# Recuperarea datelor

4

# Recuperarea datelor şi ACID

## Atomicitatea

- garantată prin refacerea efectului acțiunilor corespunzătoare tranzacțiilor necomise.

## Durabilitatea

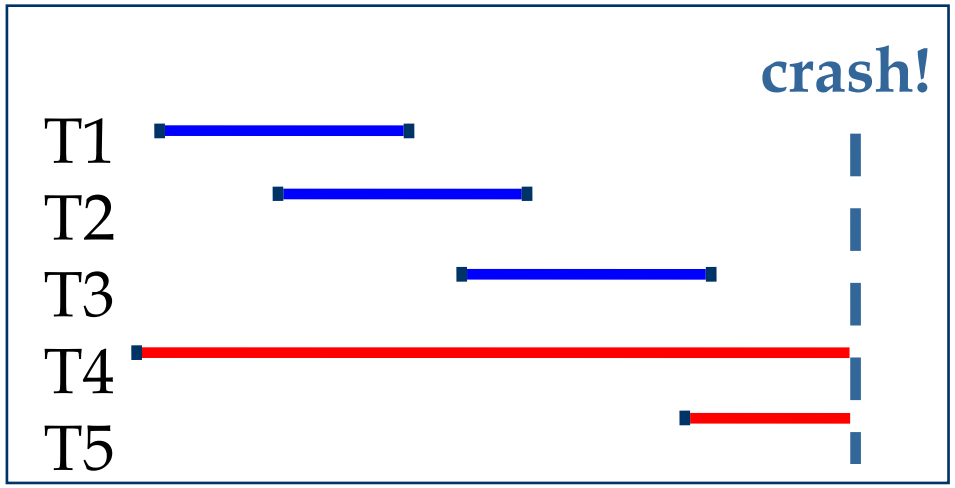
- garantată prin asigurarea faptului că toate acțiunile tranzacțiilor comise “rezistă” erorilor şi întreruperilor neaşteptate ale funcționării sistemului.

# Exemplu

- Atomicitate:
  - Execuția tranzacțiilor poate eşua.
- Durabilitate:
  - Ce se întâmplă daca SGBD-ul îşi opreşte execuția?

Comportamentul dorit după repornirea sistemului:

- T1, T2 & T3 trebuie să fie durabile.
- T4 & T5 trebuie să fie anulate (efectele nu vor persista).



# *Checkpoint*

- Este un punct de sincronizare între **memoria volatilă (RAM - Buffer Pool)** și **memoria stabilă (HDD/SSD - baza de date reală)**.

- Scop:
  - **Reducerea timpului de recuperare** după un crash al bazei de date.
  - **Evitarea reaplicării complete a Redo Log-urilor** în caz de restart.
  - **Optimizarea performanței** prin scrierea periodică a datelor pe disc în loc să o facă la fiecare tranzacție.

# *Checkpoint - Functionare*

- Oprirea temporară a tranzacțiilor noi – Sistemul de baze de date marchează începutul checkpoint-ului.

- Scrierea tuturor paginilor modificate ("Dirty Pages") din Buffer Pool pe disc – Aceasta se face pentru a persista modificările datelor (*dirty flag* = true).

- Actualizarea jurnalului de tranzacții (Redo Log & Undo Log) – Sistemul marchează checkpoint-ul în logurile tranzacțiilor pentru a indica faptul că toate tranzacțiile până la acel punct sunt confirmate.

- Reia execuția tranzacțiilor – Tranzacțiile noi pot continua, iar logurile tranzacțiilor pot fi curățate parțial.

# *Checkpoint – Recuperatea datelor*

- Daca baza de date se prăbușește, checkpoint-ul ajută la restaurarea datelor astfel:
  - Se identifică ultimul checkpoint valid în logurile tranzacțiilor.
  - Se reaplică doar tranzacțiile de după acel checkpoint din Redo Log.
  - Se anulează tranzacțiile incomplete folosind Undo Log.
  - Baza de date revine la un stadiu consistent, iar sistemul poate continua execuția normală.

