Transakční zpracování

Transakční zpracování

Dva základní požadavky na SŘBD:

- chránit data organizovaná pod daným SŘBD,
- poskytnout korektní a rychlý asynchronní přístup většímu množství uživatelů.

Řešení:

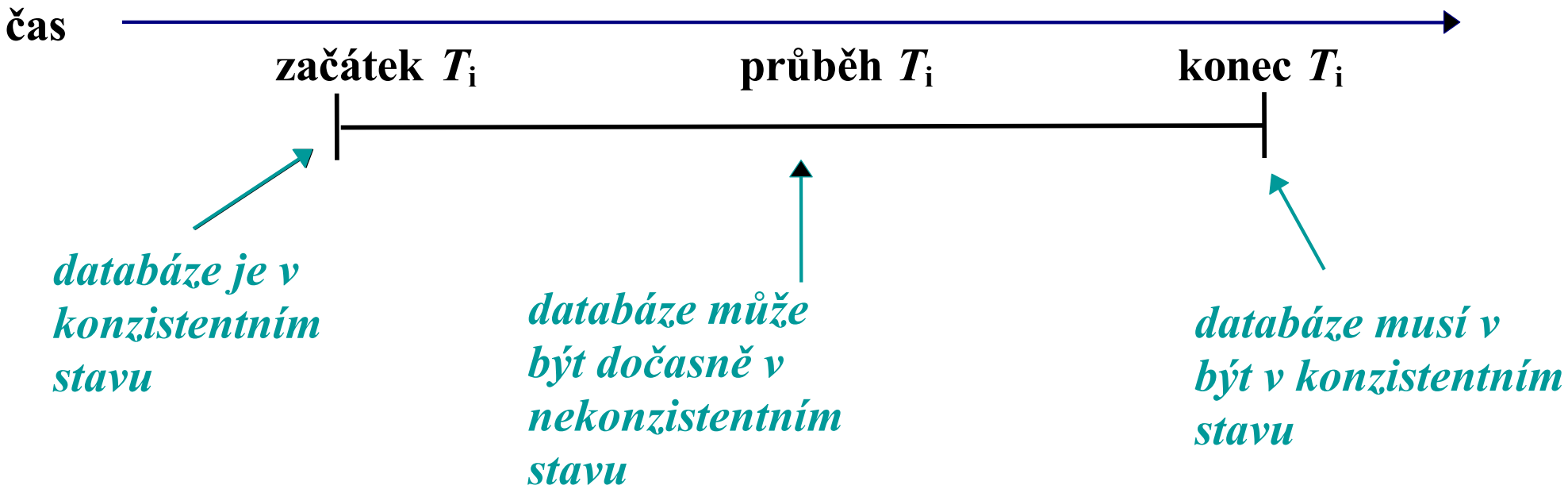
- komponenta řízení souběžného (paralelního) zpracování (concurrency control)
- komponenta zotavení z chyb (recovery). Řízení souběžného zpracování

Transakční zpracování

- Řízení souběžného zpracování
  zajišťuje uživatelům, že každý vidí **konzistentní stav databáze** bez ohledu na to, že více uživatelů přistupuje asynchronně ke stejným údajům.
- **Zotavení z chyb**
  zajišťuje, že stav databáze není narušen v případě chyby software, systému, nebo fyzického média v průběhu zpracování úlohy měnící data v databázi. Důsledkem takového incidentu nesmí být nekonzistence databáze.

Transakční zpracování

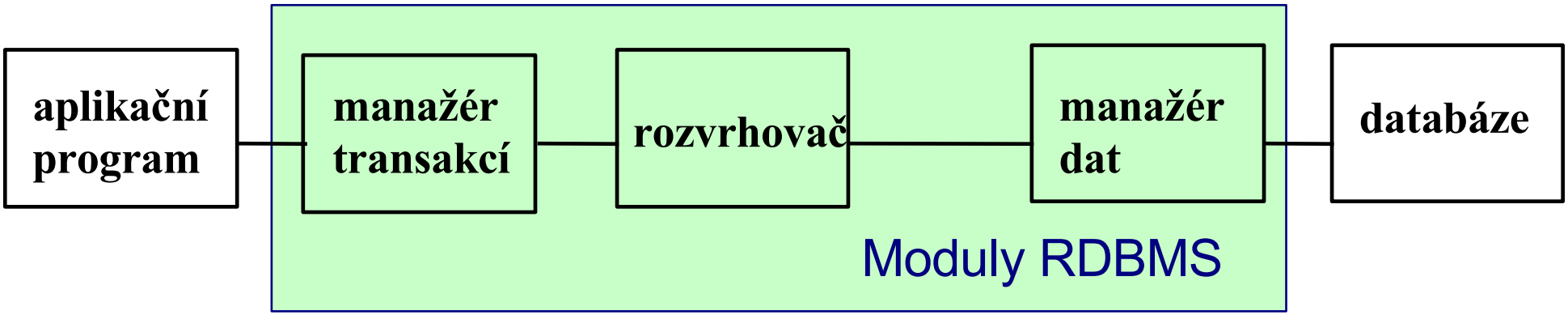
Řešením je **transakce**:
vhodná programová jednotka a vhodné mechanismy, které zabezpečí, že po skončení akce (korektním i nekorektním) zůstane databáze konzistentní (platí všechna IO definovaná ve schématu).



