PaCu Filesystem

FS distribuito transazionale con replicazione

Gabriele Cuppone – Antonio Papa

PaCu Filesystem – SD 2009/2010

Specifiche

- Tollerante ai seguenti tipi di failure:
  - Per il client: omissioni, failstop, bizantini
  - Per il server: failstop
- Presenza di Log delle operazioni (server stateful)

Assunzioni

- Qualora un server rimanga down per più di un tempo d'attesa stabilito nella configurazione, il programma provvede ad escluderlo dalla lista server ed è compito dell'amministratore ripristinare lo stato del sistema e la consistenza del server in questione.
- Non può accadere che cadano tutte le repliche.

Architettura di Sistema

Entità

- Boot Server
- Server PR (server Primary/Replica)

Come avviene la comunicazione?

- Client ➡ Boot Server
- Client ➡ Server Primary
- Server Primary ➡ Server Replica

➡ Trasparenza alla Replicazione

Spazio dei nomi

Implementazione

- Spazio dei nomi distribuito non c'è un DNS
- Ogni server PR ha una vista generale sui domini
- I server PR hanno un vista completa dei domini che gestiscono
- Il client reperisce la lista dei domini da un qualsiasi server PR
- Il client memorizza la lista dei domini in una cache

Vantaggi

- Non abbiamo un single point of failure
- Non abbiamo un collo di bottiglia
- Tutto trasparente all'utente

Boot Server

Una breve presentazione…

- Schema leader-follower con prethreading
- Entità che fornisce la lista dei server PR al client
- Non fornisce la lista dei domini
- Utilizza 2 identificativi di versione per risparmiare banda
- Configurabile in remoto dall'amministratore
- Servizio di autenticazione per l'amministratore

Vantaggi

- Non è un single point of failure
- Risparmio di banda
- Alta configurabilità

Server PR

Il centro nevralgico del sistema…

- Schema leader-follower con prethreading
- Entità che operano sia da primary che da repliche
- Nei domini in cui è primary, gestisce le operazioni richieste dal client
- Nei domini in cui è replica, rende permanenti gli aggiornamenti del primary
- Utilizzo del protocollo primary-backup bloccante
- Gestione di un insieme ”eterogeneo” di transizioni.
- Discovery (anche in background) dei server PR che incorrono in un crash

Vantaggi

- Non ci sono single point of failure
- Bilanciamento del carico: nessun server ha solo funzione di replica passiva
- Alta scalabilità

Operazioni

Operazioni User

- Operazioni di configurazione
- Operazioni in locale: ld, mount, readga, help
- Operazioni non transazionali: ls, cd, search, open, dwn
- Operazioni transazionali non ga: upload, rm, mkdir, rmdir
- Operazioni transazionali ga: upload, rm, newga, newcap, rmcap, mvcap, modcap

➡ le operazioni upload e rm hanno un comportamento polimorfo

Operazioni Admin

- Operazioni generiche: help, logout, exit
- Operazioni sul DB: chpsw, use, select, op

Protocollo di Comunicazione

Tipologie di messaggi:

- Boot Message: per la comunicazione client – boot server
- Boot Administrator Message: per la comunicazione admin – boot server
- UDP Message: per i messaggi di ping/pong
- Initialization Message: per operazioni non transazionali client – primary (utilizzati anche preparare l'ambiente per le transazioni)
- Transaction Message: per operazioni transazionali client – primary e primary-replica
- Primary – Replica Message: per la comunicazione di sistema (elezione, stati log, modifica liste) tra server PR

Formato dei messaggi

- Boot Message:

Pay Descriptor	Versione Dominio	Versione Server
1 byte	*8 byte*	*8 byte*

- Boot Admin Message:

Pay Descriptor	Nick	Dim Nick	Password	Dim Pswd	Query	Dim Query
1 byte	*30 byte*	*4 byte*	*30 byte*	*4 byte*	*1000 byte*	*4 byte*

- UDP Message:

Pay Descriptor	ID messaggio	IP sorgente	Porta
1 byte	*4 byte*	*16 byte*	*4 byte*

- Init Message:

Pay Descriptor	Risorsa	Dim Risorsa	Opzione 1	Opzione 2
1 byte	*1024 byte*	*4 byte*	*4 byte*	*4 byte*

- Tr Message:

Pay Descriptor	Id primary	Counter Tr	Counter Msg	Opzione 1	Opzione 2
1 byte	*4 byte*	*4 byte*	*30 byte*	*4 byte*	*4 byte*

- PR Message:

Pay Descriptor	Nome Server	Nome Dominio	Opzione 1	Opzione 2	Stringa Opzione
1 byte	*30 byte*	*30 byte*	*4 byte*	*4 byte*	*30 byte*

Lo schema leader-follower

- Realizzato con prethreding e mutex sull'accept()
- Il thread dispatcher si mette in ascolto di eventuali connessioni e genera un pool di dimensione statica di thread helper
- Ogni helper cerca di ottenere il lock sul mutex per chimare l'accept()
- Utilizzo del mutex per garantire il funzionamento corretto anche se la accept() è una funzione di libreria
- Scelta dei thread perchè molto più leggeri dei processi
- Possibilità di variare il numero degli helper di ogni poll attraverso variabili di configurazioni

Uno sguardo al codice I

Thread dispatcher

```
.....    // creazione del socket, bind() e listen()
pthread_t threads[N];
for(t=0 ; t < N ; t++)
 {
   rc = pthread_create (&threads[t] , NULL , func_helper , (void *) &listensd);
}
sem_wait(sem_exit_disp_s);
exit_helper = 1;

sleep(time_helper_s);
for(t=0 ; t < thread_helper_s ; t++)
{
   pthread_cancel(threads[t] );
}
.....
```

Uno sguardo al codice II

Thread helper

```
void *func_helper(void *l)
{
   int listensd = *(int *)l;
   int connsd;

   for ( ; ; )
   {
      if(exit_helper)  {  pthread_exit(NULL);  }

      sem_wait(sem_accept);
         connsd = accept(listensd,  ..............)
       sem_post(sem_accept_s);
      ......................
   }
}
```

Un DB embedded

- SQLITE è una libreria software scritta in linguaggio C che implementa un DBMS SQL di tipo ACID
- Utilizzato per la gestione delle informazioni di sistema e per i log

Nel dettaglio……

- Tab Parametri:

Id riga	Nome	Valore	Note

- Tab NameIdServer:

Id riga	Nome server	Id server

- Tab Dom:

Id riga	Nome dom	Nome server	Tipo server	Ip	Porta

- Tab ListServer:

Nome server	Ip server	Porta server

- Tab Log:

Id riga	Id server	Count TR	Ip	Porta	Id Op	Path risorsa
Path file temp	Path file salv	Opt 1	Opt2	Stato transazione		

Obiettivi

Obiettivi

- Consentire transazioni concorrenti su uno stesso file di testo
- Permettere scritture multiple su file durante una transazione

Problema

- Gestione degli offset difficile da realizzare

Patch testuali (Pseudo scritture multiple)

