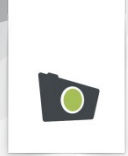
Java Memory Model for beginners and practitioners

by Vadym Kazulkin and Rodion Alukhanov, ip.labs GmbH

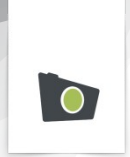


Topics

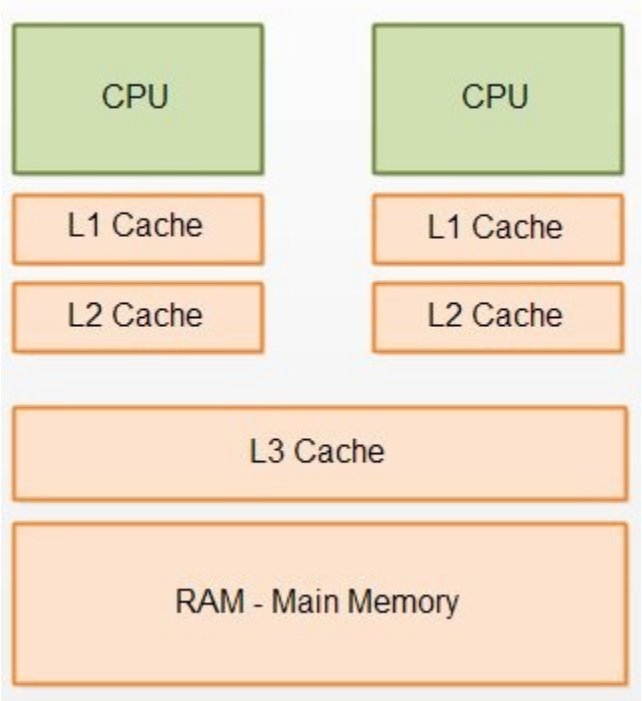
- Hardware Memory Model
- Java Memory Model
- Practical examples related to Java Memory Model
- JCStress Tool and Demo
- The future of the Java Memory Model

Quiz
What values of x can we see, if y=1 is set?

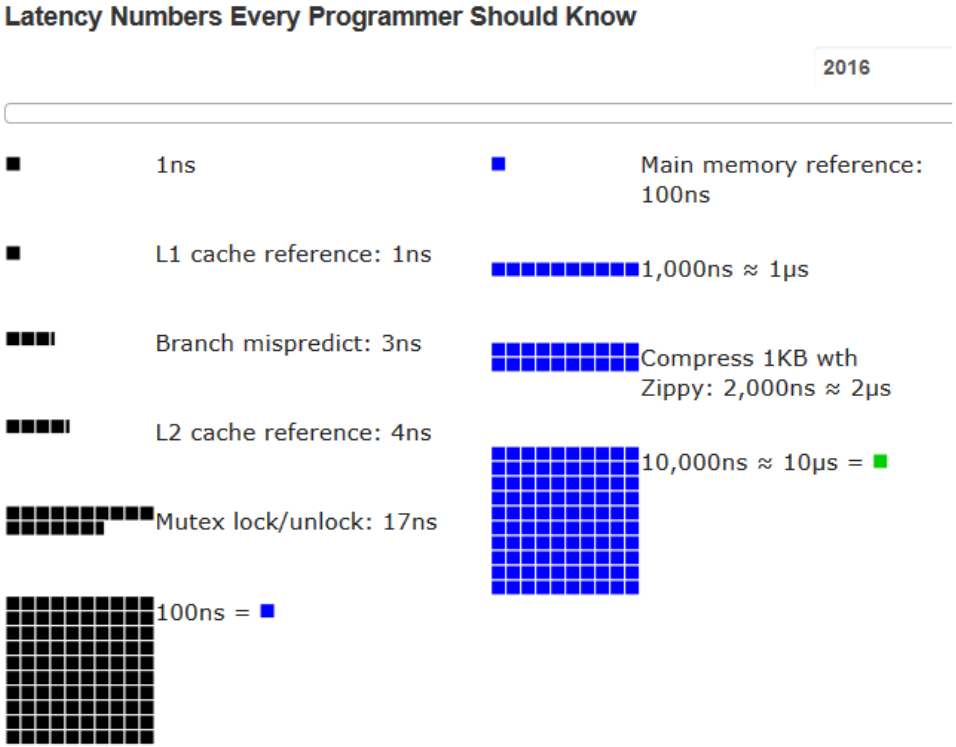
```
public class MyClass{

    int x, y;

    public void executeOnCPU1() {
        x = 1;
        x = 2;
        y = 1;
        x = 3;
    }

    public void executeOnCPU2() {
        System.out.println("x: "+x+ " y: "+y);
    }
}
```



Hardware Memory Model



Hardware Memory Model



Source : http://people.eecs.berkeley.edu/~rcs/research/interactive_latency.html

Hardware Memory Model
Cache Coherence

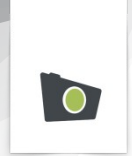
Cache coherence is the consistency of shared resource data that ends up stored in multiple local caches.

