Manažment transakcií

Kapitola 18

Transakcie

- Tvorí základ pre konkurentné vykonávanie a zotavenie sa po zlyhaní systému
- Je definovaná ako ľubovoľné vykonanie používateľského programu v DBMS
- **Transakcia** je postupnosť čítaní a zápisov
- 4 dôležité vlastnosti, ktoré transakcia musí mať (kvôli konkurentným prístupom a možnému zlyhaniu systému):
  - **A**tomic - Atomickosť (buď sa vykoná celá alebo sa databáza musí dostať do stavu, akoby sa nevykonala vôbec)
  - **C**onsistency - Konzistentnosť (po transakcii musí byť DB znova konzistentná – zachované integritné obmedzenia)
  - **I**solation - Nezávislosť (sú nezávislé od iných transakcií)
  - **D**urability - Trvanlivosť (po potvrdení vykonania sú zmeny uložené)

Atomickosť transakcií

- Transakcia môže ukončiť úspešne commitom, alebo môže byť zrušená cez abort po vykonaní niekoľkých akcií
- Z pohľadu používateľa:
  - Všetky operácie sa buď úspešne vykonajú, alebo sa nevykoná nič
  - Reálne sa však niečo vykonáva v oboch prípadoch
- DBMS robí log všetkých akcií a v prípade potreby sa urobí undo abortovaných transakcií.

Príklad

- Predpokladajme 2 transakcie:

T1	BEGIN A=A+100	B=B-100 END
T2	BEGIN A=1.06*A	B=1.06*B END

- Prvá transakcia posiela 100 EUR z účtu B na účet A
- Druhá transakcia pripočítava úrok 6%.
- Obe transakcie spustíme naraz – nevieme, čo sa bude diať skôr
- Výsledok však *musí byť ekvivalentný* buď so sekvenčným spustením T1, T2 alebo T2, T1

Plánovanie transakcií

- **Sekvenčný plán**: Plán, v ktorom sa nevykonávajú viaceré transakcie súčasne.
- **Ekvivalentný plán**: Taký plán, ktorého vykonanie by pre ľubovoľný počiatočný stav objektov databázy zanechalo databázu v rovnakom stave, ako pôvodný plán
- **Serializovateľný plán**: Taký plán, ktorý je ekvivalentný s niektorým sekvenčným plánom
    - Ak každá transakcia zachováva konzistentnosť, tak každý serializovateľný plán zachováva konzistentonosť

Príklady plánov

- Jedno možné usporiadanie operácií v pláne:

T1	A=A+100		B=B-100	
T2		A=1.06*A		B=1.06*B

- Iné usporiadanie:

T1	A=A+100			B=B-100
T2		A=1.06*A	B=1.06*B	

- Pohľad DBMS na plán:

T1	R(A)	W(A)					R(B)	W(B)
T2			R(A)	W(A)	R(B)	W(B)		

Nežiadané situácie

- Nasledujúce situácie môžu nastáť bez kontroly konkurencie:
  - Čítanie nekomitnutých dát = "dirty reads" (WR konflikty)

T1	R(A)	W(A)				R(B)	W(B)	Abort
T2			R(A)	W(A)	C			

  - Prepísanie nekomitnutých dát (WW konflikty)

T1	W(A)				W(B)	C
T2		W(A)	W(B)	C		

  - Nezopakovateľné čítanie (RW konflikty)

T1	R(A)				R(A)	W(A)	C
T2		R(A)	W(A)	C			

  - Fantómové čítanie

Nežiadané situácie

- Nasledujúce situácie môžu nastáť bez kontroly konkurencie:
  - Čítanie nekomitnutých dát = "dirty reads" (WR konflikty)
  - Prepísanie nekomitnutých dát (WW konflikty)
  - Nezopakovateľné čítanie (RW konflikty)
  - Fantómové čítanie:
    - Zamknem iba tie stránky, ktoré čítam.
    - Niekto mi pridá nový záznam, ktorý tiež vyhovuje podmienke zámku, ale už nie je zamknutý, lebo je v inej stránke.
    - Čítam v transakcii to isté 2x s iným výsledkom.

