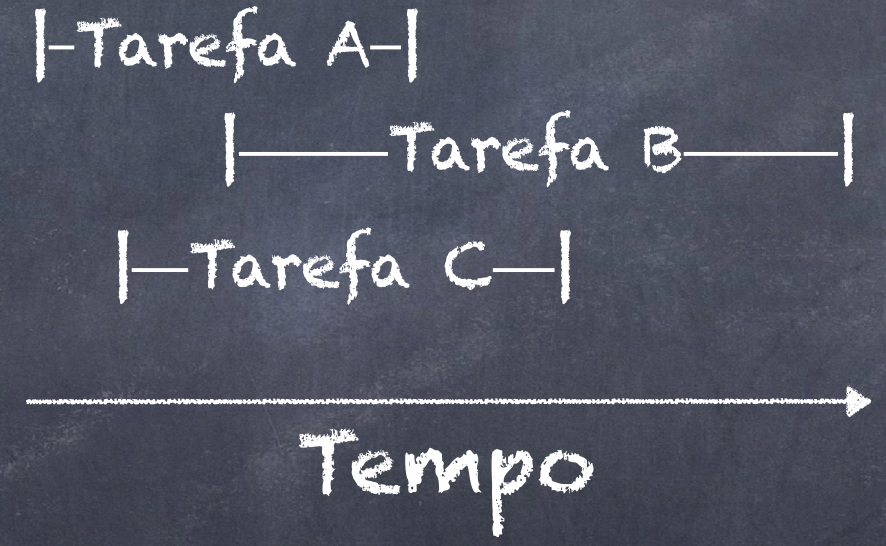
Processamento Assíncrono

Síncrono

Assíncrono



Como é feito?

Padão Java EE

=

Síncrono

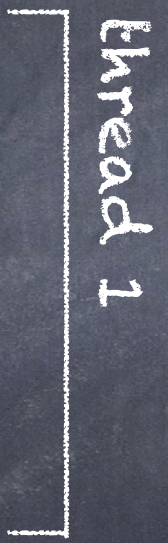
1. Métodos Assíncronos

2. JMS

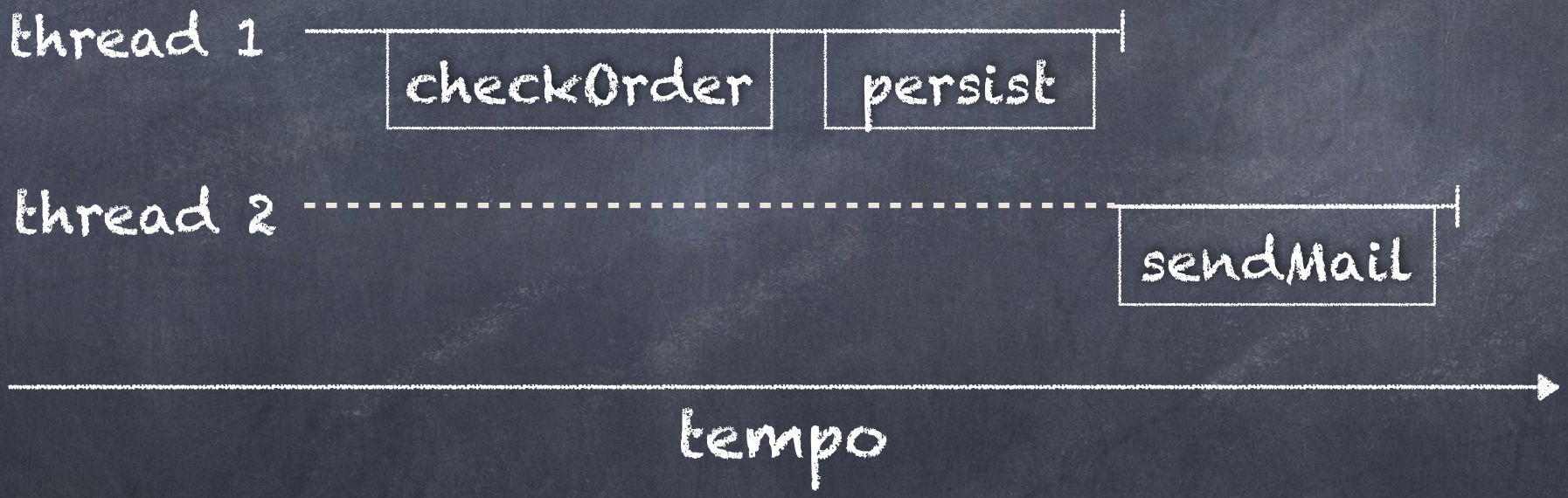
OutOfMemoryError

```
@Stateless
@LocalBean
public class MailService {

    @Asynchronous
    public void sendMail(String from, String to,

                         String subject, String text){

    }

}
```

```
@EJB
private MailService mailService;
...
public void processOrder(Order order) {
        if (checkOrder(order)) {
                persist(order);
                mailService.sendMail(...);
        }
}
...
```



