

# Enterprise Java (BI-EJA)

## Technologie programování v jazyku Java (X36TJV)

Ing. Zdeněk Troníček, Ph.D.

Katedra softwarového inženýrství

Fakulta informačních technologií ČVUT v Praze



Letní semestr 2010/2011, přednáška č. 5

<https://edux.fit.cvut.cz/courses/BI-EJA>

<https://edux.feld.cvut.cz/courses/X36TJV>

© Zdeněk Troníček, 2011

# Agenda

- Transakce
  - Řízení
  - Zámky
  - Izolační úrovně
- Security

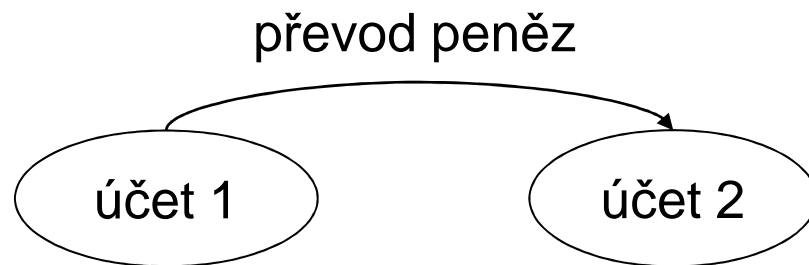
# Transakce (opak.)

Transakce je posloupnost operací, která se tváří jako jedna velká, atomická operace.

BEGIN

COMMIT

ROLLBACK



nákup v elektronickém obchodě:

- 1) platnost karty
- 2) zboží na skladě
- 3) platba kartou
- 4) expedice zboží

# Programové řízení transakcí

@Stateful

@TransactionManagement( TransactionManagementType.BEAN )

public class ManagerBean implements ManagerLocal {

@Resource UserTransaction ut;

public void addAccount( String type ) throws Exception {

try {

ut.begin();

...

ut.commit();

} catch ( Exception e ) {

ut.rollback();

}

}

}

# Deklarativní řízení transakcí (1)

@Stateful

```
public class ManagerBean implements ManagerLocal {
```

```
    @TransactionAttribute( TransactionAttributeType.REQUIRES_NEW )
```

```
    public void addAccount( String type ) {
```

```
        ...
```

```
    }
```

```
}
```

Transaction attribute types:

REQUIRED

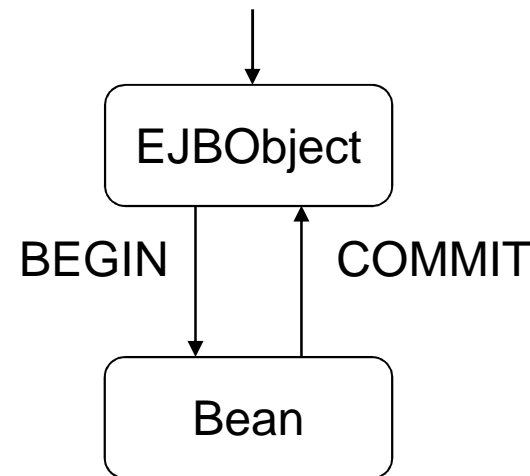
REQUIRES\_NEW

MANDATORY

NEVER


NOT\_SUPPORTED

SUPPORTS



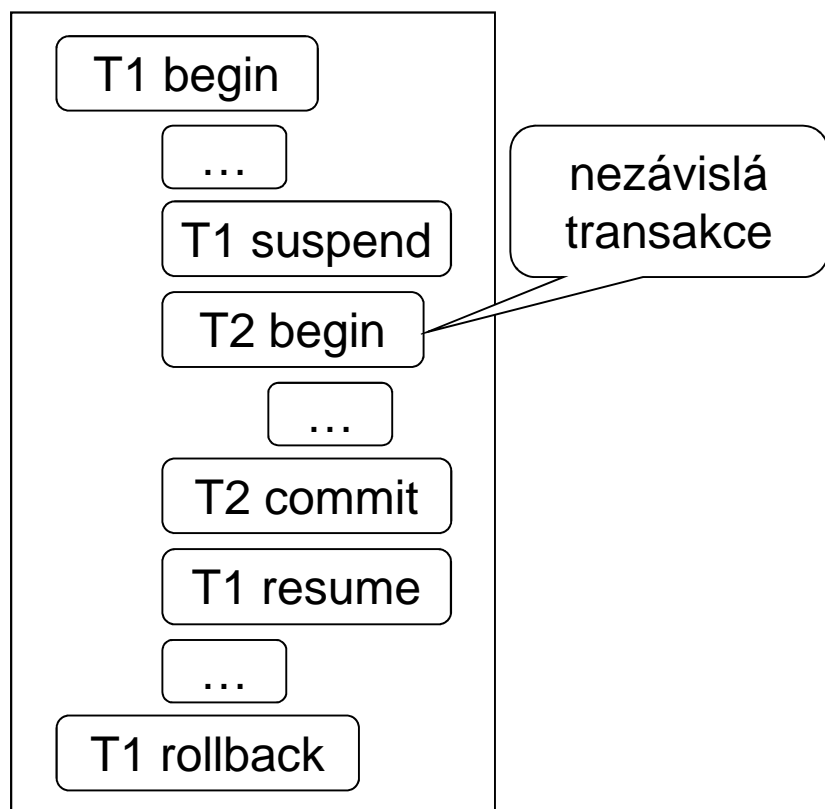
# Deklarativní řízení transakcí (2)

```
@Stateful
public class ManagerBean implements ManagerLocal {
    @Resource SessionContext ctx;
    ...
    public void addAccount( String type ) {
        if( ... ) {
            ctx.setRollbackOnly();
        }
        if( ... ) {
            throw new EJBException( ... );
        }
    }
}
```

