



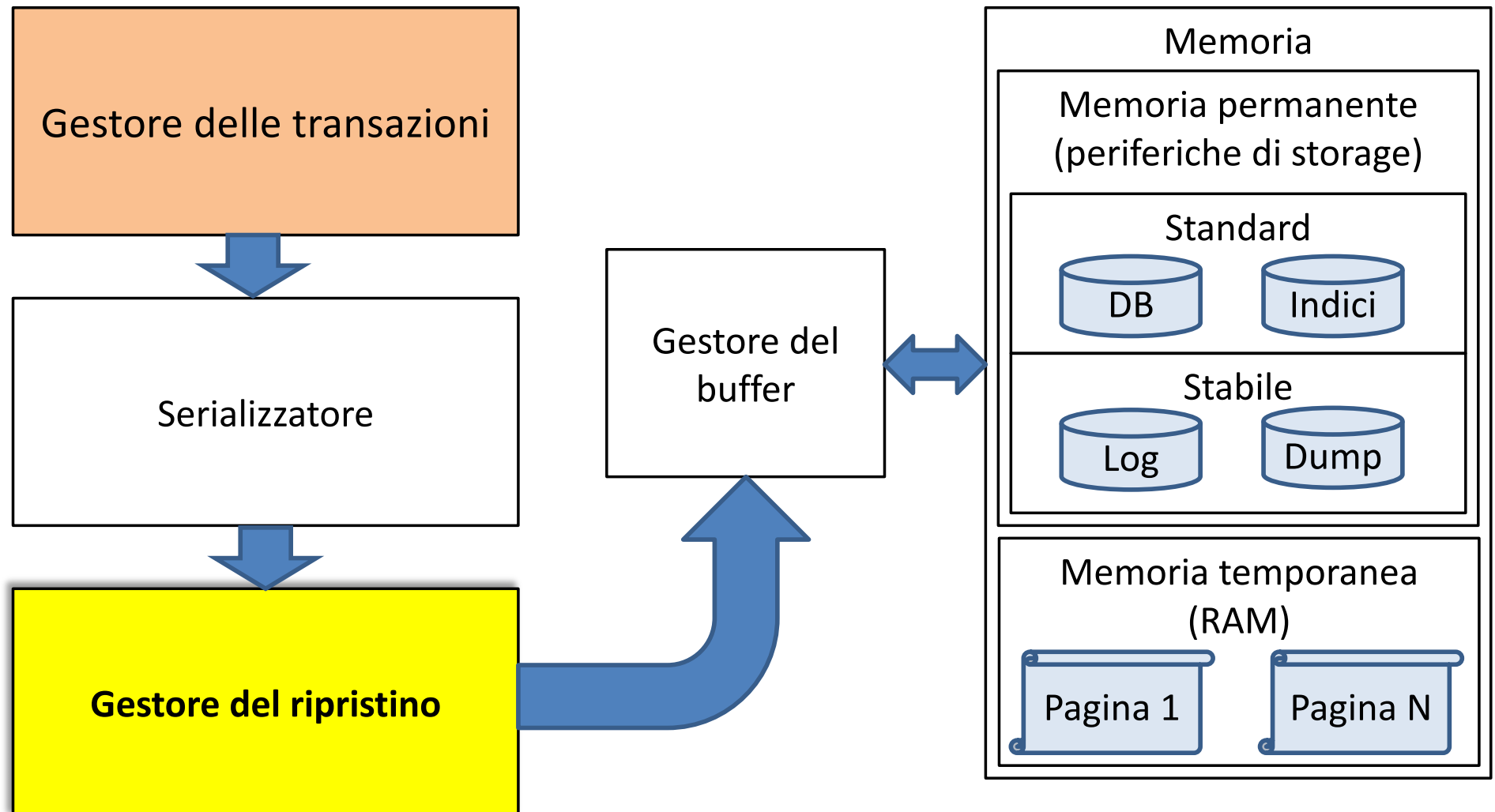
Basi di Dati

Architettura dei DBMS: gestione del ripristino

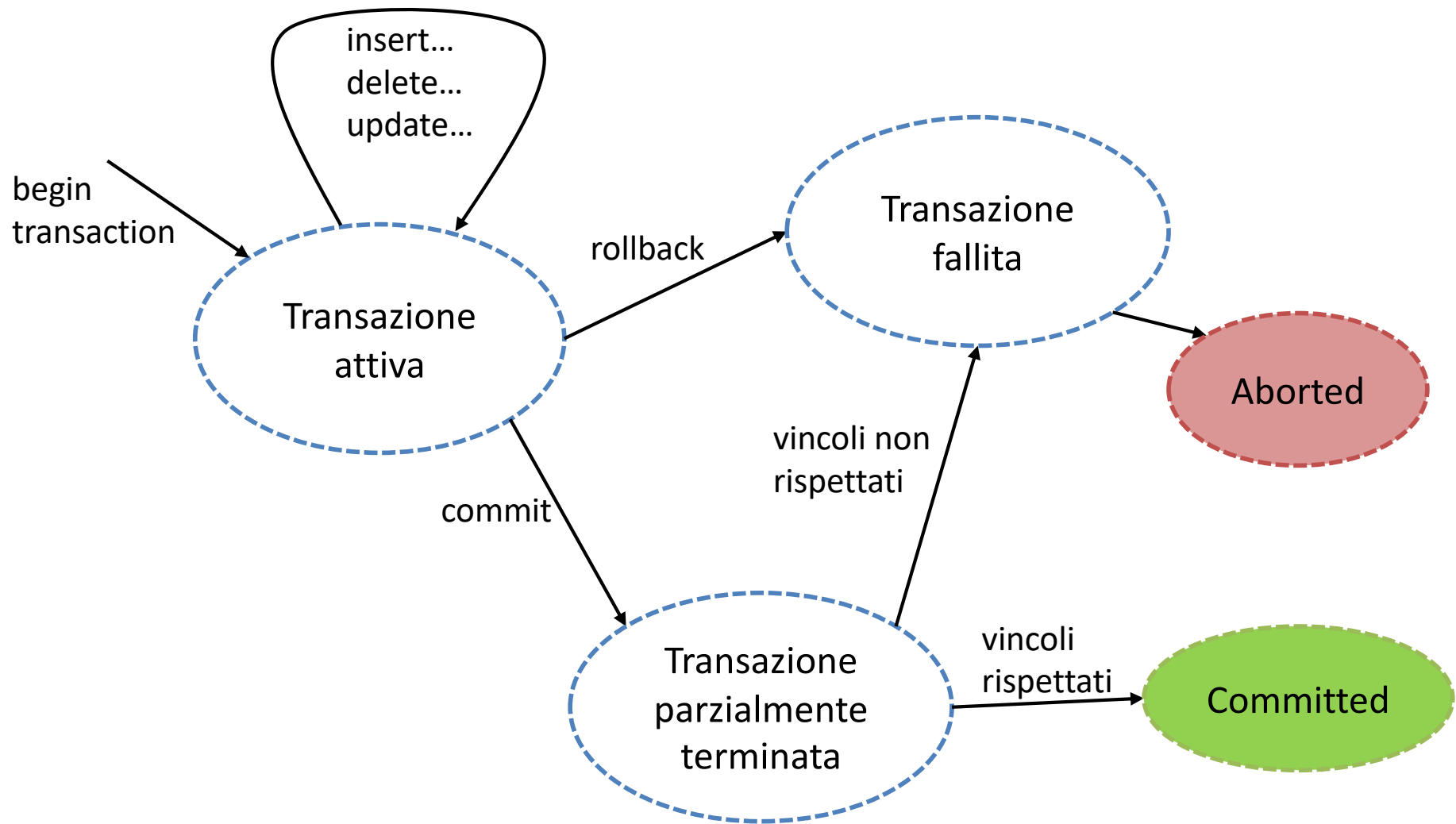
Contenuti - Roadmap

Lab (progettazione)	Corso di Teoria	Lab (SQL)
<ul style="list-style-type: none">• Metodologie e modello Entità Associazioni• Progettazione concettuale e logica	<ul style="list-style-type: none">• Modello Relazionale• Algebra relazionale• Ottimizzazione logica• Calcolo relazionale• La normalizzazione• Metodi di accesso e indici• Gestione della concorrenza• Gestione del ripristino	<ul style="list-style-type: none">• Linguaggio SQL

Architettura dei DBMS



Ciclo di vita di una transazione



Gestore del ripristino

Il **gestore del ripristino** affronta il problema dell'affidabilità.

