Fundamentos básicos de Transações

Helder da Rocha
www.argonavis.com.br

Objetivos

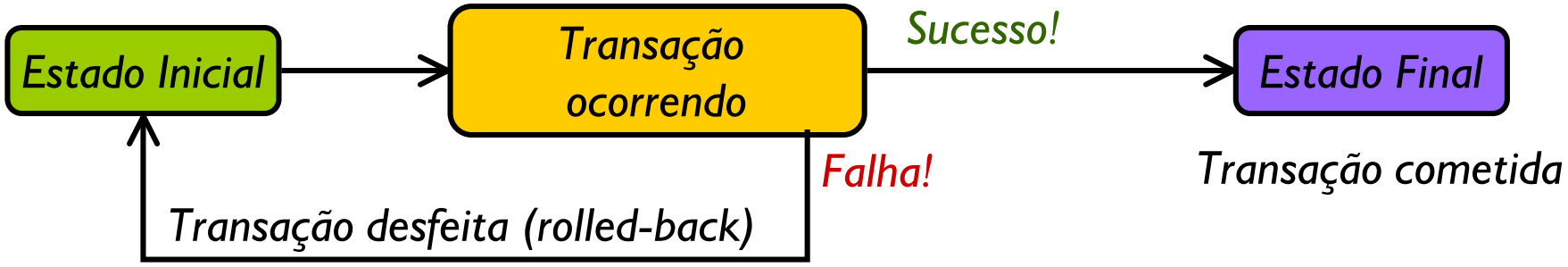
- Apresentar conceitos essenciais sobre transações em aplicações J2EE
- Este curso não aborda o assunto em profundidade.
  - Há vários tópicos importantes não mencionados
- Para uma abordagem mais profunda, consulte
  - Capítulo sobre transações no J2EE Tutorial
  - Capítulo sobre transações no livro Mastering EJB 2
  - Capítulo 17 da especificação EJB
  - Módulo 12 do curso J530 (EJB) em www.argonavis.com.br
- Roteiro
  - O que são transações; transações ACID
  - Container-managed transactions
  - Bean-managed transactions

O que são transações?

- Transações são uma técnica para simplificar a programação de aplicações
  - Programador não precisa se preocupar com recuperação de falhas e programação para sistemas multiusuário
- São unidades atômicas de procedimento
  - Sistema de transações garante que o procedimento ou termina com sucesso ou é completamente desfeita
- Suporte a transações é um componente essencial da arquitetura EJB
  - Programador EJB pode escolher entre duas formas de demarcação de transações: explícita, ou programática (Bean-Managed - BMT), e implícita, ou declarativa (Container-Managed - CMT)

Commit e Rollback

- *Uma transação engloba uma unidade de trabalho.*
  - *Durante o processo, várias sub-operações são realizadas e resultados temporários são obtidos.*
  - *No final, a transação é cometida (isto é o commit) e o resultado final é feito durável se cada sub-operação funcionar com sucesso.*
  - *Caso alguma sub-operação falhe, um processo para reverter as alterações temporárias realizadas é iniciado. Isto é o rollback, e os dados envolvidos voltam aos estados que tinham antes do início da transação*



Transações são Ácidas!

- ACID - características essenciais de uma transação: ela deve ser **A**tômica, **C**onsistente, **I**solada e **D**urável
- Atômica
  - Garante que todas as operações sejam tratadas como uma única unidade de trabalho. Todas as tarefas de uma unidade transacional devem funcionar sem erros ou todo o processo é revertido.
- Consistente
  - O estado do sistema após uma transação deve manter-se consistente (transações devem englobar processos de negócio completos)
- Isolada
  - Transação deve poder executar sem interferência de outros processos. Isto é possível utilizando sincronização.
- Durável
  - Dados alterados durante a transações devem ser guardados em meio persistente até que a transação complete com sucesso

Demarcação de transações

- O controle de transações em J2EE resume-se a demarcação de transações
  - Consiste apenas do controle de quando ela será iniciada e quando será concluída (ou desfeita)
- Há várias formas de demarcar transações
  - Podem ser demarcadas no cliente (comuns, servlets, beans, etc.) e propagadas para os componentes
  - Podem ser demarcadas no servidor, de duas formas: no container (implícita), usando declarações no DD, ou no bean (explícita), usando APIs como JTA, JDBC ou JMS

JTS e JTA

- *JTS - Java Transaction Service é um mapeamento Java-CORBA da especificação Object Transaction Service (OTS 1.1)*
  - *JTS é usado por fabricantes de containers*
  - *Desenvolvedores de EJBs não precisam usar JTS (suporte por parte do container é opcional)*
  - *Classes: org.omg.CosTransactions e outras*
- *JTA - Java Transaction API é uma especificação de interfaces para o sistema de transações*
  - *JTA é utilizado por desenvolvedores de beans que têm controle explícito (programático) de transações (BMT)*
  - *Suporte por parte do container é obrigatório*
  - *Classes: javax.transaction.UserTransaction e outras*

