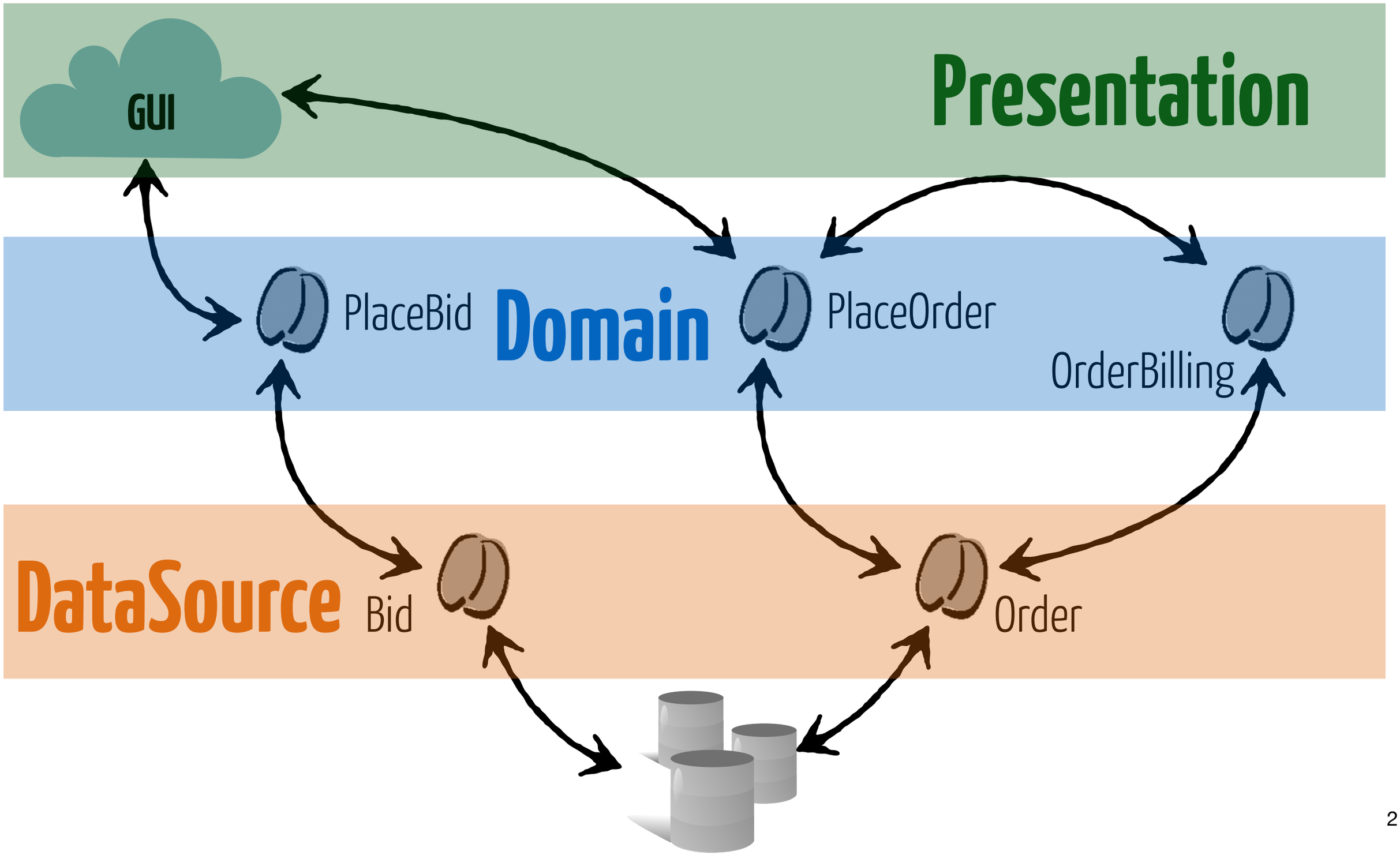
State Wars

Philippe Collet, contains 78,3% of slides from Sébastien Mosser

Lecture #5 05.04.2019

pictures: sxc.hu, pixabay

3-tiers Architecture



Different **Flavors**

Domain

Session Bean

Message-driven Bean

