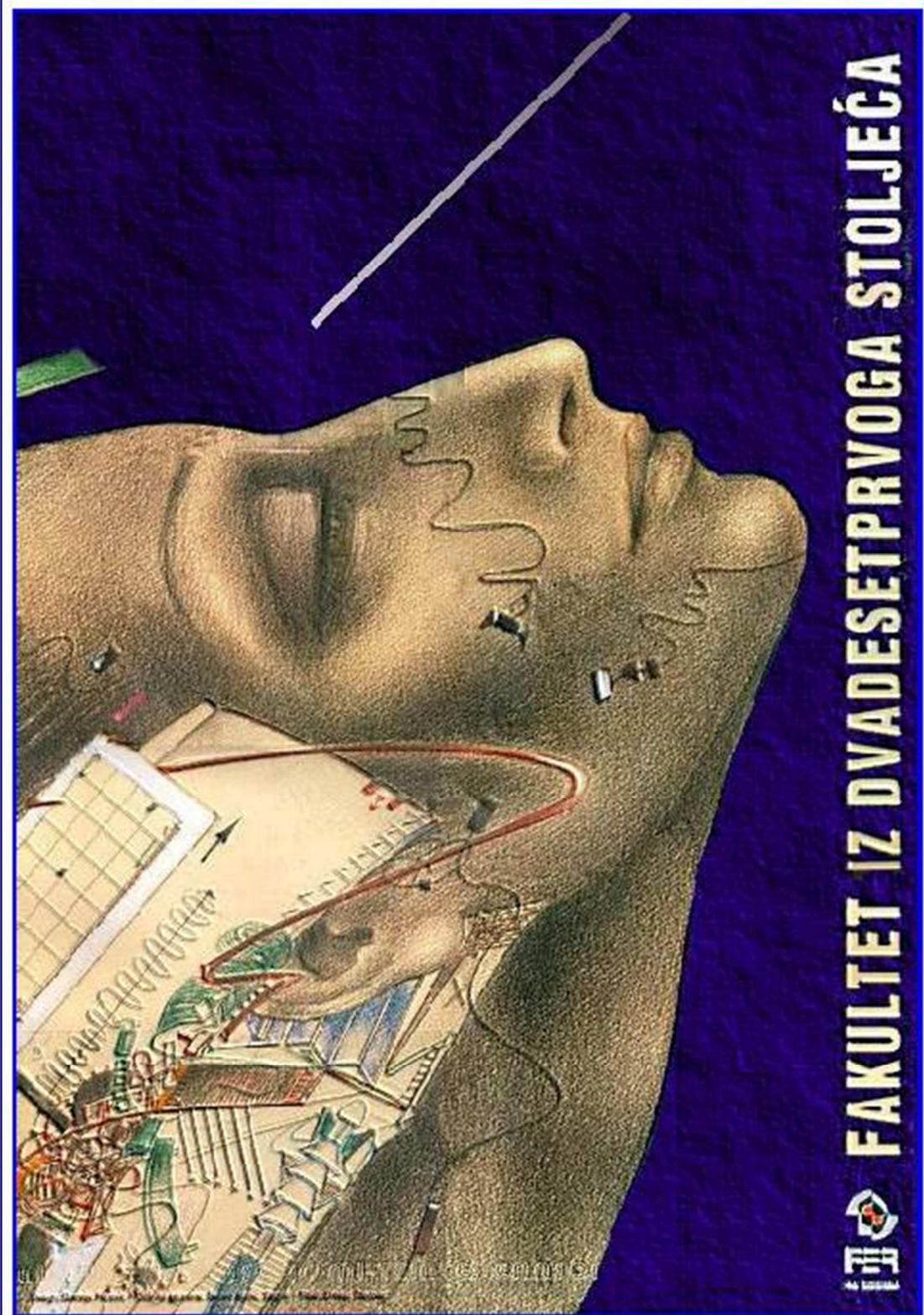


Sustavi baza podataka

Predavanja

Organizacija
predmeta

ožujak 2014.



FAKULTET IZ DVADESETPRVOGA STOLJEĆA

O predmetu

- teorijski predmet profila Programsko inženjerstvo i informacijski sustavi
 - predmet specijalizacije profila Telekomunikacije i informatika
 - **ECTS = 5.0** **opterećenje ≈ 150 sati**
- **Kratki opis**
 - Arhitektura sustava. Organizacija prostora za pohranu podataka. Fizička organizacija baza podataka. Principi optimizacije upita. Model transakcije. Obnova baze podataka. Teorija serijalizabilnosti. Postupci upravljanja istodobnim izvršavanjem transakcija. Distribuirane baze podataka.
 - **Kompetencije**
 - Produbljivanje znanja o različitim aspektima primjene i upravljanja sustavima za upravljanje bazama podataka. Studenti će naučiti najvažnije elemente administracije i podešavanja sustava za upravljanje bazama podataka. Praktičnim radom s nekim od sustava za upravljanje bazama podataka verificirati će usvojena znanja.

O predmetu

- **Što se na ovom predmetu neće izučavati?**
 - izgradnja sustava za upravljanje bazama podataka
 - izgradnja aplikacija za rad s bazama podataka
 - predmet nije tečaj za administratora baze podataka nekog konkretnog sustava za upravljanje bazama podataka

Očekivana predznanja

- Očekuje se poznavanje najvažnijih koncepata iz predmeta s preddiplomskog studija
 - Baze podataka
 - Algoritmi i strukture podataka
 - Operacijski sustavi

Literatura

- H. Garcia-Molina, J. D. Ullman, J. D. Widom: **Database Systems: The Complete Book**, Prentice Hall, 2002.
- A. Silberschatz, H.F. Korth, S. Sudarshan: **Database Systems Concepts**, 5th ed, McGraw-Hill, 2005.
- P. Bernstein, V. Hadzilacos, N. Goodman: **Concurrency control and recovery in database systems**, Addison-Wesley Publishing Company, 1987.
 - FREE download: <http://research.microsoft.com/en-us/people/philbe/ccontrol.aspx>
- M. Özsu, P. Valduriez: **Principles of distributed database systems**, 2nd ed, Prentice-Hall, 1999.
- T. M. Connolly, C. E. Begg: **Database Systems: A Practical Approach to Design, Implementation, and Management**, Addison Wesley, 2004.
- R. Elmasri, S. B. Navathe: **Fundamentals of database systems**, 3rd ed, Addison-Wesley, 2000.

Ostali nastavni materijali

- PDF folije s predavanja
- Službeni podsjetnici
- Domaće zadaće (praktične vježbe uz predavanja, zadaci za vježbu)
 - vježbe na vlastitom računalu - vježbe koje ilustriraju najvažnije koncepte
- Ispiti iz prethodnih akademskih godina
- **Prva domaća zadaća**
 - instalacija potrebne programske potpore
→ materijali → Domaće zadaće → 1. Instalacija programske potpore
- **Druga domaća zadaća**
 - pohranjene procedure (SPL), okidači
 - samostalno izučavanje odabralih poglavlja
→ materijali → Domaće zadaće → 2. Pohranjene procedure i okidači
- materijali i obavijesti na URL stranici predmeta
<http://www.fer.hr/predmet/sbp>

Izvedba

- **Nositelj**
 - Doc.dr.sc. Slaven Zakošek
- **Predavanja**
 - Doc.dr.sc. Slaven Zakošek
- **Vježbe**
 - Mr.sc. Jasenka Anzil
 - Doc.dr.sc. Ljiljana Brkić
- **Administracija**
 - gđa. Sonja Majstorović, tajnica Zavoda za primijenjeno računarstvo
- **Konzultacije**
 - na predavanjima ili prema dogovoru

Nastavni plan

1. ciklus (7 tjedana)	<ul style="list-style-type: none">▪ Arhitektura sustava▪ Fizička organizacija podataka▪ Optimizacija upita (2 predavanja)▪ Model transakcije▪ Sustav za obnovu (2 predavanja)	<ul style="list-style-type: none">▪ 1. Domaće zadaća: instalacija (odmah)▪ 2. Domaća zadaća: samostalno izučavanje SPL (<u>dovršiti prije 5. predavanja</u>)▪ Domaće zadaće (3–7): vježbe uz predavanja (prema obrađenim temama na predavanjima)
Međuispit		
2. ciklus (6 tjedana)	<ul style="list-style-type: none">▪ Upravljanje istodobnim pristupom (3 predavanja)▪ Distribuirane baze podataka (3 predavanja)	<ul style="list-style-type: none">▪ Domaće zadaće (8–9): vježbe uz predavanja
Završni ispit (pismeni i usmeni)		

Elementi ocjenjivanja

Za kontinuiranu provjeru znanja

- Domaće zadaće
 - ocjenjivanje: u 5. tjednu 2. ciklusa predavanja (2 - 7. lipnja)
- 10 bodova (prag 5)
- Međuispit
- 30 bodova (prag 9)
- Završni ispit - pismeni dio
- 40 bodova (prag 12)
- Završni ispit - usmeni dio
- 20 bodova
- Za prolaznu ocjenu potrebno je:
 - postići navedene pragove na svakoj od provjera znanja
 - ostvariti ukupno ≥ 50 bodova

Sadržaj pojedinih provjera znanja

Domaće zadaće

- Vježbe uz predavanja sa zadacima na koje treba pismeno odgovoriti
- Datoteku s rješenjima zadataka u propisanom obliku dostaviti na kraju semestra
 - predvidivi rok za predaju rješenja je 31. svibanj 2014.
- U 5. tjednu 2. ciklusa (2 - 7. lipnja) bodovanje uz individualno usmeno ispitivanje

Sadržaj pojedinih provjera znanja

Međuispit i završni ispit

- Svaka provjera znanja obuhvaća ukupno do tada obrađeno gradivo na predavanjima i domaćim zadaćama
 - s nešto većim naglaskom na gradivo iz pripadnog ciklusa
- Dvije grupe zadataka
 - približno 25% bodova na pismenoj provjeri znanja može se ostvariti na zadacima u kojima se bira jedan ili više odgovora između nekoliko ponuđenih
 - pogrešan odabir odgovora donosi negativne bodove!
 - približno 75% bodova može se ostvariti na zadacima bez ponuđenih odgovora
 - netočno rješenje ne donosi negativne bodove
- nije moguće ponavljanje međuispita niti završnog ispita

Usmeni ispit

- Provjera znanja i razumijevanja gradiva

Elementi ocjenjivanja

Za ispitne rokove

- Ispit - pismeni dio
 - **80 bodova (prag 40)**
- Ispit - usmeni dio
 - **20 bodova**
- Za prolaznu ocjenu potrebno je:
 - postići navedeni prag na pismenom dijelu ispita
 - ostvariti ukupno ≥ 50 bodova

Materijali na web stranici predmeta

Sustavi baza podataka

materijali

ljetni semestar 2010/2011 (9)

Predavanja (2)

- Uvod [157,3 KiB]
Objavljeno: Slaven Zakošek (Upravo sad!)
- 1. predavanje [553,4 KiB]
Objavljeno: Slaven Zakošek (Upravo sad!)

Domaće zadaće (5)

Instalacija programske potpore (1)

- Upute za instalaciju IDS sustava [860 KiB]
Objavljeno: Slaven Zakošek (prije 23 minute)

Pohranjene procedure i okidači (4)

- Pohranjene procedure i okidači u IDS-u [382,8 KiB]
Objavljeno: Slaven Zakošek (prije 22 minute)
- Pohranjene procedure i okidači - opis baze podataka [51,99 KiB]
Objavljeno: Slaven Zakošek (prije 20 minuta)
Uredio: Slaven Zakošek (prije 17 minuta)
- Pohranjene procedure i okidači - sql skripta za kreiranje baze podataka [825,2 KiB]
Objavljeno: Slaven Zakošek (prije 18 minuta)
Uredio: Slaven Zakošek (prije 16 minuta)
- Pohranjene procedure i okidači - zadaci za vježbu [98,33 KiB]
Objavljeno: Slaven Zakošek (prije 17 minuta)

Službeni podsjetnici (2)

- Službeni podsjetnik - SQL za predmet SBP - Ver. 3 [63,49 KiB]
Objavljeno: Slaven Zakošek (prije 4 minute)
Uredio: Slaven Zakošek (prije 2 minute)
- Službeni podsjetnik - Ver. 2 [21,14 KiB]
Objavljeno: Slaven Zakošek (prije 3 minute)

Ispiti (0)

ljetni semestar 2009/10 (35)

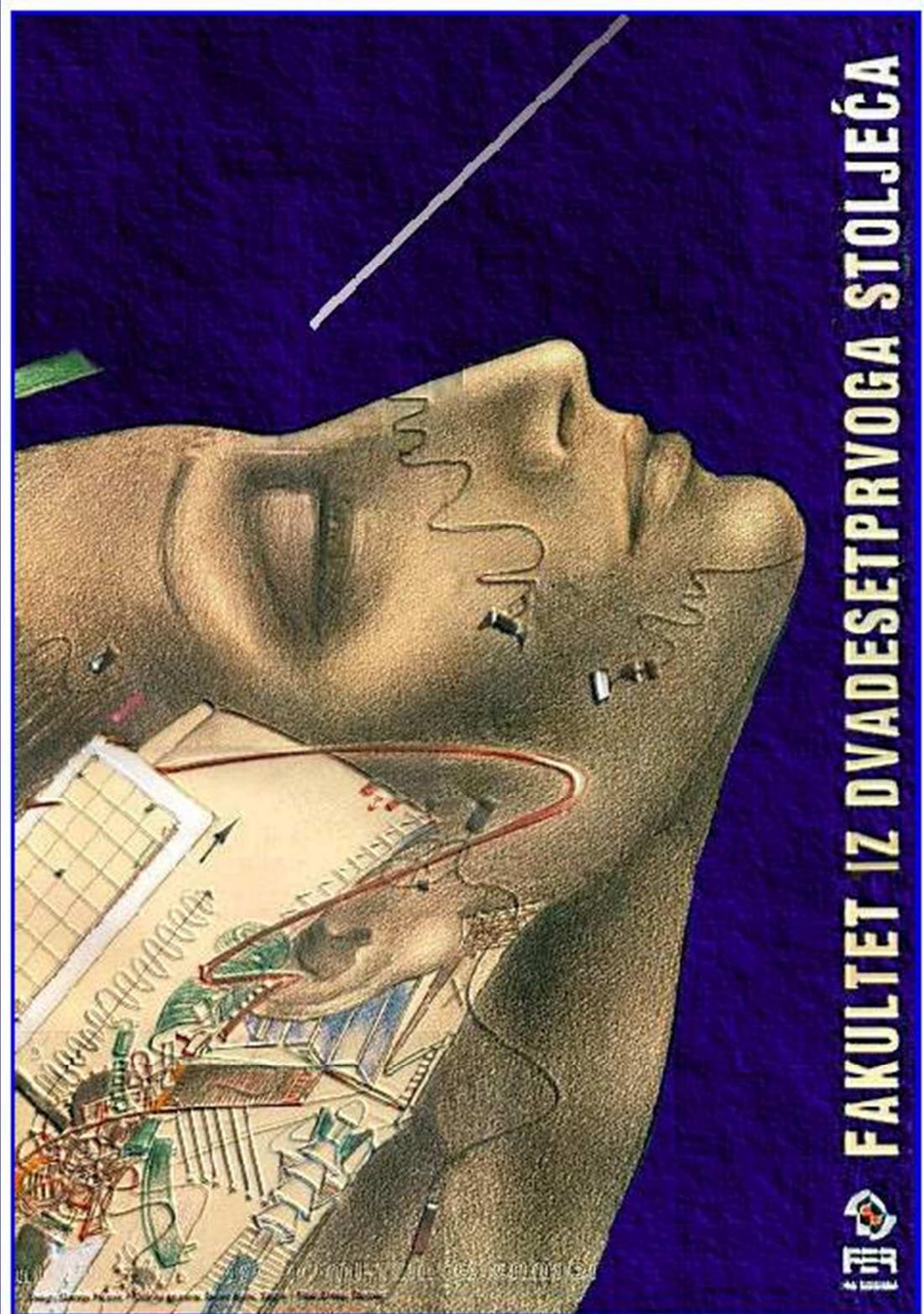
ljetni semestar 2008/09 (6)

Sustavi baza podataka

Predavanja

1. Arhitektura sustava

ožujak 2014.



FAKULTET IZ DVADESETPRVOGA STOLJEĆA

Sustavi baza podataka

Baza podataka

- BAZA PODATAKA je skup podataka koji su pohranjeni i organizirani tako da mogu zadovoljiti zahtjeve korisnika.

(*M. Vetter, 1981.*)

- BAZA PODATAKA je skup perzistentnih podataka kojeg koriste aplikacijski sustavi neke organizacije.

(*C. J. Date, 2000.*)

- BAZA PODATAKA je skup međusobno povezanih podataka, pohranjenih zajedno, uz isključenje bespotrebne zalihosti (redundancije), koji mogu zadovoljiti različite primjene. Podaci su pohranjeni na način neovisan o programima koji ih koriste. Prilikom dodavanja novih podataka, mijenjanja i pretraživanja postojećih podataka primjenjuje se zajednički i kontrolirani pristup. Podaci su strukturirani tako da služe kao osnova za razvoj budućih primjena.

(*J. Martin, 1979.*)

Sustav za upravljanje bazama podataka

- programski sustav koji upravlja istovremenim pristupom bazi podataka od strane više korisnika/aplikacija uz osiguravanje sigurnosti i integriteta baze podataka
 - složeni programski sustav - obuhvaća konglomerat znanja i desetljećima razvijanih tehnologija
 - jedan od najranije razvijenih višekorisničkih sustava
 - utjecaj na aplikacijsku programsку potporu, operacijske sustave, mrežne servise
- relacijski sustav za upravljanje bazama podataka

Zadaće sustava za upravljanje bazama podataka

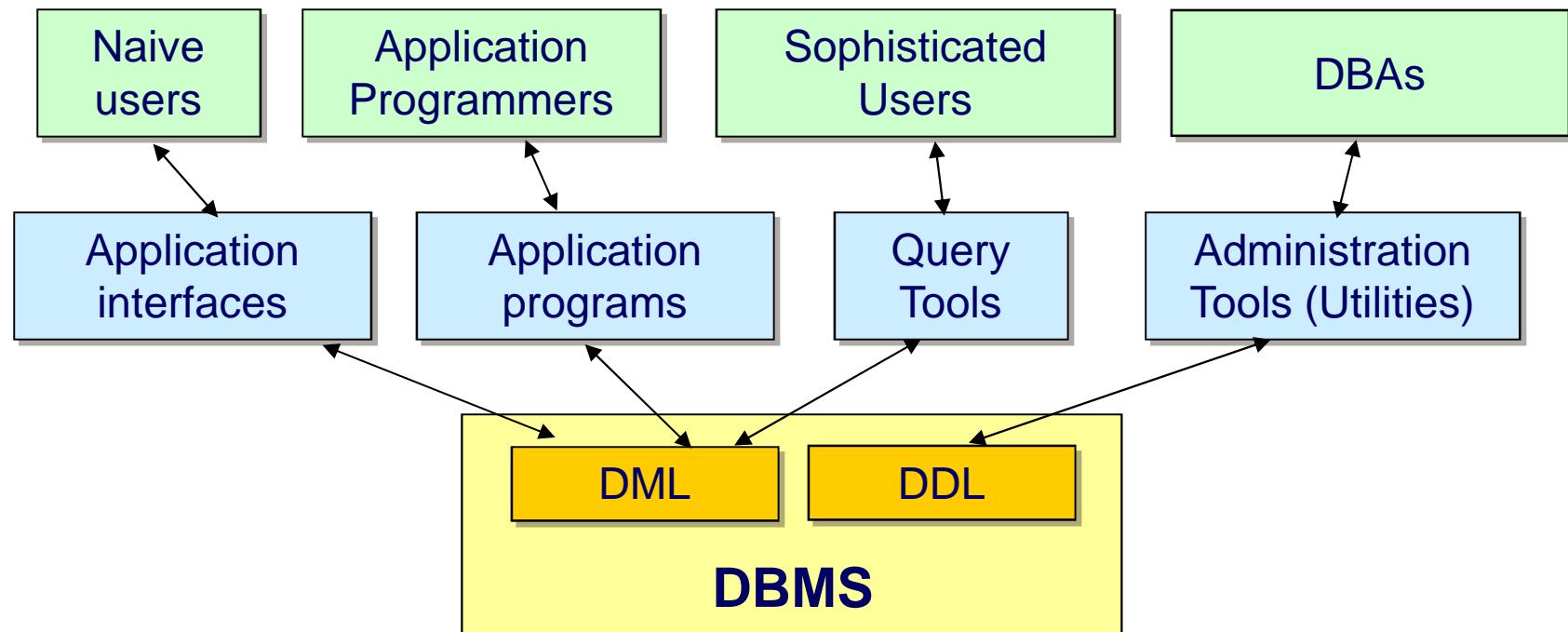
- trajna pohrana podataka (*persistent storage*)
- osiguravanje programskog sučelja (*programming interface*)
 - DDL (Data Definition Language), DML (Data Manipulation Language)
- zaštita podataka
 - integritet podataka (*integrity*)
 - pristup podacima - autorizacija, sigurnost (*security*)
 - potpora za upravljanje transakcijama
 - upravljanje istodobnim pristupom (*concurrency control*)
 - obnova u slučaju razrušenja (*recovery*)
- optimiranje metoda pristupa podacima (*query optimization*)

Sustav baza podataka

- Sustav za upravljanje bazama podataka zajedno s bazom podataka (i aplikacijama) čini sustav baze podataka
[Elmasri]
- Sustav baze podataka je kolokvijalni naziv za sustav za upravljanje bazama podataka
[Garcia-Molina]

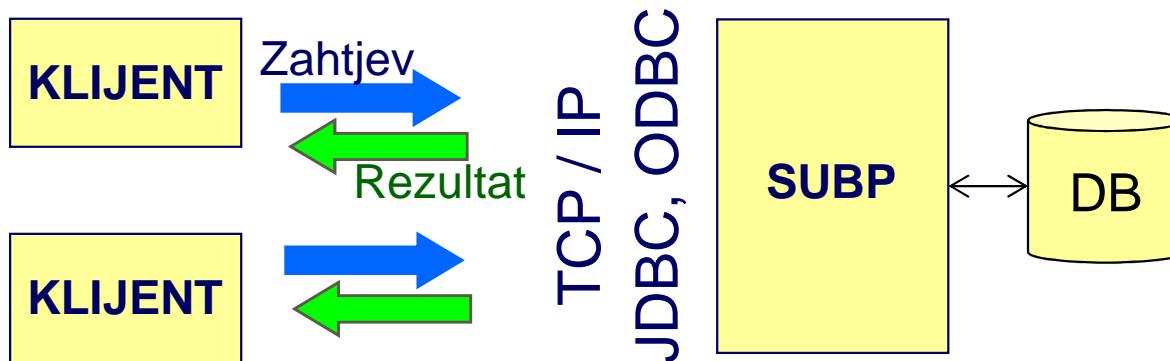
Arhitektura sustava za upravljanje bazama podataka

SUBP kao komponenta složenijih sustava



Klijent-poslužitelj arhitektura

- *Client-server (C/S)*



- sustav obuhvaća dvije komponente: klijent i poslužitelj
- koncept "zahtjev-odgovor" ("request-response"):
- klijent postavlja zahtjev - poslužitelj odgovara
- komunikacija se odvija preko dobro definiranih sučelja: npr. TCP/IP i standardna sučelja primjenskih programa (Application Program Interface - API): ODBC (Open Database Connectivity), JDBC ("Java Database Connectivity"), OLE/DB (Object Linking and Embedding/Database), itd.

Klijent-poslužitelj arhitektura i distribuirani sustavi

- postupak raslojavanja (distribucija) sustava - razdioba sustava na slojeve, od kojih svaki sadrži dio funkcionalnosti ukupnog sustava
- okruženje je tipično heterogeno
 - različito sklopovlje (*hardware*) i programska oprema, najčešće isporučena od strane različitih proizvođača
- slojevi imaju fundamentalno različite potrebe za računalnim resursima (brzina procesora, memorija, brzina i kapacitet diskova, U/I karakteristike)
 - npr. zahtjevi za sklopovljem i programskom potporom različiti su kod poslužitelja baze podataka i grafičke stanice
- C/S arhitektura se najčešće implementira korištenjem odvojenih računala, računala-klijenata i računala-poslužitelja. Međutim, oblik C/S arhitekture može se implementirati na jednom računalu. U tom slučaju klijent i poslužitelj procesi komuniciraju preko neimenovanih cjevovoda (*pipes*) ili dijeljene (*shared*) memorije.

Klijent-poslužitelj arhitektura

- tri sloja i njihovi podslojevi
 - različitim načinom grupiranja slojeva/podslojeva u komponente dobivaju se različiti tipovi arhitektura

Sloj korisničkog sučelja

Prezentacijski sloj

Sloj upravljanja dijalogom

Sloj obrade podataka

Sloj aplikacijske jezgre

Sloj upravljanja podacima

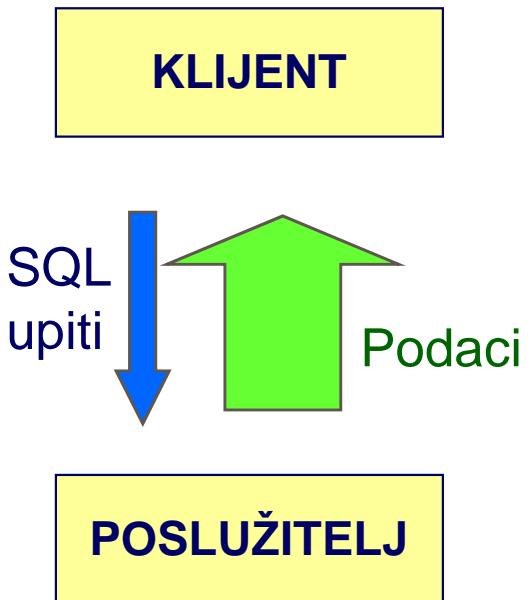
Sloj pristupa podacima

Sloj pohrane podataka

Dvoslojna (*two-tiered*) klijent-poslužitelj arhitektura oslonjena na klijenta (*client-centric*) - *fat client, thick client*

1. komponenta:

sloj korisničkog sučelja +
sloj obrade podataka



2. komponenta:

sloj upravljanja podacima

Client program:

```
...
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery(
    "SELECT * FROM racun");
int suma = 0;
while(rs.next()) {
    int brRac = rs.getInt(1);
    int iznos = rs.getInt(2);
    if (provjeraKontrolneZnamenke(brRac)) {
        suma = suma + iznos;
    }
}
...
ustmt = conn.prepareStatement(
    "INSERT INTO zbirno VALUES(?, ?)");
ustmt.setDate(1, "4.3.2004");
ustmt.setInt(2, suma);
ustmt.executeUpdate();
...
```

A large orange rectangular box contains the client code. To its right is a blue downward-pointing arrow and a green upward-pointing arrow, indicating the flow of data between the client and server.

Dvoslojna (*two-tiered*) klijent-poslužitelj arhitektura oslonjena na klijenta (*client-centric*) - *fat client, thick client*

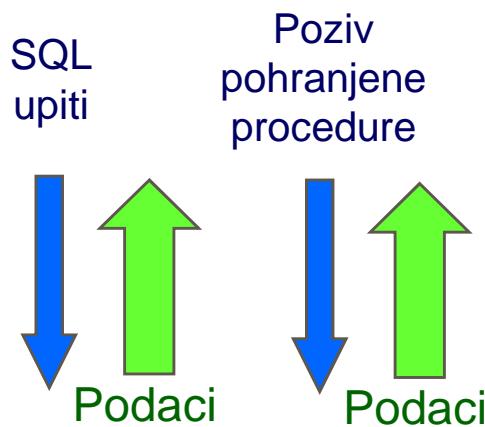
- korisničko sučelje (najčešće *GUI-graphical user interface*) usko je povezano s elementima za obradu podataka koji se u cijelosti implementiraju na klijentu
- Nedostaci:
 - *fat client sindrom* - budući da se obrada podataka obavlja na klijentu, klijent mora biti relativno snažno računalo
 - povećani troškovi opreme
 - degradacija performansi komunikacijske mreže, jer se podaci radi obrade moraju dobaviti na računalo-klijent
 - povećani troškovi održavanja programske potpore (promjene elemenata obrade podataka moraju se implementirati na svim računalima-klijentima)

Dvoslojna (*two-tiered*) klijent-poslužitelj arhitektura oslonjena na poslužitelja (*server-centric*) - *thin client*

1. komponenta:

sloj korisničkog sučelja

KLIJENT



2. komponenta:

sloj za obradu podataka +
sloj upravljanja podacima

Client program:

```
Statement stmt = conn.createStatement();
stmt.executeUpdate(
    "EXECUTE PROCEDURE proc1()");
```

Stored procedure:

```
PROCEDURE proc1 ()
DEFINE suma INTEGER; ...
LET suma = 0;
FOREACH
    SELECT brRac, iznos INTO
        pBrRac, plznos FROM racun
    IF (provjeraKontrolneZnamenke(pBrRac)) THEN
        LET suma = suma + iznos;
    END IF
END FOREACH
INSERT INTO zbirno VALUES(TODAY, suma);
END PROCEDURE
```

Dvoslojna klijent-poslužitelj arhitektura oslonjena na poslužitelja (server-centric)

- Nedostaci:
 - najčešće se koristi poseban jezik za komponentu aplikacije na strani klijenta, poseban za komponentu aplikacije na strani poslužitelja
 - dio elemenata obrade podataka se i dalje implementira na klijentu, jer jezici koji se koriste na poslužitelju nemaju sve mogućnosti koje imaju jezici koji se koriste u klijentskom sloju.
 - poslužitelj može postati usko grlo - *bottle-neck*

Troslojna (*three-tiered*) klijent-poslužitelj arhitektura

- 1. komponenta:
sloj korisničkog sučelja
- 2. komponenta:
sloj za obradu podataka + sloj pristupa podacima
- 3. komponenta:
sloj pohrane podataka



- primjer troslojne arhitekture:
 - klijent je zadužen za prezentaciju i upravljanje dijalogom: PC računalo s bogatim grafičkim korisničkim sučeljem (GUI) pisan u nekom modernom programskom jeziku (C, C#, C++, Java, VisualBasic...)
 - aplikacijski poslužitelj implementira sloj za obradu podataka i odgovoran je za upravljanje transakcijama, upravljanje opterećenjem sustava, sigurnost, itd. (BEA Tuxedo, IBM WebSphere Application server, Oracle Application Server, ...)
 - poslužitelj baze podataka je neki od sustava za upravljanje bazom podataka (Oracle, IBM DB2, IBM IDS, SQL Server...)

Višeslojna (*multi-tiered*) klijent-poslužitelj arhitektura

1. komponenta:
dio prezentacijskog
sloja



2. komponenta:
dio prezentacijskog
sloja + sloj upravljanja
dijalogom



3. komponenta:
sloj za obradu
podataka + sloj
pristupa podacima



4. komponenta:
sloj pohrane
podataka



- primjer višeslojne arhitekture:
 - klijent je internet preglednik (*browser*) koji prezentira HTML stranice (ili PDA, mobilni telefon, ...)
 - dio srednjeg sloja za upravljanje dijalogom je web poslužitelj podržan nekom od web tehnologija (JSP-Java ServerPages, ASP-Active Server Pages, ...)
 - dio srednjeg sloja s poslovним pravilima je aplikacijski poslužitelj koji podržava neku od tehnologija (EJB-Enterprise JavaBeans, CORBA-Common Object Request Broker Architecture, ...)
 - poslužitelj baze podataka je neki od sustava za upravljanje bazom podataka (Oracle, IBM DB2, ...)

Posljedice raslojavanja sustava

- pozitivne posljedice:
 - specijalizacija sklopolja i programske potpore
 - povećana skalabilnost
 - mogućnost upotrebe više aplikacijskih poslužitelja (*load balancing*)
 - smanjen broj potrebnih konekcija (otvorenih korisničkih sjednica) prema SUBP-u jer aplikacijski poslužitelj jednu konekciju može koristiti za servisiranje više korisnika
 - povećana sigurnost: sigurnosni elementi se mogu implementirati na više razina (umjesto npr. samo na razini baze podataka)
 - moguća bolja zaštita integriteta podataka: kontrola integritetskih ograničenja može se provoditi (i) u sloju aplikacijskog poslužitelja
 - nepažljiva implementacija može imati suprotni efekt. Ne preporuča se zanemarivanje kontrole integritetskih ograničenja u sloju upravljanja podacima
 - povećana modularnost: moguća zamjena elemenata u jednom sloju bez utjecaja na ostale slojeve (npr. zamjena SUBP-a)

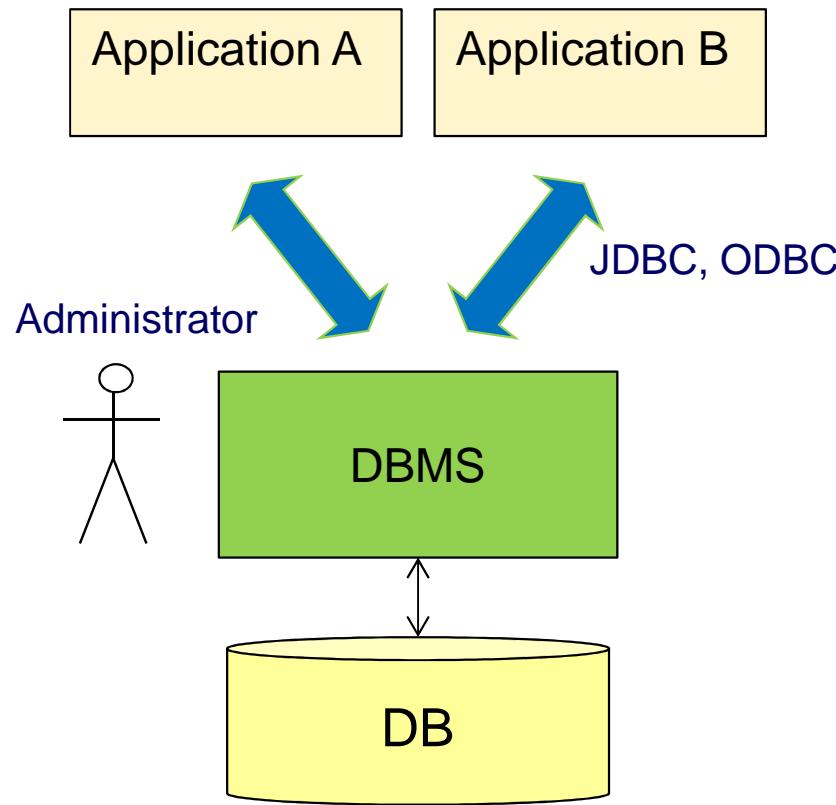
Posljedice raslojavanja sustava

- negativne posljedice:
 - povećanje konstrukcijske složenosti
 - u općem slučaju, teže je izgraditi višeslojni nego dvoslojni (ili centralizirani) sustav
 - složenija administracija sustava
 - u nekim slučajevima performansama može naštetiti fizička razdvojenost poslužitelja zaduženog za poslovnu logiku od poslužitelja zaduženog za podatke

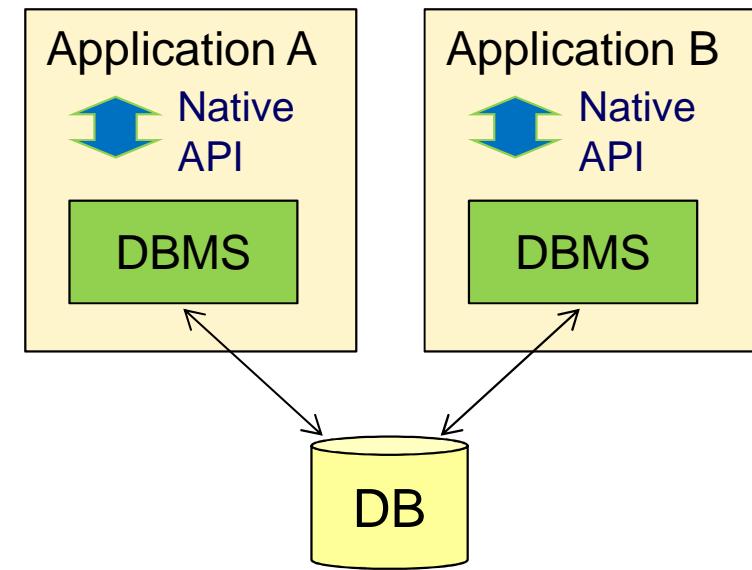
Ugrađeni sustavi za upravljanje bazama podataka

- *Embedded database management system*
 - Sustav za upravljanje bazama podataka integriran je u aplikaciju (*software product*)

"Enterprise DBMS"



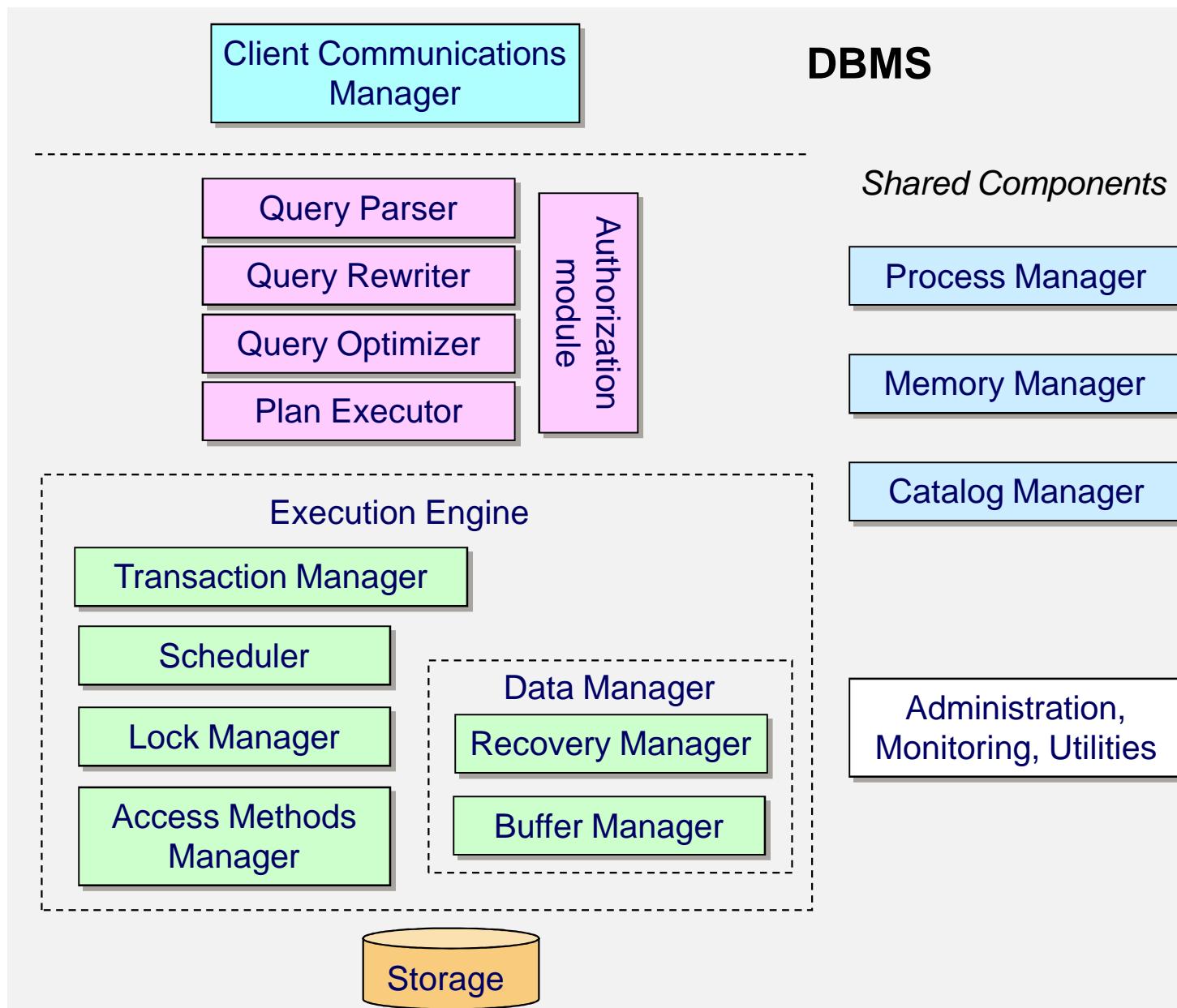
"Embedded DBMS"



- samoupravlјiv (*self-managed DBMS*)
- nezahtjevan prema resursima (*small footprint*)
- modularnost (*componentized DBMS*)

Komponente SUBP-a

Komponente SUBP-a



Komponente SUBP-a

- *"The Life of a Transaction"*

Uspostava komunikacije s korisnikom

- klijentska aplikacija (primjenski program) putem sučelja otvara korisničku sjednicu (*user session*) s menadžerom komunikacija (*Client Communications Manager*) čije su zadaće:
 - uspostaviti i održavati vezu (pamtiti stanje veze) prema klijentskoj aplikaciji
 - utvrditi sigurnosni status korisnika (npr. *login, password*)
 - prihvati zahtjeve, vraćati rezultate (podatke, kontrolne poruke)
- menadžer procesa (*Process Manager*) pokreće *server process* ili "dretvu izvršavanja" (*thread of execution, DBMS worker*) koja će biti zadužena za obrađivanje zahtjeva vezanih uz dotičnu korisničku sjednicu

Komponente SUBP-a

- "*The Life of a Transaction*" - nastavak

Prevođenje i optimizacija

- operacije transakcije (npr. SQL upiti) predaje se parseru (*Query Parser*) i modulu za pojednostavljivanje i normalizaciju upita (*Query Rewriter*)
- rezultat optimiranja upita u modulu za optimizaciju (*Query Optimizer*) je interni plan izvršavanja (*query plan*) kojim je određen "optimalni" redoslijed izvršavanja tipičnih operacija (spajanje, selekcija, projekcija, agregacija, sortiranje, itd.)
- interni plan izvršavanja predaje se izvršitelju plana (*Plan Executor*) koji implementira operacije spajanja, selekcije, itd. koristeći usluge modula na nižim razinama
- modul za autorizaciju (*Authorization Module*) može u svakoj od navedenih faza prevođenja i optimizacije biti konzultiran u vezi dozvola za izvršavanje pojedinih dijelova upita

Komponente SUBP-a

- "The Life of a Transaction" - nastavak

Izvršavanje

- modul za izvršavanje (*Execution Engine*) izvršava *transakcije*:
 - menadžer transakcija (*Transaction Manager*) objedinjuje pojedinačne zahtjeve (upite) u transakcije na temelju transakcijskih naredbi. Pojedinačne *database* operacije (*read, write*) i transakcijske (*transactional*) operacije (*start, abort, commit*) upućuje u menadžer redoslijeda izvršavanja
 - menadžer redoslijeda izvršavanja (*Scheduler*) uz pomoć menadžera zaključavanja (*Locking Manager*) određuje redoslijed izvršavanja operacija kojim se uz "*istodobno*" izvršavanje transakcija neće narušiti konzistentnost baze podataka
 - menadžer pristupa (*Access Manager*) koristi prikladne algoritme i strukture podataka za organiziranje i pristupanje podacima u sekundarnoj memoriji
 - menadžer međuspremnika (*Buffer Manager*) ima važnu ulogu osiguravanja efikasnog prijenosa podataka između primarne i sekundarne memorije
 - menadžer obnove (*Recovery Manager*) osigurava mogućnost obnove baze podataka u slučaju pogreške (kvara)

Komponente SUBP-a

- *"The Life of a Transaction"* - nastavak

Rezultati

- modul za izvršavanje dobivene rezultate smješta u međuspremnike menadžera komunikacija koji ih proslijedi klijentskoj aplikaciji
 - ako se radi o relativno velikim skupovima podataka klijentska aplikacija može dohvaćati dijelove rezultata, što će rezultirati iterativnim aktiviranjem metoda u menadžeru komunikacija i modulu za izvršavanje

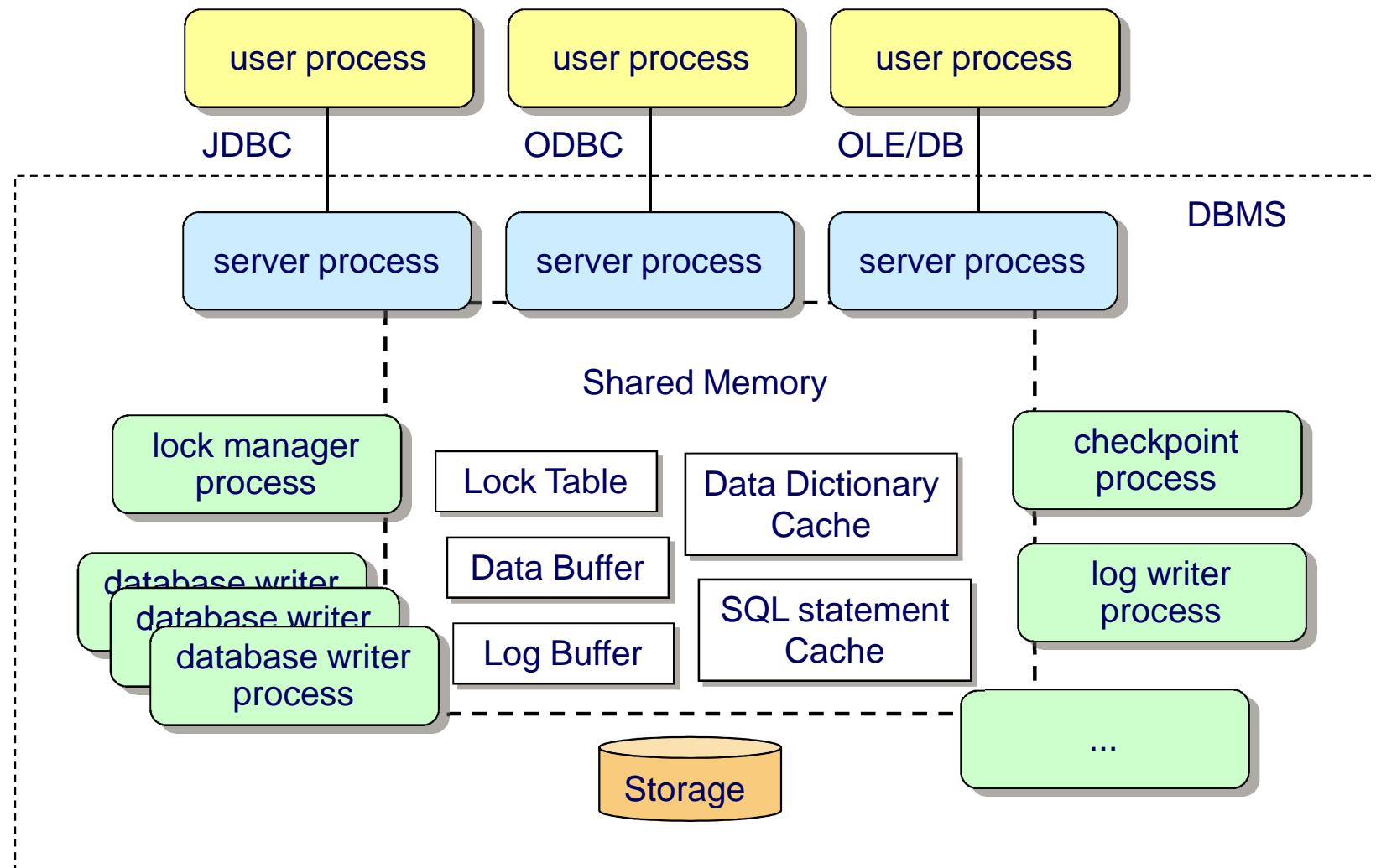
Komponente SUBP-a

- "*The Life of a Transaction*" - nastavak

Zajedničke komponente (**Shared Components**)

- zajedničke komponente koriste se kao servisi do sada opisanih modula
 - menadžer procesa (*Process Manager*) pokreće, zaustavlja i sinkronizira procese
 - menadžer memorije (*Memory Manager*) upravlja alokacijom i dealokacijom primarne memorije
 - menadžer kataloga (*Catalog Manager*) osigurava meta-podatke koji se koriste za autorizaciju, parsiranje i optimizaciju upita
- SUBP obuhvaća i niz pomoćnih (ali važnih) modula (*Administration, Monitoring, Utilities*) za administraciju i nadgledanje sustava, npr.
 - modul za upravljanje arhiviranjem i obnovom
 - modul za promjenu konfiguracijskih parametara
 - modul za nadzor i upravljanje prostorom u postojanoj memoriji
 - modul za nadzor i upravljanje procesima
 - itd.

Procesi u SUBP-u



Procesi u SUBP-u

- *User process*
 - proces klijentske aplikacije (nije SUBP proces)
- *Server process*
 - zaprima zahtjeve klijentskih aplikacija i vraća rezultate
 - upravlja korisničkom sjednicom (*user session*)
 - kontekst u kojem jedan korisnik obavlja niz SQL naredbi putem jedne veze (*SQL-Connection*) prema sustavu za upravljanje bazama podataka
- *Lock manager process*
 - implementira funkcionalnost menadžera zaključavanja
- *Database writer process*
 - jedan ili više procesa obavljaju prijenos podataka između primarne i sekundarne memorije
- *Log writer process*
 - upravlja zapisivanjem dnevnika u stabilnu memoriju
- *Checkpoint process*
 - periodički obavlja proces zapisivanja kontrolne točke

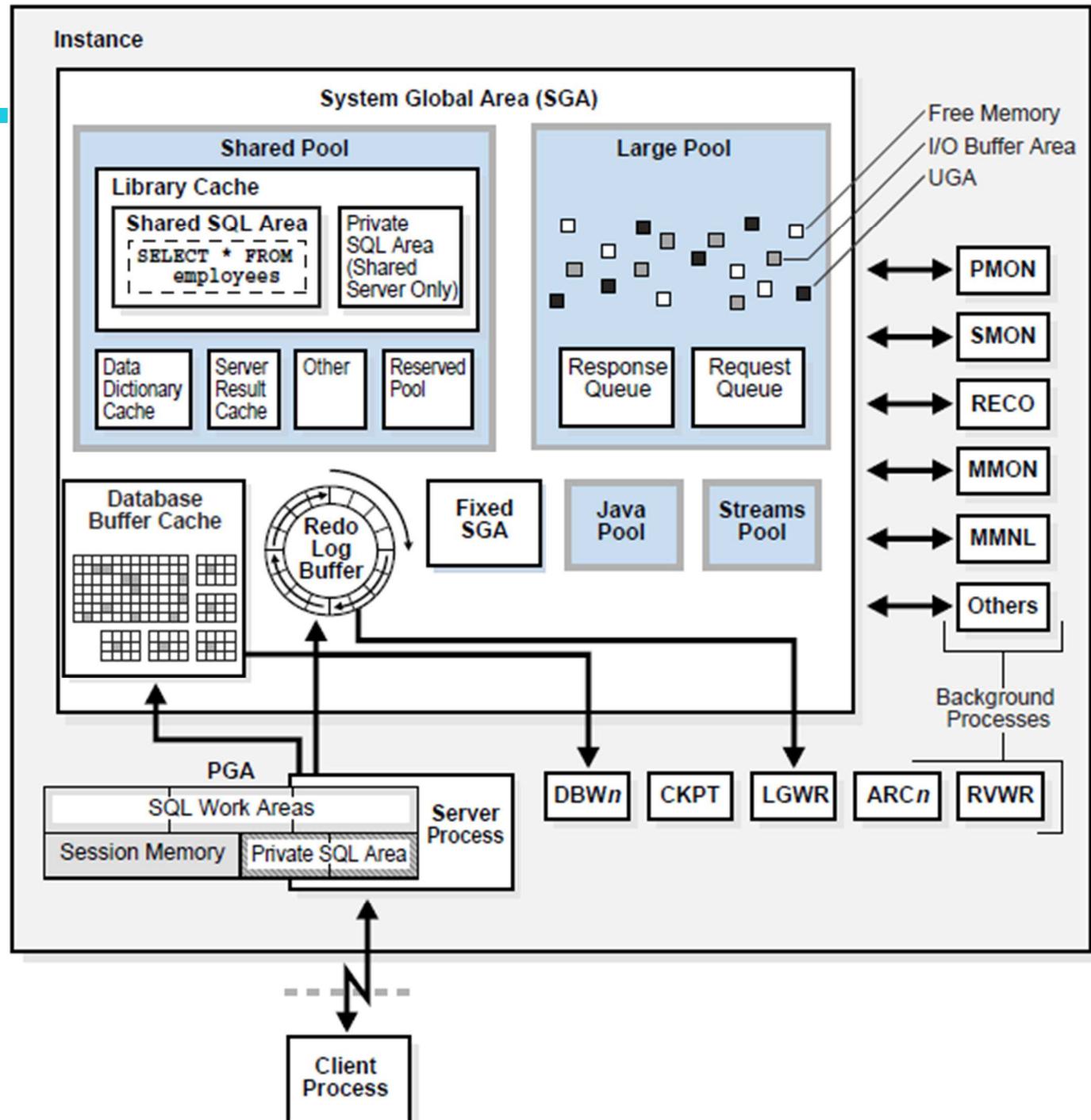
Procesi u SUBP-u

Procesi u Oracle 11g

Izvor:

Oracle® Database Concepts 11g
Release 2 (10.2)

PGA	Program Global Area
RECO	Recovery process
PMON	Process monitor
SMON	System monitor
CKPT	Checkpoint
ARCn	Archiver
DBWn	Database writer
LGWR	Log writer
D000	Dispatcher process



Procesi u SUBP-u

```
[root@localhost ~]# ps -aefd | grep oracle.*XE
oracle 3783 1 0 11:10 ? 00:00:00 xe_pmon_XE
oracle 3785 1 0 11:10 ? 00:00:00 xe_psp0_XE
oracle 3787 1 0 11:10 ? 00:00:00 xe_vktm_XE
oracle 3791 1 0 11:10 ? 00:00:00 xe_gen0_XE
oracle 3793 1 0 11:10 ? 00:00:00 xe_diag_XE
oracle 3795 1 0 11:10 ? 00:00:00 xe_dbrm_XE
oracle 3797 1 0 11:10 ? 00:00:00 xe_dia0_XE
oracle 3799 1 0 11:10 ? 00:00:00 xe_mman_XE
oracle 3801 1 0 11:10 ? 00:00:00 xe_dbw0_XE
oracle 3803 1 0 11:10 ? 00:00:00 xe_lgwr_XE
oracle 3805 1 0 11:10 ? 00:00:00 xe_ckpt_XE
oracle 3807 1 0 11:10 ? 00:00:00 xe_smon_XE
oracle 3809 1 0 11:10 ? 00:00:00 xe_reco_XE
oracle 3811 1 0 11:10 ? 00:00:01 xe_mmon_XE
oracle 3813 1 0 11:10 ? 00:00:00 xe_mmnl_XE
oracle 3815 1 0 11:10 ? 00:00:00 xe_d000_XE
oracle 3817 1 0 11:10 ? 00:00:00 xe_s000_XE
oracle 3819 1 0 11:10 ? 00:00:00 xe_s001_XE
oracle 3821 1 0 11:10 ? 00:00:00 xe_s002_XE
oracle 3823 1 0 11:10 ? 00:00:00 xe_s003_XE
oracle 3842 1 0 11:10 ? 00:00:00 xe_vkrm_XE
oracle 3844 1 0 11:10 ? 00:00:00 xe_qmnc_XE
oracle 4104 1 0 11:10 ? 00:00:00 xe_cjq0_XE
oracle 4141 1 0 11:10 ? 00:00:00 xe_q001_XE
oracle 4166 1 0 11:10 ? 00:00:00 xe_q002_XE
oracle 4306 1 0 11:15 ? 00:00:00 xe_smco_XE
oracle 15897 1 0 12:55 ? 00:00:00 xe_w000_XE
```

Organizacija procesa u SUBP-u

- organizacije procesa u kojoj se svakoj dretvi izvršavanja (*server process*) pridružuje jedan proces na razini operacijskog sustava (OS proces)
 - tip organizacije procesa korišten u ranijim implementacijama
- **OS process per DBMS thread of execution**
 - relativno jednostavna implementacija
 - koriste se mehanizmi operacijskog sustava
 - upravljanje procesima (*process switching*)
 - upravljanje dijeljenom memorijom, itd.
 - niska skalabilnost (1000 korisnika → 1000 procesa)
 - povećanjem broja korisničkih sjednica raste broj OS procesa
 - *process switching* postaje problem jer:
 - kontekst procesa obuhvaća sigurnosni kontekst, stanje memorijskog menadžera, opisnike datoteka, itd.
 - česta zamjena konteksta procesa degradira performanse

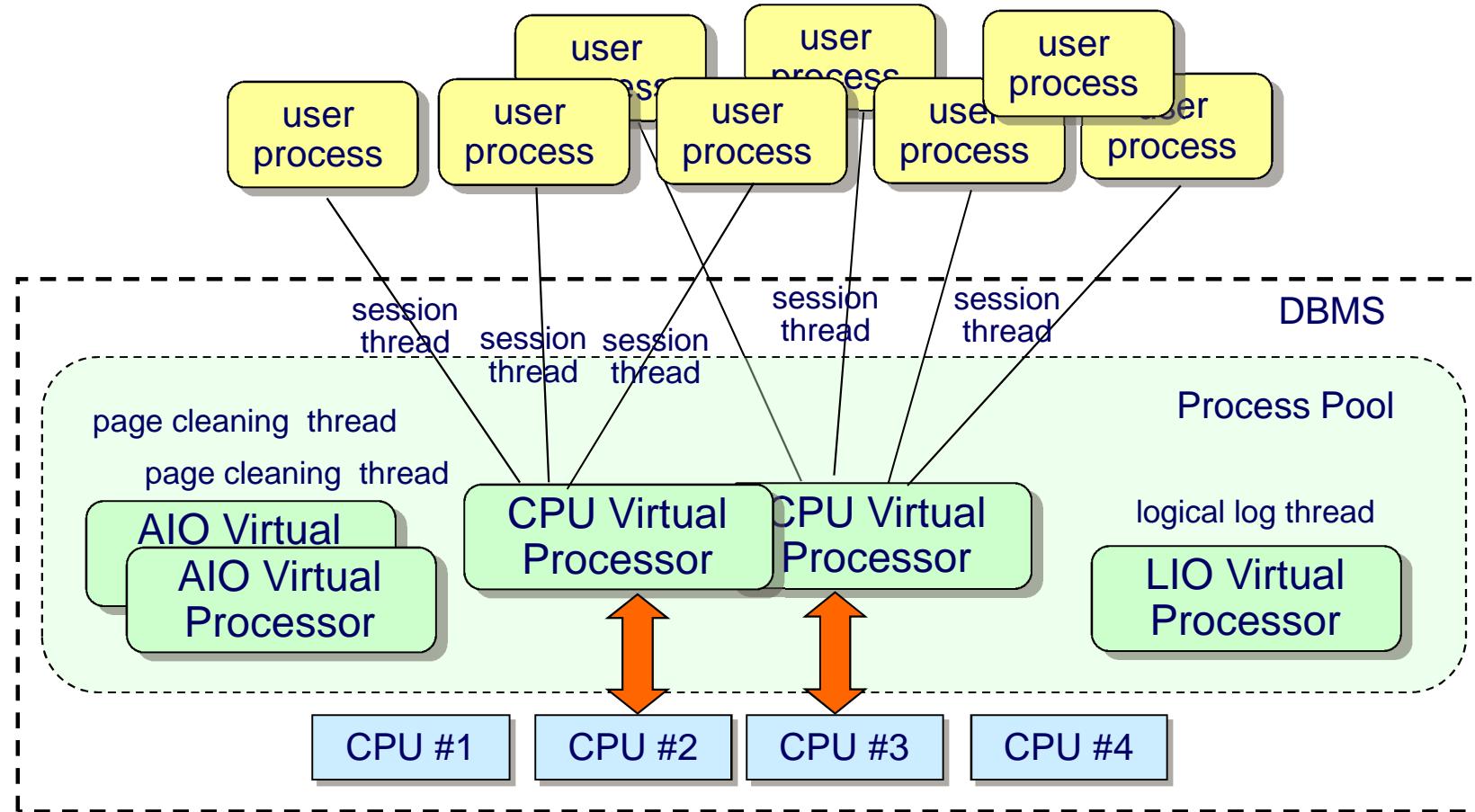
Organizacija procesa u SUBP-u

- moderni sustavi koriste organizaciju procesa u kojoj se dretva izvršavanja izvršava kao jedna od dretvi (*thread, lightweight process*) u okviru jednog OS procesa
- **{ OS thread | DBMS thread } per DBMS thread of execution**
 - jedan OS proces obavlja više dretvi izvršavanja
 - dretve izvršavanja dijele zajednički adresni prostor
 - trajanje zamjene konteksta dretve je znatno kraće od zamjene konteksta procesa
 - visoka skalabilnost
 - OS thread
 - koristi se postojeći (opći) OS API za upravljanje dretvama
 - otežana prenosivost SUBP-a na različite operacijske sustave
 - DBMS thread
 - SUBP koristi vlastiti (specijalizirani) API za upravljanje dretvama
 - vrlo složena implementacija SUBP-a

Organizacija procesa u SUBP-u

- moderni sustavi također često koriste modele organizacije procesa koji su spoj prethodno opisanih (OS proces izvršava više dretvi; postoji više OS procesa)
- npr. IBM IDS (Informix Dynamic Server) raspodjeljuje dretve korisničkih sjednica u *pool* OS procesa
- ***Process Pool (primjer za Informix IDS)***
 - OS procesi (vezani uz SUBP) se nazivaju virtualni procesori (VP)
 - virtualni procesori izvršavaju dretve izvršavanja i "interne dretve" (npr. dretve menadžera obnove, menadžera zaključavanja, itd.)
 - virtualni procesori svrstani su u različite razrede, pri čemu je VP svakog razreda specijaliziran za izvršavanje dretvi određenog tipa. Npr.
 - CPU VP: dretve korisničkih sjednica
 - AIO VP: dretve koje upravljaju U/I operacijama s podacima
 - LIO VP: dretve koje upravljaju U/I operacijama s logičkim dnevnicima
 - itd.
 - broj virtualnih procesora se može dinamički mijenjati
 - virtualni procesori se po potrebi mogu vezati (*bind*) uz fizički procesor

Virtualni procesori u sustavu IBM IDS



- u primjeru na slici su CPU virtualni procesori vezani uz fizičke procesore #2 i #3

Mediji za pohranu podataka

Hijerarhija medija za pohranu podataka

- Priručna memorija (*cache*)
 - Glavna memorija (*main memory*)
 - Sekundarna memorija
 - Tercijarna memorija
- }
- primarna
memorija
- RAM
- DRAM
- Flash-SSD, DRAM-SSD
- magnetski disk (HDD)
- optički medij (CD, DVD)
- magnetska traka
- Jukebox sustavi

Karakteristike medija za pohranu podataka

- Priručna memorija (*cache*)
 - memorija u (ili pri) središnjoj procesorskoj jedinici (CPU), vrlo velike brzine pristupa podacima (~10 ns), vrlo skupa, vrlo ograničenog kapaciteta (~ MB)
- Glavna memorija (*main memory*)
 - RAM, DRAM
 - velika brzina pristupa podacima (10-100 ns), relativno skupa, kapacitet (~ GB)
 - neprikladnost za (trajnu) pohranu baze podataka:
 - nepostojanost (gubitak napajanja, pogreška sustava)
 - cijena
 - kapacitet

Karakteristike medija za pohranu podataka

- Sekundarna memorija
 - postojana, sporija, većeg kapaciteta u odnosu na primarnu memoriju
 - Magnetski disk (HDD)
 - kapacitet: do 4TB (uz eksponencijalni rast tijekom vremena)
 - osjetljivost na mehaničke udare (~ 300g)
 - vrijeme pristupa (*seek time + rotational latency*): 5-15 ms (uz spor napredak tijekom vremena)
 - brzina prijenosa podataka: 50-300 MB/s
 - *disk-to-buffer, buffer-to-computer*
 - karakteristike medija za pohranu podataka (postojanost, kapacitet, brzina pristupa podacima, cijena) uvjetuju da se većina današnjih baza podataka pohranjuje na magnetskim diskovima

Karakteristike medija za pohranu podataka

- Sekundarna memorija (nastavak)
 - Flash-SSD
 - manji utrošak energije: ~ HDD/3
 - vrijeme pristupa podacima: ~ 0.1ms
 - brzina prijenosa podataka: ~ HDD
 - otpornost na vanjske utjecaje (-60/+90°C, otpornost na udarce ~ 1500g)
 - nedostaci: cijena, ograničeni broj ciklusa pisanja (3×10^5 - 10^6)

Karakteristike medija za pohranu podataka

- Tercijarna memorija
 - sporija, većeg kapaciteta
 - za izradu sigurnosnih kopija (*backup*) ili za vrlo velike baze podataka
 - Optički mediji (CD, DVD)
 - kapacitet do ~ 17 GB
 - vrijeme pristupa podacima ~100 ms
 - Magnetske trake
 - kapacitet 40 GB - 2 TB
 - *Jukebox* ili *tape silo* (robotizirani sustavi)
 - za pohranu vrlo velikih baza podataka (100 TB - 500 PB)
 - robotizirani sustav omogućava korištenje većeg broja jedinica medija (trake, CD, ...) u kombinaciji s manjim brojem uređaja za čitanje/pisanje
 - vrijeme pristupa: od nekoliko sekundi do nekoliko minuta

Karakteristike medija za pohranu podataka

- u kontekstu sustava za upravljanje bazama podataka treba razlikovati
 - **nepostojana (*volatile*) memorija**
 - sadržaj memorije se gubi u slučaju *pogreške sustava* (kvar napajanja, pogreška memorije, operacijskog sustava, ...)
 - RAM, DRAM
 - **postojana (*non-volatile*) memorija**
 - sadržaj memorije ostaje sačuvan u slučaju pogreške sustava
 - međutim, pogreške medija (npr. *disk crash*) će ipak izazvati gubitak sadržaja
 - HDD, Flash-SSD, DRAM-SSD, traka, DVD, ...
 - **stabilna (*stable*) memorija**
 - sadržaj memorija se *nikad** ne gubi

* garancija za "nikad" ne postoji, ali je moguće postići razumno razinu zaštite

Stabilna memorija

- implementacija: replikacijom podataka na medijima s postojanom memorijom
- najčešće: diskovi organizirani u RAID sustave - *on-line* stabilna memorija
 - *hot-pluggable, hot-spare*: eliminira se potreba za zaustavljanjem sustava u slučaju kvara medija
 - RAID ne garantira "stabilnost" memorije, npr. u slučaju požara
- korištenje trake, CD, DVD (napraviti kopiju i pohraniti medij u drugom gradu)
 - pogodno za arhivske kopije (*backup*), ali ne omogućava kontinuiranu zaštitu sadržaja: *off-line* stabilna memorija
- korištenje SAN (*Storage Area Network*) arhitekture
 - polje RAID organiziranih diskova fizički odvojenih od poslužitelja
 - povezivanje SCSI ili Fibre Channel sučeljem

Organizacija podataka u sekundarnoj memoriji

- logički zapisi (n -torke) pohranjuju se u blokove (fizičke zapise, stranice) sekundarne memorije
 - block, physical record, page*
 - veličina bloka ovisi o sklopoljtu i operacijskom sustavu (tipično: 512 B, 1 kB, 2 kB, 4 kB)
 - ovisno o veličini logičkih zapisa i blokova
 - u jedan blok može biti smješteno više logičkih zapisa (npr. n -torki relacije)
 - moguće je da će se za pohranu jednog logičkog zapisa koristiti više blokova

Primjer:

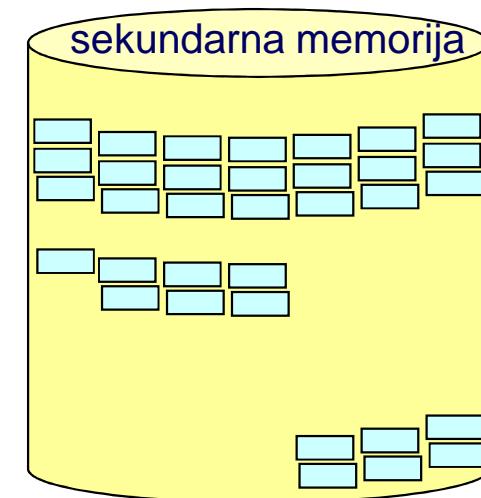
mbr	ime	prez
1999999	Ivan	Novak
1000315	Ana	Horvat
...		
1999998	Jura	Kolar
1000001	Tea	Ban

} 56 B

10^6 n-torki

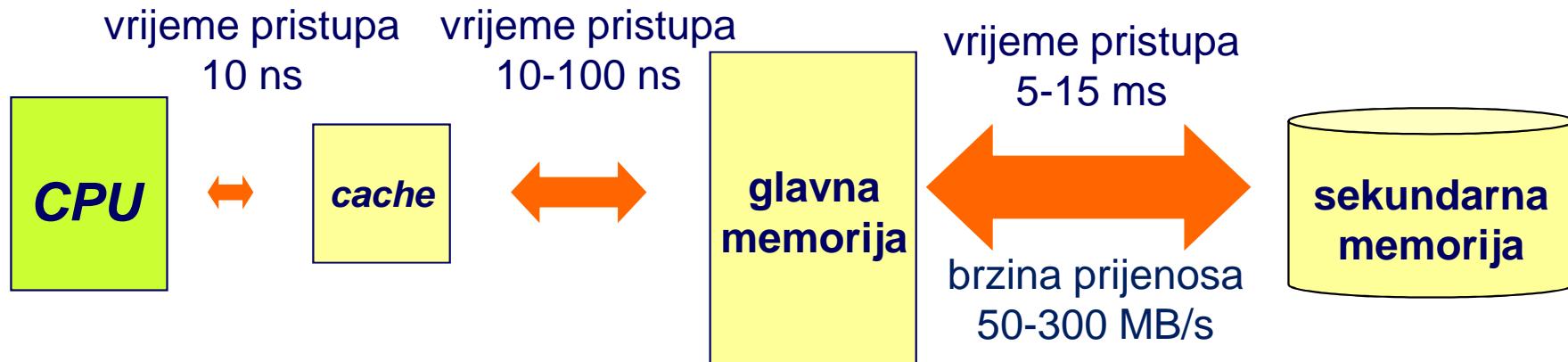
⇒ uz pretpostavku
veličine bloka 1kB

⇒ broj potrebnih
blokova u
sekundarnoj
memoriji: najmanje
54688



Pristup podacima u sekundarnoj memoriji

- koliko je podatak "daleko od procesora"



- operacije čitanje/pisanje
 - read, write*
- U/I - ulazno/izlazne operacije
 - obavljaju se u jedinicama blokova
 - input, output (I/O)*
 - disk-read, disk-write*
- zbog velike razlike u brzini (~ ns : ~ ms)
 - ⇒ **dominiraju troškovi U/I operacija**

Utjecaj broja U/I operacija na performanse sustava

- ukupno vrijeme potrebno za prijenos blokova iz sekundarne u primarnu memoriju najviše ovisi o sljedećim karakteristikama medija
 - *seek time*
 - *rotational delay (latency)*
 - *transfer time*
- znatan utjecaj ima stupanj fragmentacije (koliko su blokovi koji se prenose raspršeni po cilindrima i sektorima na disku)

Primjer:

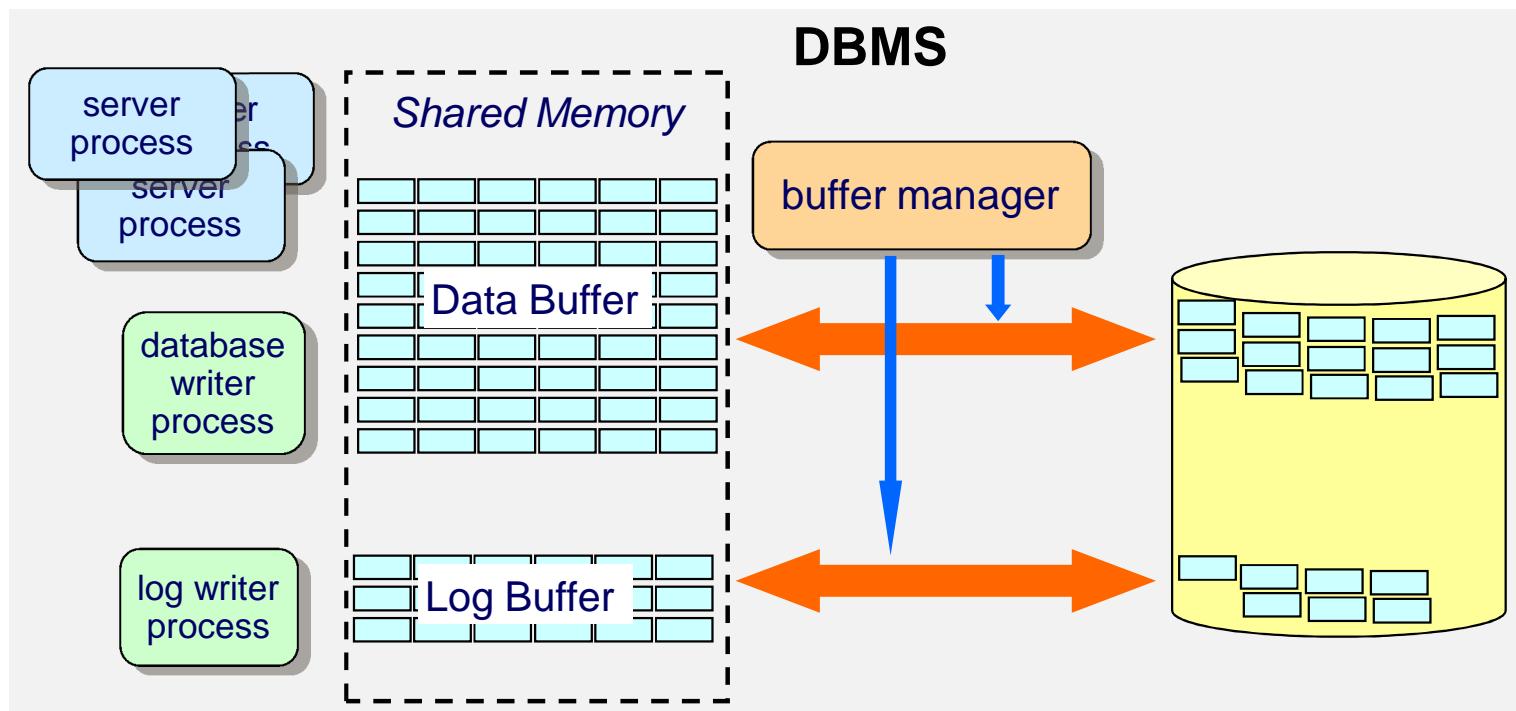
- neka se nad podacima iz prethodnog primjera obavlja upit

```
SELECT * FROM osoba  
WHERE mbr BETWEEN 1222000 AND 1222300;
```

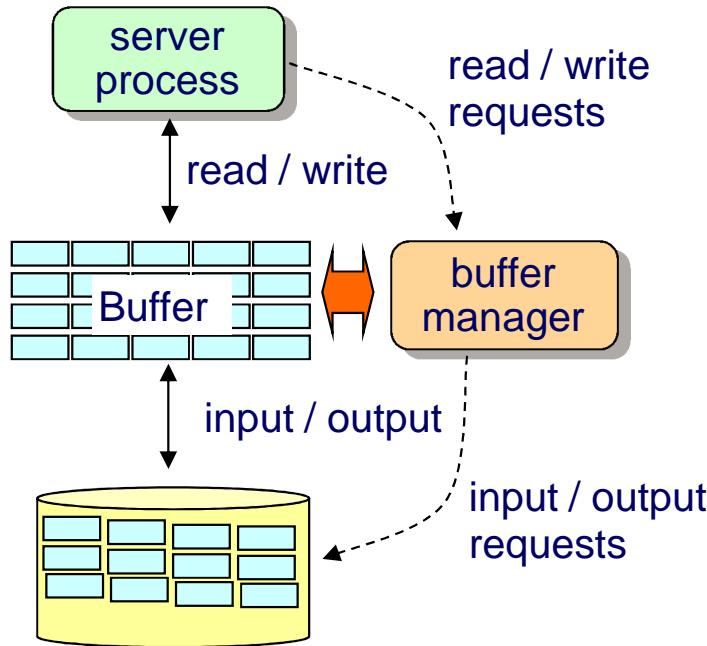
- uz pretpostavku korištenja slijednog pretraživanja, svaki od 54688 blokova mora se prenijeti u glavnu memoriju i pretražiti
 - za prijenos podataka očekuje se utrošak vremena mјeren sekundama
 - za pretraživanje svih blokova u glavnoj memoriji očekuje se utrošak vremena mјeren milisekundama

Organizacija podataka u glavnoj memoriji

- raspoloživa primarna memorija partitionirana je u blokove koji svojom veličinom u pravilu odgovaraju blokovima u sekundarnoj memoriji
 - međuspremnik (*buffer* ili *buffer pool*)
- komponente SUBP-a koriste sekundarnu memoriju posredstvom menadžera međuspremnika



Uloga menadžera međuspremnika



- proces koji treba obaviti operaciju čitanja postavlja *read* zahtjev menadžeru međuspremnika
- ako se blok koji sadrži traženi podatak ne nalazi u međuspremniku, menadžer obavlja operaciju *input* kojom se potreban blok čita iz sekundarne memorije
- proces koji treba obaviti operaciju pisanja postavlja *write* zahtjev menadžeru međuspremnika
- ako se blok s podatkom nad kojim treba obaviti operaciju pisanja ne nalazi u međuspremniku, menadžer obavlja operaciju *input* kojom se potreban blok čita iz sekundarne memorije. Operacija *output* se ne mora (uvijek) obaviti odmah nakon *write*
- poželjno bi bilo blokove čim dulje čuvati u međuspremniku (zašto?), međutim blokovi u međuspremniku ne mogu ostati zauvijek
 - izmijenjene blokove (*dirty pages*) treba zapisati u sekundarnu memoriju jer promjenu obavljenu nad podacima treba trajno zabilježiti (obaviti operaciju *output*)
 - međuspremnik ima ograničeni kapacitet te će se i blokovi koji nisu promijenjeni morati izbaciti iz međuspremnika (u tom slučaju operacija *output* nije potrebna)

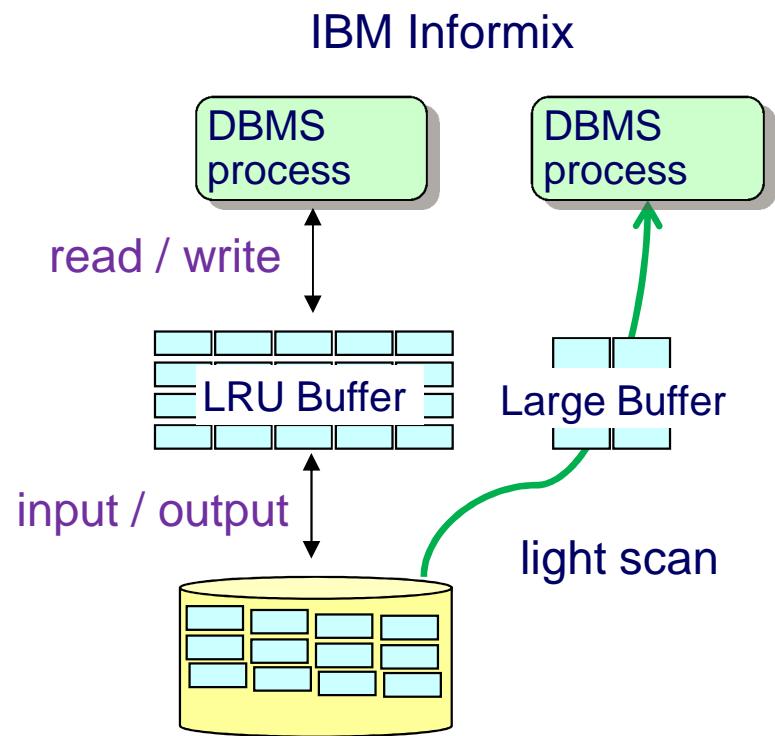
Strategije zamjene stranica

- algoritam koji određuje koje blokove (i kada) izbaciti iz međuspremnika (*buffer-replacement policy*) ima veliki utjecaj na efikasnost sustava
 - Least-Recently Used (LRU) - nisu korišteni tijekom najduljeg vremena
 - Most-Recently Used (MRU) - proteklo najmanje vremena od zadnjeg korištenja
 - First-In-First-Out (FIFO)
 - Clock Algorithm
- u današnjim sustavima najčešće se koristi LRU algoritam*
 - prepostavka: blok kojeg je neki proces koristio nedavno (*recently*) vjerojatno će uskoro opet koristiti isti ili neki drugi proces
 - iz međuspremnika izbacivati (po potrebi uz operaciju *output*) blokove koje tijekom najduljeg perioda nije koristio niti jedan proces
 - menadžer međuspremnika mora za svaki blok u međuspremniku evidentirati zadnji trenutak u kojem je neki proces koristio taj blok
- FIFO algoritam je jednostavniji za implementaciju, ali manje efikasan
 - blokovi koji se najdulje nalaze u međuspremniku, bez obzira koliko ih intenzivno procesi koriste, prvi se izbacuju iz međuspremnika

* u nekim slučajevima se operacija *output* mora obaviti "odmah", bez obzira na LRU algoritam
→ u poglavlju o obnovi baze podataka

LRU - kada se ne koristi?

- slijedno čitanje relacije čija veličina nadmašuje raspoloživu veličinu međuspremnika. Npr. čitanje relacija od 10 000 blokova, uz raspoloživu veličinu međuspremnika od 5 000 blokova
 - posljedica: pročitana stranica koristi se tek kratko vrijeme te nepotrebno uzrokuje izbacivanje drugih, možda često korištenih, stranica
 - IBM Informix rješenje: *light scans*: koristi se drugačiji međuspremnik (*large buffer*) s većim U/I blokovima (npr. umjesto karakterističnih 2K, 4K, koriste se blokovi veličine 64K ili 128K), a LRU se ne primjenjuje
 - Oracle rješenje: koristi se regularni (LRU) međuspremnik, ali se tako pročitane stranice odmah označavaju kao *least recently used* stranice

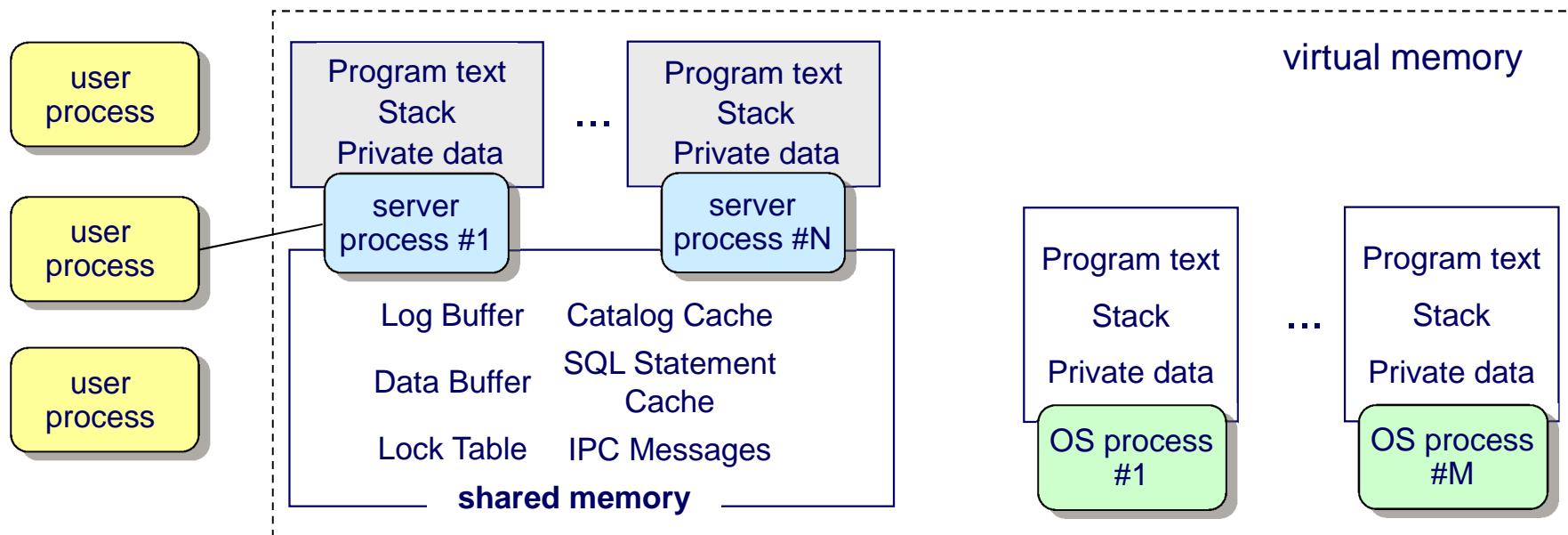


Menadžer međuspremnika i virtualna memorija OS-a

- kapacitet virtualne memorije operacijskog sustava je uobičajeno znatno veći od kapaciteta fizičke memorije
 - OS prema vlastitim algoritmima "izbacuje" stranice virtualne memorije iz fizičke memorije u tzv. *swap space*
- problem: stranice virtualne memorije koje SUBP koristi za međuspremnike mogu biti "izbačene" u *swap space* bez kontrole menadžera međuspremnika SUBP-a
- *forced residency*: postoje API funkcije operacijskih sustava koje omogućavaju da se dio logičkog adresnog prostora virtualne memorije proizvoljno dugo veže uz okvire radnog spremnika (fizičke memorije). Taj dio virtualne memorije OS neće izbacivati u *swap space*. Bolji SUBP omogućuju korištenje takvih mogućnosti OS-a
 - mogući nedostatak takvog pristupa: dio radnog spremnika unaprijed se veže uz procese SUBP-a. Ovisno o raspodjeli memorijskog opterećenja SUBP procesa i ostalih procesa na računalu, ili SUBP dio radnog spremnika ili ostatak radnog spremnika može ostati nepotpuno iskorišten
- neki SUBP omogućuju dinamičku (*on-line*) promjenu veličine dijela radnog spremnika koji se rezervira za potrebe SUBP-a
 - IBM IDS (Informix Dynamic Server): parametri RESIDENT, SHMADD, ...
 - Oracle 11g: parametri LOCK_SGA, SHARED_POOL_SIZE, ...

Uloga dijeljene memorije (*shared memory*)

- dijeljena memorija je mehanizam operacijskog sustava koji iz perspektive SUBP-a omogućava:
 - smanjenje broja U/I operacija
 - upravljanje međuspremnicima na temelju zajedničkih potreba svih procesa, umjesto na temelju svakog procesa pojedinačno
 - visoku brzinu komunikacije među procesima i dretvama (*IPC - inter-process communication*)
 - razmjena poruka među procesima se obavlja brzinom prijenosa u memoriji, nasuprot sporijim tehnikama, npr. cjevovodima (*pipes*)

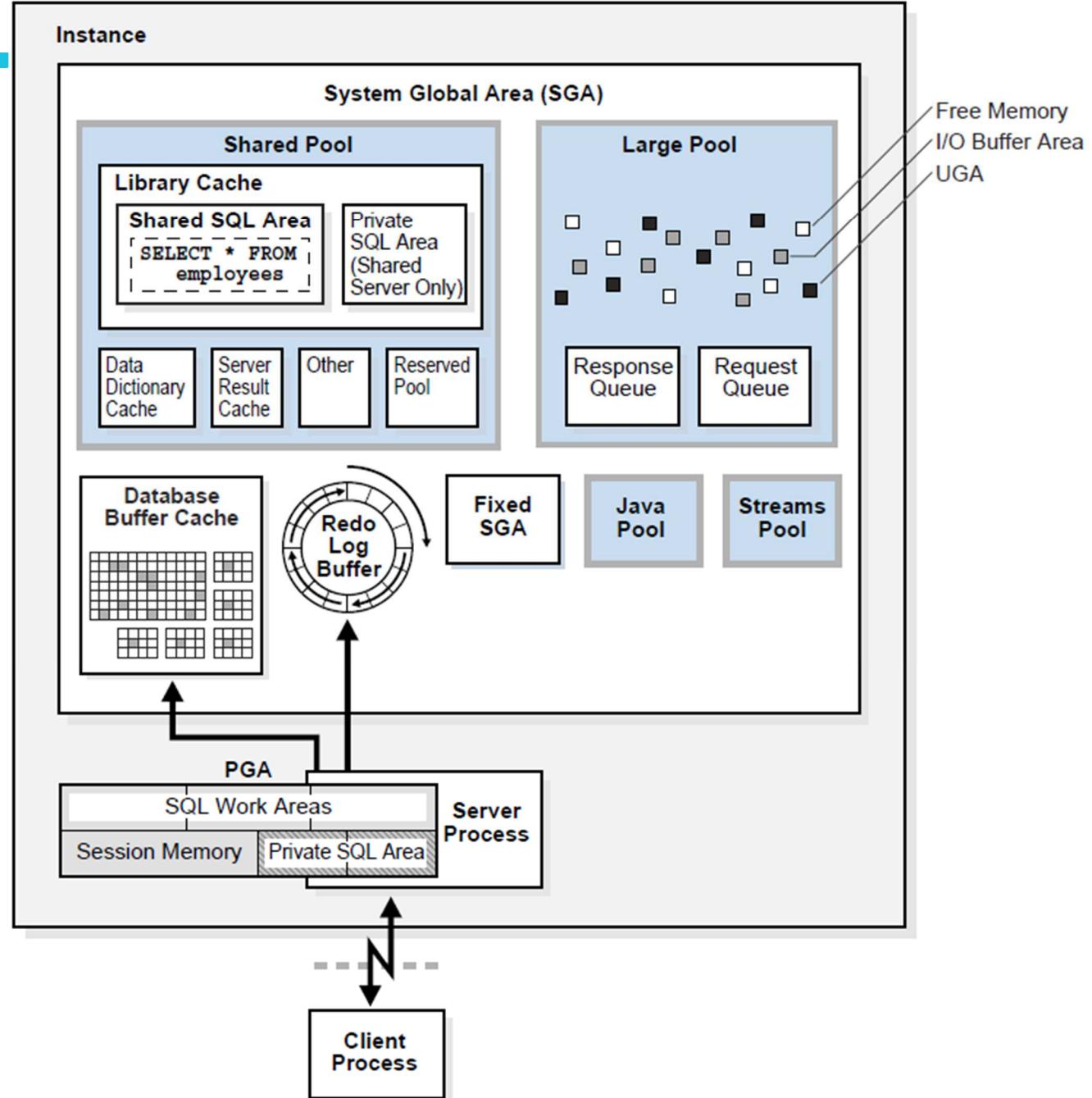


Uloga dijeljene memorije (*shared memory*)

dijeljena memorija u sustavu Oracle:

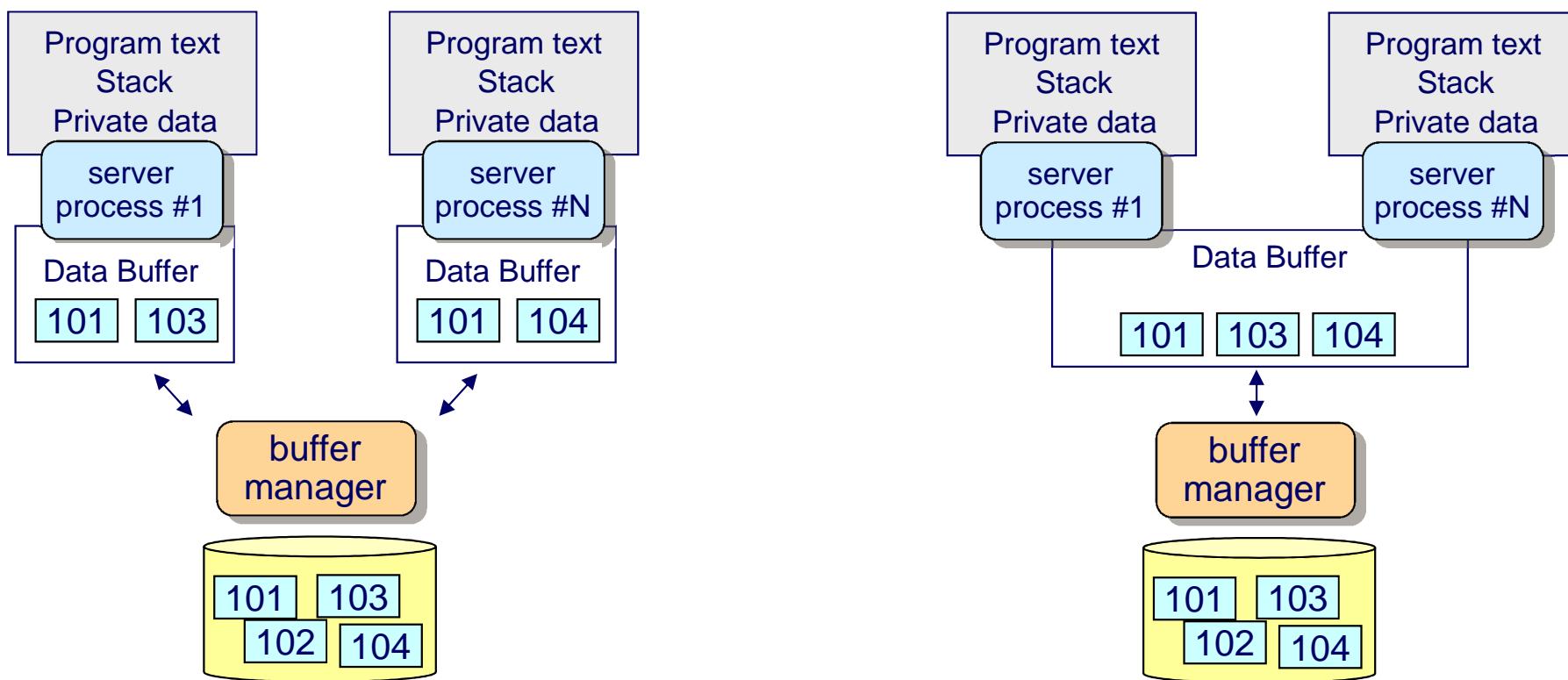
Izvor: Oracle® Database Concepts 11g

- *The SGA is a group of shared memory structures, known as SGA components, that contain data and control information for one Oracle Database instance. The SGA is shared by all server and background processes.*
- *A PGA is a nonshared memory region that contains data and control information exclusively for use by an Oracle process. One PGA exists for each server process and background process.*



Uloga dijeljene memorije (*shared memory*)

- u radnom spremniku se nalaze višestruke kopije istih blokova
- problem sinkronizacije: jesu li sve kopije istog bloka jednake?
- LRU na temelju pojedinačnih potreba procesa (loše)
- u radnom spremniku se nalazi samo jedna verzija bloka iz sekundarne memorije
- LRU na temelju zajedničkih potreba svih procesa (dobro)



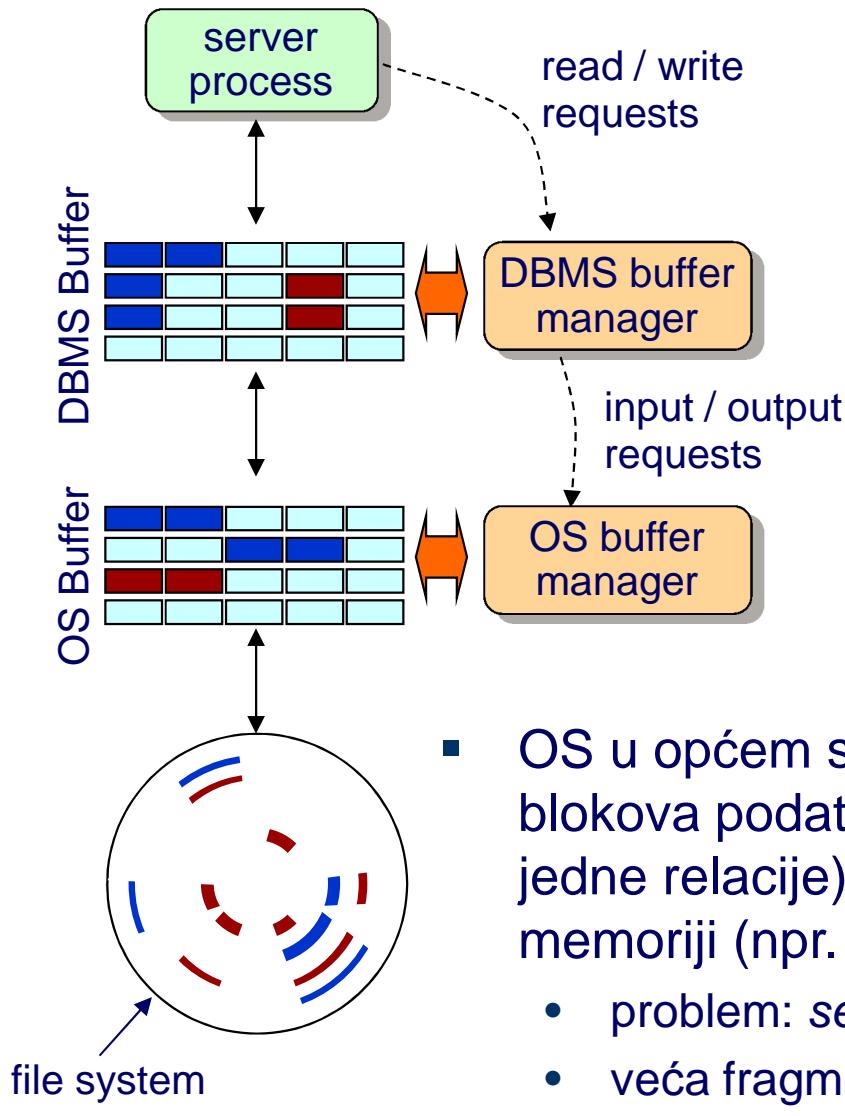
Utjecaj međuspremnika operacijskog sustava

- operacijski sustav koristi vlastite mehanizme za prijenos podataka između radnog spremnika i sekundarne memorije
 - podaci su u sekundarnoj memoriji organizirani u datotečni sustav (*file system*)
 - *block special device*: sučelje operacijskog sustava
 - podaci se prenose u blokovima, uz korištenje međuspremnika operacijskog sustava (često LRU algoritam) s ciljem poboljšanja performansi U/I operacija
- za SUBP koji za pohranu podataka u sekundarnoj memoriji koristi datotečni sustav i procedure operacijskog sustava (*system calls*) kaže se da koristi cooked files ili buffered files
- korištenje procedura operacijskog sustava rezultira jednostavnijom implementacijom SUBP-a, ali uz određene nedostatke: *double buffering*

Double buffering

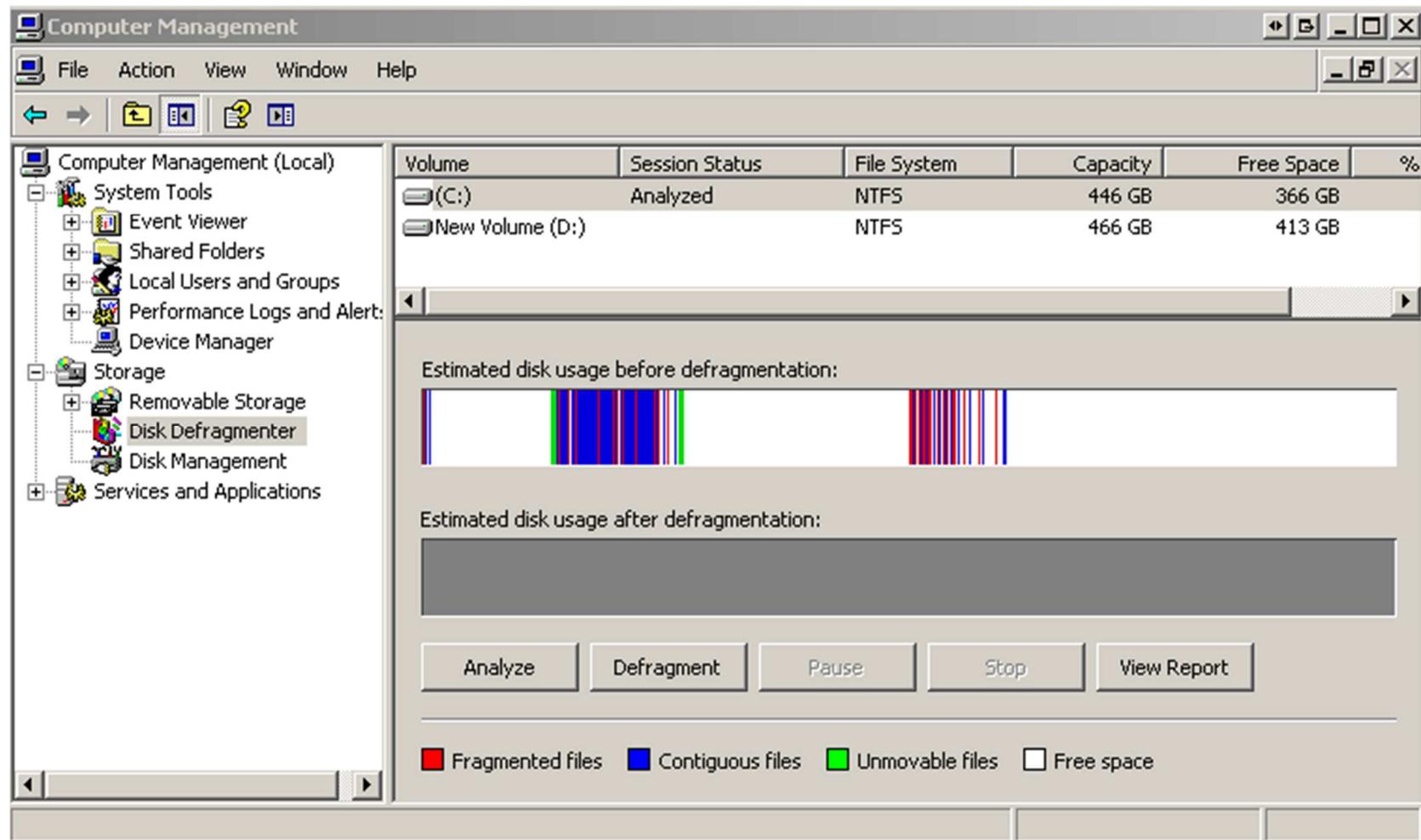
- budući da SUBP upravlja međuspremnicima optimalno u odnosu na operacije koje obavlja, međuspremni operacijskog sustava su redundantni i štetni
- troše primarnu memoriju koja se mogla bolje iskoristiti
- troše procesorsko vrijeme na nepotrebno kopiranje podataka
 - pisanje
 - međuspremnik SUBP → međuspremnik OS → sekundarna memorija
 - čitanje
 - sekundarna memorija → međuspremnik OS → međuspremnik SUBP

Cooked files



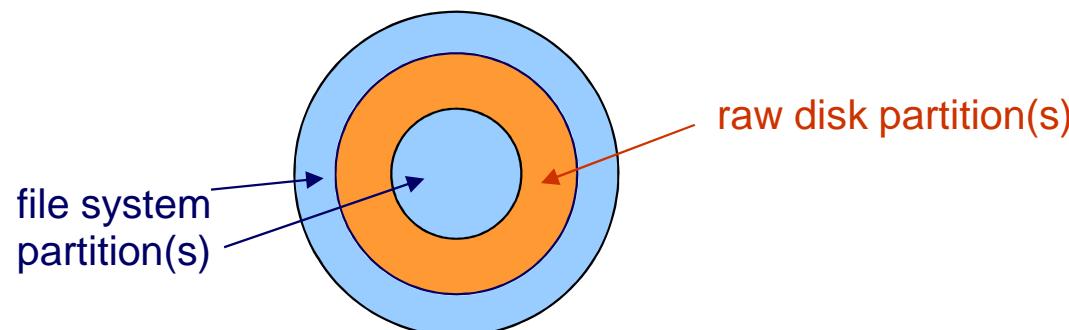
- korištenje međukoraka: disk \Leftrightarrow OS međuspremniči \Leftrightarrow SUBP međuspremniči \Rightarrow lošije performanse
- trošenje resursa s vrlo malim ili nikakvim dobitkom, korištenje procedura opće namjene (operacijski sustav) za visoko specijalizirane zadaće koje obavlja SUBP \Rightarrow lošije performanse
- OS u općem slučaju ne osigurava pohranu logički kontinuiranih blokova podataka (npr. blokova podataka koji sadrže n-torce jedne relacije) u fizički kontinuirani prostor u sekundarnoj memoriji (npr. u nekoliko susjednih traka)
 - problem: *seek time, rotational latency*
 - veća fragmentacija \Rightarrow sporiji pristup podacima \Rightarrow lošije performanse

Fragmentacija sekundarne memorije

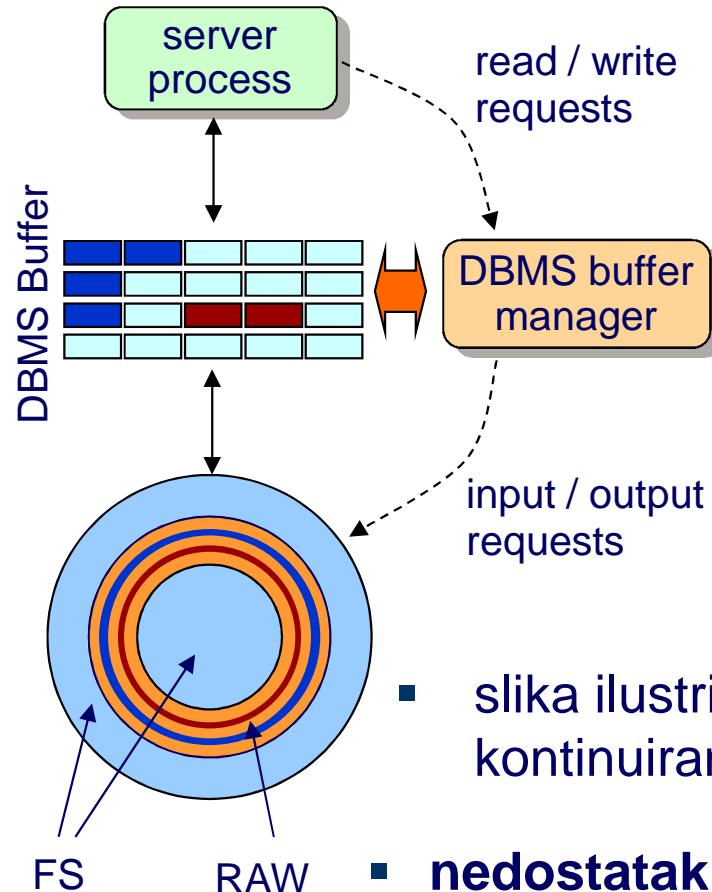


Raw disk I/O

- korištenjem *raw disk I/O* SUBP zaobilazi mehanizme međuspremnika i straničenja (*paging*) operacijskog sustava
 - SUBP u potpunosti upravlja procesima prijenosa podataka
 - npr. blokovi će se u sekundarnu memoriju zapisivati samo onda kada za to postoji potreba iz perspektive SUBP-a. U poglavljju o obnovi vidjet će se da SUBP može vrlo dugo odgađati zapisivanje promijenjenih stranica u sekundarnu memoriju
- termin *raw disk* odnosi se na način pristupa podacima u sekundarnoj memoriji: podacima se pristupa direktno, umjesto putem datotečnog sustava. Operacijski sustav omogućava *raw disk I/O* putem posebnih sučelja
 - *character special device*, *raw disk device*
- važno svojstvo: particija sekundarne memorije za koju se koristi *character special device* zauzima fizički kontinuirani prostor



Raw disk I/O



- logički kontinuirani blokovi podataka pohranjuju se u fizički kontinuiranom prostoru \Rightarrow bolje performanse
- nema međukoraka: disk \Leftrightarrow OS međuspremniči \Leftrightarrow SUBP međuspremniči \Rightarrow bolje performanse
- očekivano poboljšanje performansi (potvrđeno u praksi): 10-30%
- slika ilustrira kako se n-torce jedne relacije pohranjuju u fizički kontinuiranom prostoru
- **nedostatak:** particije sekundarne memorije se moraju unaprijed definirati (koliko i gdje prostora za SUBP, koliko i gdje za datotečni sustav)
 - nepažljivo planiranje diskovnog prostora može uzrokovati teškoće pri reorganizaciji prostora baze podataka ili datotečnog sustava

Značaj smještaja podataka u fizički kontinuiranom prostoru

- kritični faktori (HDD)
 - *seek time (3-10 ms)*
 - *rotational delay (2-5 ms)*
 - *transfer speed (~100 MB/s)*
- Sequential bandwidth to and from disk is between 10 and 100 times faster than random access, and this ratio is increasing. Disk density has been doubling every 18 months and bandwidth rises approximately as the square root of density (and linearly with rotation speed). Disk arm movement, however, is improving at a much slower rate — about 7%/year. As a result, it is critical for the DBMS storage manager to place blocks on the disk such that queries that require large amounts of data can access it sequentially.



Direct I/O

- moderni operacijski sustavi često omogućavaju korištenje direktnog pristupa memoriji u datotečnom sustavu (*Direct I/O*). Korištenjem te opcije zaobilaze se međuspremniči operacijskog sustava
 - operacijski sustav mora podržavati *Direct I/O* za primjenjeni datotečni sustav
 - Solaris UFS, Windows NTFS, ...
- performanse sustava koji koristi *Direct I/O* nad datotečnim sustavom često su sumjerljive performansama sustava koji koristi *raw disk*

Upravljanje prostorom za pohranu u SUBP-u

- različiti SUBP omogućavaju različite razine mogućnosti upravljanja prostorom za pohranu
- na primjeru sustava IBM IDS prikazan je jedan od modela. Ovdje je opisan samo dio najznačajnijih koncepata

Fizičke jedinice pohrane

- jedinice koje predstavljaju segmente sekundarne memorije (iz perspektive operacijskog sustava)
 - grumen
 - područje

Logičke jedinice pohrane

- jedinice koje predstavljaju logički povezane dijelove baze podataka (iz perspektive sustava za upravljanje bazama podataka)
 - prostor baze podataka
 - prostor relacije
- logičke jedinice pohrane se na razini SUBP-a mogu povezati s fizičkim jedinicama pohrane. Na taj način administrator SUBP-a može utjecati na fizički smještaj logičkih elemenata baze podataka

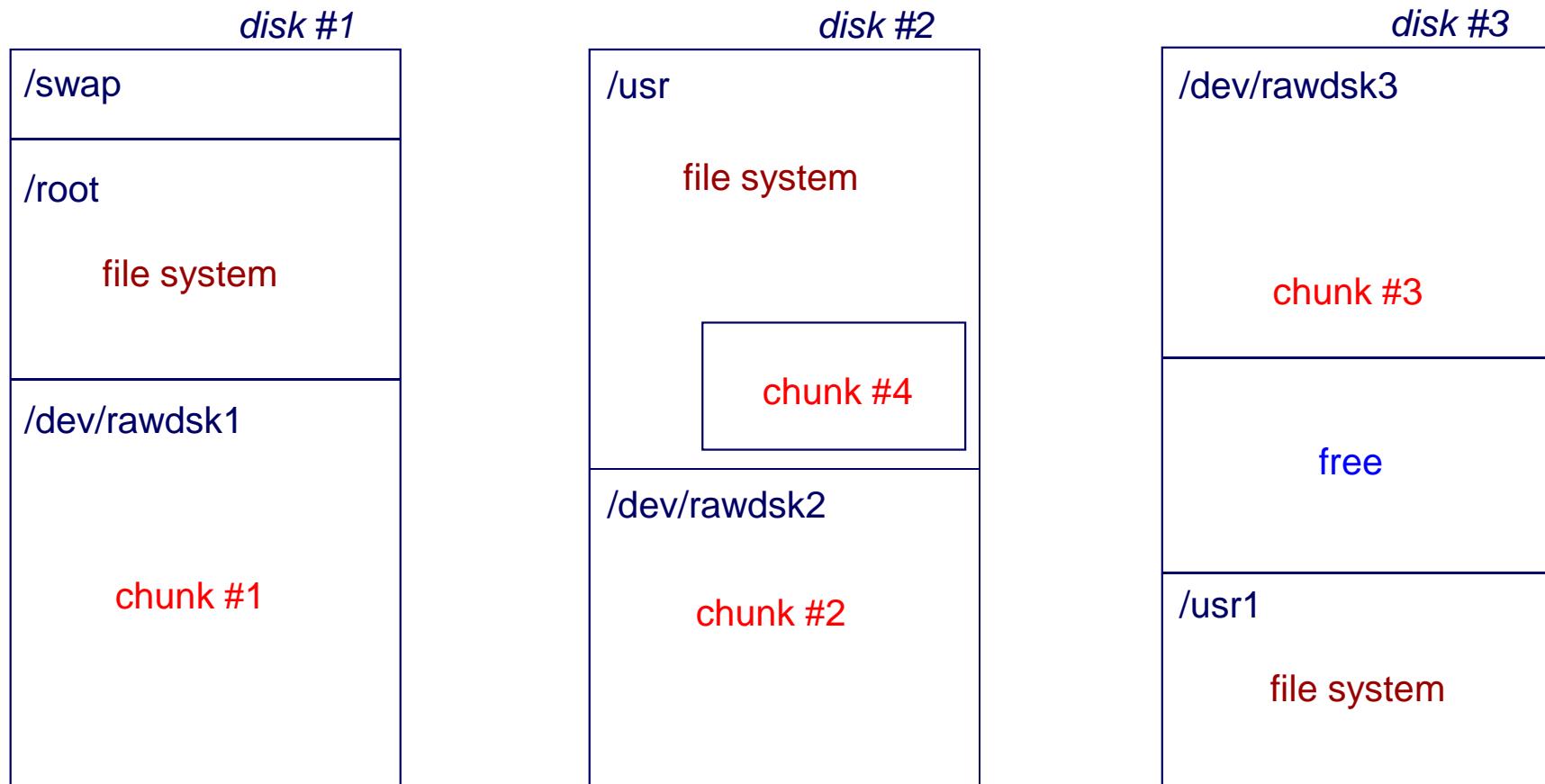
Fizičke jedinice pohrane

Grumen (*chunk*)

- grumen je najveća jedinica fizičkog prostora koja se s razine operacijskog sustava dodjeljuje sustavu za upravljanje bazama podataka. Predstavlja unaprijed alocirani prostor, u *raw device* obliku ili u obliku datoteke u datotečnom sustavu (*cooked file*).
- grumene definira administrator operacijskog sustava u dogovoru s administratorom sustava baza podataka (uobičajeno prilikom instalacije operacijskog sustava)
 - particije diska koje će se koristiti kao *raw device*
 - datoteke koje će se koristiti kao *cooked file*

Grumen (chunk)

Primjer:



- fizički kontinuirani prostor u grumenima 1, 2 i 3
- grumen 4 (datoteka) nije nužno fizički kontinuirani prostor
- dodavanje novih grumena je moguće samo na slobodnim particijama, na diskovima koji se naknadno dodaju ili u datotečnom sustavu u formi datoteka

Područje (extent)

Područje (extent)

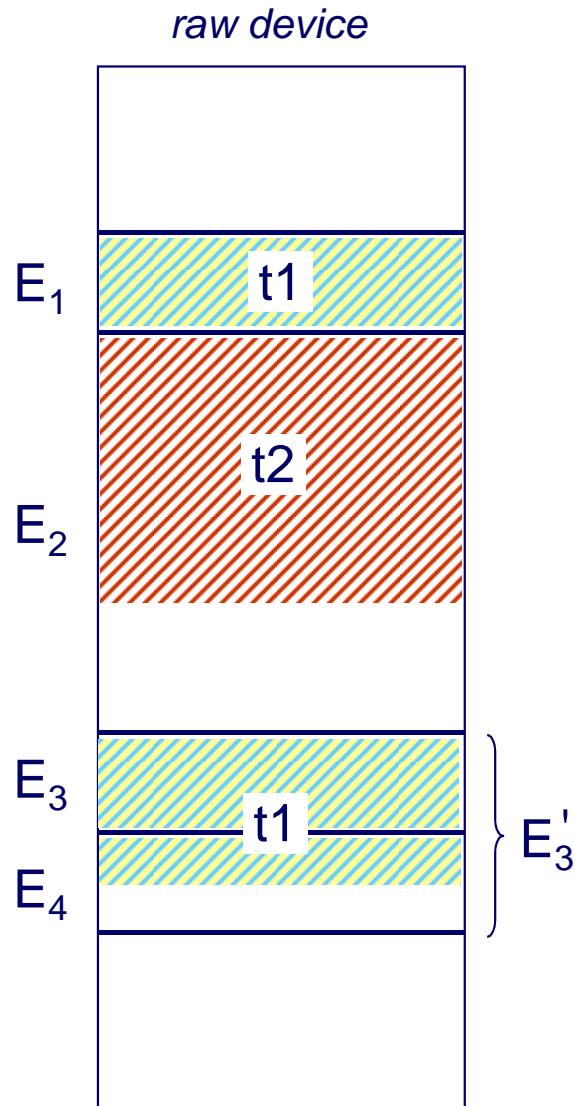
- SUBP nastoji zadržati podatke iste relacije u kontinuiranom prostoru sekundarne memorije
- u trenutku kreiranja relacije, SUBP rezervira fiksnu količinu prostora u sekundarnoj memoriji. Kada se taj inicijalno rezervirani prostor popuni, SUBP će alocirati sljedeći dio sekundarne memorije. Fizička jedinica prostora koja se rezervira u trenutku kreiranja relacije ili kasnije tijekom njenog proširenja, naziva se područje (*extent*)
- ako se koristi *raw device*, jedno područje je uvijek kontinuirani segment sekundarne memorije, inače, područje može biti više ili manje fragmentirano

```
CREATE TABLE ... (
    ...
) EXTENT SIZE m NEXT SIZE n;
```

- **m** i **n** su veličine područja izražene u kB
 - **m** veličina inicijalnog područja kod kreiranja relacije
 - **n** inkrementalna veličina dodatnog područja kod proširenja prostora relacije

Područje (extent)

Primjer:



CREATE TABLE t1 (...)
EXTENT SIZE 16 NEXT SIZE 16
⇒ alocira se $E_1 = 16\text{k}$

CREATE TABLE t2 (...)
EXTENT SIZE 64 NEXT SIZE 32
⇒ alocira se $E_2 = 64\text{k}$

INSERT INTO t1 (40 kB podataka)
⇒ popunjava se E_1 (16 kB)
⇒ alocira se E_3 (16 kB)
⇒ popunjava se E_3 (16 kB)
⇒ alocira se E_4 (16 kB)
⇒ E_3 i E_4 se spajaju u E'_3
⇒ popunjava se dio E'_3 (8 kB)

INSERT INTO t2 (40 k podataka)
⇒ popunjava se dio E_2 (40 kB)

Područje (*extent*)

- dva područja iste relacije, koja su alocirana neposredno jedno iza drugog, spajaju se u jedno područje
- inkrementalna veličina područja se može promijeniti naredbom ALTER TABLE
- **CILJ:** optimirati između
 - nepotrebnog zauzeća prostora
 - broja područja koja se pojedinačno alociraju
- ako broj fragmenata neke relacije postane prevelik, potrebno je obaviti defragmentaciju
 - npr. IDS Informix:
`EXECUTE FUNCTION admin ('defragment', 'imeBaze:imeRelacije');`

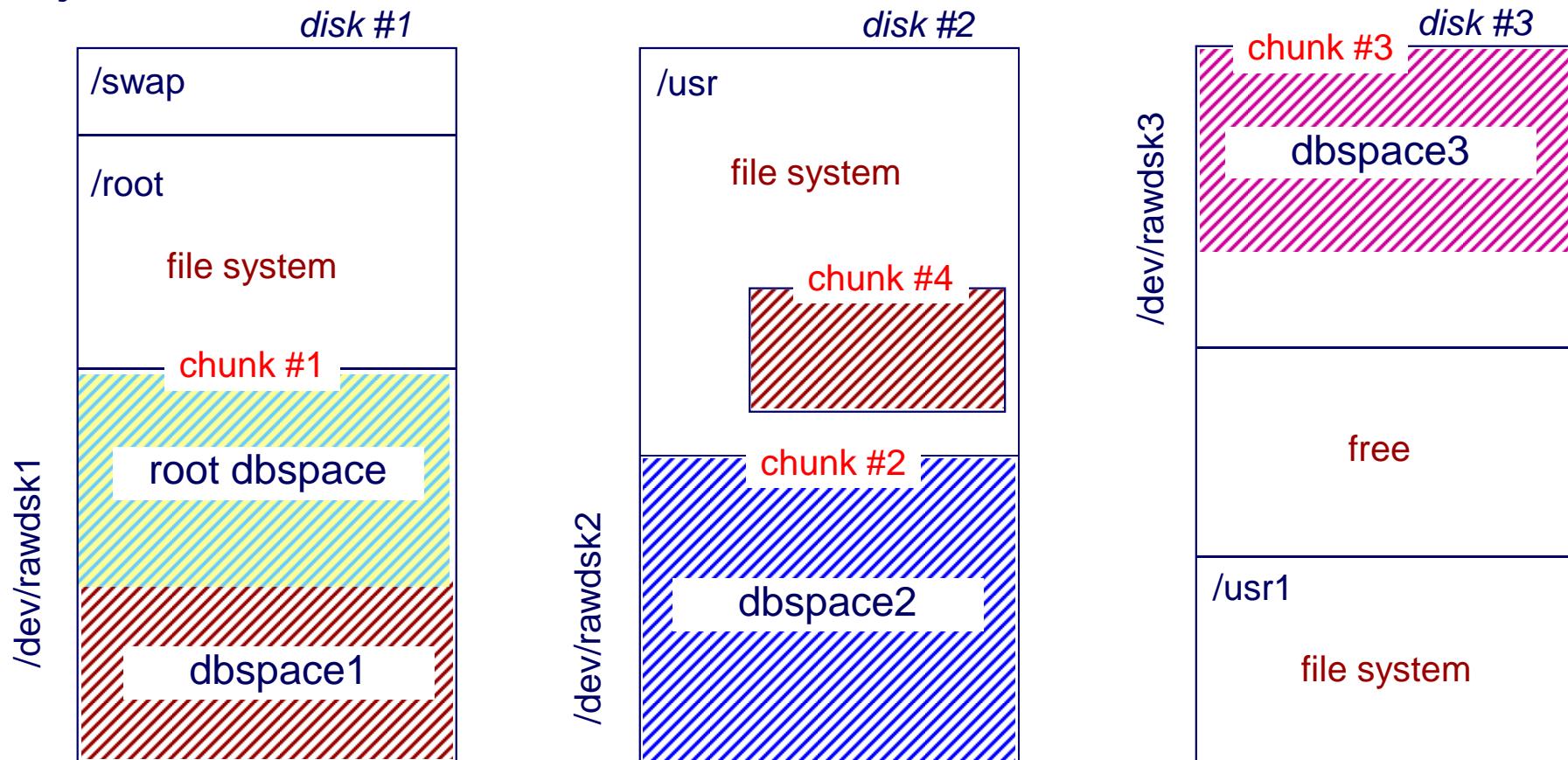
Logičke jedinice pohrane

Prostor baze podataka (*database space*)

- naziv, veličinu i smještaj prostora baze podataka određuje administrator SUBP-a
- prostor baze podataka formira se u jednom ili više dijelova jednog ili više grumena. Na taj način se najveća logička jedinica pohrane povezuje s najvećom fizičkom jedinicom pohrane
 - prostor baze podataka se može proširivati novim grumenima. Npr. ako nema dovoljno prostora u inicijalno alociranom prostoru (primjer *dbspace1* na sljedećoj stranici)
- administrator baze podataka može odabrati prostor baze podataka u kojem će kreirati objekt baze podataka (npr. relaciju). Na taj način je indirektno određen i fizički smještaj relacije
- *root dbspace* je specijalan prostor baze podataka koji se uvijek kreira u trenutku inicijalizacije instance SUBP-a
 - sadrži nužne sistemske podatke potrebne za rad poslužitelja

Prostor baze podataka

Primjer:



- *root dbspace* je kreiran u dijelu grumena #1
- *dbspace1* je kreiran u dijelu grumena #1 i grumenu #4
- *dbspace2* je kreiran u grumenu #2
- *dbspace3* je kreiran u dijelu grumena #3

Prostor baze podataka

- administrator baze podataka može odrediti gdje će biti smještena baza podataka i pojedine relacije

CREATE DATABASE dbName;

- baza podataka se pohranjuje u *rootdbspace*, a također i sve pripadne relacije za koje se CREATE TABLE naredbom ne specificira drugačije

CREATE DATABASE dbName IN dbSpace1;

- baza podataka se pohranjuje u prostor *dbspace1*, a također i sve pripadne relacije za koje se CREATE TABLE naredbom ne specificira drugačije

CREATE TABLE tabname (...);

- relacija se pohranjuje u prostoru baze podataka koji je određen pripadnom CREATE DATABASE naredbom

CREATE TABLE tabname (...) IN dbSpace1;

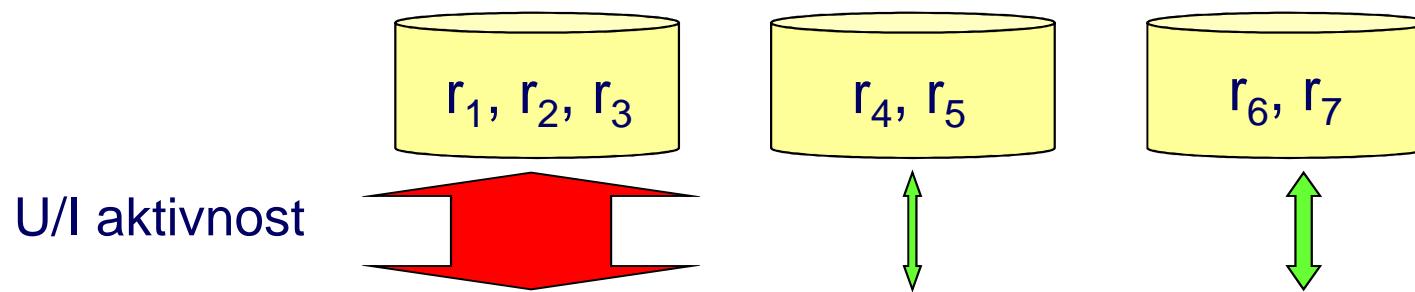
- relacija se pohranjuje u prostor *dbspace1*, bez obzira na oblik pripadne CREATE DATABASE naredbe

Prostor relacije (*table space*)

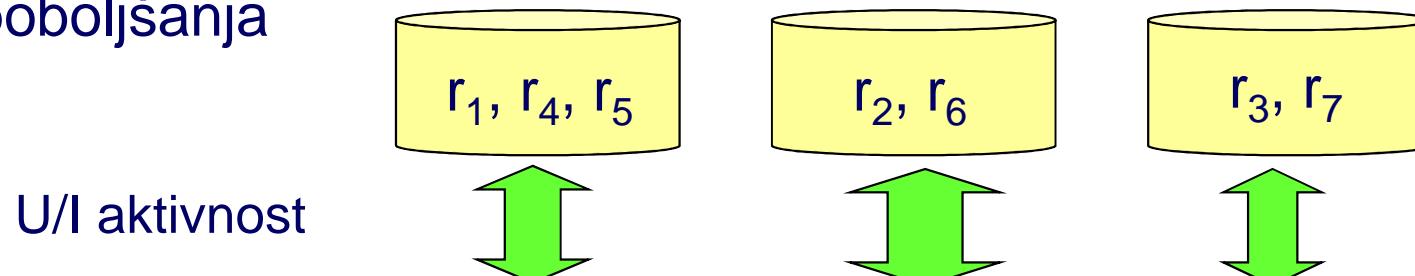
- prostor relacije je skup svih područja (*extent*) koja sadrže podatke jedne relacije
- u idealnom slučaju prostor relacije je fizički kontinuirani prostor
- ako prostor potreban za pohranu relacije nije alociran odjednom ili ako se se područja (*extent*) alociraju unutar *cooked files* grumena, podaci jedne relacije mogu biti fragmentirani, odnosno smješteni u fizički diskontinuiranom prostoru

Utjecaj fizičkog smještaja na performanse sustava

- U/I uređaji (*I/O devices*) često predstavljaju uska grla. Npr. neka su
 - relacije r_1, r_2, r_3 relacije s visokom U/I aktivnošću
 - relacije r_4, r_5, r_6, r_7 relacije s niskom U/I aktivnošću
- visoka razina natjecanja (*contention*) za pristup U/I uređajima rezultira pojavom "vrućih točaka", *hotspots*



- boljim planiranjem prostora za pohranu, mogu se postići velika poboljšanja



Kako fizičkim smještajem podataka utjecati na performanse?

