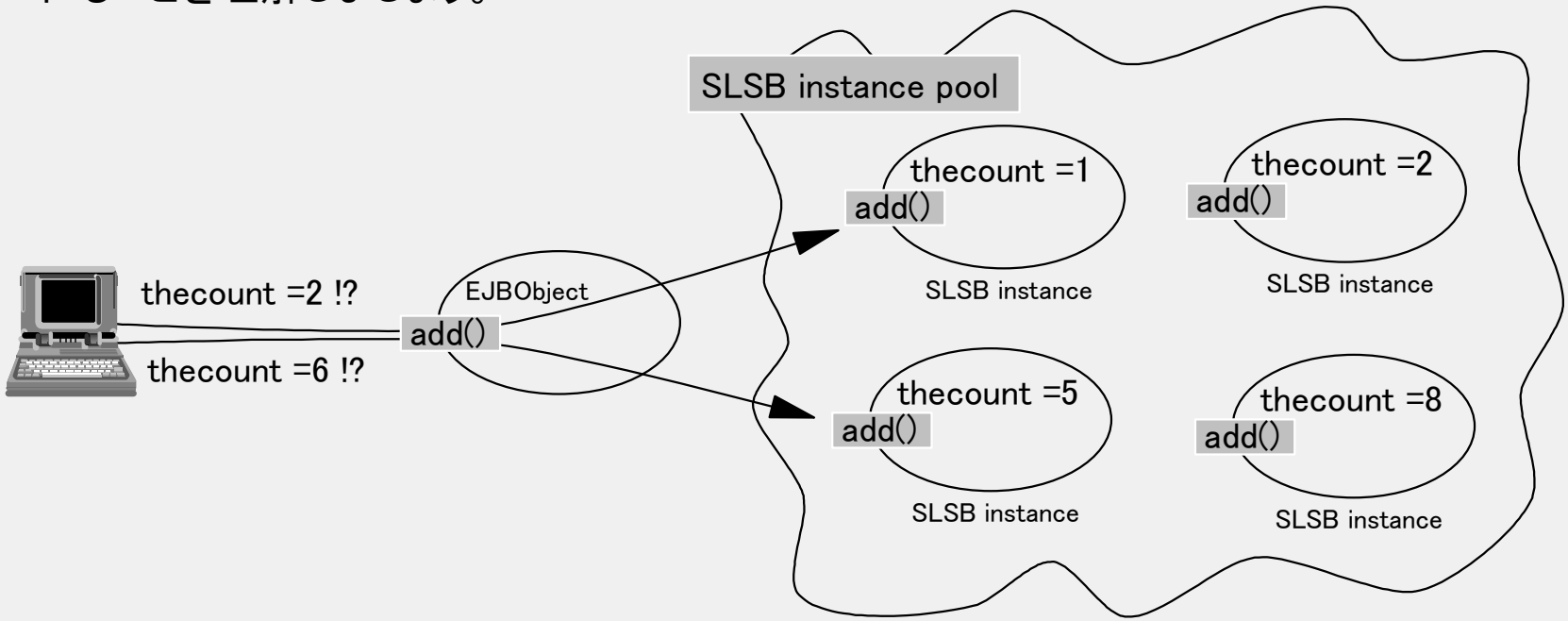
SESSION	実例から学ぶEJBトランザクション
NUMBER	*Track B – Session 6 (B-6)*
SPEAKER	小島 賢二
TITLE	主任ITスペシャリスト
AFFILIATION	日本アイ・ビー・エム・システムズ エンジニアリング株式会社 PvC&Webソリューション・システム部

目次

- EJBの基礎
  1. Stateless SessionBeanの挙動
  2. Stateful SessionBeanの挙動
  3. もっとも単純なCMP EntityBeanの作成
  4. CMP EntityBeanの永続化サービスを実感
- CMP EnittyBeanトランザクション詳細
  1. OptionAとOptionCの違いを実感
  2. 「更新の為の検索」オプションの利用
  3. 「読み取り専用」オプションの利用
  4. Dirty Readシナリオ
  5. Unrepeatable Readシナリオ
  6. Phantom Readシナリオ

EJBの基礎(1)
Stateless SessionBeanの挙動

- Stateless SessionBeanの特長を理解しましょう。
- 下図のようにSLSBインスタンスは、メソッド呼び出しの度に、プール内から動的に割り当てられることを理解しましょう。



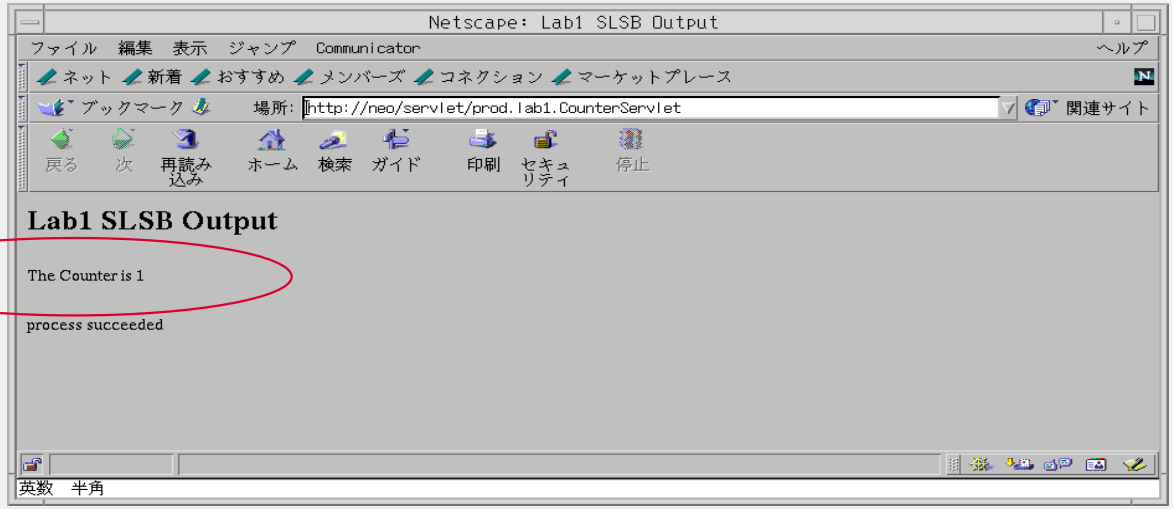
SLSBでは、複数回のメソッド呼び出しに跨って状態を保持することができないことを実感しましょう。

EJBの基礎(1)
Stateless SessionBeanの挙動(テスト手順)

以下の手順を実行します。

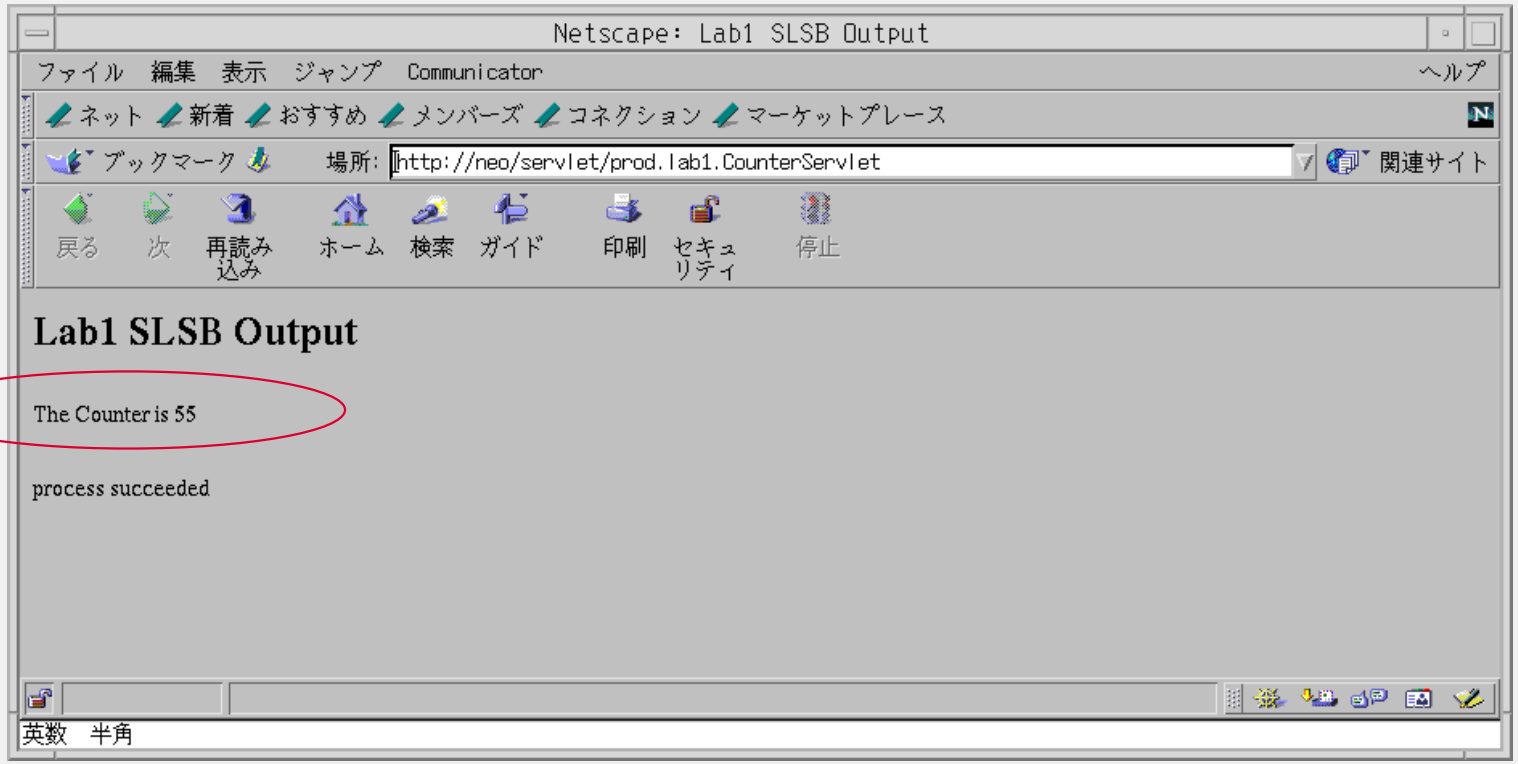
1. ブラウザーから、http://$host_name/servlet/prod.lab1.CounterServletにアクセスします。
   アクセス後に出力されるCounter値を記憶します。
2. 次のページの設定で、akstressを実行し、ストレスをかけます。
3. akstressのストレスが終了後、再度、ブラウザーから
   http://$host_name/servlet/prod.lab1.CounterServletにアクセスします。
   アクセス後に出力されるCounter値を前回と比較します。

