# EJBs

# EJBs

## O que são?!

# EJBs

Componentes corporativos que executam no lado servidor,
Podendo ser disparados local ou remotamente com todo seu ciclo de vida gerenciado por um container EJB e que implementam regras de negócio, colaborando entre si para entregar valor para o usuário final.

# EJBs

## Porque usá-los?

- Fácil implementação (POJOs)
- Catálogo de serviços operacionais
- Foco no negócio

# EJBs

## Quando usar?

Quando a aplicação demanda...

- Escalabilidade
- Segurança
- Controle transacional distribuído e consistente
- E todos os serviços que um container geralmente provê.

# EJBs

## Tipos de Beans

# EJBs

## Session Beans

**Executam uma ação específica para o cliente.**

# EJBs

## Message Driven Beans (MDBs)

**Permitem o processamento assíncrono de mensagens pela aplicação.**

# Session Beans

# Session Beans

**Encapsulam os procedimentos e regras de negócio que são executados durante uma sessão**

# Session Beans

**Podem guardar, se necessário, o estado conversacional entre cliente e servidor**

# Session Beans

Gerenciados pelo container

# Session Beans

Não são objetos persistentes

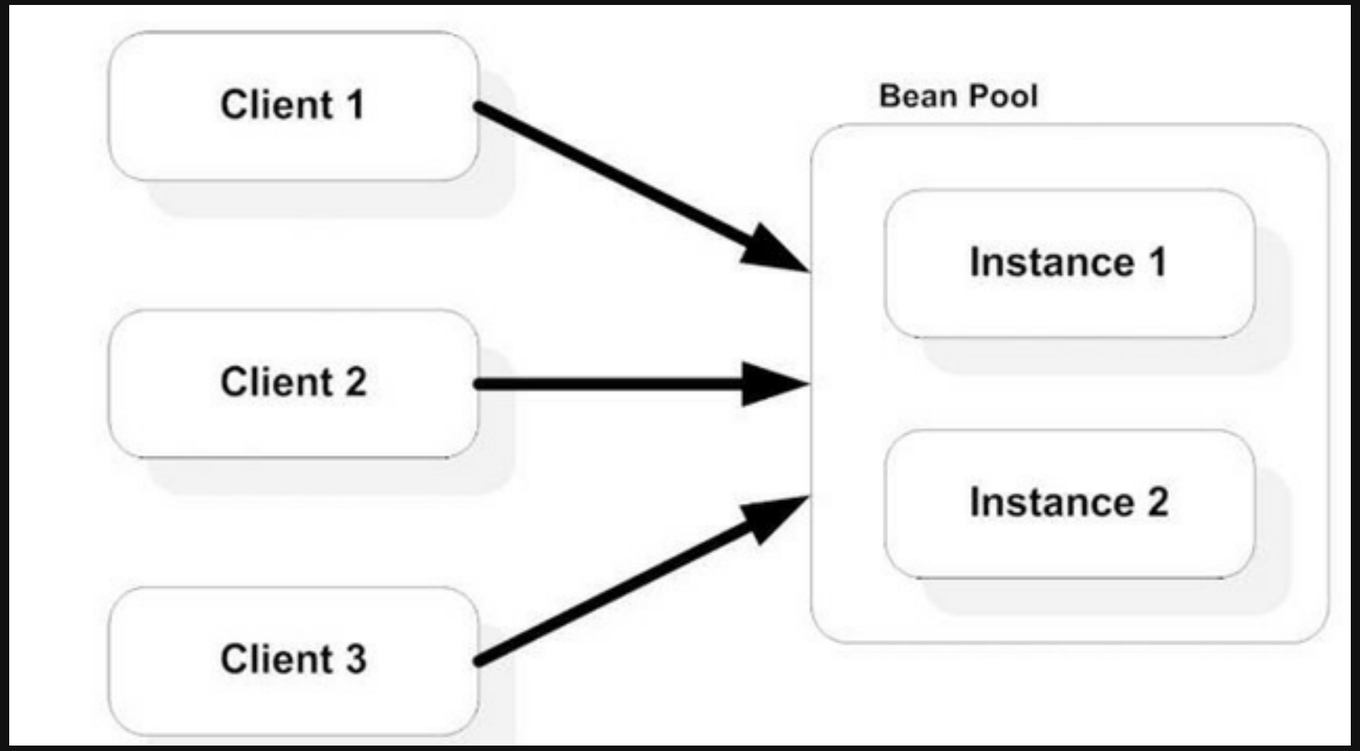
# Session Beans

## Tipos de Session Beans

# Stateless Session Beans

- **Executa procedimento específico sem manter o estado entre chamadas**
  - Ex.: verificar estoque, emprestar livro, debitar conta
- **Thread-safe**
- **Podem ser associados a qualquer cliente**
- **Anotações**
  - @Stateless
  - @EJB

# Stateless Session Beans



# Stateless Session Beans

## Exemplo

# Stateless Session Beans

## Servlet

```
@WebServlet("/ReservaMercadoriaServlet")
public class ReservaMercadoriaServlet extends HttpServlet {
    @EJB ControladorSSB controle;
protected void doGet(...) {
    String merc = request.getParameter(...);
    int reservado = controle.obterQuantidade(merc);
    ...
}
```

## Stateless Session Bean

```
@Stateless
public class ControladorSSB {
    @Inject private Estoque meuEstoque;
    public int obterQuantidade(String merc) {
        int qtdeEstoque = meuEstoque.obterQuantidade(merc);
        ...
    } }
```

# Exercício 2

**Atualize o projeto distribuidora para executar uma ação de reserva em um Estoque em memória utilizando EJBs.**

# Dicas

1. Não será necessário criar projeto ejbclient, nem interfaces no momento
2. No projeto EJB crie uma classe do tipo Estoque que possa ser injetada no EJB. Essa classe terá os dados do estoque
3. Use os parâmetros do request para indicar a mercadoria e quantidade a ser reservada
4. Utilize as anotações características do bean: @Stateless para o Session e @EJB para injeção no servlet

# Stateful Sesion Bean

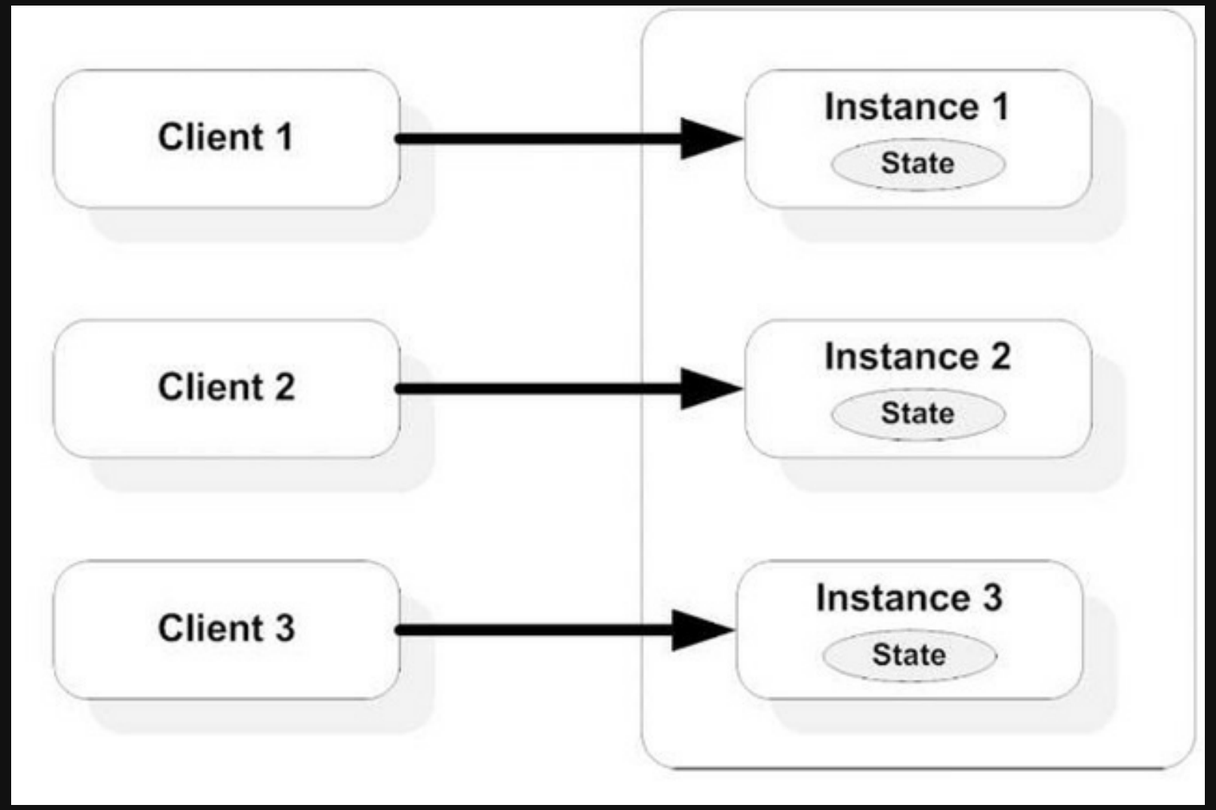
- Mantém o estado conversacional durante as interações com o cliente
- Thread-safe
- Cada cliente é associado a sua instância até que a sessão termine
- Anotação
  - @Stateful
  - @Remove

# Stateful Sesion Bean

## Quando utilizar?

- O componente deve guardar o estado do cliente durante várias chamadas
- O componente gerencia um fluxo de trabalho para o cliente
  - Exemplo clássico: wizards

