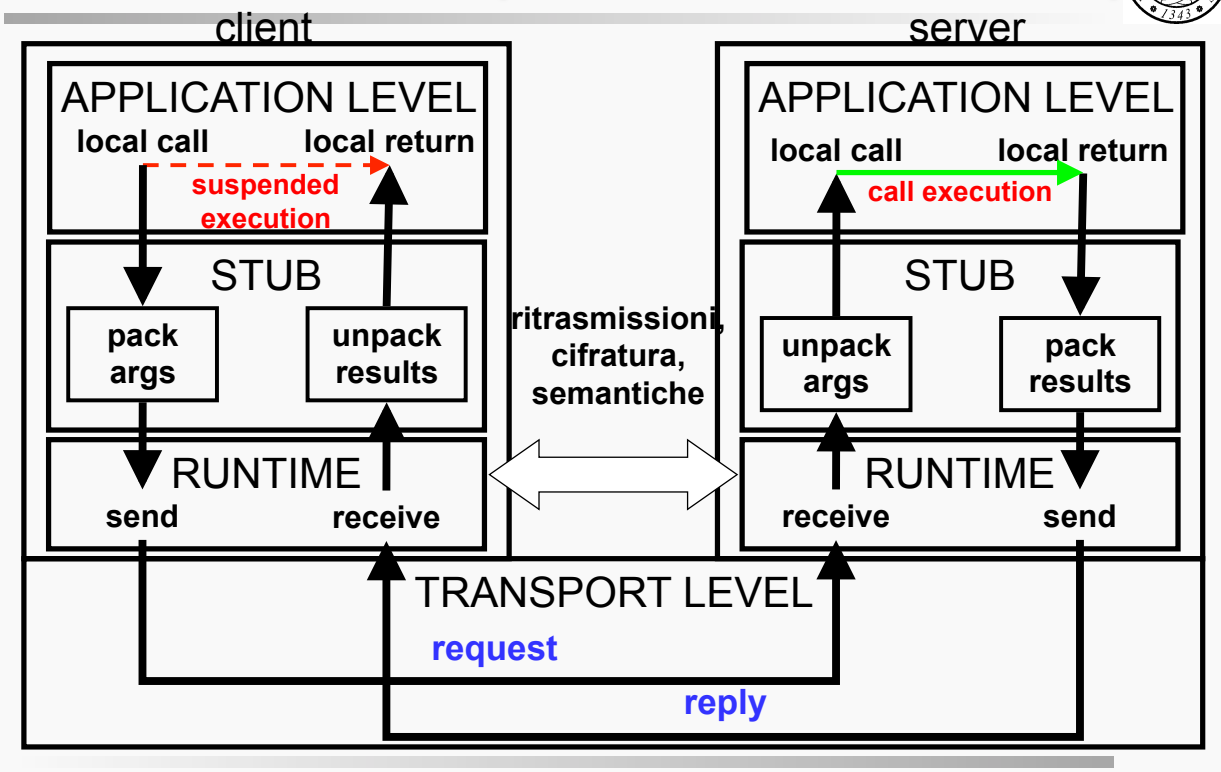
Remote Procedure Call

Concetti generali e principi di progettazione

Architettura (schema concettuale)



Implementazione

*Struttura del **request message***

- **Message Type**: specifica se il messaggio è di richiesta o di risposta (0 = request; 1 = reply)
- **Request Identifier**: identificatore unico del messaggio di richiesta; tipicamente ottenuto concatenando IP_{client}, $port_{client}$ e $counter_{client}$
- **Remote Object Reference**: riferimento all'oggetto remoto (*riferimento remoto*)
- **Method Identifier**: identificatore unico del metodo invocato
- **Arguments**: array di byte che contiene gli argomenti in formato di rete
- [**Client Identifier**: identificatore del cliente–per inviargli la risposta–è prelevato dal Request Identifier]

Implementazione

- *Struttura del **reply message***
  - **Message Type**: specifica se il messaggio è di richiesta o di risposta (0 = request; 1 = reply)
  - **Request Identifier**: identificatore unico del messaggio di richiesta (copiato dal messaggio di richiesta corrispondente)
  - **Reply status**: specifica se la richiesta ha avuto successo oppure è fallita
  - **Results**: in caso di successo specifica i risultati (array di byte in formato di rete); altrimenti specifica la ragione del fallimento

