Teoria della Replicazione

Sistemi Distribuiti a.a. 2003/2004

Prof. Roberto Baldoni, Ing. Alessia Milani

Argomenti (1)

- Introduzione alla replicazione:
  - Motivi
    - Performance
    - Disponibilità
    - Tolleranza ai guasti
  - Requisiti
    - Trasparenza
    - Consistenza
- Replicazione software:
  - Linearizability

Argomenti (2)

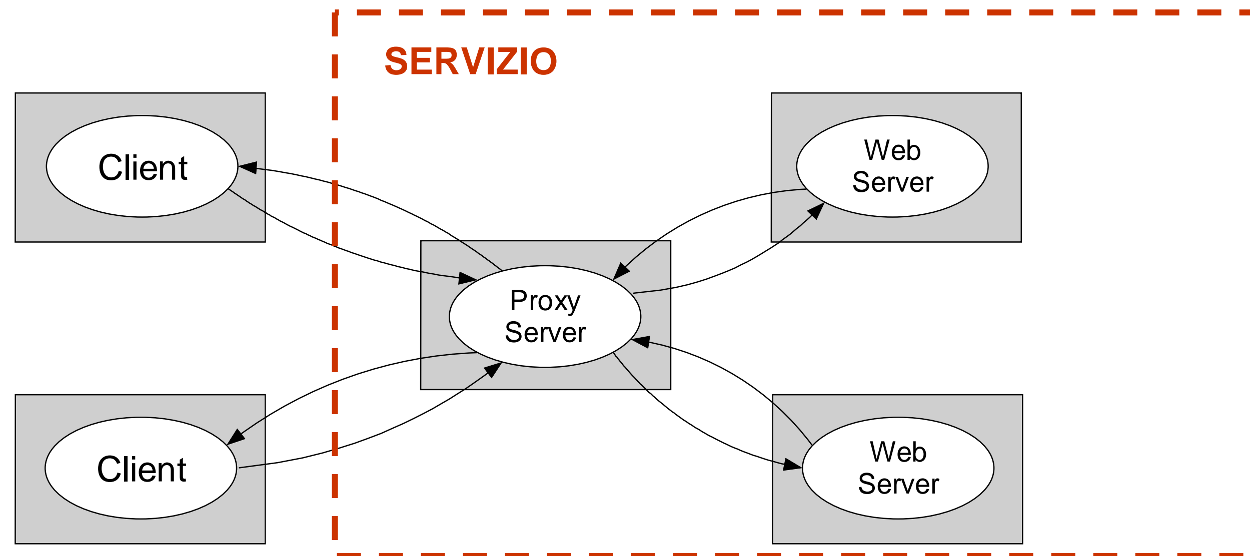
- Tecniche di replicazione
  - Primary Backup
  - Active Replication
- Memorie distribuite:
  - Sequential consistency
  - Causal consistency
  - PRAM
  - Protocolli:
    - Ahamad, limiti
    - BMT

Motivi della replicazione

- Performance
- Disponibilità
- Tolleranza ai guasti

Performance

- Proxy server e browser caching: ridurre la latenza di accesso a oggetti web.



- DNS : distribuire il carico di lavoro.

Disponibilità

- Il servizio deve essere accessibile con tempo di risposta ragionevole per una frazione di tempo prossima al 100%
  - Es. Sia **O** un oggetto replicato su ***n*** server la cui probabilità di guastarsi sia ***p*** (indipendente dalla probabilità di guasto degli altri server)***.***

Disponibilità di **O**:

$$1 - p^n$$

Tolleranza ai guasti

- Assicurare un comportamento corretto del servizio a fronte di un certo numero e tipo di guasti.

- Se su f+1 server f si guastano (crash), 1 garantisce il servizio.

Modello Base per la Replicazione (1)

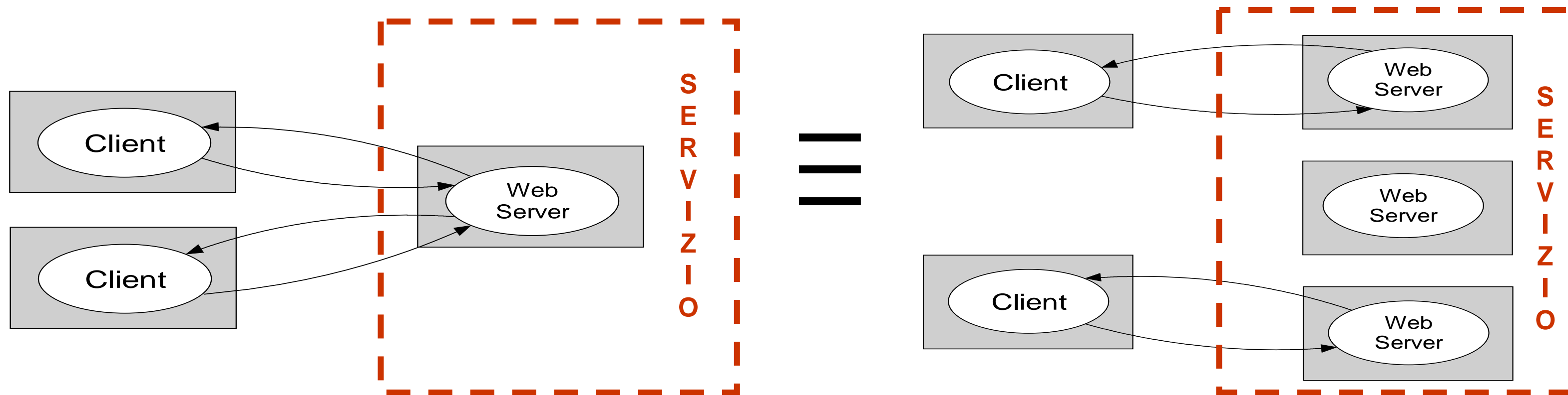
- Insieme X di oggetti (file, variabile, oggetto Java, Web page…)
- Un oggetto $x \in X$ ha uno stato
- I processi accedono allo stato dell'oggetto attraverso operazioni *o*.
- Un'operazione *o* di un processo p_i su un oggetto x è una coppia invocazione-risposta, [x *o*(arg) p_i]/[ok (res) p_i].
- p_i invoca *o* e resta bloccato fino a quando ottiene la corrispondente risposta.

Modello Base per la Replicazione (2)

- Il set di operazioni che gestiscono l'oggetto ne determinano la semantica
- Ogni "oggetto logico" x corrisponde a diverse copie fisiche, "repliche", distribuite nel sistema, denotate $x^1, x^2, \ldots x^l$.
- Le invocazioni sulla replica x^k collocata nel sito s sono gestite dal gestore della replica anch'esso in s.
- Questo modello generale può essere applicato:
  - Ambiente client-server, il gestore della replica è il server
  - Ambiente a processi, un processo è al tempo stesso client e gestore della replica

Trasparenza

- Il client deve credere di interagire con l'oggetto logico.
- La replicazione non cambia:
  - Il modo in cui un client invoca un'operazione
  - Il modo in cui vengono restituite le risposte



Consistenza(1)

- Le operazioni eseguite su una collezione di oggetti replicati devono produrre risultati che rispettano le specifiche di correttezza per tali oggetti.
- Le specifiche di correttezza dipendono dalla semantica del servizio.
- **Intuitivamente**: un servizio basato sulla replicazione è corretto se il client non è in grado di distinguere dal servizio ottenuto da un’implementazione con oggetti replicati e quello garantito da una singola replica corretta.

