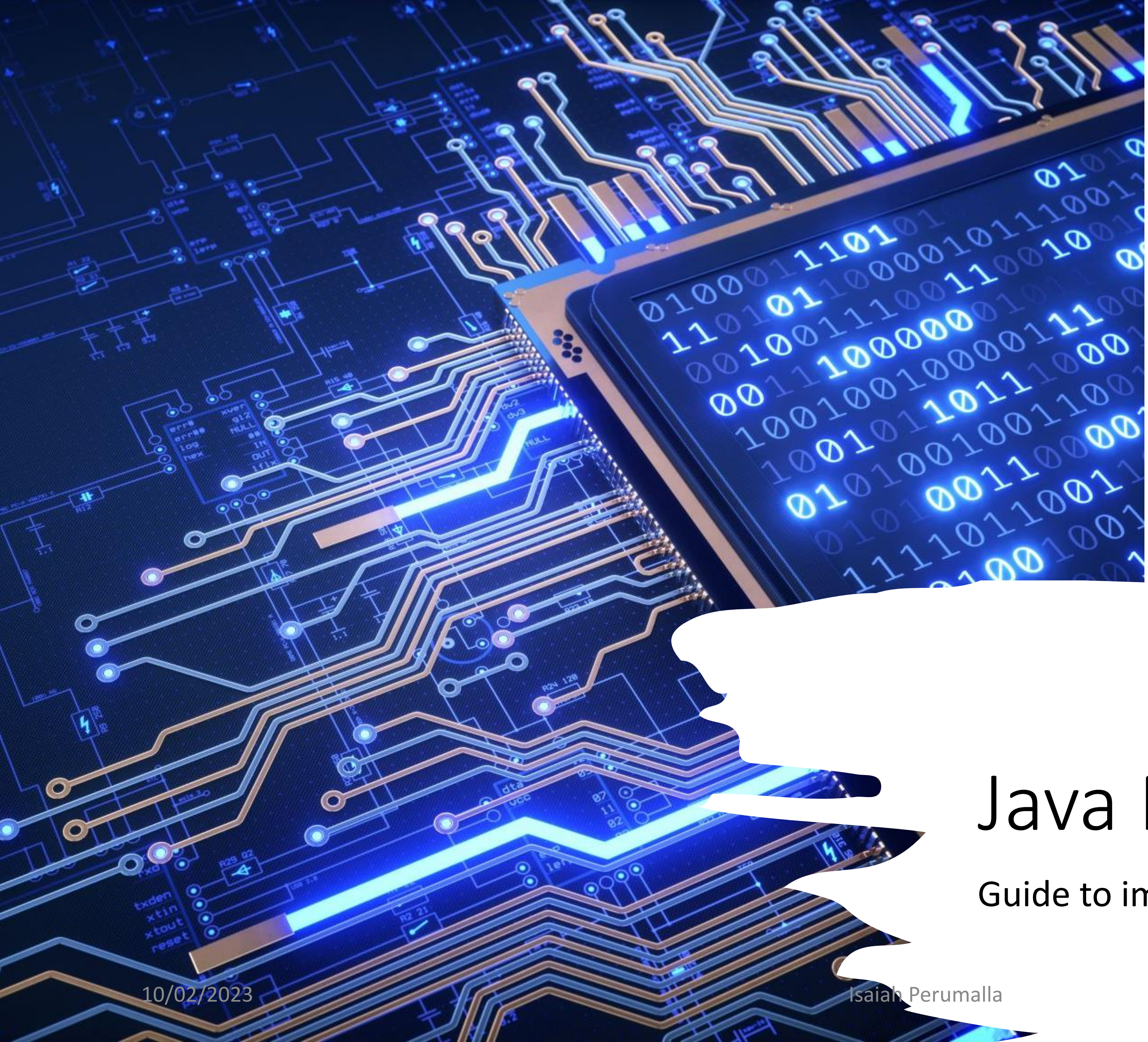
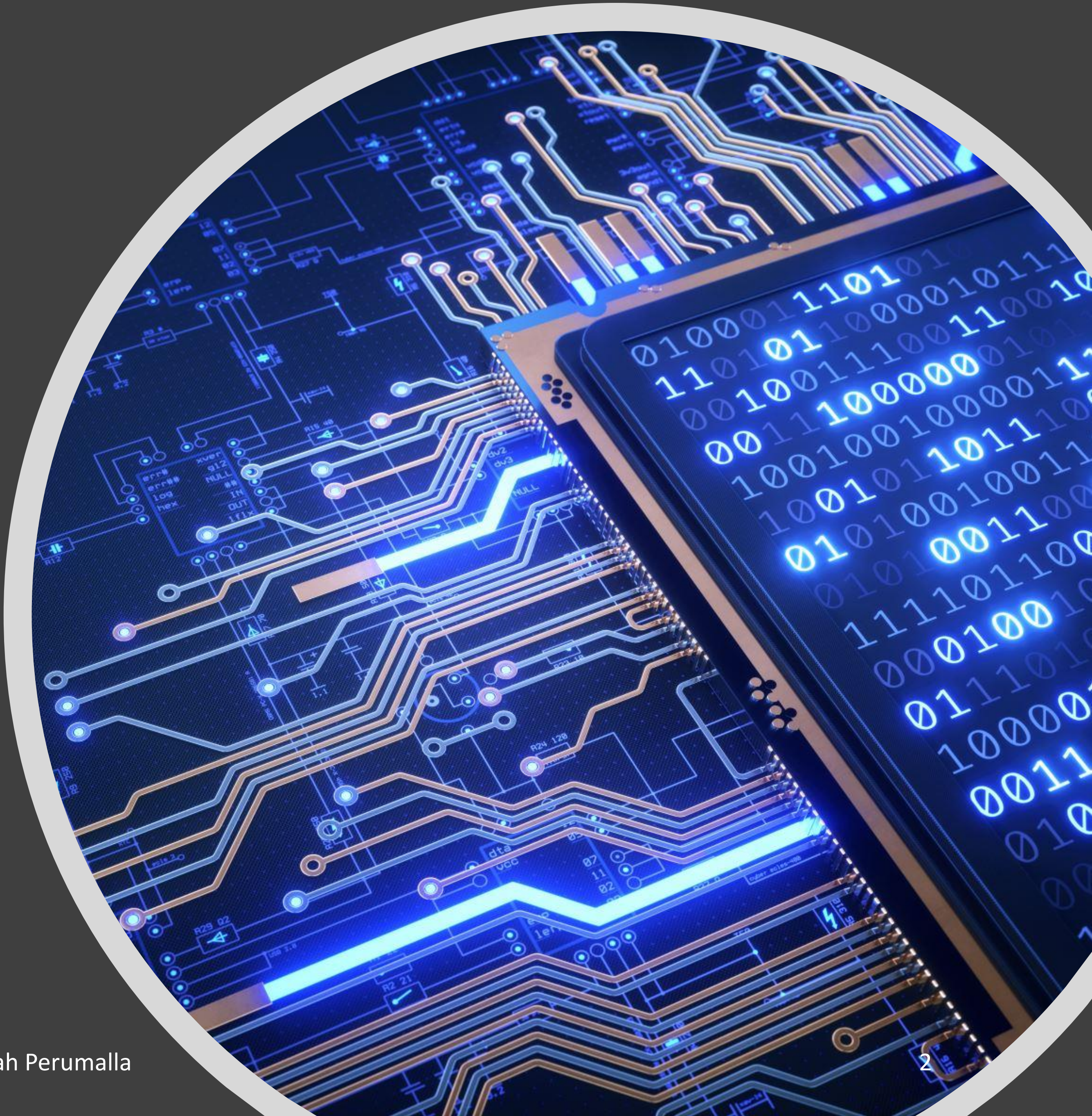


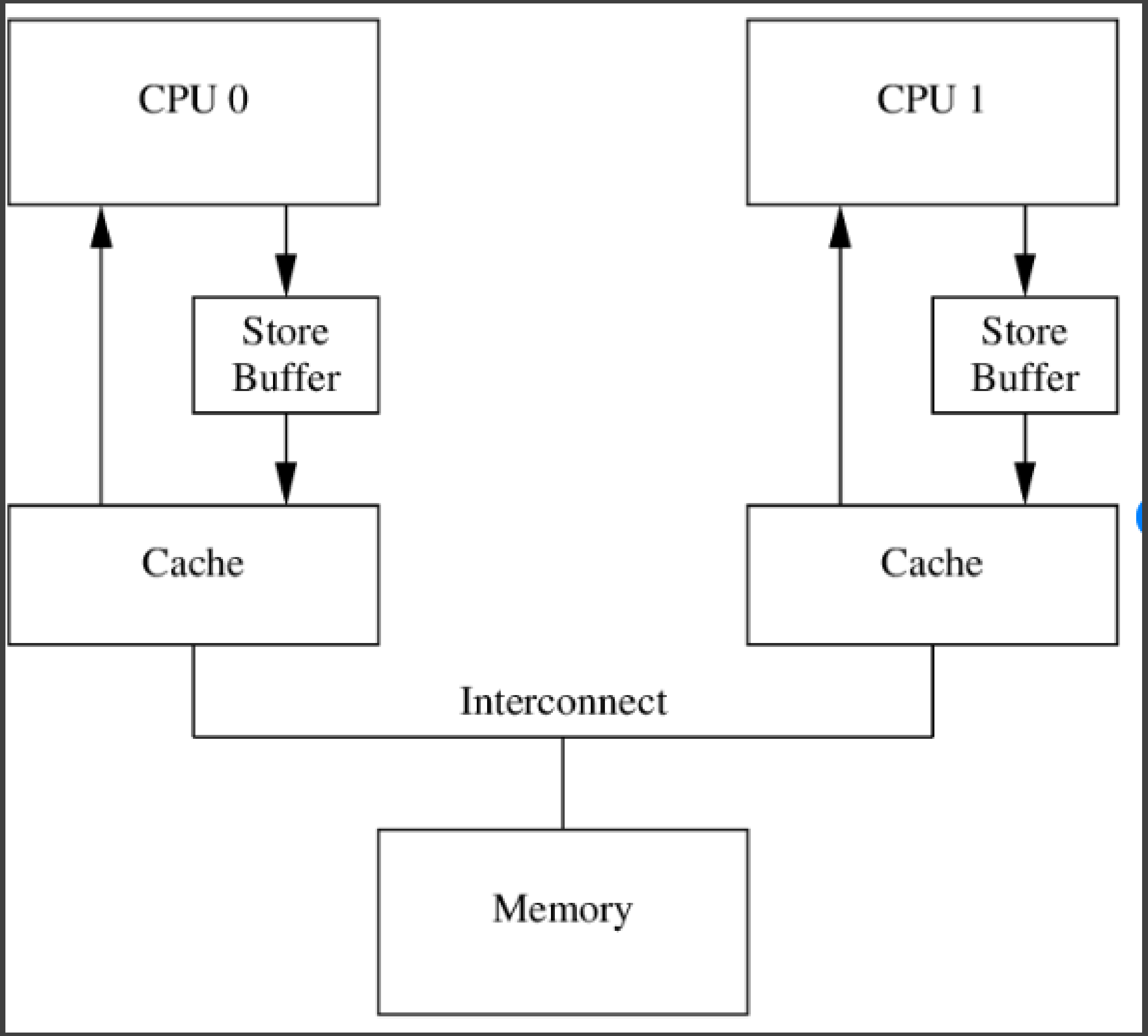
Java Memory Model

Guide to implementing lock-free data-structures

Recap from previous session “x86 model”



- X86 we just need to remember this
- (from my previous talk)
- TSO order, only store-load can get reorder by x86



Why Java memory model

1. Write once run anywhere (remember that)
2. Work not just on x86
3. Compiler optimizations
4. Compiler Re-ordering is WILD compared to hardware !

Learn by implementing a simple lock-free data structure from Linux kernel called SeqLock

What is it ?

SeqLock, Type of Readers/writer lock

- Been in kernel since 2002 nearly 20 years and works very well.
- Writer is wait-free
- Reader are lock-free (but may have to retry)

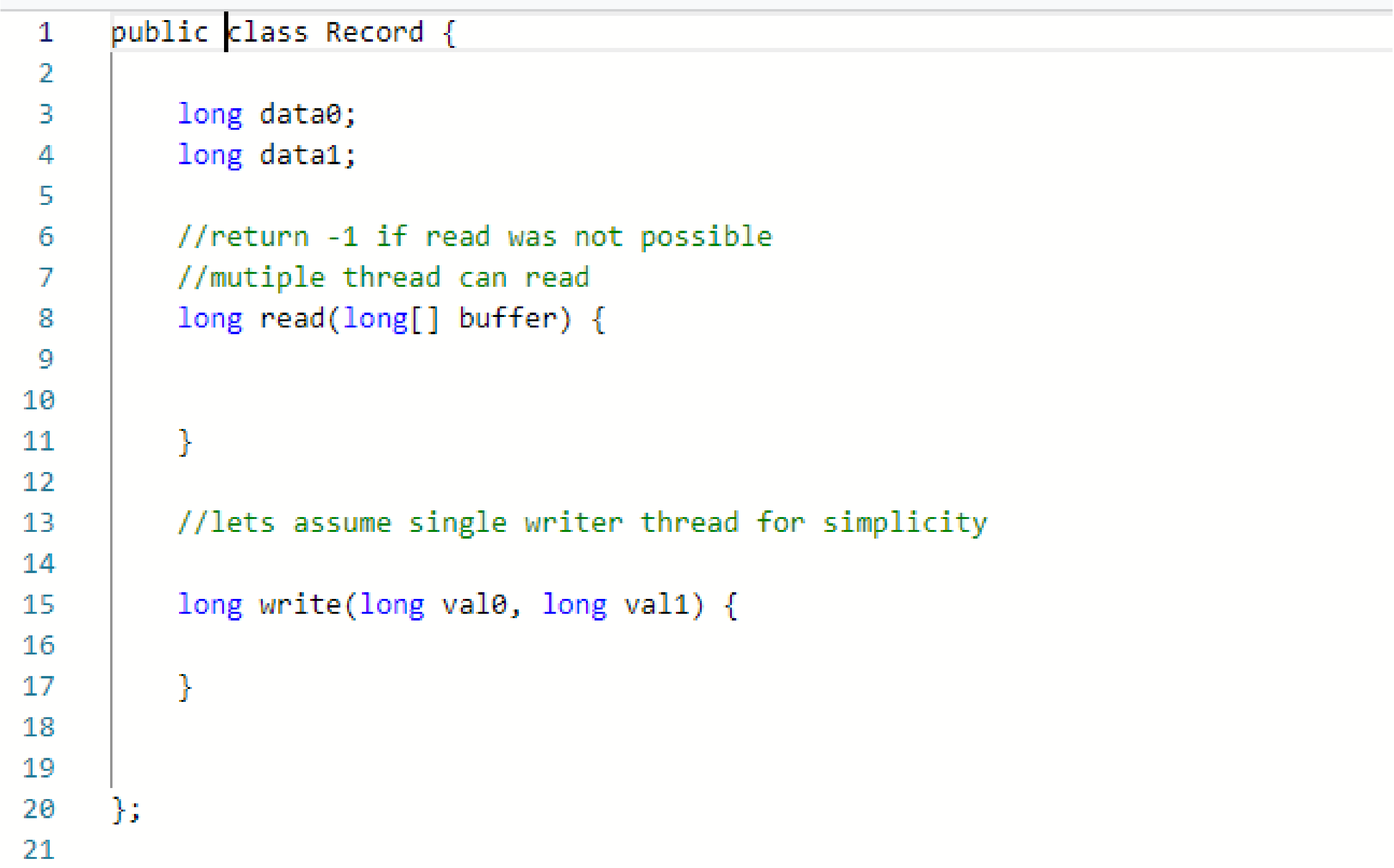
What is it

Type of Readers/writer locks

- Share some data across threads
- Single write updating the data in-place
- Multiple readers that want to read the updates
- Reader are okay to retry if writing is in progress

