The background of the slide features a chalkboard with a large white chalk 'X' drawn on it. Below the 'X', the letters 'ECCO' are written in large, stylized, white chalk. The chalkboard has a textured, light blue-green tint.

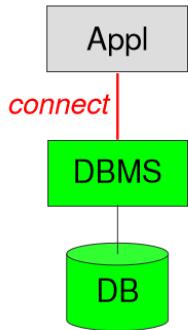
Architettura DB

DDBMS architecture

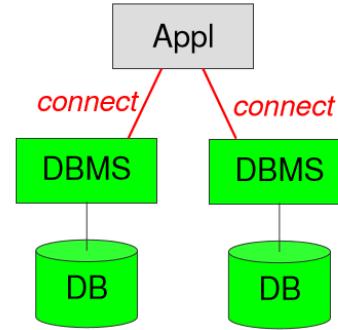
Types of DBMSs

Distributed data: summary

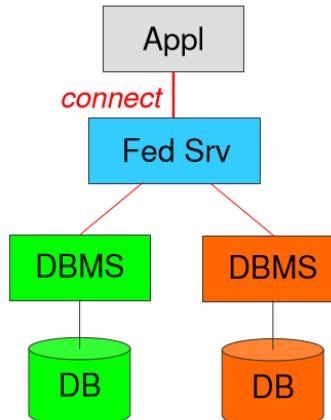
DATA MOVE: NO



Basic (single db)

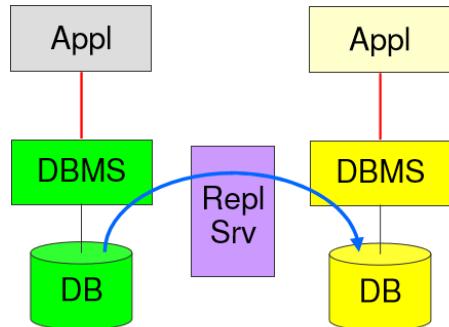


Distributed Access

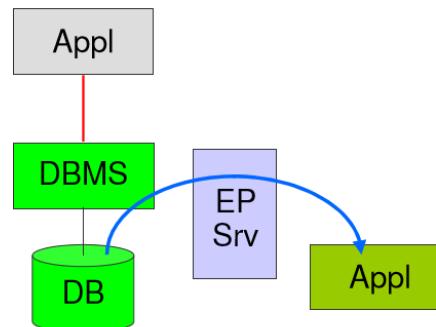


Federation

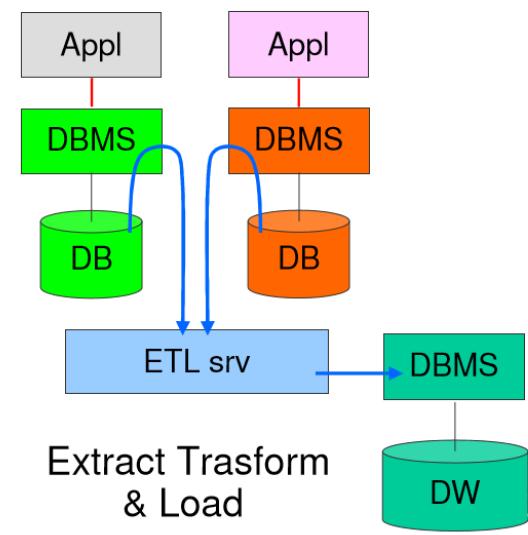
DATA MOVE: YES



Replication

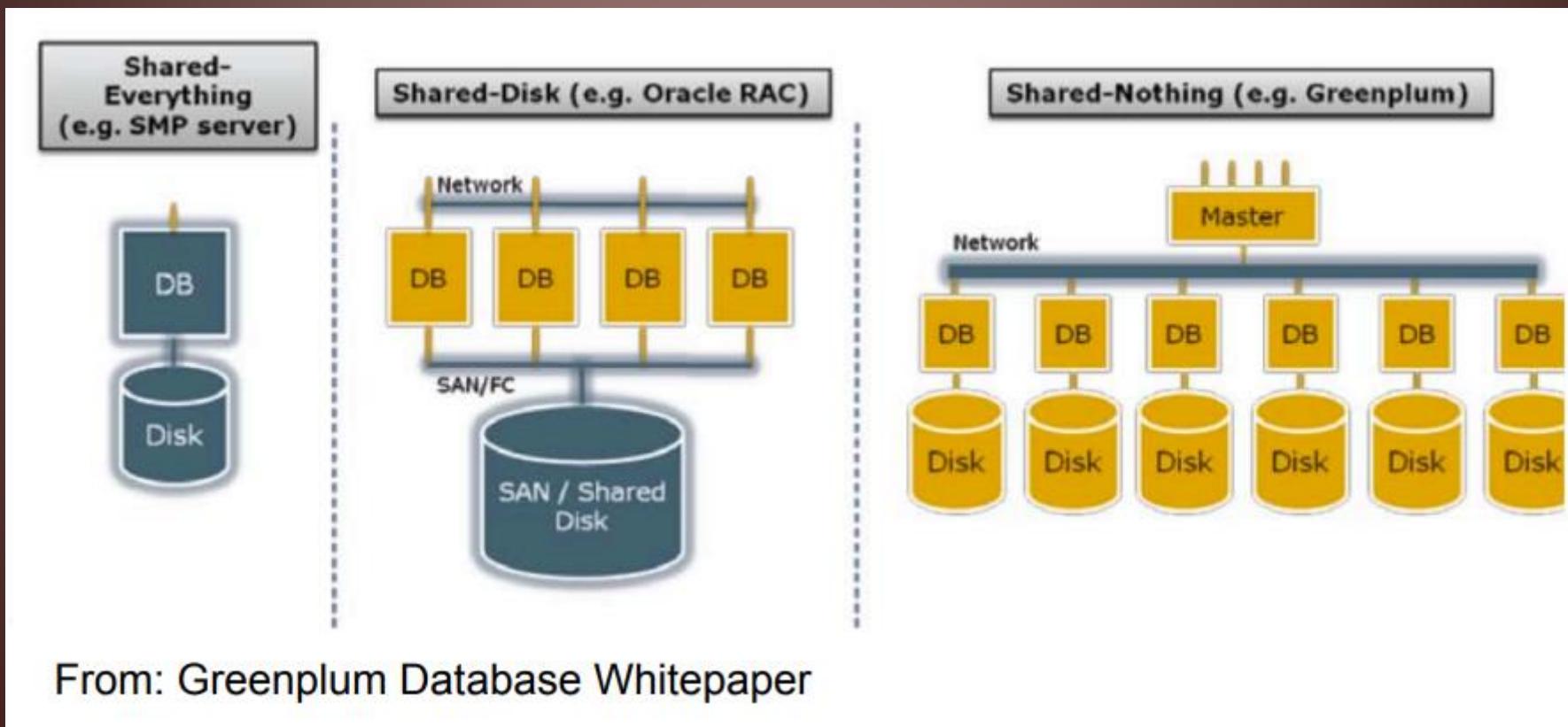


Event Publishing



Extract Trasform
& Load

Type of database architecture

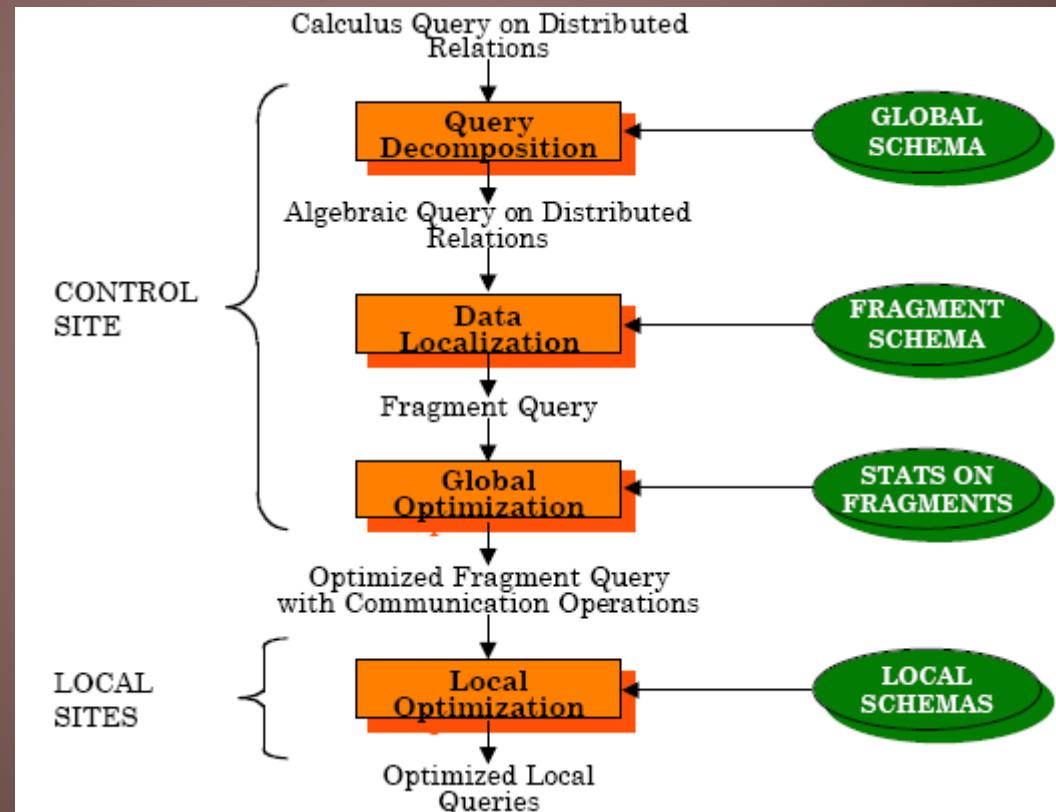


From: Greenplum Database Whitepaper

Strategie di Query processing nei DDBMS

Query distribuite

- Dove viene deciso il piano di esecuzione delle query?



Fasi del query processing nei DDBMS

- 1. Query decomposition
 - 2. Data localization
 - 3. Global query optimization
 - 4. Local optimization
-
- Vediamole in dettaglio, insieme agli input alla fase

Fasi del query processor - 1

- 1. Query decomposition \leftarrow Global schema
 - Opera sullo schema logico globale
 - Non considera la distribuzione
 - Usa tecniche di ottimizzazione algebrica analoghe a quelle centralizzate
 - Output: un query tree – non ottimizzato rispetto ai costi di comunicazione

Fasi del query processor - 1

- 2. Data localization ← Fragment schema
 - Considera la distribuzione dei frammenti
 - Ottimizza le operazioni rispetto alla frammentazione, con tecniche di riduzione
 - Output: una query che opera in modo efficiente sui frammenti – non ottimizzata

Esempio

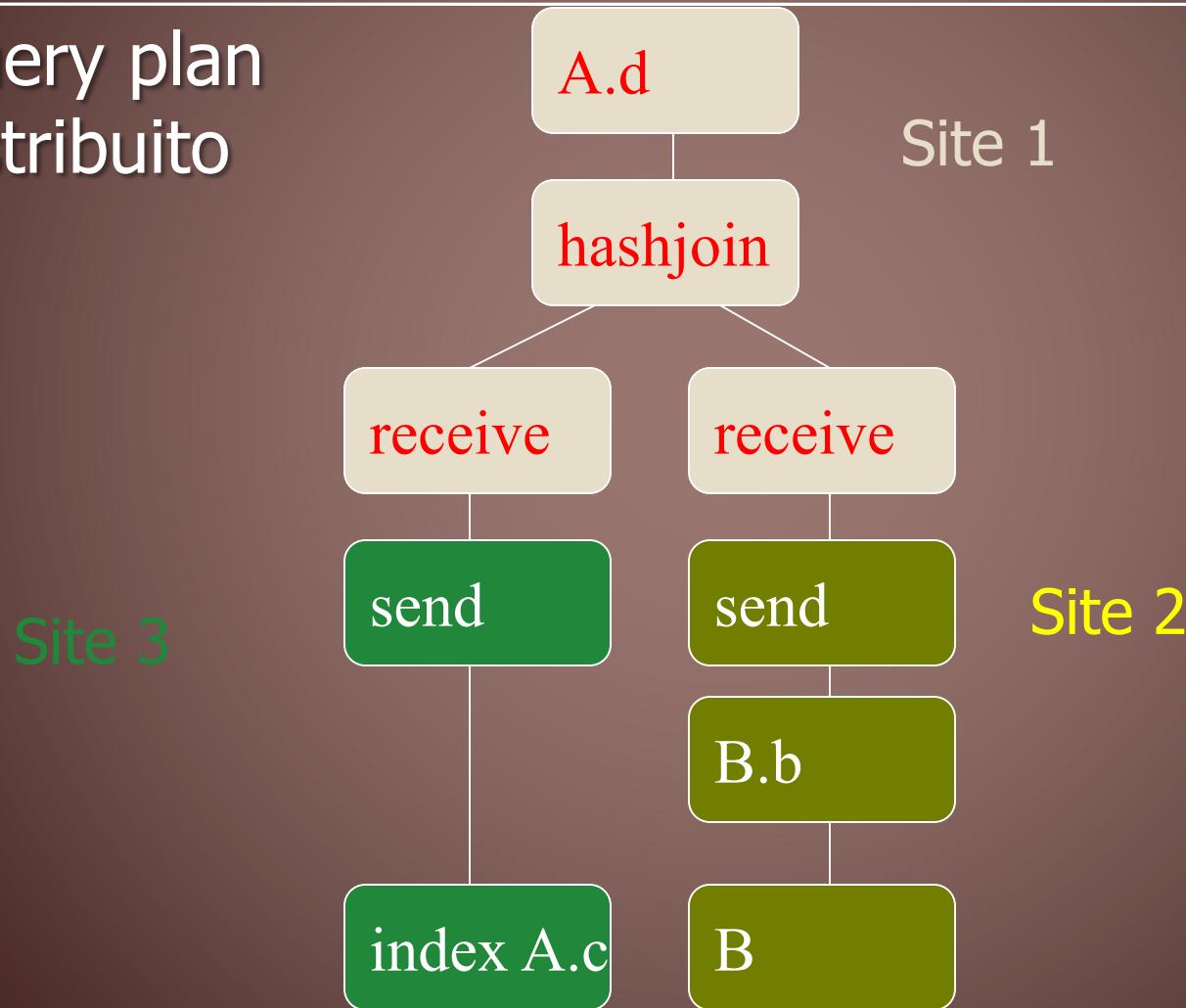
- Es. $\sigma_{\text{eno}} = \text{'E20'}$ EMP \rightarrow
- Supponiamo che EMP sia rappresentata nei nodi 1, 2, e 3.
- La query puo' essere inizialmente frammentata nella
- $\sigma_{\text{eno}} = \text{'E20'}$ EMP1 \cup $\sigma_{\text{eno}} = \text{'E20'}$ EMP2 \cup $\sigma_{\text{eno}} = \text{'E20'}$ EMP3
- Applichiamo una tecnica di riduzione (supponiamo che E20 sia solo in EMP2) $\rightarrow \sigma_{\text{eno}} = \text{'E20'}$ EMP2

Fasi del query processor - 3

- 3. Global query optimization ← statistiche sui frammenti
 - Strategia di esecuzione: nel query tree agli operatori di Algebra relazionale vengono aggiunti gli operatori di comunicazione (send/receive tra nodi)
 - Obiettivo: trovare l'ordinamento “migliore” delle operazioni definite dalla fragment query
 - Utilizza modelli di costo che tengono conto dei costi di comunicazione
 - Output: le decisioni piu’ rilevanti riguardano i join:
 - 1. L’ordine dei join n-ari →
 - 2. La scelta tra join e semijoin →

Fasi del query processor - 3

- Query plan distribuito



Fasi del query processor - 3

- Il modello di costo presentato è „ideale“
- Algoritmi di calcolo del costo adattativi
- Gestiscono gli eventi non predibili
 - Ritardo nella rete
 - I nodi possono cambiare la gestione delle policies
- Esiste una fase di re-ottimizzazione a run time
 - Si fa il monitor dell'esecuzione delle query
 - Si adatta il modello di costo
 - Nel caso si riottimizza la query se lo scarto è molto elevato

Esempio

Distributed query processing – esempio

Schema:

Employee (eno, ename, title)

AssiGN(eno, projectno, resp, dur)

dove resp indica il tipo di responsabilita'

|EMP| = 400, |ASG| = 1000

Query: "trovare nomi dei dipendenti che sono manager di progetti"

SQL (trasparenza di frammentazione):

SELECT ename

FROM Employee E JOIN AssiGN A on E.eno=A.eno

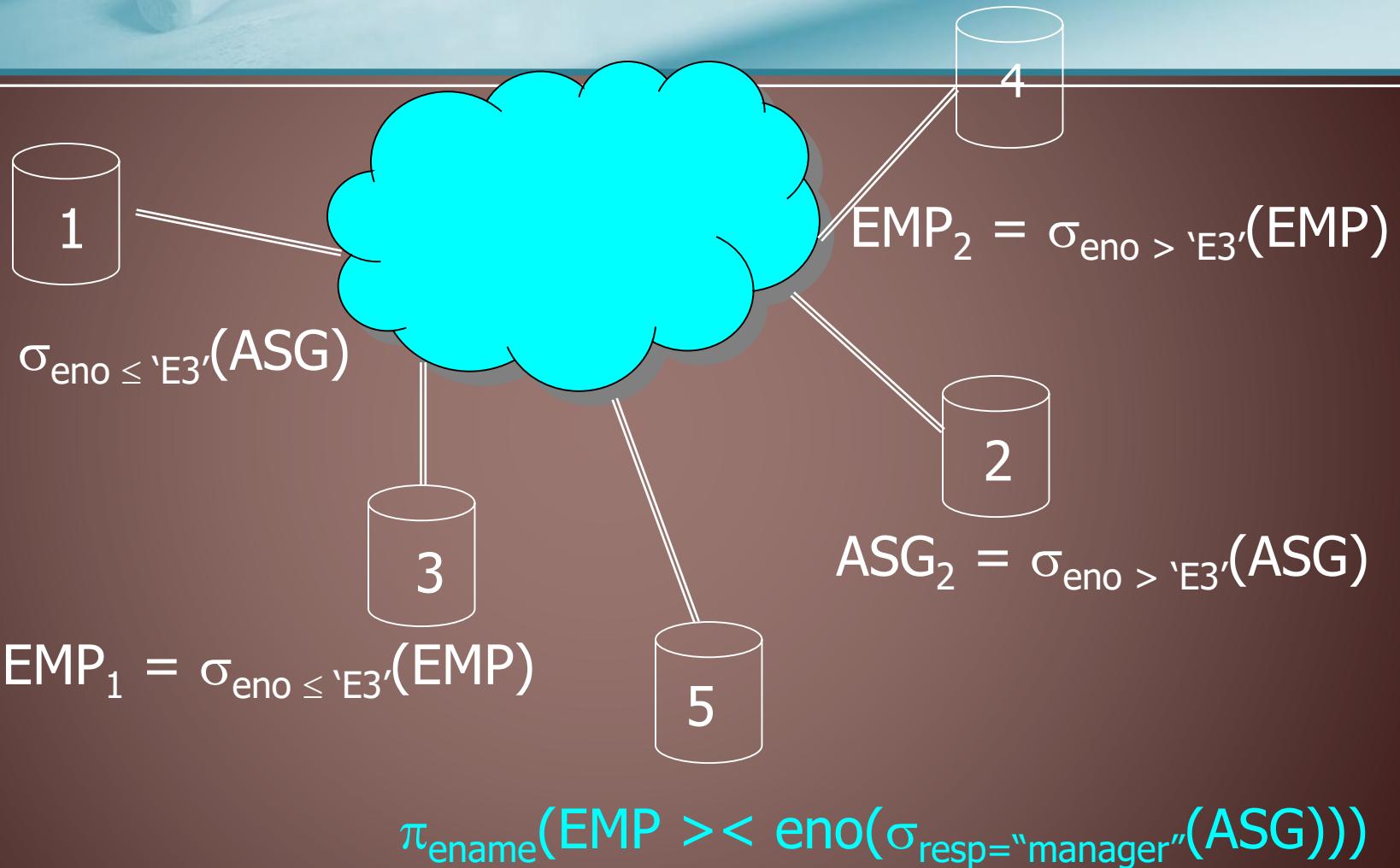
WHERE resp="manager"