DataSource

Entity Bean

Container

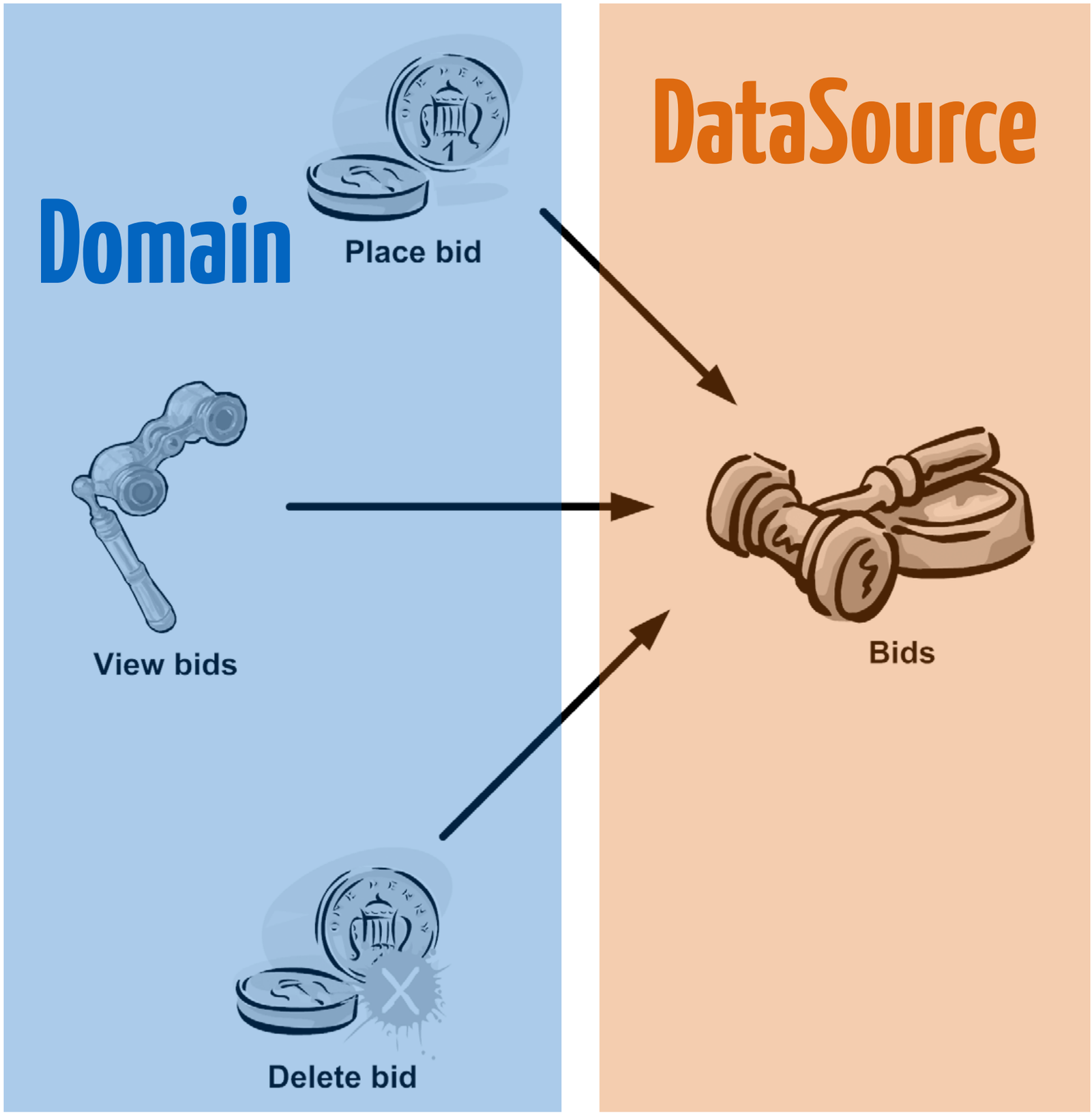
Provider

Principles

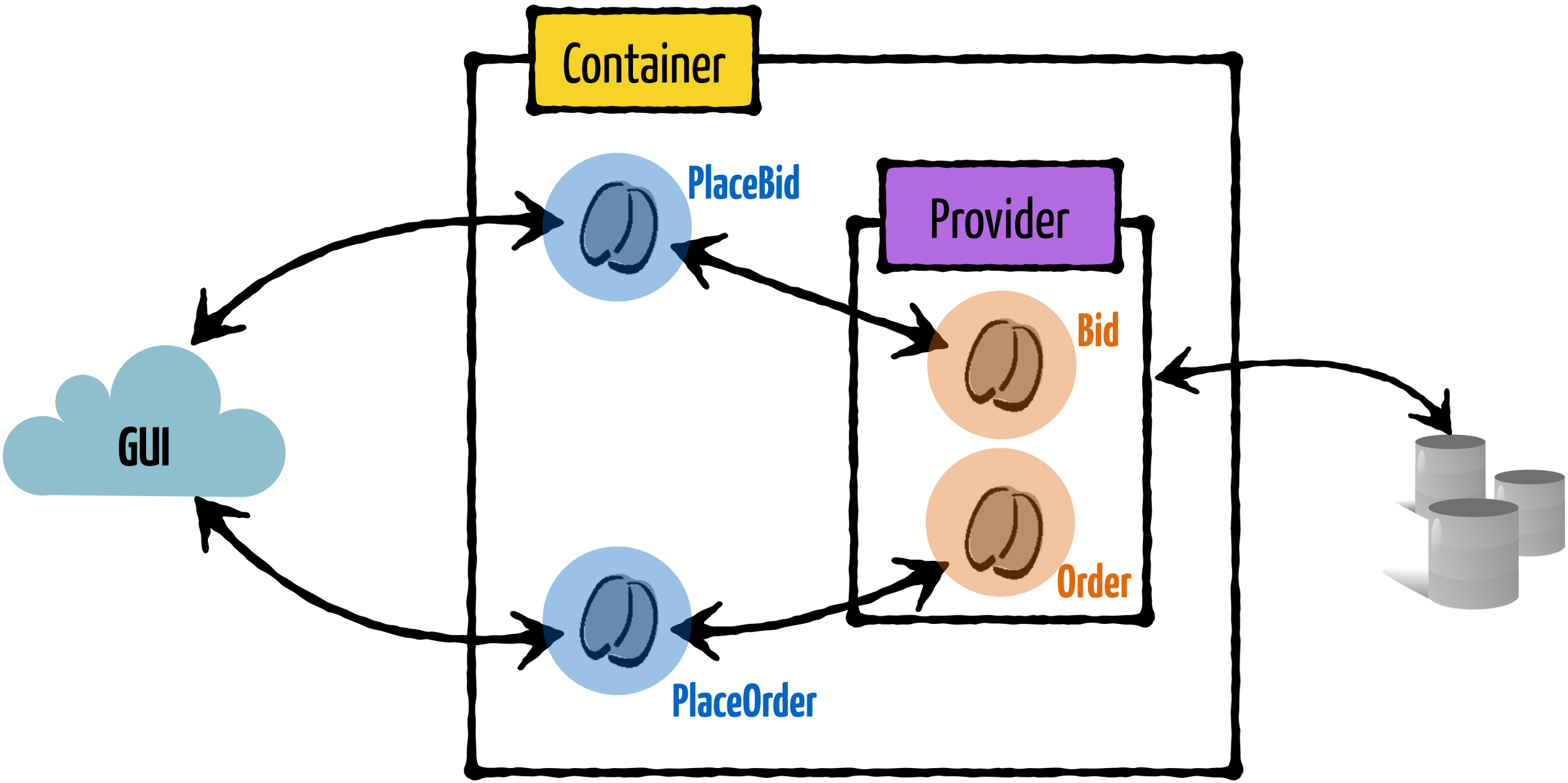
Rule of Thumb

Domain Bean interfaces as **Verbs**

DataSource Beans as **Nouns**


Domain
DataSource
Place bid
View bids
