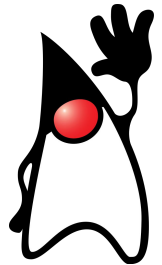


# Desenvolvimento com Frameworks e Componentes

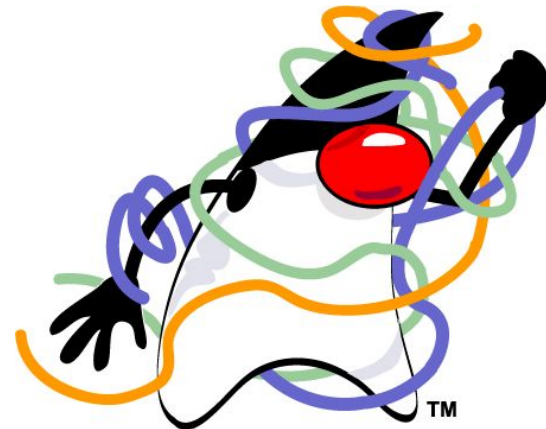
Michel Vasconcelos

`michel.vasconcelos@gmail.com`

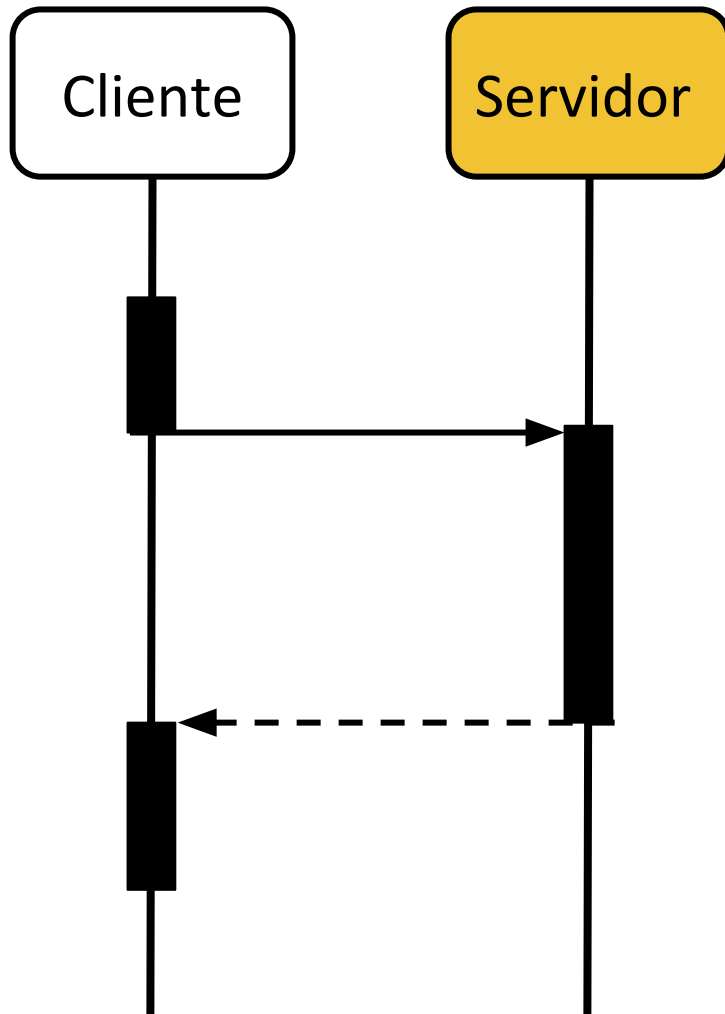


Previously on Developing with  
Frameworks and Components  
@UNI7...

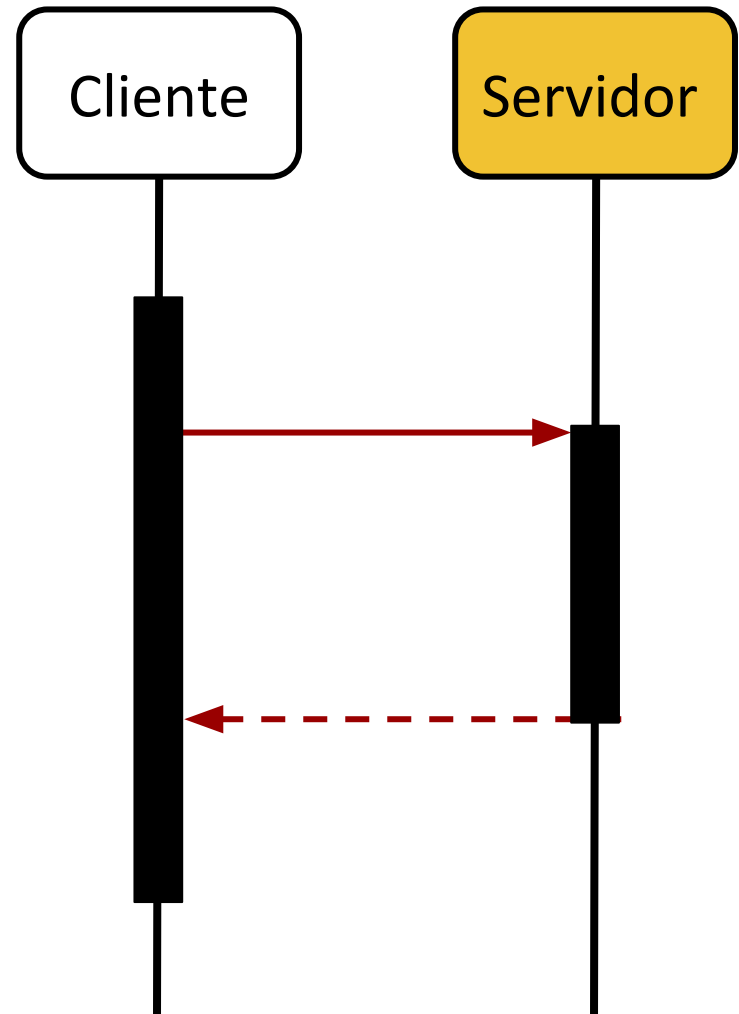
# Assincronia na Plataforma Java EE



## Síncrono



## Assíncrono



## Sincronia

## Assincronia

---

### Complexidade

Simple

Complexo

---

### Consistência

Forte

Eventual

---

### Desempenho

Dependente das rotinas

Dependente do design  
(e rotinas)

---

### Modelo

Modelo de prog. habitual

Threading, Eventos, etc

# Quando usar?

- Processamentos longos
- Melhoria no tempo de resposta
- Aumentar o *throughput*

