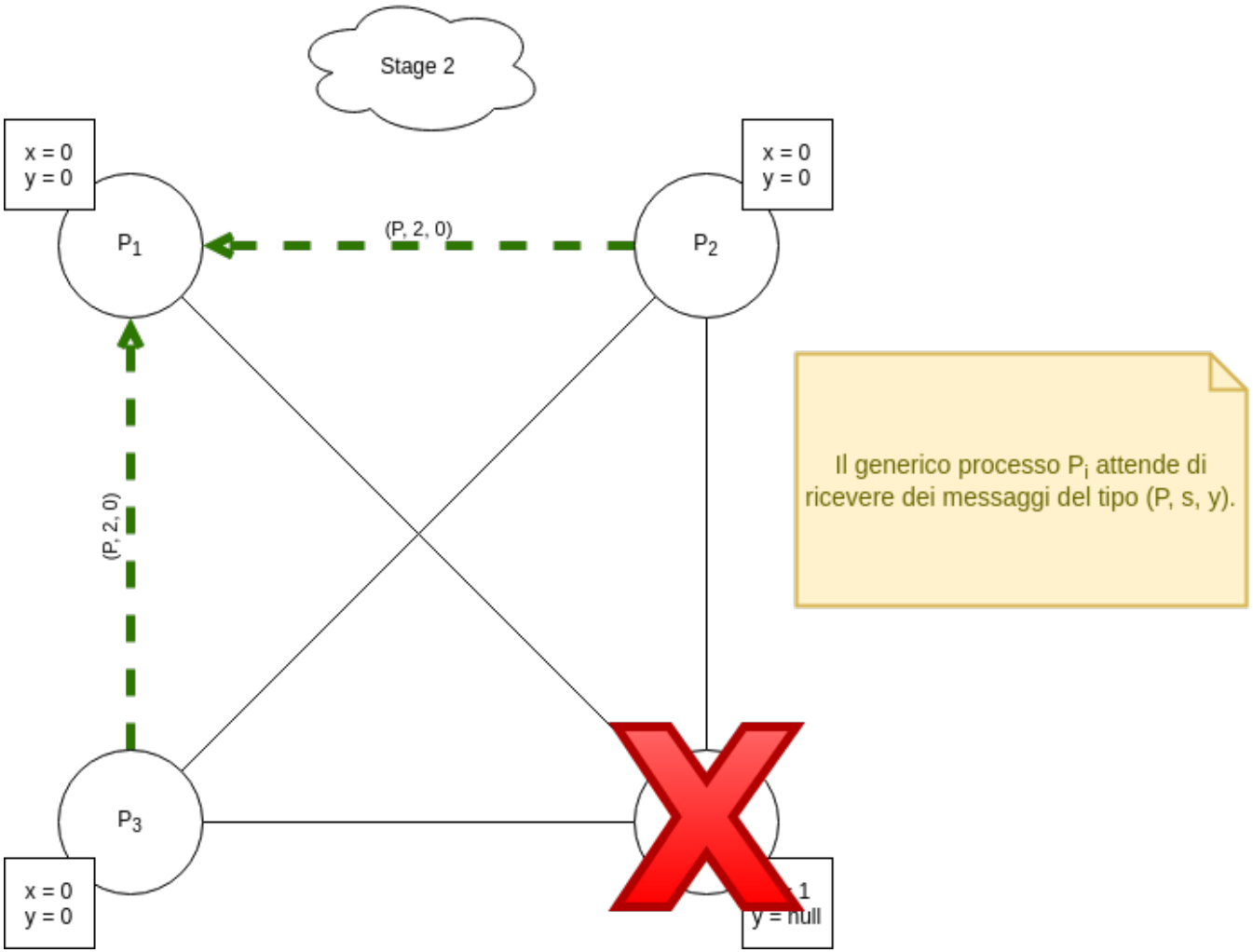
Simulazione - Stage 2 (Report 2/2)