Možnosti izolácie transakcií

Isolation level	Dirty read	Nezopakovateľné čítanie	Fantómové čítanie (pridanie k zamknutým)
Read Uncommitted (dirty reads)	Áno	Áno	Áno
Read Committed (cursor stability)	Nie	Áno	Áno
Repeatable Reads	Nie	Nie	Áno
Serializable	Nie	Nie	Nie

Concurency control

Kapitola 19

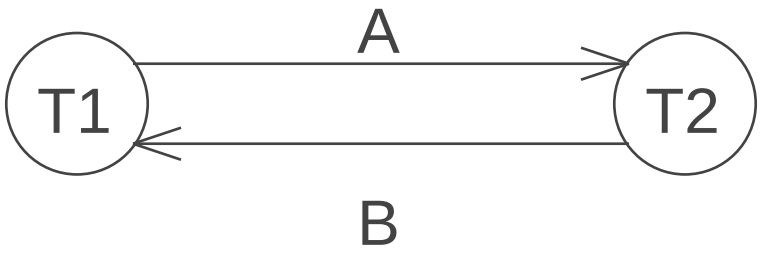
Kontrola konkurencie cez stránky

- **Striktné dvojfázové zamykanie (Strict 2PL):**
  - Každá transakcia musí získať zdieľaný zámok na objekty, ktoré ide čítať a **exkluzívny zámok** na objekty, ktoré ide meniť/zapisovať.
  - Všetky zámky sú transakciou držané pokiaľ neskončí
    - (Nie striktné) 2PL : Uvoľni zámky kedykoľvek, ale po uvoľnení nejakého zámku, už žiadne nemôžeš získať
  - Ak transakcia drží exkluzívny zámok, žiadna ďalšia transakcia nemôže získať na objekte ani jeden zo zámkov
- Striktné 2PL zabezpečuje, že všetky plány sú serializovateľné
  - Naviac zjednodušuje abort
  - (Non-strict) 2PL tiež umožňuje iba serializovateľné plány, ale môže spôsobiť kaskádové aborty

Konfliktná ekvivalencia

- Dva plány sú konfliktne ekvivalentné ak:
  - Obsahujú rovnaké operácie transakcií a
  - Každá dvojica konfliktných operácií (nad rovnakými objektmi) je vykonávaná v rovnakom poradí
- Plán P je konfliktne serializovateľný, ak P je konfliktne ekvivalentný s nejakým sekvenčným plánom
- Plán, ktorý nie je konfliktne serializovateľný:

T1	R(A)	W(A)					R(B)	W(B)
T2			R(A)	W(A)	R(B)	W(B)		



- Cyklus v grafe značí problém. Výstup T1 závisí na T2 a naopak.

View ekvivalencia

- Plány P1 a P2 sú view ekvivaletné ak:
  - T_i číta hodnotu z A ako prvá v pláne P1, tak ju číta ako prvá aj v pláne P2
  - T_i číta hodnotu z A zapísanú transakciou T_j v P1, potom ju číta aj v pláne P2
  - T_i zapisuje hodnotu do A ako posledná v pláne P1, potom ju zapisuje ako posledná aj v pláne P2

T1	R(A)		W(A)	
T2		W(A)		
T3				W(A)

T1	R(A)	W(A)		
T2			W(A)	
T3				W(A)

Manažment zámkov

- Požiadavky na zamykanie a odomykanie spracováva zámkový manažér
- O jednom zámku vieme:
  - Počet trasakcií vlastniacich zámok
  - Typ zámku (zdieľaný alebo exkluzívny)
  - Smerník na začiatok radu požiadaviek na zámok
- Zamykanie a odomykanie musia byť atomické v rámci celého DBMS
- Upgrade zámku: ak transakcia drží zdieľaný zámok, môže požiadať o exkluzívny