# Stateful Sesion Bean

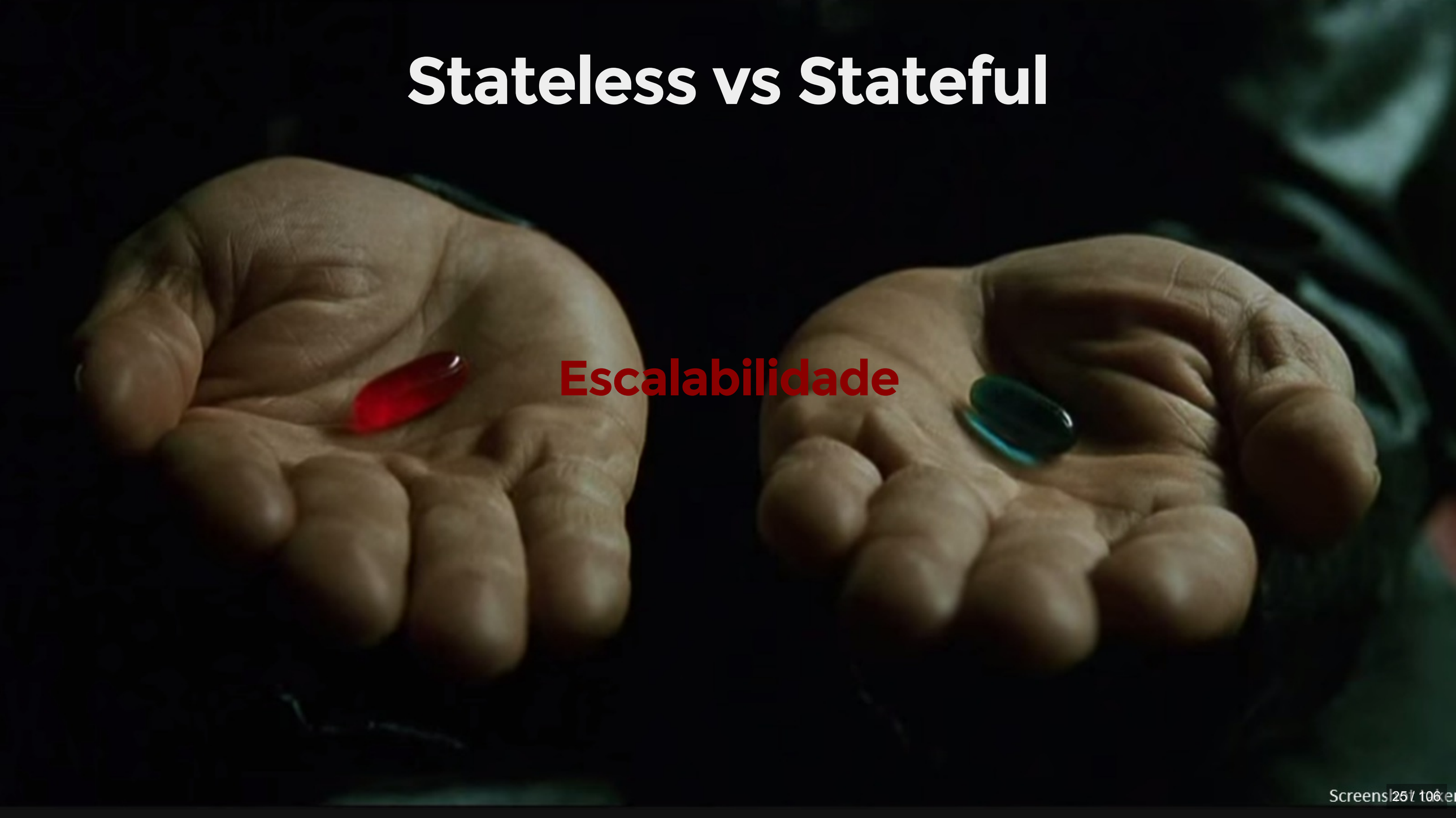


# Stateful Sesion Bean

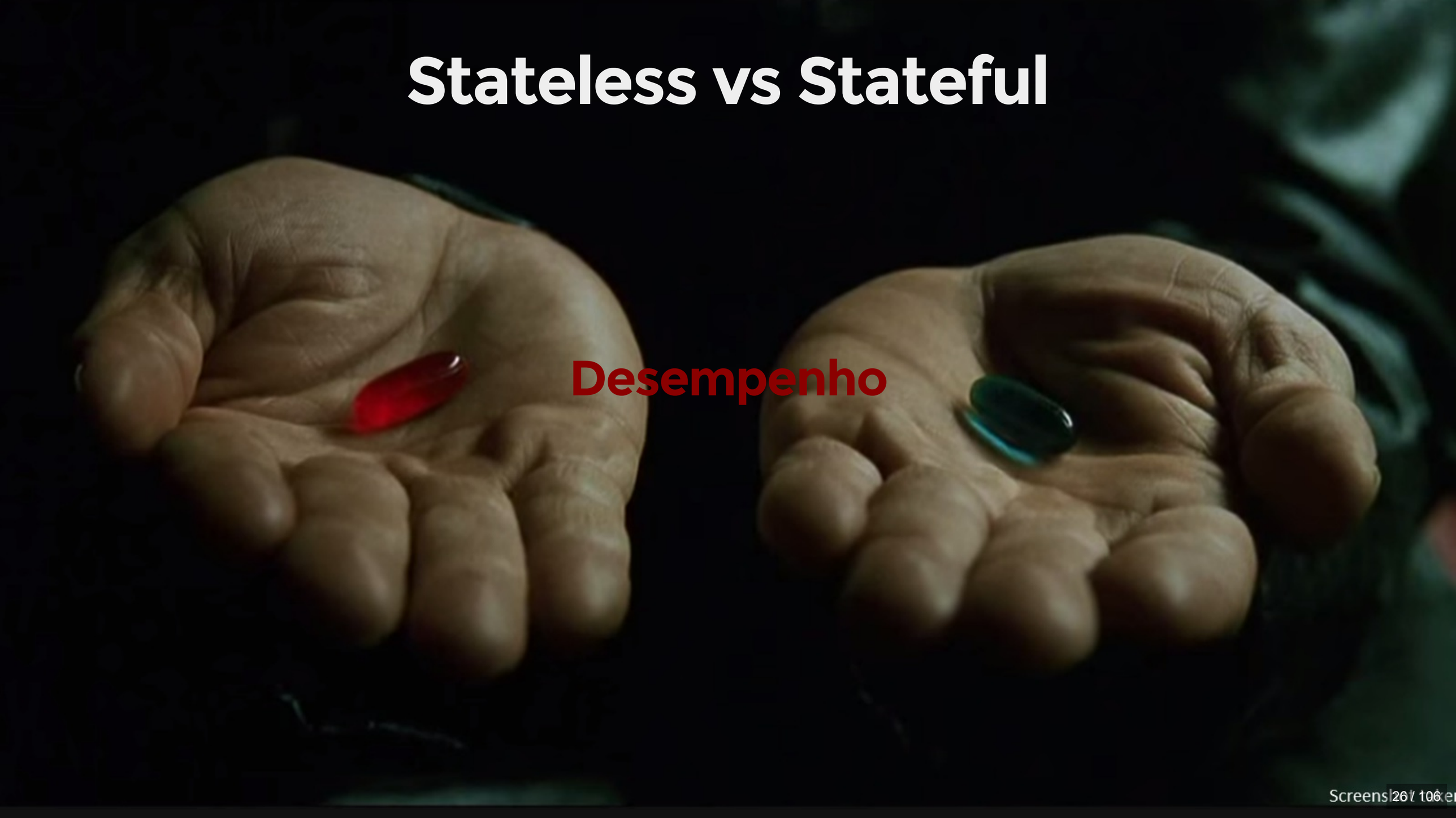
## Stateful Bean

```
@Stateful
public class ComprasBean {

    public void adicionarItem(Item item) {...}

    @Remove
    public Compra processarCompra(...) {...}
}
```