Il sistema deve garantire i dati a fronte di possibili **guasti** e **anomalie**:

*Guasti **soft** (più comuni):*

1. Crash di sistema

- Guasto hardware o software o di rete durante l'esecuzione di una transazione (ad es. guasto della memoria principale). I guasti alle periferiche di storage non vengono considerati in questa voce.

2. Errori di programma o di sistema

- Una transazione fallisce per un errore (es. divisione per zero), per un bug o perché l'utente la interrompe.

3. Eccezioni gestite dalla transazione

- Ad es. saldo insufficiente in un conto corrente. Non sono errori perché sono stati previsti dallo sviluppatore.

...

Gestore del ripristino

...

4. *Rollback di transazioni forzate dal gestore della transazione e dal serializzatore*

- Per es. per violazione della serializzabilità o per uscire da un deadlock.

*Guasti **hard** (meno comuni):*

5. *Guasti delle periferiche di storage*

- Possono anche avvenire durante un'operazione di lettura o scrittura della transazione.

6. *Eventi catastrofici*

- Per es. furti, incendi, allagamenti, guasti all'aria condizionata o all'alimentazione...

Gestore del ripristino

- Il **gestore del ripristino** è la componente del DBMS che garantisce *atomicità e durabilità*.
- **Atomicità**: se una transazione fallisce durante l'esecuzione, il gestore del ripristino deve «annullare» eventuali modifiche apportate al DB.
- **Durabilità**: se una transazione è committed, i suoi effetti devono essere conservati stabilmente in memoria secondaria.
- Il **gestore del ripristino** gestisce l'affidabilità a fronte di errori **tranne nel caso dei guasti hard** (guasti delle periferiche e eventi catastrofici).
- Considereremo in seguito come fare fronte ai guasti hard.

File di log

- Per garantire il ripristino, il **gestore del ripristino** mantiene un **file di log**.
- Il log è una sorta di «diario di bordo» che tiene traccia delle operazioni effettuate sul DB da parte delle transazioni (scritture, cancellazioni, ...).
- È un **file sequenziale append-only** memorizzato su storage, quindi non può **essere affetto dai guasti 1-4**, ma solo da guasti hard (cioè guasti allo storage e eventi catastrofici).
- Ne viene fatto **periodicamente il backup**.
- Rende possibile **effettuare *undo* e *redo*** delle operazioni delle transazioni.

File di log

- Idea di base nella gestione dei guasti:
 - **Annullare** le operazioni memorizzate su storage ma **non committed**.
 - **Ripristinare** le operazioni non memorizzate su storage che sono **committed**.

File di log

- Nel log, quindi, per ogni transazione teniamo traccia dei comandi **start transaction** / **commit transaction** / **abort transaction**.
- E per ogni operazione (**insert/delete/update***) effettuata sul DB $\langle T_i, X, BS(X), AS(X) \rangle$, che contiene:
 - l'identificativo della transazione.
 - l'oggetto X su cui è stata effettuata l'operazione (per es. un **record**).
 - **before state**: lo stato precedente di X (tranne nel caso di insert).
 - **after state**: lo stato successivo di X (tranne nel caso di delete).

* anche read se il log viene usato anche per auditing.

Struttura del log

Esempio di un log



<**T1**, START>

<**T2**, START>

<**T2**, A, $BS_0(A)$, $AS_0(A)$ >

<**T2**, COMMIT>

<**T1**, A, $BS_1(A)$, $AS_1(A)$ >

<**T3**, START>

<**T1**, COMMIT>

<**T3**, A, $BS_2(A)$, $AS_2(A)$ >

Undo e redo di un'azione

- Usando il log si può fare undo e redo di *una singola azione* (*insert, delete, update*) in questo modo...
- **Undo** di un'azione su un oggetto *X*:
 - *update, delete*: si scrive nell'oggetto *X* il valore del **before state** (*BS*)
 - *insert*: si elimina *X*
- **Redo** di un'azione su un oggetto *X*:
 - *insert, update*: si scrive nell'oggetto *X* il valore dell'**after state** (*AS*)
 - *delete*: si elimina *X*
- **Nota:** anche il processo di ripristino potrebbe fallire e quindi dovere essere rieseguito in seguito: in questo caso bisogna avere lo stesso risultato che si avrebbe se il processo non fosse mai fallito.
- Quindi le operazioni *undo* e *redo* devono essere **idempotenti**, cioè $undo(undo(X))=undo(X)$ e $redo(redo(X))=redo(X)$.

