Basi di dati distribuite

Architettura client-server

- è un modello generale di interazione tra processi software, in cui i processi che interagiscono sono suddivisi in
  - **client:** che richiedono servizi, e
  - **server:** che offrono servizi

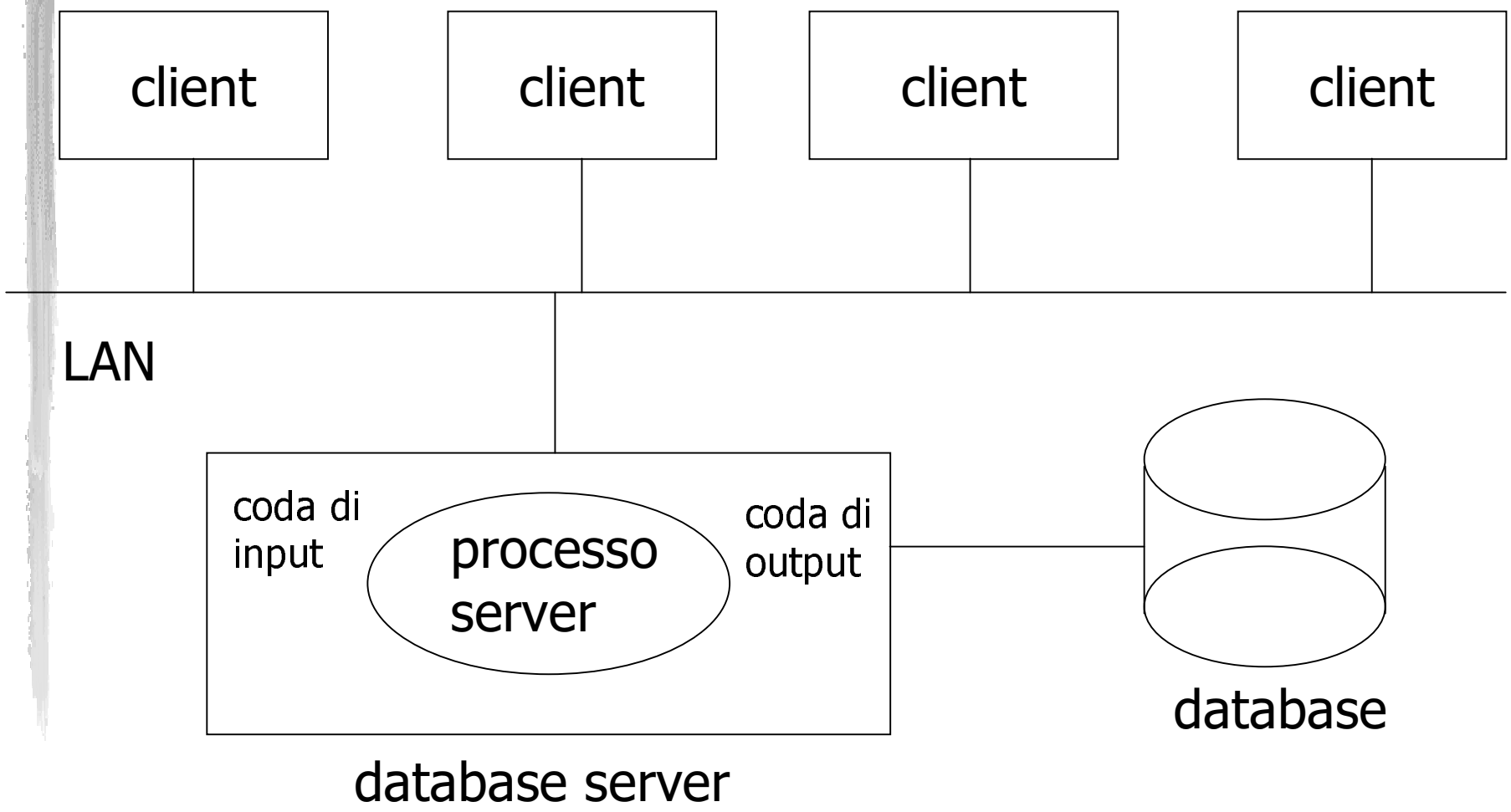
- richiede la definizione precisa di una *interfaccia di servizi*, che specifica i servizi offerti dal server

Architettura client-server

- il processo client ha un ruolo attivo
  (genera autonomamente richieste di servizi)
- il processo server è un processo reattivo (svolge una computazione solo a seguito di una richiesta da parte di un client)

- il processo client richiede un insieme limitato di servizi sequenzialmente ad uno o più processi server
- un processo server risponde a molte richieste provenienti da molti processi client

Architettura client-server



Architettura client-server

nella gestione dati, l'allocazione di processi client e server a computer differenti è ormai molto diffusa

- le funzionalità di client e server sono ben identificate
- portano a una separazione conveniente delle attività di progettazione e gestione

- le caratteristiche delle macchine destinate ai client e al server possono essere moltio diverse
  - i computer dedicati a client devono essere adatte all'interazione con l'utente
  - i computer dedicati al server devono avere una memoria di capacità elevata (per la gestione del buffer) e un disco di capacità elevata (per la memorizzazione della base di dati)

Architettura client-server

- SQL offre un paradigma di programmazione ideale per l'identificazione dell'interfaccia di servizi
  - le interrogazioni SQL sono formulate dal client e inviate al server
  - i risultati delle interrogazioni sono calcolati dal server e restituiti al client
  - grazie alla standardizzazione offerta da SQL è possibile sviluppare applicazioni client che coinvolgono diversi sistemi server

Architettura client-server

- spesso il server è *multi-threaded*
  - si comporta come un singolo processo che lavora dinamicamente per conto di diverse transazioni
  - ogni unità di esecuzione del processo server per una certa transazione è detta *thread*
- i server sono processi permanentemente attivi che controllano una *coda di input* per le richieste dei client e una *coda di outpput* per i risultati delle interrogazioni
- è spesso presente un processo *dispatcher* che distribuisce le richieste ai server e restituisce le risposte ai client

Architettura client-server

- **architettura two-tier:**
  - il client è sia l'interfaccia utente che il gestore dell'applicazione
  - si parla di *thick-client* in quanto il client contiene la logica dell'applicazione
- **architettura three-tier:**
  - è presente un secondo server, chiamato *application server*, responsabile della gestione della logica dell'applicazione che è comune a molti client
  - si parla di *thin-client* in quanto il client è responsabile unicamente dell'interfaccia con l'utente finale

Basi di dati distribuite

- una base di dati distribuita è un sistema in cui almeno un client interagisce con più server per l'esecuzione di un'applicazione

- le basi di dati distribuite richiedono di riesaminare
  - come l'utente può specificare le interrogazioni
  - come la tecnologia server può essere estesa ad un contesto distribuito

Basi di dati distribuite

- indipendenza dalla distribuzione dei dati: gli utenti non devono sapere dove i dati sono memorizzati (estende i principi di indipendenza logica e fisica dei dati)
- atomicità distribuita delle transazioni: gli utenti dovrebbero essere in grado di scrivere transazioni che accedono più siti esattamente come transazioni locali

- queste due proprietà non sono in genere supportate, per problemi di efficienza $\Rightarrow$gli utenti devono sapere dove i dati sono memorizzati

Basi di dati distribuite

- rispondono a esigenze applicative naturali

  le imprese sono strutturalmente distribuite, una gestione distribuita dei dati permette di distribuire il controllo e l'elaborazione dei dati dove questi sono generati e maggiormente utilizzati
- offrono maggiore flessibilità, modularità e resistenza ai guasti
  - possono essere configurate con aggiunta e modifica progressiva di componenti
  - anche se sono maggiormente soggette a guasti a causa della maggiore complessità strutturale, rispondono a guasti con una riduzione delle prestazioni piuttosto che con un failure totale

Classificazione delle applicazioni

- rispetto al tipo di DBMS coinvolti:
  - **basi di dati distribuite omogenee**: tutti i server hanno lo stesso DBMS
  - **basi di dati distribuite eterogenee**: i server hanno DBMS diversi
- rispetto alla rete:
  - possono utilizzare una rete locale LAN (local area network)
  - possono utilizzare una rete WAN (wide area network)

Classificazione delle applicazioni

	LAN	WAN
omogenee	applicazioni gestionali e finanziarie	applicazioni finanziarie e sistemi di prenotazione
eterogenee	sistemi informativi interdivisionali	sistemi di prenotazione integrati, sistemi interbancari

Basi di dati distribuite

- in una base di dati distribuita ogni server ha la capacità di gestire le applicazioni in modo indipendente
- una base di dati distribuita dovrebbe minimizzare le interazioni e la necessità di trasmettere dati attraverso la rete
- la distribuzione dei dati dovrebbe essere pianificata in modo tale che le applicazioni possano operare in modo indipendente su un singolo server

Frammentazione e allocazione dei dati

- data una relazione R, la sua frammentazione consiste nel determinare dei frammenti R_i di R applicando a R operazioni algebriche
  - **frammentazione orizzontale**: i frammenti R_i sono gruppi di tuple che hanno lo stesso schema di R

