»Гл. ас. д-р Георги Чолаков

»Бази от данни

Транзакции и конкурентност

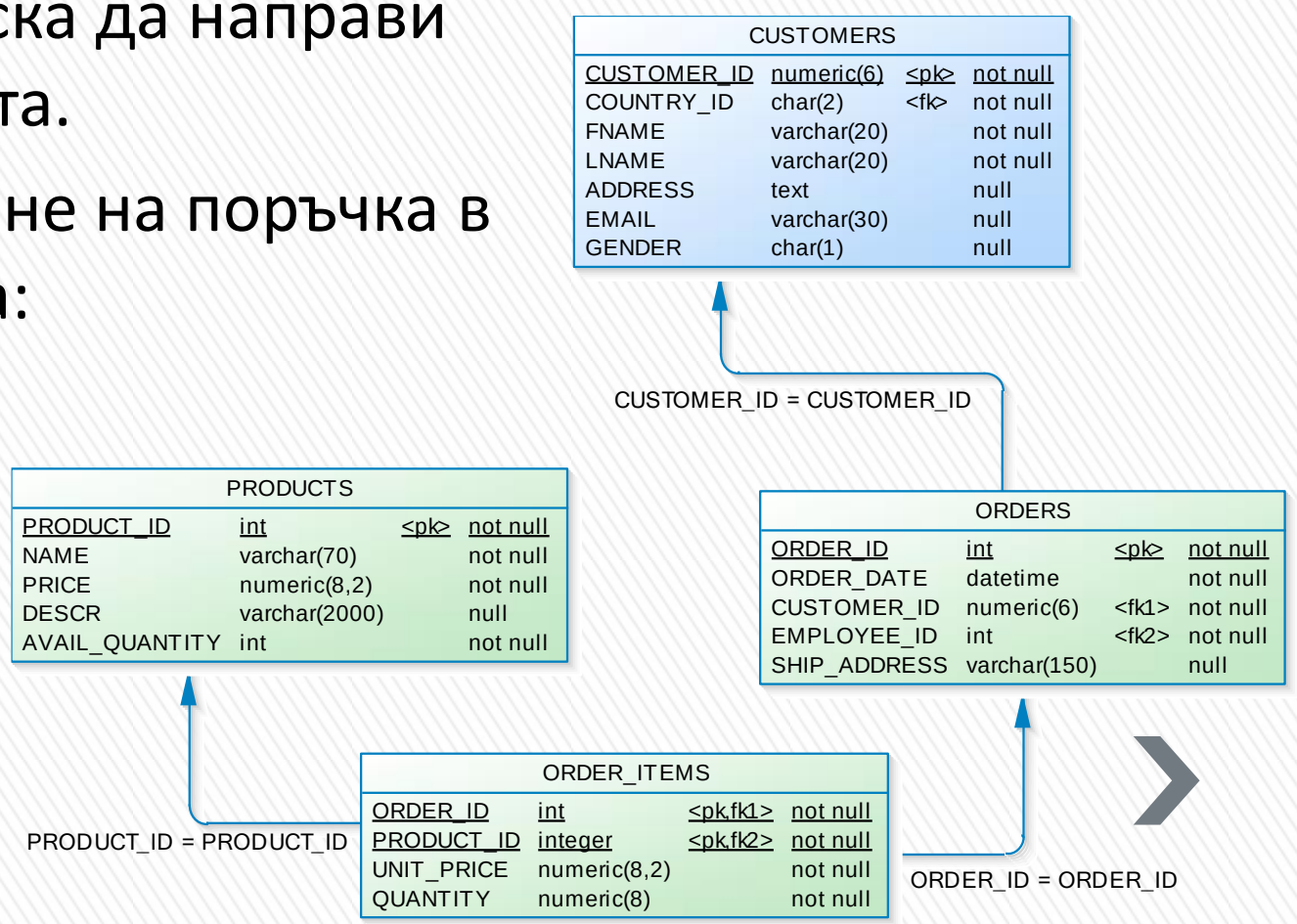
Въведение

В тази част ще разгледаме:

- Какво представляват транзакциите и какви са техните свойства;
- Какво е конкурентност и каква роля играе за интегритета на БД;
- Какво са заключванията и как работят.

Пример

- Нека имаме ситуация, в която клиент, който не е сред наличните в CUSTOMERS, иска да направи поръчка, съдържаща два продукта.
- Единичната операция за създаване на поръчка в базата данни би изглеждала така:
  - Добавяне на ред в CUSTOMERS за новия клиент;
  - Добавяне на ред в ORDERS за поръчката;
  - Добавяне на два реда за двата продукта в ORDER_ITEMS;
  - Намаляване на количествата за двата продукта в PRODUCTS.



Примерът в SQL операции

```
INSERT CUSTOMERS(CUSTOMER_ID, COUNTRY_ID, FNAME, LNAME, ADDRESS, EMAIL, GENDER)
VALUES(1001, 'BG', 'Виктория', 'Генева', 'бул. България 100, Пловдив',
'viki_97@gmail.com', 'F');

INSERT ORDERS(ORDER_ID, CUSTOMER_ID, EMPLOYEE_ID, SHIP_ADDRESS, ORDER_DATE)
VALUES(10001, 1001, 177, 'бул. България 140, Пловдив', '2019-11-23');

INSERT ORDER_ITEMS(ORDER_ID, PRODUCT_ID, UNIT_PRICE, QUANTITY)
VALUES (10001, 2289, 7.45, 2);

INSERT ORDER_ITEMS(ORDER_ID, PRODUCT_ID, UNIT_PRICE, QUANTITY)
VALUES (10001, 1004, 35.05, 1);

UPDATE PRODUCTS SET AVAIL_QUANTITY = AVAIL_QUANTITY - 2
WHERE PRODUCT_ID = 2289;

UPDATE PRODUCTS SET AVAIL_QUANTITY = AVAIL_QUANTITY - 1
WHERE PRODUCT_ID = 1004;
```

Извод

- Оказва се, че създаването на тази поръчка е съставено от 6 команди;
- Какво би се случило, ако изпълнението им бъде прекъснато след 3-тата? Поръчката ще съдържа само част от желаните продукти, а наличните количества на продуктите няма да са актуализирани – наличностите вече няма да отговарят на реалните складови такива;
- Дали е възможно да се разглеждат всички 6 команди като една?

Транзакция

- **Логическа единица за операция върху базата данни, обгръщаща множество операции с данните, т.е. представяща ги като единична операция.**
- Тя бива изцяло завършена или изцяло отхвърлена;
- Неприемливи са каквито и да било междинни състояния или частични промени на данните;
- Която и команда да пропадне цялата транзакция се отхвърля и данните се връщат в първоначалното състояние;
- Успешната транзакция променя данните от едно консистентно в следващо такова състояние.

