Gestione dei dati

Parte 3
Gestione del buffer e gestione del recovery

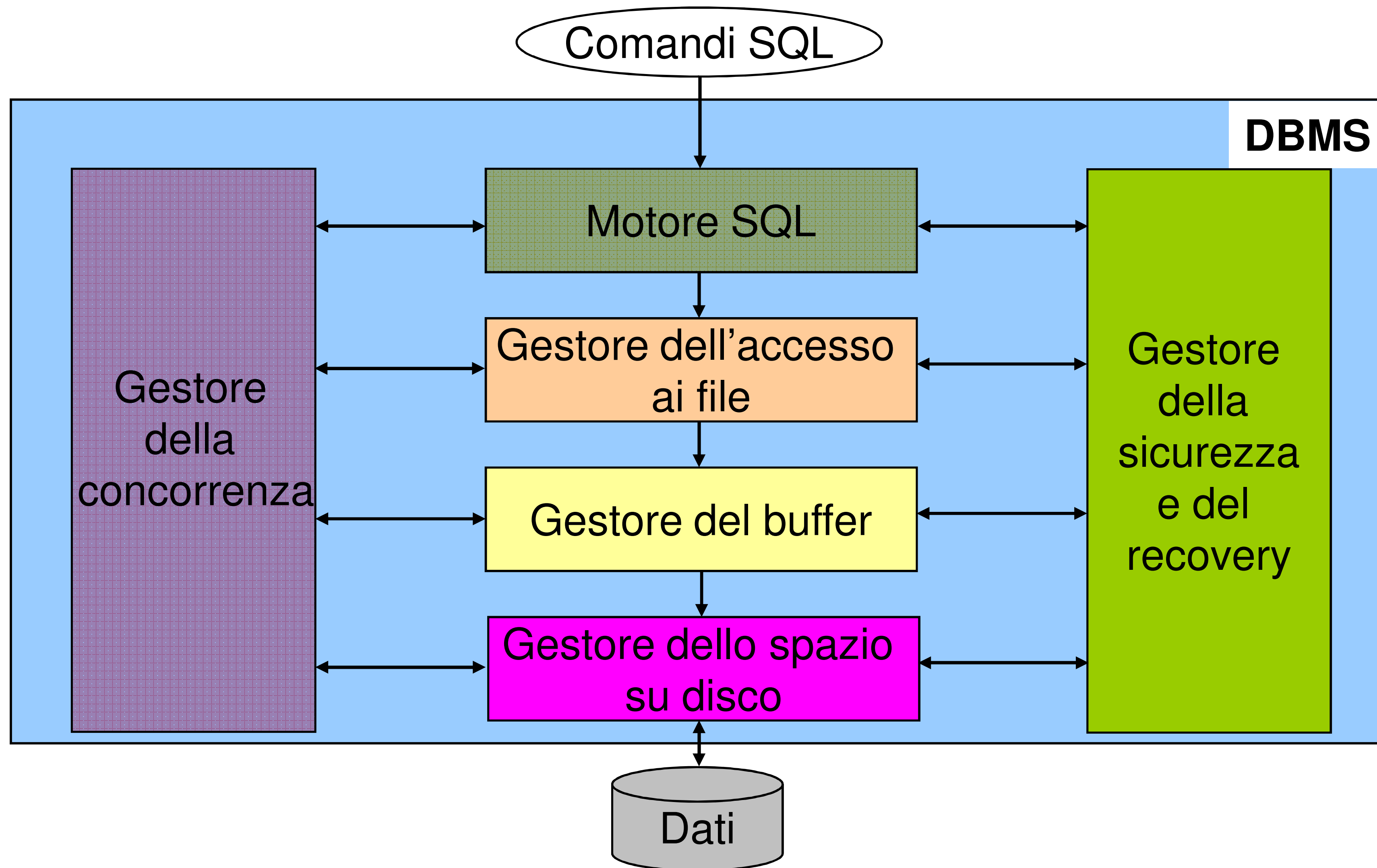
Maurizio Lenzerini, Riccardo Rosati

Facoltà di Ingegneria
Sapienza Università di Roma
Anno Accademico 2011/2012

http://www.dis.uniroma1.it/~rosati/gd/

SAPIENZA
UNIVERSITÀ DI ROMA

Architettura di un DBMS



Sommario

1. Gestione del buffer
2. Gestione del recovery

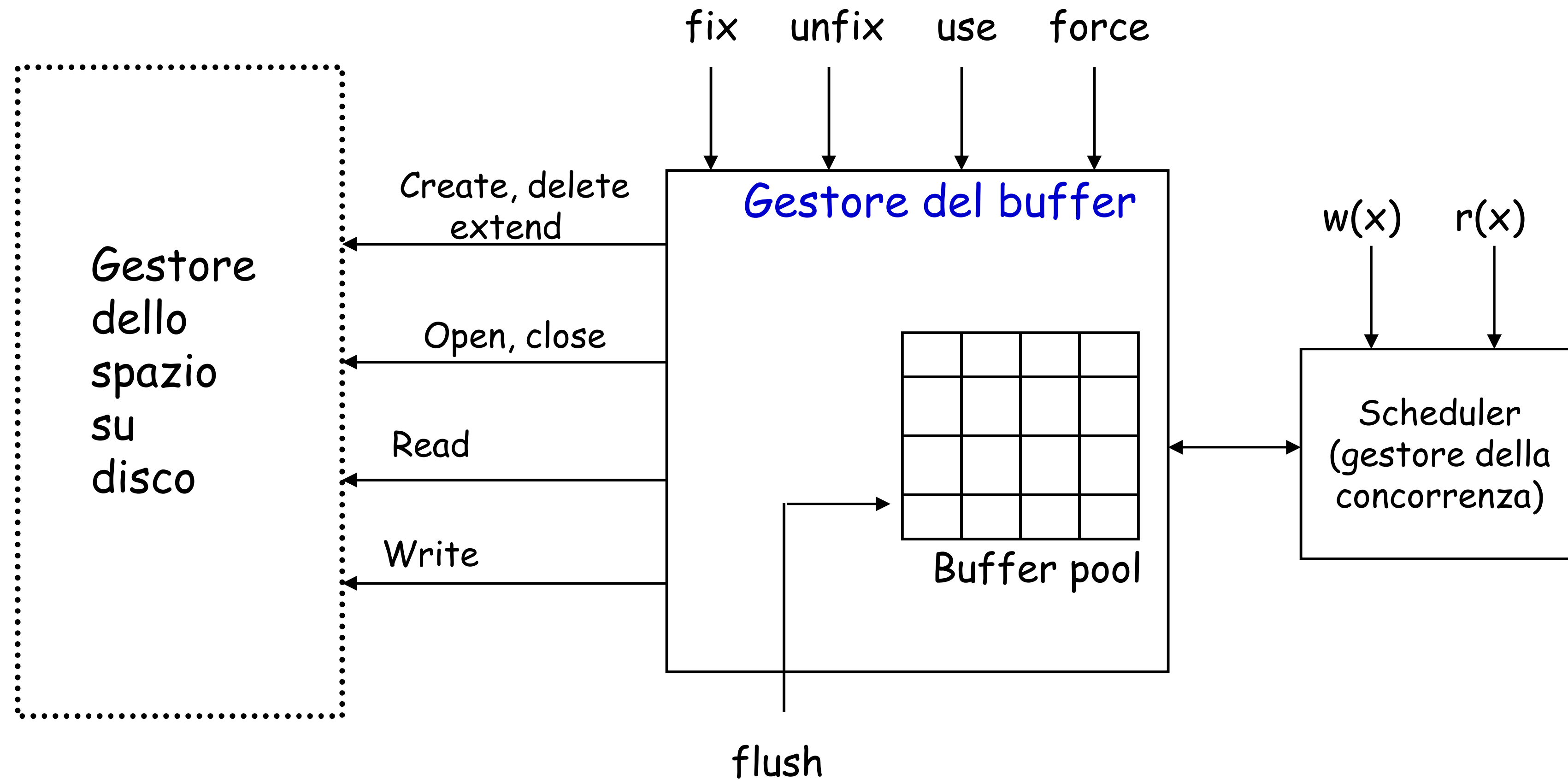
2.1
Gestione del buffer

Il Buffer

Il buffer (detto anche buffer pool) è una zona di memoria non persistente preallocata in modo che sia a disposizione del DBMS per i trasferimenti da memoria di massa a memoria centrale e viceversa. Questa area di memoria è

- condivisa da tutte le transazioni
- utilizzata anche dal sistema (e.g. dal recovery manager)

