

Cos'è una transazione?

- Una **transazione** è un'unità logica di elaborazione richiesta da un'applicazione che dà luogo a una serie di operazioni fisiche elementari (letture e scritture) sulla base di dati
- Esempi:

- Trasferimento di una somma da un conto corrente ad un altro

```
UPDATE CC
```

```
SET      Saldo = Saldo - 50
```

```
WHERE    Conto = 123
```

```
UPDATE CC
```

```
SET      Saldo = Saldo +
```

```
50
```

```
WHERE    Conto = 235
```

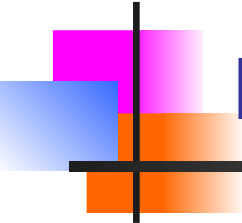
- Aggiornamento degli stipendi degli impiegati di una sede

```
UPDATE Imp
```

```
SET      Stipendio = 1.1*Stipendio
```

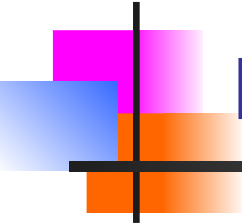
```
WHERE    Sede = 'S01'
```

- In entrambi i casi tutte le operazioni elementari devono essere eseguite



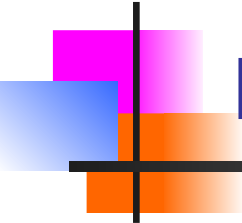
Proprietà ACID di una transazione

- L'acronimo **ACID** sta per
 - **Atomicity** = una transazione è **un'unità di elaborazione**
 - Non si può eseguirne solo una parte, ovvero il DB non deve restare per nessun motivo in uno "stato intermedio"
 - Se per qualche motivo la transazione non può terminare correttamente la sua esecuzione bisogna "disfare" (UNDO) quanto da essa fatto
 - **Consistency** = una tr. **rispetta i vincoli di integrità**
 - **Isolation** = una tr. **esegue indipendentemente dalle altre**
 - Se più transazioni eseguono in concorrenza, l'effetto netto deve essere equivalente a quello di un'esecuzione sequenziale delle stesse
 - **Durability** = gli **effetti** di una tr. che ha terminato correttamente la sua esecuzione devono essere **persistenti nel tempo**



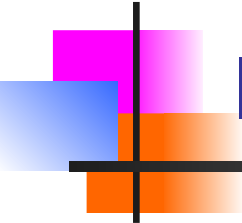
Proprietà ACID applicate ad un bonifico

- Prelevamento di una somma da un conto e versamento in altro
- Atomicità:
 - non è ammessa l'esecuzione parziale, ad esempio solo il prelevamento senza il versamento
- Consistenza:
 - l'importo prelevato deve essere identico a quello versato, ossia al termine dell'operazione la somma dei 2 saldi non cambia
- Isolamento:
 - il bonifico deve avvenire correttamente indipendentemente da eventuali altre operazioni concorrenti sugli stessi conti
- Durabilità:
 - se il bonifico va a buon fine gli effetti permangono indipendentemente da eventuali malfunzionamenti



Possibili esiti di una transazione

- Nel modello che si considera una transazione può:
 - Terminare **correttamente**
 - Questo avviene quando l'applicazione esegue una particolare istruzione, detta **COMMIT** (o COMMIT WORK), che comunica al Transaction Manager il termine delle operazioni che compongono la transazione stessa
 - Terminare **non correttamente** (anticipatamente)
 - Sono possibili 2 casi:
 - È la transazione che, per qualche motivo, decide che non ha senso continuare e quindi “abortisce” eseguendo l'istruzione **ROLLBACK** (o ROLLBACK WORK)
 - È il sistema che non è in grado (ad es. per un guasto o per la violazione di un vincolo) di garantire la corretta prosecuzione della transazione, che viene quindi abortita



Isolation: gestire la concorrenza

- Il Transaction Manager deve garantire che diverse applicazioni non interferiscano tra loro. Se ciò non avviene, si possono avere 4 tipi base di problemi, esemplificati dai seguenti scenari:

Lost Update: due persone, in due agenzie diverse, comprano entrambe l'ultimo biglietto per il concerto degli U2 a Roma (!?)

