Basi di dati
II

6 – BASI DI DATI DISTRIBUITE

Paradigmi per la distribuzione dati

I-Architettura client-server

Separazione del “database server” dal client

II- Basi di dati distribuite

Molti “database server” usati dalla stessa applicazione

III- Applicazioni di database cooperative

Ciascun server mantiene la sua autonomia

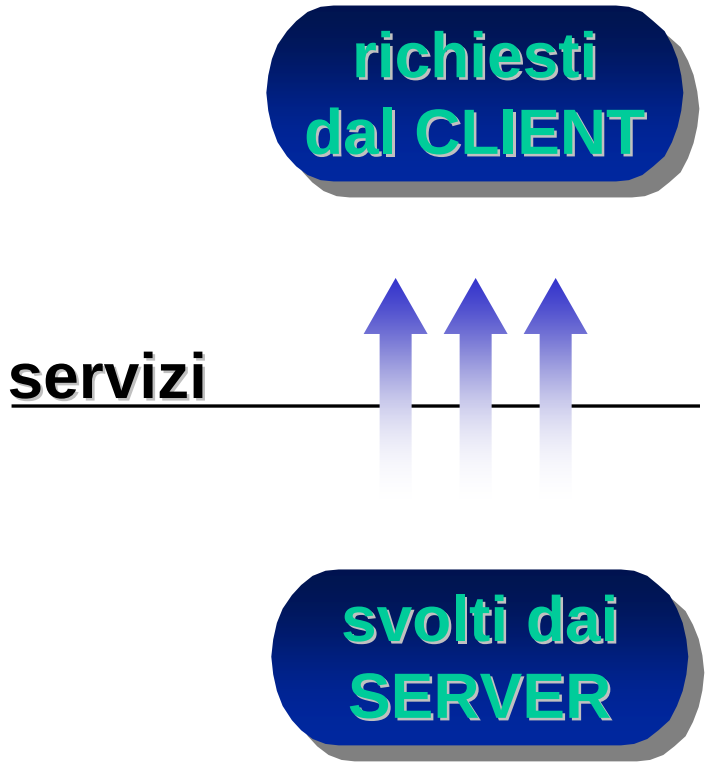
IV- Basi di dati replicate

Dati che logicamente rappresentano la stessa informazione fisicamente immagazzinati su server differenti

I- ARCHITETTURE CLIENT-SERVER

Paradigma client-server

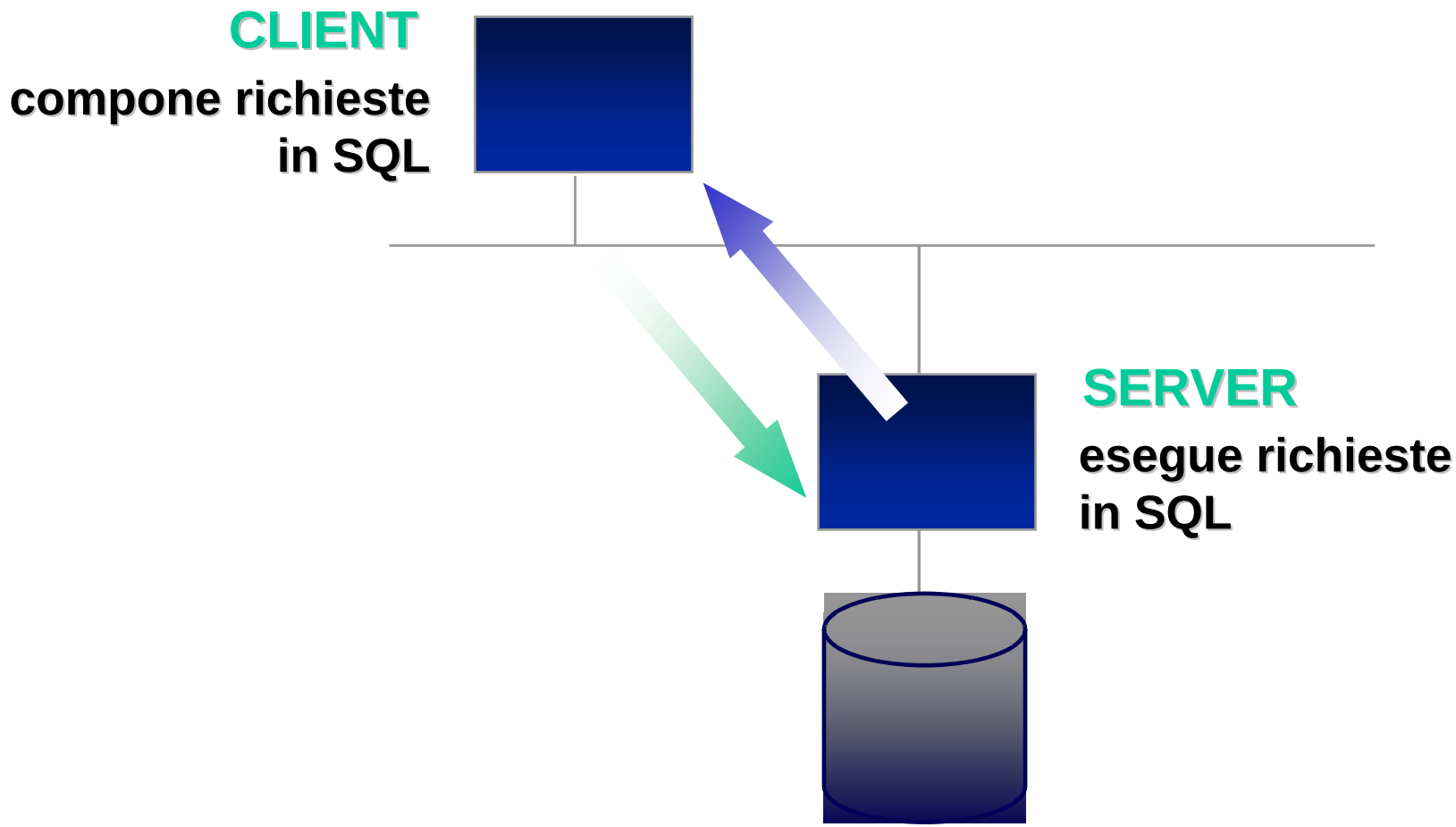
- **Tecnica per strutturare sistemi software**
- **Viene resa "pubblica" una "interfaccia di servizi"**
- **Due tipologie di sistemi:**
  CLIENT:
  richiedono i servizi
  SERVER:
  forniscono i servizi



Client-server
nei sistemi informativi

- **Separazione funzionale ideale**
  - **CLIENT : presentazione dell'informazione**
  - **SERVER : gestione dei dati**
- **SQL : il linguaggio ideale per separare gli ambienti**
  - **CLIENT : formula query, elabora risultati**
  - **SERVER : esegue query**
  - **RETE : trasferisce i comandi di attivazione (es: di procedure SQL) e i risultati**

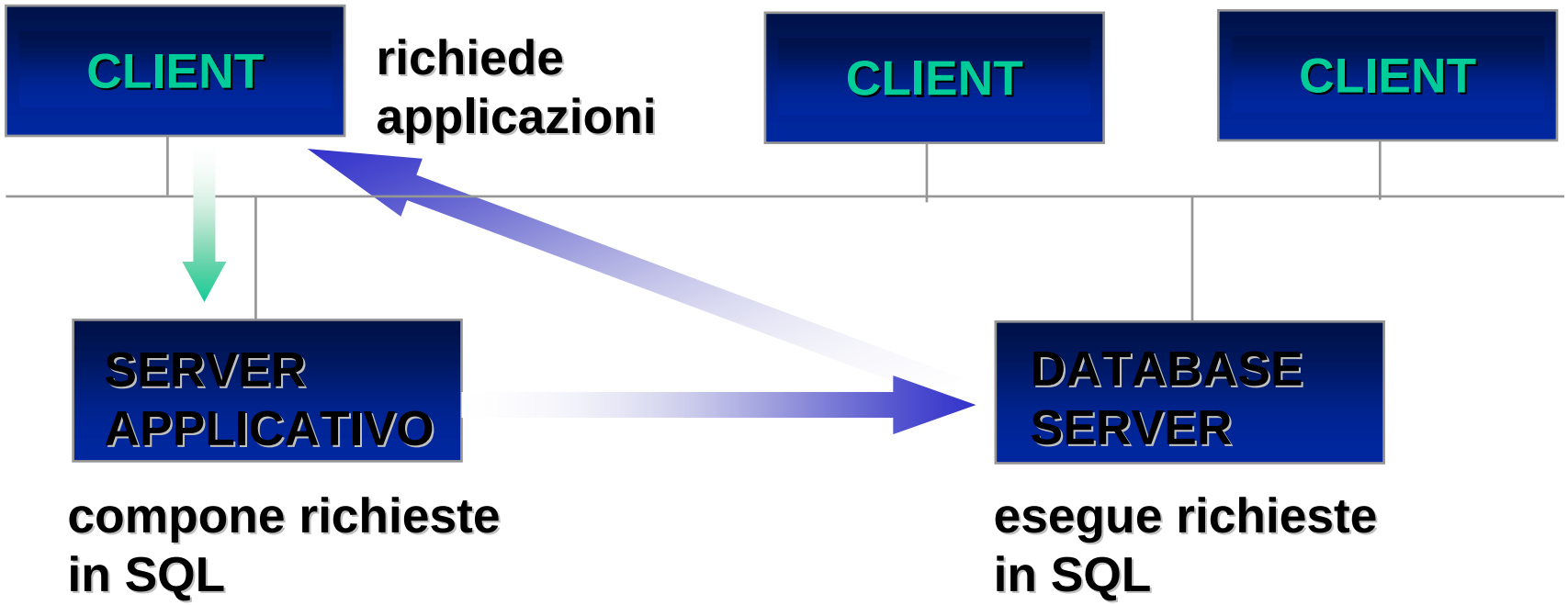
Architettura client-server classica



Client “thin” o “thick”

- **Thick Client**:
  - include la logica dell’applicazione e lascia al server solo la responsabilità della gestione dei dati; il linguaggio di comunicazione è SQL
- **Thin Client**:
  - fa il meno possibile (essenzialmente la presentazione)
  - usato con terminali poco potenti;
  - la logica dell’applicazione è sul server
  - la comunicazione è essenzialmente tramite chiamate a procedura (remota)
- **Soluzioni intermedie:**
  - una certa logica sul client e una certa logica sul server (solo se vi è una naturale separazione tra le funzioni).
  - soluzioni a tre (o più) livelli

Architettura con server applicativo



Architettura con server applicativo (2)

- **Architettura two-tier : “client” e “server”**
- **Architettura three-tier :** è presente un secondo server, noto come “application server”, responsabile della gestione della logica della applicazione comune a molti client:
  - il client è **thin** ed è responsabile dell’interfaccia con l’utente finale;
  - le applicazioni web ricadono quasi sempre in questa categoria
  - le applicazioni cooperative spesso la estendono

II – BASI DI DATI DISTRIBUITE

Motivazioni della distribuzione dei dati

- **Natura intrinsecamente distribuita delle organizzazioni**
- **Evoluzione degli elaboratori**
  - **aumento della capacità elaborativa**
  - **riduzione di prezzo**
- **Evoluzione della tecnologia dei DBMS**
- **Standard di interoperabilità**

Tipologie di basi di dati distribuite

a **RETE :**

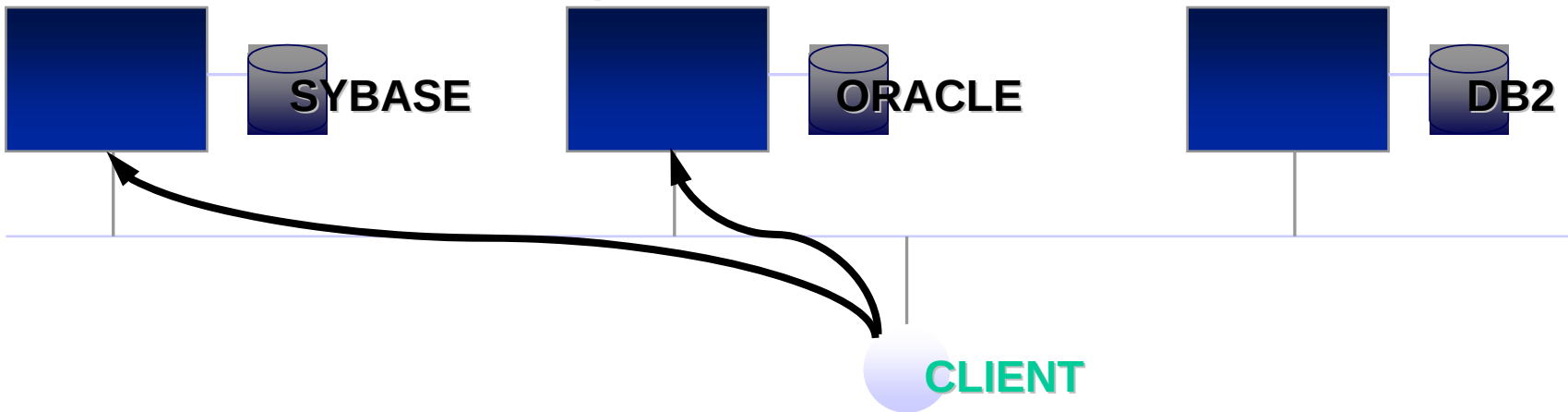
LAN (Local Area Network)

WAN (Wide Area Network)

b **DBMS :**

Sistema omogeneo

Sistema eterogeneo



Tipici esempi di applicazioni

	LAN	WAN
OMOGENEO	Applicazioni gestionali e finanziarie	Sistemi di prenotazione, applicazioni finanziarie
ETEROGENEO	Applicazioni gestionali interfunzionali	Sistemi di prenotazione integrati, sistemi interbancari

Problemi delle basi di dati distribuite

- 1. Autonomia e cooperazione
- 2. Trasparenza
- 3. Efficienza
- 4. Operazioni distribuite

1. AUTONOMIA E COOPERAZIONE

Autonomia locale e cooperazione

L'esigenza di autonomia:

- **Una reazione ai " Centri EDP "**
- **Portare competenze e controllo laddove vengono gestiti i dati**
- **Rendere la maggior parte delle applicazioni NON distribuite (!)**

L'esigenza di cooperazione:

- **Alcune applicazioni sono intrinsecamente distribuite e richiedono l'accesso a più basi di dati**

2. TRASPARENZA

Frammentazione dei dati

Scomposizione delle tabelle in modo da consentire la loro distribuzione

proprietà:

- **Completezza**
- **Ricostruibilità**

Frammentazione orizzontale

FRAMMENTI:

insiemi di tuple

COMPLETEZZA:

presenza

di tutte le tuple

RICOSTRUZIONE:

unione

Frammentazione verticale

FRAMMENTI:

insiemi di attributi

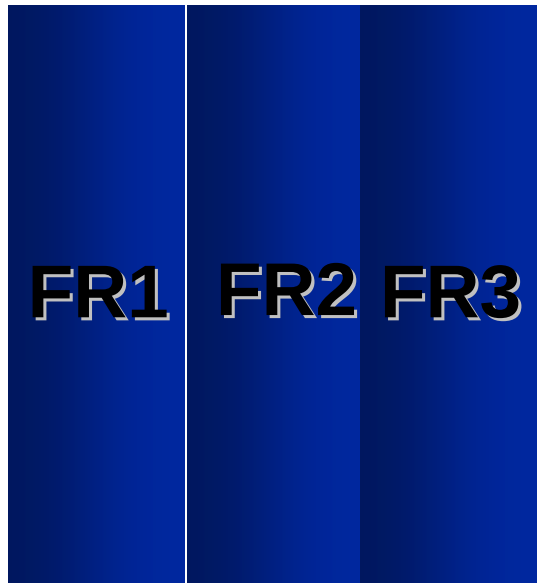
COMPLETEZZA:

presenza

di tutti gli attributi

RICOSTRUZIONE:

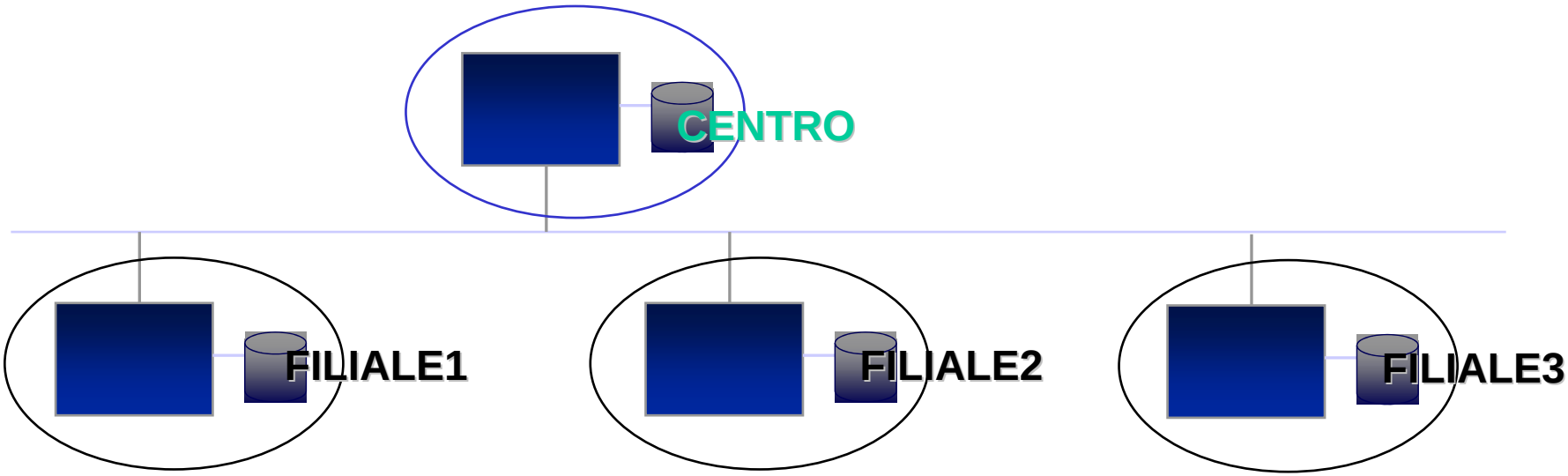
join sulla chiave



Esempio: conti correnti bancari

CONTO-CORRENTE (NUM-CC,NOME,FILIALE,SALDO)

TRANSAZIONE (NUM-CC,DATA,PROGR,AMMONTARE, CAUSALE)



Frammentazione orizzontale principale

$Ri = \sigma_{Pi} R$

esempio:

$CONTO1 = \sigma_{Filiale=1} CONTO\text{-}CORRENTE$

$CONTO2 = \sigma_{Filiale=2} CONTO\text{-}CORRENTE$

$CONTO3 = \sigma_{Filiale=3} CONTO\text{-}CORRENTE$

Frammentazione orizzontale derivata

Si = S ⋉ Ri

- **con ⋉ operatore di semi-join che calcola in Si tutte le tuple di S che soddisfano la condizione di join con Ri**

esempio:

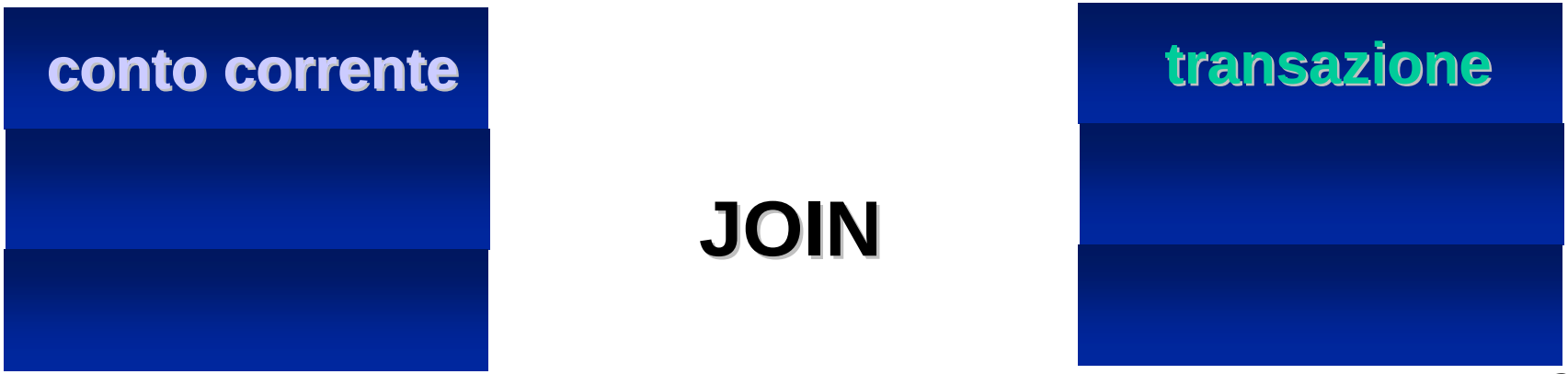
TRANS1 = TRANSAZIONE ⋉ CONTO1

TRANS2 = TRANSAZIONE ⋉ CONTO2

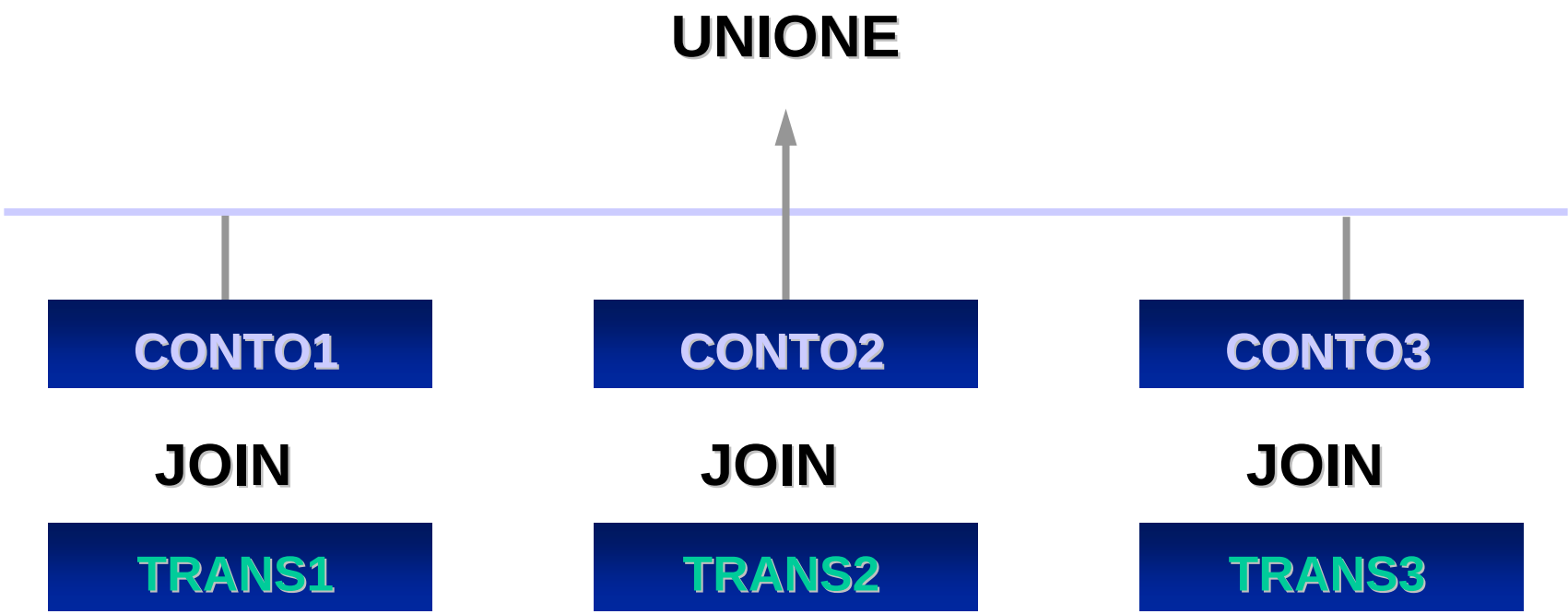
TRANS3 = TRANSAZIONE ⋉ CONTO3

Join distribuito

E' l'operazione di piu' onerosa consideriamo:



Join distribuito(2)



Requisiti per il join distribuito

I domini degli attributi di join devono essere partizionati e ogni partizione assegnata ad una coppia di frammenti

(ad esempio su valori numerici tra 1 e 300000:

- **da 1 a 100000**
- **da 100000 a 200000**
- **da 200000 a 300000)**

In molti sistemi paralleli i dati vengono inizialmente ridistribuiti sui dischi per ottenere questa distribuzione

Allocazione dei frammenti

Rete :

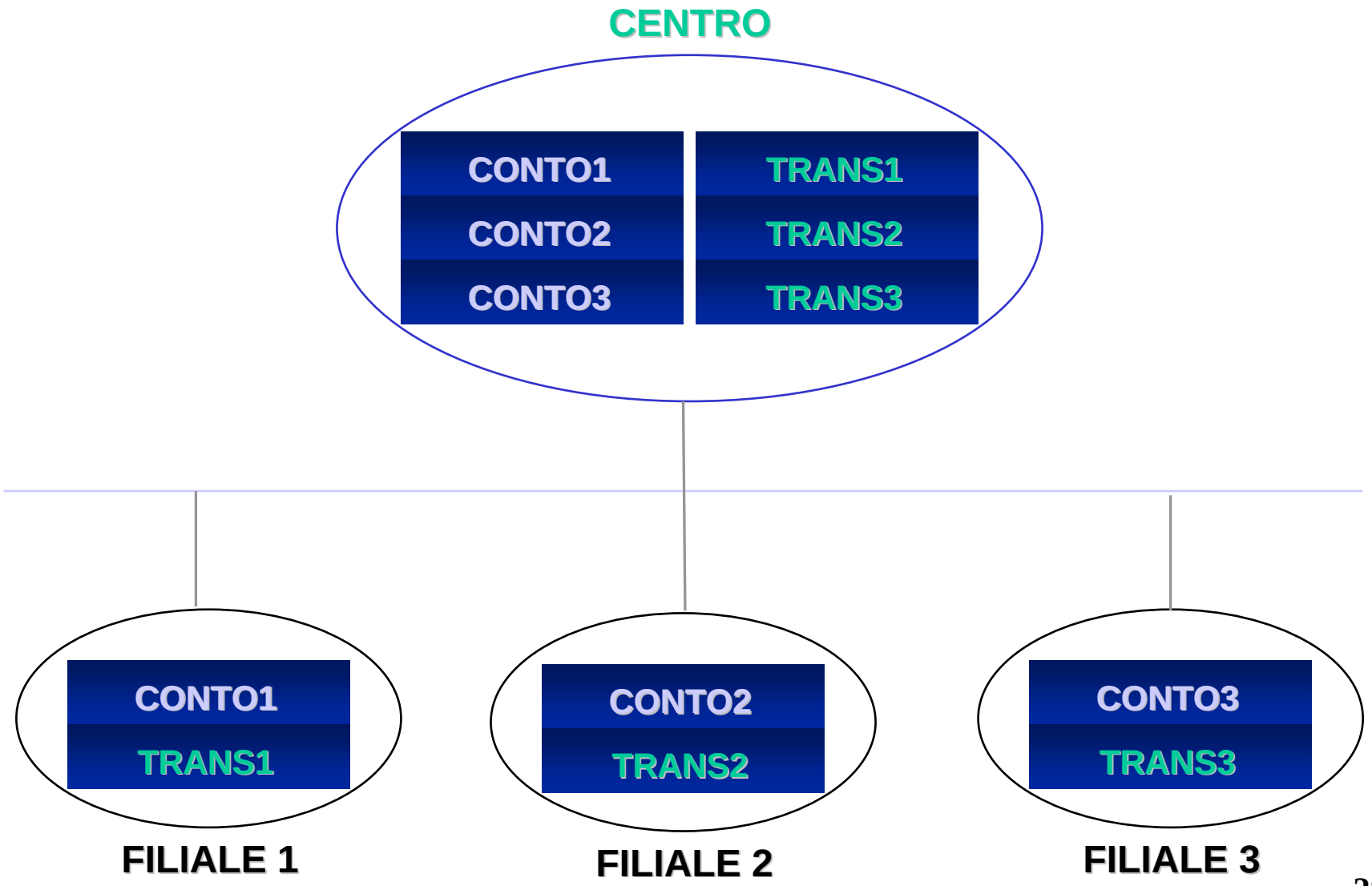
3 siti periferici, 1 sito centrale

Allocazione:

locale

centrale

Allocazione dei frammenti



Livelli di trasparenza

Modalita’ per esprimere interrogazioni offerte dai DBMS commerciali:

LIVELLI: FRAMMENTAZIONE
ALLOCAZIONE
LINGUAGGIO

Trasparenza di frammentazione

QUERY :

estrarre il saldo del conto corrente 45

```
SELECT SALDO
FROM CONTO-CORRENTE
WHERE NUM-CC=45
```

Trasparenza di allocazione

IPOTESI :

Conto corrente 45 presso la Filiale 1 (locale)

```
SELECT SALDO
FROM CONTO1
WHERE NUM-CC=45
```

Trasparenza di allocazione

IPOTESI : Allocazione incerta, probabilmente alla filiale 1

```
SELECT SALDO FROM CONTO1
        WHERE NUM-CLI=45
IF (NOT FOUND) THEN
( SELECT SALDO FROM CONTO2
        WHERE NUM-CLI=45
 UNION
 SELECT SALDO FROM CONTO3
        WHERE NUM-CLI=45 )
```

Trasparenza di linguaggio

```
SELECT SALDO FROM CONTO1@1
        WHERE NUM-CLI=45
IF (NOT FOUND) THEN
( SELECT SALDO FROM CONTO2@C
        WHERE NUM-CLI=45
  UNION
  SELECT SALDO FROM CONTO3@C
        WHERE NUM-CLI=45 )
```

Trasparenza di frammentazione

QUERY :

estrarre i movimenti dei conti con saldo negativo

```
SELECT CC-NUM, PROGR, AMMONTARE
FROM CONTO-CORRENTE  AS C
  JOIN TRANSAZIONE AS T
    ON C.NUM-CC=T.NUM-CC
WHERE SALDO < 0
```

Trasparenza di allocazione (join distribuito)

```
SELECT  CC-NUM, PROGR, AMMONTARE
        FROM CONTO1 JOIN TRANS1 ON ……
        WHERE SALDO < 0
UNION
SELECT CC-NUM, PROGR, AMMONTARE
        FROM CONTO2 JOIN TRANS2 ON ……
        WHERE SALDO < 0
UNION
SELECT  CC-NUM, PROGR, AMMONTARE
        FROM CONTO3 JOIN TRANS3 ON ……
        WHERE SALDO < 0
```

Trasparenza di linguaggio

```
SELECT  CC-NUM, PROGR, AMMONTARE
        FROM CONTO1@1 JOIN TRANS1@1 ON …...
        WHERE SALDO < 0
UNION
SELECT CC-NUM, PROGR, AMMONTARE
        FROM CONTO2@C JOIN TRANS2@C ON …...
        WHERE SALDO < 0
UNION
SELECT  CC-NUM, PROGR, AMMONTARE
        FROM CONTO3@C JOIN TRANS3@C ON …...
        WHERE SALDO < 0
```

Trasparenza di frammentazione

UPDATE :

sposta il cliente 45 dalla filiale 1 alla filiale 2

```
UPDATE CONTO-CORRENTE
SET FILIALE = 2
WHERE NUM-CC=45
      AND FILIALE=1
```

Trasparenza di allocazione (e replicazione!!)

```
INSERT INTO CONTO2
  SELECT * FROM CONTO1 WHERE NUM-CC=45

INSERT INTO TRANS2
  SELECT * FROM TRANS1 WHERE NUM-CC=45

DELETE FROM CONTO1 WHERE NUM-CC=45

DELETE FROM TRANS1 WHERE NUM-CC=45
```

Trasparenza di linguaggio

```
INSERT INTO CONTO2@2
  SELECT * FROM CONTO1 WHERE NUM-CC=45
INSERT INTO CONTO2@C
   SELECT * FROM CONTO1 WHERE NUM-CC=45
INSERT INTO TRANS2@2
  SELECT * FROM TRANS1 WHERE NUM-CC=45
INSERT INTO TRANS2@C
  SELECT * FROM TRANS1 WHERE NUM-CC=45
```

