# Tecnologia di un Database Server (centralizzato)

# Gestione dell'affidabilità

**Angelo Montanari**

**Dipartimento di Matematica e Informatica**
**Università di Udine**

# Controllo dell'affidabilità

Il controllore dell'affidabilità garantisce due proprietà fondamentali delle transazioni:

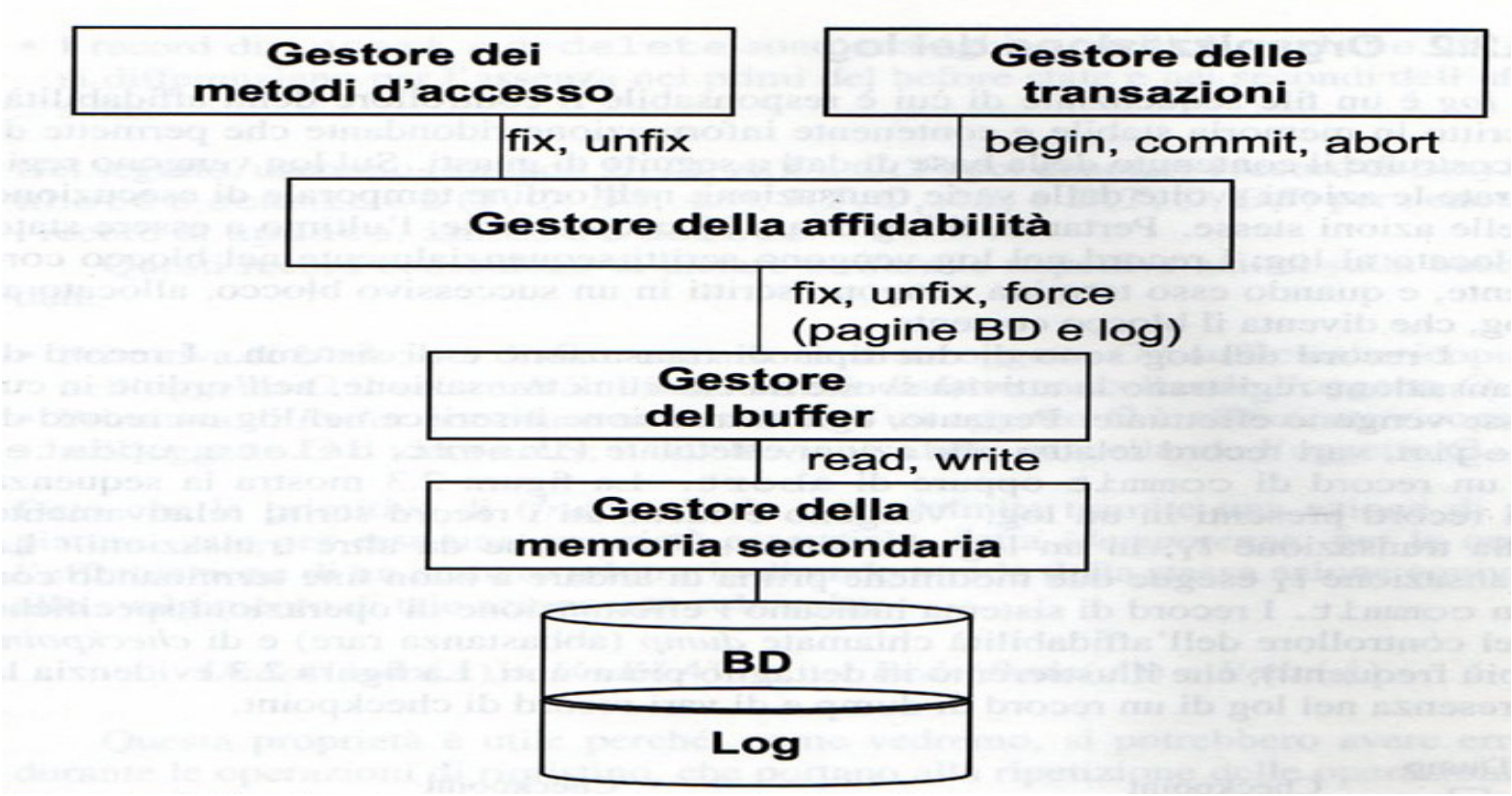
- **atomicità**;
- **persistenza**.