Dirty Read: nel programma dei concerti degli U2 figura una tappa a Bologna l'11/02/02, ma quando provate a comprare un biglietto per quella data vi viene detto che in realtà non è ancora stata fissata (!?)

Unrepeatable Read: per il concerto degli U2 (finalmente la data è stata fissata!) vedete che il prezzo è di 40 €, ci pensate su 5 minuti, ma il prezzo nel frattempo è salito a 50 € (!?)

Phantom Row: volete comprare i biglietti di tutte e due le tappe degli U2 in Italia, ma quando comprate i biglietti scoprite che le tappe sono diventate 3 (!?)

Lost Update

- Il problema si descrive sinteticamente mediante il seguente diagramma temporale, in cui T1 e T2 sono due transazioni, X è un dato del DB, R(X) indica la lettura di X e W(X) indica la scrittura di X:

T1	X	T2
R(X)	1	
X=X-1	1	
	1	R(X)
	1	X=X-1
W(X)	0	
Commit	0	
	0	W(X)
	0	Commit

- Il problema nasce perché T2 legge il valore di X prima che T1 (che lo ha già letto) lo modifichi (“entrambe vedono l’ultimo biglietto”)

Dirty Read

- In questo caso il problema è che una transazione legge un dato “che non c’è”:

T1	X	T2
R(X)	0	
X=X+1	0	
W(X)	1	
	1	R(X)
Rollback	0	
	0	...
	0	Commit

- Quanto svolto da T2 si basa su un valore di X “intermedio”, e quindi non stabile (“la data definitiva non è l’11/02/02”)
- Le conseguenze sono imprevedibili (dipende cosa fa T2) e si presenterebbero anche se T1 non abortisse

Unrepeatable Read

- Ora il problema è che una transazione legge due volte un dato e trova valori diversi (“il prezzo nel frattempo è aumentato”):

T1	X	T2
R(X)	0	
	0	R(X)
	1	X=X+1
	1	W(X)
	1	Commit
R(X)	1	
Commit	1	

- Anche in questo caso si possono avere gravi conseguenze
- Lo stesso problema si presenta per transazioni di “analisi” (ad es.: T1 somma l’importo di 2 conti correnti mentre T2 esegue un trasferimento di fondi dall’uno all’altro)

Phantom Row

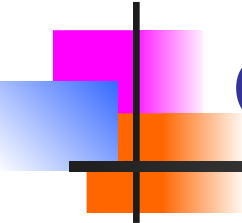
* Questo caso si può presentare quando vengono inserite tuple che un'altra transazione potrebbe logicamente considerare

```
SELECT CodProg  
FROM Prog  
WHERE Citta = 'Bologna'  
  
INSERT INTO Prog  
VALUES ('P03', 'Bologna')
```

Prog

CodProg	Citta	
P01	Milano	t1
P01	Bologna	t2
P02	Bologna	t3
P03	Bologna	t4

T1	T2
R(t2)	
R(t3)	
	Insert(t4)
	Commit
R(t2)	
R(t3)	
R(t4)	
Commit	



Come garantire l'Isolation

- Una comune tecnica usata dai DBMS per evitare i problemi visti consiste nell'uso di **lock**
 - I lock (“blocchi”) sono un **meccanismo comunemente usato dai sistemi operativi** per disciplinare l'accesso a risorse condivise
 - Per eseguire un'operazione è prima necessario “acquisire” un lock sulla **risorsa interessata** (ad es. una tupla)
 - I lock sono di vario tipo; quelli di base sono:
 - **S (Shared)**: un lock condiviso è necessario per **leggere**
 - **X (eXclusive)**: un lock esclusivo è necessario per **scrivere/modificare**

Compatibilità dei lock

- Il **Lock Manager** è un modulo del DBMS che si occupa di tener traccia di quali sono le risorse correntemente in uso e di quali transazioni le stanno usando (e in che modo)
- Quando una transazione T vuole operare su un dato X, viene inviata la richiesta di acquisizione del lock corrispondente al Lock Manager
- Il lock viene accordato a T in funzione della seguente **tabella di compatibilità**