- nekoliko pravila kojima se treba poslužiti pri planiranju prostora za pohranu.
Relacije s najvišom U/I aktivnošću:
 - smjestiti na zasebne U/I jedinice
 - ako postoje U/I jedinice različite brzine, smjestiti ih na najbrže U/I jedinice
 - smjestiti ih čim bliže središnjim cilindrima diska
 - kreirati *raw device* u blizini središnjih cilindara
 - nad njim kreirati grumen (*chunk*)
 - u njemu kreirati prostor baze podataka
 - u prostoru baze podataka kreirati relaciju
- kako smjestiti jednu relaciju koja se među ostalima ističe visokom U/I aktivnošću?
 - fragmentacija (*fragmentation, partitioning*) omogućava smještaj podataka jedne relacije na više U/I uređaja. Koriste se sljedeće sheme fragmentacije:
 - *round-robin*
 - *expression based*

Shema fragmentacije relacije: *round-robin*

- n-torce se smještaju u fragmente na "slučajan" način. Pri tome se fragment određuje kao rezultat hash funkcije nad brojem dobivenim generatorom slučajnih brojeva ili se fragmenti ciklično izmjenjuju (jedna n-torka u prvi fragment, jedna u drugi fragment, ...). Postiže se jednolika razdioba n-torki.

```
CREATE TABLE ispit (
    jmbag      CHAR(10)
, sifPred    INTEGER
, datIspit  DATE
, sifNast   INTEGER
, ocjena    SMALLINT
, PRIMARY KEY (jmbag
                , sifPred
                , datIspit)
    CONSTRAINT pkIspit
) FRAGMENT BY ROUND ROBIN
IN dbs1, dbs2, dbs3, dbs4;
```

```
SELECT sifPred, AVG(ocjena)
  FROM ispit
 WHERE YEAR(datIspit) = 2002
 GROUP BY sifPred;
```

- operacije prijenosa blokova (*input*) mogu se obavljati paralelno nad svim U/I uređajima u kojima je pohranjena relacija ispit
- shema je prikladna kada se izvršavaju upiti koji pristupaju relativno velikom broju n-torki

Shema fragmentacije relacije: *expression based*

- n-torce se smještaju u fragmente na temelju predikata kojeg zadovoljavaju

```
CREATE TABLE student (
    jmbag      CHAR(10)
,  ime       CHAR(20)
,  prez      CHAR(20)
,  sifFakul INTEGER
,  PRIMARY KEY (jmbag)
    CONSTRAINT pkStudent
) FRAGMENT BY EXPRESSION
(sifFakul=36) IN dbs1
, (sifFakul=120)IN dbs2
, (sifFakul=210)IN dbs3
, REMAINDER IN dbs4;
```

```
SELECT * FROM student
WHERE prez LIKE 'Hor%'
    AND sifFakul=120;
```

- SUBP pristupa samo onim fragmentima koji sadrže podatke koji zadovoljavaju uvjet selekcije (ako uvjet sadrži kriterij iz sheme fragmentacije)
- shema je prikladna kada se izvršavaju upiti čiji uvjeti za selekciju sadrže kriterij naveden u shemi fragmentacije

Vertikalna fragmentacija relacija

- u relaciji se nalazi 10^6 n-torki

```
CREATE TABLE osoba (
    oib          CHAR(10)
, prez_ime    CHAR(86)
, visina      INTEGER
, zivotopis  CHAR(10000)
, PRIMARY KEY (oib)
    CONSTRAINT pkOsoba);
```

- u grubo procijeniti: koliko blokova veličine 2K je potrebno prenijeti (sekundarna→primarna memorija) pri obavljanju sljedećeg upita:

```
SELECT AVG(visina)
  FROM osoba;
```

$$\rightarrow 10^6 * 10100 / 2048 \approx 4.9 * 10^6 \text{ blokova}$$

Vertikalna fragmentacija relacija

- vertikalna fragmentacija
 - stvara se nova relacija s primarnim ključem jednakim kao u originalnoj relaciji
 - u novu relaciju prebacuju se atributi koji zauzimaju veliki prostor, a u upitima se ne koriste često

```
CREATE TABLE osoba (
    oib      CHAR(10)
, prez_ime CHAR(86)
, visina   INTEGER
, PRIMARY KEY (oib)
);
```

```
CREATE TABLE biogr (
    oib      CHAR(10)
, zivotopis CHAR(10000)
, PRIMARY KEY (oib)
, FOREIGN KEY ... osoba...
);
```

- u grubo procijeniti: koliko blokova veličine 2K je potrebno prenijeti (sekundarna→primarna memorija) pri obavljanju sljedećeg upita:

```
SELECT AVG(visina)
FROM osoba;
```

$$\rightarrow 10^6 * 100 / 2048 \approx 4.9 * 10^4 \text{ blokova}$$

Literatura:

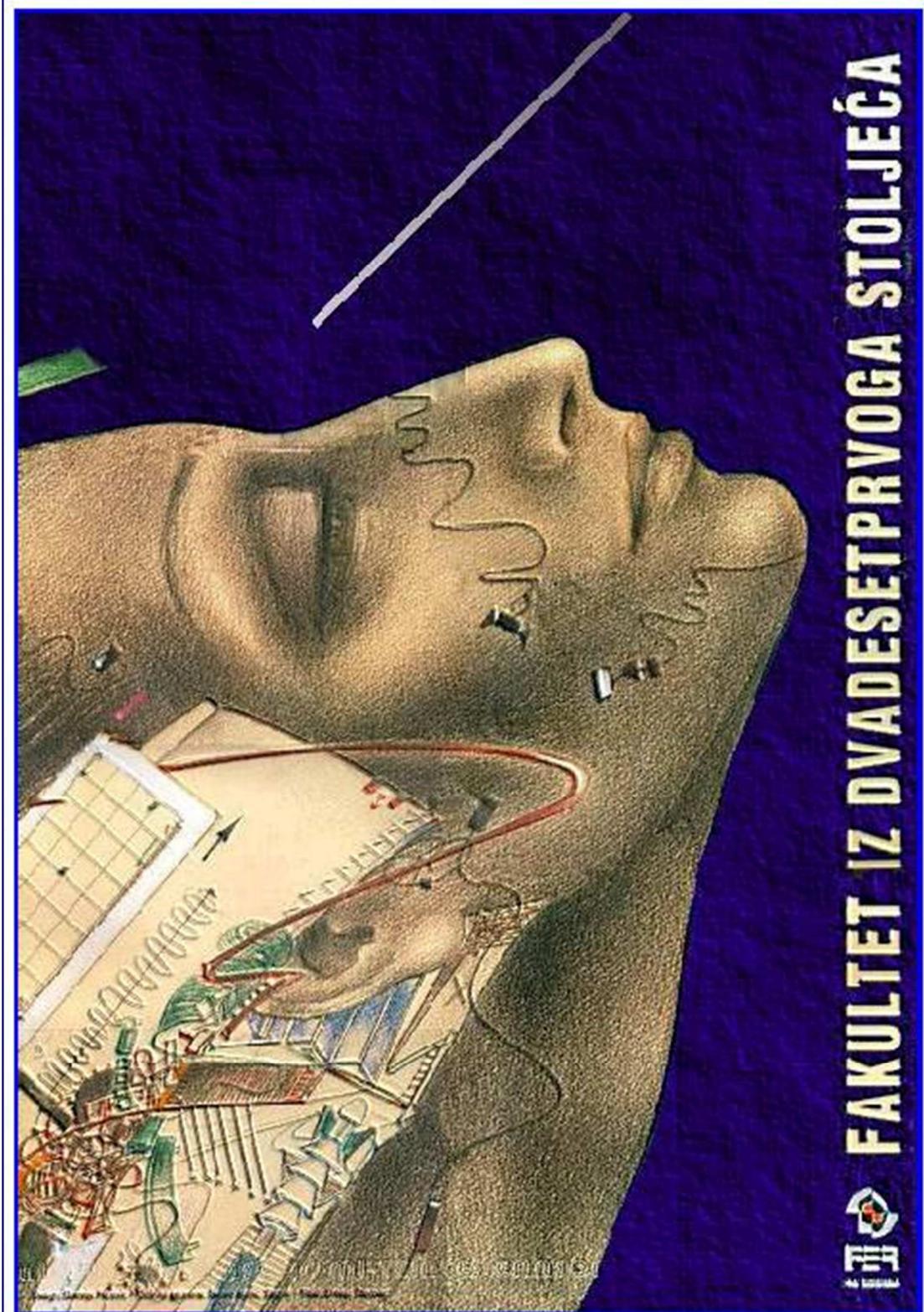
- H. Garcia-Molina, J. Ullman, J. Widom. Database system implementation. Prentice-Hall International, Inc. Upper Saddle River, New Jersey. 2000.
- A. Silberschatz, H. Korth, S. Sudarshan. Database system concepts, 4th ed. McGraw-Hill. 2002.
- R. Elmasri, S. Navathe. Fundamentals of database systems, 3rd ed. Addison-Wesley Publishing Company. Reading, Massachusetts. 2000.
- IBM Informix Dynamic Server Administrator's Reference, Version 11.1, IBM, 2007.
- IBM Informix Dynamic Server Administrator's Guide, Version 11.1, IBM, 2007.
- Oracle® Database Performance Tuning Guide, 11g Release 2 (11.2)
- Oracle® Database Concepts 11g Release 2 (11.2)

Sustavi baza podataka

Predavanja

2. Fizička organizacija

ožujak 2014.



FAKULTET IZ DVADESETPRVOGA STOLJEĆA

Fizička organizacija

- fizička organizacija podataka ima vrlo veliki utjecaj na efikasnost (*efficiency*) sustava za upravljanje bazama podataka
 - važna zadaća sustava za upravljanje bazama podataka: obavljati operacije nad velikim količinama podataka **na efikasan način**
- pojam fizičke organizacije podataka odnosi se na:
 - organizacija datoteke (*file organization*): strukture podataka primjenjene pri pohrani podataka u sekundarnoj memoriji
 - metode pristupa (*access methods*): postupci koji se primjenjuju pri obavljanju operacija nad podacima (*find, read, findNext, insert, modify, delete*)
- primjenjivost pojedinih metoda pristupa podacima ovisi o primijenjenim strukturama podataka
 - za pohranu sadržaja relacije nastojati primijeniti strukturu podataka koje će omogućiti efikasno obavljanje onih operacija koje se nad tom relacijom najčešće obavljaju
 - često je potreban kompromis

Važniji ciljevi fizičke organizacije

- minimizirati broj U/I operacija pri pohrani i dohvatu podataka, minimizirati utrošak prostora za pohranu
 - u koji fizički blok pohraniti logički zapis odnosno n-torku
 - koje je dodatne informacije potrebno pohraniti da bi se omogućio efikasan pristup podacima
- omogućiti različite metode pristupa koje se koriste za pronalaženje fizičke pozicije zapisa (ili fizičke pozicije bloka u kojem se taj zapis nalazi) na temelju vrijednosti ključa pretrage
 - ključ pretrage (*search key*) ne mora nužno biti primarni ili alternativni ključ. Ključ pretrage može biti bilo koji atribut ili skup atributa relacije ("sekundarni ključ").

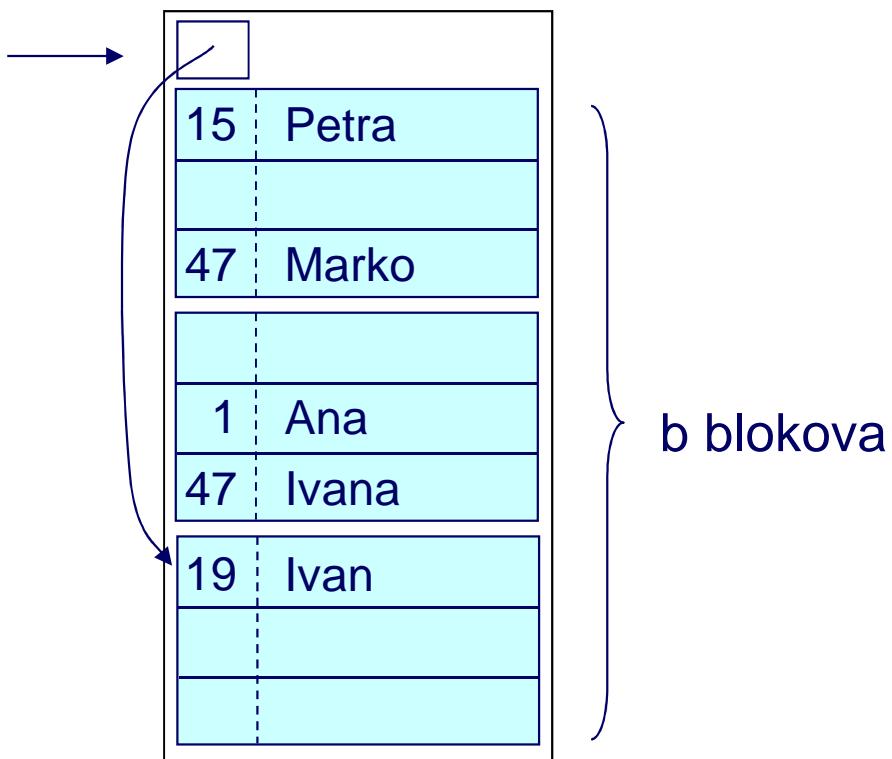
Strukture podataka i metode pristupa podacima

- primjena različitih struktura podataka omogućava različite metode pristupa podacima
- ne postoji "najbolja" metoda fizičke organizacije, ali neke od njih se u današnjim sustavima za upravljanje bazama podataka češće koriste
 - **Neporedana datoteka**
 - Poredana datoteka (*sorted file*)
 - **Raspršena datoteka**
 - Indeksno-slijedna organizacija (*index-sequential file*)
 - **B-stablo**

Neporedana datoteka

- *heap file, pile file, file of unordered records*
- pozicija zapisa u datoteci ovisi o redoslijedu unosa
- datoteka sadrži b blokova
- pristup podacima (dohvat podatka sa zadanom vrijednošću ključa pretrage) moguć je isključivo linearnim (slijednim) pretraživanjem (*linear search*)

adresa zadnjeg bloka datoteke
u zaglavlju datoteke (*file header*)



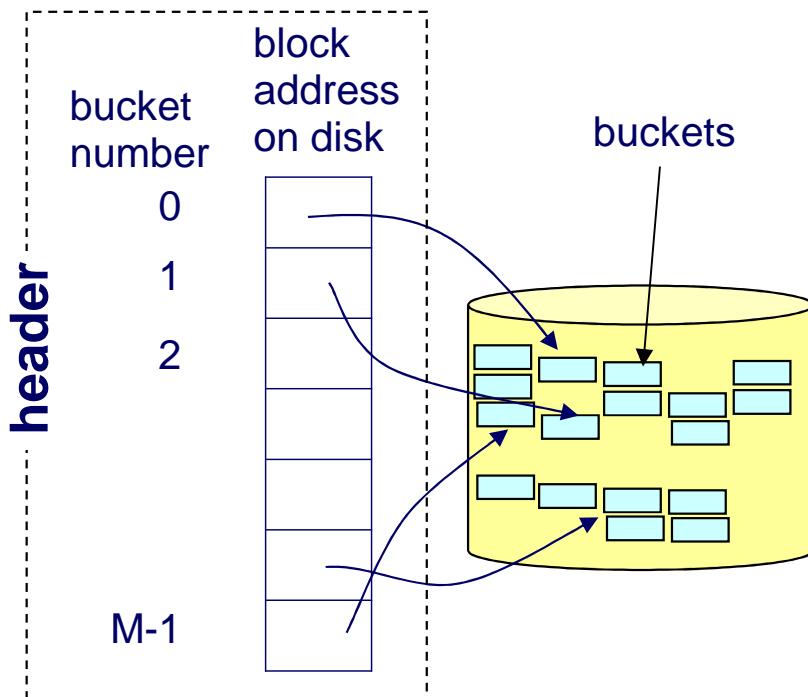
Neporedana datoteka

15	Petra
47	Marko
1	Ana
47	Ivana
19	Ivan

- operacija unosa je vrlo efikasna: zadnji blok datoteke prenosi se u međuspremnik, u blok se upiše novi zapis, blok se zapiše u sekundarnu memoriju
- traženje zapisa sa zadanim ključem
 - ako se radi o primarnom ključu zapisa koji postoji tada je prosječan broj potrebnih U/I operacija $b/2$
 - ako ključ potrage nije primarni ključ ili zapis ne postoji, broj obavljenih U/I operacija je b
- brisanje zapisa je najčešće logičko
- ova organizacija podataka se često koristi u kombinaciji s drugim organizacijama (npr. B-stablo)

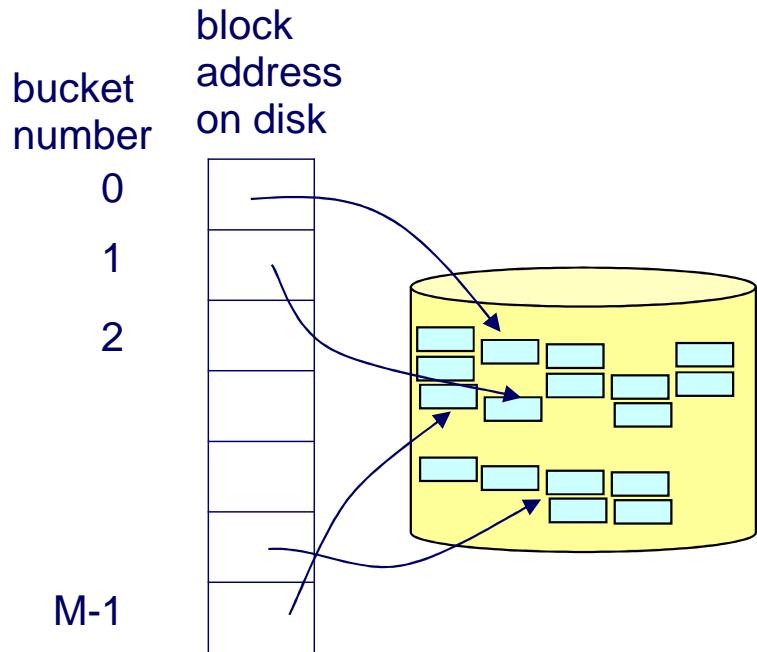
Raspršena datoteka

- *hash files*: vrlo slično raspršenoj organizaciji u primarnoj memoriji
- specifičnosti raspršene organizacije u sekundarnoj memoriji
 - *external hashing, secondary storage hashing*
- pretinac (*bucket*): jedan blok ili više uzastopnih blokova u sekundarnoj memoriji
- *hash-funkcijom* se iz ključa izračunava relativna adresa pretinca



- samo u slučaju kada je cijela datoteka pohranjena u fizički kontinuiranom prostoru, fizička adresa pretinca može se izračunati na temelju rezultata *hash* funkcije
- najčešće se (slika) u zaglavljima datoteke održava tablica koja preslikava broj pretinca (0 do M-1) u fizičke adrese pretinaca na disku

Raspršena datoteka



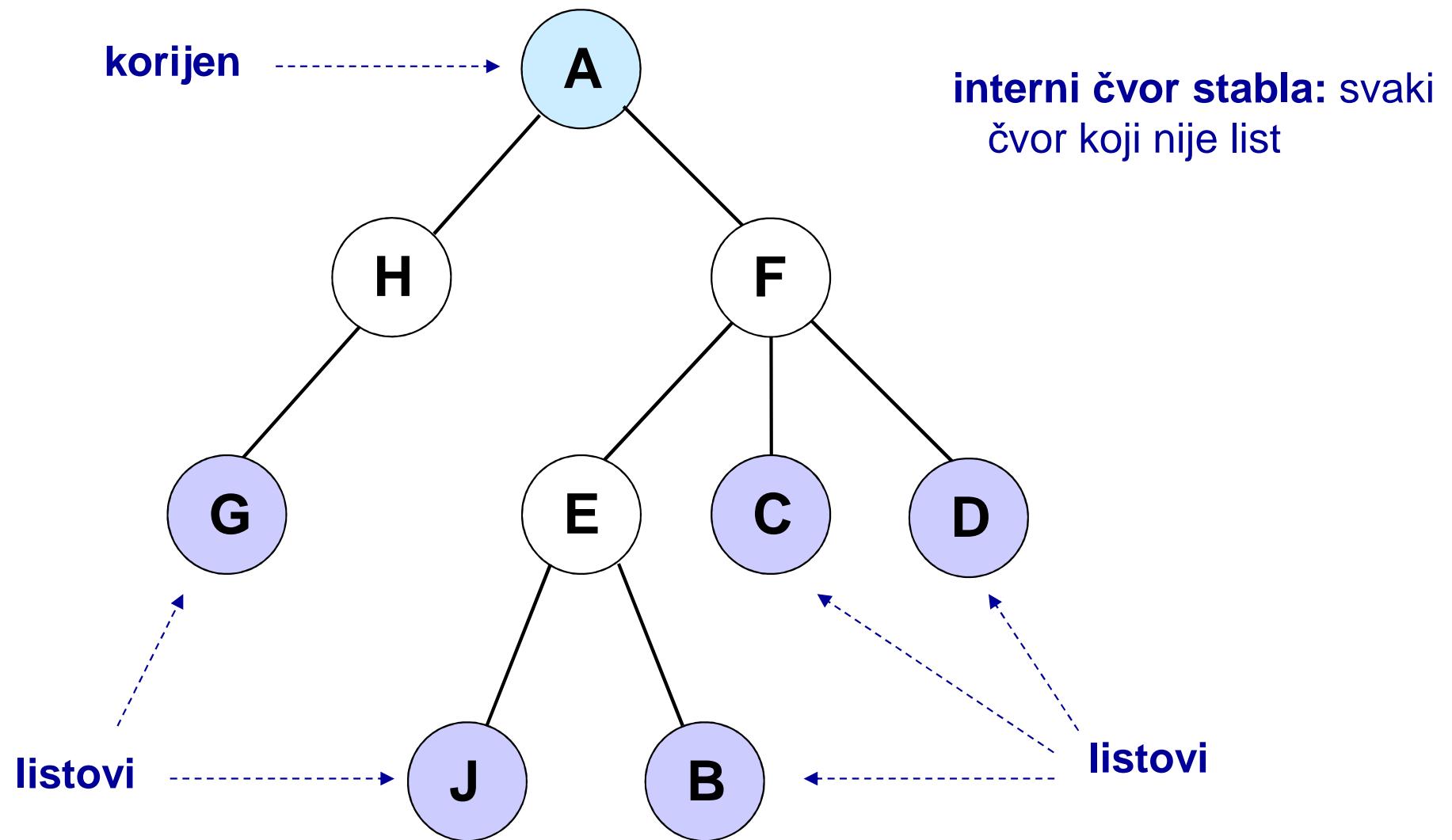
Važna svojstva

- najbrža metoda za operaciju pretraživanja prema ključu (najčešće samo jedna U/I operacija)
- efikasno dodavanje i brisanje zapisa
- problem: eventualni preljevi → ulančavanje
- pretraživanje moguće samo prema ključu (ne intervalu)
- zapisi nisu sortirani (osim za posebne hash-funkcije)

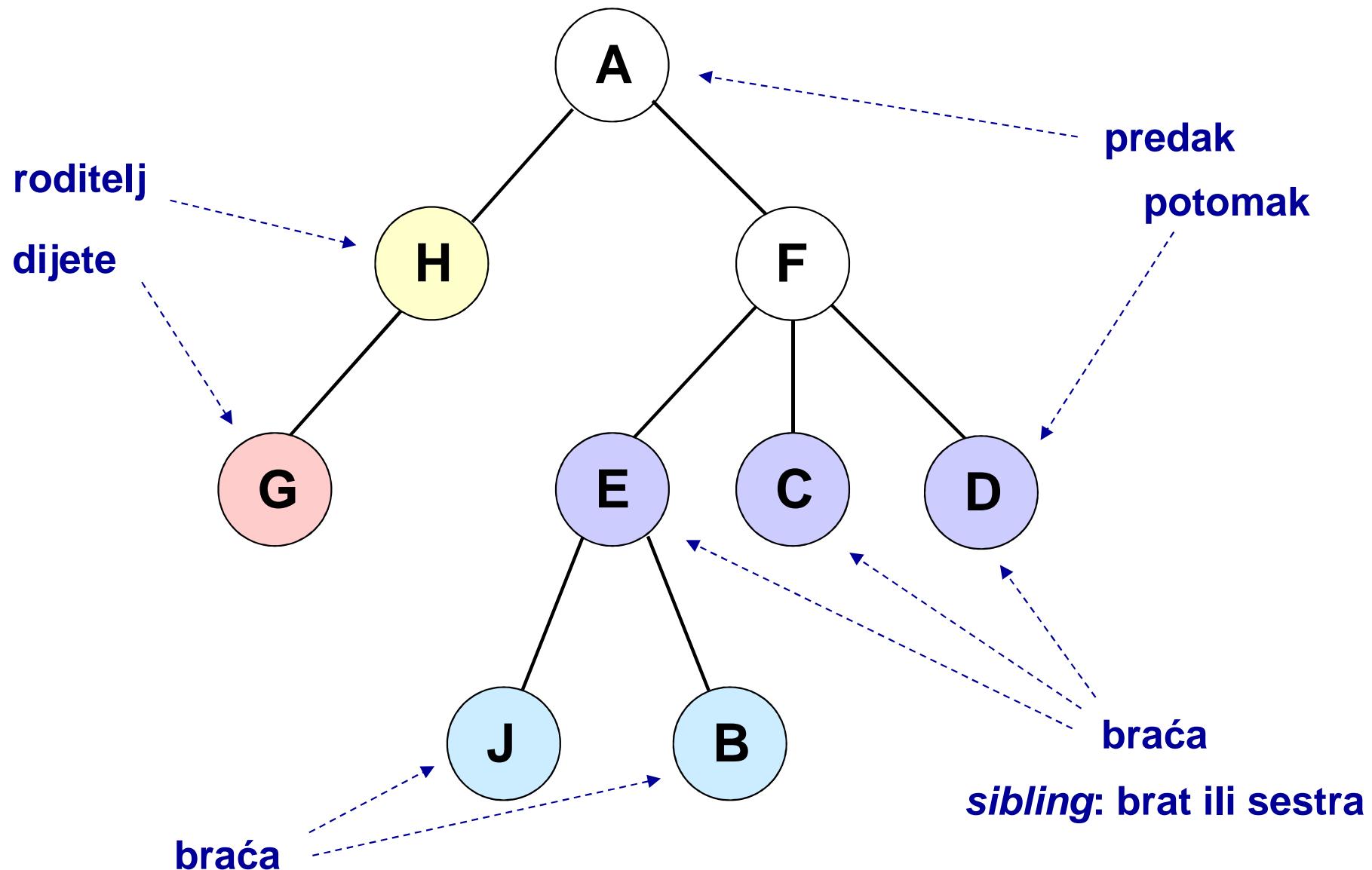
- za naprednije metode organizacije (*extendible hashing, linear hashing*) pogledati u literaturi: [Garcia-Molina] i [Elmasri]

B-stablo

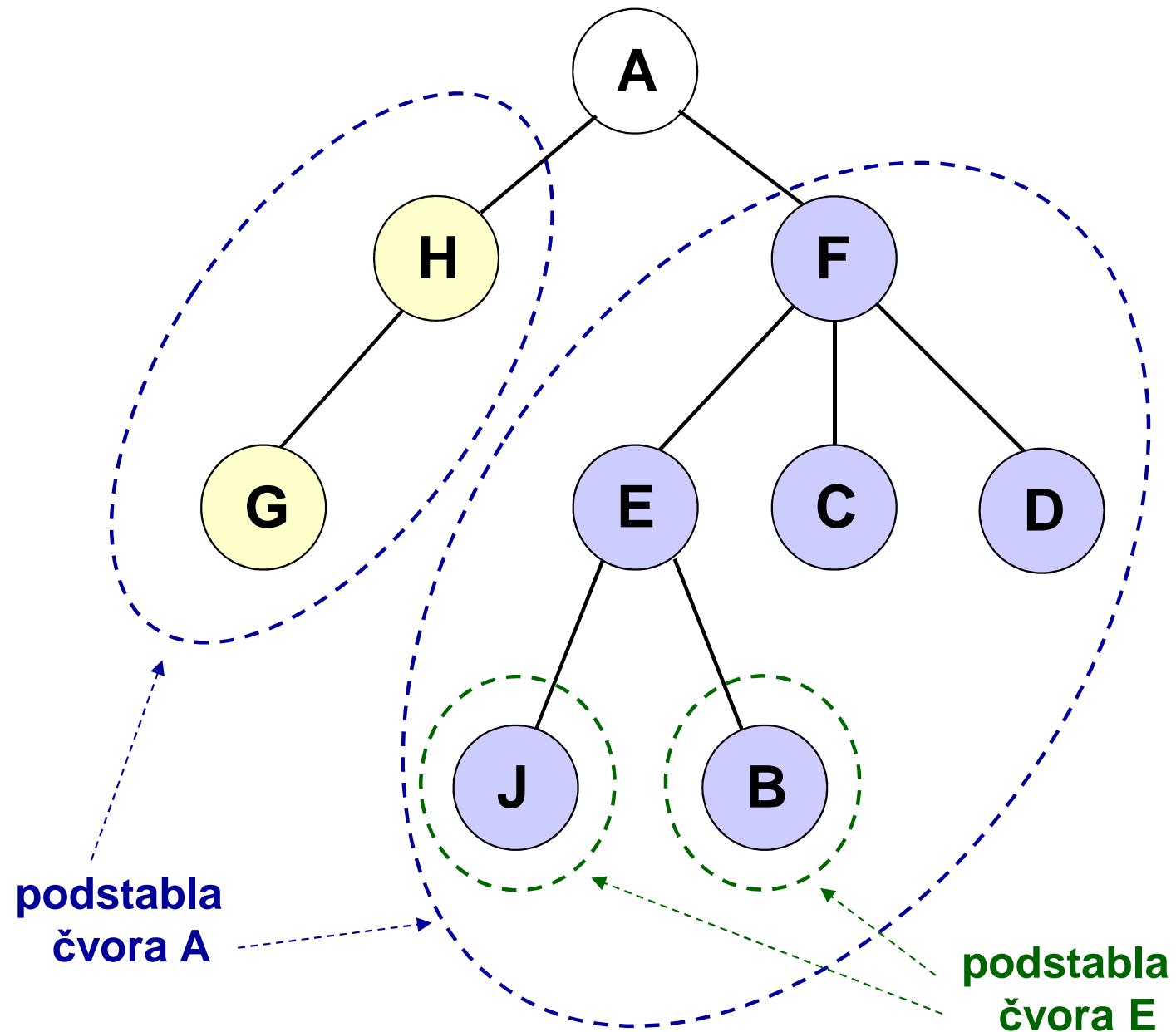
Stablo kao struktura podataka



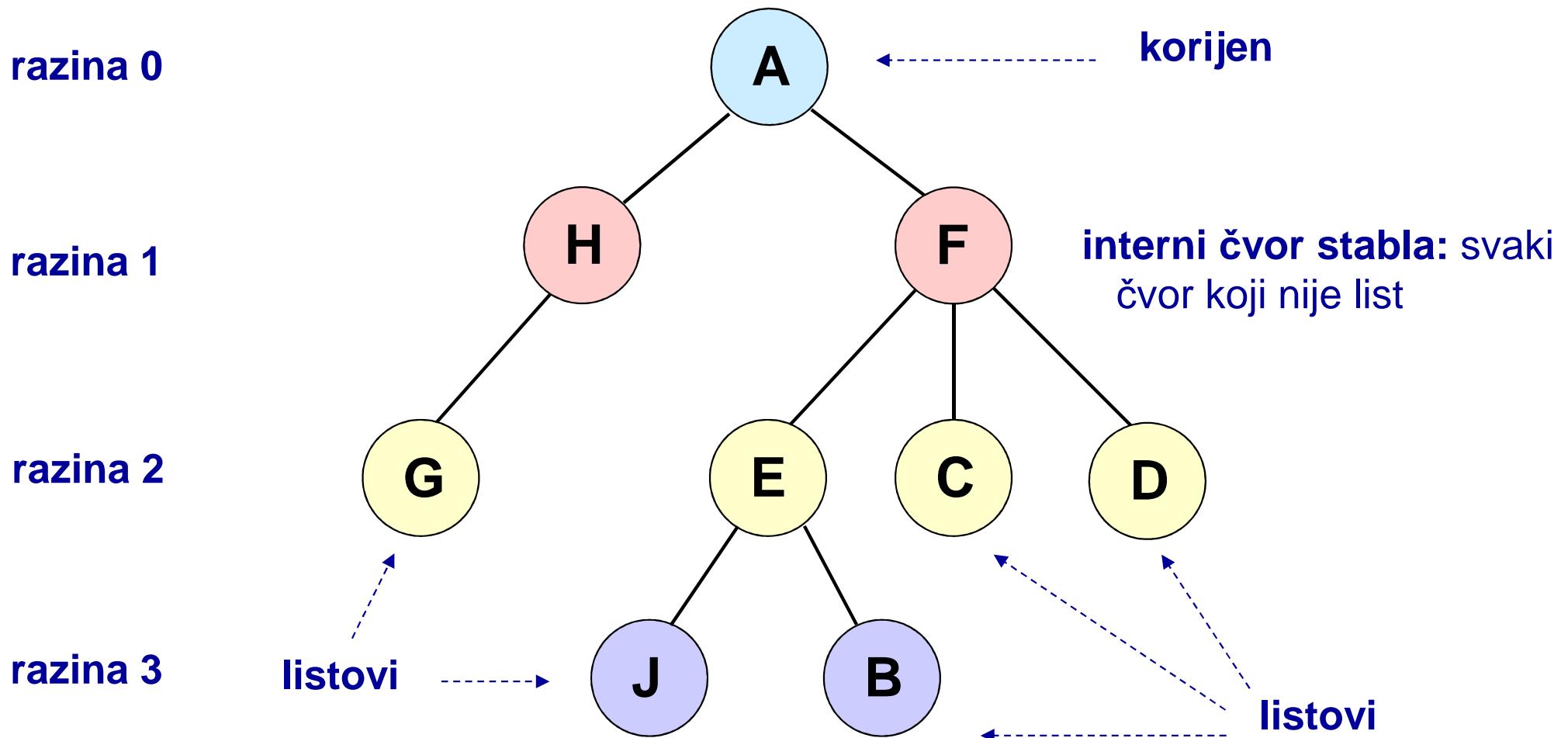
Stablo kao struktura podataka



Stablo kao struktura podataka



Stablo kao struktura podataka

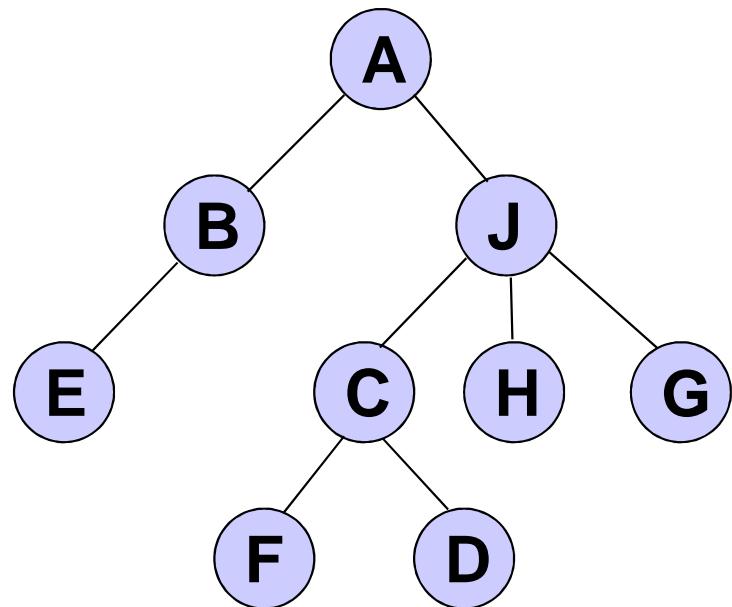


- **razina čvora (level)**: duljina puta od korijena do čvora
- **dubina stabla (depth)**: najveća duljina puta od korijena do lista
- **red stabla (order)**: najveći broj djece koje čvor može imati

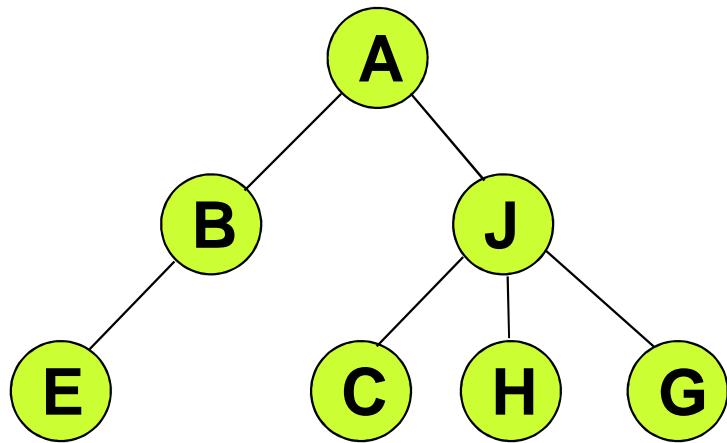
Stablo kao struktura podataka

Stablo je **balansirano (balanced)** ako je duljina puta od korijena do lista jednaka za svaki list u stablu

stablo nije balansirano

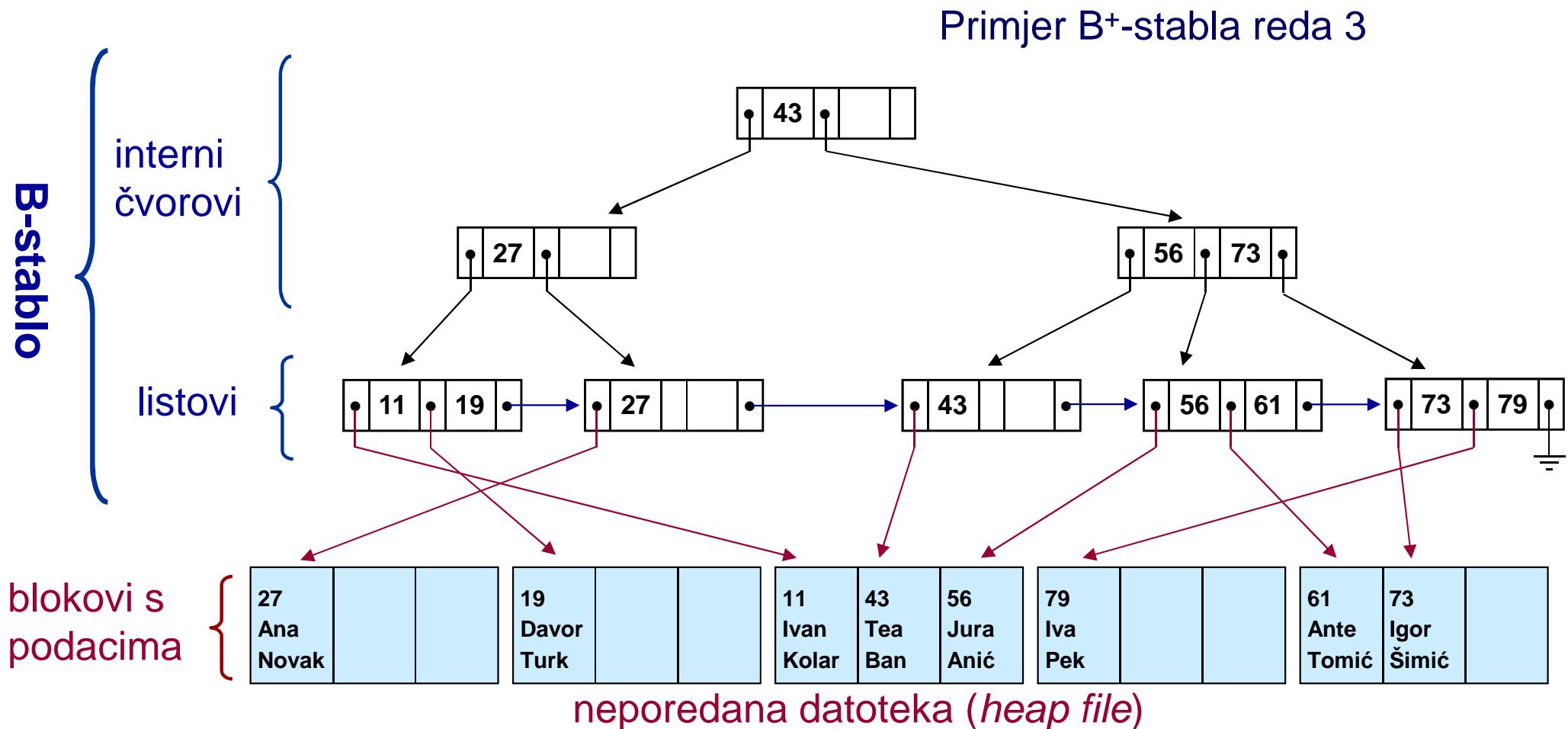


stablo je balansirano



Oznaka **B** u B-stablo znači "**balansirano**"!

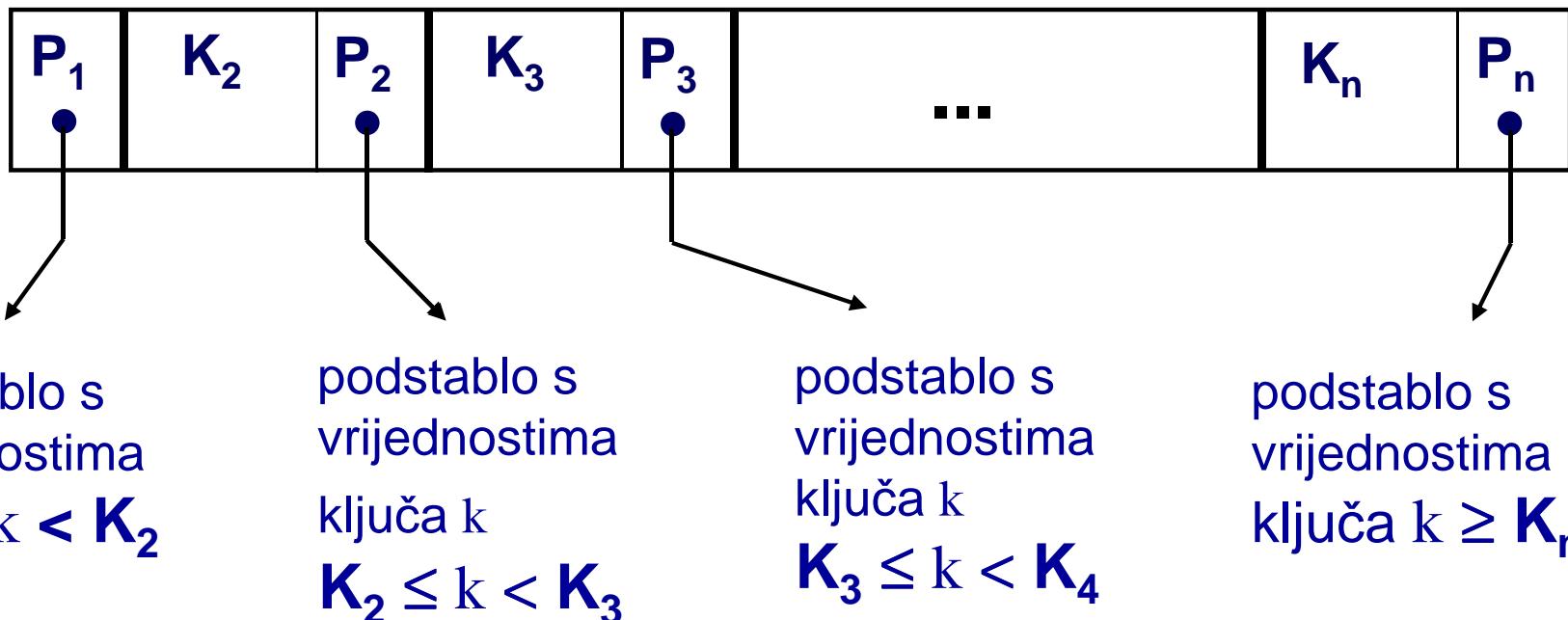
Struktura B⁺-stabla



- Shema: mbr, ime, prezime; B⁺-stablo je izgrađeno za atribut mbr
- Moguće metode pristupa podacima (za bilo koji ključ pretrage):
 - linearnim pretraživanjem (kao kod neporedane datoteke)
 - ako je ključ pretrage mbr, može se koristiti B-stablo

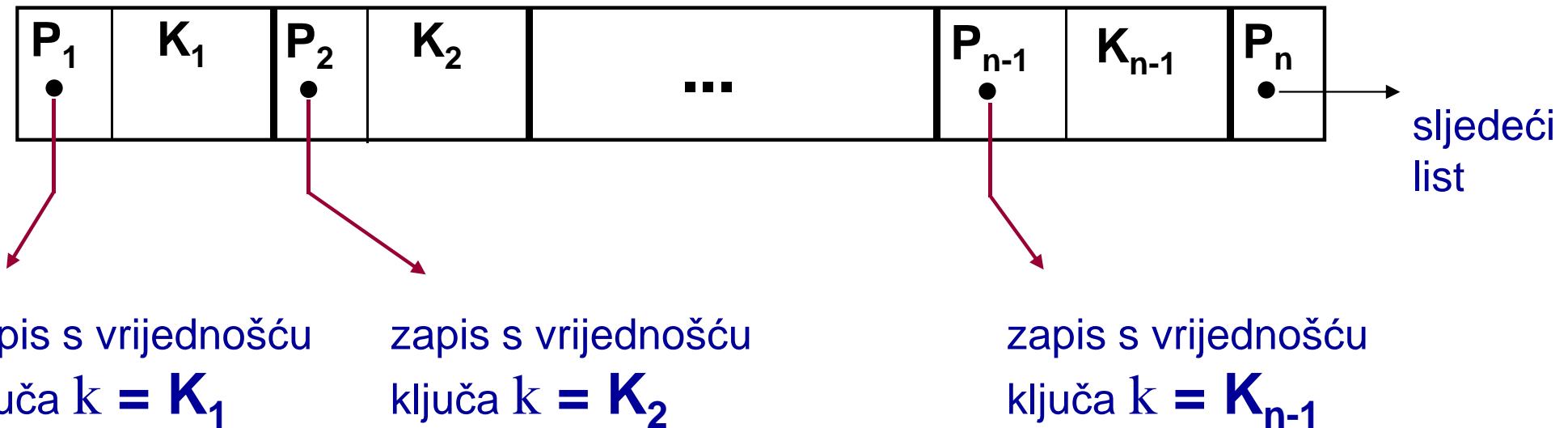
Struktura internog čvora B⁺-stabla

- U B⁺-stablu reda n , **interni čvor** sadrži:
 - najviše n kazaljki
 - najmanje $\lceil n/2 \rceil$ kazaljki $\rightarrow \lceil a \rceil$ je najmanji cijeli broj $\geq a$
 - ovo ograničenje ne vrijedi za korijen (najmanji broj kazaljki je 2)
 - uz p kazaljki u čvoru, broj pripadnih vrijednosti K_i u čvoru je $p-1$
 - K_i je vrijednost ključa



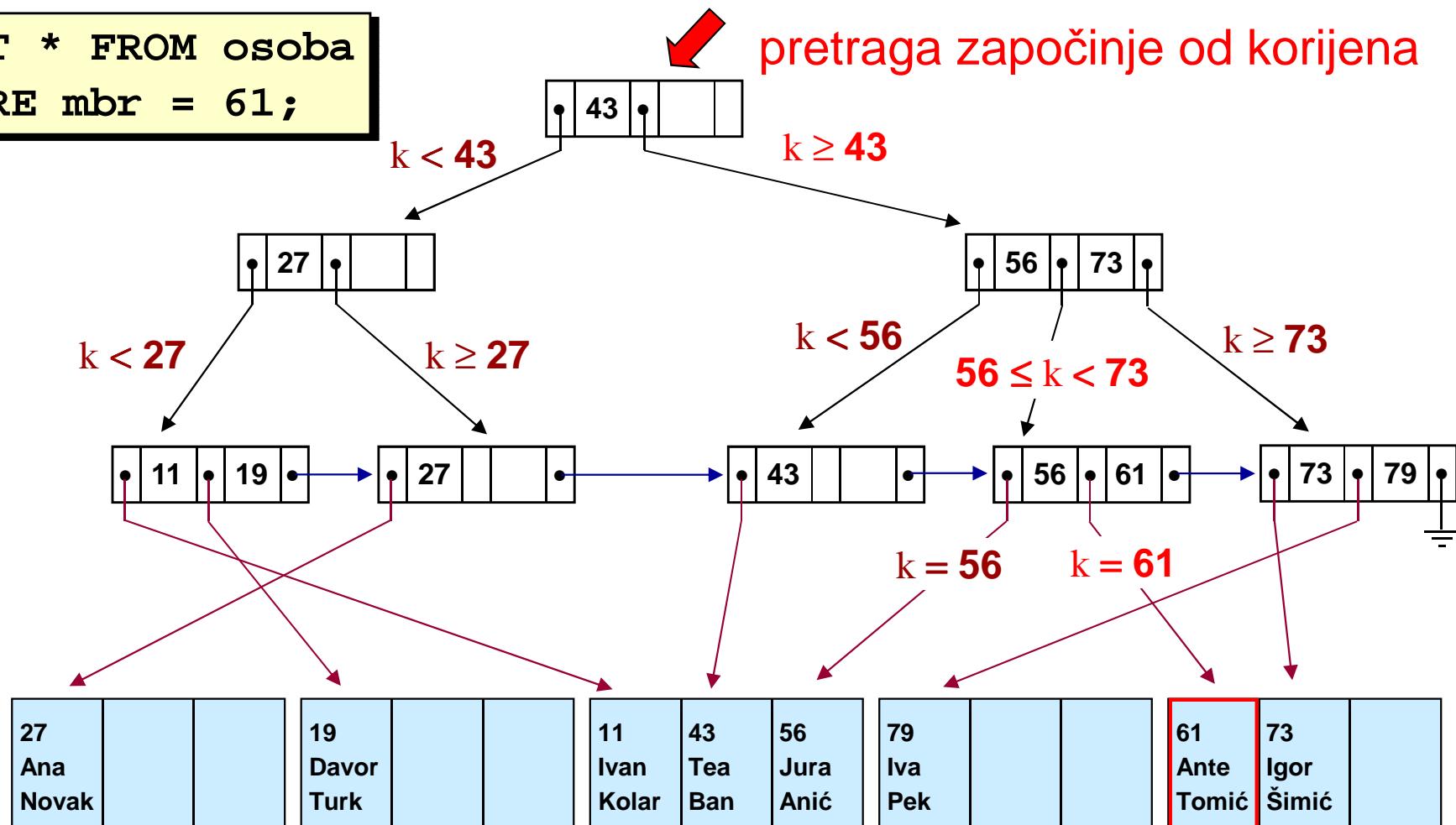
Struktura lista B⁺-stabla

- U B⁺-stablu reda n , list sadrži:
 - najviše $n-1$ vrijednosti K_i i pripadnih kazaljki na zapise
 - najmanje $\lceil (n-1)/2 \rceil$ vrijednosti K_i i pripadnih kazaljki na zapise
 - svi listovi sadrže kazaljku na sljedeći list
 - omogućava upite tipa od-do (prema zadanim granicama intervala)



Algoritam za pronalaženje zapisa putem B⁺-stabla

```
SELECT * FROM osoba  
WHERE mbr = 61;
```

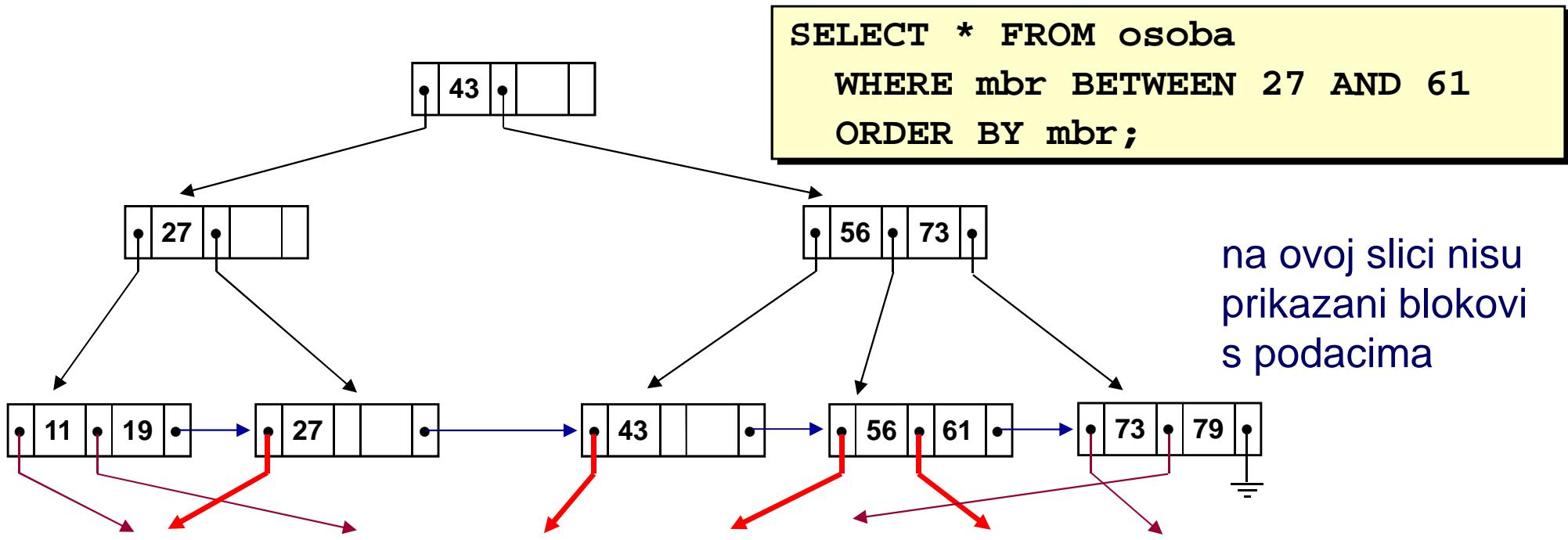


- slijediti odgovarajuću kazaljku do sljedeće razine
- postupak se ponavlja dok se ne dođe do lista u kojem će se naći kazaljka na zapis u bloku s podacima

Algoritam za pronalaženje zapisa putem B⁺-stabla

- algoritam za traženje zapisa s ključem vrijednosti k je rekurzivan
 - cilj je u svakom koraku rekurzije (pretraga i -te razine) pronaći čvor na nižoj, $(i+1)$ -voj razini, koji će voditi prema listu u kojem se nalazi ključ čija je vrijednost k
- traženje zapisa započinje od korijena (0-te razine)
- u čvoru i -te razine potrebno je pronaći najveću vrijednost ključa koja je manja ili jednaka traženoj vrijednosti k
 - za prvu kazaljku internog čvora nije navedena vrijednost ključa, pa ona "pokriva" sve vrijednosti ključeva manje od vrijednosti ključa K_2
- nakon pronalaženja odgovarajuće vrijednosti ključa, slijedi se pripadna kazaljka i time se obavlja pozicioniranje na $(i+1)$ -vu razinu
- postupak se ponavlja rekurzivno sve dok se ne dođe do lista. U njemu se mora nalaziti, ako postoji, ključ čija je vrijednost k , te pripadna kazaljka prema traženom zapisu (n -torki)

Dohvat podataka iz intervala, sortiranje



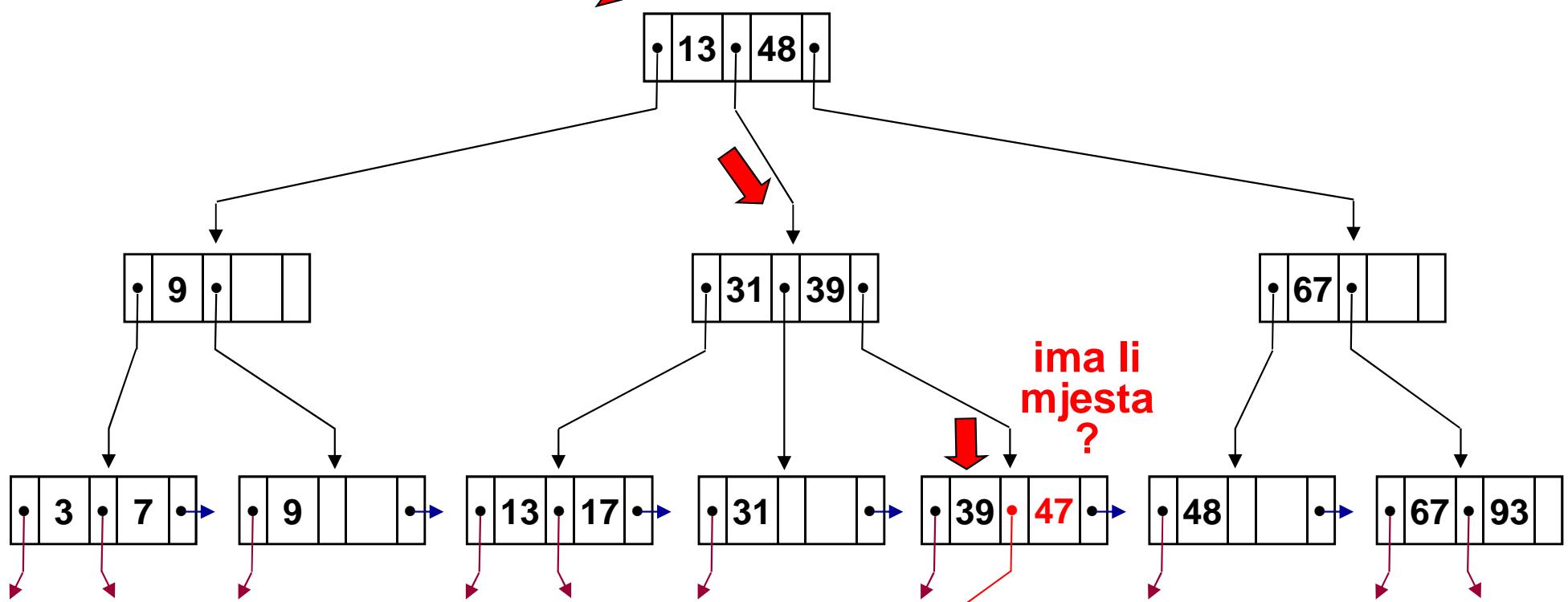
- u listu pronaći kazaljku na zapis s ključem **27**
- redom dohvaćati kazaljke i pripadne zapise dok se ne dođe do kazaljke na zapis s ključem **61**
 - taj postupak omogućavaju kazaljke među listovima
- dobiveni su svi traženi zapisi, pri tome su poredani prema mbr
- Ako se obavlja **SELECT * ... ORDER BY mbr DESC**
 - pronađene zapise jednostavno ispisati obrnutim redoslijedom

Algoritam za dodavanje zapisa u B⁺-stablu

- zapis (podaci) se upisuje u jedan od slobodnih blokova s podacima
- obavlja se procedura za pronalaženje lista L u koji pripada vrijednost ključa k
- ako čvor L nije popunjen, u čvor se dodaje zapis s ključem k i kazaljkom na pripadni zapis u datoteci, uz očuvanje poretku vrijednosti ključeva
 - nova vrijednost nikad nije prva u čvoru, osim u slučaju krajnjeg lijevog čvora
- ako je čvor L popunjen
 - stvara se novi čvor i zapisi među njima se podijele, pri čemu svakom od čvorova pripadne polovica zapisa
 - budući da je dodan novi čvor, u nadređeni čvor potrebno je dodati zapis s kazaljkom i najmanjom vrijednošću ključa u novom čvoru
 - za dodavanje novog zapisa u nadređeni čvor koristi se ista procedura kao za dodavanje zapisa u čvor na nižoj razini
 - postupak je rekurzivan i mora se obaviti za svaku nadređenu razinu (sve dok se ne dođe do korijena ili se na nekoj od razina nađe dovoljno mesta za upis vrijednosti ključa i kazaljke, bez dodavanja novih čvorova).
 - ako se dođe do korijena, može se desiti da u korijenu nema mesta za novi zapis. Tada se dodaje novi čvor i formira se novi korijen na višoj razini

Dodavanje zapisa u B⁺-stablu

dodavanje zapisa s ključem 47

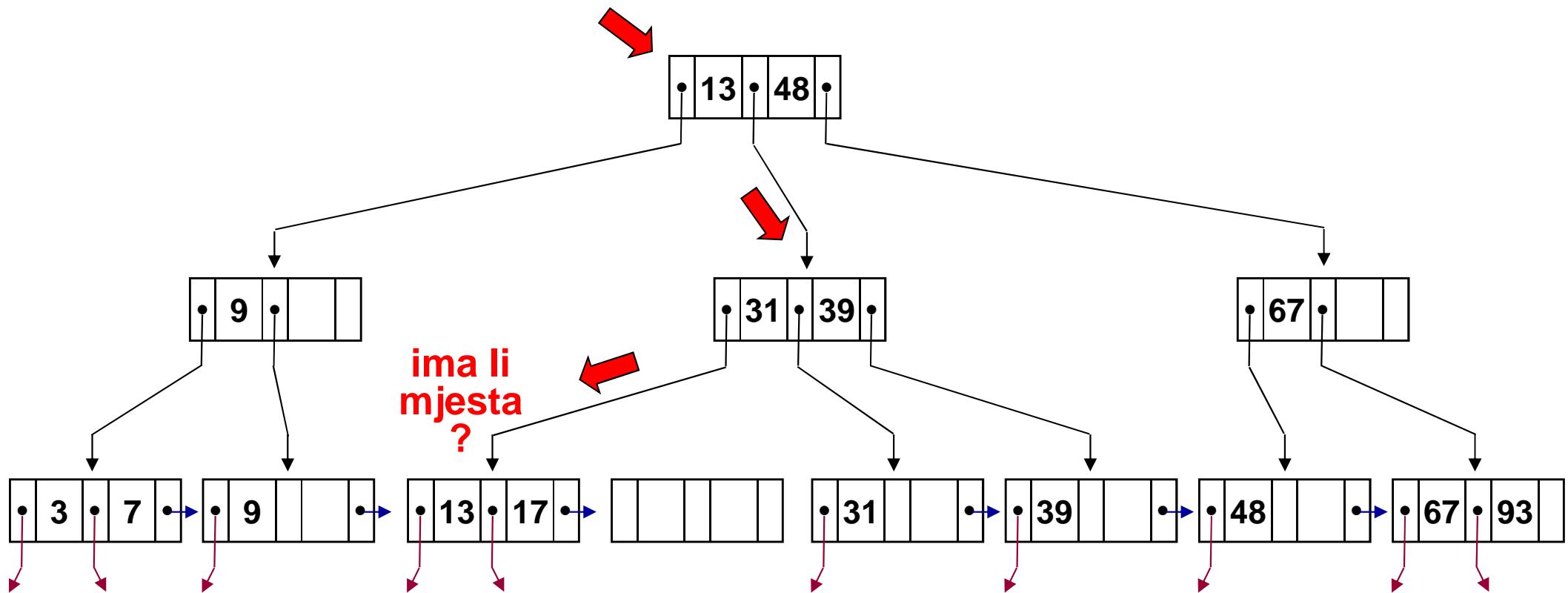


dodavanje zapisa u blok s podacima

9			93	13	39	67	34	7	47			3		17	48		31
...		
...		

Dodavanje zapisa u B⁺-stablu

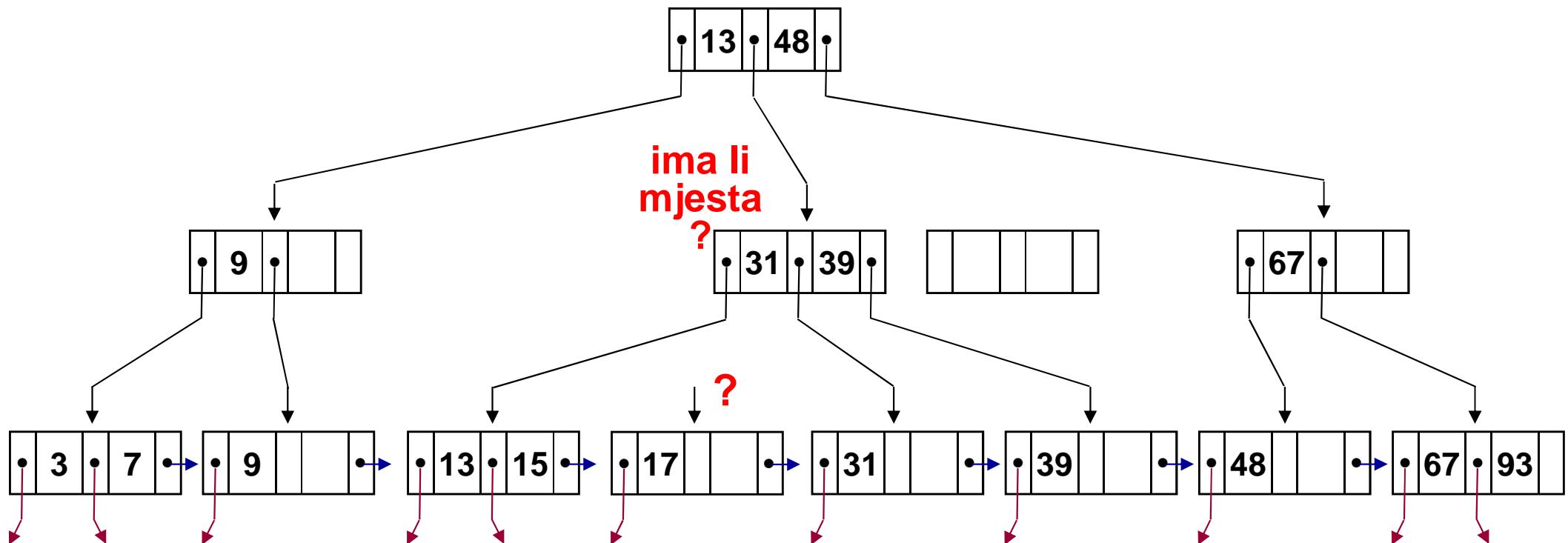
dodavanje zapisa s ključem 15



dodati novi čvor i podijeliti zapise
13, 15, 17 među čvorovima

Dodavanje zapisa u B⁺-stablu

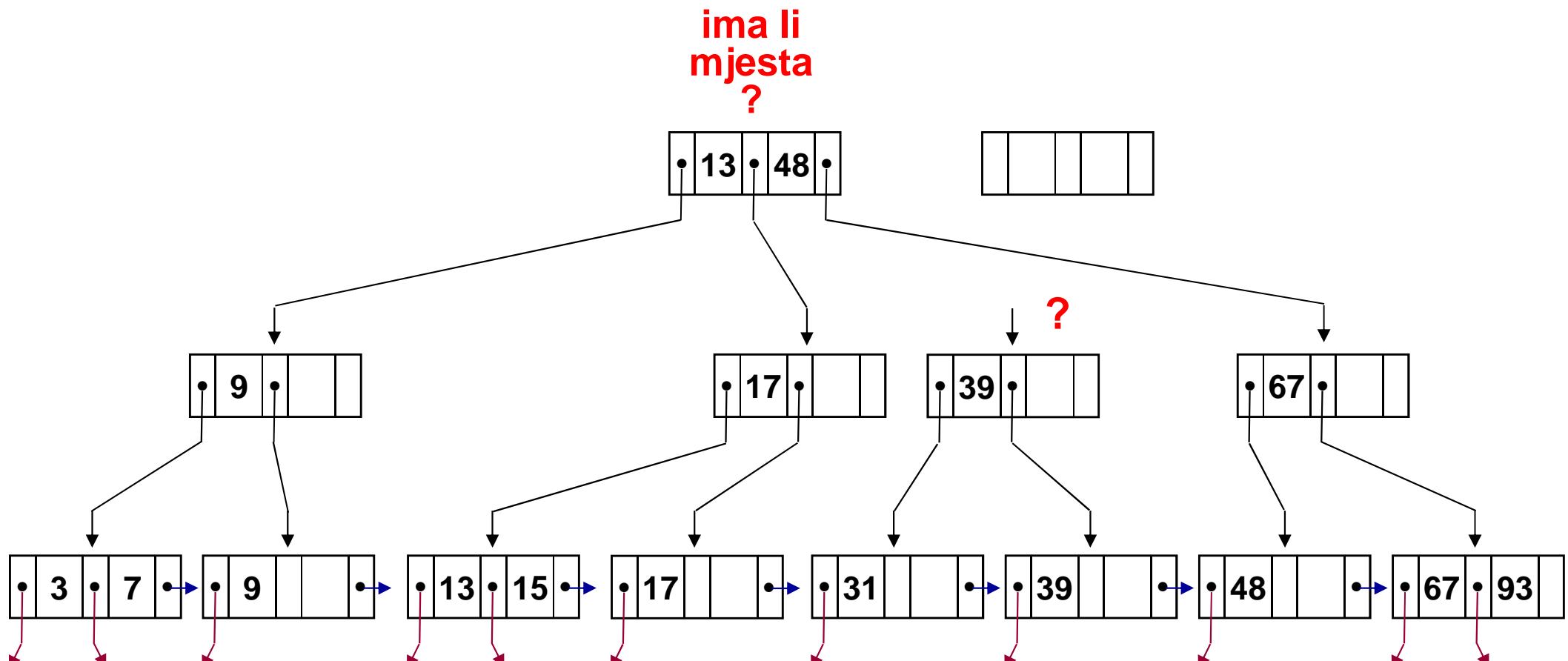
dodavanje zapisa s ključem 15



dodati novi čvor i podijeliti zapise
13, 17, 31, 39 među čvorovima

Dodavanje zapisa u B⁺-stablu

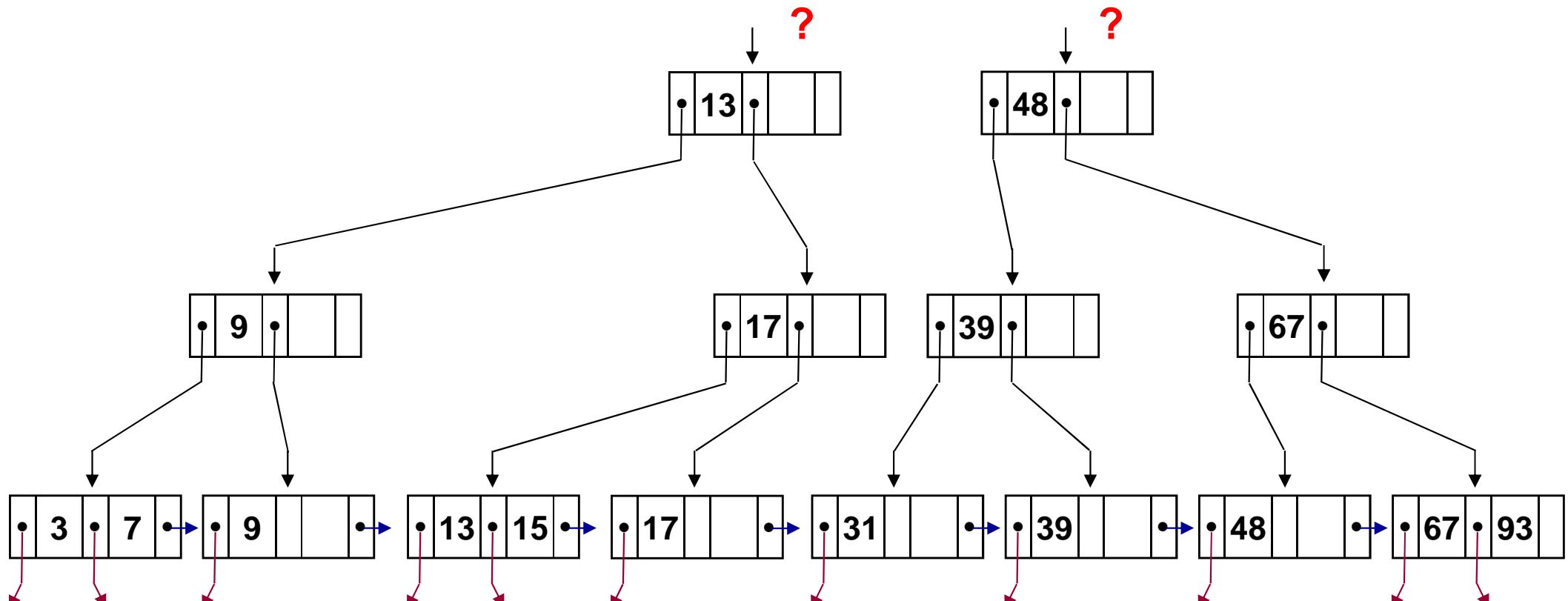
dodavanje zapisa s ključem 15



dodati novi čvor i podijeliti zapise
3, 13, 31, 48 među čvorovima

Dodavanje zapisa u B⁺-stablu

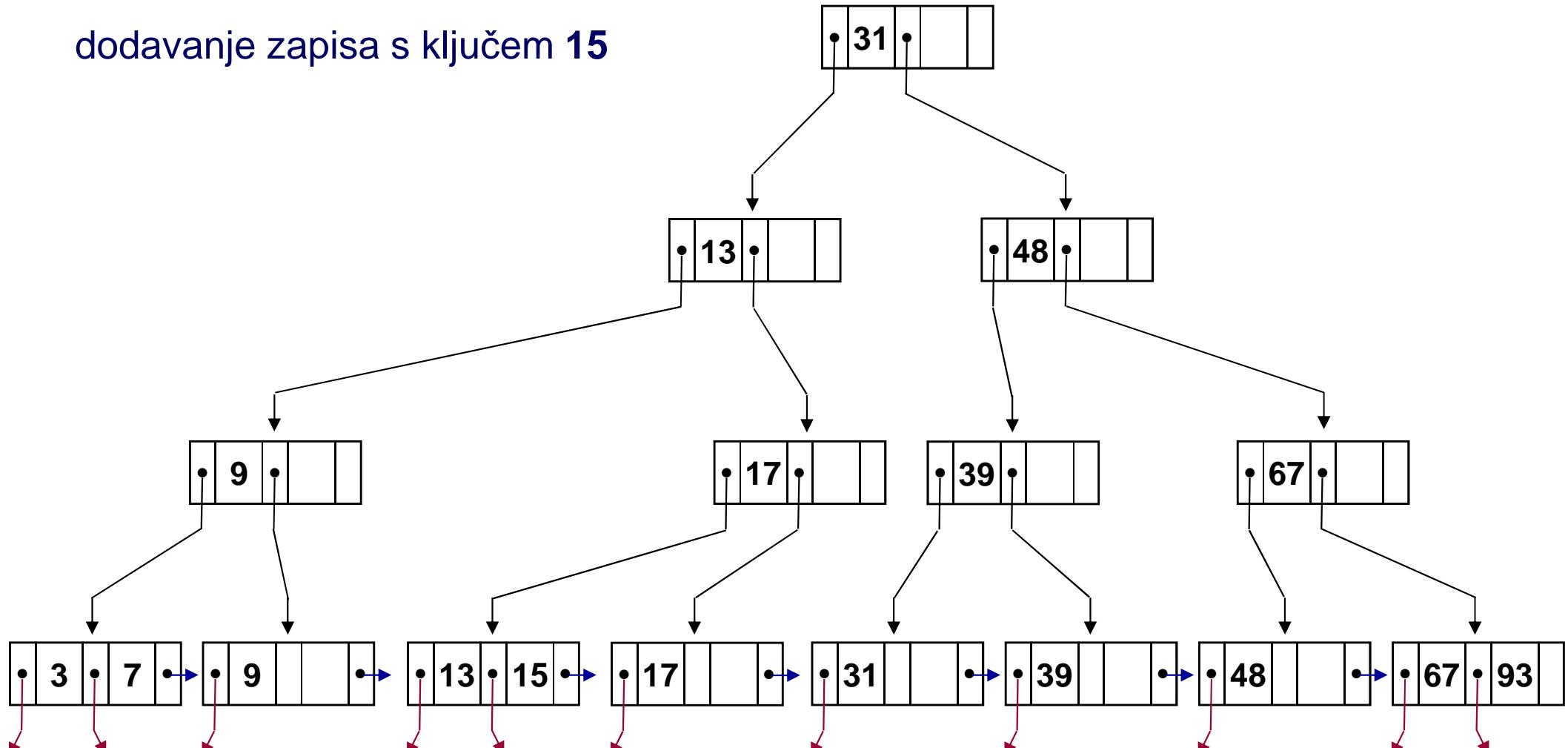
dodavanje zapisa s ključem 15



dodati novi korijen i upisati zapise 3, 31

Dodavanje zapisa u B⁺-stablu

dodavanje zapisa s ključem 15



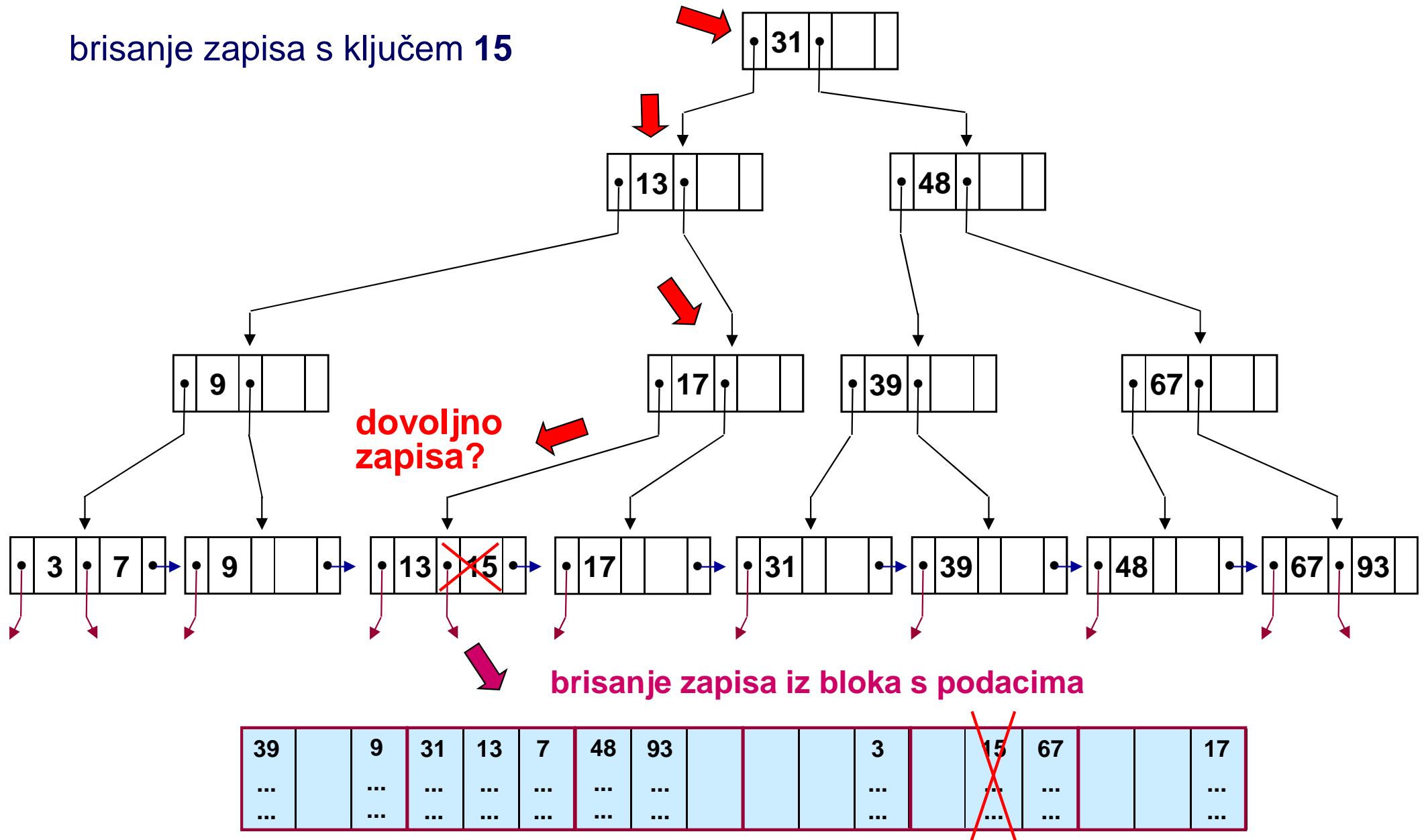
rezultat je balansirano stablo veće dubine

Algoritam za brisanje zapisa iz B⁺-stabla

- obavlja se procedura za pronalaženje lista L u kojem se nalazi ključ k
- zapis se briše iz bloka s podacima, a vrijednost ključa i kazaljka iz čvora L
- ako je potrebno (onda kada se iz lista obriše prvi zapis), mijenja se vrijednost ključa u nekom od nadređenih čvorova
- ako čvor L sadrži dovoljan broj zapisa, procedura brisanja je završena
- ako čvor L nakon brisanja nema dovoljan broj zapisa, traži se čvor-brat L_x koji se nalazi neposredno s lijeva ili s desna čvoru L i ima više od dovoljnog broja zapisa. Ako se takav ne pronađe, traži se bilo koji čvor-brat L_x koji se nalazi neposredno s lijeva ili s desna čvoru L
 - ako odabrani brat L_x ima više od dovoljnog broja zapisa, tada se zapisi podijele između čvorova L i L_x , uz zadržavanje poretku zapisa. U nadređenom čvoru mora se obaviti i izmjena vrijednosti ključa
 - inače (odabrani brat L_x ima upravo dovoljan broj zapisa), čvorovi L i L_x se spajaju u jedan čvor. U nadređenom čvoru briše se zapis za L i eventualno mijenja ključ čija kazaljka pokazuje na L_x . Brisanje zapisa u nadređenom čvoru svodi se na rekurzivno izvođenje procedure za brisanje. Ako se putem prema korijenu dođe u situaciju da treba spojiti jedina dva čvora-djeteta korijena, tada se oni spajaju, postaju novi korijen stabla, a stari se korijen briše

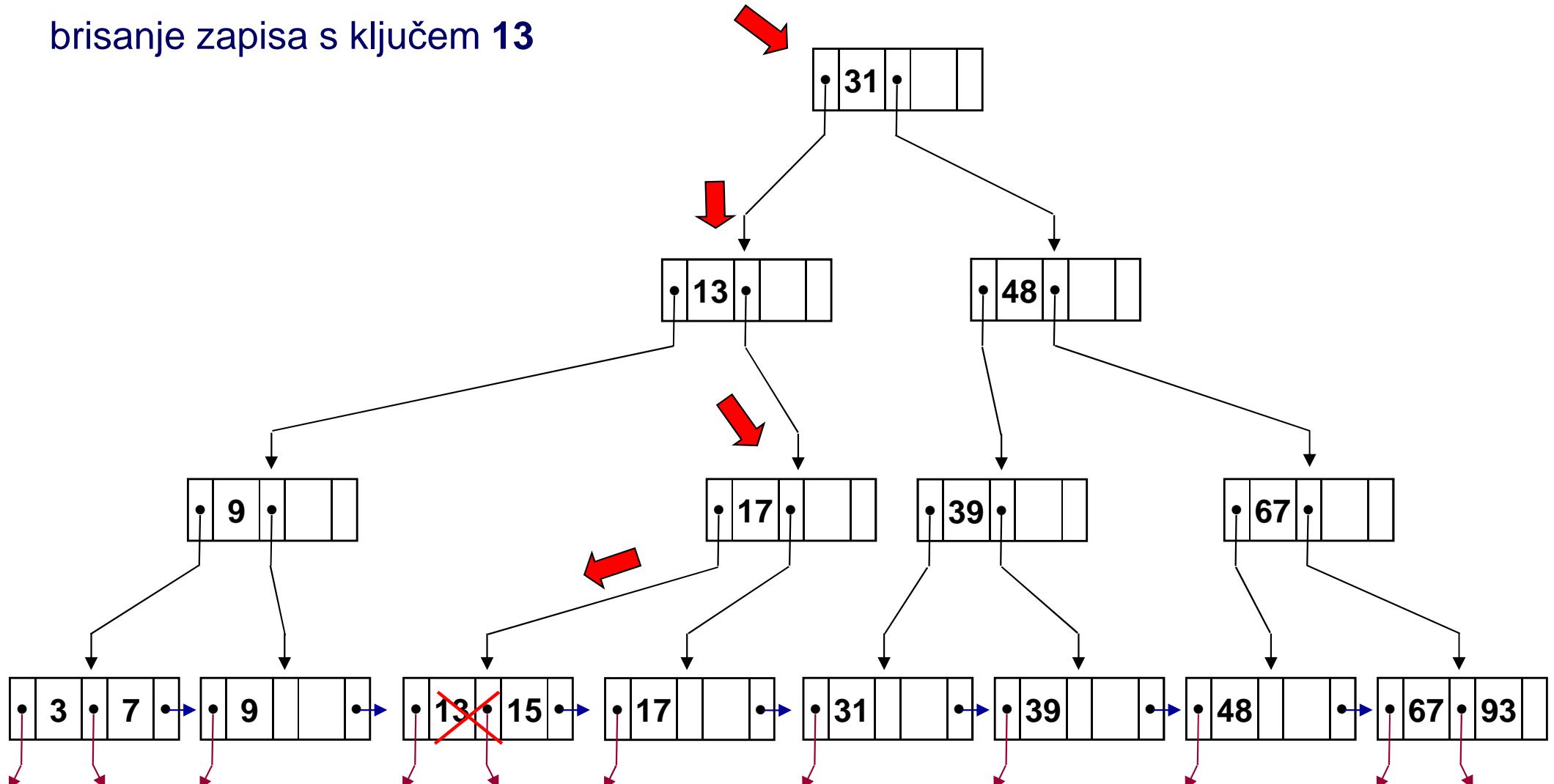
Brisanje zapisa iz B⁺-stabla

brisanje zapisa s ključem 15



Brisanje zapisa iz B⁺-stabla

brisanje zapisa s ključem 13

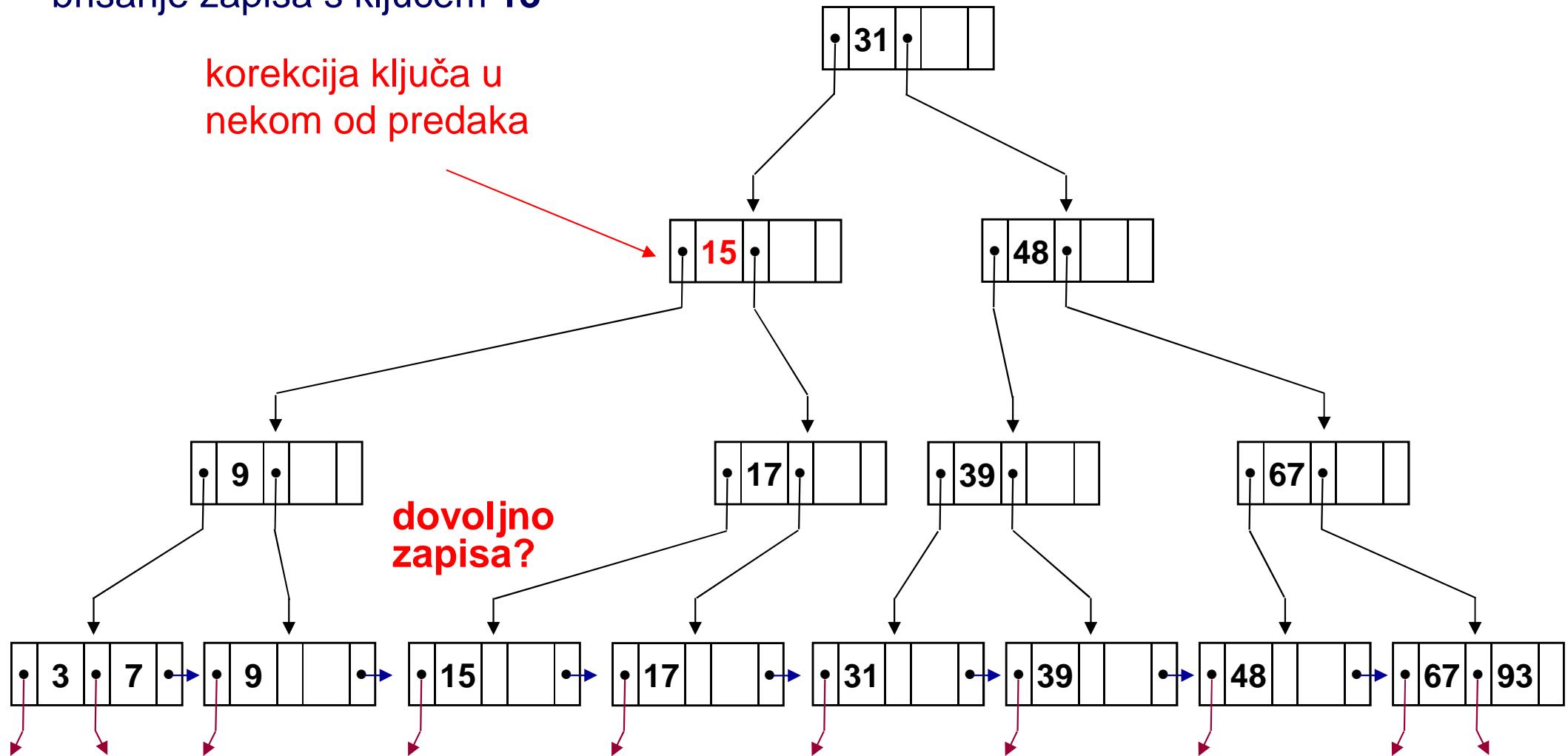


obaviti korekciju ključa u nekom od predaka

Brisanje zapisa iz B⁺-stabla

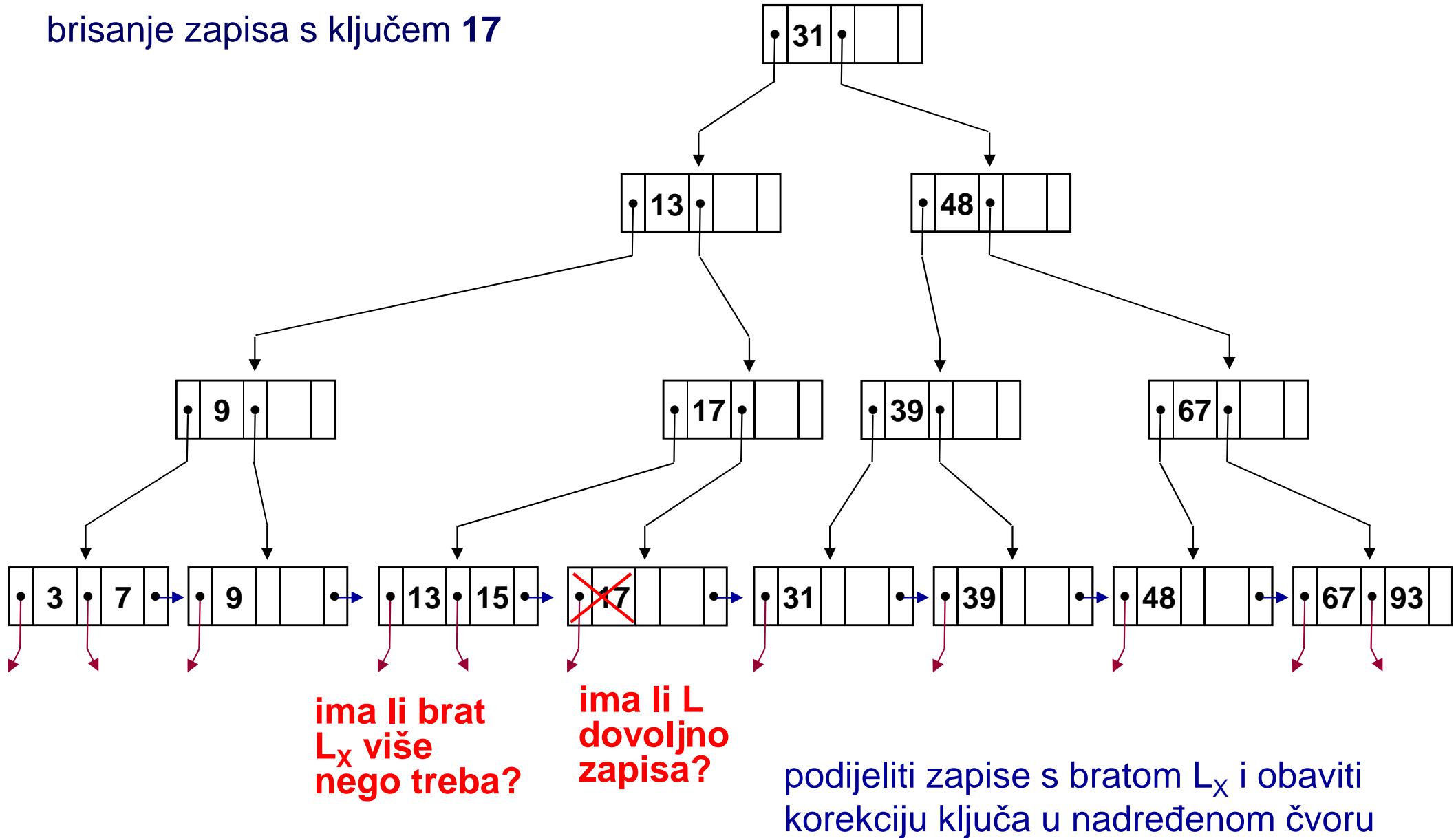
brisanje zapisa s ključem 13

korekcija ključa u
nekom od predaka



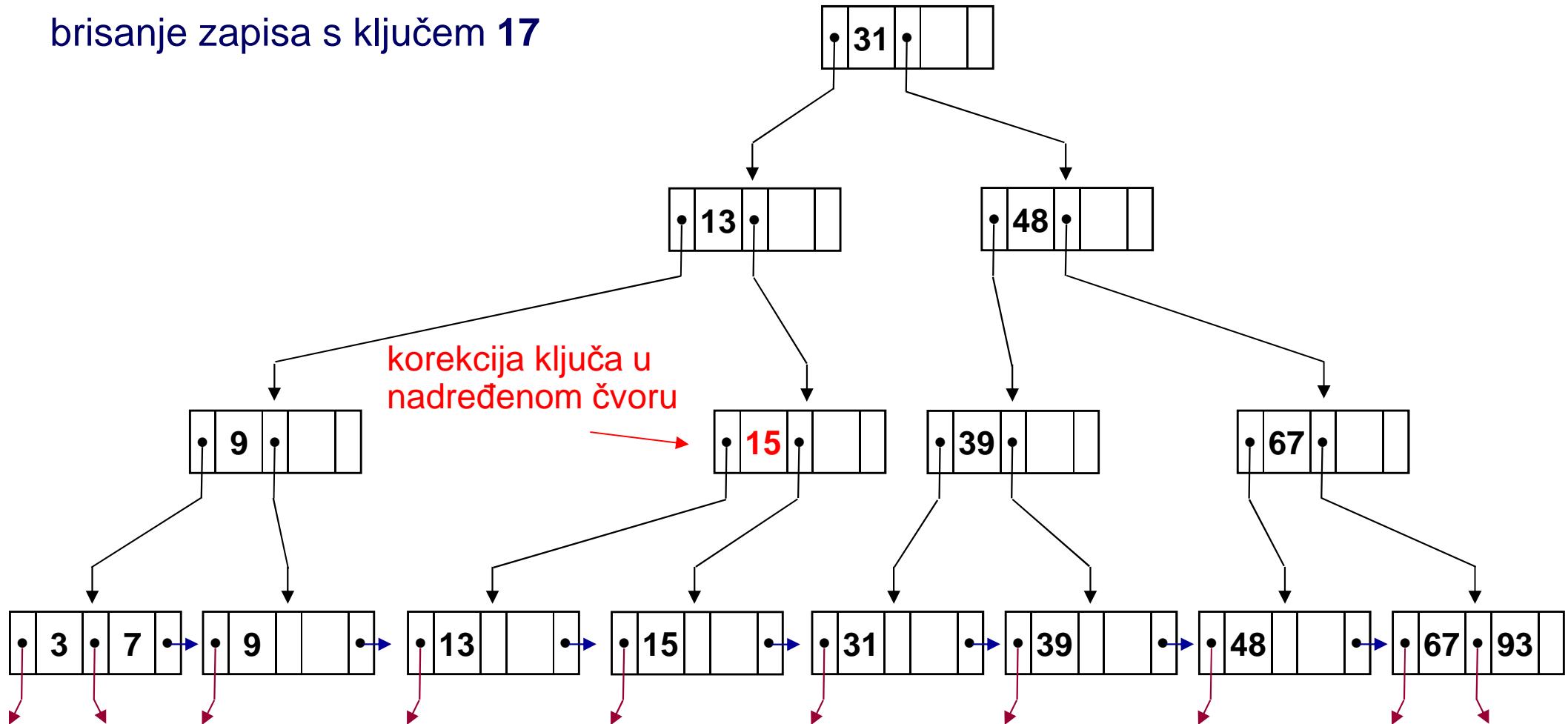
Brisanje zapisa iz B⁺-stabla

brisanje zapisa s ključem 17



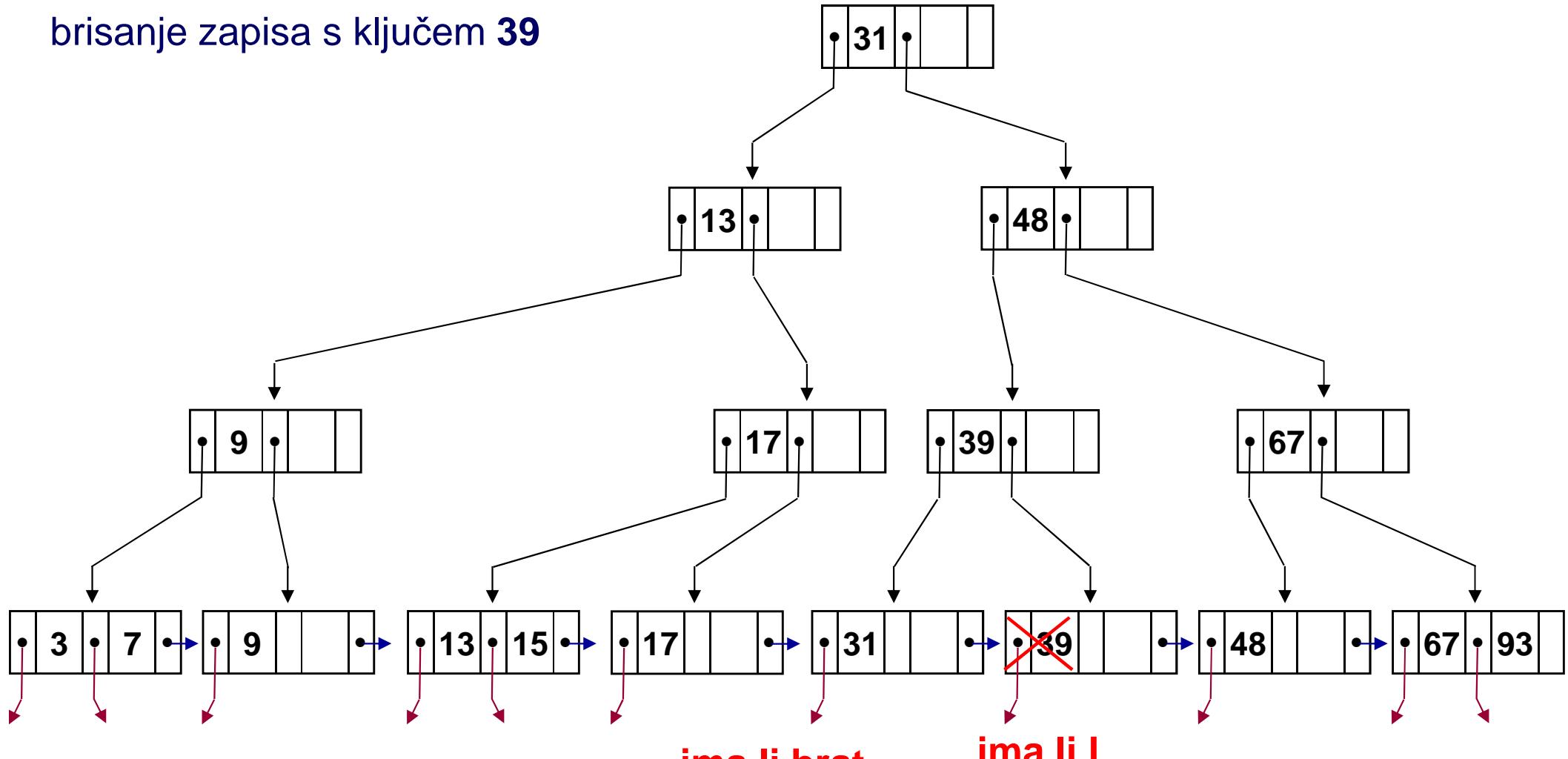
Brisanje zapisa iz B⁺-stabla

brisanje zapisa s ključem 17



Brisanje zapisa iz B⁺-stabla

brisanje zapisa s ključem 39



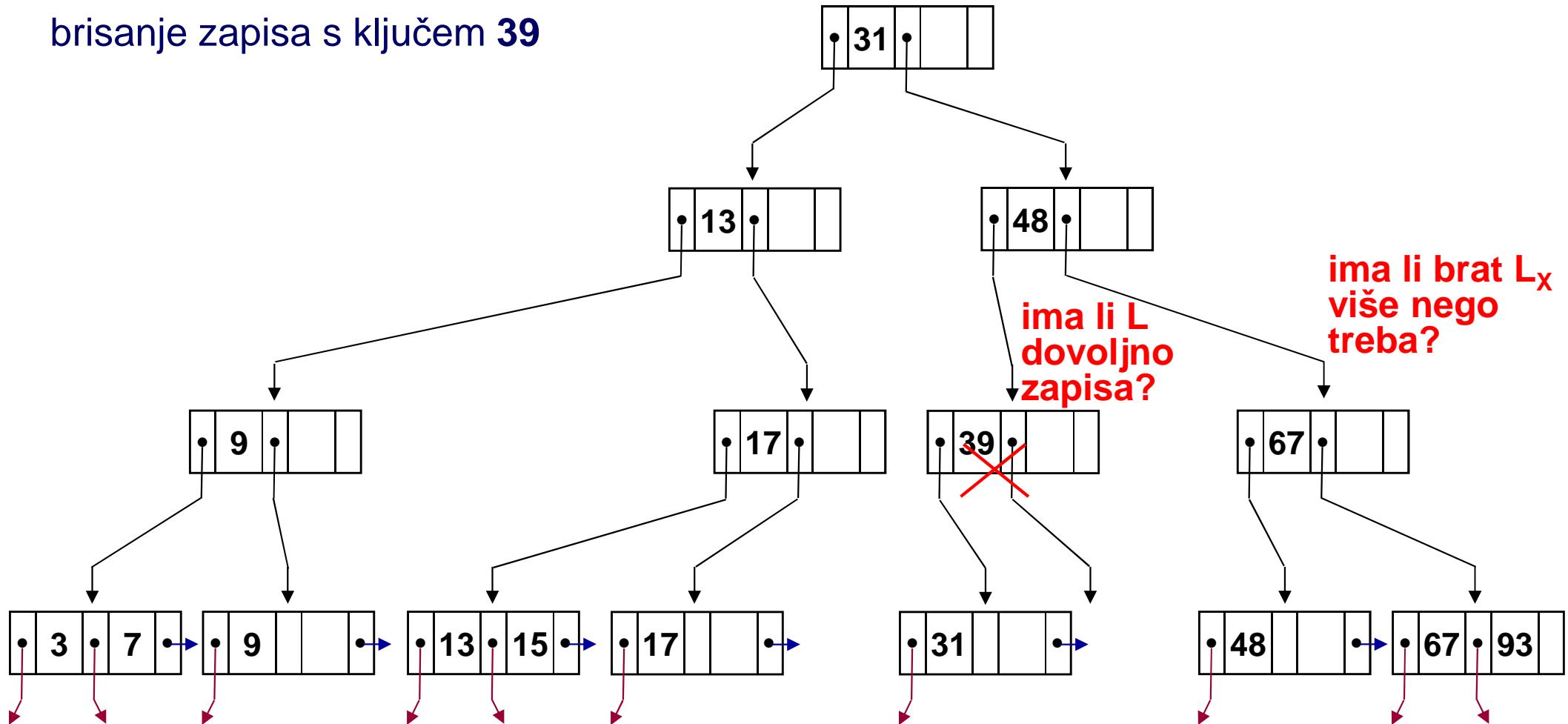
spojiti L i L_x . Obrisati kazaljku na L iz nadređenog čvora.

ima li brat
 L_x više
nego treba?

ima li L
dovoljno
zapisa?

Brisanje zapisa iz B⁺-stabla

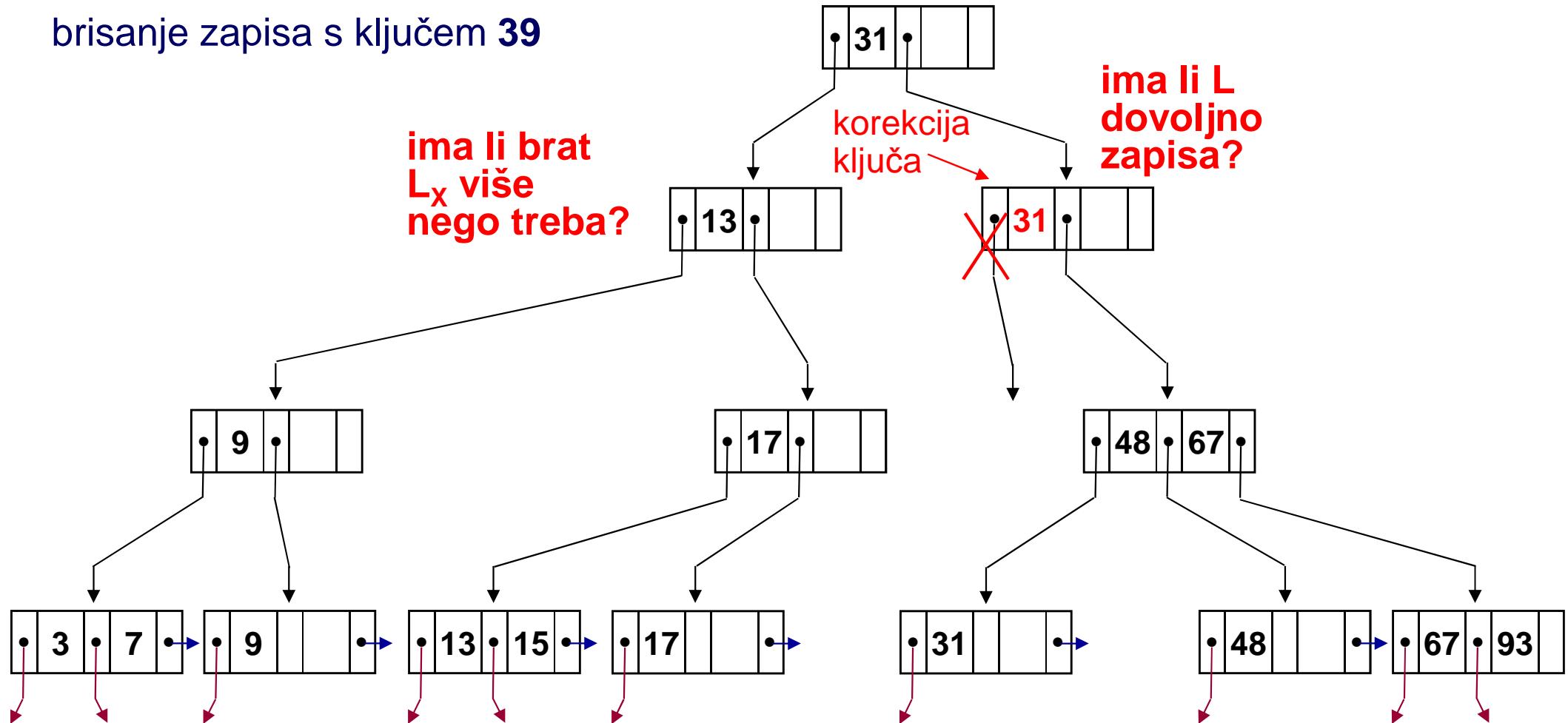
brisanje zapisa s ključem 39



spojiti L i L_x . Obrisati kazaljku na L iz nadređenog čvora i promijeniti vrijednost ključa koja pokazuje na L_x .

Brisanje zapisa iz B⁺-stabla

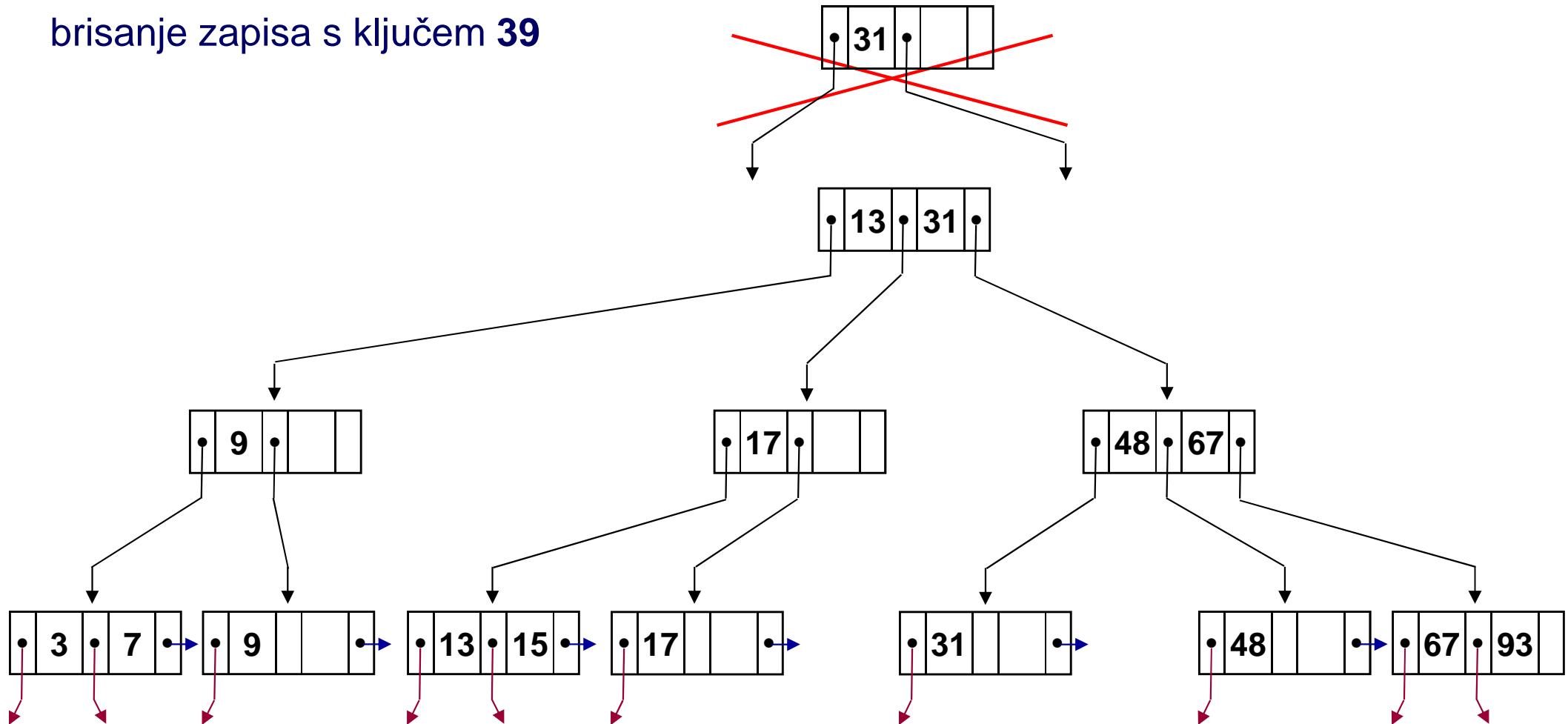
brisanje zapisa s ključem 39



spojiti L i L_x. Spajanjem L i L_x nastaje novi korijen. Obrisati stari korijen.

Brisanje zapisa iz B⁺-stabla

brisanje zapisa s ključem 39



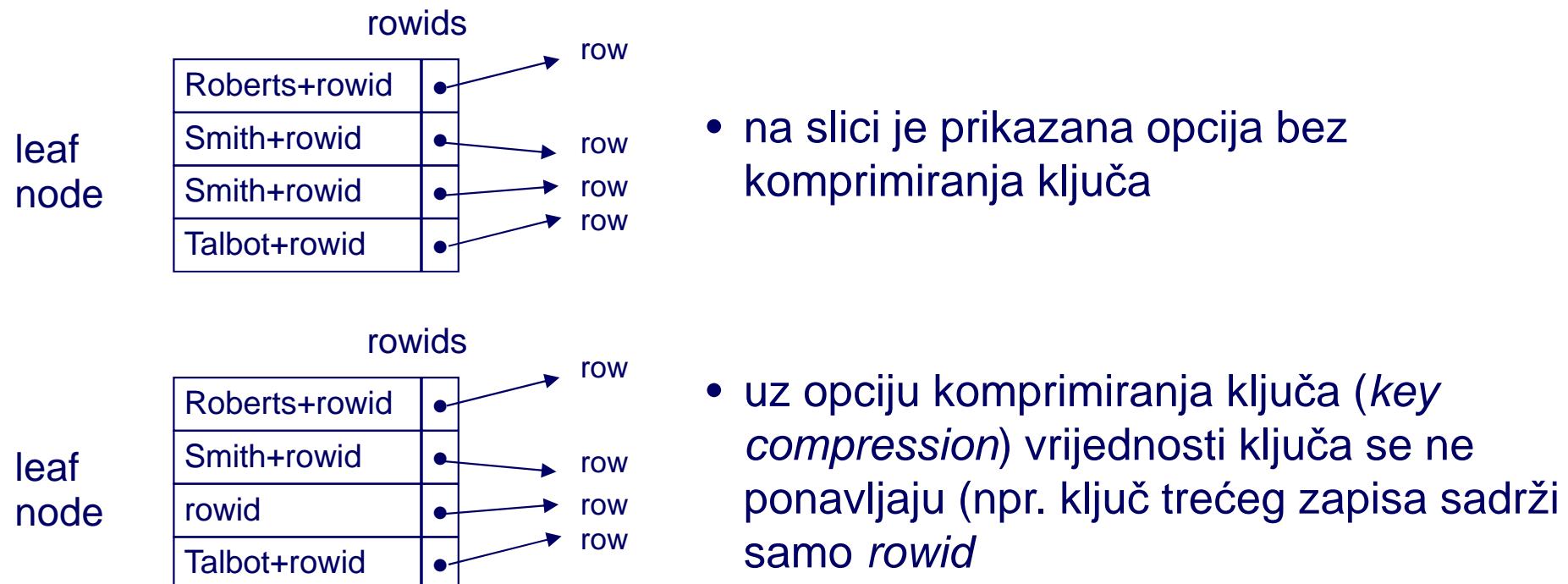
rezultat je balansirano stablo manje dubine

Zapis podataka s jednakim vrijednostima ključa

- *non-unique index*
- opisani algoritmi za B-stablo se moraju modificirati. Problemu se pristupa na različite načine. Npr:

Oracle 11g

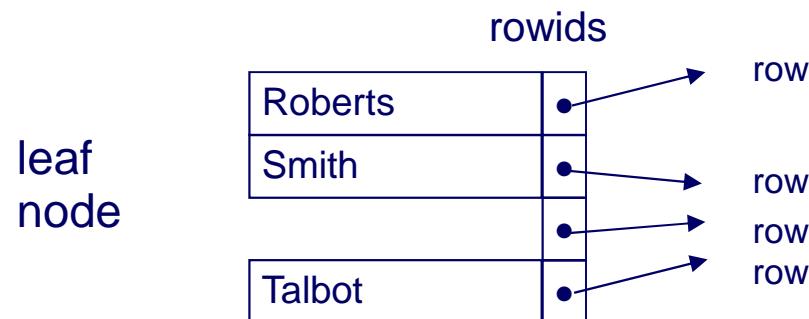
- u listovima se pokazivač na zapis podatka (zapravo *ROWID-row identifier*) ulančava s ključem (postaje dio ključa).



