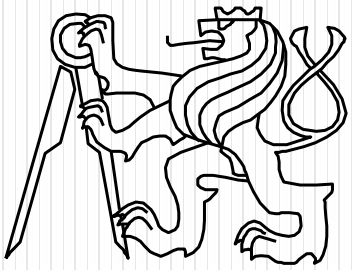
Enterprise Java (BI-EJA) Technologie programování v jazyku Java (X36TJV)

Ing. Zdeněk Troníček, Ph.D.

Katedra softwarového inženýrství

Fakulta informačních technologií ČVUT v Praze



Letní semestr 2010/2011, přednáška č. 4
https://edux.fit.cvut.cz/courses/BI-EJA
https://edux.feld.cvut.cz/courses/X36TJV

Agenda

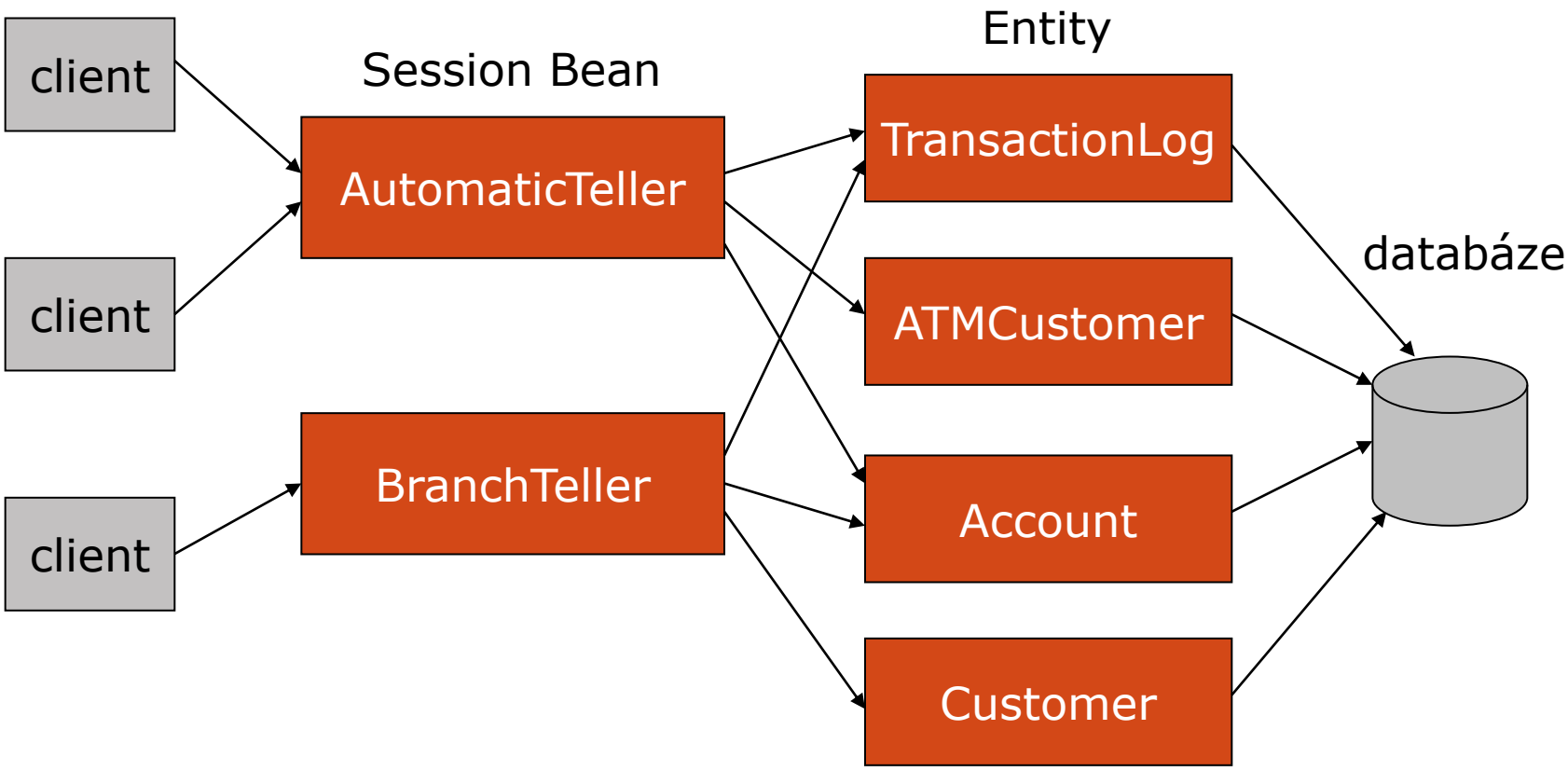
- Enterprise Java Beans (EJB) 3.1
- Java Persistence API (JPA) 2.0