Architettura del gestore del buffer



Il buffer pool

- Il buffer pool è organizzato in “frame”, che sono pagine in memoria centrale, ciascuna della dimensione di un blocco, blocco che è l’unità di trasferimento da e a memoria di massa. La dimensione tipica può andare dai 2Kb ai 64Kb.
- Essendo il buffer pool in memoria centrale, la gestione delle sue pagine è molto più efficiente (operazioni atomiche dell’ordine del miliardesimo di secondo) rispetto alle pagine della memoria di massa (operazioni atomiche dell’ordine del millesimi di secondo)
- Il buffer pool è gestito con gli stessi criteri di una memoria cache
  – Operazioni primitive di gestione dei blocchi in cache
  – Principio di località dei dati ed esigenza di strategie di replacement dei blocchi
  – Euristica: 80% delle applicazioni accede al 20% delle pagine

Il gestore del buffer

- Accetta primitive invocate dallo scheduler
- Lancia comandi al gestore dello spazio su disco
- Gestisce le pagine del buffer pool

Il gestore del buffer

Agisce mediante le seguenti primitive (che poi analizziamo in maggiore dettaglio):

- Fix: carica pagina nel buffer
- Unfix: rilascia pagina
- Use: modifica pagina nel buffer
- Force: trasferimento sincrono nella memoria di massa
- Flush: trasferimento asincrono nella memoria di massa

La primitiva Fix

- Chiede l'accesso ad una pagina e provoca il suo eventuale caricamento in un frame del buffer pool
- Per ogni frame, il gestore del buffer mantiene:
  - l'informazione su quale pagina contiene
  - pin-count: quante transazioni stanno usando la pagina contenuta nel frame (inizialmente è uguale a 0)
  - dirty: bit che dice se la pagina contenuta nel frame è stata modificata (true) o no (false); inizialmente è uguale a false

La primitiva Fix

Supponiamo che la pagina P venga richiesta mediante la primitiva Fix

1. il gestore del buffer vede se esiste un frame con la pagina P
2. se esiste, incrementa il pin-counter del frame contenente P
3. se non esiste,
   1. si sceglie un frame per portarci la pagina P secondo una certa politica di rimpiazzamento, aggiornando il pin-count del frame
   2. se il bit dirty per il frame scelto è true (e quindi si sta utilizzando una strategia "steal" -- vedi dopo), allora si scrive in memoria secondaria la pagina che era contenuta nel frame scelto
   3. si legge la pagina P dalla memoria secondaria e la si scrive nel frame scelto
4. si restituisce l'indirizzo del frame contenente la pagina P

Politiche di rimpiazzamento

Il frame viene sempre scelto tra quelli con pin-count = 0:

- Se non esiste alcun frame con pin-count=0, la richiesta viene messa in attesa oppure la transazione viene abortita.
- Se invece esiste, sono possibili diverse politiche, tra le quali:
  - LRU
  - Clock replacement

Politiche di rimpiazzamento

- LRU (least recently used): si sceglie il frame usato meno recentemente. Si realizza mediante una coda di frame con pin-count=0
- Clock replacement:
  - si usa una variabile current che punta ai frame in ordine circolare
  - ogni frame ha un bit referenced che denota se è stata referenziata recentemente la pagina in esso contenuta; referenced viene messo a true appena pin-count = 0, per indicare che recentemente è stata utilizzata la pagina
  - quando si cerca un frame per il rimpiazzamento, si analizza il frame puntato da current
    - se esso ha pin-count > 0, allora si va avanti con current e si ripete il procedimento
    - se esso ha pin-count = 0 e referenced=true, si pone referenced a false, per segnalare che ormai è passato del tempo da quando è stata referenziata per l'ultima volta, si incrementa current e si ripete il procedimento
    - se esso ha pin-count = 0 e referenced=false, il frame viene scelto per il rimpiazzamento

Strategie steal/no-steal e force/no-force

- Steal/no steal:
  - Steal: si ammette che si possa scegliere la vittima tra le pagine con dirty=true
  - No-steal: non si ammette tale possibilità
- Force/no force:
  - Force: tutte le pagine attive di una transazione attiva vengono scritte in memoria di massa appena essa fa commit
  - No-force: ci si affida al flush per scrivere la pagine di transazioni che hanno fatto commit

Confronto tra steal/no-steal e force/no-force:

- Dal punto di vista del recovery, no-steal e force sono più semplici da realizzare:
  - Con no-steal, non dobbiamo disfare (undo) le transazioni che fanno rollback
  - Con force, non dobbiamo rifare (redo) le azioni di una transazione che ha eseguito il commit, ma i cui effetti non sono stati registrati nella base di dati a fronte di un guasto
- Tuttavia
  - Con no-steal possiamo essere costretti a tenere nel buffer pool molte pagine attive
  - Con force, si possono avere numerose operazioni di I/O
- La strategia steal,no-force è in realtà la più usata, perché assicura la maggiore efficienza del gestore del buffer

Altre primitive

- Primitiva Unfix:
  - la transazione ha finito di usare una pagina in un frame
  - si decrementa pin-counter per quel frame
- Primitiva Use:
  - la transazione accede ad una pagina in un frame del buffer, e la modifica
  - si pone dirty=true
- Primitiva Force: trasferisce in modo sincrono (cioè la transazione attende la fine dell'operazione) nella memoria di massa una pagina che si trova in un frame
- Primitiva Flush: trasferisce in modo asincrono (viene eseguita quando li gestore non è indaffarato) nella memoria di massa le pagine di una transazione che si trovano nei frame

2.2
Gestione del recovery

Il gestore del recovery

É il modulo responsabile di:

- Eseguire l'inizio delle transazioni
- Eseguire il commit delle transazioni
- Eseguire il rollback delle transazioni
- Rispristinare lo stato corretto della base di dati a fronte di guasti e malfunzionamenti

Utilizza una struttura di dati complessa, chiamata file di log

Tipi di guasti e cause di rollback

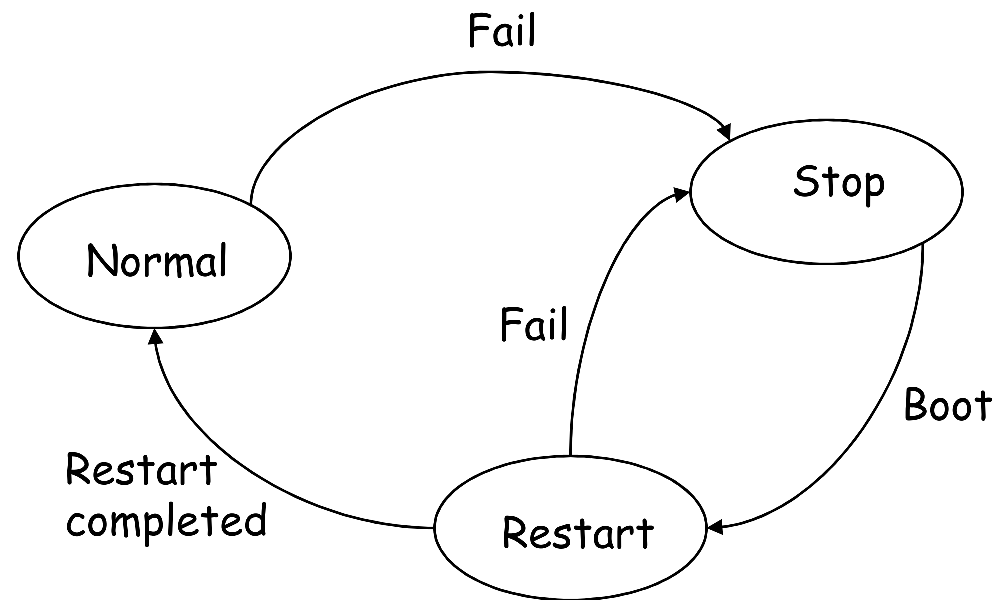
- Guasti di sistema
  - System crash (ad esempio per calo di tensione):
    - Perdita del contenuto del buffer, memoria di massa rimane valida
  - Errore di sistema o applicativo (della transazione)
    - E.g. divisione per zero
  - Condizioni locali durante l'esecuzione di una transazione (e.g. assenza di un dato)
  - Gestione della concorrenza
    - Lo scheduler può forzare il rollback e restart di una transazione
- Guasti di dispositivo di memorizzazione
  - Guasto di disco
    - Perdita fisica di dati su memoria di massa ma non del log
  - Problemi "catastrofici"
    - Incendio
    - Alluvione
    - Attentato