    (ottenibili come selezione su R)

    i frammenti orizzontali sono generalmente disgiunti
  - **frammentazione verticale**: ogni frammento R_i ha come schema un sottoschema dello schema di R

    (ottenibile come proiezione su R)

    i frammenti verticali devono includere la chiave primaria di R

Frammentazione e allocazione dei dati

- una frammentazione è corretta se è
  - *completa:* ogni dato in R deve essere presente in uno dei frammenti R_i
  - *ricostruibile:* il contenuto di R deve essere ricostruibile a partire da quello dei suoi frammenti
- la frammentazione è una tecnica di organizzazione dei dati che permette una distribuzione e un'elaborazione efficiente dei dati

Frammentazione dei dati

- consideriamo la relazione

IMPIEGATI(Imp#, Nome, Mansione, Data_A,Stipendio, Premio_P,Dip#)

- frammentazione orizzontale:
  - IMP1 = $\sigma_{\text{Dip\#=10}}$ IMPIEGATI
  - IMP2 = $\sigma_{\text{Dip\#=20 OR Dip\#=30}}$ IMPIEGATI

- la ricostruzione richiede un'unione:

$$\text{IMPIEGATI} = \text{IMP1} \cup \text{IMP2}$$

Frammentazione dei dati

IMPIEGATI(Imp#, Nome, Mansione, Data_A,Stipendio, Premio_P,Dip#)

- frammentazione verticale:
  - IMP1 = $\pi_{Imp\#,Nome,Mansione,Dip\#}$ IMPIEGATI
  - IMP2 = $\pi_{Imp\#,Data_A,Stipendio,Premio_P}$ IMPIEGATI
- la ricostruzione richiede un join naturale sul valore di chiave:

  IMPIEGATI = IMP1 |X| IMP2

Frammentazione dei dati

ESEMPIO - IMPIEGATI

Imp#	Nome	Mansione	Data_A	Stipendio	Premio_P	Dip#
7369	Rossi	ingegnere	17-Dic-80	1600,00	500,00	20
7499	Andrei	tecnico	20-Feb-81	800,00	?	30
7521	Bianchi	tecnico	20-Feb-81	800,00	100,00	30
7566	Rosi	dirigente	02-Apr-81	2975,00	?	20
7654	Martini	segretaria	28-Set-81	800,00	?	30
7698	Blacchi	dirigente	01-Mag-81	2850,00	?	30
7782	Neri	ingegnere	01-Giu-81	2450,00	200,00	10
7788	Scotti	segretaria	09-Nov-81	800,00	?	20
7839	Dare	ingegnere	17-Nov-81	2600,00	300,00	10
7844	Turni	tecnico	08-Set-81	1500,00	?	30
7876	Adami	ingegnere	28-Set-81	1100,00	500,00	20
7900	Gianni	ingegnere	03-Dic-81	1950,00	?	30
7902	Fordi	segretaria	03-Dic-81	1000,00	?	20
7934	Milli	ingegnere	23-Gen-82	1300,00	150,00	10
7977	Verdi	dirigente	10-Dic-80	3000,00	?	10

Frammentazione dei dati

IMP1

Imp#	Nome	Mansione	Data_A	Stipendio	Premio_P	Dip#
7782	Neri	ingegnere	01-Giu-81	2450,00	200,00	10
7839	Dare	ingegnere	17-Nov-81	2600,00	300,00	10
7934	Milli	ingegnere	23-Gen-82	1300,00	150,00	10
7977	Verdi	dirigente	10-Dic-80	3000,00	?	10

IMP2

Imp#	Nome	Mansione	Data_A	Stipendio	Premio_P	Dip#
7369	Rossi	ingegnere	17-Dic-80	1600,00	500,00	20
7499	Andrei	tecnico	20-Feb-81	800,00	?	30
7521	Bianchi	tecnico	20-Feb-81	800,00	100,00	30
7566	Rosi	dirigente	02-Apr-81	2975,00	?	20
7654	Martini	segretaria	28-Set-81	800,00	?	30
7698	Blacchi	dirigente	01-Mag-81	2850,00	?	30
7788	Scotti	segretaria	09-Nov-81	800,00	?	20
7844	Turni	tecnico	08-Set-81	1500,00	?	30
7876	Adami	ingegnere	28-Set-81	1100,00	500,00	20
7900	Gianni	ingegnere	03-Dic-81	1950,00	?	30
7902	Fordi	segretaria	03-Dic-81	1000,00	?	20

Livelli di trasparenza

- ci sono tre livelli di trasparenza significativi: trasparenza di frammentazione, di allocazione e di linguaggio

- si consideri la relazione IMPIEGATI, con frammenti orizzontale IMP1 e IMP2 e schema di allocazione
  - IMP1@company.london.uk
  - IMP2@company.manchester1.uk
  - IMP2@company.manchester2.uk

Trasparenza di frammentazione

a questo livello, il programmatore non si deve preoccupare del fatto che la base di dati sia distribuita o frammentata

- esempio di interrogazione:

```
procedure Query1(:num,:nome);
    SELECT Nome into :nome
    FROM IMPIEGATI
    WHERE Imp#=:num;
end procedure;
```

Trasparenza di allocazione

- a questo livello, il programmatore deve conoscere la struttura dei frammenti, ma non deve indicare la loro collocazione
- in caso di replicazione, il programmatore non deve indicare quale copia viene acceduta (trasparenza di replicazione)
- esempio di interrogazione:

```
procedure Query2(:num,:nome);
        SELECT Nome into :nome FROM IMP1 WHERE Imp#=:num;
if :empty then
        SELECT Nome into :nome FROM IMP2 WHERE Imp#=:num;
end procedure;
```

Trasparenza di linguaggio

- a questo livello, il programmatore deve indicare nell'interrogazione sia la struttura dei frammenti che la loro collocazione
- le interrogazioni espresse a livelli di trasparenza più alti sono trasformate a questo livello dall'ottimizzatore di interrogazioni, che conosce la frammentazione e l'allocazione dei dati
- esempio di interrogazione:

```
procedure Query3(:num,:nome);
SELECT Nome into :nome FROM IMP1@company.london.uk WHERE Imp#=:num;
if :empty then
SELECT Nome into :nome FROM IMP2@company.manchester1.uk WHERE
Imp#=:num;
end procedure;
```

Ottimizzazioni

- questa applicazione può essere resa più efficiente in due modi:
  - usando il parallelismo: invece di sottomettere le due richieste una dopo l'altra, queste possono essere processate in parallelo, risparmiando così sul tempo di risposta globale
  - usando informazioni sulle proprietà logiche dei frammenti (ma in questo modo si sviluppano programmi non flessibili)

```
procedure Query4(:num,:nome,:citta);
case :citta of
"London":       SELECT Nome into :nome
                FROM IMP1 WHERE Imp#=:num;
"Manchester":   SELECT Nome into :nome
                FROM IMP2 WHERE Imp#=:num;
end procedure;
```

Classificazione delle transazioni

- **richieste remote**: transazioni read-only costituite da un numero arbitrario di interrogazioni SQL, rivolte ad un singolo DBMS remoto (che può solo essere interrogato)
- **transazioni remote**: transazioni costituite da un numero arbitrario di comandi SQL (select, insert, delete, update), rivolti ad un singolo DBMS remoto (ogni transazione scrive su un DBMS)
- **transazioni dstribuite**: transazioni costituite da un numero arbitrario di comandi SQL (select, insert,delete, update), rivolti ad un numero arbitrario di DBMS remoti (ogni transazione può scrivere su più DBMS, ma ogni comando si riferisce ad un DBMS)
- **richieste distribuite**: transazioni arbitrarie, in cui ogni comando SQL può essere rivolto ad ogni DBMS remoto (richiedono un ottimizzatore distribuito)

Classificazione delle transazioni

- **esempio di transazione distribuita** (trasferimento tra due conti correnti) al livello di trasparenza di allocazione
- relazione CONTOCORRENTE(Numero,Nome,Saldo)

  frammentata in modo tale che tutti conti inferiore al 10.000 siano sul primo frammento CC1 e tutti quelli superiori siano sul secondo frammento CC2

```
begin transaction
UPDATE CC1      SET Saldo = Saldo - 100.000
                WHERE Numero = 3154;
UPDATE CC2      SET Saldo = Saldo + 100.000
                WHERE Numero = 14878;
commit work;
end transaction;
```

Tecnologia delle basi di dati distribuite

- la distribuzione dei dati non influenza la proprietà di consistenza e persistenza delle transazioni

- gli altri sottosistemi devono essere rivisti in ambito distribuito:
  - ottimizzazione delle interrogazioni
  - controllo della concorrenza
  - gestione del ripristino

Ottimizzazione di interrogazioni distribuite

- è necessaria quando un DBMS riceve una richiesta distribuita
- il DBMS interrogato è responsabile dell'ottimizzazione globale
  - decide come suddividere l'interrogazione in sottointerrogazioni, ciascuna rivolta ad uno specifico DBMS
  - costruisce un piano di esecuzione distribuito, che consiste nell'esecuzione coordinata di vari programmi in vari DBMS e nello scambio di dati tra di essi
- il costo di esecuzione di un'interrogazione distribuita include il costo di trasmissione dei dati sulla rete

