

Thread Safety with Phaser, StampedLock and VarHandle

Thread Safety with Phaser, StampedLock and VarHandle

Dr Heinz M. Kabutz

Last Updated: 2020-11-24



Javaspecialists.eu
java training

© 2020 - Heinz Max Kabutz - All Rights Reserved

Thread Safety with Phaser, StampedLock and VarHandle

← **Tweet**



Heinz Kabutz
@heinzkabutz

← **For the easily amused** ✓

In **#Java**, which of these mechanisms of waiting is compatible with fibers from Project Loom?

`wait();`

24%

`Thread.sleep(...);`

18%

`Thread.onSpinWait();`

39%

`for(;;);`

18%

98 votes · 21 hours left

Thread Safety with Phaser, StampedLock and VarHandle

Phaser



Javaspecialists.eu
java training

Phasers

- **Allows threads to coordinate by phases**
 - More flexible than `CountDownLatch` and `CyclicBarrier`
- **Registration**
 - Number of parties *registered* may vary over time
 - Same as *count* in `CountDownLatch`, *parties* in `CyclicBarrier`
 - A party can register/deregister itself at any time
- **ManagedBlocker**
 - Can be used in the `ForkJoinPool`

Thread Safety with Phaser, StampedLock and VarHandle

Demo of Cojoining Approaches

github.com/kabutz/modern-synchronizers
branch talks-20-11-26_JokerConf



Javaspecialists.eu
java training

Tiered Phasers

- **Tree of phasers can reduce contention**
- **A bit complicated to understand (at least for me)**
 - **Parent does not know what children it has**
 - **When a child is added, parent # parties increases by 1**
 - **If child's registered parties > 0**
 - **Once child arrived parties == 0, one party automatically arrives at parent**
 - **With `arriveAndAwaitAdvance()`, we wait for all parties in tree**
 - **Thus the parties in the current phaser and in the parent have to arrive**

Tiered Phasers

- **Parent parties incremented when child has parties**

```
Phaser root = new Phaser(3);  
Phaser c1 = new Phaser(root, 4);  
Phaser c2 = new Phaser(root, 5);  
Phaser c3 = new Phaser(c2, 0);  
System.out.println(c3);  
System.out.println(c2);  
System.out.println(c1);  
System.out.println(root);
```

- **outputs**

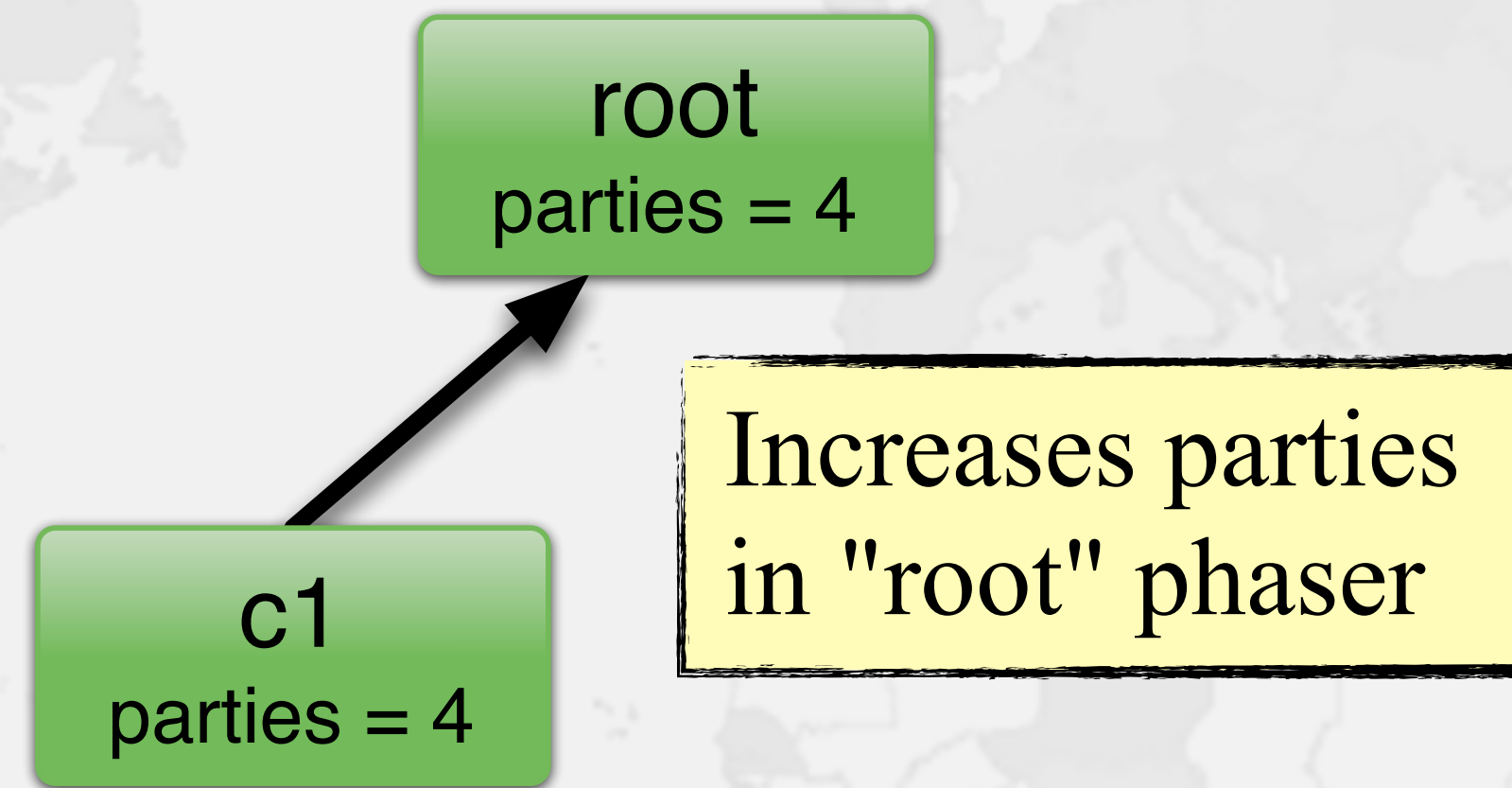
```
j.u.c.Phaser [phase = 0 parties = 0 arrived = 0] (c3)  
j.u.c.Phaser [phase = 0 parties = 5 arrived = 0] (c2)  
j.u.c.Phaser [phase = 0 parties = 4 arrived = 0] (c1)  
j.u.c.Phaser [phase = 0 parties = 5 arrived = 0] (root)
```

