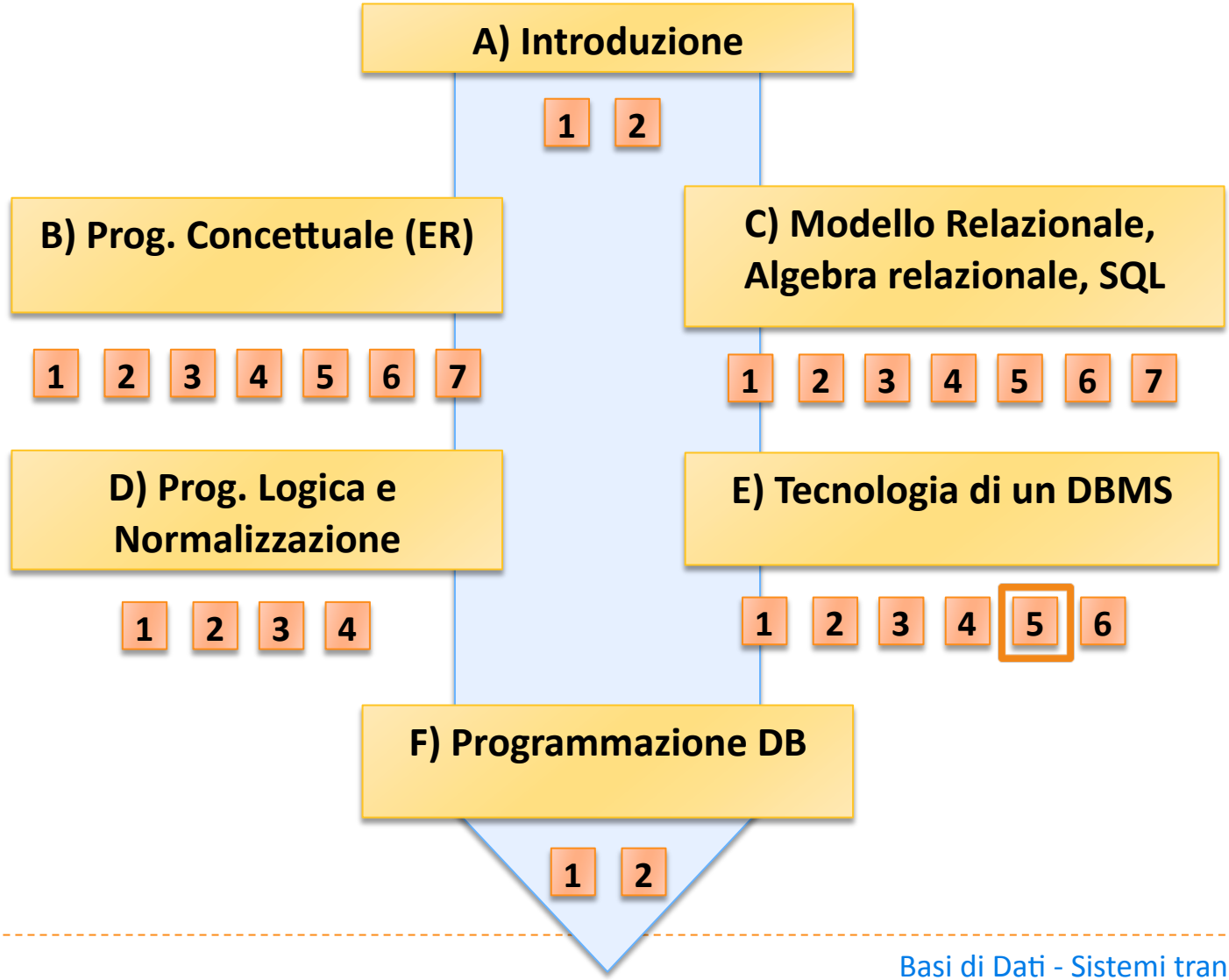
Basi di Dati

Sistemi transazionali

Basi di Dati – Dove ci troviamo?



Ricordiamo le principali caratteristiche dei DBMS

- condivisione dei dati
  - concorrenza
- qualità dei dati
  - integrità
- efficienza
  - caricamento, query, sort
- controllo dell'accesso
  - privatezza
- robustezza

↓

fuoco
su integrità,
concorrenza,
robustezza, efficienza

Il concetto di transazione

TRANSAZIONE (transaction): Complesso di OPERAZIONI tendenti a portare il DB da uno stato corretto ad un altro stato corretto.

La transazione è un'unità di elaborazione che gode di 4 proprietà (ACID):

- **Atomicità**
- **Consistenza**
- **Isolamento**
- **Durata (persistenza)**

Atomicità

ATOMICITÀ (atomicity) delle transazioni: le operazioni previste costituiscono un tutto unico, devono pertanto essere eseguite nella loro interezza o non essere eseguite per niente.

SERIALIZZAZIONE DELLE OPERAZIONI: Le operazioni eventualmente svolte in parallelo devono portare il DB ad uno stato equivalente all'esecuzione seriale delle medesime operazioni.

La transazione è quindi una TRASFORMAZIONE ATOMICA dallo stato iniziale a quello finale.

Atomicità

Una TRANSAZIONE costituisce un BLOCCO DI RECOVERY (ripristino) cioè: un insieme di operazioni delimitate da istruzioni al fine di permettere le operazioni :

UNDO(disfare): in caso di fallimento della transazione deve essere possibile "disfare" l'azione svolta sui dati

REDO(rifare): se la transazione ha avuto successo ma le modifiche al DB non sono state rese permanenti, le modifiche vanno ripetute.

Transazione "ben formata"

BEGIN TRANSACTION

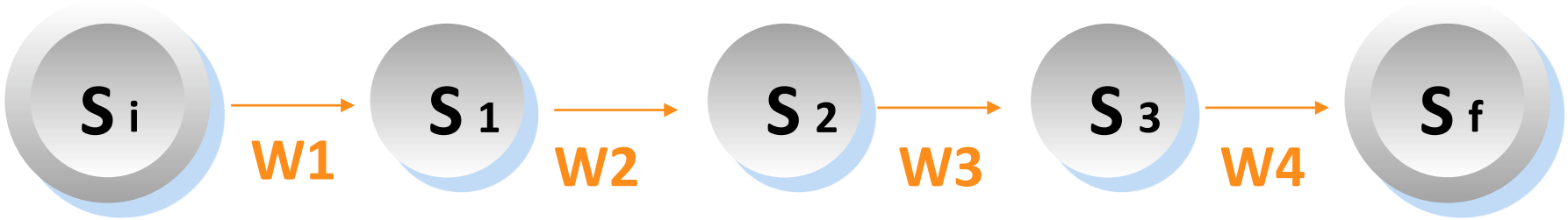
codice con manipolazione dei dati

(letture e scritture)

COMMIT WORK / ROLLBACK WORK

codice privo di manipolazione di dati

END TRANSACTION

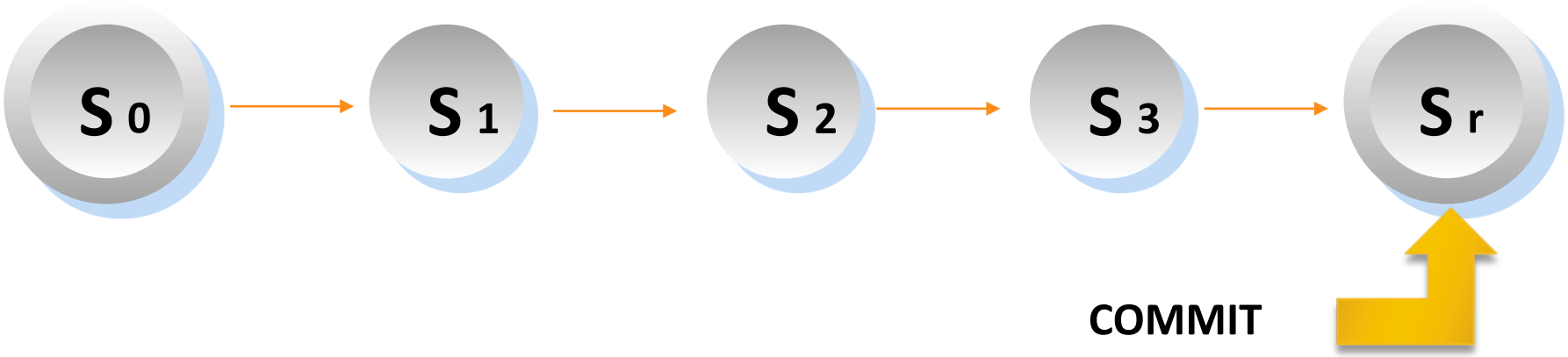


Atomicità

Comportamenti possibili:

commit work : successo

COMMIT: fine corretto della parte di modifica della transazione ("impegno" del DBMS a trasferire i dati modificati in memoria permanente)



Recovery

In caso di guasto il sistema di RECOVERY è incaricato di riportare il DataBase ad uno stato corretto precedente al guasto.

