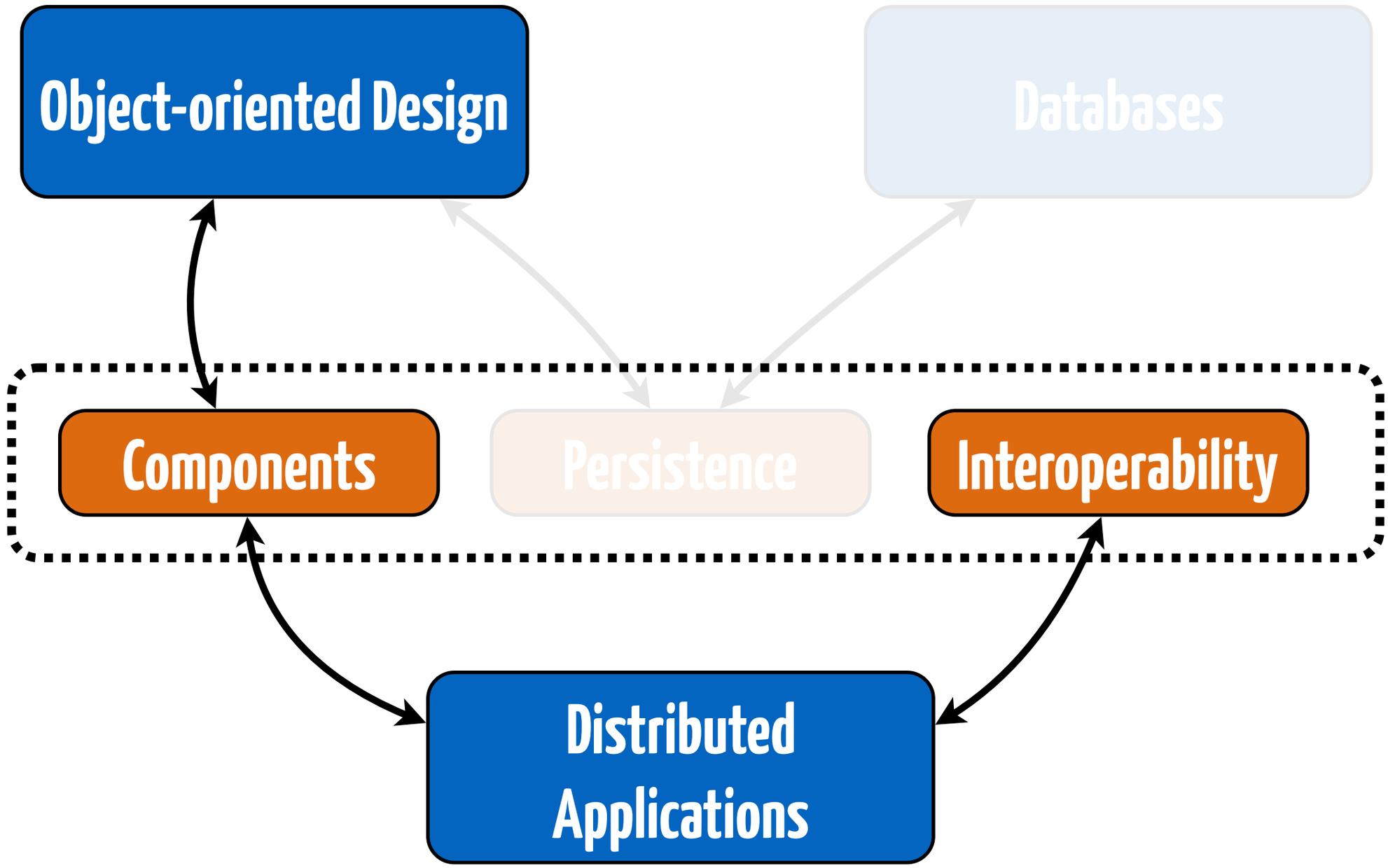
Domain Layer & EJBs: Developing **Session Beans**