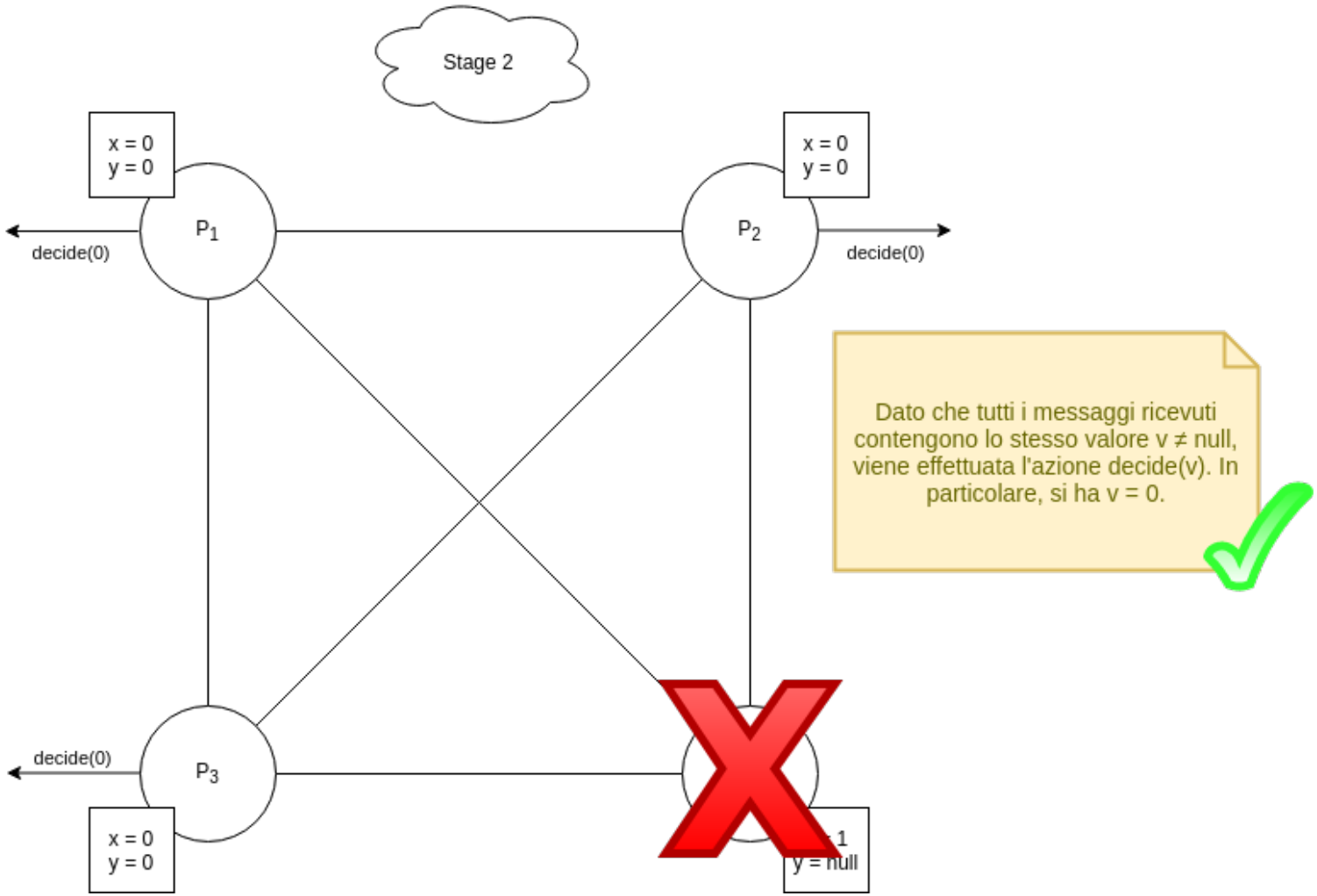
Simulazione - Stage 2 (Propose 1/2)



Simulazione - Stage 2 (Propose 2/2)



Simulazione - Stage 2 (Decisione)



Dimostrazione Correttezza

Dimostriamo che l'algoritmo BenOr risolve il problema del consenso.

Lemma

L'algoritmo BenOr garantisce la well-formedness, l'agreement e la validity.

