3C03 Concurrency: Databases Concurrency Control

Wolfgang Emmerich

Outline

- *Motivation*
- *Locking and Lock Compatibility*
- *Two-Phase Locking*
- *Hierarchical Locking*
- *Implicit vs. Explicit Locking*
- *CORBA Concurrency Control Service*

Motivation

- ***Components of distributed systems use shared resources concurrently:***
  - ***Hardware Components***
  - ***Operating system resources***
  - ***Databases***
  - ***Objects***
- ***Resources may have to be accessed in mutual exclusion.***

Motivation

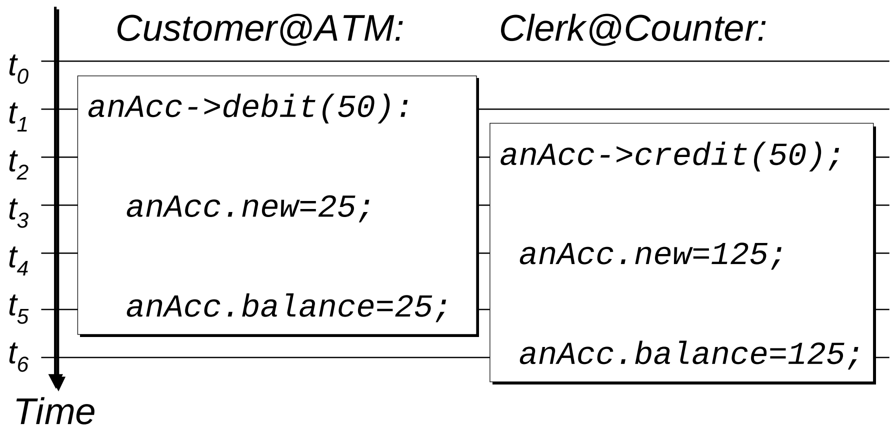
- ***Concurrent access and updates of resources may lead to:***
  - ***lost updates***
  - ***inconsistent analysis.***
- ***Example for lost updates:***
  - ***Cash withdrawal from ATM and concurrent***
  - ***Credit of cheque.***
- ***Example for inconsistent analysis:***
  - ***Funds transfer between accounts of a customer***
  - ***Sum of account balances (Report for Inland Revenue).***

Motivating Examples

```
class Account {
  protected:
    float balance;
  public:
    float get_balance() {return balance;};
    void debit(float amount){
        float new=balance-amount;
        balance=new;
    };
    void credit(float amount) {
      float new=balance+amount;
      balance=new;
    };
};
```

Lost Updates

Balance of account anAcc at t_0 is 75

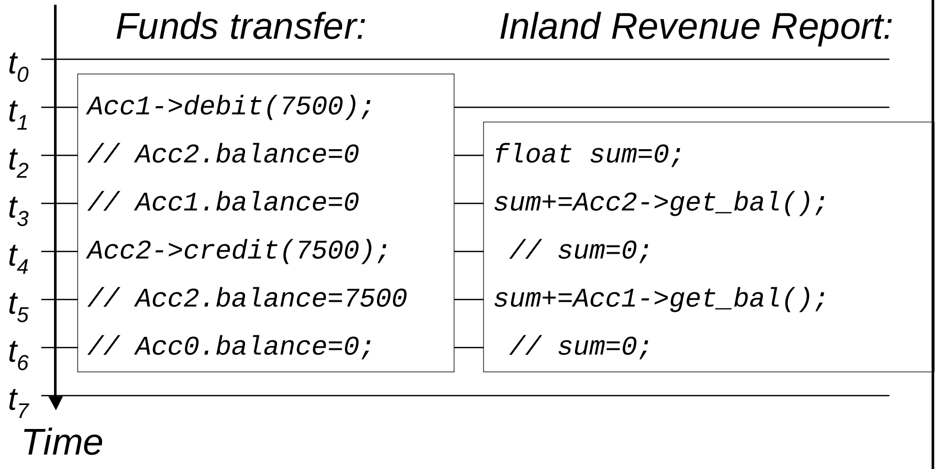


```
anAcc->debit(50):
  anAcc.new=25;
  anAcc.balance=25;
```

```
anAcc->credit(50);
  anAcc.new=125;
  anAcc.balance=125;
```