Type of Readers/writer locks

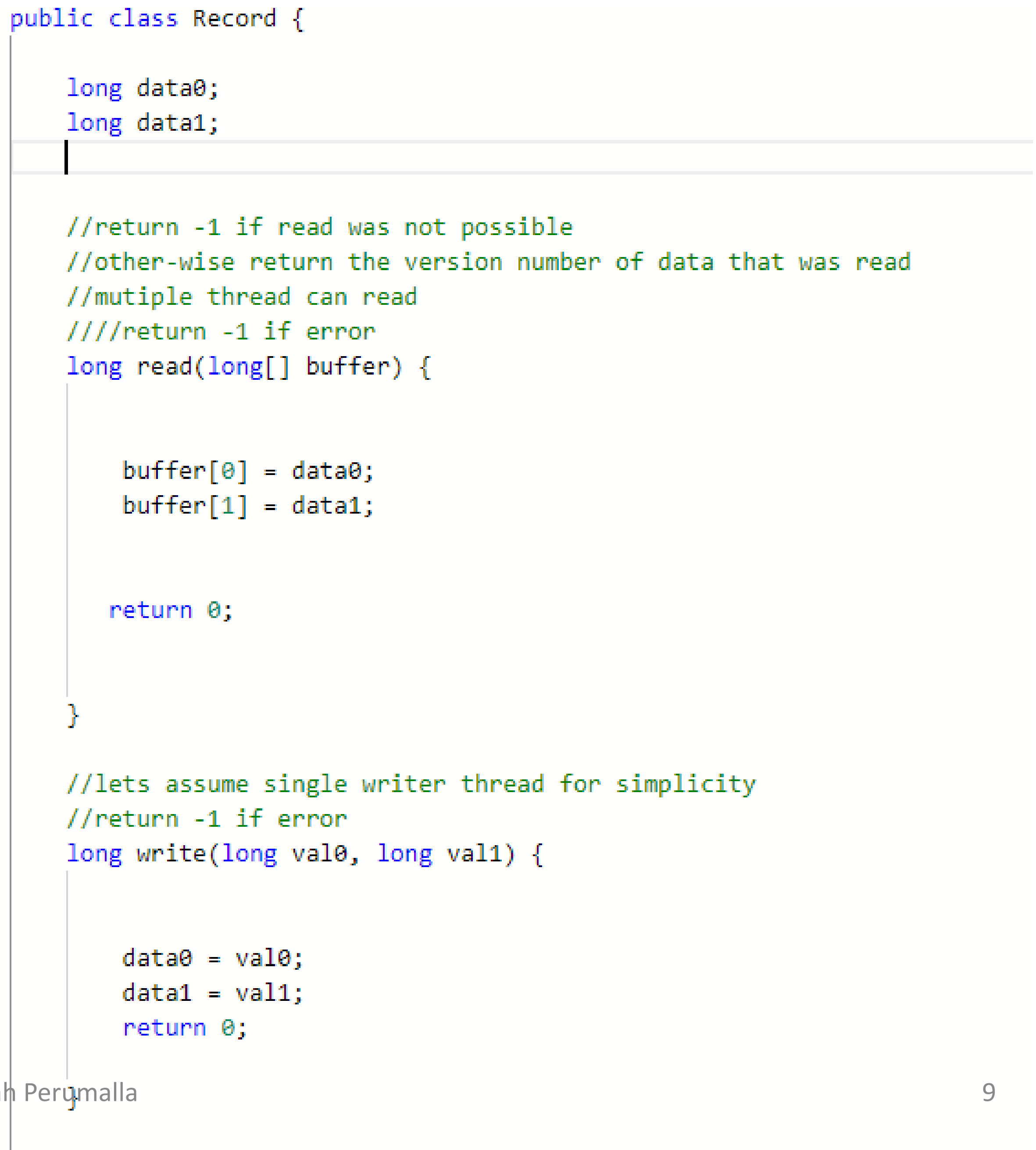
- Single thread writes/update two long values in-place
- Multiple reader want to see a consistent view of this update
- Write can never be blocked or wait



```
public class Record {

    long data0;
    long data1;

    //return -1 if read was not possible
    //mutiple thread can read
    long read(long[] buffer) {


    }

    //lets assume single writer thread for simplicity

    long write(long val0, long val1) {

    }


};

```

What could go wrong

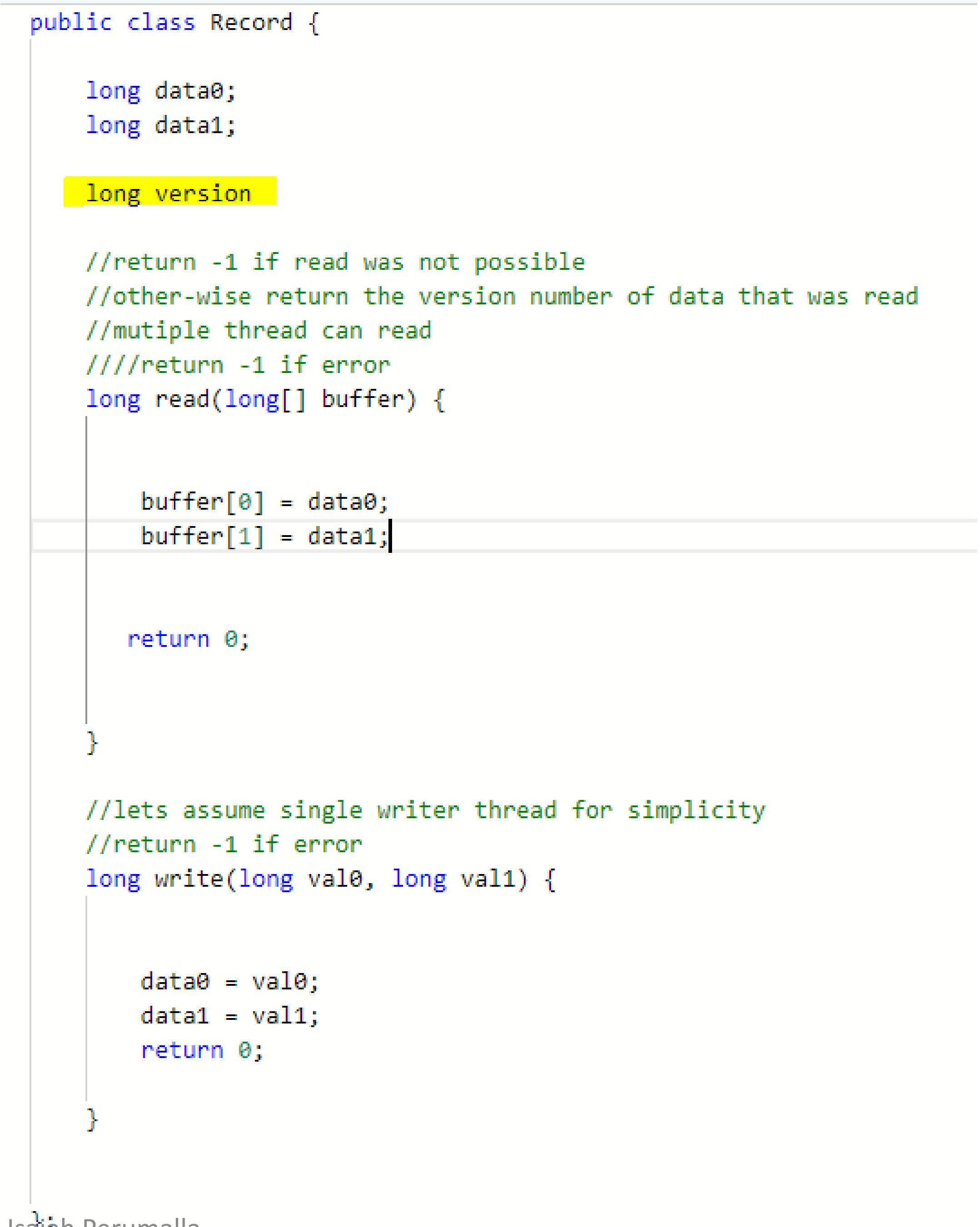
- Works fine if just one thread does both reads and writes
- Multiple threads means reader will likely see partial updates



```java
public class Record {

    long data0;
    long data1;


    //return -1 if read was not possible
    //other-wise return the version number of data that was read
    //mutiple thread can read
    ////return -1 if error
    long read(long[] buffer) {


        buffer[0] = data0;
        buffer[1] = data1;


       return 0;


    }

    //lets assume single writer thread for simplicity
    //return -1 if error
    long write(long val0, long val1) {


        data0 = val0;
        data1 = val1;
        return 0;

    }
```

Idea based of SeqLock in linux kernel

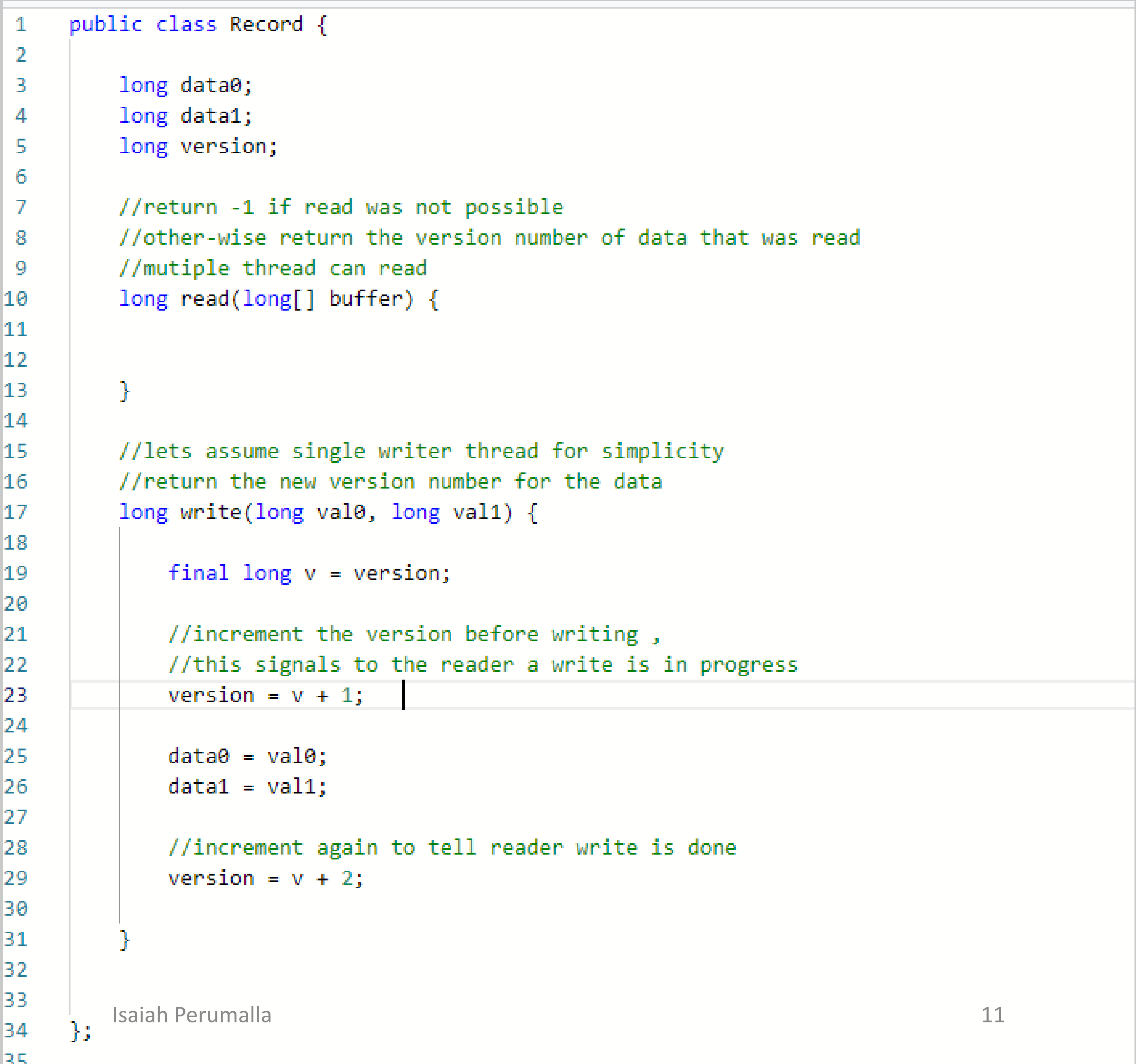
- Introduce a field version
- Think of it as version of the data
- But also used as synchronization mechanism between readers and writers



```
public class Record {

    long data0;
    long data1;

    long version

    //return -1 if read was not possible
    //other-wise return the version number of data that was read
    //mutiple thread can read
    ////return -1 if error
    long read(long[] buffer) {


        buffer[0] = data0;
        buffer[1] = data1;


       return 0;


    }

    //lets assume single writer thread for simplicity
    //return -1 if error
    long write(long val0, long val1) {


        data0 = val0;
        data1 = val1;
        return 0;

    }

};
```

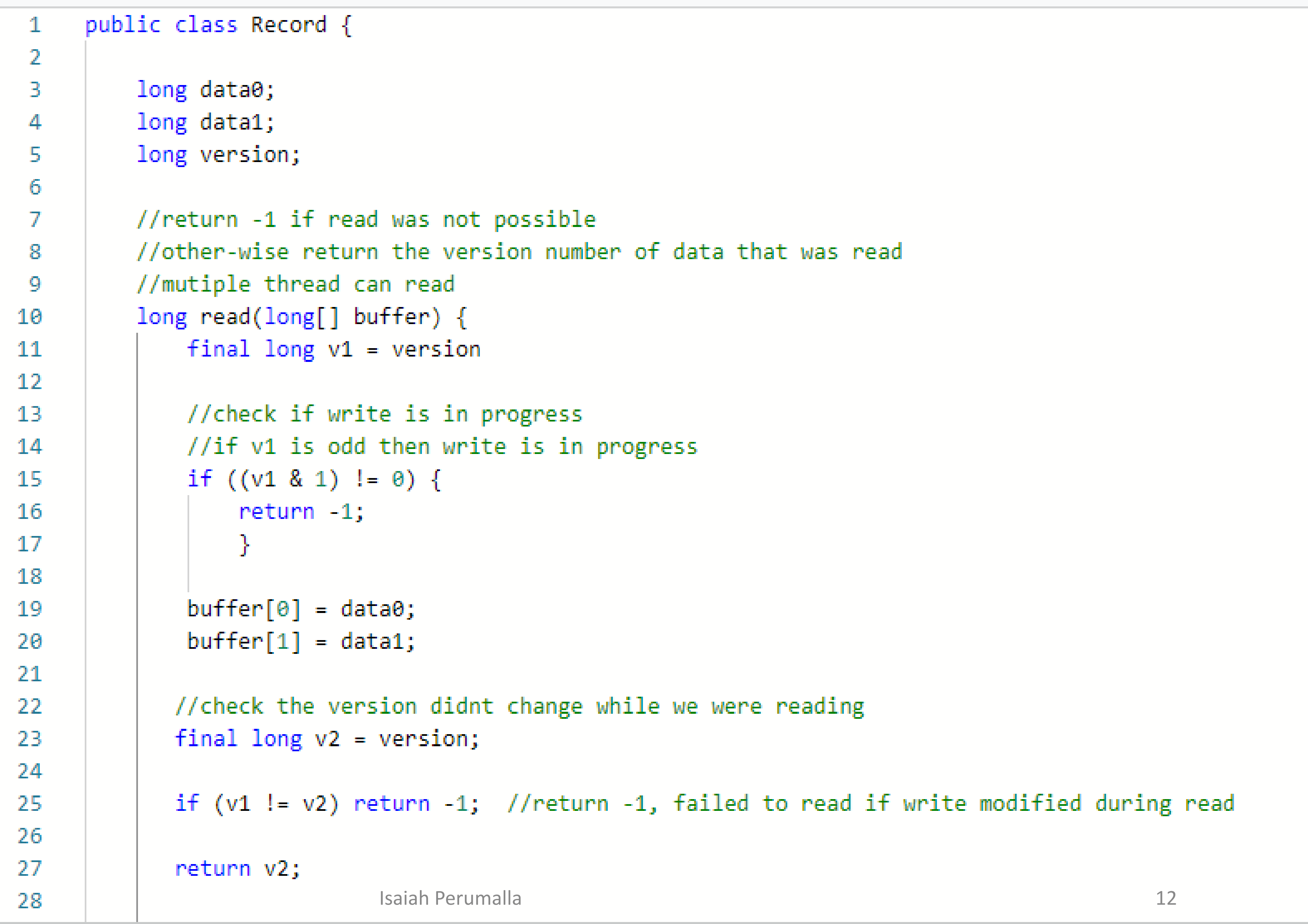
Key Idea Odd/Even version number

- *Odd* version number means write in progress
- *Even* version number means write has completed