Le strategie dipendono dal tipo di guasto

- **Guasti di sistema:**
  - Perdita del buffer, stato persistente intatto
  - Problema principale → **Atomicità**
  - Strategia di recovery:
    - Fare periodicamente la "fotografia" della situazione (**checkpoint**)
    - Ripercorrere all'indietro la storia dei cambiamenti di stato del DB
    - Disfare (undo) alcune operazioni, ripeterne altre (**redo**)
    - Utilizza informazioni sulle operazioni effettuate (**log**)

- **Guasti di dispositivo:**
  - Perdita dello stato persistente
  - Problema principale → **Durabilità**
  - Strategia di recovery:
    - Caricare il più recente stato disponibile del DB
    - Ricostruire lo stato attuale utilizzando il log a partire dall'ultimo stato della BD noto (**dump**)

Principio generale per il ripristino: il modello fail-stop



Il file di log

- Il file di log (o semplicemente log) registra le azioni sulla base di dati da parte delle transazioni, ed è in memoria stabile, cioè memoria resistente ai guasti
- La lettura e la scrittura sul log avvengono come le corrispondenti operazioni sulla base di dati, e cioè attraverso il buffer. La scrittura sulla memoria stabile avviene di norma con “force”
- La memoria stabile è un’astrazione, benché tecniche di replicazione su dispositivi fisici multipli possano portare la probabilità di perdita di dati vicina allo zero
- L’organizzazione fisica della memoria stabile può differire a seconda della criticità delle applicazioni:
  - Unità nastro
  - Dispositivi di tipo diverso accoppiati (ad esempio: un nastro e un disco)
  - Due unità disco replicate

Organizzazione del log

- Il log può essere pensato come un file sequenziale che si assume scritto su memoria stabile, cioè per la quale non vi possono essere guasti. Le operazioni tipiche su tale file sono: appendere un record alla fine, scandire sequenzialmente in avanti, scandire sequenzialmente all'indietro.

- Registra le attività svolte dalle transazione e dal sistema, in ordine cronologico

- Due tipi di record compaiono nel log:
  - Record di transazione (begin, insert, delete, update, commit, abort)
  - Record di sistema (checkpoint, dump)

- Occorre ricordare che sia la memoria di massa sia il buffer contribuiscono a formare la base di dati. Non bisogna confondere le azioni delle transazioni sulla base di dati con le operazioni sulla memoria di massa: in genere i due tipi di operazioni sono asincrone
  - Le azioni delle transazioni sulla base di dati vengono considerate effettuate quando sono registrate nel log, anche se ancora non sono state effettuate sulla memoria di massa

Record di transazione

O = oggetto (o elemento) su cui si lavora

AS = After State, valore di O dopo l'operazione

BS = Before State, valore di O prima dell'operazione

I record di transazione (per ogni transazione T) sono memorizzati nel log nel seguente modo:

- a fronte di **`begin`**: B(T)
- a fronte di **`insert`**: I(T,O,AS)
- a fronte di **`delete`**: D(T,O,BS)
- a fronte di **`update`**: U(T,O,BS,AS)
- a fronte di **`commit`**: C(T)
- a fronte di **`abort`**: A(T)

Checkpoint

- Lo scopo del checkpoint è quello di registrare l'insieme delle transazioni attive T1, …, Tn distinguendole da quelle completate (committed)
- La primitiva di checkpoint (CK) esegue due operazioni
  - Per ogni transazione committed dopo il precedente checkpoint, copia le pagine di buffer sulla memoria di massa (con la primitiva flush)
  - Scrive (con force su memoria stabile) un record CK(T1,…Tn) contentente gli identificatori delle transazioni T1,…Tn attive (uncommitted) all'atto della esecuzione del checkpoint
- Conseguenza:
  - Per tutte le transazioni T per cui Commit(T) *precede* CK(T1,…Tn), non è necessario alcun "redo" in caso di guasto

- Il checkpoint viene eseguito periodicamente, con frequenza prefissata

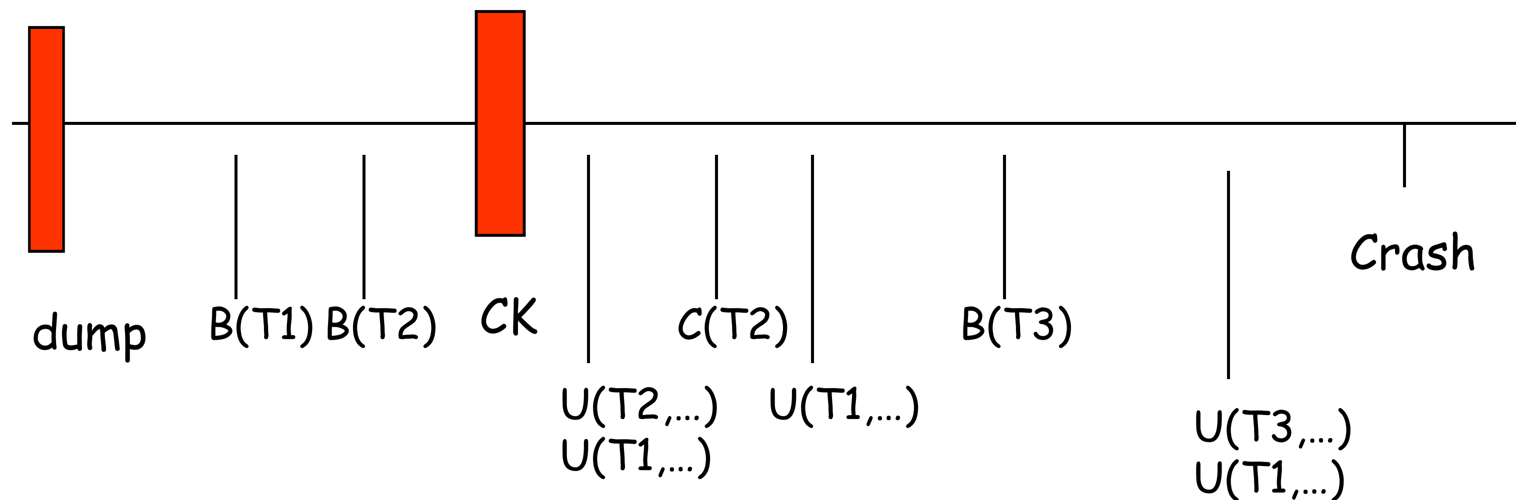
Sequenza di checkpoint

1. Sospende l'esecuzione delle transazioni
2. Flush del buffer (pagine attive non ancora trasferite di transazioni committed)
3. Scrive il record CK(T1,…Tn) su log in modo sincrono (force su memoria stabile)
4. Riprende l'esecuzione delle transazioni

Dump

- Un dump è una copia dell'intero stato del DB
- L'operazione di dump viene eseguita offline (nessuna transazione attiva)
- Produce un backup, cioè un salvataggio su memoria stabile
- Scrive (force su memoria stabile) un record di dump nel log, che segnala che in quell'istante è stata fatta una copia della BD

Esempio di log con checkpoint e dump



Record di end

- Alcuni sistemi registrano nel log anche il record di end per ogni transazione, che attesta che le operazioni di flush per una transazione sono terminate

- In presenza del record di end, è possibile individuare una classe di transazioni per le quali non è necessario effettuare nessuna azione, né di undo né di redo.

- In genere il record di end non è previsto, e noi lo ignoreremo

Undo di un'azione

- Ripristina lo stato di un elemento O al tempo *precedente* rispetto all'esecuzione di una azione

- `update`, `delete`:
  - assegna il valore *BS* ad O

- `insert`:
  - cancella O

Redo di un'azione

- Ripristina lo stato di un data item O al tempo *successivo* rispetto all'esecuzione di una azione

- `insert`, `update`:
  - assegna il valore AS ad O

- `delete`:
  - cancella O

Idempotenza di Undo e Redo

- Durante il ripristino, possono verificarsi errori
- Può essere necessario eseguire ripetutamente le azioni associate a Undo e Redo
- Valgono le proprietà di idempotenza:

(Undo(A) ; Undo(A)) = Undo(A)

(Redo(a) ; Redo(A)) = Redo(A)

Atomicità delle transazioni

- L'esito di una transazione è stabilito quando viene scritto il record Commit(T) oppure Abort(T) nel log (cioè nella memoria stabile)
  - Commit(T) viene scritto in modo sincrono (force) dal buffer nel log in memoria stabile
  - Abort(T) viene scritto in modo asincrono (flush) dal buffer nel log in memoria stabile (per il recovery non è fondamentale sapere immediatamente che una trasazione ha abortito)

- Per le transazioni che al momento di un guasto sono
  - Uncommitted: è necessario disfare le azioni, perchè non è noto quali azioni siano state effettuate sulla memoria di massa, ma deve essere garantita la atomicità → Undo
  - Committed: è necessario rifare le azioni, per garantire la durabilità (commit significa che la transazione si è impegnata a eseguire tutte le proprie azioni) → Redo