Riferimento remoto

Un riferimento remoto identifica univocamente un oggetto in rete

può essere trasmesso come argomento o valore di ritorno

Un riferimento remoto deve essere unico nel tempo e nello spazio

- ***unicità nello spazio***: ad ogni istante, oggetti diversi, possibilmente su nodi diversi, devono avere riferimenti remoti diversi
- ***unicità nel tempo***: uno stesso riferimento non può specificare oggetti diversi in tempi diversi
  un identificatore remoto non può essere riutilizzato

Tipi di riferimento remoto

Il riferimento remoto dipende dalla semantica degli oggetti

- **unicast**
  ad un certo istante esiste una sola copia dell'oggetto
- **volatile**
  il tempo di vita di un oggetto è al più pari a quello del processo che l'ha creato
- **non-rilocabile**
  l'oggetto non può essere rilocato in un altro processo

Riferimento remoto

Riferimento remoto per oggetti unicast, volatili e non-rilocabili

Internet Address	port number	time	object number	remote [interface|class]

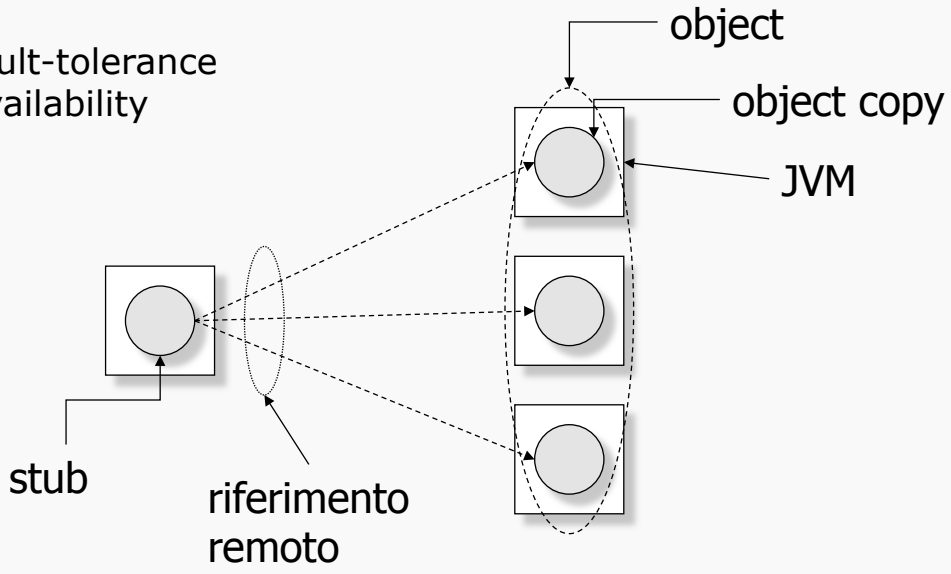
- **(Internet Address; port number)** identifica il processo (JVM) che implementa l'oggetto; il riferimento specifica quindi la locazione in rete dell'oggetto
- **Time** distingue due incarnazioni dello stesso processo (JVM)
- **Object number** identifica l'oggetto nell'ambito del processo
- **Remote interface** o **remote class** dá informazioni sull'interfaccia remota dell'oggetto e permette di determinare la classe dello stub quando è necessario creare uno stub
- Sotto le ipotesi unicast, volatile e non-rilocabile, un riferimento remoto specifica anche l'indirizzo dell'oggetto

