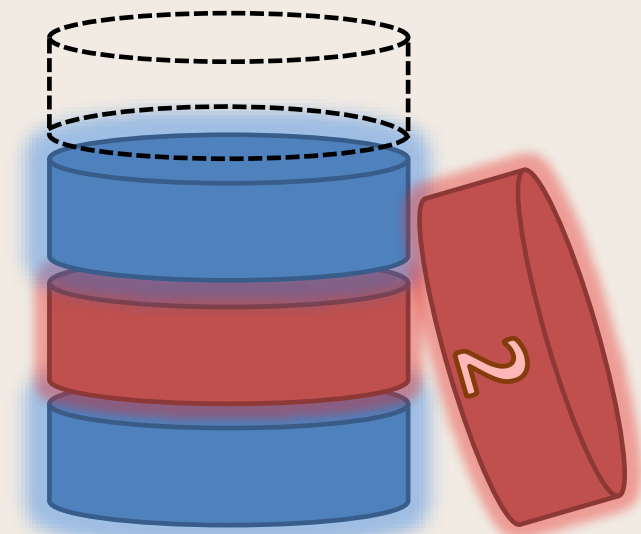


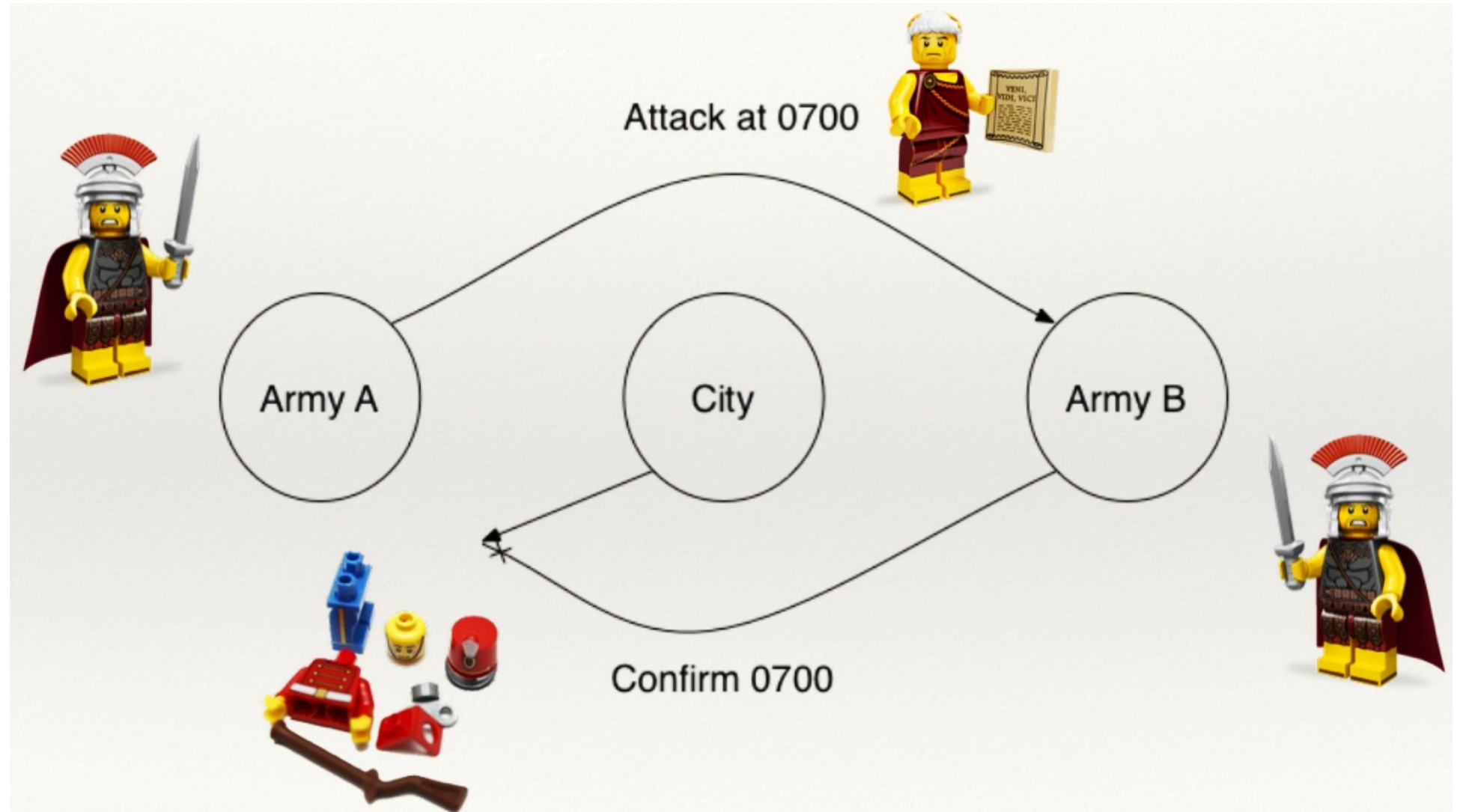
# Recuperarea datelor



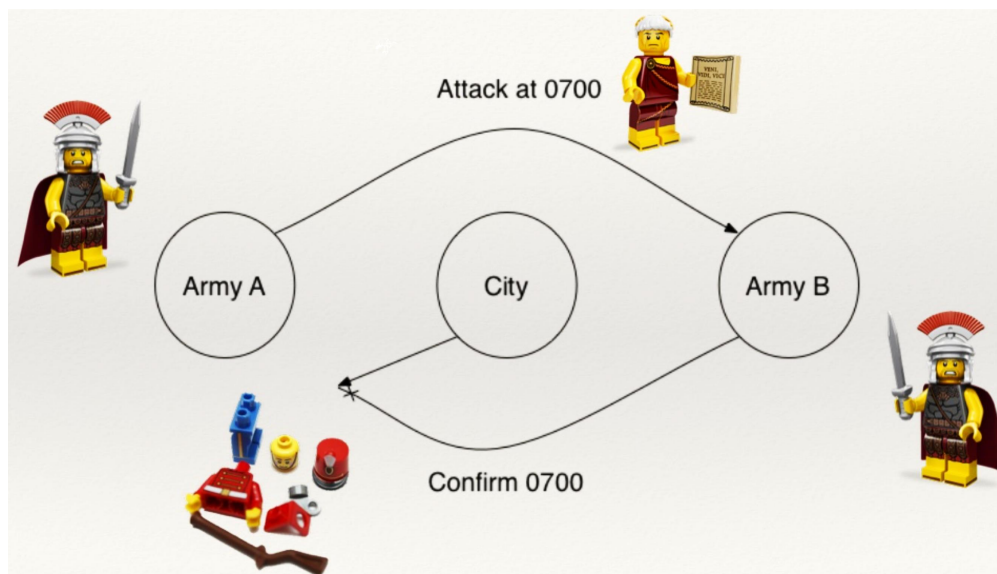
# Recuperarea distribuită

- Tipuri noi de eșec: întrerupere rețea și oprire *site*-uri
- Dacă “sub-tranzacțiile” unei tranzacții sunt executate pe *site*-uri diferite, trebuie să ne asigurăm că se vor comite toate sau nici una.
- E nevoie de un **protocol de comitere** a “sub-tranzacțiilor” unei tranzacții
  - Fiecare site are propriul log unde se vor memora acțiunile protocolului de comitere.