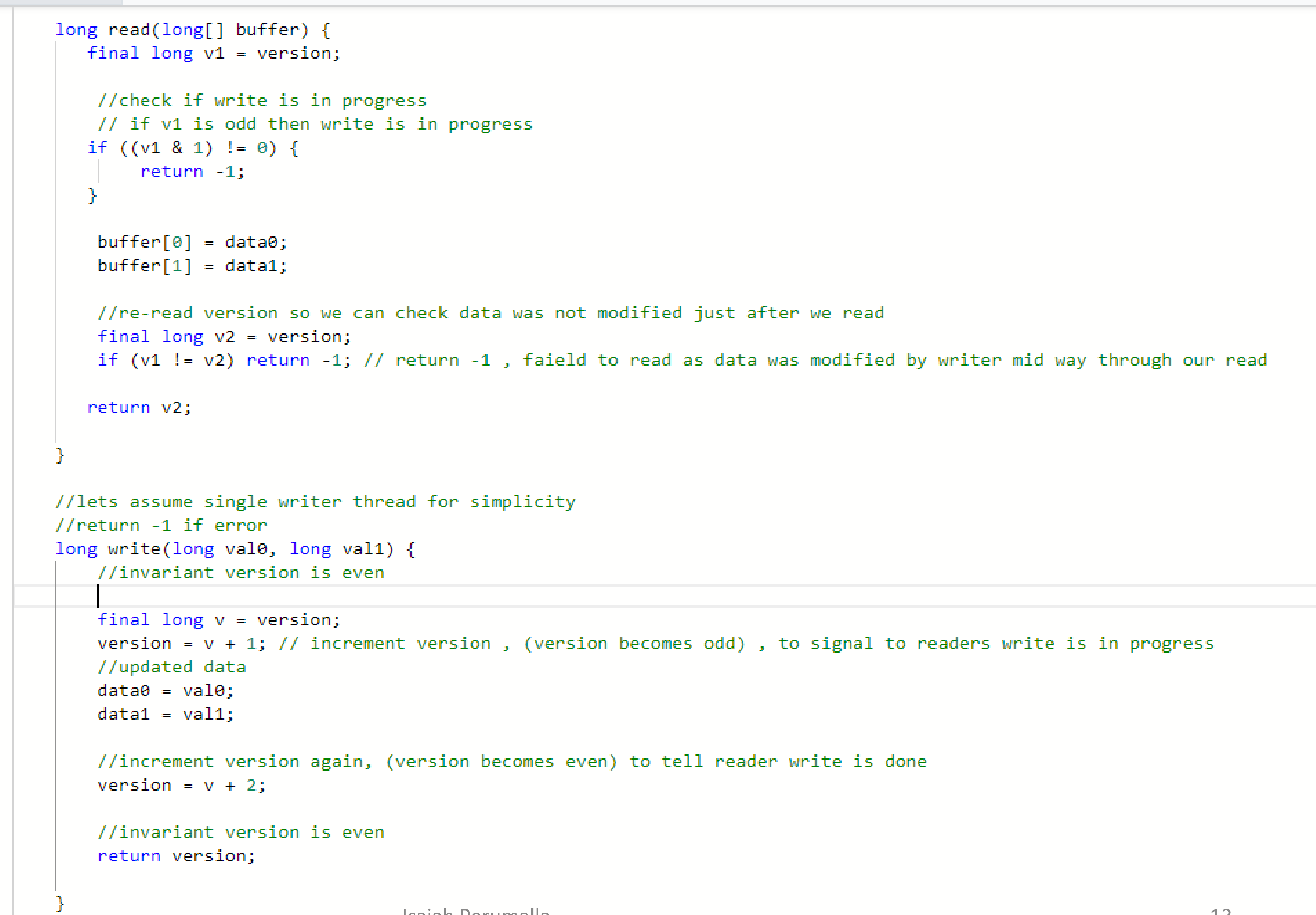
```java
public class Record {

    long data0;
    long data1;
    long version;

    //return -1 if read was not possible
    //other-wise return the version number of data that was read
    //mutiple thread can read
    long read(long[] buffer) {


    }

    //lets assume single writer thread for simplicity
    //return the new version number for the data
    long write(long val0, long val1) {

        final long v = version;

        //increment the version before writing ,
        //this signals to the reader a write is in progress
        version = v + 1;

        data0 = val0;
        data1 = val1;

        //increment again to tell reader write is done
        version = v + 2;

    }

};
```

Reader side only read even versions



```
public class Record {

    long data0;
    long data1;
    long version;

    //return -1 if read was not possible
    //other-wise return the version number of data that was read
    //mutiple thread can read
    long read(long[] buffer) {
        final long v1 = version

        //check if write is in progress
        //if v1 is odd then write is in progress
        if ((v1 & 1) != 0) {
            return -1;
            }

        buffer[0] = data0;
        buffer[1] = data1;

       //check the version didnt change while we were reading
       final long v2 = version;

       if (v1 != v2) return -1;  //return -1, failed to read if write modified during read

       return v2;
```

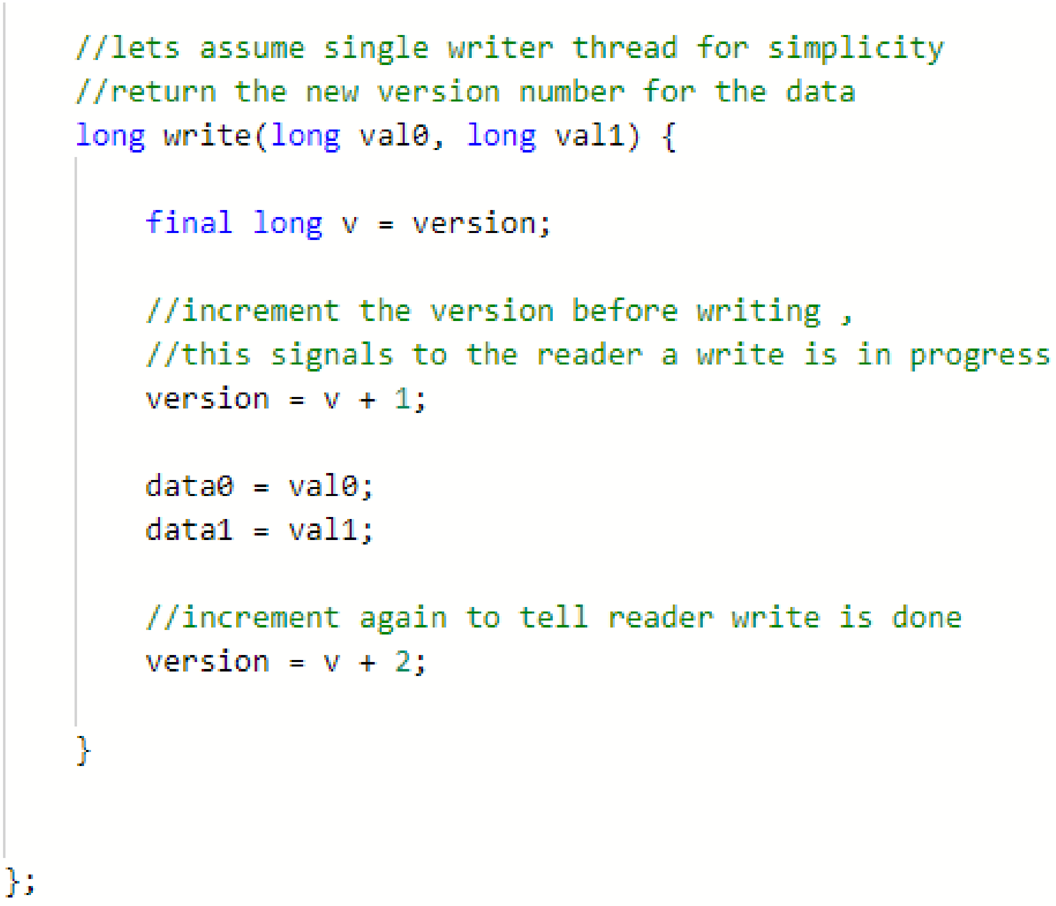
What could possibly go wrong !



```
long read(long[] buffer) {
   final long v1 = version;

    //check if write is in progress
    // if v1 is odd then write is in progress
   if ((v1 & 1) != 0) {
        return -1;
   }

    buffer[0] = data0;
    buffer[1] = data1;

    //re-read version so we can check data was not modified just after we read
    final long v2 = version;
    if (v1 != v2) return -1; // return -1 , faield to read as data was modified by writer mid way through our read

   return v2;

}

//lets assume single writer thread for simplicity
//return -1 if error
long write(long val0, long val1) {
    //invariant version is even

    final long v = version;
    version = v + 1; // increment version , (version becomes odd) , to signal to readers write is in progress
    //updated data
    data0 = val0;
    data1 = val1;

    //increment version again, (version becomes even) to tell reader write is done
    version = v + 2;

    //invariant version is even
    return version;

}
```

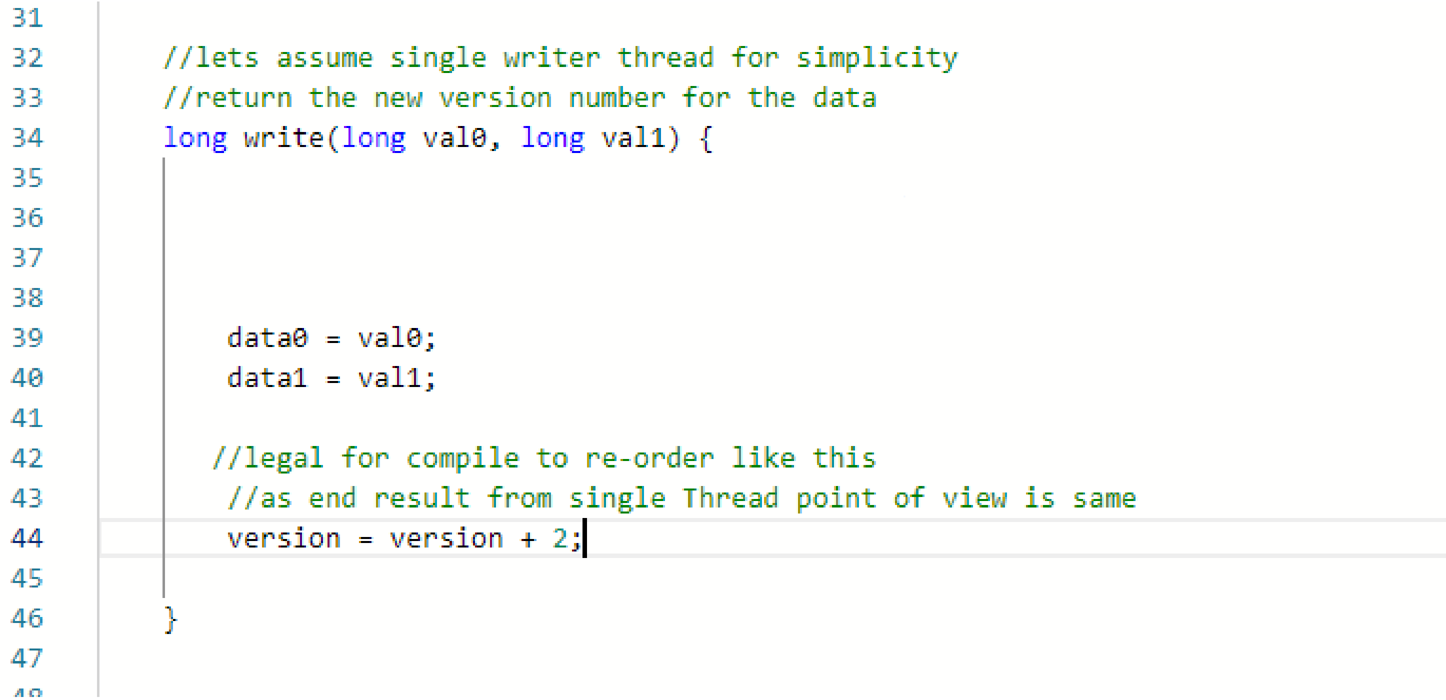
Optimising Compilers are Wild !

- Code you write is not what gets executed
- Let's look at the write() method



```
    //lets assume single writer thread for simplicity
    //return the new version number for the data
    long write(long val0, long val1) {

        final long v = version;

        //increment the version before writing ,
        //this signals to the reader a write is in progress
        version = v + 1;

        data0 = val0;
        data1 = val1;

        //increment again to tell reader write is done
        version = v + 2;

    }

};
```

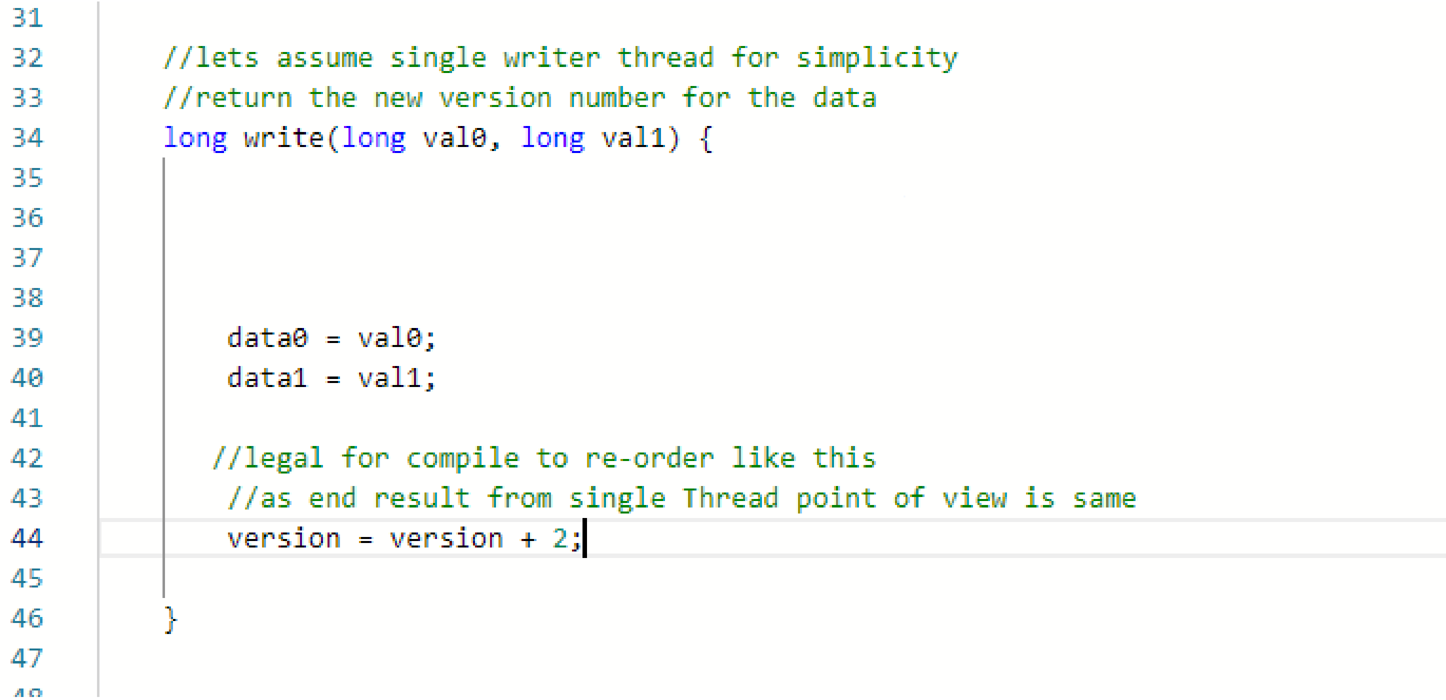
Compilers are Wild !



```
    //lets assume single writer thread for simplicity
    //return the new version number for the data
    long write(long val0, long val1) {




        data0 = val0;
        data1 = val1;

       //legal for compile to re-order like this
        //as end result from single Thread point of view is same
        version = version + 2;

    }

```

- Can be re-ordered like this
- Would totally break our invariant for out data-structure