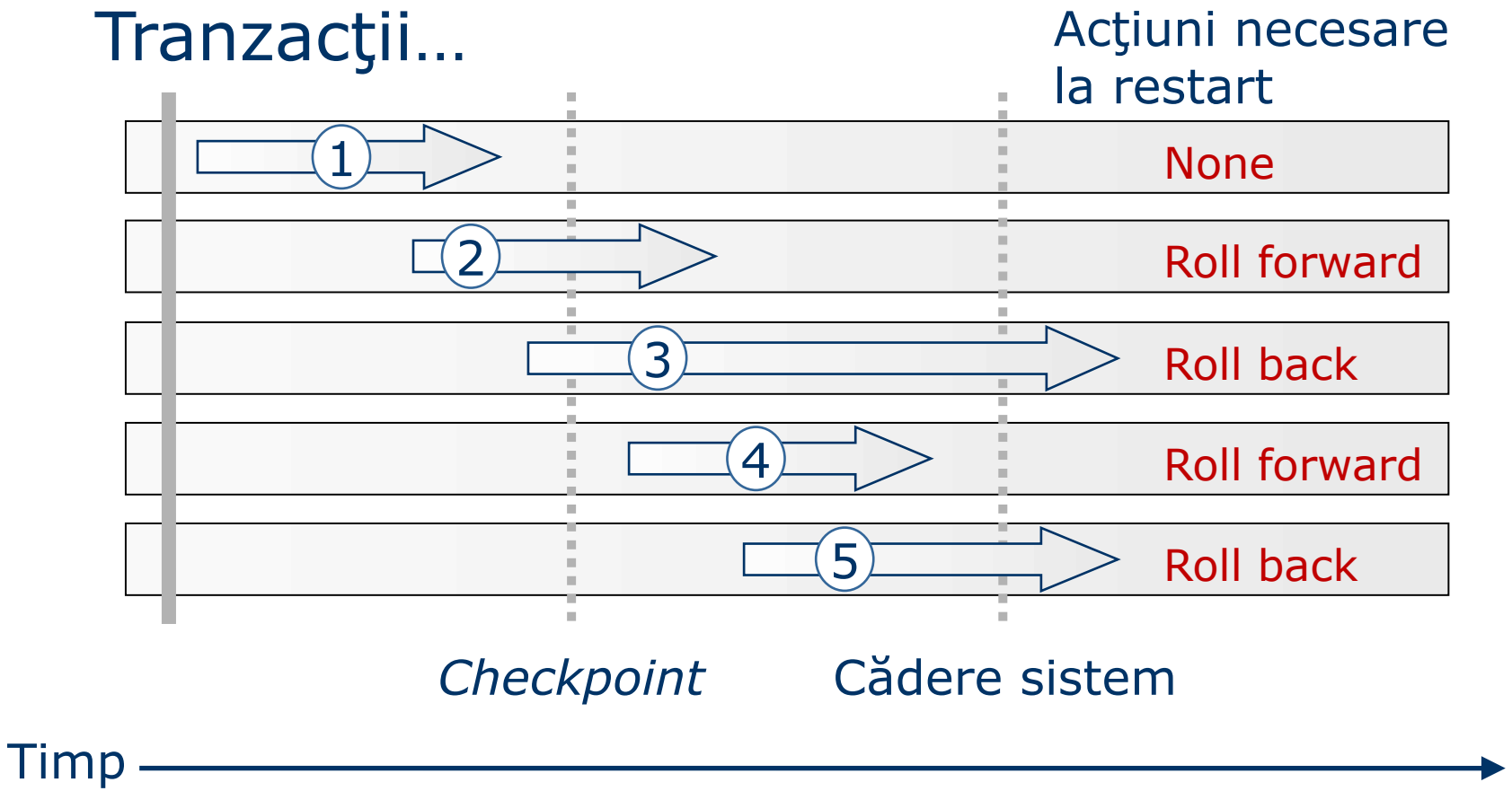
# *Checkpoint*

- Consecințe ?
  - Nu mai trebuie reexecutate acțiunile unei tranzacții care s-a comis înainte de *checkpoint*