Ottimizzazione di interrogazioni distribuite

- approccio basato sui costi: si considerano tutti i piani e si sceglie il più economico (come nel caso centralizzato)
- differenze:
  - bisogna considerare i costi di comunicazione
  - bisogna rispettare l'autonomia locale dei siti
  - nuovi metodi di join distribuiti (semijoin, bloomjoin)
- il sito a cui è rivolta l'interrogazione costruisce un piano globale, che comprende dei piani locali suggeriti ad ogni sito

  se un sito può migliorare il piano locale che gli viene suggerito è libero di farlo

Controllo della concorrenza

- la serializzabilità locale non garantisce la serializzabilità
- esempio (il primo pedice denota la transazione, il secondo pedice denota il sito)

  S1: $r_{11}(x)$ $w_{11}(x)$ $r_{21}(x)$ $w_{21}(x)$
  S2: $r_{22}(y)$ $w_{22}(y)$ $r_{12}(y)$ $w_{12}(y)$

- la serializzabilità globale di un insieme di transazioni distribuite sui nodi di una base di dati distribuita richiede l'esistenza di un unico schedule seriale S equivalente a tutti gli schedule locali S_i prodotti ad ogni nodo

Controllo della concorrenza

- se ogni scheduler di una base di dati distribuita usa il metodo di locking a due fasi in ogni nodo e completa l'azione di commit quando tutte le sotto-transazioni hanno acquisito tutto le risorse, gli schedule risultanti sono globalmente serializzabili
- se ogni transazione distribuita acquisisce un singolo timestamp e usa tale timestamp in tutte le richieste a tutti gli scheduler che usano un protocollo di controllo della concorrenza basato su timestamp, gli schedule risultanti sono globalmente serializzabili, basati sull'ordine imposto dai timestamp
- metodo di Lamport per assegnare timestamp in ambito distribuito
- metodi di risoluzione di deadlock in ambito distribuito

Guasti nei sistemi distribuiti

- **guasti dei nodi** possono verificarsi in ogni nodo del sistema
- **perdite di messaggi** lasciano l'esecuzione di un protocollo in uno stato di incertezza
  - ogni messaggio del protocollo (msg) è seguito da un messaggio di conferma (ack)
  - la perdita di uno dei due messaggi lascia il ricevente incerto circa il ricevimento del messaggio
- **partizionamento della rete** è un guasto del collegamento di comunicazione della rete che la suddivide in due sottoreti senza possibilità di comunicazione
  - una transazione può essere simultaneamente attiva in più di una sotto-rete

Protocollo di commit a due fasi

- i protocolli di commit permettono ad una transazione di raggiungere la decisione corretta relativamente al commit o all'abort in tutti i nodi che partecipano ad una transazione
- il protocollo di commit a due fasipuò essere pensato come un matrimonio, in quanto la decisione delle due parti è ricevuta e registrata da una terza parte, che la ratifica
  - i server (che rappresentano i partecipanti al matrimonio) sono detti gestori delle risorse (RM)
  - il coordinatore (celebrante) è un processo, detto transaction manager (TM)

Commit a due fasi

- il protocollo consiste in un rapido scambio di messaggi tra TM e RM e in scritture di recortd nei loro log
- il TM può utilizzare
  - meccanismi di broadcast, per trasmettere lo stesso messaggio a molti nodi, raccogliendo poi le risposte che arrivano dai vari nodi
  - comunicazione seriale con uno dei RM per volta
- record del TM
  - il record **prepare** contiene l'identità di tutti i processi RM (id di nodo e processo)
  - il record **global commit** o **global abort** descrive la decisione globale (quando il TM scrive tale record nel suo log raggiunge la decisione finale)
  - il record **complete** viene scritto alla fine del protocollo di commit a due fasi

Commit a due fasi

- record del RM
  - il record **ready** indica la disponibilità irrevocabile a partecipare al protocollo
  - può essere scritto solo quando il RM è in uno stato affidabile, cioè possiede tutti i lock sulle risorse che devono essere scritte
  - il record deve contenere anche l'id del TM
  - si hanno inoltre i record classici del caso centralizzato
- in ogni momento un RM può autonomamente abortire una sotto-transazione, disfandone gli effetti, senza partecipare al protocollo di commit a due fasi

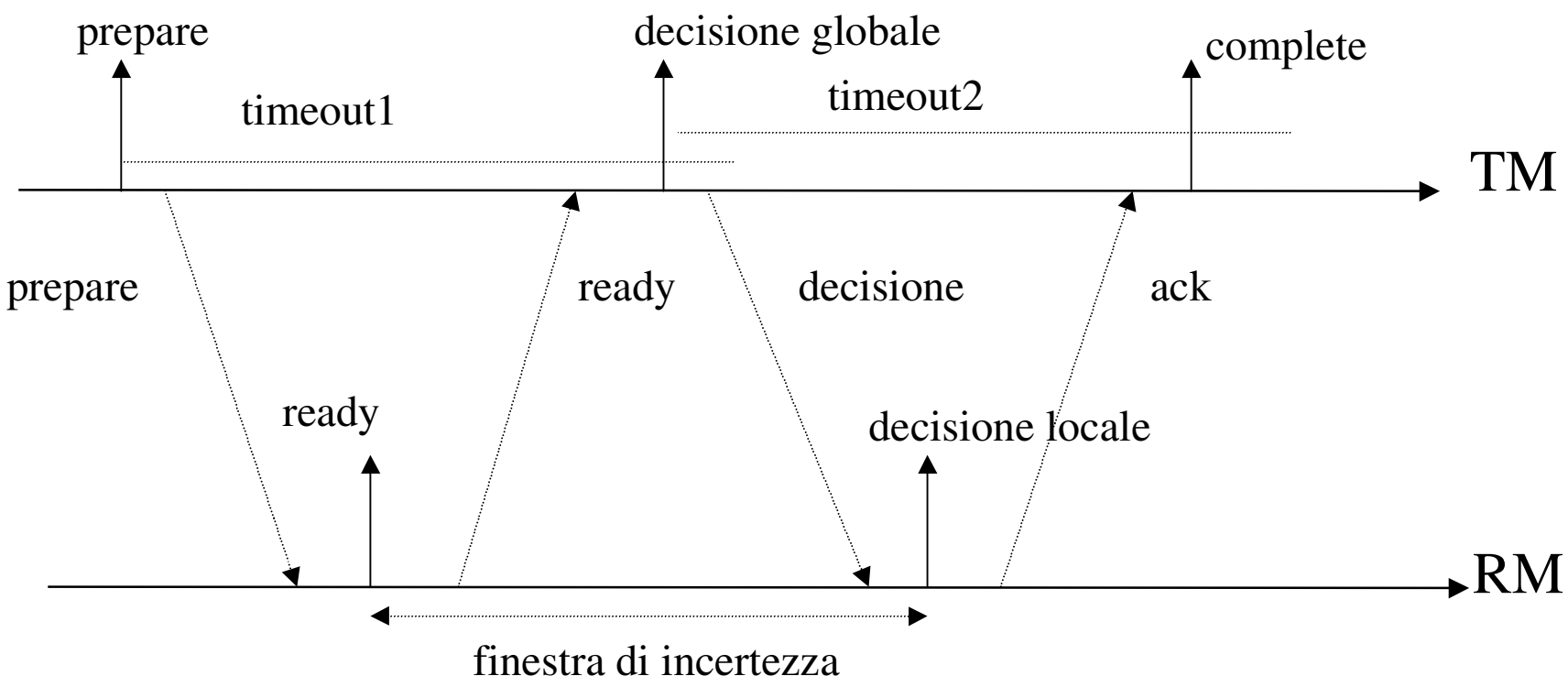
Commit a due fasi - fase 1

- il TM scrive il record **prepare** nel suo log e manda un messaggio **prepare** a tutti i RM e setta un timeout che indica il massimo tempo allocato per il completamento della prima fase
- i RM ripristinabili scrivono sul loro log un record **ready** e mandano al TM un messaggio **ready**, che indica la volontà di partecipare al commit
- i RM non ripristinabili mandano un messaggio **not-ready** e terminano il protocollo
- il TM raccoglie i messaggi di risposta da parte dei RM:
  - se riceve risposte positive da tutti i RM, scrive un record **global commit** sul suo log
  - se riceve uno o più risposte negative, o se il time-out scade senza che il TM abbia ricevuto tutte le risposte, scrive un record **global abort** sul suo log

