Kontrola vzporednosti

Iztok Savnik, FAMNIT

Prosojnice & učbenik

- Učbenik:
  - Raghu Ramakrishnan, Johannes Gehrke, *Database Management Systems, McGraw-Hill, 3rd ed., 2007.*
- *Prosojnice:*
  - *From „Cow Book“: R.Ramakrishnan, http://pages.cs.wisc.edu/~dbbook/*

Konfliktno uredljivi razporedi

- Dva razporeda sta konfliktno ekvivalentna, če:
  - Vsebujta enake akcije na istih razporedih.
  - Vsak par konfliktnih akcij je urejen na isti način.
- Razpored S je uredljiv po konfliktih, če je S konfliktno ekvivalenten nekem zaporednem razporedu.

Oba razporeda sta konfliktno uredljiva.

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

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

Razpored, ki ni uredljiv po konfliktih.

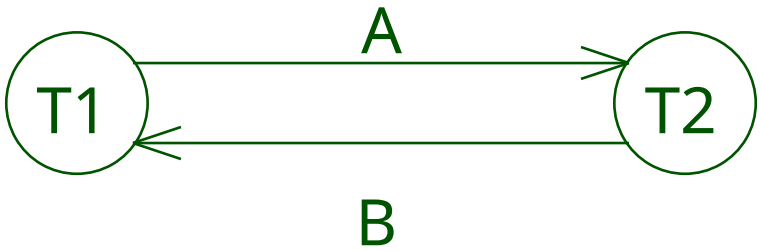
T1: R(A) W(A)
T2 W(A)
T3: W(A)

Kasneje bomo videli, da je uredljiv po oknih.

Primer

- Razpored, ki ni uredljiv po konfliktih:

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



Graf odvisnosti

- Cikel v grafu odkrije problem. T1 je odvisna od T2, in obratno.

Graf odvisnosti

- Graf odvisnosti:
  - Eno vozlišče za Xact.
  - Povezava od *Ti do Tj, če akciji Ti sledi konfliktna akcija Tj.*
- Izrek: Razporeditev je zaporedno uredljiva po konfliktih, če je graf odvisnosti acikličen.

Primeri

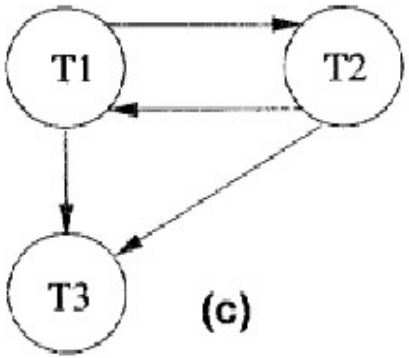
$T1$	$T2$	$T3$
$R(A)$		
	$W(A)$	
	Commit	
$W(A)$		
Commit		
		$W(A)$
		Commit

$T1$	$T2$
$X(A)$	
$R(A)$	
$W(A)$	
$X(B)$	
$R(B)$	
$W(B)$	
Commit	
	$X(A)$
	$R(A)$
	$W(A)$
	$X(B)$
	$R(B)$
	$W(B)$
	Commit

$T1$	$T2$
$S(A)$	
$R(A)$	
	$S(A)$
	$R(A)$
	$X(B)$
	$R(B)$
	$W(B)$
	Commit
$X(C)$	
$R(C)$	
$W(C)$	
Commit	

(a)

(b)



(c)

Zaporedna uredljivost po oknih

- Razporeda S1 in S2 sta "ekvivalentna po oknih" če:
  - Če Ti prebere začetno vrednost A v S1, potem Ti prebere začetno vrednost A v tudi S2
  - Če Ti prebere vrednost A, ki jo napiše Tj v S1, potem Ti prebere vrednost A, ki jo napiše Tj tudi v S2
  - Če Ti napiše končno vrednost A v S1, potem Ti napiše končno vrednost A tudi v S2

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

```
T1: R(A),W(A)
T2:             W(A)
T3:                 W(A)
```

Uredljivost po oknih

- Uredljivost po konfliktih je zadosten, ne pa vedno potreben pogoj za zaporedno uredljivost.
- Razpored je zaporedno uredljiv, če je zaporedno uredljiv po oknih.

Dosledno dvo-fazno zaklepanje

- Protokol doslednega 2FZ
  - Vsaka Xact mora pridobiti S (deljen) zaklep na objektu pred branjem in X (ekskluzivni) zaklep na objektu pred pisanjem.
  - Vsi zaklepi transakcije so sproščeni na koncu transakcije.
  - Če transakcija drži X zaklep na objektu, potem nobena druga transakcija ne more dobiti zaklepa (S ali X),
- Dosleden 2FZ dovoljuje samo razporede, katerih graf odvisnosti je acikličen.
- T1 in T2 se izvajata vzporedno
  - Če ne dostopata do istega objekta potem se izvajata neodvisno; če dostopata do istih objektov se ureditve akcij zaradi zaklepanja začnejo urejati zaporedno; ena izmed transakcij bo morala počakati, da druga izvede vse akcije.

Dosledno 2FZ

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

$T1$	$T2$
$X(A)$	
$R(A)$	
$W(A)$	
$X(B)$	
$R(B)$	
$W(B)$	
Commit	
	$X(A)$
	$R(A)$
	$W(A)$
	$X(B)$
	$R(B)$
	$W(B)$
	Commit

$T1$	$T2$
$S(A)$	
$R(A)$	
	$S(A)$
	$R(A)$
	$X(B)$
	$R(B)$
	$W(B)$
	Commit
$X(C)$	
$R(C)$	
$W(C)$	
Commit	

Dvo-fazno zaklepanje

- Nedosledno dvo-fazno zaklepanje
  - Vsaka Xact mora pridobiti S (deljen) zaklep na objektu pred branjem in X (ekskluzivni) zaklep na objektu pred pisanjem.
  - Objektov ni več možno zakleniti po tem, ko se sprosti katerikoli zaklep.
  - Če transakcija drži X zaklep na objektu, potem nobena druga transakcija ne more dobiti zaklepa (S ali X),
- Kaskadne prekinitve
  - Tudi T2 mora biti prekinjena

T1	T2
X(A),W(A)	
X(B),R(B)	
U(A)	
	X(A),R(A),W(A)
Abort	
W(B),U(B)	...
Commit	

$T1$	$T2$
$R(A)$	
$W(A)$	
	$R(A)$
	$W(A)$
	$R(B)$
	$W(B)$
	Commit
Abort	

Prekinitev transakcije

- Če je transakcija *Ti* prekinjena potem morajo biti vse njene akcije izničene.
  - Ne samo to; če *Tj* bere objekte, ki jih je napisala *Ti*, potem mora biti prekinjena tudi *Tj!*
- Večina sistemov se izogne kaskadnim prekinitvam s sprostitvijo zaklenjenih objektov tik pred potrditvijo.
  - *Ti* popravi objekt; *Tj* ga lahko prebere šele po potrditvi *Ti.*
- SUPB zapisuje kompleten dnevnik vseh popravkov.
  - Prekinjene transakcije je mogoče izničiti.
  - Isti mehanizem se uporablja za okrevanje po zrušitvi.