Sébastien Mosser

Lecture #2.2, 01.03.2018

pictures: sxc.hu

Applications Server: Dependencies



3-tiers architecture

Presentation

Handle users

Domain

Business functionality

Data Source

Handle data storage

1

Principles

2

State(less|ful) beans

3

Example

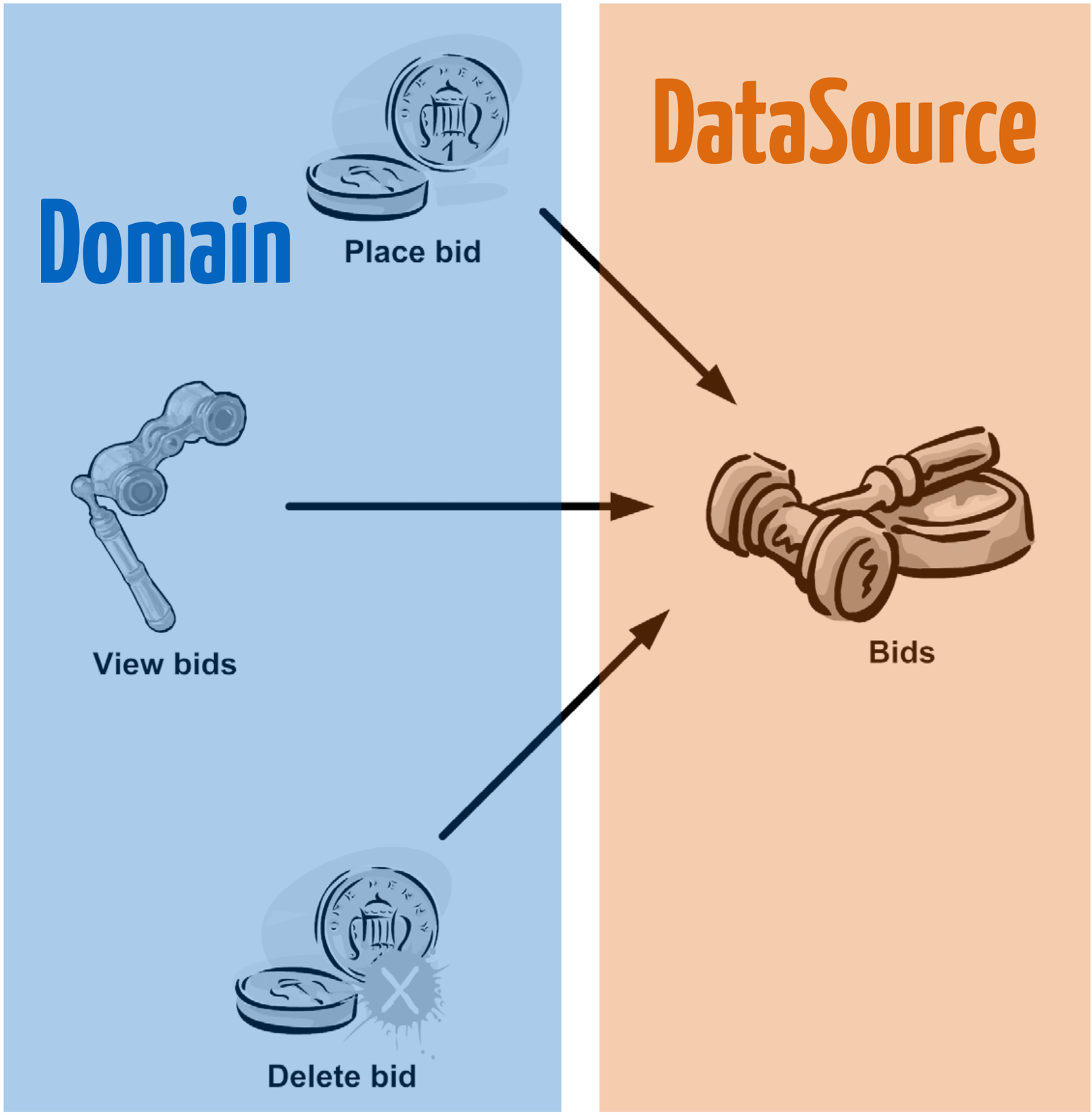
4

Principles