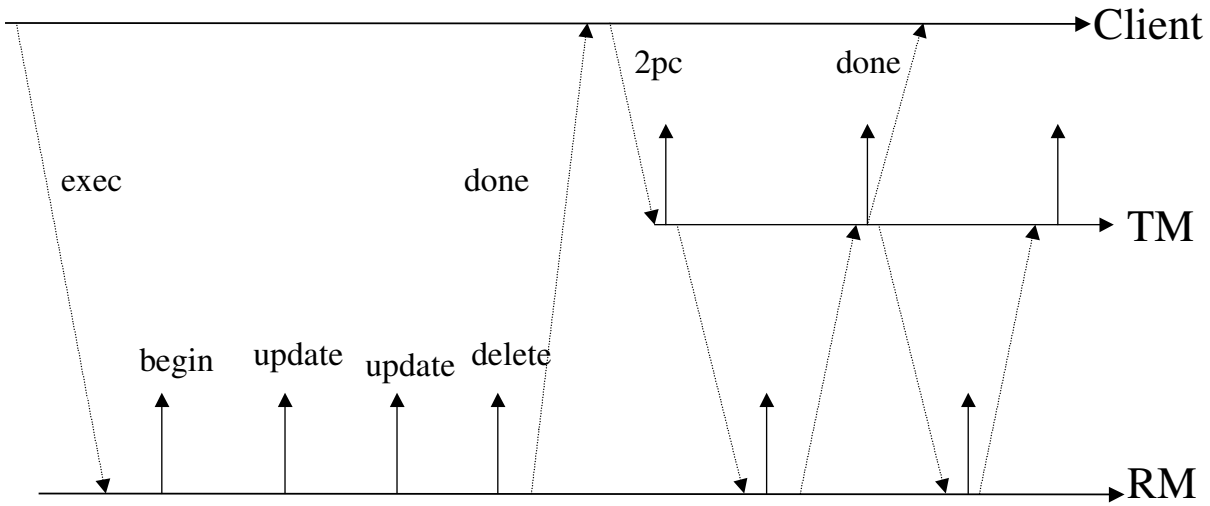
Commit a due fasi - fase 2

- il TM trasmette la sua decisione globale ai RM e setta un secondo timeout
- i RM che sono pronti a ricevere il messaggio di decisione, scrivono i record **commit** o **abort** nel loro log e mandano una conferma (**ack**) al TM
- implementano poi il commit o l'abort scrivendo le pagine nella base di dati come nel caso centralizzato
- il TM ravvoglie tutti i messaggi di **ack** dai RM coinvolti nella seconda fase
- se il timeout scade setta un altro timeout e ripete la trasmissione a tutti i RM da cui non ha ricevuto un **ack**
- quando riceve tutti gli **ack**, il TM scrive il record **complete** sul suo log

Commit a due fasi



Commit a due fasi



Ripristino dei partecipanti

- durante il protocollo di ripristino, gli identificatori delle transazioni in dubbio sono raccolti in un insieme (ready set)
- per ognuna di esse l'esito finale della transazione deve essere richiesto al TM
- questo può succedere sia come risultato di una richiesta diretta (ripristino remoto) da parte del RM o come una ripetizione della seconda fase del protocollo

Ripristino del coordinatore

- quando l'ultimo record nel log è un record **prepare**, il fallimento del TM può aver lasciato qualche RM in una situazione di blocco
- ci sono due opzioni di ripristino possibili:
  - scrivere **global abort** sul log, e procedere con la seconda fase del protocollo
  - ripetere la prima fase, cercando di arrivare a un commit globale
- quando l'ultimo record nel log è una decisione globale, alcuni RM possono essere stati correttamente informati della decisione ed altri possono essere stati lasciati in uno stato bloccato, il TM deve quindi ripetere la seconda fase

Perdita di messaggi e partizionamento della rete

- la perdita di un messaggio **prepare** non è distinguibile dal TM dalla perdita di un messaggio **ready**
- in entrambi i casi il timeout della prima fase scade e viene presa la decisione globale di abort
- allo stesso modo, la perdita di un messaggio di decisione non è distinguibile dalla perdita di un messaggio di **ack**
- in entrambi i casi il timeout della seconda fase scade e la seconda fase viene ripetuta
- un partizionamento della rete non causa problemi ulteriori, perchè la transazione verrà completata con successo solo se il TM e tutti i RM appartengono alla stessa partizione

Ottimizzazioni del 2PC

- il protocollo visto è abbastanza costoso, in particolare assume che tutte le scritture sul log siano sincrone in modo da garantirne la persistenza
- esistono varianti del protocollo che evitano la scrittura sincrona di alcuni record nel log
- protocolli di abort presunto (adottato dalla maggioranza dei DBMS commerciali) e di commit presunto
- nel protocollo di abort presunto quando un TM riceve una richiesta di ripristino remoto da parte di un partecipante in dubbio sulla cui transazione il TM non ha informazioni, viene restituita la decisione di abort

Ottimizzazioni del 2PC

- non sono più necessarie le scritture sincrone dei record di **prepare** e **global abort**, perchè nel caso fossero perse il comportamento di default sarebbe lo stesso
- il record complete inoltre non è più critico per l'algoritmo, e in alcuni sistemi è omesso
- i record da scrivere in maniera sincrona quindi sono **ready**, **global commit** e **commit**
- un'altra ottimizzazione è relativa ai partecipanti read-only
  - rispondono con un messaggio **read-only** al messaggio **prepare** e sospendono l'esecuzione del protocollo
  - vengono ignorati nella seconda fase del protocollo

Altri protocolli di commit

- principale problema del 2PC: un RM può rimanere bloccato a causa della caduta del TM
- **protocollo di commit a quattro fasi:** il processo TM viene replicato da un processo di back up, posto su un altro nodo

  scopo: garantire elevata affidabilità
- **protocollo di commit a tre fasi:** introduce una terza fase di pre-commit

  ha però il difetto di allungare la finestra di incertezza e può portare alla perdita dell'atomicità in caso di partizionamento della rete $\Rightarrow$non è utilizzato in pratica

Basi di dati sistribuite eterogenee

- l'eterogeneità può essere relativa a qualsiasi livello:
  - il software che crea e manipola i dati nei vari siti è diverso
  - il formato e la struttura dei dati nei vari siti è diverso
- l'eterogeneità quindi può manifestarsi a tutti a livelli del sistema di basi di dati:
  - applicazioni scritte in diversi linguaggi
  - diversi linguaggi di interrogazione
  - diversi modelli dei dati
  - diversi DBMS
  - diversi file system
  - ...

Cooperazione tra sistemi pre-esistenti

- il grande sviluppo delle reti negli ultimi anni ha portato a opportunità di integrazione di sistemi informativi esistenti
- le ragioni possono essere diverse:
  - integrazione di componenti sviluppati separatamente
  - cooperazione o fusione di aziende o enti diversi
- l'integrazione di basi di dati è abbastanza difficile: obiettivi di integrazione e standardizzazione troppo ambiziosi sono destinati a fallire
- il modello ideale di una base di dati completamente integrata, che possa essere interrogata in modo trasparente ed efficiente è impossibile da sviluppare e gestire

Cooperazione tra sistemi pre-esistenti

- **cooperazione:** capacità delle applicazioni di un sistema di fare uso dei servizi applicativi messi a disposizione da altri sistemi, anche gestiti da soggetti diversi
- diversi tipi di cooperazione: cooperazione centrata sui processi e cooperazione centrata sui dati
- **cooperazione centrata sui processi**: i sistemi si offrono servizi l'un l'altro, senza rendere i dati remoti esplicitamente visibili
- **cooperazione centrata sui dati:** i dati di un sistema sono visibili ad altri sistemi
- in generale, i sistemi sono distribuiti, eterogenei, autonomi (gestiti da enti e soggetti diversi)

Cooperazione centrata sui dati

- il *livello di trasparenza* misura quanto la distribuzione e l'eterogeneità dei dati siano mascherati
- la *complessità delle operazioni distribuite* misura il grado di coordinazione necessario per svolgere le operazioni sulle basi di dati cooperanti
- il *livello di attualità* dei dati è una misura di quanto sia necessario accedere a dati recenti, tenendo conto che esistono due possibilità:
  - accesso ai dati originali, nel sistema che li gestisce
  - accesso a copie o più in generale a dati derivati
- consideriamo tre architetture diverse: sistemi multidatabase, sistemi basati su dati replicati, sistemi basati su accessi ai dati esterni

Sistemi multidatabase

- ognuna delle basi di dati partecipanti continua a essere utilizzata dai propri utenti
- i sistemi possono essere acceduti anche da moduli, chiamati **mediator**, che mostrano solo la porzione di base di dati che deve essere esportata
- tale porzione viene resa disponibile a un gestore globale, che realizza l'integrazione
- in generale i dati non possono essere modificati per mezzo dei mediator, perchè ogni sistema è autonomo
- caratteristiche:
  - presenta una visione integrata agli utenti
  - alto livello di trasparenza
  - alto livello di attualità, perchè i dati sono acceduti direttamente

Sistemi basati su dati replicati

- garantiscono accesso di sola lettura a copie secondarie dell'informazione fornita esternamente
- tali dati possono essere memorizzati in un data warehouse, che contiene dati estratti da sistemi distribuiti eterogenei e offre una visione globale dei dati
- caratteristiche:
  - alto livello di integrazione
  - alto livello di trasparenza
  - ridotto livello di attualità

Sistemi basati su accessi a dati esterni

- l'integrazione dei dati è effettuata esplicitamente dall'applicazione
- caratteristiche:
  - basso grado di integrazione
  - basso grado di trasparenza
  - livello di attualità che dipende dal contesto