# Problema generalilor bizantini



# Problema generalilor bizantini

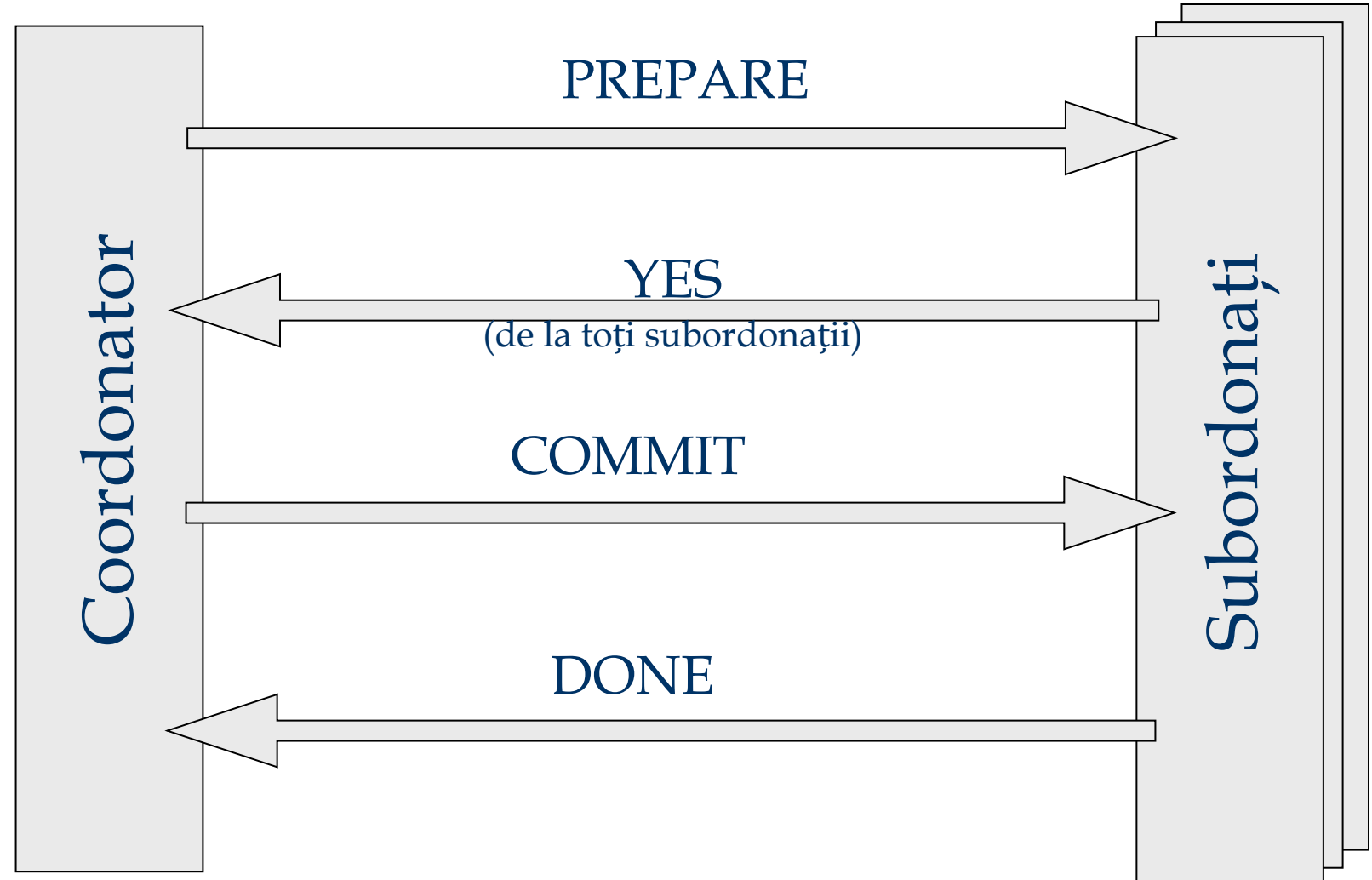


- Un oraș se află sub asediul a două armate aliate
- Fiecare armată are un general (unul dintre ei e liderul)
- Armatele trebuie să agreeze dacă atacă sau nu
- Comunică prin transmitere de mesaje
- Mesagerul poate fi capturat

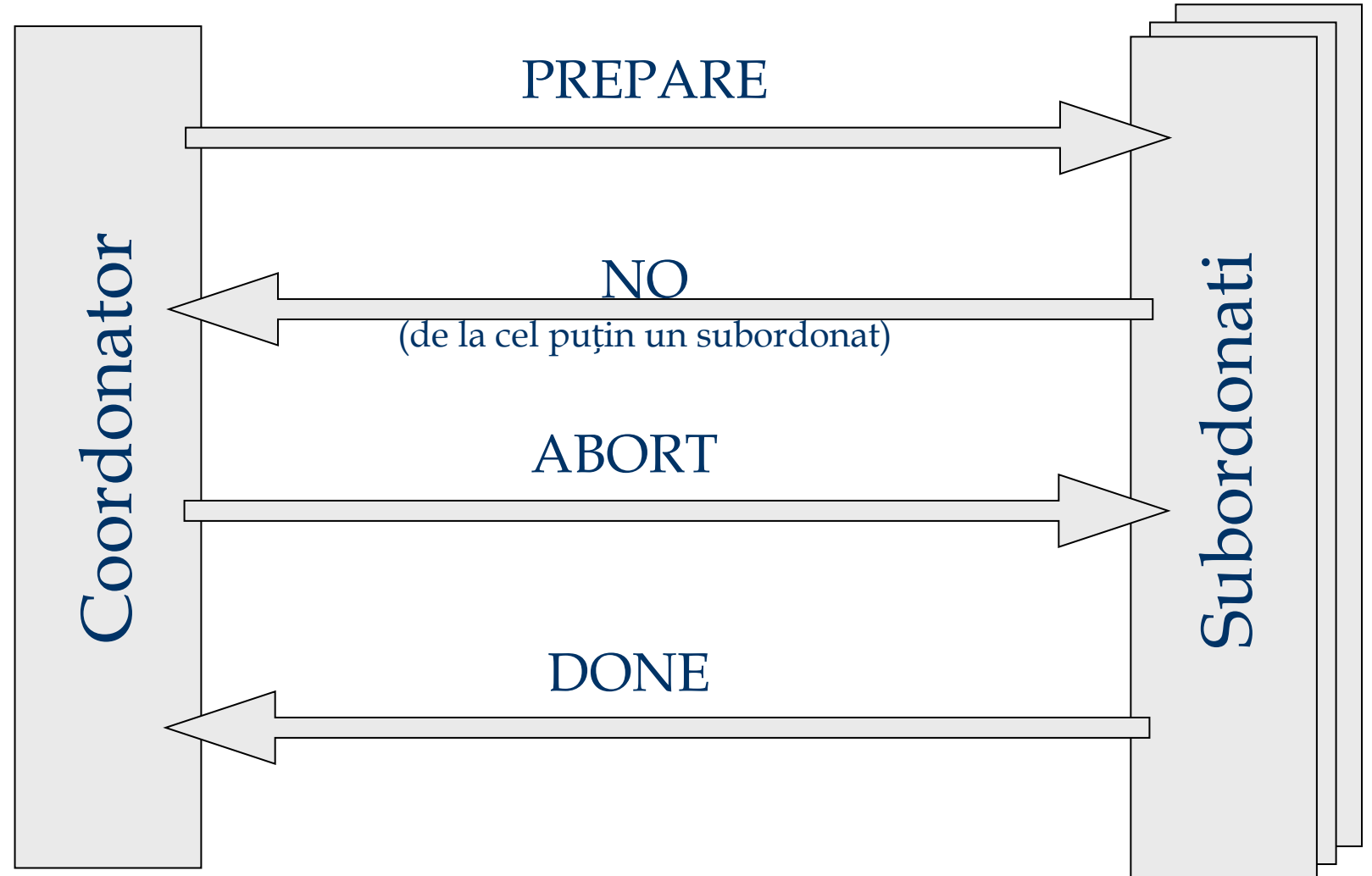
# Comitere în două faze (2PC)

- *Site*-ul de unde se generează tranzacția se numește **coordonator**; celelalte *site*-uri pe care se execută se numesc **subordonate**.
- Atunci când tranzacția comite:
  1. Coordonatorul transmite mesajul **prepare** tuturor subordonaților.
  2. Subordonații inserează **abort** sau **prepare** în log și apoi transmit mesajul **no** sau **yes** către coordonator.
  3. Dacă coordonatorul primește *yes* de la toți subordonații, inserează **commit** în log record și transmite **commit** tuturor. Altfel, inserează **abort** în log rec și transmite **abort** tuturor.
  4. Subordonații inserează **abort/commit** în log pe baza mesajului primit, apoi transmit **done** coordonatorului.
  5. Coordonatorul scrie **end** în log după ce primește toate *done*-urile.

