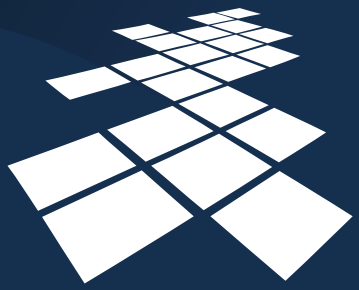
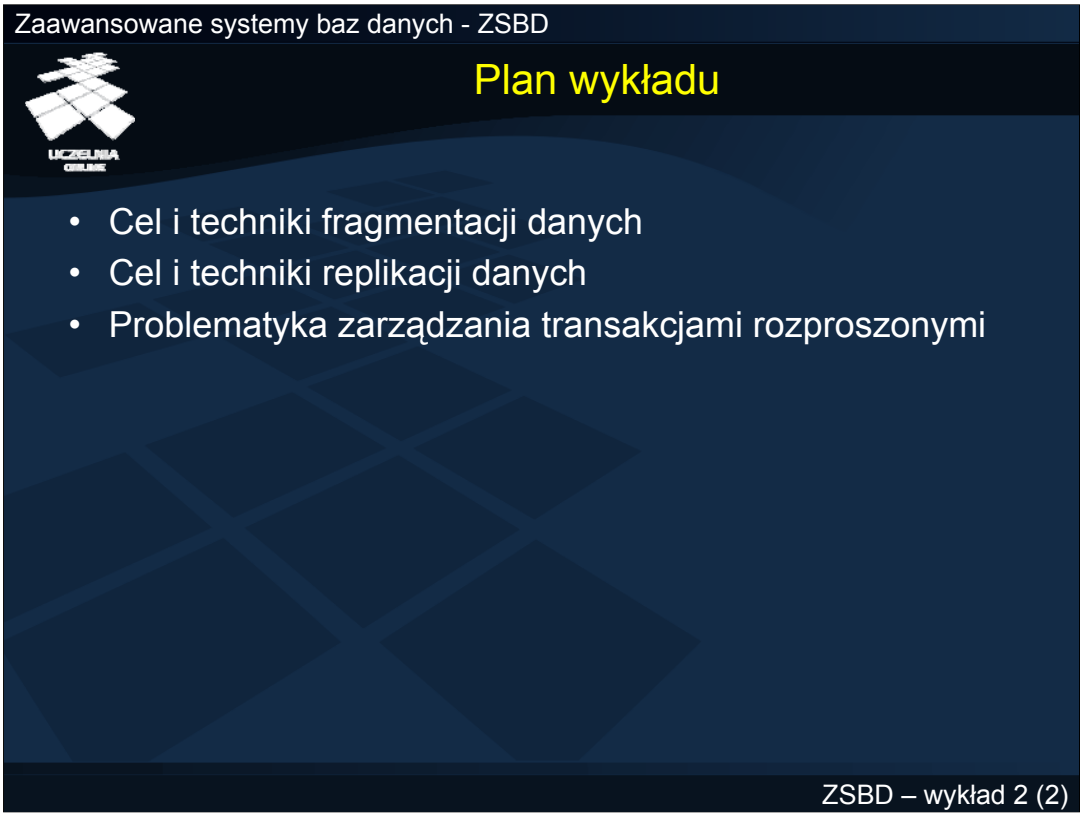
Systemy rozproszonych baz danych – 2

Fragmentacja, replikacja, zarządzanie transakcjami

Wykład przygotował:
Robert Wrembel



UCZELNIA
ONLINE



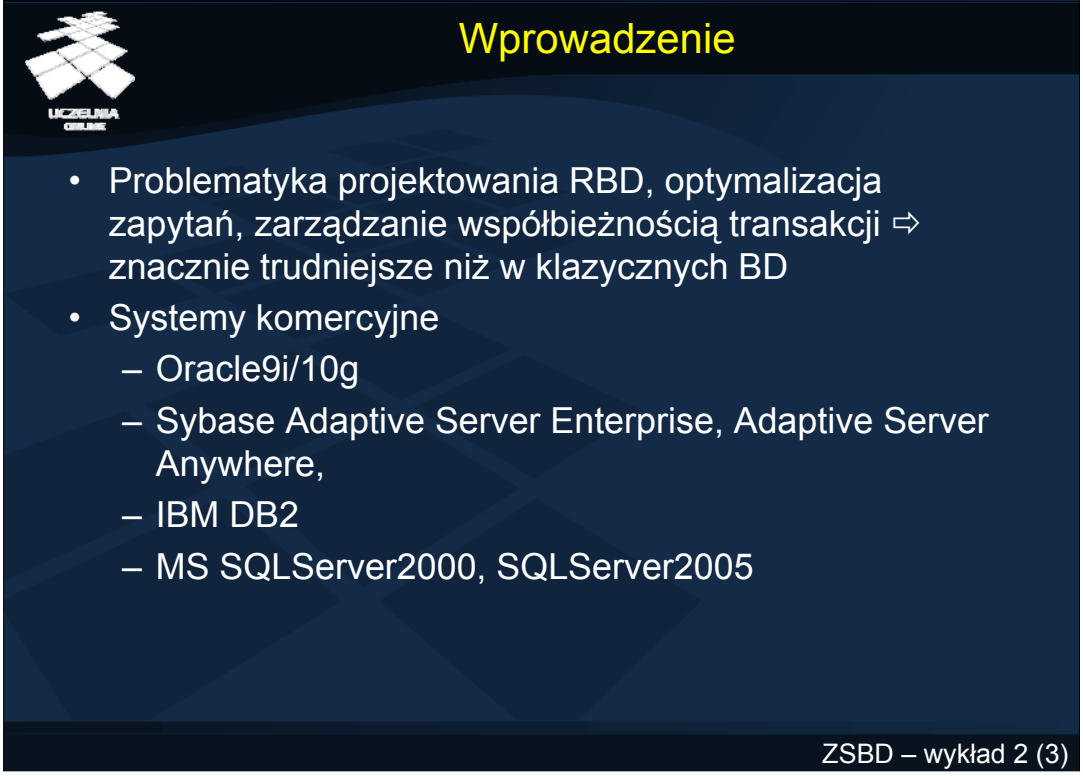
Celem wykładu jest omówienie podstawowych zagadnień związanych z implementacją systemu rozproszonej bazy danych. Omówione zostaną następujące zagadnienia:

- cel i techniki fragmentacji danych (tzw. partycjonowanie),

- cel i techniki replikacji danych,

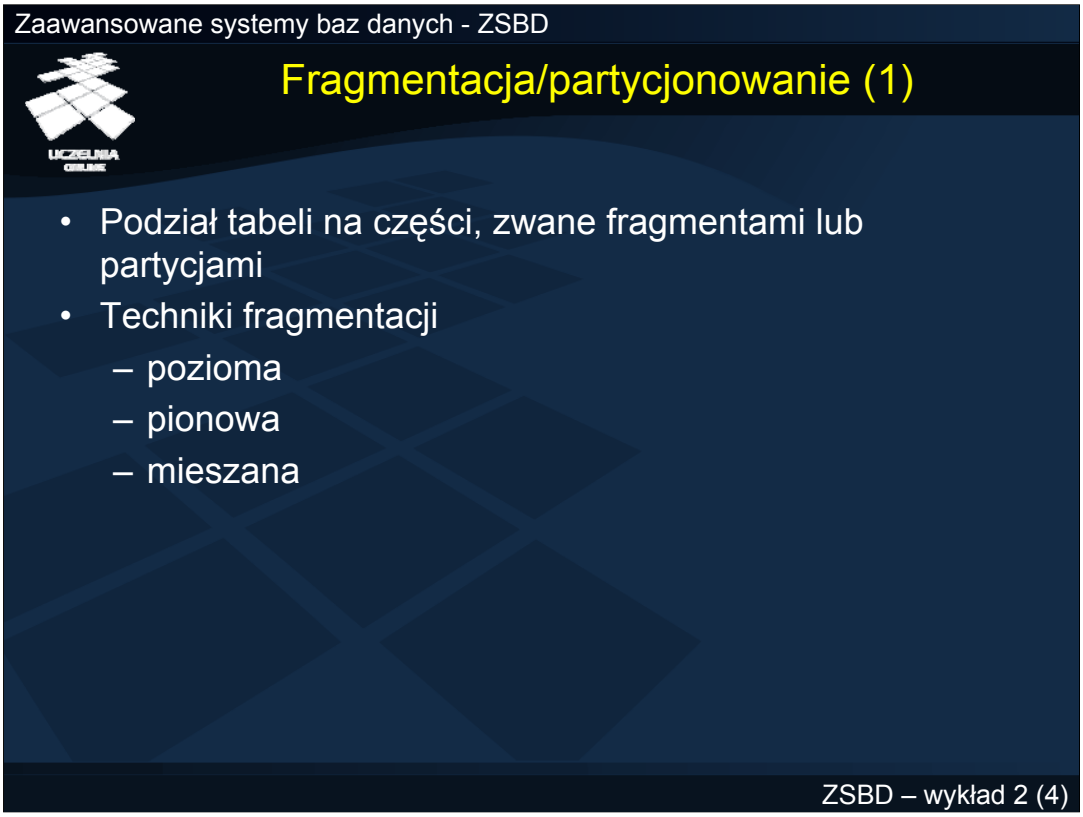
- problematyka zarządzania transakcjami rozproszonymi.

Problematyka optymalizacji zapytań rozproszonych zostanie omówiona w ramach laboratorium nr 3, wraz z praktyczną ilustracją konkretnych technik w systemie Oracle9i/10g.



Wiele problemów związanych z projektowaniem i zarządzaniem scentralizowanymi bazami danych, m.in. projektowanie struktury bazy danych, przetwarzanie i optymalizacja zapytań, zarządzanie współbieżnością transakcji, staje się znacznie trudniejsze w przypadku baz rozproszonych.

Dostępne na rynku komercyjne SZBD (Oracle9i/10g, Sybase Adaptive Server Enterprise i Sybase Adaptive Server Anywhere, IBM DB2, MS SQLServer2000 i SQLServer2005) oferują mniej lub bardziej zaawansowaną funkcjonalność wymaganą przy budowie systemów RBD.



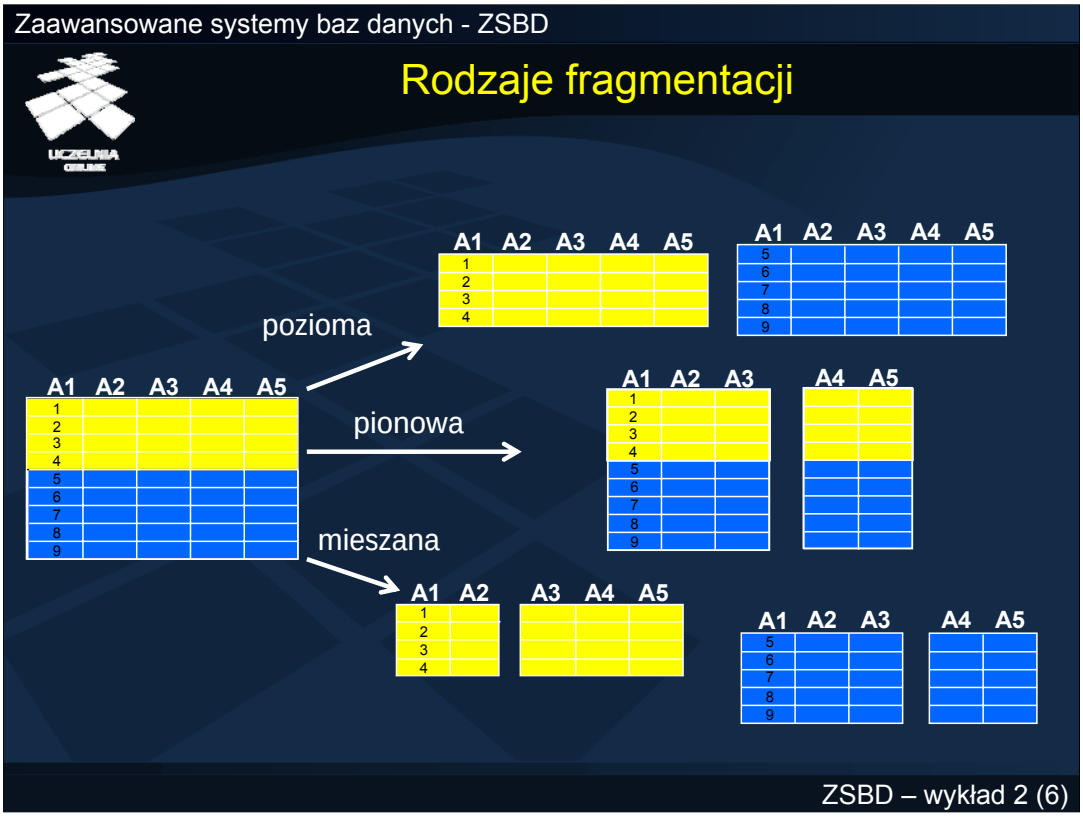
Jednym z podstawowych zagadnień projektowania RBD jest określenie sposobu podziału danych pomiędzy węzły, tak aby efektywność całego systemu była zadowalająca. Jedną z technik stosowanych w tym zakresie jest fragmentacja i alokacja.

Fragmentacja polega na podziale obiektu przechowującego dane, najczęściej tabeli, na mniejsze części, zwane fragmentami lub partycjami. W praktyce wyróżnia się trzy techniki fragmentacji, tj. fragmentację poziomą, pionową i mieszaną.

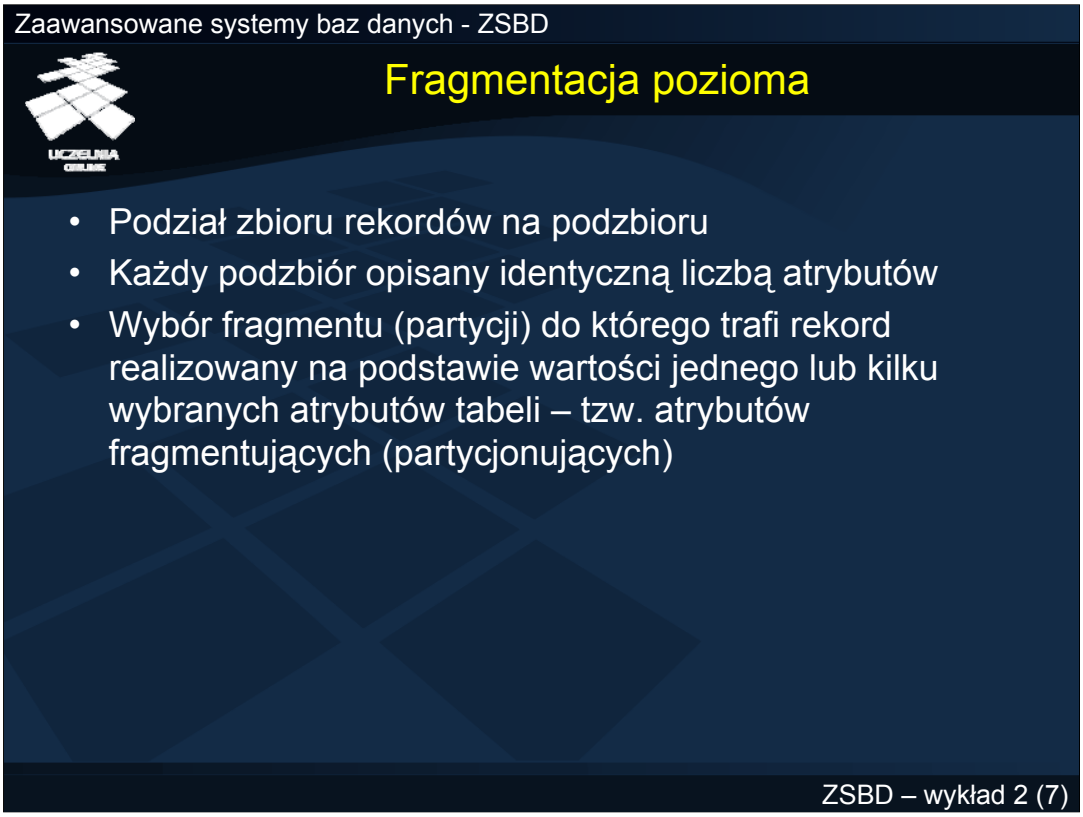
Fragmentacja/partycjonowanie (2)

- Cel: zwiększenie efektywności dostępu do danych
  - zmniejszenie rozmiaru danych, które należy przeszukać
  - zrównoleglenie operacji dostępu do dysków, na których umieszczono fragmenty
  - alokowanie fragmentów "blisko" miejsca ich wykorzystania - redukcja kosztów transmisji sieciowej

Celem fragmentacji jest zwiększenie efektywności dostępu do danych. Dzięki właściwemu doborowi kryterium podziału obiektów na fragmenty można zapewnić, że aplikacje użytkowników będą adresowały tylko wybrane interesujące je partycje, w ten sposób ograniczając wolumeny przeszukiwanych danych. Ponadto, każdy z fragmentów może być umieszczony na innym dysku, co zapewni równoległy dostęp do wielu fragmentów równocześnie. Jeżeli fragmenty zostaną umieszczone "blisko" miejsca ich wykorzystania, wówczas można zredukować koszty transmisji sieciowej w sieciach o niskiej przepustowości.