Примерът в транзакция

```
BEGIN TRANSACTION;

  INSERT CUSTOMERS(CUSTOMER_ID, COUNTRY_ID, FNAME, LNAME, ADDRESS, EMAIL, GENDER)
  VALUES(1001, 'BG', 'Виктория', 'Генева', 'бул. България 100, Пловдив', 'viki_97@gmail.com', 'F');
  INSERT ORDERS(ORDER_ID, CUSTOMER_ID, EMPLOYEE_ID, SHIP_ADDRESS, ORDER_DATE)
  VALUES(10001, 1001, 177, 'бул. България 140, Пловдив', '2019-11-23');
  INSERT ORDER_ITEMS(ORDER_ID, PRODUCT_ID, UNIT_PRICE, QUANTITY)
  VALUES (10001, 2289, 7.45, 2);
  INSERT ORDER_ITEMS(ORDER_ID, PRODUCT_ID, UNIT_PRICE, QUANTITY)
  VALUES (10001, 1004, 35.05, 1);
  UPDATE PRODUCTS SET AVAIL_QUANTITY = AVAIL_QUANTITY - 2
  WHERE PRODUCT_ID = 2289;
  UPDATE PRODUCTS SET AVAIL_QUANTITY = AVAIL_QUANTITY - 1
  WHERE PRODUCT_ID = 1004;

IF @@ERROR <> 0
   ROLLBACK TRANSACTION;
ELSE
   COMMIT TRANSACTION;
```

Транзакционни оператори

- BEGIN TRANSACTION – поставя начало на транзакция;
- COMMIT TRANSACTION:
  - Сигнализира за успешен край на транзакцията;
  - TM прави всички промени постоянни;
  - Това е допустимо, понеже се знае, че БД се намира в ново консистентно състояние.
- ROLLBACK TRANSACTION:
  - Сигнализира за неуспешен край на транзакцията;
  - Transaction Manager знае, че БД се намира в неконсистентно състояние;
  - Всички промени, направени междувременно, се анулират;
  - Така БД се връща в изходното консистентно състояние.

Един реалистичен TM изпраща също така съобщения на потребителя за резултата от операцията - например „Транзакцията е изпълнена успешно.“ или „Транзакцията не е завършена поради следните причини: ...“

Реализация на транзакции

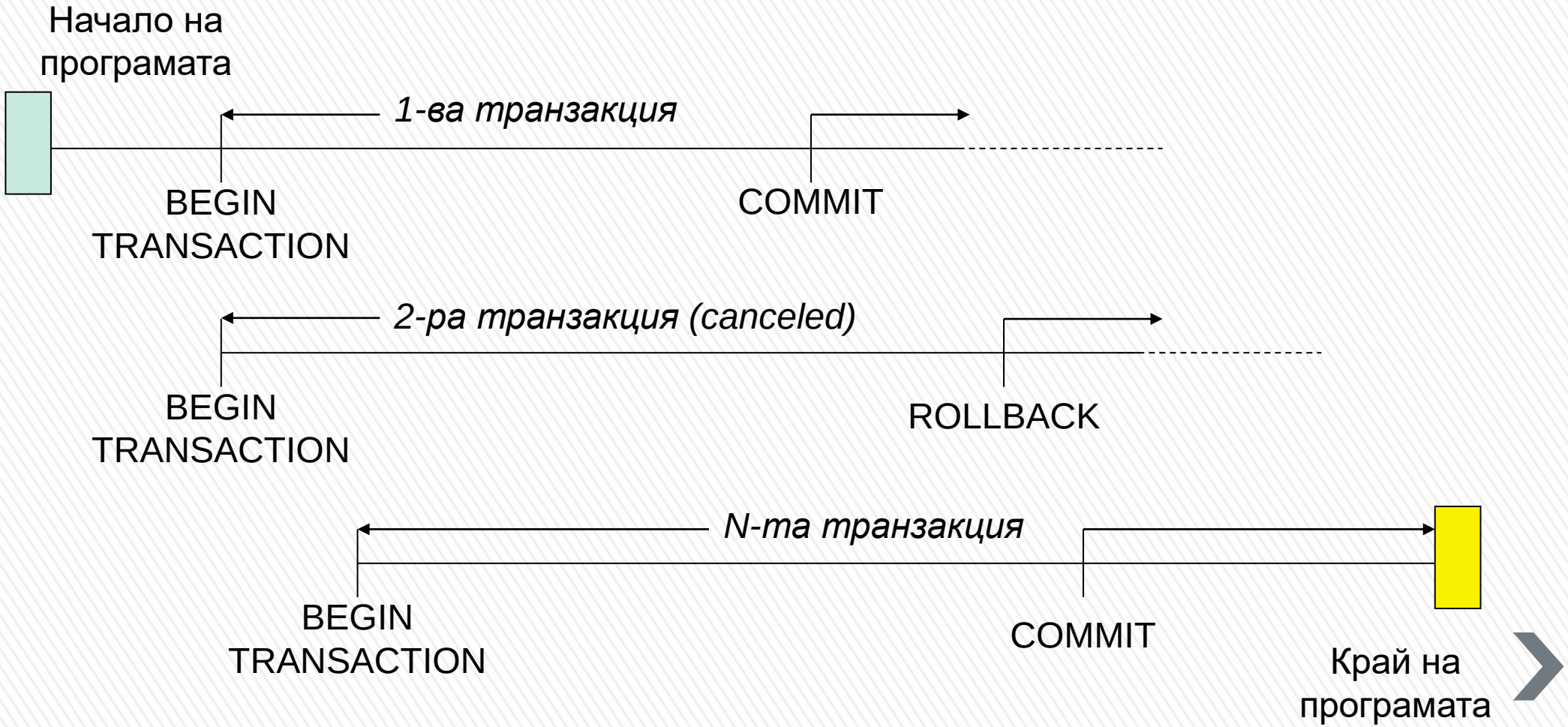
Реализацията на транзакциите се извършва по следната схема:

- СУБД води дневник (log) за всички детайли на извършващите се промени върху данните - също така и за стойностите преди и след промените;
- При анулиране ТМ търси съответната входна точка в дневника и възстановява предишното състояние;
- СУБД трябва също така да гарантира, че отделните оператори са атомарни – например при грешка по време на един UPDATE данните трябва да се променят (частично).

Свойства на транзакциите

- Атомарност (atomicity) - транзакциите се разглеждат като неразложими - всички операции в една транзакция се изпълняват успешно или промените от цялата транзакция биват отхвърлени, т.е. транзакцията се разглежда като неделима, логическа единица за операция;
- Консистентност (consistency) - трансформират БД от едно консистентно състояние в ново консистентно състояние;
- Изолираност (isolation) - отделните транзакции са изолирани една от друга – означава, че данните, които ползва една транзакция, не могат да бъдат използвани от друга, докато първата не приключи;
- Трайност (durability) - щом като една транзакция веднъж успее, направените от нея промени се запазват и не могат да бъдат отхвърлени, дори ако има последващ срив в системата.

Сценарий на използване на транзакции



Конкурентност

- Конкурентност: едновременен достъп на повече от една транзакция до едни и същи данни.

- Една многопотребителска СУБД обикновено разрешава едновременен достъп на повече транзакции към едни и същи данни;
- В СУБД съществува concurrency control mechanism за осигуряване, че конкурентните транзакции няма да си пречат взаимно;
- Контролът на конкурентността е важен, защото едновременното изпълнение на транзакции може да породи проблеми относно интегритета на данните.