Hardware Memory Model
Cache Coherence Protocol MESI

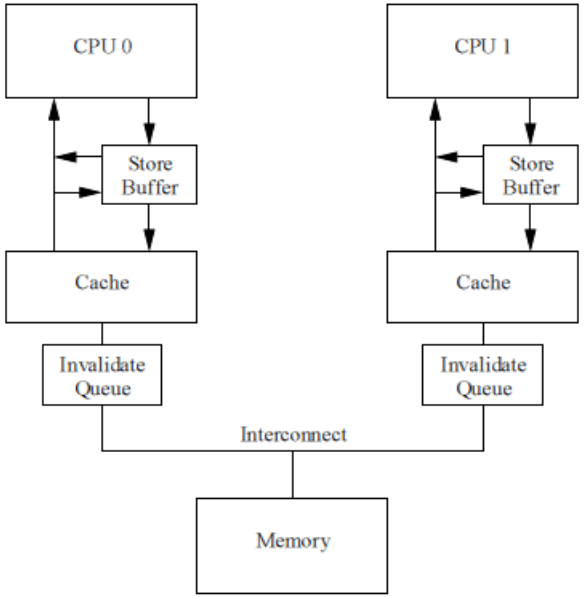
Modified
The entry is only present in the current cache and it is modified.
Exclusive
The entry is only present in the current cache and has not been modified.
Shared
More that one processor has the entry in its cache.
Invalid
Indicates that this entry in the cache is invalid.

Hardware Memory Model
Stored Buffer & Invalidate Queues

Stored Buffer & Invalidate Queues used to increase performance of CPUs



Hardware Memory Model
Stored Buffer & Invalidate Queues

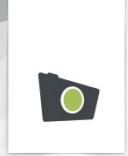


Hardware Memory Model
Memory Ordering

- A given CPU will always perceive its own memory operations as occurring in program order.
- Some other CPU may observe memory operations results in a **different order** than what's written in the program.

Hardware Memory Model
Memory Barrier

Memory barrier, aka membar or fence is the instruction that causes CPU to enforce an ordering constraint on memory operations issued before and after the barrier instruction.



Hardware Memory Model
Memory Barrier

Store Memory Barrier applies all the stores in the **store buffer**

Load Memory Barrier applies all the invalidates that are already in the **invalidate queue**.

Hardware Memory Model
Types of Memory Barrier

LoadLoad membar

Load operations before the barrier must complete their execution before any **Load** operation after the barrier.

StoreStore membar

Store operations before the barrier must complete their execution before any **Store** operation after the barrier.

Hardware Memory Model
Types of Memory Barrier

LoadStore membar

Load operations before the barrier must complete their execution before any **Store** operation after the barrier.

StoreLoad membar

Store operations before the barrier must complete their execution before any **Load** operation after the barrier.

Hardware Memory Model
Types of Memory Barrier/Example

```
int x=0; boolean done=false;

void executeOnCPU1() {
    x = 1;
    done=true;
}

void executeOnCPU2() {
    while(!done) {
      assert x=1;
     }
}
```

Hardware Memory Model
Types of Memory Barrier/Example

```
int x=0; boolean done=false;

void executeOnCPU1() {
    x = 1;
    InsertStoreStoreMembar();
    done=true;
}

void executeOnCPU2() {
    while(!done) {
      InsertLoadLoadMembar();
      assert x=1;
     }
}
```

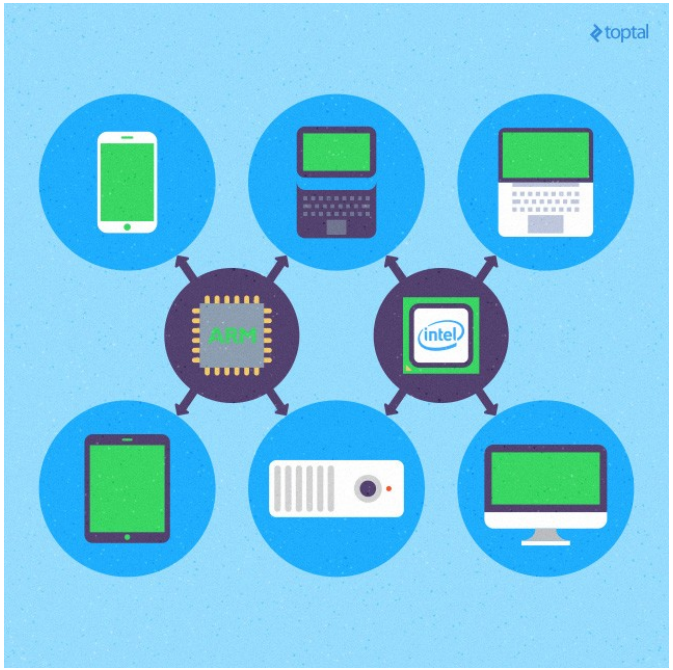
Hardware Memory Model

Allowed memory reorderings for some architectures

Type	ARMv7	x86	AMD64
LoadLoad	Y		
LoadStore	Y		
StoreStore	Y		
StoreLoad	Y	Y	Y

Source : https://en.wikipedia.org/wiki/Memory_ordering

Hardware Memory Model



Quiz for reordering - second attempt:
What values of x can we see on x86, if y=1 is set?

```
public class MyClass{
    int x, y;
    public void executeOnCPU1() {
        x = 1;
        x = 2;
        y = 1;
        x = 3;
    }
    public void executeOnCPU2() {
        System.out.println("x: "+x+ " y: "+y);
    }
}
```

Memory Model is Trade-Off

How hard it is to use a language

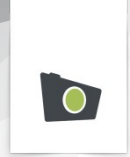
vs.