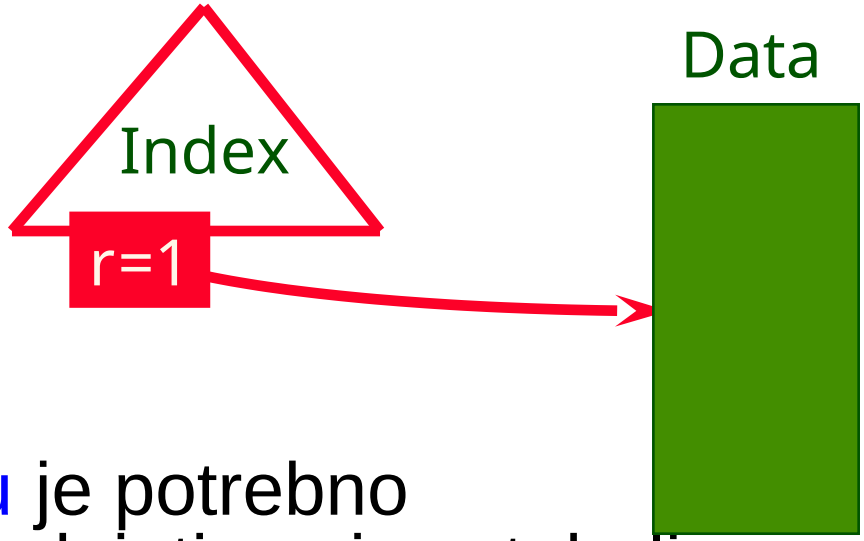
Dinamične podatkovne baze

- Fantomski problem
- Če sprostimo predpostavko, da je PB fiksna kolekcija objektov potem tudi striktno 2FZ ne zagotovi serializabilnosti:
  - T1 zaklene vse strani, ki vsebujejo zapise mornarjev z *rating* = 1 in poišče najstarejšega mornarja (npr., *age* = 71).
  - Naprej, T2 vstavi novega mornarja; *rating* = 1, *age* = 96.
  - T2 zbriše tudi najstarejšega mornarja z rating = 2 (*age* = 80) ter naredi commit.
  - T1 zdaj zaklene vse strani, ki vsebujejo zapise mornarjev z *rating* = 2 in poišče najstarejšega (npr., *age* = 63).
- Nekonsistentno stanje PB, če je T1 “korektna”!

Problem

- T1 implicitno predpostavlja, da je zaklenila celotno množico mornarjev za katere velja *rating* = 1.
  - Predpostavka velja samo, če se ne dodaja novih zapisov mornarjev medtem, ko se T1 izvaja!
  - Potrebujemo nek mehanizem, ki to zagotovi. (Zaklepanje indeksov in predikatov.)
- Primer pokaže, da zaporedna uredljivost po konfliktih garantira uredljivost samo, če je množica objektov fiksna!
- Poglejmo si bolj podrobno kako se objekti zaklenejo

Zaklepanje indeksa



- Pri sekvenčnem skeniranju je potrebno zagotoviti, da se ne sme dodajati zapisov tabeli
  - T1 mora zakleniti vse strani kot tudi datoteko/tabelo, da bi onemogočili dodajanje zapisov z *rating* = 1.
- Če imamo indeks za atribut *rating* z alternativo (2), potem bi T1 morala zakleniti indeksne strani z indeksnimi vpisi z *rating* = 1.
  - Če ni zapisov z *rating* = 1, potem mora T1 zakleniti indeksne strani kjer bi bili takšni podatki, če bi obstajali !

Zaklepanje predikatov

- Zaklep vseh zapisov, ki ustrezajo pogoju logičnega predikata, npr. *age > 2*salary*.
  - Zaklepanje indeksa je poseben primer zaklepanja predikata.
  - Indeks podpira učinkovito implementacijo zaklepanja predikata.
  - Kaj je predikat v primeru mornarjev?
- V splošnem vsebuje zaklepanje predikatov veliko dodatnega dela.
  - Zaradi tega se običajno ne uporablja.

Upravljanje zaklepanja

- Zahteve po zaklepanju in odklepanju izvršuje upravitelj zaklepanja.
  - Zaklepanje in odklepanje mora biti atomična operacija.
- Tabela transakcij
  - Zapis transakcij vsebuje vse podatke o transakciji.
  - Tudi seznam zaklepov, ki jih drži transakcija.
- Tabela zaklepanja
  - Razpršilna tabela indeksirana z identifikatorjem objekta.
  - Zapis TZ predstavlja zaklep: tabela, stran, zapis, …; število transakcij, ki drži zaklep; tip zaklepa S|X; kazalec na vrsto zaklepov.

Zaklep in odklep

- Ko Xact potrebuje dostop do objekta O
  - Zahteva po zaklepu (S ali X)
- Implementacija zaklepanja in odklepanja.
  - If S zahteva & vrsta zaklepov prazna & ni X zaklepa.

    Then (*O ima še vedno lahko S zaklep*) S zaklep se dodeli & popravi zapis tabele zaklepanja (S zaklep => inc # S zaklep).
  - If X zahteva & ni drugih zaklepov (=> vrsta je prazna)

    Then X zaklep se dodeli.
  - Otherwise, zahteva po zaklepu se ne dodeli; vstavi se v vrsto O.

Zaklep in odklep

- Po zaklučeni transakciji se zaklepi sprostijo
  - Ustrezni zapisi tabele zaklepov se popravijo
  - Iz vrste zaklepov se vzame in izvede naslednja zahteva po zaklepu
  - Če je v vrsti več S zaklepov se obravavajo vsi naenkrat
  - Zahteve za zaklep objekta O se morajo obravnavati časovno po vrsti (npr. S, X, S), sicer pride do stradanja
- Zaklep in odklep morata biti atomični operaciji
  - Upravitelj zaklepanja se izvaja vzporedno ⇒ za dostop do objekta uporabljeni sinhronizacijski mehanizmi OS (semafor)

Sprememba zaklepa

- Stopnjevanje zaklepa
  - Transakcija zahteva X zaklep, ko že ima S zaklep (dobi, če druge ne držijo zakl.)
  - Vstavi na čelo vrste zaklepov (dajanje prednosti drugim povzroči smrtni objem);
  - Ta pristop daje prednost tistim, ki že imajo S zaklep.
  - Problem se pojavi, če bi več transakcij želelo stopnjevati zaklep (prišlo bi do smrtnega objema).
- Zniževanje zaklepa
  - Boljši pristop: prvo dobi X zaklep potem znižaj; nižanje zaklepa ne krši 2FZ, če transakcija ni spremenila objekta; pristop niža vzporednost vendar poveča pretok, ker ni smrtnih objemov (uporaba v komercialnih sistemih).