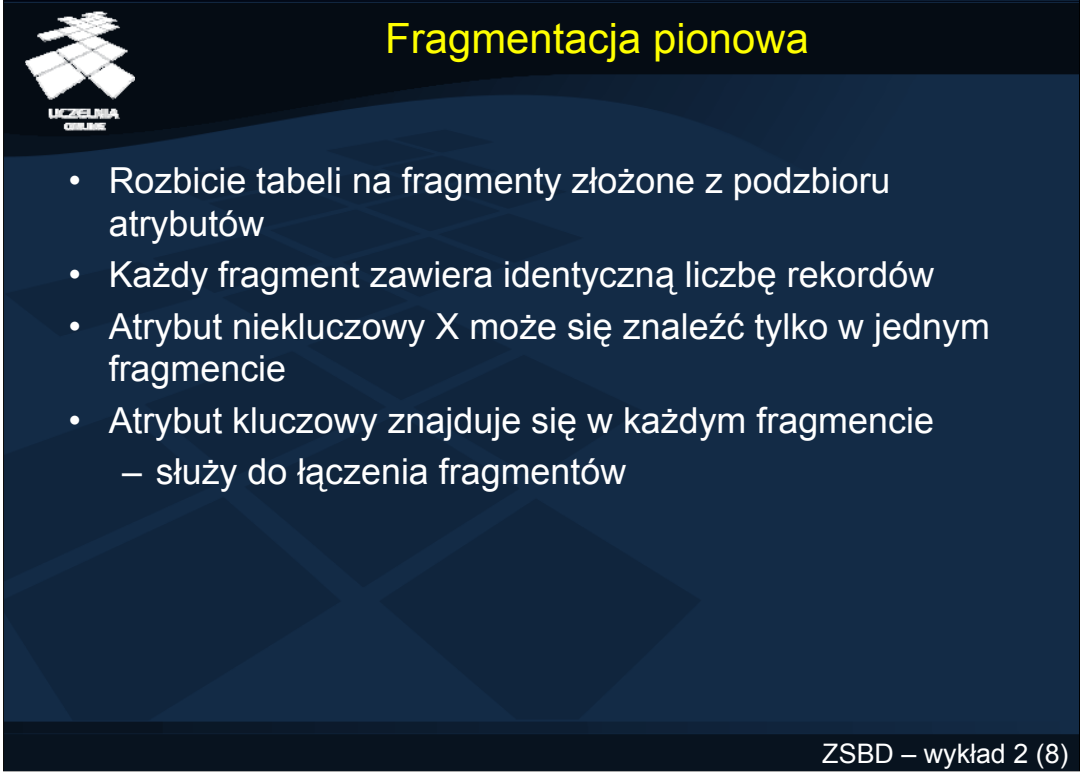


Na slajdzie w sposób schematyczny przedstawiono wspomniane wcześniej techniki fragmentacji.

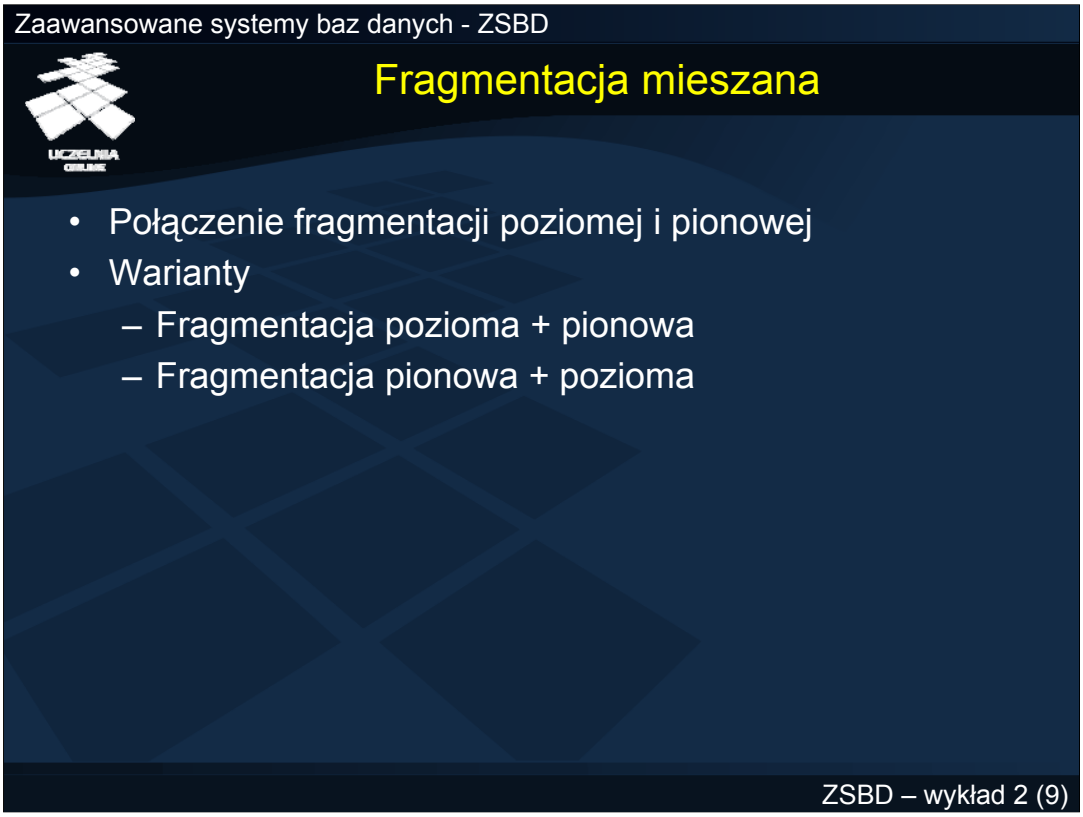


Fragmentacja pozioma (ang. horizontal fragmentation) umożliwia podział zbioru rekordów tabeli na mniejsze podzbiory, z których każdy jest opisany identyczną liczbą atrybutów.

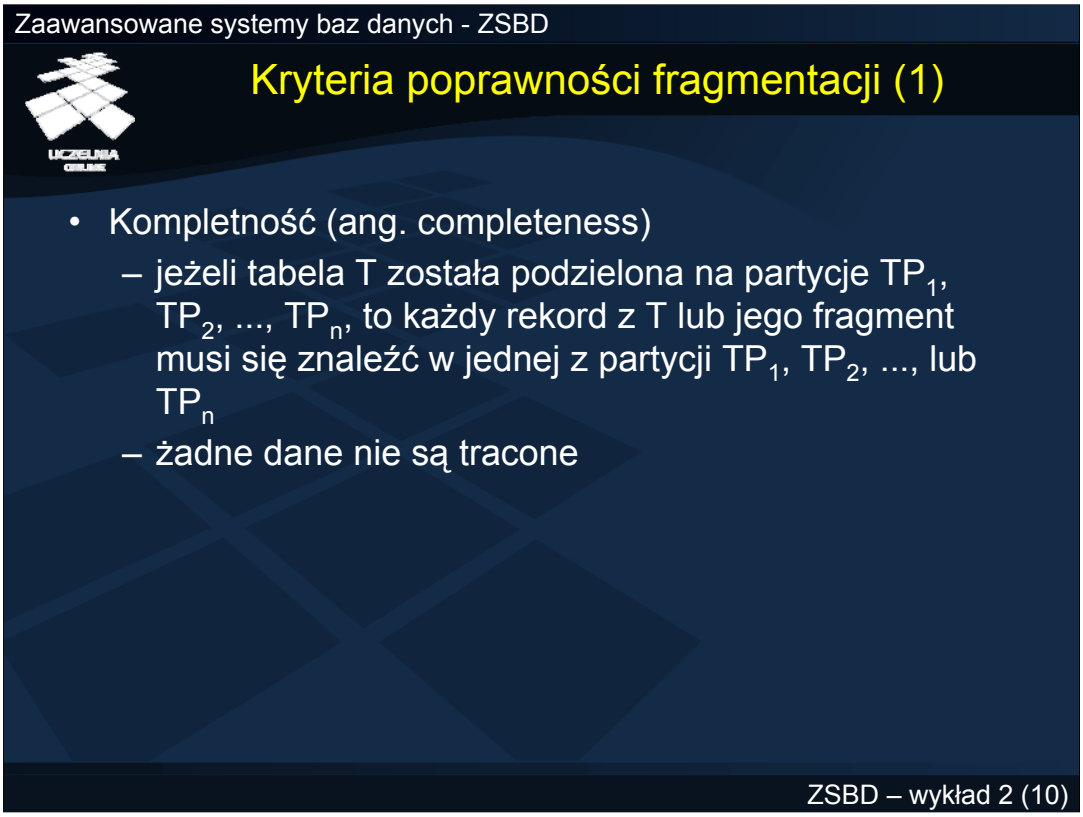
Wybór fragmentu (partycji) do którego trafi rekord jest realizowany na podstawie wartości jednego lub kilku wybranych atrybutów tabeli – tzw. atrybutów fragmentujących (partycjonujących).



Fragmentacja pionowa (ang. vertical fragmentation) umożliwia podział tabeli w pionie, tj. jej rozbicie na fragmenty złożone z podzbiorów atrybutów tabeli pierwotnej. Każdy fragment zawiera identyczną liczbę rekordów. Dany atrybut może się znaleźć tylko w jednym fragmencie. Nie dotyczy to atrybutów kluczowych, które są umieszczane w każdym fragmencie i służą one do łączenia fragmentów.



Fragmentacja mieszana (ang. hybrid fragmentation) stanowi połączenie fragmentacji poziomej i pionowej. Występuje w dwóch wariantach. W pierwszym tabela jest najpierw dzielona poziomo, a następnie wszystkie lub wybrane jej fragmenty są dalej dzielone pionowo. W wariancie drugim, tabela jest najpierw dzielona pionowo, a następnie wszystkie lub wybrane jej fragmenty są dalej dzielone poziomo.



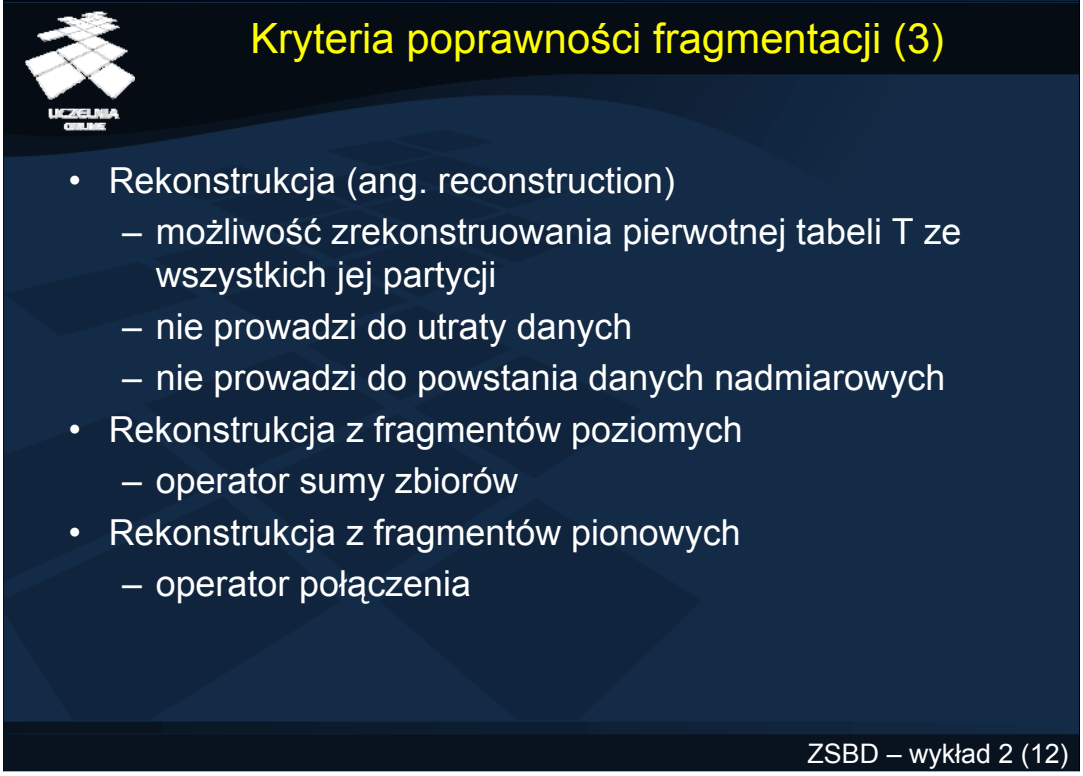
Poprawna fragmentacja musi spełniać trzy następujące kryteria: kompletność, rozłączność i rekonstrukcję.

Kompletność (ang. *completeness*) oznacza, że jeżeli tabela T została podzielona na partycje TP_1, TP_2, ..., TP_n, to każdy rekord z T lub jego fragment musi się znaleźć w jednej z partycji TP_1, TP_2, ..., lub TP_n. Kryterium to gwarantuje, że na skutek partycjonowania żadne dane z tabeli pierwotnej T nie zostaną utracone.