STATO CORRETTO è uno stato del DB che non riflette i cambiamenti dovuti a transazioni di modifica che non sono state terminate con successo.

CAUSE DI GUASTO:

1) guasto all'interno di una transazione
2) guasto all'interno del sw di base
3) guasto sulla memoria secondaria
4) guasto sul sistema hardware

Recovery

GUASTO ALL'INTERNO DI UNA TRANSAZIONE

a) **ABORT**: una transazione si interrompe per un errore software e quindi il DBMS ne esegue il ROLLBACK ("ritorno indietro" delle modifiche apportate ai dati)

b) **ROLLBACK** di una transazione: una transazione può autonomamente chiedere il rollback se scopre inconsistenza nei dati o la non fattibilità delle operazioni

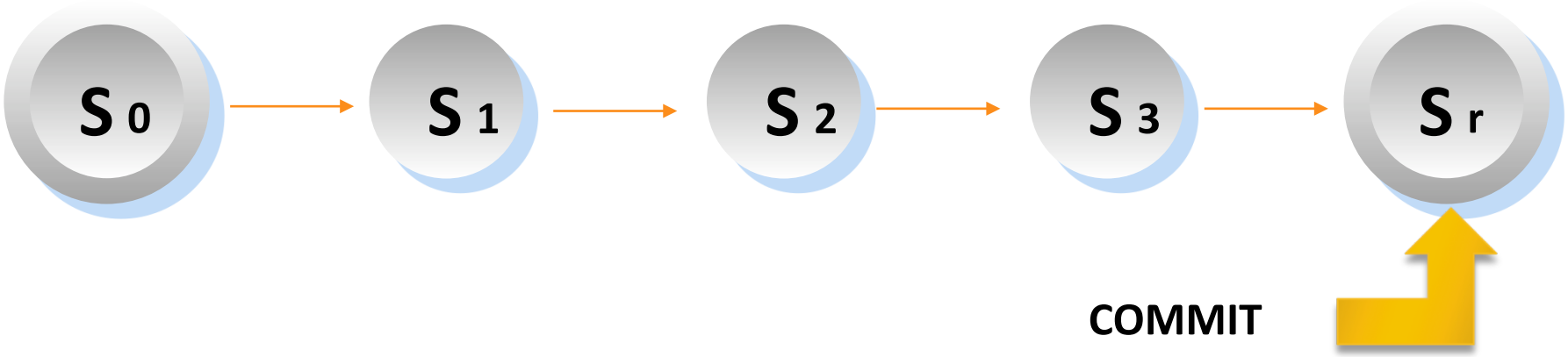
c) **ABORT FORZATO** di una transazione: una transazione "abortita" è forzata al rollback se il DBMS scopre violazione di vincoli o di autorizzazioni, situazioni di stallo (deadlock), fallimento di altre transazioni coordinate in parallelo.

Atomicità

Comportamenti possibili:

commit work : successo

COMMIT: fine corretto della parte di modifica della transazione ("impegno" del DBMS a trasferire i dati modificati in memoria permanente)

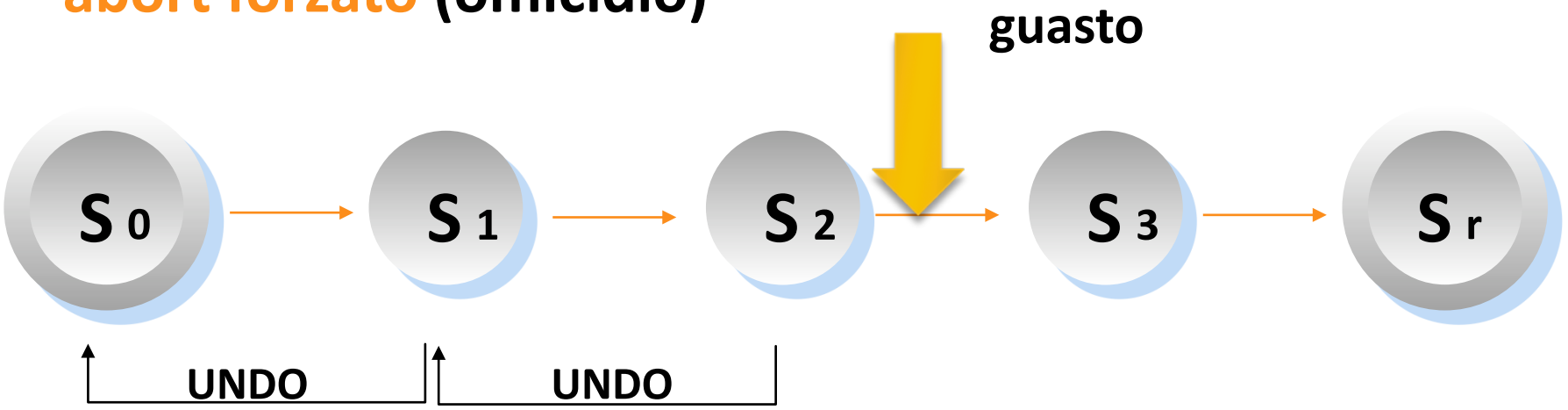


Atomicità

Comportamenti possibili:

rollback work (suicidio)

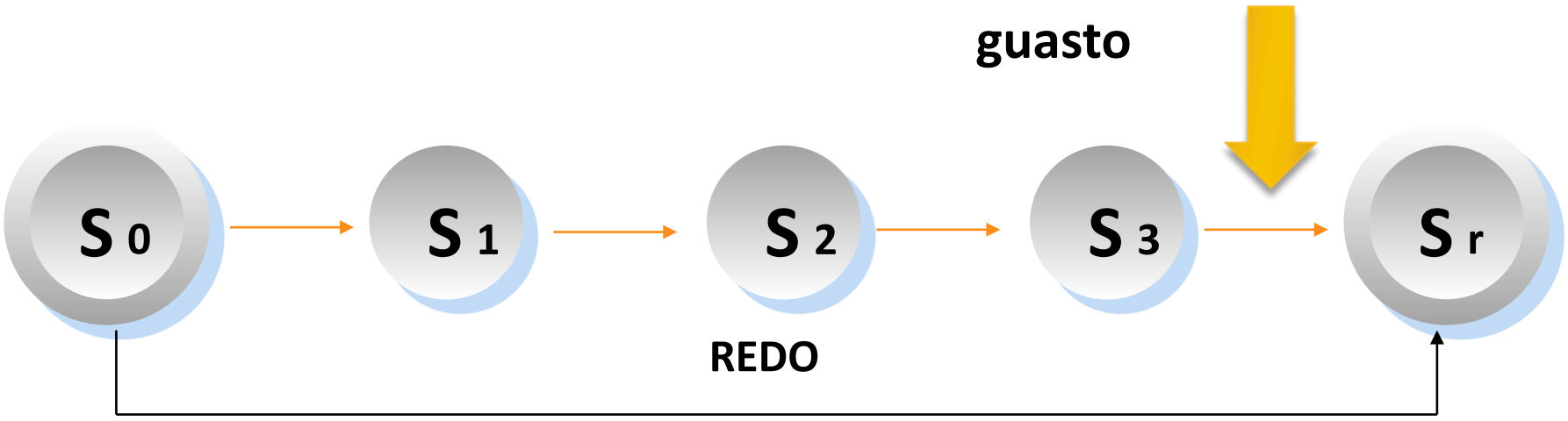
abort forzato (omicidio)



Atomicità

Comportamenti possibili:

commit work : successo della transazione ma guasto dopo il commit e prima della conclusione



