# Zkouška

5 prikladu po 20 bodech. Znamky:
3 ... 60 - 74
2 ... 75 - 89
1 ... 90 - 100
Hodnocení je celkem mírné ale látky opravdu hodně.

## Ukázky starých pisemek (vždy 5 příkladů)

Aries
One copy uspořádání
Locking stromu
Rozhodnutelnost
Rozvrh urcit vlastnosti a převod do S2PL: r1[x] r2[x] r3[x] w3[x] w2[z] w1[x] c3 r2[y] c2 c1

Urcit vlastnosti zadaneho rozvrhu (usporadatelnost, zotavitelnost, odolnost vůči kaskádovému abortu, striktnost, ...): r1[x] r2[x] w1[x] r1[z] w2[z] r1[y] w1[y] c1 c2
Popsat jak funguje plánování pomocí precedenčního grafu a naplanovat zadany rozvrh pomoci precedencniho grafu
ARIES
Sagy
Funkce TP-monitoru

MV a “one copy” uspořádání
SP a emulace HT pomoci SP
Prevence fantoma
ARIES (viz. TP manazer a zotavení systému)
prevod rozvrhu do S2PL pomocí WW (Wound/Wait strategie): r1[x] w1[y] r2[y] w2[z] w1[x] w2[y]

1) Transakce se SP(savepointy) + pravidla
2) Pohledova ekvivalence + 2 rozvrhy (je a neni ekv. SR)
3) Zamykani predchazejiciho/nasledujiciho klice - insert a delete pomoci zam. predch.
4) Zakladni TO pravidlo + rozvrh, ktery je generovany
5) Definice komplet MW (multi-version) historie + jejich eq.

Nevyřešené příklady jsou červeně

# 1. Modely transakcí

**Otázka:**

**Jak spočítat MTTF, MTTR, ...**

MTTF = 1 / FR
MTTF = suma (čas mezi logovanými chybami) / počet chyb
MTTF = čas který běžel / počet chyb

MTTR = suma (downtime) / počet chyb

MTBF = MTTR + MTTF

dostupnost = MTTF / (MTTF + MTTR)

**Příklad přednáška:**

Máte dva roky staré levnější auto a dočtete se, že 1 z 1000 dvouletých modelů se ročně rozpadne…

- Jaký je MTTF vašeho auta?
  - FR = 1 / 1000
  - MTTF = time / FR = 1rok / (1 / 1000) = 1000let
- Jaký je poměr selhání za hodinu?
  - 1/(365*24*1000) ~ 10^(-7)
- Když máte 100 takových aut, jaká je šance, že se vám jedno z nich rozpadne?
  - 100/1000 = 1/10
- Proč je odpověď na první dotaz o tolik větší, než je záruka auta?
  - je to pro 2 leté auto
- Když bude oprava trvat týden, jaká je dostupnost vašeho auta?
  - MTTF / (MTTF + MTTR) = 1000let / (1000let + tyden)

**Příklad přednáška:**

Spočítejte dostupnost a spolehlivost newyorské burzy otevřené 250 dnů v roce, 8 hodin denně. Mezi 1980-90 měla 4 výpadky: 1 den kvůli sněhu, 4 hodiny kvůli požáru, 45 sekund kvůli SW chybě a 3 hodiny kvůli panice na burze.

- Jaká je její spolehlivost?
  - MTTF = 10*250*8 / 4 = 5000 h
  - MTTR = (1*24 + 4*24 + 0,75/60 + 3) / 4 ~ 30.75 h
  - MTBF = MTTR + MTTF
- Dostupnost?
  - MTTF / (MTTF + MTTR) ~ 0,9939
- Třída dostupnosti?
  - 2 – Managed 99%

# 1.2 Pokročilé modely

*aneb SP, NT a kamarádi*

## Flat a SoC

**Otázka:**

**Vysvětlete vlastnosti ACID transakcí.**

- atomicity - Transakce proběhne celá, nebo vůbec.
- consistency - Při provádění transakcí se DB převádí z jednoho konzistentního stavu do jiného.
- isolation - Transakce o sobe navzájem nevědí, jako by běžela jen jedna.
- durability - Změny commitnuté transakce jsou trvalé.

**Otázka:**

***Vysvětlete omezení transakčního modelu* flat transactions*, která mohou bránit jeho praktickému využití v některých transakčních systémech.***

- Mají příliš velký rozsah. Abortu po stotisící operaci se nedá nijak zabránit. To se dá přičíst tomu, že nemají žádnou vnitřní strukturu.
- Žádný dílčí neúspěch není tolerován. Pokud je špatně jediné číslo konta u deseti tisíc zaměstnanců nějakého podniku, pak transakce odeslání mezd všem zaměstnancům kvůli této chybě abortuje.
- Flat transakce nemohou spolupracovat s jinými transakcemi. Nemohou s nimi sdílet data (pro sdílení dat neexistují žádná primitiva nebo pravidla), nemohou být vzájemně propojeny přes signifikantní události.
- Neřeší přirozený problém vnoření do sebe. Běžné programovací jazyky umožňují vnořovat volání procedur, take se může stát, že zavoláme v transakci proceduru, která zahájí novou vnořenou transakci.

**Otázka:**

**Vysvětlete omezení transakčního modelu *spheres of control*, která mohou bránit jeho praktickému využití v některých transakčních systémech.**

- Zavislosti mohou byt velmi slozite
- neni dobre formalizovano
- tezke na implementaci

## Zápis modelů transakcí pomocí notace z Graye

### zápis pravidel

*<rule identifier>:<preconditions> -> <rule modifier list>, <signal list>, <state transition>*

kde rule modifier list je množina modifikačních pravidel oddělených čárkou - používají se 2 druhy pravidel:

*+(<rule identifier* | *signal>)*

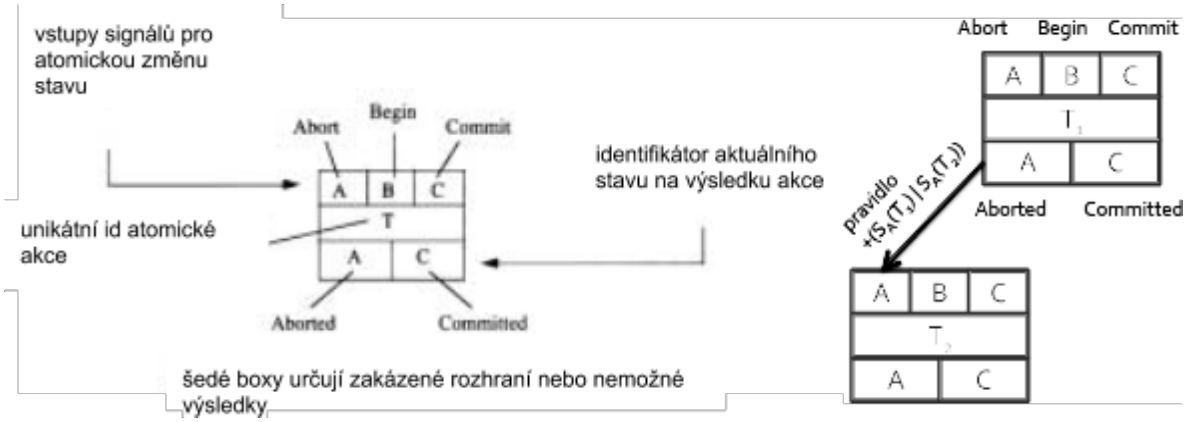
- význam: při použití pravidla *rule identifier* použij pravidlo *signal*

*delete(X)*

- význam: zrušení pravidla *X* - tj. pravidlo *X* se už nesmí používat

### grafická notace pro atomické akce

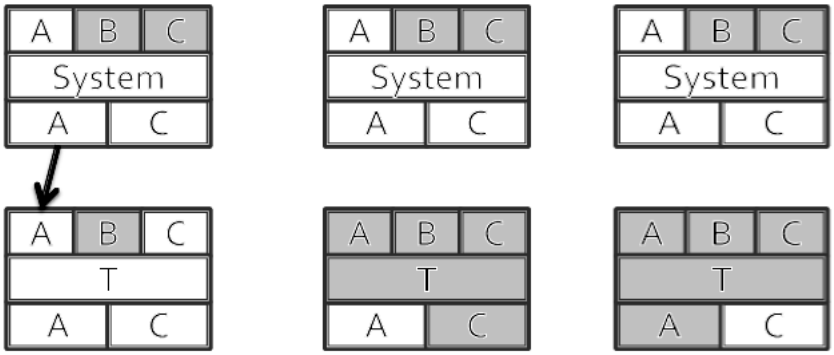
- pomocí ACID bloků = obdélníček se vstupy signálů, identifikací a výsledky




- význam:
  - zašeděné části ABC (tj. vstupní signály, které způsobují změnu stavu) jsou neaktivní signály (tj. signály, které už daná transakce nemůže přijmout)
  - zašeděné části AC (tj. stavy po provedení akce) jsou neaktivní stavy (tj. stavy, kterých už daná transakce nemůže nabýt) - většinou protože už nabyla ten druhý
  - šipky jsou triggry a jejich význam záleží na jejich počátku (odkud směřují) a konci (kam směřují):
    - z AC do ABC
      - začátek šipky určuje stav dané transakce (nebo systému), který trigger spouští
      - konec šipky určuje signál, který se dané transakci pošle
    - z AC do AC (viz commit HT)
      - začátek šipky určuje stav dané transakce (nebo systému), který trigger spouští
      - konec šipky určuje podmínku (stav transakce), která musí být splněna pro spuštění triggeru

