

Zarządzanie współbieżnością transakcji

Wielowersyjne protokoły synchronizacji,
blokady przedziałowe, protokoły
optymistyczne i wzorce synchronizacji

Ulepszenie protokołów synchronizacji współbieżnych transakcji

- Zwiększenie stopnia współbieżności konfliktowych transakcji
- Wydajna implementacja pełnej uszeregowalności (poziomu izolacji serializable)
- Skalowalność wielodostępu w DBMS
- Zapewnienia własności ACID dla platform programistycznych i komercyjnych DBMS

Ograniczenia protokołu 2PL

- **Zakleszczenia** – duża ilość zmarnowanej pracy, długa obsługa wykrywania zakleszczenia
- **Niewielka współbieżność** dla konfliktowych transakcji. Duża konfliktowość transakcji odczytujących dane.

Matryca konfliktowości operacji:

Operacja transa- kcji T_i \ Operacja transakcji T_j	read(x)	write(x)
read(x)	✓	-
write(x)	-	-

Przykład nieuszeregowalnej historii współbieżnych transakcji T_1 i T_2 :

H: $r_1[x], r_2[x], r_2[y], w_2[x], w_2[y], c_2, r_1[y], c_1$

Wielowersyjne protokoły synchronizacji

- W klasycznych protokołach synchronizacji transakcji - wszystkie modyfikacje danych polegają na zniszczeniu starej wartości danych i zastąpieniu jej nową wartością. Protokoły te są jednowersyjne.
- Wielowersyjne protokoły synchronizacji transakcji działają w ten sposób, że modyfikacje danych nie niszczą zmienianej wartości, lecz tworzą nową *wersję* danej do przechowania nowej wartości. Każda dana w bazie danych oprócz aktualnej wartości przechowuje również zbiory *historycznych* wersji wartości danej:

$$\mathbf{x} = \{ \mathbf{x}_1 = \langle \text{'Tarzan'}, \text{'Portier'}, 1500\text{zł} \rangle, \dots, \\ \mathbf{x}_{n-1} = \langle \text{'Tarzan'}, \text{'Wiceprezes'}, 15000\text{zł} \rangle, \\ \mathbf{x}_n = \langle \text{'Tarzan'}, \text{'Prezes'}, 25000\text{zł} \rangle \}$$

- Wielowersyjne protokoły synchronizacji powinny utrzymywać jedynie minimalne podzbiory wersji danych, niezbędne dla zapewnienia poprawności przetwarzania. Niepotrzebne wersje mogą być usuwane.
- Metody synchronizacji transakcji stosowane w wielowersyjnych bazach danych, mające dostęp do historycznych wersji danych, charakteryzują się większym stopniem współbieżności transakcji.

Wielowersyjne protokoły synchronizacji

Wielowersyjne protokoły synchronizacji współbieżnych transakcji pozwalając na równoległy dostęp do różnych wersji wartości tych samych zmniejszają poziom konfliktowości operacji.

$T_i \backslash T_k$	read(x_i)	write(x_i)
read(x_i)	✓	-
write(x_i)	N/A	N/A
read(x_j)	✓	✓
write(x_j)	✓	✓

Poniższa historia transakcji, niepoprawna z punktu widzenia protokołu 2PL (będzie zmieniona w wyniku działania tego protokołu), w realizacji wielowersyjnej może być uszeregowalna bez zmian kolejności operacji.

H: $r_1[x_0], r_2[x_0], r_2[y_0], w_2[x_1], w_2[y_1], c_2, r_1[y_0], c_1$



odczyt historycznej wersji
wartości danej

Wielowersyjne historie transakcji

Wielowersyjną historią transakcji mh danego zbioru transakcji τ nazywamy częściowo uporządkowany zbiór operacji tych transakcji:

$$mh(\tau) = (\bar{T}_{mh}(\tau), \alpha_{mh}, v)$$

gdzie:

- $\bar{T}_{mh}(\tau) = \bigcup_{T_i \in \tau} \bar{T}_i$ - jest sumą wszystkich operacji wszystkich transakcji ze zbioru τ ;
- $\alpha_{mh} \supseteq \bigcup_{\bar{T}_i} \alpha_i$ - jest relacją częściowego porządku na zbiorze operacji $\bar{T}_{mh}(\tau)$, pokrywającą się z porządkiem operacji w poszczególnych transakcjach.
- $v : \bar{T}_{mh}(\tau) \rightarrow \bar{T}_{mh}(\tau)$ - jest takim odwzorowaniem operacji odczytu $\{r_i(x) \in \bar{T}_{mh}\}$ w zbiór operacji zapisu $\{w_j(x) \in \bar{T}_{mh}\}$, że dla każdej operacji odczytu $r_i(x)$, jeżeli $v(r_i(x)) = w_j(x)$, to $w_j(x) \alpha_{mh} r_i(x)$ (operacje tworzenia wersji muszą poprzedzać operacje ich odczytu).
Odwzorowanie v określa wersje danych odczytywanych przez transakcje.

Uszeregowalność wielowersyjnych historii transakcji

Czy wystarczająca jest poniższa definicja uszeregowalności?

Wielowersyjna historia współbieżnych transakcji jest uszeregowalna, jeżeli jest równoważna (stanowo i obrazowo, albo konfliktowo) choć jednej wielowersyjnej historii sekwencyjnej tych samych transakcji.

Założmy, że baza danych zawiera daną x , której stan początkowy jest równy: $x=(x_0=1000)$. Dane są trzy sekwencyjne transakcje, z których każda zwiększa wartość danej x o 1000: $T_i = \{x = x + 1000\}, i=1,2,3$.

Rozważmy następującą wielowersyjną sekwencyjną historię tych transakcji:

$r_1[x_0], w_1[x_1=x_0+1000], c_1, r_2[x_0], w_2[x_2=x_0+1000], c_2, r_3[x_0],$
 $w_3[x_3=x_0+1000], c_3$

Dla powyższej historii końcowy stan bazy danych będzie następujący:

$x = \{x_0=1000, x_1=2000, x_2=2000, x_3=2000\}$

Powyższa wielowersyjna historia sekwencyjna jest niepoprawna!!!

Potrzebna jest nowa definicja poprawności wielowersyjnych historii sekwencyjnej, które będą mogły służyć za wzorzec poprawności dla historii współbieżnych.

Wielowersyjna uszeregowalność transakcji

Standardowe sekwencyjne historie wielowersyjne

Sekwencyjna historia wielowersyjna jest **standardowa**, jeżeli każda operacja odczytu danej dotyczy ostatnio utworzonej wersji tej danej.

Bardziej formalnie, dla dowolnej operacji odczytu $r_i[x]$ nie istnieje operacja zapisu $w_j[x]$ taka, że spełniony byłby warunek:

$$v(r_i[x]) \alpha_{mh} w_j[x] \wedge w_j[x] \alpha_{mh} r_i[x]$$

Kryterium uszeregowalności wielowersyjnej

Wielowersyjna historia $mh(\tau)$ zbioru transakcji τ jest uszeregowalna wtedy i tylko wtedy, gdy jest ona równoważna dowolnej standardowej sekwencyjnej historii wielowersyjnej zbioru tych transakcji.

