

# La mutua esclusione (ii)

(dai sistemi concorrenti ai sistemi distribuiti)

# Sistema distribuito

- Rispetto al modello di Lamport si aggiunge il seguente vincolo:
  - un processo non può leggere direttamente il valore delle variabili di proprietà degli altri processi, ma per leggere un valore di una variabile di un processo deve esplicitamente inviare un messaggio ed attendere una risposta che conterrà questo valore
  
- Ciò implica che rispetto al modello di Lamport si aggiungono le seguenti assunzioni
  - i processi comunicano leggendo e scrivendo variabili attraverso scambi di messaggi. Il ritardo di trasmissione di un messaggio è imprevedibile ma finito
  - I canali di comunicazione sono affidabili. Un messaggio inviato viene ricevuto correttamente dal suo destinatario. Correttamente significa che uno stesso messaggio non viene ricevuto due volte e che se un messaggio viene ricevuto allora è stato inviato da qualcuno

# Un adattamento “didattico” del Bakery

---

Local variable

j: integer in range 1,..N; num: integer, initially all 0;

choosing: Boolean, initially false;

repeat

```
1   NCS
2   choosing:= true                %inizio doorway%   %inizio trying%
3   for j≠i do
4       send num to pj
5       receive reply(v) from pj
6       num:=max(v,num)
7       num:= num+1
8   choosing:= false                %fine doorway%
```

# Un adattamento “didattico” del Bakery

```
1   for j:= 1 to n do begin                                %inizio bakery%
2       repeat
3           send choosing to pj
4           receive reply(v) from pj
5       until v
6       repeat
7           send num to pj
8           receive reply(v) from pj
9       until v=0 or {v, j } < {num,i}
10  end                                                    %fine bakery%      %fine trying%
11  CS
12  num:=0;                                                %exit protocol%
forever
```

# Un adattamento “didattico” del Bakery

## Thread di comunicazione:

Upon the arrival of a num message from process j

**send** reply(num) **to** process j

Upon the arrival of a choosing message from process j

**send** reply(choosing) **to** process j

---

# Perchè non è una “buona” soluzione?

- In questo algoritmo il processo  $P_i$  si comporta da server rispetto alle proprie variabili **num** e **choosing**
- Ogni processo si limita a recuperare (leggere) i valori locali agli altri processi con una richiesta-reply
- Ciò implica che:
  - per ogni “lettura” abbiamo  $2n$  messaggi scambiati
  - per entrare in sezione critica servono 3 scambi di messaggi (tre letture) che costano in totale  $6n$  messaggi
  - la latenza per un singolo scambio di messaggi è uguale al tempo impiegato all'accoppiata canale-processo più lenti

# Progettare algoritmi in un sistema distribuito

- Lasciare la soluzione client-server per una soluzione in cui tutti cooperano...una soluzione decentralizzata
  
- L'idea a questo punto viene **ribaltata**:
  - l'idea è che il processo non “acquisisce” il numero dagli altri ma arriva con un numero già in tasca
  - Fa quindi una richiesta allegando questo numero
  - Gli altri processi gli dicono se può entrare o no

# Algoritmo di Lamport (logical clocks) /modello

- Sistema asincrono
- No guasti: processi corretti e canali affidabili (nessun messaggio perso, cambiato, spurio)
- Topologia della rete: canali punto-punto per ogni coppia di processi (rete completamente interconnessa)
- Inoltre assunzione aggiuntiva: canali FIFO