How hard it is to build a language implementation

vs.

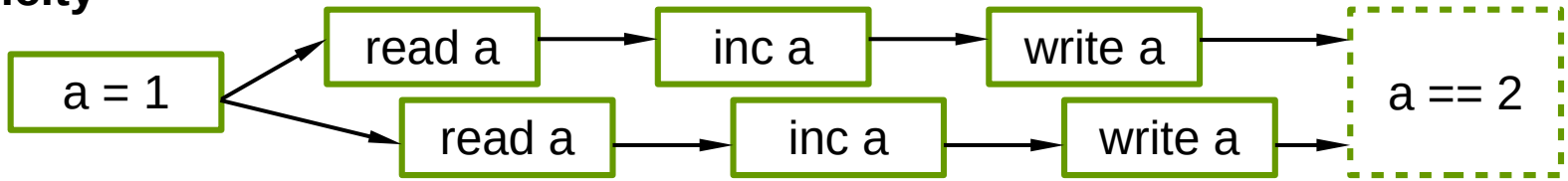
How hard it is to build appropriate hardware

Source: http://shipilev.net/talks/narnia-2555-jmm-pragmatics-en.pdf

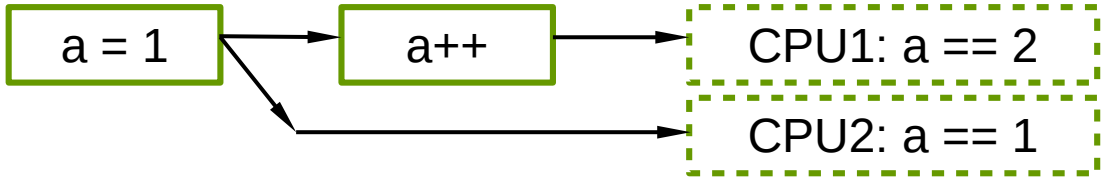


Java Memory Model

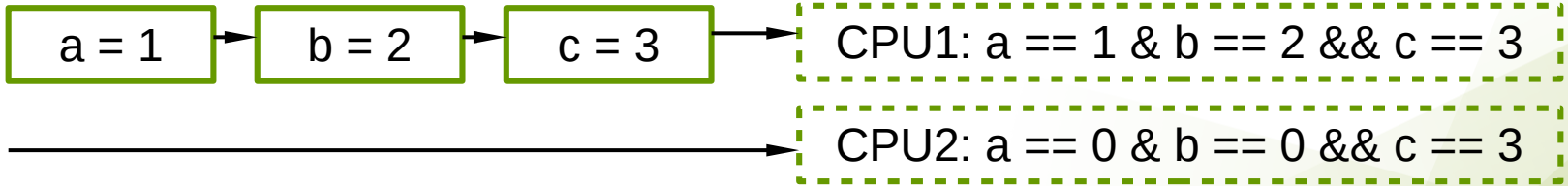
Atomicity

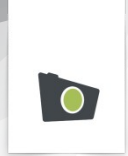


Visibility



Ordering





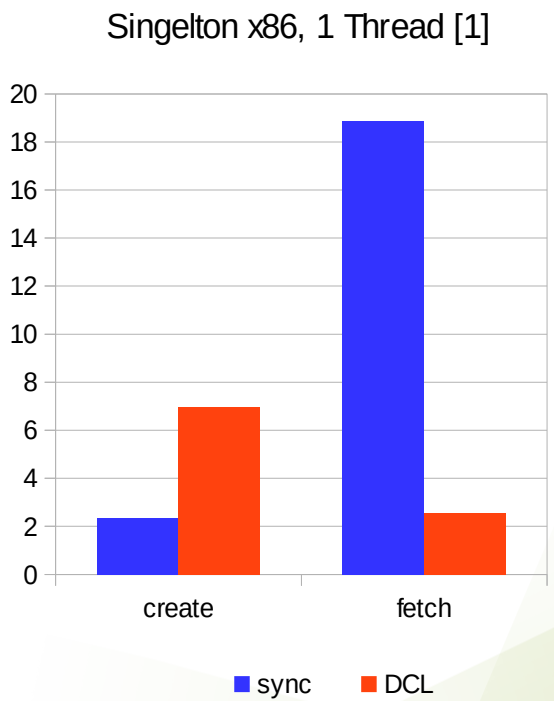
Atomicity 1

```
class Counter {
    private volatile long counter;

    public synchronize long inc() {
        counter++;
        return counter;
    }
    public void set(long v) {
        counter = v;
    }
    public long getCounter() {
        return counter.get();
    }
}
```

synchronize vs. volatile

Atomicity (DCL 1)

```
class DoubleCheckLocking {
  private volatile Resource resource;

  public Resource getResource() {
    if (resource == null) {
      synchronized (this) {
        if (resource == null) {
          resource = new Resource();
        }
      }
    }
    return resource;
  }
}
```