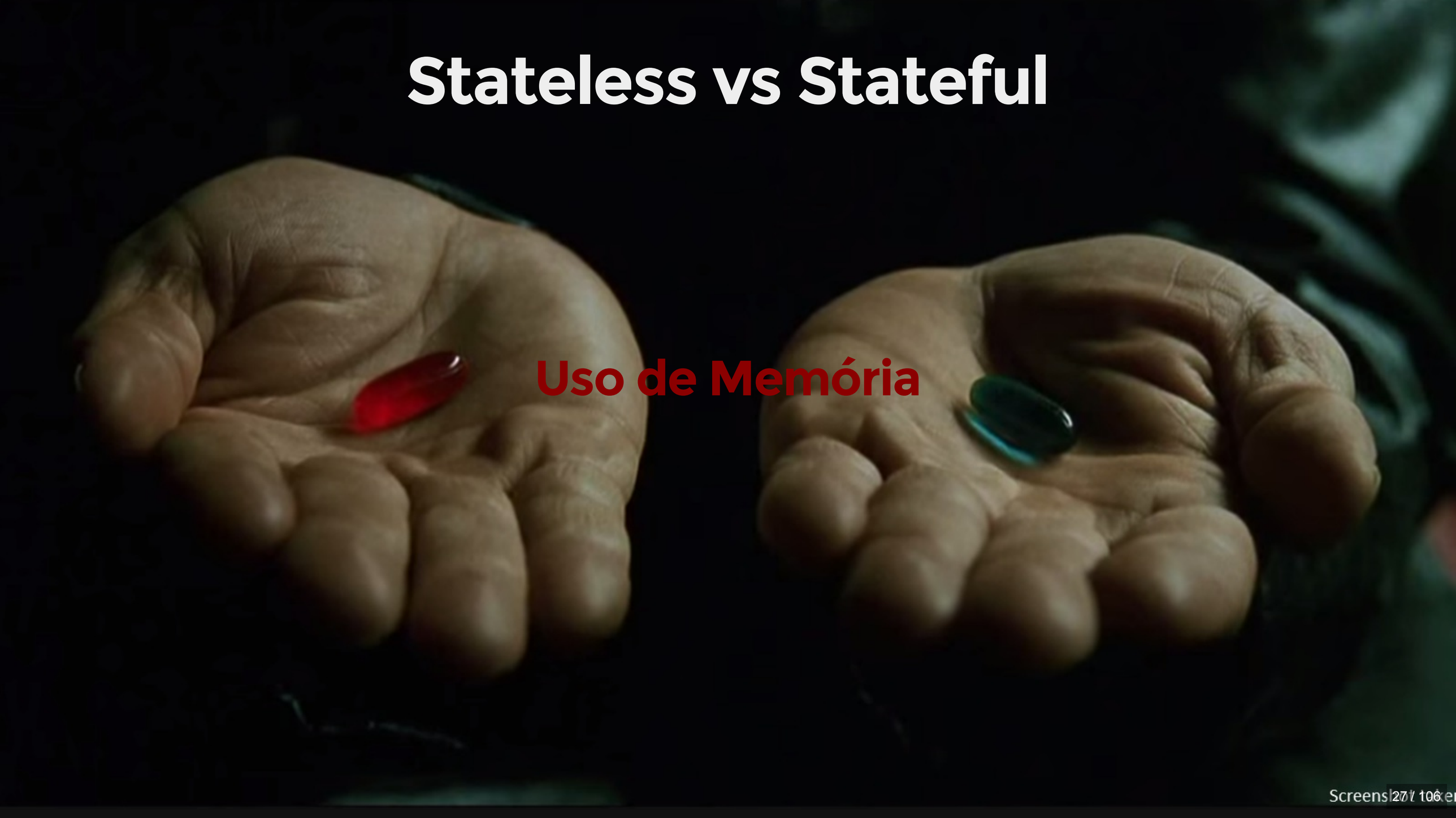
# Stateless vs Stateful



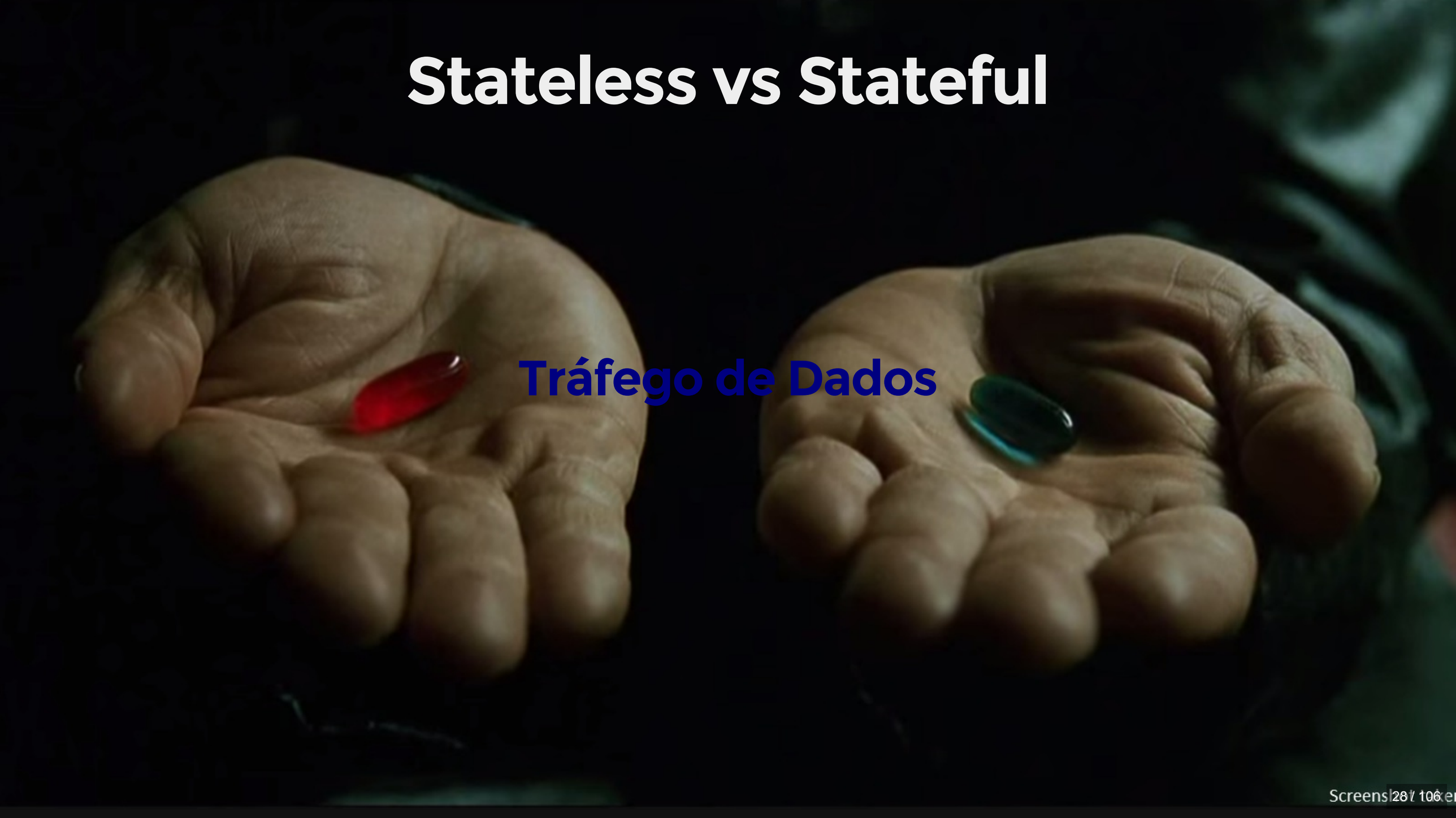
# Stateless vs Stateful



# Stateless vs Stateful



# Stateless vs Stateful



# Singleton

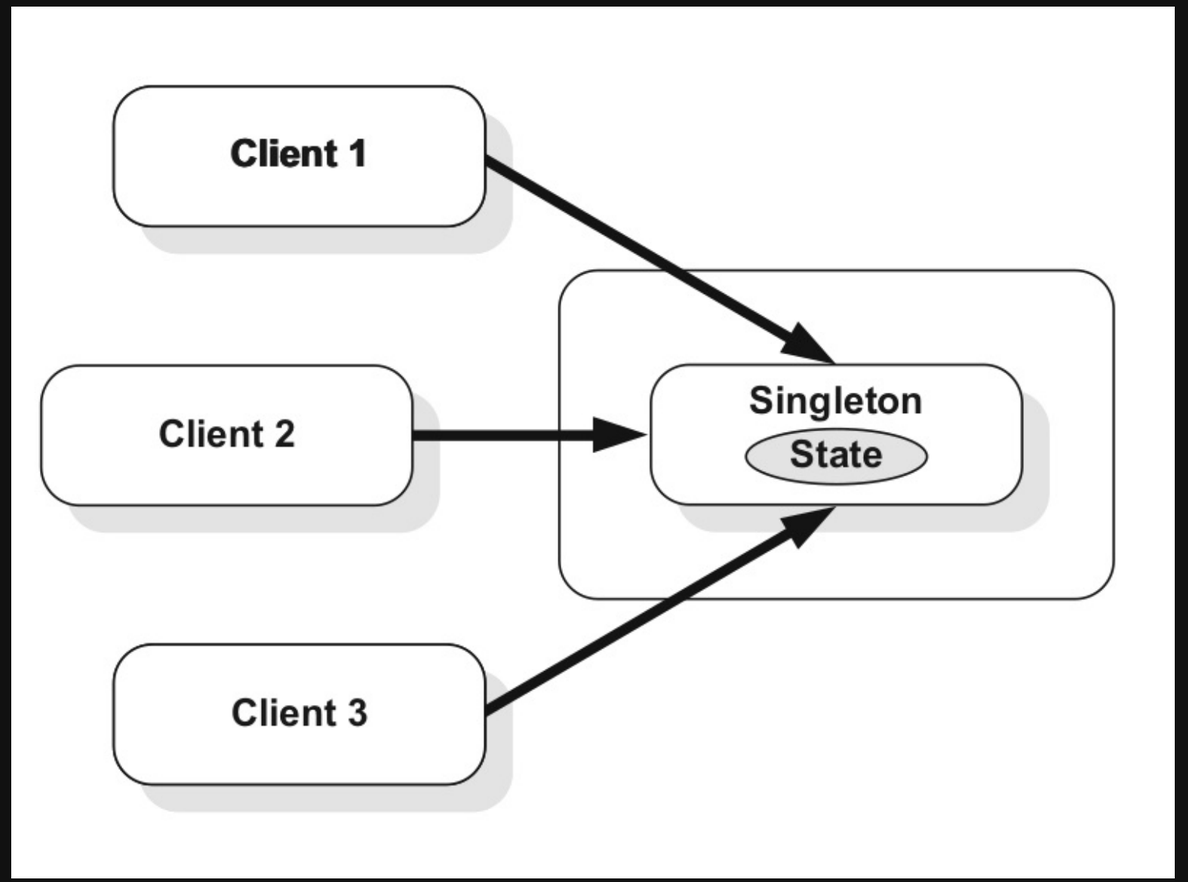
- Instanciado uma única vez e mantido durante todo o ciclo de vida da aplicação
- Mantém o estado durante as chamadas
- Alternativa a variáveis estáticas e servlets de inicialização
- Também executa ações de "limpeza" durante o encerramento da aplicação
- Anotações
  - @Stateful
  - @Startup

# Singleton

## Quando utilizar?

- Quando uma informação precisa ser compartilhada por toda a aplicação, durante todo o seu ciclo de vida
- A aplicação necessita executar alguma ação durante sua inicialização ou encerramento

# Singleton



# Singleton

## Singleton Bean

```
@Singleton
@Startup
public class DefaultFeaturedItem implements FeaturedItem {
    private Connection connection;
    ...
}
```

# Exercício 3

O projeto da distribuidora necessita carregar informações de domínio sobre os fornecedores e situação de produtos nos fornecedores. Crie um session bean que execute essa ação no início da aplicação.

# Beans e Interfaces

# Beans e Interfaces

## Formas de acesso

- Métodos públicos da classe vs Interfaces
  - Porque usar interfaces?
- Interface local vs Remota
  - Quando utilizar?