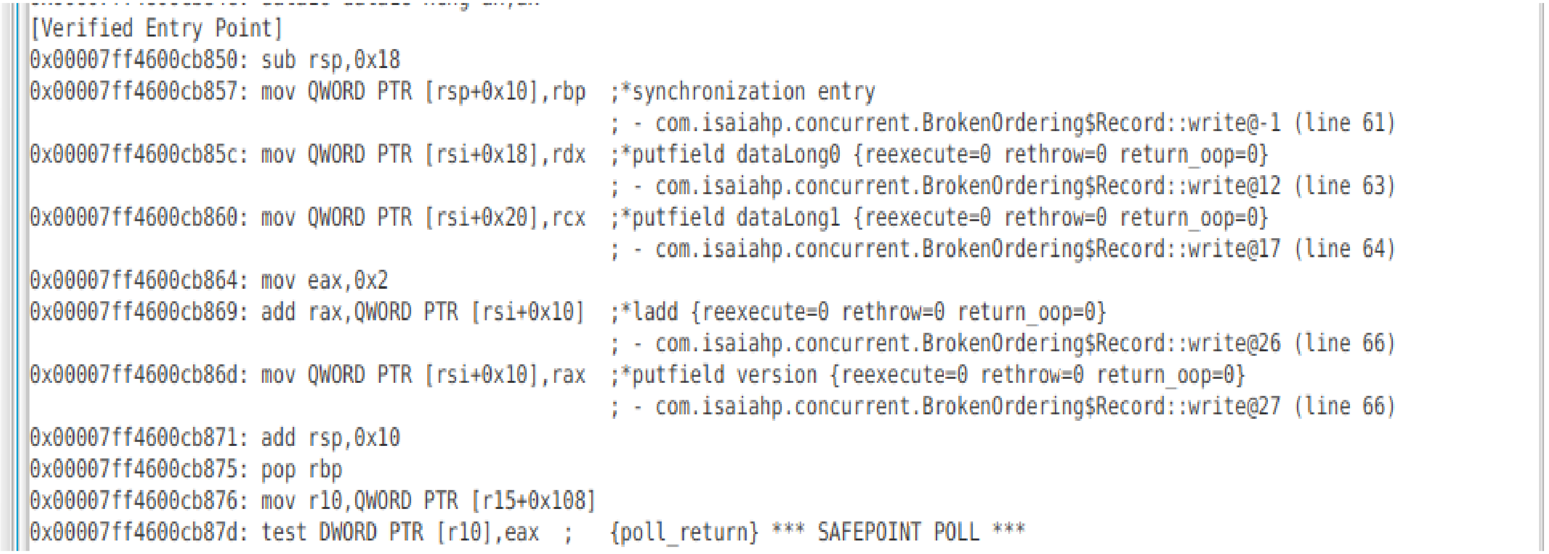
How do we know this can happen ?



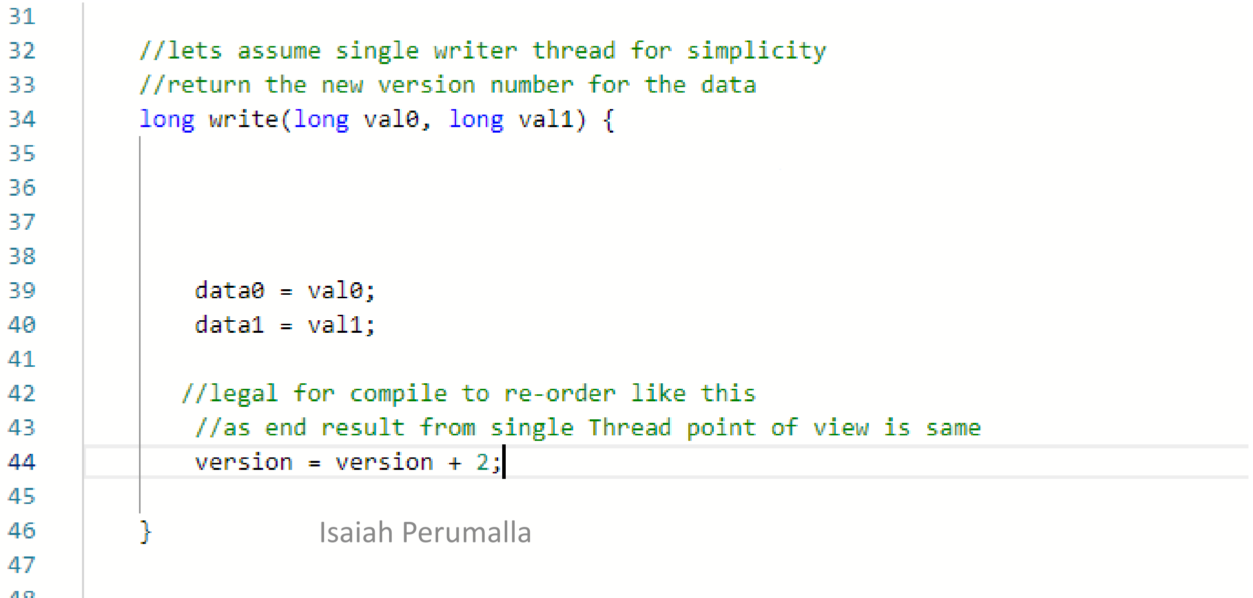
```

//lets assume single writer thread for simplicity
//return the new version number for the data
long write(long val0, long val1) {




    data0 = val0;
    data1 = val1;

   //legal for compile to re-order like this
    //as end result from single Thread point of view is same
    version = version + 2;

}

```

- Can be re-ordered like this
- Would totally break our invariant for out data-structure

Re-order in Action C2 Compiler Assembly Proves this

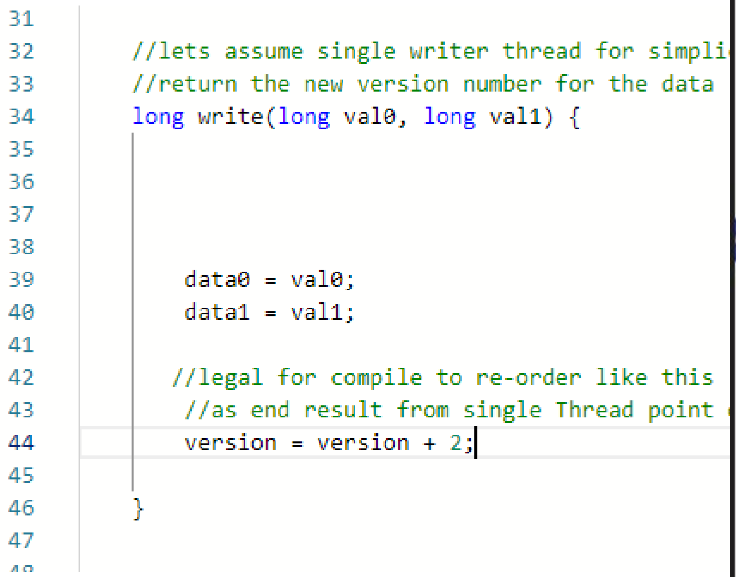


```
[Verified Entry Point]
0x00007ff4600cb850: sub rsp,0x18
0x00007ff4600cb857: mov QWORD PTR [rsp+0x10],rbp  ;*synchronization entry
                                                  ; - com.isaiahp.concurrent.BrokenOrdering$Record::write@-1 (line 61)
0x00007ff4600cb85c: mov QWORD PTR [rsi+0x18],rdx  ;*putfield dataLong0 {reexecute=0 rethrow=0 return_oop=0}
                                                  ; - com.isaiahp.concurrent.BrokenOrdering$Record::write@12 (line 63)
0x00007ff4600cb860: mov QWORD PTR [rsi+0x20],rcx  ;*putfield dataLong1 {reexecute=0 rethrow=0 return_oop=0}
                                                  ; - com.isaiahp.concurrent.BrokenOrdering$Record::write@17 (line 64)
0x00007ff4600cb864: mov eax,0x2
0x00007ff4600cb869: add rax,QWORD PTR [rsi+0x10]  ;*ladd {reexecute=0 rethrow=0 return_oop=0}
                                                  ; - com.isaiahp.concurrent.BrokenOrdering$Record::write@26 (line 66)
0x00007ff4600cb86d: mov QWORD PTR [rsi+0x10],rax  ;*putfield version {reexecute=0 rethrow=0 return_oop=0}
                                                  ; - com.isaiahp.concurrent.BrokenOrdering$Record::write@27 (line 66)
0x00007ff4600cb871: add rsp,0x10
0x00007ff4600cb875: pop rbp
0x00007ff4600cb876: mov r10,QWORD PTR [r15+0x108]
0x00007ff4600cb87d: test DWORD PTR [r10],eax  ;   {poll_return} *** SAFEPOINT POLL ***
```

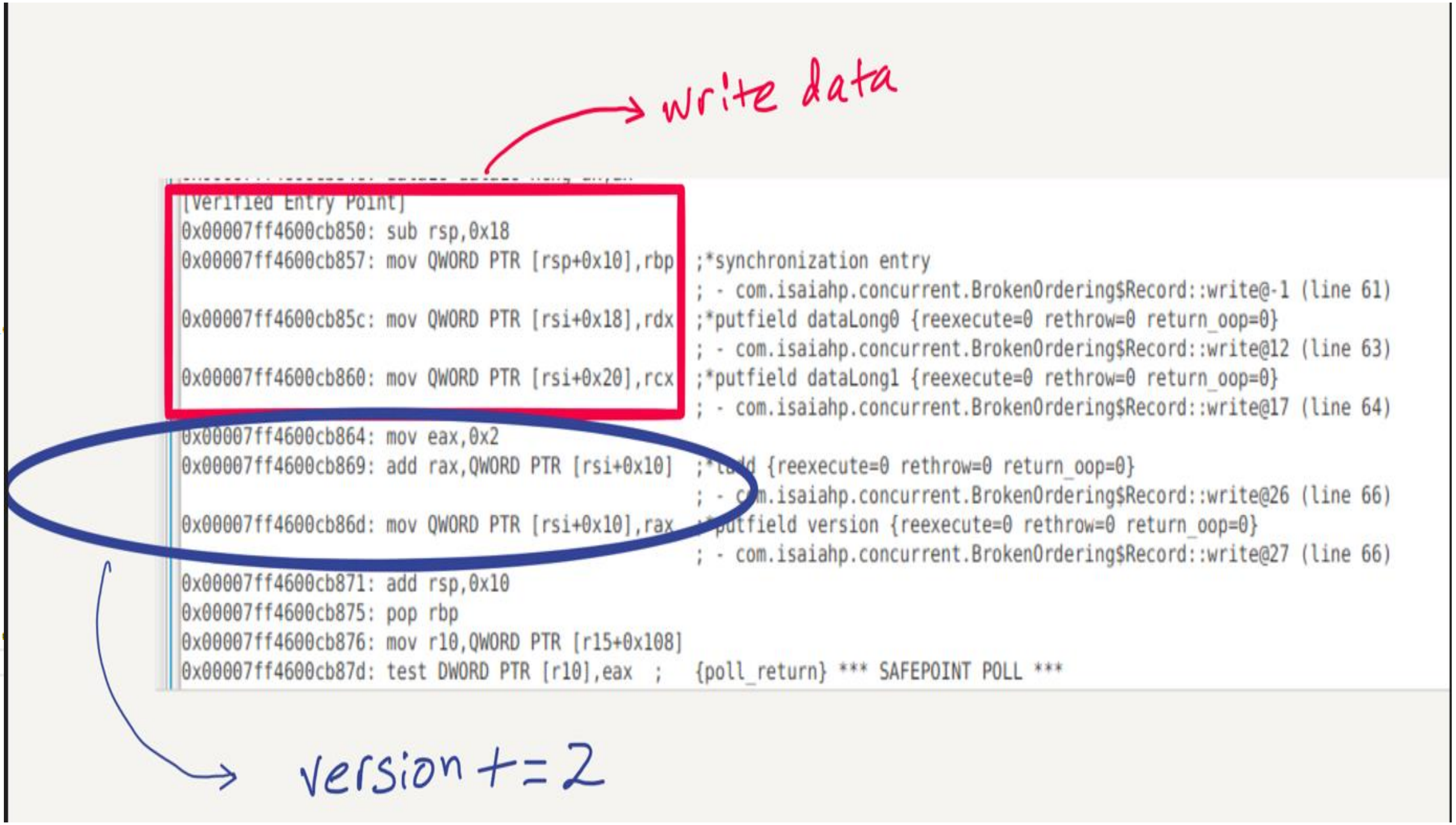


```java

    //lets assume single writer thread for simplicity
    //return the new version number for the data
    long write(long val0, long val1) {




        data0 = val0;
        data1 = val1;

      //legal for compile to re-order like this
        //as end result from single Thread point of view is same
        version = version + 2;

    }

```

Re-order in Action
C2 Compiler
Assembly
Proves this

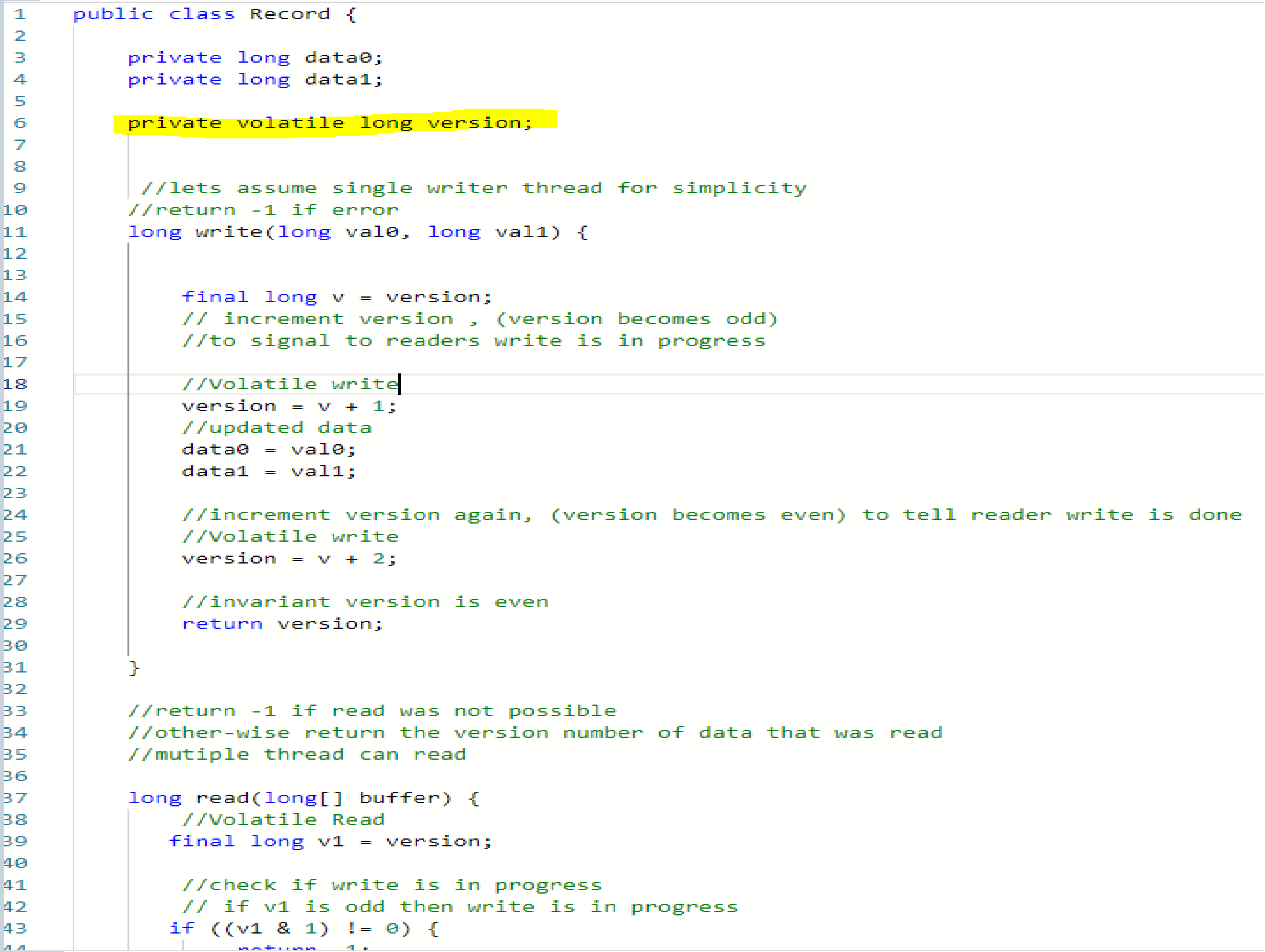


```
    //lets assume single writer thread for simpli
    //return the new version number for the data
    long write(long val0, long val1) {



        data0 = val0;
        data1 = val1;

       //legal for compile to re-order like this
        //as end result from single Thread point
        version = version + 2;

    }