Phaser "root" is Created With 3 Parties

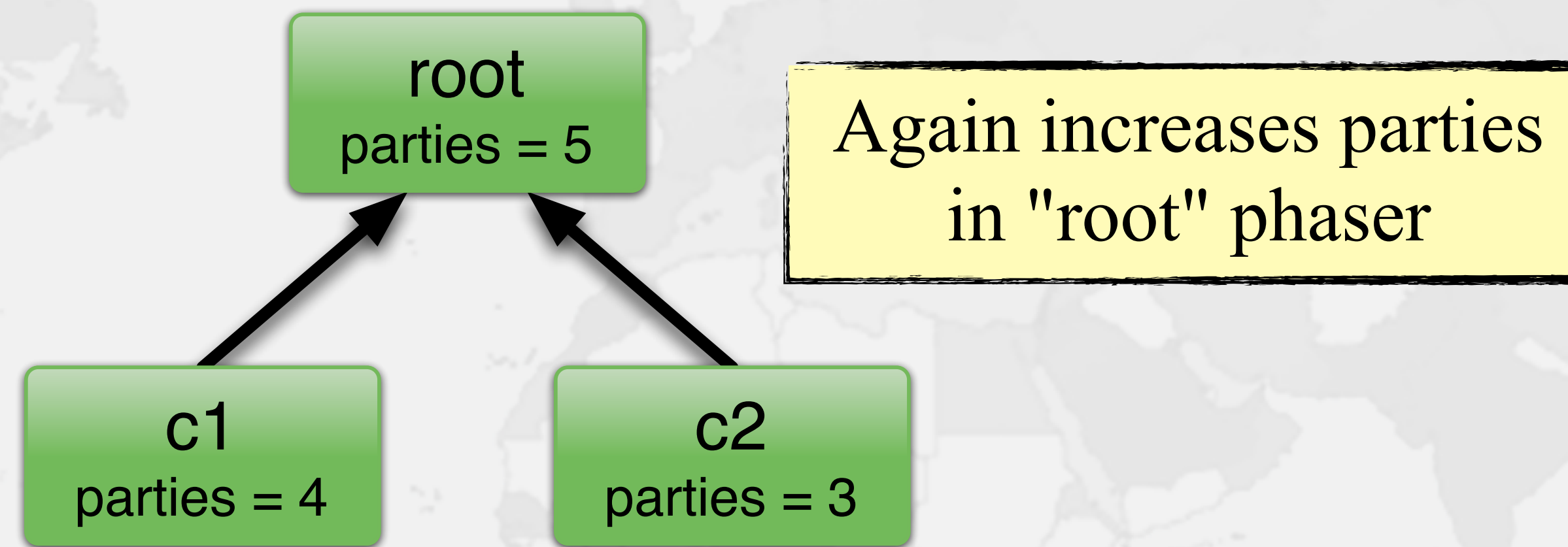


root
parties = 3

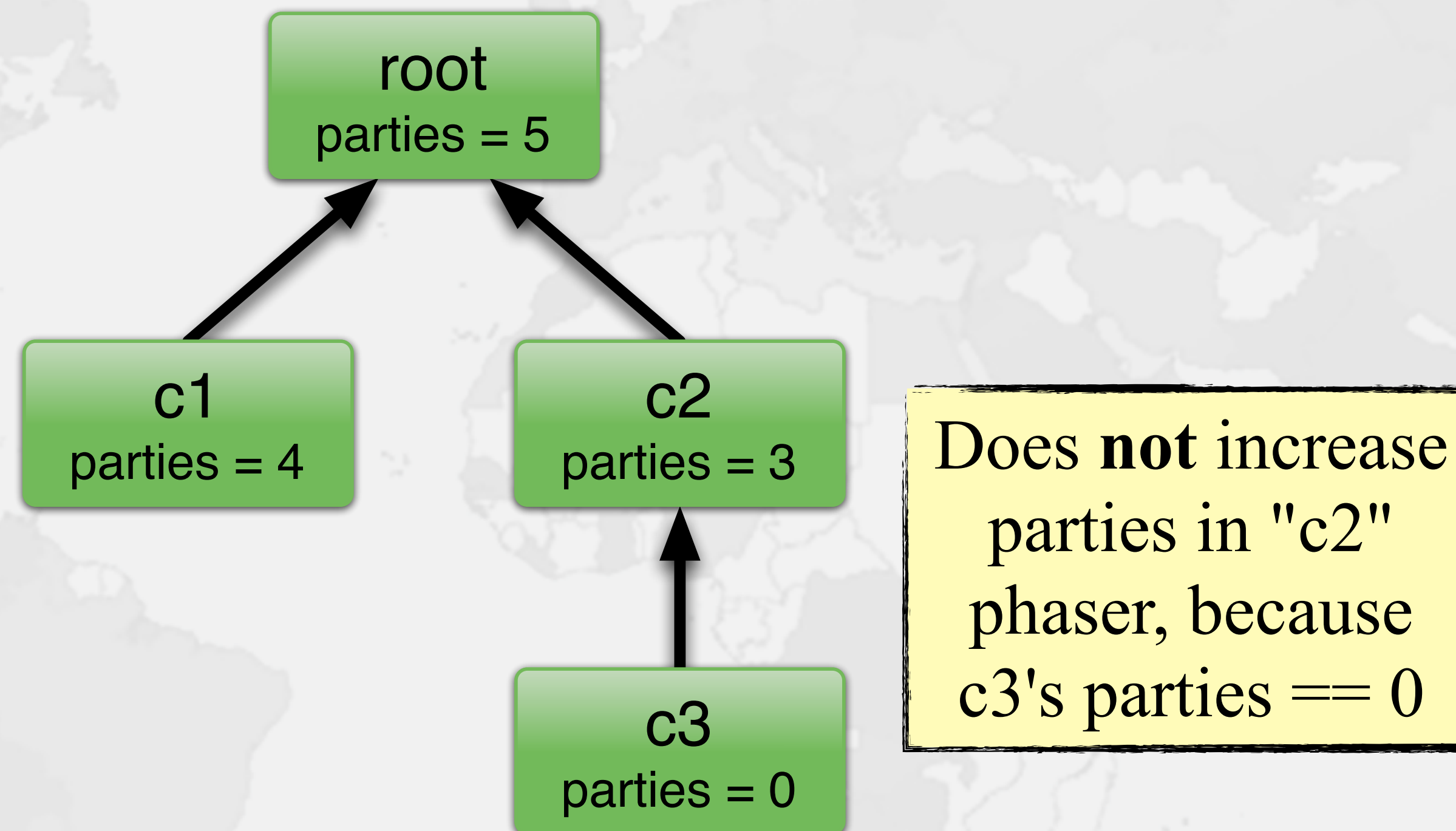
Phaser "c1" is Created With 4 Parties



Phaser "c2" is created with 3 parties



Phaser "c3" is created with 0 parties



ManagedBlocker

- **ForkJoinPool makes more threads when blocked**
 - ForkJoinPool is configured with desired parallelism
- **Uses in the JDK**
 - Java 7: Phaser
 - Java 8: CompletableFuture
 - Java 9: Process, SubmissionPublisher
 - Java 14: AbstractQueuedSynchronizer
 - ReentrantLock, ReentrantReadWriteLock, CountdownLatch, Semaphore
 - Loom: LinkedTransferQueue, SynchronousQueue, SelectorImpl

For All You Wonderful Programmers

- **Daily JGym.IO 8 minute Java microlesson**
 - 8 am UTC
 - <https://jgym.io/jokerconf2020>



StampedLock



What is StampedLock?