Проблеми на конкурентността

Всеки механизъм за контрол на конкурентността трябва да отчита следните възможни проблеми:

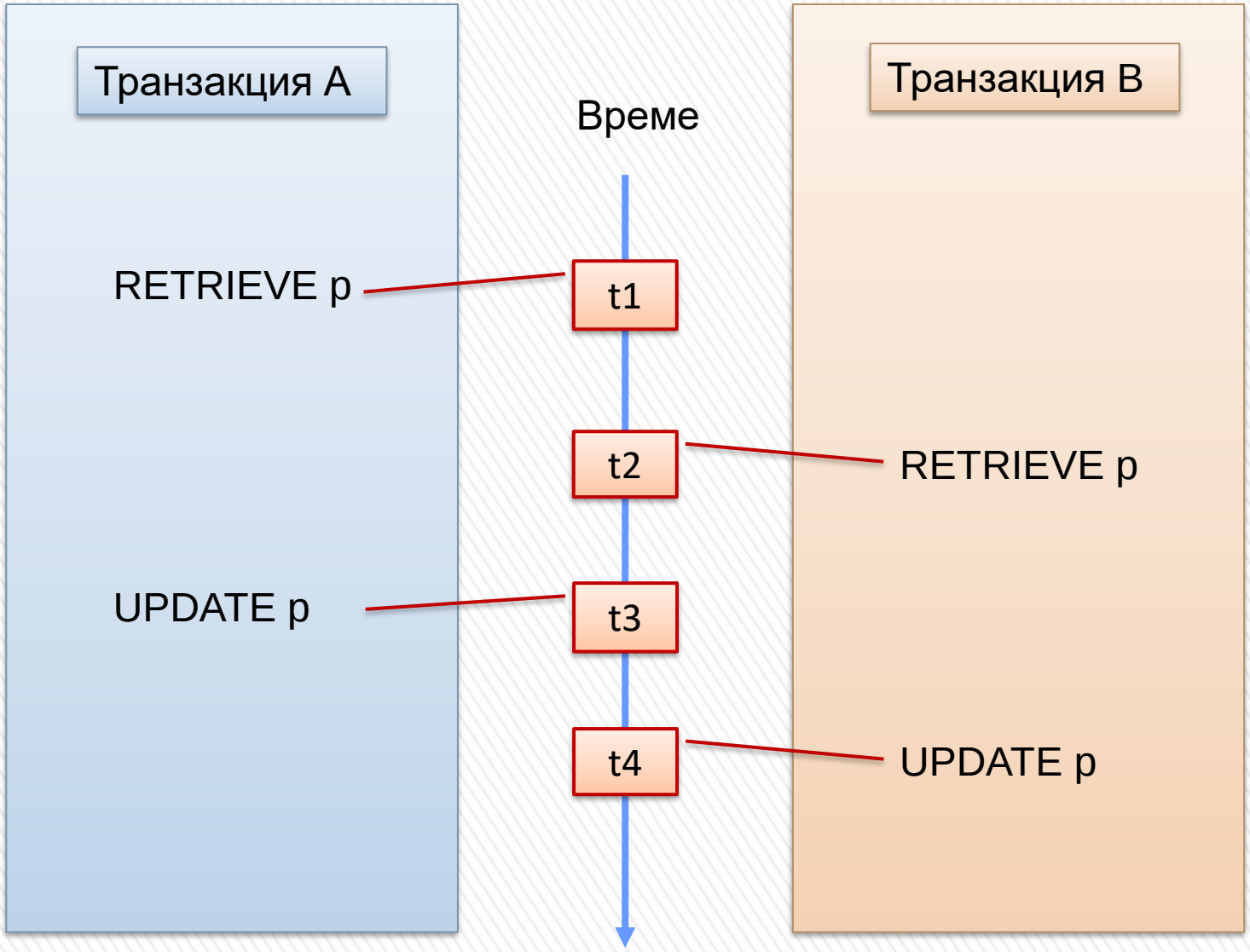
- Загуба на промяна (lost update);
- Незавършена зависимост (uncommitted dependency);
- Неконсистентен анализ (inconsistent analysis);
- Четене на редове-фантоми.

Загуба на промяна

Загуба на промяна възниква, когато две или повече транзакции избират един и същ ред и го актуализират, като се ръководят от първоначално прочетената стойност – тогава:

- Никоя транзакция няма представа за съществуването на другите транзакции;
- Последната направена актуализация припокрива промените, направени от другите транзакции и по този начин се получава загуба на промяна.

Загуба на промяна



Загуба на промяна

Нека p е един запис - транзакция A губи една промяна в t_4 както следва:

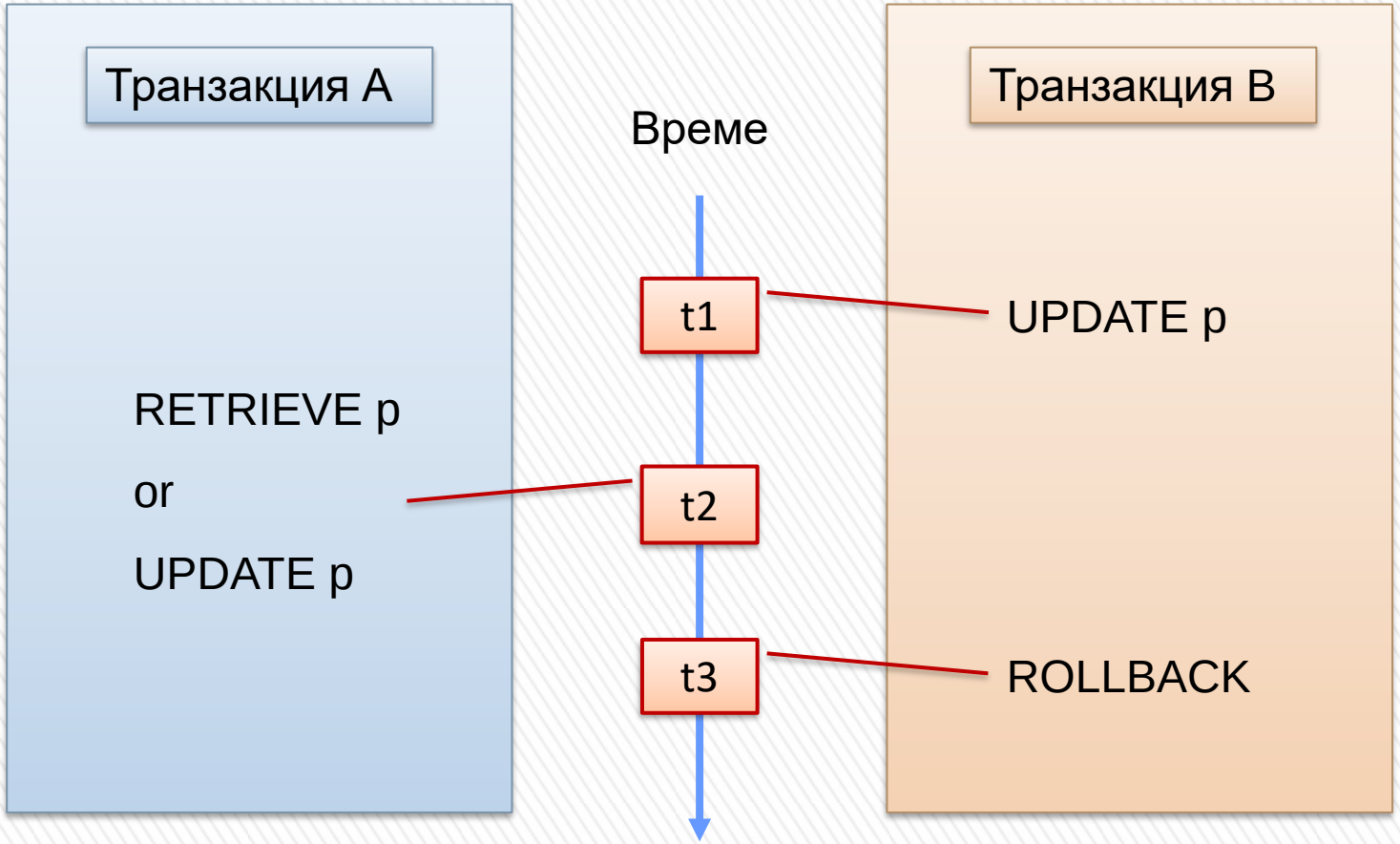
- в t_1 A чете ред p (в локалната си памет);
- в t_2 същия запис чете и B (в локалната си памет);
- в t_3 A променя p (на базата на стойностите в t_1);
- в t_4 B променя p (на базата на стойностите в t_2, които са същите както в t_1);
- в t_4 промените на A са загубени, защото записът е променен от B.