# *Checkpoint*

- Cât de des se execută un *checkpoint*?
  - La fiecare *m* minute sau *t* tranzacții

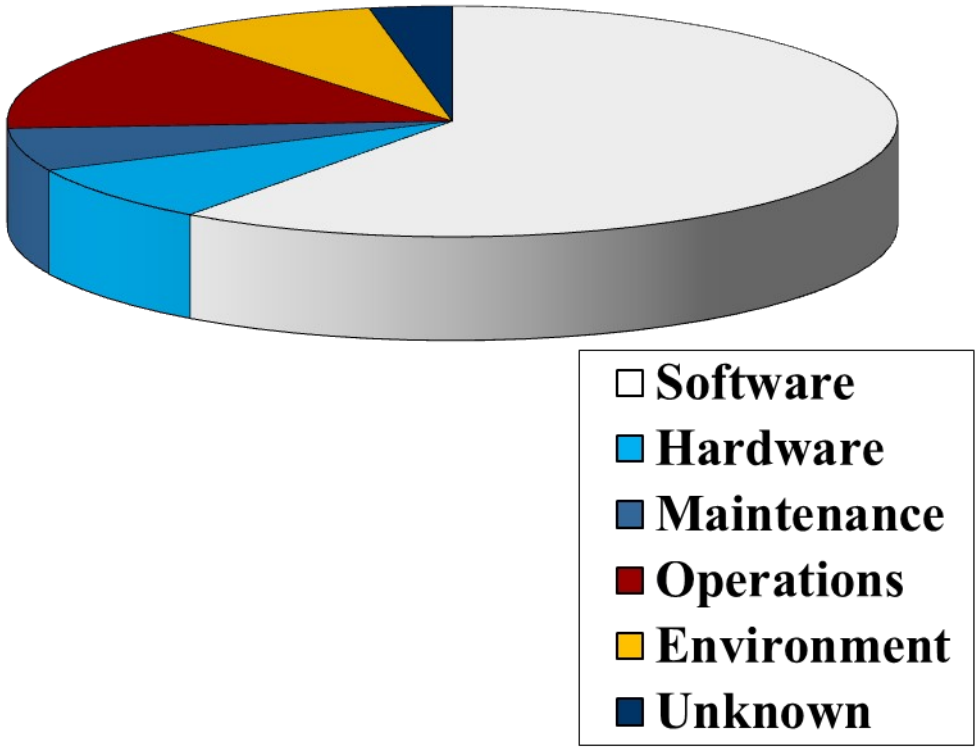
# Exemplu


Tranzacţii...
Acţiuni necesare la restart
1
None
2
Roll forward
3
Roll back
4
Roll forward
5
Roll back
Checkpoint
Cădere sistem
Timp

# Surse ale întreruperilor

- Căderi de sistem
- Erori media
- Erori ale aplicației
- Dezastre naturale
- Sabotaj
- Neglijență

# Impactul întreruperilor



# Categorii generale de întreruperi

(1) Eşuarea tranzacțiilor

- unilateral sau din cauza unui *deadlock*
- in medie 3% din tranzacții eşuează (date de intrare eronate, cicluri infinite, depăşirea limitei de resurse)

(2) Eşuarea sistemului

- Eşuarea procesorului, memoriei interne, etc…
- Conținutul memoriei interne se pierde însă memoria secundară nu este afectată