Politiche di scrittura dei record di log nella memoria stabile

Il recovery manager segue sempre questa regola:

- ***WAL* (write-ahead log)**
  - I record di log, in particolare quelli che contengono BeforeState, vengono scritti dal buffer su memoria stabile prima dei corrispondenti record nella memoria di massa
  - Questo rende efficace la procedura di Undo, perché un valore può sempre essere riscritto su memoria di massa utilizzando il BS scritto nel log in memoria stabile. In altre parole WAL consente di disfare scritture fatte da transazioni non ancora committed

Politiche di scrittura dei record di log nella memoria stabile

Il recovery manager segue sempre questa regola:

- ***Commit-Precedence***
  - I record di log, per la parte AfterState, vengono scritti dal buffer su memoria stabile prima di effettuare il commit della transazione (e quindi prima di scrivere il record di commit nel log in memoria stabile)
  - Questo rende efficace il Redo, perché se una transazione è committed al momento di un guasto, ma le sue pagine non sono ancora state scritte su disco, si può usare i valore AS che si trova nel log in memoria stabile. In altre parole la regola del Commit-Precedence consente di rifare transazioni che hanno fatto commit, anche se i loro effetti non sono stati ancora registrati in memoria di massa

Politiche di scrittura sulla memoria di massa

Per ciascuna delle tre tipologie di operazioni sulla base di dati

- Update
- Insert
- Delete

il recovery manager deve seguire delle politiche di esecuzione per la scrittura sulla memoria di massa

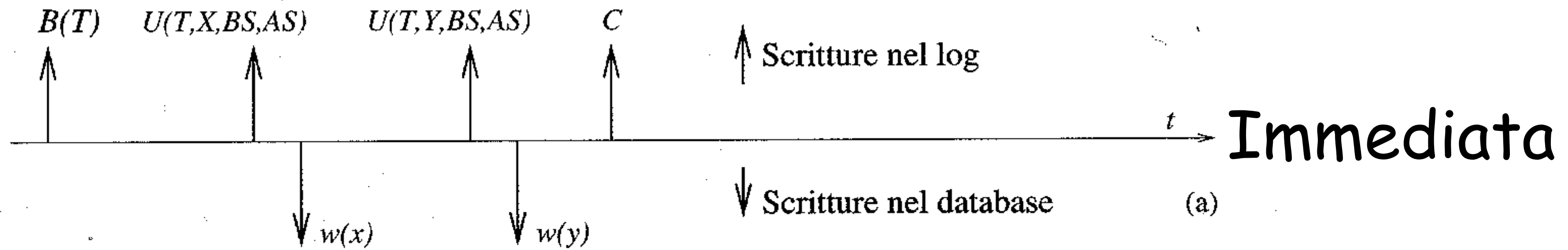
Analizzeremo la situazione per l'operazione di update, ma le considerazioni valgono per tutte le operazioni

Politiche di scrittura sulla memoria di massa

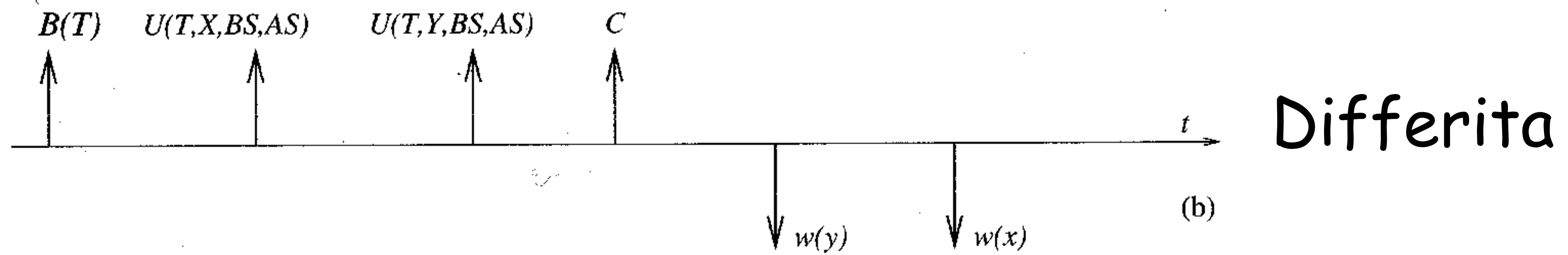
Tre possibili strategie di scrittura dei valori della base di dati nella memoria di massa, tutte coerenti con le regole WAL e commit-precedence

- Effetto immediato
  - Gli update vengono effettuati sulla base di dati in memoria di massa immediatamente (i record di log vengono comunque scritti prima dei dati corrispondenti)
  - Il buffer manager quindi si impegna (con force o flush) a scrivere sulla base di dati in memoria di massa prima del commit sul log
  - In questo modo tutte le pagine della base di dati modificate dalla transazioni sono certamente scritte su memoria di massa
- Effetto differito
  - Gli update vengono effettuati sulla base di dati in memoria di massa solo dopo il commit della transazione e la conseguente scrittura del record di commit nel log
  - I record di log vengono comunque scritti prima dei dati corrispondenti
- Effetto misto
  - Sono possibili entrambe le politiche, in funzione della efficienza (il record di log di una operazione P viene comunque scritto prima dei dati scritti sulla base di dati dalla operazione P)