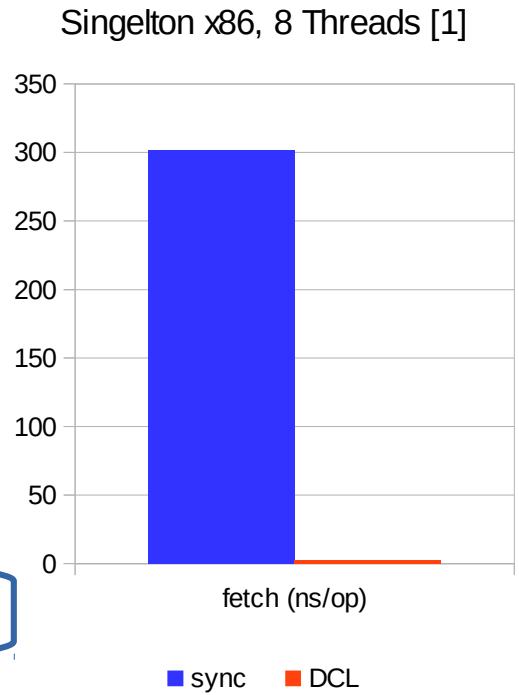
more threads?

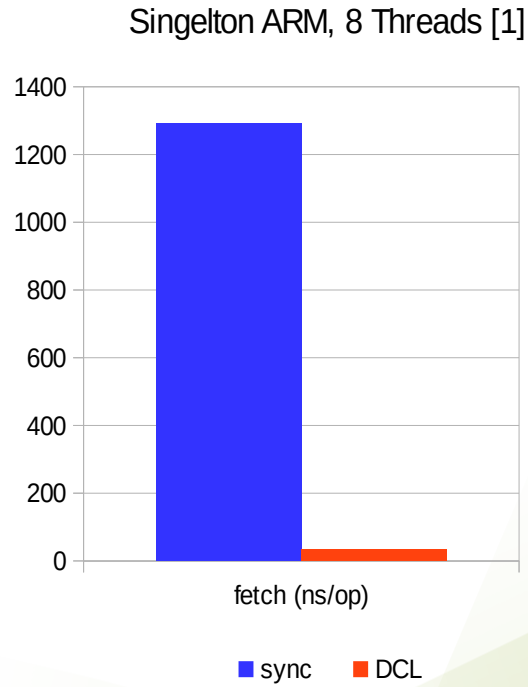
[1] https://shipilev.net/blog/2014/safe-public-construction

Atomicity (DCL 2)

```
class DoubleCheckLocking {
  private volatile Resource resource;

  public Resource getResource() {
    if (resource == null) {
      synchronized (this) {
        if (resource == null) {
          resource = new Resource();
        }
      }
    }
    return resource;
  }
}
```

without syncronized?

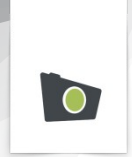




[1] https://shipilev.net/blog/2014/safe-public-construction

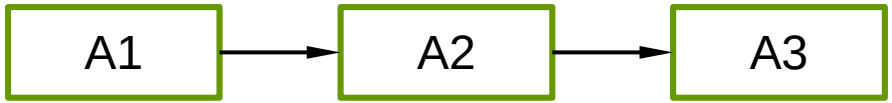
Atomicity 2

```
class Counter {
    private final AtomicLong counter = new AtomicLong();

    public long inc(long v) {
        return counter.incrementAndGet()
    }
    public void set(long v) {
        counter.set(v);
    }
    public long getCounter() {
        return counter.get();
    }
}
```