Wielowersyjne protokoły synchronizacji współbieżnych transakcji

- Na bazie definicji uszeregowalności wielowersyjnej zaproponowano kilka wielowersyjnych protokołów synchronizacji współbieżnych transakcji. Protokoły te są konstruowane jako rozwinięcie znanych jednowersyjnych protokołów synchronizacji:
 - dwuwersyjny protokół blokowania dwufazowego,
 - wielowersyjny protokół znaczników czasowych.

Dwuwersyjny protokół 2PL

Operacje modyfikacji danych tworzą nowe wersje danych. W danym momencie może istnieć tylko jedna dodatkowa wersja danej.

Zatwierdzanie transakcji modyfikujących wymaga zastąpienia starej wersji danej – nową. Operacja ta jest nazywana **certyfikacją wersji danych**. Operacja certyfikacji wymaga nowego typu blokady.

Matryca kompatybilności blokad, obejmująca blokadę certyfikacji, wygląda następująco:

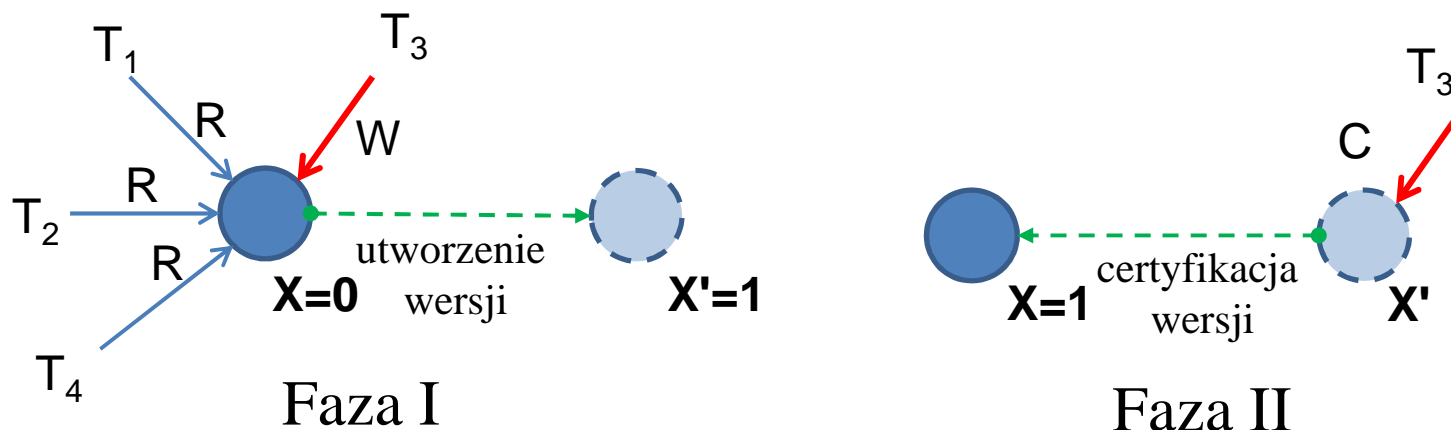
blokada założona \ blokada żądana	blokada założona		
	R	W	C
R	✓	✓	-
W	✓	-	-
C	-	N/A	N/A

Blokady certyfikacji z jednej strony charakteryzują się wysokim stopniem konfliktowości, z drugiej są to blokady krótkoterminowe. Są one zakładane tylko na czas zatwierdzania transakcji.

Dwuwersyjny protokół 2PL

Niech przykładowa dana występuje w pojedynczej wersji X . Dana ta zostaje odczytana przez transakcje T_1 i T_2 . Kolejna transakcja T_3 dokonuje modyfikacji danej X , w wyniku czego jest tworzona nowa wersja tej danej - X' . Kolejna operacja odczytu danej przez transakcję T_4 jest adresowana do zatwierdzonej wersji danej – X . Powyższe operacje tworzą pierwszą fazę przetwarzania danej.

Zatwierdzenie transakcji T_3 wymaga uzyskania dla tej transakcji blokady do certyfikacji. Uzyskanie tej blokady będzie możliwe dopiero po zakończeniu wszystkich transakcji odczytujących. Próba modyfikacji danej przez jedną z transakcji odczytujących, będzie prowadzić do zakleszczenia.



Dwuwersyjny protokół 2PL

Algorytm odczytu wersji danej

Read(x, tid) begin

B: if (LOCK(x, tid) = **C**)

// odczyt konfliktowy tylko z certyfikacją

// nie ma konfliktu z modyfikacjami

 then begin

 < wstaw żądanie do kolejki i czekaj >;

 go to B;

 end;

 else begin

 LOCK(x, tid) := **R**;

 < czytaj zatwierdzoną wersję x >;

 end;

end **Read**;

Dwuwersyjny protokół 2PL

Algorytm zapisu wersji danej

Write(x, tid) begin

B: if (LOCK(x,tid) = \emptyset or LOCK(x,tid) = **R**)

// Zapis nie jest konfliktowy z odczytem

 then begin LOCK(x, tid) := **W**;

 < twórz nową wersję x' >;

 end;

 else begin

 <wstaw żądanie do kolejki i czekaj>;

 go to B;

 end;

end **Write**;

Dwuwersyjny protokół 2PL

Algorytm certyfikacji niezatwierdzonej wersji danej

Certify(x, tid) begin

B: if (LOCK(x,tid) = \emptyset) then begin

 LOCK(X, tid) := **C**;

 <zamień starą wersję x na x'>;

 end;

 else begin

 < wstaw żądanie do kolejki i czekaj>;

 go to B;

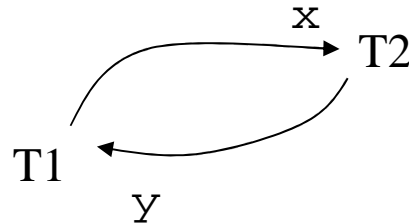
 end;

end **Certify**;

Dwuwersyjny protokół 2PL

Dana jest nieuszeregowalna historia transakcji:

$H: r_1[x], r_2[x], r_2[y], w_2[x], w_2[y], r_1[y], c_1, c_2$



Klasyczny algorytm 2PL uporządkuje powyższą historię następująco:

$H_{sv}: r_1[x], r_2[x], r_2[y], r_1[y], c_1, w_2[x], w_2[y], c_2$

Uzyskana historia charakteryzuje się niskim stopniem współbieżności.
Przetwarzanie transakcji T_2 jest zawieszone do zakończenia transakcji T_1 .

Wielowersyjny algorytm 2PL uporządkuje powyższą historię następująco:

$H_{mv}: r_1[x_0], r_2[x_0], r_2[y_0], w_2[x_1], w_2[y_1], r_1[y_0], c_1, c_2$

Powyższa historia charakteryzuje się wyższym stopniem współbieżności.

Wielowersyjny protokół znaczników czasowych

Klasyczne protokoły znaczników czasowych przypisują każdej transakcji w systemie unikalny znacznik czasowy **TS** reprezentujący moment/kolejność rozpoczęcia się transakcji. Transakcje przetwarzając dane zostawiają na nich swoje ślady, przez składowanie przy danych swoich znaczników czasowych. Z każdą daną związane są dwa znaczniki: **Write_TS** - znacznik czasowy transakcji, która jako ostatnia modyfikowała daną i **Read_TS** - znacznik czasowy najmłodszej transakcji, która czytała tę daną.

