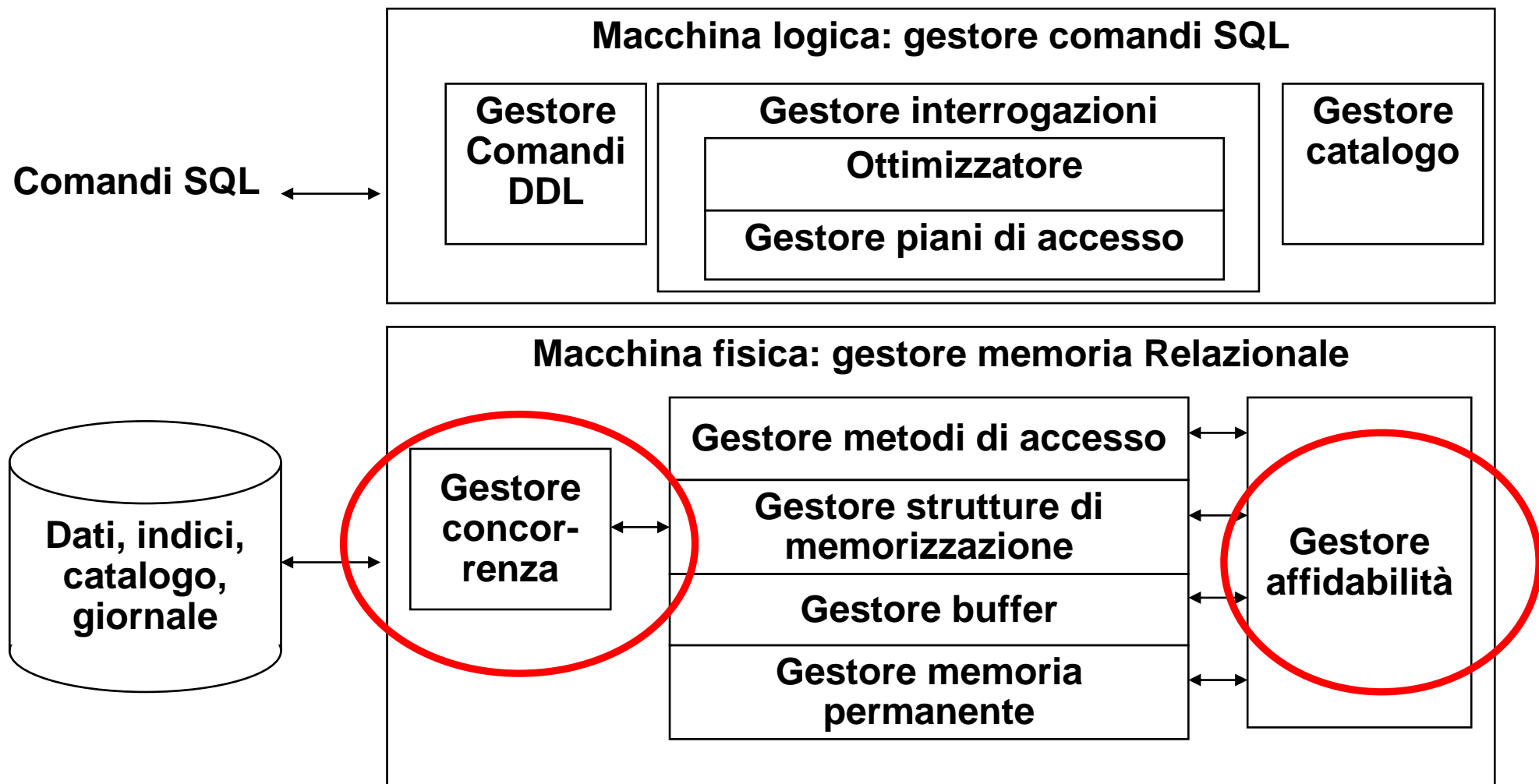

PARTE IV TRANSAZIONI

ARCHITETTURA DEI DBMS




Esempio. Gestione ordini su un sito di

e-commerce
(struttura del DB semplificata)

ITEM(Codice, Descrizione, Prezzo, Quantita)

ORDINE(Id, Data, Ordinate, codOrdinato)

```
SET NumItem=(SELECT COUNT(*) FROM ITEM WHERE
              (Codice=CodiceScelto));
IF (NumItem > 0) THEN
    UPDATE ITEM SET Quantita=Quantita-1 WHERE
    (Codice=CodiceScelto);
     INSERT INTO ORDINE(Data, Ordinate, codOrdinato) VALUES
    (Data, NomeOrdinate, CodiceScelto);
END IF;
```

Il sistema va in crash in questo punto!

Esempio. Gestione ordini su un sito di

ecommerce
(struttura del DB semplificata)

ITEM(Codice, Descrizione, Prezzo, Quantita)

ORDINE(Id, Data, Ordinate, codOrdinato)

```
SET NumItem=(SELECT COUNT(*) FROM ITEM WHERE  
(Codice=CodiceScelto));
```

```
IF (NumItem > 0) THEN
```

```
    UPDATE ITEM SET Quantita=Quantita-1 WHERE  
    (Codice=CodiceScelto);
```

```
    INSERT INTO ORDINE(Data,Ordinate,ItemOrdinato) VALUES  
    (NOW(), NomeOrdinate, CodiceScelto);
```

```
END IF;
```

Due ordini in
contemporanea
eseguono la query

Gestione delle Transazioni

- Le **transazioni** rappresentano l'unità di lavoro elementare (insiemi di istruzioni SQL) che **modificano** il contenuto di una base di dati.
- Sintatticamente un transazione è contornata dai comandi begin transaction (e end transaction); all'interno possono comparire i comandi di commit work e rollback work.

```
begin transaction
update SalariImpiegati
set conto=conto*1.2
where (CodiceImpiegato = 123)
commit work
```

```
begin transaction
update SalariImpiegati
set conto=conto-10
where (CodiceImpiegato = 123)
if conto >0 commit work;
else rollback work
```

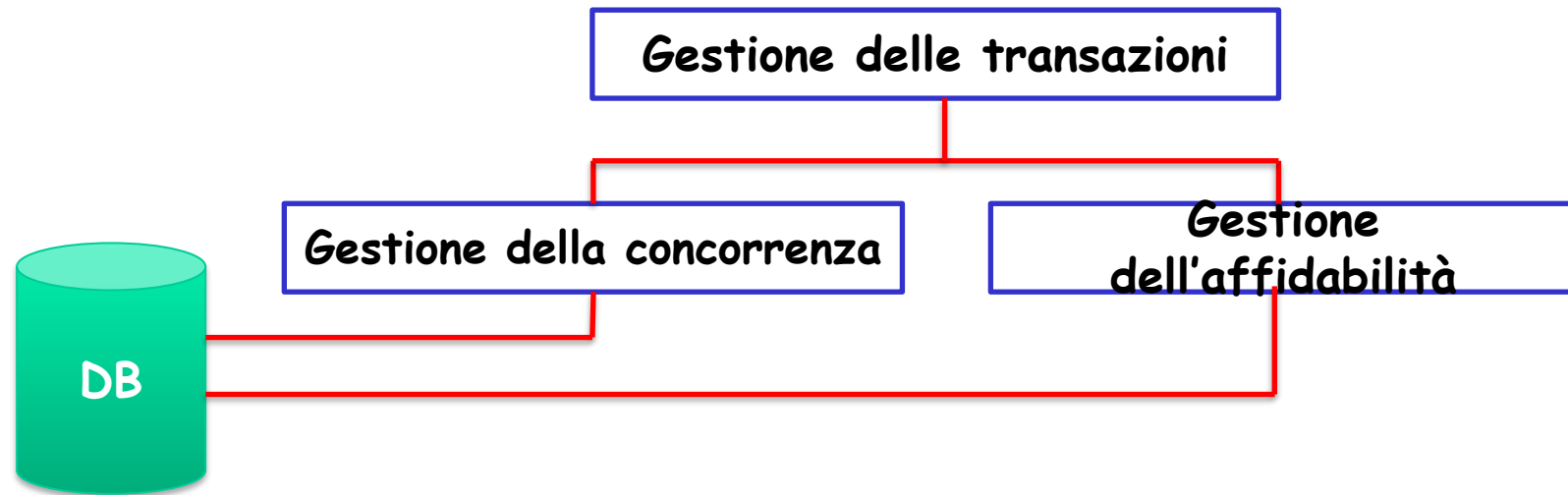
**Le transazioni
sono comprese
tra una BEGIN
transaction
ed una
commit/
rollback**