In particolare, il controllore dell'affidabilità è il responsabile della scrittura del **log**, un archivio/file persistente che registra le varie azioni svolte dal DBMS.

Ogni operazione di scrittura sulla base di dati viene protetta tramite un'azione sul log, in modo che sia possibile **disfare** (undo) le azioni a seguito di malfunzionamenti o guasti che precedono il commit oppure **rifare** (redo) queste azioni qualora la loro buona riuscita sia dubbia e le transazioni abbiano effettuato il commit.

Dato il suo ruolo fondamentale, è necessario che il log sia sufficientemente **robusto**.

# Architettura del gestore dell'affidabilità



## I compiti del controllore dell'affidabilità

- Realizza i comandi transazionali *begin transaction*, *commit work*, and *rollback work*.
- Realizza le primitive di ripristino (recovery) dopo i malfunzionamenti / guasti, dette rispettivamente ripresa a caldo e ripresa a freddo.
- Riceve le richieste di accesso alle pagine in lettura e in scrittura, che passa al gestore del buffer, e genera altre richieste di lettura e scrittura di pagine necessarie a garantire robustezza e resistenza ai guasti.
- Predispone i dati essenziali per i meccanismi di ripristino dai guasti, in particolare azioni di checkpoint e di dump.

# La nozione di memoria stabile

**Memoria stabile** = memoria resistente ai guasti.

La nozione di memoria stabile è un'**astrazione**: a nessuna memoria può essere associata una probabilità nulla di fallimento. Meccanismi di replicazione e protocolli di scrittura robusti possono rendere tale probabilità arbitrariamente prossima allo zero.

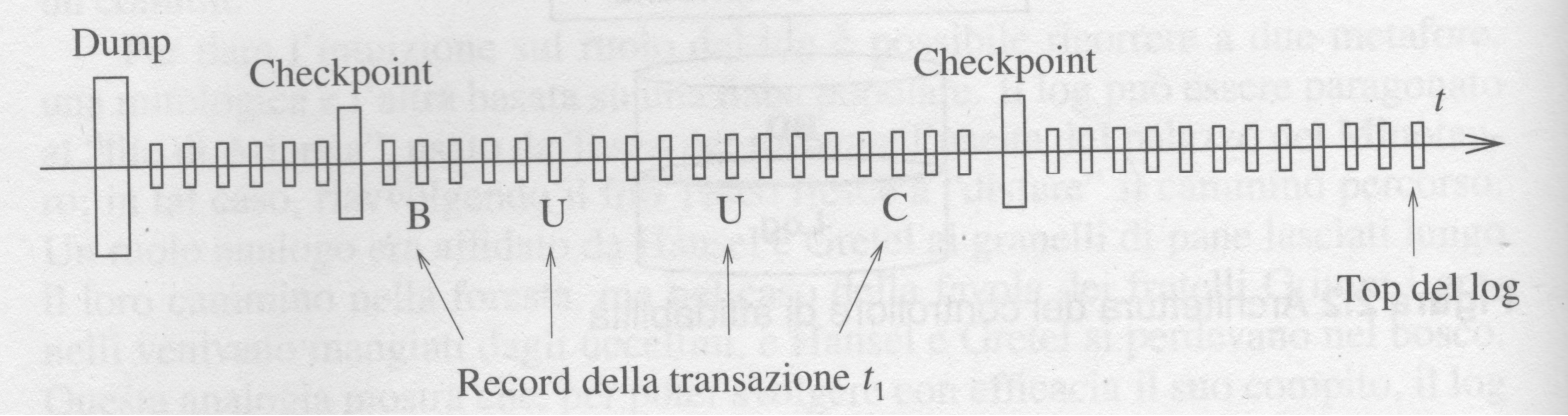
I meccanismi di controllo dell'affidabilità vengono definiti **come se** la memoria stabile fosse effettivamente esente da guasti: un guasto della memoria stabile viene considerato un evento catastrofico.

Come realizzare una memoria stabile? (i) un'unità a nastro; (ii) una coppia di dispositivi (un'unità a nastro e un disco in cui vengono registrate le stesse informazioni); (iii) **due unità a disco** a specchio (mirrored), contenenti la stessa informazione, scritte con un'operazione di scrittura attenta (ha successo solo se l'informazione viene registrata su entrambi i dischi) - l'informazione stabile è disponibile in linea.