Незавършена зависимост – четене на непотвърдени данни

Незавършена зависимост възниква в случай, когато едната транзакция избира ред, който в момента се актуализира от друга транзакция:

- Тази транзакция чете променени данни, които все още не са потвърдени - промените могат да бъдат отхвърлени от транзакцията, която ги извършва;
- Така тази транзакция става зависима от променените, но непотвърдени данни, които е прочела.

Незавършена зависимост – четене на непотвърдени данни



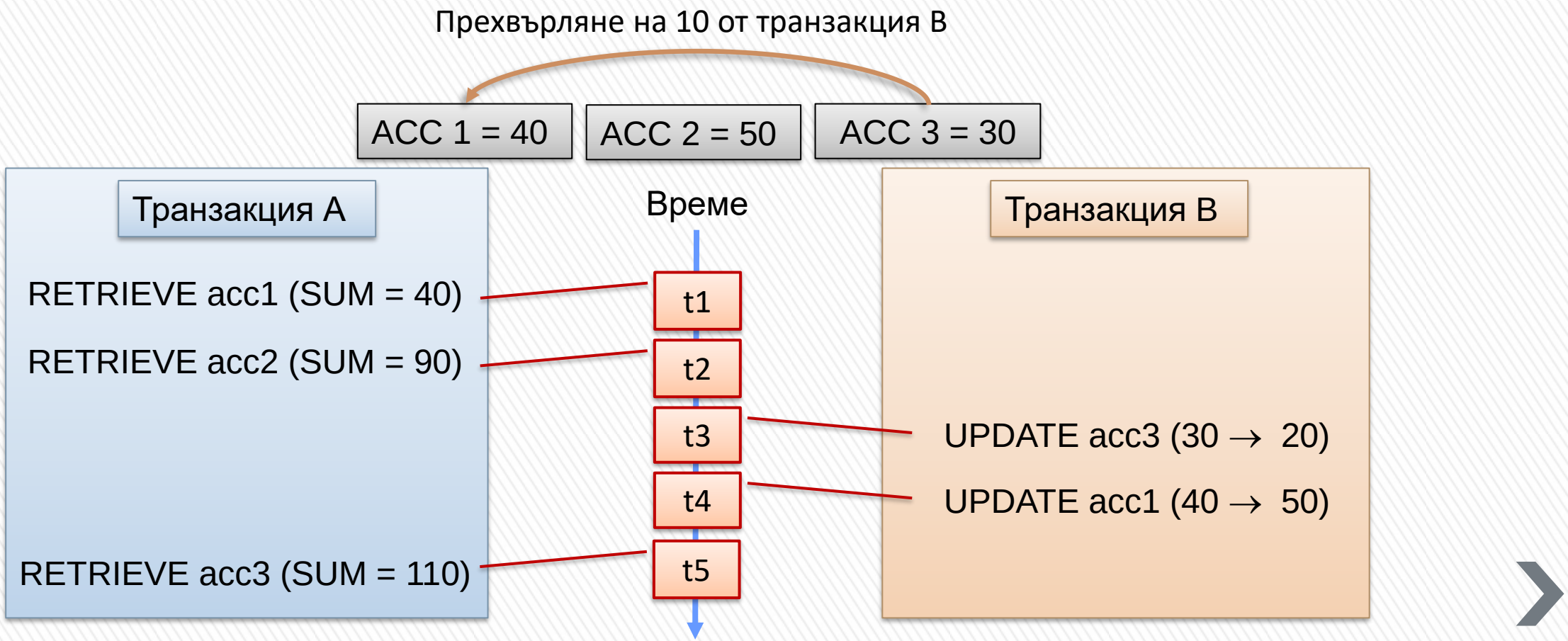
В момент t_3 транзакция A става зависима от незавършената промяна (евентуално губейки и направената промяна в t_2), защото редът p е възстановен до t_0.

Неконсистентен анализ - четене без повторяемост

Неконсистентен анализ възниква, когато едната транзакция осъществява достъп до едни и същи данни няколко пъти и всеки път те са различни:

- Ситуацията е подобна на непотвърдената зависимост - друга транзакция променя в същия момент данните, които се четат от първата;
- Но в този случай четените данни са били потвърдени от променилата ги транзакция.

Неконсистентен анализ - четене без повторяемост



Неконсистентен анализ - четене без повторяемост

- Транзакция A - образува баланса на сметките;
- Транзакция B - прехвърля от сметка 3 към сметка 1 (в случая 10);
- Резултатът, получен от A, е 110 (явно некоректен) - увеличението на сметка 1 с 10 се губи.

Четене на редове-фантоми

Четене на фантоми възниква, когато едно вмъкване или изтриване се изпълнява спрямо ред, който принадлежи на блок от редове, четен от една транзакция:

- Първото четене на блока от редове може да покаже ред, който вече не съществува при второто или следващи прочитания вследствие на изтриването му от друга транзакция;
- По същия начин при вмъкване на ред от друга транзакция второто или следващите четения на транзакцията ще покаже ред, който не е съществувал при първото четене.

Блокиране (locking)

Проблемите на конкурентността могат да бъдат решени чрез една техника на механизма за контрол на конкурентността, наречена блокиране.

Основна идея - когато една транзакция изисква увереност, че един обект няма да бъде променен по време на нейното изпълнение, тя блокира този обект.

Механизъм на блокировка

Най-широко използвания протокол за заключвания *Strict Two-Phase Locking* (Strict 2PL) има следните правила:

1. Ако транзакция A иска да прочете обект тя изисква споделена блокировка (shared lock - S lock) върху този обект.
2. Ако транзакция A иска да промени обект тя изисква извънредна блокировка (exclusive lock - X lock) върху този обект.
3. При приключване на транзакцията всички блокировки се освобождават.

Механизъм на блокировка

Ако транзакция A постави X-lock върху един запис p, тогава всяка заявка за някакъв вид блокировка от друга транзакция върху p се анулира;

Ако транзакция A постави S-lock върху един запис p, тогава:

- искане от B за X-lock върху p се анулира;
- искане от B за S-lock върху p се допуска.

Механизъм на блокировка - обобщение

Тези правила могат да се обобщят посредством матрица за съвместимост:

	X	S	-
X	N	N	Y
S	N	Y	Y
-	Y	Y	

Транзакция A

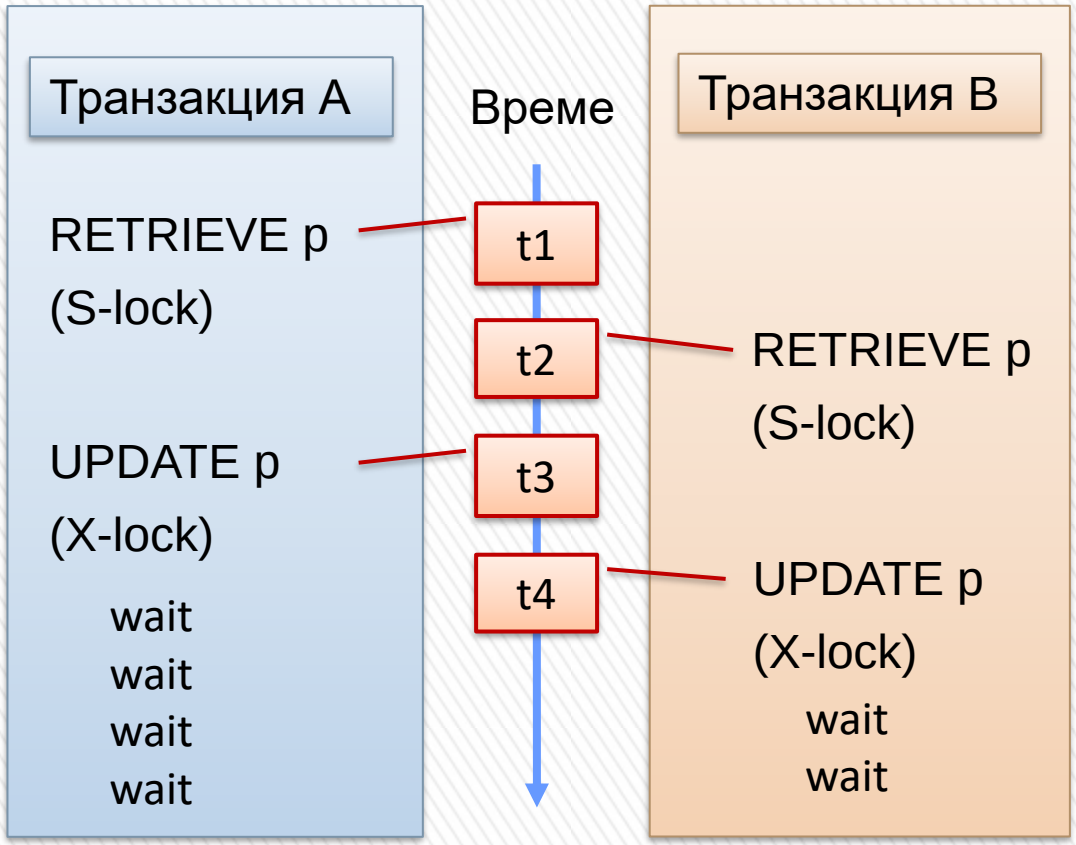
Транзакция B

Протокол за достъп до данните

- Една транзакция A, която иска да чете p, трябва първо да постави S-lock;
- Една транзакция A, която иска да променя p, трябва първо да постави X-lock;
- Ако транзакция B иска блокировка, която не се позволява заради конфликт, тогава тя преминава в състояние на изчакване;
- Блокировките са в сила до края на транзакцията (COMMIT или ROLLBACK).

Решени ли са проблемите?

Загуба на промяна

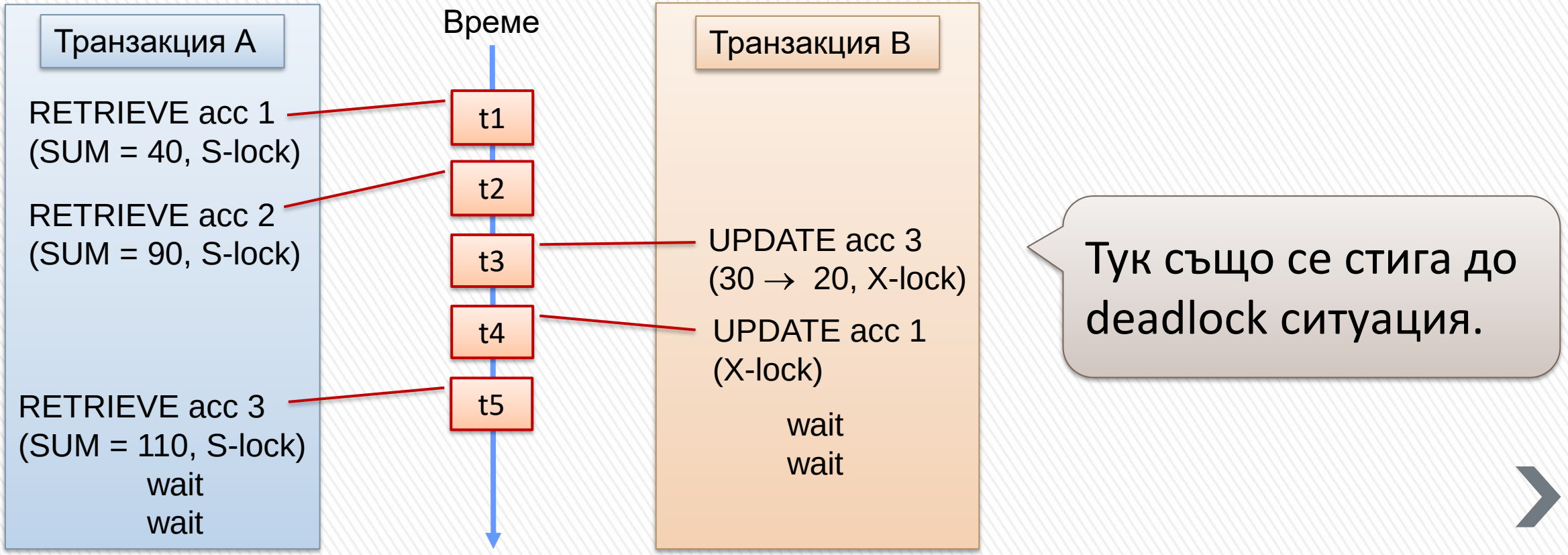


Чрез блокировките вече няма загуба на промяна, но:

- В t3 не може да бъде получена X-lock от A, понеже съществува S-lock от транзакция B – A преминава в изчакване;
- В t4 транзакция B иска X-lock, но не може да я получи, защото съществува S-lock от транзакция A и тогава B преминава в изчакване - възниква deadlock ситуация!

Решени ли са проблемите?

Неконсистентен анализ



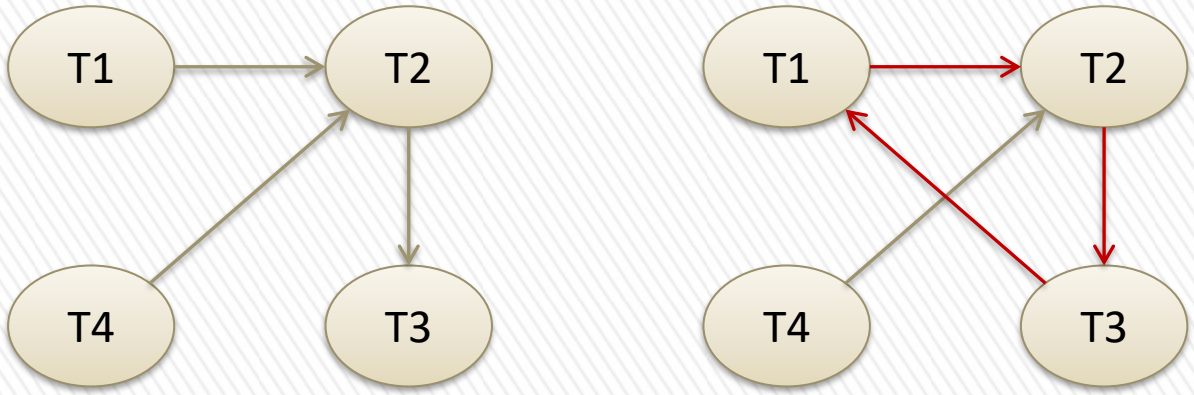
Тук също се стига до deadlock ситуация.

Deadlock

Блокировките решават трите проблема, но същевременно те създават нови.

Deadlock - ситуация, в която две или повече транзакции са едновременно в състояние на изчакване, като при това всяка от тях може да продължи едва когато другата първа предприеме определено действие.

» Ако се появи такава ситуация, желателно е самата система да я открие и да я прекъсне;
» Откриване на deadlock-ситуация означава откриване на цикъл в т.н. wait-for graph, който показва кой за какво чака.