PROPRIETA' **ACID** DELLE TRANSAZIONI

- **A**tomicità → La transazione deve essere eseguita con la regola del "tutto o niente".
- **C**onsistenza → La transazione deve lasciare il DB in uno stato **consistente**, eventuali vincoli di integrità non devono essere violati.
- **I**solamento → L'esecuzione di una transazione deve essere **indipendente** dalle altre.
- **P**ersistenza (**D**urability) → L'effetto di una transazione che ha fatto commit work non deve essere perso.

Funzioni del DBMS (in breve...)

- **Gestione dei dati**: cura la memorizzazione permanente dei dati ed il loro accesso
- **Gestione del buffer**: cura il trasferimento dei dati da memoria di massa a memoria centrale, e il caching dei dati in memoria centrale
- **Ottimizzazione delle interrogazioni**: seleziona il piano di accesso di costo ottimo con cui valutare ciascuna interrogazione



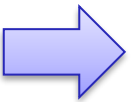
GESTIONE DELL'AFFIDABILITÀ

Cos'è una transazione?

- Una **transazione** è un'unità logica di elaborazione che corrisponde a una serie di operazioni fisiche elementari (letture/scritture) sul DB
- Esempi:
 - Trasferimento di una somma da un conto corrente ad un altro
UPDATE CC
SET Saldo = Saldo - 50
WHERE Conto = '123'
 - Aggiornamento degli stipendi degli impiegati di una sede
UPDATE Imp
SET Stipendio = 1.1*Stipendio
WHERE Sede = 'PISA'

GESTIONE DELLE TRANSAZIONI

- Una **funzionalità** essenziale di un DBMS è la **protezione dei dati** da malfunzionamenti e da interferenze dovute all'accesso contemporaneo ai dati da parte di più utenti.
- La transazione per il programmatore:
 - Una transazione è un programma sequenziale costituito da operazioni che il sistema deve eseguire garantendo:
 - **Atomicità, Consistenza, Serializzabilità , Persistenza**
 - **(Atomicity, Consistency, Isolation, Durability - ACID)**



Le transazioni: atomicità e Persistenza

- *Definizione* Una **transazione** è una sequenza di azioni di lettura e scrittura in memoria permanente e di elaborazioni di dati in memoria temporanea.
- *Gestore dell'affidabilità:*
 - **Atomicità:** Le transazioni che terminano prematuramente (aborted transactions) sono trattate dal sistema come se non fossero mai iniziate; pertanto eventuali loro effetti sulla base di dati sono annullati.
 - **Persistenza:** Le modifiche sulla base di dati di una transazione terminata normalmente sono permanenti, cioè non sono alterabili da eventuali malfunzionamenti.

Le transazioni: Serializzabilità

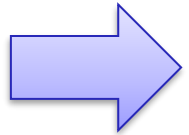
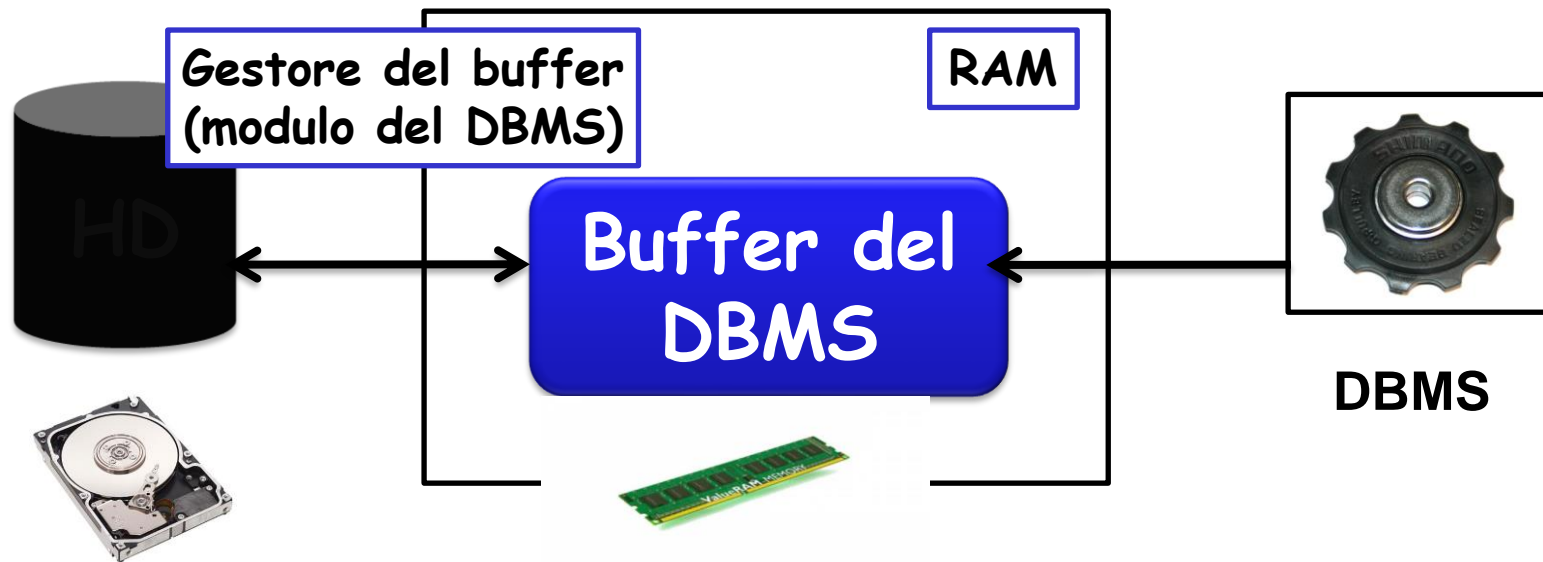
- *Definizione* Una **transazione** è una sequenza di azioni di lettura e scrittura in memoria permanente e di elaborazioni di dati in memoria temporanea:

Gestore della Concorrenza:

- **Serializzabilità:** Nel caso di esecuzioni concorrenti di più transazioni, l'effetto complessivo è quello di una esecuzione seriale.

Gestione delle Transazioni

- Per aumentare l'efficienza prestazionale, tutti i DBMS utilizzano un buffer temporaneo di informazioni in memoria principale, il quale viene periodicamente scritto su memoria secondaria.



LA TRANSAZIONE PER IL DBMS

- Una transazione può eseguire molte operazioni sui dati recuperati da una base di dati, ma al DBMS interessano solo quelle di **lettura** o **scrittura** della base di dati, indicate con $r_i[x]$ e $w_i[x]$.
- Un dato letto o scritto può essere un **record**, un **campo** di un record o una **pagina**. Per semplicità supporremo che sia una pagina.
- Un'operazione di lettura $r_i[x]$ comporta la lettura di una pagina nel buffer, se non già presente.
- Un'operazione di scrittura $w_i[x]$ comporta l'eventuale lettura nel buffer di una pagina e la sua modifica nel buffer, ma non necessariamente la sua scrittura in memoria permanente. Per questa ragione, in caso di malfunzionamento, si potrebbe perdere l'effetto dell'operazione.