AR: $\pi_{ename}(\text{EMP} ><_{\text{eno}} (\sigma_{\text{resp}=\text{"manager"}}(\text{ASG})))$

Distribuzione

- Supponiamo una frammentazione orizzontale di questo tipo
- $\text{ASG1} = \sigma_{\text{eno}} \leq 'E3'(\text{ASG})$ nodo 1
- $\text{ASG2} = \sigma_{\text{eno}} > 'E3'(\text{ASG})$ nodo 2
- $\text{EMP1} = \sigma_{\text{eno}} \leq 'E3'(\text{EMP})$ nodo 3
- $\text{EMP2} = \sigma_{\text{eno}} > 'E3'(\text{EMP})$ nodo 4
- Risultato nodo 5

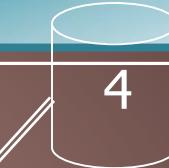
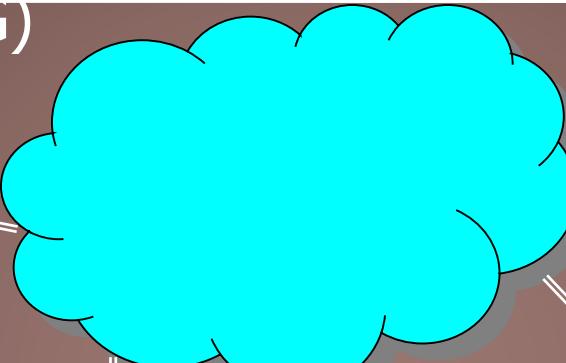
Allocazione dei frammenti



Esecuzione della query: Strategia A

$$\text{EMP}_2 = \sigma_{\text{eno} > 'E3'}(\text{EMP})$$

$$\text{ASG}_1 = \sigma_{\text{eno} \leq 'E3'}(\text{ASG})$$



$\text{ASG}'2$

$$\text{EMP}'2 \leftarrow \text{EMP2} >< \text{eno} \text{ ASG}'2$$

$$\text{ASG}'1 \leftarrow$$

$$\sigma_{\text{resp}=\text{"manager"}(\text{ASG1})}$$



$$\text{ASG}'2 \leftarrow \sigma_{\text{resp}=\text{"manager"}(\text{ASG2})}$$

$$\text{EMP}_1 = \sigma_{\text{eno} \leq 'E3'}(\text{EMP})$$

$\text{ASG}'1$

$$\text{EMP}'1 \leftarrow$$

$$\text{EMP1} >< \text{eno} \text{ ASG}'1$$



$$\text{ASG}_2 = \sigma_{\text{eno} > 'E3'}(\text{ASG})$$

$$\text{EMP}'_1 \quad \text{EMP}'_2$$

$$\text{Result} = \text{EMP}'1 \cup \text{EMP}'2 =$$

$$= \text{EMP} >< \text{eno}(\sigma_{\text{resp}=\text{"manager"}(\text{ASG})}) \rightarrow$$

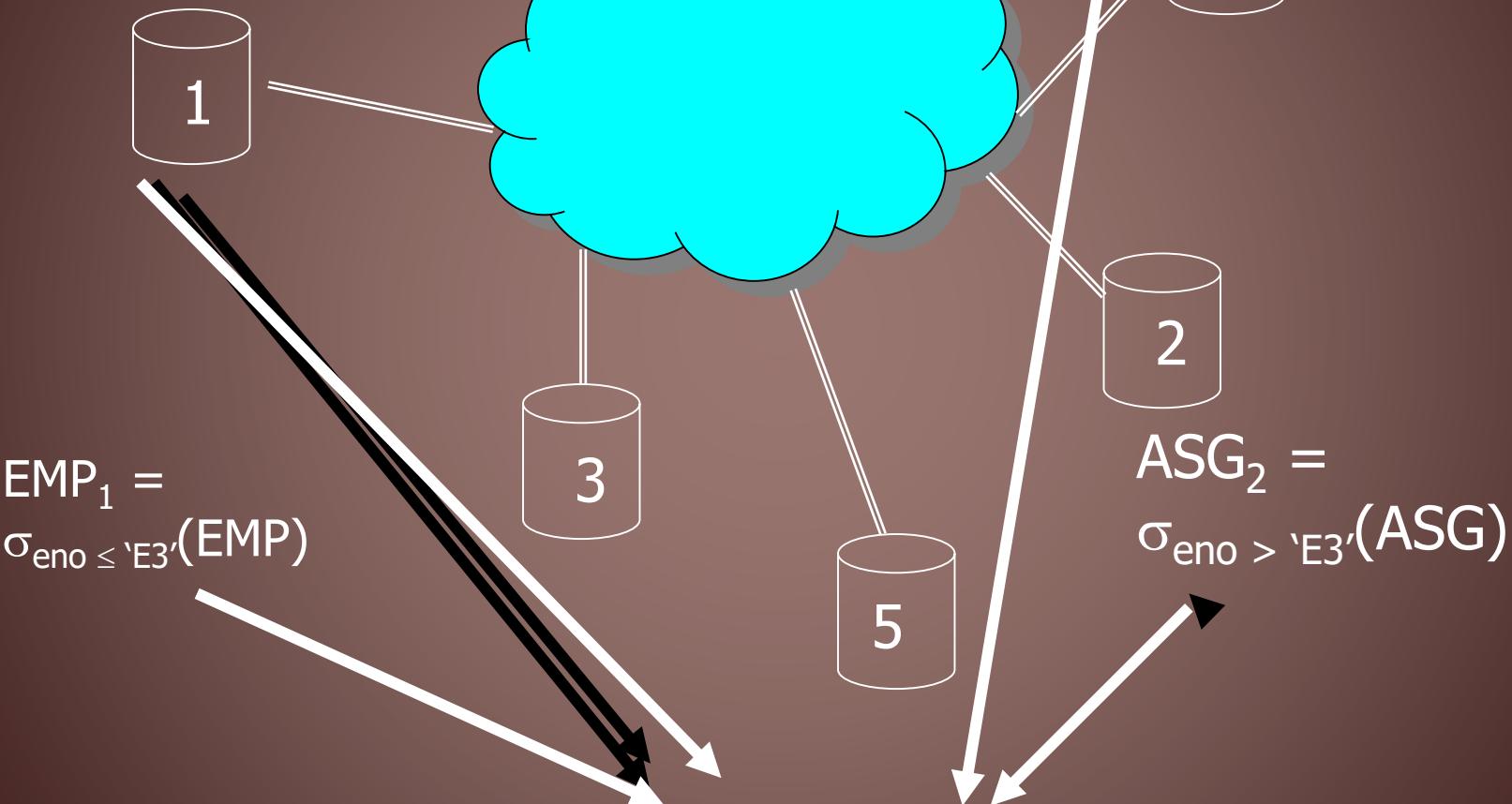
$$\pi_{\text{ename}} \text{EMP} >< \text{eno}(\sigma_{\text{resp}=\text{"manager"}(\text{ASG})})$$

Esecuzione della query:

Strategia B

$$\text{EMP}_2 = \sigma_{\text{eno} > 'E3'}(\text{EMP})$$

$$\text{ASG}_1 = \sigma_{\text{eno} \leq 'E3'}(\text{ASG})$$

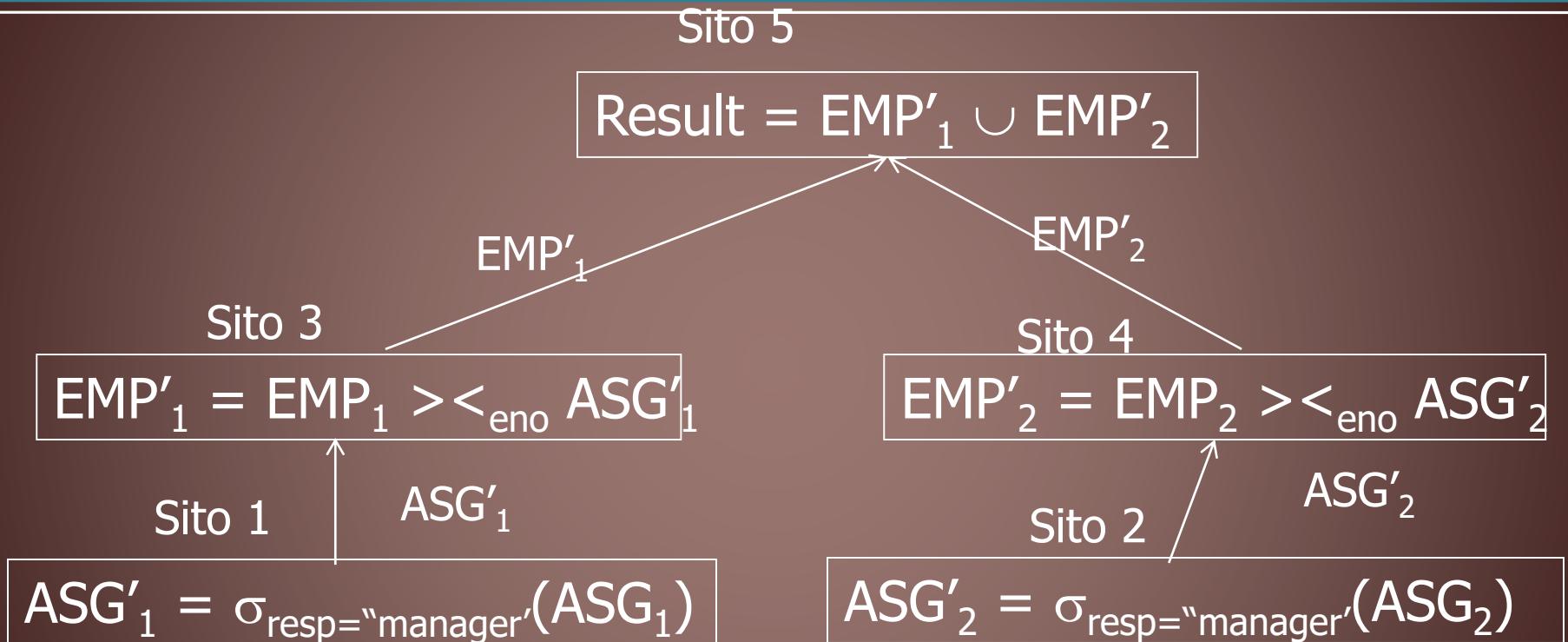


$$\text{EMP}_1 = \sigma_{\text{eno} \leq 'E3'}(\text{EMP})$$

$$\text{ASG}_2 = \sigma_{\text{eno} > 'E3'}(\text{ASG})$$

Tutte le tabelle vengono inviate al nodo 5
Dove viene calcolato il risultato

Rappresentazione della strategia A come operazioni + trasmissioni



Strategia B - Rappresentazione come operazioni + trasmissioni

Sito 5

Risultato = $(\text{EMP}_1 \cup \text{EMP}_2) > <_{\text{ENO}} \sigma_{\text{resp}=\text{"manager'}}(\text{ASG}_1 \cup \text{ASG}_2)$



Confronto di costi tra le due strategie

- Modelli di costo (semplificato):
 - Costo accesso a un record: 1
 - Costo trasferimento di un record: 10
 - $|EMP| = 400$, $|ASG| = 1000$, $|\sigma_{\text{resp}=\text{"manager}}(ASG_1)| = |\sigma_{\text{resp}=\text{"manager}}(ASG_2)| = 10$
 - Accesso diretto a EMP via ENO, ASG via RESP

Costo strategia A:

1. Calcolo ASG'_1 e $ASG'_2 \rightarrow 10+10$ per accesso diretto
 2. Trasferimento ASG'_1 e $ASG'_2 \rightarrow 20 \times 10 = 200$
 3. Calcolo EMP'_1 ed EMP'_2 : join (single-loop) $\rightarrow (10 + 10) \times 2$
 4. Trasferimento EMP'_1 , EMP'_2 : $20 \times 10 = 200$
- Totale: 460 (400 trasferimento e 60 calcolo)

Confronto di costi tra le due strategie

- Modelli di costo (semplificato):
 - Accesso a un record: 1
 - Trasferimento di un record: 10
 - $|EMP| = 400, |ASG| = 1000, |\sigma_{\text{resp}=\text{"manager}}(ASG_1)| = |\sigma_{\text{resp}=\text{"manager}}(ASG_2)| = 10$
 - Accesso diretto a EMP via ENO, ASG via RESP
- Costo strategia B (non si trasferiscono gli indici!):
 1. Trasferimento EMP_1, EMP_2 sul nodo 5: $400 \times 10 = 4.000$
 2. Trasferimento ASG_1, ASG_2 sul nodo 5: $1000 \times 10 = 10.000$
 3. Calcolo ASG' (selezione): 1.000 (no indice)
 4. Join ASG', EMP: $20 \times 400 = 8.000$ (non ricostruisce gli indici
→ nested loop)
- Totale: 23.000 (14.000 trasferimento e 9.000 calcolo)

Obiettivi del query processing distribuito: parametri utilizzati

- Costo totale =
somma dei costi delle operazioni (I/O e
CPU) +
costi di comunicazione (trasmissione)
- Response time = somma dei costi
tenendo conto del parallelismo

Costo

Rispetto al caso centralizzato, in cui i costi piu' rilevanti sono quelli di trasferimento dei blocchi da memoria secondaria a principale, consideriamo qui **i soli costi di comunicazione e trascuriamo i costi di I/O.**

Costo comunicazione = $C_{MSG} * \#msgs + C_{TR} * \#bytes$

C_{MSG} = costo fisso di spedizione/ricezione messaggio (setup)

C_{TR} = costo (fisso rispetto alla topologia!) di trasmissione dati

Tempo di risposta

Nel tempo di risposta, a differenza del costo di trasmissione, i costi delle operazioni in parallelo non si sommano

Tempo di risposta (solo comunicazione) =

$$C_{MSG} * \text{seq_}\#msg + C_{TR} * \text{seq_}\#bytes$$

dove seq_#msgs e' il massimo numero di messaggi che devono essere comunicati in modo sequenziale.

Rapporto tra costo di comunicazione e costo di I/O

- Nelle grandi reti geografiche

costo di comunicazione >> costo di I/O (fattore 1 a 10)

- Nelle reti locali

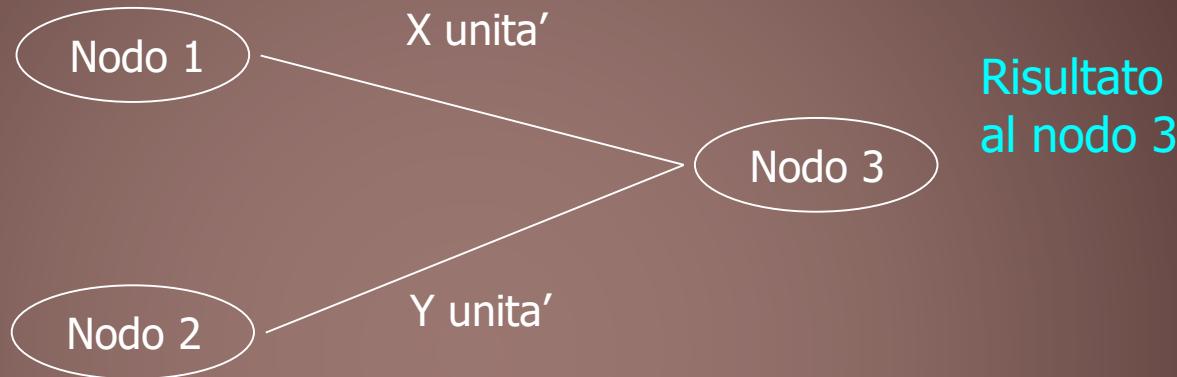
costo di comunicazione e costo di I/O sono paragonabili

- Tendenza

il costo di comunicazione e' ancora il fattore critico, ma si stanno avvicinando

Possiamo, in alternativa a trascurare i costi di I/O, utilizzare pesi nelle formule di costo

Esempio



- Costo di trasferimento di x unita' da 1 a 3 e di y unita' da 2 a 3:
- Costo comunicazione = $2 C_{MSG} + C_{TR} * (x + y)$
- Tempo di risposta = $\max(C_{MSG} + C_{TR} * x, C_{MSG} + C_{TR} * y)$

dato che x e y vengono trasferiti in parallelo

Rapporto tra tempo di risposta e costo

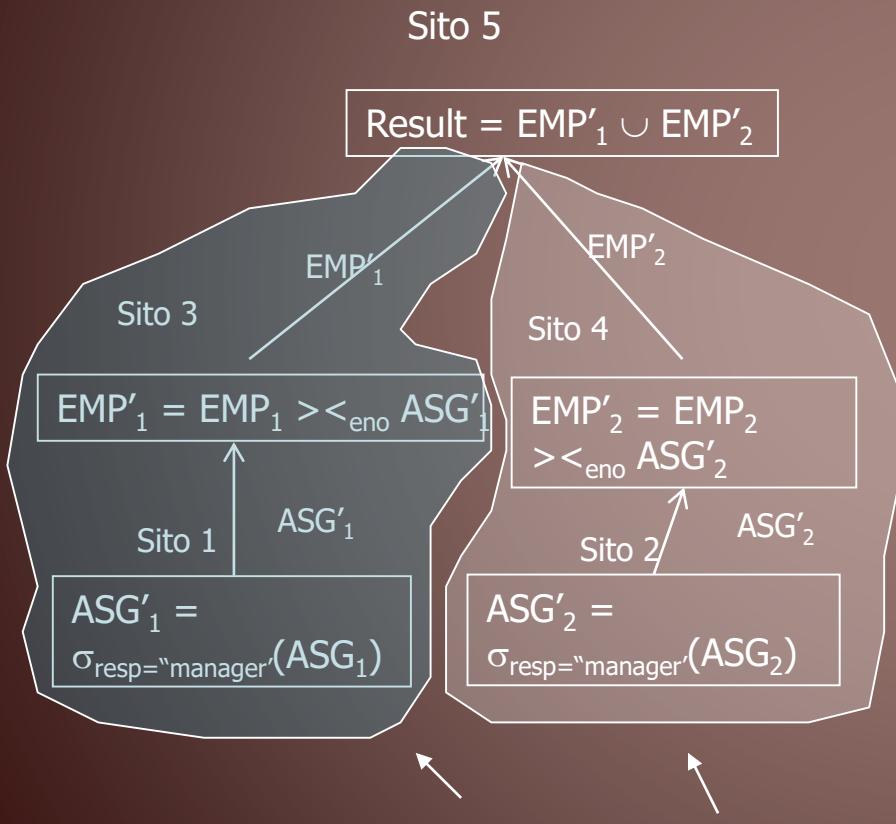
Possiamo essere interessati a:

- * Minimizzazione tempo di risposta: piu' parallelismo → puo' portare ad aumento del costo totale (maggiore numero di trasmissioni e processing locale)
- * Minimizzazione costo totale = somma dei costi senza tener conto del parallelismo: utilizza meglio le risorse → aumento del throughput (con peggioramento del response time in generale)