Smrtni objemi

- Smrtni objem:
  - Cikel transakcij, ki čakajo druga drugo na odklep virov.
- S smrtnim objemom delamo na dva načina:
  - Preprečevanja smrtnega objema
  - Detekcija smrtnega objema

Preprečevanje smrtnega objema

- Določi prioritete glede na časovne zaznamke. Ti želi zaklep, ki ga drži Tj. Dve možni politiki:
  - Počakaj-Umri: Če ima Ti višjo prioriteto potem Ti počaka na Tj; sicer Ti prekine izvajanje.
  - Rani-Čakaj: Če ima Ti višjo prioriteto potem Tj prekine izvajanje; sicer Ti počaka.
- Če se transakcija ponovno starta je potrebno zagotoviti, da ima originalno časovno oznako.

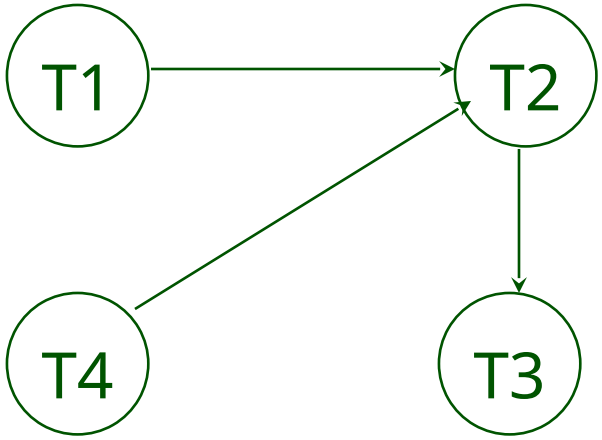
Odkrivanje smrtnega objema

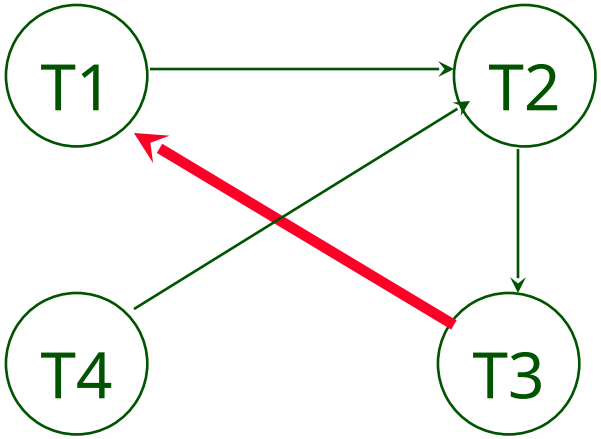
- Kreiraj čakalni graf:
  - Vozlišča so transakcije
  - Obstaja povezava od Ti do Tj, če Ti čaka na Tj za sproščanje zaklepa.
- Periodično preveri ali ima čakalni graf cikle.

Odkrivanje smrtnega objema

Primer:

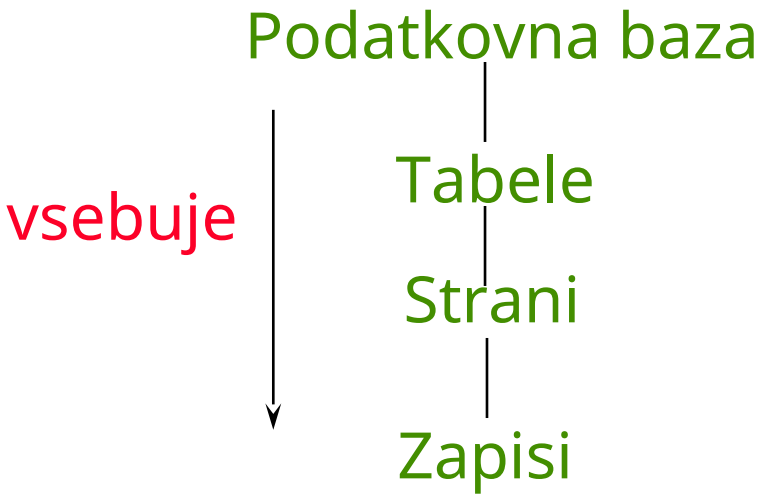
T1: S(A), R(A), S(B)
T2: X(B),W(B) X(C)
T3: S(C), R(C) X(A)
T4: X(B)





Večnivojsko zaklepanje

- Težko se je odločiti kateri nivo zaklepati (zapisi vs. strani vs. tabele).
- Nebi se smeli odločati o tem!
- Podatkovni “kontejnerji” so vgnezdeni:



Rešitev: Nova stanja zaklepanja, protokol

- Dovolimo Xact zaklepati vsak nivo
  - Uporaba protokola, ki vsebuje "namere" zaklepanja
  - Določen zaklep objekta (IS, IX, S, SIX ali X) velja objekt in za vse podrejene objekte!

- Pred zaklepom objekta mora Xact postaviti "namero" zaklepa na vseh predhodnikih.
- Pri odklepanju gremo od specifičnega proti splošnem (od-spodaj-navzgor).
- SIX način: Kot S & IX skupaj.

	--	IS	IX	S	X
--	√	√	√	√	√
IS	√	√	√	√	
IX	√	√	√		
S	√	√		√	
X	√				

Večnivojski protokol zaklepanja

- Vsaka Xact začne iz korena hierarhije.
- Da bi dobili S ali IS zaklep vozlišča moramo imeti IS ali IX na starših.
  - Kaj če Xact drži SIX na staršu? S na staršu?
- Da bi dobili X ali IX ali SIX zaklep vozlišča moramo imeti IX ali SIX na starših.
- Moramo sproščati zaklepe od spodaj navzgor.

Protokol je korekten ker je ekvivalenten protokolu, ki dela direktno z listi.

Primeri

- T1 pregleda R in popravi nekaj n-teric:
  - T1 dobi SIX zaklep na R, potem lahko dobi S zaklepe na n-tericah R in občasno spremeni stanje v X zaklep n-teric.
- T2 uporablja indeks za branje dela R:
  - T2 dobi IS zaklep na R in s tem dobi S zaklep na n-tericah R.
- T3 prebere celoten R:
  - T3 dobi S na R.
  - ALI, T3 bi se lahko obnašala kot T2; uporaba lock escalation da bi se odločila katere n-terice.