Dimostrazione Correttezza (Well-formedness) (1/2)

Va dimostrato che per ogni esecuzione:

1. Non esistono azioni ripetute $init(v)_i$, ovvero che ogni processo viene inizializzato al più una volta.
2. Non esistono azioni ripetute $decide(v)_i$, ovvero che ogni processo decide al più una volta.
3. Ogni $decide(k)_i$ è preceduta da una $init(v)_i$.

Dimostrazione Correttezza (Well-formedness) (2/2)

Dimostrazione:

1. Ogni utente può fare al più una azione di init.
2. Sebbene i processi continuino la loro esecuzione anche dopo aver deciso un valore, per definizione ogni processo eseguira l'azione di decide una sola volta.
3. Un processo non inizia la sua esecuzione finchè non riceve un'azione di init. Di conseguenza non può esistere una esecuzione che contenga una decide prima di una init.

Dimostrazione Correttezza (Validity)

Va dimostrato che se ogni processo è inizializzato allo stesso valore v, allora v è l'unico valore di decisione possibile.

Dimostrazione:

1. Per ipotesi tutte le azioni $init(v)_i$ vengono fatte con lo stesso valore v.

Dimostrazione Correttezza (Validity)

Va dimostrato che se ogni processo è inizializzato allo stesso valore v, allora v è l'unico valore di decisione possibile.

Dimostrazione:

1. Per ipotesi tutte le azioni $init(v)_i$ vengono fatte con lo stesso valore v.
2. Pertanto nel round di report dello stage 1 tutti i processi invieranno lo stesso messaggio ("R", 1, v).

Dimostrazione Correttezza (Validity)

Va dimostrato che se ogni processo è inizializzato allo stesso valore v, allora v è l'unico valore di decisione possibile.

Dimostrazione:

1. Per ipotesi tutte le azioni $init(v)_i$ vengono fatte con lo stesso valore v.
2. Pertanto nel round di report dello stage 1 tutti i processi invieranno lo stesso messaggio ("R", 1, v).
3. Segue quindi che tutti i processi riceverann almeno $n - f$ messaggi ("R", 1, v), di consequenza ogni processo invierà un messaggio ("P", 1, v).

Dimostrazione Correttezza (Validity)

Va dimostrato che se ogni processo è inizializzato allo stesso valore v, allora v è l'unico valore di decisione possibile.

Dimostrazione:

1. Per ipotesi tutte le azioni $init(v)_i$ vengono fatte con lo stesso valore v.
2. Pertanto nel round di report dello stage 1 tutti i processi invieranno lo stesso messaggio ("R", 1, v).
3. Segue quindi che tutti i processi riceverann almeno $n - f$ messaggi ("R", 1, v), di consequenza ogni processo invierà un messaggio ("P", 1, v).
4. Concludendo tutti i processi riceveranno allora $n - f$ messaggi ("P", 1, v) decidendo dunque il valore v.

Dimostrazione Correttezza (Agreement) (1/2)

Va dimostrato che tutti i valori di decisione sono uguali.

Dimostrazione:

1. Supponiamo che P_i decide un certo valore v nello stage s, e che sia il primo processo a eseguire una decisione.

Dimostrazione Correttezza (Agreement) (1/2)

Va dimostrato che tutti i valori di decisione sono uguali.

Dimostrazione:

1. Supponiamo che P_i decide un certo valore v nello stage s, e che sia il primo processo a eseguire una decisione.
2. Ciò implica che P_i abbia ricetuto $n - f$ mesaggi del tipo ("P", s, v).

Dimostrazione Correttezza (Agreement) (1/2)

Va dimostrato che tutti i valori di decisione sono uguali.