Demarcação explícita de transações

- Consiste em utilizar alguma API de controle de transações diretamente no código
- É preciso informar ao container que bean está usando Bean-Managed Transactions (BMT) através do deployment descriptor:

```
<transaction-type>Bean</transaction-type>
```

- No código, use uma API para demarcar o início e o fim das transações: JTA, JDBC ou JMS
  - Métodos de Connection (java.sql)
  - Métodos de QueueSession ou TopicSession (javax.jms)
  - Métodos de UserTransaction (javax.transaction)

Transações com JDBC - exemplo

```
(...)
public void ship (String productId,
                  String orderId,
                  int quantity) {
  try {
     con.setAutoCommit(false); // con é java.sql.Connection
       updateOrderItem(productId, orderId);
       updateInventory(productId, quantity);
     con.commit();
  } catch (Exception e) {
    try {
      con.rollback();
      throw new EJBException("Transaction failed: "
                                          + e.getMessage());
    } catch (SQLException sqx) {
      throw new EJBException("Rollback failed: "
                                          + sqx.getMessage());
    }
  }
}
```

WarehouseBean.java

Transações explícitas com JTA

```
(...)
   public void withdrawCash(double amount) {
      UserTransaction ut = sessionCtx.getUserTransaction();
      try {
         double mbState = machineBalance;
         ut.begin();
          updateChecking(amount);
          machineBalance -= amount;
          insertMachine(machineBalance);
         ut.commit();
      } catch (Exception ex) {
          try {
             ut.rollback();
             machineBalance = mbState;
          } catch (SystemException syex) {
              throw new EJBException
                 ("Rollback failed: " + syex.getMessage());
          }
          throw new EJBException
             ("Transaction failed: " + ex.getMessage());
     }
   }
```

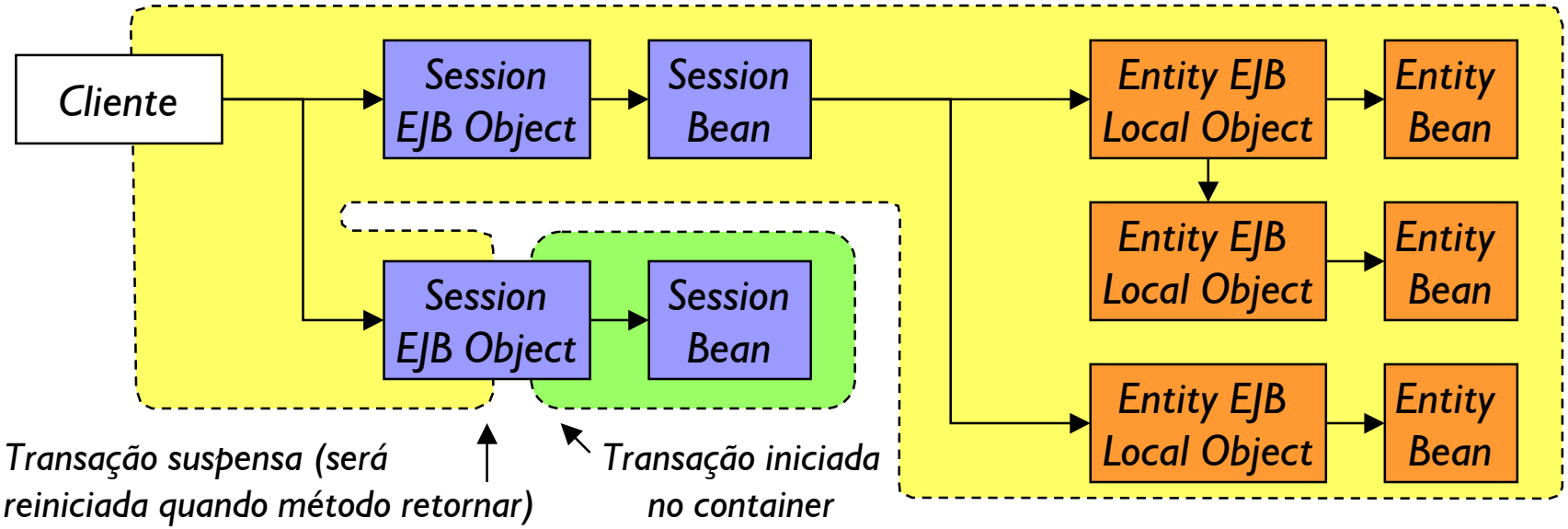
EJBContext do bean (SessionContext ou MessageDrivenContext)

Estado de variável de instância NÃO será revertido em caso de rollback()