Interoperabilità

- **portabilità** è la possibilità di trasportare programmi da un ambiente ad un altro
  - è tipicamente una proprietà relativa al tempo di compilazione
  - è facilitata da standard a livello di linguaggio (es. SQL)
- **interoperabilità** è l'abilità di interagire mostrata da sistemi eterogenei
  - è tipicamente una proprietà relativa al tempo di esecuzione
  - è facilitata da standard a livello di protocolli di accesso ai dati (es. ODBC)

Interoperabilità

- è il problema principale nello sviluppo di applicazioni eterogenee per basi di dati distribuite
- richiede la disponibilità di funzioni di adattabilità e conversione, che rendono possibile lo scambio di informazioni tra sistemi, reti e applicazioni, anche se eterogenee
- l'interoperabilità è resa possibile dall'utilizzo di protocolli standard quali FTP, SMTP/MIME, ecc
- relativamente alle basi di dati l'interoperabilità è garantita dall'adozione di standard appropriati

  (es. ODBC e JDBC, X-Open DTP)

Basi di dati distribuite eterogenee

- tre possibili categorie di soluzioni:
  - integrazione globale dello schema
  - basi di dati federate
  - architetture basate su linguaggio multidatabase
- trade-off tra condivisione e autonomia
- il termine multidatabase è anche utilizzato per indicare in generale tutte le architetture di basi di dati distribuite eterogenee

Schema globale

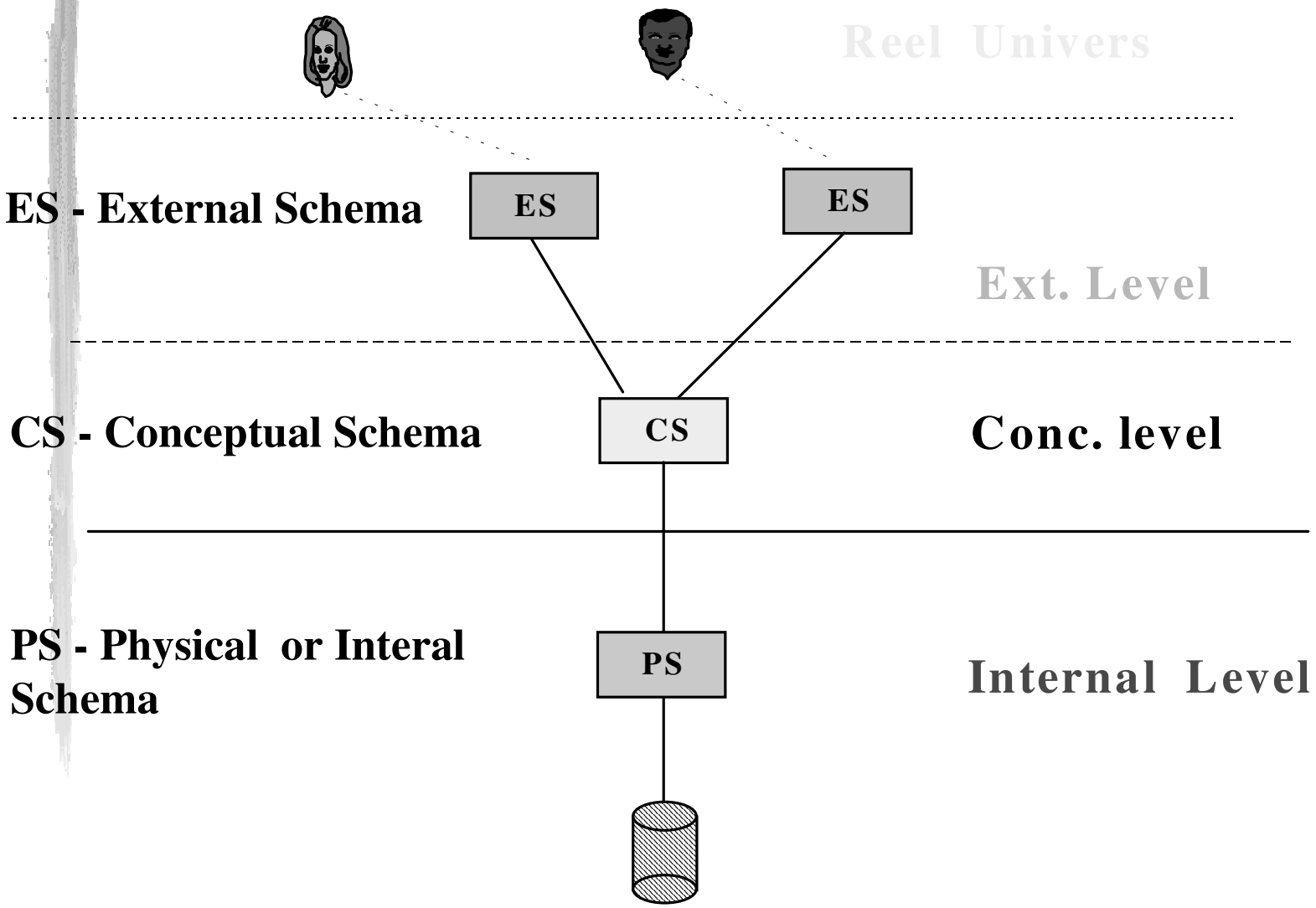
- è stato il primo approccio studiato
- vantaggio: gli utenti hanno una visione e un accesso consistente e uniforme ai dati
- svantaggi:
  - difficile da automatizzare
  - l 'autonomia deve essere sacrificata per risolvere conflitti semantici
  - ancora più difficile se le basi di dati da integrare sono più di due

Assenza di schema globale

- l 'architettura ANSI-SPARC è basata su un modello con un 'unica base di dati:
  - il mondo reale è modellato come **una** base di dati
- il mondo reale, in realtà, è molto spesso rappresentato da basi di dati **multiple**
  - autonome
  - eterogenee semanticamente
  - manipolabili attraverso un linguaggio **multibase**

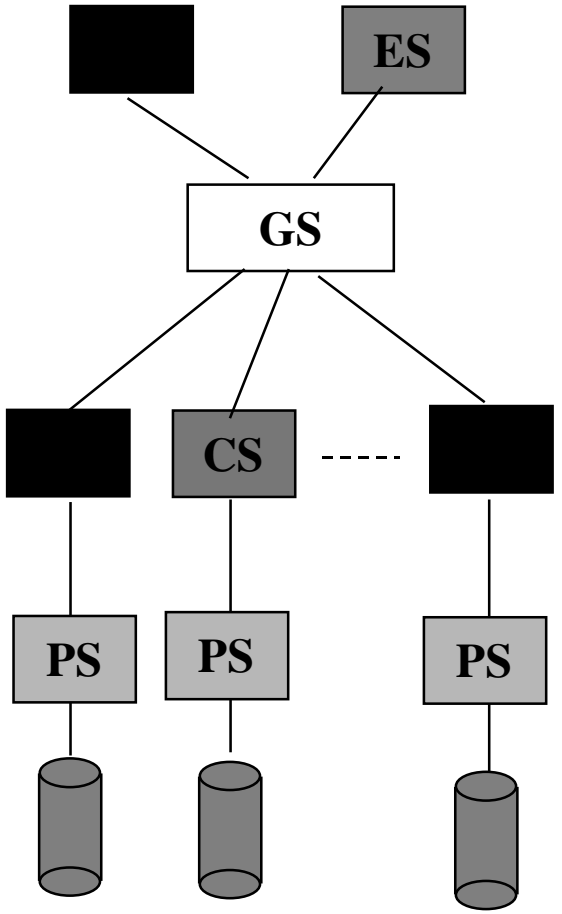
Architettura ANSI-SPARC

Base di dati centralizzata integrata (1960-70)



Problemi dello schema globale e dell'approccio "Bottom-up"

- scalabilità dello schema globale
- utilità dello schema globale per l 'utente locale
- migrazione dei dati dalle basi di dati locali
- applicazioni locali
- eterogeneità semantica
- tempo per l'integrazione/autonomia di ristrutturazione locale
- aggiornamenti
- prestazioni
- viste eterogenee

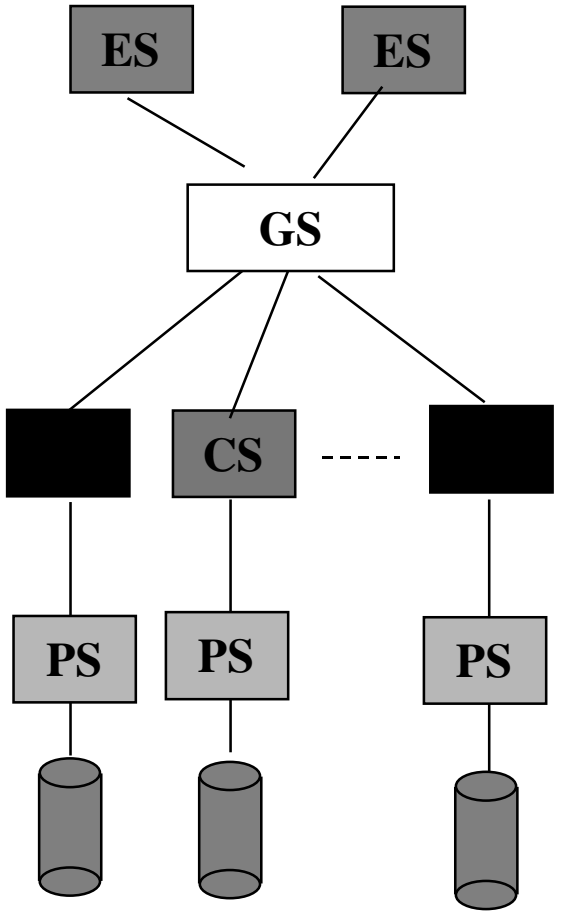


Approccio GS ("bottom-up")