1.
一回目の
アクセス
の結果



EJBの基礎(1)
Stateless SessionBeanの挙動

3.
二回目の
アクセス
の結果

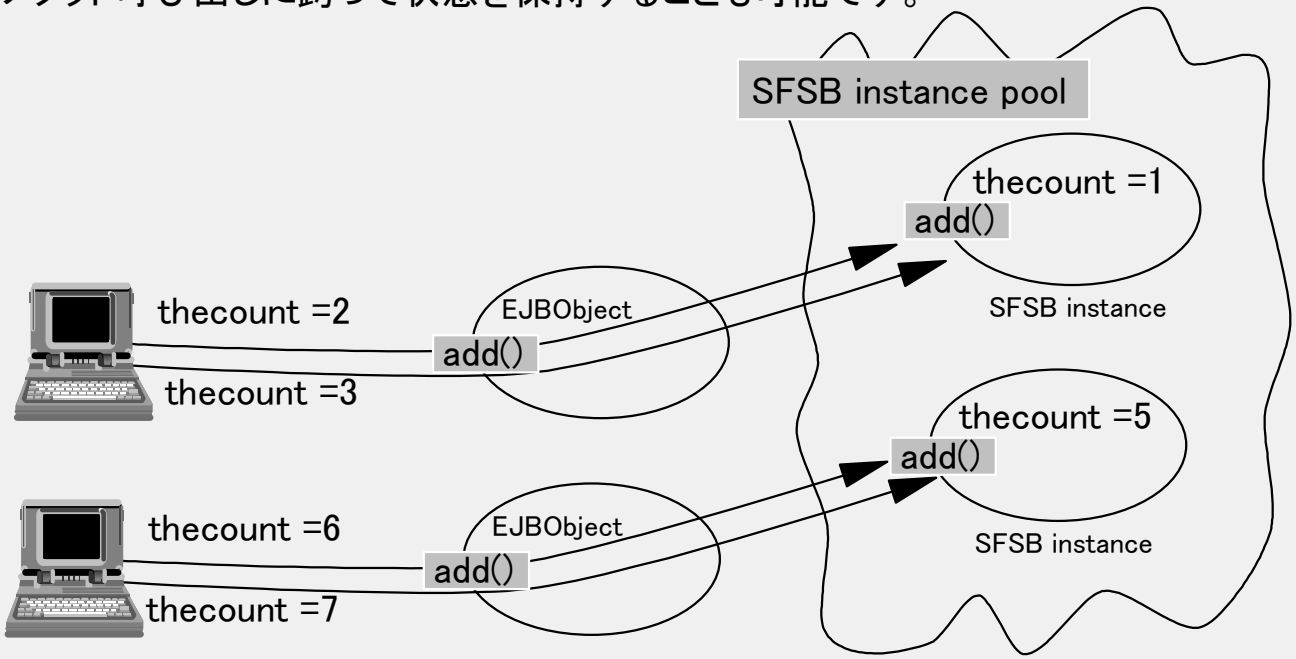


akstressツールを使用して500リクエスト終了後、再度ブラウザーからアクセスした時のCounter値は、予測不能な値となります。二回目のアクセスの結果は、SLSBインスタンスプール内で偶然割り当てられたCounterBeanインスタンスの値が返されるのです。

- このテスト結果より、SLSBは複数メソッドに跨ってインスタンス変数を引き継ぐことができないことを実感することができます。

EJBの基礎(2)
Stateful SessionBeanの挙動

- Statefull SessionBeanの特長を理解しましょう。
- 下図のように、SFSBインスタンスは、EJBClientと1対1に割り当てられます。したがって、EJBClientからの複数回のメソッド呼び出しに跨って状態を保持することも可能です。



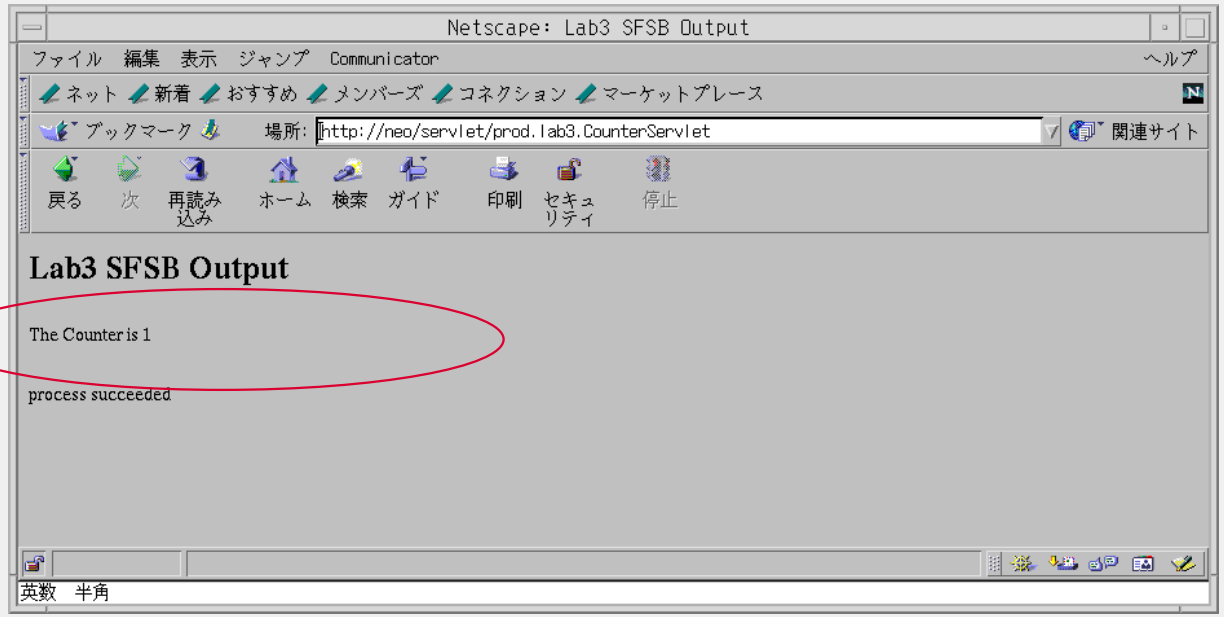
SFSBインスタンスは、EJBClientの状態を保持することが可能です。

EJBの基礎(2)
Stateful SessionBeanの挙動(テスト手順)

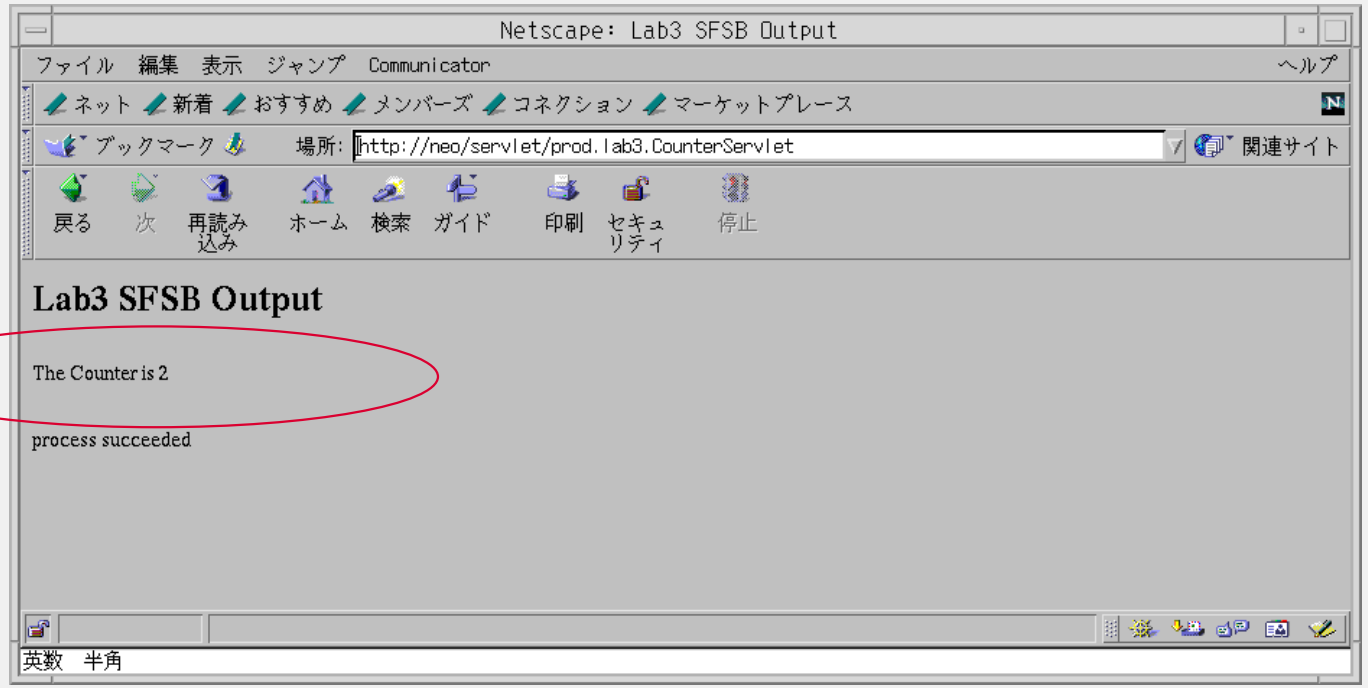
以下の手順を実行します。

1. ブラウザーから、http://$host_name/servlet/prod.lab3.CounterServletにアクセスします。
   アクセス後に出力されるCounter値を記憶します。
2. 次のページの設定で、akstressを実行し、ストレスをかけます。
3. akstressのストレスが終了後、再度、ブラウザーから
   http://$host_name/servlet/prod.lab3.CounterServletにアクセスします。
   アクセス後に出力されるCounter値と前回の値とを比較します。

SFSBのインスタンス変数(theCount_)に、各ブラウザーからのアクセス数を保持できる(状態遷移)ことを意味します。



EJBの基礎(2)
Stateful SessionBeanの挙動



- akstressツールによって、50スレッドから合計500リクエストが送信されましたが、その後に、再度ブラウザーからアクセスした時のCounter値は、2でした。これは、SFSBインスタンスが、EJBClientと1対1に割り当てられていることを示しています。
- SFSBは、特定のクライアントからの複数メソッド呼び出しに跨った状態遷移を保持することが可能であることを意味します。

EJBの基礎(3)
もっとも単純なCMP EntityBeanの作成

Remote Interface

```
import java.rmi.RemoteException;
import javax.ejb.*;
public interface Counter extends EJBObject {
        int addValue() throws  RemoteException;
        int getValue() throws RemoteException; }
```

Home Interface

```
import javax.ejb.*;
import java.rmi.RemoteException;
import java.util.Enumeration;
public interface CounterHome extends EJBHome {
        Counter create(CounterKey key) throws CreateException, RemoteException;
        Counter findByPrimaryKey(CounterKey key) throws FinderException, RemoteException;
        Enumeration findAllCounters() throws FinderException, RemoteException; }
```

PrimaryKey Class

```
import java.io.Serializable;
public class CounterKey implements Serializable {
        public String counterKey_ ;
        public CounterKey() {  this.counterKey_ = "Lab7CounterRecord"; }
        public CounterKey(String argCounterKey ) { this.counterKey_ = argCounterKey;}
        public boolean equals(Object o) {
                if(o instanceof CounterKey) {
                        CounterKey otherKey = (CounterKey) o;
                        return (((this.counterKey_ == otherKey.counterKey_)));
                }
                else {
                        return false;
                }
        }
        public int hashCode() {return ((new String(counterKey_).hashCode()));}
}
```

FinderHelper

```
public interface CounterBeanFinderHelper {
        String findAllCountersQueryString =
                "select * from ejb.counterbeantbl"; }
```