Deadlock

- Cyklus transakcií čakajúcich na uvoľnenie zámkov iných transakcií
- Sú dve možnosti ako ho riešiť:
  - Predchádzanie deadlockom
  - Odhaľovanie deadlockov

Predchádzanie deadlockom

- Priraďovanie priority transakciám podľa timestampu.
- Ak T_i chce zámok držaný transakciou T_j:
  - Wait-Die: Ak T_i má vyššiu prioritu, T_i čaká na T_j; inak T_i sa abortuje
  - Wound-wait: Ak T_i má vyššiu prioritu, T_j sa abortuje; inak T_i čaká na T_j
- Ak sa potom transakcia reštartuje, použije sa rovnaký timestamp ako mala pred abortom.

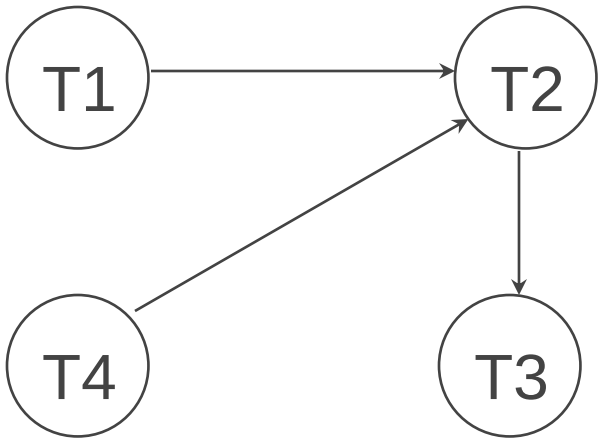
Odhaľovanie deadlockov

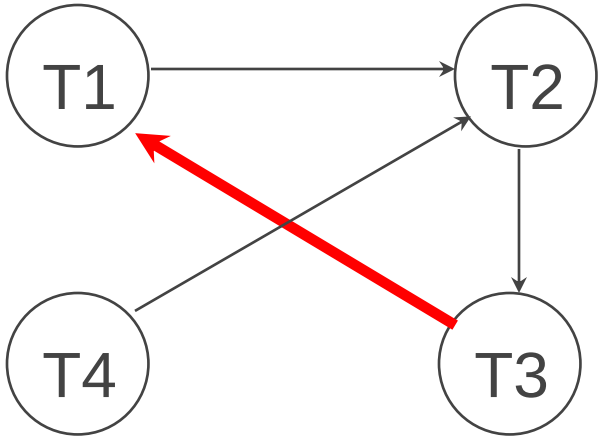
- Vytvorí sa graf závislostí
- Uzly sú transakcie
- Hrana z T_i do T_j ak T_j číta objekt predtým zmenený v T_i
- Plán je konfliktne serializovateľný práve vtedy keď jeho graf závislostí je acyklický

Odhaľovanie deadlockov

Príklad:

T1	S(A)	R(A)			S(B)					
T2			X(B)	W(B)				X(C)		
T3						S(C)	R(C)			X(A)
T4									X(B)	