# Beans e Interfaces

## Formas de acesso

- **Métodos públicos da classe do bean**
  - @EJB
- **Interface Local**
  - @local
- **Interface Remota**
  - @remote

# Beans e Interfaces

## Interface Local

```
@local
public interface MyInterface {...}
...
@Stateless
public class MyClass implements MyInterface {...}
```

```
public interface MyInterface {...}
...
@local(MyInterface.class)
@Stateless
public class MyClass implements MyInterface {...}
```

# Beans e Interfaces

## Interface Remota

```
@Remote
public interface MyRemoteInterface {...}
...
@Stateful
public class MyClass implements MyRemoteInterface {...}
```

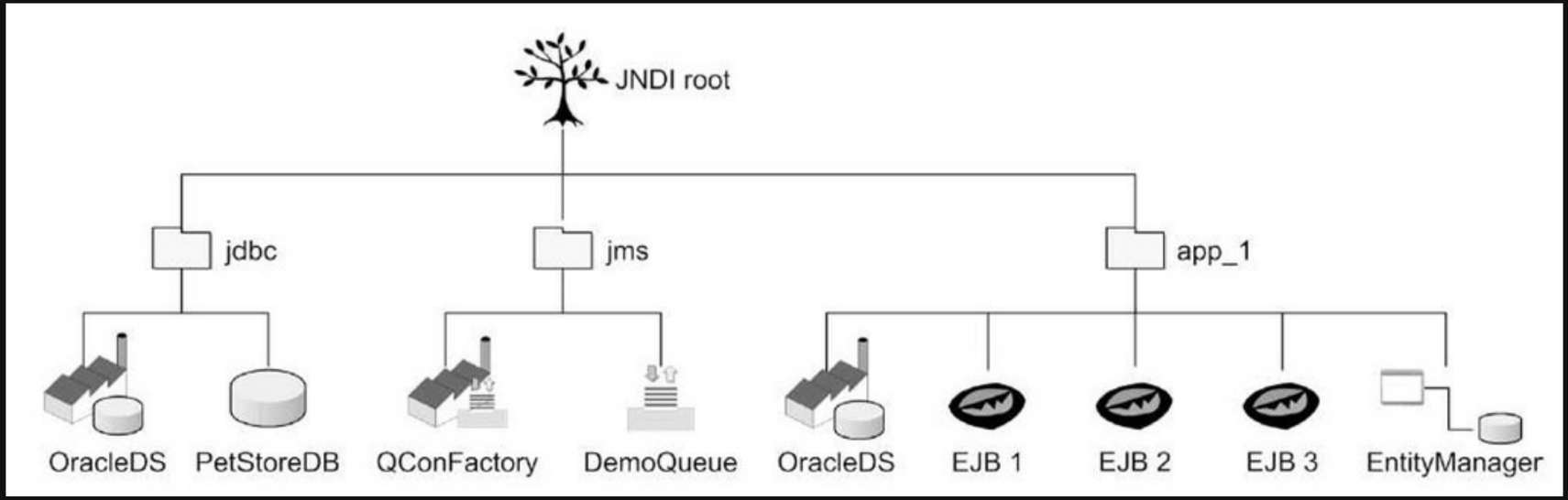
```
public interface MyRemoteInterface {...}
...
@local(MyRemoteInterface.class)
@Stateful
public class MyClass implements MyRemoteInterface {...}
```

# Beans e Interfaces

## Como localizar ou injetar beans que só estão disponíveis remotamente?

# Acessando beans

## JNDI



# JNDI

## Sintaxe da Localização

java:[namespace]/[app]/[modulo]/[bean][!interface]

# Namespaces

| Namespace | Descrição |
| --- | --- |
| global | Compartilhado por todos os módulos e componentes do ambiente |
| app | Nomes compartilhados dentro da mesma aplicação (EAR) |
| module | Nomes compartilhados dentro do mesmo módulo (EJBs e WARs) |

# Nomeando Beans

## @Stateless / @Stateful

| Atributo | Descrição |
| --- | --- |
| name | Nome do Bean. Quando não indicado é usado o nome da classe. |
| mappedName | Nome global atribuído ao Bean. Não é portável. |
| description | Descrição do Bean. |

# Exemplos

Todos os exemplos mencionados tratam de uma aplicação MyApp com módulo MyModule.

# Exemplos

```
package fa7.test;
...
@Stateful
public class MyBean {...}
```

**Nome JNDI**

java:global/MyApp/MyModule/MyBean
java:global/MyApp/MyModule/MyBean!fa7.test.MyBean
java:app/MyModule/MyBean
java:app/MyModule/MyBean!fa7.test.MyBean
java:module/MyBean
java:module/MyBean!fa7.test.MyBean

# Exemplos

```
package fa7.test;
...
@Remote(RemoteInterface.class)
@Stateless(name="RemoteBean")
public class MyBean implements RemoteInterface {...}
```

## Nome JNDI

java:global/MyApp/MyModule/RemoteBean
java:global/MyApp/MyModule/RemoteBean!fa7.test.RemoteInterface
java:app/MyModule/RemoteBean
java:app/MyModule/RemoteBean!fa7.test.RemoteInterface
java:module/RemoteBean
java:module/RemoteBean!fa7.test.RemoteInterface

# Exemplos

## Session Bean

```
package fa7.test;
...
@Remote(RemoteInterface.class)
@Stateless(name="RemoteBean")
public class MyBean implements RemoteInterface {...}
```

## Cliente

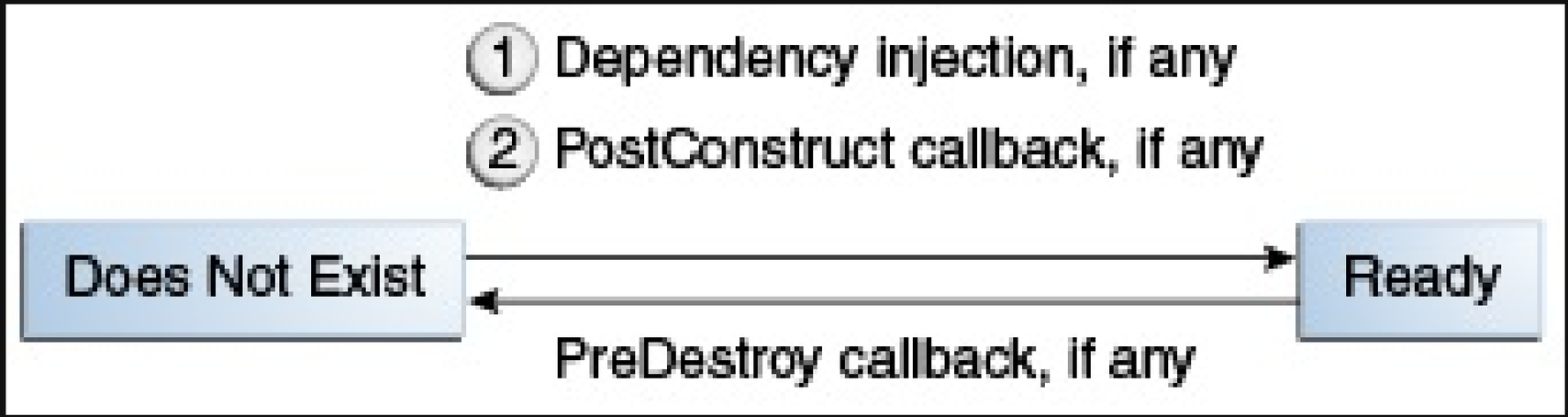
```
...
@WebServlet("/MyServlet")
public class MyServlet extends HttpServlet {
    @EJB(lookup="java:global/MyApp/MyModule/RemoteBean")
    RemoteInterface bean;
   ...
}
```

# Exercício 4

Forneça um Stateful Session Bean para controle de um "carrinho de reservas" de mercadorias onde o cliente possa incluir ou retirar produtos. O Bean deve prover uma operação para conclusão da reserva. Crie um Servlet em uma aplicação Web que não seja distribuída junto com o componente de negócio e acesse o Bean remotamente.

# Ciclo de Vida dos Beans de Sessão

# Stateless Session Beans



# Stateless Session Beans

## @PostConstruct

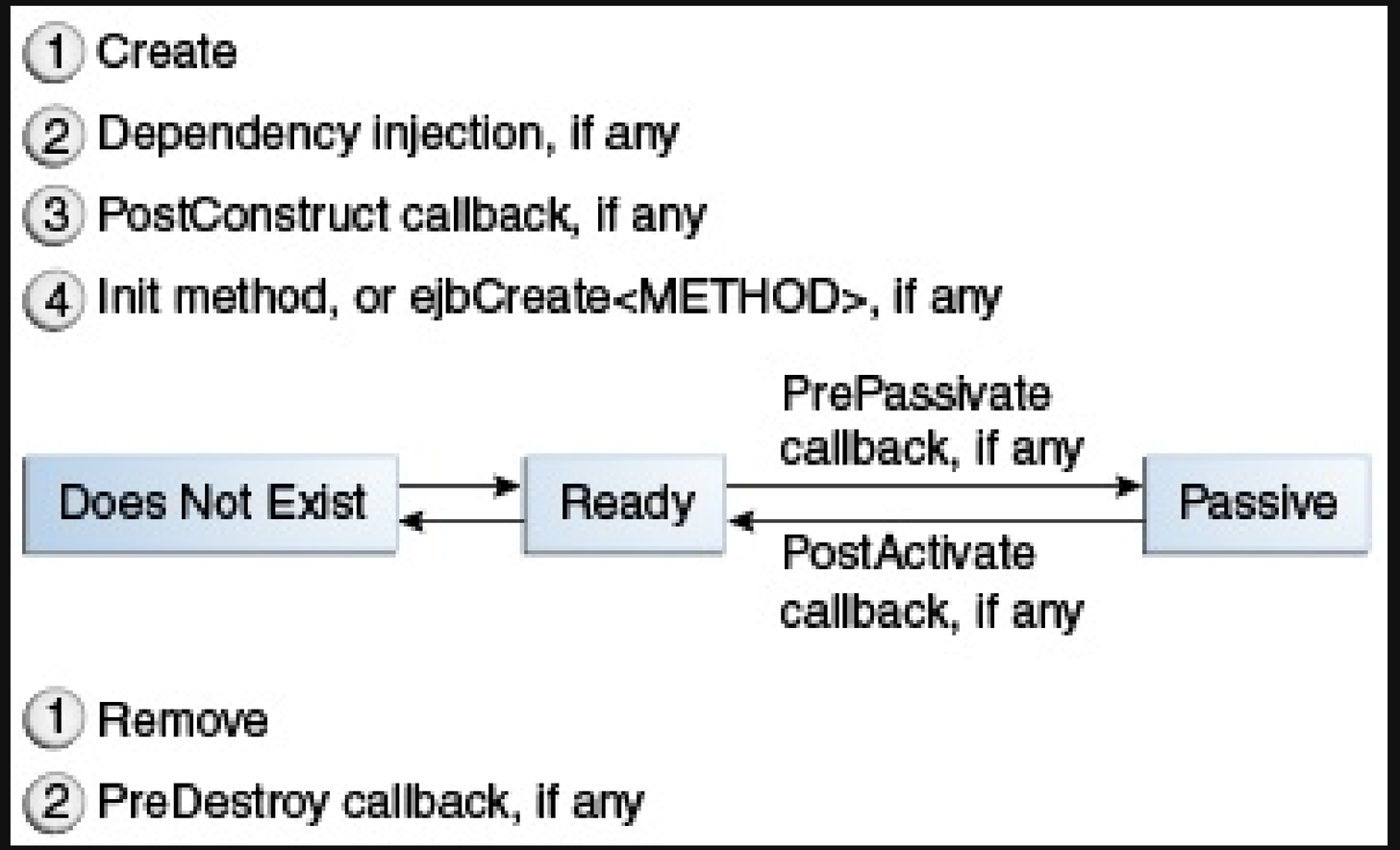
**Executado logo após a construção do Bean e a injeção dos recursos. Alocam-se todos os recursos necessários para a execução.**