TellerBean.java

Propagação de transações

- Transações terminam no mesmo lugar onde começaram
- Pode-se melhorar a performance de uma operação que faz muitas chamadas (principalmente se forem chamadas a Entity Beans) colocando-a no contexto de uma transação
  - O contexto da transação será propagado para todos os métodos chamados (se não iniciarem nova transação)
  - Se métodos chamados iniciarem nova transação, a transação principal será suspensa até que o método termine



Controle implícito (declarativo)

- *Container-Managed Transactions (CMT)*
  - *Controle de transações totalmente gerenciado pelo container*
  - *Não permite o uso de métodos commit() e rollback() de java.sql.Connection ou javax.jms.Session dentro do código*
  - *Única forma de controlar transações em Entity Beans*
- *No deployment descriptor, espeficique o uso de CMT abaixo de <session> ou <message-driven>*

```
<transaction-type>Container</transaction-type>
```

- *Depois, defina a política de transações para cada método*

```
<assembly-descriptor>
   <container-transaction>
      <method> ... </method>
      <trans-attribute>Required</trans-attribute>
   </container-transaction>
   (...)
</assembly-descriptor>
```

Demarcação declarativa - exemplo

```
<ejb-jar>
    <enterprise-beans>
        <session>
            <ejb-name>BankEJB</ejb-name>
            <home>j2eetut.bank.BankHome</home>
            <remote>j2eetut.bank.Bank</remote>
            <ejb-class>j2eetut.bank.BankBean</ejb-class>
            <session-type>Stateful</session-type>
            <transaction-type>Container</transaction-type>
            (...)
        </session>
    </enterprise-beans>
    <assembly-descriptor>
        <container-transaction>
            <method>
                <ejb-name>BankEJB</ejb-name>
                <method-name>getSavingBalance</method-name>
            </method> (... outros <method> ...)
            <trans-attribute>Required</trans-attribute>
        </container-transaction> (...)
    </assembly-descriptor>
</ejb-jar>
```

O método getSavingsBalance(...) do bean BankEJB tem controle de transações demarcado durante a sua execução usando a política Required