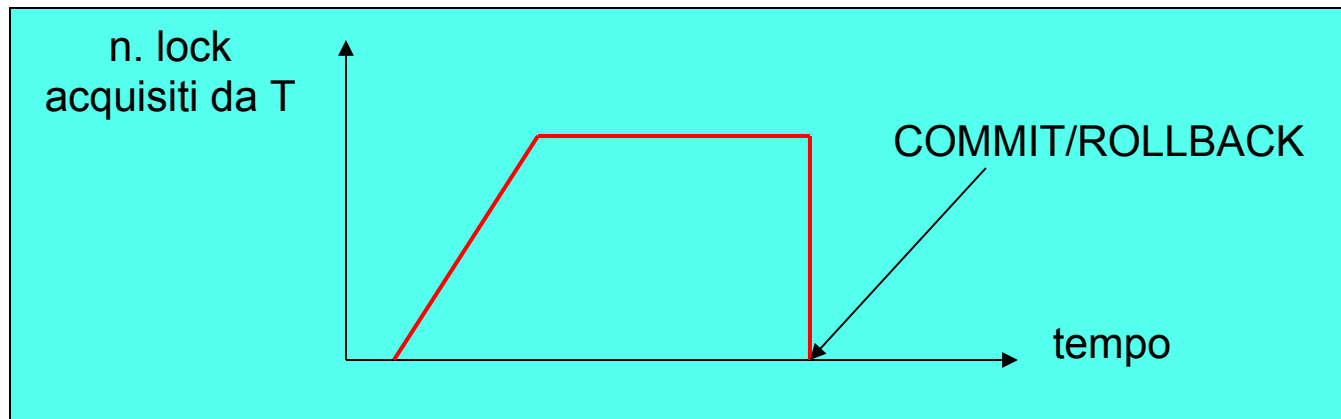
Su X è già stato acquisito
da un'altra transazione
un lock di tipo

T richiede un lock di tipo	Su X è già stato acquisito da un'altra transazione un lock di tipo	
	S	X
S	OK	NO
X	NO	NO

- Quando T ha terminato di usare X, può **rilasciare** il lock (**unlock(X)**)

Protocollo Strict 2-phase lock (Strict 2PL)

- Il modo con cui le transazioni rilasciano i lock acquisiti è la chiave per risolvere i problemi di concorrenza
- Si può dimostrare che se
 - Una transazione prima acquisisce tutti i lock necessari
 - Rilascia i lock solo al termine dell'esecuzione (COMMIT o ROLLBACK)allora l'Isolation è garantita



- Come effetto collaterale si possono verificare **deadlock**, ossia situazioni di stallo, che vengono risolte facendo abortire una transazione

Assenza di Lost Update

- L'esecuzione prima vista si modifica come segue:

T1	X	T2
S-lock(X)	1	
R(X)	1	
X=X-1	1	
	1	S-lock(X)
	1	R(X)
	1	X=X+1
X-lock(X)	1	
wait	1	X-lock(X)
wait	1	wait

- Né T1 né T2 riescono ad acquisire il lock per poter modificare X (restano in attesa, “wait”); si verifica quindi un deadlock. Se il sistema decide di abortire, ad es., T2, allora T1 può proseguire

Assenza di Dirty Read

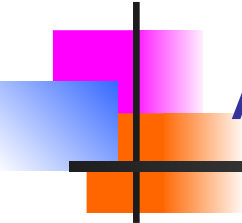
T1	X	T2
S-lock(X)	0	
R(X)	0	
X=X+1	0	
X-lock(X)	0	
W(X)	1	
	1	S-lock(X)
	0	wait
Rollback	0	wait
Unlock(X)	0	wait
	0	R(X)

- In questo caso l'esecuzione corretta richiede che T2 aspetti la terminazione di T1 prima di poter leggere il valore di X