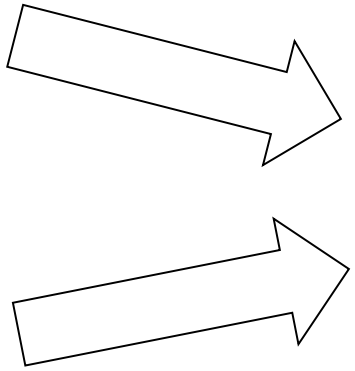
(3) Eşecuri media

- Pierdere date de pe hard disk

# Categorii generale de întreruperi

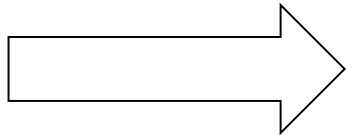
(1) Eşuarea tranzacţiilor

(2) Eşuarea sistemului



simple

(3) Eşecuri media



catastrofale

# Recuperarea datelor

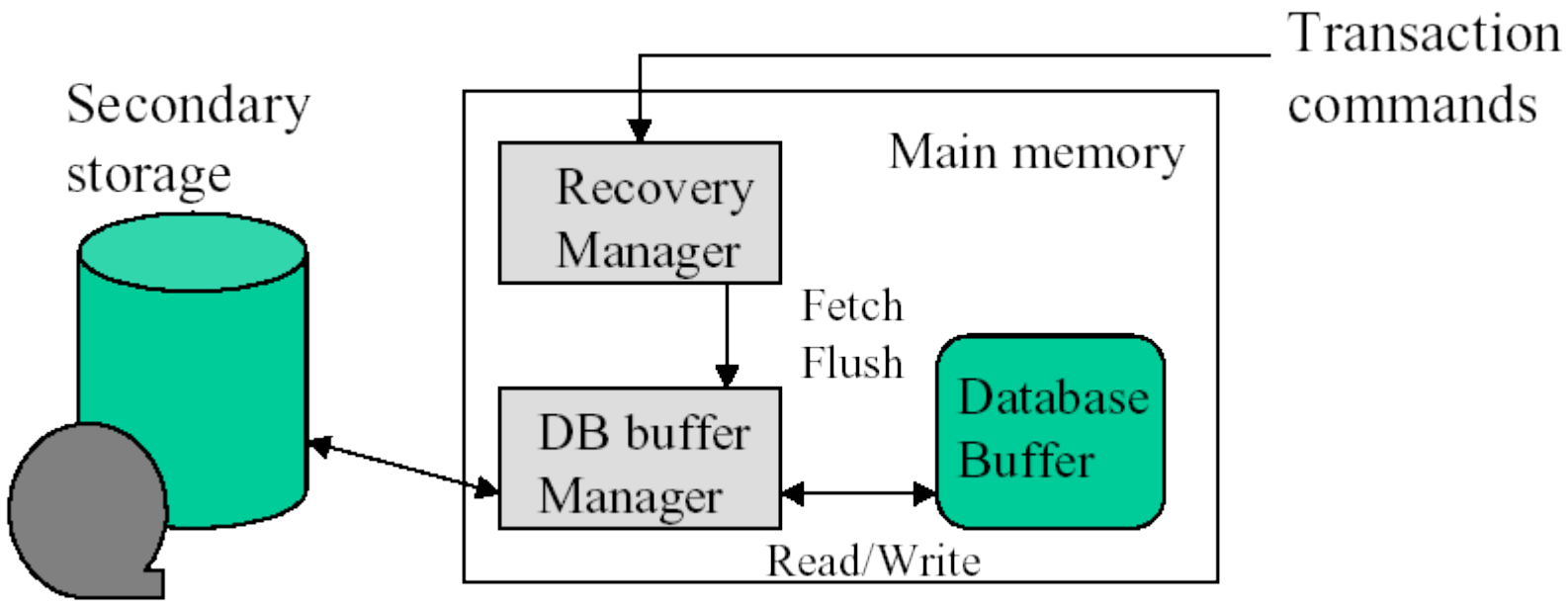
- Eşecuri simple
  - Se foloseşte logul de tranzacții
  - Anularea modificărilor prin **inversare** operații
  - **Re-executarea** unor operații
- Eşecuri catastrofale
  - Utilizarea arhivelor pentru **restaurare**
  - Reconstruirea celei mai recente stări consistente prin **re-executarea** acțiunilor tranzacțiilor comise

# Recuperarea datelor - Context

- Tranzacțiile se execută concurent
  - Strict 2PL, în particular.
- Modificările se execută "*in place*" (în același loc).
  - adică datele sunt actualizate pe disc / eliminate de pe disc din pagina de date originală.