Wielowersyjny protokół znaczników czasowych wymaga utrzymywania dla każdej danej zbioru jej wersji. Każda z wersji ma przypisane swoje własne znaczniki czasowe.

Na przykład:

$$\mathbf{x} = \{\mathbf{x}_1:(1;5), \mathbf{x}_2:(8;10), \mathbf{x}_3:(13;18), \mathbf{x}_4:(19;19)\}$$

oznacza, że dana \mathbf{x} występuje w czterech wersjach, utworzonych przez transakcje o znacznikach czasowych 1, 8, 13 i 19, i odczytanych odpowiednio przez transakcje o znacznikach czasowych 5, 10, 18 i 19.

Wielowersyjny protokół znaczników czasowych

Algorytm odczytu danej protokołu wielowersyjnego zamienia odwołanie do generycznej danej na odczyt konkretnej wersji tej danej. Odczytywana jest najmłodsza wersja z podzbioru wersji danej utworzonych przez transakcje starsze od transakcji odczytującej tę daną. Operacje odczytu są zawsze poprawne – nie są nigdy przyczyną wycofania transakcji.

Read(T_i, x) begin

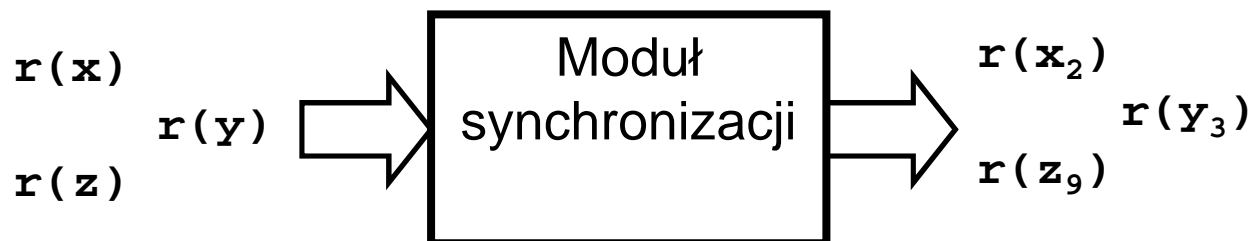
 < czytaj x_k , takie że

Write_TS(x_k) = $\max\{ \textbf{Write_TS}(x_j) : \textbf{Write_TS}(x_j) < \textbf{TS}(T_i) \}$ >;

 if (**Read_TS**(x_k) < **TS**(T_i)) then

Read_TS(x_k) = **TS**(T_i);

end **Read**;



Wielowersyjny protokół znaczników czasowych

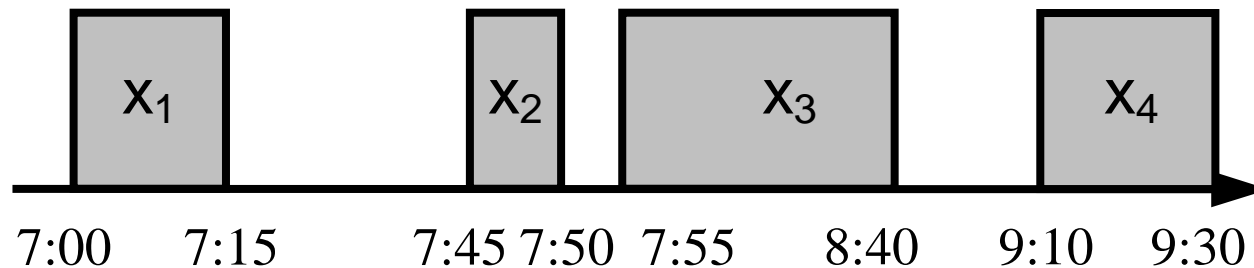
Algorytm modyfikacji danych tworzy nowe wersje danych. Operacja modyfikacji jest powodem odrzucenia transakcji, gdy w zbiorze wersji danej istnieje wersja x_i taka, że znacznik czasowy transakcji modyfikującej spełnia zależność:

$$\text{Write_TS}(x_i) < \text{TS}(T_k) < \text{Read_TS}(x_i)$$

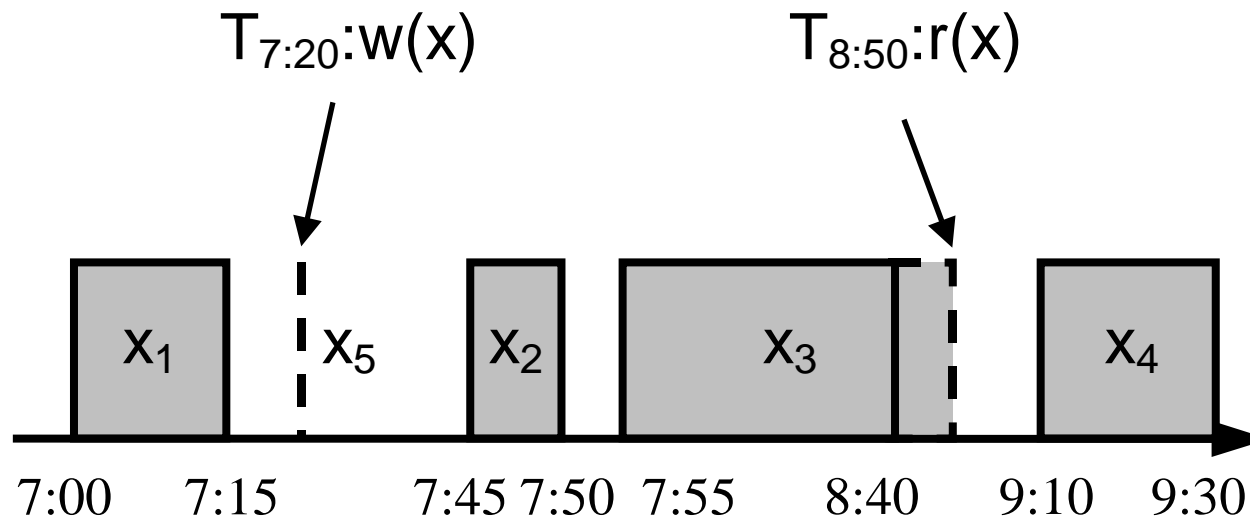
```
Write( $T_i$ ,  $x$ ) begin
    if (istnieje  $x_k$ , takie że Write_TS( $x_k$ ) < TS( $T_i$ ) < Read_TS( $x_k$ ))
    then
        < wycofaj  $T_i$  i powtórnie uruchom ją z nowym
        znacznikiem czasowym >;
    else begin
        < utwórz  $x_k$  >;
        Write_TS( $x_k$ ), Read_TS( $x_k$ ) = TS( $T_i$ );
    end;
end Write;
```

Wielowersyjny protokół znaczników czasowych

Dana jest dana x , posiadająca cztery wersje wartości: x_1 utworzoną przez transakcję z godziny 7:00 i odczytaną przez transakcję z 7:15, itd..