# Stateless Session Beans

## @PreDestroy

**Executado logo antes da finalização do Bean. Liberam-se todos os recursos alocados previamente. Após isso, o bean está elegível para o "garbage collector".**

# Stateful Session Beans



# Stateful Session Beans

## @PostConstruct / @PreDestroy

**São executados nos mesmos cenários e de forma similar aos Stateless Session Beans.**

# Stateful Session Beans

## @PrePassivate

**Executado logo antes do bean ser passivado pelo Container. De maneira análoga ao Pre Destroy, este é o momento para liberação de recursos alocados.**

# Stateful Session Beans

## @PostActivate

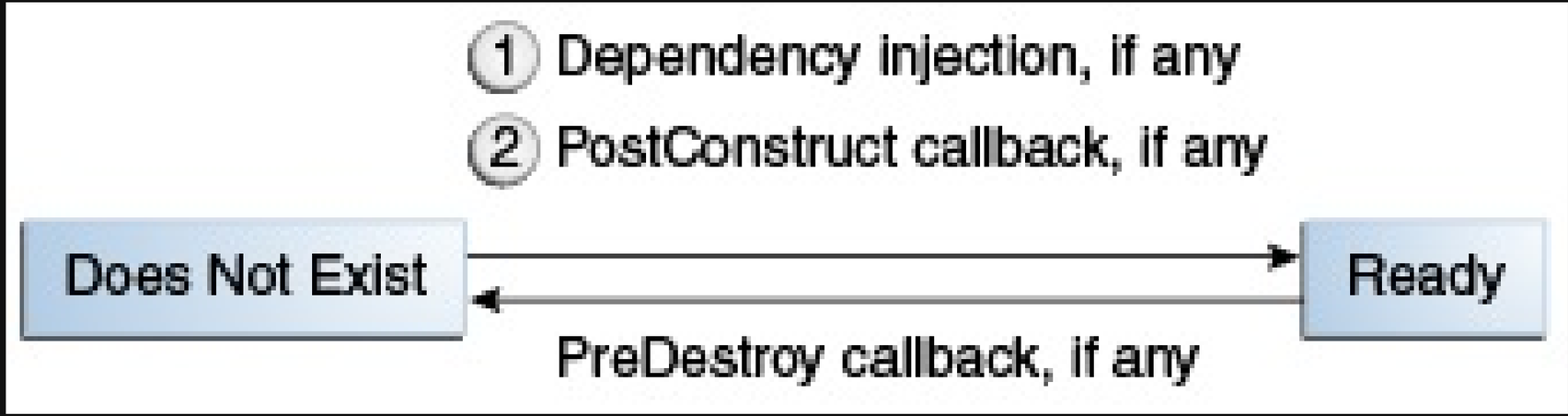
**Executado logo após a ativação do Bean e antes dele responder as chamadas de seu cliente. Alocam-se os recursos necessários.**

# Stateful Session Beans

## @Remove

**Executado logo após a expiração do Bean (bean passivado e sem responder chamadas de seu cliente) ou quando o cliente invoca algum método marcado com a anotação. O bean é marcado para a "coleta de lixo".**

# SIngleton Session Beans



# Exercício 5

# Qual o melhor design para o problema

Uma aplicação corporativa não apresente gargalos em memória, mas o cliente comenta sobre o tempo de espera ao exibir as telas mais complexas da aplicação.

As telas são formadas por diversos componentes visuais que contém dados de domínio da aplicação (por exemplo, listas dos estados, cidades, situações de um produto, etc).

Esses dados são oriundos de um sistema legado e estão armazenados em arquivos de forma sequencial. Qual a sua sugestão para melhorar o desempenho da aplicação?

# Qual o melhor design para o problema

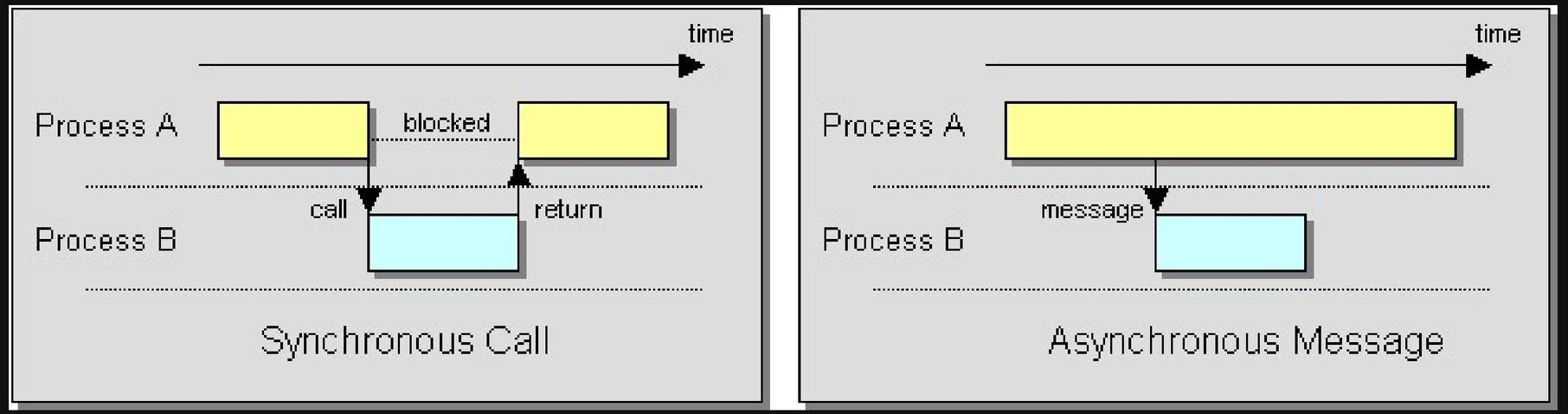
Sua empresa está substituindo o antigo sistema de e-commerce por uma solução em Java EE. Um dos requisitos do novo sistema é atender a uma vasta demanda de compras online na Internet.

Quanto a essa demanda, a indicação é que o sistema poderá ter vários clientes de tipos diferente (browser, aplicações desktop, etc) gerando um número alto de transações por segundo.

Um dos problemas do sistema anterior era o controle da banda, já que o link disponível para esse acesso não é de alta velocidade. Qual a sua proposta de solução para esse problema específico?

# Assincronia na Plataforma Java EE

# Síncrono vs Assíncrono



# Quando usar

- Procedimentos de longa duração em que você não precisa acompanhar sua execução;
- Procedimentos de longa duração em que você precisa conferir o resultado final;
- Aumentar a taxa de processamento da aplicação;
- Melhorar o tempo de resposta da aplicação.

# Como implementar

1. **Utilizar a anotação** @Asynchronous
2. **O resultado estará disponível na forma de uma implementação de** java.util.concurrent.Future<V>

## Cliente

```
public class MyServlet extends HttpServlet {
    @EJB
    MailerBean bean;
    ...
    public void processSomething(...) {
        ... // Processando algo no metodo
        Future<String> status = bean.sendMessage(email);
        ... // Continua processando
        status.get() // Checa status
    }
}
```

## Bean

```
@Stateless
public class MailerBean {
    ...
    @Asynchronous
    public Future<String> sendMessage(String email) {
        try {
            // Faça Algo...
            status = "Enviado";
        } catch (MessagingException ex) {
            status = "Erro no envio: " + ex.getMessage();
        }
        return new AsyncResult<>(status);
    }
}
```

# Sobre a Abordagem

## Pontos Relevantes

- O contexto transacional não é propagado ao usar a invocação assíncrona
- Pode ser usado em qualquer tipo de bean de sessão
- O cliente pode cancelar a execução via API da interface Future
- Essa abordagem não é tolerante à falhas ou quedas do container

# Exercício 6

O procedimento de submissão das reservas da distribuidora necessita ser alterado para realizar verificações de estoque entre os vários fornecedores. Ajuste-o para ser executado de forma assíncrona. Confirme o resultado.