Delete bid
Bids

Client **never calls** a datasource directly

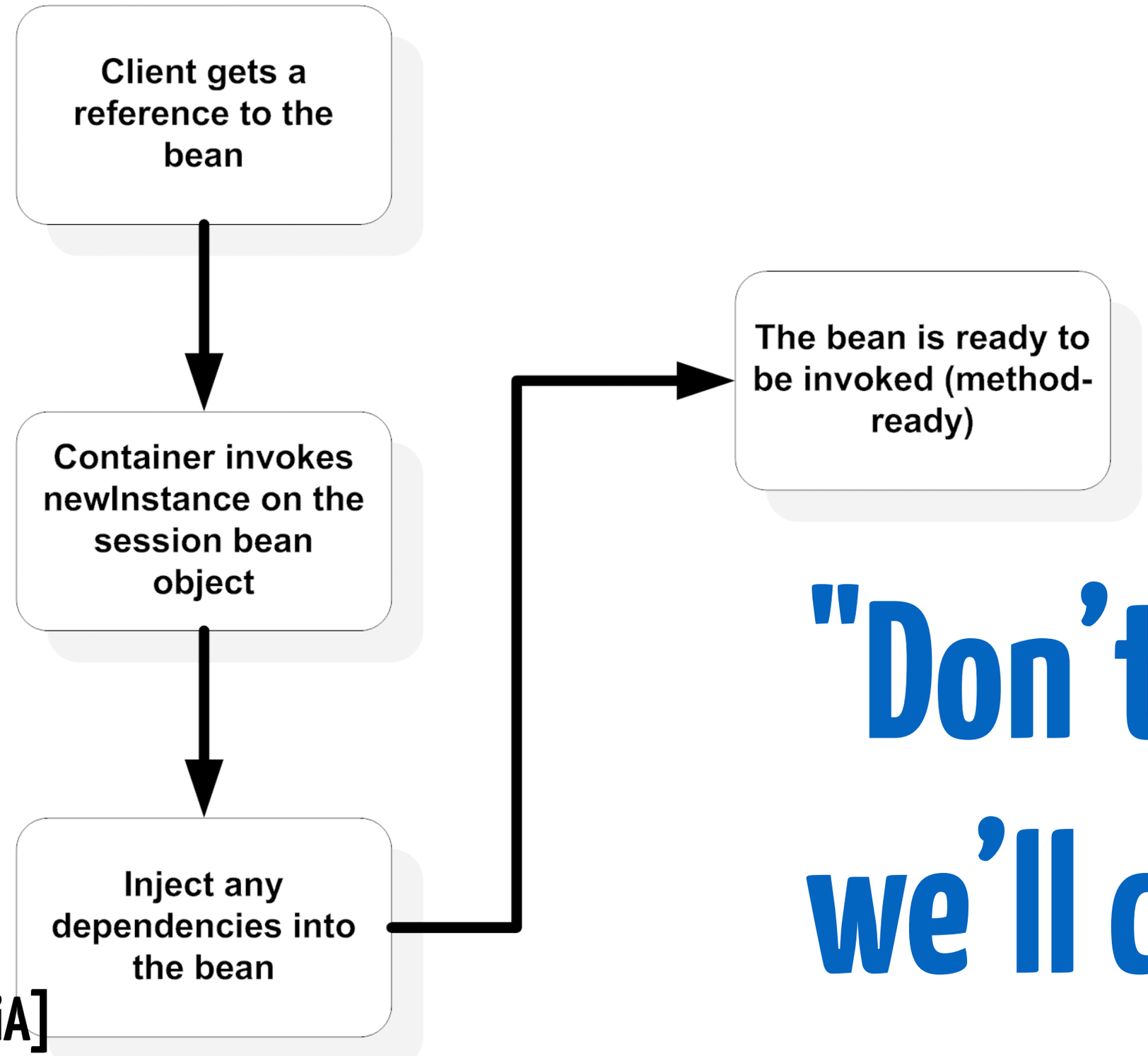


[EiA]

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

NEVER

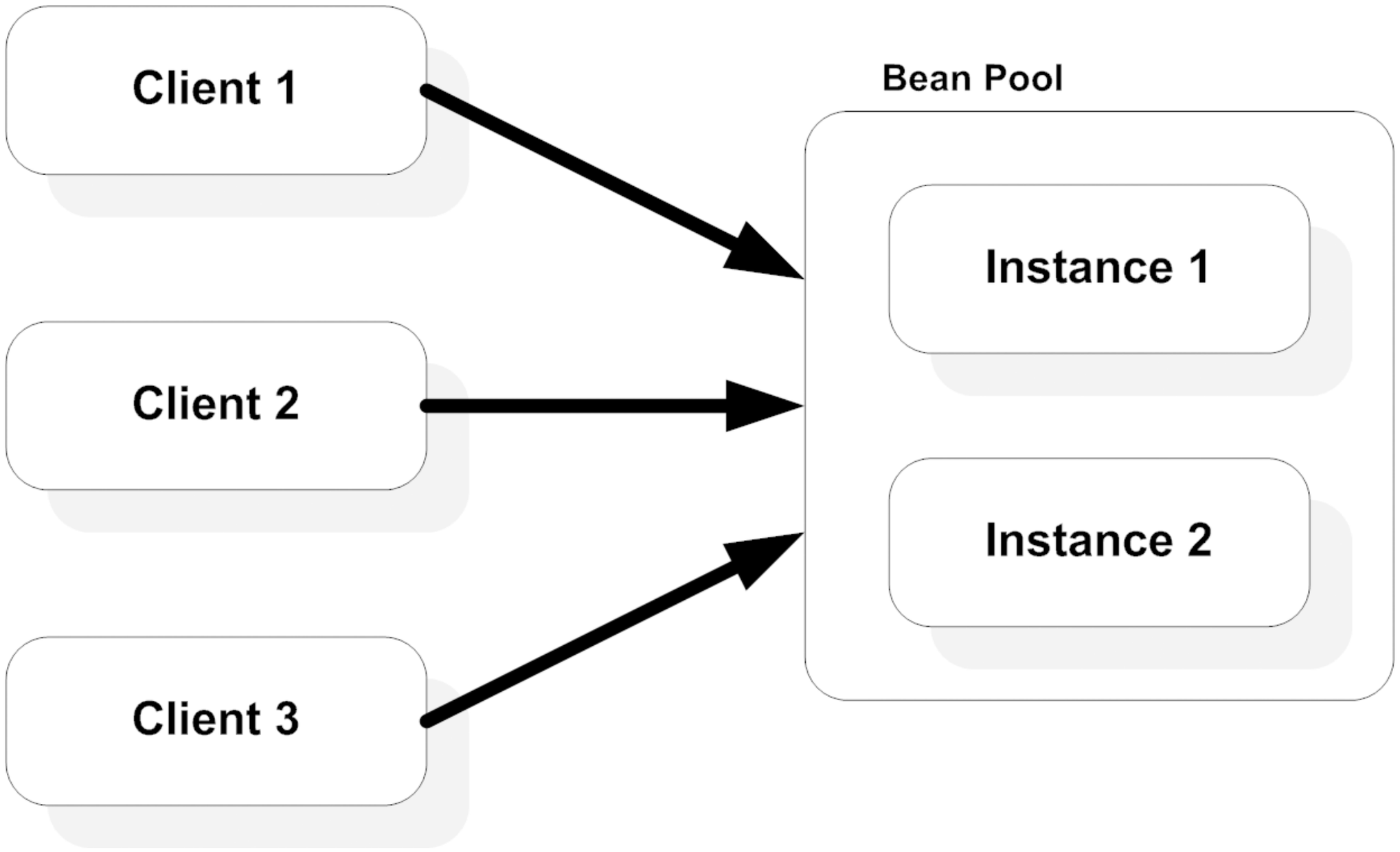
EJB's Lifecycle: **Inversion of Control**