Consistenza (2)

- Operazioni su un oggetto possono essere invocate concorrentemente da più processi.
- E' necessario dare un significato ai possibili "interleaving" delle operazioni.

Consistenza Esempio: Conto Bancario

- Supponiamo di aver un sistema di replicazione ANOMALO:
  - 2 gestori delle repliche su computer diversi A e B.
  - Ogni gestore mantiene le repliche di due conti bancari x,y.
  - I client leggono e aggiornano il conto accedendo prima alla propria copia locale e poi se questa è guasta all’altra.
  - I gestori della replica propagano gli aggiornamenti dopo aver restituito il risultato al client.
  - I conti inizialmente sono in rosso 0 $.

Consistenza Esempio: Conto Bancario

- **BEA** aggiorna **x** ad **1** $ e poi prova ad aggiornare **y** a **2** $ ma si accorge che B si è guastato
- **BEA** aggiorna **y** su A .
- **NOTA**: per il guasto B non ha inviato l'update di **x**.
- **ADA** legge il valore dei due conti **x=0** ed **y=2** **COMPORTAMENTO ANOMALO**

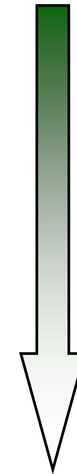
Criteri di Consistenza

- **“ Quale valore deve restituire una operazione su un oggetto? ”**

Alcuni criteri di consistenza

- Linearizability
- Sequential consistency
- Causal consistency
- PRAM

stringente



Criteri di Consistenza

Criterio di consistenza	Applicazione
Linearizability	Replicazione software
Atomico	Memorie distribuite
Sequenziale	Memorie distribuite
Causale	Memorie distribuite
PRAM	Memorie distribuite

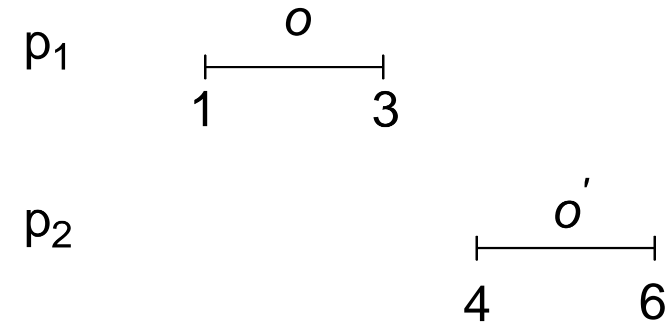
Replicazione Software

Linearizability (1)

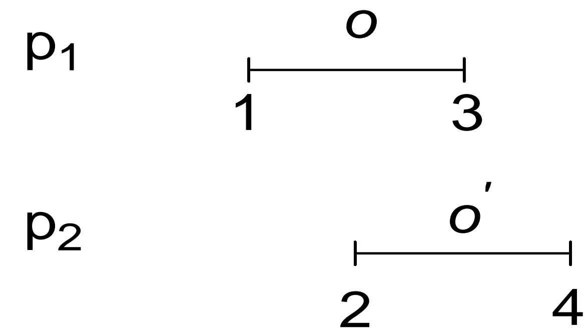
- Assunzioni:
  - Global real-time clock
  - $t_{inv}(o)$ istante in cui p_i invoca *o* sull'oggetto x
  - $t_{res}(o)$ istante in cui p_i riceve la corrispondente risposta
- Illusione: ogni operazione *o* eseguita dai processi concorrenti ha effetto istantaneamente in un tempo compreso tra $t_{inv}(o)$ e $t_{res}(o)$.

Linearizability (2)

- Due operazioni *o* e *o'*,sono dette sequenziali, denotato $o<o'$, se la risposta di *o* precede l'invocazione di *o'*. ($t_{res}(o)$ minore $t_{inv}(o')$)



- Due operazioni *o* e *o'*,sono dette concorrenti, denotato $o||o'$, se $\neg(o<o')$ e $\neg(o'<o)$.



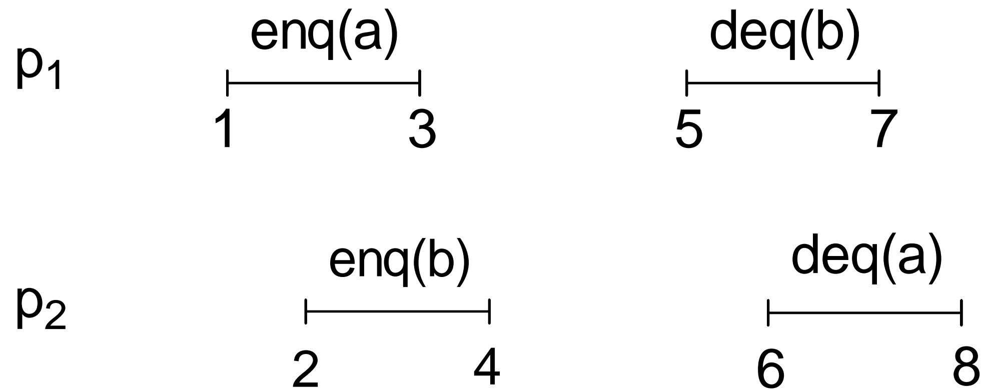
Linearizability(3)

- Un'esecuzione E è linearizzabile se esiste una sequenza S contenente tutte le operazioni in E tali che:
  - ∀ due operazioni *o* e *o'*,tali che $o<o'$, *o* appare prima di *o'* in S (real-time);
  - La sequenza S rispetta le specifiche di una (singola) copia corretta dell'oggetto.

Linearizability: Esempio (1)

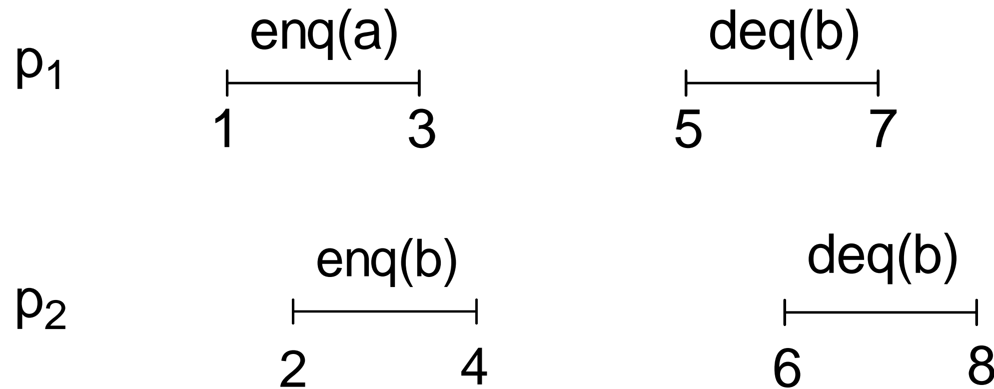
- L'oggetto x è una coda FIFO inizialmente vuota
- Le operazioni su x sono l'inserimento di un elemento in coda, *enq*, ed il prelievo del primo elemento inserito in coda, *deq*.
- Consideriamo la seguente esecuzione E

Linearizability: Esempio (2)



- o_1=enq(a); o_2=enq(b); o_3=deq(b); o_4=deq(a);
- S= o_2, o_1, o_3, o_4
- S è *legale* perché rispetta le specifiche di una coda FIFO: b è messo in coda prima di a, quindi la prima operazione di estrazione restituisce b e la seconda a.