Assenza di Unrepeatable Read

T1	X	T2
S-lock(X)	0	
R(X)	0	
	0	S-lock(X)
	0	R(X)
	0	X=X+1
	0	X-lock(X)
	0	wait
R(X)	0	wait
Commit	0	wait
Unlock(X)	0	wait
	1	W(X)

- Anche in questo caso T2 viene messa in attesa, e T1 ha quindi la garanzia di poter leggere sempre lo stesso valore di X

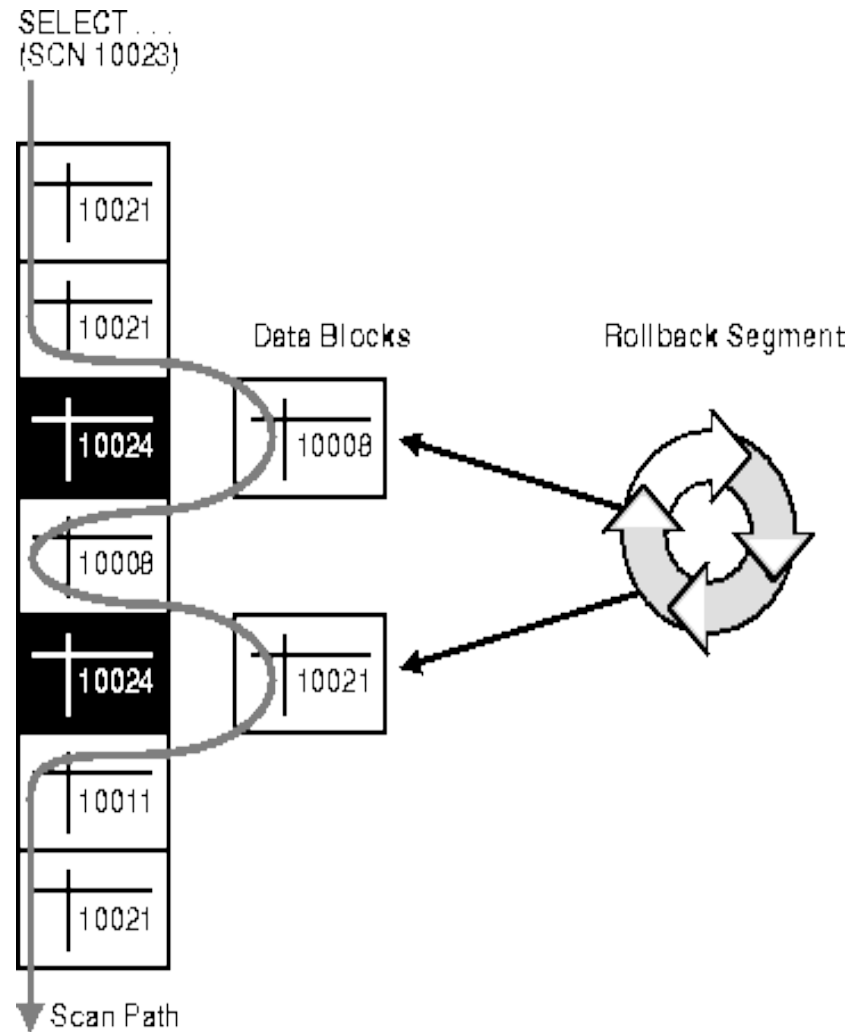


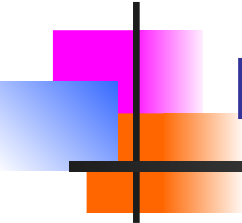
Assenza di Phantom Row

- Questo è, tra quelli visti, il problema più difficile da risolvere. Le soluzioni adottabili sono varie, e differiscono in complessità e livello di concorrenza che permettono:
 - Si può acquisire un **S-lock su tutta la table**, e poi **richiedere gli X-lock per le tuple che si vogliono modificare**
 - Si introduce un nuovo tipo di lock, detto “**predicate lock**”, che riguarda **tutte le tuple che soddisfano un predicato** (**Sede = 'Bologna'** nell'esempio)
 - Se esiste un **indice su Sede**, si pone un **lock sulla foglia** che contiene 'Bologna'
- Nei DBMS la situazione è in realtà più complessa, sia per i tipi di lock presenti, sia per la “granularità” a cui i lock possono essere richiesti e acquisiti