TIPI DI MALFUNZIONAMENTO

- Fallimenti di **transazioni**: non comportano la perdita di dati in memoria temporanea né persistente (es.: violazione di vincoli, violazione di protezione, stallo)
- Fallimenti di **sistema**: comportano la perdita di dati in **memoria temporanea** ma non di dati in memoria persistente (es.: comportamento anomalo del sistema, caduta di corrente, guasti hardware sulla memoria centrale)
- **Disastri**: comportano la perdita di dati in **memoria persistente** (es.: danneggiamento di periferica)

Gestore dell'affidabilità → verifica che siano garantite le proprietà di **atomicità e persistenza** delle transazioni.

- Responsabile di implementare i comandi di:
begin transaction, commit, rollback
- Responsabile di ripristinare il sistema dopo **malfunzionamenti software** (**ripresa a caldo**)
- Responsabile di ripristinare il sistema dopo **malfunzionamenti hardware** (**ripresa a freddo**)

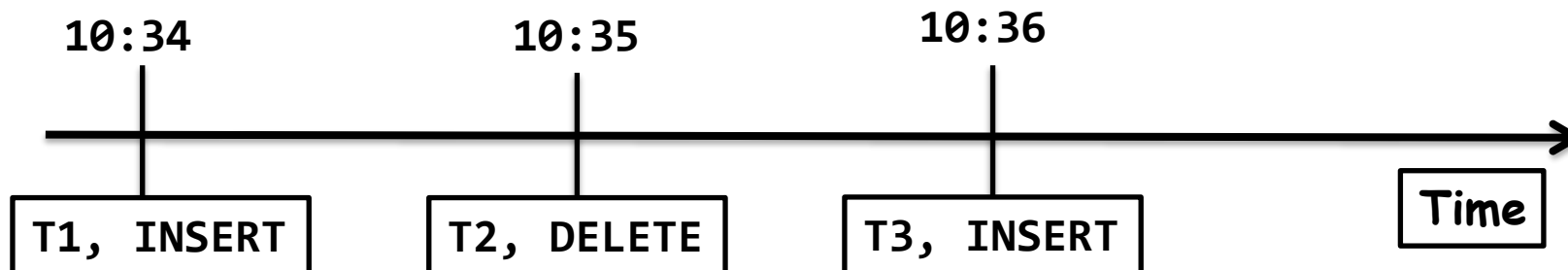
Gestione delle Transazioni - File di LOG

- Il controllore di affidabilità utilizza un log, nel quale sono indicate tutte le operazioni svolte dal DBMS.

LOG, struttura fisica

```
<10:34, 11/12/2019, Transaction: T1, INSERT(3,Mario, Rossi)
INTO IMPIEGATI>
<10:35, 11/12/2019, Transaction: T2, DELETE(3,Mario, Rossi)
INTO IMPIEGATI>
<10:36, 11/12/2019, Transaction: T3, INSERT(6,Maria, Bianchi)
INTO IMPIEGATI>
```

LOG, struttura logica





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 su un oggetto O , dove BS è before state and AS è after state.
- 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)

- Il log si presenta come un **file sequenziale** suddiviso in **record**:



- Due tipi di record:

-  **Record di transazione** → tengono traccia delle operazioni svolte da ciascuna transazione sul DBMS. Per ogni transazione, un record di begin (B), record di insert (I), delete (D) e update (U) e un record di commit (C) o di abort (A).
-  **Record di sistema** → tengono traccia delle operazioni di sistema (dump/checkpoint).

L'operazione di dump (copia del DB)

- 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.

PROTEZIONE DEI DATI DA Malfunzionamenti

- Copia della BD (**DUMP**)
- Giornale/**log**: durante l'uso della BD, il sistema registra nel giornale/log la storia delle azioni effettuate sulla BD dal momento in cui ne è stata fatta l'ultima copia.
- Contenuto del giornale/log:
 - (T, begin);
 - Per ogni operazione di modifica:
 - la transazione responsabile;
 - il tipo di ogni operazione eseguita;
 - la nuova e vecchia versione del dato modificato:
(T,write, address, oldV, newV);
 - (T, commit) o (T, abort).

- **REGOLE di SCRITTURA del LOG**

- **Regola Write Ahead Log (WAL)** → la parte **BS** (*before state*) di ogni record di log deve essere scritta **prima che la corrispondente operazione** venga effettuata nella base di dati.

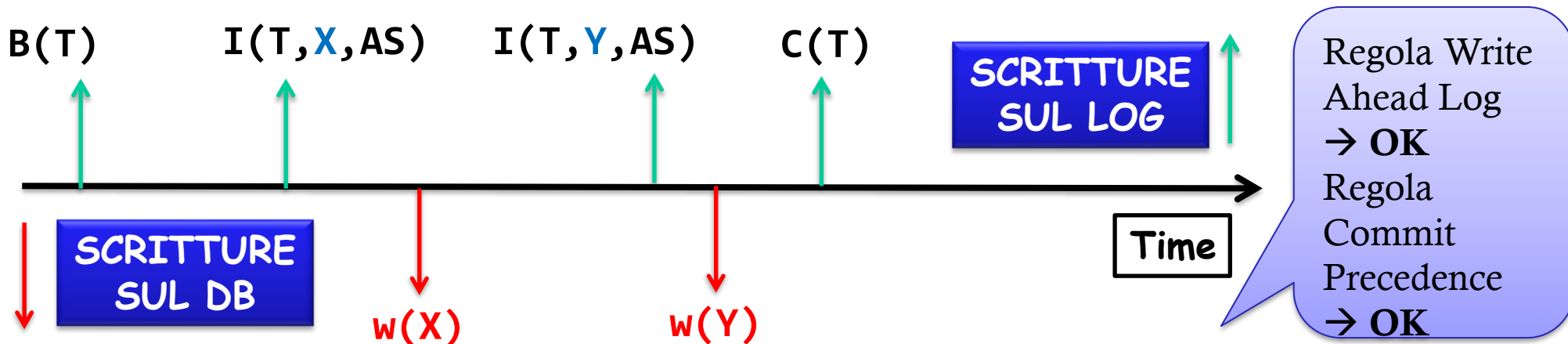
- **REGOLE di SCRITTURA del LOG**

- **Regola Write Ahead Log (WAL)** → la parte **BS** (*before state*) di ogni record di log deve essere scritta **prima che la corrispondente operazione venga effettuata nella base di dati**.
- **Regola di Commit Precedence** → la parte **AS** (*after state*) di ogni record di log deve essere scritta nel log **prima di effettuare il commit** della transazione.

Regola Write Ahead Log

- **REGOLE di SCRITTURA del LOG**

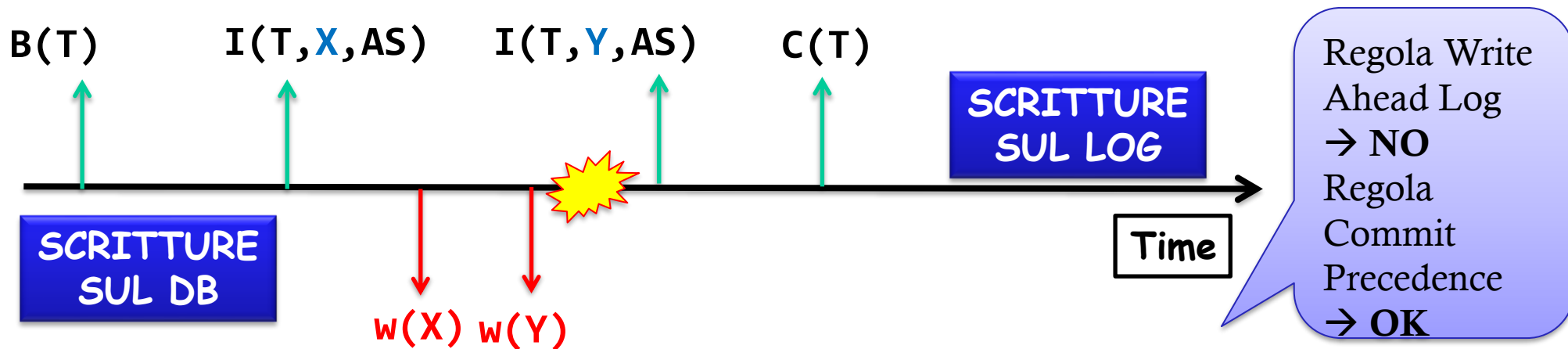
- **Regola Write Ahead Log (WAL)** → la parte **BS** (*before state*) di ogni record di log deve essere scritta **prima che la corrispondente operazione** venga effettuata nella base di dati.
- **Regola di Commit Precedence** → la parte **AS** (*after state*) di ogni record di log deve essere scritta nel log **prima di effettuare il commit** della transazione.