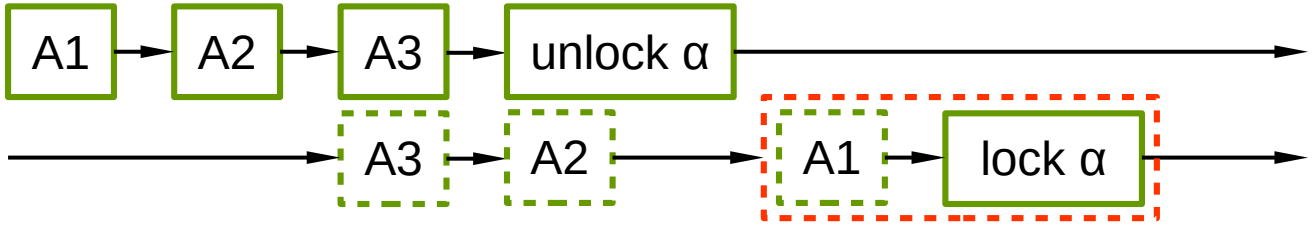


Happens-Before

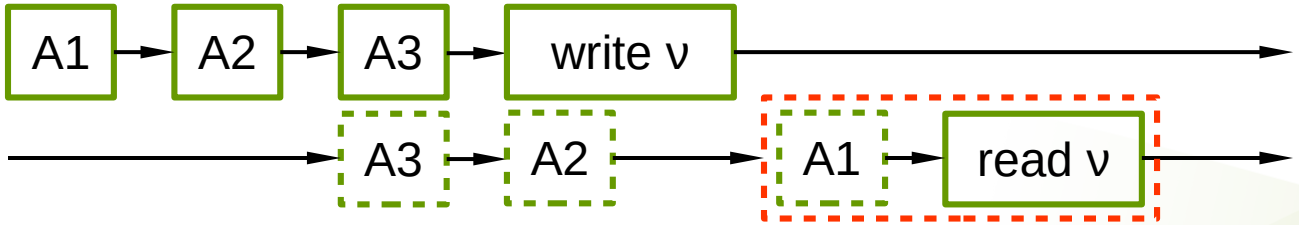
Single Thread

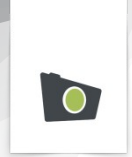


Synchronized



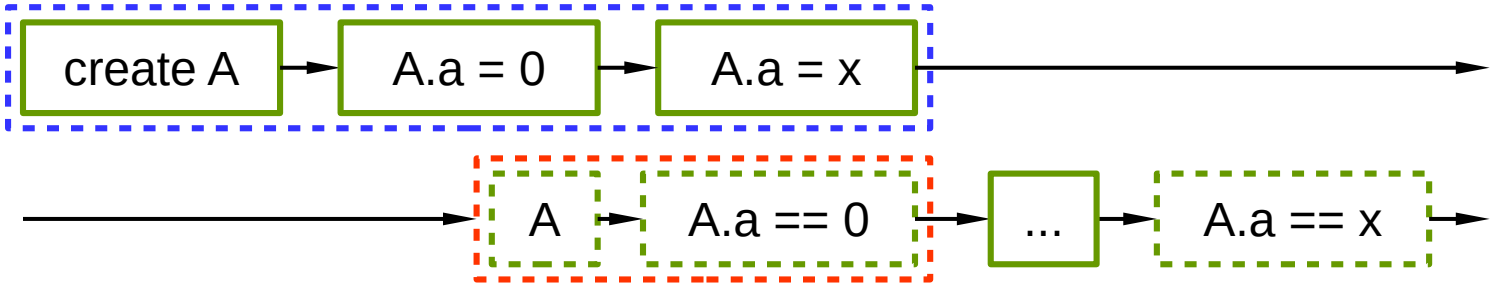
Volatile



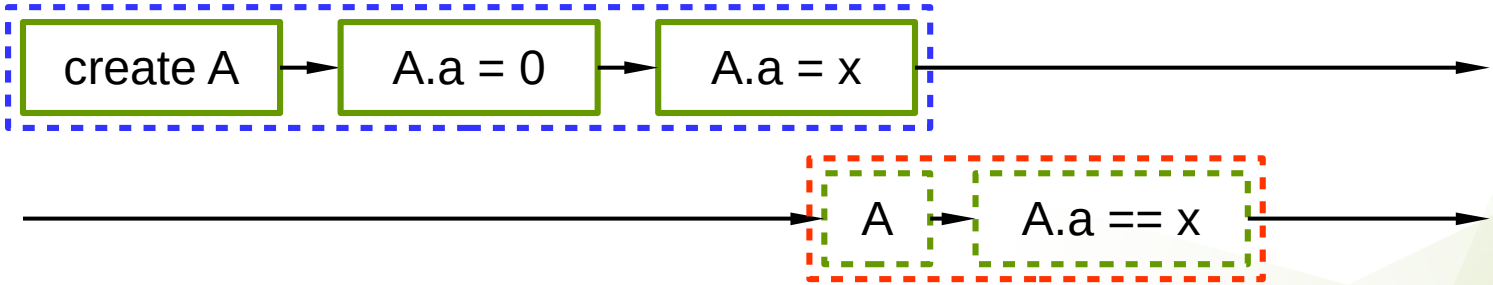


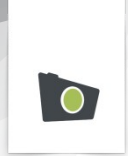
Happens-Before

Default init



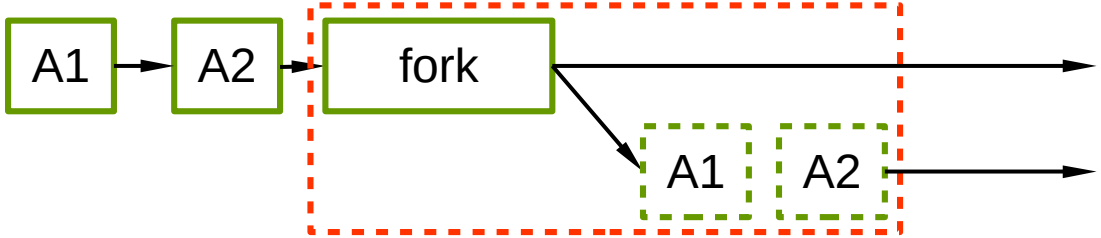
Final fields init



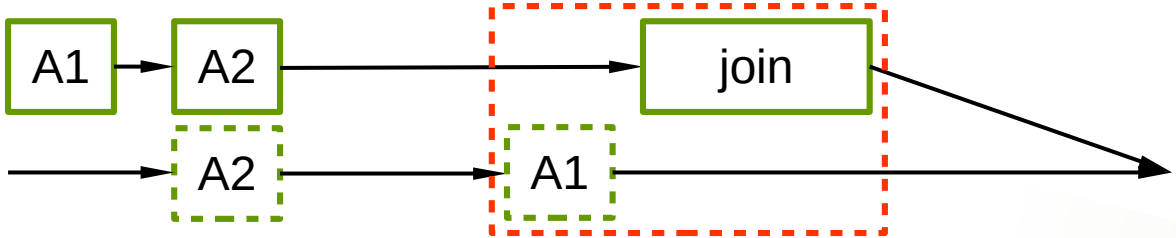


Happens-Before

Fork



Join

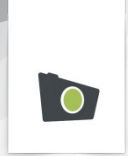


Visibility

```
class Counter {
    private volatile int counter;

    public synchronize int inc() {
        counter++;
        return counter;
    }
    public void set(int v) {
        counter = v;
    }
    public long getCounter() {
        return counter.get();
    }
}
```

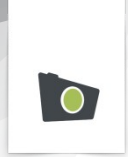
Safe Publication

```
class Interval {
    public Pair value;

    static class Pair() {
        int final lower, upper;
        public Pair(int i1, int i2) {...}
    }

    public void set(int i1, int i2) {
        if (i1 > i2) throw new SomeException();
        value = new Pair(i1, i2);
    }