# Métodos assíncronos

- `@Asynchronous`
- API Java Future

```
public class MyServlet extends HttpServlet {
    @EJB private MailerBean bean;

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

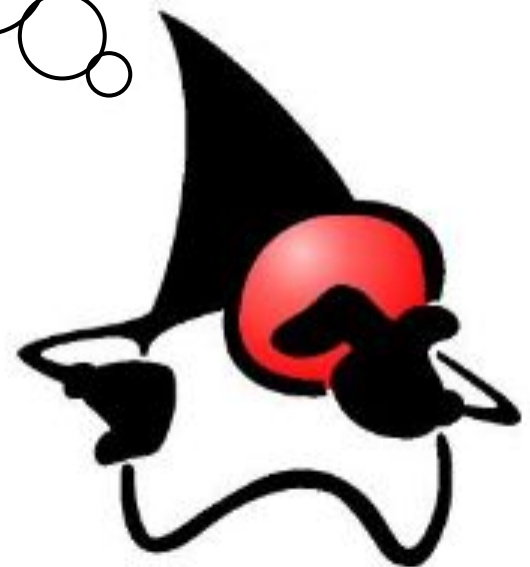
```
@Stateless
public class MailerBean {
    ...
    @Asynchronous
    public Future<String> sendMessage(String email) {
        // Faça Algo...
        return new AsyncResult<>(status);
    }
}
```



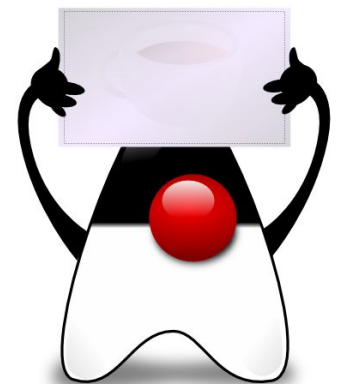
# Pontos Relevantes

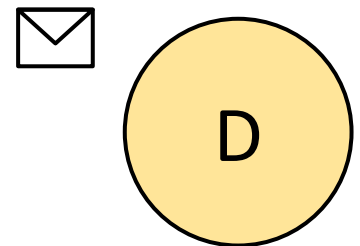
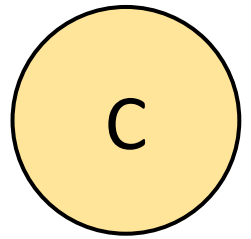
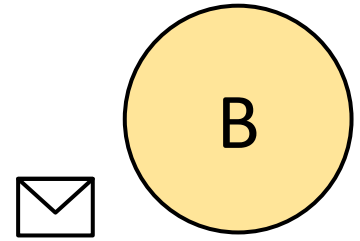
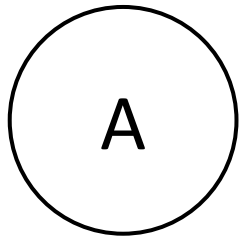
- O contexto transacional não propagado
- Usado em qualquer tipo de bean de sessão
- Cancelamento possível via API Future
- Não tolerante à falhas (Ex.: queda do container)

**Ouvi falar que há outras  
opções!**

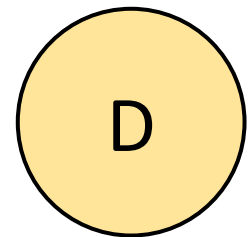
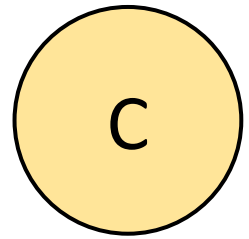
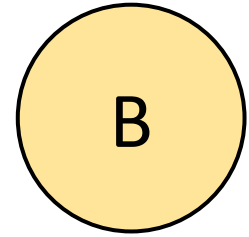
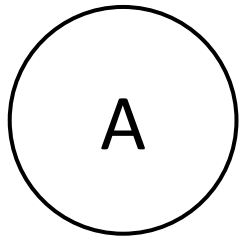


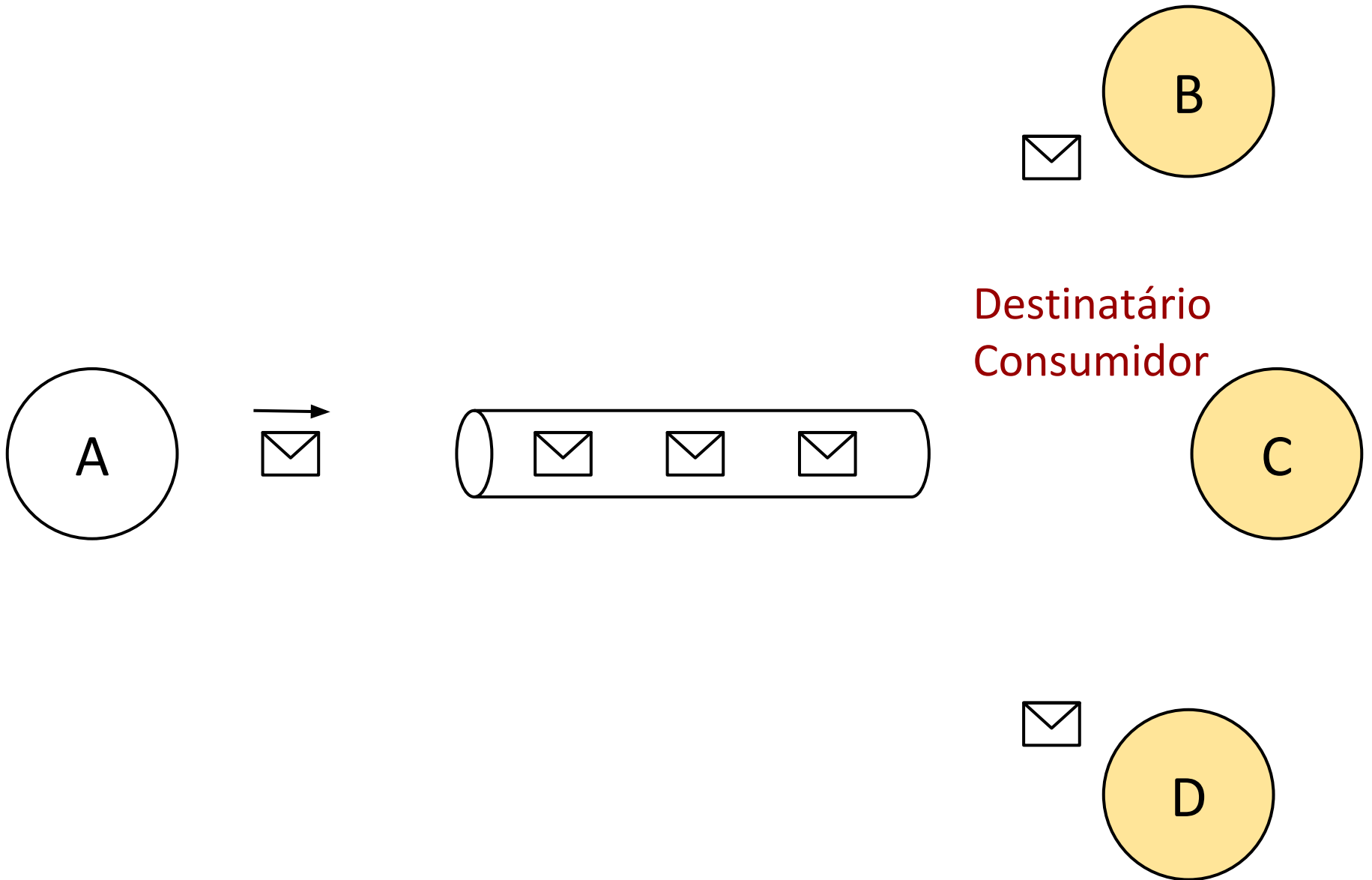
# Serviço de Troca de Mensagens

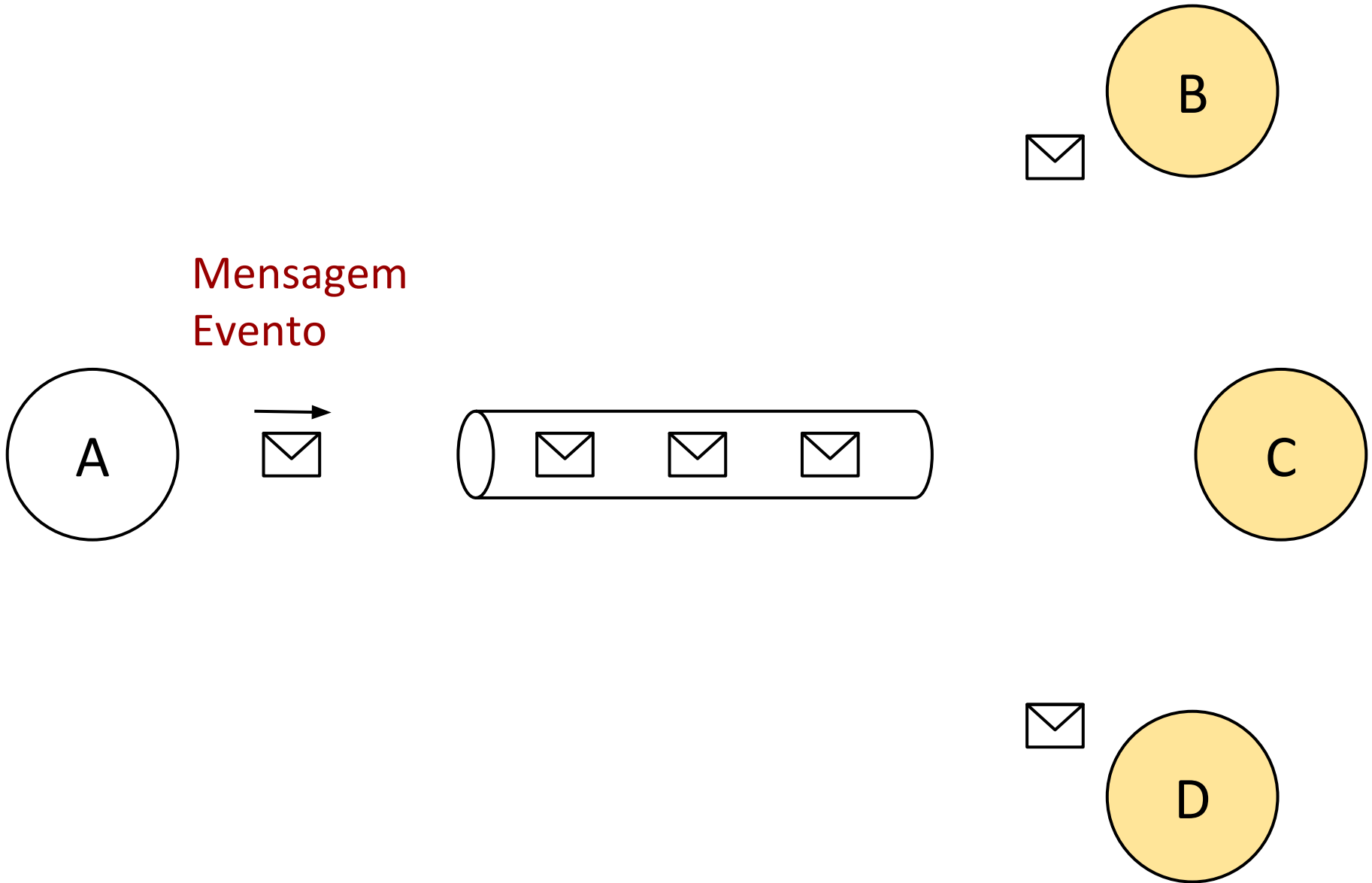


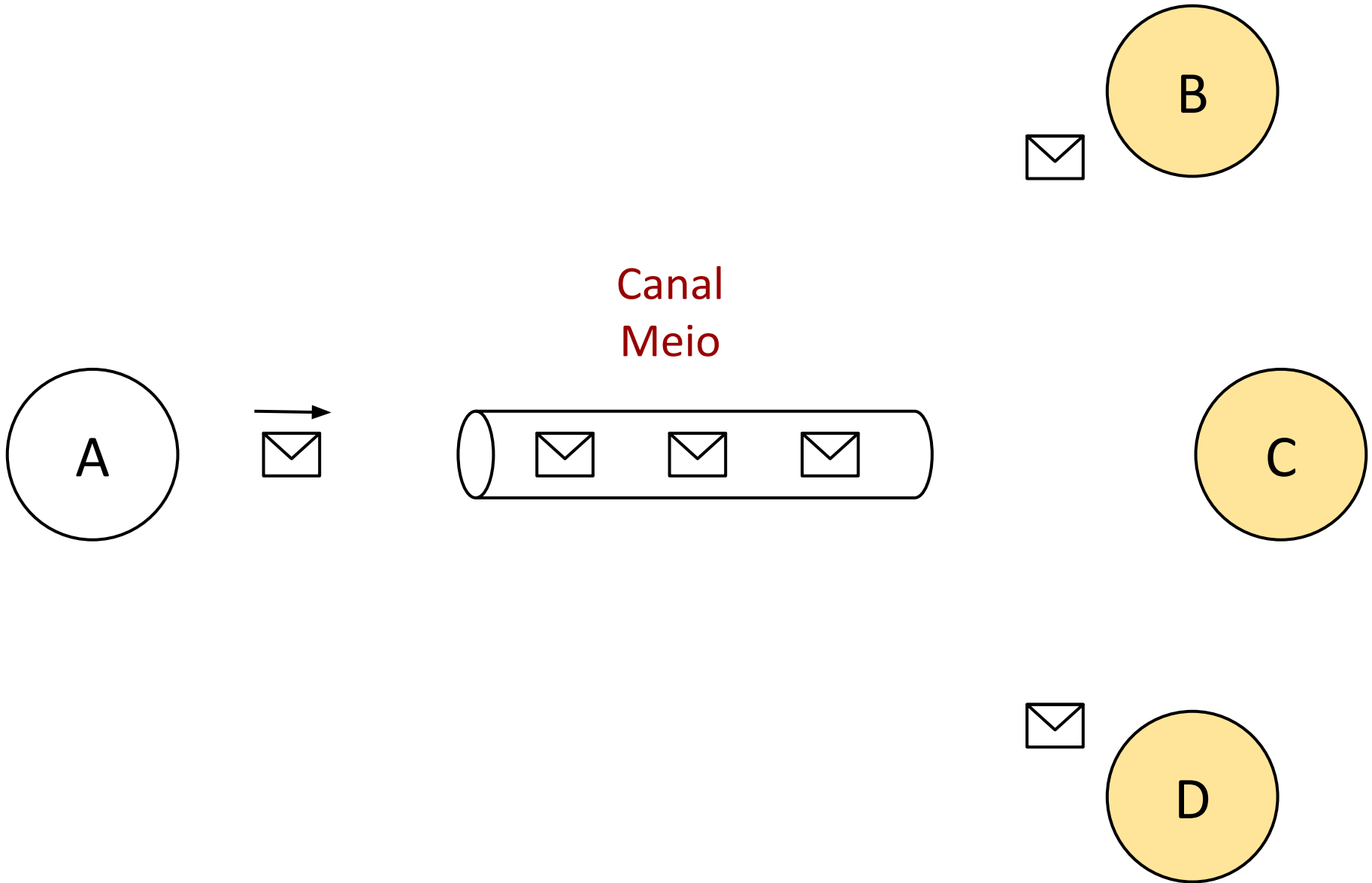


Emitente  
Produtor



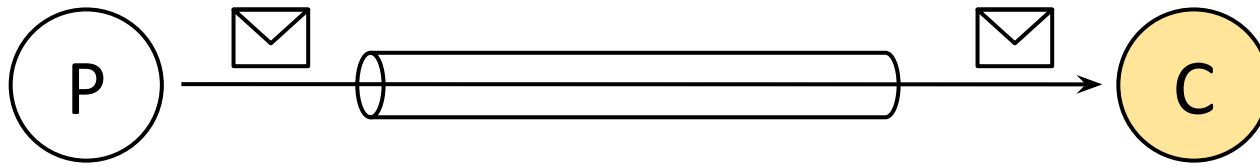




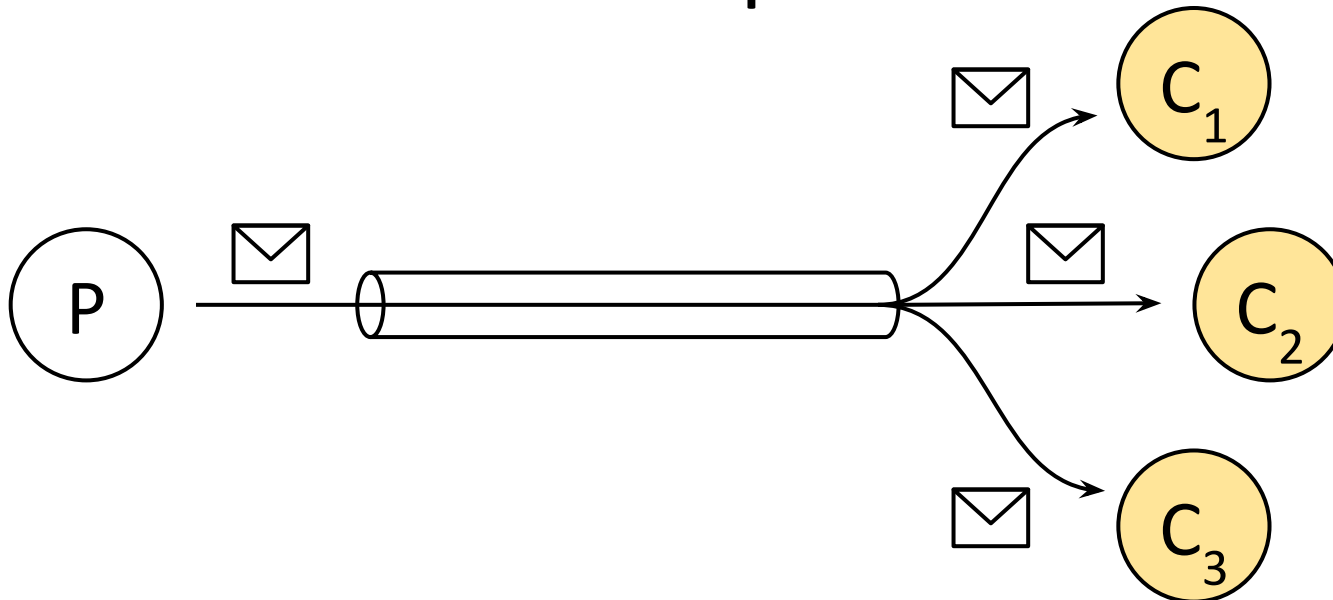




## Ponto a Ponto



## Ponto a Multiponto



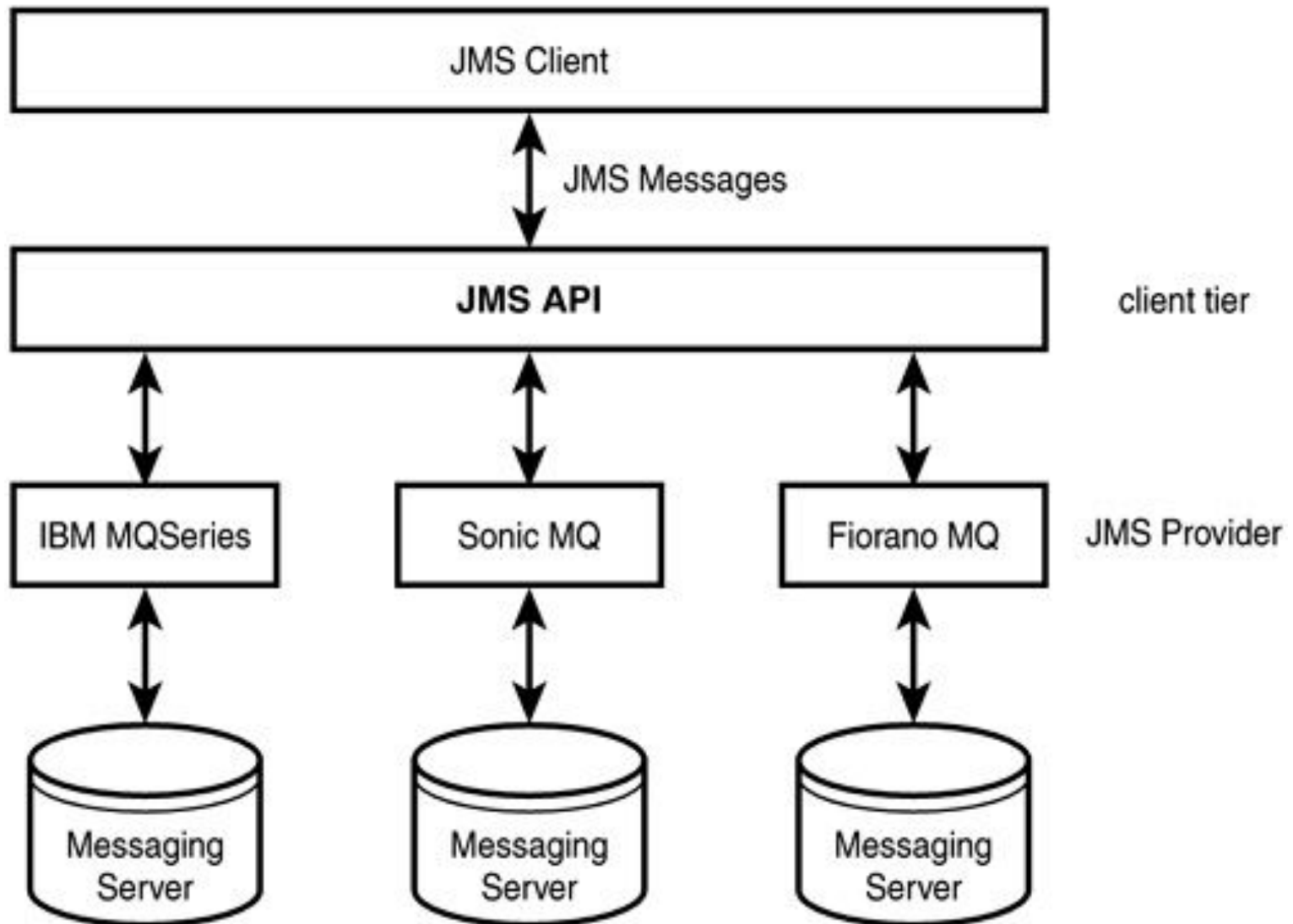
# Qual o modelo de comunicação?

- Professor explicando um assunto em sala?
- Download de um arquivo via FTP
- Download de arquivo via Torrent
- Broadcast de pacotes UDP na LAN
- Troca de segredo entre amigos confidentes

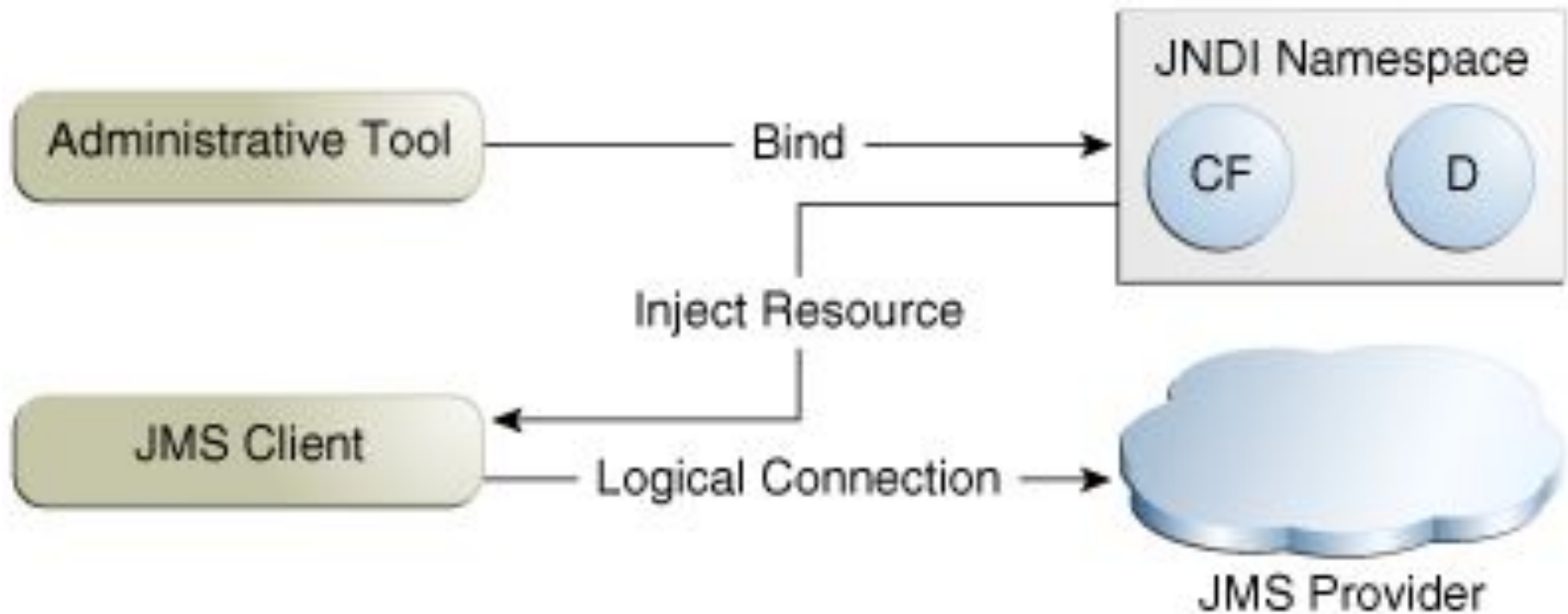
# Java Message Service

Criação, envio e recebimento de mensagens por meio de uma API transacional entregando assincronia e confiabilidade