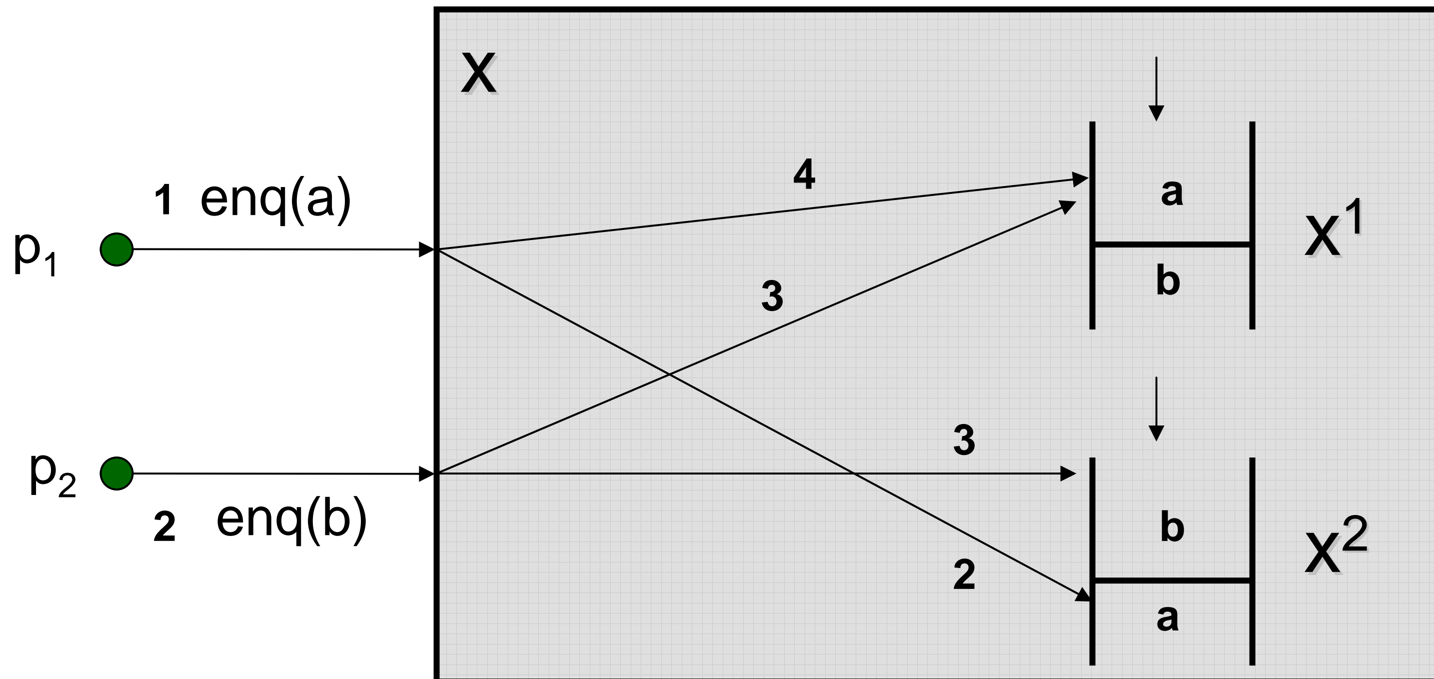
Linearizability: Esempio (2)



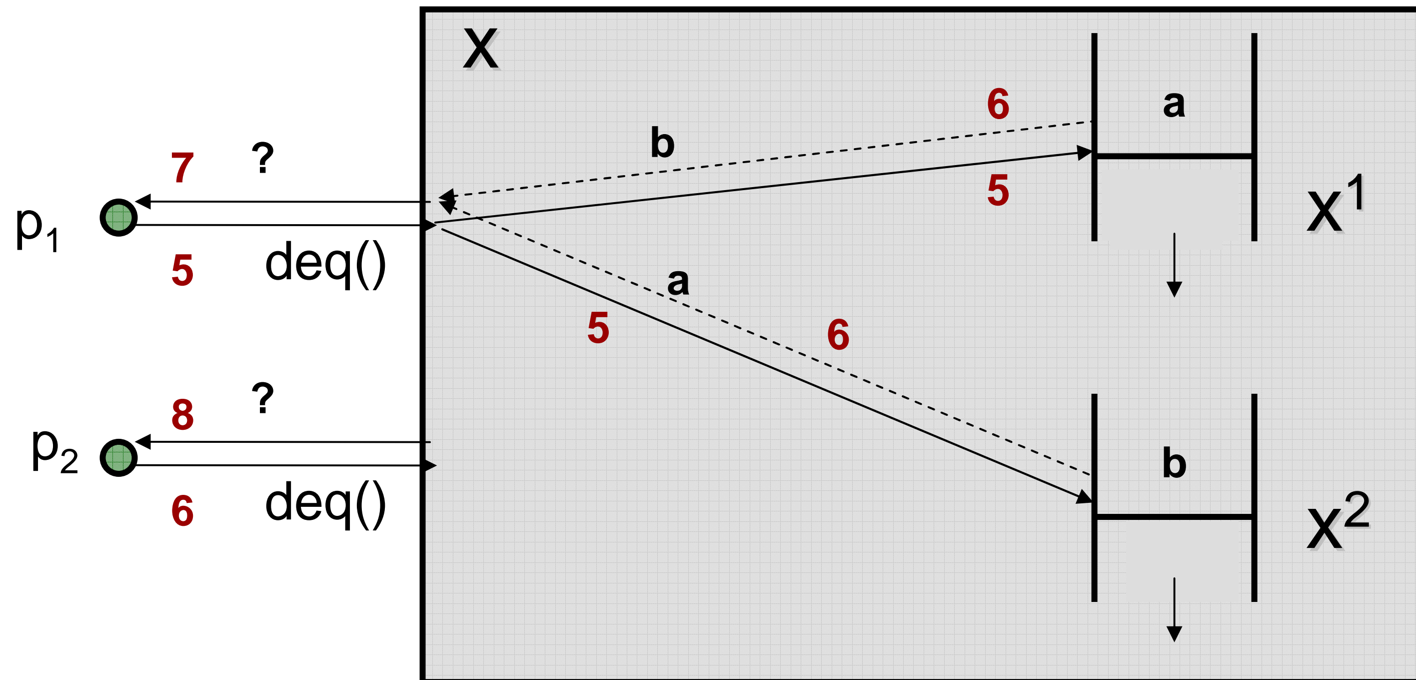
- o_1=enq(a); o_2=enq(b); o_3=deq(b); o_4=deq(b);
- Esecuzione non linearizzabile
- Non esiste una S che rispetta le specifiche di una coda FIFO.

Assicurare Linearizability(1)

- Supponiamo x, coda FIFO, implementata con 2 repliche (tollerante i guasti), x^1 e x^2.



Assicurare Linearizability(2)



- Entrambi i processi devono scegliere quale valore considerare, se la scelta è la stessa ed è b la linearizability NON VIENE RISPETTATA.

Assicurare Linearizability(3)

- Per assicurare Linearizability le seguenti condizioni devono essere rispettate:
  - Ordine: invocazioni che vengono da client distinti devono essere trattate nello stesso ordine da ogni replica
  - Atomicità: se una replica del server esegue un'invocazione allora tutte le repliche non guaste devono eseguire tale invocazione.