Formato Ad-hoc

Soluzione

##PACUFS__GA__FORMAT##21412341234213412 3##

§§§2

^^^1

Strange Deja Vu

Subconscious strange sensation
Unconscious relaxation
What a pleasant nightmare
And I can`t wait to get there again

^^^2

I get this feeling sometimes
Like theres nothing in the world that isn't mine
And I could have it all
Maybe I should have it all
I'll take you out of your life
Just the one sweet night
And you could take it all
Oh maybe I could rip it off

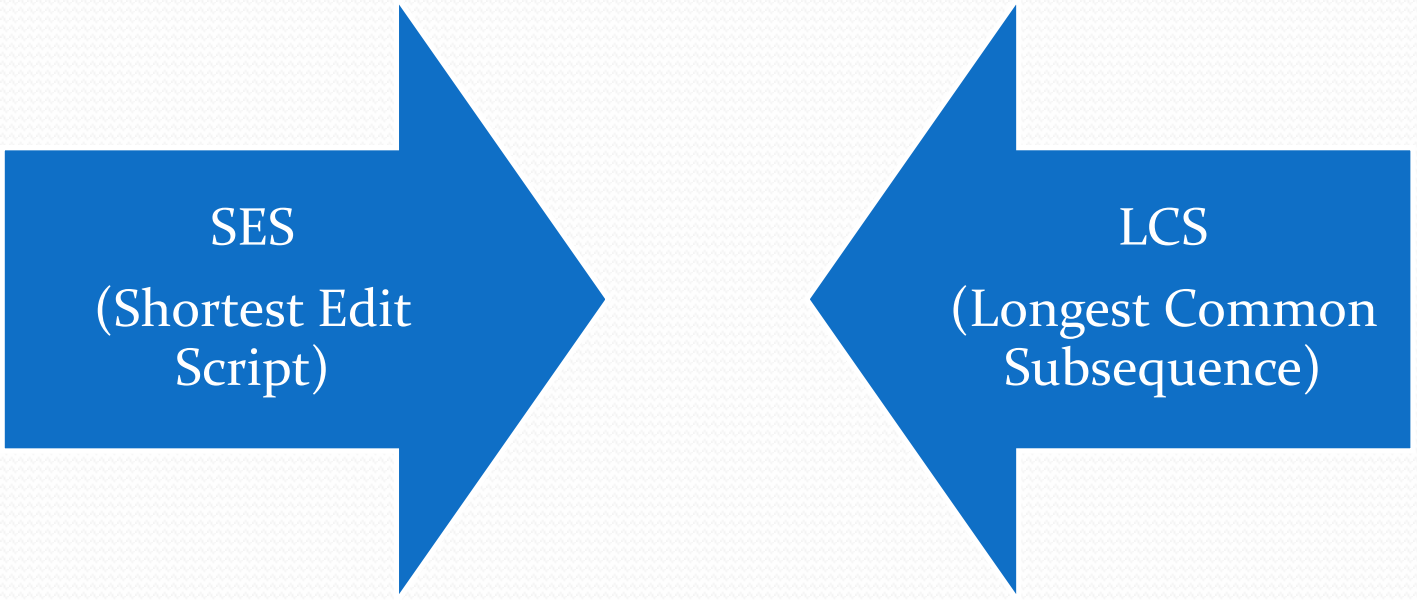
Stringa di formato che identifica il tipo GA