Transakční zpracování

Vlastnosti transakce:

- transparentnost paralelního zpracování
- Atomicita vzhledem k chybám



Transakční zpracování

Transakce je vymezená ***logická jednotka práce***

převod peněz z jednoho konta na druhé

přihlášení na zkoušku

odhlášení ze zkoušky

přehlášení na jiný termín

zápis na cvičení

odepsání ze cvičení

přehození na jiné cvičení

Transakční zpracování

Hranice: konec transakce - explicitní
- **COMMIT** (potvrzení)
- **ROLLBACK** (zrušení)

- implicitní **ROLBACK**

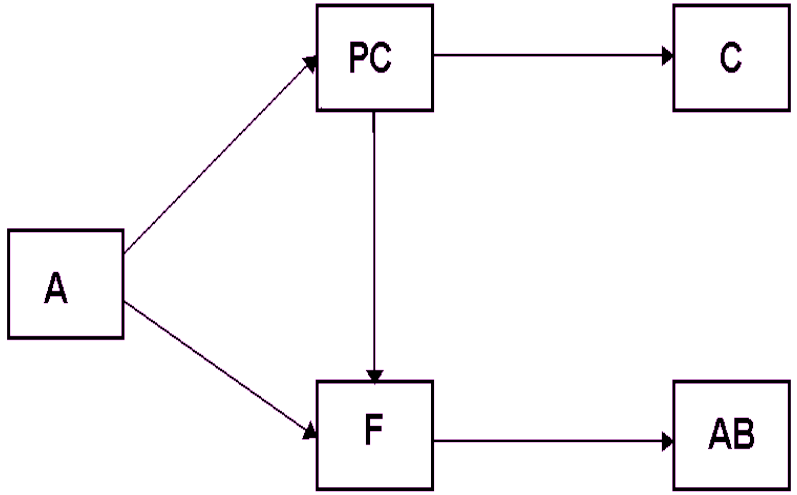
začátek transakce - implicitní

- explicitní **SET TRANSACTION**

Obnovení konzistence po incidentu

- Operace použité při obnově:
  - **UNDO**
  - **REDO**
- Využití transakčního žurnálu

Transakční zpracování stavový diagram transakce



Transakce se může dostat do jednoho z pěti stavů:

- aktivní (A) - od počátku provádění transakce,
- částečně potvrzený (PC) - po provedení poslední operace transakce,
- chybný (F) - v normálním průběhu transakce nelze pokračovat;
- zrušený (AB) - po skončení operace ROLLBACK, tj. uvedení databáze do stavu před transakcí;
- potvrzený (C) - úspěšném zakončení, tj. po potvrzení operací COMMIT.

Transakční zpracování

ACID vlastnosti transakce:

- atomicita – (**A**tomicity) - transakce musí buď proběhnout celá nebo vůbec,
- konzistence – (**C**onsistency) - transakce transformuje databázi konzist. stavu do jiného konzist. stavu,
- nezávislost – (**I**ndependence) - dílčí efekty jedné transakce nejsou viditelné jiným transakcím,
- trvanlivost – (**D**urability) - efekty úspěšně ukončené (potvrzené) transakce jsou trvale uloženy (persistence).

ZOTAVENÍ Z CHYBY

Optimistická strategie

Pesimistická strategie

ZOTAVENÍ Z CHYB — *třídy možných chyb*

Globální chyby (mají vliv na více transakcí)

- Spadnutí systému serveru (např. výpadek proudu), důsledkem je obecně ztráta obsahu vyrov. pamětí.
- Chyby systémové, které ovlivňují transakce, avšak nikoli celou databázi (např. uváznutí, odumření komunikace klienta se serverem),
- Chyby médií (např. incident na disku), které zapříčiní ohrožení databáze, nebo její části,

Lokální chyby (v jedné transakci).

- Logické chyby, které by měly být “odchyceny” a ošetřeny na úrovni transakce explicitním vyvoláním operace ROLLBACK (pokus o porušení IO při zápisu do DB, dělení nulou při výpočtu).