Zaklepanje v B+ drevesih

- Kako lahko učinkovito zaklenemo določen list drevesa?
  - Mmg, ne zamenjuj tega z večnivojskim zaklepanjem!
- Ena rešitev: ignoriraj strukturo drevesa; zakleni strani med potovanjem po drevesu; uporabljaj 2FZ.
- Učinkovitost tega algoritma je slaba!
  - Koren drevesa in višji nivoji drevesa postanejo ozko grlo, ker se vsak dostop začne pri korenu.

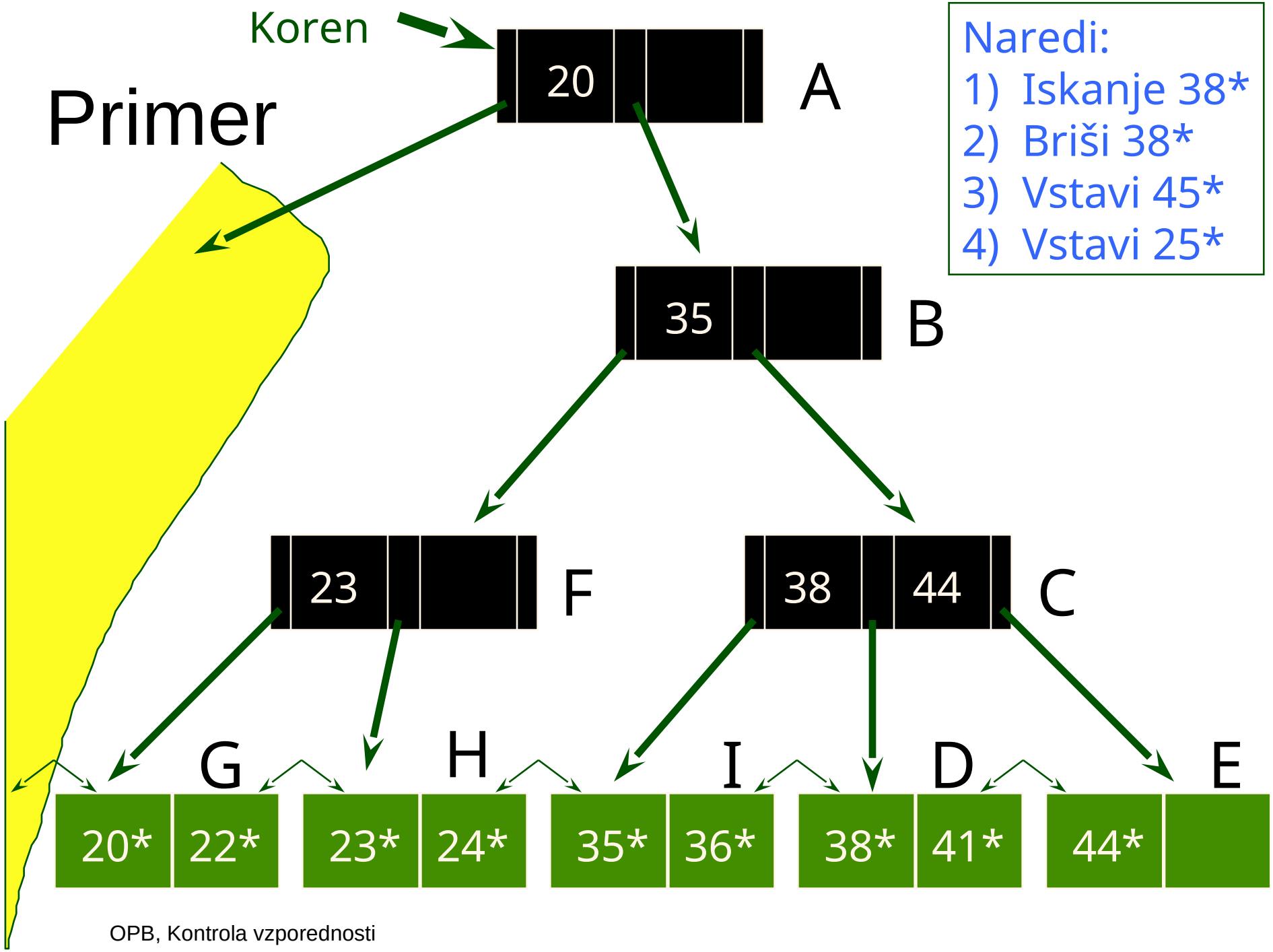
Dve koristni opazki

- Višji nivoji drevesa samo usmerjajo iskanje listov (strani).
- Pri vstavljanju morajo biti vsa vozlišča od korena do spremenjenega vozlišča zaklenjena na X način, samo v primeru, da lahko razcep prodre navzgor od spremenjenega lista. (Podobno velja tudi za brisanje.)
- Opazki lahko uporabimo pri zasnovi učinkovitih protokolov za zaklepanje, ki zagotavljajo zaporedno uredljivost *navkljub kršitvi 2FZ.*

Enostaven algoritem za zaklepanje dreves

- Iskanje: začni pri korenu in na poti do lista vedno zakleni otroka z S zaklepom in odkleni starša.
- Vstavi/Briši: Začni pri korenu in potuj navzdol tako, da pridobiš X zaklepe po potrebi. Ko je otrok zaklenjen preveri, če je varen:
  - Če je otrok varen sprosti vse zaklepe na starših.
- Varno vozlišče: Vozlišče, ki se spremeni spremembe ne bo širilo navzgor.
  - Vstavljanje: Vozlišče ni polno.
  - Brisanje: Vozlišče ni pol prazno.