Kryteria poprawności fragmentacji (2)

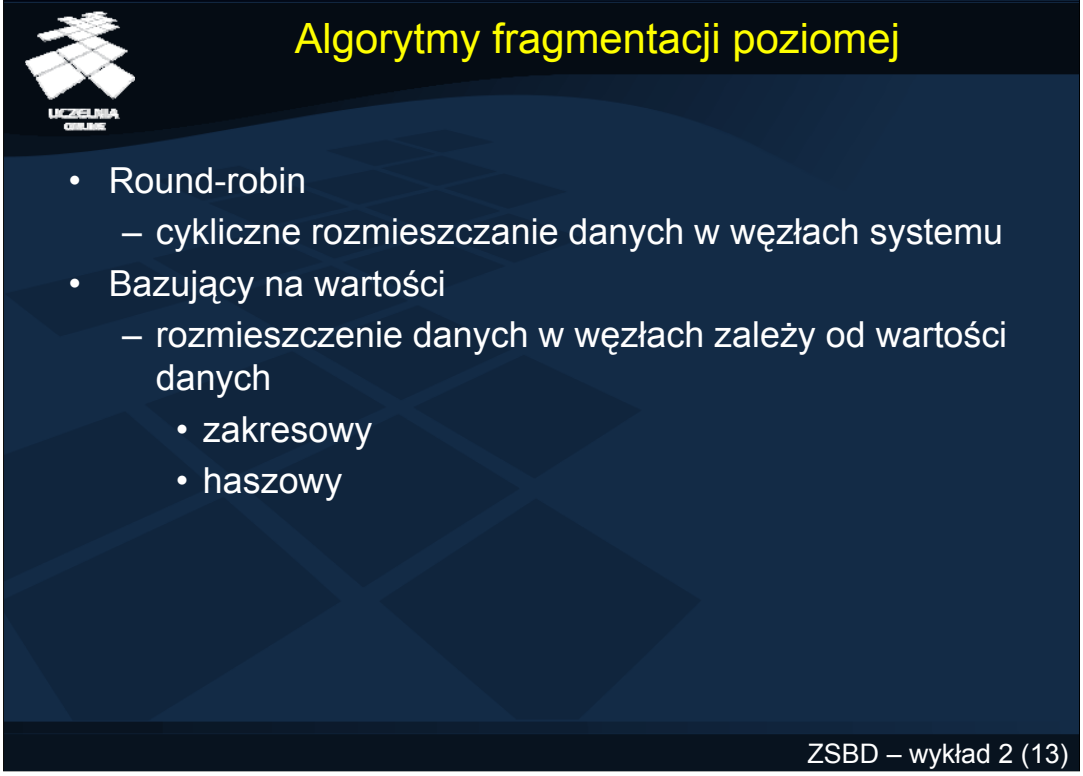
- Rozłączność (ang. disjointness)
  - jeżeli tabela T została podzielona na partycje TP_1, TP_2, ..., TP_n, to każdy rekord z T lub jego fragment nie może się znaleźć w więcej niż jednej partycji
  - nie pojawią się dane nadmiarowe
  - wyjątek: partycjonowanie pionowe
    - klucze podstawowe występują w każdej partycji

Rozłączność (ang. *disjointness*) oznacza, że jeżeli tabela *T* została podzielona na partycje TP_1, TP_2, ..., TP_n, to każdy rekord z *T* lub jego fragment nie może się znaleźć w więcej niż jednej partycji. Kryterium to gwarantuje, że na skutek partycjonowania w bazie danych nie pojawią się dane nadmiarowe. Wyjątkiem od tej reguły jest partycjonowanie pionowe, w którym wartości atrybutów stanowiących klucz podstawowy tabeli występują w każdej partycji.

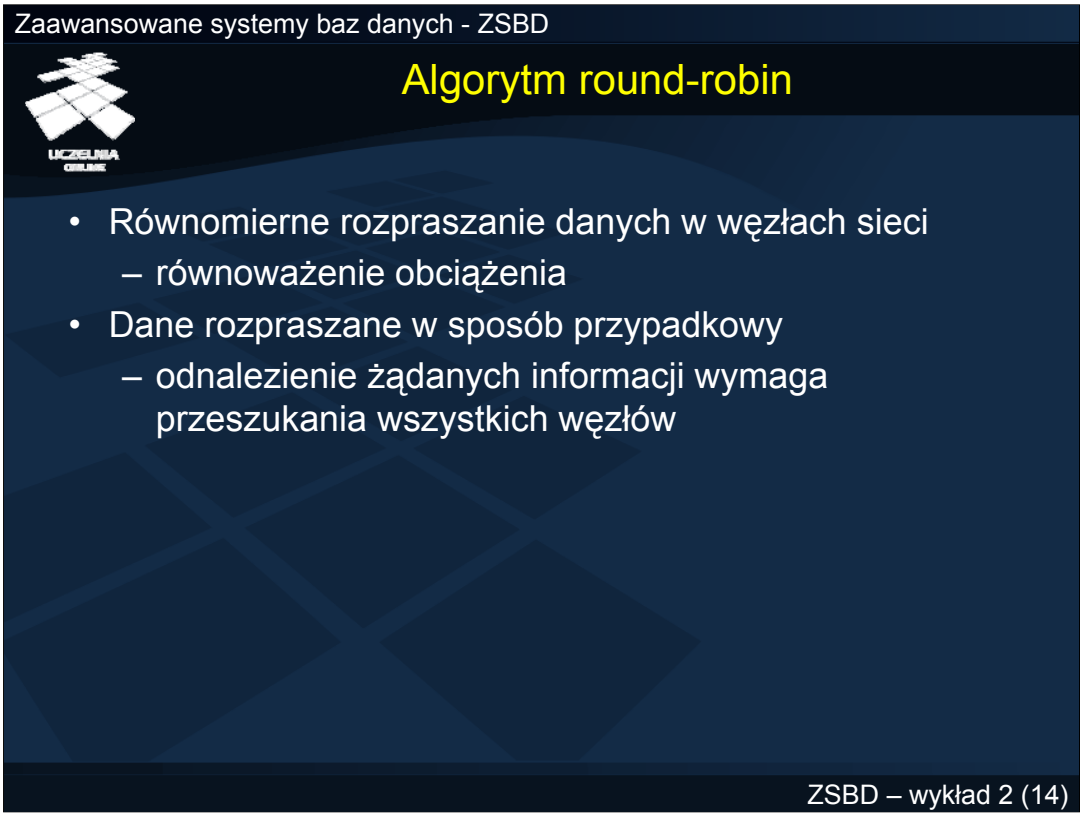


Rekonstrukcja (ang. *reconstruction*) oznacza, że musi istnieć możliwość zrekonstruowania pierwotnej tabeli *T* ze wszystkich jej partycji. Rekonstrukcja ta nie może doprowadzić ani do utraty żadnych danych, ani do powstania danych nadmiarowych.

W przypadku partycjonowania poziomego, rekonstrukcji pierwotnej tabeli dokonuje się z wykorzystaniem operatora sumy zbiorów rekordów znajdujących się we wszystkich partycjach. W przypadku partycjonowania pionowego, rekonstrukcję realizuje się za pomocą łączenia partycji wykorzystując do tego klucz podstawowy tabeli pierwotnej.



Fragmentację poziomą implementuje się z wykorzystaniem dwóch podstawowych algorytmów, tj. round-robin i bazującego na wartości. Algorytm round-robin rozmieszcza rekordy cyklicznie we wszystkich węzłach systemu. W przypadku algorytmów bazujących na wartości, tj. zakresowego i hash'owego, rozmieszczenie danych w sieci zależy od wartości samych danych.

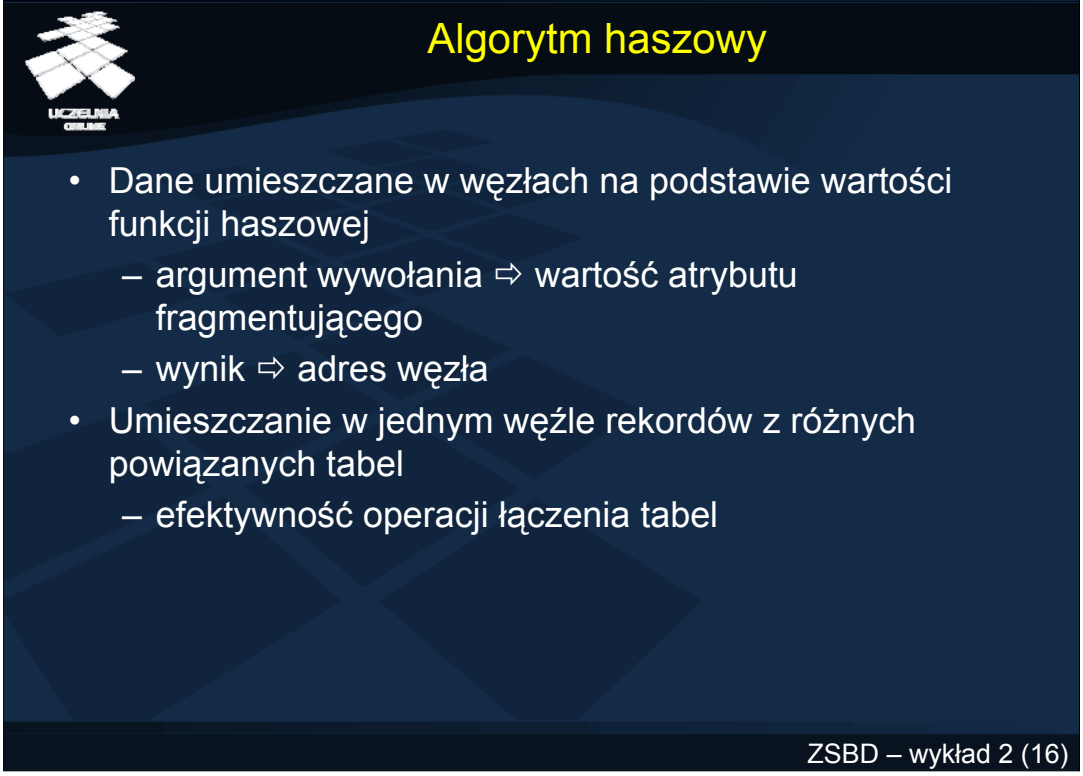


Algorytm round-robin umożliwia równomierne rozpraszanie danych w węzłach sieci. Przykładowo, jeśli w sieci znajdują się trzy węzły, to pierwszy rekord tabeli zostanie umieszczony w węźle pierwszym, drugi — w węźle drugim, trzeci rekord — w węźle trzecim, czwarty — znów w węźle pierwszym itp. Ponieważ dane są rozpraszane w sposób przypadkowy, więc odnalezienie żądanych rekordów wymaga przeszukania wszystkich węzłów, co jest wadą tego rozwiązania.

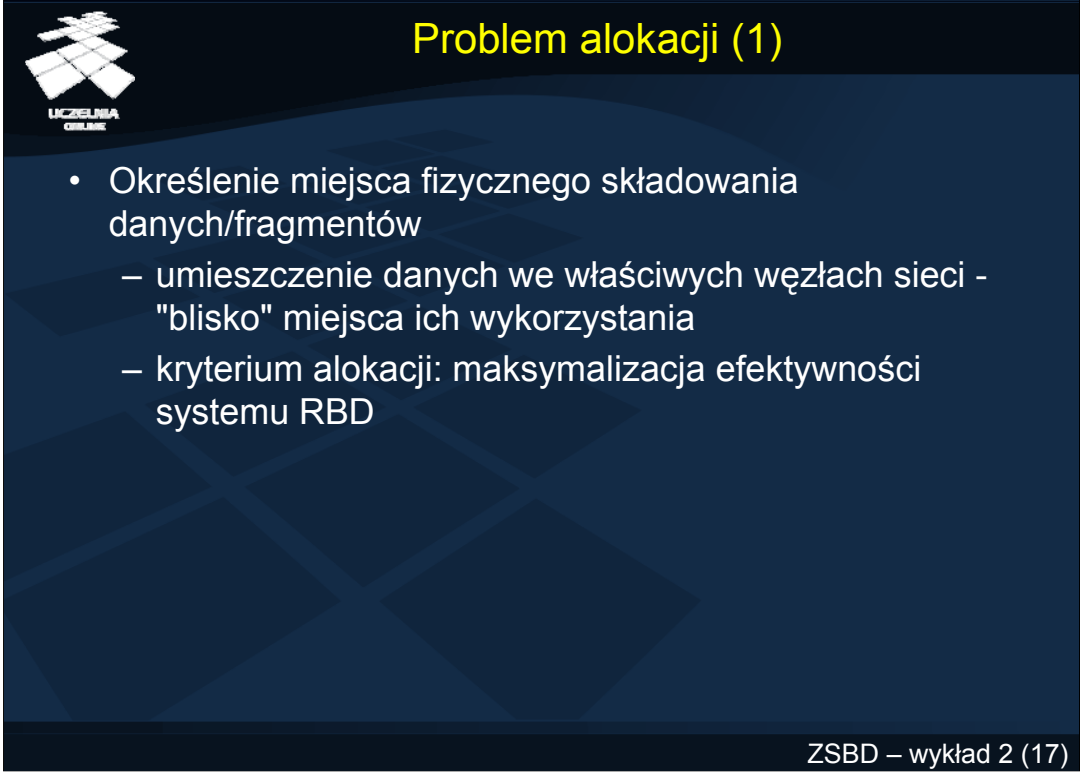
Algorym zakresowy

- Każdy węzeł składuje dane o wartościach atrybutu fragmentującego z zadanego zakresu lub o zadanej wartości

W algorytmie zakresowym, każdy węzeł przechowuje dane o wartościach atrybutu fragmentującego z zadanego zakresu lub o zadanej wartości. Przykładowo, na slajdzie przedstawiono trzy węzły BD1, BD2, BD3. Pierwszy z nich zawiera fragment tabeli faktury, przechowujący faktury z lat od 2000 do 2002. Drugi węzeł zawiera fragment tabeli przechowujący faktury z lat 2003-2005, a trzeci - zawiera fragment tabeli przechowujący faktury z roku 2006.



W przypadku algorytmu fragmentacji haszowej, dane są umieszczane w węzłach zgodnie z wartością systemowej funkcji haszowej. Argumentem wejściowym tej funkcji jest wartość atrybutu, a jej wynikiem — adres węzła, w którym zostanie umieszczony rekord. W celu odnalezienia żądanych rekordów SZBD wykorzystuję tę samą funkcję haszową, która została wykorzystana do fragmentacji danych. Zaletą tej metody jest możliwość automatycznego umieszczania w tym samym węźle rekordów pochodzących z różnych, powiązanych z sobą tabel. W ten sposób zwiększa się efektywność wykonywania operacji łączenia tabel, gdyż łączone z sobą rekordy znajdują się w tym samym węźle.



Ponieważ dane w systemie RBD mogą być składowane na dowolnym jego węźle, więc zachodzi konieczność takiego rozmieszczenia danych (np. tabel, fragmentów, indeksów), który zapewni największą efektywność całego systemu. Jest to tzw. problem alokacji.

Problem alokacji (2)

- Problem alokacji
  - dla danego zbioru fragmentów
    $F=\{F_1, F_2, ..., F_n\}$
    i zbioru węzłów systemu
    $W=\{W_1, W_2, ..., W_m\}$
    w których działają aplikacje
    $A=\{A_1, A_2, ..., A_j\}$
  - znaleźć optymalny przydział F_i (i=1..n) do W_j (j=1..m)

Problem alokacji można zdefiniować następująco: dla danego zbioru fragmentów $F=\{F_1, F_2, ..., F_n\}$ i danego zbioru węzłów systemu RBD $W=\{W_1, W_2, ..., W_n\}$ w których działa zbiór aplikacji użytkowników $A=\{A_1, A_2, ..., A_j\}$ należy znaleźć optymalny przydział fragmentów do węzłów.