Gestione buffer in memoria

- Tutte le operazioni su un DB vengono effettuate sui buffer in **memoria principale** e poi scritte su **memoria secondaria**.
- Anche **le scritture sul file di log** vengono in effetti apportate su un buffer in memoria principale e poi scritte in memoria secondaria.
- Questo ha **conseguenze sulla gestione del ripristino**.
- Infatti cosa succederebbe se una insert(X) venisse scritta su storage ma avvenisse un crash prima che anche il log venisse scritto su storage?
- Non potremmo disfare l'inserimento non committed!

Regole fondamentali per il log

- Quando si scrive il log su storage?
- Vanno rispettate due regole:
 1. **Write-Ahead Log**: Il *before state* dei record di log deve essere scritto *prima dei corrispondenti record* della base di dati.
 - Dato che per ogni modifica abbiamo la garanzia di avere il valore precedente, questa regola consente di **annullare (UNDO)** le operazioni.
 2. **Commit-Precedenza**: L'*after state* dei record di log deve essere scritto *prima di effettuare il commit*.
 - Questa regola **consente di rifare (REDO)** le scritture decise da una transazione ma eventualmente non riportate su storage.

Regole fondamentali per il log

- **Scrittura dei record di commit.** Quando una transazione richiede il **commit**, il DBMS sceglie in modo atomico e indivisibile tra abort e commit e scrive sul log in modo **sincrono (primitiva force) il record di commit.**
- Un guasto che si verifica prima del commit impone al gestore del ripristino l'undo delle azioni effettuate.
- Un guasto che si verifica dopo il commit impone il redo delle azioni effettuate.

Rollback

- Una transazione T_i può fallire:
 - quando richiede un **rollback**,
 - quando il DBMS impone una **abort** o
 - quando la transazione richiede un **commit** e almeno un vincolo non è soddisfatto.
- In questi casi il DBMS esegue un **rollback** della transazione: esegue un'azione detta **UNDO(T_i)** che disfa tutte le azioni precedentemente compiute dalla transazione T_i .
- Al termine dell'esecuzione di UNDO, nel log verrà memorizzato il record **< T_i , abort>** seguito dalla direttiva **FORCE LOG**, che forza la scrittura del log su storage.

UNDO/REDO

Quando cade il sistema e si riprende l'attività, alcune transazioni $AT = \{T_i \dots T_h\}$ (*insieme delle transazioni attive*) si trovano in uno stadio di elaborazione non terminato.

Altre transazioni $CT = \{T_j \dots T_k\}$ (*insieme delle transazioni committed*) invece sono terminate.

In prima approssimazione, il ripristino in seguito a un crash avviene con due operazioni:

- **UNDO(AT): annullamento** delle transazioni non terminate per garantire la proprietà di atomicità.
- **REDO(CT): ripristino** delle transazioni committed.

UNDO(T_1, \dots, T_h)

Si esplora il file di log **a ritroso**.

UNDO(T_1, \dots, T_h):

per ogni $\langle T_i, X, BS(X), AS(X) \rangle$ a ritroso tale che $T_i \in \{T_1, \dots, T_h\}$

/ ripristina $BS(X)$, cioè: */*

leggi la pagina che contiene X

modifica nel buffer la pagina usando $X := \mathbf{BS(X)}$

FORCE della pagina

REDO(T_1, \dots, T_k)

Si esplora il file di log **in avanti**

REDO(T_1, \dots, T_k):

per ogni $\langle T_i, X, BS(X), AS(X) \rangle$ in avanti tale che $T_i \in \{T_1, \dots, T_k\}$

