Prednáška II

Logovanie a algoritmy pre obnovu

Obsah prednášky

- Logovanie a obnova dát

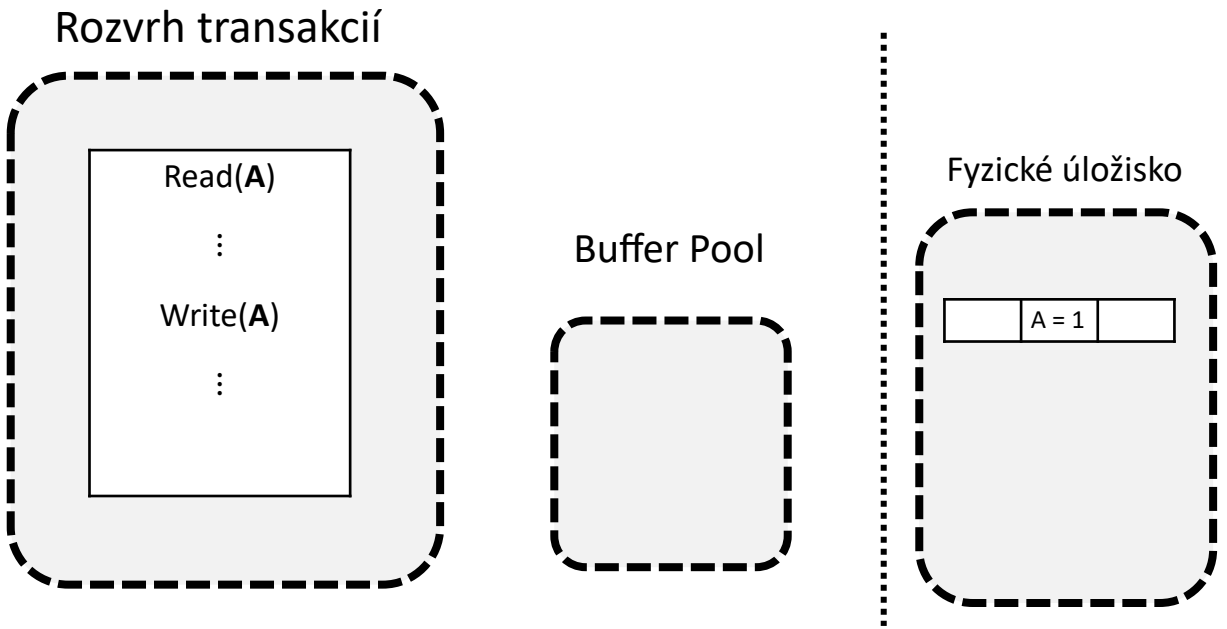
Pripomenutie

- **ACID**
  - Atomicity
  - Consistency
  - Isolation
  - Durability

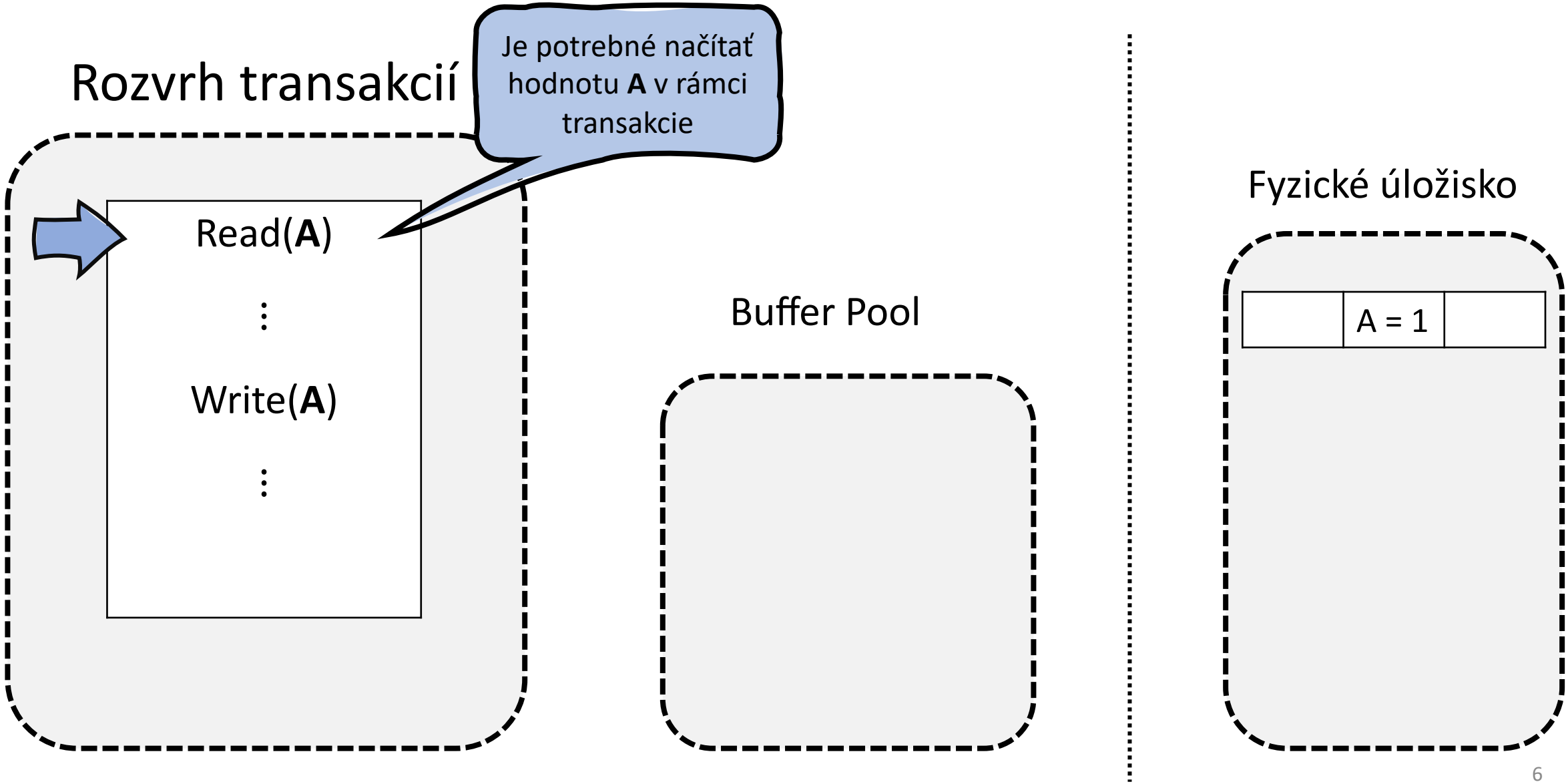
Logovanie a obnova

Pripomenutie - načítanie a ukladánie dát

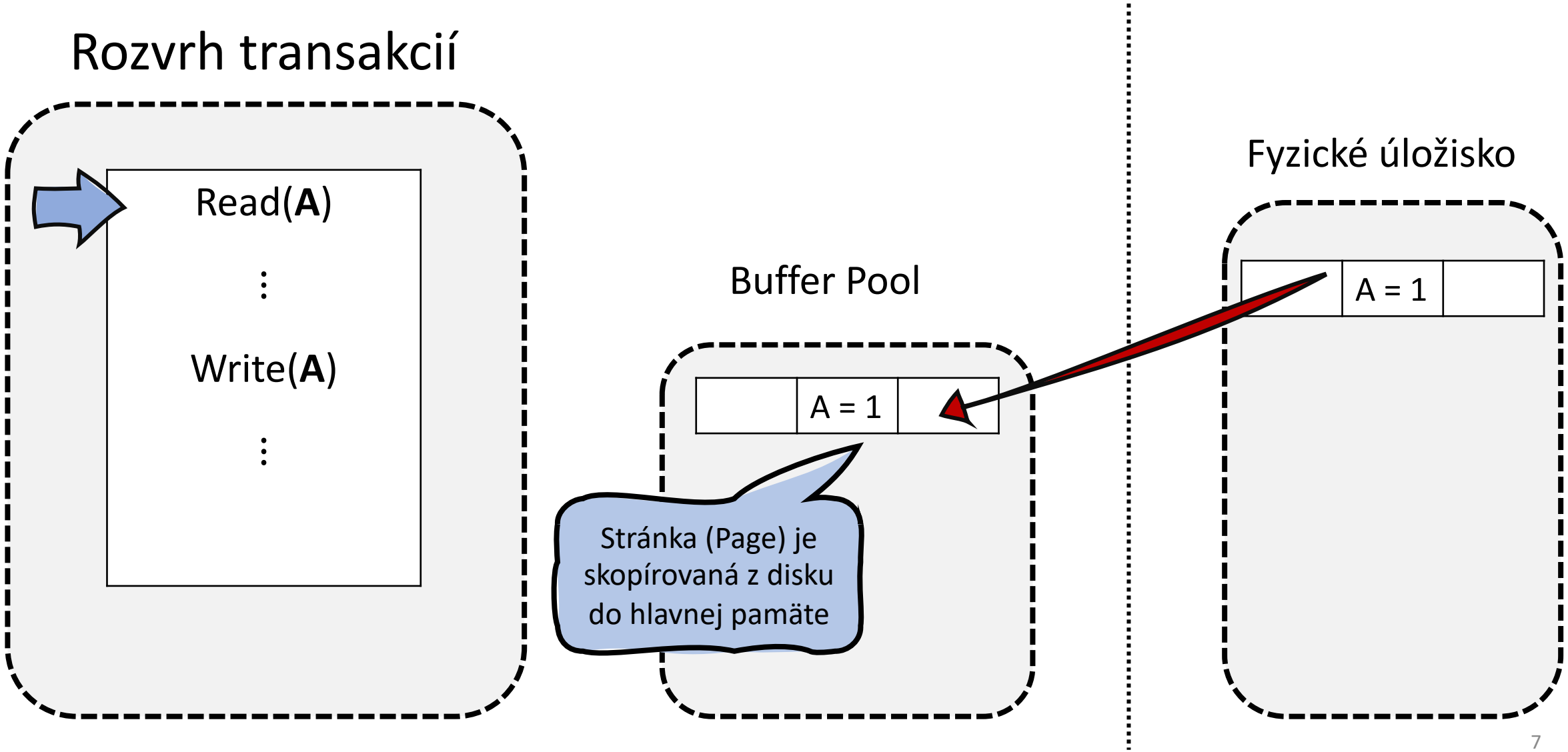
- Volatívna vs nevolatívna pamäť



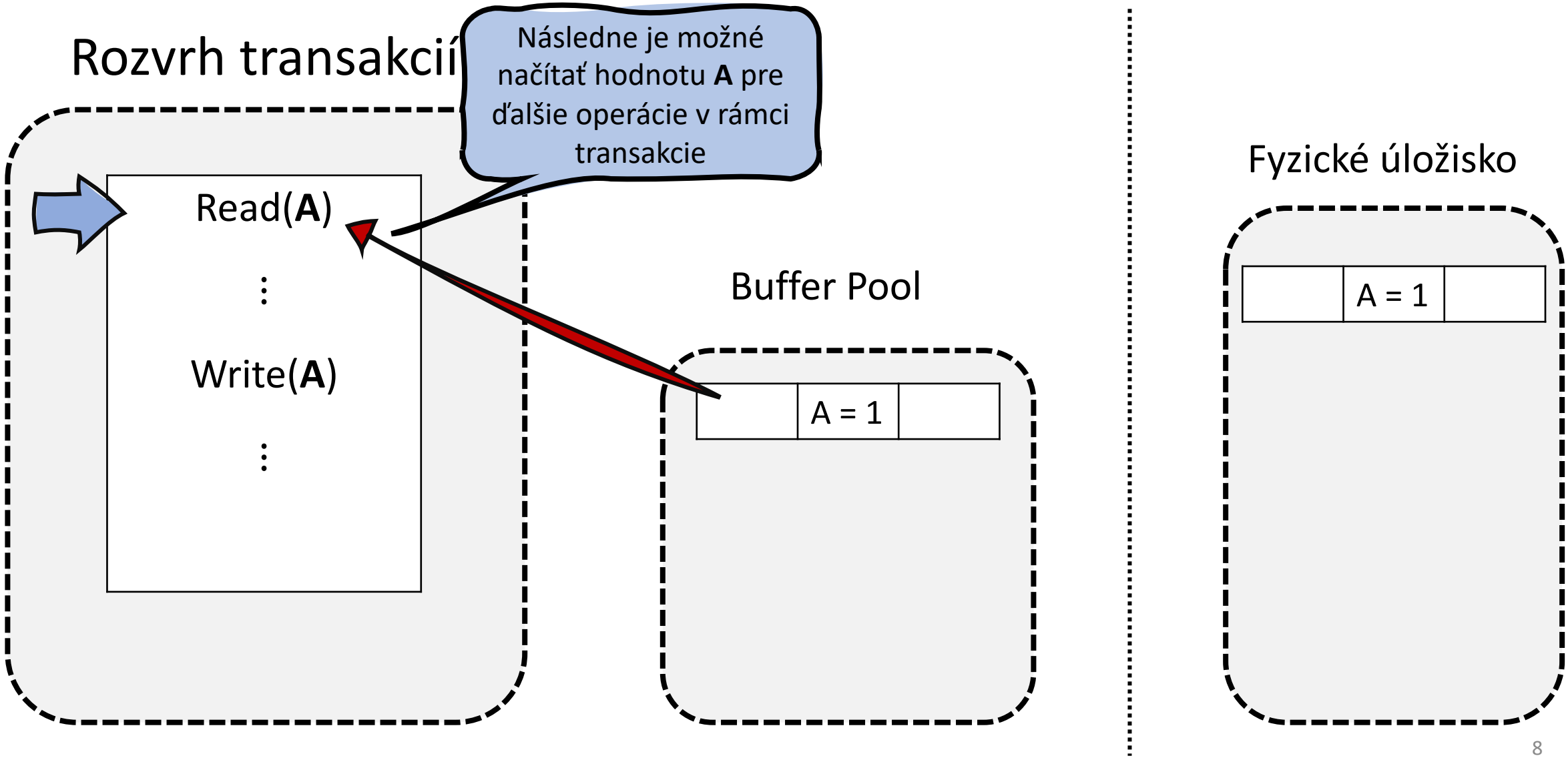
Ako prebieha načítanie a ukladanie informácii



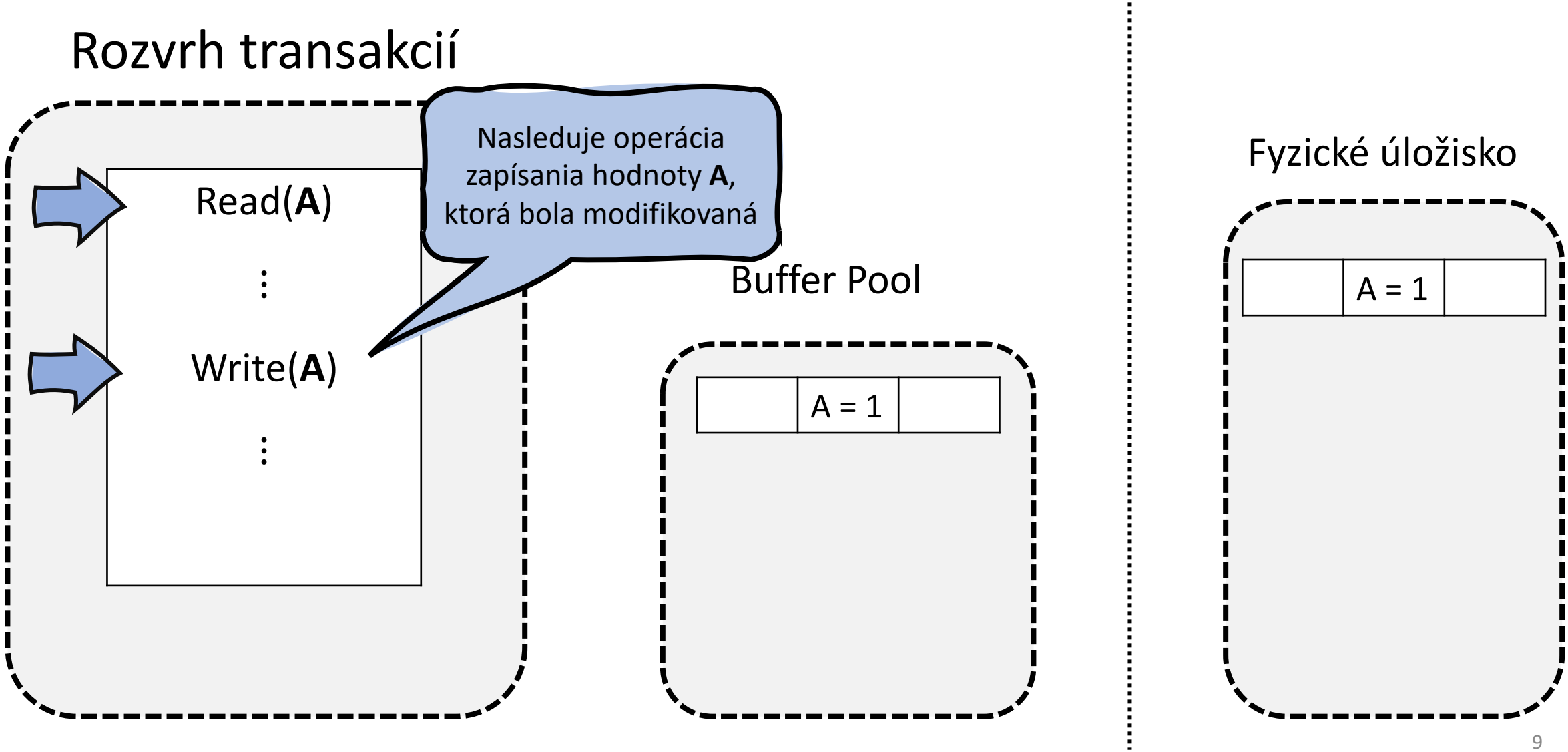
Ako prebieha načítanie a ukladanie informácii



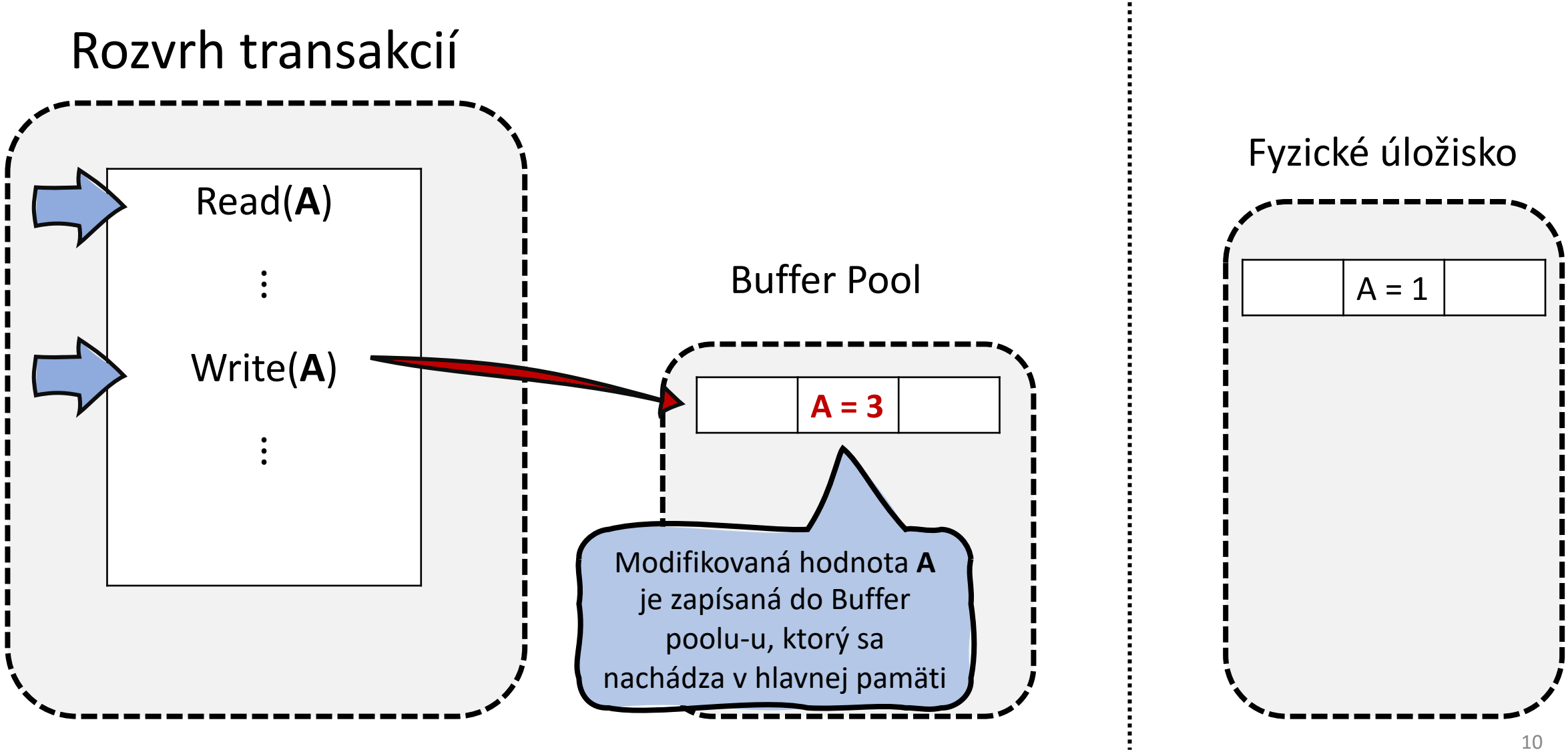
Ako prebieha načítanie a ukladanie informácii



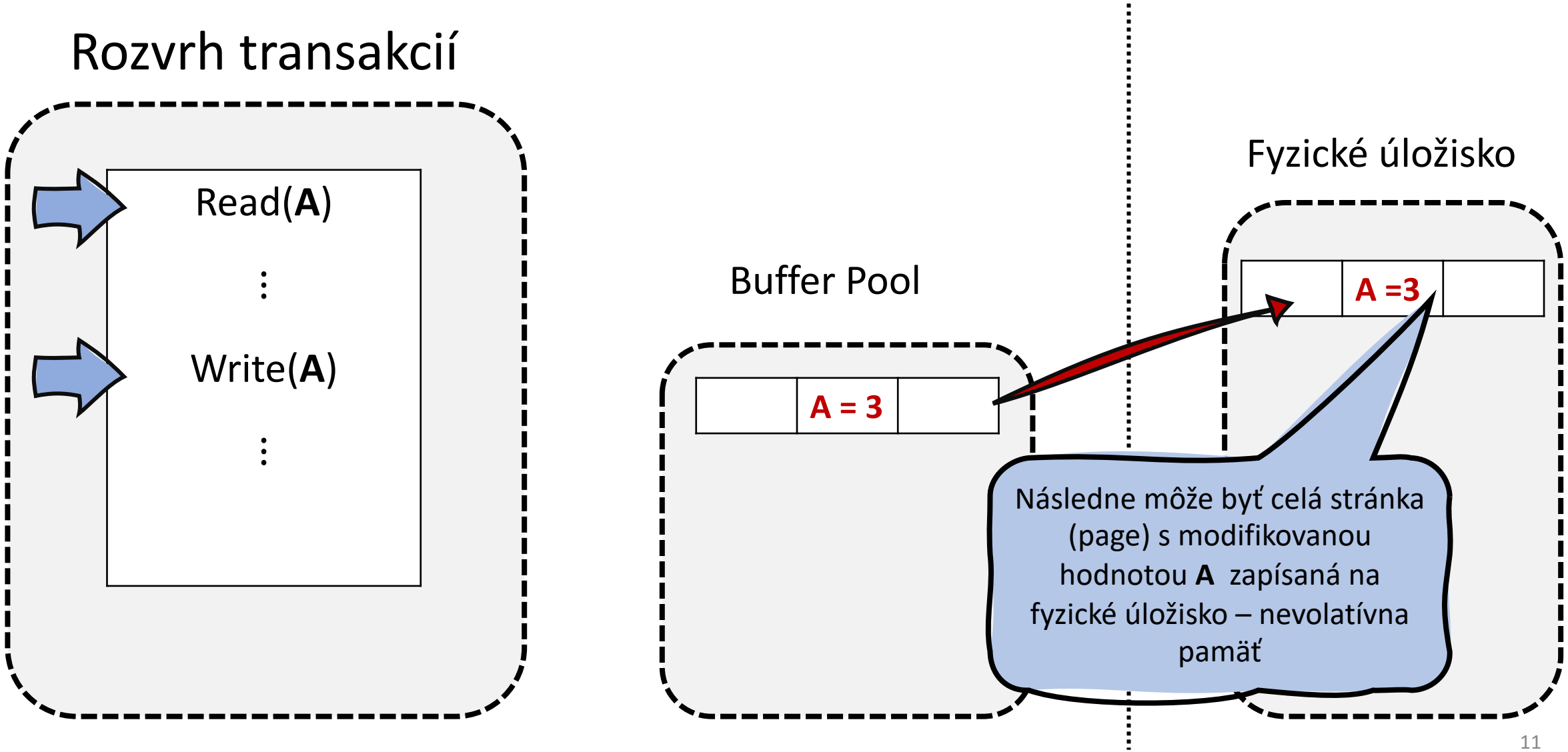
Ako prebieha načítanie a ukladanie informácii



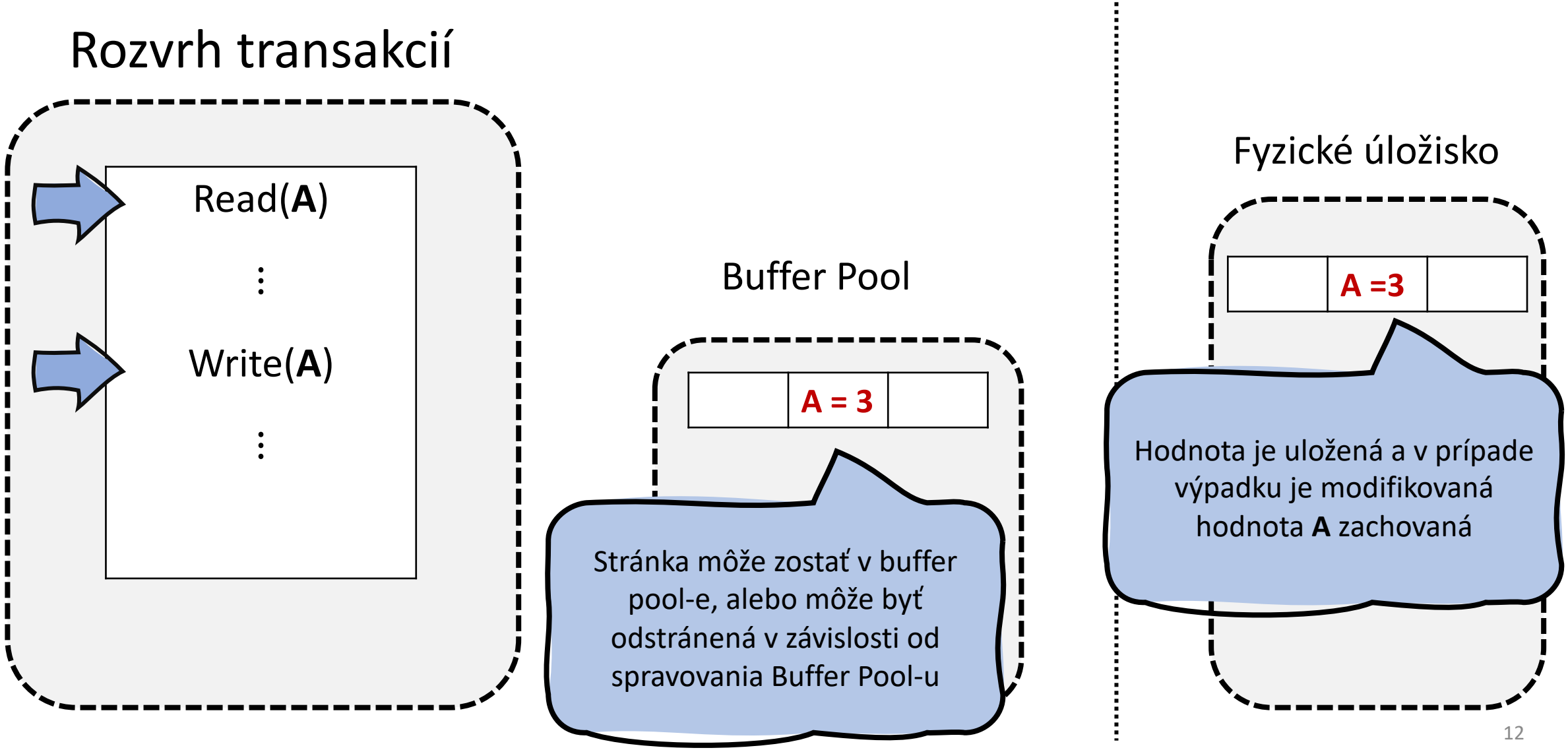
Ako prebieha načítanie a ukladanie informácii



Ako prebieha načítanie a ukladanie informácii



Ako prebieha načítanie a ukladanie informácii



Obnova databázy po zlyhaní

- Algoritmy pre obnovu (Recovery Algorithm) predstavujú mechanizmy, ktorých cieľom je zabezpečenie, že databáza bude:
  - konzistentná
  - atomicitu transakcií
  - a tiež trvácnosť (durability) databázy
- Celkovo algoritmy obnovy môžeme rozdeliť na dve časti:
  - Operácie, ktoré sa vykonávajú počas bežného spracovania transakcií – cieľom je zabezpečenie informácií, ktoré je možné použiť následne pri obnove, keď nastane zlyhanie
  - Operácie, ktoré sa vykonávajú po zlyhaní (vypádku) pre obnovenie DB pre zachovanie atomicity, konzistentnosti a trvácnosti (durability)

Motivácia - prečo riešiť obnovu dát?