Nel nostro esempio

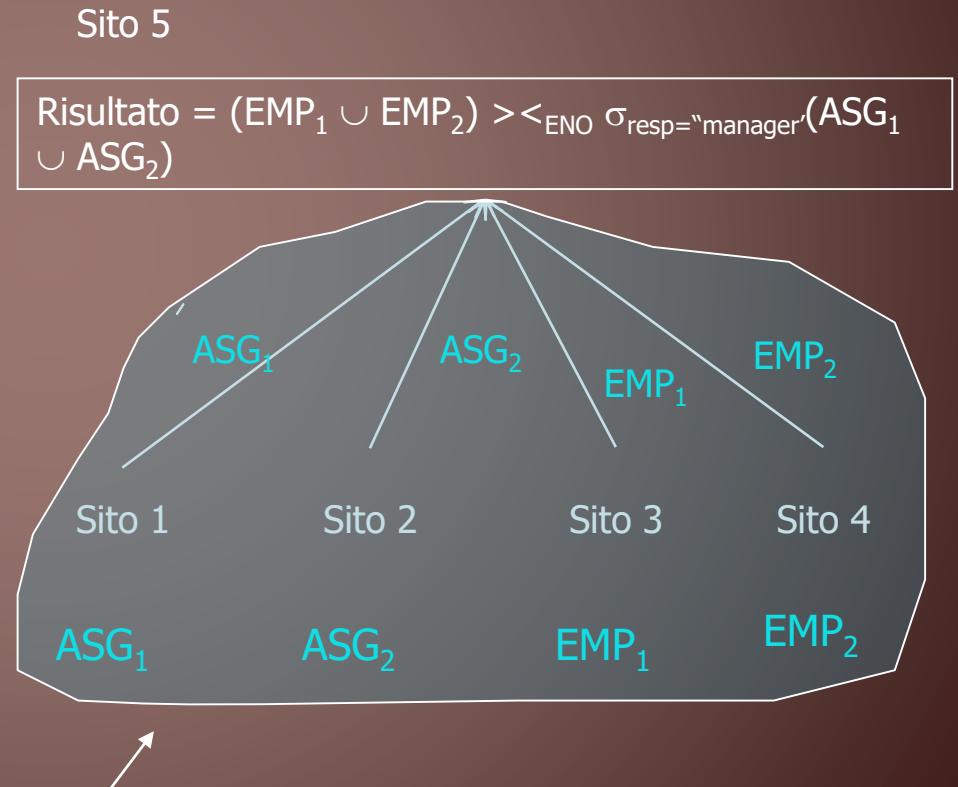
Strategia A

Ottimizza il costo



Strategia B

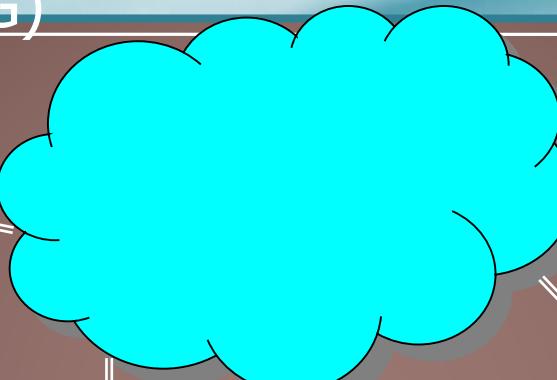
Parallelizza di piu' ma, in questo caso,
non minimizza il tempo di risposta



Operazioni e trasmissioni eseguibili in parallelo

Esempio -Strategia A

$ASG_1 = \sigma_{eno \leq 'E3'}(ASG)$



$EMP_1 = \sigma_{eno \leq 'E3'}(EMP)$

$ASG'1$

$EMP'1 \leftarrow EMP1 >< eno ASG'1$

$ASG'1 \leftarrow \sigma_{resp="manager"}(ASG1)$

$ASG'2 \leftarrow \sigma_{resp="manager"}(ASG2)$

$ASG_2 = \sigma_{eno > 'E3'}(ASG)$

$EMP'2 \leftarrow EMP2 >< eno ASG'2$

$EMP_2 = \sigma_{eno > 'E3'}(EMP)$

EMP'_1

EMP'_2

$Result = EMP'1 \cup EMP'2 =$

$= EMP >< eno(\sigma_{resp="manager"}(ASG)) \rightarrow$

$\pi_{ename} EMP >< eno(\sigma_{resp="manager"}(ASG))$

2. Join e semijoin

Situazione di partenza



Join e semijoin - 1

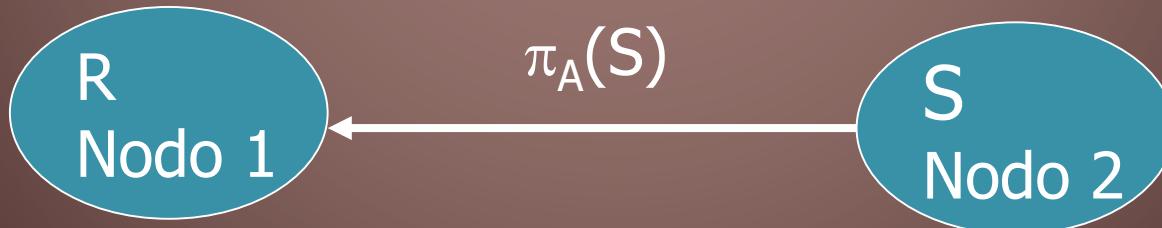
- In un DBMS distribuito l'operazione di semijoin puo' essere in alcune circostanze una alternativa piu' efficiente alla operazione di join.
- Definizione: $R \text{ semijoin}_A S \equiv \pi_{R^*}(R \text{ join}_A S)$
- Dove R^* e' l'insieme degli gli attributi di R
- Il semijoin $R \text{ semijoin}_A S$ perciò e' la proiezione sugli attributi di R della operazione di join
- Attenzione: il semijoin e' non comutativo!

Join e semijoin - 2

Osservazione 1: dalla definizione di semijoin

- $R \text{ semijoin}_\theta S \equiv \pi_{R^*}(R \text{ join}_A S)$

il calcolo di $(R \text{ semijoin}_A S)$ richiede la presenza del solo attributo $S.A$ da parte di S , cioe' solo di $\pi_A(S)$



Join e semijoin - 3

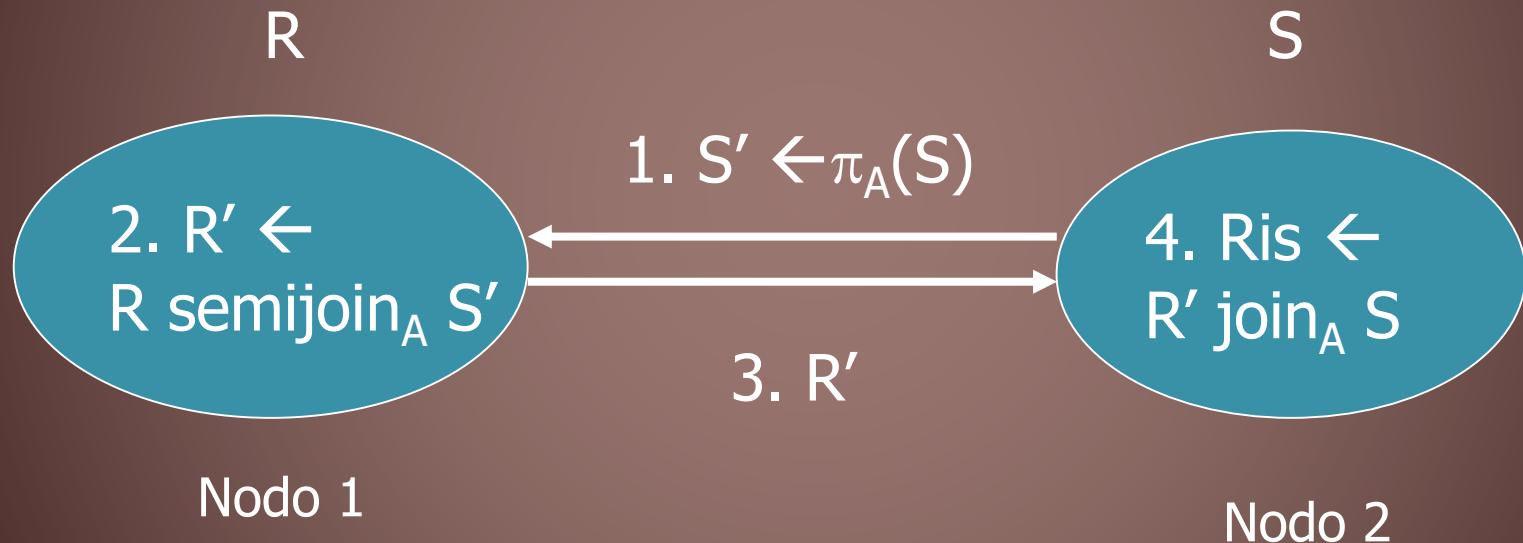
Osservazione 1: se R , S sono allocate su nodi diversi, allora $R \text{ join}_\theta S$ puo' essere calcolato tramite operazioni di semijoin piuttosto che di join. Valgono infatti le seguenti equivalenze (vedi ad es. [Ozsuvard. sec.9.3.2]) tra join e semijoin:

1. $R \text{ join}_\theta S \Leftrightarrow (R \text{ semijoin}_\theta S) \text{ join}_\theta S$
2. $R \text{ join}_\theta S \Leftrightarrow R \text{ join}_\theta (S \text{ semijoin}_\theta R)$
3. $R \text{ join}_\theta S \Leftrightarrow (R \text{ semijoin}_\theta S) \text{ join}_\theta (S \text{ semijoin}_\theta R)$

Ognuna da' luogo ad una diversa strategia. La scelta tra le strategie richiede la stima dei loro costi

Esempio – formula 1

$$R \text{ join}_A S \Leftrightarrow (R \text{ semijoin}_A S) \text{ join}_A S$$



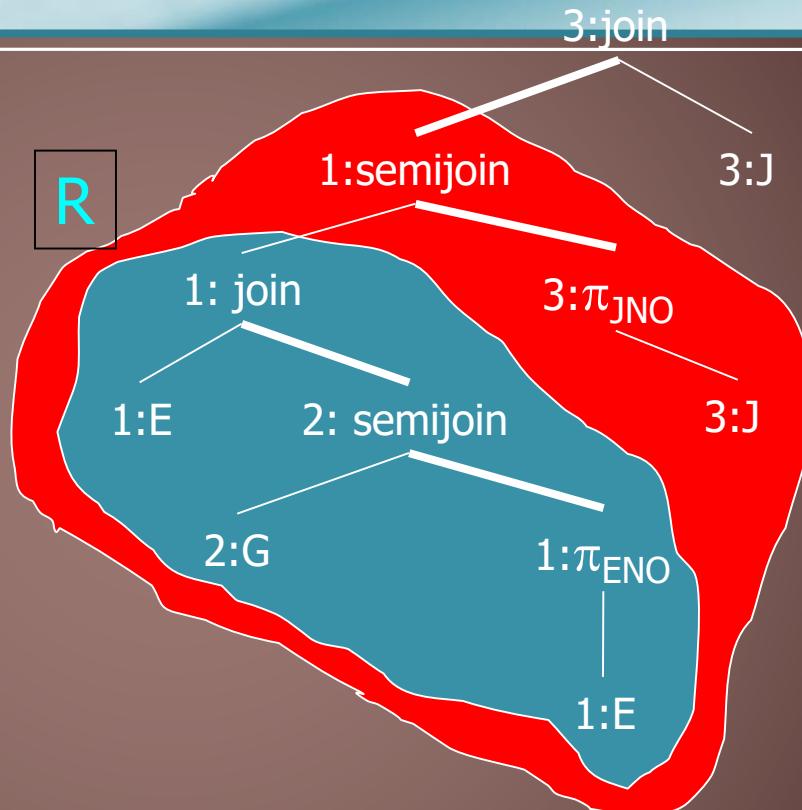
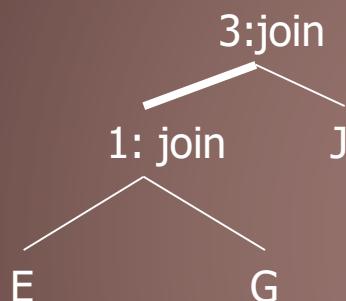
Confronto costo join e semijoin

In generale:

L'uso del semijoin e' conveniente se il costo del suo calcolo e del trasferimento del risultato e' inferiore al costo del trasferimento dell'intera relazione e del costo del join intero

Esempio con tre relazioni

$E \text{ join } G \text{ join } J$



$$R = G \text{ join}_{\text{eno}} E = (G \text{ semijoin}_{\text{eno}} E) \text{ join}_{\text{eno}} E = \pi_{G.*}(G \text{ join}_{\text{eno}} (\pi_{\text{eno}}(E))) \text{ join}_{\text{eno}} E$$

$$R \text{ join}_{\text{jno}} J = (R \text{ semijoin}_{\text{jno}} J) \text{ join}_{\text{jno}} J = \pi R.* (R \text{ join}_{\text{jno}} (\pi_{\text{jno}}(E))) \text{ join}_{\text{jno}} J$$

Fasi del query processor - 4

- 4. Local query optimization ← local schemas
 - Ogni nodo riceve una fragment query e la ottimizza in modo indipendente
 - Vengono utilizzate tecniche analoghe a quelli dei sistemi centralizzati

Tornando al peso dei costi di comunicazione

Per DDBMS in rete geografica, conviene

- 1 Global query optimization con obiettivo:
ridurre i costi di comunicazione
- 2 Seguita da local optimization

Per DDBMS in rete locale, conviene

- 1 Global query optimization con obiettivo:
aumentare il parallelismo
- 2 Seguita da local optimization

Progettazione di basi di dati distribuite

- Nella progettazione delle basi di dati distribuite si dovrebbe anche tenere conto di
- Topologia della rete
- Tipologie di query distribuite
- Stime o statistiche su query distribuite

2. Controllo di concorrenza

DDBMS e transazioni - classificazione

- Dirette ad un unico server remoto
 - Remote requests:** transazioni read-only
 - Numero arbitrario di query SQL
 - Remote transactions:** transazioni read-write
 - Numero arbitrario di operazioni SQL (select, insert, delete, update)

Classificazione - 2

- Dirette ad un numero arbitrario di server
 - **Distributed requests:**
 - Read only arbitrarie, nelle quali ogni singola operazione SQL si puo' riferire a qualunque insieme dei server
 - Richiede un ottimizzatore distribuito
 - **Distributed transactions:**
 - Numero arbitrario di operazioni SQL (select, insert, delete, update)
 - Ogni operazione e' diretta ad un unico server
 - Le transazioni possono modificare piu' di un DB
 - Richiede un protocollo transazionale di coordinamento distribuito → **two-phase commit**

Esempio di transazione distribuita: trasferimento di un importo tra conti bancari

Base dati: ACCOUNT (AccNum,Name,Total):

Frammentazione

- AccNum < 10000 → frammento ACCOUNT1 sul nodo 1
- AccNum >= 10000 → frammento ACCOUNT2 sul nodo 2

```
begin transaction
update Account1
    set Total = Total - 100000 where AccNum = 3154;
update Account2
    set Total = Total + 100000 where AccNum = 14878;
commit work;
end transaction
```

- Nota: deve comunque valere la proprieta' di atomicita'
rispetto alle due updates

DDBMS e proprietà ACID

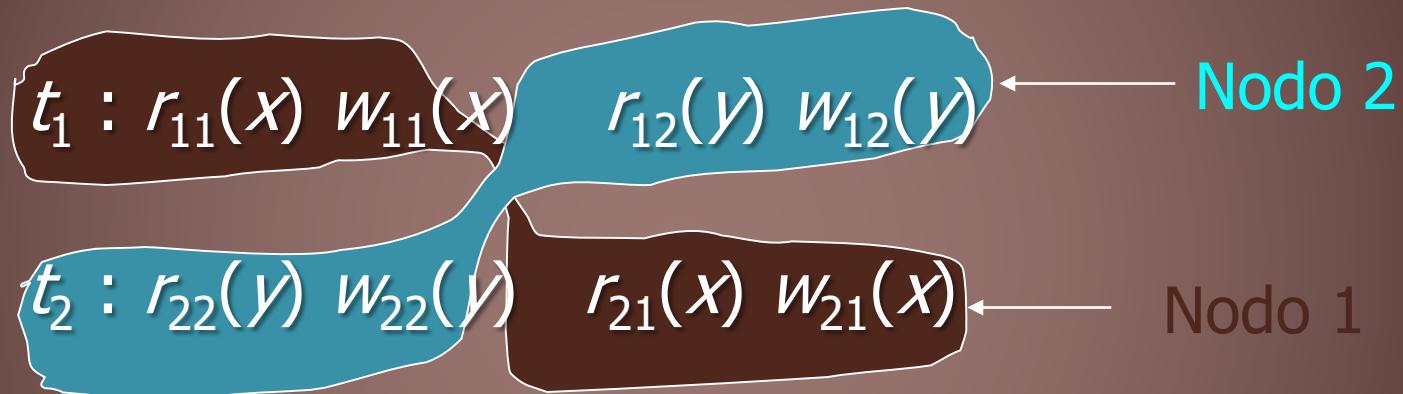
- La distribuzione non ha conseguenze su consistenza e durabilità
 - **Consistenza**: non dipende dalla distribuzione, perché i vincoli descrivono solo proprietà logiche dello schema (indipendenti dall'allocazione)
 - **Durabilità**: garantita localmente da ogni sistema
- Invece, è necessario rivedere alcuni componenti dell'architettura:
 - Concurrency control (**Isolamento**)
 - Reliability control, recovery manager (**Atomicità**)



Controllo di concorrenza (Isolamento)

DDBMS e Concurrency Control

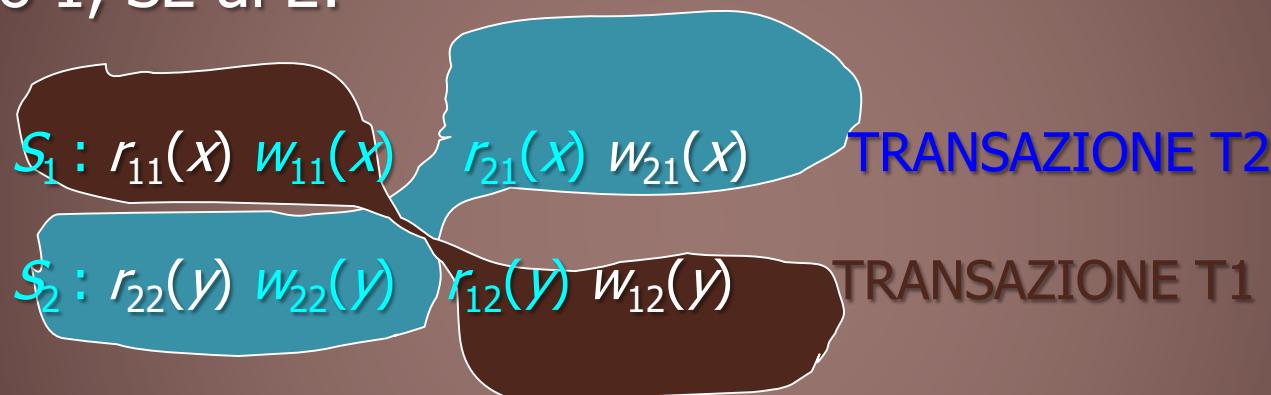
- In questo caso, una transazione t_i si scomponе in sotto-transazioni t_{ij} . La sotto-transazione t_{ij} viene eseguita sul nodo j:



- Ogni sotto-transazione viene schedulata in modo indipendente dai server di ciascun nodo
- La schedule globale dipende quindi dalle schedules locali su ogni nodo

Serializzabilita' globale

- Osservazione: la serializzabilita' locale di ogni schedule non garantisce la sua serializzabilita' globale. Esempio S1 al nodo 1, S2 al 2:



- X allocato al nodo 1, Y allocato al nodo 2
- S_1 e S_2 sono schedules locali, ciascuna e' seriale
- Il **grafo globale dei conflitti** ha un ciclo:
 - Sul nodo 1, t_1 precede t_2 ed e' in conflitto con t_2
 - Sul nodo 2, t_2 precede t_1 ed e' in conflitto con t_1