Primer

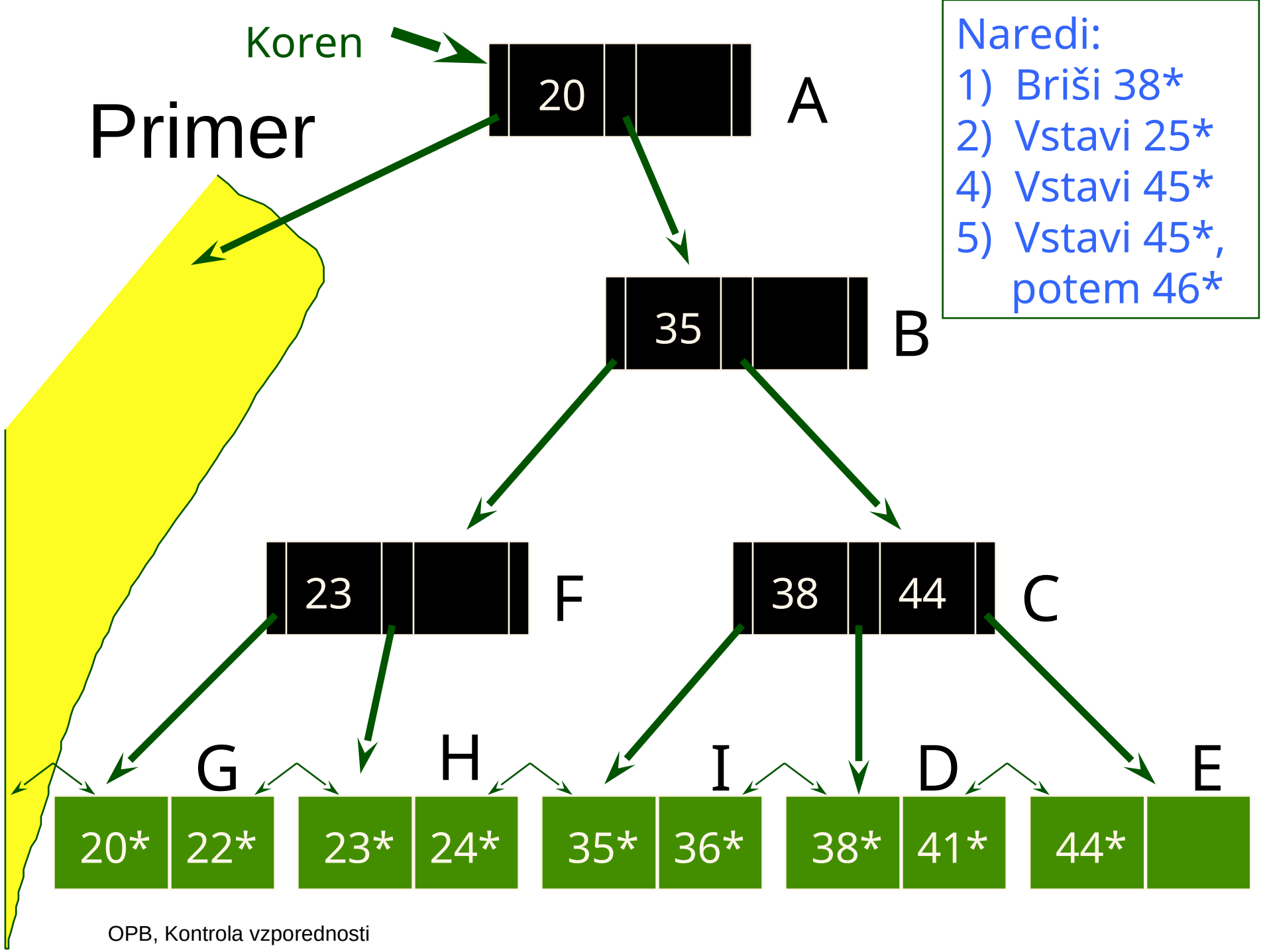


Naredi:
1) Iskanje 38*
2) Briši 38*
3) Vstavi 45*
4) Vstavi 25*

Boljši algoritem za zaklepanje drevesa

- Iskanje: Kot prej.
- Vstavljanje/Brisanje:
  - Postavi zaklepe kot v primeru iskanja, pride do lista in postavi X zaklep na listu.
  - Če list ni varen, sprosti vse zaklepe in ponovno poženi Xact s prejšnjim protokolom za Vstavljanje/Brisanje.
- Spekulira, da se bo spremenil samo list
  - Če ne so S zaklepi pri prvem prehodu nepotrebni.
  - V praksi dobimo boljši algoritem od prejšnjega.

Primer



Naredi:
1) Briši 38*
2) Vstavi 25*
4) Vstavi 45*
5) Vstavi 45*, potem 46*

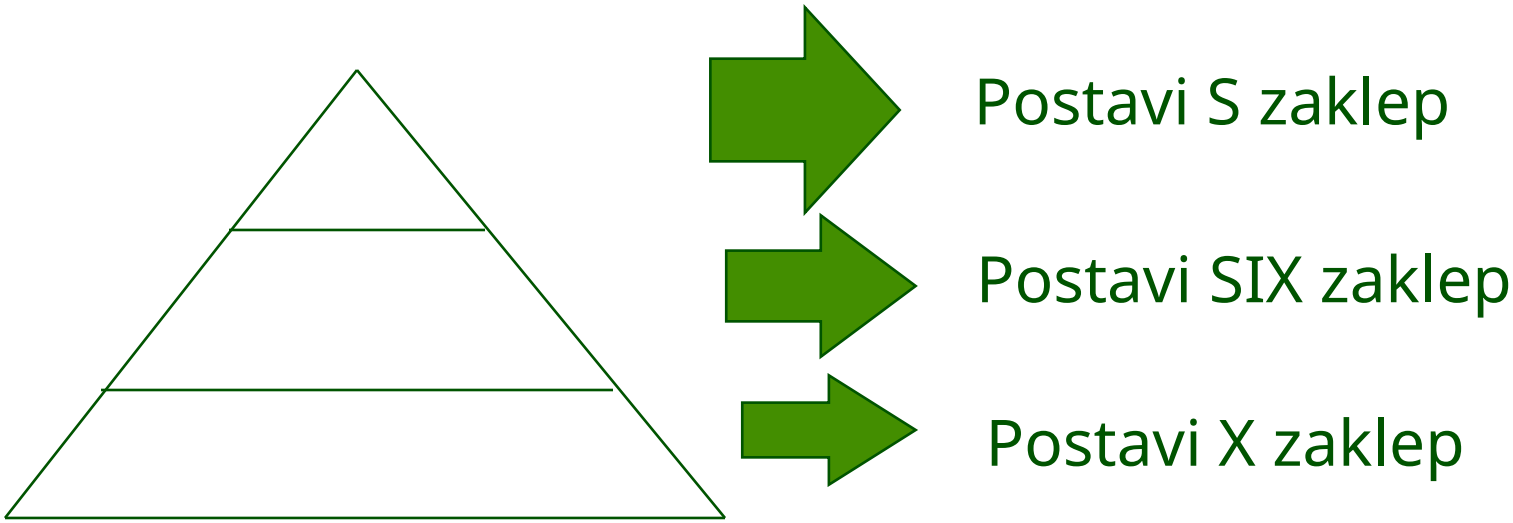
Še boljši algoritem

- Iskanje: Kot prej.
- Vstavljanje/Brisanje:
  - Uporabi orginalen protokol za vstavljanje/brisanje toda postavi IX zaklepe namesto X zaklepov.
  - Ko je list zaklenjen, prevedi vse IX zaklepe v X zaklepe od zgoraj-navzdol:
    - Začnemo v vozlišču najbljižjem korenu.
    - Protokol od-zgoraj-navzdol zmanjša možnosti za smrtni objem.

(Primerjaj uporabo IX zaklepov z uporabo v večnivojskem zaklepanju.)

