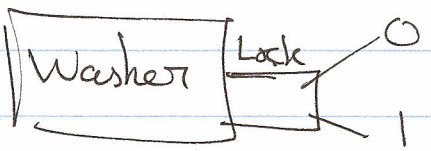
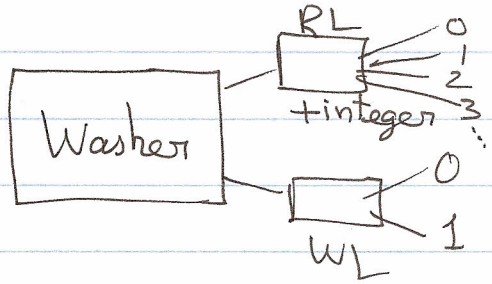
Lock

Binary
(boolean value for the lock)
0 – unlocked
1 – locked.

There is no distinction between read lock & write lock



Read lock (counter) & write lock (boolean)



RL & WL are mutually exclusive.

Binary Lock

Lock (item)
Unlock (item)

Lock(item):

```
Start: if (Lock(item) == 0)
       { // item is unlocked
           Lock(item) = 1
       }
       else // item is locked.
       {
           wait (until Lock(item) == 0) // process goes to sleep
           Lock manager wakes up the transaction
                when the lock is released.
       } goto start
```

Unlock(item):

```
Lock(item) = 0
Signal (activate) Lock Manager
 so that Lock Manage will wake up
   transactions that are waiting for this item.
```

Read Lock & Write Lock

Mode:

Read Locked (shared) with no. of readers.

Write Locked (Exclusive): 0 or 1

Unlocked

ReadLock(item), WriteLock(item), Unlock(item)

ReadLock(item):

```
Start: if (Lock(item) == 'Unlocked') {
           Lock(item) = 'Readlocked'
           no_of_readers = 1
       }
       else if (Lock(item) == 'Readlocked') {
           no_of_readers++
       }
       else // item is writelocked
       { wait (until Lock(item) == 'Unlocked'
           Lock Manager wakes up the
           transaction
       } goto start
```

WriteLock(item):

```
Start: if (Lock(item) == 'Unlocked')
       { Lock(item) = 'writelocked' }
       else{
           wait (until Lock(item) == 'Unlocked')
           Lock Manager wakes up the transaction
       } goto start
```

~~Unlock(item):~~

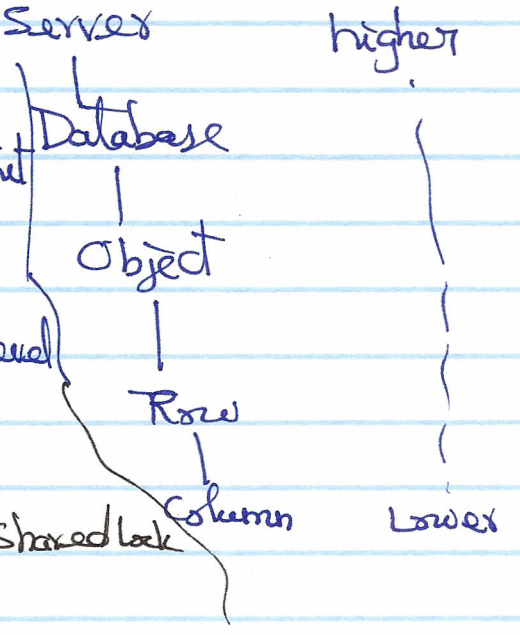
Unlock(item): (Read lock & write lock)

```
if (Lock(item) == 'Writelocked')
 {
    Lock(item) = 'Unlocked'
     Signal (activate) Lock Manager
      so that it will wake up transactions
          that are waiting for this item
 }
 else { // read locked
       No_of_readers--
        if (No_of_readers == 0)
          { Lock(item) = 'Unlocked'
            Signal (activate) Lock Manager
             so that it will wake up transactions
                that are waiting for this item
          }
```

Granularity of Locks

- Server — level
  - Database Level
    - Object (table) — Level
      - row — level
        - column — Level

Intention Locks (meaningful in a hierarchy of objects)



IS (intention shared)

placed at a higher level unit and Shared lock can be placed at a lower level

e.g. Company DB — IS

Employee table — Shared lock

IE (intention exclusive)

e.g.

T_4: Employee table — IX

on a specific row — Exclusive lock.

T_6: Read on ~~some~~ all rows ~~row~~ of employee table

readlock(row)

→ Check the states on Employee table: IX

T_6 understands all may not be available for read.