# Organizzazione dei log

Il **log** è un file sequenziale, gestito dal controllore dell'affidabilità, scritto in memoria stabile. Sul log vengono registrate le azioni eseguite dalle varie transazioni nell'ordine temporale di esecuzione.

Il log ha un blocco corrente (l'ultimo ad esso allocato). I record di log vengono scritti sequenzialmente in tale blocco.



## Legenda

I record del log sono di due tipi: record di transazione e di sistema.

I **record di transazione** registrano le attività svolte dalle transazioni nell'ordine in cui esse sono eseguite. Per ogni transazione, un record di begin (B), record di insert, delete e update (U) e un record di commit (C) o di abort (A).

Nella figura precedente vengono evidenziati i record relativi alla transazione $t_1$, che esegue due modifiche prima di terminare con successo con un commit.

I **record di sistema** registrano l'esecuzione di operazioni di dump (rare) e di checkpoint (più frequenti).

## La struttura dei record nel log

Per descrivere gli effetti di una transazione $t_i$, vengono scritti nel log i seguenti record:

- i **record** di **begin**, **commit** e **abort**, che contengono il tipo di record e l'identificativo $t_i$ della transazione;
- i **record** di **update**, che contengono l'identificativo $t_i$ della transazione, l'identificativo $O_j$ dell'oggetto modificato, i valori $BS_i$ (before state) e $AS_i$ (after state) che descrivono rispettivamente il valore di $O_j$ prima e dopo la modifica (per semplicità assumeremo che $BS_i$ e $AS_i$ contengano copie complete delle pagine modificate; in realtà, il loro contenuto è molto più sintetico);
- i **record** di **insert** e **delete**, i primi privi del valore $BS_i$, i secondi del valore $AS_i$.

## Le primitive undo e redo

**Convenzioni notazionali**: data una transazione $T$, indicheremo con $B(T)$, $C(T)$ e $A(T)$ i record di begin, commit e abort relativi a $T$, rispettivamente, e con $U(T, O, BS, AS)$, $I(T, O, AS)$ e $D(T, O, BS)$ i record di update, insert e delete, rispettivamente.

I record del log associati ad una transazione, consentono di disfare e rifare le corrispondenti azioni sulla base di dati:

- **primitive** di **undo**: per disfare un'azione su un oggetto $O$, è sufficiente ricopiare in $O$ il valore $BS$ (l'insert viene disfatto cancellando l'oggetto $O$)
- **primitiva** di **redo**: per rifare un'azione su un oggetto $O$, è sufficiente ricopiare in $O$ il valore $AS$ (il delete viene rifatto cancellando l'oggetto $O$)

## Proprietà di idempotenza

Le operazioni di undo e redo godono della proprietà di **idempotenza**:

l'esecuzione di un numero arbitrario di undo e redo di una stessa azione è equivalente ad una singola esecuzione di tale azione.

Formalmente,

$undo(undo(A)) = undo(A)$

$redo(redo(A)) = redo(A)$

Utilità di tale proprietà: qualora si verifichino errori in fase di ripristino, sarà sufficiente ripetere le operazioni fino a quando verranno portate a termine con successo.

# L'operazione di checkpoint

L'operazione di **checkpoint** viene svolta periodicamente, in stretto coordinamento con il buffer manager, per registrare le **transazioni attive** (al fine di semplificare le procedure di ripristino dopo un guasto).

1. si sospende l'accettazione di operazioni di scrittura, commit o abort, da parte di qualunque transazione;
2. si trasferiscono in memoria di massa (tramite opportune operazioni di *force*) tutte le pagine del buffer su cui sono state eseguite delle modifiche da parte di transazioni che hanno già effettuato il commit;
3. si scrive in modo sincrono (*force*) nel log un record di checkpoint che contiene gli identificatori delle transazioni attive;
4. si riprende l'accettazione delle operazioni sospese.

Si noti che in tal modo gli effetti delle transazioni che hanno eseguito il commit vengono registrati nella base di dati in modo persistente.

# L'operazione di dump

L'operazione di **dump** produce una copia completa della base di dati, effettuata in mutua esclusione con tutte le altre transazioni quando il sistema non è operativo.