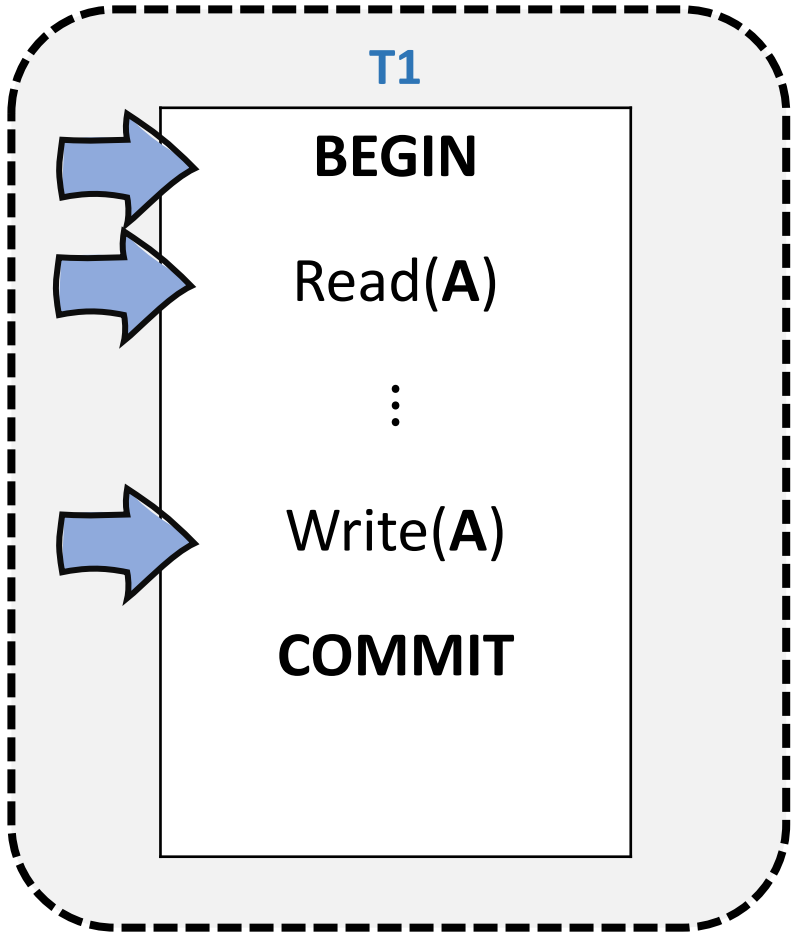
- Cieľ dosiahnuť ACID
  - v rámci algoritmou pre obnovu dát po zlyhaní vieme dosiahnuť atomicitu, konzistenciu a trvácnosť - izoláciu nám riešia protokoly pre riadenie súbežnoti napr. 2-phase locking (2PL)

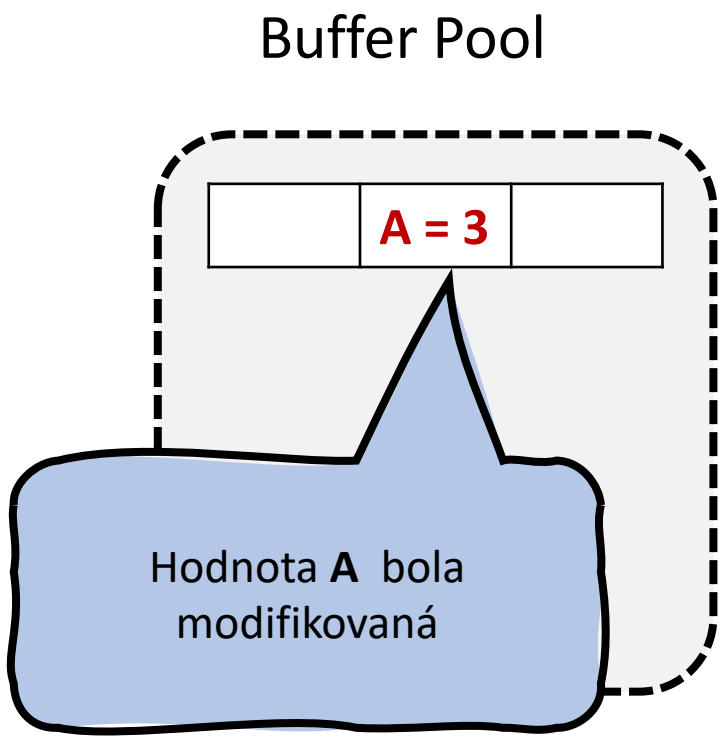
Príkad

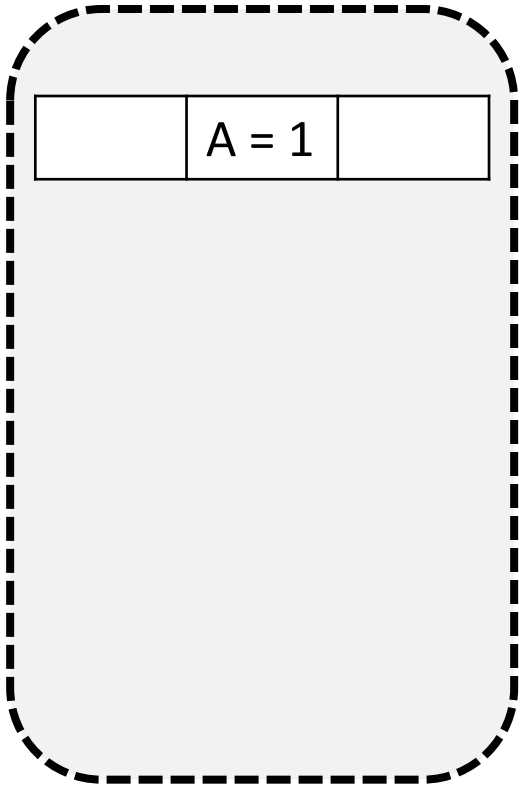
- Máme rozvrh transakcií, ktorý obsahuje transakciu **T1**
- Transakcia **T1** robí operácie nad objektom **A** (čítanie a zápis)
- Následne urobí COMMIT

Motivácia - priklad

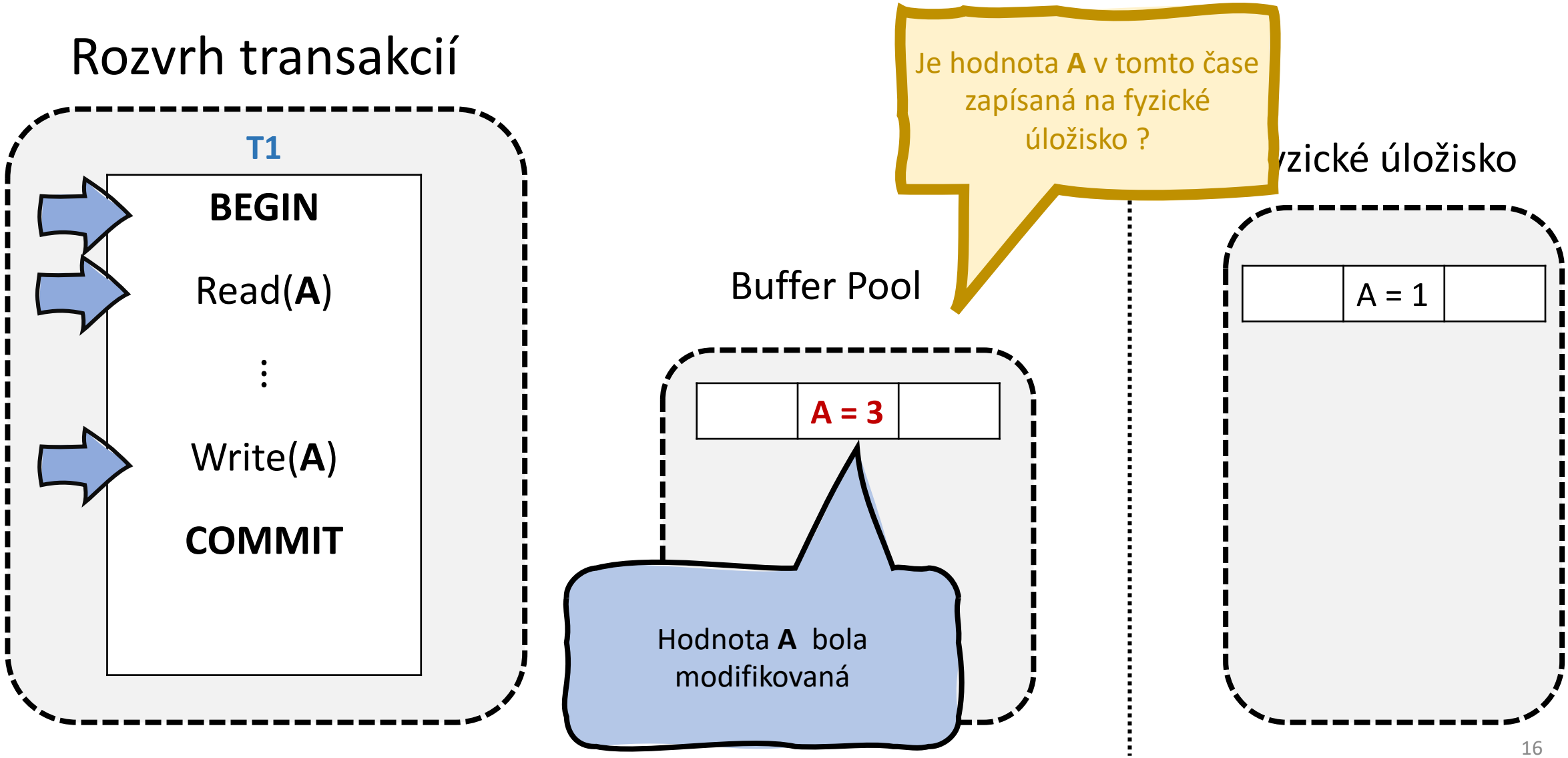
Rozvrh transakcií



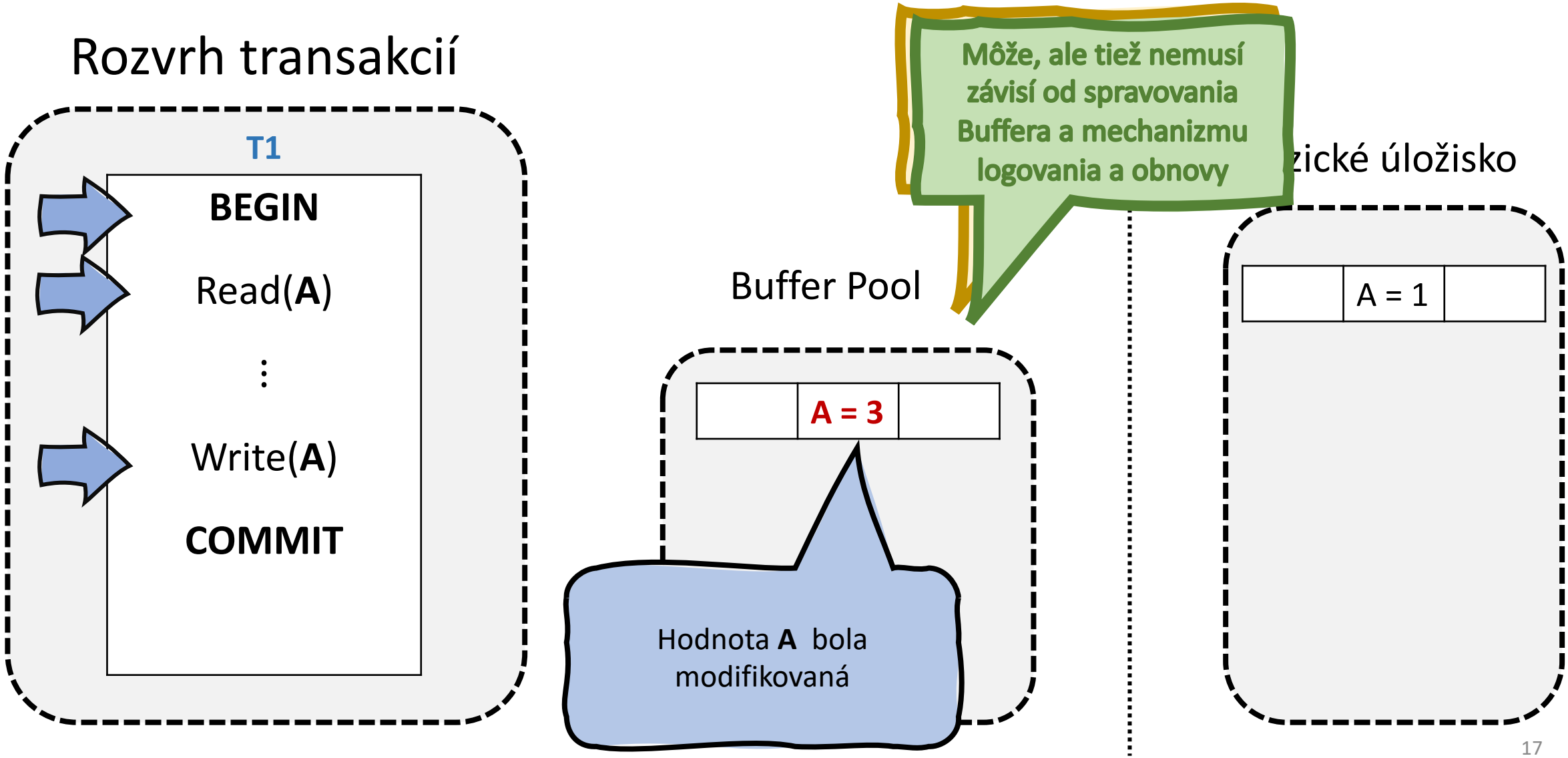




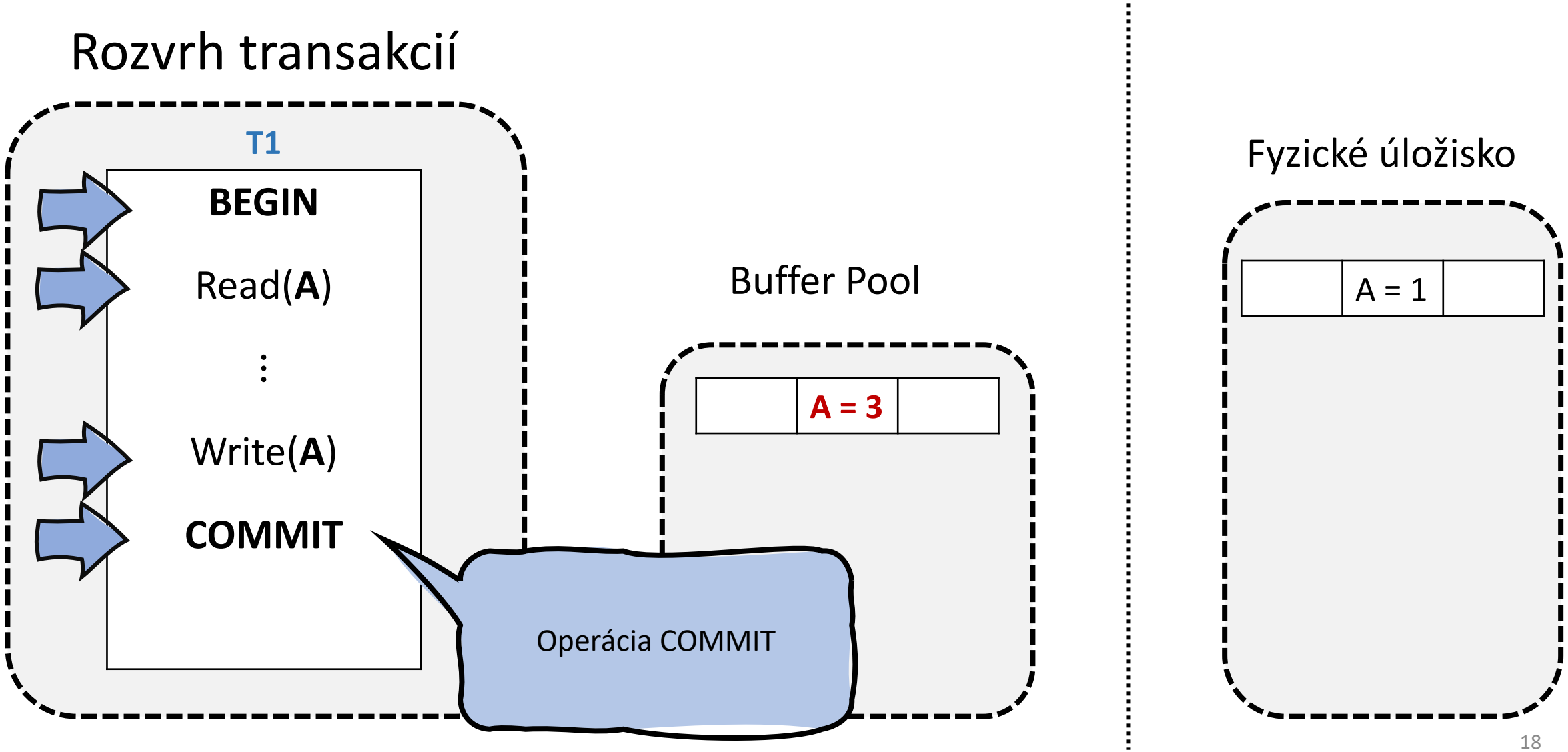
Motivácia - priklad



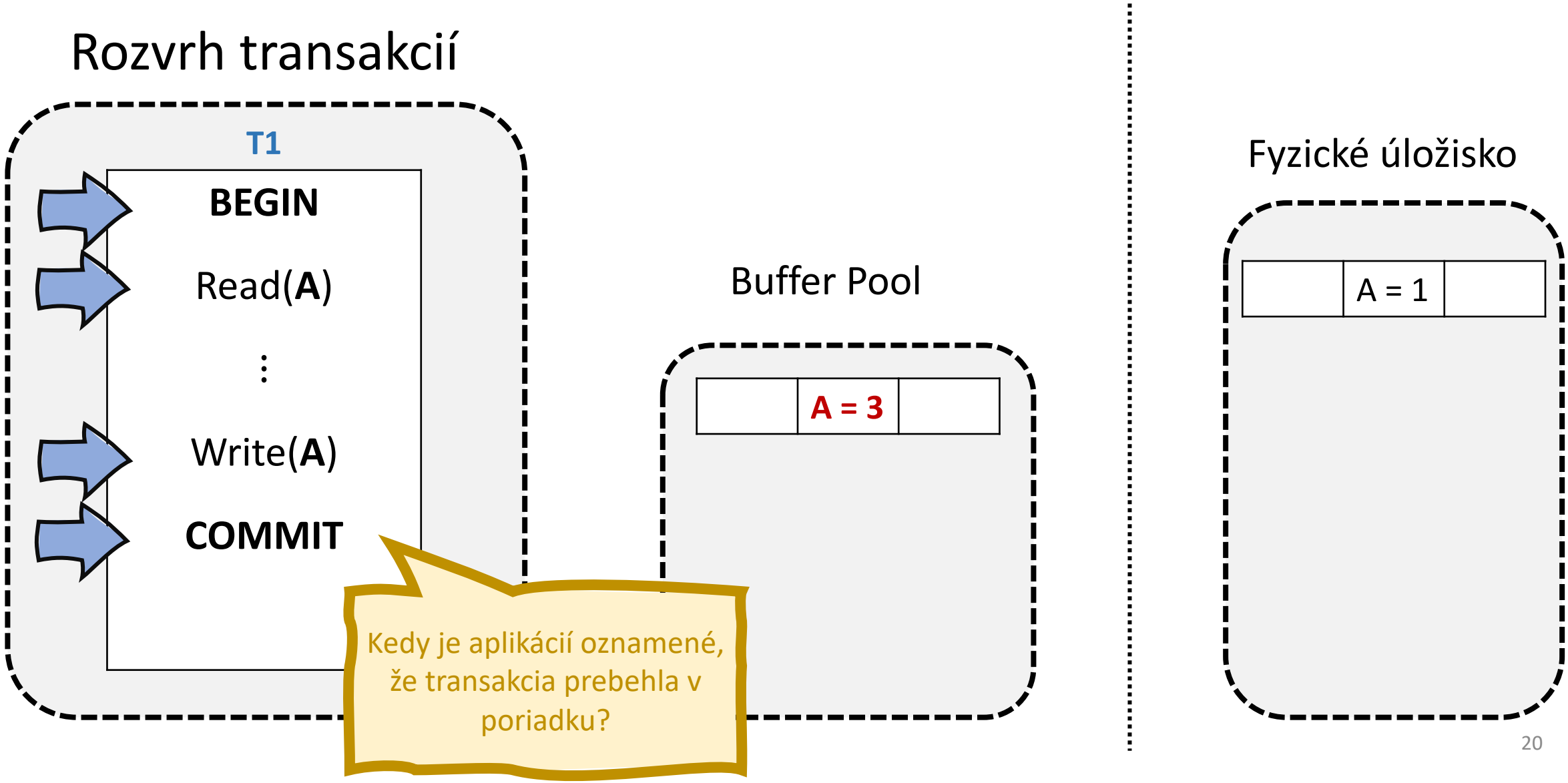
Motivácia - priklad



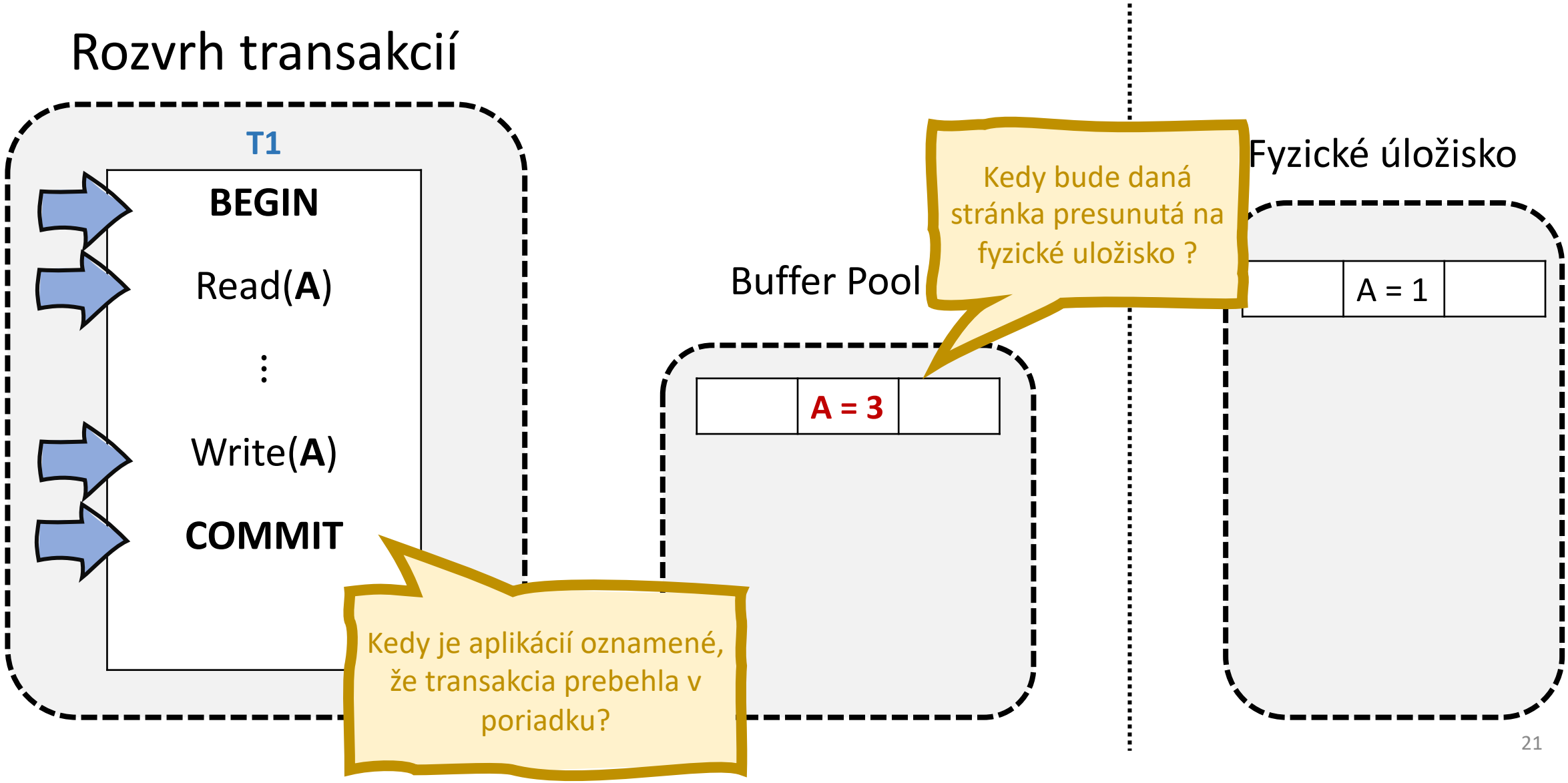
Motivácia - priklad



Motivácia - priklad

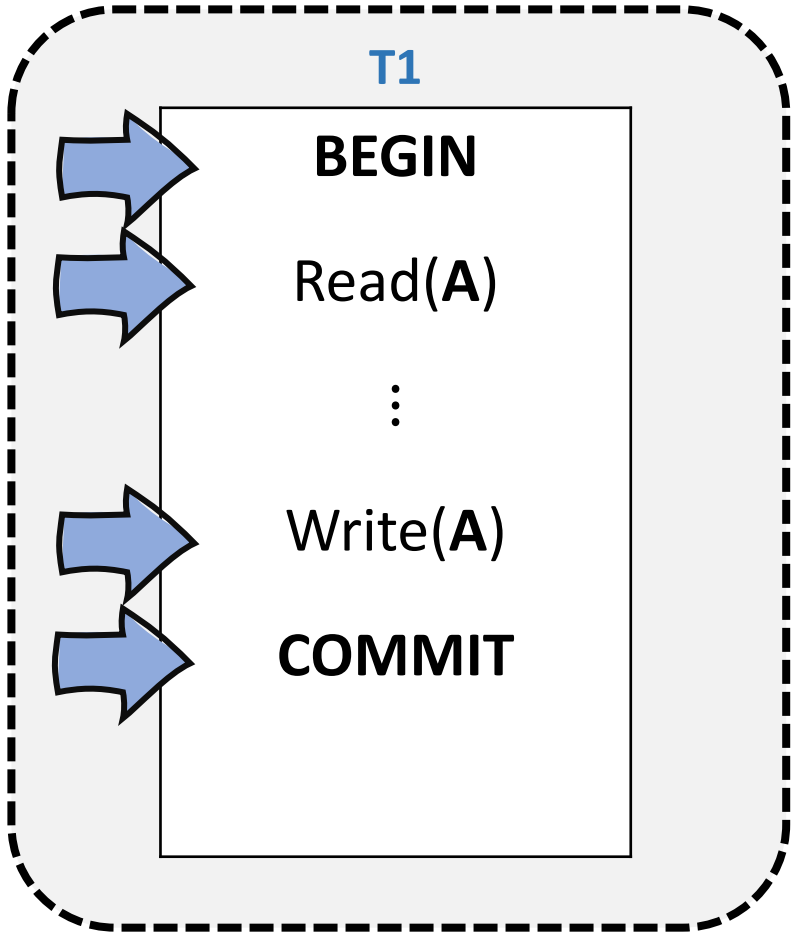


Motivácia - priklad



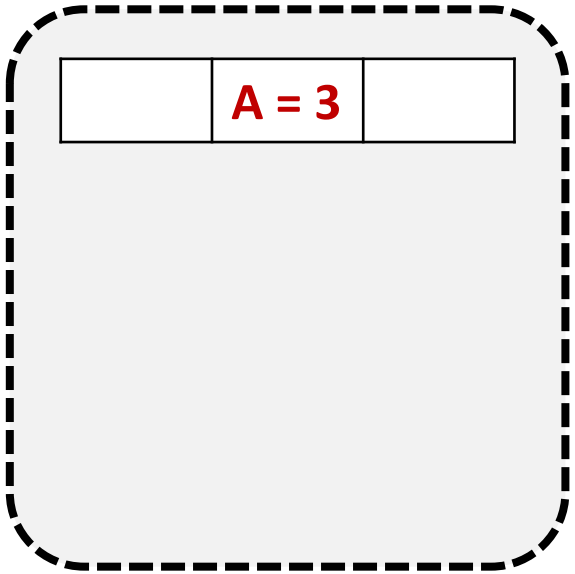
Motivácia - priklad

Rozvrh transakcií

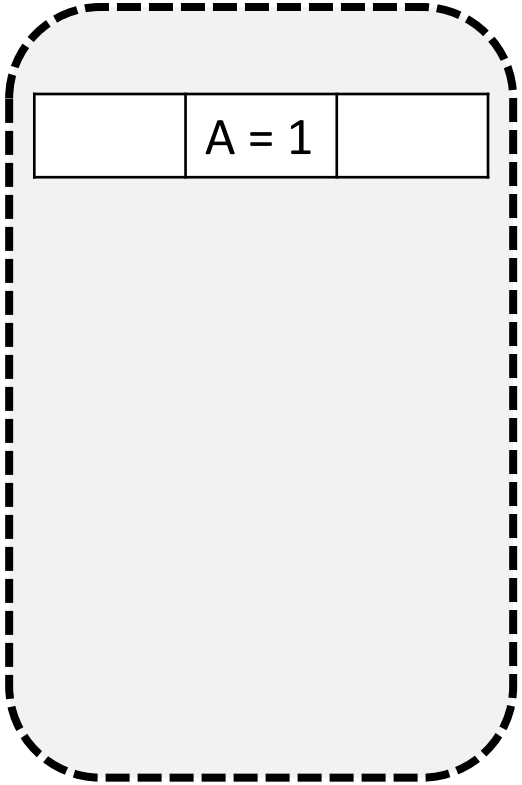


Výpadok – el. energie, útok, padnutie systému

Buffer Pool



Fyzické úložisko



Typy zlyhaní

- Rozlišujeme typy zlyhaní
  - Zlyhanie transakcie
  - Zlyhanie systému
  - Zlyhanie úložiska (Storage)

