Data concurrency and locking

IBM Information Management Cloud Computing Center of Competence
IBM Canada Labs

© 2011 IBM Corporation

Agenda

- Transactions
- Concurrency & Locking
- Lock Wait
- Deadlocks

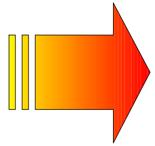
© 2011 IBM Corporation

Supporting reading material & videos

- **Reading materials**
  - Getting started with DB2 Express-C eBook
    - Chapter 13: Concurrency and locking
- **Videos**
  - db2university.com course AA001EN
    - Lesson 8: Data concurrency and locking

© 2011 IBM Corporation

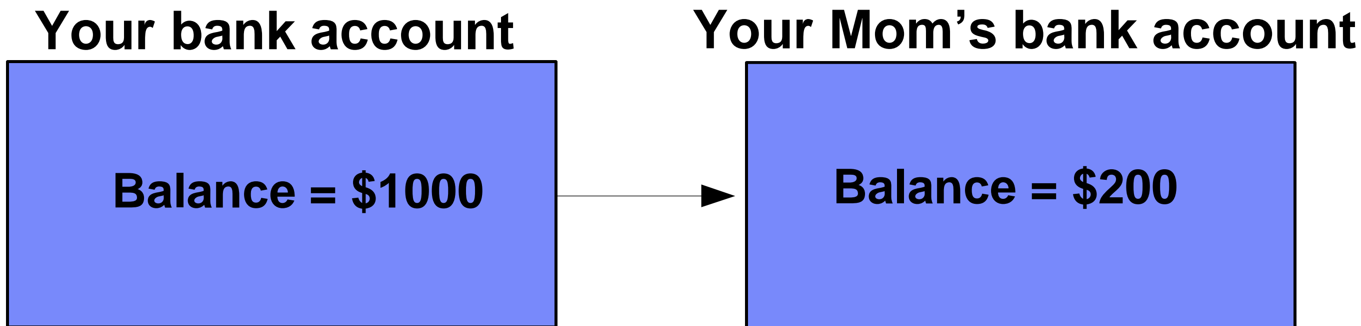
Agenda



- Transactions
- Concurrency & Locking
- Lock Wait
- Deadlocks

© 2011 IBM Corporation

What is a transaction?



Transfer $100 from your account to your Mom's account:

- Debit $100 from your bank account (Subtract $100)
- Credit $100 to your mom's bank account (Add $100)

© 2011 IBM Corporation

What is a transaction? (cont'd)

- One or more SQL statements altogether treated as one single unit
- Also known as a Unit of Work (UOW)
- A transaction starts with any SQL statement and ends with a COMMIT or ROLLBACK
- COMMIT statement makes changes permanent to the database
- ROLLBACK statement reverses changes
- COMMIT and ROLLBACK statements release all locks

© 2011 IBM Corporation

Example of transactions

First SQL statement starts transaction

```
INSERT INTO employee VALUES (100, 'JOHN')
INSERT INTO employee VALUES (200, 'MANDY')
COMMIT
```

empID	name
100	JOHN
200	MANDY

No changes applied due to ROLLBACK

```
DELETE FROM employee WHERE name='MANDY'
UPDATE employee SET empID=101 where name='JOHN'
ROLLBACK
```

empID	name
100	JOHN
200	MANDY

```
UPDATE employee SET name='JACK' where empID=100
COMMIT

ROLLBACK
```

There is nothing to rollback

empID	name
100	JACK
200	MANDY

© 2011 IBM Corporation

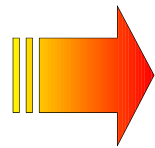
Transactions – ACID rules

- **A**tomicity
  - All statements in the transaction are treated as a unit.
  - If the transaction completes successfully, everything is committed
  - If the transaction fails, everything done up to the point of failure is rolled back.
- **C**onsistency
  - Any transaction will take the data from one consistent state to another, so only valid consistent data is stored in the database
- **I**solation
  - Concurrent transactions cannot interfere with each other
- **D**urability
  - Committed transactions have their changes persisted in the database

© 2011 IBM Corporation

Agenda

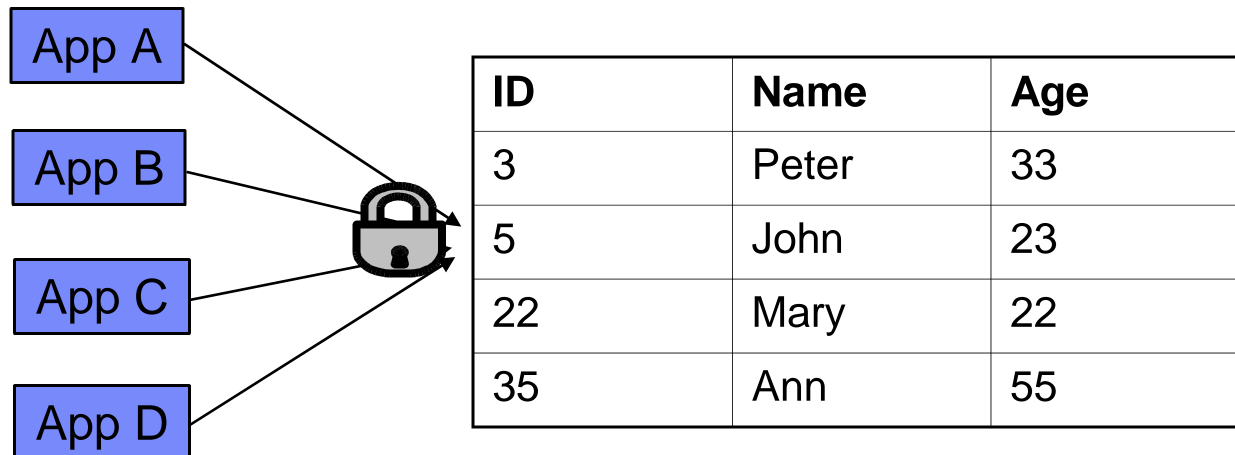
- Transactions



- Concurrency & Locking
- Lock Wait
- Deadlocks

© 2011 IBM Corporation

Concurrency and Locking



ID	Name	Age
3	Peter	33
5	John	23
22	Mary	22
35	Ann	55

- Concurrency:
  - Multiple users accessing the same resources at the same time
- Locking:
  - Mechanism to ensure data integrity and consistency

© 2011 IBM Corporation

Locking

- Locks are acquired automatically as needed to support a transaction based on "isolation levels"

- COMMIT and ROLLBACK statements release all locks

- Two basic types of locks:
  - Share locks (S locks) – acquired when an application wants to read and prevent others from updating the same row
  - Exclusive locks (X locks) – acquired when an application updates, inserts, or deletes a row

© 2011 IBM Corporation

Problems if there is no concurrency control

- Lost update
- Uncommitted read
- Non-repeatable read
- Phantom read

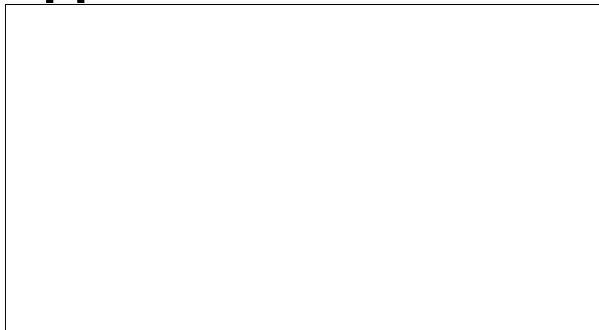
© 2011 IBM Corporation

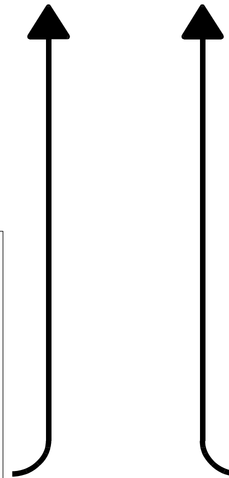
Lost update

reservations

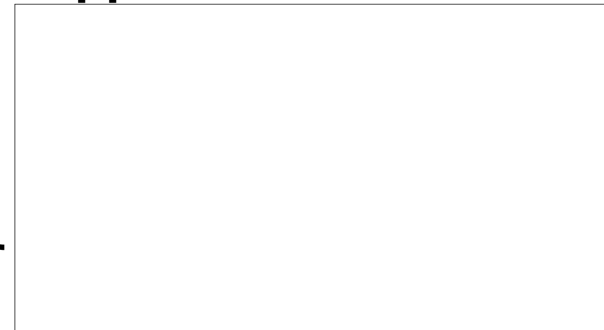
seat	name	...
7C	______	
7B	______	
...		

App A





App B



© 2011 IBM Corporation

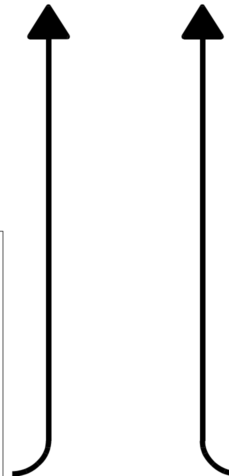
Lost update

reservations

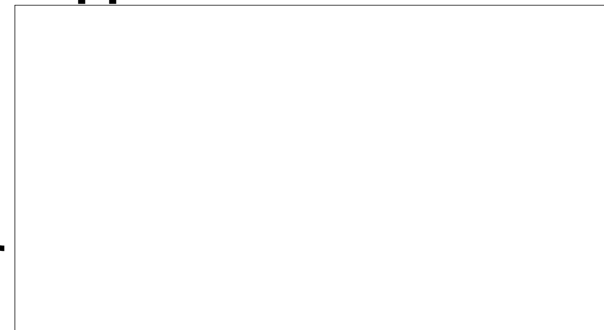
seat	name	...
7C	______	
7B	______	
...		

App A

```
update reservations
   set name = 'John'
   where seat = '7C'
```



App B



© 2011 IBM Corporation

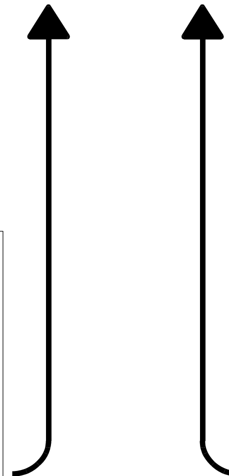
Lost update

reservations

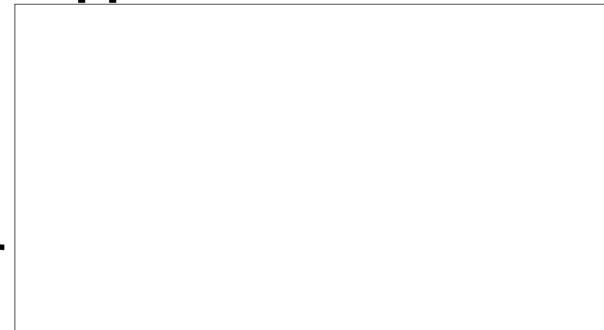
seat	name	...
7C	John	
7B	______	
...		

App A

```
update reservations
   set name = 'John'
   where seat = '7C'
```



App B



© 2011 IBM Corporation

Lost update

reservations

seat	name	...
7C	John	
7B	______	
...		

App A

```
update reservations
    set name = 'John'
    where seat = '7C'
```

App B

```
update reservations
    set name = 'Mary'
    where  seat = '7C'
```


© 2011 IBM Corporation

Lost update

reservations

seat	name	...
7C	Mary	
7B	______	
...		

App A

```
update reservations
    set name = 'John'
    where seat = '7C'
```

App B

```
update reservations
    set name = 'Mary'
    where  seat = '7C'
```


© 2011 IBM Corporation

Lost update

reservations

seat	name	...
7C	Mary ?	
7B	______	
...		