Riferimento remoto e semantica dell'oggetto

oggetto multicast

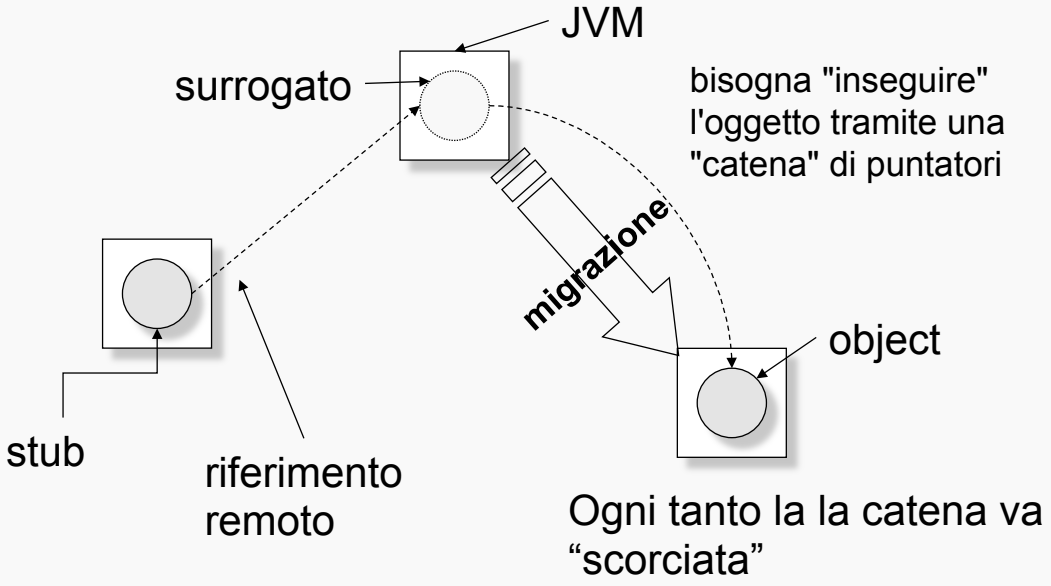
un oggetto è costituito da una o più copie

- fault-tolerance
- availability



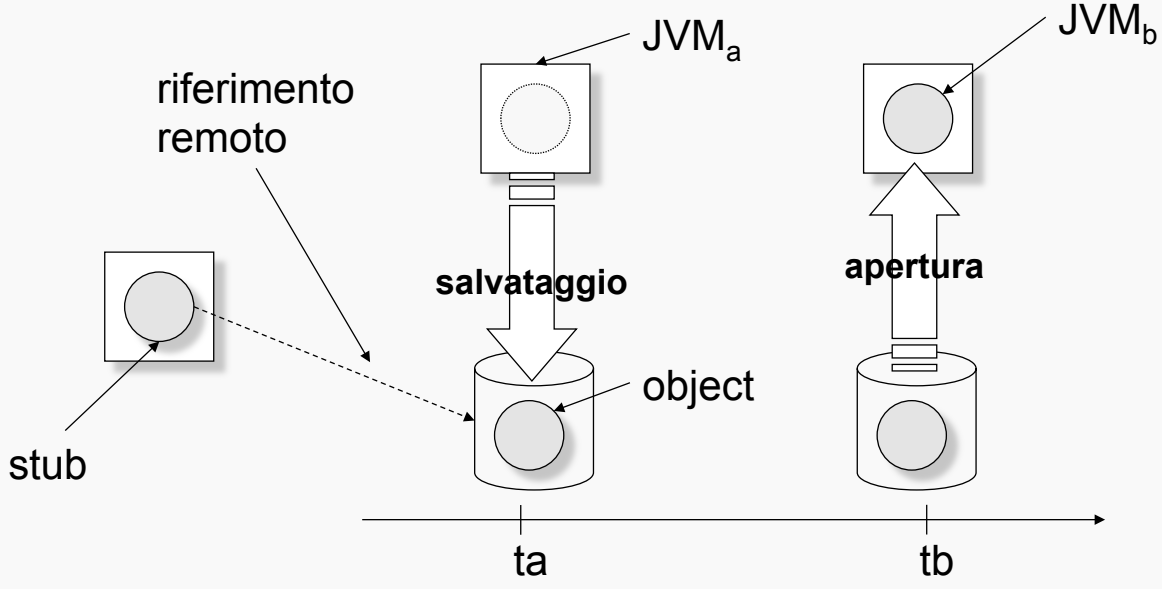
Riferimento remoto e semantica dell'oggetto