Zlyhanie transakcie

- Zlyhania transakcie môžeme rozdeliť na dve časti:
  - **Logické chyby** – transakcia nemôže byť dokončená v dôsledku nejakej vnútornej chyby napr. táto chyba vyvolá obmedzenie v DB
  - **Chyba vnútorného stavu** (Internal state error) – DBMS musí ukončiť/zrušiť transakciu v dôsledku chyby napr. vznikne deadlock

Zlyhanie systému

- **Softvérové zlyhanie** – nedostatky v implementácií DBMS napr. delenie nulou, s ktorým sa nepočítalo pri implementácií
- **Hardvérové zlyhanie**
  - Počítač, ktorý hosťuje DBMS padne napr. výpadok prúdu
  - Predpokladá sa, že výpadok nemá vplyv na nevolatívnu pamät – nie je poškodená (Fail-stop assumption)

Zlyhanie úložiska (storage)

- **Neopraviteľná hardvérová chyba**
  - zlyhanie disku napr. poškodenie sektorov, hlavy disku atď.
  - je predpoklad, že dané zlyhanie v rámci úložiska je detegovateľné napr. pomocou checksum

- DBMS sa nevie zotaviť z danej chyby – DB musí byť obnovená z nejakej zálohy

Zápis dát do nevolatívnej pamäte

- Zapísanie hodnoty môže prebehnúť dvomi spôsobmi:
  - Okamžitým zápisom dát - Immediate data modification
    - Dáta môžu byť zapísané na disk aj pred tým ako transakcia dosiahne COMMIT
    - Je nutné robiť UNDO aj REDO operácie
  - Oneskoreným zápisom dát - Deffered data modification
    - Dáta sú zapísané na disk až keď bol dosiahnutý COMMIT
    - Nie je nutné robiť UNDO operácie, postačujú REDO operácie
    - Potreba cache
- Tieto dva spôsoby súvisia s obnovovacími algoritmami a logovaním
  - preto spomenutie operácií UNDO a REDO

Manažment - Buffer pool

- Rozhoduje o tom, kedy dôjde k zápisu modifikovanej stránky na disk
  - niektoré operácie môžu vyžadovať okamžitý zápis informácií, niektoré nie
- V prípade plnej pamäte rozhoduje, ktoré stránky opustia pamäť a budú zapísané na disk
  - používajú sa algoritmy známe z OS (napr. LRU, FIFO, atď.)

- Súvisí s tým politika manažmentu buffer
  - Steal policy
  - Force policy

Manažment - buffer pool priklad

Rozvrh transakcií

T1	T2
BEGIN	
Read(**A**)	
Write(**A**)	
	BEGIN
	Read(**B**)
	Write(**B**)
	COMMIT
⋮	
ABORT	