Regola Write Ahead Log

- **REGOLE di SCRITTURA del LOG**

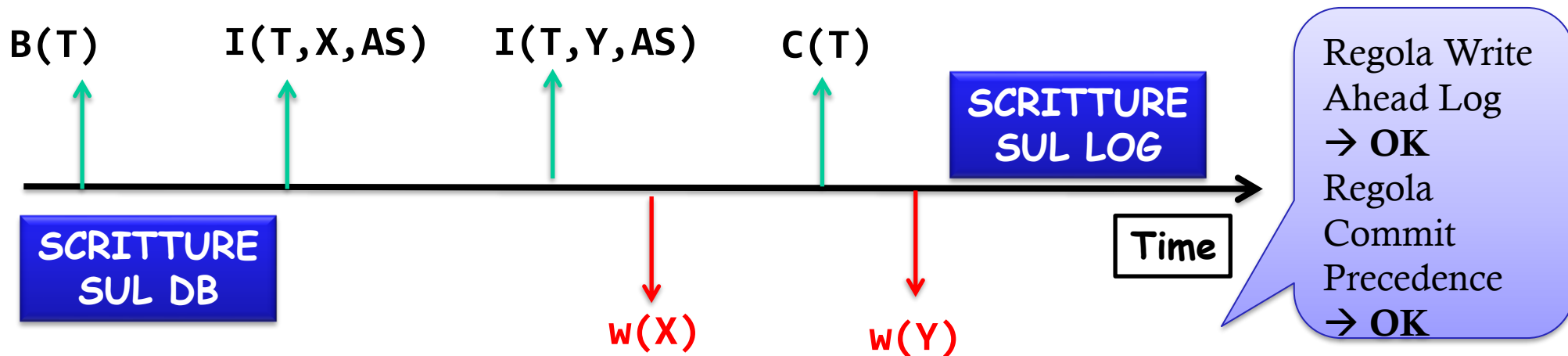
- **Regola Write Ahead Log (WAL)** → la parte **BS** (*before state*) di ogni record di log deve essere scritta **prima che la corrispondente operazione** venga effettuata nella base di dati.
- **Regola di Commit Precedence** → la parte **AS** (*after state*) di ogni record di log deve essere scritta nel log **prima di effettuare il commit** della transazione.



Regola Write Ahead Log

- **REGOLE di SCRITTURA del LOG**

- **Regola Write Ahead Log (WAL)** → la parte **BS** (*before state*) di ogni record di log deve essere scritta **prima che la corrispondente operazione** venga effettuata nella base di dati.
- **Regola di Commit Precedence** → la parte **AS** (*after state*) di ogni record di log deve essere scritta nel log **prima di effettuare il commit** della transazione.



GESTIONE DELL'AFFIDABILITA'

- Gli algoritmi si differenziano a seconda del modo in cui si trattano le scritture sulla BD e la terminazione delle transazioni
 - Disfare-Rifare ←
 - Disfare-NonRifare
 - NonDisfare-Rifare
 - NonDisfare-NonRifare
- Ipotesi: Le scritture nel giornale vengono portate subito nella memoria permanente!

- Quando si portano le modifiche nella BD ?
 - Politica della **modifica libera** : le modifiche *possono* essere portate nella BD stabile prima che la T termini (disfare o *steal*).
- Regola per poter disfare: prescrizione nel giornale ("**Log Ahead Rule**" o "**Write Ahead Log**"):
 - *se la nuova versione di una pagina rimpiazza la vecchia sulla BD stabile prima che la transazione T abbia raggiunto il punto di Commit, allora la vecchia versione della pagina deve essere portata prima sul giornale in modo permanente.*

- Come si gestisce la terminazione ?
 - *Commit libero* : una T può essere considerata terminata normalmente prima che tutte le modifiche vengano riportate nella BD stabile (occorre rifare).
- Regola per poter rifare una T: ("*Commit Rule*")
 - *Le modifiche (nuove versioni delle pagine) di una T devono essere portate stabilmente nel giornale prima che la T raggiunga il Commit (condizione per rifare).*

PUNTO DI ALLINEAMENTO ("CHECKPOINT")

- Al momento del ripristino, solo gli aggiornamenti più recenti tra quelli riportati sul giornale/log potrebbero non essere stati ancora riportati sulla base di dati. Come ottenere la certezza che non è necessario rieseguire le operazioni più vecchie?
- Periodicamente si fa un Checkpoint (CKP): si scrive la marca CKP sul giornale/log per indicare che tutte le operazioni che la precedono sono state effettivamente effettuate sulla BD.

PUNTO DI ALLINEAMENTO ("CHECKPOINT")

- Un modo (troppo semplice) per fare il CKP:
 1. si sospende l'attivazione di nuove transazioni,
 2. si completano le precedenti, si allinea la base di dati (ovvero si riportano su disco tutte le pagine "sporche" dei buffer),
 3. si scrive nel giornale/log la marca CKP.
 4. Si riprende l'esecuzione delle operazioni.

CHECKPOINT

- Si scrive sul giornale una marca di inizio checkpoint che riporta l'elenco delle transazioni attive (**BeginCkp**, {T1,...,Tn})
- In parallelo alle normali operazioni delle transazioni, il gestore del buffer riporta sul disco tutte le pagine modificate
- Si scrive sul giornale una marca di EndCkp
- La marca di **EndCkp** certifica che tutte le scritture avvenute prima del BeginCkp ora sono sul disco.
- Le scritture avvenute tra BeginCkp e EndCkp forse sono sul disco e forse no.

RIPRESA DAI MALFUNZIONAMENTI (disfare-rifare)

- Fallimenti di **transazioni**: si scrive nel giornale (T, abort) e si applica la procedura disfare.
- Fallimenti di **sistema**:
 - La BD viene ripristinata con il comando Restart (ripartenza di emergenza), a partire dallo stato al punto di allineamento, procedendo come segue:
 - **Le T non terminate vanno disfatte**
 - **Le T terminate devono essere rifatte.**
- **Disastri**: si riporta in linea la copia più recente della BD e la si aggiorna rifacendo le modifiche delle T terminate normalmente (ripartenza a freddo).