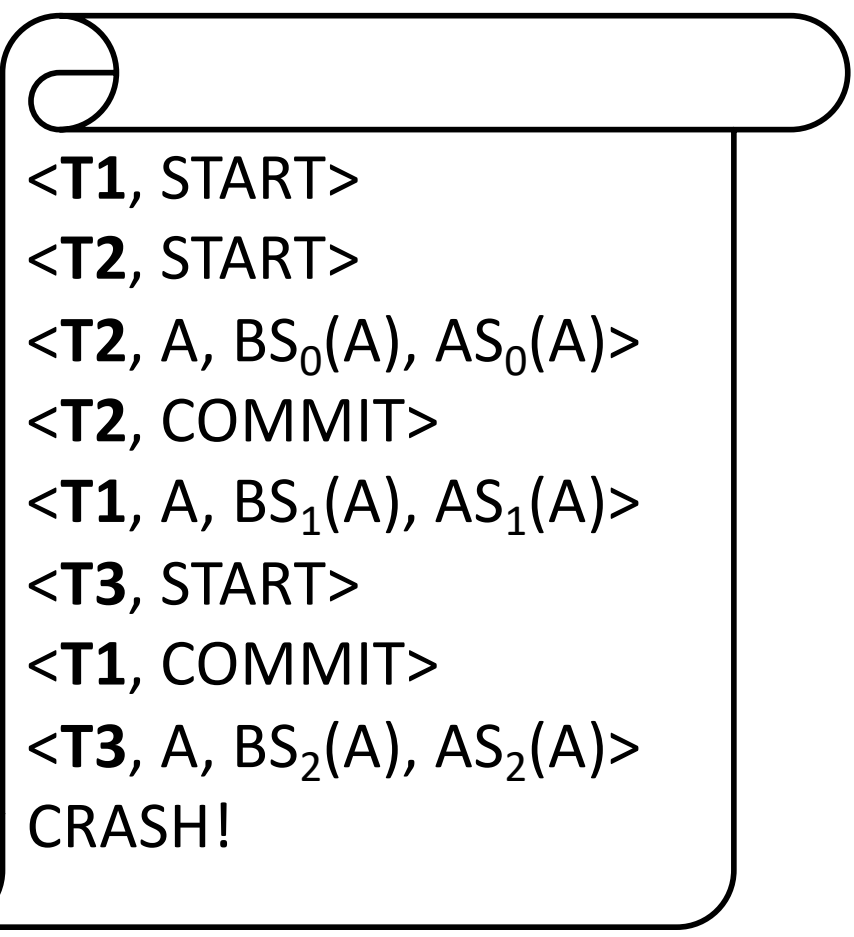
/ rendi persistente AS(X), cioè: */*

leggi la pagina che contiene X

modifica nel buffer la pagina usando $X := \mathbf{AS(X)}$

FORCE della pagina

Esempio di file di log



<T1, START>
<T2, START>
<T2, A, BS₀(A), AS₀(A)>
<T2, COMMIT>
<T1, A, BS₁(A), AS₁(A)>
<T3, START>
<T1, COMMIT>
<T3, A, BS₂(A), AS₂(A)>
CRASH!

1. T2 è la prima transazione che modifica l'oggetto A con before state BS₀(A) e after state AS₀(A)
2. La transazione T2 va in **commit**, quindi la modifica apportata ad A da T2 ha effetto sulle operazioni successive
3. Quando T1 modifica A, il suo before state sarà quindi uguale all'after state precedente, cioè BS₁(A)=AS₀(A) di T2
4. La transazione T3 entra nel sistema
5. T1 va in **commit**
6. T3 modifica l'oggetto A (con BS₂(A)=AS₁(A)), ma non riesce ad andare in commit a causa di un **crash** (infatti nel log non c'è il commit di T3)

Vediamo come **ripristinare** lo stato corretto del DB...

Lettura a ritroso

- Le **transazioni attive** (cioè quelle non terminate) sono $AT = \{T3\}$.
- Per le transazioni attive bisogna esplorare il log **all'indietro** per **annullarne** gli effetti e riportare la base di dati allo stato precedente alle modifiche.
- UNDO considera quindi a ritroso le transazioni in AT riportando gli oggetti al loro valore nel before state.

Lettura in avanti

- Le **transazioni committed** sono $CT=\{T2,T1\}$.
- Per le transazioni committed bisogna esplorare il log **in avanti** perché il ripristino deve portare la base di dati allo stato dell'ultima modifica.
- REDO rispetta esattamente questo ordine quando il file di log viene letto in avanti.