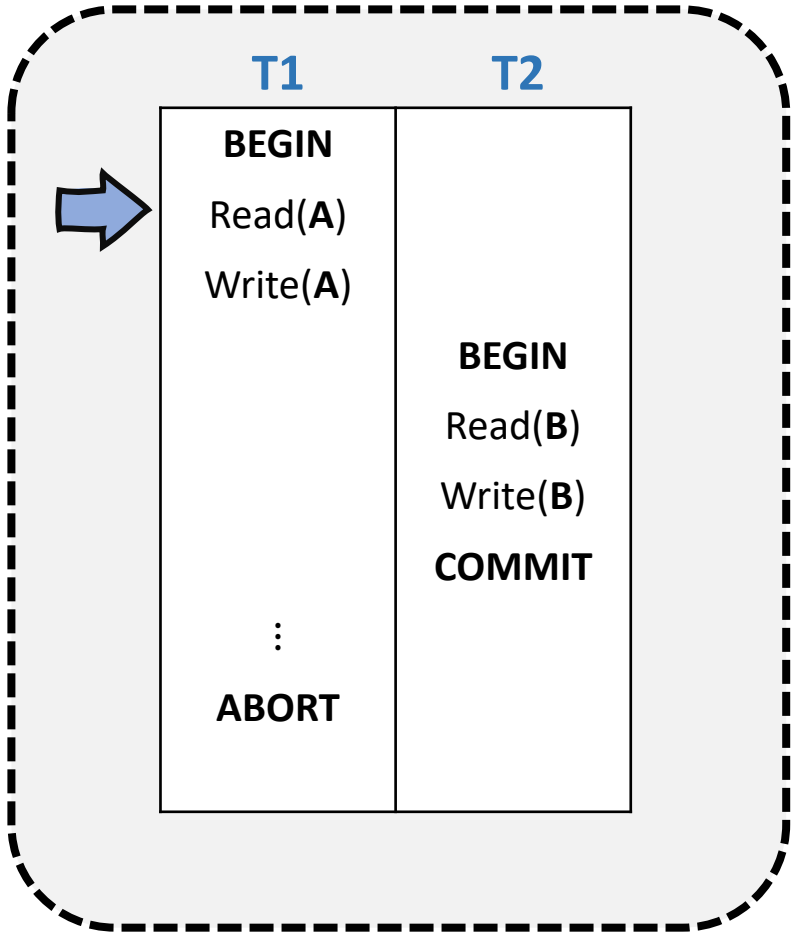
Buffer Pool

Fyzické úložisko

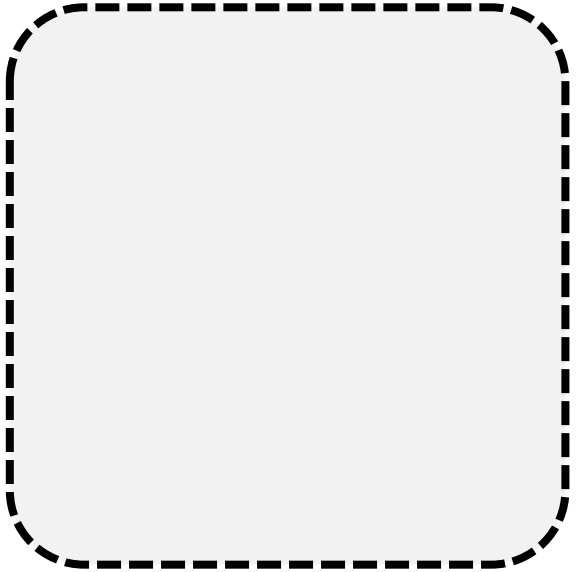
A = 4	B = 3	C = 8

Manažment - buffer pool priklad

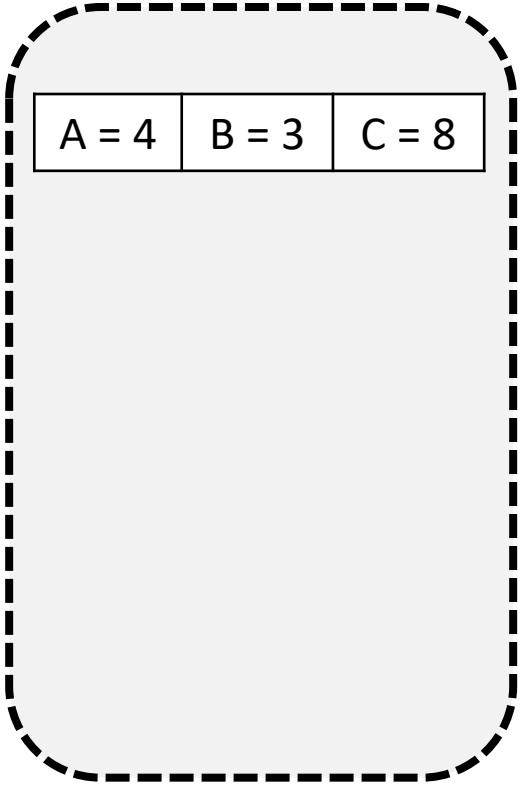
Rozvrh transakcií



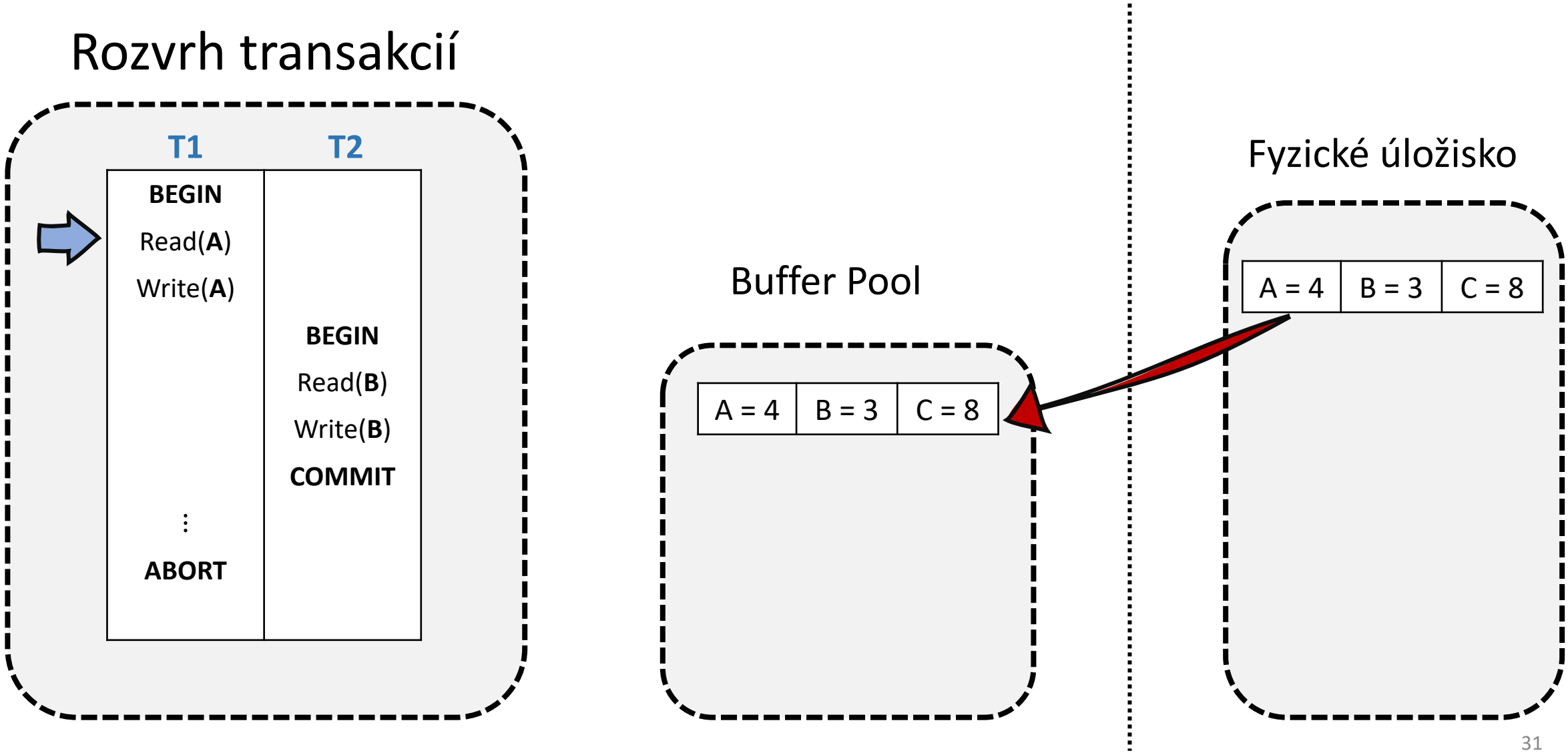
Buffer Pool



Fyzické úložisko

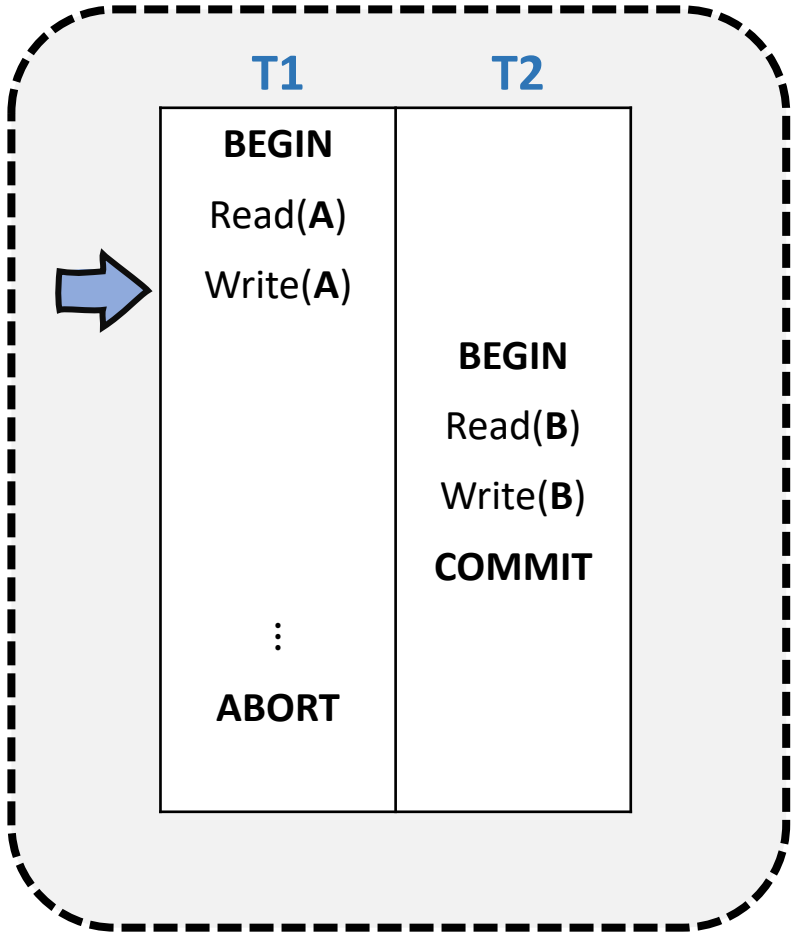


Manažment - buffer pool priklad

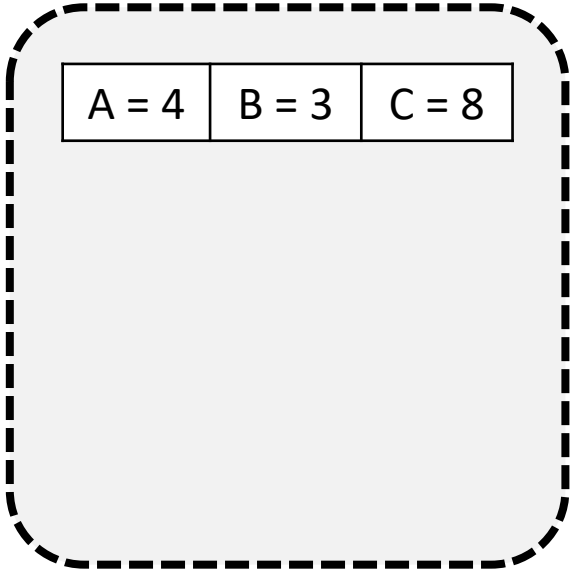


Manažment - buffer pool priklad

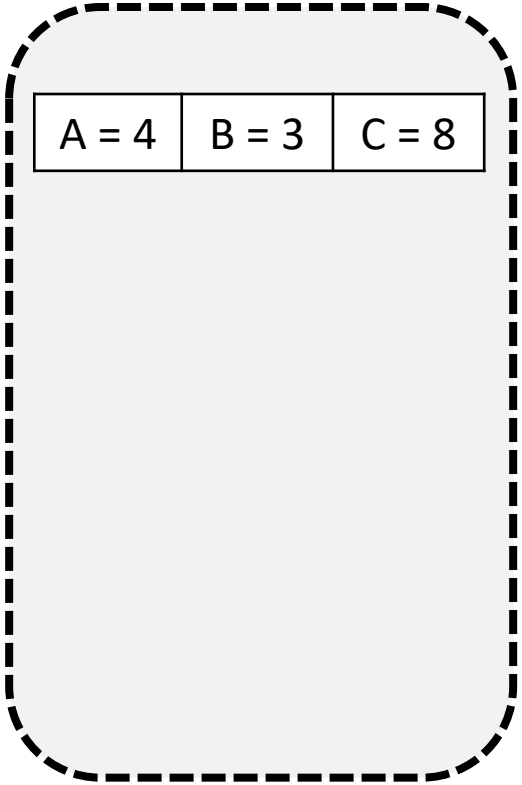
Rozvrh transakcií



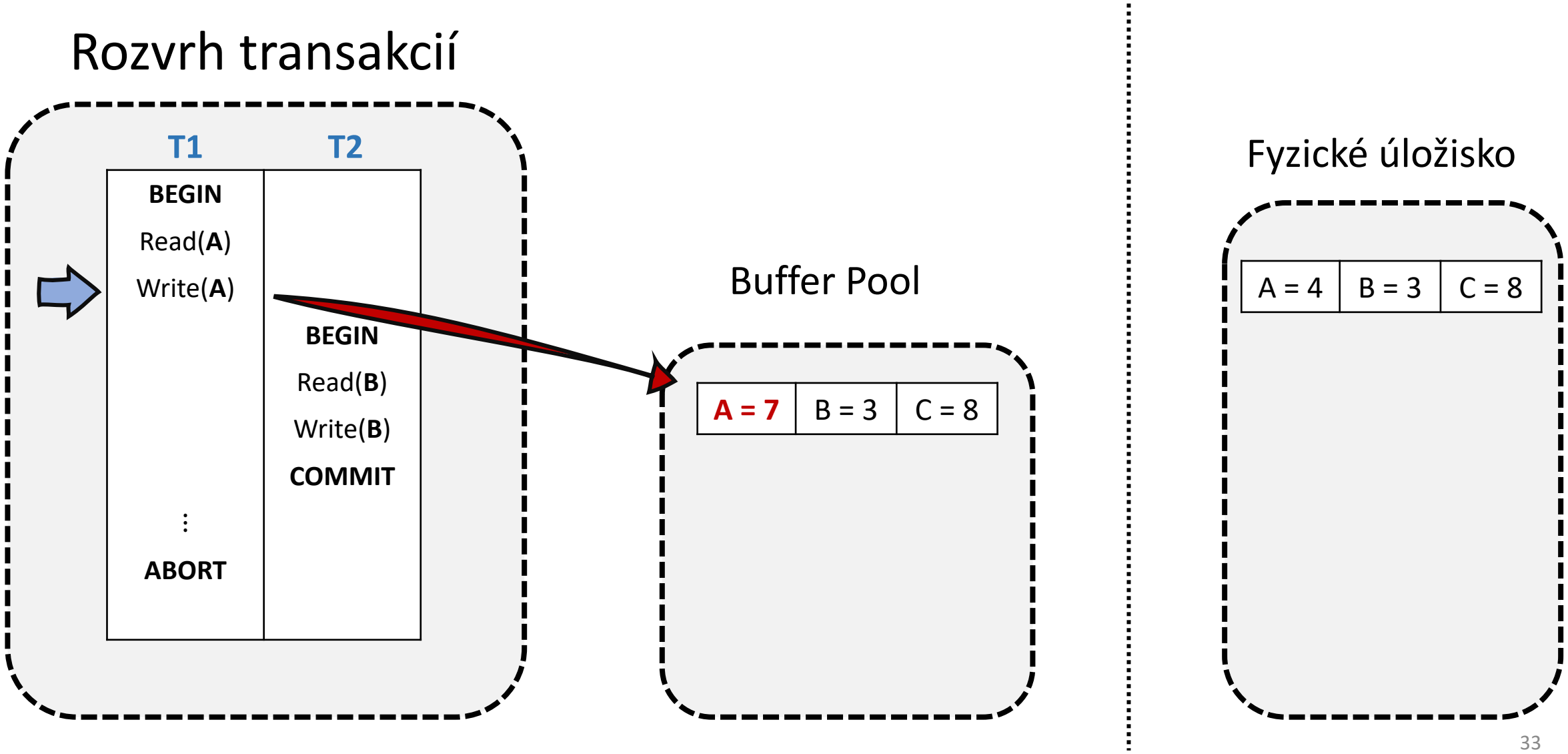
Buffer Pool



Fyzické úložisko

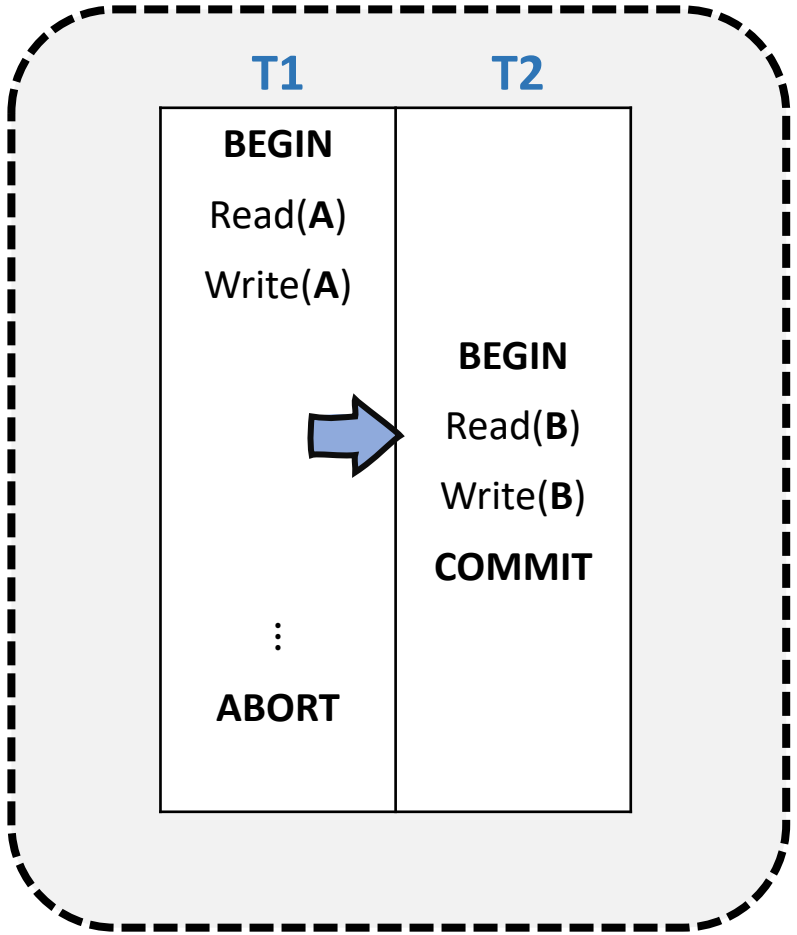


Manažment - buffer pool priklad

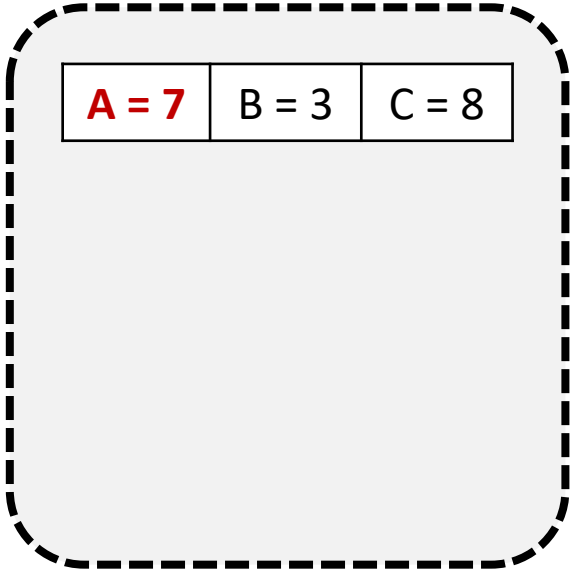


Manažment - buffer pool priklad

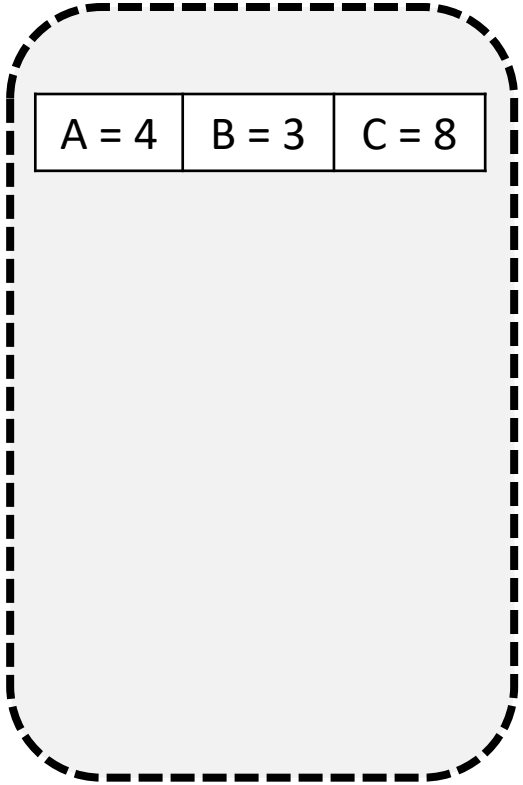
Rozvrh transakcií



Buffer Pool

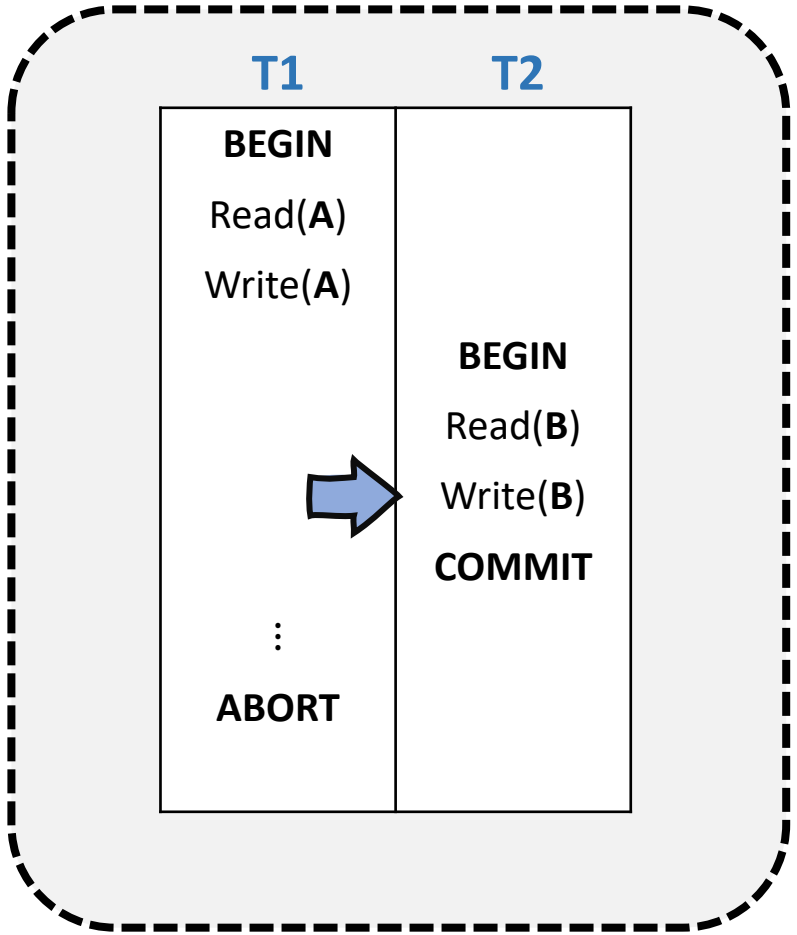


Fyzické úložisko

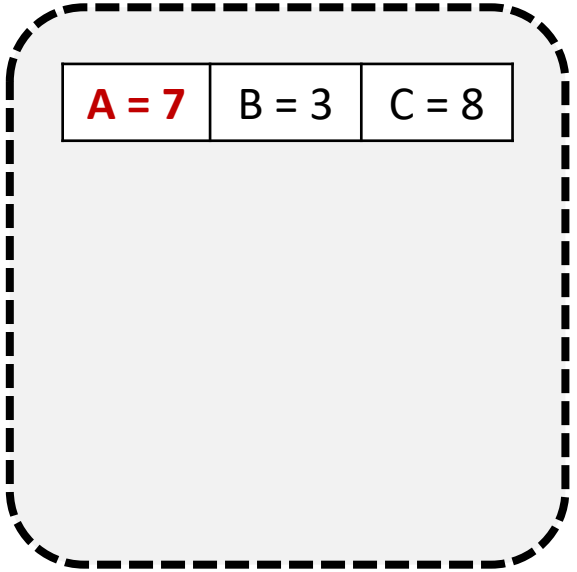


Manažment - buffer pool priklad

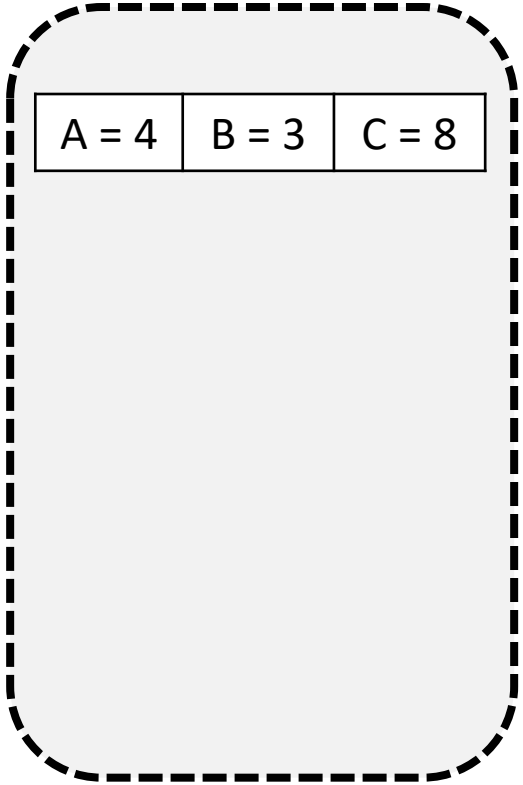
Rozvrh transakcií



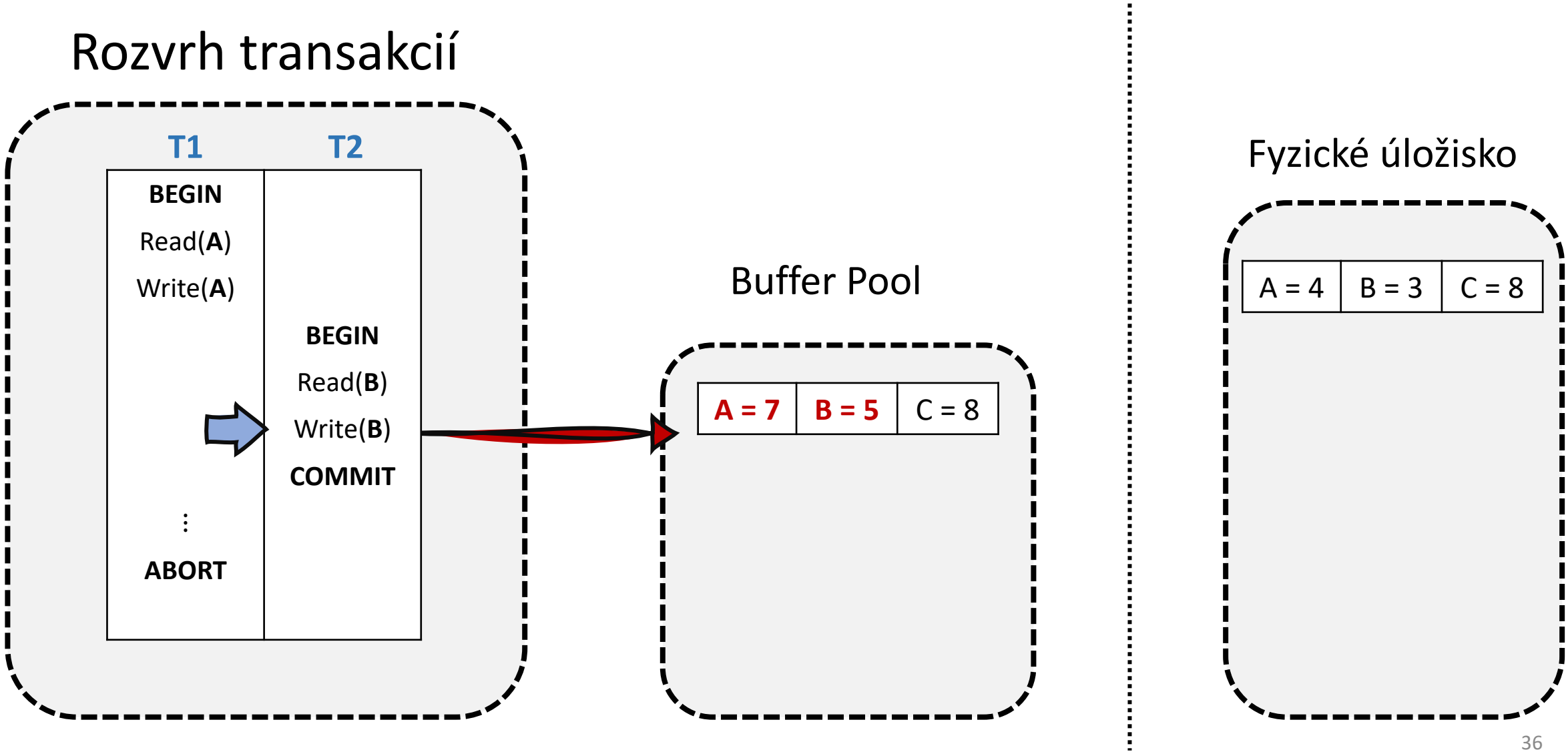
Buffer Pool



Fyzické úložisko

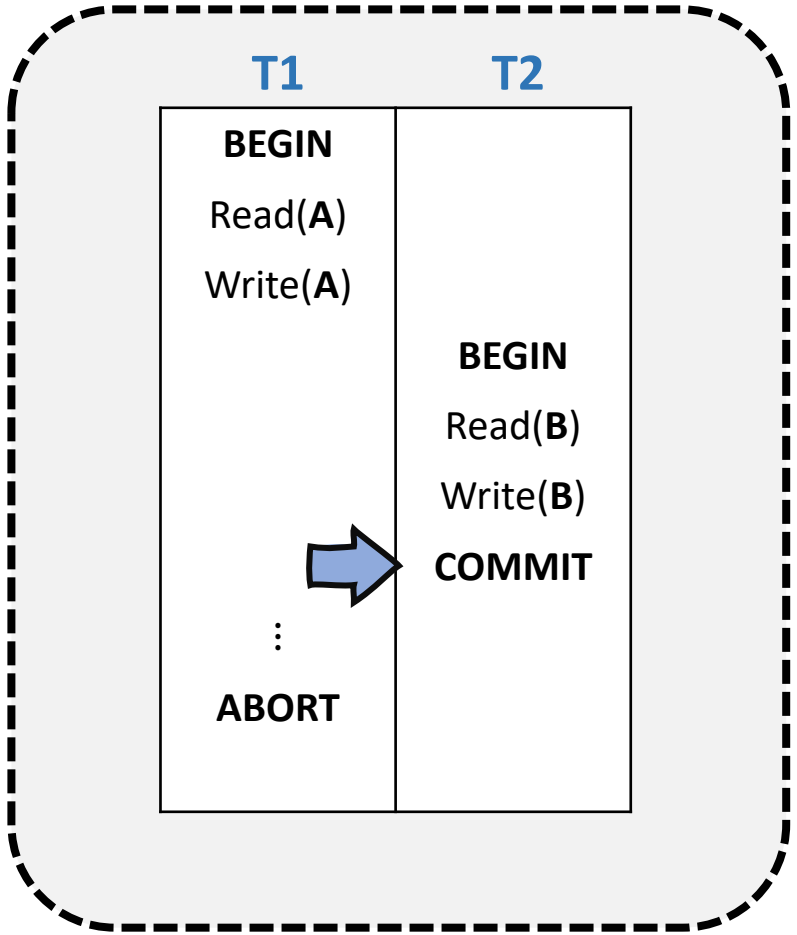


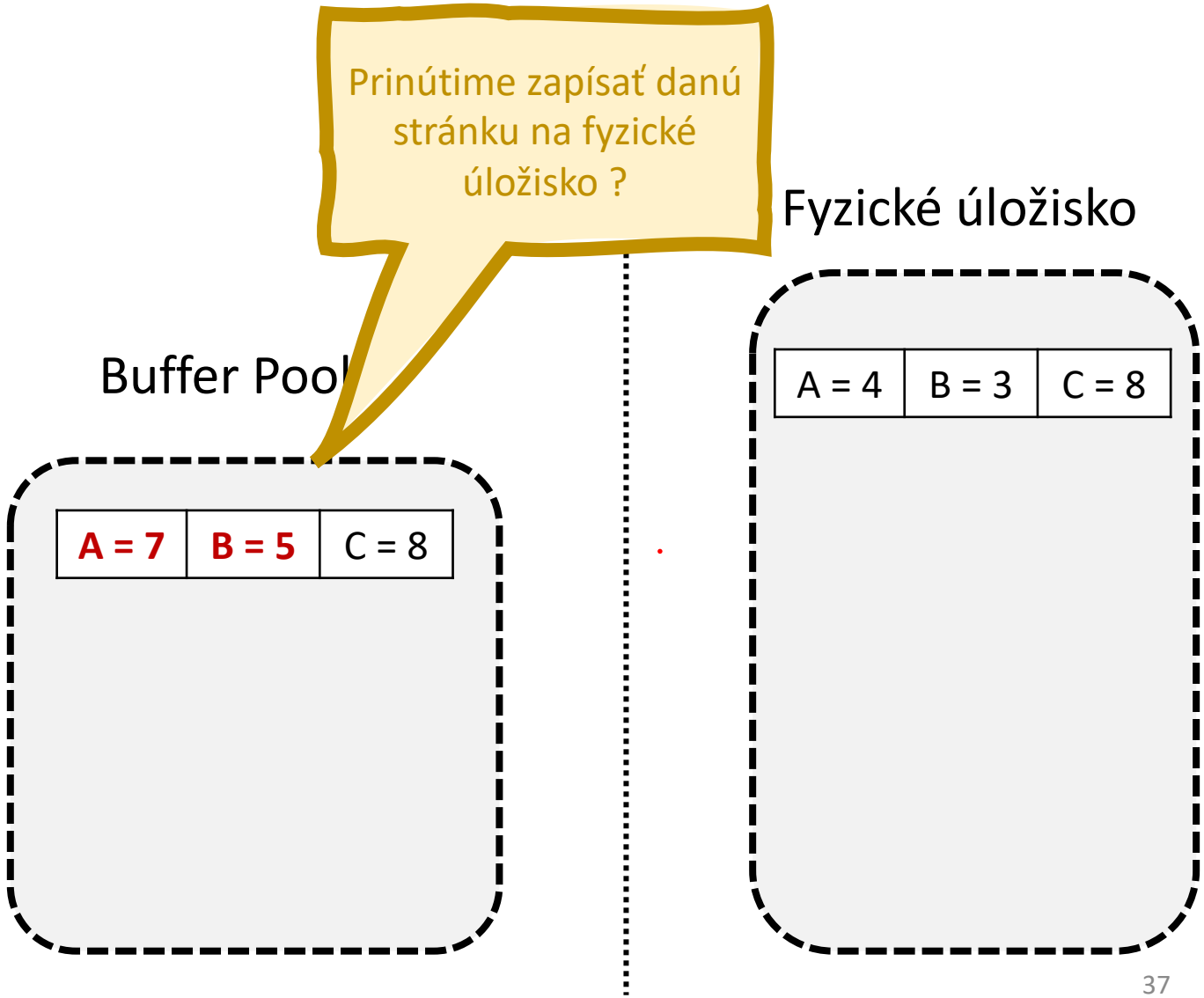
Manažment - buffer pool priklad



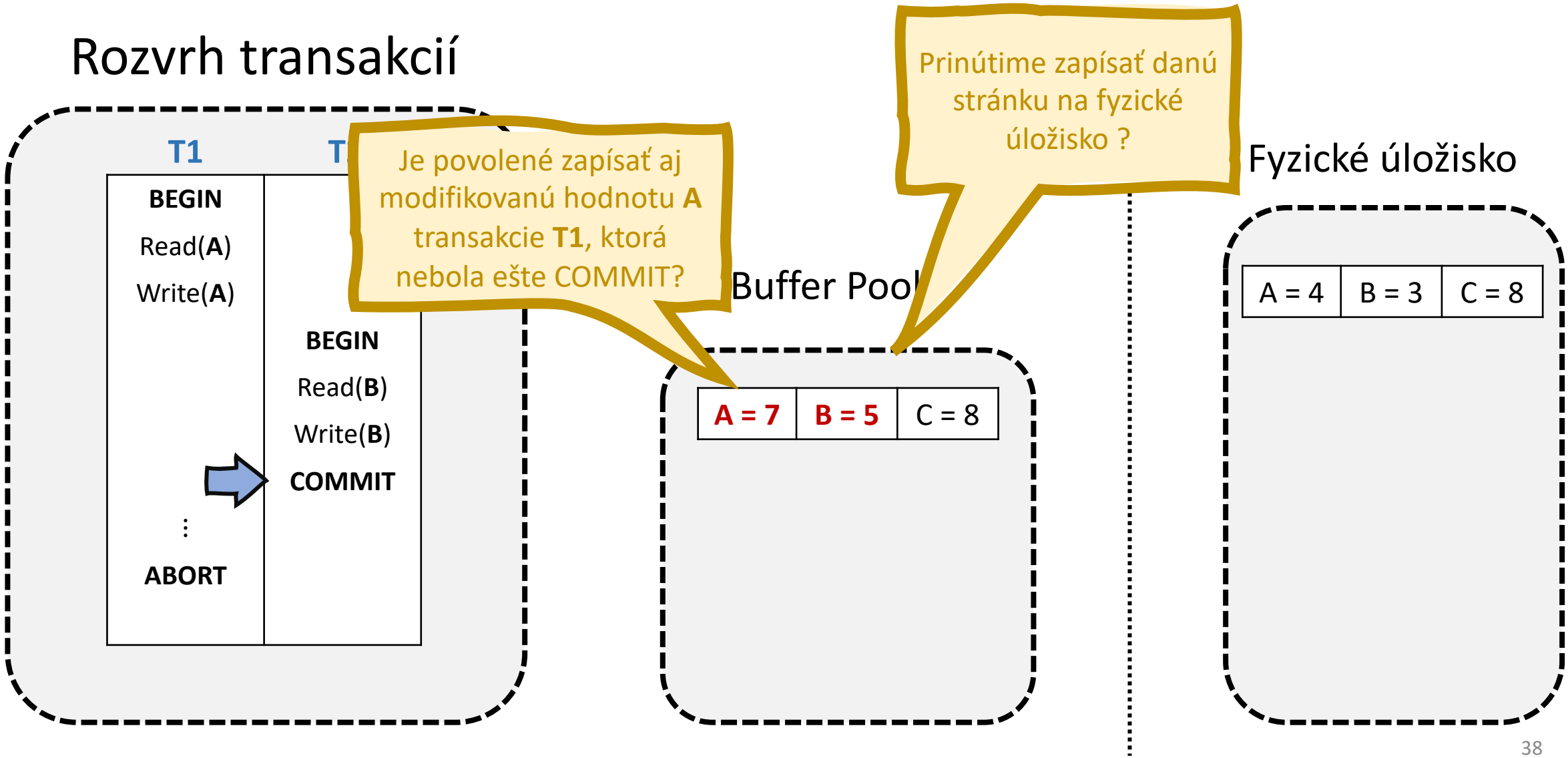
Manažment - buffer pool priklad

Rozvrh transakcií

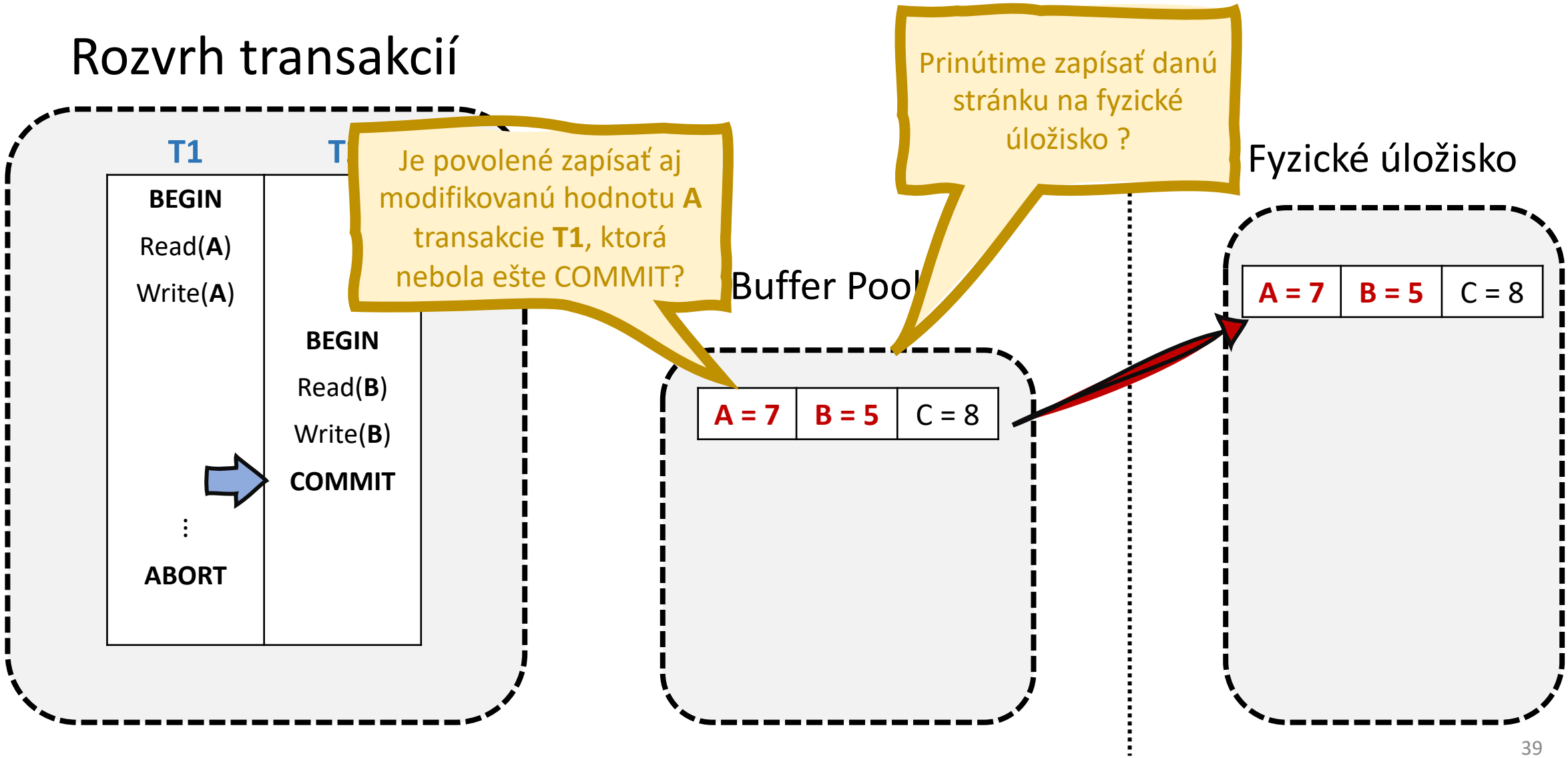




Manažment - buffer pool priklad

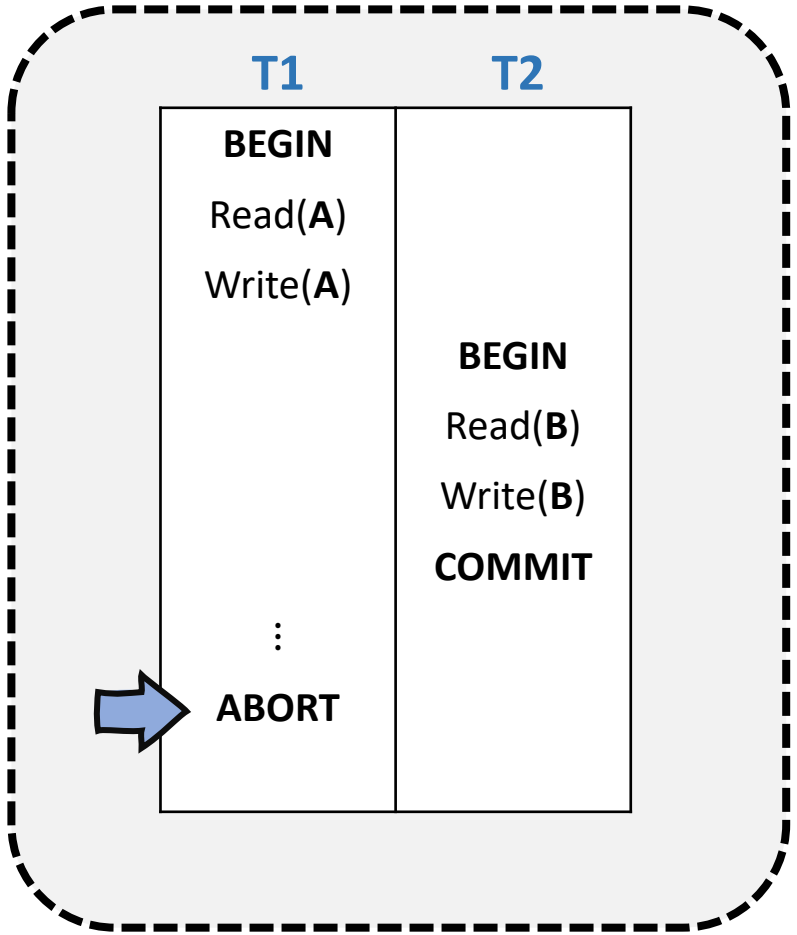


Manažment - buffer pool priklad

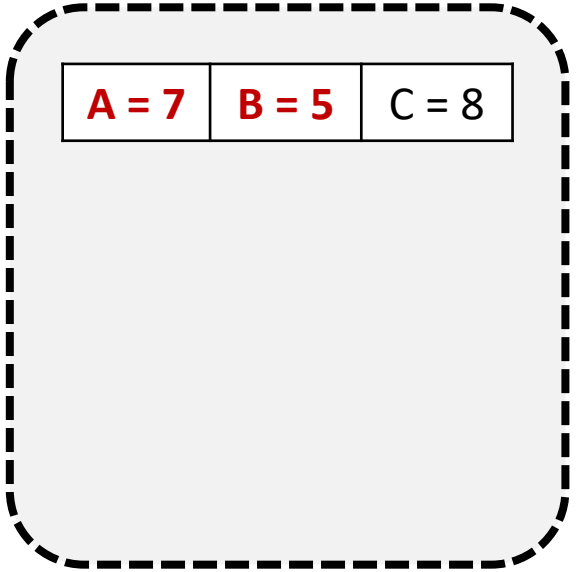


Manažment - buffer pool priklad

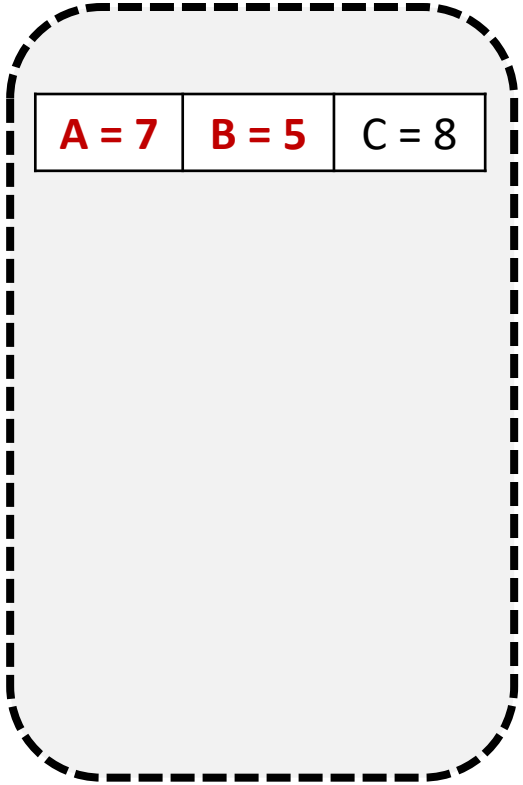
Rozvrh transakcií



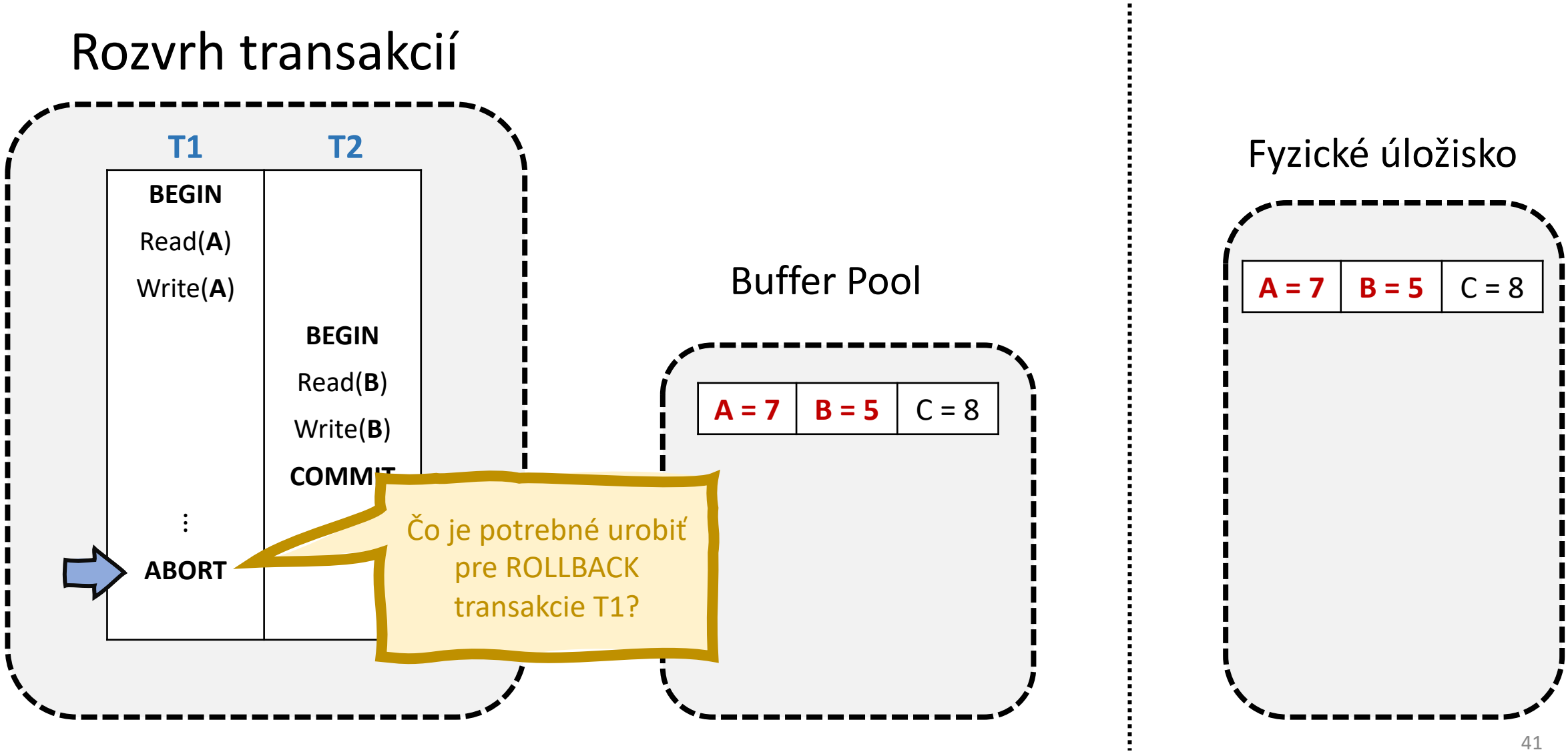
Buffer Pool



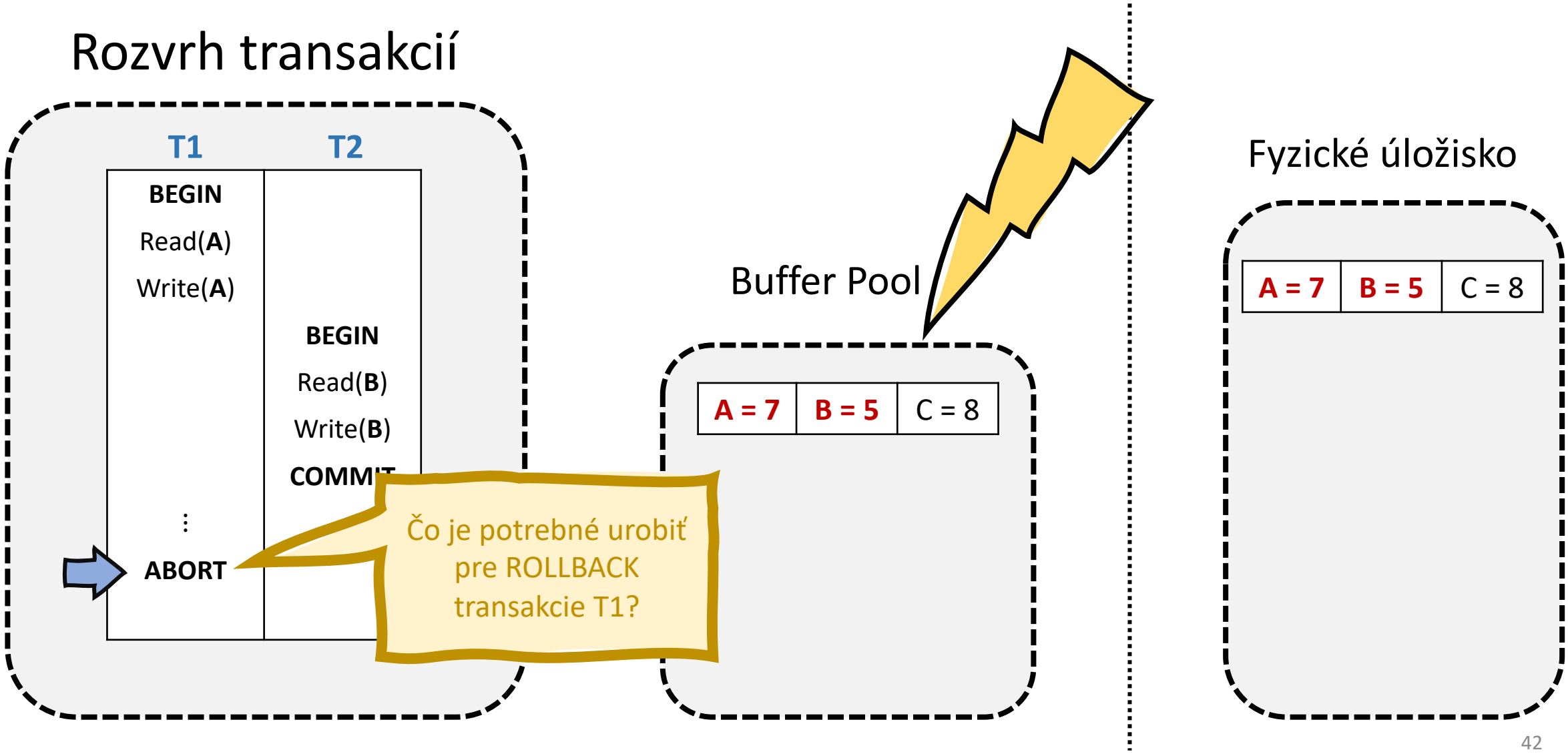
Fyzické úložisko