```

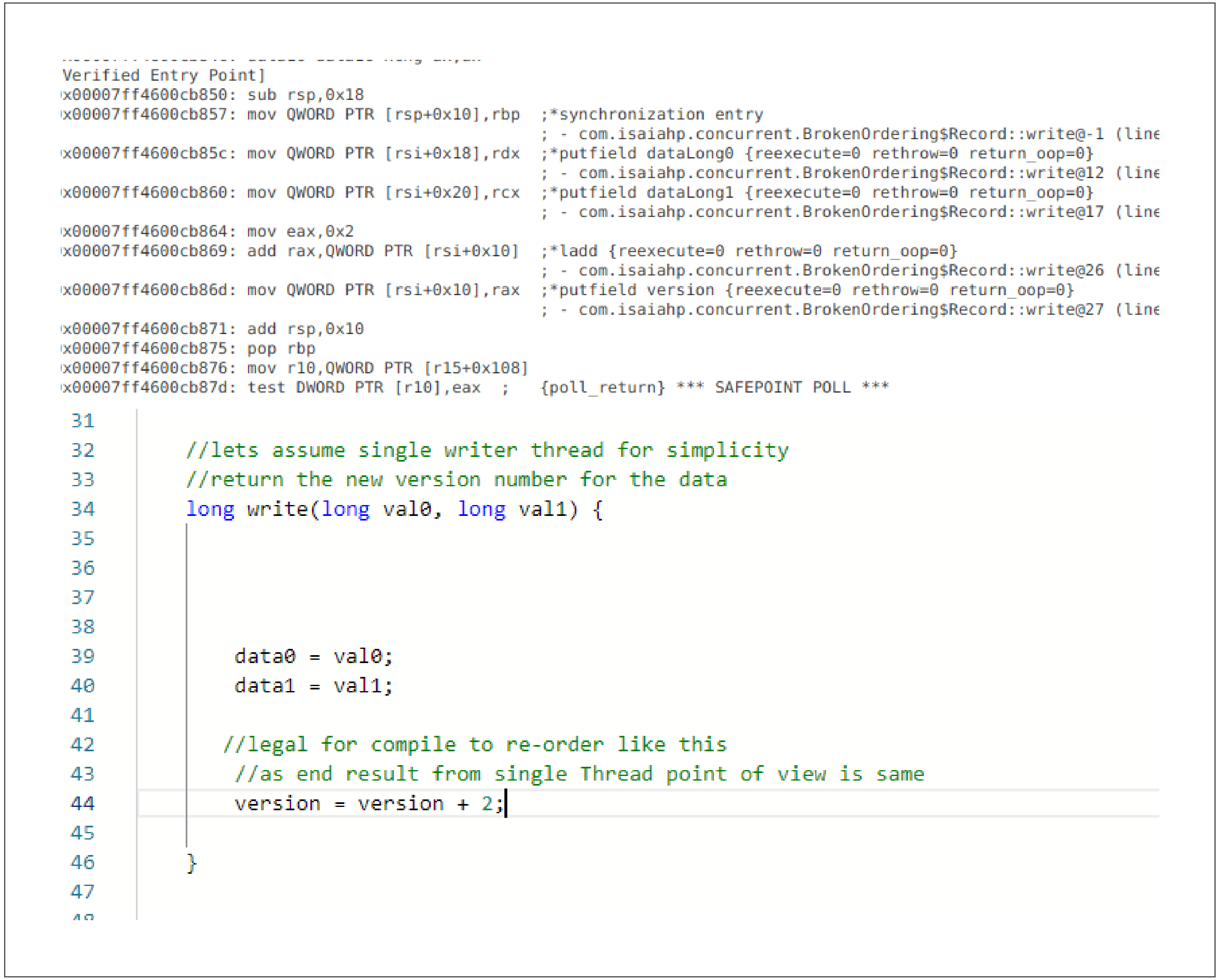


```
[Verified Entry Point]
0x00007ff4600cb850: sub rsp,0x18
0x00007ff4600cb857: mov QWORD PTR [rsp+0x10],rbp  ;*synchronization entry
                                                  ; - com.isaiahp.concurrent.BrokenOrdering$Record::write@-1 (line 61)
0x00007ff4600cb85c: mov QWORD PTR [rsi+0x18],rdx  ;*putfield dataLong0 {reexecute=0 rethrow=0 return_oop=0}
                                                  ; - com.isaiahp.concurrent.BrokenOrdering$Record::write@12 (line 63)
0x00007ff4600cb860: mov QWORD PTR [rsi+0x20],rcx  ;*putfield dataLong1 {reexecute=0 rethrow=0 return_oop=0}
                                                  ; - com.isaiahp.concurrent.BrokenOrdering$Record::write@17 (line 64)
0x00007ff4600cb864: mov eax,0x2
0x00007ff4600cb869: add rax,QWORD PTR [rsi+0x10]  ;*ladd {reexecute=0 rethrow=0 return_oop=0}
                                                  ; - com.isaiahp.concurrent.BrokenOrdering$Record::write@26 (line 66)
0x00007ff4600cb86d: mov QWORD PTR [rsi+0x10],rax  ;*putfield version {reexecute=0 rethrow=0 return_oop=0}
                                                  ; - com.isaiahp.concurrent.BrokenOrdering$Record::write@27 (line 66)
0x00007ff4600cb871: add rsp,0x10
0x00007ff4600cb875: pop rbp
0x00007ff4600cb876: mov r10,QWORD PTR [r15+0x108]
0x00007ff4600cb87d: test DWORD PTR [r10],eax  ;   {poll_return} *** SAFEPOINT POLL ***
```

Let use *VOLATILES* What could possibly go wrong !



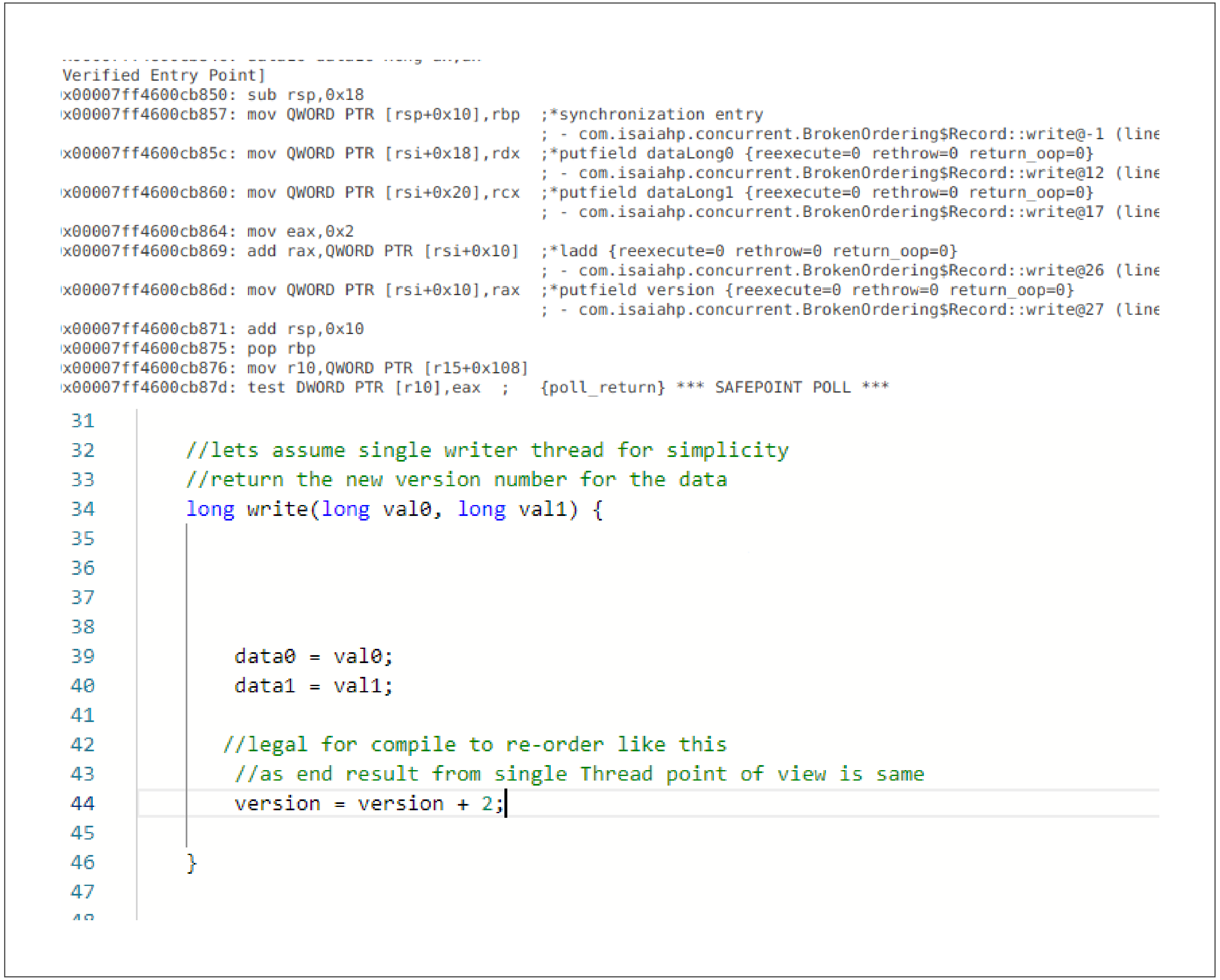
```java
public class Record {

    private long data0;
    private long data1;

    private volatile long version;


     //lets assume single writer thread for simplicity
    //return -1 if error
    long write(long val0, long val1) {


        final long v = version;
        // increment version , (version becomes odd)
        //to signal to readers write is in progress

        //Volatile write
        version = v + 1;
        //updated data
        data0 = val0;
        data1 = val1;

        //increment version again, (version becomes even) to tell reader write is done
        //Volatile write
        version = v + 2;

        //invariant version is even
        return version;

    }

    //return -1 if read was not possible
    //other-wise return the version number of data that was read
    //mutiple thread can read

    long read(long[] buffer) {
        //Volatile Read
       final long v1 = version;

        //check if write is in progress
        // if v1 is odd then write is in progress
       if ((v1 & 1) != 0) {
           return -1;
```



```
Verified Entry Point]
x00007ff4600cb850: sub rsp,0x18
x00007ff4600cb857: mov QWORD PTR [rsp+0x10],rbp  ;*synchronization entry
                                                 ; - com.isaiahp.concurrent.BrokenOrdering$Record::write@-1 (line
x00007ff4600cb85c: mov QWORD PTR [rsi+0x18],rdx  ;*putfield dataLong0 {reexecute=0 rethrow=0 return_oop=0}
                                                 ; - com.isaiahp.concurrent.BrokenOrdering$Record::write@12 (line
x00007ff4600cb860: mov QWORD PTR [rsi+0x20],rcx  ;*putfield dataLong1 {reexecute=0 rethrow=0 return_oop=0}
                                                 ; - com.isaiahp.concurrent.BrokenOrdering$Record::write@17 (line
x00007ff4600cb864: mov eax,0x2
x00007ff4600cb869: add rax,QWORD PTR [rsi+0x10]  ;*ladd {reexecute=0 rethrow=0 return_oop=0}
                                                 ; - com.isaiahp.concurrent.BrokenOrdering$Record::write@26 (line
x00007ff4600cb86d: mov QWORD PTR [rsi+0x10],rax  ;*putfield version {reexecute=0 rethrow=0 return_oop=0}
                                                 ; - com.isaiahp.concurrent.BrokenOrdering$Record::write@27 (line
x00007ff4600cb871: add rsp,0x10
x00007ff4600cb875: pop rbp
x00007ff4600cb876: mov r10,QWORD PTR [r15+0x108]
x00007ff4600cb87d: test DWORD PTR [r10],eax  ;   {poll_return} *** SAFEPOINT POLL ***
```

```java

    //lets assume single writer thread for simplicity
    //return the new version number for the data
    long write(long val0, long val1) {




        data0 = val0;
        data1 = val1;

       //legal for compile to re-order like this
        //as end result from single Thread point of view is same
        version = version + 2;

    }

```

JIT compiles at runtime and assembly code is not static can be re-generated.

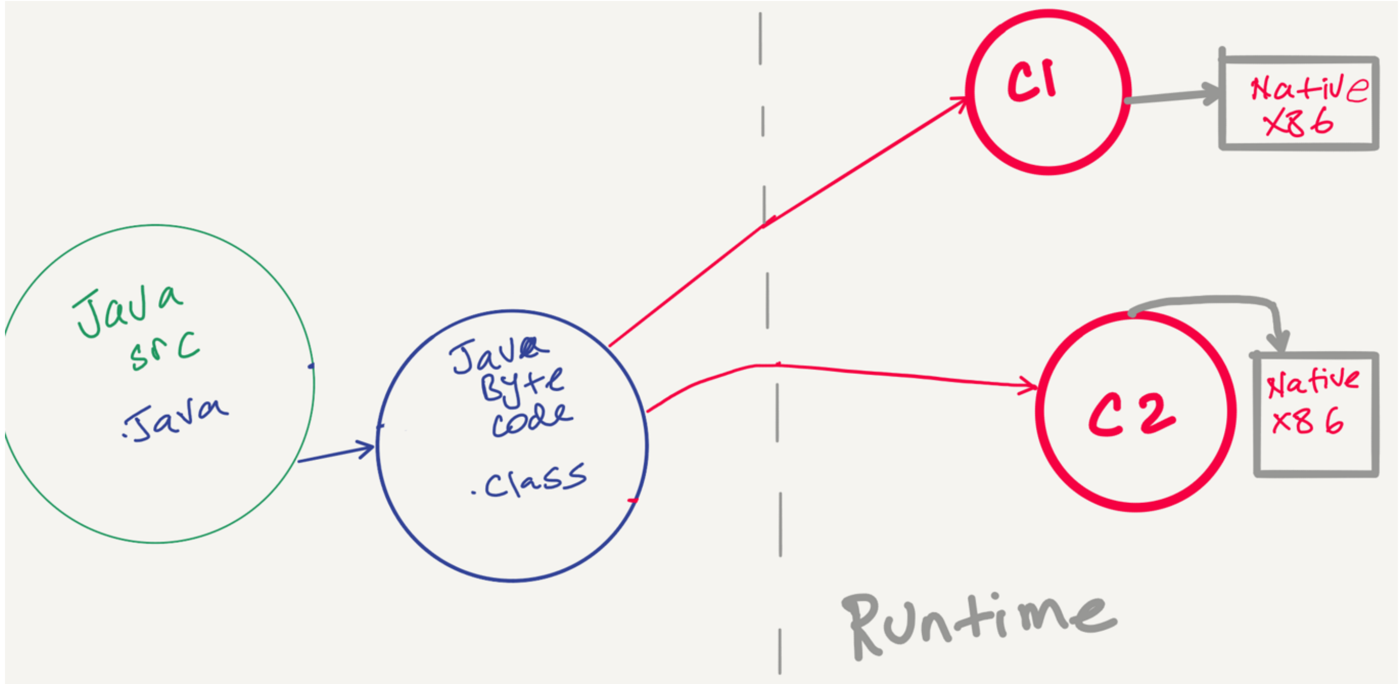


```
Verified Entry Point]
x00007ff4600cb850: sub rsp,0x18
x00007ff4600cb857: mov QWORD PTR [rsp+0x10],rbp  ;*synchronization entry
                                                 ; - com.isaiahp.concurrent.BrokenOrdering$Record::write@-1 (line
x00007ff4600cb85c: mov QWORD PTR [rsi+0x18],rdx  ;*putfield dataLong0 {reexecute=0 rethrow=0 return_oop=0}
                                                 ; - com.isaiahp.concurrent.BrokenOrdering$Record::write@12 (line
x00007ff4600cb860: mov QWORD PTR [rsi+0x20],rcx  ;*putfield dataLong1 {reexecute=0 rethrow=0 return_oop=0}
                                                 ; - com.isaiahp.concurrent.BrokenOrdering$Record::write@17 (line
x00007ff4600cb864: mov eax,0x2
x00007ff4600cb869: add rax,QWORD PTR [rsi+0x10]  ;*ladd {reexecute=0 rethrow=0 return_oop=0}
                                                 ; - com.isaiahp.concurrent.BrokenOrdering$Record::write@26 (line
x00007ff4600cb86d: mov QWORD PTR [rsi+0x10],rax  ;*putfield version {reexecute=0 rethrow=0 return_oop=0}
                                                 ; - com.isaiahp.concurrent.BrokenOrdering$Record::write@27 (line
x00007ff4600cb871: add rsp,0x10
x00007ff4600cb875: pop rbp
x00007ff4600cb876: mov r10,QWORD PTR [r15+0x108]
x00007ff4600cb87d: test DWORD PTR [r10],eax  ;   {poll_return} *** SAFEPOINT POLL ***
```

```

    //lets assume single writer thread for simplicity
    //return the new version number for the data
    long write(long val0, long val1) {




        data0 = val0;
        data1 = val1;

       //legal for compile to re-order like this
        //as end result from single Thread point of view is same
        version = version + 2;

    }