Zapisi podataka s jednakim vrijednostima ključa

IBM IDS

- u listovima se uz jedan ključ može pohraniti više pokazivača na zapise podataka



Efikasnost operacije pretrage u B⁺-stablu

- pretpostavka: stablo reda **n** sadrži kazaljke na **m** zapisa podataka
- **n** se odabire tako da se sadržaj čvora može smjestiti u jedan fizički blok
⇒ za dohvat **jednog** čvora potrebna je **jedna** U/I operacija
- broj U/I operacija u stablu pri traženju zapisa ovisi o broju razina u stablu jer se pri dohvatu zapisa mora obaviti po jedna U/I operacija za svaki čvor B-stabla na putu od korijena do lista
- B-stablo ima najveći broj razina onda kada su čvorovi najmanje popunjeni
⇒ moguće je odrediti koliko će U/I operacija biti potrebno obaviti u najlošijem slučaju

Efikasnost operacije pretrage u B⁺-stablu

- **Primjer:** za broj n-torki $m = 1\ 000\ 000$, za red stabla $n = 70$, ukupni broj razina (uključujući i razinu korijena) u najlošijem slučaju je 4:

1.  točno 1 čvor, najmanje 2 kazaljke
2.  najmanje 2 čvora, najmanje 70 kazaljki
3.  ...  najmanje 70 čvorova, najmanje 2 450 kazaljki
4.  ...  najmanje 2 450 čvorova, najmanje 85 750 kazaljki
5.  ...  najmanje 85 750 čvorova, najmanje 3 001 250 kazaljki

- B⁺-stablo koje bi imalo ukupno 5 razina, moralo bi imati **najmanje** 3 001 250 kazaljki na zapise. To znači da B-stablo reda 70 čije kazaljke u listovima pokazuju na 1 000 000 n-torki smije imati najviše 4 razine.
⇒ Za dohvatzanja prema vrijednosti ključa potrebno je najviše 5 U/I operacija (4 U/I operacije za dohvatzanje lista u kojem se nalazi kazaljka na zapis + 1 U/I operacija za dohvatzanje bloka s podacima)
- **često se korijen i njegova djeca zadržavaju u međuspremniku SUBP-a**

Efikasnost operacije pretrage u B⁺-stablu

- broj U/I operacija u stablu pri traženju zapisa ovisi o broju razina u stablu
- pretpostavka: stablo reda **n** sadrži kazaljke na **m** zapisa podataka
- stablo će imati najveći broj razina ako su čvorovi najmanje popunjeni
 - najmanja popunjenošć korijena je **2**
 - najmanja popunjenošć internog čvora: $\lceil n / 2 \rceil$
 - najmanja popunjenošć lista: $\lceil (n - 1) / 2 \rceil \approx \lceil n / 2 \rceil$, za dovoljno veliki **n**
- u korijenu (1. razina) ima 1 čvor i najmanje **2** kazaljke
- na 2. razini ima najmanje 2 čvora i zato najmanje $2 \cdot \lceil n / 2 \rceil$ kazaljki
- u čvorovima 3. razine ima najmanje $2 \cdot \lceil n / 2 \rceil \cdot \lceil n / 2 \rceil$ kazaljki
- u čvorovima *i*-te razine ima najmanje $2 \cdot \lceil n / 2 \rceil^{i-1}$ kazaljki
- za broj zapisa u podatkovnim blokovima stabla koje ima **d** razina vrijedi:
 - $m \geq 2 \cdot \lceil n / 2 \rceil^{d-1}$
- iz toga slijedi
 - $d \leq \log_{\lceil n/2 \rceil} (m / 2) + 1$
- **Primjer:** za $m = 1\ 000\ 000$ zapisa, $n = 70$, ukupni broj razina (uključujući i razinu korijena) u najgorem slučaju je 4

Popunjenošt B⁺-stabla u stacionarnom stanju

- analitički je dokazano i simulacijama potvrđeno:
 - ako se nad B⁺-stabлом obavi veliki broj operacija unosa i brisanja (slučajno odabranih vrijednosti ključeva), tada će popunjenošt stabla biti $\approx \ln 2 \approx 69\%$
- zadatak za vježbu: grubo procijeniti kolika je (u najlošijem slučaju) popunjenošt B⁺-stabla nakon što se u prazno stablo unese velik broj zapisa poredanih prema ključu

Efikasnost ostalih operacija u B⁺-stablu

- na efikasnost operacija unosa i brisanja utječe trošak reorganizacije B-stabla
- u rijetkim slučajevima dijeljenje ili spajanje čvorova se propagira sve do korijena stabla
- u B⁺-stablima koja imaju razumno velik red stabla (npr. 10 ili više) takvi su slučajevi rijetki
 - npr. za red stabla 101: koliko puta se list može podijeliti (u najgorem slučaju) ako se u stablo unese 100 zapisa
 - najgori slučaj: list je bio pun, 100 zapisa
 - 1. uneseni zapis dijeli list na dva: list od 50 i list od 51 zapisa
 - 2. - 50. uneseni zapis neće uzrokovati dijeljenje lista
 - 51. zapis uzrokuje dijeljenje lista: listovi s 50 zapisa i 51 zapisom
 - 52. - 100. uneseni zapis neće uzrokovati dijeljenje lista
 - 100 unesenih zapisa uzrokovalo je samo dvije reorganizacije na razini listova (eventualno, ali vrlo rijetko, reorganizacija će biti potrebna i na višim razinama)

Efikasnost ostalih operacija u B⁺-stablu

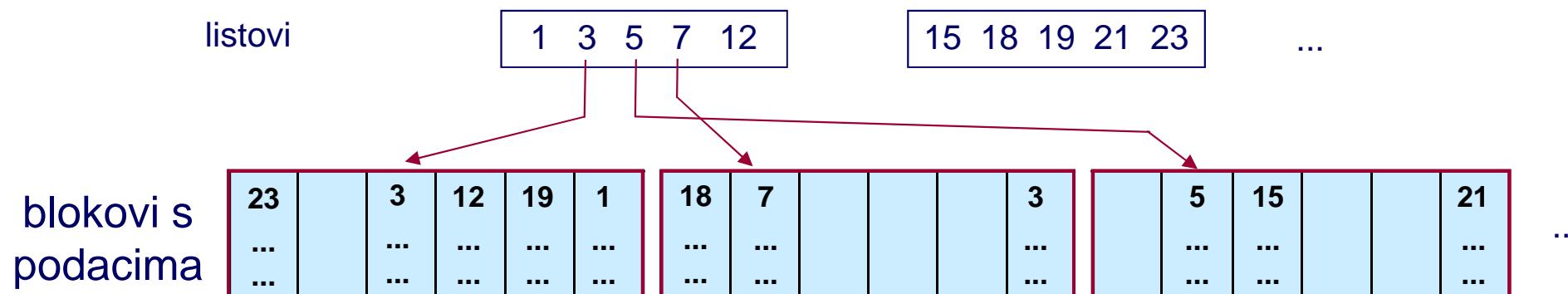
- može se zaključiti da će se za operacije unosa ili brisanja najčešće koristiti samo dvije U/I operacije više nego kod pretrage
 - **pretraga:**
 - *lookup* u *heap* datoteku - 1 U/I operacija
 - **izmjena ili brisanje:**
 - *lookup* u *heap* datoteku (1 U/I operacija)
 - *write data block* (1 U/I operacija)
 - *write leaf block* (1 U/I operacija)
- originalni algoritam za B⁺-stablo pri brisanju obavlja reorganizaciju onda kada broj zapisa u čvoru padne ispod polovice kapaciteta čvora
 - strategija *merge-at-half*
- većina današnjih sustava reorganizaciju pri brisanju provodi tek onda kada u čvoru ne preostane niti jedan zapis
 - strategija *merge-at-empty*
 - značajno niža frekvencija reorganizacija uz neznatno lošije iskorištenje prostora

Trošak korištenja indeksa

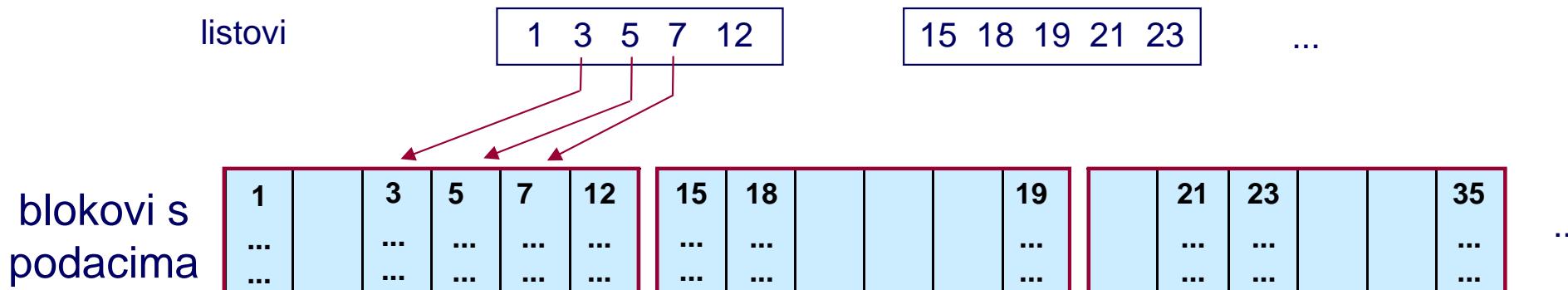
- Izgradnja i održavanje indeksa može biti skupo (prostor za pohranu, CPU, U/I aktivnosti). Administrator sustava mora osigurati da dobici pri korištenju indeksa premašuju gubitke koji nastaju zbog održavanja indeksa
- gruba procjena: obavljanje DML operacije INSERT, UPDATE ili DELETE zbog koje je potrebno obaviti održavanje jednog indeksa rezultira trostruko većim utroškom resursa. Npr, INSERT ili DELETE operacija nad relacijom za koju su izgrađena tri indeksa je desetak puta skuplja (CPU, U/I) od iste operacije nad relacijom za koju nije izgrađen niti jedan indeks
- posebnost UPDATE operacije: primjer
 - relacija $r(A, B)$, za atribut A je kreiran indeks
 - **UPDATE r SET B=x WHERE A=y**
 - nema troška održavanja indeksa, dobar odabir
 - **UPDATE r SET A=x WHERE B=y**
 - indeks ne donosi dobitak, postoji samo trošak održavanja indeksa, loš odabir

Indeks s poredanim zapisima podataka (*Cluster index*)

- dohvati podataka u intervalu vrijednosti [3, 7]



- *Cluster index*: podaci (u podatkovnim blokovima) su poredani prema ključu



- uspoređiti broj U/I operacija
- za jednu relaciju moguć je najviše jedan indeks s poredanim zapisima podataka

Indeks s poredanim zapisima podataka (*Cluster index*)

IBM IDS

- naredba kreira B⁺-stablo i sortira zapise s podacima. Nakon kreiranja indeksa zapisi su fizički poredani prema ključu

```
CREATE CLUSTER INDEX idxName ON tablename (colname, ...);
```

- obavljanje operacija unosa i izmjena može narušiti poredak zapisa s podacima te se performanse tijekom vremena mogu smanjiti
 - izmijenjeni podatak zapisuje se na istu poziciju (ako se promijenila vrijednost ključa, zapis više nije na odgovorajućoj fizičkoj poziciji)
 - unesen podatak zapisuje se na kraj datoteke s podacima
- reorganizacija po potrebi

```
ALTER INDEX idxName TO CLUSTER;
```

- zapisi s podacima se kopiraju u novi prostor relacije
 - ujedno se eliminira fragmentiranost prostora relacije

Indeks s poredanim zapisima podataka (*Cluster index*)

Oracle 11g

- ***single-table cluster***
- slično kao u sustavu IBM IDS
- novi ili izmijenjeni podaci (koji bi narušili poredak prema ključu) pohranjuju se u preljevno (*overflow*) područje, slično kao kod raspršenog adresiranja
 - veliki broj prelivenih zapisa umanjuje performanse
 - po potrebi se obavlja reorganizacija

Indeks s poredanim zapisima podataka (*Cluster index*)

Oracle 11g

- ***multi-table cluster***

- relacije čiji se podaci često koriste zajedno (npr. relacije nad kojima se često obavlja prirodno spajanje), pohranjuju se u blokove tako da se logički povezani zapisi dviju relacija smještaju u zajedničke blokove

non-clustered

blokovi sa
zapisima o
studentima

s5	s2	s1	s3	s4
...
...

...

blokovi sa zapisima
o njihovim ispitima

i5	i4	i3	i2	i1
s3	s2	s2	s1	s1
...

...

Značenje:
s1 je položio i1, i2
s2 je položio i3, i4
s3 je položio i5 ...

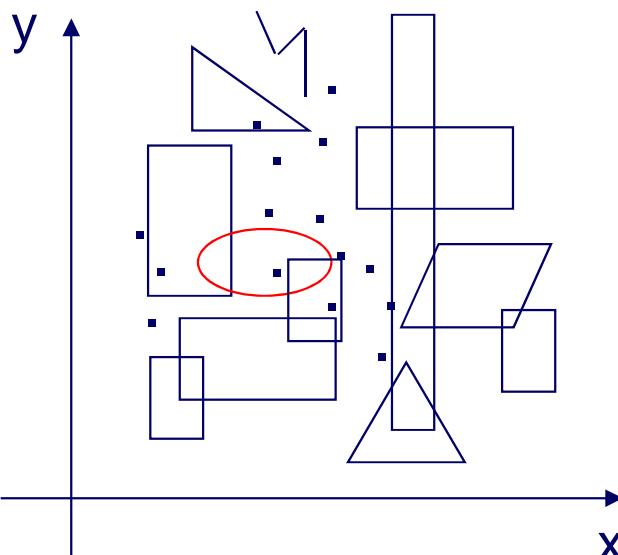
clustered

blokovi sa zapisima o studentima i zapisima o njihovim
ispitima \Rightarrow efikasno obavljanje operacije student \bowtie ispit

s1	i1	i2	s2	i3	i4	s3	i5	i6	i7
...	s1	s1	...	s2	s2	...	s3	s3	s3
...

Višedimenzionalni indeksi (*Multidimensional indexes*)

- primjena:
 - GIS
 - multimedejske baze podataka
 - potpora odlučivanju, skladišta podataka, analiza podataka (*data mining*)

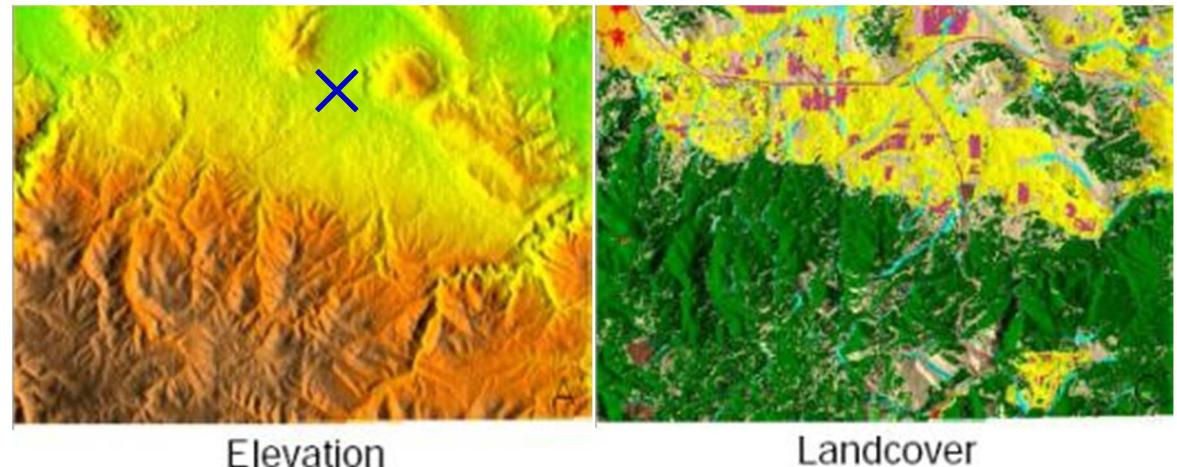


A 2D coordinate system with x and y axes. Several geometric shapes are plotted: squares, triangles, and a circle. A point within a square is highlighted with a red oval.

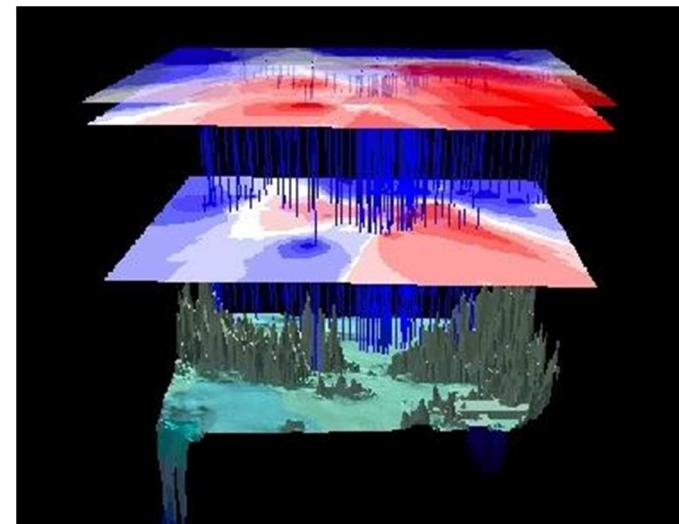
- točke (*point data*), linije (*lines*), regije (*region data*)
 - primjeri prostornih upita (*spatial query*)
 - dohvati sve objekte koji se nalaze na zadanoj udaljenosti od zadanog objekta
 - dohvati objekt zadane vrste koji se nalazi na najvećoj udaljenosti od zadanog objekta
 - dohvati objekte čije se granice sijeku sa zanim objektom

Višedimenzionalni indeksi - primjena

- GIS (2D) - pronađi lokaciju najbližu oznaci \times koja je prekrivena šumom i nalazi se na nadmorskoj visini iznad 800 metara

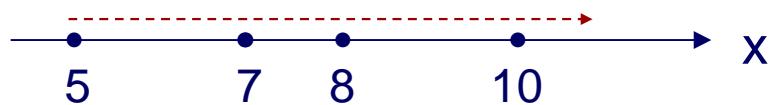


- GIS (3D) - odredi prosječnu temperaturu sloja oceana na dubini od 50 do 100 m u radijusu od 2 nm od trenutne pozicije broda



Zašto B-stabla i slične 1D strukture nisu dovoljni?

- jednodimenzionalni indeksi
 - među vrijednostima koje se pretražuju moguće je utvrditi poredak!
 - ključ pretrage podrazumijeva jednu vrijednost
 - kompozitni indeksi (indeksi sastavljeni od više ulančanih vrijednosti atributa) su također jednodimenzionalni !

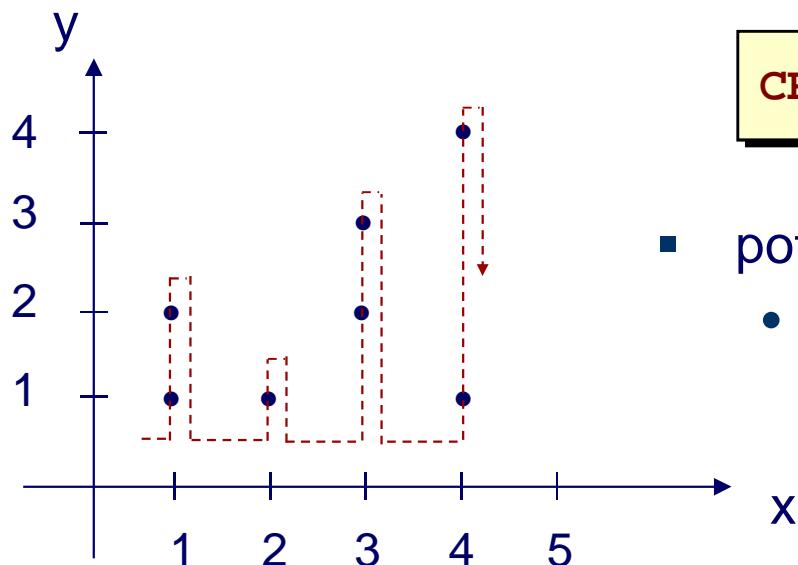


```
CREATE INDEX idxName ON tablename (x);
```

- indeks ostvaruje **potpuno** uređenje prema vrijednosti x
 - { 5, 7, 8, 10 }
- ... WHERE x = 5 ✓
- ... WHERE x BETWEEN 7 AND 10 ✓

Zašto B-stabla i slične 1D strukture nisu dovoljni?

- jednodimenzionalni kompozitni indeksi



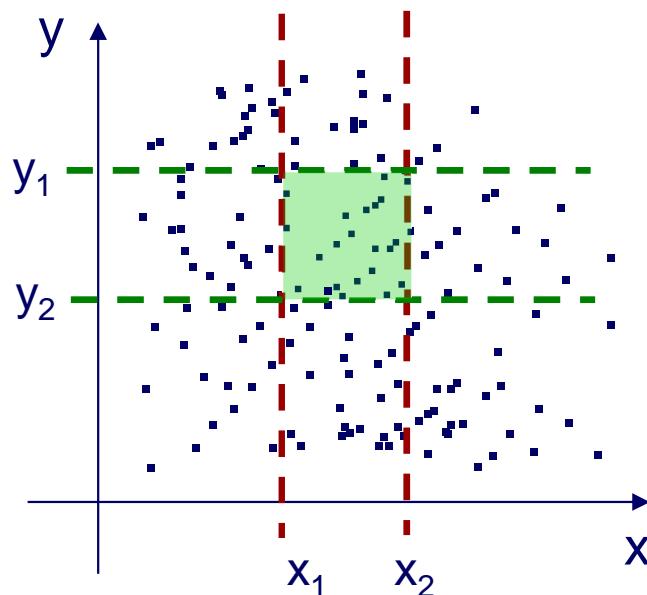
```
CREATE INDEX idxName ON tablename (x, y);
```

- potpuno uređenje prema ulančanoj vrijednosti $x \parallel y$
 - $\{ (1,1), (1,2), (2,1), (3,2), (3,3), (4,1), (4,4) \}$

- ... WHERE $x = 3$ AND $y = 2$ ✓
- ... WHERE x BETWEEN 2 AND 3 AND y BETWEEN 1 AND 3 ✗
 - indeks se može koristiti za izdvajanje skupa točaka za koje je $x \in [2,3]$, ali se tako dobiveni skup točaka mora pretražiti slijedno

Zašto B-stabla i slične 1D strukture nisu dovoljni?

- kombinirano korištenje jednodimenzionalnih indeksa za karakteristični primjer prostornog upita: pripadnost objekata zadatom području u prostoru (*range query*)



WHERE x BETWEEN x1 AND x2
AND y BETWEEN y1 AND y2

```
CREATE INDEX idxName1 ON tablename (x);  
CREATE INDEX idxName2 ON tablename (y);
```

ili

- idxName1 → izdvajanje skupa $[x_1, x_2]$
 - idxName2 → izdvajanje skupa $[y_1, y_2]$
 - idxName1 → izdvajanje skupa $[x_1, x_2]$
 - još veći problem: pronaći najbližu točku točki (x, y) - *nearest-neighbor query*
 - kako odrediti x_1, x_2, y_1 i y_2 unutar kojeg će se tražiti takva točka?
- nakon toga presjek skupova nakon toga slijedna pretraga ⇒ neefikasno obavljanje upita

Koncept

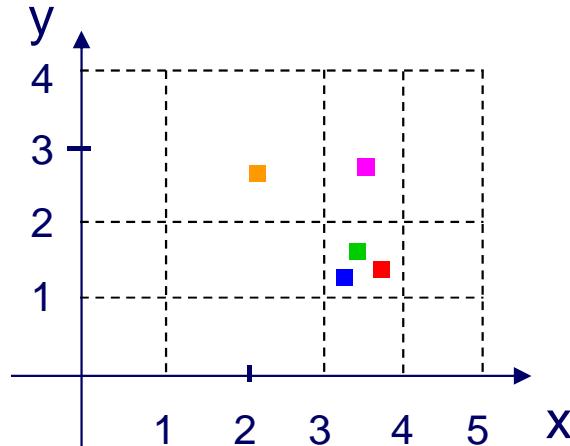
- jednodimenzionalni indeks uzastopno dijeli prostor (jednodimenzionalni) u manje dijelove
 - npr. binarno stablo podijeli prostor na dva dijela - ključevi koji pripadaju lijevoj particiji se smjeste lijevo, ključevi koji pripadaju desnoj particiji se smjeste desno. Dobivene particije se dijele dalje prema istom principu
 - slično, B⁺-stablo dijeli prostor (jednodimenzionalni) na n dijelova
- višedimenzionalni indeks uzastopno dijeli prostor (višedimenzionalni) u manje dijelove
- vrste indeksnih struktura
 - grid file
 - KD tree
 - quad tree
 - R-tree

Grid file

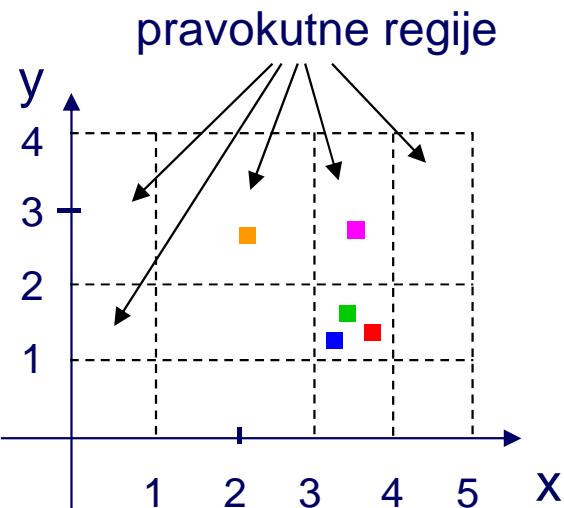
- prikladno za točke (*point data*)
 - pravcima koji "sijeku" x os i y os prostor se dijeli u mrežu pravokutnih regija
 - mreža je u cijelosti definirana točkama na x i y osima u kojima ih pravci sijeku (m točaka na x osi i n točaka na y osi)
-
- definicija mreže može se pohraniti u dva vektora. U primjeru:
 - linearna skala po x osi: pohranjuje se vektor [1, 3, 4, 5]
 - linearna skala po y osi: pohranjuje se vektor [1, 2, 4]
 - utrošak memorije (proporcionalan s): $m + n$
 - dakle, stranice s linearnim skalama koje definiraju mrežu mogu se bez teškoća držati u primarnoj memoriji

 - slično, 3D prostor bi se plohama dijelio u kvadre

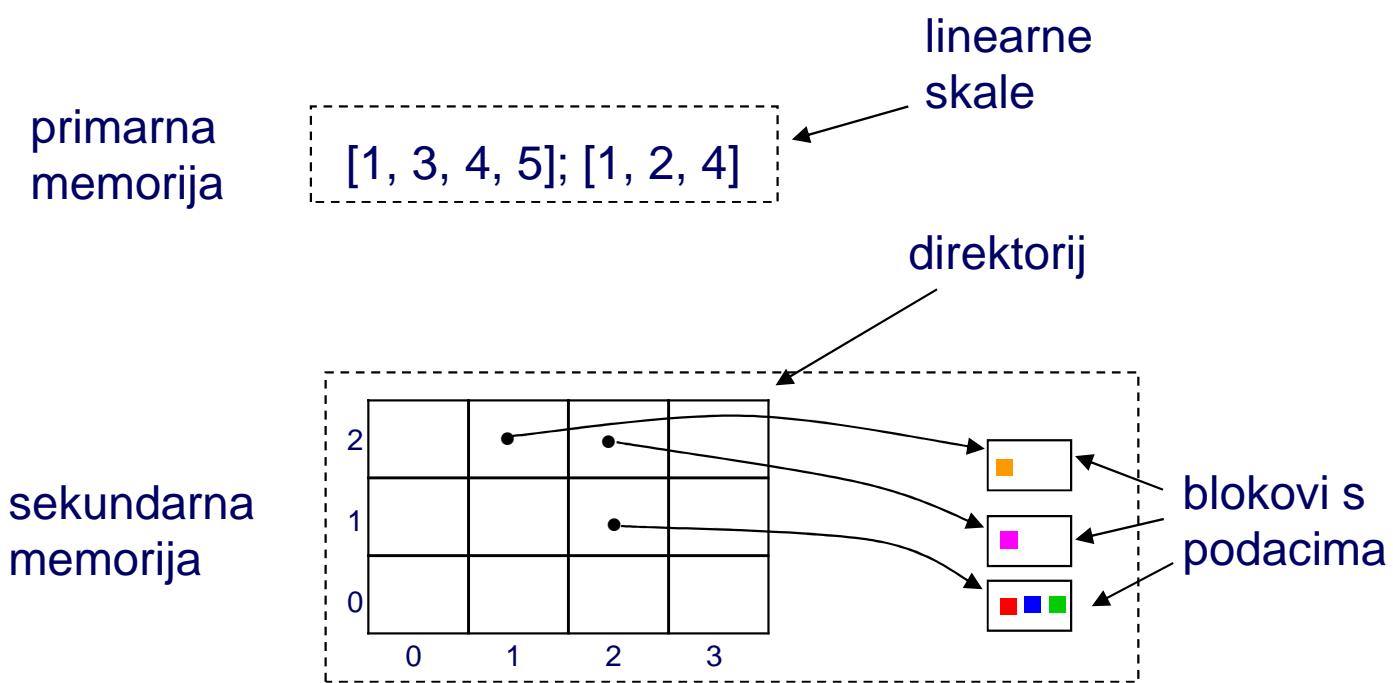
Primjer:



Grid file

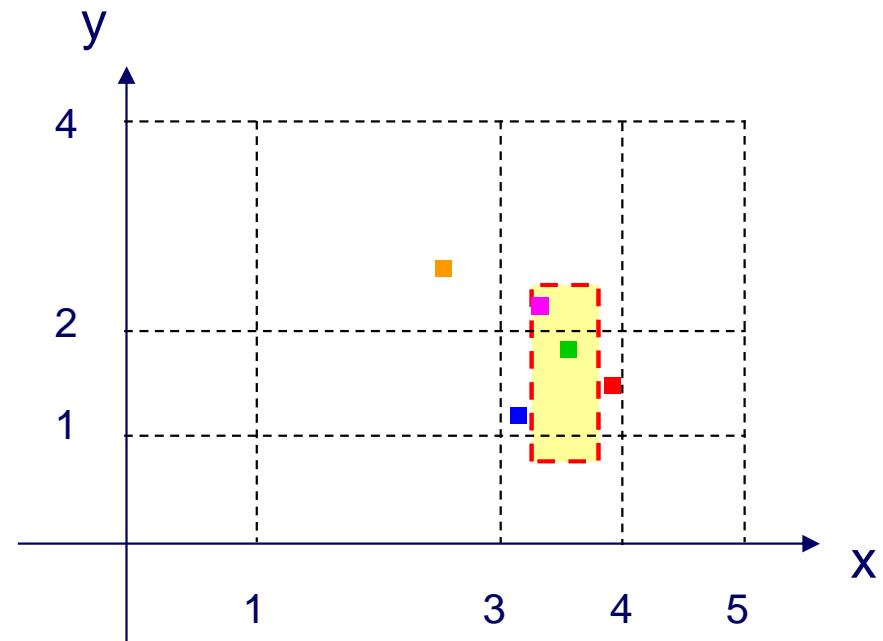
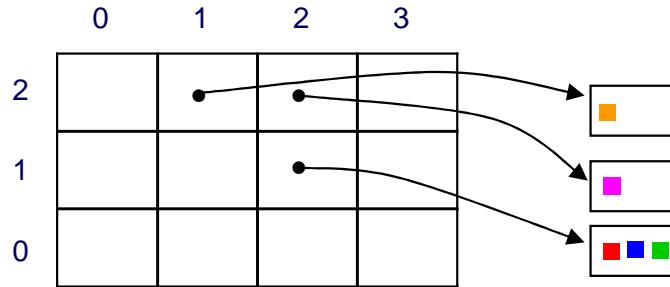


- direktorij (*grid directory*) je polje dimenzija $m \times n$ (ako se radi o 2D prostoru).
- svaka pravokutna regija se preko linearnih skala pridružuje točno jednom elementu direktorija
- element direktorija sadrži adresu bloka s podacima o svim točkama koji pripadaju pravokutnoj regiji koja je pridružena elementu direktorija



Grid file

- pretraga (npr. *range query*)
 - dohvati sve točke koji se nalaze u regiji $[3.2, 3.8] \times [0.8, 2.3]$



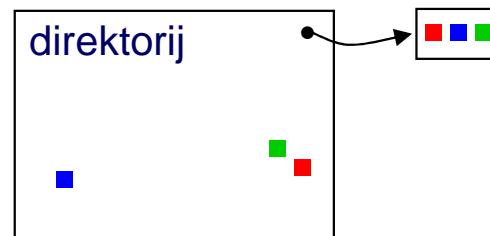
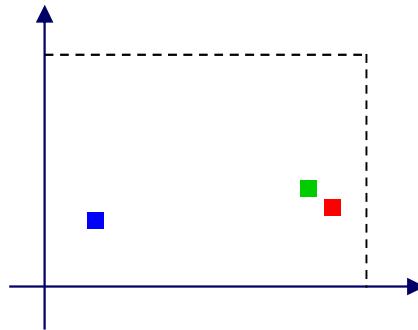
- iz podataka o skalama se bez obavljanja U/I operacija (jer linearne skale su uvijek u primarnoj memoriji) može pronaći pozicija u direktoriju na kojoj se nalazi adresa bloka s podacima o traženim točkama: u primjeru su to pozicije s indeksima (1, 2), (2, 1), (2, 2)
- iz direktorija se čitaju adrese blokova (direktnim pristupom jer se direktorij pohranjuje po retcima ili po stupcima)
- čitaju se blokovi s podacima
 - u učitanom bloku se odrede točke koje zadovoljavaju zadani uvjet pretrage (jer se u blokovima mogu naći i one točke koje ne zadovoljavaju zadani uvjet pretrage)
- za vježbu: kako bi se pronašle točke koje se nalaze u regiji $[1.5, 3.0] \times [0.1, 3.9]$

Grid file

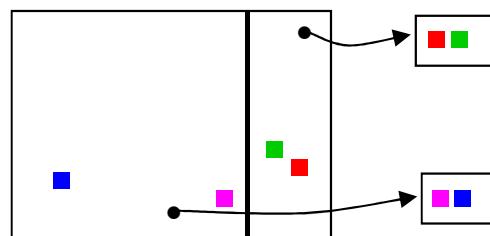
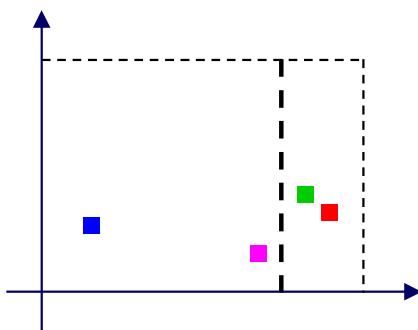
- za dohvat podataka o nekoj točki na koordinatama x, y potrebne su točno 2 U/I operacije
 - jedna za dohvat adresu bloka u direktoriju
 - jedna za dohvat bloka s podacima
- broj U/I operacija za *range query* ovisi o području pretrage i načinu na koji je definirana mreža regija
- u prethodnom primjeru nije objašnjen način na koji su određene skale
 - problem unosa podataka
- kapacitet bloka s podacima je ograničen: ako je blok s podacima u kojem treba upisati podatak popunjen, dodaje se novi blok
- dvije mogućnosti:
 - novi blok povezati s postojećim (kao preljevni blok) - uzrokuje degradaciju performansi (umjesto 2 U/I operacije, moguće više)
 - podijeliti stupac (ili redak) u direktoriju na dva stupca (ili dva retka)
 - ili dva kvadra u 3D
 - postoje različite strategije podjele direktorija

Grid file

- Primjer:**
- neka je kapacitet bloka s podacima 3 zapisa
 - na sljedećim slikama direktorij je prikazan simbolički (bez prikaza linearne skale)



- kako podijeliti regiju nakon prekoračenja kapaciteta bloka?
 - po x osi? po y osi?
 - po pola? tako da u svakoj polovici ostane približno jednako točaka?

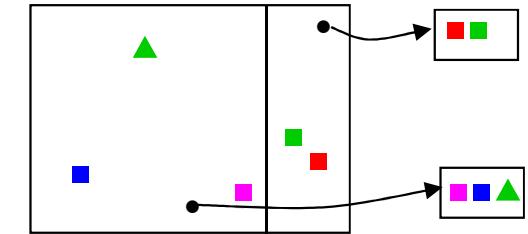


- na sljedećoj slici prikazan je direktorij nakon dodavanja točke ■

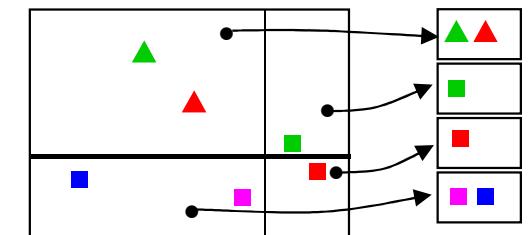
- na sljedećoj slici prikazan je direktorij nakon dodavanja točke ▲

Grid file

- jedna od mogućih strategija odabira osi po kojoj se obavlja podjela: *round-robin*
- svaka sljedeća podjela obavlja se po sljedećoj dimenziji, npr.
 - x, y, z, x, y, z, ...

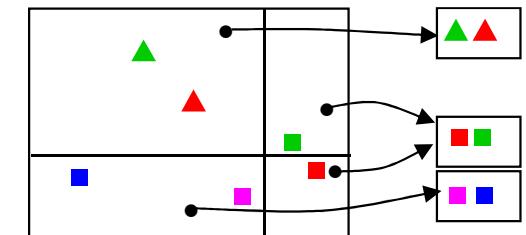


- na sljedećoj slici prikazan je direktorij nakon dodavanja točke ▲

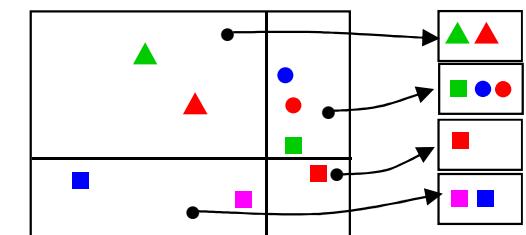


Grid file

- tijekom uzastopnih podjela regija moguća je pojava velikog broja gotovo praznih blokova
- poboljšanje: dopušta se da na različitim pozicijama u direktoriju budu upisane iste adrese blokova s podacima. Sljedeća slika ilustrira primjenu te metode:

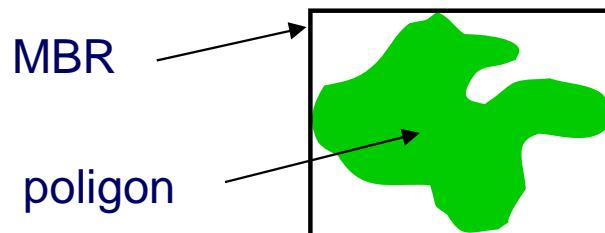


- kada se blok koji sadrži podatke iz dvije ili više regija prepuni nije potrebno dijeliti stupac ili redak u direktoriju: dovoljno je podijeliti blok s podacima
- na slici je prikazan direktorij nakon dodavanja točaka ● i ●



R-stablo

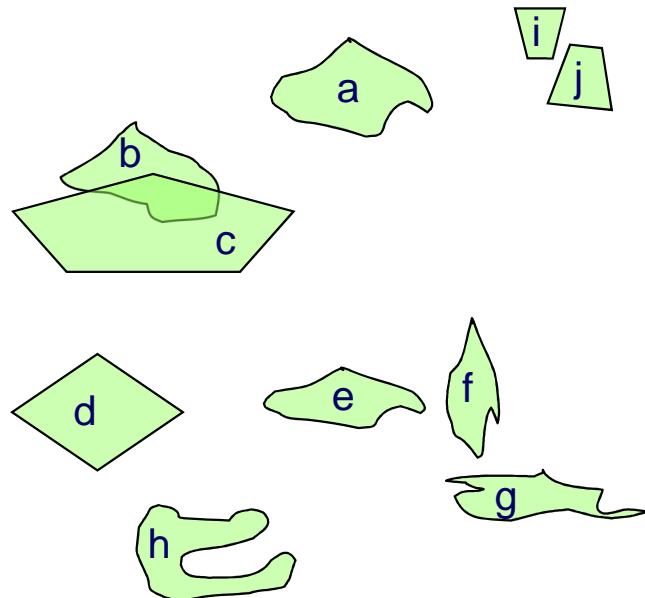
- *Grid file* nije prikladan za indeksiranje poligona i linija
- Ideju B⁺-stabla prilagoditi na 2D ili 3D
- R-stablo prikladno i za regije i za linije i za točke
- R-stablo je balansirano stablo u kojem je ključ pretrage minimalni granični okvir, *minimum bounding rectangle*, MBR
 - minimalni pravokutnik (2D) ili minimalni kvadar (3D) koji u cijelosti obuhvaća lik (2D) ili tijelo (3D)



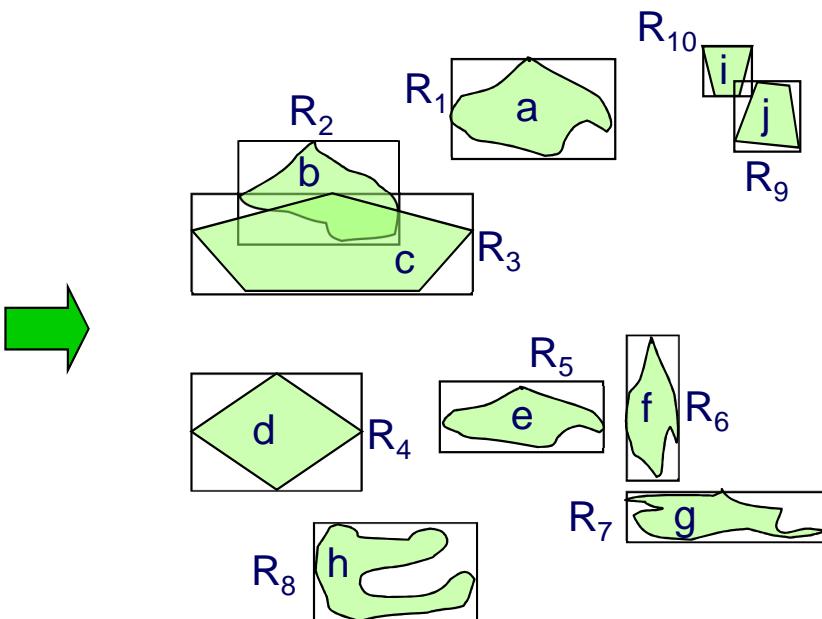
- MBR: x-low, x-high, y-low, y-high

R-stablo

- primjer: izgraditi R-stablo reda 3 za objekte na sljedećoj slici



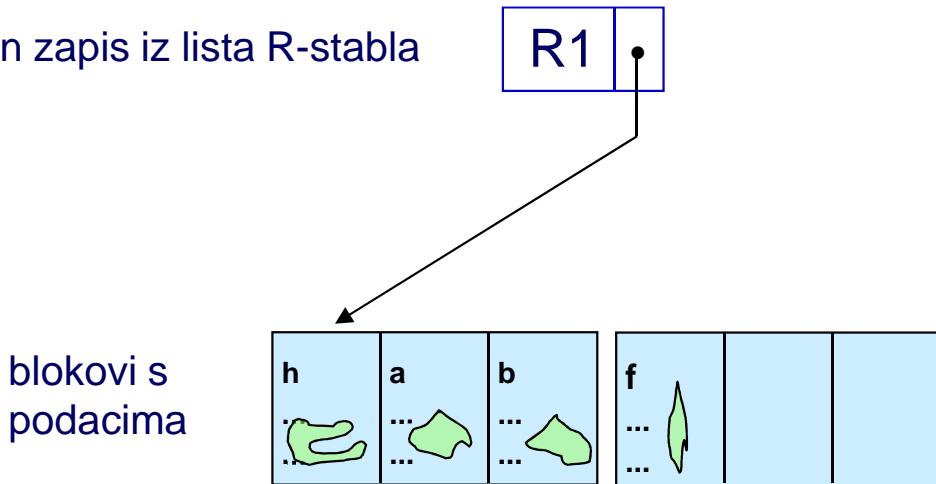
- odrediti MBR-ove za objekte



R-stablo

- struktura lista

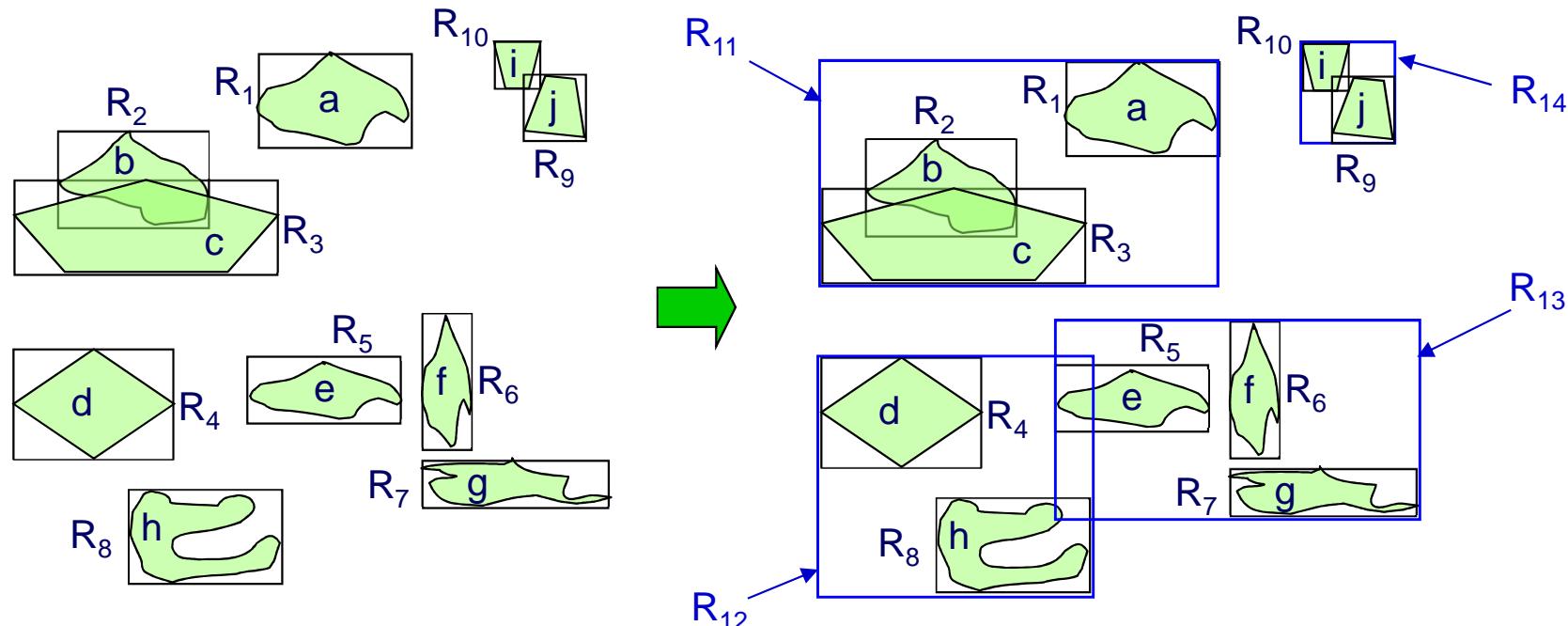
jedan zapis iz lista R-stabla



- jedan zapis u listu R-stabla sadrži podatke o MBR-u (x-low, x-high, y-low, y-high) i pokazivač na **blok** podataka u kojem se nalaze podaci o objektu pokrivenom dotičnim MBR-om
 - slično kao što jedan zapis u listu B-stabla sadrži vrijednost ključa i pokazivač na **blok** podataka u kojem se nalaze podaci za n-torku s dotičnim ključem

R-stablo

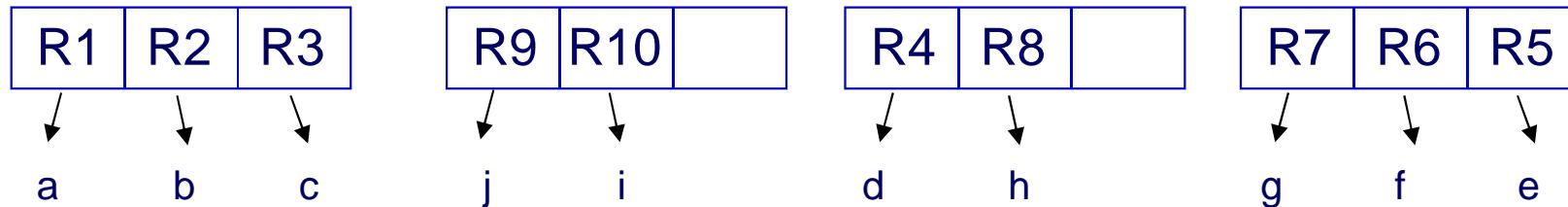
- primjer (nastavak):
- grupirati međusobno bliske MBR-ove u po jedan MBR koji ih prekriva. Broj MBR-ova u jednoj grupi ograničen je kapacitetom čvora, tj. redom stabla
 - R-stablo je balansirano stablo, minimalna popunjenošć čvorova (osim korijena) je 50%



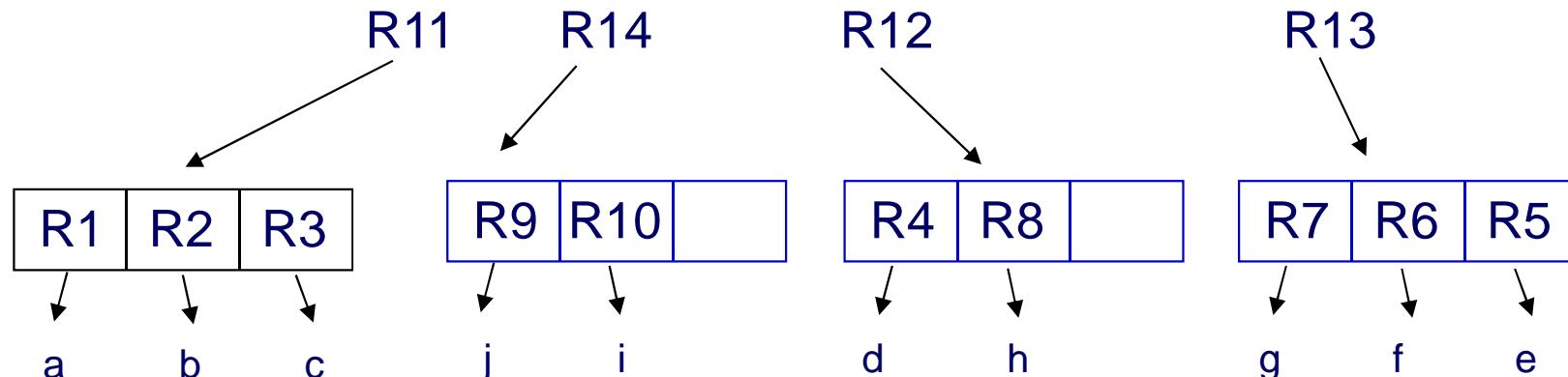
- način grupiranja nije jednoznačan, ali nastoji se minimizirati površina MBR-a
 - * na sljedećim slikama, radi preglednosti, objekti nisu ucrtani u MBR-ove

R-stablo

- primjer (nastavak):
- zapisi za MBR-ove jedne grupe upisuju se u jedan list

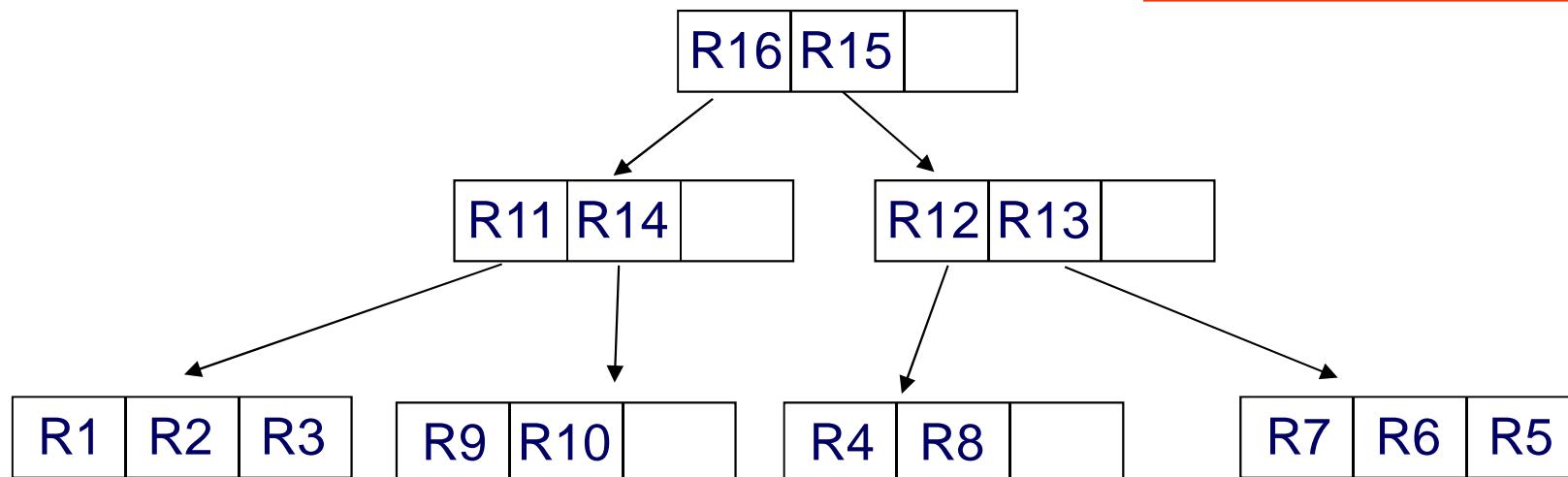
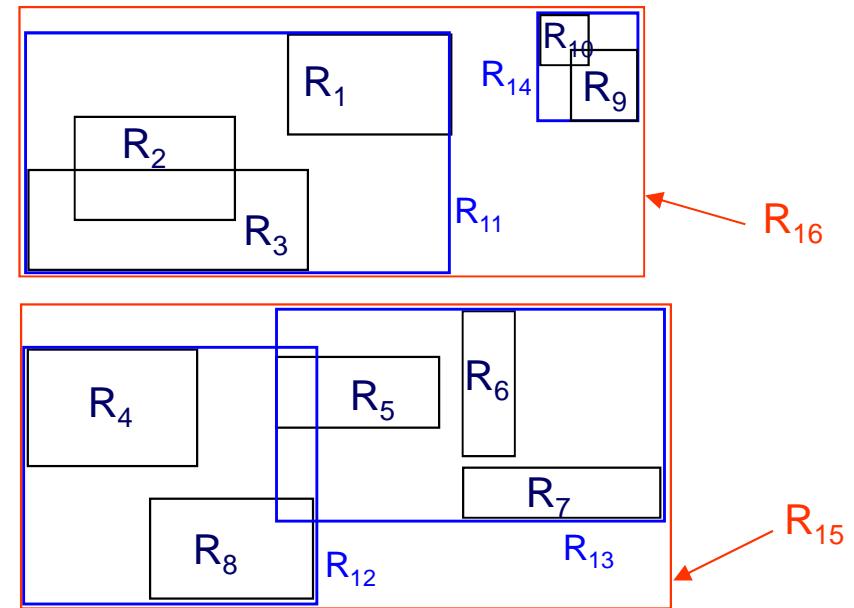


- na svaki list pokazuje po jedan zapis s više razine stabla. U zapisu se nalaze podaci o MBR-u koji obuhvaća grupu MBR-ova iz lista i pokazivač na list



R-stablo

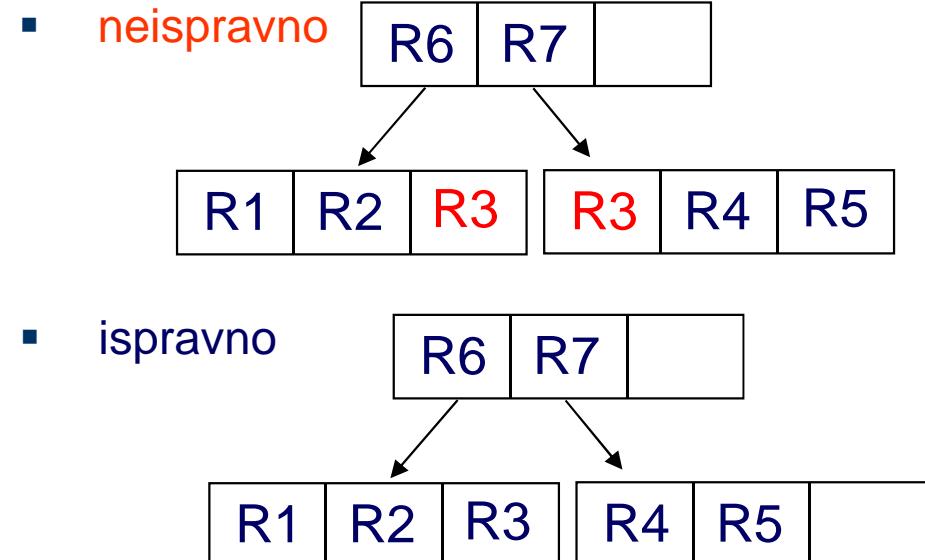
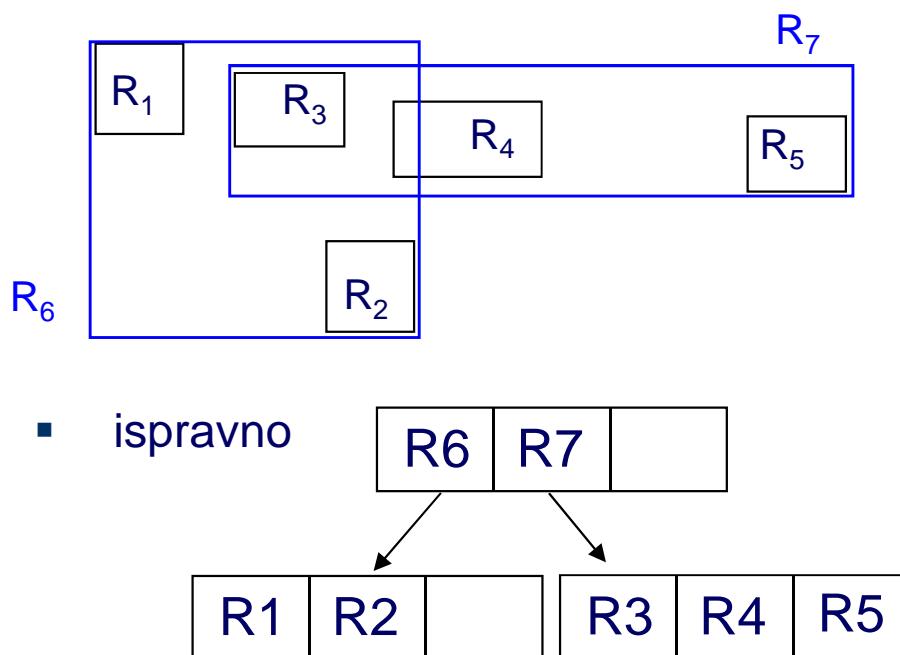
- primjer (nastavak):
- postupak se ponavlja rekurzivno. MBR-ovi s razine iznad lista grupiraju se u grupe međusobno bliskih MBR-ova, zapisi jedne grupe upisuju se u jedan interni čvor ...
- postupak se ponavlja dok se ne dođe do razine korijena R-stabla



R-stablo

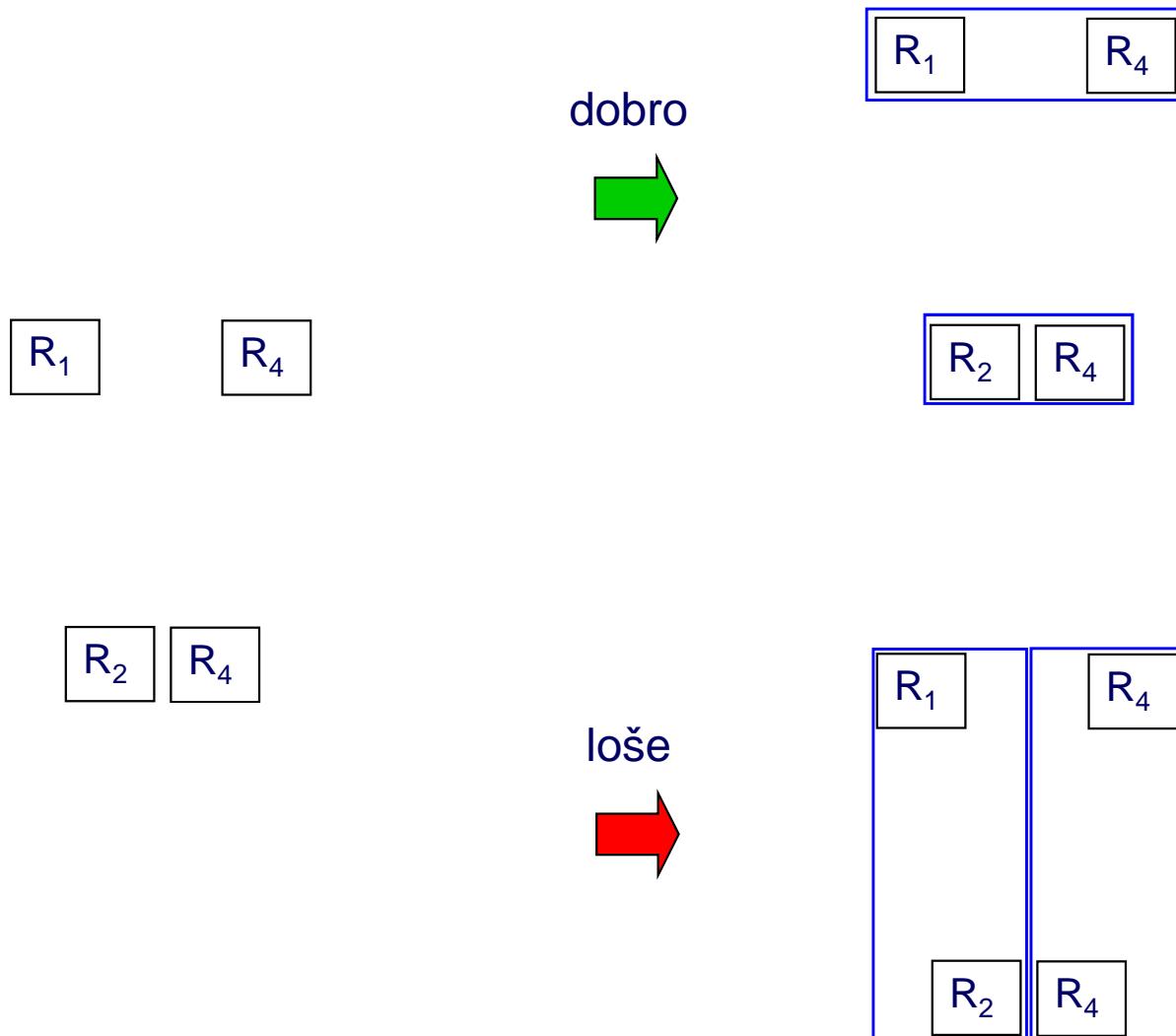
- uočiti:

- svaki MBR-roditelj kompletno prekriva svoju djecu
- MBR-ovi iste razine se smiju preklapati (čim manje - to bolje). Npr. preklapaju se R12 i R13 u prethodnom primjeru
- zapis za jedan MBR se u R-stablu uvijek nalazi u samo jednom čvoru
 - ako se MBR_X i MBR_Y na i -toj razini preklapaju, moguće je da će neki MBR_Z na razini $i+1$ biti obuhvaćen i sa MBR_X i sa MBR_Y . U takvom slučaju, MBR_Z se dodaje ili u čvor na kojeg pokazuje MBR_X ili u čvor na kojeg pokazuje MBR_Y (ne u oba)

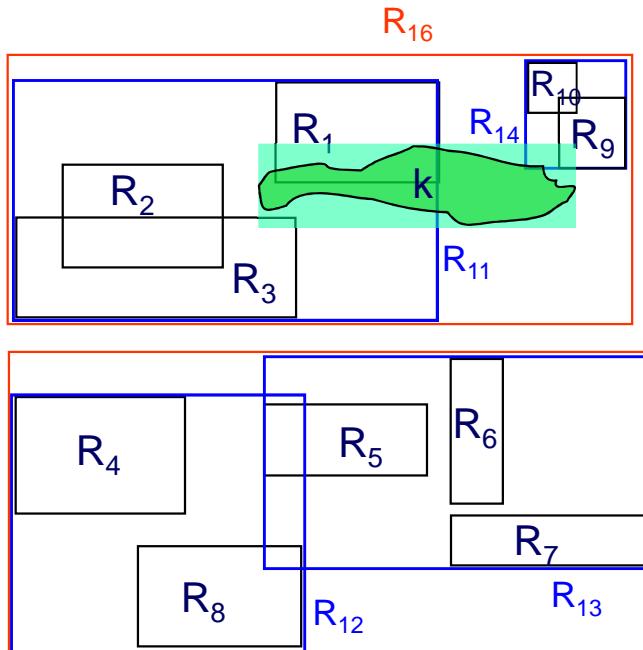


R-stablo

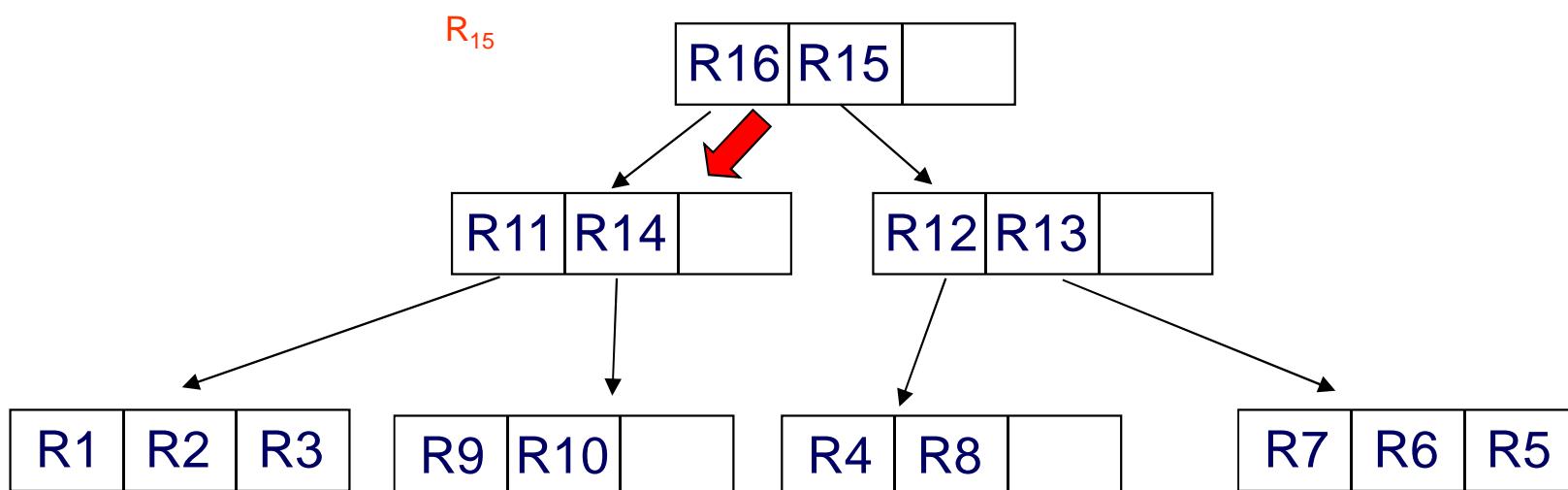
- algoritmi za određivanje minimalnog graničnog okvira su relativno kompleksni
- primjer dobrog i lošeg odabira



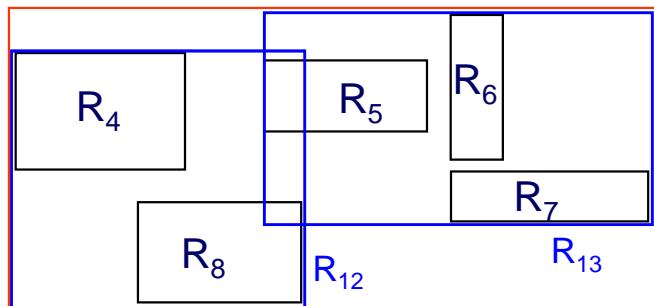
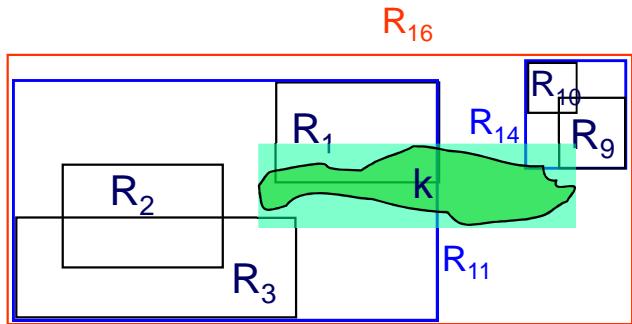
R-stablo - pretraga



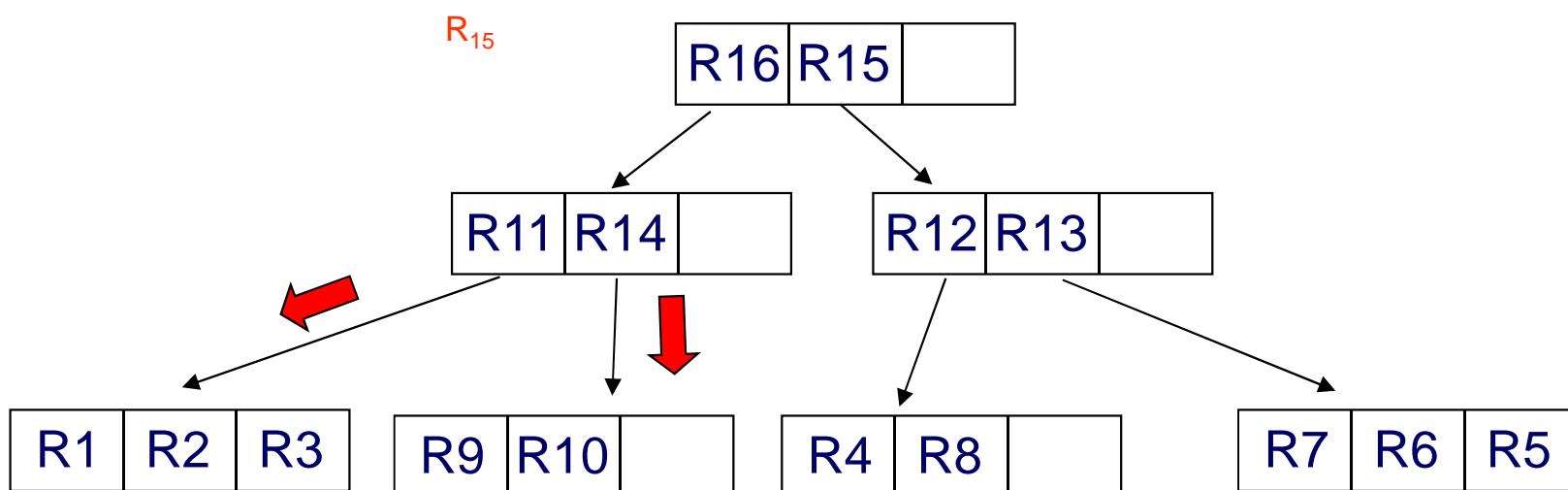
- primjer prostornog upita: dohvatiti sve objekte koji se sijeku sa zadanim objektom k (pričadni MBR objekta k je MBR_x)
- pretraga kreće od korijena
- naći MBR-ove u presjeku s MBR_x i slijediti sve pričadne kazaljke \Rightarrow R₁₆
- ponavljati na svakoj razini do lista



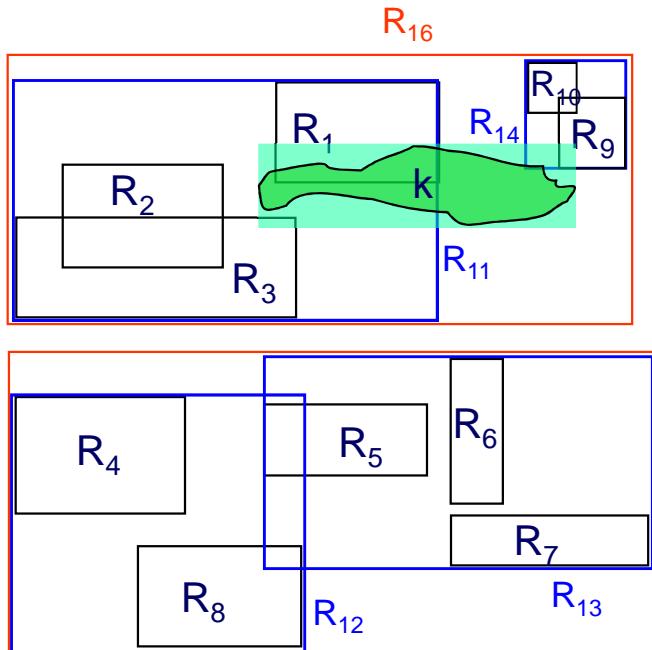
R-stablo - pretraga



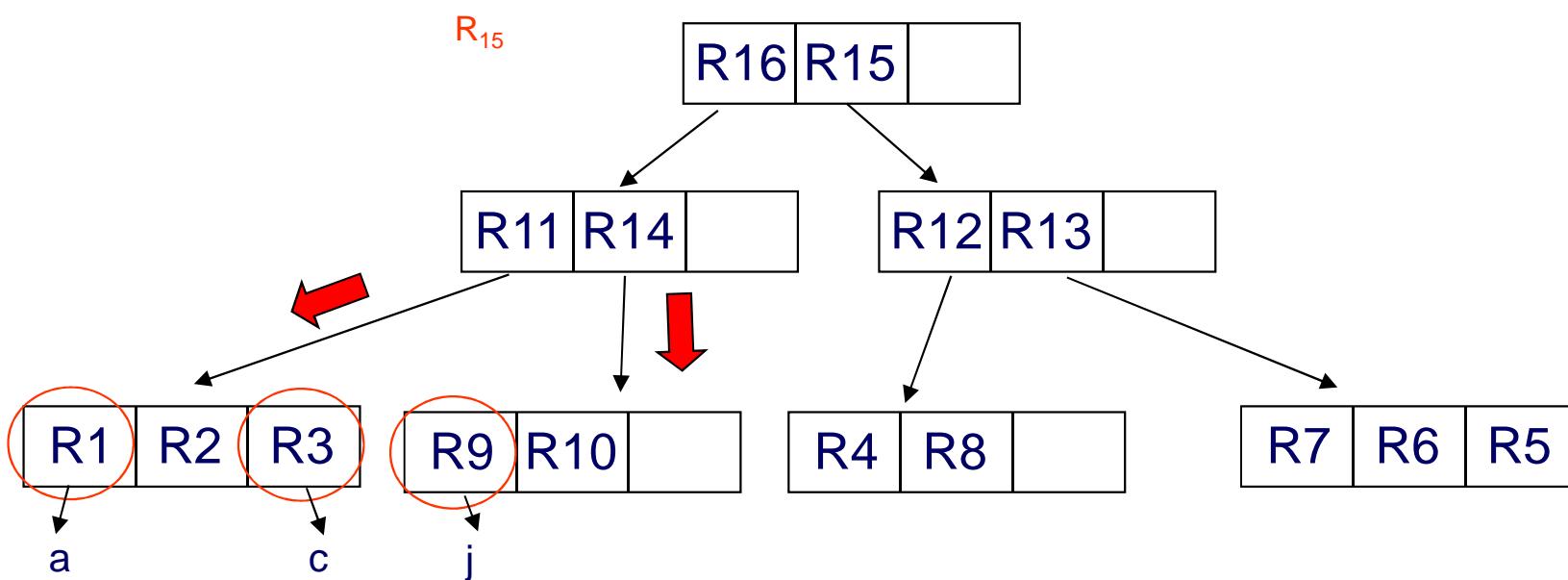
- na sljedećoj razini (razini 1) pronaći MBR-ove u presjeku s MBR_x i slijediti sve pripadne kazaljke $\Rightarrow R11, R14$
- očito, za razliku od B-stabla, moguće je da će se morati slijediti kazaljke u više podstabala



R-stablo - pretraga



- na sljedećoj razini (razini 2) pronaći MBR-ove u presjeku s $MBR_x \Rightarrow R_1, R_3, R_9$ (priпадni objekti su a, c, j)
- provjeriti koji od objekata a, c, j zaista jest u presjeku s objektom k $\Rightarrow a$
- za razliku od B-stabla, pretragom R-stabla može se "pronaći" više objekata nego treba, ali nikako ne manje (višak se eliminira naknadnom provjerom)

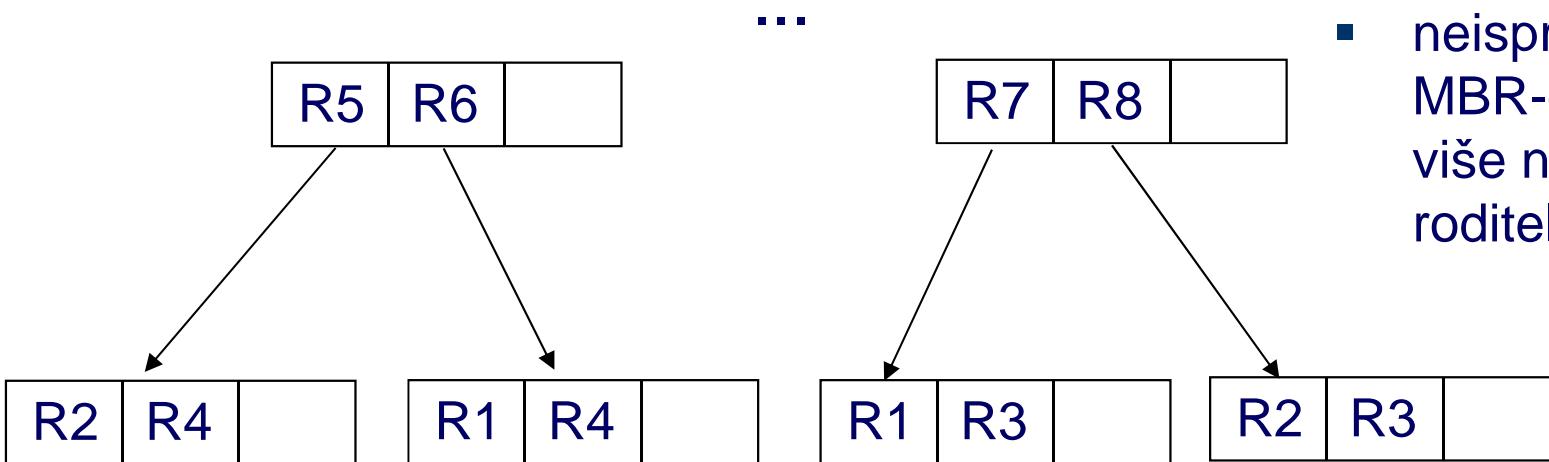
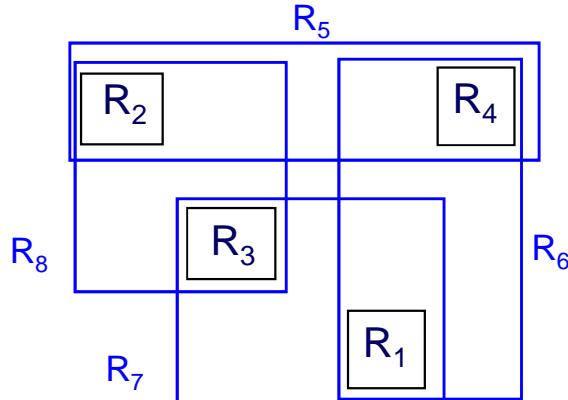


R-stablo - primjer

- svaki roditelj kompletno prekriva svoju djecu
- MBR-ovi se smiju preklapati

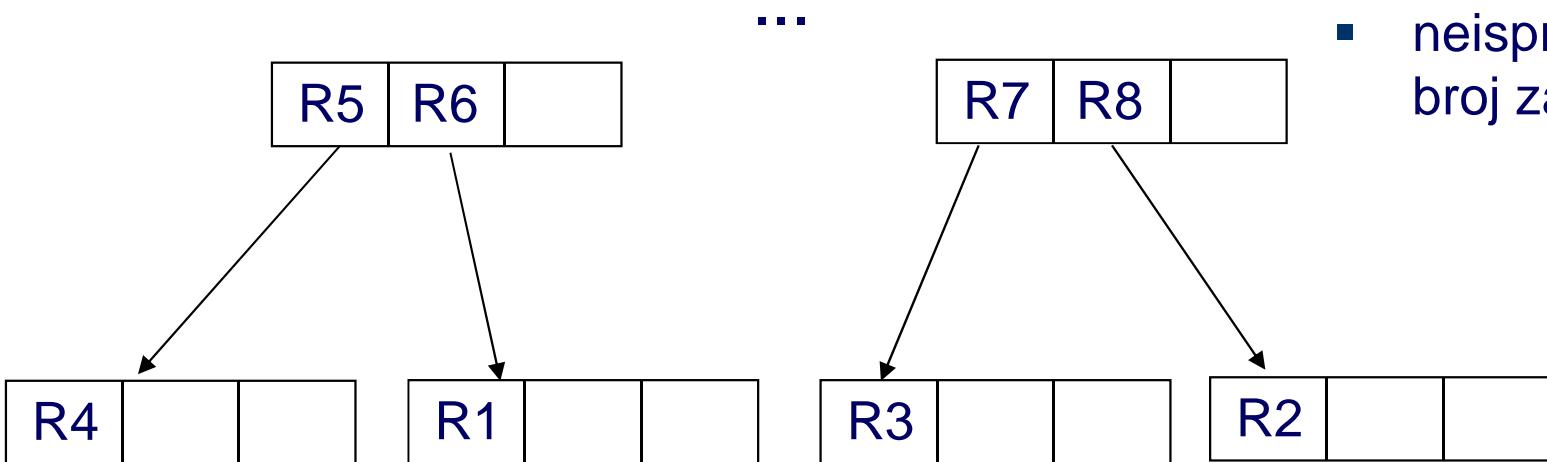
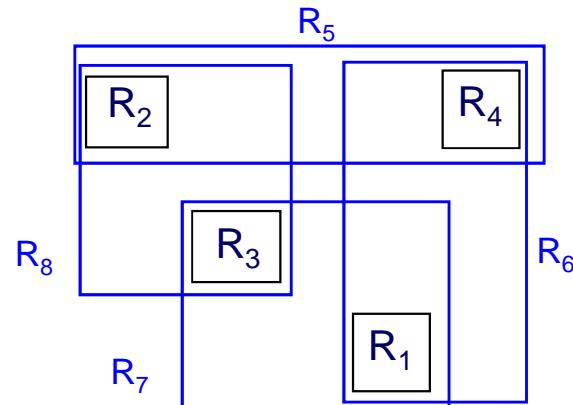
ALI

- ako se MBR-ovi preklapaju i neko dijete je prekriveno s dva (ili više MBR-ova), u stablu se dijete smije pridružiti **samo jednom** od roditelja koji ga prekrivaju



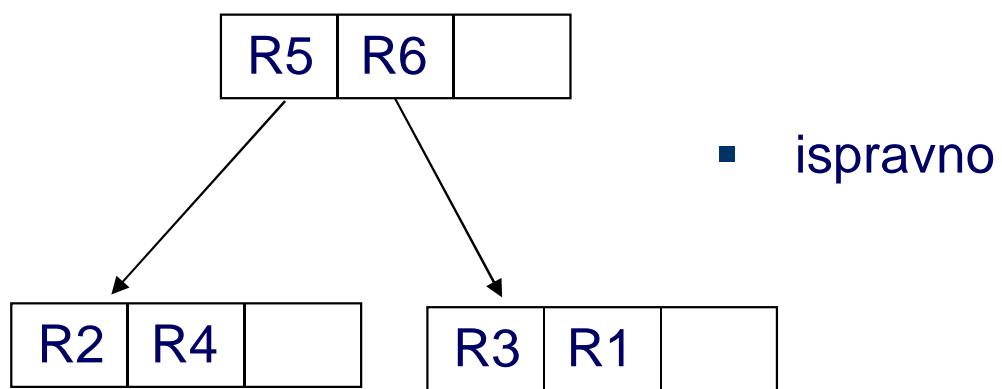
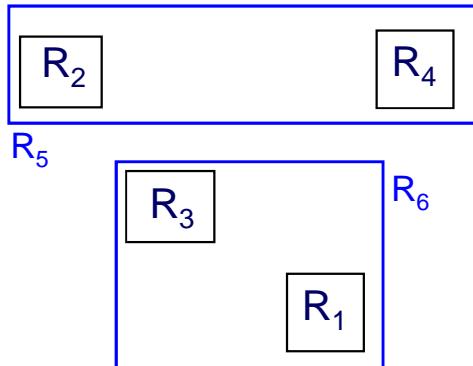
- neispravno - neki MBR-ovi su pridruženi više nego jednom roditelju

R-stablo - primjer (nastavak)



- neispravno - premalen broj zapisa u listovima

R-stablo - primjer (nastavak)



Literatura:

- H. Garcia-Molina, J. Ullman, J. Widom. Database system implementation. Prentice-Hall International, Inc. Upper Saddle River, New Jersey. 2000.
- A. Silberschatz, H. Korth, S. Sudarshan. Database system concepts, 4th ed. McGraw-Hill. 2002.
- R. Elmasri, S. Navathe. Fundamentals of database systems, 3rd ed. Addison-Wesley Publishing Company. Reading, Massachusetts. 2000.

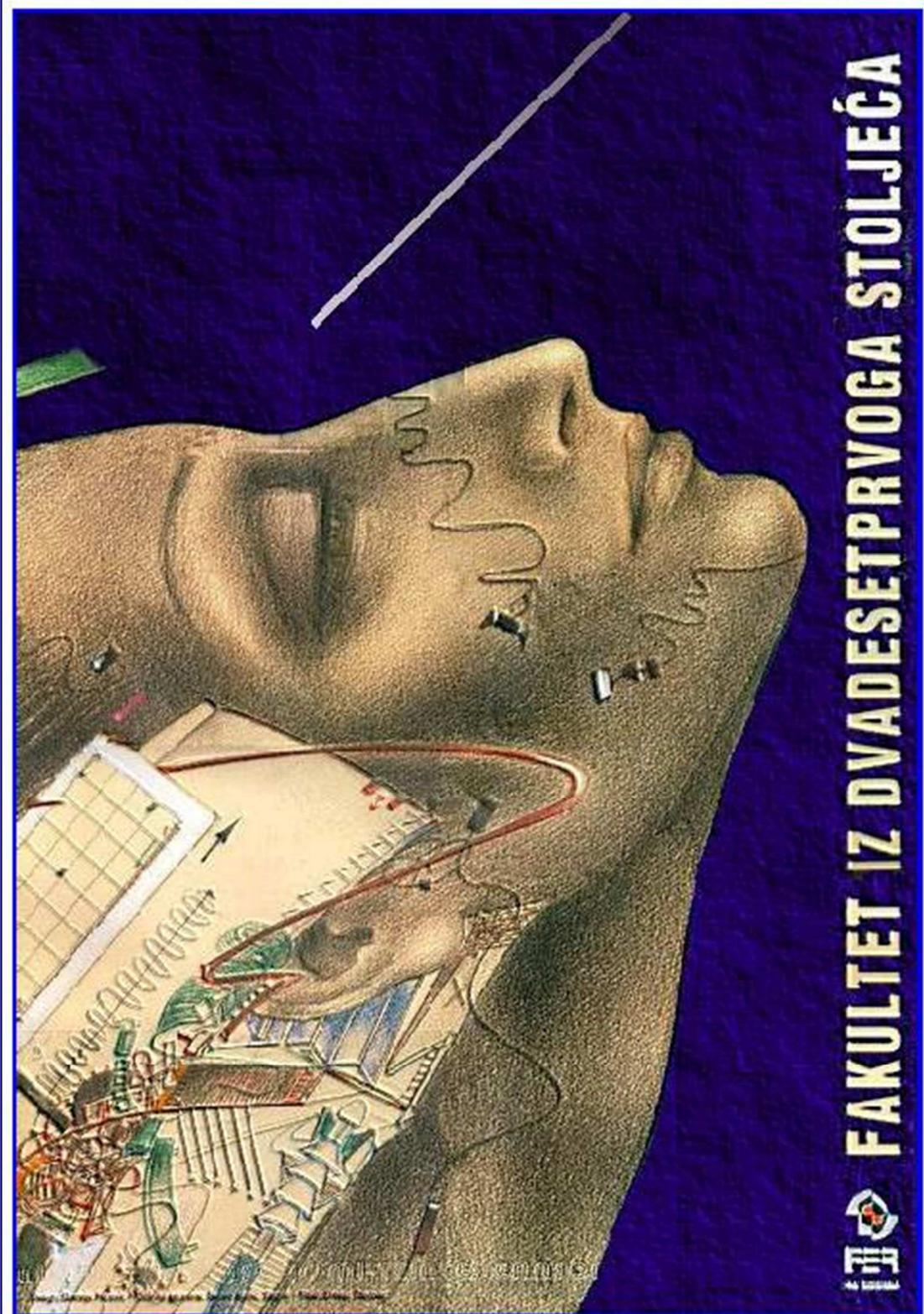
Sustavi baza podataka

Predavanja

3. Obavljanje upita

(1. dio)

ožujak 2014.

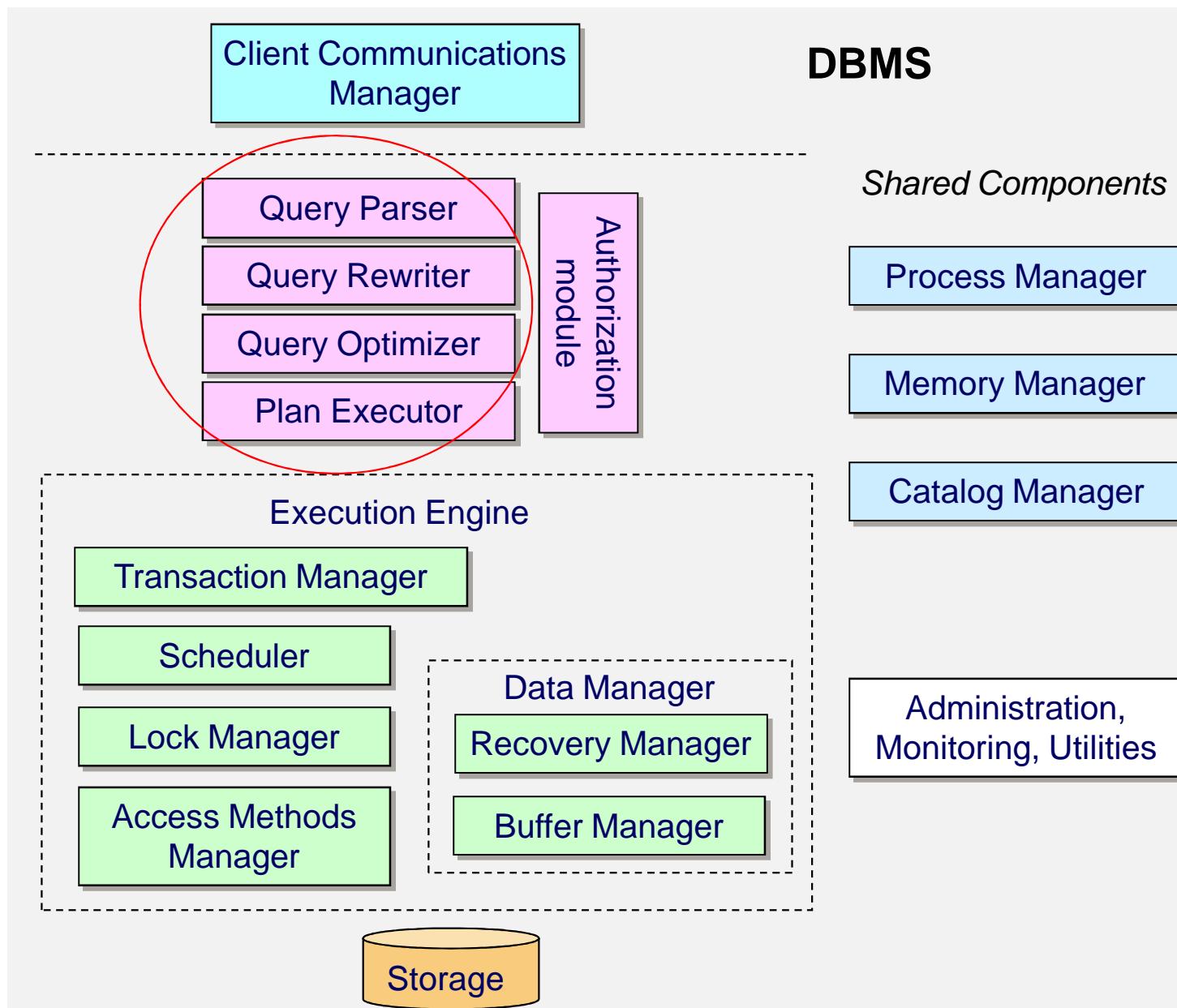


FAKULTET IZ DVADESETPRVOGA STOLJEĆA

Obavljanje upita

- obavljanje upita (*query processing*)
 - skup aktivnosti vezanih uz ekstrakciju podataka iz baze podataka
- starije generacije sustava za upravljanje bazama podataka
 - mrežni, hijerarhijski model podataka
 - upitni jezik ugniježđen u proceduralni jezik treće generacije (npr. COBOL, PL/I). Navigacijski upitni jezici: *Network DML*, *Hierarchical DML*
 - odgovornost programera pri odabiru "optimalne" strategije ekstrakcije podataka
- moderni sustavi za upravljanje bazama podataka
 - relacijski, objektno-relacijski, objektni modeli podataka
 - SQL je neproceduralni jezik. Korisnik/programer definira *što* rezultat treba sadržavati, a ne *kako* do rezultata doći
 - SUBP odabire najpogodniju strategiju izvršavanja upita
 - oslobađa korisnika/programera odgovornosti (i potrebe da zna *kako*)
 - korisnik/programer nije u prilici odabrati lošu strategiju

Komponente SUBP-a



Obavljanje upita

Primjer:

```
SELECT DISTINCT imeStud, prezStud
  FROM stud
    JOIN ispit ON stud.mbr = ispit.mbrStud
    JOIN mjesto ON stud.pbrStan = mjesto.pbr
 WHERE ocjena = 4
   AND nazMjesto = 'Zagreb';
```

$$\pi_{imeStud, prezStud}(\sigma_{nazMjesto='Zagreb' \wedge ocjena=4}((stud \bowtie ispit) \bowtie mjesto))$$

```
SELECT DISTINCT imeStud, prezStud
  FROM mjesto CROSS JOIN stud
    CROSS JOIN ispit
 WHERE nazMjesto = 'Zagreb'
   AND ocjena = 4
   AND stud.pbrStan = mjesto.pbr
   AND ispit.mbrStud = stud.mbr;
```

$$\pi_{imeStud, prezStud}(\sigma_{nazMjesto='Zagreb' \wedge ocjena=4 \wedge pbrStan=pbr \wedge mbrStud=mbr}(mjesto \times stud \times ispit))$$

SUBP će u oba slučaja obaviti istu operaciju:

$$\pi_{imeStud, prezStud}(\sigma_{ocjena=4}(ispit) \bowtie (stud \bowtie \sigma_{nazMjesto='Zagreb'}(mjesto)))$$

Obavljanje upita

```
SELECT AVG(ocjena)
  FROM stud
    JOIN ispit ON stud.mbr = ispit.mbrStud
    JOIN mjesto ON stud.pbrStan = mjesto.pbr
 WHERE nazMjesto = 'Zagreb';
```

card(mjesto) = 10 000
card(stud) = 50 000
card(ispit) = 1 000 000
bez indeksa

→ 1.8 sec

Eksplicitni zahtjev optimizatoru za promjenom plana izvršavanja istog upita

- ORDERED: relacije spajati redoslijedom kojim su navedene u FROM listi

```
SELECT {+ORDERED} AVG(ocjena)
       FROM stud
             JOIN ispit ON stud.mbr = ispit.mbrStud
             JOIN mjesto ON stud.pbrStan = mjesto.pbr
      WHERE nazMjesto = 'Zagreb';
```

(stud >< ispit) >< $\sigma_{\text{nazMjesto}='Zagreb'}$ (mjesto)
mbr=mbrStud pbrStan=pbr

→ 10.5 sec

Bag (multiset, multiskup) verzije operacija

- specifičnost SQL-a u odnosu na relacijski model podataka
 - tablica (*table*) u SQL-u nije nužno relacija (skup n-torki), nego multiskup (*multiset, bag*) n-torki. Stoga se i skup operacija mora proširiti operacijama koje kao operande koriste multiskupove

"*bag*" verzije operacija unije, presjeka i razlike

- neka su r i s "relacije" u kojima je dopuštena pojava više istih n-torki
- ako se n-torka t u "relaciji" r nalazi m puta, u "relaciji" s se nalazi n puta, tada
 - u rezultatu $r \cup^B s$ n-torka t se nalazi $n+m$ puta **UNION ALL**
 - u rezultatu $r \cap^B s$ n-torka t se nalazi $\min(m, n)$ puta **INTERSECT ALL**
 - u rezultatu $r \setminus^B s$ n-torka t se nalazi $\max(0, m-n)$ puta **EXCEPT ALL**

"*bag*" verzija operacija projekcije

- $\pi_L^B(r)$ - izdvajanje vertikalnog podskupa relacije prema listi atributa L (ali bez eliminacije duplikata)

Bag (multiset, multiskup) verzije operacija

Primjer:

r	E	F
1	A	
1	A	
2	B	
2	B	
2	B	

s	E	F
1	A	
1	A	
2	B	

t	F	G
A	5	
A	6	
C	7	

$r \cup^B s$

	E	F
1	A	
1	A	
1	A	
1	A	
2	B	
2	B	
2	B	
2	B	

$r \cap^B s$

	E	F
1	A	
1	A	
2	B	

$r \setminus^B s$

	E	F
2	B	
2	B	

slično i za ostale operacije:

$\sigma_{E=1}^B(s)$

	E	F
1	A	
1	A	

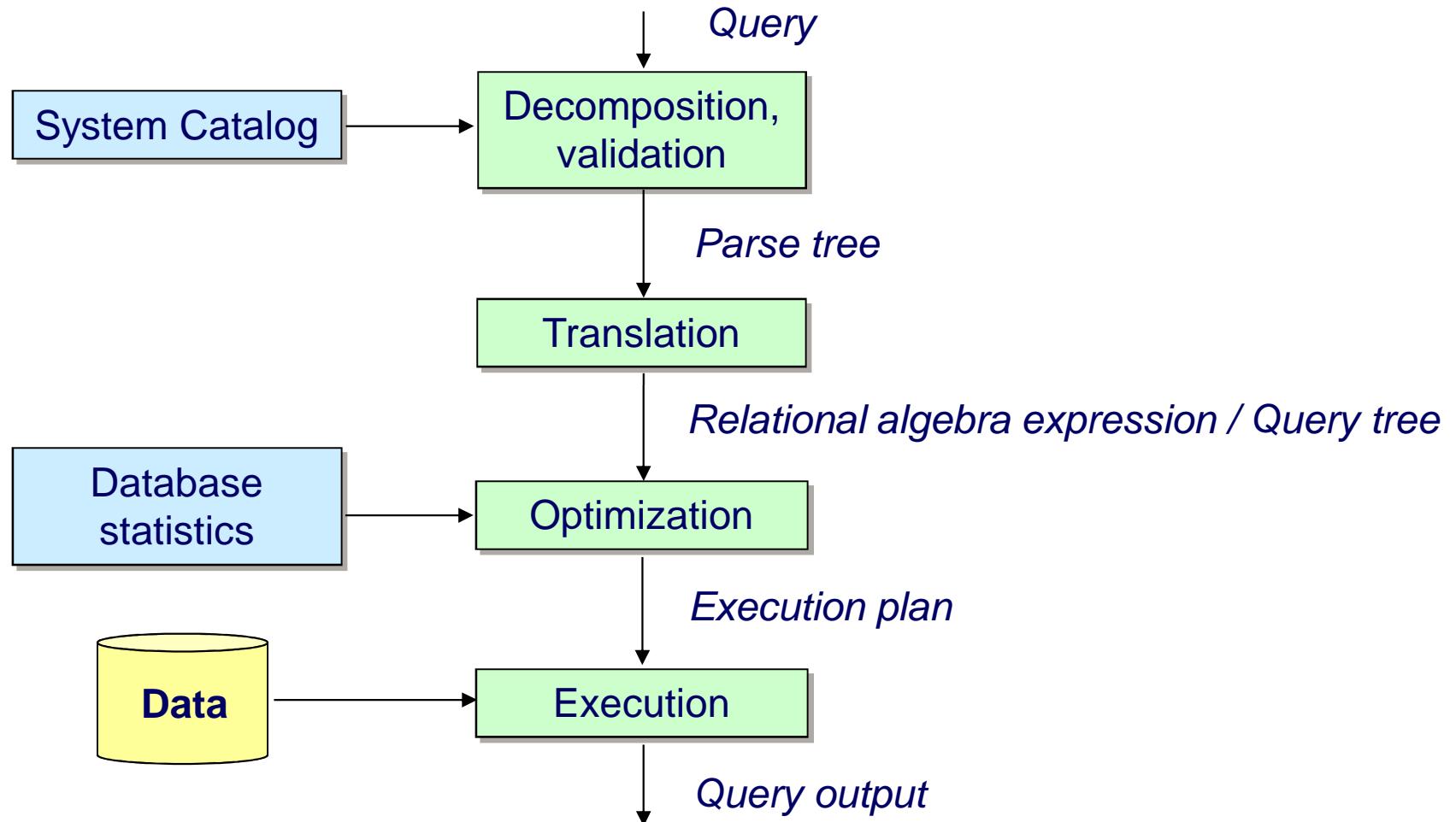
$\pi_E^B(s)$

E
1
1
2

$s \triangleright \triangleleft^B t$

E	F	G
1	A	5
1	A	5
1	A	6
1	A	6

Obavljanje upita



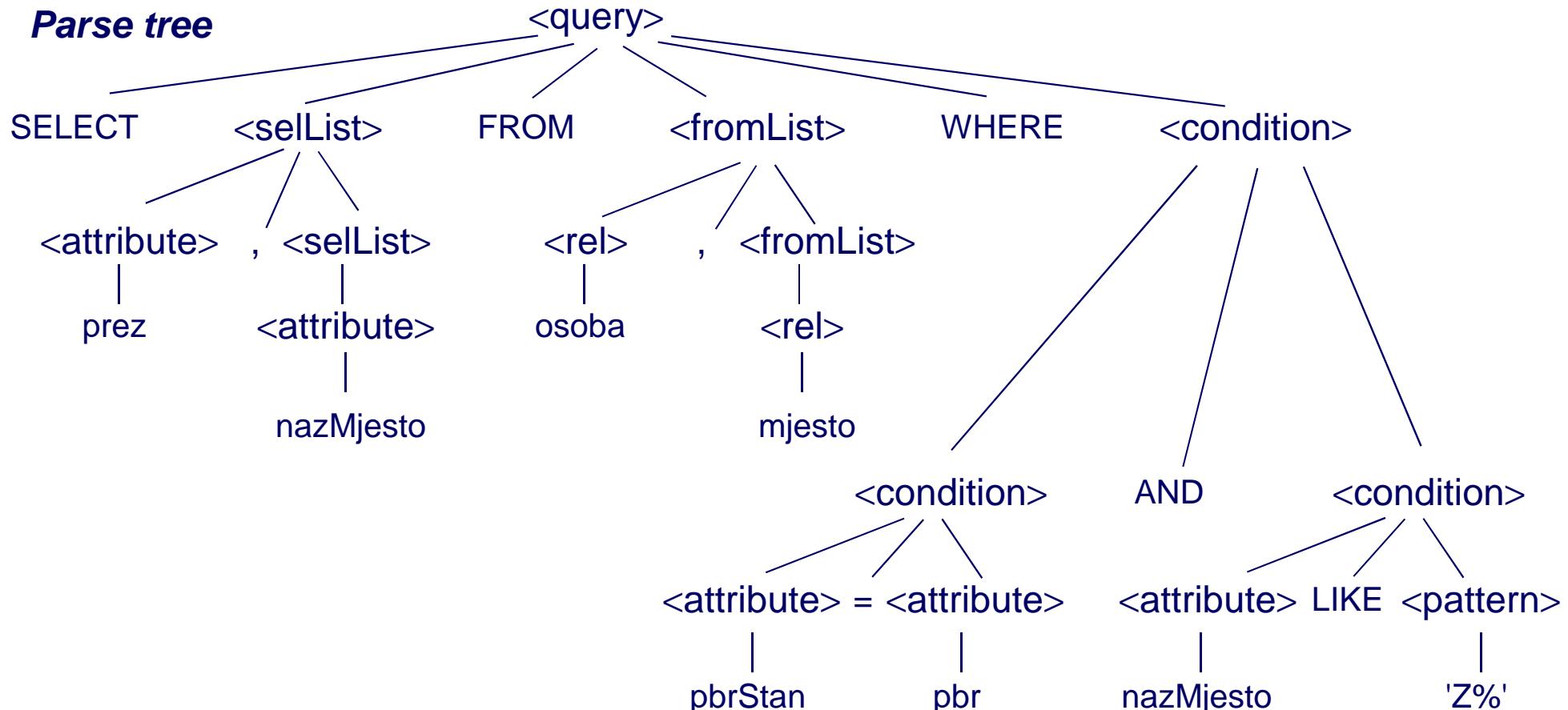
Dekompozicija i validacija

- leksička i sintaktička analiza (tehnike prevodenja programskih jezika)

Primjer:

```
SELECT prez, nazMjesto FROM osoba, mjesto  
WHERE osoba.pbrStan = mjesto.pbr  
AND mjesto.nazMjesto LIKE 'Z%'
```

Parse tree



Dekompozicija i validacija

- obavlja se zamjena virtualne relacije (*view*) stablom (*parse tree*) koje proizlazi iz definicije virtualne relacije u rječniku podataka (modifikacija upita)
 - materijalizirane virt. relacije se promatraju jednako kao temeljne relacije
- obavlja se transformacija podupita
- validacija (semantička analiza):
 - ispitati korištenje naziva relacija. Npr. svaka "relacija" iz FROM liste mora biti relacija opisana u rječniku podataka
 - verificirati ispravno korištenje naziva atributa
 - razriješiti imena atributa (kojoj relaciji pripada koji atribut, postoje li dvosmislenosti)
 - nalazi li se atribut u dosegu (scope) relacije spomenute u pripadnoj FROM listi (→Baze podataka→podupiti)
 - provjeriti tipove podataka i primjenjivost operacija nad objektima (npr. množenje cijelog broja i niza znakova)
- pojednostavljivanje i normalizacija *(query rewriter)*
 - npr. predikati za selekciju → konjunktivna normalna forma
 - eliminacija tautologija i kontradikcija

Translacija u inicijalni izraz relacijske algebre

- *parse tree* se transformira u inicijalni izraz relacijske algebre koji se može prikazati kao stablo upita (*query tree*)
- pravilo za transformaciju stabla (*parse tree*) u inicijalni izraz relacijske algebre - za jednostavne upite
 1. načiniti Kartezijev produkt svih relacija navedenih u `<fromList>`
 2. nad rezultatom prvog koraka obaviti operaciju selekcije σ_F gdje je F uvjet naveden u `<condition>`
 3. nad rezultatom drugog koraka obaviti operaciju projekcije π_L gdje je L lista izraza navedenih u `<selList>`
 - *bag* verziju operacije, ako nije navedeno DISTINCT

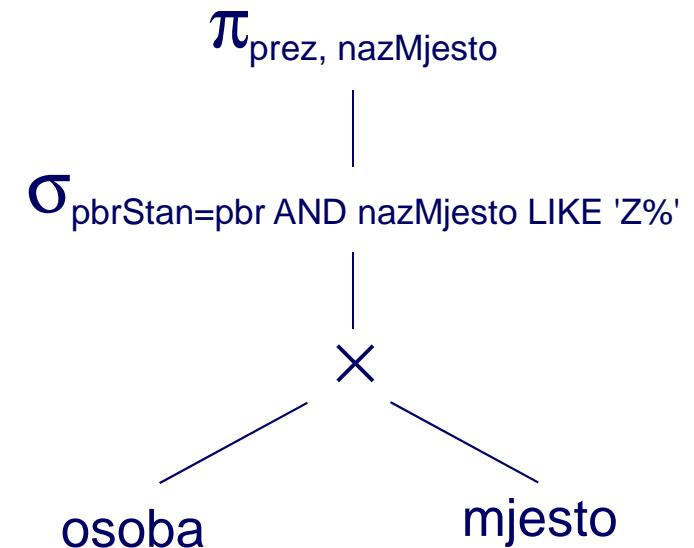
Translacija u inicijalni izraz relacijske algebre

Primjer:

```
SELECT DISTINCT prez, nazMjesto  
  FROM osoba, mjesto  
 WHERE osoba.pbrStan = mjesto.pbr  
   AND mjesto.nazMjesto LIKE 'Z%'
```

- izraz relacijske algebre:

$$\pi_{\text{prez, nazMjesto}}(\sigma_{\text{pbrStan=pbr AND nazMjesto LIKE 'Z%'}}(\text{osoba} \times \text{mjesto}))$$

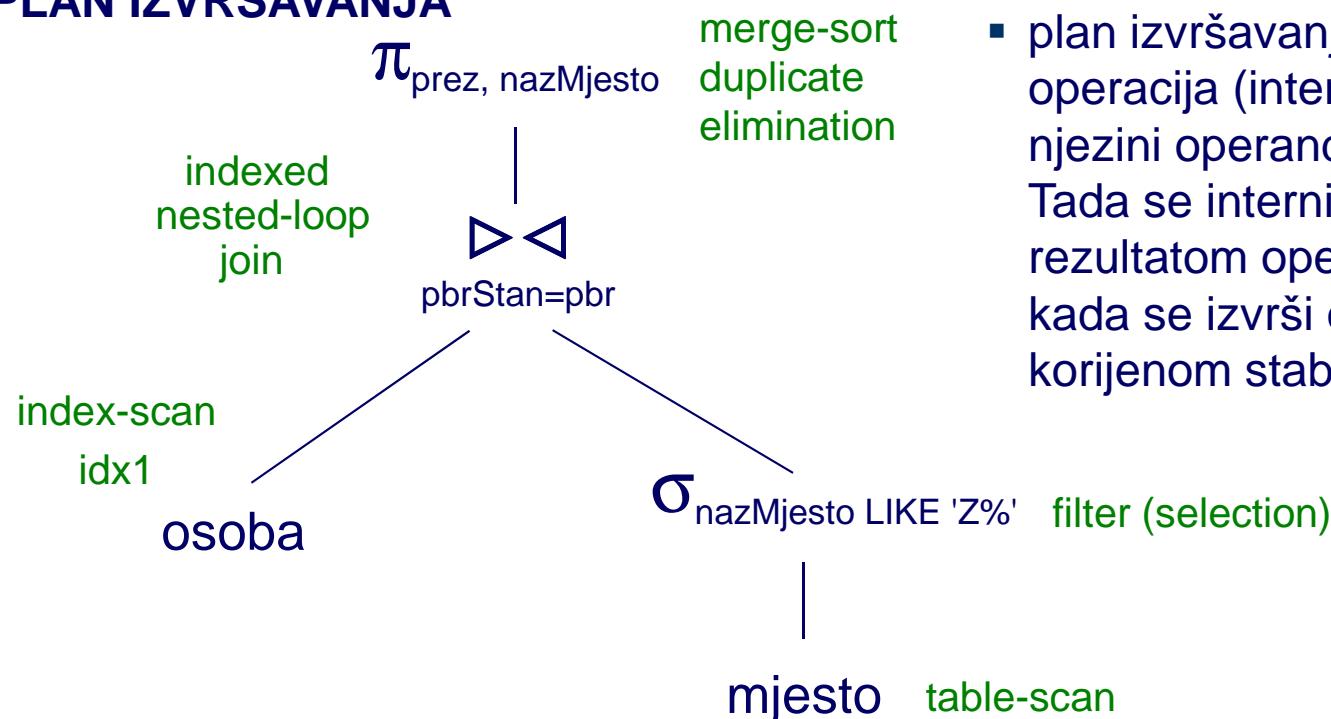


- stablo upita (*query tree*) je način prikaza izraza relacijske algebre
- listovi stabla su relacije, a interni čvorovi su operacije relacijske algebre

Stablo upita → Plan izvršavanja upita

- plan izvršavanja upita (*execution plan, execution strategy*) obuhvaća
 - stablo upita: operande i operacije, redoslijed njihovog izvršavanja
 - fizičke metode izvršavanja algebarskih operacija, odnosno fizičke operatore (*physical operators*)
 - index-scan, table-scan, nested-loop join, merge join, hash join, ...*
 - način proslijđivanja međurezultata

PLAN IZVRŠAVANJA



plan izvršavanja upita se evaluira tako da se operacija (interni čvor) izvršava onda kada njezini operandi (djeca) postanu raspoloživi. Tada se interni čvor zamjenjuje relacijom - rezultatom operacije. Izvršavanje završava kada se izvrši operacija predstavljena korijenom stabla

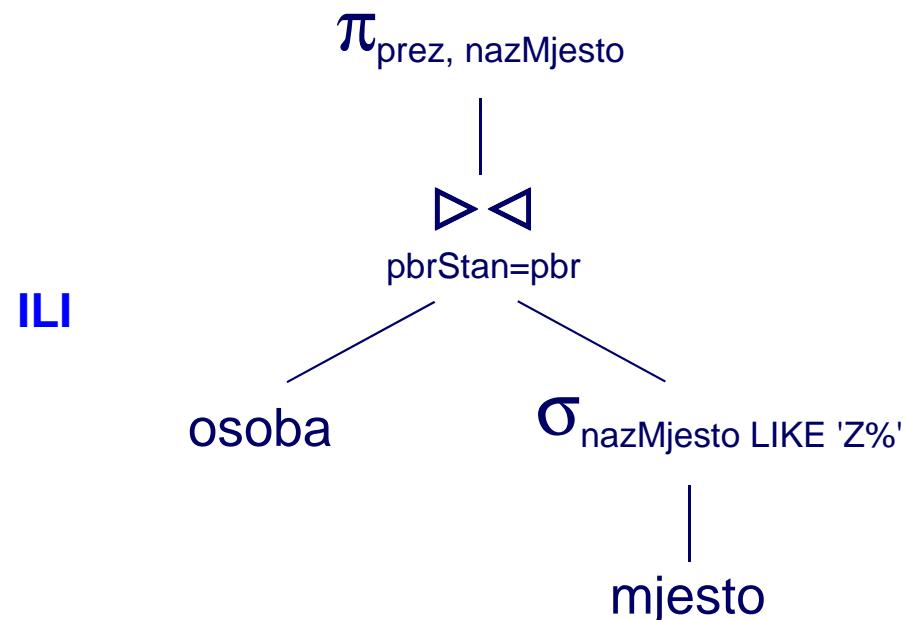
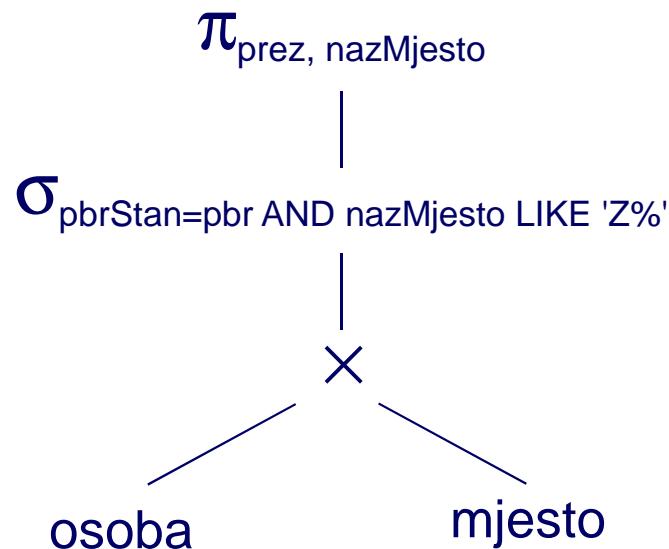
- na slici, zbog preglednosti, nisu opisani načini proslijđivanja međurezultata

Plan izvršavanja nije jednoznačno određen upitom

1. operandi i operacije relacijske algebre, redoslijed njihovog izvršavanja

- u općem slučaju se izraz relacijske algebre može zamijeniti ekvivalentnim, alternativnim izrazom relacijske algebre (jednim od mnogih)
 - dva izraza relacijske algebre su ekvivalentni ako primijenjeni nad svakom instancom baze podataka daju međusobno jednak rezultat
- alternativni izrazi (stabla upita) se određuju na temelju pravila za transformaciju izraza relacijske algebre

Primjer:

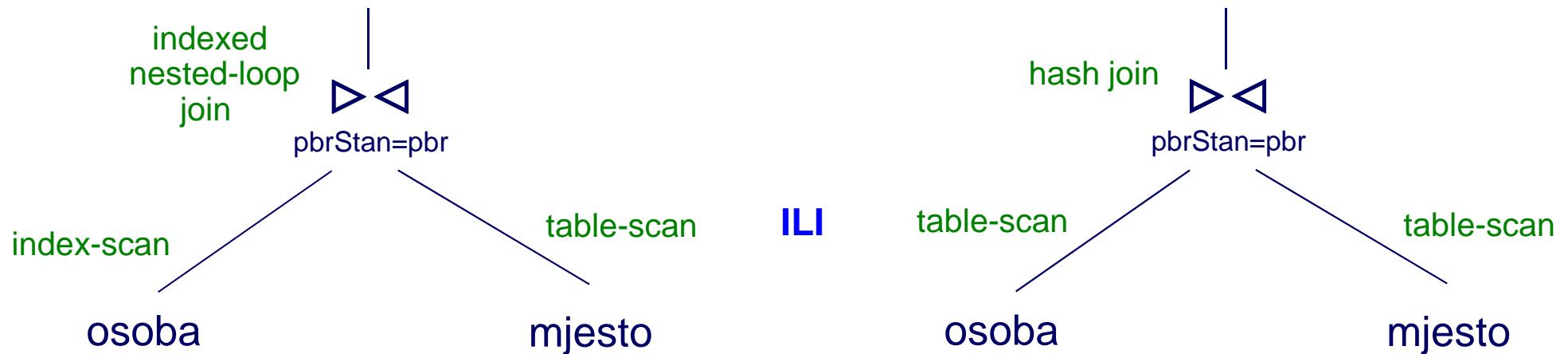


Plan izvršavanja nije jednoznačno određen upitom

2. fizičke metode izvršavanja algebarskih operacija

- SUBP implementira različite metode izvršavanja karakterističnih algebarskih operacija (selekcija, spajanje, projekcija, ...)
- mogućnost primjene pojedine metode ovisi o konkretno primijenjenoj fizičkoj organizaciji (npr. postoji li odgovarajući indeks u konkretnoj relaciji)
- razmatraju se alternativne metode izvršavanja algebarskih operacija, njihova primjenjivost i efikasnost

Primjer:



3. način proslijđivanja međurezultata (kasnije)

Optimizacija

⇒ za jedan upit postoji više (potencijalno mnogo) alternativnih planova izvršavanja

- izvršavanjem "lošeg" plana (umjesto "dobrog" plana), utrošak resursa se može povećati za nekoliko redova veličine
- proces odabira *najprikladnijeg* plana izvršavanja naziva se optimizacija upita (*query optimization*)
 - donekle pogrešan termin: odabrani plan nije uvijek optimalan
 - određivanje optimalnog plana moglo bi biti vremenski prezahtjevno ili neizvedivo zbog nedostatka potrebnih informacija
 - cilj "optimizacije" upita jest odabrati *razumno efikasan* plan izvršavanja - stoga bi bolji termin bio "planiranje strategije izvršavanja"

Dinamička i staticka optimizacija upita

- *dynamic query optimization*
 - optimizacija se provodi svaki puta kada se pokrene izvršavanje upita
 - dobro: informacije na temelju kojih se provodi optimizacija su ažurne
 - loše: veći utrošak resursa za postupke optimizacije
- *static query optimization*
 - upit se optimizira jednom, generirani plan izvršavanja ("izvršni kôd") se pohranjuje, te se koristi pri pokretanju svakog "sličnog" upita
 - neovisno o SQL sjednici
 - dobro: optimizacija se obavlja relativno rijetko pa se može provesti uz razmatranje većeg broja alternativnih planova
 - loše: podaci na temelju kojih je provedena optimizacija (statistički podaci) se mijenjaju tijekom vremena. U jednom trenutku dobar plan može u promijenjenim okolnostima postati loš
- *hybrid approach*
 - koristi se staticka optimizacija, ali sustav ponavlja postupak optimizacije upita kada se statistički podaci značajnije promijene

Fizički operatori

- implementacija i procjena troškova

Implementacija operacija relacijske algebre

- fizički operator je konkretna implementacija (u SUBP-u) algoritama za obavljanje operacija relacijske algebre:
 - selekcija, spajanje, Kartezijev produkt, projekcija
 - grupiranje i agregatne funkcije
 - unija, presjek, razlika
- za obavljanje dodatnih operacija iz SQL jezika:
 - eliminacija duplikata
 - sortiranje
- ostalih fizičkih operacija, npr:
 - čitanje n-torki slijednim čitanjem blokova podataka (*table-scan*)
 - čitanje n-torki uz korištenje indeksa (*index-scan*)
- koriste se i operatori koji kombiniraju neke od prethodno spomenutih operatora, npr:
 - sortirano čitanje svih zapisa relacije (*sort-scan*)

Selekcija

- način obavljanja operacije ovisi o uvjetu selekcije (*filter*) i raspoloživim metodama pristupa (*access methods*)
 - *filter* - u ovom području često korišten alternativni pojam za uvjet selekcije

Jednostavna selekcija

$\sigma_F(r)$ gdje je F jednostavna formula oblika $a \theta c$

- a atribut
- θ je operacija iz skupa $\{ <, \leq, >, \geq, = \}$
- c je konstanta

Mogući načini obavljanja

1. linearna pretraga (*table-scan, sequential scan, linear search, brute force*) za čitanje svih n-torki uz izdvajanje onih koje zadovoljavaju formulu F
2. indeks za dohvat skupa n-torki prema formuli $a \theta c$ ako za atribut a postoji indeks

Indeks se također može koristiti za formule oblika *range query*: $a \text{ BETWEEN } c1 \text{ AND } c2$

Jednostavna selekcija pomoću indeksa

- a) čitanje n-torki pomoću indeksa (B-stabla)
 - *index-scan*
 - indeks se koristi za pronalaženje n-torki koje zadovoljavaju uvjet selekcije, n-torka (ili dio n-torke) se dohvaća iz bloka s podacima
- b) čitanje vrijednosti ključeva iz listova indeksa (B-stabla)
 - *key-only index-scan*
 - indeks se koristi za pronalaženje n-torki koje zadovoljavaju uvjet selekcije, pri čemu su vrijednosti koje treba pročitati samo one koje se nalaze u listovima B-stabla. To znači da nije potrebno dohvaćati n-torke ili dijelove n-torki iz blokova s podacima

Jednostavna selekcija pomoću indeksa

c) *index self-join*

- složeni indeks (A, B, C) se (ipak) može koristiti i za uvjete selekcije oblika $B=x$
 - ali samo ako su vrijednosti atributa A slabo raspršene

Primjer:

```
CREATE INDEX idxOso ON osoba (spol, sifra);
SELECT * FROM osoba
    WHERE sifra BETWEEN 20 AND 30;
```

- SUBP će u prvom čitanju indeksa dohvatiti različite vrijednosti v atributa **spol**, a zatim po jednom, za svaku dohvaćenu vrijednost $v \in \{ 'M' , 'Ž' \}$ obaviti:

```
SELECT * FROM osoba
    WHERE spol = v AND sifra BETWEEN 20 AND 30;
```
- ako vrijednosti atributa A nisu slabo raspršene, index self-join se ne koristi!

osoba	
spol	sifra
Ž	1
Ž	14
Ž	15
Ž	19
Ž	21
Ž	24
Ž	37
Ž	59
M	12
M	23
M	64
M	72
M	83
M	91
M	94

Selekcija

Složena selekcija - konjunktivna forma

- Uvjet selekcije je oblika $F_1 \wedge F_2 \wedge F_3 \dots$, gdje su F_i jednostavne formule
- 1. konjunktivna selekcija korištenjem jednog indeksa
 - ako se za neki F_i može primijeniti indeks, tada se pomoću indeksa dohvate n-torce koji zadovoljavaju F_i , a nad svakom tako dohvaćenom n-torkom primjenjuje se filter sastavljen od preostalih izraza $F_1 \wedge F_2 \dots$
 - ako su na raspolaganju indeksi koji se mogu koristiti za više od jedne formule F_i , odabire se onaj F_i (i koristi pripadajući indeks) kojim će se dohvatiti najmanji broj n-torki
- 2. konjunktivna selekcija korištenjem složenog (kompozitnog) indeksa
 - ako se u konjunktivnom obliku formule s relacijskim operatorima usporedbe ($<$, \leq , $>$, \geq , $=$) koriste atributi za koje postoji složeni indeks, npr.
 - ako postoji index `idx(a,b,c)`, može se iskoristiti za predikat:
 - $a = 7 \text{ AND } b > 'P' \text{ AND } c \leq 110.0$
- 3. linearna pretraga

Selekcija

Složena selekcija - konjunktivna forma

Primjer:

```
SELECT * FROM ispit
  WHERE ocjena = 3
    AND sifPred = 201
    AND datIspit = TODAY
```

- broj n-torki u relaciji 100000
- broj različitih predmeta 100
- broj različitih ocjena 5

- moguće je koristiti složeni indeks (**sifPred**, **ocjena**) + filter
- moguće je koristiti indeks za **sifPred** + filter
- moguće je koristiti indeks za **ocjena** + filter
- ako postoji indeks za **sifpred** i indeks za **ocjena**, koristi se indeks za **sifPred** + filter
 - zašto?
- moguće je koristiti kompozitni indeks za (**sifpred**, **datIspit**) + filter
- moguće je koristiti kompozitni indeks za (**datIspit**, **ocjena**, **sifPred**) + filter

Selekcija

Složena selekcija - disjunktivna forma (disjunktivna selekcija)

- Uvjet selekcije je oblika $F_1 \vee F_2 \vee F_3 \dots$, gdje su F_i jednostavne formule
 - samo u slučaju ako postoji indeks za svaki F_i , zasebno se dohvaća po jedan skup identifikatora n-torki za svaki uvjet F_i uz korištenje pripadnog indeksa. Rezultat se dobije unijom dobivenih skupova
 - inače, linearna pretraga

zašto se isti princip ne primjenjuje za složenu selekciju u konjunktivnoj formi?