Algoritmo di ripristino (**UNDO/REDO**)

Quando avviene un crash il sistema cade e poi dovrà ripartire senza nessuna transazione in corso.

Il ripristino prevede i seguenti passi:

1. **AT** := insieme delle transazioni non terminate
2. **CT** := insieme delle transazioni che hanno raggiunto il commit
3. **UNDO(AT)**
4. **REDO(CT)** (sempre dopo la UNDO)

Si esegue prima UNDO per riportare il sistema allo stato originale e poi REDO per riapplicare le operazioni con commit.

Ripristino delle transazioni in abort?

Nel ripristino non abbiamo parlato delle transazioni in abort.

Infatti, grazie al fatto che, se nel log c'è un record di **ABORT**, il rollback è già stato eseguito e le modifiche sono state rese persistenti, il **processo di ripristino può ignorare le transazioni abortite.**

Ottimizzazione del ripristino

- Nei grossi sistemi informativi, il file di log può contenere centinaia di migliaia di record e, se avvenisse un crash di sistema, il **ripristino** **risulterebbe estremamente costoso.**
- Anche se non sono frequenti, i crash possono avvenire in qualsiasi momento, **anche in pieno orario di attività** ed è necessario ridurre al minimo i tempi necessari al ripristino.
- Infatti il ripristino richiede il blocco dell'attività transazionale, comportando, ad esempio, il blocco di tutti gli sportelli del bancomat.

Checkpoint

La soluzione consiste nell'introduzione, nel file di log, di un passo detto **checkpoint**.

Il checkpoint garantisce che il processo di ripristino possa **considera solo una parte del file di log**.

<T1, START>

<T2, START>

<T2, A, BS₀(A), AS₀(A)>

<T2, COMMIT>

<T1, A, BS₁(A), AS₁(A)>

<T3, START>

...

----- checkpoint -----

...

<T1, COMMIT>

<T3, A, BS₂(A), AS₂(A)>

Checkpoint

Periodicamente (ad es. ogni 10/15 minuti) avviene un processo di checkpoint che aggiunge un record al file di log:

1. Si **sospendono tutte** le transazioni
2. Si costruisce il record di checkpoint contenente l'elenco delle transazioni che in quel momento sono **attive** col relativo **puntatore** alla posizione dello start della transazione nel file di log
3. Si esegue un **FORCE LOG**
4. Si esegue un **FORCE** delle pagine delle transazioni committed
5. Si aggiunge un flag di **OK** nel record di checkpoint e si esegue un nuovo **FORCE LOG**
6. Si riavviano le transazioni sospese

Nel ripristino sarà quindi inutile scorrere la porzione di log precedente al checkpoint. Infatti:

- grazie al passo 4 si possono ignorare le transazione committed.
- grazie al passo 2 si possono ricavare le sole transazioni attive da ripristinare.

Rimangono da considerare le transazioni dopo il checkpoint.

Creazione del checkpoint: esempio

- **T1** e **T3** sono transazioni iniziate ma non terminate, quindi vengono aggiunte al record di checkpoint con i relativi puntatori
- **T2** è iniziata e terminata, quindi non viene aggiunta al record
- Il record di log è reso persistente con una **FORCE LOG**
- Le modifiche di **T2** vengono rese persistenti con una **FORCE** delle sue pagine
- Si aggiunge il flag **OK**
- Si esegue un nuovo **FORCE LOG**

<T1, START>
<T2, START>
<T2, A, BS₀(A), AS₀(A)>
<T2, COMMIT>
<T1, A, BS₁(A), AS₁(A)>
<T3, START>

cp:	T1, p	T3, p	OK
-----	-------	-------	----

Ripristino con checkpoint

- Il **ripristino con checkpoint** considera l'ultimo checkpoint, che dà l'elenco delle transazioni attive in quel momento (**T1** e **T3**).
- ...