Consistenza

La transazione rispetta i vincoli di integrità, come conseguenza: se lo stato iniziale è corretto anche lo stato finale è corretto

Isolamento

La transazione è isolata dalle altre transazioni concorrenti (non espone i suoi stati intermedi prima della sua conclusione). Si evita l' "effetto domino"

Persistenza

Gli effetti di un commit work dureranno "per sempre" (indipendentemente dai guasti del sistema)

Per garantire le proprietà acid

- controllo di affidabilita' : atomicità, persistenza
- controllo di concorrenza : isolamento
- controllo di esecuzione : consistenza

- Controllo di affidabilità sul database server
- gestione ingresso-uscita
- gestione memorie
- gestione buffer
- gestione commit work / rollback work
- gestione giornale

Persistenza delle memorie

- **memoria centrale :**

  non è persistente

- **memoria di massa :**

  è persistente ma può danneggiarsi

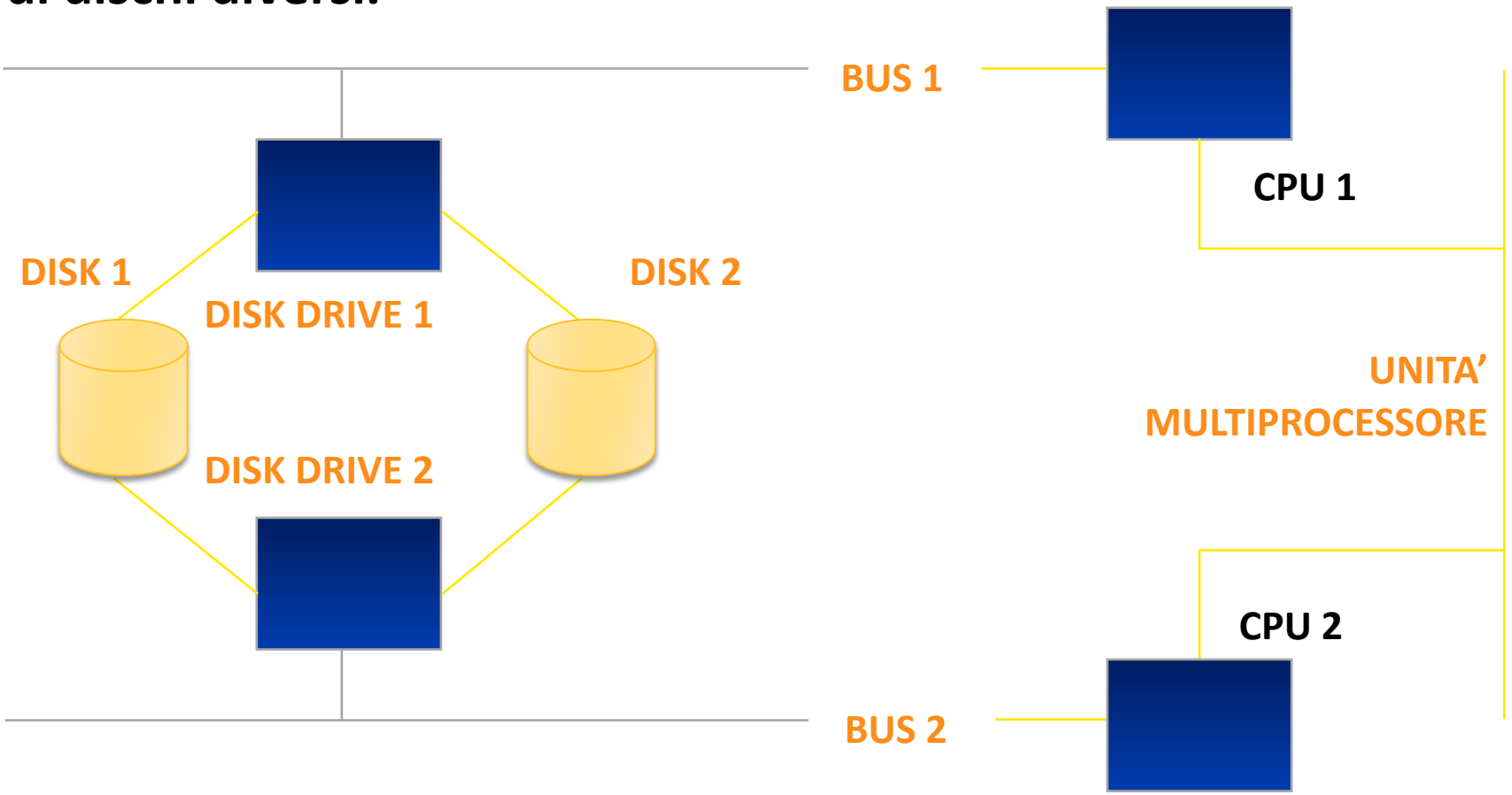
- **memoria stabile :**

  memoria che anche se si danneggia non perde i dati e non interrompe il servizio:

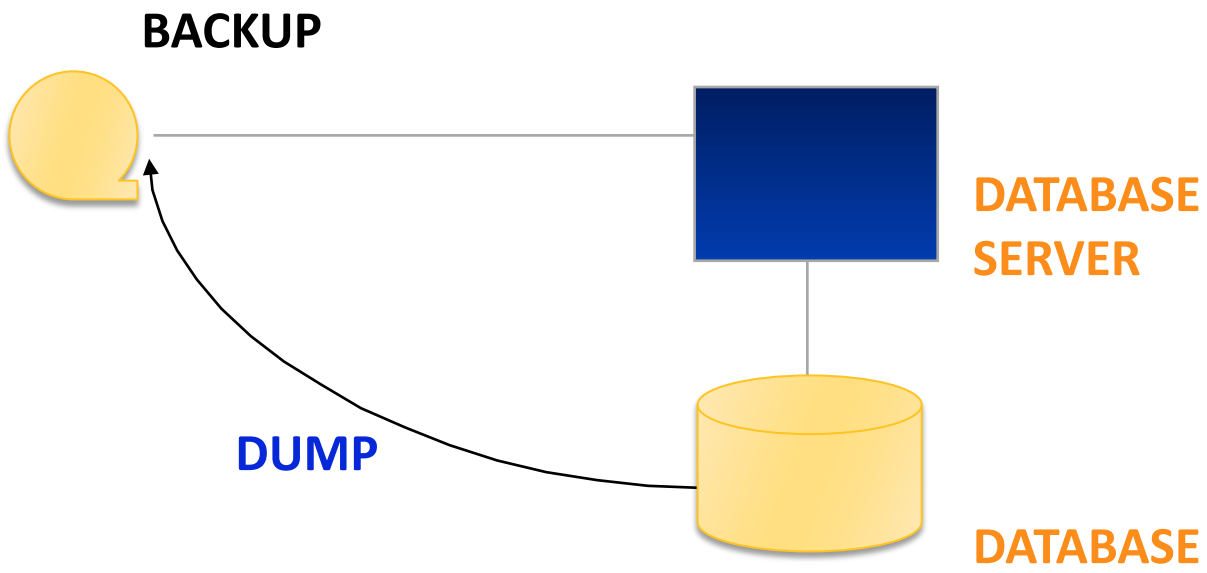
  DISK MIRRORING e RAID

Replicazione on-line: mirroring

Le informazioni vengono parallelemente riportate su sistemi di dischi diversi.

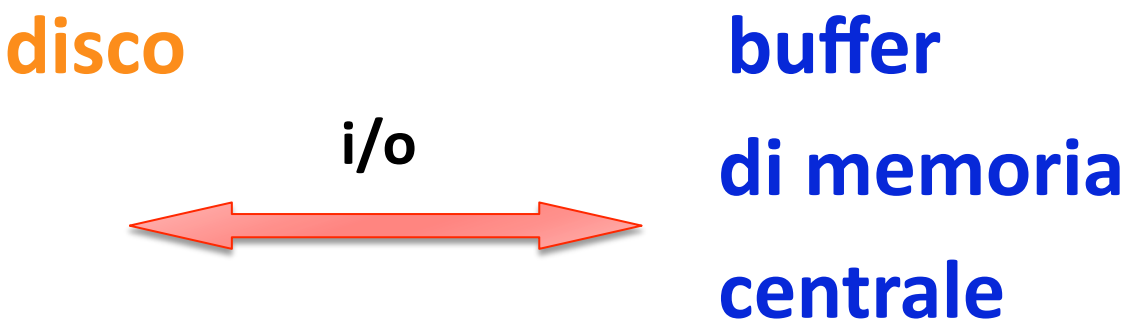


Replicazione off-line : unità di backup



Il DBMS tiene sempre una vecchia copia del DB su un altro dispositivo (nastro o disco) aggiornata periodicamente (DUMP).