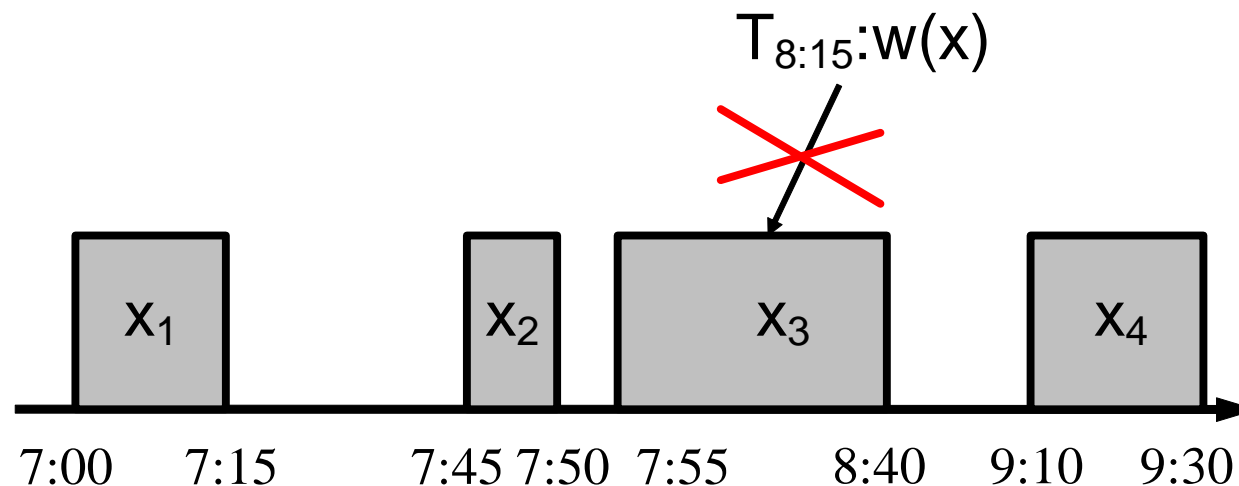


Następnie modyfikację tej danej wykonała transakcja z 7:20, a odczyt transakcja z 8:50:



Wielowersyjny protokół znaczników czasowych

Przykładem konfliktu, który będzie wymagał wycofania transakcji jest próba modyfikacji danej x przez transakcję z 8:15. Operacja ta nie może być wykonana poprawnie. Transakcja z 8:40 zdążyła już odczytać wersję danej utworzoną przez transakcję z 7:55. Poprawną może być jedynie historia, w której transakcja z 8:40 czyta wersję z 8:15 albo historia bez transakcji z 8:15, czyli historia, w której transakcja z 8:15 jest wycofana.



Przedstawiony protokół jest modyfikacją podstawowego protokołu znaczników czasowych (*basic timestamp ordering*), który pozwala na brudne odczyty danych.

Protokół synchronizacji w DBMS Oracle

DBMS Oracle używa zmodyfikowanego wielowersyjnego protokołu znaczników czasowych. Zastosowane rozwiązanie używa jedynie znaczników czasowych Write_TS.

Dla poziomu izolacji *read committed* znaczniki czasowe są przypisywane nie transakcjom, ale pojedynczym poleceniom SQL (*read consistency*). Dla izolacji *serializable* znaczniki czasowe są przypisywane transakcjom (*transaction consistency*).

Blokady wyłączone do zapisu mają charakter techniczny. Są stosowane, aby uniemożliwić utrzymywanie w bazie danych dwóch niezatwierdzonych danych. Starsze wersje danych są przechowywane w wyróżnionej przestrzeni systemowej (*rollback segment*). Rollback segment zawiera wyłącznie zatwierdzone wersje danych. Baza danych zawiera dokładnie jedną, zatwierdzoną lub nie, wersję danych.

Dodatkowo, do eliminacji anomalii *lost update*, jest stosowana metoda *first committer wins*. Metoda wycofuje konfliktową operację zapisu po zatwierdzeniu transakcji, która jako pierwsza uzyskała dostęp do danej. W wypadku wycofania transakcji blokującej, transakcja oczekująca dostanie blokadę i będzie kontynuowana.

Anomalie fantomów

- Model obliczeniowy przyjęty w definicjach poprawności współbieżnego wykonania transakcji przyjmuje, że przetwarzanymi przez transakcje strukturami danych są rekordy (wiersze) identyfikowane, niezależnie od ich wartości, przez systemowe identyfikatory i przetwarzane za pomocą operacji odczytu i zapisu.

$r_1[x], w_1[x], \dots, c_1$
odczyt, zapis danej o identyfikatorze x

- Zaletą tego modelu jest jego uniwersalność. Może być stosowany dla różnych logicznych modeli danych.
- Jego wadą jest to, że nie uwzględniając semantyki logicznego modelu danych, nie umożliwia weryfikacji pewnej klasy anomalii współbieżnego wykonania. Dla relacyjnego modelu danych, nie identyfikowaną klasą anomalii są fantomy.

Anomalie fantomów

- Przykład anomalii fantomów:

Transakcja T1

```
select * from Rezerwacje
where nr_pokoju = 102
and data_do < '1-4-2012'
and data_od > '10-4-2012';
```

-- brak konfliktowych rezerwacji
-- żadne dane nie są odczytane

```
if not exists
```

```
insert into Rezerwacje
values (102, '1-4-2012', ...);
```

Transakcja T2

```
select * from Rezerwacje
where nr_pokoju = 102
and data_do < '30-3-2012'
and data_od > '5-4-2012';
```

```
if not exists
```

```
insert into Rezerwacje
values (102, '30-3-2012', ...);
```

-- mimo weryfikacji wpisanie niespójnych danych, konfliktowe rezerwacje

Anomalie fantomów

- W modelu obliczeniowym, służącym do weryfikacji uszeregowalności transakcji, przedstawiona historia współbieżnych transakcji wygląda następująco:

`/* puste odczyty */,w1[x],w2[y],c1,c2`

- Fantomy są wynikiem niewykrycia na poziomie operacji odczytów i zapisów rekordów danych, konfliktowych operacji wykonywanych w modelu relacyjnym. Takie przypadki, występują dla operacji SELECT, UPDATE i DELETE zawierających warunki selekcji oraz operacji INSERT.

