PA152: Efektivní využívání DB

11. Zotavení z chyb

Vlastislav Dohnal

Integrita nebo správnost dat

- Chtěli bychom, aby data byla stále „bezesporná“ a „správná“.

Zaměstnanci

Jméno	Věk
Novák	52
Starý	3421
Svoboda	1

Integrita nebo správnost dat

- Integritní omezení
  - Hlavní nástroj integrity DB
  - Predikáty, které musí data splňovat
- Příklady:
  - *x* je primární klíč relace *R*
  - Funkční závislost: $x \rightarrow y$
  - Doména(x) = {červená, modrá, zelená}
  - *a* je platná hodnota atributu *x* v *R* (cizí klíč)
  - Žádný zaměstnanec by neměl mít záporný plat.

Integrita nebo správnost dat

- Konzistentní stav
  - splňuje všechna omezení
- Konzistentní DB
  - DB v konzistentním stavu

Limity integritních omezeních

- Nemusí zajistit „plnou správnost“
- Příklady: (Transakční omezení)
  - Žádný zaměstnanec by neměl mít více než dvojnásobek průměrného platu.
  - Pokud měníme plat,
    nový plat > původní plat
  - Pro smazání bankovního účtu musí platit
    balance = 0

Limity integritních omezeních

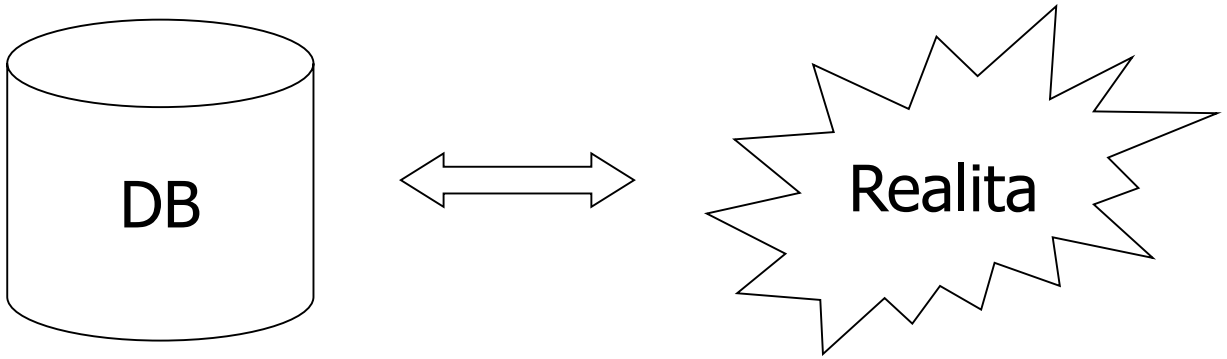
- Některé z nich mohou být „emulovány“ pomocí standardních omezení
  - Smazání účtu nahradit příznakem smazání

účet

č.ú.	…	zůstatek	smazaný

Limity integritních omezeních

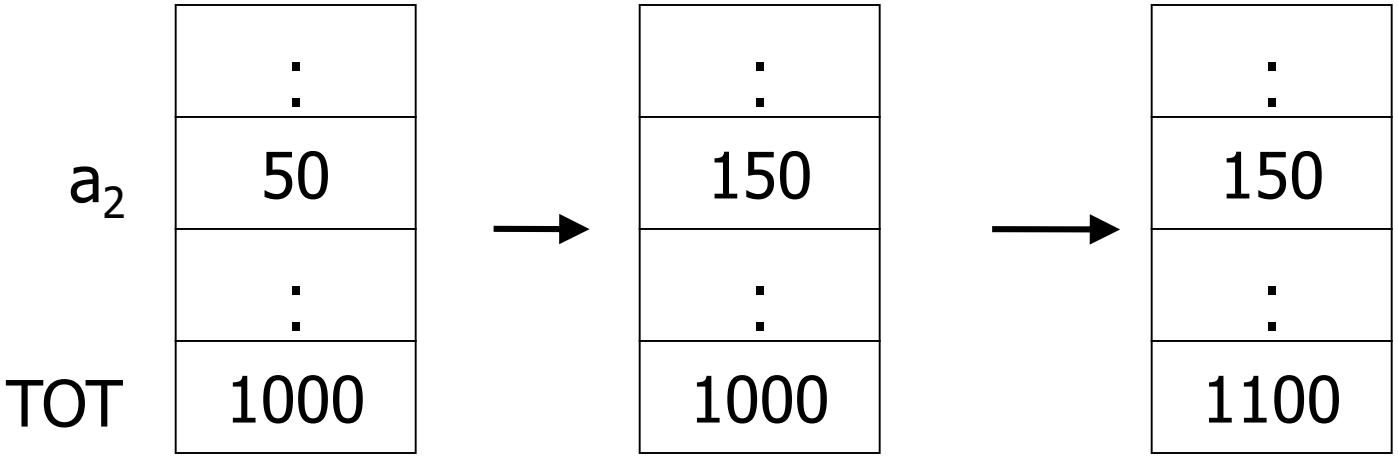
- Databáze by měla odpovídat skutečnému světu.



- Prováděj kontrolu existujících podmínek
  - přestože některé prvky „reality“ nelze definovat podmínkou omezení nebo DB není identická realitě
- Pozorování
  - DB nemůže být konzistentní stále.

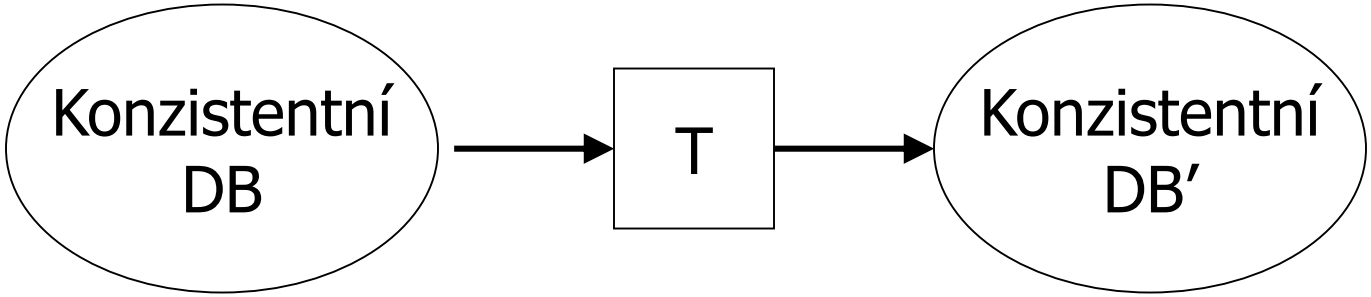
Příklad nekonzistentního stavu

- Příklad omezení
  - $a_1 + a_2 + \ldots a_n = TOT$
- Aplikace provádí vložení 100 Kč na účet a_2
  - $a_2 \leftarrow a_2 + 100$
  - $TOT \leftarrow TOT + 100$



Řešení průběžných nekonzistencí

- Transakce
  - Soubor akcí měnící dat, ale udržujících konzistenci



Transakční zpracování

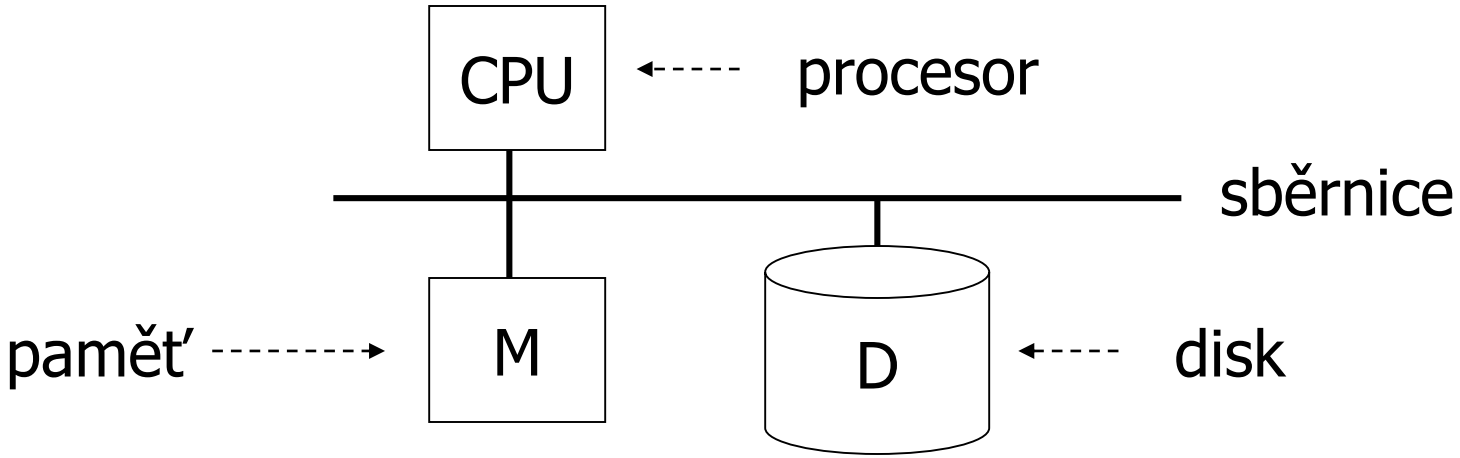
- Předpoklad
  - Pokud T začíná v konzistentním stavu a T běží samostatně
  - $\rightarrow$ T končí také v konzistentním stavu
- Správnost
  - Pokud přerušíme běžící transakci, DB zůstane konzistentní
  - Každá transakce vidí konzistentní DB

Porušení konzistence

- Příčiny
  - Chyba transakce
  - Chyba DB systému
  - Výpadek hardwaru
    - Např. výpadek úložiště poruší stav účtu na disku
  - Sdílení dat
    - Např. T1: přidej 10% platu programátorům
      T2: změň programátor → systémový analytik