# Mensageria, JMS e MDBs

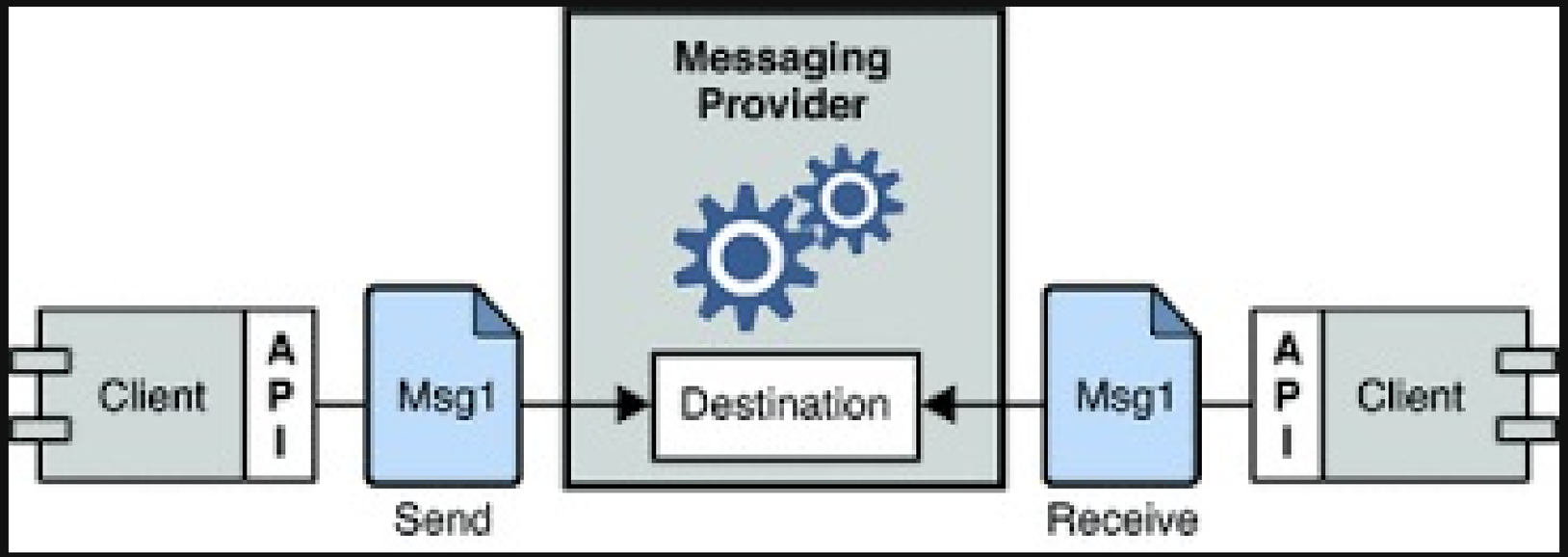
# Tópicos

- Arquitetura orientada a mensagens
- Modelos de troca de mensagem
- JMS
- MDBs

# Arquitetura

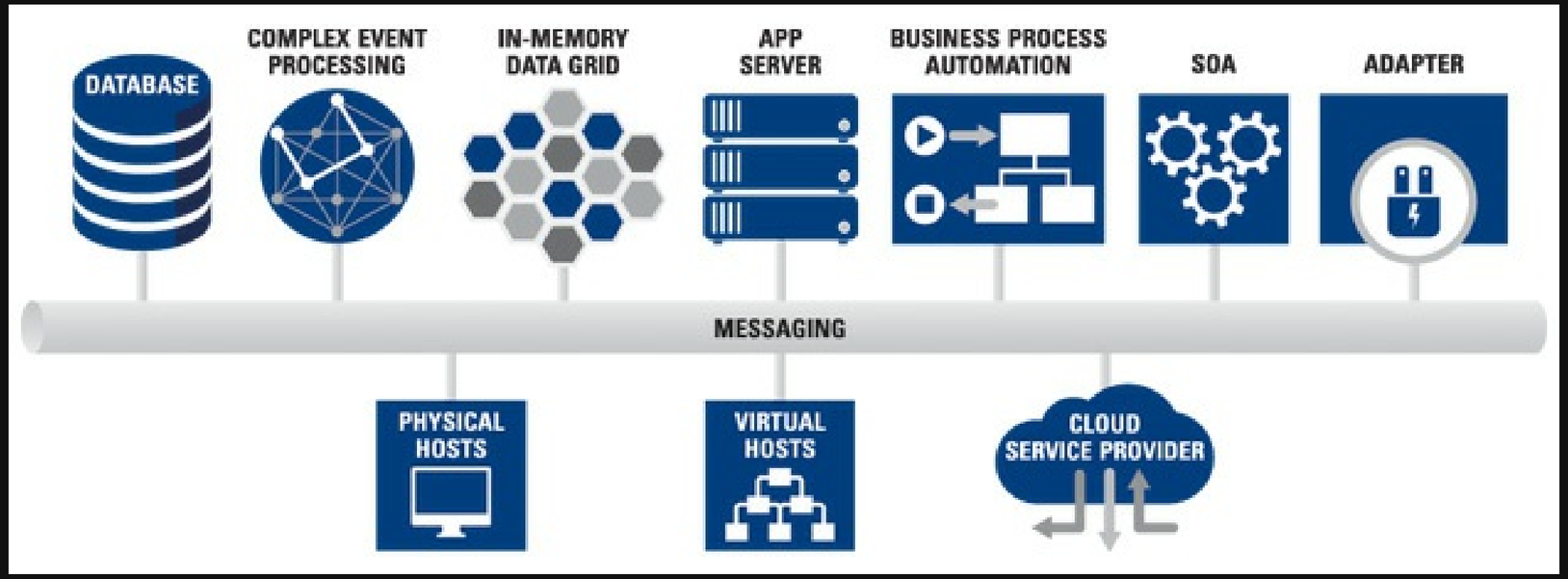
- Mensagens integram os diversos componentes da aplicação
- Assincronia
- Facilita a distribuição
- Dificulta a lógica de programação e debug
- Centrado em um middleware (MOM)

# Arquitetura



# Arquitetura

## Message Oriented Middleware (MOM)



# Conceitos

## Emitente \ Produtor \ Publicador

Gerador de informação que será encaminhada para algum destino.

# Conceitos

## Destinatário \ Consumidor \ Subscritor

Consumidor ou interessado em informação que é trafegada por um meio.

# Conceitos

## Mensagem \ Evento

**É estrutura que carrega a informação trafegada pelo meio de comunicação.**

# Conceitos

## Canal

**Meio pelo qual a mensagem ou evento é trafegado entre o produtor, consumidor e as outras estruturas do meio de comunicação.**

# Conceitos

## Fila

**Estrutura capaz de abstrair a interação entre produtor e consumidor armazenando mensagens para posterior consumo. Desassocia o envio do momento do consumo da mensagem.**