Anatomia da Execução



Retornando objetos em métodos assíncronos

```
@Stateless
@LocalBean
public class CorporateDataService {

    @Asynchronous
    public Future<List<City>> findAllCities(Long stateId){

        List<City> cities = webService.find(...); //network

        return new AsyncResult (cities);

    }

}
```

```
@EJB

private CorporateDataService corpService;


public void longRunning (...) {

      Future<List<City>> asyncResult =

                  corpService.findAllCities(...);

      ...

      ...

      List<City> cities = asyncResult.get(); //pode esperar até terminar

      ...

}
```

```
@Resource

private SessionContext context;

@Asynchronous
public Future<Long> import (List<Employee> list){

      for (Employee emp: list){

            if(!context.wasCancelCalled()) {

                  persist(emp);

            }

            ...

} } }
```

Cancelando uma chamada assíncrona

```
Future<Long> asyncResult = employeeService.import(...);

...

...

if(!asyncResult.isDone()){

    asyncResult.cancel(true);

}

...
```

Exercícios (6)

- Realize a importação assíncrona do arquivo 'employee.xml'
- Este arquivo contém um lista de mapas cujas chaves são os nomes dos atributos de Employee
- Exemplo.:

```
Map itemMap = list.get(0);

String name = itemMap.get("name");

String dayOfBirth = itemMap.get("dayOfBirth");
```

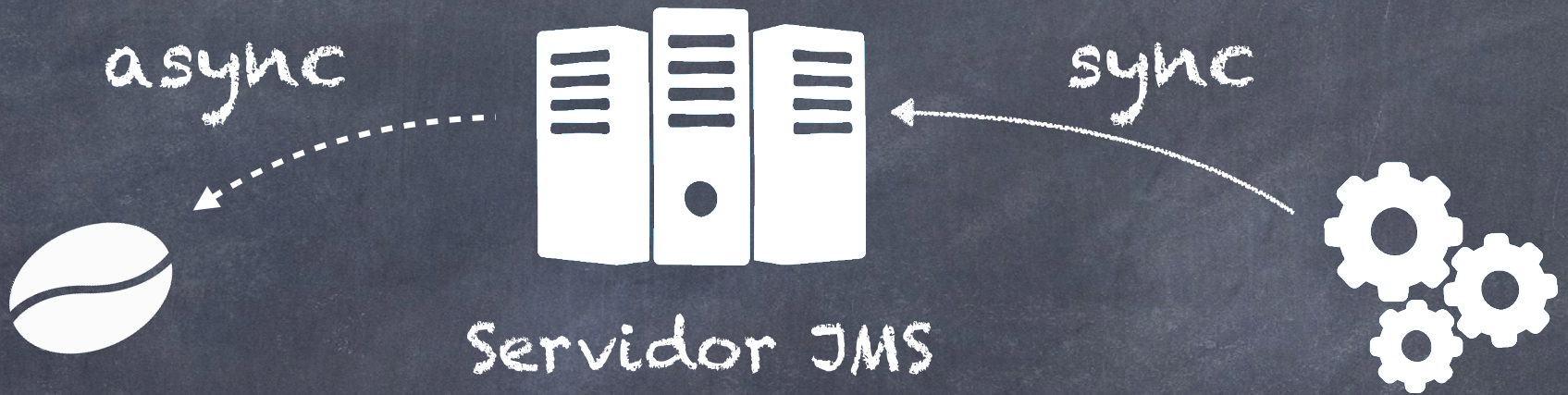
- Crie um método na interface EmployeeService
  - public void importItems(List<Map<String, Object>> items);
- Carregue o arquivo no TestCase
  - new XStream().fromXML(new File("employees.xml"));
  - Um help: new SimpleDateFormat("dd/MM/yyyy").parse(textDate);