La copia viene memorizzata in memoria stabile (backup). Al termine del dump, viene scritto nel log un record di dump, che segnala l'avvenuta esecuzione dell'operazione in un dato istante. Il sistema riprende, quindi, il suo funzionamento normale.

Convenzioni notazionali. Useremo $DUMP$ per denotare il record di dump e $CK(T_1, T_2, \ldots, T_n)$ per denotare il record di checkpoint, dove $T_1, T_2, \ldots, T_n$ sono gli identificatori delle transazioni attive all'istante del checkpoint.

## La regola write-ahead-log

Durante il funzionamento normale delle transazioni. il gestore dell'affidabilità garantisce il rispetto delle seguenti **due regole** che impongono dei requisiti minimi che consentono di ripristinare la correttezza della base di dati in caso di guasti:

- La **regola write-ahead-log** (**WAL**) impone che la parte before-state dei record di un log venga scritta nel log (con un'operazione di scrittura in memoria stabile) **prima** di effettuare la corrispondente operazione sulla base di dati.

  Questa regola consente di disfare le scritture già effettuate in memoria di massa da parte di una transazione che non ha ancora effettuato un commit (per ogni aggiornamento viene reso disponibile in modo affidabile il valore precedente la scrittura).

## La regola di commit-precedenza

- La **regola di commit-precedenza** impone che la parte after state dei record di un log venga scritta nel log (con un'operazione di scrittura in memoria stabile) **prima** di effettuare il commit.

  Questa regola consente di rifare le scritture di una transazione che ha effettuato il commit le cui pagine modificate non sono ancora state trascritte dal buffer manager in memoria di massa.

In realtà, anche se le regole fanno riferimento a before state e after state, nella pratica entrambe le componenti del record di log vengono scritte assieme. **Regole semplificate**: (i) i record di log siano scritti prima dei corrispondenti record della base di dati (regola WAL); (ii) i record di log siano scritti prima dell'esecuzione dell'operazione di commit (regola di commit-precedenza).

## La scrittura dei record di commit e di abort

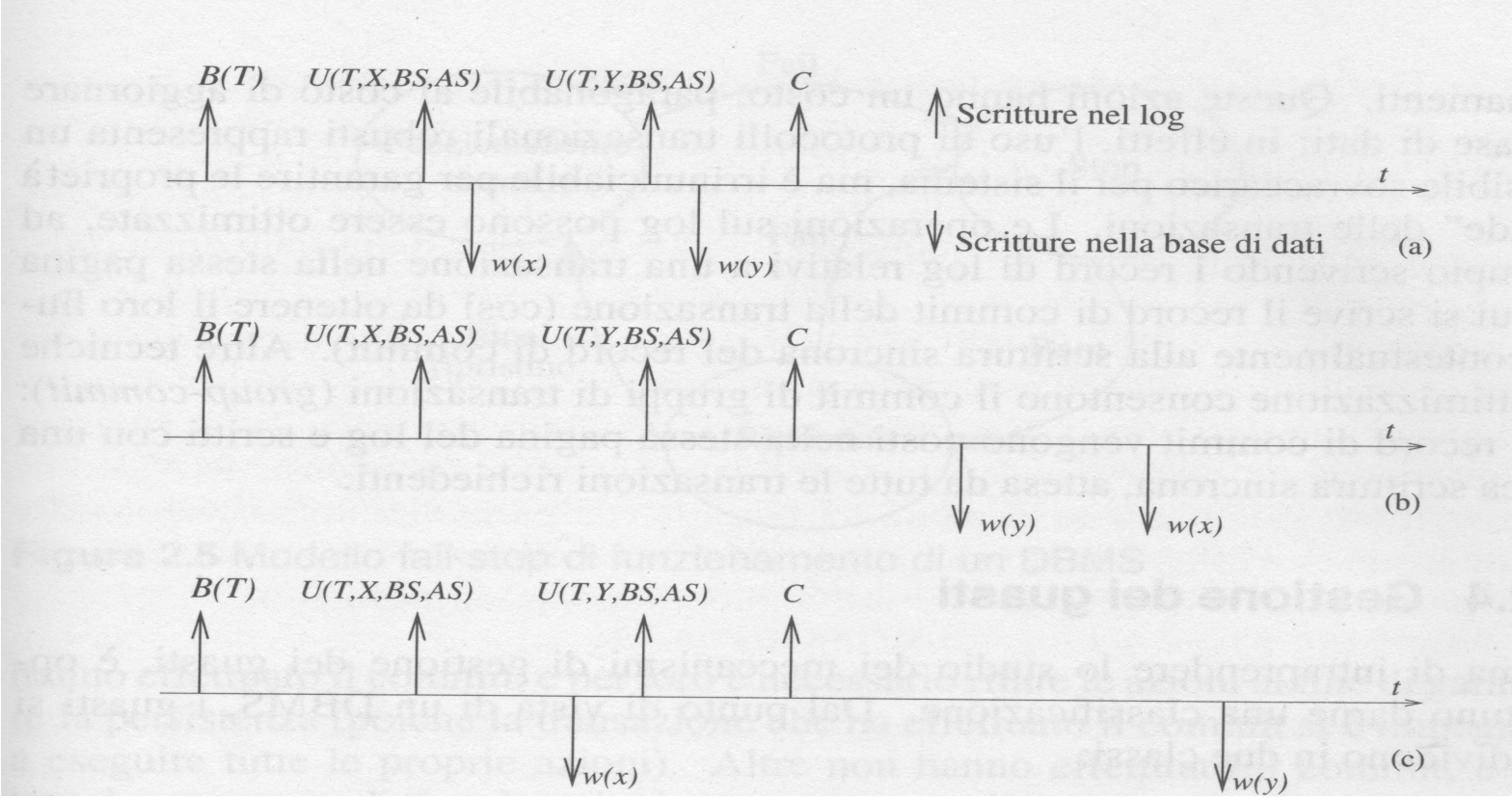
La **scrittura** dei **record di commit**. La transazione sceglie in modo atomico e indivisibile tra abort e commit nel momento in cui scrive sul log, in modo asincrono (primitiva force), il record di commit.