Dimostrazione:

1. Supponiamo che P_i decide un certo valore v nello stage s, e che sia il primo processo a eseguire una decisione.
2. Ciò implica che P_i abbia ricetuto $n - f$ mesaggi del tipo ("P", s, v).
3. Pertanto ogni altro processo P_j che completa lo stage s, avrà ricevuto almeno $n - 2f$ messaggi, questo perchè riceverà gli stessi $n - f$ messaggi che ha ricevuto P_i, tolti al più f messaggi, nel caso in cui f processi falliscano dopo aver inviato il messaggio a P_i. Si noti che $n - 2f = (n - f) - f$

Dimostrazione Correttezza (Agreement) (2/2)

4. Abbiamo quindi due alternative per ogni altro processo P_j che completa lo stage s:
   - P_j ha ricevuto $n - f$ messaggi ("P", s, v) contenenti lo stesso valore v.
     $\implies$ P_j decide v e manda a tutti ("R", s, v).
   - P_j ha ricevuto $n - 2f \leq k < n - f$ messaggi ("P", s, v) contenenti lo stesso valore v.
     $\implies$ P_j setta $x := v$ e manda a tutti ("R", s, v).

   In ogni caso allo stage successivo $s + 1$ tutti i processo riceveranno almeno $n - f$ messaggi ("R", s + 1, v) e invieranno pertanto un messaggio ("P", s + 1, v), di conseguenza tutti i processi riceveranno $n - f$ messagi ("P", s + 1, v) e decideranno quindi v.

Quasi certo VS *Certo*

Si noti che probabilità 1 non vuol dire certezza. In seguito dimostreremo che la probabilità di terminazione tende a 1 solo dopo un numero **infinito** di stage s. Quindi possiamo solo essere sicuri che prima o poi accadrà solo se supponiamo di poter aspettare tempi infiniti.

Un esecuzione (finita) senza decisioni

Possiamo immaginare un'esecuzione fair del BenOr relativa a $3f + 1$ in cui nessuno di questi decide; affinche ciò accada in ogni stage s deve accadere che:

- m processi, con $f + 1 \leq m \leq 2f$, presentano $x = 0$ mentre i rimanenti hanno $x = 1$.
- Al termine della fase di Report, tutti i processi avranno $y = null$
- Nella fase di Propose assegneranno a x un nuovo valore randomico.
- m processi assegnano $x = 0$ e i rimanenti assegnano $x = 1$

Nello stage $s + 1$ si ritorna quindi nella situazione iniziale, che potrebbe pertanto ripetersi un numero considerevole di volte.

Probabilità di terminazione

Lemma

Per ogni stage s, tutti i processi decideranno entro lo stage successivo $s+1$ con una probabilità p tale che:

$$p \geq 1 - (1 - \frac{1}{2^n})^s$$

Dimostrazione (1/8)

Se $s = 0$ allora la dimostrazione è banale infatti si ha che:

$$1 - (1 - \frac{1}{2^n})^s = 1 - (1 - \frac{1}{2^n})^0 = 1 - 1 = 0$$

I.e., si sta dicendo che ogni processo termina con probabilità $p \geq 0$, il che è ovvio.

Dimostrazione (2/8)

Consideriamo ora il caso $s \geq 1$.
Dimostriamo innanzitutto che alla fine dello stage s, ogni processo imposta x allo stesso valore v con una probabilità $p \geq \frac{1}{2^n}$.

1. Sia α l'esecuzione più corta per cui un processo P_i ha ricevuto $n - f$ messaggi ("R", s, *).

Dimostrazione (2/8)

Consideriamo ora il caso $s \geq 1$.
Dimostriamo innanzitutto che alla fine dello stage s, ogni processo imposta x allo stesso valore v con una probabilità $p \geq \frac{1}{2^n}$.

1. Sia α l'esecuzione più corta per cui un processo P_i ha ricevuto $n - f$ messaggi ("R", s, *).
2. Se almeno $f + 1$ di questi messaggi contengono lo stesso valore v, allora diciamo che v è un valore *buono* per α.

Dimostrazione (2/8)

Consideriamo ora il caso $s \geq 1$.
Dimostriamo innanzitutto che alla fine dello stage s, ogni processo imposta x allo stesso valore v con una probabilità $p \geq \frac{1}{2^n}$.