Contatore del numero di capitoli

ID capitolo

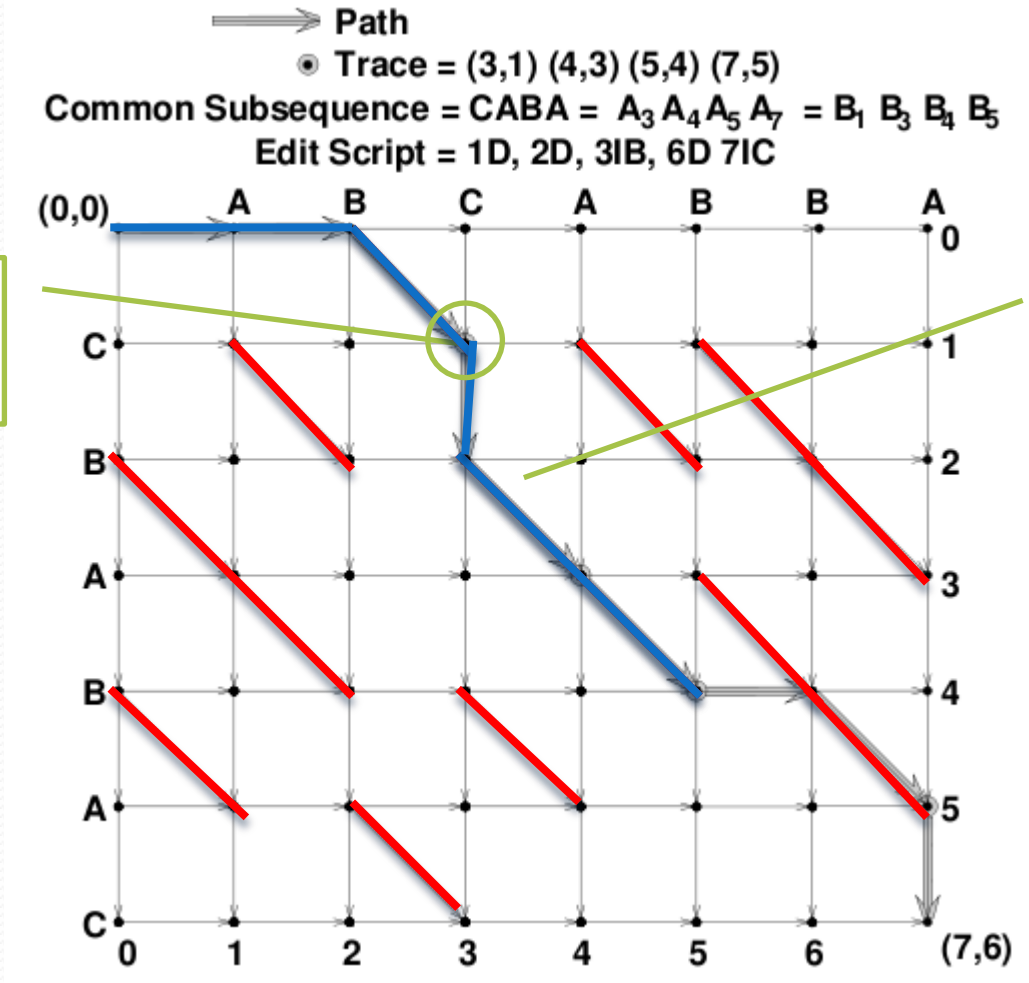
Patch testuali

ALGORITMO DI MYER



Patch testuali

Match Point



3-Path

Algoritmo Greedy

LEMMA 1

Un D-path deve finire su una diagonale k appartenente a

{-D, -D+2, -D+4,..., D-2, D}

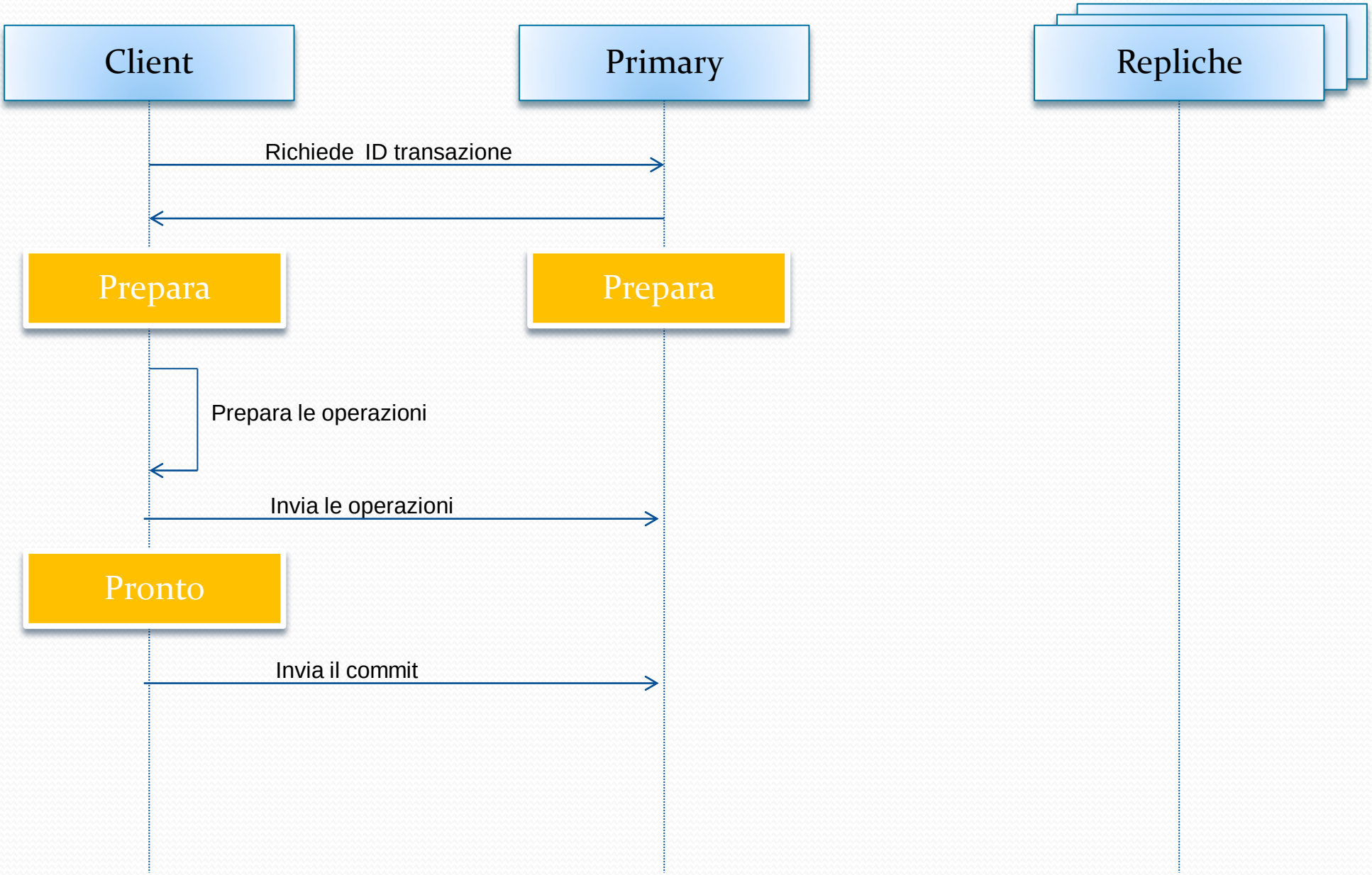
LEMMA 2

Uno 0-path più lontano da raggiungere terminerà alle cordinate (x,x) dove x è min(z-1 tale che a_z diverso da b_z oppure z>M o z>N). Un D-path FR (furthest reaching) sulla diagonale k può essere scomposto in un D-1path FR sulla diagonale k-1 seguito da un arco orizzontale e dallo snake più lungo possibile. Altrimenti può essere scomposto in FR sulla diagonale k+1 seguito da un arco verticale e dallo snake più lungo possibile

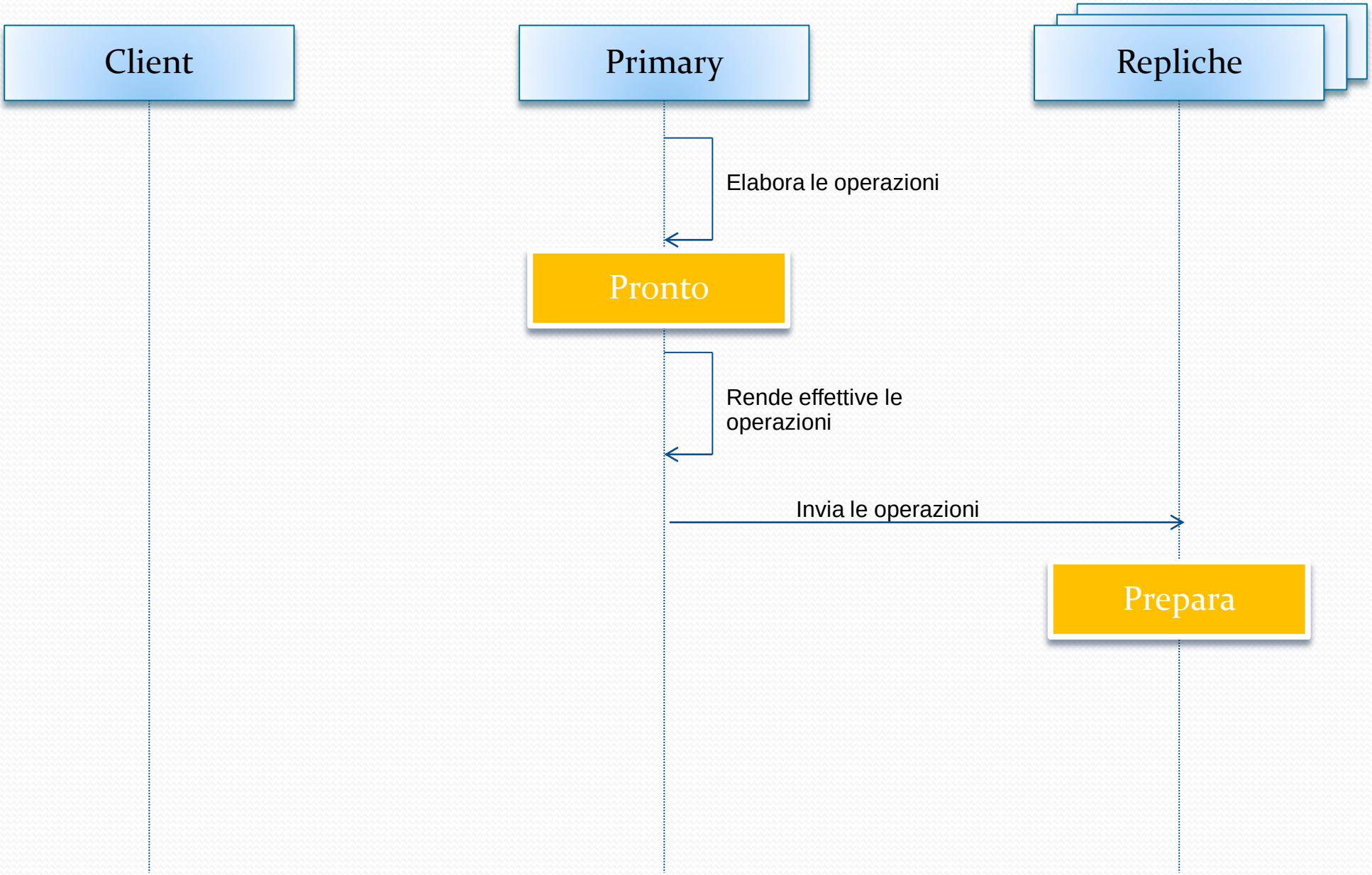
Algoritmo Greedy

```
V[ 1 ] = 0;
for ( int d = 0 ; d <= N + M ; d++ )
{
 for ( int k = -d ; k <= d ; k += 2 )
  {
        // down or right?
         bool down = ( k == -d || ( k != d && V[ k - 1 ] < V[ k + 1 ] ) );
        int kPrev = down ? k + 1 : k – 1;
         // start point
         int xStart = V[ kPrev ];
        int yStart = xStart – kPrev;
         // mid point
        int xMid = down ? xStart : xStart + 1;
        int yMid = xMid – k;
        // end point
        int xEnd = xMid;

        int yEnd = yMid;

         // follow diagonal

        int snake = 0;while ( xEnd < N && yEnd < M && A[ xEnd ] == B[ yEnd ] ) { xEnd++; yEnd++; snake++; }

                    // save end point
        V[ k ] = xEnd;
        // check for solution
        if ( xEnd >= N && yEnd >= M ) /* solution has been found */
         }
}
```