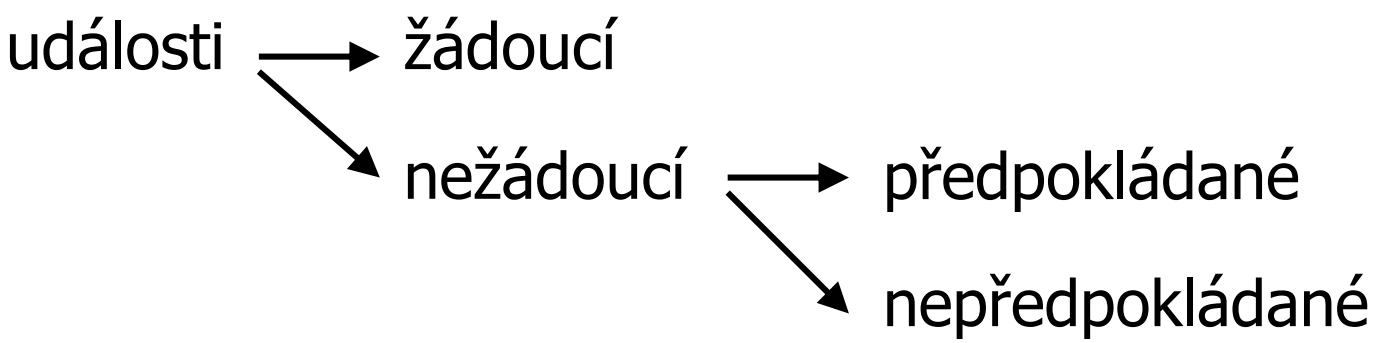
Zajištění konzistence

- Model výpadků
  - Shromáždění možných rizik
  - Řešíme výpadky jednotlivých komponent



Zajištění konzistence

- Model výpadků
  - Kategorizace událostí (rizik)



Zajištění konzistence

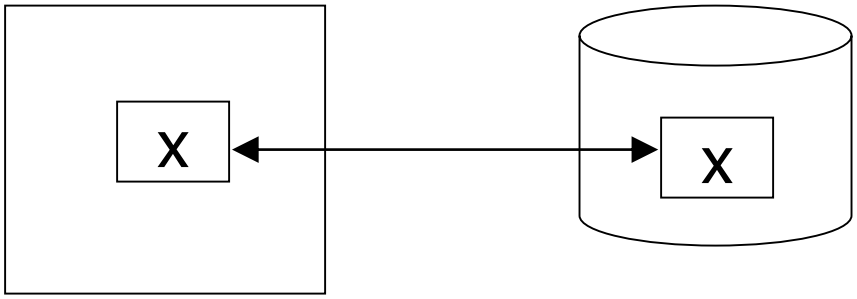
- Události
  - Žádoucí
    - Viz manuál DB systému ☺
  - Nežádoucí očekávané
    - Ztráta obsahu paměti
    - Zastavení procesoru, reset procesoru
    - Násilné vypnutí počítače
  - Nežádoucí neočekávané (vše ostatní)
    - Ztráta dat na disku
    - Chyba paměti bez zastavení procesoru
    - Živelné pohromy, …

Model výpadků

- Přístup k omezení rizik
  - Přidání kontrol na nejnižší úrovni
  - Redundance zvýší pravděpodobnost zachování podmínek
- Např.
  - Stabilní uložení dat (replikace disku, RAID)
  - Lepší paměť (kontrola paritní, ECC)
  - Kontrola CPU

Model výpadků

- Soustřeďujeme se na organizaci paměti



- Klíčový problém
  - Nedokončené transakce
  - Příklad

Omezení: A=B

Transakce T1: $A \leftarrow A \cdot 2$

$B \leftarrow B \cdot 2$

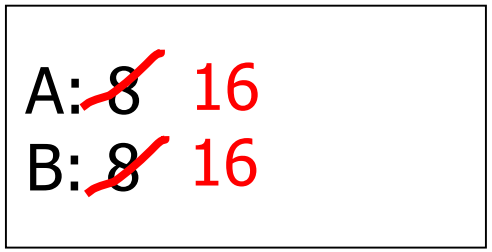
Transakce

- Základní operace
  - Input (x): blok obsahující x → paměť
  - Read (x,t): a. *Input(x)*, pokud je potřeba
    b. t ← hodnota x v bloku
  - Write (x,t): a. *Input(x)*, pokud je potřeba
    b. hodnota x v bloku ← t
  - Output (x): blok obsahující x → disk

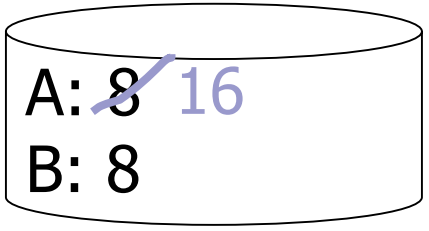
Příklad transakce T1

T1:
Read (A,t);
t ← t · 2;
Write (A,t);
Read (B,t);
t ← t · 2;
Write (B,t);
Output (A);
Output (B);

výpadek!



paměť



disk

Transakce

- Atomičnost
  - Řešení problému nedokončených transakcí
  - Provedení všech akcí transakce nebo vůbec žádné

- Jak implementovat?
  - Logování provedených změn
    - Tj. vytvoření žurnálu (souboru se záznamy o změnách)

Žurnálování

- Běh transakce produkuje záznamy o změnách do žurnálu
  - Začátek, konec, uložení, aktualizace, …
- Využití
  - Výpadek systému → znovu provedení/zrušení změn podle žurnálu
  - Obnova dat z archivu → znovu provedení změn podle žurnálu

Žurnálování

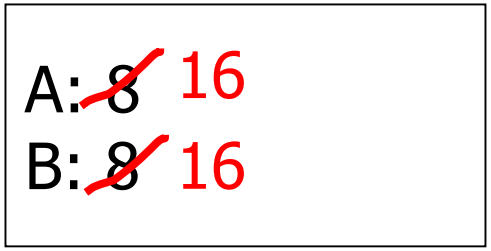
- Při obnově po výpadku systému
  - Některé transakce se provedou znovu
    - REDO
  - Některé transakce se zruší
    - UNDO

Undo logging

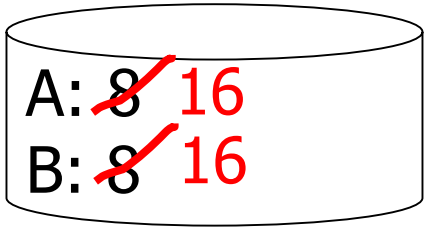
- „Zrušení podle žurnálu“
- Vlastnost
  - Změny prováděné transakcí jsou *ihned ukládány* na disk
- Pokud není jistota (100%) uložení změn dokončené transakce
  - Pak se změny podle žurnálu odstraní
    - Tj. obnovení předchozího stavu DB
  - → Transakce nikdy nebyla spuštěna

Undo logging: Transakce T1

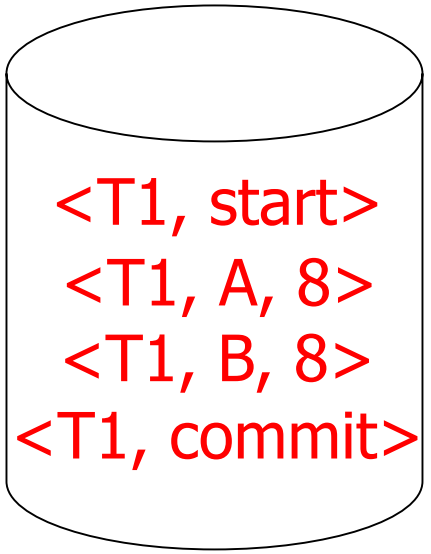
T1:
Read (A,t);
t ← t · 2;
Write (A,t);
Read (B,t);
t ← t · 2;
Write (B,t);
Output (A);
Output (B);