Un oggetto rilocabile può migrare in un'altra JVM



Riferimento remoto e semantica dell'oggetto

Un oggetto persistente può essere creato/salvato da una JVM e aperto da un’altra



Trasparenza

- L'obiettivo originario di RMI/RPC era la trasparenza
  - nessuna distinzione sintattica a livello di chiamata
  - la RMI/RPC machinery (marshalling, retransmission, location, …) è trasparente al chiamante
- In pratica la trasparenza non può essere ottenuta
  - guasti
  - latenza
- Si adotta una soluzione di compromesso
  - La trasparenza è a livello della sintassi di chiamata ma la differenza tra chiamata remota e locale è espressa nell'interfaccia (CORBA, RMI)

Semantiche di RMI

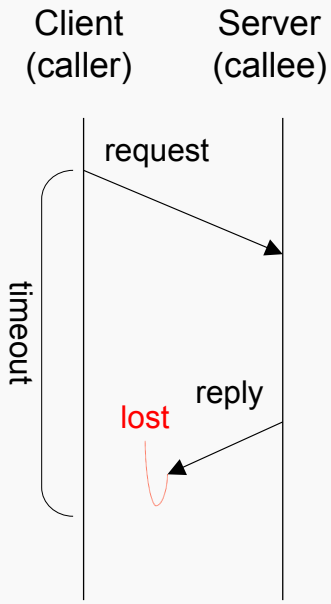
Il messaggio di richiesta e/o di risposta può andare perduto

- In caso di congestione o guasti ***temporanei*** della rete

RMI runtime può tentare di ripetere la chiamata remota

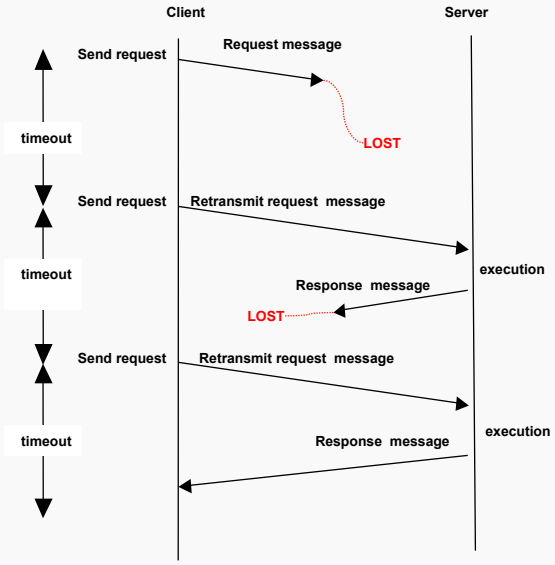
- may-be (forse)
- at least once (almeno una volta)
- at most once (al più una volta)

Semantica May-be



- Nessuna ritrasmissione
- Se il client riceve la reply allora può concludere che il server ha eseguito la request
- Quando scatta il timeout, il client (chiamante) non può dire niente
- Questo approccio non è usuale

Semantica At-least-once



- Se il chiamante riceve il reply allora conclude che l'operazione è stata eseguita ***almeno una volta***
- In effetti, l'operazione può essere eseguita più volte
- Una contromisura è progettare operazioni ***idempotenti***
- Una contromisura alternativa è quella di permettere al server di riconoscere richieste ripetute

Operazioni idempotenti

- Un'operazione è ***idempotente*** se la sua esecuzione ripetuta produce gli stessi effetti che se l'operazione fosse eseguita una sola volta
  - L'operazione "scrivi 1 nella variabile x" è idempotente
  - L'operazione "aggiungi 1 alla variabile x" non è idempotente
- La trasformazione di un'interfaccia che contiene operazioni non-idempotenti in un'interfaccia di operazioni idempotenti può non essere semplice

Operazioni idempotenti: esempio

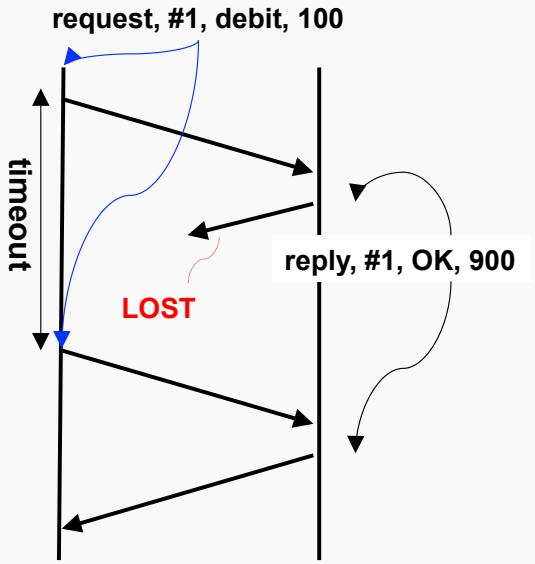
L'interfaccia *file ad accesso sequenziale*