App A

```
update reservations
    set name = 'John'
    where seat = '7C'
```

App B

```
update reservations
    set name = 'Mary'
    where  seat = '7C'
```

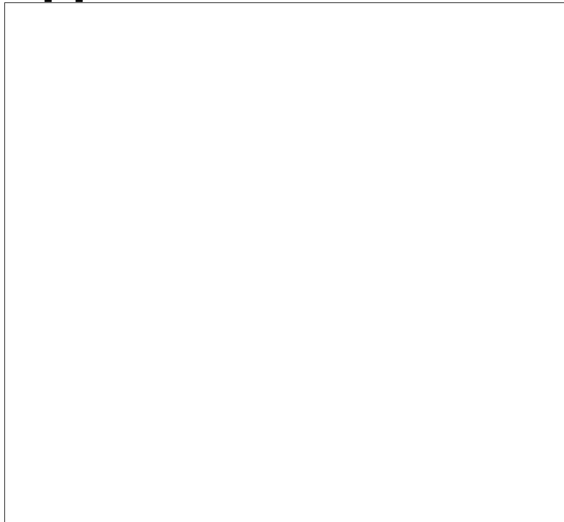

© 2011 IBM Corporation

Uncommitted read (also known as “dirty read”)

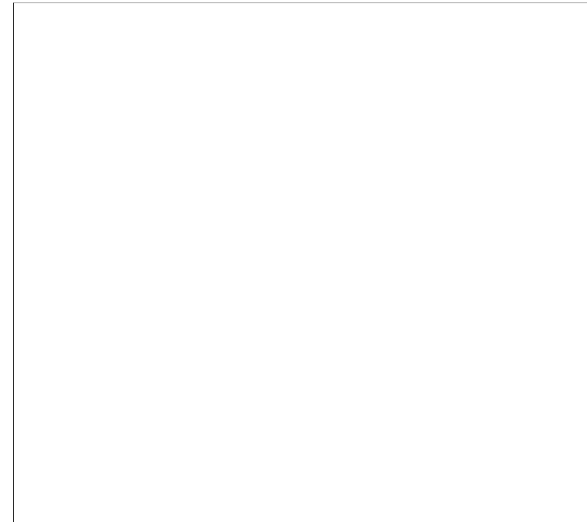
reservations

seat	name	...
7C	______	
7B	______	
...		

App A



App B



© 2011 IBM Corporation

Uncommitted read (also known as "dirty read")

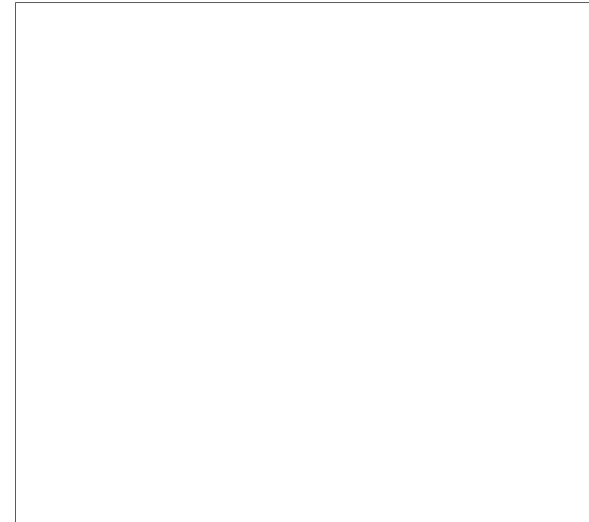
reservations

seat	name	...
7C	_____	
7B	_____	
...		

App A

```
update reservations
   set name = 'John'
   where seat = '7C'
```

App B



© 2011 IBM Corporation

Uncommitted read (also known as "dirty read")

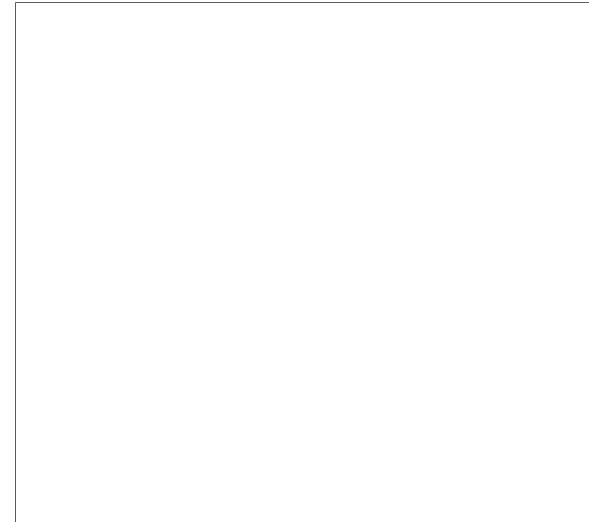
reservations

seat	name	...
7C	John	
7B	_____	
...		

App A

```
update reservations
   set name = 'John'
   where seat = '7C'
```

App B



© 2011 IBM Corporation

Uncommitted read (also known as "dirty read")

reservations

seat	name	...
7C	John	
7B	_____	
...		

App A

```
update reservations
    set name = 'John'
    where seat = '7C'
```

App B

```
Select name
    from reservations
    where seat is '7C'
```


© 2011 IBM Corporation

Uncommitted read (also known as "dirty read")

reservations

seat	name	...
7C	John	
7B	______	
...		

App A

```
update reservations
    set name = 'John'
    where seat = '7C'
```

App B

```
Select name
    from reservations
    where seat is '7C'
```

John

© 2011 IBM Corporation

Uncommitted read (also known as "dirty read")

reservations

seat	name	...
7C	John	
7B	______	
...		

App A

```
update reservations
    set name = 'John'
    where seat = '7C'

Roll back
```

App B

```
Select name
    from reservations
    where seat is '7C'
```

John

© 2011 IBM Corporation

Uncommitted read (also known as "dirty read")

reservations

seat	name	...
7C	______	
7B	______	
...		

App A

```
update reservations
    set name = 'John'
    where seat = '7C'

Roll back
```

App B

```
Select name
    from reservations
    where seat is '7C'
```

John

© 2011 IBM Corporation

Uncommitted read (also known as "dirty read")

reservations

seat	name	...
7C	______	
7B	______	
...		

App A

```
update reservations
    set name = 'John'
    where seat = '7C'

Roll back
```

App B

```
Select name
    from reservations
    where seat is '7C'
```

John

Further processing in App B uses incorrect / uncommitted value of "John"

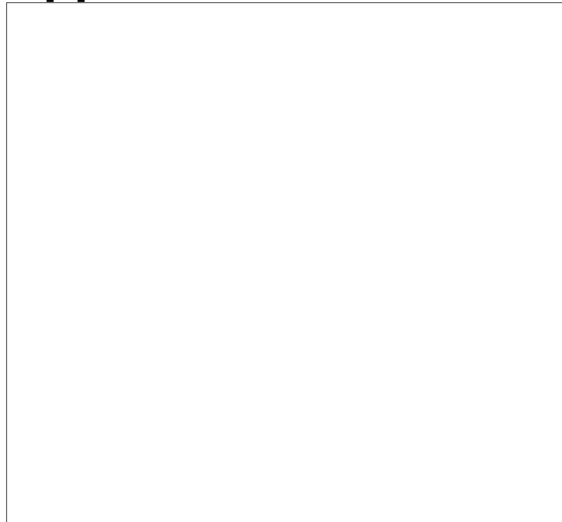
© 2011 IBM Corporation

Non-repeatable read

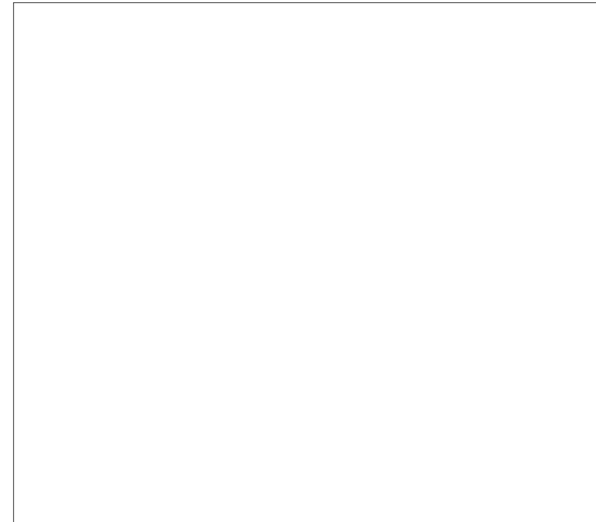
reservations

seat	name	...
7C	______	
7B	______	
...		

App A



App B



© 2011 IBM Corporation

Non-repeatable read

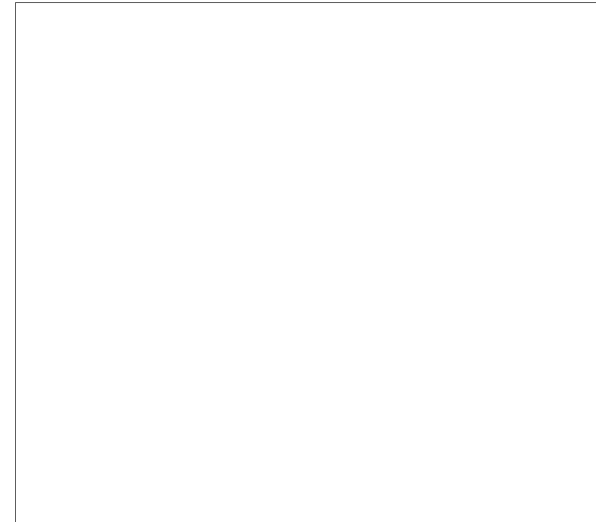
reservations

seat	name	...
7C	______	
7B	______	
...		

App A

```
select seat
  from reservations
  where name is NULL
```

App B



© 2011 IBM Corporation

Non-repeatable read

reservations

seat	name	...
7C	_____	
7B	_____	
...		

7C
7B

App A

```
select seat
  from  reservations
  where name is NULL
```

App B

© 2011 IBM Corporation

Non-repeatable read

reservations

seat	name	...
7C	_____	
7B	_____	
...		

7C
7B

App A

```
select seat
  from  reservations
  where name is NULL
```

App B

```
update reservations
    set name = 'John'
    where seat = '7C'
```


© 2011 IBM Corporation

Non-repeatable read

reservations

seat	name	...
7C	John	
7B	______	
...		

7C
7B

App A

```
select seat
  from  reservations
  where name is NULL
```

App B

```
update reservations
    set name = 'John'
    where seat = '7C'
```


© 2011 IBM Corporation

Non-repeatable read

reservations

seat	name	...
7C	John	
7B	_____	
...		

7C
7B

App A

```
select seat
  from  reservations
  where name is NULL

 ...
select seat
  from  reservations
  where name is NULL
```

App B

```
update reservations
    set name = 'John'
    where seat = '7C'
```


© 2011 IBM Corporation

Non-repeatable read

reservations

seat	name	...
7C	John	
7B	_____	
...		

App A

7C
7B

```
select seat
  from  reservations
  where name is NULL

  ...
select seat
  from  reservations
  where name is NULL
```

7B

App B

```
update reservations
    set name = 'John'
    where seat = '7C'
```


© 2011 IBM Corporation

Non-repeatable read

reservations

seat	name	...
7C	John	
7B	_____	
...		

App A

7C
7B

```
select seat
  from  reservations
  where name is NULL

 ...
select seat
  from  reservations
  where name is NULL
```

7B

App B

```
update reservations
    set name = 'John'
    where seat = '7C'
```

The same SELECT (read) returns a different result: Less rows (in this case '7C' doesn't show anymore). This is a non-repeatable read

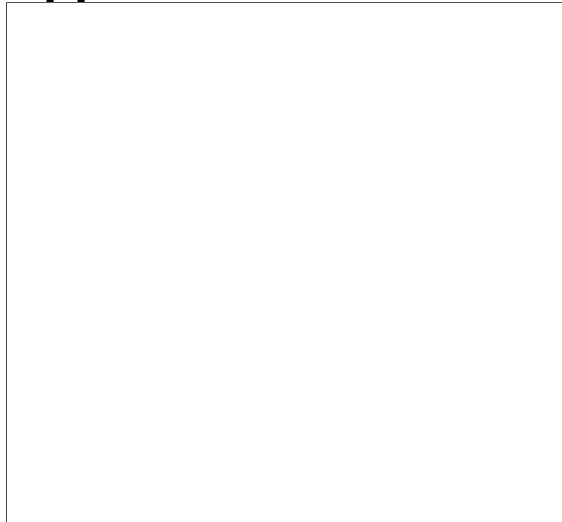
© 2011 IBM Corporation

Phantom read

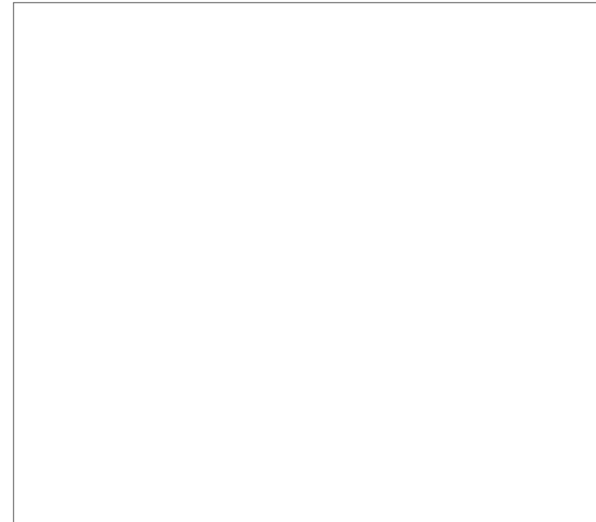
reservations

seat	name	...
7C	Susan	
7B	______	
...		