# Comitere în două faze (2PC)



# Comitere în două faze (2PC)



# Comentarii asupra 2PC

- Două runde de comunicare: **votare** urmat de **terminare**. Ambele sunt inițiate de coordonator.
- Orice site poate decide eșuarea tranzacției.
- Fiecare mesaj reflectă o decizie; pentru a garanta că această decizie rezistă unor erori, ea este inserată mai întâi într-un log .
- Toate intrările în log conțin *TransactionID* și *CoordinatorID*. Comenzile abort/commit logate de către coordonator includ id-urile tuturor subordonaților.



## 2PC – Recuperarea datelor

- Dacă avem un **commit** sau **abort** logat pentru tranzacția T, dar nu este un **end**, se apelează *redo/undo* pentru T.
  - Dacă site-ul este coordonator pentru T, se vor transmite mesaje **commit/abort** către subordonați până se recepționează **done**.

## 2PC – Recuperarea datelor

- Dacă avem un **prepare** logat pentru tranzacția T, dar nu este **commit/abort**, iar site-ul este subordonat lui T.
  - se contactează coordonatorul în mod repetat pentru verificarea stării lui T, apoi se inserează **commit/abort** în log rec + redo/undo aplicat asupra lui T; se inserează **end** în log.

## 2PC – Recuperarea datelor

- Dacă nu apare nici măcar un **prepare** în log pentru T, T se va termina unilateral
  - Acest site poate fi chiar coordonator!

## 2PC - Blocări

- Când coordonatorul pentru tranzacția T eșuează , subordonații care au votat **yes** nu se vor putea decide dacă să se termine cu *commit* sau *abort* până când coordonatorul își revine.
  - T este blocat.
  - Chiar dacă toți subordonații ar putea comunica între ei (prin extra info transmisă cu mesajul **prepare**) ei rămân blocați până când unul din ei transmite **no**.

## 2PC - Eșuarea rețelei / a unui site

- Dacă un site nu răspunde în timpul derulării protocolului de comitere pentru tranzacția T:
  - dacă site-ul curent este coordonator pentru T, T va trebui întrerupt.
  - dacă site-ul curent este un subordonat și nu a transmis încă *yes*, T va trebui întrerupt.
  - dacă site-ul curent un subordonat și a transmis *yes*, este blocat până când coordonatorul răspunde.

## 2PC - Observații

- Mesajul **done** e folosit pentru a informa coordonatorul că poate “ignora” o tranzacție; tranzacția T rămâne în tabela de tranzacții până aceasta recepționează toate mesajele **done**.
- Dacă coordonatorul eșuează după trimiterea mesajului **prepare** și înainte de scrierea în log a instrucțiunilor **commit/abort**, la revenire tranzacția se va termina fără succes.
- Dacă o sub-tranzacție nu modifică BD, faptul că ea se comite sau nu este *irelevant*.

## 2PC cu eșuare dedusă

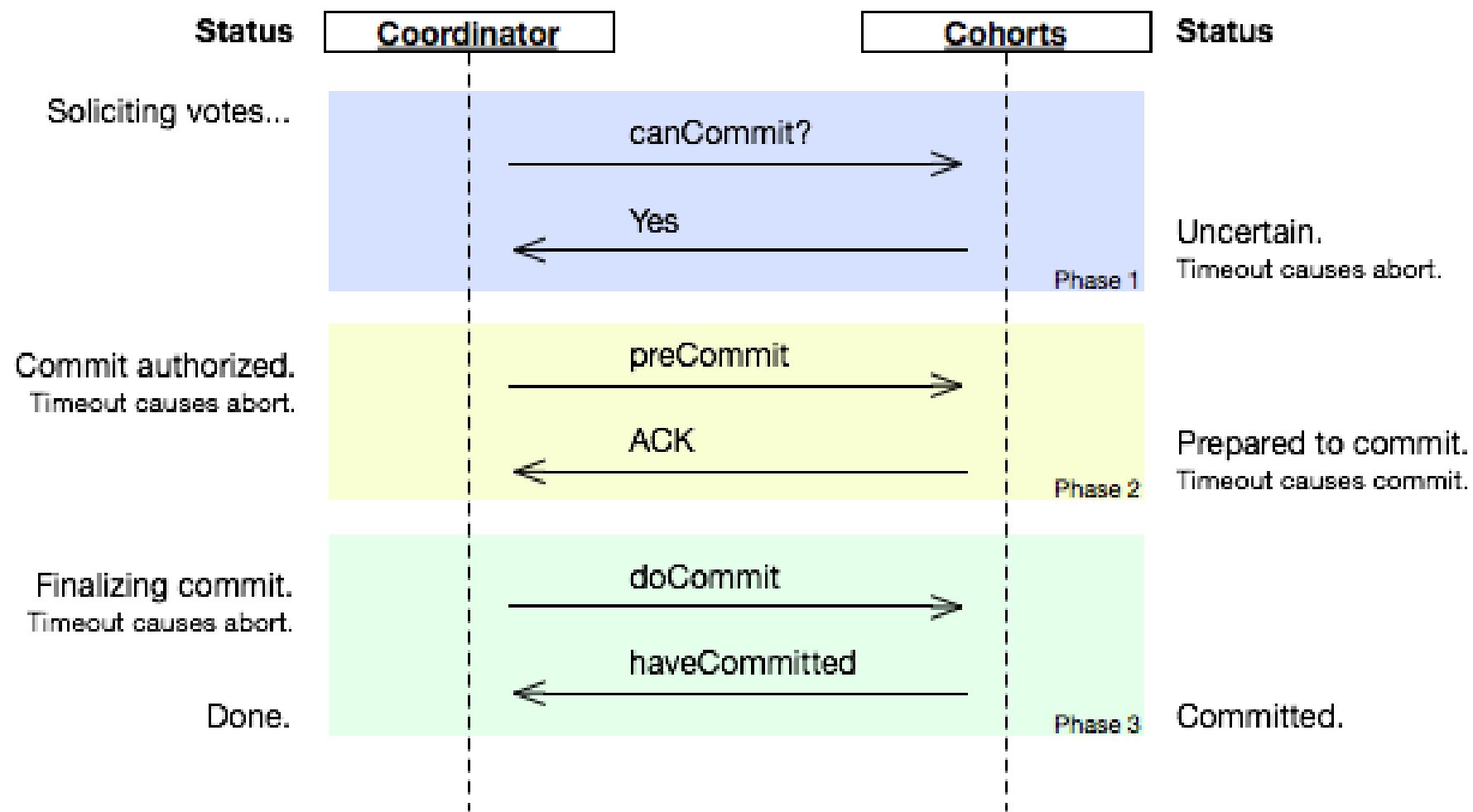
- Atunci când coordonatorul întrerupe tranzacția T, reface contextul de dinaintea executiei lui T și o elimină imediat din tabela de tranzacții.
  - Mesajele **done** nu se mai așteaptă; avem “eșec dedus” dacă transacția nu se află în tabela de tranzacții. Intrarea **abort** din log nu conține în acest caz numele subordonaților.

# 2PC cu eșuare dedusă

- Subordinații nu transmit **done** la eșec
- Dacă sub-tranzacțiile nu modifică BD, acestea răspund la **prepare** cu **reader** în loc de **yes/no**.
- Coordonatorul va ignora tranzacțiile "*reader*".
- Dacă toate sub-transacțiile sunt "*reader*" a doua fază nu este necesară.