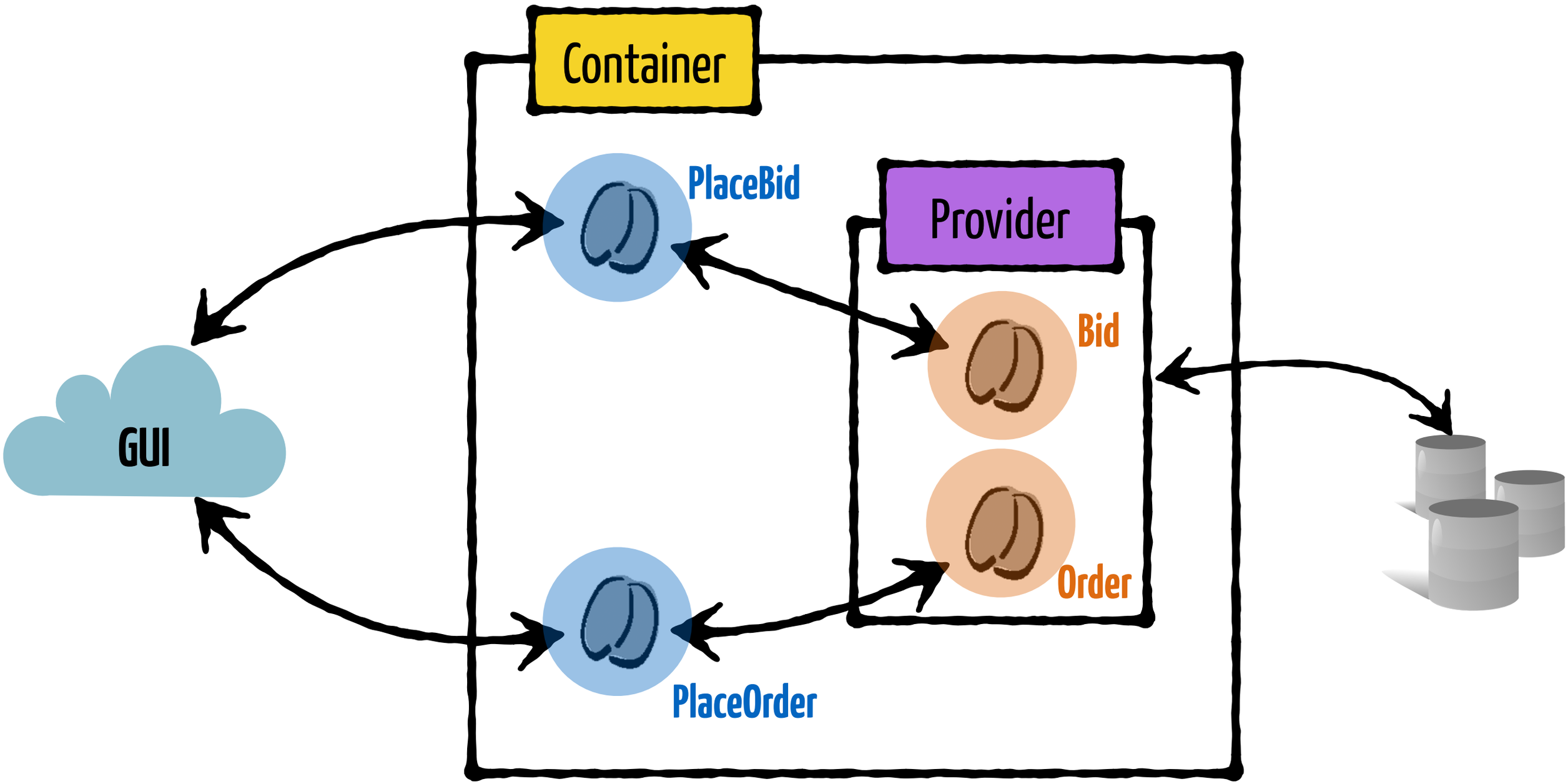
Rule of Thumb

Domain Bean interfaces as **Verbs**

DataSource Beans as **Nouns**


Domain
DataSource
Place bid
View bids
Bids
Delete bid

Client **never calls** a datasource directly

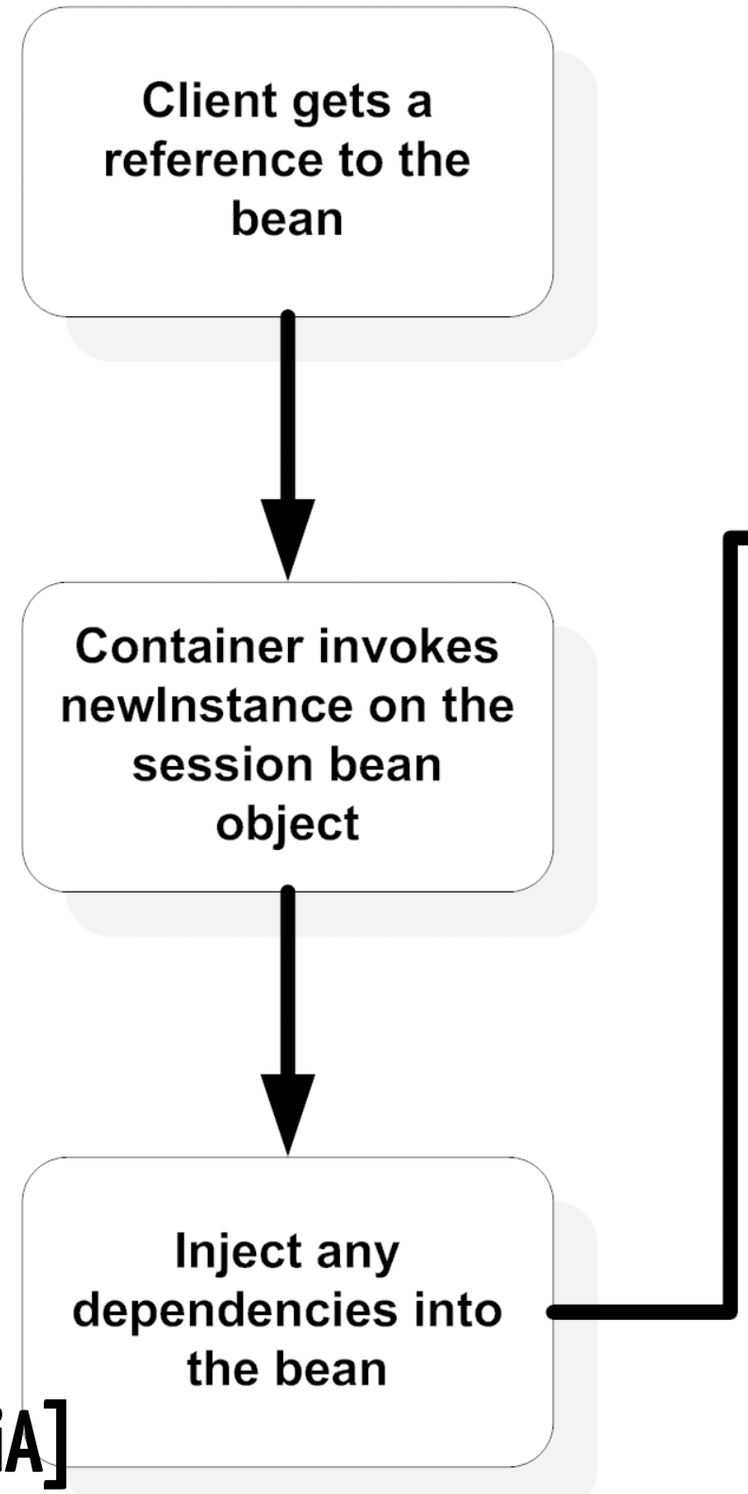


[EiA]

You'll **never** instantiate a domain bean.