paměť



disk

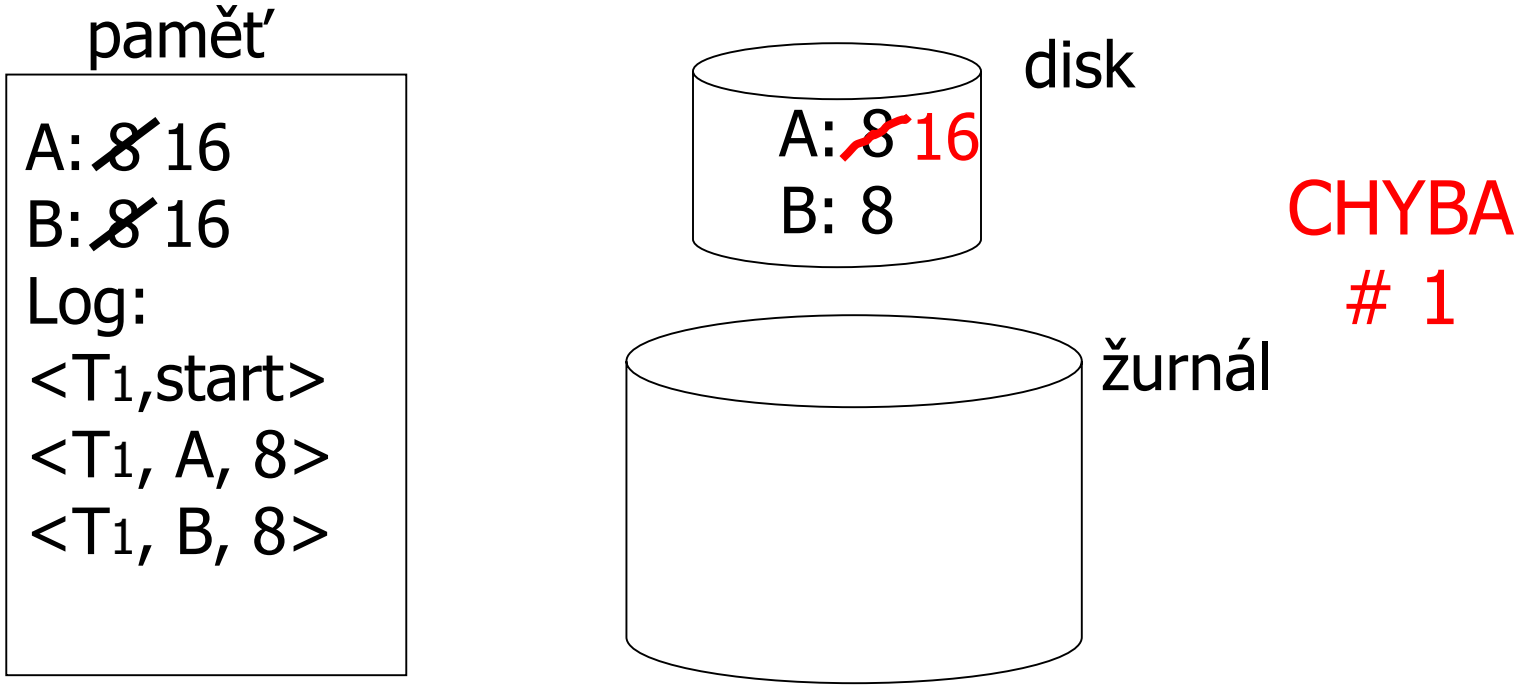


žurnál

Pozn: vyžadujeme platnost A=B

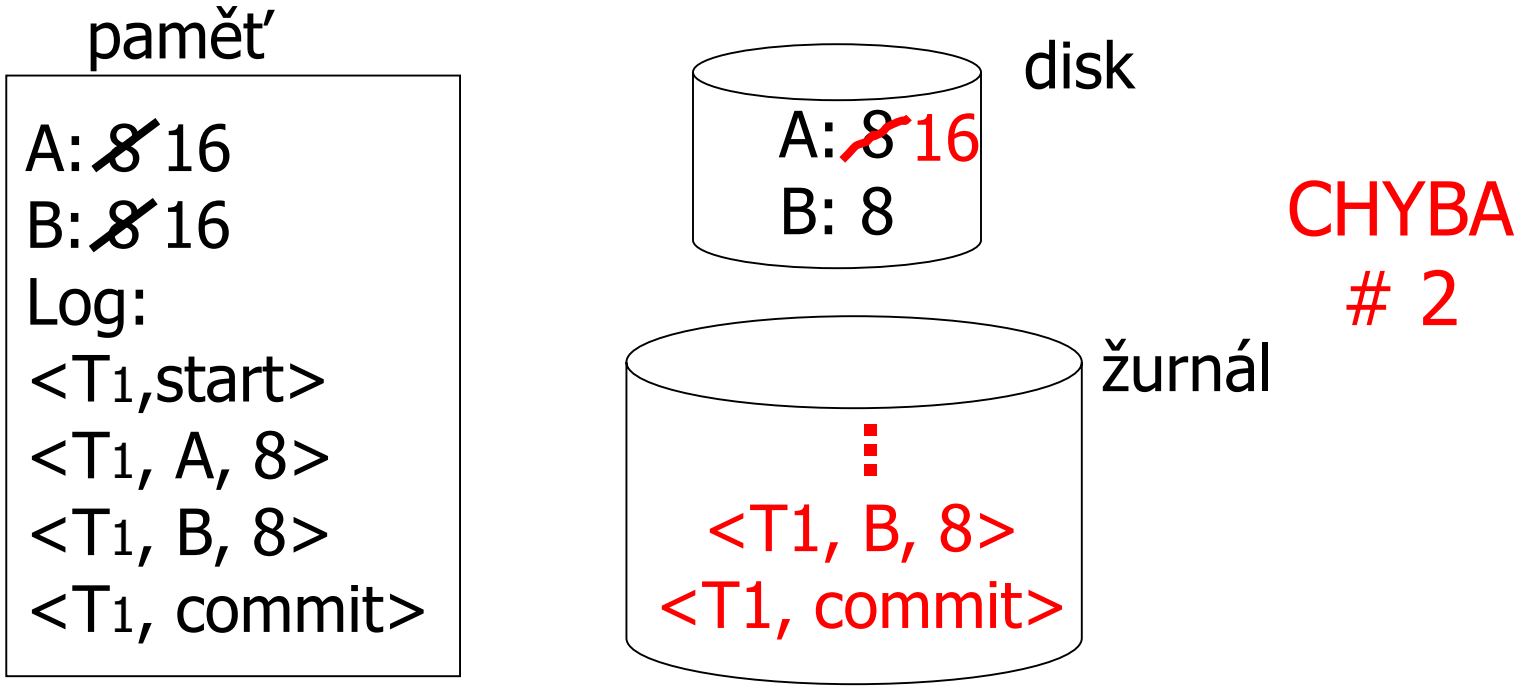
Undo logging

- Komplikace
  - Žurnálování používá správce vyrovnávací paměti → tvořen v paměti, později uložen.



Undo logging

- Komplikace
  - Žurnálování používá správce vyrovnávací paměti → tvořen v paměti, později uložen.



Undo logging

- Pravidla
  1. Pro každou akci **write**(X,t) vytvoř v žurnálu záznam obsahující starou hodnotu *X*
  2. Před změnou *X* na disku (**output**(X)) musí být na disku záznamy žurnálu týkající se *X*
     - Tzv. *write ahead logging* (WAL)
  3. Před vytvořením záznamu <T, commit> v žurnálu musí být všechny zápisy transakce uloženy na disku.

Undo logging – obnova po výpadku

- Pro každé T_i, které má <T_i, start> v žurnálu:
  - Pokud existuje <T_i, commit> nebo <T_i, abort>, nedělej nic
  - Jinak pro každé <T_i, X, v> v žurnálu:
    - write(X, v)
    - output(X)
    - zapiš <T_i, abort> do žurnálu

Je to správně?

Undo logging – obnova po výpadku

1. S = množina transakcí
   - Které mají <T_i, start> v žurnálu,
   - ale nemají <T_i, commit> ani <T_i, abort>
2. Pro každé <T_i, X, v> v žurnálu
   - Proveď v obráceném pořadí (nejnovější $\rightarrow$ nejstarší)
   - Pokud $T_i \in S$, pak write(X, v) a output (X)
3. Pro každé $T_i \in S$
   - zapiš <T_i, abort> do žurnálu
     - po úspěšném zapsání všech output(X) na disk

Undo logging – obnova po výpadku

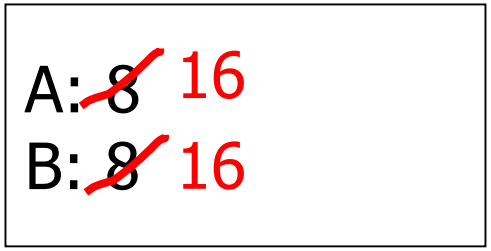
- Výpadek během obnovy
  - Nevadí
    - UNDO lze provést i opakovaně (je idempotentní)
    - Provádí se pouze pro nedokončené transakce

Redo logging

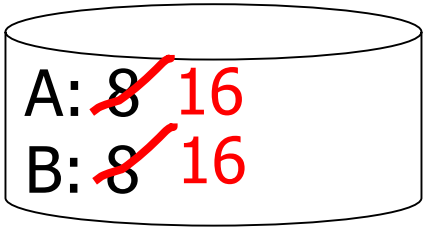
- „Znovu provedení podle žurnálu“
- Vlastnost
  - Změny provedené transakcí jsou *ukládány později*
    - Tj. při potvrzení (commit)
    - Ušetření zápisů na disk
  - Při obnově jsou ignorovány nedokončené transakce
  - Vyžaduje uložení žurnálu před uložením změn v DB.
    - Žurnálují se nové hodnoty

Redo logging: Transakce T1

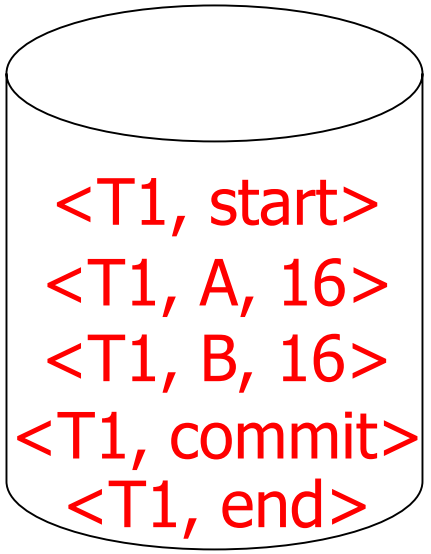
T1: Read (A,t);
$t \leftarrow t \cdot 2$;
Write (A,t);
Read (B,t);
$t \leftarrow t \cdot 2$;
Write (B,t);
Output (A);
Output (B);



paměť



disk



žurnál

Redo logging

- Pravidla
  1. Pro každou akci **write**(X,t) vytvoř v žurnálu záznam obsahující novou hodnotu *X*
  2. Před změnou *X* na disku (v DB) (**output**(X)) musí být na disku všechny záznamy žurnálu (včetně commit) pro transakci měnící *X*
     1. Ulož záznamy žurnálu na disk
     2. Ulož změněná data na disk
     3. Zapiš end do žurnálu

Redo logging – obnova po výpadku

- Pro každé T_i, které má <T_i, commit> v žurnálu, proveď:
  - Pro každé <T_i, X, v> v žurnálu proveď:
    - write(X, v)
    - output(X)

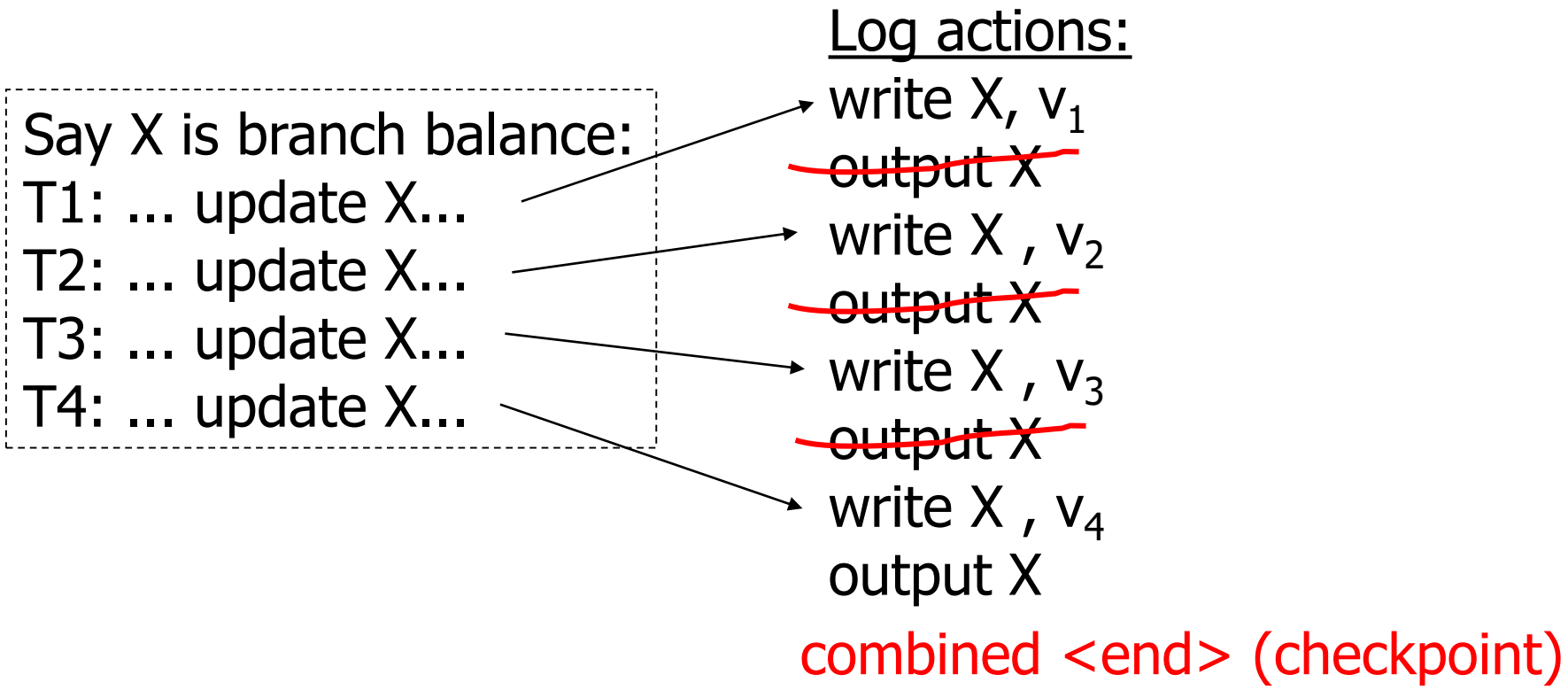
Je to správně?