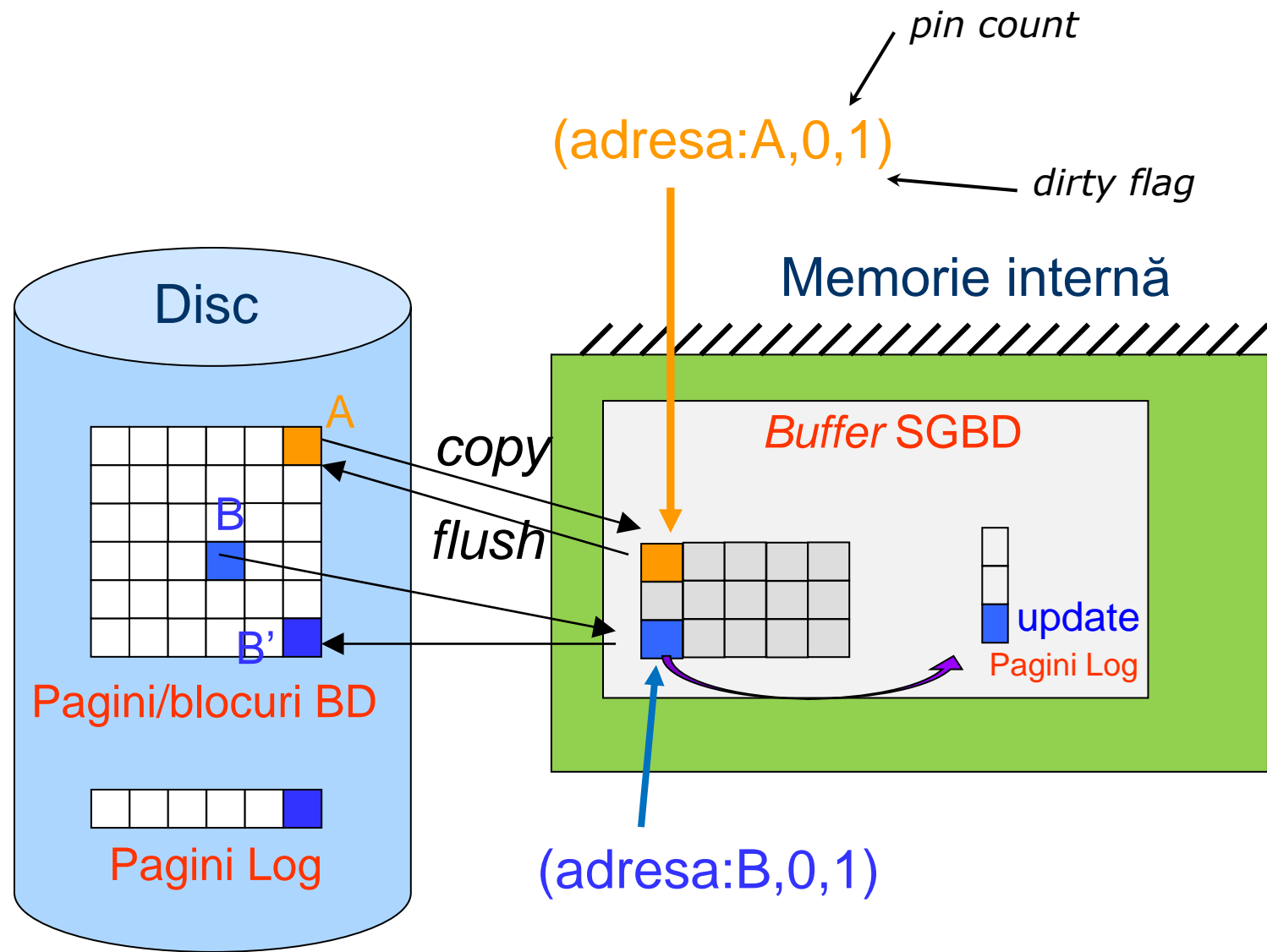
# Protocol de comitere în trei faze (3PC)