App A



App B



© 2011 IBM Corporation

IBM

Phantom read

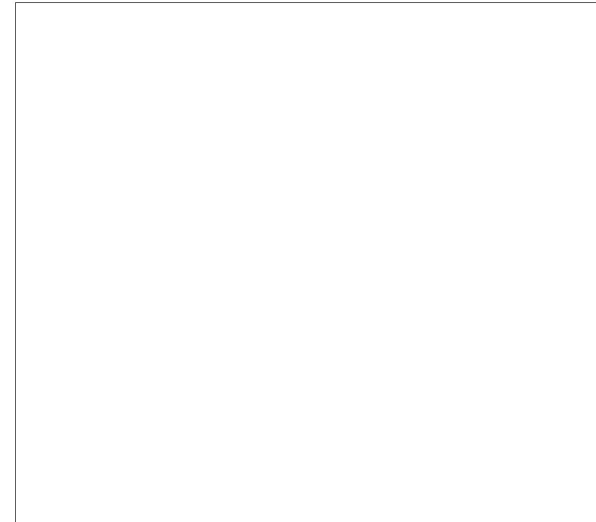
reservations

seat	name	...
7C	Susan	
7B	______	
...		

App A

```
select seat
  from  reservations
  where name is NULL
```

App B



© 2011 IBM Corporation

Phantom read

reservations

seat	name	...
7C	Susan	
7B	____	
...		

7B

App A

```
select seat
  from  reservations
  where name is NULL
```

App B

© 2011 IBM Corporation

Phantom read

reservations

seat	name	...
7C	Susan	
7B	_____	
...		

7B

App A

```
select seat
  from  reservations
  where name is NULL
```

App B

```
update reservations
    set name = NULL
    where seat = '7C'
```


© 2011 IBM Corporation

Phantom read

reservations

seat	name	...
7C	______	
7B	______	
...		

7B

App A

```
select seat
  from  reservations
  where name is NULL
```

App B

```
update reservations
    set name = NULL
    where seat = '7C'
```


© 2011 IBM Corporation

Phantom read

reservations

seat	name	...
7C	______	
7B	______	
...		

7B

App A

```
select seat
  from  reservations
  where name is NULL

  ...
select seat
  from  reservations
  where name is NULL
```

App B

```
update reservations
    set name = NULL
    where seat = '7C'
```


© 2011 IBM Corporation

Phantom read

reservations

seat	name	...
7C	______	
7B	______	
...		

7B

7C

7B

App A

```
select seat
  from  reservations
  where name is NULL

  ...
select seat
  from  reservations
  where name is NULL
```

App B

```
update reservations
    set name = NULL
    where seat = '7C'
```


© 2011 IBM Corporation

Phantom read

reservations

seat	name	...
7C	______	
7B	______	
...		

App A

```
select seat
  from  reservations
  where name is NULL

  ...
select seat
  from  reservations
  where name is NULL
```

7B

7C
7B

App B

```
update reservations
    set name = NULL
    where seat = '7C'
```

The same SELECT (read) returns a different result: More rows (phantom rows, in this case '7C', is shown) This is a phantom read

© 2011 IBM Corporation

Isolation levels

- **“Policies” to control when locks are taken**
- **DB2 provides different levels of protection to isolate data**
  - Uncommitted Read (UR)
  - Cursor Stability (CS)
    - Currently committed (CC)
  - Read Stability (RS)
  - Repeatable Read (RR)

© 2011 IBM Corporation

Setting the isolation levels

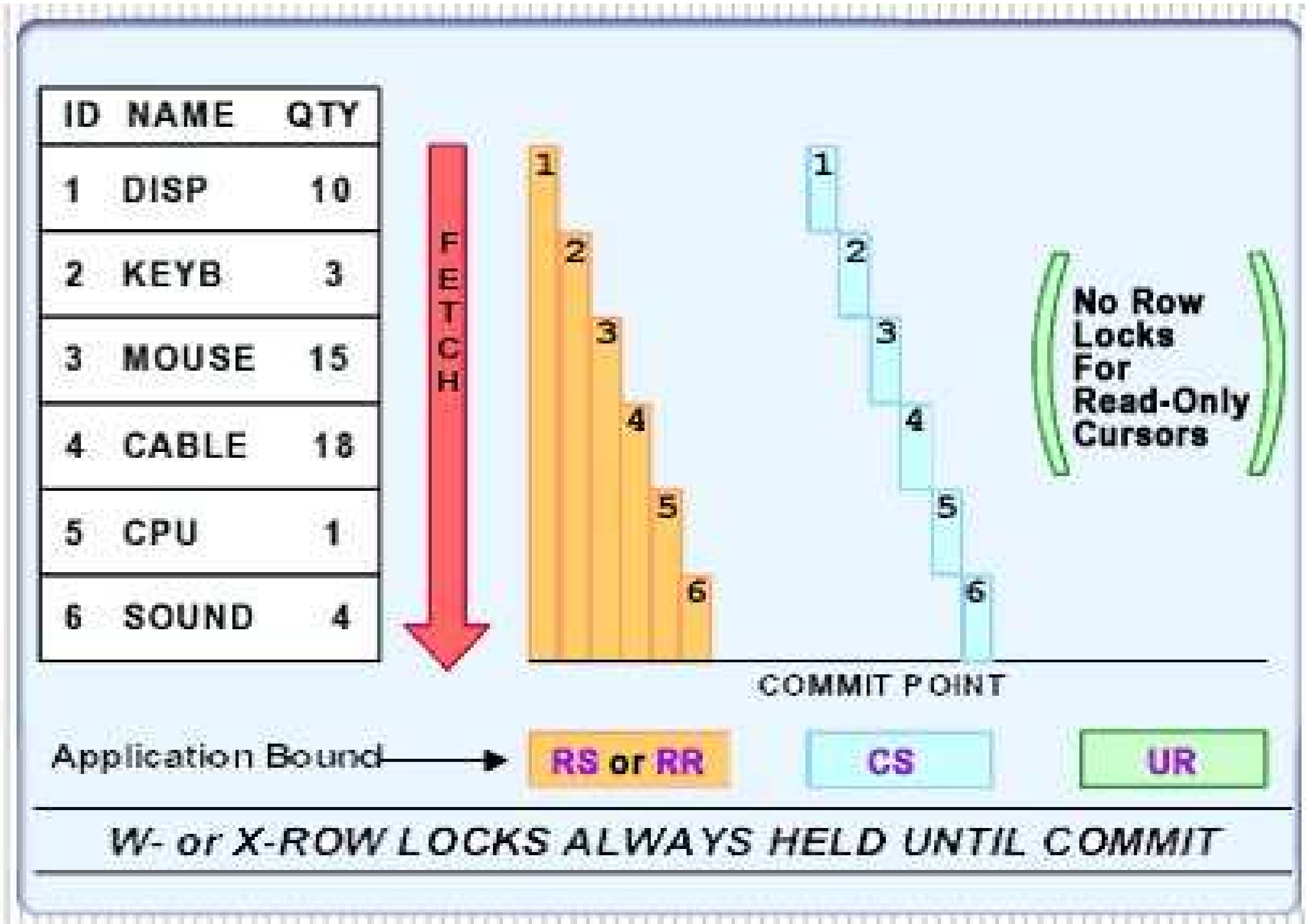
- **Isolation level can be specified at many levels**
  - Session (application),
  - Connection,
  - Statement
    - For statement level, use the WITH {RR, RS, CS, UR} clause:

```
SELECT COUNT(*) FROM tab1 WITH UR
```

- **For embedded SQL, the level is set at bind time**
- **For dynamic SQL, the level is set at run time**

© 2011 IBM Corporation

Comparing isolation levels



© 2011 IBM Corporation

Cursor stability with currently committed (CC) semantics

- Cursor stability with *currently committed* semantics is the default isolation level
- Use *cur_commit* db cfg parameter to enable/disable
- Avoids timeouts and deadlocks

Cursor stability

Situation	Result
Reader blocks Reader	No
Reader blocks Writer	Maybe
Writer blocks Reader	Yes
Writer blocks Writer	Yes

Cursor stability with currently committed

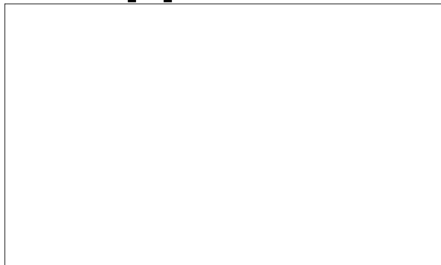
Situation	Result
Reader blocks Reader	No
Reader blocks Writer	No
Writer blocks Reader	No
Writer blocks Writer	Yes

© 2011 IBM Corporation

Cursor stability with currently committed (CC) semantics

Cursor stability *without* currently committed

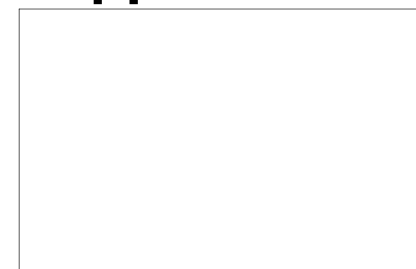
App A



reservations

seat	name	...
7C	Susan	
7B	_____	
...		

App B



Cursor stability *with* currently committed (Default behavior)

App A

reservations

seat	name	...
7C	Susan	
7B	_____	
...		

App B

© 2011 IBM Corporation

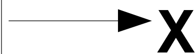
Cursor stability with currently committed (CC) semantics

Cursor stability *without* currently committed

reservations

App A

update
  reservations
  set name = 'John'
  where seat = '7C'



seat	name	...
7C	Susan	
7B	______	
...		

App B

Cursor stability *with* currently committed (Default behavior)

reservations

App A

seat	name	...
7C	Susan	
7B	______	
...		

App B

© 2011 IBM Corporation

Cursor stability with currently committed (CC) semantics

Cursor stability *without* currently committed

App A

```
update
  reservations
  set name = 'John'
  where seat = '7C'
```

→ X

reservations

seat	name	...
7C	John	
7B	_____	
...		

App B

Cursor stability *with* currently committed (Default behavior)

App A

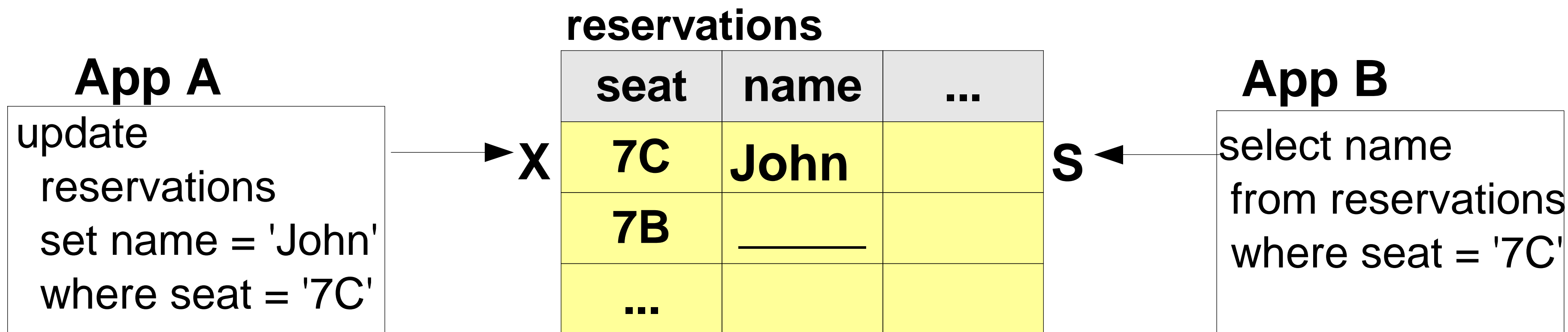
reservations

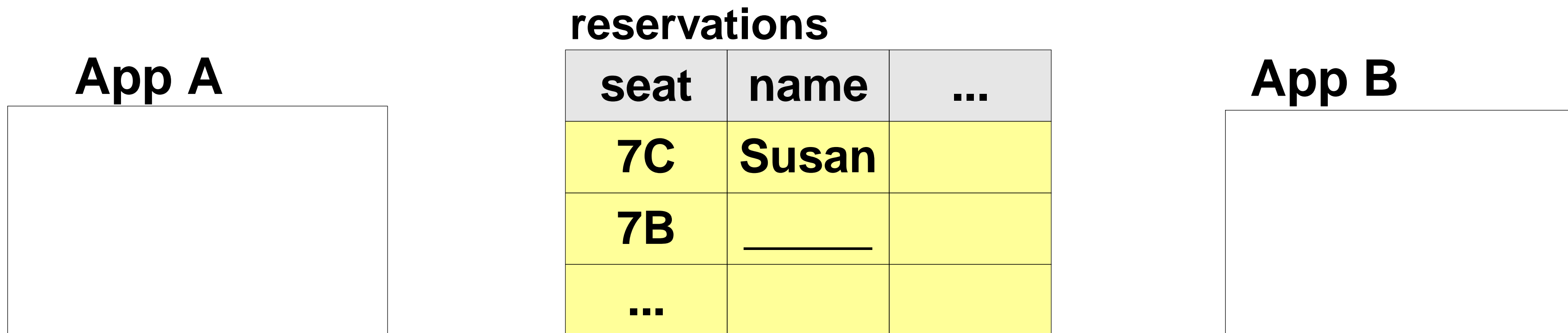
seat	name	...
7C	Susan	
7B	_____	
...		