NEVER

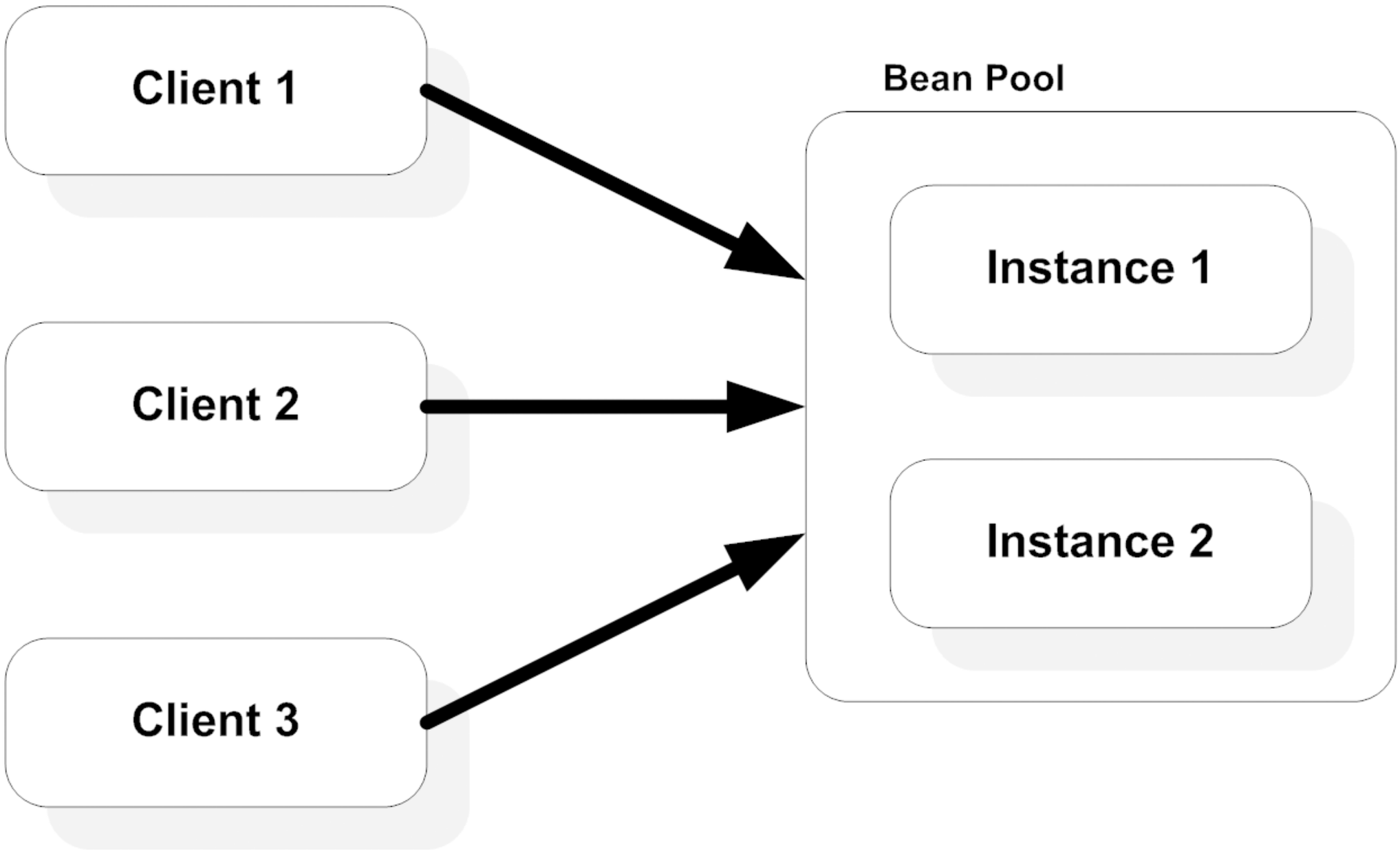
EJB's Lifecycle: **Inversion of Control**



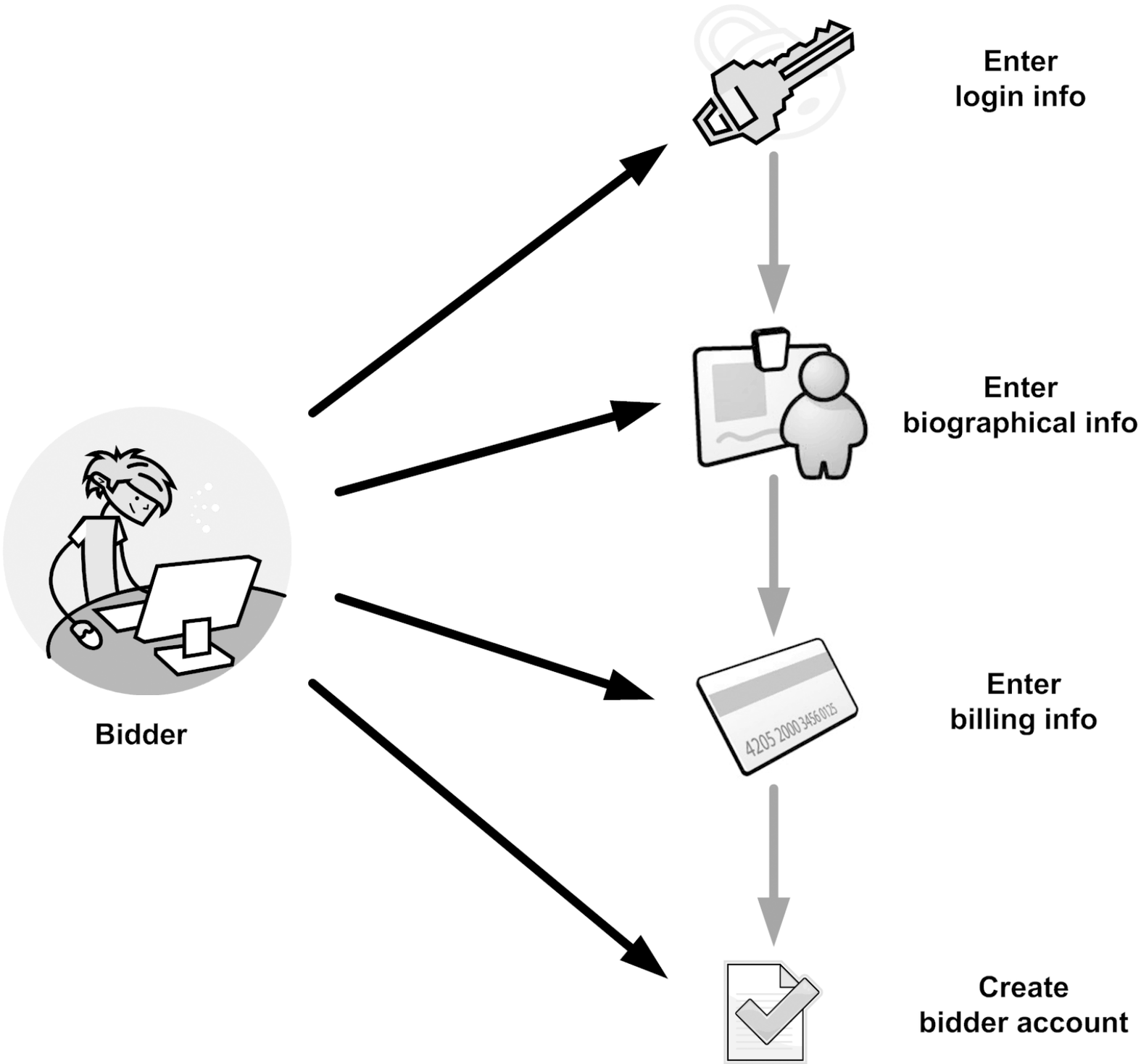
"Don't call us, we'll call you"