Hibriden algoritem

- Verjetnost, da res potrebujemo zaklep tipa X se manjša s pomikanjem višje po drevesu.
- Hibriden pristop:

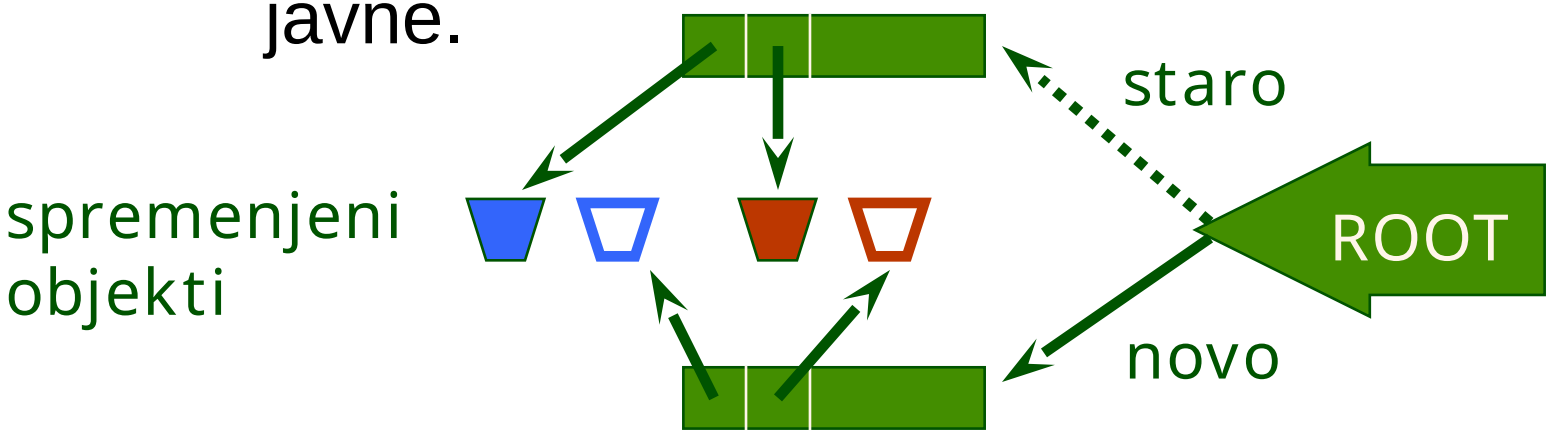


Optimistična kontrola vzporednosti (Kung-Robinson)

- Zaklepanje je konzervativni pristop, ki prepreči konflikte. Slabe lastnosti:
  - Dodatna koda za zaklepanje.
  - Preverjanje/reševanje smrtnega objema.
  - Zasičenost z zaklepi za vse pogosto uporabljene objekte.
- Če so konflikti redki lahko pridobimo na vzporednosti tako, da ne zaklepamo. Namesto tega preverimo konflikte pred potrjevanjem Xacts.
- Optimistična kontrola je hitrejša od zaklepanja, če jje malo konfliktov (slabša pri velikem številu konfliktov)

Kung-Robinson model

- Xact imajo tri faze:
  - READ: Xact prebere iz pod.baze vendar naredi spremembe na lokalni kopiji objektov.
  - VALIDATE: Preveri konflikte.
  - WRITE: Naredi lokalne kopije sprememb javne.

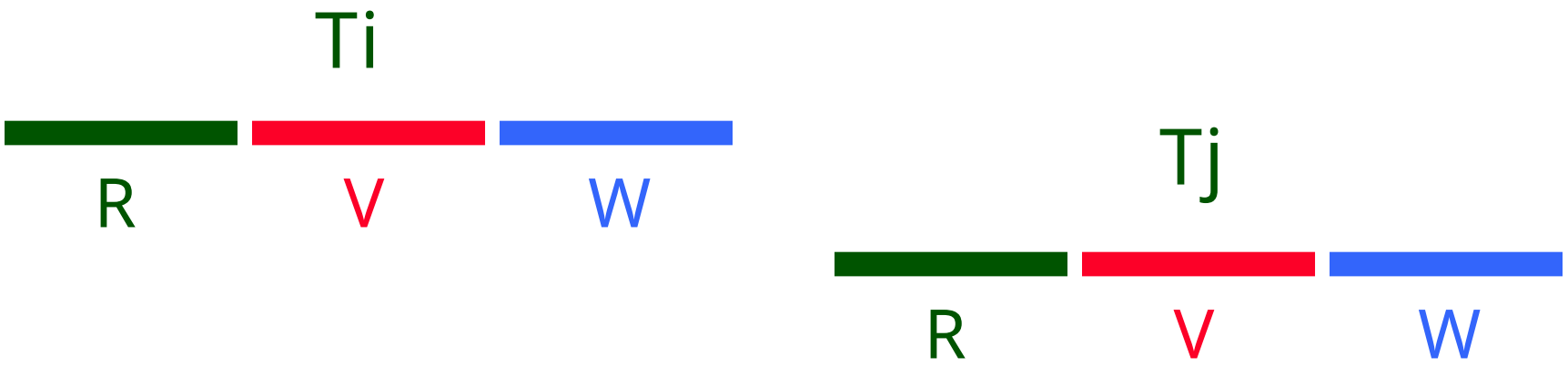


Validacija

- Preveri pogoje, ki so zadostni za to, da se ne pojavi konflikt.
- Vsaka transakcija dobi numerični id.
  - Uporabi časovni žig.
- Transakcijam se priredijo id-ji na koncu faze BERI, tik pred validacijo. (zakaj potem?)
- ReadSet(Ti): Množica objektov, ki jih Ti prebere.
- WriteSet(Ti): Množica objektov, ki jih Ti spremeni.

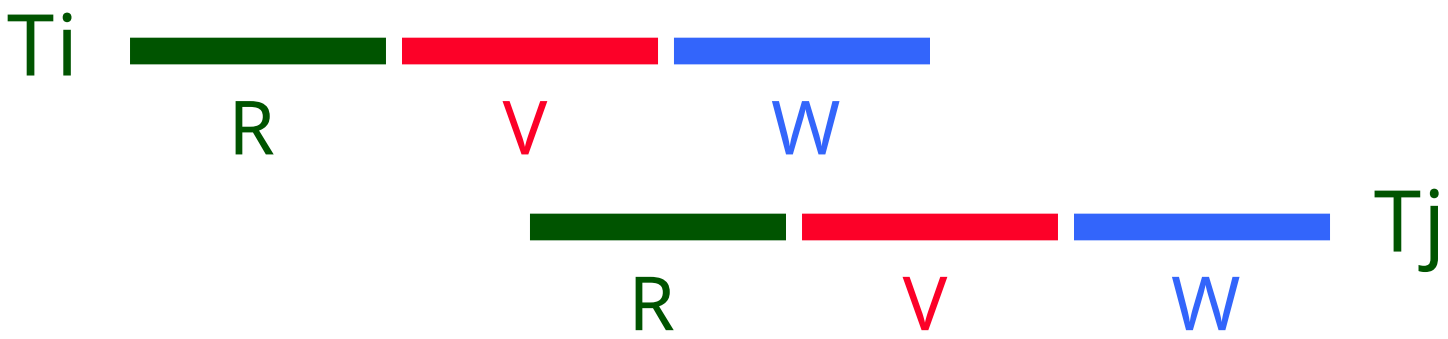
Test 1

- Za vse i in j tako da Ti < Tj, preveri če se Ti zaključi preden Tj začne.