Lost update Example

not following 2-PL

T_1

+ Lock(X)
read(X)
X = X − N
Write(X)
− Unlock(X)
+ Lock(Y)
Read(Y)
Y = Y + N
Write(Y)
− Unlock(Y)

T_2

+ Lock(X)
Read(X)
X = X + M
Write(X)
− Unlock(X)

T_1

Following Basic 2-PL

+ Lock(X)
read(X)
Growth phase
X = X − N
Write(X)
+ Lock(Y)
Read(Y)
Shrink phase
− Unlock(X)
Y = Y + N
Write(Y)
− Unlock(Y)

Two phase Locking Protocol

Each transaction has two distinct phases:

1. Growth phase
   Acquires locks for DB items w/o releasing any locks
2. Shrink phase
   Releases locks of DB items w/o acquiring new locks

Benefit: Any schedule made of transactions that follows two-phase locking protocol is guaranteed to be conflict serializable.

However a deadlock can occur with two phase locking protocol transactions.

T1

Conservative 2PL

+ Lock(X)
+ Lock(Y)
read(X)
X=X-N
write(X)
- UnLock(X)
Read(Y)
Y=Y+N
Write(Y)
- Unlock(Y)

This will not lead to Deadlock

T3

Strict 2PL

+RLock(A)
Read(A)
+WLock(B)
Read(B)
B *= B+100
- Unlock(A)
Write(B)
Commit.
- Unlock(B)

Growth Phase

Shrink Phase

T3

Rigorous 2PL

+RLock(A)
Read(A)
+WLock(B)
Read(B)
B=B+100
Write(B)
Commit
-Unlock(B)
-Unlock(A)

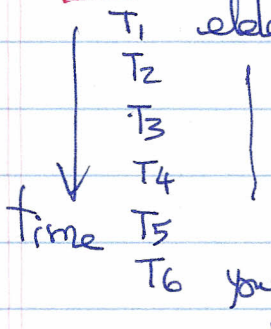
DeadLock

Using Basic 2PL

	Manufacturing T5 (Ryan)		Sales T6 (Elena)	
t_1	Lock (Washer) ✓	GrP		
t_2			Lock(Dryer) ✓	
t_3	Read (washer) ✓			
t_4			Read (Dryer) ✓	GrP
t_5	Lock (Dryer) ? wait			
t_6	Deadlock state		Lock (Washer) ? wait	
t_7	Read (Dryer)	SP	Read (Washer)	
	Unlock (Washer)		Unlock (Dryer)	SP
	Unlock (Dryer)		Unlock (Washer)	

Deadlock Prevention

With Time stamp (chronological order)



a) Wait-Die

	Elder T_5 (Ryan)	Younger T_6 (Elena)
t_1	Lock(Washer) ✓	
t_2		Lock(Dryer) ✓
t_3	Read(Washer) ✓	
t_4	Lock(T_6 Dryer)? wait	Read(Dryer) ✓
t_5		Lock(T_5 Washer)? selfabort
t_6	Read(Dryer) ✓	
	Unlock(Washer) ✓	
	Unlock(Dryer) ✓	

unlock(Dryer) → rollback

T6 gets ~~restricted~~ restarted with previous timestamp T_6

b) Wound-wait

	T_5 (Ryan)	T_6 (Elena)
t_1	Lock(Washer) ✓	
t_2		Lock(Dryer) ✓
t_3	Read(Washer) ✓	
t_4		Read(Dryer) ✓
t_5	Lock(T_6 Dryer)? abort → T6 gets aborted	
t_6	Read(Dryer) ✓	
	Unlock(Washer) ✓	
	Unlock(Dryer) ✓	

unlock(Dryer) → rollback

T6 gets restarted with the same timestamp

Timestamp Ordering

(no locks are used)

For each active DB item (i.e. DB item in memory)
a read time stamp (RTS)
and a write time stamp (WTS) are associated with the item.

RTS: for an item:
The TS of the youngest transaction that read the item.

WTS: for an item:
The TS of the youngest transaction that performed write operation on the item.

T_1 eldest
T_2
T_3
T_4
time
youngest

		RTS
t_1	T_1.Read(A)	1
t_2	T_4.Read(A)	4
t_3	T_2.Read(A)	4
t_4	T_5.Write(A)	WTS 5
t_5	T_9.Write(A)	9
t_6	T_7.Write(A)	9