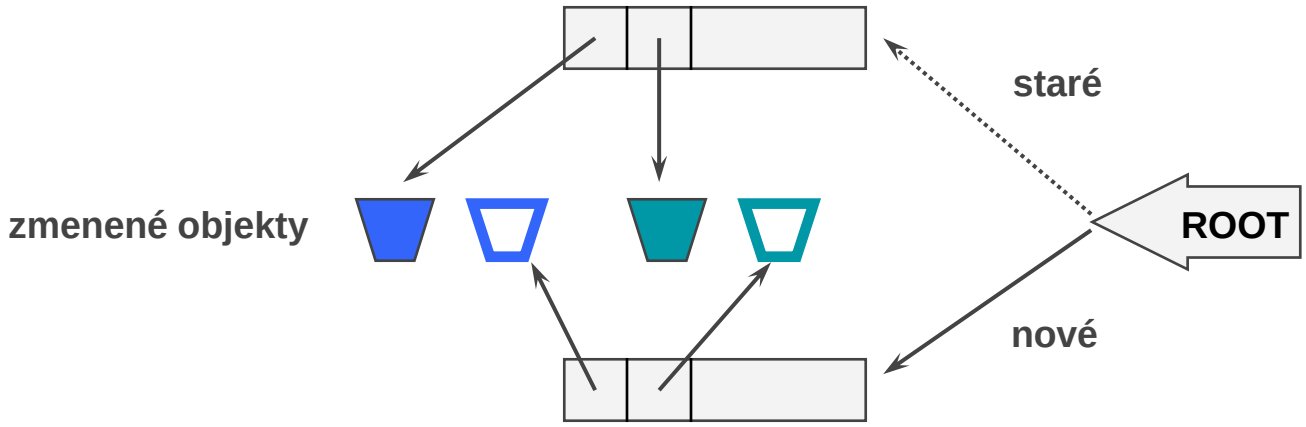


Nevýhoda zamykania

- Zamykanie je pesimistická metóda
- Nevýhody zamykania:
  - Potreba manažmentu zámkov
  - Odhaľovanie/prevencia deadlockov
  - Súperenie o zámky nad často používanými objektmi
- Iné možnosti:
  - Optimistická kontrola konkurencie
  - Kontrola konkurencie pomocou časovej pečiatky

Optimistické kontrola konkurencie

- Transakcie majú tri fázy:
  - Čítanie: Transakcia číta z databázy, ale zmeny robí do vlastnej kópie
  - Overenie: Pred commitom overí či nemôže nastať konflikt. Ak áno, tak je transakcia prerušená a vlastná kópia sa zmaže.
  - Zápis: Ak v druhom kroku neodhalí konflikt, tak nahradí pôvodnú kópiu vlastnou.

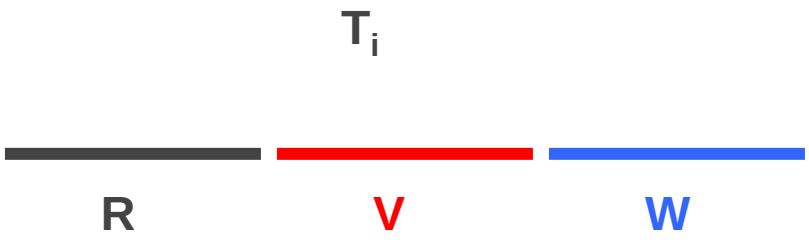


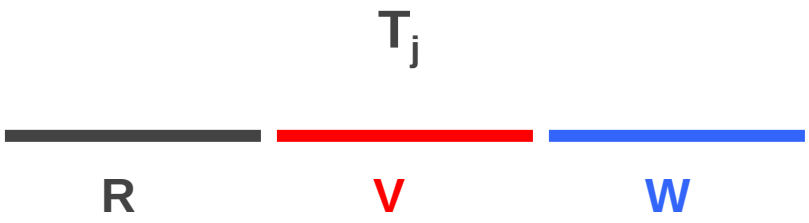
Overenie

- Testovanie podmienok na nekonfliktnosť
- Každá transakcia má ID (najčastejšie **timestamp**).
- ID transakcie je pridelené na konci čítacej fázy (prečo vtedy?)
- ReadSet(T_i): Množina objektov čítaných transakciou T_i.
- WriteSet(T_i): Množina objektov zmenených transakciou T_i
- Pridelenie id transakcie, overovanie a fáza zápisu sa dejú v kritickej sekcii
  - Vtedy nič ďalšie nemôže paralelne bežať
  - Ak máme dlhú fázu zápisu, môže to spomaľovať systém
- Optimalizácia pre Read-only transakcie:
  - Nepoužijeme kritickú sekciu (lebo nemáme fázu zápisu).

