

Concurrency Control (critical section)

Lock

granularity

- table
- database
- row
- field

Binary Locks:

lock_item(X):

B: if LOCK (X) = 0 (*item is unlocked*)
then LOCK (X) := 1 (*lock the item*)

else

begin

wait (until lock (X) = 0) and
the lock manager wakes up the transaction);

goto B

end;

unlock_item(X):

LOCK (X) := 0 (*unlock the item*)

if any transactions are waiting then

wake up one of the waiting the transactions;

Read/Write Locks

```
read_lock(X):  
B: if LOCK (X) = "unlocked" then  
    begin LOCK (X) := "read-locked";  
        no_of_reads (X) := 1;  
    end  
else if LOCK (X) = "read-locked" then  
    no_of_reads (X) := no_of_reads (X) + 1  
else  
    begin  
        wait (until LOCK (X) = "unlocked" and  
            the lock manager wakes up the transaction);  
        go to B  
    end;  
end;
```

```
write_lock(X):  
B: if LOCK (X) = "unlocked" then  
    LOCK(X) := "write_locked"  
else  
    begin  
        wait(until LOCK(X) = "unlocked")  
        go to B  
    end
```

```

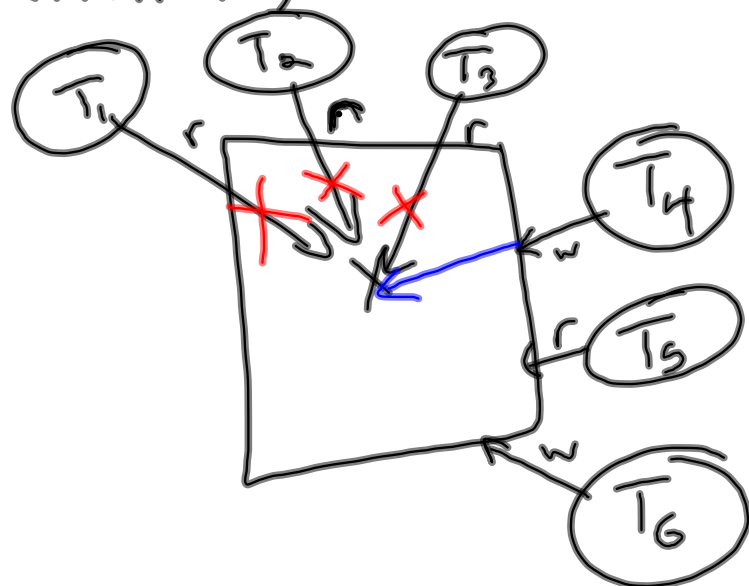
unlock(X):
if LOCK (X) = "write-locked" then
    begin LOCK (X) := "unlocked";
        wakes up one of the transactions, if any
    end
else if LOCK (X) "read-locked" then
    begin
        no_of_reads (X) := no_of_reads (X) -1
        if no_of_reads (X) = 0 then
            begin
                LOCK (X) = "unlocked";
                wake up one of the transactions, if any
            end
        end
    end;
end;

```

lock-item (x)

Read (x)

unlock-item (x)



upgrade read to write lock
downgrade write to read lock

Two phase locking protocol

Locking (growing) phase
- locking and upgrading

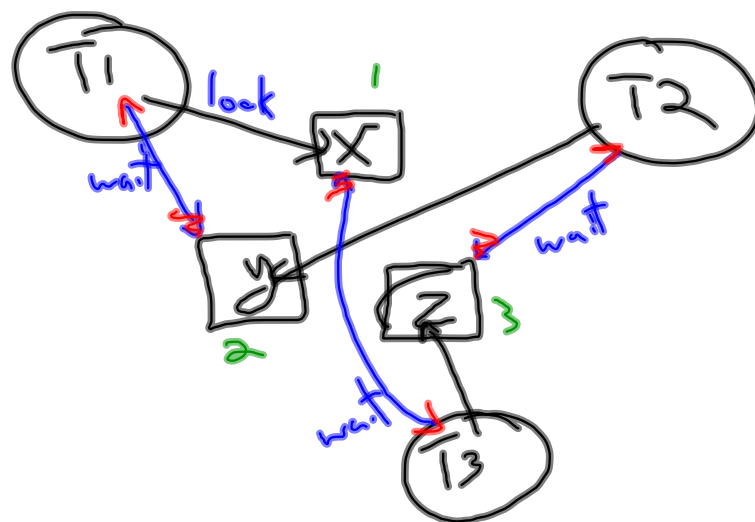
Unlocking (shrinking) phase
- unlocking and downgrading

Basic: phase 1 completes
before phase 2 starts

Conservative: lock every resource used,
prior to beginning of trans.

Strict: unlock everything
after commit/abort/rollback

Deadlock



- lock all items
 - if you can't get a lock, quit/retry (release locks)
- order all resources
 - transactions must lock in numerical order
- cautious wait
 - only wait on a resource held a transaction that is not waiting
- detect - "wait for graph"
 - find cycles
 - kill a transaction involved
- avoid - make sure a request does not create a cycle

Starvation

- higher priority trans. go first
- trans. keeps restarting