- **Java 8 synchronizer**
- **Allows optimistic reads**
 - **ReentrantReadWriteLock only has pessimistic reads**
- **Not reentrant**
 - **This is not a feature**
- **Use to enforce invariants across multiple fields**
 - **For simple classes, synchronized/volatile is easier and faster**
- **Can split locking and unlocking between threads**



Pessimistic Exclusive Lock (write)

```
public class StampedLock {  
    long writeLock() // never returns 0, might block  
  
    // new write stamp if successful; otherwise 0  
    long tryConvertToWriteLock(long stamp)  
  
    void unlockWrite(long stamp) // needs write stamp  
  
    // and a bunch of other methods left out for brevity
```


Pessimistic Non-Exclusive Lock (read)

```
public class StampedLock { // continued ...  
    long readLock() // never returns 0, might block  
  
    // new read stamp if successful; otherwise 0  
    long tryConvertToReadLock(long stamp)  
  
    void unlockRead(long stamp) // needs read stamp  
  
    void unlock(long stamp) // unlocks read or write
```

Optimistic Non-Exclusive Read (No Lock)

```
public class StampedLock { // continued ...  
    // could return 0 if a write stamp has been issued  
    long tryOptimisticRead()  
  
    // return true if stamp was non-zero and no write  
    // lock has been requested by another thread since  
    // the call to tryOptimisticRead()  
    boolean validate(long stamp)
```


Code Idiom for Optimistic Read

```
public double optimisticRead() {  
    long stamp = sl.tryOptimisticRead();  
    double currentState1 = state1,  
           currentState2 = state2, ... etc.;  
    if (!sl.validate(stamp)) {  
        stamp = sl.readLock();  
        try {  
            currentState1 = state1;  
            currentState2 = state2, ... etc.;  
        } finally {  
            sl.unlockRead(stamp);  
        }  
    }  
    return process(currentState1, currentState2);  
}
```

Code Idiom for Optimistic Read

```
public double optimisticRead() {  
    long stamp = sl.tryOptimisticRead();  
    double currentState1 = state1,  
           currentState2 = state2, ... et  
    if (!sl.validate(stamp)) {  
        stamp = sl.readLock();  
        try {  
            currentState1 = state1;  
            currentState2 = state2, ... etc.;  
        } finally {  
            sl.unlockRead(stamp);  
        }  
    }  
    return process(currentState1, currentState2);  
}
```

We get a stamp to use for the optimistic read

Code Idiom for Optimistic Read

```
public double optimisticRead() {  
    long stamp = sl.tryOptimisticRead();  
    double currentState1 = state1,  
           currentState2 = state2, ... etc.;  
    if (!sl.validate(stamp)) {  
        stamp = sl.readLock();  
        try {  
            currentState1 = state1;  
            currentState2 = state2, ... etc.;  
        } finally {  
            sl.unlockRead(stamp);  
        }  
    }  
    return process(currentState1, currentState2);  
}
```

We read field values into local fields

Code Idiom for Optimistic Read

```
public double optimisticRead() {  
    long stamp = sl.tryOptimisticRead();  
    double currentState1 = state1,  
           currentState2 = state2, ... etc.;  
    if (!sl.validate(stamp)) {  
        stamp = sl.readLock();  
        try {  
            currentState1 = state1;  
            currentState2 = state2;  
        } finally {  
            sl.unlockRead(stamp);  
        }  
    }  
    return process(currentState1, currentState2);  
}
```

Next we validate
that no write locks
have been issued
in the meanwhile

Code Idiom for Optimistic Read

```
public double optimisticRead() {  
    long stamp = sl.tryOptimisticRead();  
    double currentState1 = state1;  
    double currentState2 = state2;  
    if (!sl.validate(stamp)) {  
        stamp = sl.readLock();  
        try {  
            currentState1 = state1;  
            currentState2 = state2, ... etc.;  
        } finally {  
            sl.unlockRead(stamp);  
        }  
    }  
    return process(currentState1,  
    ...  
    );  
}
```