Selekcija

Složena selekcija - disjunktivna forma (disjunktivna selekcija)

Primjer:

```
SELECT * FROM ispit
WHERE datIspit = TODAY
OR sifPred = 201
```

- ako postoji samo kompozitni indeks (**sifPred , datIspit**), koristi se linearna pretraga
- ako postoji samo indeks za **sifPred** ili samo indeks za **datIspit**, linearna pretraga!
- ako postoji indeks za **sifpred** i indeks za **datIspit**, dohvaća se skup identifikatora n-torki za koje je **datIspit=TODAY**, skup identifikatora n-torki za koje je **sifPred=201**, nakon čega se obavlja unija tih skupova
 - u ovakvim slučajevima, ako su skupovi veliki, optimizator može ipak odabrat linearu pretragu (→vidjeti primjer u domaćoj zadaći)

Statistički podaci iz rječnika podataka

- $B(r)$ - broj blokova relacije r
 - $N(r)$ - broj n -torki relacije r
 - $d(idx)$ - dubina B-stabla za indeks idx
 - $V(A, r)$ - broj različitih vrijednosti atributa A u relaciji r
 - $V(A, r) = G_{COUNT(*)}(\pi_A(r))$
 - A može biti skup atributa
 - $\min(A, r), \max(A, r)$ - najmanja i najveća vrijednost atributa A u relaciji r
 - umjesto toga, često: *second-largest and second-smallest value of A*
 - "ekstremne" vrijednosti se odbacuju - zašto?
 - histogrami
-
- SUBP ne ažurira statističke podatke pri svakoj promjeni podataka
 - u vremenu smanjenog opterećenja sustava, nakon većih promjena u relaciji
 - IBM IDS: administrator obavlja naredbu

UPDATE STATISTICS FOR TABLE r

Selekcija - procjena troškova

- procjena troška: procjenjuje se broj U/I operacija koje će biti potrebno obaviti u najlošijem slučaju
- bez indeksa, linearna pretraga: $B(r)$
- pomoću indeksa idx (B-stabla)
 - jedna n-torka (preko ključa relacije): $d(idx) + 1$
 - više n-torki
 - *clustered index*: $d(idx) + \text{broj blokova u kojima se nalaze tražene n-torke}$
 - *non-clustered index*: vrijednosti u listovima indeksa jesu sortirane, ali tražene n-torke mogu biti znatno raspršene među blokovima s podacima. Ekstremni slučaj: za dohvrat svake n-torke potrebna je jedna U/I operacija. Ako je broj n-torki koje se dohvaćaju velik, trošak može biti i veći nego trošak linearne pretrage pa se tada indeks ne koristi

I u slučajevima kada indeks nije "CLUSTERED", SUBP u rječniku podataka može imati podatak o "stupnju grupiranja" (*degree of clustering*). Npr. IBM IDS:

```
SELECT idxname, clust FROM sysindices WHERE idxname = 'idx1';
```

Manji brojevi znače bolje grupiranje (više sliči *clustered* indeksu), veći brojevi znače veće raspršenje

Sortiranje

- sortiranje je jedna od temeljnih fizičkih operacija
 - zbog ORDER BY u SQL naredbi
 - zbog fizičkih operatora koji zahtijevaju da jedan ili oba argumenta budu sortirani (vidjeti kasnije npr. *sort-merge join*)
- ulazni argument je relacija $r(R)$ i lista atributa $L \subseteq R$ prema kojima treba obaviti sortiranje
 - ako postoji odgovarajući indeks (B-stablo)
 - tada se pomoću *index-scan* dobije r sortiran prema L
 - operacija se naziva sortiranje tijekom čitanja (*sort-scan*)
 - **NE KORISTI SE UVIJEK:** ako je stupanj grupiranja slab, *table-scan* + sortiranje u glavnoj memoriji može biti bolji izbor
 - inače, ako se $r(R)$ može odjednom smjestiti u raspoloživi segment glavne memorije
 - tada se obavlja *table-scan* te se pročitana relacija sortira u glavnoj memoriji. Koriste se uobičajeni algoritmi za sortiranje, npr. *quicksort*
 - inače (relacija $r(R)$ je prevelika)
 - koristi se vanjsko sortiranje (*external sort*)

```
SELECT * FROM osoba  
ORDER BY prezime
```

Sortiranje

- vanjsko sortiranje
 - koristi se za sortiranje relacija koje se ne mogu odjednom smjestiti u raspoloživi segment glavne memorije (segment glavne memorije kojeg proces ima na raspolaganju za obavljanje operacije sortiranja)
 - najčešće korištena metoda je vanjsko sortiranje s uparivanjem: *external sort-merge*
 - obavlja se u dvije faze
 - faza sortiranja
 - faza uparivanja

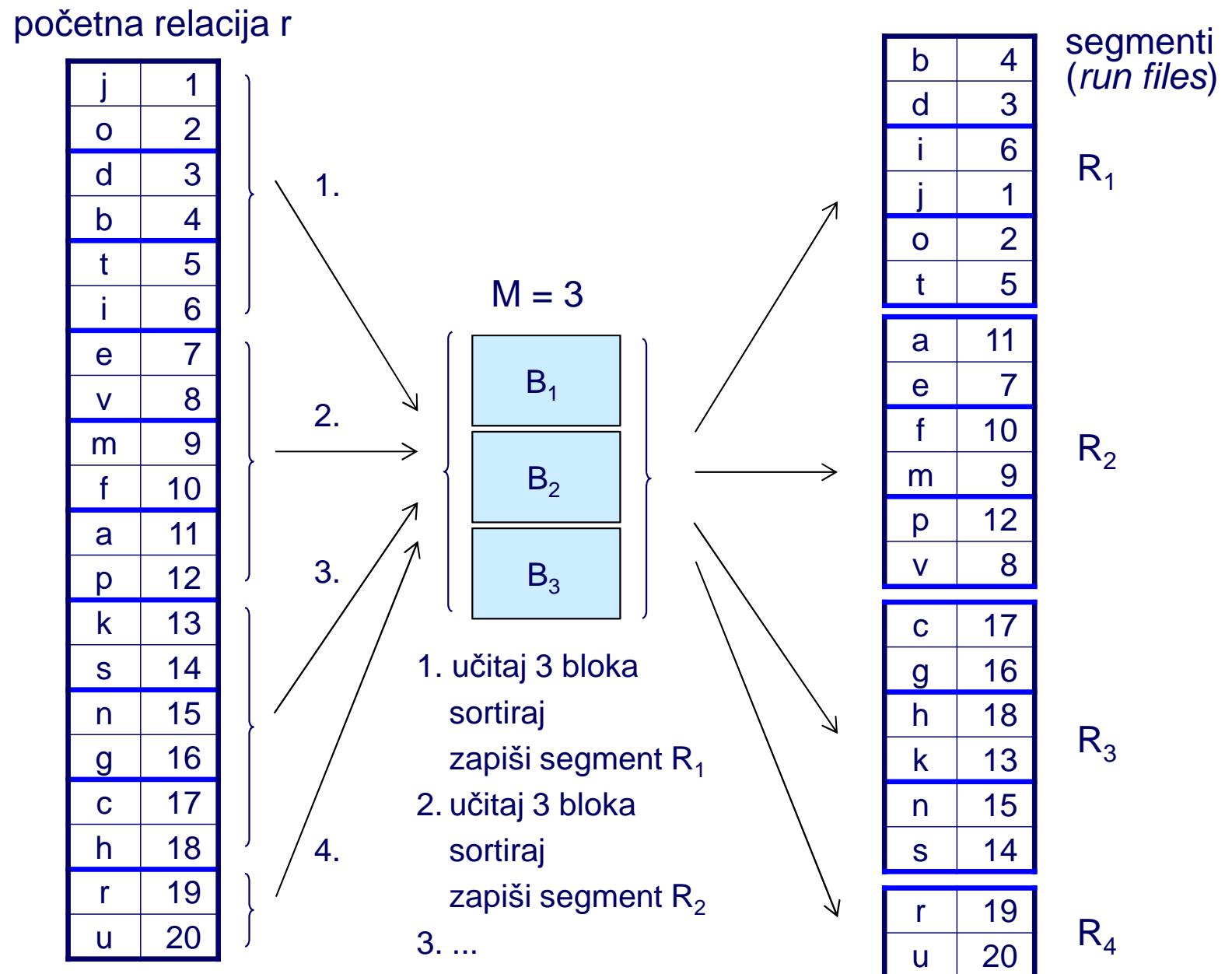
External sort-merge - 1. faza - sorting phase

- $B(r)$ - broj blokova relacije r
- M - raspoloživi broj blokova glavne memorije
- relacija r se dijeli na N segmenata veličine M , $N = \lceil B(r) / M \rceil$
 - svaki segment (*run file*) se sortira zasebno i zapisuje u postojanu memoriju

```
i = 0
repeat
    read M blocks of r or the rest of the r
    sort the in-memory part
    write the sorted data to run file Ri
    i = i + 1
until the end of the r
```

External sort-merge - 1. faza - sorting phase

Primjer:
zapisa u bloku = 2
 $B(r) = 10$
 $M = 3$



External sort-merge - 2. faza - merging phase

- uparivanje sortiranih segmenata
- odjednom se uparuju zapisi iz d_m segmenata (d_m je *degree of merging*)
- $d_m = \min(N, M-1)$
- ako je $N < M$, tada je $d_m = N$, te je za uparivanje dovoljan jedan korak
- ako je $N \geq M$, uparivanje je potrebno provesti u više koraka
 - uparuje se po $M-1$ segmenata. Time se broj segmenata u svakom koraku reducira za faktor $M-1$
 - postupak se ponavlja dok broj segmenata ne bude manji od M

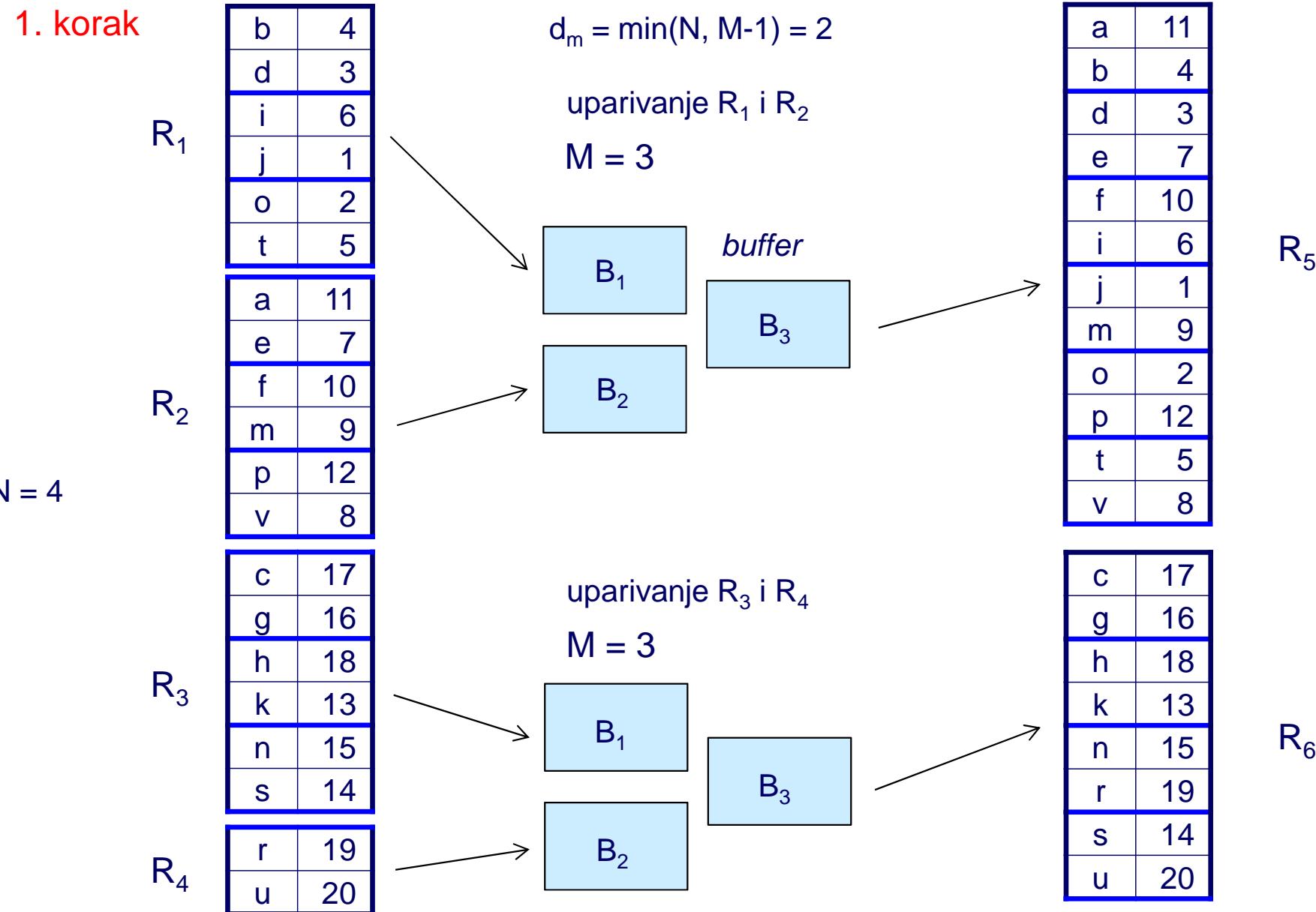
External sort-merge - 2. faza - merging phase

- u glavnoj memoriji alocirati po jedan ulazni blok za svaki od segmenata + jedan izlazni blok za (*buffered!*) zapisivanje rezultata, ukupno ne više od M blokova

```
read first block of each run  $R_i$  into main memory block  $B_i$ 
repeat
    choose the first tuple (in sort order) among all  $B_i$ 
    write the tuple to the output* and delete it from  $B_i$ 
    if  $B_i$  is empty and not end-of-file( $R_i$ )
        read the next block of  $R_i$  into  $B_i$ 
until all blocks  $B_i$  are empty
```

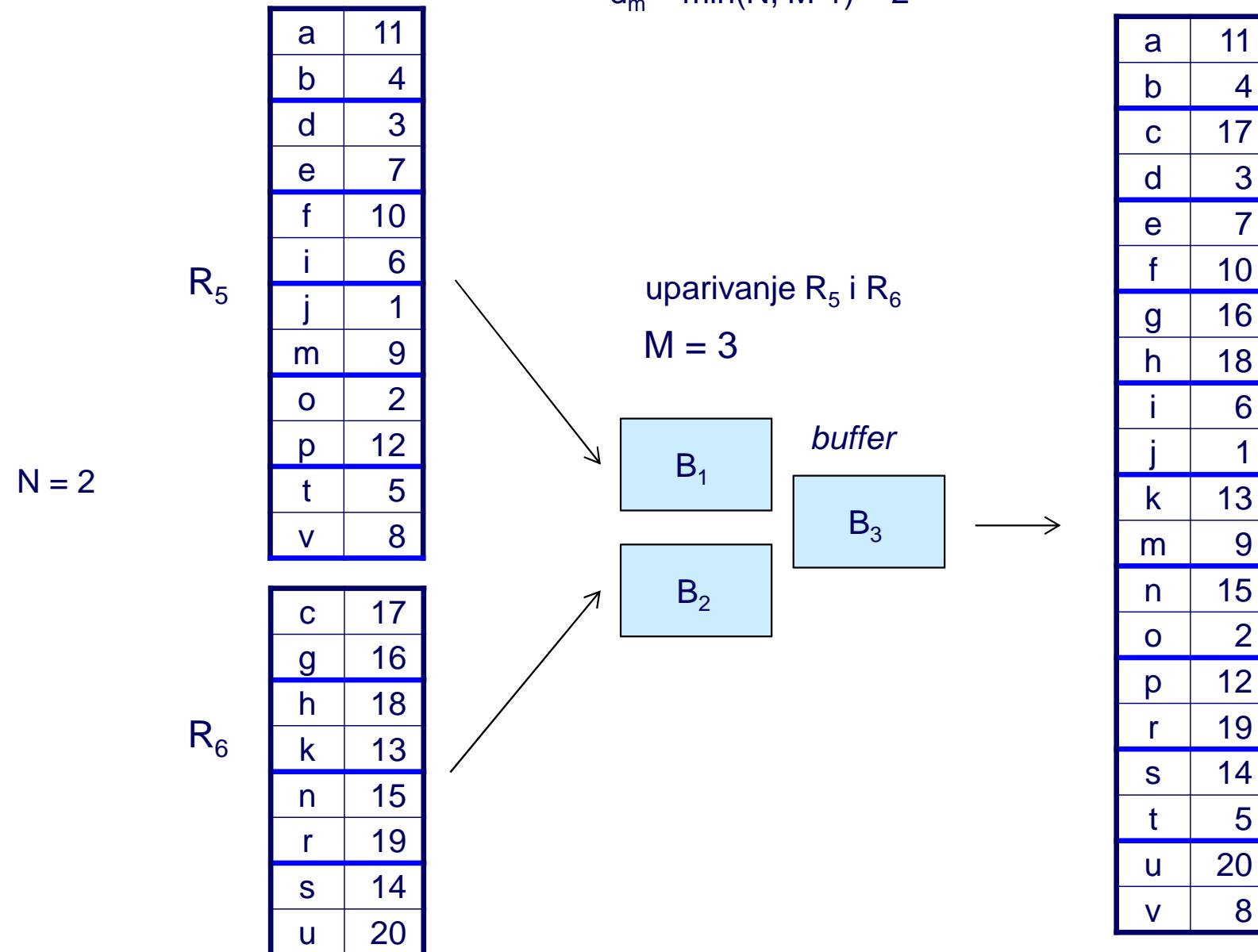
* *write the tuple to the output*: zapisivanje u postojanu memoriju (uz korištenje međuspremnika)
ili prosljeđivanje na ulaz u sljedeću operaciju (*pipelining*)

External sort-merge - 2. faza - merging phase



External sort-merge - 2. faza - merging phase

2. korak



External sort-merge - procjena troškova

- svaki korak druge faze (uparivanje) zahtijeva $2 B(r)$ U/I operacija
 - $B(r)$ *input* + $B(r)$ *output* operacija
- u prvom koraku druge faze obrađuje se $\lceil B(r) / M \rceil$ segmenata
- u drugom koraku obrađuje se $\lceil B(r) / M \rceil / (M-1)$ segmenata
- u i -tom koraku obrađuje se $\lceil B(r) / M \rceil / (M-1)^{i-1}$ segmenata
- u zadnjem koraku ne smije biti više od $M-1$ segmenata
 - $\lceil B(r) / M \rceil / (M-1)^{i-1} \leq M-1$
 - $i \geq \log_{M-1} \lceil B(r) / M \rceil$
 - \Rightarrow potreban broj koraka u drugoj fazi: $\lceil \log_{M-1} \lceil B(r) / M \rceil \rceil$
- ukupan broj U/I operacija za 2. fazu: $2 B(r) \lceil \log_{M-1} \lceil B(r) / M \rceil \rceil$
- broj U/I operacija za 1. fazu (sortiranje): $2 B(r)$
- ukupan broj potrebnih U/I operacija: $2 B(r) + 2 B(r) \lceil \log_{M-1} \lceil B(r) / M \rceil \rceil$
 - uz uključen trošak zapisivanja konačnog rezultata, inače za $B(r)$ manje!
 - ako se koristi *pipelining*, ne postoji trošak zapisivanja konačnog rezultata
- u prethodnom primjeru: broj potrebnih U/I operacija = 60 (uz uračunati trošak zapisivanja rezultata)

Spajanje

1. spajanje s ugniježđenim petljama: *nested-loop join*
2. blokovsko spajanje s ugniježđenim petljama: *block nested-loop join*
3. indeksirano spajanje s ugniježđenim petljama: *indexed nested-loop join*
4. spajanje uparivanjem sortiranih relacija: *sort-merge join*
5. spajanje raspršenim adresiranjem: *hash join*

Nested-loop join

- par ugniježdenih petlji s unaprijed poznatim brojem ponavljanja (for petlji)

```
r >< S  
F  
for each tr in r  
    for each ts in s  
        if F(tr, ts)  
            add tr·ts to the result
```

vanjska relacija (*outer relation*)

unutarnja relacija (*inner relation*)

- veliki trošak (uspoređuje se $N(r) \cdot N(s)$ n-torki)
- najlošiji slučaj: samo po jedan blok svake relacije stanu u glavnu memoriju
 - procjena troška: $N(r) \cdot B(s) + B(r)$ (nisu uključeni troškovi zapisivanja rezultata)
- najbolji slučaj: obje relacije ili samo manja relacija (manja relacija se tada koristi kao unutarnja) stanu u glavnu memoriju
 - procjena troška: $B(r) + B(s)$ (nisu uključeni troškovi zapisivanja rezultata)

$t_r \cdot t_s$: n-torka dobivena ulančavanjem n-torki t_r i t_s

$N(r)$, $N(s)$: broj n-torki u relacijama r i s

$B(r)$, $B(s)$: broj blokova u relacijama r i s

Block nested-loop join

```
for each block B(r) in r
    for each block B(s) in s
        for each tr in B(r)
            for each ts in B(s)
                if F(tr, ts)
                    add tr·ts to the result
```

- najlošiji slučaj: samo po jedan blok svake relacije stanu u glavnu memoriju
 - procjena troška: $B(r) \cdot B(s) + B(r)$ (nisu uključeni troškovi zapisivanja rezultata)
- najbolji slučaj: (jednako kao u *nested-loop join*)
 - procjena troška: $B(r) + B(s)$ (nisu uključeni troškovi zapisivanja rezultata)

Ako je za operaciju spajanja na raspolaganju glavna memorija veličine M blokova

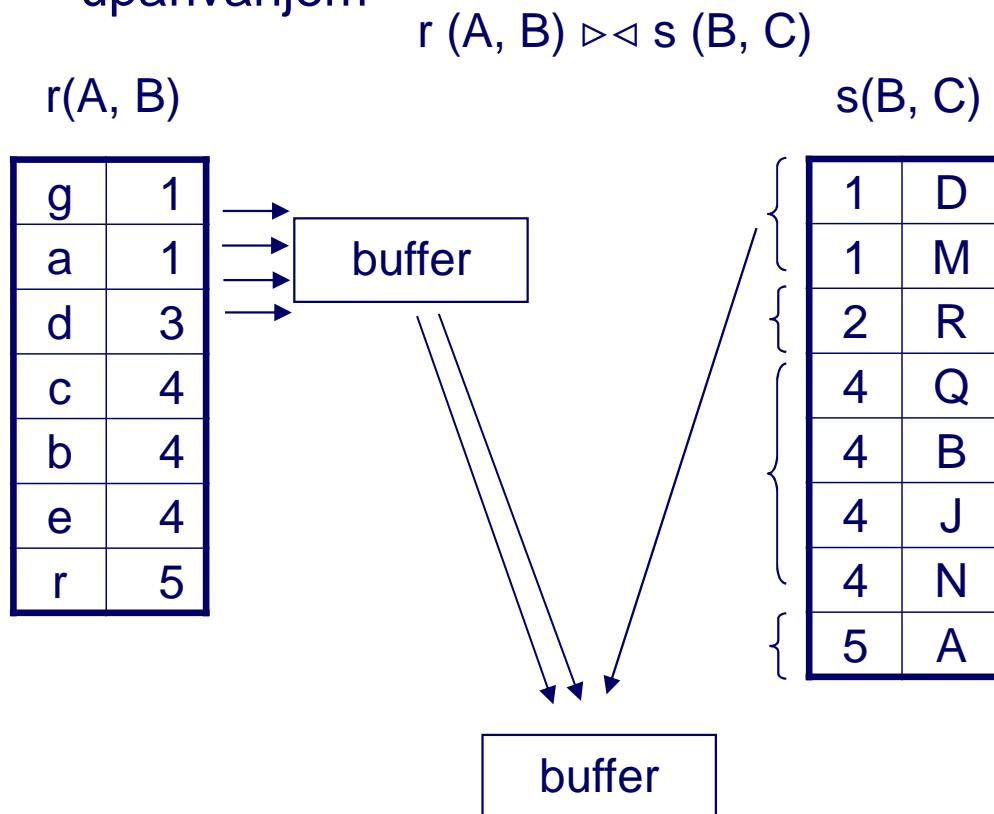
- 1 blok za čitanje unutarnje relacije, 1 blok za izlazni međuspremnik
- vanjska petlja: umjesto po jedan blok, $B(r) / (M - 2)$ puta učitava po M-2 blokova
- procjena troška: $B(r) / (M - 2) \cdot B(s) + B(r)$
(nisu uključeni troškovi zapisivanja rezultata)

Indexed nested-loop join

- preuvjet: na raspolaganju je indeks nad unutarnjom relacijom za atrIBUTE prema kojima se obavlja spajanje. Može se koristiti:
 - indeks koji već postoji nad unutarnjom relacijom
 - indeks koji se nad unutarnjom relacijom kreira samo za potrebe izvršavanja tog upita: *autoindex*
 - *autoindex* se koristi onda kada se više isplati kreirati indeks nego obavljati po jedan *table-scan* za svaku n-torku iz r
 - za svaku t_r iz r, pomoću indeksa se pronađe odgovarajuće n-torce iz s
 - najlošiji slučaj: samo po jedan blok svake relacije stanu u glavnu memoriju
 - procjena troška: $N(r) \cdot c + B(r)$ (nisu uključeni troškovi zapisivanja rezultata)
 - što ako obje relacije imaju indeksirane atrIBUTE prema kojima se obavlja spajanje?
 - u općem slučaju bolje je za vanjsku relaciju odabrati onu s manje n-torki
- c : broj U/I operacija za dohvrat n-torke iz s pomoću indeksa (~ dubina stabla)

Sort-merge join

- može se koristiti za prirodno spajanje i spajanje uz uvjet s izjednačavanjem (*equi-join*) i spajanje uz uvjet na temelju operatora $>$, $<$, \leq , \geq
- r i s su sortirane prema atributima prema kojima se obavlja spajanje
- postupak spajanja je sličan drugoj fazi u algoritmu za vanjsko sortiranje s uparivanjem



- grupe n-torki relacije s koje imaju jednake vrijednosti atributa prema kojima se obavlja spajanje prebacuju se u međuspremnik
- n-torce iz r koje imaju odgovarajuće vrijednosti atributa spajaju se sa svakom n-torkom koja se nalazi u međuspremniku

Sort-merge join

- postupak ilustriran prethodnom slikom podrazumijeva da se svaka grupa n-torki iz relacije s (grupa s jednakim vrijednostima atributa prema kojima se obavlja spajanje) može odjednom smjestiti u međuspremnik. Uz tu pretpostavku:
 - procjena troška: $B(r) + B(s)$ (nisu uključeni troškovi zapisivanja rezultata)
 - ako relacije nisu unaprijed sortirane, dodati i broj U/I operacija potrebnih za sortiranje relacija r i s (vidjeti *external sort-merge*)
- ako se neka grupa n-torki iz relacije s ne može odjednom smjestiti u međuspremnik, broj U/I operacija će se povećati jer će se spajanje n-torki iz relacije r s n-torkama takve grupe morati obaviti pomoću *nested-loop join* postupka

Hash join

- može se koristiti za prirodno spajanje i spajanje uz uvjet s izjednačavanjem (equi-join)
 $r(A, B) \bowtie s(B, C)$
- funkcija raspršenja h preslikava B-vrijednost n-torke u adresu pretinca (*bucket*)
 - adresa pretinca = $h(t(B))$
 - B ne mora biti samo jedan atribut, može biti skup atributa
- *build phase*
 - n-torce t_s relacije s raspršuju se u pretince, adresa pretinca = $h(t_s(B))$
- *probe phase*
 - za svaku n-torku t_r relacije r izračunava se adresa pretinca $h(t_r(B))$
 - ako je $t_r(B) = t_s(B)$, dodaj $t_r \cdot t_s$ u rezultat
 - nužna provjera jer funkcija raspršenja za različite $t(B)$ može izračunati istu adresu pretinca
- ako se relacija s može odjednom smjestiti u glavnu memoriju
 - procjena troška: $B(r) + B(s)$ (nisu uključeni troškovi zapisivanja rezultata)

Hash join

- ako se relacija s ne može smjestiti u glavnu memoriju \Rightarrow *partitioned hash join*
- *partitioning phase*: primjenom funkcije raspršenja $h(t(B))$, zasebno se particioniraju relacije r i s . U sekundarnoj memoriji nastaju particije:
 - r_1, r_2, \dots, r_M
 - s_1, s_2, \dots, s_M
- za postupak kreiranja M particija, najmanja potrebna veličina glavne memorije je $M+1$ blok
 - jedan blok u funkciji ulaznog međuspremnika (za čitanje relacije), te po jedan blok za svaku particiju (u funkciji izlaznog međuspremnika za i -tu particiju)
- *probing phase*: za svaki $i = 1..M$, obavlja se spajanje n -torki iz particija r_i, s_i
 - npr. primjenom *nested-loop*: manja od dviju particija iz para smješta se u glavnu memoriju
 - druga mogućnost: za manju od dviju particija kreira se *in-memory* tablica raspršenja (*hash-table*)
- procjena troška: $3 * (B(r) + B(s))$ (nisu uključeni troškovi zapisivanja rezultata)
 - za vježbu objasniti zašto. Također, uočiti da će trošak biti veći od navedenog u slučaju kada neka particija ne stane u međuspremnik (tada se za spajanje tih particija koristi *blocked nested-loop join*)

Usporedba: *Hash join* i (*auto-index*) *nested-loop join*

- *hash join* u općem slučaju brže evaluira ukupan rezultat
- *nested-loop* brže evaluira "prve n-torke"
 - koristi se ako je važno da korisnik čim prije dobije barem dio rezultata
- *hash join* je efikasniji, ali se može koristiti samo za "equi-join" uvjete spajanja

Primjer:

linija	
let	udaljenost
CA-825	700
LH-412	4800
BA-722	15000
CA-311	13000

zrakoplov	
tip	dolet
B747	13000
A320	5400
DC-9	3100

- sljedeća operacija spajanja se ne može obaviti pomoću raspršenog adresiranja

```
SELECT *
  FROM linija JOIN zrakoplov
    ON dolet >= udaljenost;
```

Vanjsko spajanje

$$r(A, B) \underset{F}{\ast\rhd\lhd} s(B, C)$$

- koristi se modificirani oblik jednog od prethodno opisanih algoritama
- npr. modificira se *nested-loop join*
 - u slučaju lijevog vanjskog spajanja, lijeva relacija mora se koristiti kao vanjska relacija
 - ako se za t_r iz vanjske relacije r ne pronađe odgovarajuća t_s iz unutarnje relacije s , u rezultat se dodaje $t_r \cdot t_{NULL}$ (gdje je t_{NULL} n-torka čije su vrijednosti atributa NULL)
- prirodno vanjsko spajanje (*natural outer join*) i vanjsko spajanje s izjednačavanjem (*outer join with equi-join condition*) se (osim pomoću *nested-loop*) također mogu realizirati modificiranim oblicima *sort-merge join* i *hash-join* algoritama
- realizacija operacije *full outer join* je kompleksnija!

Eliminacija duplikata

$\delta(r)$

- pomoću *external sort-merge*
 - već za vrijeme faze sortiranja, eliminiraju se duplikati pronađeni unutar istog segmenta (smanjuje se inicijalna veličina segmenata)
 - preostali duplikati se jednostavno eliminiraju tijekom faze uparivanja
 - procjena troška: jednak kao kod sortiranja
- pomoću raspršenog adresiranja
 - primjenom funkcije raspršenja $h(t)$ nastanu particije r_1, r_2, \dots, r_M
 - za svaku particiju r_i provodi se postupak:
 - formira se nova *in-memory hash table*, pri čemu funkcija raspršenja mora biti drugačija od h (zašto?)
 - za svaku n-torku t_r iz r_i
 - t_r se dodaje u *hash table* samo ako se već ne nalazi u njoj
 - n-torke iz *hash table* se dodaju u rezultat
 - procjena troška: $3 * B(r)$ (nisu uključeni troškovi zapisivanja rezultata)

Projekcija

$$\pi_L(r) = \delta(\pi_L^B(r))$$

- bag verzija projekcije (izdvajanje atributa) je jednostavna
- za eliminaciju duplikata se koristi jedan od opisanih postupaka (*external sort-merge, hash*)
- procjena troška: jednaka kao kod operacije eliminacije duplikata

Agregacija i grupiranje

$A_1, A_2, \dots, A_m G_{\mathcal{AF}_1(B_1), \mathcal{AF}_2(B_2), \dots, \mathcal{AF}_n(B_n)}(r)$

Ako je rezultat prevelik za raspoloživi prostor glavne memorije

- postupak kao kod eliminacije duplikata (*sort-merge ili hash*)
 - umjesto eliminacije duplikata, za svaku grupu izračunavaju se agregatne funkcije
 - procjena troška(za *hash*): $3 * B(r)$ (nisu uključeni troškovi zapisivanja rezultata)

Ako se rezultat može pohraniti u raspoloživi prostor glavne memorije

- rezultat se pohranjuje u *in-memory hash table*
- za svaku n-torku s ključem A_1, A_2, \dots, A_m
 - ako se ne nalazi u *in-memory hash table*
 - dodati n-torku $A_1, A_2, \dots, A_m, \mathcal{AF}_1, \mathcal{AF}_2, \dots, \mathcal{AF}_n$
 - izračunati nove vrijednosti $\mathcal{AF}_1, \mathcal{AF}_2, \dots, \mathcal{AF}_n$
 - procjena troška: $B(r)$ (nisu uključeni troškovi zapisivanja rezultata)

Unija, presjek, razlika

- pomoću external sort-merge
 - obje relacije sortirati po svim atributima
 - uparivanjem
 - ∪ ■ n-torku iz r ili s dodati u rezultat, preskočiti sve jednake n-torke u obje relacije
 - ∩ ■ n-torku iz r koja se nalazi i u s dodati u rezultat, preskočiti sve jednake n-torke u obje relacije
 - \ ■ n-torku koja se nalazi u r, a ne nalazi se u s, dodati u rezultat, preskočiti sve jednake n-torke u obje relacije
- procjena troška:
 - ako su r i s sortirane
 - $B(r) + B(s)$
 - ako r i s nisu sortirane
 - trošak sortiranja (r) + trošak sortiranja (s) + $B(r) + B(s)$

Unija, presjek, razlika

- pomoću raspršenog adresiranja
 - primjenom funkcije raspršenja $h(t)$ nastanu particije
 - $r_1, r_2, \dots, r_M, s_1, s_2, \dots, s_M$
 - ponavljati za svaki r_i
 - $r_i \rightarrow \text{in-memory hash table}$
- ▷ □ za svaku n-torku t_s iz s_i koja se već ne nalazi u *in-memory hash table*
 - dodati t_s u *in-memory hash table*
 - sadržaj *in-memory hash table* dodati u rezultat
- ∩ □ za svaku n-torku t_s iz s_i koja se nalazi u *in-memory hash table*
 - dodati t_s u rezultat
- \ □ za svaku n-torku t_s iz s_i koja se nalazi u *in-memory hash table*
 - izbaciti t_s iz *in-memory hash table*
 - sadržaj *in-memory hash table* dodati u rezultat

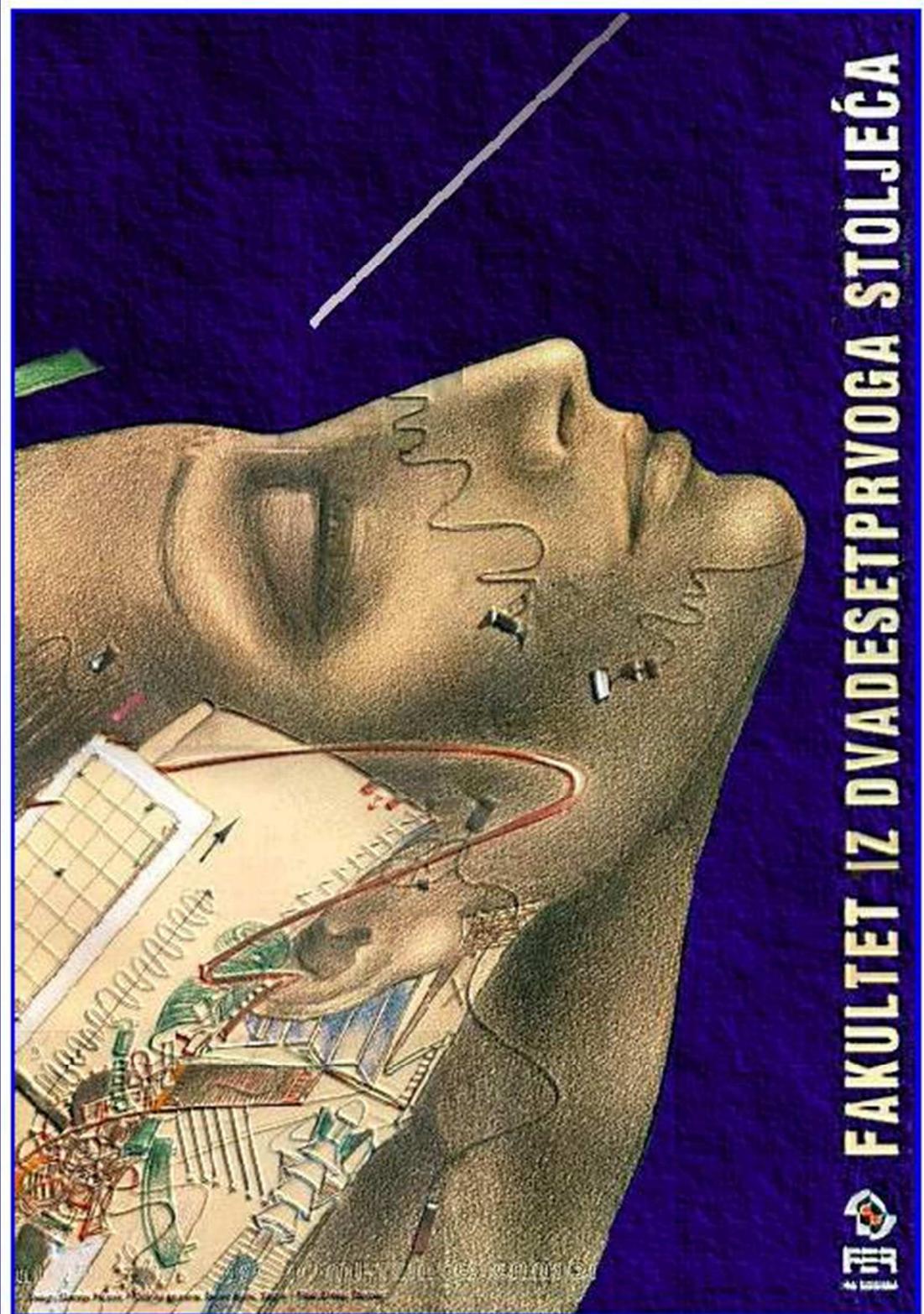
Literatura:

- H. Garcia-Molina, J. Ullman, J. Widom. Database system implementation. Prentice-Hall International, Inc. Upper Saddle River, New Jersey. 2000.
- A. Silberschatz, H. Korth, S. Sudarshan. Database system concepts, 4th ed. McGraw-Hill. 2002.
- R. Elmasri, S. Navathe. Fundamentals of database systems, 3rd ed. Addison-Wesley Publishing Company. Reading, Massachusetts. 2000.
- IBM Informix Dynamic Server Performance Guide, Version 11.50, IBM, 2008.

Sustavi baza podataka

Predavanja

4. Obavljanje upita
(2. dio)
ožujak 2014.



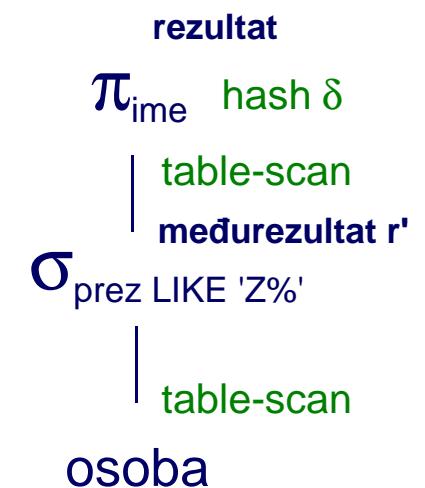
FAKULTET IZ DVADESETPRVOGA STOLJEĆA

Složeni izrazi relacijske algebre

- način prosljeđivanja međurezultata

Način prosljedivanja međurezultata

- rezultat izraza relacijske algebre evaluira se kao niz pojedinačnih operacija relacijske algebre
- osim redoslijeda obavljanja operacija i odabira fizičkih operatora, planom se treba definirati i način postupanja s rezultatima pojedinačnih operatora (međurezultatima) jer rezultat jednog operatora predstavlja ulazni argument sljedećeg operatora



A) Materijalizacija (*materialization*)

- međurezultat se kao cjelina pohranjuje
 - ili u međuspremniku
 - ili u sekundarnu memoriju (ako je međurezultat prevelik za raspoložive međuspremniku). Problem: trošak U/I operacija radi zapisivanja (a kasnije opet čitanja) međurezultata

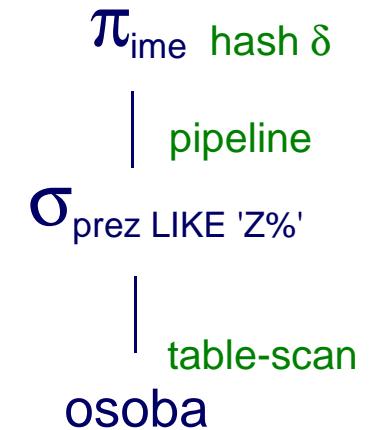
Primjer:

- obavi $\sigma_{\text{prez LIKE } 'Z\%'}$ i pohrani međurezultat r'
- obavi $\pi_{\text{ime}}(r')$

Način prosljeđivanja međurezultata

B) Cjevovodi (*pipelining, stream-based processing*)

- fizički operator s niže razine (proizvođač, *producer*) producira pojedinačne n-torke koje koristi operator na višoj razini (potrošač, *consumer*)
 - *demand-driven (pulling)* - češća implementacija
 - *producer-driven (pushing)* - redovi, sinkronizacija



Primjer (*pulling*):

- operator π zahtijeva od operadora σ sljedeću n-torku, obrađuje ju i prosljeđuje kao rezultat. Operator σ , na zahtjev za isporukom sljedeće n-torke, zahtijeva od operadora *table-scan* sljedeću n-torku, obrađuje ju i prosljeđuje operatoru π
- koji će se model (materijalizacija ili cjevovodi) koristiti u konkretnom slučaju najviše ovisi o obliku fizičkog operadora, npr.
 - *external sort-merge* nije naročito pogodan za primjenu u modelu cjevovoda: ne može producirati n-torke dok ne prihvati cijeli međurezultat s prethodne razine i započne s posljednjom fazom uparivanja
 - operator selekcije, spajanje s ugniježđenim petljama, *bag* verzija unije, primjeri su operadora pogodnih za primjenu u modelu cjevovoda

Prosljeđivanje rezultata i iteratori

- kod implementacije *pulling* modela, SUBP koristi poseban oblik sučelja: *iterator interface*
- time se zakrivaju interni implementacijski detalji za svaki fizički operator. Za svaki operator implementiraju se tri metode:
 - `open()`: metoda koja inicijalizira stanje iteratora, alocira potrebnu memoriju (npr. ulazne i izlazne redove)
 - `getNext()`: metoda pozivatelju vraća "sljedeću n-torku"
 - `close()`: oslobađa zauzete resurse
- iteratori se mogu izvršavati svaki unutar svojeg vlastitog procesa ili dretve
 - omogućava se paralelno izvršavanje upita

Iteratori - *table scan* iterator

```
TableScan {
    open(r) {
        bR := the first block of r;
        tR := the first tuple of bR;
        FOUND := true; /* global variable */
    }
    getNext() {
        if tR is past last tuple on block bR
            increment bR to the next block;
            if there is no next block in r
                FOUND := false;
                return;
            else
                tR := the first tuple of bR;
        return_tR := tR;
        increment tR to the next tuple of bR;
        return return_tR;
    }
    close() { }
}
```

- pseudokod je znatno pojednostavljen - nedostaju oznake programskih blokova {}, definicije varijabli, rukovanje praznim relacijama, alokacija memorije u open() i oslobođanje memorije u close(), ...

Iteratori - *union (bag)* iterator

```
BagUnion {
    open(r, s) {
        tscanR := new TableScan(); tscanR.open(r);
        currentRel := r;
    }
    getNext() {
        if currentRel = r
            t := tscanR.getNext();
        if FOUND
            return t;
        else
            tscanR.close();
            currentRel := s;
            tscanS := new TableScan(); tscanS.open(s);
        return tscanS.getNext();
    }
    close() {
        tscanS.close();
    }
}
```

- pseudokod je znatno pojednostavljen - nedostaju oznake programskih blokova {}, definicije varijabli, ...

Složeni izrazi relacijske algebre

- procjena troškova

Složeni izrazi relacijske algebre - procjena troškova

Primjer:

```
SELECT *
  FROM stud AS r, ispit AS s
 WHERE mbr = mbrStud
   AND datIspit = '17.6.2008';
```

$$\sigma_{\text{datIsp}='17.6.2008' \wedge \text{mbr}=\text{mbrStud}}(\text{stud} \times \text{ispit})$$
$$\sigma_{\text{datIsp}='17.6.2008'} \underset{\text{mbr}=\text{mbrStud}}{(\text{stud} \triangleright\triangleleft \text{ispit})}$$
$$\text{stud} \triangleright\triangleleft (\sigma_{\text{datIsp}='17.6.2008'} \underset{\text{mbr}=\text{mbrStud}}{(\text{ispit})})$$

ekvivalentni izrazi
relacijske algebre

- procijeniti broj U/I operacija koje će se obaviti za svaki od navedenih ekvivalentnih izraza relacijske algebre

Složeni izrazi relacijske algebre - procjena troškova

Pretpostavke:

stud	ispit
■ $N(r) = 5\ 000$	■ $N(s) = 100\ 000$
■ $B(r) = 500$	■ $B(s) = 3000$
■ $V(mbr, r) = 5000$	■ $V(datlsp, s) = 200$
■ veličina n-torke $t_r = 0.05$ blokova	■ veličina n-torke $t_s = 0.02$ blokova

Uočiti: $B(r)$ može biti veći od $N(r) \cdot$ veličina n-torke t_r jer blokovi ne moraju biti u cijelosti popunjeni n-torkama

- svi međurezultati se zapisuju u sekundarnu memoriju
- za spajanje (i Kartezijev produkt) koristiti *block nested-loop join*. Prepostaviti najlošiji slučaj: na raspolaganju je $M=3$ (broj blokova glavne memorije).
- nad relacijama nema kreiranih indeksa, ne koristi se *autoindex*
- primarni ključ relacije r je $mbrStud$

Složeni izrazi relacijske algebre - procjena troškova

$\sigma_{\text{datlsp}='17.6.2008' \wedge \text{mbr}=\text{mbrStud}}(\text{stud} \times \text{ispit})$

- trošak za $r_1 = \text{stud} \times \text{ispit}$ - *blocked nested-loop*
 - čitanje: $B(r) + B(r) \cdot B(s) = 500 + 1.5 \cdot 10^6 \approx 1.5 \cdot 10^6$ U/I operacija
 - pisanje međurezultata (r_1):
 - $N(r_1) = N(r) \cdot N(s) = 500 \cdot 10^6$
 - veličina n-torke rezultata: 0.07 blokova (zašto?)
 - $B(r_1) = 35 \cdot 10^6$ blokova $\Rightarrow 35 \cdot 10^6$ U/I operacija
 - trošak za $r_2 = \sigma_{\text{datIsp}='17.6.2008' \wedge \text{mbr}=\text{mbrStud}}(r_1)$ - *table-scan*
 - $35 \cdot 10^6$ U/I operacija
 - ukupni trošak (bez troškova zapisivanja rezultata): $\approx 71.5 \cdot 10^6$ U/I operacija

Složeni izrazi relacijske algebre - procjena troškova

$$\sigma_{\text{datisp}='17.6.2008'} (\text{stud} \bowtie \text{ispit})$$

mbr=mbrStud

- trošak za $r_1 = \text{stud} \bowtie \text{ispit}$ - *blocked nested-loop join*
 - čitanje: $B(r) + B(r) \cdot B(s) = 500 + 1.5 \cdot 10^6 \approx 1.5 \cdot 10^6$ U/I operacija
 - pisanje međurezultata (r_1):
 - $N(r_1) = 100\ 000$ - jer se svaka t_s može spojiti s najviše jednom t_r
 - veličina n-torke rezultata: 0.07 blokova
 - $B(r_1) = 7000$ blokova $\Rightarrow 7000$ U/I operacija
- trošak za $r_2 = \sigma_{\text{datisp}='17.6.2008'}(r_1)$ - *table-scan*
 - 7000 U/I operacija
- ukupni trošak (bez troškova zapisivanja rezultata): $\approx 1.51 \cdot 10^6$ U/I operacija

Složeni izrazi relacijske algebre - procjena troškova

$\text{stud} \triangleright\triangleleft (\sigma_{\text{datlsp}='17.6.2008'}(\text{ispit})$
mbr=mbrStud

- trošak za $r_1 = \sigma_{\text{datlsp}='17.6.2008'}(\text{ispit})$ - *table-scan*
 - čitanje: $B(s) = 3000$ U/I operacija
 - pisanje međurezultata (r_1):
 - $N(r_1) = 100\ 000 / V(\text{datlsp}, s)$ - pretp: jednolika distribucija po datlsp
 - veličina n-torke rezultata: 0.02 blokova
 - $B(r_1) = 10$ blokova $\Rightarrow 10$ U/I operacija
- trošak za $r_2 = \text{stud} \triangleright\triangleleft r_1$ - *blocked nested-loop join*
 - $B(r_1) + B(r_1) \cdot B(r) = 5010$ U/I operacija
- ukupni trošak (bez troškova zapisivanja rezultata): 8020 U/I operacija
- za vježbu: procijeniti trošak za svaki od prikazanih primjera ako su na raspolaganju $M=102$ blokova glavne memorije

Složeni izrazi relacijske algebre - procjena troškova

- trošak izvršavanja operacija relacijske algebre nad temeljnim relacijama i materijaliziranim pogledima ovisi o primjenjenom fizičkom operatoru
- u složenim izrazima relacijske algebre važna je i veličina međurezultata
- međurezultati se (u osnovi) pohranjuju kao neporedane datoteke
 - posljedica: broj U/I operacija koje se obavljaju nad međurezultatima je proporcionalan veličini međurezultata
- prethodni primjer pokazuje kako je za procjenu veličine međurezultata važan i broj n-torki i veličina n-torke
 - veličina n-torke je podatak lako dostupan iz rječnika podataka. Ako se radi o n-torci varijabilne duljine, može se koristiti podatak o prosječnoj veličini n-torke
 - broj n-torki u rezultatu obavljanja neke operacije nije moguće saznati prije nego se operacija obavi
- budući da se ne može unaprijed znati koliko će n-torki imati međurezultat, koriste se pravila za procjenu broja n-torki u rezultatu operacije relacijske algebre
- cilj procjene NIJE predvidjeti točan broj n-torki u rezultatu

Procjena broja n-torki u rezultatu operacije rel. algebre

- za relaciju $r(A, B, \dots)$ u rječniku podataka pohranjeni su statistički podaci:
 - $N(r)$ - broj n-torki relacije r
 - $V(A, r)$ - broj različitih vrijednosti atributa A u relaciji r
 - $V(A, r) = G_{\text{COUNT}(\cdot)}(\pi_A(r))$
 - pri tome A može biti jedan atribut ili skup atributa
 - $\min(A, r), \max(A, r)$ - najmanja i najveća vrijednost atributa A u relaciji r
 - histogrami

Neki sustavi (npr. IBM IDS) umjesto *min* i *max* pohranjuju *second-min* i *second-max* (smatra se da su *min* i *max* često ekstremne vrijednosti koje je bolje ne uzimati u obzir)

Jednostavna selekcija

$$\sigma_{A=v}(r)$$

- uz pretpostavku: jednolika razdioba vrijednosti atributa A
 $N(r) / V(A, r)$
- ako $V(A, r)$ nije poznata: $N(r) / 10$
- ako razdioba nije jednolika, mogu pomoći histogrami (ako su na raspolaganju)

Histogrami:

- za svaki raspon vrijednosti atributa A (granice raspona ovise o primjenjenoj rezoluciji) u rječniku podataka se evidentira broj n-torki, npr.
- ispit(mbrStud, datIspit, sifPred, ocjena)
 - $N(ispit) = 100\ 000$
 - $V(ocjena, ispit) = 5$



Procjena broja n-torki za $\sigma_{ocjena=2}(ispit)$

- bez histograma: 20 000
- uz histogram: 10 000

Jednostavna selekcija

$\sigma_{A \leq v}(r)$

- ako je vrijednost v poznata u trenutku optimizacije
 - za $v < \min(A, r)$ 0
 - za $v \geq \max(A, r)$ $N(r)$
 - inače $N(r) \cdot (v - \min(A, r)) / (\max(A, r) - \min(A, r))$
- ako vrijednost v ili $\min(A, r)$, $\max(A, r)$ nisu poznate u trenutku optimizacije
 - $N(r) / 3$
 - u tom slučaju se isti izraz koristi i za nejednakosti oblika $<$, \geq , $>$

$\sigma_{A \text{ LIKE } \text{izraz}}(r)$

$N(r) / 5$

Selekcija - uvjet selekcije s negacijom

$\sigma_{\neg F}(r)$

- ako se procijenjeni broj n-torki za operaciju $\sigma_F(r)$ označi s $N_F(r)$ tada je procijenjeni broj n-torki u rezultatu operacije $\sigma_{\neg F}(r)$

$$N(r) - N_F(r)$$

- prethodni izraz se može iskoristiti za procjenu npr.

$$\sigma_{A \neq v}(r) = \sigma_{\neg(A=v)}(r)$$

- također

$$\sigma_{A > v}(r) = \sigma_{\neg(A \leq v)}(r)$$

- ali ako v ili $\min(A, r)$, $\max(A, r)$ nisu poznati u trenutku optimizacije, tada se opet procjenjuje na $N(r) / 3$, a ne $N(r) - N(r) / 3$

- također

$$\sigma_{A \text{ NOT LIKE } \text{izraz}}(r) = \sigma_{\neg(A \text{ LIKE } \text{izraz})}(r)$$

Selektivnost predikata (*selectivity*)

- $f(F, r)$: broj iz intervala $[0, 1]$
- omjer broja n-torki iz r koje zadovoljavaju predikat F i ukupnog broja n-torki u r
- $f(F, r) = N(\Sigma_F(r)) / N(r)$

Složena selekcija - konjunktivna forma

$$\sigma_{F_1 \wedge F_2 \wedge \dots \wedge F_m}(r)$$

- za svaki F_i se procjenjuje veličina rezultata $N_{F_i}(r) = N(\sigma_{F_i}(r))$
- $N_{F_i}(r)$ je procijenjeni broj n-torki za operaciju $\sigma_{F_i}(r)$
 - procjena se obavlja prema prethodno opisanim pravilima
- vjerojatnost da n-torka zadovoljava pojedinačni uvjet F_i je $N_{F_i}(r) / N(r)$
 - odnosno $f(F_i, r)$
- ako su F_i nezavisni, vjerojatnost da n-torka zadovoljava sve F_i jest
$$(N_{F_1}(r) / N(r)) \cdot (N_{F_2}(r) / N(r)) \cdot \dots \cdot (N_{F_m}(r) / N(r))$$
 - odnosno $f(F_1, r) \cdot f(F_2, r) \cdot \dots \cdot f(F_m, r)$
- procijenjeni broj n-torki u rezultatu je dakle
$$N(r) \cdot (N_{F_1}(r) \cdot N_{F_2}(r) \cdot \dots \cdot N_{F_m}(r)) / (N(r))^m$$
 - odnosno $N(r) \cdot f(F_1, r) \cdot f(F_2, r) \cdot \dots \cdot f(F_m, r)$

Nezavisnost pojedinačnih uvjeta u složenim predikatima

- prethodno navedeni izrazi podrazumijevaju nezavisnost pojedinačnih uvjeta
osoba

mbr	pbrRod	pbrStan	spol
101	51000	51000	Ž
102	51000	51000	M
103	51000	51000	Ž
104	51000	21000	M
105	51000	51000	Ž
106	51000	51000	M
107	51000	51000	Ž
108	51000	51000	M
109	21000	21000	Ž
110	21000	21000	M
111	21000	21000	M
112	21000	21000	Ž
113	21000	21000	Ž
114	21000	21000	M
115	21000	51000	Ž
116	21000	21000	M

- ocijeniti nezavisnost pojedinačnih uvjeta u složenim predikatima sljedećih algebarskih operacija

$$s = \sigma_{pbrRod=51000 \wedge pbrStan=21000} (\text{osoba})$$

$$N(s) = 16 \cdot 0.5 \cdot 0.5 = 4$$

?

$$s = \sigma_{pbrRod=51000 \wedge spol='\check{Z}'} (\text{osoba})$$

$$N(s) = 16 \cdot 0.5 \cdot 0.5 = 4$$

?

Složena selekcija - disjunktivna forma

$$\sigma_{F_1 \vee F_2 \vee \dots \vee F_m}(r)$$

- vjerojatnost da n-torka ne zadovoljava pojedinačni uvjet F_i je $1 - N_{F_i}(r) / N(r)$
 - odnosno $1 - f(F_i, r)$
- ako su F_i nezavisni, vjerojatnost da n-torka ne zadovoljava niti jedan F_i jest
$$(1 - N_{F_1}(r) / N(r)) \cdot (1 - N_{F_2}(r) / N(r)) \dots \cdot (1 - N_{F_m}(r) / N(r))$$
 - odnosno $(1 - f(F_1, r)) \cdot (1 - f(F_2, r)) \cdot \dots \cdot (1 - f(F_m, r))$
- vjerojatnost da zadovoljava barem jedan od F_i
$$1 - (1 - N_{F_1}(r) / N(r)) \cdot (1 - N_{F_2}(r) / N(r)) \dots \cdot (1 - N_{F_m}(r) / N(r))$$
 - odnosno $1 - (1 - f(F_1, r)) \cdot (1 - f(F_2, r)) \cdot \dots \cdot (1 - f(F_m, r))$
- procijenjeni broj n-torki u rezultatu je dakle
$$N(r) \cdot (1 - (1 - N_{F_1}(r) / N(r)) \cdot (1 - N_{F_2}(r) / N(r)) \dots \cdot (1 - N_{F_m}(r) / N(r)))$$
 - odnosno $N(r) (1 - (1 - f(F_1, r)) \cdot (1 - f(F_2, r)) \cdot \dots \cdot (1 - f(F_m, r)))$

Spajanje

$r(R) \bowtie s(S)$

- ako je $R \cap S = \emptyset \Rightarrow N(r) \cdot N(s)$
- ako je $R \cap S = \text{ključ u } R \Rightarrow N(s)$ (jer se svaka t_s može spojiti s najviše jednom t_r)
- ako je $R \cap S = \text{ključ u } S \Rightarrow N(r)$ (jer se svaka t_r može spojiti s najviše jednom t_s)
- $R \cap S = A$, A je atribut ili skup atributa koji nije ključ niti za R niti za S
- promatra se jedna n-torka t_r (ona ima neku konkretnu A-vrijednost)
- n-torki t_s koje imaju A-vrijednost jednaku A-vrijednosti n-torke t_r ima $N(s) / V(A, s)$
- dakle svaka t_r će se spojiti sa $N(s) / V(A, s)$ n-torki t_s . Procijenjeni broj n-torki $N(r) \cdot N(s) / V(A, s)$
- procjena za $s \bowtie r$: $N(s) \cdot N(r) / V(A, r)$
- očito, ako je $V(A, r) \neq V(A, s)$, procjene se razlikuju! U takvom slučaju, kao točnija procjena uzima se manja od dviju prikazanih, dakle $N(r) \cdot N(s) / \max(V(A, r), V(A, s))$

Unija, presjek, razlika, projekcija, agregacija

$$\sigma_{F_1}(r) \cup \sigma_{F_2}(r)$$

unija podskupova iste relacije

- procjenjuje se jednako kao rezultat operacije $\sigma_{F_1 \vee F_2}(r)$

$$\sigma_{F_1}(r) \cap \sigma_{F_2}(r)$$

presjek podskupova iste relacije

- procjenjuje se jednako kao rezultat operacije $\sigma_{F_1 \wedge F_2}(r)$

$$r \cup s$$

$$\Rightarrow N(r) + N(s)$$

$$r \cap s$$

$$\Rightarrow \min(N(r), N(s))$$

$$r \setminus s$$

$$\Rightarrow N(r)$$

$$\pi_L(r)$$

$$\Rightarrow V(L, r)$$

$${}_L G_{\mathcal{AF}_1(B_1), \mathcal{AF}_2(B_2), \dots, \mathcal{AF}_n(B_n)}(r) \Rightarrow V(L, r)$$

Primjer

```
CREATE TABLE ispit (
    mbrStud
, sifPred
, sifNast
, datIspit
, ocjena);
```

- procijeniti broj n-torki u rezultatu operacije:

$$\sigma_{\text{ocjena}=2 \wedge \text{datIspit}='16.4.2008'}(\text{ispit})$$

$$N_{\text{ocjena}=2}(\text{ispit}) = N(\text{ispit}) / V(\text{ocjena}, \text{ispit})$$

$$N_{\text{datIspit}='16.4.2008'}(\text{ispit}) = N(\text{ispit}) / V(\text{datIspit}, \text{ispit})$$

$$N(r) \cdot (N_{F_1}(r) \cdot N_{F_2}(r) \cdot \dots \cdot N_{F_m}(r)) / (N(r))^m$$

$$N(\text{rez}) = N(\text{ispit}) / (V(\text{ocjena}, \text{ispit}) \cdot V(\text{datIspit}, \text{ispit})) \approx 27$$

$$N(\text{ispit}) = 200\,000$$

$$V(\text{ocjena}, \text{ispit}) = 5$$

$$V(\text{datIspit}, \text{ispit}) = 1500$$

$$V(\text{sifNast}, \text{ispit}) = 100$$

$$V(\text{sifPred}, \text{ispit}) = 200$$

$$\min(\text{sifNast}, \text{ispit}) = 101$$

$$\max(\text{sifNast}, \text{ispit}) = 200$$

Primjer

```
CREATE TABLE ispit (
    mbrStud
, sifPred
, sifNast
, datIspit
, ocjena);
```

- procijeniti broj n-torki u rezultatu operacije:

 $\sigma_{\text{ocjena}=2 \wedge \text{sifNast} \leq 120}(\text{ispit})$

$$N_{\text{ocjena}=2}(\text{ispit}) = N(\text{ispit}) / V(\text{ocjena}, \text{ispit}) = 40\ 000$$

$$N(r) \cdot (v - \min(A, r)) / (\max(A, r) - \min(A, r))$$

$$N_{\text{sifNast} \leq 120}(\text{ispit}) = N(\text{ispit}) \cdot 19 / 99 \approx 38\ 380$$

$$N(r) \cdot (N_{F_1}(r) \cdot N_{F_2}(r) \cdot \dots \cdot N_{F_m}(r)) / (N(r))^m$$

$$N(\text{rez}) \approx N(\text{ispit}) \cdot (40\ 000 \cdot 38\ 380) / N(\text{ispit})^2 \approx 7677$$

$$\begin{aligned}N(\text{ispit}) &= 200\ 000 \\V(\text{ocjena}, \text{ispit}) &= 5 \\V(\text{datIspit}, \text{ispit}) &= 1500 \\V(\text{sifNast}, \text{ispit}) &= 100 \\V(\text{sifPred}, \text{ispit}) &= 200 \\\min(\text{sifNast}, \text{ispit}) &= 101 \\\max(\text{sifNast}, \text{ispit}) &= 200\end{aligned}$$

Primjer

```
CREATE TABLE ispit (
    mbrStud
, sifPred
, sifNast
, datIspit
, ocjena);
```

- procijeniti broj n-torki u rezultatu operacije:

$$\sigma_{\text{sifNast}=120 \vee \text{sifPred}=320}(\text{ispit})$$

$$N_{\text{sifNast}=120}(\text{ispit}) = N(\text{ispit}) / V(\text{sifNast}, \text{ispit}) = 2\,000$$

$$N_{\text{sifPred}=320}(\text{ispit}) = N(\text{ispit}) / V(\text{sifPred}, \text{ispit}) = 1\,000$$

$$N(r) \cdot (1 - (1 - N_{F_1}(r) / N(r)) \cdot (1 - N_{F_2}(r) / N(r)) \cdots (1 - N_{F_m}(r) / N(r)))$$

$$N(\text{rez}) = N(\text{ispit}) \cdot (1 - (1 - 2000 / 200000) \cdot (1 - 1000 / 200000)) = 2990$$

$$N(\text{ispit}) = 200\,000$$

$$V(\text{ocjena}, \text{ispit}) = 5$$

$$V(\text{datIspit}, \text{ispit}) = 1500$$

$$V(\text{sifNast}, \text{ispit}) = 100$$

$$V(\text{sifPred}, \text{ispit}) = 200$$

$$\min(\text{sifNast}, \text{ispit}) = 101$$

$$\max(\text{sifNast}, \text{ispit}) = 200$$

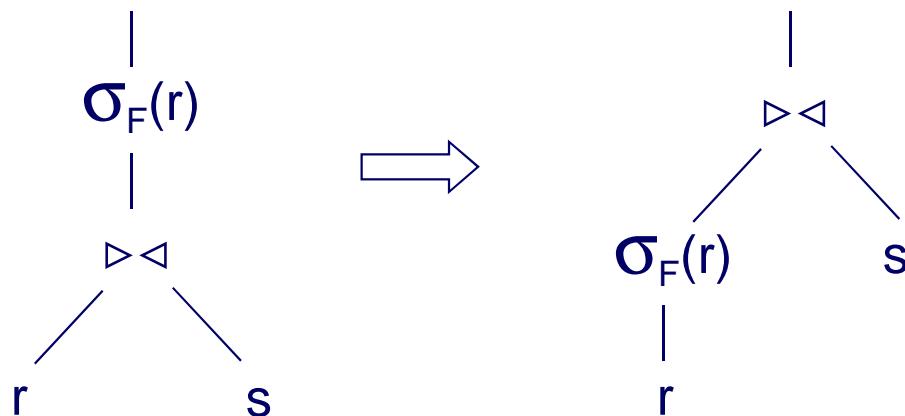
Optimizacija

Tehnike optimizacije

- **optimizacija temeljena na primjeni heurističkih pravila**
 - *heuristic optimization*
 - niz pravila (*equivalence rules*) kojima se upit transformira u ekvivalentan upit
 - pravila čijom se primjenom, u većini slučajeva (ali ne uvijek!), dobije efikasniji plan izvršavanja

Primjer:

- potiskivanje selekcije: operaciju selekcije obaviti u čim ranijoj fazi
- ako F sadrži samo attribute iz R , tada $\sigma_F(r \bowtie s) \equiv \sigma_F(r) \bowtie s$



Tehnike optimizacije

- **optimizacija temeljena na procjeni troškova**
 - *cost based optimization*
 - optimizator izračunava ukupni trošak svakog ekvivalentnog plana izvršavanja (ili mnogih ekvivalentnih planova). Pri tome procjenjuje:
 - broj U/I operacija
 - veličina korištene glavne memorije
 - veličina korištene sekundarne memorije
 - trošak procesorskog vremena (značajno za male baze podataka i za *solid-memory databases*)
 - komunikacijski troškovi (značajno za distribuirane baze podataka)
 - odabire plan izvršavanja s najmanjim ukupnim troškom
 - nedostatak: procjena troška obavlja se za veliki broj alternativnih planova izvršavanja. Trošak same optimizacije postaje značajan faktor
- kao jednostavan primjer optimizacije temeljene na procjeni troškova može poslužiti prethodno izložen primjer procjene troškova u složenim operacijama relacijske algebre. U primjeru su troškovi procijenjeni za (samo) tri alternativna plana, te se usporedbom ukupnog troška moglo procijeniti koji je plan najbolji

Tehnike optimizacije

- često se dvije tehnike kombiniraju: primjenom heurističkih pravila se odredi ograničeni broj "potencijalno dobrih" planova izvršavanja, te se na tako ograničenom skupu planova provodi optimizacija temeljena na procjeni troškova
- sustavi za upravljanje bazama podataka omogućavaju određeni stupanj kontrole nad odabirom tehnike optimizacije

Primjer:

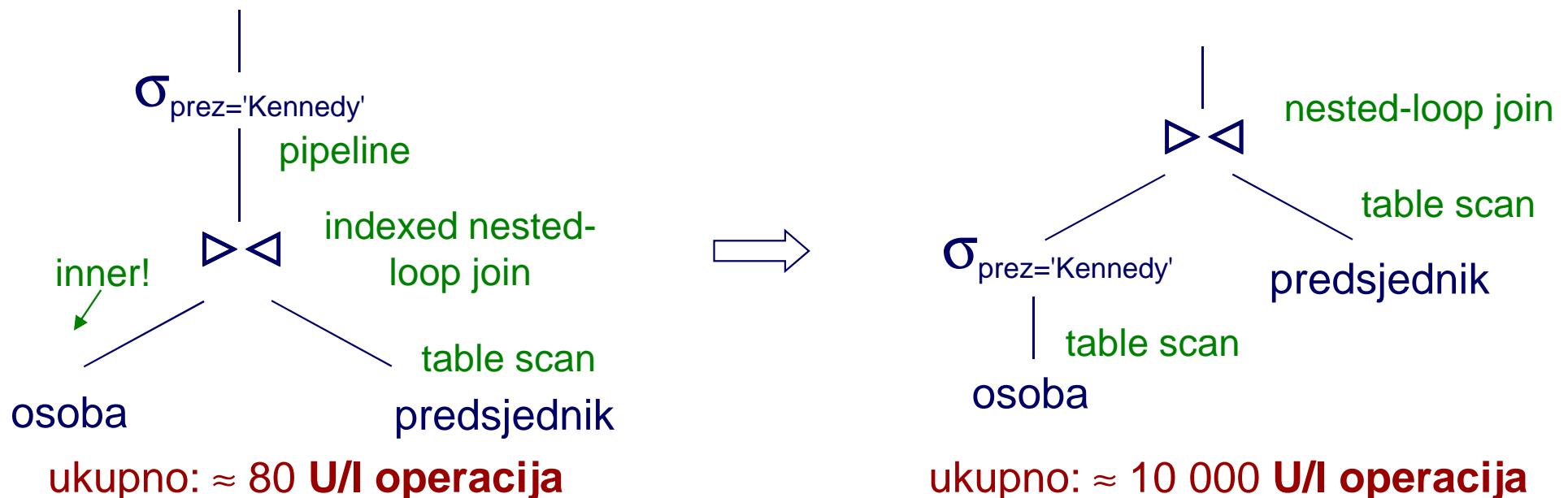
- *The algorithm that a SET OPTIMIZATION HIGH statement invokes is a sophisticated, cost-based strategy that examines all reasonable query plans and selects the best overall alternative. For large joins, this algorithm can incur more overhead than desired. In extreme cases, you can run out of memory.*
- *The alternative algorithm that a SET OPTIMIZATION LOW statement invokes eliminates unlikely join strategies during the early stages, which reduces the time and resources spent during optimization. However, when you specify a low level of optimization, the optimal strategy might not be selected because it was eliminated from consideration during early stages of the algorithm.*

Samo heuristička optimizacija nije dovoljna

- optimizator se ne bi smio osloniti samo na pravila za heurističku optimizaciju
- primjer pokazuje kako se primjenom isključivo heurističke optimizacije može dobiti lošiji plan od inicijalnog

```
SELECT * FROM osoba JOIN predsjednik  
    ON osoba.sif = predsjednik.sifOso  
 WHERE prez = 'Kennedy';
```

- $N(\text{osoba}) = 1\ 000\ 000$
- $N(\text{predsjednik}) = 20$
- $V(\text{prez}, \text{predsjednik}) = 20$
- $V(\text{prez}, \text{osoba}) = 5\ 000$
- idx za $\text{osoba}.\text{sif}$, $d(\text{idx}) = 4$
- $B(\text{osoba}) = 10\ 000$
- $B(\text{predsjednik}) = 1$



Heuristička optimizacija

Pravila za transformaciju izraza relacijske algebre

- transformacije se temelje na pravilima ekvivalentnosti izraza relacijske algebre
- *equivalence rules, algebraic laws*
- ovdje će biti navedena samo najvažnija pravila za transformaciju, ona koja su važna u postupcima heurističke optimizacije
- prošireni popis pravila nalazi se na kraju predavanja
- oznake
 - relacije $r(R)$, $s(S)$, $t(T)$, ...
 - formule F , F_1 , F_2 , G , ...
 - skupovi atributa L_1 , L_2 , M , N , ...

1. Pravila vezana uz operaciju selekcije

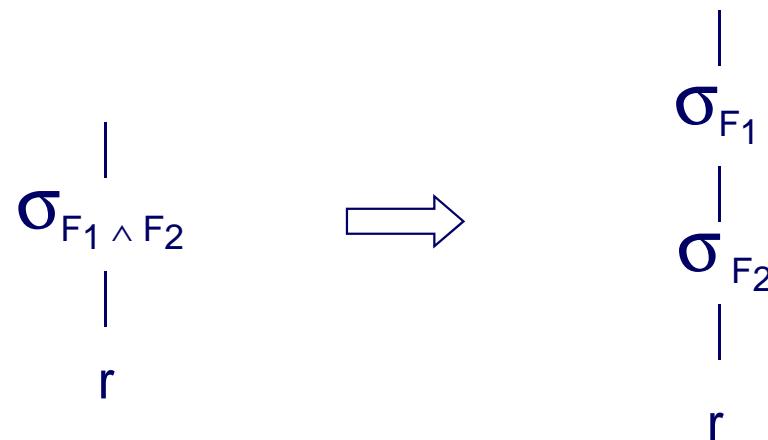
$$1.1. \quad \sigma_{F_1 \wedge F_2}(r) \equiv \sigma_{F_1}(\sigma_{F_2}(r))$$

$$1.2. \quad \sigma_{F_1}(\sigma_{F_2}(r)) \equiv \sigma_{F_2}(\sigma_{F_1}(r))$$

Primjena na stablu upita

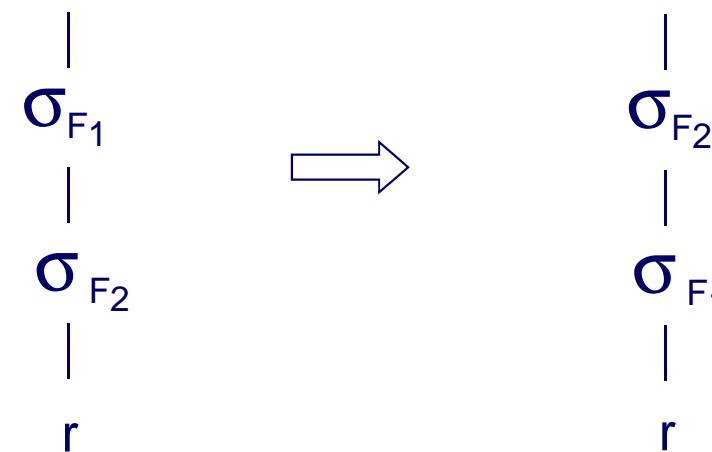
dekompozicija operatora selekcije s konjunktivnim oblikom formule

$$\sigma_{F_1 \wedge F_2}(r) \equiv \sigma_{F_1}(\sigma_{F_2}(r))$$



promjena redoslijeda operacija selekcije

$$\sigma_{F_1}(\sigma_{F_2}(r)) \equiv \sigma_{F_2}(\sigma_{F_1}(r))$$



Pravila za transformaciju izraza relacijske algebre

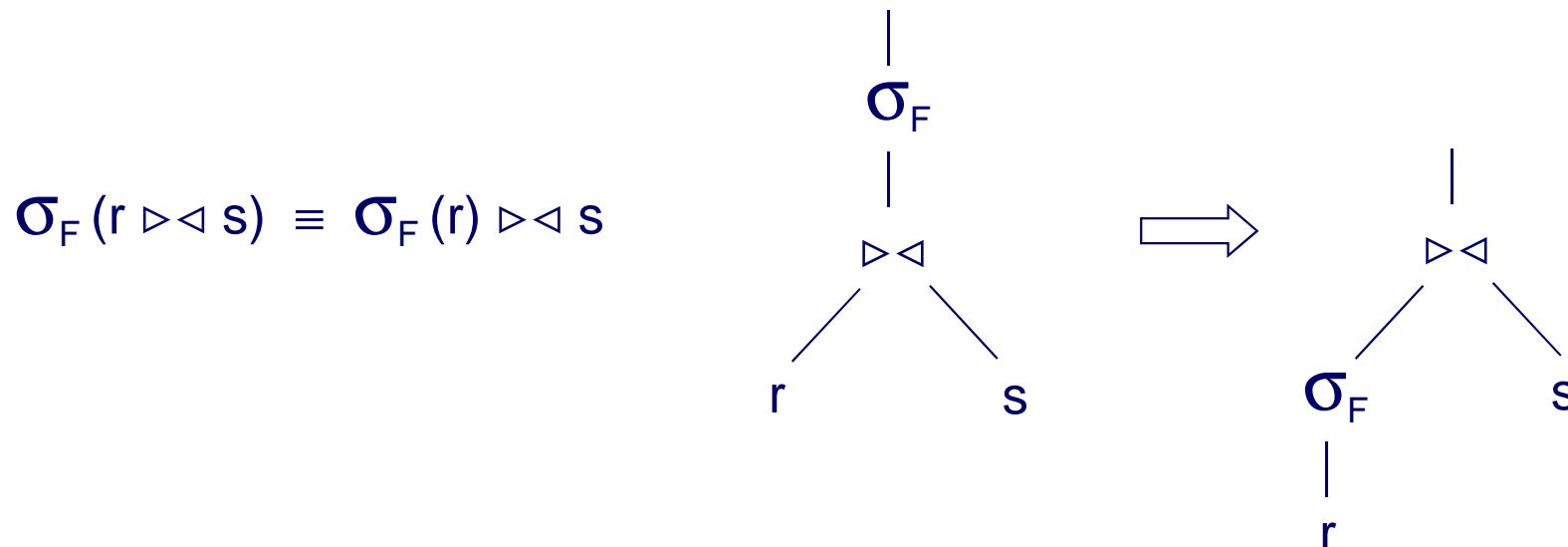
- ako F sadrži samo attribute iz R

$$1.3. \quad \sigma_F(r \bowtie s) \equiv \sigma_F(r) \bowtie s$$

$$1.4. \quad \sigma_F(r \underset{G}{\bowtie} s) \equiv \sigma_F(r) \underset{G}{\bowtie} s$$

$$1.5. \quad \sigma_F(r \times s) \equiv \sigma_F(r) \times s$$

Prikaz često korištene transformacije: potiskivanje selekcije

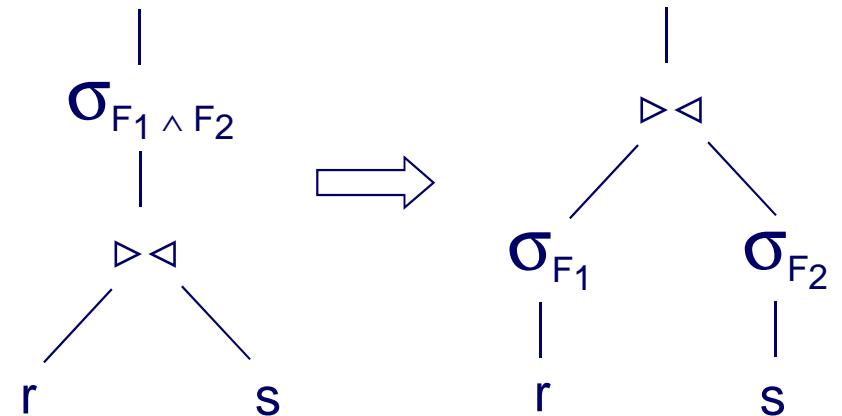


Pravila za transformaciju izraza relacijske algebre

- slično, ako F_1 sadrži samo attribute iz R , a F_2 sadrži samo attribute iz S

$$1.6. \quad \sigma_{F_1 \wedge F_2}(r \triangleright\!\triangleleft s) \equiv \sigma_{F_1}(r) \triangleright\!\triangleleft \sigma_{F_2}(s)$$

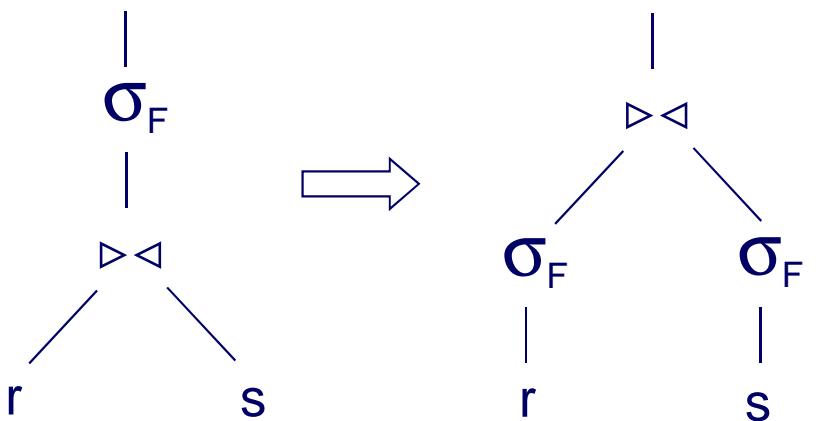
$$1.7. \quad \sigma_{F_1 \wedge F_2}(r \triangleright\!\triangleleft s) \equiv \underset{G}{\sigma_{F_1}}(r) \triangleright\!\triangleleft \underset{G}{\sigma_{F_2}}(s)$$



- slično, ako i R i S sadrže sve attribute iz F

$$1.8. \quad \sigma_F(r \triangleright\!\triangleleft s) \equiv \sigma_F(r) \triangleright\!\triangleleft \sigma_F(s)$$

$$1.9. \quad \sigma_F(r \triangleright\!\triangleleft s) \equiv \underset{G}{\sigma_F}(r) \triangleright\!\triangleleft \underset{G}{\sigma_F}(s)$$

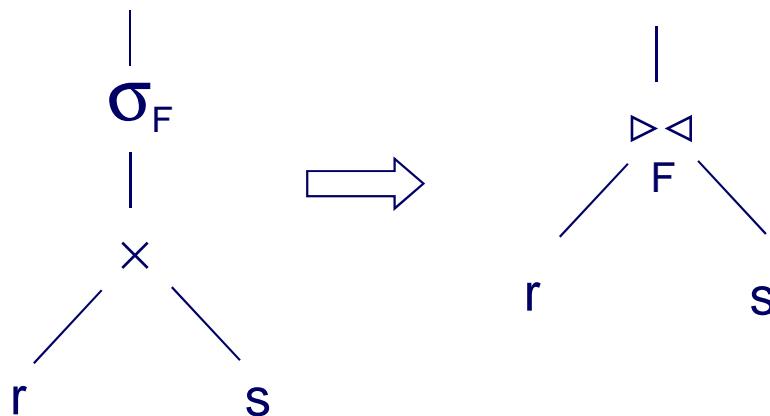


Pravila za transformaciju izraza relacijske algebre

- važno pravilo o selekciji, Kartezijevom produktu i spajanju
 - ako F odgovara uvjetu spajanja (θ -join) između r i s , tada

1.10.

$$\sigma_F(r \times s) \underset{F}{\equiv} r \bowtie s$$



Pravila za transformaciju izraza relacijske algebre

2. Komutativnost

$$2.1. \quad r \times s \equiv s \times r$$

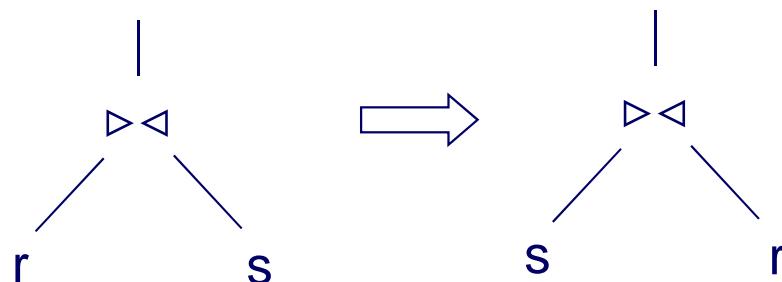
$$2.2. \quad r \triangleright\triangleleft s \equiv s \triangleright\triangleleft r$$

$$2.3. \quad r \triangleright\triangleleft s \equiv s \triangleright\triangleleft r$$

F

F

Primjer: slobodna promjena redoslijeda operanada



Pravila za transformaciju izraza relacijske algebre

3. Asocijativnost

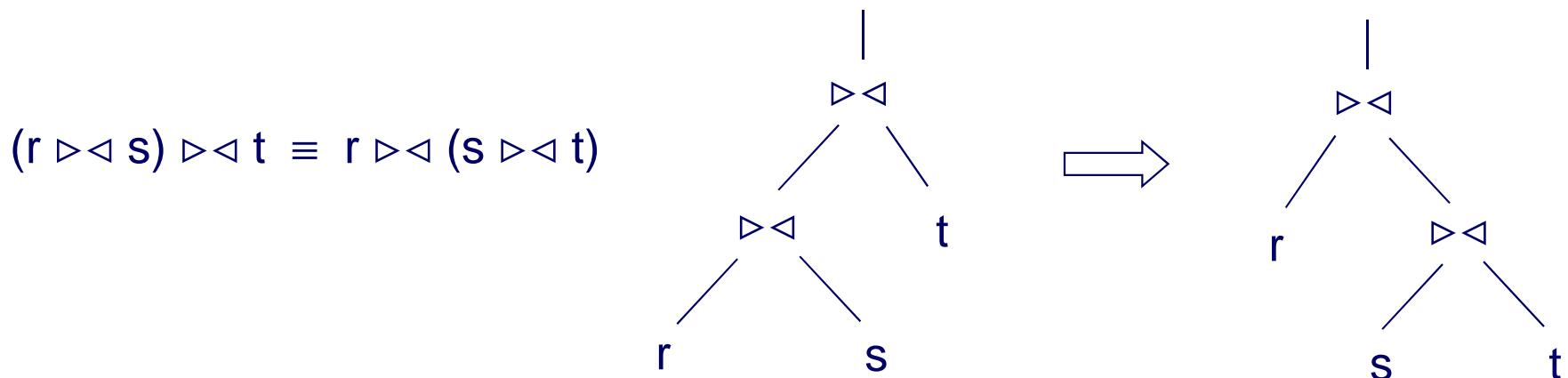
$$3.1. \quad (r \times s) \times t \equiv r \times (s \times t)$$

$$3.2. \quad (r \triangleright\triangleleft s) \triangleright\triangleleft t \equiv r \triangleright\triangleleft (s \triangleright\triangleleft t)$$

$$3.3. \quad (r \triangleright\triangleleft s) \triangleright\triangleleft t \equiv r \triangleright\triangleleft (s \triangleright\triangleleft t)$$

F₁ F₂ F₁ F₂

Primjer: često korištena transformacija: promjena redoslijeda spajanja



Osnovni koraci u heurističkoj optimizaciji upita

1. dekomponirati operatore selekcije koji sadrže konjunktivne izraze
 - postiže se veći stupanj slobode u pomicanju operatora selekcije unutar stabla upita
2. potisnuti operatore selekcije što je moguće dublje prema listovima stabla upita
 - obavljanjem operacija selekcije u najranijim mogućim fazama smanjuje se veličina međurezultata koji se koristi u dalnjim operacijama što najčešće dovodi do smanjenja ukupnog troška
3. Kartezijev produkt kombinirati s operacijom selekcije u operaciju spajanja uz uvjet (θ -join)
4. redoslijed spajanja odrediti tako da se prvo obavljaju operacije spajanja koje rezultiraju najmanjim brojem n-torki (ili veličinom međurezultata)
 - ovdje su navedena tek najvažnija (najčešće korištena) pravila

Navedena pravila koristiti za rješavanje zadataka u kojima se traži heuristička optimizacija

Primjer

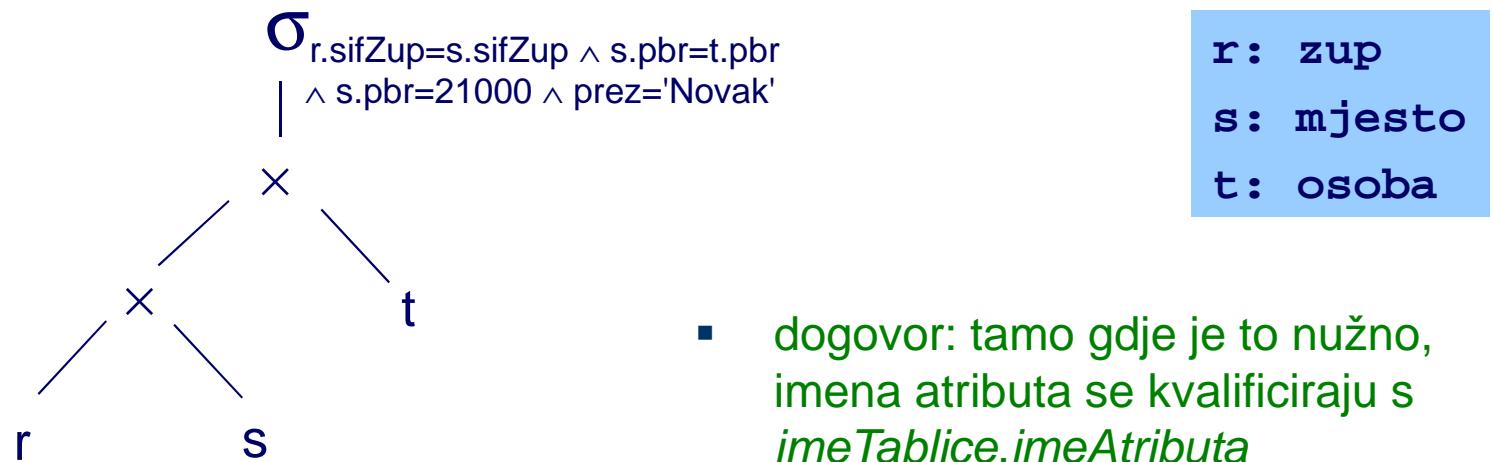
```
SELECT *
  FROM zup AS r, mjesto AS s, osoba AS t
 WHERE zup.sifZup = mjesto.sifZup
   AND mjesto.pbr = osoba.pbr
   AND mjesto.pbr = 21000
   AND osoba.prez = 'Novak';
```

zup
sifZup
nazZup

mjesto
pbr
nazMj
sifZup

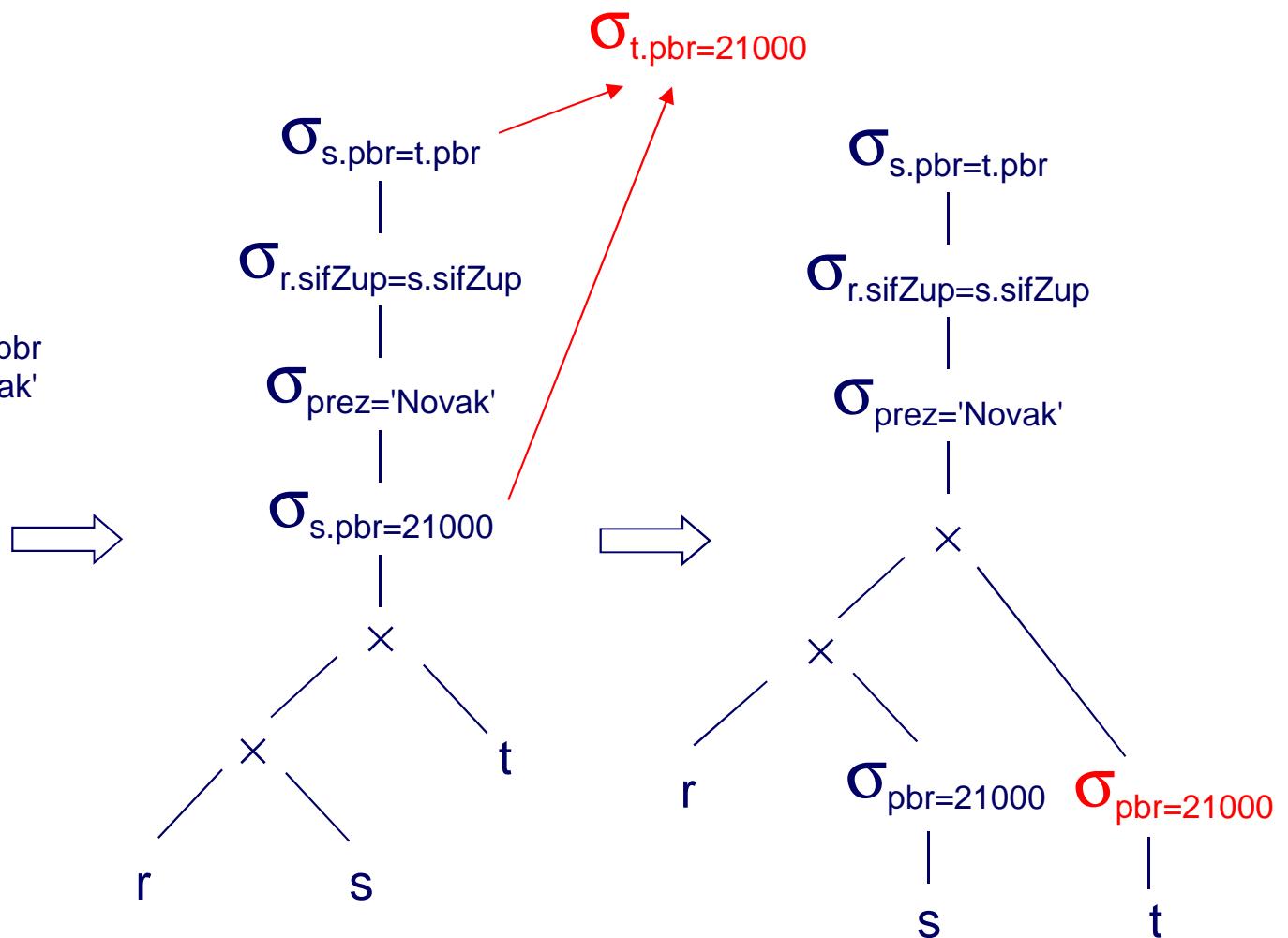
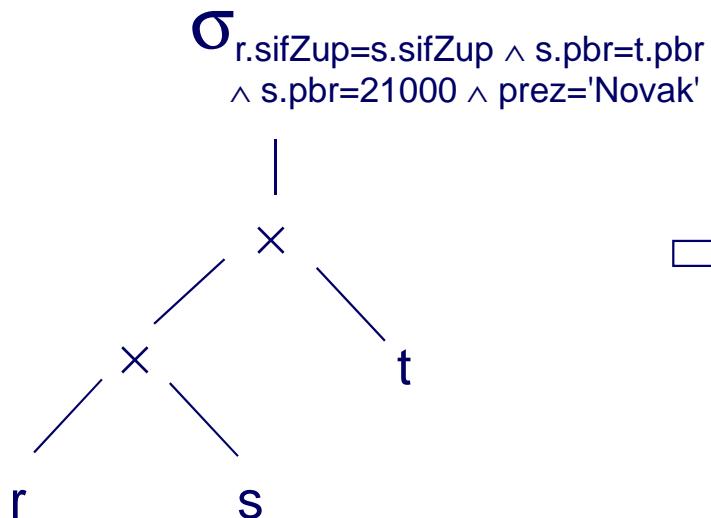
osoba
sifOso
nazMj
ime
prez
pbr

- nacrtati inicijalno stablo upita. Redoslijed operacija Kartezijskog produkta (uočiti da originalni SQL upit ne sadrži operacije spajanja!) u inicijalnom stablu mora odgovarati redoslijedu kojim su relacije navedene u upitu
- provesti heurističku optimizaciju



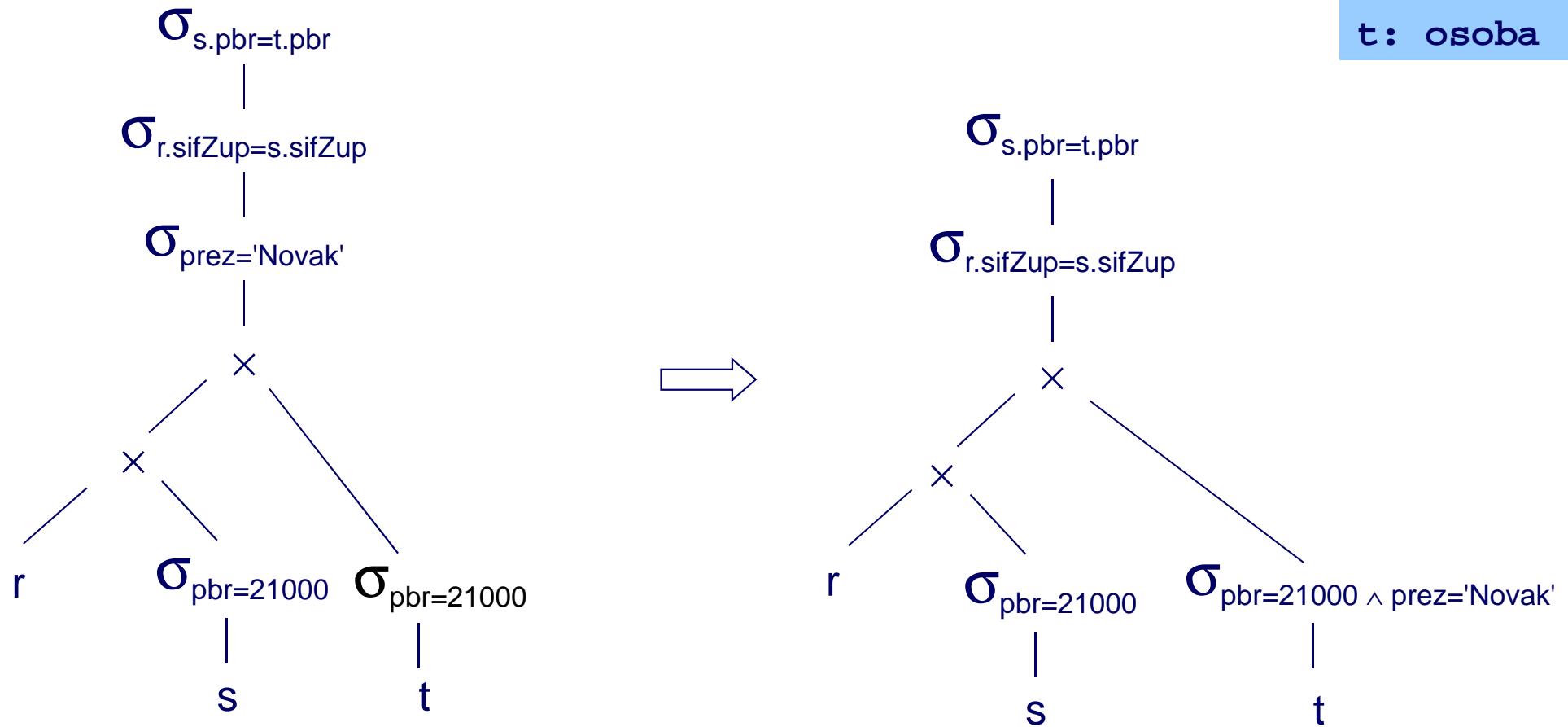
Primjer

r: zup
s: mjesto
t: osoba



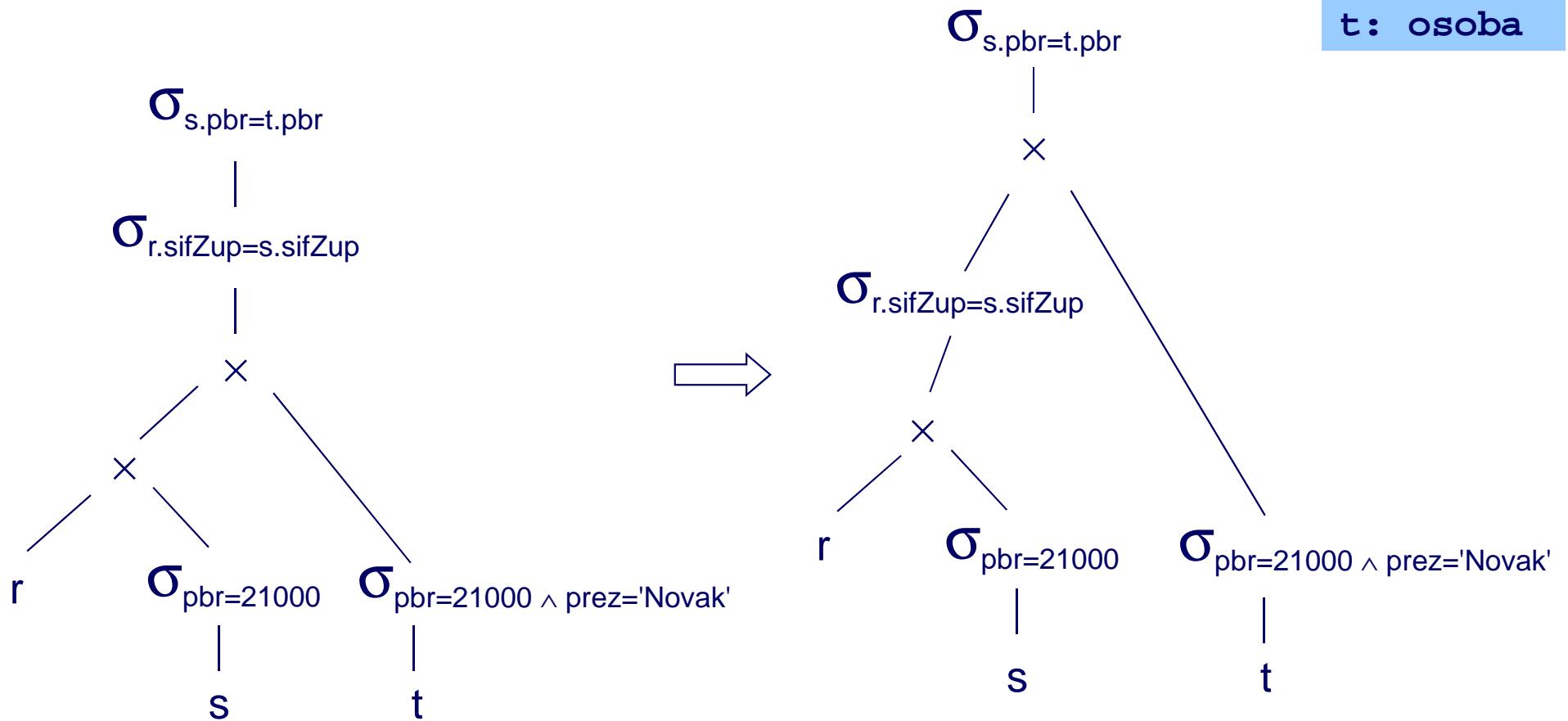
Primjer

r: zup
s: mjesto
t: osoba

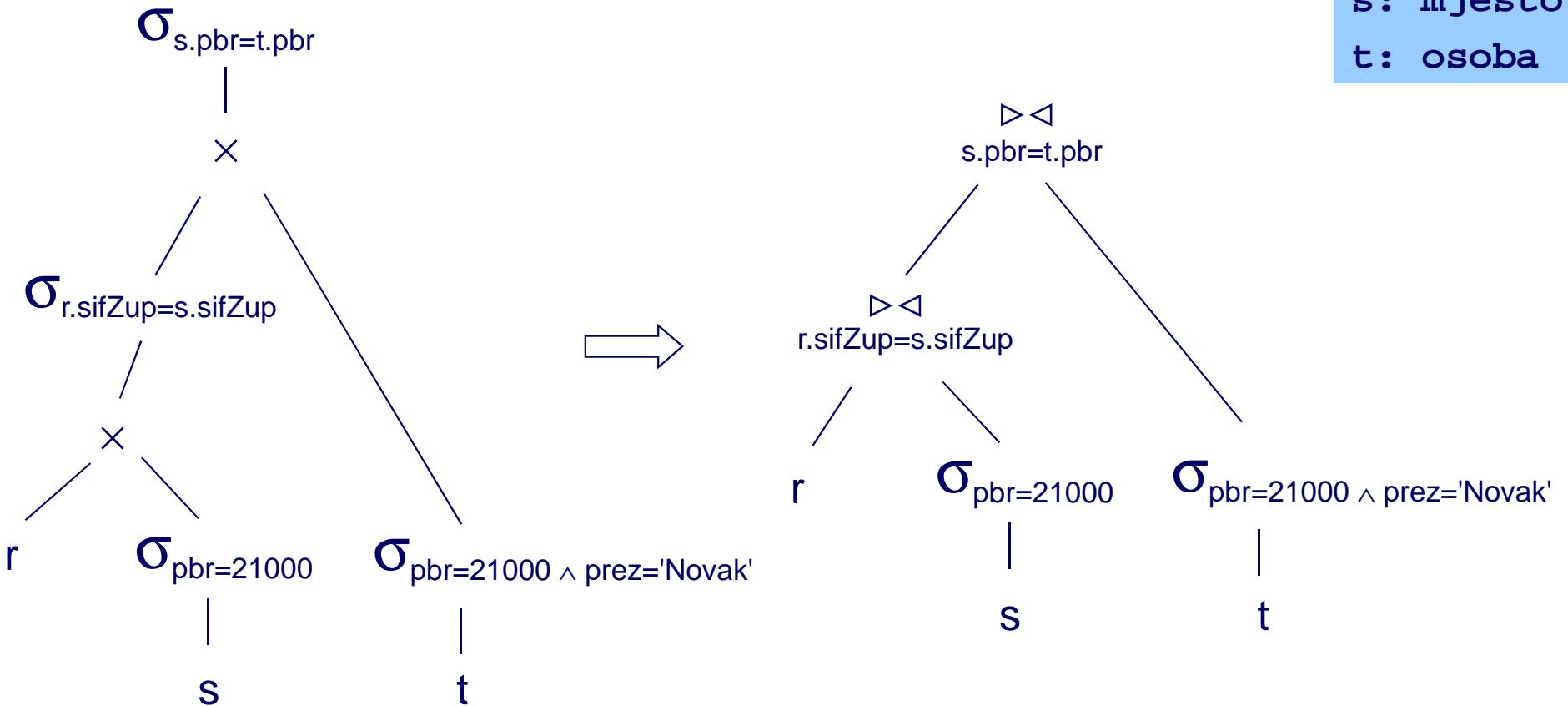


Primjer

r: zup
s: mjesto
t: osoba

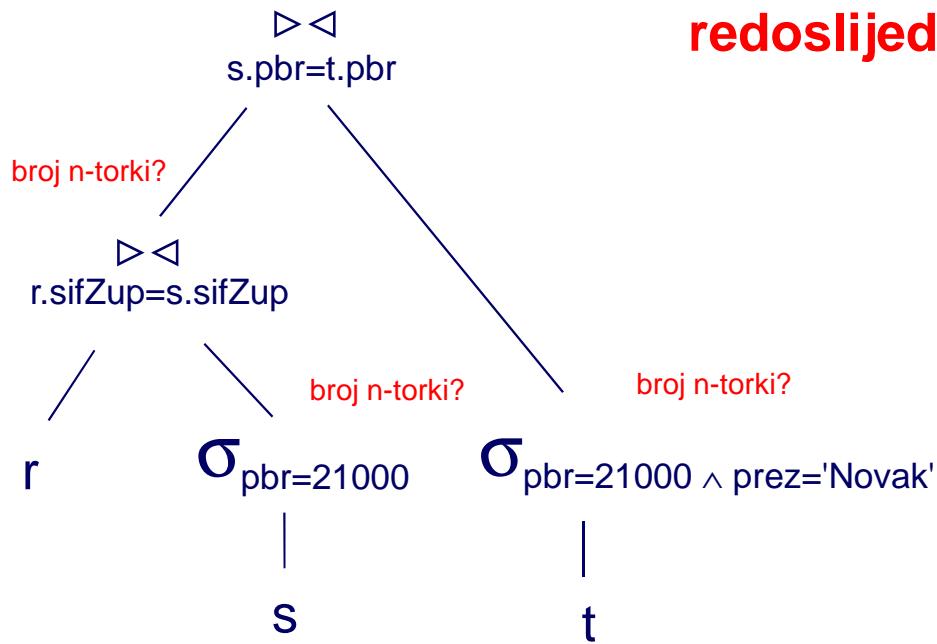


Primjer

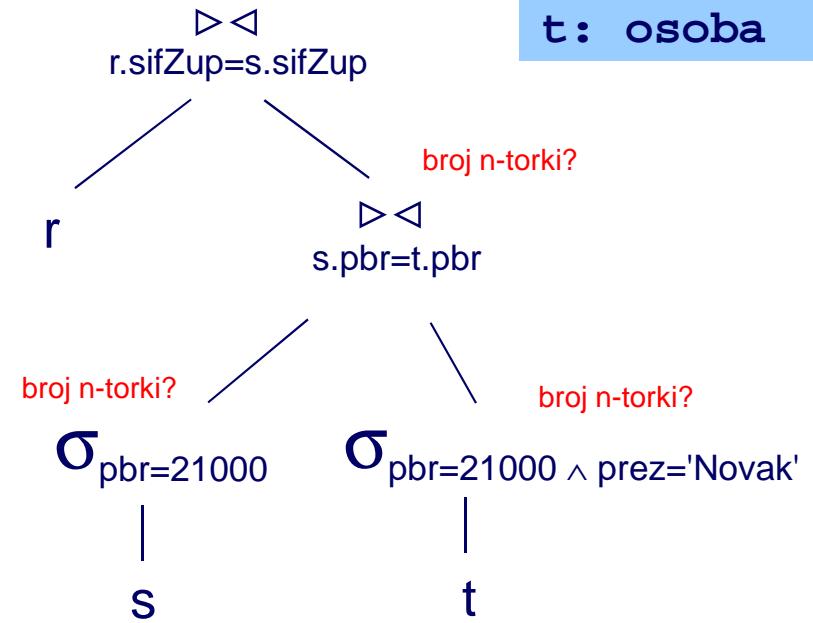


Primjer

r: zup
s: mjesto
t: osoba



11



- **za vježbu!** Radi pojednostavljenja, pri procjeni zanemariti sve troškove osim troškova pisanja međurezultata. Pretpostaviti da se svaki međurezultat mora materijalizirati. Zanemariti veličinu n-torke te trošak izraziti u broju n-torki. Usporediti ukupni broj n-torki u međurezultatima ovih planova. Na raspolaganju su sljedeći podaci iz rječnika podataka:

$$PK(r) = sifZup$$

$$PK(s) = p_{br}$$

$$PK(t) = sifOso$$

$$N(r) = 20$$

$$N(s) = 2\,000$$

$$N(t) = 200\ 000$$

$$V(p_{br}, t) = 10$$

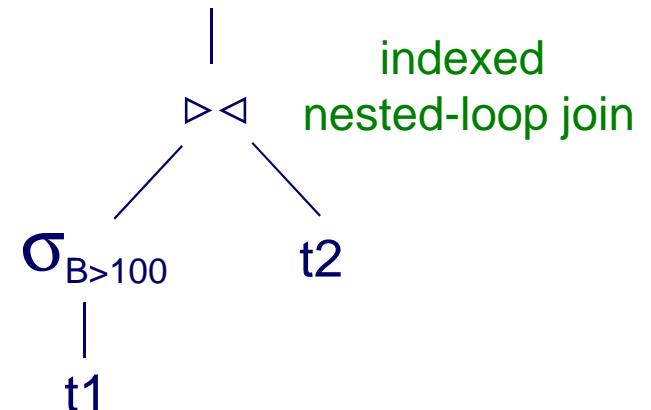
$$V(\text{pres}, t) = 50$$

Upute optimizatoru (*optimizer hints, optimizer directives*)

- mogućnost koju treba koristiti vrlo štedljivo jer:
 - uvođenjem uputa optimizatoru odgovornost za "optimalno" izvršavanje upita prebacuje se sa SUBP-a na administratora (ili programera ili korisnika)
 - ne postoji standardna sintaksa

Primjer:

```
SELECT *
  FROM t1 JOIN t2
    ON t1.A = t2.A
   WHERE t1.B > 100;
```



- postoji indeks nad atributom `t2.a`
- ako optimizator procijeni da će uvjet `t1.B > 100` vratiti veliki broj n-torki iz t1, odustat će od *indexed nested-loop join* i primijeniti *hash join*. Zašto?
- ako administrator ili programer (ili vrlo napredni korisnik), prema značenju podataka ocijeni da će rezultat selekcije ipak sadržavati relativno malen broj n-torki, može uz SELECT naredbu optimizatoru uputiti sljedeću direktivu:

```
SELECT {+USE_NL(t2)} *
  FROM t1 JOIN t2
    ON t1.A = t2.A
   WHERE t1.B > 100;
```

- uputa: spajanje s t2 obavi pomoću *nested-loop join*

Primjer prikaza plana izvršavanja (IBM IDS)

```
SELECT *
  FROM ispit
    , stud
    , mjesto
 WHERE mjesto.pbr = stud.pbrStan
   AND stud.mbr = ispit.mbrStud
   AND mjesto.pbr = 10805
   AND ispit.ocjena = 1;
```

Estimated Cost: 8409

Estimated # of Rows Returned: 42

1) informix.stud: SEQUENTIAL SCAN
Filters: informix.stud.pbrstan = 10805

2) informix.mjesto: SEQUENTIAL SCAN
Filters:
Table Scan Filters: informix.mjesto.pbr = 10805

DYNAMIC HASH JOIN
Dynamic Hash Filters: informix.mjesto.pbr = informix.stud.pbrstan
3) informix.ispit: SEQUENTIAL SCAN
Filters:
Table Scan Filters: informix.ispit.ocjena = 1

DYNAMIC HASH JOIN (Build Outer)
Dynamic Hash Filters: informix.stud.mbr = informix.ispit.mbrstud

Primjer prikaza plana izvršavanja (IBM IDS)

- Procijenjeni trošak: **8409**
- Procijenjeni broj n-torki u rezultatu: **42**
- (1) *table scan* za stud, uz filter pbrstan = 10805 → međurezultat 1
- (2) *table scan* za mjesto, uz filter pbr = 10805 → međurezultat 2
 - prethodna dva međurezultata (1 i 2) spoji pomoću *hash join* → međurezultat 3
- (3) *table scan* za ispit, uz filter ocjena = 1 → međurezultat 4
 - prethodna dva međurezultata (3 i 4) spoji pomoću *hash join*

Primjer prikaza plana izvršavanja (IBM IDS)

Query statistics:

Table map :

Internal name	Table name
t1	stud
t2	mjesto
t3	ispit

type	table	rows_prod	est_rows	rows_scan	time	est_cost	
scan	t1	8	16	5000	00:00.01	369	
type	table	rows_prod	est_rows	rows_scan	time	est_cost	
scan	t2	1	1	1000	00:00.00	46	
type	rows_prod	est_rows	rows_bld	rows_prb	novrflo	time	est_cost
hjoin	8	9	1	8	0	00:00.01	417
...							

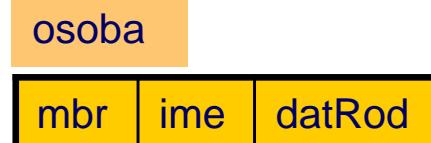
Primjer prikaza plana izvršavanja (IBM IDS)

- nakon izvršavanja upita, optimizator u izvještaj o planu izvršavanja dodaje i statističke podatke o upitu, npr.
- za dio koji se odnosi na operaciju *table scan* nad relacijom stud
 - procijenjeni trošak: 369
 - procijenjeni broj n-torki u rezultatu: 16
 - stvarni broj n-torki u rezultatu (poznato tek nakon izvršavanja upita): 8
 - broj pročitanih n-torki: 5000

Funkcijski indeks

- koristi li se indeks `idxDat` u sljedećem upitu?

```
CREATE INDEX idxDat ON osoba (datRod);
SELECT * FROM osoba
    WHERE DAY(datRod) BETWEEN 30 AND 31;
```



- NE, slijedna pretraga

- postoji rješenje - koristiti funkcijski indeks:

```
CREATE FUNCTION danMjeseca(datum DATE)
    RETURNING INTEGER WITH(NOT VARIANT)
    RETURN DAY(datum);
END FUNCTION;
CREATE INDEX idxDat ON osoba (danMjeseca(datRod));
SELECT * FROM osoba
    WHERE danMjeseca(datRod) BETWEEN 30 AND 31;
```

- NOT VARIANT: znači da funkcija za isti ulazni argument UVIJEK vraća isti rezultat
- IBM IDS: ugrađene funkcije (npr. DAY) ne mogu se direktno koristiti za izgradnju funkcijskog indeksa, ali se može načiniti SPL "omotač", kao u primjeru

Pravila za transformaciju izraza relacijske algebre

- prošireni popis pravila za transformaciju izraza relacijske algebre, koji je prikazan na ovoj i sljedećim stranicama, ove godine neće biti predmet provjere znanja
- oznake
 - relacije $r(R)$, $s(S)$, $t(T)$, ...
 - formule F , F_1 , F_2 , G , ...
 - skupovi atributa L_1 , L_2 , M , N , ...

