

A4: Peterson Mutex

```
public class PetersonMutex implements Mutex {
    private boolean[] enter = { false, false };
    private int turn = 0;
    // ...

    public void lock() {
        int index = getIndex();
        int otherIndex = 1 - index;

        enter[index] = true;
        turn = otherIndex;
        while ( enter[otherIndex] && turn != index) { }
    }

    public void unlock() {
        enter[getIndex()] = false;
    }
}
```

A4: Peterson Mutex: Visibility Problems

```
public void lock() {  
    int index = getIndex(); int otherIndex = 1 - index;  
    enter[index] = true;    // (1)  
    turn = otherIndex;      // (3)  
    while ( enter[otherIndex] && turn != index) { }  
}  
  
public void unlock() { enter[getIndex()] = false; } // (2)
```

- **Problems**

- (1) Is value stored to enter[index] in lock seen by the other thread?
- (2) Is value stored to enter[getIndex()] in unlock seen by this thread?
- (3) Is value stored to turn in lock seen by the other thread?

A4: Peterson Mutex: Visibility Problems

T0:

```
boolean[] enter = [false, false]
int turn        = 0
```

T1:

```
public void lock() {
    int index = 0;
    int otherIndex = 1;
    enter[0] = true;
    turn = 1;
    while ( enter[1] &&
            turn != 0) {}
}

public void unlock() {
    enter[0] = false;
}
```

```
public void lock() {
    int index = 1;
    int otherIndex = 0;
    enter[1] = true;
    turn = 0;
    while ( enter[0] &&
            turn != 1) {}
}

public void unlock() {
    enter[1] = false;
}
```

- (1) Is value stored to enter[index] in lock seen by the other thread?
- (2) Is value stored to enter[getIndex()] in unlock seen by this thread?
- (3) Is value stored to turn in lock seen by the other thread?

A4: Peterson Mutex: Version 1

```
public class PetersonMutex implements Mutex {  
    private boolean[] enter = { false, false };  
    private volatile int turn = 0;  
    public void lock() {  
        int index = getIndex(); int otherIndex = 1 - index;  
        enter[index] = true; // (1)  
        turn = otherIndex; // (3)  
        while ( enter[otherIndex] && turn != index) { }  
    }  
  
    public void unlock() { enter[getIndex()] = false; } // (2)  
}
```

- **Comments**

- Solves (3), turn is seen by any other thread
- Does not solve (2) as turn is not written in unlock
- Does not solve (1) as turn is not read before enter[otherIndex] is read

A4: Peterson Mutex: Visibility Problems

T0:

```
boolean[] enter = [false, false]
volatile int turn = 0
```

T1:

```
public void lock() {
    int index = 0;
    int otherIndex = 1;
    enter[0] = true;
    turn = 1;
    while ( enter[1] &&
            turn != 0) {}
}

public void unlock() {
    enter[0] = false;
}
```

hb
(3)

```
public void lock() {
    int index = 1;
    int otherIndex = 0;
    enter[1] = true;
    turn = 0;
    while ( enter[0] &&
            turn != 1) {}
}

public void unlock() {
    enter[1] = false;
}
```

- Solves (3), turn is seen by any other thread
- Does not solve (2) as turn is not written in unlock
- Does not solve (1) as turn is not read before enter[otherIndex] is read

A4: Peterson Mutex: Version 2

```
public class PetersonMutex implements Mutex {  
    private boolean[] enter = { false, false };  
    private volatile int turn = 0;  
    public void lock() {  
        int index = getIndex(); int otherIndex = 1 - index;  
        enter[index] = true;    // (1)  
        turn = otherIndex;      // (3)  
        while ( turn != index && enter[otherIndex] ) { }  
    }  
  
    public void unlock() { enter[getIndex()] = false; } // (2)  
}
```

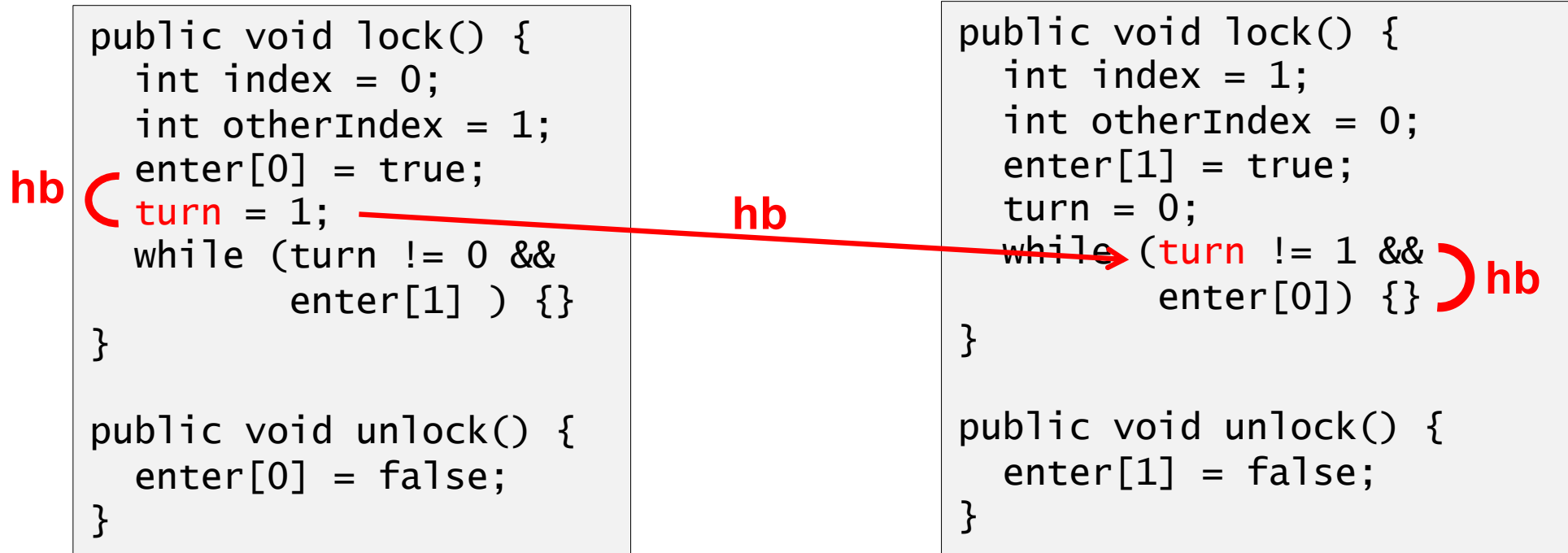
- **Comments**
 - Problems (1) and (3) are solved, but not (2)