EJBの基礎(3)
もっとも単純なCMP EntityBeanの作成

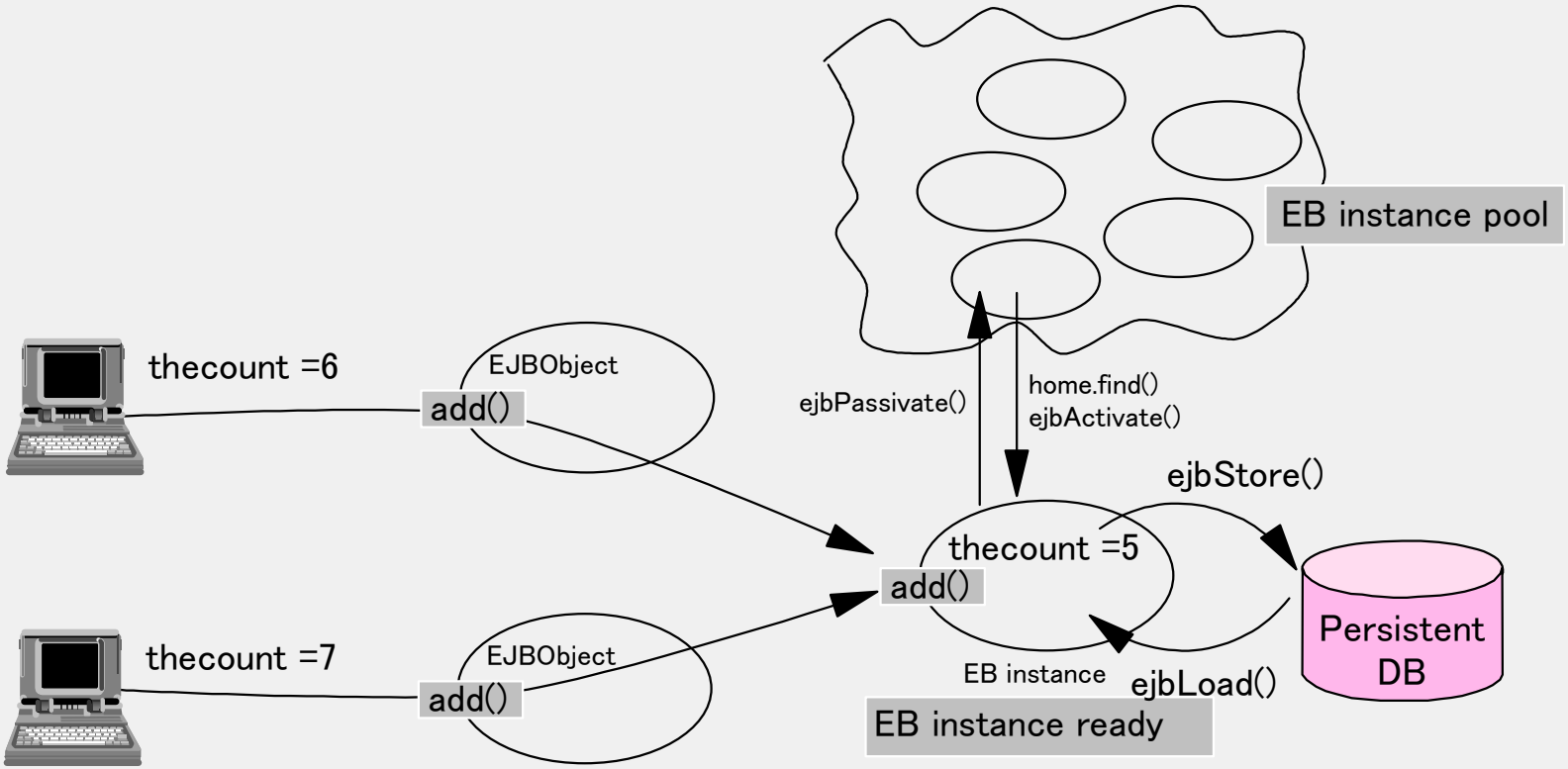
EntityBean

```
import javax.ejb.*;
import java.rmi.RemoteException;
import com.ibm.ejs.ras.*;
public class CounterBean implements EntityBean {
        private EntityContext ctx_;
        // For Tr
        private static final TraceComponent tc =
Tr.register(CounterBean.class);
        // Container-managed state fields. They must be public.
        public String      counterKey_ ;
        public int         theCount_ ;

        public CounterBean() { Tr.event(tc, "called CounterBean() method"); }

        public void setEntityContext(EntityContext argCtx) {
                this.ctx_ = argCtx;
                Tr.event(tc, "called setEntityContext() method "); }

        public void unsetEntityContext() {
                this.ctx_ = null;
                Tr.event(tc, "called unsetEntityContext() method"          }

        public void ejbCreate(CounterKey argCounterKey) {
                this.counterKey_ = argCounterKey.counterKey_;
                this.theCount_ = 0;
                Tr.event(tc, "called ejbCreate() method "); }

        public void ejbPostCreate() {
                Tr.event(tc, "called ejbPostCreate() method "); }
```

```
public void ejbActivate() { Tr.event(tc, "called ejbActivate() method "); }

public void ejbLoad() { Tr.event(tc, "called ejbLoad() method "); }

public void ejbStore() { Tr.event(tc, "called ejbStore() method "); }

public void ejbPassivate() { Tr.event(tc, "called ejbPassivate() method "); }

public void ejbRemove() { Tr.event(tc, "called ejbRemove() method "); }

public int getValue() throws RemoteException {
                Tr.event(tc, "called getValue() method ");
                return this.theCount_ ; }

public int addValue() throws RemoteException {
          Tr.event(tc, "called addValue() method" );
           this.theCount_++;
           return this.theCount_ ; }

}
```

EJBの基礎(4)
CMP EBの永続化サービスを実感 !!

- 複数のEJBClientから共有アクセスするEBを作成し、SQLコーディングなしで自動的にDBへ保存されたり、共有アクセスの排他制御が行われることを実感しましょう。



CMP　EBでは、永続化や排他制御を意識しないで開発することができます。

EJBの基礎(4)
CMP EBの永続化サービスを実感 !!

手順

1. ブラウザーから以下のURLにアクセスしてください。そして、Counter値を確認してください。
   http://$host_name/servlet/prod.lab7.CounterServlet

2. AdminConsoleを使用して、AppServerを再起動してください。

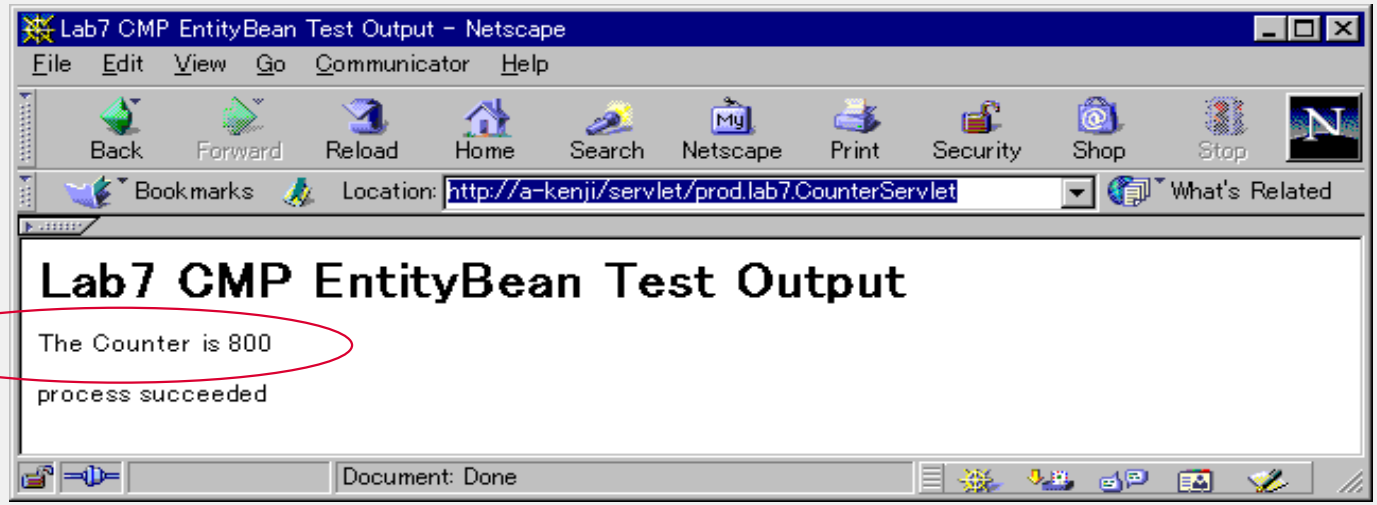
3. 再び、ブラウザーから以下のURLにアクセスしてください。そしてCounter値を確認してください。
   http://$host_name/servlet/prod.lab7.CounterServlet

*このCounter値が加算されたいくことから、Servletの初期化やEJSの再起動には影響を受けず
EJBContainerによって永続化されていることを確認できます。

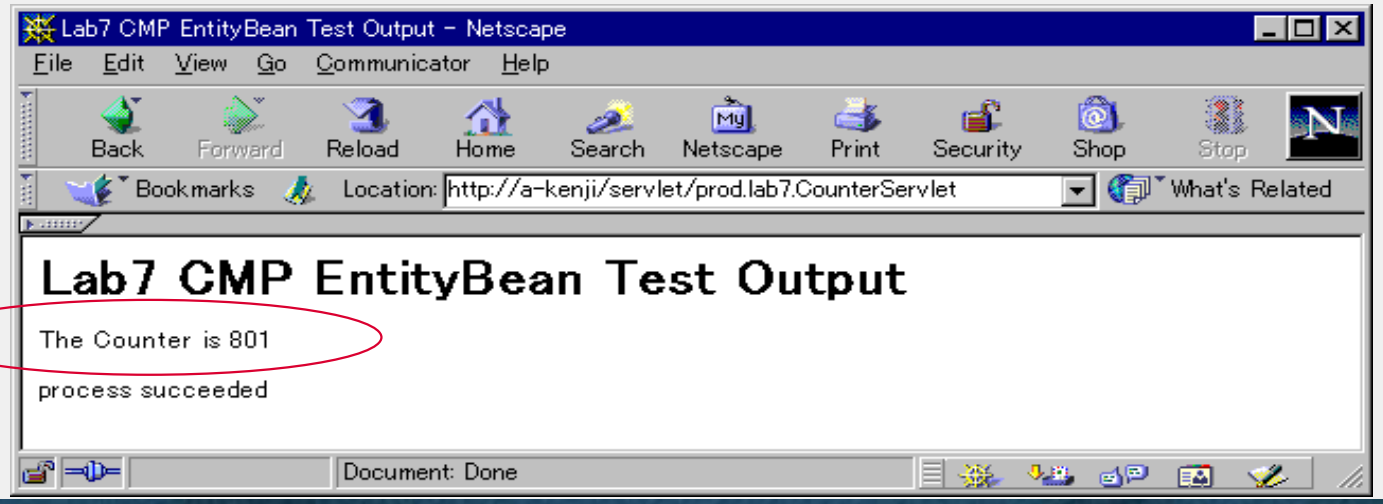
7. データベース内のレコードの内容を確認してください。

EJBの基礎(4)
ServletでCMP EBにアクセス

ステップ3



ステップ6



EJBの基礎(4)
UDBの中身を確認

1. db2インスタンスユーザーで、EJS Serverマシンにログオンします。

2. CMP用DBとして定義したCounterDBに接続します。

```
db2inst1@[/home/db2inst1]db2 connect to counter
Database Connection Information
Database server        = DB2/6000 6.1.0
SQL authorization ID   = DB2INST1
Local database alias   = COUNTER
```