Architettura multidatabase

Linguaggio MDB

- mancanza di schema globale
- gli utenti possono avere i dati memorizzati in basi di dati compatibili con l 'architettura ANSI-SPARC
- ognuno di questi può contenere più classi
- in generale, potrebbe essere impossibile creare uno schema globale
- un multidatabase è una collezione di basi di dati con un linguaggio MDB (es. MSQL)



Approccio GS ("bottom-up")

Architettura multidatabase

Linguaggio MDB

linguaggio per la definizione e la manipolazione di *collezioni* di basi di dati (multidatabase)

- definizione degli schemi esterni del multidatabase
  - ed eventualmente degli schemi globali
- definizione delle dipendenze del multidatabase
  - semantica, integrità, sicurezza, manipolazione...
- formulazione esplicita di interrogazioni sul multidatabase
  - facendo riferimento alle basi di dati attraverso il loro nome
  - con join multidatabase

trova nel DB michelin e nel DB gaumont tutti i ristoranti '**' con un cinema nella stessa strada

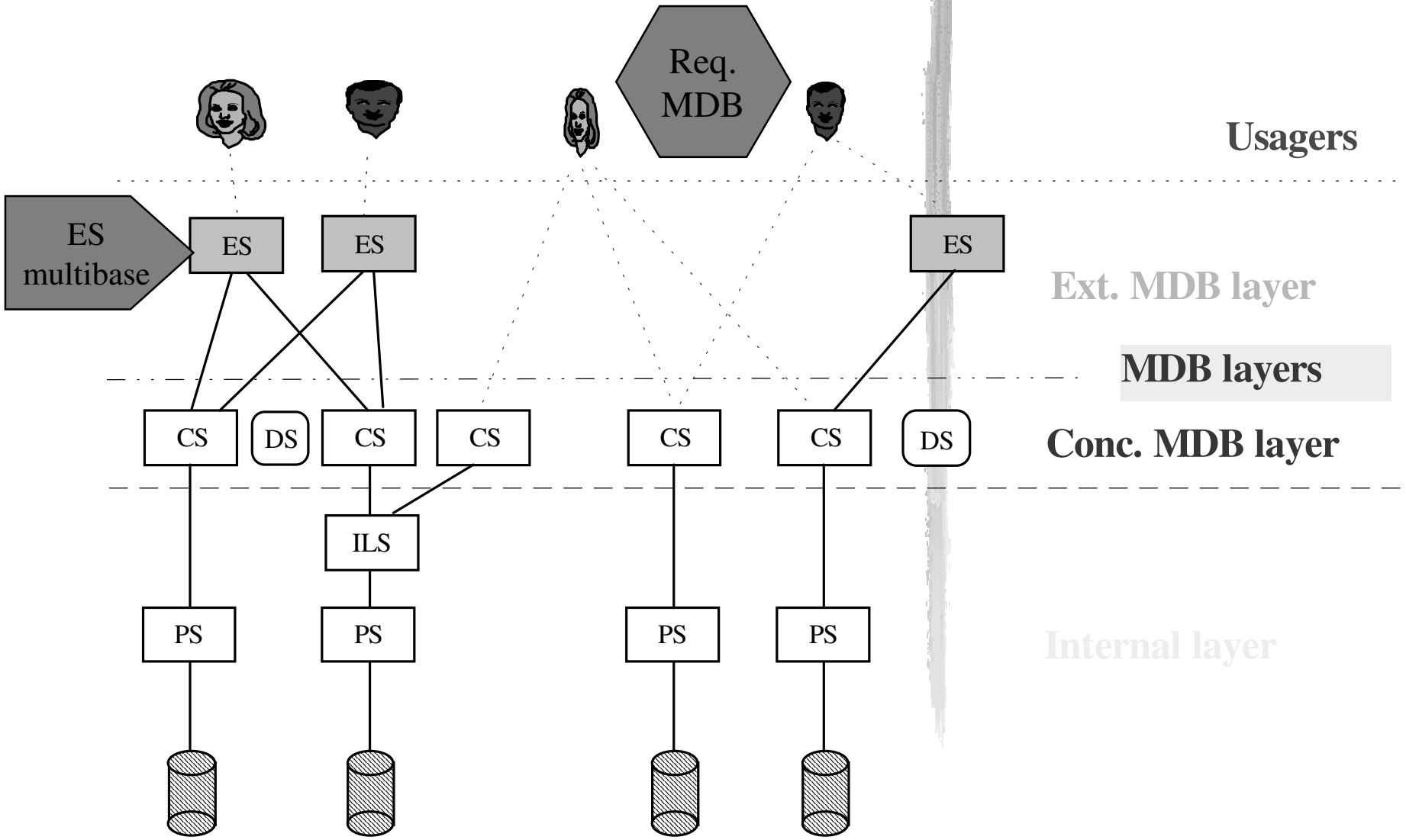
Architettura multidatabase

Internal Logical Sublayer

- i modelli dei dati legacy possono essere eterogenei
  - diversi dialetti SQL
  - relazionali, gerarchici, reticolari
  - OO e object-relational
- è preferibile avere un unico modello al livello MDB
  - è necessario un sotto-livello per le traduzioni
- inoltre i DBA locali potrebbero non voler mostrare alcuni dei dati al livello MDB
- **soluzione: ILS - internal logical schema**
  - non compare nell'ANSI-SPARC
  - chiamato anche **gateway**

Architettura multibase

(W. Litwin & Al, Anni 1980)



Architettura federata

(Hambiger & Mcleod, Anni 1980)

- ogni base di dati dovrebbe essere **autonoma**
- in generale, non ci sarà schema globale
  - l 'integrazione globale è contro l 'autonomia
- le basi di dati utilizzate insieme formano una federazione di basi di dati autonome
- per ogni base di dati in una federazione devono essere specificati tre schemi:
  - ES: export schema
  - IS: import schema
  - PS: private schema : per tutti i dati privati, inclusi quelli dell ' ES e dell 'IS
- ci deve essere un qualche dizionario della federazione (FD)

Architettura federata

(Hambiger & Mcleod, Anni 1980)