1. Pravila vezana uz operaciju selekcije

$$1.1. \quad \sigma_{F_1 \wedge F_2}(r) \equiv \sigma_{F_1}(\sigma_{F_2}(r))$$

$$1.2. \quad \sigma_{F_1 \vee F_2}(r) \equiv \sigma_{F_1}(r) \cup \sigma_{F_2}(r)$$

$$1.3. \quad \sigma_{F_1}(\sigma_{F_2}(r)) \equiv \sigma_{F_2}(\sigma_{F_1}(r))$$

$$1.4. \quad \sigma_F(r \cup s) \equiv \sigma_F(r) \cup \sigma_F(s)$$

$$1.5. \quad \sigma_F(r \setminus s) \equiv \sigma_F(r) \setminus \sigma_F(s) \equiv \sigma_F(r) \setminus s$$

Pravila za transformaciju izraza relacijske algebre

- ako F sadrži samo attribute iz R

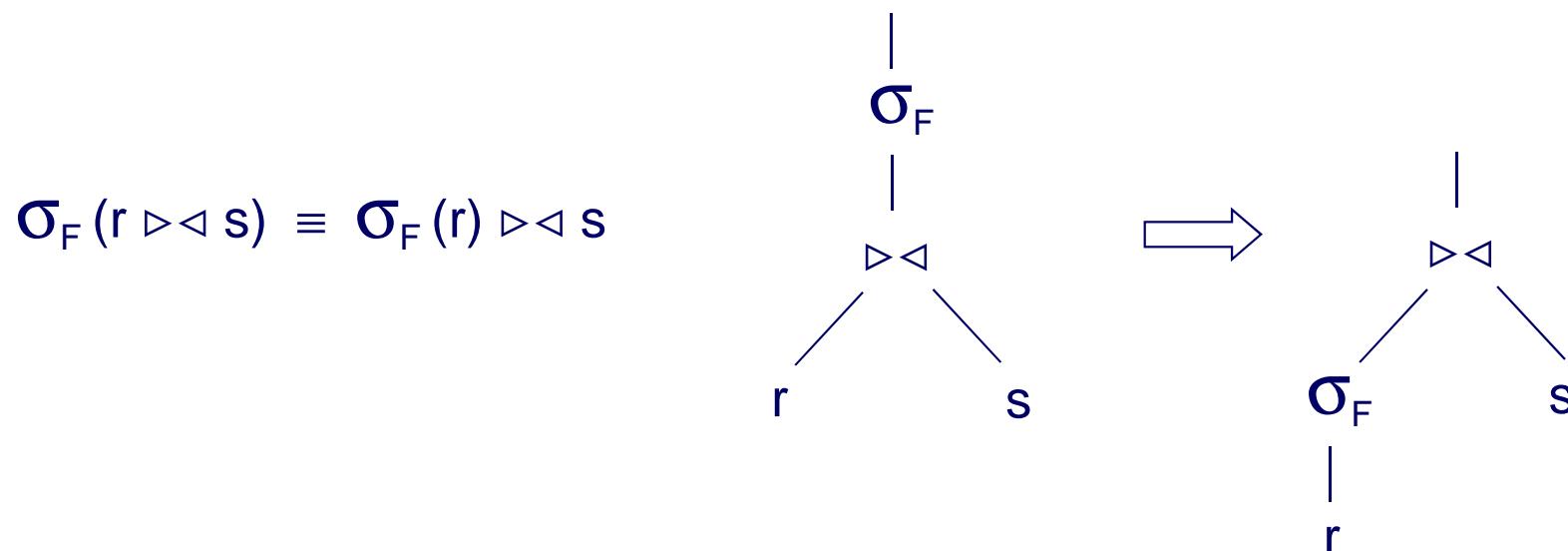
$$1.6.1. \quad \sigma_F(r \times s) \equiv \sigma_F(r) \times s$$

$$1.6.2. \quad \sigma_F(r \triangleright\!\triangleleft s) \equiv \sigma_F(r) \triangleright\!\triangleleft s$$

$$1.6.3. \quad \sigma_F(r \triangleright\!\triangleleft_{\textcolor{red}{G}} s) \equiv \sigma_F(r) \triangleright\!\triangleleft_{\textcolor{red}{G}} s$$

$$1.6.4. \quad \sigma_F(r \cap s) \equiv \sigma_F(r) \cap s$$

Primjer: često korištena transformacija: potiskivanje selekcije



Pravila za transformaciju izraza relacijske algebre

- slično, ako F_1 sadrži samo attribute iz R , a F_2 sadrži samo attribute iz S

$$1.7.1. \quad \sigma_{F_1 \wedge F_2}(r \times s) \equiv \sigma_{F_1}(r) \times \sigma_{F_2}(s)$$

$$1.7.2. \quad \sigma_{F_1 \wedge F_2}(r \triangleright\!\triangleleft s) \equiv \sigma_{F_1}(r) \triangleright\!\triangleleft \sigma_{F_2}(s)$$

$$1.7.3. \quad \sigma_{F_1 \wedge F_2}(r \triangleright\!\triangleleft_G s) \equiv \sigma_{F_1}(r) \triangleright\!\triangleleft_G \sigma_{F_2}(s)$$

$$1.7.4. \quad \sigma_{F_1 \wedge F_2}(r \cap s) \equiv \sigma_{F_1}(r) \cap \sigma_{F_2}(s)$$

- slično, ako i R i S sadrže sve attribute iz F

$$1.8.1. \quad \sigma_F(r \times s) \equiv \sigma_F(r) \times \sigma_F(s)$$

$$1.8.2. \quad \sigma_F(r \triangleright\!\triangleleft s) \equiv \sigma_F(r) \triangleright\!\triangleleft \sigma_F(s)$$

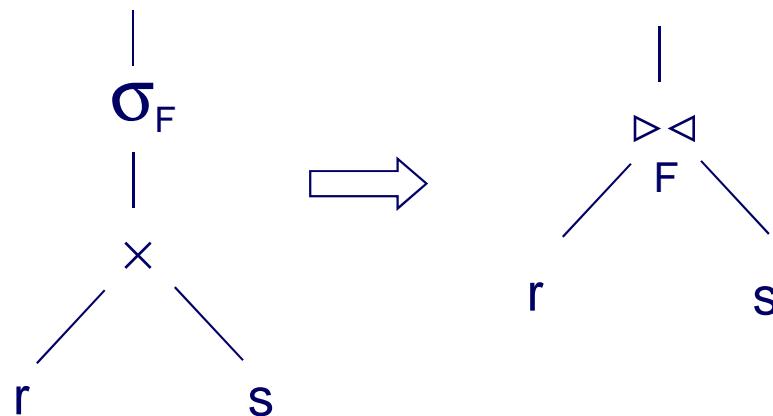
$$1.8.3. \quad \sigma_F(r \triangleright\!\triangleleft_G s) \equiv \sigma_F(r) \triangleright\!\triangleleft_G \sigma_F(s)$$

$$1.8.4. \quad \sigma_F(r \cap s) \equiv \sigma_F(r) \cap \sigma_F(s)$$

Pravila za transformaciju izraza relacijske algebre

- važno pravilo o selekciji, Kartezijevom produktu i spajanju
 - ako F odgovara uvjetu spajanja (θ -join) među r i s , tada

$$1.9. \quad \sigma_F(r \times s) \equiv r \triangleright\!\triangleleft_{F}^{} s$$



Pravila za transformaciju izraza relacijske algebre

2. Pravila vezana uz operaciju projekcije

- 2.1 - 2.3: operator projekcije se može dodati na bilo koje mjesto u stablu upita, pod uvjetom da se u "višim dijelovima stabla" ne koriste atributi koji su eliminirani operacijom projekcije

$$2.1. \quad \pi_L(r \times s) \equiv \pi_L(\pi_M(r) \times \pi_N(s))$$

- uz uvjet da je $L \subseteq M \cup N$, $M \subseteq R$, $N \subseteq S$

$$2.2. \quad \pi_L(r \triangleright\!\triangleleft s) \equiv \pi_L(\pi_M(r) \triangleright\!\triangleleft \pi_N(s))$$

- uz uvjet da je $L \subseteq M \cup N$, $M \supseteq R \cap S$, $N \supseteq R \cap S$

$$2.3. \quad \pi_L(r \triangleright\!\triangleleft_{F\!} s) \equiv \pi_L(\pi_M(r) \triangleright\!\triangleleft_F \pi_N(s))$$

- uz uvjet da je $L \subseteq M \cup N$, M sadrži atribute iz R koji se spominju u F, N sadrži atribute iz S koji se spominju u F

Pravila za transformaciju izraza relacijske algebre

2.4. $\pi_L(\pi_M(r)) \equiv \pi_L(r)$

2.5. $\pi_L(r \cup s) \equiv \pi_L(r) \cup \pi_L(s)$

2.6. $\pi_L(\sigma_F(r)) \equiv \sigma_F(\pi_L(r))$

- uz uvjet da F sadrži samo attribute iz L

Pravila za transformaciju izraza relacijske algebre

3. Komutativnost

$$3.1. \quad r \cup s \equiv s \cup r$$

$$3.2. \quad r \cap s \equiv s \cap r$$

$$3.3. \quad r \times s \equiv s \times r$$

$$3.4. \quad r \triangleright\triangleleft s \equiv s \triangleright\triangleleft r$$

$$3.5. \quad r \triangleright\triangleleft s \equiv s \triangleright\triangleleft r$$

$\text{F} \qquad \qquad \text{F}$

Pravila za transformaciju izraza relacijske algebre

4. Asocijativnost

$$4.1. \quad (r \cup s) \cup t \equiv r \cup (s \cup t)$$

$$4.2. \quad (r \cap s) \cap t \equiv r \cap (s \cap t)$$

$$4.3. \quad (r \times s) \times t \equiv r \times (s \times t)$$

$$4.4. \quad (r \triangleright\triangleleft s) \triangleright\triangleleft t \equiv r \triangleright\triangleleft (s \triangleright\triangleleft t)$$

$$4.5. \quad (r \triangleright\triangleleft s) \triangleright\triangleleft t \equiv r \triangleright\triangleleft (s \triangleright\triangleleft t)$$

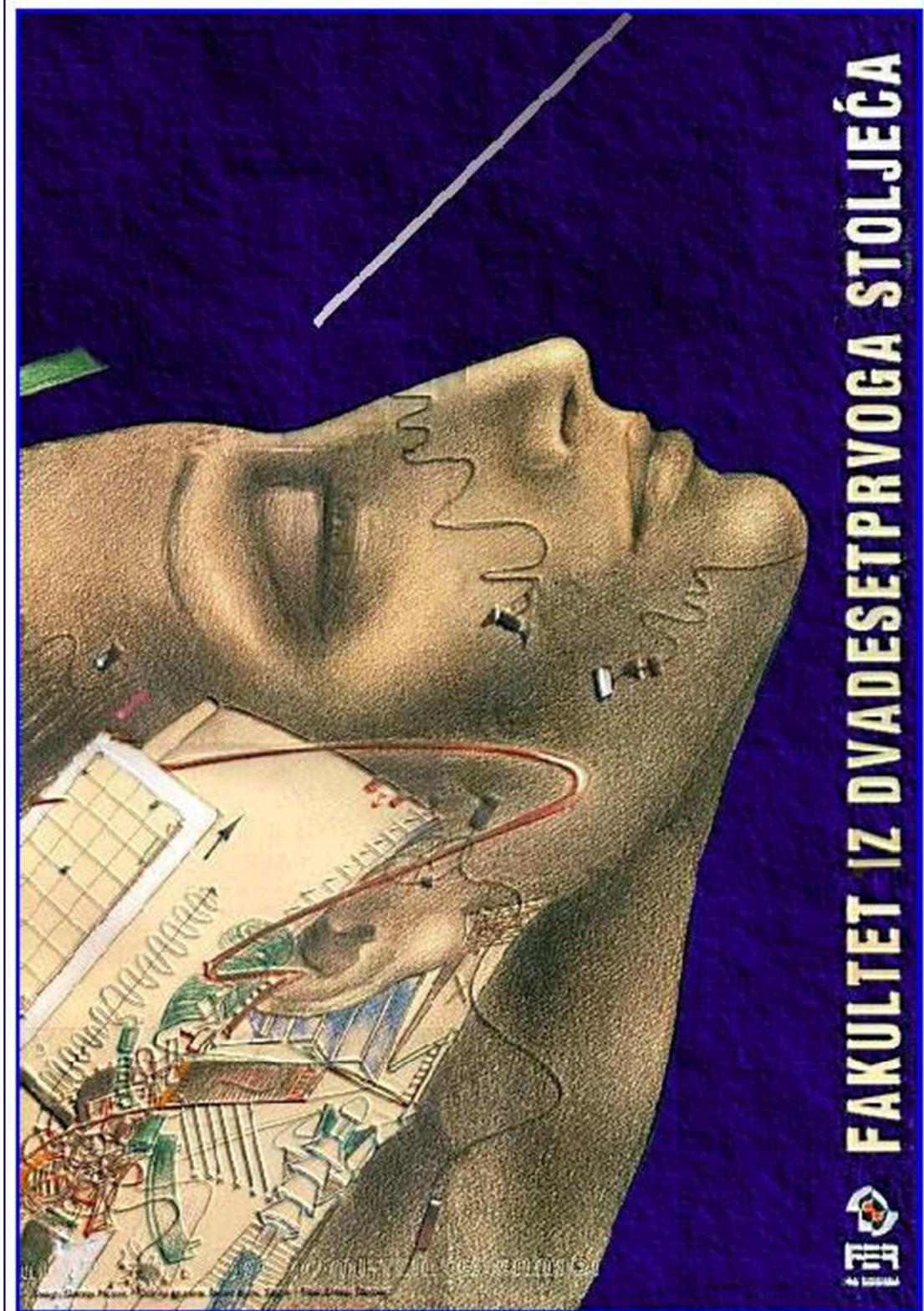
$F_1 \qquad F_2 \qquad F_1 \qquad F_2$

Sustavi baza podataka

Predavanja

5. Transakcija

travanj 2014.



FAKULTET IZ DVADESETPRVOGA STOLJEĆA

Transakcija

- Izvršavanje programa koji pristupa bazi podataka radi obavljanja neke administrativne (poslovne) funkcije
Bernstein, Newcomer: Principles of Transaction Processing
- Niz SQL naredbi koje mijenjaju podatke i koje se moraju izvršiti u cijelosti ili se (u cijelosti) ne smiju izvršiti
IBM Informix Guide to SQL: Tutorial
- Logička jedinica posla (*logical unit of work*) koja sadrži jednu ili više SQL naredbi. Transakcija grupira SQL naredbe tako da su te naredbe potvrđene (primijenjene na bazi podataka), ili poništene.

Oracle 11g Release 2 (11.2)

Definicija: Transakcija je niz logički povezanih operacija koje se izvršavaju kao cjelina i prevode bazu podataka iz jednog u drugo **konzistentno stanje***

* baza podataka je konzistentna ako zadovoljava sva definirana integritetska ograničenja

Transakcija

Primjer:

- zadani iznos sredstava prebaciti s prvog zadanog na drugi zadani račun (stanje prvog računa umanjiti, a drugog računa uvećati).

racun	
brRac	stanje
101	1000.00
102	500.00
103	2500.00
104	200.00

Logički povezane operacije:

umanjiti stanje na prvom računu (SQL: UPDATE)
uvećati stanje na drugom računu (SQL: UPDATE)

Ili obje UPDATE operacije moraju biti obavljene, ili niti jedna!

Sintaksa - sa stanovišta korisnika

- transakcija se opisuje kao program ili odsječak programa, SPL, Java+JDBC, C+ODBC, itd, kojim se izvršavaju SQL naredbe:
 - **SELECT, UPDATE, INSERT, DELETE, ...**
 - **BEGIN [WORK], COMMIT [WORK], ROLLBACK [WORK]**
 - **BEGIN [WORK]** nije u skladu s ISO/ANSI SQL standardom
 - ISO/ANSI SQL: **START TRANSACTION**
- transakcije se ne mogu ugnježđivati - svaka korisnička sjednica (*user session*), može u jednom trenutku imati najviše jednu aktivnu transakciju

Sintaksa - sa stanovišta korisnika

Primjer:

- napisati programski kôd za transakciju kojom se zadani iznos sredstava prebacuje s prvog zadanog na drugi zadani račun (stanje prvog računa umanjiti, a drugog računa uvećati).

variable:	brRac1	← broj prvog računa
	brRac2	← broj drugog računa
	iznos	← iznos kojeg treba prebaciti

racun	brRac	stanje
	101	1000.00
	102	500.00
	103	2500.00
	104	200.00

Sintaksa - ako se koristi implicitni početak transakcije

Oracle (PL/SQL) ili IBM IDS (SPL)

- **implicitni početak transakcije:** početkom transakcije smatra se prva SQL naredba izvršena u okviru korisničke sjednice ili prva SQL naredba izvršena neposredno nakon završetka prethodne transakcije
- **kraj transakcije:** mora biti eksplicitno zadan
 - COMMIT [WORK] ili ROLLBACK [WORK]
 - ako korisnička sjednica završi prije nego se obavi COMMIT ili ROLLBACK, podrazumijeva se ROLLBACK

Primjer:

```
-- započela je korisnička sjednica
--> ovdje se početak transakcije podrazumijeva
UPDATE racun SET stanje = stanje - iznos WHERE brRac = brRac1;
UPDATE racun SET stanje = stanje + iznos WHERE brRac = brRac2;
COMMIT WORK; -- ili ROLLBACK WORK
--> ovdje se početak transakcije podrazumijeva
sljedeća SQL naredba;
```

Sintaksa - ako se koristi eksplisitni početak transakcije

IBM IDS (SPL)

- **početak transakcije:** mora biti eksplisitno zadan
 - BEGIN [WORK]
- **kraj transakcije:** mora biti eksplisitno zadan
 - COMMIT [WORK] ili ROLLBACK [WORK]
 - ako korisnička sjednica završi prije nego se obavi COMMIT ili ROLLBACK, podrazumijeva se ROLLBACK

Primjer:

```
BEGIN WORK;  
UPDATE racun SET stanje = stanje - iznos WHERE brRac = brRac1;  
UPDATE racun SET stanje = stanje + iznos WHERE brRac = brRac2;  
COMMIT WORK;
```

Sintaksa - bez određivanja granica transakcija

IBM IDS (SPL)

- ako granice transakcija nisu određene (niti implicitnim početkom i eksplisitnim završetkom, niti eksplisitnim početkom i završetkom), smatra se da transakcija započinje neposredno prije početka, a završava neposredno nakon završetka svake zasebne SQL naredbe. Podrazumijeva se COMMIT ako je naredba uspješno završila, inače se podrazumijeva ROLLBACK.

Primjer:

```
-- ovdje se podrazumijeva početak transakcije (kao da je zaista obavljen BEGIN)
UPDATE racun SET stanje = stanje - iznos WHERE brRac = brRac1;
-- ovdje se podrazumijeva kraj transakcije i to:
    COMMIT ako je SQL naredba obavljena bez pogreške, ROLLBACK inače

-- ovdje se podrazumijeva početak transakcije (kao da je zaista obavljen BEGIN)
UPDATE racun SET stanje = stanje - iznos WHERE brRac = brRac1;
-- ovdje se podrazumijeva kraj transakcije i to:
    COMMIT ako je SQL naredba obavljena bez pogreške, ROLLBACK inače
```

- `setAutoCommit(false)` : implicitni početak transakcije. Kraj transakcije i ovdje mora biti eksplisitno zadan

Java varijable:

<code>brRac1</code>	← broj prvog računa
<code>brRac2</code>	← broj drugog računa
<code>iznos</code>	← iznos kojeg treba prebaciti
<code>conn</code>	← <i>Connection</i> objekt

```
Statement stmt = conn.createStatement();
conn.setAutoCommit(false);

// transakcija započinje prvom izvršnom SQL naredbom
stmt.executeUpdate("UPDATE racun SET stanje = stanje - " + iznos +
                    " WHERE brRac = " + brRac1 );
stmt.executeUpdate("UPDATE racun SET stanje = stanje + " + iznos +
                    " WHERE brRac = " + brRac2 );
conn.commit(); // ili conn.rollback()

// pozivom sljedeće execute... metode započinje nova transakcija
...
```

- `setAutoCommit(true)` : granice transakcija nisu određene. Svaka DML naredba predstavlja zasebnu transakciju. Sljedeći odsječak stoga ne garantira konzistentnost!

Java varijable:

<code>brRac1</code>	← broj prvog računa
<code>brRac2</code>	← broj drugog računa
<code>iznos</code>	← iznos kojeg treba prebaciti
<code>conn</code>	← <i>Connection</i> objekt

```
Statement stmt = conn.createStatement();
conn.setAutoCommit(true);

// ovdje se podrazumijeva BEGIN
stmt.executeUpdate("UPDATE racun SET stanje = stanje - " + iznos +
                   " WHERE brRac = " + brRac1 );
// ovdje se podrazumijeva COMMIT/ROLLBACK

// ovdje se podrazumijeva BEGIN
stmt.executeUpdate("UPDATE racun SET stanje = stanje + " + iznos +
                   " WHERE brRac = " + brRac2 );
// ovdje se podrazumijeva COMMIT/ROLLBACK
```

Sintaksa - sa stanovišta SUBP-a

- sa stanovišta SUBP-a, transakcija je niz operacija čitanja i/ili pisanja koje su omeđene transakcijskim operacijama
 - operacije čitanja ili pisanja (*database operations*)
 - **read(x, p)**, **write(y, p)**
 - u varijablu p učitaj vrijednost elementa x, u element y upiši vrijednost varijable p
 - transakcijske operacije (*transaction operations*)
 - **start**: početak nove transakcije
 - **commit**: završetak transakcije uz trajnu pohranu svih efekata transakcije
 - **abort**: prekid transakcije (kojeg prati poništavanje svih efekata transakcije)

```
start;  
read(x, p);  
write(x, r);  
commit;
```

```
r[x], w[x], c
```

Model transakcije

$r[x] \rightarrow w[x] \rightarrow c$

ACID svojstva transakcije

- SUBP mora osigurati sljedeća svojstva transakcija:

Atomicity Atomarnost

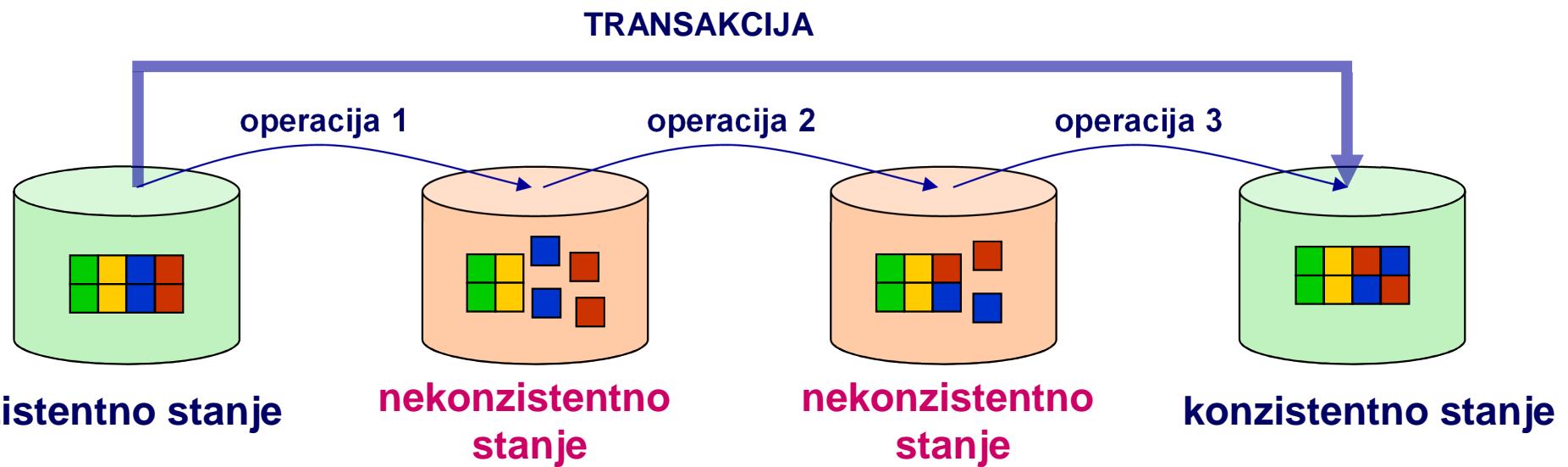
Consistency Konzistentnost

Isolation Izolacija

Durability Izdržljivost

Konzistentnost

- transakcija je **korektna** ako konzistentnu bazu podataka, u odsustvu drugih transakcija i pogrešaka*, prevodi iz jednog u drugo konzistentno stanje
 - * pogreške će se detaljnije razmatrati u poglavlju o obnovi sustava



- Konzistentnost (Consistency)** - transakcija zadovoljava svojstvo konzistentnosti ako je korektna

Konzistentnost

Primjer:

- prebaciti zadani **iznos** sredstava s računa **brRac1** na račun **brRac2**
- korektna transakcija

```
BEGIN WORK;  
UPDATE racun SET stanje = stanje - iznos WHERE brRac = brRac1;  
UPDATE racun SET stanje = stanje + iznos WHERE brRac = brRac2;  
COMMIT WORK;
```

- nije korektna transakcija (posljedica npr. logičke pogreške u programu)

```
BEGIN WORK;  
UPDATE racun SET stanje = stanje - iznos WHERE brRac = brRac1;  
UPDATE racun SET stanje = stanje - iznos WHERE brRac = brRac2;  
COMMIT WORK;
```

Konzistentnost

- na koji način osigurati svojstvo konzistentnosti?
- osigurati korektnost transakcije:
 - kad god je moguće koristiti eksplicitno opisana integritetska ograničenja
 - CHECK, PRIMARY, UNIQUE, FOREIGN KEY
 - okidači
 - pohranjene procedure
 - odgovornost programera
 - ili naprednog korisnika

Izolacija

Primjer:

- početno stanje računa A je 1000 kn. Jedna iza druge se obavljaju dvije transakcije
- transakcija kojom se 600 kn isplaćuje s računa A

```
BEGIN WORK;  
SELECT stanje INTO p1 FROM racun WHERE brRac = 'A';  
LET p1 = p1 - 600;  
UPDATE racun SET stanje = p1 WHERE brRac = 'A';  
COMMIT WORK;
```

novo stanje na računu A: 400 kn

- transakcija kojom se 2000 kn uplaćuje na račun A

```
BEGIN WORK;  
SELECT stanje INTO p2 FROM racun WHERE brRac = 'A';  
LET p2 = p2 + 2000;  
UPDATE racun SET stanje = p2 WHERE brRac = 'A';  
COMMIT WORK;
```

novo stanje na računu A: 2400 kn

- transakcije su korektne - nakon njihovog obavljanja baza podataka je konzistentna

Izolacija

- ako SUBP operacije tih istih transakcija izvodi istodobno (naizmjence operacije jedne i druge transakcije)

```
BEGIN WORK;
SELECT stanje INTO p1 FROM racun WHERE brRac = 'A';
LET p1 = p1 - 600;
BEGIN WORK;
SELECT stanje INTO p2 FROM racun WHERE brRac = 'A';
LET p2 = p2 + 2000;
UPDATE racun SET stanje = p2 WHERE brRac = 'A';      -- stanje=3000
UPDATE racun SET stanje = p1 WHERE brRac = 'A';      -- stanje=400
COMMIT WORK;
COMMIT WORK;
```

novo stanje na računu A: 400 kn

- unatoč tome što su pojedinačne transakcije korektne, ako se njihove operacije obavljaju naizmjence, mogu narušiti konzistentnost baze podataka
- **Izolacija (*Isolation*)** - učinak transakcija koje se obavljaju istodobno (paralelno) mora biti jednak učinku tih istih transakcija koje bi se obavljale jedna iza druge

Atomarnost

Primjer:

- prebaciti zadani **iznos** sredstava s računa **brRac1** na račun **brRac2**

```
BEGIN WORK;  
UPDATE racun SET stanje = stanje - iznos WHERE brRac = brRac1;  
UPDATE racun SET stanje = stanje + iznos WHERE brRac = brRac2;  
COMMIT WORK;
```

Primjetiti: baza podataka u ovom trenutku nije konzistentna

- bez obzira na moguće događaje tijekom izvršavanja transakcije (nestanak napajanja, kvar računala, ...) dopušten je samo jedan od dva ishoda:
 - efekti obje naredbe pohranjeni su u bazi podataka
 - baza podataka je u stanju u kakovom je bila prije početka transakcije
- Atomarnost (Atomicity)** - ili su efekti svih operacija transakcije pravilno pohranjeni u bazi podataka ili transakcija nije utjecala na stanje baze podataka
 - all or nothing*

Izdržljivost

- neka je sljedeća transakcija uspješno obavljena, tj. njezini efekti su u cijelosti pohranjeni u bazi podataka:

```
BEGIN WORK;  
UPDATE racun SET stanje = stanje - iznos WHERE brRac = brRac1;  
UPDATE racun SET stanje = stanje + iznos WHERE brRac = brRac2;  
COMMIT WORK;
```

- ako se nakon obavljanja naredbe COMMIT dogodi kvar medija (*disk failure*) na kojem je pohranjena baza podataka, efekti potvrđene transakcije, koju korisnik smatra obavljrenom, mogli bi biti izgubljeni
- Izdržljivost (Durability)** - izmjene u bazi podataka koje su rezultat potvrđenih transakcija ne smiju biti izgubljene neovisno o vrsti kvara sustava koji bi se mogao dogoditi nakon što je transakcija potvrđena

Osiguravanje ACID svojstava transakcije

- svojstvo konzistentnosti transakcije (iz skupa ACID svojstava) odnosi se samo na korektnost transakcije (iako na konzistentnost baze podataka itekako može utjecati narušavanje svojstava atomarnosti i izolacije)
- svojstva atomarnosti i izdržljivosti osigurava poseban dio SUBP-a: sustav za obnovu
- svojstvo izolacije osigurava poseban dio SUBP-a: sustav za kontrolu istodobnog pristupa
- programski kôd svake pojedinačne transakcije bi bio izuzetno složen ako bi se u svakoj transakciji zasebno implementirala svojstva atomarnosti, izdržljivosti i izolacije

Transactions relieve the application programmer of handling many error conditions. If things get too complicated, the programmer calls AbortTransaction which cleans up the state by resetting everything back to the beginning of the transaction.

J. Gray. Why do computers stop and what can be done about it. Invited paper, 5th Symposium on Reliability in distributed software and database systems. Los Angeles, California. 1986.

Dodatak uz svojstvo konzistentnosti: odgođena provjera integritetskih ograničenja

- baza podataka može biti u nekonzistentnom stanju tijekom obavljanja transakcije
 - što se za to vrijeme dešava s ugrađenim integritetskim ograničenjima?
- provjera integritetskih ograničenja može se odgoditi do kraja transakcije SQL naredbama:

```
SET CONSTRAINTS constrName { IMMEDIATE | DEFERRED }
SET CONSTRAINTS ALL { IMMEDIATE | DEFERRED }
```
- **IMMEDIATE**: integritetska ograničenja provjeravaju se nakon svake izvedene DML naredbe (ne za vrijeme obavljanja DML naredbe!). Ako konačni rezultat DML naredbe narušava integritetska ograničenja, efekti DML naredbe se automatski poništavaju (*statement-level rollback*)
- **DEFERRED**: integritetska ograničenja se provjeravaju pri izvršavanju naredbe **COMMIT**
 - ako se utvrdi da je neko integritetsko ograničenje narušeno, transakcija se poništava

Dodatak uz svojstvo konzistentnosti: odgođena provjera integritetskih ograničenja

Primjer:

Svaki nastavnik predaje u točno jednoj grupi. Zamijeniti grupe za nastavnike 107 i 102

nastava	sifNast	sifGrupa
	107	1.01
	102	1.02
	105	1.03

```
CREATE TABLE nastava ( IBM IDS
    sifNast      INTEGER
    , sifGrupa   CHAR(4)
    , PRIMARY KEY (sifNast)
        CONSTRAINT pkNastava
    , UNIQUE (sifGrupa)
        CONSTRAINT unqNastava
);
```

```
CREATE TABLE nastava ( Oracle
    sifNast      INTEGER
    , sifGrupa   CHAR(4)
    , CONSTRAINT pkNastava
        PRIMARY KEY (sifNast)
    , CONSTRAINT unqNastava
        UNIQUE (sifGrupa) DEFERRABLE
);
```

```
BEGIN WORK;
UPDATE nastava SET sifGrupa = '1.02'
    WHERE sifNast = 107; → POGREŠKA
UPDATE nastava SET sifGrupa = '1.01'
    WHERE sifNast = 102;
COMMIT WORK;
```

```
BEGIN WORK;
SET CONSTRAINTS unqNastava DEFERRED;
UPDATE nastava SET sifGrupa = '1.02'
    WHERE sifNast = 107;
UPDATE nastava SET sifGrupa = '1.01'
    WHERE sifNast = 102;
COMMIT WORK; → O.K.
```

```
BEGIN WORK;
SET CONSTRAINTS unqNastava DEFERRED;
UPDATE nastava SET sifGrupa = '1.02'
    WHERE sifNast = 107;
COMMIT WORK; → POGREŠKA i ROLLBACK
```

Prekid i poništavanje (*abort* i *rollback*) transakcije

- program ili korisnik obavlja SQL naredbu ROLLBACK*
 - u kontroliranim uvjetima utvrđeno je da transakcija ne može biti obavljena. Npr. u PL/SQL proceduri je utvrđeno da bi dalnjim obavljanjem transakcije došlo do narušavanja integritetskih ograničenja, procedurom se obavlja naredba ROLLBACK)
 - korisnik je odlučio putem SQL sučelja izazvati prekid transakcije
 - abnormalni prekid (korisničkog) programa: SUBP automatski obavlja naredbu ROLLBACK
 - prekid transakcije izazvan od strane sustava za upravljanje bazama podataka
 - abnormalni prekid rada SUBP-a (npr. nestanak napajanja). Sustav je prestao raditi, baza podataka je u nekonzistentnom stanju, ali će se efekti eventualno nezavršenih transakcija poništiti tijekom obnove, koja se obavlja prije nego sustav počne zaprimati nove transakcije.
 - sustav prekida i poništava jednu ili više transakcija tijekom razrješavanja potpunog zastoja
- * SQL naredba ROLLBACK [WORK] predstavlja zahtjev za prekidom transakcije (*abort*) nakon kojeg SUBP odmah obavlja i poništavanje efekata transakcije (*rollback*).

Primjer: Oracle PL/SQL

- zadani iznos sredstava prebacuje s prvog zadanog na drugi zadani račun (stanje prvog računa umanjiti, a drugog računa uvećati), uz integritetsko ograničenje: stanje niti jednog računa ne smije biti manje od nule

PL/SQL varijable:

```
brRac1      ← broj prvog računa  
brRac2      ← broj drugog računa  
iznos       ← iznos kojeg treba prebaciti  
stanje1    ← novo stanje na prvom računu
```

```
-- transakcija započinje prvom izvršnom SQL naredbom  
UPDATE racun SET stanje = stanje - iznos WHERE brRac = brRac1;  
UPDATE racun SET stanje = stanje + iznos WHERE brRac = brRac2;  
SELECT stanje INTO stanje1 FROM racun WHERE brRac = brRac1;  
IF stanje1 < 0.00 THEN  
    ROLLBACK;  
ELSE  
    COMMIT;  
END IF;
```

Primjer: JAVA + JDBC API

Java varijable:

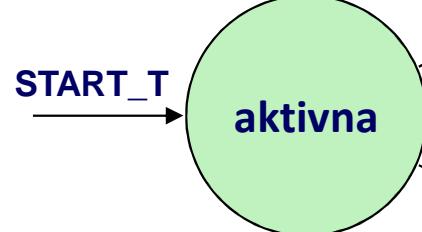
```
brRac1      ← broj prvog računa  
brRac2      ← broj drugog računa  
iznos       ← iznos kojeg treba prebaciti  
stanje1     ← novo stanje na prvom računu  
conn        ← Connection objekt
```

```
Statement stmt = conn.createStatement();  
conn.setAutoCommit(false);  
stmt.executeUpdate("UPDATE racun SET stanje = stanje - " + iznos +  
                   " WHERE brRac = " + brRac1 );  
stmt.executeUpdate("UPDATE racun SET stanje = stanje + " + iznos +  
                   " WHERE brRac = " + brRac2 );  
ResultSet rs = stmt.executeQuery("SELECT stanje FROM racun " +  
                                 " WHERE brRac = " + brRac1);  
rs.next();  
BigDecimal stanje1 = rs.getBigDecimal(1);  
if (stanje1.doubleValue() >= 0.0) {  
    conn.commit();  
} else {  
    conn.rollback();  
}
```

Dijagram stanja transakcije

- sa stanovišta SUBP-a, transakcija se u svakom trenutku nalazi u jednom od stanja prikazanih na slici

aktivna (active): operacije se obavljaju bez pogreške ili se dešavaju samo one pogreške kojima transakcija eksplicitno upravlja (aplikacijska logika) i za koje postoji kompenzacijnska akcija nakon koje se transakcija može nastaviti.



Ako ne postoji odgovarajuća kompenzacijnska akcija, aplikacija upućuje eksplicitni zahtjev za poništavanjem transakcije.

djelomično završena (partially committed): sve operacije osim naredbe za potvrđivanje transakcije su obavljene

potvrđena (committed): svi efekti transakcije su **trajno** zabilježeni u bazi podataka

završena (completed): u stanju potvrđena ili poništena

neispravna (failed): ustanovljena je pogreška zbog koje daljnje izvršavanje transakcija nije moguće

poništena (terminated): svi efekti operacija koje su obavljene u okviru transakcije su poništeni i baza podataka je vraćena u stanje prije početka transakcije

Dijagram stanja transakcije

- prekid transakcije (*abort*) - korisnik ili SUBP je prekinuo transakciju
 - poništavanje transakcije (*rollback*) - postupak kojim sustav poništava efekte transakcije
-
- naredbom END TRANSACTION (na slici END_T) sustavu se signalizira da su sve *read* i *write* operacije transakcije završene
 - provjere integriteta (ako se koristi odgođena provjera integriteta) obavljaju se za vrijeme dok je transakcija u stanju "djelomično završena"
 - najčešće se ne koriste posebne naredbe END TRANSACTION i ABORT
 - SQL naredbom COMMIT transakcija prelazi iz stanje "aktivna" u stanje "djelomično završena", a zatim (ako za potvrđivanje transakcije ne postoji prepreka) u stanje "potvrđena"
 - SQL naredbom ROLLBACK transakcija prelazi iz stanje "aktivna" u stanje "neispravna", a zatim u stanje "poništена"

Prekid i poništavanje transakcija na razini SQL naredbe

- neovisno o tome jesu li granice transakcije eksplicitno ili implicitno zadane, svaka pojedinačna SQL naredba se obavlja u cijelosti (ili se njezini efekti poništavaju u cijelosti)
- ako se SQL naredba ne može uspješno obaviti u cijelosti, obavlja se poništavanje na razini SQL naredbe (*statement-level rollback*)

Prekid i poništavanje transakcija na razini SQL naredbe

```
CREATE TABLE ispit (
    mbrSt  INTEGER
,  ocj    INTEGER
    CHECK (ocj BETWEEN 1 AND 5)
);
```

ispit	mbrSt	ocj
	101	3
	102	4
	103	5

```
UPDATE ispit SET ocj = ocj + 1;
```

neke ocjene su možda bile promijenjene tijekom izvršavanja naredbe, ali u trenutku kada je naredba prekinuta zbog pogreške (narušavanja integritetskog ograničenja), svi njezini efekti su poništeni

```
BEGIN WORK;
...
ostale SQL naredbe
UPDATE ispit SET ocj = ocj + 1;
...
ostale SQL naredbe
COMMIT ili ROLLBACK WORK;
```

već u ovom trenutku poništeni su efekti naredbe UPDATE - sve su ocjene vraćene na početne vrijednosti. Efekti ostalih SQL naredbi iz ove transakcije ovog trenutka još nisu poništeni. Efekti ostalih SQL naredbi mogu se poništiti obavljanjem naredbe ROLLBACK

Klasifikacija transakcija

- ***on-line transakcije***: obavljaju se u relativno kratkom vremenu (tipično do nekoliko sekundi) i pristupaju malom broju elemenata baze podataka
- ***batch transakcije***: trajanje takvih transakcija je izraženo u minutama ili satima, praćeno pristupanjem velikom broju elemenata baze podataka
- ***konverzacijske ili interaktivne transakcije***: za vrijeme obavljanja transakcije postoji interakcija s korisnikom

Atomarnost transakcije i interakcija s "vanjskim svijetom"

- budući da su operacije pisanja koje se obavljaju tijekom transakcije pod nadzorom SUBP-a, njihove efekte SUBP po potrebi može i poništiti (pojednostavljeno: vraćanjem starih vrijednosti elemenata)
- ako transakcija obavlja operacije koje utječu na okolinu SUBP-a (terminal ili slična U/I naprava) svojstvo atomarnosti transakcije može biti dovedeno u pitanje

Atomarnost transakcije i interakcija s "vanjskim svijetom"

Primjer:

- što bi se moglo dogoditi kada bi bankomat radio ovako?

```
BEGIN WORK
IF (SELECT stanje FROM racun
    WHERE brRac = p_brRac) >= p_iznos THEN
    UPDATE racun SET stanje = stanje - p_iznos
        WHERE brRac = p_brRac
    SYSTEM "prebroji_i_predaj " || iznos
    COMMIT WORK
ELSE
    ROLLBACK WORK
END IF
```

- bi li pomoglo kad bi se operacija **SYSTEM** premjestila
iza naredbe **COMMIT**?

Model transakcije

Pojmovi i oznake

- transakcije se označavaju oznakama $T_1, T_2, \dots, T_i, T_j, \dots$
- baza podataka se promatra kao statički skup (u smislu nepromjenjivosti kardinalnog broja skupa) imenovanih elemenata ili objekata
- kao element baze podataka može se promatrati dio n-torke (*field*), n-torka (*record*), fizička stranica (*page*, *block*), relacija, baza podataka. Veličina promatranog elementa određena je granulacijom (*granularity*). U većem dijelu razmatranja koje slijedi granulacija je irelevantna
- kao oznake elemenata baze podataka koriste se mala slova x, y, z, itd.
- sve operacije nad elementima baze podataka odnose se na dvije osnovne vrste operacija, operaciju čitanja (*read*) i operaciju izmjene, odnosno pisanja (*write*). Navodi se vrsta operacije i naziv objekta. Vrijednost elementa je irelevantna
- operacije koje transakcija s oznakom T_i obavlja nad elementom x baze podataka označavat će se općenito s $o_i[x]$, a konkretno za operacije čitanja i pisanja, koristit će se oznake $r_i[x]$ i $w_i[x]$
- transakcijske operacije koje obavlja transakcija s oznakom T_i označavaju se s a_i (*abort*) i c_i (*commit*)

Parcijalni poredak

- Parcijalni poredak*
 - Parcijalni poredak (*partial order*) L je uređeni par $(\Sigma, <)$. Skup Σ je domena parcijalnog poretku; relacija $<$ je relacija poretna. Relacija poretna je irefleksivna, asimetrična i tranzitivna relacija na skupu Σ . Ako za elemente $a, b \in \Sigma$ vrijedi $a < b$, odnosno uređeni par $(a, b) \in <$, tada se kaže da a prethodi b , odnosno da b slijedi nakon a . Ako ne vrijedi $a < b$ niti vrijedi $b < a$, tada su a i b neusporedivi.

*formalno ispravnije - *striktni* parcijalni poredak

- Relacija ρ nad skupom P je:
 - irefleksivna ako $(\forall a \in P) \neg (a \rho a)$
 - asimetrična ako $(\forall a, b \in P) (a \rho b \Rightarrow \neg (b \rho a))$
 - tranzitivna ako $(\forall a, b, c \in P) (a \rho b \wedge b \rho c \Rightarrow a \rho c)$

Primjer:

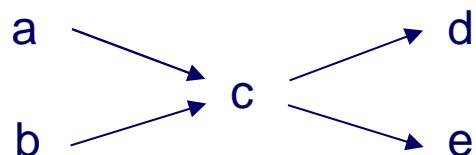
- Domena $\Sigma = \{ a, b, c, d, e \}$
- Relacija poretna $< = \{ (a, c), (a, d), (a, e), (b, c), (b, d), (b, e), (c, d), (c, e) \}$
- Parcijalni poredak $L = (\Sigma, <)$

Parcijalni poredak i usmjereni graf

- Usmjereni graf i usmjereni aciklički graf
 - Usmjereni graf $G = (N, E)$ sastoji se od skupa elemenata N koji se nazivaju čvorovi, te skupa E uređenih parova koji se nazivaju lúkovi. Lük između elemenata a i b označavat će se s (a, b) . Usmjereni aciklički graf je usmjereni graf koji ne sadrži petlje.
 - Parcijalni poredak $L = (\Sigma, <)$ se može prikazati kao usmjereni aciklički graf $G = (N, E)$ pri čemu je $N = \Sigma$, dok je E skup lúkova (a, b) , $a, b \in \Sigma$, takvih da je $a < b$.

Primjer:

- usmjereni aciklički graf za prikaz parcijalnog poretku iz prethodnog primjera:



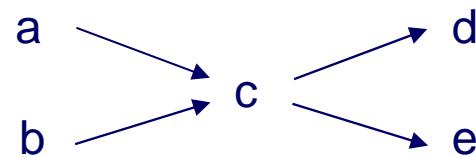
- lúkove implicirane svojstvom tranzitivnosti nije potrebno crtati

Topološki poredak usmjerenog grafa

- Topološki poredak
 - Topološki poredak usmjerenog acikličkog grafa $G = (N, E)$ je niz sastavljen od svih elemenata iz N poredanih tako da se element a nalazi u nizu ispred elementa b samo ako u G ne postoji put od čvora b do čvora a .

Primjer:

- prikaz mogućih topoloških poredaka za usmjereni aciklički graf iz prethodnog primjera:
 - a, b, c, d, e
 - b, a, c, d, e
 - a, b, c, e, d
 - b, a, c, e, d
- jednostavan postupak: iz grafa uzastopno uklanjamo čvorove koji nemaju prethodnike među preostalim čvorovima u grafu. Svaki iz grafa uklonjeni čvor dodajemo na kraj niza koji predstavlja topološki poredak



Model transakcije

- Transakcija se modelira kao parcijalni poredak
- Neka je Σ_i skup operacija transakcije T_i , $<_i$ je relacija koja određuje međusobni poredak operacija iz Σ_i . Transakcija $T_i = (\Sigma_i, <_i)$ pri čemu vrijedi:
 1. $\Sigma_i \subseteq \{ r_i[x], w_i[x] \mid x \text{ je objekt u bazi podataka} \} \cup \{ a_i, c_i \}$
 2. $a_i \in \Sigma_i$ ako i samo ako $c_i \notin \Sigma_i$
 3. ako je operacija $t \in \{ a_i, c_i \}$, tada za sve operacije p , $p \in \Sigma_i \wedge p \neq t$ vrijedi $p <_i t$
 4. za svake dvije operacije $r_i[x], w_i[x] \in \Sigma_i$, vrijedi ili $r_i[x] <_i w_i[x]$ ili $w_i[x] <_i r_i[x]$

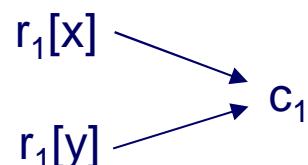
Objašnjenje:

- 1: definira koje su operacije transakcije dopuštene
- 2: u transakciji se nalazi jedna i samo jedna od operacija *abort* ili *commit*
- 3: operacija *abort* ili *commit* mora biti posljednja operacija svake transakcije
- 4: ako transakcija obavlja operaciju čitanja i operaciju pisanja nad istim elementom x , redoslijed obavljanja tih operacija utječe na rezultat operacije čitanja. Svojstvom se garantira da je za operaciju pisanja elementa x određen poredak u odnosu na operaciju čitanja istog elementa x

Parcijalni poredak, graf transakcije

Primjer:

- $T_1 = (\Sigma_1, <_1)$
 - $\Sigma_1 = \{ r_1[x], r_1[y], c_1 \}$
 - $<_1 = \{ (r_1[x], c_1), (r_1[y], c_1) \}$
- transakcija T_1 , odnosno parcijalni poredak $(\Sigma_1, <_1)$ također se može prikazati sljedećim usmjerenim acikličkim grafom:

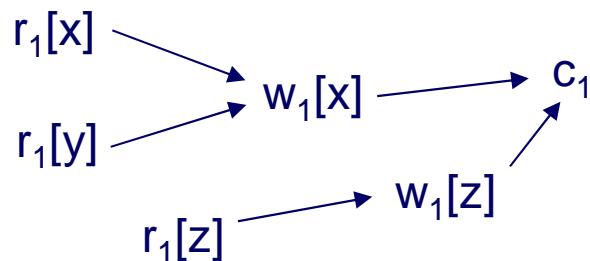


- operacije $r_1[x]$ i $r_1[y]$ su neusporedivi elementi transakcije - nije utvrđen njihov međusobni poredak
- operacija c_1 se mora dogoditi nakon $r_1[x]$ i $r_1[y]$

Parcijalni poredak, graf transakcije

Primjer:

- $T_1 = (\Sigma_1, <_1)$
 - $\Sigma_1 = \{ r_1[x], r_1[y], w_1[x], r_1[z], w_1[z], c_1 \}$
 - $<_1 = \{ (r_1[x], w_1[x]), (r_1[y], w_1[x]), (r_1[z], w_1[z]), (r_1[x], c_1), (r_1[y], c_1), (w_1[x], c_1), (r_1[z], c_1), (w_1[z], c_1) \}$
- transakcija T_1 , odnosno parcijalni poredak $(\Sigma_1, <_1)$ također se može prikazati sljedećim usmjerenim acikličkim grafom:



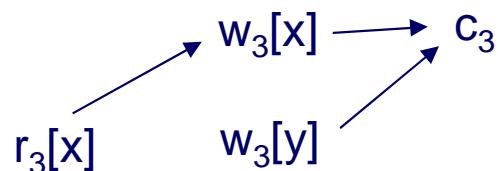
- $w_1[x]$ se mora dogoditi nakon $r_1[x]$ i $r_1[y]$
- nije određeno kojim redoslijedom se moraju obaviti npr. $r_1[x]$ i $r_1[y]$.
- iako u relaciji poretku postoji par $(r_1[x], c_1)$, u grafu taj lük nije potrebno crtati jer iz $r_1[x] < w_1[x]$ i $w_1[x] < c_1$, temeljem svojstva tranzitivnosti proizlazi $r_1[x] < c_1$.

Parcijalni poredak, graf transakcije

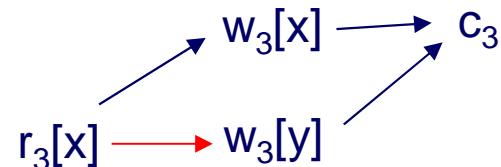
Dodatna razmatranja:

- osim lúkova koje graf **mora** sadržavati prema definiciji modela transakcije (sve operacije čitanja i pisanja nad istim elementom moraju biti poredane, sve operacije čitanja i pisanja prethode operacijama *commit/abort*), u graf se ucrtavaju i lúkovi koji proizlaze iz semantike transakcije

Primjer:



vrijednost koju zapisuje operacija $w_3[y]$ ne ovisi o rezultatu operacije $r_3[x]$

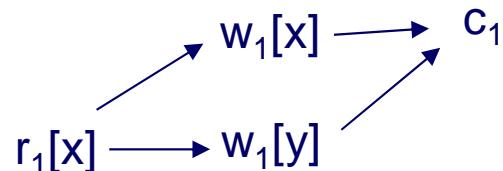


ako transakcija novu vrijednost za y određuje uz pomoć vrijednosti elementa x , dodaje se lúk između operacija $r_3[x]$ i $w_3[y]$

Topološki poredak

Primjer:

- Transakcija T_1 je prikazana sljedećim usmjerenim acikličkim grafom



- Topološkim poretkom čvorova grafa može se dobiti prikaz transakcije u obliku niza operacija
 - $r_1[x]$, $w_1[y]$, $w_1[x]$, c_1
ili
 - $r_1[x]$, $w_1[x]$, $w_1[y]$, c_1

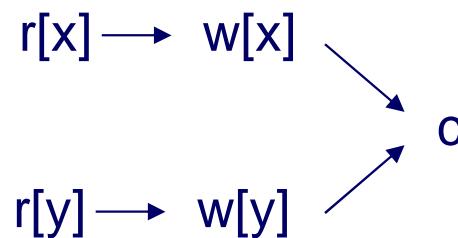
Graf transakcije

Primjer: zadani iznos sredstava prebaciti s prvog zadanog na drugi zadani račun
(stanje prvog računa umanjiti, a drugog računa uvećati).

```
x   y
SELECT stanje INTO p FROM racun WHERE brRac = zadBrRac1;
SELECT stanje INTO q FROM racun WHERE brRac = zadBrRac2;
LET r = p - zadIznos;
LET s = q + zadIznos;
UPDATE racun SET stanje = r WHERE brRac = zadBrRac1;
UPDATE racun SET stanje = s WHERE brRac = zadBrRac2;
COMMIT;
```

```
start;
read(x, p);
read(y, q);
-- izracunaj r, s
write(x, r);
write(y, s);
commit;
```

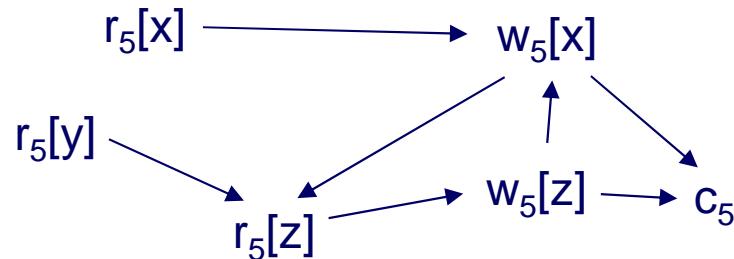
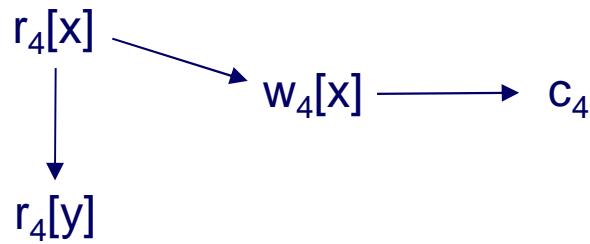
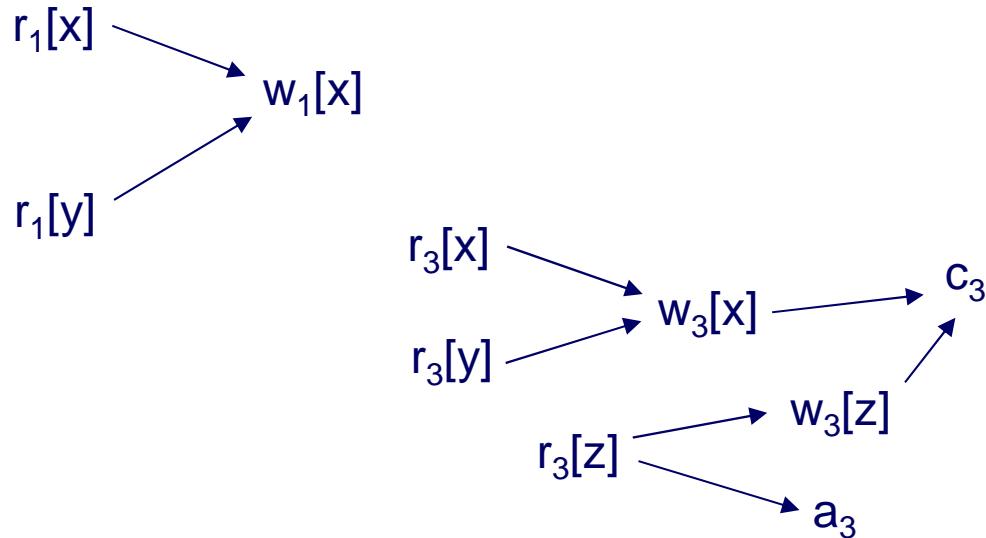
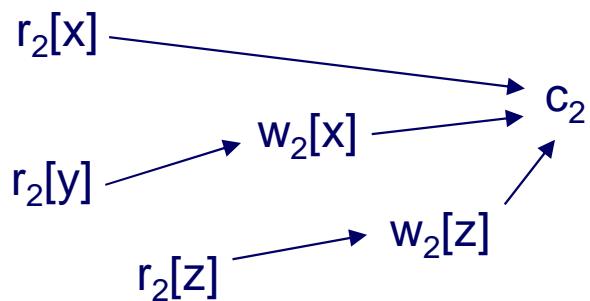
Model transakcije



Graf transakcije

Primjer:

- sljedeći grafovi nisu ispravni grafovi transakcija



- za vježbu: utvrditi pogrešku na svakom od grafova

Subtransakcije

Ugnježđivanje transakcija nije dopušteno:

- pokušaj parcijalnog poništavanja transakcije na pogrešan način

```
BEGIN WORK;
SQL_1;
SQL_2;
BEGIN WORK;
SQL_3;
SQL_4;
IF (uvjet_1) THEN
    COMMIT WORK;
ELSE
    ROLLBACK WORK;
END IF;
SQL_5;
COMMIT WORK;
```

- ispravan način za poništavanje samo određenih dijelova transakcije: *savepoints*

Savepoints

- ROLLBACK poništava sve promjene u bazi podataka koje su transakcijom obavljene nakon početka transakcije
- u transakciji T je moguće označiti jednu ili više točki (*savepoints*)
- unutar transakcije T je moguće poništiti promjene koje su obavljene nakon svake od označenih točaka (*savepoint*)
- SQL naredbe
 - **SAVEPOINT savepoint_id;**
 - **ROLLBACK TO SAVEPOINT savepoint_id;**

Savepoints

```
BEGIN WORK;
SQL_1;
SAVEPOINT t_1;
SQL_2;
SAVEPOINT t_2;
SQL_3;
IF (uvjet_1) THEN
    ROLLBACK TO SAVEPOINT t_2;
    SQL_4;
END IF;
SQL_5;
IF (uvjet_2) THEN
    ROLLBACK TO SAVEPOINT t_1;
    SQL_6;
END IF;
SQL_7;
IF (uvjet_3) THEN
    COMMIT WORK;
ELSE
    ROLLBACK WORK;
END IF
```

- obavlja se naredba SQL_1
- označava se savepoint t_1
- obavlja se SQL_2
- označava se savepoint t_2
- obavlja se SQL_3
- ako vrijedi uvjet_1 poništavaju se promjene koje je obavila SQL_3 i obavlja se SQL_4
- obavlja se SQL_5
- ako vrijedi uvjet_2 poništavaju se promjene koje su obavile sve SQL naredbe obavljene nakon t_1 i obavlja se SQL_6
- obavlja se SQL_7
- ako vrijedi uvjet_3 potvrđuju se sve obavljene promjene (koje u međuvremenu nisu poništene)
- ako ne vrijedi uvjet_3, poništavaju se sve promjene (koje u međuvremenu već nisu poništene)

Primjer transakcije (IBM IDS)

korisnik *horvat* obavlja:

```
CREATE DATABASE testTran
    IN dbspace1 WITH LOG;
CREATE TABLE racun (
    sifRacun      INTEGER
, stanje        DECIMAL(9,2)
, dopustMinus  DECIMAL(9,2)
, PRIMARY KEY (sifRacun)
    CONSTRAINT pkRacun
, CHECK (stanje + dopustMinus >= 0)
    CONSTRAINT chkRacunStanje
);

```

```
CREATE TABLE promet (
    sifPromet     SERIAL
, sifRacun      INTEGER
, iznos         DECIMAL(9,2)
, PRIMARY KEY (sifPromet)
    CONSTRAINT pkPromet
, FOREIGN KEY (sifRacun)
    REFERENCES racun (sifRacun)
    CONSTRAINT fkPrometRacun
);

```

- narušavanje integritetskih ograničenja izazvat će pogrešku (SQL error, ISAM error, Error data):

(-268, -100, horvat.pkracun)

(-530, 0, horvat.chkracunstanje)

(-268, -100, horvat.pkpromet)

(-691, -111, horvat.fkprometracun)

Primjer transakcije (IBM IDS) - nastavak

- napisati pohranjenu proceduru **unosPromet**
- ulazni argumenti: šifra računa i iznos. Unijeti n-torku u relaciju promet (serijski broj, zadanu šifru računa i zadani iznos). Promijeniti stanje zadanog računa tako da postojećem stanju pribroji zadani iznos). Ove dvije operacije se moraju obaviti u okviru transakcije
- ako stanje na računu padne ispod dopuštenog (ograničenje chkRacunStanje) tada u pozivajući program treba dojaviti pogrešku -746, 0, 'Nedopušten minus'
- u slučaju bilo koje druge pogreške, u pozivajući program dojaviti originalnu pogrešku
- procedura ne vraća ništa: smatra se da je uspješno obavljena ako ne vrati pogrešku
- procedura samostalno upravlja granicama transakcije, tj. započinje i potvrđuje/poništava transakciju: započeta transakcija po završetku procedure mora biti u stanju završena (ili potvrđena ili poništена)

Primjer transakcije (IBM IDS) - nastavak

```
CREATE PROCEDURE unosPromet(pSifRacun LIKE promet.sifRacun
                           , pIznos LIKE promet.iZNOS)
  DEFINE sqle, isame INTEGER;
  DEFINE errdata CHAR(80);
  ON EXCEPTION SET sqle, isame, errdata
    ROLLBACK WORK;
    IF sqle = -530 AND errdata LIKE '%chkracunstanje%' THEN
      RAISE EXCEPTION -746, 0, 'Nedopušten minus';
    ELSE
      RAISE EXCEPTION sqle, isame, errdata;
    END IF
  END EXCEPTION;
  BEGIN WORK;
  INSERT INTO promet VALUES (0, pSifRacun, pIznos);
  UPDATE racun SET stanje = stanje + pIznos
    WHERE sifRacun = pSifRacun;
  COMMIT WORK;
END PROCEDURE;
```

Primjer transakcije (IBM IDS) - nastavak

- primjer izvršavanja iz interaktivnog alata (npr. Server Studio)

racun	sifRacun	stanje	dopustMinus
	1	100.00	200.00
	2	500.00	500.00

promet	sifPromet	sifRacun	iznos

```
EXECUTE PROCEDURE unosPromet(2, -1000.00);
```



racun	sifRacun	stanje	dopustMinus
	1	100.00	200.00
	2	-500.00	500.00

promet	sifPromet	sifRacun	iznos
	1	2	-1000.00

```
EXECUTE PROCEDURE unosPromet(1, -500.00);
```



-746, 0, Nedopushten minus

```
EXECUTE PROCEDURE unosPromet(3, 500.00);
```



-691, -111, horvat.fkprometraracun

Primjer transakcije (IBM IDS) - nastavak

- poziv iz java klijentske aplikacije

```
...
Statement stmt = conn.createStatement();
conn.setAutoCommit(true); // granicama transakcije ne upravlja klijent
try {
    stmt.executeUpdate("EXECUTE PROCEDURE unosPromet(x, y)");
} catch (SQLException exc) {
    System.out.println(exception.getErrorCode() + " " + exception.getMessage());
}
```

- ako se java program izvrši nakon što se x, y zamijene vrijednostima 1, -500.00



-746 Nedopušten minus

- ako se java program izvrši nakon što se x, y zamijene vrijednostima 3, 500.00



-691 horvat.fkprometracun

Primjer transakcije (IBM IDS) - nastavak

- transakcija realizirana pomoću java + JDBC API

```
void unosPromet(Connection connection
                 , Integer sifRacun
                 , BigDecimal iznos) throws SQLException {
    Statement stmt = connection.createStatement();

    connection.setAutoCommit(false); // granicama transakcije upravlja klijent
    try {
        stmt.executeUpdate("INSERT INTO promet VALUES (0, " + sifRacun + ", " + iznos + ")");
        stmt.executeUpdate("UPDATE racun SET stanje = stanje + " + iznos +
                           " WHERE sifRacun = " + sifRacun);
    }
    catch (SQLException exception) {
        connection.rollback();
        if (exception.getErrorCode() == -530 &&
            exception.getMessage().indexOf("chkracunstanje") != -1) {
            throw new SQLException("Nedopušten minus", exception.getSQLState(), -746);
        }
        else {
            throw exception;
        }
    }
    connection.commit();
}
```

Primjer transakcije (IBM IDS) - nastavak

- primjeri poziva

```
// podrazumijeva se da je objektom connection uspostavljena sjednica
```

```
...
try {
    unosPromet(connection, sifRacun, iznos);
}
catch (SQLException exception) {
    System.out.println(exception.getErrorCode() + " " + exception.getMessage());
}
```

- ako su vrijednosti za sifRacun i iznos: 1, -500.00



-746 Nedopušten minus

- ako su vrijednosti za sifRacun i iznos: 3, 500.00



-691 horvat.fkprometracun

Primjer transakcije (IBM IDS) - nastavak

- postupci u prethodno prikazanim primjerima mogli bi se dodatno razmotriti. Trebalo bi uzeti u obzir da:
 1. naredba za *commit* može izazvati pogrešku
 - tipičan primjer: ako se koristi CONSTRAINTS DEFERRED
 2. naredba za *rollback* može izazvati pogrešku
 - malo vjerojatan problem: što bi se u takvom slučaju uopće moglo učiniti?
 3. SUBP u nekim slučajevima može prekinuti i poništiti transakciju tijekom izvršavanja neke SQL naredbe koja je dio transakcije
 - tipičan primjer: ako SQL naredba (npr. UPDATE) izazove potpuni zastoj, SUBP smjesta prekida i poništava cijelu transakciju
- za vježbu: analizirati što bi se dogodilo u prethodno prikazanim primjerima kada bi se dogodile pogreške opisane pod 1, 2, 3.

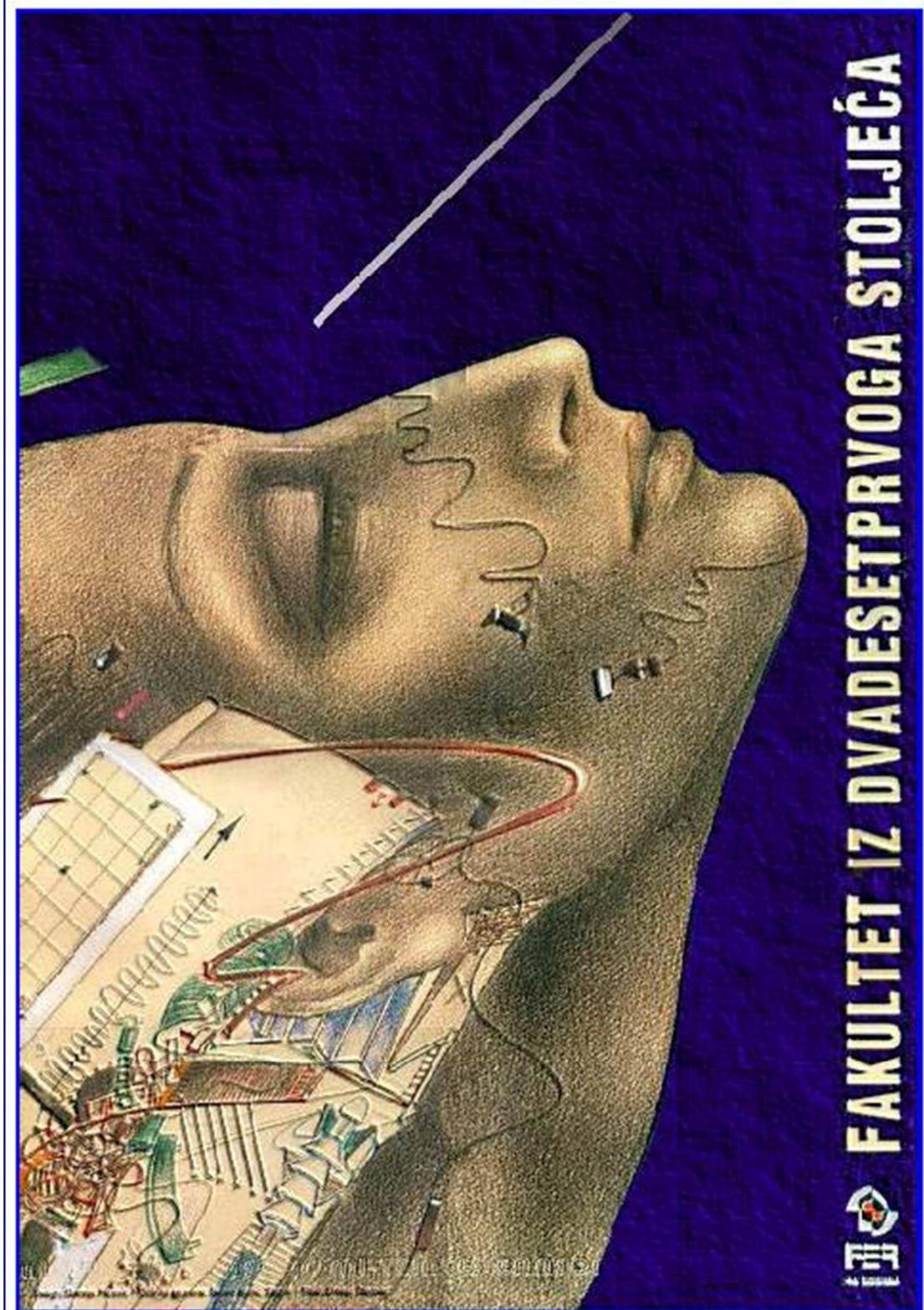
Literatura:

- P. Bernstein, V. Hadzilacos, N. Goodman. Concurrency control and recovery in database systems. Addison-Wesley Publishing Company. Reading, Massachusetts. 1987.
- H. Garcia-Molina, J. Ullman, J. Widom. Database system implementation. Prentice-Hall International, Inc. Upper Saddle River, New Jersey. 2000.
- R. Elmasri, S. Navathe. Fundamentals of database systems, 3rd ed. Addison-Wesley Publishing Company. Reading, Massachusetts. 2000.
- A. Silberschatz, H. Korth, S. Sudarshan. Database system concepts, 4th ed. McGraw-Hill. 2002.

Sustavi baza podataka

Predavanja

**6. Sustav za obnovu
(1. dio)**
travanj 2014.



Pogreške u SUBP-u

- različiti uzroci pogrešaka:
 - pogreške računalne opreme (softverske, hardverske)
 - prekid napajanja
 - požar, potres, eksplozija, sabotaža
 - itd.
- svaki od tih događaja može dovesti do gubitka informacije
- takve događaje moglo bi se pokušati spriječiti korištenjem "besprijekornih" komponenata i "besprijekornim" dizajnom sustava → previsoka cijena

Rješenje:

- korištenje sustava za obnovu (*recovery system*)

Zadaci sustava za obnovu

- u slučaju pogreške dovesti SUBP u stanje ekvivalentno zadnjem konzistentnom stanju sustava prije pogreške
- pri tome:
- ostvariti čim veću sposobnost brzog oporavka (*resilience*) SUBP-a
 - sustav treba biti u stanju oporaviti se nakon različitih vrsta pogrešaka
- ostvariti čim veću raspoloživost (*availability*) SUBP-a minimiziranjem vremena potrebnog za oporavak sustava nakon pogreške
 - omjer vremena tijekom kojeg je sustav sposoban obavljati predviđenu funkciju i ukupnog promatranog vremena. Ovisi o pouzdanosti (*reliability*) i vremenu potrebnom za oporavak

$$Av = MTBF / (MTBF + MTTR)$$

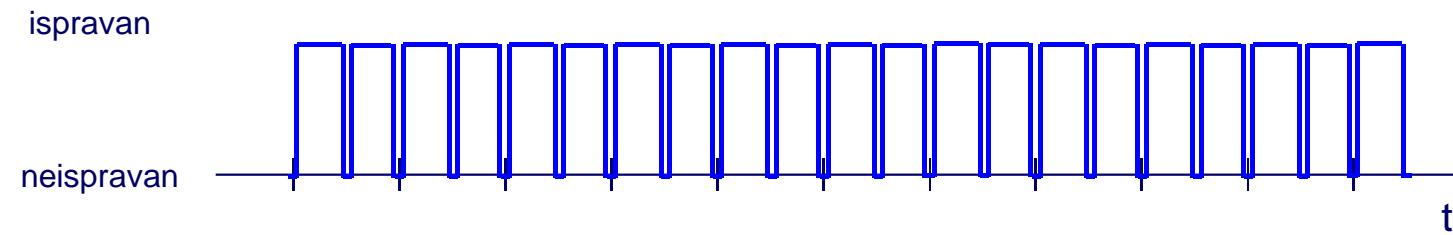
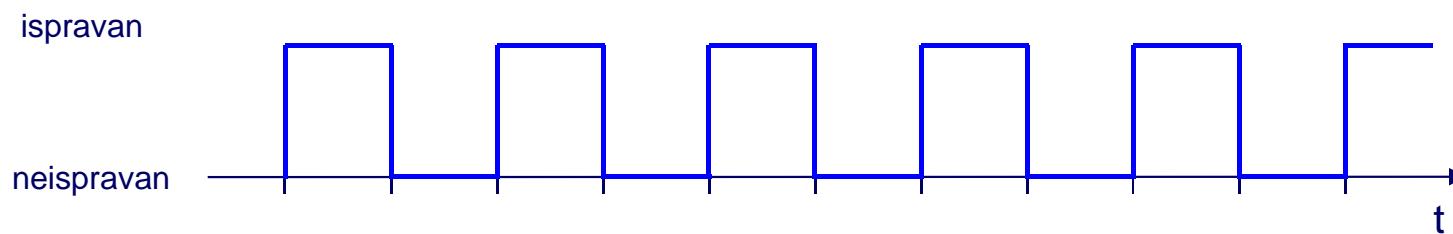
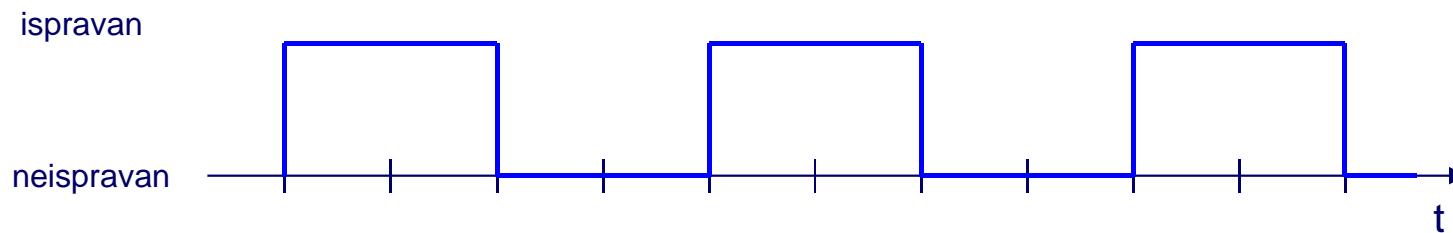
Mean Time Between Failures
Mean Time To Repair

sustav u kvaru	raspoloživost
1 sat/dan	95.8 %
1 sat/mjesec	99.86 %
1 sat/godinu	99.9886 %

Pouzdanost (*reliability*): svojstvo sustava koje se odnosi na vjerojatnost da će sustav funkcionirati ispravno (tj. bez pogreške koja se može zamijetiti izvan granica sustava) tijekom određenog vremenskog intervala. Nije zadaća sustava za obnovu.

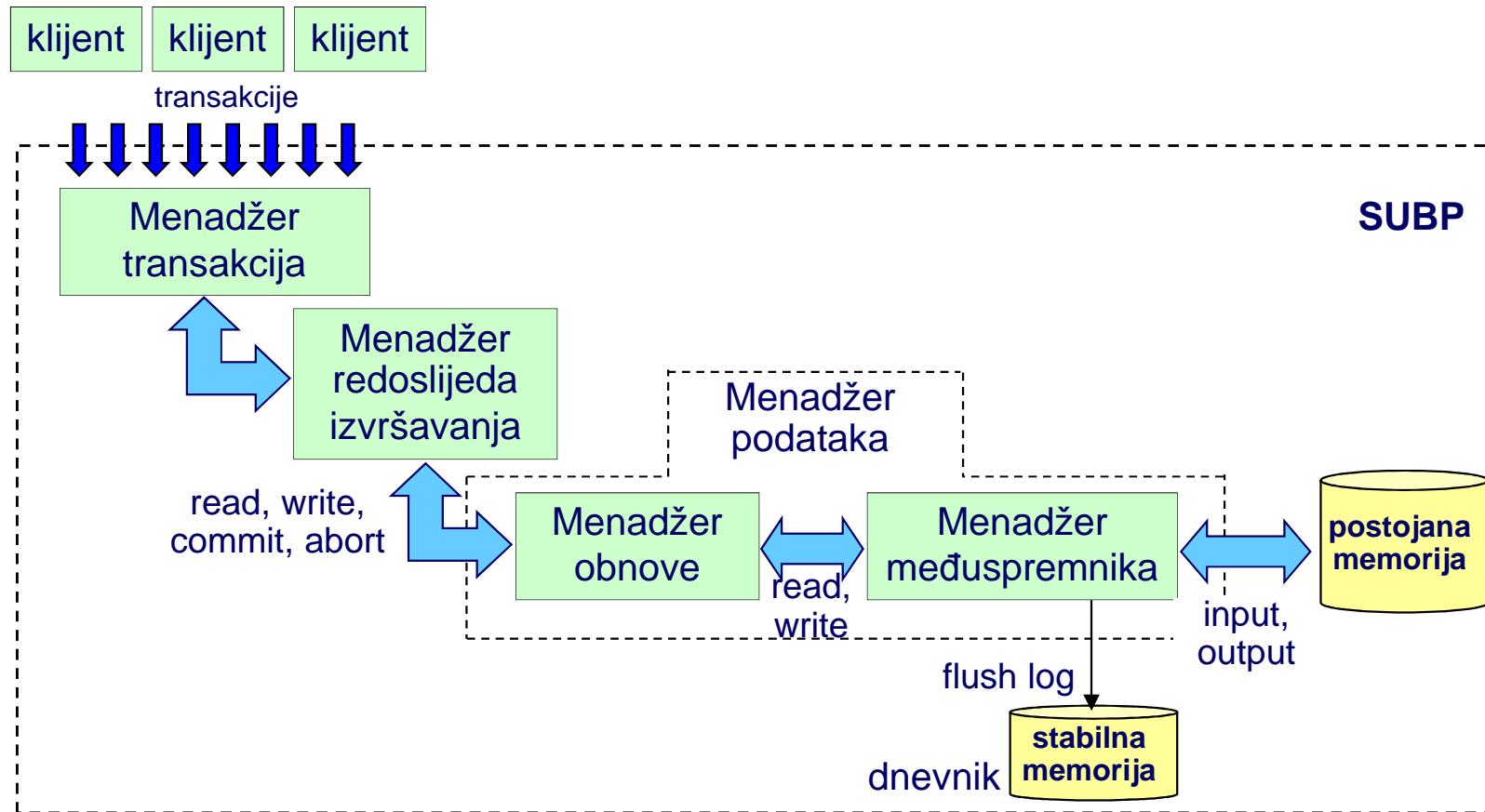
Raspoloživost, pouzdanost

- usporediti raspoloživost, sposobnost brzog oporavka i pouzdanost sljedećih sustava



Model SUBP-a

- apstraktni model sustava za upravljanje bazama podataka [Bernstein]



- na slici su prikazani samo oni dijelovi sustava za upravljanje bazama podataka koji upravljaju obnovom i istodobnim pristupom (npr. nedostaje *query optimizer*)

Model SUBP-a

- Menadžer transakcija (*transaction manager*) upravlja početkom i završetkom transakcije, zaprimljene operacije transakcija upućuje prema menadžeru redoslijeda izvršavanja (*scheduler*). U distribuiranim sustavima ima dodatnu zadaću usmjeravanja operacija transakcija prema drugim SUBP
- Menadžer redoslijeda izvršavanja (*scheduler*) upravlja redoslijedom obavljanja operacija transakcija: obavljanjem, odbijanjem ili odgađanjem obavljanja pojedinih operacija. Upravlja istodobnim (ili "paralelnim") pristupom
- Menadžer obnove (*recovery manager*) osigurava pouzdano potvrđivanje ili poništavanje transakcija. Podsistav je odgovoran za to da baza podataka sadrži sve efekte potvrđenih transakcija i niti jedan efekt poništenih transakcija. Između ostalog, mora biti neosjetljiv na gubitak sadržaja nepostojane (*volatile*) memorije ili na gubitak sadržaja ili kvar postojane (*non-volatile*) memorije.
- Menadžer međuspremnika (*buffer manager*) je podsustav zadužen za prebacivanje podataka između međuspremnika i postojane memorije.
- Menadžer obnove i menadžer međuspremnika čine veću cjelinu koja se uobičajeno naziva menadžer podataka (*data manager*)

Uspostavljanje sustava za obnovu

- identificirati pogreške koje se u pojedinim komponentama sustava mogu dogoditi, te utjecaj pojedine vrste pogreške na sposobnost oporavka i raspoloživost sustava
- definirati postupke obnove (*recovery algorithms*) koji će osigurati otpornost sustava na pogreške, odnosno omogućiti očuvanje konzistentnosti baze podataka bez obzira na pogreške

Postupci obnove

1. postupci koji se provode tijekom redovitog funkcioniranja sustava i osiguravaju prikupljanje dovoljno podataka za obnovu baze podataka nakon pogreške
2. postupci pomoću kojih se u slučaju pogreške sustav na pouzdan način vraća u konzistentno stanje

Temeljna ideja

Korištenje redundancije - svaki podatak koji je izgubljen zbog pogreške u jednom dijelu sustava, mora se moći rekonstruirati iz podataka redundantno pohranjenih u nekom drugom dijelu sustava

Vrste pogrešaka

- pogreške koje utječu na funkcioniranje SUBP-a klasificiraju se na sljedeći način:
 - pogreške transakcija (*transaction failures*)
 - pogreške sustava (*system failures, crashes*)
 - pogreške medija (*media failures*)

Pogreške transakcije

- pogreška koja je uzrokovana ulaskom transakcije u stanje koje onemogućava daljnje ispravno obavljanje operacija transakcije. SUBP **implicitno** prekida i poništava dotičnu transakciju
 - neočekivana aritmetička pogreška (npr. dijeljenje s nulom, preliv) koja nastaje kao posljedica pogrešne ulazne vrijednosti transakcije ili logičke pogreške programa
 - pokušaj obavljanja operacije koja narušava pravilo integriteta ili pravila sigurnosti
 - pogreške koje nastaju zbog interakcije transakcije s drugim transakcijama
 - detekcija stvarnog ili potencijalnog potpunog zastoja
 - narušavanje serijalizabilnog izvršavanja
 - zahtjev za prekidom transakcije od strane korisnika
- u svim navedenim slučajevima SUBP mora poništiti transakciju
 - poništavanje transakcije obavlja sustav za obnovu

Nepredviđene pogreške kojima transakcije ne rukuje eksplicitno: ako je npr. transakcija opisana u jeziku SPL, tada su to one pogreške koje se ne presreću ili se ne obrađuju u ON EXCEPTION blokovima

Pogreške transakcije

- važne karakteristike pogrešaka transakcija:
 - ne dolazi do gubitka sadržaja postojane niti nepostojane memorije
 - ne dolazi do zastoja u funkcioniranju SUBP-a (transakcije koje nisu sudjelovale u pogrešci nastavljaju se izvršavati bez zastoja)

Pogreške transakcije

Primjer:

```
begin work
    read(x, p)
    read(y, q)
    p ← p / q
    write(x, p)
commit work
```

- Ako je vrijednost elementa y jednaka 0, dogodit će se **pogreška transakcije** uzrokovana dijeljenjem s nulom. SUBP će prekinuti i poništiti takvu transakciju.

Eksplisitno rukovanje pogreškama

- pogrešku koja se dogodi pri obavljanju neke operacije transakcije SUBP signalizira transakciji. Time se transakciji omogućava **eksplisitno rukovanje pogreškom**, npr. tako da pod određenim programskim uvjetima eksplisitno prekine transakciju ili obavi niz kompenzacijskih operacija i nastavi s izvršavanjem transakcije
- ako transakcija eksplisitno rukuje pogreškom, tada se pogreška **ne smatra pogreškom transakcije**, bez obzira obavi li se pri tome eksplisitno prekidanje transakcije ili se transakcija nastavlja nakon obavljanja niza kompenzacijskih operacija
 - iako se ne radi o pogrešci transakcije, za poništavanje transakcije i ovdje se koriste mehanizmi sustava za obnovu

Eksplicitno rukovanje pogreškama

Primjer:

```
begin work
    read(x, p)
    read(y, q)
    p ← p / q
    write(x, p)
commit work
on exception (division by
    zero)
    p ← p / 2
end exception with continue
```

Ako je vrijednost elementa y jednaka 0, transakcija će obaviti kompenzaciju akciju. **Ne smatra se pogreškom transakcije**

- iako se ne radi o pogrešci transakcije, za poništavanje transakcije i ovdje se (u primjeru na desnoj strani) koriste mehanizmi sustava za obnovu

```
begin work
    read(x, p)
    read(y, q)
    p ← p / q
    write(x, p)
commit work
on exception (division by
    zero)
    rollback work
end exception
```

Ako je vrijednost elementa y jednaka 0, transakcija će se eksplicitno prekinuti te poništiti. **Ne smatra se pogreškom transakcije**

Pogreške sustava

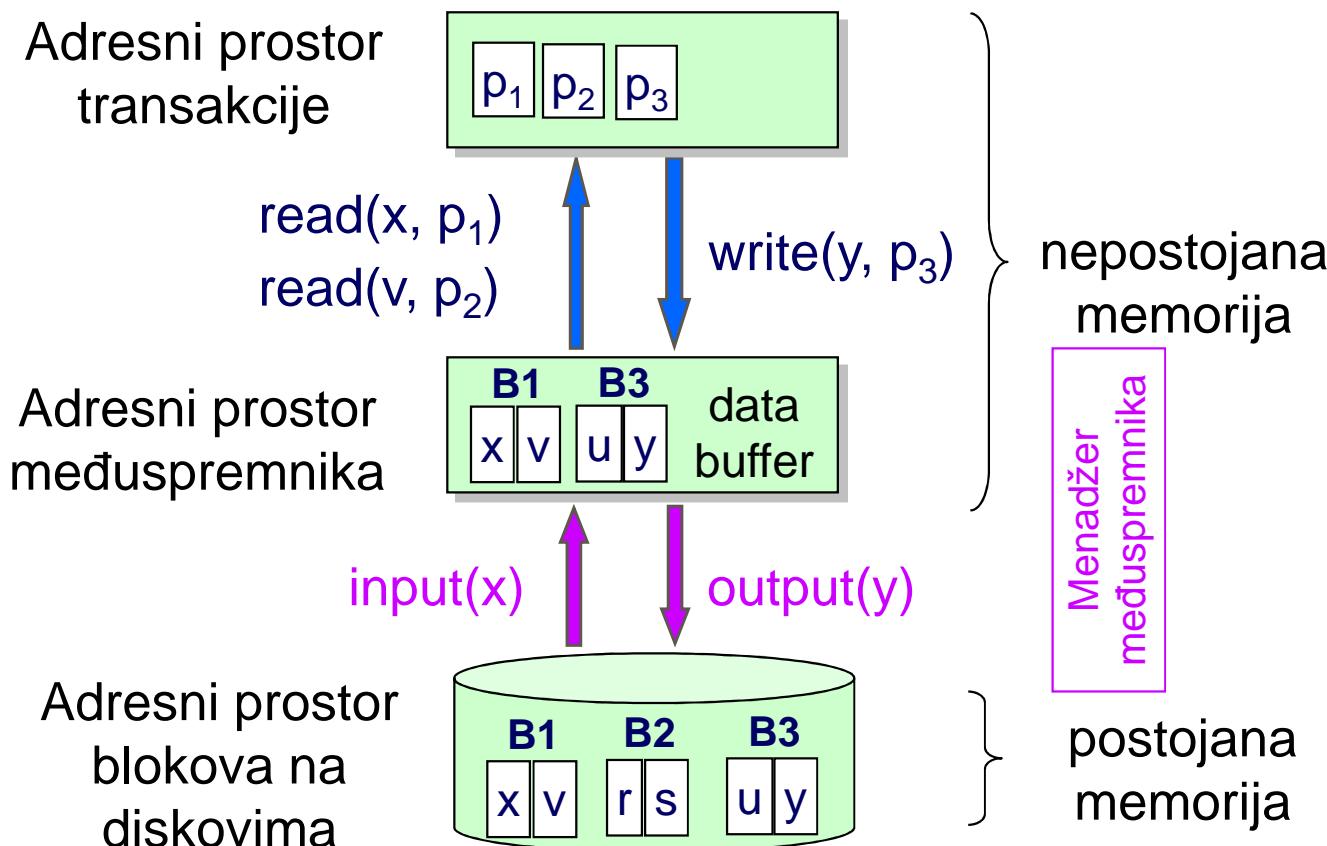
- SUBP dio podataka privremeno pohranjuje u nepostojanoj memoriji, u međuspremnicima baze podataka
- pogreška sklopolja, pogreška programske podrške (operacijskog sustava ili programske podrške SUBP-a), nestanak napajanja i slično, uzrokuje
 - trenutačni prekid rada sustava (SUBP procesa ili OS)
 - gubitak sadržaja međuspremnika
- pretpostavka o prekidu rada sustava u slučaju pogreške (*fail-stop* ili *fail-fast assumption*) je važan mehanizam kojim se osigurava da hardverske ili softverske pogreške izazivaju trenutačno zaustavljanje sustava (*system halt*), s ciljem sprječavanja korupcije postojane memorije
 - djelovanje sukladno ovoj pretpostavci omogućeno je mehanizmima samoprovjere na hardverskoj i softverskoj razini koje posjeduju današnji sustavi (*CPU checks, memory parity, RAID*)
- važne karakteristike pogrešaka sustava:
 - gubitak sadržaja nepostojane memorije
 - prekid rada sustava
 - postojana memorija je očuvana

Pogreške medija

- Pogreške medija odnose se na djelomični ili potpuni gubitak podataka u postojanoj memoriji
 - također mogu biti posljedica neispravnosti sklopolja, programske potpore, operacijskog sustava, SUBP-a
 - požar, potres, vandalizam, teroristički napad, posljedice ratnog djelovanja
 - sabotaža, namjerno uništavanje podataka
 - pogreške tijekom prijenosa podataka između nepostojane i postojane memorije
 - *disk crash, head crash*
- Pogreška medija u loše podešenom sustavu za obnovu u pravilu izaziva *katastrofu*
 - katastrofa (*disaster*) je tehnički pojам kojim se opisuje pogreška nakon koje nije moguća obnova

Temeljne operacije (*primitive operations*) transakcije

- SUBP koristi tri adresna prostora
 - lokalni adresni prostor transakcija
 - adresni prostor međuspremnika baze podataka
 - adresni prostor blokova na diskovima



Temeljne operacije (*primitive operations*) transakcije

Temeljne operacije

- $\text{input}(x)$ - iz adresnog prostora diska kopiraj blok koji sadrži element x u međuspremnik
- $\text{read}(x, p)$ - ako blok u kojem se nalazi x nije u međuspremniku, obavi $\text{input}(x)$. Kopiraj sadržaj x iz međuspremnika u lokalnu varijablu transakcije p
- $\text{write}(x, p)$ - ako blok koji sadrži x nije u međuspremniku, obavi $\text{input}(x)$. Kopiraj sadržaj lokalne varijable transakcije p u međuspremnik
- $\text{output}(x)$ - kopiraj blok međuspremnika u kojem se nalazi x u odgovarajući blok na disku

Temeljne operacije (*primitive operations*) transakcije

- u cilju smanjenja broja U/I operacija, menadžer međuspremnika dopušta da se rezultat obavljene izmjene neko vrijeme (i uz zadovoljene određene uvjete) zadrži u nepostojanoj memoriji

- Primjer:
- dva **objekta** u bazi podataka ($x = 3$, $y = 5$)
 - integritetsko ograničenje: $y = x + 2$
 - p - lokalna varijabla transakcije T
 - x i y se ne nalaze u istom bloku na disku

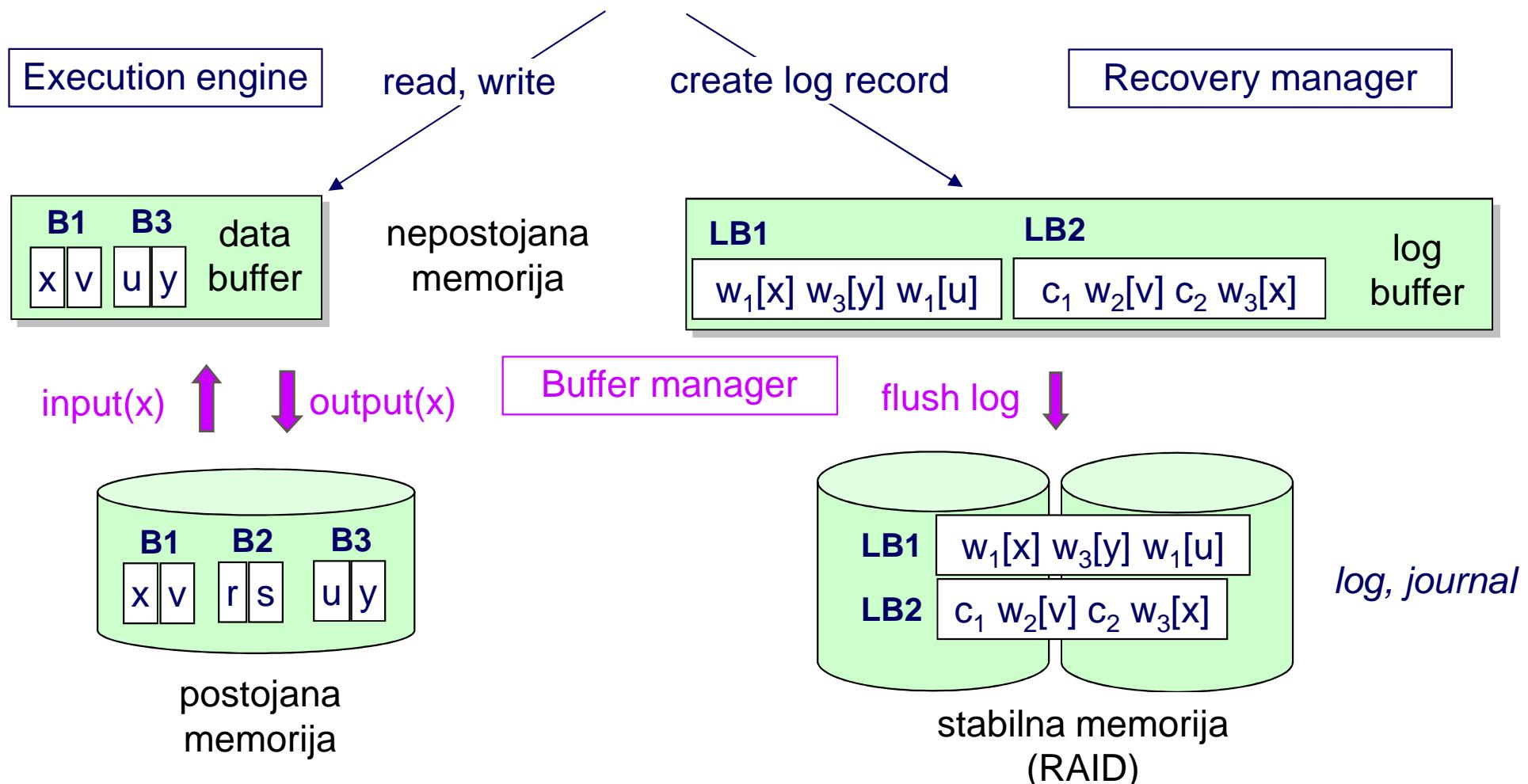
T	Buffer manager	p	buff x	buff y	disk x	disk y
begin						
read(x, p)						
$p \leftarrow p * 2$	input(x)	3	3		3	5
write(x, p)					3	5
$p \leftarrow p + 2$		6	3		3	5
					3	5
					3	5
write(y, p)	output(x)	8	6		3	5
commit					6	5
					6	5
					6	5
	input(y)	8	6	8	6	5
	output(y)	8	6	8	6	8

- ako se pogreška sustava dogodi prije obavljanja $\text{output}(x)$ ili nakon obavljanja $\text{output}(y)$, konzistentnost baze podataka nije upitna
- ako se pogreška sustava dogodi nakon obavljanja $\text{output}(x)$, a prije obavljanja $\text{output}(y)$, baza podataka ostaje u nekonzistentnom stanju

Dnevnik, zapis dnevnika

- aktivnosti transakcija se bilježe u **dnevniku** (*log, journal*)

$w_1[x], r_2[v], w_3[y], r_3[u], w_1[u], c_1, r_3[x], w_2[v], c_2, r_3[v], w_3[x], a_3, r_4[v], w_4[x] \dots$



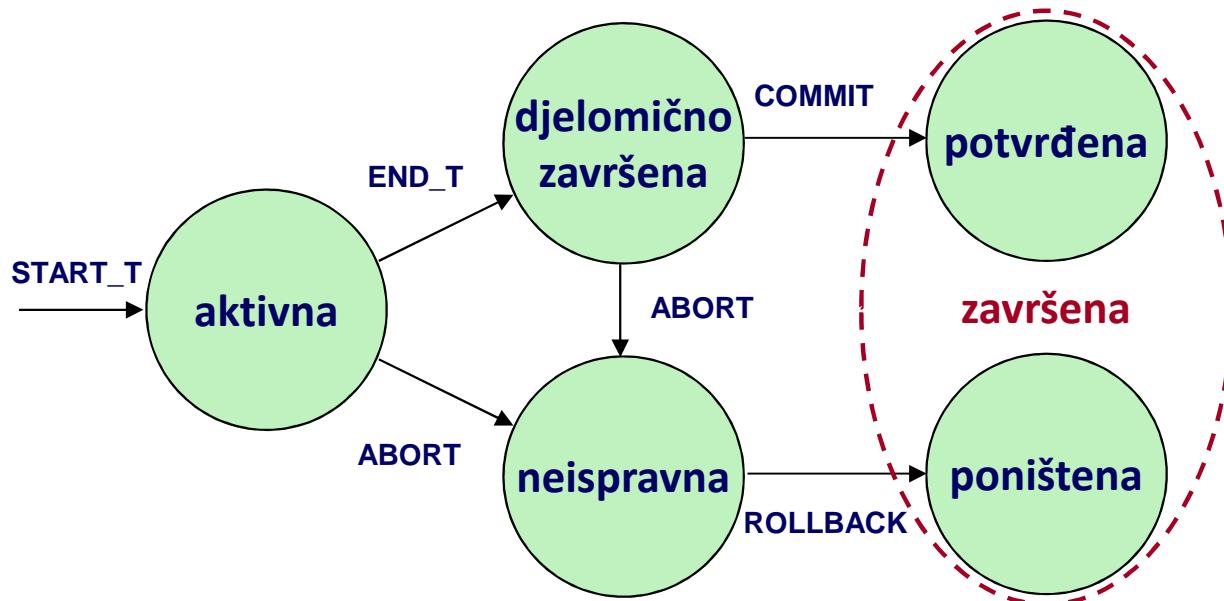
Dnevnik, zapis dnevnika

- aktivnosti transakcija se bilježe u **dnevniku** (*log, journal*)
 - dnevnik obuhvaća niz **zapisa dnevnika** (*log records*) u obliku datoteke u koju se zapisi dodaju isključivo na začelje (*append-only file*)
 - u isti dnevnik se bilježe aktivnosti svih transakcija koje su u tijeku
 - u dnevniku se zapisi različitih transakcija međusobno isprepliću (*interleaves*)
- zapisi dnevnika se stvaraju (*create log record*) u nepostojanoj memoriji (*log buffers*)
- menadžer međuspremnika upravlja zapisivanjem zapisa dnevnika u **stabilnu** memoriju: u propisanim trenucima obavlja operaciju *flush log*.
- kada se točno obavljaju *output* operacije, a kada *flush log* operacije, ovisi o primjenjenoj tehnici obnove

Dnevnik izmjena, zapis dnevnika izmjena

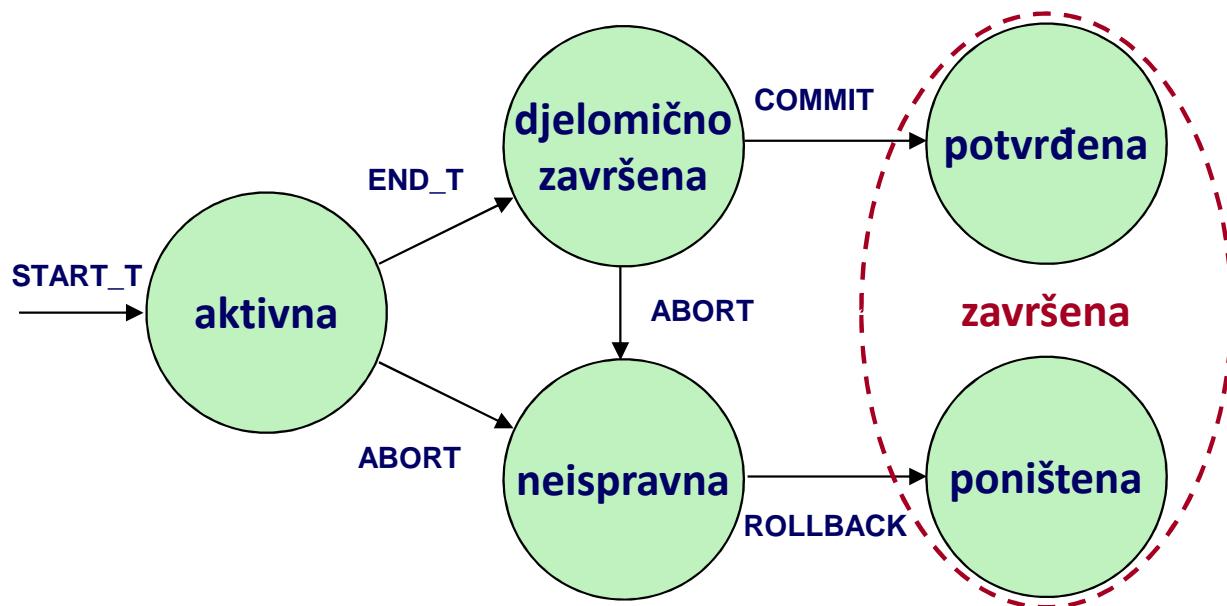
- **dnevnik izmjena** (*update log*) sadrži niz **zapisa dnevnika izmjena** (*update log record*)
 - u dnevniku izmjena se bilježi nekoliko vrsta zapisa dnevnika izmjena
 - zapis koji opisuje početak transakcije: $\langle \text{start } T_i \rangle$
 - započela je transakcija s identifikatorom T_i
 - zapis koji opisuje operaciju pisanja: $\langle T_i, x_j, V_{\text{old}}, V_{\text{new}} \rangle$
 - transakcija s identifikatorom T_i promijenila je vrijednost elementa x_j iz prethodne vrijednosti V_{old} u novu vrijednost V_{new}
 - zapis dnevnika izmjene se stvara kao posljedica operacije *write* (a ne operacije *output*)
 - zapis koji opisuje potvrđivanje transakcije: $\langle \text{commit } T_i \rangle$
 - zapis koji opisuje prekid transakcije: $\langle \text{abort } T_i \rangle$

Točka potvrđivanja



- završena transakcija (*completed transaction*) - u dnevniku je zabilježen ishod transakcije (*commit* ili *abort*). Pri tome je irelevantno jesu li blokovi **podataka** izmijenjeni tijekom transakcije također pohranjeni u postojanu memoriju
- nekompletna transakcija (*incomplete transaction*) - u dnevniku ne postoji niti zapis *abort*, niti zapis *commit*. Pri tome je irelevantno jesu li blokovi **podataka** izmijenjeni tijekom transakcije pohranjeni u postojanu memoriju. Zapis o nekompletnoj transakciji u dnevniku posljedica je pogreške sustava nastale prije nego je u dnevniku zabilježen ishod transakcije (*commit* ili *abort*)

Točka potvrđivanja



- točka potvrđivanja (*commit point*) - transakcija T_i je dosegla točku potvrđivanja u trenutku kada je u stabilnu memoriju upisan zapis dnevnika $\langle \text{commit } T_i \rangle$. Sve izmjene koje je transakcija načinila prije točke potvđivanja mogu se smatrati provizornim ili tentativnim (*tentative*)
- ako se poštuje svojstvo izdržljivosti transakcija, efekti transakcije koja je dosegla točku potvrđivanja niti u kojem slučaju ne bi smjeli biti izgubljeni (bez obzira na vrstu kvara)

Korištenje dnevnika za obnovu nakon pogreške sustava

- u općem slučaju, tijekom postupka obnove nakon pogreške sustava
 - neke transakcije treba ponovo obaviti
 - neke transakcije treba poništiti

T	Buffer manager	p	buff x	buff y	disk x	disk y
begin						
read(x, p)						
$p \leftarrow p * 2$	input(x)	3	3		3	5
write(x, p)		6	3		3	5
$p \leftarrow p + 2$		6	6		3	5
		8	6		3	5
write(y, p)	output(x)	8	6		6	5
commit	input(y)	8	6	8	6	5
	output(y)	8	6	8	6	8

- ako se pogreška sustava dogodila prije operacije *commit*, efekte transakcije T treba pomoću zapisa iz dnevnika poništiti
- ako se pogreška dogodila nakon operacije *commit*, pomoću zapisa iz dnevnika osigurati da su svi efekti transakcije T zapisani u postojanu memoriju

- mehanizam poništavanja efekata transakcije pomoću dnevnika također se koristi **na jednak način**:
 - u slučaju pogreške transakcije
 - u slučaju eksplicitnog prekida transakcije

Korištenje dnevnika za obnovu nakon pogreške medija

- u općem slučaju, tijekom postupka obnove nakon pogreške medija
 - obnoviti bazu podataka pomoću arhivske kopije
 - sustav se vraća u stanje u kojem je bio u trenutku izrade arhivske kopije
 - korištenjem dnevnika izmjena obaviti potvrđene transakcije koje su pokrenute nakon trenutka u kojem je izrađena arhivska kopija

Tehnike obnove

- postoje različite tehnike obnove. O primjenjenoj tehnici ovisi:
 - što sadrži zapis dnevnika izmjene
 - redoslijed obavljanja operacija *output* i *flush log*
 - postupak obnove
- *undo (undo/no-redo)*
 - tijekom obnove neke se transakcije poništavaju
- *redo (no-undo/redo)*
 - tijekom obnove neke se transakcije ponovno obavljaju
- *undo/redo*
 - tijekom obnove neke se transakcije poništavaju, neke ponovno obavljaju
 - najčešće korištena tehnika u današnjim SUBP
- *no-undo/no-redo ili shadow paging* - neće se razmatrati
 - bez dnevnika: koriste se kopije blokova s podacima (stari blok, novi blok, *current directory*, *shadow directory*)

Undo - sadržaj dnevnika i pravila vođenja dnevnika

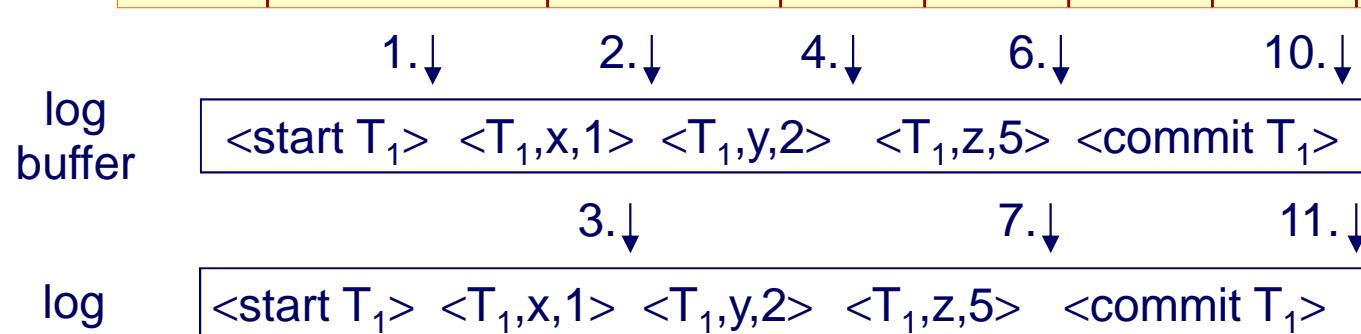
- zapis koji opisuje operaciju pisanja sadrži samo staru vrijednost elementa:
 $\langle T_i, x_j, V_{old} \rangle$
 - transakcija s identifikatorom T_i promijenila je vrijednost elementa x_j čija je **prethodna** vrijednost bila V_{old} na novu vrijednost V_{new}
- pravila vođenja dnevnika:
 1. ako transakcija T obavlja operaciju $write(x, V_{new})$, operacija *flush log* koja će upisati zapis dnevnika $\langle T, x, V_{old} \rangle$ mora biti obavljena prije operacije *output(x)*
 2. operacija *flush log* kojom se zapis dnevnika $\langle commit T \rangle$ upisuje u stabilnu memoriju obavlja se isključivo nakon što su obavljene sve *output(x)* operacije transakcije T

Pravilo koje zahtijeva da se zapisi dnevnika upisuju u stabilnu memoriju **prije** nego se u postojanu memoriju upiše promijenjena vrijednost elementa naziva se *write-ahead logging (WAL)*

Undo - sadržaj dnevnika i pravila vođenja dnevnika

Primjer: ■ integritetska ograničenja: $y = x * 2$; $z = x + 4$;

	T_1	Buffer manager	buff x	buff y	buff z	disk x	disk y	disk z
1.	begin ₁					1	2	5
2.	write ₁ (x, 3)		3			1	2	5
3.		flush log	3			1	2	5
4.	write ₁ (y, 6)		3	6		1	2	5
5.		output(x)	3	6		3	2	5
6.	write ₁ (z, 7)		3	6	7	3	2	5
7.		flush log	3	6	7	3	2	5
8.		output(y)	3	6	7	3	6	5
9.		output(z)	3	6	7	3	6	7
10.	commit ₁		3	6	7	3	6	7
11.		flush log	3	6	7	3	6	7



Undo - postupak obnove nakon pogreške sustava

- *restart recovery*
- za vrijeme obnove, sustav ne zaprima nove transakcije
- čitati dnevnik od kraja prema početku te za svaki zapis iz dnevnika
 - ako pročitaš zapis $\langle \text{commit } T_i \rangle$ dodaj T_i u listu potvrđenih transakcija
 - ako pročitaš zapis $\langle T_i, x, V \rangle$
 - ako je T_i u listi potvrđenih transakcija, činiti ništa*
 - inače ako T_i nije u listi potvrđenih transakcija, obavi operaciju $\text{undo}(x, V)$, tj. promijeni vrijednost elementa x na vrijednost V^{**}
- * ako je u dnevniku zabilježen zapis $\langle \text{commit } T_i \rangle$, tada su sve *output* operacije transakcije T_i sigurno uspješno izvršene već prije pogreške
- ** ako u dnevniku nije zabilježen zapis $\langle \text{commit } T_i \rangle$, možda su neke *output* operacije transakcije T_i obavljene, a neke nisu. Poništiti sve efekte transakcije T_i
- što ako se pogreška sustava opet dogodi za vrijeme obnove?
 - ponavljanje postupka obnove, bez obzira u kojem je trenutku prekinut, neće izazvati probleme jer je operacija *undo* idempotentna
 - $\text{undo}(x, V), \text{undo}(x, V), \dots \equiv \text{undo}(x, V)$

Undo - postupak obnove nakon pogreške sustava

Primjer:

- neka se u prethodnom primjeru pogreška sustava dogodila nakon 5. koraka
- na početku obnove sadržaj dnevnika jest: $\boxed{\langle \text{start } T_1 \rangle \langle T_1, x, 1 \rangle}$
 - tijekom obnove
 - u element y, čija je vrijednost 2, zapisuje se vrijednost 2
 - u element x, čija je vrijednost 3, zapisuje se vrijednost 1

Za vježbu odgovoriti na pitanja:

- što će se desiti ako se pogreška sustava dogodi nakon 8. koraka? Nakon 10. koraka? Nakon 11. koraka?
- što će se desiti ako se pogreška sustava dogodila nakon 8. koraka, a za vrijeme obnove se opet dogodila pogreška nakon obavljanja operacije *undo(y, 2)*

Undo - nedostaci

- potreba obavljanja većeg broja U/I operacija prije potvrđivanja transakcije:
 - zapis dnevnika $\langle \text{commit } T_i \rangle$ mora se u dnevnik upisati čim je prije moguće (zbog izdržljivosti transakcije). S druge strane, $\langle \text{commit } T_i \rangle$ smije biti upisan u dnevnik tek nakon što su obavljene sve pripadne *output* operacije. Zbog toga je menadžer međuspremnika prisiljen tijekom svake transakcije obaviti sve *output* operacije koje su povezane s izmijenjenim elementima. Time se uvećava broj U/I operacija koje se moraju obaviti tijekom transakcije
- ograničena mogućnost obnove baze podataka pomoću arhivske kopije
 - u slučaju pogreške medija (baza podataka je uništena) koristi se arhivska kopija. Dnevnik nije upotrebljiv za ponovno obavljanje transakcija koje su izvršene nakon posljednjeg arhiviranja baze

Redo

Redo - sadržaj dnevnika

- zapis koji opisuje operaciju pisanja sadrži samo novu vrijednost elementa:
 $\langle T_i, x_j, V_{new} \rangle$
 - transakcija s identifikatorom T_i promijenila je vrijednost elementa x_j na **novu vrijednost V_{new}**

Redo - pravila vođenja dnevnika

1. svaka operacija $output(x)$ transakcije T treba se obaviti tek nakon što su **svi** zapisi dnevnika $\langle T, x, V_{new} \rangle$ **i zapis** $\langle commit T \rangle$ transakcije upisani u stabilnu memoriju

Redo

Redo - postupak obnove

- čitati dnevnik **od početka prema kraju**
 - ako za transakciju T_i u dnevniku postoji zapis $\langle \text{commit } T_i \rangle$ tada obaviti operaciju $\text{redo}(x_j, V)$ za svaki zapis $\langle T_i, x_j, V \rangle$ iz dnevnika
 - inače činiti ništa*

* ako u dnevniku ne postoji zapis $\langle \text{commit } T_i \rangle$ tada niti jedna $\text{output}(x)$ operacija transakcije T_i sigurno nije obavljena

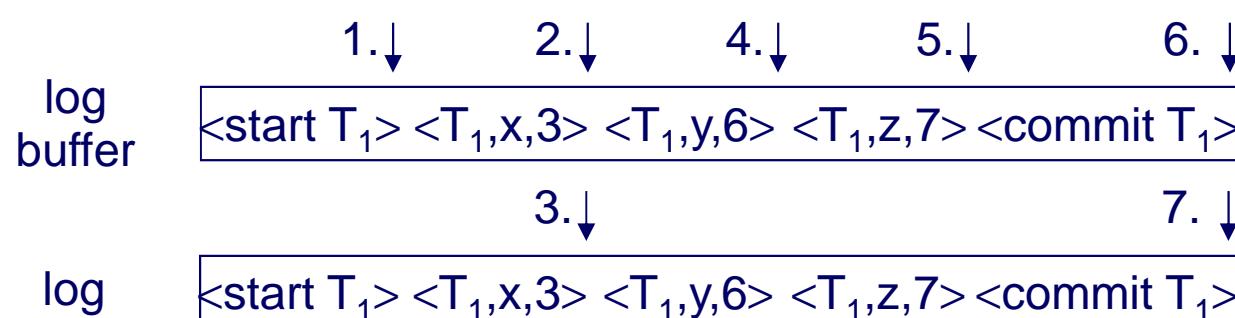
Redo - pogreška sustava tijekom obnove

- ponovno obavljanje opisanog postupka obnove, bez obzira u kojem je trenutku prekinut, neće izazvati probleme jer je operacija *redo* idempotentna
 - $\text{redo}(x, V), \text{redo}(x, V), \dots \equiv \text{redo}(x, V)$

Redo - sadržaj dnevnika i pravila vođenja dnevnika

Primjer: ■ integritetska ograničenja: $y = x * 2$; $z = x + 4$;

	T_1	Buffer manager	buff x	buff y	buff z	disk x	disk y	disk z
1.	begin ₁					1	2	5
2.	write ₁ (x, 3)		3			1	2	5
3.		flush log	3			1	2	5
4.	write ₁ (y, 6)		3	6		1	2	5
5.	write ₁ (z, 7)		3	6	7	1	2	5
6.	commit ₁		3	6	7	1	2	5
7.		flush log	3	6	7	1	2	5
8.		output(x)	3	6	7	3	2	5
9.		output(y)	3	6	7	3	6	5
10.		output(z)	3	6	7	3	6	7



- *output* niti jednog elementa se ne smije obaviti prije nego je commit zapis dnevnika za transakciju zapisan u stabilnu memoriju

Redo - postupak obnove nakon pogreške sustava

Primjer:

- neka se u prethodnom primjeru pogreška sustava dogodila nakon 4. koraka
- na početku obnove sadržaj dnevnika jest:

$\langle \text{start } T_1 \rangle \ \langle T_1, x, 3 \rangle$

- u dnevniku nema zapisa $\langle \text{commit } T_1 \rangle$ \rightarrow činiti ništa

Primjer:

- neka se u prethodnom primjeru pogreška sustava dogodila nakon 8. koraka
- na početku obnove sadržaj dnevnika jest:

$\langle \text{start } T_1 \rangle \ \langle T_1, x, 3 \rangle \ \langle T_1, y, 6 \rangle \ \langle T_1, z, 7 \rangle \ \langle \text{commit } T_1 \rangle$

- tijekom obnove u x se zapisuje vrijednost 3, u y se zapisuje 6, u z se zapisuje 7

Redo - prednosti i nedostaci

Nedostaci (u odnosu na *undo* tehniku):

- zahtjev za velikim međuspremnikom podataka:
 - vrijednosti iz međuspremnika s podacima koje je promijenila transakcija T_i se ne smije upisati u postojanu memoriju (i time osloboditi međuspremnik) prije nego se zapis dnevnika $\langle \text{commit } T_i \rangle$ zapiše u stabilnu memoriju. Zbog tog se pravila uvećava potrebna veličina međuspremnika
 - primjer: transakcija koja je izmijenila veliki broj elemenata i duže vrijeme ne obavi naredbu commit

Prednosti (u odnosu na *undo* tehniku):

- manja ograničenja redoslijeda obavljanja operacija
 - menadžer međuspremnika može po volji odgađati obavljanje operacija *output* i time optimirati broj obavljenih U/I operacija
 - transakcija može biti potvrđena (i njezina izdržljivost garantirana) i onda kada nisu obavljene sve pripadne *output* operacije

Redo - prednosti i nedostaci

Prednosti (u odnosu na *undo* tehniku):

- veća mogućnost obnove baze podataka iz arhivske kopije
 - u slučaju pogreške medija (baza podataka je uništena) koristi se arhivska kopija. Obavljanjem *redo* operacija za zapise dnevnika potvrđenih transakcija baza podataka se može dovesti u stanje koje odgovara stanju neposredno prije pogreške medija

Redo tehnikom (jednako tako i *undo/redo* tehnikom prikazanom kasnije) omogućeno je odgađanje svih *output* operacija tijekom vrlo dugog vremena. U ekstremnom slučaju, SUBP bi mogao odgoditi obavljanje svih *output* operacije sve do trenutka zaustavljanja.

Zašto takvu mogućnost ipak ne bi trebalo koristiti?