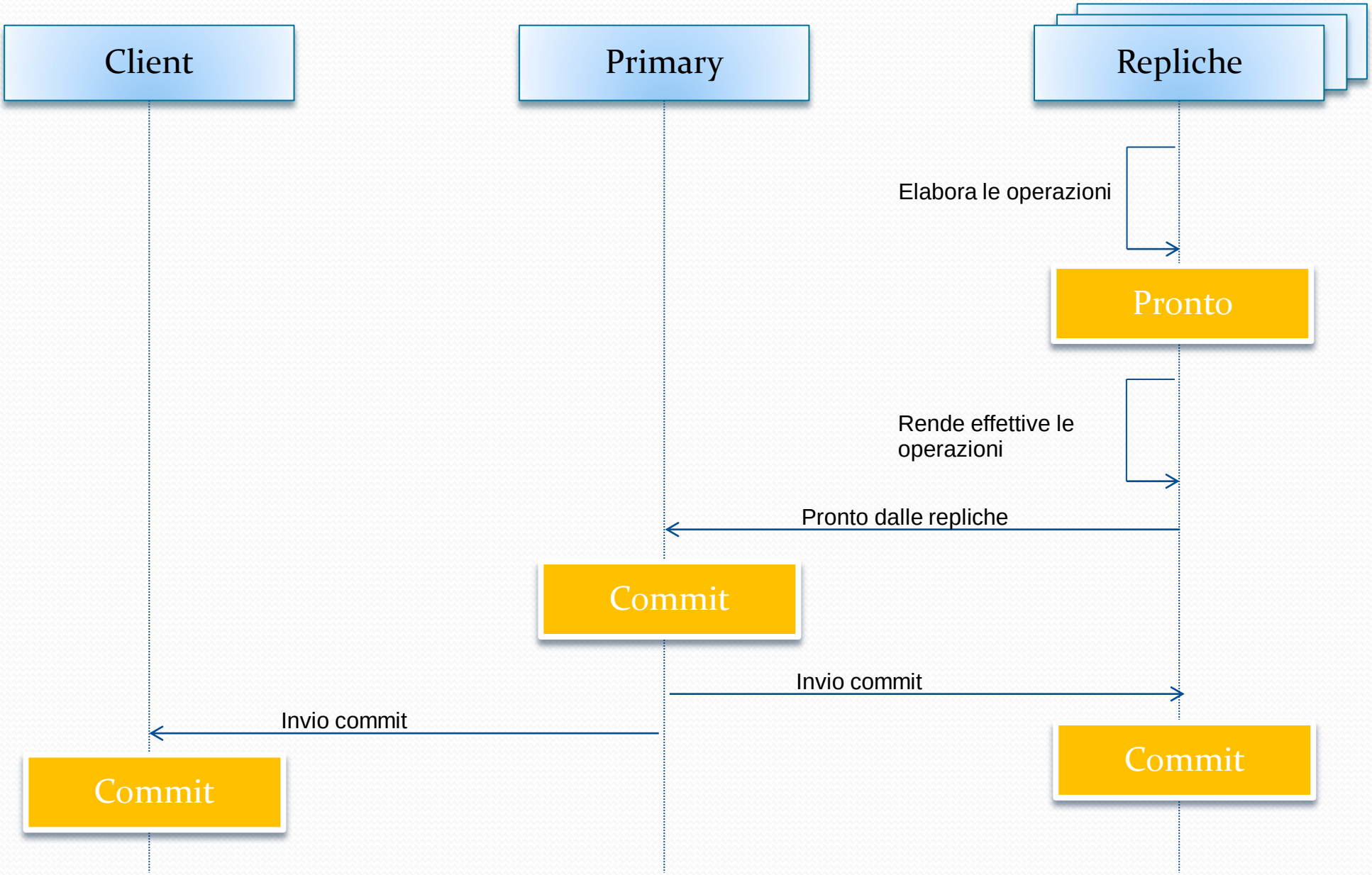
Two Commit


Client
Primary
Repliche
Richiede ID transazione
Prepara
Prepara
Prepara le operazioni
Invia le operazioni
Pronto
Invia il commit