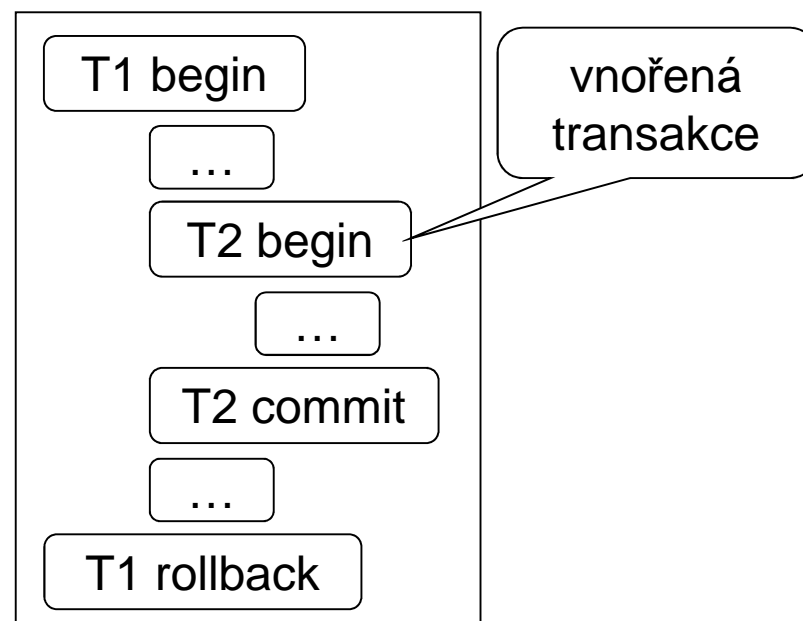


# Transakční model

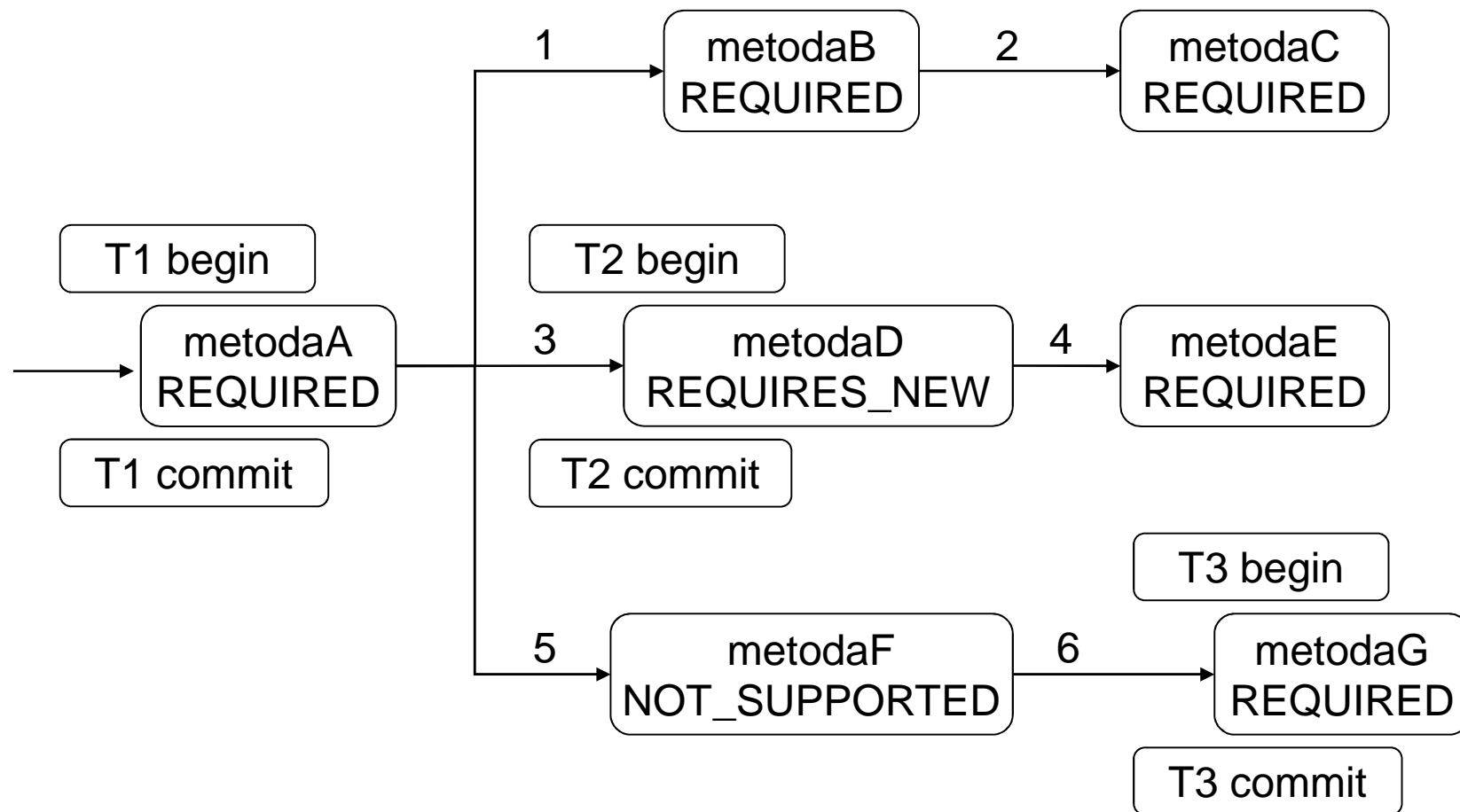
Flat model (JEE)