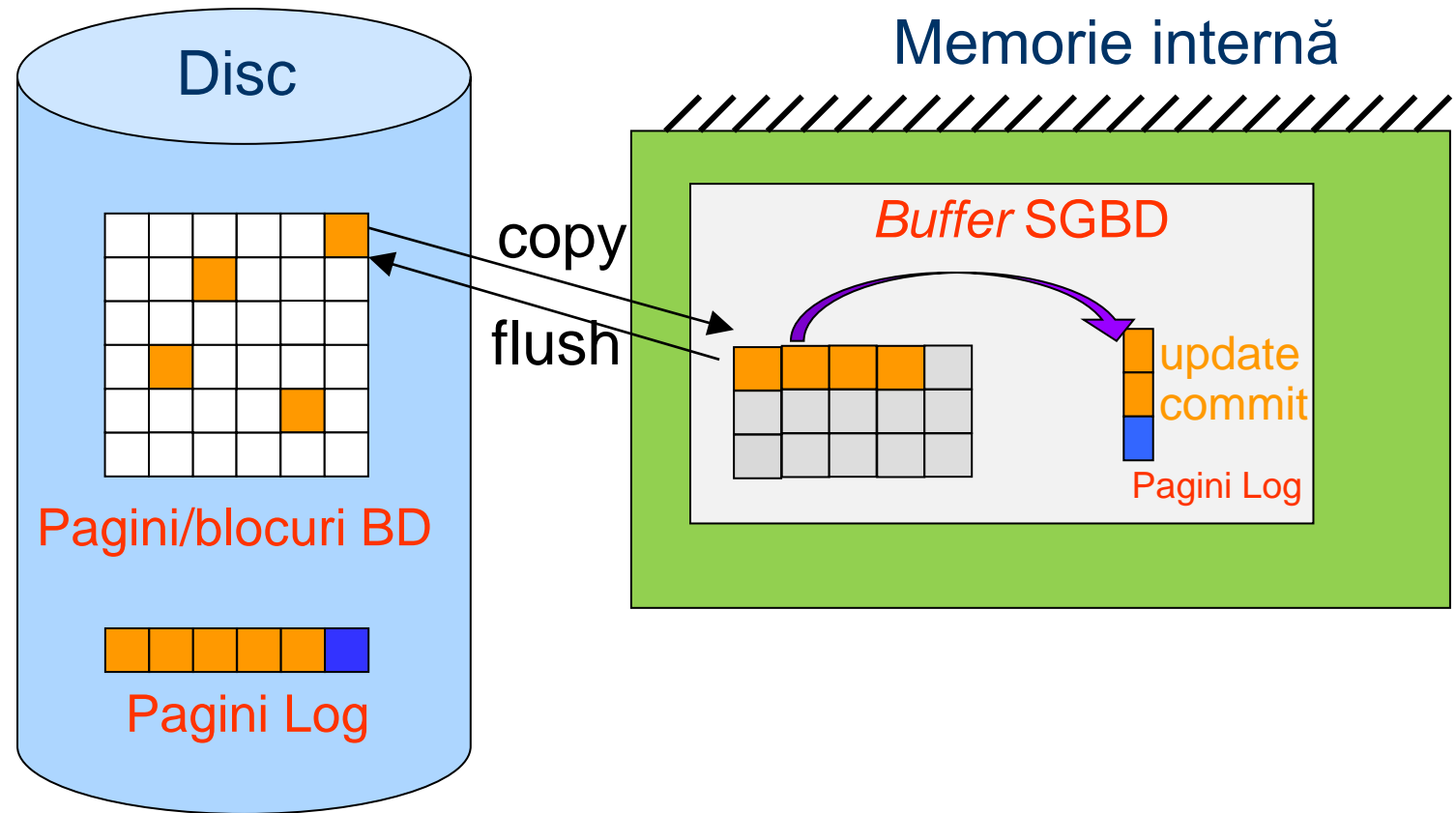


# Recuperarea datelor într-un context nedistribuit

# Actualizarea datelor

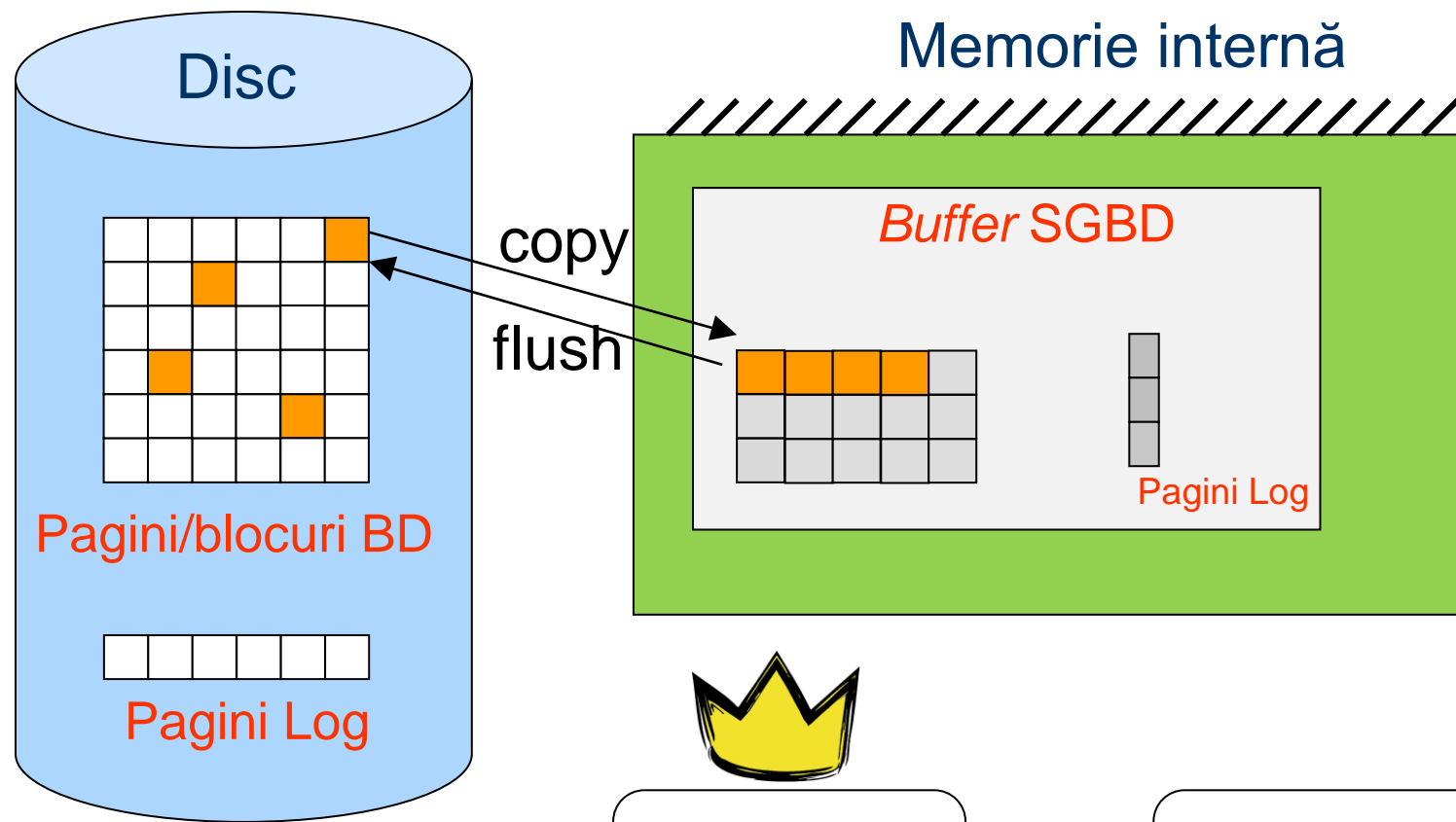
- **Actualizare imediată:** De îndată ce s-a realizat o modificare în *buffer*, este actualizat și corespondenta paginii de date de pe disc.
- **Actualizare amânată:** Toate datele modificate în *buffer* sunt actualizate pe disc după ce execuția unei tranzacții sau a unui număr fix de tranzacții este finalizată.
- **Actualizare "*in-place*" :** Versiunea originală a paginii ce conține datele pe disc este suprascrisă de corespondenta sa din *buffer*.
- **Actualizare "*shadow*":** Pagina de date din buffer nu se copiază peste corespondenta sa originală de pe disc, ci peste o copie a acesteia memorată la o adresă diferită.





### Protocol **Write-Ahead Logging** (WAL):

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**.



Buffer  
Manager

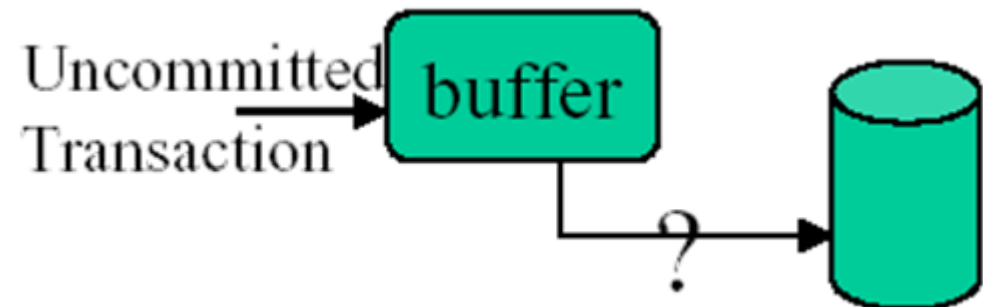
Performanță

vs

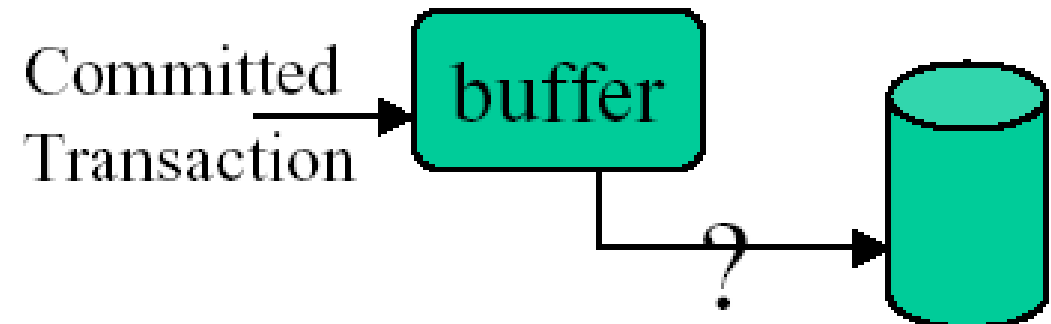
Recovery  
Manager

Atomicitate & Durabilitate

- Poate decide *Buffer Manager*-ul salvarea anumitor pagini (modificate de o tranzacție) din *buffer* pe disc fără a aștepta instrucțiuni specifice de la *Recovery Manager*?
  - Decizie *steal / no-steal*
  - *No-steal* înseamnă că RM păstrează referința către paginile modificate din *buffer*



- Poate *Recovery Manager* “forța” *Buffer Manager* să salveze anumite pagini din *buffer* pe disc la finalul executării unei tranzacții?
- Decizie *force / no-force*





- Se forțează salvarea pe disc a fiecărei modificări?
  - Timpuri mari de răspuns.
  - Garantează durabilitatea.
  - Garantează atomicitatea.

		No Steal	Steal
Force	Force	Trivial	
	No Force		Ideal

- Se permite salvarea unor pagini de memorie modificate de tranzacții ce nu s-au comis?
  - Dacă nu, concurență redusă, anumite tranzacții fiind blocate.
  - Dacă da, cum se poate garanta atomicitatea?