(in modo analogo: 2 coppie di DELETE)

Trasparenza di linguaggio

DELETE FROM CONTO1@1 WHERE NUM-CC=45
DELETE FROM CONTO1@C WHERE NUM-CC=45

DELETE FROM TRANS1@1 WHERE NUM-CC=45
DELETE FROM TRANS1@C WHERE NUM-CC=45

3. EFFICIENZA

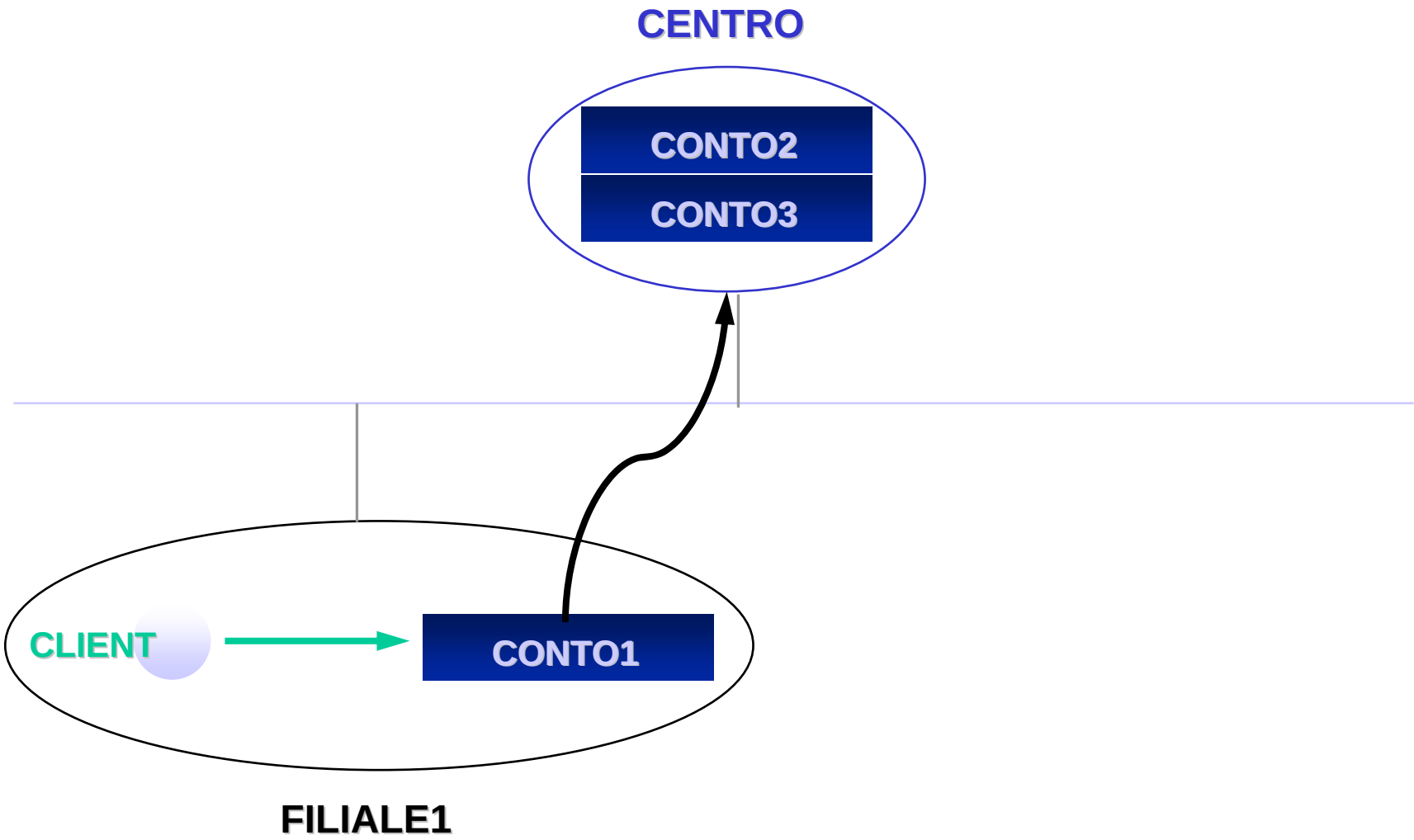
Ottimizzazioni

- Le applicazioni distribuite possono essere rese più efficienti:
  - Usando la **conoscenza delle proprietà logiche dei frammenti** (ma allora i programmi non sono flessibili):

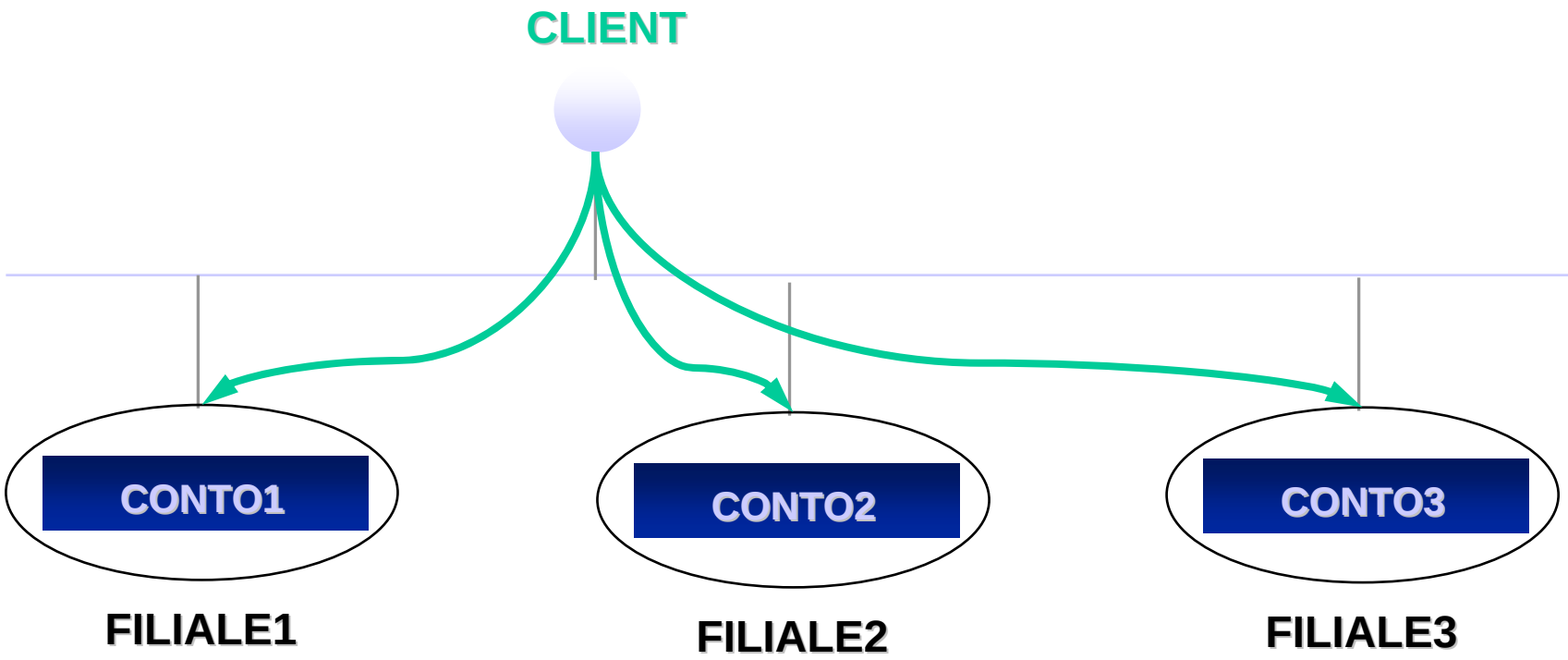
```
procedure Query (:num-cc, :saldo, :location);
case :location of
"Filiale1":        select   Saldo into :saldo
                   from    CONTO1
                   where  Num-cc = :num-cc;
"Centro":          select  Saldo into :saldo
                   from    CONTO2
                   where  Num-cc = :num-cc;
end procedure;
```

  - Usando il **parallelismo**: invece di sottomettere le due richieste in sequenza, esse possono essere processate in parallelo riducendo così il tempo globale di risposta.

Esecuzione seriale



Esecuzione parallela



4. OPERAZIONI DISTRIBUITE

Classificazione delle operazioni

- **Richieste remote** *:* transazioni read-only fatte di un numero qualsiasi di *query SQL* indirizzate ad un **singolo** DBMS remoto
  - Il DBMS remoto può essere solo interrogato
- **Transazioni remote:** fatte di un numero qualsiasi di *comandi SQL* (select, insert, delete, update) diretti ad un **singolo** DBMS remoto.
  - Ciascuna transazione legge/scrive da/su un solo DBMS
- **Transazioni distribuite:** fatte di un numero qualsiasi di *comandi SQL* (select, update, ...) diretti **ad un numero arbitrario di DBMS** remoti, ma con **ciascun comando SQL riferito ad un singolo DBMS**.
  - Le transazioni possono aggiornare più di un DBMS
    - Richiede la gestione delle transazioni distribuite (2PC)
- **Richieste Distribute** sono transazioni arbitrarie fatte da un numero qualsiasi di *comandi* SQL ciascuno diretto **ad un numero arbitrario di DBMS**, in cui **ciascun comando SQL può riferirsi a più di un DBMS**.
  - Le transazioni possono aggiornare più di un DBMS
    - Richiede 2PC
    - Richiede un ottimizzatore distribuito.

Ottimizzazione query distribuite

- Necessaria quando un DBMS riceve una **richiesta distribuita**; il DBMS interrogato è responsabile della 'ottimizzazione globale':
  - decide sulla divisione della query SQL in più sottoquery, ciascuna indirizzata ad uno specifico DBMS;
  - definisce un "piano" della esecuzione distribuita consistente nella esecuzione coordinata dei vari programmi sui vari DBMS e nello scambio di dati fra di essi.
- I fattori di costo di una query distribuita includono la quantità dei dati trasmessi sulla rete:

$$C_{total} = C_{I/O} \times n_{I/O} + C_{cpu} \times n_{cpu} + C_{tr} \times n_{tr}$$

n_{tr}: quantità di dati transmessi sulla rete

C_{tr}: costo unitario di trasmissione

Tipica transazione distribuita: trasferimento fondo

- Assumiamo:
  - CONTO(Num-Cli, Nome, Saldo) con i conti al di sotto di 10000 allocati sul frammento **CONTO1@1** e i conti al di sopra di 10000 allocati sul frammento **CONTO2@2**
- **Bisogna trasferire 500 € dal cliente con Num-Cli = 14878 al cliente con Num-Cli = 3154.**

Transazioni distribuite

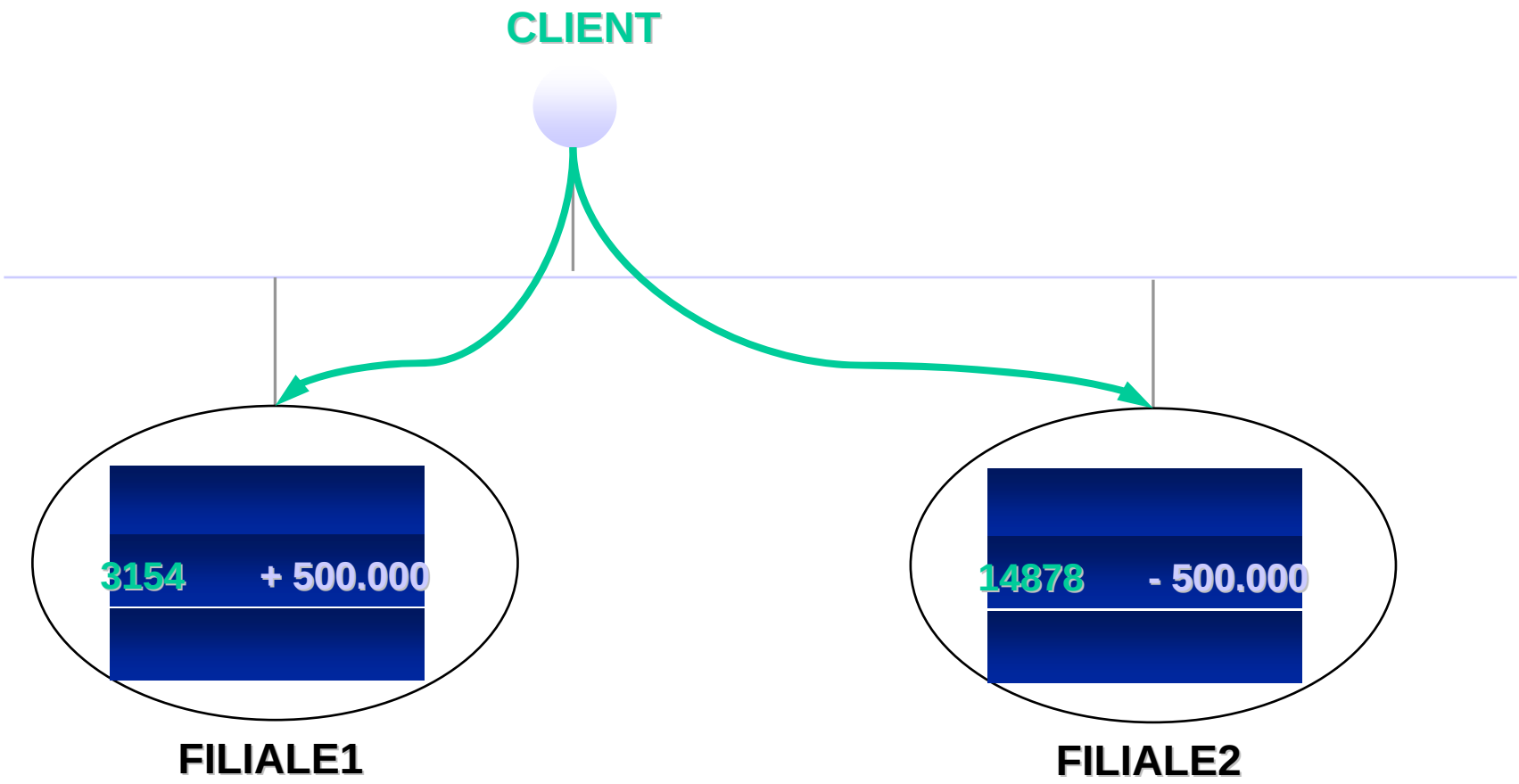
Codice:

```
BEGIN TRANSACTION
    UPDATE CONTO1@1
    SET SALDO=SALDO + 500
    WHERE NUM-CLI=3154;
    UPDATE CONTO2@2
    SET SALDO=SALDO - 500
    WHERE NUM-CLI=14878;
COMMIT-WORK
END TRANSACTION
```

Commento:

- è inaccettabile violazione della atomicità che una delle modificazioni sia eseguita e l'altra No.

Transazioni distribuite(2)



Proprietà acide dell'esecuzione distribuita

- **Isolamento**

  se ciascuna sottotransazione e' gestita da un "lock a due fasi stretto" la transazione e' globalmente serializzabile.

- **Persistenza**

  se ciascuna sottotransazione gestisce correttamente i log, i dati sono globalmente persistenti.

- **Consistenza**

  se ciascuna sottotransazione preserva l'integrità locale i dati sono globalmente consistenti.

- **Atomicità**

  e' il principale problema delle transazioni distribuite

Tecnologia dei database distribuiti

- La distribuzione dei dati non influenza consistenza e persistenza
  - la **consistenza** di transazioni non dipende dalla distribuzione dei dati, perché i vincoli di integrità descrivono solo proprietà locali

    (un limite della effettiva tecnologia dei DBMS)
  - la **persistenza** non dipende dalla distribuzione dei dati, perché ciascun sottosistema garantisce la persistenza locale usando meccanismi di recovery locali (log, checkpoint e dump)
- Le altre caratteristiche e sottosistemi richiedono miglioramenti:
  - il controllo di concorrenza (per l' **isolamento**)
  - il controllo dell'affidabilità (per l' **atomicità**)

Controllo della concorrenza

- In un sistema distribuito, una transazione t_i può eseguire varie sotto-transazioni t_{iF} , dove il secondo indice denota il nome del nodo del sistema sul quale la sotto-transazione lavora.