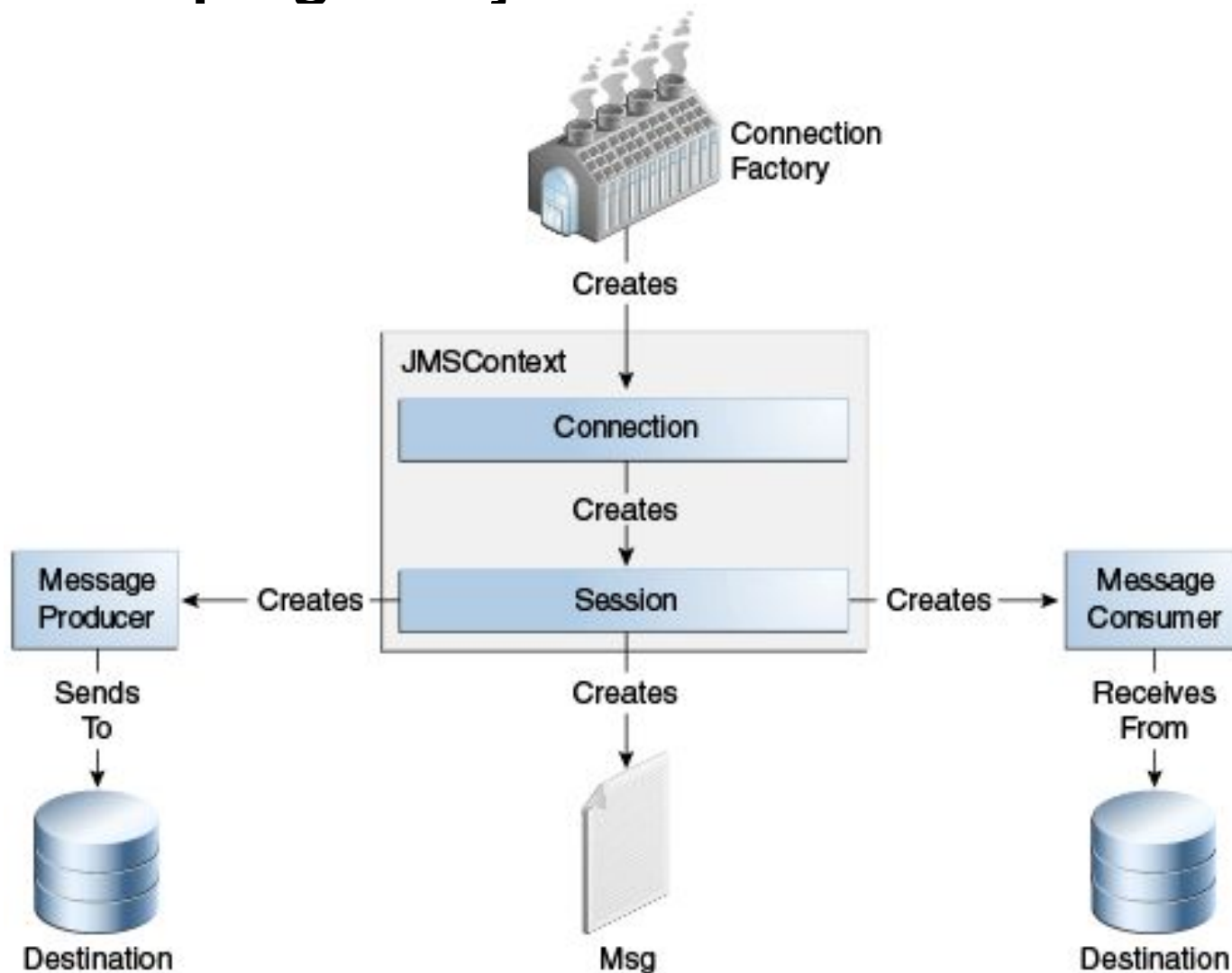
## JMS Architecture



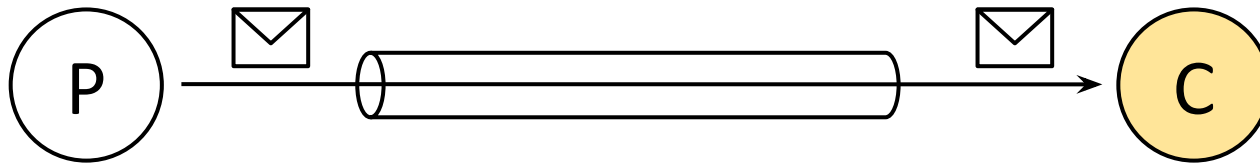
# Arquitetura da API



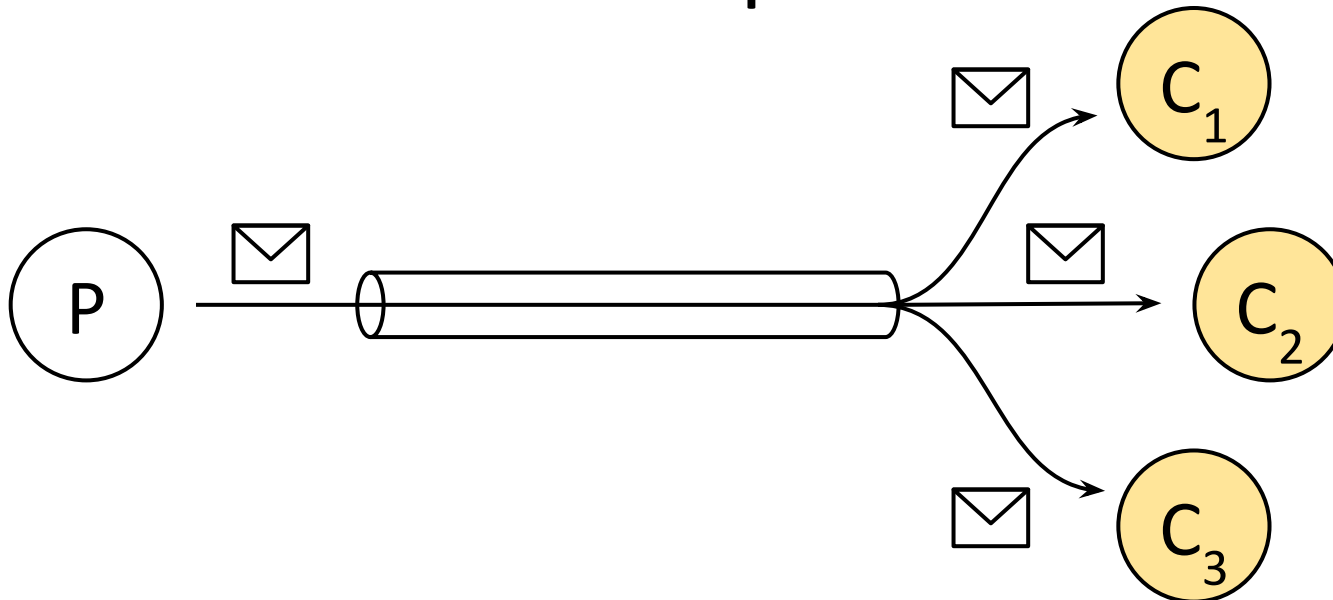
# Modelo de programação



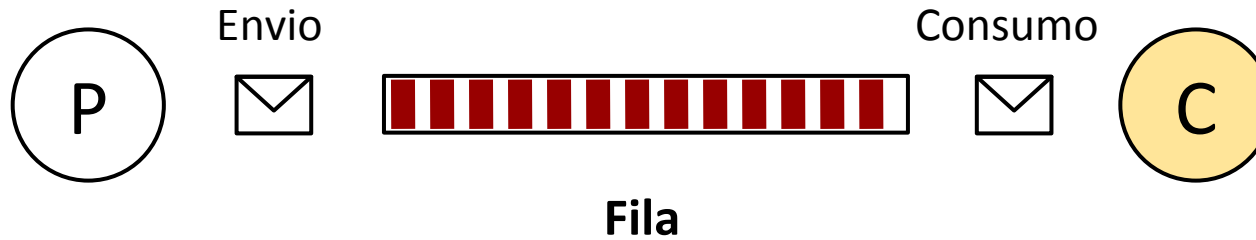
## Ponto a Ponto



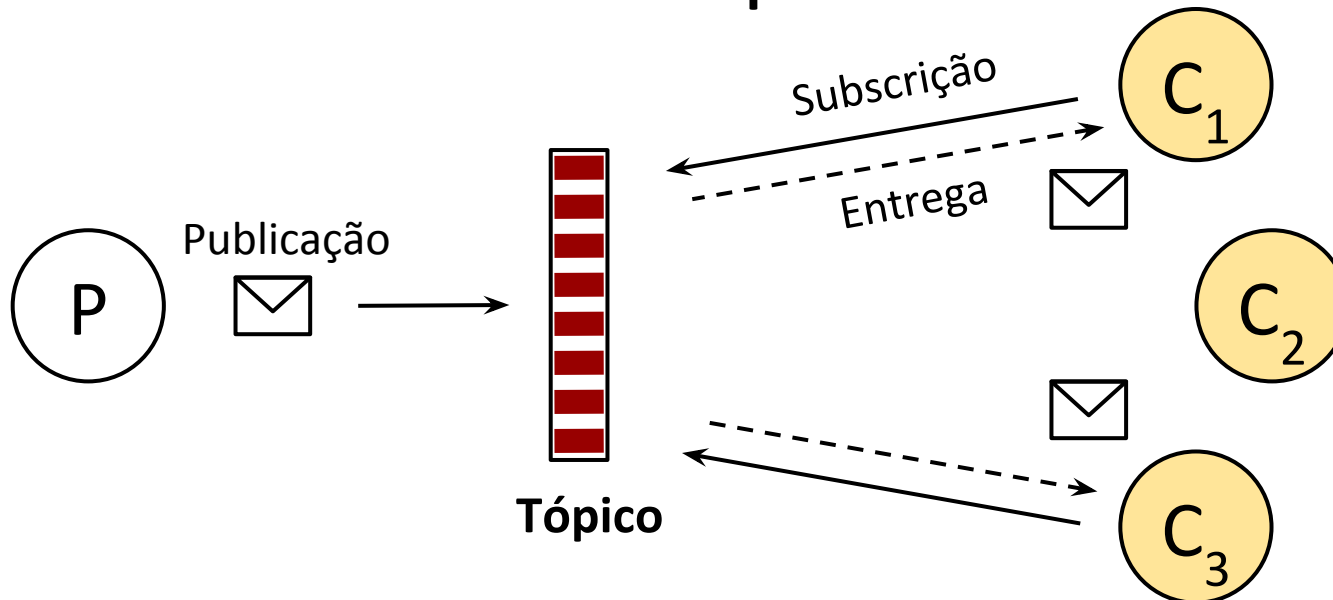
## Ponto a Multiponto



## Ponto a Ponto



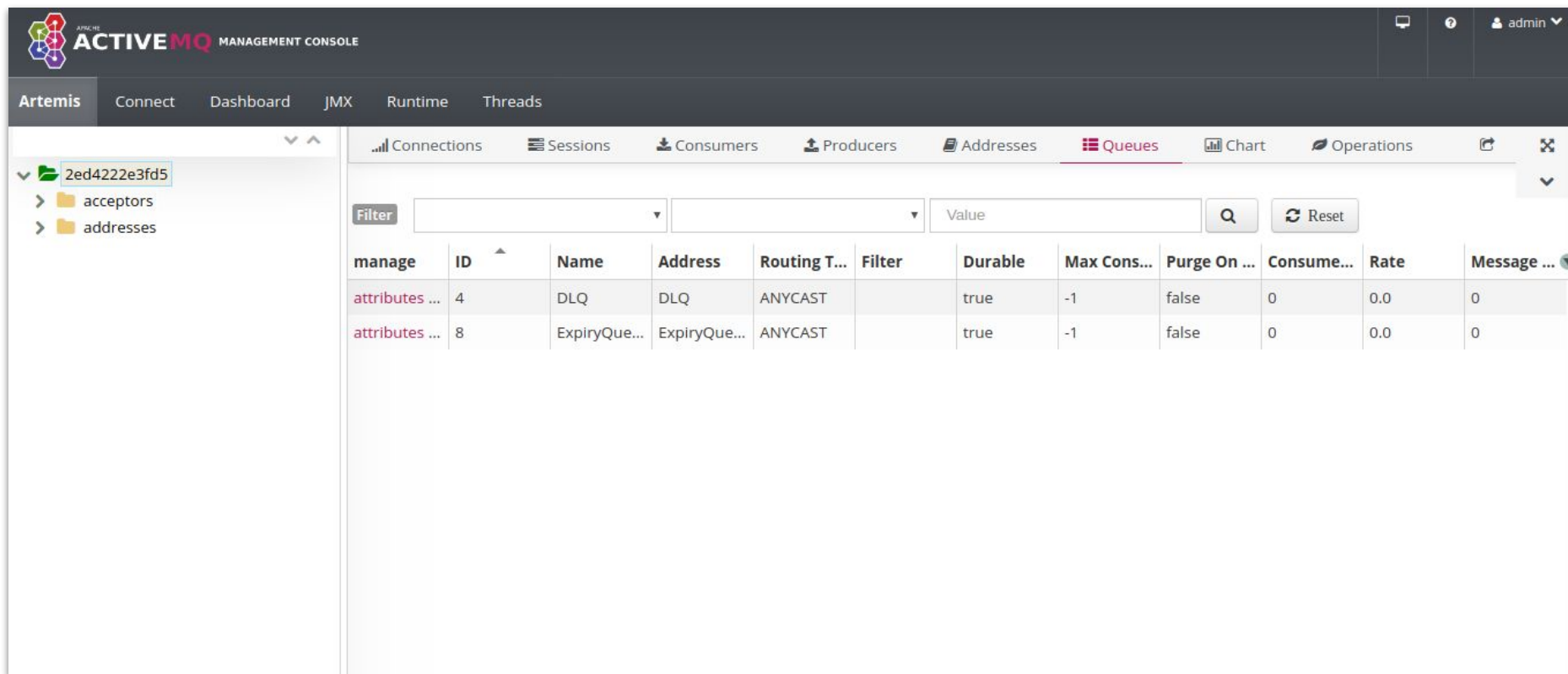
## Ponto a Multiponto





**Criar recursos  
e objetos administrativos**

# Ferramenta administrativa



The screenshot displays the Apache ActiveMQ Management Console interface. The top navigation bar includes the 'Artemis' logo and tabs for 'Connect', 'Dashboard', 'JMX', 'Runtime', and 'Threads'. The main content area is divided into a left sidebar and a right pane. The sidebar shows a tree view with a selected node '2ed4222e3fd5' containing sub-items 'acceptors' and 'addresses'. The right pane features a tabbed interface with 'Queues' selected. Above the table, there are filters and a search bar. The table lists queue details with columns: manage, ID, Name, Address, Routing T..., Filter, Durable, Max Cons..., Purge On ..., Consume..., Rate, and Message ...

manage	ID	Name	Address	Routing T...	Filter	Durable	Max Cons...	Purge On ...	Consume...	Rate	Message ...
attributes ...	4	DLQ	DLQ	ANYCAST		true	-1	false	0	0.0	0
attributes ...	8	ExpiryQue...	ExpiryQue...	ANYCAST		true	-1	false	0	0.0	0

# Arquivos de configuração

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <messaging-deployment xmlns="urn:jboss:messaging-activemq-deployment:1.0">
3   <server name="default">
4     <jms-destinations>
5       <jms-queue name="prd-ent">
6         <entry name="jms/queue/prd-ent"/>
7         <entry name="java:jboss/exported/jms/queue/prd-ent"/>
8       </jms-queue>
9     </jms-destinations>
10  </server>
11 </messaging-deployment>
```