Nested model



# Transakce - příklad





# Zamykání dat (JavaDB)

## Zámky

Shared (S) – pro čtení  
Exclusive (X) – pro zápis

↓  
čas

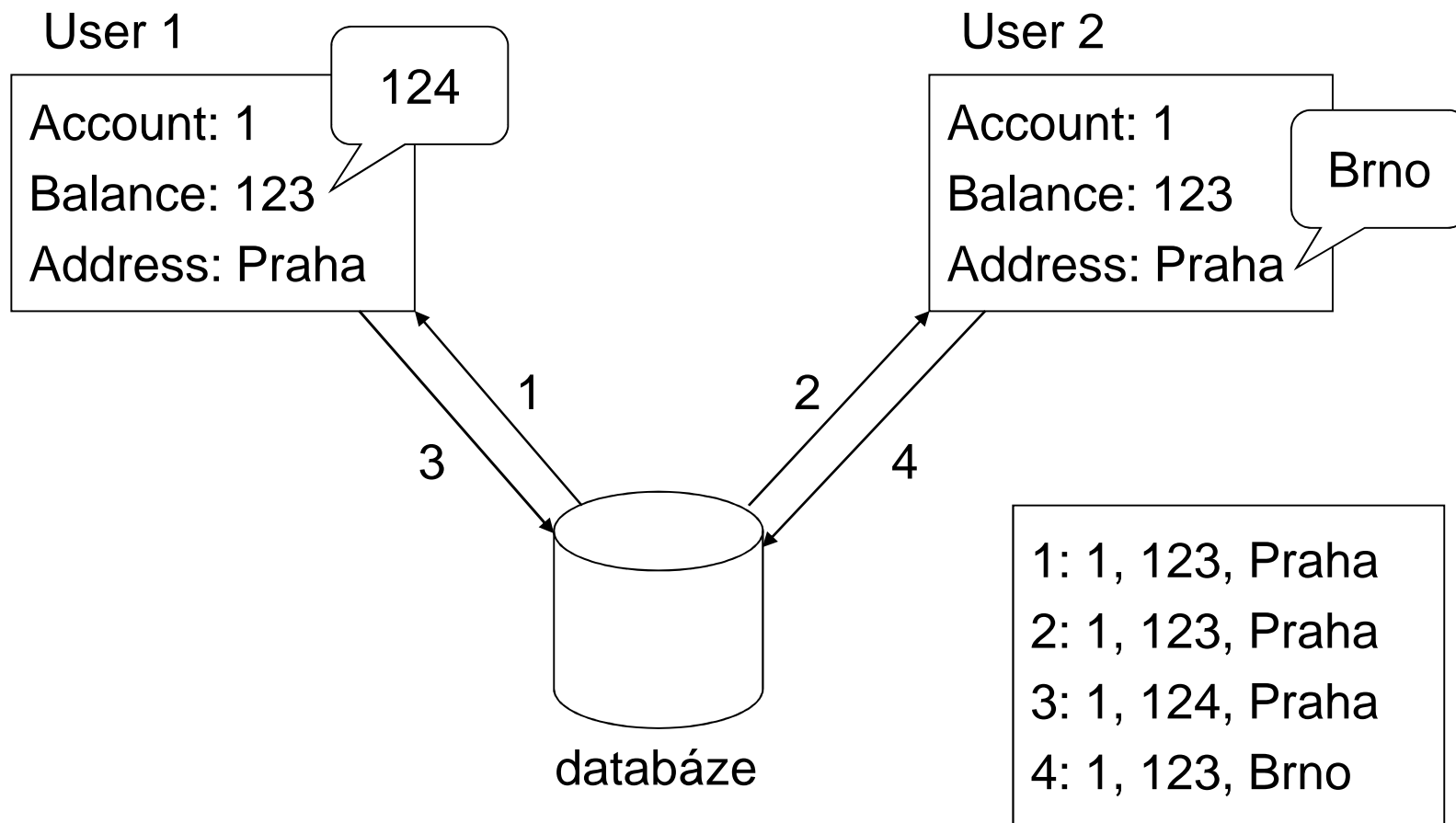
### Transakce 1

```
begin
zamkne řádek A (zámek X)
...
commit (uvolní zámek)
```