Enterprise Java Beans 3.1

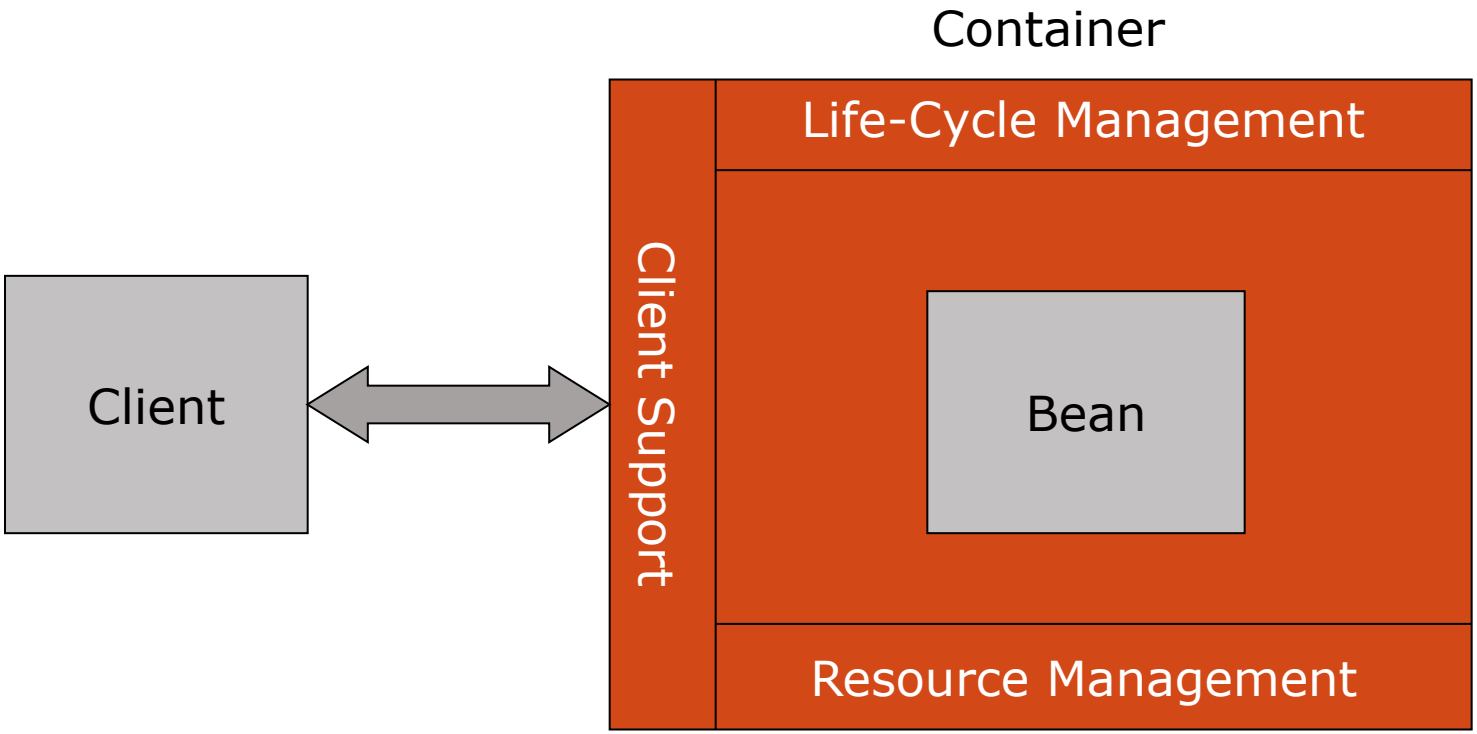
- Session Beans
- (Entity Beans 2.1)
- Message Driven Beans

Dependency Injection – kontejner se postará o nastavení odkazu např. na jinou beanu

EJB aplikace



EJB container



Session EJB

- Plain Old Java Object (POJO)
- bezestavové (@Stateless)
- stavové (@Stateful)
- singleton (@Singleton) (EJB 3.1)
- remote a local business interface
- no interface (EJB 3.1)

Implementace

Business Interface

```
import javax.ejb.Local;

@Local
public interface HelloLocal {
    String sayHello();
}
```