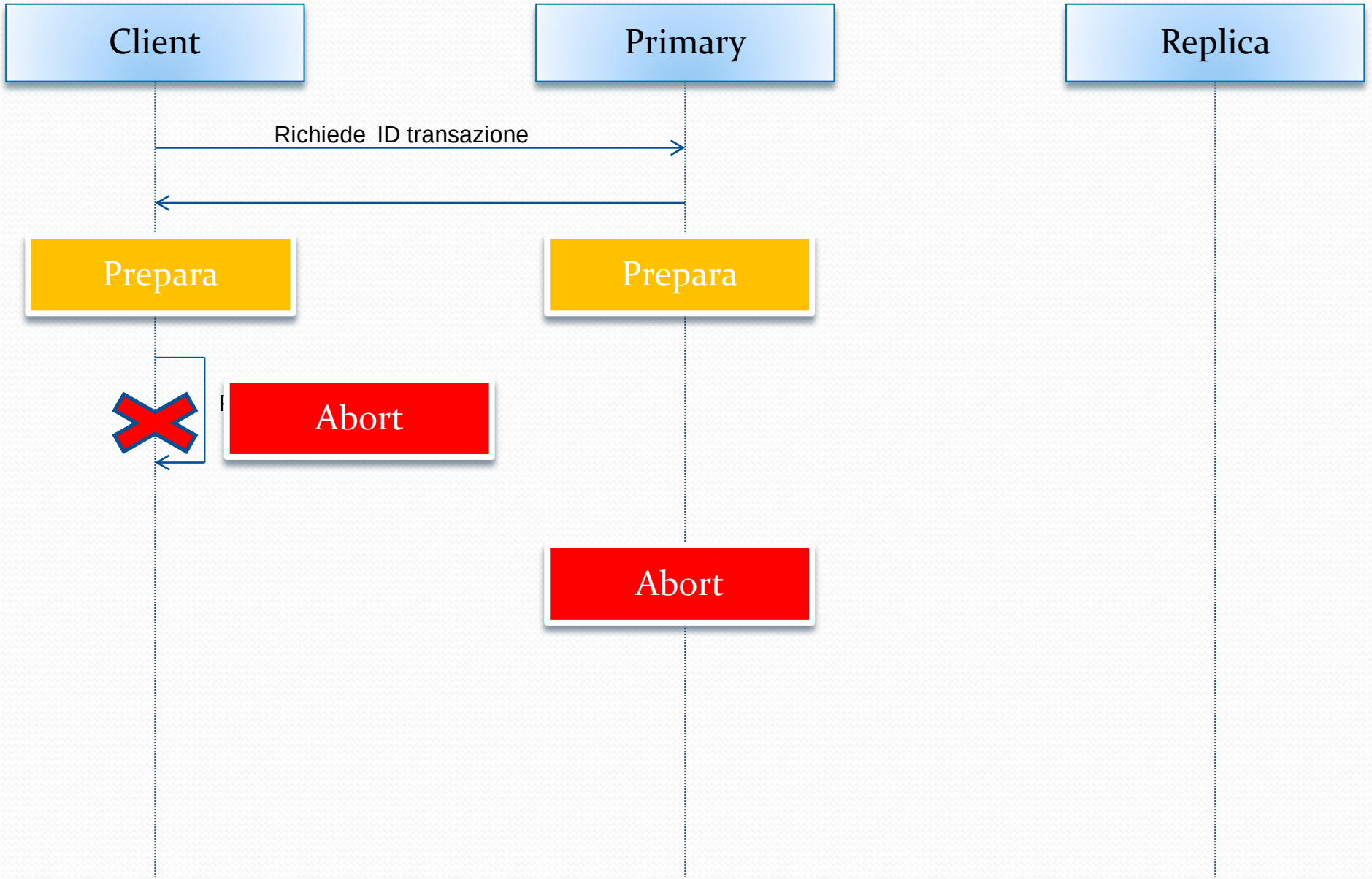
Two Commit


Client
Primary
Repliche
Elabora le operazioni
Pronto
Rende effettive le operazioni
Invia le operazioni
Prepara

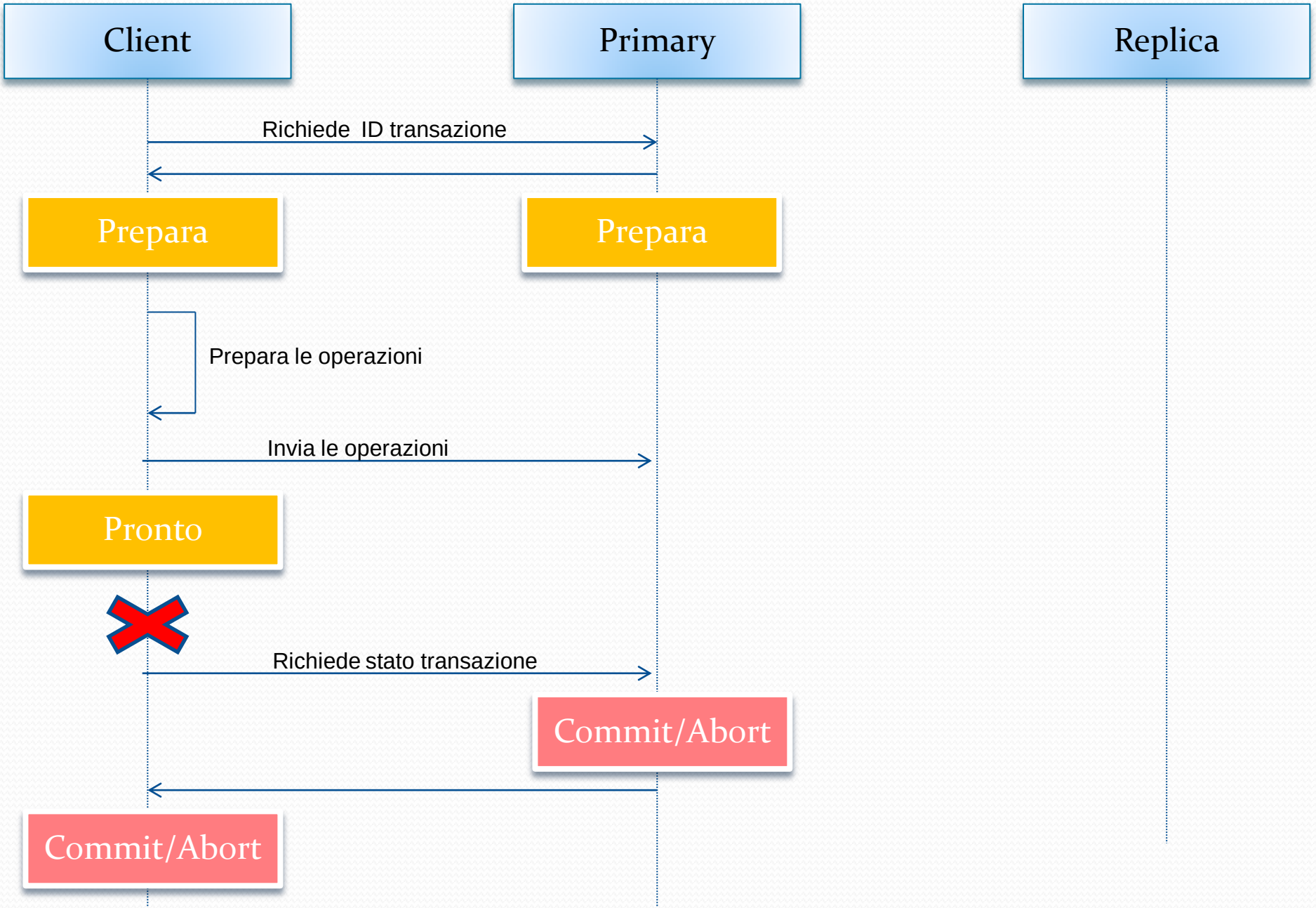
Two Commit


Client
Primary
Repliche
Elabora le operazioni
Pronto
Rende effettive le operazioni
Pronto dalle repliche
Commit
Invio commit
Invio commit
Commit
Commit

