

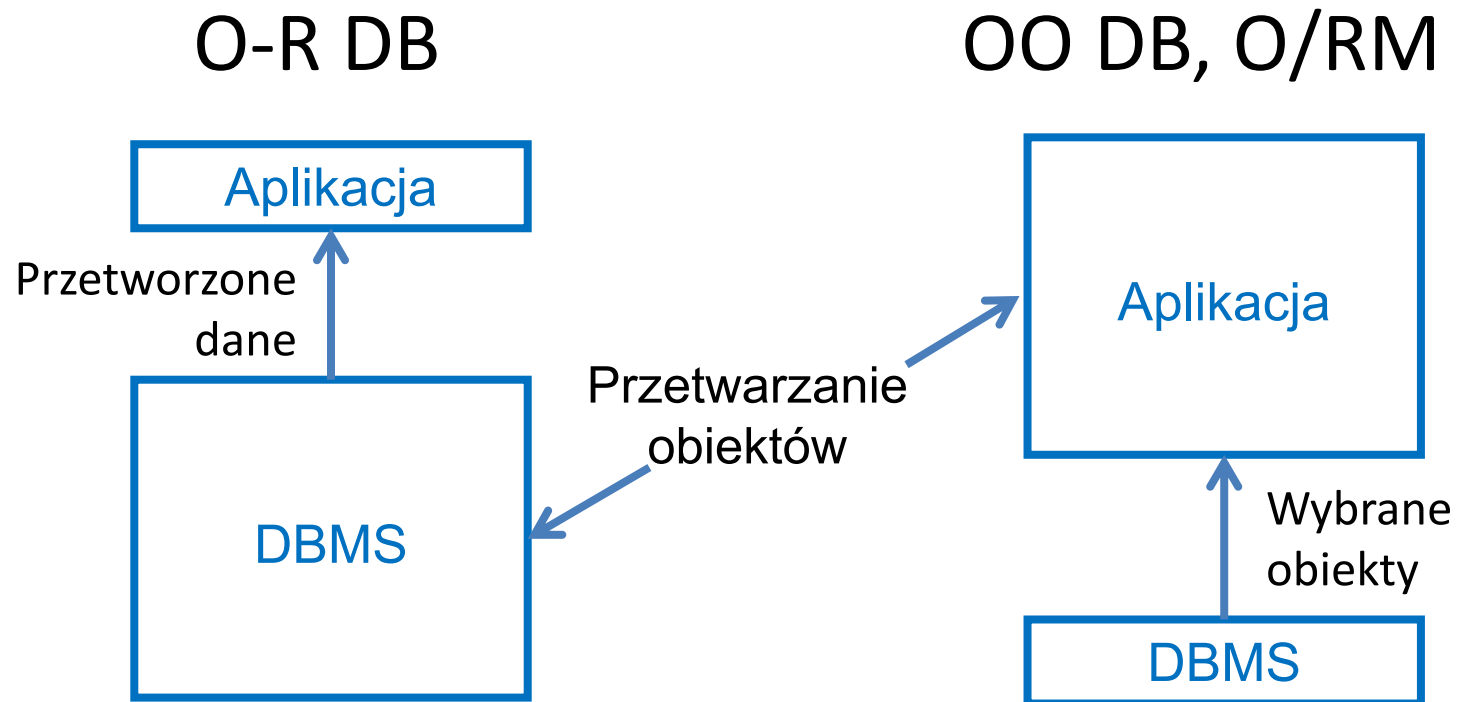
Obiektowe bazy danych

**Wykład prowadzi:
Tomasz Koszlajda**

Przykład systemu obiektowej bazy danych - db4o

- Projekt Open Source
- Uproszczenie architektury systemu bazy danych
- Dostępne różne architektury aplikacji
- Różne języki dostępu: QBE, SODA, Native Query Language
- Skalowalność
- Wieloplatformowość – Java, .Net, VBA
- Wydajność – dla zapytań specyficznych dla obiektowych baz danych

Nowa architektura przetwarzania



Prostota budowy aplikacji

```
// zdefiniowanie klasy
public class Pracownik {
    public Pracownik(String nazwisko, String etat,
        float placa)
    ...}
// otworzenie prywatnej bazy danych
ObjectContainer db = Db4o.openFile("BazaDanych");
Pracownik p=new Pracownik("Tarzan","Prezes",25000);
// składowanie obiektu w bazie danych
db.store(p);
// zamknięcie bazy danych
db.close();
```