- Există o metodă simplă care să garanteze Atomicitatea & Durabilitatea?

# *Recovery Manager*




- Memorie volatilă : memoria pricipală (conține *buffer*)
- Memorie stabilă : disc magnetic (sau variante). Rezistent la erori, iar datele se pierd numai atunci când are loc o eroare fizică sau un atac intenționat

# Logarea acțiunilor

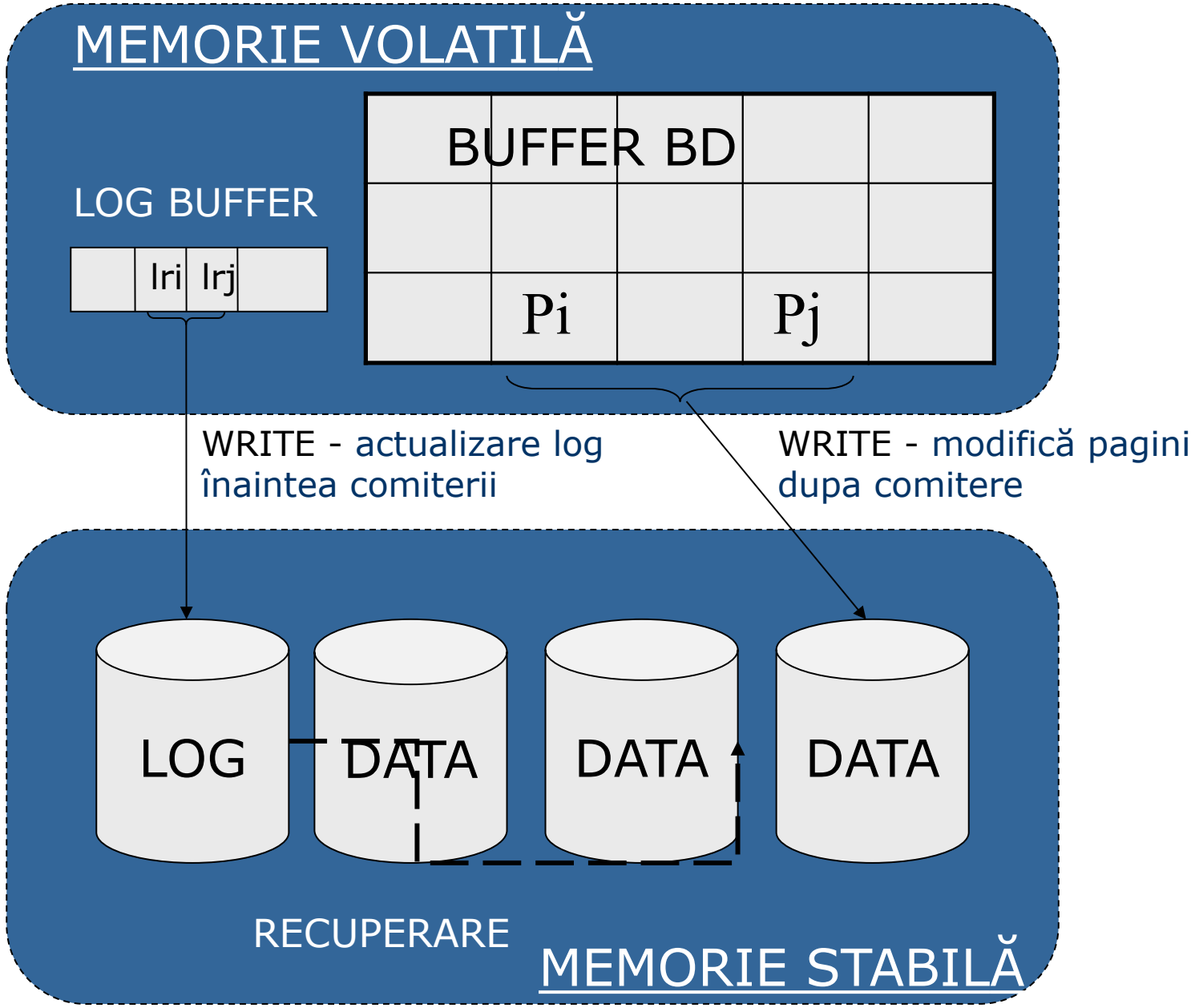
- Fiecare modificare → o intrare în log.
  - citirile nu se loghează
- De ce este nevoie de log?
  - Utilizat pentru a garanta atomicitatea şi durabilitatea.
- De obicei, logul se stochează pe un disc diferit de cel pe care se află baza de date.