Exercícios (6.cont)

- Os dados devem ser persistidos usando a entidade Employee:
  - Long id
  - String name
  - Date dayOfBirth
  - BigDecimal salary

MessageDriven Beans

Arquitetura JMS



Consumidor

MessageDriven Bean

Produtor

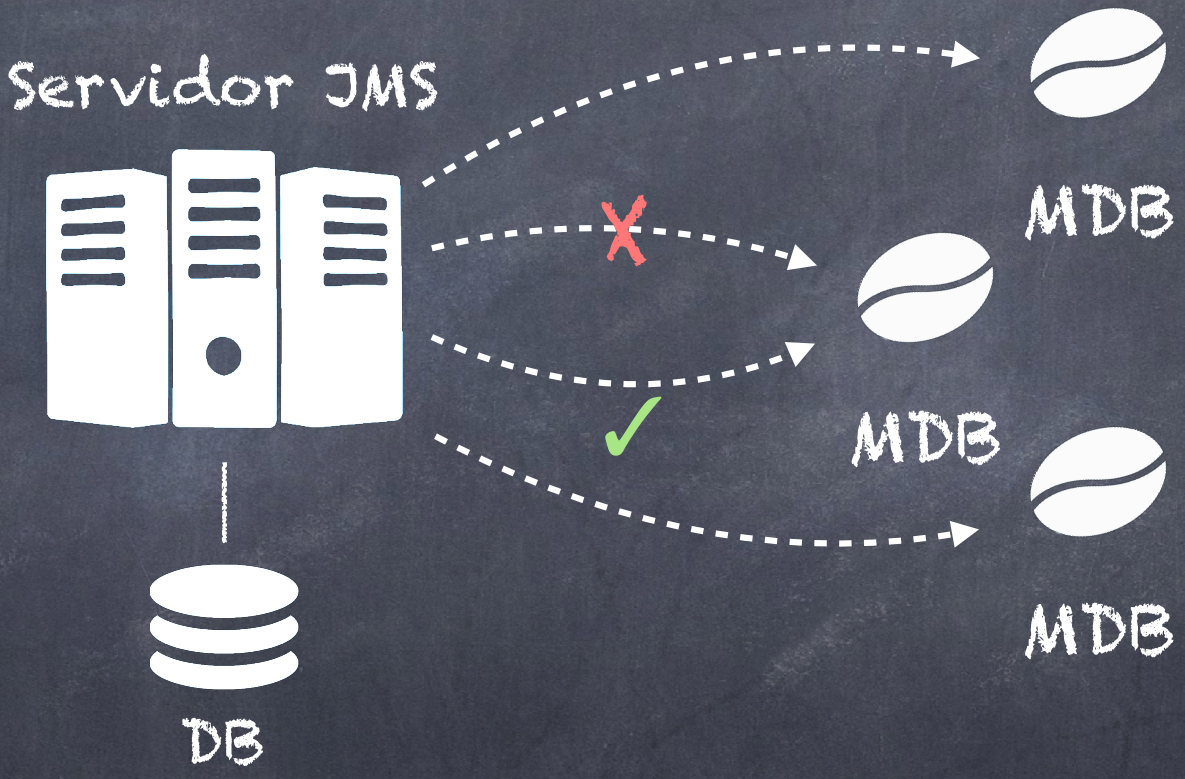
Sistema Interno
ou Externo

Características de JMS

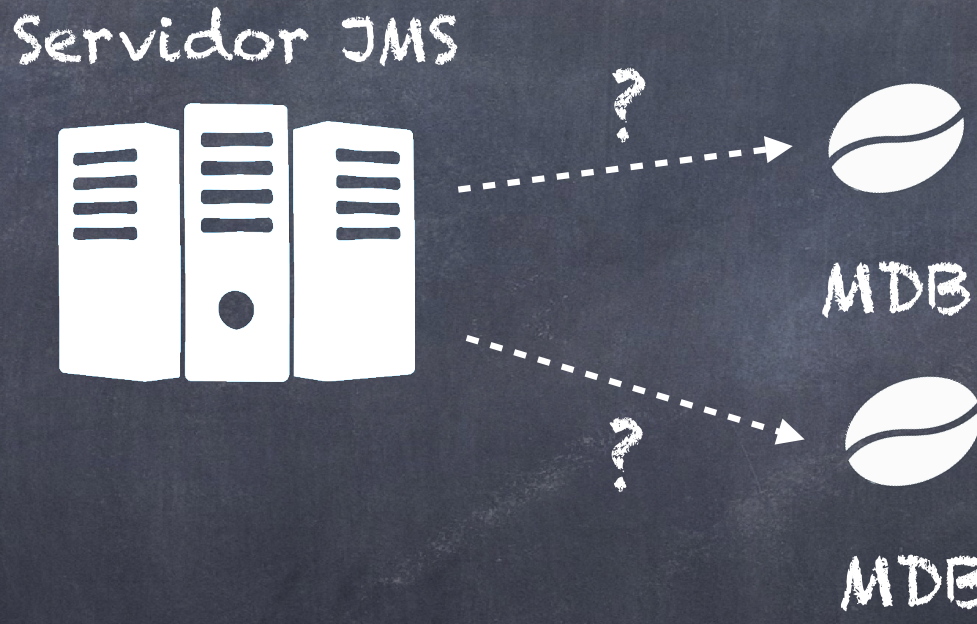
* Mensagens Persistentes

* Garantia de Entrega

* Escalável



Tipos de Entrega



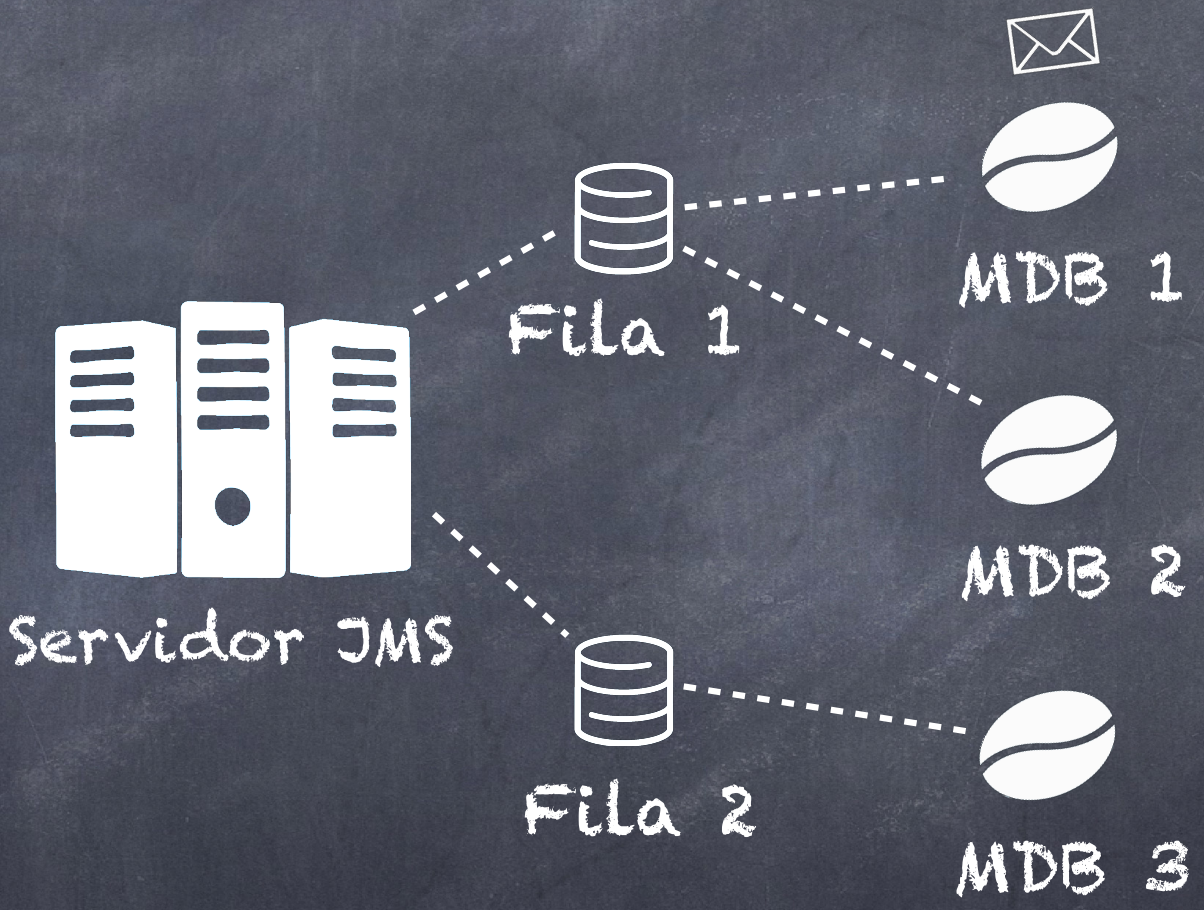
* Filas
  P2P

* Tópicos
  Pub-Sub

Filas

* Destino
  = Endereço
    do consumidor

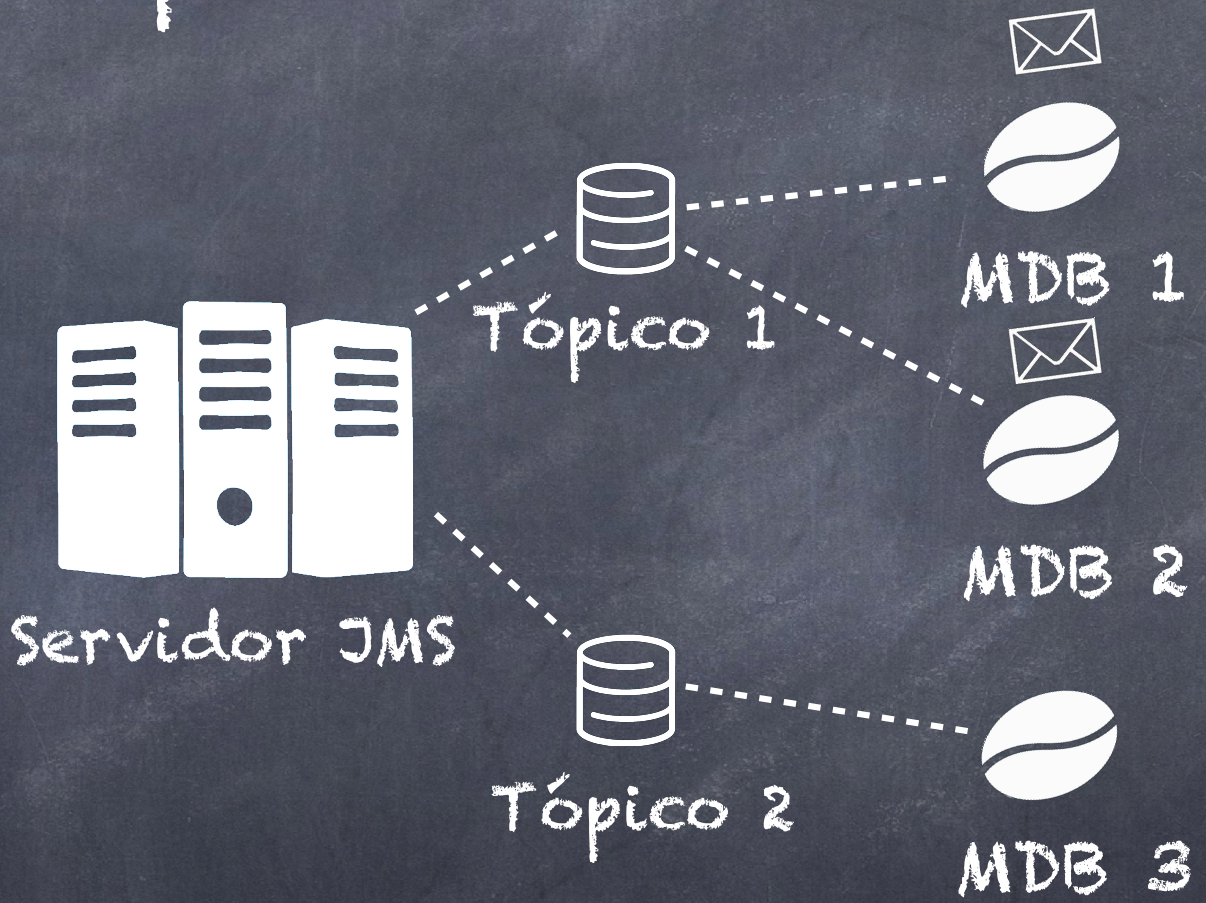