Klasa: Object Container

- Klasa `object Container` reprezentuje system bazy danych
- Obsługuje ona dostęp do lokalnego, prywatnego pliku danych
- Wspiera transakcyjne przetwarzanie danych
- Zarządza referencjami do obiektów przechowywanych w bazie danych
- Zapewnia wydajne przetwarzanie danych w bazie danych, przez używanie indeksów

Schemat bazy danych

Definicje klas w aplikacji = schemat bazy danych

```
// zdefiniowanie klasy
public class Pracownik {
    public Pracownik(String nazwisko, String etat,
        float placa){
        ...
    }
    Pracownik p=new Pracownik("Tarzan","Prezes",25000);
    // Jeżeli jest to pierwszy obiekt klasy Pracownik
    // składowany w bazie danych, do schematu bazy danych
    // dodawana jest definicja klasy Pracownik
    db.store(p);
```

Hierarchia rozszerzeń klas

Wystąpienia podklas tworzą podzbiór wystąpień nadklas

```
class Kierownik extends Pracownik {...}  
Kierownik k = new Kierownik("Kowalski", 4500.00);  
db.store(k);
```

```
...
```

/* Obiekt k będzie osiągalny zarówno jako element zbioru Kierowników i jaki Pracowników */

// Kowalski znajdzie się w wyniku zapytania

```
ObjectSet wynik = db.get(Pracownik.class);
```

Języki zapytań w db4o

System bazy danych db4o oferuje trzy różne języki zapytań:

- QBE – Query By Example
- SODA – Simple Object Database Access
- NativeQuery

Preferowane własności języków zapytań:

1. Języki do *masowego* przetwarzania danych
2. Czyste języki obiektowe
3. Statyczna weryfikacja typów danych
4. Jednorodność z językiem aplikacji
5. Systemowa optymalizacja zapytań

Język zapytań QBE

Prosty język zapytań QBE – Query by Example

Wymaga utworzenia obiektu, który będzie wzorcem dla zapytania

```
// wyszukiwanie obiektów - QBE  
Pracownik p=new Pracownik(null,"Prezes",0.0f);  
// nie można pytać o wartości: null, 0, ""  
ObjectSet wynik = db.get(p);  
// obiektem wzorcowym może być klasa  
wynik = db.get(Pracownik.class);
```

Nie umożliwia definiowania złożonych warunków zapytań: AND, OR, NOT. Konstruktor obiektu musi umożliwić tworzenie obiektu bez zainicjowanych atrybutów : null, "".

Język zapytań SODA – Simple Object Database Access

Język SODA umożliwia definiowanie złożonych zapytań w postaci drzew, których węzłami są warunki selekcji.

Funkcjonalność języka zapytań jest zaimplementowana w dwóch podstawowych klasach: **Constraint** i **Query**

Ich funkcjonalność obejmuje następujące operacje:

Query:

- descend
- constrain
- sortBy
- Execute
- orderAscending

Constraint:

- and (Constraint)
- contains
- equal
- greater
- identity

- like
- not
- or (Constraint)
- smaller

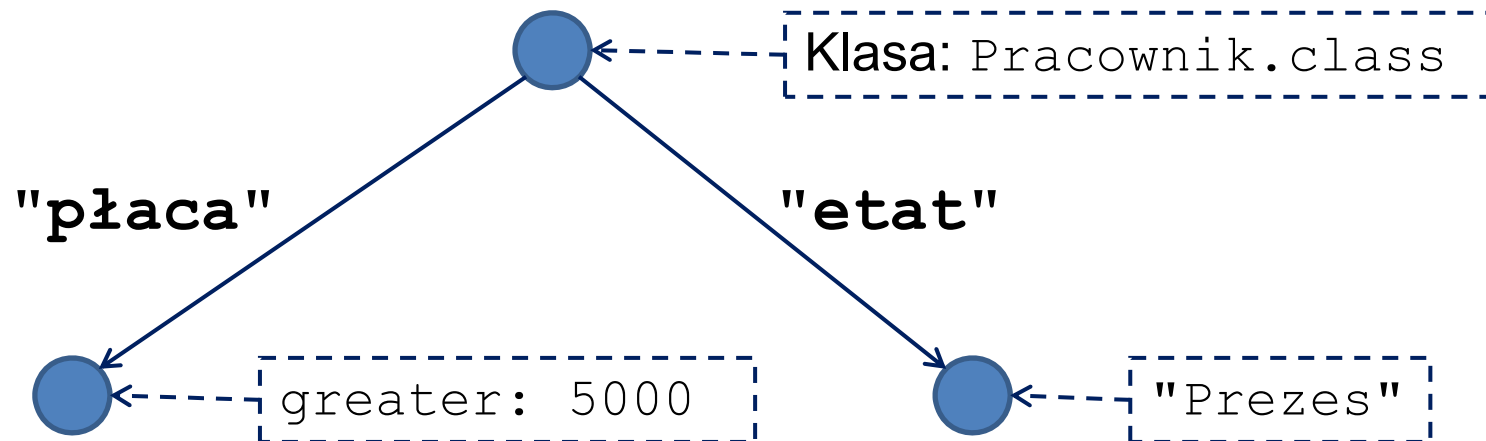
Język zapytań SODA – Simple Object Database Access

```
Query query=db.query();  
// określenie klasy zapytania  
query.constrain(Pracownik.class);  
// określenie warunku prostego  
Constraint warunek =  
    query.descend("placa").constrain(5000).greater();  
// określenie drugiego warunku prostego  
// i złożenie warunków  
query.descend("etat").constrain("Prezes").or(warunek);  
// wykonanie zapytania  
ObjectSet wynik=query.execute();  
  
System.out.println(wynik.size());  
while(wynik.hasNext()) {  
    Pracownik znaleziony = (Pracownik) wynik.next();  
    System.out.println(znaleziony); }  

```

Język zapytań SODA – drzewo zapytania

```
Query query=db.query();  
query.constrain(Pracownik.class);  
query.descend("placa").constrain(5000).greater().or  
(descend("etat").constrain("Prezes"))
```



Języki zapytań w obiektowych bazach danych

OQL

```
String oql =  
"select * from o in Osoby where p.wiek < 20";  
OQLQuery query = new OQLQuery(oql);  
Object osoba = query.execute();
```

Zapytania jako tekst



```
graph TD
    A["wiek < 20"] --> B["wiek < 20"]
    A --> C["wiek"]
```

JDOQL

```
Query query =  
persistenceManager.newQuery(Osoba.class, "wiek < 20");  
Collection osoby = (Collection)query.execute();
```

db4o SODA - C#

```
Query query = database.Query();  
query.Constrain(typeof(Osoba));  
query.Descend("wiek").Constrain(20).Smaller();  
IList osoby = query.Execute();
```

Słabości języków zapytań w bazach danych

- Brak syntaktycznej i semantycznej weryfikacji zapytań w trakcie kompilacji aplikacji
- Brak wsparcia dla refaktoryzacji
- Brak wsparcia dla wielokrotnego użytku: wywoływania metod, polimorfizmu, redefinicji
- Nieodporność na ataki typu „wstrzykiwanie kodu”
- Wymagany dostęp do prywatnych atrybutów klas
- Brak systemowej optymalizacji zapytań dla dużych zbiorów danych
- Brak skalowalności

Konstrukcja języka zapytań NativeQuery

Warunki logiczne służące do selekcji obiektów są wyrażane w obiektowym języku programowania:

```
prac.Placa() > min
```

Złożone wyrażenia logiczne zapytania są tworzone w całości w obiektowym języku programowania:

```
prac.Etat().equals("Prezes") ||  
    (prac.Placa() > min && prac.Placa() < max)
```

Mechanizm przesyłania obiektów do zapytań:

// pseudo kod

```
(Pracownik prac) {  
    return prac.Etat().equals("Prezes") ||  
        (prac.Placa() > min && prac.Placa() < max) }
```

Język zapytań NativeQuery

Generyczna klasa systemowa ***Predicate*** z abstrakcyjną metodą ***match*** zwracającą wartości logiczne.

```
public abstract class Predicate <ExtentType> {  
    public <ExtentType> Predicate () {}  
    public abstract boolean match(ExtentType candidate) ; }
```

```
ObjectSet sp=db.query(new Predicate<Pracownik>()  
// redefinicja metody match
```

```
{ public boolean match(Pracownik prac) {  
    return prac.wezNazwisko().equals("Tarzan")  
        || (prac.Placa() > 4500  
            && prac.Placa() < 10000) ; } ;  
};
```