# Buffer Pages

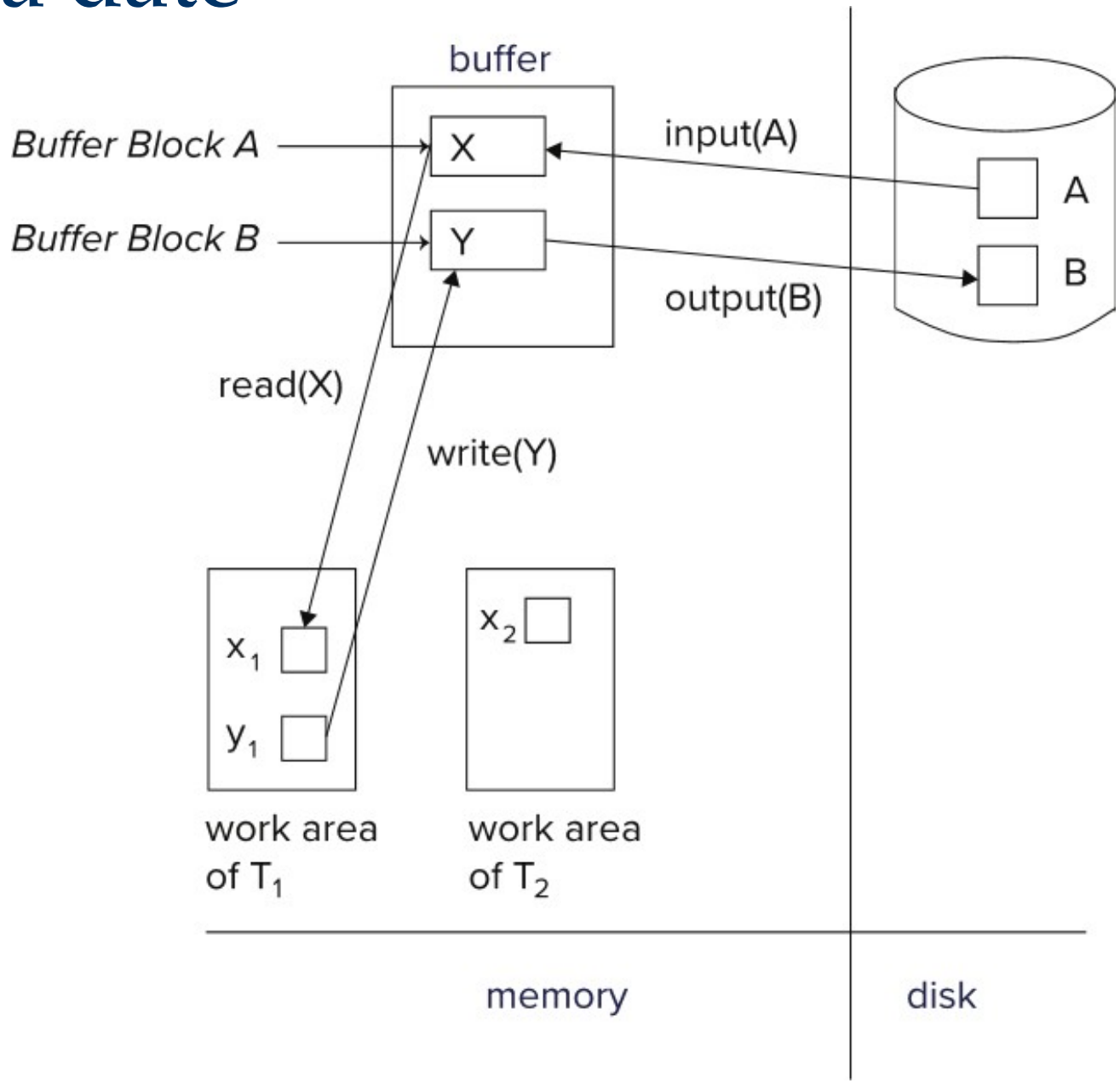
- Este o pagină de memorie în RAM care conține o copie a unei pagini de date de pe disc.
- Baza de date folosește aceste pagini pentru a citi și scrie date fără a accesa direct stocarea secundară (HDD/SSD), ceea ce îmbunătățește viteza de execuție a interogărilor.
- Caracteristici:
  - Reprezintă o copie temporară a unei pagini de date de pe disc.
  - Dimensiunea unei pagini variază în funcție de sistemul de baze de date (ex. 4KB, 8KB, 16KB).
  - Paginile sunt gestionate de un mecanism de alocare numit Buffer Pool.