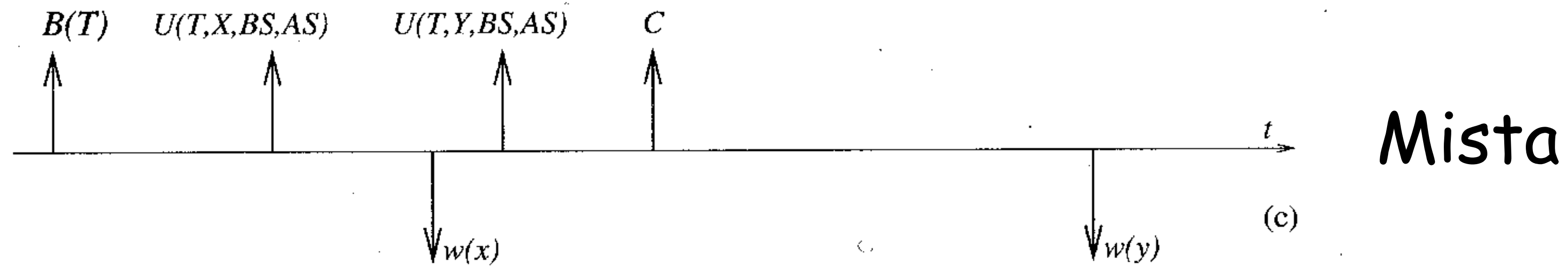
Esempi



Immediata



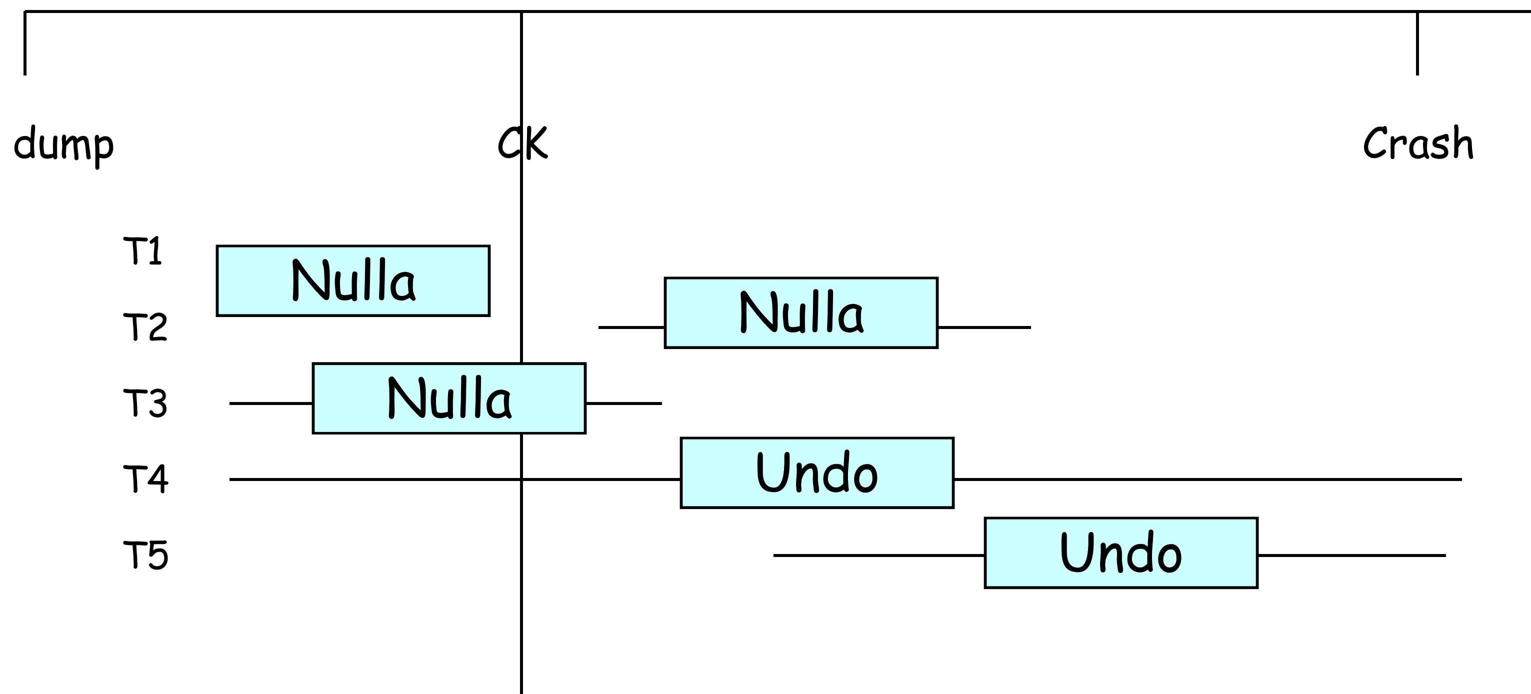
Differita



Mista

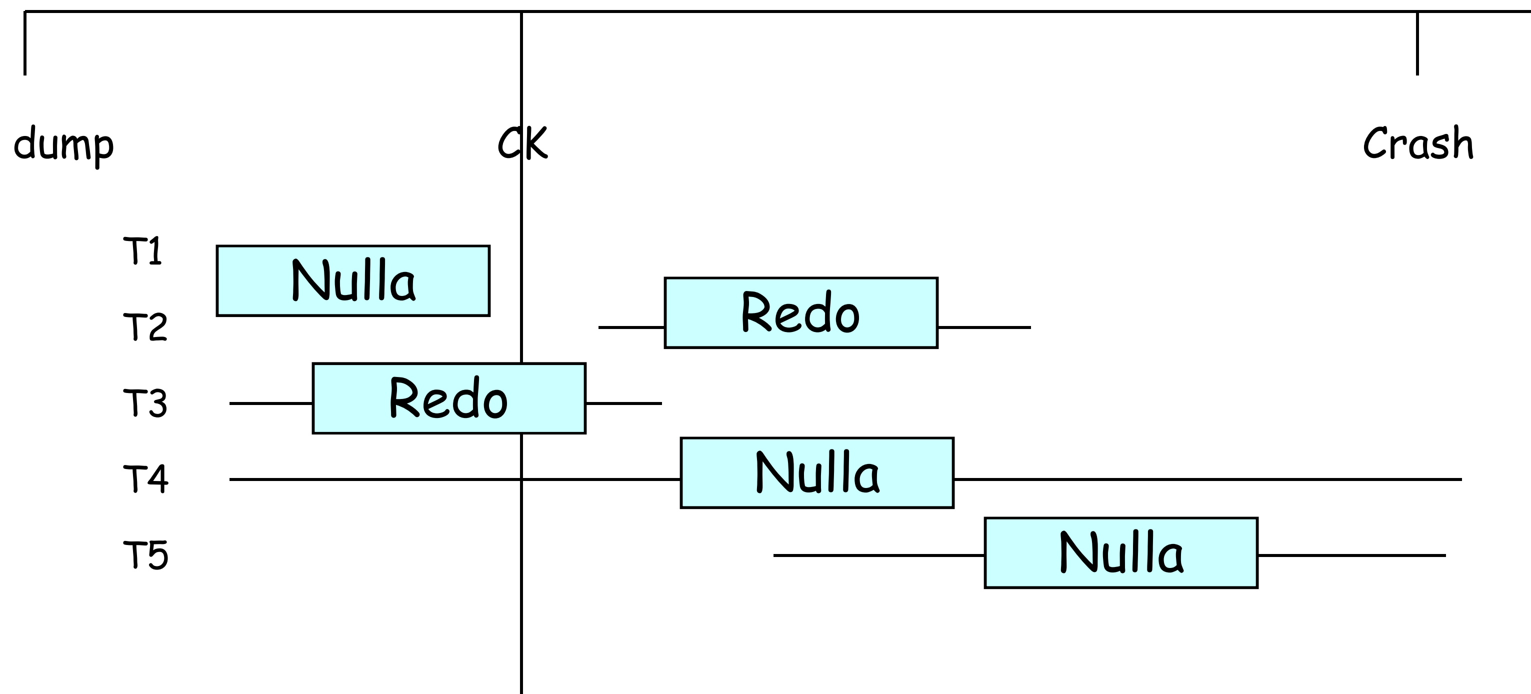
Modalità immediata

- La memoria di massa contiene valori AS provenienti da transazioni uncommitted
- Richiede Undo delle operazioni di transazioni uncommitted al momento del guasto
- Non richiede Redo (se nel log è presente il commit di T, le pagine sono state già scritte da T in memoria di massa)



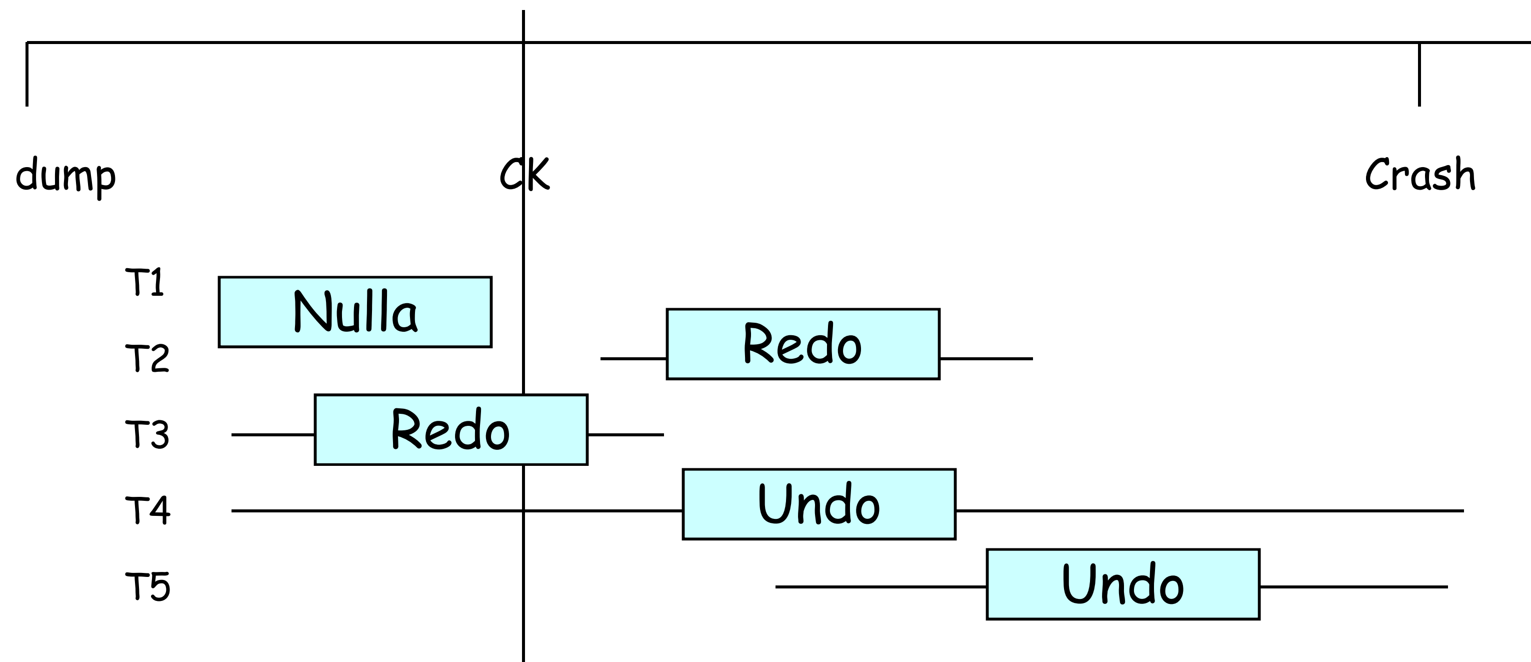
Modalità differita

- La memoria di massa non contiene valori AS provenienti da transazioni uncommitted
- Rende superflua la procedura di Undo (perché al momento dell'abort, nulla è stato fatto sulla memoria di massa)
- Richiede Redo