Wait-for граф преди и след deadlock

За създаването на добър механизъм за контрол на конкурентността чрез блокировки трябва да се имат предвид следните няколко въпроса:

- Трябва да се използва предотвратяване или откриване на deadlock ситуации?
- Ако използваме откриване на deadlock колко често трябва да се проверява?
- Ако използваме откриване и идентифицираме deadlock, тогава коя транзакция трябва да бъде прекъсната?

Решение на deadlock-ситуация:

- Една от транзакциите се извежда от ситуацията с ROLLBACK, като същевременно се анулират нейните блокировки;
- Ако това не реши проблема, тогава се извежда втора транзакция и т.н. до решаване на проблема;
- Практически не всички системи поддържат такива механизми - те са свързани със загуба на много време.

Избор на транзакцията-жертва

Когато е открита deadlock ситуация изборът за това коя транзакция да бъде прекъсната може да бъде направен по няколко критерия:

- ✓ Тази с най-малко заключвания;
- ✓ Тази, извършила най-малко работа до момента;
- ✓ Тази, която е най-далеч от приключване на своята работа;
- ✓ И т.н.

Още повече, транзакция може последователно да бъде рестартирана и избирана за жертва на deadlock. Такава транзакция може да бъде фаворизирана при установяване на deadlock и оставена да приключи своята работа.

Основни техники за контрол на deadlock ситуации

1. Предотвратяване – транзакция, искаща нова блокировка, бива прекъсната ако има възможност за изпадане в deadlock ситуация. Всички нейни промени са отхвърлени, а тя е планирана за ново изпълнение. Този метод работи чрез избягване на условията, водещи до deadlock ситуации.
2. Откриване – базата данни периодично претърсва за deadlock ситуации. Ако такава е намерена, едната транзакция се отхвърля (rollback и след това restart).
3. Избягване – транзакцията трябва да получи всички нужни блокировки преди да бъде изпълнена. Тази техника избягва отхвърляне на конфликтни транзакции, но намалява възможностите за едновременно изпълнение на транзакции.

Контрол на конкурентността чрез Time Stamping метод

- Поставя глобални уникални етикети на конкурентните транзакции – стойността на този етикет (timestamp) създава изричен ред на изпълнение на транзакциите;
- Всички операции от една транзакция трябва да имат един и същ етикет; СУБД изпълнява конфликтните операции в реда на етикетите;
- Ако две транзакции са в конфликт, едната бива отхвърлена и планирана за ново изпълнение, като й се присвоява нов етикет;
- Недостатък на тоя метод е нуждата от допълнителни ресурси – за съхранение на всяка стойност в базата се изисква съхранение на две допълнителни стойности – последно извличане и последна промяна; също е възможно много транзакции да бъдат отхвърлени и планирани отново, което увеличава нуждата от системни ресурси.

Wait/Die и Wound/Wait схеми – решение коя е жертвата

» Предполагаме, че имаме две транзакции в конфликт: T1 с етикет 100 и T2 с етикет 200 – т.е. T2 е по-новата от двете.

Изискваща заключване	Притежаваща заключването	Wait/Die схема	Wound/Wait схема
T1 (100)	T2 (200)	T1 чака T2 да завърши и освободи заключените ресурси.	T1 предизвиква rollback на T2, която е планирана отново със същия етикет.
T2 (200)	T1 (100)	T2 бива отхвърлена и планирана отново със същия етикет.	T2 чака T1 да завърши и освободи заключените ресурси.

Контрол на конкурентността чрез оптимистичен метод

- Базира се на допускането, че повечето операции в базата не стигат до конфликт;
- Не изисква заключвания или поставяне на етикети;
- Транзакциите се изпълняват без ограничения докато бивat потвърдени;
- Всяка транзакция преминава през 3 фази – четене, валидация, запис;
- Приемлив за ситуации, в които основните операции са четене на данни, а промените са малко.

Контрол на конкурентността чрез оптимистичен метод

1. **Четене** – прочита данните, прави евентуални изчисления и записва промените в собствени копия на стойностите. Всички update операции се записват във временна памет (файл), до която други транзакции нямат достъп.
2. **Валидация** – транзакцията е валидирана с оглед на това дали не води до неконсистентност на данните. Ако резултатът е положителен транзакцията минава към фаза за запис, иначе промените й се отхвърлят и тя се рестартира.
3. **Запис** – промените от транзакцията се правят постоянни в базата.

Нива на изолация

Нивото на изолация представлява степента, до която една транзакция трябва да бъде изолирана от другите транзакции, т.е. кои от изброените проблеми при четенето ще допуска.

- По-ниското ниво увеличава възможностите за едновременна работа, но за сметка на коректността на данните;
- И обратно, по-високо ниво на изолация гарантира коректност на данните, но може да се отрази отрицателно на едновременната работа.

Нива на изолация

Стандартът SQL-92 дефинира следните нива на изолация:

- Непотвърдено четене (read uncommitted) – най-ниското ниво, на което транзакциите са изолирани само дотолкова, че да не могат да четат физически повредени данни;
- Потвърдено четене (read committed);
- Повторяемо четене (repeatable read);
- Сериализируемо (serializable) – най-високото ниво, на което транзакциите са напълно изолирани една от друга.

Нива на изолация

Ниво на изолация	Незавършена зависимост	Неконсистентен анализ	Четене на фантоми
Непотвърдено четене	Да	Да	Да
Потвърдено четене	Не	Да	Да
Повторяемо четене	Не	Не	Да
Сериализируемо	Не	Не	Не