Manažment - buffer pool priklad



Manažment - buffer pool priklad



Steal policy

- **STEAL** nastáva vtedy, keď dochádza k prepísaniu hodnoty v nevolatívnej pamäti hodnotou transakcie, ktorá nie je COMMIT.
  - dochádza k prepísaniu hodnoty COMMIT-ovanej transakcie

- **STEAL** – je povolené
- **NO-STEAL** – nie je povolené

Force policy

- **FORCE** všetky zmenené hodnoty sú zapísané do nevolatívnej pamäte pred tým než je samotný COMMIT dovolený (oznámený aplikácií)
  - to znamená, že aktualizované hodnoty sú zapísané do nevolatívnej pamäte a až následne je oznámený commit

- **FORCE** – je vyžadované uskutočňovanie FORCE
- **NO-FORCE** – nie je vyžadované uskutočňovanie FORCE

Porovnanie Steal a Force policy

Výkonosť systému

	No-steal	Steal
No-Force	-	Rýchle
Force	Pomalé	-

Rýchlosť obnovy

	No-steal	Steal
No-Force	-	Pomalé
Force	Rýchle	-

- Skoro každý DBMS používa **NO-FORCE + STEAL**

Operácie UNDO a REDO

- **UNDO** operácia - proces, ktorý odstraňuje efekt na DB nekompletných transakcií alebo zrušených (Abort)
  - vracia hodnoty do pôvodného stavu ako boli pred začatím transakcie

- **REDO** operácia – proces, ktorý opätovne zavádza zmeny vykonané COMMIT transakciami (Commited)
  - kvôli výpadku nie sú uchované v DB ale je ich možné obnoviť

- To ako DBMS podporuje tieto operácie závisí od implementácie spravovania buffer pool-u (napr NO-STEAL + FORCE)

Porovnanie Steal a Force policy

Výkonosť systému

	No-steal	Steal
No-Force	-	Rýchle
Force	Pomalé	-

Rýchlosť obnovy

	No-steal	Steal
No-Force	-	Pomalé
Force	Rýchle	-

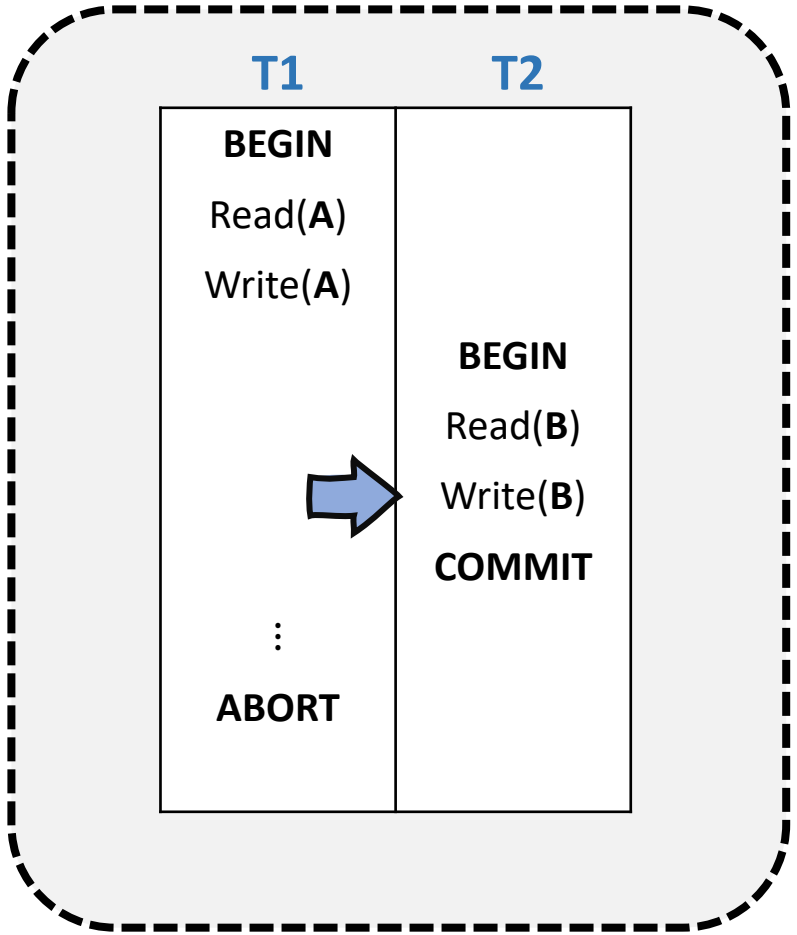
UNDO + REDO

NO UNDO + NO REDO

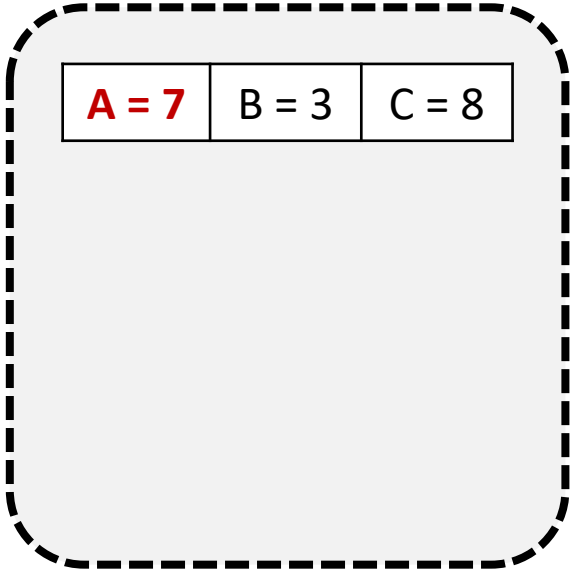
- Skoro každý DBMS používa **NO-FORCE + STEAL**