Redo logging – obnova po výpadku

1. S = množina transakcí
   - Které mají <T_i, commit > v žurnálu,
   - ale nemají <T_i, end>
2. Pro každé <T_i, X, v> v žurnálu
   - Proveď v dopředném pořadí (nejstarší → nejnovější)
   - Pokud $T_i \in S$, pak write(X, v) a output (X)
3. Pro každé $T_i \in S$
   - zapiš <T_i, end> do žurnálu

Combining <Ti, end> Records

- Want to delay DB flushes for hot objects

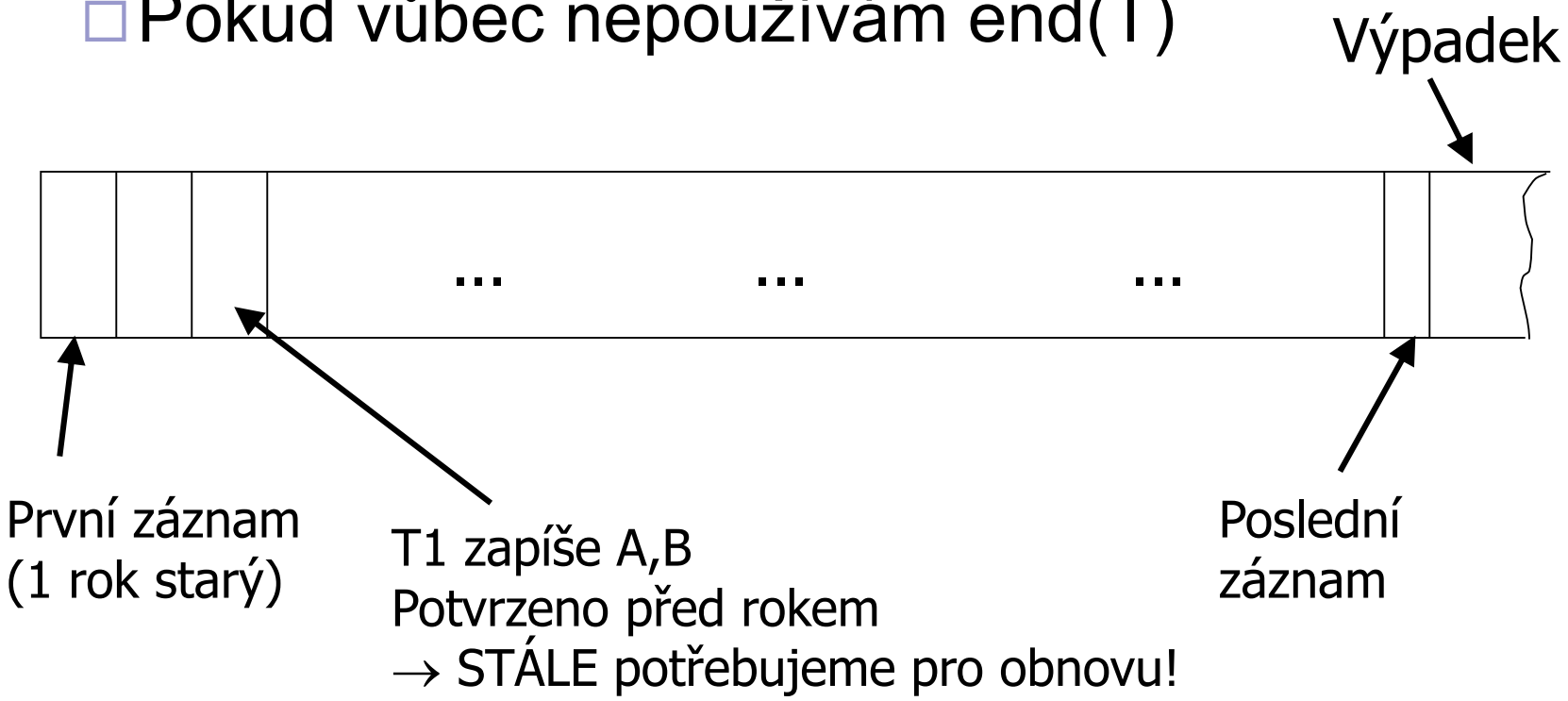


Redo logging – obnova po výpadku

- Ukládání změn output(X)
  - Pokud je více transakcí měnících X,
  - pak stačí provést output(X) pouze pro poslední záznam $<T_i, X, v>$ v žurnálu
  - Také záznam *end* lze zkombinovat pro více transakcí

Redo logging – obnova po výpadku

- Obnova je velice pomalá
  - Pokud vůbec nepoužívám end(T)

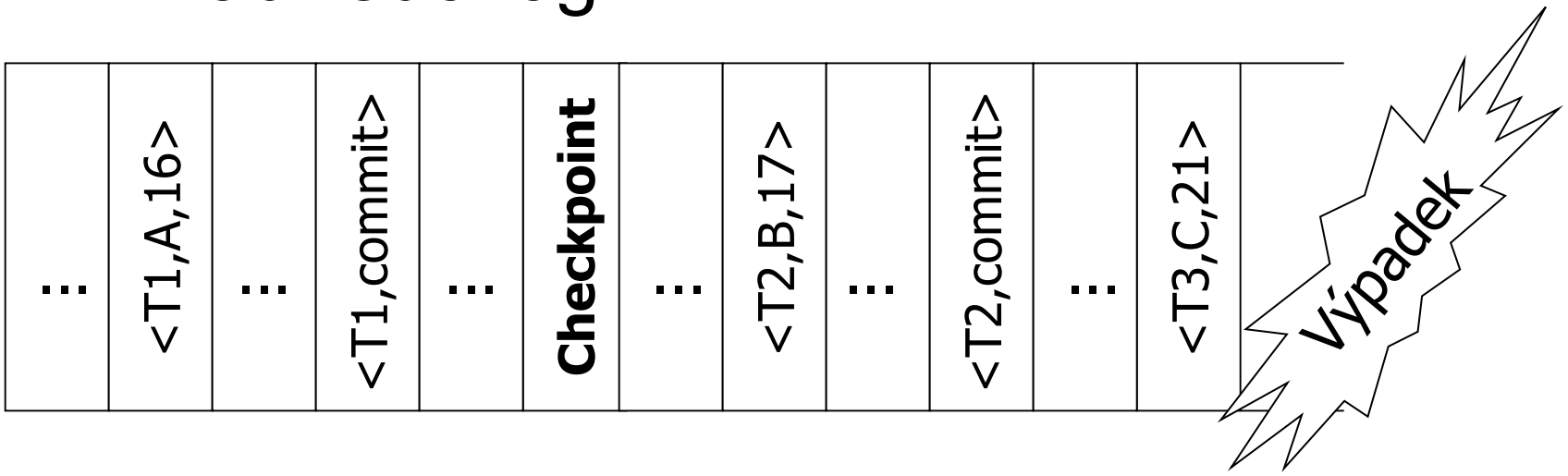


Žurnálování – obnova po výpadku

- Řešení pomalosti
  → kontrolní body (checkpoints)
- Implementace:
  1. Přestaň přijímat nové transakce
  2. Počkej na ukončení všech transakcí
  3. Ulož všechny záznamy žurnálu na disk
  4. Ulož všechny bloky na disk (DB)
  5. Zapiš záznam „checkpoint“ na disk do žurnálu
  6. Pokračuj ve zpracování transakcí

Žurnálování – obnova po výpadku

- Postup při obnově
  - Vyhledej poslední kontrolní bod
  - Od něj proveď obnovu
- Příklad redo log:



Žurnálování

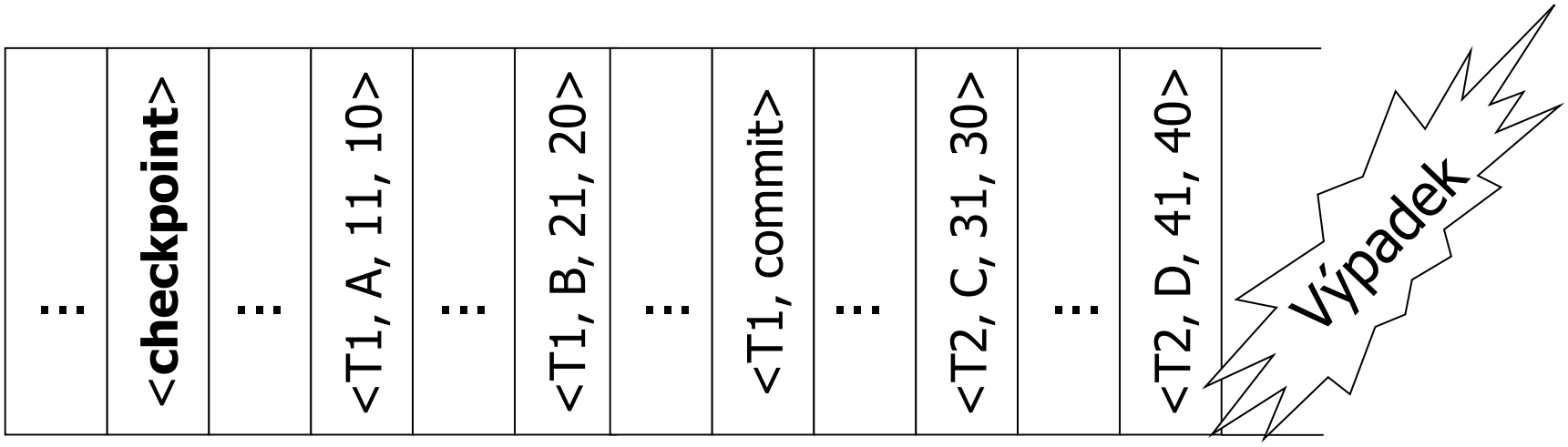
- Hlavní nevýhody
  - Undo logging
    - Ze zálohy DB nelze vytvořit aktuální stav DB
  - Redo logging
    - Všechny modifikované bloky musíme držet v paměti až do potvrzení (commit) transakce
  - Zápisy na disk jsou vynuceny pravidly žurnálu a ne přístupem k datům
- Řešení: Undo/Redo logging
  - Záznam v žurnálu obsahuje starou i novou hodnotu: <T_i, x, nová X, původní X>

Undo/Redo logging

- Pravidla
  - Hodnota *X* může být uložena před i po potvrzení T_i
  - Před zapsáním hodnoty *X* na disk, musí být na disk zapsán odpovídající záznam žurnálu (WAL)
  - Ulož žurnál při potvrzení transakce
- Obnova
  - Ukončené transakce zopakujeme (redo) od začátku
  - Nedokončené transakce vrátíme (undo) od konce

Undo/Redo logging – obnovení

- Příklad undo/redo log:

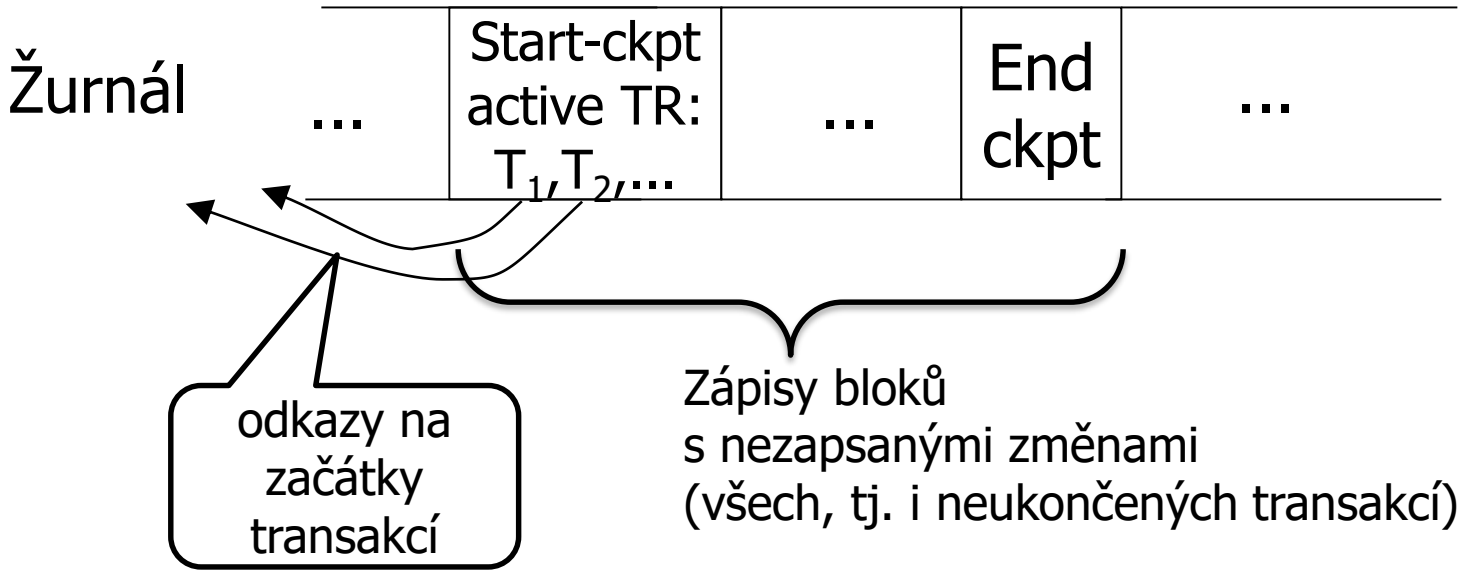


Kontrolní body

- Jednoduchý checkpoint
  - Během vytváření nesmí žádné transakce běžet
  - Významné snížení průchodnosti DB
- Řešení
  - Průběžné kontrolní body (Non-quiescent Checkpoint)
    - Evidence nedokončených transakcí
    - UNDO/REDO logging:
      - všechny změněné záznamy (bloky) jsou uloženy na disk

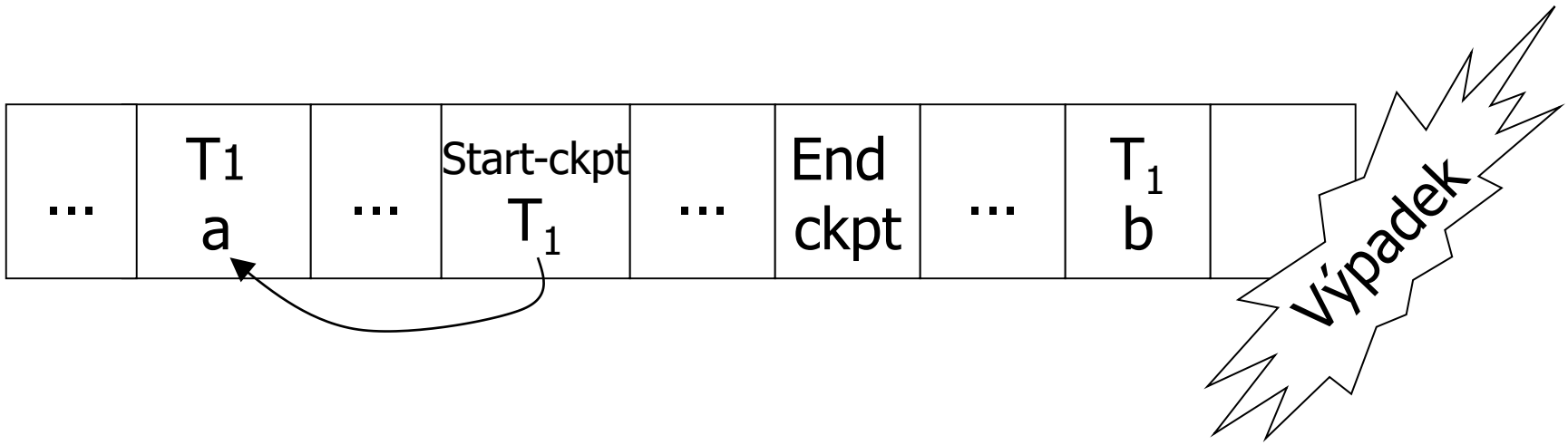
Průběžné kontrolní body

- Uloží se počátek a konec kontrolního bodu



Průběžné kontrolní body

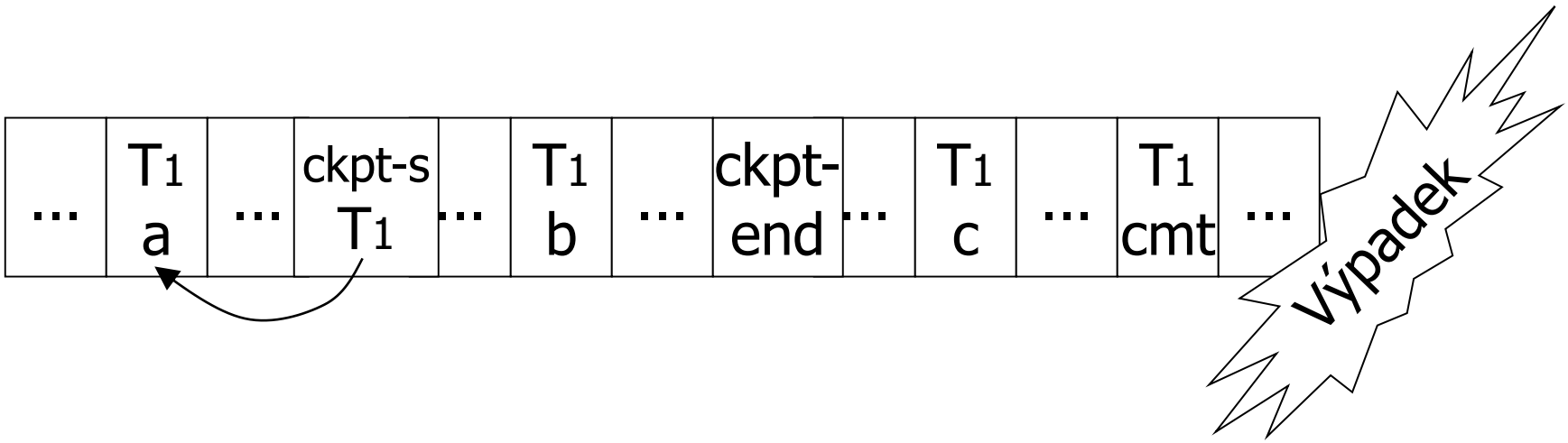
- Obnova 1



- T_1 nemá commit → Undo T_1 (undo b,a)