# Código-fonte

```
1 @JMSConnectionFactoryDefinition(name = "java:jms/MyConnFactory")
2 @JMSDestinationDefinition(interfaceName = "Queue", name = "java:jms/MyQueue")
3 @Singleton
4 @Startup
5 public class MyBean {
6     // do something
7 }
```

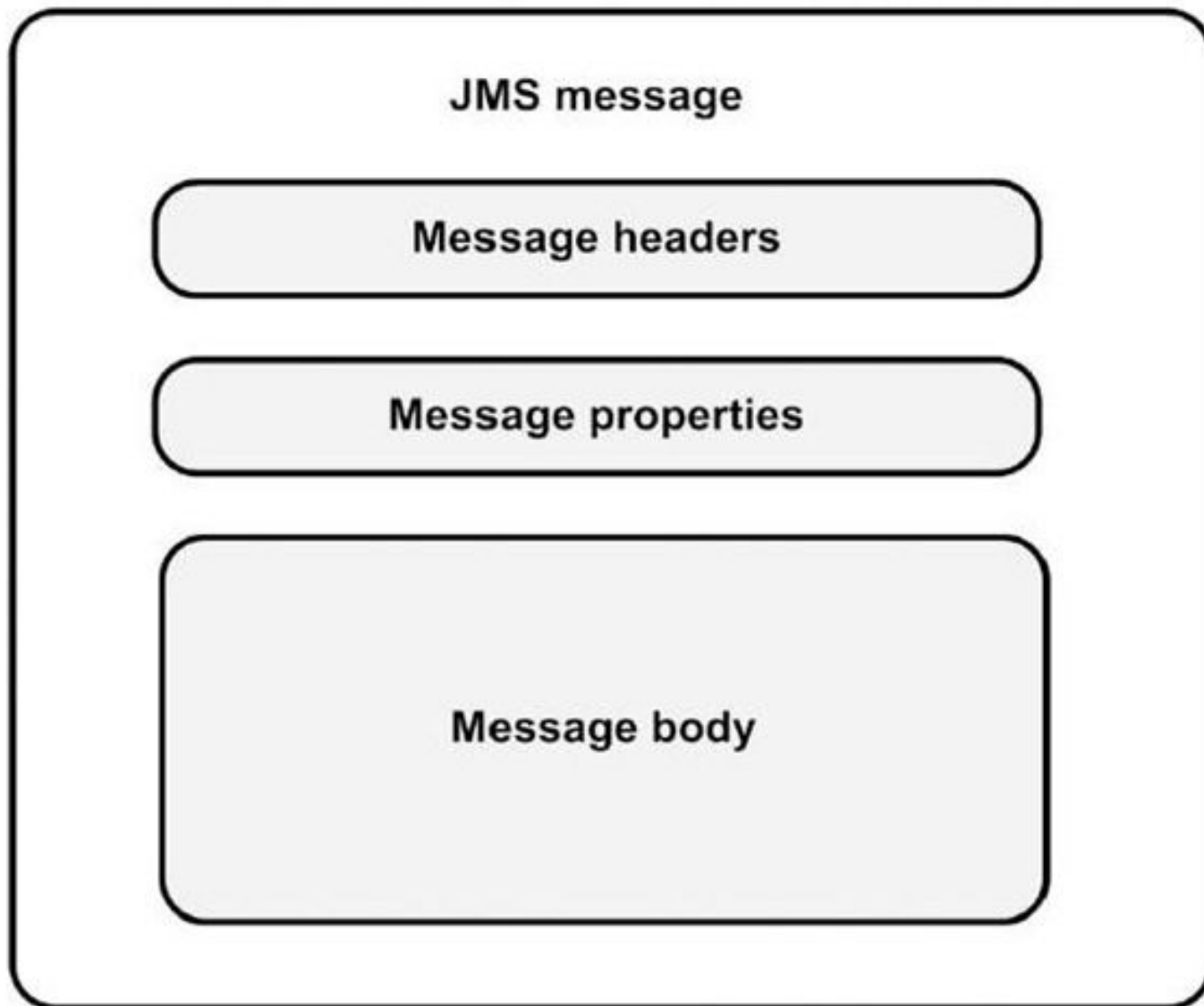
**Lado Produtor**

# Using ConnectionFactory

```
1 public class SimpleMessageClient {
2     @Resource(lookup = "java:comp/ConnectionFactory")
3     private ConnectionFactory connectionFactory;
4     @Resource(lookup = "jms/MyQueue")
5     private Queue queue;
6
7     public static void main(String[] args) {
8         try (JMSContext context = connectionFactory.createContext();) {
9             context.createProducer().send(queue, text);
10             // do something
11         }
12 }
13
```

# Using JMSContext

```
1 public class SimpleMessageClient {
2
3     @Inject
4     @JMSConnectionFactory("jms/MyCF")
5     private JMSContext context;
6
7     @Resource(lookup = "jms/MyQueue")
8     private Queue queue;
9
10    public void sendMessage(String text) {
11        context.createProducer().send(queue, text);
12        // do something
13    }
14 }
15
```





# Tipos de Mensagem

<b>Tipo</b>	<b>Conteúdo</b>
TextMessage	String
MapMessage	Mapa chave - valor
ByteMessage	Array de bytes
ObjectMessage	Objeto serializável
StreamMessage	Stream
Message	Sem conteúdo

# Trabalhando com mensagem

```
TextMessage message = context.createTextMessage();  
message.setText(msg_text); // msg_text is a String  
context.createProducer().send(message);
```

# Atalho

```
String message = "This is a message";  
context.createProducer().send(dest, message);
```

\* Somente válido para *TextMessage*, *BytesMessage*, *MapMessage*, or *ObjectMessage*.