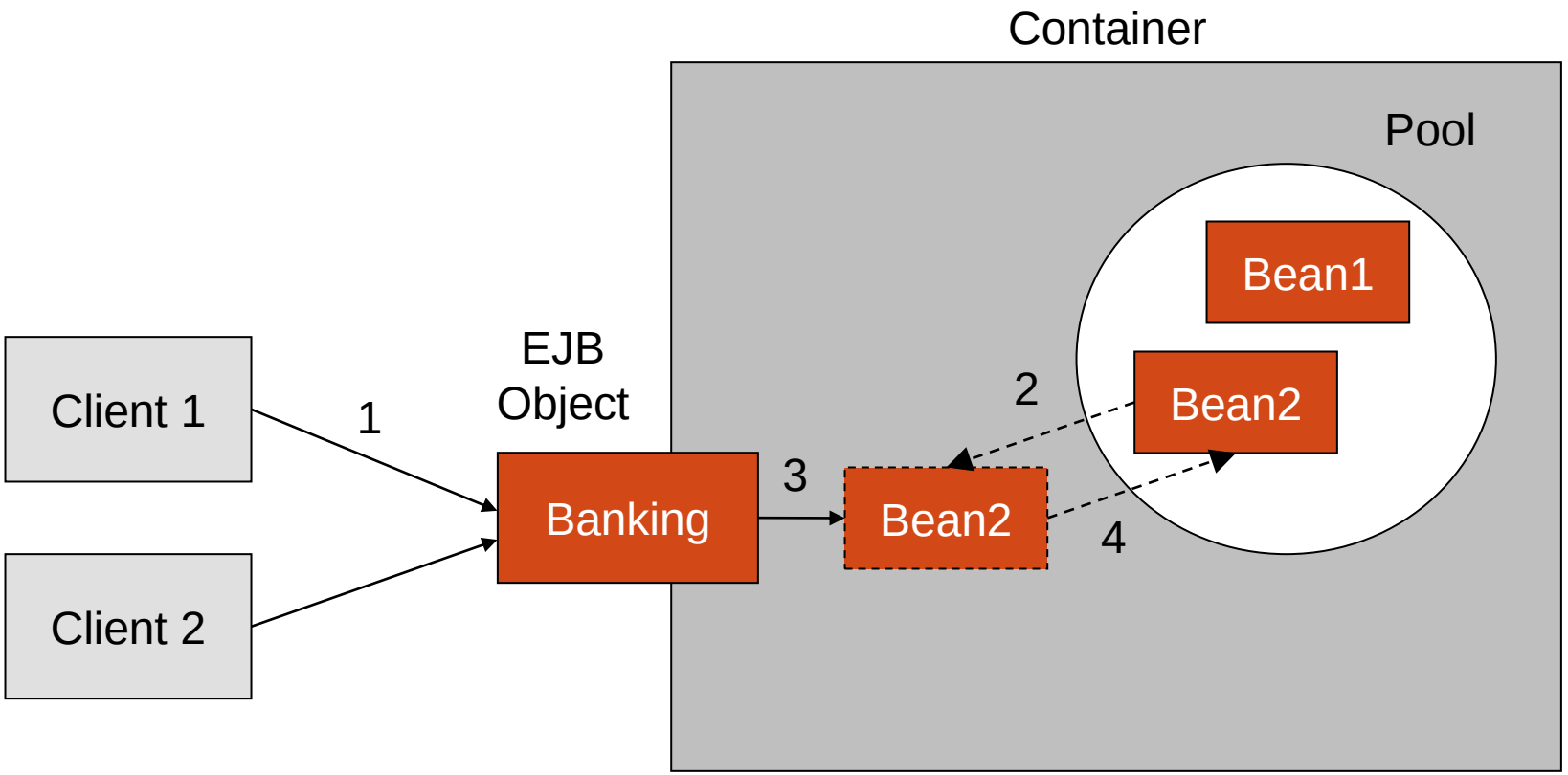
@Local
@Remote

Bean

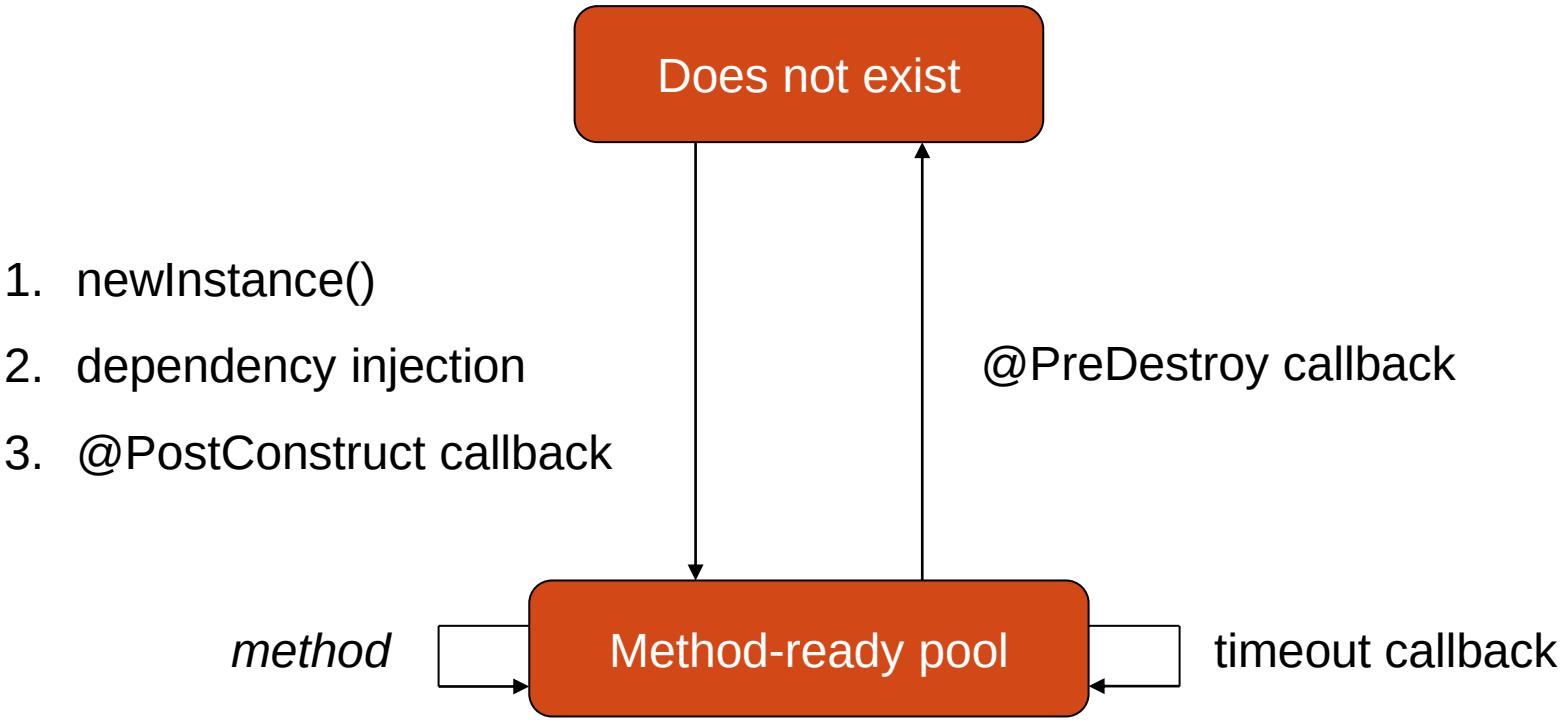
```
import javax.ejb.Stateless;

@Stateless
public class HelloBean implements HelloLocal {
    public String sayHello() {
        return "Hello!";
    }
}
```