Průběžné kontrolní body

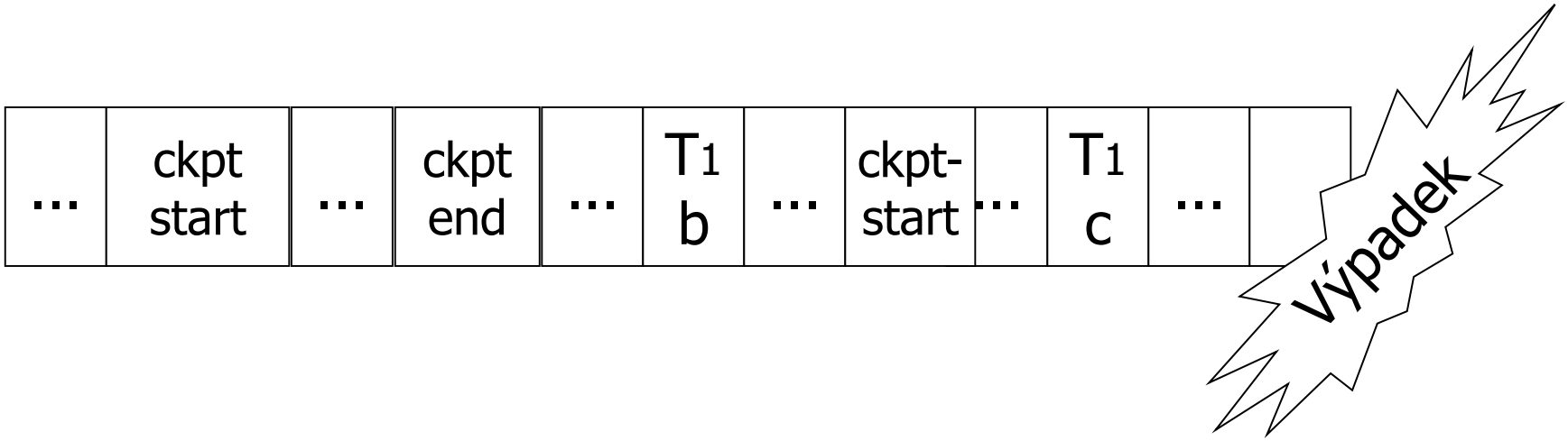
- Obnova 2



- T_1 má commit $\rightarrow$ Redo T_1 (redo b,c)

Průběžné kontrolní body

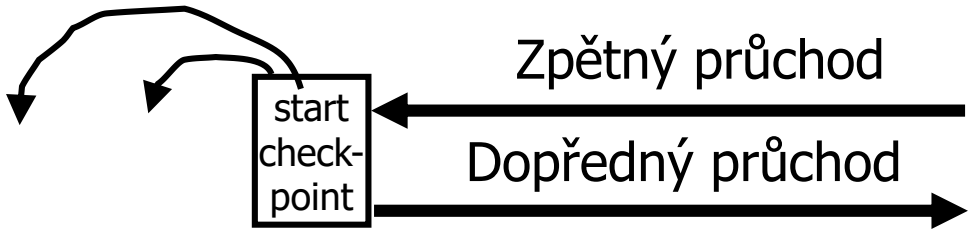
- Obnova 3



- Není dokončený kontrolní bod
  → najdi poslední úspěšný kontrolní bod
  - Proveď undo/redo transakcí

Proces obnovy z výpadku

- Zpětný průchod
  (konec žurnálu → začátek posledního ukončeného kontrolního bodu)
  1. Vytvoř množinu *S* potvrzených transakcí
  2. Vrať (undo) akce transakcí, které nejsou v *S*
- Pozn.: Vrácení nedokončených transakcí
  - Včetně nedokončených transakcí v kontrolním bodu
- Dopředný průchod
  (začátek posledního kontrolního bodu → konec žurnálu)
  - Opakuj akce transakcí v množině S (bez *end*)

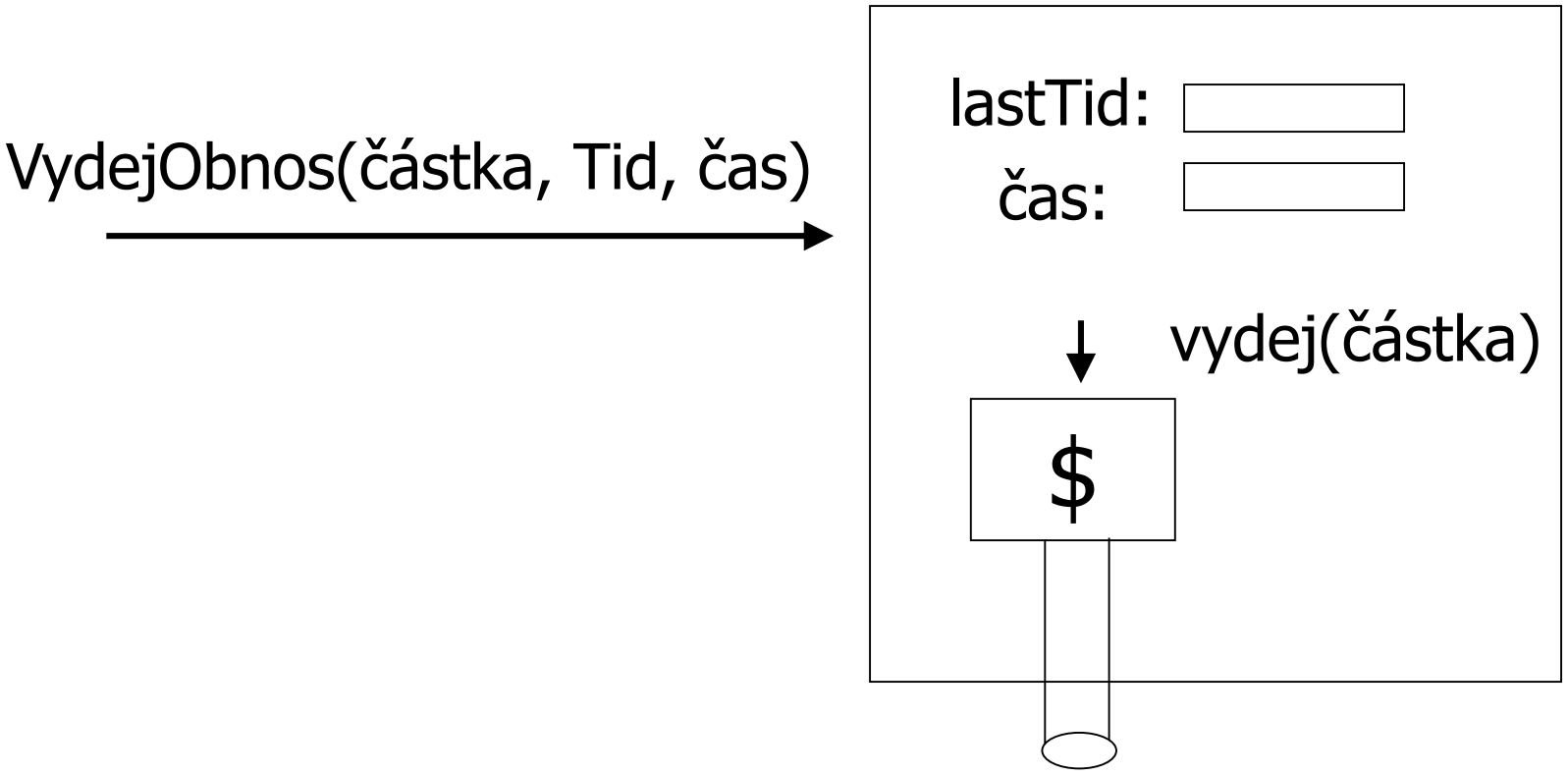


Reálný příklad transakce

- Vybírání hotovosti z bankomatu
  - Informace o účtech v DB banky
  - HW zařízení bankomatu
- Implementace
  - Transakce v databázi
  - Fyzický výdej peněz
- Postup
  - Proveď transakci, výdej peněz až po commit.
  - Příkaz k výdeji peněz by měla být idempotentní operace.

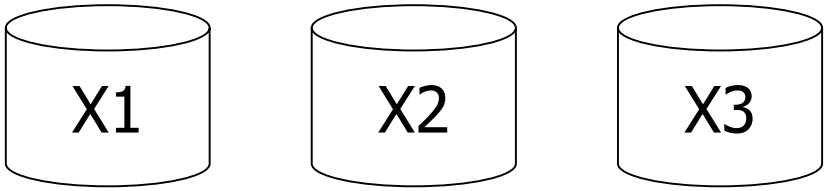
Reálný příklad transakce

- Po provedení transakce v DB je poslán „signál“ pro výdej



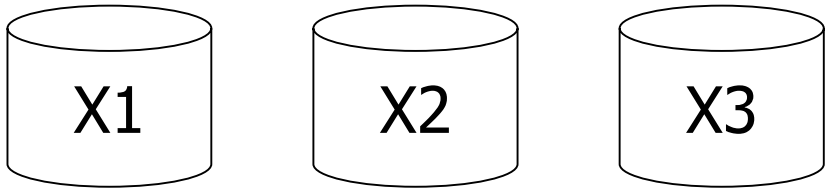
Výpadek úložného média

- RAID
- Kopie dat na nižší úrovni
  - Např.
    - 3 identické kopie
    - Output(X)
      → ulož do 3 úložišť
    - Input(X)
      → načti ze 3 úložišť a zvol správný výsledek (voting)



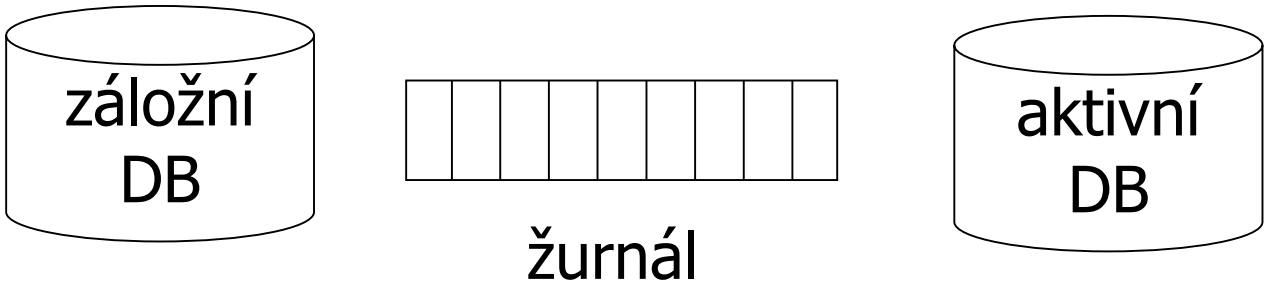
Výpadek úložného média

- Kopie dat na nižší úrovni
  - Jiný příklad
    - 3 identické kopie
    - Output(X)
      → ulož do 3 úložišť
    - Input(X)
      → načti z prvního (pokud je ok, pokračuj)
      → načti z druhého, …
    - Podmínka
      - Chybná data lze detekovat