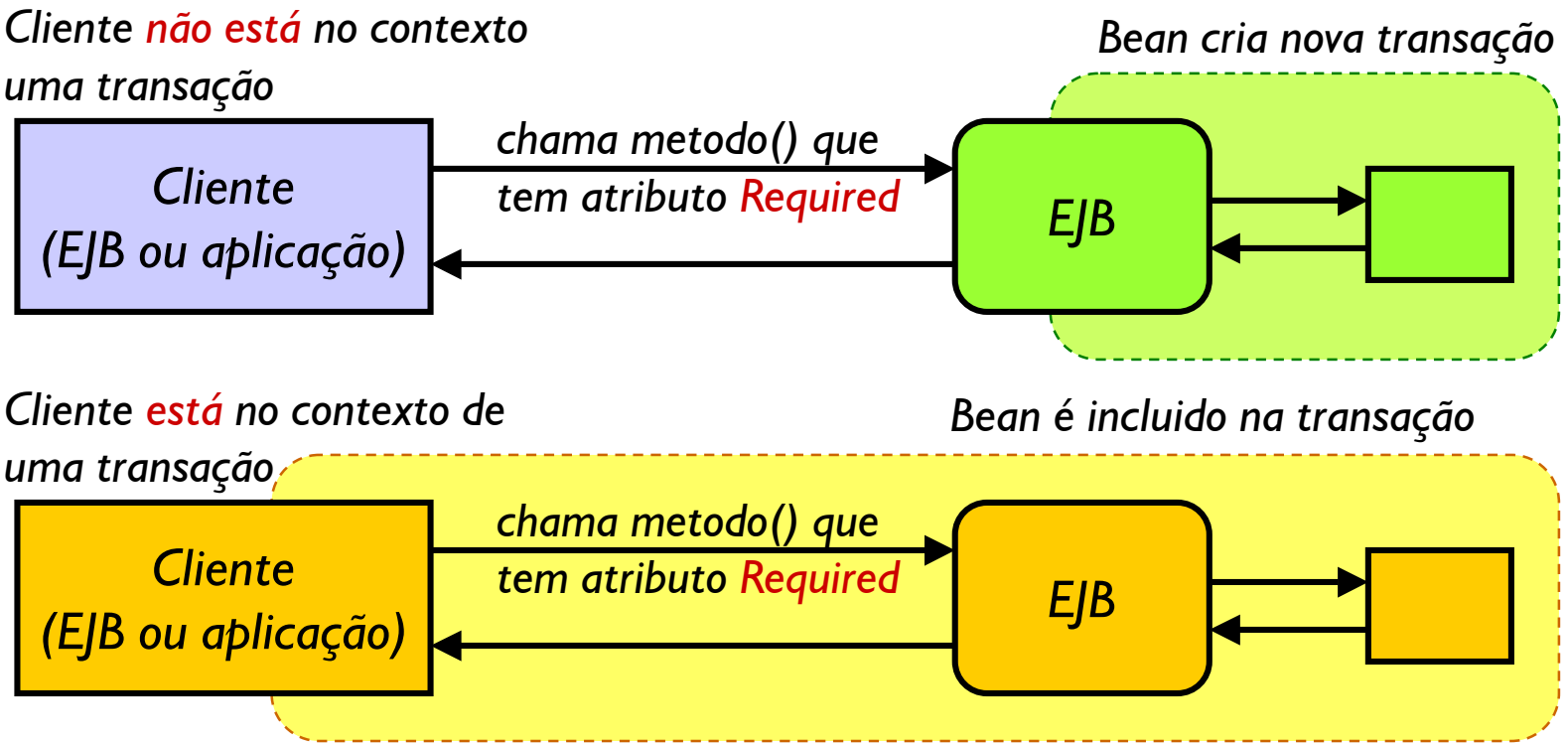
bank-ejb-jar.xml

Atributos (políticas transacionais) em CMT

- O elemento <trans-attribute> define a política transacional do componente
  - Define como ele irá reagir quando o seu método for chamado por um cliente dentro ou fora do contexto de uma transação
- Os valores suportados para este elemento (depende do tipo de bean) são
  - NotSupported
  - Supports
  - **Required** ← Use este sempre que possível em Entity Beans, Message-driven beans e Session Beans
  - RequiresNew
  - Mandatory
  - Never

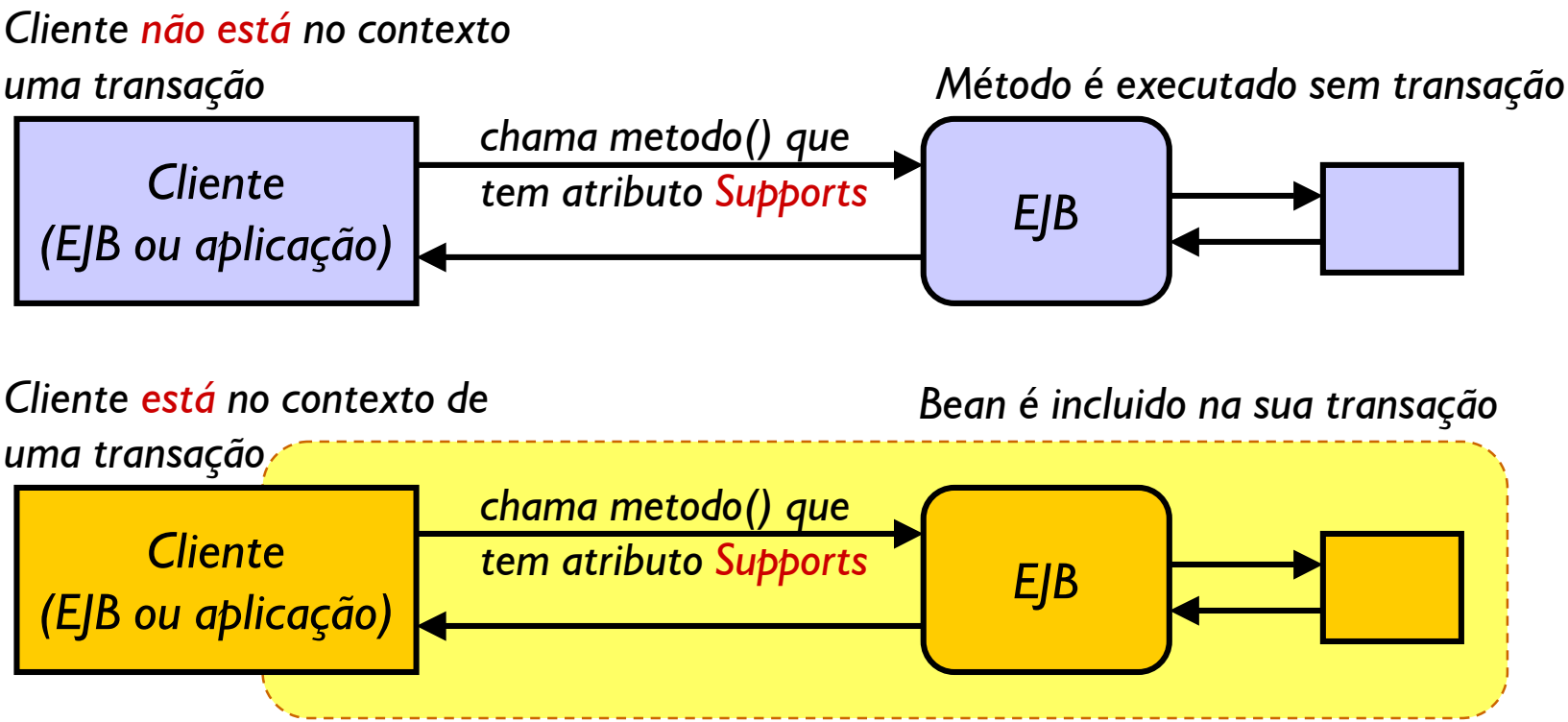
Required

- *Indica que o método tem que ser chamado dentro do escopo de uma transação*
  - *Se não existe transação, uma nova é criada e dura até que o método termine (é propagada para todos os métodos chamados)*
  - *Se já existe uma transação iniciada pelo cliente, o bean é incluído no seu escopo durante a chamada do método*