- **file.read(rec)** legge da **file** il prossimo record e lo copia in **rec**
- **file.append(rec)** appende il record **rec** in coda al **file**

può essere trasformata in *file ad accesso casuale*

- **file.read(rec, pos)** legge da **file** il record in posizione **pos** e lo copia in **rec**
- **file.write(rec, pos)** scrive il record **rec** nel **file** nella posizione **pos**
- **file.getLastRecordPos()** ritorna la posizione dell'ultimo record di **file**

Semantica at-most-once



HISTORY

Request Identifier	Reply to be sent
…	…
#1	(OK, 900)
…	…

```
if ( history.search(requestId) )
{
   reply = history.get();
   return reply;
}
else
{
   reply = process request;
   history.add(requestId, reply);
}
```

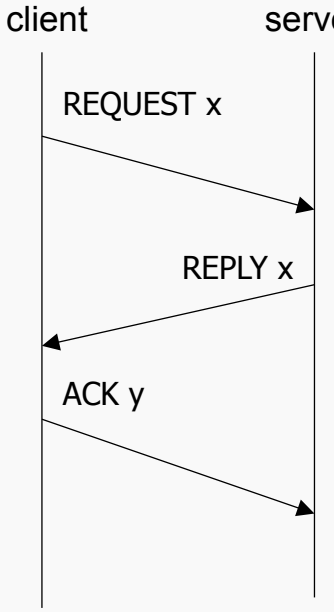
- Se il chiamante riceve il **reply** allora può concluedere che la procedura è stata eseguita al più una volta

Semantica at-most once

- ***Controllo della dimensione di history***
  - ***ovvero, quando il server può rimuovere un record dalla history?***
- ***Ipotesi: i client inviano le richieste in modo sequenziale***
  - Il server può rimuovere una entry relativa ad un client quando riceve la prossima richiesta da quel client.
  - Un server deve solo memorizzare l'ultimo reply per ciascun client
  - Quando un client termina non conferma la ricezione dell'ultimo reply, perció, dopo un periodo di tempo prestabilito, un record della history viene cancellato
  - ***Questa soluzione presenta dei problemi***
    - se un server ha molti client, la history può diventare comunque troppo grande
    - se un client è multi-threaded?

Semantica at-most once

Client multi-thread



- Il client mantiene il ***contatore delle richieste*** che viene incrementato per ogni richiesta inviata
- Il client tiene traccia delle ***richieste pendenti***
- Si estende il protocollo con un messaggio di ACK (***request-reply-ack***)
  - Con il messaggio **ack y** il cliente notifica che ha ricevuto **tutte** le reply ***fino*** alla reply **y**-esima compresa
  - Quando riceve il messaggio **ack y**, il server cancella dalla history tutti i record con ***request id*** minore o uguale a **y**

Semantica At-most-once

Gestione delle richieste sul lato-client

possono essere rimosse

Il client ha ricevuto risposta fino alla richiesta **Y**

La richiesta **X** è l'ultima inviata

Tecnologie e semantiche

Tecnologia	Semantica
Java RMI Corba	At most once
Corba(*)	Maybe
SUN RPC	At-least-once

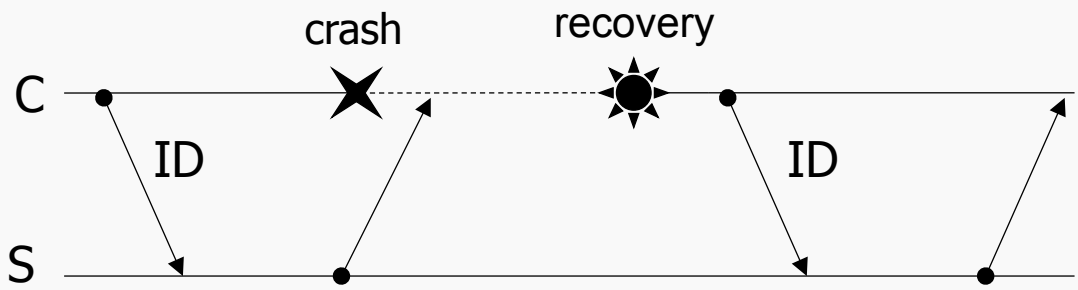
(*) Metodi che non ritornano un valore.