}
```



Safe Initialization

```
class Interval {
    public volatile Pair value;

    static class Pair() {
        int final lower, upper;
        public Pair(int i1, int i2) {...}
    }

    public void set(int i1, int i2) {
        if (i1 > i2) throw new SomeException();
        value = new Pair(i1, i2);
    }
}
```



Compare And Set 1

```
class Fibonacci { // 1, 1, 2, 3, 5, 8, 13, 21...
    private Pair v = new Pair(0, 1);

    static class Pair() {
        int lower, upper;
        public Pair(int i1, int i2) {...}
    }

    public synchronize int next() {
        v = new Pair(v.upper, v.lower + v.upper);
        return v.lower;
    }
}
```

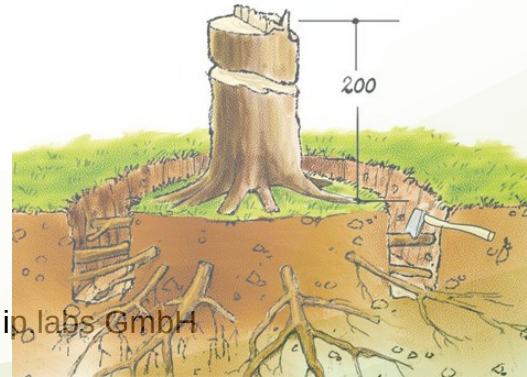
volatile?

AtomicInteger?

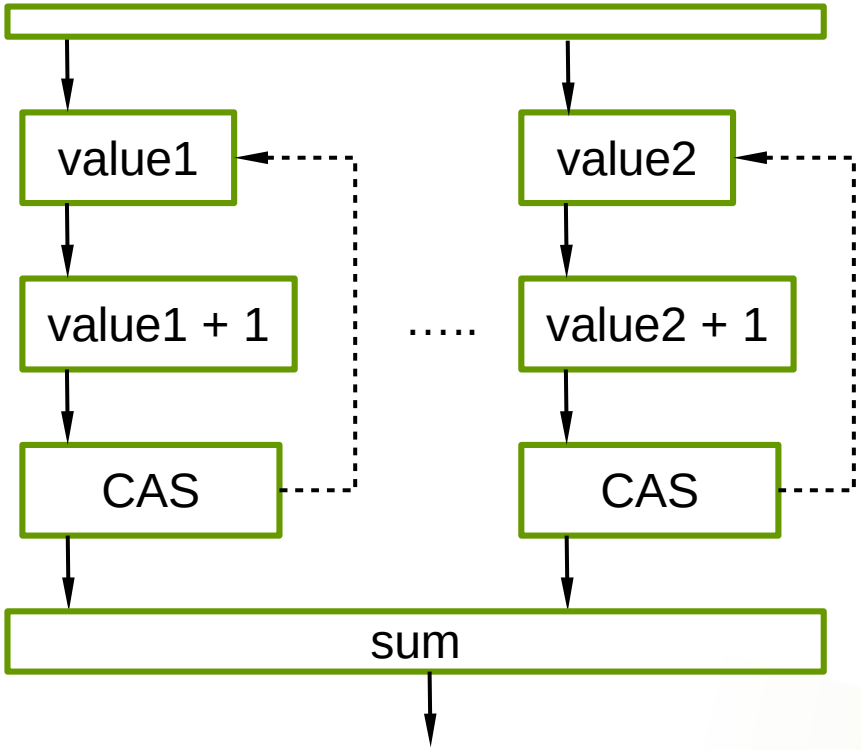
Compare And Set 2

```
class Fibonacci { // 0, 1, 1, 2, 3, 5, 8, 13, 21...
  private final AtomicReference v = new AtomicReference();

  public int next() {
    Pair newValue;
    do {
      Pair old = v.get();
      newValue = new Pair(old.upper, old.lower + old.upper);
    } while(v.compareAndSet(old, newValue));
    return newValue.lower;
  }
}
```

spin lock?