Test 1

Pre všetky i a j také, že $T_i < T_j$, zisti, či sa T_i ukončí skôr ako začne T_j.

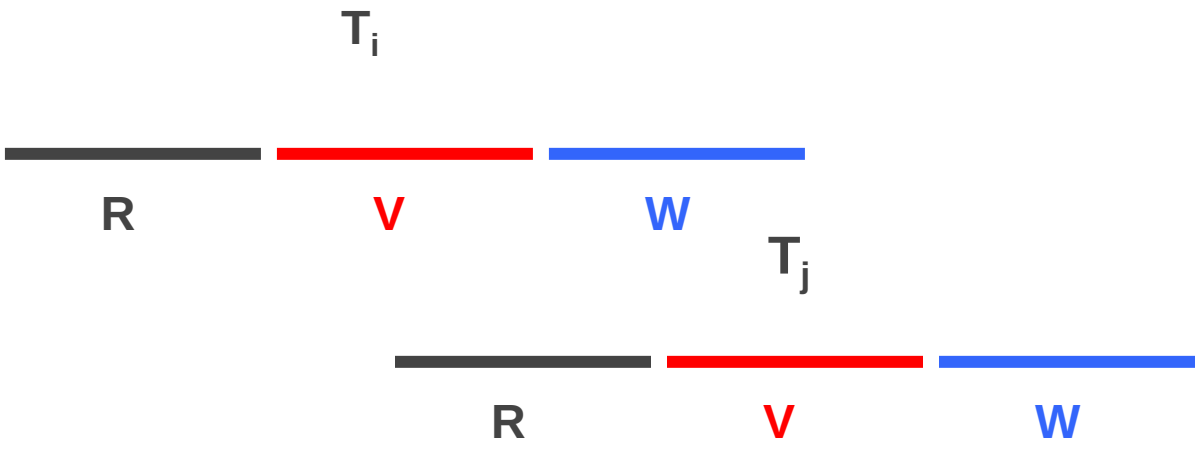




Test 2

Pre všetky i a j také, že $T_i < T_j$, over, či:

- T_i skončí predtým, než T_j začne svoju fázu zápisu a či
- $WriteSet(T_i) \cap ReadSet(T_j)$ je prázdny.

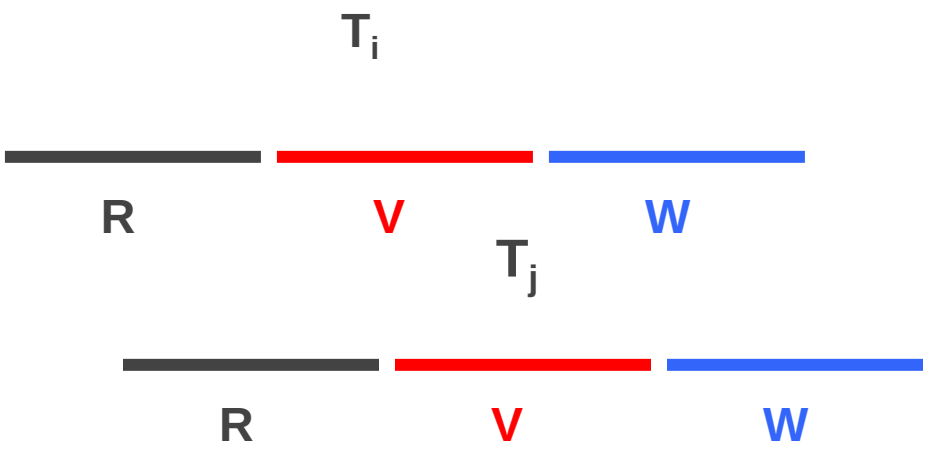


Číta T_j zmenené dáta?