Inconsistent Analysis

Balances at t_0 `Acc1: 7500, Acc2: 0`



Funds transfer:

```
Acc1->debit(7500);
// Acc2.balance=0
// Acc1.balance=0
Acc2->credit(7500);
// Acc2.balance=7500
// Acc0.balance=0;
```

Inland Revenue Report:

```
float sum=0;
sum+=Acc2->get_bal();
 // sum=0;
sum+=Acc1->get_bal();
 // sum=0;
```

Two Phase Locking (2PL)

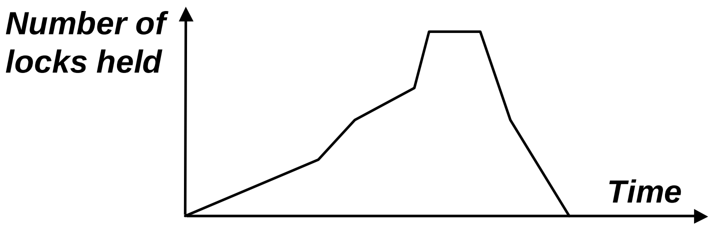
- ***The most popular concurrency control technique. Used in:***
  - ***RDBMSs (Oracle, Ingres, Sybase, DB/2, etc.)***
  - ***ODBMSs (O2, ObjectStore, Versant, etc.)***
  - ***Transaction Monitors (CICS, etc)***
- ***Concurrent processes acquire locks on shared resources from lock manager.***
- ***Lock manager grants lock if request does not conflict with already granted locks.***
- ***Guarantees serialisability.***

Locks

- A lock is a token that indicates that a process accesses a resource in a particular mode.
- Minimal lock modes: read and write.
- Locks are used to indicate to concurrent processes the current use of that resource.

Locking

- Processes acquire locks before they access shared resources and release locks afterwards.
- 2PL: Processes do not acquire locks once they have released a lock.
- Typical 2PL locking profile of a process:



Lock Compatibility

- *Lock manager grants locks.*
- *Grant depends on compatibility of acquisition request with modes of already granted locks.*
- *Compatibility defined in lock compatibility matrix.*
- *Minimal lock compatibility matrix:*

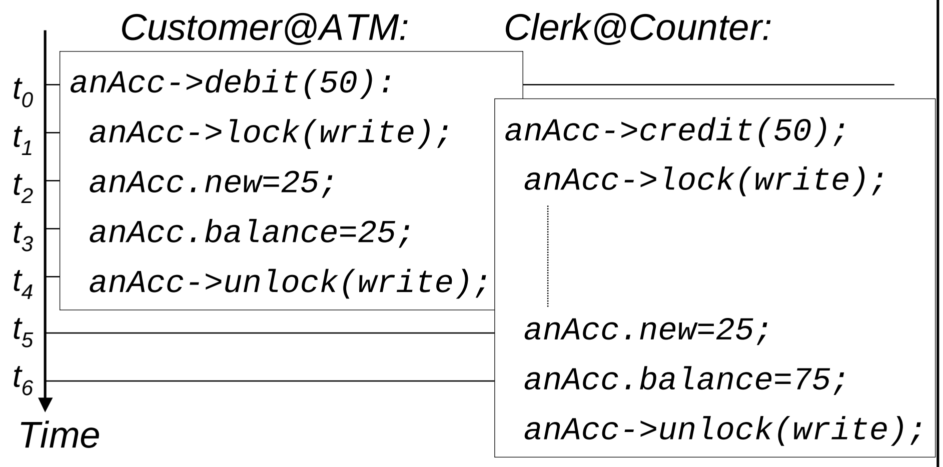
	Read	Write
Read	+	-
Write	-	-

Locking Conflicts

- *Lock requests cannot be granted if incompatible locks are held by concurrent processes.*
- *This is referred to as a locking conflict.*
- *Approaches to handle conflicts:*
  - *Force requesting process to wait until conflicting locks are released.*
  - *Tell process or thread that lock cannot be granted.*

Example (Avoiding Lost Updates)