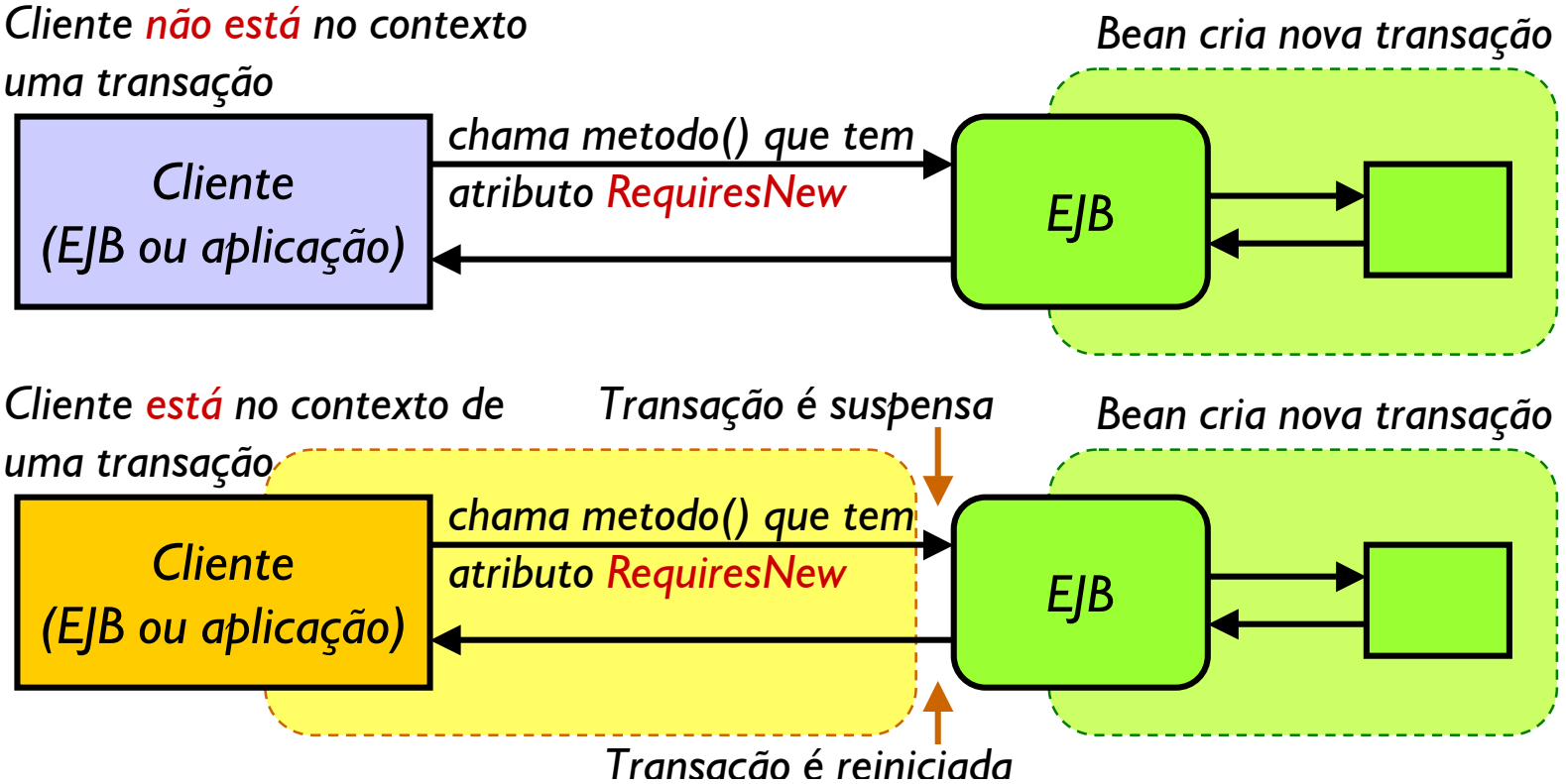
Supports

- *Indica que o método suporta transações*
  - *Será incluído no escopo da transação do cliente se existir*
  - *Se ele for chamado fora do escopo de uma transação ele realizará suas tarefa sem transações e pode chamar objetos que não suportam transações*