Tecniche di replicazione

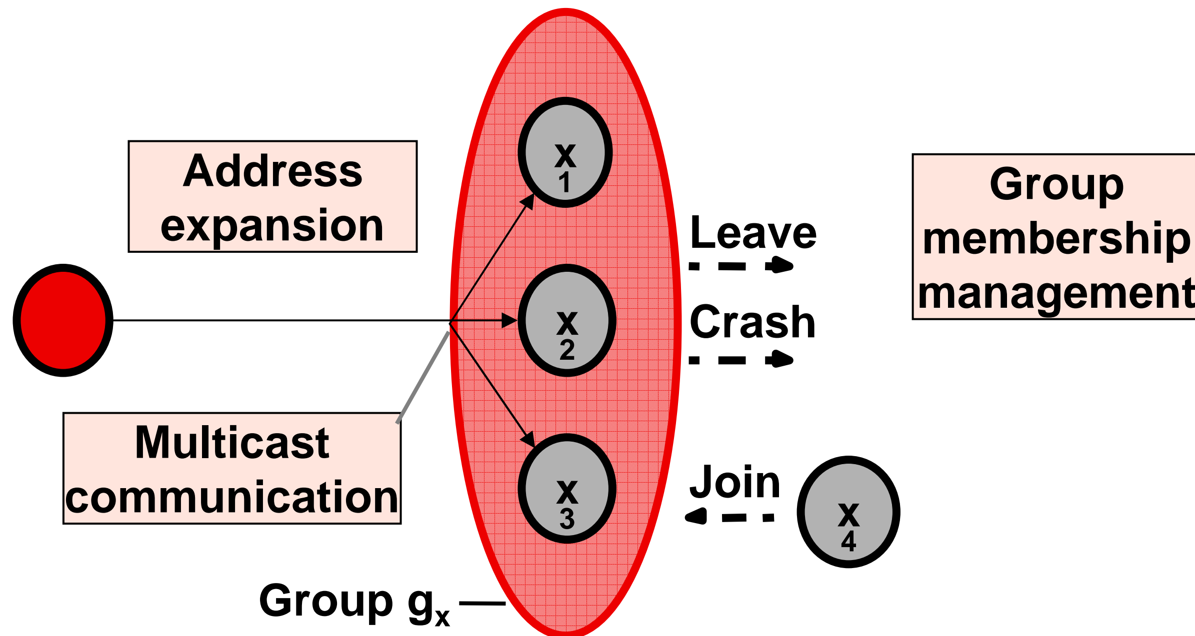
- Due fondamentali tecniche di replicazione che assicurano linearizability sono:
  - Primary Backup
  - Active Replication

Modello di sistema

- N processi sequenziali $P=\{p_1,p_2...p_n\}$
- Modello di comunicazione a scambio di messaggi
- Canali asincroni e affidabili (Perfect link)
- Un processo può essere:
  - **corretto**, si comporta secondo le sue specifiche
  - **non corretto**, guasto per crash.

Primary-Backup: Gruppi Dinamici

- La tecnica Primary-backup si basa su gruppi dinamici



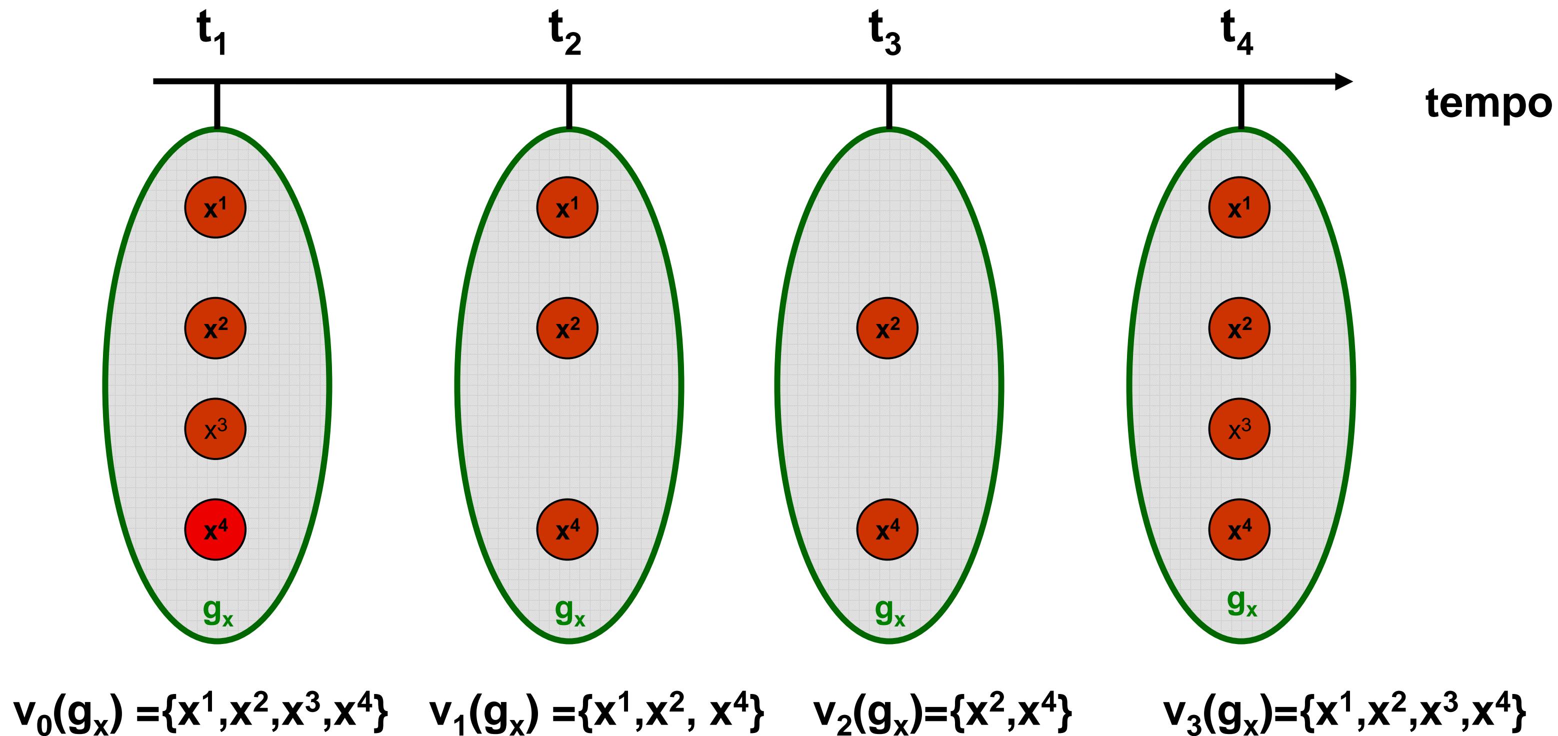
Gruppi dinamici (1)

- Un gruppo dinamico è un gruppo in cui i membri cambiano durante il ciclo di vita del sistema.
- Se una replica x^k si guasta viene rimossa dal gruppo.
- Se x^k viene ripristinata, viene riaggiunta al gruppo g_x.
- g_x cambia nel tempo: concetto di "*vista*" usato per modellare l'evolvere di g_x.