3. 以下のDB2コマンドを実行し、テーブルの存在を確認します。

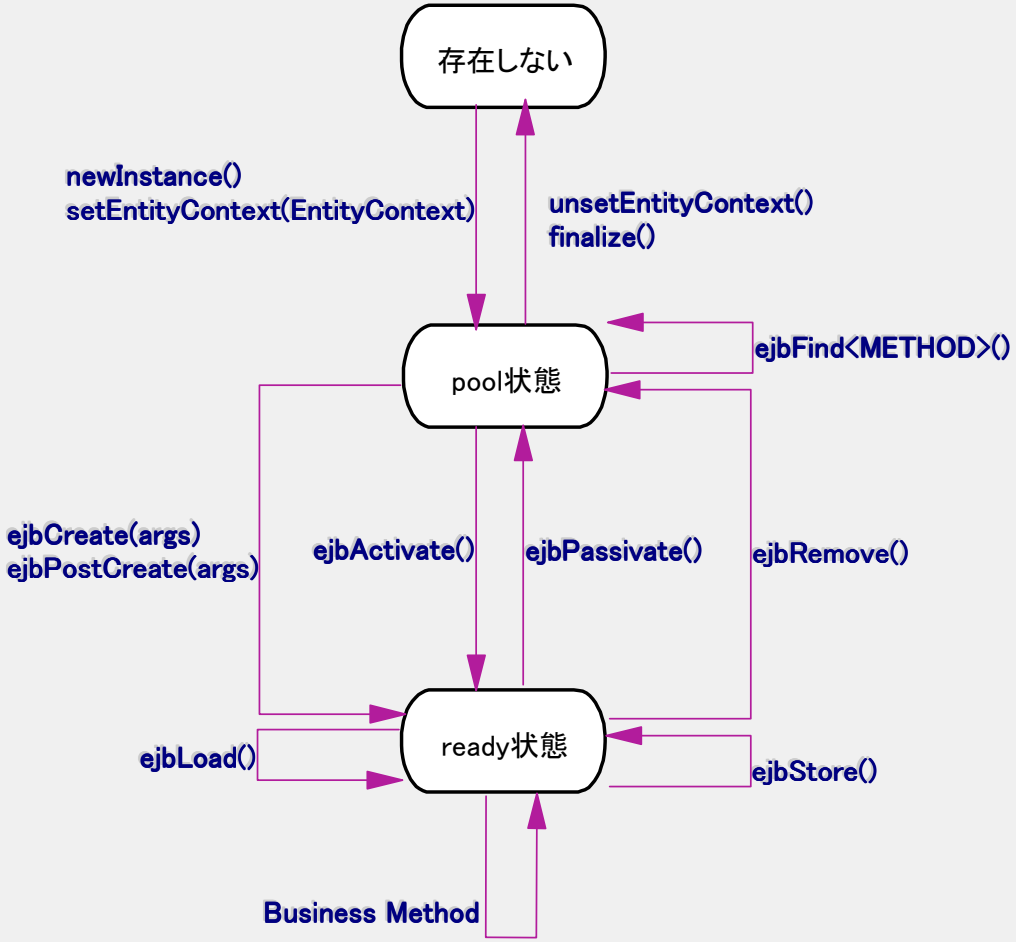
```
db2inst1@[/home/db2inst1]db2 list tables for schema EJB
Table/View                      Schema    Type  Creation time
------------------------------- --------------- ----- --------------------------
COUNTERBEANTBL    EJB             T     2000-08-23-17.06.00.679296
1 record(s) selected.
```

4. 以下のSQLを発行し、レコードの内容を確認します。これが、現在のCounterの値となります。

```
db2inst1@[/home/db2inst1]db2 "select * from EJB.COUNTERBEANTBL"
THECOUNT_             COUNTERKEY_
--------------------  ----------------------------------------
    801                Lab7CounterRecord
1 record(s) selected.
```

- 4より、CPM EBインスタンスが、DB内の1レコードに対するObject的なViewを提供していることを理解することができます。

EJBの基礎(4)
CMP EBインスタンスのライフサイクル



<Entityが存在する場合>

1. CounterBean()　　[インスタンスが不足している場合のみ]
2. setEntityContext()　[インスタンスが不足している場合のみ]

3. ejbFind<METHOD>()
   3-1. ejbActivate()
   3-2. ejbLoad()
   3-3. ejbPassivate()

6. ejbActivate()
7. ejbLoad()
8. Business Method
9. ejbStore()
10. ejbPassivate()

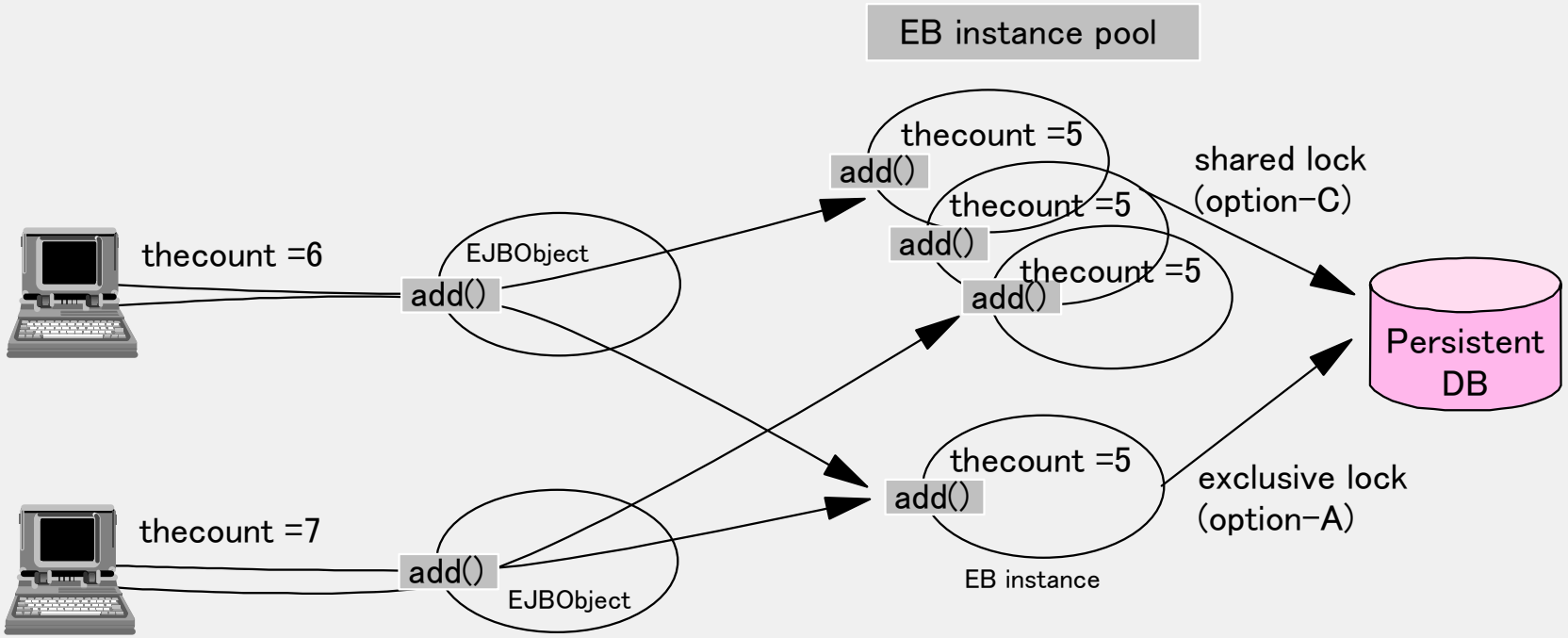
<Entityが存在しない場合>

1. CounterBean()　　[インスタンスが不足している場合のみ]
2. setEntityContext()　[インスタンスが不足している場合のみ]

3. ejbCreate()
4. ejbPostCreate()
5. Business Method
6. ejbStore()
7. ejbPassivate()

CMP EntityBean
OptionAとCの違いを実感 !!

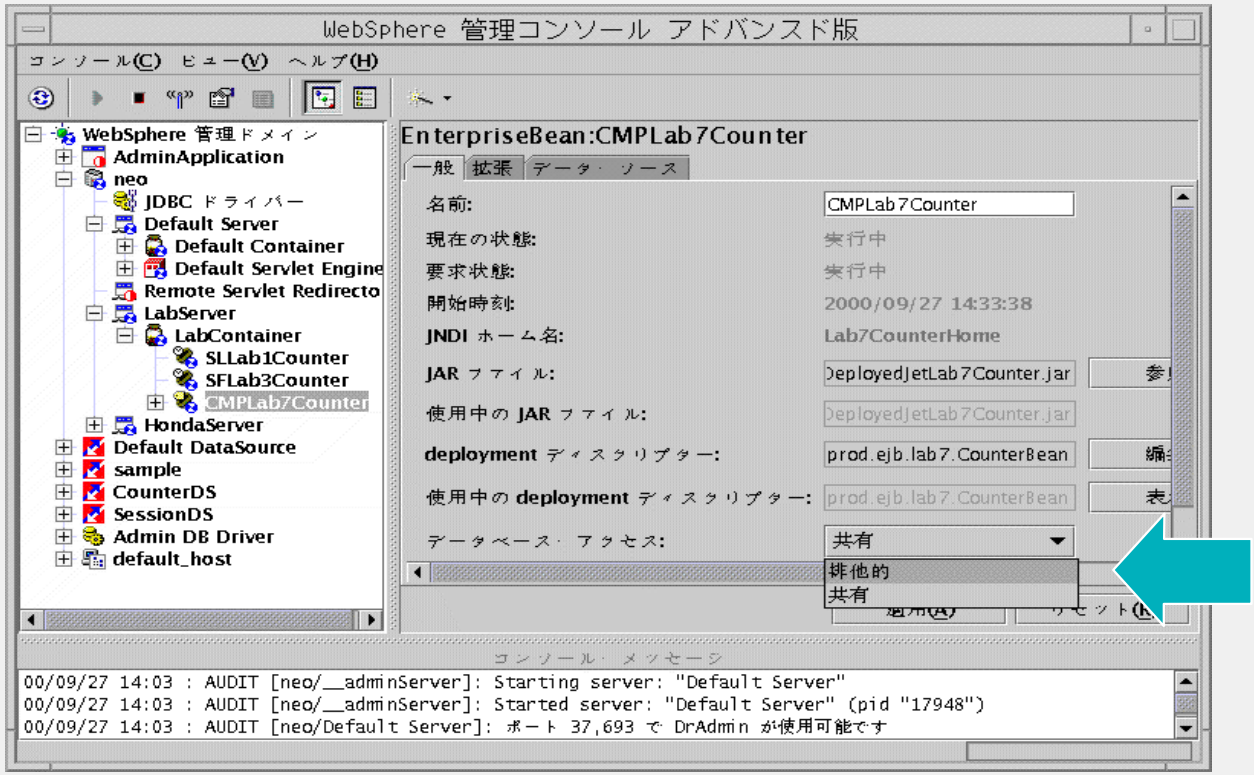
- コミットオプション(A,C)の違いを理解しましょう。
- 特に、オプションAを使用した場合、ejbPassivate/ejbActivateが実行されないことを理解しましょう。



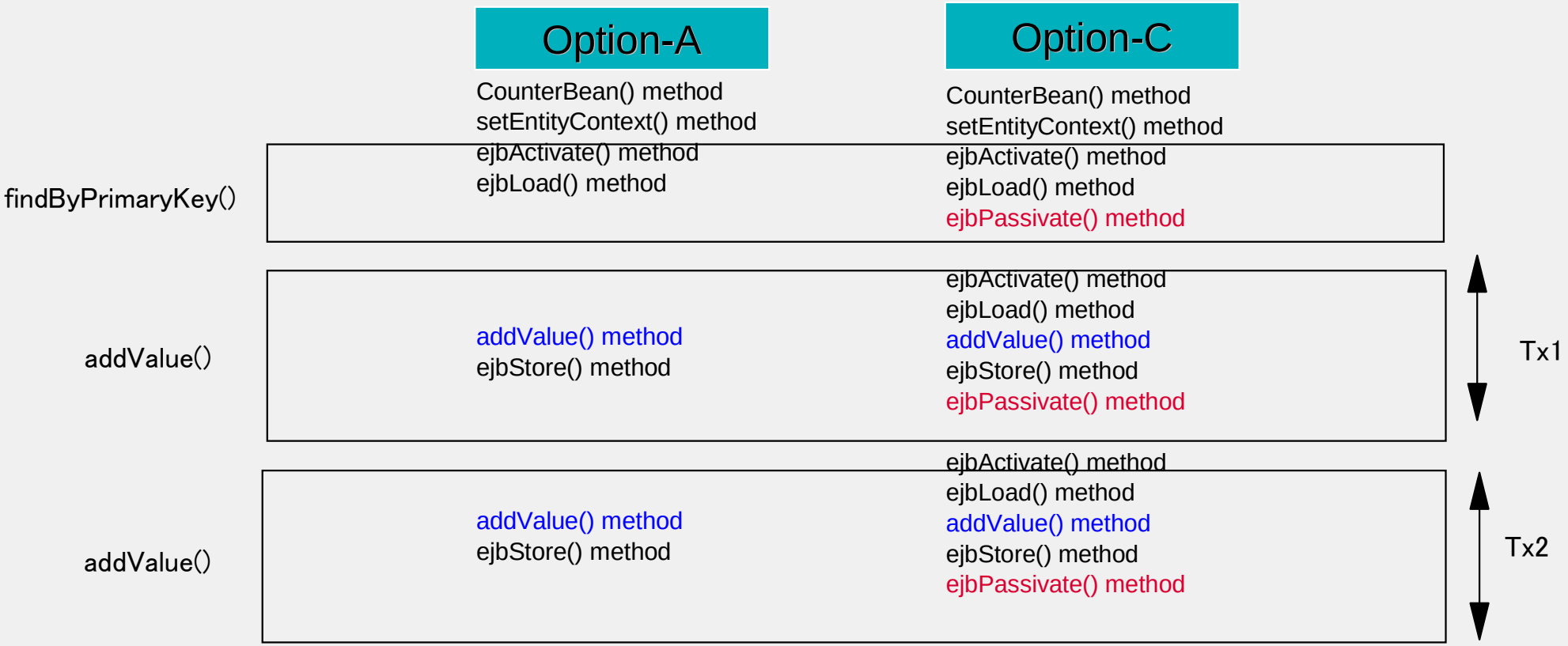
WLM環境では、オプションCしか使用できません。

Optionの変更方法

以下の構成画面において、Option AまたはCを設定することが可能



OptionAとCの違いを実感 !!