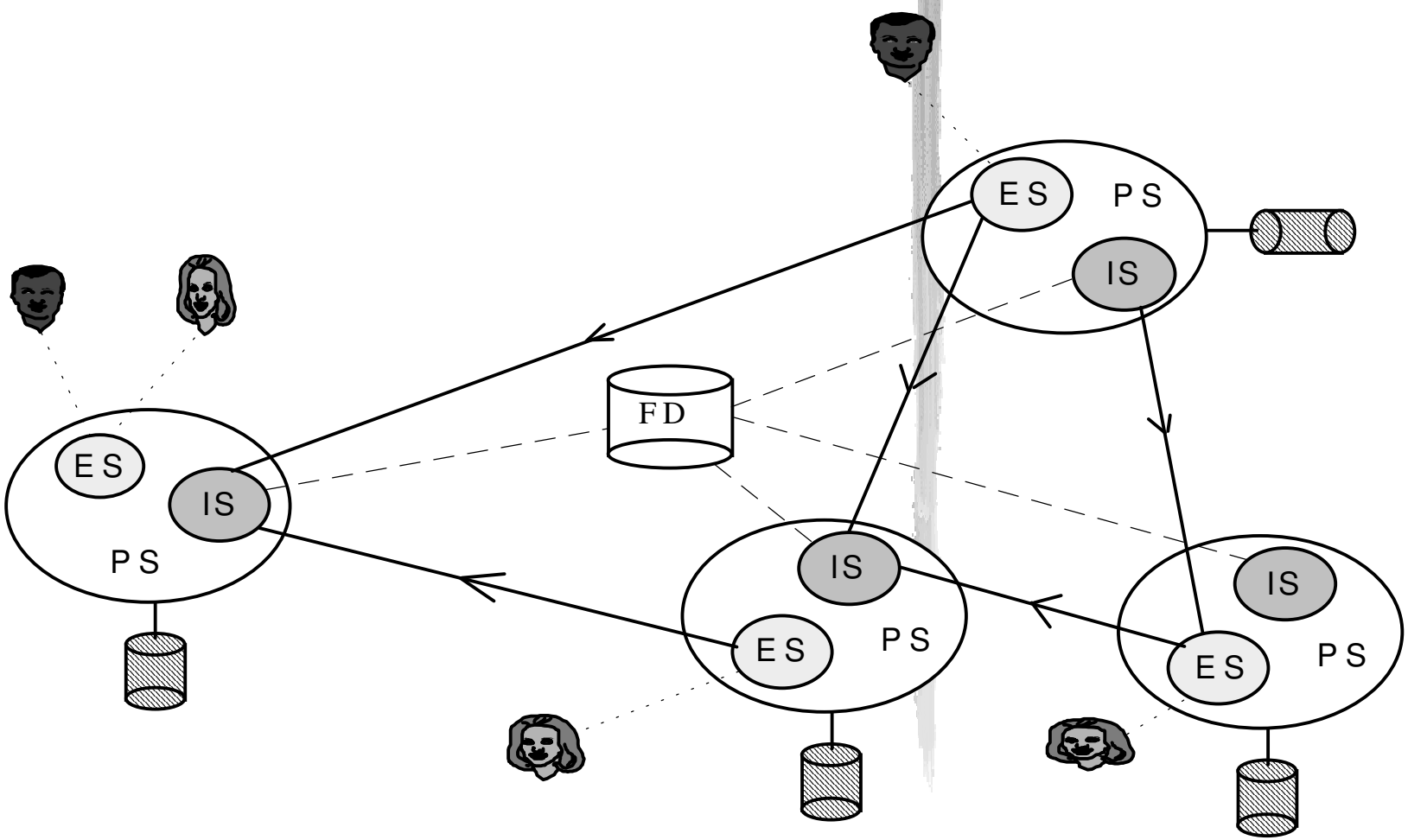


Fig 3. Federated databases architecture

Confronto

- l'architettura MDB è centrata sul concetto di linguaggio MDB
- l'architettura federata è centrata sul concetto di autonomia
  - nessun linguaggio MDB
  - ma la nozione di autonomia è presente anche nell'architettura MDB
- l'architettura MDB è più decentralizzata
- entrambe le architetture sono popolari

Confronto

MDB Arch.	Fed.Arch.
Multidatabase	Federation
Autonomy	Autonomy
MDB Lang.	
¬ GS	¬ GS
CS	PS
MDB ES	IS
DB ES	ES
ILS	ES
DSs	Fed Dict.

Autonomia locale

- possibilità per il DBA di controllare la base di dati locale
  - naming
  - tipi
  - strutture dati
  - strutture fisiche
  - esecuzione delle interrogazioni
  - controllo dell 'accesso
  - priorità assegnata alle interrogazioni locali
- diversi aspetti dell 'autonomia
  - autonomia di progettazione
  - autonomia di comunicazione
  - autonomia di esecuzione
  - autonomia di associazione

Eterogeneità semantica

- differenze nella rappresentazione degli stessi oggetti e proprietà del modo reale
- nomi:André $\Leftrightarrow$ Andrew
- tipi:
  - rappresentazione
  - unità di misura cm/s $\Leftrightarrow$ pied/h
  - precisione 1 g $\Leftrightarrow$ 1 Kg
- strutture dati:

  una tabella in 2 NF $\Leftrightarrow$ più tabelle in 3 NF

Alcune soluzioni possibili

- schemi + descrizione
- protocolli + descrizione
- dizionari dei dati
- Thesaurus
- conversione automatica di rappresentazione
- conversione automatica di unità
- modelli e linguaggi di manipolazione a più alto livello

Ottimizzazione di interrogazioni in sistemi eterogenei

- differenze rispetto ai sistemi distribuiti omogenei:
  - diverse funzionalità dei DBMS componenti
  - diversi modelli dei costi dei DBMS componenti
  - difficoltà di spostare dati in blocco
  - gli ottimizzatori locali hanno comportamenti diversi
  - distribuzione sia rispetto ai siti che rispetto ai DBMS nello stesso sito

dei dati

- la replicazione dei dati è un servizio essenziale per la creazione di molte applicazioni distribuite
- **replicazione sincrona:** tutte le copie di una relazione (frammento) modificato devono essere aggiornate prima del commit della transazione che ha effettuato l'aggiornamento
  - la replicazione dei dati è resa trasparente all'utente
- **replicazione asincrona:** è le copie di una relazione modificata vengono aggiornate solo periodicamente
  - diverse copie dello stato possono essere relative a stati diversi della base di dati, quindi contenere valori diversi
  - gli utenti devono essere consci della replicazione e distribuzione dei dati

Sistemi basati su replicazione dei dati

- è fornita da prodotti chiamati data replicators, la cui funzione principale è mantenere la consistenza tra le copie
- tali prodotti operano in maniera trasparente rispetto alle applicazioni in esecuzione sul DBMS server
- in generale, c'è una copia principale e varie copie secondarie, e gli aggiornamenti sono propagati in modo asincrono (senza il 2PC)
- la propagazione è incrementale quando si basa su variazioni nei dati
- l'uso della replicazione rende un sistema meno sensibile ai guasti, perchè se la copia principale non è disponibile è sempre possibile usare una delle copie secondarie

Replicazione sincrona

- **voting:** la transazione deve scrivere la maggioranza delle copie per modificare un oggetto e deve leggere abbastanza copie da essere sicura di vedere almeno una delle copie più recenti
  - es. 10 copie, 7 aggiornate, 4 lette
  - ogni copia ha un numero di versione
  - generalmente non molto applicabile perchè le letture sono frequenti
- **read-any write-all:** le scritture sono più lente e le letture sono più veloci, rispetto al voting
  - è l'approccio più comune per la replicazione sincrona
- a seconda della tecnica scelta i lock da acquisire sui dati sono diversi

Replicazione sincrona

- il costo della replicazione sincrona è elevato
- prima che una transazione possa effettuare il commit deve ottenere lock su tutte le copie modificate
  - invia richieste di lock ai siti remoti, e mentre attende la risposta, tiene bloccati mediante lock altri dati
  - se un sito o un collegamento si guasta la transazione non può effettuare il commit finchè non sono ripristinati
  - anche in assenza di guasti il commit deve seguire un protocollo costoso con molti messaggi
- questo è il motivo per cui la replicazione asincrona è più utilizzata

Replicazione asincrona

- permette alla transazione che esegue le modifiche di effettuare il commit prima che tutte le copie siano state modificate (e nelle letture si guarda comunque un'unica copia)
- gli utenti devono essere consapevoli di quale copia stanno leggendo e le copie possono essere non sincronizzate per brevi periodi di tempo
- due approcci: sito primario e peer-to-peer
- la differenza sta in quante copie sono "aggiornabili" o "copie principali"

Replicazione peer-to-peer

- **replicazione simmetrica:** le modifiche possono essere eseguite su una qualsiasi delle copie
- i cambiamenti a una copia devono essere propagati alle altre copie in qualche modo
- è possibile introdurre conflitti, perchè due copie della stessa informazione sono gestite in modo concorrente senza controllo della concorrenza
- tecniche capaci di rivelare inconsistenze se si verificano, la cui correzione dipende dall'applicazione
- utilizzata soprattutto quando non possono sorgere conflitti:
  - ogni copia master è relativa a un frammento disgiunto
  - i diritti di aggiornamento sono riservati ad un master alla volta

Replicazione con sito primario

- esattamente una copia di una relazione è designata copia primaria o master
- le repliche negli altri siti non possono essere aggiornate direttamente
- la copia primaria è accessibile a tutti
- gli altri siti sottoscrivono (frammenti) di tale relazione, che sono copie secondarie
- il problema principale è come propagare i cambiamenti della copia primaria alla copia principale
- viene effettuato in due passi: prima si catturano i cambiamenti effettuati dalle transazioni che hanno effettuato il commit e poi questi cambiamenti vengono effettuati