**Příklad přednáška:**

**Flat model**



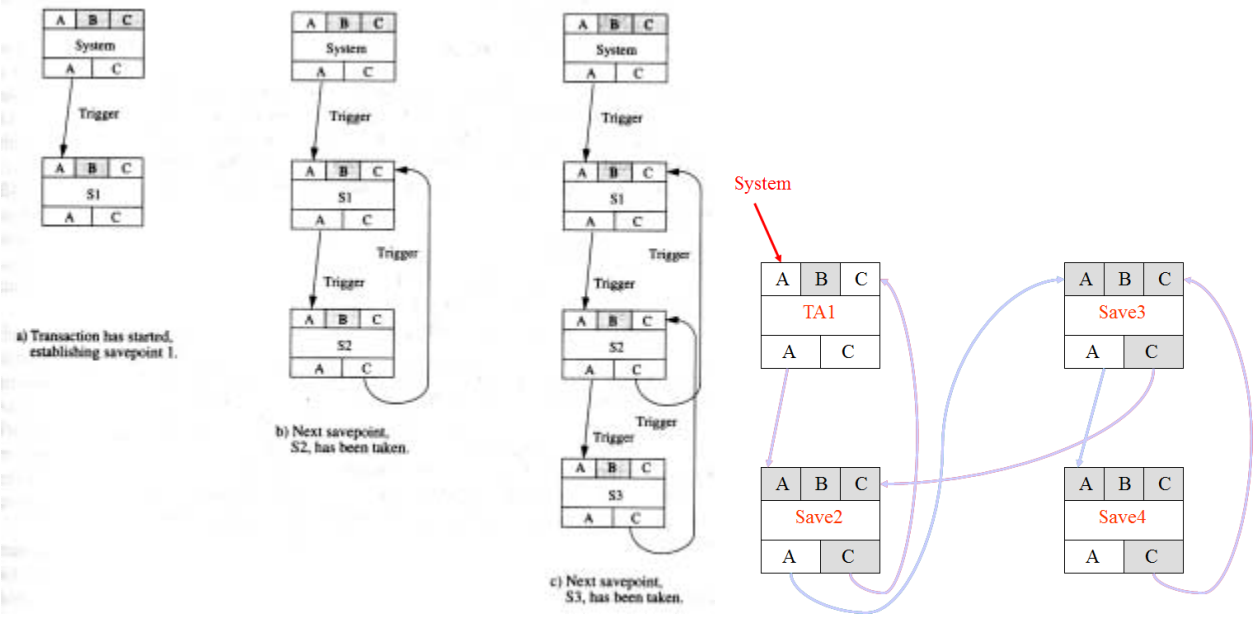

| | | |
|---|---|---|
| **transakce T je aktivní**<br>(pro begin pravidlo transakce T)<br>- šipka má význam “abort systému způsobí abort transakce T”<br>- systém nemůže začít (B) ani přijmout signál pro commit (C)<br>- transakce T nemůže začít (B) (tj. 1 transakci nemůže spustit 2x ve stejný čas) | **trans. T je ABORTovaná**<br>(pro abort pravidlo trans. T)<br>- všechny vstupy T jsou zakázané<br>- transakce T nemůže nic a je ve stavu aborted (A)<br>- systém nemůže začít (B) ani přijmout signál pro commit (C) | **transakce T je COMMITovaná**<br>(pro commit pravidlo transakce T)<br>- všechny vstupy T jsou zakázané<br>- transakce T nemůže nic a je ve stavu commited (C)<br>- systém nemůže začít (B) ani přijmout signál pro commit (C) |

**Pravidla**

| Podmínky | Rule modifiers | Signals | Změna stavu |
|---|---|---|---|
| B(T) | delete(B(T)), +(A(System) \| A(T)) | | Begin Work |
| A(T) | delete(C(T)) | | Rollback Work |
| C(T) | delete(A(T)) | | Commit Work |

## Savepointy (SP)

- podporuje MS SQL Server 2008 a Oracle
- obrazek v pravo ukazuje stav po Rollback(2)




- Všechny kroky mezi SaveWork příkazy jsou zahrnuty v atomické akci (Save1,Save2,...). Tyto atomické akce jsou spojeny commit, abort vztahy.
    - Pokud je jeden z nich abortován následující jsou také abortovány.
    - Pokud je 1 commitován předchozí jsou také commitovány.
- rollbacky nejsou persistentní (narušovalo by to atomicitu) => neřeší tvrdé výpadky

| Rule Id | Prec. | Rule modifiers | Signals | Význam |
|---|---|---|---|---|
| B(S1) | | +(A(System) \| A(S1)), delete(B(S1)) | | **Begin work** |
| A(S1,R) | R<=S1 | delete(C(S1)), delete(S(S1)) | | **Rollback Work**<br>flexibilní rollback - provádí se pouze k transakci R |
| C(S1) | | delete(C(S1,R)), delete(S(S1)) | | **Commit work** |
| S(S1) | | | B(S2) | S znamená vytvoření nového savepointu - to způsobí vytvoření navazující transakce S2 |
| ... | | | | |
| B(Sn) | | +(A(Sn-1) \| A(Sn)), delete(B(Sn)) | | **Begin work** |
| A(Sn,R) | R<=Sn | tady si nejsem jistej, ale kvůli dopředným rollbackům bych sem žádný delete nedával | A(Sn-1,R) | **Rollback Work**<br>při rollbacku dojde k rollbacku podtransakcí na základě triggeru (viz B(Sn)); navíc se musí poslat signál o abortu i rodičovi |
| C(Sn) | | delete(A(S1,R)), delete(S(S1)) | C(Sn-1) | **Commit work**<br>komit se provádí pouze v nejhlubší transakci a způsobí komit všeho => pošleme signál rodičovi |
| S(Sn) | | | B(Sn+1) | |

**Dopředný rollback**
Dopředný rollback u SP si představuju jako akci zpět a dopředu u prohlížečů.

Akce zpět způsobí klasický rollback k danému SP (tj. v prohlížeči se vrátit o X stránek zpět).
Akce dopředu způsobí, že se zruší rollback a vrátíš se k původně smazanému SP (tj. v prohlížeči jít o X stránek dopředu).

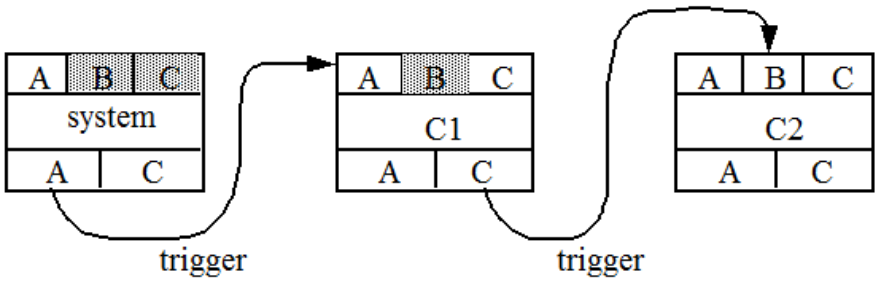
V praxi si myslím, že to funguje následovně (předpoklad: těsně před zavolání rollbacku se vytvoří SP):

SP1 ... SP2 ... SP3 ... SP4 ... SP5 + Rollback(SP2)
... SP6 ... Rollback(SP5)

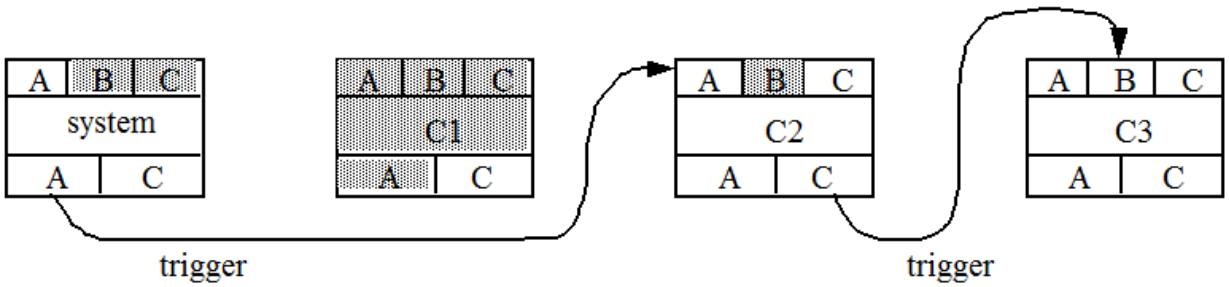
Vyzvětlivky:
- Rollback(SP2) způsobí vytvoření SP5 a zrušení efektu akcí mezi SP2 a SP5;
- Rollback(SP5) funguje jako dopředný rollback - tj. smaže se efekt akcí mezi SP2 a SP6 a znovu se projeví efekt akcí mezi SP2 a SP5.

Ale tadyto jsem nikde nevyčetl, je to spíš moje představa o tom, jak by to mohlo fungovat.