Modalità mista

- La scrittura può avvenire in modalità sia immediata che differita, a scelta del gestore del buffer (e per questo è la più usata). In particolare, questa modalità consente l'ottimizzazione delle operazioni di flush
- Richiede sia Undo che Redo



Due tipi di recovery

A seconda del tipo di guasto

- In caso di guasto di sistema:
  - Warm restart (ripresa a caldo)

- In caso di guasto di dispositivo:
  - Cold restart (ripresa a freddo)

Warm restart

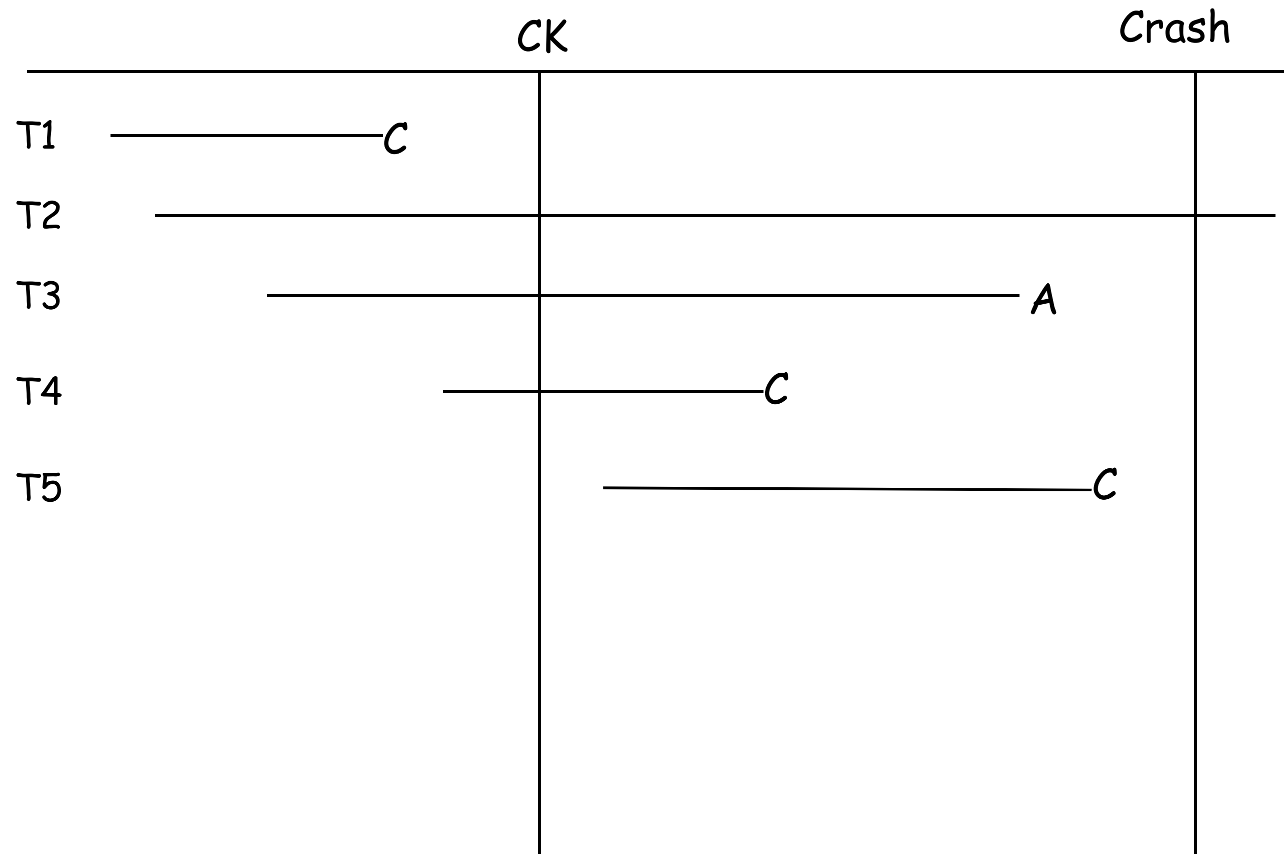
Assumiamo la modalità mista.

Il warm restart si compone di 5 fasi:

1. Si ripercorre il log all'indietro fermandosi all'ultimo checkpoint (il più recente) nel log
2. Si pone set(UNDO) = { transazioni attive al checkpoint } set(REDO) = { }
3. Si analizza il log in avanti aggiungendo a set(UNDO) le transazioni con begin, e spostando in set(REDO) quelle che hanno eseguito il commit
4. Fase Undo: si ripercorre il log *all'indietro* disfacendo le transazioni nel set Undo fino al begin della transazione più vecchia (si può andare indietro prima dell'ultimo checkpoint)
5. Fase Redo: si ripercorre il log *in avanti* rifacendo le transazioni nel set Redo

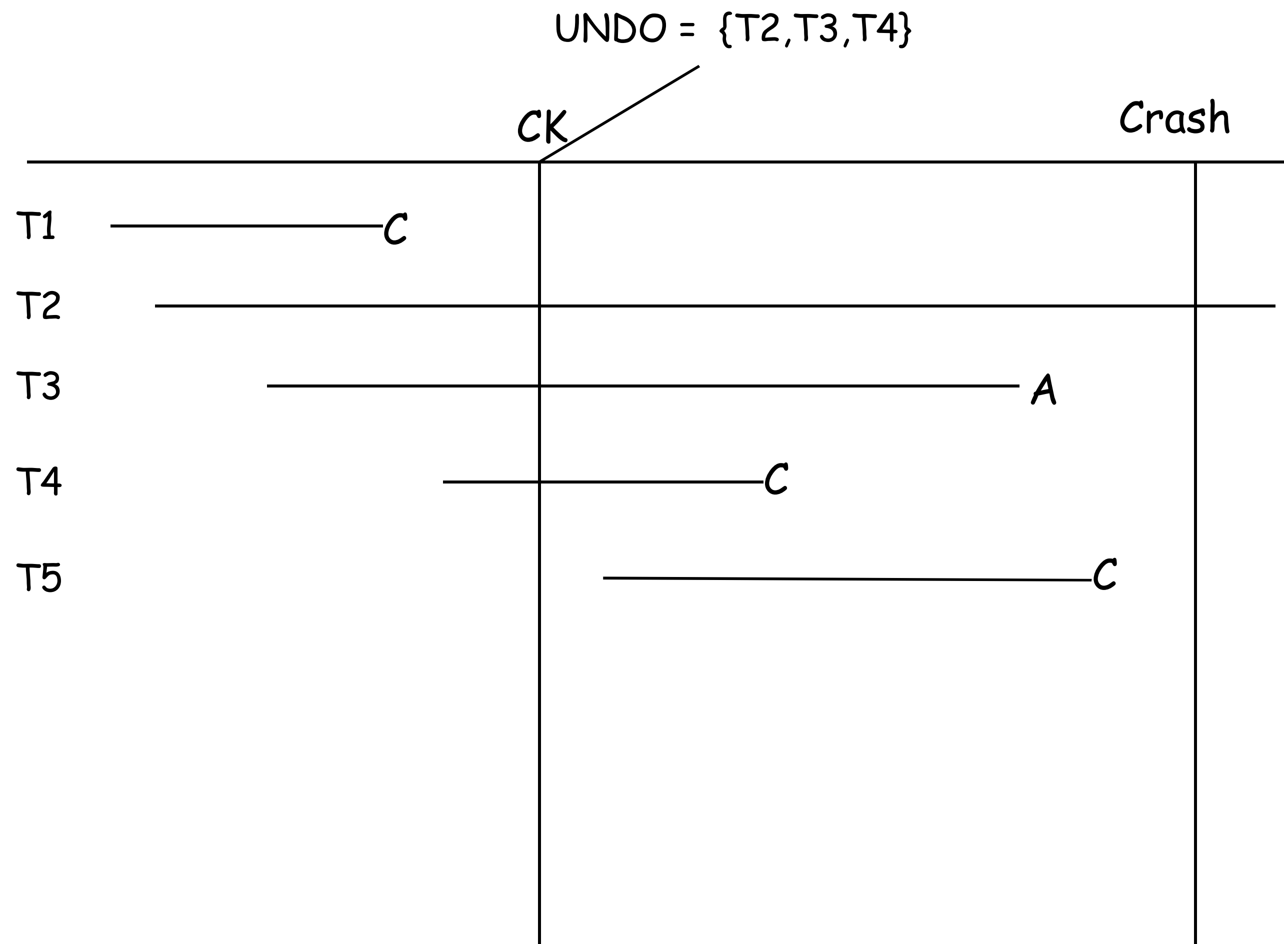
Esempio di warm restart

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,O5,B7)
A(T3)
C(T5)
I(T2,O6,A8)