- Na przykład:

```
delete from pracownicy  
where etat='Prezes';
```

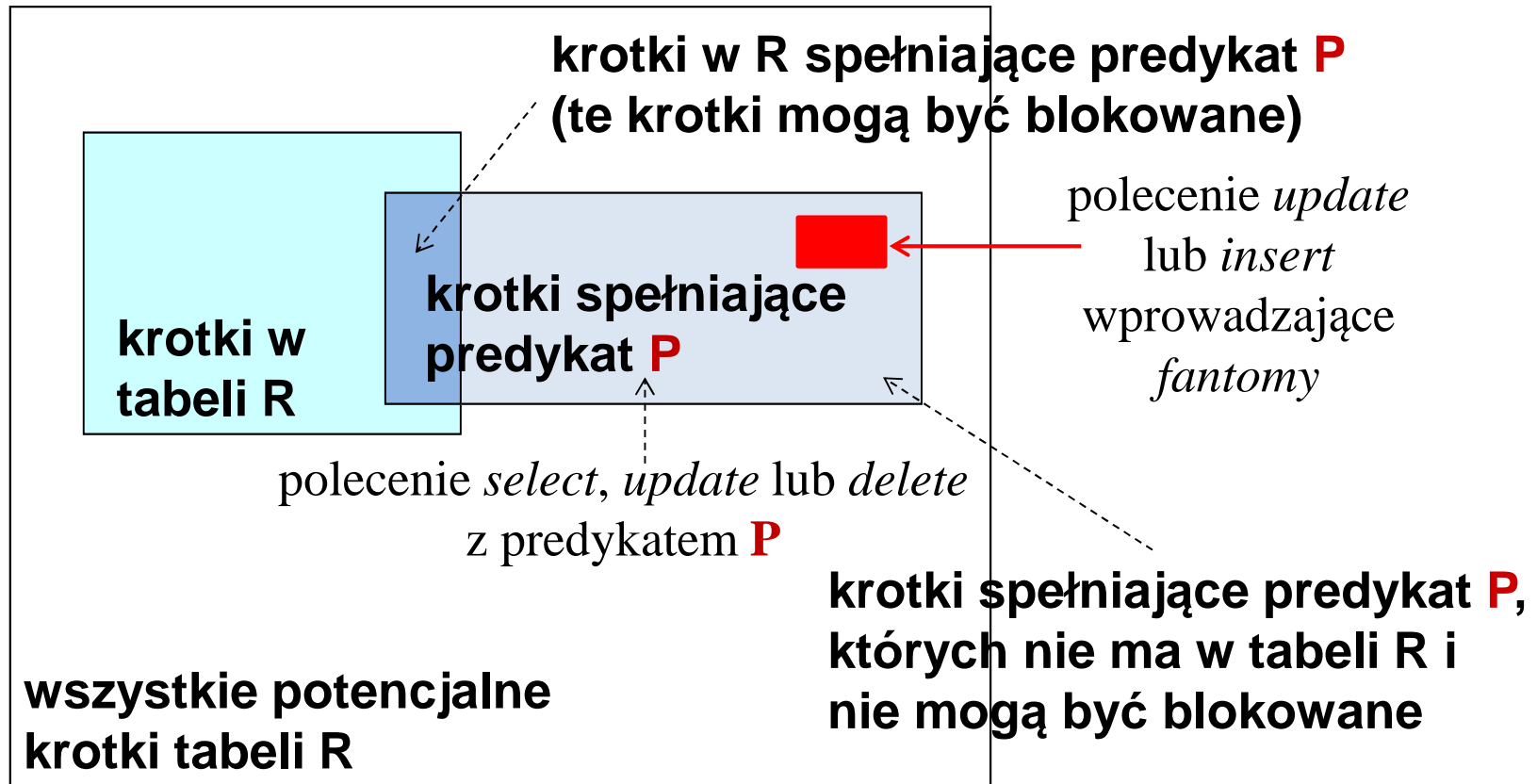
```
/* konflikt operacji relacyjnych */  
/* nie wykryty na poziomie rekordów */
```

```
update pracownicy  
set etat='Prezes'  
where id_prac=102
```



Anomalie fantomów

- Fantomy są konsekwencją faktu, że warunki logiczne użyte w poleceniach SQL są spełnione nie tylko przez krotki składowane w danym stanie w bazie danych, ale również przez krotki, których nie ma w danym stanie bazy danych



Implementacja poziomu izolacji *serializable*

- DBMS **DB2** – odczyt lub modyfikacja danych dla poziomu izolacji *serializable* powoduje automatyczne blokowanie całych tabel. Gwarantuje to poprawność przetwarzania, ale istotnie ogranicza współbieżność równoległych transakcji.

```
select * from Rezerwacje
where nr_pokoju = 102
and data_do < '1-4-2012'
and data_od > '10-4-2012';
```

```
/* R[Rezerwacje] */
```

```
if not exists
insert into Rezerwacje
values (102,'1-4-2012',...);
```

```
/* w[inserted tuple] */
```

```
/* conflict and wait */
```

```
-- deadlock detection and one transaction rollback
```

```
-- R[Rezerwacje]
```

```
select * from Rezerwacje
where nr_pokoju = 102
and data_do < '30-3-2012'
and data_od > '5-4-2012';
```

```
if not exists
```

```
-- w[inserted tuple] conflict and wait
```

```
insert into Rezerwacje
values (102,'30-3-2012',...);
```

Implementacja poziomu izolacji *serializable*

- DBMS **Oracle** - systemowo eliminuje jedynie niewielki podzbiór fantomów; za pomocą znaczników czasowych gwarantowana jest jedynie stałość zbioru odczytywanych danych (*repeatable read*) ze względu na fantomy.

```
select * from Rezerwacje
where nr_pokoju = 102
and data_do < '1-4-2012'
and data_od > '10-4-2012';
```

```
if not exists
```

```
insert into Rezerwacje
values (102, '1-4-2012', ...);
commit;
```

```
/* powyższa zatwierdzona -> */
/* rezerwacja nie zostanie -> */
/* wyświetlona -> */
```

```
/* wstawienie konfliktowej -> */
/* rezerwacji -> */
```

```
select * from Rezerwacje
where nr_pokoju = 102
and data_do < '30-3-2012'
and data_od > '5-4-2012';
```

```
select * from Rezerwacje
where nr_pokoju = 102
and data_do < '30-3-2012'
and data_od > '5-4-2012';
```

```
if not exists
```

```
insert into Rezerwacje
values (102, '30-3-2012', ...);
```

Blokowanie predykatów - "predicate lock"

Jednym z rozwiązań wykrywania konfliktów między operacjami relacyjnymi jest wprowadzenie nowej jednostki blokowania – predykatów, tj. warunków logicznych operatora selekcji w poleceniach relacyjnych: *select*, *update* i *delete*. Format nowego typu blokad jest następujący:

`<identyfikator transakcji, typ blokady, warunek logiczny>`

Typy blokad są analogiczne jak dla pojedynczych danych: blokada współdzielona dla operacji *select* i blokady wyłączone dla operacji *update* i *delete*.

Dwie blokady predykatowe:

`<tidi, blokadai, predykati>` i `<tidj, blokadaj, predykatj>`

są kompatybilne jeżeli:

- $tid_i = tid_j$, bo transakcja nie jest konfliktowa sama ze sobą;
- obydwie blokady są współdzielone (do odczytu);
- Nie istnieje dana, która spełnia jednocześnie obydwa predykaty, bo wyrażenie logiczne $(predykat_i \cap predykat_j)$ jest fałszywe.

Blokowanie predykatów - "predicate lock"

Polecenie: `select * from Rezerwacje
where nr_pokoju = 102
and data_do < '1-4-2012' and data_od > '10-4-2012';`

spowoduje założenie blokady współdzielonej na predykacie:

$\sigma(\text{Rezerwacje})_{[\text{nr_pokoju} = 102 \text{ and } \text{data_od} \geq '1-4-2012' \text{ and } \text{data_do} \leq '10-4-2012']}$

Próba wykonania polecenia:

`insert into Rezerwacje values (102, '30-3-2012', '30-3-2012'),`

zostanie zablokowana ze względu na, że wstawiana krotka spełnia powyższy predykat, więc wystąpi konflikt między blokadą do zapisu wstawianej krotki, a blokadą założoną na predykacie.