  $t_1 : r_{1A}(x)\ w_{1A}(x)\ r_{1B}(y)\ w_{1B}(y)$
  $t_2 : r_{2B}(y)\ w_{2B}(y)\ r_{2A}(x)\ w_{2A}(x)$

- **La serializzabilità locale all'interno degli schedule non è sufficiente per garantire la serializzabilità.**
- Consideriamo i due schedule ai nodi A e B:

  $SA : r_{1A}(x)\ w_{1A}(x)\ r_{2A}(x)\ w_{2A}(x)$
  $SB : r_{2B}(y)\ w_{2B}(y)\ r_{1B}(y)\ w_{1B}(y)$

- Questi sono localmente conflict-serializzabili, non globalmente:
  - sul nodo A, t_1 precede t_2 and è in conflitto con t_2
  - sul nodo B, t_2 precede t_1 and è in conflitto con t_1

Serializzabilità globale

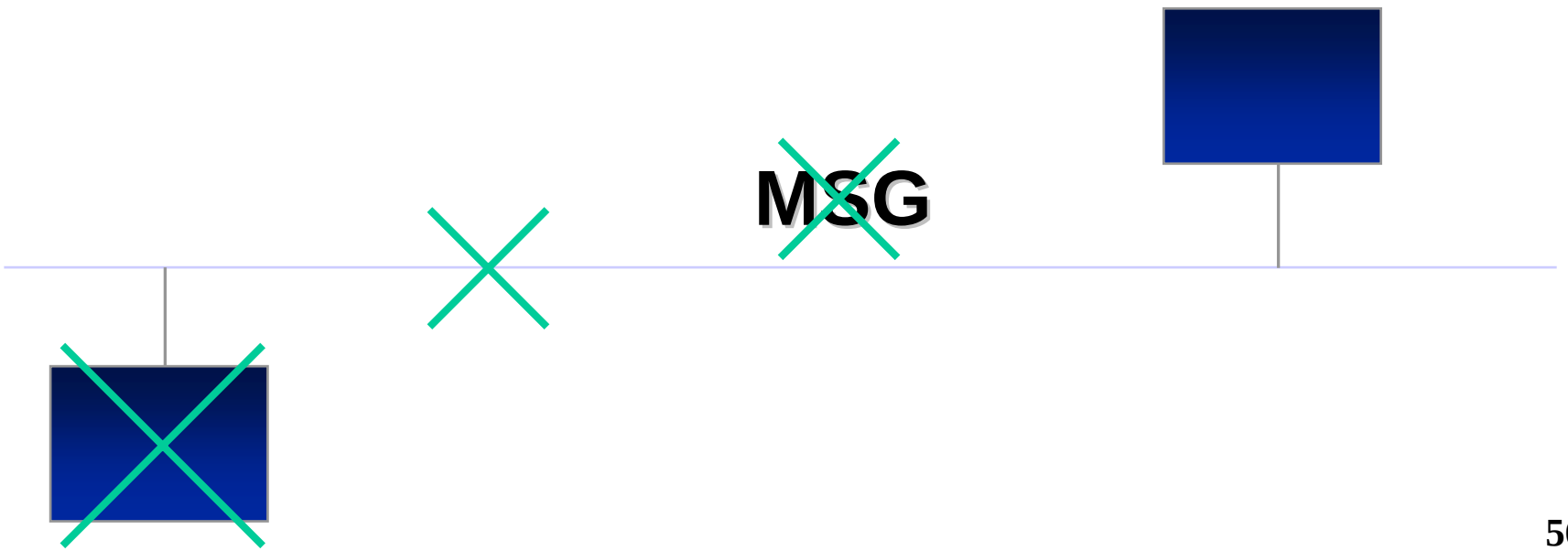
- La serializzabilità globale delle transazioni distribuite sui nodi del database distributo richiede l'esistenza di un unico **schedule seriale** *S* equivalente a tutti gli schedule locali S_i prodotti a ciascun nodo.
- Sono valide le seguenti proprietà:
  - se ciascuno scheduler del database distribuito usa il metodo di **locking a due fasi stretto** su ciascun nodo (e così esegue il commit quando tutte le sotto-transazioni hanno acquisito tutte le risorse), allora gli schedule risultanti sono globalmente conflict serializzabili.
  - se ciascuna transazione distribuita acquisisce un singolo timestamp e lo usa in tutte le richieste a tutti gli scheduler che usano il controllo della concorrenza basata sul timestamp, gli schedule risultanti sono globalmente seriali, basati sull' ordine imposto dai timestamp.

Deadlock distribuiti

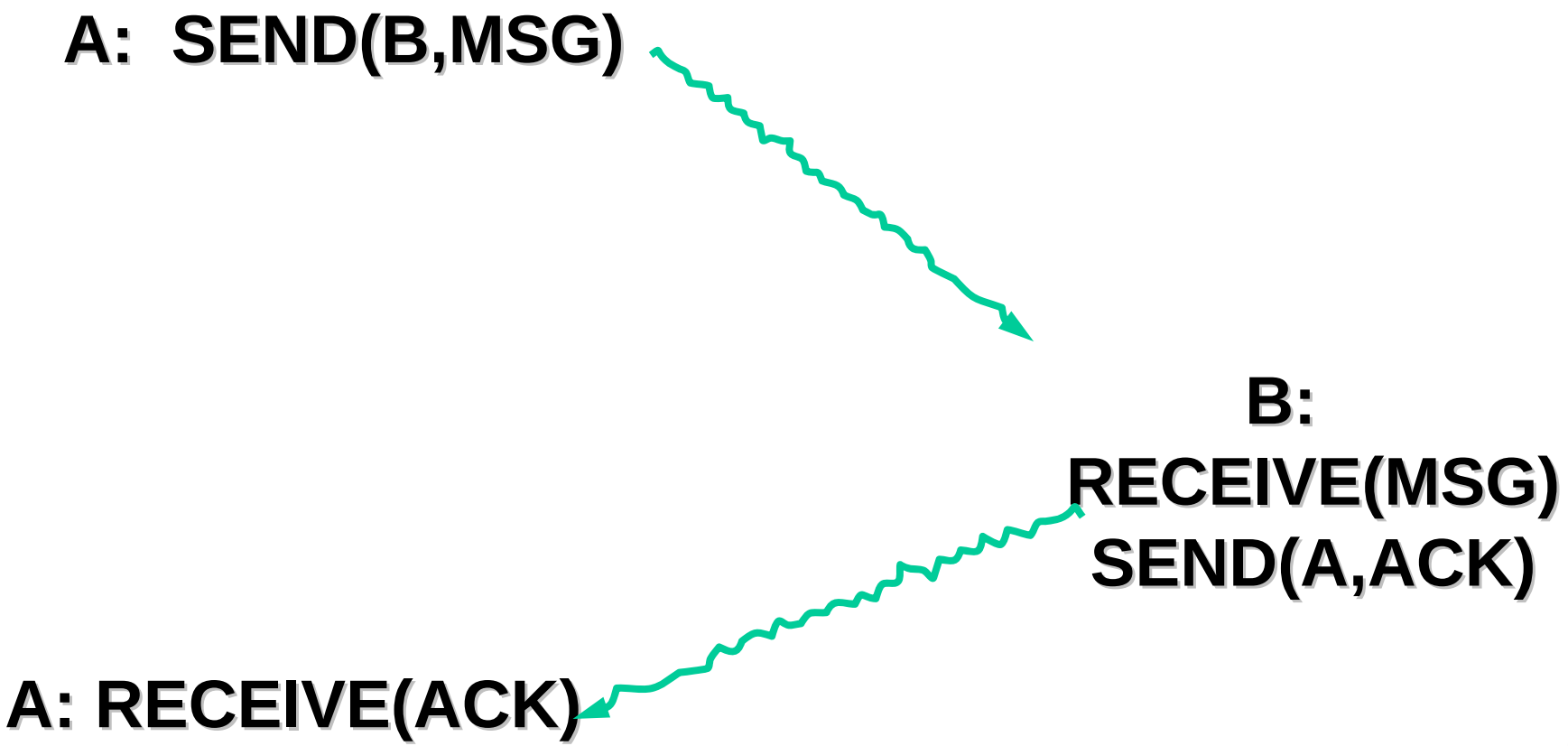
- I deadlock distribuiti possono essere dovuti a situazioni di attesa circolare fra due o più nodi
- Il metodo di time-out è valido ed è molto usato dai DBMS distribuiti.
- La risoluzione del deadlock può essere fatta con un protocollo asincrono e distribuito (implementato in a versione distribuita del DB2 dalla IBM)

Guasti in un sistema distribuito

- **Caduta di nodi (Soft o Hard)**
- **Perdita di messaggi (… messaggi di ack)**
- **Interruzione della rete (…partizionamento)**



Protocolli distribuiti e perdita di messaggi



Commit a due fasi

Protocollo per garantire l'atomicita' di sotto-transazioni distribuite

Protagonisti

- un coordinatore C (Transaction Manager, TM)
- molti partecipanti P_i (Resource Manager, RM)

Come un matrimonio

- fase uno: dichiarazione di intenti
- fase due: dichiarazione del matrimonio

Nuovi record nel log del coordinatore (C)

- Records of TM (C)
  - Il record **prepare** contiene l'identità di tutti i processi RM (cioè, gli identificatori di nodi e processi)
  - Il record **global commit (global abort)** descrive la decisione globale. Quando il TM (C) scrive nel suo log il record global commit (global abort), esso raggiunge la decisione finale.
  - Il record **complete** è scritto alla fine del protocollo di commit a due fasi.

Nuovi record del log del partecipante

- Records of RM (P_i)
  - Il record **ready** indica la irrevocabile disponibilità a partecipare al protocollo di commit a due fasi, pertanto contribuendo a una decisione di commit. Può essere scritto solo quando RM (P_i) è “in stato affidabile”, cioè possiede i lock su tutte le risorse che devono essere scritte. Su questo record è anche scritto l’identificatore (di processo e di nodo) del TM.
  - In aggiunta, i record **begin**, **insert**, **delete**, and **update** sono scritti come nei server centralizzati
- In ogni momento (prima del **ready**) un RM (P_i) può autonomamente abortire una sotto-transazione, disfacendone gli effetti, senza partecipare al protocollo di commit a due fasi.

Record nel log del coordinatore

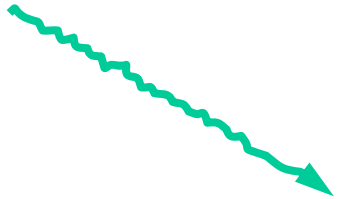
a. **PREPARE: identità dei partecipanti**
b. **GLOBAL COMMIT/ABORT: decisione**
c. **COMPLETE: termine del protocollo**

Record nel log del partecipante

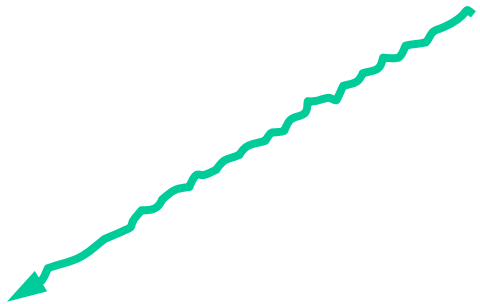
a. **READY: disponibilità al commit**
b. **LOCAL COMMIT/ABORT: decisione ricevuta**

Fase 1

C: WRITE-LOG(PREPARE)
SET TIME-OUT
SEND (Pi,PREPARE)

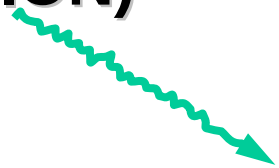


Pi: RECEIVE(C,PREPARE)
IF OK THEN WRITE-LOG(READY)
MSG=READY
ELSE MSG=NO
SEND (C,MSG)

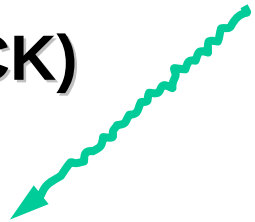


Fase 2

C: RECEIVE(P_i,MSG)
IF TIME-OUT OR ONE(MSG)=NO
THEN WRITE-LOG(GLOBAL-ABORT)
DECISION=ABORT
ELSE WRITE-LOG(GLOBAL-COMMIT)
DECISION=COMMIT
SEND(Pi,DECISION)

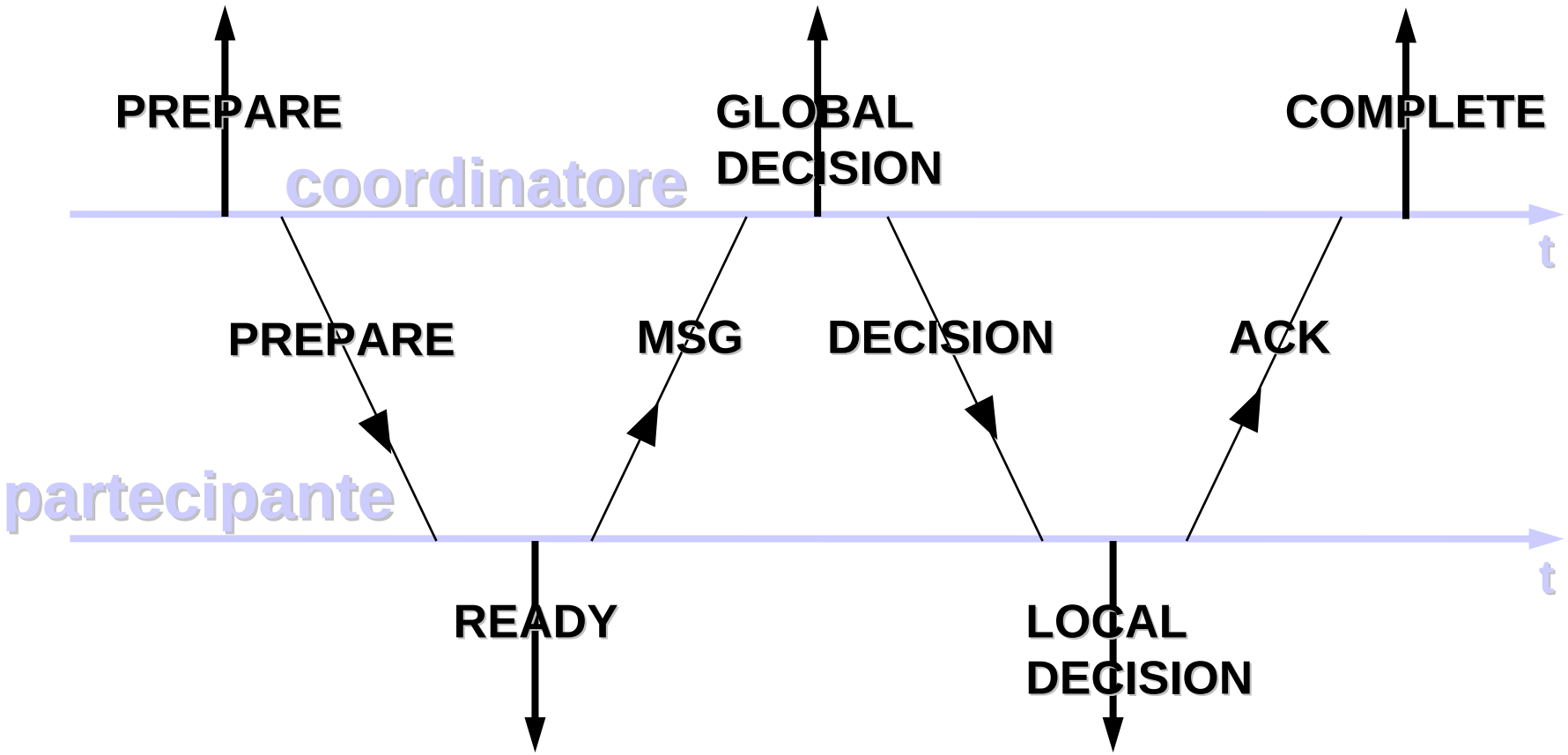


Pi:RECEIVE(C,DECISION)
IF DECISION=COMMIT THEN WRITE-LOG(COMMIT)
ELSE WRITE-LOG(ABORT)
SEND (C,ACK)