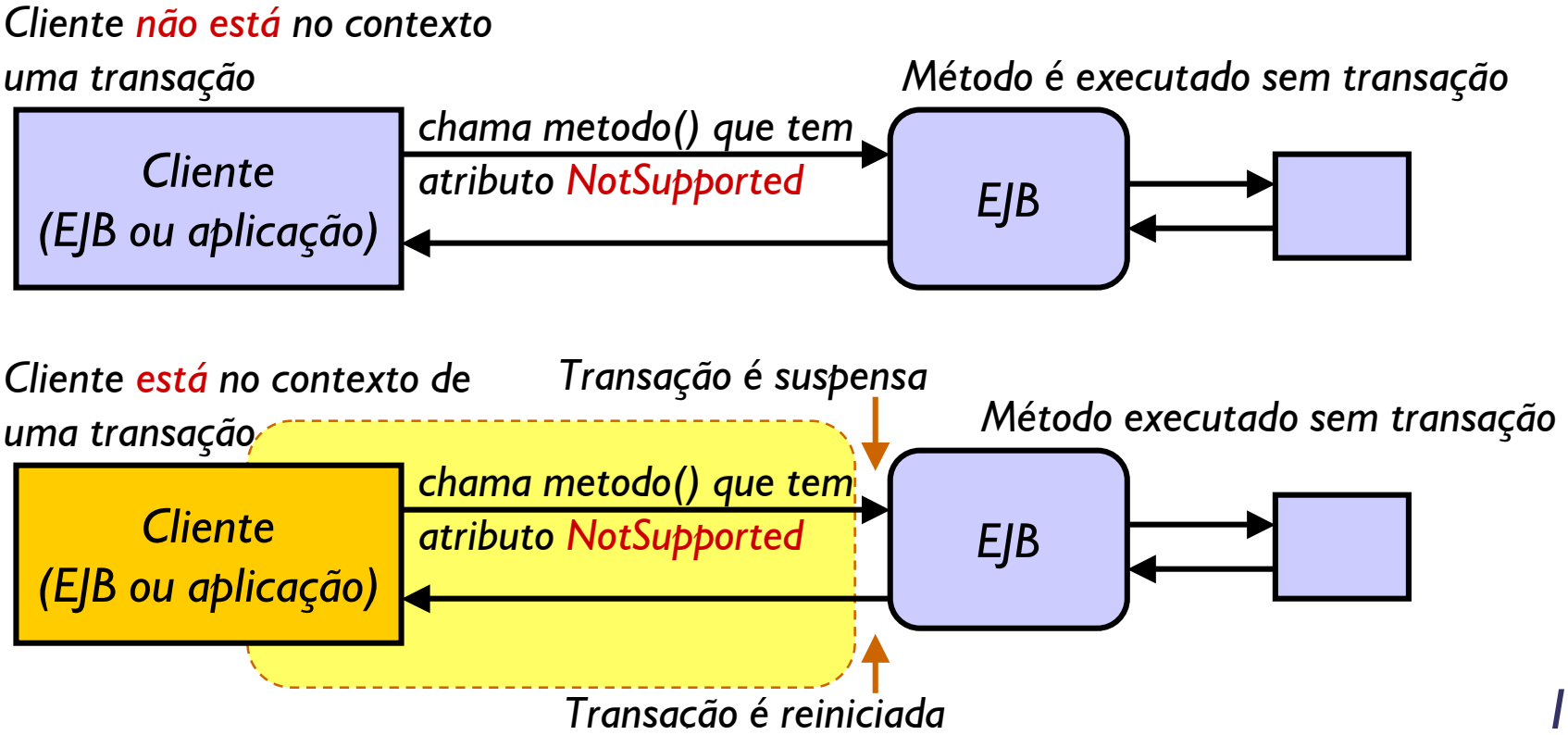
RequiresNew

- *Indica que uma nova transação, iniciada no escopo do bean, é sempre criada*
  - *Estando ou não o cliente no escopo de uma transação, o bean irá iniciar uma nova transação que iniciará e terminará no bean.*