<T1, START>
<T2, COMMIT>
<T1, A, BS₁(A), AS₁(A)>
<T3, START>

cp:	T1, p	T3, p	OK
-----	-------	-------	----

<T1, COMMIT>
<T4, START>
<T5, START>
<T4, A, BS₃(A), AS₃(A)>
<T4, COMMIT>
<T3, A, BS₄(A), AS₄(A)>
<T5, B, BS₅(B), AS₅(B)>

Ripristino con checkpoint

- Per le transazioni che hanno raggiunto il commit **prima** del checkpoint (**T2**) non c'è bisogno di ripristino (stiamo considerando solo guasti soft).
- Per le transazioni AT non ancora terminate sia iniziate prima del checkpoint (**T3**) che dopo (**T5**) si fa UNDO.
- Per le transazioni CT con il commit dopo il checkpoint sia iniziate prima del checkpoint (**T1**) che dopo (**T4**) si fa REDO.

1

<T1, START>
<T2, COMMIT>
<T1, A, BS₁(A), AS₁(A)>
<T3, START>

cp:	T1, p	T3, p	OK
-----	-------	-------	----

<T1, COMMIT>
<T4, START>
<T5, START>
<T4, A, BS₃(A), AS₃(A)>
<T4, COMMIT>
<T3, A, BS₄(A), AS₄(A)>
<T5, B, BS₅(B), AS₅(B)>

2 (crash!)

Ripresa a caldo (hot restart)

1. Trova l'ultimo checkpoint
2. Recupera la lista **AT** delle transazioni ancora **attive** **durante il crash**
3. Recupera la lista **CT** delle transazioni che hanno raggiunto il commit **dopo l'ultimo checkpoint**
4. **UNDO(AT)**
5. **REDO(CT)**

Ripristino con checkpoint

Insieme delle transazioni attive durante il crash:

$$AT = \{T3, T5\}$$

Insieme delle transazioni con commit dopo il checkpoint:

$$CT = \{T1, T4\}$$

Ripristino:

1. **UNDO**(T3, T5)
2. **REDO**(T1, T4)

<T1, START>
<T2, COMMIT>
<T1, A, BS₁(A), AS₁(A)>
<T3, START>

cp:	T1, p	T3, p	OK
-----	-------	-------	----

<T1, COMMIT>
<T4, START>
<T5, START>
<T4, A, BS₃(A), AS₃(A)>
<T4, COMMIT>
<T3, A, BS₄(A), AS₄(A)>
<T5, B, BS₅(B), AS₅(B)>
(crash!)

Ripristino da eventi catastrofici

- I guasti hard (fallimento dello storage e eventi catastrofici) richiedono l'utilizzo di memorie secondarie considerate "**stabili**", che consideriamo esenti da guasto perché hanno una **robustezza maggiore** rispetto alla memoria secondaria standard.

Memoria stabile

- Il concetto di **memoria stabile** è però teorico in quanto non esiste memoria esente **da guasto**.
- Si può realizzare concretamente la memoria stabile **approssimando la proprietà di robustezza con la tecnica della duplicazione**, cioè copiando i dati su nastro magnetico o storage offline possibilmente situati in luoghi sicuri fisicamente separati e geograficamente distanti.
- Per rendere efficiente la gestione della memoria stabile, le informazioni da memorizzare devono essere **di dimensioni contenute**.

Dump

- **Copia completa ("di riserva", backup) della base di dati**
 - Solitamente prodotta mentre il sistema **non è operativo**.
Quando viene prodotta quando il sistema è attivo, si chiama *hot backup*.
 - **Salvata in memoria stabile**, come il nastro, possibilmente in un posto geograficamente distante.
 - Si svuota il **file di log** e si aggiunge un **record di dump** che indica il momento in cui il dump è stato effettuato (e dettagli pratici come file, dispositivo, ...).
- **Il file di log** solitamente è molto più piccolo della base di dati e **si può memorizzare direttamente in memoria stabile**.

Ciclo operativo del sistema e log

- Nei sistemi informativi reali si può riscontrare un ciclo di vita dell'attività transazionale che dura circa 24 ore.
- Esiste un momento (di solito di notte) in cui l'attività della base di dati è minima.
- In questo momento la base di dati si considera corretta: si può interrompere l'attività, rendere persistenti tutte le modifiche delle transazioni committed, annullare le altre e azzerare il file di log.

Ripresa a freddo (cold restart)

Per ripristinare la base di dati dopo un guasto hard si effettua una **ripresa a freddo** usando i backup:

1. Si ripristinano i dati a partire dai backup (**restore**)
2. Si legge **tutto il log** e si effettua il **REDO(CT)**
3. Si effettua la ripresa a caldo