[EiA]

EJBs live in a "**pool**"

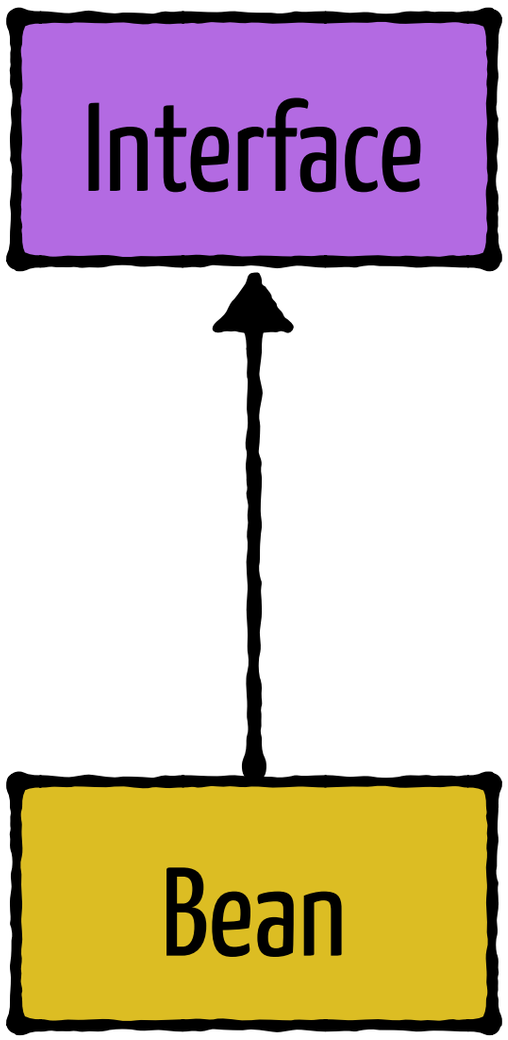


Domain beans live during a **Session**



State(less|ful) beans

Stateless beans : POJO + Annotations

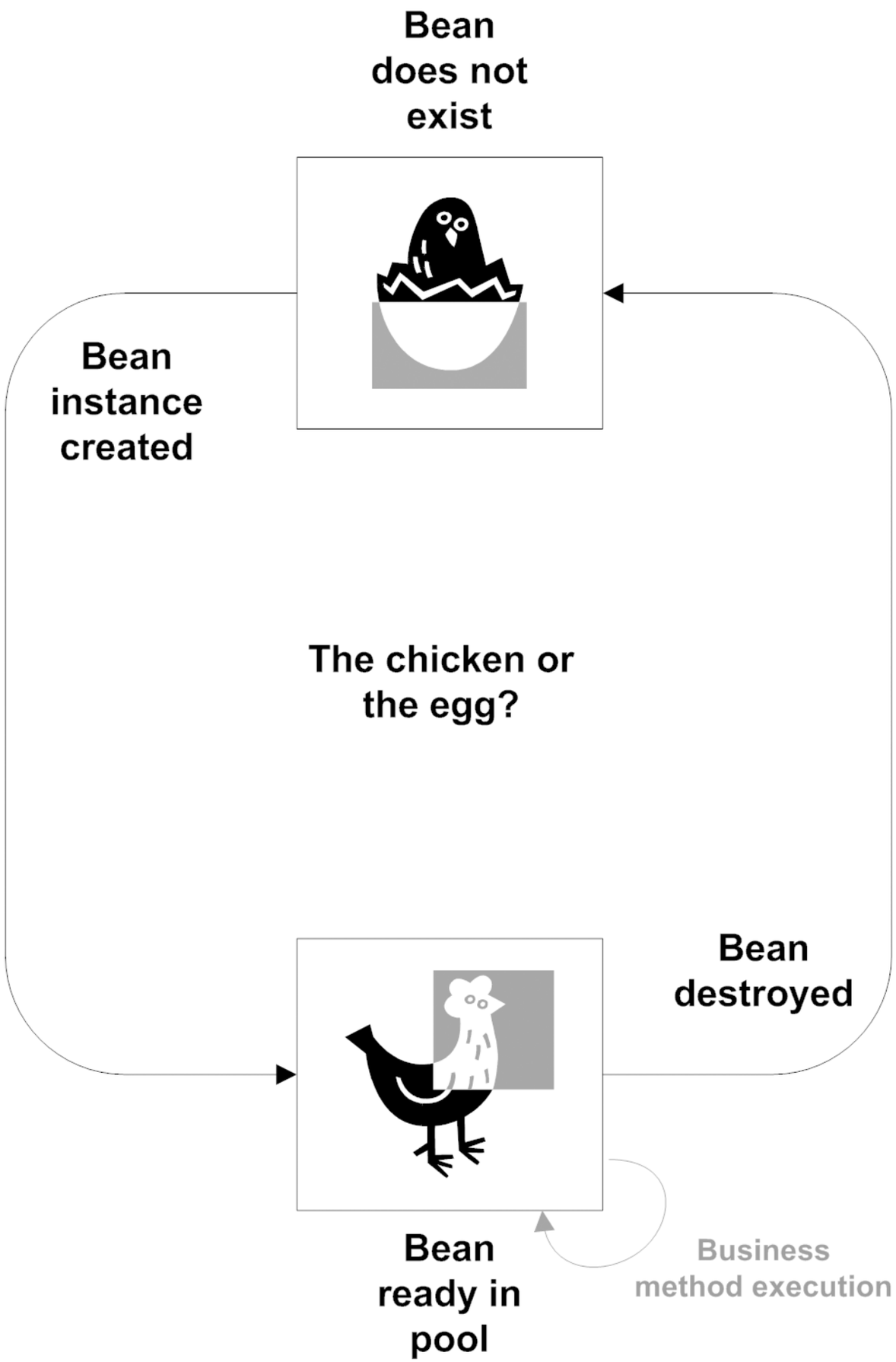



```
public interface PetManager {
    public Pet create(String name);
}
```

```
@Stateless
public class PetManagerBean implements PetManager {

    @PersistenceContext
    EntityManager entityManager;

    @Override
    public Pet create(String name){
        Pet p = new Pet(name);
        entityManager.persist(p);
        return p;
    }

}
```