Problem alokacji (3)

- Optimum jest definiowane w kontekście
  - łącznego kosztu
    - przechowywania F_i w węźle W_j
    - odczytu F_i z węzła W_j
    - zmodyfikowania F_i w W_j
    - komunikacji z W_j
  - efektywności systemu
    - mierzona jego przepustowością w każdym z węzłów

Optimum przydziału jest definiowane w kontekście łącznego kosztu i efektywności systemu. Łączny koszt uwzględnia:

- koszt przechowywania fragmentu F_i w węźle W_j,

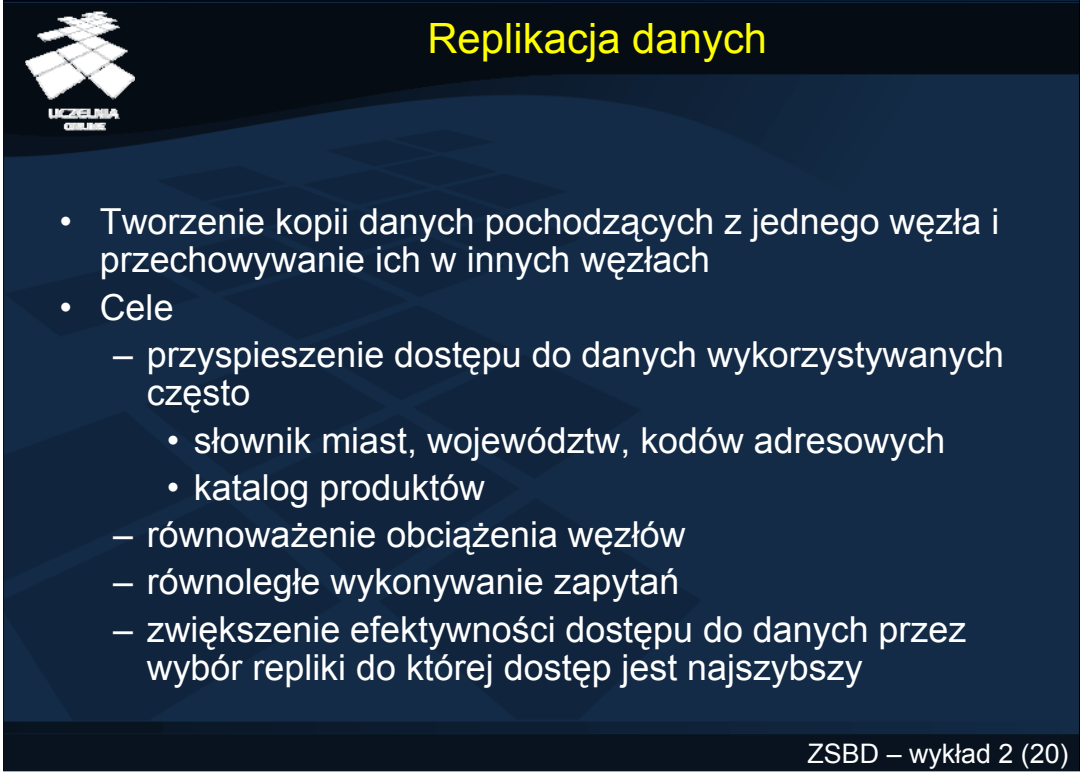
- koszt odczytu fragmentu F_i z węzła W_j,

- koszt zmodyfikowania fragmentu F_i w węźle W_j,

- koszt komunikacji z węzłem W_j.

Efektywność systemu jest natomiast mierzona przepustowością w każdym z jego węzłów.

W literaturze zaproponowano wiele algorytmów alokacji. Ich omówienie wykracza jednak poza zakres tego wykładu.



Drugim podstawowym zagadnieniem projektowania RBD jest określenie sposobu duplikowania danych w wielu węzłach systemu RBD. Duplikowanie takie będziemy dalej nazywali replikacją danych.

Replikacja polega na tworzeniu kopii danych pochodzących z jednego węzła i przechowywaniu tych kopii w innych węzłach.

Cele replikacji są następujące:

- przyspieszenie dostępu do danych poprzez ich umieszczenie w węźle, który z nich intensywnie korzysta; przykładami mogą być tabele słownikowe miast, województw i kodów adresowych w systemach ewidencji, katalogi produktów w systemach sprzedaży;

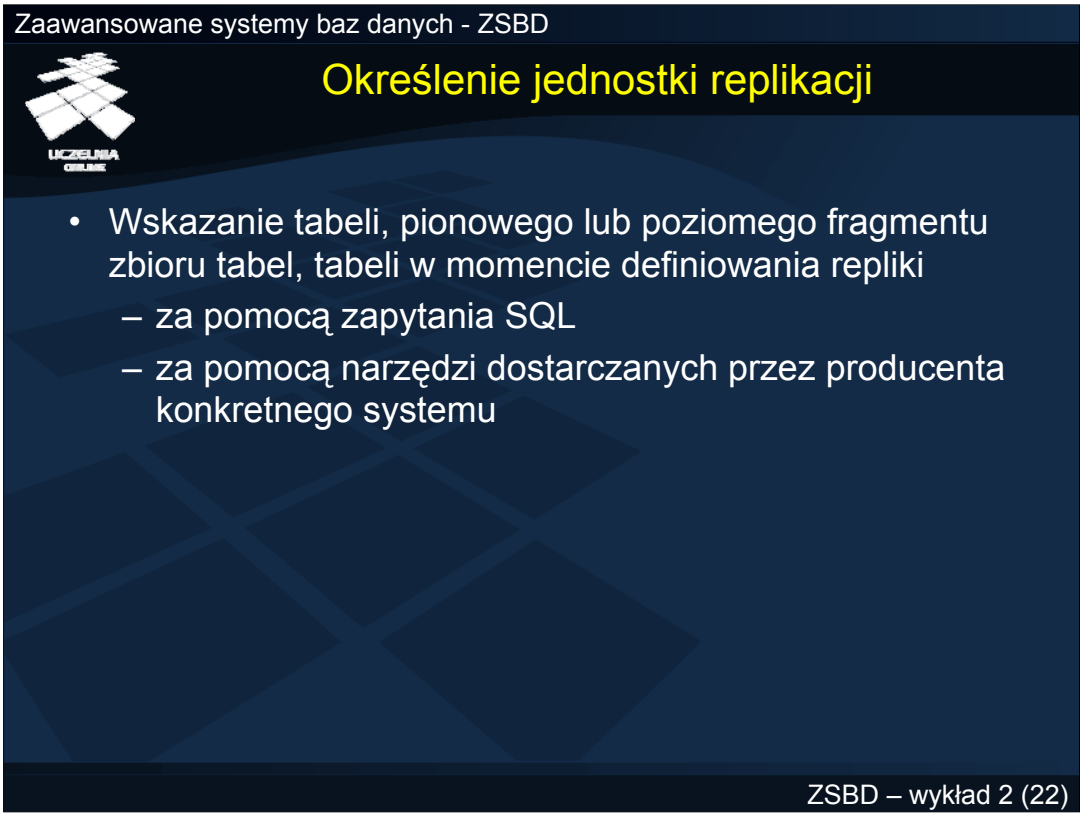
- równoważenie obciążenia węzłów poprzez powielanie intensywnie wykorzystywanych (odczytywanych) danych do innych węzłów;

- równoległe wykonywanie zapytań adresujących te same dane; w takim przypadku 2 zapytania odwołujące się do tej samej tabeli mogą zostać wykonane na dwóch różnych replikach tej tabeli;

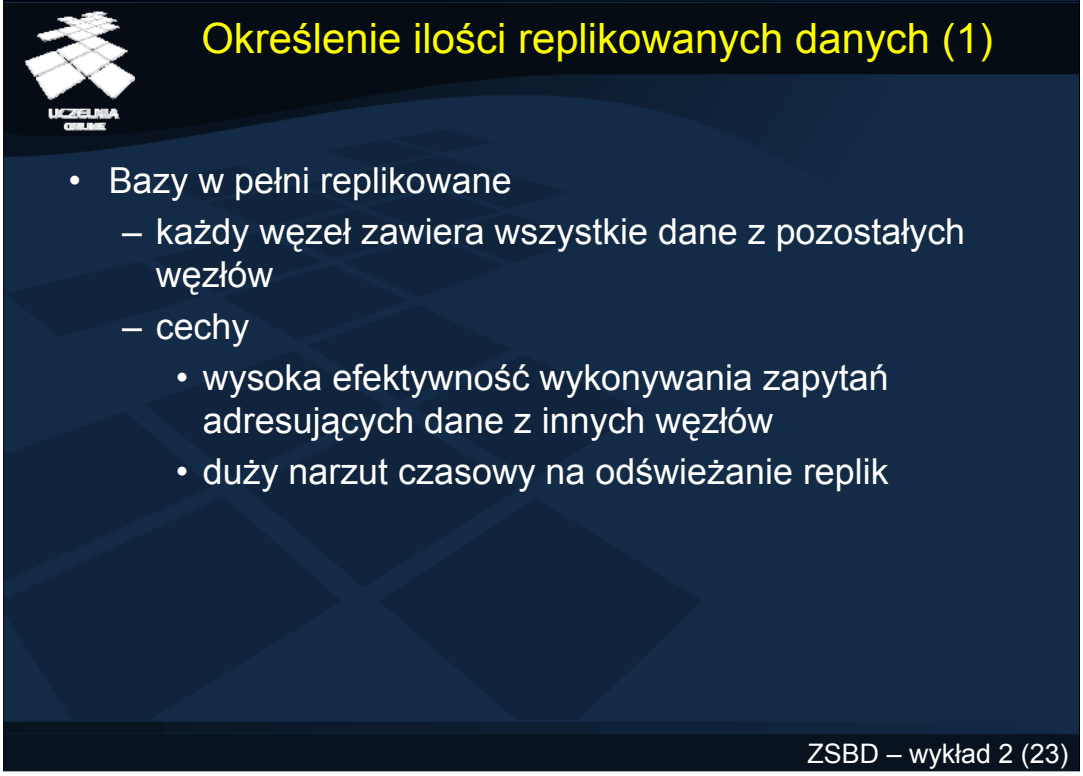
- zwiększenie efektywności dostępu do danych przez wybór repliki, do której dostęp jest najszybszy.

Problematyka replikacji

- Określenie jednostki replikacji
- Określenie ilości replikowanch danych
- Określenie momentu odświeżania
- Określenie sposobu odświeżania

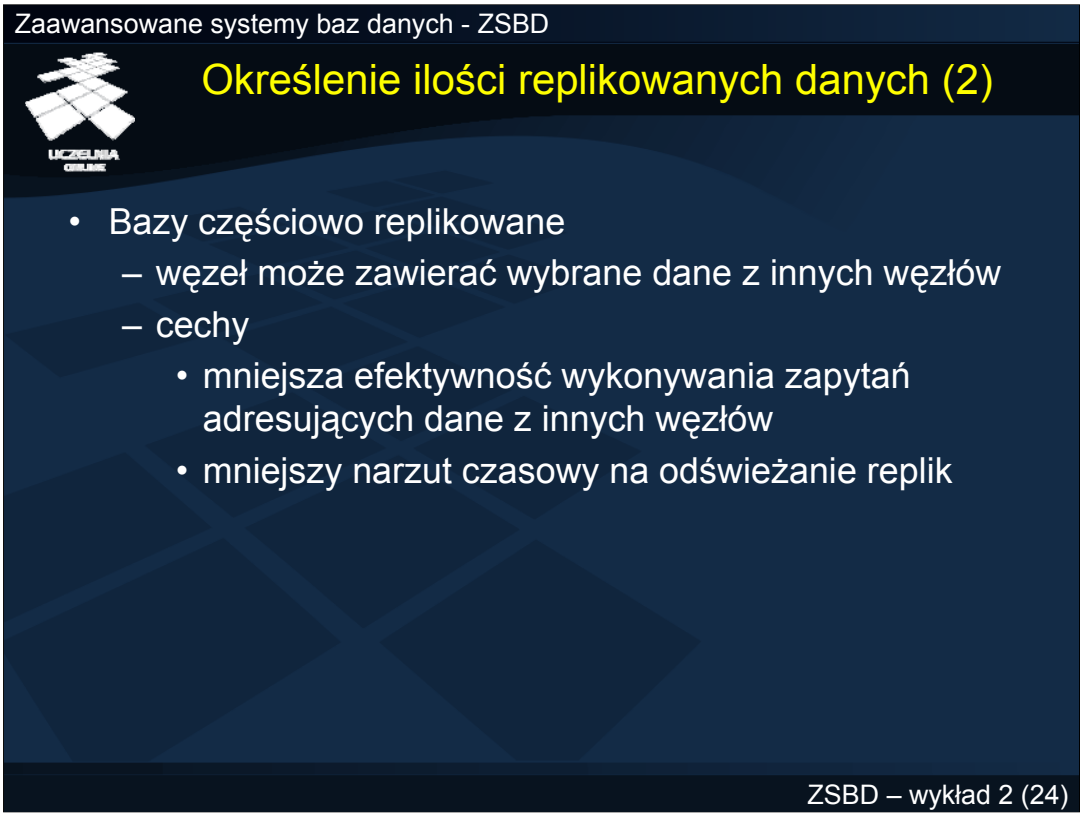


Określenie jednostki replikacji polega na wskazaniu jak duża jest zawartość pojedynczej repliki. Replika może być zbudowana na pojedynczej tabeli, pionowym lub poziomym jej fragmencie, bądź na zbiorze tabel. Jednostkę replikacji określa się w momencie definiowania repliki albo za pomocą zapytania SQL albo za pomocą narzędzi programistycznych dostarczanych przez producenta konkretnego systemu.

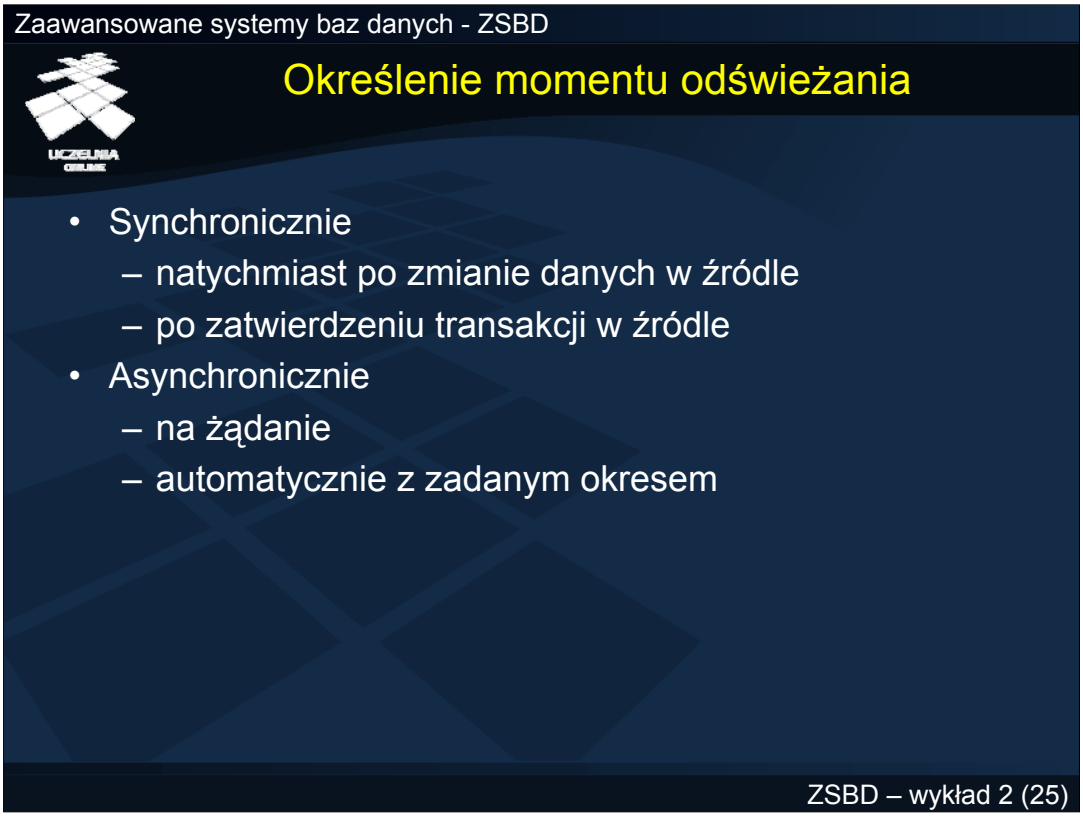


W zakresie określenia ilości replikowanych danych wyróżnia się bazy danych w pełni replikowane (ang. fully replicated databases) i bazy danych częściowo replikowane (ang. partially replicated databases).

W bazach w pełni replikowanych każdy węzeł zawiera wszystkie dane z pozostałych węzłów. Bazy takie cechują się wysoką efektywnością wykonywania zapytań, ponieważ każde zapytanie może zostać wykonane lokalnie. Wadą tego typu baz danych jest duży narzut czasowy na odświeżanie replik, ponieważ jest ich zbyt dużo. W praktyce tego typu bazy danych stosuje się niezwykle rzadko.