Esempio: ricerca dell'ultimo checkpoint

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,O5,B7)
A(T3)
C(T5)
I(T2,O6,A8)



Esempio: set(UNDO) e set(REDO)

B(T1)
B(T2)
8. U(T2, O1, B1, A1)
I(T1, O2, A2)
B(T3)
C(T1)
B(T4)
7. U(T3,O2,B3,A3)
9. U(T4,O3,B4,A4)

1. C(T4)
2. B(T5)
6. U(T3,O3,B5,A5)
10. U(T5,O4,B6,A6)
5. D(T3,O5,B7)
A(T3)
3. C(T5)
4. I(T2,O6,A8)

0. UNDO = {T2,T3,T4}. REDO = {}

---

1. C(T4) → UNDO = {T2, T3}. REDO = {T4}

2. B(T5) → UNDO = {T2,T3,T5}. REDO = {T4}

3. C(T5) → UNDO = {T2,T3}. REDO = {T4, T5}

Esempio: fase di UNDO

B(T1)
B(T2)
8. U(T2, O1, B1, A1)
I(T1, O2, A2)
B(T3)
C(T1)
B(T4)
7. U(T3,O2,B3,A3)
9. U(T4,O3,B4,A4)

1. C(T4)
2. B(T5)
6. U(T3,O3,B5,A5)
10. U(T5,O4,B6,A6)
5. D(T3,O5,B7)
A(T3)
3. C(T5)
4. I(T2,O6,A8)

0. UNDO = {T2,T3,T4}. REDO = {}

---

1. C(T4) → UNDO = {T2, T3}. REDO = {T4}

2. B(T5) → UNDO = {T2,T3,T5}. REDO = {T4}

3. C(T5) → UNDO = {T2,T3}. REDO = {T4, T5}

---

4. D(O6)

5. O5 =B7

6. O3 = B5

7. O2 =B3

8. O1=B1

Fase di Undo

Esempio: fase di REDO

B(T1)
B(T2)
8. U(T2, O1, B1, A1)
I(T1, O2, A2)
B(T3)
C(T1)
B(T4)
7. U(T3,O2,B3,A3)
9. U(T4,O3,B4,A4)

1. C(T4)
2. B(T5)
6. U(T3,O3,B5,A5)
10. U(T5,O4,B6,A6)
5. D(T3,O5,B7)
A(T3)
3. C(T5)
4. I(T2,O6,A8)

0. UNDO = {T2,T3,T4}. REDO = {}

---

1. C(T4) → UNDO = {T2, T3}. REDO = {T4}

2. B(T5) → UNDO = {T2,T3,T5}. REDO = {T4}

3. C(T5) → UNDO = {T2,T3}. REDO = {T4, T5}

Fase di Undo

4. D(O6)
5. O5 =B7
6. O3 = B5
7. O2 =B3
8. O1=B1

Fase di Redo

9. O3 = A4
10. O4 = A6

Cold restart

Comprende tre fasi:

1. Ricerca del record di dump più recente nel log, e caricamento del dump nella memoria di massa (in pratica, si esegue una copia selettiva dei frammenti danneggiati della BD)
2. Forward recovery dallo stato del dump:
    1. si riapplicano tutte le azioni del log, nel loro ordine
    2. si ottiene così lo stato della BD immediatamente precedente al crash
3. Si esegue un warm restart come descritto in precedenza

Esercizio: cold restart

- Si consideri il seguente log: DUMP, B(T1), B(T2), B(T3), I(T1,O1,A1), D(T2,O2,B2), B(T4), U(T4,O3,B3,A3), U(T1,O4,B4,A4), C(T2), CK(T1,T3, T4), B(T5), B(T6), U(T5,O5,B5,A5), A(T3), CK(T1,T4,T5,T6), B(T7), A(T4), U(T7,O6,B6,A6), U(T6,O3,B7,A7), B(T8), C(T7)

- A questo punto della esecuzione il sistema subisce un guasto di dispositivo. Si assuma che la modalità di scrittura nella memoria di massa sia quella mista, e quindi richieda sia undo sia redo.

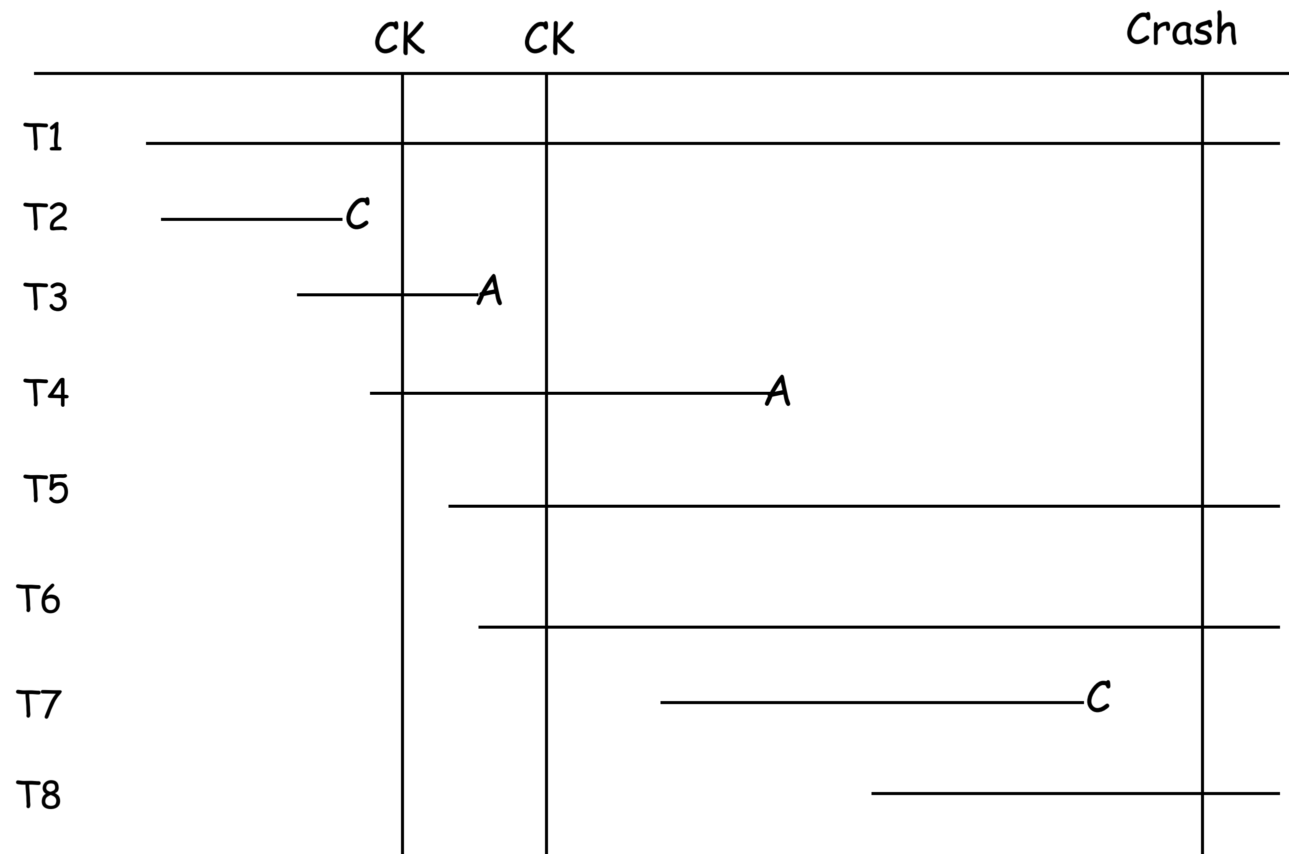
Soluzione: ricostruzione dal DUMP

- DUMP, B(T1), B(T2), B(T3), I(T1,O1,A1), D(T2,O2,B2), B(T4), U(T4,O3,B3,A3), U(T1,O4,B4,A4), C(T2), CK(T1,T3, T4), B(T5), B(T6), U(T5,O5,B5,A5), A(T3), CK(T1,T4,T5,T6), B(T7), A(T4), U(T7,O6,B6,A6), U(T6,O3,B7,A7), B(T8), C(T7)

- Si accede all'ultimo DUMP (la prima operazione riportata nel log), e si ricopia dal back-up in modo selettivo la parte di basi di dati danneggiata
- Si ripercorre in avanti il log a partire da B(T1), e si eseguono tutte le operazioni fino a C(T7)
- Si effettua un warm restart