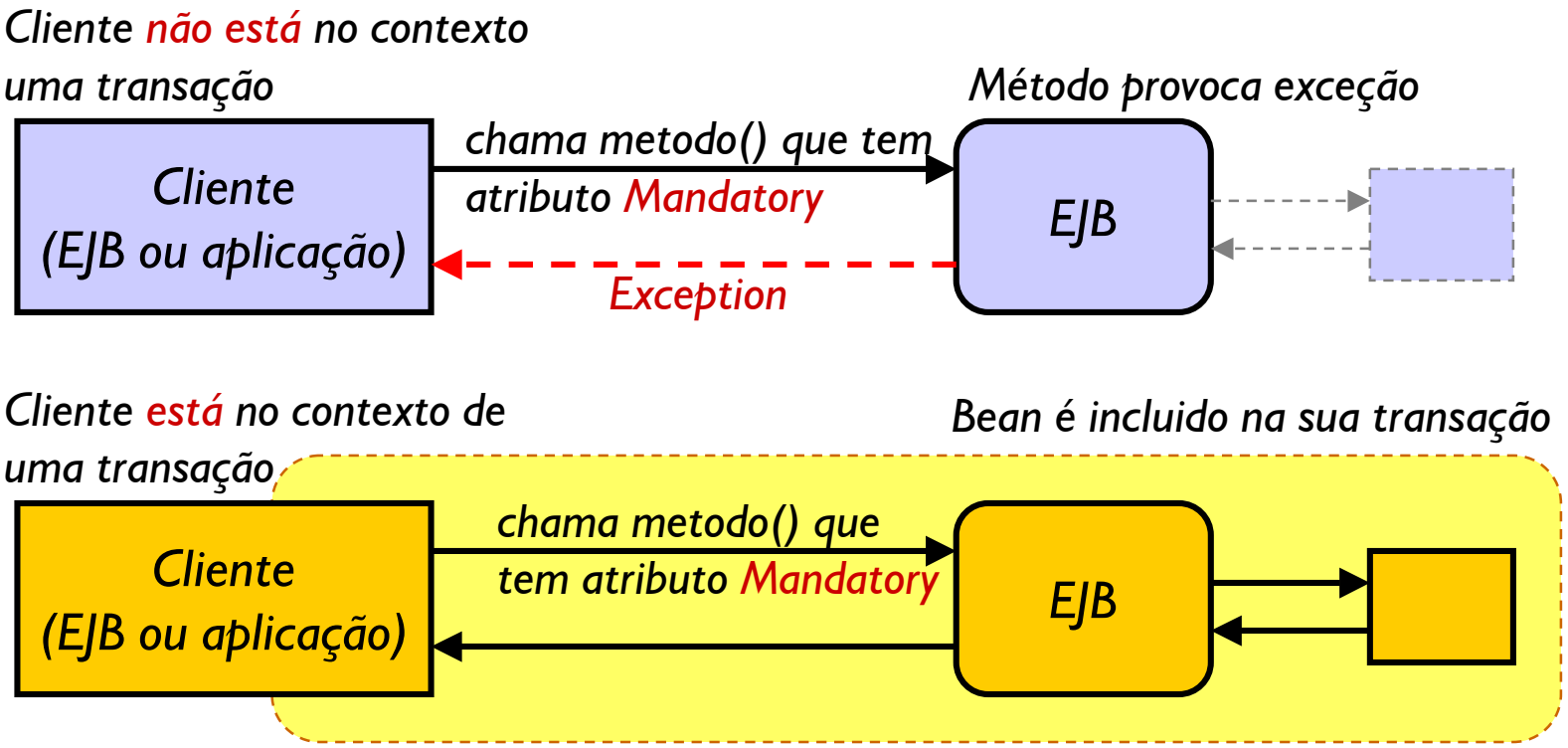
NotSupported

- *Indica que o método não suporta transações*
  - *Se o método for chamado pelo cliente no escopo de uma transação, a mesma será suspensa enquanto durar a chamada do método (não haverá propagação de transações do cliente)*