Livello di trasporto

Implementare RR su TCP o UDP?

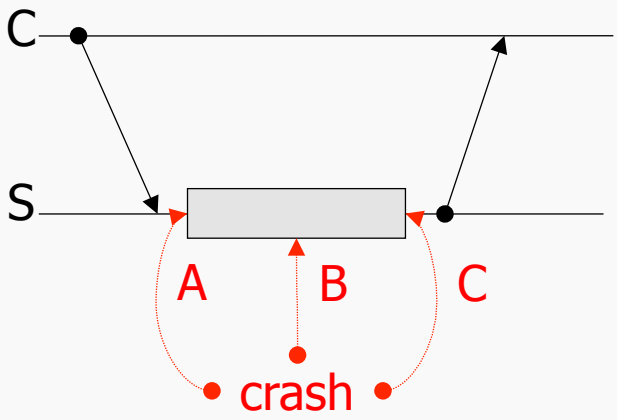
- *Dimensione dei messaggi*
  - La dimensione di un datagramma UDP è limitata, perciò è l'applicazione che deve gestire frammentazione e ri-assemblaggio
  - Con un flusso TCP si possono trasmettere argomenti e valori di ritorno di dimensione (teoricamente) illimitata
- *Affidabilità della comunicazione*
  - TCP gestisce la ritrasmissione ed il filtraggio dei duplicati
  - Su UDP, l'applicazione deve gestirli autonomamente
- *Costo*
  - Stabilire e mantenere una connessione TCP ha un costo, ma tale costo però può essere ripartito tra più RPC utilizzando per esse la stessa connessione

Guasto sul cliente



- Prima di ricevere la risposta il cliente C va in crash: si genera un chiamata orfana (*problema degli orfani*)
- Quando il cliente riparte può ripetere la chiamata: l'operazione può essere ripetuta
- La soluzione del problema degli orfani viene demandato al livello applicativo
- Semantica transazionale

Guasto sul server

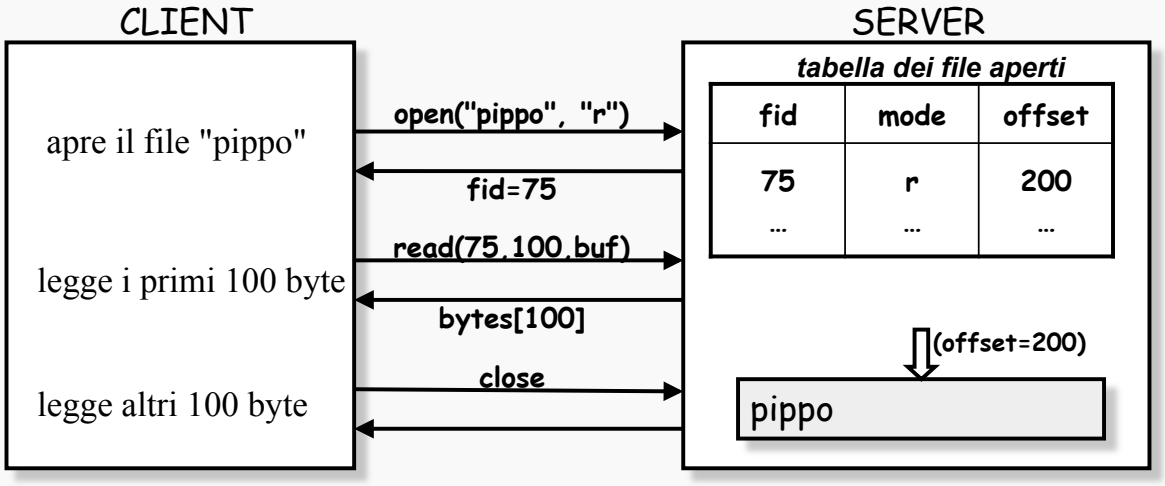


- Nel caso A il crash del server non dà problemi
- Nei casi B e C il crash del server può dare problemi se il server mantiene uno stato persistente
- Semantica transazionale