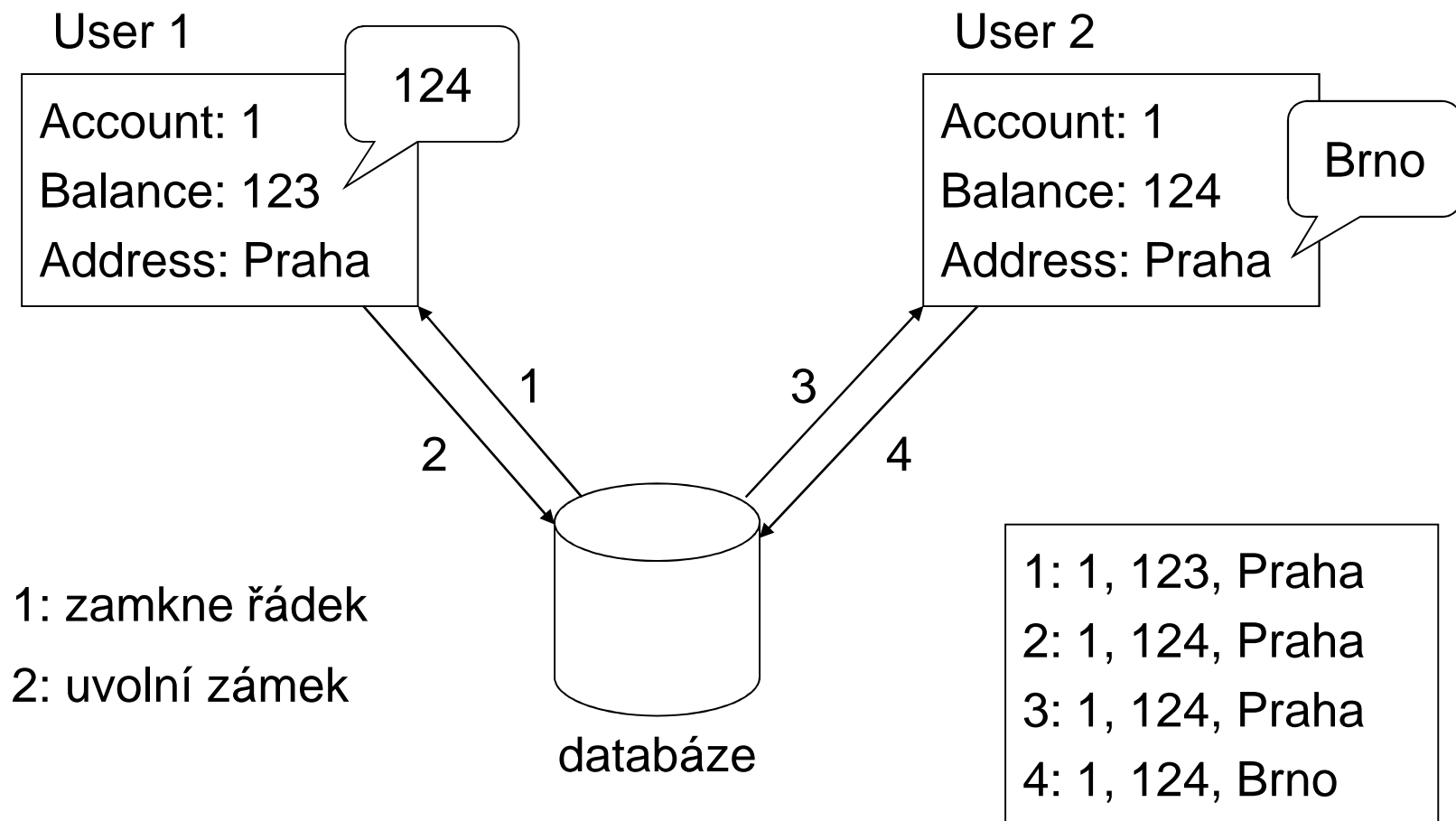
### Transakce 2

```
begin
čeká na zámek k řádku A
...
zamkne řádek A (zámek X)
...
```