Serializzabilita' globale nel caso non replicato

- Se il DB non e' replicato e ogni schedule locale e' serializzabile, allora

- la schedule globale e' serializzabile se gli ordini di serializzazione sono gli stessi per tutti i nodi coinvolti

Serializzabilità e controllo delle repliche

- 1

- Quando il DB contiene repliche le cose cambiano ..
 - Semplice esempio di problema di serializzazione:
 - DB su due nodi, X replicata nei due nodi, chiamiamo x_1 e x_2 le due copie.
 - Due transazioni:
 - T1: **Read (x) $x \leftarrow x + 5$ Write (x) Commit**
 - T2: **Read (x) $x \leftarrow x * 10$ Write (x) Commit**
 - Che possiamo trascrivere in
 - T1: $r1(x)$ $w1(x)$
 - T2: $r2(x)$ $w2(x)$

Serializzabilità e controllo delle repliche

- 2

Consideriamo le seguenti due schedule che possono essere generate ai due nodi 1 e 2:

S1 al nodo 1 : $r_{11}(x) w_{11}(x)$, $r_{21}(x) w_{21}(x)$

S2 al nodo 2 : $r_{22}(x) w_{22}(x)$ $r_{12}(x) w_{12}(x)$

Sia S1 che S2 sono seriali, ma serializzano T1 e T2 in ordine opposto, come nell'esempio precedente

In questo caso, viene violata la *mutua consistenza* dei due DB locali, per la quale tutte le copie devono avere lo stesso valore al termine della transazione

Se inizialmente $x_1 = x_2 = 1$, alla fine: $x_1 = 60$, $x_2 = 15$

Abbiamo bisogno di un protocollo di controllo delle repliche

Protocollo ROWA di controllo delle repliche:

- Read Once Write All (ROWA)
- Dato un item X con copie X_1, \dots, X_n :
- X e' detto item logico, X_1, \dots, X_n sono items fisici
- Le transazioni "vedono" solo X
- Il protocollo di controllo mappa:
 - Read(X) su una (qualsiasi) delle copie
 - Write(X) su tutte le copie
- Questa condizione puo' essere rilassata con protocolli asincroni, piu' efficienti, ma non ce ne occupiamo
- E' implementato nelle estensioni a 2PL – prossimo argomento



Il 2 Phase Locking in ambito DDBMS

Estensione di 2PL al caso distribuito

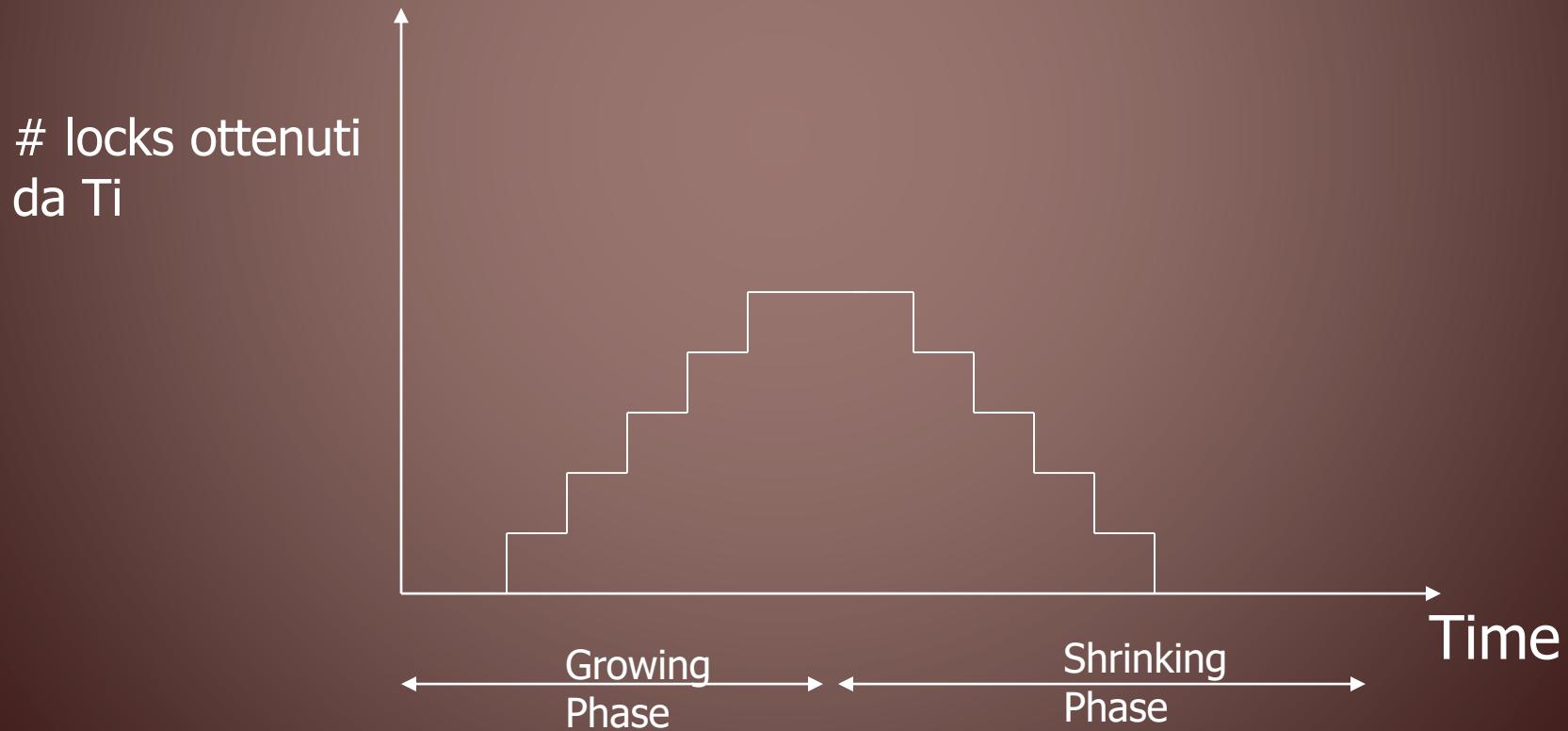
- L'algoritmo 2PL si estende al caso distribuito
- Due strategie:
 - Centralized (or Primary Site) 2PL [Ozsuv-Vald 11.3.1] basato sui siti
 - Primary copy 2PL [Ozsuv-Vald 11.3.2] basato sulle copie

Two-Phase Locking

$T_i = \dots \cdot l_i(A) \cdot \dots \cdot u_i(A) \cdot \dots$

no unlocks

no locks



Strategia centralized 2PL

- Attori
 - Ogni nodo ha un Lock Manager, uno viene eletto LM coordinatore
 - Gestisce i locks per l'intero DDB
 - Il Transaction Manager del nodo dove inizia la transazione e' considerato TM coordinatore
 - La transazione e' anche eseguita su altri Data Processor e corrispondenti nodi

Strategia centralized 2PL

- Strategia
 - Il TM coordinatore formula al LM coordinatore le richieste di lock
 - Il LM le concede, utilizzando un 2PL
 - Il TM le comunica ai DP
 - I DP comunicano al TM e il TM al LM la fine delle operazioni

Centralized 2PL

DP dei siti
partecipanti

TM
Coordinatore

LM centrale



- **Problema: il nodo dell'unico lock manager diventa un collo di bottiglia**

Strategia Primary copy 2PL

- Per ogni risorsa e' individuata una **copia primaria**
- La copia primaria e' individuata prima della assegnazione dei lock
- Diversi nodi hanno lock managers attivi, ognuno gestisce una partizione dei lock complessivi, relativi alle risorse primarie residenti nel nodo
- Per ogni risorsa nella transazione, il TM comunica le richieste di lock al **LM responsabile della copia primaria, che assegna i lock**
- **Conseguenze**
 - Evita il bottleneck
 - Complicazione: e' necessario determinare a priori il lock manager che gestisce ciascuna risorsa.
 - E' necessaria una directory globale

Gestione dei deadlock

Deadlock distribuito

- Causato da un' attesa circolare tra due o piu' nodi
- Gestito comunemente nei DDBMS tramite time-out
- Vediamo un' algoritmo di rilevazione del deadlock in ambiente distribuito
- L'algoritmo e' **asincrono e distribuito**
- Notazione
- t_{ij} sottotransazione della transazione t_i al nodo j

Deadlock distribuito - definizione

- Assumiamo che le sotto-transazioni siano attivate in modo sincrono (tramite Remote Procedure Call bloccante): t_1 **attende** t_2
- Questo puo' dare origine a due tipi di attesa:
 - *ATTESA DA REMOTE PROCEDURE CALL*
 - t_{11} sul nodo 1 (secondo pedice) attende t_{12} sul nodo 2 perche' aspetta la sua terminazione
 - *ATTESA DA RILASCIO DI RISORSA*
 - t_{11} sul nodo 1 attende t_{21} sullo stesso nodo perche' attende il rilascio di una risorsa
- La composizione dei due tipi di attesa puo' dar luogo a uno stato di deadlock globale

Condizioni di attesa

- E' possibile caratterizzare le condizioni di attesa **su ciascun nodo** tramite condizioni di precedenza
- Notazione
 - EXT_i : external, chiamata da un nodo remoto i
 - $X < Y$: X attende il rilascio di una risorsa da Y (puo' essere EXT)
- La **sequenza di attesa** generale al nodo k e' della forma:
$$\text{EXT} < t_{ik} < t_{jk} < \text{EXT}$$
- Esempio
- Su DBMS1: $\text{EXT2} < t_{21} < t_{11} < \text{EXT2}$
- Su DBMS2: $\text{EXT1} < t_{12} < t_{22} < \text{EXT1}$

Esempio

Nodo 1: $EXT2 < t_{21} < t_{11} < EXT1$ Nodo 2: $EXT1 < t_{12} < t_{22} < EXT2$

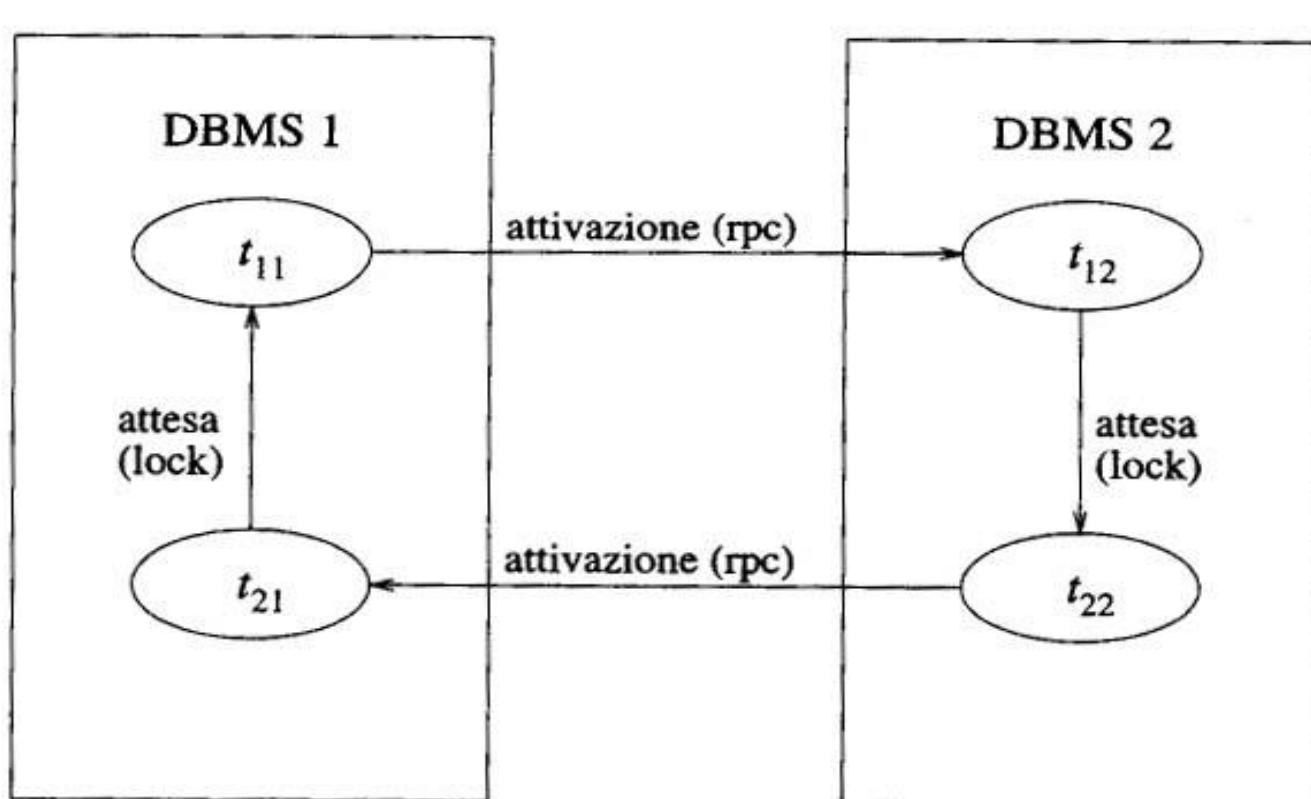


Figura 10.7 Esempio di un deadlock distribuito

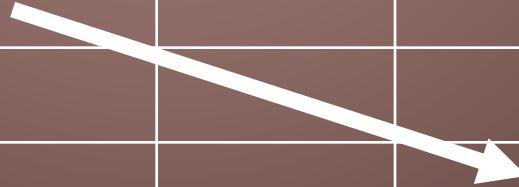
Algoritmo di risoluzione del deadlock distribuito - 1

- Attivato periodicamente sui diversi nodi del DDBMS:
 - 1. In ogni nodo, **integra** la sequenza di attesa con le condizioni di attesa locale degli altri nodi logicamente legati da condizioni EXT
 - 2. **Analizza** le condizioni di attesa sul nodo e **rileva** i deadlock locali
 - 3. **Comunica** le sequenze di attesa ad altre istanze dello stesso algoritmo (cioe' agli altri nodi)

Algoritmo di risoluzione del deadlock distribuito - 2

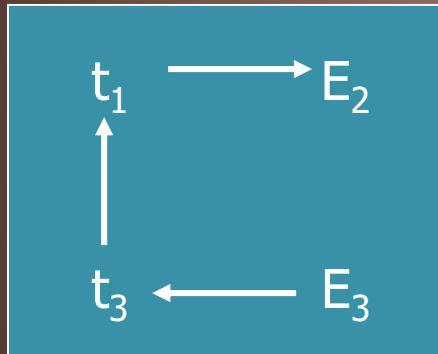
- E' possibile che lo stesso deadlock venga riscoperto piu' volte. Per evitare questo problema, e rendere piu' efficiente l'algoritmo, l'algoritmo invia le sequenze di attesa:
 - **in avanti**, verso il nodo ove e' attiva la sottotransazione t_i attesa da t_j
 - **Solamente quando $i > j$** dove i e j sono gli identificatori dei nodi