LongAdder



JCStress

The Java Concurrency Stress tests (jcstress) is an experimental harness and a suite of tests to aid the research in the correctness of concurrency support in the JVM, class libraries, and hardware.

Source: http://openjdk.java.net/projects/code-tools/jcstress/

JCStress

- requires at least 2 online CPUs .
- few threads executing the test concurrently, sometimes rendez-vous'ing over the shared state.
- multiple state objects generated per each run. Threads then either mutate or observe that state object.
- test harness is collecting statistics on the observed states.

Quiz for Reordering - last attempt:
What values of x can we see, if y=1 is set?

```
public class MyClass{
    int x, y;
    public void executeOnCPU1() {
        x = 1;
        x = 2;
        y = 1;
        x = 3;
    }
    public void executeOnCPU2() {
        System.out.println("x: "+x+ " y: "+y);
    }
}
```

Answer to Quiz : UnfencedAcquireReleaseTest

JVM options: [-client] Iterations: 5 Time: 1000

Observed state	Occurrence	Expectation	
0, 0	407449	ACCEPTABLE	Before observing releasing write to, any value is OK for $x.
0, 1	7	ACCEPTABLE	Before observing releasing write to, any value is OK for $x.
0, 2	114	ACCEPTABLE	Before observing releasing write to, any value is OK for $x.
0, 3	13399	ACCEPTABLE	Before observing releasing write to, any value is OK for $x.
1, 0	0	ACCEPTABLE_INTERESTING	Without fence or volatile can read the default or old value for $x after $y is observed.
1, 1	0	ACCEPTABLE_INTERESTING	Without fence or volatile can read the default or old value for $x after $y is observed.
1, 2	7	ACCEPTABLE	Can see a released value of $x if $y is observed.
1, 3	22165994	ACCEPTABLE	Can see a released value of $x if $y is observed.

JVM options: [-server] Iterations: 5 Time: 1000

Observed state	Occurrence	Expectation	
0, 0	354385	ACCEPTABLE	Before observing releasing write to, any value is OK for $x.
0, 1	2	ACCEPTABLE	Before observing releasing write to, any value is OK for $x.
0, 2	44	ACCEPTABLE	Before observing releasing write to, any value is OK for $x.
0, 3	18029	ACCEPTABLE	Before observing releasing write to, any value is OK for $x.
1, 0	1	ACCEPTABLE_INTERESTING	Without fence or volatile can read the default or old value for $x after $y is observed.
1, 1	0	ACCEPTABLE_INTERESTING	Without fence or volatile can read the default or old value for $x after $y is observed.
1, 2	4	ACCEPTABLE	Can see a released value of $x if $y is observed.
1, 3	24198455	ACCEPTABLE	Can see a released value of $x if $y is observed.

JVM options: [-server, -XX:-TieredCompilation] Iterations: 5 Time: 1000

Observed state	Occurrence	Expectation	
0, 0	415111	ACCEPTABLE	Before observing releasing write to, any value is OK for $x.
0, 1	6	ACCEPTABLE	Before observing releasing write to, any value is OK for $x.
0, 2	38	ACCEPTABLE	Before observing releasing write to, any value is OK for $x.
0, 3	22785	ACCEPTABLE	Before observing releasing write to, any value is OK for $x.
1, 0	6	ACCEPTABLE_INTERESTING	Without fence or volatile can read the default or old value for $x after $y is observed.
1, 1	0	ACCEPTABLE_INTERESTING	Without fence or volatile can read the default or old value for $x after $y is observed.
1, 2	63	ACCEPTABLE	Can see a released value of $x if $y is observed.
1, 3	22030031	ACCEPTABLE	Can see a released value of $x if $y is observed.

Answer to Quiz : UnfencedAcquireReleaseTest

Even on strong hardware memory models such as x86 reordering occurs because the java compiler und Just-In-Time compiler also reorder in absence of happens-before.

So (y,x)=(1,0) and (y,x)=(1,1) are also possible.

Declaring x&y as **volatile** fixes the problem.

UnsafePublicationTest

JVM options: [-Xint] Iterations: 5 Time: 1000

Observed state	Occurrence	Expectation	
-1	1332765	ACCEPTABLE	The object is not yet published
0	0	ACCEPTABLE	The object is published, but all fields are 0.
1	0	ACCEPTABLE	The object is published, at least 1 field is visible.
2	0	ACCEPTABLE	The object is published, at least 2 fields are visible.
3	0	ACCEPTABLE	The object is published, at least 3 fields are visible.
4	566055	ACCEPTABLE	The object is published, all fields are visible.