Definicja anonimowej klasy dziedziczącej
po klasie Predicate

Język zapytań NativeQuery

Wyrażenia ścieżkowe w NativeQuery

```
ObjectSet sp=db.query(new Predicate<Pracownik>()  
// dziedziczenie i redefinicja metody match  
{ public boolean match(Pracownik prac) {  
    return  
        prac.Zespol().nazwa().equals („Zarząd”);  
    }  
});
```

Język zapytań NativeQuery – C#

Zastosowanie typu danych **delegate**

```
ObjectSet sp=db.query<Pracownik> (  
                                delegate Pracownik p)  
// definicja operacji uruchamianej jako delegate p  
    {  
        return  
        prac.wezNazwisko().equals("Tarzan")  
        || (prac.Placa() > 4500  
        && prac.Placa() < 10000);  
    }  
);
```

Usuwanie i modyfikacja obiektów

```
Pracownik p=new Pracownik(null,null,10000.00);
ObjectSet wynik = db.get(p);
while(wynik.hasNext()) {
    Pracownik prac=(Pracownik)wynik.next();
// usunięcie
    db.delete(prac); }
...
while(wynik.hasNext()) { kopia obiektu
    Pracownik prac=(Pracownik)wynik.next();
// modyfikacja obiektu na podstawie jego OID
    prac.podniesPlace(1500);
    db.store(prac); // uspojnienie kopii z oryginałem
}
```

Powiązania między danymi

System db4o respektuje związki między obiektami zdefiniowanymi w aplikacji

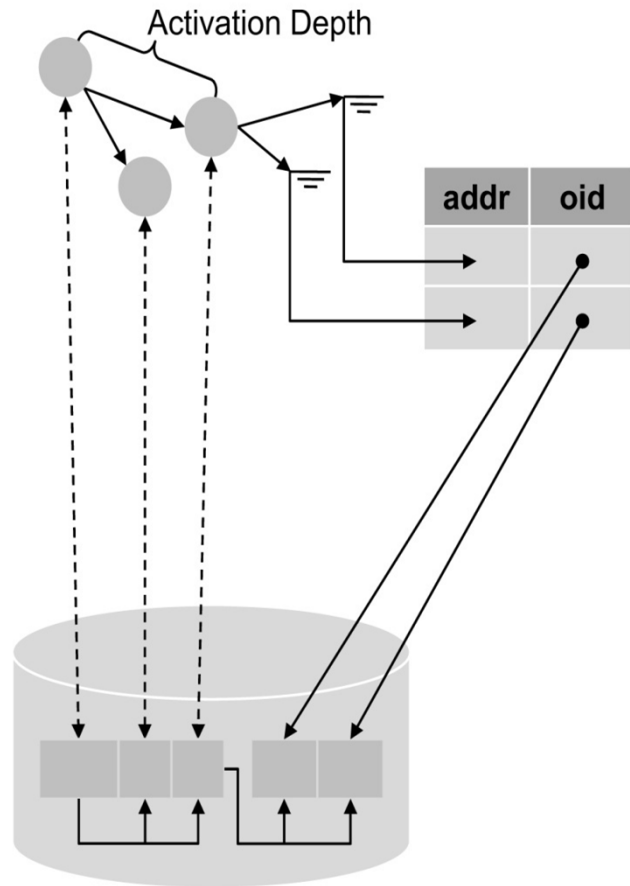
```
public class Pracownik {  
    ...  
    private Pracownik szef;  
    private Set<Pracownik> podwladni;  
    public Pracownik(String nazwisko,String etat) {  
        this.podwladni= new HashSet<Pracownik>();  
    }  
    public void dodajPodwladnego(Pracownik pod) {  
        this.podwladni.add(pod);  
    }  
}
```

Składowanie złożonych obiektów

Metoda set gwarantuje niejawne zapisanie do bazy danych wszystkich powiązanych obiektów

```
Pracownik p = new Pracownik(  
    "Tarzan", "Prezes", 25000) ;  
Pracownik pp1 = new Pracownik(  
    "Nowak", "Sekretarka", 2500) ;  
Pracownik pp2 = new Pracownik(  
    "Kula", "Kierwoca", 3000) ;  
// Przypisanie szefowi podwładnych  
p.dodajPodwladnego(pp1) ;  
p.dodajPodwladnego(pp2) ;  
// Składowanie obiektów w bazie danych  
db.store(p) ;  
// W bazie danych znajdą się wszystkie powiązane obiekty
```

Odczyt powiązanych obiektów



Załadowanie do pamięci operacyjnej obiektu jako wyniku zapytania może wymagać dostępu do kolejnych powiązanych obiektów. Stosowane są dwie strategie:

1. Natychmiastowe załadowanie wszystkich powiązanych obiektów (cała baza danych).
2. *Leniwe* ładowanie obiektów na żądanie.