	Nodo 1	Nodo2	Nodo3
T1			
T2			
T3			

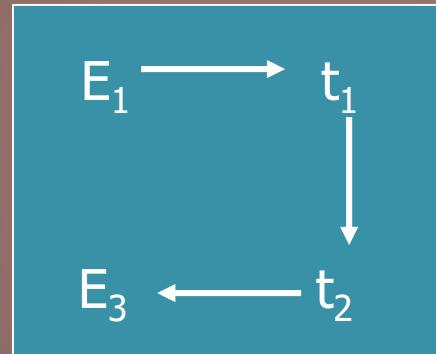


Esempio: situazione di partenza

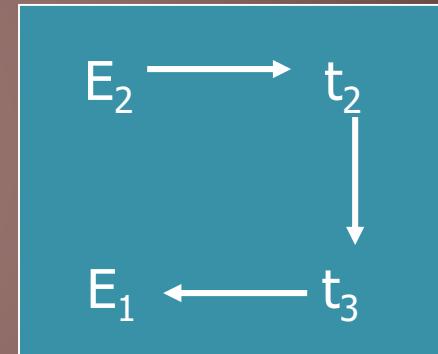
DBMS1



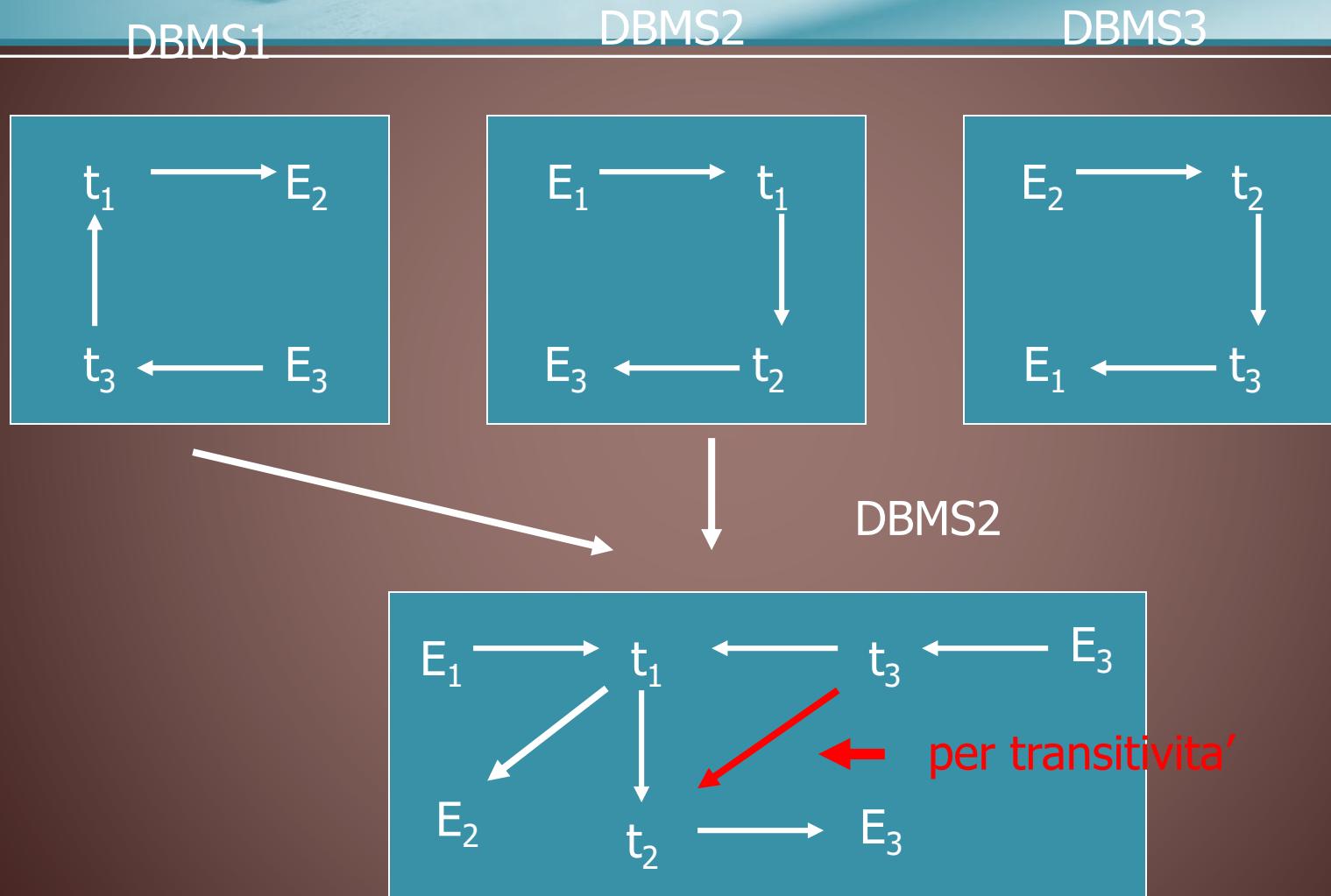
DBMS2



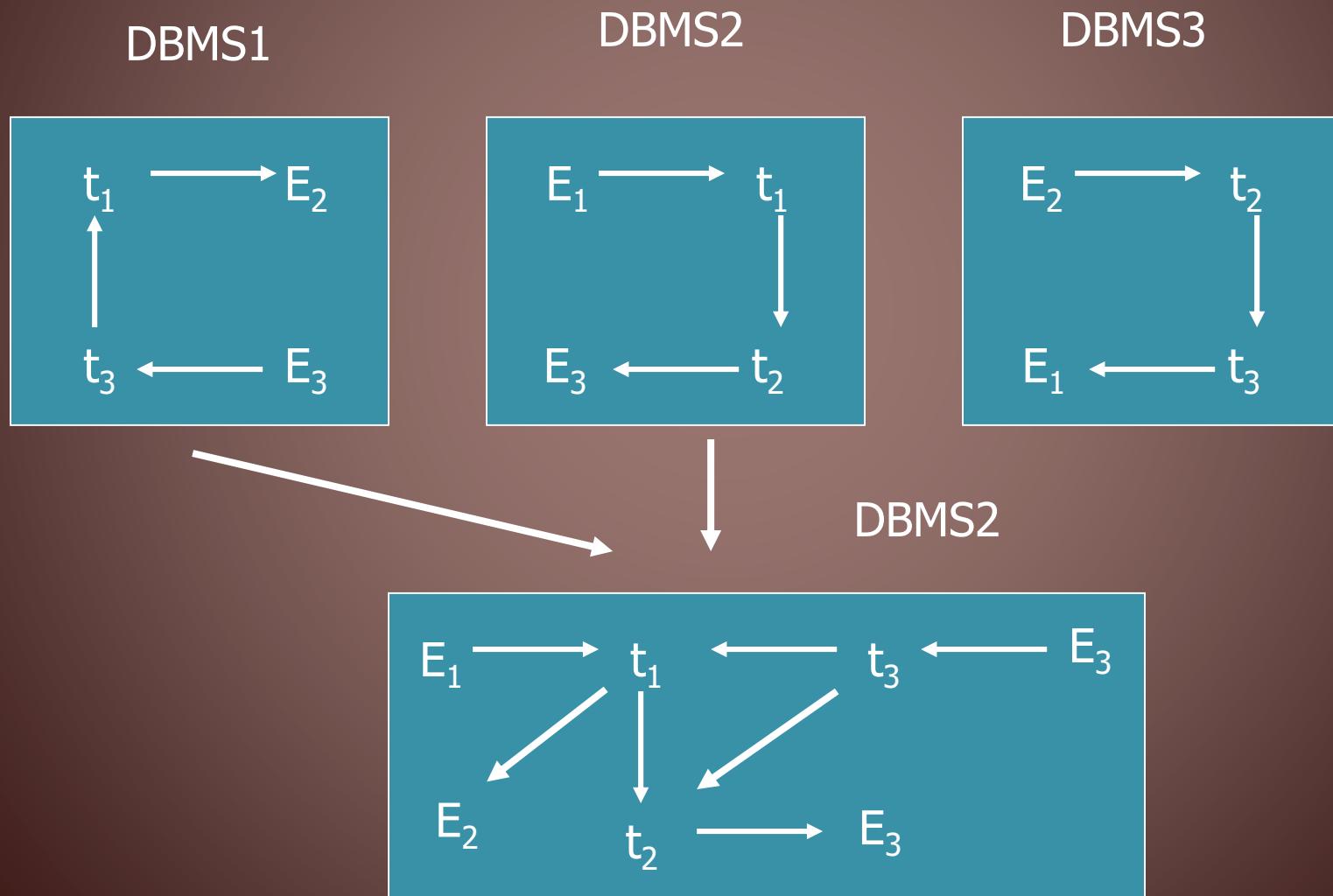
DBMS3



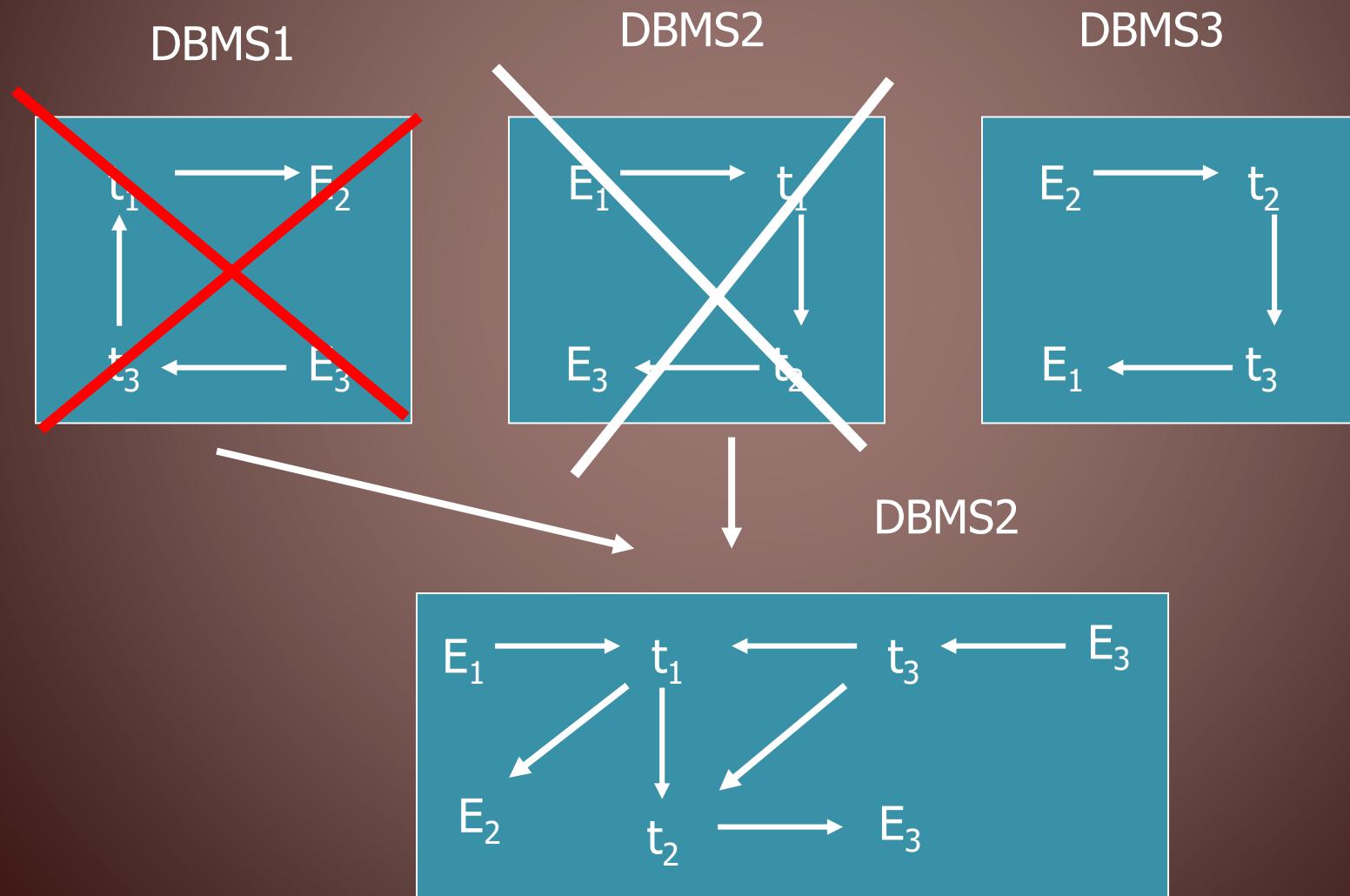
Esempio: primo passo



Esempio: primo passo



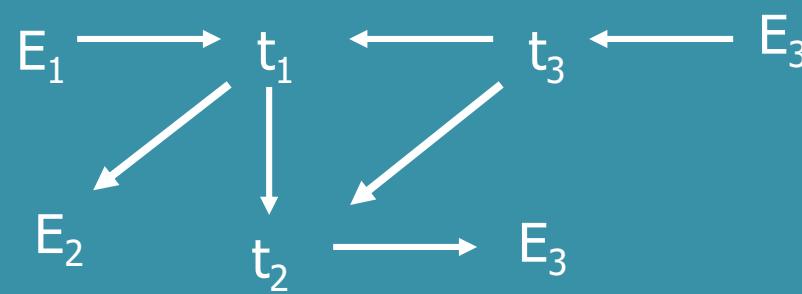
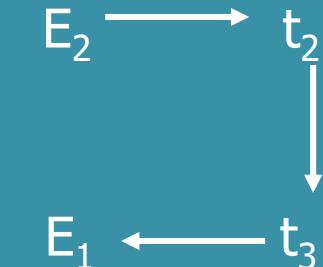
Esempio: primo passo



Esempio: primo passo

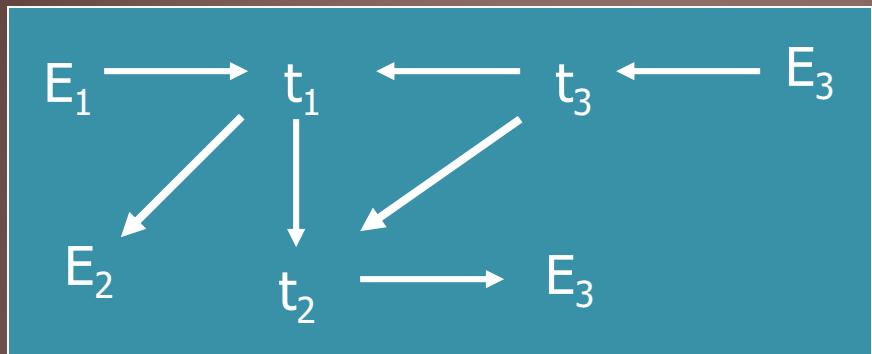
DBMS2

DBMS3

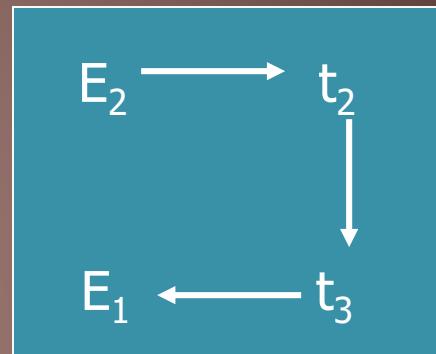


Esempio: secondo passo

DBMS2

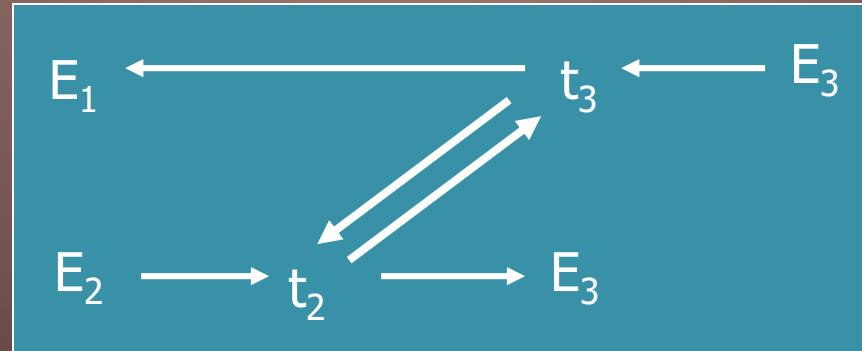


DBMS3



DBMS3

Deadlock!



Recovery management (Atomicità)

Gestione dei guasti nei DDBMS

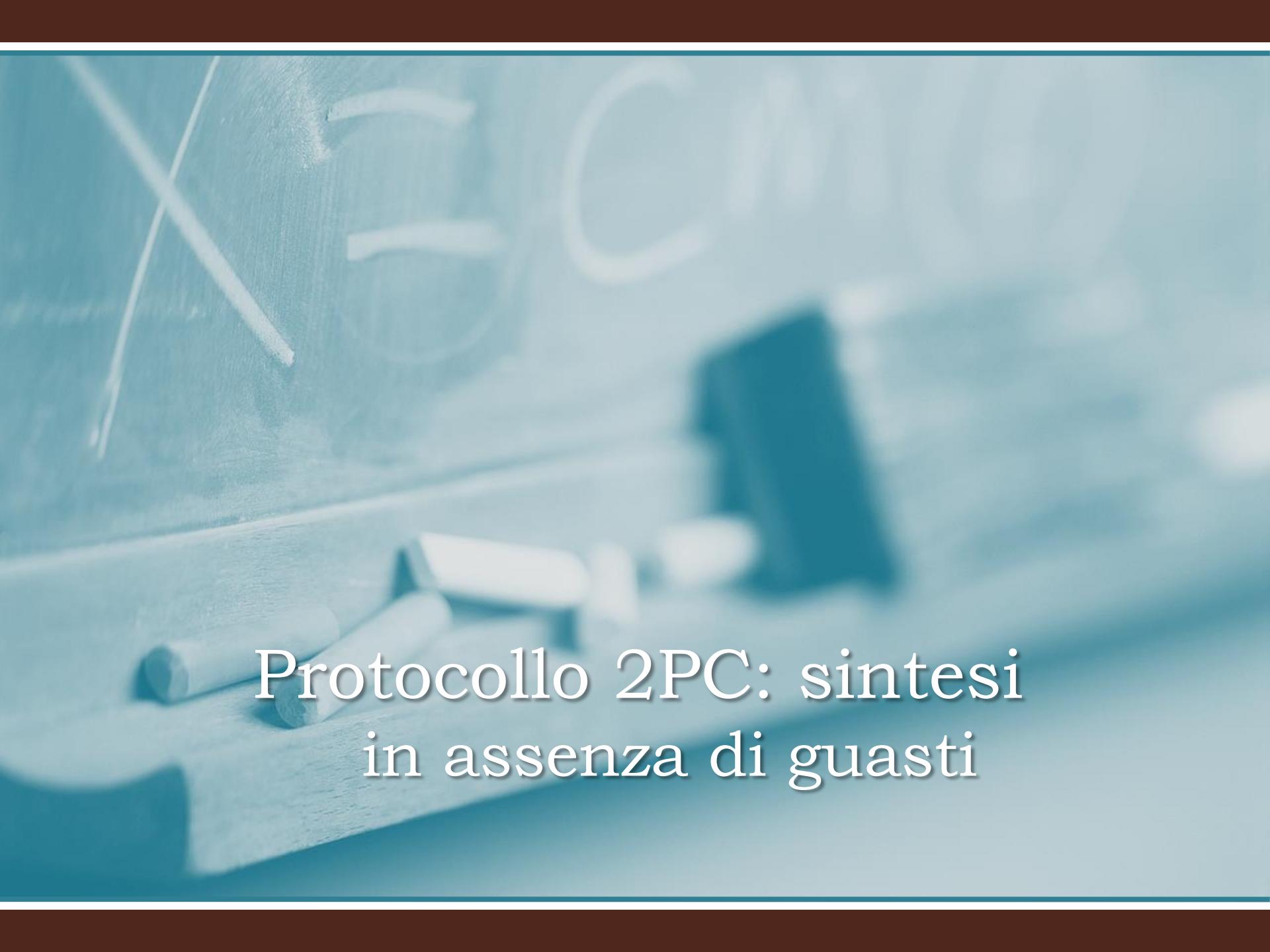
- Un sistema distribuito e' soggetto oltre a guasti locali, anche a perdita di messaggi su rete e partizionamento della rete (isolamento dei nodi)
- Tipi di guasti e conseguenze
 - **Guasti nei nodi:** possono essere soft o hard – già discussi
 - **Perdita di messaggi:** lasciano l'esecuzione di un protocollo in una situazione di indecisione
 - Ogni messaggio del protocollo e' seguito da un acknowledgment
 - In caso di perdita del messaggio oppure dell'ack, si genera una situazione di incertezza: non e' possibile decidere se il messaggio e' stato ricevuto, e quindi se e' arrivata l'informazione
 - **Partizionamento della rete**
 - Una transazione distribuita puo' essere attiva contemporaneamente su piu' sottoreti temporaneamente isolate



Il protocollo two phase commit (2PC)

Protocollo two-phase commit

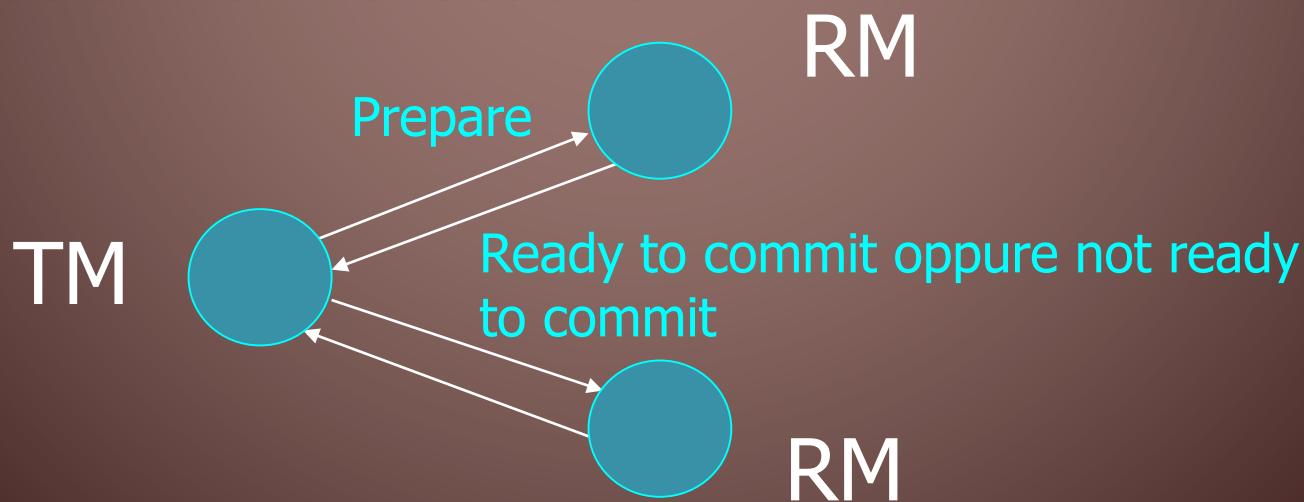
- I protocolli di commit consentono ad una transazione di giungere ad una decisione di abort/commit su ciascuno dei nodi che partecipano ad una transazione
- Come decidere il commit in ambiente distribuito senza o con presenza di guasti?
- Idea: la decisione di commit/abort tra due o piu' partecipanti e' coordinata e certificata da un ulteriore partecipante
 - I server sono chiamati *resource managers* (RM)
 - Il coordinatore e' chiamato *transaction manager* (TM)
- Il protocollo 2PC si basa sullo scambio di messaggi tra TM e RM, i quali mantengono ognuno il proprio log



Protocollo 2PC: sintesi in assenza di guasti

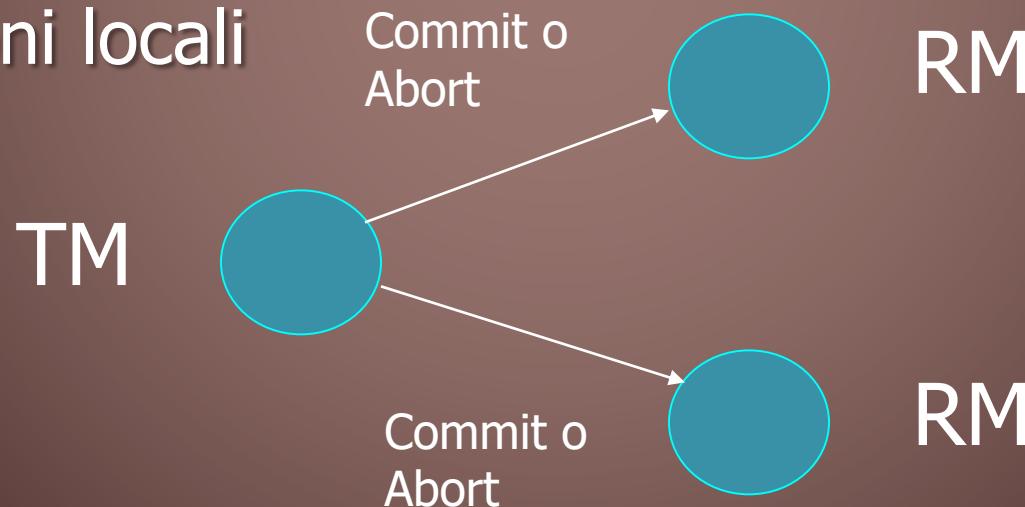
Protocollo 2PC: sintesi

- Prima fase
 - Il TM chiede a tutti i nodi come intendano terminare la transazione
 - Ogni nodo decide autonomamente se commit o abort e comunica unilateralmente la sua decisione irrevocabile