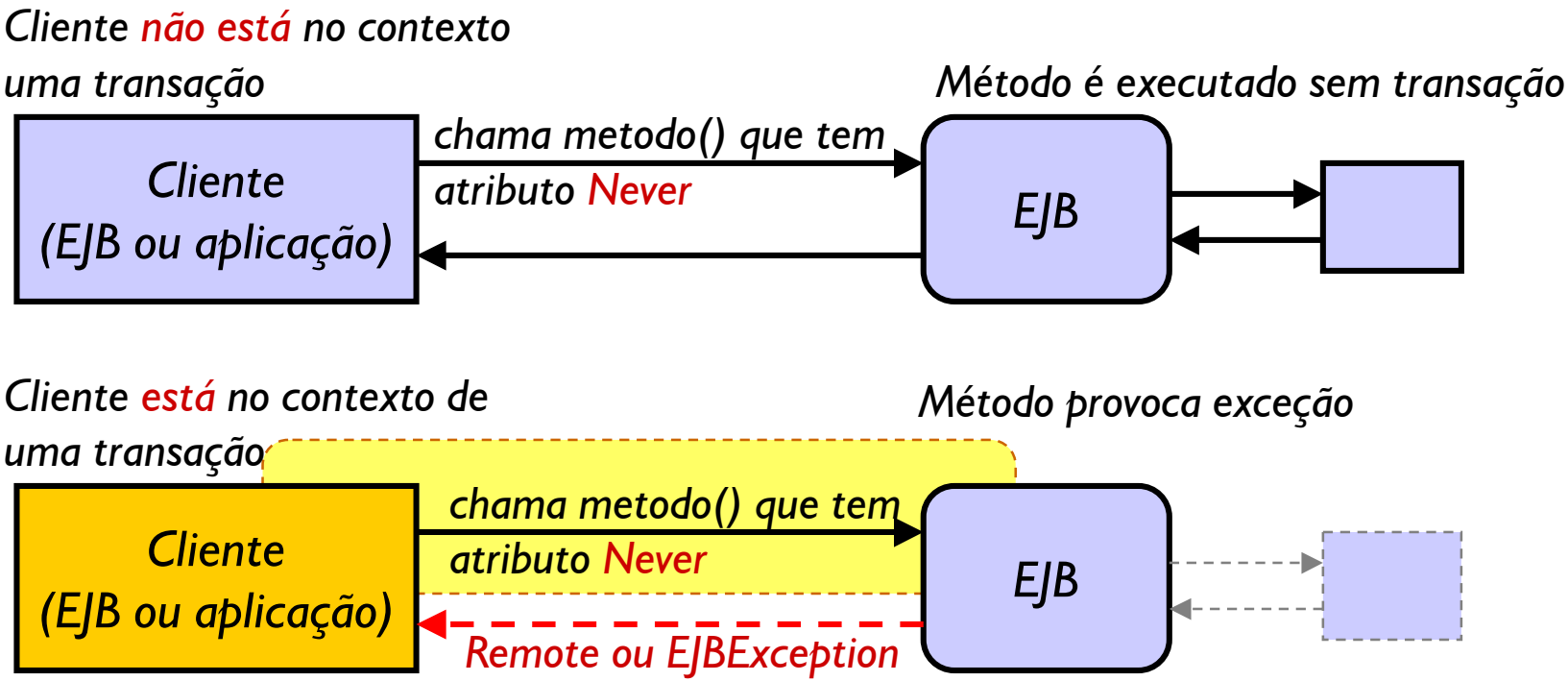
Mandatory

- *Indica que o método só pode ser chamado no escopo de uma transação do cliente*
  - *Se o método for chamado fora de uma transação, ele causará javax.transaction.TransactionRequiredException (ou javax.ejb.TransactionRequiredLocalException)*



Never

- *Indica que o método nunca pode ser chamado no escopo de uma transação*
  - *Se o cliente que chama o método for parte de uma transação, o bean irá provocar um RemoteException (ou EJBException em clientes locais)*



Suporte de atributos por tipo de bean

- Nem todos os atributos são suportados em qualquer tipo de bean. A tabela abaixo relaciona os atributos suportados por cada tipo.

Atributo	Session Beans			Entity Beans	Message Driven Beans
	Stateful com SessionSynchronization	Stateful	Stateless		
Required	✓	✓	✓	✓	✓
RequiresNew	✓	✓	✓	✓	✗
Supports	✗	✓	✓	✗	✗
NotSupported	✗	✓	✓	✗	✓
Mandatory	✓	✓	✓	✓	✗
Never	✗	✓	✓	✗	✗

Fonte: [2]

Transações e exceções

- A especificação J2EE classifica exceções em 2 tipos
  - System exceptions
  - Application exceptions
- System exceptions
  - São exceções que indicam erros no servidor e serviços usados pela aplicação
  - São instâncias de RemoteException e RuntimeException
  - Causam rollback automático de transações
- Application exceptions
  - São exceções que indicam erros na lógica da aplicação
  - Consistem de todas as subclasses de Exception exceto as classes que são System Exceptions
  - Por mais graves que sejam, não causam rollback

Conclusões

- O serviço de transações é importante para garantir a integridade dos dados
  - Use sempre que necessário (obrigatório em Entity Beans)
- Use CMT sempre que possível
  - Muito mais simples
  - Menos risco de deadlock e gargalos de performance
  - Separação de papéis (deployer pode ajustar o mehor emprego de políticas transacionais)
- Utilize apenas onde realmente for necessário
  - Nem sempre um método precisa estar em uma transação
- Se bem planejadas, o uso de transações pode melhorar a performance de aplicações EJB
  - A sincronização (load/store) ocorre uma vez por transação

Fontes

[1] *Ed Roman et al. Mastering Enteprise JavaBeans, 2nd. Edition, Chapter 10: Transactions. John Wiley & Sons, 2002.*

[2] *Linda de Michiel et al. Enterprise JavaBeans 2.1 Specification, Chapter 17: Support for Transactions. Sun Microsystems, 2003.*

[3] *Richard Monson-Haefel. Enterprise JavaBeans, 3rd. Edition. O'Reilly and Associates, 2001*

[4] *Dale Green. J2EE Tutorial: Transactions. Sun J2EE Tutorial, Sun Microsystems, 2002*

[5] *Jim Farley et al. Java Enterprise in a Nutshell, 2nd. Edition. O'Reilly & Associates 2002.*

helder@argonavis.com.br

argonavis.com.br