Phantom row e Multi-versioning

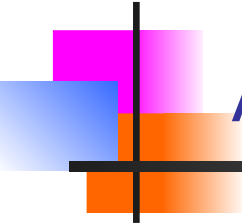
- di norma quando una tupla viene modificata i valori precedenti si perdono
- l'idea del multi-versioning è introdurre il concetto di versione delle tuple modificate
- es: quando una query viene inoltrata viene determinato il System Change Number (SCN) corrente (e.g. 10023)
- si leggono solo i blocchi con $SCN \leq 10023$
- le nuove tuple ins. avranno un SCN maggiore e non saranno considerate dalla query





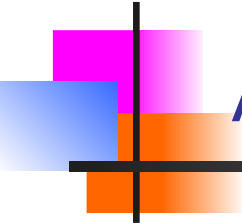
Livelli di Isolamento ANSI/ISO SQL-92

- Livello 0: READ UNCOMMITTED
 - consentito solo per read-only transaction
 - nessuna anomalie è eliminata tranne lost update
 - non viene posto alcun tipo di lock
- Livello 1: READ COMMITTED
 - impedisce le anomalie dirty read
 - non pone Shared lock per le letture
- Livello 2: REPEATABLE READ
 - garantisce il livello precedente
 - impedisce anomalie lost update e unrepeatable read
 - pone Shared e eXclusive lock
- Livello 3: SERIALIZABLE
 - garantisce il livello precedente
 - impedisce le anomalie phantom read (tecniche precedenti)



Atomicity e Durability: convivere con i guasti

- L’“altra faccia” della gestione delle transazioni riguarda il trattamento dei “guasti” (**failure**), ovvero di tutti quegli **eventi anomali che possono pregiudicare il corretto funzionamento delle transazioni**.
- I tipi di malfunzionamenti sono essenzialmente 3:
 - **Transaction failure**: è il caso in cui una transazione abortisce
 - Gli effetti della transazione devono essere annullati
 - **System failure**: il sistema ha un guasto hardware o software che provoca l’interruzione di tutte le transazioni in esecuzione, senza però danneggiare la memoria permanente (dischi)
 - **Media (o device) failure**: in questo caso il contenuto (persistente) della base di dati viene danneggiato



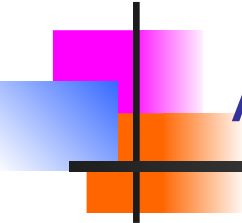
Atomicity e Durability: il Log

- Per far fronte ai malfunzionamenti, un DBMS fa uso di diversi strumenti, in particolare:
 - **DataBase Dump**: copia di archivio del DB (o parte di esso)
 - **Log file** (“giornale”): file in cui vengono registrate le operazioni di modifica eseguite dalle transazioni
 - Se **una pagina P del DB viene modificata da T**, il log contiene un record del tipo
$$(\underline{\text{LSN}}, T, \text{PID}, \text{before}(P), \text{after}(P), \text{prevLSN})$$
 - **LSN** = Log Sequence Number (n. progressivo del record)
 - **T** = identificatore della transazione
 - **PID** = identificatore della pagina modificata
 - **before(P)** = è la cosiddetta **before image** di P, ovvero il contenuto di P prima della modifica
 - **after(P)** = è l'**after image** di P, ossia il contenuto di P dopo la modifica
 - **prevLSN** = LSN del precedente record del Log relativo a T

Esempio di Log

- Il Log contiene anche record che specificano l'inizio (**BEGIN**) di una transazione e la sua terminazione (**COMMIT** o **ABORT**)

LSN	T	PID	before(P)	after(P)	prevLSN
...					
235	T1	BEGIN			-
236	T2	BEGIN			-
237	T1	P15	(abc, 10)	(abc, 20)	235
238	T2	P18	(def, 13)	(ghf, 13)	236
239	T1	COMMIT			237
240	T2	P19	(def, 15)	(ghf, 15)	238
241	T3	BEGIN			-
242	T2	P19	(ghf, 15)	(ghf, 17)	240
243	T3	P15	(abc, 20)	(abc, 30)	241
244	T2	ABORT			242
245	T3	COMMIT			243
...					