## Zřetězené transakce (ZT), Chained transactions (CT)




a) The first transaction in the chain has been started; start of the second one will be triggered by commit of the first one.

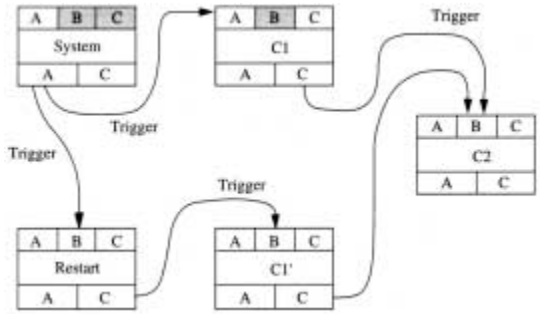



b) The first transaction in the chain has committed; the second one got started as part of commit of C1. Now the second one is structurally dependent on "system".

| Podmínky | Rule modifiers | Signals | Změna stavu |
|---|---|---|---|
| B(Cn) | delete(B(Cn)), +(A(System) \| A(Cn)) | | Begin Work |
| A(Cn) | delete(C(Cn)) | | Rollback Work |
| C(Cn) | delete(A(Cn)) | B(Cn+1) | Commit Work |

**<u>Otázka:</u> Porovnání SP a ZT**

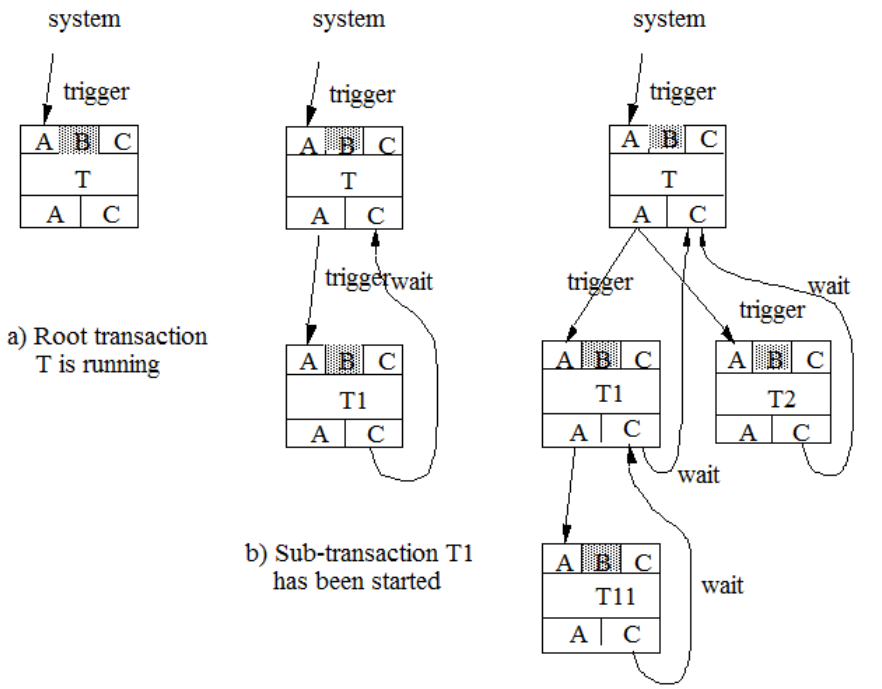
- Workflow
  - U obou po chybě není úplně všechna práce ztracena
  - Oba modely umožňují zachování pomocných struktur (např. kurzory)
- Commit vs. SP
  - U SP lze provést rollback ke kterémukoliv vytvořenému SP (flexibilita), u ZT k posl.commitu
  - Commit naopak umožní uvolnit zdroje (např. zámky)
- Restart
  - ZT model – odolný vůči systémovému abortu
  - Problém s rekonstrukcí stavu u SP i ZT




- k nedokončené transakci C1 se vytvoří po restartu transakce C1', která provede nedokončený zbytek transakce C1 a pak se normálně zřetězí s dalími transakcemi, které měly následovat po C1. Otázka je, kde vzít vstupy pro C1'.

## Nested transactions (NT), Hnízděné transakce (HT)

- výpočet jako hierarchie transakcí
  - podporuje MS SQL Server 2008




- **Pravidla pro podtransakce:**
  - **COMMIT pravidlo** - po potvrzení podtransakce jsou výsledky dostupné jen v rodičovské transakci, podtran. skutečně potvrdí až s potvrzením kořenové transakce
  - **ROLLBACK pravidlo** - pokud je podtransakce zrušena, bere sebou všechny své potomky (i když už lokálně potvrdili), když je zrušena kořenová transakce
  - **pravidlo viditelnosti změn** - zmeny v podtransakci jsou po lokálním COMMITu vidět jen v rodiči, objekty držené rodičem jsou přístupné potomkům, potomci jsou izolovaní v případě souběžného spuštění
  - rodič rozhoduje o řešení zrušení potomka (restartovat, zvolit alternativní výpočet, abortovat)

| Podmínky | Rule modifiers | Změna stavu |
|---|---|---|
| B(Tkn) | +(A(Tk) \| A(Tkn)), delete(B(Tkn)) | Begin Work |
| A(Tkn) | delete(C(Tkn)) | Rollback Work |
| C(Tkn) : C(Tk) | delete(A(Tkn)) | Commit Work |

**Otázka:**

**Srovnání flat a NT**

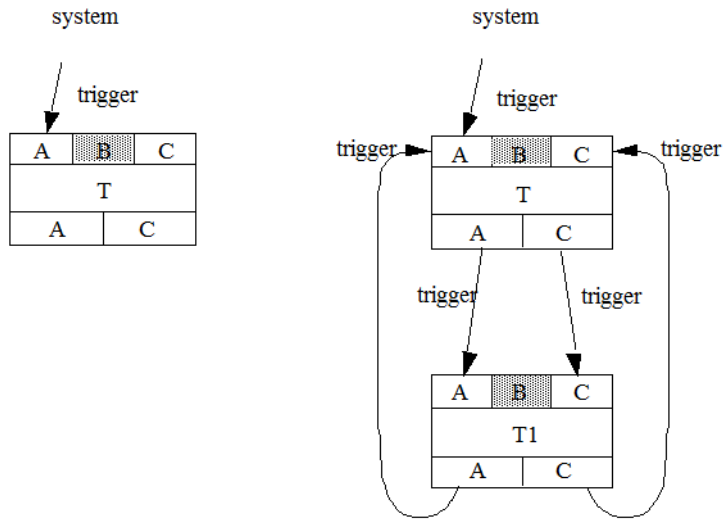
- A - podtr. jsou A jen z rodiče, C - podtr. jsou C vzhledem lok.fci., I - silná, T - neplatí kvuli commit prav.

**Otázka:**

**Emulace HT pomoci SP**

- SP pro každý uzel hierarchie, dokonce možnost vracet se k hotovým SP
- problém **paralelní podtransakce** - SP se řadí lineárně za sebou, rollback „jen přes některé“ není možný
- problém **zámky**
  - HT - rozhoduje o přidělení zámků rodič, a vrací se po potvrzení podtransakce
  - SP - patří všechno všem

## Distribuované Transakce

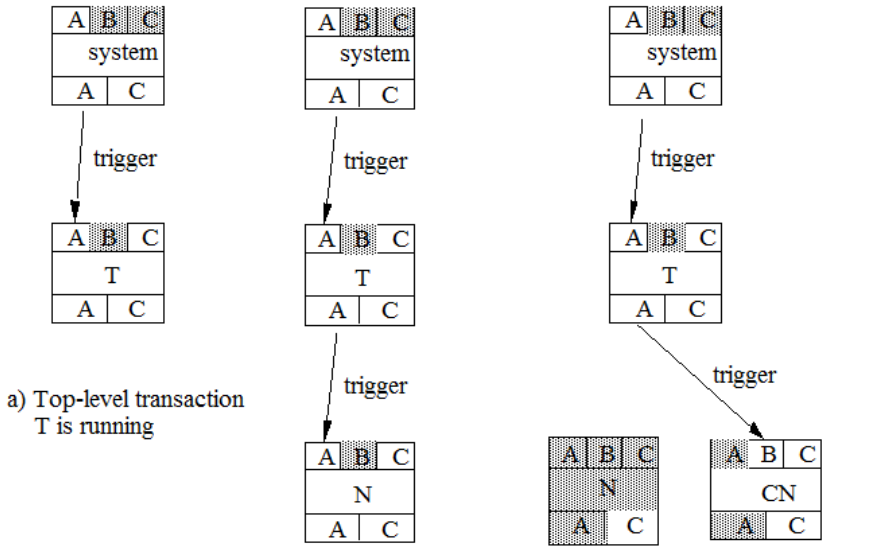



a) Root transaction T is running

b) Remote sub-transaction T1 has been started

## Multi-Level transakce (MLT)

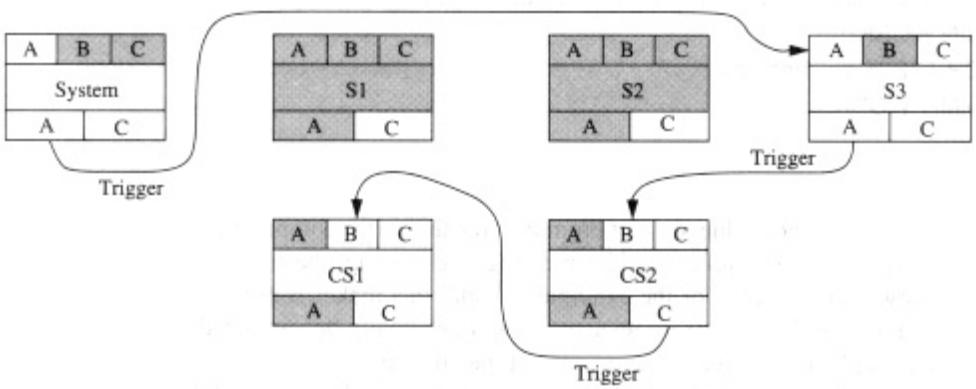
- pre-commit (uvolní zámky, zpřístupní výsledky) a kompenzační trans. CSi (pro ABORT)
  - kompenzační transakce musí vždy COMMITovat