Lifecycle

Handled by the container



Lifecycle **Hooks**: Construct, Destroy

```
@PostConstruct
public void initialize() {
   System.out.println("Initializing PetManager");
}

@PreDestroy
public void cleanup() {
   System.out.println("Destroying PetManager");
}
```

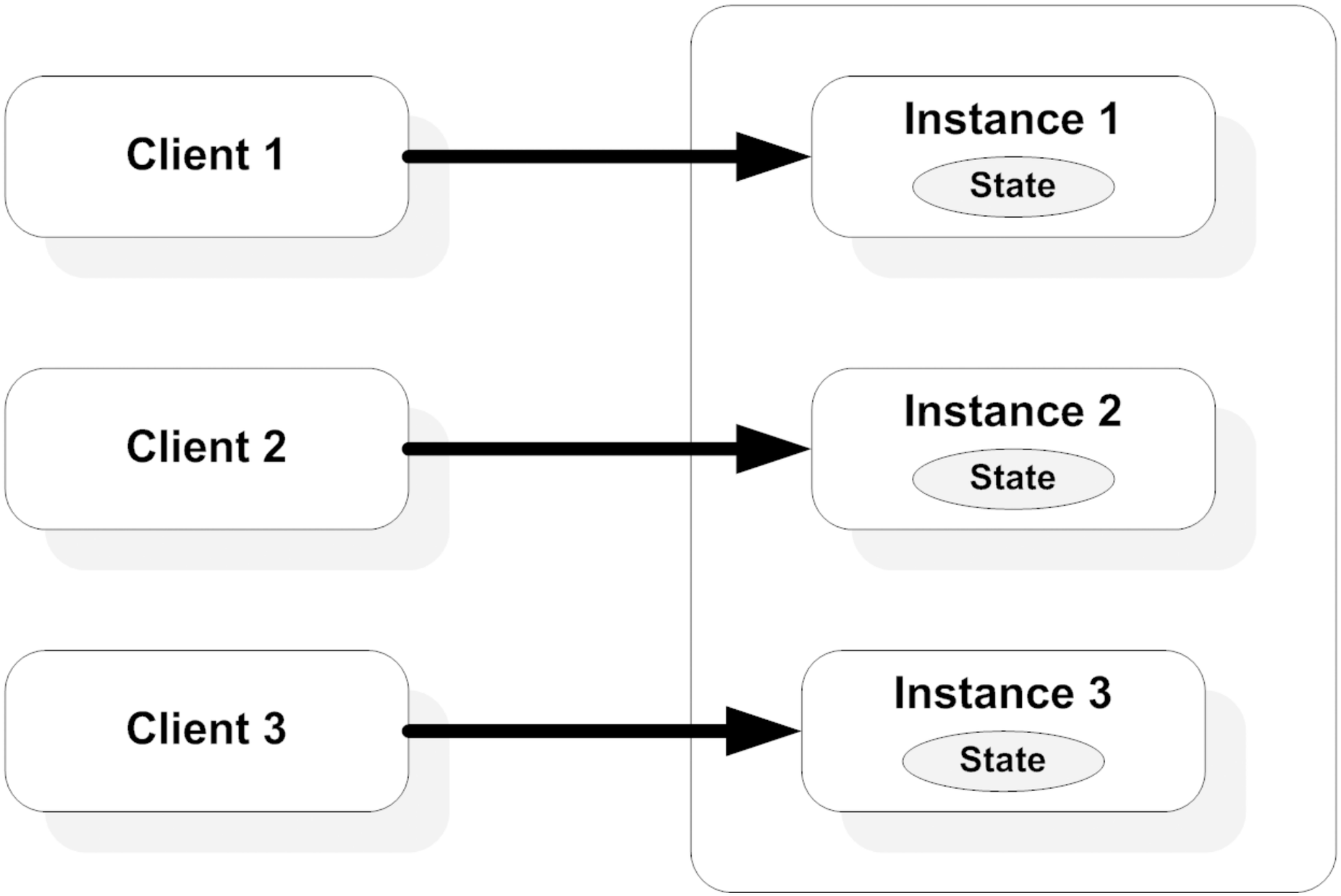
Consuming a Bean: Inversion of Control

```
@EJB
private PetManager manager;
```

Interface

```
@Test
public void testCreation() throws Exception {
   Pet jinx = manager.create("Jinx");
   assertEquals(jinx.name, "Jinx");
}
```