ZOTAVENÍ Z CHYBY *po restartu systému*

Při optimistické strategii, po restartu:

- Na transakce, jejichž stav bude v důsledku incidentu nedefinovaný je nutné použít ROLLBACK.
- Transakce, které byly úplné před započetím chyby systému, avšak které nebyly dokončeny fyzicky (např. odeslání vyrovnávacích pamětí na disk), je nutné zopakovat (REDO).

synchronizační body - vyznačují v žurnálu hranice mezi dvěma po sobě následujícími transakcemi.

ZOTAVENÍ Z CHYBY MÉDIÍ

- Natáhnutí celé databáze ze záložní kopie (Backup)
- Použití žurnálu k REDO všech ukončených transakcí až do té chvíle, kdy ještě žurnál poskytuje potřebné informace.

PARALELNÍ ZPRACOVÁNÍ TRANSAKCÍ

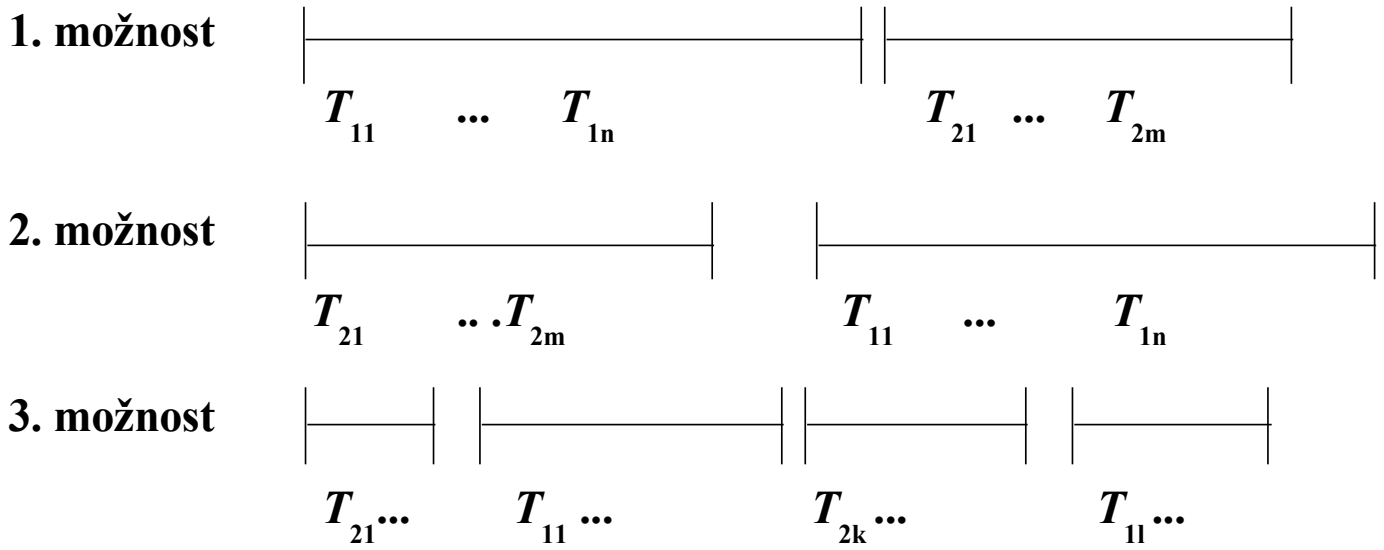
Předpoklad ***ploché transakce***, objekty ***atomické***, FETCH, resp. OUTPUT přesunou stránku vyrovnávací paměti z/na disk,

READ(X,x) - přiřazuje hodnotu atributu X lokální proměnné x. Není-li stránka s danou hodnotou ve VP, provede se FETCH(X), následuje přiřazení hodnoty X z VP proměnné x.

WRITE(X,x) - přiřazuje hodnotu lokální proměnné x atributu X ve vyrovnávací paměti. Operace má opět dvě fáze - není-li žádaná stránka ve vyrovnávací paměti, provede se FETCH(X), následuje přiřazení proměnné x atributu X ve vyrovnávací paměti.

PARALELNÍ ZPRACOVÁNÍ TRANSAKCÍ

Transakce T_1 a T_2 se skládají z akcí $\{T_{11},...,T_{1n}\}$ a $\{T_{21},...,T_{2m}\}$.



Stanovení pořadí provádění dílčích akcí více transakcí v čase nazveme **rozvrhem**. Maximalizace paralelismu zpracování, tj. vytváření rozvrhu, je věcí **rozvrhovače**.

PROBLÉMY S PARALELNÍM ZPRACOVÁNÍM

- nebezpečí **ztráty aktualizace** (při “prokládání” transakcí)

T_1	T_2	stav
READ(X)		X = 80
X:=X-5		zrušení 5 rezervací
	READ(X)	X = 80
	X:=X+4	přidání 4 rezervací
WRITE(X)		X = 75
READ(Z)		
	WRITE(X)	X = 84
Z:=Z+N		

Přestože by hodnota X v databázi měla být 79, je 84.