W **db4o** możliwe jest elastyczne dopasowanie zbioru ładowanych obiektów za pomocą parametru **Activation Depth** (domyślna wartość = 5).

Kaskadowe usuwanie obiektów złożonych

- Usunięcie z bazy danych obiektu, który ma powiązania z innymi obiektami może wymagać usunięcia tych obiektów – semantyka związku kompozycji
- W tym celu, należy wykonać na klasie obiektu złożonego metodę `cascadeOnDelete()`
- Głębokość kaskadowego usuwania może być konfigurowana za pomocą metody: `set(object, depth)`

Indeksy w db4o

- System db4o umożliwia zakładanie indeksów na wybranych atrybutach obiektów składowanych w bazie danych.
- Indeks musi być utworzony przed otwarciem obiektu ObjectContainer/ObjectServer:

// w bazie danych składowane są obiekty klasy Pracownik

```
class Pracownik {  
    String nazwisko;  
    ...  
}
```

```
Db4o.configure().objectClass(Pracownik.class).  
    objectField("nazwisko").indexed(true);
```


Ewolucja schematu bazy danych

- W db4o schemat tworzą definicje klas i ich wzajemne zależności: referencje i dziedziczenie. Schemat bazy danych musi nadążać za zmianami w klasach definiowanych w aplikacjach: dodawanie i usuwanie klas, zmiana własności klas, zmiana sieci powiązań między klasami.
- Zmiany w schemacie wiążą się również z refaktoryzacją kodu programów. W czysto obiektowych bazach danych refaktoryzacja jest prostsza niż w bazach O-R i O/RM ze względu na jeden spójny schemat bazy danych i definicje klas w aplikacjach.

Transakcyjność db4o

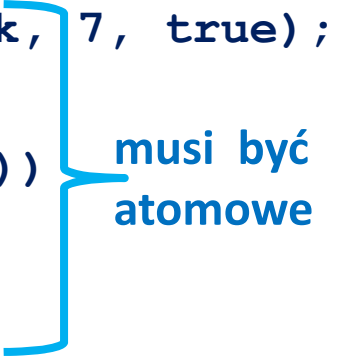
- Przetwarzanie danych w db4o ma charakter transakcyjny. Niejawnym rozpoczęciem transakcji jest otwarcie bazy danych. Niejawnym zakończeniem jest zamknięcie bazy danych.
- Funkcjonalność bazy danych obejmuje jawne zatwierdzanie i wycofywanie transakcji.
- Synchronizacja transakcji nie jest transparentna dla programisty. Aby ją zapewnić należy jawnie wywoływać dodatkowe funkcje systemowe. Systemowo zapewniona jest własność ***read committed***.

```
String bd = "BazaDanych";  
ObjectContainer db = Db4o.openFile(bd);  
Pracownik p=new Pracownik("Tarzan","Prezes",25000);  
// składowanie obiektu w bazie danych  
db.store(p);  
// zatwierdzenie transakcji  
db.commit();
```

Synchronizacja współbieżnych transakcji

Unikanie anomalii **lost update**, będącej konsekwencją optymistycznego zarządzania współbieżnością, może być realizowane z zastosowaniem metody systemowej **peekPersisted** klasy **ExtObjectContainer**. Metoda tworzy kopię wartości wskazanego obiektu, do określonego poziomu zagnieżdżenia, nie powiązanej z obiektem w bazie danych. Kopia ta może zostać wykorzystana do weryfikacji aktualnego stanu obiektu przed jego modyfikacją w bazie danych.

```
Nowak = (Pracownik)result[0]; // odczyt obiektu z bazy danych
Nowak_przed_modyf = Nowak.clone(); // kopia obiektu
... // przetwarzanie obiektu Nowak
Pracownik akt_kopia = oc.ext().peekPersisted(Nowak, 7, true);
// porównaj wartości, w przypadku różnicy wycofaj transakcję
if (akt_kopia.płaca() == Nowak_przed_modyf.płaca())
{
    oc.commit(); }
else {
    oc.rollback(); }
```