Zbog moguće pogreške sustava. Tijekom obnove bi se morao obaviti veliki broj *redo* operacija. Kako se rješava taj problem → kontrolna točka (kasnije u predavanjima)

Undo/Redo

Undo/Redo - sadržaj dnevnika

- zapis koji opisuje operaciju pisanja: $\langle T_i, x_j, V_{old}, V_{new} \rangle$
 - transakcija s identifikatorom T_i promijenila je vrijednost elementa x_j iz prethodne vrijednosti V_{old} u novu vrijednost V_{new}

Undo/Redo - pravila vođenja dnevnika

1. ako transakcija T obavlja operaciju $write(x, V_{new})$, operacija *flush log* kojom je zapis dnevnika $\langle T, x, V_{old}, V_{new} \rangle$ upisan u stabilnu memoriju mora biti obavljena prije operacije $output(x)$
-
- razlika u odnosu na *redo*:
 - nije propisan redoslijed između zapisivanja $\langle commit T_i \rangle$ u stabilnu memoriju i zapisivanja blokova s podacima u postojanu memoriju

Undo/Redo - postupak obnove nakon pogreške sustava

- čitati dnevnik **od početka prema kraju**
 - ako za transakciju T_i u dnevniku postoji zapis $\langle \text{commit } T_i \rangle$ tada obaviti operaciju $\text{redo}(x_j, V_{\text{new}})$ za svaki zapis $\langle T_i, x_j, V_{\text{old}}, V_{\text{new}} \rangle$ iz dnevnika*
- čitati dnevnik **od kraja prema početku**
 - ako za transakciju T_i u dnevniku ne postoji zapis $\langle \text{commit } T_i \rangle$ tada obaviti operaciju $\text{undo}(x_j, V_{\text{old}})$ za svaki zapis $\langle T_i, x_j, V_{\text{old}}, V_{\text{new}} \rangle$ iz dnevnika**

* zbog transakcije koja je potvrđena, a neke od njenih *output* operacije nisu obavljene

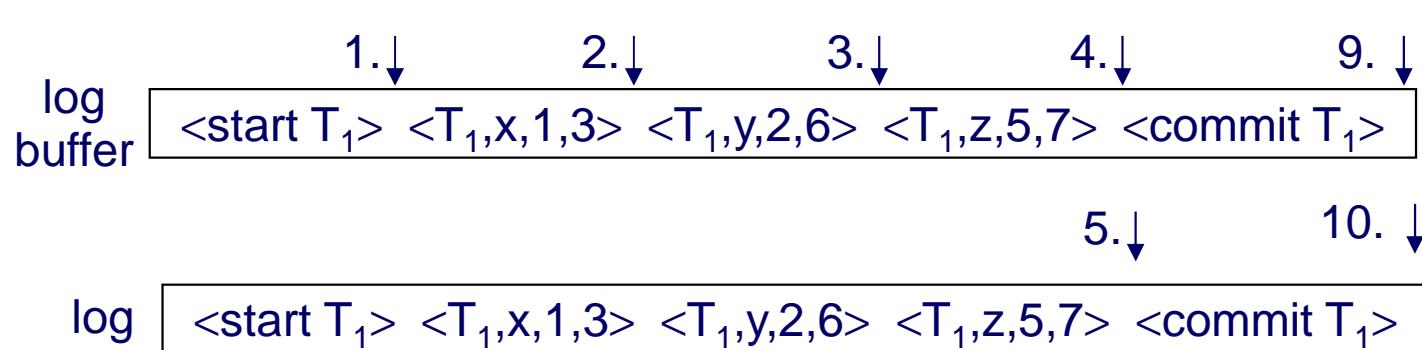
** zbog transakcije koja nije potvrđena, a neke od njenih *output* operacija jesu obavljene

- što ako se pogreška sustava opet dogodi za vrijeme obnove?
 - ponavljanje opisanog postupka obnove, bez obzira u kojem je trenutku prekinut, neće izazvati probleme jer su operacije *undo* i *redo* idempotentne

Undo/Redo - sadržaj dnevnika i pravila vođenja dnevnika

Primjer: ■ integritetska ograničenja: $y = x * 2$; $z = x + 4$;

	T ₁	Buffer manager	buff x	buff y	buff z	disk x	disk y	disk z
1.	begin ₁					1	2	5
2.	write ₁ (x, 3)		3			1	2	5
3.	write ₁ (y, 6)		3	6		1	2	5
4.	write ₁ (z, 7)		3	6	7	1	2	5
5.		flush log	3	6	7	1	2	5
6.		output(x)	3	6	7	3	2	5
7.		output(y)	3	6	7	3	6	5
8.		output(z)	3	6	7	3	6	7
9.	commit ₁		3	6	7	3	6	7
10.		flush log	3	6	7	3	6	7

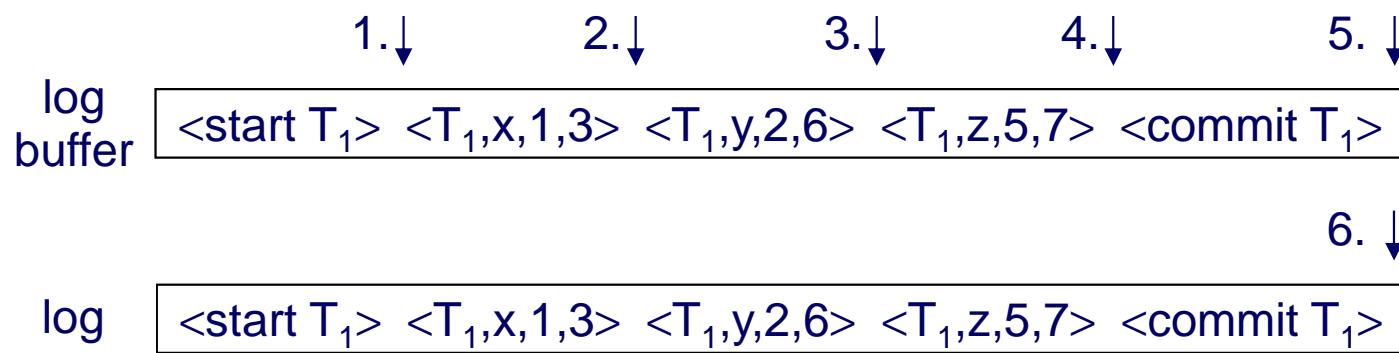


- ako se pogreška sustava dogodi nakon koraka 7. tijekom obnove obaviti će se operacije $undo(z,5)$, $undo(y,2)$ i $undo(x,1)$

Undo/Redo - sadržaj dnevnika i pravila vođenja dnevnika

Primjer: ■ integritetska ograničenja: $y = x * 2$; $z = x + 4$;

	T ₁	Buffer manager	buff x	buff y	buff z	disk x	disk y	disk z
1.	begin ₁					1	2	5
2.	write ₁ (x, 3)		3			1	2	5
3.	write ₁ (y, 6)		3	6		1	2	5
4.	write ₁ (z, 7)		3	6	7	1	2	5
5.	commit ₁		3	6	7	1	2	5
6.	flush log		3	6	7	1	2	5
7.	output(x)		3	6	7	3	2	5
8.	output(y)		3	6	7	3	6	5
9.	output(z)		3	6	7	3	6	7



- ako se pogreška sustava dogodi nakon koraka 7. tijekom obnove obavit će se operacije $redo(x, 3)$, $redo(y, 6)$ i $redo(z, 5)$

Undo/Redo - prednosti i nedostaci pred redo tehnikom

Prednost (u odnosu na *redo* tehniku):

- nema potrebe za velikim međuspremnikom podataka
 - menadžer međuspremnika može (ali ne mora) obaviti *output* operacije neke transakcije T odmah nakon upisivanja pripadnih zapisu dnevnika u stabilnu memoriju. Time što ne mora čekati završetak transakcije (kao kod *redo* tehnike) omogućava se optimalan odabir veličine međuspremnika u odnosu na učestalost obavljanja U/I operacija

Nedostatak (u odnosu na *undo* i *redo* tehniku):

- povećani utrošak prostora za pohranu dnevnika
 - bilježe se i stare i nove vrijednosti elementa koji je promijenjen

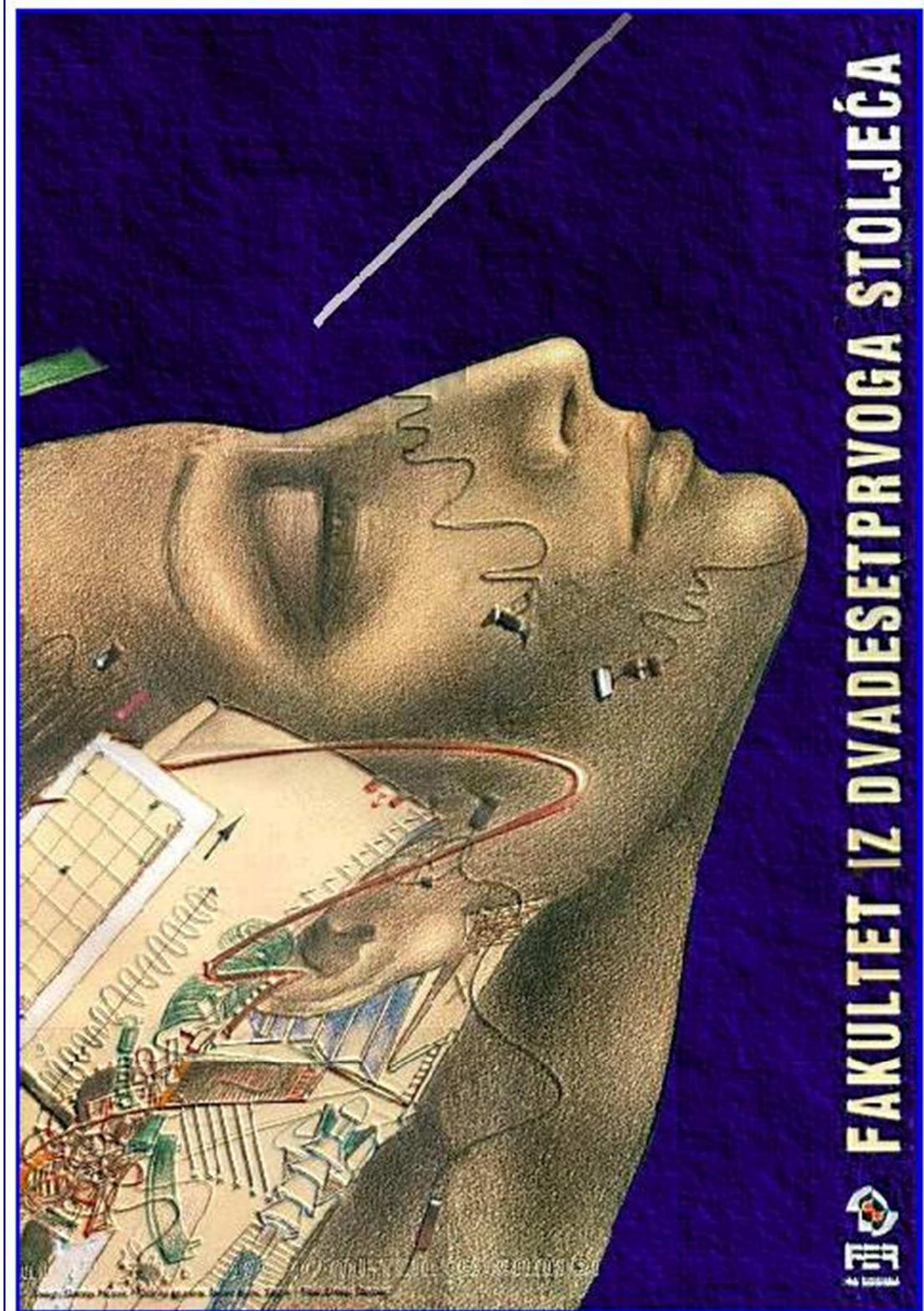
Literatura:

- H. Garcia-Molina, J. Ullman, J. Widom. Database system implementation. Prentice-Hall International, Inc. Upper Saddle River, New Jersey. 2000.
- R. Elmasri, S. Navathe. Fundamentals of database systems, 3rd ed. Addison-Wesley Publishing Company. Reading, Massachusetts. 2000.
- A. Silberschatz, H. Korth, S. Sudarshan. Database system concepts, 4th ed. McGraw-Hill. 2002.
- P. Bernstein, V. Hadzilacos, N. Goodman. Concurrency control and recovery in database systems. Addison-Wesley Publishing Company. Reading, Massachusetts. 1987.
- Archive and Backup Guide for Informix Dynamic Server, Version 7.3, Informix Press. 1998.
- Oracle Database Backup and Recovery Advanced User's Guide 10g Release 2 (10.2)

Sustavi baza podataka

Predavanja

7. Sustav za obnovu
(2. dio)
travanj 2014.



Dnevnik, dnevnik izmjena, fizički dnevnik, logički dnevnik

- aktivnosti transakcija se bilježe u **dnevniku** (*log, journal*)
 - dnevnik obuhvaća niz **zapisa dnevnika** (*log records*)
- jedan oblik implementacije dnevnika je **dnevnik izmjena** (*update log*).
 - sadrži zapise dnevnika izmjena
 - $\langle \text{start } T_i \rangle, \langle T_i, x_j, V_{\text{old}}, V_{\text{new}} \rangle, \langle \text{commit } T_i \rangle, \langle \text{abort } T_i \rangle$
- **novi pojmovi: fizički dnevnik, logički dnevnik**
 - nekonzistentna upotreba ovih pojmoveva u literaturi
 - Silberschatz
 - fizički dnevnik sadrži zapise oblika *old-value, new-value*
 - logički dnevnik sadrži opise *operacija* koje su obavljene
 - Garcia-Molina
 - fizički dnevnik sadrži zapise oblika: *old-block, new-block*
 - logički dnevnik sadrži zapise oblika: *old-value, new-value*
 - u nastavku se koriste termini iz Garcia-Molina

Fizički dnevnik

physical log zapis dnevnika sadrže stare i/ili nove vrijednosti cijelih blokova

- nepotreban utrošak prostora za dnevnik: ako neka operacija izmjene djeluje na manji dio bloka (npr. vrijednost jednog atributa) veliki dio zapisa dnevnika će biti redundantan
- problem poništavanja transakcija

Fizički dnevnik

- primjer za problem poništavanja transakcija:
 - neka se elementi x i y nalaze u istom bloku. Moguć je sljedeći sadržaj dnevnika: $\langle T_1, xy, a_1b_1, a_2b_1 \rangle \langle T_2, xy, a_2b_1, a_2b_2 \rangle \langle T_2 \text{ commit} \rangle \langle T_1 \text{ abort} \rangle$
 - oznakom xy je označen blok koji sadrži n-torce x i y
 - T_1 je promijenila x iz vrijednosti a_1 u a_2 (u fizički dnevnik je upisana i vrijednost elementa y jer se on nalazi u istom bloku)
 - slično, T_2 je promijenila y iz vrijednosti b_1 u b_2
 - poništavanjem T_1 u skladu sa sadržajem prvog zapisa dnevnika, vrijednost bloka xy će se postaviti na a_1b_1 , što je neispravno: izgubljena je promjena koju je obavila T_2
- problem poništavanja transakcija se u ovakovom slučaju može pokušati riješiti upotrebom zaključavanja na razini blokova, ali tada se T_2 ne bi mogla početi izvršavati prije nego završi T_1 (ograničava se mogućnost istodobnog izvršavanja transakcija). O tome će više riječi biti u kasnijim poglavljima.

Logički dnevnik

logical log zapis dnevnika sadrže podatke o izmjenama unutar bloka

- zapisi logičkog dnevnika "na nižoj logičkoj razini" se koriste u slučaju jednostavnijih operacija, npr.
 - u zapisu dnevnika evidentira se vrsta operacije *update*, pozicija podatka unutar bloka, stara i nova vrijednost podatka koji je promijenjen
 - slično, za operacije *insert* i *delete*
- zapisi logičkog dnevnika "na višoj logičkoj razini" se koriste npr. pri obavljanju operacija nad B-stablolom
 - T_1 upiše par (ključ, kazaljka) u čvor B-stabla N
 - T_2 rascijepi čvor N, što rezultira s dva čvora, N i N_1
 - T_2 je potvrđena
 - T_1 je prekinuta i mora se poništiti
 - zapisi logičkog dnevnika moraju omogućiti poništavanje efekta samo transakcije T_1 , tj. brisanje para (ključ, kazaljka iz N ili N_1)

Primjer sadržaja logičkog dnevnika u sustavu IBM IDS

osoba		
oib	ime	prez
12345678901	Ivica	Horvat

Korisnici **miha** i **jure** obavljaju naredbe:

```
(m) BEGIN WORK;
(m) UPDATE osoba SET
    ime='Marko', prez='Ban'
    WHERE oib = '12345678901';
(j) BEGIN WORK;
(j) INSERT INTO osoba
VALUES('22233322233',
'Mendo', 'Slavica');
(m) INSERT INTO osoba
VALUES('3334444555',
'Ivica', 'Horvat');
(m) COMMIT WORK;
(j) COMMIT WORK;
```

# onlog -l -n 3360						
addr	len	type	xid	id	link	
48018	52	BEGIN	22	3360		12/22/2008 19:40:17 31 miha
	34000000	200d0100	00000000	00000000	4...
	[skraćeno]				
4804c	104	HUPDAT	22	48018	200f5e	101 0 31 31 2
	68000000	00004900	12000000	00000000	h.....I.
	00000000	00000000	16000000	18800400	
	7e16e100	5e0f2000	5e0f2000	01010000	~...^. . ^.
	00000000	1f001f00	02000000	00000000	
	7075626c	000b0006	49766963	61204d61	publ.... Ivica Ma	
	726b6f20	00150007	486f7276	61742042	rko Horvat B	
	616e2020	20202020			an	
480b4	52	BEGIN	24	3360		12/22/2008 19:40:29 41 jure
	34000000	200d0100	00000000	00000000	4...
	[skraćeno]				
480e8	96	HINSERT	24	480b4	200f5e	201 31
	[skraćeno]				
	32323233	33333232	3233334d	656e646f	22233322 233Mendo	
	20202020	20536c61	76696361	20202000	Sla vica .	
48148	96	HINSERT	22	4804c	200f5e	102 31
	[skraćeno]				
	33333334	34343435	35352049	76696361	33344445 55 Ivica	
	20202020	20486f72	76617420	20202000	Hor vat .	
481a8	48	COMMIT	22	48148	12/22/2008 19:40:36	
	[skraćeno]					

addr – Log record address

len – record length

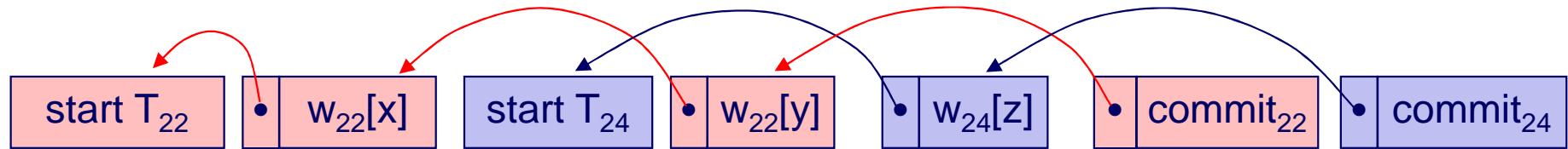
type – record type name

xid – transaction number

id – logical log number

link – link to the previous record in the transaction

Sadržaj dnevnika: omogućavanje poništavanje transakcija



- pokazivač prema prethodnom zapisu iste transakcije u dnevniku
 - omogućava poništavanje transakcija zbog pogreške transakcije ili eksplisitnog zahtjeva za poništavanjem transakcije

Kontrolna točka

Problem:

- za sve do sada prikazane tehnike vrijedi da se u slučaju **pogreške sustava** obnova mora provesti na temelju **cijelog** dnevnika
- koliko je dnevnik velik?
 - ovisi o učestalosti operacija izmjena, veličini podataka koji se mijenjaju, primjenjenoj tehnici obnove

ISVU - Informacijski sustav visokih učilišta u RH (stanje 2009. godine)

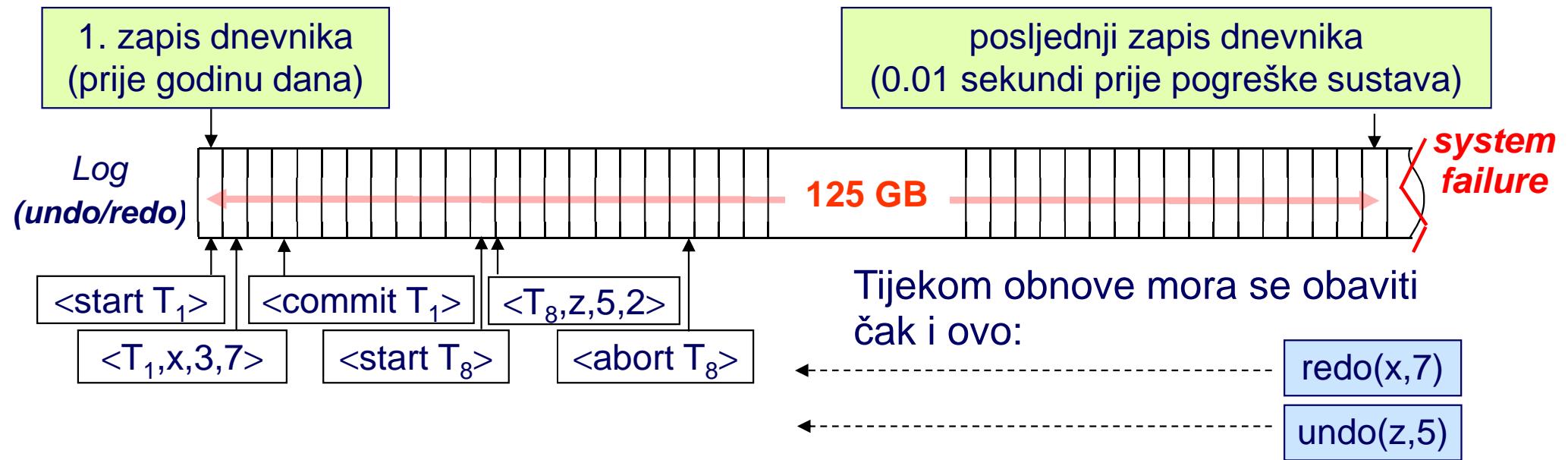
350 000 studenata

1 000 000 izdanih potvrda

5 000 000 ispita

prirast dnevnika: prosječno 350 MB/dan \Rightarrow 125 GB/godinu

Kontrolna točka



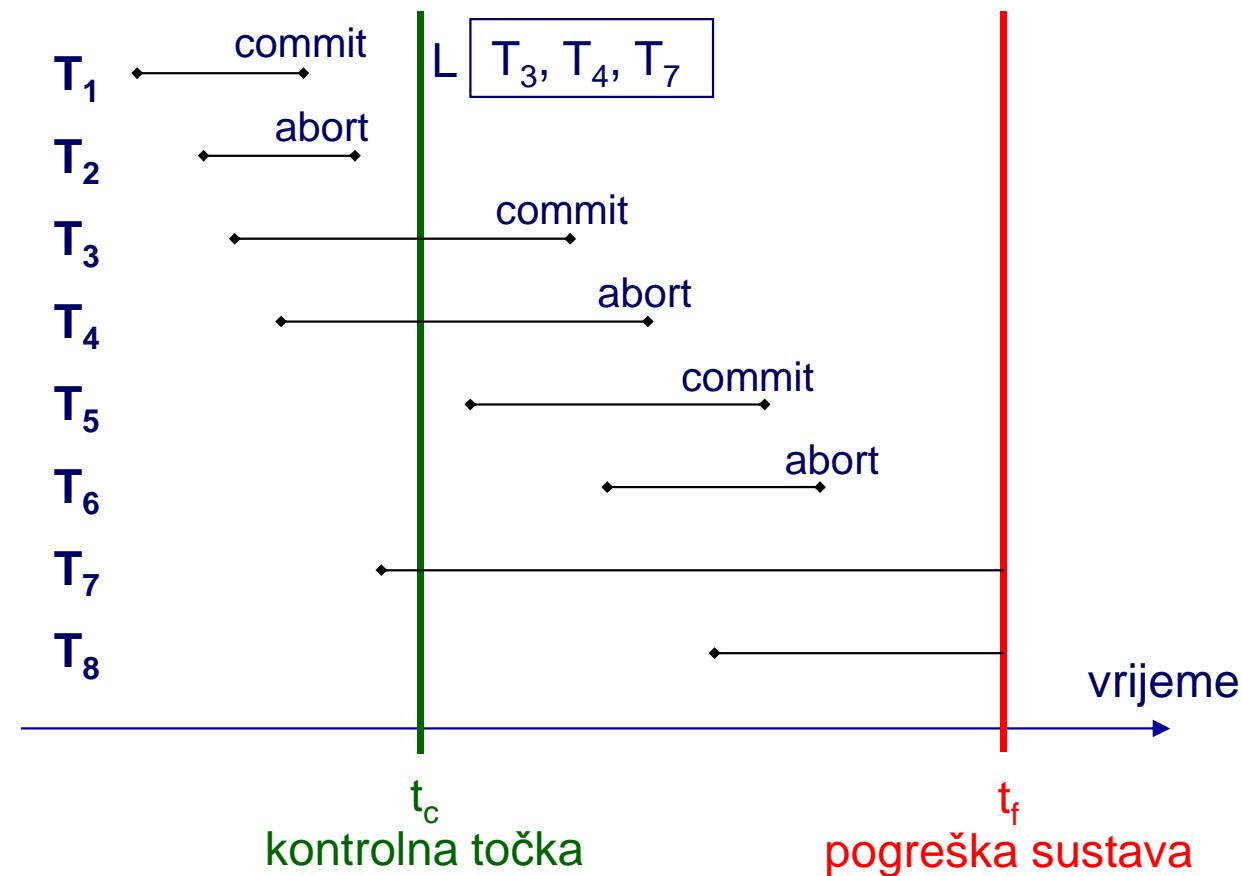
- **Trajanje obnove:** satima? danima?
- Kako bi takav postupak utjecao na sposobnost brzog oporavka (*resilience*) i raspoloživost sustava (*availability*)?

Kontrolna točka

Rješenje: korištenje kontrolnih točaka

- periodički (npr. svakih 10 minuta) sustav određuje kontrolnu točku (*checkpoint*)
 1. SUBP prestaje obavljati operacije koje zahtijevaju pisanje u međuspremnik podataka ili međuspremnik dnevnika
 2. sadržaj međuspremnika dnevnika (*log buffer*) se zapisuje u stabilnu memoriju
 3. sadržaj međuspremnika podataka (*data buffer*) se zapisuje u postojanu memoriju
 4. u dnevnik se upisuje zapis $\langle \text{chkpt } L \rangle^*$
 5. SUBP nastavlja obavljati operacije koje zahtijevaju pisanje u međuspremnik podataka ili dnevnika
- * L je lista identifikatora svih transakcija T_1, T_2, \dots, T_k koje su bile aktivne u trenutku kontrolne točke.

Kontrolna točka



- T_1 i T_2 su započele i završile prije t_c . T_1 je potvrđena, T_2 je poništена
- T_3 i T_4 su započele prije t_c , završile između t_c i t_f . T_3 je potvrđena, T_4 je poništена
- T_5 i T_6 su započele i završile između t_c i t_f . T_5 je potvrđena, T_6 je poništена
- T_7 je započela prije t_c i bila je aktivna u trenutku t_f
- T_8 je započela između t_c i t_f i bila je aktivna u trenutku t_f

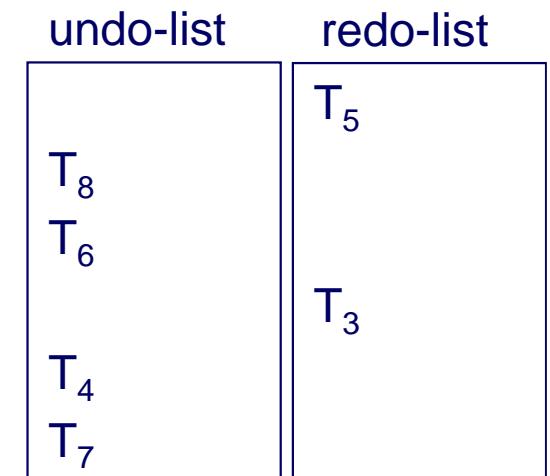
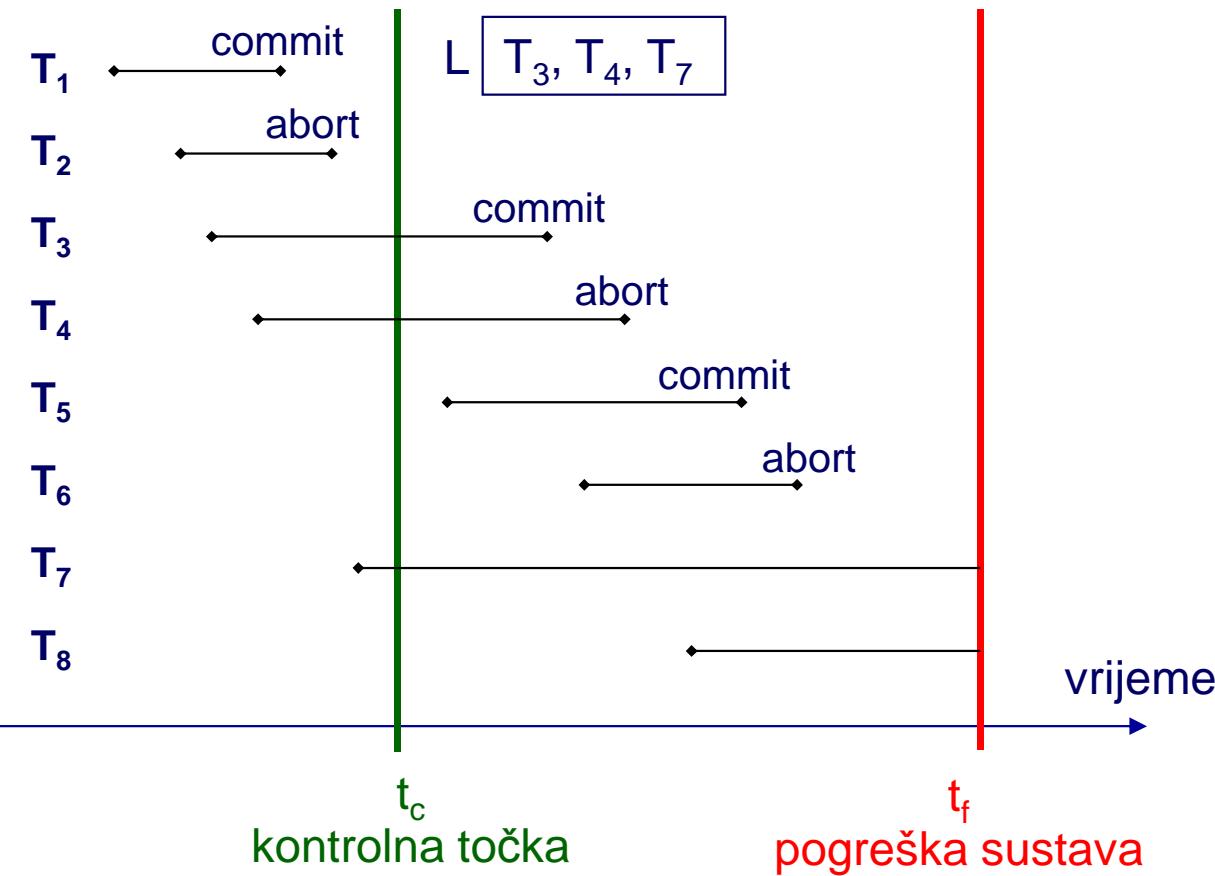
Postupak obnove nakon pogreške sustava

- formiraj dvije prazne liste: redo-list i undo-list
- čitaj dnevnik unatrag od kraja dnevnika do prvog $\langle \text{chkpt } L \rangle$ zapisa
 - za svaki zapis oblika $\langle \text{commit } T_i \rangle$ dodaj T_i u redo-list
 - za svaki zapis oblika $\langle \text{start } T_i \rangle$, ako se T_i ne nalazi u redo-list, dodaj T_i u undo-list
- svaku transakciju T_i iz L koja nije u redo-list dodaj u undo-list

- čitaj dnevnik unatrag od kraja
 - za svaki zapis $\langle T_i, x, V_1, V_2 \rangle$, ako je T_i u undo-list, obavi $undo(x, V_1)$
 - čitanje dnevnika unatrag prestaje kad se pronađu zapisi $\langle \text{start } T_i \rangle$ za svaku transakciju T_i iz undo-list
- čitaj dnevnik od zadnje kontrolne točke do kraja dnevnika (roll forward)
 - za svaki zapis $\langle T_i, x, V_1, V_2 \rangle$, ako je T_i u redo-list, obavi $redo(x, V_2)$

Postupak obnove nakon pogreške sustava

Primjer:



- *undo operacije će se obaviti za sve zapise dnevnika transakcija T_4, T_6, T_7 i T_8*
- *redo operacije će se obaviti za zapise dnevnika transakcija T_3 i T_5 koji su nastali nakon t_c*

Fizički dnevnik i brza obnova u sustavu IBM IDS

- IBM IDS koristi tehniku "brze obnove" (*fast recovery*) čiji je važan element "fizički dnevnik" (*physical log*) u kojem se bilježe predslike blokova podataka
 - predslika bloka (*before image*) je originalna vrijednost bloka podataka, kakva je bila u trenutku posljednje kontrolne točke. Čuvaju se predslike svih blokova koji su izmijenjeni nakon posljednje kontrolne točke.
-
- uočiti: ne zapisuje se zapis oblika *old-block, new-block* - zapisuje se samo *old-block* kakav je bio u trenutku posljednje kontrolne točke

Fizički dnevnik i brza obnova u sustavu IBM IDS

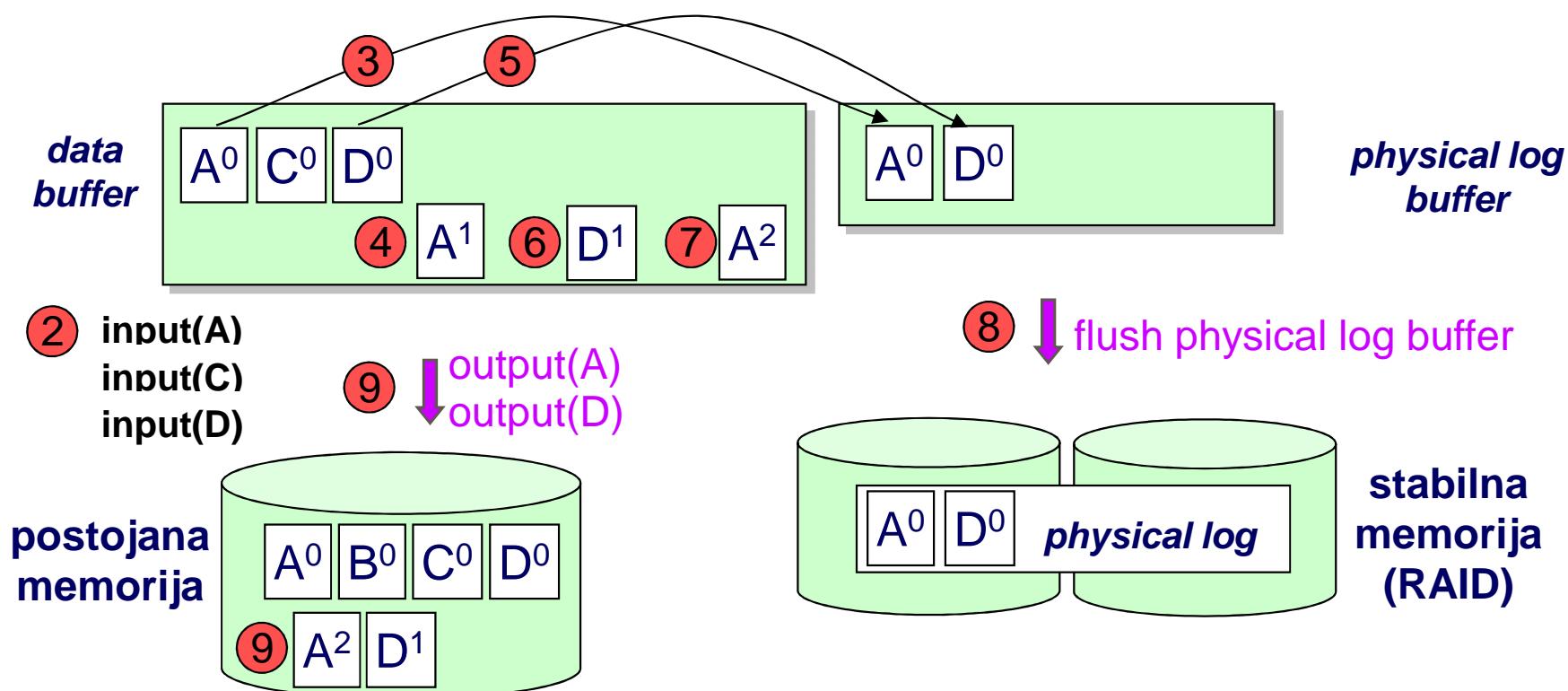
Pravila vođenja fizičkog dnevnika

- fizički dnevnik se počinje puniti predslikama blokova nakon kontrolne točke. Neposredno nakon kontrolne točke fizički dnevnik je prazan!
- prije prve izmjene u bloku međuspremnika podataka, predslika bloka se upisuje u međuspremnik fizičkog dnevnika (*physical log buffer*)
 - blok može biti izmijenjen više puta nakon kontrolne točke, ali se pohranjuje samo jedna predslika
 - ne pohranjuju se predslike blokova koji se nisu promijenili
- prije nego se izmijenjeni blok podataka pohrani u postojanoj memoriji, njegova predslika se obavezno pohranjuje u fizički dnevnik u stabilnoj memoriji
 - posljedica: u stabilnoj memoriji postoji predslika svakog bloka podataka koji je u postojanoj memoriji izmijenjen nakon posljednje kontrolne točke
- u kontrolnoj točki se svi blokovi iz međuspremnika podataka zapisuju u postojanu memoriju, a fizički dnevnik se prazni (jer u tom trenutku više ne postoje predslike koje bi se razlikovale od originala)

Fizički dnevnik i brza obnova u sustavu IBM IDS

- ① **read(A), read(C), read(D)**
- ④ **write(A)**
- ⑥ **write(D)**
- ⑦ **write(A)**

- kopiranje originalne verzije bloka (3. korak) će se obavezno obaviti prije 4. koraka
- u koraku 4. se vrijednost bloka A mijenja: iz A^0 u A^1
- 5. će se obaviti obavezno prije 6.
- prije 7. nije potrebno ponovno pohranjivati predsliku bloka A
- 7. ne zahtijeva zapisivanje nove predslike
- 8. će se obaviti obavezno prije 9.



Fizički dnevnik i brza obnova u sustavu IBM IDS

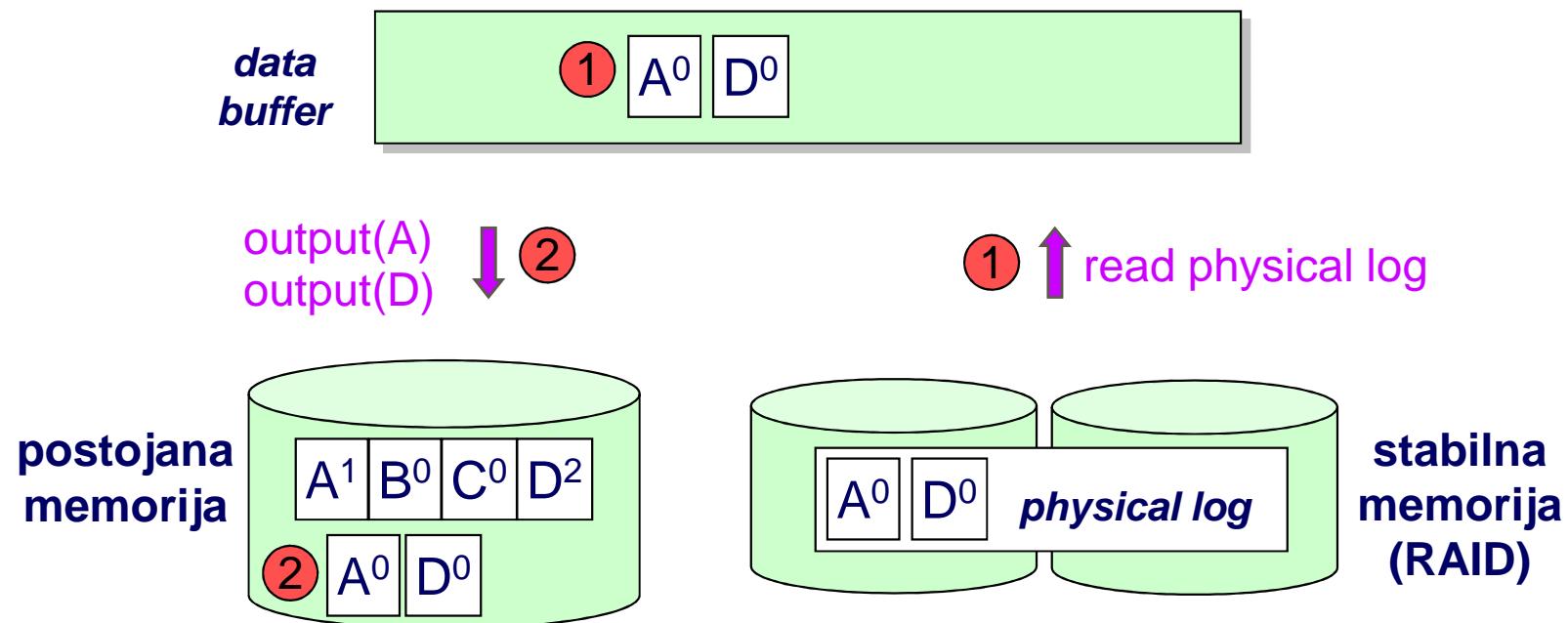
Postupak brze obnove

- ako u trenutku (ponovnog) pokretanja sustava fizički dnevnik nije prazan, može se zaključiti da je sustav bio zaustavljen zbog pogreške sustava
 ⇒ menadžer obnove provodi postupak brze obnove
1. sve predslike iz fizičkog dnevnika prebacuju se iz fizičkog dnevnika u međuspremnike podataka i zapisuju u blokove podataka u postojanoj memoriji. U tom su trenutku blokovi podataka u stanju u kojem su bili u trenutku posljednje kontrolne točke: fizička konzistentnost (*physical consistency*)
 2. obavlja se *redo* operacija za sve zapise dnevnika nastale nakon posljednje kontrolne točke
 3. obavlja se *undo* operacija za zapise dnevnika samo onih transakcija koje nisu bile potvrđene do trenutka u kojem je nastala pogreška sustava
 - tim postupkom mogu biti obuhvaćeni i neki zapisi dnevnika koji su nastali prije posljednje kontrolne točke
 - nakon točke 3. postignuta je logička konzistentnost

Fizički dnevnik NIJE ZAMJENA za logički dnevnik! Fizički dnevnik je dodatni mehanizam u postupku obnove i izrade arhive bez zaustavljanja sustava (*online backup*).

Fizički dnevnik i brza obnova u sustavu IBM IDS

- ilustracija prve faze brze obnove
- blokovima A i D se vrijednost mijenja iz A^1 i D^2 u A^0 i D^0 (stanje u prethodnoj kontrolnoj točki)



- za drugu i treću fazu brze obnove koristi se logički dnevnik izmjena!

Fizički dnevnik i brza obnova u sustavu IBM IDS

- je li baza podataka (sustav) koja je u stanju fizičke konzistentnosti uvijek i logički konzistentna?
 - NE. Za logičku nekonzistentnost je dovoljno da je barem jedna transakcija obavljala izmjene i prije i poslije zadnje kontrolne točke
- što ako se zbog kvara medija izgubi fizički dnevnik
 - nužna je obnova pomoću arhivske kopije baze podataka i logičkih dnevnika (nema mogućnosti za brzu obnovu)

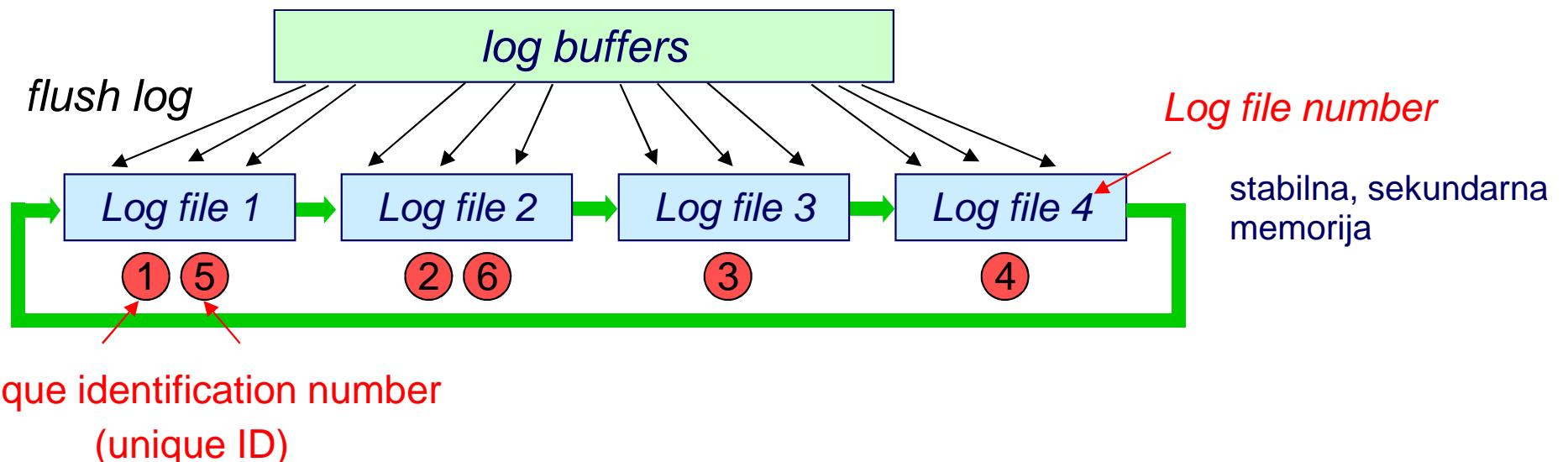
Upravljanje dnevnicima

- s obzirom da sustavi u kojima je prirast dnevnika nekoliko ili nekoliko desetaka GB na dan nisu rijetkost, potreban je efikasan mehanizam upravljanja dnevnicima
- zapisi dnevnika se zapisuju u (*on-line*) stabilnu memoriju (*raw* ili *cooked device*)

Upravljanje dnevnicima

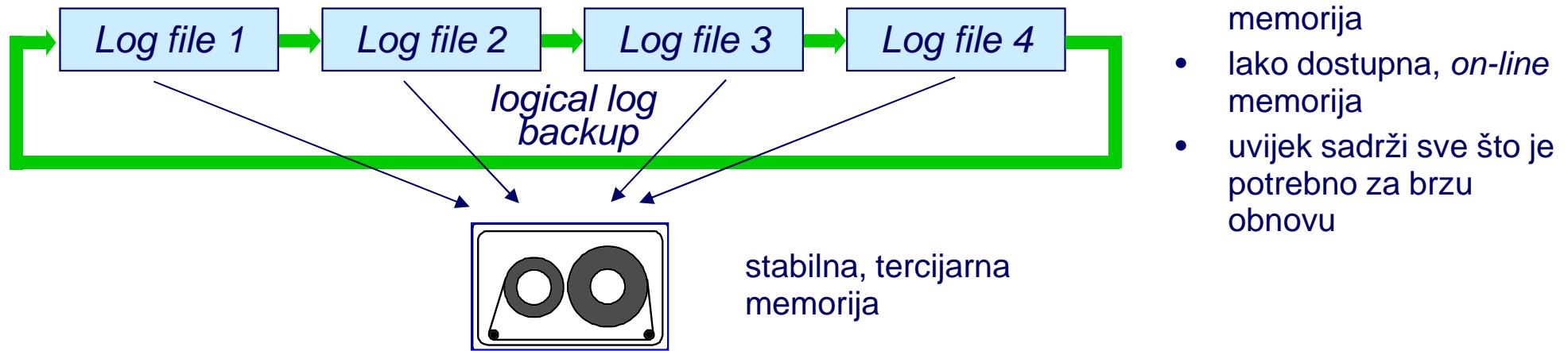
IBM IDS

- prostor za pohranu dnevnika (*logical log*) smješten je u jedan od prostora baze podataka (npr. rootdbspace), u stabilnoj memoriji
- prostor za pohranu dnevnika je podijeljen u zasebne cjeline koje se nazivaju datoteke dnevnika (*logical log file*). Oracle: *redo log group*
- datoteke dnevnika se ciklički izmjenjuju: kad se jedna datoteka dnevnika popuni, zapisi dnevnika se počinju upisivati u sljedeću. Kad se popuni posljednja datoteka dnevnika u nizu, ponovo se koristi prva
- sustav posjeduje najmanje tri datoteke dnevnika, ali se najčešće koristi veći broj, npr. 50 ili 500



Upravljanje dnevnicima

- čim se popuni, datoteka dnevnika se može arhivirati (*logical log backup*)
 - na traku, disk, ili privremeno na disk nakon čega slijedi arhiviranje na traku
 - na udaljeni sustav za pohranu podataka (*remote backup site*)



- stabilna, sekundarna memorija
- lako dostupna, *on-line* memorija
- uvijek sadrži sve što je potrebno za brzu obnovu

Arhiviranjem dnevnika:

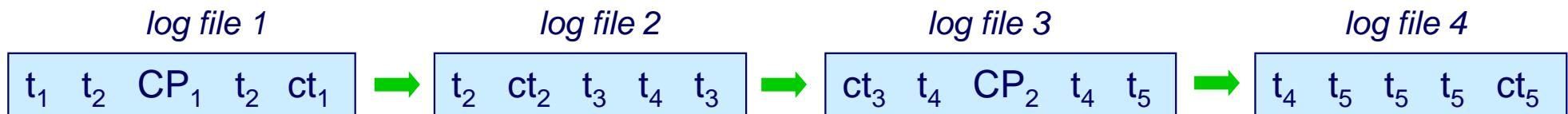
- sadržaj dnevnika se (dodatno) štiti od pogreške medija
 - jer dnevnik je vrlo važan za mogućnost obnove nakon pogreške medija
- omogućava se ponovno korištenje prostora za pohranu dnevnika
- može li se prostor datoteke dnevnika koja je arhivirana odmah početi puniti sa zapisima nove datoteke dnevnika?
 - **ne uvjek!**

Koliko dugo čuvati dnevnik u dostupnoj stabilnoj memoriji

- zapisi dnevnika koji se odnose na trenutno aktivne transakcije potrebni su za poništavanje neispravne (*failed*) transakcije, a u postupku obnove nakon pogreške sustava nužni su svi zapisi dnevnika o transakcijama koje su bile aktivne tijekom posljednje kontrolne točke ili kasnije
- stoga se u lako dostupnoj, stabilnoj sekundarnoj memoriji moraju čuvati:
 - datoteke dnevnika koje još uvijek nisu arhivirane i sve datoteke dnevnika koje slijede iza njih
 - datoteka dnevnika koja sadrži posljednju kontrolnu točku i sve datoteke dnevnika koje slijede iza nje
 - datoteke dnevnika koje sadrže zapise transakcija koje su bile aktivne u trenutku posljednje kontrolne točke i sve datoteke dnevnika koje slijede iza njih

Koliko dugo čuvati dnevnik u dostupnoj stabilnoj memoriji

Primjer:



- t_i je oznaka za zapis dnevnika koji pripada transakciji T_i
- ct_i je oznaka zapisa dnevnika <commit T_i >
- CP je oznaka kontrolne točke
- uz pretpostavku da su sve prikazane datoteke dnevnika arhivirane
 - datoteka dnevnika 3. (i sve datoteke koje slijede) se mora čuvati jer datoteka dnevnika 3. sadrži posljednju kontrolnu točku
 - datoteka dnevnika 2. (i sve datoteke koje slijede) se mora čuvati jer sadrži zapise transakcije T_4 koja je bila aktivna tijekom posljednje kontrolne točke

Problem dugih transakcija

- transakcije koje relativno dugo traju ili produciraju veliki broj zapisa dnevnika mogu uzrokovati stanje u kojem na raspolaganju nema slobodnih datoteka dnevnika
 - ⇒ blokiranje sustava
 - npr. ako transakcija T_4 iz prethodnog primjera ne završi prije nego na red za punjenje dođe datoteka *log file* 2, sustav prestaje zaprimati nove transakcije

Problem dugih transakcija

IBM IDS - Rješenja

1. sustav se može konfigurirati tako da se u slučaju potrebe omogući automatsko ili ručno dodavanje datoteka dnevnika
 - potrebno je unaprijed predvidjeti prostor u stabilnoj memoriji
- ili
2. DBA definira granične vrijednosti ukupne popunjenoosti dnevnika (LTXHWM i LTXEHWM)
 - prvi znak za opasnost (*LTXHWM-Long Transaction High Water Mark*): npr. nakon što je 70% ukupne veličine dnevnika popunjeno, sustav prestaje prihvaćati nove transakcije
 - drugi znak za opasnost (*LTXEHWM-Long Transaction Exclusive access High Water Mark*): npr. nakon što je 90% ukupne veličine dnevnika popunjeno, sustav počinje poništavati sve aktivne transakcije

Kako ispravno odrediti veličinu i broj datoteka dnevnika

IBM IDS

- LOGSIZE: veličina datoteke dnevnika ⇒ ukupna veličina dnevnika = LOGSIZE×LOGFILES
- LOGFILES: broj datoteka dnevnika
- pri određivanju ukupne veličine dnevnika treba uzeti u obzir:
 - tipično vrijeme trajanja transakcija (prevladavaju li konverzacijalne transakcije, *on-line* transakcije ili *batch* transakcije)
 - broj operacija koje transakcije obavljaju
 - vrstu operacija koje transakcije obavljaju (čitanje ili pisanje)
 - opterećenje sustava (može produljiti vrijeme odziva, odnosno trajanje transakcija)
- pri određivanju veličine datoteke dnevnika treba uzeti u obzir:
 - veće datoteke dnevnika pružaju bolje performance
 - vrijedi li riskirati?
 - koliko transakcija smije biti izgubljeno u slučaju kvara medija na koji se pohranjuju datoteke dnevnika?
 - male datoteke dnevnika će biti brže arhivirane

It is difficult to predict how much logical-log space your database server system requires until the system is fully in use (IBM Informix Dynamic Server Performance Guide)

Obnova nakon pogreške medija

Problem:

- sadržaj postojane (ili možda čak i *on-line* stabilne) memorije je izgubljen
 - sadržaj baze podataka u postojanoj memoriji
 - datoteke dnevnika u *on-line* stabilnoj memoriji

Rješenje:

- korištenje redundantnih kopija (arhiva) sadržaja postojane i stabilne *on-line* memorije
- cilj: minimizirati vjerojatnost da će sve kopije podataka potrebnih za obnovu baze podataka biti uništene

Obnova nakon pogreške medija

Smještaj redundantnih kopija:

- postojanje većeg broja kopija smanjuje vjerojatnost gubitka svih kopija, ali važnije od broja kopija je mjesto na kojem se te kopije čuvaju!
 - kopija se moraju čuvati na mediju s nezavisnim modalitetom kvarova (*independant failure mode*). To znači da niti jedan zasebni događaj neće uzrokovati istovremeni kvar oba medija

događaj	primjer uređaja s nezavisnim modalitetom kvarova
kvar diska	jedinica trake u istom računalu
kvar kontrolera	disk jedinica u istom računalu na zasebnom kontroleru
požar u prostoriji	disk jedinica u susjednoj prostoriji
požar u zgradи	jedinica trake u susjednoj zgradи
katastrofalan potres	disk jedinica u drugom, dovoljno udaljenom gradu

Arhiviranje baze podataka

- periodičko kopiranje cijelog sadržaja baze podataka (*database dump*, *database backup*) na uređaj s nezavisnim modalitetom kvarova
- u slučaju kvara medija, baza podataka se uz pomoć te kopije može vratiti u stanje u kojem je bila **u trenutku izrade posljednje arhive**

Problemi i rješenja:

- potreba zaustavljanja sustava za vrijeme izrade arhive (*quiescent archiving*)
 - *non-quiescent archiving* (IBM IDS: *on-line backup*, Oracle 11g: *hot backup*)
- ako se sustav vrati u stanje u kojem je bio pri izradi posljednje arhive, bit će izgubljen rezultat svih transakcija koje su obavljene u vremenu nakon izrade posljednje arhive
 - arhiviranje svih datoteka dnevnika nastalih nakon posljednje arhive
- postupak arhiviranja *cijele* baze podataka je dugotrajan i zahtijeva puno prostora; obnova u posljednje konzistentno stanje pomoću svih dnevnika nastalih prije davno napravljene arhive može (pre)dugo trajati
 - inkrementalno arhiviranje

Arhiviranje baze podataka bez zaustavljanja sustava

IBM IDS

- *on-line backup*
 - za vrijeme izrade arhive bez zaustavljanja sustava dolazi tek do manjeg usporavanja rada sustava
- u trenutku kada *započinje* izrada arhive inicira se kontrolna točka (*archive checkpoint*) i zabilježi se vremenska oznaka (*timestamp*) početka arhiviranja TS
- u arhivu se zapisuju blokovi podataka čija je vremenska oznaka zadnje izmijene manja od vremenske oznake TS
- za blokove podataka čija je vremenska oznaka zadnje izmijene veća od TS, umjesto blokova podataka zapisuju se njihove predslike iz fizičkog dnevnika
- ako se fizički dnevnik za vrijeme izrade arhive mora isprazniti, predslike koje su potrebne u postupku arhiviranja upisuju se u privremeni prostor (*temporary database space*). Privremeni prostor se prazni nakon završetka arhiviranja

Arhiviranje datoteka dnevnika

- koliko dugo čuvati arhivske kopije datoteka dnevnika
 - za obnovu nakon pogreške sustava dovoljne su datoteke dnevnika koje se nalaze u stabilnoj memoriji (*on-line*)
- za obnovu nakon pogreške medija potrebna je
 - posljednja arhiva
 - sve datoteke dnevnika koje su nastale nakon početka izrade posljednje arhive
- preporuča se čuvati barem tri posljednje arhive i sve pripadne datoteke dnevnika
 - u slučaju gubitka posljednje arhive (npr. *tape read error*), moguća je obnova uz pomoć prethodne arhive i svih dnevnika nastalih nakon te arhive

Arhiviranje datoteka dnevnika

Primjer:



- obnova sustava je moguća pomoću:
 - arhiva 29.1. + datoteke dnevnika 2719 - 2720 ili
 - arhiva 22.1. + datoteke dnevnika 1253 - 2720 ili
 - arhiva 15.1. + datoteke dnevnika 115 - 2720

Inkrementalno arhiviranje baze podataka

- arhiva razine 0 (*full backup*): arhiviranje **svih** blokova podataka
- **problem:** broj zapisa dnevnika koje treba obraditi tijekom obnove nakon pogreške medija
- **rješenje:** često izrađivati arhivu razine 0
- **problem:** česta izrada arhive razine 0 smanjila bi veličinu dnevnika, ali dovodi do problema prostora potrebnog za pohranu arhive i trajanja izrade arhive (performance sustava)
- **rješenje:** česta izrada arhive samo onih blokova podataka koji su promijenjeni nakon posljednje arhive
 - arhiviranje na više razina
 - arhiva razine i : arhiviranje blokova podataka koji su promijenjeni nakon posljednje arhive razine $i - 1$

Inkrementalno arhiviranje NIJE zamjena za dnevnik! Inkrementalnim arhiviranjem se samo smanjuju broj zapisa dnevnika koje treba primijeniti tijekom obnove.

Inkrementalno arhiviranje baze podataka

IBM IDS

- arhiva razine 0: *full backup*
- arhiva razine 1: blokovi podataka koji su promijenjeni nakon posljednje arhive razine 0
- arhiva razine 2: blokovi podataka koji su promijenjeni nakon posljednje arhive razine 1

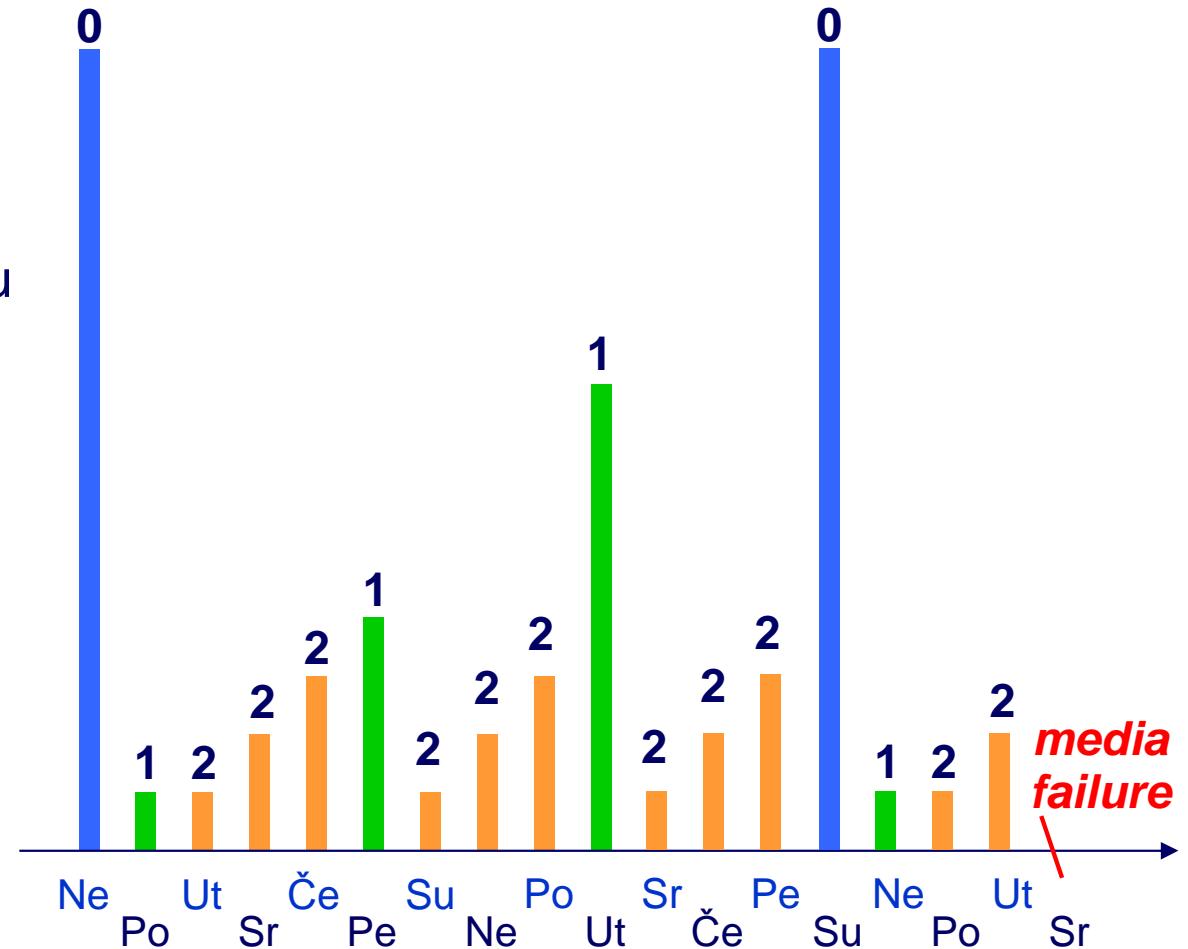
Oracle 11g

- arhiva razine 0: *full backup*
- kumulativna arhiva razine 1: blokovi podataka koji su promijenjeni nakon posljednje arhive razine 0
 - veći utrošak prostora, ali brža obnova: nakon obnove iz arhive razine 0 potrebno je primijeniti samo posljednju arhivu razine 1
- diferencijalna arhiva razine 1: blokovi podataka koji su promijenjeni nakon posljednje arhive razine 0 ili 1
 - manji utrošak prostora, ali sporija obnova: nakon obnove iz arhive razine 0 potrebno je primijeniti sve arhive razine 1 nastale nakon izrade arhive razine 0

Inkrementalno arhiviranje baze podataka

Primjer: IBM IDS

- visinom stupca ilustrirana je veličina arhive. Boja stupca i broj na vrhu stupca označavaju razinu arhive
- slika je nacrtana s pretpostavkom visoke volatilnosti baze podataka i jednolike distribucije blokova koji se mijenjaju: u svakom vremenskom intervalu promijenjen je jednak broj novih blokova



- obnova nakon pogreške medija u trenutku označenom na slici moguća je pomoću:
 - zadnje arhive razine 0 + zadnje arhive razine 1 + zadnje arhive razine 2 + datoteka dnevnika nastalih nakon početka izrade zadnje arhive razine 2

Inkrementalno arhiviranje baze podataka

- pri definiranju sheme arhiviranja u obzir treba uzeti karakterističnu periodičku raspodjelu opterećenosti sustava (ako takva postoji). Mnogi sustavi (**ne svi**) su nedjeljom manje opterećeni u odnosu na druge dane u tjednu, te manje opterećeni noću u odnosu na druge dijelove dana. Ovdje su prikazani primjeri nekih od shema arhiviranja zasnovanih na toj pretpostavci:
 - Shema 1:**
 - arhiva razine 0: svake četvrte nedjelje
 - arhiva razine 1: svake nedjelje
 - arhiva razine 2: tijekom noćnih sati svakog dana osim nedjelje
 - arhiviranje datoteka dnevnika: non-stop
 - Shema 2:**
 - arhiva razine 0: svake nedjelje
 - arhiva razine 1: svake noći osim nedjelje
 - arhiviranje datoteka dnevnika: non-stop
 - Shema 3:**
 - arhiva razine 0: svake nedjelje
 - arhiviranje datoteka dnevnika: non-stop
- Oracle 11g - *a good rule of thumb*: izradu arhive razine 0 pokrenuti onda kada se veličina arhive razine 1 poveća do otprilike 20% veličine arhive 0

Performance sustava ↔ mogućnost obnove

IBM IDS

- baza podataka se s obzirom na način postupanja s logičkim dnevnicima može kreirati na tri različita načina
- CREATE DATABASE dbname
 - ne kreiraju se zapisi logičkog dnevnika. Nije moguće poništavanje transakcija niti poništavanje na razini naredbe (*statement level rollback*)
 - može se koristiti npr. pri inicijalnom punjenju baze podataka
- CREATE DATABASE dbname WITH BUFFERED LOG
 - zapisi logičkog dnevnika se ne zapisuju u stabilnu memoriju tijekom potvrđivanja transakcije. Naizgled potvrđene transakcije mogu biti poništene u slučaju pogreške sustava
 - koristiti samo onda kada može dopustiti gubitak određenog broja transakcija
- CREATE DATABASE dbname WITH LOG
 - zapisi logičkog dnevnika koji pripadaju nekoj transakciji zapisuju se u stabilnu memoriju najkasnije u trenutku potvrđivanja te transakcije

Performance sustava ↔ mogućnost obnove

Oracle 11g

- COMMIT WRITE BATCH NOWAIT
- naredba kojom se procesu *log writer (LGWR)* dopušta odgađanje zapisivanje zapisa iz međuspremnika logičkog dnevnika u *redo log* datoteke

Veličina prostora za pohranu ↔ mogućnost obnove

IBM IDS

- arhiviranje datoteka dnevnika se može suspendirati parametrom LTAPEDEV u konfiguracijskoj datoteci
 - **LTAPEDEV NUL** (Windows OS) ili **LTAPEDEV /dev/null** (Unix/Linux OS)
 - obnova nakon pogreške sustava je moguća, ali obnova nakon pogreške medija se može provesti samo do stanja u trenutku izrade posljednje archive
- korisno pri obavljanju većih administrativnih zahvata u kontroliranim uvjetima, npr. kreiranje i inicijalno punjenje baze podataka podacima iz tekstualnih datoteka (*database load*). U slučaju pogreške medija cijeli se postupak jednostavno ponavlja od početka

Oracle 11g

- vrlo slično, s jednakim posljedicama, arhiviranje *redo log* datoteka se može suspendirati korištenjem modaliteta NOARCHIVELOG

Osigurati svojstvo izdržljivosti transakcije!

- ne dopustiti gubitak efekata potvrđenih transakcija. Potrebno je:
 - spriječiti gubitak zapisa o potvrđenim transakcijama iz međuspremnika dnevnika (kao posljedice pogreške sustava)
 - ne koristiti opcije poput WITH BUFFERED LOG
 - spriječiti gubitak sadržaja dnevnika (kao posljedice pogreške medija)
 - za dnevnik koristiti stabilnu memoriju, a datoteke dnevnika arhivirati na medij s nezavisnim modalitetom kvarova
 - ne koristiti prevelike pojedinačne datoteke dnevnika
 - spriječiti gubitak sadržaja baze podataka (kao posljedice pogreške medija)
 - arhivirati bazu podataka na medij s nezavisnim modalitetom kvarova

Literatura:

- H. Garcia-Molina, J. Ullman, J. Widom. Database system implementation. Prentice-Hall International, Inc. Upper Saddle River, New Jersey. 2000.
- R. Elmasri, S. Navathe. Fundamentals of database systems, 3rd ed. Addison-Wesley Publishing Company. Reading, Massachusetts. 2000.
- A. Silberschatz, H. Korth, S. Sudarshan. Database system concepts, 4th ed. McGraw-Hill. 2002.
- P. Bernstein, V. Hadzilacos, N. Goodman. Concurrency control and recovery in database systems. Addison-Wesley Publishing Company. Reading, Massachusetts. 1987.
- IBM Informix Backup and Restore Guide, Version 11.70, Informix Product Library, 2011.
- Oracle Database Backup and Recovery Advanced User's Guide 11g Release 2 (11.2)

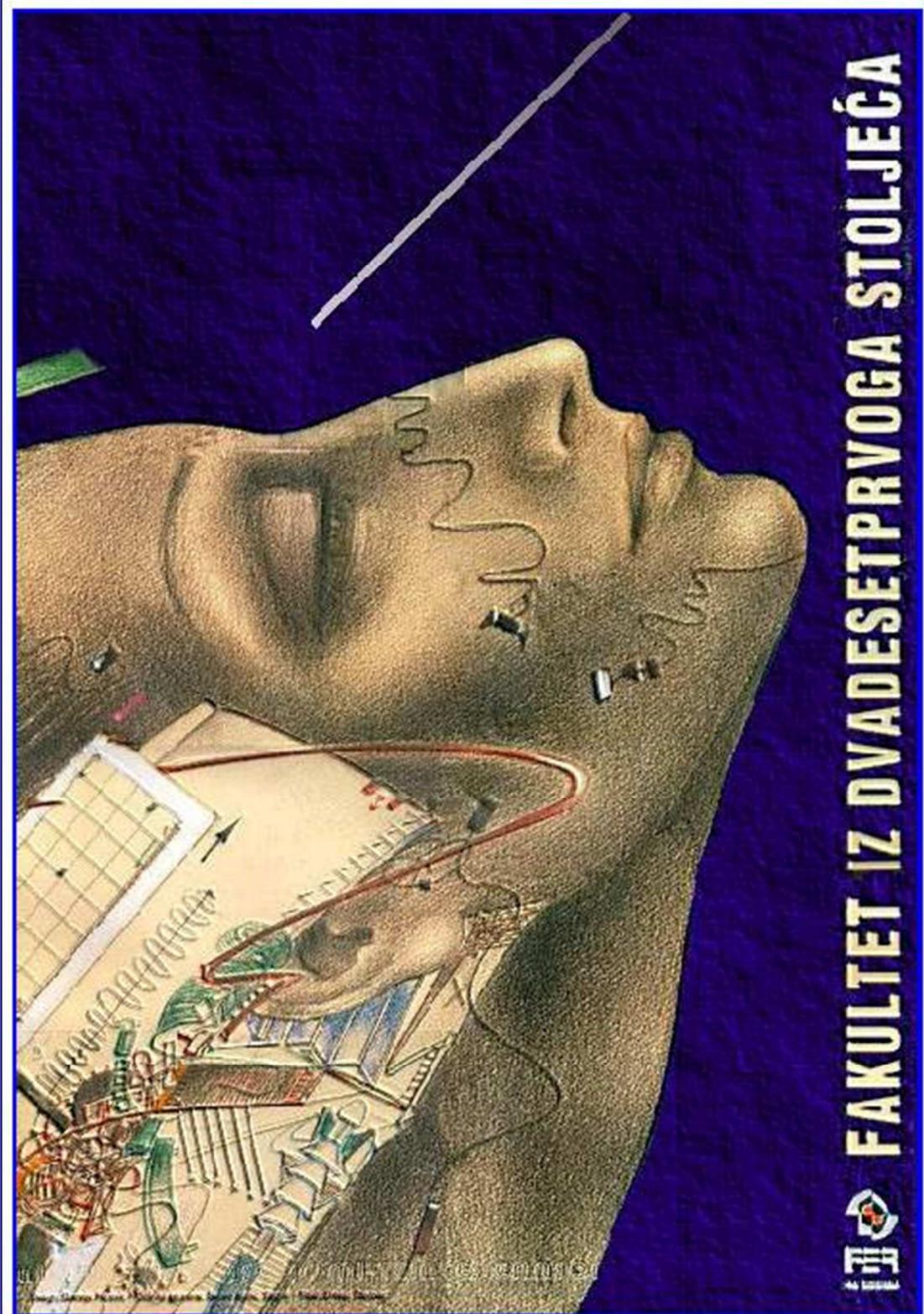
Sustavi baza podataka

Predavanja

8. Kontrola istodobnog pristupa

(1. dio)

svibanj 2014.



Jednokorisnički i višekorisnički SUBP

- jednokorisnički (*single-user*) SUBP
 - u jednom trenutku aktivna je najviše jedna korisnička sjednica (*user session*)
 - transakcije se mogu izvršavati isključivo serijski: tek kad jedna završi, može započeti sljedeća
- višekorisnički (*multiuser*) SUBP
 - istodobno aktivne korisničke sjednice

Višekorisnički SUBP

Jednostavan, ali neefikasan način korištenja višekorisničkog SUBP-a

- transakcije se izvršavaju jedna iza druge (serijsko izvršavanje transakcija) - sustav počinje izvršavati sljedeću transakciju tek kad je prethodna završena
 - slaba ukupna iskoristivost sustava: npr. za vrijeme dok transakcija obavlja slijedno čitanje velike relacije, procesor je slabo iskorišten
 - relativno kratka transakcija može (nepredvidivo) dugo čekati na završetak neke druge relativno dugе transakcije

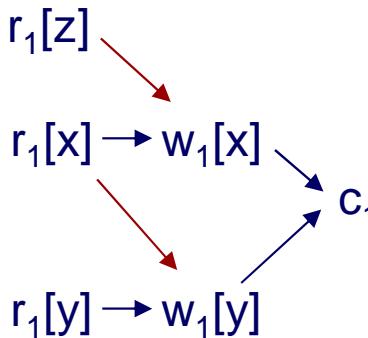
Bolje rješenje:

- paralelno obavljanje operacija različitih transakcija omogućilo bi istodobno korištenje različitih resursa računala, tj. efikasno korištenje svih dijelova računala
 - uvećava se ukupna iskoristivost sustava (*utilization*) i broj transakcija obavljen u jedinici vremena (*throughput*)
 - prosječno vrijeme koje protekne između aktiviranja i završetka transakcije (*average response time*) se smanjuje

Ponavljanje: model transakcije

Primjer: ■ za transakciju T_1 opisanu pseudokodom nacrtati graf transakcije

```
 $T_1$ 
read(x, p)
read(z, q)
 $r \leftarrow p + q$ 
write(x, r)
read(y, s)
 $s \leftarrow s + p$ 
write(y, s)
commit
```



Korektne transakcije i svojstvo izolacije (ACID)

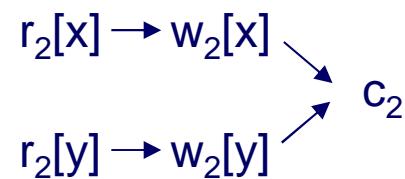
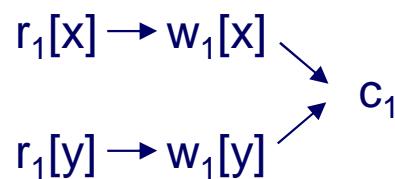
- Primjer:
- dva objekta u bazi podataka (početne vrijednosti $x = 25$, $y = 25$)
 - integritetsko ograničenje: $x = y$

T_1

```
read(x, p)
p ← p + 100
write(x, p)
read(y, r)
r ← r + 100
write(y, r)
```

T_2

```
read(x, p)
p ← p * 2
write(x, p)
read(y, r)
r ← r * 2
write(y, r)
```



- transakcije su **korektne**
 - korektna transakcija prevodi bazu podataka iz jednog konzistentnog u drugo konzistentno stanje
 - ako se korektne transakcije izvršavaju međusobno izolirano (bez interferencije), tada neće narušiti konzistentnost baze podataka
 - ne zaboraviti: korektna transakcija može narušiti konzistentnost baze podataka i u slučaju pogreške, ali to ovdje nije predmet razmatranja

Serijsko izvršavanje transakcija

Primjer:

a) redoslijed T_1, T_2

T_1	T_2	x	y
		25	25
read(x, p)			
$p \leftarrow p + 100$			
write(x, p)		125	
read(y, r)			
$r \leftarrow r + 100$			
write(y, r)		125	
	read(x, p)		
	$p \leftarrow p * 2$		
	write(x, p)	250	
	read(y, r)		
	$r \leftarrow r * 2$		
	write(y, r)	250	

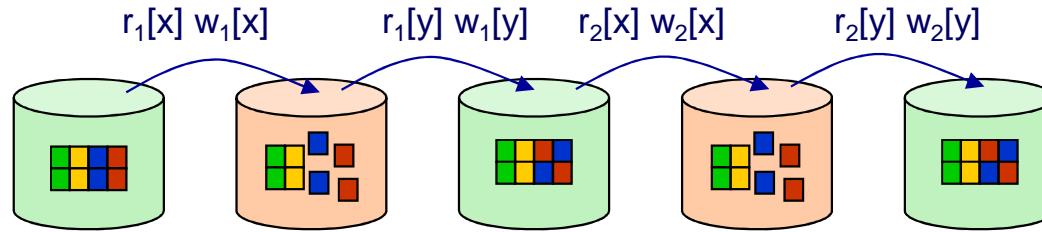
b) redoslijed T_2, T_1

T_1	T_2	x	y
		25	25
	read(x, p)		
	$p \leftarrow p * 2$		
	write(x, p)	50	
	read(y, r)		
	$r \leftarrow r * 2$		
	write(y, r)	50	
	read(x, p)		
	$p \leftarrow p + 100$		
	write(x, p)	150	
	read(y, r)		
	$r \leftarrow r + 100$		
	write(y, r)	150	

primjer iz [Garcia-Molina]

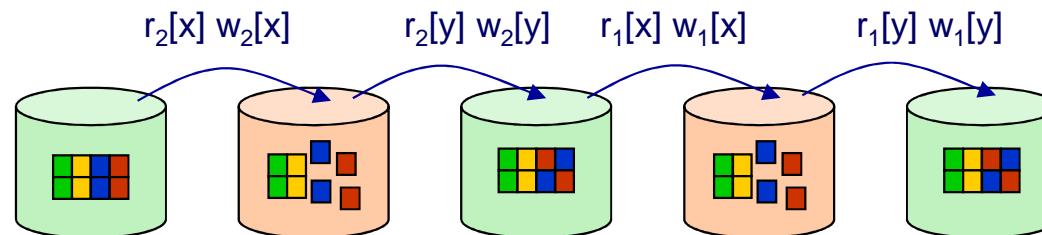
Serijsko izvršavanje transakcija

a) serijski
 T_1, T_2



rezultat je
konzistentno stanje
baze podataka

b) serijski
 T_2, T_1



rezultat je
konzistentno stanje
baze podataka

Istodobno izvršavanje transakcija

c) redoslijed izvršavanja koji narušava konzistentnost baze podataka

T_1	T_2	x	y
		25	25
read(x, p)			
$p \leftarrow p + 100$			
write(x, p)		125	
	read(x, p)		
	$p \leftarrow p * 2$		
	write(x, p)	250	
	read(y, r)		
	$r \leftarrow r * 2$		
	write(y, r)	50	
read(y, r)			
$r \leftarrow r + 100$			
write(y, r)		x ≠ y	
			150

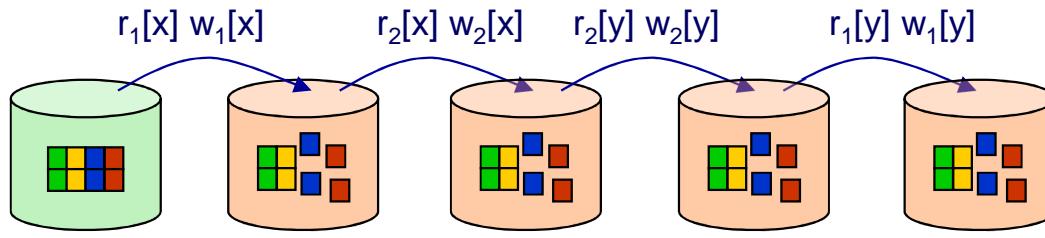
d) redoslijed izvršavanja koji ne narušava konzistentnost baze podataka

T_1	T_2	x	y
		25	25
read(x, p)			
$p \leftarrow p + 100$			
write(x, p)		125	
	read(x, p)		
	$p \leftarrow p * 2$		
	write(x, p)	250	
	read(y, r)		
	$r \leftarrow r + 100$		
	write(y, r)	50	
read(y, r)			
$r \leftarrow r * 2$			
write(y, r)		125	
	read(y, r)		
	$r \leftarrow r * 2$		
	write(y, r)	O.K.	
			250

Istodobno izvršavanje transakcija

c) istodobno

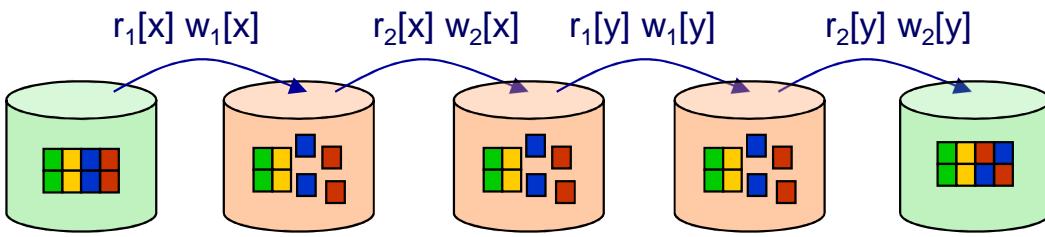
T_1
 T_2



rezultat je
nekonzistentno
stanje baze
podataka

d) istodobno

T_1
 T_2

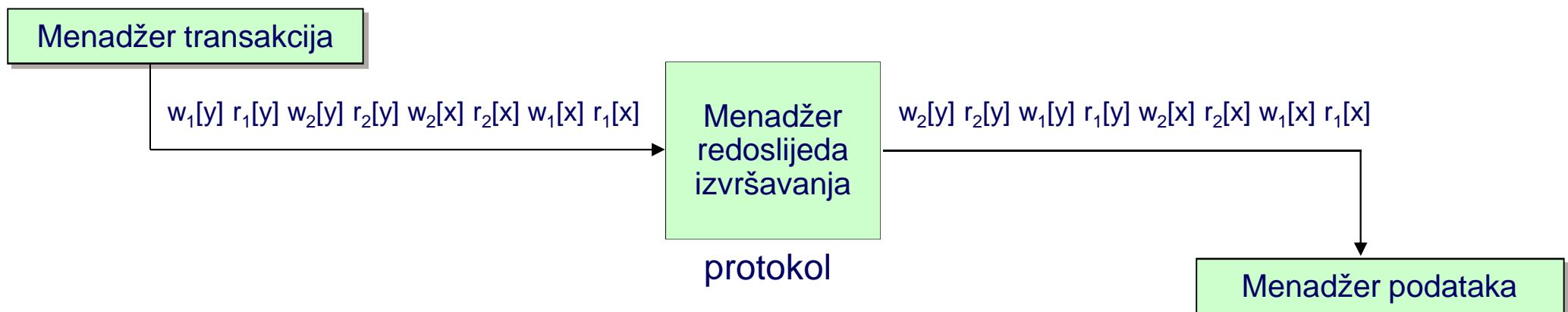


rezultat jednak
rezultatu koji bi se
dobio serijskim
izvršavanjem T_1, T_2

- Niz operacija transakcija nad bazom podataka je serijalizabilan (*Serializable*) ako je rezultat izvršavanja niza operacija ekvivalentan (bilo kojem) serijskom izvršavanju transakcija.

Uloga menadžera redoslijeda izvršavanja (scheduler)

- kako ocijeniti jesu li operacije izvršene nekim konkretnim redoslijedom narušile konzistentnost baze podataka?
 - ima li odgovor na ovo pitanje praktičnu vrijednost?
- bolje pitanje: kako osigurati izvršavanja operacija transakcija u redoslijedu kojim se neće narušiti konzistentnost baze podataka?



- važno uočiti: ne smije se promijeniti redoslijed operacija unutar transakcije. Zašto?

Aritmetička koincidencija

- sustav zadužen za određivanje redoslijeda izvršavanja operacija na raspolaganju ima samo zahtjeve za obavljanje čitanja, pisanja i terminiranja transakcija
- ako zadani niz operacija na bilo koji način može narušiti konzistentnost baze podataka, pretpostavlja se da će taj niz operacija to i učiniti. Drugim riječima, ne uzimaju se u obzir **aritmetičke koincidencije**.

Aritmetička koincidencija

Primjer:

- dva objekta u bazi podataka ($x = 14$, $y = 14$); integritetsko ograničenje: $x = y$

T_1	T_2	x	y
		14	14
read(x , p)			
$p \leftarrow p \% 6$			
write(x , p)		2	
read(y , r)			
$r \leftarrow r \% 6$			
write(y , r)			
	read(x , p)		
	$p \leftarrow p * 2$		
	write(x , p)	2	
	read(y , r)		
	$r \leftarrow r * 2$		
	write(y , r)		4

T_1	T_2	x	y
		14	14
read(x , p)			
$p \leftarrow p \% 6$			
write(x , p)		2	
	read(x , p)		
	$p \leftarrow p * 2$		
	write(x , p)	4	
	read(y , r)		
	$r \leftarrow r \% 6$		
	write(y , r)	28	4

- bi li redoslijed izvršavanja bio "ispravan" kad bi početne vrijednosti bile $x = 5$, $y = 5$

Karakteristični problemi istodobnog pristupa

Nekonzistentna analiza (*inconsistent analysis*)

- ako transakcija T_1 čita nekoliko elemenata, neke elemente pročita prije, a neke nakon što ih je promijenila transakcija T_2

Primjer:

T_1	T_2	x	y
		25	25
read(x, p1)	write(x, 10)	10	
	write(y, 10)		10
read(y, p2)			
$p \leftarrow p_1 + p_2$			
display (p)			

 ←

$r_1[x] \rightarrow r_1[y] \rightarrow c_1$
↓
 $w_2[x] \rightarrow w_2[y] \rightarrow c_2$

- kad bi se transakcije T_1 i T_2 izvršavale serijski, transakcija T_1 bi kao rezultat morala prikazati ili 50 ili 20, ovisno o redoslijedu izvršavanja transakcija
- slične prirode je i problem neponovljivog čitanja (*nonrepeatable read*), o kojem će biti više riječi kasnije

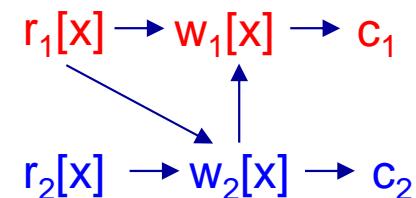
Karakteristični problemi istodobnog pristupa

Izgubljena izmjena (*lost update*)

- ako transakcija T_1 na temelju pročitane vrijednosti elementa x promijeni vrijednost elementa x , pri tome ne uzevši u obzir izmjenu elementa x koju je u međuvremenu obavila transakcija T_2

Primjer:

T_1	T_2	x
		25
read(x , p) $p \leftarrow p + 10$	read(x , p) $p \leftarrow p + 20$ write(x , p)	45
	write(x , p)	35



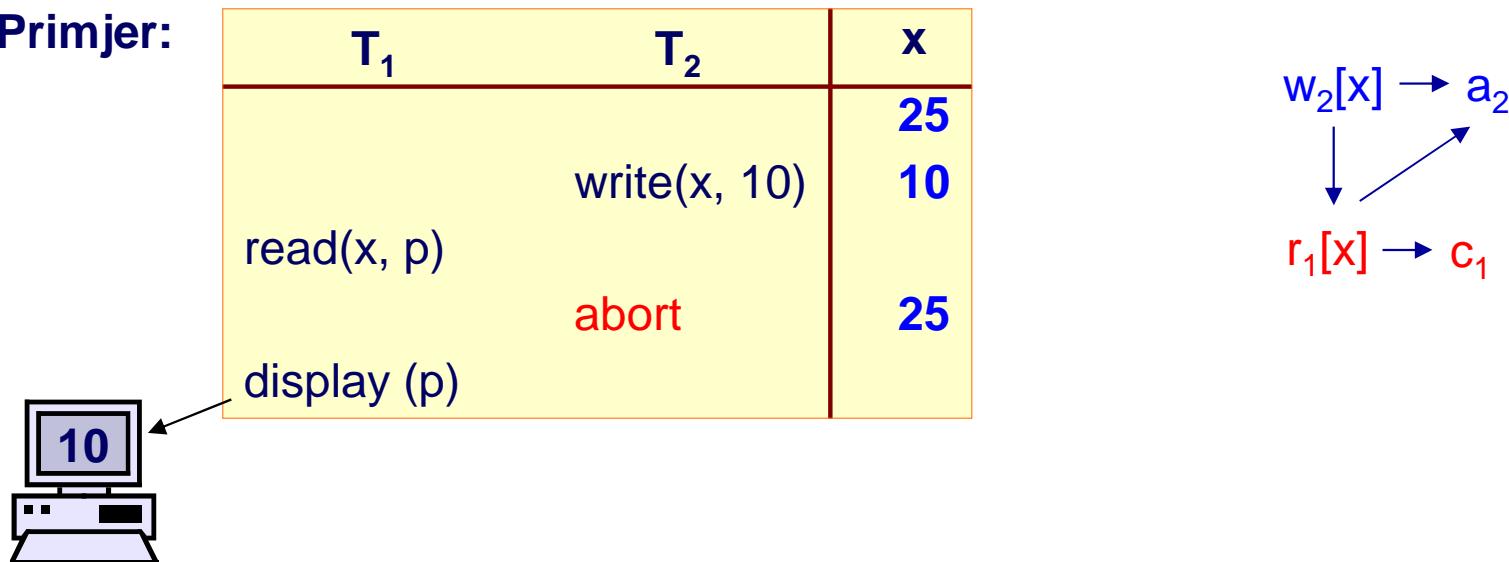
- kad bi se transakcije T_1 i T_2 izvršavale serijski, vrijednost elementa x bi morala biti 55 (bez obzira na redoslijed izvršavanja transakcija). U ovom slučaju izmjena koju je obavila transakcija T_2 je izgubljena (kao da T_2 uopće nije bila izvršena)

Karakteristični problemi istodobnog pristupa

Prljavo čitanje (*dirty read*)

- ako transakcija T_1 pročita element nakon što ga je promijenila transakcija T_2 , a prije nego je T_2 potvrđena

Primjer:



- kad bi se transakcije izvršavale serijski, rezultat transakcije T_1 bi morao biti 25
- transakcija T_1 je pročitala "prljavu" vrijednost elementa x, vrijednost koju element x "nikad nije imao" (jer poništена transakcija niti u jednom trenutku nije smjela imati nikakav vidljiv utjecaj na stanje baze podataka)

Teorija serijalizabilnosti

Teorija serijalizabilnosti (*serializability theory*)

- teorija serijalizabilnosti je matematički alat kojim se može ispitati ispravnost protokola koji upravljaju redoslijedom izvršavanja operacija transakcija
 - protokol mora osigurati *serijalizabilno* izvršavanje transakcija: omogućiti istodobno izvršavanje operacija transakcija, ali tako da je rezultat izvršavanja jednak rezultatu serijskog izvršavanja transakcija

Povijest (*history, schedule*) - neformalno

- modelom transakcije određen je redoslijed operacija unutar jedne transakcije
- *povijest* modelira redoslijed operacija pri istodobnom izvršavanju više transakcija

Primjer:

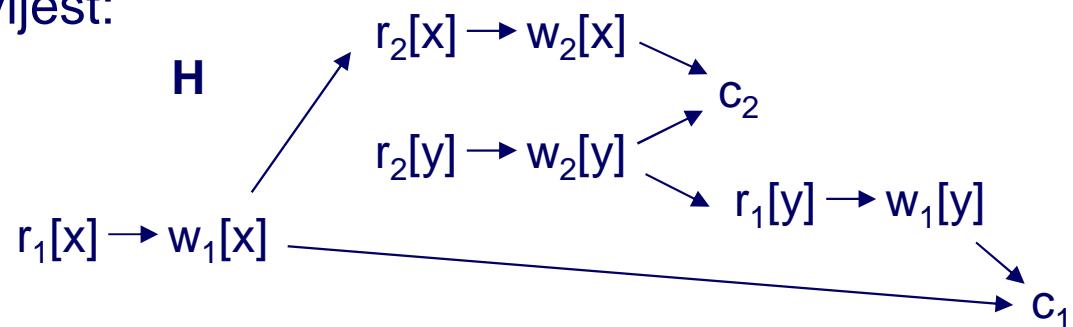
T_1
read(x, p)
 $p \leftarrow p + 100$
write(x, p)
read(y, r)
 $r \leftarrow r + 100$
write(y, r)

T_1
 $r_1[x] \rightarrow w_1[x]$
 $r_1[y] \rightarrow w_1[y]$

T_2
read(x, p)
 $p \leftarrow p * 2$
write(x, p)
read(y, p)
 $r \leftarrow r * 2$
write(y, r)

T_2
 $r_2[x] \rightarrow w_2[x]$
 $r_2[y] \rightarrow w_2[y]$

Povijest:



- kao oznaka za povijest koristi se slovo H

Topološki poredak za H (jedan od mogućih):

$r_1[x], w_1[x], r_2[x], w_2[x], r_2[y], w_2[y], c_2, r_1[y], w_1[y], c_1$

Konfliktne operacije

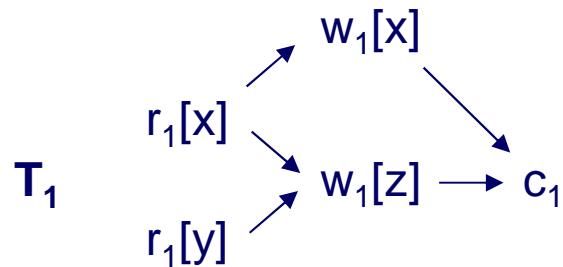
- konfliktne operacije su operacije čija bi zamjena redoslijeda izvršavanja mogla dovesti do promjene konačnog rezultata
- operacije $r_i[x]$ i $w_i[x]$ transakcije T_i su konfliktne operacije jer
 - rezultat operacije $r_i[x]$ ovisi o tome hoće li se izvršiti prije ili poslije operacije $w_i[x]$
 - redoslijed ovih operacija je utvrđen semantikom transakcije. To znači da na redoslijed izvršavanja tih operacija sustav ne smije utjecati
- operacije $r_i[x]$ i $w_j[x]$ transakcija T_i i T_j su konfliktne operacije
 - rezultat operacije $r_i[x]$ ovisi o tome hoće li se izvršiti prije ili poslije operacije $w_j[x]$
- operacije $w_i[x]$ i $w_j[x]$ su konfliktne operacije
 - konačna vrijednost elementa x ovisi o tome koja je operacija pisanja izvršena posljednja

Zaključak:

- dvije operacije su međusobno konfliktne ako djeluju na isti element baze podataka i barem jedna od tih operacija je operacija pisanja

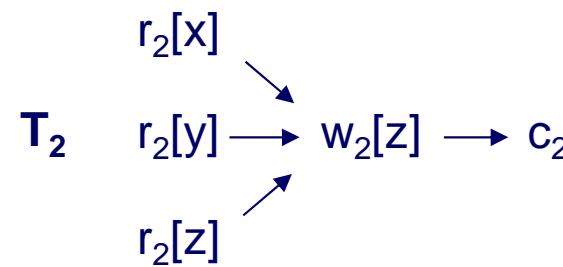
Konfliktne operacije

Primjer: ■ utvrditi konfliktne operacije unutar i među transakcijama T_1 i T_2



■ konfliktne operacije unutar transakcije T_1

$r_1[x], w_1[x]$



■ konfliktne operacije unutar transakcije T_2

$r_2[z], w_2[z]$

■ konfliktne operacije među operacijama transakcija T_1 i T_2

$w_1[x], r_2[x]$

$w_1[z], r_2[z]$

$w_1[z], w_2[z]$

Povijest

- povijest je temeljni koncept u teoriji serijalizabilnosti
- opisuje vremenski određen redoslijed obavljanja operacija jedne ili više transakcija

Definicija:

- Neka je $T = \{ T_1, T_2, \dots, T_n \}$ skup transakcija, $T_i = (\Sigma_i, <_i)$, $1 \leq i \leq n$. Povijest (*history*) H je parcijalni poredak $(\Sigma_H, <_H)$ pri čemu vrijedi:

$$1. \quad \Sigma_H = \bigcup_{i=1}^n \Sigma_i$$

$$2. \quad <_H \supseteq \bigcup_{i=1}^n <_i$$

3. za svake dvije konfliktne operacije $p, q \in \Sigma_H$, vrijedi ili $p <_H q$ ili $q <_H p$

Povijest

Objašnjenje definicije:

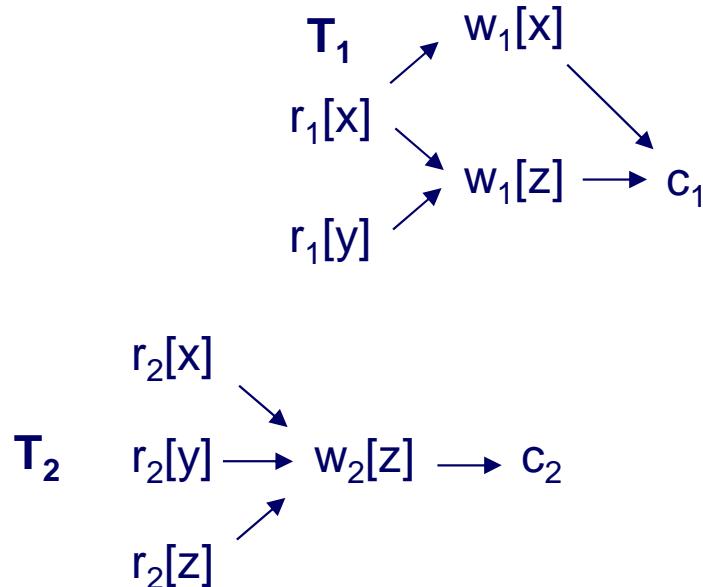
- svojstvo 1. propisuje da povijest sadrži točno one operacije koje su sadržane u transakcijama iz skupa T
- svojstvo 2. određuje da poredak kojeg operacije imaju unutar svake transakcije T_i , mora biti zadržan i u H
- svojstvo 3. propisuje da redoslijed izvršavanja operacija bude određen ne samo za konfliktne operacije unutar transakcija (za svaku transakciju T_i to je određeno relacijom $<_i$), već da je poredak određen za sve konfliktne operacije neovisno od toga kojoj transakciji pripadaju

Napomena:

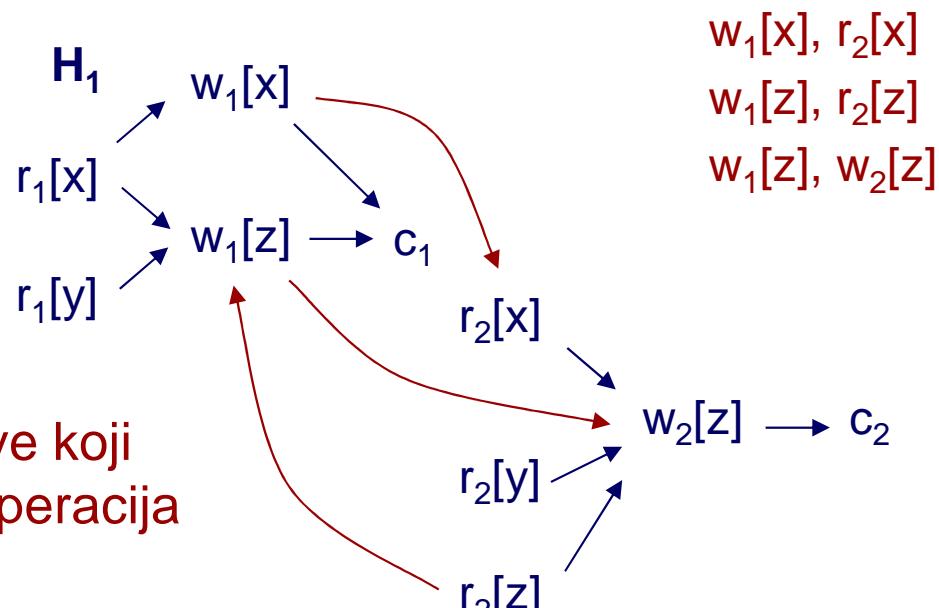
U [Bernstein] se također koriste pojmovi kompletna povijest (*complete history*), povijest (*history*) i potvrđena projekcija povijesti (*committed projection of history*). Navedeni pojmovi su važni za razmatranje slučajeva nepotpunih izvršavanja pojedinih transakcija iz povijesti, ali ovdje se takva razmatranja neće provoditi.

Povijest

- Primjer:**
- nacrtati povijest H_1 (jednu od mogućih) koja obuhvaća transakcije T_1 i T_2
 - ispisati topološki poredak (jedan od mogućih) za povijest H_1



- grafove za T_1 i T_2 precrati bez promjene
- utvrditi konfliktne operacije među operacijama transakcija T_1 i T_2



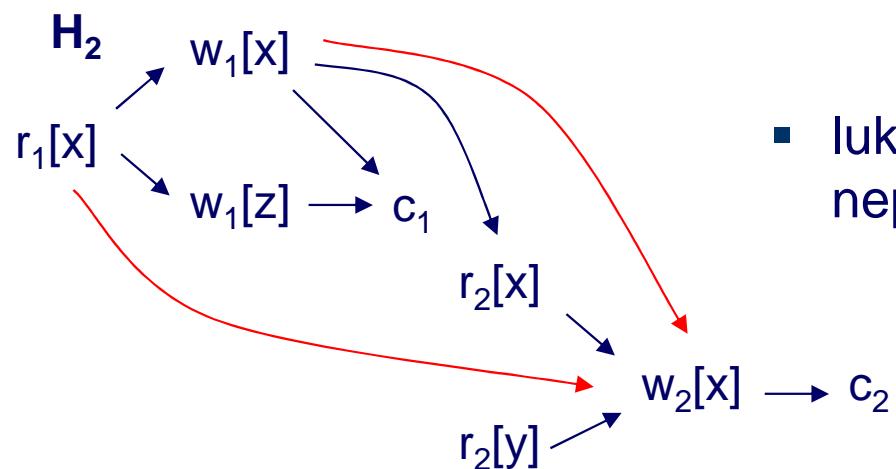
- u graf H_1 ucrtati lúkove koji određuju redoslijed operacija među konfliktnim operacijama transakcija

$r_1[x], r_2[z], r_2[y], r_1[y], w_1[x], w_1[z], c_1, r_2[x], w_2[z], c_2$

Povijest

- konfliktne operacije nije uvijek nužno direktno povezati lukovima: ako se poredak dviju konfliktnih operacija u povijesti može utvrditi preko tranzitivnosti, luk među tim operacijama nije nužno ucrtati

Primjer:



- lukovi označeni crvenom bojom su nepotrebni

Zadatak za vježbu:

- nacrtati povijest koja odgovara prikazanom nizu operacija transakcija T_1 i T_2
 - $r_1[x], w_1[x], r_2[x], w_2[x], r_2[y], w_2[y], c_2, r_1[y], w_1[y], c_1$
 - $r_1[x], w_1[x], r_2[x], w_2[x], r_1[y], w_1[y], c_1, r_2[y], w_2[y], c_2$

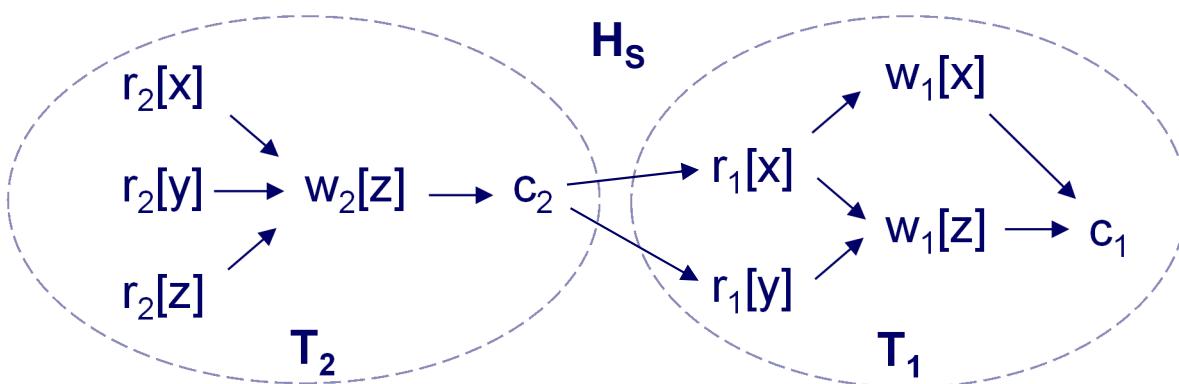
Serijska povijest

- povijest u kojoj se transakcije izvršavaju serijski (jedna iza druge) interesantna je kao polazni koncept jer niz korektnih transakcija koje se izvršavaju u međusobnoj izolaciji sigurno neće narušiti konzistentnost baze podataka

Definicija:

- Povijest $H_S = (\Sigma_S, \prec_S)$ je serijska povijest (*serial history*) ako za svake dvije transakcije $T_i, T_j \in H_S$, $T_i = (\Sigma_i, \prec_i)$, $T_j = (\Sigma_j, \prec_j)$ vrijedi da sve operacije iz Σ_i prethode svim operacijama iz Σ_j ili sve operacije iz Σ_j prethode svim operacijama iz Σ_i .

Primjer:



- slika prikazuje serijsku povijest H_S za transakcije T_2 i T_1 iz povijesti H_1
- za vježbu nacrtati serijsku povijest u kojoj operacije iz T_1 prethode svim operacijama iz T_2

Redoslijed transakcija u serijskoj povijesti

- definicijom serijske povijesti ne propisuje se absolutni redoslijed izvršavanja transakcija - svaka povijest u kojoj se transakcije izvršavaju jedna iza druge, neovisno o redoslijedu, smatra se ispravnom
- ipak, jasno je da redoslijed obavljanja transakcija može utjecati na rezultat

Redoslijed transakcija u serijskoj povijesti

Primjer:

T_1 : uplata na račun (*iznos, brRačun*)
uvećaj stanje računa

T_2 : isplata s računa (*iznos, brRačun*)
umanji stanje računa
ako je stanje računa < -1000
evidentiraj nedopušteni ulazak u minus

- rezultat će biti različit ako se za račun broj 12345, sa stanjem = -500kn obavi
 - $T_1(200, 12345)$, $T_2(600, 12345)$ u odnosu na
 - $T_2(600, 12345)$, $T_1(200, 12345)$
 - međutim, obje povijesti se smatraju korektnima
- ako postoji potreba za izvršavanjem transakcija T_1 i T_2 točno određenim redoslijedom, tada je odgovornost korisnika osigurati redoslijed na sljedeći način:
 - predati na izvršavanje prvu transakciju i pričekati potvrdu (*committed*)
 - predati na izvršavanje drugu transakciju

Serijalizabilna povijest

- u stvarnosti, operacije različitih transakcija se izvršavaju naizmjence (*interleaved*)
- postoje li povijesti koje nisu serijske, a imaju isti učinak (rezultate koje prezentiraju korisniku i novo stanje baze podataka) kao neka (bilo koja) serijska povijest istih transakcija?

Definicija:

- Povijest H je ekvivalentna povijesti H' , $H \equiv H'$, ako i samo ako je rezultat izvršavanja povijesti H jednak rezultatu izvršavanja povijesti H'

Definicija:

- Povijest H je serijalizabilna (SR) ako je ekvivalentna nekoj (bilo kojoj) serijskoj povijesti H_S , odnosno $H \equiv H_S$

Serijalizabilna povijest

- izvršavanjem serijalizabilne povijesti ne bi se narušila konzistentnost baze podataka, a omogućilo bi se *istodobno* izvršavanje operacija
 - problem: kako "usporedbom rezultata" ispitati ekvivalentnost s nekom serijskom poviješću? Dvije (loše) mogućnosti:
 1. za neku već izvršenu povijest H utvrditi je li ona serijalizabilna, odnosno vrijedi li $H \equiv H_S$. Uspoređivati rezultat povijesti H s rezultatima svih mogućih serijskih povijesti H_S dok se ne pronađe (ili se ne pronađe) serijska povijest koja daje jednake rezultate kao H
 - teško! Što uostalom učiniti ako se utvrdi da H nije serijalizabilna? Poništiti sve transakcije pa pokušati ponovo? A izdržljivost transakcije?
 2. izvršavati operacije iz H tek kad se utvrdi hoće li povijest H dati jednake rezultate kao neka povijest H_S
 - nemoguće! Povijest nije unaprijed poznata, redoslijed obavljanja operacija ovisi o opterećenju sustava, redoslijedu pristizanja operacija, interakciji s korisnikom itd.
- ⇒ ekvivalentnost povijesti temeljena na "jednakom rezultatu" nije prikladan koncept

Konflikt - ekvivalentna povijest

- koristit će se stroži uvjet ekvivalentnosti, konflikt-ekvivalentnost
 - stroži: neke povijesti koje jesu ekvivalentne u smislu prethodne definicije neće biti prepoznate kao ekvivalentne i prema sljedećoj definiciji

Definicija:

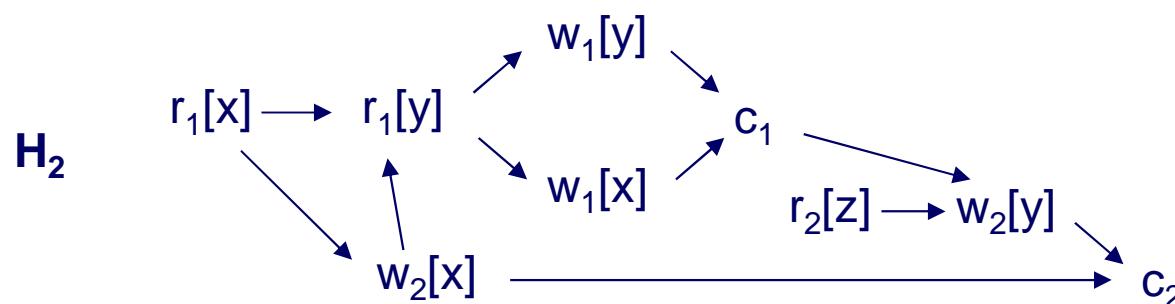
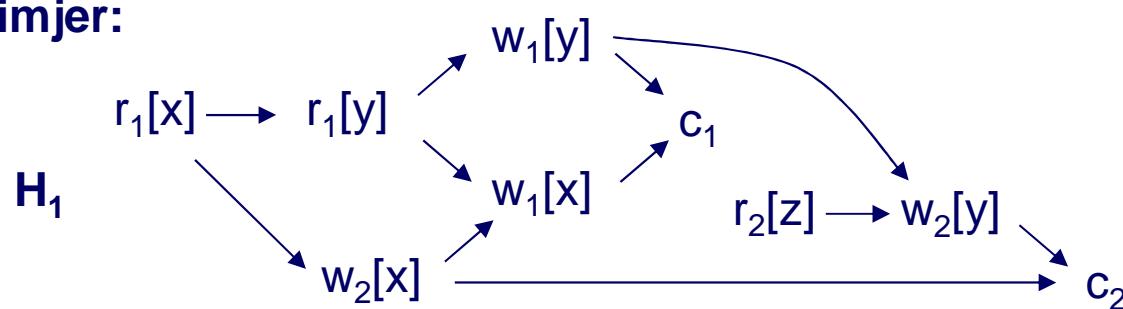
- Dvije povijesti $H = (\Sigma, <)$ i $H' = (\Sigma', <')$ su konflikt-ekvivalentne (*conflict-equivalent*), $H \equiv_C H'$, ako vrijedi:
 1. $\Sigma = \Sigma'$, tj. H i H' su definirane nad istim skupovima operacija
 2. za svaki par konfliktnih operacija p_i i q_j koje pripadaju transakcijama $T_i, T_j \in \Sigma$ vrijedi da ako je $p_i <_H q_j$, tada $p_i <_{H'} q_j$.

Objašnjenje:

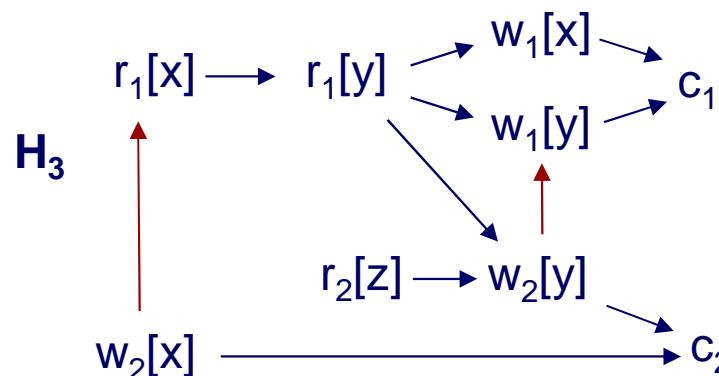
- svojstvo 1. propisuje da povijesti H i H' sadrže iste operacije
- svojstvo 2. propisuje da je svaki par konfliktnih operacija dviju istih transakcija poredan na jednak način u obje povijesti, H i H'
- rezultat obavljanja dviju povijesti koje su ekvivalentne prema ovoj definiciji sigurno je jednak jer je redoslijed operacija kod svakog para operacija, čija bi promjena redoslijeda mogla utjecati na konačni rezultat, jednak u obje povijesti

Konflikt - ekvivalentna povijest

Primjer:



Provjeriti za vježbu:



a) $H_1 \equiv_C H_2$

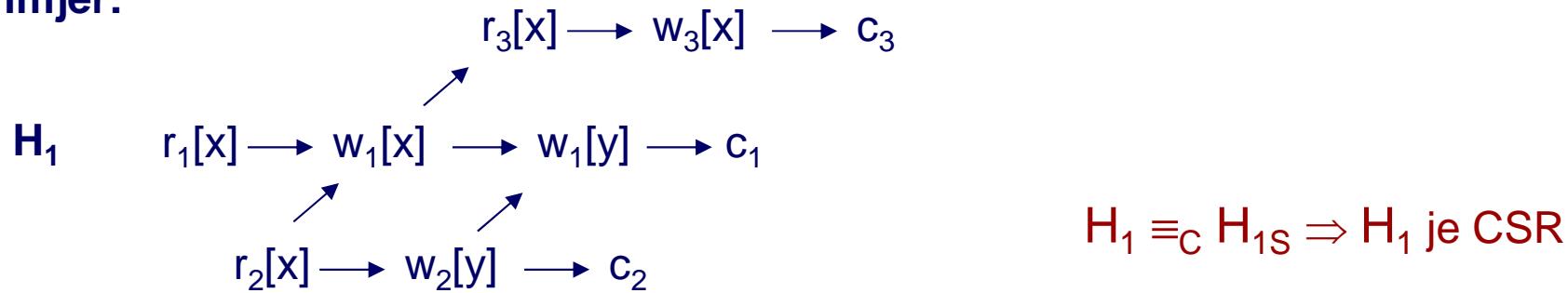
b) $\neg (H_3 \equiv_C H_1)$

Konflikt - serijalizabilna povijest

Definicija:

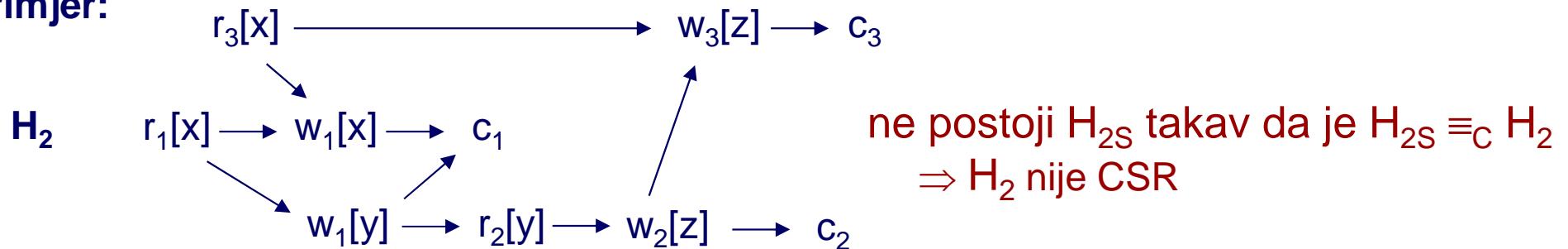
- Povijest H je konflikt-serijalizabilna (CSR) ako je H konflikt-ekvivalentna nekoj serijskoj povijesti H_s

Primjer:



$$H_1 s \rightarrow r_2[x] \rightarrow w_2[y] \rightarrow c_2 \rightarrow r_1[x] \rightarrow w_1[x] \rightarrow w_1[y] \rightarrow c_1 \rightarrow r_3[x] \rightarrow w_3[x] \rightarrow c_3$$

Primjer:



Serijalizacijski graf

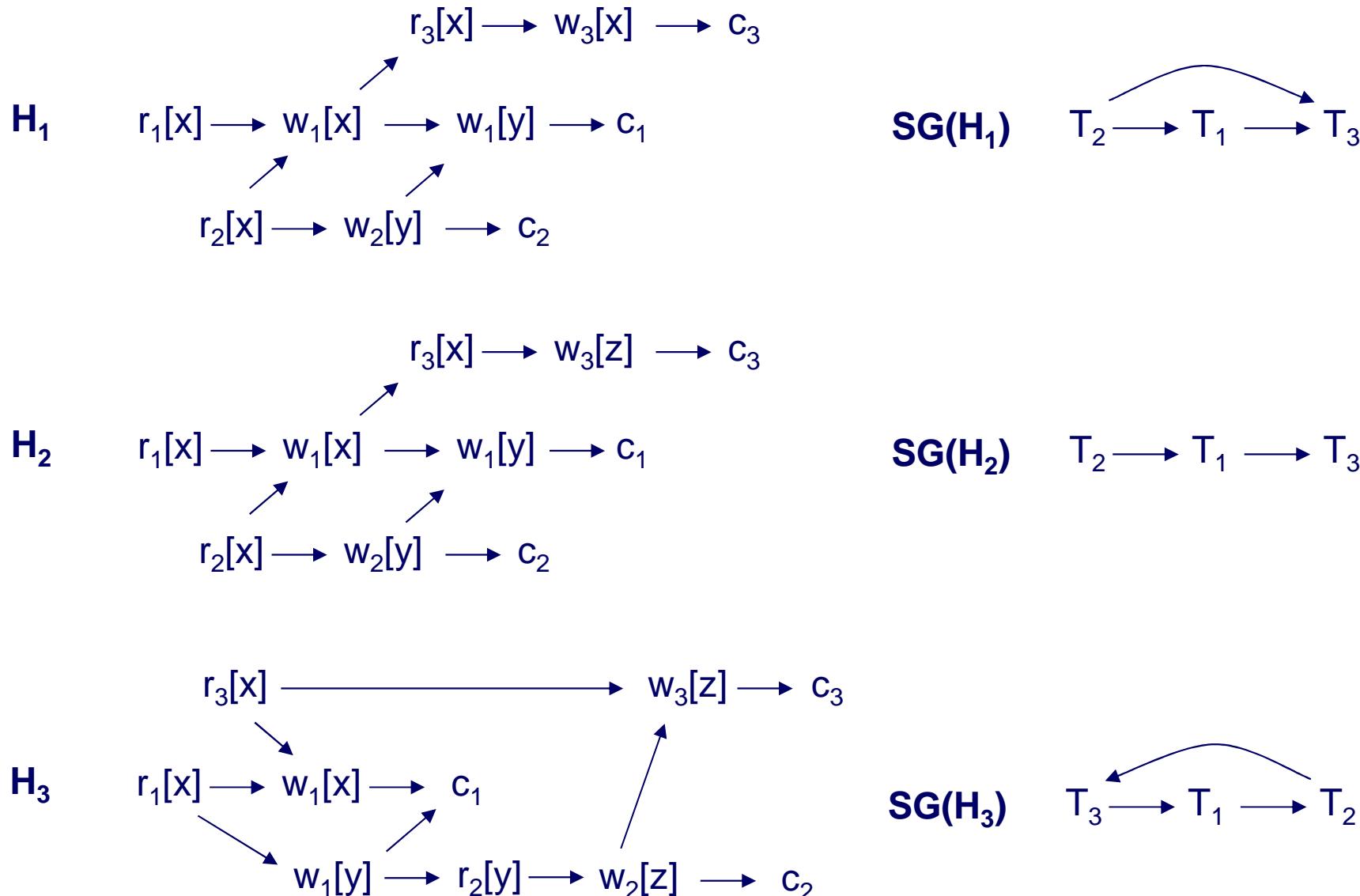
- serijalizacijski graf (*serialization graph*), uz teorem o serijalizabilnosti, predstavlja koristan alat kojim se olakšava ispitivanje svojstva konflikt-serijalizabilnosti bilo koje povijesti H
- u odnosu na graf povijesti, u serijalizacijskom grafu su zakriveni detalji (pojedine operacije) transakcija, a naznačeni su samo oni odnosi među transakcijama koji su važni za utvrđivanje konflikt-serijalizabilnosti

Definicija:

- Serijalizacijski graf (SG) za povijest H , označen sa $SG(H)$, je usmjereni graf čiji su čvorovi transakcije iz H , a lúkovi su (T_i, T_j) , $i \neq j$, ako postoji operacija p_i iz T_i koja prethodi operaciji q_j iz T_j i pri tome su p_i i q_j konfliktne operacije.

Serijalizacijski graf

Primjer:



Teorem o serijalizabilnosti

Sljedeće razmatranje nije dokaz*, ali predstavlja intuitivnu osnovu za fundamentalni teorem teorije serijalizabilnosti:

- ako u grafu $SG(H)$ postoji put među čvorovima T_i i T_j tada u H barem jedna operacija iz T_i prethodi barem jednoj operaciji iz T_j
- ako je H konflikt-serijalizabilna, tada za H postoji konflikt-ekvivalentna serijska povijest H_S , u kojoj sve operacije iz T_i moraju prethoditi svim operacijama iz T_j
- ako u $SG(H)$ postoji put od T_j prema T_i , tada u H sigurno postoji operacija transakcije T_j koja prethodi nekoj operaciji transakcije T_i , što znači da H nije konflikt-ekvivalentna s H_S

* dokaz teorema se može naći u [Bernstein], str. 33.