PROBLÉMY S PARALELNÍM ZPRACOVÁNÍM

- nebezpečí **dočasné aktualizace** (při chybě systému)

T_1	T_2
READ(X)	
X:=X-5	
WRITE(X)	
	READ(X)
	X:=X+4
	WRITE(X)
READ(Z)	
--- chyba transakce	

Po provedení operace ROLLBACK u transakce T1 bude aktualizace provedená transakcí T2 založena na posléze odvolaných změnách takže výsledek bude chybný.

PROBLÉMY S PARALELNÍM ZPRACOVÁNÍM

- nebezpečí **nekorektního použití agregační funkce**

T_1	T_3	stav
	sum:=0	čítač počtu rezervací
	READ(A)	
	sum:=sum + A	
…	...	
READ(X)		
X:=X-5		
WRITE(X)		
	READ(X)	čte po změně X
	sum:=sum + X	
	READ(Z)	
	sum:=sum + Z	čte před změnou Z
WRITE(Z)		

PROBLÉMY S PARALELNÍM ZPRACOVÁNÍM

- nebezpečí **neopakovatelného čtení**

Transakce T1 používající daný kurzor se snaží znovu přečíst řádek, který již jednou četla dříve, ten však již obsahuje jiné hodnoty (nebo neexistuje) díky aktualizaci transakcí T2, která probíhá paralelně.

- **fantóm**

Transakce T1 přečte množinu řádků (jeden příkaz SELECT) a zpracovává je. Mezitím některé z těchto řádků změní transakce T2 (DELETE nebo INSERT). Použije-li T1 týž příkaz (pro jiné kalkulace), obdrží již jinou množinu řádků.

PROBLÉMY S PARALELNÍM ZPRACOVÁNÍM

Povolené anomálie v SQL92:

	Úroveň izolace			
Anomálie	**0**	**1**	**2**	**3**
dočasná aktualizace	×	-	-	-
neopakovatelné čtení	×	×	-	-
fantóm	×	×	×	-

ANSI SQL nařizuje implicitně úroveň 3

USPOŘÁDATELNOST ROZVRHU

Sériové rozvrhy zachovávají operace každé transakce pohromadě. Pro provedení N transakcí tedy lze naplánovat ***N!*** různých sériových rozvrhů.

Lze vytvořit i rozvrh, kde jsou operace navzájem **prokládány** ... **paralelní rozvrh**

Přirozeným požadavkem na korektnost je, aby **efekt** paralelního zpracování transakcí byl týž, jako podle nějakého sériového rozvrhu.
Rozvrh je **korektní**, když je v **nějakém smyslu ekvivalentní** kterémukoliv sériovému rozvrhu.
O transakčním zpracování, které zaručuje tuto vlastnost se říká, že zaručuje **uspořádatelnost**.

USPOŘÁDATELNOST ROZVRHU

definici ekvivalence založené na komutativitě operací READ a WRITE:

Dvě operace jsou **konfliktní**, jestliže výsledky jejich různého sériového volání nejsou ekvivalentní.

Operace, které nejsou konfliktní nazýváme **kompatibilní.**

komutativita	$READ_j(A)$	$WRITE_j(A)$
$READ_i(A)$	+	-
$WRITE_i(A)$	-	-

USPOŘÁDATELNOST ROZVRHU

Definice 6.2.1: Máme-li rozvrhy S1 a S2 pro množinu transakcí T = {T1,...,TN}, pak **S1 s S2 jsou ekvivalentní** (vzhledem ke konfliktům), jsou-li splněny dvě podmínky:

1. Jestliže se v prvním rozvrhu vyskytuje READ(A) v Ti a tato hodnota vznikla z WRITE(A) v transakci Tj, potom totéž musí být zachováno v druhém rozvrhu.

2. Jestliže se v prvním rozvrhu vyvolá poslední operace WRITE(A) v Ti, pak totéž musí být i v druhém rozvrhu.

Relativní pořadí konfliktních operací je stejné v S1 i S2.

USPOŘÁDATELNOST ROZVRHU

Příklad 6.2.1: Mějme transakce

T4: {READ(A),akce1(A),WRITE(A),READ(B),akce2(B),WRITE(B)}

T5: {READ(A),akce3(A),WRITE(A),READ(B),akce4(B),WRITE(B)}

S 3	
T_4	T_5
READ(A)	
akce1(A)	
WRITE(A)	
	READ(A)
	akce3(A)
	WRITE(A)
READ(B)	
akce2(B)	
WRITE(B)	
	READ(B)
	akce4(B)
	WRITE(B)

S 4	
T_4	T_5
READ(A)	
akce1(A)	
	READ(A)
	akce3(A)
	WRITE(A)
	READ(B)
WRITE(A)	
READ(B)	
akce2(B)	
WRITE(B)	
	akce4(B)
	WRITE(B)

S1:{T4,T5}, S2:{T5,T4}. jsou sériové rozvrhy.