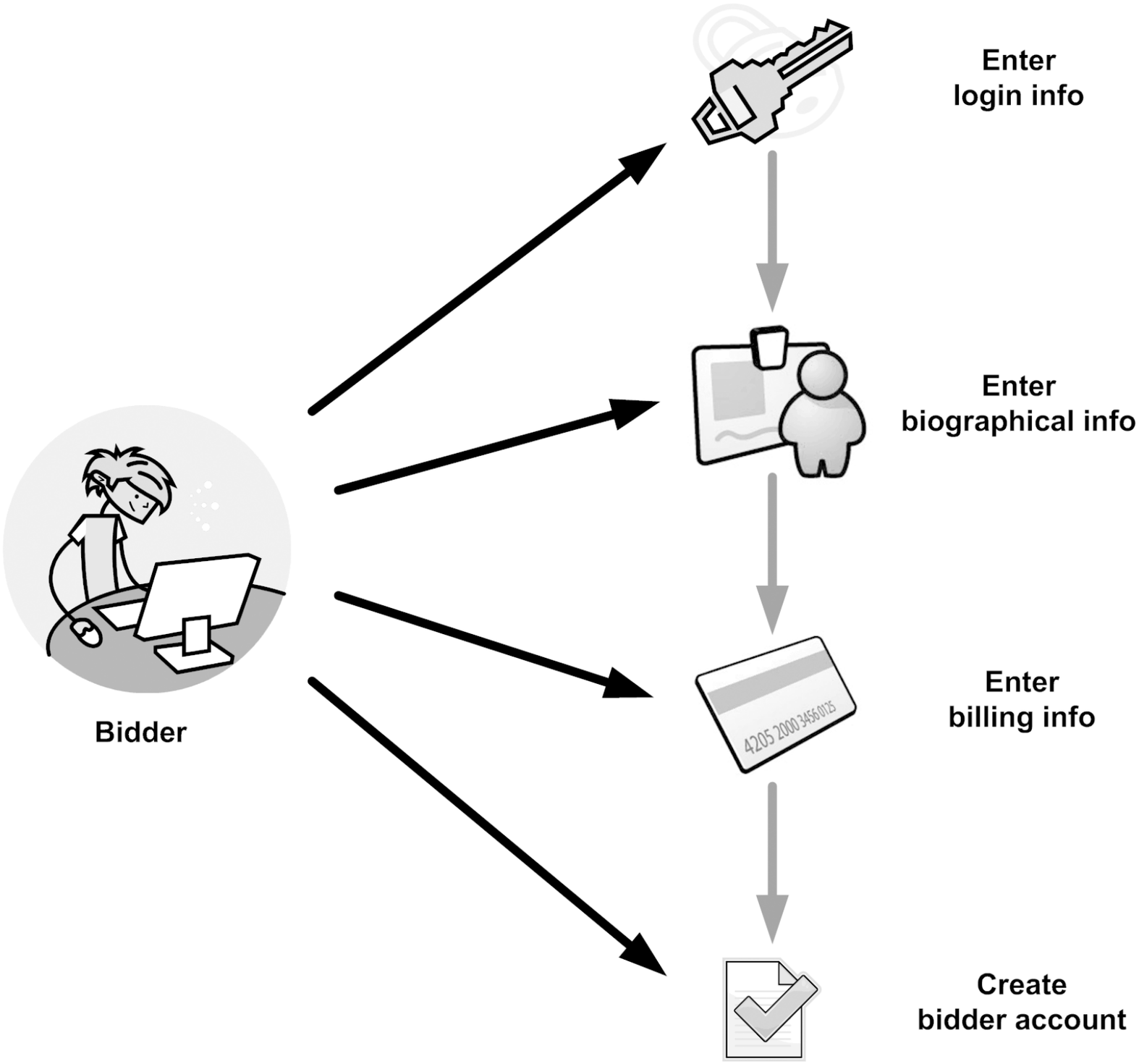
"Don't call us, we'll call you"

[EiA]

EJBs live in a "**pool**"