@Stateless
@Stateful

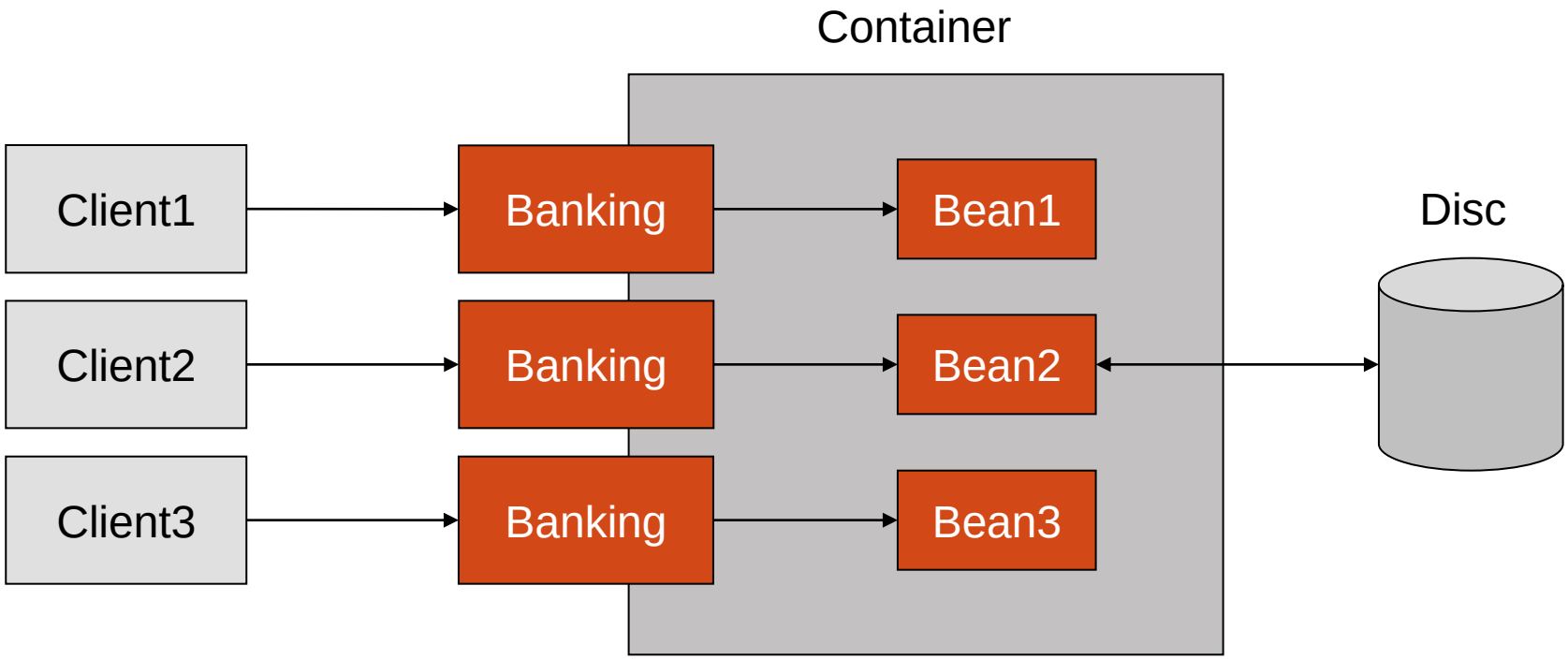
Stateless Session EJB



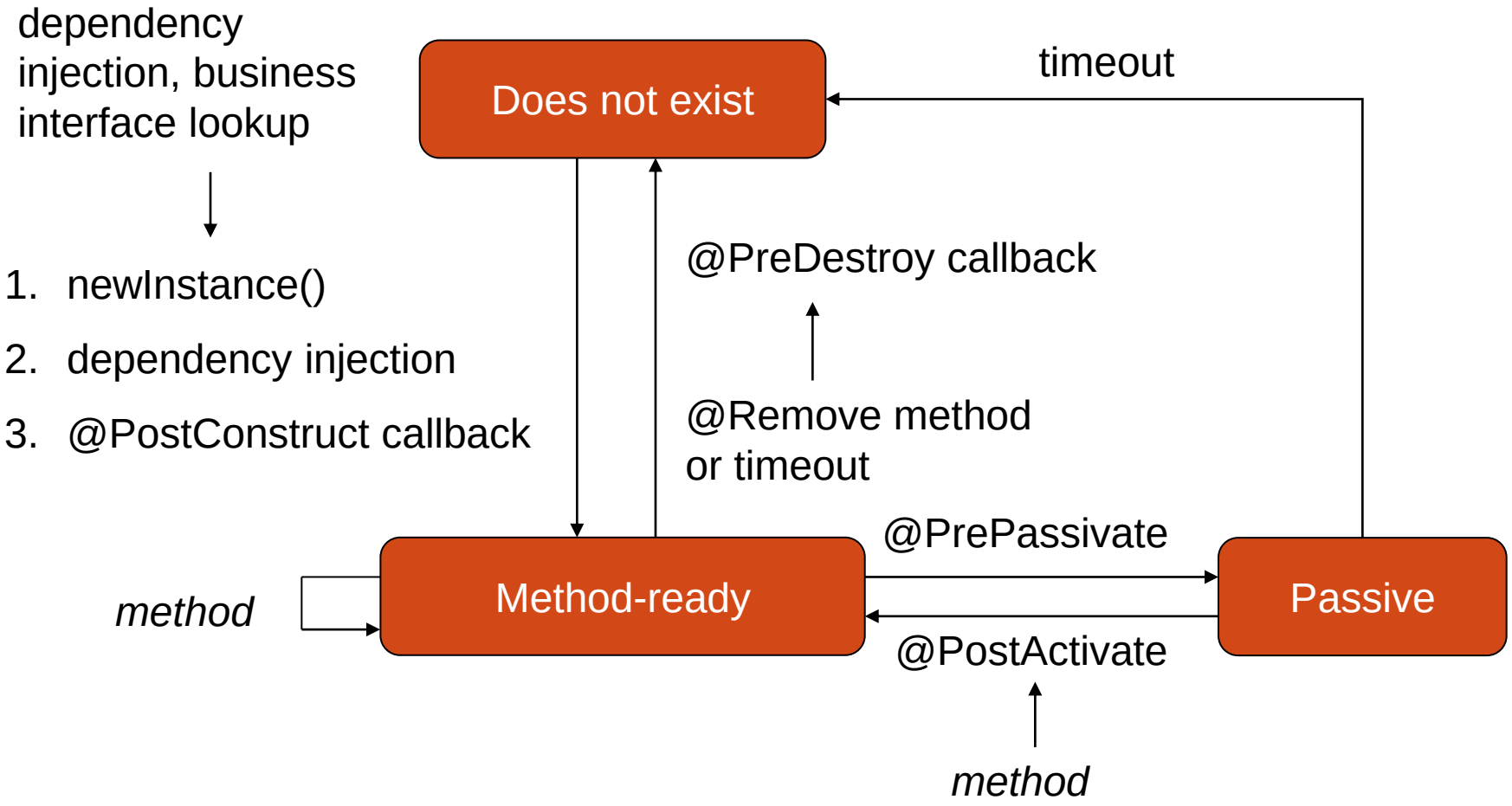
Stateless Session EJB Life Cycle



Stateful Session EJB



Stateful Session EJB Life Cycle



Singleton Session Beans

- Pouze jedna instance
- Lze stanovit pořadí inicializací (@DependsOn)
- Inicializace může být *eager* (@Startup)

```
@Startup
@Singleton( name = "A" )
@DependsOn( "B" )
public class ASessionBean { … }

@Startup
@Singleton( name = "B" )
public class BSessionBean { … }
```

Concurrency Management (1)

```
@Singleton
//@ConcurrencyManagement(
//      ConcurrencyManagementType.CONTAINER )
public class SingletonSessionBean {

    @Lock( LockType.READ )
    public void doSomething() { … }

    @Lock( LockType.WRITE )
    public void doSomethingElse() { … }
}
```

Concurrency Management (2)

```
@Singleton
@ConcurrencyManagement(
    ConcurrencyManagementType.BEAN )
public class SingletonSessionBean {

   public void doSomething() {
      …
      synchronized(this) { … }
   }
   public synchronized void doSomethingElse() { … }
}
```

Global JNDI Names

- java:global[/<app-name>]/<module-name>/<bean-name>[!<fully-qualified-interface-name>]
- java:app/<module-name>/<bean-name>[!<fully-qualified-interface-name>]
- java:module/<bean-name>[!<fully-qualified-interface-name>]

```
java:global/fooweb/FooBean
java:global/fooweb/FooBean!com.acme.Foo
java:app/fooweb/FooBean
java:app/fooweb/FooBean!com.acme.Foo
java:module/FooBean
java:module/FooBean!com.acme.Foo
```

Calendar Based Timer Service

Anotace @Schedule s atributy:

- year, month, dayOfMonth, dayOfWeek
- hour, minute, second
- timezone

```
@Singleton
public class ServiceBean {

    @Schedule( dayOfWeek = "Sun", hour = "2", minute = "30" )
    public void cleanDatabase() { … }
}
```