S3 a S4 nejsou sériové.

Zřejmě S3 ≡ S1, S3 ≠ S2.

Rozvrh je uspořádatelný, jestliže existuje sériový rozvrh s ním ekvivalentní.

USPOŘÁDATELNOST ROZVRHU

O rozvrhu jsme řekli, že je **uspořádatelný**, jestliže existuje sériový rozvrh s ním ekvivalentní.

Mohou však existovat rozvrhy, které nejsou ekvivalentní se žádným sériovým rozvrhem podle této definice a přesto produkují stejný výsledek jako nějaký sériový rozvrh. Pojem konfliktu je zde založen na **znalosti sémantiky operací**.

Existují možnosti definovat smysluplná kritéria korektnosti, která vůbec nejsou založena na pojmu uspořádatelnost.

TESTOVÁNÍ USPOŘÁDATELNOSTI

Precedenční graf rozvrhu. Jde o orientovaný graf {U,H}
$U = \{T_i \mid i = 1, 2, ..., n\}$, $H=\{h_{ik}(A)\}$

$h_{ik}(A)$... vzhledem k manipulacím s objektem A vede hrana od uzlu T_i k uzlu T_k , čímž chceme říci, že rozvrh může být ekvivalentní pouze s takovým sériovým rozvrhem, kde T_i předchází T_k.

TESTOVÁNÍ USPOŘÁDATELNOSTI

Konstrukce ***precedenčního grafu*** rozvrhu S pro $\{T_i, T_k\}$.
Od uzlu T_j povede hraha k uzlu T_k, jestliže:

(i) poslední WRITE(A) v T_j **je před** posledním voláním WRITE(A) v T_k

(ii) T_j volá WRITE(A) před tím, než T_k volá READ(A)
(v T_k se čte z databáze hodnota A, vzniklá v T_j)

(iii) T_j volá READ(A) před tím, než T_k volá WRITE(A)
(v T_j se čte z dB hodnota A, dříve, než se v T_k změní)

TESTOVÁNÍ USPOŘÁDATELNOSTI

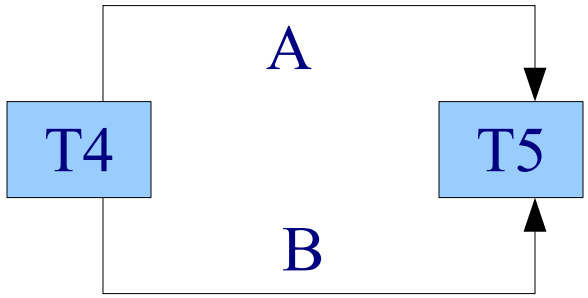
Precedenční grafy k př. 6.2.1

S 3		*S 3'*	
T_4	T_5	T_4	T_5
READ(A)		READ(A)	
akce1(A)		akce1(A)	
WRITE(A)		WRITE(A)	
	READ(A)		READ(A)
	akce3(A)		akce3(A)
	WRITE(A)		WRITE(A)
READ(B)			READ(B)
akce2(B)			akce4(B)
WRITE(B)			WRITE(B)
	READ(B)	READ(B)	
	akce4(B)	akce2(B)	
	WRITE(B)	WRITE(B)	

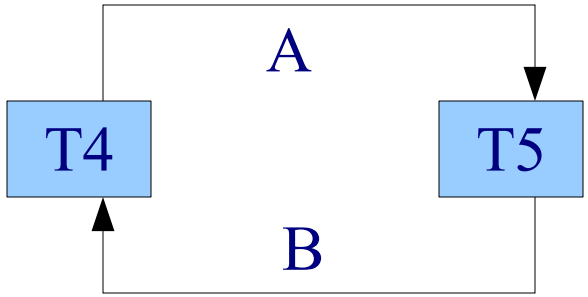
S1:

S2:

S3:



S3':



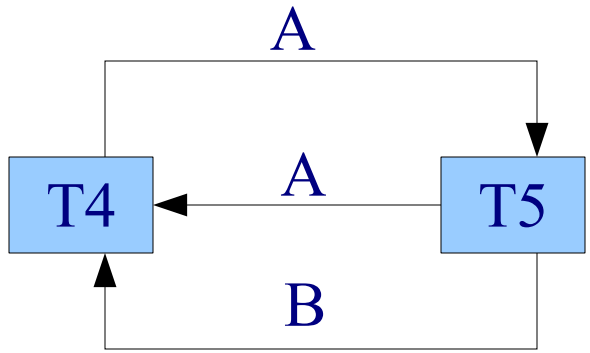
Rozvrhy S1{T1, T2} a S2 {T2, T1} jsou sériové.