Guasti che occorrono prima (**undo**) e dopo (**redo**) la scrittura del record di commit. I guasti che si verificano prima impongono l'undo delle azioni effettuate, in modo da riportare la base di dati nello stato iniziale; quelli che si verificano dopo impongono il redo delle azioni effettuate, in modo da produrre con certezza lo stato finale della transazione.

La **scrittura** dei **record di abort**. Il record di abort definisce in modo atomico la decisione di abortire la transazione (suicidio/omicidio). Dato che tale decisione non modifica le decisioni del gestore della recovery, la scrittura del record di abort può essere fatta in modo asincrono con un'operazione di flush.

# Protocolli di scrittura del log e della base di dati

Le due regole precedenti impongono i seguenti protocolli per la scrittura del log e della base di dati (per semplicità, si considerano solo operazioni di update):



## Schema (a)

La transazione scrive inizialmente il record $B(T)$. Successivamente, esegue le operazioni di update scrivendo prima il record di log $U(T, O, BS, AS)$ e poi la pagina della base di dati, che passa dal valore $BS$ al valore $AS$.

Tutte queste pagine devono essere effettivamente scritte (autonomamente dal gestore del buffer o con esplicite richieste di force) prima del commit, il quale comporta una scrittura sincrona (force).

Al commit, tutte le pagine della base di dati modificate dalla transazione sono state scritte in memoria di massa. Tale schema **non** richiede operazioni di **redo**.

## Schemi (b) e (c)

Nello schema (b), la scrittura dei record di log precede quella delle azioni sulla base di dati, che avvengono dopo la decisione di commit e la conseguente scrittura sincrona del record di commit sul log. Tale schema **non** richiede operazioni di **undo**.

Nello schema (c) (**più generale** e **comunemente usato**), le scritture sulla base di dati, una volta protette dalle opportune scritture sul log, possono avvenire in un qualunque momento rispetto alla scrittura del record di commit sul log. Tale schema consente al gestore del buffer di ottimizzare le operazioni di flush relative ai suoi buffer, indipendentemente dal controllore dell'affidabilità. Tale schema **può** richiedere operazioni sia di **undo** sia di **redo**.

## I costi

Tutti e tre gli schemi rispettano le due regole fondamentali e scrivono il record di commit in modo sincrono.

Si differenziano per il momento in cui scrivono le pagine della base di dati.