# Lost Update

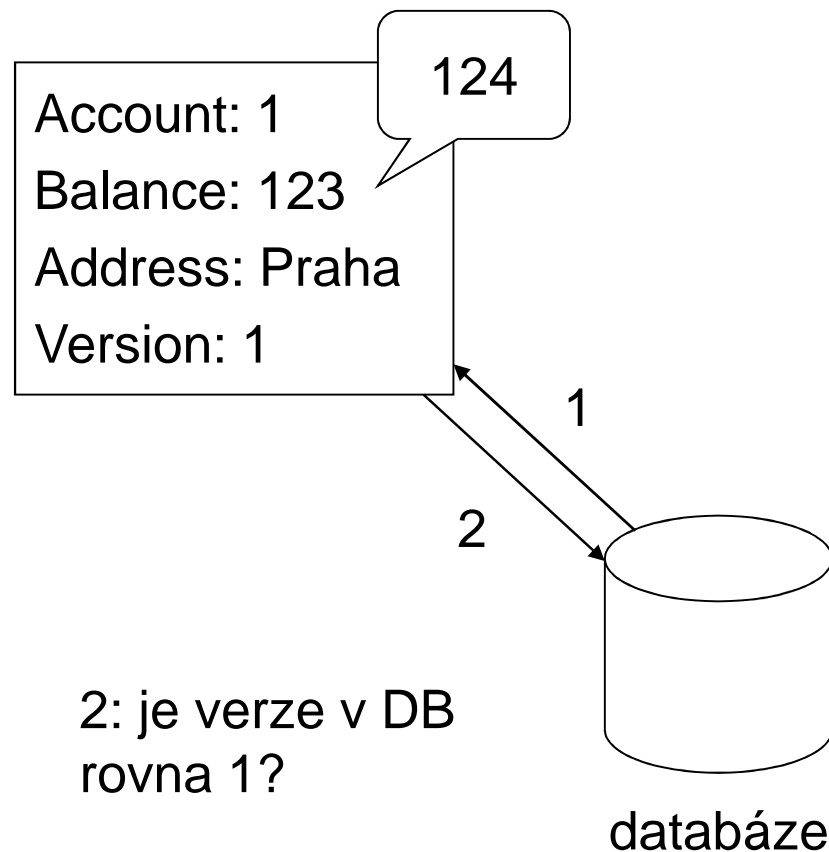


# Pessimistic Locking

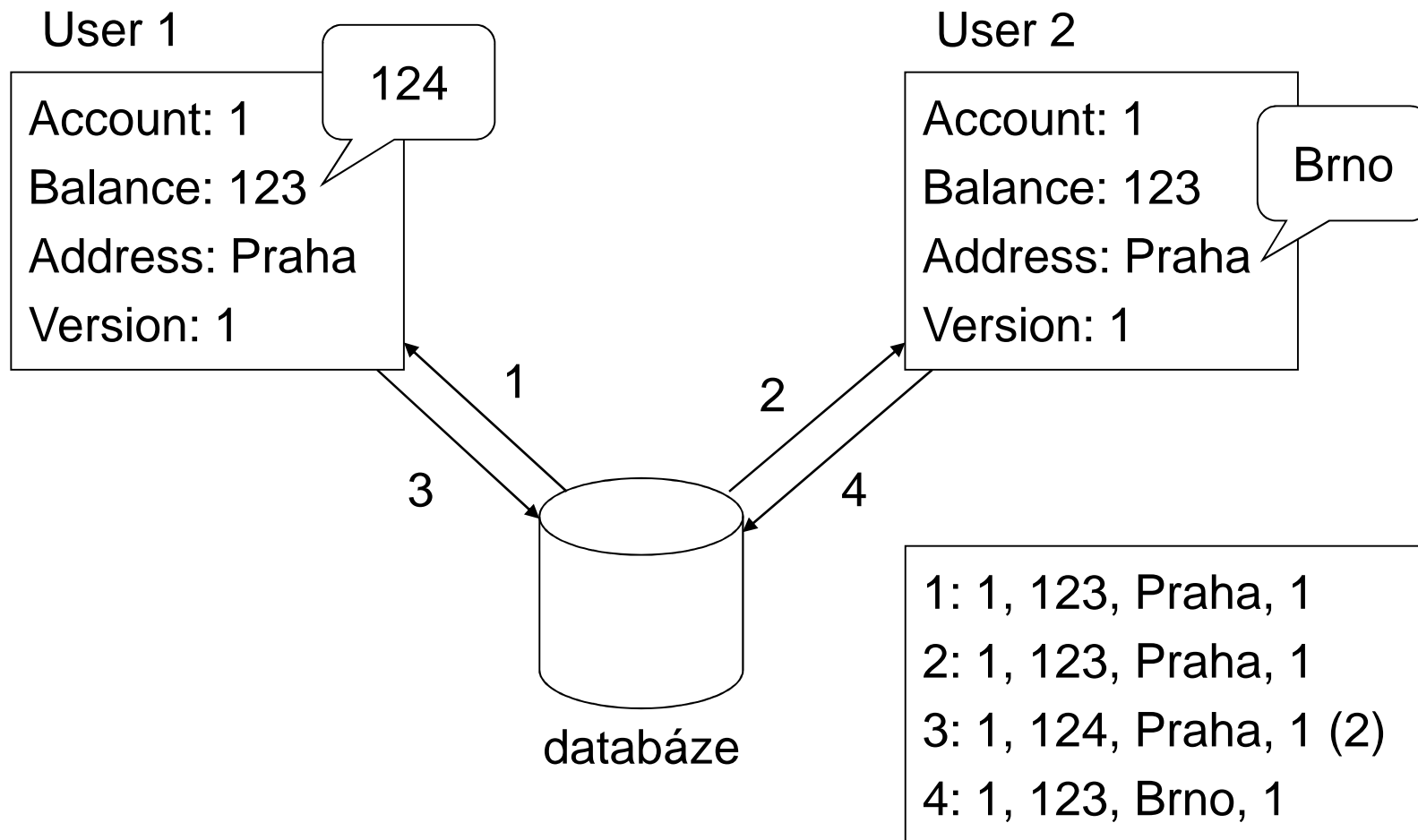


# Optimistic Locking (1)

- přidáme sloupec „verze“
- před každou změnou zkontrolujeme číslo verze: pokud nesouhlasí, změnu odmítneme
- po každé změně zvýšíme číslo verze o 1



# Optimistic Locking (2)



# Java Persistence API

```
@Entity
public class Account {
    @Version
    private Long version;
    ...
}
```

## Podporované typy

- short, Short
- int, Integer
- long, Long
- Timestamp

```
@PersistenceContext
private EntityManager em;

// may throw OptimisticLockException
public void updateAccount( Account acc ) {
    em.merge( acc );
}
```

# Proč potřebujeme transakce?

HOSPODY

NÁZEV	PIVO	CENA
Bar 11	Budvar	20
Bar 11	Kozel	25

současně:

Rumcajs: (max) (min)

Manka: (del) (ins)

SELECT MAX(CENA) FROM HOSPODY WHERE NÁZEV='Bar 11'  
SELECT MIN(CENA) FROM HOSPODY WHERE NÁZEV='Bar 11'

(max)

(min)

DELETE FROM HOSPODY WHERE NÁZEV='Bar 11'  
INSERT INTO HOSPODY VALUES( 'Bar 11', 'Radegast', 30 )

(del)

(ins)

# Řazení operací

Nepoužijeme-li transakce, pak jediná omezení jsou:

- (max) před (min)
- (del) před (ins)

Může tedy nastat: (max) (del) (ins) (min)

Bar 11	Budvar	20
Bar 11	Kozel	25

(max): 25

(del)

(ins)