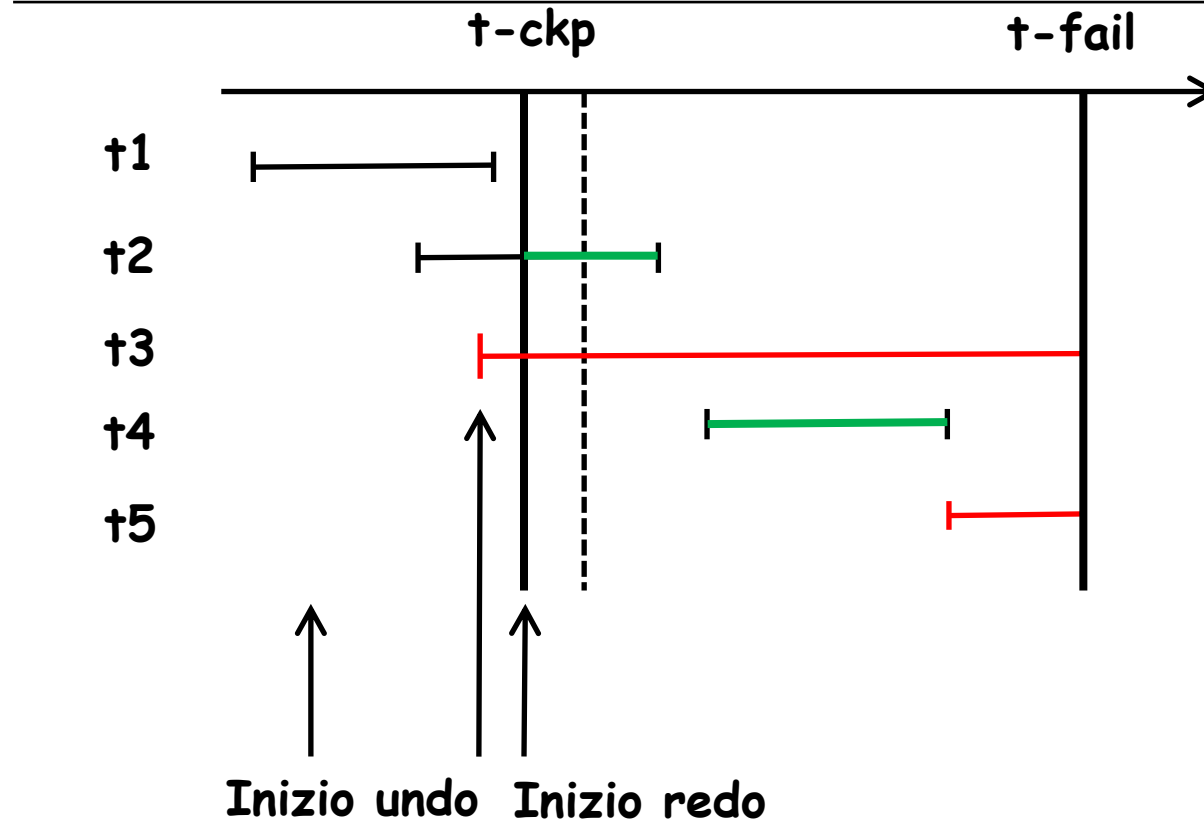
Ripresa a caldo

- Garantisce atomicità e persistenza delle transazioni
- Quattro fasi:
 - trovare l'ultimo checkpoint (ripercorrendo il log a ritroso)
 - costruire gli insiemi UNDO (transazioni da disfare) e REDO (transazioni da rifare)
 - ripercorrere il log all'indietro, **fino alla più vecchia azione** delle transazioni in UNDO, disfacendo tutte le azioni delle transazioni in UNDO
 - ripercorrere il log in avanti, rifacendo tutte le azioni delle transazioni in REDO

Ripresa a freddo

- Risponde ad un guasto che provoca il deterioramento della BD
 - Si ripristinano i dati a partire dal backup
 - Si eseguono le operazioni registrate sul giornale fino all'istante del guasto
 - Si esegue una ripresa a caldo

Ripartenza dopo un fallimento



- T_1 va ignorata
- T_2 e T_4 vanno rifatte
- T_3 e T_5 vanno disfatte

- trovare l'ultimo checkpoint (ripercorrendo il log a ritroso)
- costruire gli insiemi UNDO (transazioni da disfare) e REDO (transazioni da rifare)
- ripercorrere il log all'indietro, fino alla più vecchia azione delle transazioni in UNDO, disfacendo tutte le azioni delle transazioni in UNDO
- ripercorrere il log in avanti, rifacendo tutte le azioni delle transazioni in REDO

Esercizio Applicare il protocollo rifare disfare

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), B(T6)

U(T3, O3, B5, A5), U(T5, O4, B6, A6),

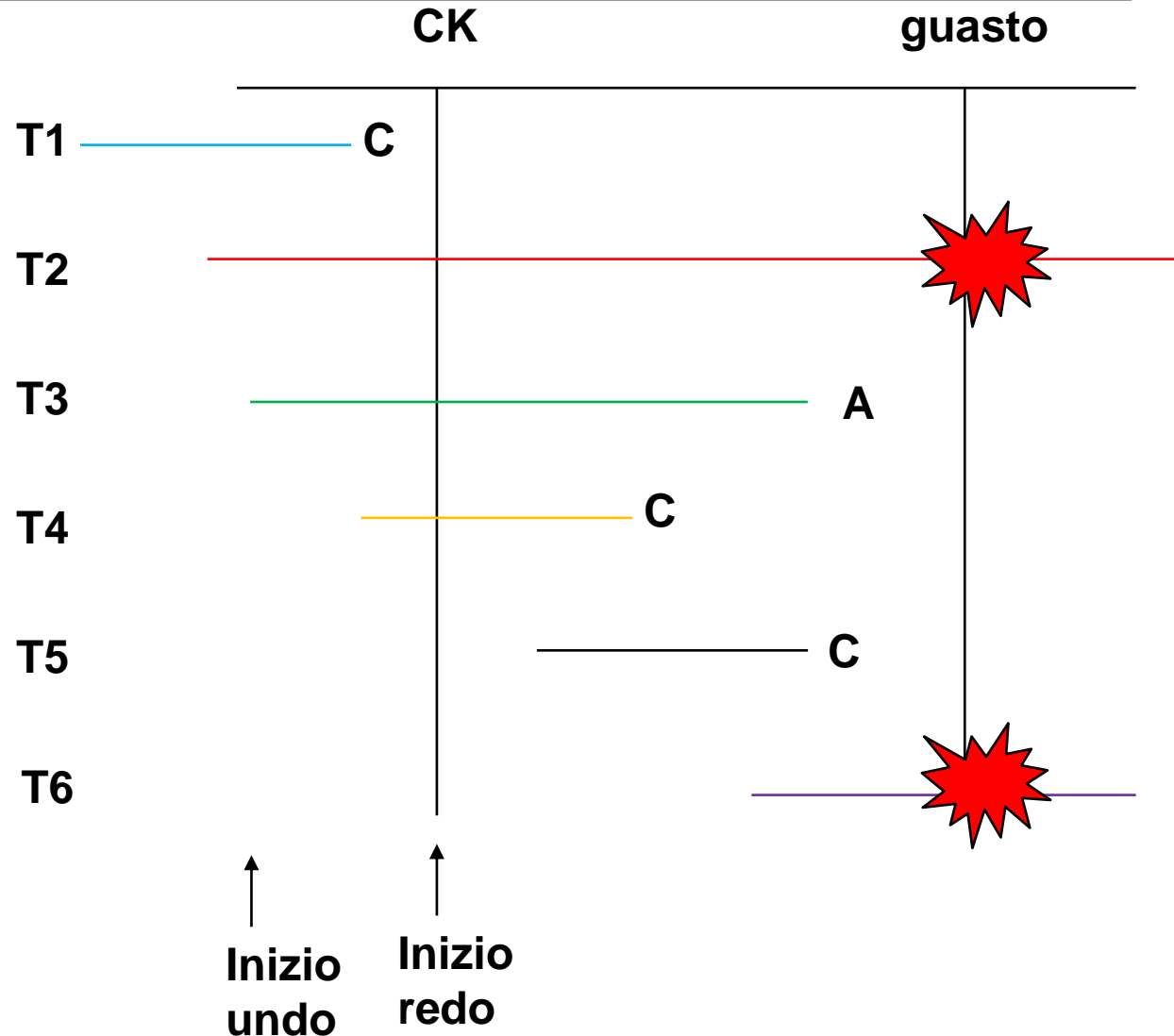
D(T3, O5, B7), U(T6, O6, B7, A7)