App B

© 2011 IBM Corporation

Cursor stability with currently committed (CC) semantics



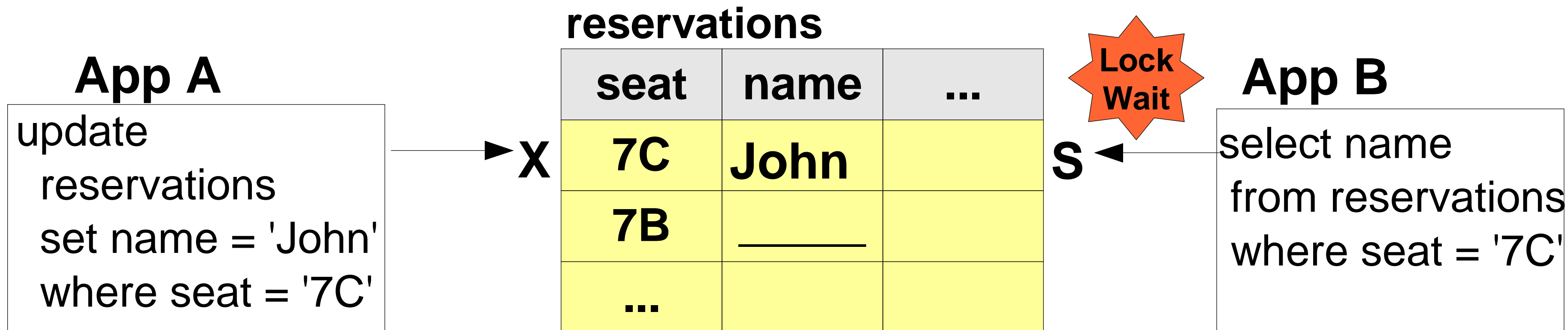


© 2011 IBM Corporation

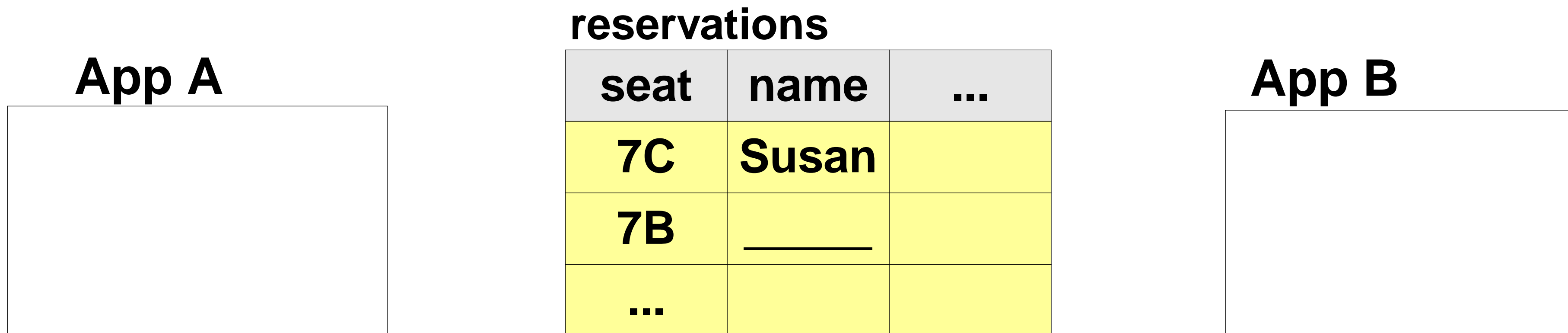
IBM

Cursor stability with currently committed (CC) semantics

Cursor stability ***without*** currently committed

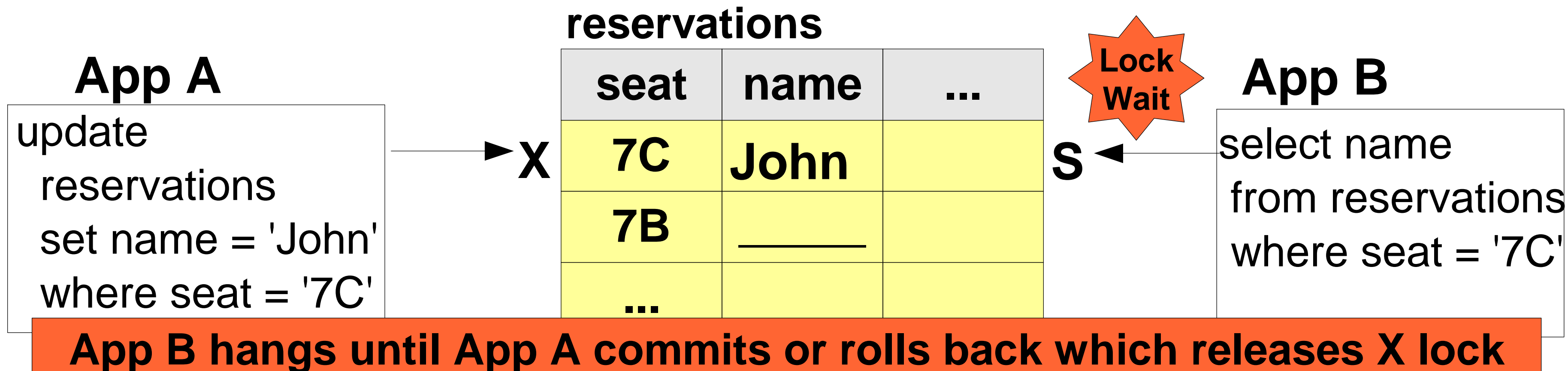


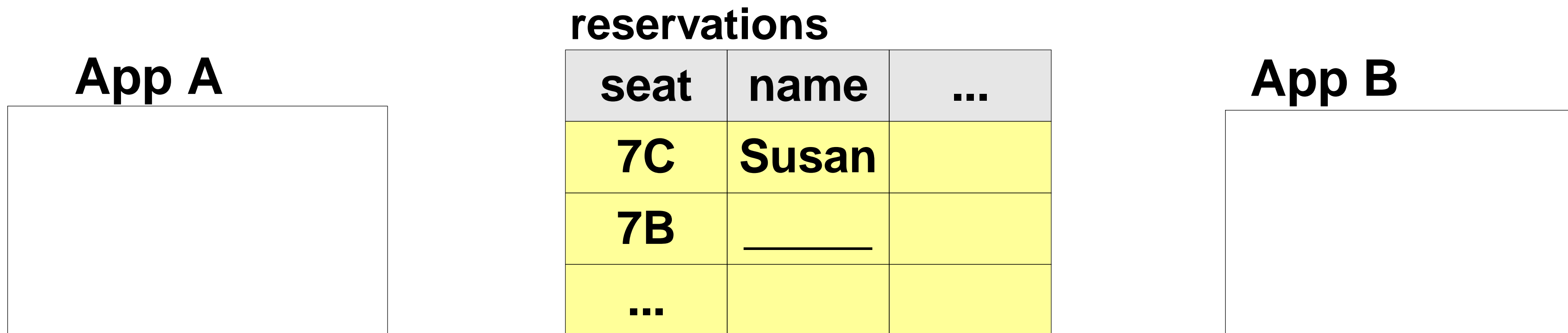
Cursor stability ***with*** currently committed (Default behavior)



© 2011 IBM Corporation

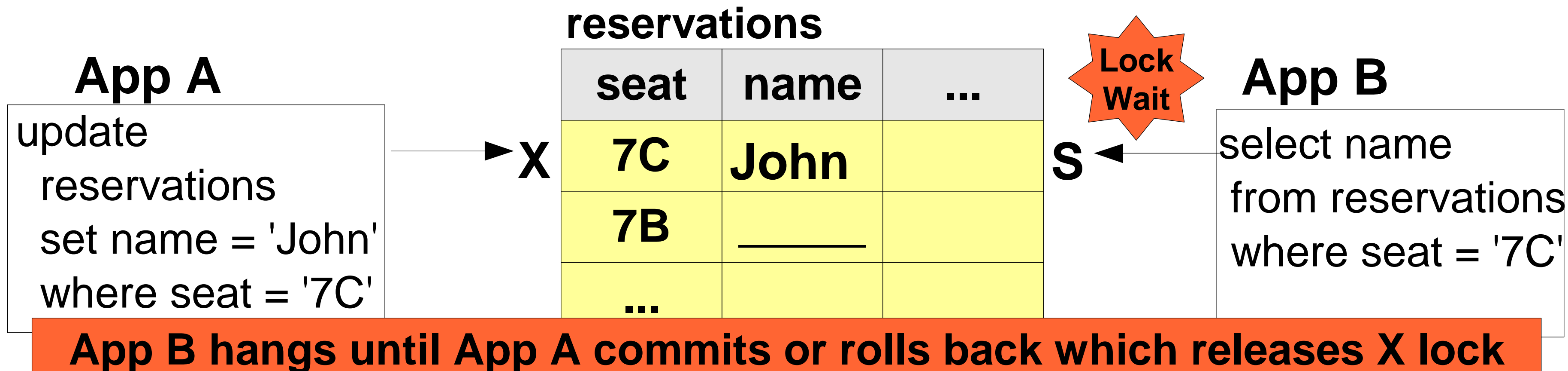
Cursor stability with currently committed (CC) semantics

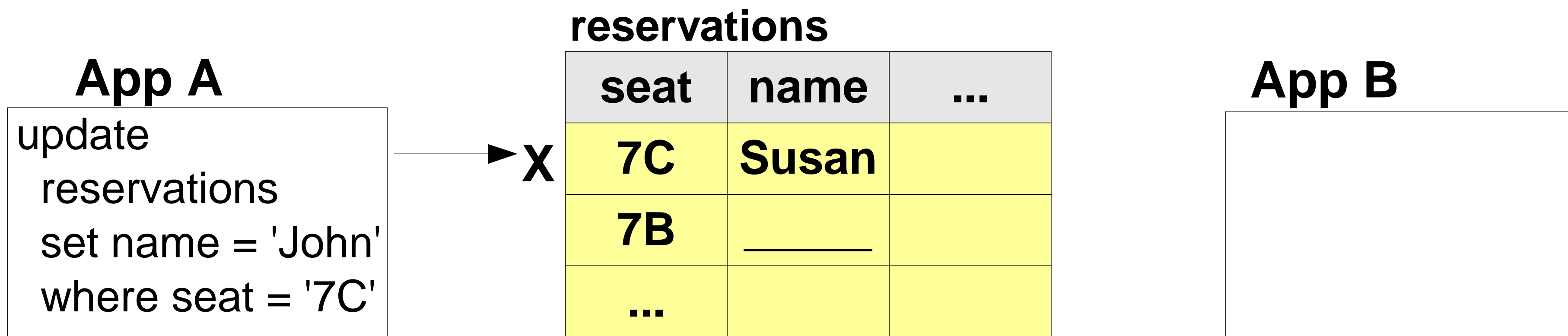




© 2011 IBM Corporation

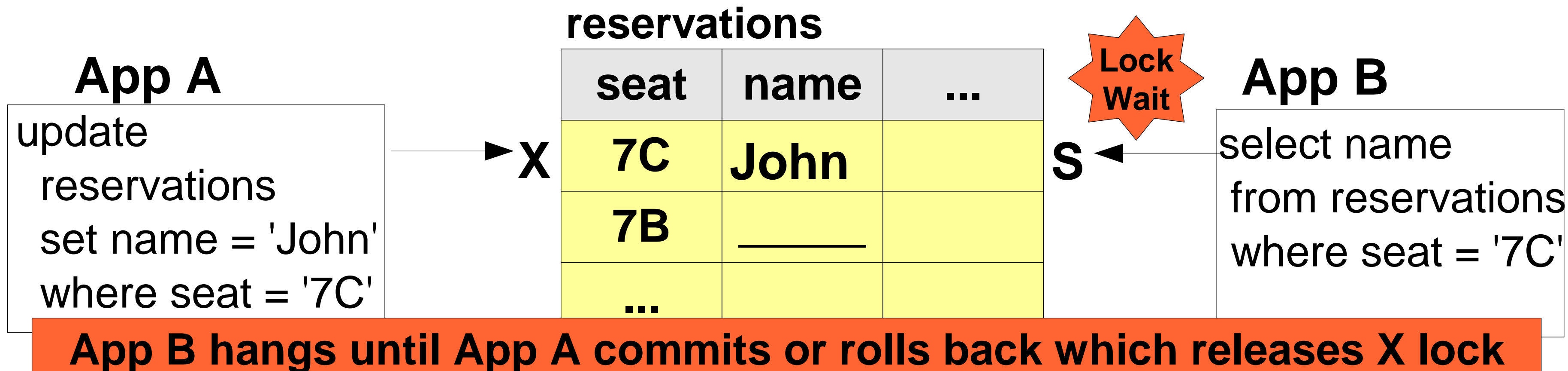
Cursor stability with currently committed (CC) semantics