No steal + Force

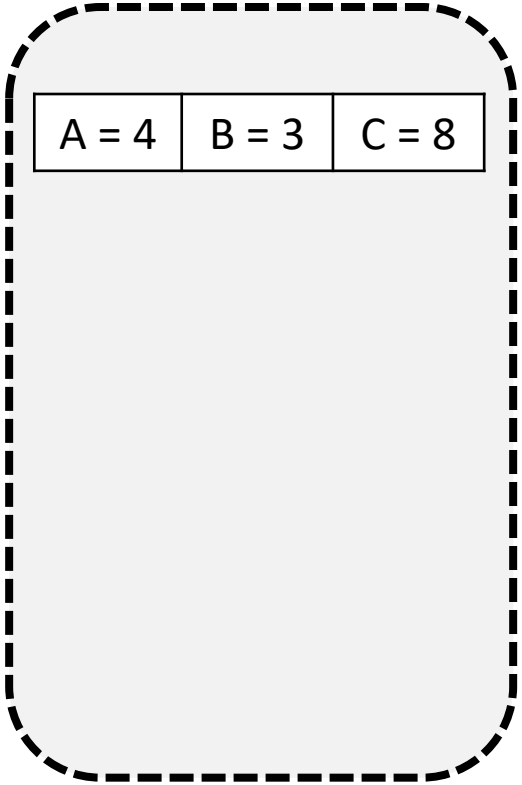
Rozvrh transakcií



Buffer Pool

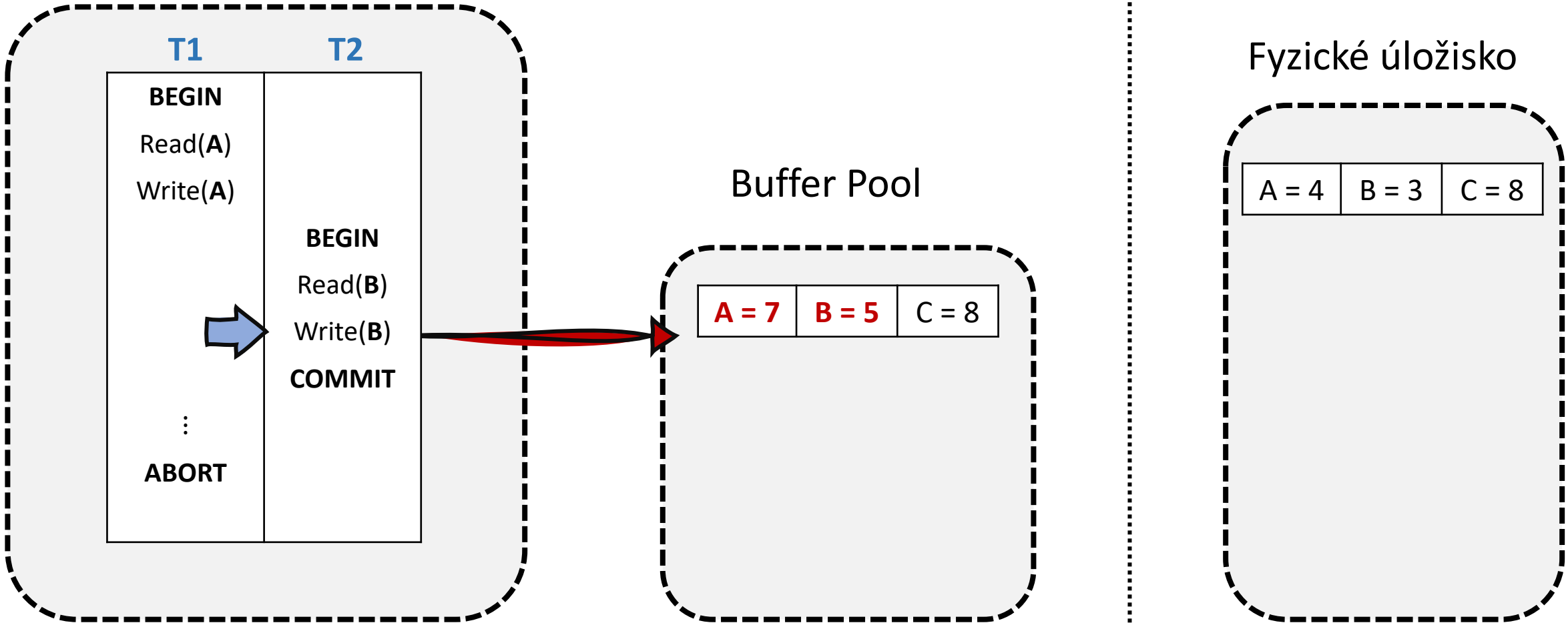


Fyzické úložisko



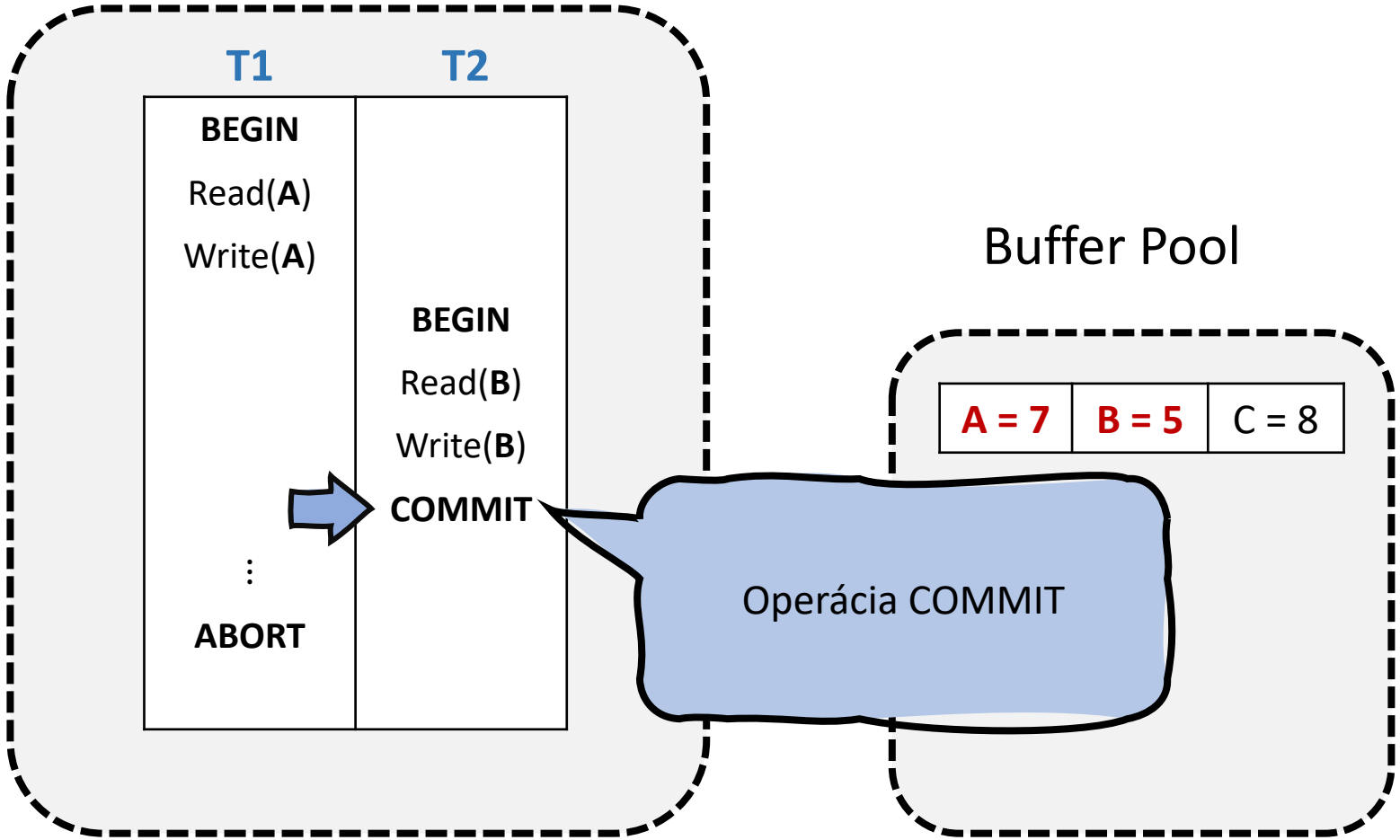
No steal + Force

Rozvrh transakcií

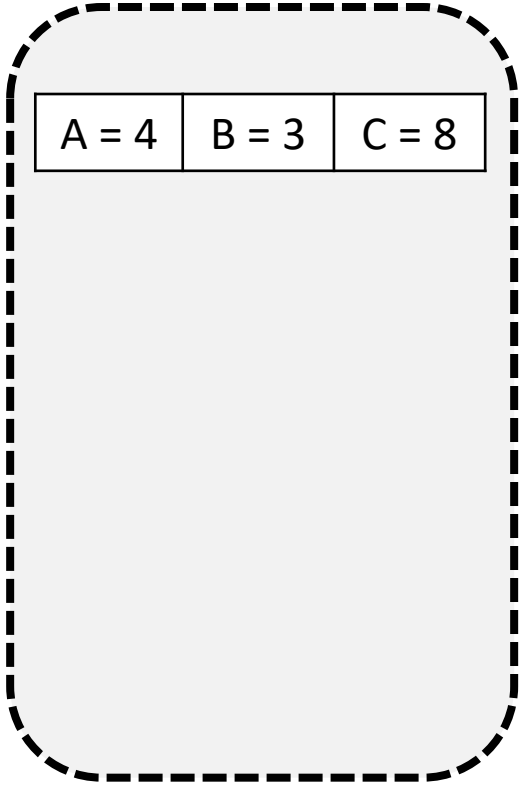


No steal + Force

Rozvrh transakcií

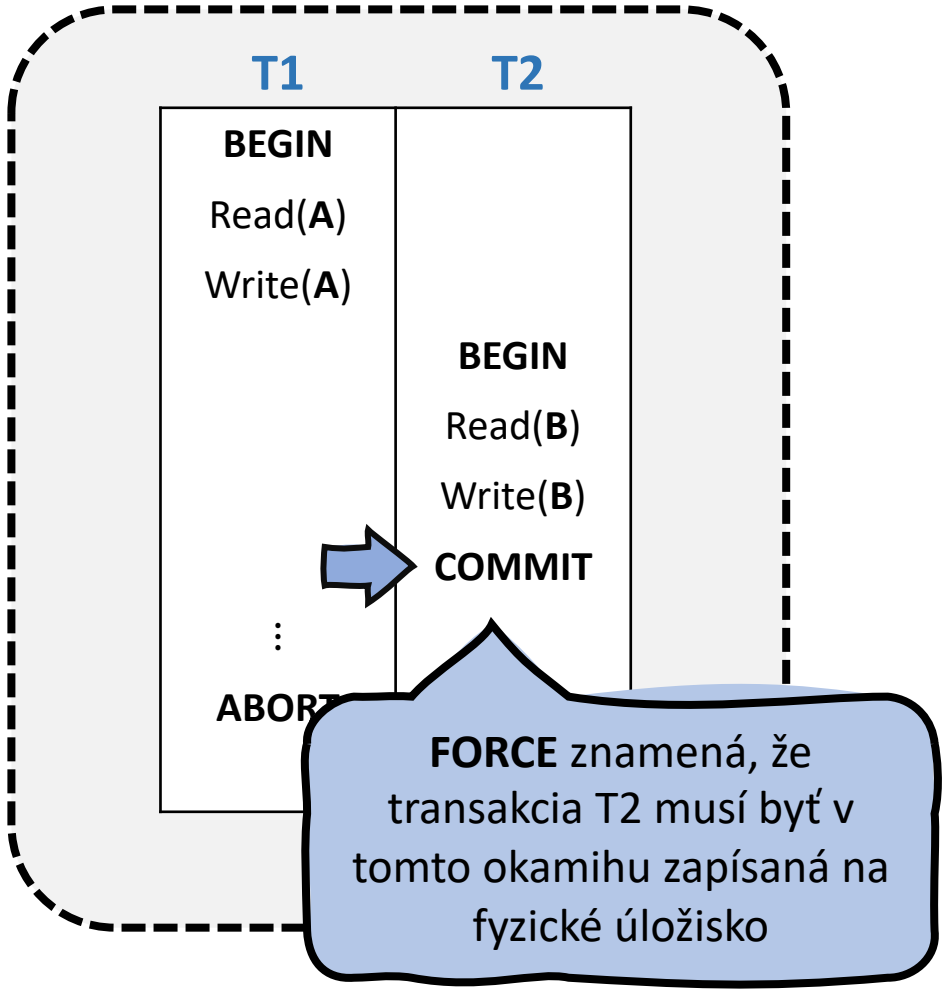


Fyzické úložisko

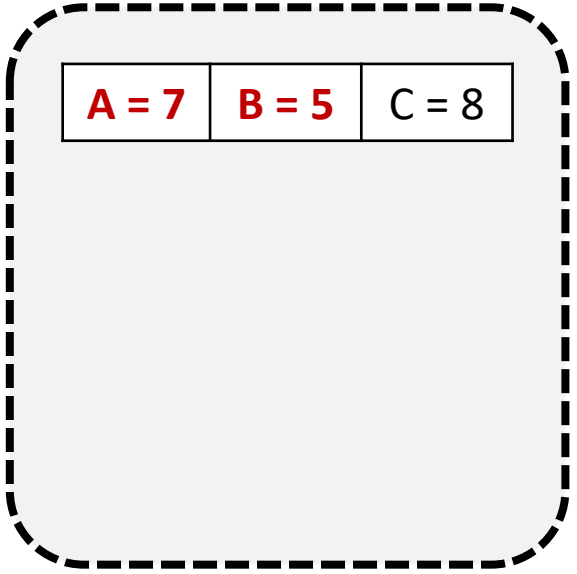


No steal + Force

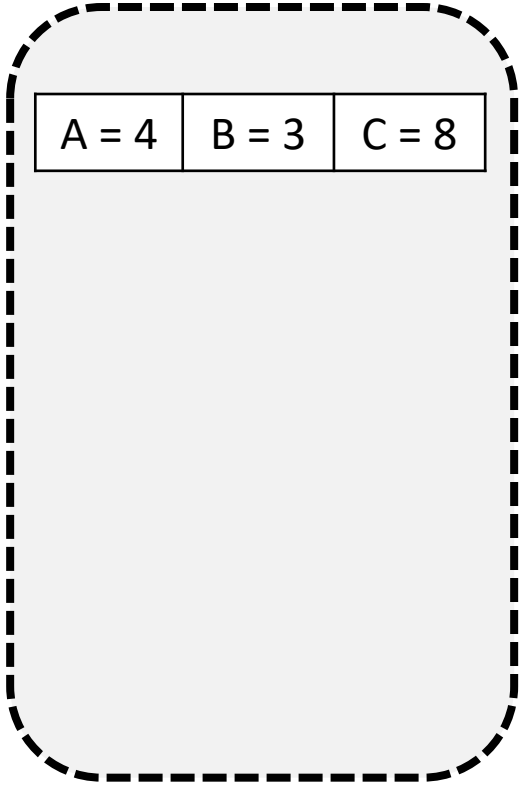
Rozvrh transakcií



Buffer Pool

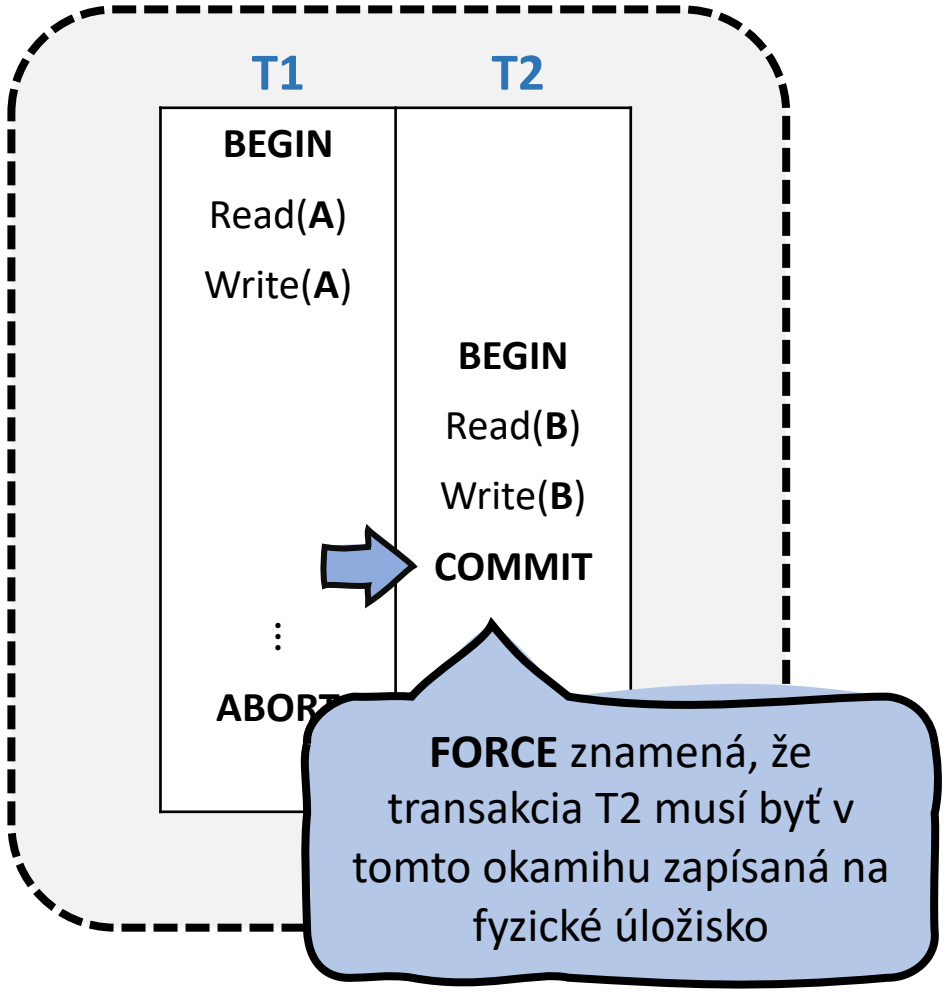


Fyzické úložisko



No steal + Force

Rozvrh transakcií



NO STEAL nám neumožňuje zapísanie hodnoty **A**, ktorá ešte nejde byť COMMIT

Buffer Pool

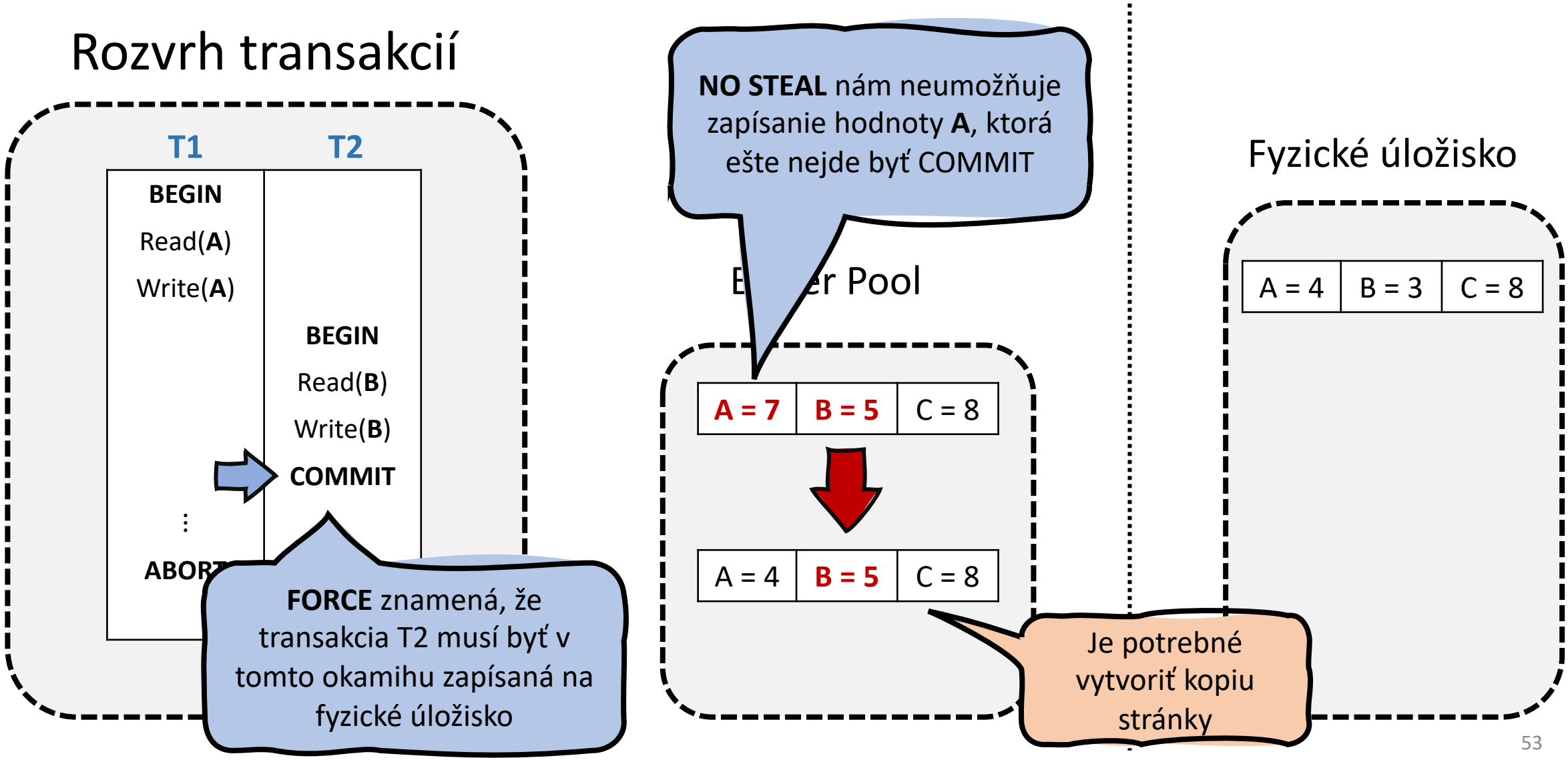
A = 7	B = 5	C = 8

FORCE znamená, že transakcia T2 musí byť v tomto okamihu zapísaná na fyzické úložisko

Fyzické úložisko

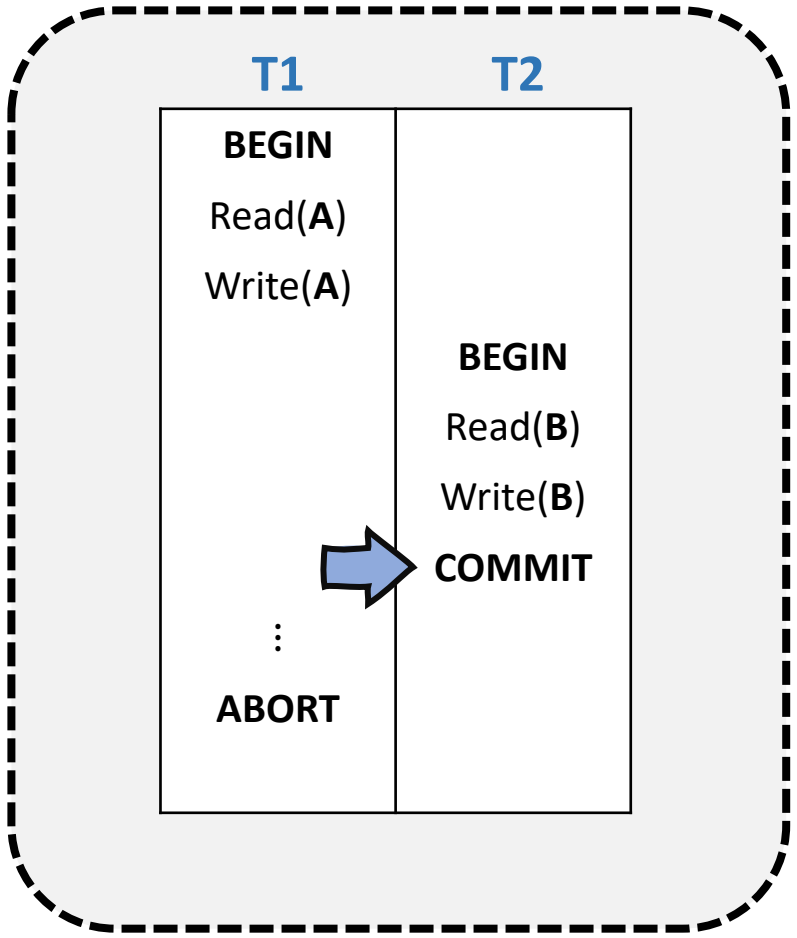
A = 4	B = 3	C = 8

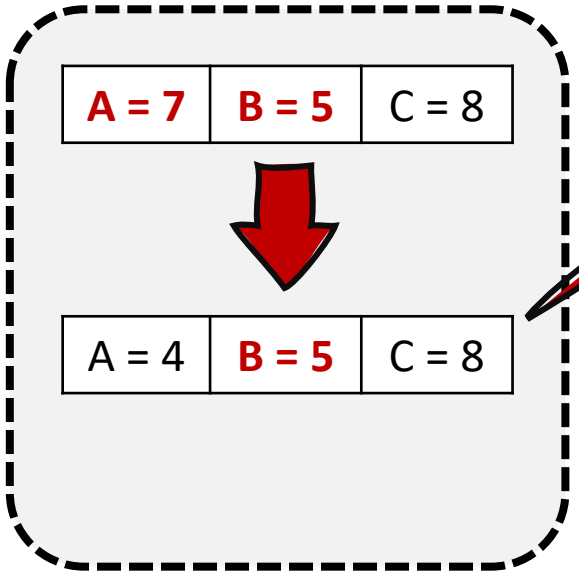
No steal + Force

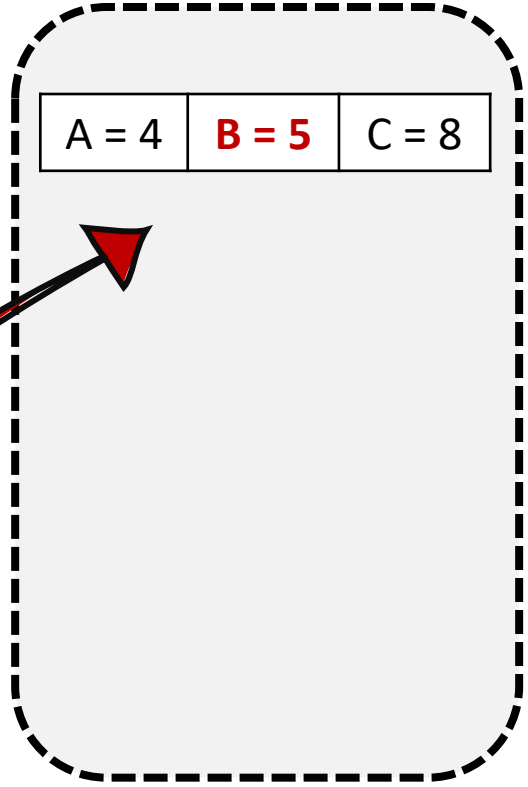


No steal + Force

Rozvrh transakcií

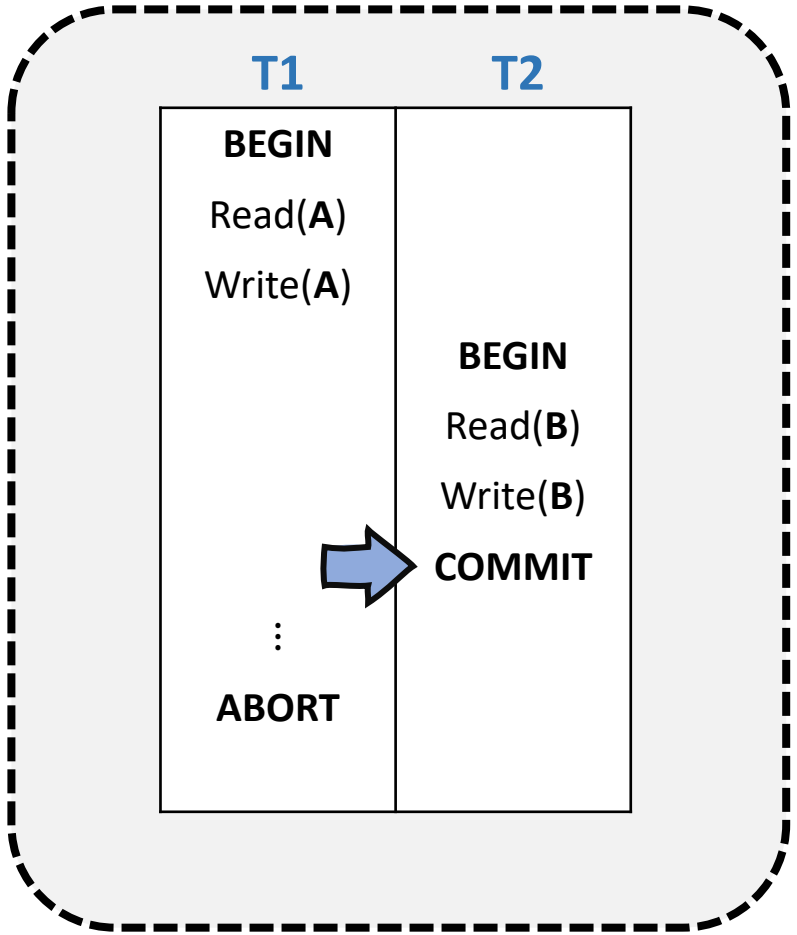




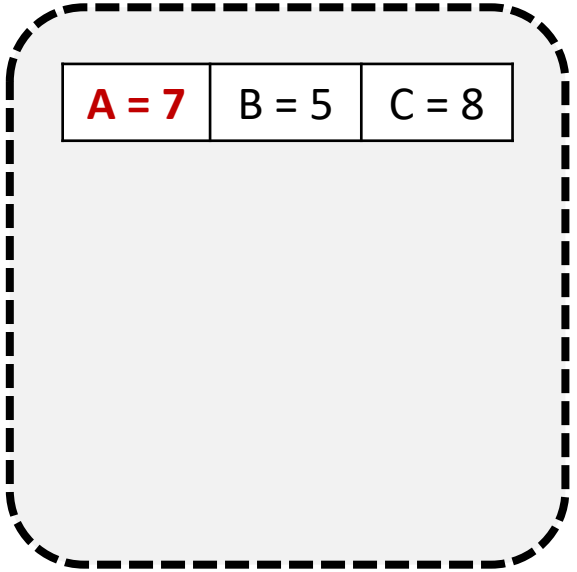


No steal + Force

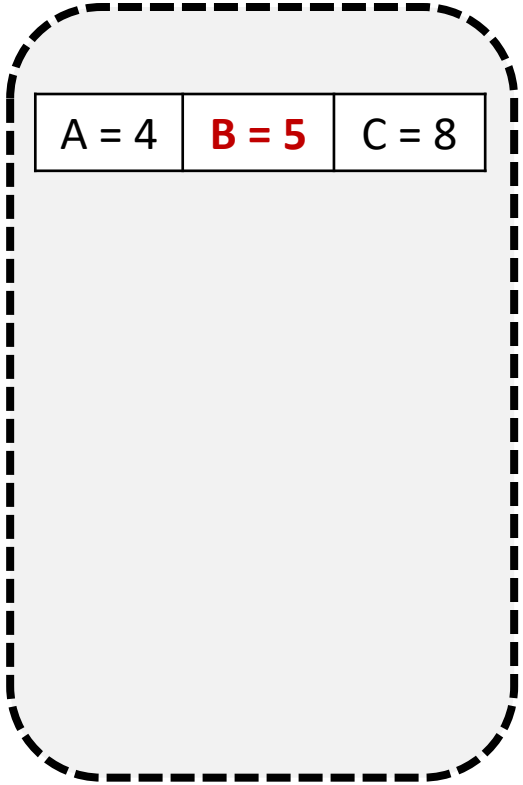
Rozvrh transakcií



Buffer Pool

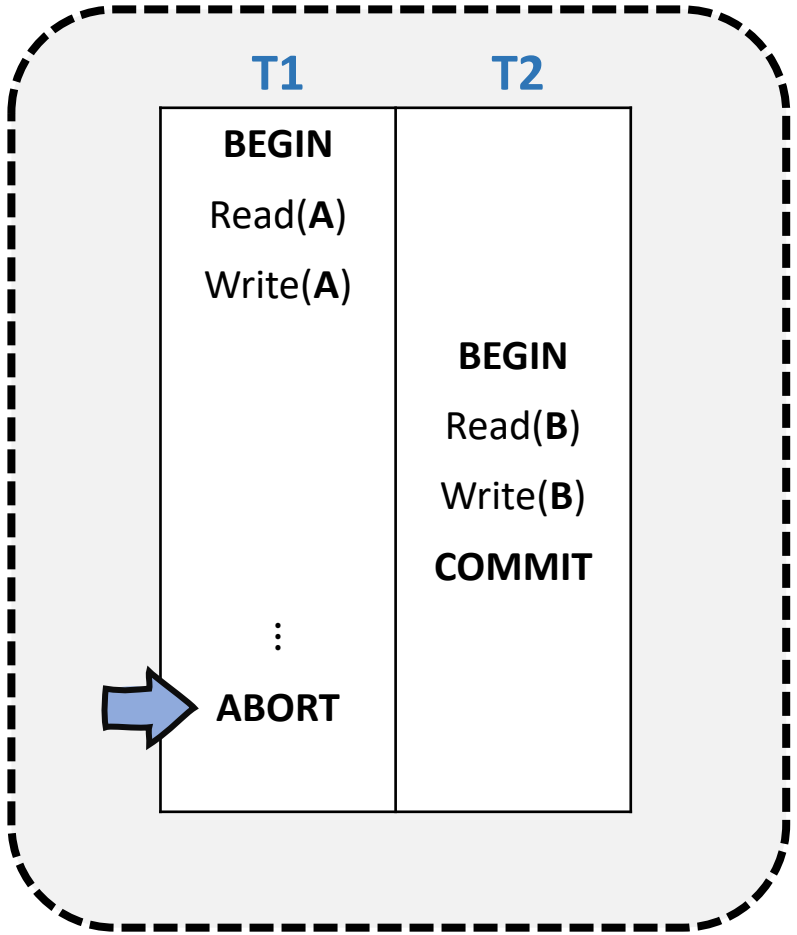


Fyzické úložisko

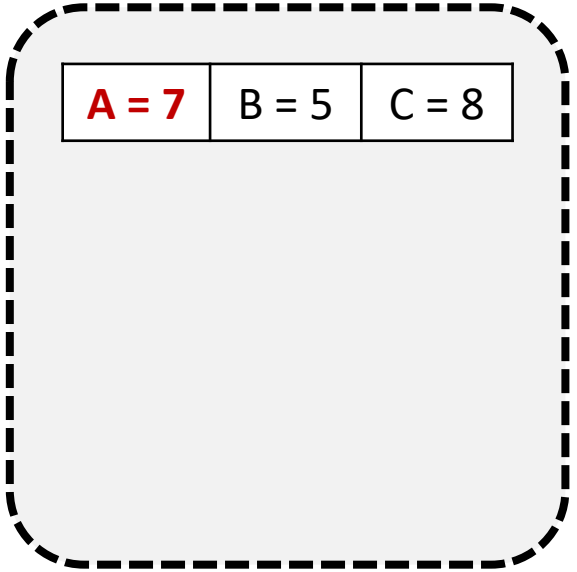


No steal + Force

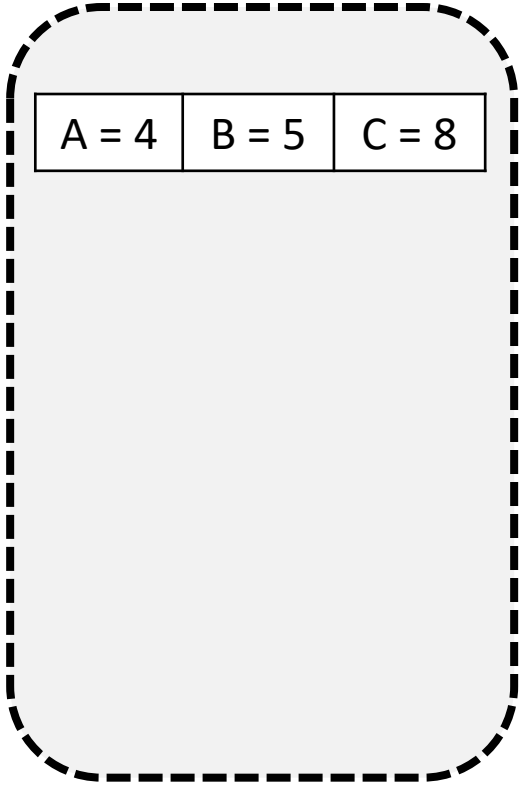
Rozvrh transakcií



Buffer Pool

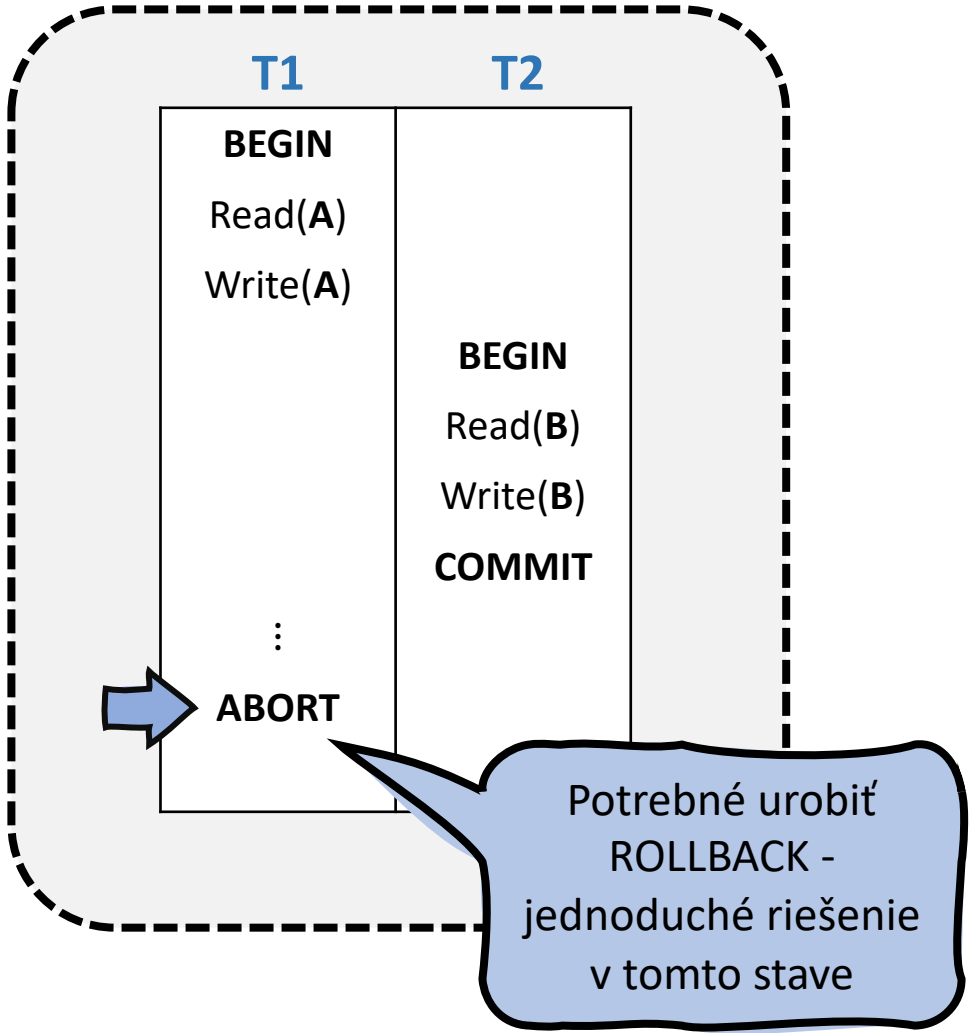


Fyzické úložisko

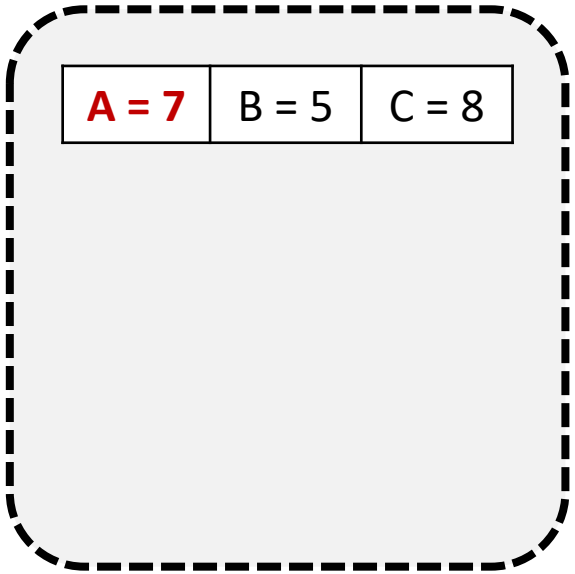


No steal + Force

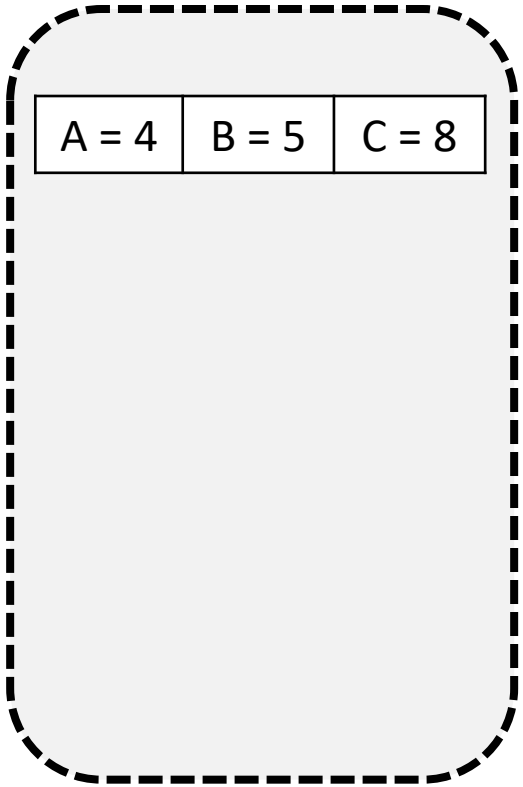
Rozvrh transakcií



Buffer Pool



Fyzické úložisko



No steal + Force

- Ľahká implementácia
  - v prípade zrušenia transakcie (ABORT) nie je potrebné robiť UNDO operácie, pretože zmeny nie sú zapísane na disku
  - v prípade COMMITu nie je potrebné robit REDO operácie, pretože zmeny sú zapísané na disku

- Príkladom No steal + force policy je **Shadow Paging**

Atomicita a durabillity

- **Atomicita**
  - zaručuje nám to, že transakcia je buď vykonaná úplne alebo vôbec
- **Durabillity**
  - zaručuje nám, že dáta prežijú aj prípadný výpadok

- Vieme ich dosiahnuť pomocou dvoch prístupov
  - **Logovanie**
    - log file
  - **Shadow-paging**