A4: Peterson Mutex: Visibility Problems

T0:

```
boolean[] enter = [false, false]
volatile int turn = 0
```

T1:



- Problems (1) and (3) are solved, but not (2)

A4: Peterson Mutex: Version 3

```
public class PetersonMutex implements Mutex {
    private boolean[] enter = { false, false };
    private volatile int turn = 0;
    public void lock() {
        int index = getIndex(); int otherIndex = 1 - index;
        enter[index] = true; // (1)
        turn = otherIndex; // (3)
        while ( turn != index && enter[otherIndex] ) { }
    }

    public void unlock() {
        enter[getIndex()] = false; turn = turn; // (2)
    }
}
```

- **Comments**

- Assignment `turn = turn` guarantees that the changed enter field becomes visible by other threads reading the turn field

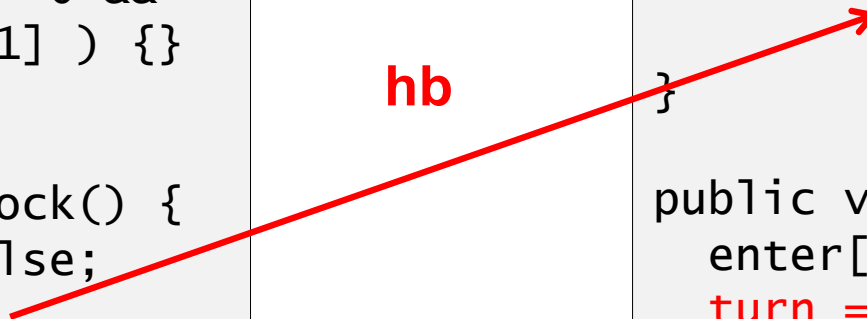
A4: Peterson Mutex: Visibility Problems

T0:

```
boolean[] enter = [false, false]
volatile int turn = 0
```

T1:

```
public void lock() {
    int index = 0;
    int otherIndex = 1;
    enter[0] = true;
    turn = 1;
    while (turn != 0 &&
           enter[1] ) {}
}
```

hb 

```
public void unlock() {
    enter[0] = false;
    turn = turn;
}
```

```
public void lock() {
    int index = 1;
    int otherIndex = 0;
    enter[1] = true;
    turn = 0;
    while (turn != 1 &&
           enter[0]) {} hb
}
```

```
public void unlock() {
    enter[1] = false;
    turn = turn;
}
```

- Assignment `turn = turn` guarantees that the changed enter field becomes visible by other threads reading the turn field

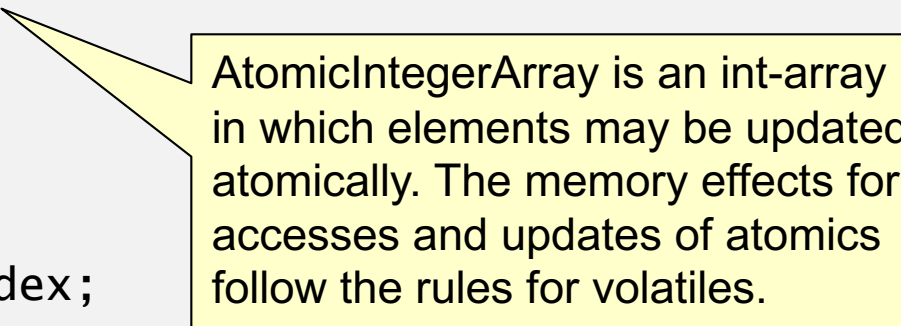
A4: Peterson Mutex: Version 4

```
public class PetersonMutex implements Mutex {
    private static class VolatileRef {
        volatile boolean b = false;
    }
    private final VolatileRef[] enter
        = { new VolatileRef(), new VolatileRef() };
    private volatile int turn = 0;
    // ...
    public void lock() {
        int index = getIndex();
        int otherIndex = 1 - index;

        enter[index].b = true;
        turn = otherIndex;
        while ( enter[otherIndex].b && turn != index ) { }
    }
    public void unlock() { enter[getIndex()].b = false; }
}
```

A4: Peterson Mutex: Version 5

```
public class PetersonMutex implements Mutex {  
    private final AtomicIntegerArray enter  
        = new AtomicIntegerArray(new int[] {0, 0} );  
    private int turn = 0;  
    // ...  
  
    public void lock() {  
        int index = getIndex();  
        int otherIndex = 1 - index;  
  
        turn = otherIndex;  
        enter.set(index, 1);  
        while ( enter.get(otherIndex) == 1 && turn != index) { }  
    }  
  
    public void unlock() { enter.set(getIndex(), 0); }  
}
```



AtomicIntegerArray is an int-array in which elements may be updated atomically. The memory effects for accesses and updates of atomics follow the rules for volatiles.