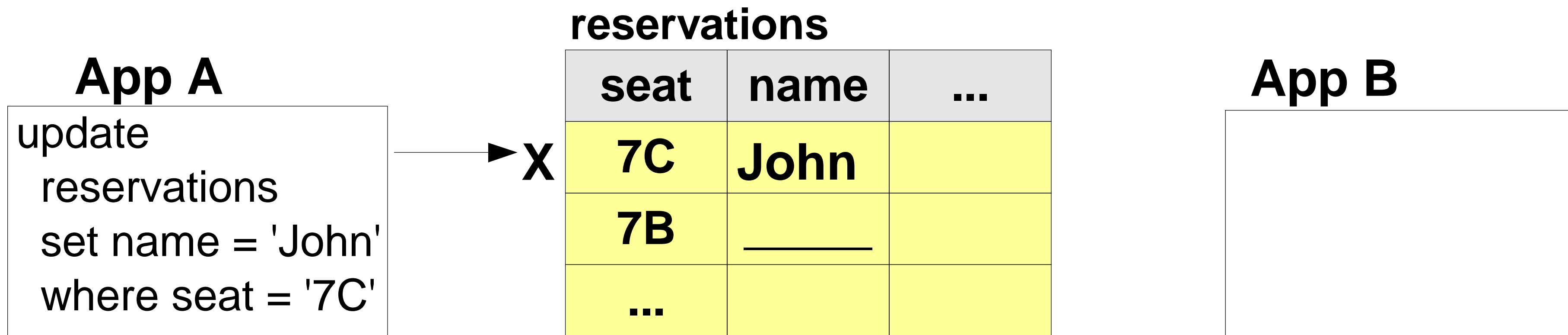




© 2011 IBM Corporation

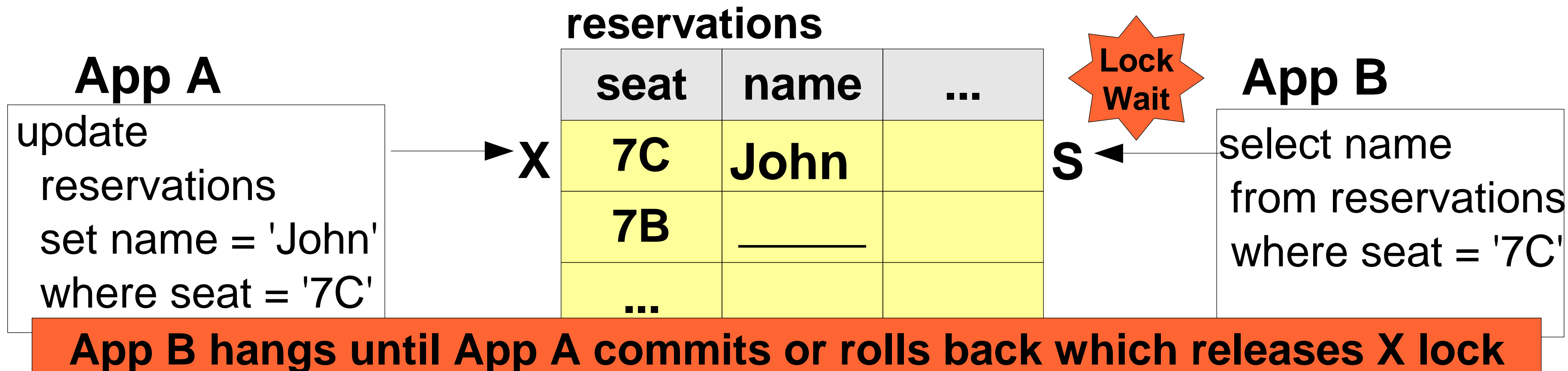
Cursor stability with currently committed (CC) semantics

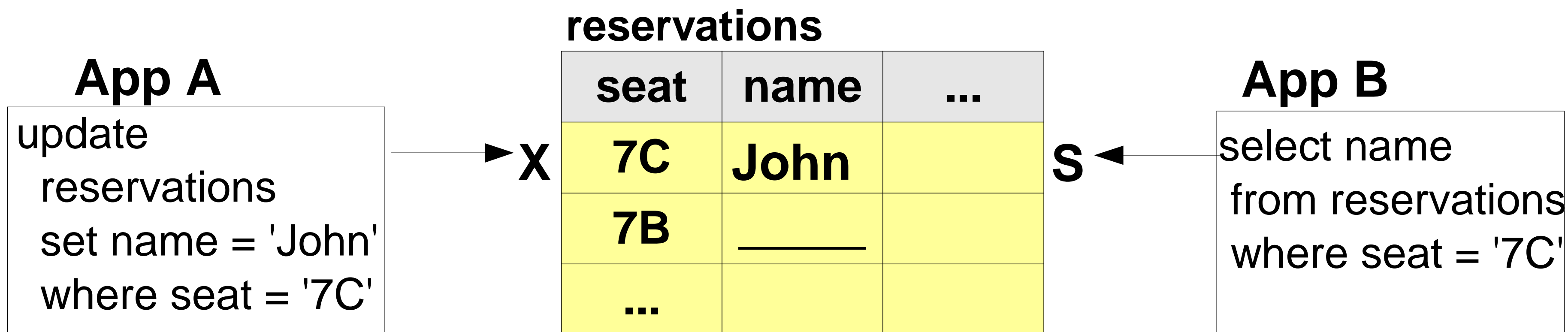




© 2011 IBM Corporation

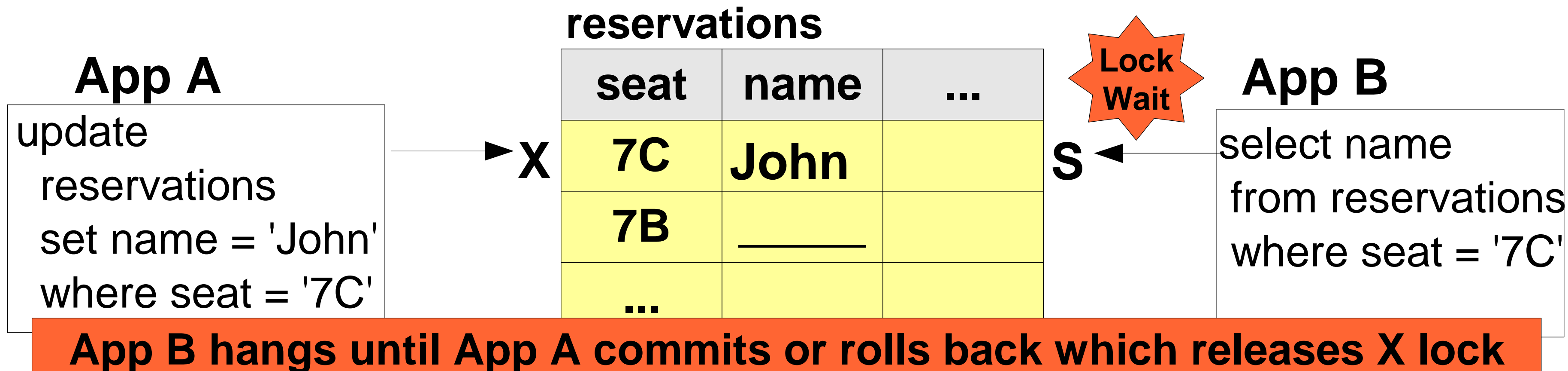
Cursor stability with currently committed (CC) semantics

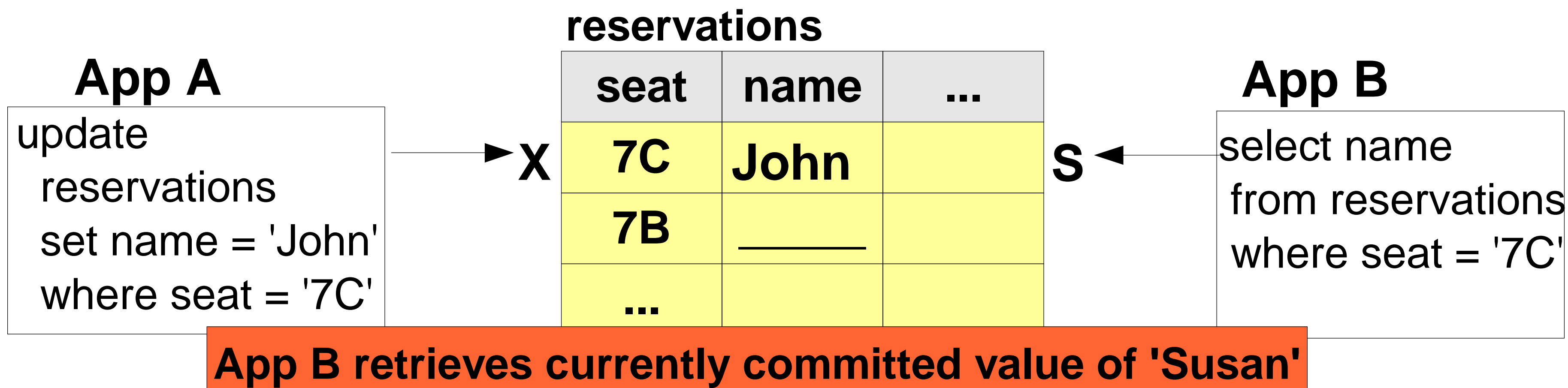




© 2011 IBM Corporation

Cursor stability with currently committed (CC) semantics





© 2011 IBM Corporation

Comparing and choosing an isolation level

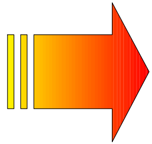
Isolation Level	Lost update	Dirty Read	Non-repeatable Read	Phantom Read
Repeatable Read (RR)	-	-	-	-
ReadStability (RS)	-	-	-	Possible
Cursor Stability (CS)	-	-	Possible	Possible
Uncommitted Read (UR)	-	Possible	Possible	Possible

Application Type	High data stability required	High data stability not required
Read-write transactions	RS	CS
Read-only transactions	RS or RR	UR

© 2011 IBM Corporation

Agenda

- Transactions
- Concurrency & Locking



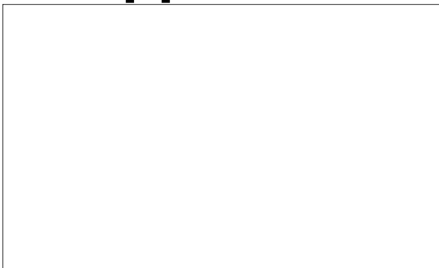
- Lock Wait
- Deadlocks

© 2011 IBM Corporation

Lock wait

- By default, an application waits indefinitely to obtain any needed locks
- **LOCKTIMEOUT** (db cfg):
  - Specifies the number of seconds to wait for a lock
  - Default value is -1 or *infinite* wait
- Example: (Same as when using isolation CS without CC):

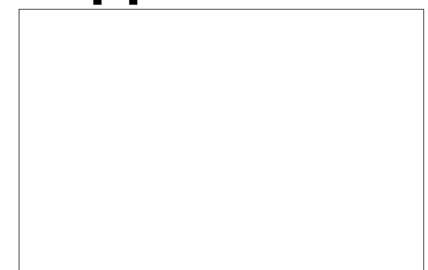
App A



reservations

seat	name	...
7C	Susan	
7B	______	
...		