Protocollo 2PC: sintesi

- Seconda fase
 - Il TM prende la decisione globale (se uno solo vuole abort, abort per tutti, altrimenti commit)
 - Il TM comunica a tutti la decisione per le azioni locali



Log nel caso centralizzato

- Nel log compaiono due tipi di record:
 - Record di transazione
 - Informazioni sulle operazioni effettuate
 - Record di sistema
 - Evento di Checkpoint
 - Evento di Dump



b. Protocollo 2PC in assenza di guasti
ma con aggiornamento del log

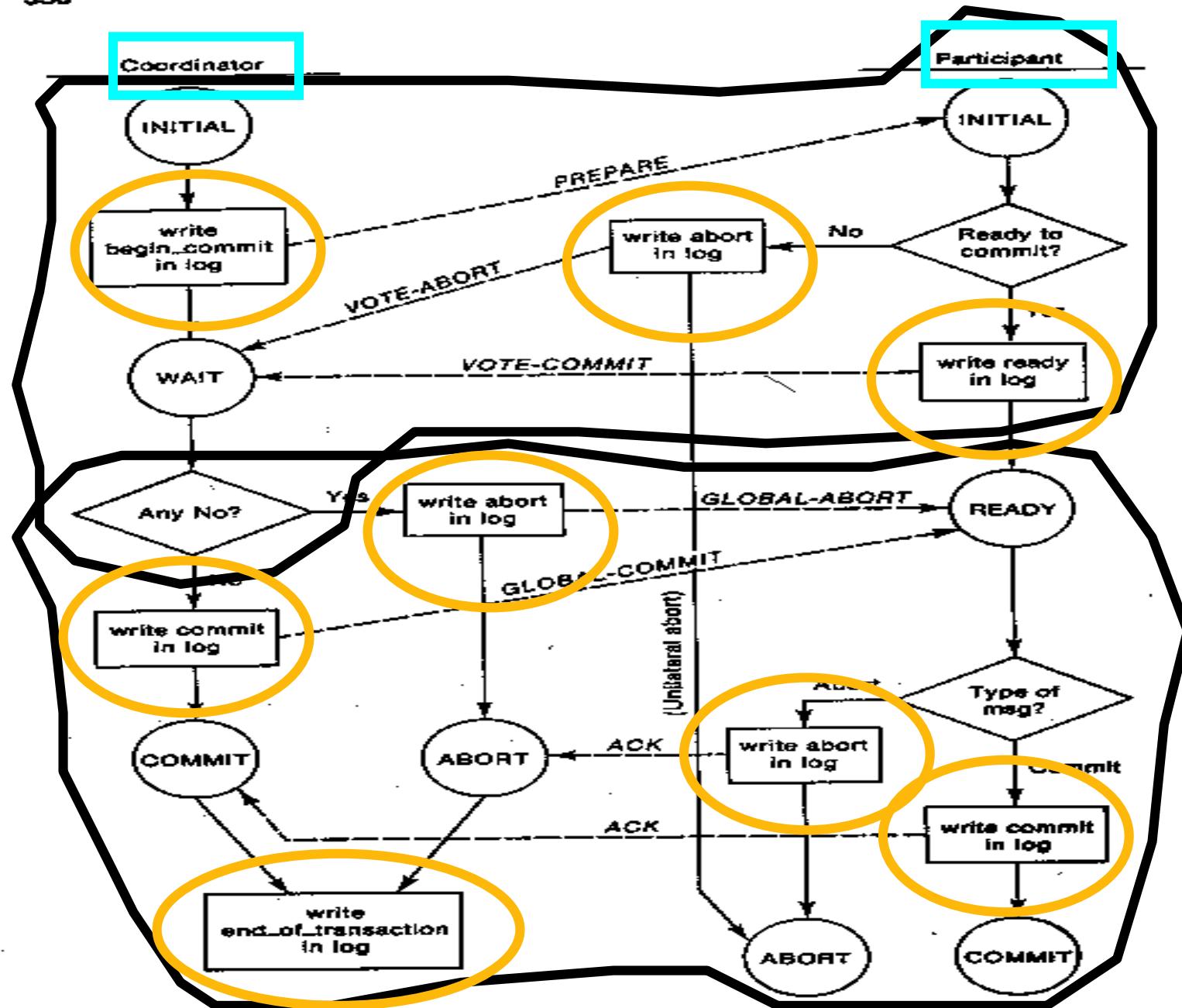


Figure 12.14 2PC Protocol Actions



c. Informazioni da aggiungere ai log

Nuovi record di log -- TM

- Prepare **record** (`begin_commit` **in figura**): contiene l'identita' di tutti i RM (nodi + transazioni)
- `global commit` o `global abort` **record**: descrive la decisione globale. La decisione del TM diventa esecutiva quando il TM scrive nel proprio log il record `global commit` o `global abort`
- Complete (`end of transaction` **in figura**) **record**: scritto alla fine del protocollo

Nuovi record di log -- RM

- **ready record:** disponibilita' irrevocabile del RM a partecipare alla fase di commit
 - Puo' assumere diverse politiche sul protocollo di 2PL: recoverable, 2PL, ACR, strict 2PL.
 - Contiene anche l'identificatore del TM
- Inoltre, come nel caso centralizzato vengono scritti anche i records begin, insert, delete, update, commit
- **not ready (abort in figura) record:** indisponibilita' del RM al commit

Gestione dei timeout

- Sia nella prima fase che nella seconda fase possono avvenire guasti.
- In entrambe le fasi tutti i partecipanti devono poter prendere delle decisioni connesse allo stato in cui si trovano
- Questo avviene utilizzando timers e stabilendo un intervallo di tempo di timeout

messaggi

 Azioni sul log

 Decisioni

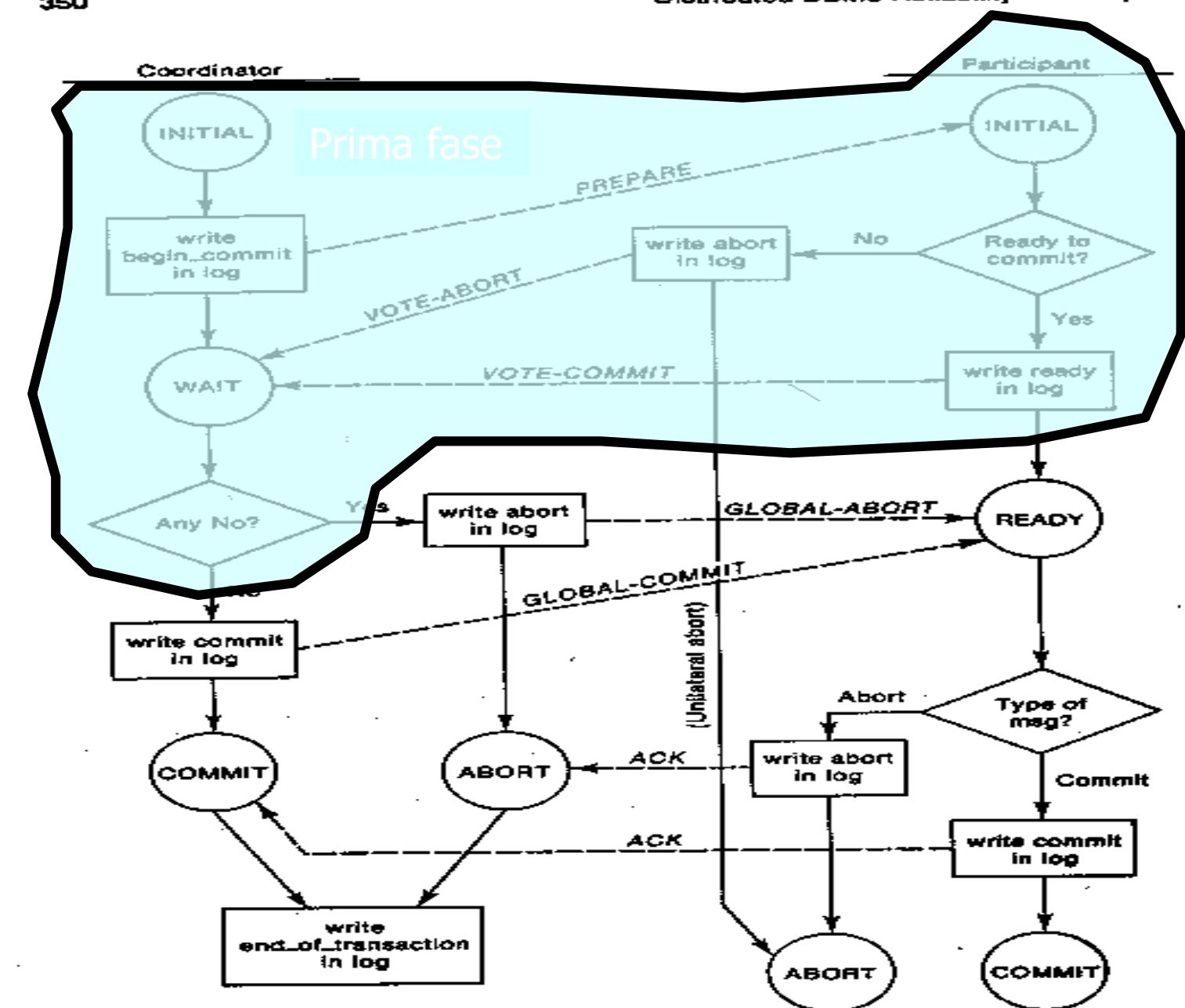


Figure 12.14 2PC Protocol Actions

Prima fase di 2PC

- TM scrive `prepare` nel suo log e invia un messaggio `prepare` a tutti i RM. Fissa un timeout per indicare il massimo intervallo di tempo di attesa per le risposte
- Gli RM che sono recoverable, cioe' sono pronti a fare il commit, scrivono `ready` nel loro log record e inviano un messaggio `ready` al TM
- Gli RM che *non* sono recoverable, perche', ad esempio devono abortire per un deadlock, inviano un messaggio `not-ready` e terminano il protocollo, effettuando un abort unilaterale

Prima fase di 2PC - 2

- Il TM raccoglie i messaggi di risposta dagli RM:
 - Se *tutti* gli RM rispondono positivamente, scrive global commit nel suo log
 - Se riceve almeno un messaggio not-ready o scatta il timeout, scrive global abort nel suo log

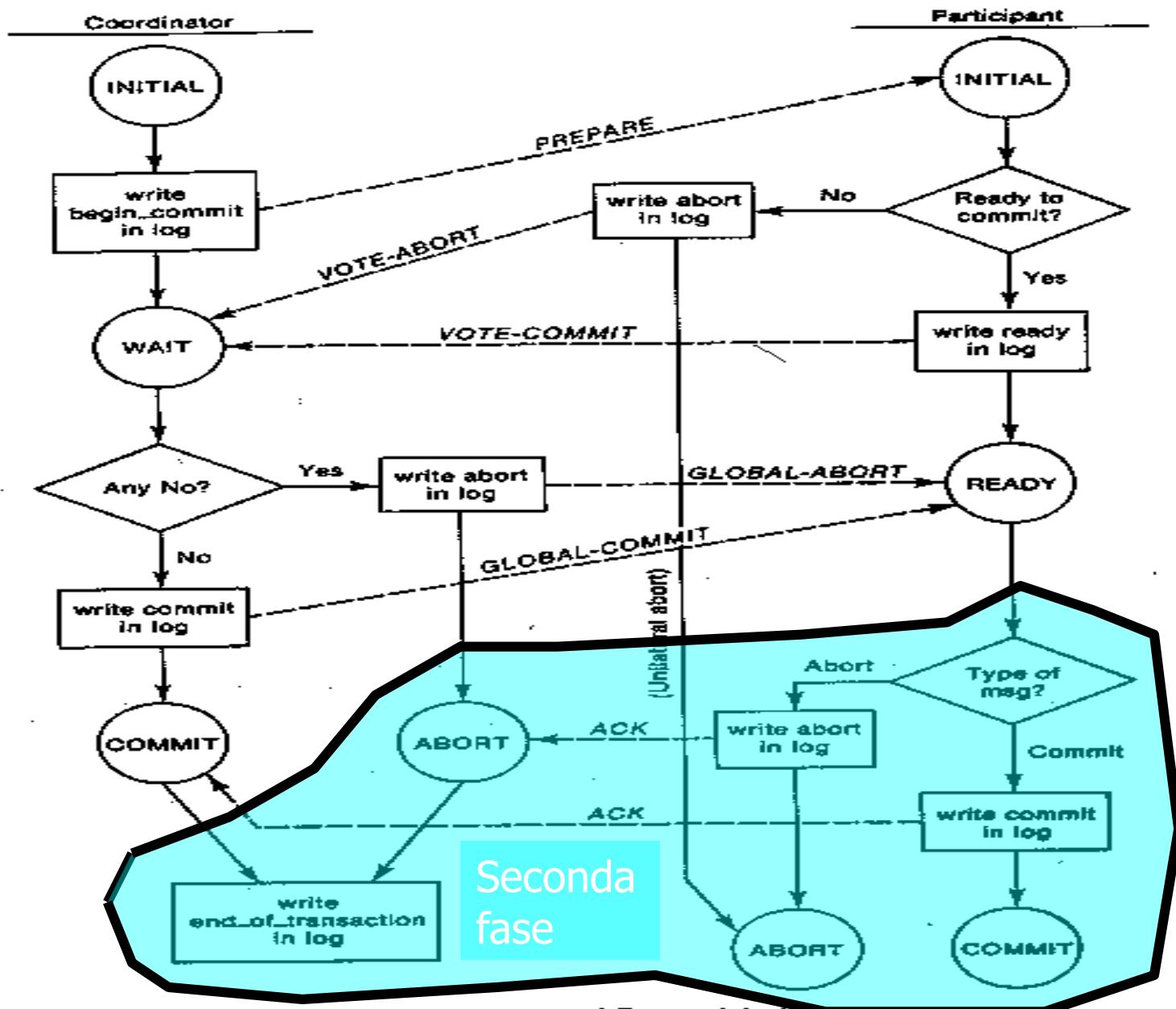


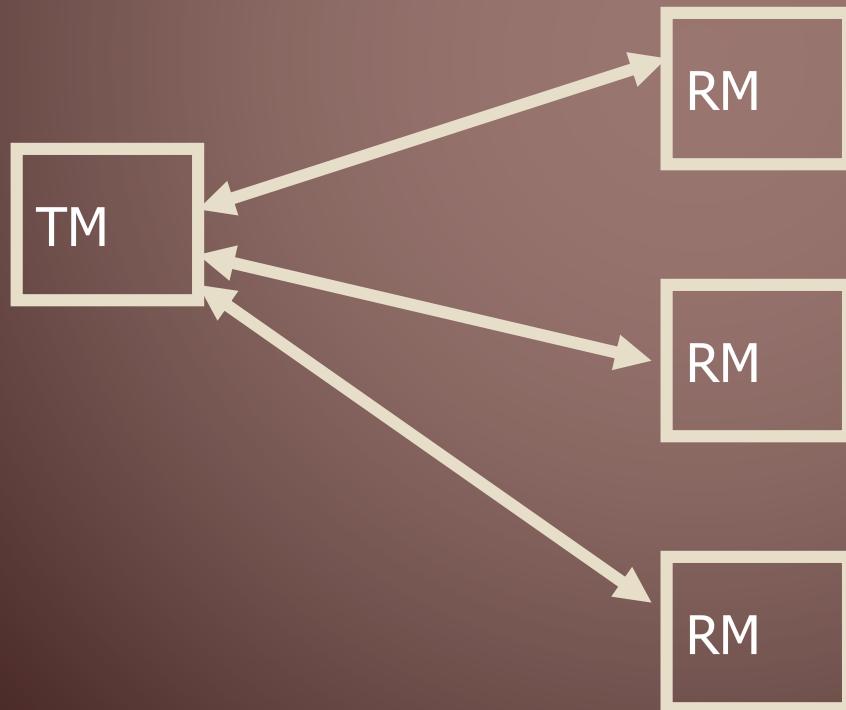
Figure 12.14 2PC Protocol Actions

Seconda fase di 2PC

- Il TM trasmette la decisione globale agli RM e **fissa un nuovo timeout**
- Gli RMs che sono **ready** ricevono il messaggio, scrivono **commit** o **abort** nel loro log, e inviano un **acknowledgment** al TM. Poi eseguono il loro **commit** o **abort** locale
- Il TM raccoglie tutti i messaggi di *acknowledgement* dagli RM. **Se scatta il time-out, poiche' ha già deciso in modo irrevocabile, fissa un nuovo time-out e ripete la trasmissione a tutti i RMs dai quali non ha ancora ricevuto un ack**
- Quando tutti gli *acknowledgement* sono arrivati, il TM **scrive complete** nel suo log

Paradigmi di comunicazione tra TM e RMs

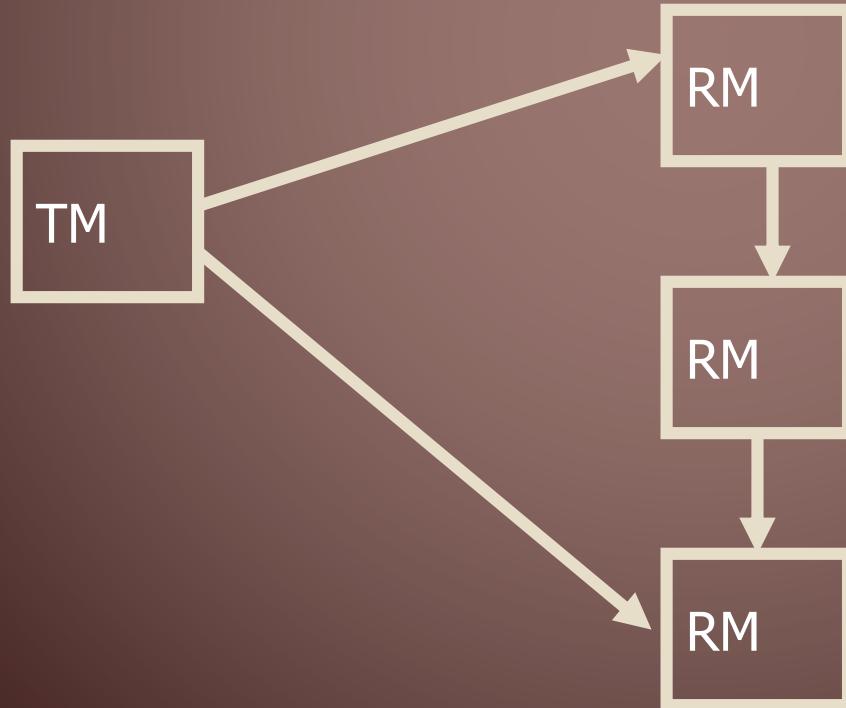
- **Centralizzato (e' quello che abbiamo visto)**
 - La comunicazione avviene solo tra TM e ogni RM
 - Nessuna comunicazione tra RM



Paradigmi di comunicazione tra TM e RMs

- **Lineare:**

- Gli RM comunicano tra loro secondo un ordine prestabilito
- Il TM e' il primo nell'ordine
- Utile solo per reti senza possibilita' di broadcast