# Importante

- Envio assíncrono é possível
- Altere cabeçalho ou propriedades da mensagem para modificar o comportamento no consumidor
  - Seletores
  - Confirmação de recebimento
  - Maiores detalhes na especificação da API

**Lado Consumidor**

# Método Síncrono

```
1 public class SynchConsumer {
2
3     @Inject
4     @JMSConnectionFactory("java:jms/MyCF")
5     private JMSContext context;
6
7     @Resource(lookup = "jms/MyQueue")
8     private Queue queue;
9
10    public void receiveMsg() {
11        JMSConsumer consumer;
12        // Inicializar outras variaveis
13
14        consumer = context.createConsumer(dest);
15        Message m = consumer.receive(1000);
16
17        if (m instanceof TextMessage) {
18            // Processe
19        }
20    }
21 }
```

# Método Assíncrono

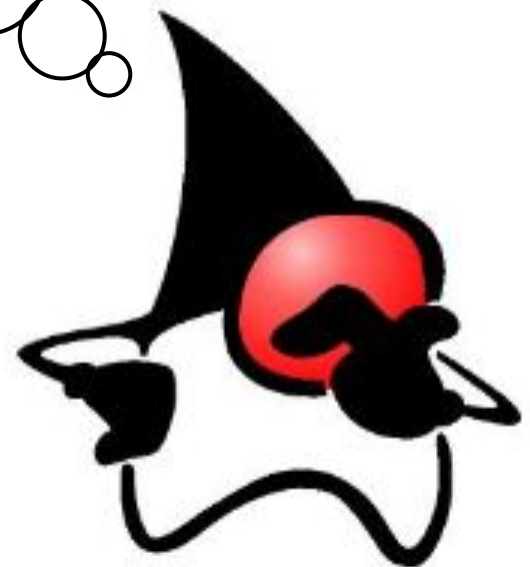
```
1 public class SynchConsumer implements MessageListener {
2     @Inject
3     @JMSConnectionFactory("java:jms/MyCF")
4     private JMSContext context;
5
6     @Resource(lookup = "jms/MyQueue")
7     private Queue queue;
8
9     public void register() {
10         JMSConsumer consumer;
11         consumer = context.createConsumer(dest);
12         consumer.setMessageListener(this);
13     }
14
15     public void onMessage(Message m) {
16         if (m instanceof TextMessage) {
17             String body = m.getBody(String.class);
18             // Do something
19         }
20     }
21 }
```

# Resumindo...

1. Crie os objetos administrativos
2. Injete os objetos no produtor
3. Crie a mensagem e a envie
4. Injete os objetos no consumidor
5. Receba a mensagem
6. Cheque o tipo e faça o processamento



**Existe outra forma de  
processar mensagens na  
plataforma?**



# **Message Driven Beans**

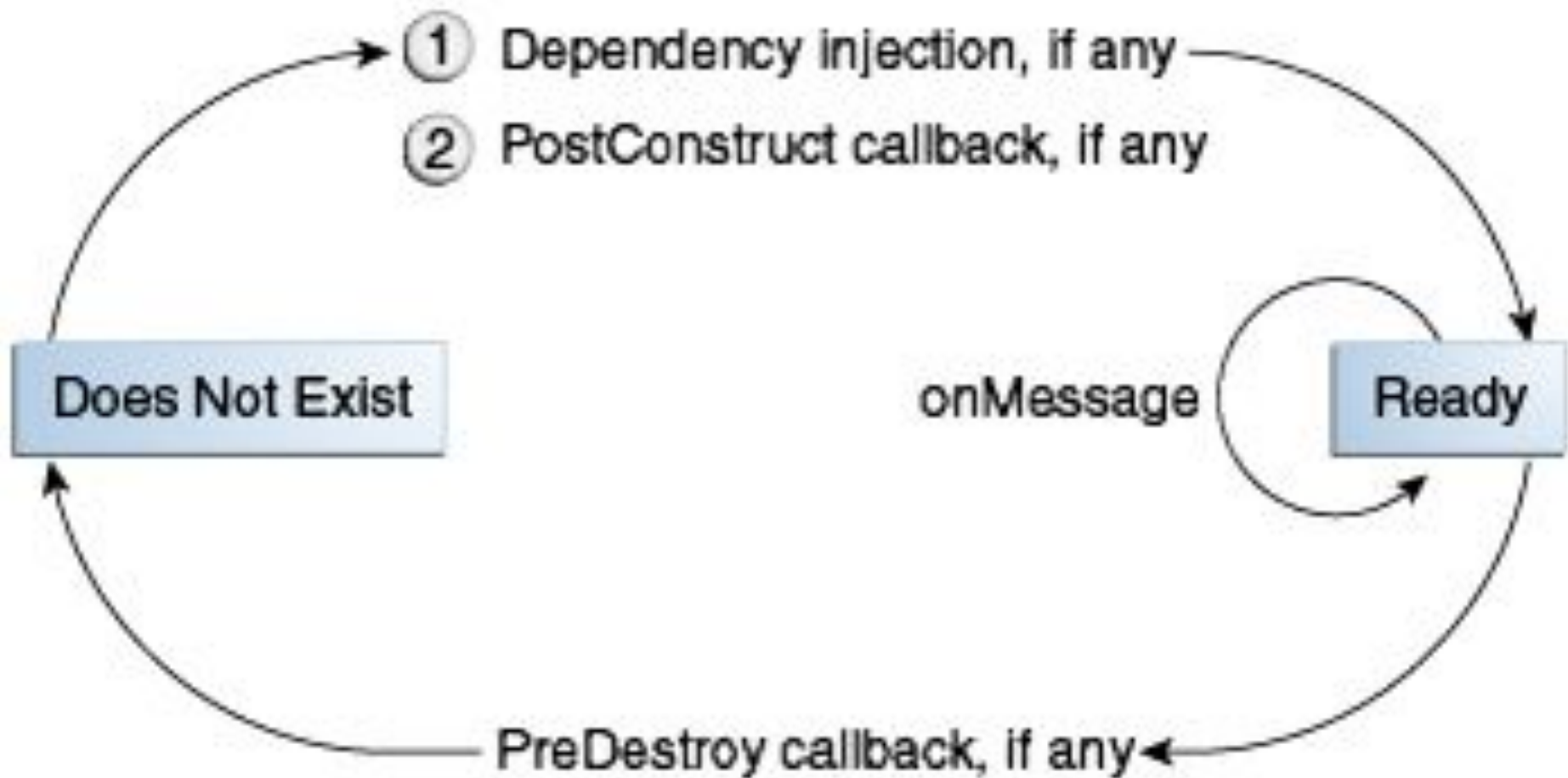


## O que é?

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

# Características

- Assincronia
- Multithreaded
- Sem estado
- Construtor Padrão
- Acionados pelo container
- `MessageListener`



# @MessageDriven

Propriedade	Descrição
ActivationConfig	Propriedades de ativação do Bean
mappedName	Nome global do bean
description	Descrição do Bean
MessageListenerInterface	Interface acionada quando da chegada de uma mensagem
name	Nome atribuído ao Bean

# @ActivationConfigProperty

Propriedade	Descrição
acknowledgeMode	Modo de confirmação de recebimento de mensagem
messageSelector	Especifica filtros para recebimento de mensagem
destinationType	Queue ou Topic
destinationLookup	Nome JMS do recurso

# Código

```
1 @MessageDriven(activationConfig = {
2     @ActivationConfigProperty(propertyName = "destinationLookup",
3         propertyValue = "jms/MyQueue"),
4     @ActivationConfigProperty(propertyName = "destinationType",
5         propertyValue = "javax.jms.Queue")
6 })
7 public class SimpleMessageBean implements MessageListener {
8
9     public void onMessage(Message inMessage) {
10         if (inMessage instanceof TextMessage) {
11             String body = inMessage.getBody(String.class);
12             // Do something
13         }
14 }
```



# **Web Services na Plataforma Java EE**



# O que são?

Operações acionadas remotamente utilizando o protocolo HTTP como base.

# Decisão

## Big Web Services

- JAX-WS
- Contrato Formal
- Troca de documentos XML
- Operação

## Restful

- JAX-RS
- Simplicidade
- JSON
- Recurso

# Big Web Services

- @WebService / @WebMethod
- Classe e métodos públicos
- Migrado para Java SE

# Restful Web Services

- Stateless
- Métodos HTTP (GET, POST, DELETE, etc)
- Identificação de recursos por URI
  - `/resource1/1234`
  - `/user/123/deps`
- Mensagens auto-descritivas (Restful)

# @Path

- Associa o caminho ao recurso que a classe representa
  - `@Path("/usuario")`
- Pode incluir variáveis
  - `@Path("/usuario/{uid}")`
  - `@Path("users/{username: [a-zA-Z][a-zA-Z_0-9]*}")`
- `@PathParam`
  - `@PathParam("uid") String uid`

# Métodos HTTP

- @GET, @POST, @HEAD, @DELETE, etc
- Indica o método HTTP que o método da classe irá processar

# Tipos MIME

- **@Produces**
  - `@Produces("text/plain")`
  - `@Produces({"application/XML", "application/json" })`
- **@Consumes**
  - `@Consumes("text/plain")`
  - `@Consumes({"application/XML", "application/json" })`



# Extraindo Info do Request

- @QueryParam

```
1 @Path("smooth")
2 @GET
3 public Response smooth(
4     @DefaultValue("2") @QueryParam("step") int step,
5     @DefaultValue("true") @QueryParam("min-m") boolean hasMin,
6     @DefaultValue("true") @QueryParam("max-m") boolean hasMax,
7     @DefaultValue("true") @QueryParam("last-m") boolean hasLast,
8     @DefaultValue("blue") @QueryParam("min-color") ColorParam minColor,
9     @DefaultValue("green") @QueryParam("max-color") ColorParam maxColor,
10    @DefaultValue("red") @QueryParam("last-color") ColorParam lastColor
11    ) { ... }
```

# Extraindo Info do Request

- **@PathParam** e **@QueryParam**
  - Todos tipos primitivos e Wrappers exceto Char
  - Qualquer classe que aceite uma String no construtor
  - Qualquer classe com método `valueOf(String)`
  - `List<T>`, `Set<T>`, `SortedSet<T>` para os casos acima

# Configurando a aplicação (1)

- Extenda a classe `javax.ws.rs.core.Application`
- Use a anotação `@ApplicationPath`

```
1 @ApplicationPath("/webapi")  
2 public class MyApplication extends Application { ... }
```

## Configurando a aplicação (2)

- Configure o mapeamento no web.xml

```
1 <servlet-mapping>
2   <servlet-name>javax.ws.rs.core.Application</servlet-name>
3   <url-pattern>/webapi/*</url-pattern>
4 </servlet-mapping>
```

# Exercício 03



## Exercício 03

Crie um serviço rest que receba um payload qualquer via post e o repasse para uma mdb utilizando uma fila

## Exercício 03

Dicas:

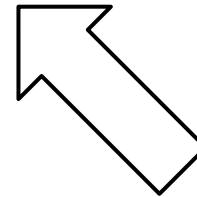
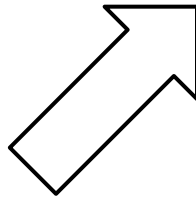
- Mude o runtime do server para utilizar a configuração full (standalone-full.xml)
- Crie a fila antes de realizar o deploy ou então use arquivos que realizem a configuração da fila automaticamente (ver ex03 no repositório)