Il costo delle scritture del log è paragonabile al costo dell'aggiornamento della base di dati: l'uso di protocolli transazionali robusti introduce, quindi, un notevole **sovraccarico** per il sistema, ma garantisce le proprietà acide.

## Ottimizzazione delle operazioni sul log

Le **operazioni sul log** possono essere **ottimizzate**.

Una possibilità: scrivere i record di log relativi ad una transazione nella **stessa pagina** in cui si scrive il record di commit della transazione (il loro flush viene eseguito contestualmente alla scrittura sincrona del record di commit).

In alternativa, può essere eseguito un **group-commit**: vari record di commit vengono posti nella stessa pagina del log e scritti con un'unica scrittura sincrona, attesa da tutte le transazioni richiedenti.

Sistemi con un elevato numero di transazioni per secondo (tps) possono adottare **schemi paralleli di scrittura** del log.

# Gestione dei guasti

Distinguiamo due classi di guasti.

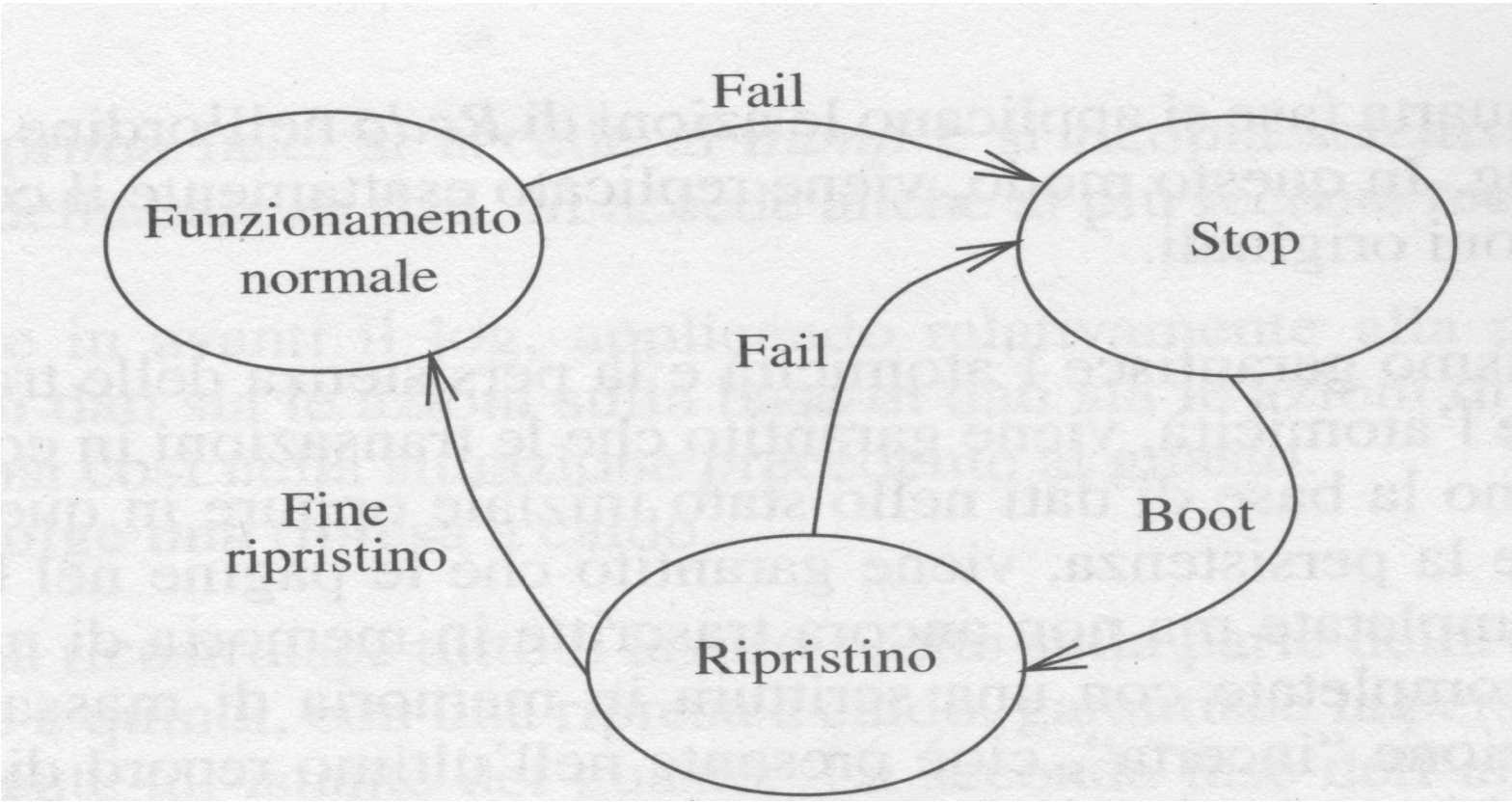
**Guasti di sistema**. Guasti causati da bachi software, ad esempio di sistema operativo, oppure da interruzioni del funzionamento dei dispositivi, ad esempio cali di tensione, che portano il sistema in uno stato inconsistente.