Bar 11	Radegast	30
--------	----------	----

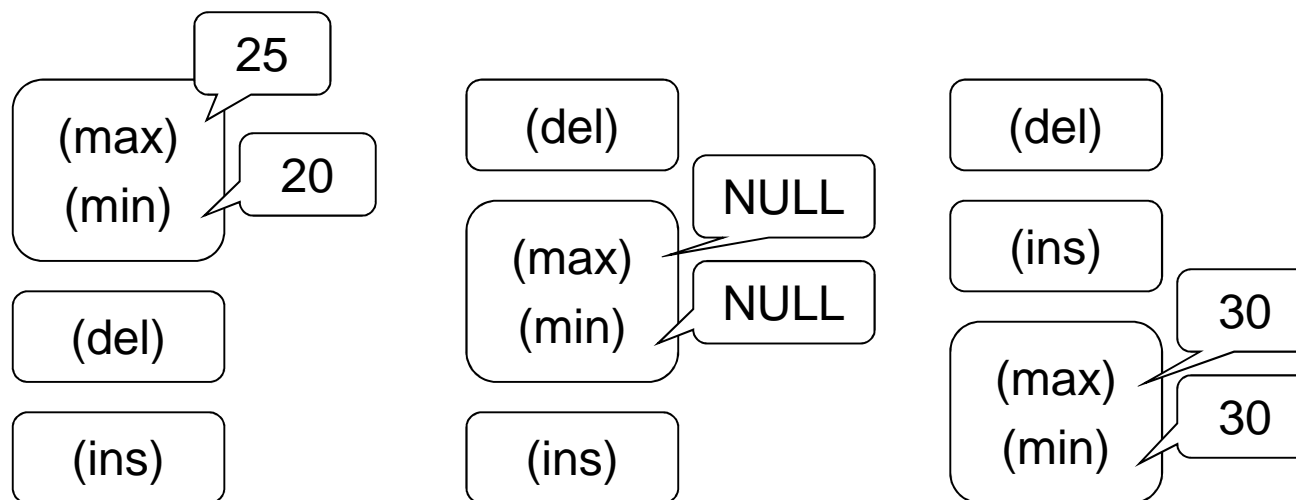
(min): 30

max < min



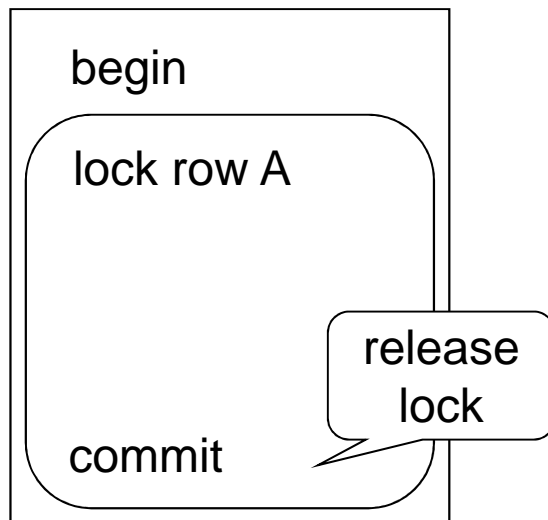
# Řešení: transakce

(max) a (min) provedeme v transakci



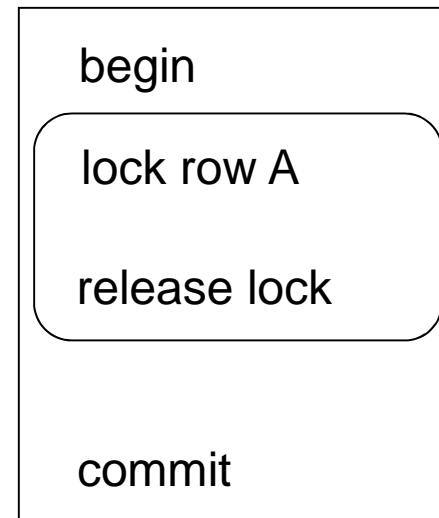
# Shared Locks

Long duration



zámky držíme do  
konce transakce

Short duration



zámky uvolníme  
bezprostředně  
po vyčtení dat

# Phantom

ACCOUNT	
ID	BALANCE
1	101
2	50
3	105

T1 BEGIN

SELECT \* FROM ACCOUNT  
WHERE BALANCE > 100

vybere 1  
řádek

T2 BEGIN

INSERT INTO ACCOUNT  
VALUES( 3, 105 )

T2 COMMIT

SELECT \* FROM ACCOUNT  
WHERE BALANCE > 100

vybere 2  
řádky

# Non-repeatable Read

ACCOUNT

ID	BALANCE
1	101
2	50

T1 BEGIN

SELECT \* FROM ACCOUNT  
WHERE BALANCE>100

uvolníme  
zámky

T2 BEGIN

UPDATE ACCOUNT SET  
BALANCE=200 WHERE ID=1

T2 COMMIT

SELECT \* FROM ACCOUNT  
WHERE BALANCE>100

vrátí jiný  
výsledek

ACCOUNT

ID	BALANCE
1	200
2	50

# Izolační úrovně

úroveň	long duration	short duration	
SERIALIZABLE	X, S		zamyká predikáty
REPEATABLE_READ	X, S		
READ_COMMITTED	X	S	
READ_UNCOMMITTED	X		nepoužívá zámky S

nižší izolace → vyšší souběžnost (concurrency)

anomálie: phantoms  
non-repeatable read  
dirty read

# Izolační úrovně v Javě

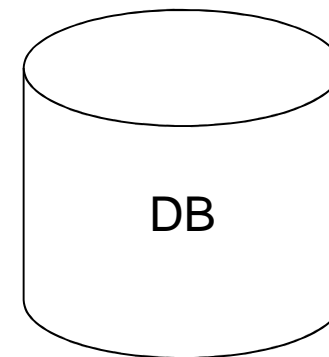
Connection:

```
setTransactionIsolation( int level )
```