Test 3

Pre všetky i a j také, že $T_i < T_j$, zisti či:

- T_i skončí čítanie skôr ako T_j a
- WriteSet(T_i) ∩ ReadSet(T_j) je prázdny a
- WriteSet(T_i) ∩ WriteSet(T_j) je prázdny.



Číta T_j zmenené dáta? Prepisuje T_i zmeny z T_j?

Nevýhody optimistickej kontroly konkurencie

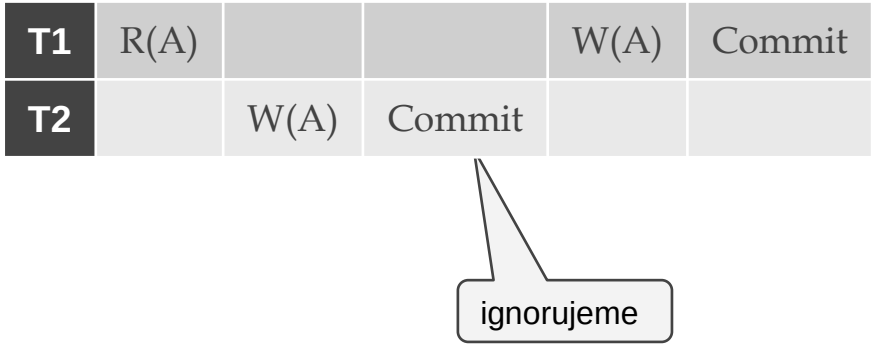
- Je potrebné vytvárať ReadSet and WriteSet pre každú transakciu.
- Počas overovania zisťujeme konflikty a nemáme konkurentné zápisy
  - Kritická sekcia zhoršuje paralelizmus
- Reštartujú sa transakcie, ktoré neprejdú cez overenie.
  - Všetka doterajšia robota počas fázy čítania sa zahadzuje

Timestamp kontrola konkurencie

- Každá transakcia dostane timestamp (TS) na začiatku a môžeme zabezpečiť, že ak akcia a_i transakcie T_i je v konflikte s akciou a_j transakcie T_j, a $TS(T_i) < TS(T_j)$, tak a_i sa musí vykonať skôr ako a_j. Inak reštartujeme transakciu T_i.
- Ďalej, každý objektu dostane read-timestamp (RTS) a write-timestamp (WTS).
- Keď T chce čítať objekt O:
  - Ak TS(T) < WTS(O), porušili sme poradie timestampov transakcie T vzhľadom na objekt O a preto reštartneme T s novým TS (Ak by sme zachovali TS, skončili by sme znova!)
  - Ak TS(T) > WTS(O), tak
    - T môže čítať O
    - RTS(O) nastaví na max(RTS(O), TS(T)).
  - Zmena RTS(O) sa musí zapísať na disk (čím rastie réžia transakcií).

Keď T chce zapísať objekt O

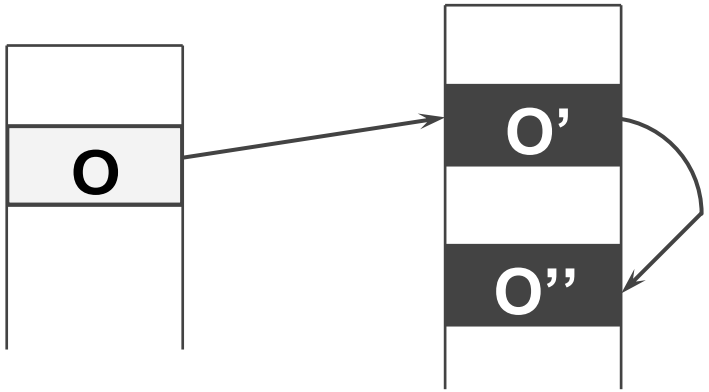
- Ak TS(T) < RTS(O), porušuje poradie timestampov transakcie T vzhľadom na objekt O. T sa preruší a reštartne.
- Ak TS(T) < WTS(O), porušuje poradie timestampov transakcie T vzhľadom na objekt O. Takýto neaktuálny zápis môžeme ignorovať a pokračovať ďalej (**Thomas Write Rule) .**
- Inak, umožni T zapísať objekt O.