Shadow-Paging

- Obsahuje dve kópie stromu Master a Shadow
  - Koreň (root) ukazuje na Master
  - Aktualizácie sú uskutočnené v rámci Shadow

- Pre uplatnenie zmien je potrebné aktualizovať ROOT, ktorý bude ukazovať na shadow zapísane shadow stránky --> stanu sa master

Logovanie

- Logovací súbor (log file) predstavuje sekvenčný súbor, ktorý zaznamenáva zmeny uskutočnené jednotlivými transakciami
  - predpokladá sa, že log file je uložený na stabilnom úložisku (nevolatívna pamäť)
  - garantovanie atomicity diskových prístupov je dôležitá pre log file
  - musí obsahovať všetky potrebné informácie pre uskutočnenie prípadných UNDO a REDO operácií, ktoré súvisia s obnovou DB
- Log file je uložený separátne od súborov s dátami
- Zmeny v log file sa zapíšu okamžite na disk

- **Filozofia WRITE-AHEAD**
  - pred tým ako DBMS zapíše zmeny v DB na disk **musia** byť tieto zmeny zapísane do log file na disku (nestačí aby boli zapísane len v pamäti)
- Buffer pool policy: STEAL + NO-FORCE

Write-Ahead Logging (WAL) protokol

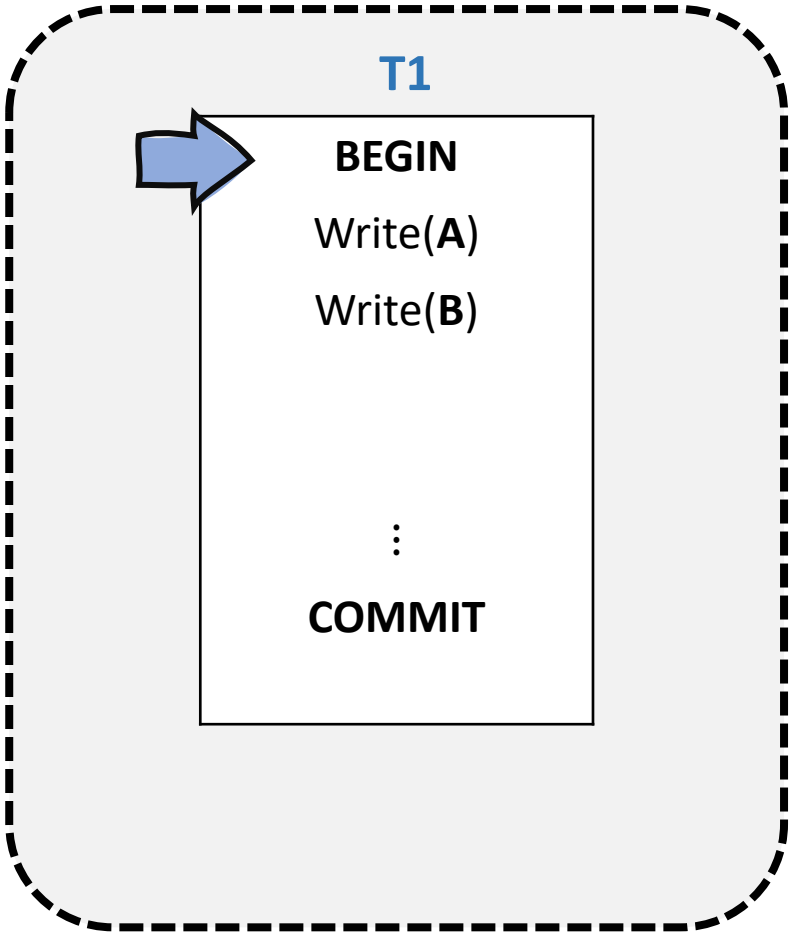
- Jednotlivé logy transakcií sú ukladané DBMS do volatívnej pamäte
- Všetky log záznamy týkajúce sa aktualizovanej stránky sú zapísane do nevolatívnej pamäte **pred tým** ako je samotná stránka zapísaná do nevolatívnej pamäte
- Transakcia nie je považovaná za COMMIT, pokiaľ jej všetky log záznamy nie sú zapísané do nevolatívnej pamäte.
  - informácia pre aplikáciu, že transakcia je COMMIT nastáva až po zapísaní všetkých log záznamov – je potrebné garantovať, že dané dáta sú skutočne aktualizované

Write-Ahead Logging - file

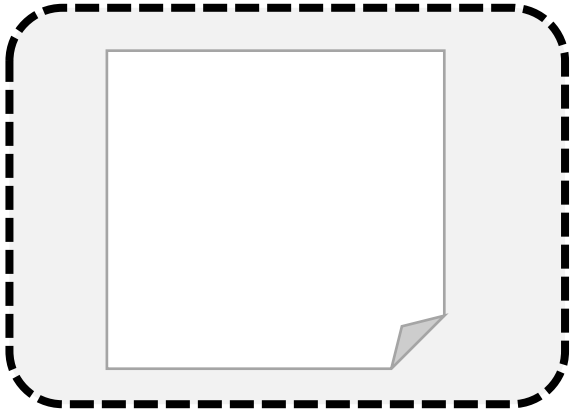
- Záznam log file musí obsahovať
  - identifikáciu transakcie
  - Id objektu, ktorého sa daná zmena týka
  - Hodnotu pred zmenou
  - Hodnotu po zmene
- V rámci záznamov musí byť označené:
  - kedy daná transakcia začala <BEGIN> alebo <START>
  - COMMIT – transakcie <COMMIT>
- Môže obsahovať ďalšie hodnoty záznamy ako napr. CHECKPOINT, DUMP atď.

