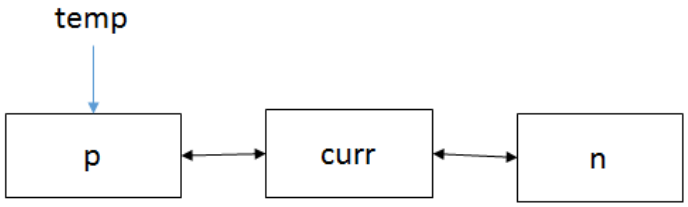
Sneha Banerjee
snehaban@buffalo.edu

# CSE505: Assignment 2 – RW Locks Revisited

Part 2:

```
temp = curr.p;                                  ---------- 1
synchronized (temp)                             ---------- 2
{
    synchronized (curr)                         ---------- 3
    {
        If (temp == curr.p)                     ---------- 4
        {
            synchronized (curr.n)               ---------- 5
            {
                …
            }
        }
    }
}
```




1. Should p be marked volatile in the class definition for Element?
   - **Yes,** p needs to be marked volatile. The reference to p is stored in a local variable temp, which does not change after Line 1 and a lock can be acquired on temp on line 2, if it is not already locked by another thread.
   - Consider curr.p is locked and its value has been changed by a thread. Now just before this thread releases its lock, another thread reads *temp = curr.p*. If p wasn't declared volatile, its value would not have been read immediately into the temp variable before the previous thread released its lock.
   - In this case, temp == curr.p (*line 4*) would be false, because temp would have read the old value of p, leading to a data race. To prevent this, p needs to be declared volatile.

2. Should n be marked volatile in the class definition for Element?
   - **No,** n does not need to be marked volatile. Once a lock has been acquired on the curr variable (*Line 3*), no other thread can modify this object or its variables p and n until its lock has been released.
   - Even if any other thread holds a lock on n before the previous thread can acquire a lock on line 5, another element cannot be inserted between curr and n, since curr is locked by another thread.

This prevents any inconsistencies and data races with respect to n. Thus, n doesn't need to be marked volatile.

3. Should the temporary variable temp be marked volatile?
   - **No,** the temporary variable does not need to be marked volatile. The temporary variable is local to a thread and is being written to (*Line 1*) only once, but read multiple times (*Lines 2 and 4*) by the same thread.
   - Since this variable is local to a thread, other threads do not read it or modify it, preventing any inconsistencies and data races.
   - This variable is not changing once its value has been assigned and hence can be stored in the local cache, thus eliminating the need to mark it as volatile.

---

## Part 3:

Difference between synchronized blocks and methods:

1. Synchronized methods acquire a single lock on the entire class object, whereas synchronized blocks can acquire different locks on different objects or variables. Synchronized blocks can also lock the class object using the keyword 'this' as shown below.
2. Synchronized blocks provide more concurrency and flexibility. Different locks can be acquired on different objects simultaneously whereas the entire object is locked in case of a synchronized method, thus restricting concurrent access.
3. Synchronized method provides coarse-grained locking, whereas synchronized block provides fine-grained locking.

A synchronized block can be encoded as a synchronized method as shown below. The lock is obtained on the class object using the keyword 'this'.

Synchronized Block:

```
public synchronized int read()
{
    return data;
}
```

Synchronized Method:

```
public int read()
{
    synchronized(this)
    {
        return data;
    }
}
```