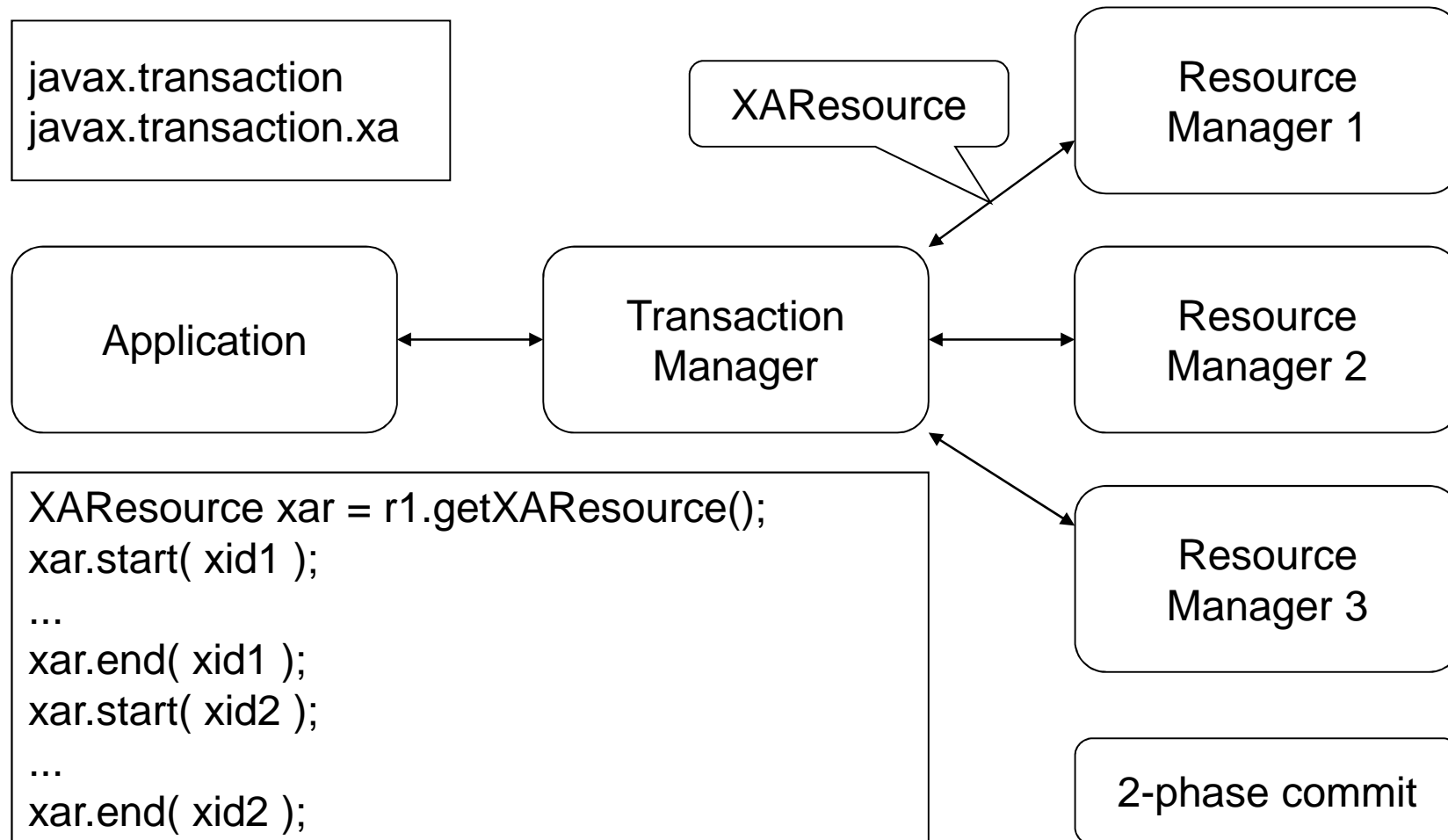
```
TRANSACTION_SERIALIZABLE  
TRANSACTION_REPEATABLE_READ  
TRANSACTION_READ_COMMITTED  
TRANSACTION_READ_UNCOMMITTED
```

izolační úroveň určuje,  
jak vidíme DB

SERIALIZABLE →  
READ\_COMMITTED →



# Distribuvované transakce



# Security

## Základní pojmy

- autentizace = ověření totožnosti
- autorizace = přidělení práv
- confidentiality
- integrity

```
@Stateless
@DeclareRoles( "manager" )
public class BankManager {
    ...
    @RolesAllowed( "manager" )
    public void addCustomer( ... ) {
        ...
    }
}
```