Teorem:

- Povijest H je konflikt-serijalizabilna ako i samo ako je $SG(H)$ aciklički graf

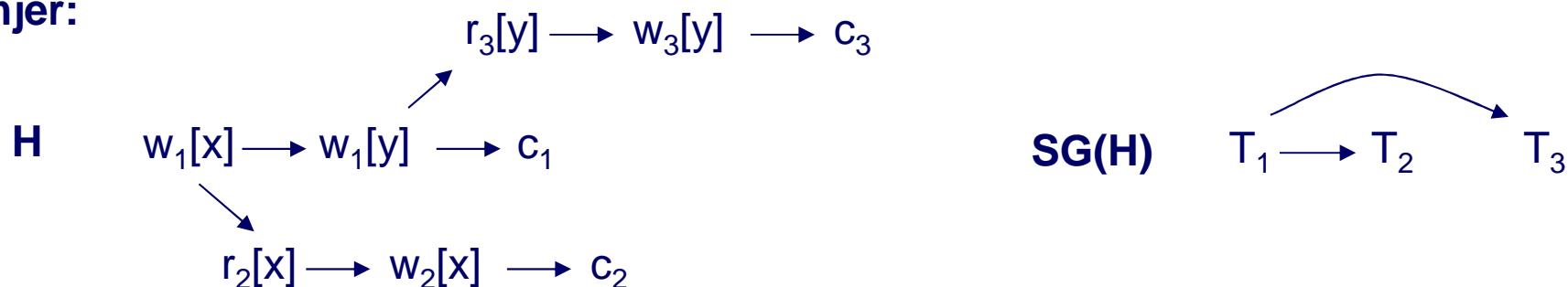
Primjer:

- H_1 i H_2 iz prethodnog primjera su konflikt-serijalizabilne povijesti
- H_3 iz prethodnog primjera nije konflikt-serijalizabilna povijest

Određivanje konflikt - ekvivalentne H_S na temelju $SG(H)$

- pomoću $SG(H)$ se za povijest H lako može odrediti konflikt-ekvivalentna serijska povijest (zapravo, sve moguće konflikt-ekvivalentne serijske povijesti)

Primjer:



- budući da je $SG(H)$ acikličan, moguće je odrediti topološki poredak. Bilo koji topološki poredak elemenata $SG(H)$ predstavlja konflikt-ekvivalentnu serijsku povijest za H

$$H_{1S} \equiv_C H$$

$$T_1, T_2, T_3 \quad w_1[x] \rightarrow w_1[y] \rightarrow c_1 \rightarrow r_2[x] \rightarrow w_2[x] \rightarrow c_2 \rightarrow r_3[y] \rightarrow w_3[y] \rightarrow c_3$$

$$H_{2S} \equiv_C H$$

$$T_1, T_3, T_2 \quad w_1[x] \rightarrow w_1[y] \rightarrow c_1 \rightarrow r_3[y] \rightarrow w_3[y] \rightarrow c_3 \rightarrow r_2[x] \rightarrow w_2[x] \rightarrow c_2$$

Pogled - serijalizabilnost (*view - serializability*)

Definicija:

- Povijesti H i H' su pogled-ekvivalentne (*view-equivalent*), $H \equiv_v H'$, ako vrijedi:
 1. za svaki objekt x , ako transakcija T_i u povijesti H čita inicijalnu vrijednost objekta x , tada inicijalnu vrijednost od x mora pročitati i u povijesti H'
 2. za svaki objekt x , ako transakcija T_i u povijesti H čita vrijednost objekta x koja je nastala obavljanjem operacije $w_j[x]$ transakcije T_j , tada transakcija T_i u povijesti H' također mora pročitati vrijednost x koja je nastala obavljanjem iste operacije $w_j[x]$ transakcije T_j
 3. za svaki objekt x , transakcija T_i koja obavlja završnu operaciju pisanja u povijesti H , obavlja završnu operaciju pisanja i u povijesti H'

Objašnjenje definicije:

- svojstva 1. i 2. osiguravaju da svaka transakcija čita iste vrijednosti u obje povijesti, te se time osigurava ekvivalentno izračunavanje vrijednosti koje će eventualno biti zapisane ili prezentirane korisniku
- svojstvo 3. osigurava da će konačne vrijednosti koje će biti zapisane temeljem obje povijesti biti jednake

Pogled - serijalizabilnost

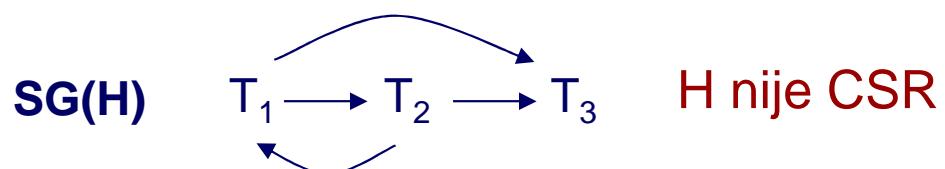
Definicija:

- Povijest H je pogled-serijalizabilna (VSR) ako je H pogled-ekvivalentna nekoj serijskoj povijesti H_S

Primjer: $H \quad r_1[x], w_1[x], w_2[x], w_2[y], c_2, w_1[y], r_3[x], w_3[x], w_3[y], c_3, w_1[z], c_1$
 $H_S \quad r_1[x], w_1[x], w_1[y], w_1[z], c_1, w_2[x], w_2[y], c_2, r_3[x], w_3[x], w_3[y], c_3$

$$H \equiv_V H_S \Rightarrow H \text{ je VSR}$$

Primjer: $SG(H)$ za povijest H iz prethodnog primjera



Pogled - serijalizabilnost

- moguće je dokazati da je svaka CSR povijest ujedno i VSR povijest, dok svaka VSR povijest nije ujedno i CSR povijest
- razlog zašto se pogled-serijalizabilnost, iako manje restriktivan kriterij, ne koristi kao kriterij za utvrđivanje serijalizabilnosti povijesti je taj što za ispitivanje pogled-ekvivalentnosti ne postoji efikasan algoritam (radi se o NP-kompletном problemu)



Serijalizabilnost i problemi vezani uz obnovu

- do sada se problem istodobnog pristupa promatrao odvojeno od problema obnove
- kriterij serijalizabilnosti je dovoljan samo onda kada se koristi sustav u kojem ne postoje pogreške (transakcija, sustava, medija)

Definicija:

- Transakcija T_j čita iz transakcije T_i ako postoji element x za kojeg vrijedi:
 1. T_j obavlja $r_j[x]$ nakon što je T_i obavila $w_i[x]$
 2. T_i nije poništena prije nego je obavljena operacija $r_j[x]$
 3. svaka druga transakcija T_k koja je eventualno obavila operaciju $w_k[x]$ u vremenu između operacije $w_i[x]$ i $r_j[x]$ je poništena prije obavljanja operacije $r_j[x]$

Serijalizabilnost i problemi vezani uz obnovu

Primjer: $H \quad w_1[x], w_3[x], a_3, r_2[x], w_2[y], c_2, a_1$

- H je CSR
- transakcija T_2 čita iz transakcije T_1 (uočiti: ne čita iz T_3 , jer T_3 je poništena prije nego je T_2 obavila operaciju $r_2[x]$)
- nakon poništavanja T_1 postalo je jasno da je T_2 pročitala pogrešnu vrijednost elementa x . Trebalo bi poništiti i već potvrđenu T_2 (narušeno je svojstvo izdržljivosti iz ACID)
- H nije obnovljiva (*recoverable*)

Napomena:

- H ne bi bila obnovljiva niti onda kada bi se operacija a_1 zamijenila operacijom c_1 , jer se uvijek mora računati na mogućnost pojave pogreške koja će uzrokovati poništavanje transakcije T_1

$H \quad w_1[x], w_3[x], a_3, r_2[x], w_2[y], c_2, c_1$

Obnovljiva povijest (RC)

Definicija:

- povijest H je obnovljiva (*recoverable* ili RC) ako za svaku potvrđenu transakciju T_i koja čita iz T_j , vrijedi da je $c_j < c_i$.
 - drugim riječima, transakcija smije biti potvrđena tek nakon što su potvrđene sve transakcije iz kojih je čitala

Primjer: H_1 $w_1[x], r_2[x], w_2[y], c_1, c_2$

- ako SUBP umjesto c_1 obavi a_1 , smije li se transakcija T_2 potvrditi? Ne!
- izdržljivost transakcije nije narušena, ali SUBP će morati poništiti T_2 samo zato jer je čitala iz poništene transakcije T_1
- H_1 jest obnovljiva, ali ne izbjegava kaskadno poništavanje (*avoids cascading aborts*)

Primjer: H_2 $w_1[x], r_2[x], w_2[y], r_3[y], w_3[z], c_1, c_2, c_3$

- ako c_1 ne uspije, T_2 se mora poništiti jer je čitala iz poništene transakcije T_1
- tada se i T_3 mora poništiti jer je čitala iz poništene transakcije T_2

Povijest koja izbjegava kaskadno poništavanje (ACA)

Definicija:

- povijest H izbjegava kaskadno poništavanje (*avoids cascading aborts* ili ACA) ako za svaku transakciju T_i koja operacijom $r_i[x]$ čita iz T_j , vrijedi da je $c_j < r_i[x]$.
 - drugim riječima, transakcija smije čitati samo one vrijednosti koje je zapisala već potvrđena transakcija

Primjer: $H = r_1[x], w_1[x], w_2[x], a_1, r_2[x], c_2, r_3[x], w_3[x], c_3$

- H je ACA jer:
 - T_2 ne čita iz T_1
 - T_3 čita iz T_2 , ali je $c_2 < r_3[x]$

Problem: log $\langle \text{start } T_1 \rangle \langle T_1, x, 1, 3 \rangle \langle \text{start } T_2 \rangle \langle T_2, x, 3, 6 \rangle \langle \text{abort } T_1 \rangle \langle \text{commit } T_2 \rangle \dots$

- poništavanje operacije $w_1[x]$ zbog a_1 : obavlja se $undo(x, 1)$
- izgubljen je rezultat operacije $w_2[x]$: vrijednost elementa x je 1, a trebala je biti 6
- povijest **nije striktna**

Striktna povijest (ST)

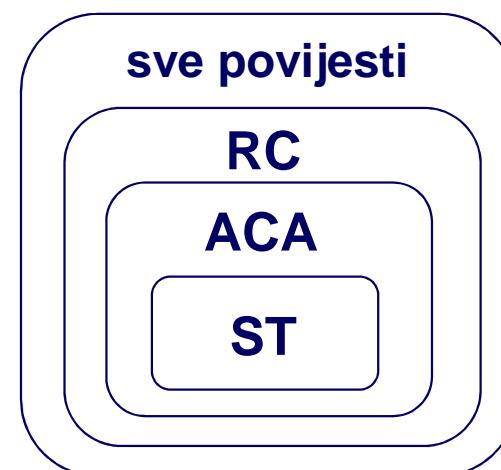
Definicija:

- povijest H je striktna (*strict* ili ST) ako za svake dvije transakcije T_i i T_j vrijedi da ako je $w_j[x] < o_i[x]$, tada ili $a_j < o_i[x]$ ili $c_j < o_i[x]$
 - drugim riječima, niti jedan element se ne smije niti pročitati niti izmijeniti dok sve transakcije koje su prije toga pisale u taj element ne terminiraju (obave *abort* ili *commit*)

Primjer: $H = r_1[z], w_1[x], r_2[z], w_2[y], r_3[z], a_1, r_3[x], w_3[x], w_2[z], c_2, w_3[y], c_3$

- H je ST

- svaka ST povijest jest ujedno i ACA povijest
- svaka ACA povijest jest ujedno i RC povijest



Primjer:

- za svaku od sljedećih povijesti (prikazanih u obliku topološkog poretka) odrediti jesu li RC, ACA, ST

H_1 $r_1[x], w_1[y], w_2[y], c_1, c_2$

H_2 $r_1[x], w_1[y], r_2[y], w_2[y], c_2, c_1$

H_3 $r_1[x], w_1[y], r_2[y], w_2[y], c_1, c_2$

H_4 $r_1[x], w_1[y], w_2[x], c_1, w_2[y], c_2$

- H_1 : RC, ACA, nije ST
- H_2 : nije RC \rightarrow nije ACA, nije ST
- H_3 : RC, nije ACA \rightarrow nije ST
- H_4 : RC, ACA, ST

Literatura:

- P. Bernstein, V. Hadzilacos, N. Goodman. Concurrency control and recovery in database systems. Addison-Wesley Publishing Company. Reading, Massachusetts. 1987.
- H. Garcia-Molina, J. Ullman, J. Widom. Database system implementation. Prentice-Hall International, Inc. Upper Saddle River, New Jersey. 2000.
- A. Silberschatz, H. Korth, S. Sudarshan. Database system concepts, 4th ed. McGraw-Hill. 2002.
- R. Elmasri, S. Navathe. Fundamentals of database systems, 3rd ed. Addison-Wesley Publishing Company. Reading, Massachusetts. 2000.

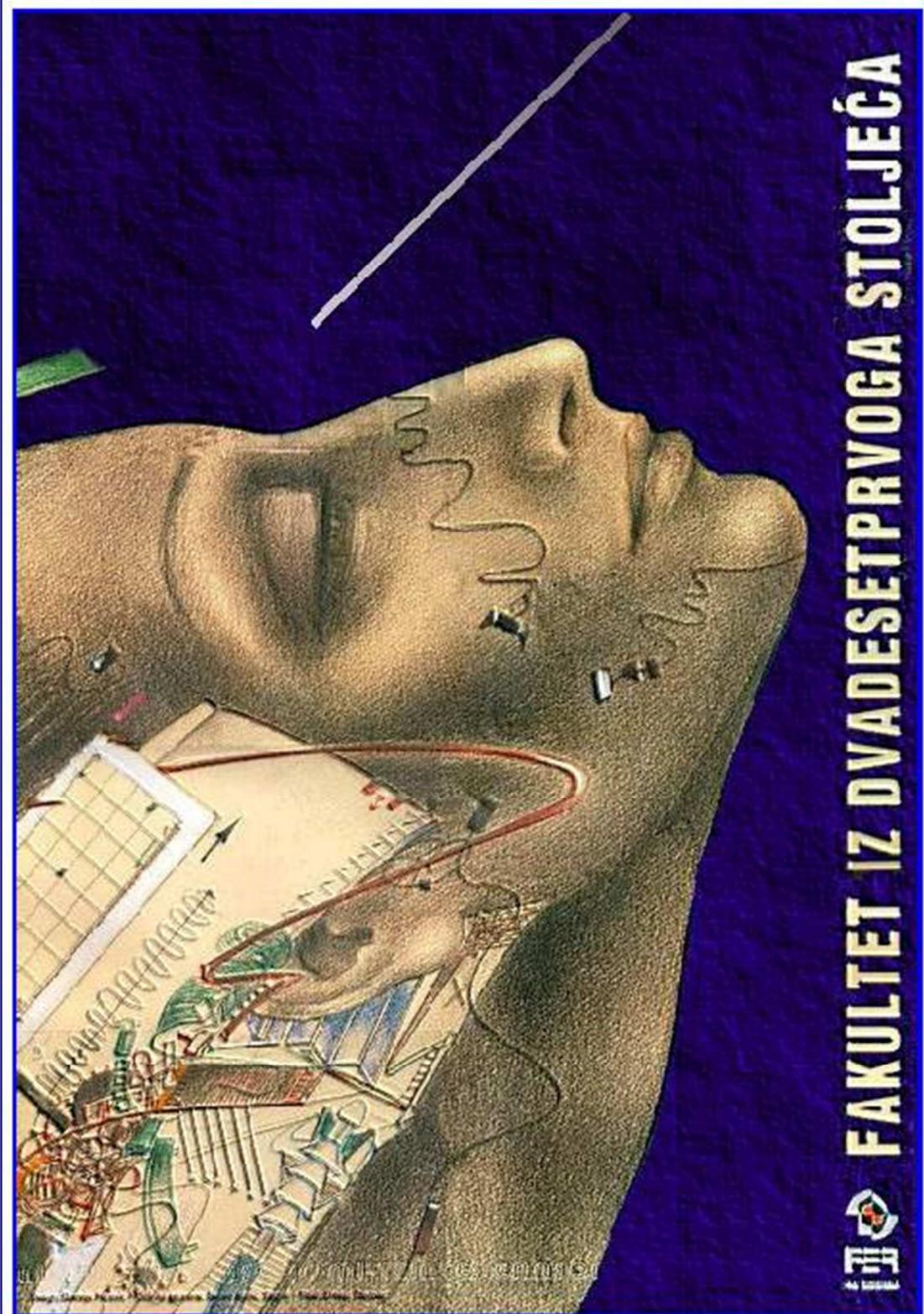
Sustavi baza podataka

Predavanja

9. Kontrola istodobnog pristupa

(2. dio)

svibanj 2014.

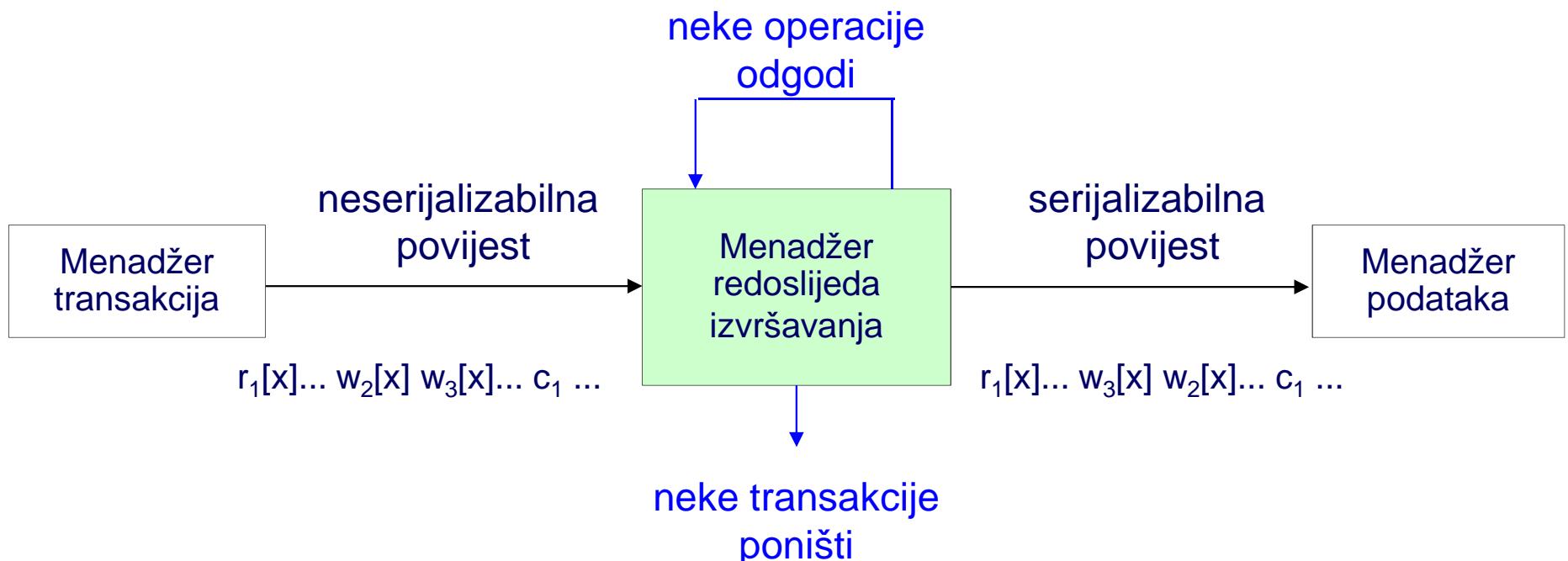


Osiguravanje serijalizabilnog izvršavanja transakcija

- koriste se protokoli za koje je unaprijed dokazano da produciraju serijalizabilne povijesti
- dvofazni protokol zaključavanja (*two-phase locking protocol*) - 2PL protokol
 - temeljni 2PL protokol (*basic 2PL protocol*)
 - striktni 2PL protokol (*strict 2PL protocol*)
 - rigorozni 2PL protokol (*rigorous 2PL protocol*)
 - serijalizabilnost se osigurava odgađanjem obavljanja operacija transakcija
 - moguća pojava potpunog zastoja (zahtijeva poništavanje transakcija)
- protokol vremenskih oznaka (*timestamp ordering protocol*) - TO protokol
 - operacije se ne odgađaju, serijalizabilnost se osigurava poništavanjem transakcija, potpuni zastoj se ne može dogoditi
- multiverzijski protokol (*multiversion protocol*) - MV protokol
- protokol validacije
- protokol temeljen na grafovima

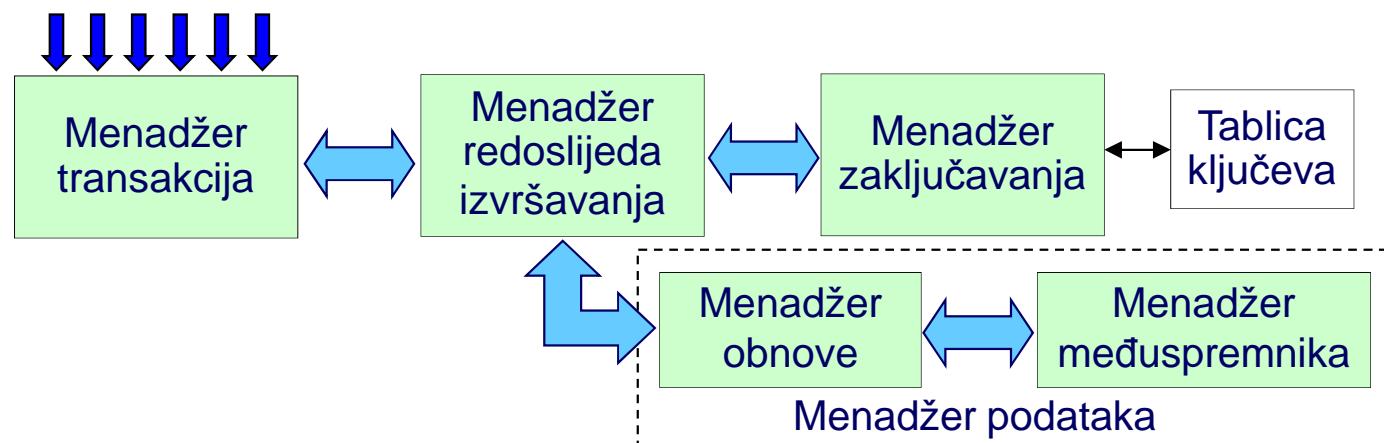
Serijalizabilnost povijesti - uloga menadžera redoslijeda izvršavanja

- u općem slučaju:
 - neke operacije ili neke transakcije se odgađaju: obavljaju se tek u trenutku kada njihovo obavljanje neće uzrokovati neserijalizabilnost
 - neke transakcije se poništavaju
- komponenta sustava zadužena za implementaciju protokola: menadžer redoslijeda izvršavanja (*scheduler*)



Protokoli temeljeni na zaključavanju

- zaključavanje je uobičajen način rješavanja problema sinkroniziranja pristupa dijeljenim podacima
- menadžer zaključavanja (*LM-locking manager*) može ključem (*lock*) zaključati element baze podataka
- prije obavljanja operacije $o_i[x]$ LM ispituje je li x zaključan
 - ako x nije zaključan, LM ga zaključava u korist T_i - operacija $o_i[x]$ se tek tada može obaviti
 - ako je x zaključan (LM ga je zaključao nekompatibilnim ključem u korist neke druge transakcije T_j), obavljanje operacije $o_i[x]$ se odgađa dok LM ne otključa x
- odgađanjem izvršavanja nekih operacija transakcija utječe se na produciranu povijest - rezultat je serijalizabilna povijest



Vrste ključeva

- operacije čitanje/čitanje nisu konfliktne operacije
 - ključevi koji se postavljaju zbog tih operacija ih međusobno ne isključuju
- operacije čitanje/pisanje te operacije pisanje/pisanje jesu konfliktne operacije
 - ključevi koji se postavljaju zbog tih operacija ih moraju međusobno isključivati
- stoga se koriste dvije vrste ključeva
 - S-ključ (*S-lock*)
 - X-ključ (*X-lock*)

Vrste ključeva i operacije zaključavanja

Pregled pojmova:

- dijeljeni ključ, S-ključ
 - *shared lock, S-lock, read lock*
- ekskluzivni ključ, X-ključ
 - *exclusive lock, X-lock, write lock*
- zaključati element x S-ključem/X-ključem, postaviti S-ključ/X-ključ na element x
- otključati element x, ukloniti ključ s elementa x

- pored operacija $r_i[x]$, $w_i[x]$, a_i , c_i , u korist transakcije T_i obavljaju se i sljedeće operacije:
 - $sL_i[x]$ - element x zaključaj S-ključem
 - $xL_i[x]$ - element x zaključaj X-ključem
 - $u_i[x]$ - otključaj element x

Protokoli temeljeni na zaključavanju

- transakcija prije operacije $r[x]$ mora S-ključem ili X-ključem zaključati x
 - uspjet će samo onda kada x nije zaključan X-ključem neke druge transakcije (inače će morati čekati)
- transakcija prije operacije $w[x]$ mora X-ključem zaključati x (ili može promovirati prethodno postavljeni **vlastiti** S-ključ u X-ključ)
 - uspjet će samo onda kada x nije zaključan niti S niti X-ključem neke druge transakcije (inače će morati čekati)
- transakcija mora otpustiti sve svoje ključeve najkasnije u trenutku završetka

Tablica kompatibilnosti ključeva

T_i postavila ključ \rightarrow

	S	X
S	T	\perp
X	\perp	\perp

T_j traži ključ \downarrow

element u jednom trenutku može biti zaključan:
ili

- jednim ili više S-ključeva različitih transakcija
- samo jednim X-ključem

Temeljni dvofazni protokol zaključavanja

2PL protokol

$2PL_1$: za $r_i[x]$ u H moraju postojati $sL_i[x] < r_i[x] < u_i[x]$ ili $xL_i[x] < r_i[x] < u_i[x]$;

za $w_i[x]$ u H moraju postojati $xL_i[x] < w_i[x] < u_i[x]$

$2PL_2$: $sL_j[x]$ ili $xL_j[x]$ mogu slijediti $xL_i[x]$ tek nakon $u_i[x]$;

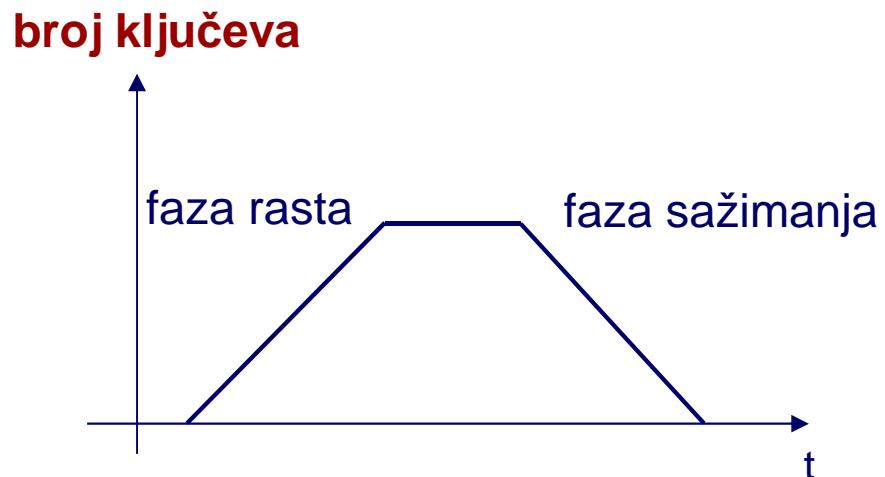
$xL_j[x]$ može slijediti $sL_i[x]$ tek nakon $u_i[x]$;

$2PL_3$: sve operacije sL_i i xL_i u H moraju prethoditi svim operacijama u_i

- $2PL_1$ propisuje da transakcija T_i prije obavljanja operacije mora zatražiti (i dobiti) ključ odgovarajuće vrste, te da ga nakon operacije mora otpustiti (ne nužno odmah)
- $2PL_2$ propisuje da element može biti zaključan ili jednim X-ključem ili s jednim ili više S-ključeva
- $2PL_3$ propisuje da transakcija T_i mora zaključati sve potrebne elemente **prije** nego otključa bilo koji element, odnosno transakcija koja je otpustila ključ ne može nakon toga postavljati nove ključeve
 - pravilo je izvor imena protokola jer proces zaključavanja ima dvije faze:
 - faza rasta (*growing phase*) u kojoj se elementi zaključavaju (i ne otključavaju)
 - faza sažimanja (*shrinking phase*) u kojoj se elementi otključavaju (i ne može se zaključati niti jedan element)

Temeljni dvofazni protokol zaključavanja

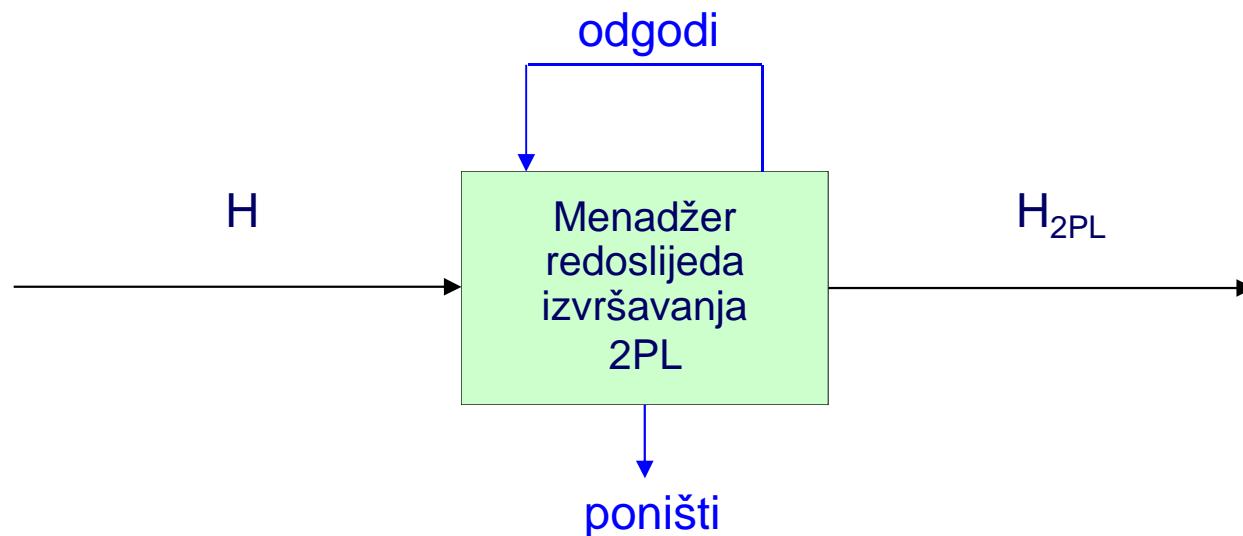
- Ilustracija pravila 2PL₃
- broj ključeva koje je postavila transakcija T_i



Temeljni dvofazni protokol zaključavanja

Teorem:

- svaka povijest H koja je nastala obavljanjem operacija u skladu s 2PL protokolom je serijalizabilna
- dokaz teorema se može naći u [Bernstein], str. 53.

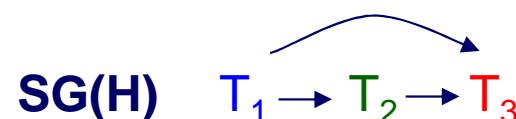
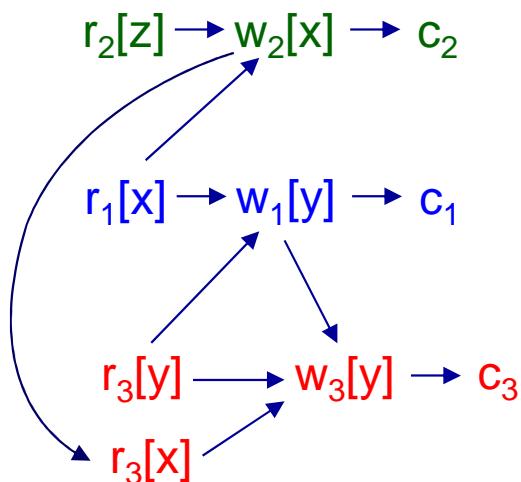


Temeljni dvofazni protokol zaključavanja

Primjer:

- menadžer transakcija upućuje menadžeru redoslijeda izvršavanja sljedeći niz operacija:

$H \quad r_2[z] \quad r_1[x] \quad w_2[x] \quad c_2 \quad r_3[x] \quad r_3[y] \quad w_1[y] \quad c_1 \quad w_3[y] \quad c_3$



H nije CSR

Za vježbu: o kojem karakterističnom problemu istodobnog pristupa je ovdje riječ?

- nekonzistentna analiza?
- izgubljena izmjena?
- prljavo čitanje?

Temeljni dvofazni protokol zaključavanja

Primjer (nastavak) : H r₂[z] r₁[x] w₂[x] c₂ r₃[x] r₃[y] w₁[y] c₁ w₃[y] c₃

	obavlja se	x	y	z
r ₂ [z]	sL ₂ [z], r ₂ [z]			S: T ₂
r ₁ [x]	sL ₁ [x], r ₁ [x]	S: T ₁		S: T ₂
w ₂ [x]	xL ₂ [x], odbijen X-ključ na x, T ₂ čeka	S: T ₁		S: T ₂
r ₃ [x]	sL ₃ [x], r ₃ [x]	S: T ₁ , T ₃		S: T ₂
r ₃ [y]	sL ₃ [y], r ₃ [y]	S: T ₁ , T ₃	S: T ₃	S: T ₂
w ₁ [y]	xL ₁ [y], odbijen X-ključ na y, T ₁ čeka	S: T ₁ , T ₃	S: T ₃	S: T ₂
w ₃ [y]	xL ₃ [y], w ₃ [y]	S: T ₁ , T ₃	X: T ₃	S: T ₂
c ₃	c ₃ , u ₃ [x], u ₃ [y]	S: T ₁		S: T ₂
w ₁ [y]	xL ₁ [y], w ₁ [y]	S: T ₁	X: T ₁	S: T ₂
c ₁	c ₁ , u ₁ [x], u ₁ [y]			S: T ₂
w ₂ [x]	xL ₂ [x], w ₂ [x]	X: T ₂		S: T ₂
c ₂	c ₂ , u ₂ [x], u ₂ [z]			

čekaj



Temeljni dvofazni protokol zaključavanja

Primjer (nastavak) :

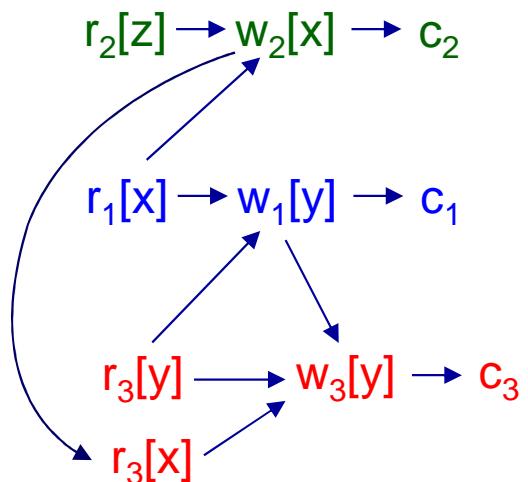
- producirana povijest s prethodne stranice
 $sL_2[z], r_2[z], sL_1[x], r_1[x], sL_3[x], r_3[x], sL_3[y], r_3[y], xL_3[y], w_3[y], \rightarrow \text{nastavak}$
 $c_3, u_3[x], u_3[y], xL_1[y], w_1[y], c_1, u_1[x], u_1[y], xL_2[x], w_2[x], c_2, u_2[x], u_2[z]$

- napomena: operacije zaključavanja i otključavanja u dalnjem tekstu će se prikazivati u povijestima samo onda kada su važne za objašnjavanje nekog koncepta
- u rješenjima zadataka (npr. na završnom ispitу) operacije zaključavanja i otključavanja treba prikazivati u povijestima samo onda kada se to eksplicitno zahtijeva tekstrom zadatka
 - međutim, treba uzeti u obzir njihovo djelovanje kada se u zadatku traži prikaz povijesti koju će producirati 2PL protokol

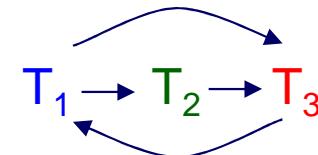
Temeljni dvofazni protokol zaključavanja

Primjer (nastavak) :

- ulazna H $r_2[z]$ $r_1[x]$ $w_2[x]$ c_2 $r_3[x]$ $r_3[y]$ $w_1[y]$ c_1 $w_3[y]$ c_3

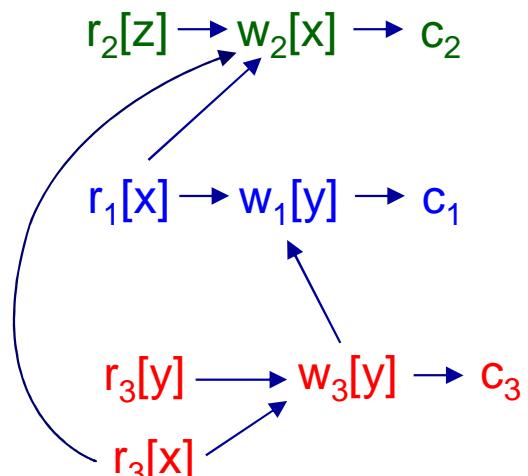


$SG(H)$



H nije CSR

- producirana H_{2PL} $r_2[z], r_1[x], r_3[x], r_3[y], w_3[y], c_3, w_1[y], c_1, w_2[x], c_2$



$SG(H_{2PL})$



H_{2PL} je CSR

Temeljni dvofazni protokol zaključavanja

- 2PL protokol je "stroži" nego je nužno za osiguravanje konflikt-serijalizabilnosti
- Primjer: povijest H jest konflikt-serijalizabilna (CSR)

$H \quad r_1[x], w_2[x], r_1[y], w_2[y], c_1, c_2$

$\text{SG}(H) \quad T_1 \rightarrow T_2$

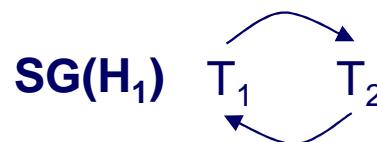
- 2PL protokol će za ulaznu povijest H producirati drugačiju izlaznu povijest (za vježbu odrediti koju). Činjenica da neka povijest H jest CSR, ne znači da takvu povijest 2PL protokol može producirati (nepromijenjenu), odnosno, ulazna povijest koja jest CSR će ipak, u nekim slučajevima, biti promijenjena 2PL protokolom

Ilustracija važnosti pravila 2PL₃

Primjer:

- menadžer transakcija upućuje menadžeru redoslijeda izvršavanja sljedeći niz operacija:

$H_1 \quad r_1[x], w_2[y], w_2[x], c_2, r_1[y], c_1$



$SG(H_1) \quad H_1 \text{ nije CSR}$

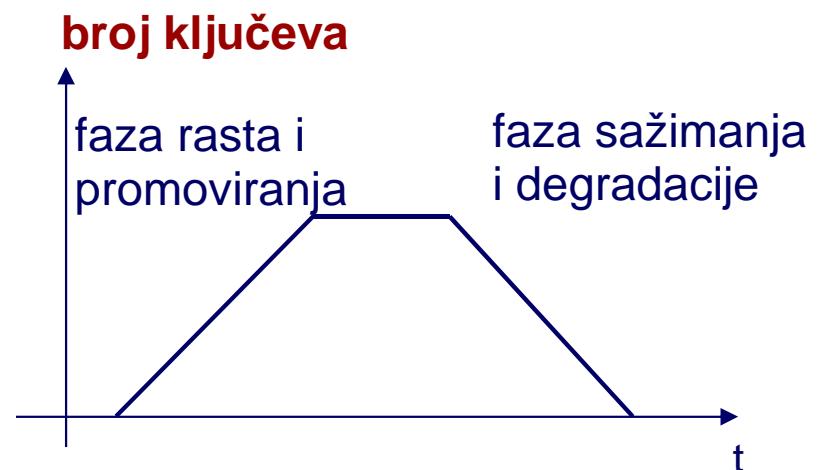
- menadžer redoslijeda izvršavanja koji ne bi primjenjivao 2PL₃ dopustio bi takvu neserializabilnu povijest:

$sL_1[x], r_1[x], u_1[x], xL_2[y], w_2[y], xL_2[x], w_2[x], c_2, u_2[y], u_2[x], sL_1[y], r_1[y], c_1, u_1[y]$

- T_1 je otključala x u fazi rasta, što se prema 2PL₃ ne bi smjelo dopustiti

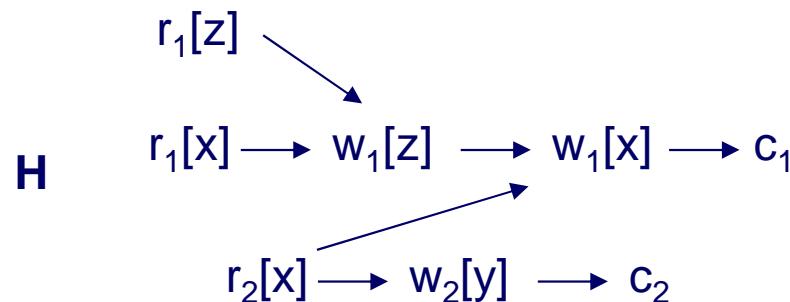
Konverzija ključeva

- transakcija T_i koja prvo čita, a zatim piše u element x se ponaša "priateljski" prema ostalim transakcijama: prije obavljanja operacije $r_i[x]$ postavlja samo S-ključ, a zatim tek prije obavljanja operacije $w_i[x]$ postavlja X-ključ. Taj se postupak naziva konverzija ključeva (*lock conversion*), a omogućava ostalim transakcijama da u vremenu između operacija $r_i[x]$ i $w_i[x]$ postave svoj S-ključ i pročitaju element x
 - konverzija ključeva je omogućena činjenicom da ključevi iste transakcije nikad nisu u međusobnom konfliktu
- teoretski, konverzija ključeva može ići u smjeru promocije (*upgrade lock*), kada se S-ključ promovira u X-ključ, a također i u smjeru degradacije (*downgrade lock*), kada se X-ključ degradira u S-ključ
- promocija se smije obavljati isključivo u fazi rasta (*growing phase*), a degradacija isključivo u fazi sažimanja (*shrinking phase*)
- degradacija se u praksi ne koristi (razlog: vidjeti kasnije striktni 2PL protokol)



Konverzija ključeva

Primjer:



$sL_1[x], r_1[x], sL_1[z], r_1[z] \rightarrow$ nastavak

$sL_2[x], r_2[x], xL_1[z], w_1[z], xL_2[y], w_2[y], c_2, u_2[x], u_2[y] \rightarrow$ nastavak

$xL_1[x], w_1[x], c_1, u_1[x], u_1[z]$

- transakcija T_1 je promovirala ključ nad elementom x (umjesto da ga je odmah zaključala X-ključem)
 - transakcija T_2 uspjela je u vremenu između operacija $sL_1[x]$ i $xL_1[x]$ obaviti operaciju $r_2[x]$

Striktni i rigorozni 2PL protokol

Striktni 2PL protokol

- temeljnim 2PL protokolom garantira se da je povijest CSR, ali ne i RC, ACA i ST
- temeljni 2PL protokol nadopunjava se dodatnim pravilom
 - **X-ključevi smiju se otpustiti tek poslije točke potvrđivanja transakcije**
- ako je transakcija T_i pisala u element x, transakcija T_j ne smije čitati niti pisati u element x dok T_i ne terminira (jer T_i drži X-ključ na x do kraja transakcije), a to je kriterij striktne (ST) povijesti
- iz toga proizlazi i ime protokola: **striktni 2PL protokol**
- striktni 2PL protokol po definiciji ne dopušta degradaciju X-ključeva

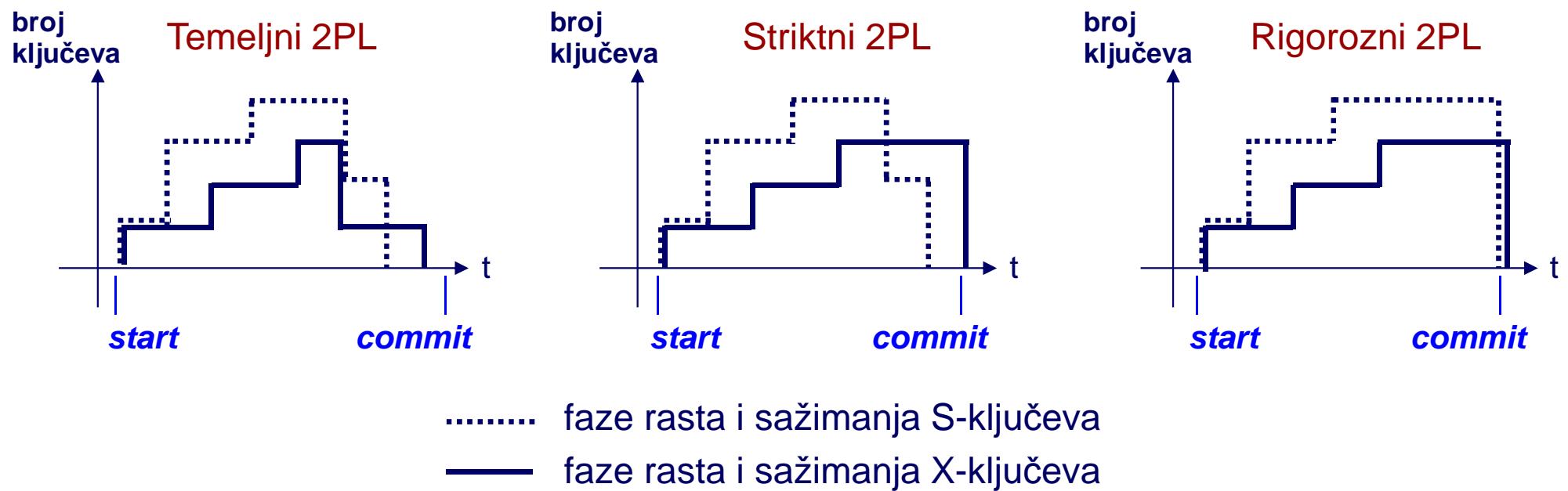
Striktni i rigorozni 2PL protokol

Rigorozni 2PL protokol

- iz praktičnih razloga, najčešće se implementira protokol s dodatnim pravilom
 - **svi (a ne samo X-ključevi) otpuštaju se tek na kraju transakcije**
- praktični razlog: protokolom se olakšava implementacija dvofaznosti protokola - transakcija (usput) pomoću *commit* ili *abort* naredbe menadžera transakcija signalizira da neće tražiti postavljanje novih ključeva i da može početi s otpuštanjem ključeva (inače, za tu svrhu bi morale postojati posebne naredbe)
- rigorozni 2PL protokol po definiciji ne dopušta degradaciju ključeva

Značajke 2PL protokola

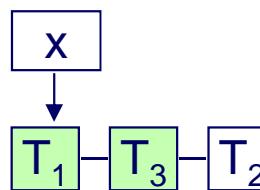
- slikom je ilustrirana razlika između temeljne, striktne i rigorozne verzije 2PL protokola. Prikazane su faze rasta i faze sažimanja jedne transakcije



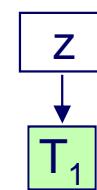
- u rigoroznom 2PL se S-ključevi mogu otpustiti neposredno prije točke potvrđivanja (*commit point*), a X-ključevi neposredno nakon nje. U praksi se, efikasnosti radi, obje vrste ključeva otpuštaju neposredno nakon točke potvrđivanja

Implementacija menadžera zaključavanja

- menadžer redoslijeda izvršavanja upućuje zahtjeve za postavljanje i uklanjanje ključeva menadžeru zaključavanja (LM)
- implementacija menadžera zaključavanja znatno ovisi o arhitekturi SUBP-a, stoga je ovdje opisana načelno
- za svaki element koji je trenutno zaključan održava se povezana lista, čiji svaki element sadrži: identifikator transakcije koja je zatražila postavljanje ključa, vrstu ključa i informaciju o tome je li postavljenje ključa odobreno



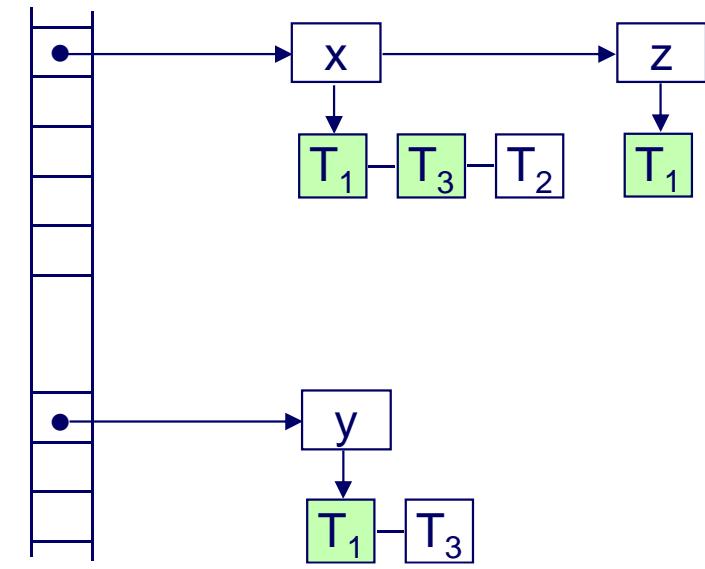
na slikama je izostavljena
informacija o vrsti ključeva



- postavljeni su ključevi na x za T_1 i T_3 , a ključ za T_2 nije odobren (čeka odobrenje)
- postavljen je ključ na z za T_1
- liste se pohranjuju u tablicu s raspršenim adresiranjem. Ključ pretrage je identifikator elementa baze podataka.

Implementacija menadžera zaključavanja

tablica ključeva (*lock table*)



- funkcija raspršenja je za elemente x i z izračunala istu adresu
- x je zaključan u korist transakcija T₁ i T₃, transakcija T₂ čeka na postavljanje ključa na x
- z je zaključan u korist transakcije T₁
- y je zaključan u korist transakcije T₁, transakcija T₃ čeka na postavljanje ključa na y

Hash table

- kada LM dobije zahtjev za postavljanje ključa na element x, provjerava postoji li lista za element x. Ako ne postoji, lista se kreira. Zahtjev se dodaje na kraj liste (*queue*). Ako se ključ ne može postaviti, transakcija ulazi u stanje čekanja
- sadržaj tablice ključeva prikazan u primjeru na slici mogao bi biti posljedica pokušaja obavljanja sljedećeg niza operacija

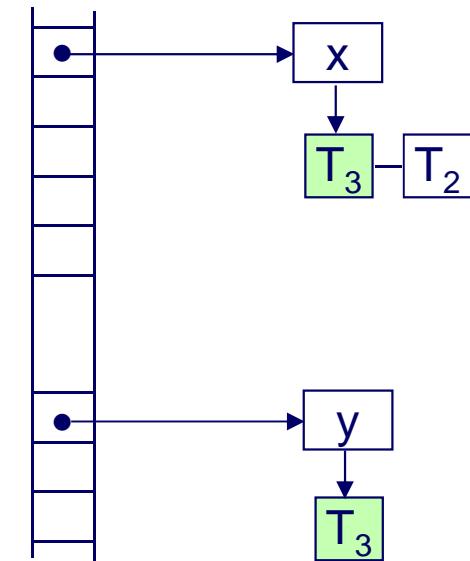
r₁[x], r₁[z], r₃[x], w₁[y], **w₂[x]**, r₃[y]

Implementacija menadžera zaključavanja

- kada LM dobije zahtjev za uklanjanje ključa s elementa x , izbacuje odgovarajući zapis iz liste i provjerava postoji li u toj listi neki zahtjev koji se može odobriti. Ako se zahtjev odobri, signalizira se procesu koji je čekao na postavljanje ključa
- npr. ako transakcija T_1 iz prethodnog primjera terminira, transakcija T_3 će moći nastaviti s obavljanjem operacije $r_3[y]$

... $w_1[y]$, $w_2[x]$, $r_3[y]$, c_1 , $r_3[y]$

u ovom trenutku, uklonjeni su ključevi za T_1 , odobren X-ključ za y , signalizirano T_3 da može nastaviti s operacijom koja je čekala odobrenje ključa za y



- mehanizam kojim se osigurava blokiranje transakcije, a kasnije njezino deblokiranje, ovisi o načinu sinkronizacije među procesima/dretvama u operacijskom sustavu ili SUBP (semafori, monitori)
- na slikama nije prikazano, ali struktura tablice ključeva se mora optimirati za efikasno uklanjanje svih ključeva koje je postavila jedna transakcija. To je postupak koji se za rigorozni 2PL obavlja vrlo često (pri svakom prekidu ili potvrđivanju transakcije)

Modalitet zaključavanja

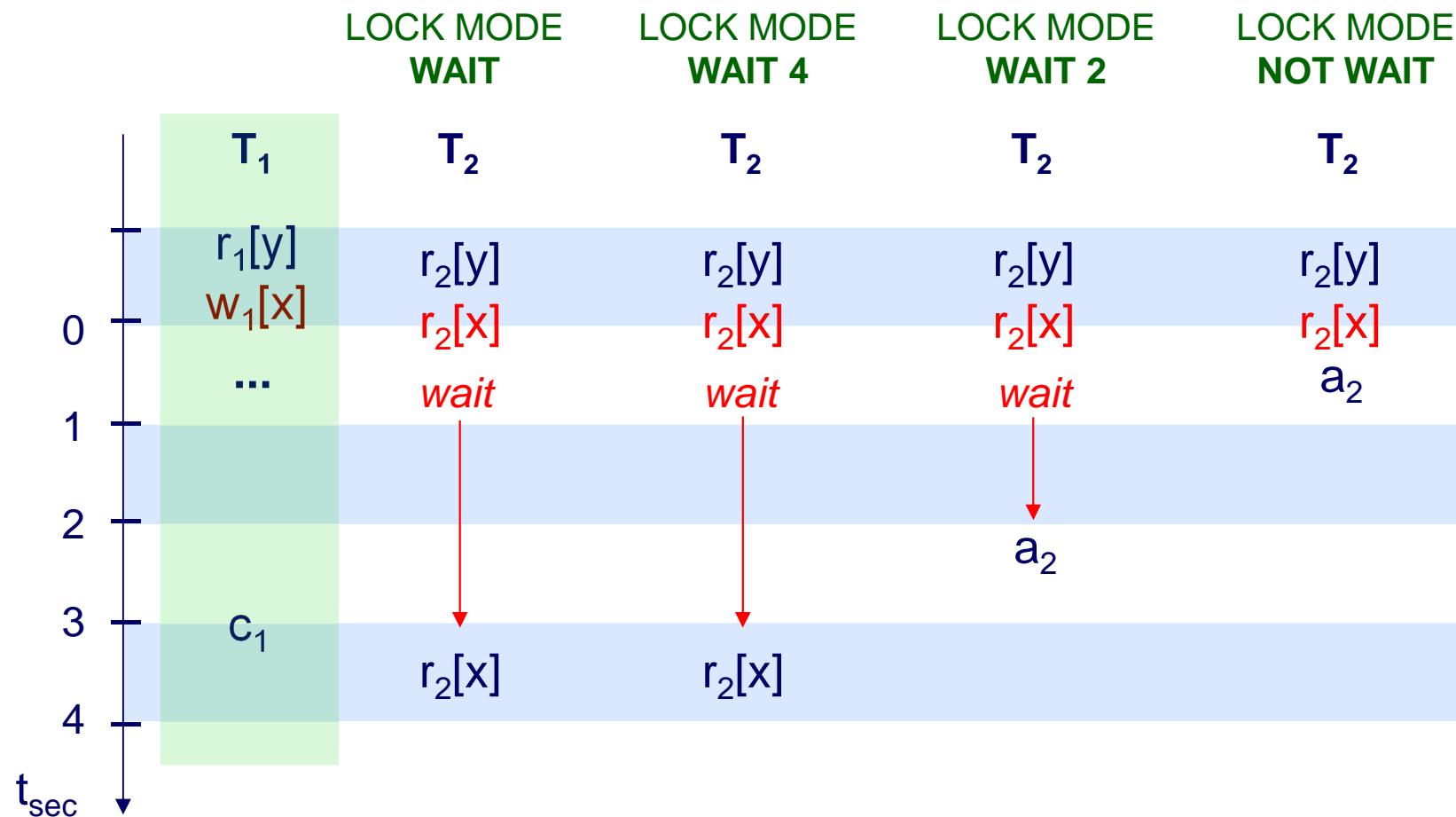
- kada LM ne može ispuniti zahtjev za postavljanjem ključa, zahtjev se stavlja u red čekanja, a transakcija se blokira do trenutka kada će se zahtjev moći ispuniti
- umjesto blokiranja, moguće je odabrati drugačiji pristup:
 - LM prekida operaciju/transakciju čiji zahtjev za postavljanje ključa nije u mogućnosti ispuniti tijekom zadanog perioda
 - LM prekida operaciju/transakciju čiji zahtjev za postavljanjem ključa ne može ispuniti odmah

IBM IDS:

- **SET LOCK MODE TO WAIT**
- **SET LOCK MODE TO WAIT n**
 - n je broj sekundi, najdulje dopušteno vrijeme čekanja na ispunjenje zahtjeva
- **SET LOCK MODE TO NOT WAIT**

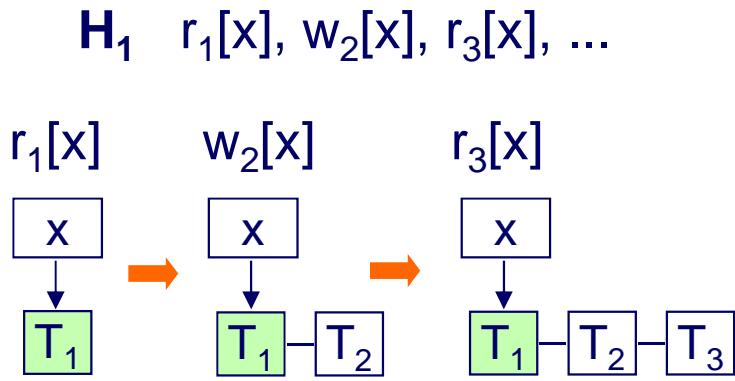
Modalitet zaključavanja

Primjer:



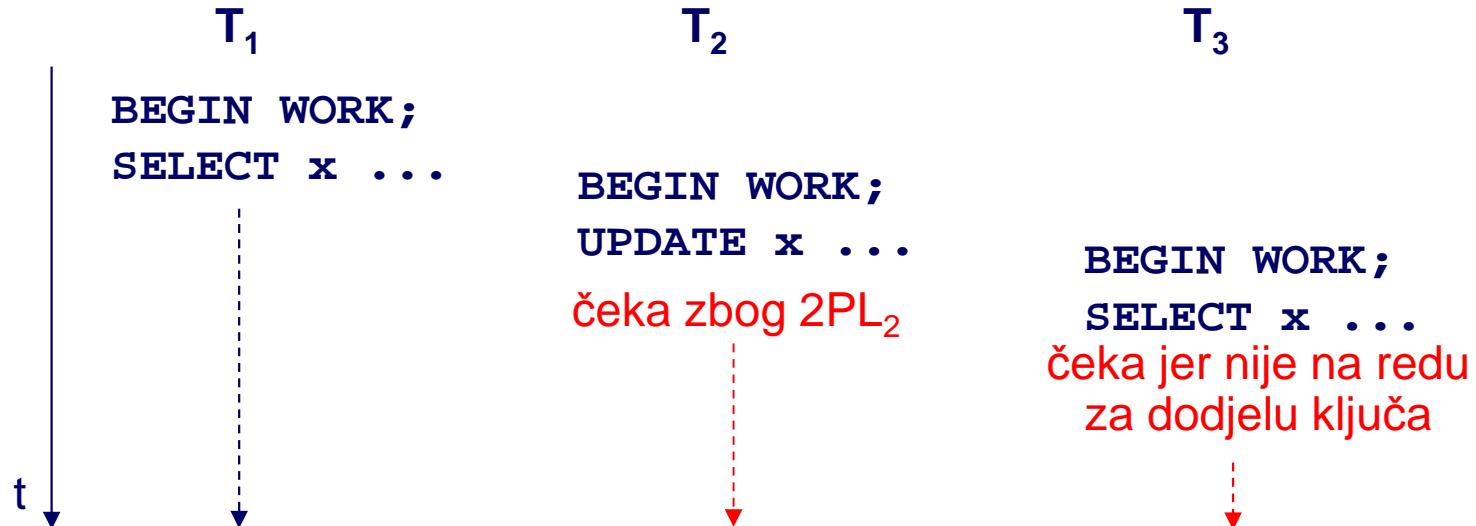
Implementacija menadžera zaključavanja

- zaključavanje elementa se **mora** provoditi prema principu *first come - first served*. U suprotnom, moguća je pojava izgladnjivanja (*starvation*) transakcija



- vodeći se isključivo pravilima 2PL protokola, LM je mogao odobriti ključ na x za T_3 . Međutim, takvim pristupom bi transakcije T_4, T_5, \dots , koje u sustav pristižu nakon T_2 , svojim uzastopnim zahtjevima za postavljanjem S-ključa na x, mogle dovesti do izgladnjivanja transakcije T_2

IBM IDS:



Potpuni zastoj

Potpuni zastoj

- važan problem kod 2PL protokola je mogućnost nastanka potpunog zastoja
 - 2PL producira CSR povijest bez obzira kojim redoslijedom operacije pristižu u menadžer transakcija. Međutim, neke povijesti sustav mogu dovesti u stanje u kojem nije moguće nastaviti s 2PL protokolom bez da se prekrši neko od 2PL pravila

$H \quad r_1[x], w_2[y], w_1[y], w_2[x], c_1, c_2 \quad sL_1[x], r_1[x], xL_2[y], w_2[y], xL_1[y], xL_2[x]$

$\uparrow \quad \uparrow$
wait wait

- T_1 čeka postavljanje ključa na y (nije moguće prije završetka transakcije T_2),
 T_2 čeka postavljanje ključa na x (nije moguće prije završetka transakcije T_1)
 - nastaje potpuni zastoj (*deadlock*)
- shema potpunog zastoja može biti složenija od gore opisane (u kojoj sudjeluju samo dvije transakcije)
- potpuni zastoj nastaje onda kada postoji neki skup transakcija $T = \{ T_0, T_1, T_2, \dots, T_n \}$ takvih da T_0 čeka zbog ključa kojeg je postavila T_1 , T_1 čeka zbog ključa kojeg je postavila T_2, \dots, T_n čeka zbog ključa kojeg je postavila T_0 . Niti jedna od transakcija iz skupa T ne može nastaviti s radom.

Potpuni zastoj

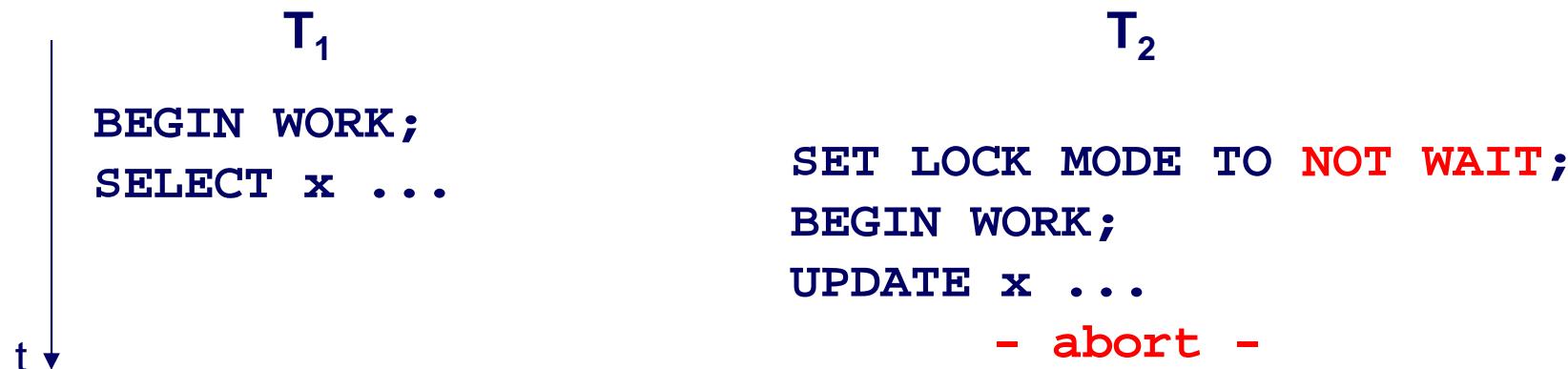
- problemu potpunih zastoja može se pristupiti na dva načina:
 1. prevencija (izbjegavanje) potpunih zastoja: 2PL protokol se dopunjava pravilima koja onemogućuju pojavu potpunih zastoja
 - zabrana čekanja
 - korištenje konzervativnog 2PL protokola
 - propisivanje redoslijeda zaključavanja
 - metode s vremenskim oznakama transakcija
 2. detekcija i oporavak od potpunog zastoja: potpuni zastoji se događaju, ali ih sustav detektira te poništavanjem jedne ili više transakcija prekida potpuni zastoj
 - praćenje vremena čekanja na postavljanje ključeva
 - korištenje grafa čekanja (WFG)

Zabrana čekanja

- kada transakcija T_i pokuša zaključati element kojeg je nekompatibilnim ključem već zaključala transakcija T_j , T_i se prekida i ponistiava (i ponovo pokreće s malom vremenskom odgodom)
- vrlo jednostavna metoda koja onemogućava pojavu potpunog zastoja, ali drastično ograničava mogućnost istodobnog izvršavanja transakcija. Može uzrokovati ponistiavanje velikog broja transakcija koje zapravo nisu bile u potpunom zastaju

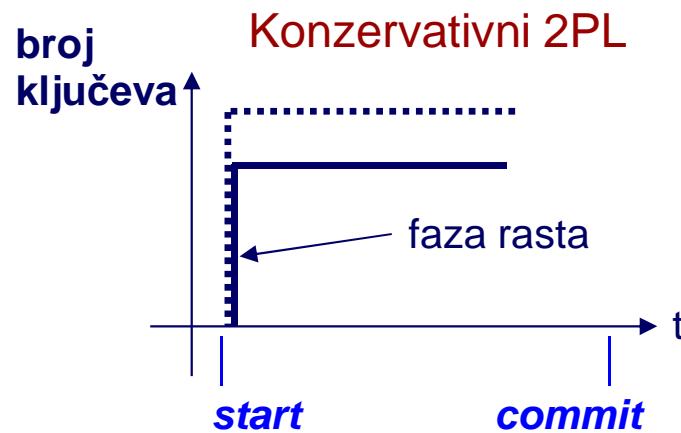
Primjer:

- opisani način rukovanja potpunim zastojima u sustavu IBM IDS može se postići korištenjem modaliteta zaključavanja:



Konzervativni 2PL protokol

- transakcija mora na početku deklarirati skup podataka koji će koristiti za čitanje i skup podataka koji će koristiti za pisanje (*read-set, write-set*)
- svi potrebni ključevi se postavljaju na samom početku transakcije u jednom, nedjeljivom koraku. Transakcija koja nije uspjela postaviti sve ključeve ne smije zadržati niti jedan



- rijetki su slučajevi u kojima se na samom početku transakcije može predvidjeti *read-set* i *write-set*
- niska iskoristivost sustava jer su elementi zaključani tijekom relativno dugog perioda u kojem se zapravo ne koriste. Taj je nedostatak manje značajan u slučaju vrlo kratkih transakcija

Propisivanje redoslijeda zaključavanja

- utvrđuje se potpuni poredak među elementima baze podataka
 - $x_1 < x_2 < x_3 \dots < x_n$
- svaka transakcija mora zaključavati elemente u redoslijedu koji je konzistentan s utvrđenim poretkom elemenata. Ako je T zaključala element x_i , tada u nastavku ne može zaključati x_j za kojeg vrijedi $x_j < x_i$
- metoda je nepraktična jer zahtijeva da korisnik (programer) ili sustav vodi računa o utvrđenom poretku elemenata

Metode s vremenskim oznakama transakcija

- *transaction timestamp*
- metode se temelje na vremenskim oznakama transakcija (*timestamp*). Sustav svakoj transakciji T_i pridjeljuje jedinstvenu vremensku oznaku $TS(T_i)$ u trenutku kada se počne izvršavati
- kada transakcija T_i pokuša zaključati element kojeg je nekompatibilnim ključem već zaključala transakcija T_j :

Metoda *wait-die*:

- ako je T_i starija, $TS(T_i) < TS(T_j)$, T_i čeka; ako je T_i mlađa, T_i se prekida, poništava i ponovo pokreće s vremenskom oznakom $TS(T_i)$
 - starija čeka da mlađa otpusti ključeve
 - mlađoj nije dopušteno čekanje na otpuštanje ključeva koje drži starija

Objašnjenje naziva

- iz perspektive T_i koja čeka na zaključavanje elementa kojeg je zaključala T_j
 - ako sam starija, čekam (*wait*)
 - ako sam mlađa, poništavam se (*die*) i ponovo pokrećem

Metode s vremenskim oznakama transakcija

- kada transakcija T_i pokuša zaključati element kojeg je nekompatibilnim ključem već zaključala transakcija T_j :

Metoda *wound-wait*:

- ako je T_i starija, $TS(T_i) < TS(T_j)$, T_j se prekida, poništava i ponovo pokreće s vremenskom oznakom $TS(T_j)$; ako je T_i mlađa, T_i čeka;
 - starija uzrokuje prekid mlađe koja drži ključeve
 - mlađoj je dopušteno čekanje na otpuštanje ključeva koje drži starija

Objašnjenje naziva

- iz perspektive T_i koja čeka na zaključavanje elementa kojeg je zaključala T_j
 - ako sam starija, prekidam mlađu (*wound*) koja se ponovo pokreće
 - ako sam mlađa, čekam (*wait*)

Metode s vremenskim oznakama transakcija

Metoda *wait-die*:

- moguće je da starija transakcija čeka zbog ključa kojeg je postavila mlađa, ali mlađa nikad ne čeka zbog ključa kojeg je postavila starija (umjesto toga, mlađa se prekida)
- dokaz da se metodom sprečava potpuni zastoj:
 - ako su transakcije T_1, T_2, \dots, T_n u potpunom zastaju: $T_1 \rightarrow T_2 \rightarrow \dots \rightarrow T_n \rightarrow T_1$
 - to znači da T_1 čeka na T_2 , T_2 na T_3 , ..., T_n na T_1
 - samo starije transakcije mogu čekati mlađe
 - to znači $TS(T_1) < TS(T_2) < \dots < TS(T_n) < TS(T_1)$
 - kontradikcija \Rightarrow potpuni zastoj je nemoguć

Metode s vremenskim oznakama transakcija

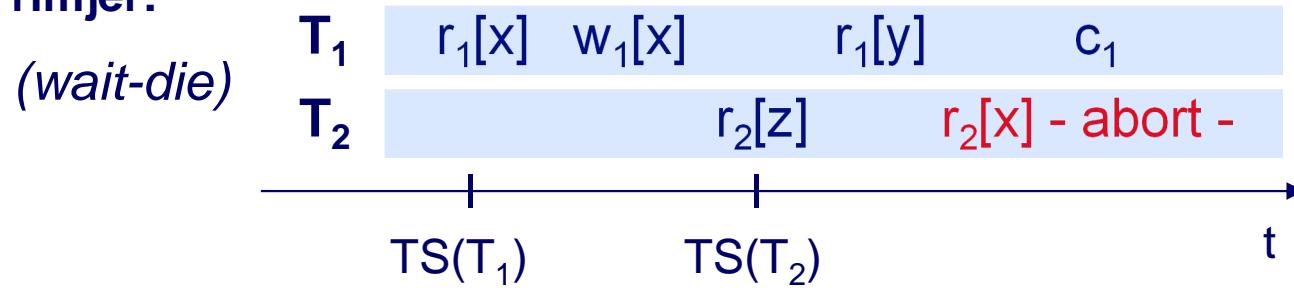
Metoda *wound-wait*:

- moguće je da mlađa transakcija čeka zbog ključa kojeg je postavila starija, ali starija nikad ne čeka zbog ključa kojeg je postavila mlađa (umjesto toga, mlađa se prekida)
- dokaz da se metodom sprečava potpuni zastoj:
 - ako su transakcije T_1, T_2, \dots, T_n u potpunom zastaju: $T_1 \rightarrow T_2 \rightarrow \dots \rightarrow T_n \rightarrow T_1$
 - to znači da T_1 čeka na T_2 , T_2 na T_3 , ..., T_n na T_1
 - samo mlađe transakcije mogu čekati starije
 - to znači $TS(T_1) > TS(T_2) > \dots > TS(T_n) > TS(T_1)$
 - kontradikcija \Rightarrow potpuni zastoj je nemoguć

Metode s vremenskim oznakama transakcija

- u obje metode mlađa transakcija može biti poništena (i ponovo pokrenuta). Izgladnjivanje "mlađe" transakcije je spriječeno time što se prekinuta transakcije ponovo pokreće sa svojom "starom" vremenskom oznakom, čime postaje "sve starija"
- metode ponekad poništavaju transakcije i onda kada potpuni zastoj zapravo ne postoji

Primjer:



- T_2 je nepotrebno poništena. Mogla je nastaviti nakon c_1

Za vježbu:

- načiniti primjer u kojem *wound-wait* poništava transakciju iako do potpunog zastoja nije moglo doći

Praćenje vremena čekanja na postavljanje ključeva

- *timeout-based* metoda
- djelomično pripada u grupu metoda prevencije, djelomično u grupu metoda detekcije
- ako transakcija T_i čeka na postavljanje ključa dulje od parametrom `LOCK_TIMEOUT` definiranog perioda, pretpostavlja se da je T_i uključena u potpuni zastoj, prekida se i poništava
- vrlo jednostavna metoda koja onemogućava pojavu potpunog zastoja, ali može uzrokovati poništavanje transakcija koje zapravo nisu bile u potpunom zastaju
- određivanje parametra `LOCK_TIMEOUT` je kritično
 - prekratko vrijeme čekanja uzrokuje poništavanje transakcija koje možda samo čekaju na završetak neke dugotrajne transakcije
 - predugo vrijeme čekanja uzrokuje nepotrebno zadržavanje u slučajevima kada se potpuni zastoj zaista pojavio
- parametar je naročito teško odrediti kada se u sustavu izvršavaju i kratke i duge transakcije
- metoda se često koristi u distribuiranim sustavima, u kojima bi implementacija drugih metoda bila previše složena

Graf čekanja, WFG

- *wait-for graph (WFG)*
- WFG je usmjereni graf, $WFG = (N, E)$, u kojem čvorovi predstavljaju transakcije, a lúkovi predstavljaju ovisnosti među transakcijama: transakcija T_i čeka zbog ključa kojeg je postavila transakcija T_j
- konstrukcija WFG
 - u graf se novi čvor s označom T_i dodaje pri pokretanju transakcije T_i
 - u graf se dodaje lúk $T_i \rightarrow T_j$ ako transakcija T_i čeka zbog ključa kojeg drži transakcija T_j
 - lúk $T_i \rightarrow T_j$ se izbacuje iz grafa kada ključ čiji je vlasnik T_j prestane blokirati T_i
 - čvor T_i (i svi pripadni lúkovi) se izbacuje iz grafa kada T_i terminira

Uvjet za postojanje potpunog zastoja:

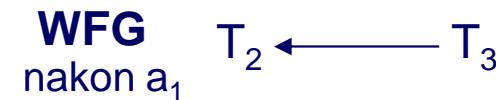
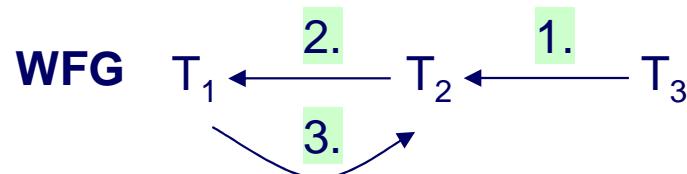
- u sustavu postoji potpuni zastoj ako i samo ako u WFG postoji petlja

Graf čekanja, WFG

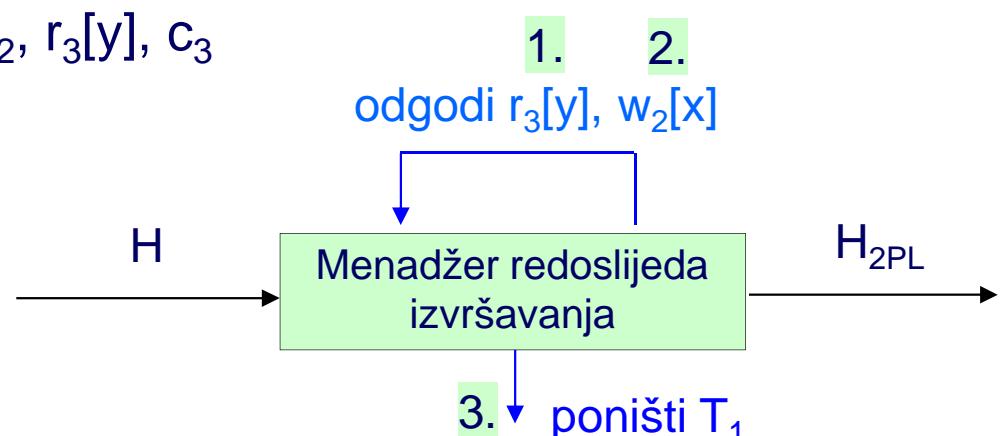
Primjer: $H = r_1[x], r_2[z], w_2[y], r_3[y], w_2[x], r_1[z], w_1[z], c_1, c_2, c_3$

$H_{2PL} = r_1[x], r_2[z], w_2[y], \textcolor{red}{r_3[y]}, \textcolor{red}{w_2[x]}, r_1[z], \textcolor{red}{w_1[z]}, a_1, w_2[x], c_2, r_3[y], c_3$

zbog utvrđene petlje u WFG



$H_{2PL} = r_1[x], r_2[z], w_2[y], r_1[z], a_1, w_2[x], c_2, r_3[y], c_3$



Graf čekanja, WFG

Različiti pristupi detekciji potpunog zastoja:

- ispitivanjem WFG svaki puta kada se u WFG doda novi lûk
 - ako su potpuni zastoji relativno rijetki, može izazvati nepotreban utrošak vremena
- u određenim vremenskim intervalima ili nakon dodavanja određenog broja lûkova
 - potpuni zastoj može ostati neotkriven relativno dugo
 - (potpuni zastoji koji se relativno dugo ne otkriju ipak će biti otkriveni, jer se potpuni zastoj nikad ne razriješi sam od sebe)

Razriješenje potpunog zastoja:

- menadžer redoslijeda izvršavanja mora odabrati i prekinuti jednu od transakcija uključenih u potpuni zastoj. Ta se transakcija naziva žrtva (*victim*)
- poništavanje žrtve će izbaciti čvor i pripadne lûkove iz WFG
- kao žrtvu je poželjno odabrati transakciju čiji će prekid izazvati minimalni trošak
- pojam minimalnog troška nije precizno definiran i ovisi o mnogim faktorima

Graf čekanja, WFG

Faktori koji utječu na trošak prekida transakcije koja sudjeluje u potpunom zastoju:

- među transakcijama koje sudjeluju u potpunom zastoju:
 - odabrati transakciju koja je obavila najmanju količinu ukupnog posla (u to su uključene i operacije pisanja i operacije čitanja)
 - odabrati transakciju čije će poništavanje uzrokovati obavljanje manjeg broja *undo* operacija (dakle onu koja je obavila manji broj operacija pisanja)
 - izbjegavati za žrtvu odabrati transakciju koja je blizu završetka
 - rijetko primjenjiv kriterij, jer SUBP bi trebao imati informaciju o budućem ponašanju transakcije
 - izbjegavati za žrtvu odabrati transakciju koja je u prethodnim pokušajima izvršavanja već bila odabrana kao žrtva (jednom ili više puta). U suprotnom, transakcijama koje uzastopno ulaze u potpuni zastoj i uzastopno ih se odabire kao žrtvu, prijeti izgladnjivanje

Literatura:

- P. Bernstein, V. Hadzilacos, N. Goodman. Concurrency control and recovery in database systems. Addison-Wesley Publishing Company. Reading, Massachusetts. 1987.
- H. Garcia-Molina, J. Ullman, J. Widom. Database system implementation. Prentice-Hall International, Inc. Upper Saddle River, New Jersey. 2000.
- A. Silberschatz, H. Korth, S. Sudarshan. Database system concepts, 4th ed. McGraw-Hill. 2002.
- R. Elmasri, S. Navathe. Fundamentals of database systems, 3rd ed. Addison-Wesley Publishing Company. Reading, Massachusetts. 2000.
- H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, P. O'Neil. A critique of ANSI SQL isolation levels. Proceedings of the 1995 ACM SIGMOD Int. conf. on Management of Data. San Jose, California. 1995.
- IBM Informix Guide to SQL: Reference, Version 11.1, IBM
- Oracle Database Concepts, 10g Release 2, Oracle
- Oracle Database SQL Reference, 10g Release 2, Oracle

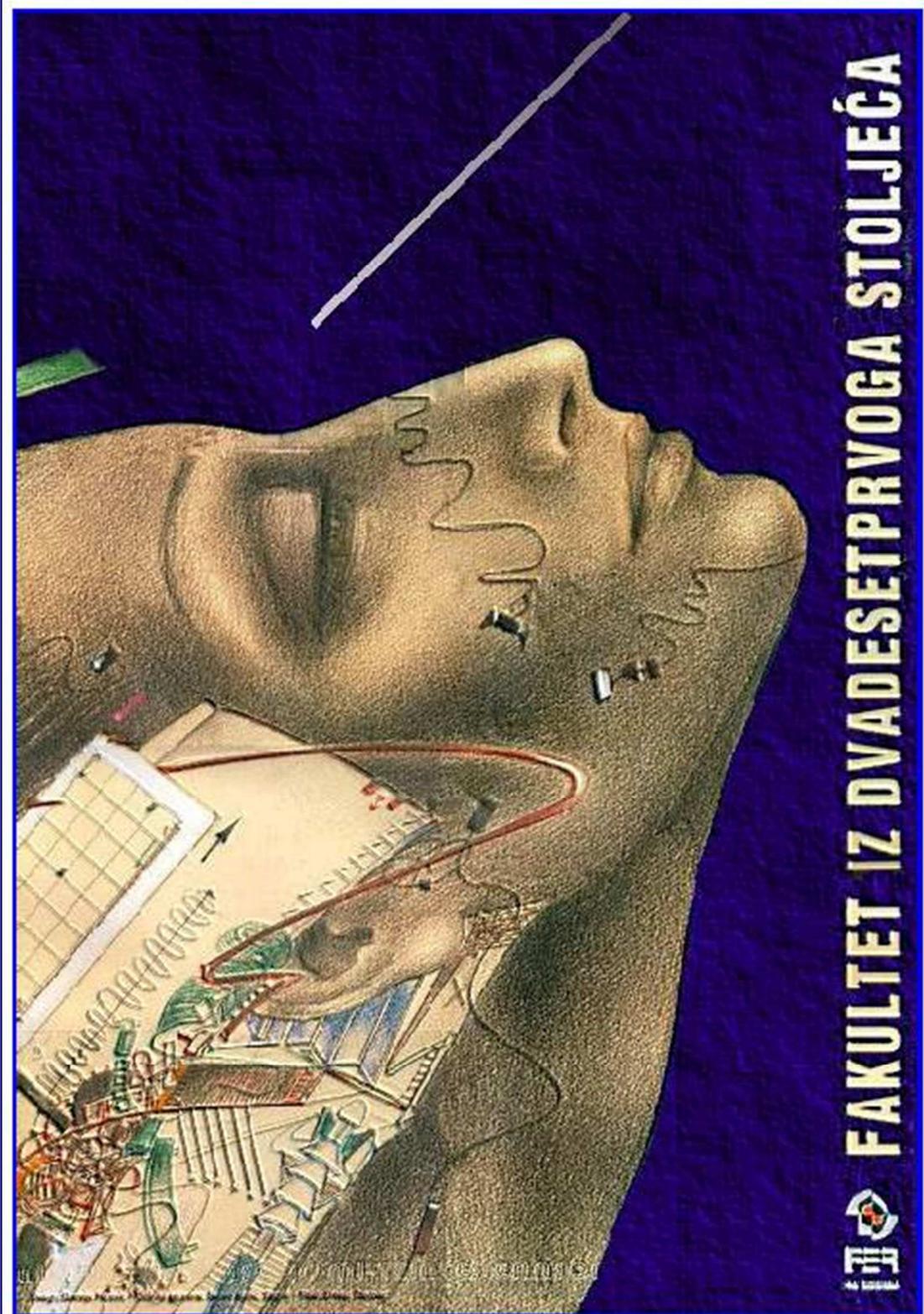
Sustavi baza podataka

Predavanja

10. Kontrola istodobnog pristupa

(3. dio)

svibanj 2014.



FAKULTET IZ DVADESETPRVOGA STOLJEĆA

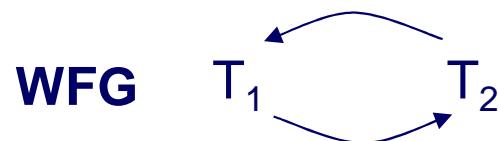
Promoviranje ključeva i potpuni zastoj

- promoviranje ključeva često dovodi do potpunog zastoja
- npr. usporedno izvršavanje transakcija oblika ... $r[x]$... $w[x]$

$H \quad r_1[x], r_2[x], w_1[x], c_1, w_2[x], c_2$

$H_{2PL} \quad r_1[x], r_2[x], w_1[x], w_2[x]$

↑ ↑
wait wait



Promoviranje ključeva i potpuni zastoj

Primjer: $r_1[x]$ $r_2[x]$ $r_3[x]$ c_3 $w_1[x]$ c_1 $w_2[x]$ c_2

		x
$r_1[x]$	T_1 : S -ključ na x, $r_1[x]$	S : T_1
$r_2[x]$	T_2 : S -ključ na x, $r_2[x]$	S : T_1 S : T_2
$r_3[x]$	T_3 : S -ključ na x, $r_3[x]$	S : T_1 S : T_2 S : T_3
c_3	T_3 : otpusti S -ključ na x, c_3	S : T_1 S : T_2
$w_1[x]$	T_1 : odbijen S \rightarrow X -ključ na x, T_1 čeka	S : T_1 S : T_2
$w_2[x]$	T_2 : odbijen S \rightarrow X -ključ na x, T_2 čeka	S : T_1 S : T_2
Potpuni zastoj: T_1 čeka T_2 , T_2 čeka T_1		

- Rješenje: uvođenje treće vrste ključa - ključa za izmjenu

Ključ za izmjenu

- ključ za izmjenu, *update-lock*, *U-lock*, *promotable-lock*
 - $uL_i[x]$ - element x zaključaj U-ključem
1. ako T_i zaključa element x U-ključem
 - T_i može čitati x
 - T_i može U-ključ promovirati u X-ključ (pod uvjetom da u trenutku promocije x nije zaključan S-ključevima ostalih transakcija)
 2. T_i može postaviti U-ključ na element koji je već zaključan S-ključevima drugih transakcija
 3. ako T_i zaključa element x U-ključem, ostale transakcije na element x ne smiju postavljati U-ključeve ili X-ključeve
 4. T_i ne smije S-ključ promovirati u U-ključ ili X-ključ
 - tj. ako T_i namjerava pisati u x, mora x ili izravno zaključati X-ključem, ili x čitati uz postavljen U-ključ te ga prije pisanja promovirati u X-ključ
 - tj. transakcija koja čita element x, a kasnije će ga mijenjati, ne smije x prethodno zaključati S-ključem

Ključ za izmjenu

Proširena tablica kompatibilnosti ključeva (U-ključ)

T_i postavila ključ →

T_j traži ključ ↓

	S	X	U
S	T	⊥	T
X	⊥	⊥	⊥
U	T	⊥	⊥

jednako kao u
temeljnoj tablici

Ključ za izmjenu

Primjer: $r_1[x]$ $r_2[x]$ $r_3[x]$ c_3 $w_1[x]$ c_1 $w_2[x]$ c_2

		x
$r_1[x]$	T_1 : U -ključ na x, $r_1[x]$	U : T_1
$r_2[x]$	T_2 : odbijen U -ključ na x, T_2 čeka	U : T_1
$r_3[x]$	T_3 : S -ključ na x, $r_3[x]$	U : T_1 S : T_3
c_3	T_3 : otpusti S -ključ na x, c_3	U : T_1
$w_1[x]$	T_1 : U \rightarrow X -ključ na x, $w_1[x]$	X : T_1
c_1	T_1 : otpusti X -ključ na x, c_1	
$r_2[x]$	T_2 : U -ključ na x, $r_2[x]$	U : T_2
$w_2[x]$	T_2 : U \rightarrow X -ključ na x, $w_2[x]$	X : T_2
c_2	T_2 : otpusti X -ključ na x, c_2	

- potpuni zastoj je spriječen, izvršavaju se operacije:

$r_1[x]$ $r_3[x]$ c_3 $w_1[x]$ c_1 $r_2[x]$ $w_2[x]$ c_2

Ključ za izmjenu

- osim pravila koja proizlaze iz tablice kompatibilnosti, treba paziti na primjenu pravila prema kojem se S-ključ ne smije promovirati u X-ključ (čak niti posredno, $S\text{-ključ} \rightarrow U\text{-ključ} \rightarrow X\text{-ključ}$). Kad se to pravilo ne bi primjenjivalo, U-ključevi bi bili beskorisni po pitanju izbjegavanja potpunih zastoja

Primjer: $r_1[x] \ r_2[x] \ r_3[x] \ c_3 \ w_1[x] \ c_1 \ w_2[x] \ c_2$

		x
$r_1[x]$	T_1 : U -ključ na x, $r_1[x]$	U : T_1
$r_2[x]$	T_2 : S -ključ na x, $r_2[x]$	U : T_1 S : T_2
$r_3[x]$	T_3 : S -ključ na x, $r_3[x]$	U : T_1 S : T_2 S : T_3
c_3	T_3 : otpusti S -ključ na x, c_3	U : T_1 S : T_2
$w_1[x]$	T_1 : odbijen $\mathbf{U} \rightarrow \mathbf{X}$ -ključ na x, T_1 čeka	U : T_1 S : T_2
$w_2[x]$	T_2 : odbijen $\mathbf{S} \rightarrow \mathbf{X}$ -ključ na x, T_2 čeka	U : T_1 S : T_2
Potpuni zastoj: T_1 čeka T_2 , T_2 čeka T_1		

Ključ za izmjenu

- **IBM IDS:** *cursor for update*
 - u trenutku dohvata n-torke kurzorom (FOREACH ... FOR SELECT ... INTO ...), na dohvaćenu n-torku se postavlja U-ključ
 - operacija UPDATE ... WHERE CURRENT OF ... postavljeni U-ključ promovira u X-ključ

Granulacija podataka

- što je, u stvarnosti, element baze podataka x koji se zaključava?
 - n-torka
 - fizička stranica
 - relacija
 - baza podataka
- finija granulacija
- grublja granulacija
- granulacija podataka pri zaključavanju ne utječe na korektnost do sada razmatranih protokola, ali bitno utječe na performanse sustava
 - odabirom finije granulacije uvećava se konkurentnost i troškovi postavljanja ključeva
 - odabirom grublje granulacije smanjuje se konkurentnost i troškovi postavljanja ključeva
- koja je granulacija "najbolja"?
 - ovisi o konkretnim operacijama transakcije

Granulacija podataka

Primjer:

osoba	oib	ime	prez	visina
	56789012345	Ana	Kolar	175
...
	12345678901	Ivica	Horvat	179

} 100 000 n-torki

T_1

```
SELECT ime, prez FROM osoba  
WHERE oib='12345678901';
```

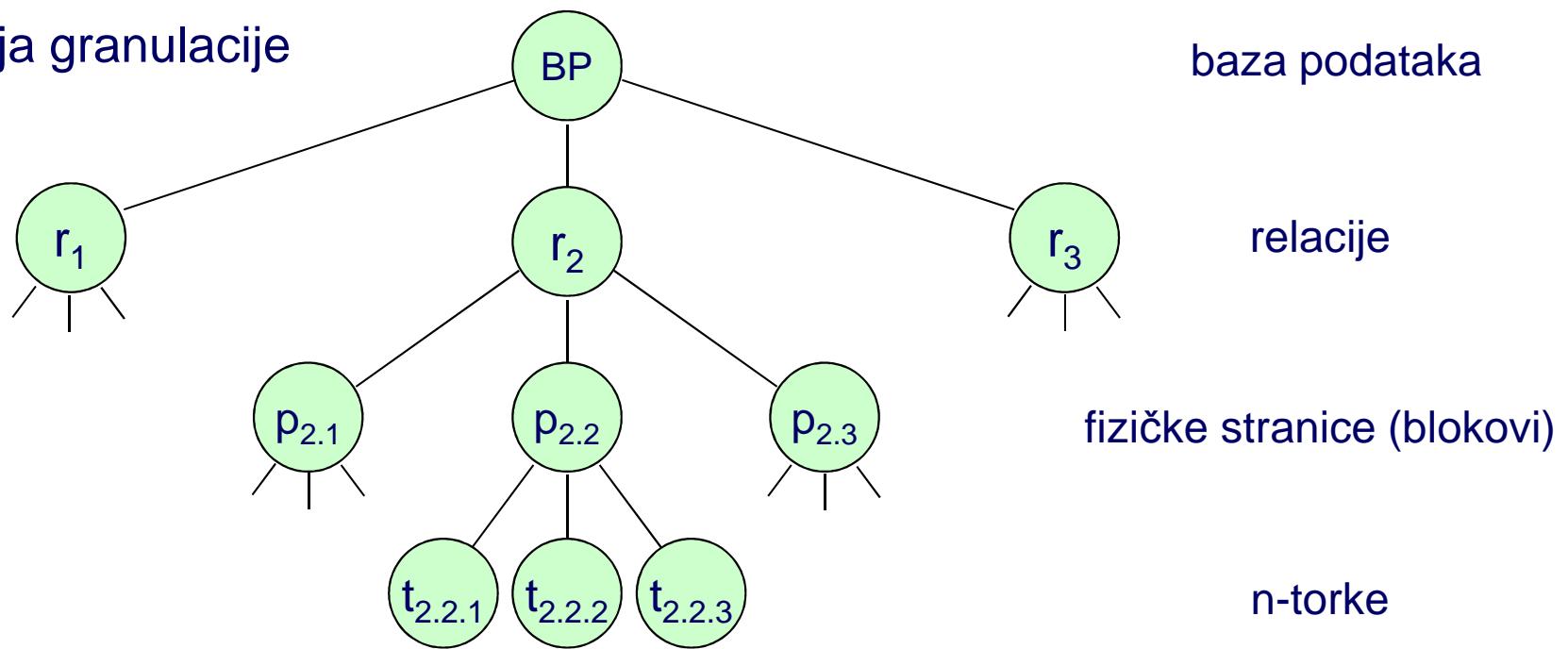
T_2

```
SELECT AVG(visina) FROM osoba;
```

- uz granulaciju određenu relacijom
 - ⇒ slaba konkurentnost, nepotrebno se ograničava pristup svim n-torkama relacije
- koristiti granulaciju određenu n-torkom!
- očito, sustav mora podržavati zaključavanje na više razina granulacije (*multiple granularity locking, MGL*)
- uz granulaciju određenu n-torkom
 - ⇒ loše performanse, pojedinačno se postavlja 100 000 ključeva
- koristiti granulaciju određenu relacijom!

Zaključavanje na više razina granulacije

- hijerarhija granulacije



- eksplisitno se ključ može postaviti na element bilo koje razine
- eksplisitnim zaključavanjem elementa x, implicitno se zaključavaju svi elementi potomci elementa x

Zaključavanje na više razina granulacije

Primjer:

osoba

	oib	ime	prez	visina
	56789012345	Ana	Kolar	175

	12345678901	Ivica	Horvat	179

100 000 n-torki

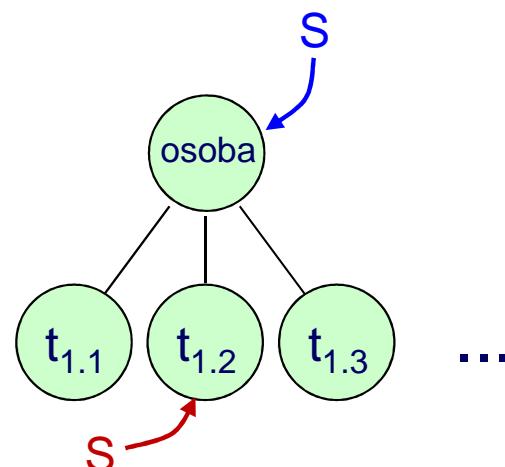
T₁

```
SELECT ime, prez FROM osoba  
WHERE oib='12345678901';
```

T₂

```
SELECT AVG(visina) FROM osoba;
```

- transakcija T₁ eksplicitno će zaključati n-torku relacije osoba
- transakcija T₂ eksplicitno će zaključati relaciju osoba, a sve n-torce relacije će time biti implicitno zaključane



Zaključavanje na više razina granulacije

Primjer:

osoba

	oib	ime	prez	visina
	56789012345	Ana	Kolar	175

	12345678901	Ivica	Horvat	179

} 100 000 n-torki

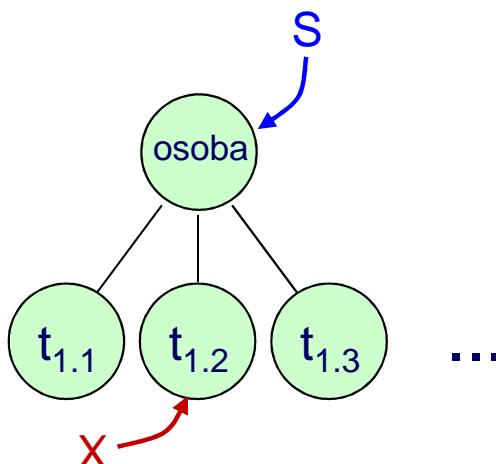
T₁

```
UPDATE osoba SET ime='Ivan'  
WHERE oib='12345678901';
```

T₂

```
SELECT AVG(visina) FROM osoba;
```

- transakcija T₁ eksplicitno će zaključati n-torku
- koje provjere LM mora obaviti prije nego za T₂ postavi S-ključ na relaciju?



Zaključavanje na više razina granulacije

- teškoće u implementaciji:
 - LM prije zaključavanja relacije osoba za transakciju T_2 mora provjeriti kompatibilnost ključa s eventualno već postavljenim ključevima. Provjera se mora obaviti za relaciju, ali također i za sve n-torke (100 000)
 - preveliki broj provjera koje se moraju provesti pri svakom zaključavanju elementa na višoj razini!
 - rješenje je u primjeni nove vrste ključeva - ključeva upozorenja (*warning lock* ili *intention lock*)

Zaključavanje na više razina granulacije

Primjer:

osoba

	oib	ime	prez	visina
	56789012345	Ana	Kolar	175

	12345678901	Ivica	Horvat	179

100 000 n-torki

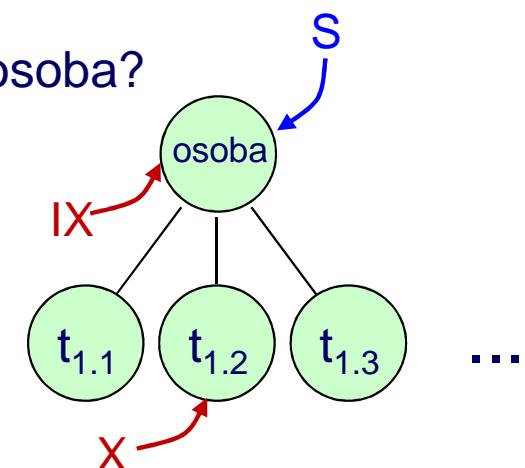
T₁

```
UPDATE osoba SET ime='Ivan'  
WHERE oib='12345678901';
```

T₂

```
SELECT AVG(visina) FROM osoba;
```

- transakcija T₁ na relaciju postavlja ključ namjere ili upozorenja IX koji ostale transakcije upozorava da su na nižim razinama postavljeni "pravi" ekskluzivni ključevi. Tek nakon toga X-ključem zaključava n-torku.
- što T₂ sada mora provjeriti prije zaključavanja relacije osoba?



Ključevi upozorenja

- *intention-shared lock (IS)*
 - IS-ključ na elementu x signalizira da na nižim razinama stabla, u podstablu čiji je korijen x, postoje elementi koji su (ili će uskoro biti) zaključani S-ključem
- *intention-exclusive lock (IX)*
 - IX-ključ na elementu x signalizira da na nižim razinama stabla, u podstablu čiji je korijen x, postoje elementi zaključani X ili S-ključem
- *shared and intention-exclusive lock (SIX)*
 - SIX-ključ je kombinacija S i IX ključeva
 - SIX-ključ na elementu x signalizira da je na x postavljen eksplisitni S-ključ, dok u podstablu čiji je korijen x, postoje elementi zaključani X-ključem
 - npr. transakcija koja čita veliki broj n-torki relacije, a zatim mijenja samo neke od pročitanih n-torki, postavlja SIX-ključ na relaciju
 - postavljeni SIX-ključ je omogućio čitanje svih n-torki relacije ("S dio SIX-ključa") i omogućio postavljanje eksplisitnih X-ključeva na neke n-torke relacije ("IX dio SIX-ključa"). Važno: ovdje se pretpostavilo da se n-torka u hijerarhiji granulacije nalazi **neposredno** ispod relacije
- ne zaboraviti: zaključavanjem nekog elementa S-ključem (ili X-ključem), implicitno su S-ključem (ili X-ključem) zaključani svi elementi u podstablu zaključanog elementa

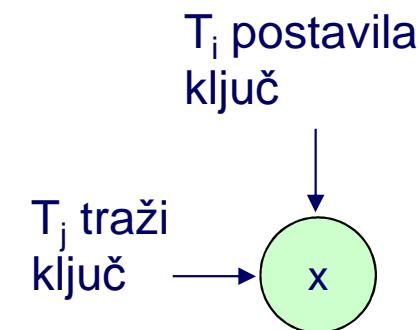
Ključevi upozorenja

Tablica kompatibilnosti ključeva

T_i postavila ključ \rightarrow

T_j traži ključ \downarrow

	S	X	IS	IX	SIX
S	T	⊥	T	⊥	⊥
X	⊥	⊥	⊥	⊥	⊥
IS	T	⊥	T	T	T
IX	⊥	⊥	T	T	⊥
SIX	⊥	⊥	T	⊥	⊥



Ključevi upozorenja

Pravila
promoviranja
ključeva

T_k posjeduje ključ

T_k treba ključ

	S	X	IS	IX	SIX
S	S	X	S	SIX	SIX
X	X	X	X	X	X
IS	S	X	IS	IX	SIX
IX	SIX	X	IX	IX	SIX
SIX	SIX	X	SIX	SIX	SIX

- povećani broj vrsta ključeva rezultira većim brojem mogućnosti promoviranja ključeva
- objašnjenje tablice: transakcija T_k je na element x postavila određenu vrstu ključa određenu stupcem j . Ako transakcija nad istim elementom zatreba novu vrstu ključa, određenu retkom i , tada je elementom tablice s indeksom (i, j) definirana vrsta ključa u koju transakcija mora promovirati postojeći ključ
- npr. ako transakcija nad elementom x posjeduje S-ključ, a potreban joj je IX-ključ, tada postojeći S-ključ mora promovirati u SIX-ključ
 - voditi računa o tome da će promocija ključa biti dopuštena jedino u slučaju kada novi ključ nije u konfliktu s ključevima na elementu x koje su postavile druge transakcije

Protokol zaključavanja na više razina granulacije

Multiple-granularity locking protocol (MGL) ili Warning protocol

MGL₁: T_i može zaključati element x ako je traženi ključ kompatibilan s ključevima koje su na x postavile ostale transakcije (vidjeti tablicu kompatibilnosti ključeva)

MGL₂: zaključavanje mora početi od korijena. Korijen se može zaključati bilo kojom vrstom ključa (ako je zadovoljeno pravilo MGL₁)

MGL₃: T_i može zaključati element x ključem S ili IS jedino ako je T_i prethodno zaključala roditelja od x ključem IS ili IX

MGL₄: T_i može zaključati element x ključem X, IX ili SIX jedino ako je T_i prethodno zaključala roditelja od x ključem IX ili SIX

MGL₅: T_i može zaključati element x jedino ako prije toga nije otključala niti jedan drugi element (sukladno pravilu 2PL₃)

MGL₆: T_i može otključati element x jedino ako niti jedan potomak elementa x nije zaključan od strane T_i

- mogući potpuni zastoji se izbjegavaju ili razrješuju na već opisane načine

Protokol zaključavanja na više razina granulacije

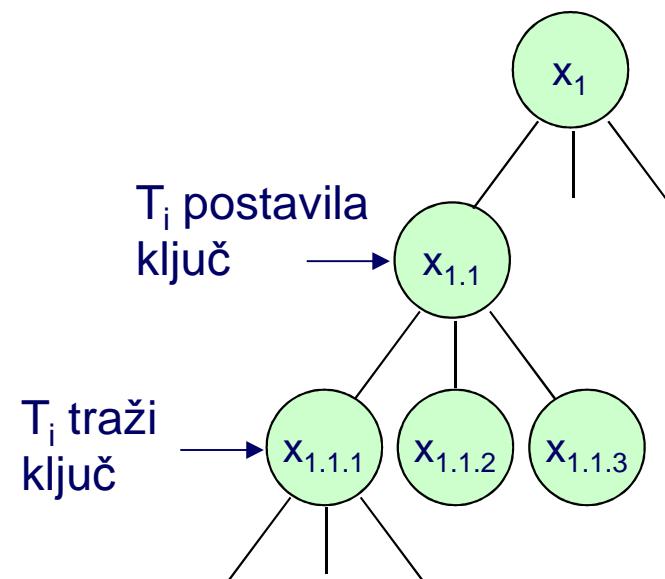
- pravila MGL₃ i MGL₄ se mogu sažeto prikazati sljedećom tablicom:

Vrste ključeva koje **ista transakcija** smije postavljati na elemente koji su **u odnosu roditelj-dijete**

T_i postavila ključ na roditelju →

T_i traži ključ na djetu ↓

	-	IS	IX	SIX
S	⊥	T	T	⊥
X	⊥	⊥	T	T
IS	⊥	T	T	⊥
IX	⊥	⊥	T	T
SIX	⊥	⊥	T	T

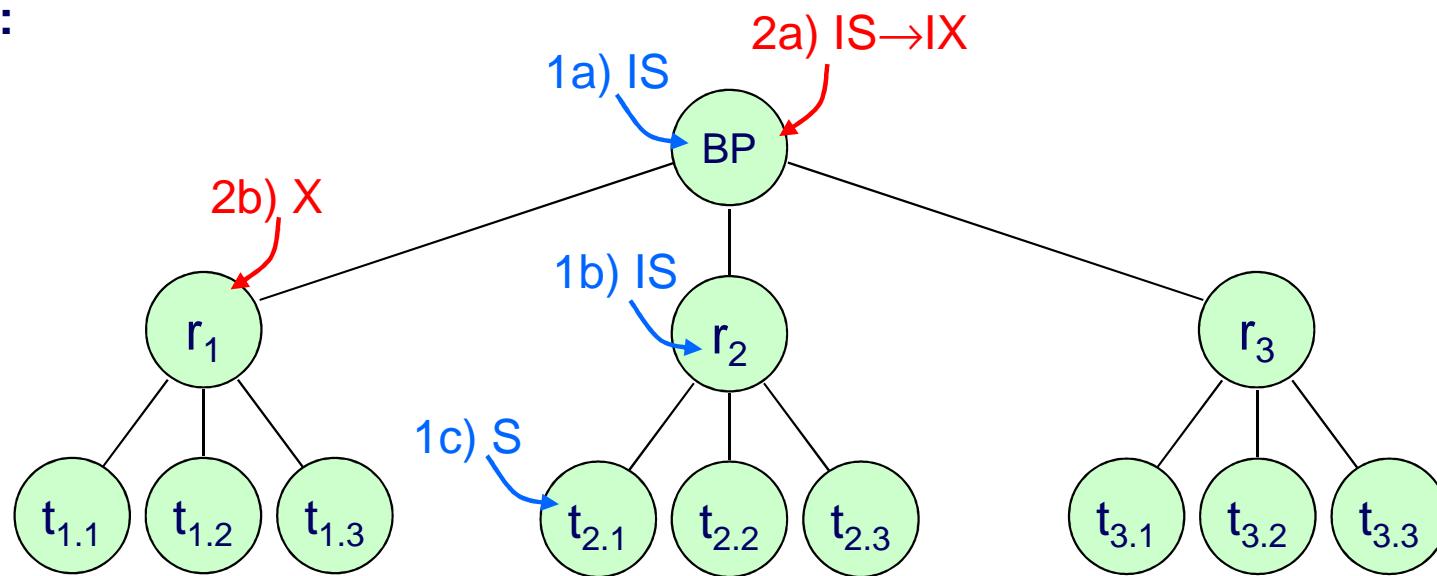


- tablicu ne treba konzultirati kad se zaključava korijen, jer se prema pravilu MGL₂ korijen može zaključati bilo kojom vrstom ključa

- zašto u tablici nema stupaca za S i X?

Protokol zaključavanja na više razina granulacije

Primjer:

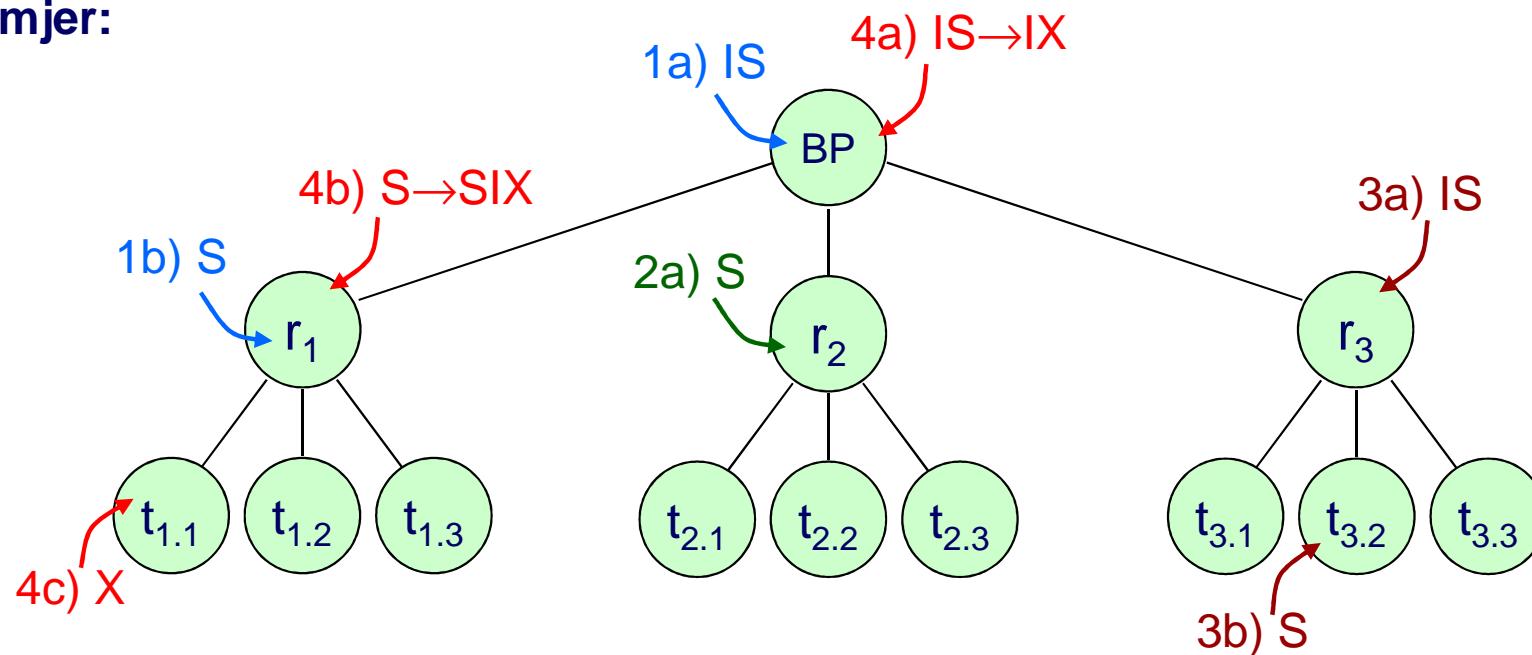


- koje ključeve postavlja T_1 dok obavlja sljedeće operacije?
 1. čitanje n-torke $t_{2.1}$
 2. zatim izmjena svih n-torki u relaciji r_1
- nakon što T_1 zaključa sve potrebne elemente, može li T_2 obaviti operaciju:

1. izmjena n-torke $t_{2.2}$	⇒ da	(koje ključeve je postavila?)
2. čitanje n-torke $t_{1.3}$	⇒ ne	(koje ključeve je postavila?)
3. izmjena n-torke $t_{3.1}$	⇒ da	(koje ključeve je postavila?)

Protokol zaključavanja na više razina granulacije

Primjer:



- koje ključeve postavlja T_1 dok redom obavlja sljedeće operacije?
 1. čitanje svih n-torki relacije r_1
 2. čitanje svih n-torki relacije r_2
 3. čitanje n-torke $t_{3.2}$
 4. izmjena n-torke $t_{1.1}$
- nakon što T_1 zaključa sve potrebne elemente, može li T_2 obaviti operaciju:
 1. čitanja n-torke $t_{1.2}$
 2. izmjene n-torke $t_{2.1}$
 3. izmjene n-torke $t_{3.1}$

Zaključavanje n-torki / blokova / relacija

IBM IDS:

```
CREATE TABLE tab1 (
    ...
) LOCK MODE ROW;
```

```
CREATE TABLE tab2 (
    ...
) LOCK MODE PAGE;
```

Primjer:

```
CREATE TABLE racun (
    brRac INTEGER ...
) LOCK MODE ROW;
```

```
CREATE TABLE stavkaRacuna (
    brRac INTEGER
    rbrStavka ...
) LOCK MODE PAGE;
```

```
LOCK TABLE tab1 IN EXCLUSIVE MODE;
LOCK TABLE tab1 IN SHARE MODE;
```

Operacija brisanja

- do sada se baza podataka promatrala kao "statički skup" (u smislu nepromjenjivosti kardinalnog broja skupa) imenovanih elemenata ili objekata
- osim operacija čitanja i pisanja (izmjene postojećih elemenata baze), SUBP mora podržavati operacije brisanja elemenata i unosa novih elemenata u bazu podataka
- skup operacija koje obavlja transakcija T_i proširuje se operacijama:
 - $del_i[x]$, $ins_i[x]$
- operacija brisanja je u konfliktu sa svim operacijama
 - $r_i[x], del_j[x] \neq del_j[x], r_i[x]$
 - $w_i[x], del_j[x] \neq del_j[x], w_i[x]$
 - $ins_i[x], del_j[x] \neq del_j[x], ins_i[x]$
 - $del_i[x], del_j[x] \neq del_j[x], del_i[x]$
- rješenje: prije operacije brisanja, element zaključati X-ključem
- slično: nakon operacije unosa, element zaključati X-ključem
- **nije dovoljno ako se koriste operacije nad skupovima \Rightarrow SQL!**

Operacija unosa

Primjer:

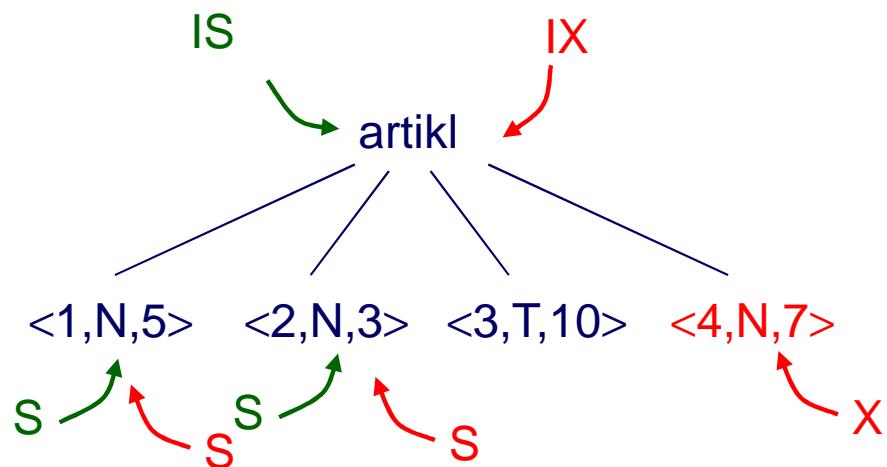
artikl		
sif	grupa	tez
1	N	5
2	N	3
3	T	10

T_1

```
1. SELECT AVG(tez) INTO v  
   FROM artikl  
   WHERE grupa = 'N';  
6. write(x, v);  
7. commit;
```

T_2

```
2. INSERT INTO artikl  
   VALUES (4, 'N', 7);  
3. SELECT AVG(tez) INTO v  
   FROM artikl  
   WHERE grupa = 'N';  
4. write(x, v);  
5. commit;
```



- rezultat: $x = 4$, a trebao je biti $x = 5$
- zašto? Zar 2PL (MGL) ne funkcionira?

Sablasna n-torka

- u prethodnom su primjeru operacije čitanja transakcije T_1 trebale blokirati operaciju unosa koju obavlja transakcija T_2
- T_1 nije postavila S-ključeve na sve n-torce na koje ih je trebala postaviti
 - sve n-torce na koje je trebalo postaviti S-ključeve u tom trenutku nisu postojale
- za transakciju T_1 n-torka $\langle 4, N, 7 \rangle$ predstavlja sablasnu n-torku (*phantom row*)
- to je n-torka koja "ne postoji" u trenutku kada transakcija T_i postavlja ključeve na n-torce koji zadovoljavaju neki predikat P, a transakcija T_j nakon toga unese novu n-torku koja također zadovoljava predikat P
- sablasna n-torka može također biti i posljedica operacije izmjene. U prethodnom primjeru T_2 je mogla "stvoriti" sablasnu n-torku za T_1 tako što bi umjesto operacije unosa $\langle 4, N, 7 \rangle$, obavila operaciju izmjene $\langle 3, T, 10 \rangle \rightarrow \langle 3, N, 10 \rangle$
- operacija brisanja (ako se koristi ST protokol) ne uzrokuje sablasne n-torce za transakciju T_i jer T_j koja bi pokušala brisati element x neće uspjeti postaviti X-ključ na x ako ga je T_i pročitala

Sablasna n-torka

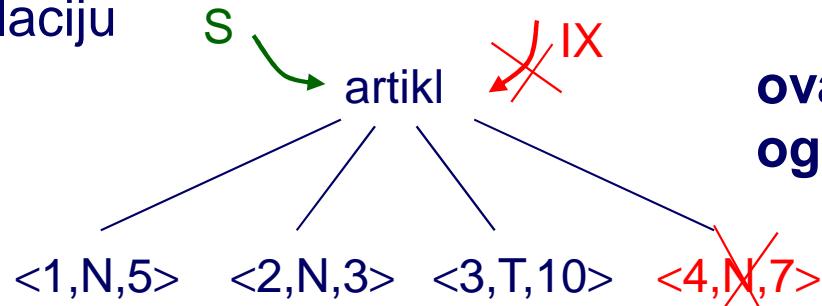
- zašto "2PL ne funkcionira" u ovom slučaju:
 - T_1 je osim čitanja vrijednosti n-torki, također implicitno čitala podatak *koje se sve n-torke nalaze u skupu*. Stoga je trebala postaviti S-ključ na skup. Transakcija T_2 koja mijenja dotični skup (mijenja mu kardinalnost), trebala je na skup (pokušati) postaviti X-ključ. Tada bi T_2 bila blokirana do završetka transakcije T_1
 - nažalost, element *skup svih n-torki koje u relaciji imaju vrijednost atributa grupa='N'* ne postoji u bazi podataka, stoga se ne može zaključati

Sablasna n-torka

- Zaključavati n-torke?
 - konkurentnost je visoka, ali se pojavljuju sablasne n-torke \Rightarrow neserijalizabilne povijesti
- Zaključavati relacije?
 - spriječit će se pojava sablasnih n-torki, ali će se bitno smanjiti konkurentnost
- SUBP zaključava relaciju ako nije u mogućnosti koristiti protokol zaključavanja indeksa

Sablasna n-torka

- sprječavanje pojave sablasnih n-torki zaključavanjem relacije:
 - transakcija T_1 koja dohvaća n-torke prema predikatu grupa='N' postavlja S-ključ na cijelu relaciju
 - transakcija T_2 koja pokušava unijeti n-torku (bilo koju) neće moći postaviti IX-ključ na relaciju



ovakav pristup bitno ograničava konkurentnost

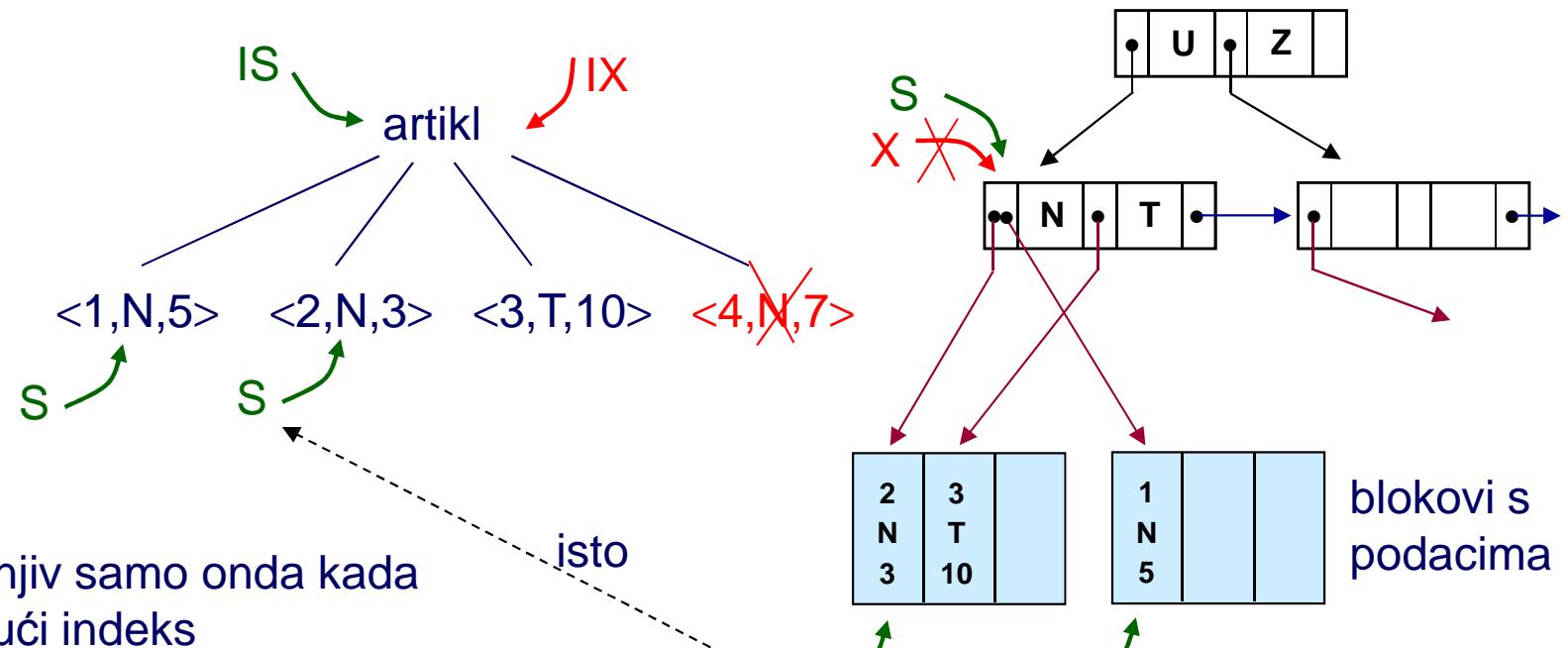
- npr. transakcija T_3 koja ubacuje n-torku $<4,T,7>$ bit će nepotrebno spriječena jer je T_1 nepotrebno zaključala cijelu relaciju
- T_1 je trebala zaključati samo "sadašnje i buduće" n-torke čija je grupa='N'
- kako zaključati "buduće" n-torke čija je grupa='N'
- rješenje: koristiti indeks (B-stablo) \Rightarrow protokol zaključavanja indeksa

T_3

```
INSERT INTO artikel  
VALUES (4, 'T', 7);  
SELECT AVG(tez) INTO v  
FROM artikel  
WHERE grupa = 'T';  
write(y, v);  
commit;
```

Protokol zaključavanja indeksa (*Index-locking protocol*)

- sprječavanje pojave sablasnih n-torki korištenjem indeksa:
 - transakcija T_i koja čita skup n-torki koje zadovoljavaju neki predikat, mora S-ključem zaključati odgovarajuće zapise u listovima B-stabla
 - transakcija T_j kojom se unosi n-torka, mora postaviti X-ključ na zapis u listu B-stabla (ne cijeli list) u kojem će se nalaziti pokazivač na novu n-torku



- protokol je primjenjiv samo onda kada postoji odgovarajući indeks
- ako T_i dohvata n-torce prema atributu za kojeg ne postoji indeks, morat će S-ključem zaključati cijelu relaciju!