Paradigmi di comunicazione tra TM e RMs

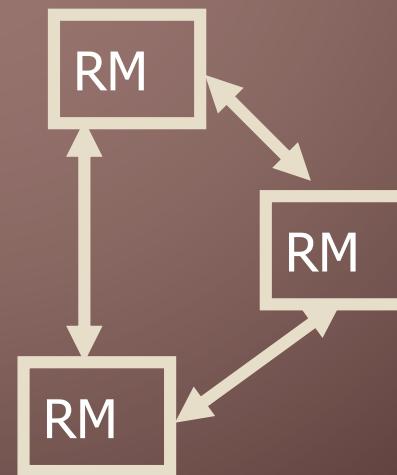
- **Distribuito:**
 - Nella prima fase, il TM comunica con gli RMs



Paradigmi di comunicazione tra TM e RMs

- **Distribuito:**

- Nella prima fase, il TM comunica con gli RMs
- Gli RMs inviano le loro decisioni a tutti gli altri partecipanti
- Ogni RM decide in base ai voti che “ascolta” dagli altri
- Non occorre la seconda fase di 2PC





d. 2PC in caso di guasto

2PC in caso di guasto - 1

- Un RM nello stato ready perde la sua autonomia e attende la decisione del TM
- Un guasto nel TM lascia l' RM in uno stato di incertezza
- Le risorse allocate alla transazione restano bloccate
- L'intervallo tra la scrittura di ready nel log dei RMs e la scrittura di commit o abort è detta *finestra di incertezza*

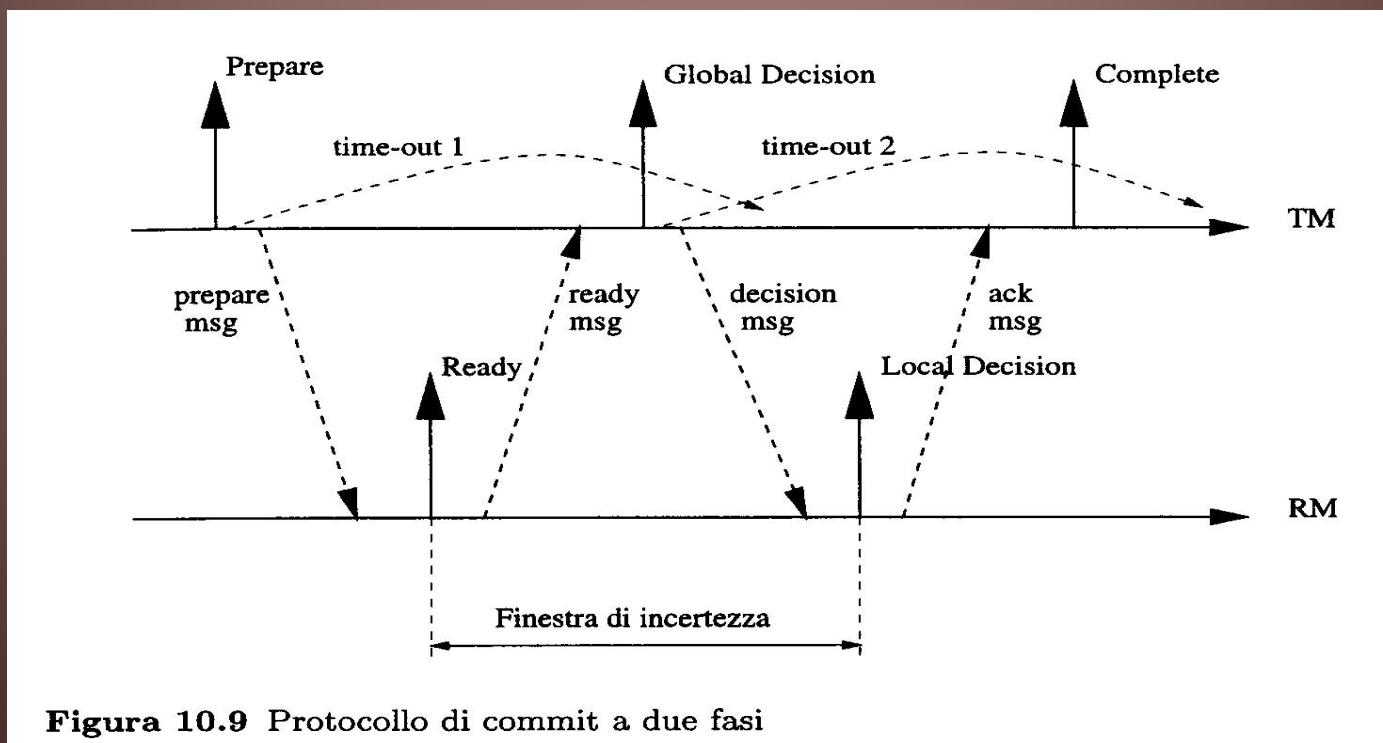


Figura 10.9 Protocollo di commit a due fasi

2PC in caso di guasto - 2

- In questo intervallo tutte le risorse del sistema acquisite tramite meccanismi di lock sono bloccate
 - 2PC riduce al minimo questo intervallo di tempo, che esiste comunque
- In seguito a guasti, TM o RMs utilizzano protocolli di recovery

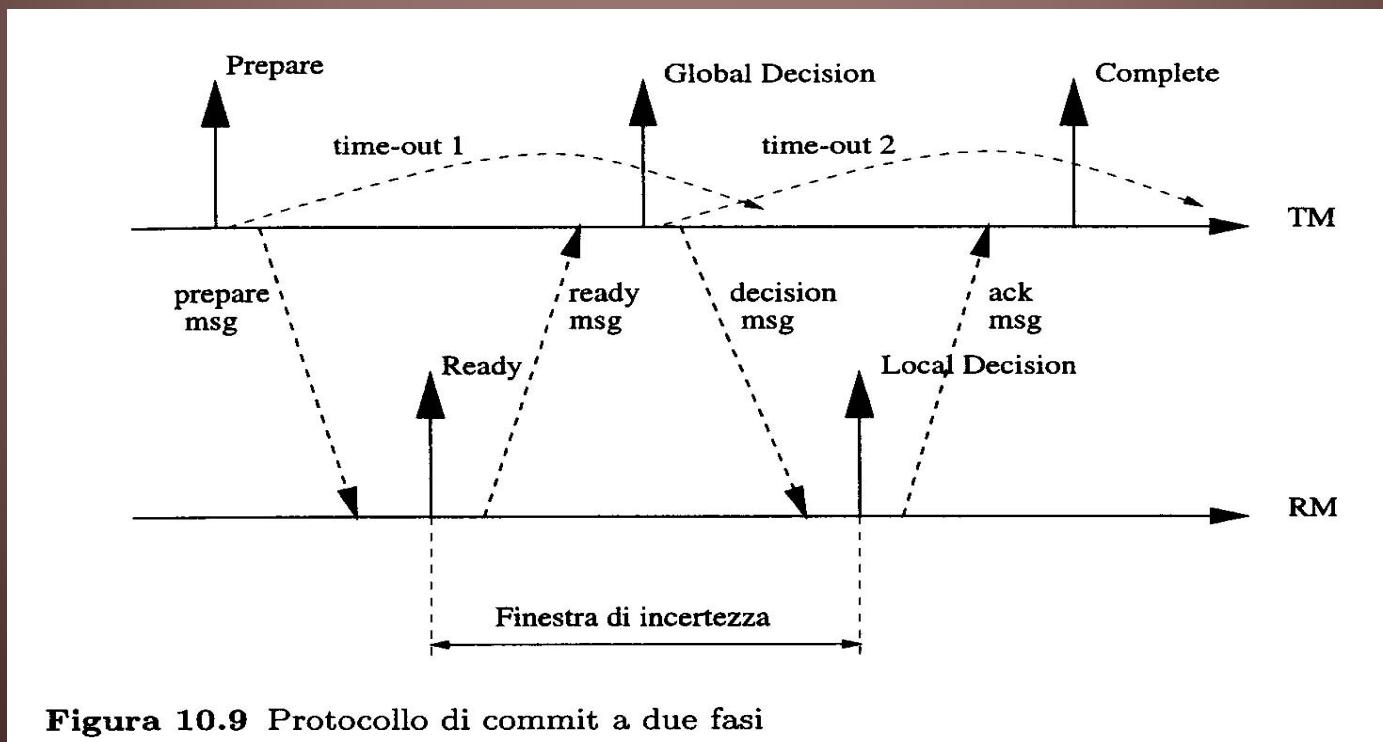


Figura 10.9 Protocollo di commit a due fasi

Tipi di guasti da governare con protocolli

- 1. GC Guasti di componenti
 - 1.1. TM Transaction manager
 - 1.2. RM Resource manager
- 2. PM Perdita di messaggi e Partizionamento della rete

1. Guasti di componenti

- Devono essere utilizzati protocolli con due diversi compiti:
- Assicurare la terminazione della procedure
- → PT. protocolli di terminazione
- Assicurare il ripristino
- → PR. protocolli di recovery
- Si puo' dimostrare che i protocolli funzionano nella ipotesi di guasto di un solo partecipante
- Faremo questa ipotesi



Guasti di un componente
Protocolli di terminazione
guasto di un resource manager

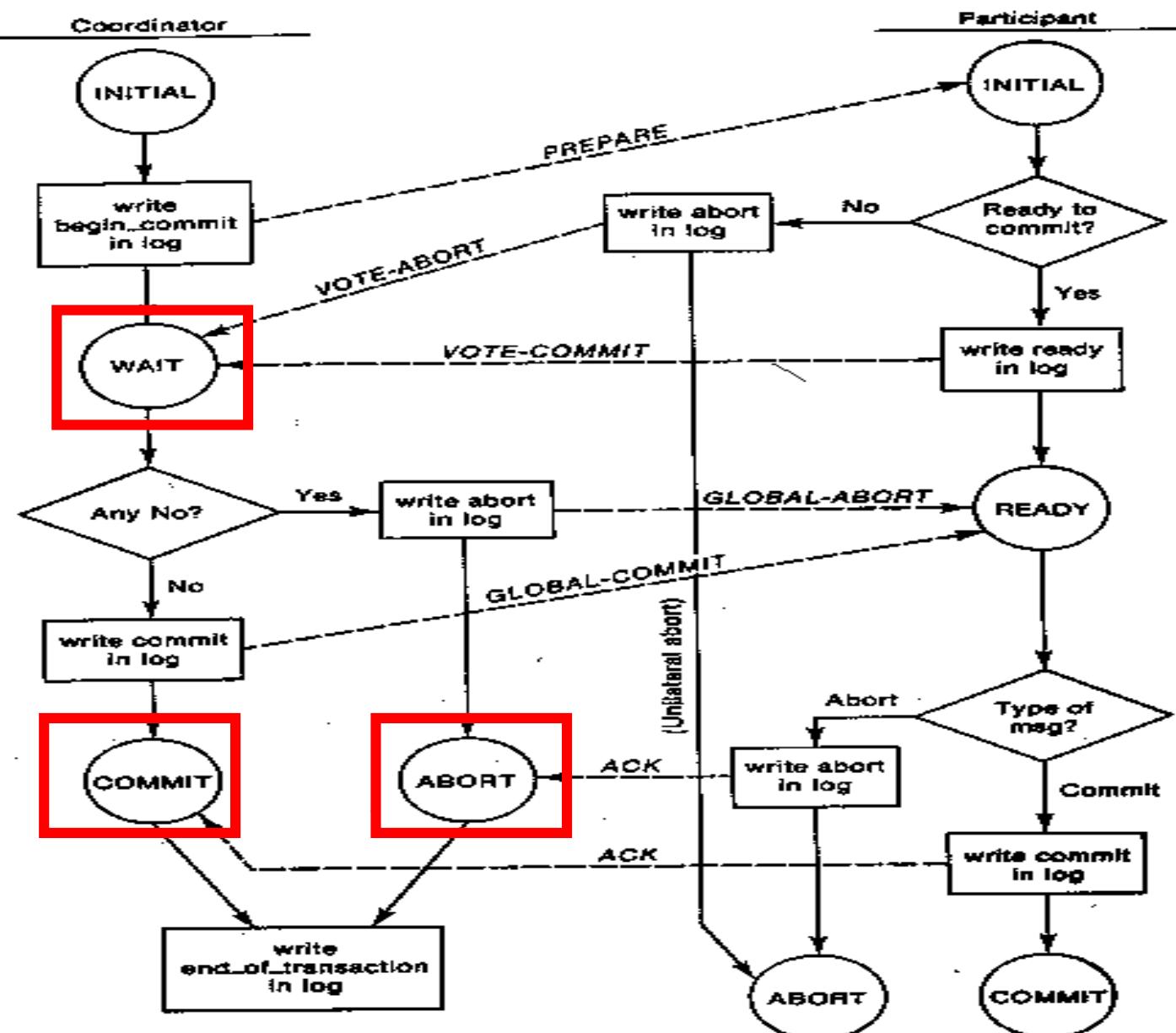
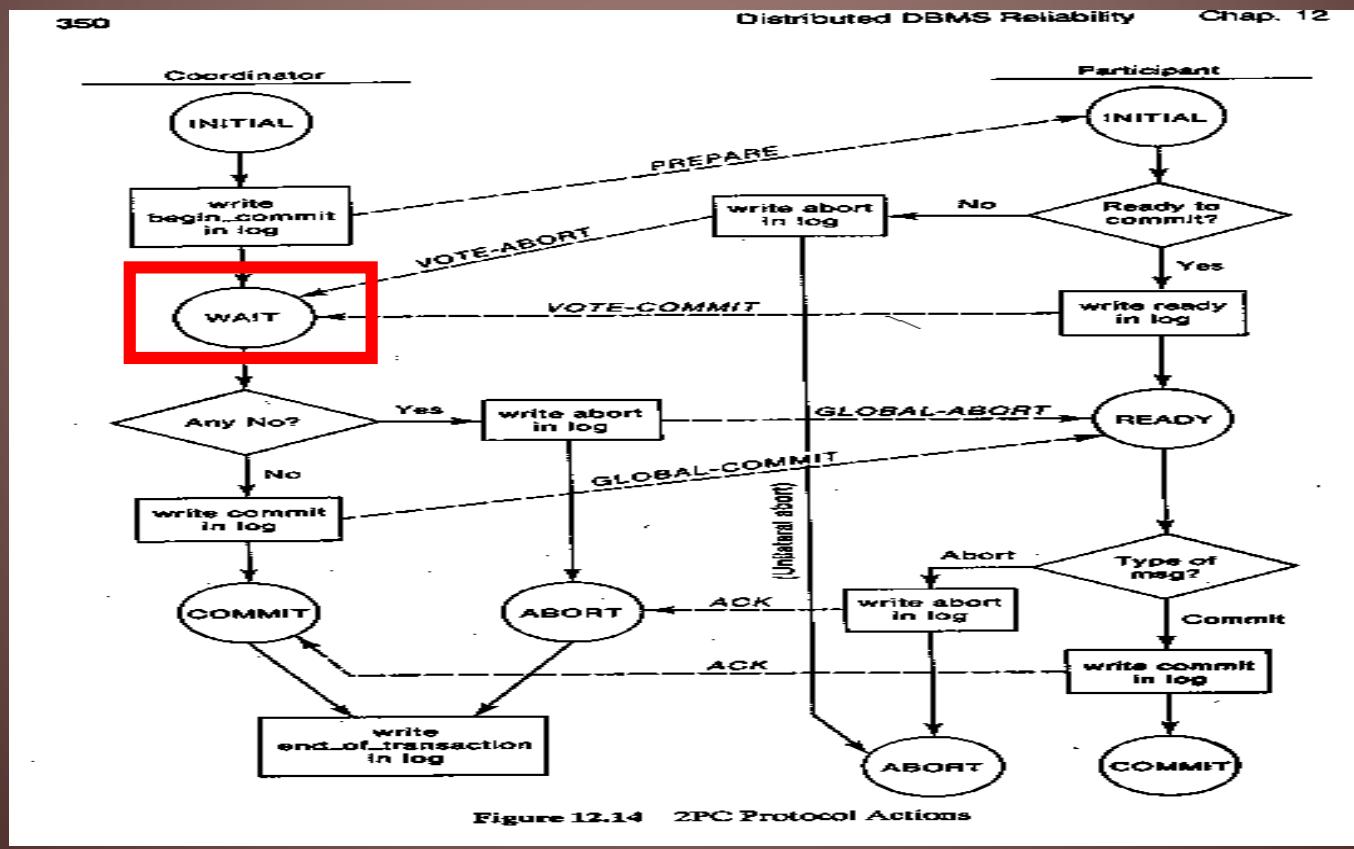


Figure 12.14 2PC Protocol Actions

Timeout del TM

- **Timeout nello stato WAIT:**

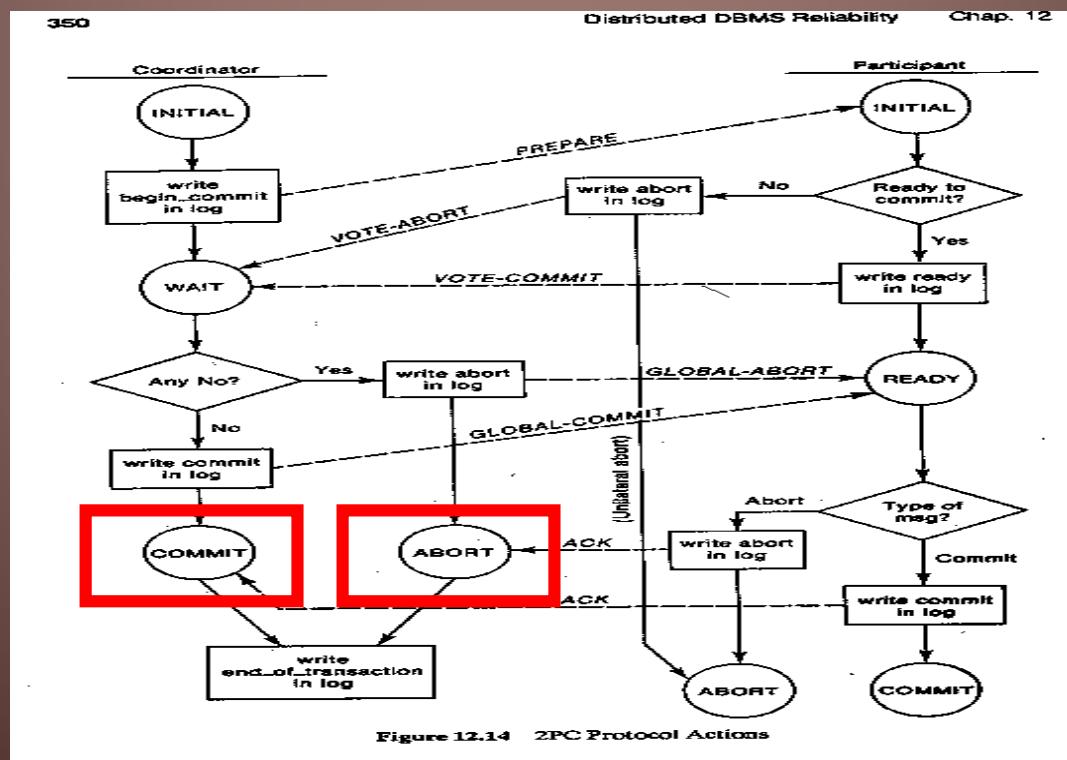
- Il TM attende la risposta dei RMs
- Puo' solo decidere global-abort



Timeout del TM

- **Timeout nello stato COMMIT o ABORT:**

- Il TM non puo' sapere se le procedure di commit/abort sono state completate dai recovery manager di ogni nodo, ma e' obbligato a mantenere la decisione globale presa.
- Il TM deve continuare a inviare lo stesso messaggio global-commit oppure global-abort e ad aspettare il consenso
- Quando l'RM con guasto riprende, manda un messaggio di consenso al TM