TESTOVÁNÍ USPOŘÁDATELNOSTI

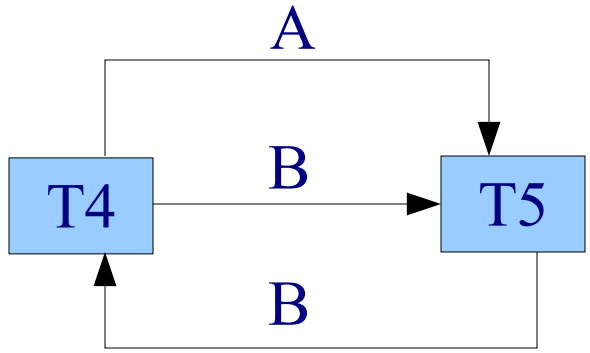
Precedenční grafy k př. 6.2.1

S	*4*	*S*	*4'*
T_4	T_5	T_4	T_5
READ(A)		READ(A)	
akce1(A)		akce1(A)	
	READ(A)		READ(A)
	akce3(A)		akce3(A)
	WRITE(A)		WRITE(A)
	READ(B)		READ(B)
			akce4(B)
WRITE(A)			
READ(B)			
akce2(B)		READ(B)	
WRITE(B)		akce2(A,B)	
			WRITE(B)
	akce4(B)		
	WRITE(B)	WRITE(B)	

S4:



S4':



TESTOVÁNÍ USPOŘÁDATELNOSTI

Tvrzení: Rozvrh je **uspořádatelný,** (tedy ekvivalentní některému sériovému rozvrhu), jestliže v jeho precedenčním grafu neexistuje cyklus.

Podle tohoto tvrzení:
rozvrh S4 není uspořádatelný,
rozvrh S3 je uspořádatelný.

Tvrzení: Dva rozvrhy jsou **ekvivalentní**, jestliže mají stejný precendenční graf.

UZAMYKACÍ PROTOKOLY

Testování uspořádatelnosti jakéhokoliv rozvrhu není to nejlepší pro praxi. Časové nároky takového přístupu by zřejmě přesahovaly rozumnou míru.

Přístup z druhé strany:

Konstruovat transakce podle předem daných pravidel tak, že za určitých předpokladů bude každý jejich rozvrh uspořádatelný. Soustavě takových pravidel se obecně říká **protokol**.

UZAMYKACÍ PROTOKOLY

Nejznámější protokoly jsou založeny na dynamickém **zamykání** a **odmykání** objektů v databázi.

Jednoduchý model: objekt může být v daném čase uzamčen nejvýše jednou transakcí - Lock(A), Unlock(A)

Transakce, která uzamkla objekt, má právo na něm provádět operace READ a WRITE (a další operace).

UZAMYKACÍ PROTOKOLY

Legální rozvrh:

- objekt bude nutné mít v transakci uzamknutý, kdykoliv k němu chce tato transakce přistupovat,
- transakce se nebudou pokoušet uzamknout objekt již uzamknutý jinou transakcí (čekájí na uvolnění zámku).

Pozn:
Samotná legálnost rozvrhu nezaručuje uspořádatelnost.

UZAMYKACÍ PROTOKOLY

S7 a S8 nemají stejné výsledky. V S7 se *akce7* provádí s jinou hodnotou A, než v S8.

Zamykání a odmykání samo ještě nevede k uspořádatelném u rozvrhu.

S 7	
T_9	T_{10}
LOCK(B)	
READ(B)	
akce5(B)	
WRITE(B)	
UNLOCK(B)	
	LOCK(A)
	READ(A)
	UNLOCK(A)
	LOCK(B)
	READ(B)
	UNLOCK(B)
	A:=A+B
	WRITE(A)
LOCK(A)	
READ(A)	
akce7(A)	
WRITE(A)	
UNLOCK(A)	

S 8	
T_9	T_{10}
LOCK(B)	
READ(B)	
akce5(B)	
WRITE(B)	
UNLOCK(B)	
LOCK(A)	
READ(A)	
akce7(A)	
WRITE(A)	
UNLOCK(A)	
	LOCK(A)
	READ(A)
	UNLOCK(A)
	LOCK(B)
	READ(B)
	UNLOCK(B)
	A:=A+B
	WRITE(A)

UZAMYKACÍ PROTOKOLY

V případech některých rozvrhů může dokonce nastat **uváznutí** (deadlock).

Uváznutí lze řešit tak, že provedeme ROLLBACK jedné transakce. Co tato uzamkla, bude odemknuto, čímž se druhá transakce odblokuje.