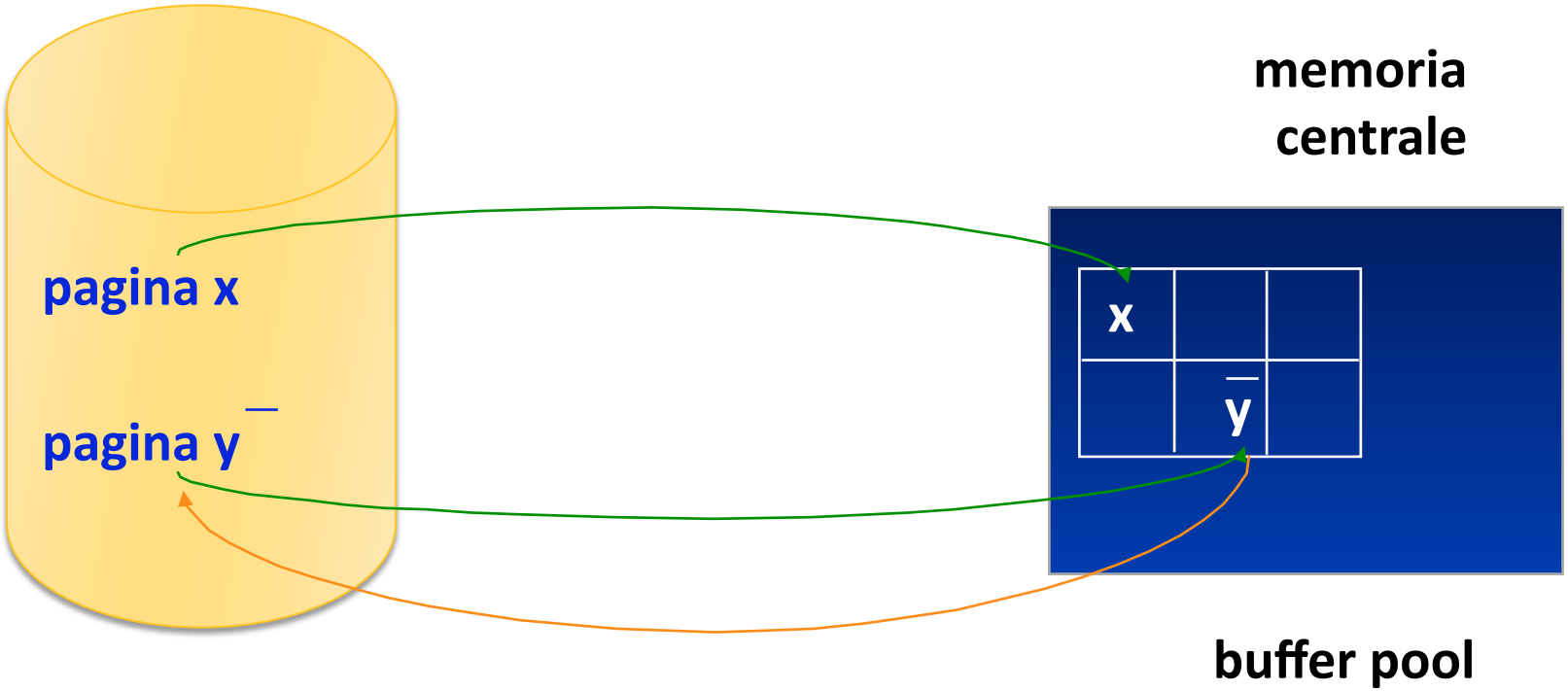
Gestione della memoria centrale



razionale:

- **uso frequente dei dati già nel buffer**
- **scrittura differita della base di dati ottimizzando le scritture su disco**

Uso della memoria centrale



Per ogni transazione c'è un certo numero di pagine disponibili nel buffer pool, il numero dipende dal numero di transazioni e dalle loro richieste.
Le pagine contenenti modifiche devono essere riscritte.

Politiche di gestione del buffer

a STEAL - pagine con dati dirty, cioè modificati ma ancora senza commit, sottratte a una transazione attiva e riportate su disco

NO STEAL

b FORCE - pagine scritte al commit-work

NO FORCE

Normalmente :

NO-STEAL, NO-FORCE

Le transazioni rilasciano le pagine alla fine, quelle modificate verranno riscritte in memoria permanente. Se necessario le pagine delle transazioni vengono rimpiazzate con la politica LRU (Least Recently Used).

Giornale di transazione (Log file)

Il LOG registra in memoria stabile le azioni svolte dalla transazione sotto forma di coppie di valori:

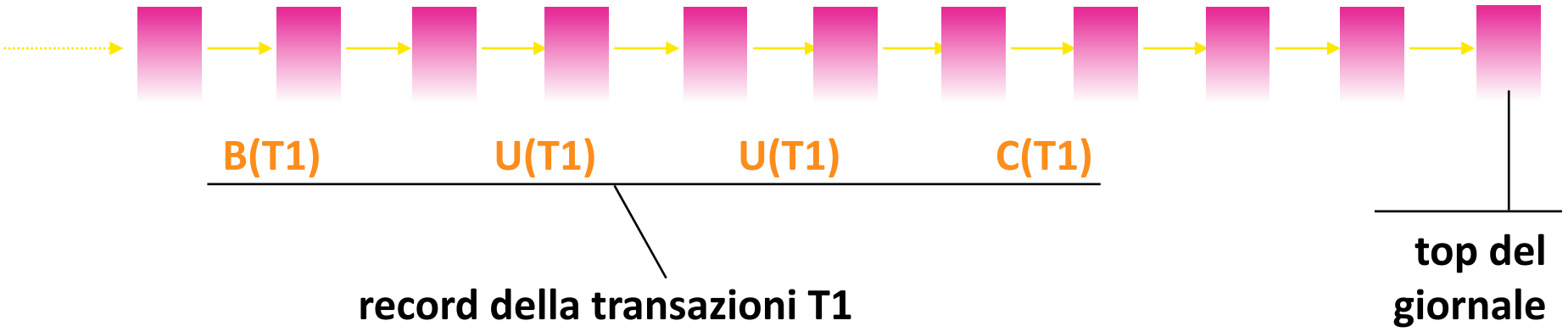
UPDATE (U) trasforma : val1 ⇒ val2

registrazione: BEFORE-STATE(U) = val1
AFTER-STATE(U) = val2
(chiamate anche before/after image)

TIPI DI REGISTRAZIONI NEL GIORNALE:

1. **BEGIN-TRANSACTION**
2. **UPDATE / DELETE / INSERT**
3. **COMMIT / ABORT**
4. **CHECKPOINT**

Il giornale è sequenziale



Le informazioni registrate sono del tipo :

1) identificatore della transazione;
2) codice di operazione;
3) numero di sequenza nel log;
4) puntatore all'ultimo log record della transazione;
5) identificatore del file ed indirizzo del record (tid);
6) vecchio valore, nuovo valore;