Maintaining **States** during **Sessions**

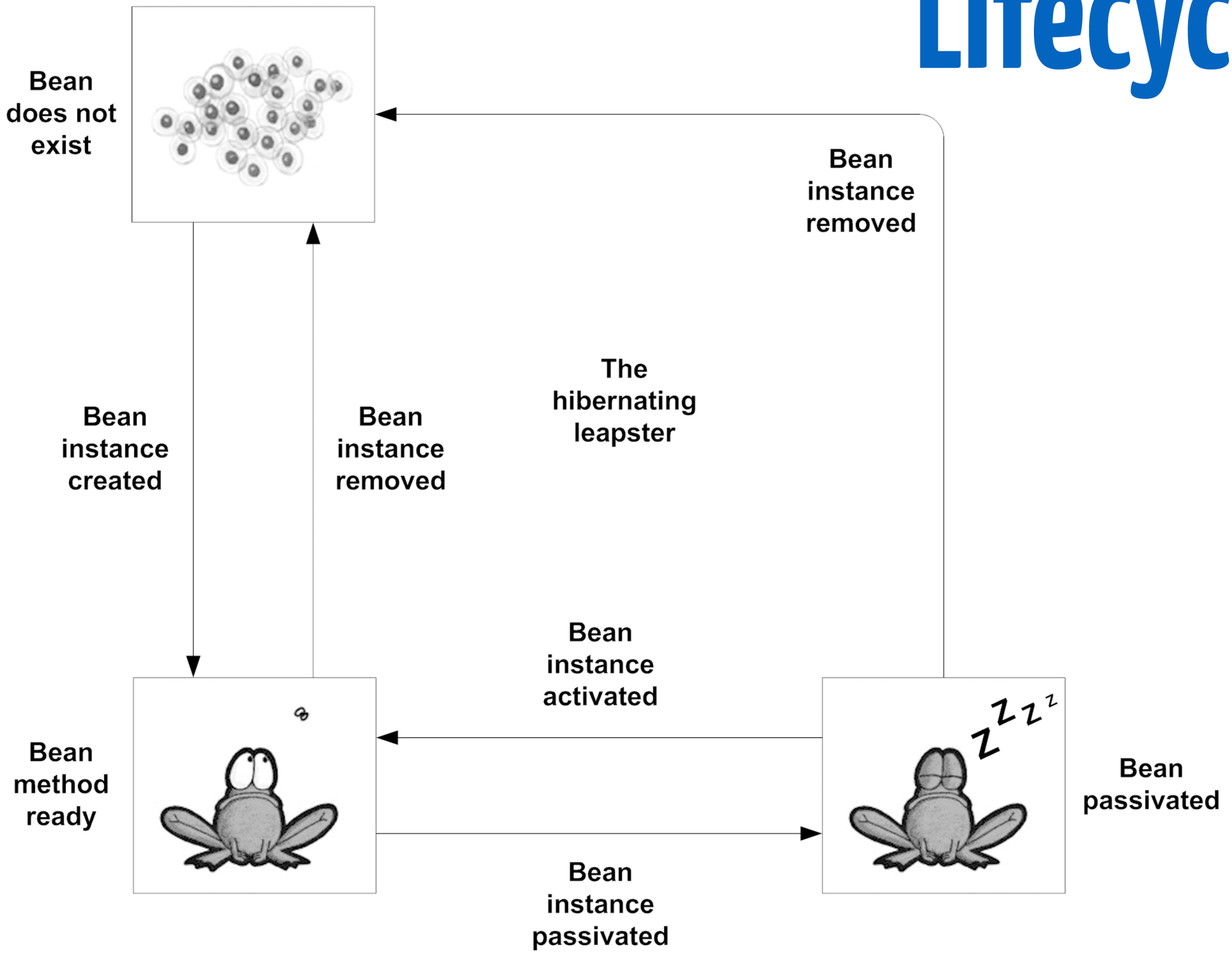


Stateful Bean:Classical Interface

```
public interface PetCart {

    public void addPet(Pet p);

    public List<Pet> getContents();

}
```

```
@Stateful
public class PetCartBean implements PetCart {

   private ArrayList<Pet> _contents =
      new ArrayList<Pet>();

   @Override
   public void addPet(Pet p) {
      _contents.add(p);
   }

   @Override
   public List<Pet> getContents() {
      return _contents;
   }
}
```

Lifecycle



[EiA]

Lifecycle **Hooks**: Stateless + Passivate

@PostConstruct

@PreDestroy

@PrePassivate

@PostActivate

Stateless versus Statefull beans

Features	Stateless	Stateful
Conversational state	No	Yes
Pooling	Yes	No
Performance problems	Unlikely	Possible
Lifecycle events	`PostConstruct`, `PreDestroy`	`PostConstruct`, `PreDestroy`, `PrePassivate`, `PostActivate`
Timer (discussed in chapter 5)	Yes	No
`SessionSynchronization` for transactions (discussed in chapter 6)	No	Yes
Web services	Yes	No
Extended `PersistenceContext` (discussed in chapter 9)	No	Yes