# Algoritmo di Lamport

## *Strutture dati:*

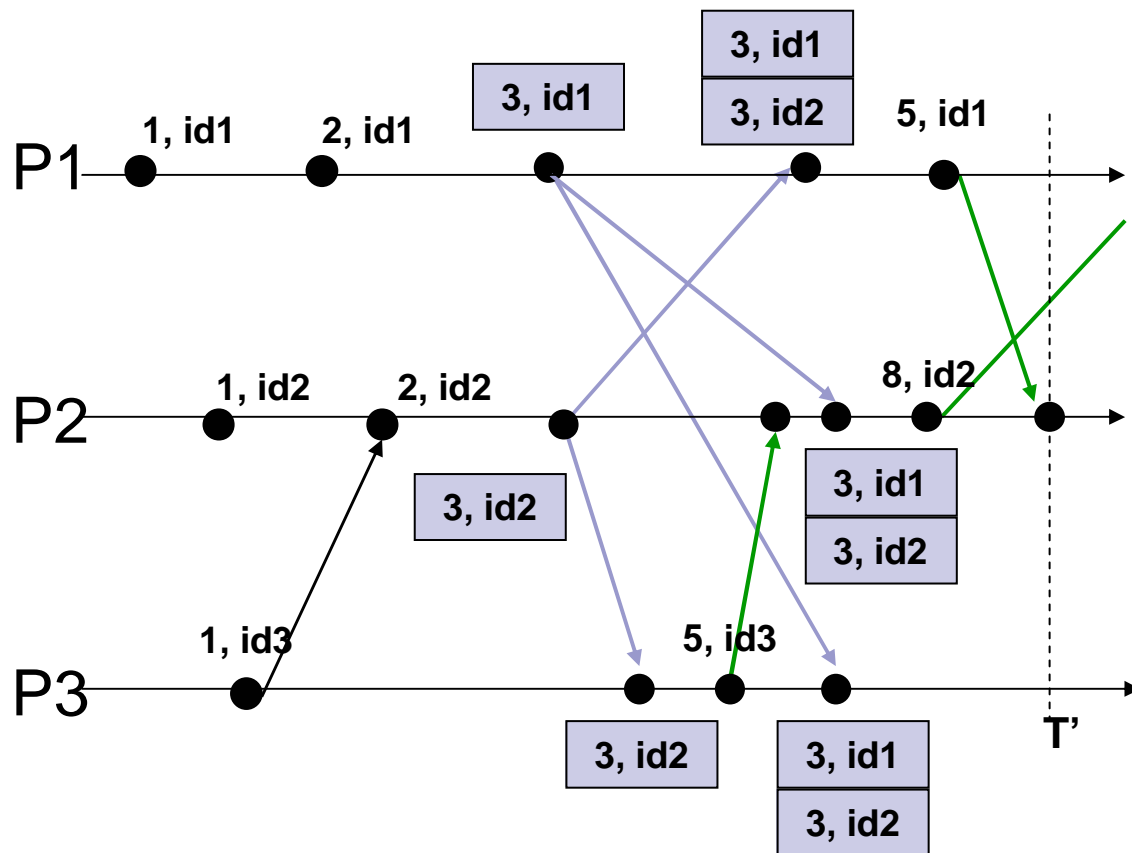
Ogni processo mantiene un clock logico per *timestamping scalare* ed una *coda* (per memorizzare le richieste di accesso alla sezione critica).

**NOTA:** gli eventi di invio/ricezione propri dell'algoritmo di mutua esclusione sono eventi rilevanti insieme a quelli della generica computazione (tutti devono essere etichettati con il clock logico).

## *Regole dell'algoritmo:*

- *Richiesta sezione critica:*  $P_i$  manda un msg di richiesta+timestamp a tutti gli altri processi e aggiunge la richiesta+timestamp alla coda
- *Ricezione di una richiesta:* la richiesta (con il suo timestamp) sono memorizzati nella coda ed un msg di ack è inviato indietro
- *Rilascio sezione critica:*  $P_i$  manda un msg di release a tutti gli altri ed elimina la richiesta dalla coda;
- *Ricezione di una release:* la richiesta è eliminata dalla coda
- *Accesso alla sezione critica se e solo se:*
  - Il processo ha una richiesta in coda con timestamp  $t$
  - $t$  è il timestamp più piccolo tra quelli presenti in coda
  - Il processo ha già ricevuto da ogni altro processo una msg (ack) con timestamp più grande di  $t$