T_{11}	T_{12}
LOCK(B)	
READ(B)	
akce5(B)	
WRITE(B)	
	LOCK(A)
	READ(A)
čeká➔	LOCK(B)
	READ(B)
	UNLOCK(B)
	A:=A+B
	WRITE(A)
čeká➔ LOCK(A)	
UNLOCK(B)	
READ(A)	

Lépe je uváznutí nepřipustit vhodným návrhem. Co pomůže zde?

UZAMYKACÍ PROTOKOLY

Omezíme se dále na tzv. **dobře formované transakce**, které podporují přirozené požadavky na transakce:

a) transakce zamyká objekt, chce-li k němu přistupovat,

b) transakce nezamyká objekt, když ho již zamkla,

c) transakce neodmyká objekt, který nezamkla,

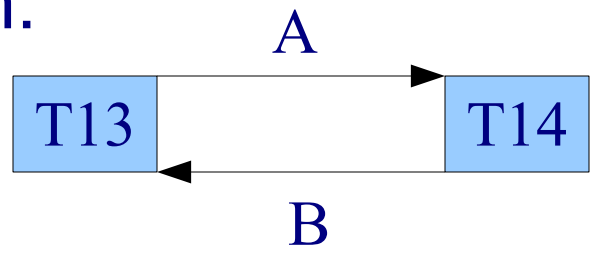
d) na konci transakce nezůstane žádný objekt zamčený.

Stále ještě neznáme postačující podmínku pro to, aby všechny ***legální rozvrhy pro dobře formované transakce*** byly uspořádatelné. Konzistence databáze totiž není zaručena tím, že objekt odemkneme bezprostředně po manipulaci s ním.

UZAMYKACÍ PROTOKOLY

T_{13}	T_{14}
LOCK(A)	
READ(A)	
WRITE(A)	
UNLOCK(A)	
	LOCK(A)
	READ(A)
	WRITE(A)
	UNLOCK(A)
	LOCK(B)
	READ(B)
	WRITE(B)
	UNLOCK(B)
LOCK(B)	
READ(B)	
WRITE(B)	
UNLOCK(B)	

Precendenční graf pro tento rozvrh:



není tedy uspořádatelný.

Pomůže, když uvažované transakce budou ***dvoufázové.***
Prohodíme-li *UNLOCK(A) s LOCK(B) v T14 bude tato transakce dvoufázová.*

UZAMYKACÍ PROTOKOLY

Uzamykací protokol je obecně množina pravidel, které udávají, kdy transakce bude uzamykat a odmykat objekty databáze.

T_{15}
LOCK(B)
READ(B)
akce5(B)
WRITE(B)
LOCK(A)
READ(A)
akce7(A)
WRITE(A)
UNLOCK(B)
UNLOCK(A)

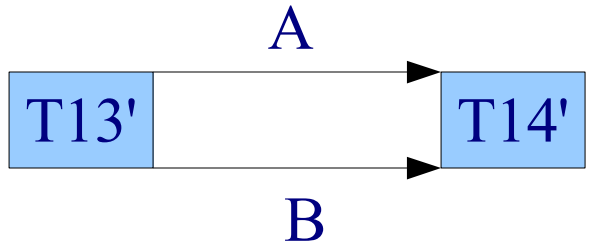
Dvoufázová transakce:
1. fáze - uzamyká se, nic neodemyká
2. fáze - od prvního odemknutí, do konce se už nic nezamyká

Tvrzení 6.2. Jestliže všechny transakce v dané množině transakcí T jsou *dobře formované* a *dvoufázové*, pak každý jejich *legální rozvrh* je uspořádatelný.

UZAMYKACÍ PROTOKOLY

T_{13}'	T_{14}'
LOCK(A)	
READ(A)	
Akce, WRITE(A)	
UNLOCK(A)	
	LOCK(A)
	READ(A)
	Akce, WRITE(A)
LOCK(B)	
READ(B)	
Akce, WRITE(B)	
UNLOCK(B)	
	LOCK(B)
	READ(B)
	UNLOCK(A)
	Akce, WRITE(B)
	UNLOCK(B)

Precendenční graf pro tento rozvrh:



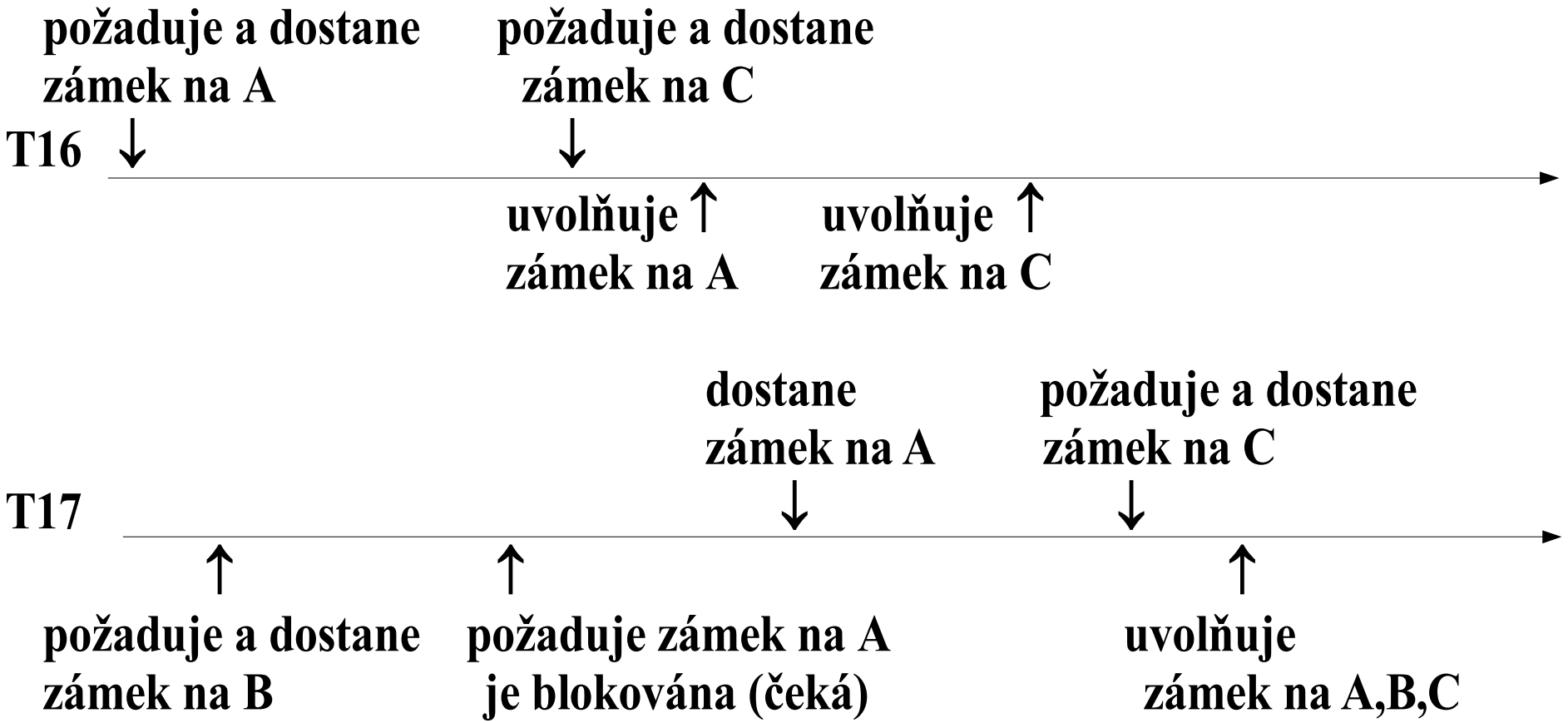
Je tedy uspořádatelný, ač T13' není dvoufázová. Nevylučujeme existenci uspořádatelného legálního rozvrhu (ne)dobře formovaných nedvoufázových transakcí.

UZAMYKACÍ PROTOKOLY

T_{16}	T_{17}
LOCK(A)	
READ(A)	
	LOCK(B)
	READ(B)
LOCK(C)	
READ(C)	
WRITE(A)	
UNLOCK(A)	
	LOCK(A)
	READ(A)
	WRITE(B)
WRITE(C)	
UNLOCK(C)	
	LOCK(C)
	READ(C)
	WRITE(C)
	UNLOCK(A)
	UNLOCK(B)
	UNLOCK(C)

Jeden z uspořádatelných dvoufázových rozvrhů T16 a T17.
Transakce T17 jednou čeká na uvolnění zámku pro A, jednou na uvolnění zámku C.
Kdyby LOCK(A), resp. LOCK(C) byly v rozvrhu před UNLOCK(A), resp. UNLOCK(C) v transakci T16, rozvrh by se stal nelegálním.

UZAMYKACÍ PROTOKOLY



UZAMYKACÍ PROTOKOLY

Uváznutí jsou pro uzamykání dle dvoufázového protokolu typická.

	T_{19}		T_{20}
1	LOCK(A)	4	LOCK(B)
2	READ(A)	5	READ(B)
3	WRITE(A)	6	WRITE(B)
	LOCK(B)		LOCK(A)
	READ(B)		READ(A)
	UNLOCK(A)		UNLOCK(B)
	WRITE(B)		WRITE(A)
	UNLOCK(B)		UNLOCK(A)

Rozvrh, plánující naznačené provádění kroků transakcí vede k **uváznutí**. Nelze pokračovat ani s LOCK(B), ani s LOCK(A). Tomuto nedostatku lze předejít tím, že ke společným objektům se v obou transakcích bude přistupovat ve stejném pořadí.