```

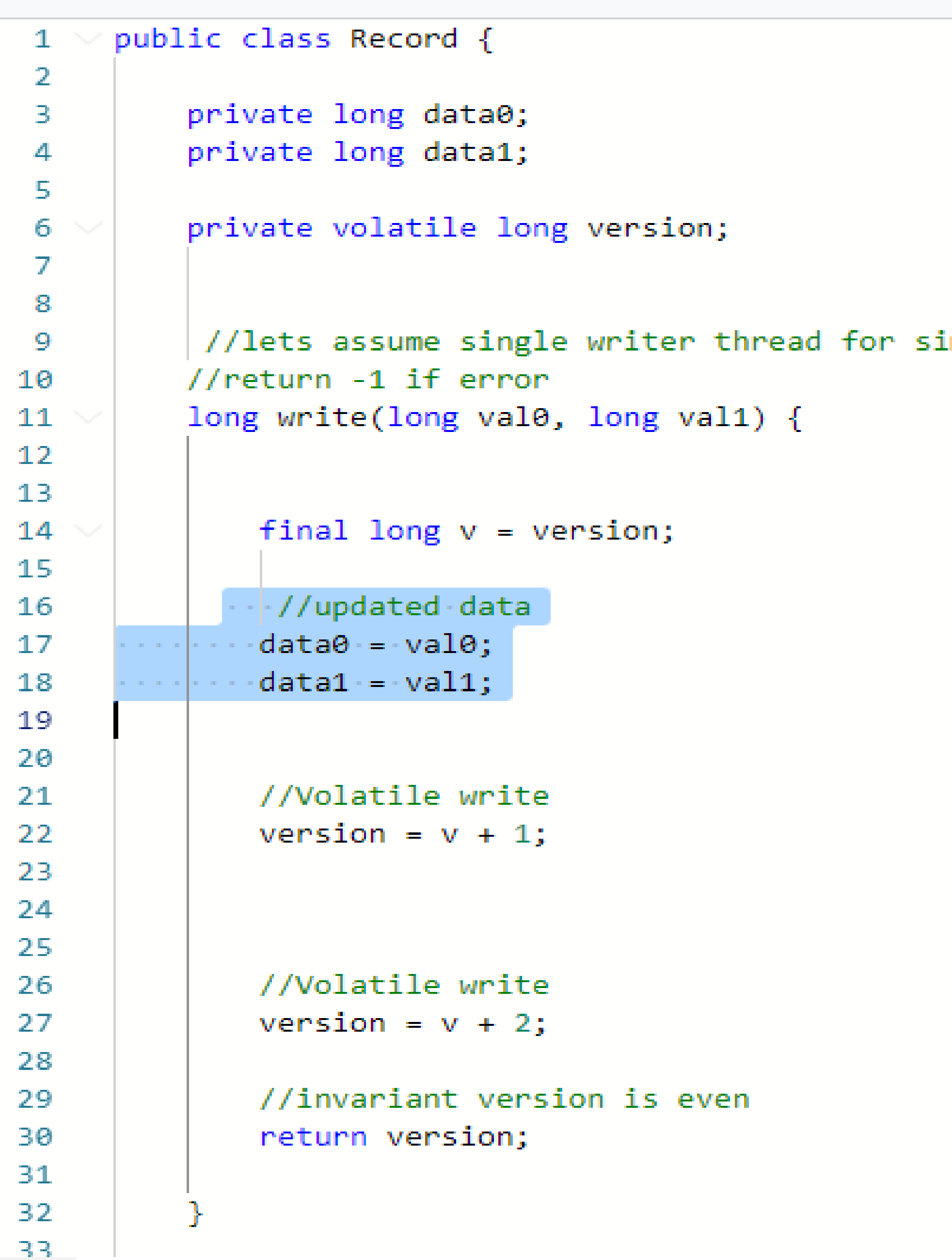
JIT compiles at runtime and assembly code is not static can be re-generated.

JIT can generate native code many times using different compilers for different methods !



Compilers are Wild ! A new re-order possible !

- Can be re-ordered like this and legal event with volatile version !
- Again Would totally break our invariant for out data-structure
- ***I have Jcstress Test case to prove this can and will happen time to time***



```java
public class Record {

    private long data0;
    private long data1;

    private volatile long version;


    //lets assume single writer thread for simplicity
    //return -1 if error
    long write(long val0, long val1) {


        final long v = version;

        //updated data
        data0 = val0;
        data1 = val1;


        //Volatile write
        version = v + 1;



        //Volatile write
        version = v + 2;

        //invariant version is even
        return version;

    }
```

Program language Memory Model

- Java was the first mainstream language to try to produce one
- More recently C++ and Rust have improved on this and is lot more precise
- Java 11 onwards is moving towards similar model to C++

Program language Memory Model

- It a contract between programmer , compiler and hardware
- Allows programmer to tell compiler not to re-order in some situations
- Big topic but just need to know enough

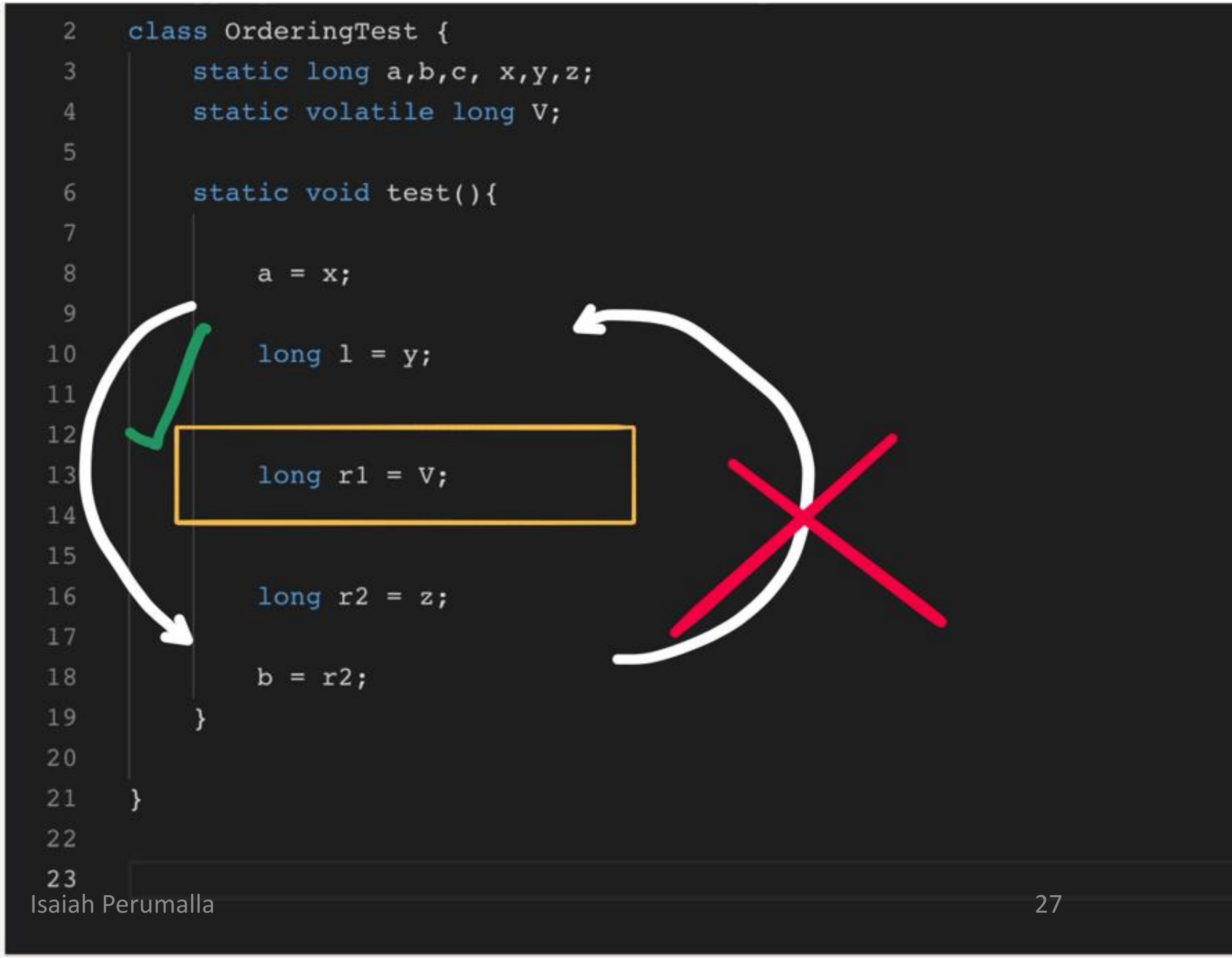
Volatile Reads

- What is Allowed

Volatile Reads

- Load and Store that come after the volatile read, ***cannot*** move before a Volatile Read

- However prior loads & stores ***can*** move after it !

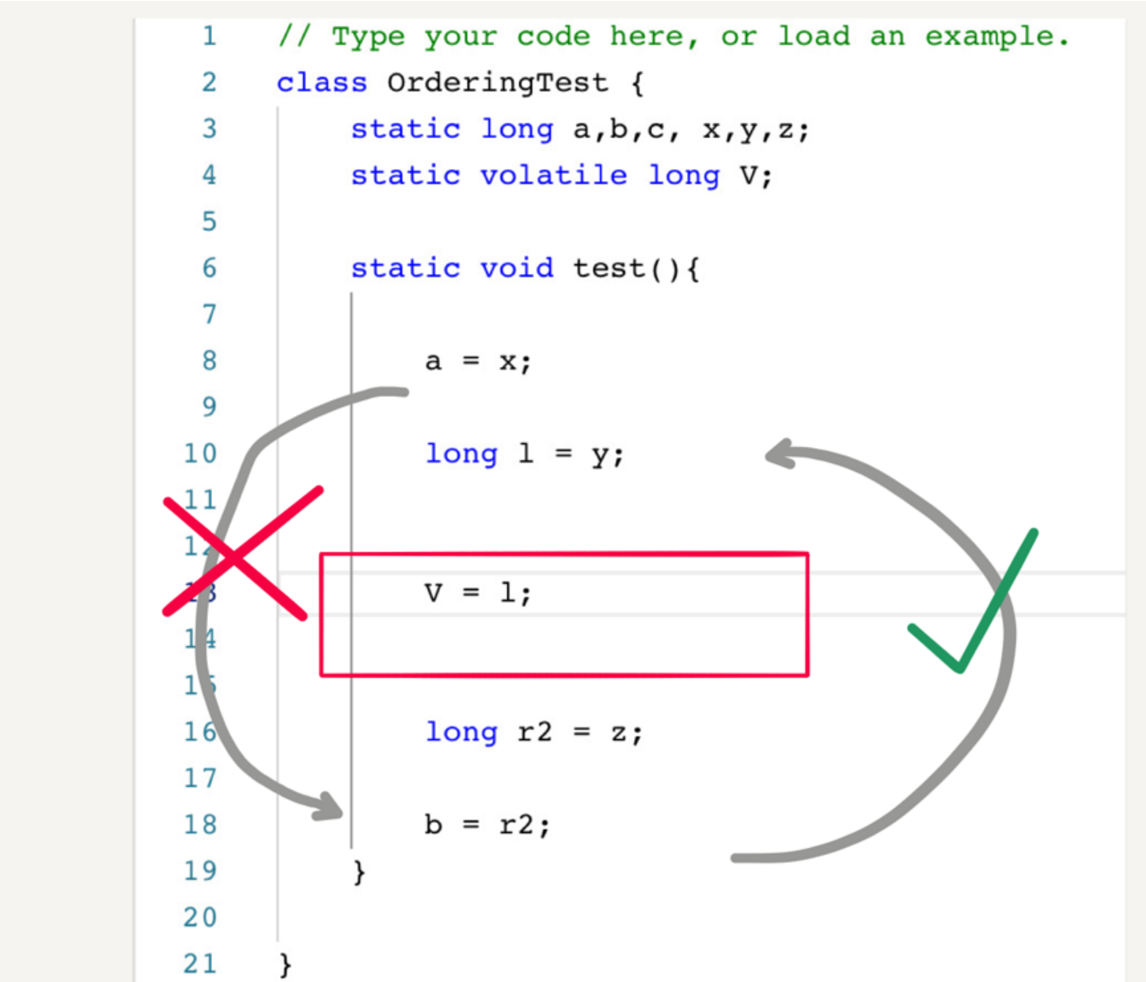


```
class OrderingTest {
    static long a,b,c, x,y,z;
    static volatile long V;

    static void test(){

        a = x;

        long l = y;

        long r1 = V;

        long r2 = z;

        b = r2;
    }

}
```

Volatile Writes

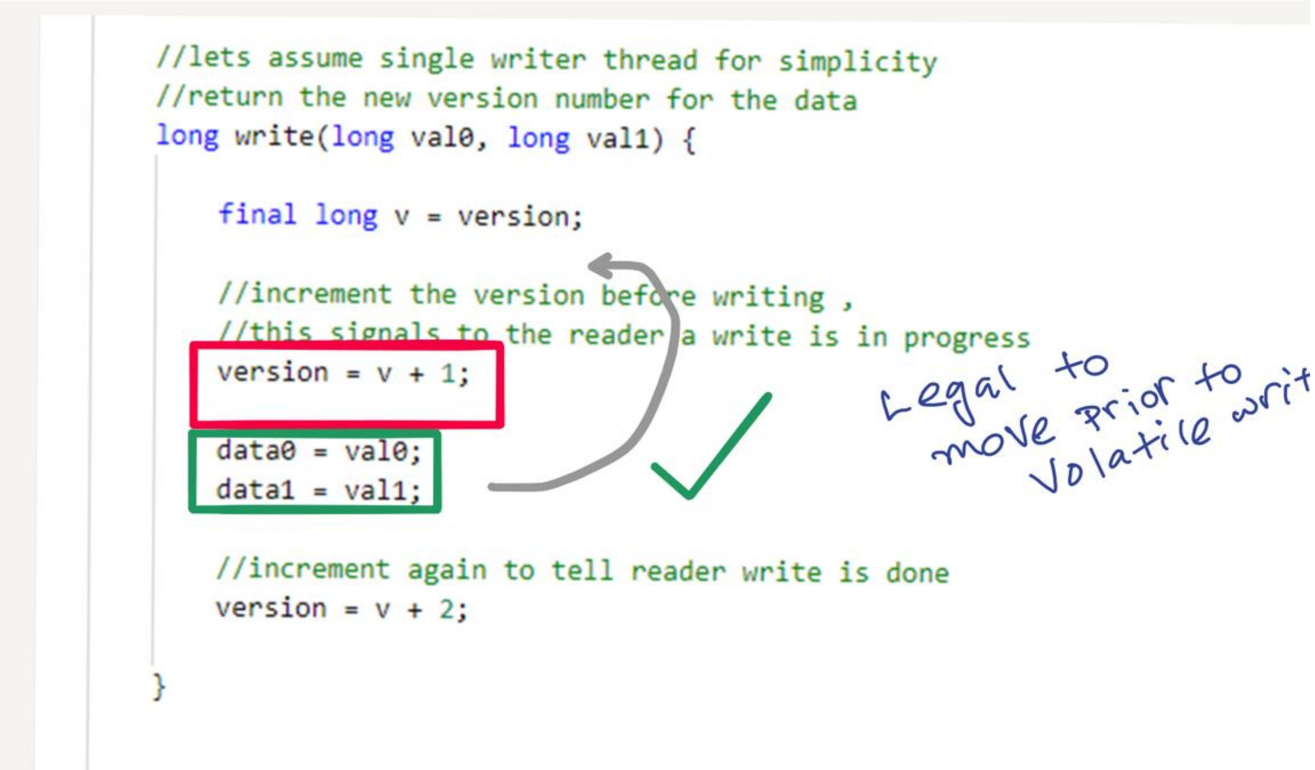
- Load and Store that come after the volatile ***write***, ***can*** move ***before*** a Volatile write