Protokol zaključavanja indeksa (*Index-locking protocol*)

- protokol zaključavanja indeksa također pomaže u slučaju brisanja n-torce:

H $xL_1[x]$, $del_1[x]$, $(ins_2[x^{new}], xL_2[x^{new}])$, a_1 , $u_1[x]$, c_2 , $u_2[x^{new}]$

x^{new} ima isti primarni ključ kao x

x^{new} je unesen i
istovremeno zaključan

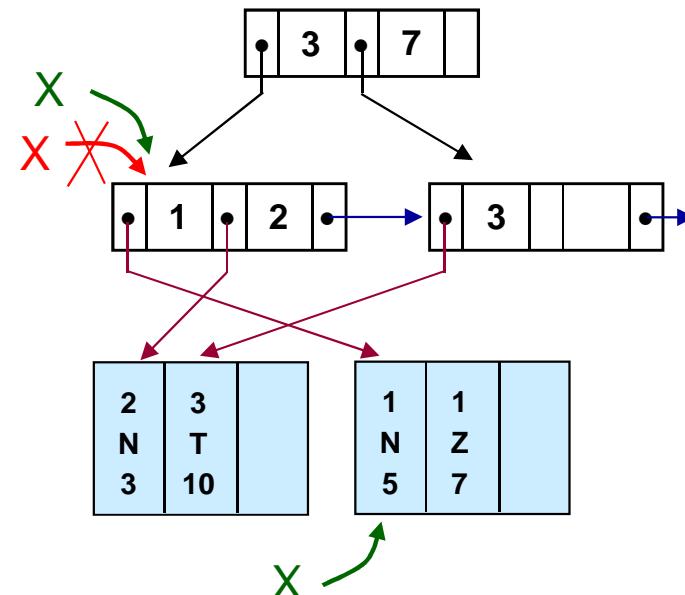
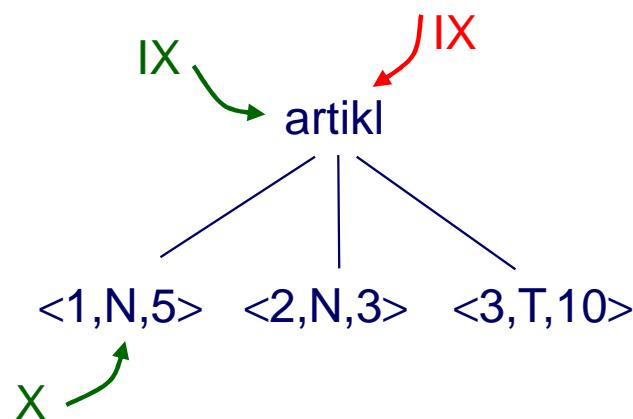
rezultat: dva elementa s
istim primarnim ključem.
Pomaže protokol
zaključavanja indeksa

T_1

```
1. DELETE FROM artikel  
   WHERE sif = 1;  
3. abort;
```

T_2

```
2. INSERT INTO artikel  
   VALUES (1, 'Z', 7);  
4. commit;
```



- T_2 neće moći postaviti X-ključ na list indeksa dok T_1 ne završi

Slabe razine konzistentnosti

- *weak levels of consistency*
- u sustavima koji koriste 2PL protokol konkurentnost se povećava, uz istodobno smanjenje garancije konzistentnosti, tako što se neka pravila 2PL protokola vezana uz postavljanje S-ključeva* ne primjenjuju striktno
- dok se dio transakcija izvršava serijalizabilno (čitaju isključivo konzistentno stanje baze podataka), nekim je transakcijama dopušteno čitati (moguće) tranzijentno nekonzistentno stanje baze podataka
 - u sustavu se istovremeno mogu izvršavati transakcije s različitim razinama izolacije
- korištenjem slabih razina konzistentnosti moguće je (uz oprez!) povećati konkurentnost uz zadržavanje zadovoljavajuće razine konzistentnosti

* pravila koja se odnose na X-ključeve se uvijek striktno primjenjuju (X-ključevi se otpuštaju tek na kraju transakcije) jer se u suprotnom ugrožava mogućnost obnove uz pomoć predslika iz dnevnika izmjena (ST-povijest)

Slabe razine konzistentnosti

- podsjetnik na ST povijest
 - transakcija T_i niti jedan element ne smije:
 - niti pročitati, niti izmijenitidok sve transakcije T_j koje su prije toga pisale u taj element ne terminiraju (obave *abort* ili *commit*)

Prljavo pisanje (*dirty write*): SUBP to NIKAD ne dopušta

$H \quad w_2[x] \dots w_1[x] \dots a_2 \dots c_1$

- prikazana povijest koja nije ST zbog *prljavog pisanja*
- ako bi se dopustilo prljavo pisanje, dnevnikom izmjena ne bi se uvijek moglo ispravno poništiti efekte transakcije. SUBP **nikad** ne dopušta prljavo pisanje, bez obzira koja se razina izolacije primjenjuje

Prljavo čitanje (*dirty read*): dopušta se, uz postavljen zahtjev SUBP-u

- transakciji je dopušteno čitati iz nepotvrđenih transakcija
- kako se postiže: SUBP za operacije čitanja dotične transakcije ne postavlja S-ključeve. Rezultat je smanjena razina konzistentnosti

Slabe razine konzistentnosti

- ANSI SQL definira anomalije koji se mogu pojaviti tijekom izvršavanja neserijalizabilnih povijesti koje su posljedica slabih razina konzistentnosti
 - prljavo čitanje
 - neponovljivo čitanje
 - sablasne n-torke
- **Prljavo čitanje (*Dirty read*)**
 - transakcija T_1 obavi operaciju čitanja elementa x prije nego je transakcija T_2 koja je pisala u element x terminirala
 $H \quad w_2[x] \dots r_1[x] \dots a_2 \dots c_1$
 - povijest koja nije ST zbog *prljavog čitanja*
 - odgovara karakterističnom problemu istodobnog pristupa opisanom na početku predavanja - prljavo čitanje

Slabe razine konzistentnosti

- oprez!
- transakcije koje pročitaju tranzijentno nekonzistentno stanje baze mogu korisniku prikazati neispravan rezultat ili, ako pri tome mijenjaju sadržaj baze podataka, čak uzrokovati trajnu nekonzistentnost baze podataka. Trajnu nekonzistentnost onda dalje mogu propagirati čak i transakcije koje se izvršavaju serijalizabilno
 - procjena o tome koje se transakcije smiju izvršavati uz smanjenu razinu konzistentnosti je odgovornost programera/dizajnera aplikacije
 - na temelju analize transakcije i značenja podataka ocjenjuje se da li će se transakcija ispravno obaviti i uz slabiju razinu konzistentnosti
- *at every isolation level lower than SERIALIZABLE, bad things can happen!*

Slabe razine konzistentnosti

- **Neponovljivo čitanje (*Non-repeatable read*)**
 - transakcija T_1 pročita element x , T_2 promijeni x . Ako T_1 ponovo pročita x , pročitat će drugačiju vrijednost nego u prvom čitanju

H $r_1[x] \dots w_2[x] \dots r_1[x] \dots c_1$
- anomalija je povezana sa sljedećim karakterističnim problemima istodobnog pristupa: nekonzistentna analiza i izgubljena izmjena
- nekonzistentna analiza
 - vidjeti primjer nekonzistentne analize s početka predavanja: kada bi transakcija T_1 ponovno pročitala element x , npr. nakon operacije *read(y, p2)*, otkrila bi da je element x u međuvremenu promijenio vrijednost \Rightarrow obavljeno je neponovljivo čitanje
- izgubljena izmjena
 - vidjeti primjer izgubljene izmjene s početka predavanja: kada bi transakcija T_1 ponovno pročitala element x prije nego obavi operaciju *write(x, p)*, otkrila bi da je element x u međuvremenu promijenio vrijednost \Rightarrow obavljeno je neponovljivo čitanje

Slabe razine konzistentnosti

- **Sablasne n-torke (*Phantom rows*)**

- transakcija T_1 pročita skup podataka koji zadovoljava predikat P . Transakcija T_2 unese ili obriše* element koji zadovoljava predikat P . Ako T_1 ponovo pročita skup podataka, dobit će drugačiji rezultat

$$H \quad r_1[\{ x \mid P(x) \}] \dots \text{ins}_2[x \mid P(x)] \dots r_1[\{ x \mid P(x) \}] \dots c_1$$

* ili izmijeni neki od elemenata tako da element koji je zadovoljavao predikat P prestane zadovoljavati P , ili element koji nije zadovoljavao P počne zadovoljavati P

- sablasne n-torke: ponovnim izvršavanjem iste SELECT naredbe (u istoj transakciji) dobije se skup u kojem se nalaze neke druge n-torke
- neponovljivo čitanje: ponovnim izvršavanjem iste SELECT naredbe (u istoj transakciji) dobije se skup u kojem se nalaze iste n-torke, ali s drugačijim vrijednostima atributa

ANSI SQL razine izolacije

- na temelju tih anomalija, ANSI SQL definira četiri razine izolacije. Svakom višom razinom izolacije osigurava se izbjegavanje većeg broja anomalija

A n o m a l i j e

Razine izolacije	Prljavo čitanje	Neponovljivo čitanje	Sablasne n-torke
READ UNCOMMITTED	DA	DA	DA
READ COMMITTED	NE	DA	DA
REPEATABLE READ	NE	NE	DA
SERIALIZABLE	NE	NE	NE

ANSI SQL razine izolacije

- jedan od načina kako relacijski SUBP koji koristi 2PL protokol može implementirati ANSI SQL razine izolacije:
- READ UNCOMMITTED
 - operacija $r[x]$ ne postavlja S-ključ na x niti provjerava je li x zaključan X-ključem
- READ COMMITTED
 - operacija $r[x]$ postavlja kratkotrajni (*short duration*) S-ključ na x samo u trenutku obavljanja operacije i odmah ga otpušta (namjerno narušavanje 2PL₃)
- REPEATABLE READ
 - operacija $r[x]$ postavlja uobičajeni dugotrajni (*long duration*) S-ključ na x u skladu sa striktnim 2PL protokolom, ali ne koristi protokol zaključavanja indeksa (niti zaključava cijelu relaciju) u cilju izbjegavanja sablasnih n-torki
- SERIALIZABLE
 - provodi se striktni 2PL protokol i protokol zaključavanja indeksa (ili ako odgovarajući indeks ne postoji, zaključava se cijela relacija)

READ UNCOMMITTED

- potencijalno područje primjene:
 - čitanje relacija koje se vrlo rijetko mijenjaju (npr. katalog županija)
 - podaci koji se mogu pročitati su sigurno potvrđeni i sigurno neće biti promijenjeni zbog pravila poslovnih procesa koji se primjenjuju (vidjeti primjer)
 - čitanje kod kojeg je pogreška rezultata zbog eventualne interferencije s drugim transakcijama zanemariva:
 - npr. *ad-hoc* upit nad relacijom **ispit** koja sadrži 500000 n-torki: kolika je prosječna ocjena pozitivno ocijenjenih ispita u posljednjih 10 godina
 - jednokorisnički režim rada (npr. izvršavanje *batch* transakcija noću)

Primjer:

- ispis Dopunske isprave o studiju (Supplement)
 - student je diplomirao!
 - nastavni program (popis predmeta) i ocjene - smije li transakcija čitati te podatke uz razinu izolacije *read uncommitted* ?

READ COMMITTED

- potencijalno područje primjene:
 - onda kada bi čitanje nepotvrđenog podatka moglo uzrokovati nekonzistentnost, ali izmjena podatka koju bi unutar kratkog vremena nakon čitanja obavila neka druga transakcija nije bitna (ili čak nije moguća)

Primjer:

- T_1 uvećava ukupni broj operacija koje je stroj obavio (ukupni broj operacija stroja može se samo povećavati)
- T_2 u relaciju *potrebnaZamjena* dodaje identifikator stroja ako je stroj obavio 1000 ili više operacija
- kada bi se T_2 izvršavala uz razinu izolacije UNCOMMITTED, mogla bi na temelju nepotvrđene izmjene transakcije T_1 upisati stroj u relaciju *potrebnaZamjena*
- za T_2 je razina izolacije COMMITTED dovoljna, budući da stroju koji je (potvrđeno) dosegao 1000 operacija, broj obavljenih operacija sigurno neće biti umanjen

aktivnostStroja

idStroja	brojOperacija
150	1002
151	981

potrebnaZamjena

idStroja
150

REPEATABLE READ i SERIALIZABLE

REPEATABLE READ

- potencijalno područje primjene:
 - kada postoji opasnost od anomalija:
 - izgubljena izmjena
 - nekonzistentna analiza
 - (vidjeti primjere s početka predavanja)

SERIALIZABLE

- potencijalno područje primjene
 - kada se unosom ili izmjenom n-torke utječe na njezinu pripadnost skupu n-torki koje zadovoljavaju neki predikat koji je važan u opisu nekog integritetskog ograničenja
 - (vidjeti primjere u poglavlju o sablasnoj n-torci)
- izazov za dizajnera aplikacije
 - primijeniti najnižu moguću razinu izolacije koja će osigurati konzistentnost baze podataka

ANSI SQL razine izolacije

- voditi računa o tome da konkretna implementacija SUBP-a može biti nekonzistentna s ANSI SQL definicijom razine izolacije

	IBM IDS (ANSI mod)	IBM IDS	Oracle 10g
READ UNCOMMITTED	ISOLATION LEVEL READ UNCOMMITTED	SET ISOLATION TO DIRTY READ	-
READ COMMITTED	ISOLATION LEVEL READ COMMITTED	SET ISOLATION TO COMMITTED READ	ISOLATION LEVEL READ COMMITTED
REPEATABLE READ	-	-	-
SERIALIZABLE	ISOLATION LEVEL SERIALIZABLE	SET ISOLATION TO REPEATABLE READ	ISOLATION LEVEL SERIALIZABLE *
dodatane razine izolacije		CURSOR STABILITY COMMITTED READ LAST COMMITTED	CURSOR STABILITY

- * Oracle koristi oblik MV protokola kojim se umjesto *serializable* postiže *snapshot isolation level*, koji u nekim slučajevima dopušta neserijalizabilno izvršavanje (dakle, mogućnost narušavanja konzistentnosti baze podataka). Dobitak je na strani konkurentnosti, koja je povećana u odnosu na onu koja je moguća uz npr. razinu izolacije *SERIALIZABLE* u sustavu IBM IDS

Isolation level SERIALIZABLE ? Oracle 10g / IBM IDS

Primjer: racun
za Oracle 10g

sifRac	iznos
1	70
2	50

- integritetsko ograničenje: suma iznosa na računima 1 i 2 mora biti veća od nule

T₁

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
1. SELECT SUM(iznos) INTO v
   FROM racun
   WHERE sifRac IN (1, 2);
2. IF (v > 80) THEN
   UPDATE racun SET iznos=iznos-80
   WHERE sifRac = 1;
5. COMMIT;
```

T₂

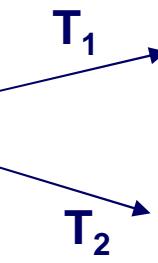
```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
3. SELECT SUM(iznos) INTO v
   FROM racun
   WHERE sifRac IN (1, 2);
4. IF (v > 90) THEN
   UPDATE racun SET iznos=iznos-90
   WHERE sifRac = 2;
6. COMMIT;
```

- T₂ čita konzistentnu verziju (*snapshot*) baze podataka, kakva je bila u trenutku (ili prije trenutka) kada se T₂ počela izvršavati (v = 120)

T₁ i T₂ su korektne!

konzistentno stanje

sifRac	iznos
1	70
2	50



konzistentno stanje

sifRac	iznos
1	-10
2	50

sifRac	iznos
1	70
2	-40

konzistentno stanje

nekonzistentno stanje

sifRac	iznos
1	-10
2	-40

ovaj dio slike prikazuje što T₁ i T₂ "vide" neposredno prije potvrđivanja

Isolation level SERIALIZABLE ? Oracle 10g / IBM IDS

- u prethodnom primjeru IBM IDS bi odgodio ili poništio T_2 i tako osigurao serijalizabilno izvršavanje (striktni 2PL protokol)
- međutim, u nekim slučajevima Oracle 10g omogućava viši stupanj konkurentnosti

Primjer: racun

	sifRac	iznos
	1	350
	2	150

za Oracle 10g

T_1

```
1. UPDATE racun SET iznos=iznos-100  
   WHERE sifRac = 1;  
3. UPDATE racun SET iznos=iznos+100  
   WHERE sifRac = 2;  
4. COMMIT;
```

T_2

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;  
2. SELECT SUM(iznos)  
   FROM racun;  
5. COMMIT;
```

- uz *snapshot isolation* (npr. Oracle 10g) T_2 uvijek izračunava ispravnu sumu: sumu dobivenu na temelju posljednjeg konzistentnog stanja u kojem je bila baza podataka prije nego je T_2 započela
- sustav koji koristi striktni 2PL (npr. IBM IDS) poništava ili odgađa T_2 (smanjena konkurentnost)
 - korištenje READ UNCOMMITTED razine izolacije bi moglo rezultirati izračunom neispravne sume

Ostale SQL razine izolacije

- neki sustavi nemaju implementirane sve razine izolacije koje su definirane SQL standardom
- neki sustavi koriste pogrešne nazine za razine izolacije
- neki sustavi imaju implementirane dodatne razine izolacije
 - Oracle: CURSOR STABILITY
 - Informix: CURSOR STABILITY, COMMITTED READ LAST COMMITTED
 - Microsoft SQL server: SNAPSHOT

Protokol vremenskih oznaka

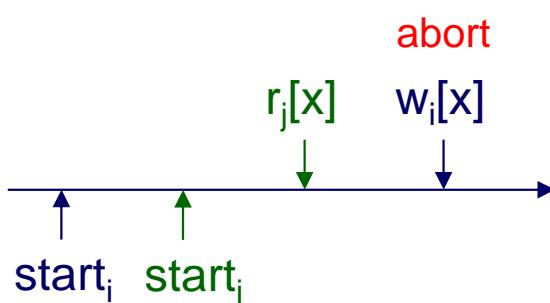
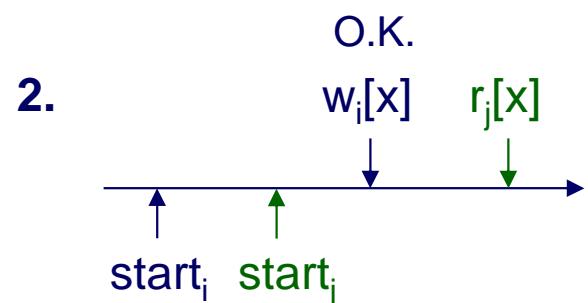
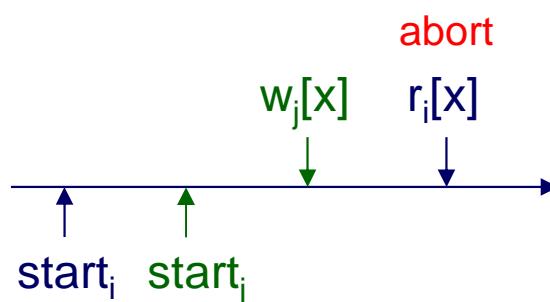
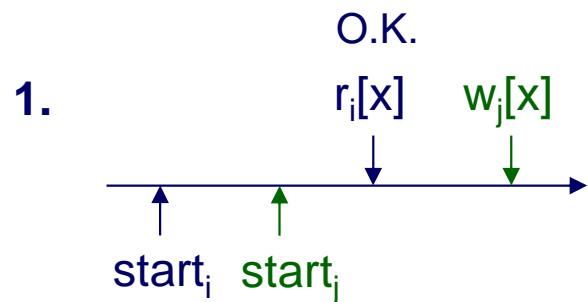
- *timestamp-ordering protocol (TO protocol)*
- transakcija na početku izvršavanja dobiva jedinstvenu vremensku oznaku (*timestamp*)
 - sistemski sat (*system clock*) ili generator monotono rastućeg niza brojeva
 - ako starija transakcija T_i ima vremensku oznaku $TS(T_i)$ tada novija transakcija T_j mora imati oznaku $TS(T_j)$ za koju vrijedi $TS(T_i) < TS(T_j)$
- protokol vremenskih oznaka osigurava da povijest za svake dvije transakcije za koje vrijedi $TS(T_i) < TS(T_j)$ bude ekvivalentna serijskoj povijesti u kojoj je $T_i < T_j$
 - iako će se pojedine operacije transakcije T izvršavati u nekom rasponu vremena, treba osigurati da rezultat bude ekvivalentan onom koji bi se dobio kada bi se sve operacije transakcije T izvršile odjednom, u trenutku $TS(T)$
- za svaki element baze podataka x bilježe se:
 - $WTS(x)$: najveća vrijednost vremenske oznake transakcije koja je uspješno obavila operaciju $w[x]$
 - $RTS(x)$: najveća vrijednost vremenske oznake transakcije koja je uspješno obavila operaciju $r[x]$

Protokol vremenskih oznaka

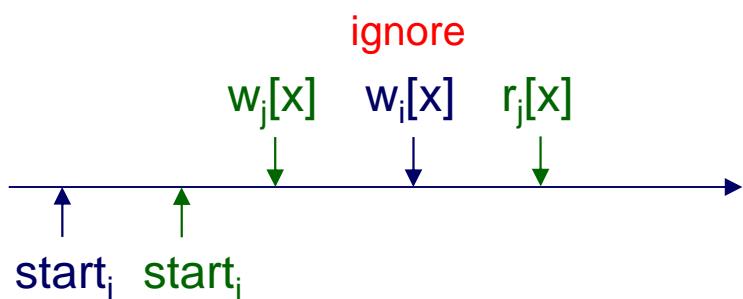
Pravila za TO protokol

1. po zaprimanju zahtjeva za obavljanje $r_i[x]$
 - ako je $TS(T_i) \geq WTS(x)$
 - obavi $r_i[x]$
 - $RTS(x)$ postavi na $\max(RTS(x), TS(T_i))$
 - inače
 - poništi T_i (*jer T_i je zakasnila*)
2. po zaprimanju zahtjeva za obavljanje $w_i[x]$
 - ako je $TS(T_i) < RTS(x)$
 - poništi T_i (*jer T_i je zakasnila*)
 - inače ako je $TS(T_i) < WTS(x)$
 - ignoriraj $w_i[x]$ (*Thomas' write rule: u ovom slučaju ne treba obaviti operaciju, jednostavno nastaviti sa sljedećim operacijama T_i*)
 - inače
 - obavi $w_i[x]$
 - $WTS(x)$ postavi na $TS(T_i)$

Protokol vremenskih oznaka



Thomas' write rule



Protokol vremenskih oznaka

- Nedostaci u odnosu na 2PL:
 - prostor i vrijeme potrebno za vođenje TS oznaka za svaki element
 - protokol je potrebno proširiti dodatnim pravilima kako bi se osiguralo ACA-svojstvo i onemogućilo prljavo čitanje
 - npr. uz svaki element baze podataka vodi se informacija o tome je li posljednja transakcija koja ga je mijenjala terminirala. Transakcija T_i čija je vremenska oznaka takva da bi mogla obaviti operaciju nad elementom x , ali je nad elementom x posljednju izmjenu obavila transakcija T_j koja još nije terminirala, mora čekati dok transakcija T_j ne završi
- Prednost u odnosu na 2PL:
 - potpuni zastoj se ne može pojaviti
 - transakcija nikad ne čeka na zaključavanje, već se u slučaju pokušaja obavljanja konfliktne operacije odmah poništava
 - ali je zbog toga moguće izgladnjivanje transakcija
- postoje povijesti koje je moguće producirati 2PL protokolom, a nije ih moguće producirati TO protokolom. Vrijedi i obratno!

Protokol vremenskih oznaka

Primjer:

$H \ r_2[x], w_3[x], c_3, w_1[y], c_1, r_2[z], w_2[z], c_2$

- povijest H se može producirati pomoću TO protokola

$$\begin{aligned} TS(T_1) &= 3 \\ TS(T_2) &= 1 \\ TS(T_3) &= 2 \end{aligned}$$

TS	op	RTS(x)	WTS(x)	RTS(y)	WTS(y)	RTS(z)	WTS(z)
0		< 1	< 1	< 1	< 1	< 1	< 1
1	$r_2[x]$	1	< 1	< 1	< 1	< 1	< 1
2	$w_3[x]$	1	2	< 1	< 1	< 1	< 1
3	$w_1[y]$	1	2	< 1	3	< 1	< 1
4	$r_2[z]$	1	2	< 1	3	1	< 1
5	$w_2[z]$	1	2	< 1	3	1	1

- povijest H se ne može producirati pomoću 2PL protokola

odgađa se $w_3[x]$

$sL_2[x], r_2[x], \cancel{xL_3[x]}$

Protokol vremenskih oznaka

Primjer:

$H \ r_1[y], r_2[x], w_2[x], c_2, r_1[x], w_1[x], c_1$

- povijest H se ne može producirati pomoću TO protokola

$$\begin{aligned} TS(T_1) &= 1 \\ TS(T_2) &= 2 \end{aligned}$$

TS	op	RTS(x)	WTS(x)	RTS(y)	WTS(y)
0		< 1	< 1	< 1	< 1
1	$r_1[y]$	< 1	< 1	1	< 1
2	$r_2[x]$	2	< 1	1	< 1
3	$w_2[x]$	2	2	1	< 1
4	$r_1[x]$	$TS(T_1) < WTS(x) \Rightarrow \text{abort } T_1$			

- povijest H se može producirati pomoću 2PL protokola

$sL_1[y], r_1[y], sL_2[x], r_2[x], xL_2[x], w_2[x], u_2[x], c_2, sL_1[x], r_1[x], xL_1[x], w_1[x], u_1[x], c_1$

Literatura:

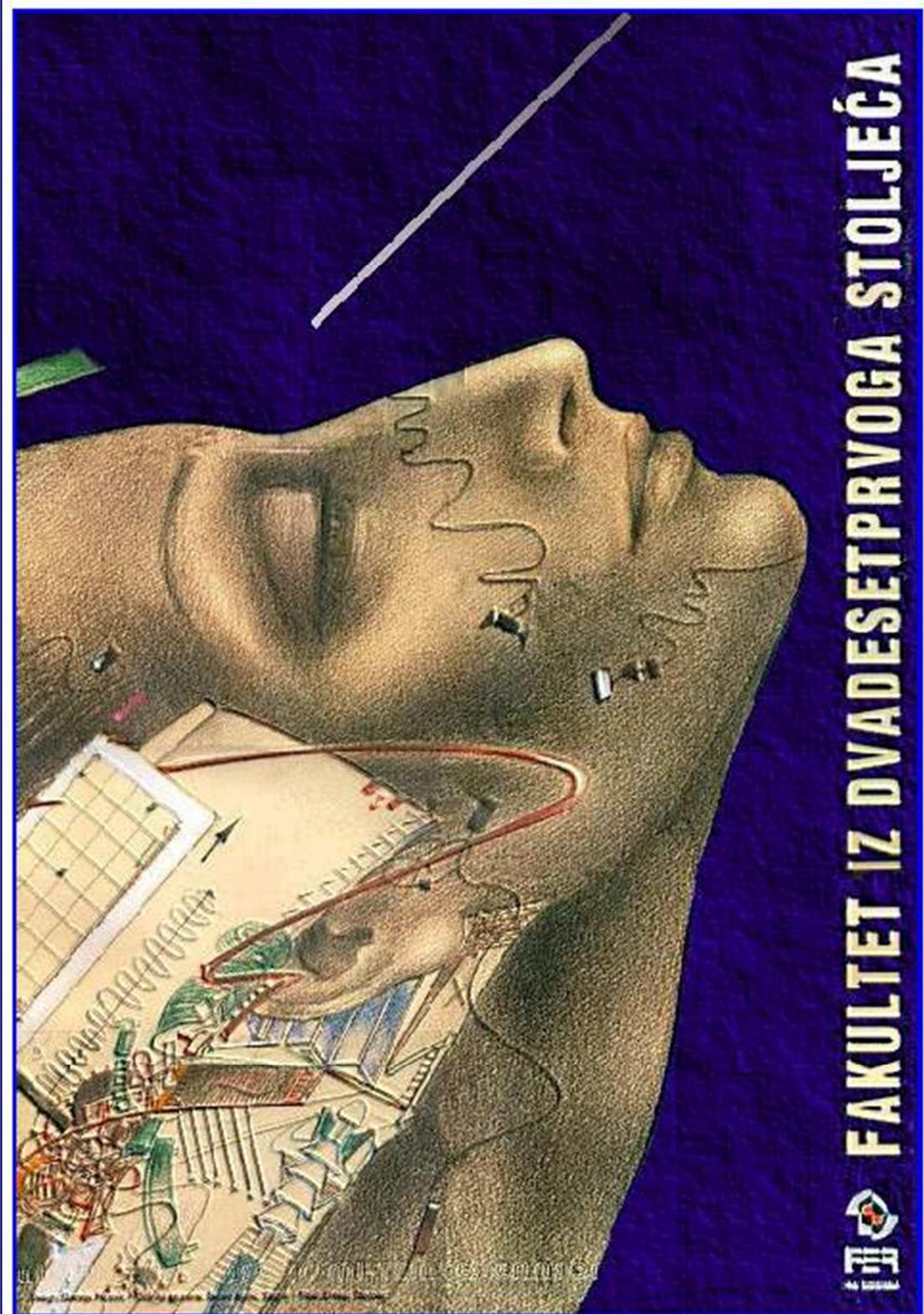
- P. Bernstein, V. Hadzilacos, N. Goodman. Concurrency control and recovery in database systems. Addison-Wesley Publishing Company. Reading, Massachusetts. 1987.
- H. Garcia-Molina, J. Ullman, J. Widom. Database system implementation. Prentice-Hall International, Inc. Upper Saddle River, New Jersey. 2000.
- A. Silberschatz, H. Korth, S. Sudarshan. Database system concepts, 4th ed. McGraw-Hill. 2002.
- R. Elmasri, S. Navathe. Fundamentals of database systems, 3rd ed. Addison-Wesley Publishing Company. Reading, Massachusetts. 2000.
- H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, P. O'Neil. A critique of ANSI SQL isolation levels. Proceedings of the 1995 ACM SIGMOD Int. conf. on Management of Data. San Jose, California. 1995.
- IBM Informix Guide to SQL: Reference, Version 11.1, IBM
- Oracle Database Concepts, 10g Release 2, Oracle
- Oracle Database SQL Reference, 10g Release 2, Oracle

Sustavi baza podataka

Predavanja

11. Distribuirane baze podataka

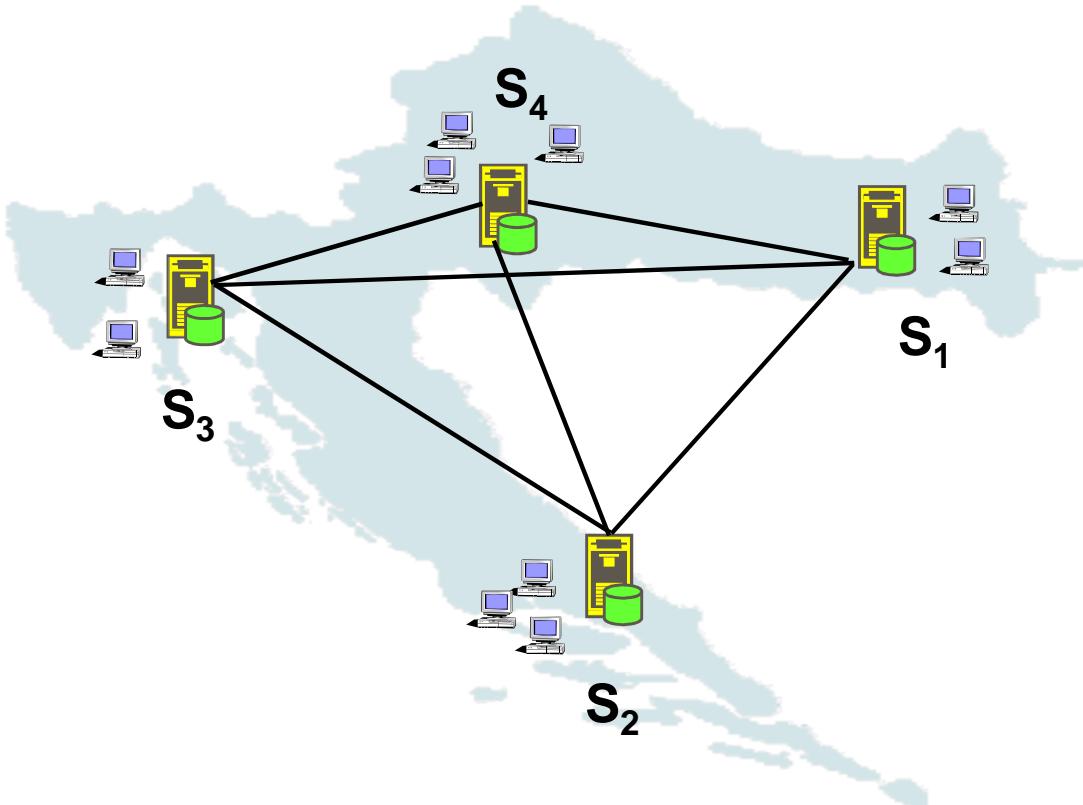
lipanj 2014.



FAKULTET IZ DVADESETPRVOGA STOLJEĆA

Distribuirana baza podataka i distribuirani SUBP

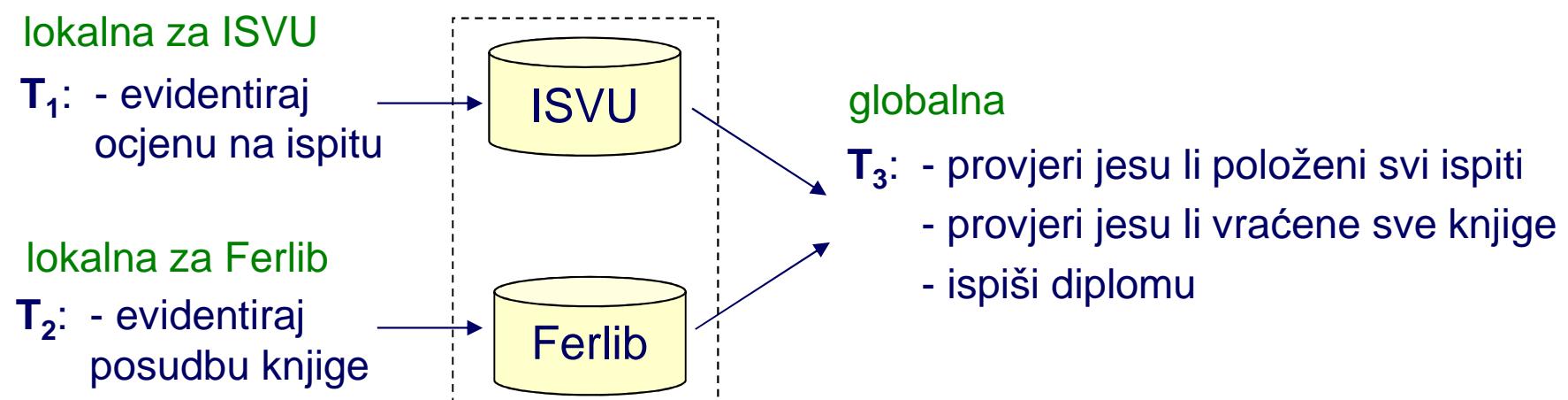
- **distribuirana baza podataka** (DBP) je skup logički povezanih baza podataka razmještenih **u različitim čvorovima** računalne mreže (LAN, MAN, WAN)
- **distribuirani sustav za upravljanje bazama podataka** (DSUBP) je programski sustav koji upravlja distribuiranom bazom podataka na takav način da je distribuiranost sustava transparentna prema korisnicima



- DSUBP obuhvaća n lokalnih SUBP. Svaki lokalni SUBP, označen sa S_i , ($i = 1, \dots, n$) predstavlja jedan čvor (*site, node*) distribuiranog sustava
- svaki čvor S_i može direktno ili indirektno komunicirati sa svakim čvorom S_j , tj. postoji dvosmjerna veza između svaka dva čvora
- čvorovi distribuiranog sustava za upravljanje bazama podataka ne dijele zajedničke fizičke komponente (disk, memorija, procesor)

Distribuirana baza podataka i distribuirani SUBP

- čvorovi su sposobni obavljati transakcije koje zahtijevaju isključivo lokalni pristup podacima (lokalne transakcije), ali također i transakcije koje zahtijevaju pristup podacima iz različitih čvorova (globalne transakcije)
 - čvorovi posjeduju određeni stupanj lokalne autonomije
- lokalne aplikacije (transakcije)
- globalne aplikacije (transakcije)
- baza podataka je distribuirana ako podržava barem jednu globalnu aplikaciju



Prednosti DSUBP u odnosu na centralizirani SUBP

- prilagodljivost organizacijskoj strukturi (organizacije su po prirodi "distribuirane")
 - decentralizacija javne uprave
 - elektronička trgovina
 - bankarski i telekomunikacijski sustavi
 - sustavi za upravljanje proizvodnjom, ...
- snižavanje komunikacijskih troškova
 - približavanje podataka mjestu njihovog nastanka i korištenja
- moguće povećanje dostupnosti podataka
 - kvar jednog čvora ne znači prestanak funkciranja cijelog sustava (u centraliziranom sustavu postoji *single-point-of-failure*)
- moguće povećanje performansi
 - raspodjela opterećenja (*transaction load*) u više čvorova
 - paralelno obavljanje dijelova istog upita u nekoliko čvorova
- omogućavanje modularnog razvoja sustava
 - dodavanje novih čvorova: održiv porast veličine BP, uvećanje procesorske snage

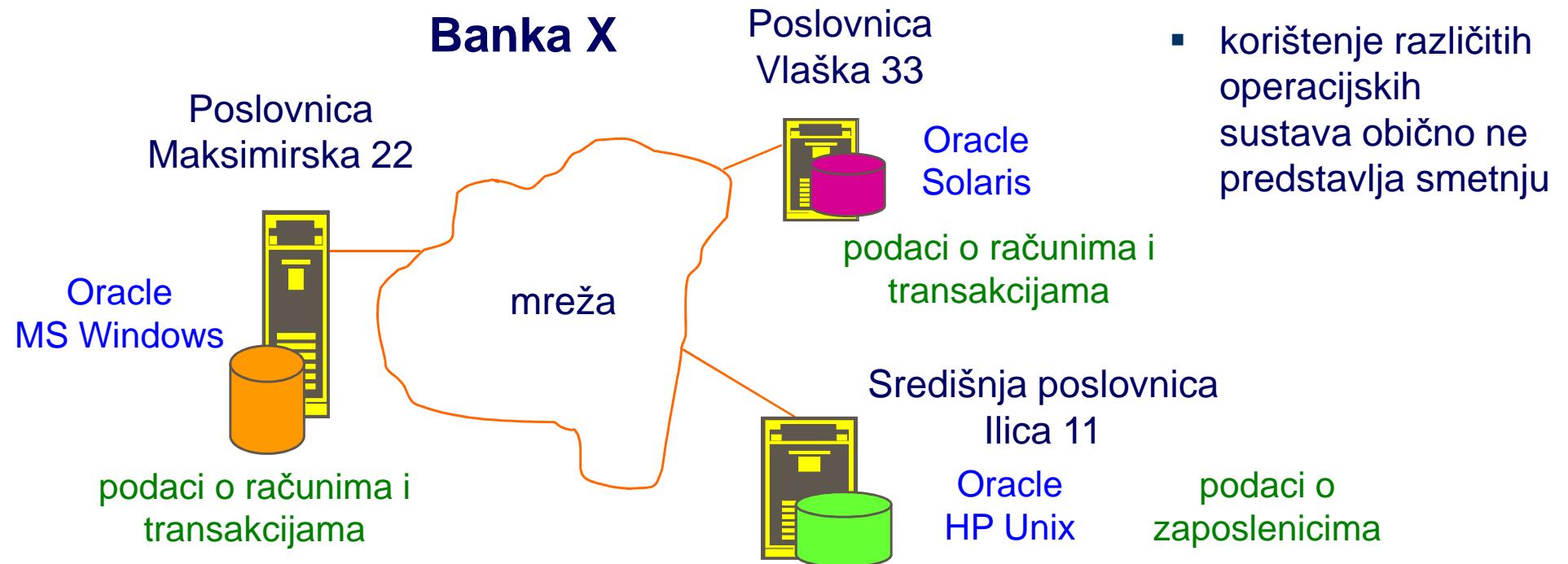
Nedostaci DSUBP u odnosu na centralizirani SUBP

- bitno veća složenost sustava
- povećanje troškova, npr.
 - skuplja programska podrška
 - potreba angažiranja većeg broja administratora sustava
- veći problemi sigurnosti
- veći troškovi u osiguravanju integriteta podataka
- nedostatak standarda
- nedostatak iskustva
- povećanje složenosti postupka projektiranja baze podataka
- loša implementacija distribuirane baze podataka može uzrokovati
 - povećanje komunikacijskih troškova
 - smanjenje dostupnosti podataka
 - smanjenje performansi

Funkcionalnost i tehnike DSUBP, koje su rezultat mnogobrojnih provedenih istraživanja, nisu u cijelosti implementirane niti u jednom danas raspoloživom komercijalnom sustavu.

Homogeni DSUBP

- svi čvorovi koriste iste ili slične verzije istog SUBP-a
 - znatno pojednostavljenje razvoja i upravljanja sustavom
 - najčešće nastaju kao rezultat izgradnje cijelog sustava "od početka"

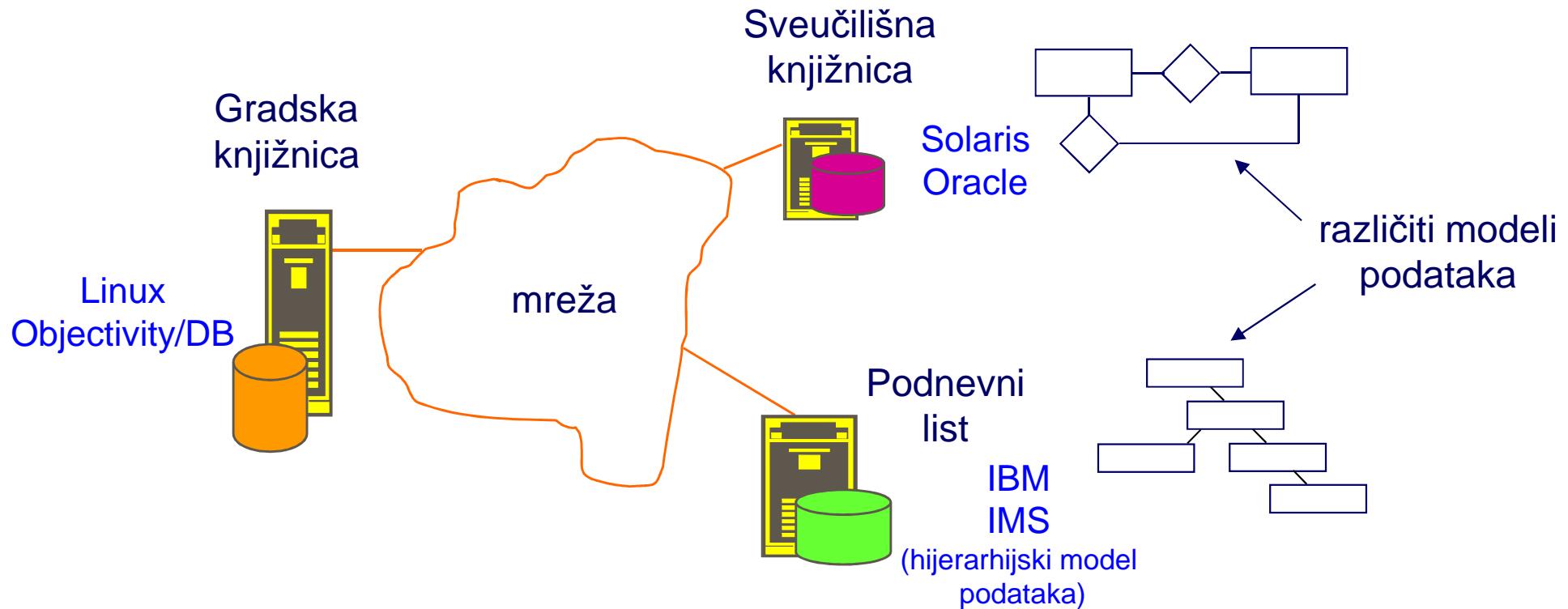


- globalna shema baze podataka je unija lokalnih shema baze podataka

Heterogeni DSUBP

- problem: kako izgraditi homogeni distribuirani sustav integriranjem već postojećih baza podataka?
 - tehničke poteškoće: troškovi ujednačavanja programske potpore
 - političke poteškoće: zadržavanje autonomije postojećih sustava
- čvorovi koriste različite:
 - operacijske sustave, sklopolje, mrežne protokole i/ili
 - sustave za upravljanje bazama podataka
 - problem osiguravanja ACID svojstava transakcija
 - modele podataka (mrežni, hijerarhijski, relacijski, objektno-relacijski, objektni) i/ili
 - načine opisivanja pravila integriteta i/ili
 - upitne jezike (ili različite verzije istih upitnih jezika)
- i/ili se lokalni čvorovi međusobno razlikuju prema
 - značenju, interpretaciji i namjeni istih ili srodnih podataka
→ semantička heterogenost
 - problem nazivlja (sinonimi, homonimi)

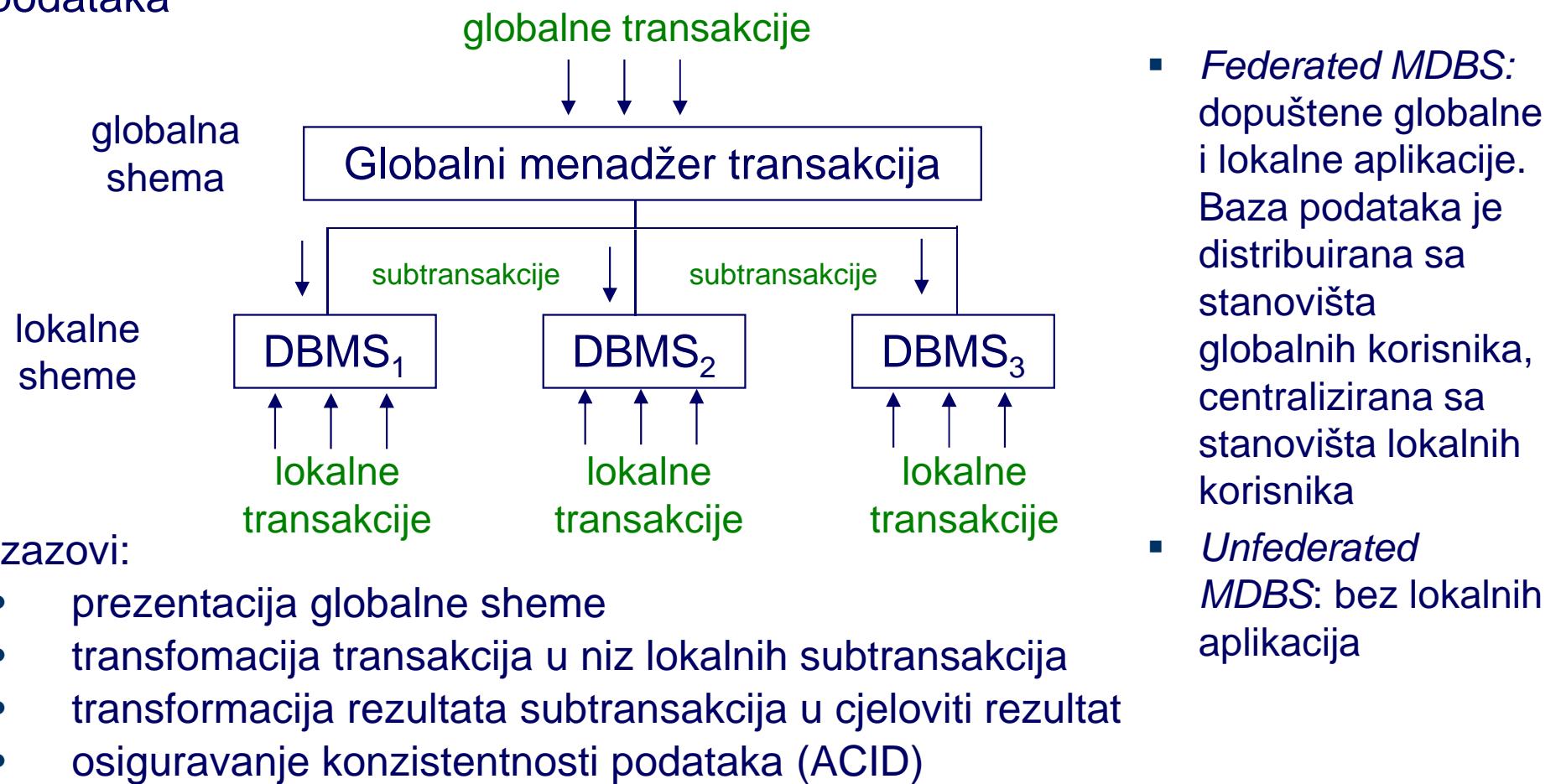
Heterogeni DSUBP



- pri obavljanju upita - najveći je problem razlika u shemama
 - semantička heterogenost otežava definiranje globalne sheme
- pri obavljanju transakcija: najveći je problem različita programska potpora
 - osiguravanje ACID svojstava transakcija

Sustavi multibaza podataka (*Multidatabase systems*)

- integracija heterogenih baza podataka
- čvorovi zadržavaju autonomiju
- dodaje se novi sloj kojim se dobiva unificirani pogled na distribuiranu bazu podataka



Oblikovanje distribuirane baze podataka

- važan dio postupka oblikovanja distribuirane baze podataka jest određivanje načina distribucije podataka. Cilj je optimalno iskorištenje sustava:
 - podaci se smještaju u čvorove u kojima se najčešće koriste
 - minimalizira se mrežni promet

oblikovanje distribucije = fragmentacija + alokacija

- shema fragmentacije
 - podjela baze podataka u disjunktni skup fragmenata koji obuhvaćaju sve podatke u bazi podataka uz zadovoljenje pravila da se baza podataka može rekonstruirati iz tih fragmenata bez gubitka informacije
 - relacije mogu biti razdijeljene u fragmente horizontalno ili vertikalno (ili horizontalno i vertikalno)
- shema alokacije
 - shema kojom se opisuje koji je fragment pridružen kojem čvoru distribuiranog sustava

Fragmentacija

- omogućuje paralelno obavljanje operacija nad fragmentima relacije
- omogućuje podjelu relacije na fragmente tako da se fragmenti mogu smjestiti u čvorove u kojima se najčešće koriste

Fragmentacija se smatra korektnom ukoliko posjeduje svojstva:

Obnovljivost

- ako se relacija r dekomponira u fragmente r_1, r_2, \dots, r_n , tada mora postojati operacija ∇ takva da je $r = \nabla r_i$, ($i = 1, \dots, n$). Pravilom se osigurava mogućnost rekonstrukcije relacije r bez gubitka informacija

Disjunktnost

- ako se relacija r horizontalno fragmentira u fragmente r_1, r_2, \dots, r_n , tada vrijedi da je $r_i \cap r_j = \emptyset$, $\forall i, j \in \{1, \dots, n\}$, $i \neq j$
- ako se relacija r sa shemom R vertikalno fragmentira u fragmente r_1, r_2, \dots, r_n , sa shemama R_1, R_2, \dots, R_n , tada $R_i \cap R_j, \forall i, j \in \{1, \dots, n\}$, $i \neq j$, sadrži samo attribute primarnog ključa relacije r

Horizontalna fragmentacija

- Neka je $\{ F_1, F_2, \dots, F_n \}$ skup formula definiranih na shemi R relacije r. Neka $\forall i, j \in \{ 1, \dots, n \}, i \neq j$, i za svaku n-torku $t \in r$ vrijedi da je $F_i(t) \wedge F_j(t) = \perp$ i $F_1(t) \vee F_2(t) \vee \dots \vee F_n(t) = T$. Horizontalnom fragmentacijom se relacija r sa shemom R transformira u skup relacija (fragmenata) r_1, r_2, \dots, r_n sa shemom R, pri čemu je $r_i = \sigma_{F_i}(r)$, ($i = 1, \dots, n$)
- Operacija ∇ kojom se rekonstruira relacija r je operacija unije, tj.
$$r = \bigcup r_i, (i = 1, \dots, n)$$

Formula

- neka je r relacija definirana na shemi R i neka su atributi $A, B \in R$. Neka je ϑ relacijski operator iz skupa $\{ =, \geq, >, \leq, <, \neq \}$. Neka je c konstanta iz skupa domene atributa A. Formula F je definirana rekurzivnim izrazom:
 1. $A \vartheta B, A \vartheta c, c \vartheta A$ su formule.
 2. Ako su G i H formule, tada su $G \vee H, G \wedge H, \neg G, \neg H$ također formule.
 3. Ništa drugo nije formula

Horizontalna fragmentacija

Primjer:

fakultet

sifFakultet

nazFakultet

K = { sifFakultet }

student

mbrStud

prezStud

imeStud

sifFakultet

K = { mbrStud }

- u relaciji student evidentirani su studenti koji studiraju na jednom od tri fakulteta koji su identificirani šiframa 36, 102, 81
- relacija student se može horizontalno fragmentirati na sljedeći način:

$\text{student}_1 = \sigma_{\text{sifFakultet} = 36} (\text{student})$

$\text{student}_2 = \sigma_{\text{sifFakultet} = 102} (\text{student})$

$\text{student}_3 = \sigma_{\text{sifFakultet} = 81} (\text{student})$

- fragment može obuhvaćati cijelu relaciju, npr:

$\text{fakultet}_1 = \text{fakultet}$

Vertikalna fragmentacija

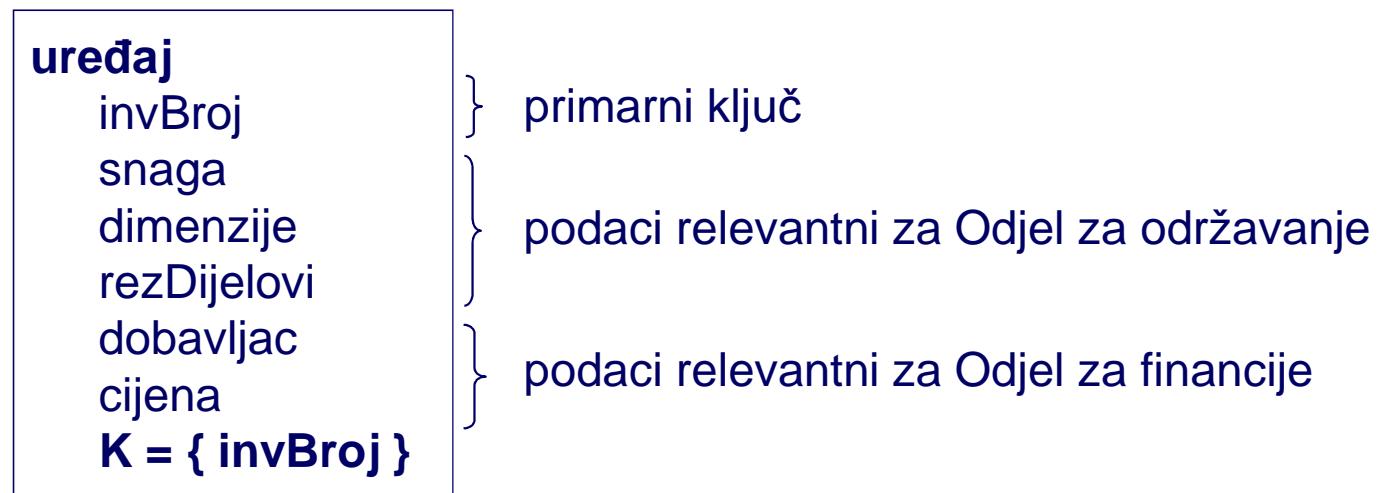
- Vertikalna fragmentacija relacije r sa shemom R i primarnim ključem K provodi se dekomponiranjem relacije r na relacije (fragmente) r_1, r_2, \dots, r_n s pripadnim shemama R_1, R_2, \dots, R_n , pri čemu vrijedi:
 - $\cup R_i = R$, ($i=1, \dots, n$), te vrijedi $R_i \cap R_j = K$, $\forall i, j \in \{1, \dots, n\}$, $i \neq j$
 - $r_i = \pi_{R_i}(r)$.
- Operacija ∇ kojom se rekonstruira relacija r je operacija prirodnog spajanja, tj.

$$r = r_1 \triangleright \triangleleft r_2 \dots \triangleright \triangleleft r_n$$

Vertikalna fragmentacija

Primjer:

- vertikalna fragmentacija relacije uređaj



$$\text{UREĐAJ}_1 = \{ \text{invBroj}, \text{snaga}, \text{dimenzije}, \text{rezDijelovi} \}$$

$$\text{UREĐAJ}_2 = \{ \text{invBroj}, \text{dobavljac}, \text{cijena} \}$$

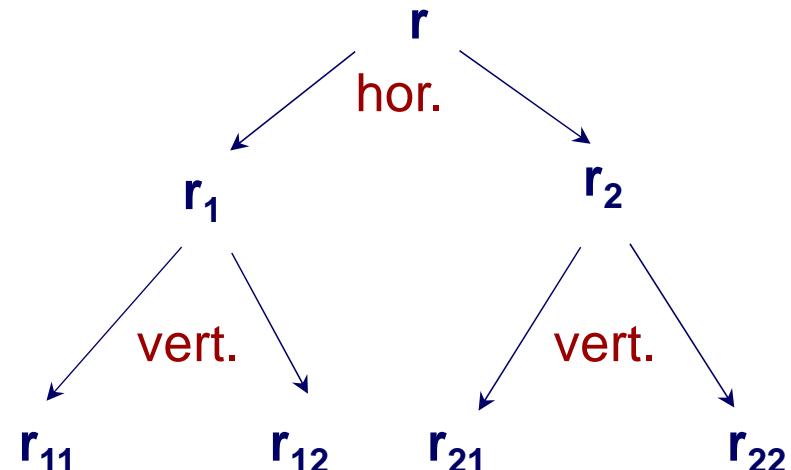
$$\text{uređaj}_1 = \pi_{\text{UREĐAJ}_1}(\text{uređaj})$$

$$\text{uređaj}_2 = \pi_{\text{UREĐAJ}_2}(\text{uređaj})$$

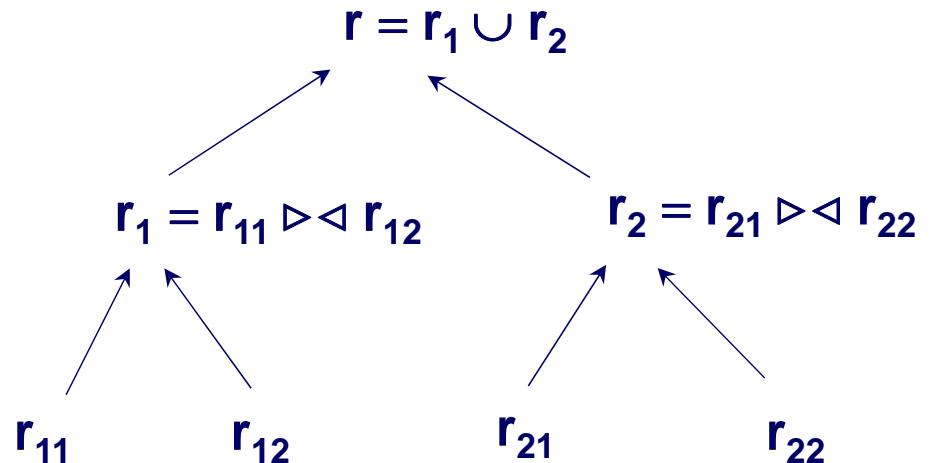
Hibridna fragmentacija

- fragmente dobivene horizontalnom ili vertikalnom fragmentacijom moguće je provesti kroz još jednu ili više faza fragmentacije. Slika ilustrira slučaj u kojem je relacija r najprije horizontalno fragmentirana, te je nad dobivenim fragmentima provedena vertikalna fragmentacija.

Hibridna fragmentacija relacije r



Rekonstrukcija relacije r



Alokacija

Stupanj replikacije fragmenta

- broj čvorova u kojima je fragment alociran
- svaki fragment mora biti alociran u barem jednom čvoru!
- **particionirana (ili nereplicirana) baza podataka**
 - svaki od fragmenata alociran je u točno jednom čvoru, tj. stupanj replikacije svakog fragmenta = 1
- **potpuno replicirana baza podataka**
 - svaki od fragmenata alociran je u svim čvorovima - svaki čvor sadrži repliku cijele baze podataka, tj. stupanj replikacije svakog fragmenta = n (broj čvorova u DSUBP)
- **parcijalno replicirana baza podataka**
 - baza podataka nije niti particionirana niti potpuno replicirana (svaki od fragmenata može biti alociran u jednom, više ili svim čvorovima)

Alokacija

Primjer:

- S_1 je čvor na fakultetu sa šifrom 36, S_2 je čvor na fakultetu sa šifrom 102, ...
 - shema alokacije
 - fragment student₁ alociran je u čvoru S_1 ,
 - fragment student₂ alociran je u čvoru S_2
 - fragment student₃ alociran je u čvoru S_3
 - fragment fakultet₁ alociran je u čvorovima S_1, S_2, S_3
- parcijalno replicirana baza podataka

Primjer:

- S_1 je čvor na Odjelu za održavanje, S_2 je čvor na Odjelu za financije
 - shema alokacije
 - fragment uređaj₁ alociran je u čvoru S_1
 - fragment uređaj₂ alociran je u čvoru S_2
- particionirana baza podataka

Transparentnost podataka

- od korisnika DSUBP-a ne smije se zahtijevati znanje o tome gdje su podaci fizički smješteni niti na koji im način treba pristupati. Svojstvo se naziva transparentnost podataka i ima tri važna aspekta:

Transparentnost fragmentacije:

- korisnici (ili aplikacije) ne trebaju voditi računa o načinu na koji je relacija fragmentirana

Transparentnost lokacije:

- korisnici (ili aplikacije) ne trebaju znati u kojem je čvoru alociran koji fragment. Svakom podatku moraju moći pristupiti ukoliko im je poznat identifikator podatka

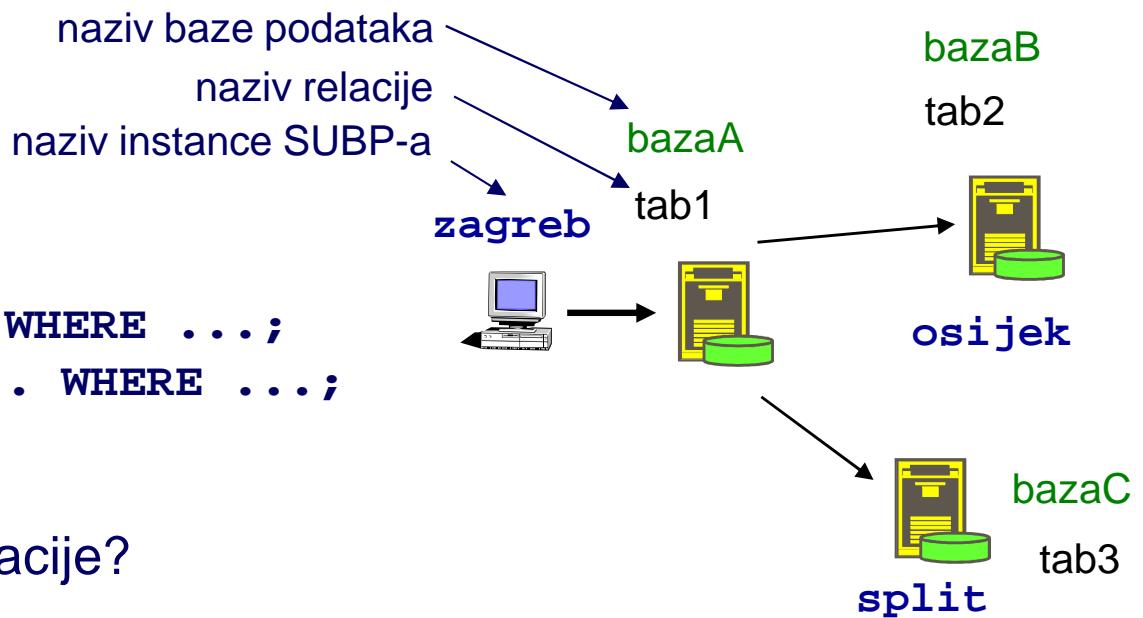
Transparentnost replikacije:

- korisnici (ili aplikacije) ne trebaju voditi računa o postojanju kopija fragmenata (replikama)

Transparentnost podataka

Primjer:

```
DATABASE bazaA;  
BEGIN WORK;  
INSERT INTO tab1 VALUES (...);  
DELETE FROM bazaB@osijek:tab2 WHERE ...;  
UPDATE bazaC@split:tab3 SET ... WHERE ...;  
COMMIT WORK;
```



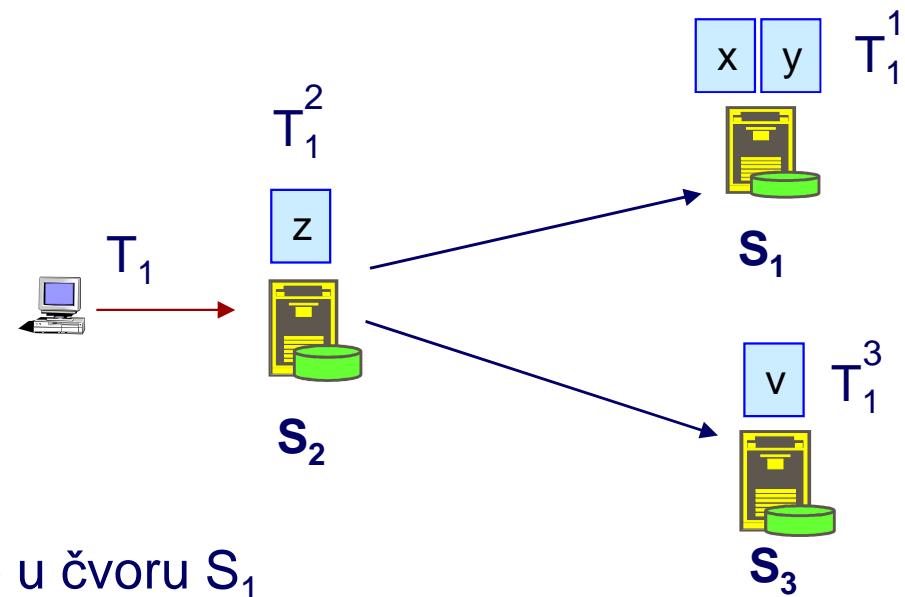
- kako postići transparentnost lokacije?
 - npr, definiranjem sinonima:
 - CREATE PUBLIC SYNONYM tab2 FOR bazaB@osijek:tab2
 - CREATE PUBLIC SYNONYM tab3 FOR bazaC@split:tab3
- kako postići transparentnost fragmentacije?
 - npr, definiranjem virtualnih relacija i sinonima
- kako postići transparentnost replikacije?
 - znatno teži problem

Transakcije u DSUBP-u

Globalne transakcije, subtransakcije i lokalne transakcije

Primjer: čvorovi DSUBP-a: S_1, S_2, S_3 ; elementi BP: x, y, z, v
stupanj replikacije = 1; shema alokacije: $S_1: x, y$ $S_2: z$ $S_3: v$

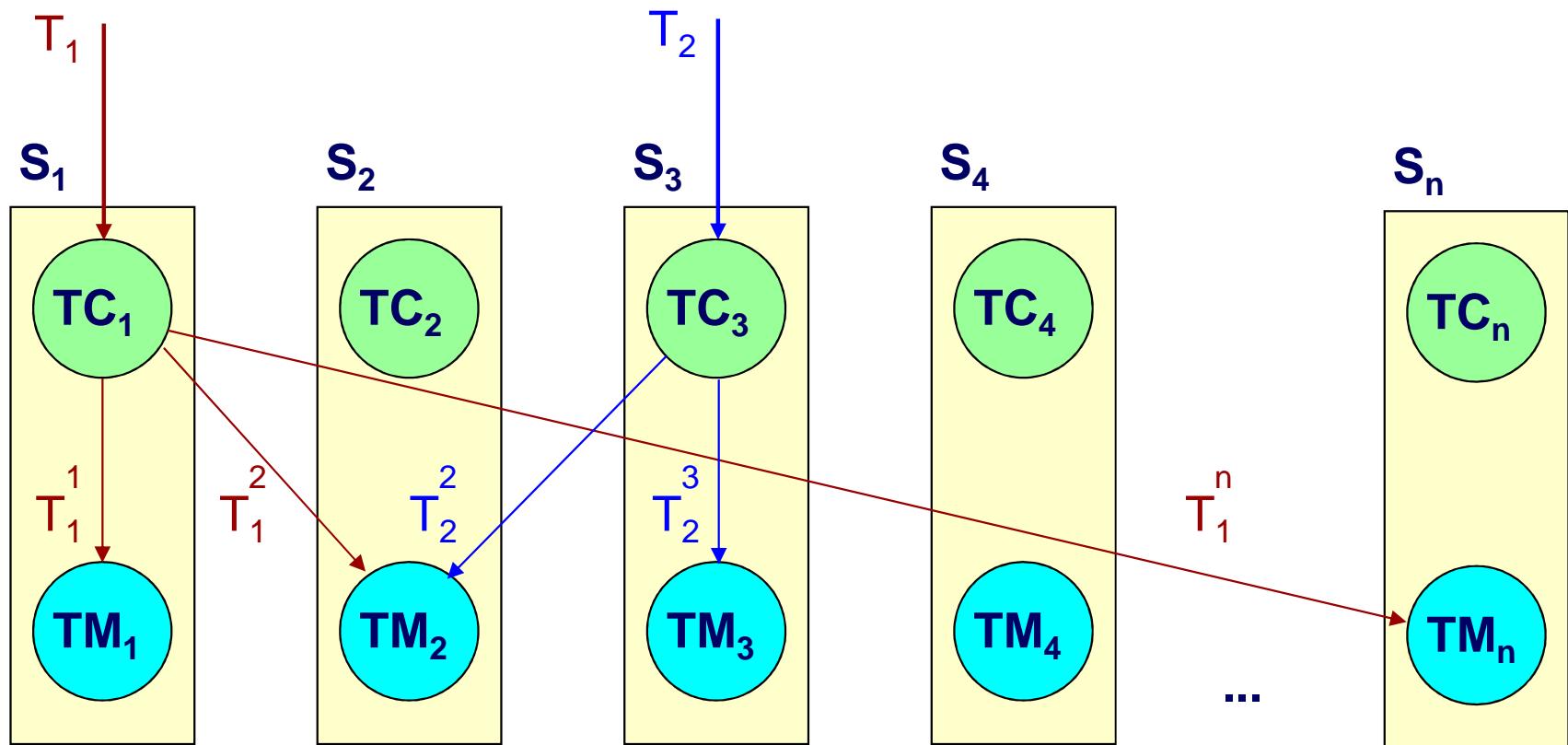
- korisnik inicira transakciju $T_1: w_1[x] \ r_1[y] \ w_1[z] \ r_1[v]$ u čvoru S_2
 - T_1 je globalna transakcija (zašto?)
- transakcija T_1 se preslikava u skup subtransakcija T_1^1, T_1^2, T_1^3 . Svaka od subtransakcija sadrži operacije koje se obavljaju u pripadnom čvoru
- oznake: T_i^j subtransakcija globalne transakcije T_i koja se izvršava u čvoru S_j
- $T_1^1: w_1^1[x] \ r_1^1[y]$
- $T_1^2: w_1^2[z]$
- $T_1^3: r_1^3[v]$
- primjer lokalne transakcije inicirane u čvoru S_1
 $T_2: r_2[x] \ r_2[y] \ w_2[y]$



Komponente DSUBP-a

- **u svakom čvoru nalazi se menadžer transakcija (TM)**
 - zadaće jednake onima u centraliziranom sustavu: obnova, izolacija, ...
 - razlika: osim lokalnih transakcija, obavljaju se i subtransakcije koje se izvršavaju na njegovoj lokaciji
- **u svakom čvoru nalazi se koordinator transakcija (TC)**
 - pokreće globalnu transakciju koja ima izvor na njegovoj lokaciji
 - distribuira subtransakcije u odgovarajuće čvorove - daje naloge pojedinim TM da obave subtransakciju
 - upravlja završetkom globalne transakcije koja ima izvor na njegovoj lokaciji na način da pripadne subtransakcije budu potvrđene u svim čvorovima ili poništene u svim čvorovima

Komponente DSUBP-a



- TC čvora u kojem je inicirana globalna transakcija koordinira izvršavanjem subtransakcija

Kvarovi u DSUBP-u

- upravljanje kvarovima u DSUBP-u je složenije nego u centraliziranim sustavima
 - centralizirani sustav funkcionira u cijelosti ili ne funkcionira
 - podsjetnik: pretpostavka o prekidu rada sustava u slučaju pogreške (*fail-stop* ili *fail-fast assumption*)
 - dijelovi DSUBP-a mogu biti u kvaru, a dijelovi nastaviti s radom
- osim kvarova koji su karakteristični za centralizirane sisteme (npr. pogreške programske podrške, sklopoljja, uništenja diska), u DSUBP-u su moguće dodatne vrste kvarova:
 - prestanak rada jednog ili više čvorova
 - gubitak veze među čvorovima
 - gubitak poruka
 - podjela mreže: mreža je podijeljena (particionirana) kad je podijeljena u nekoliko podsustava koji međusobno ne mogu komunicirati. Naročiti problem: čvor S_i ne može utvrditi je li se dogodila podjela mreže ili je neki čvor S_j prestao raditi

Transakcija u DSUBP-u

- u svakom čvoru se nalazi zasebni, potpuno funkcionalan sustav za upravljanje bazama podataka
- čvorovi distribuiranog sustava za upravljanje bazama podataka ne dijele zajedničke fizičke komponente (disk, memorija, procesor)
- transakcija se više ne može promatrati kao (samo) niz logički povezanih operacija koje se izvršavaju u istom memorijskom prostoru, pod kontrolom jednog menadžera transakcija, jednog menadžera redoslijeda izvršavanja, jednog menadžera obnove, ...
- globalna transakcija je skup subtransakcija koje koordinirano izvršavaju sustavi za upravljanje bazama podataka u više čvorova i pri tome prevode *distribuiranu* bazu podataka iz jednog u drugo konzistentno stanje
- **ACID?**
 - Svojstvo **Consistency** se relativno jednostavno ostvaruje uobičajenim mehanizmima
 - Svojstvo **Durability** osiguravaju menadžeri obnove u čvorovima
 - jer čvor TM_i garantira izdržljivost subtransakcije koja se izvršava u S_i
 - znatno teži problem: svojstva **Atomicity** i **Isolation**

Atomarnost

- atomarnost subtransakcija osiguravaju lokalni čvorovi
- kako osigurati atomarnost globalne transakcije?
 - za vrijeme obavljanja globalne transakcije može se dogoditi prekid veze između jednog ili više čvorova ili se u jednom ili više čvorova može dogoditi kvar
- atomarnost globalne transakcije znači da TM u svim čvorovima u kojima se izvršavaju pripadne subtransakcije moraju donijeti i provesti jednaku odluku o ishodu obavljanja transakcije: ili su sve subtransakcije globalne transakcije obavljene, ili nije niti jedna
- TC obavlja protokol potvrđivanja globalne transakcije
 - 2PC - *two-phase commit* (protokol dvofaznog potvrđivanja)

2PC protokol

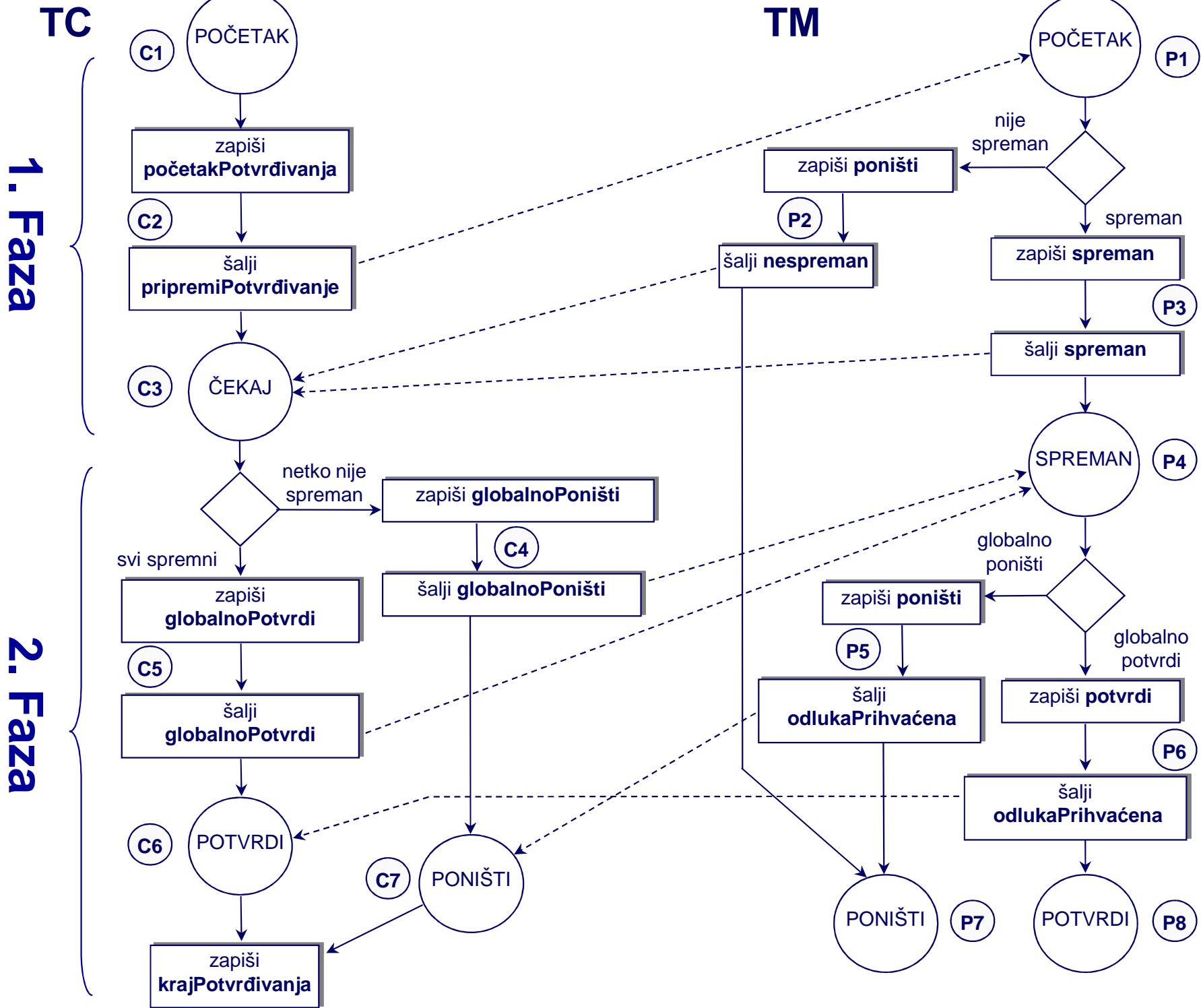
- TC koji se nalazi u izvornom čvoru transakcije distribuiru subtransakcije prema odgovarajućim TM
- nakon obavljanja subtransakcija, svi TM izvješćuju TC o uspješnom obavljanju operacija subtransakcija. **Tek tada započinje 2PC**

1. FAZA

- TC šalje svim TM poruku *pripremiPotvrđivanje*. Svaki pojedini TM odgovara *spreman* ili *nespreman* ili ne odgovara

2. FAZA

- ukoliko je sa svih lokacija stigla poruka *spreman* → TC odluku upisuje u svoj dnevnik i svim TM šalje poruku *globalnoPotvrdi*
- u slučaju da je neki od TM odgovorio *nespreman* ili se tijekom zadanog vremena neki od TM nije odazvao → TC odluku upisuje u dnevnik i svim TM šalje poruku *globalnoPoništi*
- svaki TM zapisuju odluku TC-a u svoj dnevnik, prema TC-u šalju potvrdu o prihvaćanju odluke i potvrđuju ili poništava svoju subtransakciju
- kada TC dobije potvrde svih TM, u logički dnevnik upisuje oznaku *krajPotvrđivanja*



Opis protokola

- ako zbog kvara u mreži ili kvara udaljenog čvora poruka ne stigne u unaprijed zadano vremenu (*timeout*), TC ili TM nastoje nastaviti s obavljanjem operacija u cilju izbjegavanja blokiranja transakcije

C3: TC čeka odluku jednog ili više TM. TC može donijeti odluku o poništavanju globalne transakcije

C6, C7: TC ne može zaključiti jesu li svi TM proveli odluku o potvrđivanju ili poništenju subtransakcije. TC uzastopno proziva TM čije poruke o prihvaćanju odluke nije zaprimio

P1: TM očekuje poruku od TC o početku pripreme za potvrđivanje. TM može nakon isteka *timeout* unilateralno poništiti subtransakciju, a ako TC nakon toga pošalje poruku *pripremiPotvrđivanje*, TM šalje poruku *nespreman*

P4: TM je poslao poruku *spreman*, ali nije mu poznata konačna odluka TC-a. TM mora čekati ponovnu uspostavu komunikacije s TC

Opis protokola

- TC koji tijekom obnove pomoću zapisa iz dnevnika utvrdi da je u trenutku kvara sudjelovao u 2PC protokolu, ovisno o trenutku u kojem se dogodio kvar obavlja sljedeće akcije:

C1: nakon obnove, TC može ponovo započeti 2PC protokol na uobičajeni način

C2, C3: TC je prekinuo rad nakon što je u dnevnik zapisao *početakPotvrđivanja*. Nakon obnove nastaviti će obavljanje protokola ponovnim slanjem poruke *pripremiPotvrđivanje*

C4, C5, C6, C7: TC je prekinuo rad nakon što je u dnevnik zapisao *globalnoPotvrdi* ili *globalnoPoništi*. Nakon obnove, ponovo će poslati odgovarajuće poruke prema TM

Opis protokola

- TM koji tijekom obnove pomoću zapisa iz dnevnika utvrdi da je u trenutku kvara sudjelovao u 2PC protokolu, ovisno o trenutku u kojem se dogodio kvar obavlja sljedeće akcije:

P1: TM je prekinuo rad prije nego je u dnevnik zapisao *poništi* ili *spreman*. Tijekom obnove TM unilateralno poništava transakciju

P2: TM je prekinuo rad nakon što je u dnevnik zapisao *poništi*. TM poništava subtransakciju i TC-u prepušta da nakon isteka zadanog vremena čekanja globalno poništi transakciju

P3, P4: TM je prekinuo rad nakon što je u dnevnik zapisao *spreman*. TM šalje prema TC poruku *spreman* i čeka odgovor jer bez TC ne može donijeti konačnu odluku

P5, P6: TM poznaje sudbinu globalne transakcije i postupa u skladu s time

P7, P8: TM ne treba poduzeti ništa jer se nalazi u stanju u kojem je završio transakciju

Mogućnost blokiranja protokola

- protokol je blokirajući ukoliko postoji mogućnost da ispravan čvor (TC ili TM) neće moći završiti transakciju zbog prekida veze ili kvara nekog drugog čvora

Primjer:

- točka P4 na prethodnoj slici
 - TM je prema TC poslao poruku *spreman* i nalazi se u stanju čekanja na odluku TC o sudbini globalne transakcije. U tom trenutku dogodi se kvar na vezi prema TC (ili kvar sustava u kojem se nalazi TC)
 - TM ne smije unilateralno poništiti lokalnu transakciju jer ne zna kakvu je odluku donio TC (možda je TC svim ostalim čvorovima uspio poslati poruku *globalnoPotvrdi*)
 - TM mora čekati uspostavu veze s TC (ili obnovu sustava u kojem se nalazi TC)

⇒ 2PC protokol je blokirajući protokol

Nezavisnost protokola s obzirom na mogućnost obnove

- protokol je nezavisan s obzirom na mogućnost obnove ukoliko svaki čvor (TC ili TM), nakon što se u njemu dogodio kvar sustava (ili medija) može samostalno, bez komunikacije s ostalim čvorovima, donijeti odluku o sudbini svih (sub)transakcija koje su se u tom čvoru odvijale u trenutku kvara

Primjer:

- točka P4 na prethodnoj slici
 - TM je u dnevnik zapisao *spreman* i prema TC poslao poruku *spreman*. U tom trenutku se dogodi kvar sustava na TM
 - kada TM započne obnovu utvrdit će da je u trenutku prekida rada sudjelovao u 2PC protokolu. Bez TC ne može donijeti odluku treba li subtransakciju potvrditi ili poništiti, stoga prema TC šalje poruku *spreman* i čeka odgovor

⇒ 2PC protokol nije nezavisan s obzirom na mogućnost obnove

Varijante 2PC protokola (dvije od mnogih)

Linearni 2PC protokol

- između $TM_1, TM_2, TM_3, \dots TM_n$ utvđuje se potpuni poredak. TC šalje poruku *pripremiPotvrđivanje* prema TM_1 , TM_1 šalje poruku *spreman* ili *nespreman* prema TM_2 , itd. sve do TM_n . TM_n šalje poruku *globalnoPotvrđi* ili *globalnoPoništi* prema TM_{n-1} , taj proslijeđuje poruku prema TM_{n-2} , i tako sve do TC. Kada poruka konačno stigne do TC, protokol završava
- smanjuje se ukupni broj poruka, ali se onemogućuje istodobno obavljanje operacija protokola u TM-ovima

Varijante 2PC protokola

2PC protokol s upisivanjem liste ključeva u dnevnik

- svrha: ublažavanje problema zavisnosti protokola s obzirom na mogućnost obnove nakon kvara u TM čvoru
 - ako se u TM_i dogodio kvar nakon što je TM_i u dnevnik upisao spreman, a prije nego je u dnevnik upisao potvrđi ili poništi (tj. u točkama P3 ili P4), obnova TM-a zahtijeva komunikaciju s TC-om (jer treba doznati konačnu odluku o sudbini transakcije) → usporavanje obnove
- u prvoj fazi protokola, umjesto spreman, TM u dnevnik zapisuje (spreman, L). L je lista X-ključeva dotične subtransakcije u trenutku zapisivanja u dnevnik
- pri obnovi, TM_i umjesto čekanja na razrješenje (*commit/rollback*), za svaku upitnu subtransakciju ponovo postavlja sve ključeve iz njoj pripadne liste L
- sada se može nastaviti obavljanje novih transakcija, a potvrđivanje ili poništavanje upitnih transakcija se može obavljati istodobno s obavljanjem novih transakcija
 - uz posljedicu: nove lokalne transakcije i subtransakcije u TM_i neće moći zaključavati podatke koje su mijenjale subtransakcije čija sudbina još nije razrješena

Globalna serijalizabilnost

Jedan od načina osiguravanja globalne serijalizabilnosti:

- globalna serijalizabilnost u particioniranim bazama podataka postiže se kombiniranom uporabom **rigoroznog** 2PL protokola i 2PC protokola
 - u repliciranim bazama podataka problem je složeniji (koju ili koje od kopija podataka treba zaključati?)

Primjer: čvorovi DSUBP-a: S_1, S_2 ; elementi BP: x, y

stupanj replikacije = 1; shema alokacije: $S_1: x \quad S_2: y$

- globalne transakcije $T_1: r_1[x], r_1[y]; \quad T_2: w_2[x], w_2[y]$
- neka je globalna povijest: $r_1[x], w_2[x], w_2[y], r_1[y]$

$T_1^1 : r_1^1[x]$

Lokalni H^1

$T_2^1 : w_2^1[x]$

$r_1^1[x], w_2^1[x]$

Lokalni SG^1

$T_1 \longrightarrow T_2$

Globalni $SG = SG^1 \cup SG^2$

$T_1^2 : r_1^2[y]$

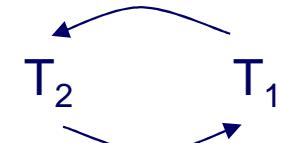
Lokalni H^2

$T_2^2 : w_2^2[y]$

$w_2^2[y], r_1^2[y]$

Lokalni SG^2

$T_2 \longrightarrow T_1$



očito, globalna povijest
nije serijalizabilna

Globalna serijalizabilnost

Primjer (nastavak): $r_1^1[x], w_2^1[x], w_2^2[y], r_1^2[y]$

- oznakom $2PC_i$ označena je operacija globalnog $2PC$ potvrđivanja transakcije T_i u koju su uključene operacije lokalnog potvrđivanja svih subtransakcija transakcije T_i
- globalna povijest s operacijama postavljanja ključeva - dva slučaja:
- ako se koristi striktni $2PL$ protokol (S-ključeve smije se otpustiti i prije kraja subtransakcije), sustav bi mogao producirati povijest koja nije serijalizabilna:
 $sL_1^1[x], r_1^1[x], u_1^1[x], xL_2^1[x], w_2^1[x], xL_2^2[y], w_2^2[y], 2PC_2, u_2^1[x], u_2^2[y], sL_1^2[y], r_1^2[y], u_1^2[y], 2PC_1$
- treba primijetiti da čvorovi dosljedno provode striktni $2PL$ protokol
- ako se koristi rigorozni $2PL$ protokol (svi ključevi se otpuštaju tek po završetku $2PC$), sustav će osigurati serijalizabilnost globalne transakcije:
 $sL_1^1[x], r_1^1[x], xL_2^1[x], sL_1^2[y], r_1^2[y], 2PC_1, u_1^1[x], u_1^2[y], xL_2^2[x], w_2^1[x], xL_2^2[y], w_2^2[y], 2PC_2, u_2^1[x], u_2^2[y]$

Zaključavanje u DSUBP-u

Centralizirani LM (*Lock Manager* - menadžer zaključavanja)

- LM se nalazi u odabranom čvoru S_i
- kad transakcija treba zaključati neki podatak – šalje zahtjev za postavljanjem ključa u čvor S_i i LM zaključuje može li se ključ dodijeliti
- **prednosti:**
 - jednostavnija detekcija potpunog zastoja - centralizirani LM održava globalni WFG (*wait-for graph*)
- **nedostaci:**
 - čvor u kojem se nalazi LM može postati usko grlo
 - *single-point-of-failure* – čvor u kojem se nalazi LM

Distribuirani LM

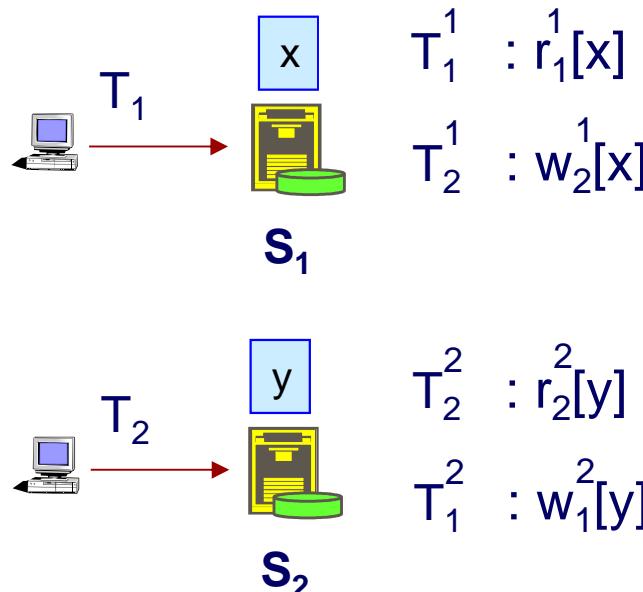
- LM je implementiran u svakom čvoru
- LM upravlja istodobnim obavljanjem subtransakcija i lokalnih transakcija u "svom" čvoru
- **prednosti:**
 - posao je raspodijeljen – manja osjetljivost na pogreške
- **nedostaci:**
 - problem detekcije potpunog zastoja

Detekcija potpunih zastoja pri korištenju distribuiranog LM

Održavanje WFG u svakom čvoru zasebno nije dovoljno

Primjer: čvorovi DSUBP-a: S_1, S_2 ; elementi BP: x, y
stupanj replikacije = 1; shema alokacije: $S_1: x \quad S_2: y$

- globalna transakcija $T_1: r_1[x], w_1[y]$ inicirana u čvoru S_1
- globalna transakcija $T_2: r_2[y], w_2[x]$ inicirana u čvoru S_2
- neka je globalna povijest: $r_1[x], r_2[y], w_2[x], w_1[y]$



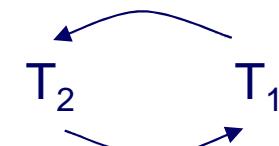
Lokalni WFG¹

$T_2 \longrightarrow T_1$

Lokalni WFG²

$T_1 \longrightarrow T_2$

Globalni WFG =
 $WFG^1 \cup WFG^2$



Detekcija potpunih zastoja pri korištenju distribuiranog LM

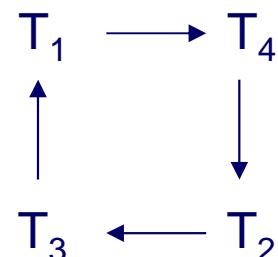
Primjer za vježbu:

čvorovi DSUBP-a: S_1, S_2 ; elementi BP: x, y, v, z

stupanj replikacije = 1; shema alokacije: $S_1: x, y \quad S_2: v, z$

- globalna transakcija $T_1: r_1[x], r_1[z]$ inicirana u čvoru S_1
- globalna transakcija $T_2: r_2[v], r_2[y]$ inicirana u čvoru S_2
- lokalna transakcija $T_3: w_3[y], w_3[x]$ inicirana u čvoru S_1
- lokalna transakcija $T_4: w_4[z], w_4[v]$ inicirana u čvoru S_2
- neka je globalna povijest: $r_1[x], r_2[v], w_3[y], w_4[z], r_1[z], r_2[y], w_3[x], w_4[v]$
- nacrtati lokalne WFG i globalni WFG

Za provjeru rezultata:



$$\text{Globalni WFG} = \text{WFG}^1 \cup \text{WFG}^2$$

Detekcija potpunih zastoja pri korištenju distribuiranog LM

- u današnjim sustavima obično se koristi distribuirani LM - lokalni potpuni zastoji detektiraju se pomoću WFG, a globalni potpuni zastoji upotrebom mehanizma *deadlock-timeout*
 - čekanje na postavljanje ključa koje traje dulje od **parametrom** zadanog vremena tumači se kao posljedica globalnog potpunog zastoja
 - IBM Informix: DEADLOCK_TIMEOUT
 - Oracle: DISTRIBUTED_LOCK_TIMEOUT
 - moguće je utvrđivanje "lažnog" potpunog zastoja, npr. ako se radi o pukoj sporosti sustava ili se samo čeka na otključavanje nekog podatka
 - važno je pažljivo postaviti vrijednost parametra (uzeti u obzir svojstva mreže, trajanje transakcija, opeterećenju sustava, itd.)
 - previsoka vrijednost: transakcije se predugo zadržavaju u stanju potpunog zastoja
 - preniska vrijednost: poništavaju se transakcije koje nisu zaista sudjelovale u potpunom zastoju

Replikirane baze podataka

Replicirane baze podataka

- fragment je repliciran ako je alociran u dva ili više čvorova
- za jedan logički element x (n-torku, fragment, relaciju) postoji više fizičkih elemenata (kopija, replika), x^1, x^2, \dots , u čvorovima S_1, S_2, \dots

Prednosti repliciranih BP

- povećanje dostupnosti
 - ako je čvor u kojem je pohranjena kopija fragmenta nedostupan, sustav može pristupiti kopiji fragmenta u nekom drugom čvoru
- smanjenje volumena prijenosa podataka
 - podacima koji se često koriste u više čvorova aplikacije mogu pristupati lokalno
- paralelno obavljanje dijelova istog upita
 - upit koji obrađuje fragment može se dekomponirati, te se svaki dio upita može obavljati nad jednom od kopija fragmenta

Replcirane baze podataka

Nedostaci repliciranih BP

- problem konzistentnosti kopija istog elementa
 - sustav mora osigurati konzistentnost svih kopija. Operacija pisanja (unos, brisanje, izmjena) jedne kopije fragmenta mora se propagirati prema svim čvorovima u kojima je taj fragment alociran
 - w = učešće operacija pisanja u ukupnom broju operacija [0, 1]
 - r = učešće repliciranih elemenata u ukupnom broju elemenata [0, 1]
 - $w \times r$ = faktor replikacije
 - veći faktor replikacije povlači za sobom veći broj operacija koje je potrebno obaviti radi održavanja konzistentnosti kopija
 - veći broj operacija koje treba obaviti u većem broju čvorova može uzrokovati smanjenje dostupnosti i povećanje broja potpunih zastoja (pri sinkronoj replikaciji) ili narušavanje konzistentnosti (pri asinkronoj replikaciji)

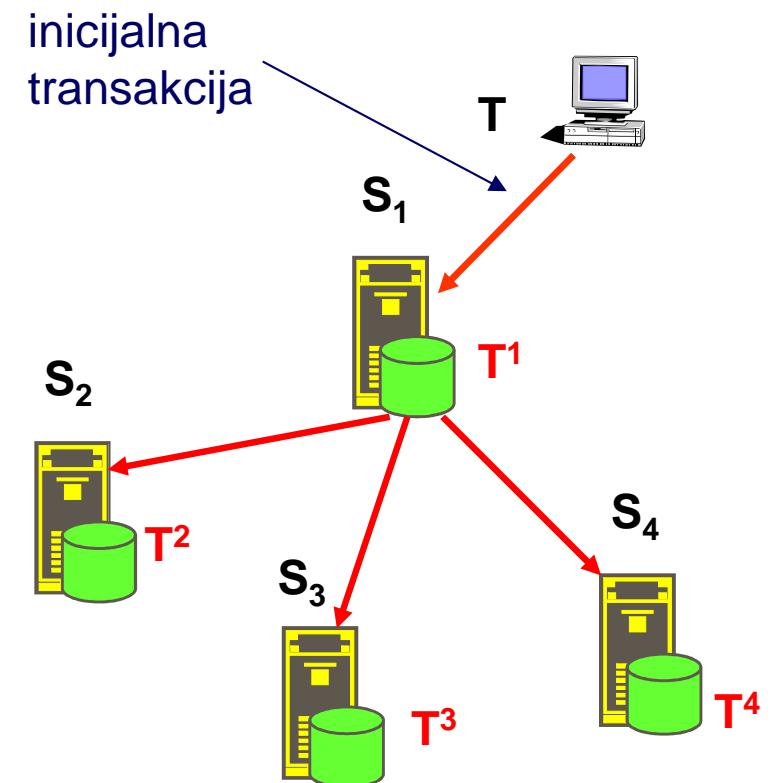
Svojstva repliciranih baza podataka

Doseg inicialne transakcije

- svojstvo određuje obavljuju li se izmjene svih kopija u okviru inicialne transakcije ili u posebnim propagacijskim (*refresh*) transakcijama

Sinkroni (*eager*) protokoli

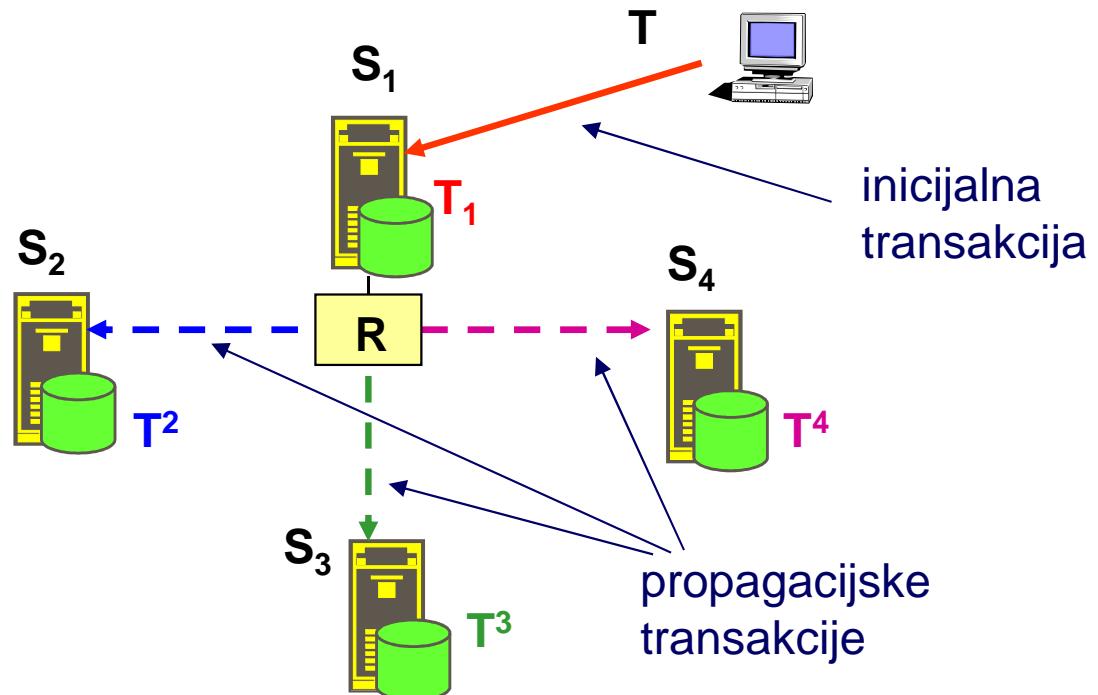
- sve fizičke operacije koje proizlaze iz logičkih operacija inicialne transakcije obavljaju se unutar granica inicialne transakcije, tj. sve kopije se moraju izmjeniti u okviru inicialne transakcije
- osigurana potpuna konzistentnost, dobre performance pri čitanju, slabije pri izmjeni
- produljenje trajanja transakcije, povećanje broja potpunih zastoja, niska dostupnost (ispadom samo jednog čvora operacije izmjene postaju neizvedive)



Svojstva repliciranih baza podataka

Asinkroni (*lazy*) protokoli

- operacije inicialne transakcije obavljaju se isključivo u inicialnom čvoru i niti na koji način ne ovise o komunikaciji s ostalim čvorovima
- inicialna transakcija može završiti prije nego su obavljene izmjene nad svim kopijama. Izmjene ostalih kopija obavljaju se asinkrono
- visoka dostupnost, dobre performance
- velika mogućnost pojave nekonzistentnih podataka



Svojstva repliciranih baza podataka

Vlasništvo

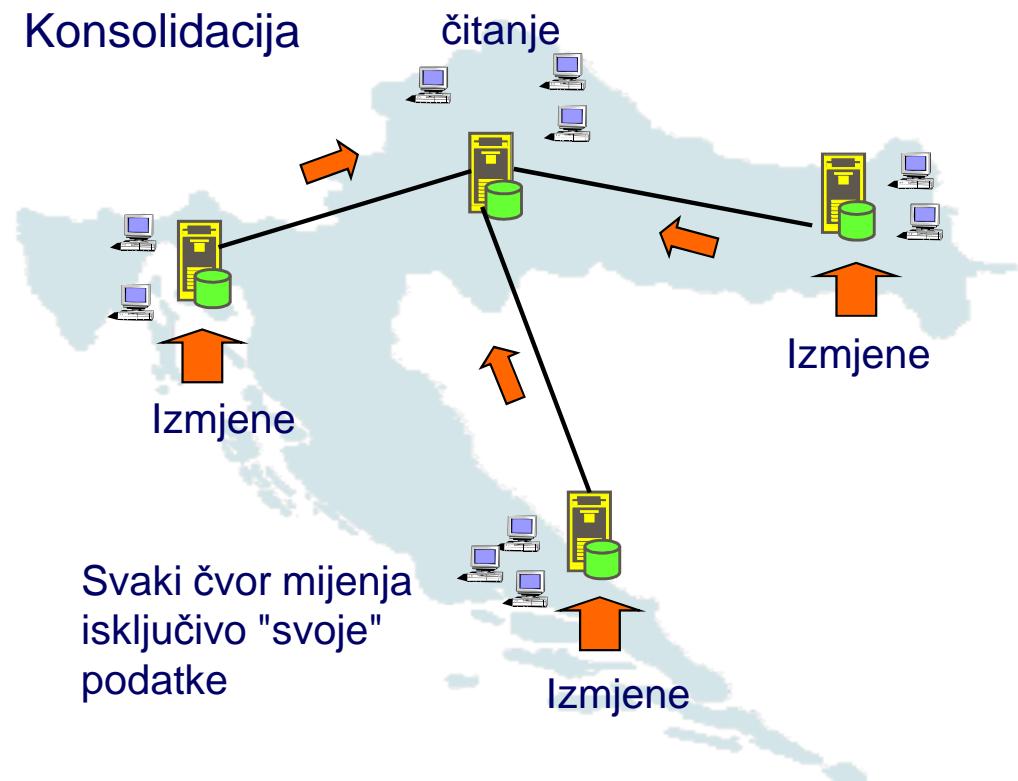
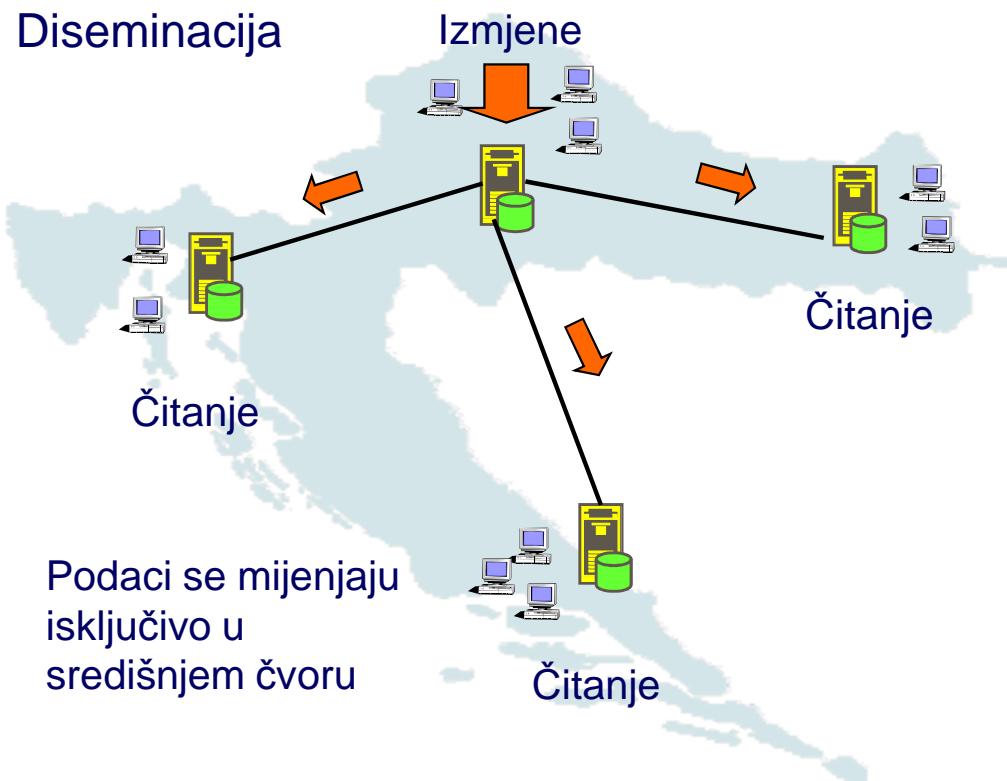
- svojstvo određuje stupanj slobode sustava pri određivanju kopije elementa x nad kojom se smije obaviti prva operacija pisanja $w[x]$

Jednosmjerni (*one-way, master-ownership, primary-copy*) protokoli

- za svaki logički element x određuje se samo jedan fizički element x^p koji se proglašava primarnom kopijom, te se operacija izmjene koju nad elementom x obavlja inicijalna transakcija mora prvo obaviti nad kopijom x^p
- svaki čvor u kojem je alocirana primarna kopija barem jednog elementa naziva se ravnatelj (*master*)
 - sustav s jednim ravnateljem (*single-master*), u kojem se nalaze sve primarne kopije
 - sustav s više ravnatelja (*multi-master*), u kojem su primarne kopije različitih podataka smještene u različitim čvorovima

Često korišteni oblici jednosmjernih protokola

- *1Master-nSlaves* (diseminacija podataka)
 - izmjene se obavljaju u točno jednom nadređenom (*master*) čvoru i propagiraju prema podređenim čvorovima (*slave*). Podređeni čvorovi ne smiju obavljati transakcije koje sadrže operacije pisanja
- *nMasters-1Slave* (konsolidacija podataka)
 - izmjene obavljene u više nadređenih (*master*) čvorova propagiraju se prema točno jednom podređenom (*slave*) čvoru. Podređeni čvor ne smije obavljati transakcije koje sadrže operacije pisanja



Svojstva repliciranih baza podataka

Dvosmjeri (two-way, n-way, peer-to-peer, group-ownership, update-anywhere) protokoli

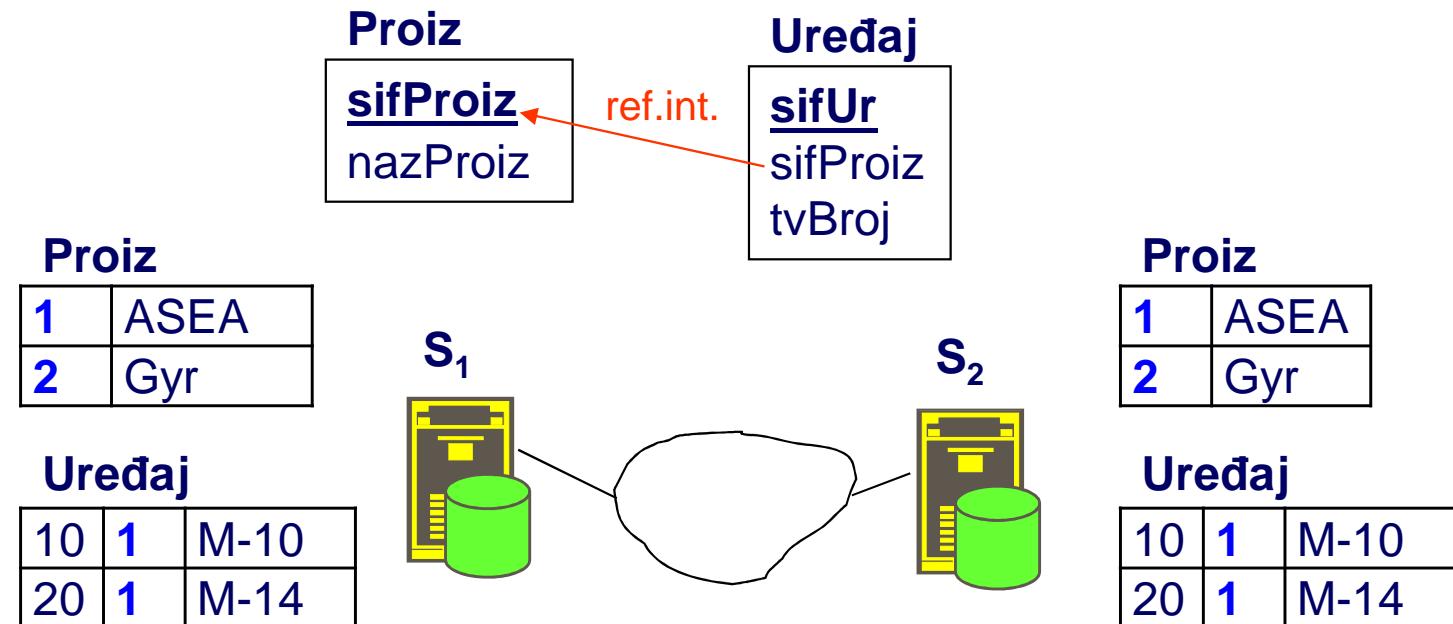
- inicijalna transakcija može operaciju izmjene obaviti nad bilo kojom fizičkom kopijom
- dostupnost sustava se bitno povećava u odnosu na jednosmjerne sustave
- koristi li se u kombinaciji s asinkronim protokolom serijalizabilnost izvršavanja transakcija se ne može garantirati

Važni nedostaci dvosmjernih asinkronih protokola

- neserijalizabilno obavljanje transakcija može dovesti do teško ispravljivog narušavanja konzistentnosti podataka
- problem detekcija konfliktata: neki konflikti mogu biti otkriveni tek nakon propagacije izmjena (kad je inicijalna transakcija već potvrđena)
- problem razrješavanja konfliktata: može zahtijevati poništavanje potvrđene transakcije → narušavanje svojstva izdržljivosti transakcije
- automatsko rješavanje konfliktata često nije moguće – potrebna je intervencija čovjeka

Primjer:

potpuno replicirana
baza podataka



Važni nedostaci dvosmjernih asinkronih protokola

	S_1		S_2																						
	proiz	uredaj	proiz	uredaj																					
9:40	<table border="1"> <tr><td>1</td><td>ASEA</td></tr> <tr><td>2</td><td>Gyr</td></tr> </table>	1	ASEA	2	Gyr	<table border="1"> <tr><td>10</td><td>1</td><td>M-10</td></tr> <tr><td>20</td><td>1</td><td>M-14</td></tr> </table>	10	1	M-10	20	1	M-14	<table border="1"> <tr><td>1</td><td>ASEA</td></tr> <tr><td>2</td><td>Gyr</td></tr> </table>	1	ASEA	2	Gyr	<table border="1"> <tr><td>10</td><td>1</td><td>M-10</td></tr> <tr><td>20</td><td>1</td><td>M-14</td></tr> </table>	10	1	M-10	20	1	M-14	
1	ASEA																								
2	Gyr																								
10	1	M-10																							
20	1	M-14																							
1	ASEA																								
2	Gyr																								
10	1	M-10																							
20	1	M-14																							
9:41	DELETE FROM proiz WHERE sifProiz=2	<table border="1"> <tr><td>1</td><td>ASEA</td></tr> <tr><td>2</td><td>Gyr</td></tr> </table>	1	ASEA	2	Gyr	<table border="1"> <tr><td>10</td><td>1</td><td>M-10</td></tr> <tr><td>20</td><td>1</td><td>M-14</td></tr> </table>	10	1	M-10	20	1	M-14	<table border="1"> <tr><td>1</td><td>ASEA</td></tr> <tr><td>2</td><td>Gyr</td></tr> </table>	1	ASEA	2	Gyr	<table border="1"> <tr><td>10</td><td>1</td><td>M-10</td></tr> <tr><td>20</td><td>1</td><td>M-14</td></tr> </table>	10	1	M-10	20	1	M-14
1	ASEA																								
2	Gyr																								
10	1	M-10																							
20	1	M-14																							
1	ASEA																								
2	Gyr																								
10	1	M-10																							
20	1	M-14																							
9:42		<table border="1"> <tr><td>1</td><td>ASEA</td></tr> <tr><td>20</td><td>1</td><td>M-14</td></tr> </table>	1	ASEA	20	1	M-14	INSERT INTO uredaj VALUES (30, 2, M-16)	<table border="1"> <tr><td>1</td><td>ASEA</td></tr> <tr><td>2</td><td>Gyr</td></tr> </table>	1	ASEA	2	Gyr	<table border="1"> <tr><td>10</td><td>1</td><td>M-10</td></tr> <tr><td>20</td><td>1</td><td>M-14</td></tr> <tr><td>30</td><td>2</td><td>M-16</td></tr> </table>	10	1	M-10	20	1	M-14	30	2	M-16		
1	ASEA																								
20	1	M-14																							
1	ASEA																								
2	Gyr																								
10	1	M-10																							
20	1	M-14																							
30	2	M-16																							
9:43	INSRT INT uredaj VALUES (30, 2, M-16)	<table border="1"> <tr><td>1</td><td>ASEA</td></tr> <tr><td>20</td><td>1</td><td>M-14</td></tr> </table>	1	ASEA	20	1	M-14	DLTE FRM proiz WHERE sifProiz=2	<table border="1"> <tr><td>1</td><td>ASEA</td></tr> <tr><td>2</td><td>Gyr</td></tr> </table>	1	ASEA	2	Gyr	<table border="1"> <tr><td>10</td><td>1</td><td>M-10</td></tr> <tr><td>20</td><td>1</td><td>M-14</td></tr> <tr><td>30</td><td>2</td><td>M-16</td></tr> </table>	10	1	M-10	20	1	M-14	30	2	M-16		
1	ASEA																								
20	1	M-14																							
1	ASEA																								
2	Gyr																								
10	1	M-10																							
20	1	M-14																							
30	2	M-16																							
Sinkronizacija	ERROR-referential integrity: missing row		ERROR-referential integrity: still referencing row																						

- rezultat: system delusion (zaluđenost sustava)

[Gray, Helland, ...1996.]

Važni nedostaci dvosmjernih asinkronih protokola

- moderni sustavi podržavaju dvosmjernu asinkronu replikaciju, ali sveobuhvatno rješenje za prethodno opisani problem ne postoji
- u različitim sustavima se nude različite ugrađene opcionalne funkcionalnosti koje mogu pomoći u specifičnim slučajevima. Npr. u nekim sustavima je moguće
 - umjesto direktnog propagiranja SQL naredbe, propagirati poziv pohranjene procedure (koju definira korisnik) i koja razriješava moguće konflikte
 - ako se uz pomoć vremenskih oznaka (*timestamp*) utvrdi mogući konflikt, poništava se inicijalna transakcija (kako to utječe na izdržljivost?)
 - *last wins, first wins, greatest value wins, ...*

Izolacija u repliciranim bazama podataka

- umjesto serijalizabilnosti, potrebno je osigurati svojstvo jedno-kopajske serijalizabilnosti (*one-copy serializability*): rezultat istodobnog obavljanja transakcija mora biti ekvivalentan rezultatu serijskog obavljanja transakcija u nerepliciranoj bazi podataka
- radi osiguranja serijalizabilnosti potrebno je postavljati globalne ključeve na logičkim elementima
- ako se koristi distribuirani menadžer zaključavanja, jedini način na koji se to može ostvariti jest postaviti jedan ili više ključeva na fizičke elemente
- zaključati logički element zaključavanjem svih fizičkih kopija tog elementa?
 - za potpuno repliciranu bazu podataka sa n čvorova, zaključavanje svih kopija logičkog elementa x znači zaključavanje elemenata u jednom lokalnom i $n-1$ udaljenom čvoru
 - $n-1$ poruka *zaključaj*
 - $n-1$ poruka *zaključano/nije zaključano*
 - $n-1$ poruka *otključaj*

Izolacija u repliciranim bazama podataka

Zaključavanje uz pomoć primarne kopije (*primary-copy locking*)

- za svaki logički element x utvrđuje se primarna lokacija p
- kad transakcija treba zaključati element x , ona mora zatražiti ključ na primarnoj lokaciji tog elementa (zaključava fizički element x^p)
- time implicitno dobiva ključ nad svim replikama tog podatka, odnosno postavlja ključ na logički element x
- ako se alokacija fragmenata obavi na ispravan način, transakcije će se (najčešće) izvršavati u čvorovima u kojima se nalaze primarne kopije logičkih elemenata koje te transakcije koriste

- prednosti
 - kontrola istodobnog pristupa repliciranim podacima obavlja se slično kao i za nereplicirane podatke - jednostavna implementacija
- nedostaci
 - ako čvor u kojem se nalazi kopija x^p postane nedostupan, element x postaje nedostupan iako u drugim čvorovima postoje kopije elementa x

Izolacija u repliciranim bazama podataka

Većinski protokol (*Majority Protocol*)

- moraju se zaključati kopije elementa x u više od polovice čvorova u kojima je x alociran
- operacija nad elementom x se ne obavlja sve dok se ne dobije ključ za većinu kopija elementa x
- prednosti:
 - primjenjiv i onda kad su neki čvorovi nedostupni
- nedostaci:
 - radi zaključavanja jednog logičkog elementa potrebno je zaključati $\lceil (n + 1) / 2 \rceil$ kopija
 - mogućnost potpunog zastoja čak i s jednim elementom – npr. svaka od 3 transakcije može postaviti ključeve na 1/3 kopija elementa x

Izolacija u repliciranim bazama podataka

Pristrani protokol (*Biased Protocol*)

- sličan većinskom protokolu
- prednost se daje operacijama čitanja (protokol je "pristrand")
- za obavljanje operacije čitanja dovoljno je postaviti S-ključ nad jednom kopijom
- za obavljanje operacije pisanja X-ključevi se moraju postaviti na svim kopijama
- operacija pisanja obavlja se nad svim kopijama (*Read-One, Write-All -ROWA*)
- prednosti:
 - manji trošak postavljanja ključeva radi operacija čitanja (koje uobičajeno prevladavaju među operacijama u bazi podataka)
- nedostaci:
 - dodatno otežano obavljanje operacija pisanja
 - jednaki problemi s potpunim zastojem kao kod većinskog protokola

Izolacija u repliciranim bazama podataka

Protokol usuglašavanja kvorumom (*Quorum Consensus Protocol*)

- protokol je generalizacija većinskog i pristranog protokola
- svakom čvoru S_i dodjeljuje se težinski faktor w_i (pozitivni cijeli broj), te se izračunava njihova suma Σ
- za svaki element x odabiru se dvije vrijednosti: *read-quorum* Q_r i *write-quorum* Q_w , takve da vrijedi
 - $Q_r + Q_w > \Sigma$ i $2 * Q_w > \Sigma$
- pri svakom čitanju mora se zaključati dovoljno replika: suma težinskih faktora w_i u čvorovima koji su odobrili zaključavanje mora biti $\geq Q_r$
- pri svakom pisanju mora se zaključati dovoljno replika: suma težinskih faktora w_i u čvorovima koji su odobrili zaključavanje mora biti $\geq Q_w$
- pouzdanijim čvorovima se mogu dodijeliti veći težinski faktori, čime se smanjuje broj ključeva koje treba postaviti

Literatura:

- M. Özsü, P. Valduriez. Principles of distributed database systems, 2nd ed. Prentice-Hall International, Inc. Upper Saddle River, New Jersey. 1999.
- P. Bernstein, V. Hadzilacos, N. Goodman. Concurrency control and recovery in database systems. Addison-Wesley Publishing Company. Reading, Massachusetts. 1987.
- H. Garcia-Molina, J. Ullman, J. Widom. Database system implementation. Prentice-Hall International, Inc. Upper Saddle River, New Jersey. 2000.
- A. Silberschatz, H. Korth, S. Sudarshan. Database system concepts, 4th ed. McGraw-Hill. 2002.
- J. Gray, P. Helland, P. O'Neil, D. Shasha. The danger of replication and a solution. Proceedings of the 1996 ACM SIGMOD Int. conf. on Management of Data. Montreal, Canada. 1996.