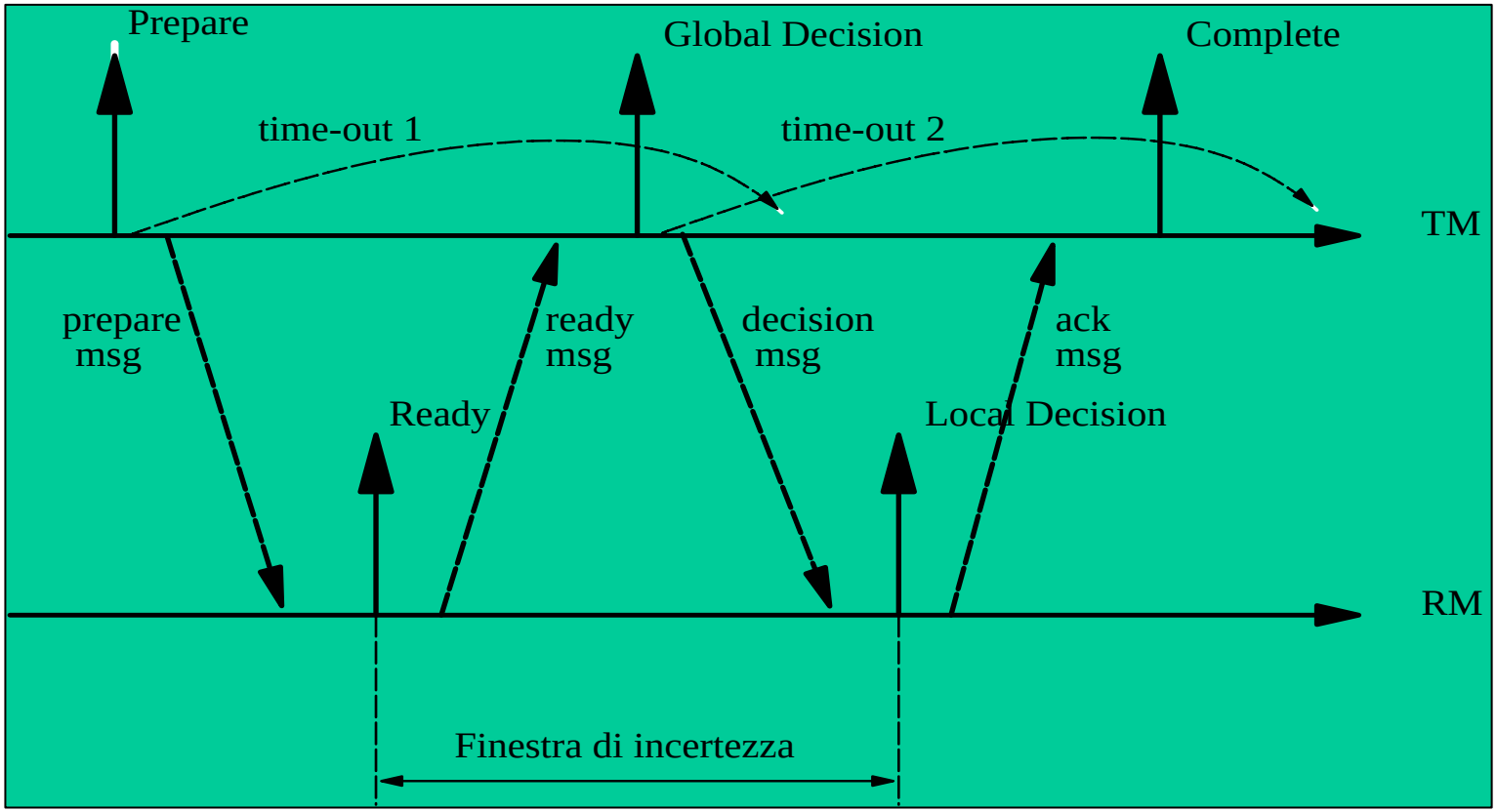


C: RECEIVE(Pi,ACK)
WRITE-LOG(COMPLETE)

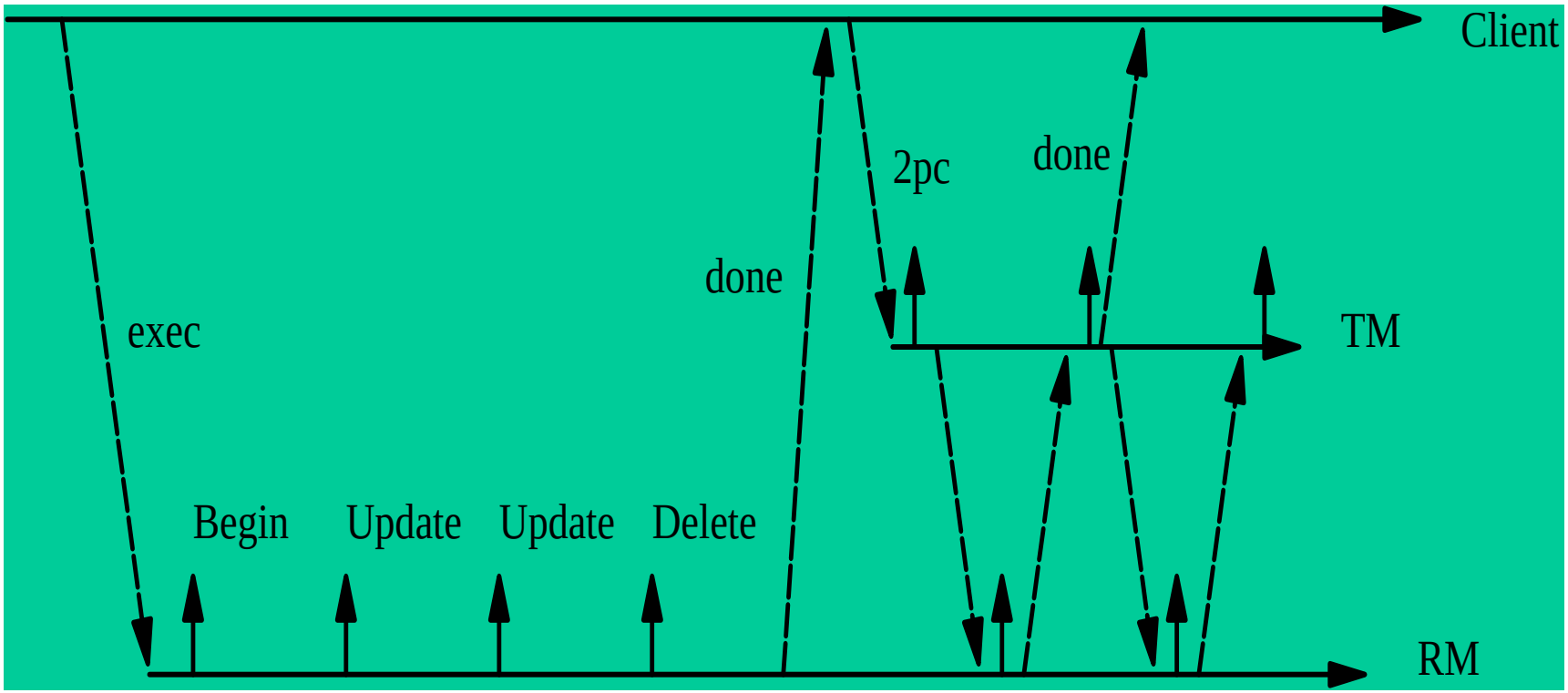
Diagramma del commit a due fasi



Protocollo con time-out e finestra di incertezza



Protocollo nel contesto di una transazione completa



Blocking, incertezza, protocolli di recovery

- **Un RM dopo essersi dichiarato “ready” perde la sua autonomia e attende la decisione del TM. Un guasto del TM lascia l’ RM in uno stato di incertezza, in cui tutte le risorse acquisite con lock sono bloccate.**
- **L’intervallo tra la scrittura dei record `ready` e `commit` o `abort` è chiamato *finestra di incertezza*. Il protocollo è progettato per minimizzare la sua durata.**
- **I protocolli di recovery sono svolti dai TM o RM dopo i guasti; ristabiliscono uno stato finale corretto che dipende dalla decisione globale del TM.**

Complessità del protocollo

Deve poter gestire tutti i guasti:

- **caduta di uno o più partecipanti**
- **caduta del coordinatore**
- **perdite di messaggi**

Recovery dei partecipanti

- **Viene svolto durante la loro ripresa a caldo: per ogni transazione dipende dall'ultimo record nel log:**
  - **Se è una azione generica oppure un "`abort`" tutte le azioni sono disfatte; se è un "`commit`", le azioni vengono rifatte; la recovery non dipende dal protocollo.**
  - **Se è un "`ready`", il guasto è accaduto durante il commit a due fasi; il partecipante è in dubbio circa il suo esito.**

- **Durante il protocollo di ripresa a caldo, si collezionano tutte le transazioni in dubbio; di esse si chiede l'esito ai rispettivi TM (richiesta di remote recovery).**

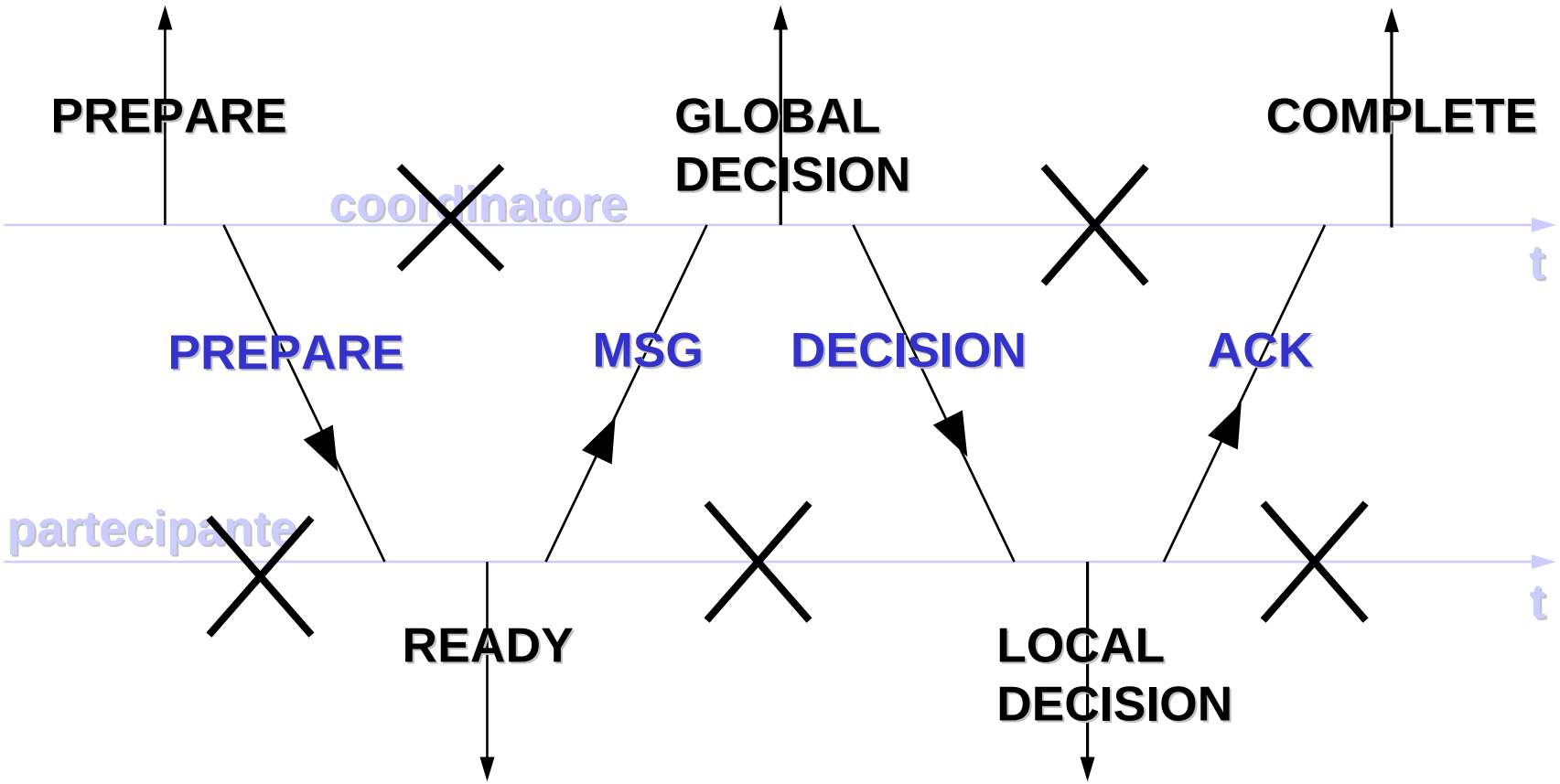
Recovery del coordinatore

- Quando l'ultimo record nel log è un "*prepare*", il guasto del TM **può essere la causa di un blocco di qualche RM**. Il TM ha due opzioni:
  - **scrivere "`global abort`" sul log**, e **ripetere la seconda fase** del protocollo di commit a due fasi
  - **ripetere la prima fase**, provando a raggiungere un commit globale
- Quando l'ultimo record è la una "*global decision*", **alcuni RMs potrebbero essere bloccati**. Il TM deve
  - **ripetere la seconda fase** del protocollo.

Perdita di messaggi partizione di rete

- La perdita dei **messaggi** di “prepare” o “ready” sono indistinguibili; in entrambi i casi **scatta il time-out** sul TM e *viene deciso un* “**global abort**”.
- La perdita dei **messaggi** “*decision*” or “*ack*” sono pure indistinguibili. in entrambi i casi **scatta il time-out** sul TM e viene **ripetuta la seconda fase**.
- Un *partizionamento della rete* non causa problemi: si perviene a “*global commit*” solo se RM e TM appartengono alla stessa partizione.

Recovery del protocollo 2PC



Protocollo di “abort presunto”

- E’ una ottimizzazione, usata da tutti i sistemi commerciali:
  - Se un TM riceve una richiesta di “**remote recovery**” da una transazione in dubbio, risponde per default che la transazione ha fatto un “**global abort**”.
  - **Come conseguenza, se vengono persi “prepare” e “global abort” si ottiene comunque un corportamento corretto => non è necessario scriverli in modo sincrono sul log.**
- Inoltre, il record “`complete`” non può essere omesso.
- In conclusione, gli unici record che devono essere scritti in ***modo sincrono*** (con una **force**) sono: `ready`, `global commit` e `local commit`.