Effetto: perdita del contenuto della memoria principale (buffer), conservazione del contenuto della memoria di massa (base di dati e log).

**Guasti di dispositivo**. Guasti relativi ai dispositivi di gestione della memoria di massa (ad esempio, strisciamento delle testine di un disco, che causa la perdita del suo contenuto). Dato che il log risiede in memoria stabile, un guasto di dispositivo può causare unicamente una perdita (di parte) del contenuto della base di dati, non del log.

# Il modello fail-stop

Modello ideale di funzionamento di un DBMS



## Legenda

**Modello fail-stop**. Quando il sistema rileva un guasto (di sistema o di funzionamento), esso forza un arresto completo delle transazioni e il successivo ripristino del corretto funzionamento del sistema operativo (boot).

Viene, quindi, avviata una procedura di ripristino (recovery), denominata **ripresa a caldo** (warm restart) nel caso di guasti di sistema e **ripresa a freddo** (cold restart) nel caso di guasti di dispositivo.

Al termine delle procedure di ripresa, il sistema torna a disposizione delle transazioni. Il buffer è completamente vuoto e può riprendere a caricare pagine della base di dati e del log.

## Come funzionano le procedure di ripresa?

Al verificarsi del guasto, esiste un insieme di **transazioni potenzialmente attive**, delle quali si ignora se hanno ultimato le loro azioni sulla base di dati (in quanto il buffer manager ha perso ogni informazione utile).

Tali transazioni possono essere suddivese in due classi, in base alle informazioni presenti nel log.

Transazioni che hanno effettuato il **commit**. Per esse è necessario rifare le azioni al fine di garantire la proprietà di persistenza.

Transazioni che **non** hanno effettuato il **commit**. Per esse è necessario disfare le azioni, in quanto non è noto quali azioni siano state effettivamente eseguite (la base di dati deve essere lasciata nello stato precedente l'esecuzione della transazione).

## L'uso del record di end

Alcuni sistemi, al fine di semplificare i protocolli di ripristino, aggiungono un ulteriore record al log, denominato **record di end**, nel momento in cui le operazioni di trascrizione (flush) delle pagine da parte del buffer manager sono terminate.

Ciò consente di individuare una terza classe di transazioni, per le quali non occorre né rifare né disfare le azioni.

In genere, per non complicare la gestione delle transazioni, non viene previsto alcun record di end.

Nel seguito, faremo riferimento al modello fail-stop dei guasti, senza record di end.

# Ripresa a caldo - 1

La ripresa a caldo è articolata in 4 fasi successive:

(i) Si accede all'ultimo blocco del log, corrente all'istante di guasto, e si ripercorre all'indietro il log fino al record di checkpoint.

(ii) Si individuano le transazioni da rifare e da disfare. Si costruiscono due insiemi, detti di UNDO e di REDO, contenenti identificativi di transazioni. All'inizio, l'insieme di UNDO contiene l'insieme delle transazioni attive al checkpoint; l'insieme di REDO è vuoto. Si percorre, quindi, il log in avanti, aggiungendo all'insieme di UNDO tutte le transazioni di cui è presente un record di begin e spostando dall'insieme di UNDO all'insieme di REDO tutti gli identificativi delle transazioni di cui è presente il record di commit. Al termine di tale fase, gli insiemi di UNDO e REDO contengono rispettivamente gli identificativi delle transazioni da disfare e quelli delle transazioni da rifare.