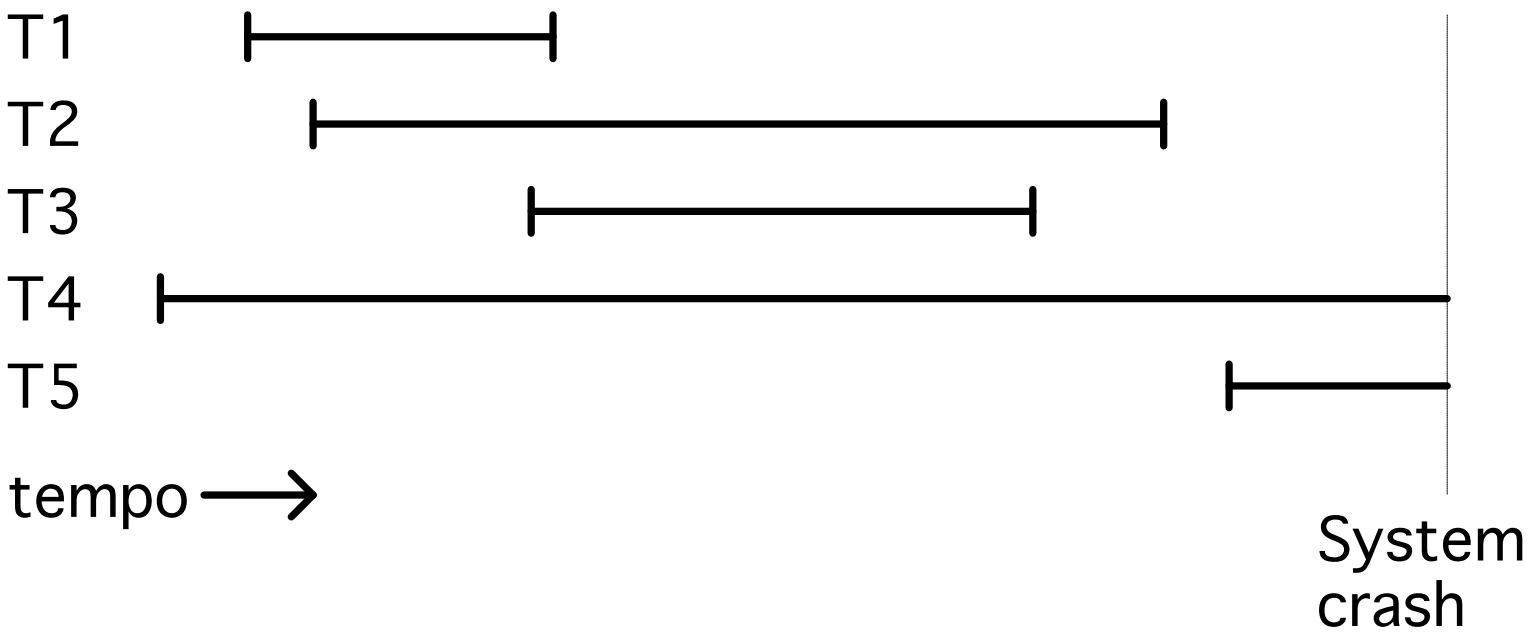
Giornale

Per il LOG (che contiene l'insieme di informazioni necessarie e sufficienti per riportare il DB in uno stato corretto) si utilizza un protocollo di tipo WAP (*Write Ahead Protocol*):

Si scrive un record di BEGIN sul LOG prima di iniziare l'esecuzione di una transazione
Le modifiche effettuate vengono registrate sul disco del LOG (chiamato anche audit trail) PRIMA di venire estese alla memoria secondaria

Il LOG è presente in tutti i DBMS di una certa importanza.
In sistemi che eseguono molte transazioni può occupare uno spazio molto elevato (LOG di centinaia di milioni di caratteri al giorno in sistemi commerciali)..

ESEMPIO



T1 , T2 e T3 sono OK. Per T4 e T5 il sistema deve riportare i dati allo stato corretto anteriore al loro inizio.

METODI PER LA GESTIONE DELLE TRANSAZIONI

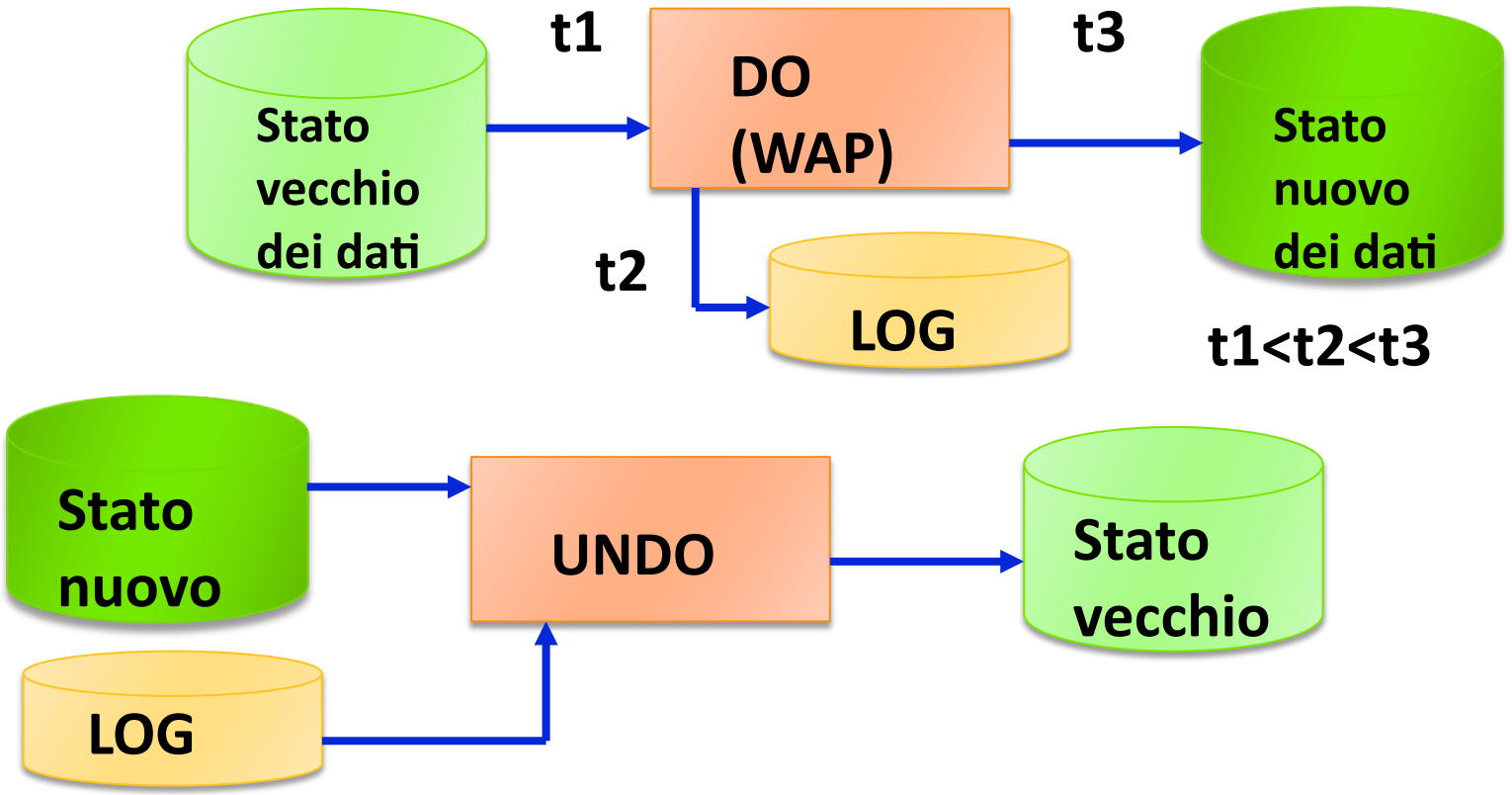
1) *UNDO* / *REDO*
2) *UNDO* / *NO REDO*
3) *NO UNDO* / *REDO*
4) *NO UNDO* / *NO REDO*

a) fare o non fare UNDO dipende dalla politica di gestione delle modifiche.

b) fare o non fare REDO dipende dalla gestione del BUFFER e del COMMIT. Il REDO va fatto perché anche se si raggiunge il COMMIT sul LOG, non si è sicuri che le pagine siano state tutte scaricati dal buffer.