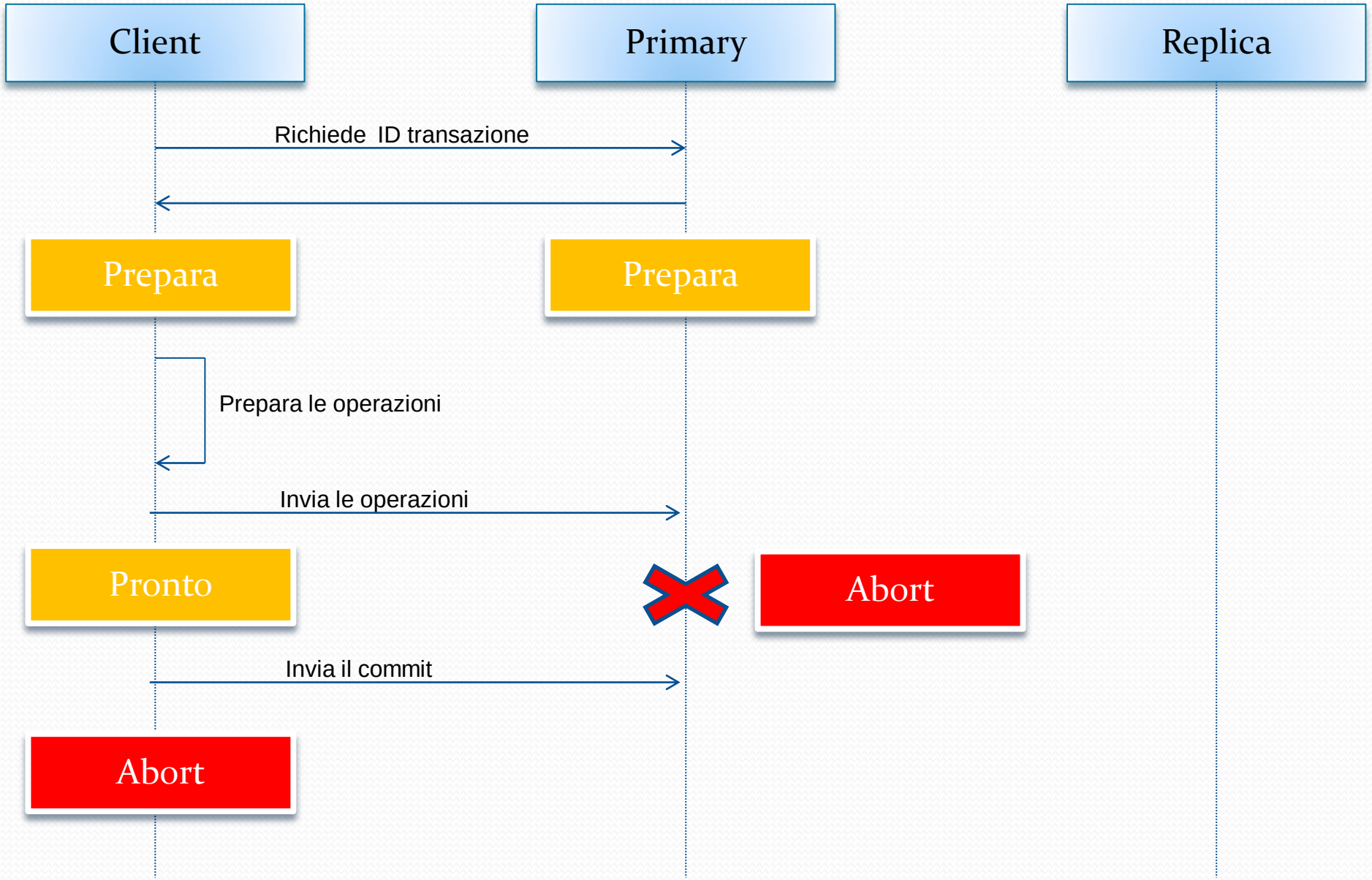
Two Commit


Client
Primary
Replica
Richiede ID transazione
Prepara
Prepara
Abort
Abort

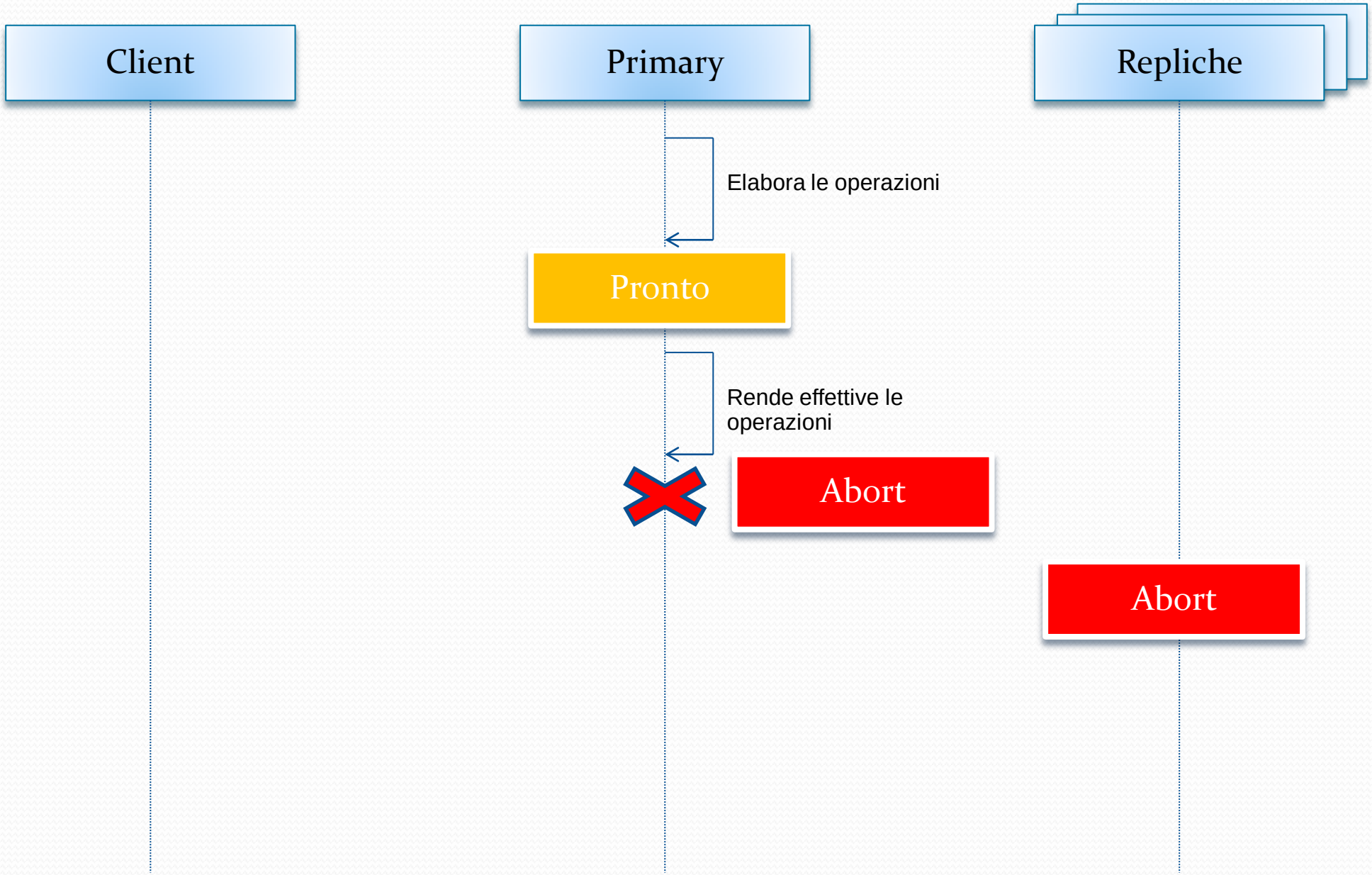
Two Commit


Client
Primary
Replica
Richiede ID transazione
Prepara
Prepara
Prepara le operazioni
Invia le operazioni
Pronto
Richiede stato transazione
Commit/Abort
Commit/Abort