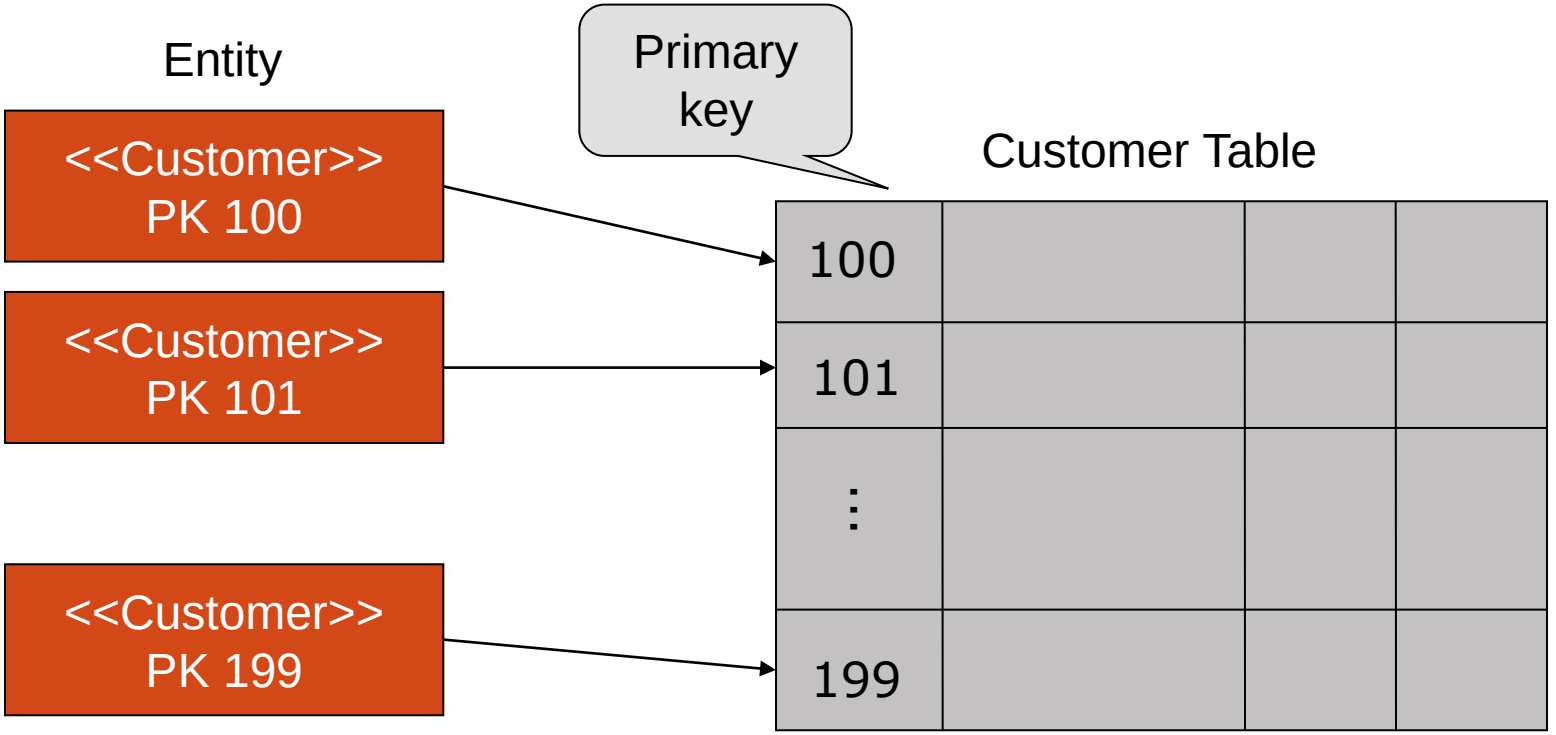
Asynchronní volání

- Metodu EJB lze volat asynchronně
- Návratovou hodnotou asynchronní metody je Future

```
@Stateless
public class MathSessionBean {
    @Asynchronous
    public Future<Integer> compute( Integer x, Integer y ) {
        Integer z = …
        return new AsyncResult<Integer>( z );
    }
}
```

```
Future<Integer> r = mathBean.compute( 20, 11 );
while ( !r.isDone() ) { … }
Integer i = r.get();
```

Entities


Entity
Primary key
Customer Table
<<Customer>>
PK 100
<<Customer>>
PK 101
<<Customer>>
PK 199
100
101
199