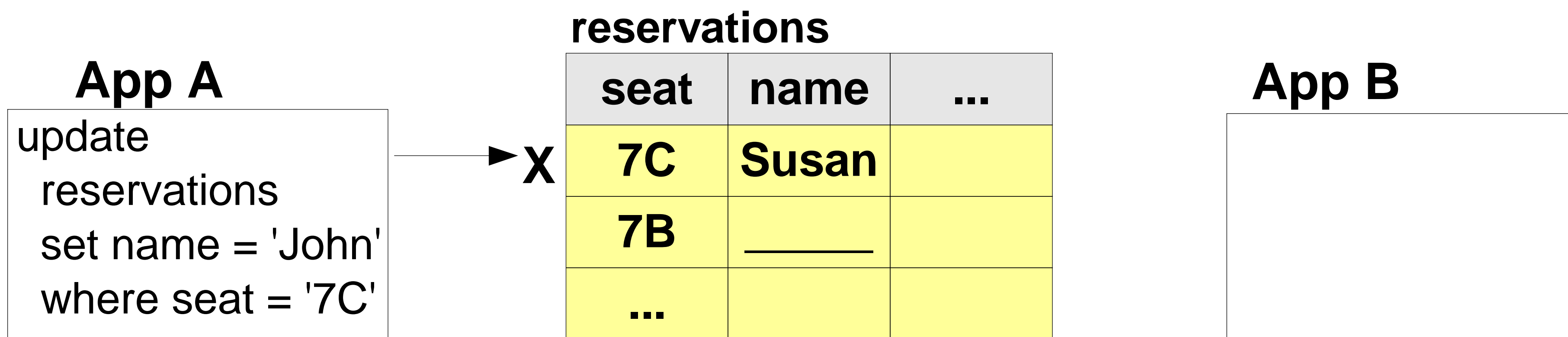
App B



© 2011 IBM Corporation

Lock wait

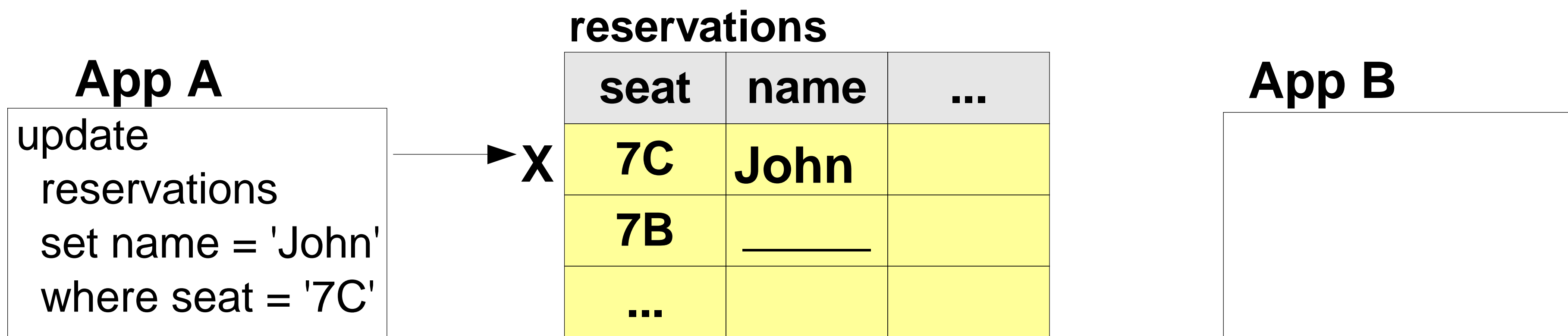
- By default, an application waits indefinitely to obtain any needed locks
- **LOCKTIMEOUT** (db cfg):
  - Specifies the number of seconds to wait for a lock
  - Default value is -1 or *infinite* wait
- Example: (Same as when using isolation CS without CC):



© 2011 IBM Corporation

Lock wait

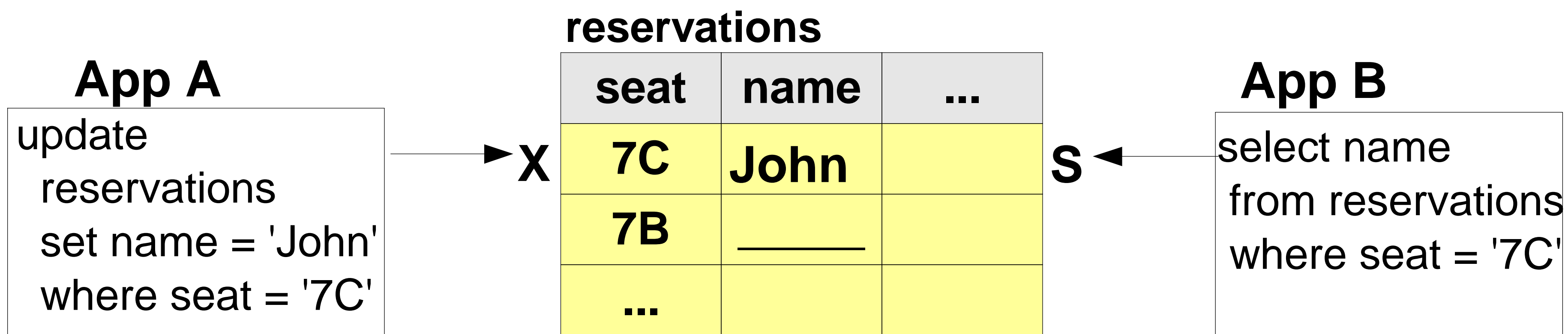
- By default, an application waits indefinitely to obtain any needed locks
- **LOCKTIMEOUT** (db cfg):
  - Specifies the number of seconds to wait for a lock
  - Default value is -1 or *infinite* wait
- Example: (Same as when using isolation CS without CC):



© 2011 IBM Corporation

Lock wait

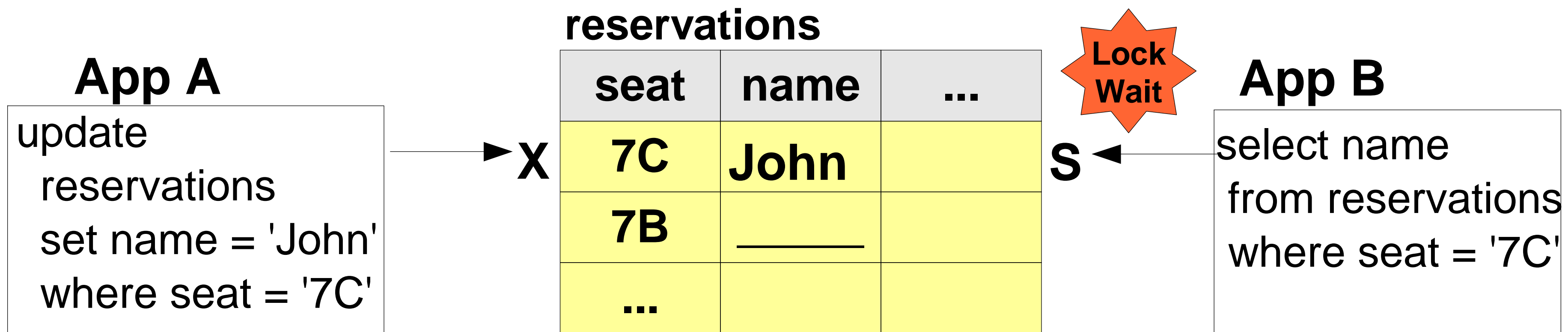
- By default, an application waits indefinitely to obtain any needed locks
- **LOCKTIMEOUT** (db cfg):
  - Specifies the number of seconds to wait for a lock
  - Default value is -1 or *infinite* wait
- Example: (Same as when using isolation CS without CC):



© 2011 IBM Corporation

Lock wait

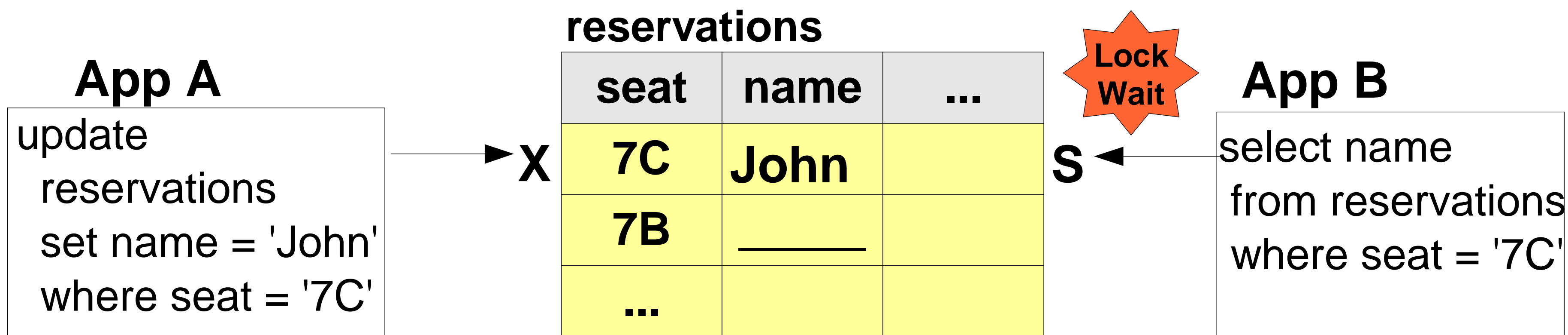
- By default, an application waits indefinitely to obtain any needed locks
- **LOCKTIMEOUT** (db cfg):
  – Specifies the number of seconds to wait for a lock
  – Default value is -1 or *infinite* wait
- Example: (Same as when using isolation CS without CC):



© 2011 IBM Corporation

Lock wait

- By default, an application waits indefinitely to obtain any needed locks
- **LOCKTIMEOUT** (db cfg):
  - Specifies the number of seconds to wait for a lock
  - Default value is -1 or *infinite* wait
- Example: (Same as when using isolation CS without CC):

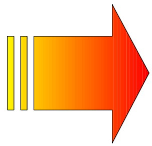


© 2011 IBM Corporation

IBM

Agenda

- Transactions
- Concurrency & Locking
- Lock Wait



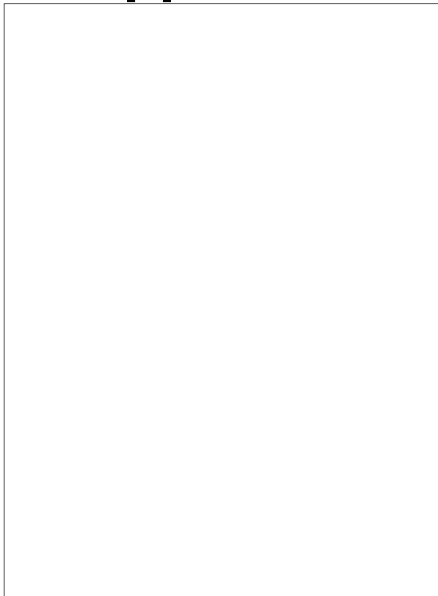
- Deadlocks

© 2011 IBM Corporation

Deadlocks

- Occurs when two or more applications wait indefinitely for a resource
- Each application is holding a resource that the other needs
- Waiting is never resolved
- In the example, assume we are using isolation CS without CC

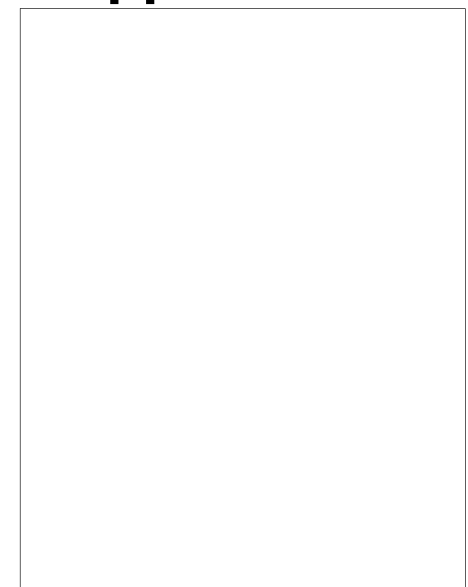
App A



reservations

seat	name	...
7C	Susan	
7B	_____	
...		
8E	Raul	
9F	Jin	

App B



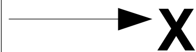
© 2011 IBM Corporation

Deadlocks

- Occurs when two or more applications wait indefinitely for a resource
- Each application is holding a resource that the other needs
- Waiting is never resolved
- In the example, assume we are using isolation CS without CC

App A

update
reservations
set name = 'John'
where seat = '7C'