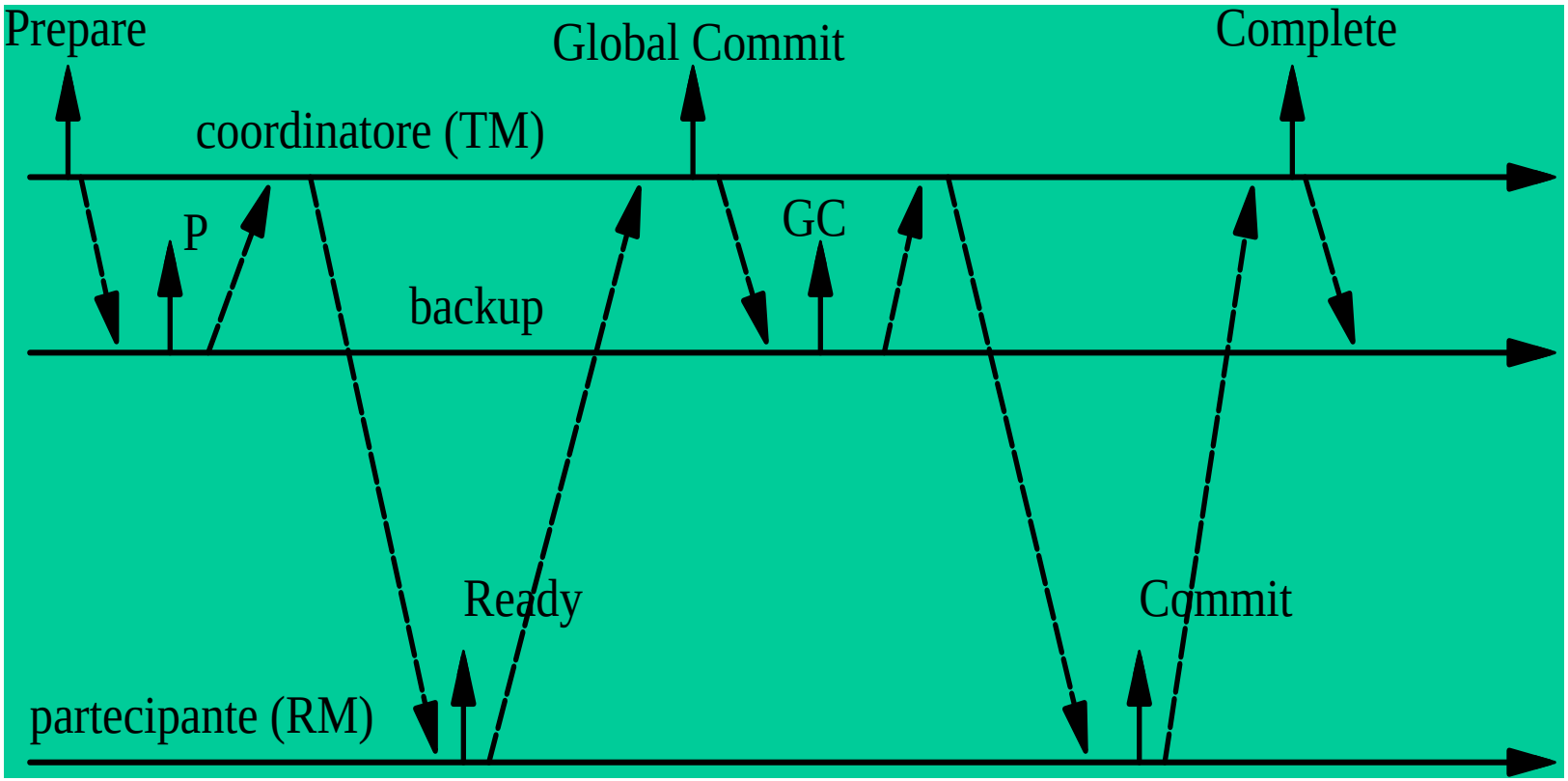
Ottimizzazione “read-only”

- Quando un RM ha svolto solo operazioni di lettura,
  - Risponde `read-only` al messaggio di `prepare` e esce dal protocollo.
  - Il TM ignora tutti gli RM *“read-only” nella seconda fase del protocollo*.

Protocollo di commit a quattro fasi

- Il processo TM viene replicato tramite un processo di backup collocato su un nodo differente.
  - Il TM informa il backup delle sue decisioni prima di comunicarle agli RM.
- Se il TM ha un guasto, il backup può intervenire:
  - Come prima cosa attiva un altro backup.
  - Successivamente continua la esecuzione del protocollo di commit.

Diagramma del protocollo di commit a quattro fasi



III-APPLICAZIONI COOPERATIVE

Cooperazione tra sistemi preesistenti

- La *Cooperazione* è la capacità delle applicazioni di un sistema **di fare uso dei servizi applicativi resi disponibili da altri sistemi**, eventualmente gestiti da differenti organizzazioni.
- La necessità della cooperazione è dovuta a differenti ragioni che vanno dalla semplice domanda di integrazione di componenti sviluppati separatamente all'interno della *stessa* organizzazione, alla cooperazione o fusione di *differenti* organizzazioni.
- La cooperazione tra basi di dati è molto difficile: il modello "ideale" **altamente integrato,** interrogabile in maniera **trasparente** ed **efficiente** è in pratica impossibile da sviluppare e gestire.

Iniziative di cooperazione (1)

- Le iniziative di cooperazione possono essere occasione per *razionalizzare i sistemi oggetto di cooperazione*, modificandone di conseguenza eterogeneità, distribuzione, autonomia. Ciò richiede in generale una *valutazione costi benefici* e una *rimozione di vincoli normativi*.
- Esempio:
  - da 30 anni esistono in Italia due basi delle automobili e dei proprietari di automobili, gestite da Ministero dei Trasporti e ACI, tra loro non coerenti.
  - Recentemente è stata fatta (c'è voluta) una legge per unificarle.

Iniziative di cooperazione (2)

- Esempio 2:

– Nel sistema INPS, INAIL, Camere di Commercio, si è colta l'occasione della creazione del *record indici*, per effettuare "*record matching*" tra le tre basi di dati **riconciliando** record che rappresentano la *stessa* impresa, per migliorare la qualità dei dati delle tre basi.

Cooperazione basata sui dati

- Due tipi di co-operazione
  - *cooperazione process-centered* : i sistemi si scambiano servizi, scambiando messaggi, informazioni o documenti, o facendo scattare attività, senza rendere visibili esplicitamente i dati remoti.
  - *cooperazione data-centered*, in cui i dati sono naturalmente *distribuiti, eterogenei e autonomi*, e accessibili da locazioni remote in accordo ad un fissato patto di cooperazione.
- Ci concentreremo sulla cooperazione centrata sui dati, caratterizzata da distribuzione, eterogeneità e autonomia dei dati.

Cooperazione basata sui dati: caratteristiche

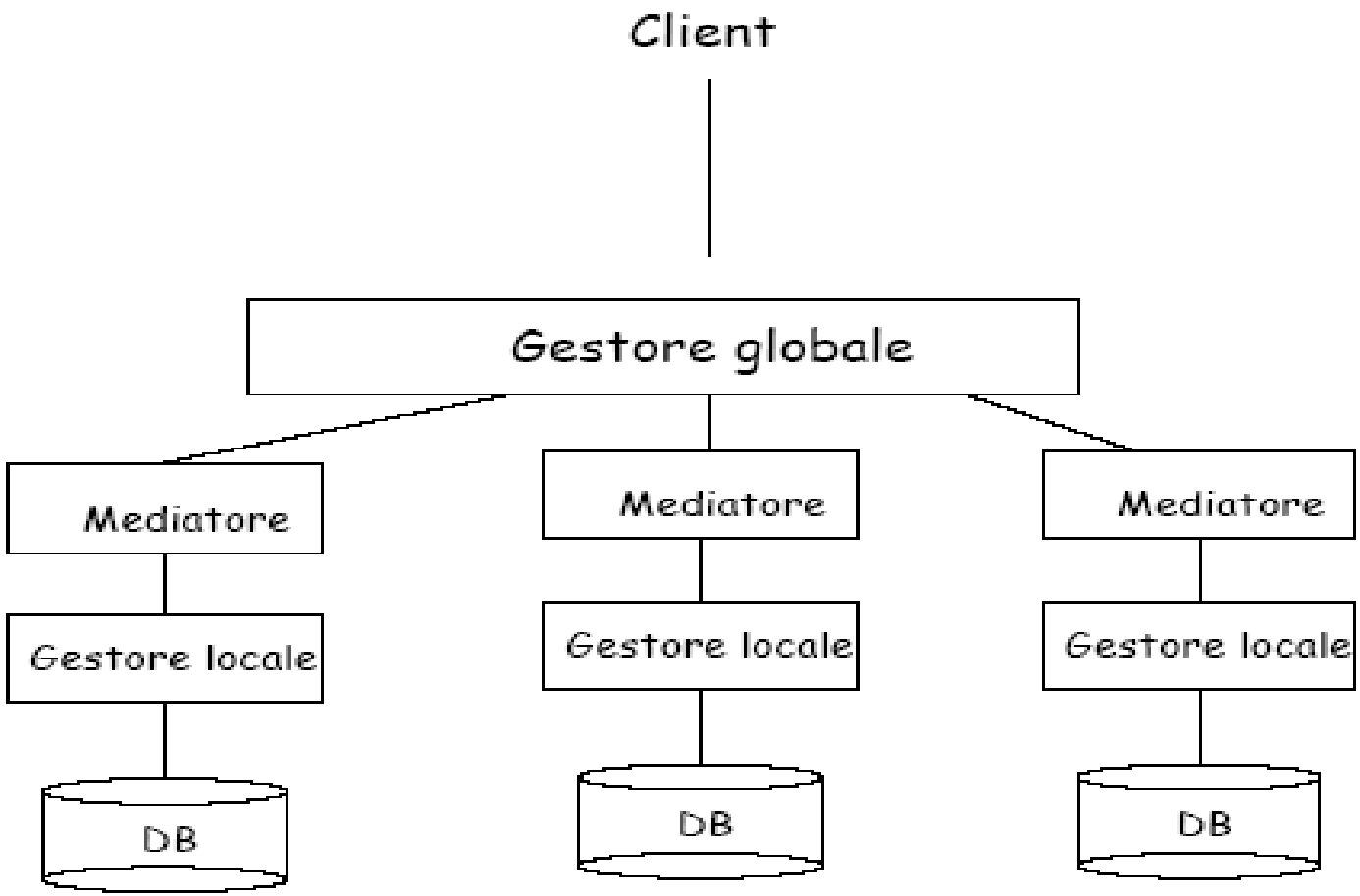
- Le tre sotto elencate caratteristiche:
  - Il **livello di trasparenza** misura come sono mascherate ***distribuzione*** **ed** ***eterogeneità*** dei dati
  - La **integrazione delle operazioni distribuite** misura il grado di coordinamento necessario a svolgere operazioni sui database **autonomi** e *cooperanti*.
  - Il **livello di attualità dei dati** indica se i dati a cui accedere sono aggiornati o non.

  identificano tre tipi di architetture per la co-operazione basata sui dati.

Multidatabase

- Ciascuno dei database partecipanti continua ad essere utilizzato dai rispettivi utenti (programmi o utenti finali)
- I sistemi sono anche acceduti da moduli, chiamati ***mediatori***, che mostrano solo la "**vista**" di database che deve essere esportata. I **mediatori rendono disponibile tale "vista" dei dati al gestore globale che realizza l'integrazione**
- In generale i dati non vengono modificati dai mediatori , poiché ciascuna sorgente è autonoma.
- Caratteristiche:
  - Il livello di **trasparenza** è elevato.
  - La **integrazione** può essere elevata anche se spesso è preferita la gestione locale delle modifiche.
  - Il livello di **attualità** è anche alto, in quanto si accede direttamente ai dati dei sistemi componenti.

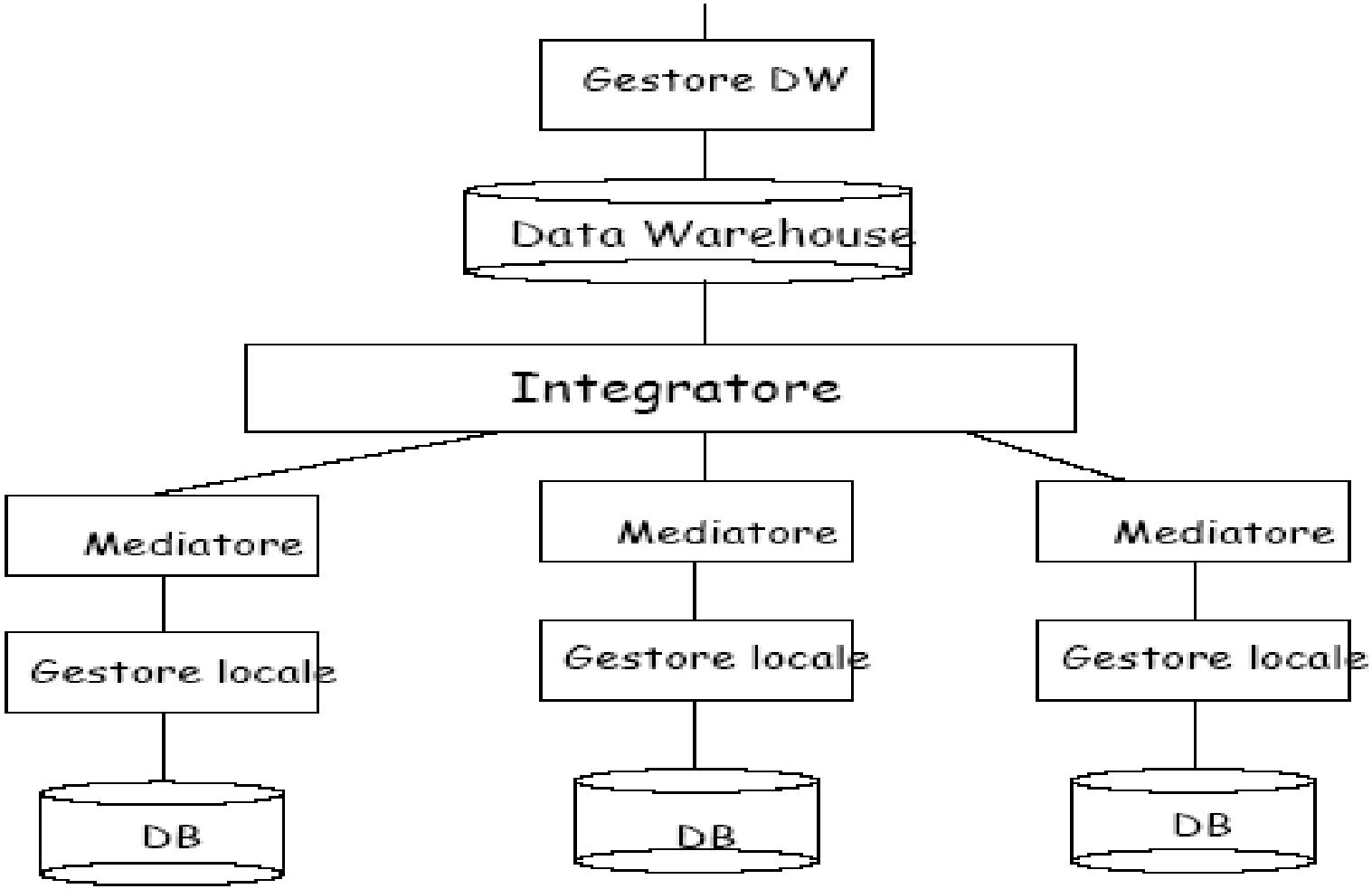
Multidatabase



Sistemi basati su dati replicati

- Garantiscono l'accesso read-only a copie secondarie di informazioni provenienti dall'esterno.
- Queste possono essere immagazzinate in un **data warehouse**, che contiene i **dati estratti** da vari sistemi eterogenei distribuiti e offre una **vista globale** dei dati.
- Caratteristiche:
  - alto livello di **trasparenza**
  - alto livello di **integrazione**,
  - un livello ridotto di **attualità**.