Domain beans live during a **Session**



In the following, we are talking about **state**

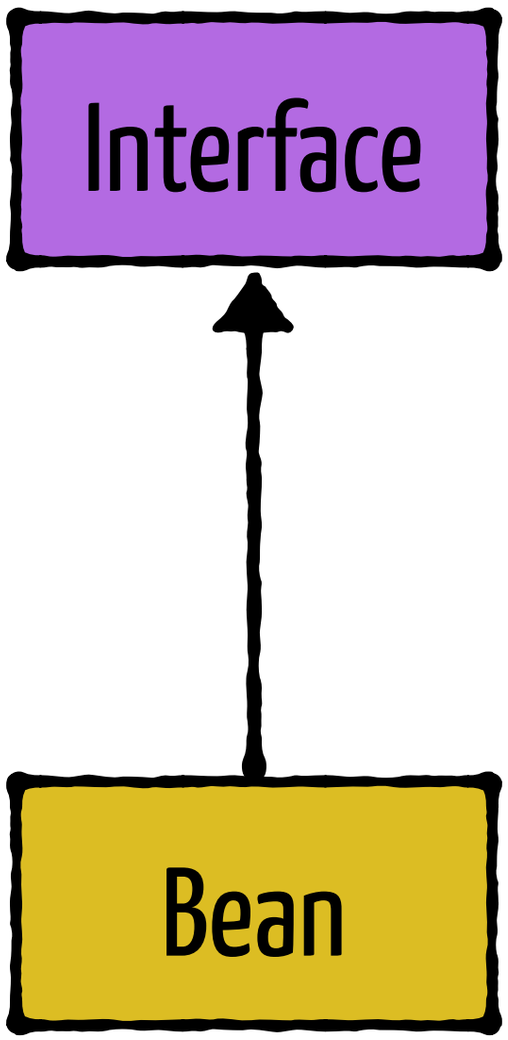
This will refer to the

conversational state

during a session

State(less|ful) beans

Stateless beans : POJO + Annotations

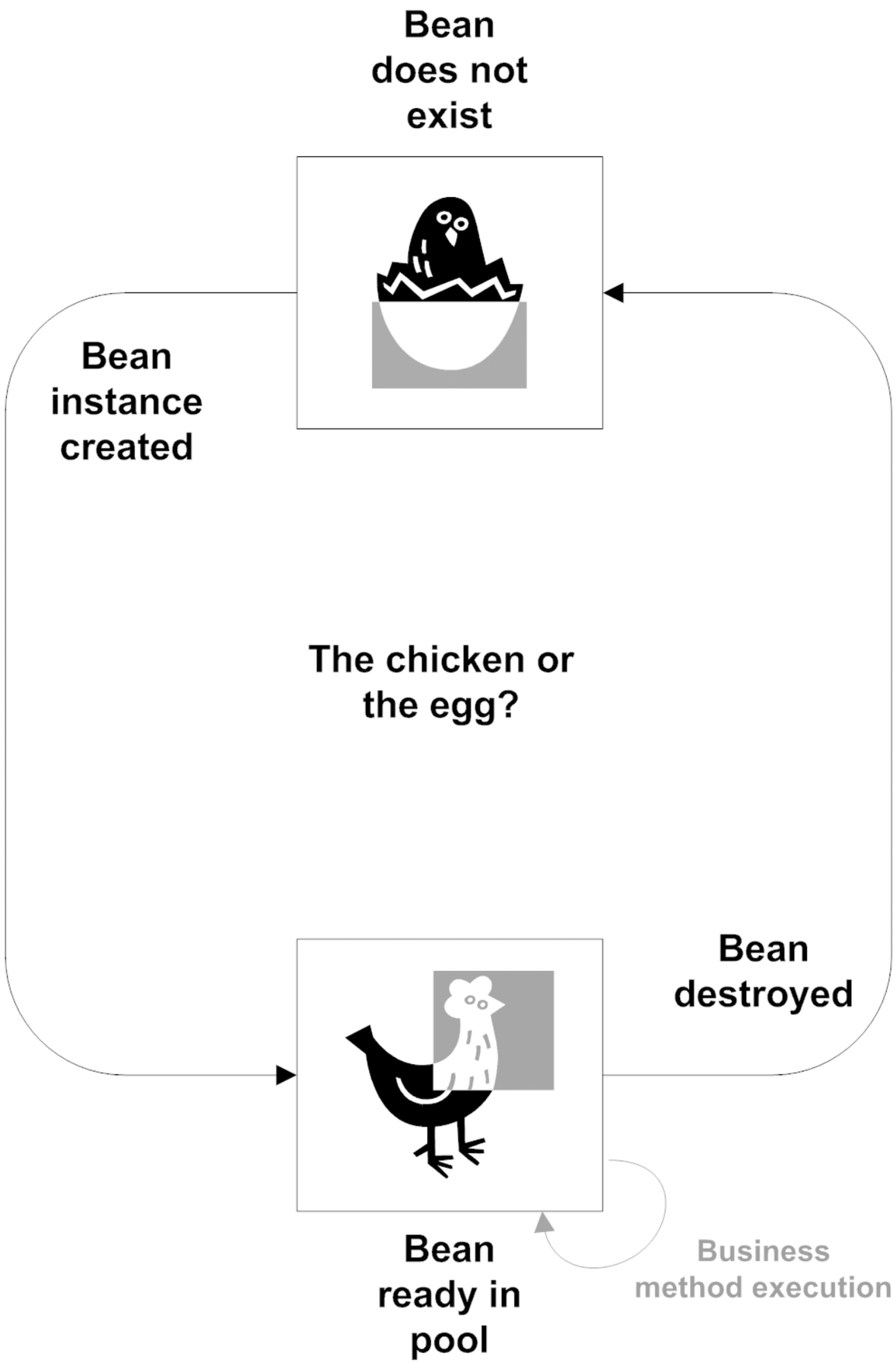



```
public interface PetManager {
    public Pet create(String name);
}
```

```
@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");
}
```