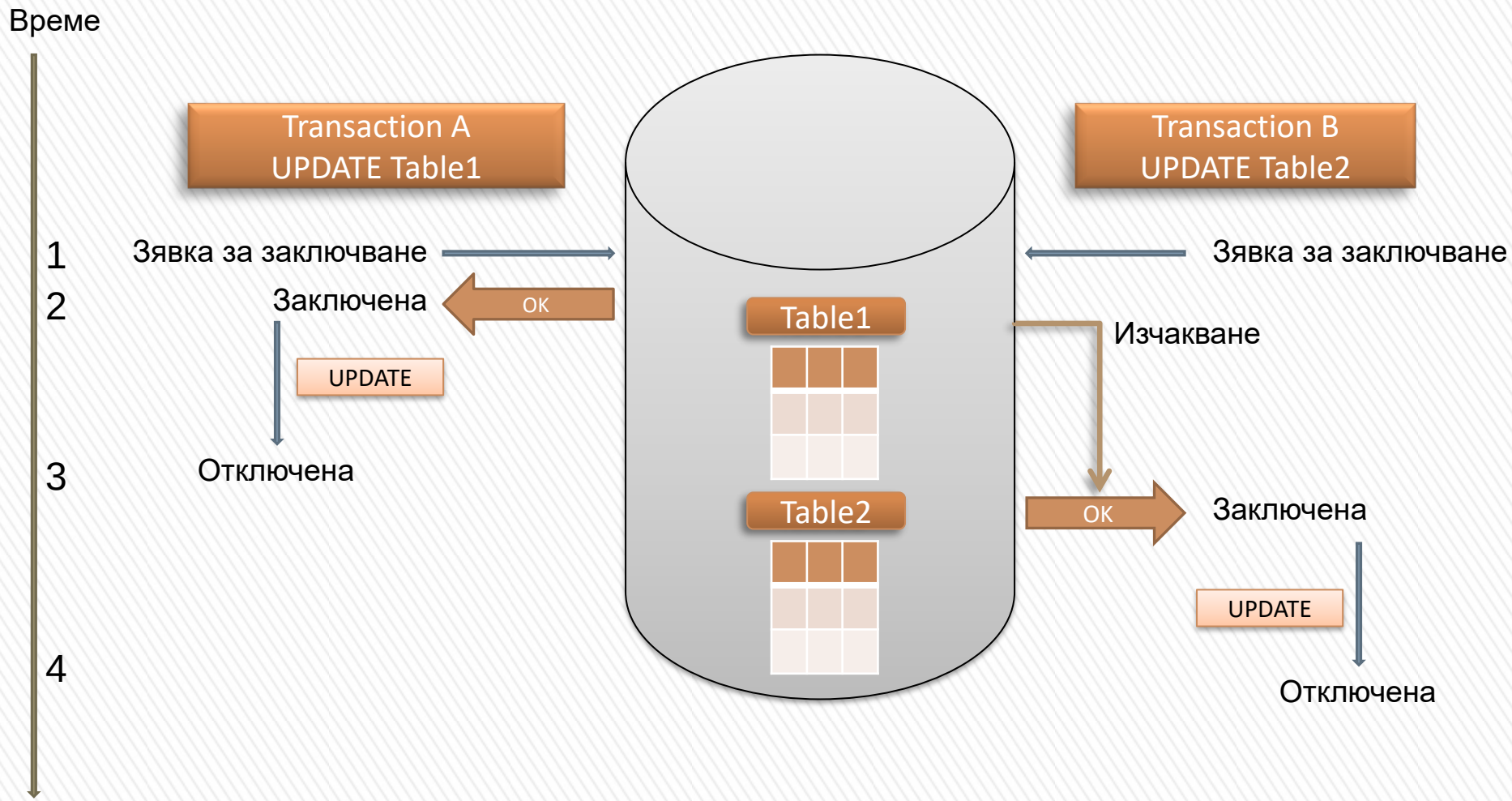
Целенасочено блокиране

- До сега разглеждахме само възможности за блокиране на записи (редове);
- Не съществуват причини блокировките да не могат да се разширят върху другите елементи на базите данни:
  - Цялата БД
  - Цяла таблица
  - Цяла страница
  - Ред
  - Специфичен атрибут
- Колкото по-фина е гранулираността на блокировката, толкова по-висока е степента на конкурентността.

Блокировка на цялата база

- Заключване на цялата база - по този начин докато една транзакция не е приключила друга не може да стартира, дори и да използват различни таблици;
- Удобно за batch processing;
- Неподходящо за многопотребителска online база – би довело до неприемливо забавяне.

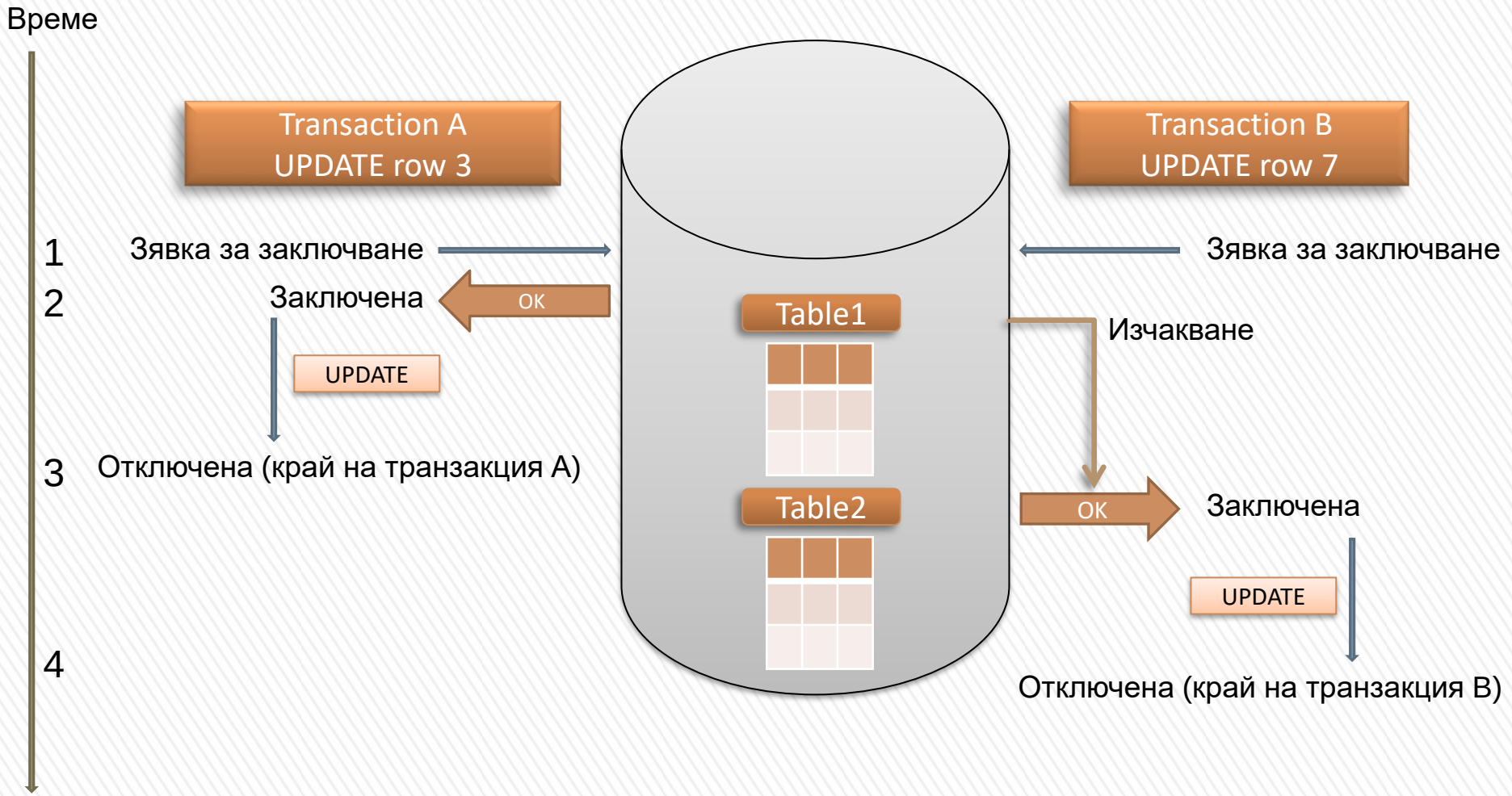
Блокировка на цялата база



Блокировка на таблица

- Заключване на цялата таблица, забранявайки достъп до кой да е ред докато не е приключила транзакцията;
- Две транзакции могат да достъпват базата, стига да използват различни таблици;
- Отново неприложимо за многопотребителска, предизвиква опашка от транзакции, чакащи за различни части от една и съща таблица.