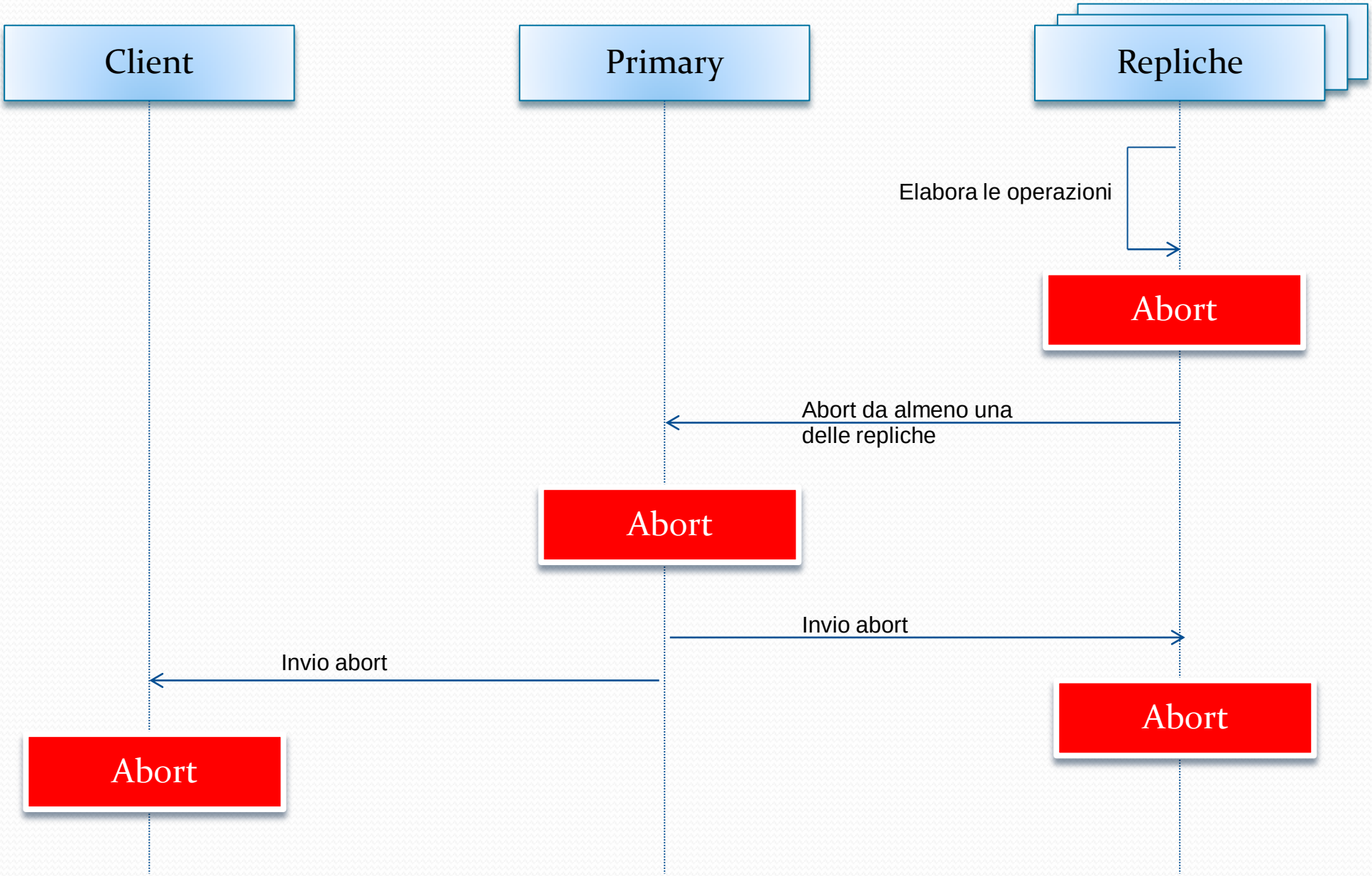
Two Commit


Client
Primary
Replica
Richiede ID transazione
Prepara
Prepara
Prepara le operazioni
Invia le operazioni
Pronto
Abort
Invia il commit
Abort