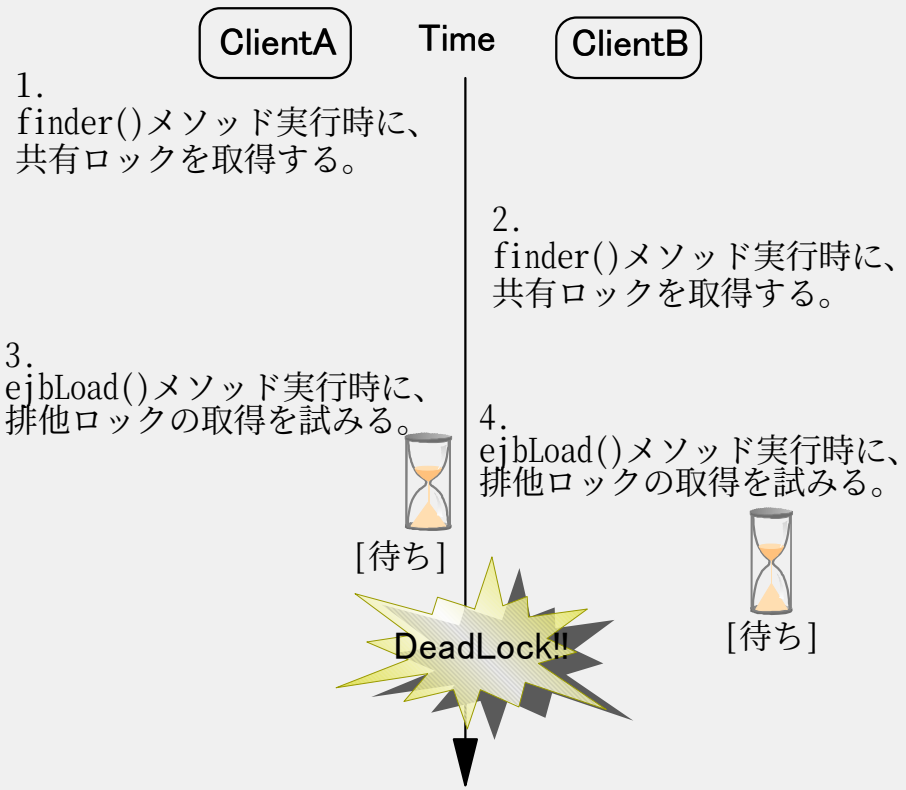


- OptionAでは、Tx1のコミット時にejbPassivate()メソッドが実行されません。
- OptionAでは、EJBContainerが最新のデータを保持している為、Transactionの開始時にejbActivate(),ejbLoad()メソッドが実行されません。

「更新の為の検索」オプションの利用

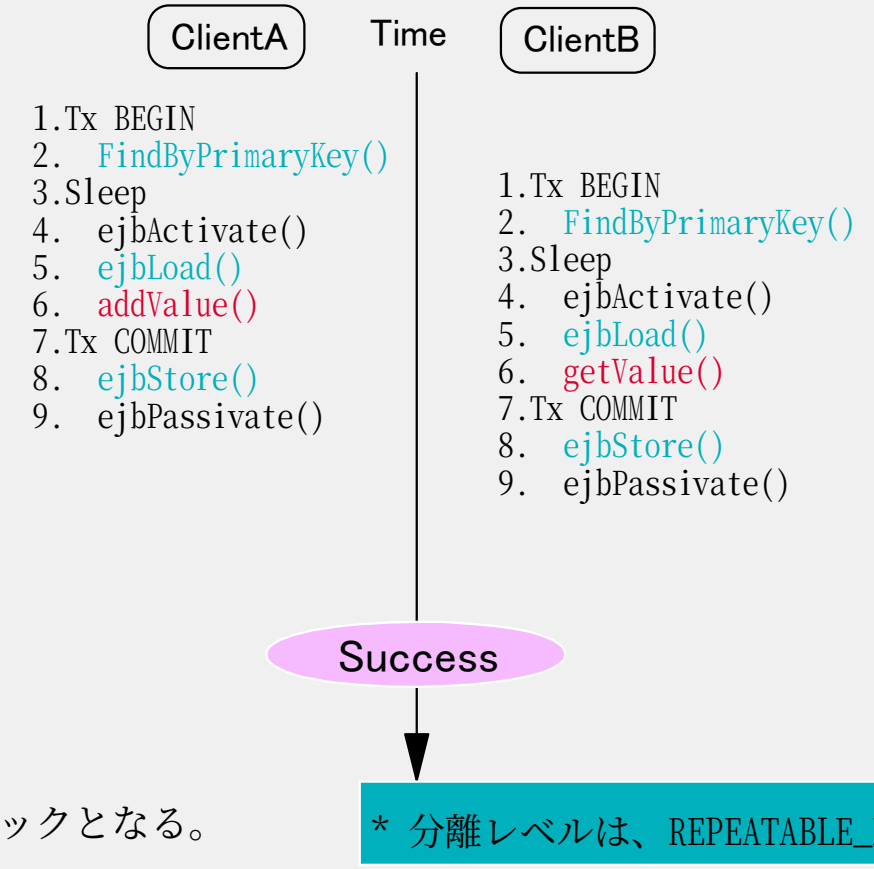
- デットロックを回避する為に、"更新の為の検索"オプションを理解しましょう。

[デットロックとなるケース]



両クライアントが、お互いのロック待ちをする為、デットロックとなる。

[メソッド実行シーケンス]



* 分離レベルは、REPEATABLE_READ

デッドロックの発生 !!

E:\sw224>java prod.lab9.ClientA a-kenji Lab7CounterRecord

Transaction to begin
Get a read lock for findByPrimaryKey()
This thread sleeps for 30 sec. Please start ClientB threads!

Get a write lock for ejbLoad()/ejbStore()
Transaction to commit
The Counter is 768

ClientA

DB2の実装ではデッドロックが発生した場合、一方(ClientA)を継続処理させ、他方(ClientB)をロールバックさせます。

E:\sw224>java prod.lab9.ClientB a-kenji Lab7CounterRecord

Transaction to begin
Get a read lock for findByPrimaryKey()
This thread sleeps for 30 sec.

Get a write lock for ejbLoad()/ejbStore()
Exception: CORBA TRANSACTION_ROLLEDBACK 0 No;
nested exception is:
org.omg.CORBA.TRANSACTION_ROLLEDBACK:

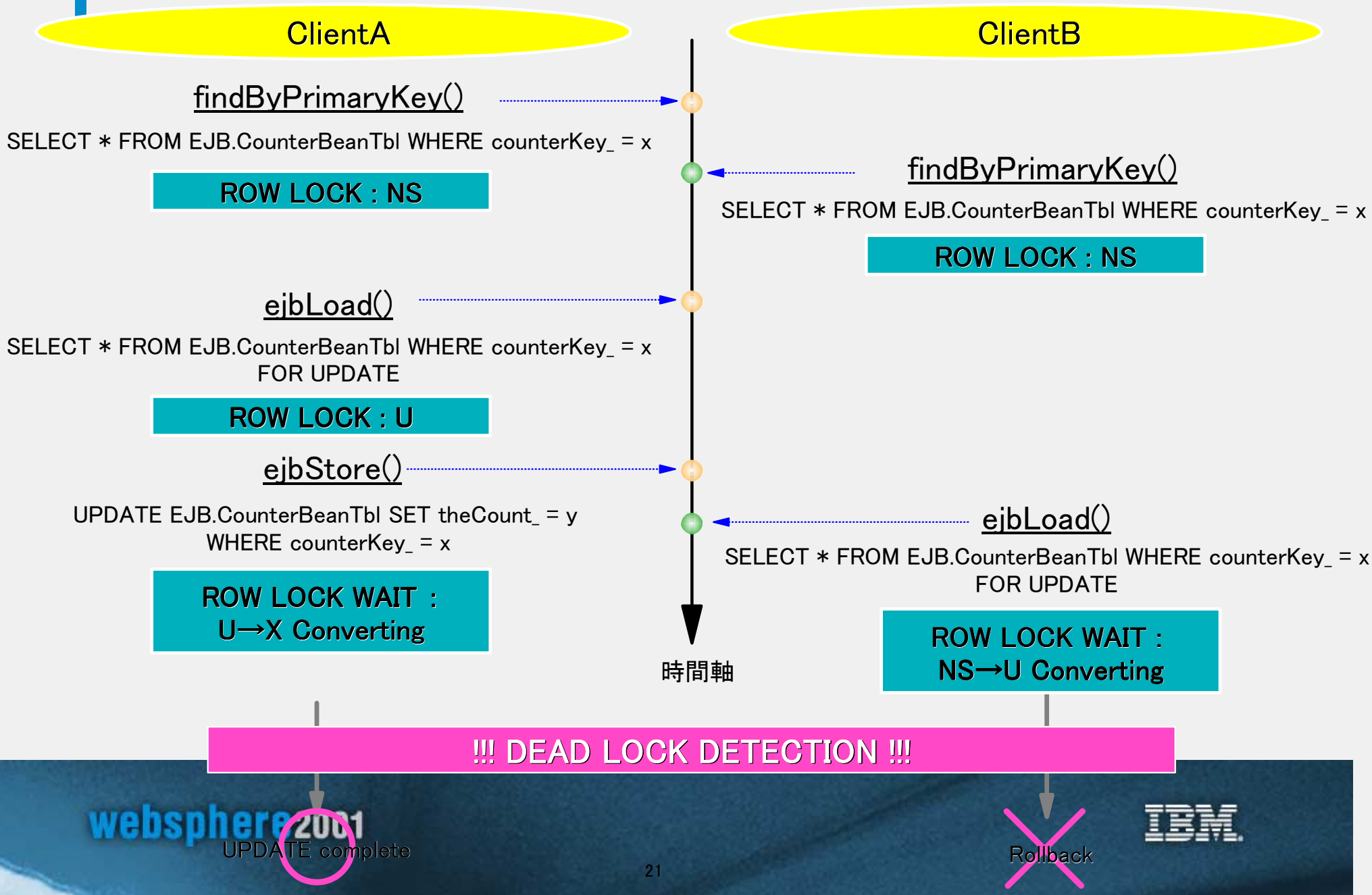
ClientB

DeadLock!!