Gruppi dinamici (2)

- *Vista:* lista dei membri correnti del gruppo, ognuno associato ad un identificatore univoco.
- Una nuova vista è generata quando un processo è aggiunto/escluso al/dal gruppo
- $v_o(g_x)$ insieme iniziale dei membri di g_x
- $v_i(g_x)$ *i*-esimo insieme dei membri di g_x

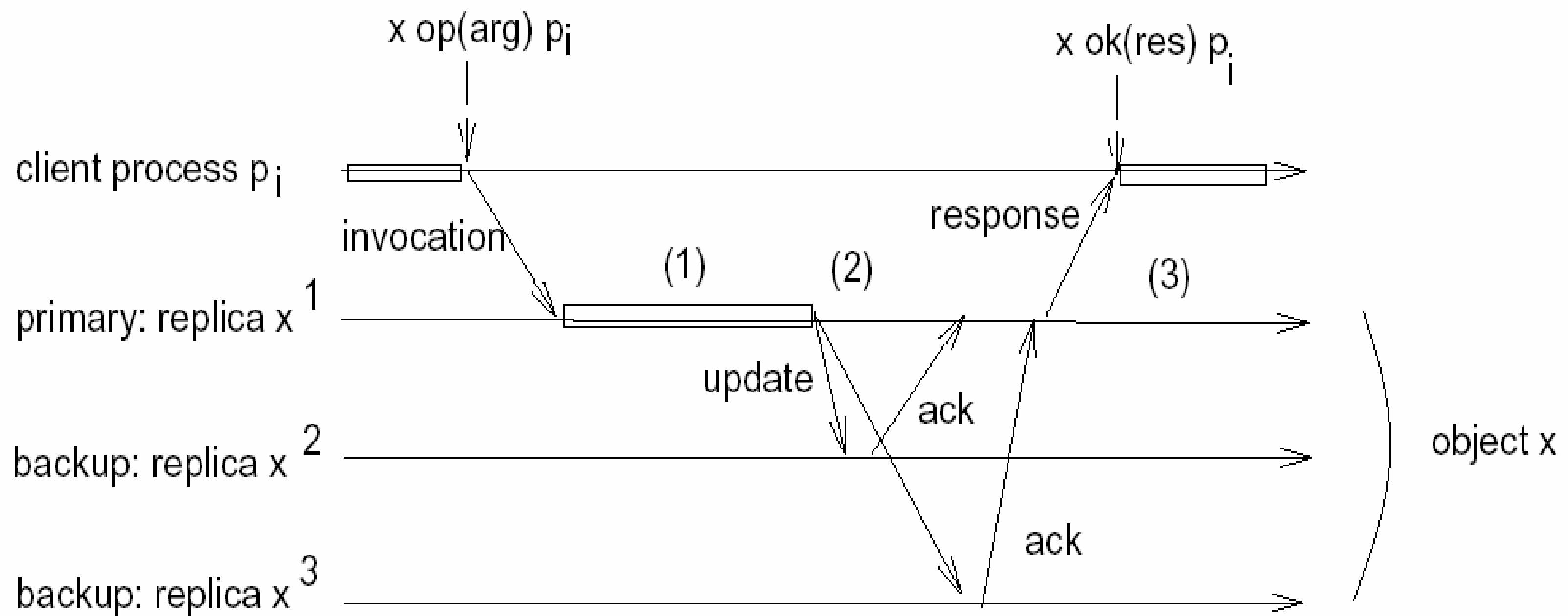
Gruppi Dinamici Esempio



Primary Backup

- Una delle repliche, detta *primary*, ha un ruolo speciale: riceve le invocazioni dal client e restituisce la risposta.
- Dato un oggetto x, la primary di x è denotata *prim(*x*).*
- Le altre repliche sono dette *backup*.
- Le *backup* interagiscono solo con la *prim(*x*).*

Primary Backup: Scenario



Primary Backup in Assenza di Crash (1)

- p_i invoca o(arg) su x.
- *prim(x)* riceve l'invocazione ed esegue l'operazione.
- Dopo l'esecuzione *prim(x)* ha aggiornato il suo stato e la risposta, res, è disponibile.
- *prim(x)* invia dei messaggi di update, (invId,res,state-update), ai *backup*.

Primary Backup in Assenza di Crash (2)

- Quando ricevono gli update, i *backup* aggiornano il loro stato e inviano l'ack al *prim(*x*)*.
- La *prim(*x*) a*spetta gli ack di tutti i *backup* corretti e poi invia la risposta, res, al client.
- L'ordine in cui *prim(*x*)* riceve le invocazioni determina l'ordinamento totale delle invocazioni sull'oggetto.

Primary Backup in Presenza di Crash (1)

- Distinguiamo tre casi del *primary:*
  1. Guasto prima di inviare i messaggi di update
  2. Guasto dopo aver inviato i messaggi di update ma prima di aver ricevuto gli ack
  3. Guasto dopo che il client ha ricevuto la risposta.
- In tutti e tre i casi si deve eleggere un nuovo *primary*, *elezione del leader*.

Primary Backup in Presenza di Crash (2)

- Nei casi 1 e 2, il client non ottiene una risposta e quindi sospetta il guasto.
- Il 3 caso è trasparente al client.
- Dopo aver conosciuto la nuova identità della *primary* il client reinvoca l'operazione:
  - Nel caso 1 l'invocazione è considerata come se fosse nuova,
  - Nel caso 2 la situazione è più complessa.

Primary Backup in Presenza di Crash (3)

- Caso 2. Garantire atomicità: l'update è ricevuto da tutti o da nessuno.
  - Nessun backup ha ricevuto l'update ⇒ si torna al caso 1
  - Tutti i backup hanno ricevuto l'update ⇒ ATTENZIONE l'operazione è stata eseguita ma il client non ha avuto risposta, la seconda invocazione deve essere ignorata. Per evitare inconsistenze si usano le informazioni (invId,res).

Primary Backup in Presenza di Crash (4)

- Quando la nuova *primary* riceve l'invocazione con lo stesso invId spedisce immediatamente la risposta al client.

Primary-Backup: Problemi

- LEADER ELECTION: elezione di una nuova primary quando quella corrente si guasta per crash.
- ATOMICITA': gli update devono essere ricevuti da tutte le repliche di backup o da nessuna.

Primary-Backup : Leader Election (1)

- GARANTIRE UNA NUOVA PRIMARY QUANDO QUELLA CORRENTE SI GUASTA.
- Supponiamo che ci sia una regola R che ordina le repliche all’interno di una vista.
- La primary può essere definita come la prima replica nella vista corrente in accordo ad R.
- Consegna ordinata delle viste.

Primary-Backup Leader Election (2)

- Es. R=ordine crescente dell'identificativo della replica:
  - x^1,x^2,x^4 *OK*
  - x^2,x^1,x^4 *ERRORE*
- Definizione primary:
  - $v_0(x^1,x^2,x^4)$, primary x^1
  - $v_1(x^2,x^4)$, primary x^2

Consegna Ordinata delle Viste (1)

- Ogni vista è consegnata da ogni processo corretto in $g_x \Rightarrow$ ogni replica è in grado di conoscere l'identità della primary.
- **NOTA**: data la sequenza di viste che definiscono la storia di un gruppo è IRRILEVANTE se una replica è stata eliminata dal gruppo perchè realmente guasta oppure perché erroneamente sospettata.

Consegna Ordinata delle Viste (2)

- Ogni processo nel gruppo consegna una serie di *viste* $v_0(g)=\{p\}$, $v_1(g)=\{p,p'\}$, $v_2(g)=\{p\}$ ecc.
- Poiché diversi cambiamenti nella composizione del gruppo possono avvenire concorrentemente, il sistema impone un ordinamento sulla sequenza delle viste fornite ad ogni processo.