## Ripresa a caldo - 2

(iii) Si ripercorre all'indietro il log disfacendo le azioni delle transazioni contenute nel set di UNDO, risalendo fino alla prima azione della transazione più *vecchia* fra quelle presenti nei due insiemi di UNDO e REDO (tale azione potrebbe precedere il record di checkpoint nel log).

(iv) Nell'ultima fase, si applicano le azioni delle transazioni dell'insieme di REDO nell'ordine in cui sono state registrate nel log. In tal modo, viene replicato esattamente il comportamento delle transazioni originali.

## Atomicità e persistenza delle transazioni

La soluzione proposta garantisce atomicità e persistenza delle transazioni.

**Atomicità**. Le transazioni in corso all'istante del guasto lasciano la base di dati nello stato iniziale o nello stato finale, sulla base dell'assenza o della presenza del relativo record di commit.

**Persistenza**. Le pagine presenti nel buffer relative a transazioni completate con successo, ma non ancora trascritte in memoria di massa, vengono effettivamente completate con una scrittura in memoria di massa.

Si noti che ogni **transazione incerta**, presente nell'ultimo record di checkpoint o iniziata dopo l'ultimo checkpoint, viene disfatta, se l'ultimo suo record scritto nel log è un record che descrive un'azione o è un record di abort, o rifatta, se l'ultimo suo record nel log è il record di commit.

## Un esempio

Vediamo ora un esempio di **applicazione del protocollo**.

Si supponga che nel log vengano registrate nell'ordine le seguenti azioni:

$B(T1), B(T2), U(T2, O1, B1, A1), I(T1, O2, A2), B(T3), C(T1),$

$B(T4), U(T3, O2, B3, A3), U(T4, O3, B4, A4), CK(T2, T3, T4),$

$C(T4), B(T5), U(T3, O3, B5, A5), U(T5, O4, B6, A6), D(T3, 05, B7),$

$A(T3), C(T5), I(T2, O6, A8).$

Successivamente, si verifica un guasto.

## L'applicazione del protocollo all'esempio - 1

1. Si accede al record di checkpoint; UNDO $= \{T2, T3, T4\}$ e REDO $= \{\}$.

2. Si percorre in avanti il record di log e si aggiornano gli insiemi di UNDO e di REDO:

   (a) $C(T4)$: UNDO $= \{T2, T3\}$ e REDO $= \{T4\}$

   (b) $B(T5)$: UNDO $= \{T2, T3, T5\}$ e REDO $= \{T4\}$

   (c) $C(T5)$: UNDO $= \{T2, T3\}$ e REDO $= \{T4, T5\}$

## L'applicazione del protocollo all'esempio - 2

3. Si ripercorre indietro il log fino all'azione $U(T2, O1, B1, A1)$ eseguendo le seguenti azioni di *Undo*:

   (a) *Delete*(06

   (b) *Re-insert* $(05 = B7)$

   (c) $03 = B5$

   (d) $02 = B3$

   (e) $01 = B1$

4. Vengono svolte le azioni di *Redo*:

   (a) $O3 = A4$ (si noti che $A4 = B5$)

   (b) $04 = A6$

## Ripresa a freddo

La **ripresa a freddo** reagisce ad un guasto che provoca il deterioramento di una parte della base di dati. Si articola in tre fasi successive:

1. durante la prima fase, si accede al più recente record di dump nel log; sulla base delle informazioni in esso contenute, si accede al dump e si ricopia selettivamente la parte deteriorata della base di dati;
2. a partire dal record di dump, si ripercorre in avanti il log, applicando, relativamente alla parte deteriorata della base di dati, sia le azioni sulla base di dati sia le azioni di commit e abort, riportandosi in tal modo alla situazione precedente il verificarsi del guasto;
3. infine, si esegue una ripresa a caldo.

## Effetti sulla base di dati

La soluzione proposta ricostruisce passo passo tutto il lavoro svolto sulla parte della base di dati soggetta al guasto; successivamente, con una ripresa a caldo, garantisce le proprietà di persistenza e di atomicità relativamente all'istante di guasto.

La seconda fase dell'algoritmo può essere ottimizzata, ad esempio, effettuando unicamente le azioni di transazioni che abbiano eseguito il commit prima del verificarsi del guasto.