- However prior loads & stores ***cannot*** move after it !



```
// Type your code here, or load an example.
class OrderingTest {
    static long a,b,c, x,y,z;
    static volatile long V;

    static void test(){

        a = x;

        long l = y;


        V = l;


        long r2 = z;

        b = r2;
    }

}
```

This is a legal re-order by JIT compiler



```
//lets assume single writer thread for simplicity
//return the new version number for the data
long write(long val0, long val1) {

    final long v = version;

    //increment the version before writing ,
    //this signals to the reader a write is in progress
    version = v + 1;

    data0 = val0;
    data1 = val1;

    //increment again to tell reader write is done
    version = v + 2;

}
```

- Writes after volatile write

We need a way to instruct the compiler and cpu to restrict re-ordering

- This what memory fences provide
- ***They impose a partial ordering of memory options on either side of barrier***

We need something more precise and fine grained

- Memory Fences help with this
- ***Volaliles provide a implicit weak one way barrier***
- ***Sometimes we need stronger ones as we saw previous example***

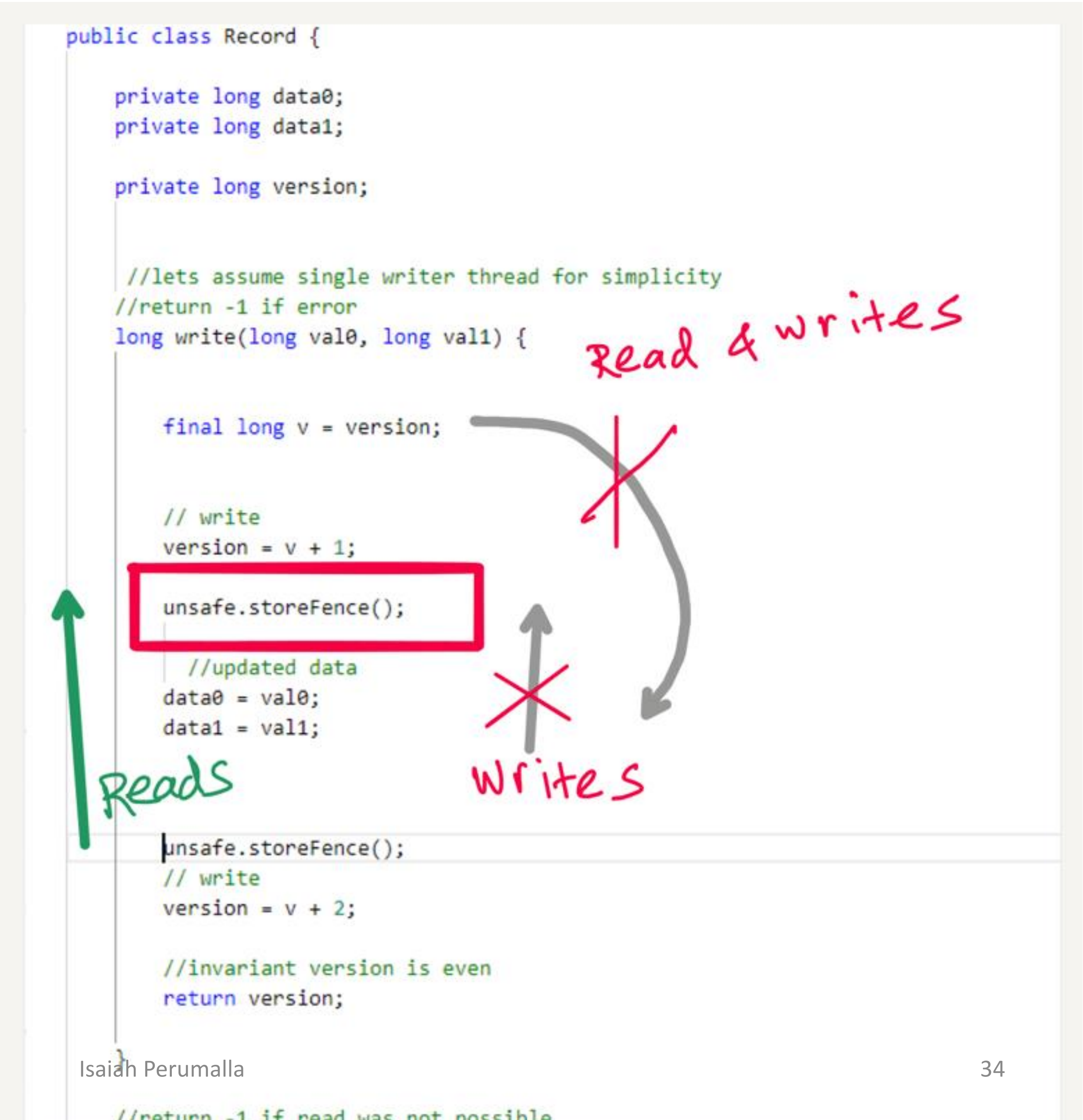
Not all Fences are the same

- Some fences guarantee much stronger properties
- But don’t over fence, as it has a cost

storeFence

- In JAVA StoreFence gives a guarantee that all the Writes and Read operations specified ***before*** the fence will appear to happen before all the STORE operations
- It does not say anything for *Reads* after the fence

Store-Fence

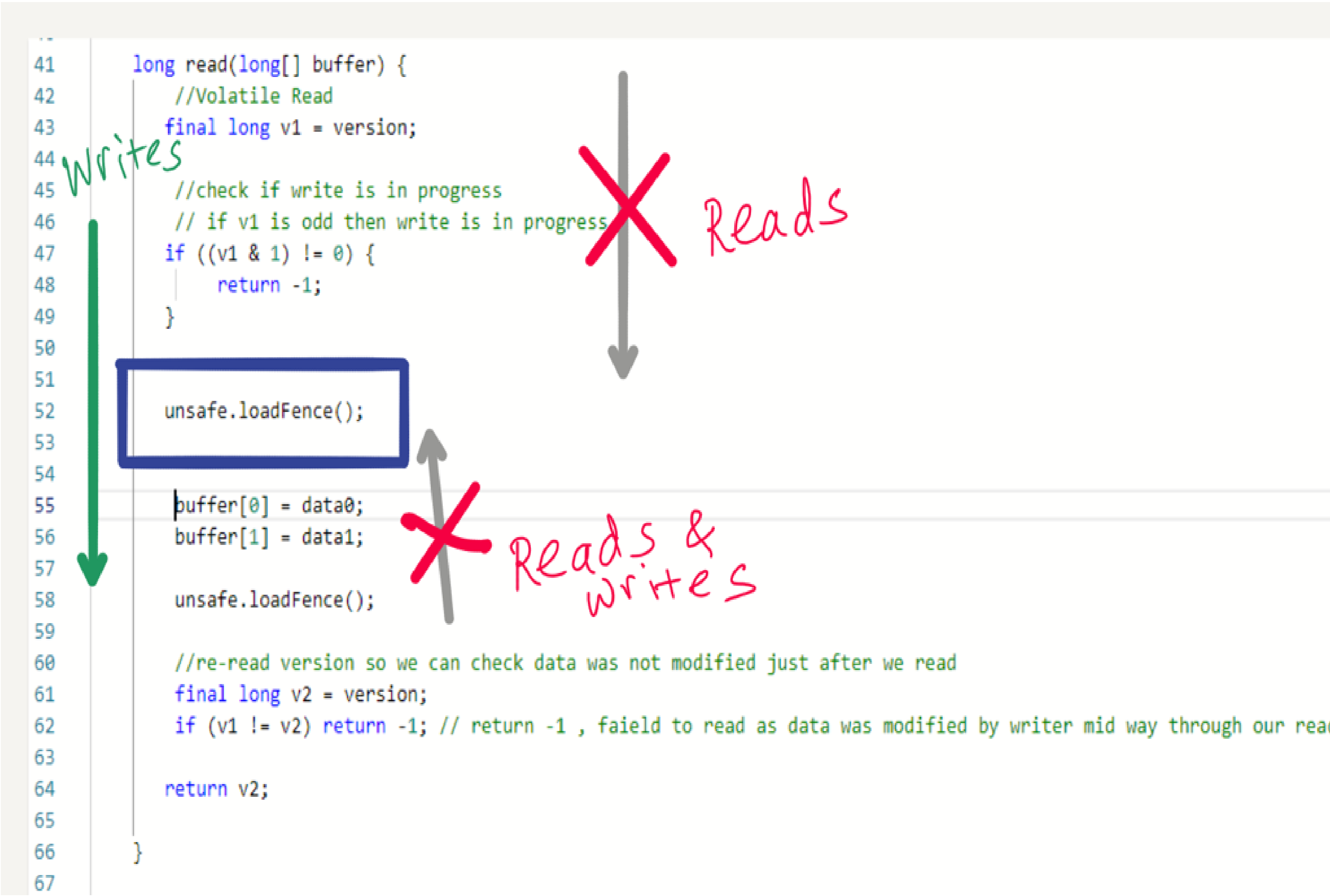


```
public class Record {

    private long data0;
    private long data1;

    private long version;


     //lets assume single writer thread for simplicity
    //return -1 if error
    long write(long val0, long val1) {


        final long v = version;


        // write
        version = v + 1;

        unsafe.storeFence();

          //updated data
        data0 = val0;
        data1 = val1;



        unsafe.storeFence();
        // write
        version = v + 2;

        //invariant version is even
        return version;

    }

    //return -1 if read was not possible
```

Load-Fence

- In JAVA Load-Fence gives a guarantee that ***loads*** before the fence will not be reordered with *loads and stores* after the fence;
- It does not say anything for *Writes* **before** the fence

Load-Fence



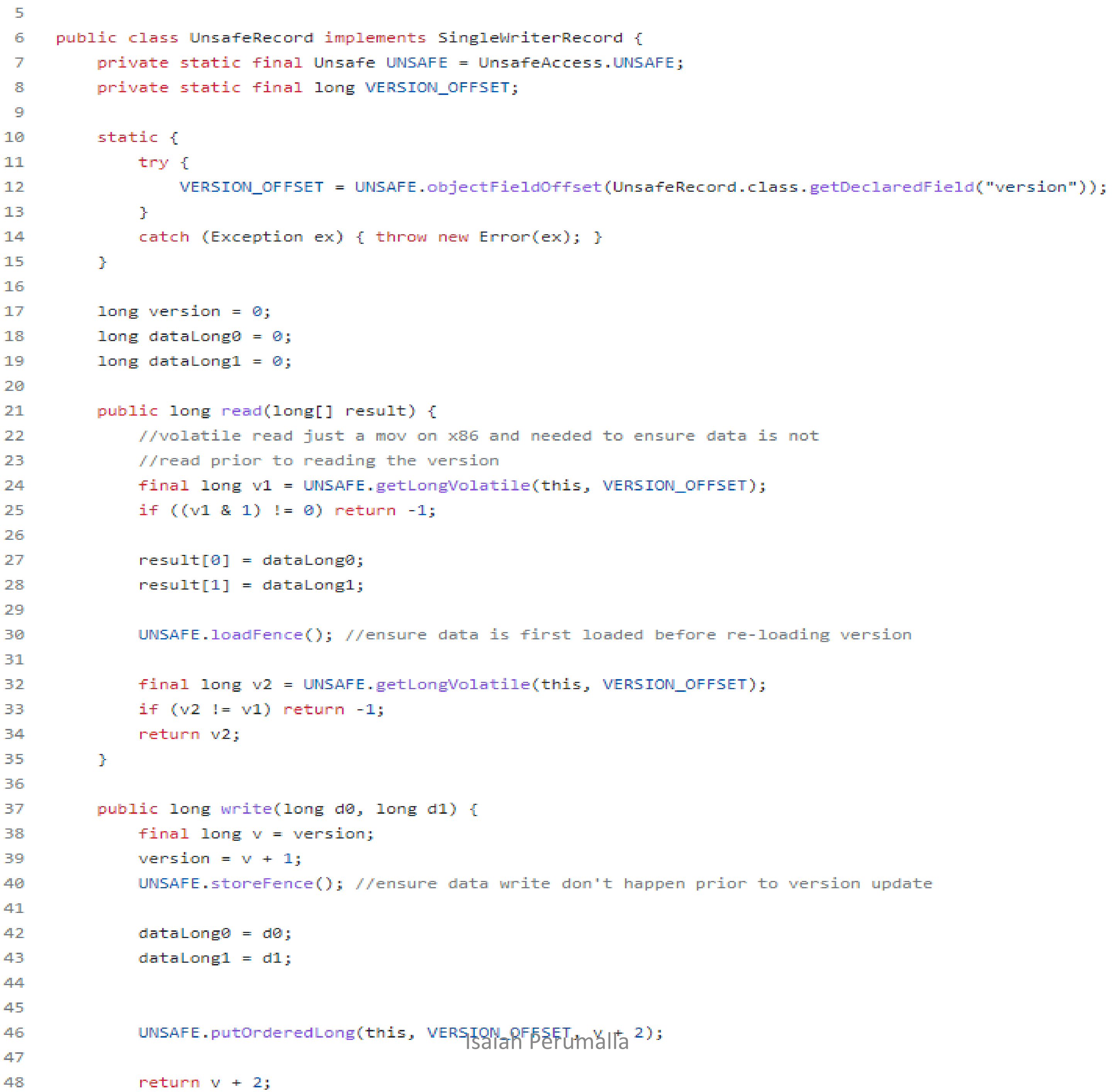
```
    long read(long[] buffer) {
        //Volatile Read
       final long v1 = version;

        //check if write is in progress
        // if v1 is odd then write is in progress
       if ((v1 & 1) != 0) {
           return -1;
       }


       unsafe.loadFence();


        buffer[0] = data0;
        buffer[1] = data1;

        unsafe.loadFence();

        //re-read version so we can check data was not modified just after we read
        final long v2 = version;
        if (v1 != v2) return -1; // return -1 , faield to read as data was modified by writer mid way through our rea

       return v2;

    }
```

Most important Take-way when using Fences

- Fences usually come in pairs

- A lack of pair , usually means there is an error
- Eg if one part of code has load-fence, there must be some where else that is using a store-fence

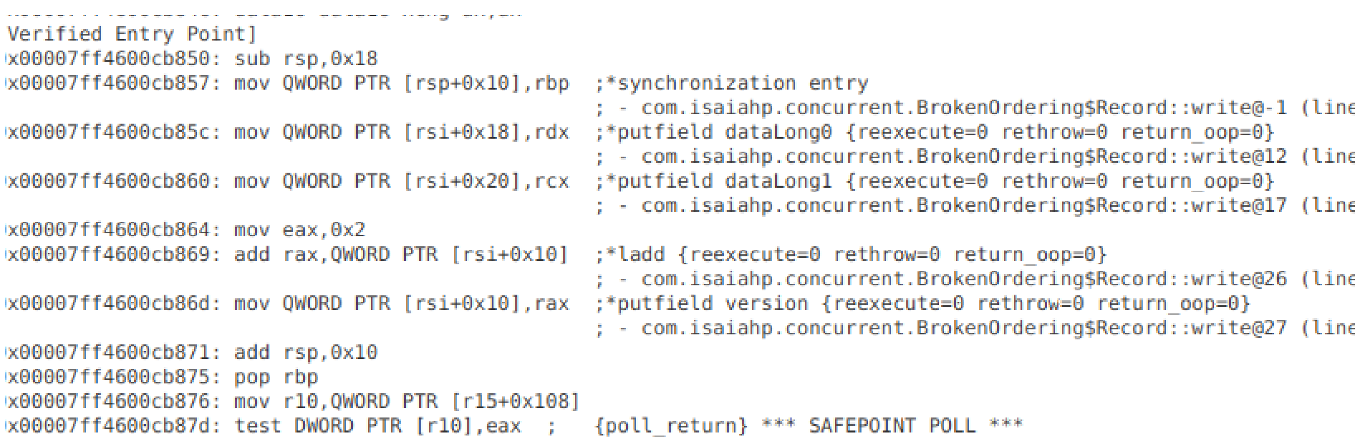
Most important Take-way when using Fences



```java

public class UnsafeRecord implements SingleWriterRecord {
    private static final Unsafe UNSAFE = UnsafeAccess.UNSAFE;
    private static final long VERSION_OFFSET;

    static {
        try {
            VERSION_OFFSET = UNSAFE.objectFieldOffset(UnsafeRecord.class.getDeclaredField("version"));
        }
        catch (Exception ex) { throw new Error(ex); }
    }

    long version = 0;
    long dataLong0 = 0;
    long dataLong1 = 0;

    public long read(long[] result) {
        //volatile read just a mov on x86 and needed to ensure data is not
        //read prior to reading the version
        final long v1 = UNSAFE.getLongVolatile(this, VERSION_OFFSET);
        if ((v1 & 1) != 0) return -1;

        result[0] = dataLong0;
        result[1] = dataLong1;

        UNSAFE.loadFence(); //ensure data is first loaded before re-loading version

        final long v2 = UNSAFE.getLongVolatile(this, VERSION_OFFSET);
        if (v2 != v1) return -1;
        return v2;
    }

    public long write(long d0, long d1) {
        final long v = version;
        version = v + 1;
        UNSAFE.storeFence(); //ensure data write don't happen prior to version update

        dataLong0 = d0;
        dataLong1 = d1;


        UNSAFE.putOrderedLong(this, VERSION_OFFSET, v + 2);

        return v + 2;
```