- výhody: oproti HT mají více přístupných prostředků, jsou ACID
- **Open Nested Transactions**
  - COMMITy dětí jsou nezávislé na rodičích a naopak, unprotected actions (bez sfér vlivu)
  - vhodné pro client-server prostředí

## Ságy (vychází z MLT)

- řetězec(=Sága) transakcí Si a k nim odpovídající kompenzační trans. CSi (pro ABORT)
  - **Př:** kompenzační transakce např. mohou počítat penále za storno převodu peněz
  - kompenzační transakce musí vždy COMMITovat
- 2 možné průběhy:
  - posloupnost při COMMITu **celé** transakce: S1, S2,..., Sn
  - posloupnost při ABORTu transakce: S1, S2,..., Sj, ABORT, CSj-1,...,CS1




## Dlouho trvající transakce (DTT), Long-lived transactions (LLT)

Transakční kontext
Mini-Batch

## Rekapitulace

- Rozšíření flat ACID modelu pro konkrétní úlohy
    - Snaha o zachování korektně provedené práce
    - Podpora flexibilnějšího sdílené prostředků
    - Externalizace hotové práce
    - Ale stále jde o systémové zachování izolovanosti…

# 2. TP systémy

*aneb TP monitory*

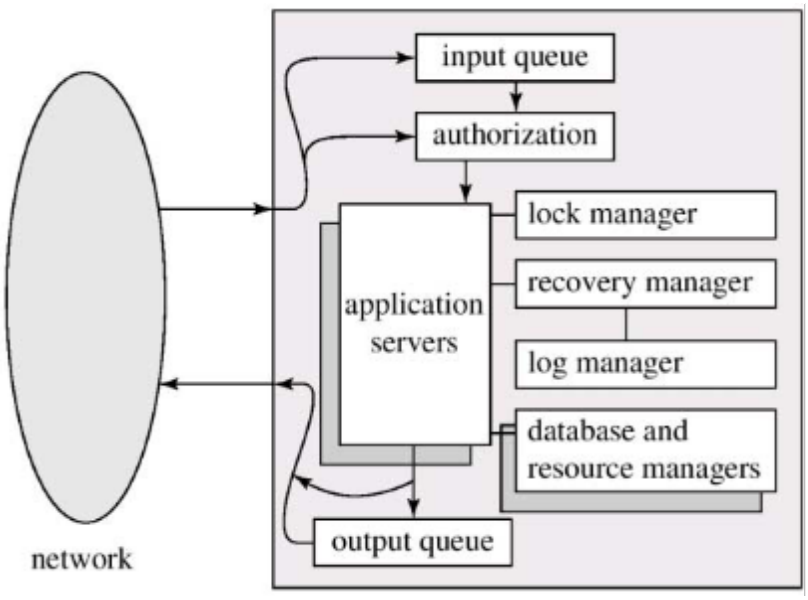
TP systém je systém zahrnující operační systém, sítě, jednu nebo více databází, generátory aplikací, ap. Současně obsahuje alespoň jeden TP monitor, který řídí práci s transakcemi. Transakcemi rozumíme ACID operace. Základními slubami (core services) TP-systému (tedy vlastně potamo transakčního monitoru) jsou:
− TRPC,
− Transakční Manager,
− Log Manager,
− Lock Manager.

Transakční monitor (nebo také TP monitor) se stará se o řízení a monitoring transakcí. Jeho jádrem je transakční manager (který se stará o vlastní řízení transakcí, o řízení jejich stavů, řízení commitů a restartů ap.), dále využívá služby log managera a lock managera.

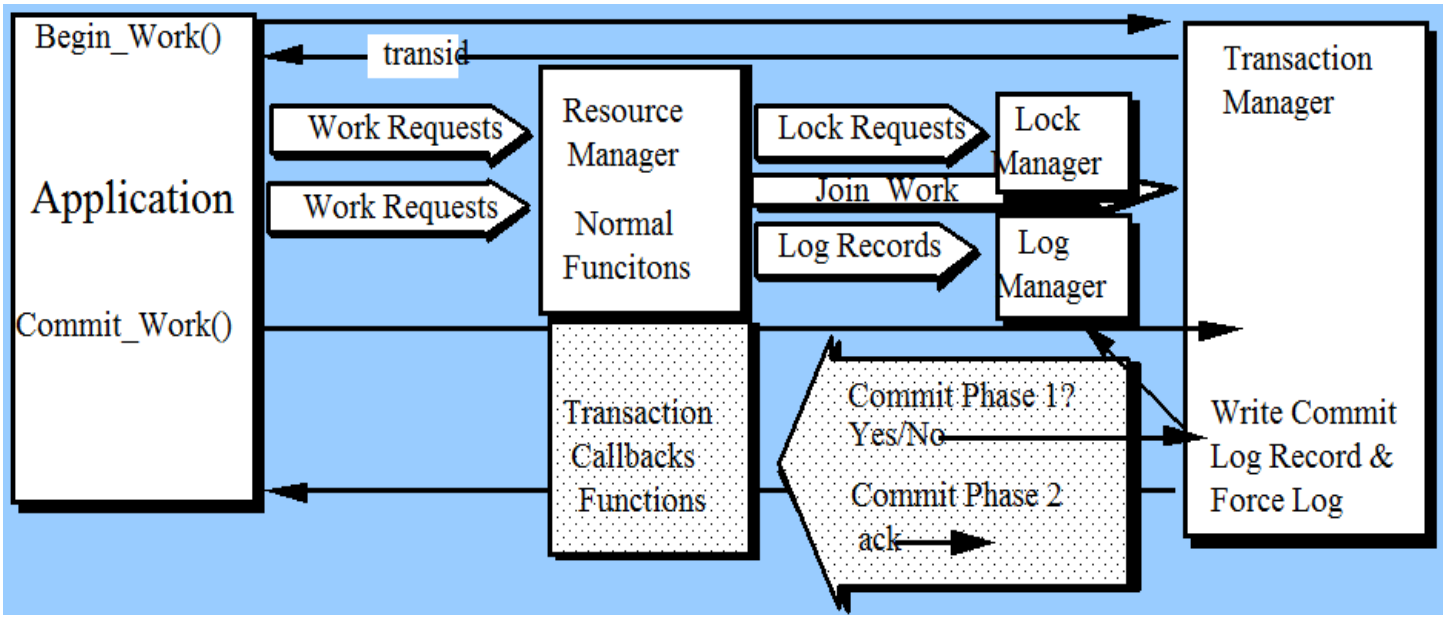
TPM (Transaction Processing Monitor) spojuje predevsim spojuje klienty a servery. Klientum umoznuje zakladat a ridit transakce. Pozadavky dispatchuje na servery. Servery jsou typicky startovany on demand a obsahuji pouze business logiku. Servery predavaji pozadavky Resource managerum. Commit (2PC) ridi opet TPM. TPM se stara jeste o dalsi veci - recovery, logging, load balancing (frontovani pozadavku) apod.