Alcune precisazioni

- Affinché il Log sia utile, è importante che prima di scrivere su disco una pagina P modificata, il corrispondente Log record sia già stato scritto nel Log (protocollo WAL = “Write Ahead Log”)
- Quando una transazione T modifica una pagina P, il DBMS (o, meglio, il Buffer Manager) ha 2 possibilità:
 - Politica No-steal: Mantenere la pagina P nel buffer, e attendere che T abbia completato correttamente la sua esecuzione prima di scriverla su disco
 - Politica Steal: Scrivere P quando “più conviene” (per far spazio nel buffer), possibilmente anche prima della terminazione di T
- Anche in fase di COMMIT si hanno due possibilità:
 - Politica Force: prima di scrivere il record di COMMIT sul Log, si forza la scrittura su disco di tutte le pagine modificate da T
 - Politica No-force: si scrive subito il record di COMMIT sul Log; quindi quando T termina alcune delle sue modifiche ancora non sono state rese persistenti
- DBMS quali ORACLE e DB2, per motivi di efficienza, adottano la combinazione Steal/No-force

Transaction failure

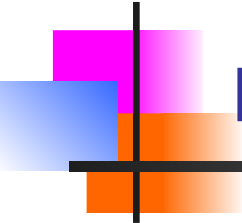
- Con Steal/No-force, se una transazione T abortisce è possibile che alcune pagine da essa modificate siano già state scritte su disco
- Per annullare (**UNDO**) queste modifiche si scandisce il Log a ritroso (usando i prevLSN) e si ripristinano nel DB le **before image** delle pagine modificate da T

LSN	T	PID	before(P)	after(P)	prevLSN
...					
236	T2	BEGIN			-
237	T1	P15	(abc, 10)	(abc, 20)	235
238	T2	P18	(def, 13) ← (ghf, 13)		236
239	T1	COMMIT			237
240	T2	P19	(def, 15) ← (ghf, 15)		238
241	T3	BEGIN			-
242	T2	P19	(ghf, 15) ← (ghf, 17)		240
243	T3	P15	(abc, 20)	(abc, 30)	241
244	T2	ABORT			242

System failure

- Nel caso di system failure, vengono disfatte tutte le transazioni il cui COMMIT record non si trova nel Log
- Se una transazione T ha eseguito COMMIT, non è garantito che tutte le sue modifiche siano state registrate su disco (*politica No-force*); pertanto T va "rifatta" (REDO), riscrivendo le *after image* che si trovano sul Log
- La procedura di "restart" che ne deriva può risultare molto onerosa; per ridurre i tempi di ripristino, periodicamente si può eseguire un "checkpoint", ovvero una *scrittura forzata su disco delle pagine modificate*