Test 2

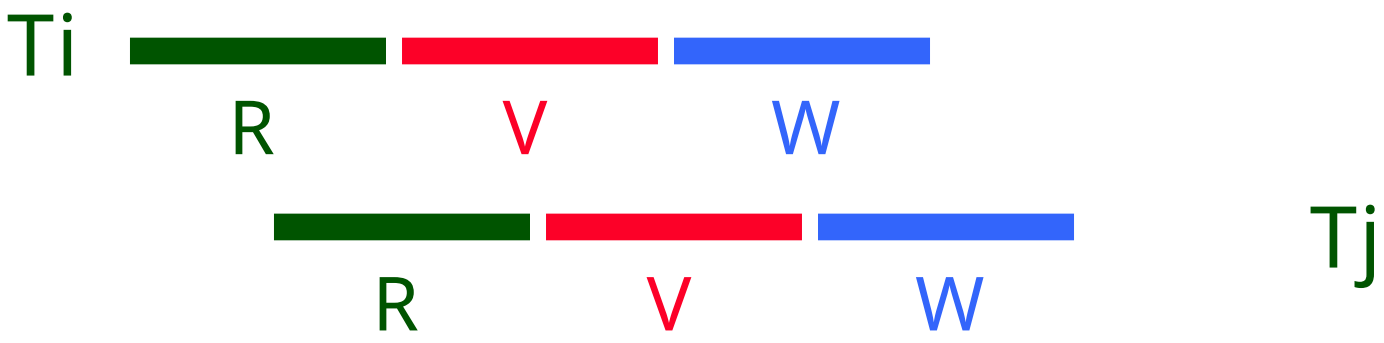
- Za vse i in j tako da Ti < Tj, preveri:
  - Ti zaključi preden Tj začne s pisalno fazo **+**
  - WriteSet(Ti) $\cap$ ReadSet(Tj) je prazna.



Ali Tj prebere “umazane” podatke?
Ali Ti prepiše zapise Tj?

Test 3

- Za vse i in j tako da Ti < Tj, preveri da:
  - Ti zaključi bralno fazo pred Tj **+**
  - WriteSet(Ti) $\cap$ ReadSet(Tj) je prazna **+**
  - WriteSet(Ti) $\cap$ WriteSet(Tj) je prazna.



Ali Tj prebere “umazane” podatke? Ali Ti prepiše zapise Tj?

Aplikacija Testov 1 & 2: Zaporedna validacija

- Validacija Xact T:

```
valid = true;
// S = set of Xacts that committed after Begin(T)
< foreach  Ts in S do {
   if ReadSet(T) does intersect WriteSet(Ts)
         then valid = false;
   }
   if valid then { install updates; // Write phase
                  Commit T } >
              else Restart T
```

end of critical section

Zaporedna validacija: komentarji

- Aplicira Test 2, kjer T igra vlogo Tj in vsaka Xact v Ts ustreza Ti.
- Prirejanje Xact id, validacija, in pisalna faza so znotraj kritične sekcije!
  - Nič drugega se ne izvaja vzporedno.
  - Če je pisalna faza dolga potem je to problem.
- Optimizacija (samo) za bralne Xacts:
  - Ne potrebujemo kritične sekcije (ker ni pisalne faze).

Zaporedna validacija

- Večfazna zaporedna validacija:
  - Validiraj po korakih. Vsak korak validiranja T se izvede nad podmnožico Xacts, ki so potrjene po začetku T.
  - Samo zadnja faza mora biti znotraj kritične sekcije.
- Stradanje:
  - Poženi stradajočo Xact v kritični sekciji (!!)
- Prostor za WriteSets: Za validacijo Tj, moramo imeti WriteSets za vse Ti za katere velja Ti < Tj in Ti je aktivna, ko Tj začne. Lahko je preveč takšnih Ti in zmanjka prostora.
  - Validacija Tj ne uspe, če manjka neka WriteSet.
  - Ni problema, če so žigi transakcije prirejeni na začetku bralne faze.

Problemi optimistične kontrole vzporednosti

- Mora zabeležiti vse aktivnosti branja/pisanja v ReadSet in WriteSet za posamezno Xact.
  - Mora kreirati in sprostiti množice po potrebi.
- Mora preveriti konflikte pred validacijo in mora izvajati validirana pisanja ``globalno''.
  - Kritična sekcija lahko zmanjša vzporednost!
  - Pisanje “globalno” lahko razdrobi skupke strani po tabelah in indeksih.
- Optimistična KV ponovno požene Xact, ki se ne validira.
  - Delo narejeno do takrat je izgubljeno; potrebuje še čiščenje.

``Optimističen'' 2FZ

- Lahko bi naredili tudi sledeče:
  - Postavi S zaklepe kot običajno.
  - Naredi spremembe na privatnih kopijah objektov.
  - Pridobi vse X zaklepe na koncu Xact, naredi pisanja globalna in potem sprosti zaklepe.
- Za razliko od (Optimistične KV) Kung-Robinson, so rezultati te sheme blokirane transakcije, ki čakajo na zaklep.
  - Ni validacije, ni ponovnega izvajanja transakcij (modulo smrtni objemi).

Kontrola vzporednosti s časovnimi žigi

- Ideja:
  - Priredi vsakemu objektu bralni časovni žig (RTS) in pisalni časovni žig (WTS),
  - Priredi vsaki Xact časovni žig (TS) ko se začne:
- Algoritem:
  - Če je akcija ai od Xact Ti v konfliktu z akcijo aj od Xact Tj, in TS(Ti) < TS(Tj), potem se mora ai zgoditi pred aj.
  - Sicer, ponovno poženi Xact.

Ko Xact T želi prebrati objekt O...

- Če TS(T) < WTS(O), potem je kršena urejenost časovnih žigov T nasproti pisalcu O.
  - Prekini T in jo ponovno zaženi z večjim TS. (sicer spet nebi uspela)
  - Primerjaj uporabo TS v 2PZ za izogibanje smrtnim objemom.
- Če TS(T) > WTS(O):
  - Dovoli T prebrati O.
  - Resetiraj RTS(O) na max(RTS(O), TS(T))
- Spremembe RTS(O) ob branju se morajo zapisati na disk!
  - To in ponovni zagoni so dodatno delo.

Ko Xact T želi zapisati objekt O...