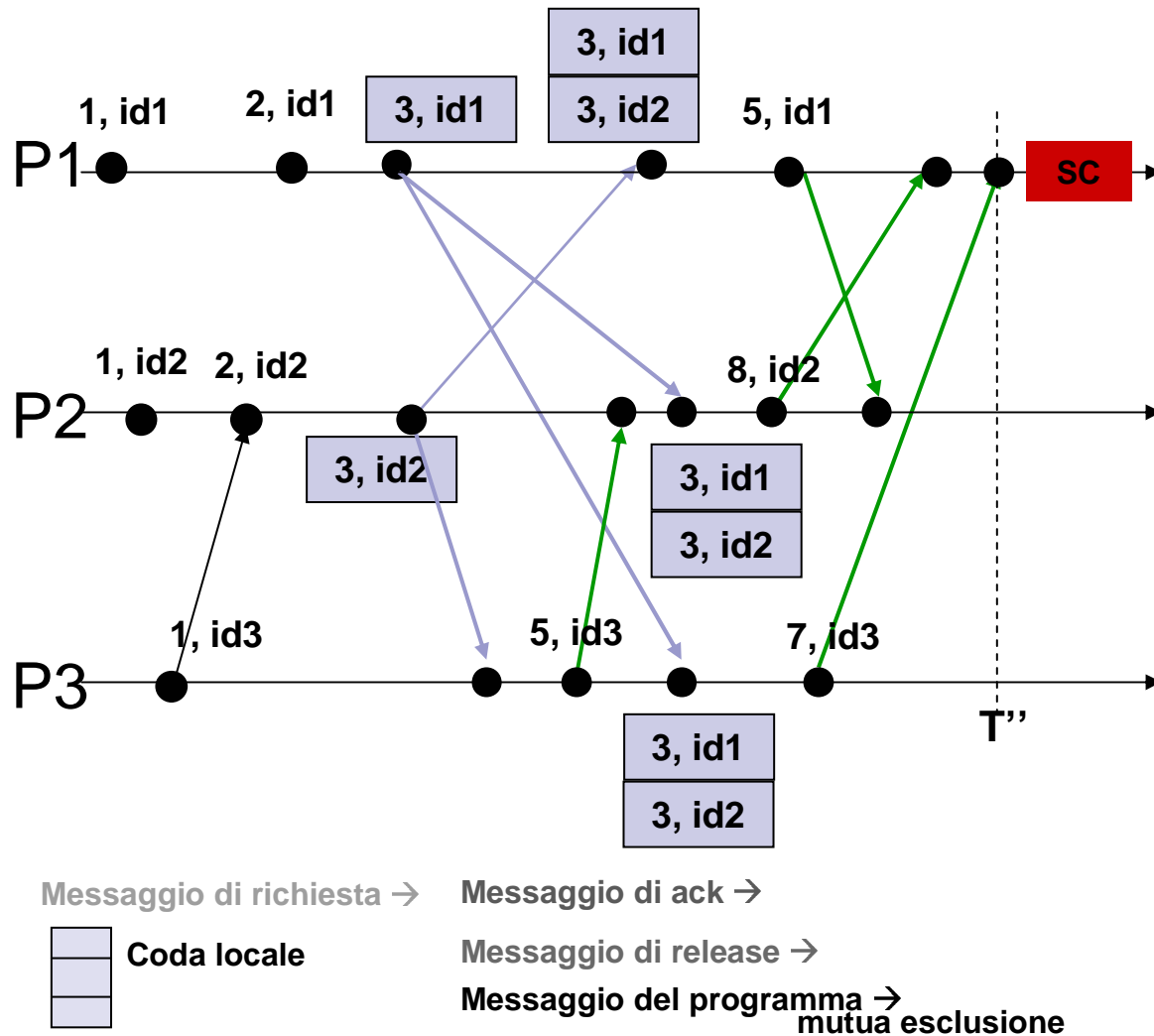
# Algoritmo di Lamport/esempio



## Tempo T':

- P2 ha ricevuto due ack con timestamp ([5,id3], [5,id1]) più grandi di quello relativo alla sua richiesta ([3,id2])
- P2 ha in coda una richiesta con timestamp più piccolo ([3,id1])
- P2 non può ancora accedere alla sezione critica!