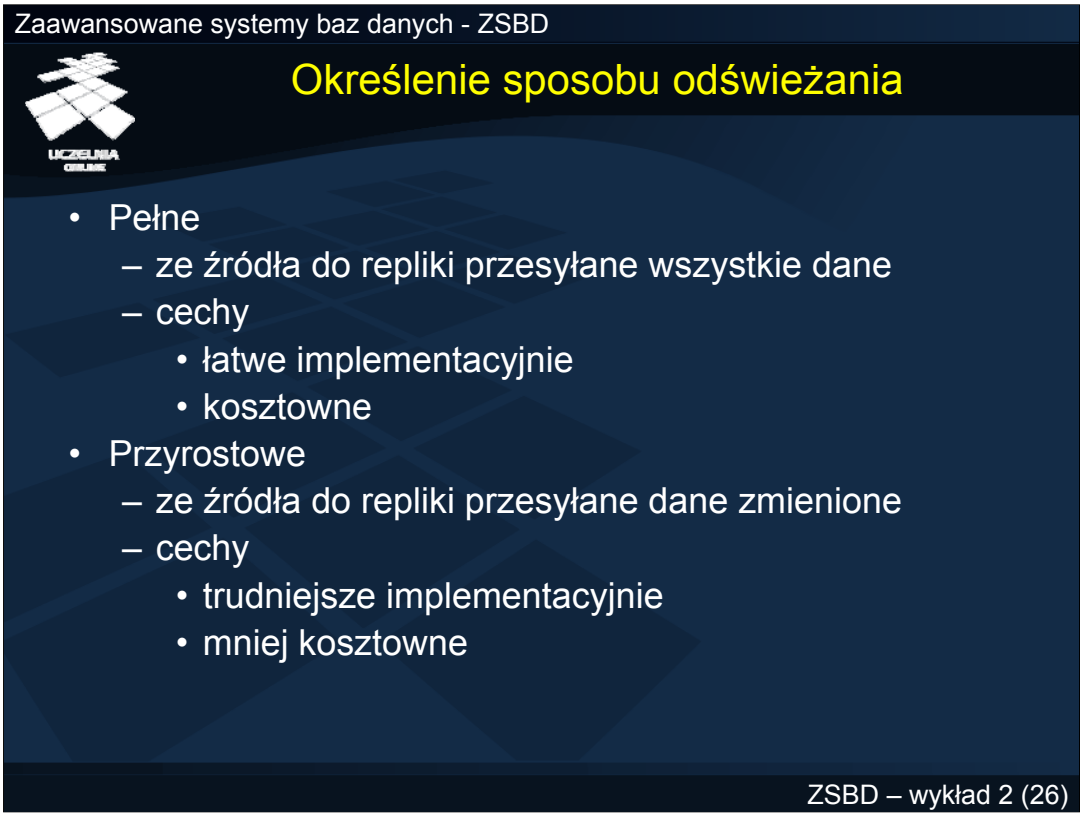
W bazach danych częściowo replikowanych każdy węzeł może zawierać wybrane dane z innych węzłów. Tego typu bazy danych cechuje mniejsza efektywność od poprzednich ponieważ część zapytań będzie wymagała sięgnięcia do zdalnych węzłów. Mniejszy jest też narzut czasowy na odświeżanie replik, ponieważ replik tych jest mniej niż w bazie danych w pełni replikowanej.



Ważną cechą odświeżania repliki jest wybór momentu odświeżania. W praktyce stosuje się dwie techniki, tj. odświeżanie synchroniczne i odświeżanie asynchroniczne.

Pierwsza technika polega na propagowaniu danych ze źródła do repliki albo natychmiast po zmianie zawartości źródła albo po zatwierdzeniu transakcji w źródle. Drugie rozwiązanie jest stosowane w praktyce.

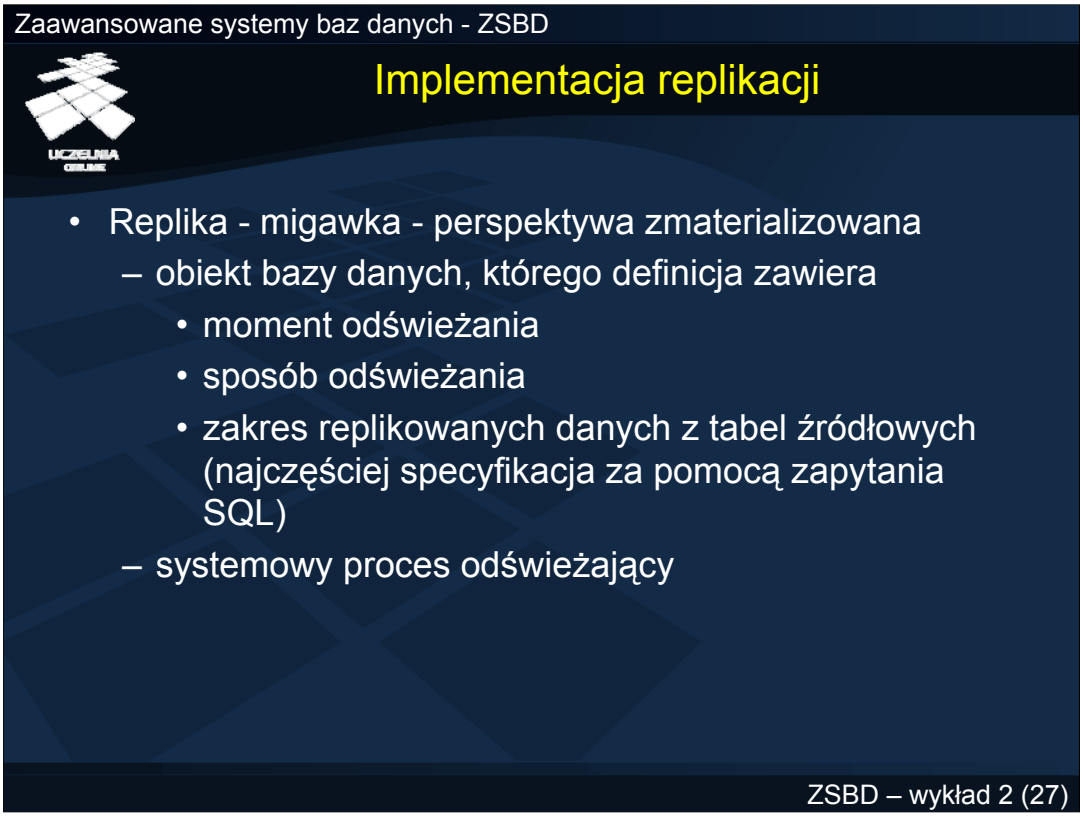
Odświeżanie asynchroniczne polega na propagowaniu danych do repliki po jakimś czasie od wprowadzenia zmian w źródle. Możliwe są tu dwa rozwiązania, albo odświeżanie na żądanie repliki, albo odświeżanie automatyczne z zadanym okresem. W praktyce stosuje się oba rozwiązania.



Kolejną ważną cechą odświeżania repliki jest określenie sposobu jej odświeżania. W praktyce stosuje się dwie techniki, tj. odświeżanie pełne i odświeżanie przyrostowe.

Odświeżanie pełne polega na każdorazowym przesyłaniu ze źródła do repliki wszystkich danych, które replika udostępnia. Ta technika odświeżania charakteryzuje się łatwością implementacji i dużym kosztem odświeżania ze względu na dużą ilość przesyłanych danych.

Odświeżanie przyrostowe polega na przesyłaniu do repliki wyłącznie zmian dokonanych w źródle od czasu ostatniego odświeżenia. Ta technika cechuje się trudniejszą implementacją i niższymi kosztami odświeżania.



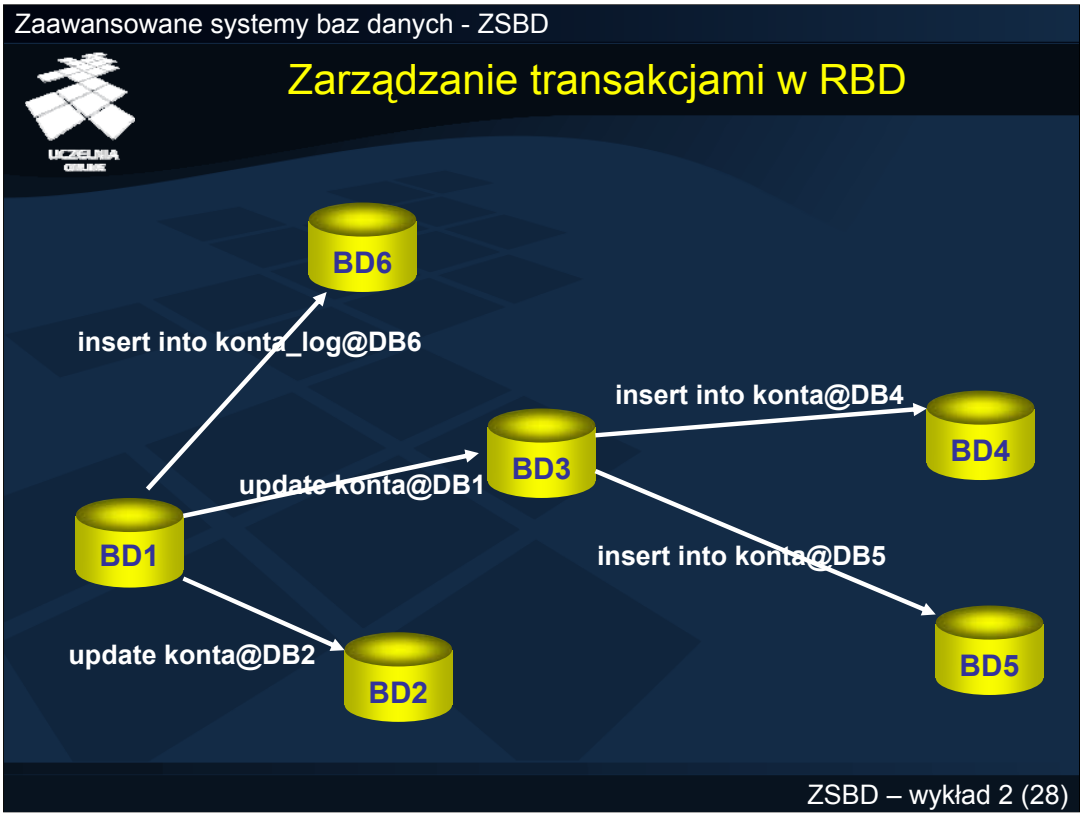
W praktyce replika jest implementowana jako pewien obiekt bazy danych, którego definicja zawiera:

- moment odświeżania,

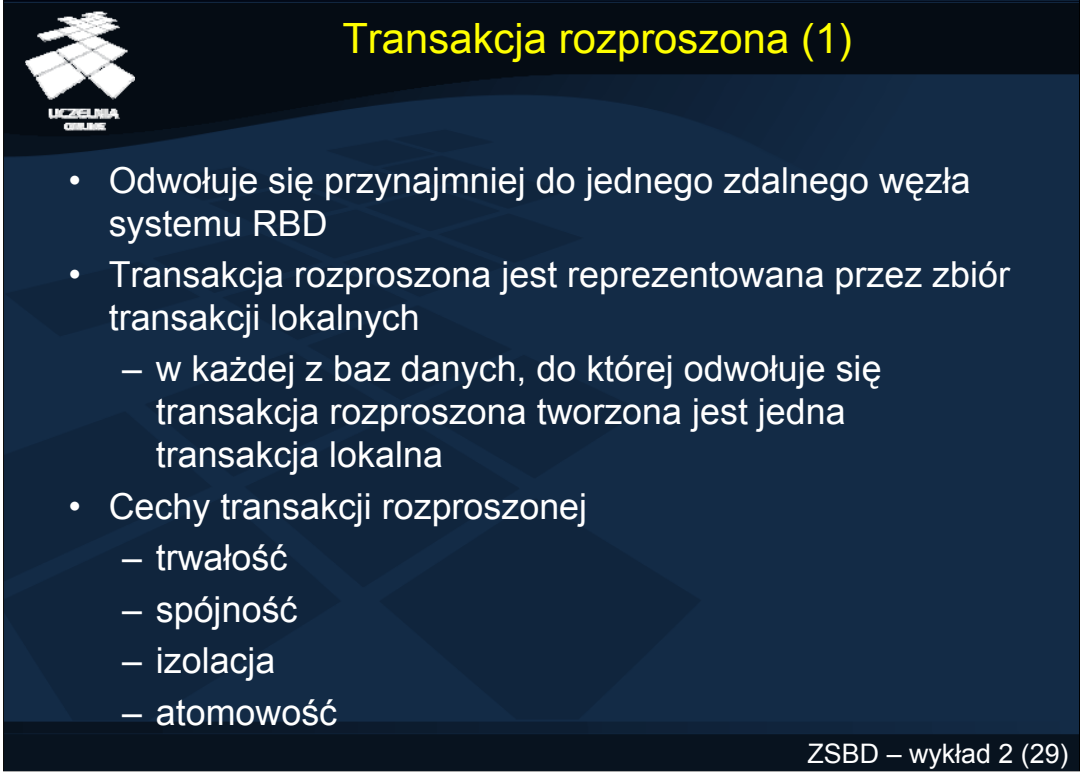
- sposób odświeżania,

- zakres replikowanych danych z tabel źródłowych; zakres ten najczęściej specyfikuje się za pomocą zapytania SQL.

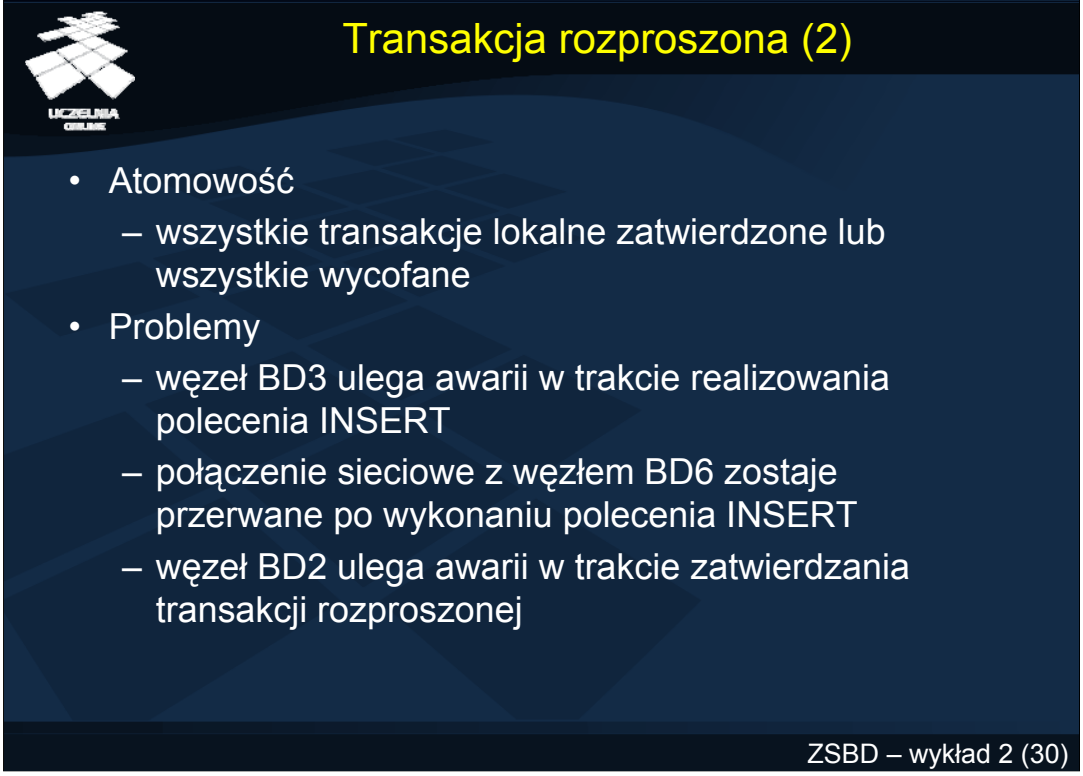
Z repliką jest związany systemowy proces odpowiedzialny za jej odświeżanie.