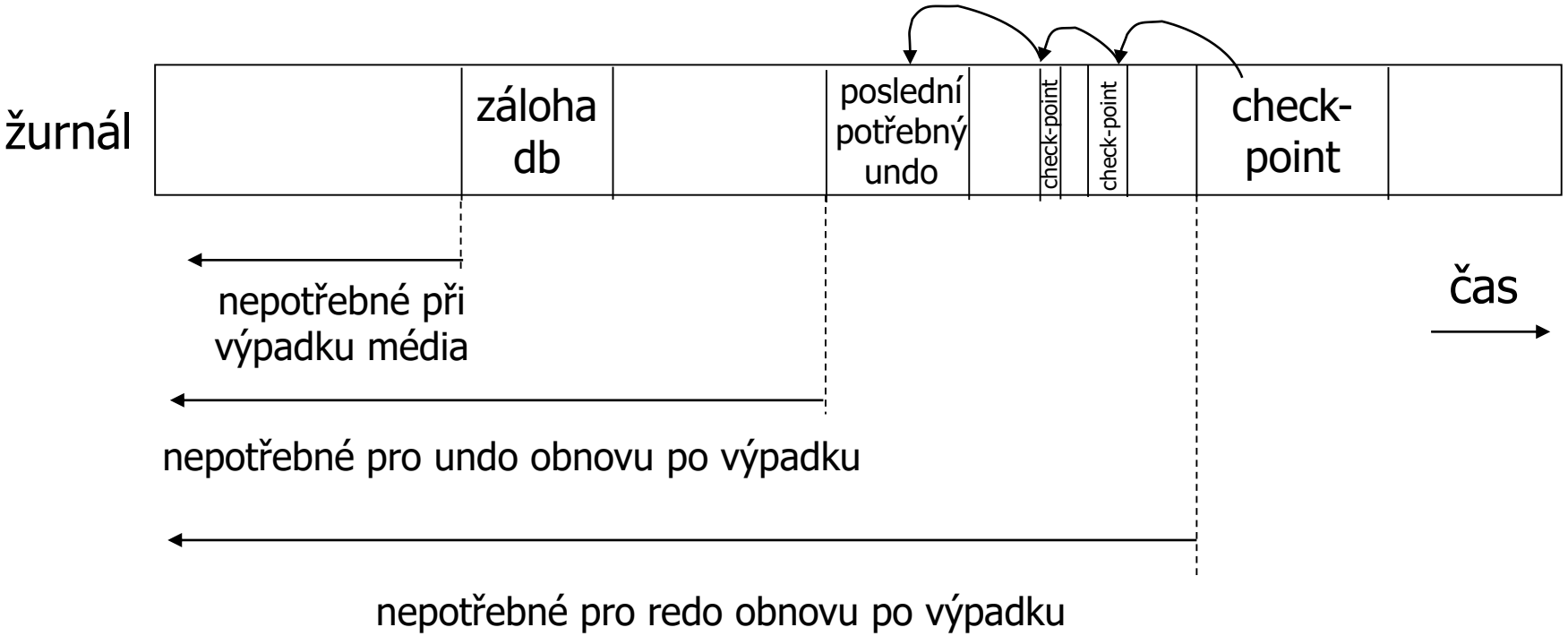
Výpadek úložného média

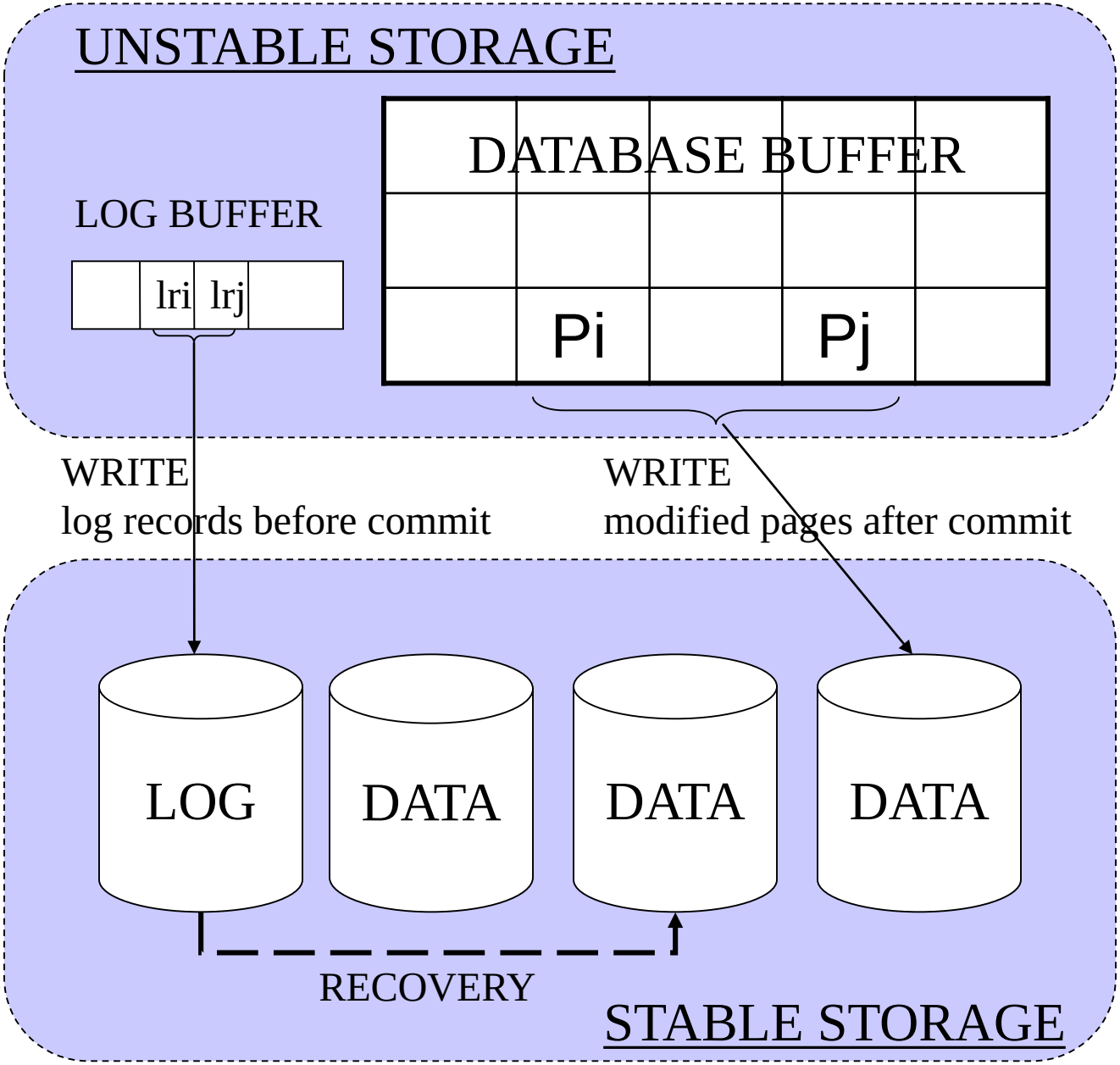
- Záloha DB
  - Obnovení ze zálohy (archivu)
  - Zpracování žurnálu
    - Zopakuj redo záznamy pro každou transakci nedokončenou v době provedení zálohy



Mazání žurnálu

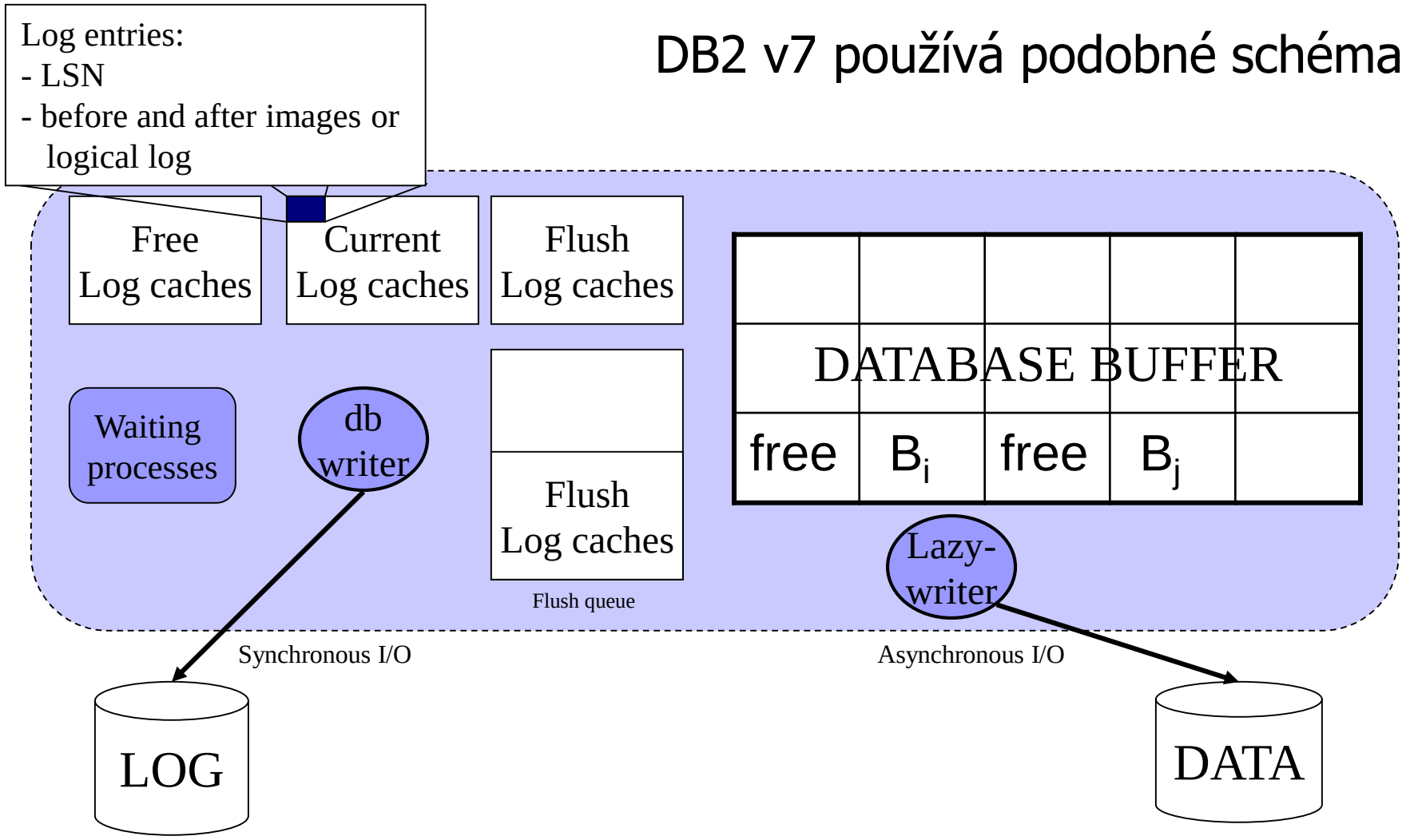
- Kdy lze žurnál zkrátit?
  - V případě UNDO/REDO logging