Trudnościami w wydajnej implementacji przedstawionej metody są:

- **koszt weryfikacji** konfliktowości blokad zakładanych na predykatach; problem spełnialności jest NP-zupełny;
- nadmierny ***pesymizm*** metody; brak kompatybilności predykatów dla dowolnego stanu bazy danych, nie oznacza braku kompatybilności dla danego stanu bazy danych

$\sigma(\text{Rezerwacje})_{\text{nr_pokoju} = 102}$? $\sigma(\text{Rezerwacje})_{\text{typ_pokoju} = 'apartament'}$

Blokady przedziałowe - "key-range locks"

Potrzeba nowych mechanizmów synchronizacji transakcji dla zapewnienia pełnej uszeregowalności, przy wydajnej implementacji

- Blokowanie pojedynczych wierszy
 - Nie zapobiega anomalii fantomów
 - Generuje historie nieuszeregowalne
 - Ma zadowalający stopień współbieżności
- Blokowanie całych tabel
 - Zapobiega anomalii fantomów
 - Generuje historie uszeregowalne
 - Ma niski stopień współbieżności
- Blokowanie predykatów
 - Zapobiega anomalii fantomów
 - Generuje historie uszeregowalne
 - Zadowalający stopień współbieżności
 - Kosztowna implementacja

Blokady przedziałowe

- Zbiór blokowanych typów obiektów (krotki, strony dyskowe, indeksy, relacje, schematy, bazy danych) jest rozszerzony o przedziały wartości (*key range*).
- Zbiór wartości danego atrybutu jest dzielony na rozłączne przedziały, np. atrybut płaca może być podzielony na przedziały:
 $\langle 1250; 1570 \rangle, \langle 1570; 2100 \rangle, \dots, \langle 7890; 57100 \rangle$
- Na tak zdefiniowane przedziały wartości mogą być zakładane standardowe blokady: wyłączna i współdzielona.

Zapytanie: `select * from pracownicy
where płaca between 1650 and 2000`

założy blokadę na przedziale wartości $\langle 1570; 2100 \rangle$.

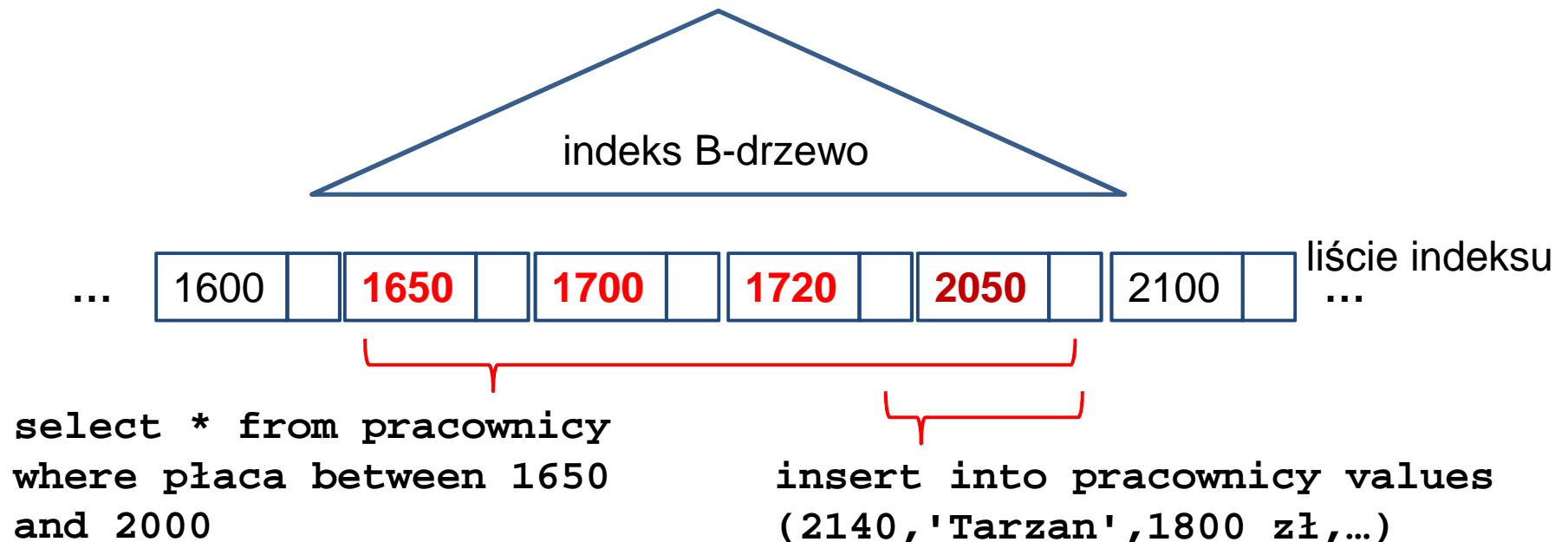
Uniemożliwi to współbieżne wykonanie w trakcie działania transakcji, która zawiera to zapytanie, np. operacji:

`insert into pracownicy values (2140, 'Tarzan', 1800 zł, ...)`

Wartość 1800 zł zawiera się w zablokowanym przedziale: $\langle 1570; 2100 \rangle$.

Blokady przedziałowe – SQL Server

W DBMS SQL Server włączenie trybu izolacji *serializable*, uruchamia mechanizm blokad przedziałowych. Blokada przedziałowa są implementowane przez blokowanie kluczy w indeksach zakładanych na atrybutach, do których odwołują się predykaty logiczne poleceń SQL. Blokowane są wszystkie klucze spełniające warunek logiczny (n) i dodatkowo jeden następny (1) (*next key value*): w sumie n+1 kluczy.



Blokady przedziałowe – SQL Server

Korzystanie z blokad przedziałowych wymaga odpowiednich umiejętności od programisty aplikacji baz danych. Blokada przedziałowa będzie poprawnie utrzymywana gdy:

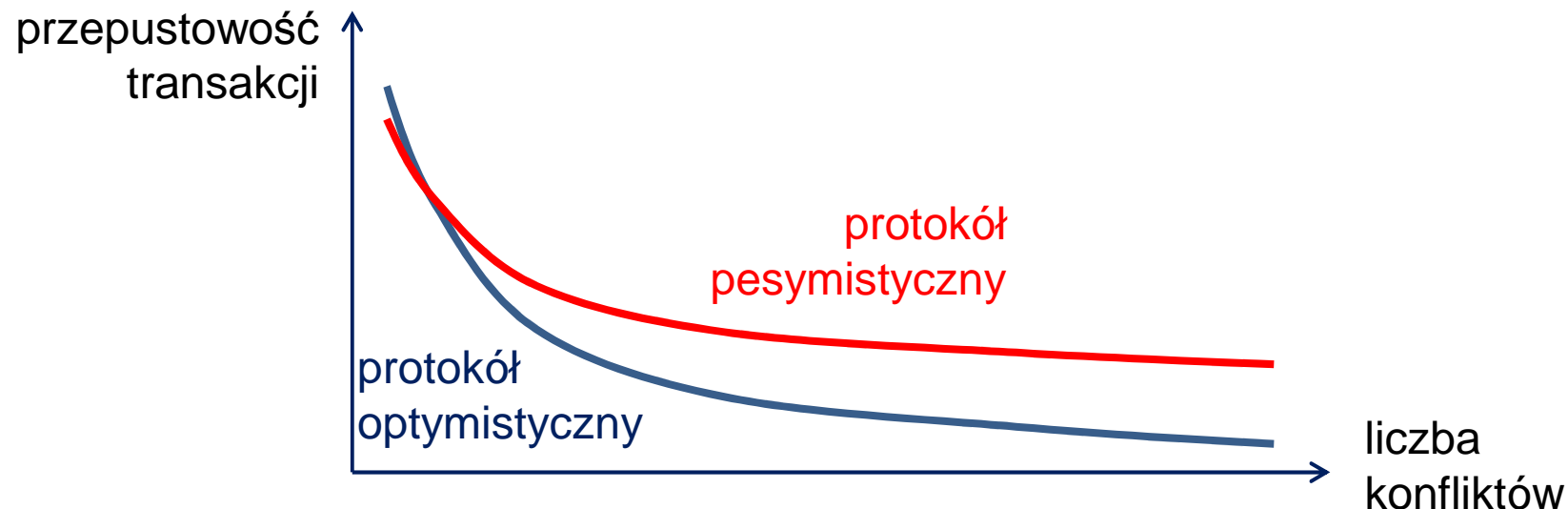
- na atrybutach występujących w warunkach selekcji są założone indeksy;
- plan zapytania uwzględnia wykorzystanie tych indeksów;

Jeżeli na tabeli nie utworzono indeksu, do którego odwołują się warunki selekcji, blokowane są klucze indeksu podstawowego.

Brak indeksu podstawowego skutkuje blokowaniem całej tabeli.

Optymistyczna synchronizacja transakcji

- W większości protokołów gwarantujących poprawność współbieżnego przetwarzania transakcji implementowane jest *pesymistyczne* podejście do synchronizacji, polegające na modyfikacji historii konfliktowych transakcji przez czasowe zawieszenie działania jednej z nich.
- Optymistyczne protokoły synchronizacji transakcji przesuwają wykrywanie potencjalnych konfliktów do czasu akceptacji transakcji. Wykryte konflikty są obsługiwane przez wycofanie jednej z konfliktowych transakcji.



Optymistyczna synchronizacja transakcji

- Schemat protokołu optymistycznej synchronizacji współbieżności zakłada przeprowadzenie każdej transakcji przez trzy kolejne fazy:
 - Faza odczytów i lokalnych modyfikacji – potencjalnie długa i obejmująca interakcje z użytkownikiem transakcji.
 - Faza walidacji, która zaczyna się w punkcie akceptacji transakcji i służy do weryfikacji możliwość poprawnego zakończenia transakcji.
 - Synchronizowana transakcyjnie faza zapisu danych buforowanych w PAO do bazy danych.
- Zaletami optymistycznej synchronizacji transakcji są: niewystępowanie w tym protokole zakleszczeń transakcji oraz mniejsze obciążenie zasobów DBMS (brak blokad).
- Optymistyczna synchronizacja transakcji nie gwarantuje pełnej uszeregowalności – nie obsługuje anomalii fantomów.

Optymistyczna synchronizacja transakcji

Algorytm fazy walidacji jest następujący [H.T.Kung, J.T.Robinson – 1981]:

Validation (T_j)

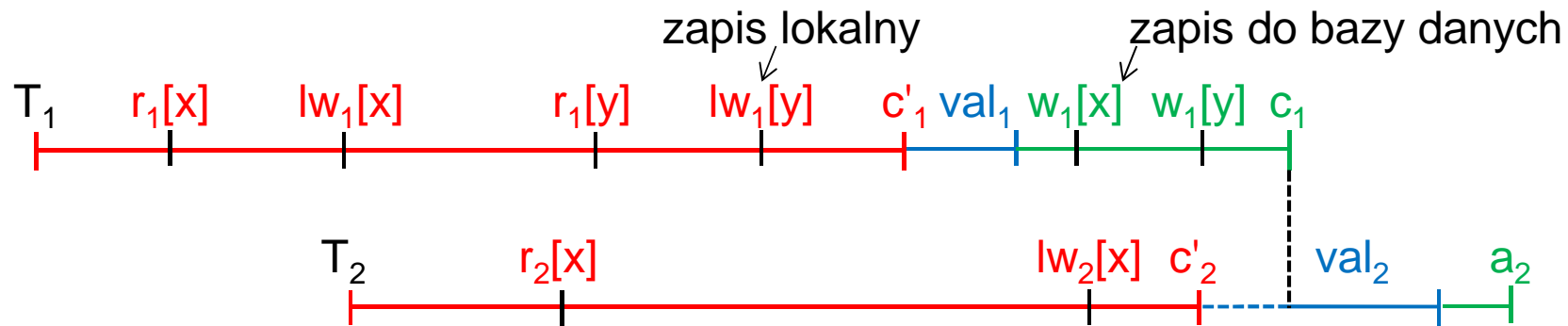
```
<valid := true;    // początek sekcji krytycznej
  for  $TNR_i$  from  $TNR_{start+1}$  to  $TNR_{finish}$  do
    if  $RS_j \cap WS_i \neq \emptyset$  then valid:=false;
  if valid then
  begin
    (write);
     $TNR_j := TNC$ ;
     $TNC := TNC+1$ 
  end> // koniec sekcji krytycznej
  if not valid then (rollback);
```

gdzie:

TNR_{start} jest największym znacznikiem transakcji w momencie startu transakcji T_j , TNR_{finish} jest największym znacznikiem transakcji w momencie rozpoczęcia fazy walidacji, a RS i WS są zbiorami czytanych i modyfikowanych danych.

Działanie algorytmu

Przykład synchronizacji dwóch transakcji T_1 i T_2 .



$$TNR(T_1) = 10$$

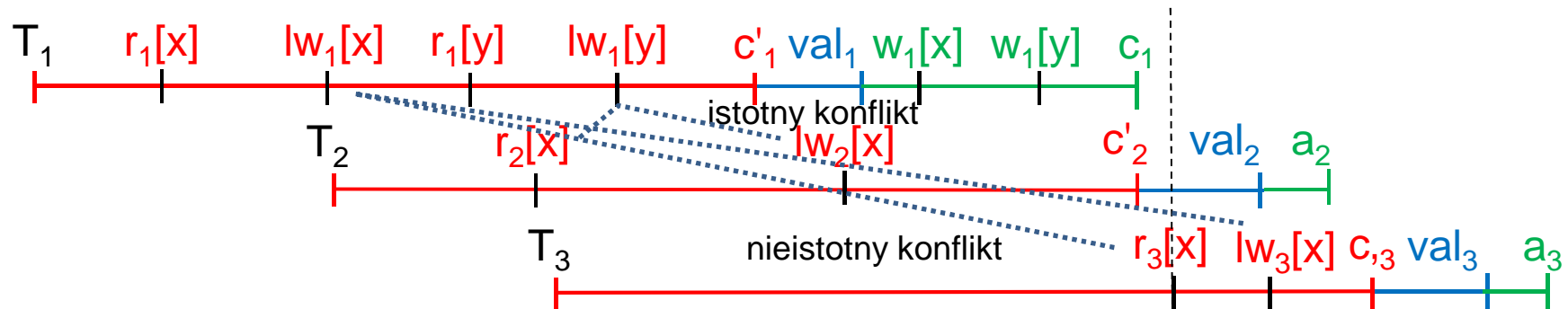
Walidacja transakcji T_2 :

- $TNR_{start+1} = 10$, $TNR_{finish} = 10$
- $RS(T_2) = \{x\}$, $WS(T_1) = \{x, y\}$,
- skąd: $RS(T_2) \cap WS(T_1) = \{x\} \neq \emptyset$

Transakcja T_2 musi zostać wycofana.