W systemie RBD użytkownik może wykonać transakcję, która adresuje wiele węzłów. Przykład takiej transakcji przedstawiono na slajdzie. W skład systemu wchodzi 6 baz danych BD1-BD6. Z bazy BD1 użytkownik wykonuje transakcję która wstawia dane do tabeli *konta_log* w bazie BD6, modyfikuje zawartość tabeli *konta* w bazie BD2 i BD1. Ta ostatnia modyfikacja uruchamia polecenia *insert* do tabel w bazach DB4 i BD5. Transakcję taką będziemy nazywali rozproszoną (ang. distributed transaction) lub globalną (ang. global transaction).



Transakcja rozproszona odwołuje się przynajmniej do jednego zdalnego węzła systemu RBD. Transakcja rozproszona jest reprezentowana przez zbiór **transakcji lokalnych**. W każdej z baz danych, do której odwołuje się transakcja rozproszona tworzona jest jedna transakcja lokalna. Zarówno każda z transakcji lokalnych jaki i rozproszona mają cechy trwałości, spójności, izolacji i atomowości.



Cecha atomowości w odniesieniu do transakcji rozproszonej oznacza, że wszystkie transakcje lokalne wchodzące w skład transakcji rozproszonej muszą zostać zatwierdzone. Jeśli choć jedna z transakcji lokalnych nie może zostać zatwierdzona, wówczas całą transakcję rozproszoną należy wycofać.

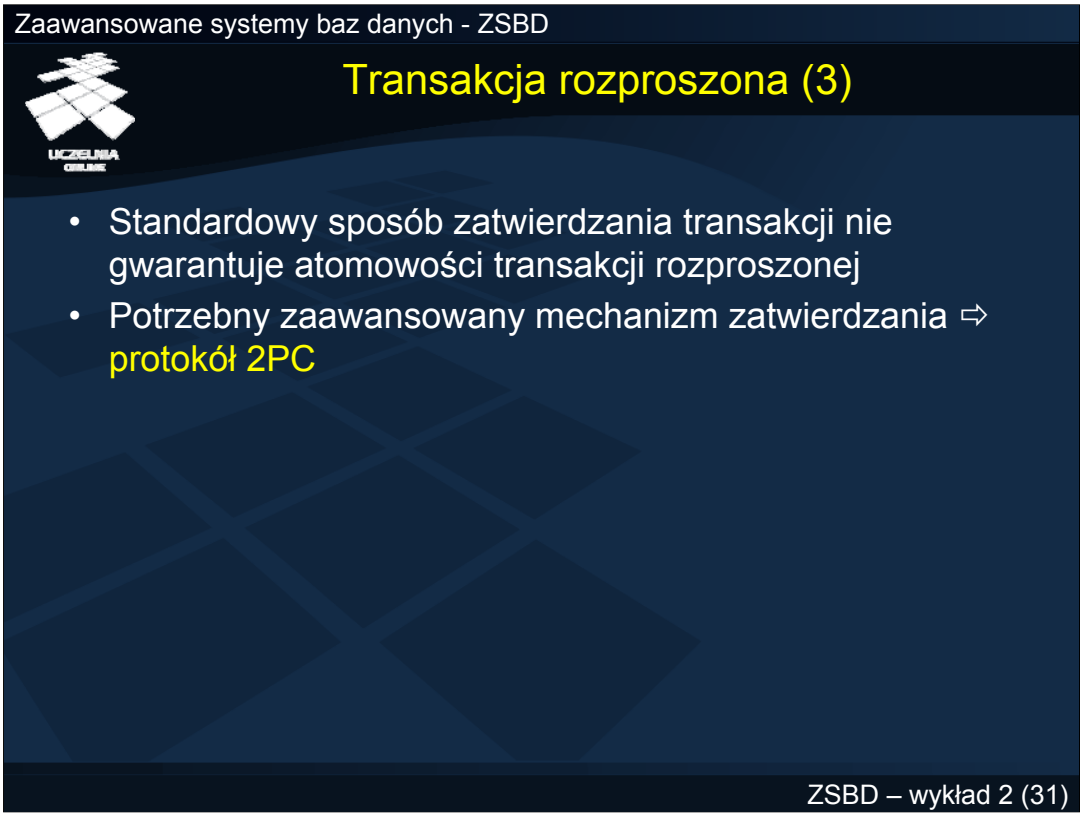
W czasie realizowania transakcji rozproszonej system RBD może ulec częściowej awarii. Przykładowo:

- węzeł BD3 (ze slajdu 28) może ulec awarii w trakcie realizowania polecenia INSERT,

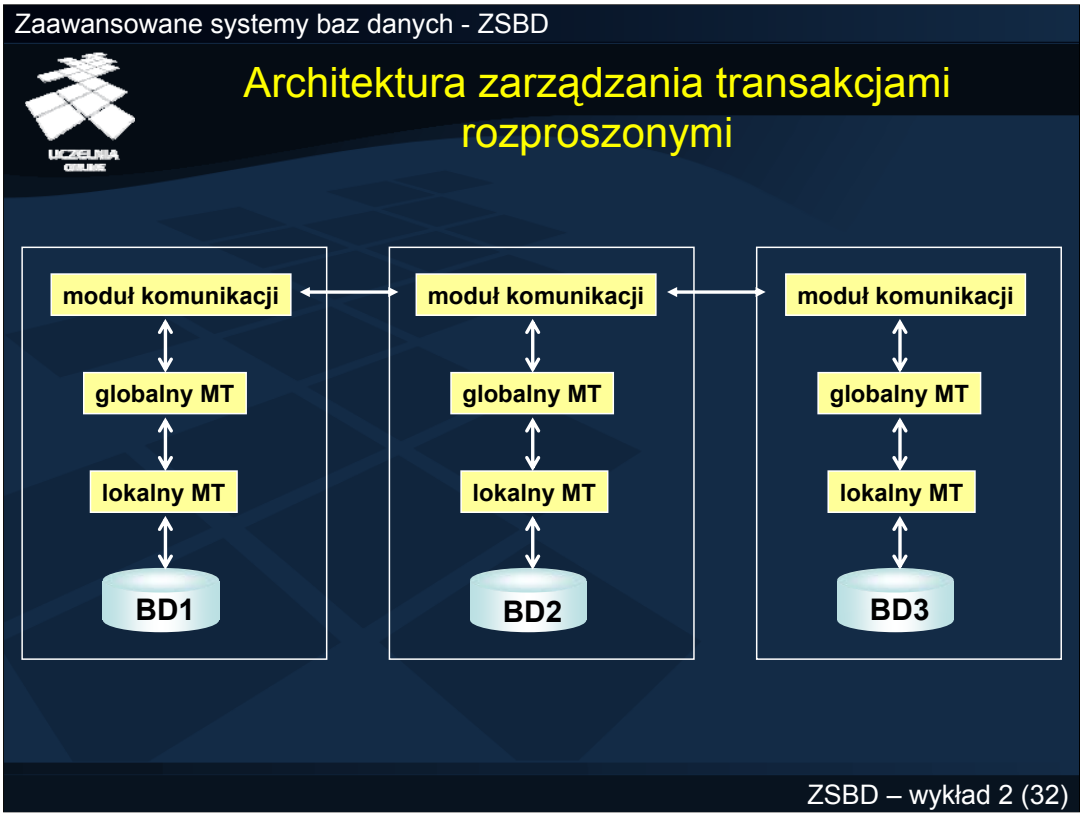
- połączenie sieciowe z węzłem BD6 może zostać przerwane po wykonaniu polecenia INSERT,

- węzeł BD2 może ulec awarii w trakcie zatwierdzania transakcji rozproszonej.

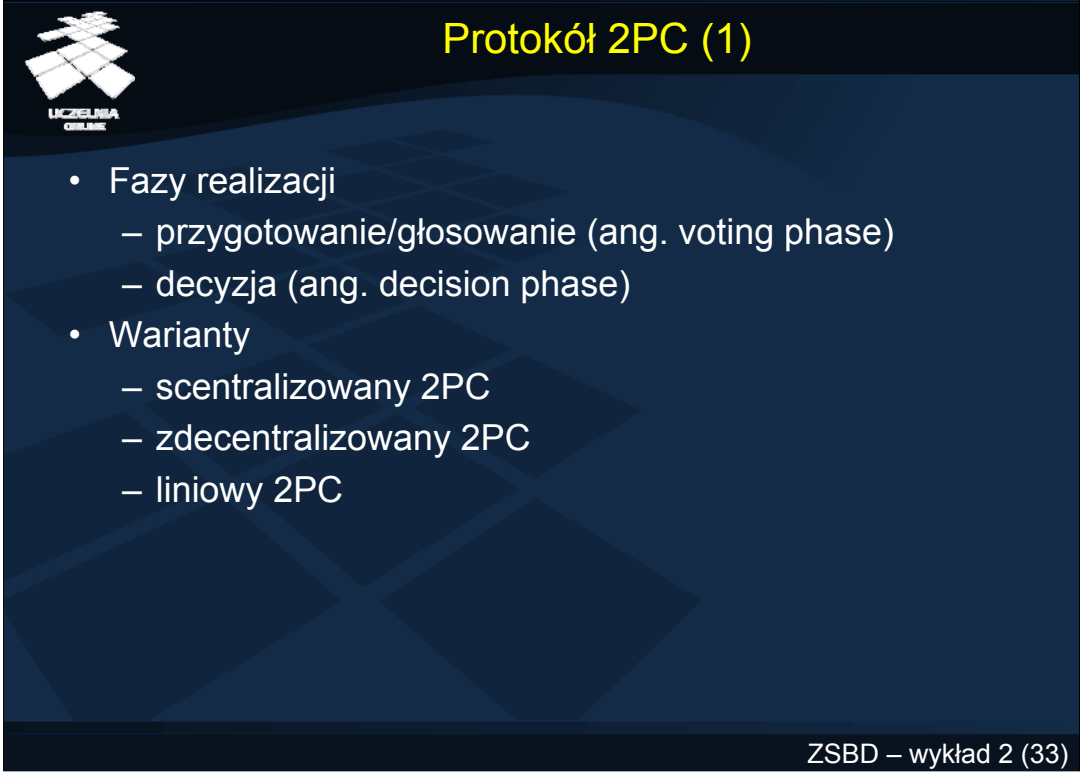
W tych i wielu innych przypadkach system musi zagwarantować zakończenie rozproszonej transakcji, w taki sposób aby zagwarantować cztery jej omówione wcześniej cechy.



Głównym problemem jest zagwarantowanie atomowości transakcji rozproszonej a standardowy mechanizm zatwierdzania transakcji nie gwarantuje jej atomowości. W związku z tym, zatwierdzanie lub wycofywanie transakcji rozproszonej, gwarantujące atomowość jest realizowane za pomocą specjalizowanego mechanizmu, tzw. **protokołu zatwierdzania dwu-fazowego** — *2PC* (ang. *two-phase commit*).



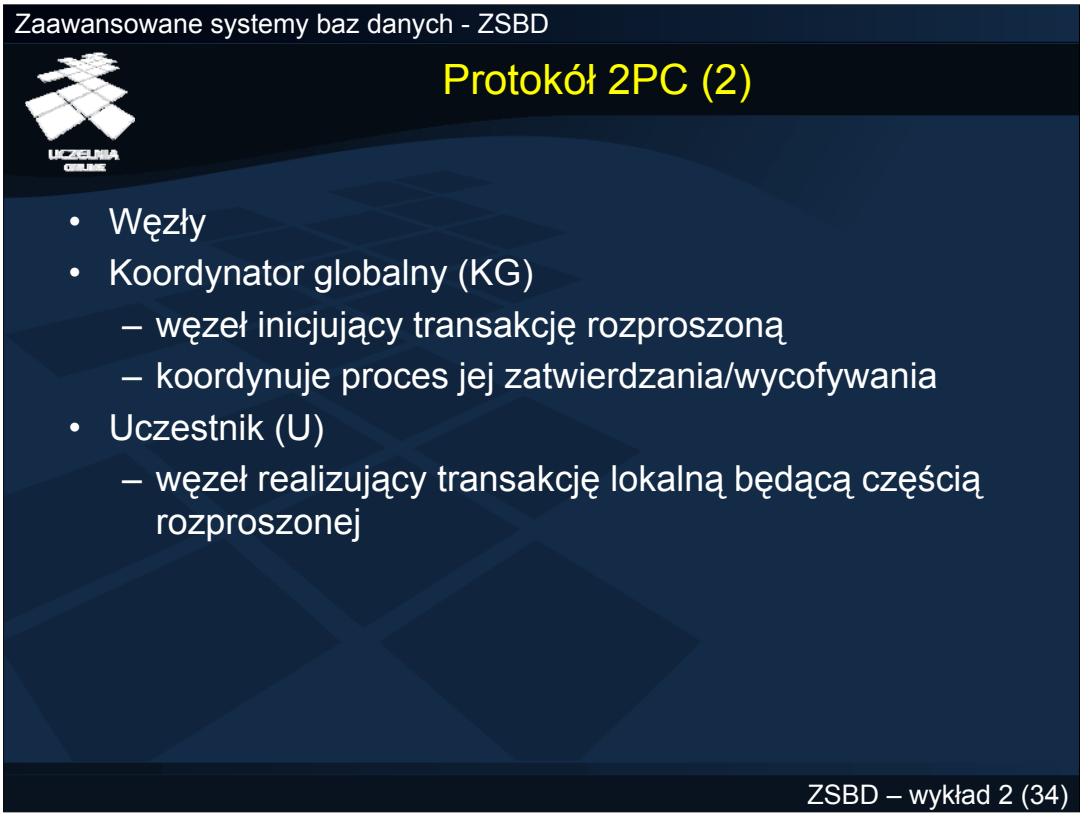
Podstawową architekturę zarządzania transakcjami rozproszonymi przedstawiono na slajdzie. Każda z trzech baz danych BD1, BD2, BD3 posiada swój własny moduł *lokalnego menadżera transakcji* (lokalny MT), identycznie jak w standardowej scentralizowanej bazie danych. Ponadto, do zarządzania transakcjami rozproszonymi jest niezbędny moduł *globalnego menadżera transakcji* (globalny MT). Jego zadaniem jest koordynowanie wykonania zarówno lokalnych jak i rozproszonych transakcji zainicjowanych w jego węźle. Poszczególne węzły realizujące transakcję rozproszoną komunikują się za pośrednictwem *modułu komunikacji*, istniejącego w każdym węźle.



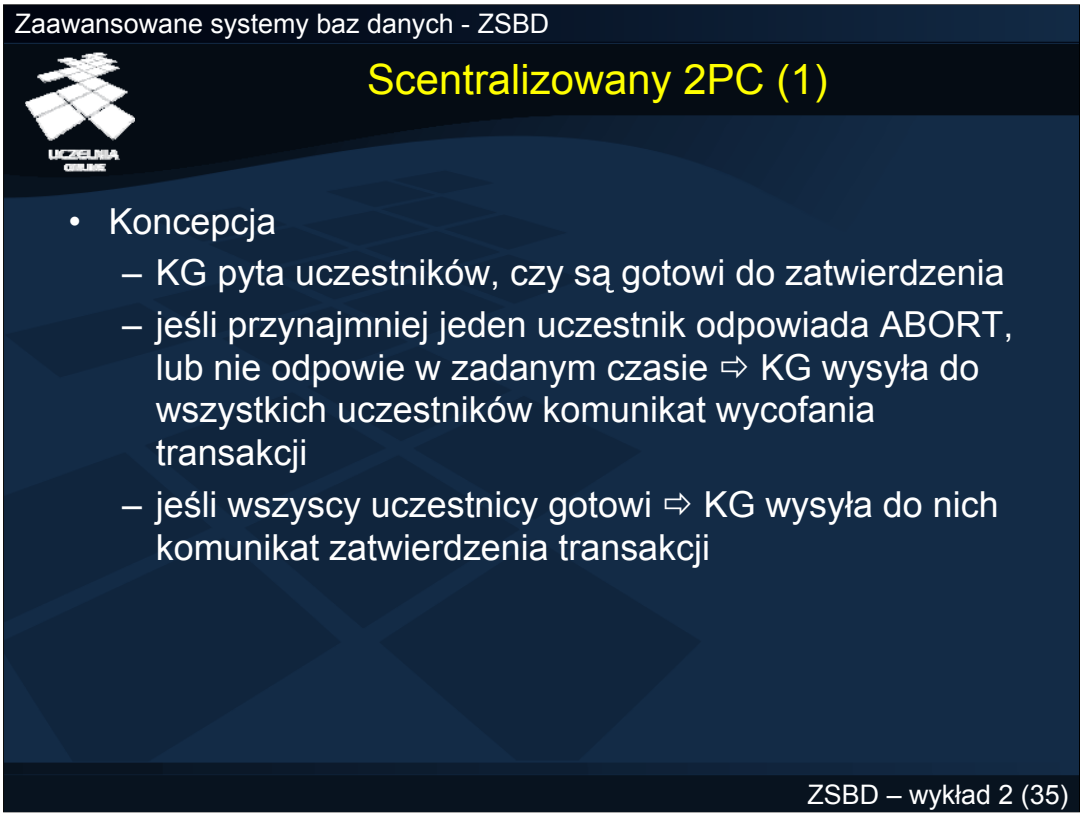
Protokół zatwierdzania 2-fazowego składa się z 2 faz: przygotowania, zwanej również głosowaniem (ang voting phase) i decyzji (ang. decision phase).