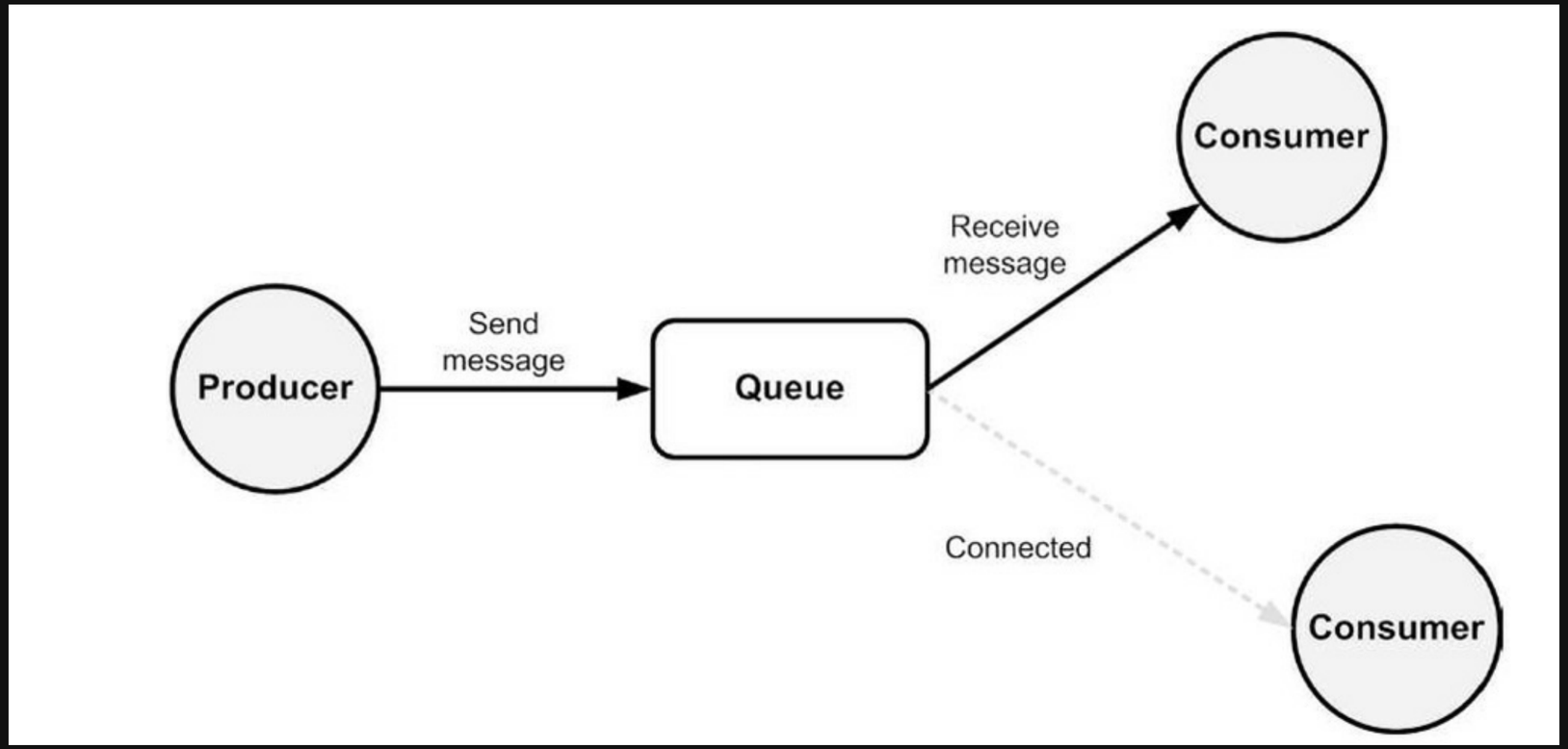
# Conceitos

## Tópico

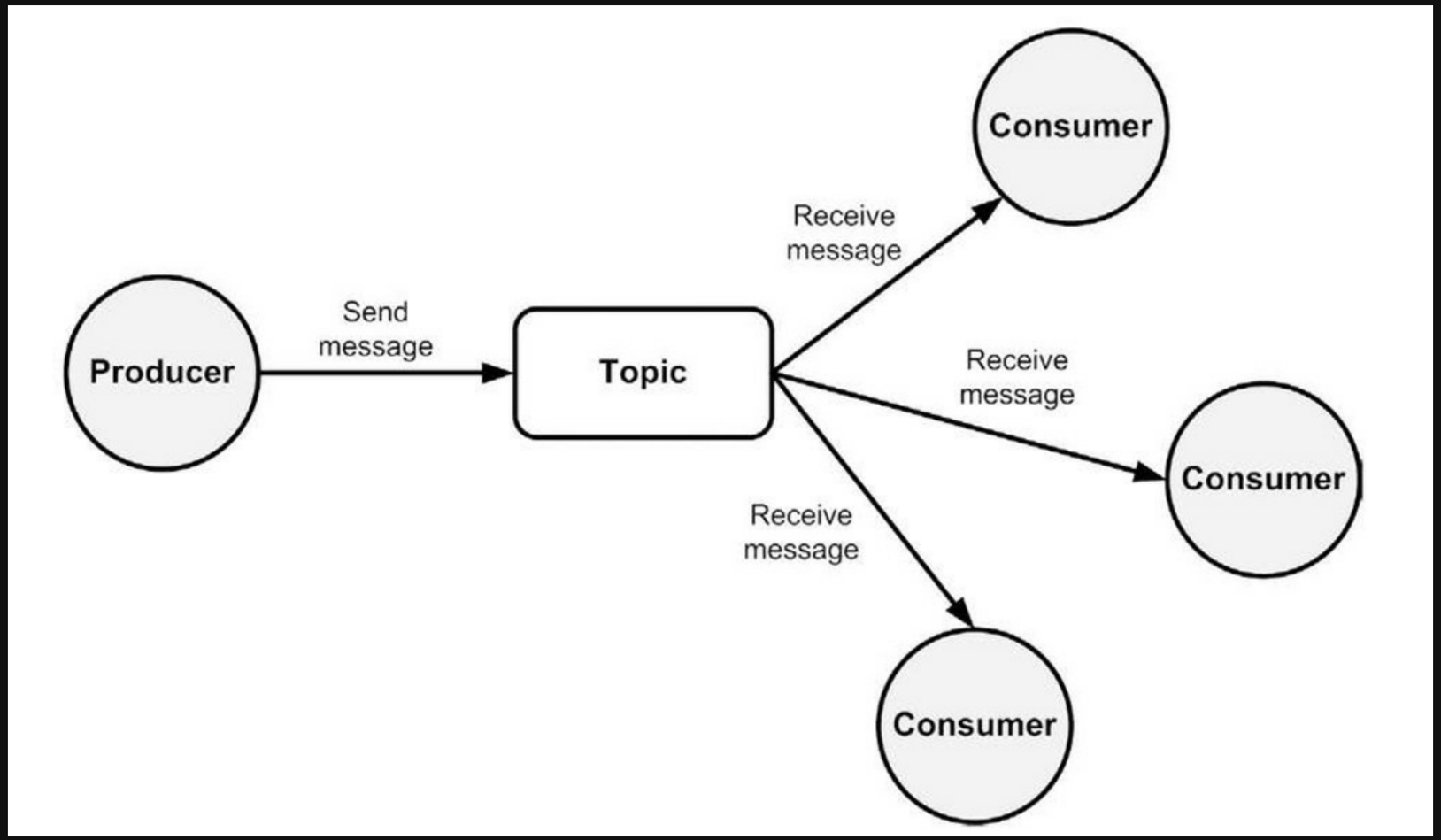
**Estrutura lógica que relaciona um assunto a diversos consumidores desta informação. Enquanto a fila se propõe (naturalmente) a uma relação** um - um **, o tópico favorece um modelo** um - muitos **..**

# Modelos de Comunicação

# Ponto a Ponto



# Publicação \ Subscrição



# Implementações

- Request - Reply
- Master - Worker
- Scatter - Gather
- Map - Reduce

# Exercício 7

## Identifique as estruturas e o modelo de comunicação nos exemplos a seguir:

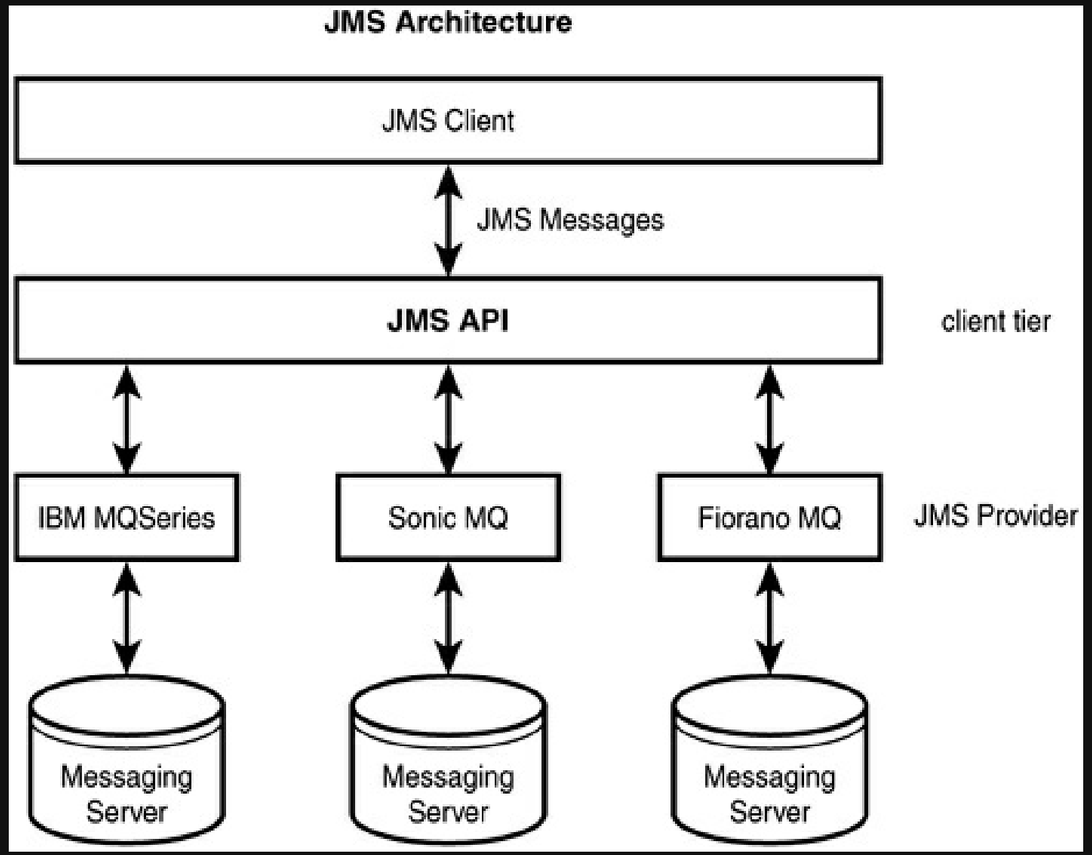
1. **Estabelecimento de uma conexão FTP entre sua máquina e um servidor**
2. **O broadcast de pacotes UDP na sua LAN**
3. **A troca de um segredo entre duas amigas confidentes**

# JMS

- Permite criar, enviar e receber mensagens
- API define um conjunto de interfaces que desacoplam as aplicações dos provedores do serviço
- Garante assincronia e confiabilidade
- Transacional

# JMS

## Arquitetura



# JMS

## Principais Conceitos

- Cliente JMS
- Provedor JMS
- Mensagem
- Objetos

# JMS

## Modelo de Comunicação

- Ponto a Ponto
- Publicação / Subscrição

# JMS

## Consumo de Mensagens

- Síncrono
- Assíncrono

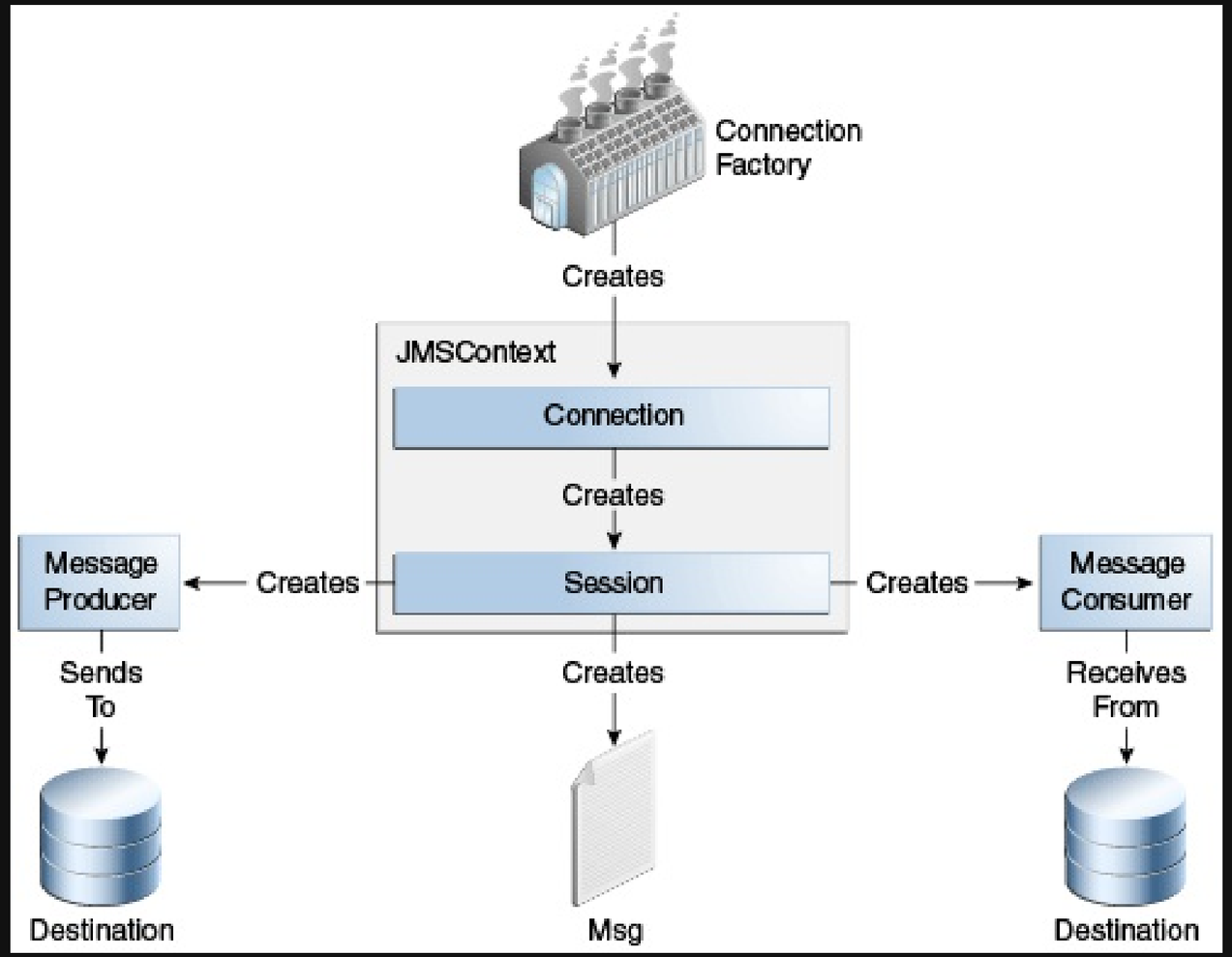
# JMS

## Estruturas de Programação

- ConnectionFactory
- Connection
- Context
- Message Producers
- Message Consumers
- Queue **e** Topic
- Message