	No Steal	Steal
Force	Trivial	
No Force		Ideal

## ■ Steal / No-force

- BM poate salva modificări intermediare ale tranzacțiilor.  
RM salvează doar un *commit*

## ■ Steal / force

- BM poate salva modificări intermediare ale tranzacțiilor.  
RM salvează toate modificările (*flush*) înainte de *commit*

## ■ No-steal / no-force

- Nici una din paginile modificate nu se salvează decât la *commit*. RM salvează un *commit* și elimină referințele către paginile modificate.

## ■ No-steal / force

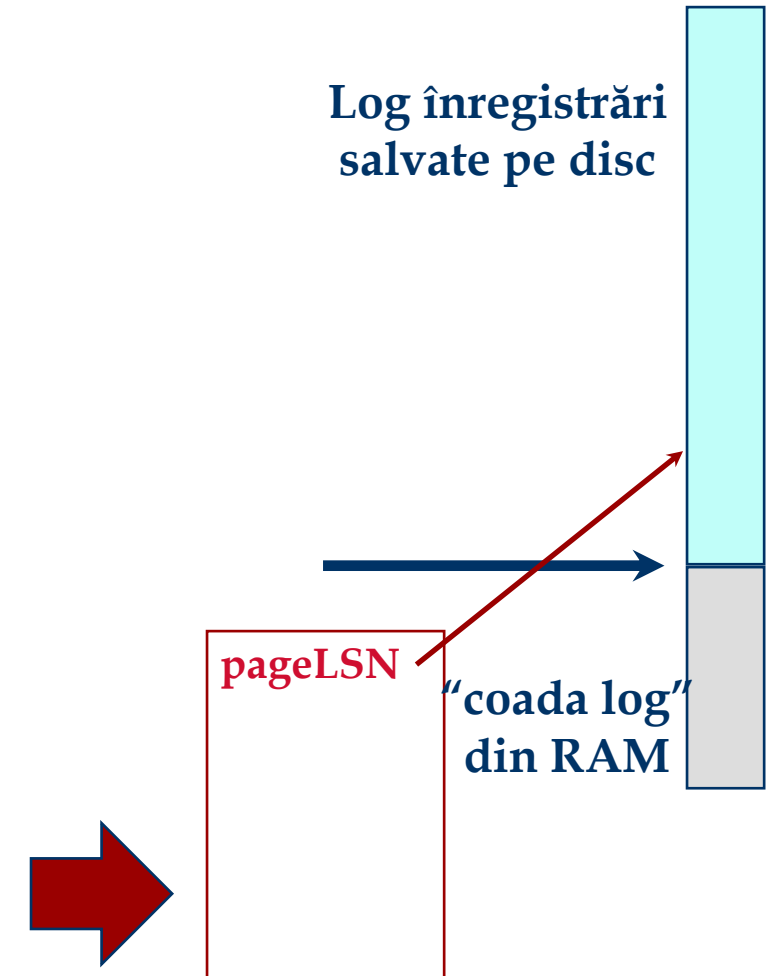
- Nici una din paginile modificate nu se salvează decât la *commit*. RM salvează toate modificările (*flush*) la *commit*

- **STEAL** (de ce garantarea *Atomicității* e dificilă)
  - *To steal frame F*: Pagina curentă memorată în F (să spunem P) este copiată pe disc; este posibil ca anumite tranzacții să blocheze anumite obiecte memorate în P.
    - Ce se întâmplă dacă tranzacția k, ce bloca anumite obiecte din P, eșuează?
    - Trebuie memorată vechea valoare a lui P (pentru a aplica **UNDO** modificărilor apărute în pagina P).
  
- **NO FORCE** (de ce garantarea *Durabilității* e dificilă)
  - Ce se întâmplă dacă sistemul se blochează înainte ca o pagină modificată să fie copiată pe disc?
  - În momentul comiterii unei tranzacții este necesar să se scrie pe disc informația minimă pentru ca modificările tranzacției să poată fi reproduse.

# Contextul WAL



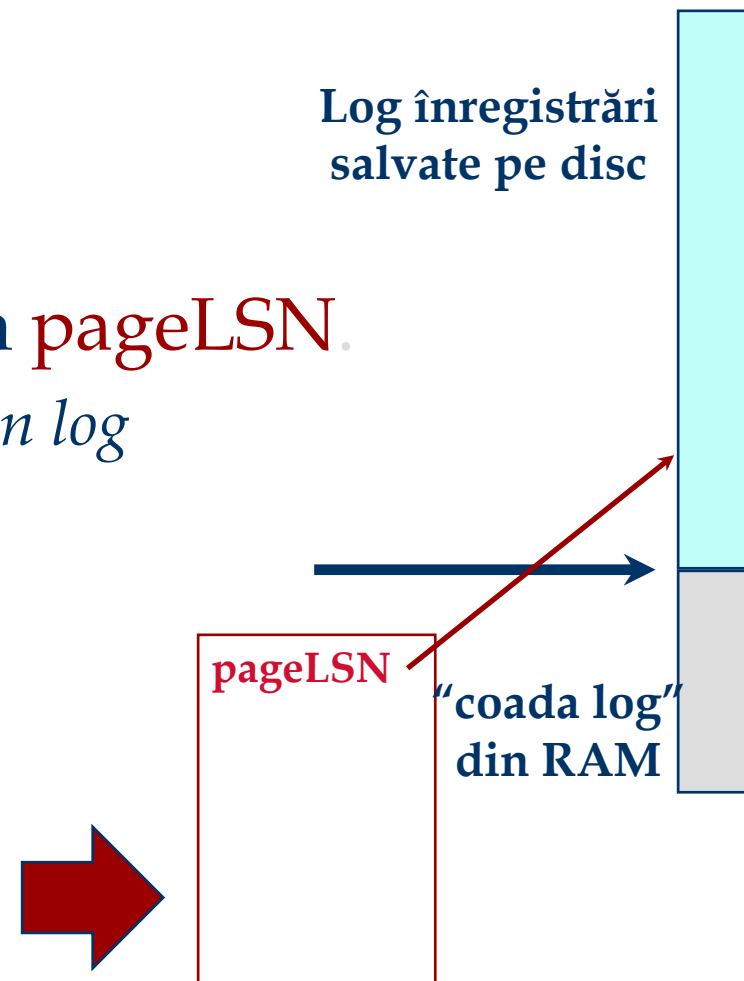
- Fiecare intrare din log are un **Log Sequence Number (LSN)**.
  - LSN crește incremental.