**Otázka:**

**Funkce TP-monitoru**




+ někam ještě TP-manažer


Begin_Work()
Application
Commit_Work()
transid
Work Requests
Work Requests
Resource Manager
Normal Funcitons
Lock Requests
Join_Work
Log Records
Lock Manager
Log Manager
Transaction Manager
Transaction Callbacks Functions
Commit Phase 1? Yes/No
Commit Phase 2
ack
Write Commit Log Record & Force Log

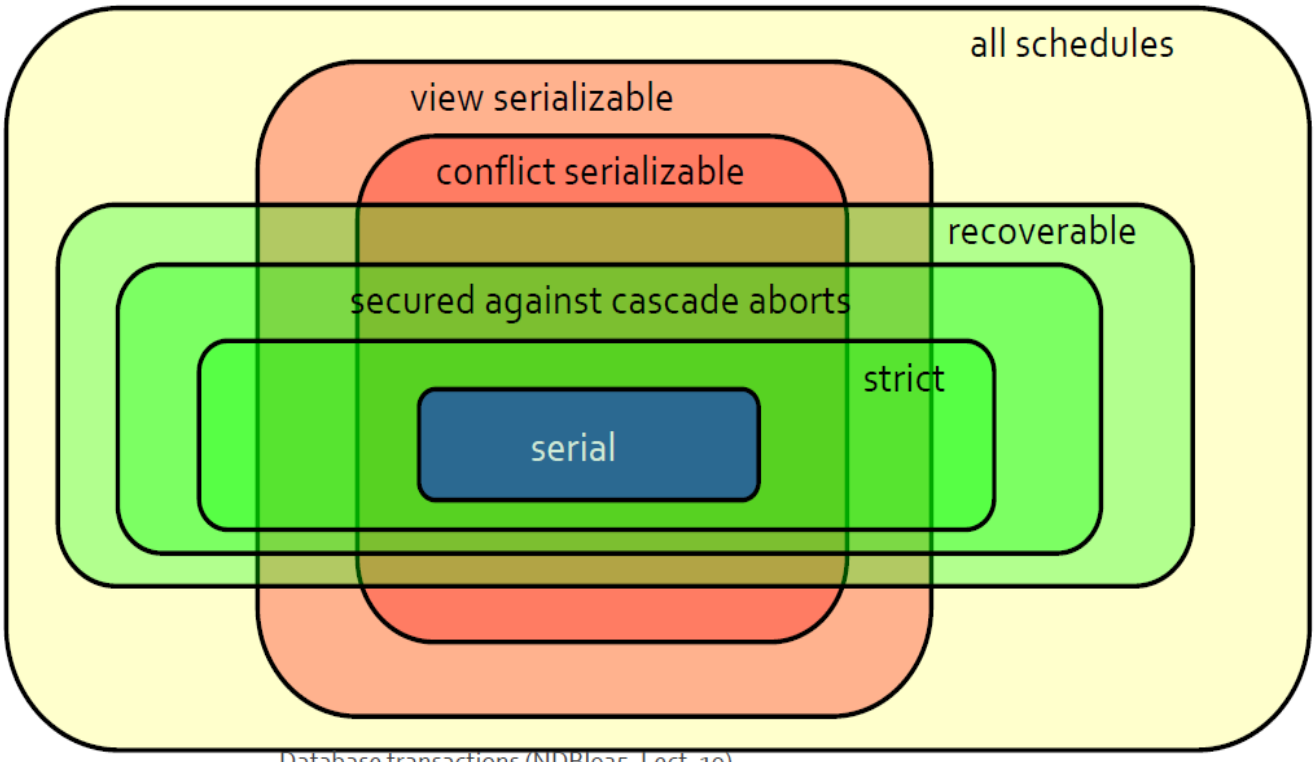
# 3. Teorie uspořádatelnosti

*aneb VSR ⊃ CSR a RC ⊃ ACA ⊃ ST*

### Definice

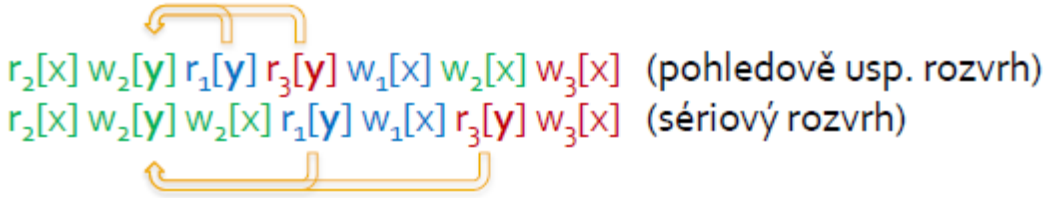
- vykonani rozvrhu vede ke konzistentnimu stavu databaze => je **usporadatelny**(=**serializovatelny**) (stav DB je stejný, jako po provedení nějakého SR)
    - je **pohledove usporadatelny (VSR)**, pokud je pohledove ekvivalentni se seriovym rozvrhem na stejnych transakcich.
      (čtou se stejná data jako v nějakém SR; detekce řešitelná, ale NP úplný problém)
        - rozvrhy S1, S2 jsou **pohledove ekvivalentni**, pokud splnuji nasledujici:
            - Pokud transakce Ti cte pocatecní hodnotu v S1, cte ji i v S2
            - Pokud transakce Ti cte hodnotu zapsanou Tj v S1, cte ji i v S2
            - Pokud transakce Ti provede posledni zapis hodnoty v S1, provede jej i v S2

          (Zaměření jen na r, w operace s DB, neřešíme aborty)
        - je **konfliktove usporadatelny (CSR)** <=> precedencni graf neobsahuje cykly.
          (stejné konfliktní dvojice jako nějaký SR)
- je **zotavitelny (RC)** <=> kdykoliv cte Ti hodnoty z Tj (i =/= j) a ci in H, pak cj < ci.
  (abort neovlivní data potvrzených transakcí)
    - **Příklad:** H = w1[x] r2[x] w2[y] c1 c2
    - predchazí **kaskadovemu abortu (ACA)** <=> kdykoliv cte Ti hodnotu x z Tj (i =/= j) v H, pak cj < ri[x].
        - je **striktní (ST)** <=> kdykoliv wj[x] < oi[x] (i =/= j), bud' aj < oi[x] nebo cj < oi[x], kde oi[x] in {ri[x], wi[x]}
          (čtení a zápis pouze potvrzených dat)
            - **Příklad:** H = w1[x] c1 w2[x] c2



**Příklad přednáška:**

**Pohledová ekvivalence**

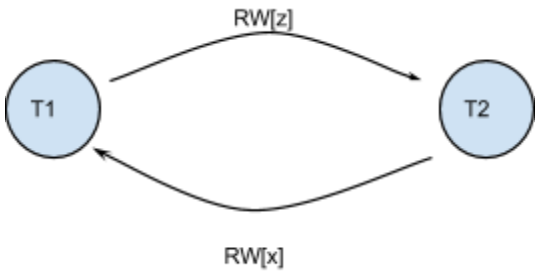



**Příklad:**

**Urcit vlastnosti zadaneho rozvrhu (usporadatelnost, zotavitelnost, odolnost vůči kaskádovému abortu, striktnost, ...): r1[x] r2[x] w1[x] r1[z] w2[z] r1[y] w1[y] c1 c2**

| T1 | T2 | |
|---|---|---|
| r[x] | | načtení 10 |
| | r[x] | načtení 10 |
| w[x] | | zápis 9 |
| r[z] | | načtení 10 |
| | w[z] | zápis 0 |
| r[y] | | načtení 10 |
| w[y] | | zápis 11 |
| c | | commit T1 |
| | c | commit T2 |

Seriové vykonání:
S(T1,T2) → x=9, y=11, z=0
S(T2,T1) → x=9, y=11, z=0




Paralelní vykonání:
P → x=9, y=11, z=0

- je usporadatelny, není pohledově uspořádatelný
- je zotavitelný, predchazi kaskadovemu abortu, je striktní

**Další příklady:**

- http://siret.ms.mff.cuni.cz/lokoc/UK/Transakce/Mikes-rozvrhy.pdf
- http://labts.troja.mff.cuni.cz/~machl5bm/zapisky/db/
- http://www.martinvseticka.eu/index.php?sekce=browse&page=200

**Příklad přednáška:**

**Uspořadatelnost**

| T1 | T2 | U1 | U2 |
|---|---|---|---|
| Read(U1, @x) | | 1000 | 1000 |
| Read(U2, @y) | | | |
| @x = @x – 100 | | | |
| @y = @y + 100 | | | |
| Write(U1, @x) | | 900 | 1000 |
| Write(U2, @y) | | 900 | 1100 |
| | Read(U2, @x) | | |
| | Read(U1, @y) | | |
| | @x = @x – 100 | | |
| | @y = @y + 100 | | |
| | Write(U2, @x) | 900 | 1000 |
| | Write(U1, @y) | 1000 | 1000 |

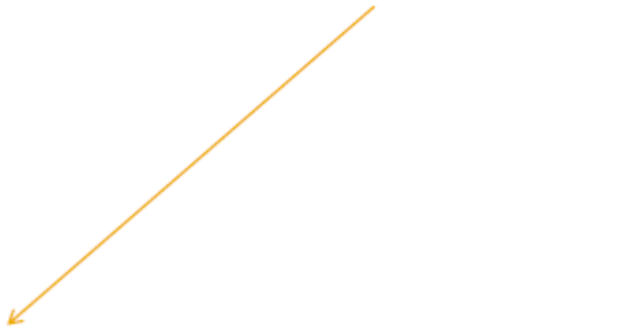
| T1 | T2 | U1 | U2 |
|---|---|---|---|
| | Read(U2, @x) | 1000 | 1000 |
| | Read(U1, @y) | | |
| | @x = @x – 100 | | |
| | @y = @y + 100 | | |
| | Write(U2, @x) | 1000 | 900 |
| | Write(U1, @y) | 1100 | 900 |
| Read(U1, @x) | | | |
| Read(U2, @y) | | | |
| @x = @x – 100 | | | |
| @y = @y + 100 | | | |
| Write(U1, @x) | | 1000 | 900 |
| Write(U2, @y) | | 1000 | 1000 |