* Ponto-a-Ponto



Tópicos

* Publish-Subscribe

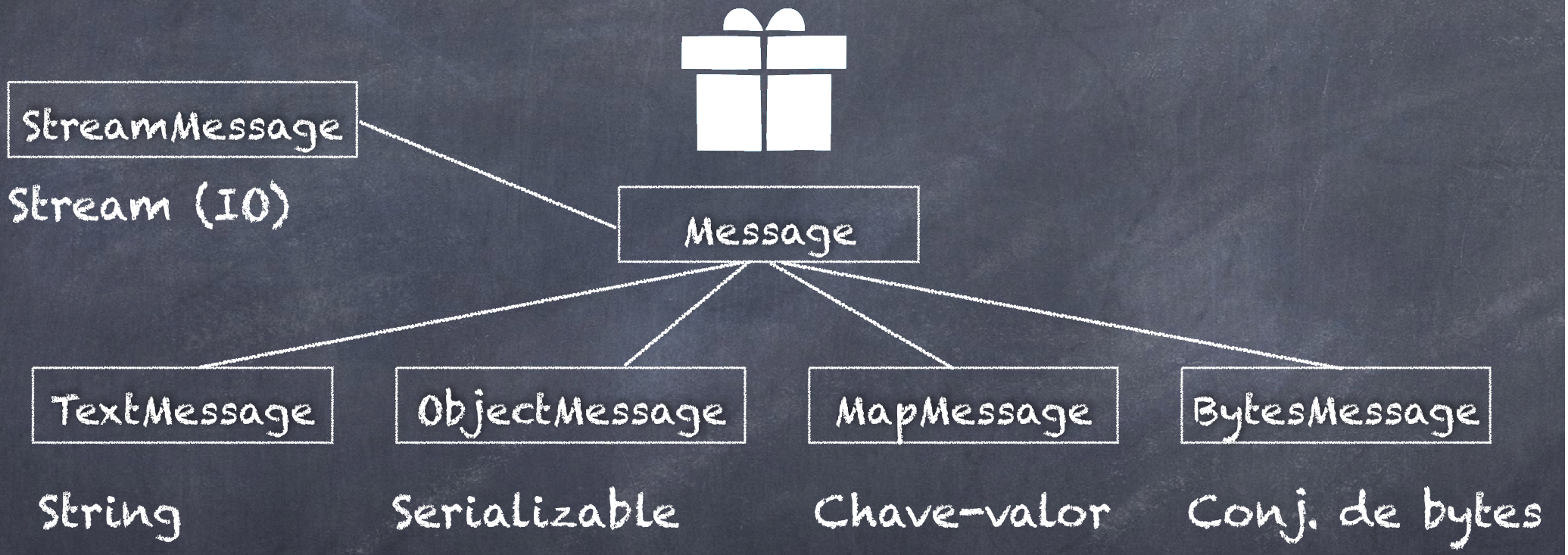
* Newsletter



```
@MessageDriven (activationConfig = {
                    @ActivationConfigProperty (propertyName = "destinationType",
                                               propertyValue = "javax.jms.Queue"),
                    @ActivationConfigProperty (propertyName = "destination",
                                               propertyValue = "queue/MailQueue")})
 public class MailSenderMDB implements
                                MessageListener {


     public void onMessage (Message message) {

             ...

     }

 }
```

queue/MailQueue = endereço

Tipos de Mensagens



```java
@MessageDriven
public class MailSenderMDB implements MessageListener {


  public void onMessage(Message message) {

        if (message instanceOf ObjectMessage) {

            ObjectMessage objMsg = (ObjectMessage) message;

            Serializable myObj = objMsg.getObject();

            ...

        }

  }
```

Publicando uma Mensagem

Algoritmo de "6" passos:

1. Busque a ConnectionFactory e a Fila
2. Crie uma conexão
3. Crie uma sessão a partir da conexão
4. Crie um produtor e uma mensagem
5. Envie a mensagem
6. Feche a conexão

Código de um EJB

```
@Resource (name="jms/MailQueue")
private Queue queue;
@Resource (name="jms/ConnectionFactory")
private ConnectionFactory connFactory;
public void postMail (String message) {
        Connection conn = connFactory.createConnection();
        Session session = conn.createSession(); //em ambientes Java EE 7
        MessageProducer producer = session.createProducer(queue);
        TextMessage textMessage = session.createTextMessage();
        textMessage.setText(message);
        producer.send(textMessage);
        conn.close();
}
```