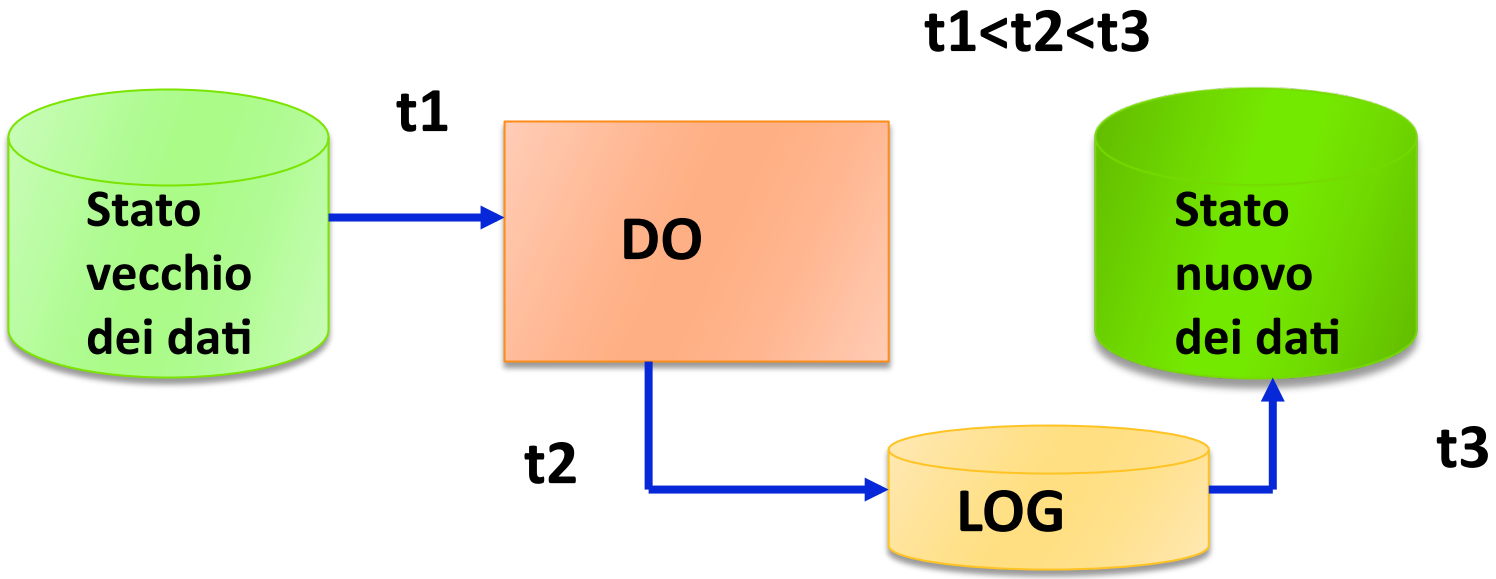
Politica di undo (gestione delle modifiche)

POLITICA a 1 FASE : (UNDO) le modifiche vengono portate subito sul DB durante lo svolgimento della transazione, prima della terminazione (STEAL).



Politica di undo (gestione delle modifiche)

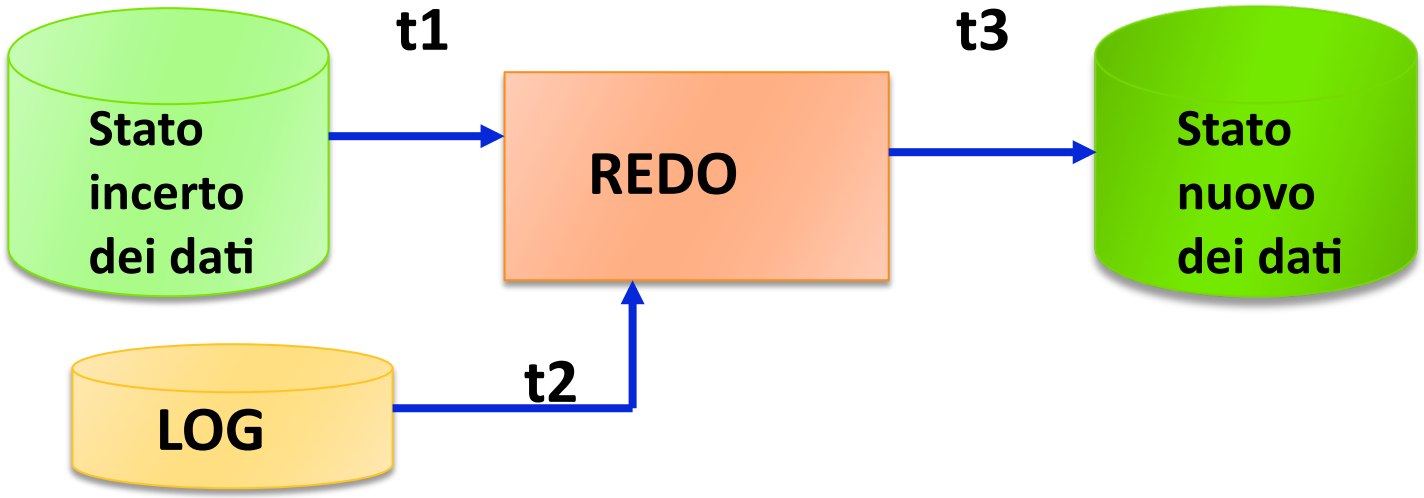
POLITICA a 2 FASI : (NO UNDO) tutte le modifiche sono registrate sul LOG temporaneamente e non sul DB;
solo se la transazione termina correttamente allora vengono portate sul DB stabile (con una operazione atomica) (NO STEAL).



Politica di commit (gestione del buffer)

COMMIT posticipato: (NO REDO) il commit è definitivo solo dopo che le modifiche sono migrate sul DB (prima le modifiche, poi il commit sul log) (FORCE)

COMMIT anticipato: (REDO) il commit è registrato subito sul LOG prima che le modifiche siano completate sul DB (NO FORCE).



Politiche di recovery

UNDO/REDO richiede before and after image,

- **lascia la gestione delle pagine al gestore del buffer che può ottimizzare il trasferimento su disco,**
- **migliora il funzionamento normale,**
- **peggiora il funzionamento sia in caso di guasti di sistema al RESTART che di guasti di transazioni,**
- **permette un più sollecito rilascio dei lock.**

UNDO/NO REDO richiede after image,

- **migliora il caso di RESTART ,**
- **peggiora il caso di guasto di transazione,**
- **peggiora il funzionamento normale del buffer poiché forza il gestore del buffer a scaricare le pagine per terminare la transazione,**
- **non permette un sollecito rilascio dei lock.**

Politiche di recovery

NO UNDO / REDO non richiede before image,

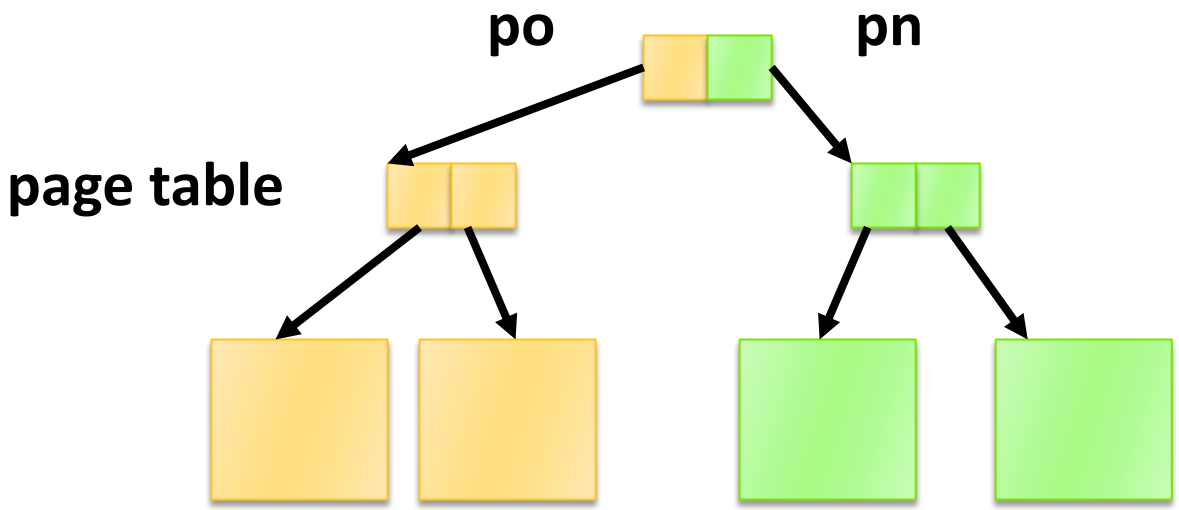
- **favorisce i casi di fallimenti delle transazioni.**