WAL priklad

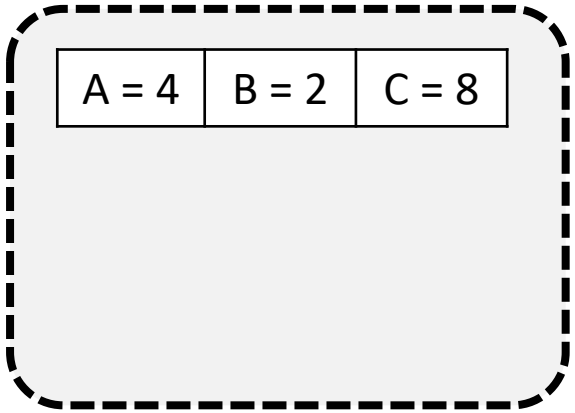
Rozvrh transakcií



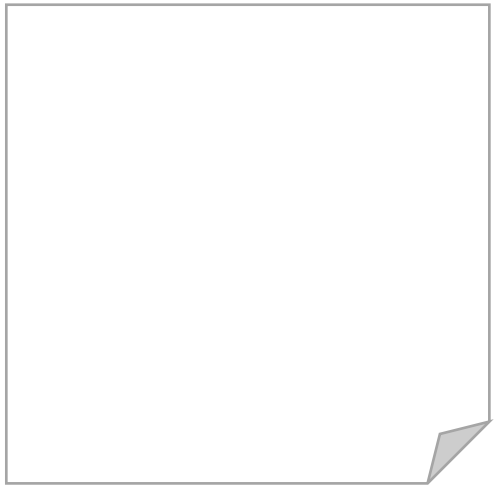
WAL buffer



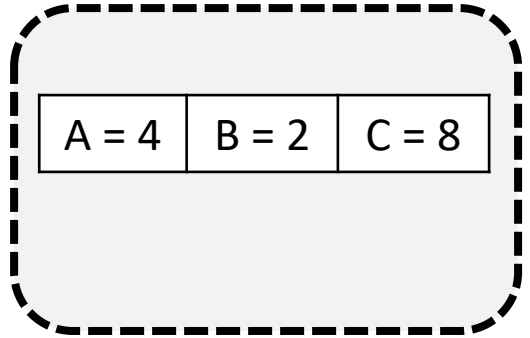
Buffer Pool



Fyzické úložisko

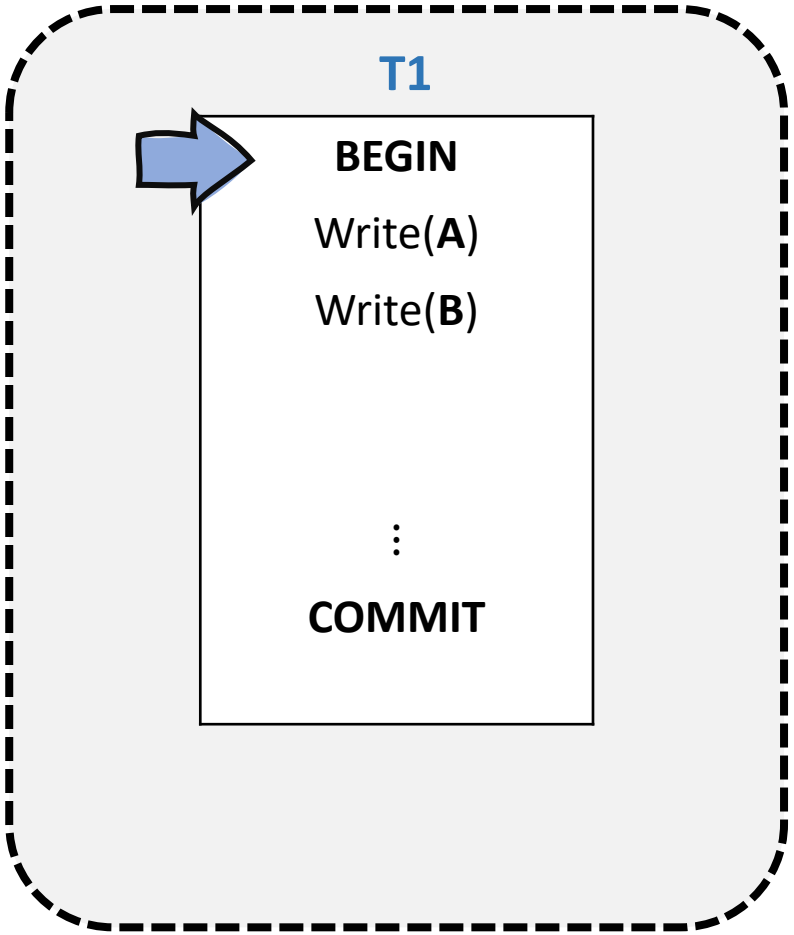


Databáza

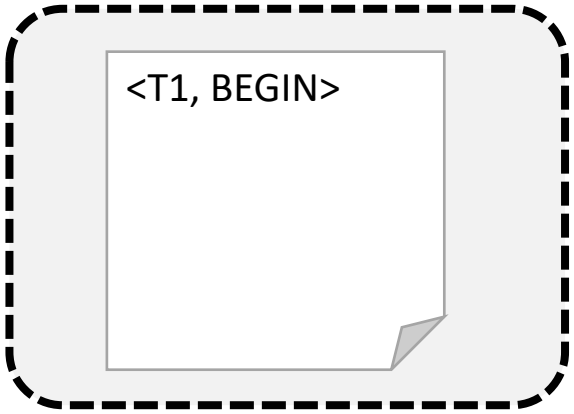


WAL priklad

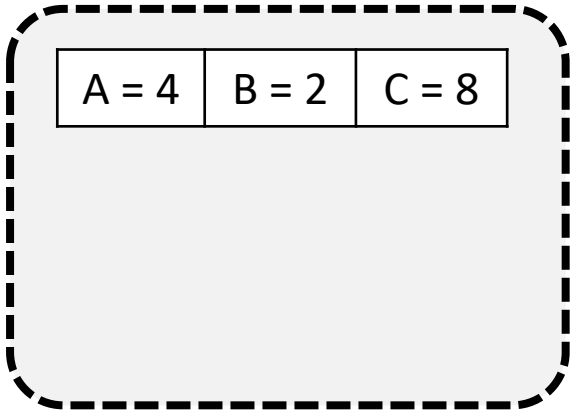
Rozvrh transakcií



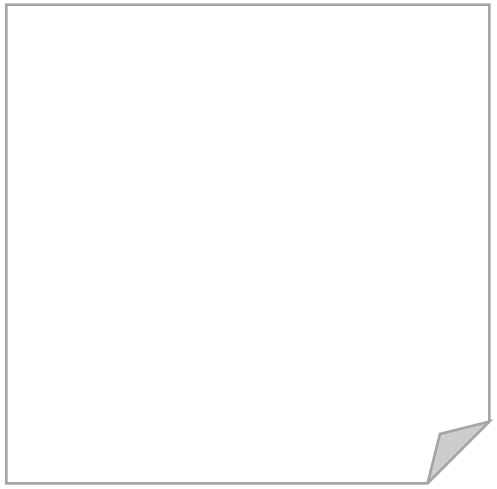
WAL buffer



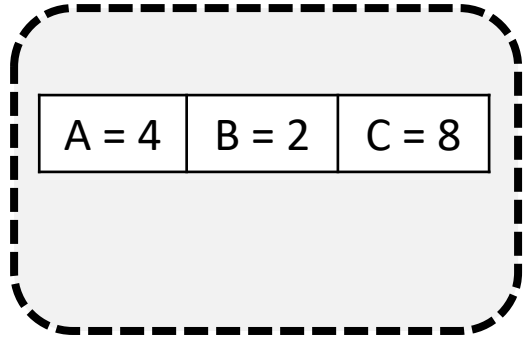
Buffer Pool



Fyzické úložisko

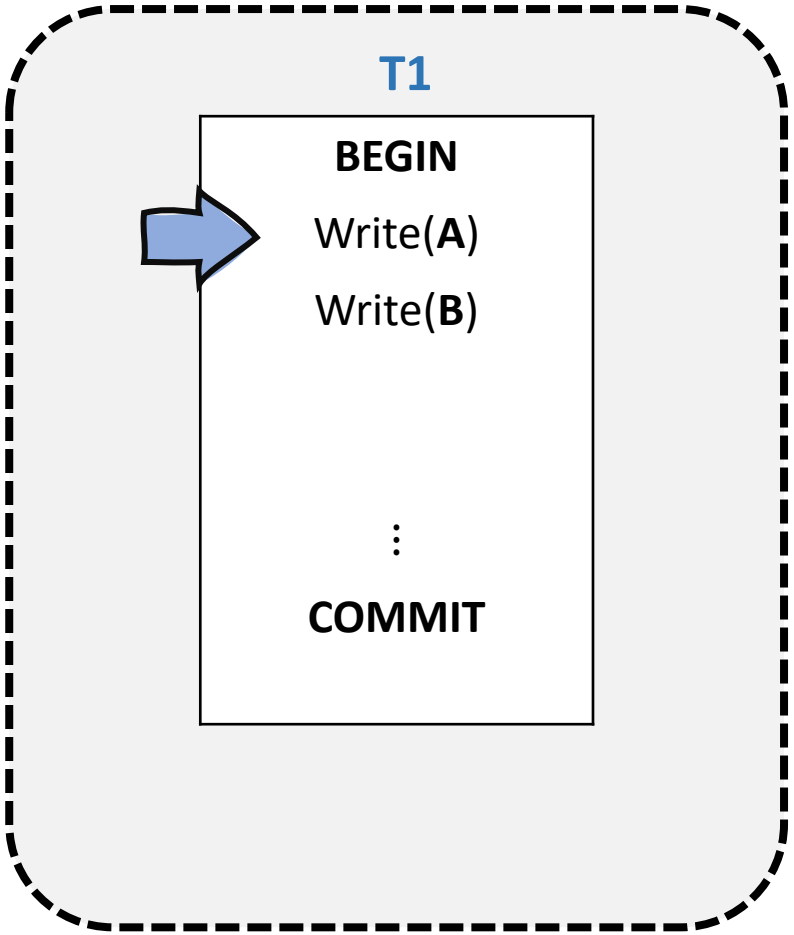


Databáza

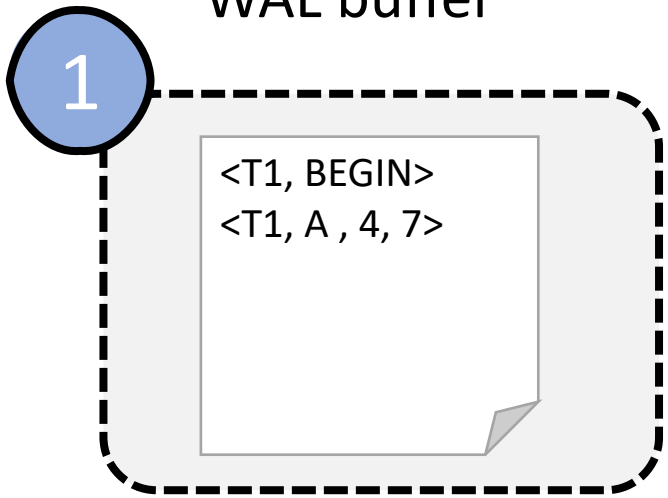


WAL priklad

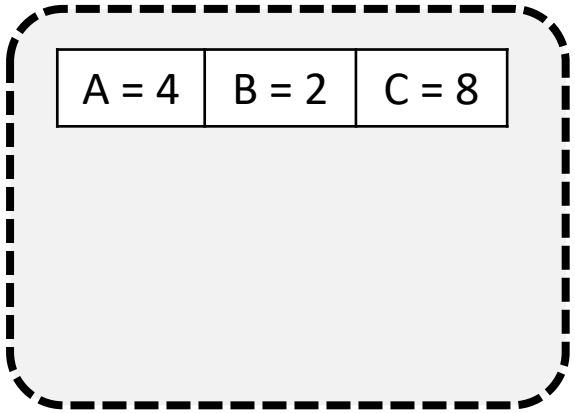
Rozvrh transakcií



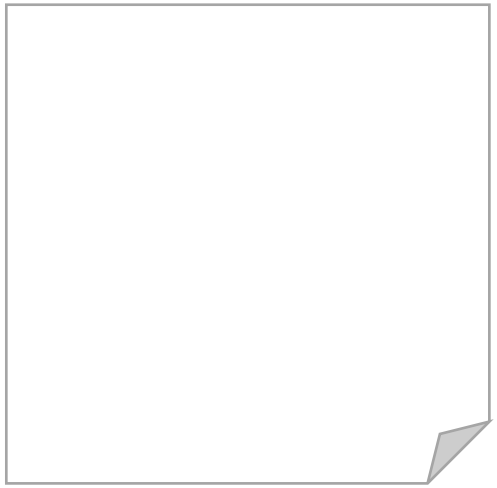
WAL buffer



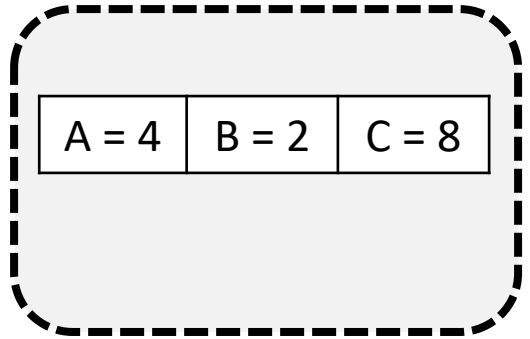
Buffer Pool



Fyzické úložisko

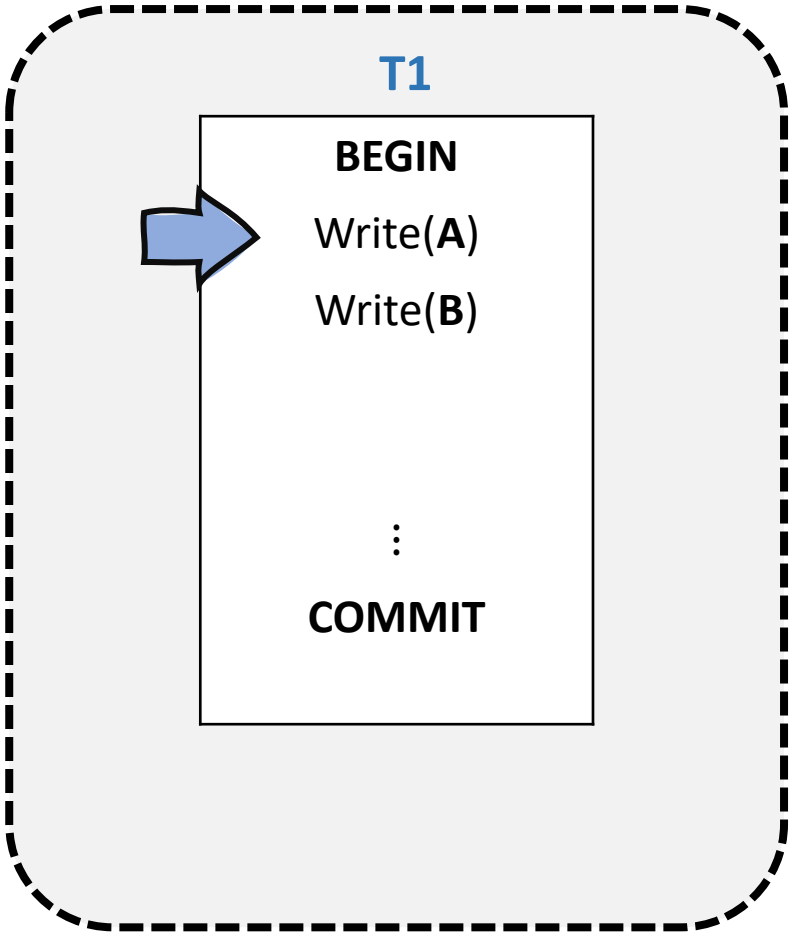


Databáza

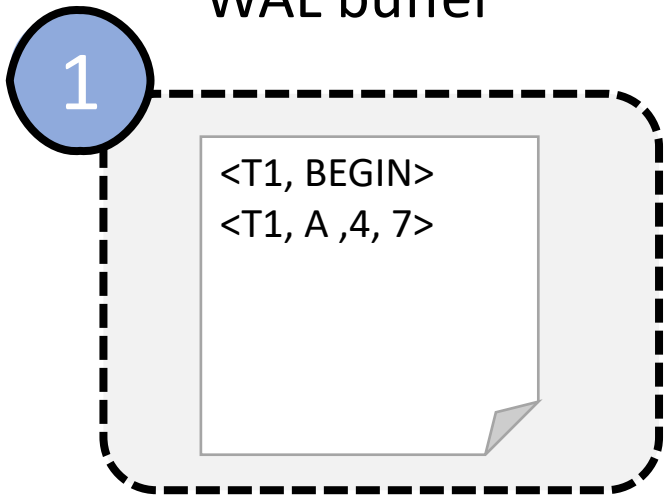


WAL priklad

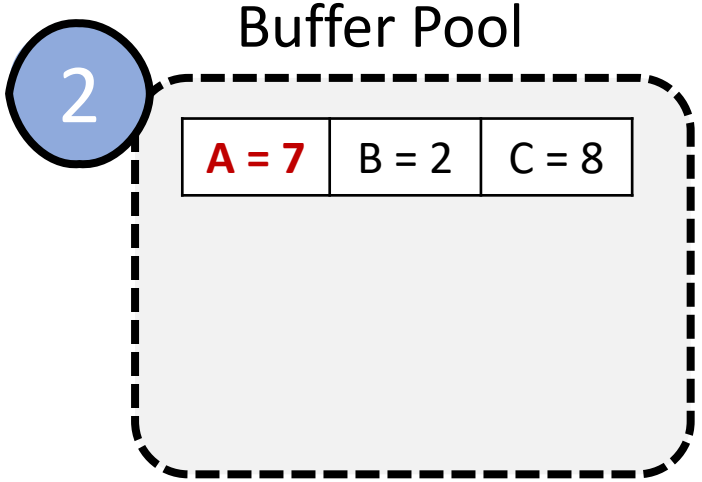
Rozvrh transakcií



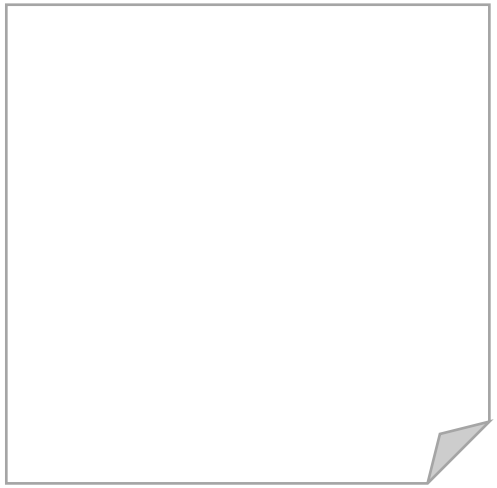
WAL buffer



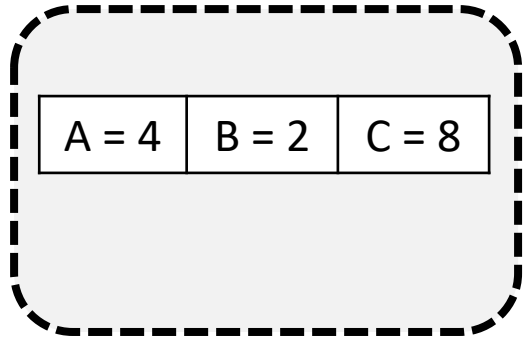
Buffer Pool



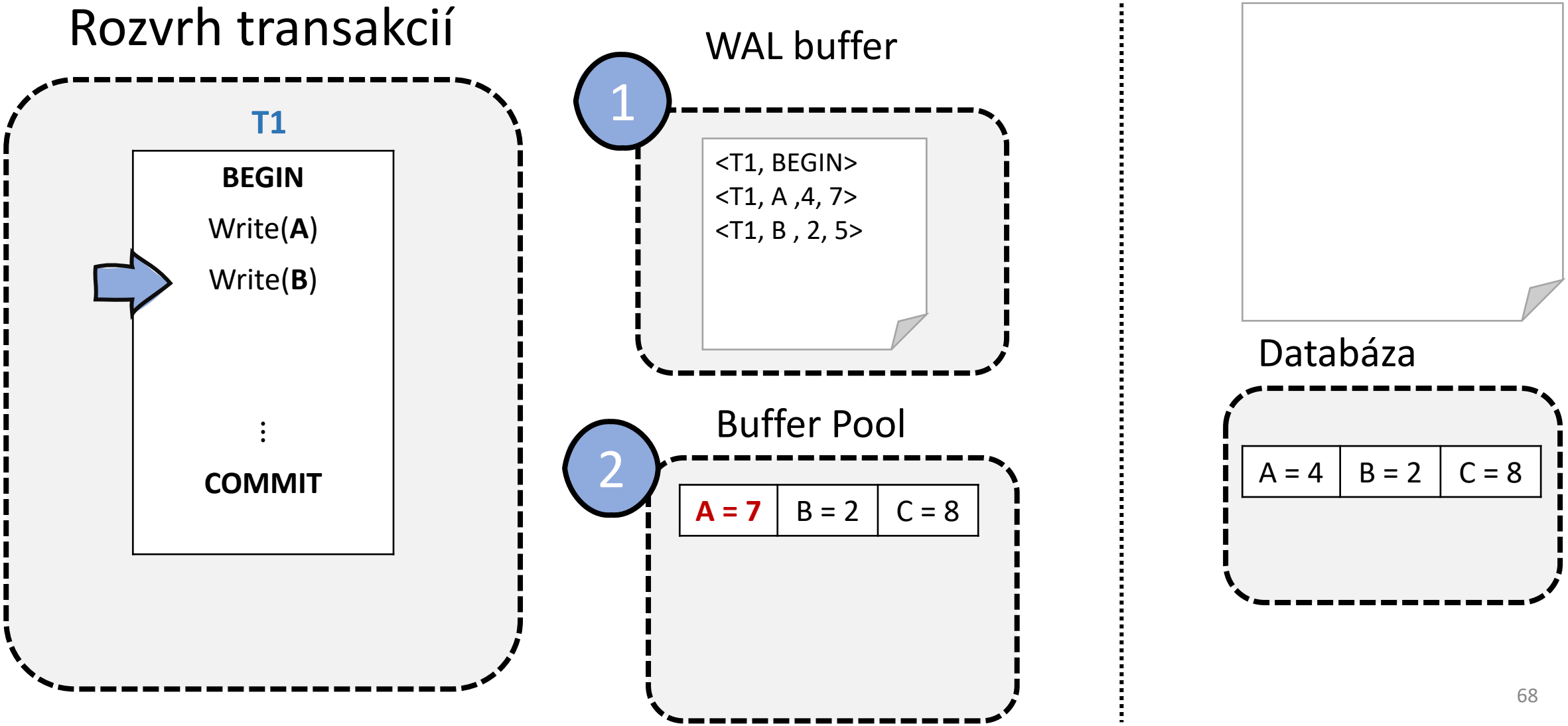
Fyzické úložisko



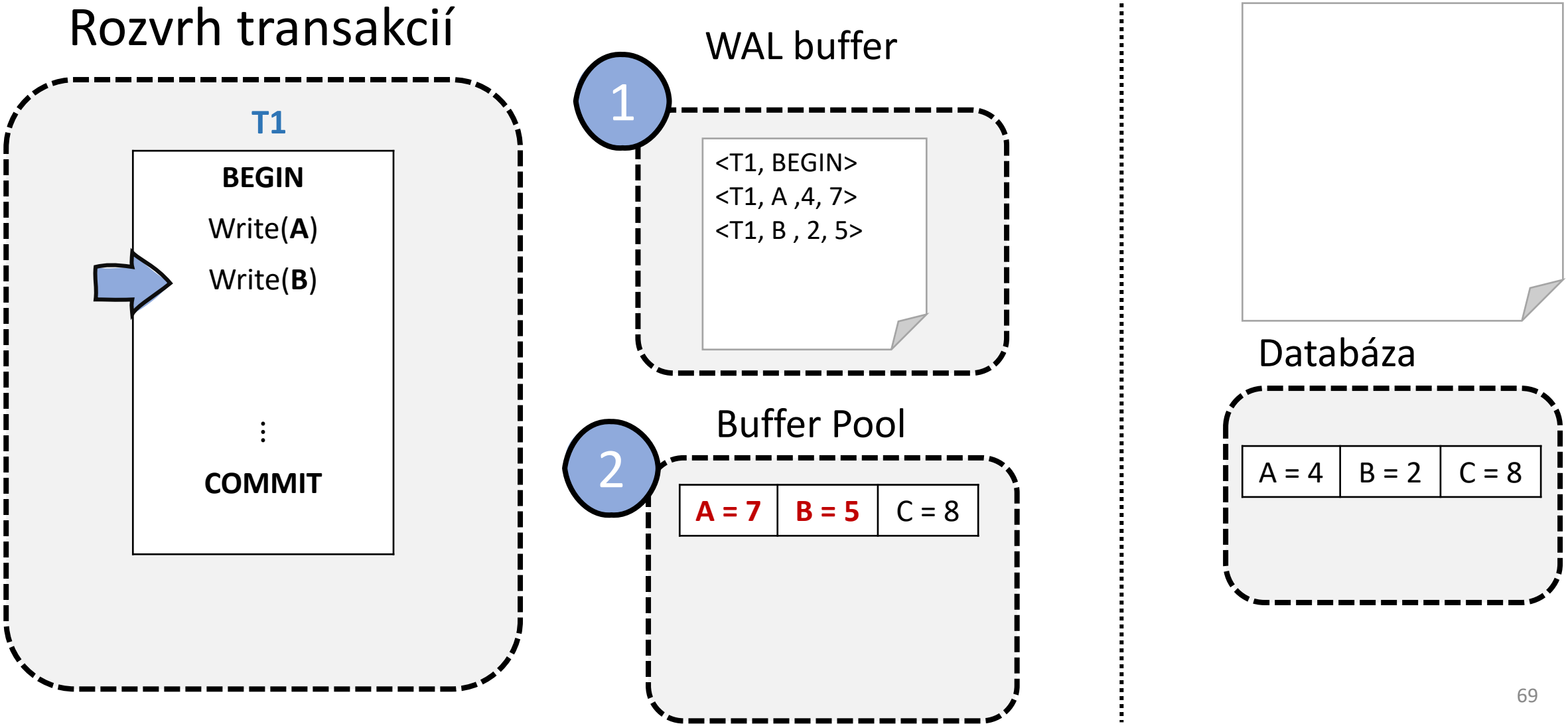
Databáza



WAL priklad

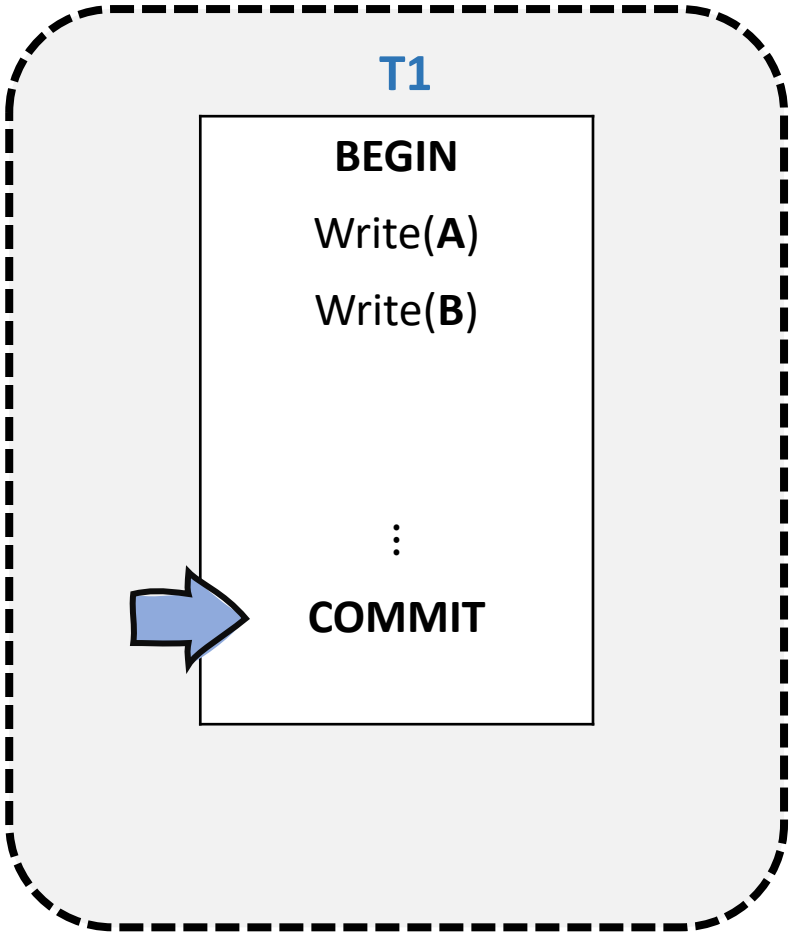

Rozvrh transakcií
T1
BEGIN
Write(A)
Write(B)
⋮
COMMIT
WAL buffer
1
<T1, BEGIN>
<T1, A ,4, 7>
<T1, B , 2, 5>
Buffer Pool
2
A = 7
B = 2
C = 8
Fyzické úložisko
Databáza
A = 4
B = 2
C = 8

WAL priklad


Rozvrh transakcií
T1
BEGIN
Write(A)
Write(B)
COMMIT
WAL buffer
1
<T1, BEGIN>
<T1, A ,4, 7>
<T1, B , 2, 5>
Buffer Pool
2
A = 7
B = 5
C = 8
Fyzické úložisko
Databáza
A = 4
B = 2
C = 8

WAL priklad

Rozvrh transakcií



WAL buffer

1

<T1, BEGIN>
<T1, A ,4, 7>
<T1, B , 2, 5>
<T1, COMMIT>

Buffer Pool

2

A = 7	B = 5	C = 8

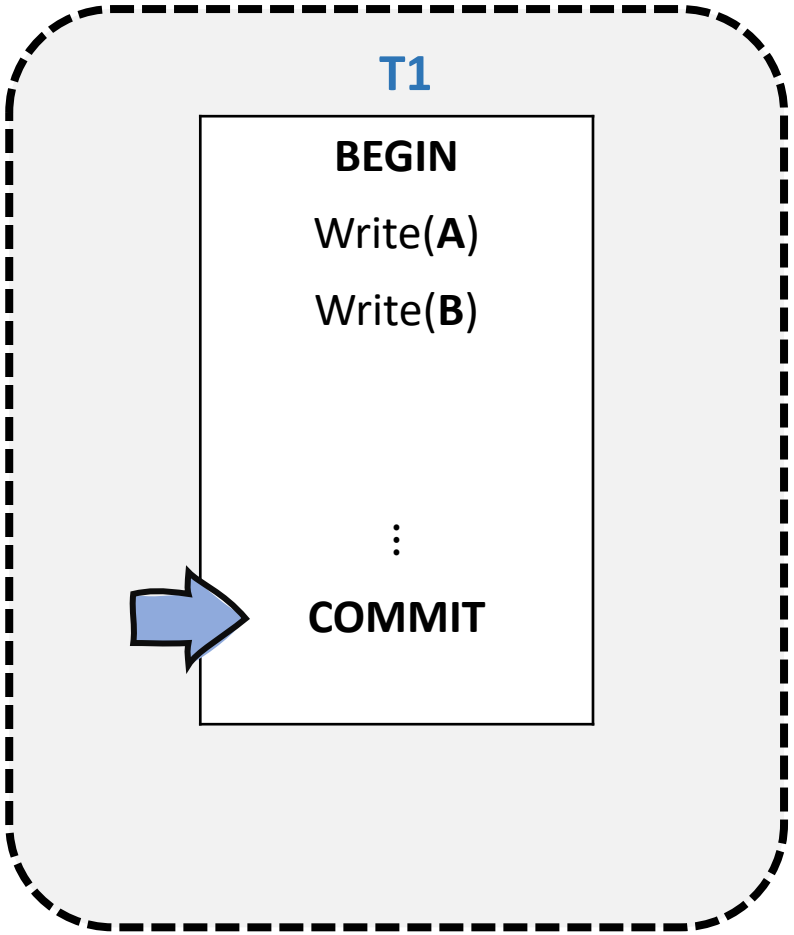
Fyzické úložisko

Databáza

A = 4	B = 2	C = 8

WAL priklad

Rozvrh transakcií



WAL buffer

1

<T1, BEGIN>
<T1, A ,4, 7>
<T1, B , 2, 5>
<T1, COMMIT>

Buffer Pool

2

A = 7	B = 5	C = 8

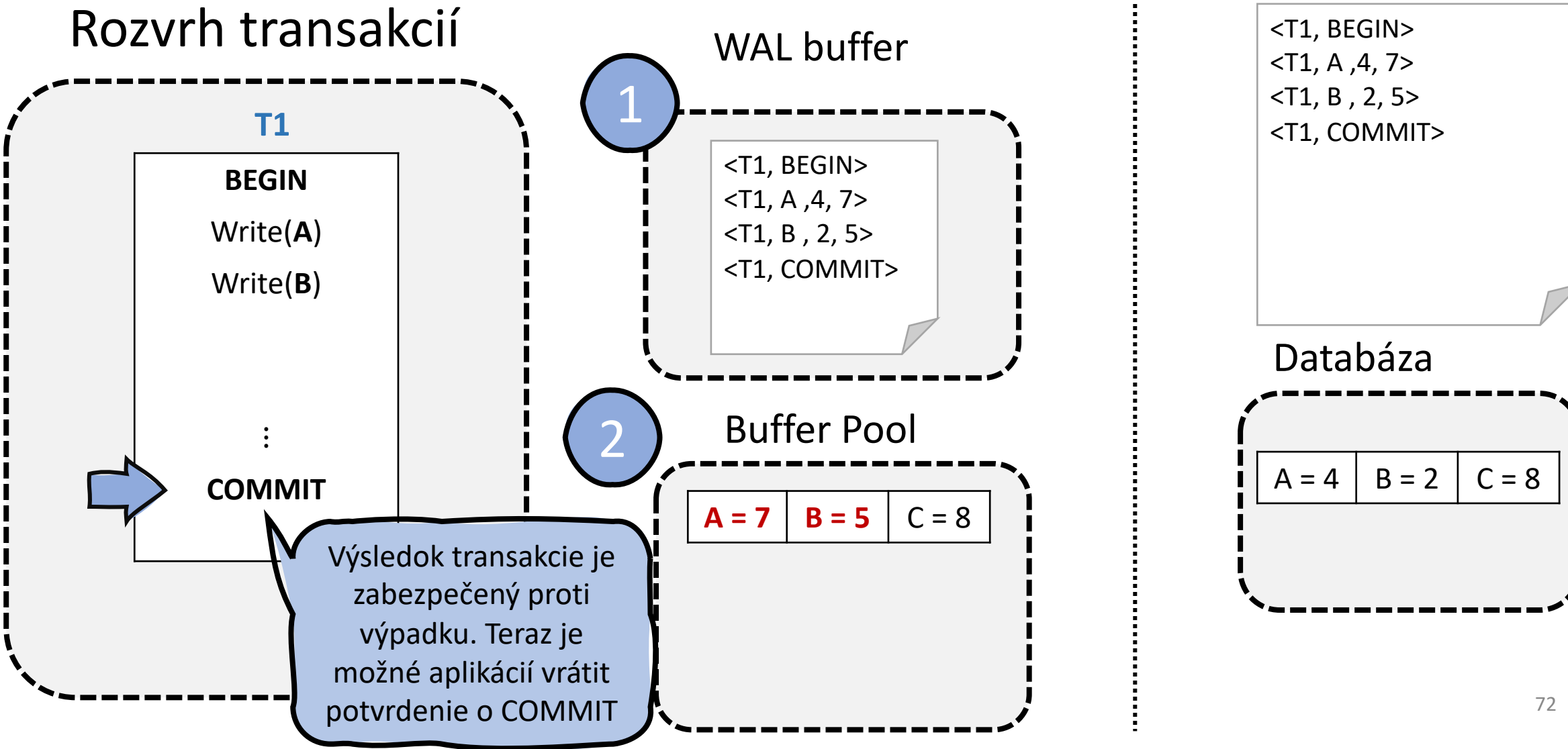
Fyzické úložisko

<T1, BEGIN>
<T1, A ,4, 7>
<T1, B , 2, 5>
<T1, COMMIT>

Databáza

A = 4	B = 2	C = 8

WAL priklad


Rozvrh transakcií
T1
BEGIN
Write(A)
Write(B)
COMMIT
Výsledok transakcie je zabezpečený proti výpadku. Teraz je možné aplikácií vrátit potvrdenie o COMMIT
1
WAL buffer
<T1, BEGIN>
<T1, A ,4, 7>
<T1, B , 2, 5>
<T1, COMMIT>
2
Buffer Pool
A = 7
B = 5
C = 8
Fyzické úložisko
<T1, BEGIN>
<T1, A ,4, 7>
<T1, B , 2, 5>
<T1, COMMIT>
Databáza
A = 4
B = 2
C = 8

WAL implementácia

- Kedy má DBMS zapísať log file na disk ?

- Kedy má DBMS zapisať dirty záznamy na disk ?

WAL implementácia

- Kedy má DBMS zapísať log file na disk ?
  - keď transakcia robí COMMIT.
  - možnosť zápisu viacerých log záznamov na disk naraz – zrýchlenie výkonu systému

- Kedy má DBMS zapisať dirty záznamy na disk ?
  - Pri každej aktualizáci objektu alebo len raz pri COMMIT ?

Checkpoints

- Log file rastie donekonečná
  - nie ideálne riešenie do praxe
  - dlhý recovery time
- Po výpadku je potrebné prejsť cely log file a uskutočniť jednotlivé zmeny, ktoré vykonali jednotlivé transakcie

- Checkpoint
  - pomáhajú redukovať dĺžku log-filov a skracujú čas potrebných na obnovu systému

Checkpoint

- Uvažuje o atomicku procedúru, ktorá obsahuje nasledujúce časti:
  - Zapíše všetky log záznamy na disk (nevolatívna pamäť)
  - Zapíše všetky dáta na disk
  - zapíš **Checkpoint** do log file – môžu sa zapísať aj ďalšie informácie napr. zoznam aktívnych transakcií
  - skrať log-file – garbage collector
    - skratí sa o časť, ktorá sa nachádza pred najstaršou aktívnou transakciou – pred jej BEGIN

Algoritmus obnovy

- Je závislý od
  - použitého concurrency control protokolu
  - použitej techniky zápisu na disk

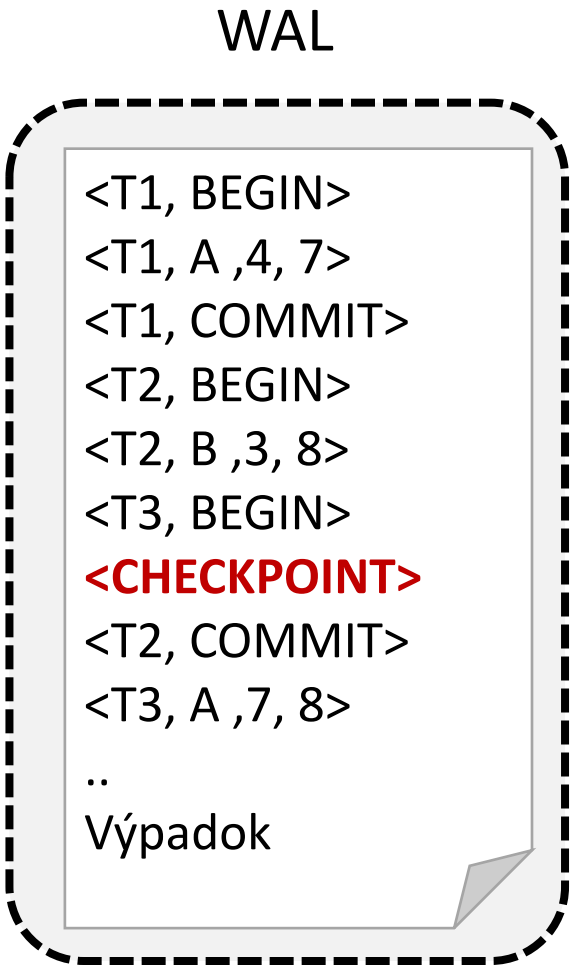
Checkpoint - algoritmus obnovy

Všeobecná metóda obnovy sa uskutočnuje v nasledujúcich krokoch:

1. prechádzajú sa záznamy zostupne (od posledného po začiatok) a vytvára sa UNDO a REDO list. Zároveň sa uskutočňuje UNDO operácia
2. Po dosiahnuti záznam CHECKPOINT sa pokračuje zostupne až po prejdenie všetkých aktívnych transakcií v čase CHECKPOINTU
3. Spusti sa REDO fáza vzostupne od posledného záznamu najdenom v kroku 1

Checkpoint priklad

- Transakcie s COMMIT pred CHECKPOINT sú ignorované (T1)
- T2 a T3 neprebehol COMMIT pred CHECKPOINT a je preto treba urobiť
  - REDO operáciu pre T2 (COMMIT po CHECKPOINT)
  - UNDO pre operáciu T3 – nebol COMMIT pred výpadkom



ARIES

- Algorithms for Recovery and Isolation Exploiting Semantics
- Vynájdený v roku 1990 spoločnostou IBM pre ich DB2 DBMS
- V súčasnosti sa využíva stále tento algoritmus aj keď nie úplne presne v rovnakej podobe

ARIES - fázy

- **Analýza** – čítanie WAL od posledného záznamu pre identifikovanie dirty pages a aktívnych transakcií v čase výpadku
- **REDO** – opakovanie akcií pre vrátenie stavu pred výpadkom.
- **UNDO** – vrátenie všetkých hodnôt transakcií, ktorým neprebehol COMMIT do času výpadku

Teoria obnoviteľnosti

- naspäť k rozvrhom z predchádzajúcej prednášky

- Seriovateľnosť rozvhov **nie je dostačujúca** pre možnosti obnovy

Príklad

- **Rozvrh S1:** **r1(A), w1(A), r2(A), w2(C), c2, c1**

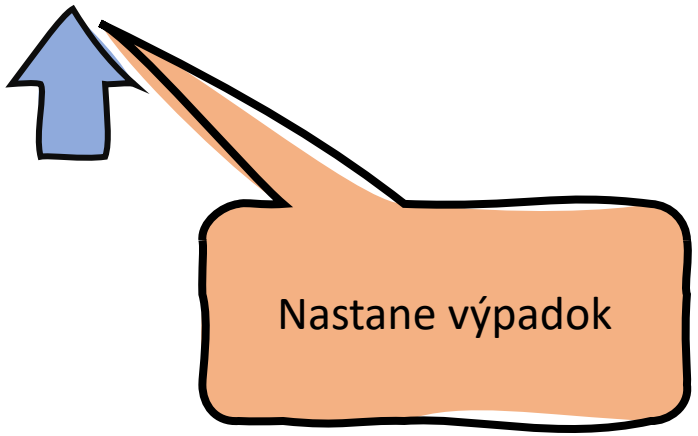
Teoria obnoviteľnosti

- naspäť k rozvrhom z predchádzajúcej prednášky

- Seriovateľnosť rozvhov **nie je dostačujúca** pre možnosti obnovy

Príklad

- Rozvrh S1: **r1(A), w1(A), r2(A), w2(C), c2, c1**



Teoria obnoviteľnosti

- **Rozvrh S1: r1(A), w1(A), r2(A), w2(C), c2, c1**
  - COMMIT transakcie T2 (označený ako C2) je navždy zapísaný v DB
  - T2 čítala hodnotu od T1, ktorá však nebola zapísana do DB v dôsledku výpadku
  - **Čo ak hodnota C závisela od hodnoty A ?**

Dirty read

- čítanie hodnoty zapísanej inou transakciou predtým ako prebehol COMMIT transakcie, ktorá modifikovala danú hodnotu

Obnoviteľný rozvrh

- Rozvrh je obnoviteľný (RC) práve vtedy, keď pre každú COMMIT transakciu T2 čítajúcu nekomitovanú hodnotu od T1 (označuje sa ako DIRTY READ) platí, že T1 commituje tiež a zároveň T1 COMMIT je v danom rozvrhu skôr ako T2 COMMIT.
- Predchádzajúci rozvrh S1 **nie je obnoviteľný**

Príklad

- rozvrh S2: w1(X) r2(X) w2(Y) c1 c2
- je obnoviteľný

- Problém rozvrhu S2 je hroziaci kaskádový abort (cascading abort) v prípade, že T1 sa rozhodne pre ABORT, tak T2 bude musieť skončiť ako ABORT tiež (Ak ma byť tento rozvrh považovaný za obnoviteľný)

Obnoviteľný rozvrh

Príklad

- **rozvrh S2**: **w1(X), r2(X), w2(Y), c1, c2**
- **je obnoviteľný**

- Problém rozvrhu **S2** je hroziaci **kaskádový abort** (cascading abort) v prípade, že T1 sa rozhodne pre ABORT, tak T2 bude musieť skončiť ako ABORT tiež (Ak ma byť tento rozvrh považovaný za obnoviteľný)

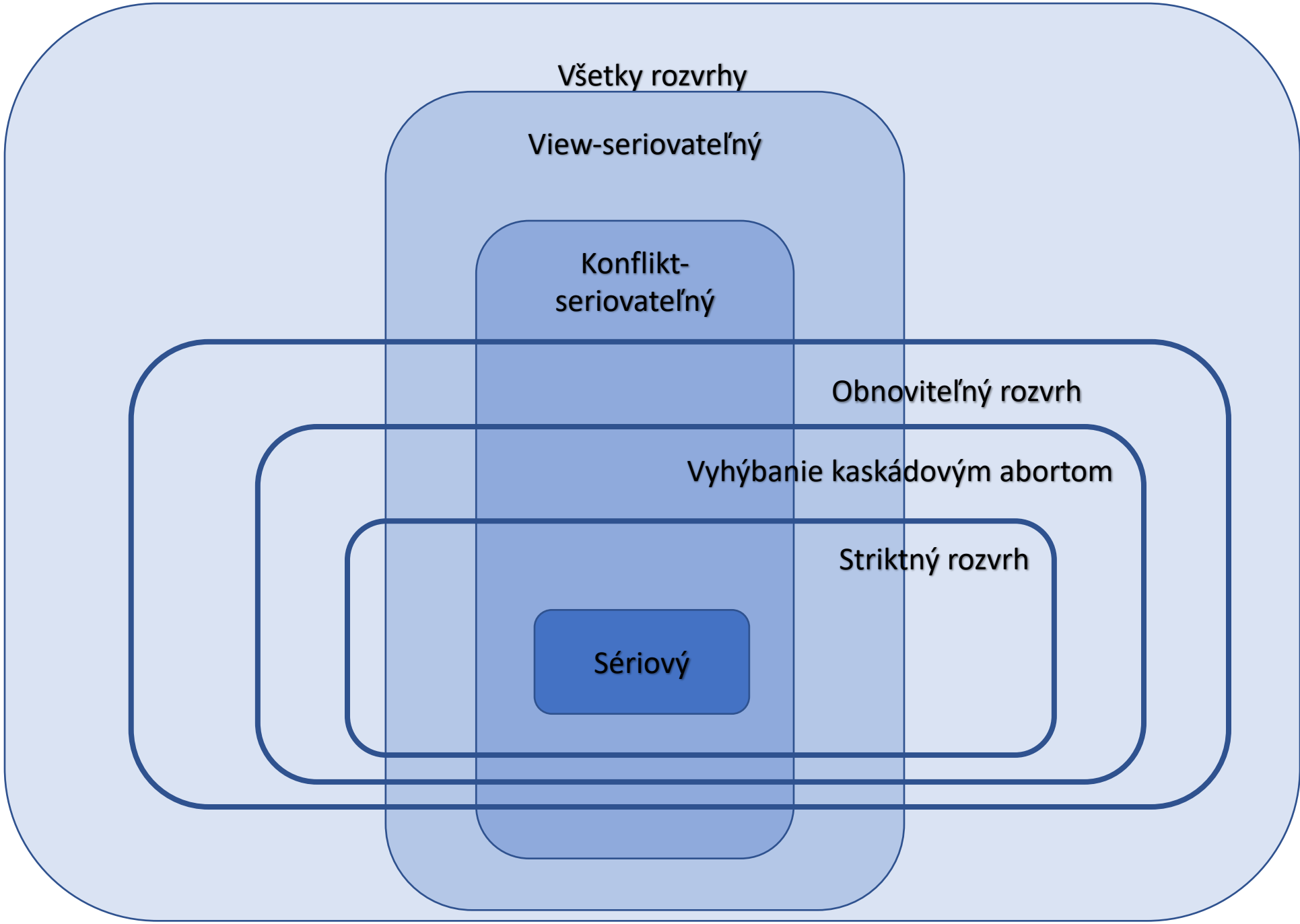
Kaskadový abort - priklad

- Máme rozvrh S3:
  - **r1(x),w1(x),r2(x),w2(y),r3(y),w3(z), Abort1**

- Rozvrh, ktorý sa vyhýba kaskádovym abortom (Avoids Cascading Aborts), **práve vtedy keď neobsahuje dirty read**

- **Dirty read** – čítanie hodnoty zapísanej inou transakciou predtým ako prebehol COMMIT transakcie, ktorá modifikovala danú hodnotu

Rozvrhy


Všetky rozvrhy
View-seriovateľný
Konflikt-
seriovateľný
Obnoviteľný rozvrh
Vyhýbanie kaskádovým abortom
Striktný rozvrh
Sériový

Striktne obnoviteľný rozvrh

- Striktný rozvrh – je taký rozvrh, ktorý neobsahuje **dirty reads ani dirty write**
- **Dirty write**
  - prepis hodnoty, ktorá nebola COMMIT inou transakciou
- Takýto rozvrh umožnuje rýchlejšiu obnovu dát – nie je nutné uskutočňovať REDO operáciu (nemusí sa druhykrát prechádzať log)
- Takýto rozvrh vie generovať striktný 2PL

Zhrnutie

- Logovanie je potrebné pre zachovanie ACID a tiež pre možnosti obnovy dát pri výpadku
- Write-Ahead logovanie je vo väčšine prípadov výhodnejšie (Steal + No Force)
- CHECKPOINT – zrýchľuje obnovu a tiež skracuje log-file
- Pre obnovu sú potrebné operácie REDO a UNDO
  - v závislosti od spôsobu zapisovania