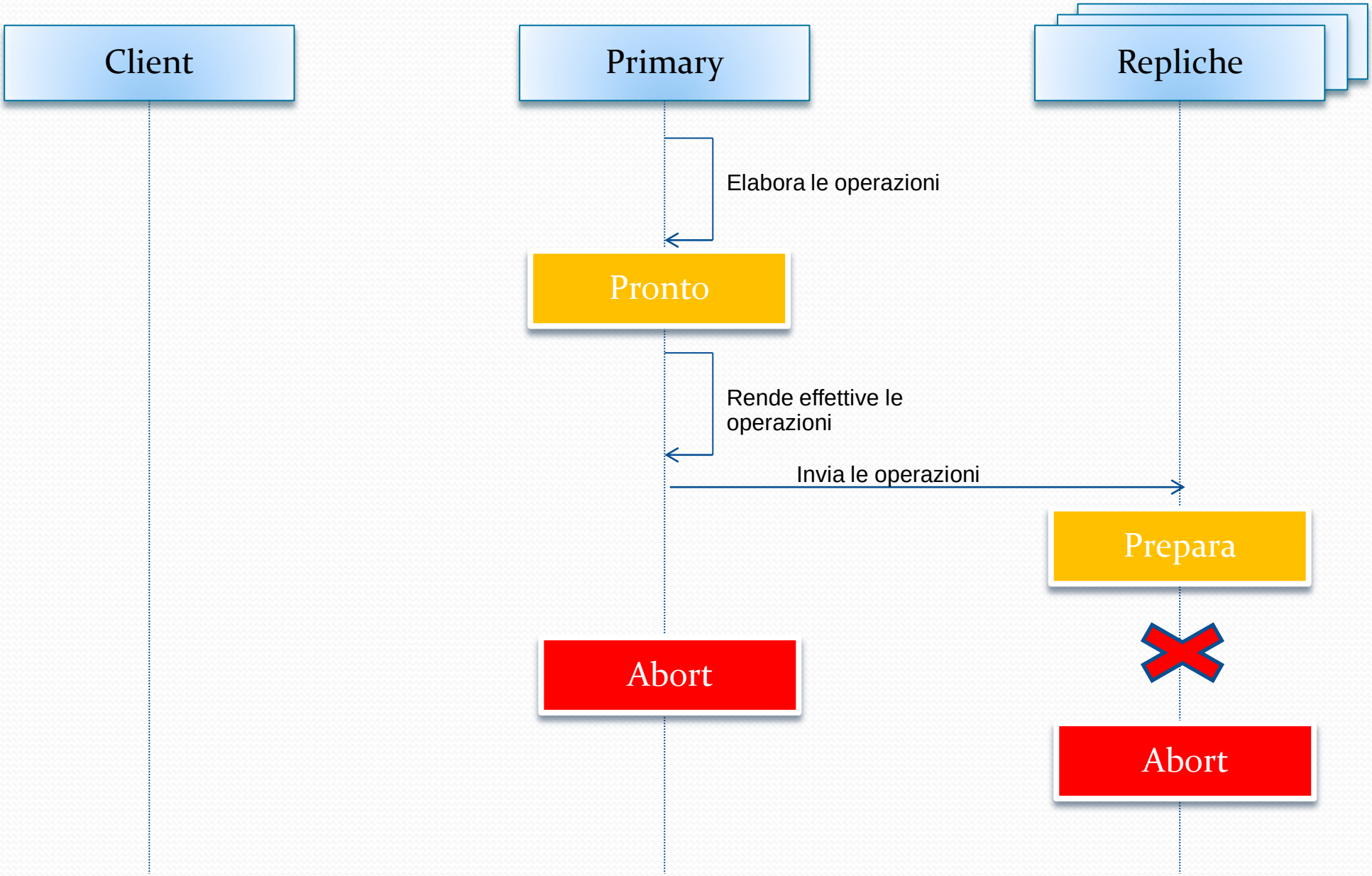
Two Commit


Client
Primary
Repliche
Elabora le operazioni
Pronto
Rende effettive le operazioni
Abort
Abort

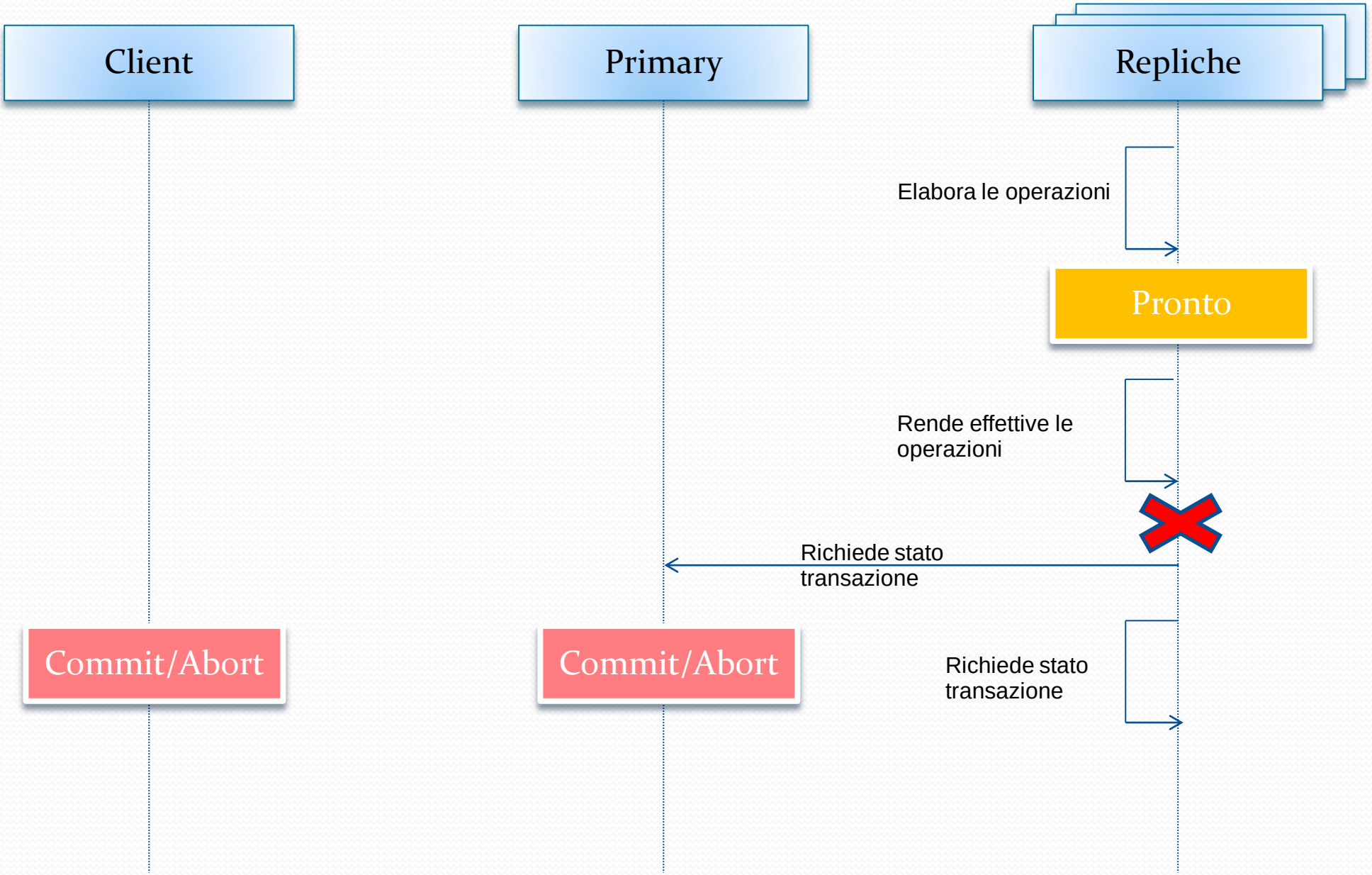
Two Commit


Client
Primary
Repliche
Elabora le operazioni
Abort
Abort da almeno una
delle repliche
Abort
Invio abort
Invio abort
Abort
Abort

Two Commit


Client
Primary
Repliche
Elabora le operazioni
Pronto
Rende effettive le operazioni
Invia le operazioni
Prepara
Abort
Abort

Two Commit


Client
Primary
Repliche
Elabora le operazioni
Pronto
Rende effettive le operazioni
Richiede stato transazione
Commit/Abort
Commit/Abort
Richiede stato transazione

Meccanismo di elezione

Lock logico dei capitoli

Ogni file GA è rappresentato dalla struttura Regions.

```
typedef struct
{
        struct list_el *lista;

        int inode;

        int access;

}regions;
```

Puntatore all'header della lista dei capitoli

Inode del file

Numero di thread che stanno accedendo al file

Lock logico dei capitoli

Ogni capitolo è rappresentato dalla seguente struttura:

```
struct list_el {
        int init;
        Two_rooms *cap_mutex;
        struct list_el *next;
};
```

Identificativo del capitolo

Struttura di sincronizzazione

Puntatore al capitolo successivo