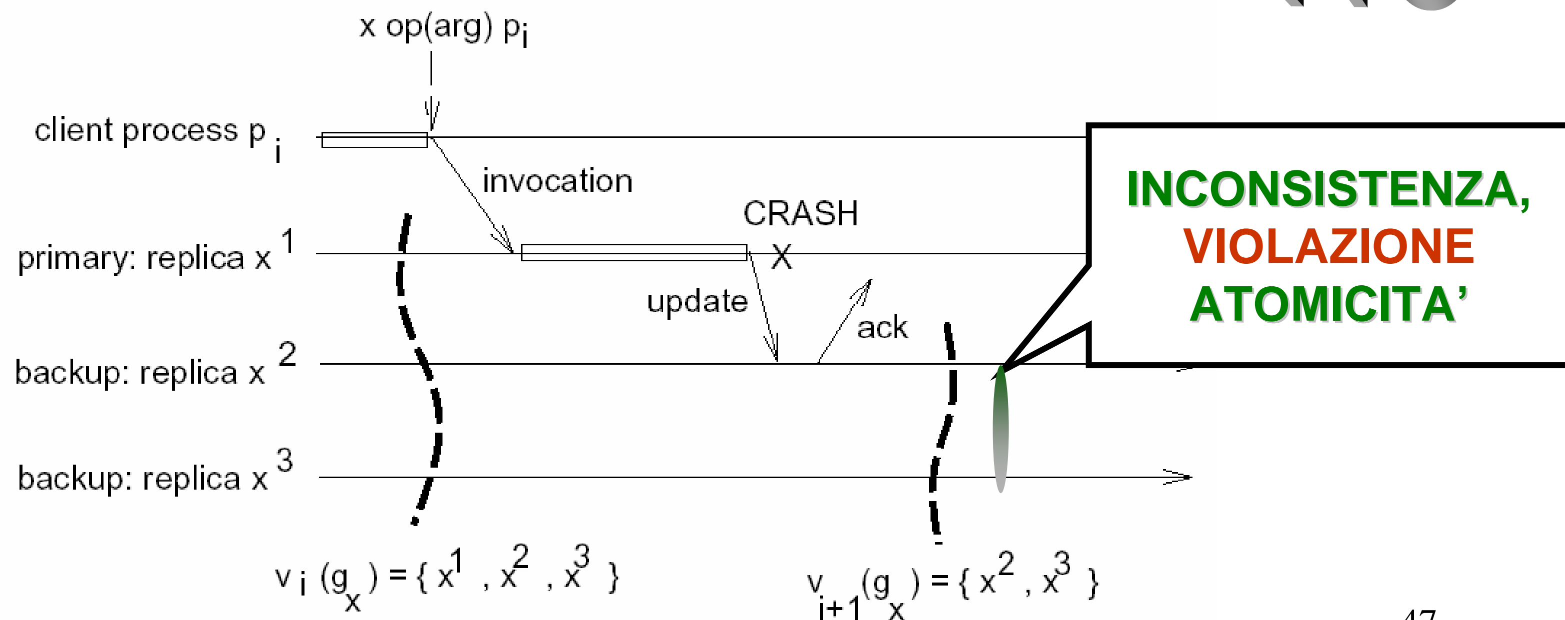
Consegna Ordinata delle Viste: Specifica

- Quando un processo riceve una vista la bufferizza fino a quando tutti i membri del gruppo sono d'accordo nel fare la corrispondente consegna.
- Assunzioni di base per la consegna di una vista:
  - Order: se un processo p consegna v(g) e poi v'(g), nessun altro processo può consegnare v'(g) e poi v(g).
  - Integrity: se un processo p consegna v(g) allora p $\in$v(g).
  - Non-triviality: se un processo q si unisce al gruppo ed è o diventa indefinitamente raggiungibile dal processo p$\neq$q, allora alla fine q appartiene sempre alle *viste* consegnate da p.

Primary-Backup
Problema Atomicità

- E’ sufficiente garantire ordinamento delle invocazioni e delle viste per garantire CORRETTEZZA?

NO



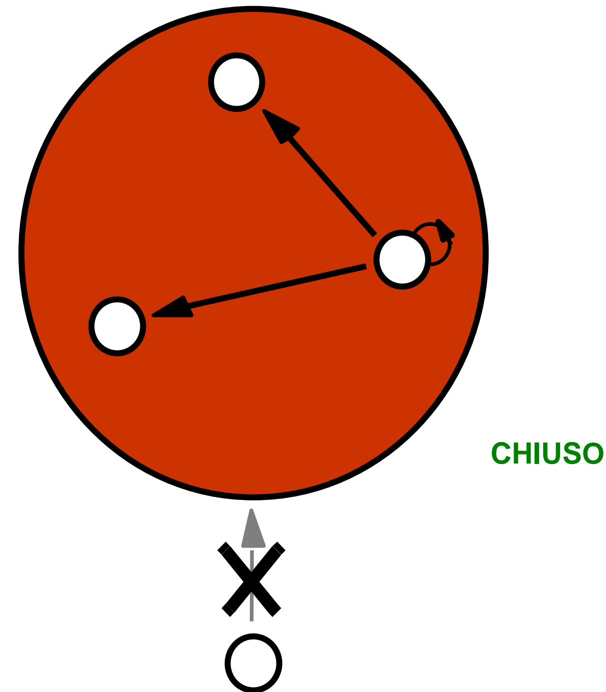
Primary-Backup
Soluzione Atomicità

- Il messaggio di update della primary deve essere ricevuto da tutte o da nessuna delle repliche *backup*.

VIEW
SYNCHRONOUS
MULTICAST

Gruppi Chiusi

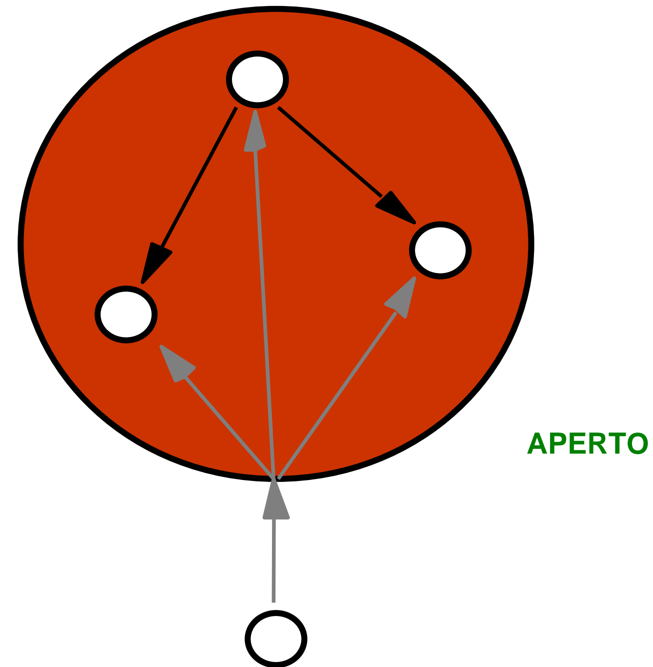
- **CHIUSO**:
  - soltanto i membri del gruppo possono spedire messaggi al gruppo
  - ogni membro consegna anche i messaggi spediti da se stesso



Gruppi Aperti

- **APERTO:**

  Un processo esterno al gruppo può inviare messaggi al gruppo stesso.



View-synchronous Multicast: Specifica

- Estende la semantica del reliable multicast prendendo in considerazione i cambiamenti delle viste del gruppo.
- Garanzie:
  - **Agreement**: processi corretti consegnano lo stesso set di messaggi in ogni data vista. Se un processo fa la delivery di un messaggio *m* nella vista *v(g)* e poi consegna la nuova vista *v'(g)*, allora un qualsiasi processo membro di $v(g) \cap v'(g)$ deve consegnare *m* nella vista *v(g)*.
  - **Integrità**: Un processo non può consegnare un messaggio *m* due volte.
  - **Validità** (gruppi chiusi): processi corretti consegnano sempre i messaggi che essi stessi spediscono. In pratica: sia p un processo corretto che consegna un messaggio m nella vista v(g). Se un qualche processo $q \in v(g)$ non consegna m, allora la nuova vista v'(g) che p consegnerà non conterrà q.