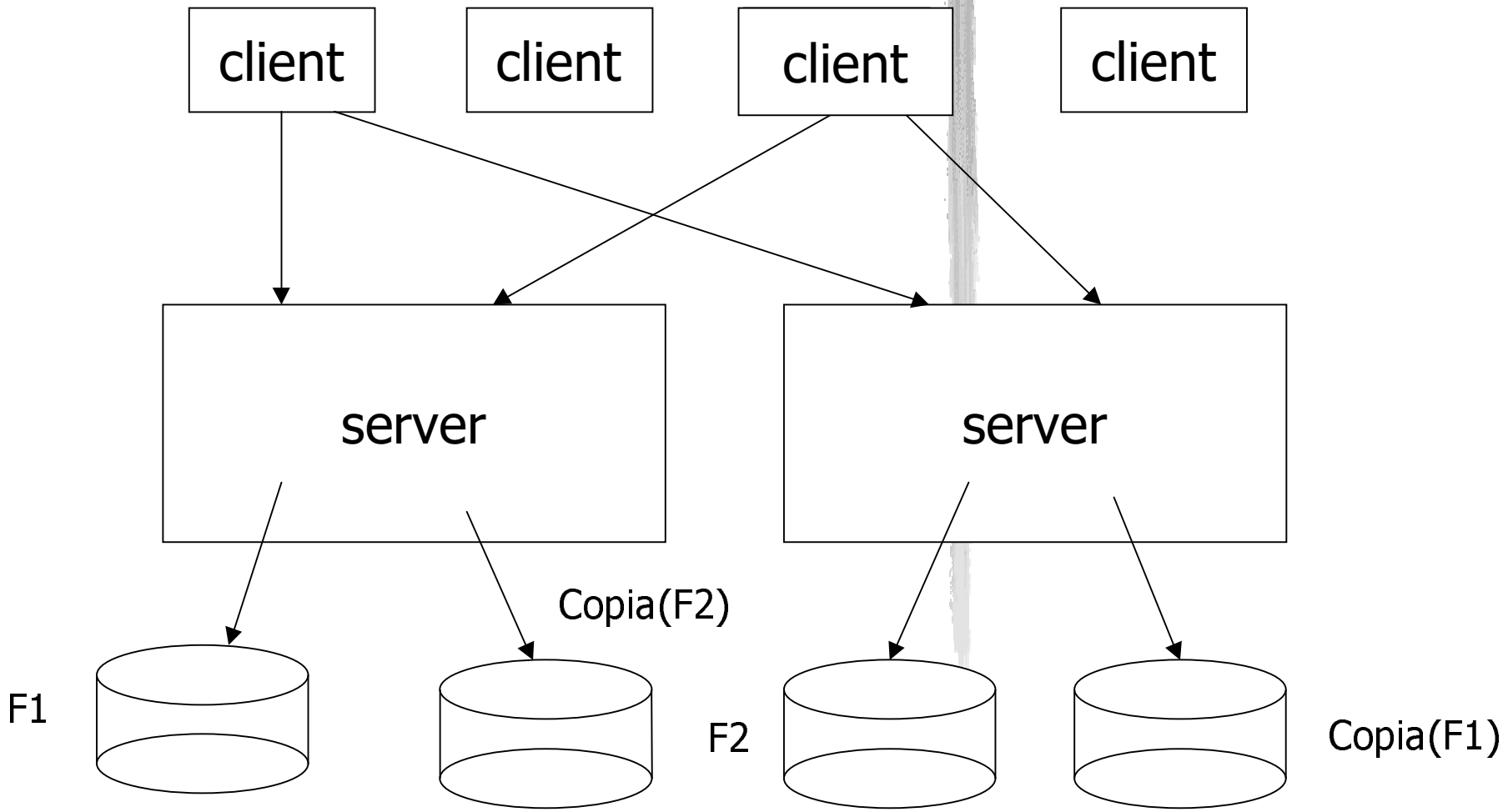
Replicazione con sito primario - passo capture

- **capture basato su log:** il log (mantenuto per la gestione del ripristino) viene utilizzato per generare una tabella delle modifiche (Change Data Table CDT)
- se questo viene effettuato quando il log viene scritto su disco bisogna poi rimuovere in qualche modo i cambiamenti effettuati da transazioni che hanno poi abortito
- **capture procedurale:** una procedura che viene invocata automaticamente (trigger) che effettua il capture, prendendo uno snapshot della copia primaria
- capture basato su log è meglio (più economico e più veloce) ma si basa su dettagli del log proprietari

Replicazione con sito primario - passo apply

- i processi apply sul sito secondario ricevono periodicamente uno snapshot o dei cambiamenti contenuti nella CDT dal sito primario, e aggiornano la copia
- le repliche possono essere viste come viste sulla relazione modificata
- in quest'ottica la replicazione consiste nell'aggiornare incrementalmente la vista materializzata al variare della relazione
- capture basato su log + apply continuo: si minimizzano i ritardi nella propagazione delle modifiche
- capture procedurale + apply application-driven: modo più flessibile di gestire le modifiche

Sistemi basati su replicazione dei dati



Sistemi basati su replicazione dei dati

due siti identici: ogni sito gestisce l'intera base di dati; metà è la copia principale e l'altra metà è la copia secondaria

- tutte le transazioni sono inviate alla copia principale e poi ridirezionate alla copia secondaria
- gli "access point" del sistema sono collegati ad entrambi i siti
  - in caso di un fallimento che coinvolga solo un sito, il sistema è in grado di ridirezionare quasi istantaneamente tutte le transazioni all'altro sito, che è abbastanza potente da reggere l'intero carico
  - quando il problema viene risolto, il gestore della replicazione ripristina i dati in modo trasparente e risetta i due siti alle operazioni normali

Sistemi basati su replicazione dei dati

- **replicazione disconnessa**
- si verifica nei sistemi mobili, in cui la connessione con la base di dati si può interrompere
  - es: rappresentante
  - si collega a base di dati per verificare disponibilità della merce e per caricare gli ordini ricevuti
  - normalmente è disconnesso dalla base di dati e lavora sulla copia
  - la copia viene "riconciliata" con la base di dati alla fine dell'attività di vendita

Basi di dati e parallelismo

- settore sviluppato negli anni novanta con la diffusione delle architetture multiprocessore, dopo il fallimento delle architetture speciali per basi di dati (*database machines*) degli anni ottanta
- il parallelismo è possibile con architetture multiprocessore sia con memoria condivisa, che senza, sebbene con diverse soluzioni tecniche
- in una **base di dati parallela** è si cerca di migliorare le prestazioni mediante l'implementazione parallela di varie operazioni
- anche se i dati possono essere memorizzati in modo distribuito, la distribuzione è guidata solo da considerazioni legate alle prestazioni

Parallelismo

- la ragione del successo del parallelismo in ambito basi di dati è che le operazioni di gestione dati sono di natura abbastanza ripetitiva, e possono essere svolte efficientemente in parallelo
- una scansione completa di una grande base di dati può essere eseguita mediante *n* scansioni, ognuna su una porzione della base di dati
- se la base di dati è memorizzata su *n* dischi diversi, gestiti da n diversi processori, il tempo di risposta sarà circa *1/n* il tempo richiesto da una scansione sequenziale

Parallelismo inter-query

- il parallelismo è chiamato *inter-query* quando diverse interrogazioni vengono eseguite in parallelo
- il carico imposto al DBMS è caratterizzato tipicamente da transazioni semplici e frequenti (fino a migliaia di transazioni al secondo)
- il parallelismo viene introdotto per moltiplicare il numero di server e allocare il numero ottimale di richieste ad ogni server
- in molti casi, le interrogazioni sono ridirezionate ai server da un processo *dispatcher*
- utile per sistemi OLTP

Parallelismo intra-query

- il parallelismo è chiamato *intra-query* quando diverse parti della stessa interrogazione vengono eseguite in parallelo
- il carico del DBMS è caratterizzato tipicamente da poche interrogazioni molto complesse, che vengono decomposte in varie sottointerrogazioni parziali, eseguite in parallelo
- in genere, le interrogazioni soo eseguite l'una dopo l'altra, usando tutto il sistema multi-processore per ogni interrogazione
- utile per sistemi OLAP

Parallelismo e frammentazione dei dati

- il parallelismo è generalmente associato alla frammentazione dei dati: il frammenti sono distribuiti tra molti processori e allocati a diversi dispositivi di memorizzazione secondaria
- consideriamo le relazioni

CONTO(NumConto,Nome,Saldo)
TRANSAZIONE(NumConto,Data,NumProgr,TipoTrans,Ammontare)

- frammentata in base a intervalli predefiniti del numero di conto

Parallelismo inter-query e frammentazione dei dati

- un esempio tipico di interrogazione OLTP con parallelismo inter-query è:

```
procedureQuery5(:num-conto, :totale);
  SELECT Saldo INTO :totale
  FROM CONTO
  WHERE NumConto = :num-conto;
end procedure;
```

- un'interrogazione di questo tipo è direzionata verso il frammento specifico a seconda del predicato di selezione

Parallelismo intra-query e frammentazione dei dati

- un esempio tipico di interrogazione OLAP con parallelismo intra-query è:

```
procedureQuery6();
        SELECT NumConto,SUM(Ammontare)
        FROM CONTO NATURAL JOIN TRANSACTION
        WHERE Data >= 1.1.1998 AND Data < 1.1.1999
        GROUP BY NumConto
        HAVING SUM(Ammontare) > 100000;
end procedure;
```

- un'interrogazione di questo tipo è portata avanti sui vari frammenti in parallelo

Join distribuiti

- consideriamo i join di coppie di frammenti che corrispondono allo stesso intervallo di numeri di conto
- i join tra i vari frammenti che matchano possono essere effettuati in parallelo
- è essenziale per il parallelismo intr-query
- l'esecuzione parallela di *n* join di frammenti di dimensione *1/n* è ovviamente preferibile all'esecuzione di un singolo join che coinvolge tutta la relazione
- in generale, quando la frammentazione iniziale non permette l'esecuzione distribuita dei join presenti nell'interrogazione, i dati vengono ridistribuiti dinamicamente per permettere join distribuiti

Speed-up e scale-up

- la curva di **speed-up** caratterizza solo il parallelismo inter-query e misura il miglioramento di servizi, misurato in tps (transazioni al secondo) rispetto all'incremento nel numero di processori
  - nella situazione ideale, i servizi crescono circa linearmente nell'incremento di processori
- la curva di **scale-up** caratterizza sia il parallelismo inter-query che quello intr-query e misura il costo medio di una singola transazione rispetto all'incremento nel numero di processori
  - nella situazione ideale, il costo medio rimane circa costante con un aumento dei processori
  - si dice che il sistema "scala" bene