# Persistência na Plataforma Java EE





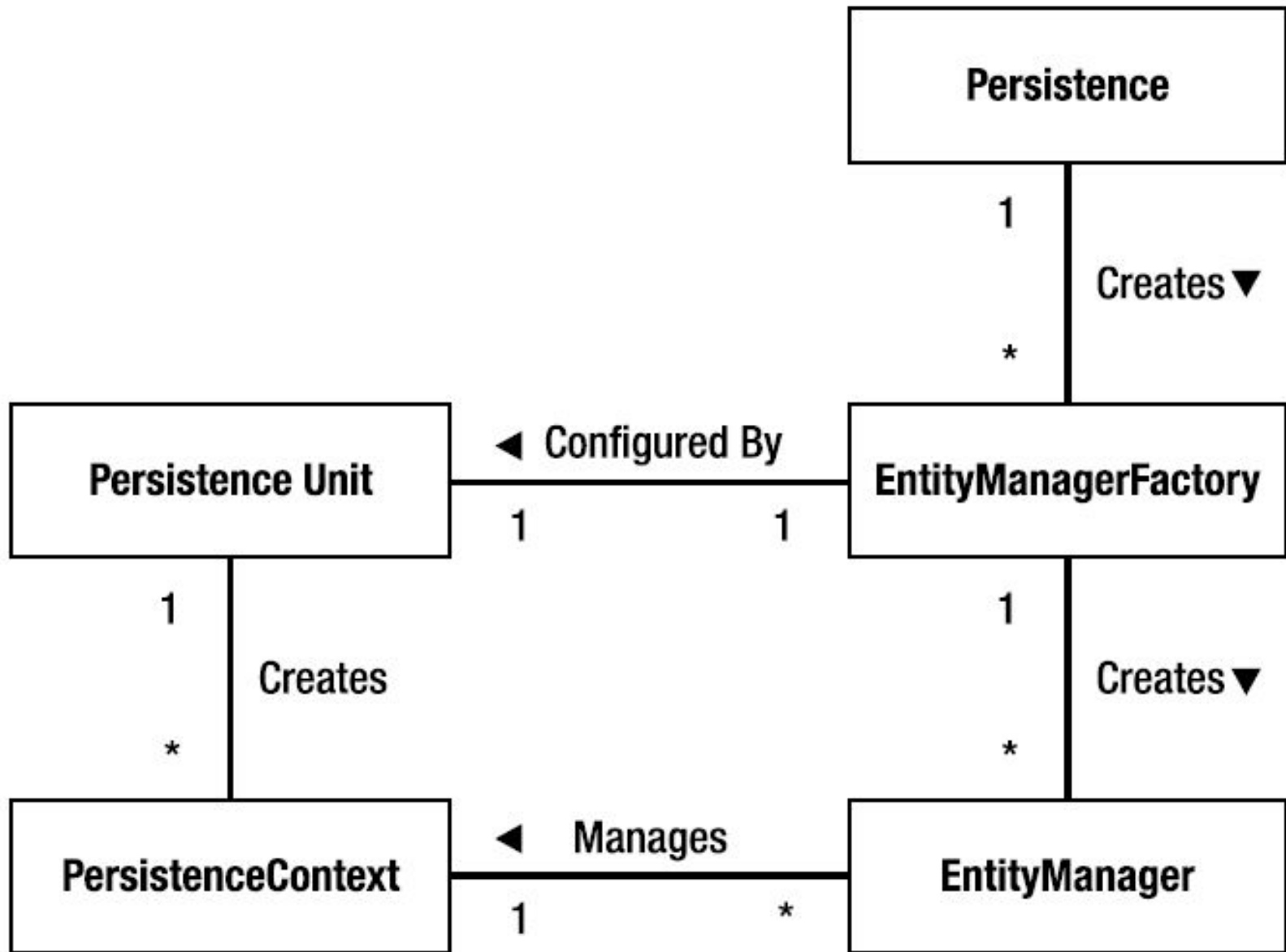
**JPA**



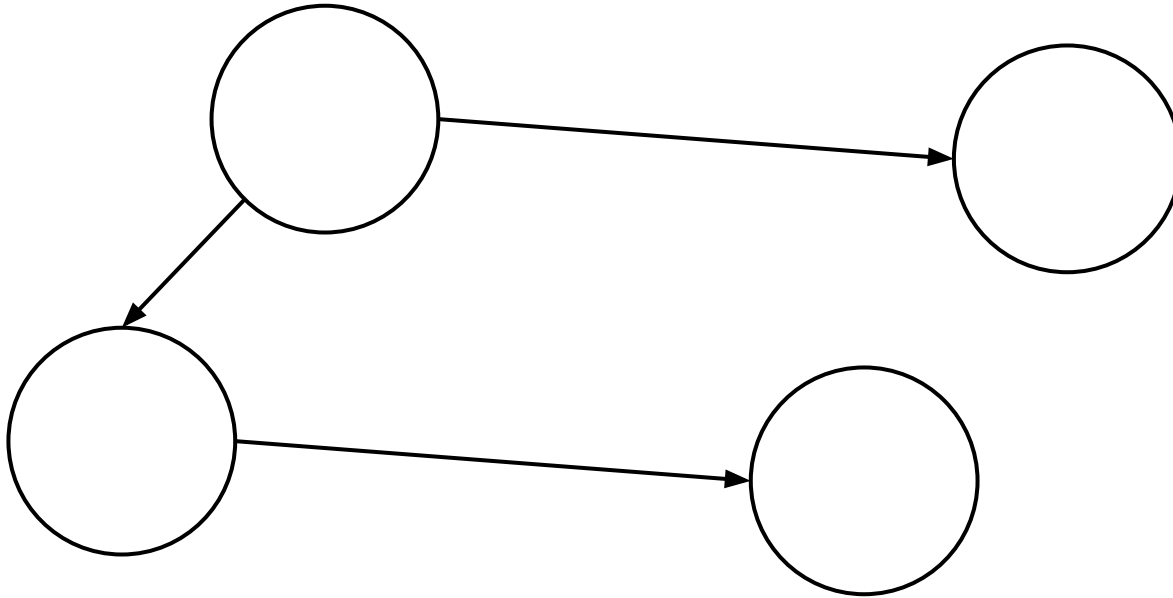
**Hibernate**



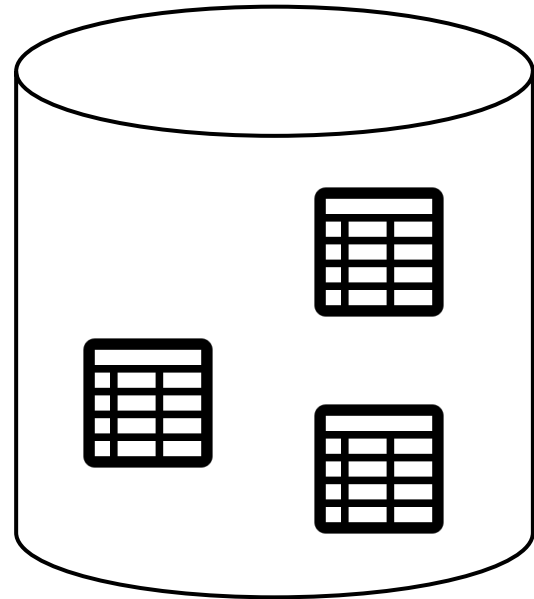
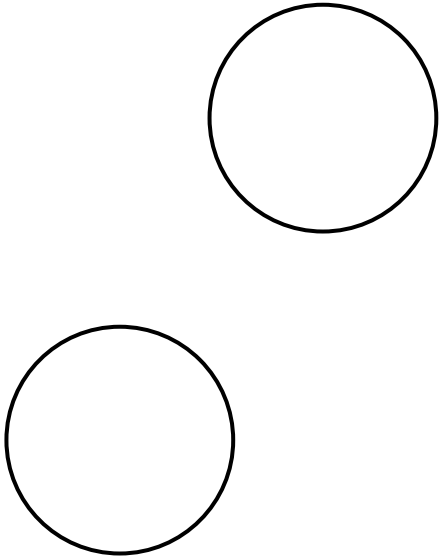
**EclipseLink**



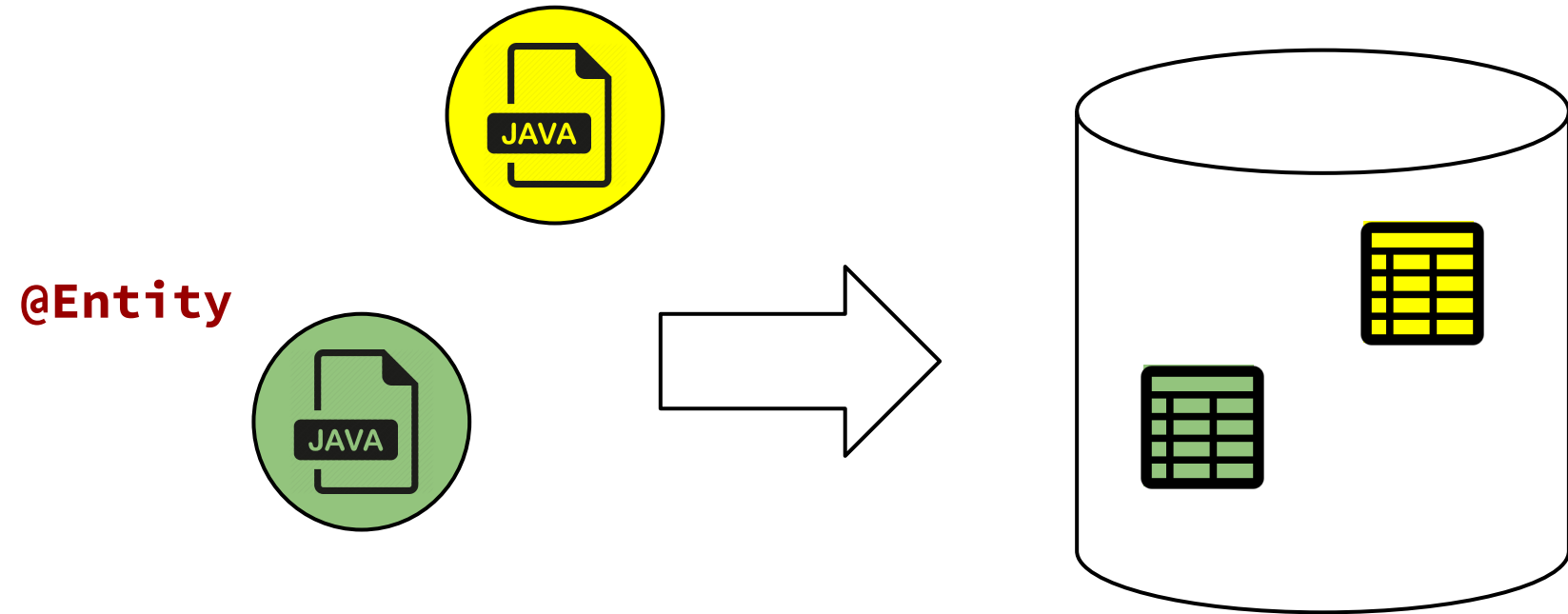
# Entity



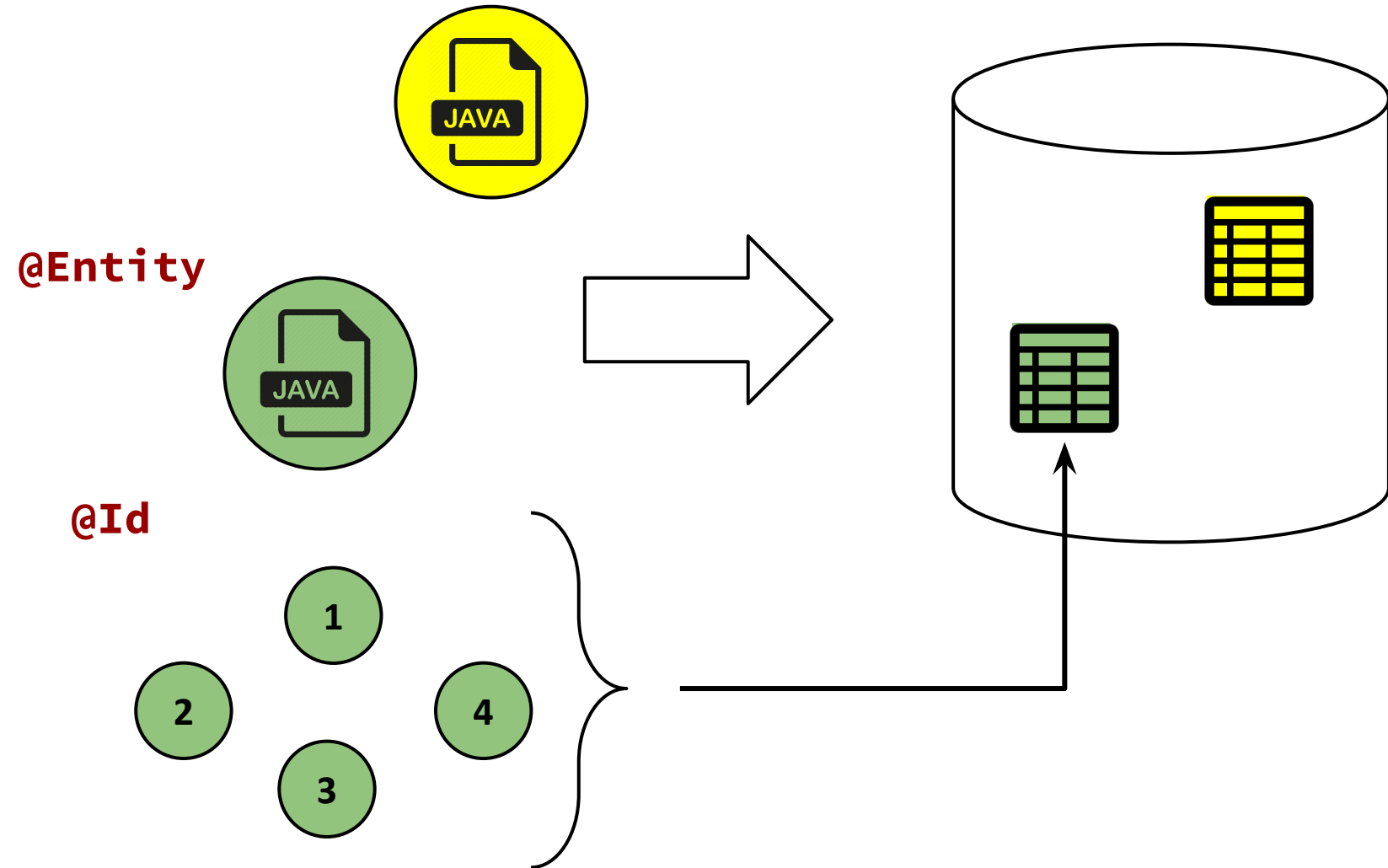
# Entity



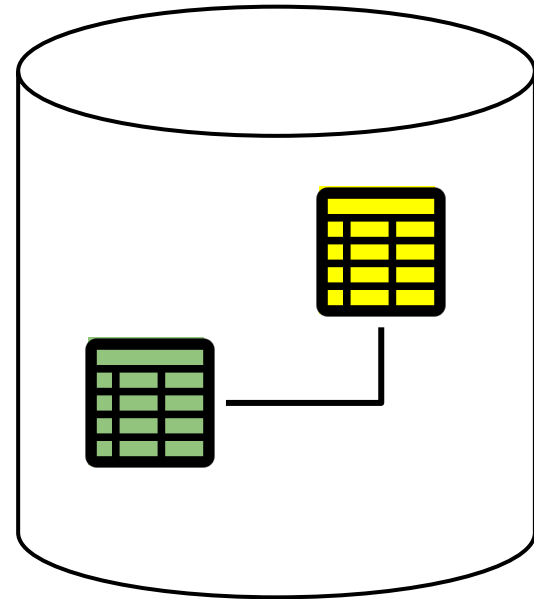
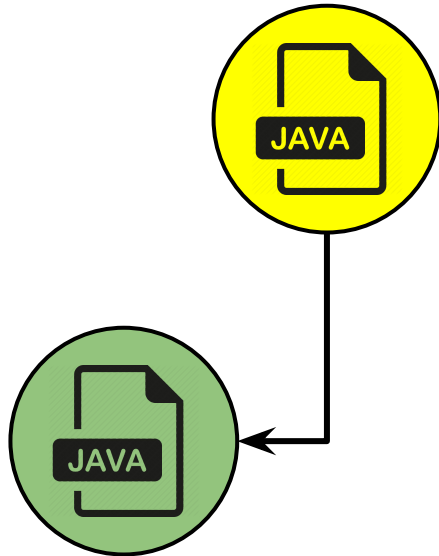
# Entity