#TeamStateless

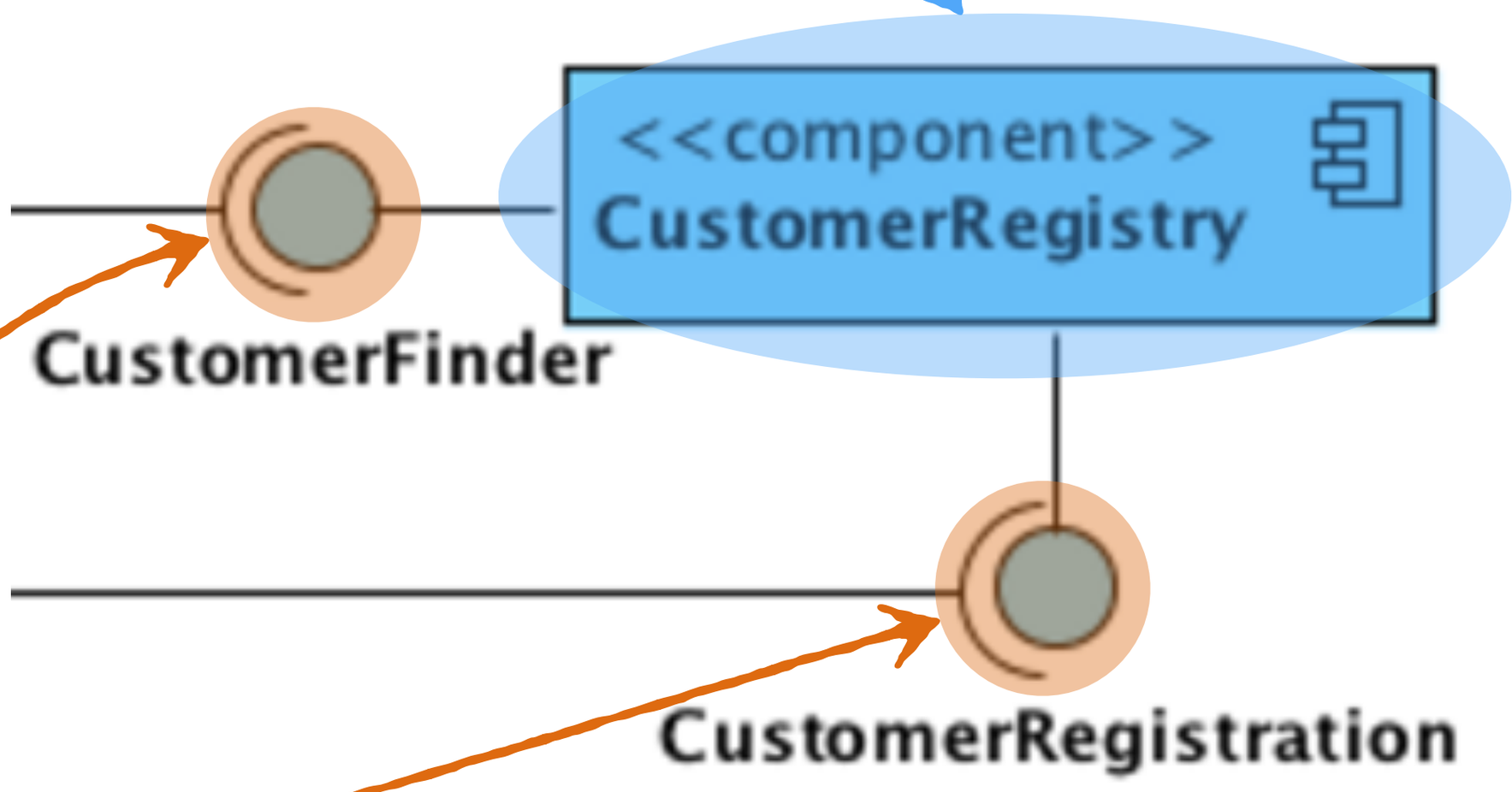
versus

#TeamStateful

Example

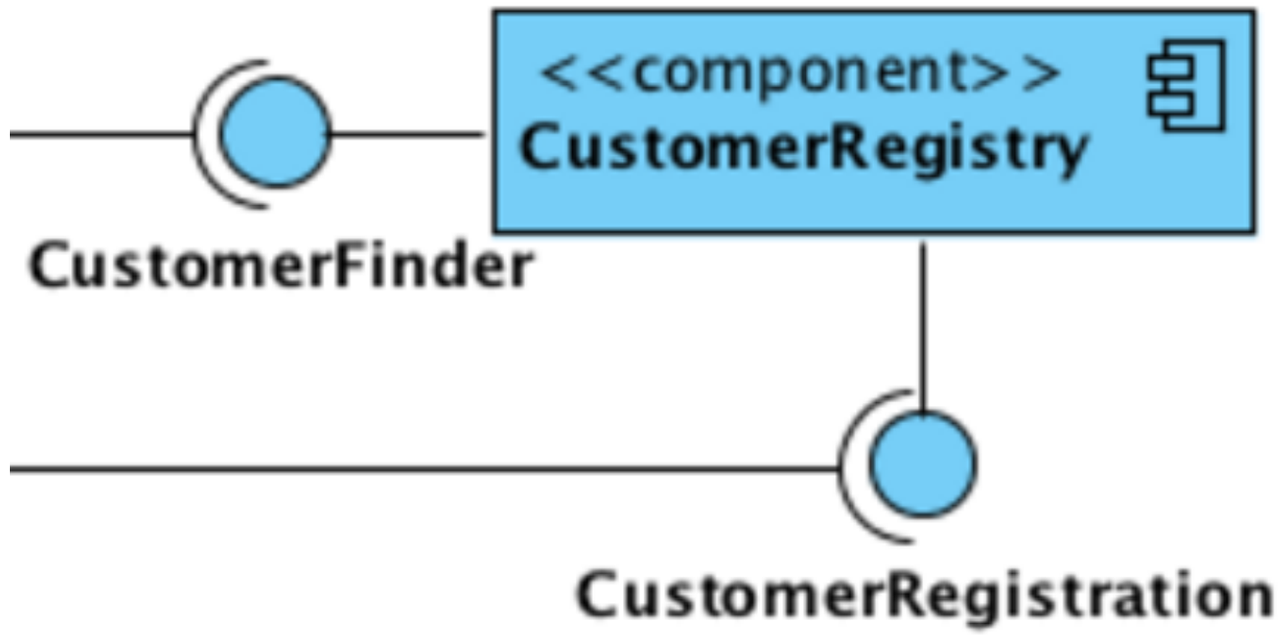
3

Stateless Bean



Interface

```java
@Local
public interface CustomerFinder {

    Optional<Customer> findByName(String name);

}
```




```java
@Local
public interface CustomerRegistration {

    void register(String name, String creditCard)
            throws AlreadyExistingCustomerException;

}
```

Inversion of control

```java
@Stateless
public class CustomerRegistryBean
        implements CustomerRegistration, CustomerFinder {

    @EJB
    private Database memory;

    /**********************************************
     ** Customer Registration implementation **
     **********************************************/

    @Override
    public void register(String name, String creditCard)
            throws AlreadyExistingCustomerException {
        if(findByName(name).isPresent())
            throw new AlreadyExistingCustomerException(name);
        memory.getCustomers().put(name, new Customer(name, creditCard));
    }


    /**********************************************
     ** Customer Finder implementation **
     **********************************************/

    @Override
    public Optional<Customer> findByName(String name) {
        if (memory.getCustomers().containsKey(name))
            return Optional.of(memory.getCustomers().get(name));
        else
            return Optional.empty();
    }

}
```

Persistence mock