Kontrola konkurencie cez multiverzie

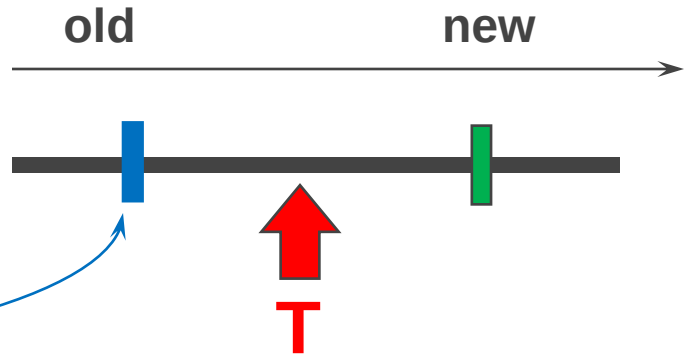
- Cieľ: aby čítania boli umožnené vždy
- Idea: vytvoriť niekoľko verzií každého objektu, kde každý objekt má WTS. Transakcia T číta najaktuálnejšiu verziu.
- Každá transakcia je pri spustení označená ako čitateľ alebo zapisovateľ objektu (nie je isté, že zapisovateľ bude zapisovať, ale čitateľ nemôže).

Hlavný Segment (Aktuálna verzia objektu)



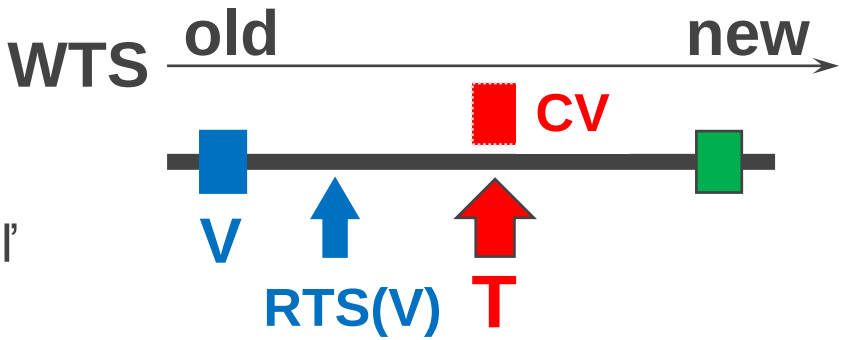
Zoznam verzií - staršie verzie môžu byť užitočné pre aktívne čítania (repeatable reads)

Čitateľ



- Pre každý objekt, ktorý ide čítať:
  - Nájdi **najnovšiu verziu** s WTS < TS(T)
    (skúšame od aktuálnej verzie a ak treba pokračujeme cez staršie verzie)
- Predpokladáme, že takú verziu nájdeme
- Čitateľ sa nikdy nereštartuje
  - Môže byť však blokovaný, pokiaľ sa nekomitne zápis verzie, ktorú chceme čítať (read commited).

Zapisovateľ



- Ak chce čítať, robí to čo čitateľ
- Ak chce zapisovať:
  - Nájdi **najnovšiu verziu V** kde WTS < TS(T).
  - Ak RTS(V) < TS(T), vytvor kópiu V ozn. CV, so smerníkom na V, a s WTS(CV) = TS(T), RTS(CV) = TS(T). (Zápis je v buffri, pokiaľ T nie je komitnutá; ostatné transakcie vidia, že existuje nová verzia CV, ale nevedia ju zatiaľ čítať)
  - Inak, nedovoľ zápis.