# Buffer Pages – Mod de functionare

- Când o interogare necesită date din baza de date:
  - Căutarea în Buffer Pool: Dacă pagina este deja în memorie, aceasta este utilizată direct (evitând accesul la disc).
  - Citirea de pe Disc (dacă pagina nu este în memorie): Dacă pagina nu este în Buffer Pool, baza de date o încarcă din memoria stabilă (HDD/SSD).
  - Actualizarea paginii (pentru operațiuni de scriere): Dacă o pagină este modificată, aceasta devine "Dirty" și trebuie scrisă pe disc la un moment dat.
  - Evacuarea paginilor (Page Replacement): Când Buffer Pool este plin, baza de date eliberează pagini neutilizate, aplicând algoritmi precum LRU (Least Recently Used).

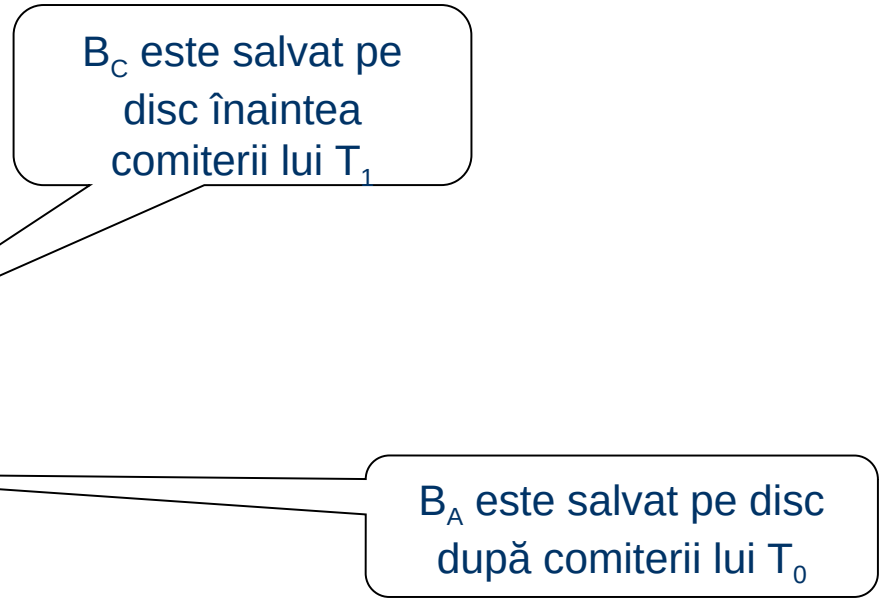

MEMORIE VOLATILĂ
LOG BUFFER
lri
lrj
BUFFER BD
Pi
Pj
WRITE - actualizare log
înaintea comiterii
WRITE - modifică pagini
dupa comitere
LOG
DATA
DATA
DATA
RECUPERARE
MEMORIE STABILĂ