# JMS

## Estruturas de Programação



# JMS

## Produzindo mensagens

# JMS

## APIs anteriores

```
...
@Resource("jms/Qcf")
QueueConnectionFactory factory;
...
@Resource(lookup = "java:/jms/myQueue")
private Queue myQueue;
...
Connection conn = factory.createConnection(...);
Session session = conn.createSesion(...);
session.createMessage(...);
session.CreateMessageProducer(...);
producer.sendMessage(message);
```

# JMS

## Java EE 7

```
@Inject
@JMSConnectionFactory("java:/ConnectionFactory")
private JMSContext ctx;
...
@Resource(lookup = "java:/jms/myQueue")
private Queue myQueue;
...
ObjectMessage message = context.createObjectMessage();
// Faça algo com a mensagem
```

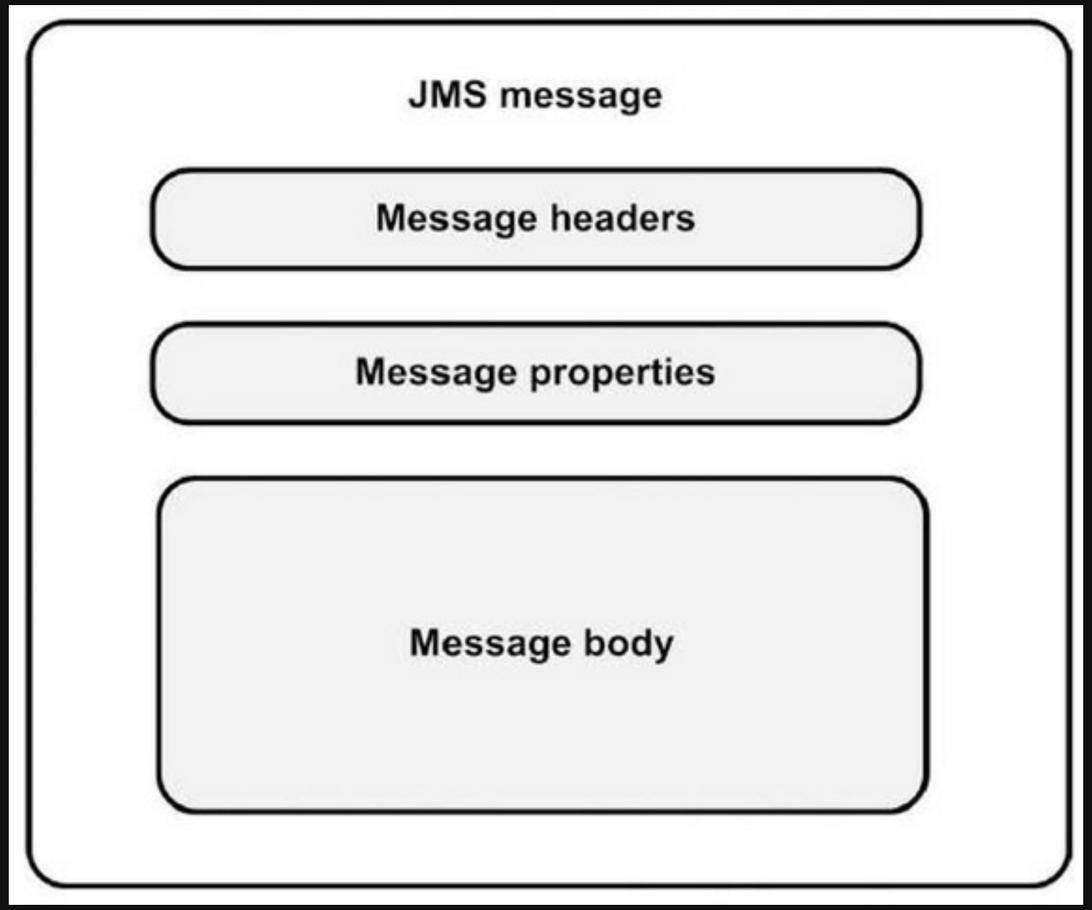
# JMS

## Resumindo...

1. **Crie as estruturas JMS (filas, tópicos, fábricas, etc);**
2. **Injete o contexto JMS no seu Bean indicando a fábrica;**
3. **Use o contexto para criar a mensagem e o respectivo produtor;**
4. **Envie a mensagem para o destino adequado.**

# JMS

## Mensagem



# JMS

## Tipos de Mensagem

| Tipo | Descrição |
| --- | --- |
| TextMessage | Mensagem é um texto do tipo java.lang.String |
| MapMessage | Um conjunto de pares chave - valor |
| ByteMessage | Mensagem formada por bytes |
| ObjectMessage | Mensagem é um objeto serializado |
| StreamMessage | Mensagem é uma stream |
| Message | Mensagem sem conteúdo |

# Qual a abordagem para consumir mensagens em uma aplicação Java EE?

# Message Driven Beans

# MDBs

## Definição

Bean corporativo que permite a execução de uma ação de forma assíncrona dentro de um contexto transacional.

# MDBs

## Características

- **Assincronia**
- **Multithreaded**
- **Sem estado**
- **Não são invocados diretamente, mas sim acionados pelo container**
- **Deve possuir construtor Padrão**
- **Implementa a interface** MessageListener

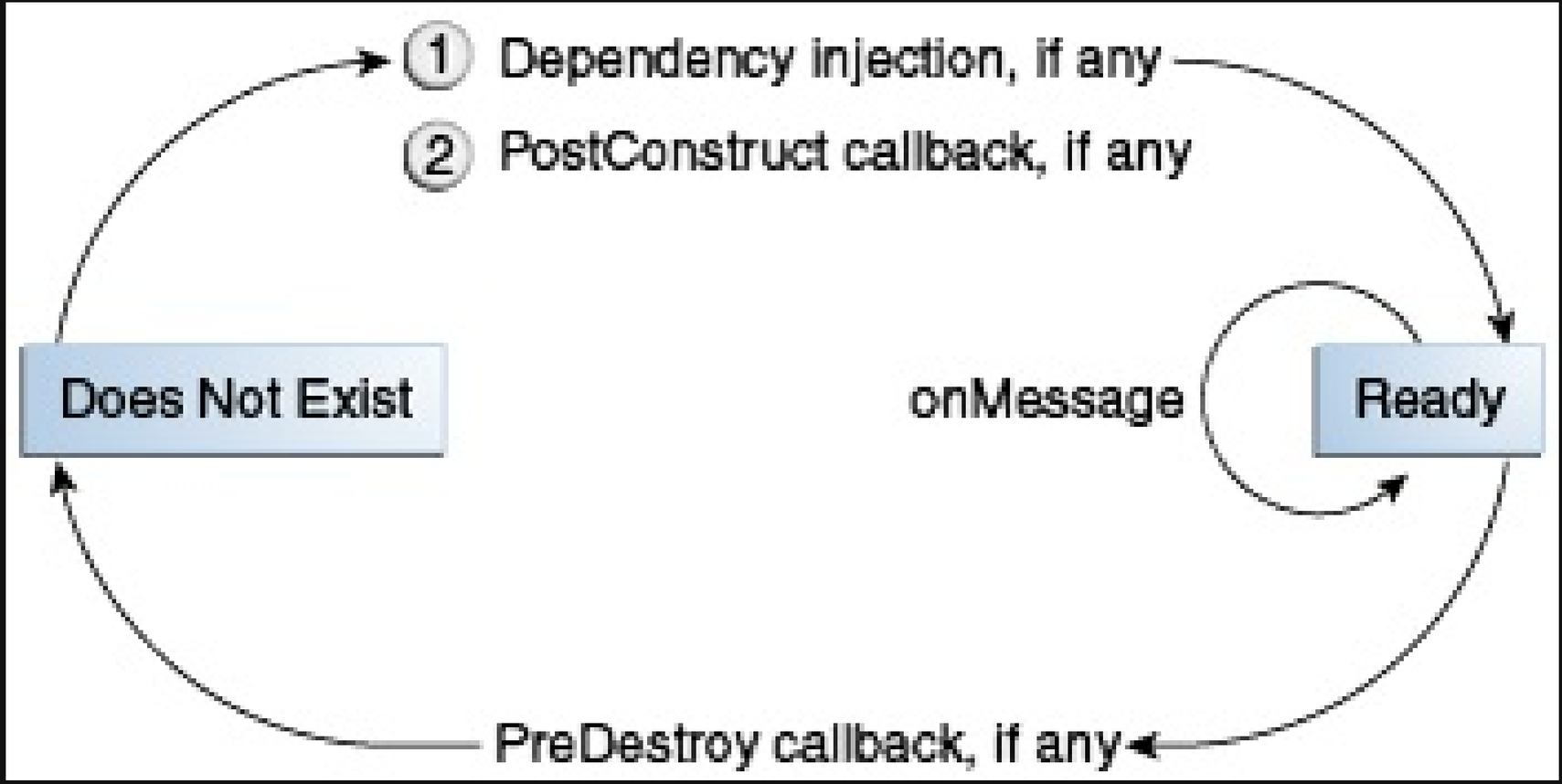
# MDBs

## Quando usar?

- Ganho de taxa de processamento
- Processamentos de longa duração
- Desacoplar processo cliente do servidor
- etc

# MDBs

## Ciclo de Vida



# MDBs

## @MessageDriven

| Atributo | Descrição |
| --- | --- |
| ActivationConfigProperty | Propriedades de ativação do Bean. |
| mappedName | Nome global atribuído ao Bean. Não é portável. |
| description | Descrição do Bean. |
| MessageListenerInterface | Interface acionada quando da chegada de uma mensagem. |
| name | Nome atribuído ao EJB. |

# MDBs

# Implementação

```
    @MessageDriven(
        activationConfig = { @ActivationConfigProperty(
        propertyName = "destination", propertyValue = "myQueue"),
        @ActivationConfigProperty(
        propertyName = "destinationType", propertyValue = "javax.jms.Queue")
        }, mappedName = "myQueue")
    public class ComprasMDB implements MessageListener {

    public ComprasMDB() {...}

    public void onMessage(Message message) {
        ObjectMessage objMessage = (ObjectMessage) message;
        // Processar a Mensagem
    }
}
```

# Exercício 8

Ajuste a comunicação assíncrona implementada no exercício 6 para que ela seja realizada pelo MDB. Fique livre para utilizar o tipo de mensagem que desejar.