1. Sia α l'esecuzione più corta per cui un processo P_i ha ricevuto $n - f$ messaggi ("R", s, *).
2. Se almeno $f + 1$ di questi messaggi contengono lo stesso valore v, allora diciamo che v è un valore *buono* per α.
3. Sono possibili due casi:
   - E' presente un solo valore *buono*, i.e., P_i ha ricevuto o almeno $f + 1$ ("R", s, 0) oppure almeno $f + 1$ ("R", s, 1).
   - Sono presenti due valori *buoni*, i.e., P_i ha ricevuto almeno $f + 1$ ("R", s, 0) e almeno $f + 1$ ("R", s, 1).

Dimostrazione (3/8): Un solo valore buono

4. Nel caso in cui P_i ha ricevuto un solo valore *buono* v, ogni altro processo P_j avrà ricevuto almeno 1 ($f + 1 - f = 1$) messaggi del tipo (”R”, s, v).

Dimostrazione (3/8): Un solo valore buono

4. Nel caso in cui P_i ha ricevuto un solo valore *buono* v, ogni altro processo P_j avrà ricevuto almeno 1 $(f + 1 - f = 1)$ messaggi del tipo ("R", s, v).
5. Non appena P_j avrà ricevuto $n - f$ messaggi ("R", s, *), non potrà inviare un messagio del tipo ("P", s, $\overline{v}$), in quanto per definizione può farlo solamente se ha rivecuto esattamente $n - f$ copie di $\overline{v}$, ma per quanto detto soprà sappiamo che ciò è impossibile in quanto tra gli $n - f$ messaggi che ha ricevuto contengono almeno 1 copia di v.

Dimostrazione (3/8): Un solo valore buono

4. Nel caso in cui P_i ha ricevuto un solo valore *buono* v, ogni altro processo P_j avrà ricevuto almeno 1 $(f + 1 - f = 1)$ messaggi del tipo ("R", s, v).
5. Non appena P_j avrà ricevuto $n - f$ messaggi ("R", s, *), non potrà inviare un messagio del tipo ("P", s, $\overline{v}$), in quanto per definizione può farlo solamente se ha rivecuto esattamente $n - f$ copie di $\overline{v}$, ma per quanto detto soprà sappiamo che ciò è impossibile in quanto tra gli $n - f$ messaggi che ha ricevuto contengono almeno 1 copia di v.
6. Pertanto ogni messaggio ("P", s, *) che verrà inviato conterrà o il valore v oppure *null*.

Dimostrazione (4/8): Un solo valore buono

7. Nel momento in cui un processo P_j riceve $n - f$ messaggi ("P", s, v) allora, o avrà ricevuto almeno $n - 2f$ messaggi contenenti il valore *buono* v e imposterà x a v, oppure ne avrà ricevuti di meno e sceglierà il valore di x in manierò random.

Dimostrazione (4/8): Un solo valore buono

7. Nel momento in cui un processo P_j riceve $n - f$ messaggi ("P", s, v) allora, o avrà ricevuto almeno $n - 2f$ messaggi contenenti il valore *buono* v e imposterà x a v, oppure ne avrà ricevuti di meno e sceglierà il valore di x in manierò random.
8. La probabilità che tutti i processi che scelgono in maniera random, scelgano lo stesso valore v è: $\geq (\frac{1}{2})^n$, in quanto ogni processo ha probabilita $\frac{1}{2}$ di scegliere il valore v.

Dimostrazione (5/8): Due valori buoni

9. Nel caso in cui P_i ha ricevuto due valori *buoni*, ogni altro processo P_j avrà ricevuto almeno 1 messaggio del tipo ("R", s, 0) e almeno 1 messaggio del tipo ("R", s, 0).

Dimostrazione (5/8): Due valori buoni

9. Nel caso in cui P_i ha ricevuto due valori *buoni*, ogni altro processo P_j avrà ricevuto almeno 1 messaggio del tipo ("R", s, 0) e almeno 1 messaggio del tipo ("R", s, 0).
10. Non appena P_j avrà ricevuto $n - f$ messaggi ("R", s, *), non potrà inviare un nè messagi del tipo ("P", s, 0), nè messagi del tipo ("P", s, 1).