* 分離レベルは、REPEATABLE_READ

* 分離レベルは、REPEATABLE_READ

デッドロック・シーケンス



ロックモードの互換性（IBM UDB）

- 互換性一覧
  - ○：ロック取得可能
  - ×：ロック取得不可能となり、ロック待ちとなる

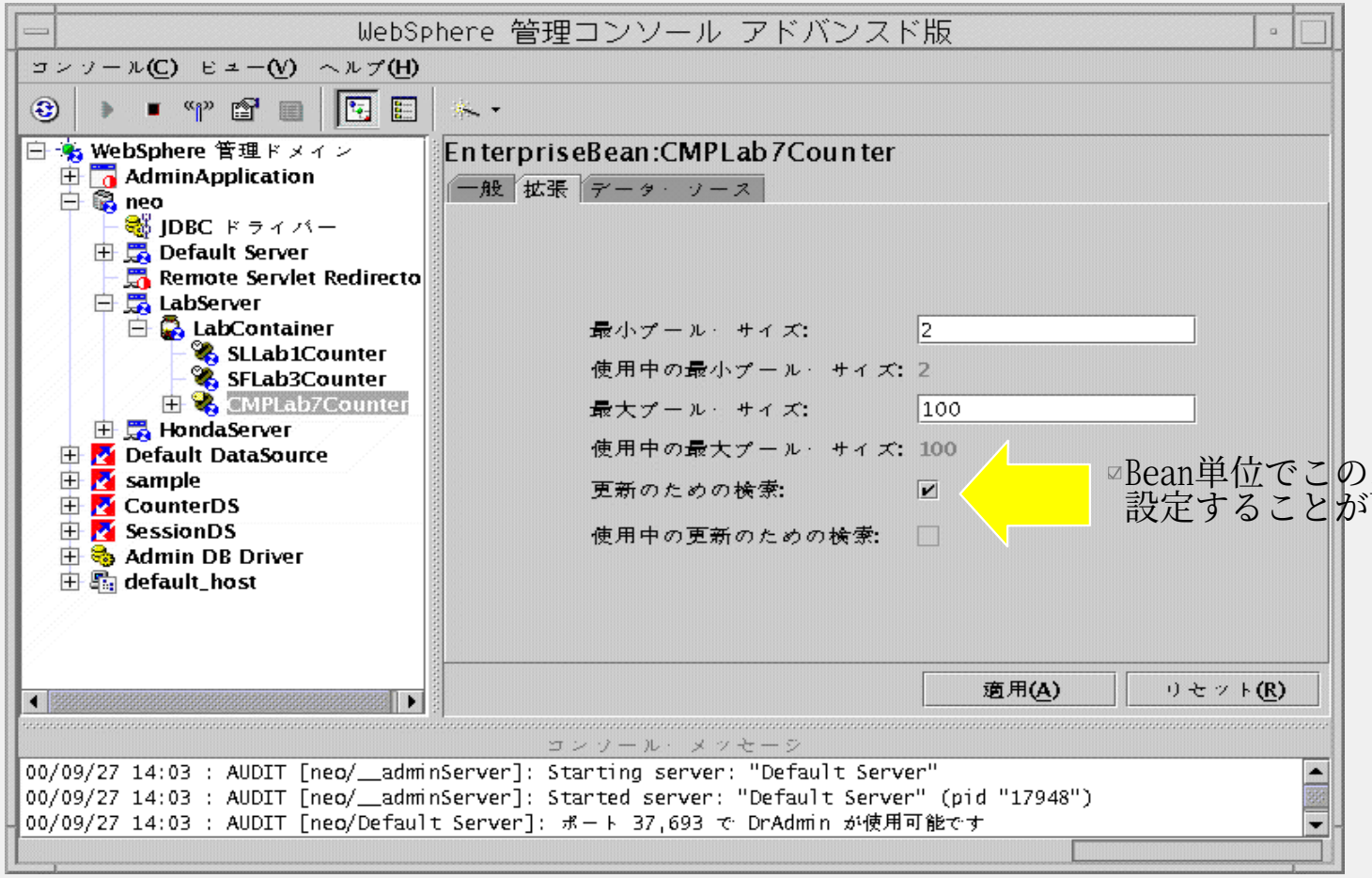
すでに相手に取得されているロック

取得要求されているロック
（自分が取得しようとしているロック）

モード	None	IN	IS	NS	S	IX	SIX	U	NX	X	Z	NW	W
None	○	○	○	○	○	○	○	○	○	○	○	○	○
IN	○	○	○	○	○	○	○	○	○	○	×	○	○
IS	○	○	○	○	○	○	○	○	×	×	×	×	×
NS	○	○	○	○	○	×	×	○	○	×	×	○	×
S	○	○	○	○	○	×	×	○	×	×	×	×	×
IX	○	○	○	×	×	○	×	×	×	×	×	×	×
SIX	○	○	○	×	×	×	×	×	×	×	×	×	×
U	○	○	○	○	○	×	×	×	×	×	×	×	×
NX	○	○	×	○	×	×	×	×	×	×	×	×	×
X	○	○	×	×	×	×	×	×	×	×	×	×	×
Z	○	×	×	×	×	×	×	×	×	×	×	×	×
NW	○	○	×	○	×	×	×	×	×	×	×	×	○
W	○	○	×	×	×	×	×	×	×	×	×	○	×

「更新の為の検索」オプションの構成

以下の画面で、デットロック回避の為の"更新のための検索"オプションを構成できます。



Bean単位でこのオプションを設定することが可能

「更新の為の検索」オプションを使用した場合

E:\sw224>java prod.lab9.ClientA a-kenji Lab7CounterRecord

Transaction to begin
Get a read lock for findByPrimaryKey()
This thread sleeps for 30 sec. Please start ClientB threads!

Get a write lock for ejbLoad()/ejbStore()
Transaction to commit
The Counter is 770

ClientA

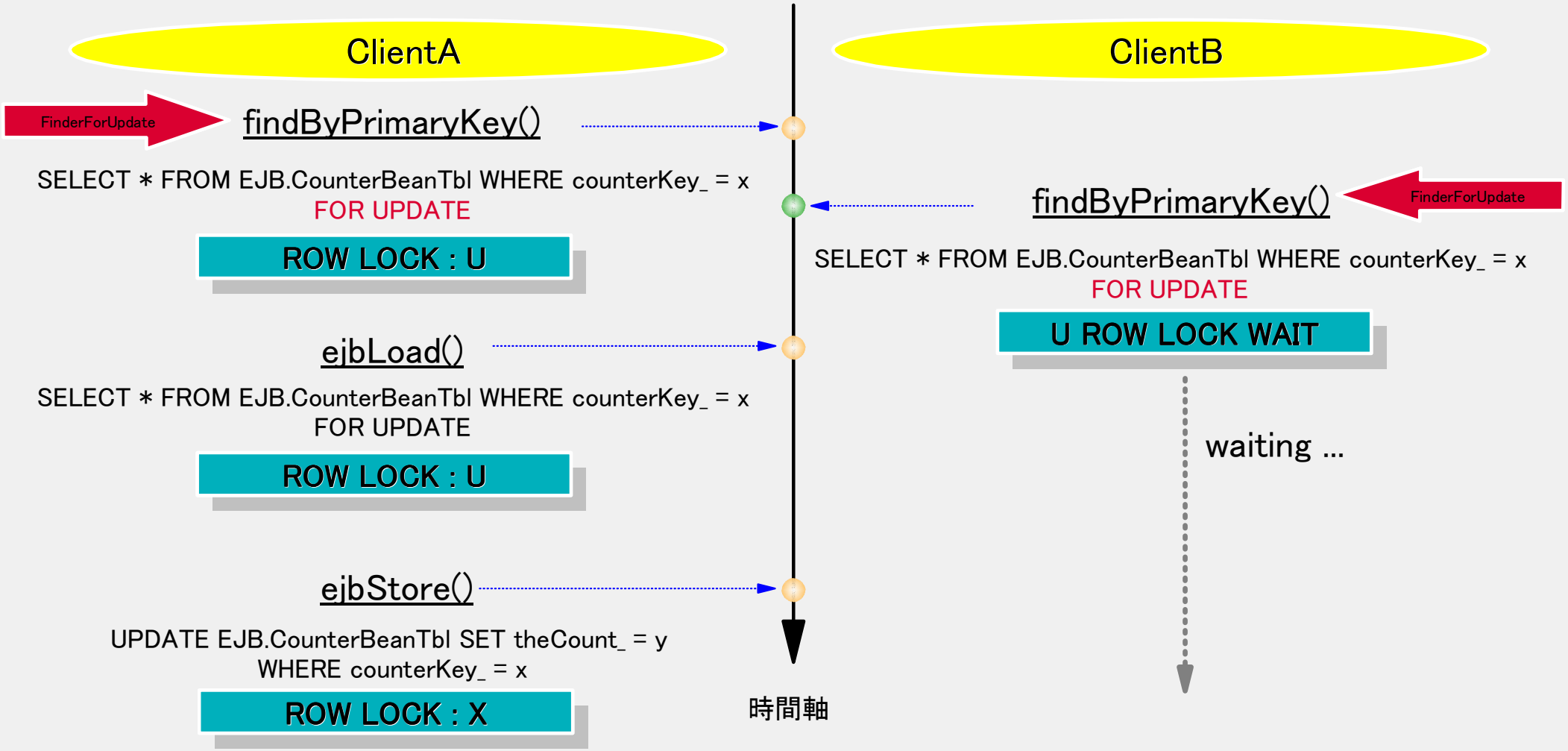
"更新のための検索"オプションを使用しているので、デットロックは回避できることが分かる。
ClientBは、ClientAが終了するまで、findByPrimaryKey()メソッドでロック待ちの状態となる。

E:\sw224>java prod.lab9.ClientB a-kenji Lab7CounterRecord
Transaction to begin
Get a read lock for findByPrimaryKey()
<Lock Wait>
This thread sleeps for 30 sec.

Get a write lock for ejbLoad()/ejbStore()
Transaction to commit
The Counter is 770

ClientB

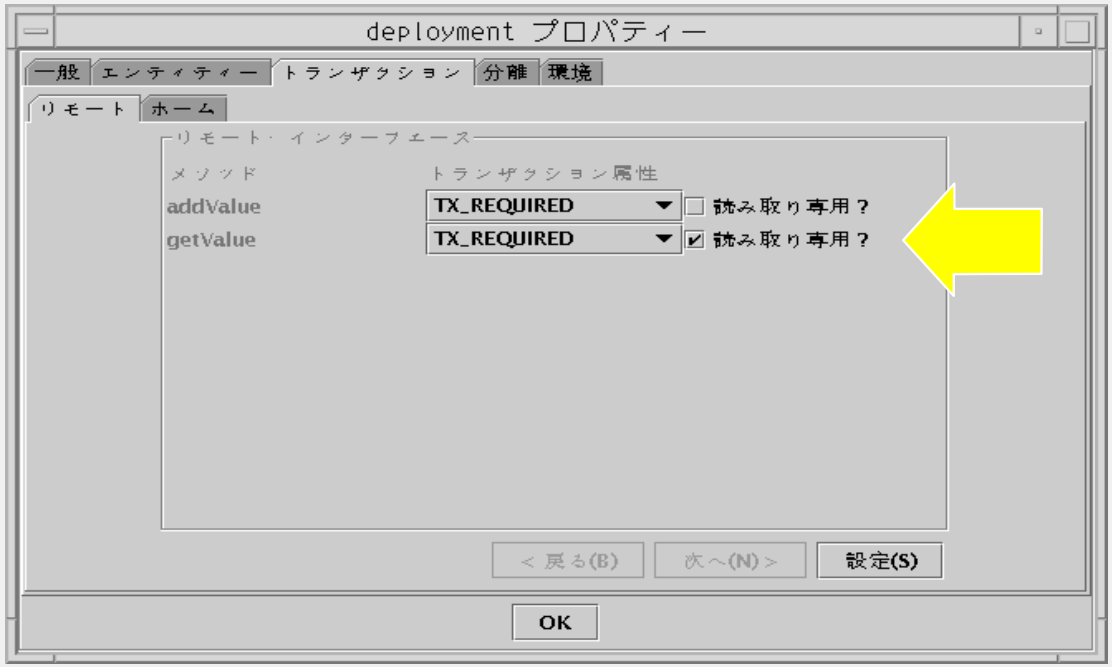
「更新の為の検索」オプションを使用した場合



読み取り専用オプションの利用

- 管理コンソールで、ClientBが使用するgetValue()メソッドを「読み取り専用」に設定する。

  1. ejbLoad()メソッドの起動時に実行されるSELECT文に for update がつかなくなる。
     U lockではなく、NS lockを使用する。
  2. 「読み取り専用」の為、getValue()メソッドの実行後、ejbStoreは実行されない。



読み取り専用オプションの利用

E:\sw224>java prod.lab9.ClientA a-kenji Lab7CounterRecord

Transaction to begin
Get a read lock for findByPrimaryKey()
This thread sleeps for 30 sec. Please start ClientB threads!