Prokládané spuštění T1 a T2

| T1 | T2 | U1 | U2 |
|---|---|---|---|
| Read(U1, @x) | | 1000 | 1000 |
| Read(U2, @y) | | | |
| @x = @x – 100 | | | |
| @y = @y + 100 | | | |
| Write(U1, @x) | | 900 | 1000 |
| | Read(U2, @x) | | |
| Write(U2, @y) | | 900 | 1100 |
| | Read(U1, @y) | | |
| | @x = @x – 100 | | |
| | @y = @y + 100 | | |
| | Write(U2, @x) | 900 | 900 |
| | Write(U1, @y) | 1000 | 900 |

Rozvrh **není uspořádatelný**

Důsledek – z banky zmizí 100Kč

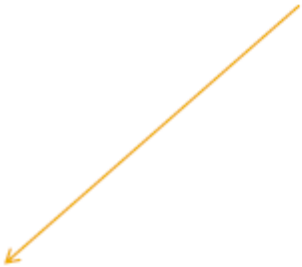


Jinak prokládané spuštění T1 a T2

| $T_1$ | $T_2$ | $U_1$ | $U_2$ |
|---|---|---|---|
| Read($U_1$, @x) | | 1000 | 1000 |
| Read($U_2$, @y) | | | |
| @x = @x – 100 | | | |
| @y = @y + 100 | | | |
| | Read($U_2$, @x) | | |
| | Read($U_1$, @y) | | |
| Write($U_1$, @x) | | 900 | 1000 |
| Write($U_2$, @y) | | 900 | 1100 |
| | @x = @x – 100 | | |
| | @y = @y + 100 | | |
| | Write($U_2$, @x) | 900 | 900 |
| | Write($U_1$, @y) | 1100 | 900 |

Rozvrh **není uspořádatelný**

Důsledek – sice nezmizí peníze,
ale zmizí efekt $T_1$ !!! (= lost update)



Stejně prokládané spuštění T1 a T2, ale teď T1 převádí 0 Kč

| T1 | T2 | U1 | U2 |
|---|---|---|---|
| Read(U1, @x) | | 1000 | 1000 |
| Read(U2, @y) | | | |
| @x = @x – 0 | | | |
| @y = @y + 0 | | | |
| | Read(U2, @x) | | |
| | Read(U1, @y) | | |
| Write(U1, @x) | | 1000 | 1000 |
| Write(U2, @y) | | 1000 | 1000 |
| | @x = @x – 100 | | |
| | @y = @y + 100 | | |
| | Write(U2, @x) | 1000 | 900 |
| | Write(U1, @y) | 1100 | 900 |

**Tj.** T1:Exec Úhrada(0, 'U1', 'U2');
T2:Exec Úhrada(100, 'U2', 'U1');

Rozvrh **je uspořádatelný!**

Důsledek – uspořádatelnost závisí i na hodnotách v programu

Proto není možné jednoduše a levně detekovat uspořádatelnost

Dále se proto omezíme pouze na operace s DB – Read a Write

| | R | W |
|---|---|---|
| R | ok | x |
| W | x | x |

# 4. Plánování běhu transakcí

*aneb Zámky a Protokoly vs. Fantom a Deadlock (Uváznutí)*

## Zámky

- **zámek** - atomická a levná operace
    - rl,ru - shared (S)
    - wl,wu - exklusive (X)
- **Dobře formovaná transakce** - Před r[x] je vždy rl[x] a před w[x] je vždy wl[x]
    - **Příklad:** rl[x] r[x] wl[y] w[y] ru[x] wu[y] c
- **Dvoufázová transakce** - Dvě fáze – 1. pouze zamykání a 2. pouze odemykání
    - **Příklad:** rl[x] r[x] wl[y] ru[x] w[y] wu[y] c
- **Legální historie** - Transakce dostane na objekt požadovaný zámek jen tehdy, je-li příslušný zámek k dispozici (podle tabulky konfliktů)
    - **Příklad:** wl2 [x] w2 [x] wu2 [x] rl1 [x] r1 [x] ru1 [x]

**Locking theorems**

- **Teorém 1** – Pokud jsou všechny transakce dobře formované a dvoufázové, pak je jakákoliv legální historie vzniklá z jejich konkurentního spuštění uspořádatelná.
    - **Důsledek T1** - dobře formované a dvoufázové transakce garantují korektní konkurentní běh těchto transakcí
- **Teorém 2** – Pokud není nějaká transakce dobře formovaná nebo není dvoufázová, je možné vytvořit další transakci (již dobře formovanou a dvoufázovou) takovou, že výsledný pár transakcí může vést k historii s cyklem v příslušném precedenčním grafu.
    - **Důsledek T2** - pokud není nějaká transakce dobře formovaná nebo není dvoufázová, tak to MŮŽE (ale nemusí) vést k problémům

## Predikátové zámky

- zámek je nastaven na záznamy vyhovující predikátu (např. WHERE Věk > 38), a konflikt nastává když se překrývají
- **Problémy:**
    - **náklady na realizaci** - hledání konfliktu je NP-úplný problém
    - pesimismus
    - **zdroj predikátu** - kdo ho vůbec vzít třeba při updatu?
    - => nejsou moc použitelné

## Rozsahové zámky

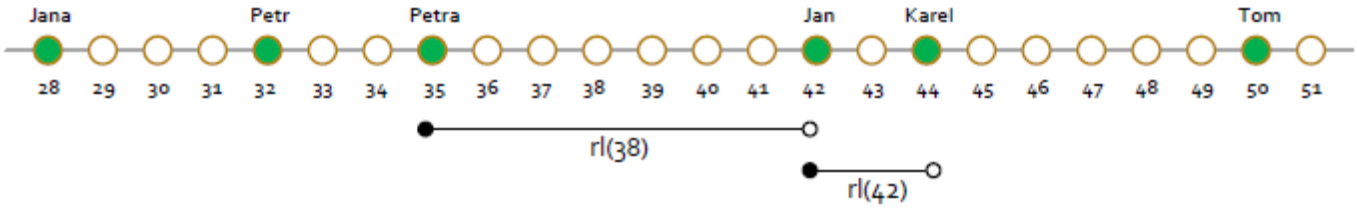
- zjednodušení predikátových na intervaly uspořádaných indexy

**<u>Otázka:</u>**

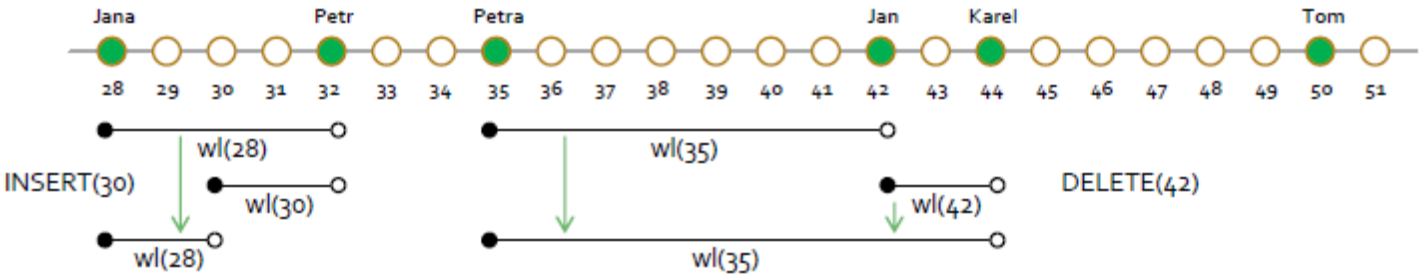
**Zamykani predchazejiciho/nasledujiciho klice - insert a delete pomoci zam. predch.**

- Key Range Locking, které pro posloupnost položek X, Y, Z a zámek na Y implikuje zámek na
    - (X, Y] (**Zamykání následujícího klíče**) – schopnost zamknout +∞, pro neexistující klíč zamkne první větší tj. ten *následující*
    - [Y,Z) (**Zamykání předcházejícího klíče**) – schopnost zamknout nejnižší hodnotu, pro neexistující klíč zamkne první menší tedy *předcházející*

- **načtení unikátního a následujícího záznamu** pomocí **previous locking**
    - Načtení **unikátního** x mezi w a y
        - Pokud x existuje, zamkne pro čtení x, což implikuje zámek [x, y)
        - Pokud x neexistuje, zamkne pro čtení w, což implikuje zámek [w, y)
    - Načtení **následujícího** y poté, co načte unikátní x mezi w a y
        - Již zamknuto pro čtení x, což implikuje existenci zámku [x, y)
        - Zamkne pro čtení y, což implikuje zámek [y, z)




- **vkládání a mazání** pomocí **previous locking**
    - **Vložení** x mezi w a y
        - Zamkne pro zápis w, což implikuje zámek [w, y)
        - Zamkne pro zápis x, což implikuje zámek [x, y)
        - Vloží x, změní se interval zámku z [w, y) na [w, x)
    - **Smazání** x mezi w a y
        - Zamkne pro zápis w, což implikuje zámek [w, x)
        - Zamkne pro zápis x, což implikuje zámek [x, y)
        - Smaže x, zámek na w pak implikuje interval zámku [w, y)




## Granulované zámky

- hrubé - delší čekání až seriový rozvrh
- jemné - větší režie zamykání, větší šance na deadlock