musi być atomowe

Synchronizacja współbieżnych transakcji

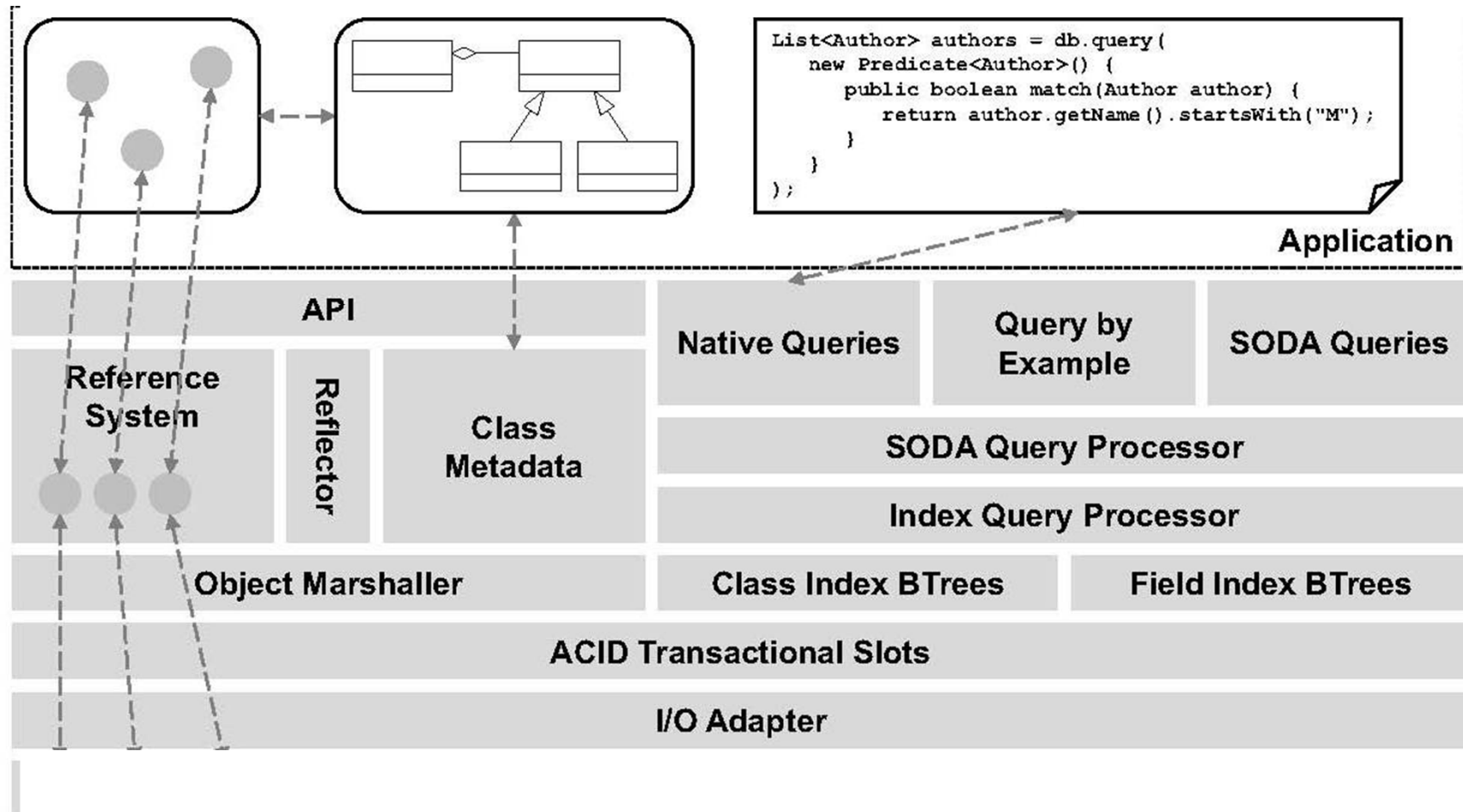
- System db4o umożliwia synchronizację współbieżnych transakcji przez jawne zakładanie blokad na cały system.
- Blokady służą do synchronizacji dostępu do sekcji krytycznych.
- Blokady są globalne, ale rozróżnialne przez nazwy

```
ObjectContainer oc = Db4o.openClient(...);  
...  
if (oc.ext().setSemaphore("LOCK_"+oc.ext().getID(obj), 1000)) {  
    // wejdź do sekcji krytycznej  
    ...  
    // zwolnij blokadę (semafor) po zakończeniu sekcji krytycznej  
    oc.ext().releaseSemaphore("LOCK_"+oc.ext().getID(obj));  
}
```

czas czekania na
zwolnienie blokady [ms]