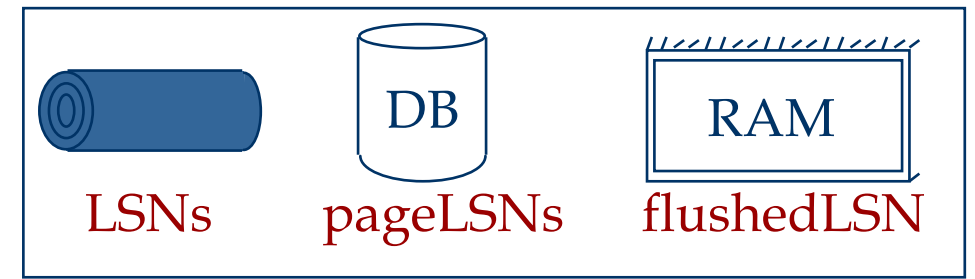
# Contextul WAL



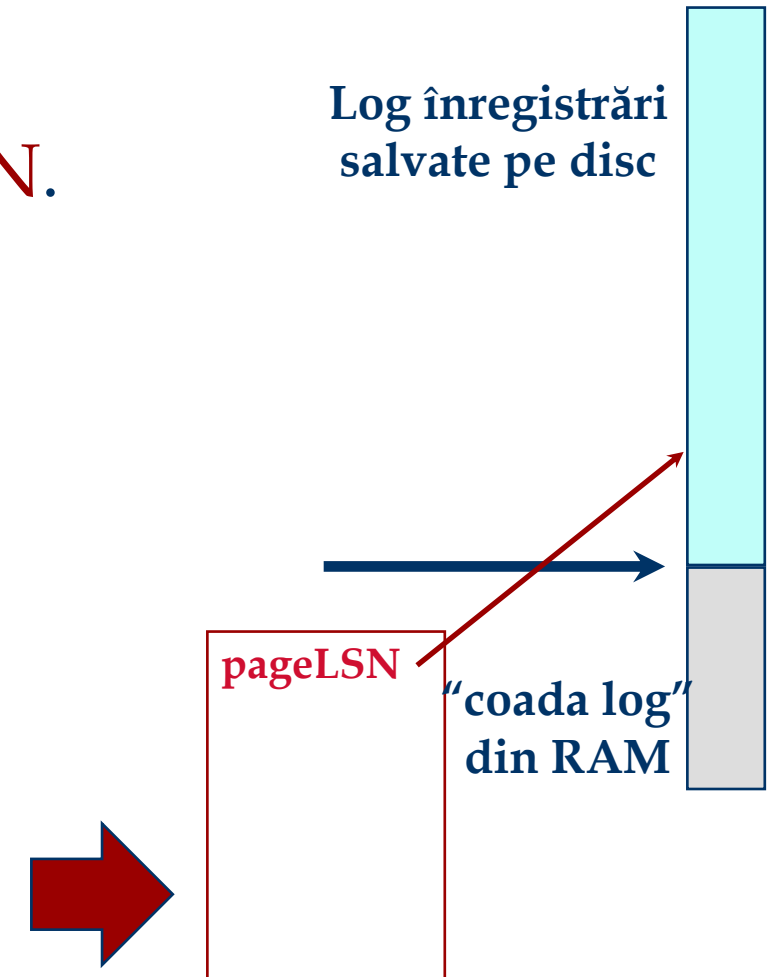
- Fiecare *pagină de date* conține un **pageLSN**.  
= LSN al celei mai recente *intrări din log* a unei modificări din pagină.



# Contextul WAL



- Sistemul mai reține **flushedLSN**.
  - LSN maxim până la care totul e salvat pe disc.
- WAL:
  - $\text{pageLSN} \leq \text{flushedLSN}$



# Intrări ale *log*-ului

## Câmpurile intrărilor :

	LSN
	prevLSN
	TransID
	type
Doar pentru modificări	pageID
	length
	offset
	before-image after-image

Tipuri posibile de intrări:

- **Update**
- **Commit**
- **Abort**
- **Checkpoint**
- **End** (semnifică terminarea unui *commit* sau *abort*)
- **Compensation Log Records (CLRs)**
  - pentru UNDO



# Compensation Log Record (CLR)

- Utilizat în faza de recuperare a datelor
- Este adăugat chiar înainte de anularea unei modificări marcate printr-o intrare în log
- Conține un câmp numit **undoNextLSN**
  - LSN-ul următoarei intrări de tip *update* ce trebuie anulată pentru o anumită tranzacție
  - Se inițializează cu *prevLSN* al intrării curente
- Indică ce acțiuni au fost deja anulate
- Previne anularea de mai multe ori a aceleiași acțiuni

# Alte construcții utilizate de RM

## ■ Tabela de tranzacții:

- O înregistrare pentru fiecare tranzacție activă.
- Conține **XID** (id tranzacție), **stare** (running / committed / aborted ) și **lastLSN**.

## ■ Tabela paginilor cu modificări (*Dirty Page Table*):

- O înregistrare pentru fiecare pagină cu modificări din *buffer*.
- Conține **recLSN** – LSN al primei intrări din log care a adus o modificare paginii.

# Execuția normală a unei tranzacții

## Context

- Secvență de citiri & modificări, urmate de commit sau abort
  - Vom presupune că scrierea unei pagini pe disc e atomică
- *Strict 2PL*.
- Abordare gestiune *buffer*: STEAL, NO-FORCE
- Write-Ahead Logging.

# Vedere de ansamblu



## Intrări Log

prevLSN  
XID  
type  
pageID  
length  
offset  
before-image  
after-image



## Pagini de date

fiecare  
cu un  
pageLSN

## Master record



## Tabelă tranzacții

lastLSN  
stare

## Tabelă pagini modif.

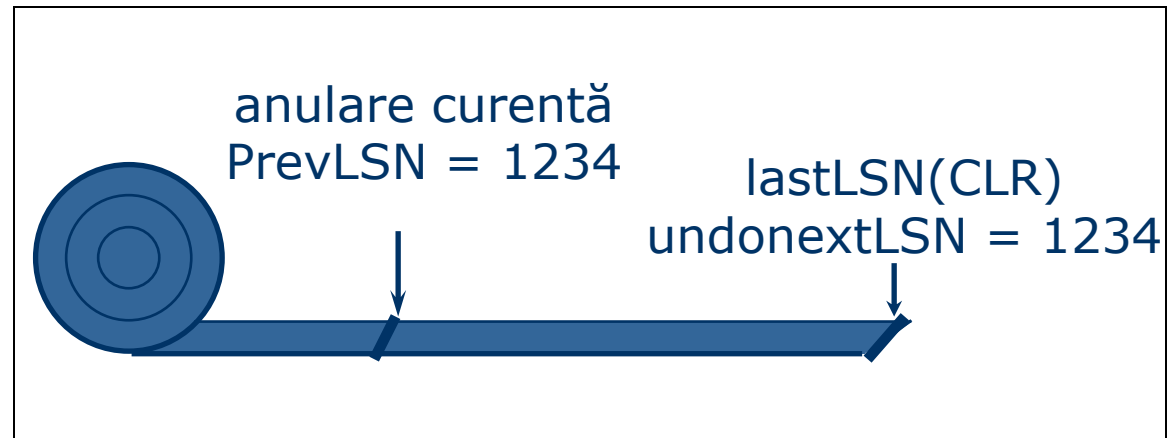
recLSN

**flushedLSN**

# Exemplu:

## Întreruperea simplă a unei tranzacții

- Se consideră întreruperea explicită a unei tranzacții.
- Se parcurge log-ul în ordine inversă, anulând modificările.
  - Se pornește de la **lastLSN** al tranzacției din tabela de tranzacții
  - Se parcurge lista de intrări ale log-ului urmând câmpul **prevLSN**
  - Înainte de anulare se adaugă o înregistrare *Abort* în log
    - utilă la recuperarea în cazul unei întreruperi în timpul operației de anulare a modificărilor!



- Obiectul a căreia modificare se anulează va fi blocat!
- Înainte de salvarea noii valori se adaugă un CLR:
  - Log-ul se actualizează și pe parcursul anulării!
  - Câmpul **undonextLSN** al CLR referă următoarea intrare din log pentru anulat (adică *prevLSN* al înregistrării anulate).
  - Intrările de tip CLR nu se anulează *niciodată*
- La finalul anulării tuturor modificărilor tranzacției se inserează o intrare **end** în log.

# Comiterea unei tranzacții

- Se inserează o intrare **commit** în log.
- Toate intrările de log corespunzătoare tranzacției se salvează pe disc (până la **lastLSN**).
  - Garantează că **flushedLSN**  $\geq$  **lastLSN**.
  - Inserările în log se fac secvențial, sincron pe disc
  - Există mai multe intrări de log per pagină.
- Se inserează o intrare **end** în log.