Get a write lock for ejbLoad()/ejbStore()
<Lock Wait>
Transaction to commit
The Counter is 774

ClientA

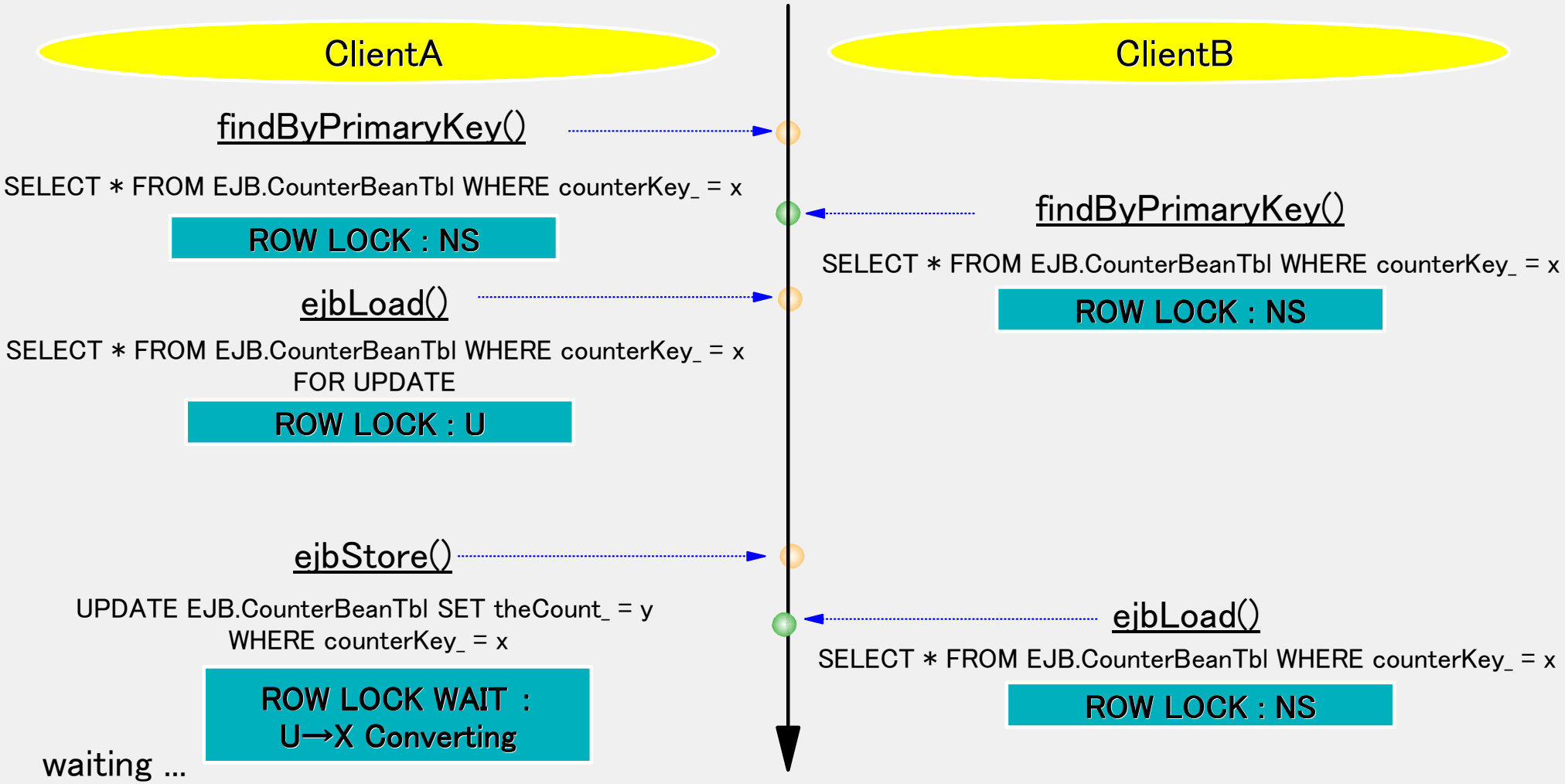
"読み取り専用"オプションを使用しているので、デットロックは回避できることが分かる。
ClientAは、ClientBが終了するまで、ロック待ちの状態となる。

E:\sw224>java prod.lab9.ClientB a-kenji Lab7CounterRecord
Transaction to begin
Get a read lock for findByPrimaryKey()
This thread sleeps for 30 sec.

Get a read lock for ejbLoad() only
Transaction to commit
The Counter is 773

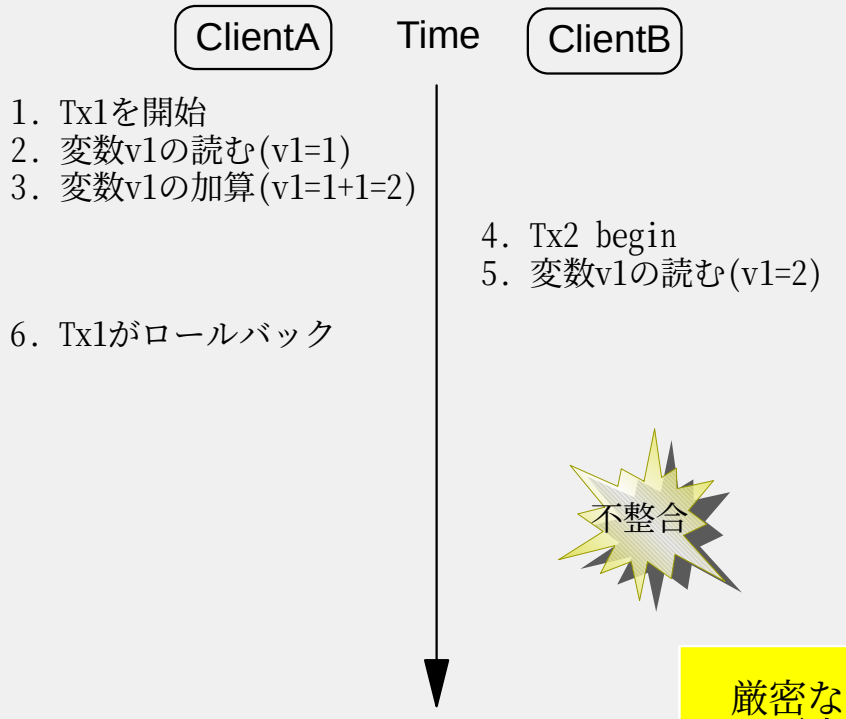
ClientB

読み取り専用オプションの利用



CMP EB Dirty Read シナリオ

- CMP EBにおけるDrity Readシナリオを理解しましょう。



READ_UNCOMMITEDの場合、左図のように、Tx1がロールバックが発生した場合、データの不整合が発生する。

厳密なデータの整合性を要求しない、または高パフォーマンスを要求される場合のみ、READ_UNCOMMITEDが使用される。

READ_UNCOMMITEDでの挙動

```
E:\sw224>java prod.lab10.ClientA a-kenji rollback
Transaction to begin
Read the counter
The Counter is 775
Add the counter
The Counter is 776
This thread sleeps for 30 sec. Please start a ClientB thread !

Transaction to rollback
The Counter is 775
```

ClientA

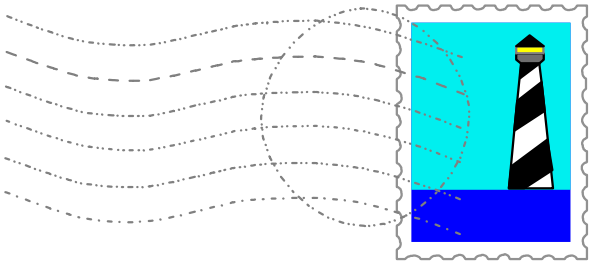
本来、READ_UNCOMMITEDであれば、ClientBは、Counter値として776を読み込むはずですが、
では775を取得します。

```
E:\sw224>java prod.lab10.ClientB a-kenji
Transaction to begin
Read the counter
The Counter is 775
Transaction to commit
The Counter is 775
```

ClientB

EntityBeanでDirty Readはありえない ??

To: 会場の皆様へ



EJB 2.0 spec describes the ambiguous invocation time of ejbStore() and ejbLoad() at Container's view on page170 as following.

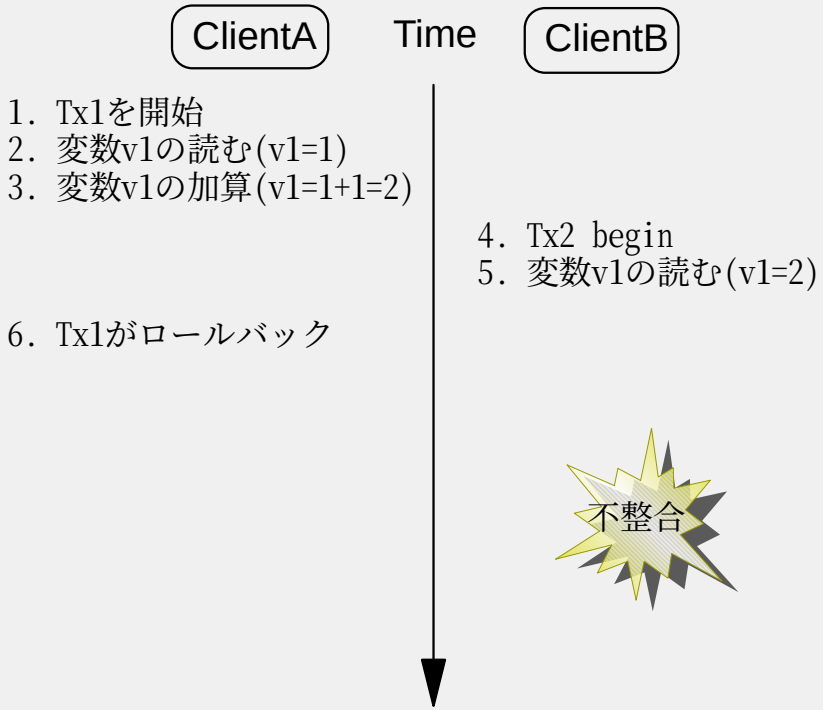
★ ejbStore()
This synchronization always happens at the end of a transaction. However, the container may also invoke this method when it passivates the instance in the middle of a transaction, or when it needs to transfer the most recent state of the entity object to another instance for the same entity object in the same transaction.

★ ejbLoad()
The exact times that the container invokes ejbLoad depend on the configuration of the component and the container, and are not defined by the EJB architecture. Typically, the container will call ejbLoad before the first business method within a transaction.