Protokół ten może być implementowany w jednym z trzech wariantów:

- scentralizowanego 2PC,
- zdecentralizowanego 2PC,
- liniowego 2PC.



Z punktu widzenia protokołu 2PC w transakcji rozproszonej biorą udział dwa rodzaje węzłów, tj. koordynator globalny i uczestnik. Koordynator globalny jest węzłem inicjującym transakcję rozproszoną i koordynującym jej zatwierdzanie lub wycofywanie. Uczestnik jest węzłem realizującym transakcję lokalną, która to transakcja jest częścią transakcji rozproszonej.

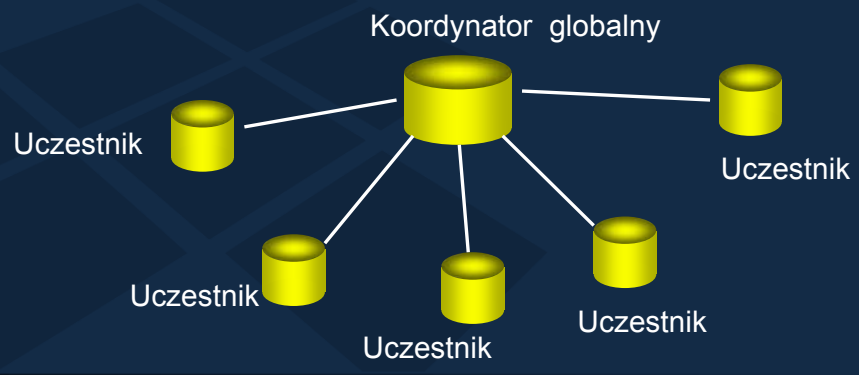


Koncepcja pracy protokołu 2PC jest następująca.

Koordynator globalny, pyta uczestników czy są gotowi do zatwierdzenia swoich transakcji lokalnych. Jeśli przynajmniej jeden uczestnik odpowiada komunikatem ABORT (nie jest gotowy), lub nie odpowie w zadanym czasie, wówczas koordynator globalny wysyła do wszystkich uczestników komunikat wycofania transakcji. Jeśli natomiast wszyscy uczestnicy zgłoszą gotowość do zatwierdzenia swoich transakcji lokalnych, wówczas koordynator globalny wysyła do nich komunikat żądający zatwierdzenia transakcji.

Scentralizowany 2PC (1)

- Topologia komunikacyjna



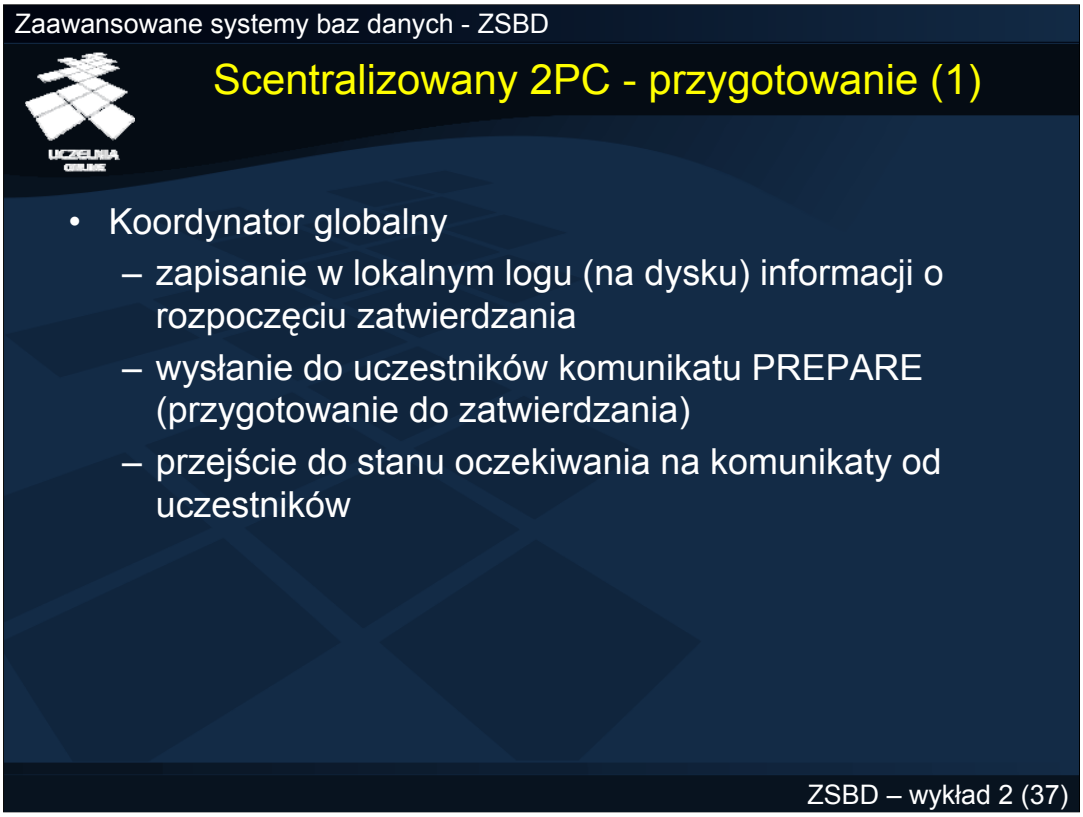
Jak wspomniano, protokół 2PC może być implementowany w jednym z trzech wariantów:

- scentralizowanego 2PC,

- zdecentralizowanego 2PC,

- liniowego 2PC.

Topologia komunikacyjna scentralizowanego protokołu 2PC została przedstawiona na slajdzie. Jak widać, wszystkie węzły komunikują się z centralnym węzłem, którym jest koordynator globalny.



Na kolejnych slajdach zostaną szczegółowo omówione poszczególne fazy protokołu 2PC w topologii scentralizowanej. W praktyce ta topologia jest wykorzystywana najczęściej.

W fazie przygotowania, koordynator globalny wykonuje następujące czynności:

- zapisuje na dysku w swoim lokalnym pliku logu rekord mówiący o rozpoczęciu zatwierdzania transakcji;

- następnie wysyła do uczestników żądanie przygotowania do zatwierdzania; jest to komunikat PREPARE;

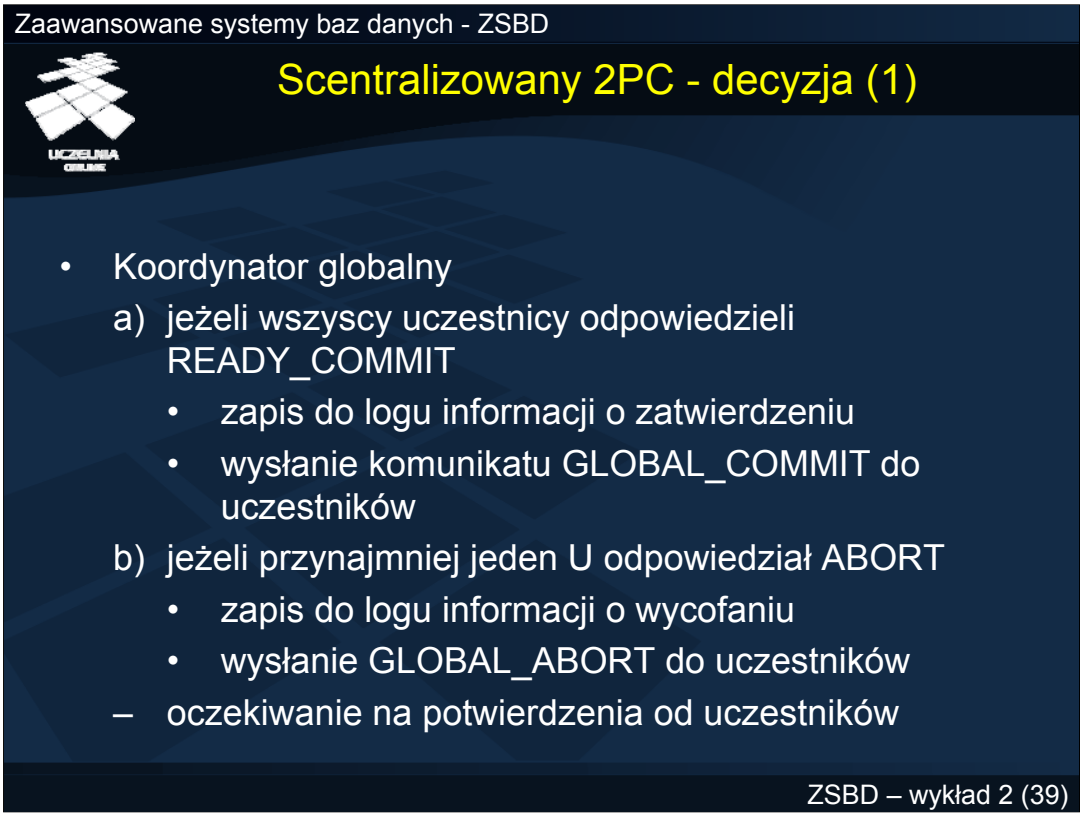
- po wysłaniu tego komunikatu KG przechodzi w stan oczekiwania na komunikaty od uczestników.

Scentralizowany 2PC - przygotowanie (2)

- Uczestnik
  - odebranie komunikatu PREPARE od KG
  - a) przygotowanie do zatwierdzenia
    - zapisanie do logu stanu transakcji i rekordu o gotowości do zatwierdzania
    - wysłanie do KG komunikatu READY_COMMIT
  - b) przygotowanie do wycofania
    - zapisanie do logu stanu transakcji i rekordu o wycofywaniu
    - wysłanie go KG komunikatu ABORT
  - przejście do stanu oczekiwania na kolejny komunikat od KG

W fazie przygotowania, uczestnik odbiera komunikat PREPARE od koordynatora globalnego i przygotowuje się do zatwierdzenia swojej transakcji lokalnej. Przygotowanie to polega na zapisaniu w swoim lokalnym pliku logu stanu transakcji i rekordu mówiącego o gotowości do zatwierdzania. Następnie uczestnik wysyła do KG komunikat READY_COMMIT, potwierdzający przygotowanie.

W przypadkach, gdy uczestnik nie może się przygotować do zatwierdzenia lub odbierze od KG komunikat ABORT (przygotowanie do wycofania), wówczas zapisuje w swoim logu stan transakcji i rekord mówiący o wycofywaniu transakcji. Następnie uczestnik wysyła komunikat ABORT do KG i przechodzi do stanu oczekiwania na kolejny komunikat.

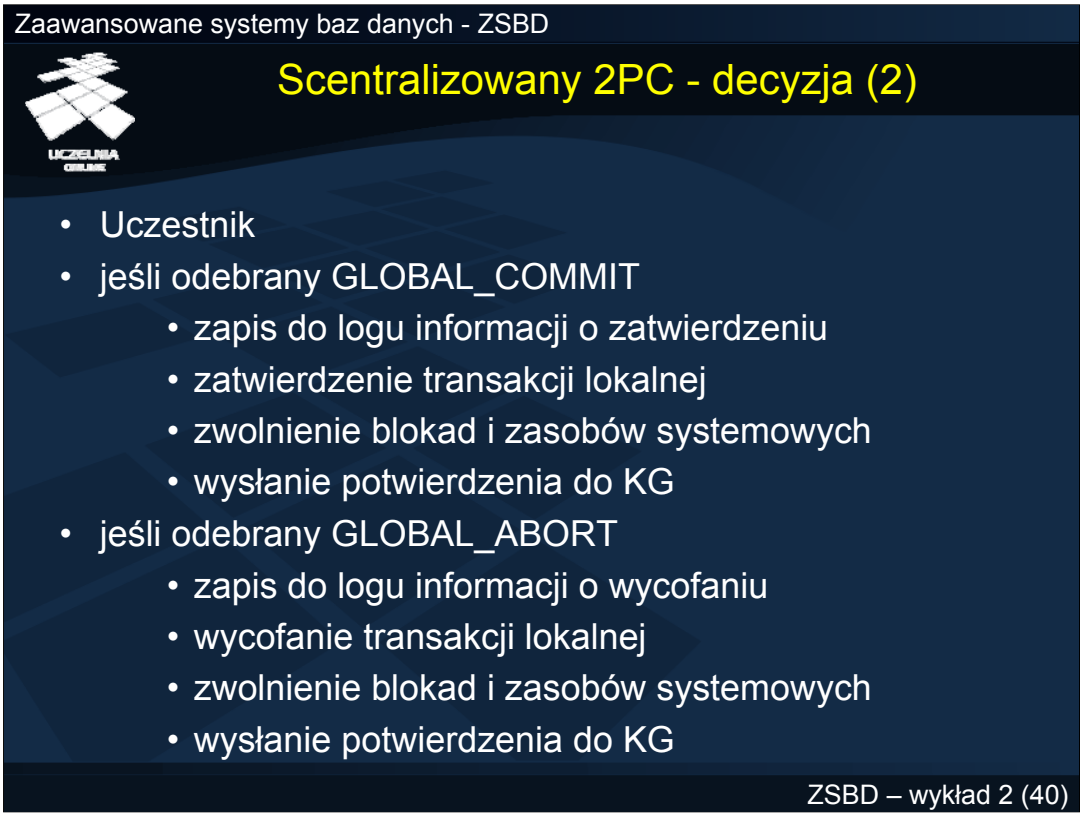


W fazie decyzji, koordynator globalny wykonuje następujące czynności.

Jeżeli wszystkie odebrane komunikaty to READY_COMMIT, wówczas KG zapisuje do swojego logu rekord mówiący o zatwierdzeniu transakcji rozproszonej. Następnie wysyła komunikat GLOBAL_COMMIT do uczestników.

Jeżeli przynajmniej jeden uczestnik odpowiedział komunikatem ABORT, wówczas KG zapisuje do swojego logu rekord mówiący o wycofaniu transakcji rozproszonej. Następnie wysyła komunikat GLOBAL_ABORT do uczestników.

Po wysłaniu komunikatu, KG przechodzi w stan oczekiwania na potwierdzenia uczestników.



W fazie decyzji uczestnik wykonuje następujące czynności.

Jeżeli odebrał komunikat GLOBAL_COMMIT, wówczas:

- zapisuje w swoim logu rekord informujący o zatwierdzeniu swojej transakcji lokalnej,

- zatwierdza transakcję lokalną,

- zwalnia blokady założone przez transakcję lokalną,

- zwalnia zasoby systemowe przeznaczone do obsługi transakcji lokalnej,

- wysyła potwierdzenie zatwierdzenia transakcji do KG.