Dimostrazione (5/8): Due valori buoni

9. Nel caso in cui P_i ha ricevuto due valori *buoni*, ogni altro processo P_j avrà ricevuto almeno 1 messaggio del tipo ("R", s, 0) e almeno 1 messaggio del tipo ("R", s, 0).
10. Non appena P_j avrà ricevuto $n - f$ messaggi ("R", s, *), non potrà inviare un nè messagi del tipo ("P", s, 0), nè messagi del tipo ("P", s, 1).
11. Pertanto ogni messaggio ("P", s, *) che verrà inviato conterrà o il valore *null*.

Dimostrazione (6/8)

12. Ogni processo riceverà quindi unicamente messaggi del tipo ("P", s, *null*), dovendo quindi impostare il valore di x in maniera random.

Dimostrazione (6/8)

12. Ogni processo riceverà quindi unicamente messaggi del tipo ("P", s, *null*), dovendo quindi impostare il valore di x in maniera random.
13. Anche in questo caso la probabilità che tutti i processi impostino x allo stesso valore è: $\geq (\frac{1}{2})^n$.

Dimostrazione (7/8)

14. Abbiamo quindi dimostrato che in entrambi i casi alla fine dello stage s, con probabilità $p \geq \frac{1}{2^n}$, ogni processo imposta x allo stesso valore v.
15. Per **negazione** si ha che la probabilità p che **non** venga scelto un valore unico per x è $p \leq 1 - \frac{1}{2^n}$.
16. Poichè la le scelte che vengono fatte in uno stage sono **indipendenti** tra di loro abbiamo che, la probabilità p che non venga scelto un valore unico per x nei primi s stage è: $p \leq (1 - \frac{1}{2^n})^s$
17. Ancora una volta per **negazione** si ha che la probabilità p che venga scelto un valore unico per x entro lo stage s è: $p \geq 1 - (1 - \frac{1}{2^n})^s$

Dimostrazione (8/8)

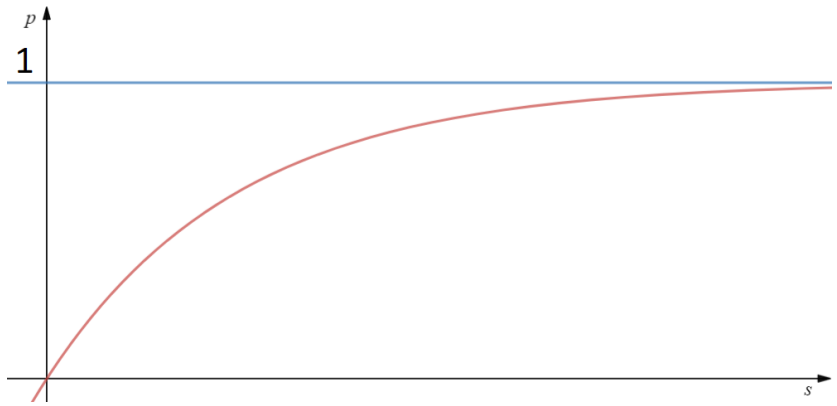
18. Se tutti i processi impostano lo stesso valore per x alla fine di uno stage s, allora per definizione dell'algoritmo tutti i processi *decideranno* quel valore di x alla fine dello stage $s+1$.
19. Per quanto detto prima la probabilità p che venga scelto un valore uni per x entro lo stage s è: $p \geq 1 - (1 - \frac{1}{2^n})^s$.
20. Dunque la probabilità di decidere un valore entro lo stage $s+1$ è:

$$p \geq 1 - (1 - \frac{1}{2^n})^s.$$

Terminazione con probabilità 1

E' facile vedere che:

$$\lim_{s\to\infty} 1 - (1 - \frac{1}{2^n})^s = 1$$



Questo vuol dire che al crescere del numero di stage la probabilità di decidere un valore entro lo stage $s + 1$.

All'infinito questo valore diventa pari a 1. Ovvero si ha che con probabilità 1, prima o poi tutti i processi decidono.

Ringraziamenti

Grazie per l'attenzione!

Riferimenti bibliografici:

Nancy A. Lynch, *Distributed Algorithms*, 1996
Marcos Kawazoe Aguilera e Sam Toueg, *Correctness Proof of Ben-Or's Randomized Consensus Algorithm*, 1998