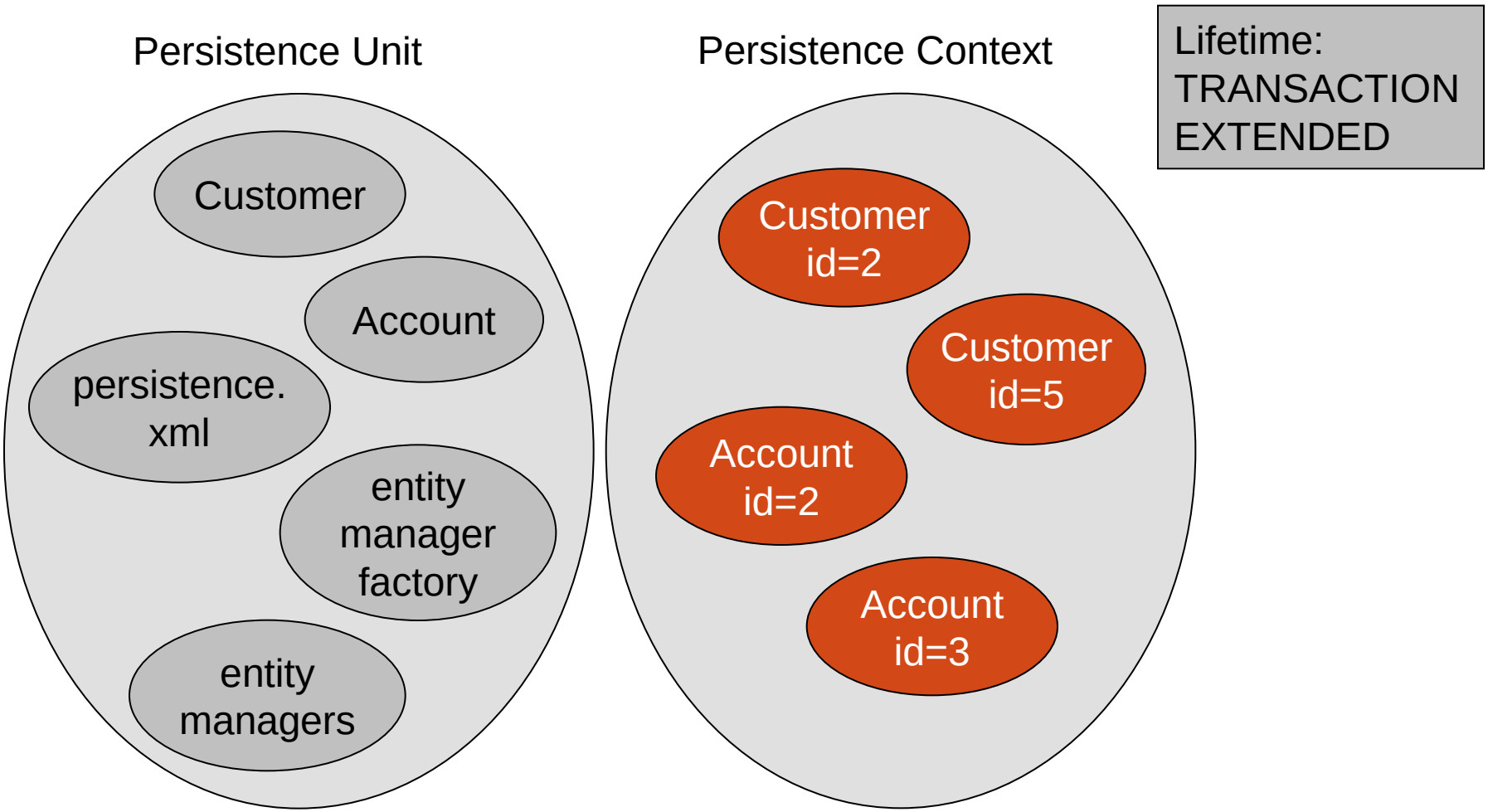
Entity

```
@Entity
public class Customer {
    @Id @Column(name = "IDENT")
    private Integer id;
    @Column(name = "JMENO",
        nullable = false)
    private String name;
    //public Customer() { }
    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    ...
}
```

```
@Entity
public class Account {
    @Id
    private Integer id;
    private BigDecimal balance;
    //public Account() { }
    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    ...
}
```

Configuration by exception

Persistence Unit & Context



Entity Manager

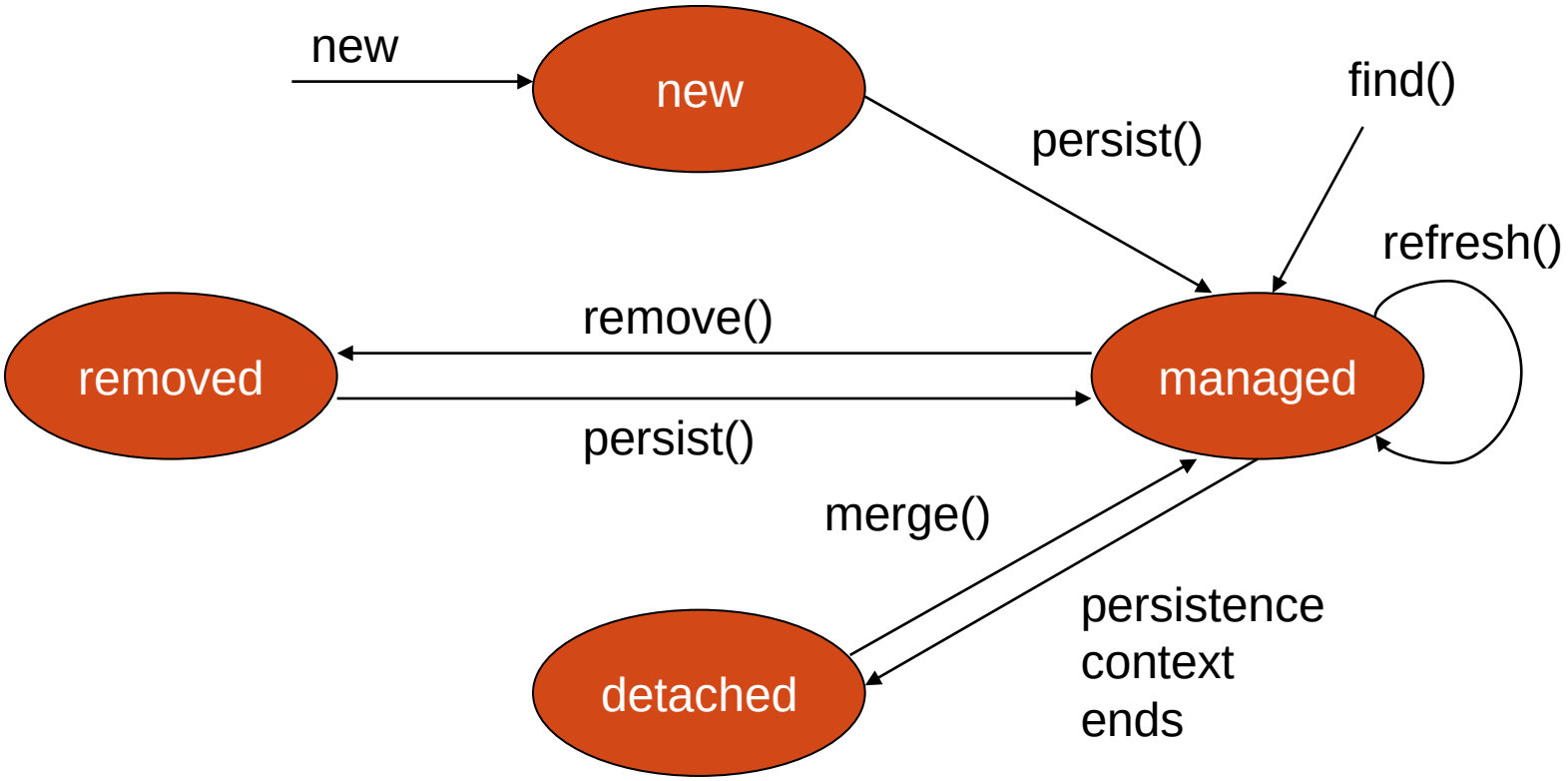
Java EE

```
@PersistenceContext private EntityManager em;

public Account openAccount( String owner ) {
    Account account = new Account();
    account.setOwner( owner );
    em.persist( account );
    return account;
}
```

Java SE

```
EntityManagerFactory emf =
    Persistence.createEntityManagerFactory( "pu1" );
EntityManager em = emf.createEntityManager();
em.getTransaction().begin();
em.persist(new Customer( 1, "Rumcajs" ));
em.getTransaction().commit();
em.close();
emf.close();
```