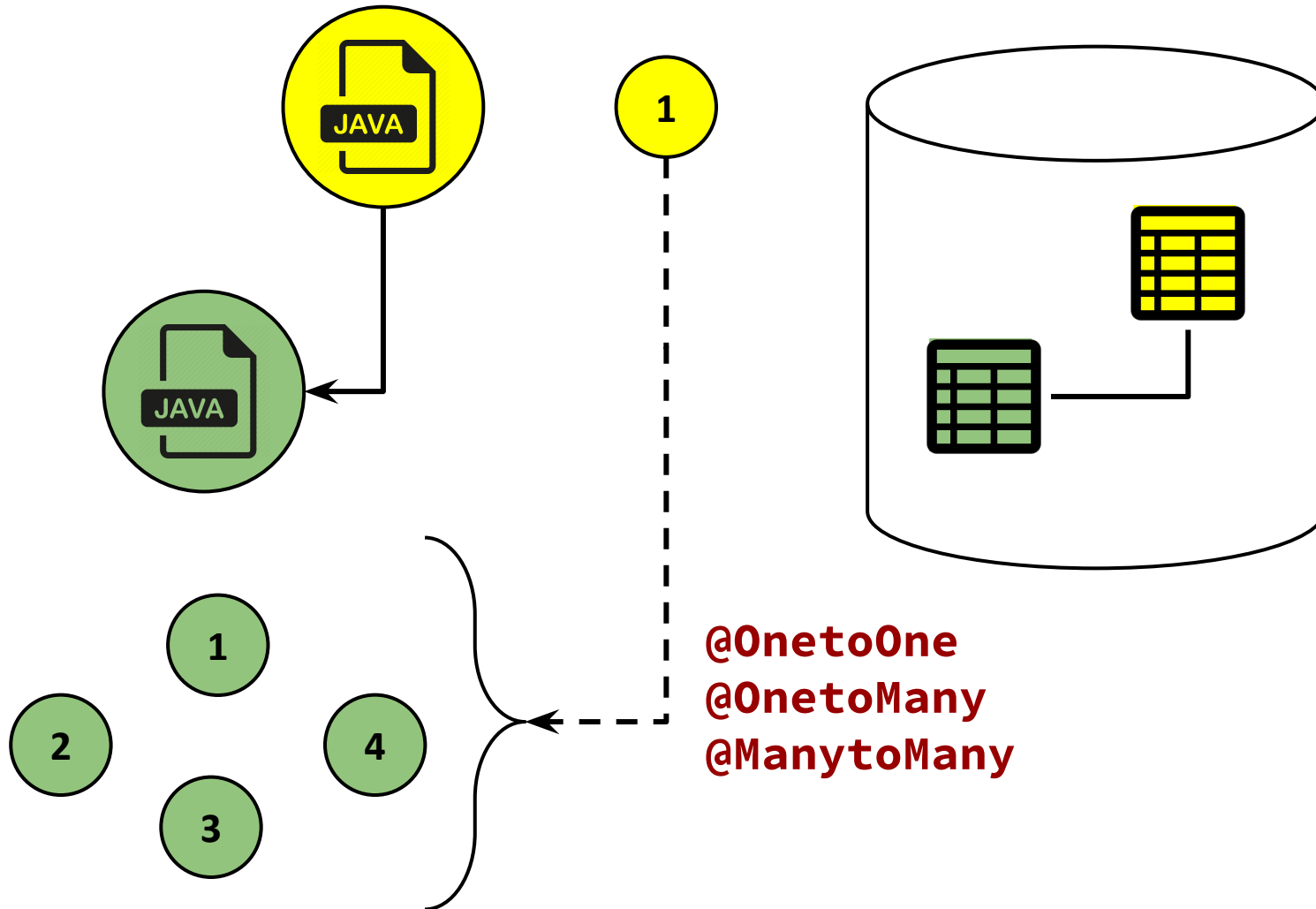
# Entity



# Entity



# Entity





# Entity

```
1 @Entity
2 public class Fornecedor implements Serializable {
3     @Id
4     private Integer id;
5     private String nome;
6
7     @ManyToOne
8     @JoinColumn(name="id_situacao")
9     private SituacaoFornecedor situacaoFornecedor;
10
11     public Integer getId() {...}
12     public void setId(Integer id) {...}
13 }
```

# Importante

- Configuráveis por anotação ou metadados
  - persistence.xml
- Suporte a herança e polimorfismo
- Persistent fields vs Persistent properties

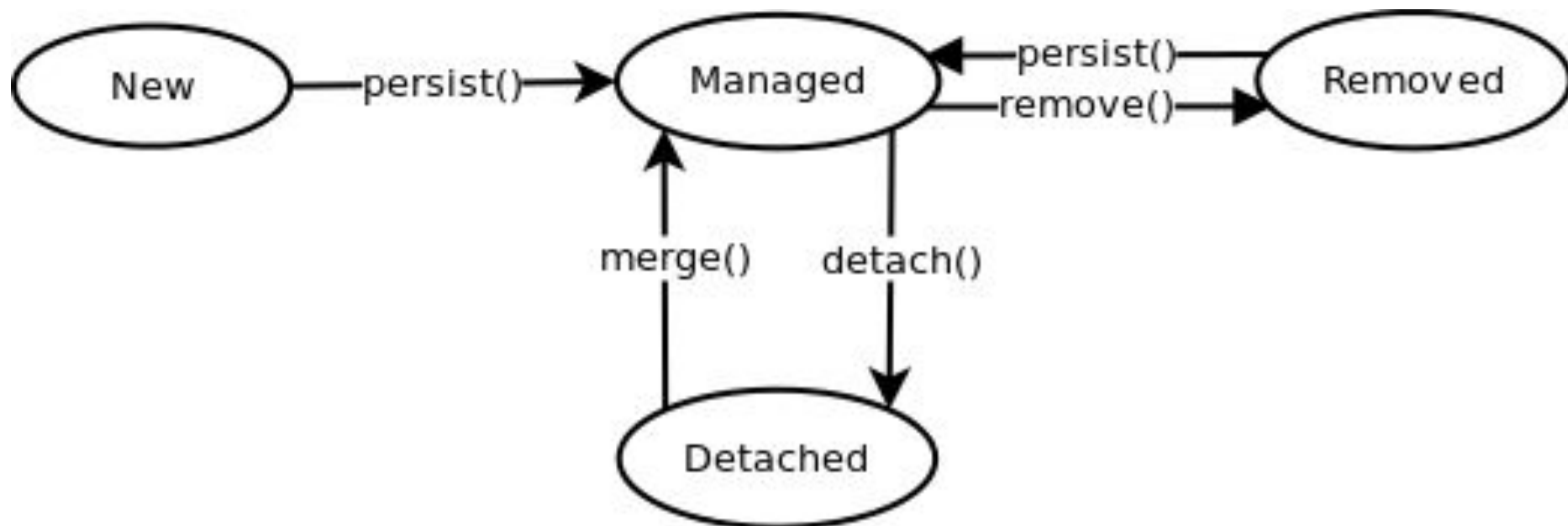
# Principais anotações

- @Entity
- @Table
- @Column
- @Id
- Mapeamentos
  - @OneToOne
  - @OneToMany
  - @ManyToMany
- Injeção
  - @PersistenceUnit
  - @PersistenceContext
- @NamedQuery

# Entity Manager

- Gerencia as entidades no Contexto de Persistência
- Oferece uma API para inserção, deleção, atualização e busca de entidades
- Operações
  - persist
  - detach
  - merge
  - remove
  - find

# Ciclo de vida



# Contexto

- Gerenciado pelo Container
  - Contexto propagado por toda a aplicação
  - @PersistenceContext

```
1 @Stateless
2 public class DistribuidoraAS {
3     ...
4     @PersistenceContext
5     EntityManager em;
6 }
```

# Contexto

- Gerenciado pela aplicação
  - Construído e destruído pelo desenvolvedor
  - @PersistenceUnit

```
1 @Stateless
2 public class DistribuidoraAS {
3     ...
4     @PersistenceUnit
5     EntityManagerFactory ef;
6     ...
7     EntityManager em = ef.createEntityManager();
8 }
```

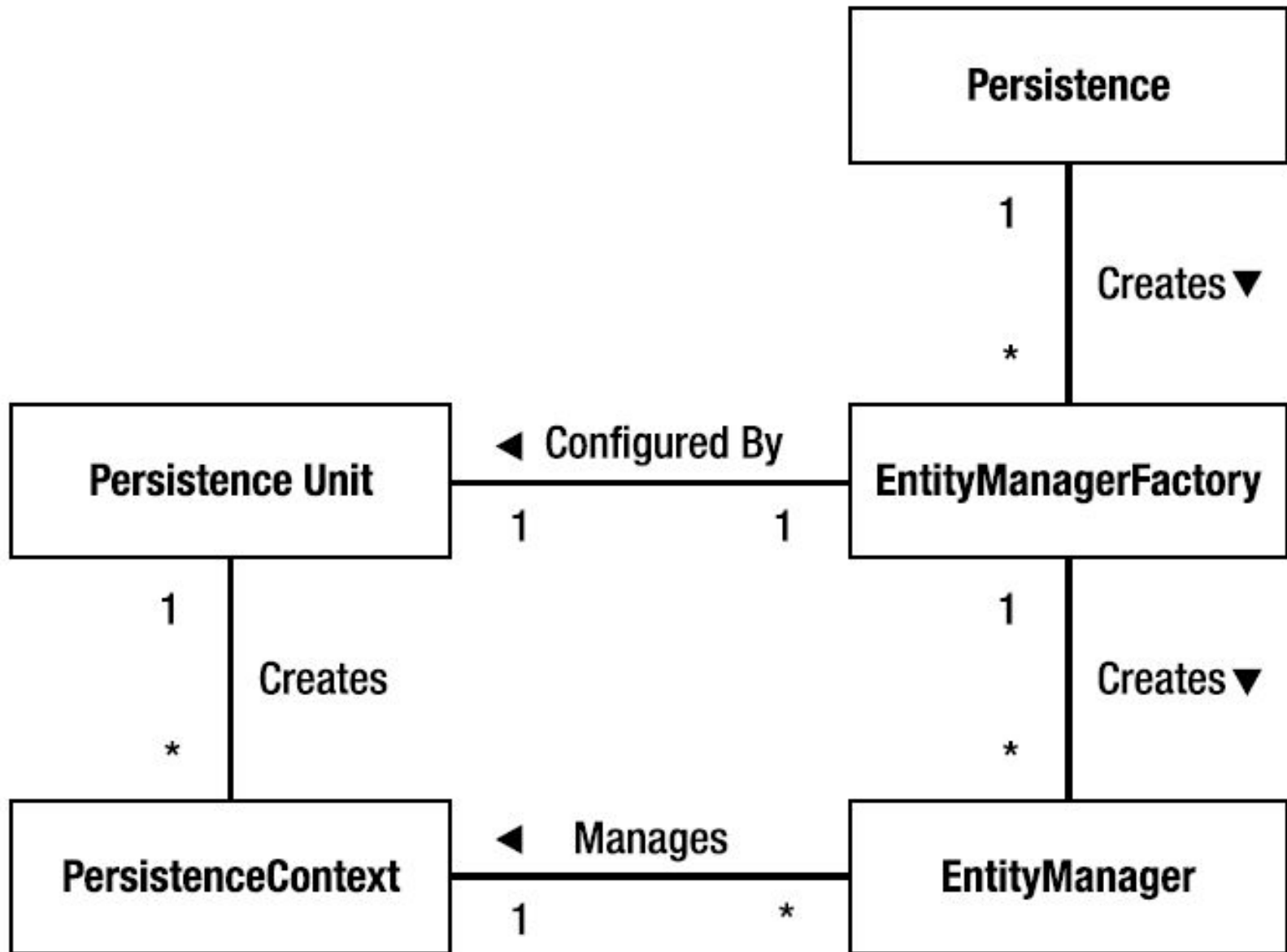
# Persistence Unit

- Define as entidades gerenciadas pelo Entity Manager
- Definido no persistence.xml
- Empacotado no WAR ou EJB-JAR