A(T3)

C(T5)

I(T2, O6, A8)

guasto



Esempio di ripresa a caldo: fase 1

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), B(T6)

U(T3, O3, B5, A5), U(T5, O4, B6, A6),

D(T3, O5, B7), U(T6, O6, B7, A7)

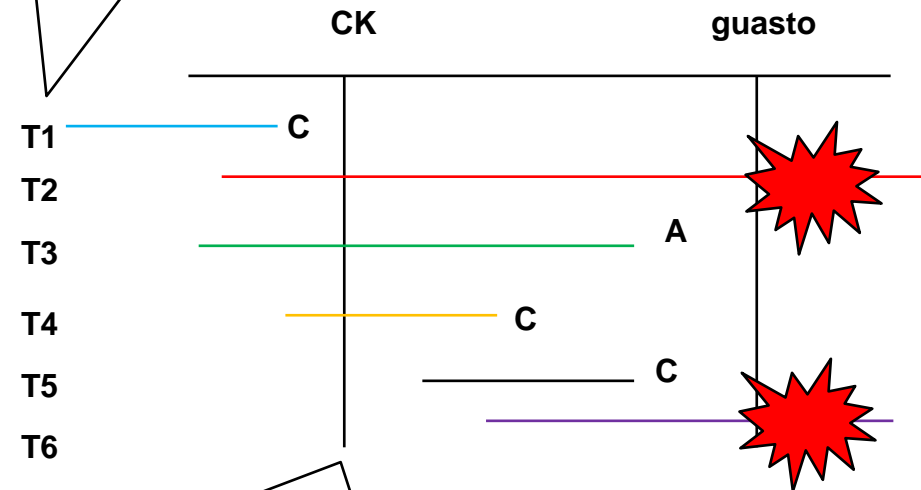
A(T3)

C(T5)

I(T2, O6, A8)

guasto

T1 fa il commit
prima del
checkpoint



Transazioni presenti nel record di Checkpoint:
{T2, T3, T4}

Inizializziamo UNDO= {T2, T3, T4} e REDO={}

Esempio di ripresa a caldo: fase 2

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), B(T6)

U(T3, O3, B5, A5), U(T5, O4, B6, A6),

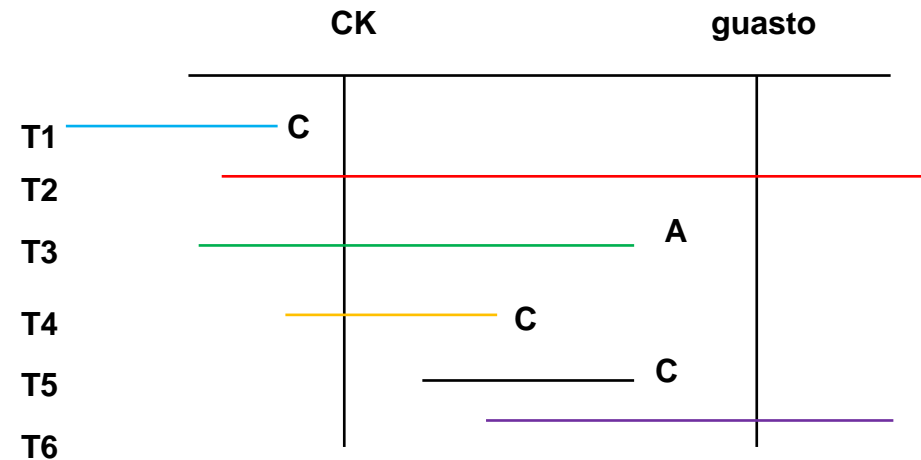
D(T3, O5, B7), U(T6, O6, B7, A7)

A(T3)

C(T5)

I(T2, O6, A8)

guasto



UNDO = {T2, T3, T4} ?

C(T4) ⇒

UNDO = {T2, T3}

REDO = {T4}

B(T5) ⇒

UNDO = {T2, T3, T5}

REDO = {T4}

B(T6) ⇒

UNDO = {T2, T3, T5, T6}

REDO = {T4}

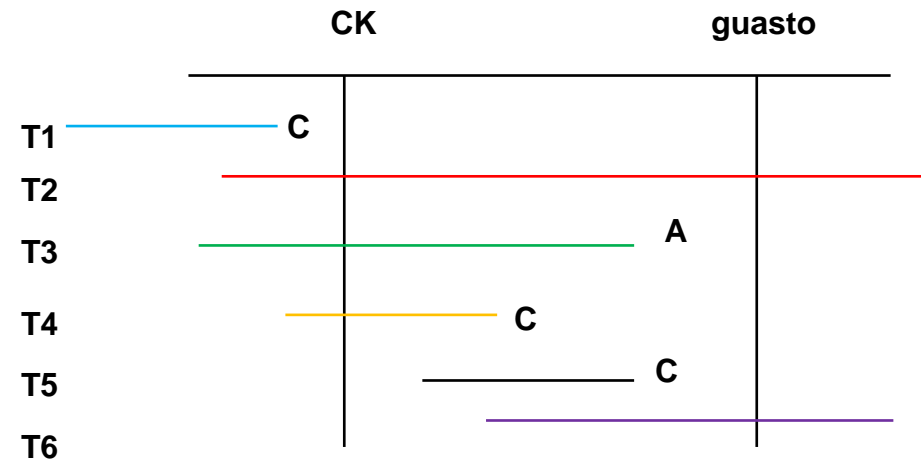
C(T5) ⇒

UNDO = {T2, T3, T6}

REDO = {T4, T5}

Esempio di ripresa a caldo: fase 3 - UNDO

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), B(T6)
 U(T3, O3, B5, A5), U(T5, O4, B6, A6),
 D(T3, O5, B7), U(T6, O6, B7, A7)
 A(T3)
 C(T5)
 I(T2, O6, A8)
guasto



UNDO = {T2, T3, T6}
 REDO = {T4, T5}

D(O6) → T2
 U(O4 = B7) → T6
 I(O5 = B7) → T3
 U(O3 = B5) → T3
 U(O2 = B3) → T3
 U(O1 = B1) → T2

Esempio di ripresa a caldo: fase 4 - REDO

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), B(T6)

U(T3, O3, B5, A5), U(T5, O4, B6, A6),

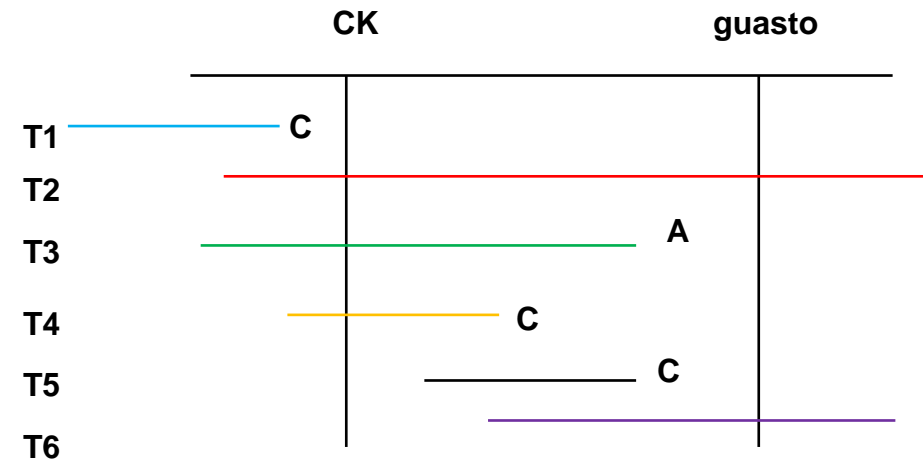
D(T3, O5, B7), U(T6, O6, B7, A7)

A(T3)

C(T5)

I(T2, O6, A8)

guasto



UNDO = {T2, T3, T6}

REDO = {T4, T5}

U(O3 = A4) → T4

U(O4 = A6) → T5

GESTIONE DELL'AFFIDABILITA'

- Gli algoritmi si differenziano a seconda del modo in cui si trattano le scritture sulla BD e la terminazione delle transazioni
 - Disfare-Rifare ←
 - Disfare-NonRifare
 - NonDisfare-Rifare
 - NonDisfare-NonRifare
- Ipotesi: Le scritture nel giornale vengono portate subito nella memoria permanente!