NO UNDO / NO REDO

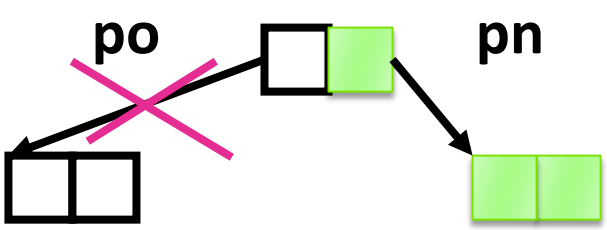
- **NO REDO: tutte le modifiche devono essere nel DB prima che la transazione sia considerata terminata.**
- **NO UNDO: nessuna modifica deve essere portata sul DB prima che la transazione sia considerata terminata.**
- **Perciò una operazione atomica deve trasferire i dati e registrare il Commit.**

Si usa la tecnica delle pagine ombra (SHADOW PAGES) che è molto veloce ma richiede molta memoria

SHADOW PAGES

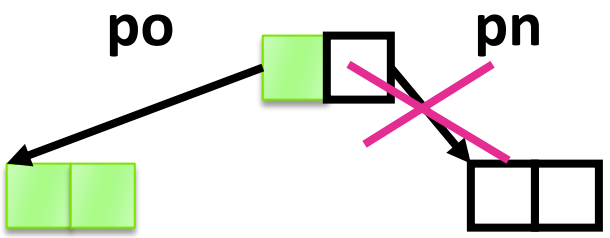


doppio puntatore dell'applicazione alla page table



commit

(operazione atomica)

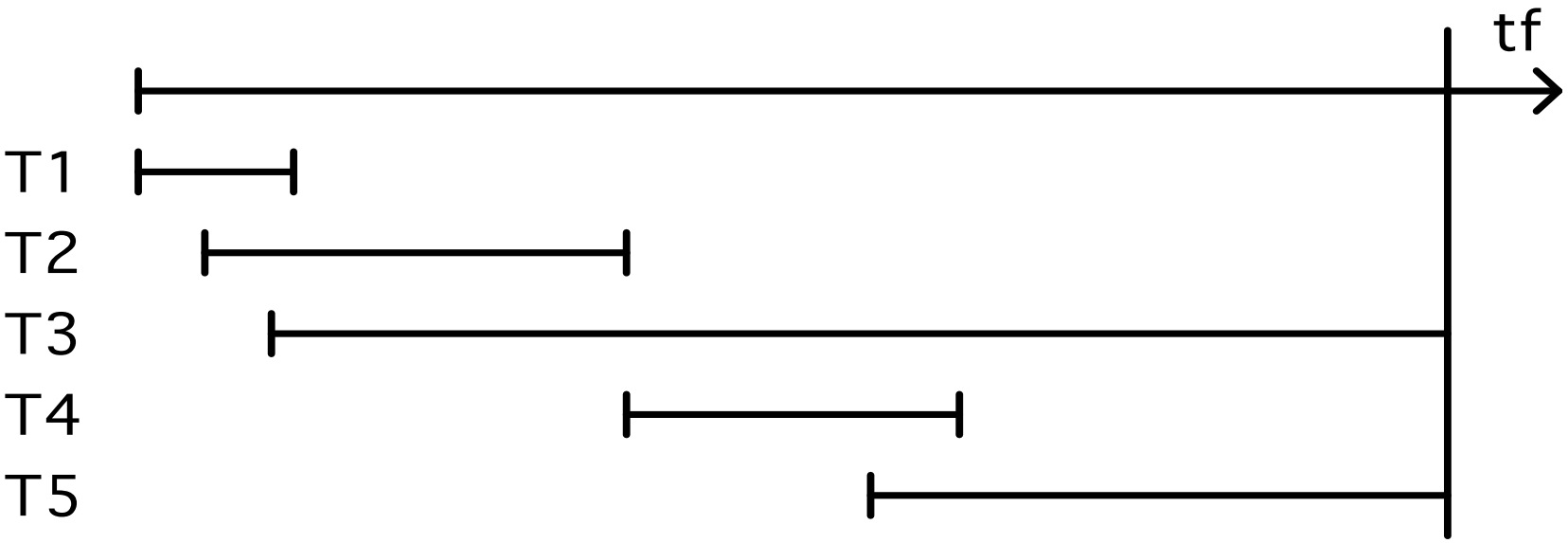


undo

INCERTEZZA DEL RECOVERY NEL CASO DI SYSTEM CRASH

Metodo del "CHECKPOINT" nella politica UNDO/REDO

In caso di system crash il contenuto della memoria principale e degli I/O buffers è perduto.



CHECKPOINT

T3 e T5 sicuramente non sono state completate e devono essere sottoposte alla procedura di UNDO.

Per T1, T2 e T4, che sono terminate, non è sicuro se gli aggiornamenti sono stati definitivamente copiati su memoria ausiliaria.

Bisogna quindi controllare sul LOG e se queste hanno raggiunto il COMMIT (commit record nel Log) bisogna farne il REDO .

Quanto indietro nel LOG bisogna andare, per essere sicuri di eseguire il recovery di tutte le transazioni terminate in modo corretto ?

CHECKPOINT

CHECKPOINT SYSTEM

Periodicamente il sistema esegue il *check* del DB:

metodo 1

fa terminare le transazioni non ancora terminate e ricopia tutto il contenuto del buffer di memoria centrale destinato al DB sul disco ed inserisce un "*checkpoint record*" nel LOG.

Il sistema DBMS, dopo il restart del sistema di calcolo, cerca nel LOG l'ultimo checkpoint record ed esegue la sua procedura di RECOVERY sulle transazioni iniziate dopo. Se il guasto avviene durante l'operazione di checkpoint è valido il checkpoint precedente.

CHECKPOINT

CHECKPOINT SYSTEM

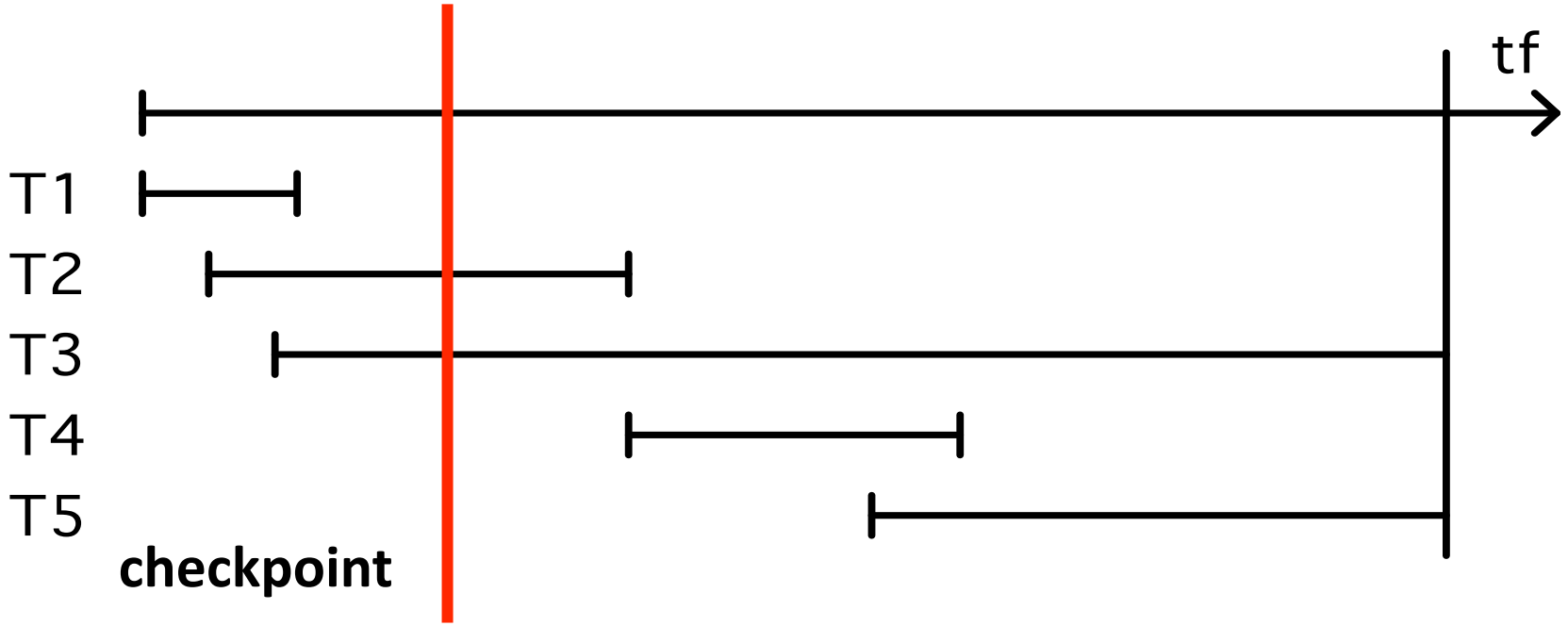
Periodicamente il sistema esegue il *check* del DB:

metodo 2

ricopia tutte le pagine di transazioni terminate sul disco ed inserisce un "*checkpoint record*" nel LOG, registra nel checkpoint record gli identificatori delle transazioni non ancora terminate.