- Če TS(T) < RTS(O), potem to krši urejenost časovnih žigov T nasproti pisalcu O; prekini in ponovno zaženi T.
- Če TS(T) < WTS(O), krši urejenost časovnih žigov T nasproti pisalcu O.
  - Thomas-ovo pisalno pravilo: Varno lahko ignoriramo prestara pisanja; ni potrebe po ponovnem zagonu T!
  - Pisanju T sledijo druga pisanja, brez vmesnih branj.
  - Dovoljuje zaporedne ureditve ne pa tudi ureditve po konfliktih.
- Sicer, dovoli T zapisati O.

T1	T2
R(A)	
	W(A)
	Commit
W(A)	
Commit	

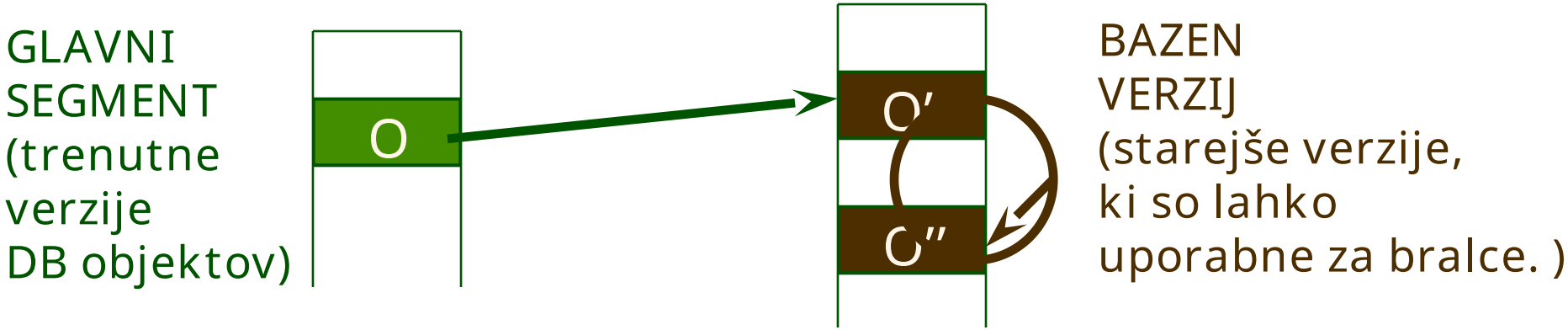
Kontrola vzporednosti in obnovljivost

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

- Na žalost so možne neobnovljive razporeditve:
- Kontrola vzporednosti s časovnimi žigi se da prilagoditi tako da dovoli samo obnovljive razporeditve:
  - Shrani vse zapise dokler pisalec ne potrdi (popravi tudi WTS(O) ko je pisanje dovoljeno.)
  - Blokiraj bralce T (kjer TS(T) > WTS(O)) dokler pisalec O ne potrdi.
- Podobno pisalcem, ki držijo X zaklep do potrditve, vendar še vedno ni zares 2FZ.

Multi-verzijska kontrola vzporednosti s časovnimi žigi

- **Ideja:** Naj pisalci naredijo novo kopijo medtem ko bralci berejo staro:

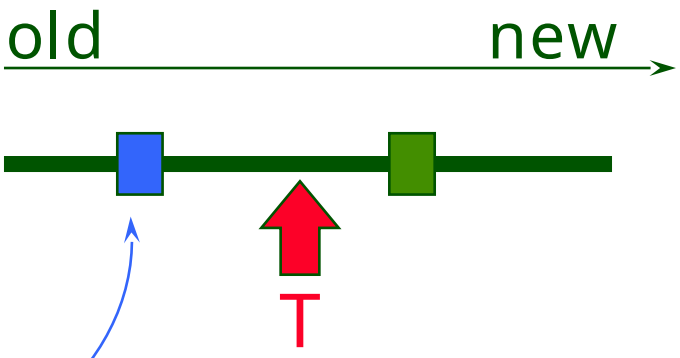


- Bralci lahko vedno delajo.
  - Lahko so blokirani dokler pisalci ne potrdijo pisanje.

Multi-verzijska kontrola vzporednosti (2)

- Vsaka verzija objekta ima TS pisalca WTS in TS Xact bralca, ki je zadnji prebral to verzijo kot RTS.
- Verzije so povezane nazaj; opustimo lahko verzije, ki so “prestare”.
- Vsaka Xact se klasificira kot Bralec ali Pisalec.
  - Pisalec lahko napiše kakšen objekt; Bralec ne bo nikoli.
  - Xact pove ali je Bralec, ko se začne izvajat.

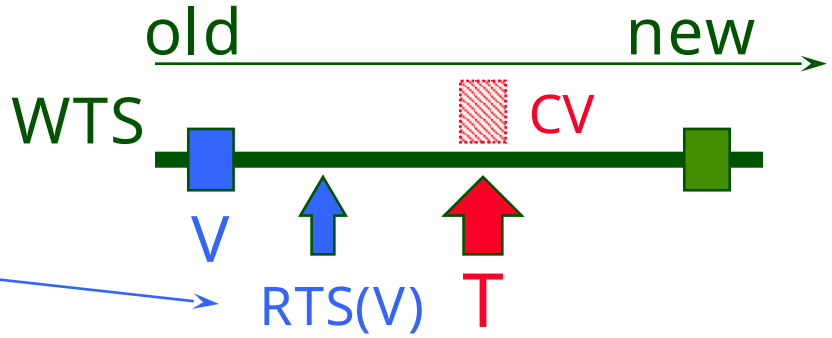
Bralec Xact



- Za vsak objekt, ki bo prebran:
  - Poišči **najnovejšo verzijo** z WTS < TS(T).
  - Začni s trenutno verzijo in se premikaj nazaj po seznamu.
- Če predpostavljamo, da najdemo neko verzijo vsakega objekta od začetka časa, potem bralčeve Xact ni potrebno nikoli ponovno zagnati.
  - Lahko blokira dokler pisalec ne potrdi prave verzije.

Pisalec Xact

- Za branje objekta uporabi protokol bralca.
- Za zapisovanje objekta:
  - Poišči najnovejšo verzijo V z WTS < TS(T).
  - Če RTS(V) < TS(T), potem T naredi kopijo CV verzije V, s kazalcem na V in WTS(CV) = TS(T), RTS(CV) = TS(T).
  - Vse zapisano je hranjeno dokler ni potrjeno; ostale Xacts lahko vidijo vrednosti TS vendar ne more prebrati verzije CV.
  - Sicer, zavrni pisanje.



Podpora transakcijam v SQL-92

- Vsaka transakcija ima način dostopa, predvideno velikost, in nivo izolacije.

Isolation Level	Dirty Read	Unrepeatable Read	Phantom Problem
Read Uncommitted	Maybe	Maybe	Maybe
Read Committed	No	Maybe	Maybe
Repeatable Reads	No	No	Maybe
Serializable	No	No	No