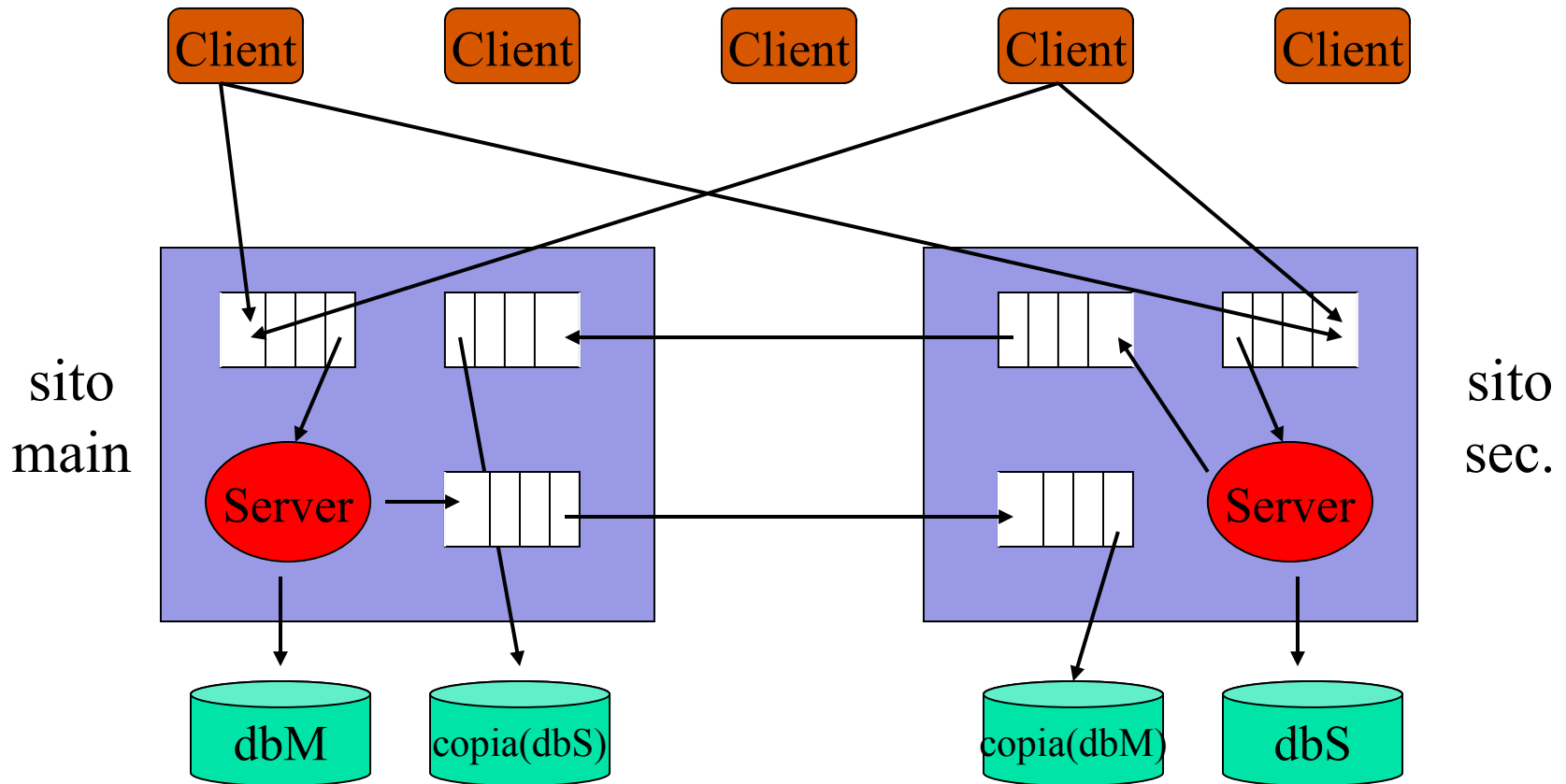
LSN	T	PID	before(P)	after(P)	prevLSN
...					
235	T1	BEGIN			-
236	T2	BEGIN			-
237	T1	P15	(abc, 10)	(abc, 20)	235
238	T2	P18	(def, 13)	→ (ghf, 13)	236
239	T1	COMMIT			237
...					



Media failure

- Nel caso di media failure si ha un ripristino che usa una copia archiviata del DB (**DataBase Dump**)
- Facendo uso del Log, si rifanno quindi tutte le transazioni che hanno eseguito COMMIT
- Un approccio alternativo consiste nel fare uso di **dischi multipli** e adottare una tecnica di “**mirroring**” in cui **ogni scrittura viene eseguita su tutti i dischi in parallelo**
- Tecnologie più sofisticate si basano sull'uso dei cosiddetti sistemi **RAID** (**Redundant Array of Inexpensive Disks**), in cui si fa uso di codici a correzione di errore

Failure e Replicazione dei dati



- Due siti distinti (main e secondary) che gestiscono un unico db replicato