GESTIONE DELLA CONCORRENZA

Serializzazione

- Uno schedule **S** si dice **seriale** se **le azioni di ciascuna transazione appaiono in sequenza**, senza essere inframmezzate da azioni di altre transazioni.

$$S = \{T_1, T_2, \dots T_n\}$$

- Schedule seriale ottenibile se:

- (i) Le transazioni sono **eseguite una alla volta** (scenario non realistico)
- (ii) Le transazioni sono **completamente indipendenti** l'una dall'altra (improbabile)

ACID: Proprietà di isolamento delle transazioni

- Il DBMS transazionale gestisce questi problemi garantendo la proprietà di **isolamento**
- La proprietà di isolamento di una transazione garantisce che essa sia eseguita come se non ci fosse concorrenza
- Questa proprietà è assicurata facendo in modo che ciascun insieme di transazioni concorrenti sottoposte al sistema sia "**serializzabile**"

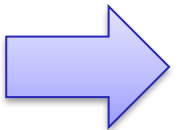
Gestione delle Transazioni - Problematica

- In un sistema reale, le transazioni vengono eseguite in concorrenza per ragioni di efficienza / scalabilità.
- ... Tuttavia, l'esecuzione concorrente determina un insieme di **problematiche** che devono essere gestite ...

T1= Read(x); x=x+1; Write(x); Commit Work

T2= Read(x); x=x+1; Write(x); Commit Work

- Se $x=3$, al termine delle due transazioni x vale 5 (*esecuzione sequenziale*)
... cosa accade in caso di **esecuzione concorrente**?



T1= Read(x); x=x+1; Write(x); Commit Work

T2= Read(x); x=x+1; Write(x); Commit Work

X=3	Transazione1 (T1)	Transazione2 (T2)	T2 scrive 4
	Read(x)		
	x=x+1		
		Read(x)	
T1 scrive 4		x=x+1	
		Write(x)	
		Commit work	
	Write(x) Commit work		

T1= Read(x); x=x+1; Write(x); Rollback Work
T2= Read(x); Commit Work

X=3

Transazione1 (T1)	Transazione2 (T2)
Read(x)	
x=x+1	
Write(x)	
	Read(x)
	Commit work
Rollback work	

T2
legge 4!

T1= Read(x); Read(x); Commit Work
T2= Read(x); x=x+1; Write(x); Commit Work

X=3

T1
legge 3!

T1
legge 4!

Transazione1 (T1)	Transazione2 (T2)
Read(x)	
	Read(x)
	x=x+1
	Write(x)
	Commit work
Read(x) Commit work	

GESTIONE DELLA CONCORRENZA - ANOMALIE

- L'esecuzione concorrente di transazioni è essenziale per un buon funzionamento del DBMS.
- Il DBMS deve però garantire che l'esecuzione concorrente di transazioni avvenga senza interferenze in caso di accessi agli stessi dati.

Gestione CC: X=1000 euro

T1	tempo	T2
	↓	
begin	t1	-
r[x]	↓	begin
-	t2	r[x]
-	↓	$x := x - 800$
$x := x + 500$	t3	-
-	t4	w[x]
w[x]	t5	*Commit*
Commit	t6	
	↓	

Se eseguite in serie avrei:
 $1000 + 500 - 800 = 700$ euro

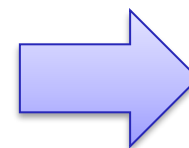
Se in concorrenza
potrei avere
un'anomalia

SERIALITÀ E SERIALIZZABILITÀ

- **Definizione** Un'esecuzione di un insieme di transazioni $\{T_1, \dots, T_n\}$ si dice **seriale** se, per ogni coppia di transazioni T_i e T_j , tutte le operazioni di T_i vengono eseguite prima di qualsiasi operazione T_j o viceversa.
- **Definizione** Un'esecuzione di un insieme di transazioni si dice **serializzabile** se produce lo stesso effetto sulla base di dati di quello ottenibile eseguendo serialmente, in un qualche ordine, le sole transazioni terminate normalmente.

Controllo della concorrenza

- Nella pratica i DBMS implementano tecniche di controllo di concorrenza che garantiscono direttamente la serializzabilità delle transazioni concorrenti.
- Tali tecniche si dividono in due classi principali:
 - Protocolli **pessimistici**/conservativi: tendono a «**ritardare**» l'esecuzione di transazioni che potrebbero generare conflitti, e quindi anomalie, rispetto alla transazione corrente. Cercano quindi di prevenire.
 - Protocolli **ottimistici**, che permettono l'esecuzione sovrapposta e non sincronizzata di transazioni ed effettuano un controllo sui possibili conflitti generati **solo a valle del commit**.



Controllo della concorrenza ottimistico

- Ogni transazione effettua «liberamente» le proprie operazioni sugli oggetti della base di dati secondo l'ordine temporale con cui le operazioni stesse sono generate.
- Al commit, viene effettuato un controllo per stabilire se sono stati riscontrati eventuali conflitti, e nel caso, viene effettuato il rollback delle azioni delle transazioni e la relativa riesecuzione.
- In generale, un protocollo di controllo di concorrenza **ottimistico** è basato su 3 fasi:
 - Fase di **lettura**: ogni transazione legge i valori degli oggetti della BD su cui deve operare e li memorizza in variabili (copie) locali dove sono effettuati eventuali aggiornamenti
 - Fase di **validazione**: vengono effettuati dei controlli sulla serializzabilità nel caso che gli aggiornamenti locali delle transazioni dovessero essere propagati sulla base di dati
 - Fase di **scrittura**: gli aggiornamenti delle transazioni che hanno superato la fase di validazione sono propagati definitivamente sugli oggetti della BD.

- Nella pratica i DBMS implementano tecniche di controllo di concorrenza che garantiscono direttamente la serializzabilità delle transazioni concorrenti.
- Tali tecniche si dividono in due classi principali:
 - Metodi basati su lock
 - Metodi basati su timestamp