# Persistence Unit

```
1 <persistence-unit name="DistribuidoraEJB">
2   <jta-data-source>java:/DistribuidoraDS</jta-data-source>
3   <class>fa7.distribuidora.persistence.Estoque</class>
4   <class>fa7.distribuidora.persistence.Fornecedor</class>
5   <class>fa7.distribuidora.persistence.Mercadoria</class>
6   <class>fa7.distribuidora.persistence.Reserva</class>
7   <class>fa7.distribuidora.persistence.SituacaoFornecedor</class>
8 </persistence-unit>
```



# Consultas

# JPQL

- Evolução do EJBQL
- Consultas operam em objetos
- Permite junções, projeções, etc
- Suporte parâmetros posicionais e nomeados
  - ?1 e :param
  - setParameter(posicao, valor)
  - setParameter(nome, valor)

# Consulta Dinâmica

- Consulta editada e encaminhadas para o método `createQuery` do `EntityManager`

```
1 public long queryEmpSalary(String deptName, String empName) {  
2     String query = "SELECT e.salary " +  
3     "FROM Employee e " +  
4     "WHERE e.department.name = '" + deptName +  
5     "' AND " + "e.name = '" + empName + "'";  
6     return em.createQuery(query, Long.class).getSingleResult();  
7 }
```

# Consulta Nomeada

- Consulta associada a um nome e localizada na definição de uma classe \ entidade da aplicação
  - @NamedQuery
  - @NamedQueries

```
1 @Entity
2 @NamedQueries({
3     @NamedQuery(name="Employee.findAll",
4         query="SELECT e FROM Employee e"),
5     @NamedQuery(name="Employee.findByName",
6         query="SELECT e FROM Employee e WHERE e.name = :name")
7 })
8 public Employee {...}
```

```
1 public class EmployeeService {
2     @PersistenceContext
3     EntityManager em;
4     ...
5     public Employee findEmployeeByName(String name) {
6         return em.createNamedQuery("Employee.findByName",
7             Employee.class).setParameter("name", name).getSingleResult();
8     }
9 }
```

# JPQL

- Resultados
  - Tipos básicos
  - Entidades da aplicação
  - Array de objetos
  - Tipos definidos pelo usuário

# Array de Objetos

```
1 public void displayProjectEmployees(String projectName) {
2     List result = em.createQuery(
3         "SELECT e.name, e.department.name " +
4         "FROM Project p JOIN p.employees e " +
5         "WHERE p.name = ?1 " +
6         "ORDER BY e.name")
7     .setParameter(1, projectName)
8     .getResultList();
9     int count = 0;
10    for (Iterator i = result.iterator(); i.hasNext();) {
11        Object[] values = (Object[]) i.next();
12        System.out.println(++count + ": " +
13            values[0] + ", " + values[1]);
14    }
```



# Tipos definidos por usuário

```
1 public void displayProjectEmployees(String projectName) {
2     List<EmpMenu> result =
3         em.createQuery("SELECT NEW example.EmpMenu(" +
4             "e.name, e.department.name) " +
5             "FROM Project p JOIN p.employees e " +
6             "WHERE p.name = ?1 " +
7             "ORDER BY e.name", EmpMenu.class)
8         .setParameter(1, projectName)
9         .getResultList();
10    for (EmpMenu menu : result) {
11        System.out.println(menu.getEmployeeName() + ", " +
12            menu.getDepartmentName());
13    }
14 }
```

# Debate

DAO x Entity Controller x Entity Manager

# Java Transaction API



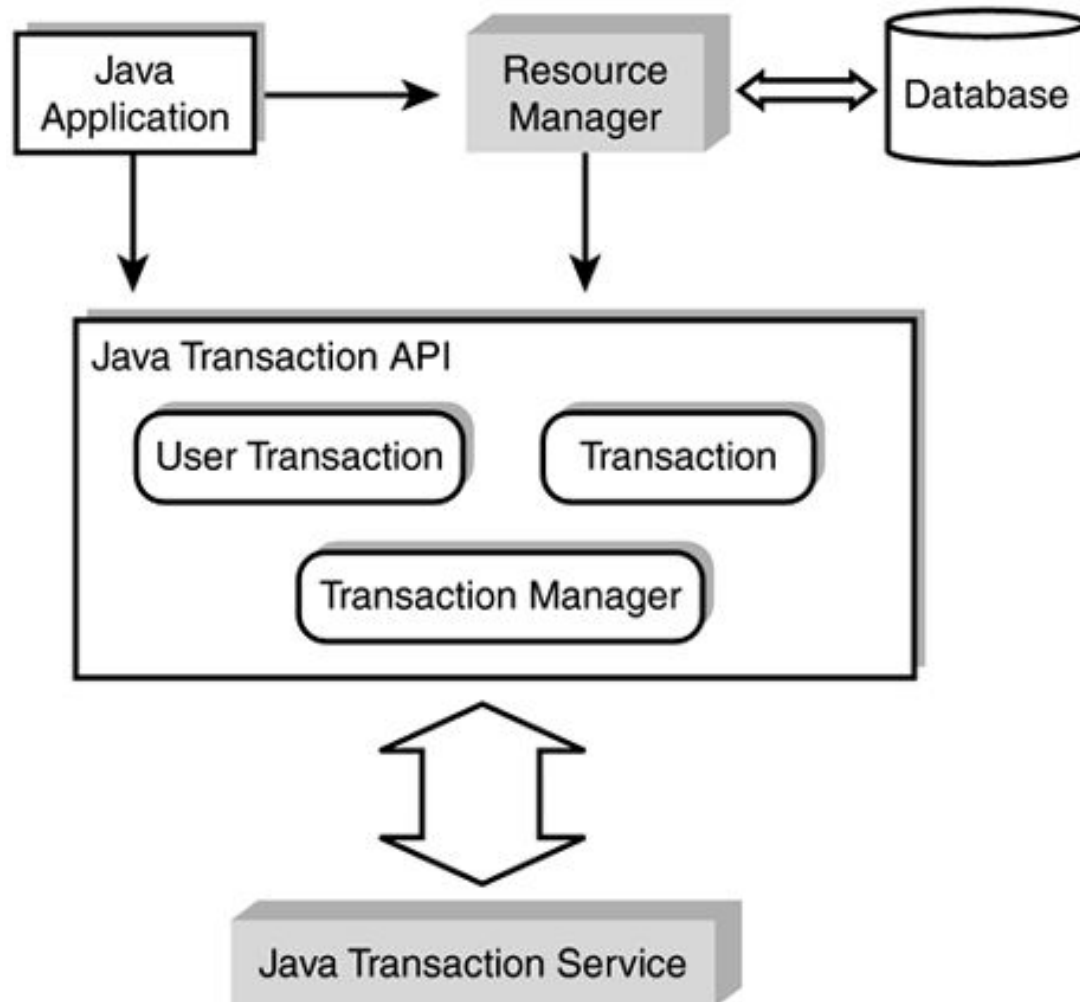
# Transação

Contexto que demarca um conjunto de ações que devem ser executadas de forma atômica levando todos os recursos envolvidos a um estado consistente. Do contrário, essas ações são revertidas.

# JTA

- API de acesso às transações independente de implementações
- Permite o controle de transações distribuídas pelo Container ou Aplicação

# JTA



# Container Managed Transaction

- Demarcadas por anotações em classes ou métodos.
- `@TransactionAttribute`
  - Required (Padrão)
  - RequiresNew
  - Mandatory
  - NotSupported
  - Supports
  - Never

# Container Managed Transaction

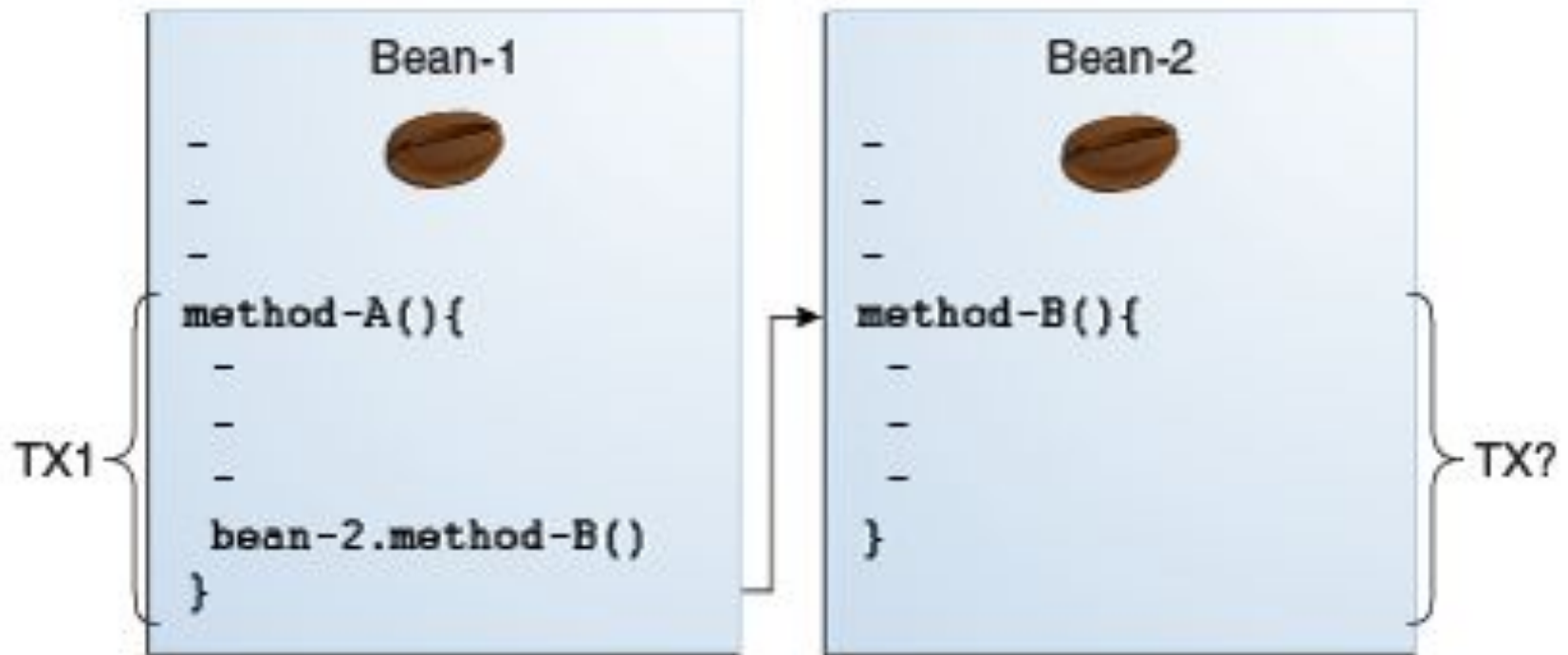
- Rollback
  - Exceção de sistemas
  - `EJBContext.setRollbackOnly`
  - `@ApplicationException(rollback=true)`



# Container Managed Transaction

```
1 @TransactionAttribute(NOT_SUPPORTED)
2 @Stateful
3 public class TransactionBean implements Transaction {
4 ...
5     @TransactionAttribute(REQUIRES_NEW)
6     public void firstMethod() {...}
7
8     @TransactionAttribute(REQUIRED)
9     public void secondMethod() {...}
10
11     public void thirdMethod() {...}
12
13     public void fourthMethod() {...}
14 }
```

# Container Managed Transaction



<b>Atributo</b>	<b>Cliente (B1)</b>	<b>Negócio (B2)</b>
Required	Nenhuma	T2
Required	T1	T1
RequiresNew	Nenhuma	T2
RequiresNew	T1	T2
Mandatory	Nenhuma	Erro
Mandatory	T1	T1
NotSupported	Nenhuma	Nenhuma
NotSupported	T1	Nenhuma
Supports	Nenhuma	Nenhuma
Supports	T1	T1
Never	Nenhuma	Nenhuma
Never	T1	Erro

# Bean Managed Transaction (BMT)

- Bean controla toda a transação
- Desenvolvedor deve obter a UserTransaction
  - Begin
  - Commit
  - Rollback

# Bean Managed Transaction (BMT)

```
1 @Stateless
2 public class ExampleBean {
3     @Resource
4     private UserTransaction utx;
5     ...
6     public void executaAlgo() {
7         // Inicia a transacao
8         utx.begin();
9         ...
10        // Acao
11        utx.commit();
12    }
13 }
```

```
1 @Stateless
2 public class ExampleBean {
3     @Resource
4     private SessionContext ctx;
5     ...
6     public void executaAlgo() {
7         UserTransaction utx = ctx.getUserTransaction();
8         utx.begin();
9         // Fazendo algo
10        utx.commit();
11    }
12 }
```

# Dicas

- Use CMT quando possível
- Pode fazer sentido utilizar JDBC em cenários específicos
- Stateless Session Bean não pode concluir o método sem realizar commit ou rollback