**Intent lock**

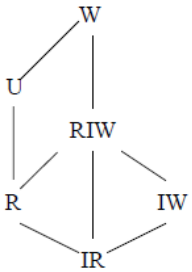
| *chci\mám* | *nic* | *irl* | *iwl* | *rl* | *riwl* | *ul* | *wl* |
|---|---|---|---|---|---|---|---|
| ***irl*** | ***yes*** | ***yes*** | ***yes*** | ***yes*** | ***yes*** | ***no*** | ***no*** |
| ***iwl*** | ***yes*** | ***yes*** | ***yes*** | ***no*** | ***no*** | ***no*** | ***no*** |
| ***rl*** | ***yes*** | ***yes*** | ***no*** | ***yes*** | ***no*** | ***no*** | ***no*** |
| ***riwl*** | ***yes*** | ***yes*** | ***no*** | ***no*** | ***no*** | ***no*** | ***no*** |
| ***ul*** | ***yes*** | ***yes*** | ***no*** | ***yes*** | ***no*** | ***no*** | ***no*** |
| ***wl*** | ***yes*** | ***no*** | ***no*** | ***no*** | ***no*** | ***no*** | ***no*** |

**Další příklady:**

- http://siret.ms.mff.cuni.cz/lokoc/UK/Transakce/michalko-GranulovaneZamky.pdf

## Konverze a eskalace zámků

- bere se supremum podle tohoto svazu (tedy: “rl[x] a pak wl[x]” = wl[x]):

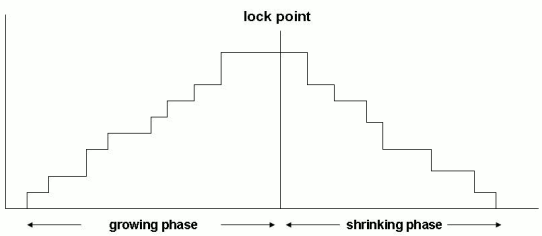


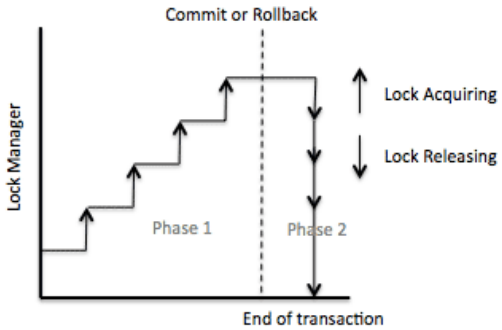

- **eskalace** -

## Protokoly

**Otázka:**

vlastnosti protokolů




| protokol | popis | konfliktová uspořadatelnost | zotavitelnost rozvrhu | proti kask. abortům | zabraňuje deadlocku |
|---|---|---|---|---|---|
| 2PL | nejdrive T jenom zamyká, pak jenom odemyká | ANO | NE | NE | NE |
| S2PL | uvolňuje zámky až po konci T (COMMIT / ABORT) | ANO | ANO | ANO | NE |
| K2PL (SSPL) | uzamykání pouze na začátku transakce, odem. na konci | ANO | ANO | ANO | ANO |

- **Vztahy**:
  - Gen(K2PL)⊂Gen(S2PL)⊂Gen(2PL)
  - Gen(S2PL) ⊆ CSR∩ST

**Příklad:**

Převod rozvrhu do S2PL pomocí WW (Wound/Wait strategie): r1[x] w1[y] r2[y] w2[z] w1[x] w2[y]

| T1 | T2 | |
|---|---|---|
| rl[x] | | |
| r[x] | | |
| wl[y] | | |
| w[y] | | T2 by chtěla rl[y] a má nižší prioritu => počká |
| wl[x] | | |
| w[x] | | |
| c | | |
| wu[y] | | |
| wu[x] | | |
| | rl[y] | |
| | r[y] | |
| | wl[z] | |
| | w[z] | |
| | wl[y] | |
| | w[y] | |

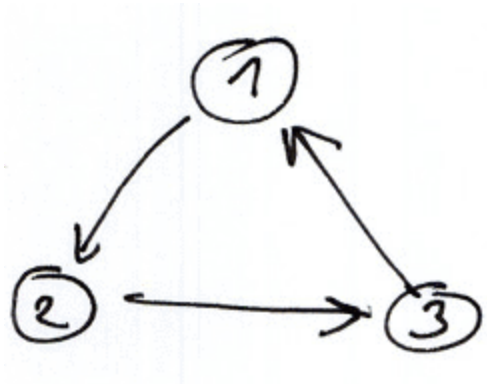
**Příklad DBI025:**

**Definujte 3 transakce na DB objektech A,B,C,D a sestavte rozvrh podle S2PL protokolu (dvoufázový) tak, aby obsahoval uváznutí (deadlock). K rozvrhu sestrojte graf pro detekci uváznutí.**

- chceme aby došlo k čekání transakcí, musíme tedy použít interpretaci zámků jako “žádost o uzamčení” (jinak k žádnému čekání nemůže dojít)

| T1 | T2 | T3 |
|---|---|---|
| rl[a] | | |
| | rl[a] | |
| | r[a] | |
| r[a] | | |
| wl[b] | | |
| w[b] | | |
| | wl[c] | |
| | w[c] | |
| | | wl[d] |
| | | w[d] |
| iwl[c] | | |
| | iwl[d] | |
| | | iwl[b] |

- teď vypadá graf detekce deadlocku:



## Deadlock (Uváznutí) a Fantom

**Otázka:**

**Prevence fantoma**

- **Fantom:** transakce pracuje s množinou entit a druhá jí během toho změní => nekonzistence DB
- Triviálně lze řešit zamknutím celé DB SR rozvrh
- V praxi se spíš řeší jemnějším zamykáním
    - Predikátové zámky
    - Rozsahové zámky
    - Granulované zámky

**Otázka:**

**Detekce, řešení vs. prevence deadlocku**

- **Deadlock:** transakce na sebe čekají v kruhu
    - **podmínky** pro Deadlock (musí být splněny všechny)
        - vzájemné vyloučení
        - držení prostředků
        - neodnímatelnost
        - čekání v kruhu
- **Detekce, řešení**
    - **detekce:** cyklus v grafu waits-for
    - abort + restart transakce z cyklu podle nějakého kriteria (např. nejnovější)
- **Prevence deadlocku**
    - K2PL
    - narušení jedné z podmínek
    - prioritizování/timestamp, T2 žádá lock T1:
        - **Wait/Die**
            - T2 dřívě spuštěná => T2 počká
            - T2 později spuštěná => T2 se abortuje a restartuje (se stejnou timestamp)
        - **Wound/Wait**
            - T2 dřívě spuštěná => abortuje a restartuje T1 (se stejnou timestamp)
            - T2 později spuštěná => T2 počká

## Příklad DBI025:

**Definujte 3 transakce na tabulce Pilot a sestavte rozvrh podle 2PL protokolu (dvoufázový) tak, aby obsahoval fantoma. K rozvrhu sestrojte graf pro detekci uváznutí (deadlock).**

- Máme stav databáze: {[1, Petr], [3, Karel]}

| T1 | T2 | T3 |
|---|---|---|
| | | |
| | INSERT(4, Xavier) | |
| uzamkni všechny piloty | | |
| S(Petr) | | |
| S(Karel) | | |
| S(Xavier) | | |
| M=MaxID{R(Petr), R(Karel), R(Xavier)} | | |
| | DELETE(4, XAVIER) | |
| INSERT(M) //do nejake jine tabulky | | |
| | **Commit** | |
| **Commit** | | |
| | | INSERT(2, Pavel) |
| | | **Commit** |

- Na transakci T3 vůbec nezáleží, je zde jen proto, aby bylo dodrženo zadání. V T1 nejsou zapsané unlocky, předpokládá se, že proběhnou v rámci COMMITu, ale mohou tam být klidně napsané.
- Stav databáze po provedení bude: [Xavier], což neodpovídá žádnému sériovému rozvrhu transakcí.

## Příklad DBI025:

**Pro dané transakce sestavte rovrh podle 2PL protokolu (dvoufázový) tak, aby obsahoval deadlock. Dokažte deadlock grafem pro detekci uváznutí.**

**Algoritmus:**

1. Nejprve sestavit konfliktově uspořadatelný rozvrh.
2. Pro každou transakci na začátku zamkneme jiný objekt (exkluzivně).
3. Dostatečně (ne)vhodně rozmístíme zbylé zámky.

## Příklad DBI025:

**Definujte 3 transakce na tabulce Zaměstnanec(ZID, Jméno) a sestavte rozvrh podle K2PL protokolu (dvoufázový) tak, aby obsahoval fantoma.**

- Lehce upravené zadání, je přidán *věk* v tabulce Zaměstnanec. Transakce T3 je zde jen kvůli zadání, mohla by se škrtnout a stejně by vznikl fantom.
- Stav databáze: {[Pepa, 52, m], [Jaroslav, 46, m], [Eva, 55, ž], [Dáša, 30, ž]}
- Transakce:

| **T1** | **T2** | **T3** |
|---|---|---|
| | | Insert(Xavier, 20, m) |
| | | COMMIT |
| | Insert(František, 72, m) | |
| uzamknu vsechny muze | | |
| S(Pepa) | | |
| S(Jaroslav) | | |
| S(Frantisek) | | |
| S(Xavier) | | |
| uzamknu vsechny zeny | | |
| S(Eva) | | |
| S(Dasa) | | |
| M = max{R(Pepa), | | |
| R(Jaroslav), | | |
| R(Frantisek), | | |
| R(Xavier)} | | |
| | Insert(Marie, 100, z) | |
| | COMMIT | |
| Z = max{R(Eva), | | |
| R(Dasa)} //tady o marii nevi | | |
| | | |
| Insert(M,Z) //vlozim do statistiky | | |
| COMMIT | | |

- Stav databáze bude: (František, Eva). Toto je však špatně, protože to neodpovídá ani sériovému rozvrhu T1, T2 (výsledek by byl: [Pepa, Eva]), ani T2, T1 (výsledek by byl: [František, Marie]), jak jsem psal výše T3 nic neovlivní.