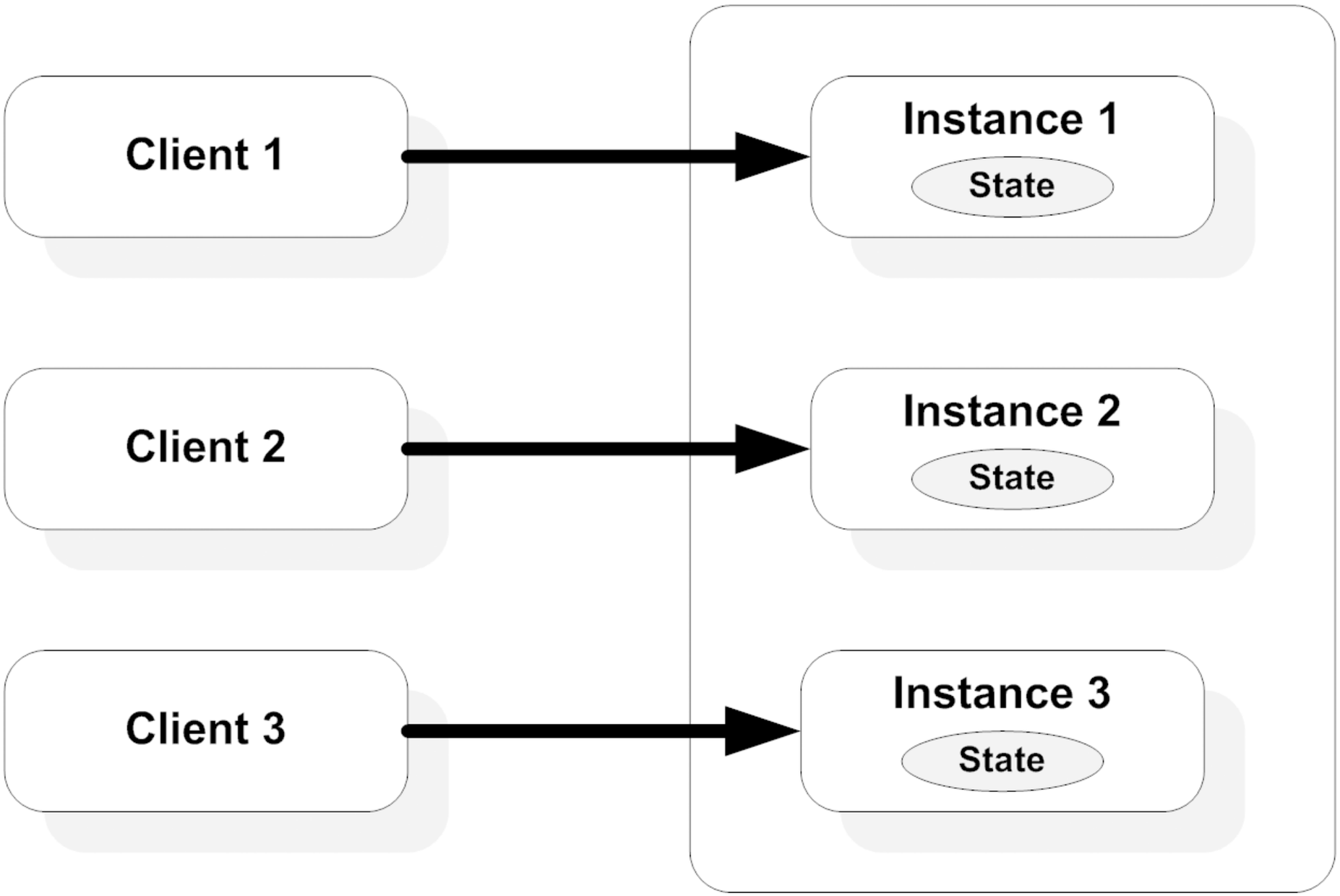
Consuming a Bean: Inversion of Control

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

Interface

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

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

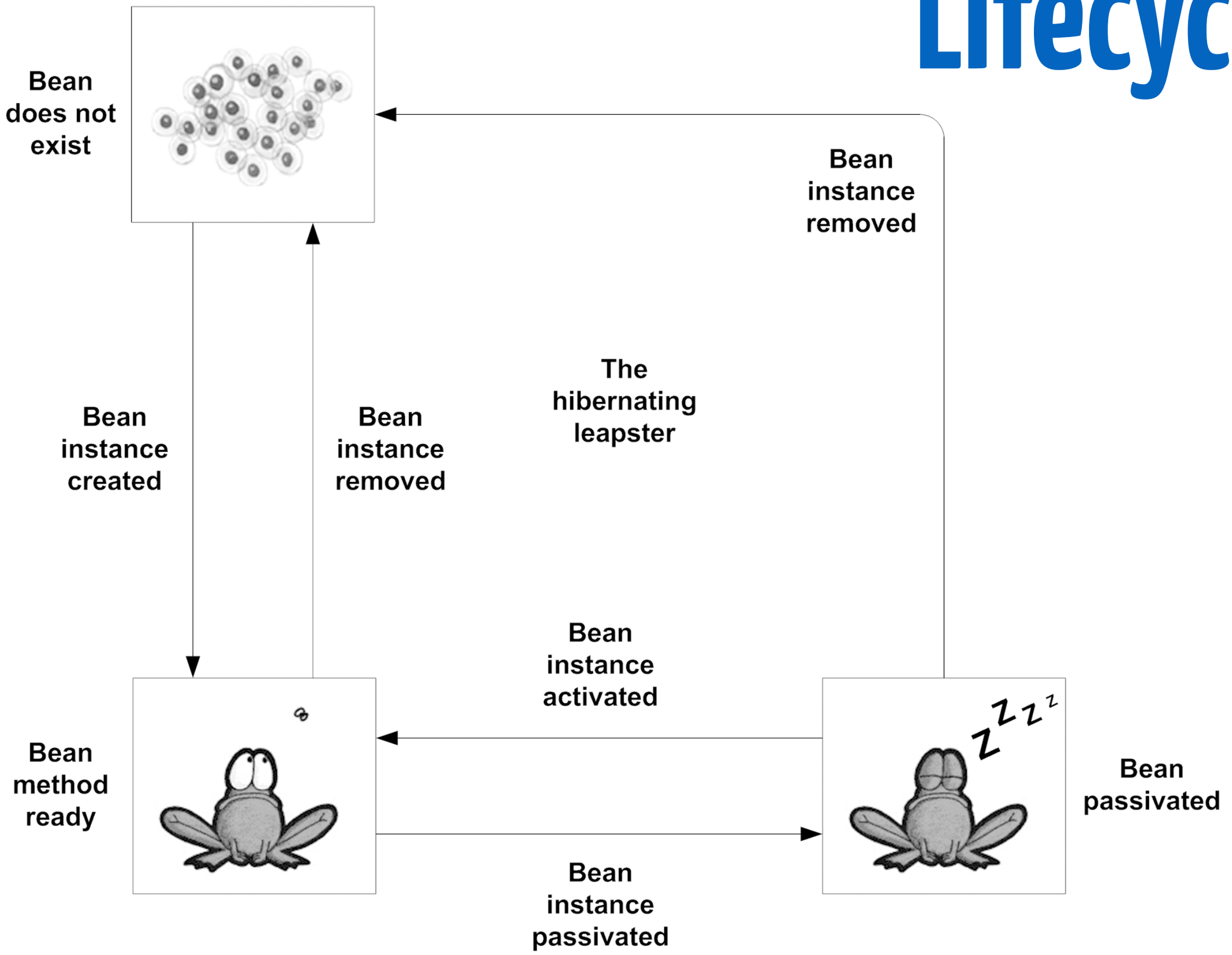


Stateful Bean:Classical Interface

```