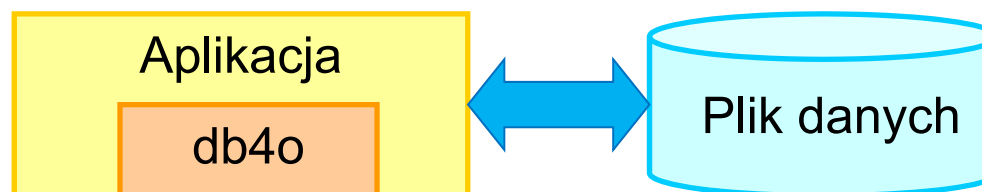
blokada na obiekcie *obj*

Architektura systemu db4o



Architektura aplikacji db4o

Pojedyncza aplikacja zintegrowana z plikiem danych



// otworzenie bazy danych

```
String bd = "BazaDanych";
```

```
ObjectContainer db = Db4o.openFile(bd);
```

```
Pracownik p=new Pracownik("Tarzan","Prezes");
```

// niejawne zdefiniowanie schematu i

// składowanie obiektu w bazie danych

```
db.store(p);
```

// zamknięcie bazy danych

```
db.close();
```

Architektura aplikacji db4o

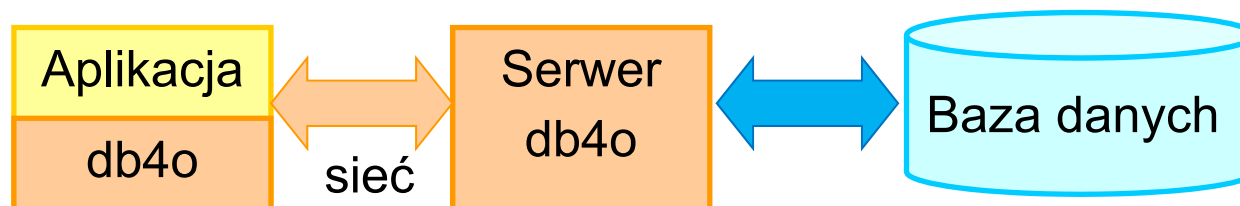
Lokalny serwer bazy danych



```
String bd = "BazaDanych";
ObjectServer server = Db4o.openServer(db, 0);
// port = 0, serwer lokalny
try { // otwarcie klienta
    ObjectContainer client = server.openClient();
    ...
    client.close();
}
finally {
    server.close();
}
```

Architektura aplikacji db4o

Architektura klient - serwer



```
String bd = "BazaDanych";  
// port <> 0, serwer zdalny  
ObjectServer server=Db4o.openServer(db,1521);  
server.grantAccess("user","password");  
// otwarcie klienta  
ObjectContainer client = server.openClient(  
    "localhost",port,user,password);  
...  
client.close();  
server.close();
```

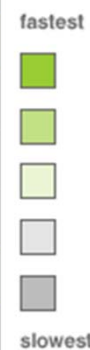

Wydajność

- Benchmark OO7
- Benchmark Pole Position

Barcelona circuit

- 100 zapytań na 30 000 obiektów o 5 poziomach dziedziczenia

barcelona benchmarks	read	write	query	delete
db4o/4.5.200	1.0	1.0	1.0	1.0
Hibernate / MySQL	20.8	32.2	6.7	17.3
Hibernate / HSQLDB	10.4	5.4	536.0	3.9
JDBC / MySQL	10.8	14.6	1.7	6.5
JDBC / HSQLDB	0.4	1.7	677.8	0.7
JDBC / Derby	3,696.0	12.9	1,299.7	7.1
JDO/VOA/MySQL	4.4	14.8	3.0	2.4



fastest

slowest

Producenci OODBs

- db4objects
- Objectivity/DB
- Progress ObjectStore Enterprise
- Versant Object Database, FastObject .Net
- GemStone/S, Gemstone Facets
- EyeDB – GNU Open Source