# Exemplu de acces la date



# Exemplu

| Log | Write | Output |
|---|---|---|
| <$T_0$ **start**> | | |
| <$T_0$, A, 1000, 950> | | |
| <$T_0$, B, 2000, 2050> | | |
| | $A$ = 950<br>$B$ = 2050 | |
| <$T_0$ **commit**> | | |
| <$T_1$ **start**> | | |
| <$T_1$, C, 700, 600> | | |
| | $C$ = 600 | |
| | | $B_B$ , $B_C$ |
| <$T_1$ **commit**> | | |
| | | $B_A$ |




- Obs: $B_X$ reprezintă un bloc de memorie ce îl conține pe $X$.

# Log-ul bazei de date

- Logul conține înregistrări (sau intrări) adăugate mereu la final.
- Pentru recuperare logul este citit în ordine inversă
- O intrare in log conține:
  - Identificatorul tranzacției
  - Tipul operației (*inserare, ștergere, modificare*)
  - Obiectul accesat de către operație
  - Vechea valoare a obiectului
  - Noua valoare a obiectului
  - …

# Log-ul bazei de date

- Log-ul mai poate conține
  - *begin-transaction,*
  - *commit-transaction,*
  - *abort-transaction*.
  - *end*

- Dacă o tranzacție T e întreruptă, atunci se realizează un *rollback* → scanare inversă a log-ului, iar când se întâlnesc acțiuni ale tranzacției T, valoarea inițială a obiectului modificat este salvată în BD.

# Log-ul bazei de date

- ***begin-transaction*** **-** pentru oprirea căutării inverse

- La refacere contextului după o întrerupere:
  - ***commit*** → tranzacțiile *complete*
  - tranzacțiile *active* → ***abort***.

# Modificările bazei de date

O tranzacție T modifică obiectul x aflat în *buffer*. Dacă apare o întrerupere înainte de finalizarea execuției tranzacției:

**Scenariul 1**: Modificarea nu a reuşit să se salveze pe disc
→ T este anulată. BD consistent

# Modificările bazei de date

O tranzacție T modifică obiectul x aflat în *buffer*. Dacă apare o întrerupere înainte de finalizarea execuției tranzacției:

**Scenariul 2**: Modificarea lui x se salvează pe disc, dar întreruperea a survenit înaintea modificării logului → Nu se poate face *rollback* deoarece nu există informația despre valoarea anterioară a lui x → BD inconsistent.

# Modificările bazei de date

O tranzacție T modifică obiectul x aflat în *buffer*. Dacă apare o întrerupere înainte de finalizarea execuției tranzacției:

**Scenariul 3**: Modificarea lui x fost logată și s-a actualizat şi baza de date → T este anulată şi valoarea originală este utilizată pentru a înlocui valoarea din baza de date → BD consistent.

## *Write-Ahead Logging* (WAL)

Modificările unei înregistrări
trebuie inserate în *log*
înaintea actualizării bazei de date!

# *Write-Ahead Logging* (WAL)

- ***Write-Ahead Logging*** Protocol:

  1. Trebuie asigurată adăugarea unei intrări coresp. unei modificări în log *înainte* ca pagina ce conține înregistrarea sa fie salvată pe disc.

  2. Trebuie adăugate toate intrările corespunzătoare unei tranzacții înainte de *commit*. (Logul trebuie să fie salvat pe disc înainte ca modificarea să fie aplicată în buffer pool.)

  3. Dacă o tranzacție eșuează, se poate face un **rollback** folosind logurile.

- #1 garantează Atomicitatea.
- #2 garantează Durabilitatea.
- ARIES (*Algorithm for Recovery and Isolation Exploiting Semantics*) – metodă specifică de logare si recuperare a datelor