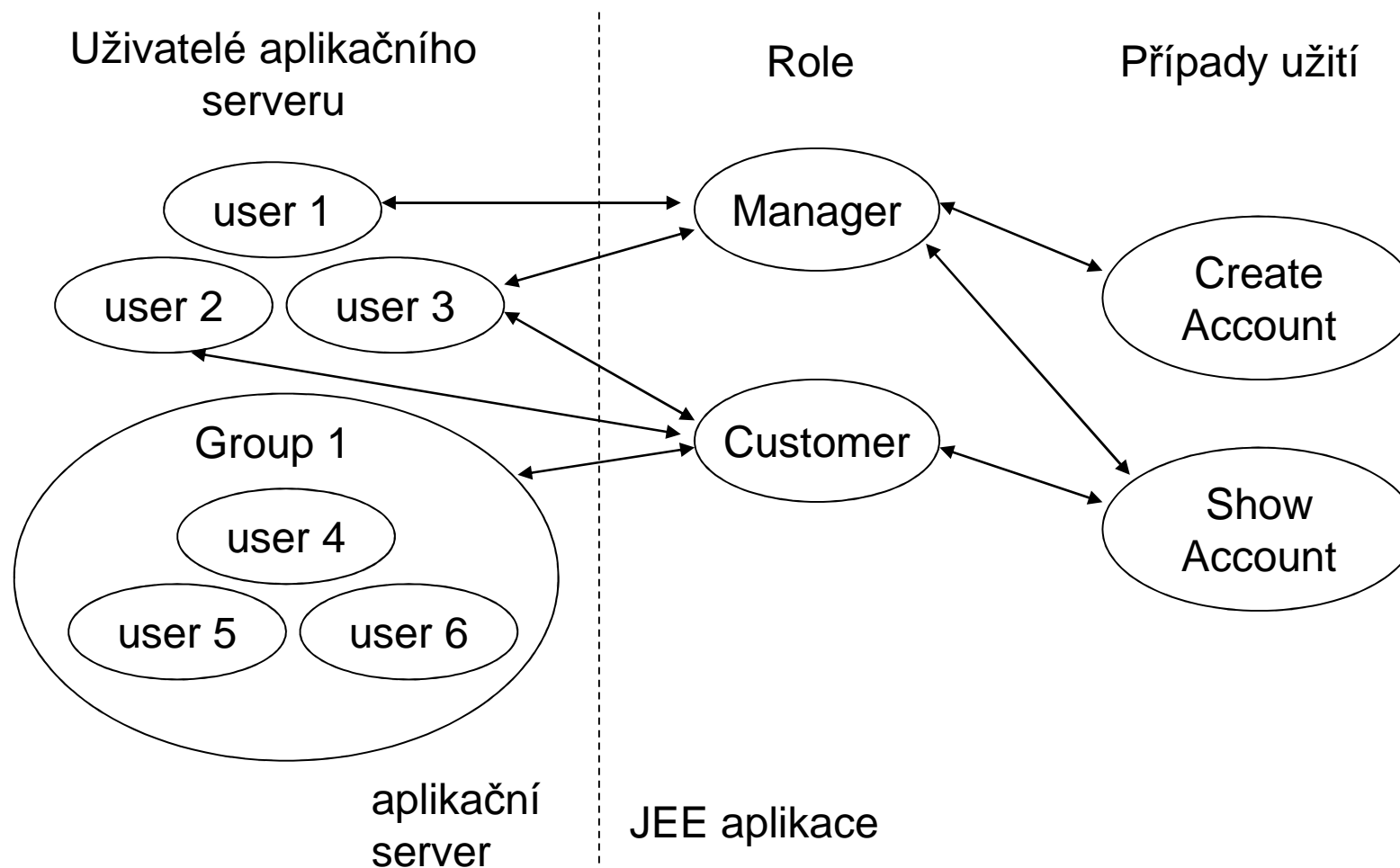
## Vlastnosti

- container-managed
- role-based
- end-to-end

- @DeclareRoles
- @RolesAllowed
- @PermitAll
- @DenyAll



# Uživatelské role



# Autentizace v HTTP

- HTTP BASIC – nešifrované
- HTTP DIGEST - pouze kontrolní součet
- Form Based
- HTTPS Client - používá certifikáty

Formulářová autentizace:

```
<form method="POST" action="j_security_check">  
  <input type="text" name="j_username">  
  <input type="password" name="j_password">  
  <input type="submit" value="login">  
</form>
```

# Certifikáty

## Kryptografie

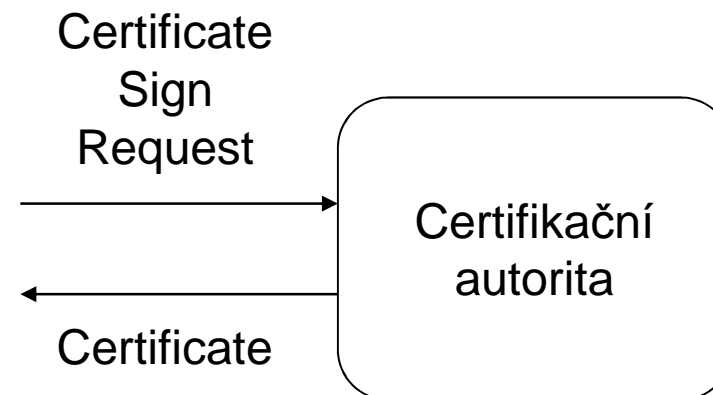
- symetrická (tajný klíč)
- asymetrická (soukromý a veřejný klíč)

## Algoritmy

- DES
- RSA
- MD5, SHA
- ...

## Certifikát

- veřejný klíč
- jméno vlastníka
- platnost do
- název vydavatele (CA)
- sériové číslo
- digitální podpis



# Otázky & odpovědi

tronicek@fit.cvut.cz