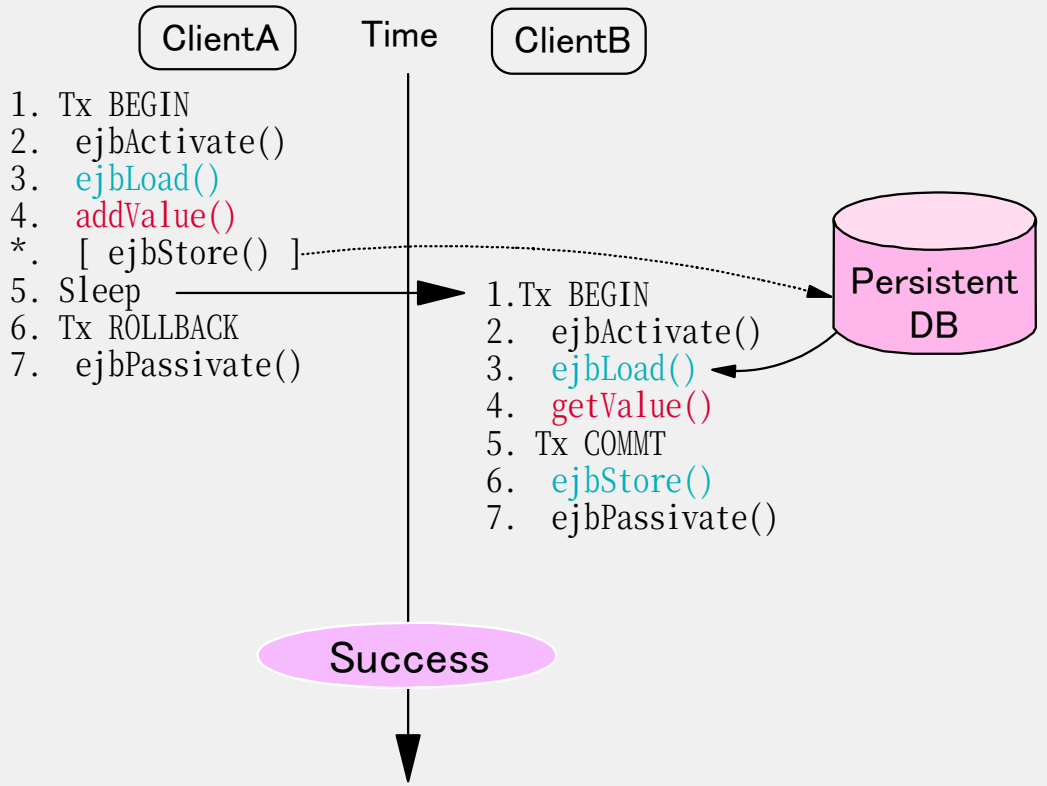
How do you interpret above the sentence, especially in blue line?

EntityBeanでDirty Readはありえない ??

[DirtyReadシーケンス]



[メソッド実行シーケンス]



READ_COMMITEDでの挙動

次に、同一のシナリオをREAD_COMMITED分離レベルで確認します。

```
E:\sw224>java prod.lab10.ClientA a-kenji rollback
Transaction to begin
Read the counter
The Counter is 775
Add the counter
The Counter is 776
This thread sleeps for 30 sec. Please start a ClientB thread !

Transaction to rollback
The Counter is 775
```

ClientA

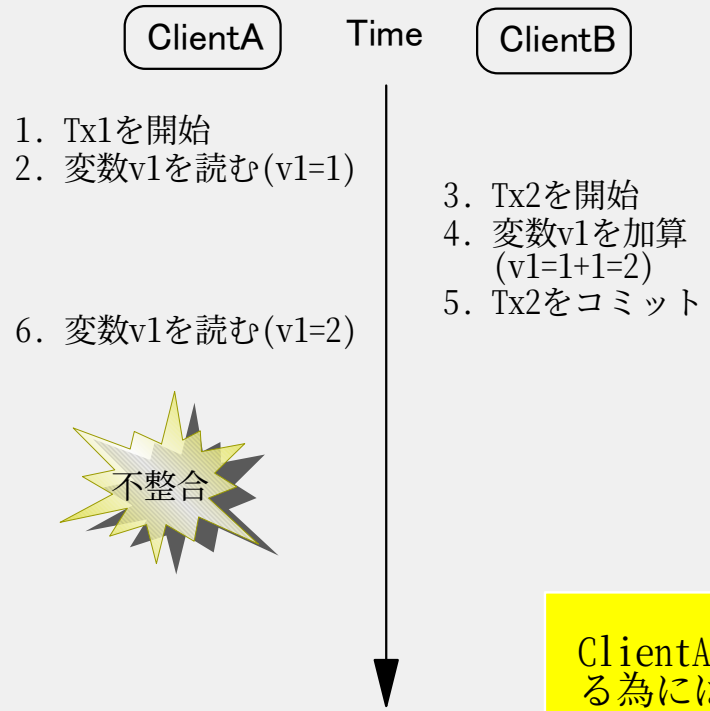
READ_COMMITEDでは、正しく挙動していることを確認できます。

```
E:\sw224>java prod.lab10.ClientB a-kenji
Transaction to begin
Read the counter
The Counter is 775
Transaction to commit
The Counter is 775
```

ClientB

CMP EB Unrepeatable Read シナリオ

- CMP EBにおけるUnrepeatable Readシナリオにおける挙動を理解しましょう。



READ_COMMITEDの場合、左図のように、ClientAが再度読み込みを行う(Step5)と、前回と異なる値を取得することになる。

ClientAのトランザクションにおいて、反復読み込みを可能にする為には、REPEATBLE_READ分離レベルが必要となります。

READ_COMMITEDでの挙動

ClientA

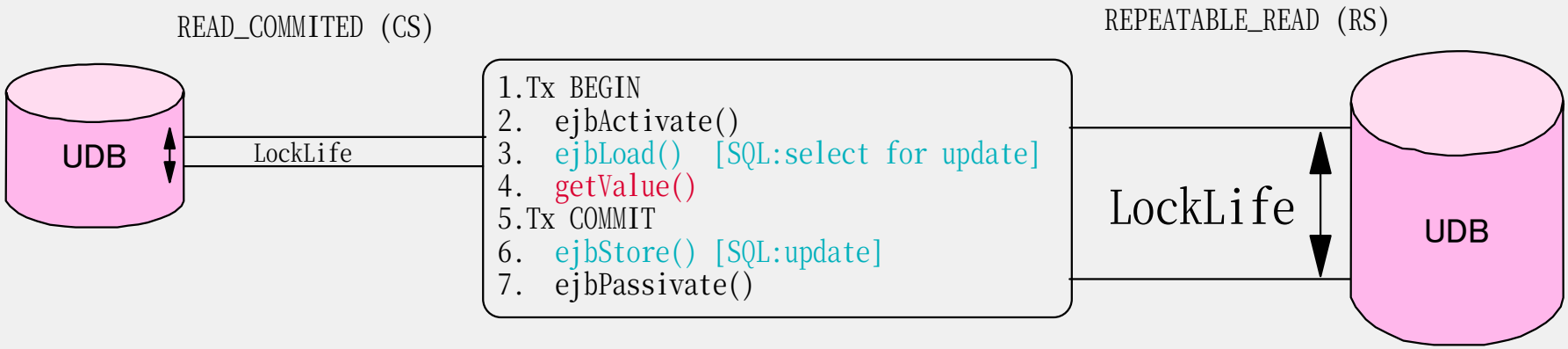
```
E:\sw224>java prod.lab11.ClientA a-kenji
Transaction to begin
Read the counter
First getValue() method returns 777
This thread sleeps for 30 sec. Please start a ClientB thread!

Read the counter
Second getValue() method returns  777
Transaction to commit
The Counter is 777
```

本来、READ_COMMITEDであれば、ClientAの反復した読み込み値は異なるはずであるが、
同一の値を取得できています。

ClientB

```
E:\sw224>java prod.lab11.ClientB a-kenji
Transaction to begin
addValue() method returns 778
Transaction to commit
The Counter is 778
```

EntityBeanでREAD_COMMITEDは危険 !?



1. READ_COMMITTED(UDBのCS)では、ejbLoadのselect for updateの実行時点でのみUロックが取得され、処理後そのUロックは外れます。

2. これがREPEATABLE_READ(UDBのRS)になると、ejbLoadのselect for updateで取得したUロックが、トランザクションがCOMMITされるまで維持されます。

►今回のケースでは、実装したCMP EntityBeanのビジネスロジック(getValue)は、ビジネスロジック実行中に対象行を、
他のアプリに更新されては問題になるロジックである。つまり、

『SELECTされた結果が、自分が更新する前に誰かに更新されないように』

という意図を持ったejbLoad (select for update)を必要とします。このようなビジネスロジックの場合、REPEATABLE_READが必須となります。

＊この事象は、DBベンダーの実装(Oracleなど)にも依存します。本資料は、UDB V6.1を使用して作成しています。

REPEATABLE_READの挙動

次に、同一のシナリオをREPEATABLE_READ分離レベルで確認します。

ClientA

```
E:\sw224>java prod.lab11.ClientA a-kenji
Transaction to begin
Read the counter
First getValue() method returns 777
This thread sleeps for 30 sec. Please start a ClientB thread!

Read the counter
Second getValue() method returns  777
Transaction to commit
The Counter is 778
```

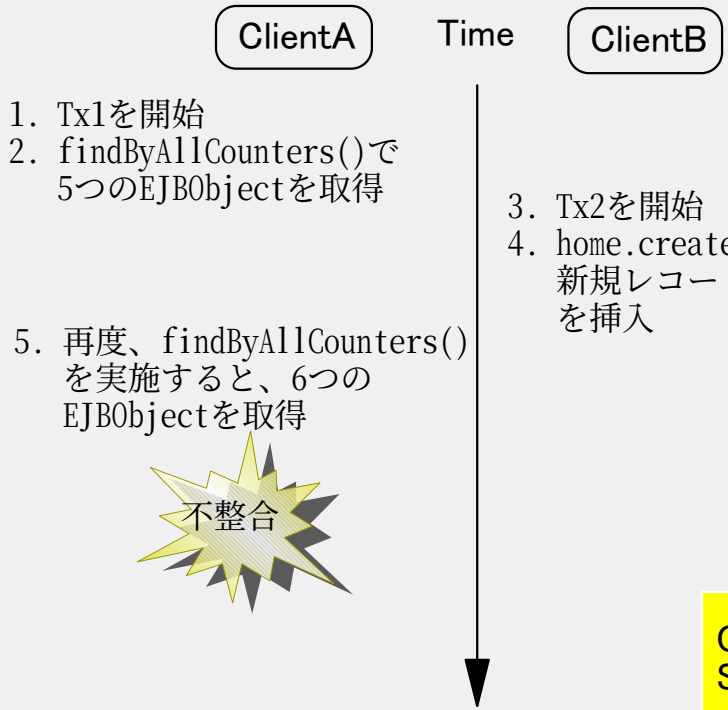
REPEATABLE_READでは、正しく挙動していることを確認できます。

ClientB

```
E:\sw224>java prod.lab11.ClientB a-kenji
Transaction to begin
<LOCK WAIT>

addValue() method returns 778
Transaction to commit
The Counter is 778
```

Phantom Read シナリオ

- CMP EBにおけるPhantom Readシナリオを理解しましょう。



REPEATABLE_READの場合、新規レコードの挿入が可能です。

ClientAのfindByAllCounters()メソッドの戻り値を同一にするには、SERIALIZABLE分離レベルを必要となります。

REATABLE_READ での挙動

ClientA

```
E:\sw224>java prod.lab12.ClientA a-kenji
The number of records = 2
This thread sleeps for 30 sec. Please start ClientB threads!

The number of records = 3
Transaction to commit
```

同一トランザクション内で、一回目の検索結果が2レコードで、二回目の検索結果が3レコードとなっていることから、Phantom Readしていることが確認できます。

ClientB

```
E:\sw224>java prod.lab12.ClientB a-kenji newrecord1
The number of records = 2 before inserting a new record
Insert the new record = newrecord1
The number of records =3
Transaction to commit
```

SERIALIZABLEでの挙動

ClientA

```
E:\sw224>java prod.lab12.ClientA a-kenji
The number of records = 3
This thread sleeps for 30 sec. Please start ClientB threads!

The number of records = 3
Transaction to commit
```

同一トランザクション内で、一回目の検索結果と二回目の検索結果が3レコードとなっていることから、Phantom Readしていないことを意味します。

また、ClientBは、Insert処理(EJBHome.create() Methodの実行)をする前にLockWaitします。

ClientB

```
E:\sw224>java prod.lab12.ClientB a-kenji newrecord1
The number of records = 3 before inserting a new record
<Lock Wait>
Insert the new record = newrecord1
The number of records =4
Transaction to commit
```