Guasti di un componente
Protocolli di terminazione
guasto del transaction manager

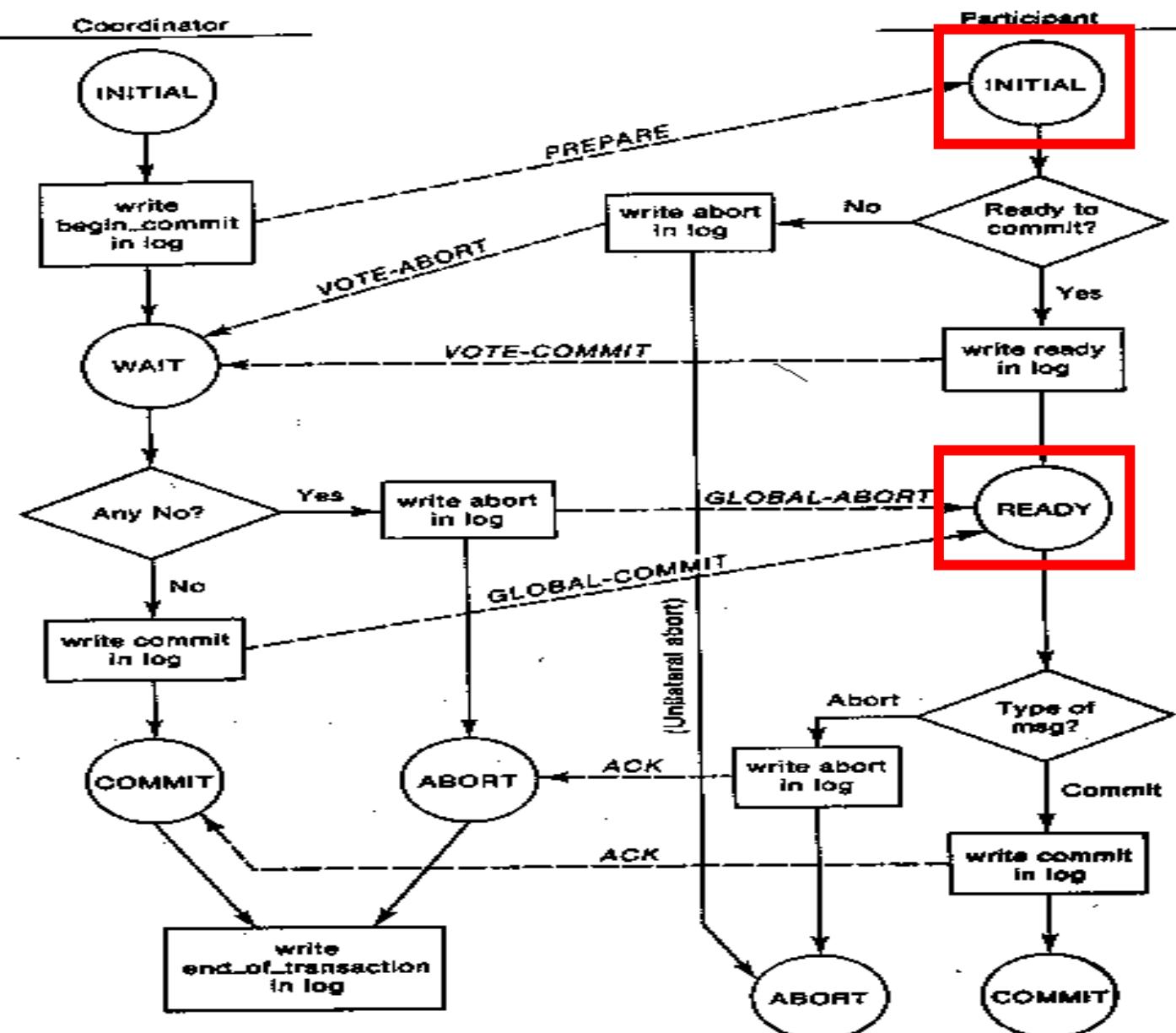


Figure 12.14 2PC Protocol Actions

Timeout dei RM

- Facciamo l'ipotesi che anche gli RM possano attivare un timeout
- Timeout nello stato INITIAL:

- Il RM attende il messaggio *prepare*
- Il TM deve essere caduto nello stato INITIAL
- Il RM puo' fare un abort unilaterale

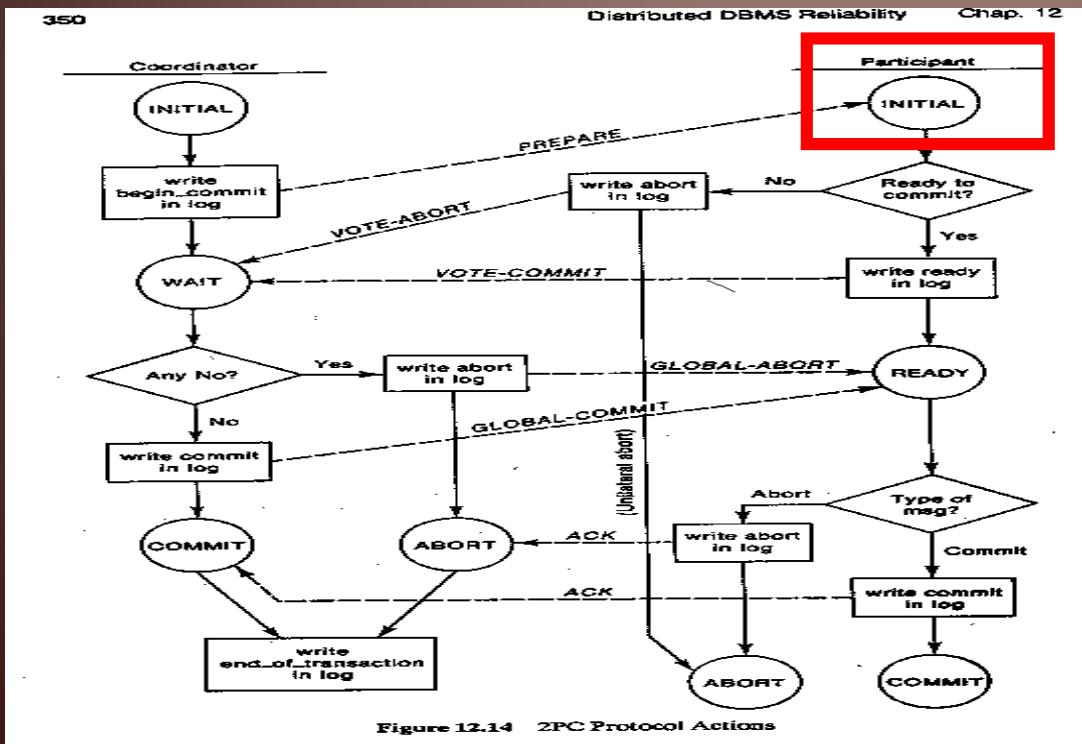
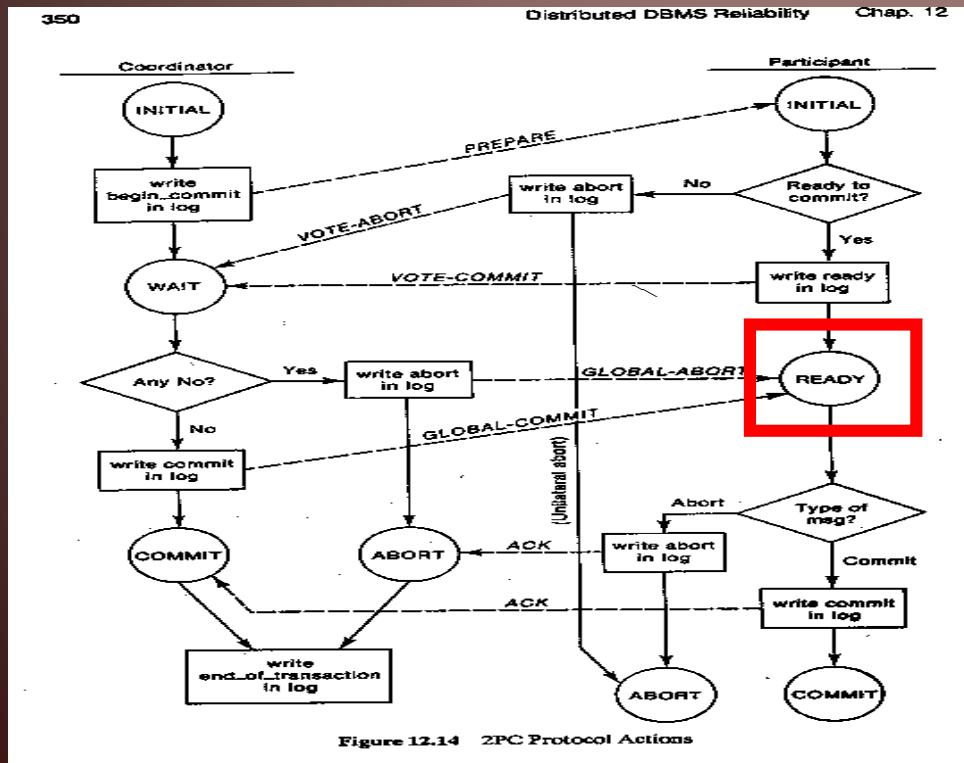


Figure 12.14 2PC Protocol Actions

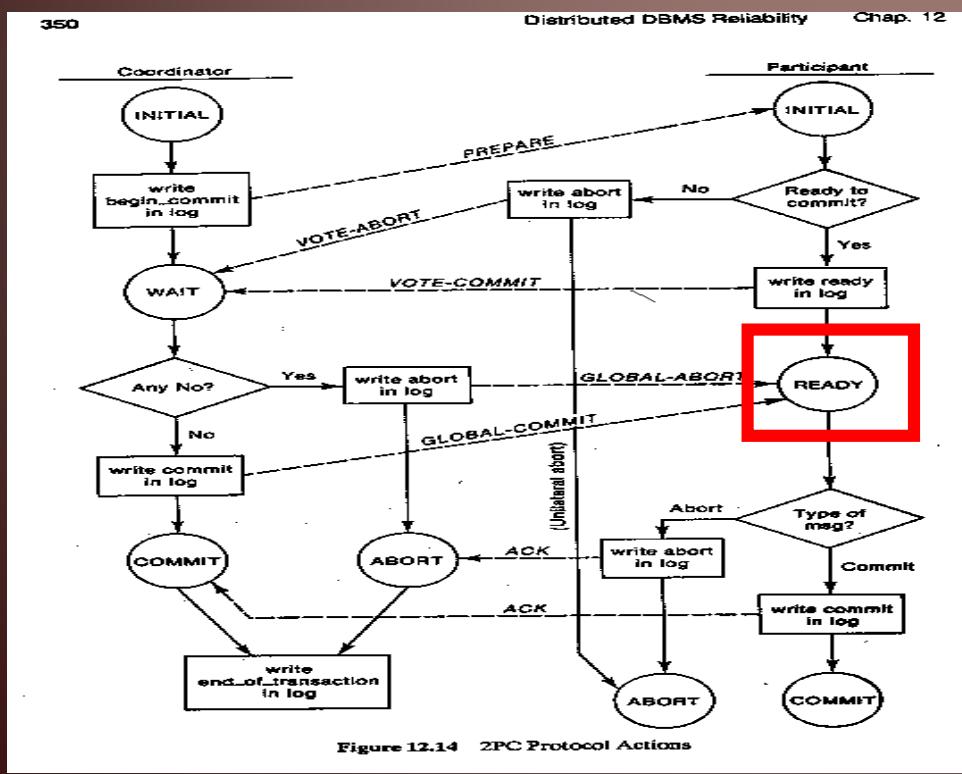
Timeout dei RM

- Timeout nello stato READY:
 - In questo stato, l' RM ha votato per il commit;
 - Attende la decisione del TM
 - Non e' in grado di prendere una decisione unilaterale
 - Resta bloccato in attesa di ulteriori informazioni
 - Puo' unilateralmente decidere di abortire.
 - In questo caso, se e' caduto il TM, quando riprende vota abort



Timeout dei RM – Stato READY

- Timeout nello stato READY:
- Se gli RMs sono in grado di comunicare
 - tra loro, e' possibile sbloccare la situazione
 - in assenza del TM chiedendo agli altri RMs di aiutarlo a prendere una decisione
 - [omessi dettagli sui casi possibili...]





2PR/TM Protocolli di ripristino guasto del Transaction manager

Caduta del coordinatore (TM)

- Casi possibili
- 1. l'ultimo record del log e' prepare
 - Il guasto del TM puo' avere bloccato alcuni RM
 - Due opzioni di recovery:
 - Decidere global abort, e procedere con la seconda fase di 2PC
 - Ripetere la prima fase, sperando di giungere a un global commit
- 2. l'ultimo record nel log e' global-commit o global-abort
 - alcuni RMs potrebbero non essere stati informati, e altri possono essere bloccati
 - Il TM deve ripetere la seconda fase
- 3. l'ultimo record nel log e' una complete
 - la caduta del coordinatore non ha effetto



2PR/RM

Protocolli di ripristino guasto di un Resource manager

Caduta di un partecipante (RM)

- Casi possibili
- 1. l'ultimo record nel log e' di azione, di Abort o Commit → come nel caso centralizzato
 - Usa la sequenza di warm restart
 - Abort o azione: undo della transazione
 - Commit: redo della transazione
- 2. l'ultimo record nel log e' Ready → caso nuovo
 - il RM si blocca perche' non conosce la decisione del TM
 - Durante il warm restart, gli ID delle transazioni in dubbio sono inserite nel *ready* set. Due possibilita':
 - Prima soluzione: il partecipante chiede al coordinatore cosa e' accaduto (richiesta di remote recovery)
 - Seconda soluzione: il coordinatore riesegue la seconda fase del protocollo



PM. Perdita di messaggi e partizionamento della rete

Perdita di messaggi e partizionamento della rete

- Il TM non e' in grado di distinguere tra perdita di messaggi prepare o ready nella prima fase
 - In entrambi i casi, la decisione globale e' abort in seguito a timeout nella prima fase
- Anche la perdita di messaggi di acknowledgement o di decisioni da parte dei RMs non sono distinguibili
 - In entrambi i casi, la seconda fase viene ripetuta in seguito a timeout
- Un partizionamento della rete non causa problemi ulteriori, dato che una transazione puo' avere successo solo se il TM e tutti i RM appartengono alla stessa partizione



e. Ottimizzazioni
nel protocollo 2PC

Obiettivi delle ottimizzazioni

- Ridurre il numero di messaggi trasmessi tra coordinatore e partecipanti
- Ridurre il numero di scritture nei log

Ottimizzazione read-only

- Quando un RM sa che la propria transazione non contiene operazioni di scrittura, non deve influenzare l'esito finale della transazione:
 - Risponde `read-only` al messaggio di `prepare` e termina l'esecuzione del protocollo
- Il TM ignora i partecipanti `read-only` nella seconda fase
- I partecipanti per cui e' noto a priori che siano di sola lettura, possono essere esclusi dal protocollo.

Ottimizzazione presumed abort

- Regola “**Scordarsi gli abort, ricordarsi i commit**”
- Il TM abbandona la transazione subito dopo avere deciso per abort, e non scrive global abort nel log
 - → Non aspetta risposta dai RMs
- Percio’: Quando il TM riceve una richiesta di remote recovery da parte di un RM che non sa come procedere in seguito a un guasto, e il TM non trova abort nel log, l’RM puo’ chiedere solo in uno stato in cui non c’e’ stato il commit → Il TM decide sempre per il global abort
- Gli unici record che vengono scritti sono global commit, ready e commit
- Non occorre scrivere i record di prepare e global abort

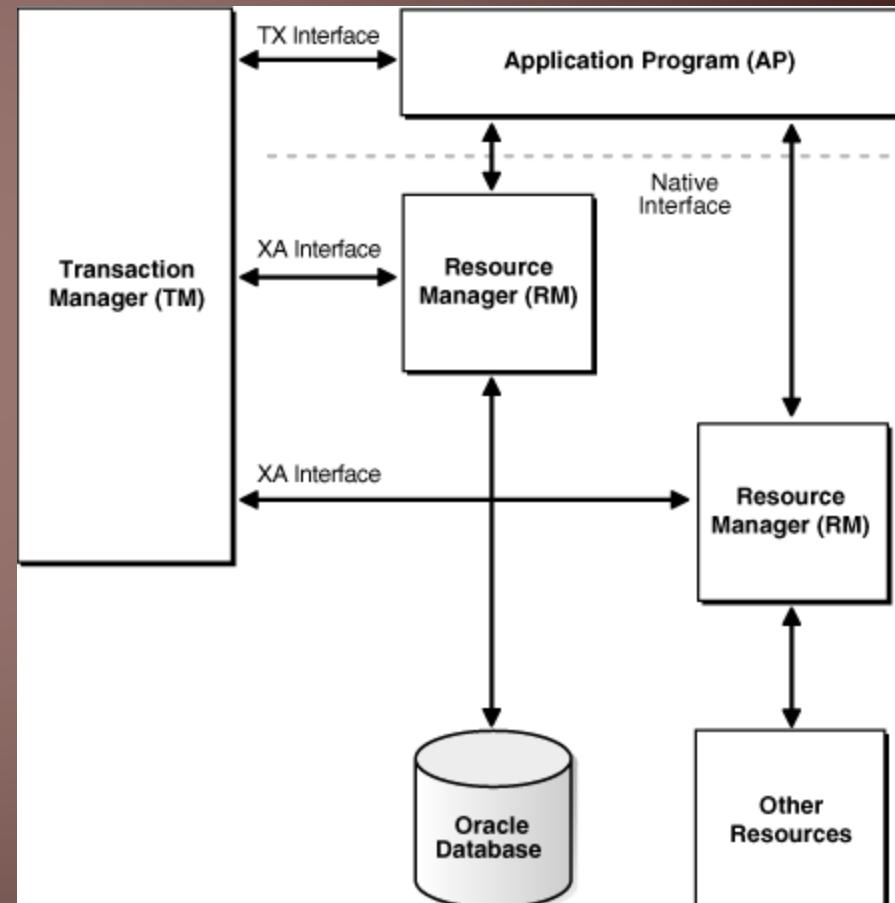
Transazioni distribuite il protocollo xopen DTP

X/Open DTP

- X/Open Ltd. is a consortium of vendors who are defining portability standards for the UNIX environment.
- One standard X/Open has defined is for distributed transaction processing (DTP)
- To define a standard communication architecture through which multiple application programs may share resources while coordinating their work into transactions that may execute concurrently.

Architettura

- The AP, the RM, and the TM communicate via three distinct interfaces: native, TX, and XA.
- The native interface is the medium by which the AP makes requests directly to the RM.
- This interface is RM specific.
 - either Embedded SQL or Client-Library.
- The TX Interface is the medium between the AP and the TM. The AP uses TX calls to delineate transaction boundaries.
 - This interface is TM specific.
- The XA Interface is the medium between the RM and the TM.
- Using XA calls, the TM tells the RM when transactions start, commit, and roll back. The TM also handles recovery.



XA

XA Subroutine	Description
<code>xa_open()</code>	Connects to the RM.
<code>xa_close()</code>	Disconnects from the RM.
<code>xa_start()</code>	Starts a new transaction and associates it with the given transaction ID (XID), or associates the process with an existing transaction.
<code>xa_end()</code>	Disassociates the process from the given XID.
<code>xa_rollback()</code>	Rolls back the transaction associated with the given XID.
<code>xa_prepare()</code>	Prepares the transaction associated with the given XID. This is the first phase of the two-phase commit protocol.
<code>xa_commit()</code>	Commits the transaction associated with the given XID. This is the second phase of the two-phase commit protocol.
<code>xa_recover()</code>	Retrieves a list of prepared, heuristically committed, or heuristically rolled back transactions.
<code>xa_forget()</code>	Forgets the heuristically completed transaction associated with the given XID.