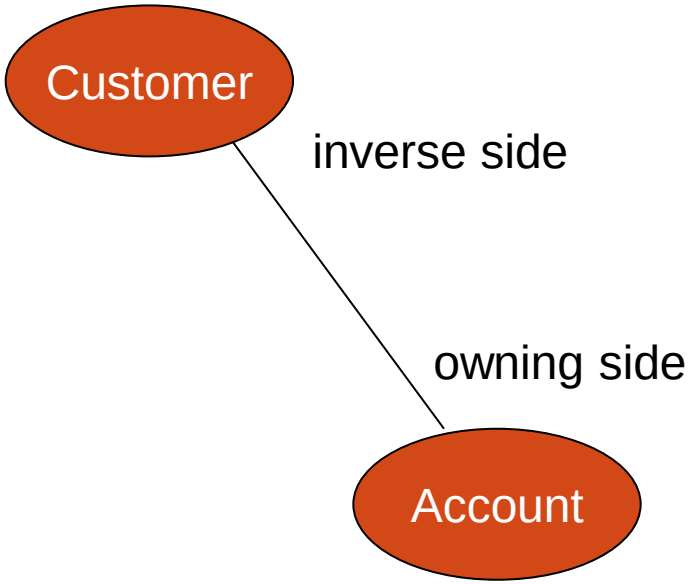
Entity Life Cycle



Vztahy mezi entitami

- One-to-one
- One-to-many
- Many-to-one
- Many-to-many

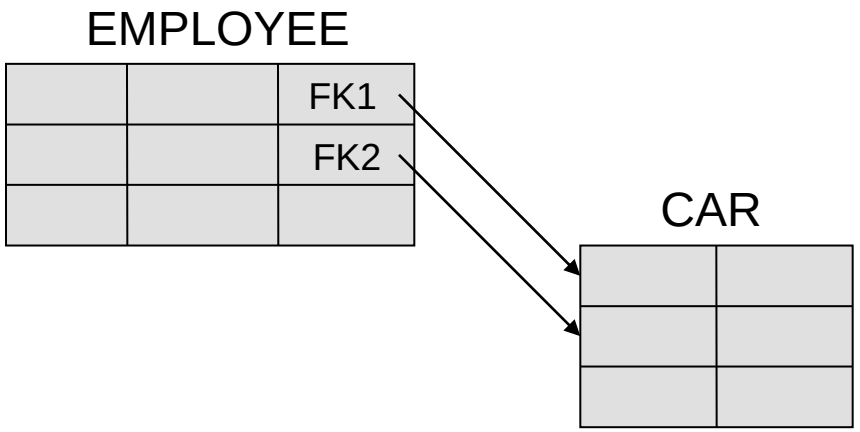
- unidirectional
- bidirectional



Unidirectional one-to-one

```
@Entity
public class Employee {
  @OneToOne
    private Car car;
    public Car getCar() {
        return car;
    }
    public void setCar( Car car ) {
        this.car = car;
    }
    ...
}
```

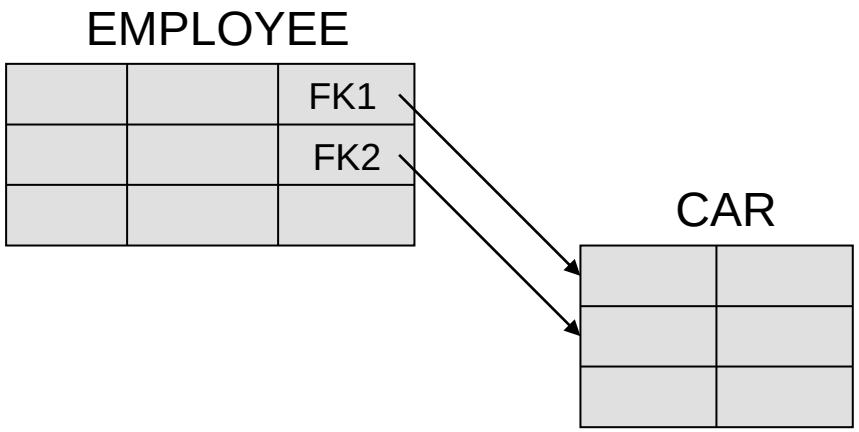
owner



Bidirectional one-to-one

```
@Entity
public class Employee {
    @OneToOne
    private Car car;
    ...
}
@Entity
public class Car {
    @OneToOne( mappedBy="car" )
    private Employee emp;
    ...
}
```

owner



Unidirectional one-to-many

```
@Entity
public class Customer {
    @OneToMany
    private Collection<Account> accounts;
    ...
}
```

owner

JPA 2.0: foreign key

CUSTOMER

C1	
C2	

CUSTOMER_ACCOUNT

C1	A1
C1	A2

unique constraint

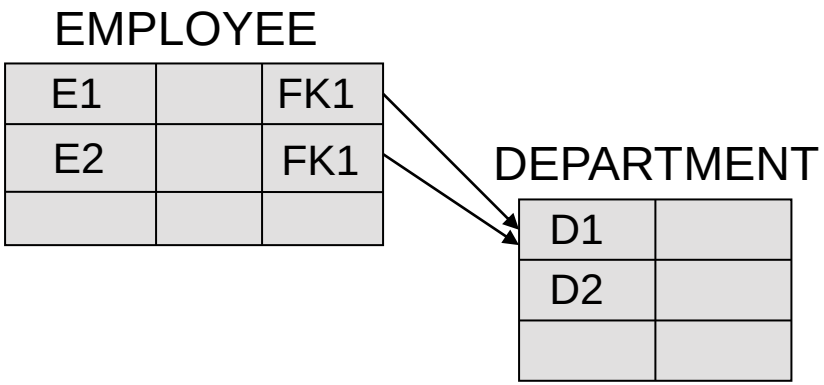
ACCOUNT

A1	
A2	

Bidirectional many-to-one/one-to-many

```
@Entity
public class Employee {
    @ManyToOne
    private Department dep;
    ...
}
@Entity
public class Department {
    @OneToMany( mappedBy="dep" )
    private Collection<Employee> employees;
    ...
}
```

owner



Unidirectional many-to-many

owner

```
@Entity
public class Employee {
    @ManyToMany
    private Collection<Project> projects;
    ...
}
```