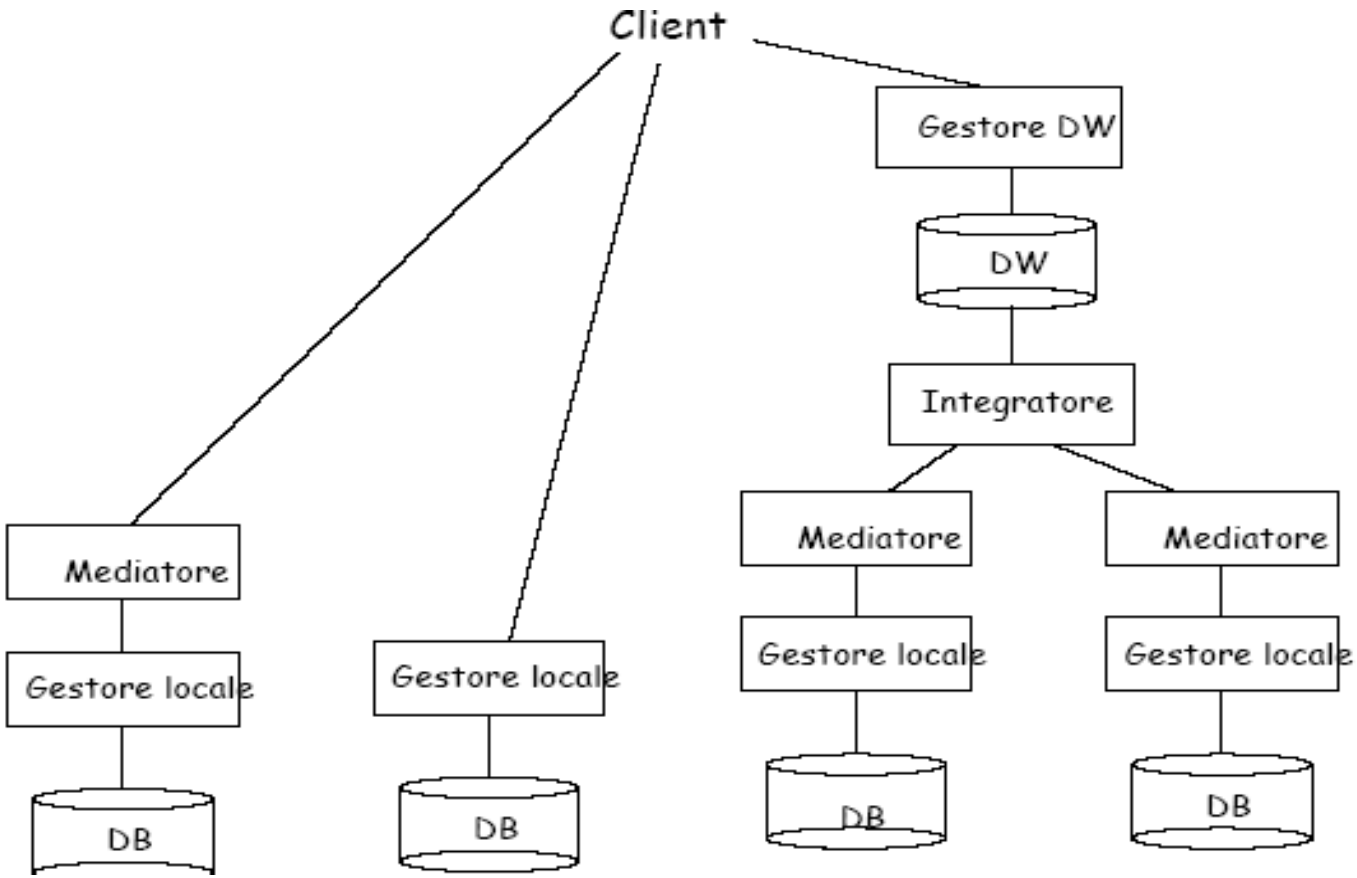
Sistemi basati su dati replicati



Sistemi basati sull'accesso a dati esterni

- **L'integrazione dei dati è realizzata esplicitamente dalla applicazione:**
  - nell'esempio seguente, sono integrate tre sorgenti : un database esterno, un database locale e una data warehouse, che a sua volta usa due sorgenti informative.
- Caratteristiche:
  - basso livello di **trasparenza**
  - basso livello di **integrazione**,
  - livello di **attualità** che dipende dalle specifiche esigenze.

Sistemi basati sull'accesso a dati esterni



Interoperabilità

- L’interoperabilità è il principale problema 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 per mezzo di protocolli standard quali FTP, SMTP/MIME, e così via.
- Con riferimento ai database , l’interoperabilità è garantita dall’adozione di opportuni standard.

Open Database Connectivity (ODBC)

- E’ una interfaccia applicativa proposta da Microsoft nel 1991 per la costruzione di applicazioni database eterogenee, supportate da molti prodotti relazionali.
- Il linguaggio supportato da ODBC è un SQL ristretto, caratterizzato da un minimo set di istruzioni.
- Le applicazioni interagiscono con i server DBMS per mezzo di un driver, cioè una libreria che è dinamicamente connessa alle applicazioni. Il driver maschera le differenze di interazione dovuta al DBMS, al sistema operativo ed al protocollo di rete.
  - Per esempio, la tripla (Oracle, Windows/NT, Novell) identifica uno specifico driver.
- ODBC non supporta il protocollo 2PC.

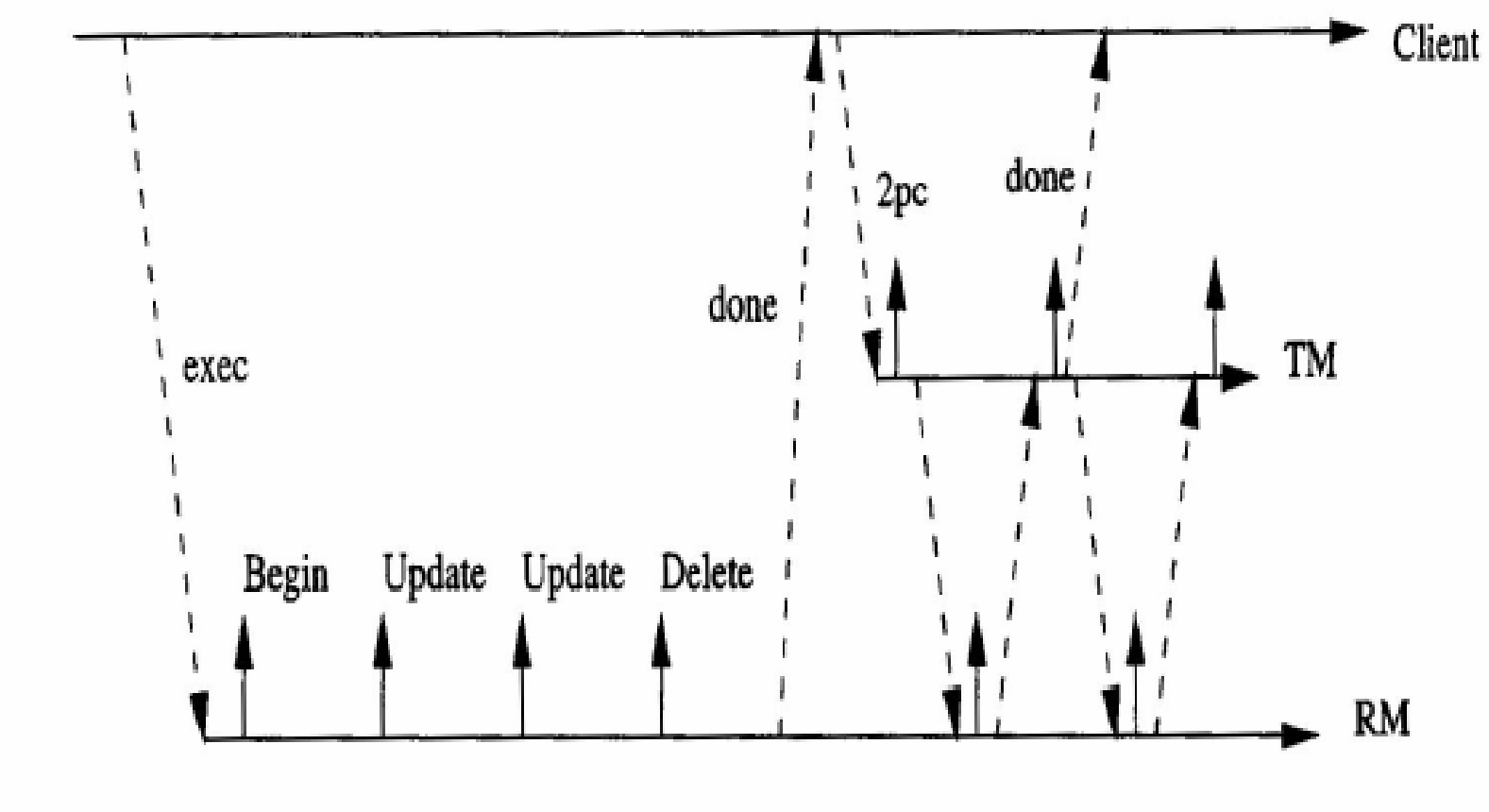
Componenti ODBC

- La *applicazione* richiama le query SQL
- The *gestore del driver* carica i driver a richiesta dell' applicazione e prevede le funzioni di conversione dei nomi. Questo software è fornito da Microsoft
- I *driver* effettuano le funzioni ODBC. Essi eseguono le query SQL traducendole e, se possibile, adattandole alla sintassi e semantica dei prodotti specifici.
- La *sorgente dati* è il sistema remoto DBMS, che esegue le funzioni trasmesse dal client.

X-Open Distributed Transaction Processing (DTP)

- Un protocollo che garantisce l'interoperabilità delle **computazioni transazionali su DBMS di differenti fornitori**.
- Ipotizza la presenza di un client, molti RM e un TM
- Il protocollo consiste di due interfacce:
  - fra client e TM, chiamata *TM-interface*
  - fra TM e ciascun RM, chiamata *XA-interface*
- I DBMS relazionali **devono** prevedere l' interfaccia XA
- Vari prodotti specializzati in gestione transazioni, come *Encina* (un prodotto di "Transarc company") e *Tuxedo* (da Unix Systems) forniscono il componente TM.

Interazioni tra client, RM, TM per il protocollo X-Open



Caratteristiche di X-Open DTP

- I RM sono passivi; essi rispondono alle chiamate a procedura remota effettuate dal TM.
- Il protocollo usa il two-phase commit con "*abort presunto*" e "ottimizzazione *read-only*".
- Il protocollo supporta *decisioni euristiche*, che in presenza di guasti consentono l'evoluzione di una transazione sotto il controllo dell'operatore:
  - quando un RM è bloccato a causa di un guasto del TM, un operatore può imporre una decisione euristica (generalmente abort), consentendo il rilascio delle risorse;
  - quando decisioni euristiche causano una perdita di atomicità, il protocollo garantisce la notifica ai processi client;
  - quando decisioni euristiche erronee causano inconsistenze, la risoluzione delle medesime è affidata all'applicazione.

Interfaccia TM

- `tm_init` and `tm_exit` iniziano e terminano il dialogo client-TM
- `tm_open` and `tm_term` aprono and chiudono la sessione con il TM
- `tm_begin` inizia una transazione
- `tm_commit` richiede un “global commit”.

Interfaccia XA

- `xa_open` and `xa_close` aprono e chiudono una sessione tra TM e un dato RM
- `xa_start` and `xa_end` attivano e completano una transazione
- `xa_precom` richiede che RM completi the prima fase del protocollo di commit; il processo RM può rispondere positivamente al call solo se è in uno stato “recoverable”.
- `xa_commit` and `xa_abort` comunicano la decisione globale circa la transazione.
- `xa_recover` inizia una di procedura recovery dopo il guasto di un processo (TM o RM); RM consulta il suo log and costruisce tre insiemi di transazioni :
  - transazioni *in dubbio*
  - transazioni decise da un *commit euristico*
  - transazioni decise da un *abort euristico*
- `xa_forget` consente ad un RM di dimenticare transazioni decise in maniera euristica.

IV - BASI DI DATI REPLICATE

Replicazione dei dati

E' un ingrediente fondamentale dei sistemi informativi

motivazioni:

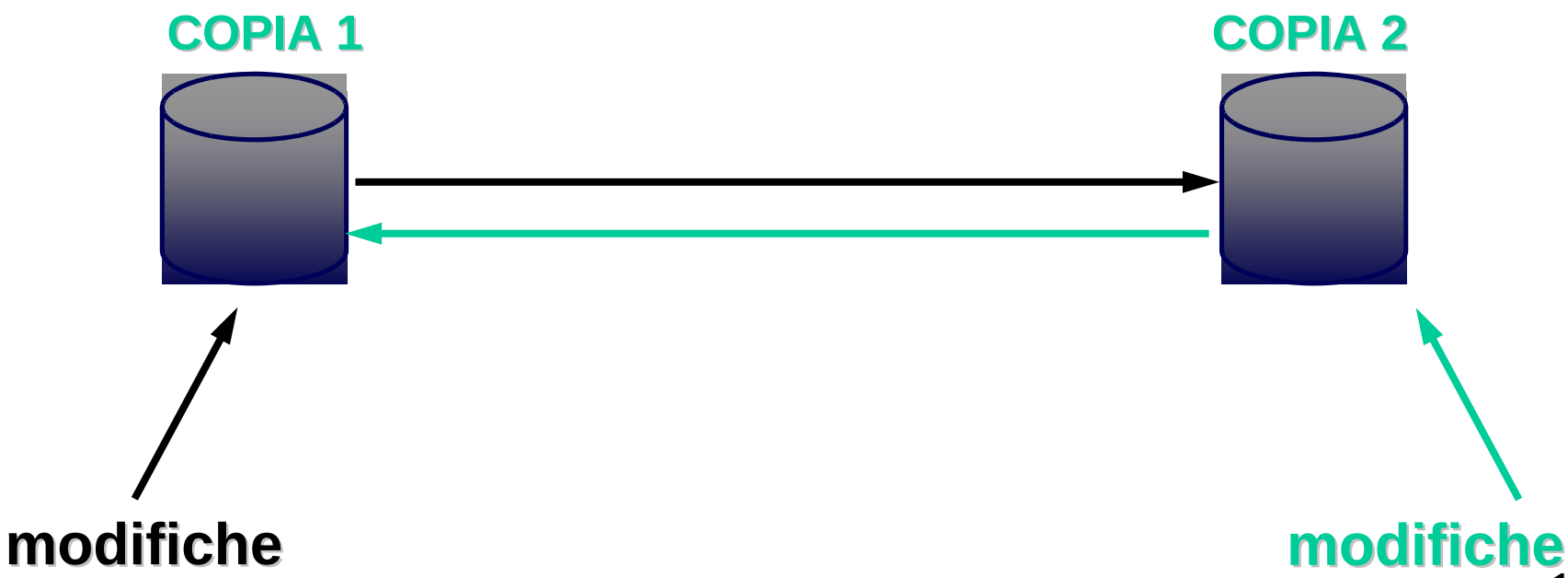
- **efficienza**
- **affidabilità**
- **autonomia**

Modalità di replicazione

- **Asimmetrica**

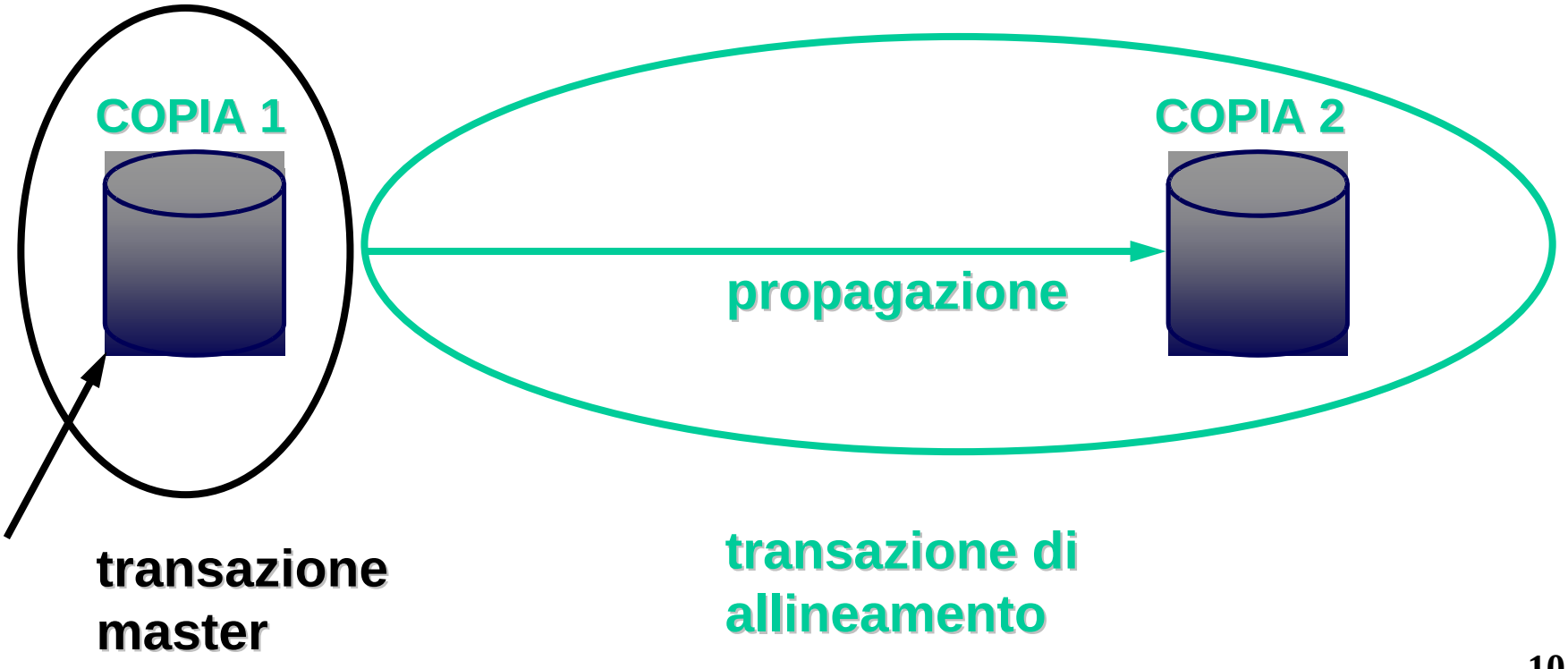
Modalità di replicazione

- **Simmetrica**
  (Problema della concorrenza)