Use a test harness for concurrent Code

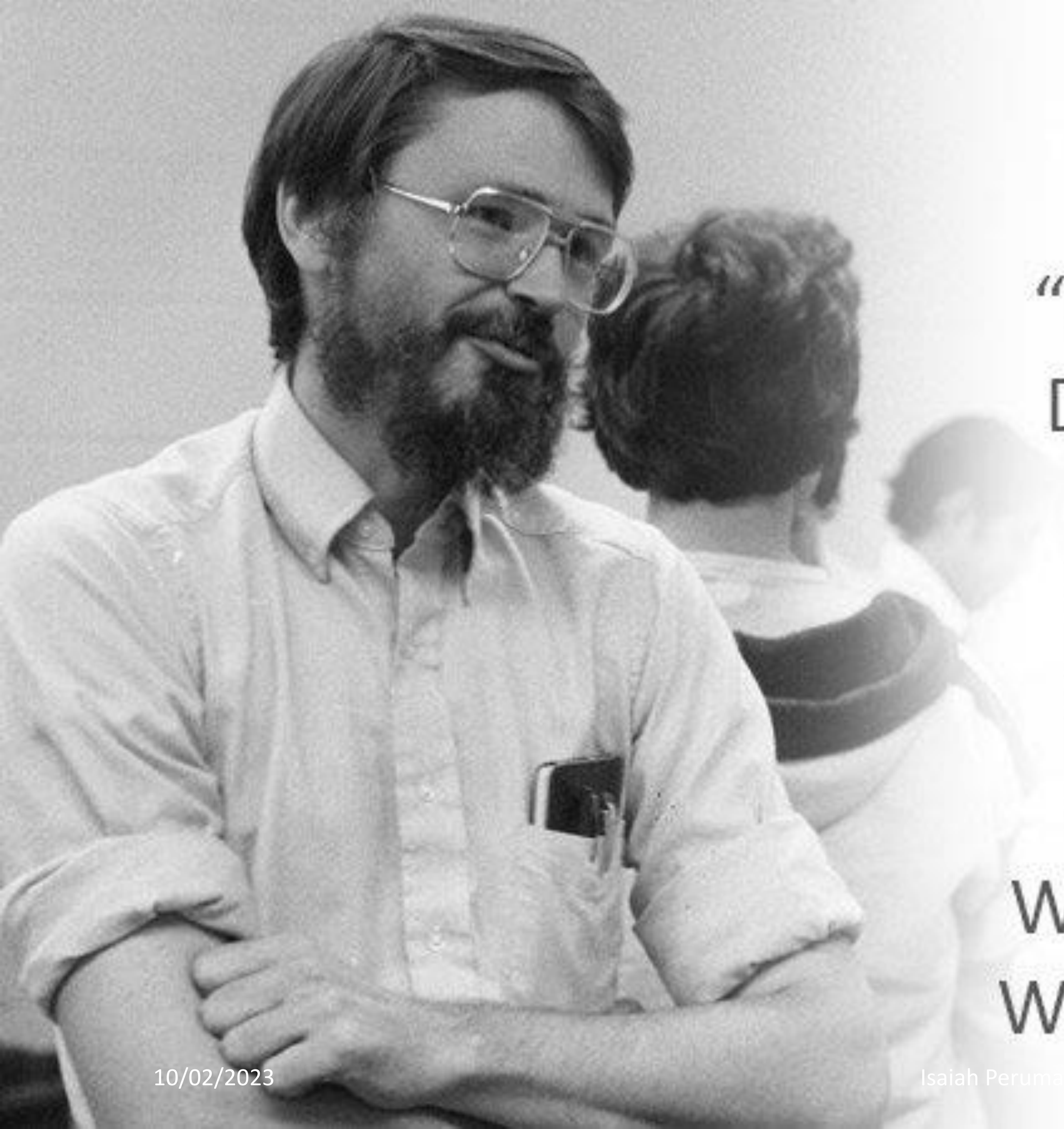
-



```
Verified Entry Point]
x00007ff4600cb850: sub rsp,0x18
x00007ff4600cb857: mov QWORD PTR [rsp+0x10],rbp  ;*synchronization entry
                                                 ; - com.isaiahp.concurrent.BrokenOrdering$Record::write@-1 (line
x00007ff4600cb85c: mov QWORD PTR [rsi+0x18],rdx  ;*putfield dataLong0 {reexecute=0 rethrow=0 return_oop=0}
                                                 ; - com.isaiahp.concurrent.BrokenOrdering$Record::write@12 (line
x00007ff4600cb860: mov QWORD PTR [rsi+0x20],rcx  ;*putfield dataLong1 {reexecute=0 rethrow=0 return_oop=0}
                                                 ; - com.isaiahp.concurrent.BrokenOrdering$Record::write@17 (line
x00007ff4600cb864: mov eax,0x2
x00007ff4600cb869: add rax,QWORD PTR [rsi+0x10]  ;*ladd {reexecute=0 rethrow=0 return_oop=0}
                                                 ; - com.isaiahp.concurrent.BrokenOrdering$Record::write@26 (line
x00007ff4600cb86d: mov QWORD PTR [rsi+0x10],rax  ;*putfield version {reexecute=0 rethrow=0 return_oop=0}
                                                 ; - com.isaiahp.concurrent.BrokenOrdering$Record::write@27 (line
x00007ff4600cb871: add rsp,0x10
x00007ff4600cb875: pop rbp
x00007ff4600cb876: mov r10,QWORD PTR [r15+0x108]
x00007ff4600cb87d: test DWORD PTR [r10],eax  ;   {poll_return} *** SAFEPOINT POLL ***
```

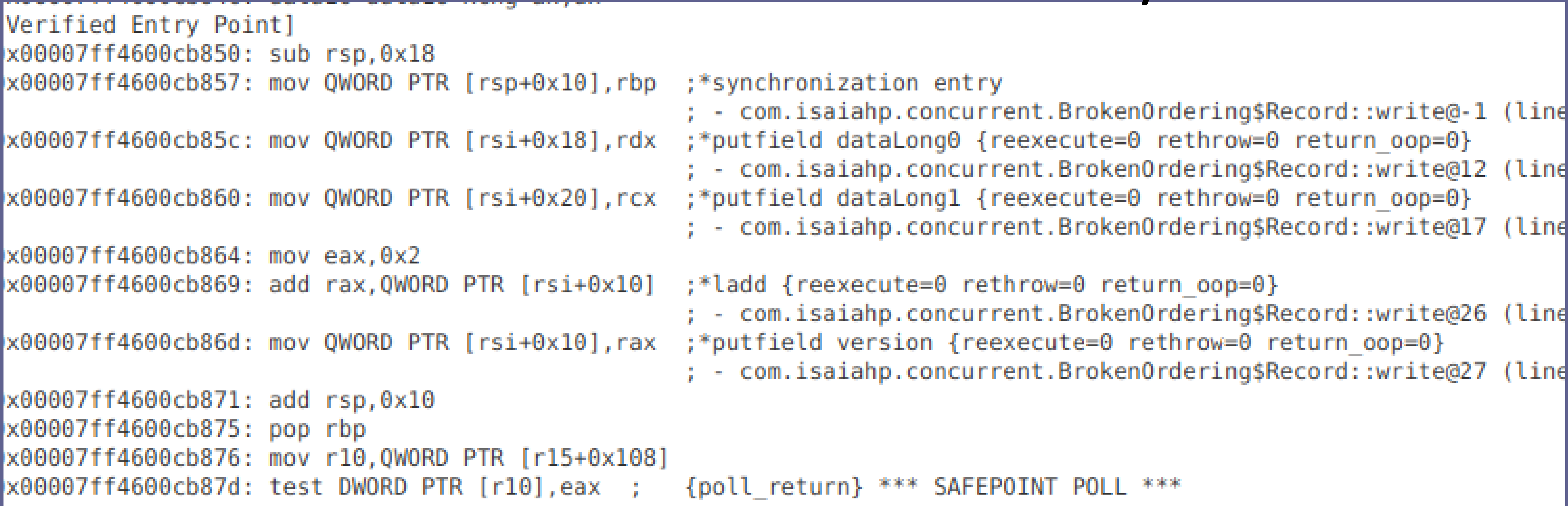
Debugging and Correctness

- Cannot just rely on testing
- Tests are a guide like the Rails
- Ability to reason about the program is more important



“EVERYONE KNOWS THAT DEBUGGING IS TWICE AS HARD AS WRITING A PROGRAM IN THE FIRST PLACE. SO IF YOU'RE AS CLEVER AS YOU CAN BE WHEN YOU WRITE IT, HOW WILL YOU EVER DEBUG IT?”

Quick Detour into x86 Assembly ?



```
Verified Entry Point]
x00007ff4600cb850: sub rsp,0x18
x00007ff4600cb857: mov QWORD PTR [rsp+0x10],rbp  ;*synchronization entry
                                                 ; - com.isaiahp.concurrent.BrokenOrdering$Record::write@-1 (line
x00007ff4600cb85c: mov QWORD PTR [rsi+0x18],rdx  ;*putfield dataLong0 {reexecute=0 rethrow=0 return_oop=0}
                                                 ; - com.isaiahp.concurrent.BrokenOrdering$Record::write@12 (line
x00007ff4600cb860: mov QWORD PTR [rsi+0x20],rcx  ;*putfield dataLong1 {reexecute=0 rethrow=0 return_oop=0}
                                                 ; - com.isaiahp.concurrent.BrokenOrdering$Record::write@17 (line
x00007ff4600cb864: mov eax,0x2
x00007ff4600cb869: add rax,QWORD PTR [rsi+0x10]  ;*ladd {reexecute=0 rethrow=0 return_oop=0}
                                                 ; - com.isaiahp.concurrent.BrokenOrdering$Record::write@26 (line
x00007ff4600cb86d: mov QWORD PTR [rsi+0x10],rax  ;*putfield version {reexecute=0 rethrow=0 return_oop=0}
                                                 ; - com.isaiahp.concurrent.BrokenOrdering$Record::write@27 (line
x00007ff4600cb871: add rsp,0x10
x00007ff4600cb875: pop rbp
x00007ff4600cb876: mov r10,QWORD PTR [r15+0x108]
x00007ff4600cb87d: test DWORD PTR [r10],eax  ;   {poll_return} *** SAFEPOINT POLL ***
```

Quick detour into x86 Assembly

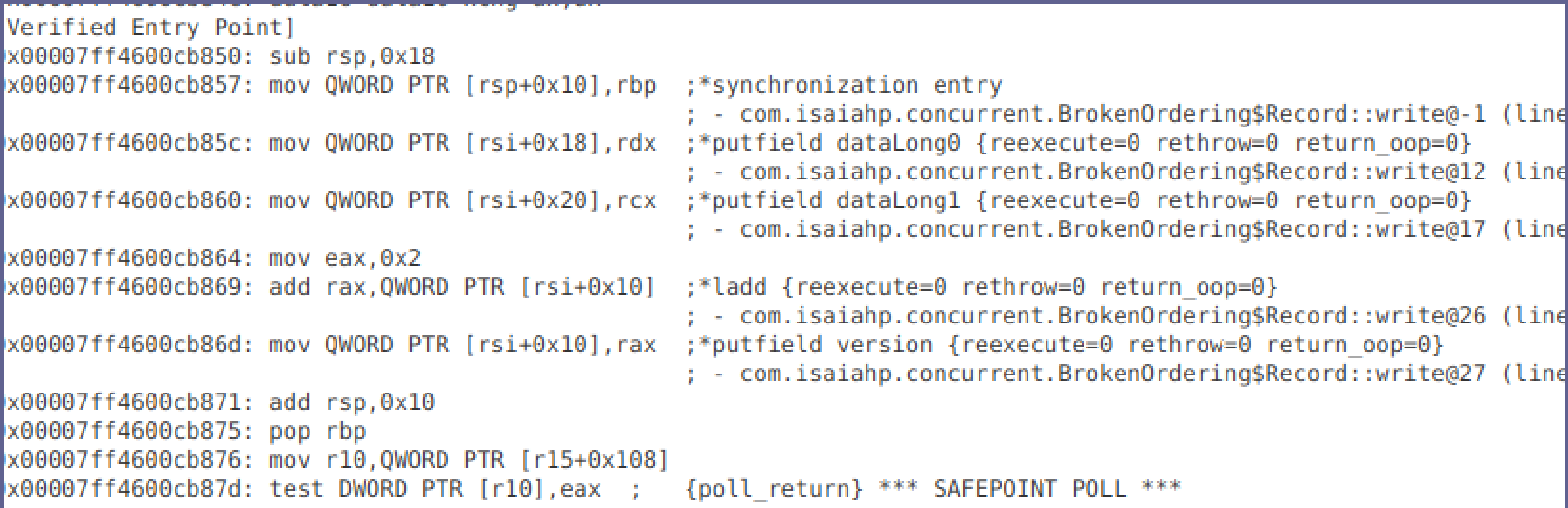
64-bit int registers

1. rbp, rdx, rcx, rax, ..
2. rdi, rsi, rcx, rdx (function args)
3. rax is return value

Reference to memory
Read from memory address [r14]

1. mov eax, QWORD PTR [r14]

Write to memory address, the value in rdx
mov QWORD PTR [r14], rdx



```
Verified Entry Point]
x00007ff4600cb850: sub rsp,0x18
x00007ff4600cb857: mov QWORD PTR [rsp+0x10],rbp  ;*synchronization entry
                                                 ; - com.isaiahp.concurrent.BrokenOrdering$Record::write@-1 (line
x00007ff4600cb85c: mov QWORD PTR [rsi+0x18],rdx  ;*putfield dataLong0 {reexecute=0 rethrow=0 return_oop=0}
                                                 ; - com.isaiahp.concurrent.BrokenOrdering$Record::write@12 (line
x00007ff4600cb860: mov QWORD PTR [rsi+0x20],rcx  ;*putfield dataLong1 {reexecute=0 rethrow=0 return_oop=0}
                                                 ; - com.isaiahp.concurrent.BrokenOrdering$Record::write@17 (line
x00007ff4600cb864: mov eax,0x2
x00007ff4600cb869: add rax,QWORD PTR [rsi+0x10]  ;*ladd {reexecute=0 rethrow=0 return_oop=0}
                                                 ; - com.isaiahp.concurrent.BrokenOrdering$Record::write@26 (line
x00007ff4600cb86d: mov QWORD PTR [rsi+0x10],rax  ;*putfield version {reexecute=0 rethrow=0 return_oop=0}
                                                 ; - com.isaiahp.concurrent.BrokenOrdering$Record::write@27 (line
x00007ff4600cb871: add rsp,0x10
x00007ff4600cb875: pop rbp
x00007ff4600cb876: mov r10,QWORD PTR [r15+0x108]
x00007ff4600cb87d: test DWORD PTR [r10],eax  ;   {poll_return} *** SAFEPOINT POLL ***
```

Quick detour into x86 Assembly