Ulepszona wersja algorytmu

- Pomijanie nieistotnych konfliktów



- Transakcja T_3 zostanie wycofana, mimo że dysponuje zatwierdzoną wartością danej x . Wykryty konflikt jest nieistotny dla poprawności przetwarzania, ale jest przyczyną wycofania transakcji.
- Modyfikacja protokołu OC o rozszerzenie zbioru danych czytanych przez transakcję o znaczniki końców współbieżnych transakcji. Dla transakcji T_3 , zbiór czytanych danych $RS_3 = \{EOT_1, x\}$.
- Weryfikowany w fazie walidacji warunek zostanie zmodyfikowany do postaci: $RS_{ij} \cap W_i \neq \emptyset$, gdzie RS_{ij} jest podzbiorem tylko tych danych, które zostały odczytane po zakończeniu transakcji T_i . $RS_{13} = \emptyset$, co znaczy, że w zmodyfikowanym protokole transakcja T_3 nie zostanie wycofana.

Protokoły optymistyczne w aplikacjach internetowych

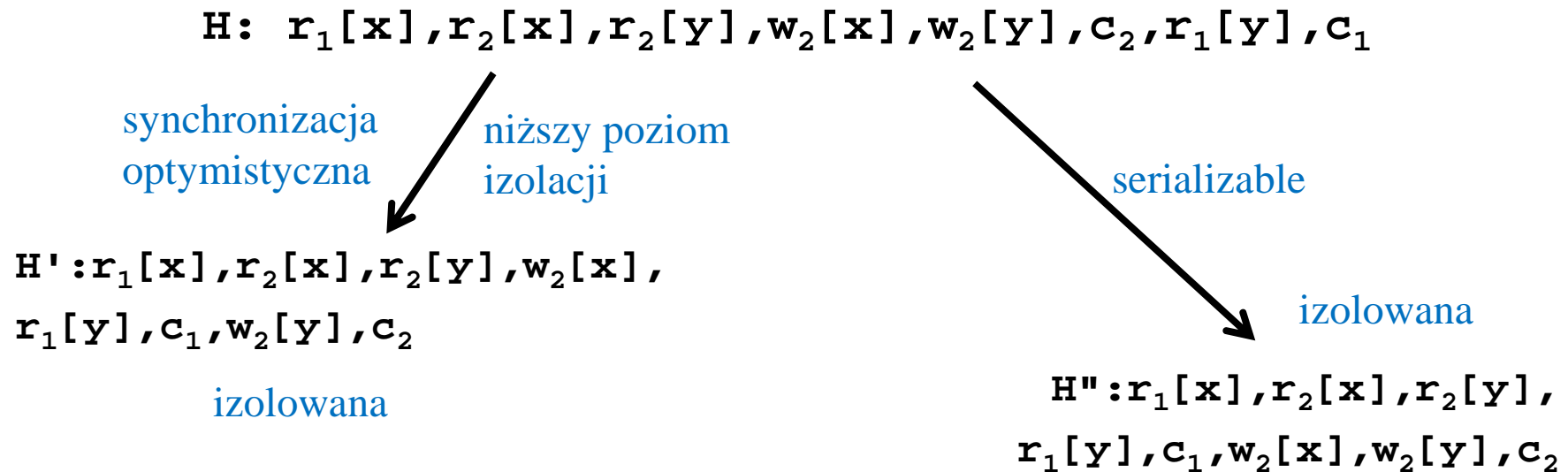
W internetowych aplikacjach baz danych sugerowanym protokołem synchronizacji jest protokół optymistyczny. Jednak zbieżność nazwy z klasycznymi protokołami optymistycznymi jest dość przypadkowa. W środowiskach projektowania internetowych aplikacji baz danych (.NET, JPA, itd.), proponowane protokoły charakteryzują się następującymi cechami:

- unikanie zakładania blokad na dane – dla operacji odczytu;
- minimalizacja czasu zakładania blokad – dla operacji odczytu i zapisu;
- implementacja transakcji biznesowej/aplikacyjnej jako zbioru transakcji systemu bazy danych – dla minimalizacji liczby połączeń;
- weryfikacja odczytanej i lokalnie modyfikowanej wersji danej przed jej ponownym zapisem, w celu uniknięcia anomalii *lost update*;
- zapisy lokalnie modyfikowanych danych w kilku transakcjach bazy danych; JPA przewiduje dwa tryby aktywacji procesu *flush*: AUTO przed zapytaniami i COMMIT.

Przy powyższych założeniach transakcje aplikacyjne nie posiadają żadnej z cech ACID.

Poprawne zarządzanie transakcjami

- Niższe poziomy izolacji transakcje i optymistyczna synchronizacja transakcji są możliwe do przyjęcia w sytuacjach, gdy gwarantują generowanie uszeregowalnych historii, przy ograniczeniu narzutów systemowych.
- Tylko historie współbieżnych transakcji, które są tak samo poprawnie synchronizowane dla niższego poziomu izolacji albo w schemacie optymistycznym, jak przez kompletną transakcję bazy danych w zarządzaną w trybie *serializable*, tylko bardziej wydajnie.



Wzorce obsługi transakcji

Określone cechy transakcji wymagają od programistów świadomego używania określonych systemowych mechanizmów transakcyjnych.

- Wyznaczanie przetworzonych wartości na podstawie kilku zapytań SQL - wymagany poziom izolacji wyższy niż READ COMMITTED.
- Logika przetwarzania zawierająca warunki wykonania bazujące na odczytanych wartościach danych i podejmowanie na ich podstawie decyzji o operacjach zapisów - wymagany poziom izolacji wyższy niż READ COMMITTED.
- Anomalia skrótnego zapisu – SELECT ... FOR UPDATE
- Więcej niż jedna para operacji odczytów i zapisów, kilka zapisów - potrzeba unikania zakleszczeń przez odpowiedni kod aplikacji.
- Hot spoty (nieduże podzbiory intensywnie przetwarzanych danych) – unikanie wyłącznego dostępu do tych danych przez odpowiedni kod aplikacji.
- Duża liczba konfliktów – tryb NO WAIT.
- Fantomy – jawne blokowanie całych tabel (LOCK TABLE), zastępcze blokowanie *zewnętrznych* rekordów typu *master*.