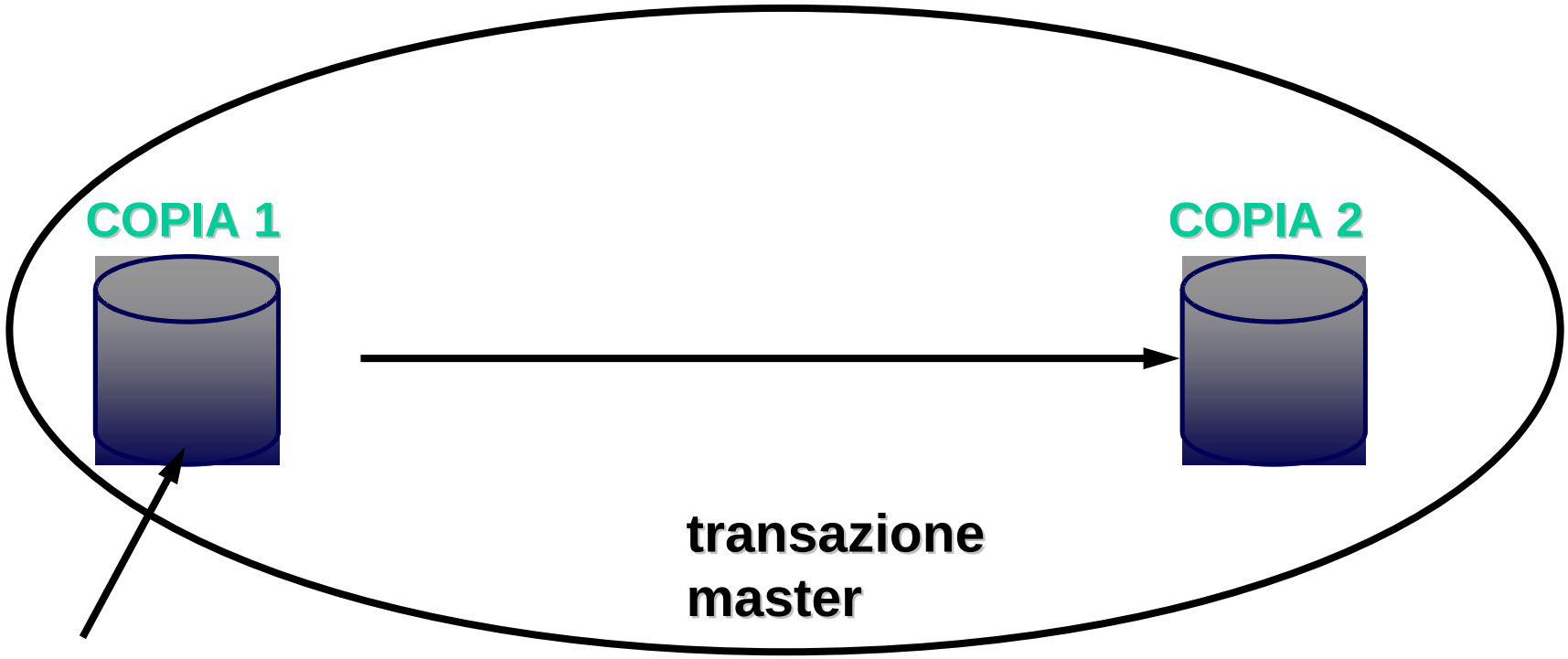
Modalità di trasmissione delle variazioni

- **Trasmissione asincrona**



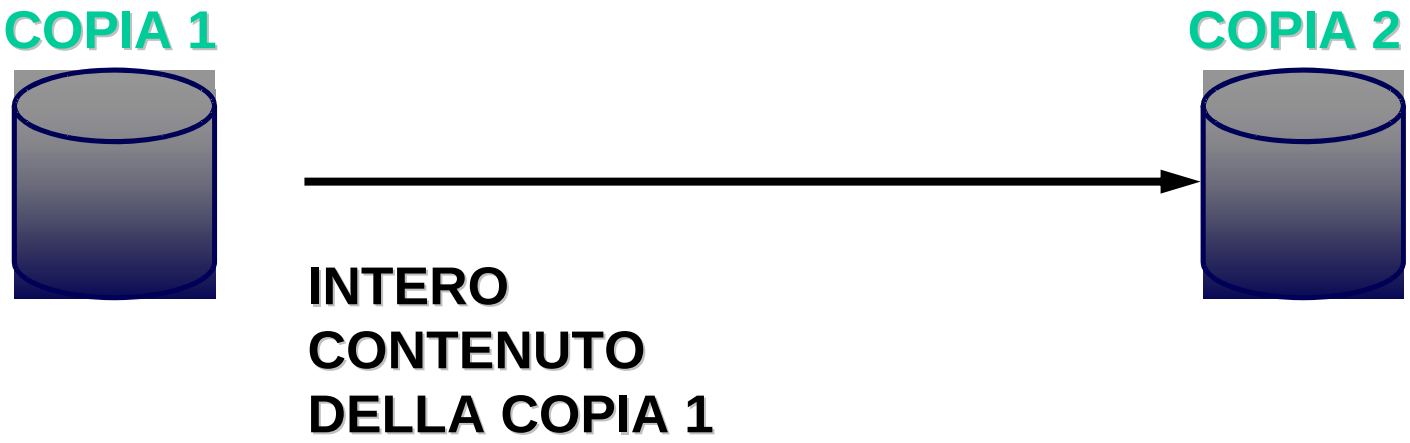
Modalità di trasmissione delle variazioni

- **Trasmissione sincrona**
  (problema del 2PC)



Modalità di allineamento

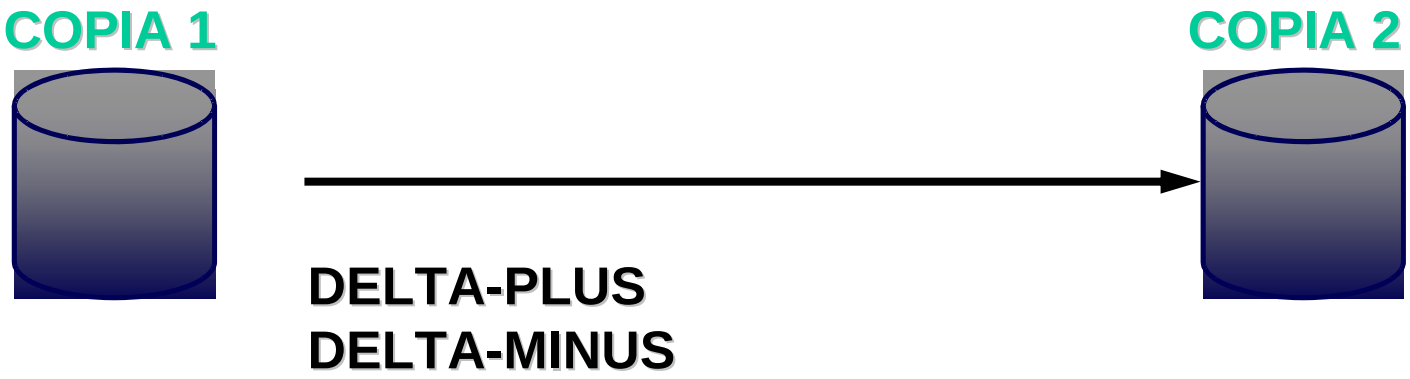
- **Refresh**



- **Allineamento:**
  - **periodico**
  - **a comando**
  - **ad accumulo di variazione**

Modalità di allineamento

- **Incrementale**



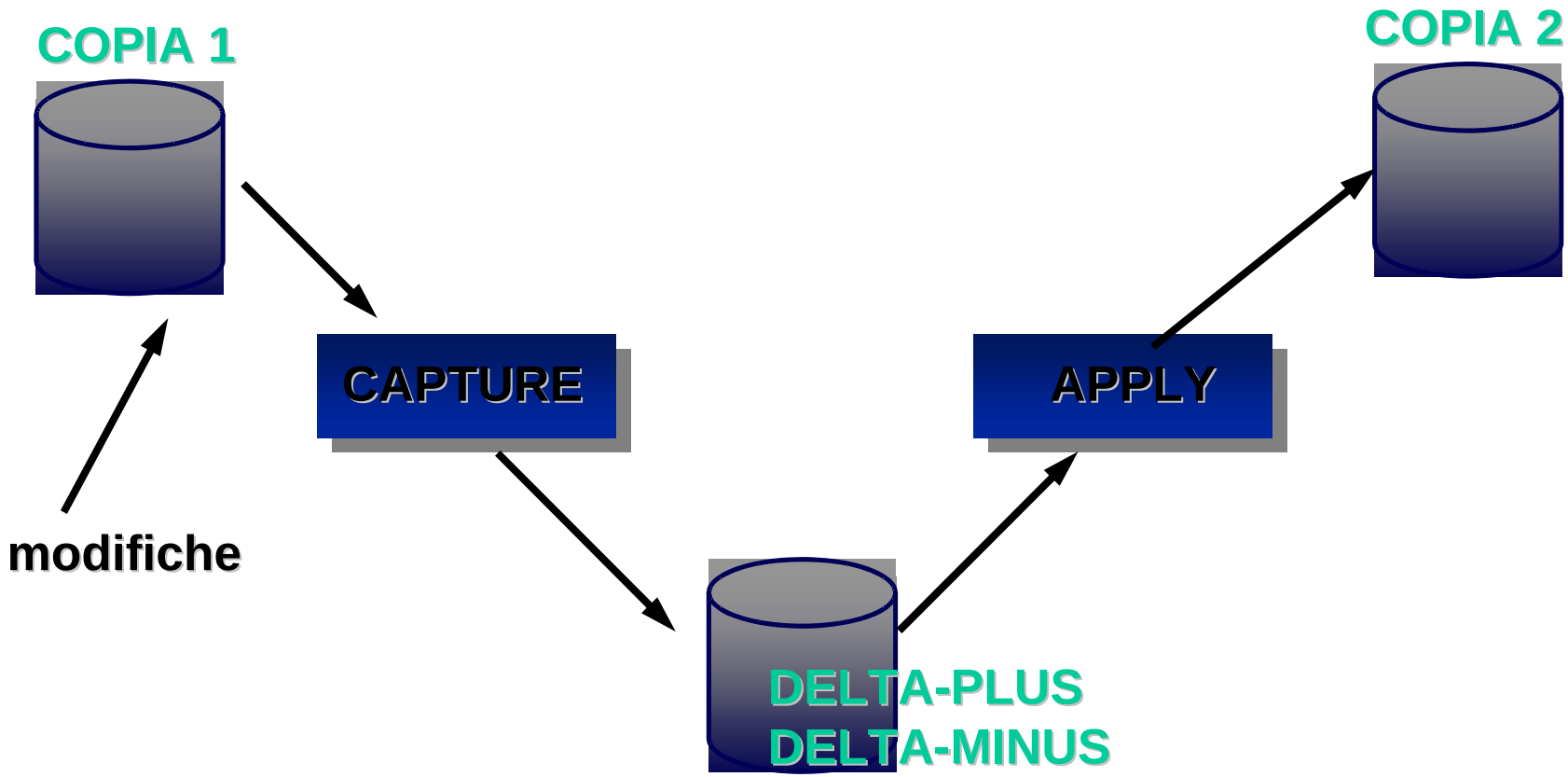
- **Allineamento:**
  - **periodico**
  - **a comando**
  - **ad accumulo di variazione**

Meccanismi per la replicazione

asimmetrica, asincrona e incrementale

Prodotto: Replication Manager

- con due moduli : CAPTURE, APPLY

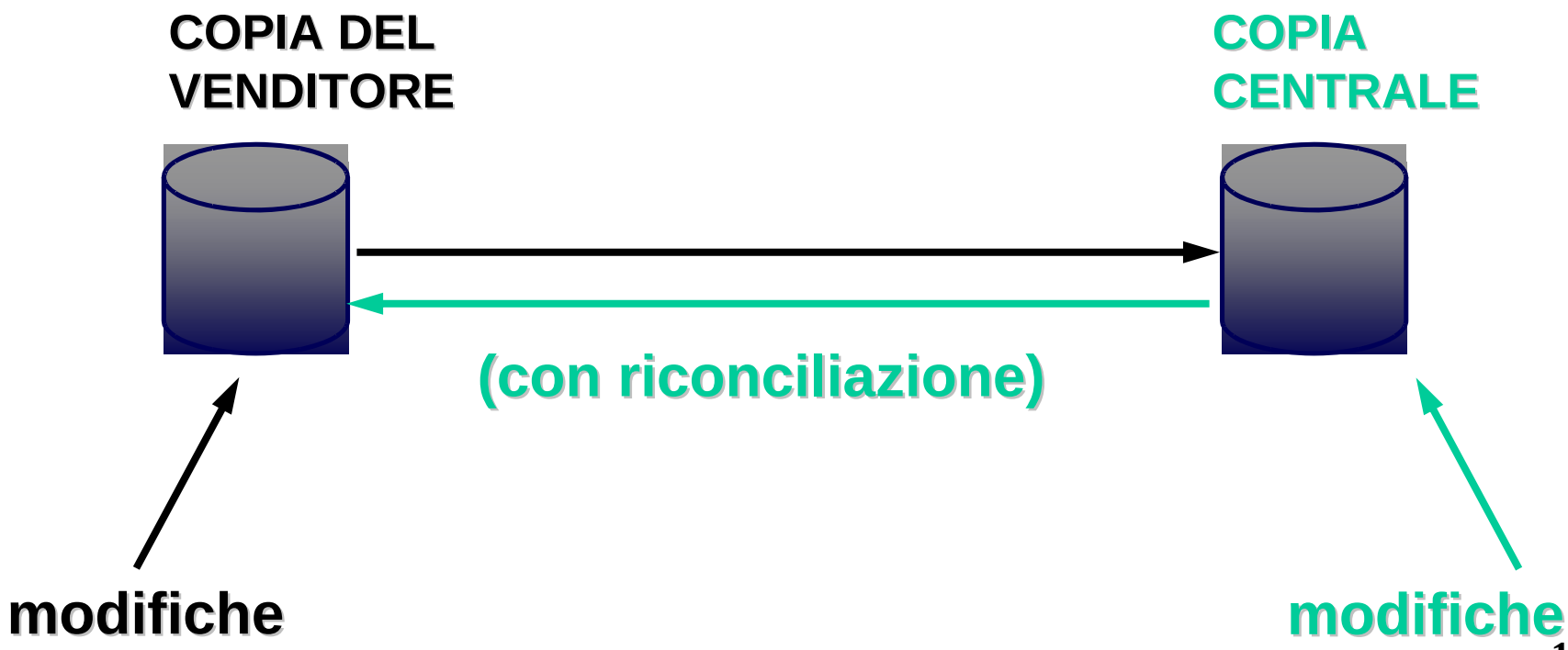


Un caso particolare: replicazione in computer mobili

- **Computer mobili :**
  saltuariamente collegati ad una rete
- **Copie disconnesse per ore o giorni intere, poi riconnesse (riconciliazione)**
- **Applicazione :**
  agenti di vendita mobili

Allineamento di copie disconnesse

- **Richiede spesso la replicazione simmetrica**



Trigger di replicazione

catturano le variazioni ai dati nelle tabelle DELTA-PLUS e DELTA-MINUS in modo trasparente alle applicazioni

Trigger di replicazione

CREATE TRIGGER CAPTURE-INS
AFTER INSERT ON PRIMARY
FOR EACH ROW
INSERT INTO DELTA-PLUS VALUES (NEW.*)

CREATE TRIGGER CAPTURE-DEL
AFTER DELETE ON PRIMARY
FOR EACH ROW
INSERT INTO DELTA-MINUS VALUES (OLD.*)

CREATE TRIGGER CAPTURE-UPD
AFTER UPDATE ON PRIMARY
FOR EACH ROW
BEGIN
INSERT INTO DELTA-PLUS VALUES (NEW.*)
INSERT INTO DELTA-MINUS VALUES (OLD.*)
END