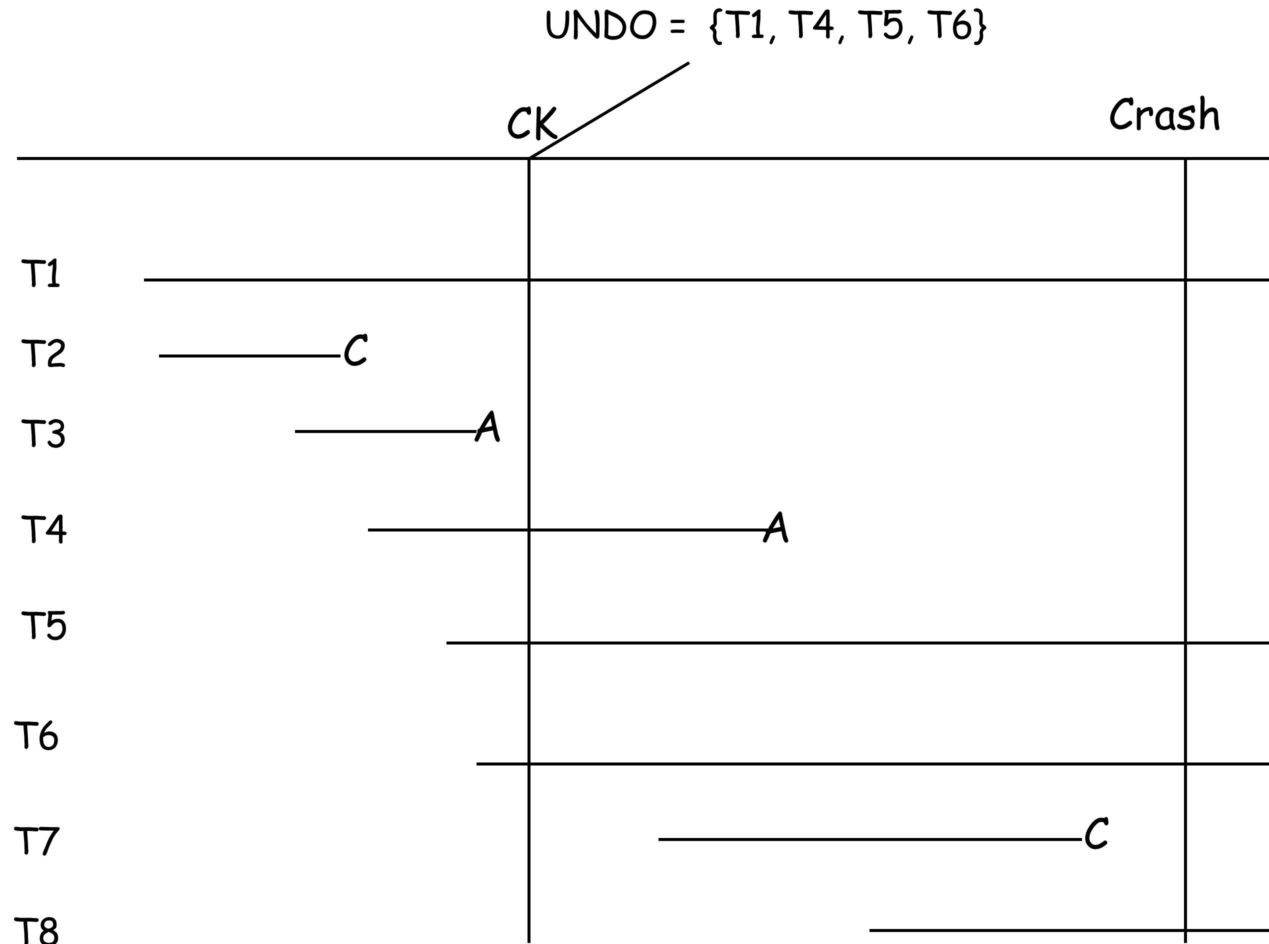
Soluzione: sequenza di warm restart

B(T1),
B(T2),
B(T3),
I(T1,O1,A1),
D(T2,O2,B2),
B(T4),
U(T4,O3,B3,A3),
U(T1,O4,B4,A4),
C(T2),
CK(T1,T3, T4),
B(T5),
B(T6),
U(T5,O5,B5,A5),
A(T3),
CK(T1,T4,T5,T6),
B(T7),
A(T4),
U(T7,O6,B6,A6),
U(T6,O3,B7,A7),
B(T8),
C(T7)

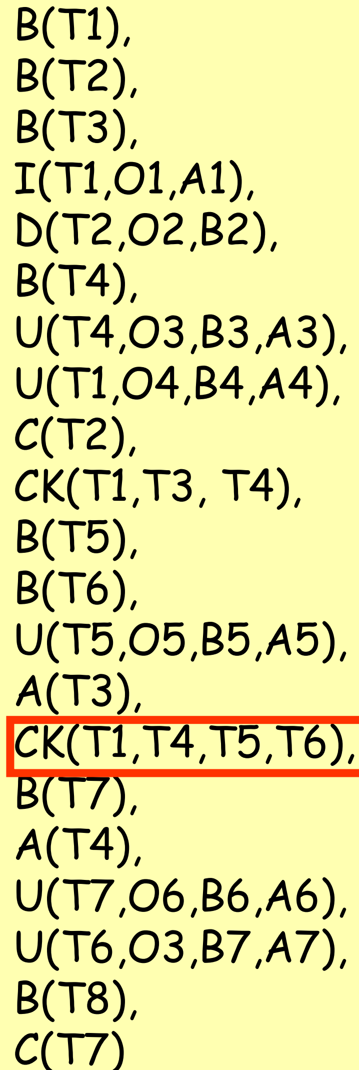


Soluzione: checkpoint più recente

B(T1),
B(T2),
B(T3),
I(T1,O1,A1),
D(T2,O2,B2),
B(T4),
U(T4,O3,B3,A3),
U(T1,O4,B4,A4),
C(T2),
CK(T1,T3, T4),
B(T5),
B(T6),
U(T5,O5,B5,A5),
A(T3),
CK(T1,T4,T5,T6),
B(T7),
A(T4),
U(T7,O6,B6,A6),
U(T6,O3,B7,A7),
B(T8),
C(T7)



Soluzione: insiemi UNDO e REDO



0. UNDO = {T1, T4, T5, T6}. REDO = {}

---

1. B(T7) → {T1, T4, T5, T6, T7}. REDO = {}
2. B(T8) → {T1, T4, T5, T6, T7, T8}. REDO = {}
3. C(T7) → {T1, T4, T5, T6, T8}. REDO = {T7}

Soluzione: fase di UNDO

B(T1),
B(T2),
B(T3),
I(T1,O1,A1),
D(T2,O2,B2),
B(T4),
U(T4,O3,B3,A3),
U(T1,O4,B4,A4),
C(T2),
CK(T1,T3, T4),
B(T5),
B(T6),
U(T5,O5,B5,A5),
A(T3),
CK(T1,T4,T5,T6),
B(T7),
A(T4),
U(T7,O6,B6,A6),
U(T6,O3,B7,A7),
B(T8),
C(T7)

0. UNDO = {T1, T4, T5, T6}. REDO = {}

---

1. B(T7) → {T1, T4, T5, T6, T7}. REDO = {}
2. B(T8) → {T1, T4, T5, T6, T7, T8}. REDO = {}
3. C(T7) → {T1, T4, T5, T6, T8}. REDO = {T7}

---

4. O3 = B7
5. O5 = B5
6. O4 = B4
7. O3 = B3
8. D(O1)

Undo

Soluzione: fase di REDO

B(T1),
B(T2),
B(T3),
I(T1,O1,A1),
D(T2,O2,B2),
B(T4),
U(T4,O3,B3,A3),
U(T1,O4,B4,A4),
C(T2),
CK(T1,T3, T4),
B(T5),
B(T6),
U(T5,O5,B5,A5),
A(T3),
CK(T1,T4,T5,T6),
B(T7),
A(T4),
U(T7,O6,B6,A6),
U(T6,O3,B7,A7),
B(T8),
C(T7)

0. UNDO = {T1, T4, T5, T6}. REDO = {}

---

1. B(T7) → {T1, T4, T5, T6, T7}. REDO = {}
2. B(T8) → {T1, T4, T5, T6, T7, T8}. REDO = {}
3. C(T5) → {T1, T4, T5, T6, T8}. REDO = {T7}

---

4. O3 = B7
5. O5 = B5
6. O4 = B4
7. O3 = B3
8. D(O1)

Undo

---

9. O6 = A6

Redo