View-synchronous multicast: Implementazione

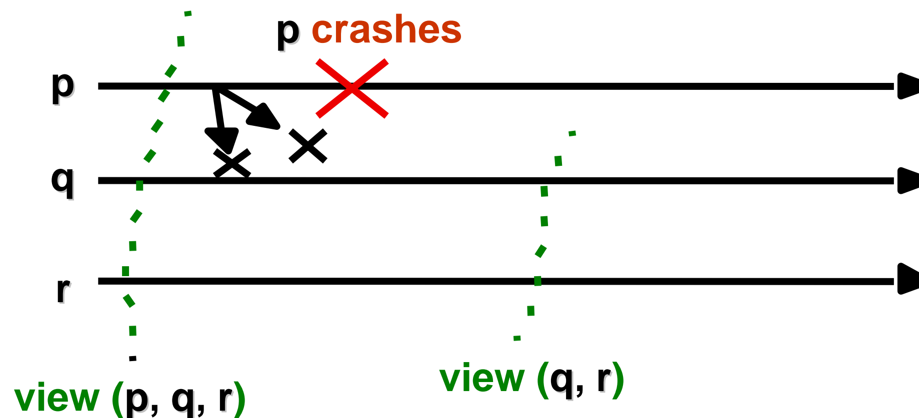
- Consideriamo un gruppo dinamico g_x ed una sequenza di viste …,$v_i(g_x)$, $v_{i+1}(g_x)$,….
- Sia $t^k(i)$ il tempo locale in cui una replica x^k consegna un messaggio contenente la composizione della vista $v_i (g_x)$.
- Da $t^k(i)$ in poi, il timestamp che x^k inserisce nei messaggi è i, m(i).
- Ogni messaggio con timestamp i è inviato in multicast a tutti i membri di $v_i (g_x)$.

View-synchronous multicast: Implementazione

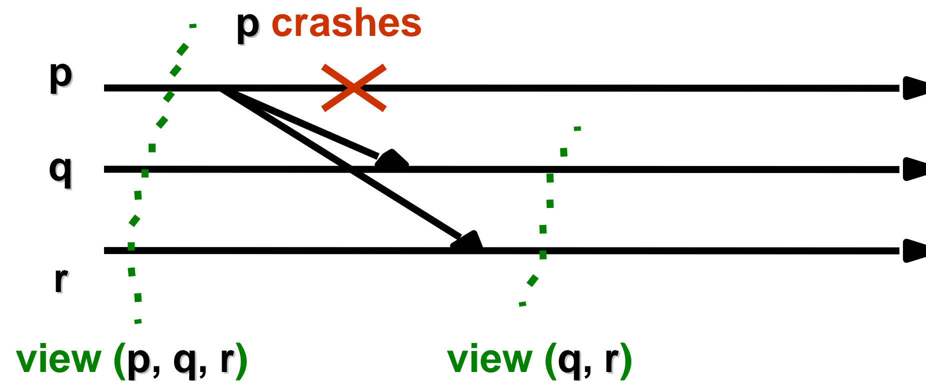
- Supponiamo che $v_{i+1}(g_x)$ sia la nuova vista $\Rightarrow$ tutti i membri $\in [v_i(g_x) \cap v_{i+1}(g_x)]$ consegneranno prima m(i) e poi la nuova vista $v_{i+1}(g_x)$ oppure nessuno consegnerà m(i).

View-synchronous multicast (Es.1)

- Gruppo: p,q,r
- p si guasta, q ed r sono corretti



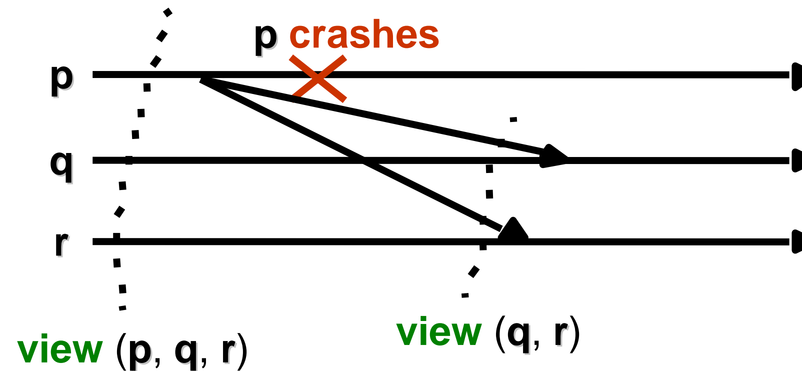
View-synchronous multicast (Es.2)



- Almeno uno dei due processi q,r ha ricevuto m quando p si guasta $\Rightarrow$ q ed r prima consegnano m e poi la nuova vista view(q,r).

View-synchronous multicast (Es. 3)

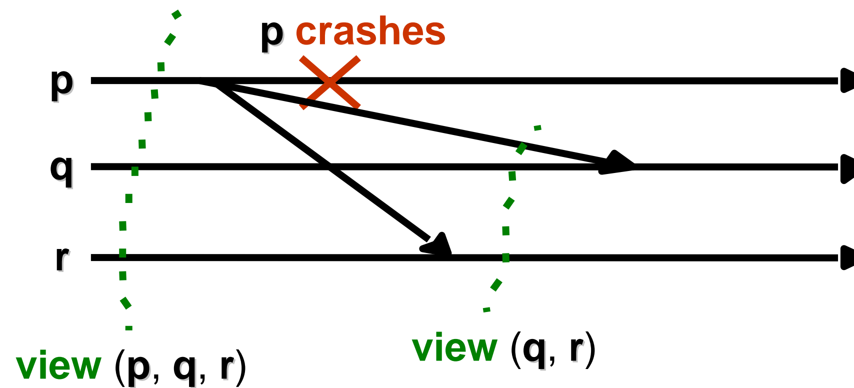
c. NO



- q ed r non possono consegnare un messaggio il cui mittente sanno essere guasto.