1

2

3

4

5

6

Java EE 7

```
public void postMail (String message) {
        Connection conn = connFactory.createConnection();
        try {
            Session session = conn.createSession(); //em ambientes Java EE 7
           MessageProducer producer = session.createProducer(queue);
           TextMessage textMessage = session.createTextMessage();
           textMessage.setText(message);
           producer.send(textMessage);
        } finally {
           conn.close(); //fechar a conexão
        }
}
```

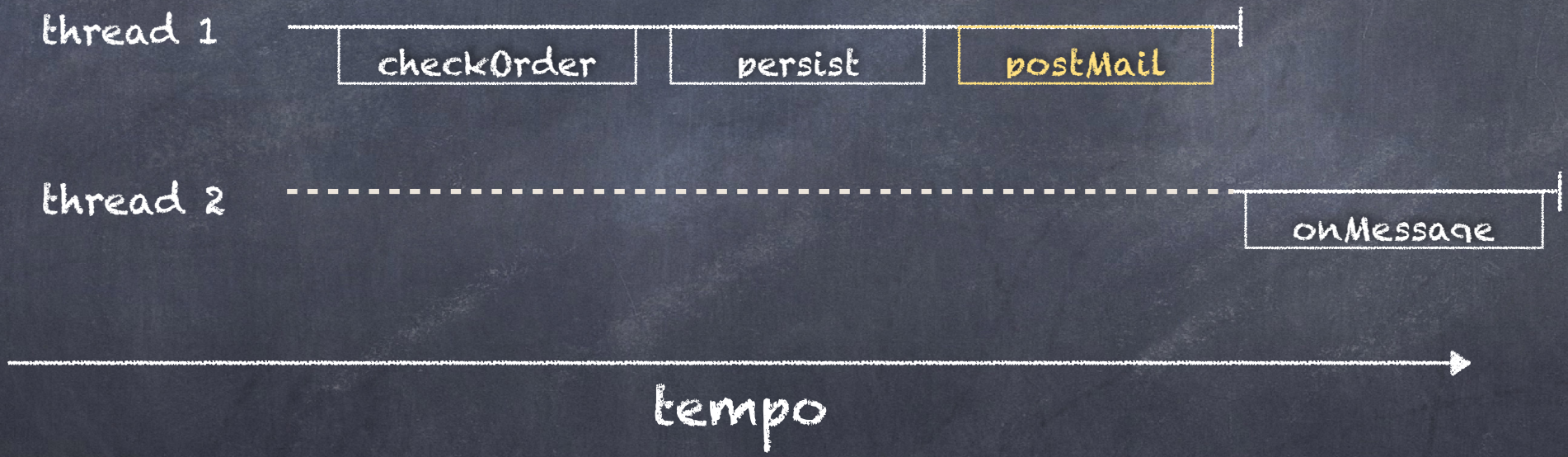
Java EE 6

```
public void postMail (String message) {
        Connection conn = connFactory.createConnection();
        try {
            Session session = conn.createSession(true,
                                Session.AUTO_ACKNOWLEDGE); //em ambientes Java EE 6
           MessageProducer producer = session.createProducer(queue);
           TextMessage textMessage = session.createTextMessage();
           textMessage.setText(message);
           producer.send(textMessage);
        } finally {
           conn.close(); //fechar a conexão
        }
}
```

Refatorando o exemplo

```
@EJB
private MailSender mailSender;
...
public void processOrder(Order order) {
        if (checkOrder(order)) {
            persist(order);
            mailSender.postMail(...);
        }
}
...
```

Anatomia da Execução



Dicas de uso

Publique via EJB/CDI = maior encapsulamento

Use injeção de dependência = sem lookup JNDI

HTTP/REST para aplicações remotas = protocolo mais amigável

Exercícios (7)

- Realize a importação do arquivo employee.xml utilizando MDBs
- Crie um método na interface EmployeeService
  - public void queueImportItems(List<Map<String, Object>> items);
- O método queueImportItems deve dividir a lista dos 10.000 registros em sublistas de 1.000, totalizando 10 sublistas.
  - Publique as sublistas na Fila java:/jms/EmployeeQueue
- Crie uma classe utilitária para publicar mensagens na fila
  - JMSUtils.publishMessage(queue,connFactory,message);