Progetto ed implementazione del server

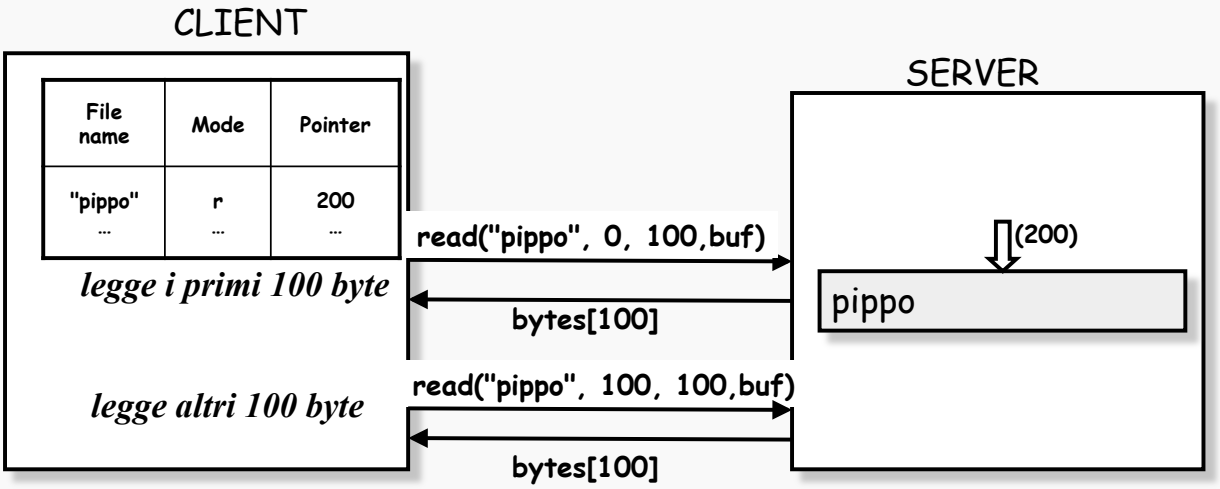
- **Server stateful**
  - Il server mantiene informazioni sullo stato del servizio
  - Tale stato può essere utilizzato per soddisfare una RMI in modo più efficiente
- **Server stateless**
  - Il server non mantiene alcuna informazione sullo stato del servizio
  - Ogni RMI deve specificare tutte le informazioni per servire correttamente la richiesta

Esempio di server stateful



- **FileIdentifier open(filename, mode)**
- **int read(fileidentifier, number of bytes, buffer pointer)**
- **int write(fileidentifier, number of bytes, buffer pointer)**
- **int close(fileidentifier)**

Esempio di server stateless



- int read(filename, offset, number of bytes, buffer pointer)
- int write(filename, offset, number of bytes, buffer pointer)

Server stateless vs stateful

- Il server stateful ha due vantaggi rispetto a quello stateless:
  - È più facile programmare il cliente poiché questo non deve mantenere stato
  - Lo stato mantenuto dal server può essere utilizzato per fare delle ottimizzazioni
- Il server stateless ha un vantaggio rispetto ad un server stateful
  - È più facile gestire le situazioni di guasto

Scelte di progetto

MISURE DI TOLLERANZA AI GUASTI			SEMANTICA	TECNOLOGIA
Ritrasmissione del messaggio di richiesta	*Filtraggio delle richieste duplicate*	*Riesecuzione della procedura o Ritrasmissione della risposta*		
NO	-	-	FORSE	CORBA*
SI	NO	RIESECUZIONE	ALMENO-UNA-VOLTA	SUN RPC
SI	SI	RITRASMISSIONE	AL-PIÙ-UNA-VOLTA	JAVA RMI, CORBA
*Quando il metodo non ritorna alcun valore				