EMPLOYEE

E1	
E2	

EMPLOYEE_PROJECT

E1	P1
E1	P2
E2	P2

PROJECT

P1	
P2	

Bidirectional many-to-many

owner

```
@Entity
public class Employee {
    @ManyToMany
    private Collection<Project> ps;
    ...
}
@Entity
public class Project {
    @ManyToMany( mappedBy="ps" )
    private Collection<Employee> es;
    ...
}
```

EMPLOYEE

E1	
E2	

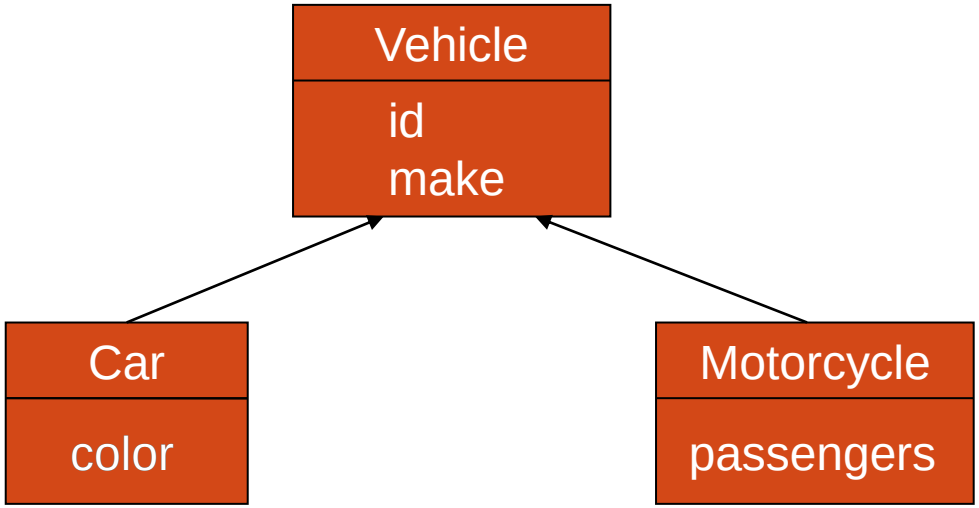
EMPLOYEE_PROJECT

E1	P1
E1	P2
E2	P2

PROJECT

P1	
P2	

Mapování hierarchie tříd



- jedna tabulka (SINGLE_TABLE)
- jedna tabulka pro každou třídu (JOINED)
- jedna tabulka pro každou neabstraktní třídu (TABLE_PER_CLASS)

Single Table per Class Hierarchy

VEHICLE

ID	DTYPE	MAKE	COLOR	PASSENGERS
1	Car	Škoda	černá	NULL
2	Motorcycle	Jawa	NULL	2

```
@Entity
@Inheritance( strategy = InheritanceType.SINGLE_TABLE )
@DiscriminatorColumn( name = "DTYPE" )
public abstract class Vehicle { ... }

@Entity
@DiscriminatorValue( "Car" )
public class Car extends Vehicle { ... }
```

Joined Subclass Strategy

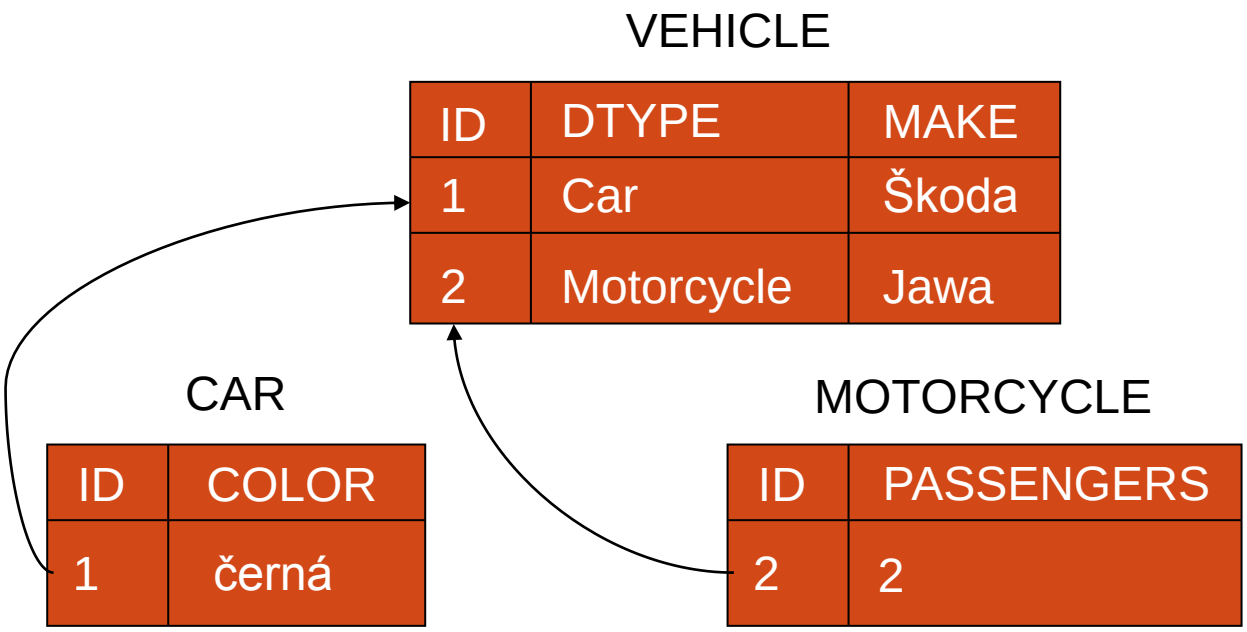


Table per Concrete Class

CAR

ID	MAKE	COLOR
1	Škoda	černá

MOTORCYCLE

ID	MAKE	PASSENGERS
2	Jawa	2

Nalezení entity

```
@Stateless
public class VehicleManagerBean implements VehicleManagerLocal {
    @PersistenceContext
    private EntityManager em;

    public Vehicle findVehicle( Long id ) {
        return em.find( Vehicle.class, id );
    }
    ...
}
```

Polymorphism:
id=1 Car
id=2 Motorcycle

Query

```
@Stateless
public class VehicleManagerBean implements VehicleManagerLocal {
    @PersistenceContext
    private EntityManager em;

    public Collection<Vehicle> findAllVehicles() {
        Query query = em.createQuery(
            "SELECT v FROM Vehicle v" );
        return query.getResultList();
    }
    ...
}
```

Java Persistence Query Language

Polymorfizmus: výsledek může obsahovat instance různých tříd

Otázky & odpovědi

tronicek@fit.cvut.cz