**Může nastat uváznutí?**

- Uváznutí nastat nemůže, jelikož konzervativní 2PL je jeden ze způsobů prevence deadlocku.

**Zkonstruujte graf pro detekci uváznutí.**

- Graf uváznutí nikdy nebude obsahovat cyklus na všech vrcholech, nicméně by v zadání mělo být uvedeno k jakému momentu se má graf vytvořit, protože vrcholy grafu odpovídají aktivním transakcím!

## Další příklady:

- http://siret.ms.mff.cuni.cz/lokoc/UK/Transakce/Hlusickova-Deadlock.pptx

# 5. Alternativní protokoly

*aneb TO, PG, Certifikace, Int. plánovače a MV*

## Časová razítka - Time Ordering (TO)

**Otázka:**

**Zakladni TO pravidlo + rozvrh, ktery je generovany**

- **TO pravidlo** – konfliktní operace jsou seřazeny dle odpovídajících časových razítek
    - Pokud pi[x] a qj[x] jsou konfliktní operace, pak databáze zpracuje pi[x] před qj[x] právě tehdy, když ts(Ti) < ts(Tj)

- **Zakladni TO pravidlo**
    - Jednoduchá agresivní implementace TO pravidla
    - Plánovač přeposílá pi[x] z monitoru do DB (FIFO)
    - Posloupnost pi[x] qj[x] oi[x] která vede ke konfliktu poruší TO pravidlo - oi[x] požadavek přišel pozdě
    - V takovém případě (odmítnutí oi[x]) dojde k abortu Ti, Ti pak musí dostat nové časové razítko
- **Striktní TO pravidlo**
    - w-nepotvrzeno[x] je buď 0 nebo 1, protože zápis blokuje další zápisy
    - w-nepotvrzeno je nastaveno na 0 pouze tehdy, když plánovač obdrží od DB potvrzení o ci nebo ai
    - Když nastane ci nebo ai, w-nepotvrzeno[x] je nastaveno na 0 pro jakékoliv x pro které bylo wi[x] zasláno databázi transakcí Ti
- **Konzervativní TO pravidlo**
    - Pozdržet některé operace (čekat na operace starších transakcí )

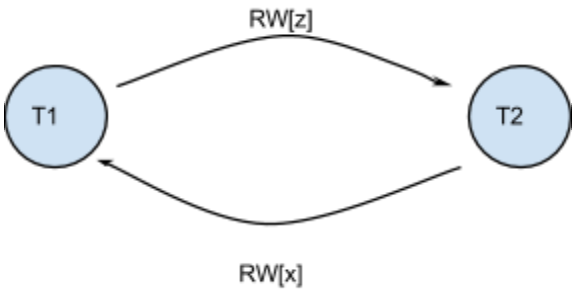
## Precedenční graf

**Příklad:**

**popsat jak funguje plánování pomocí precedenčního grafu a naplanovat zadany rozvrh pomoci precedencniho grafu: r1[x] r2[x] w1[x] r1[z] w2[z] r1[y] w1[y] c1 c2**

- **základní** - vkládáme nepotvrzené transakce jako uzly, odebíráme ukončené
    - přijmeme operaci pi[x] z monitoru, pro každou operaci s kterou je pi[x] v konfliktu vložíme hranu
        - pokud vznikne cyklus abortujeme Ti (a smažeme ho z grafu i s hranami!)
        - pokud nevznikne plánujeme pi[x] po COMMITu všech starších op. konfliktních s ním
- **konzervativní** - namísto abortování pozdržuje operace
- (obě verze vytvářejí uspořádatelné historie, ale ne RC,ACA a ST)

| T1 | T2 | |
|---|---|---|
| r[x] | | |
| | r[x] | |
| w[x] | | |
| r[z] | | |
| | w[z] | pozor cyklus! |
| | abort | |
| r[y] | | |
| w[y] | | |
| c | | |
| | restart T2 | |




## Certifikace

- optimistické plánování
- Každá operace je ihned odeslána DB, Korektnost se testuje až při COMMITu
- Algoritmy
    - 2PL
    - TO -
    - PG

## Integrované plánovače

- Plánování RW, WR a WW
- **Thomas Write pravidlo (TWR)**
    - 

## Multi-version (MV)

**Otázka:**

**MV a “one copy” historie?**

**Otázka:**

**Definice komplet MV (multi-version) historie + jejich eq.**

**MV historie** je podobná jednoverzové, až na

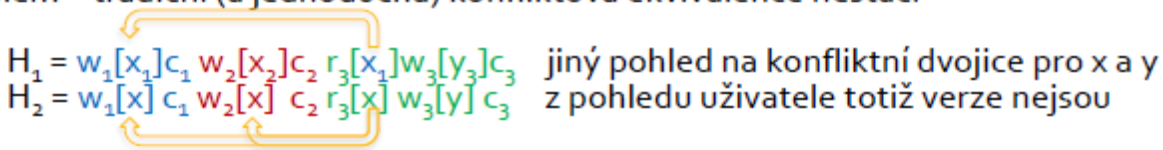
- wi[x] nové transakce Ti vždy vytvoří novou verzi značenou xi
- ri[x] transakce Ti vždy specifikuje, jakou verzi číst, značeno ri[xj]

**Kompletní MV historie (CMV)** je MV historie, kde

- Každé ri[xj] je předcházeno wj[xj] – **čtení existující verze**
- Každé ri[x] po wi[xi] je ri[xi] – **čtení poslední verze v rámci Ti**
- Každá operace ci po ri[xj] je předcházena operací cj – **zotavitelnost**

**Ekvivalence**

- Opět nás zajímá taková MV historie, která je „nějak“ ekvivalentní nějaké 1V sériové historii
    - Problém – tradiční (a jednoduchá) konfliktová ekvivalence nestačí




    - Pouze $w_1[x_1]$ a $r_3[x_1]$ tvoří konfliktní dvojici $H_1$ (jiná verze = jiná proměnná!)
    - Pořadí $w_1[x]$ a $r_3[x]$ je přitom stejné jako v 1V sériové $H_2$
    - Ale $r_3$ v $H_1$ čte jinou hodnotu x než $r_3$ v $H_2$ – tj. stejné konfliktní dvojice nestačí
- Proto se musí použít pohledová ekvivalence (rozlišuje odkud se čte)
- Budou nás zajímat „**one copy**“ sériové MV historie (1SMV), kde 1SMV je historie, která je pohledově ekvivalentní nějaké 1V sériové historii
- Problém – nelze přímo porovnávat MV a 1V historie

- Pohledově ekvivalentní MV historie
    - Obsahují stejné operace
    - Čtou stejné verze dat

- U pohledové ekvivalence v 1V historii jsou ještě potřeba stejné finální zápisy, ale v MV jsou vlastně všechny finální

- Specifikováním verze čtených dat v MV historii se redukuje problém ekvivalence MV historií na triviální požadavek , a to, že ekvivalentní MV historie mají stejné operace

# 6. TP manažer a zotavení systému

*aneb není moc vypracováno :)*

**Otázka:**

**ARIES (Algorithms for Recovery and Isolation Exploiting Semantics)**

*- no a aries jsem vzala poněkud volně - patří do něj ten základní princip zotavování - a vlastně cokoliv jsem si vzpomněla na cca osmi slidech jsem tam nacpala, pak jsem řekla, že aries je totéž, ale lepší :)*

Algoritmus navržen pro přístup **no-fix/no-flush**

- COMMIT - zapíše do logu informace o potvrzení, plánovači pošle zprávu o COMMIT
- ABORT - buffer manažer mohl zapisovat kdykoliv => udělá se UNDO transakce, plánovači pošle zprávu o ABORT
- Zotavení - částečné REDO a globální UNDO od začátku logu

Využívá mnoho optimalizací pro větší efektivitu

Hlavní principy ARIES

- **W**rite **A**head **L**ogging – do logu by se mělo zapsat vždy dříve, než se provede a uloží samotná změna objektu do perzistentní paměti
- Opakování historie během REDO - při restartu po pádu ARIES znovu projede akce před pádem a vrátí systém do stavu před pádem, pak udělá UNDO na transakce aktivní během pádu (nestihly se potvrdit)
- Logování změn během UNDO (neopakování undo operací během opakovaného pádu systému) - zmeny v databázi během undování transakcí se logují aby se neopakovaly při případných opakovaných restartech

# 7. Distribuované transakce

*aneb není moc vypracováno :)*

## 2PC protokol

1. prepare - každy Resource Manager odpoví jestli je připraven na commit
2. ABORT/commit