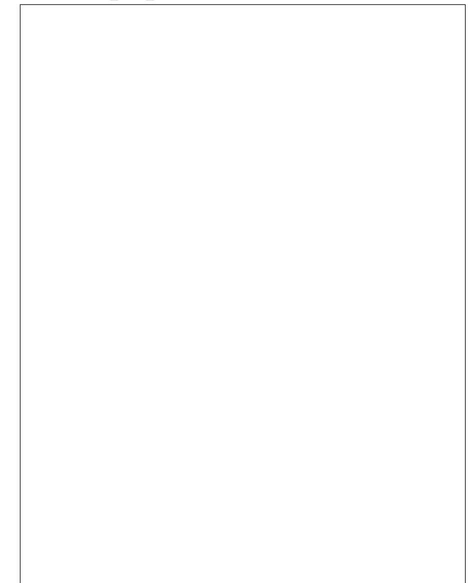


X

reservations

seat	name	...
7C	Susan	
7B	______	
...		
8E	Raul	
9F	Jin	

App B



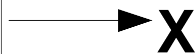
© 2011 IBM Corporation

Deadlocks

- Occurs when two or more applications wait indefinitely for a resource
- Each application is holding a resource that the other needs
- Waiting is never resolved
- In the example, assume we are using isolation CS without CC

App A

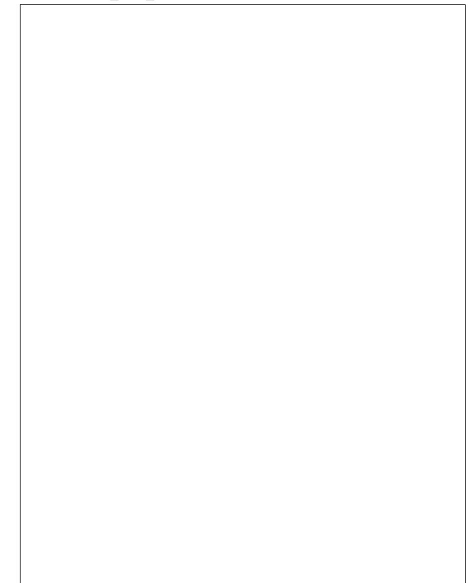
```
update
  reservations
  set name = 'John'
  where seat = '7C'
```



reservations

seat	name	...
7C	John	
7B	______	
...		
8E	Raul	
9F	Jin	

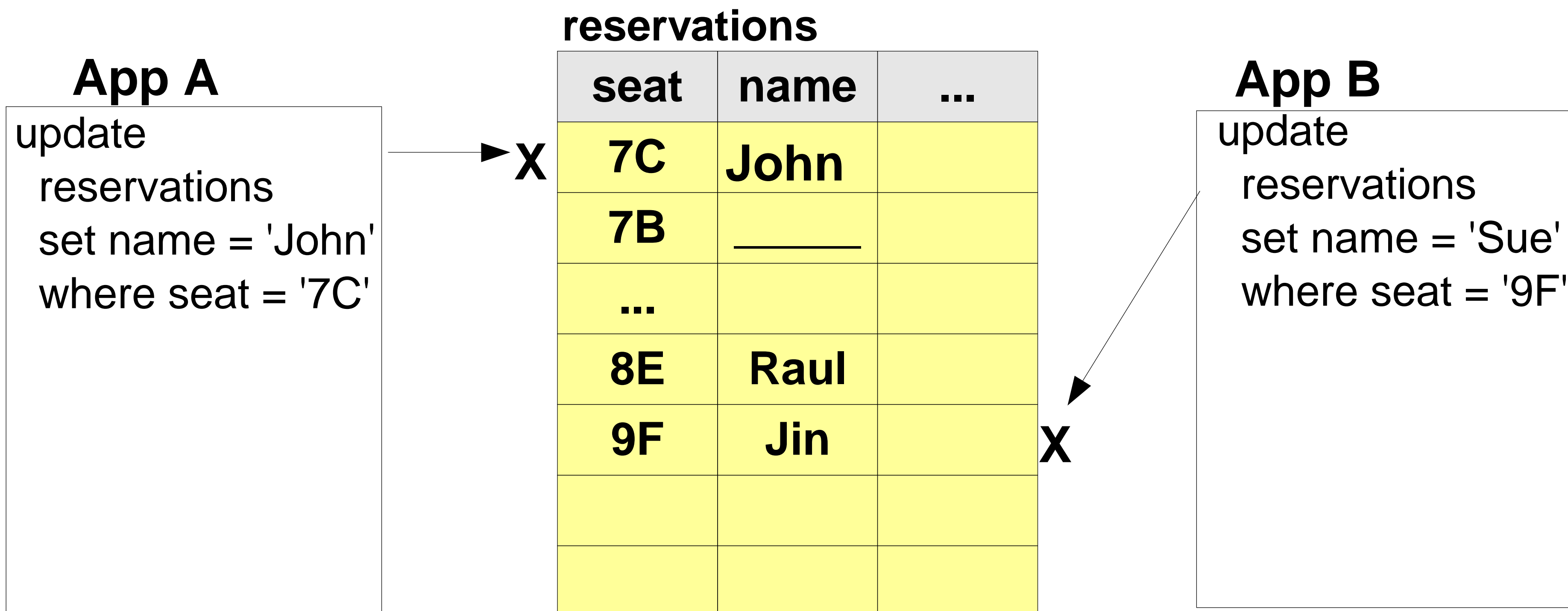
App B



© 2011 IBM Corporation

Deadlocks

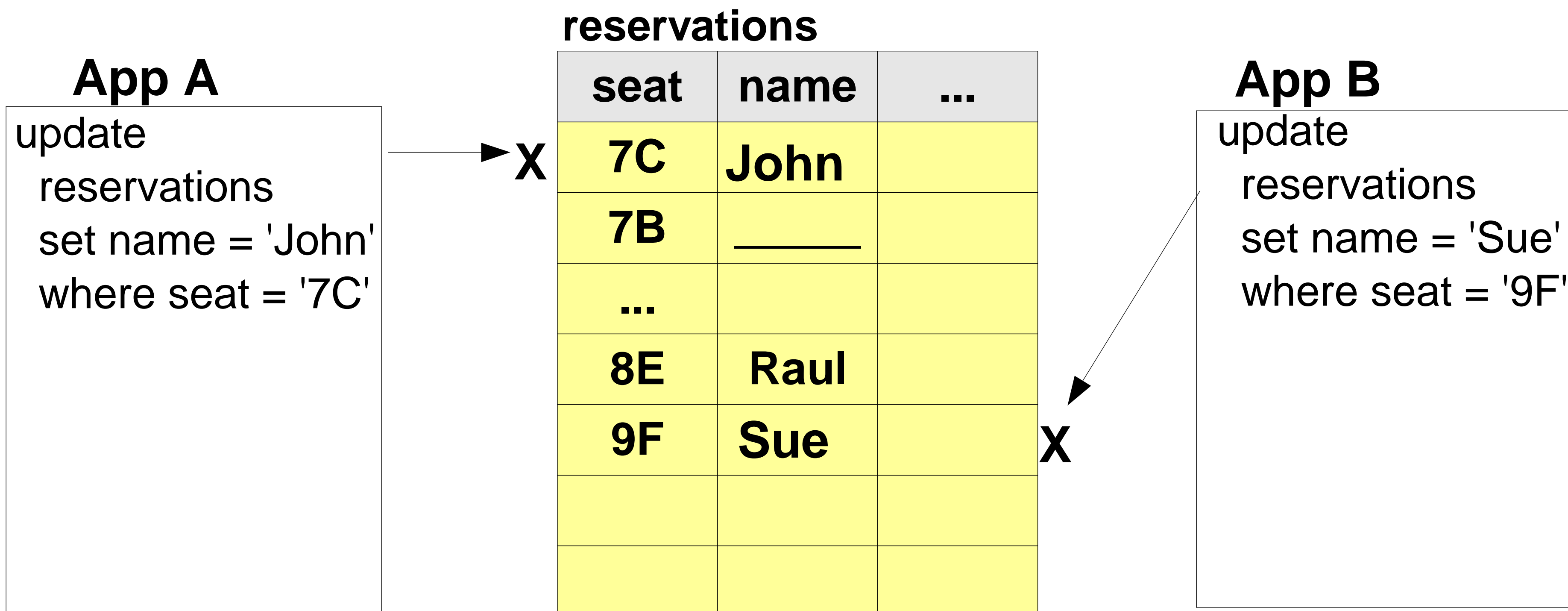
- Occurs when two or more applications wait indefinitely for a resource
- Each application is holding a resource that the other needs
- Waiting is never resolved
- In the example, assume we are using isolation CS without CC



© 2011 IBM Corporation

Deadlocks

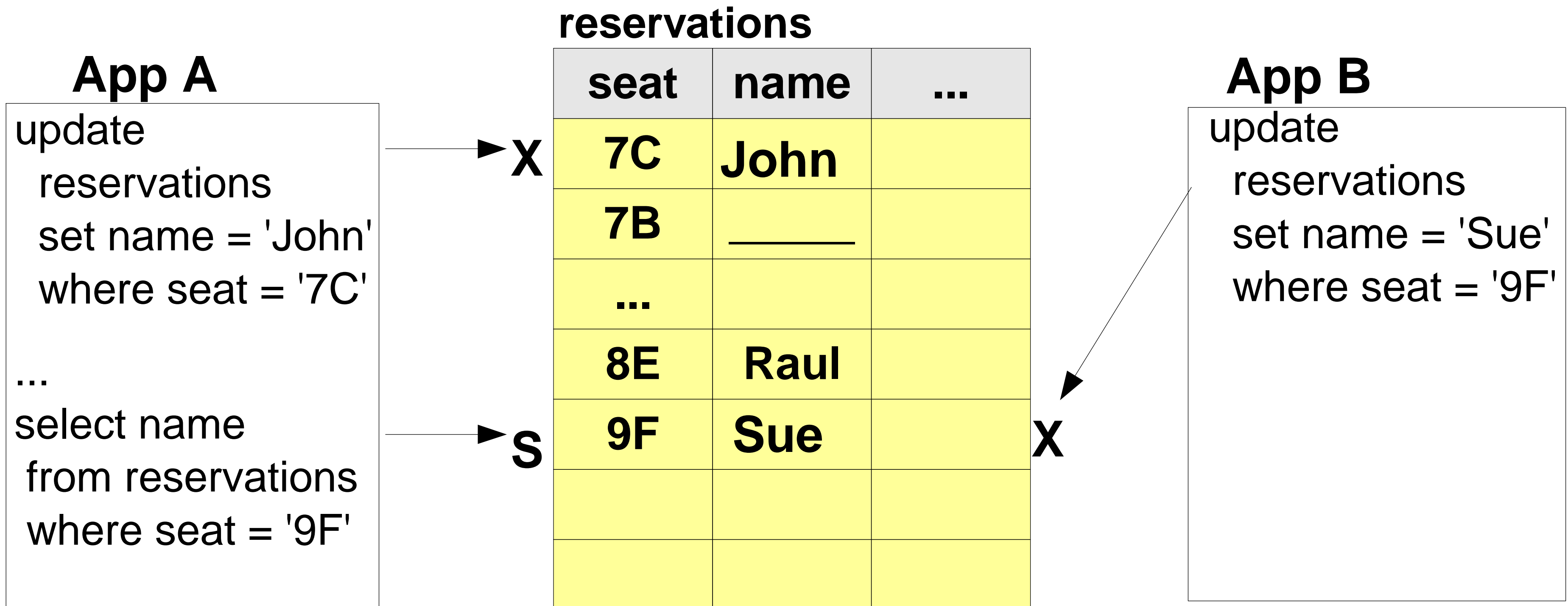
- Occurs when two or more applications wait indefinitely for a resource
- Each application is holding a resource that the other needs
- Waiting is never resolved
- In the example, assume we are using isolation CS without CC