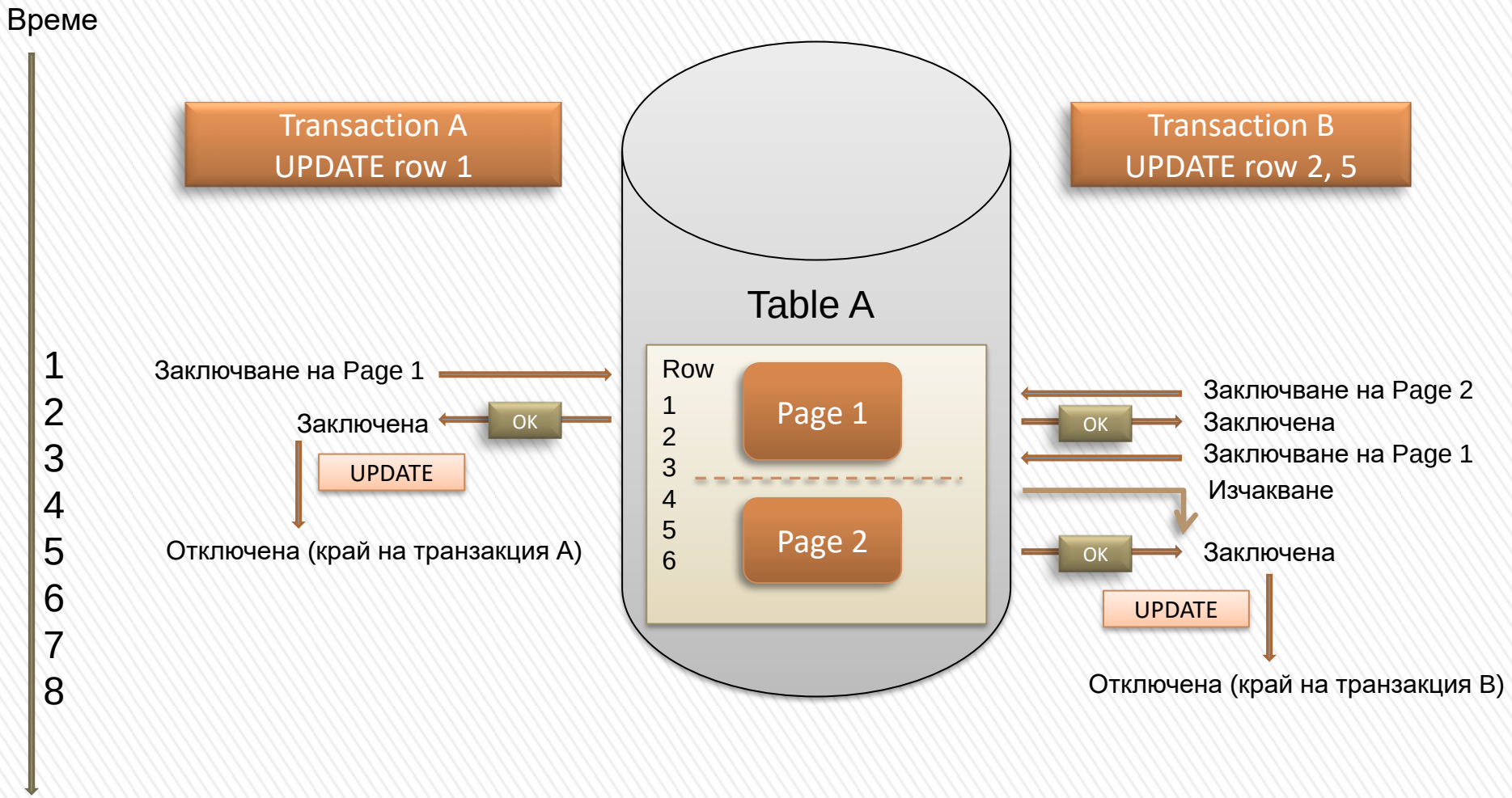
Блокировка на таблица



Блокировка на страница

- Заключване на цяла страница от вторичната памет – секция от паметта (блок от 4K, 8K, 16K…), директно достъпна за четене, промяна и записване обратно в паметта;
- Таблица може да заеме няколко страници, а страница може да съдържа редове от една или повече таблици;
- Най-често използваното ниво на блокировки при многопотребителски бази;
- Ако транзакция B иска достъп до Page 1 ще трябва да изчака транзакция A да завърши, въпреки че променят различни редове.

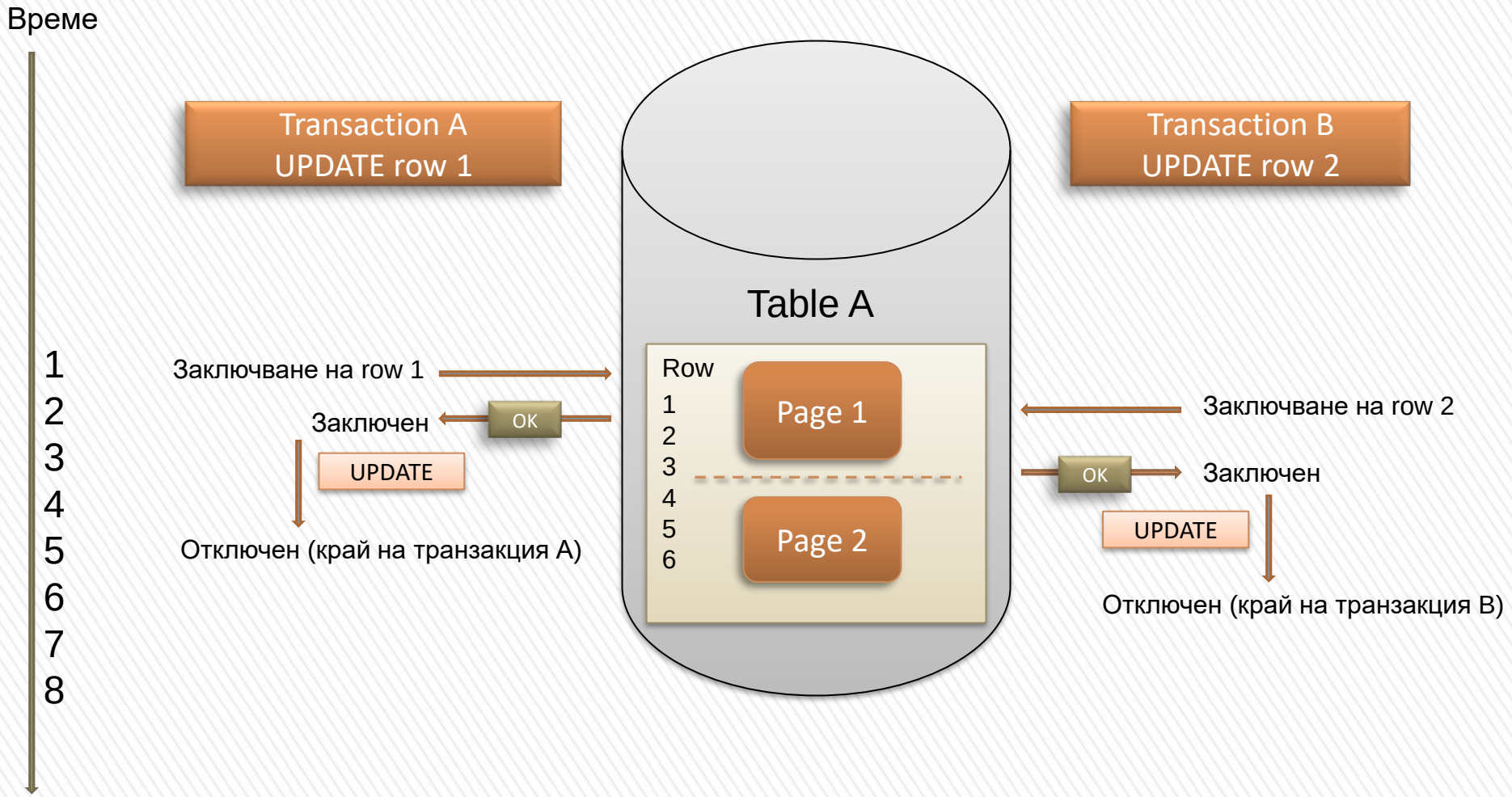
Блокировка на страница



Блокировка на ред

- Най-малко рестриктивно от изброените досега;
- Позволява достъп на транзакции до различни редове, макар и в една страница;
- Доста скъпо от гледна точка на бързодействие – заключване на всеки ред от всяка таблица!

Блокировка на ред



Блокировка на поле

- Позволява достъп на транзакции до един и същ ред, стига те да работят с различни полета от него;
- Най-гъвкав достъп при многопотребителски режим;
- Рядко се реализира заради изключително високата цена от страна на бързодействие.