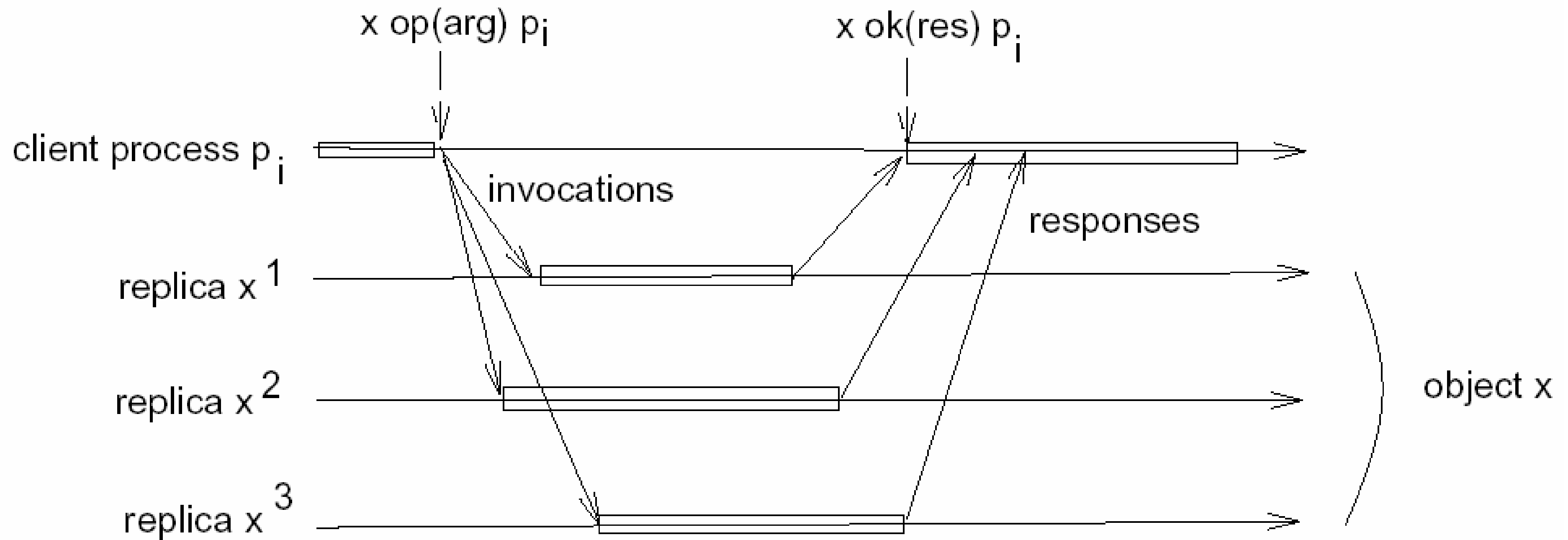
View-synchronous multicast (Es.4)

d NO.



- I processi devono rispettare uno stesso ordinamento nella consegna dei messaggi e delle viste.

Active Replication scenario



Active Replication (1)

- Tutte le repliche giocano lo stesso ruolo, non c'è un controllo centralizzato
- Quando un processo p_i invoca un'operazione sull'oggetto x :
  - L'invocazione è spedita a tutte le repliche
  - Ogni replica processa l'invocazione, aggiorna il suo stato e restituisce la risposta al client.

Active Replication (2)

- Il client aspetta una sola risposta
- Questa tecnica richiede che le invocazioni del client arrivino alle repliche non guaste in uno stesso ordine $\Rightarrow$ primitiva di comunicazione adeguata: atomic multicast.

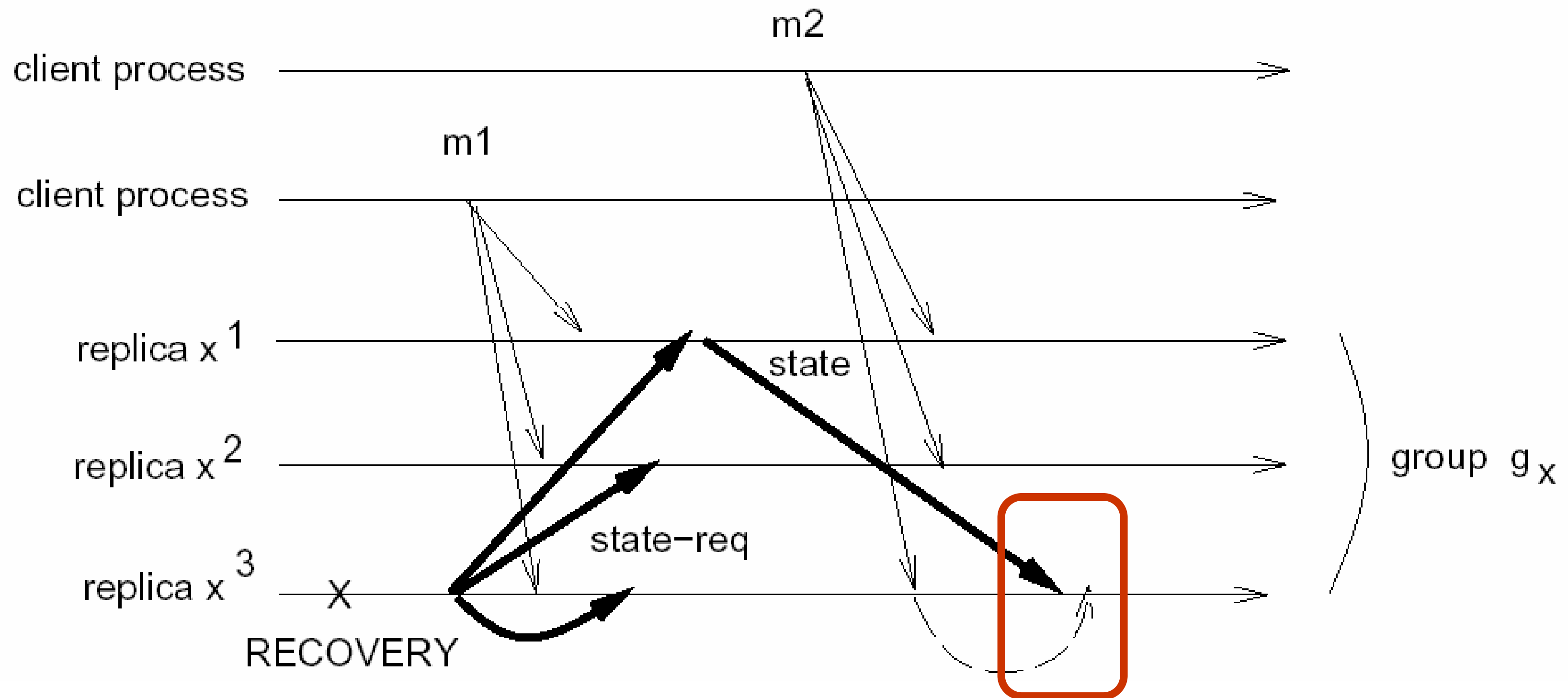
Active Replication: replica recovery

- ***Hp***: una replica x^k si guasta al tempo *t* e viene ripristinata all'istante *t'* *(t< t')*.
- x^k dovrebbe aver consegnato tutti i messaggi spediti fino a *t'* (*atomicità*).
- quando x^k viene ripristinata, viene aggiornata da un'altra replica operativa x^j attraverso il meccanismo di *State transfer*.

State Transfer: Implementazione

- x^k invia un messaggio di richiesta dello stato (state-req,x^k) attraverso una primitiva di atomic multicast.
- Ogni replica dopo aver consegnato il messaggio (state-req,x^k) invia il suo stato a x^k.
- x^k aspetta di aver consegnato il suo messaggio di richiesta dello stato.
- x^k aspetta lo stato corrente da uno dei membri del gruppo g_x .
- Nel frattempo x^k bufferizza i messaggi ricevuti dopo aver inviato (state-req,x^k).
- Dopo aver aggiornato il suo stato li gestirà.

State Transfer: Esempio



Confronto tra le tecniche (1)

- La tecnica Active Replication richiede che le operazioni sulle repliche siano deterministiche:
  - Determinismo vuol dire che l'outcome di una operazione dipende soltanto dallo stato iniziale della replica e dalla sequenza di operazioni precedentemente eseguite dalla replica.

Confronto tra le tecniche (2)

- Per la Active Replication un guasto di una replica è sempre trasparente al client (mai reinvocazione operazione).
- Per la Primary Backup, il guasto è trasparente solo se la replica è un backup.
  - Se la replica è il primary allora la latenza sperimentata dal client può essere eccessiva: non accettabile per applicazioni REAL-TIME.
- Active Replication usa più risorse della Primary Backup.