© 2011 IBM Corporation

Deadlocks

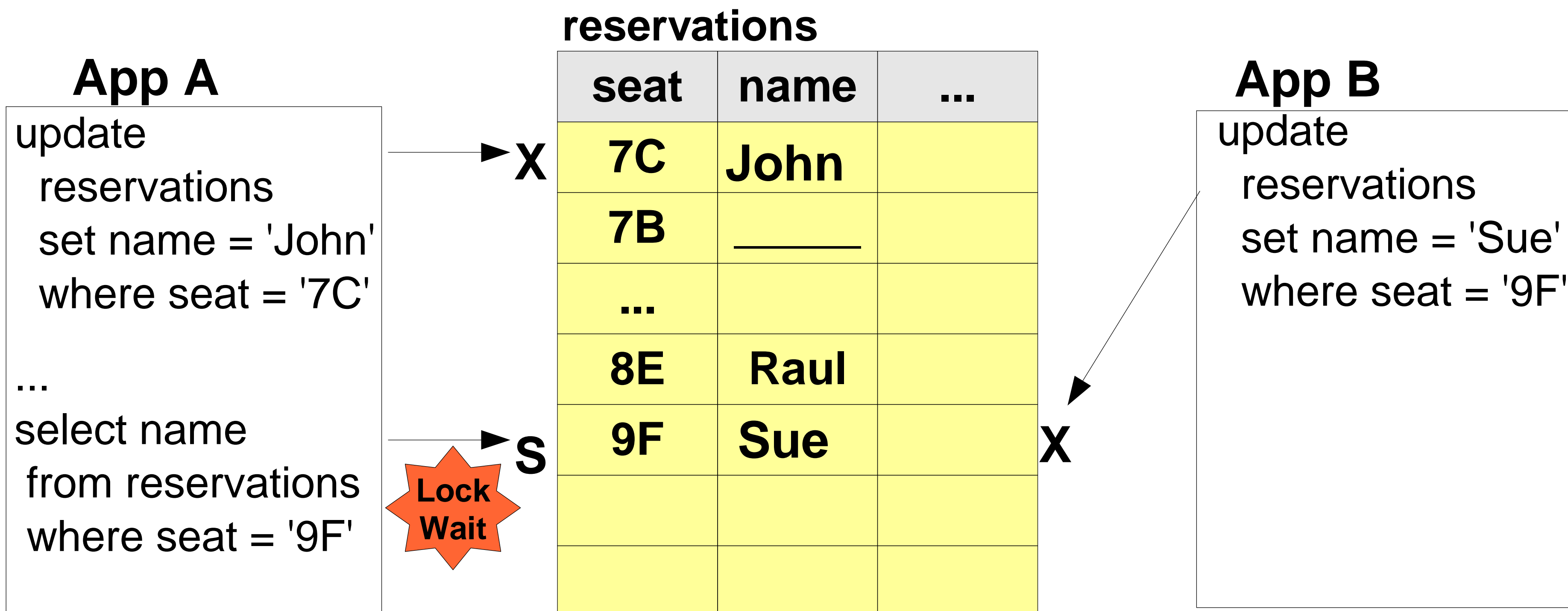
- Occurs when two or more applications wait indefinitely for a resource
- Each application is holding a resource that the other needs
- Waiting is never resolved
- In the example, assume we are using isolation CS without CC



© 2011 IBM Corporation

Deadlocks

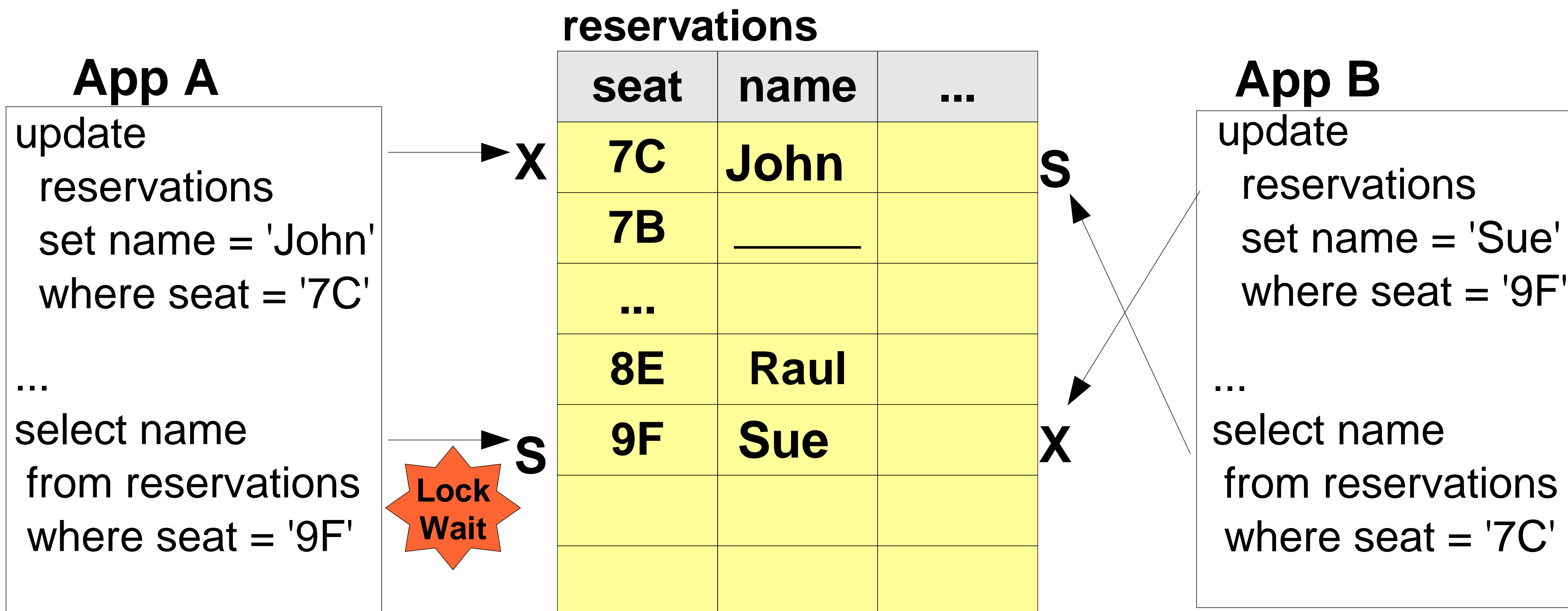
- Occurs when two or more applications wait indefinitely for a resource
- Each application is holding a resource that the other needs
- Waiting is never resolved
- In the example, assume we are using isolation CS without CC



© 2011 IBM Corporation

Deadlocks

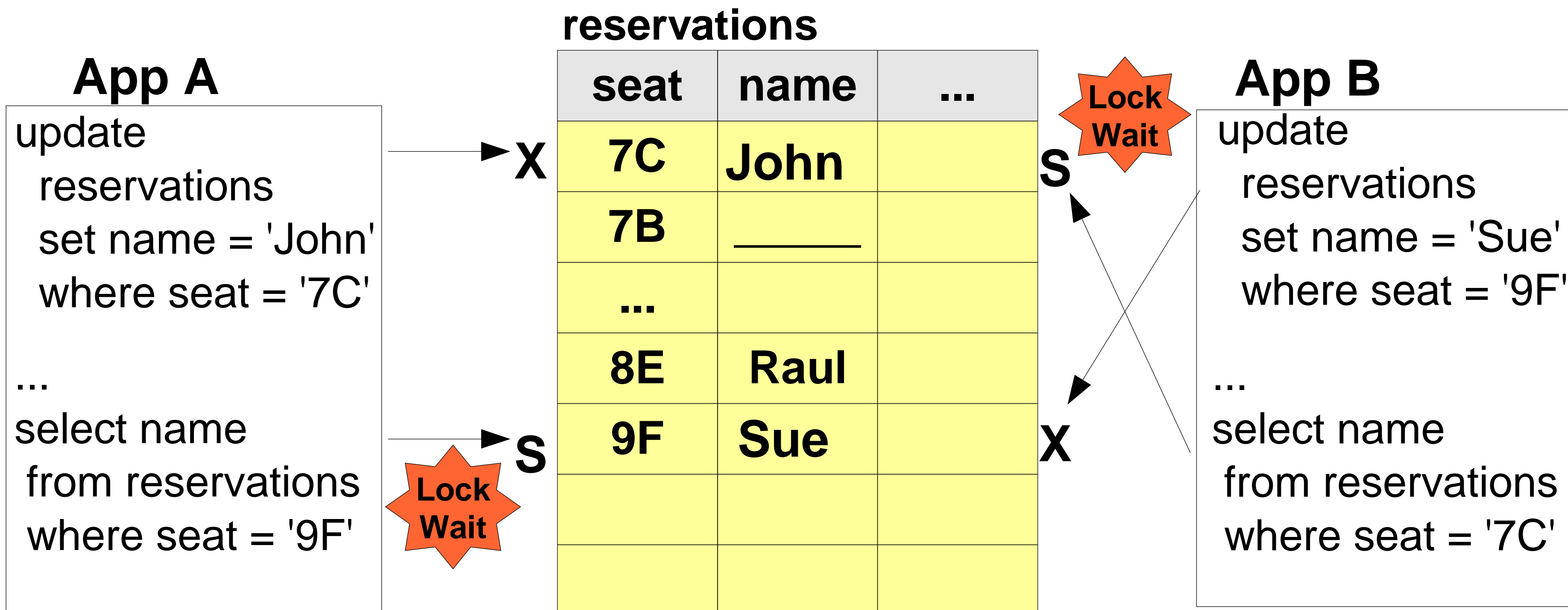
- Occurs when two or more applications wait indefinitely for a resource
- Each application is holding a resource that the other needs
- Waiting is never resolved
- In the example, assume we are using isolation CS without CC



© 2011 IBM Corporation

Deadlocks

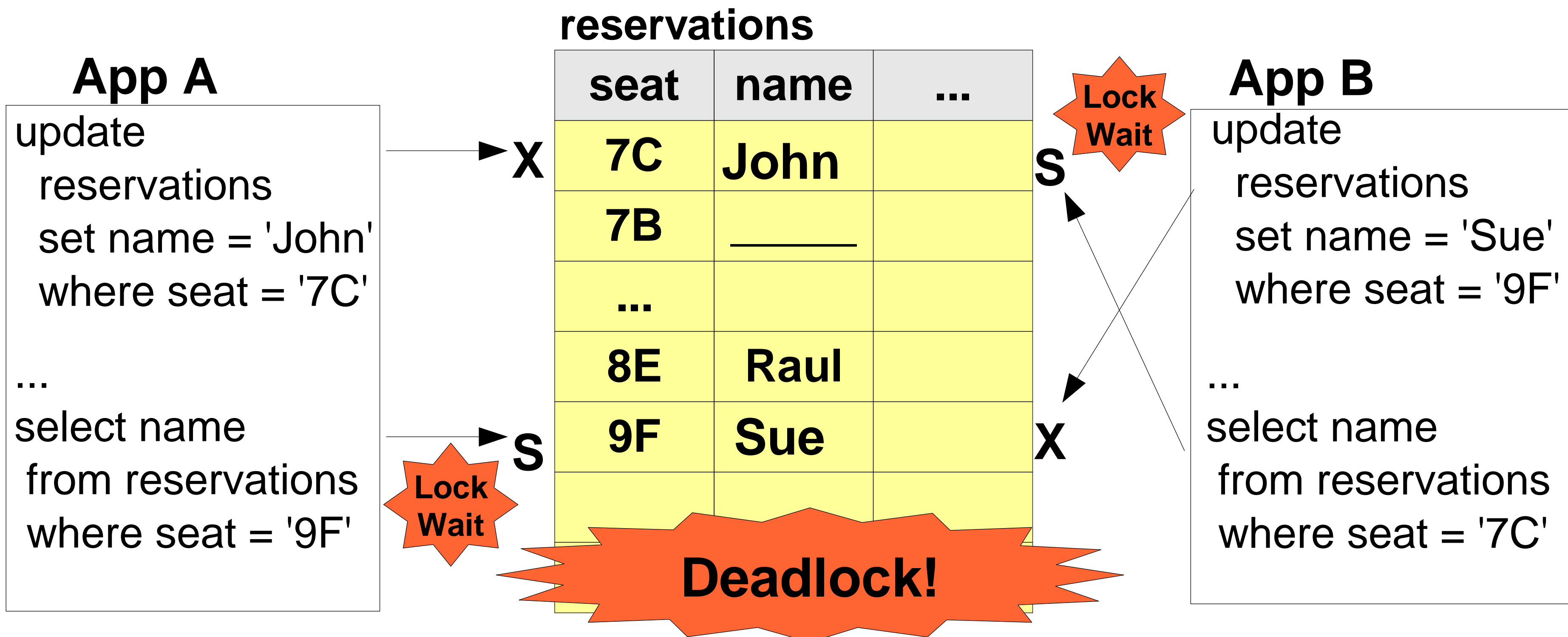
- Occurs when two or more applications wait indefinitely for a resource
- Each application is holding a resource that the other needs
- Waiting is never resolved
- In the example, assume we are using isolation CS without CC



© 2011 IBM Corporation

Deadlocks

- Occurs when two or more applications wait indefinitely for a resource
- Each application is holding a resource that the other needs
- Waiting is never resolved
- In the example, assume we are using isolation CS without CC



© 2011 IBM Corporation

Deadlocks

- **Deadlocks are commonly caused by poor application design**
- **DB2 provides a deadlock detector**
  - Use **DLCHKTIME** (db cfg) to set the time interval for checking for deadlocks
  - When a deadlock is detected, DB2 uses an internal algorithm to pick which transaction to roll back, and which one to continue.
  - The transaction that is forced to roll back gets a SQL error. The rollback causes all of its locks to be released

© 2011 IBM Corporation

Thank you!