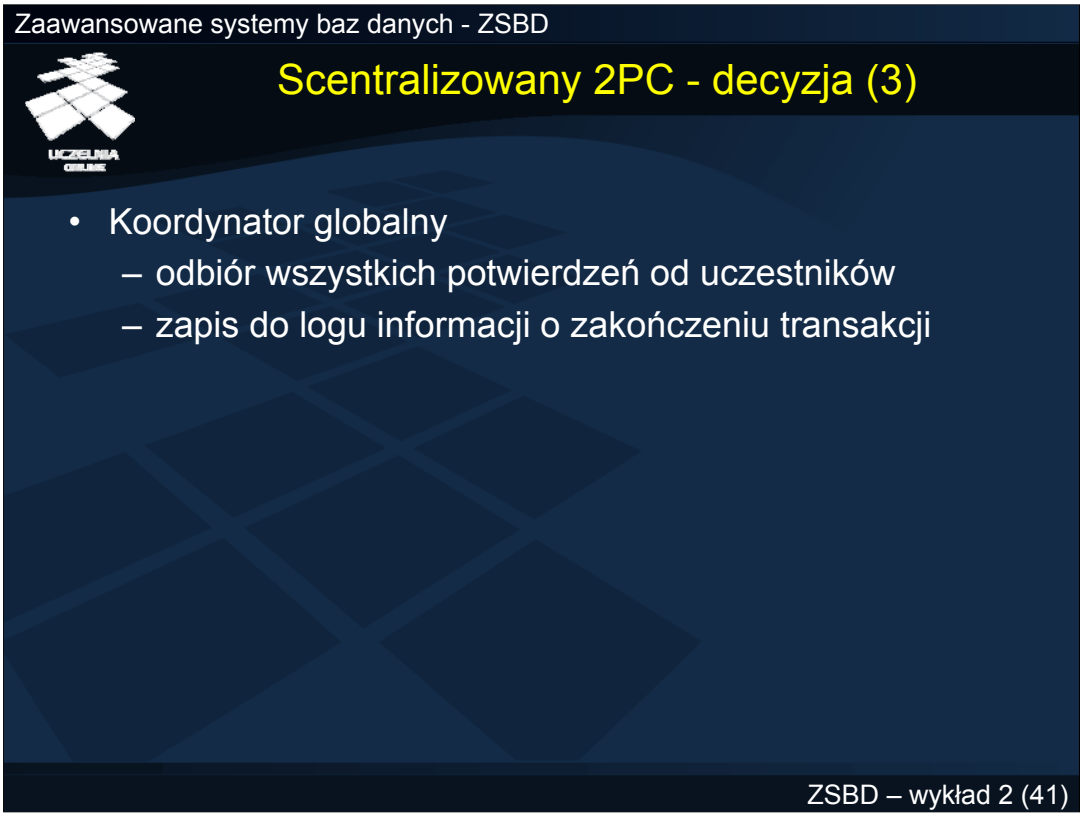
Jeżeli uczestnik odebrał komunikat GLOBAL_ABORT, wówczas:

- zapisuje w swoim logu rekord informujący o wycofaniu swojej transakcji lokalnej,

- wycofuje transakcję lokalną,

- zwalnia blokady i zasoby systemowe,

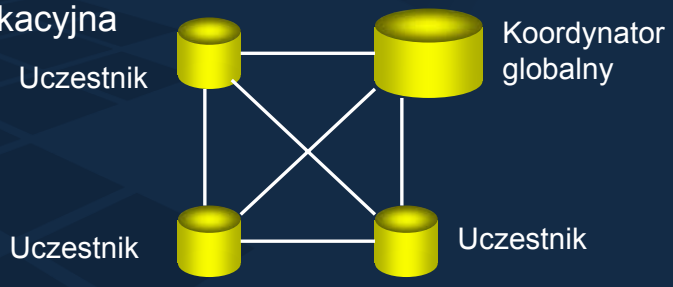
- wysyła potwierdzenie wycofania transakcji do KG.



W ostatnim kroku fazy decyzji, KG odbiera potwierdzenia o zatwierdzeniu od wszystkich uczestników i zapisuje do logu rekord zakończenia transakcji.

Zdecentralizowany 2PC (1)

- Topologia komunikacyjna

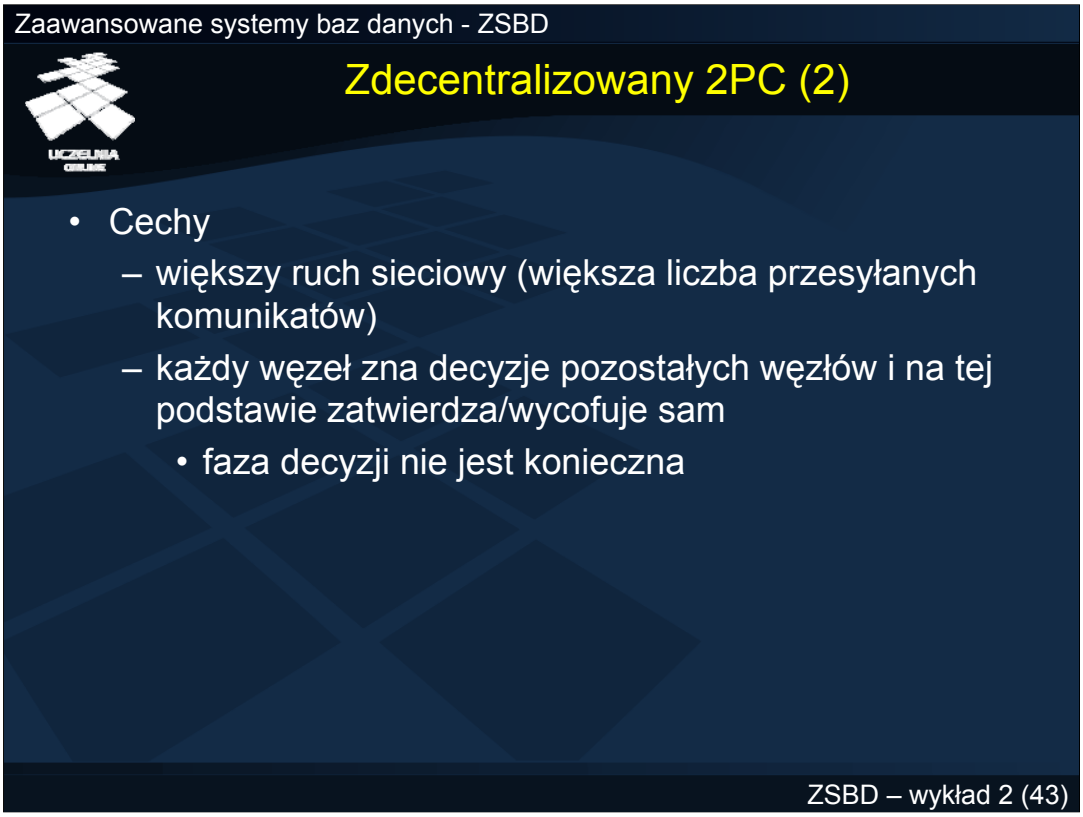


- Koncepcja
  - KG wysyła komunikat GLOBA_COMMIT (GLOBAL_ABORT) do wszystkich uczestników
  - każdy uczestnik wysyła odpowiedz do wszystkich pozostałych węzłów

W architekturze zdecentralizowanej nie ma centralnego węzła - każdy węzeł komunikuje się z każdym.

Koncepcja działania protokołu 2PC w architekturze zdecentralizowanej jest następująca.

KG wysyła komunikat GLOBA_COMMIT lub GLOBAL_ABORT do wszystkich uczestników. Uczestnicy przygotowują się w sposób identyczny do omówionego wcześniej, ale komunikat READY_COMMIT bądź ABORT wysyłają do wszystkich węzłów. W ten sposób, każdy węzeł ma pełen obraz stanu transakcji rozproszonej.



Zdecentralizowany protokół 2PC cechuje się:

- większym ruchem sieciowym ze względu na większą liczbę przesyłanych komunikatów,

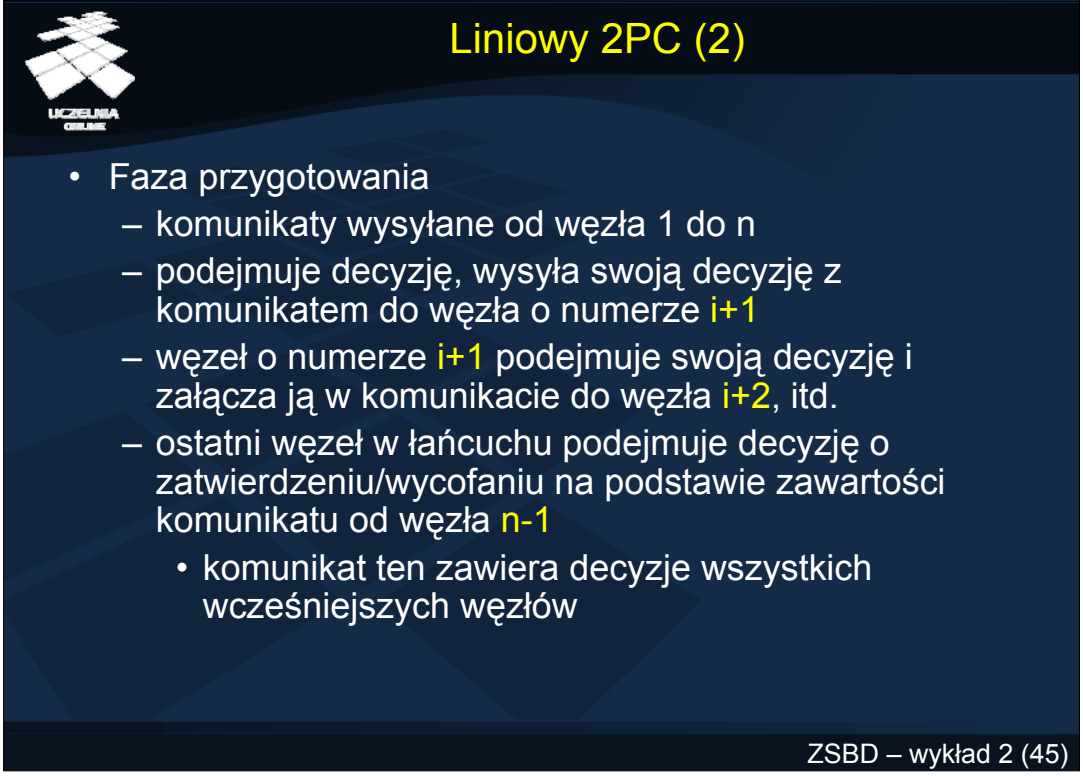
- możliwością wyeliminowania fazy decyzji ponieważ każdy uczestnik zna odpowiedź pozostałych uczestników i na jej podstawie może podjąć decyzję o zatwierdzeniu lub wycofaniu swojej transakcji lokalnej.

Liniowy 2PC (1)

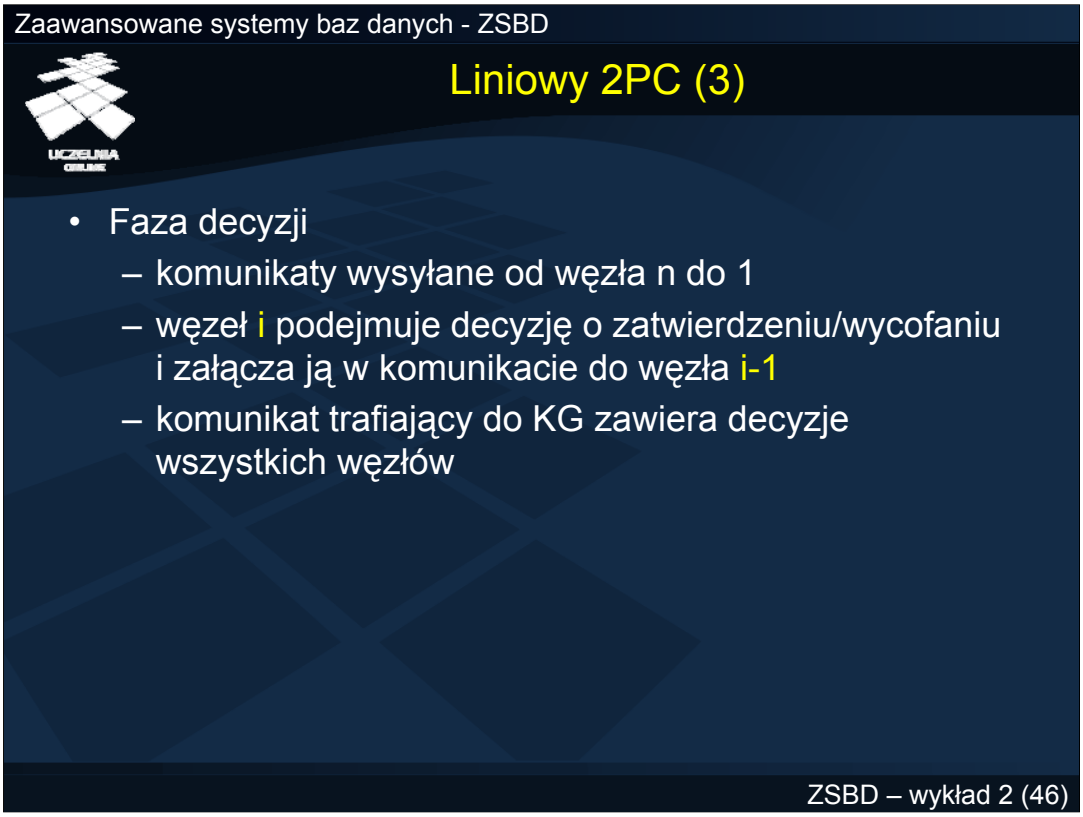
- Topologia komunikacyjna

- Koncepcja
  - węzły są uporządkowane liniowo
  - każdy węzeł otrzymuje numer od 1 (KG) do n (U)
  - węzeł o numerze i komunikuje się tylko z węzłami sąsiednimi, tj. o numerach i-1, i+1

W architekturze liniowego 2PC węzły są uporządkowane liniowo i każdy z nich otrzymuje numer. Węzeł KG otrzymuje numer 1. Uczestnicy otrzymują kolejne numery, 2, 3, do n. W czasie wymiany komunikatów, każdy węzeł komunikuje się tylko z węzłami sąsiednimi, tj. węzłem bezpośrednio go poprzedzającym i węzłem bezpośrednio po nim następującym.



W fazie przygotowania, komunikaty są wysyłane przez KG do węzła o numerze 2. Węzeł ten po przygotowaniu się dokłada do komunikatu otrzymanego od KD swoją decyzję. Taki rozszerzony komunikat jest następnie wysyłany do węzła 3. Węzeł 3 po przygotowaniu się dokłada swoją decyzję do komunikatu otrzymanego od węzła 2 i wysyła tak rozszerzony komunikat do węzła 4. Każdy kolejny węzeł rozszerza otrzymany komunikat o swoją decyzję dopóki komunikat nie dotrze do węzła ostatniego. Po otrzymaniu komunikatu, węzeł ostatni podejmuje decyzję o zatwierdzeniu lub wycofaniu swojej transakcji. Decyzja ta jest podejmowana na podstawie zawartości komunikatu, który otrzymał. Zawiera on bowiem informacje o przygotowani lub nieprzygotowaniu się wszystkich wcześniejszych węzłów.



Po podjęciu decyzji o zatwierdzeniu (wycofaniu) ostatni węzeł umieszcza tę decyzję w komunikacie wysyłanym do węzła poprzedniego (o numerze n-1). W fazie decyzji komunikat wędruje od węzła ostatniego do KG. Każdy węzeł po drodze dopisuje do otrzymanego komunikatu swoją decyzję. W ten sposób, komunikat docierający do KG zawiera decyzje wszystkich węzłów. Na tej podstawie KG podejmuje decyzję o ostatecznym zatwierdzeniu lub wycofaniu transakcji. Decyzja ta jest wysyłana komunikatem do węzła nr 2 i propaguje się do węzła ostatniego.