Balance of account anAcc at t_0 is 75



```
anAcc->debit(50):
  anAcc->lock(write);
  anAcc.new=25;
  anAcc.balance=25;
  anAcc->unlock(write);
```

```
anAcc->credit(50);
  anAcc->lock(write);


  anAcc.new=25;
  anAcc.balance=75;
  anAcc->unlock(write);
```

Deadlocks

- ***2PL may lead to processes waiting for each other to release locks.***
- ***These situations are called deadlocks.***
- ***Deadlocks have to be detected by the lock manager.***
- ***Deadlocks have to be resolved by aborting one or several of the processes involved.***
- ***This requires to undo all the actions that these processes have done.***

Locking Granularity

- ***2PL applicable to resources of any granularity.***
- ***High degree of concurrency with small locking granularity.***
- ***For small granules large number of locks required.***
- ***May involve significant locking overhead.***
- ***Trade-off between degree of concurrency and locking overhead.***
- ***Hierarchical locking as a compromise.***

Hierarchical Locking

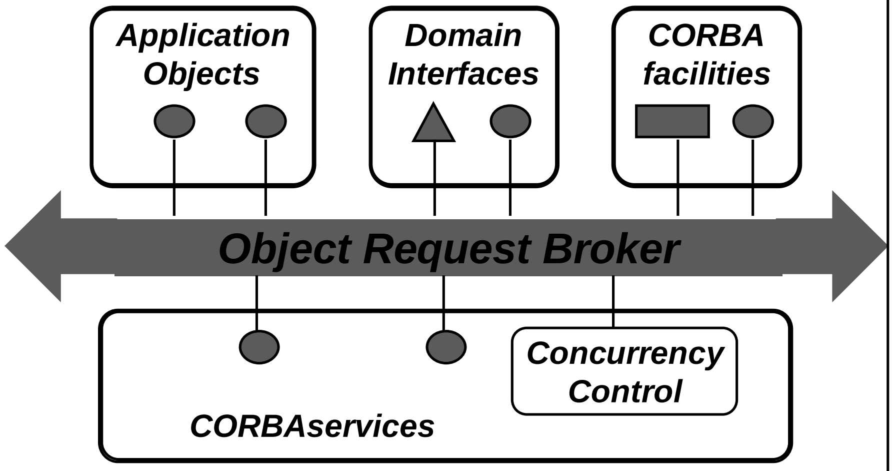
- ***Used with container resources, e.g.***
  - ***file (containing records)***
  - ***set or sequence (containing objects)***
- ***Lock modes intention read (IR) and intention write (IW).***
- ***Lock compatibility:***

	IR	R	IW	W
IR	+	+	+	-
R	+	+	-	-
IW	+	-	+	-
W	-	-	-	-

Transparency of Locking

- ***Who is acquiring locks?***
  - *Concurrency control infrastructure*
  - *Implementation of components*
  - *Clients of components*
- ***First option desireable but not always possible:***
  - *Infrastructure must manage all resources*
  - *Infrastructure must know all resource accesses.*
- ***Last option is undesirable and avoidable!***

CORBA Concurrency Control Service



Lock Compatibility Matrix

- ***CORBA concurrency control service supports hierarchical locking.***
- ***Upgrade locks for decreasing probability of deadlocks.***
- ***Compatibility matrix:***

	IR	R	U	IW	W
IR	+	+	+	+	-
R	+	+	+	-	-
U	+	+	-	-	-
IW	+	-	-	+	-
W	-	-	-	-	-

Locksets

- ***A lockset is associated to a resource (usually in the implementation of that resource).***
- ***Each shared resource has a lockset.***
- ***Operations of that resource acquire locks before they access or modify the resource.***

The IDL Interfaces

```
interface LocksetFactory {
  LockSet create();
};
interface Lockset {
  void lock(in lock_mode mode);
  boolean try_lock(in lock_mode mode);
  void unlock(in lock_mode mode);
  void change_mode(in lock_mode held,
                   in lock_mode new);
};
```

Summary

- *Lost Updates and Inconsistent Analysis*
- *Locking and Lock Compatibility*
- *Two-Phase Locking*
- *Hierarchical Locking*
- *Implicit vs. Explicit Locking*
- *CORBA Concurrency Control Service*