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 Stateful 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

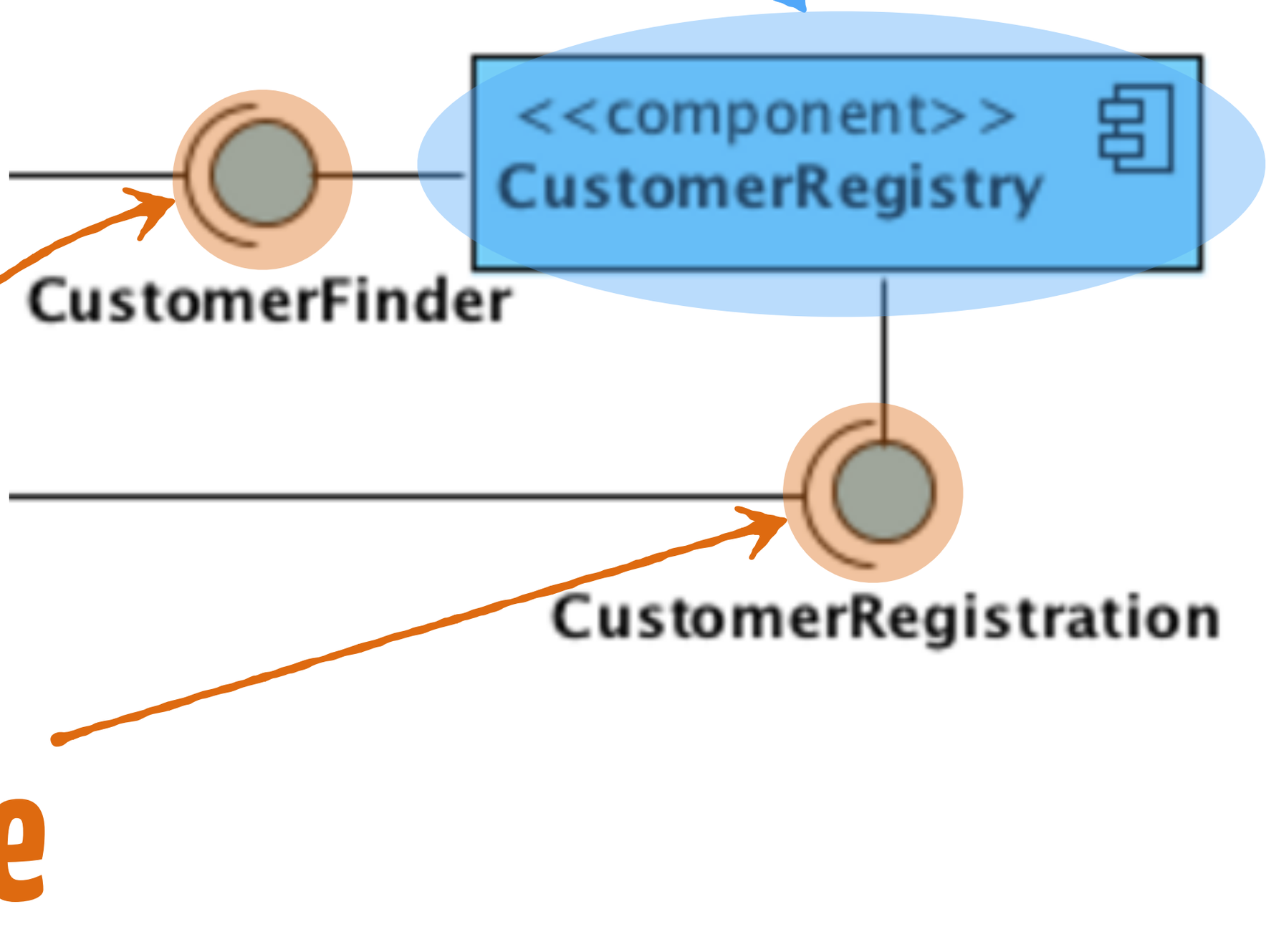
versus

#TeamStateful

Examples

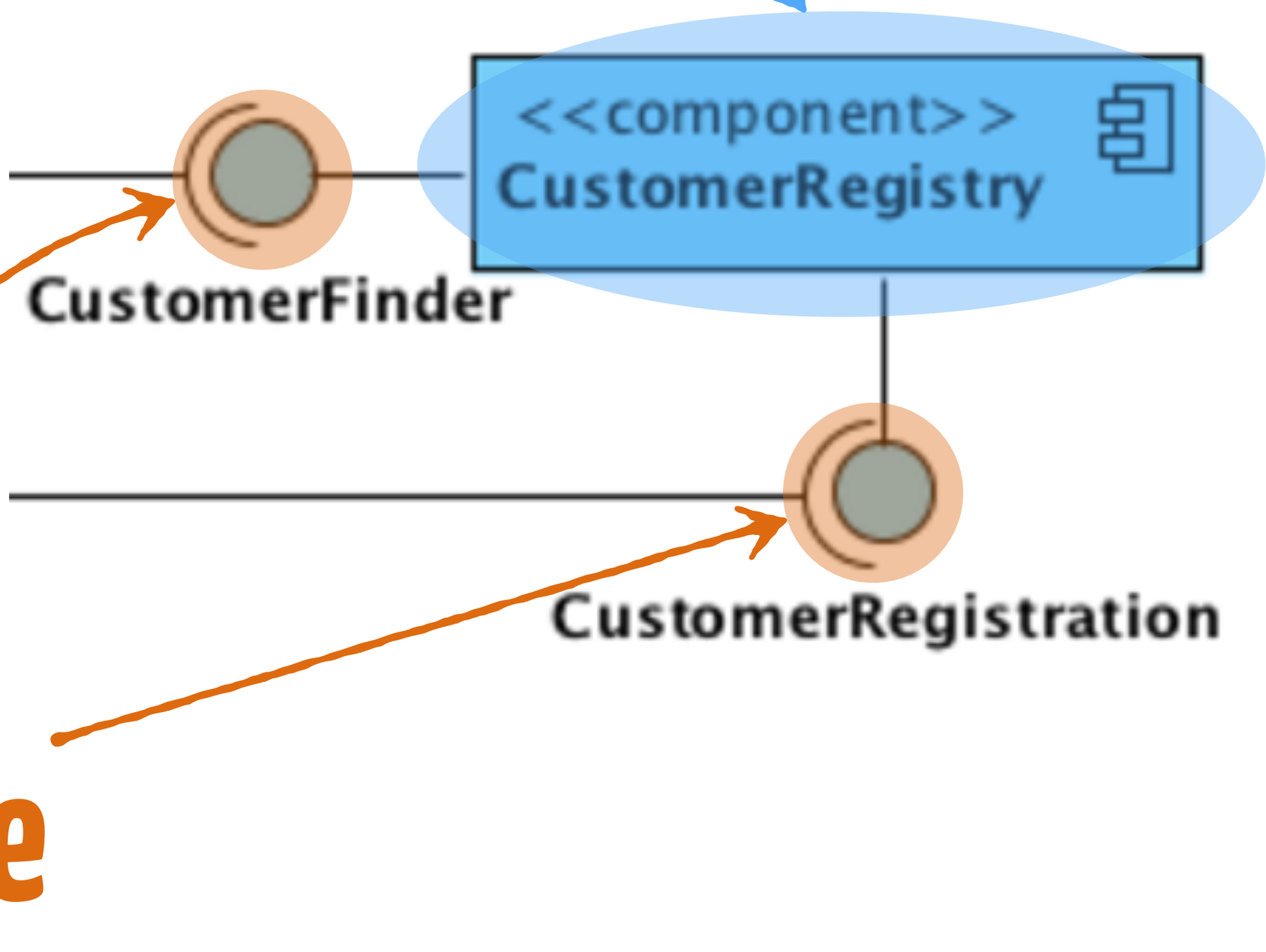
3

Stateless Bean



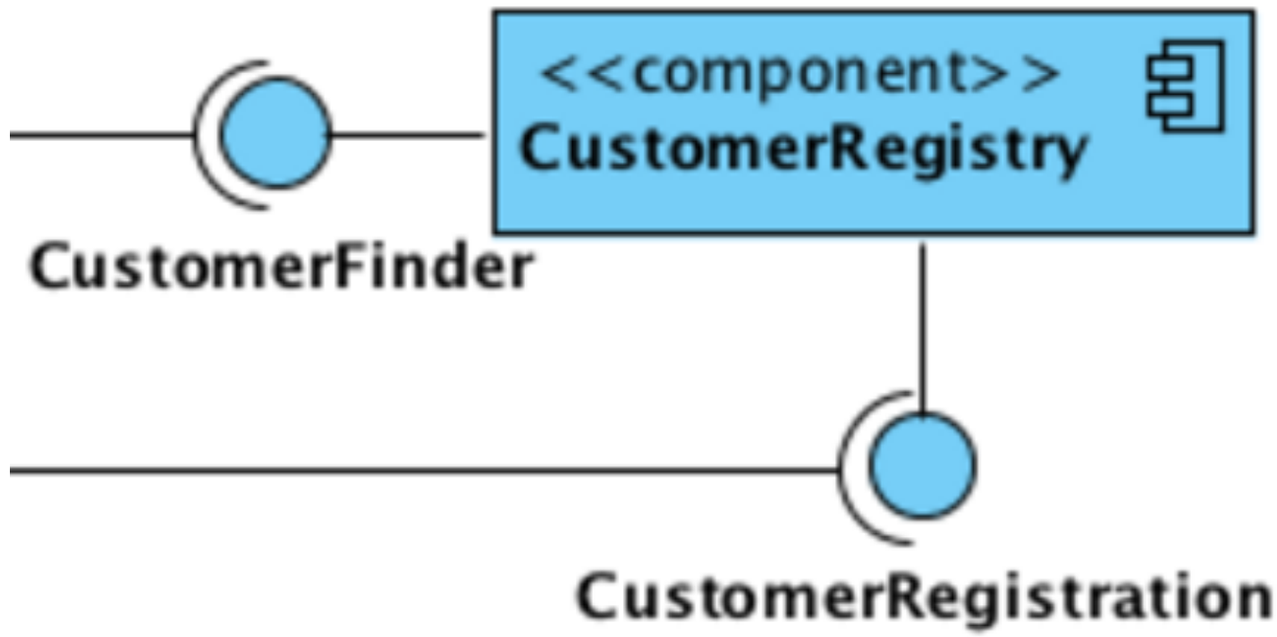
Interface

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