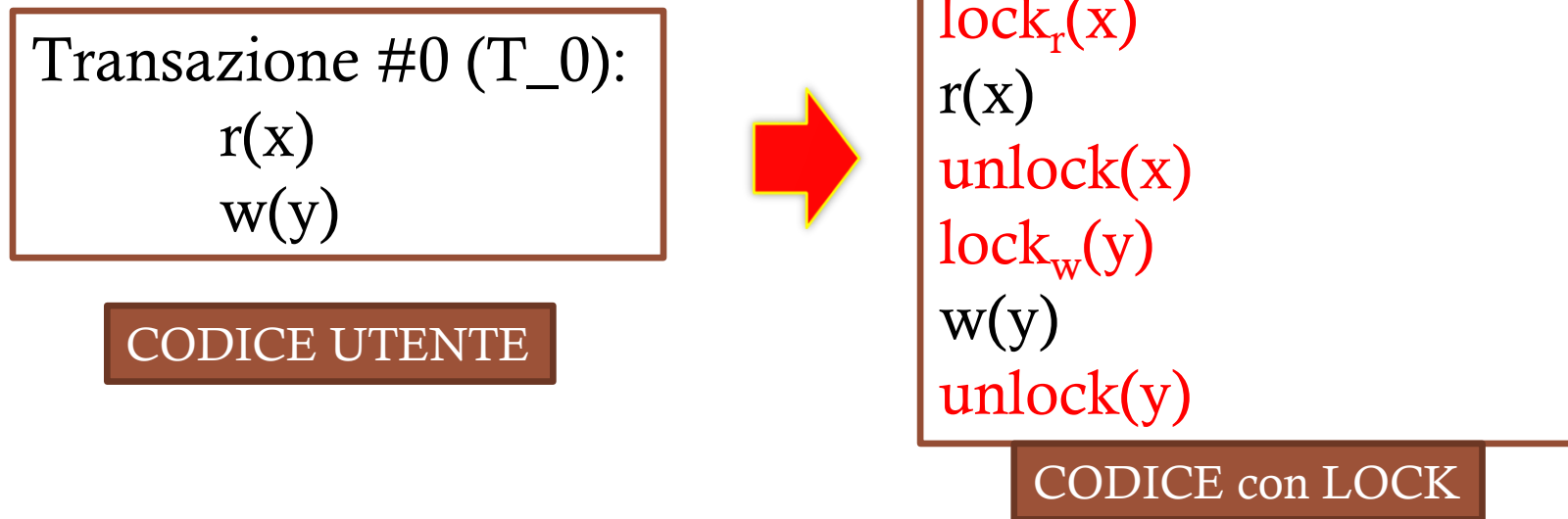
- Come implementare il **controllo della concorrenza**?
- I DBMS commerciali usano il meccanismo dei **lock** → per poter effettuare una qualsiasi operazioni di lettura/scrittura su una risorsa (tabella o valore di una cella), **è necessario aver precedentemente acquisito il controllo (lock) sulla risorsa stessa.**
- **Lock in lettura** (*accesso condiviso*)
- **Lock in scrittura** (*mutua esclusione*)

Gestione di transazioni mediante lock

- I DBMS per evitare anomalie nelle transazione concorrenti usano diverse tecniche
- Una delle più comuni è basata su lock
- Il **lock** è un meccanismo che blocca l'accesso ai dati ai quali una transazione accede ad altre transazioni
 - lock a **livello di riga, tabella, pagina** (multi **granularità**)
 - lock in operazioni di scrittura (*mutua esclusione*) / lettura (*accesso condiviso*) (**multimodale**)
- In generale, quando una risorsa è bloccata, le transazioni che ne richiedono l'accesso vengono in genere messe in coda
 - quindi devono aspettare (che il lock sia rimosso)
- In sostanza, questo è un meccanismo efficace, ma influisce sulle prestazioni

Gestione delle Transazioni

- Su ogni lock possono essere definite **due operazioni**:
 - **Richiesta** del lock in lettura/scrittura.
 - **Rilascio** del lock (**unlock**) acquisito in precedenza.



SERIALIZZATORE 2PL STRETTO

- Il gestore della concorrenza (serializzatore) dei DBMS ha il compito di stabilire l'ordine secondo il quale vanno eseguite le singole operazioni per rendere serializzabile l'esecuzione di un insieme di transazioni.
- **Definizione** Il protocollo del blocco a due fasi stretto (**Strict Two Phase Locking**) è definito dalle seguenti regole:
 1. Ogni transazione, prima di effettuare un'operazione acquisisce il blocco corrispondente (chiede il lock)
 2. Transazioni diverse non ottengono blocchi in conflitto.
 3. I blocchi/lock si rilasciano alla terminazione della transazione (cioè al commit)

Gestione delle Transazioni

- **Strict Two Phase Lock (2PL)** → I lock di una transazione sono rilasciati solo dopo aver effettuato le operazioni di commit/abort.
- **PROBLEMA:** I protocolli 2PL possono generare schedule con situazioni di deadlock (stallo).

TRANSAZIONI

$T_1 = r(x), w(y), \text{Commit}$
 $T_2 = r(y), w(x), \text{Commit}$

SCHEDULE



T_1	T_2
$r_lock(x)$	
	$r_lock(y)$
$r(x)$	
	$r(y)$
$w_lock(y)$	
	$w_lock(x)$

- Per gestire le situazioni di **deadlock** causate dal gestore della concorrenza, si possono usare **tre tecniche**:
 1. **Uso dei timeout** → ogni operazione di una transazione ha un timeout entro il quale deve essere completata, pena annullamento (abort) della transazione stessa.
- T_1 : `r_lock(x, 4000)`, `r(x)`, `w_lock(y, 2000)`, `w(y)`, `commit`, `unlock(x)`, `unlock(y)`

- Per gestire le situazioni di **deadlock** causate dal gestore della concorrenza, si possono usare **tre tecniche**:

2. **Deadlock avoidance** → prevenire le configurazioni che potrebbero portare ad un deadlock ... **COME?**

✧ **Lock/Unlock** di tutte le risorse allo stesso tempo.

✧ Utilizzo di **time-stamp** o di **classi di priorità** tra transazioni
(problema: può determinare **starvation!**)

✧ **starvation**: quando una transazione è impossibilitata a proseguire la sua esecuzione per un periodo di tempo indefinito, mentre le altre transazioni del sistema proseguono tranquillamente.

- Per gestire le situazioni di **deadlock** causate dal gestore della concorrenza, si possono usare **tre tecniche**:

3. **Deadlock detection** → utilizzare **algoritmi per identificare eventuali situazioni di deadlock**, e prevedere meccanismi di recovery dal deadlock
 - **Grafo delle richieste/risorse** utilizzato per identificare la presenza di cicli (corrispondenti a deadlock)
 - In caso di ciclo, si fa abort delle transazioni coinvolte nel ciclo in modo da eliminare la mutua dipendenza ...

CONDIZIONI DI STALLO

- Il problema si può risolvere con tecniche che prevengono queste situazioni (deadlock prevention), oppure con tecniche che rivelano una situazione di stallo e la sbloccano facendo abortire una o più transazioni in attesa (deadlock detection and recovery).

- Un metodo alternativo al 2PL per la gestione della concorrenza in un DBMS prevede l'utilizzo dei **time-stamp delle transazioni** (metodo **TS**).
 - Ad ogni transazione si associa un **timestamp** che rappresenta il momento di inizio della transazione.
 - Ogni transazione **non può leggere o scrivere un dato scritto da una transazione con timestamp maggiore.**
 - Ogni transazione **non può scrivere su un dato già letto da una transazione con timestamp maggiore.**

Livelli di isolamento/consistenza per ogni transazione

- SERIALIZABLE assicura che
 - la transazione T legge solo cambiamenti fatti da transazioni concluse (che hanno fatto il commit)
 - nessun valore letto o scritto da T verrà cambiato da altre transazione finché T non è conclusa
 - se T legge un insieme di valori acceduti secondo qualche condizione di ricerca, l'insieme non viene modificato da altre transazione finché T non è conclusa
- REPEATABLE READ assicura che
 - la transazione T legge solo cambiamenti fatti da transazioni concluse (che hanno fatto il commit)
 - nessun valore letto o scritto da T verrà cambiato da altre transazione finché T non è conclusa
- READ COMMITTED assicura che
 - la transazione T legge solo cambiamenti fatti da transazioni concluse (che hanno fatto il commit)
 - T non vede nessun cambiamento eventualmente effettuato da transazioni concorrenti non concluse tra i valori letti all'inizio di T
- READ UNCOMMITTED
 - a questo livello di isolamento una transazione T può leggere modifiche fatte ad un oggetto da un transazione in esecuzione; ovviamente l'oggetto può essere cambiato mentre T è in esecuzione. Quindi T è soggetta a effetti fantasma