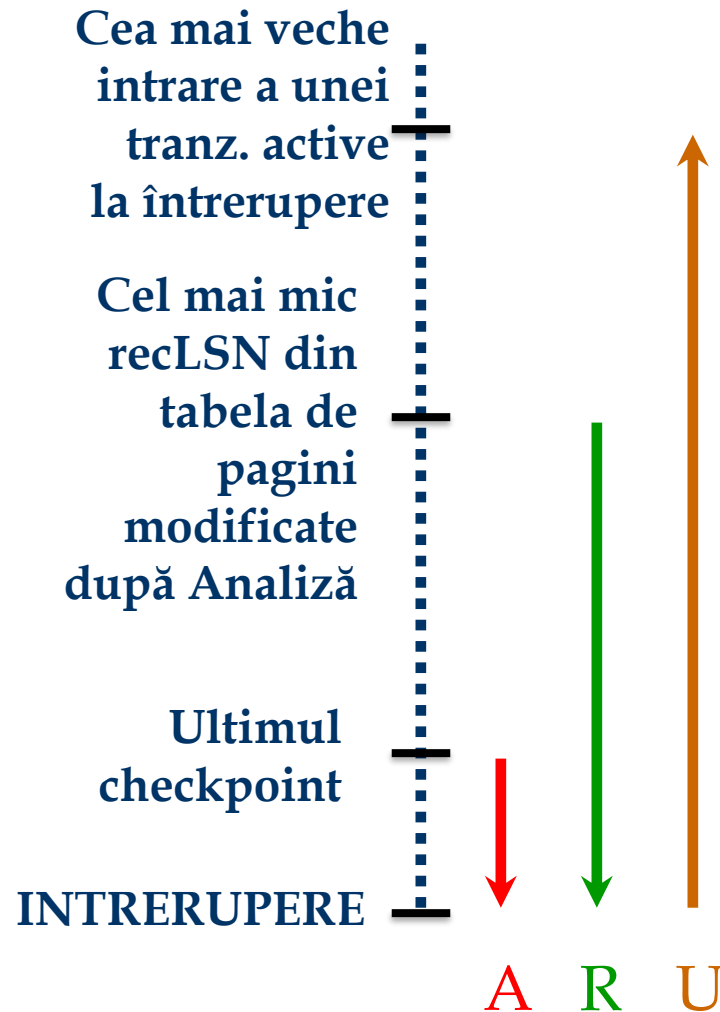
# Faze ale ARIES

(*Algorithm for Recovery and Isolation Exploiting Semantics*)

- Analiză: Se parcurge *log*-ul de la cel mai recent *checkpoint* spre final pentru identificarea tuturor tranzacțiilor active și a tuturor paginilor modificate existente în *buffer* la momentul întreruperii
- Redo: Reface toate modificările paginilor din *buffer*, corespunzătoare tranzacțiilor comise înainte de întrerupere, pentru a asigura că toate modificările s-au salvat pe disc.
- Undo: Modificările tuturor tranzacțiilor active în momentul întreruperii se anulează (folosind *valoarea anterioară* prezentă în intrare), mergând din spate în față.



# LOG



Se pornește de la ultimul *checkpoint* (din *master record*).

Trei faze:

- Aflarea tranzacțiilor active sau cele comise de la ultimul checkpoint (**Analiza**).
- Reexecutarea tuturor acțiunilor tranz. comise (repetare istoric **REDO**)
- Anularea efectelor tranzacțiilor eșuate (**UNDO**).

# Exemplu



## Tabelă Tranz

lastLSN

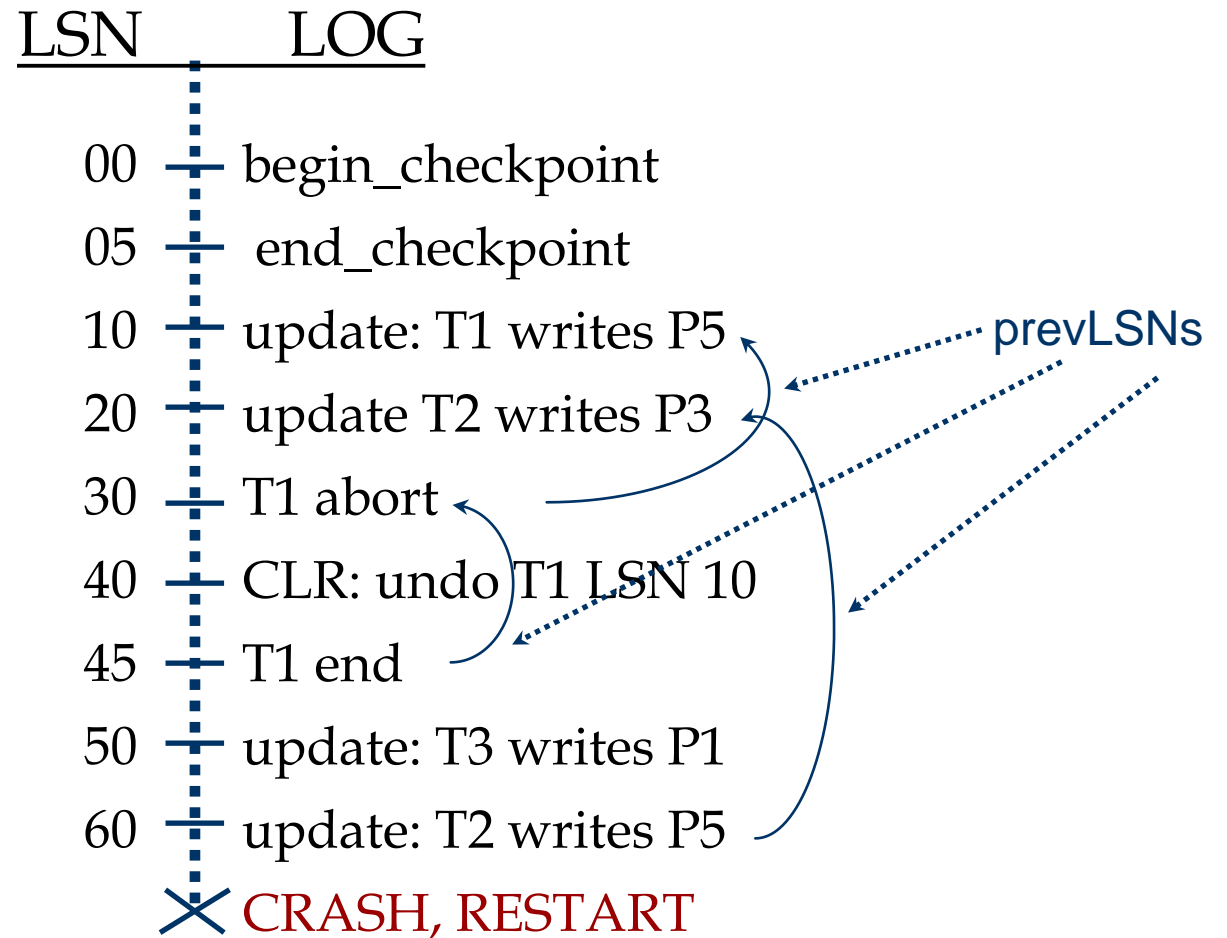
stare

## Tabelă Pagini Mod

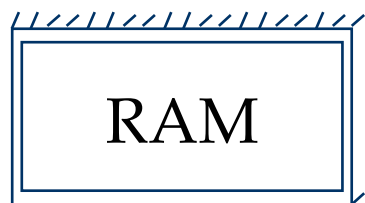
## recLSN

flushedLSN

## ToUndo



# Exemplu



Tabelă Tranz  
lastLSN  
stare  
Tabelă Pagini Mod  
recLSN  
flushedLSN

ToUndo

LSN	LOG
00,05	begin_checkpoint, end_checkpoint
10	update: T1 writes P5
20	update T2 writes P3
30	T1 abort
40,45	CLR: undo T1 LSN 10, <b>T1 end</b>
50	update: T3 writes P1
60	update: T2 writes P5
	✗ CRASH, RESTART
70	CLR: undo T2 LSN 60
80,85	CLR: undo T3 LSN 50, <b>T3 end</b>
	✗ CRASH, RESTART
90,95	CLR: undo T2 LSN 20, <b>T2 end</b>

undonextLSN

# Probleme suplimentare

- Pot să apară întreruperi în timpul recuperării bazei de date:
  - Se aplică *redo* și *undo* o singură dată unei înregistrări, sau
  - *Redo* și *undo* se construiesc ca acțiuni idempotente
- Limitarea duratei fazei de REDO:
  - Salvări asincrone de pagini.
- Limitarea duratei fazei de UNDO:
  - Evitarea tranzacțiilor ce durează mult.