```java
@Entity
public class Customer implements Serializable {

        @Id
        @GeneratedValue(strategy = GenerationType.AUTO)
        private int id;

        @NotNull
        private String name;

    @NotNull
    @Pattern(regexp = "\\d{10}+", message = "Invalid creditCardNumber")
    private String creditCard;

    @OneToMany(cascade = {CascadeType.REMOVE, CascadeType.MERGE}, fetch = FetchType.LAZY, mappedBy = "customer")
    private Set<Order> orders = new HashSet<>();

    @ElementCollection
    private Set<Item> cart = new HashSet<>();

    public Customer() {
        // Necessary for JPA instantiation process
}
```

```java
@Embeddable
public class Item implements Serializable {

        @Enumerated(EnumType.STRING)
        @NotNull
        private Cookies cookie;

        @NotNull
        private int quantity;
```

```java
@Local
public interface CartProcessor {

    Set<Item> contents(Customer c);

    double price(Customer c);

    String validate(Customer c) throws PaymentException, EmptyCartException;
```

```java
@Local
public interface CartModifier {

        boolean add(Customer c, Item item);

        boolean remove(Customer c, Item item);

}
```

```java
public abstract class CartBean implements CartModifier, CartProcessor {

    @EJB
    protected Payment cashier;

    @Override
    @Interceptors({CartCounter.class})
    public String validate(Customer c) throws PaymentException, EmptyCartException {
        if (contents(c).isEmpty())
            throw new EmptyCartException(c.getName());
        String id = cashier.payOrder(c, contents(c));
        contents(c).clear();
        return id;
    }
```

CartBean (cont'd)

```
/**
 * Protected method to update the cart of a given customer, shared by both stateful and stateless beans
 */
protected Set<Item> updateCart(Customer c, Item item) {
    Set<Item> items = contents(c);
    Optional<Item> existing = items.stream().filter(e -> e.getCookie().equals(item.getCookie())).findFirst();
    if (existing.isPresent()) {
        items.remove(existing.get());
        Item toAdd = new Item(item.getCookie(), item.getQuantity() + existing.get().getQuantity());
        if (toAdd.getQuantity() > 0) {
            items.add(toAdd);
        }
    } else {
        items.add(item);
    }
    return items;
}
```

```java
@Stateful(name = "cart-stateful")
public class CartStatefulBean extends CartBean {

        private Map<Customer, Set<Item>> carts = new HashMap<>();

        @Override
        public boolean add(Customer c, Item item) {
                carts.put(c, updateCart(c, item));
                return true;
        }

        @Override
        public Set<Item> contents(Customer c) {
                return carts.getOrDefault(c, new HashSet<Item>());
        }
```

```java
@Stateless(name = "cart-stateless")
public class CartStatelessBean extends CartBean {

        @PersistenceContext private EntityManager entityManager;

        @Override
        public boolean add(Customer customer, Item item) {
                Customer c = entityManager.merge(customer);
                c.setCart(updateCart(c, item));
                return true;
        }

        @Override
        public Set<Item> contents(Customer customer) {
                Customer c = entityManager.merge(customer);
                return c.getCart();
        }

}
```