JVM options: [-server, -XX:-TieredCompilation] Iterations: 5 Time: 1000

Observed state	Occurrence	Expectation	
-1	5502508	ACCEPTABLE	The object is not yet published
0	118	ACCEPTABLE	The object is published, but all fields are 0.
1	15	ACCEPTABLE	The object is published, at least 1 field is visible.
2	122	ACCEPTABLE	The object is published, at least 2 fields are visible.
3	12	ACCEPTABLE	The object is published, at least 3 fields are visible.
4	14365445	ACCEPTABLE	The object is published, all fields are visible.

JVM options: [-client] Iterations: 5 Time: 1000

Observed state	Occurrence	Expectation	
-1	2234766	ACCEPTABLE	The object is not yet published
0	163	ACCEPTABLE	The object is published, but all fields are 0.
1	27	ACCEPTABLE	The object is published, at least 1 field is visible.
2	221	ACCEPTABLE	The object is published, at least 2 fields are visible.
3	65	ACCEPTABLE	The object is published, at least 3 fields are visible.
4	9840558	ACCEPTABLE	The object is published, all fields are visible.

UnsafePublicationTest

Declaring **all** instance variables as **final** is essential to seeing all of them correctly set after the constructor execution is completed.

Java Memory Model Wrap Up

- JMM also supports synchronized-with (acquire-release) semantics

  implemented via LoadStore|LoadLoad for acquire and LoadStore|StoreStore for release

  if there is synchronzed-with, there is also happens-before

- JMM describes synchronized-with, happens-before a.s.o. and has no notion of the memory barrier (it's an implementation detail)

  synchronize (new MyObject()) doesn't have any sence and will be removed

Java Memory Model Wrap Up

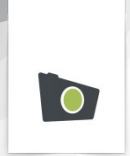
- JMM also supports synchronized-with (acquire-release) semantics

  implemented via LoadStore|LoadLoad for acquire and LoadStore|StoreStore for release

  if there is synchronzed-with, there is also happens-before

- JMM describes synchronized-with, happens-before a.s.o. and has no notion of the memory barrier (it's an implementation detail)

  synchronize (new MyObject()) doesn't have any sence and will be removed



Future of Java Memory Model

Old spec JSR 133 is outdated (lazySet & weakCompareAndSet methods of Atomics not specified in sense of re-ordering).

New spec JEP 188: Java Memory Model Update :

- C11/C++11 compatibility.
- Volatile for automicy for long/double on 64bit.
- Final field initialization problem.
- Testing/Tool support.
- Out-of-Thin-Air (OoTA) Problem.

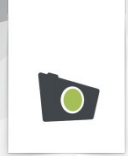
Source: http://openjdk.java.net/jeps/188

Out-of-Thin-Air Problem

Only one outcome is possible a=b=0;

```
int a = 0, b = 0;
```

`r1 = a;`	`r2 = b;`
`if (r1 != 0)`	`if (r2 != 0)`
`  b = 42;`	`  a = 42;`



Out-of-Thin-Air Problem

But runtime/hardware can speculate

```
int a = 0, b = 0;
```

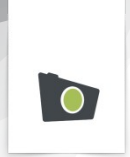
```
int r1 = a;
if (r1 != 0)
    b = 42;
```

→

```
int r1 = a;
b = 42;
if (r1 == 0)
  b = 0;
```

→

```
b = 42;
int r1 = a;
if (r1 == 0)
  b = 0;
```



Out-of-Thin-Air Problem

.., so a=b=42, which is forbidden by JVM(Out-of-Thin-Air values)

```
int a = 0, b = 0;
```

`b = 42;`	
	`r2 = b;`
	`if (r2 != 0)`
	`    a = 42;`
`r1 = a;`	
`if (r1 == 0)`	
`    b = 0;`	

Take-aways

- Understanding of Java Memory Model is crucial in order to develop the correct code running concurrently.
- Don't rely on CPU (re)ordering-policy, compiler and JIT-compiler also make optimizations and therefore re-order the instructions.
- Code against the specification not the implementation.
- Use JCStress to test your assumptions.

Useful links

- JSR 133 (Java Memory Model) FAQ
  https://www.cs.umd.edu/~pugh/java/memoryModel/jsr-133-faq.html
- Memory Model
  https://docs.oracle.com/javase/specs/jls/se7/html/jls-17.html#jls-17.4
- Aleksey Shipilev „Java Memory Model Pragmatics“
  http://shipilev.net/blog/2014/jmm-pragmatics/
- Aleksey Shipilev „Close Encounters of The Java Memory Model Kind“
  http://shipilev.net/blog/2016/close-encounters-of-jmm-kind/
- JEP 188: Java Memory Model Update
  http://openjdk.java.net/jeps/188
- JCStress
  http://openjdk.java.net/projects/code-tools/jcstress/

Questions?

Thank You!