# Algoritmo di Lamport/esempio



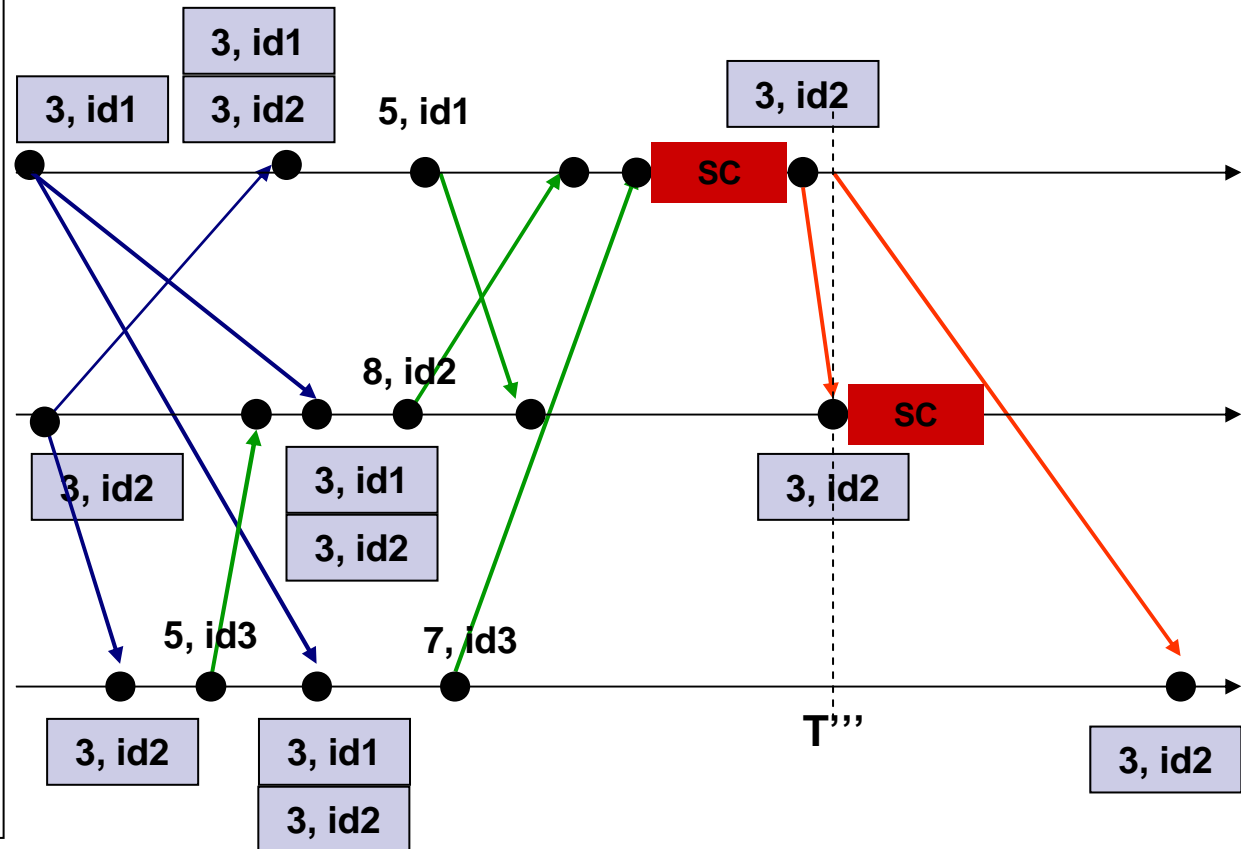
**Tempo T'':**

- P1 ha ricevuto due ack con timestamp ([8,id2], [7,id3]) più grandi di quello relativo alla sua richiesta ([3,id1])
- P1 non ha in coda una richiesta con timestamp più piccolo di [3,id1]
- **P1 accede alla sezione critica!**

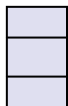
# Algoritmo di Lamport/esempio

## Tempo T''':

- P2 ha ricevuto due ack con timestamp ([5,id3], [5,id1]) più grandi di quello relativo alla sua richiesta ([3,id2])
- P2 riceve il msg di release da P1: elimina dalla coda la richiesta con timestamp [3,id1]
- **P2 accede alla sezione critica!**