Il sistema DBMS, dopo il restart del sistema di calcolo, cerca nel LOG l'ultimo checkpoint record ed esegue la sua procedura di RECOVERY sulle transazioni iniziate dopo e su quelle registrate nel checkpoint record.

CHECKPOINT



T1 è ok, per T2 e T4 si fa REDO , per T3 e T5 UNDO

Funzionamento del recovery manager

- **Il recovery manager , al restart del sistema, esegue un protocollo del tipo:**
  1. **Legge su un file di *RESTART* (sempre contenuto nel LOG) l'indirizzo dell'ultimo CHECKPOINT; nel record di checkpoint sono contenuti gli identificatori delle transazioni attive al momento del checkpoint.**
  2. **Prepara due file: UNDO LIST con gli identificatori delle transazioni attive, REDO LIST vuoto.**

Funzionamento del recovery manager

3. **Legge il LOG partendo dall'ultimo checkpoint:**
   se trova Begin Transaction registra la transazione sulla UNDO LIST,
   se trova Commit la porta nella REDO LIST.

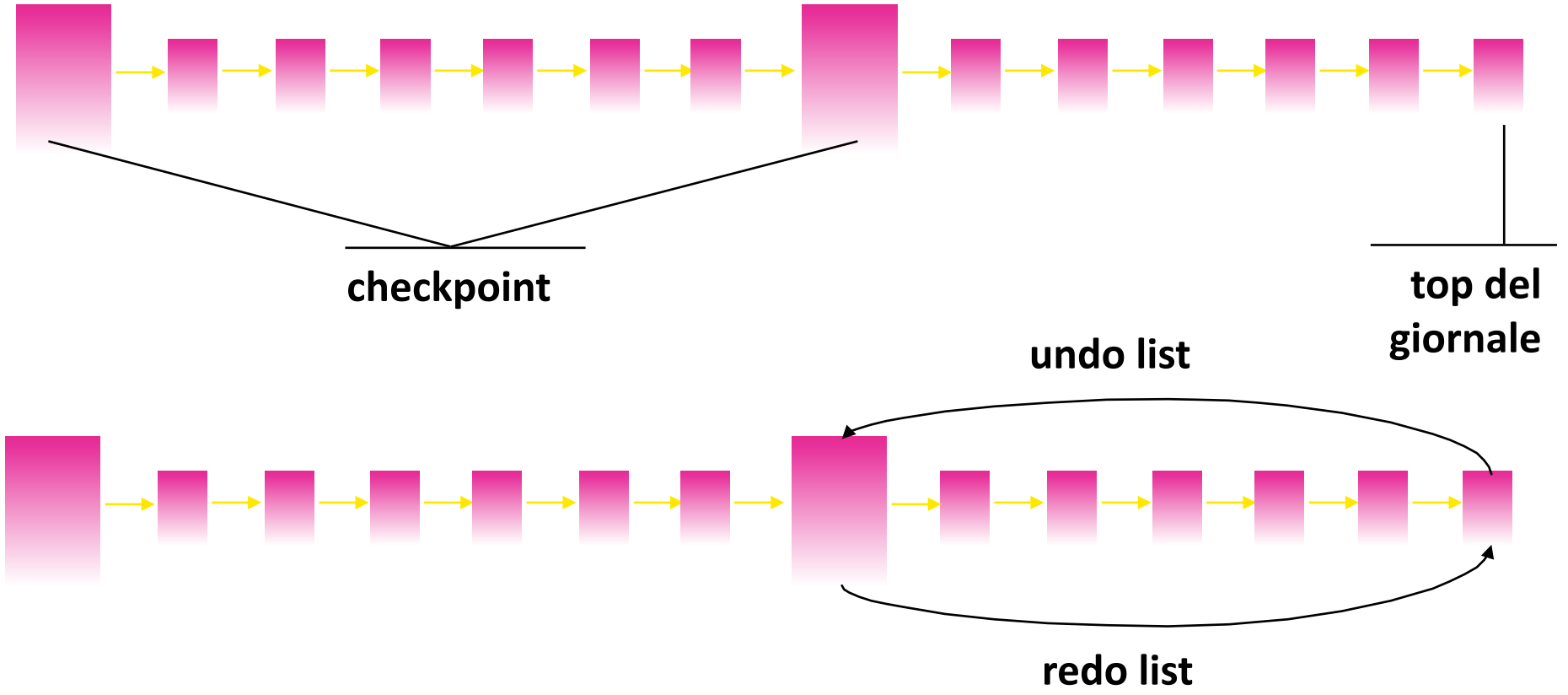
- **Al termine la UNDO LIST contiene la lista delle transazioni da DISFARE, la REDO LIST quella delle transazioni da rifare.**

4. **Il LOG viene rielaborato all'indietro per compiere gli UNDO con i vecchi valori**
5. **Il LOG viene rielaborato in avanti per rifare (REDO) le transazioni da rifare.**

- **Nessun utente è attivo durante il RESTART.**

Checkpoint

Al momento del guasto e dopo:



In caso di guasto di sistema

a **guasto soft** :

perdita in memoria centrale e

RIPRESA (RESTART) A CALDO
procedura di recovery undo/redo
come già visto o altre simili nei casi
no undo/redo ecc.

b **guasto hard**

danneggiamento della memoria disco e

RIPRESA A FREDDO

Ripresa a freddo

- si ripristinano i dati a partire dall'ultimo backup

- si eseguono le operazioni registrate sul log fino all'istante del guasto;
  per sicurezza si tiene il log su un disco diverso da quello dei dati (spesso i log sono 2).

- si esegue una ripresa a caldo

Altre tecniche di RECOVERY

COPIE MULTIPLE:

Si mantiene un numero dispari di copie del DB. In caso di guasto, facendo dei confronti fra le varie copie, in base ad un protocollo di maggioranza si ottiene la copia corretta.
Durante una modifica, una delle copie viene utilizzata per scrivere i nuovi valori. Un flag viene settato per indicare un "update in progress" anche per le altre copie.
Successivamente la modifica viene estesa in parallelo (per quanto possibile) a tutte le altre copie. Esse quindi sono sempre corrette (ed uguali), esclusi gli istanti in cui si esegue la scrittura.
Tecnica molto utilizzata nelle applicazioni avanzate (spaziali, militari, etc....).

FORWARD ERROR RECOVERY

1) **BACKWARD ERROR RECOVERY**:

è quella già vista che stabilisce che il ripristino del DB avvenga ritornando ad uno stato passato corretto. (Se non è possibile, si cerca di riportare il DB in uno stato almeno consistente). Sono le tecniche usate dai DBMS per i sistemi aziendali.

2) **FORWARD ERROR RECOVERY**:

tecnica che prosegue normalmente l'esecuzione (se possibile) tentando una compensazione degli errori. Non permette l'uso di tecniche generali ma solo di algoritmi specifici. Usata in sistemi strategici o in casi in cui non c'è tempo per il recovery backward.

Altre tecniche di RECOVERY

HW CRASH:

SISTEMI RESILIENTI: sistemi completamente duplicati che svolgono esattamente lo stesso lavoro:
il sistema "slave" sostituisce immediatamente il "master" in caso di rottura.

In sistemi strategici i sottosistemi sono più di due: può sussistere il problema (grave) che il master non lavori più in modo corretto e consideri guasti gli slave che a loro volta lo considerano guasto. Gli algoritmi per la gestione di questa situazioni sono molto complessi.

Modello fail-stop