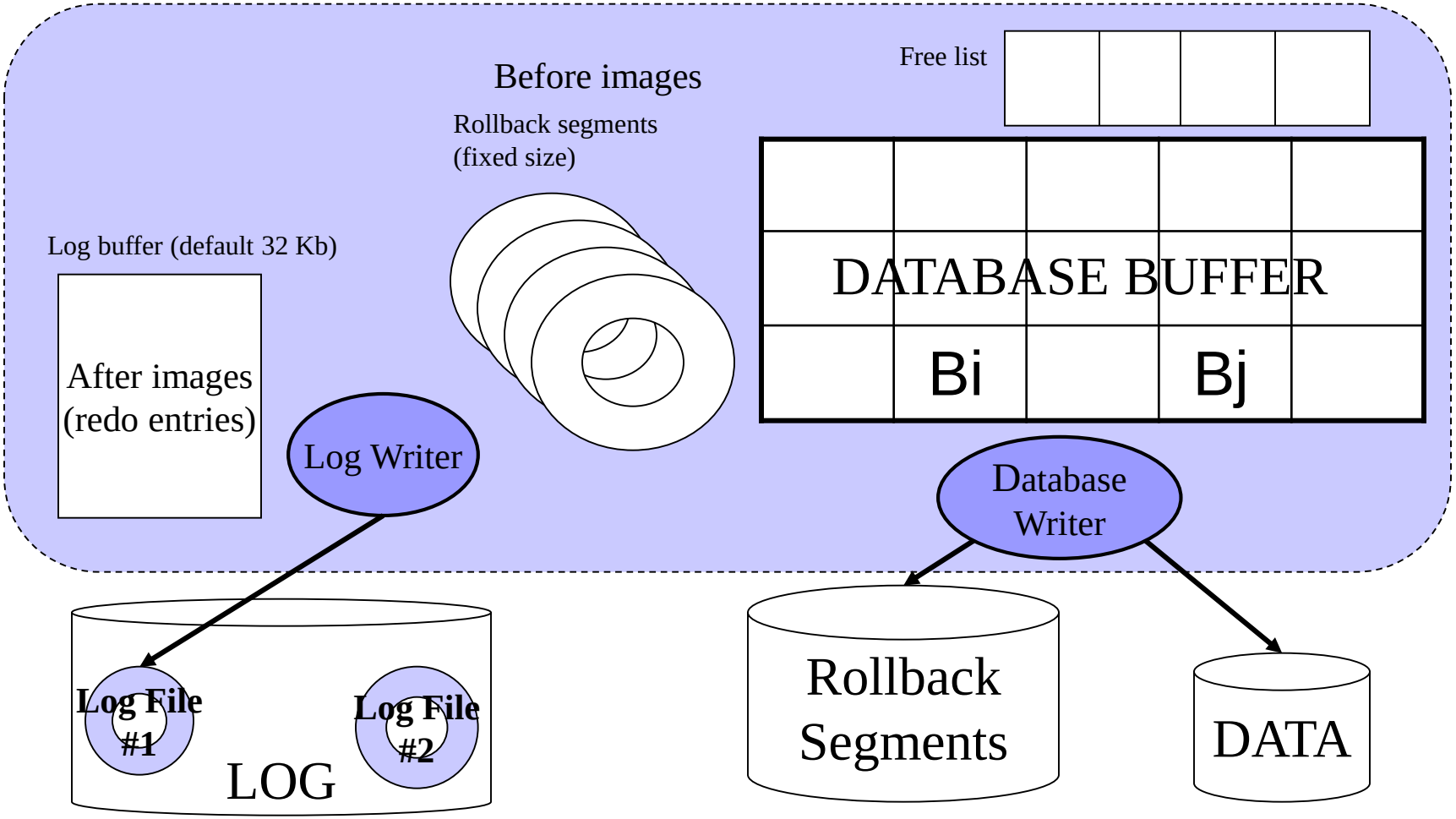

UNSTABLE STORAGE
DATABASE BUFFER
LOG BUFFER
lri
lrj
Pi
Pj
WRITE
log records before commit
WRITE
modified pages after commit
LOG
DATA
DATA
DATA
RECOVERY
STABLE STORAGE

Žurnálování v SQLServer 2000

DB2 v7 používá podobné schéma



Žurnálování v Oracle 8i

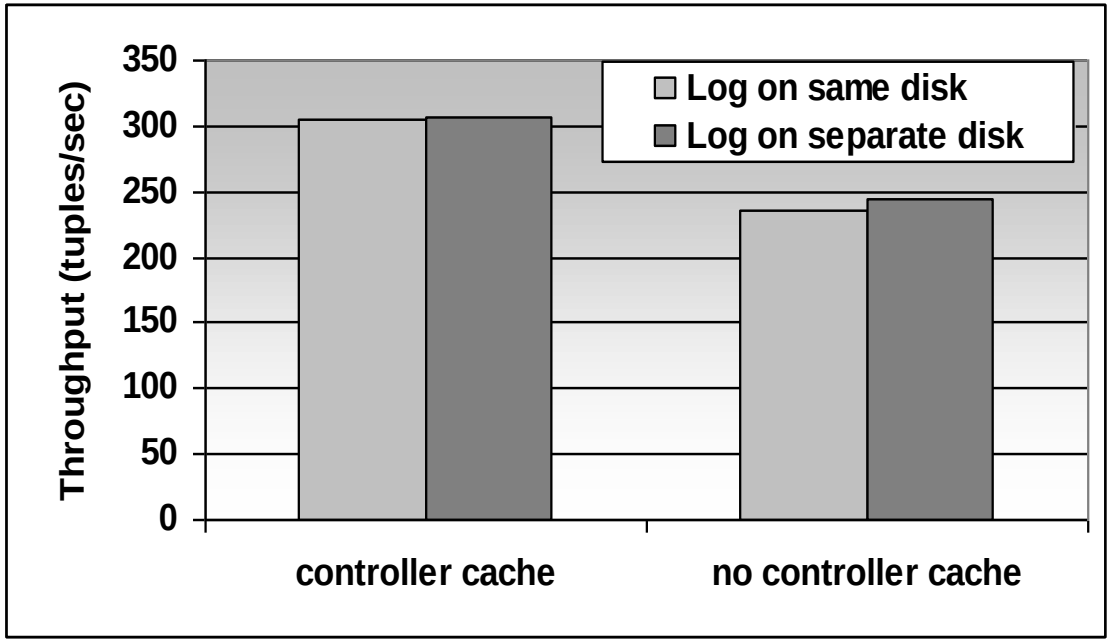


Ukládání žurnálu

- Na samostatný disk
- Záznamy žurnálu jsou ukládány sekvenčně
- Sekvenční zápis je násobně rychlejší než náhodný

Na disku žurnálu by neměla být žádná další data
+ sekvenční V/V
+ výpadek žurnálu není závislý na výpadku DB

Ukládání žurnálu



300 000 transakcí
Každá transakce = 1x INSERT

DB2 v7.1 server
5% zlepšení při žurnálu ukládaném samostatně

Cache řadiče snižuje negativní důsledek při nededikovaném ukládání

HW: střední server, Adaptec RAID controller (80Mb RAM), 2x18Gb disk.

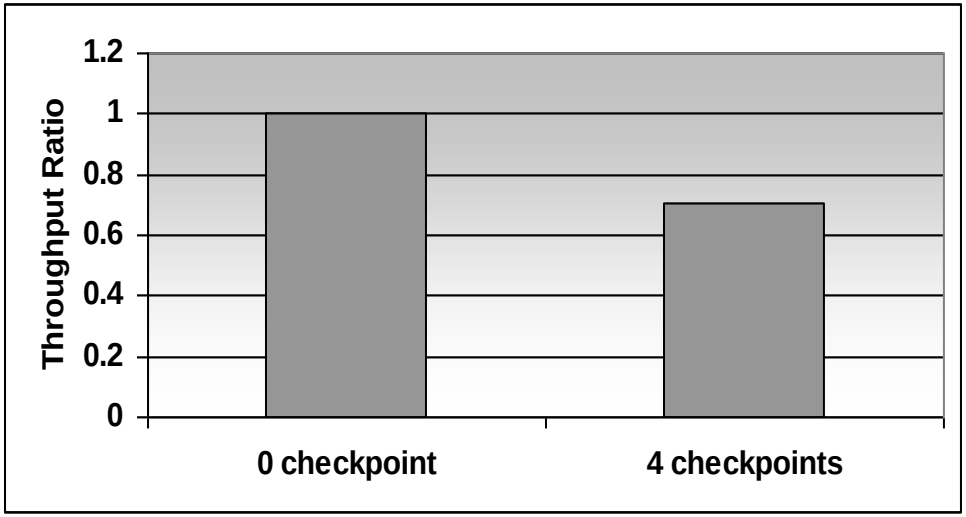
Ukládání vyrovnávacích pamětí

- Ukládání čekajících dat (dirty data)
  - Při překročení parametru v počtu stránek (Oracle 8)
  - Při překročení procenta volných stránek (méně než 3% - SQLServer 7)
  - Při provedení kontrolního bodu
  - V pravidelných intervalech

Vytváření kontrolních bodů

Vliv na výkonnost (snížení propustnosti)
Redukuje velikost žurnálu
Zkracuje čas obnovy po výpadku

300 000 transakcí
Každá transakce = jeden INSERT
Oracle 8i na Windows 2000



Shrnutí

- Konzistence dat
  - Jeden zdroj problémů: výpadky
    - Řešení:
      - žurnálování
      - redundance
  - Další zdroj problémů: sdílení dat
    - Řešení:
      - Zamykání dat při zpracování transakcí
        - Nebudeme v přednáškách řešit