Messaggio di richiesta →



Coda locale

Messaggio di ack →

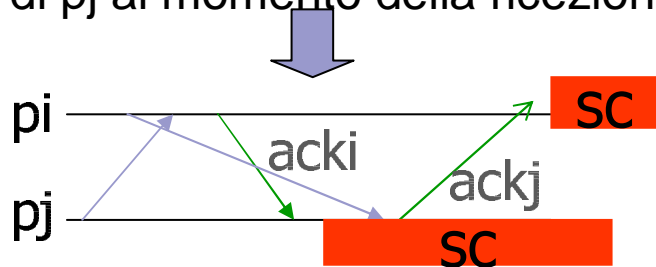
Messaggio di release →

Messaggio del programma →  
mutua esclusione

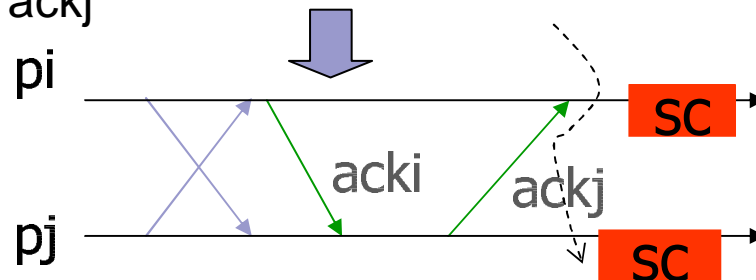
# Algoritmo di Lamport/correttezza

## Safety (sketch)

- Supponi per contraddizione che due processi  $p_i$  e  $p_j$  entrino contemporaneamente in sezione critica  $\Rightarrow$  entrambi i processi  $p_i$  e  $p_j$  hanno inviato un messaggio di ack all'altro:  $ack_i$  e  $ack_j$  tali che
  - $ack_i$  contiene un timestamp più grande di  $t_j$  ( $t_j$  =timestamp della richiesta di  $p_j$ ) **(1)**
  - $ack_j$  contiene un timestamp più grande di  $t_i$  ( $t_i$ =timestamp della richiesta di  $p_i$ ) **(2)**
- Dalla **(1)** e dalla proprietà FIFO dei canali segue che  $p_j$  ha già ricevuto la richiesta di  $p_i$  al momento della ricezione di  $ack_i$
- Dalla **(2)** e dalla proprietà FIFO dei canali segue che  $p_i$  ha già ricevuto la richiesta di  $p_j$  al momento della ricezione di  $ack_j$



questo scenario non si può verificare (violazione FIFO)!



da questo scenario si vede che  $p_i$  e  $p_j$  hanno in coda tutt'e due le richieste prima di entrare in sezione critica

Dalla proprietà di ordine totale dei timestamp si arriva subito alla contraddizione.

# Algoritmo di Lamport/performance

- richiede  $3(N-1)$  per l'invocazione di una sezione critica:  $N-1$  messaggi di richiesta,  $N-1$  messaggi di ack,  $N-1$  messaggi di release
  
- C'è un tempo di ritardo di due messaggi per aver il permesso di accesso alla sezione critica
  
- Ricart-Agrawala: la struttura è la stessa di Lamport, ma combinando la funzionalità di messaggi di ack e release si risparmiano messaggi. Non occorre FIFO.
  - MECCANISMO BASE: il messaggio di ack è inviato solo quando tutte le richieste che precedono quella corrente sono state servite
  - Quindi Ricart-Agrawala permette di usare solo  $2(N-1)$  messaggi per l'invocazione della sezione critica

# Algoritmo di Ricart-Agrawala

Local variable

#replies: integer initially set to zero  
 state: in set {requesting, CS, NCS} initially set to NCS  
 Q: queue of pending request {T,i} initially set to empty  
 Last\_req: integer ;  
 Num: integer initially set to zero

begin

1. state:= requesting
  2. num:=num+1; last\_req:=num;
  3. **send** REQUEST(last\_req) **to** p1..pn
  4. **wait until** #replies=(n-1)
  5. state:= CS
  6. CS
  7. **send** REPLY **to** any request in Q; Q:= $\emptyset$ ; state:=NCS; #replies:=0
- end

Upon the receipt of REQUEST(t) from process j

1. **if** state=CS **or** (state=requesting **and** {last\_req, i } < {t,j})
2. **then** insert.Q({t, j})
3. **else send** REPLY **to** Pj
4. num:=max(t,num)

Upon the receipt of REPLY from process j

1. #replies++;