If they have,
then we don't
know if our
state is clean

Thus we acquire a
pessimistic read
lock and read the
state into local
fields

Code Idiom for Optimistic Read

```
public double optimisticRead() {  
    long stamp = sl.tryOptimisticRead();  
    double currentState1 = state1,  
           currentState2 = state2, ... etc.;  
    if (!sl.validate(stamp)) {  
        stamp = sl.readLock();  
        try {  
            currentState1 = state1;  
            currentState2 = state2, ... etc.;  
        } finally {  
            sl.unlockRead(stamp);  
        }  
    }  
    return process(currentState1, currentState2);  
}
```

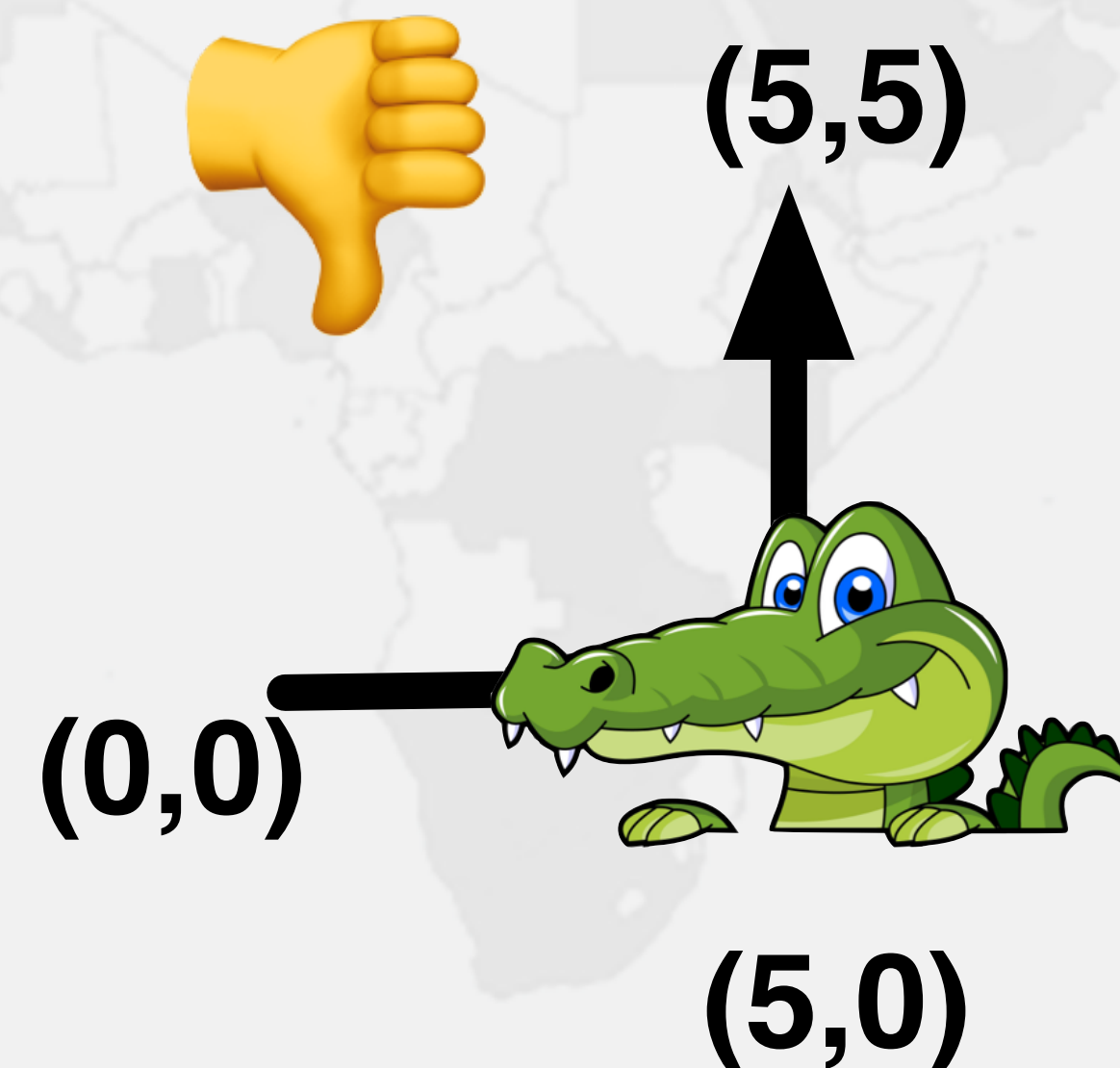
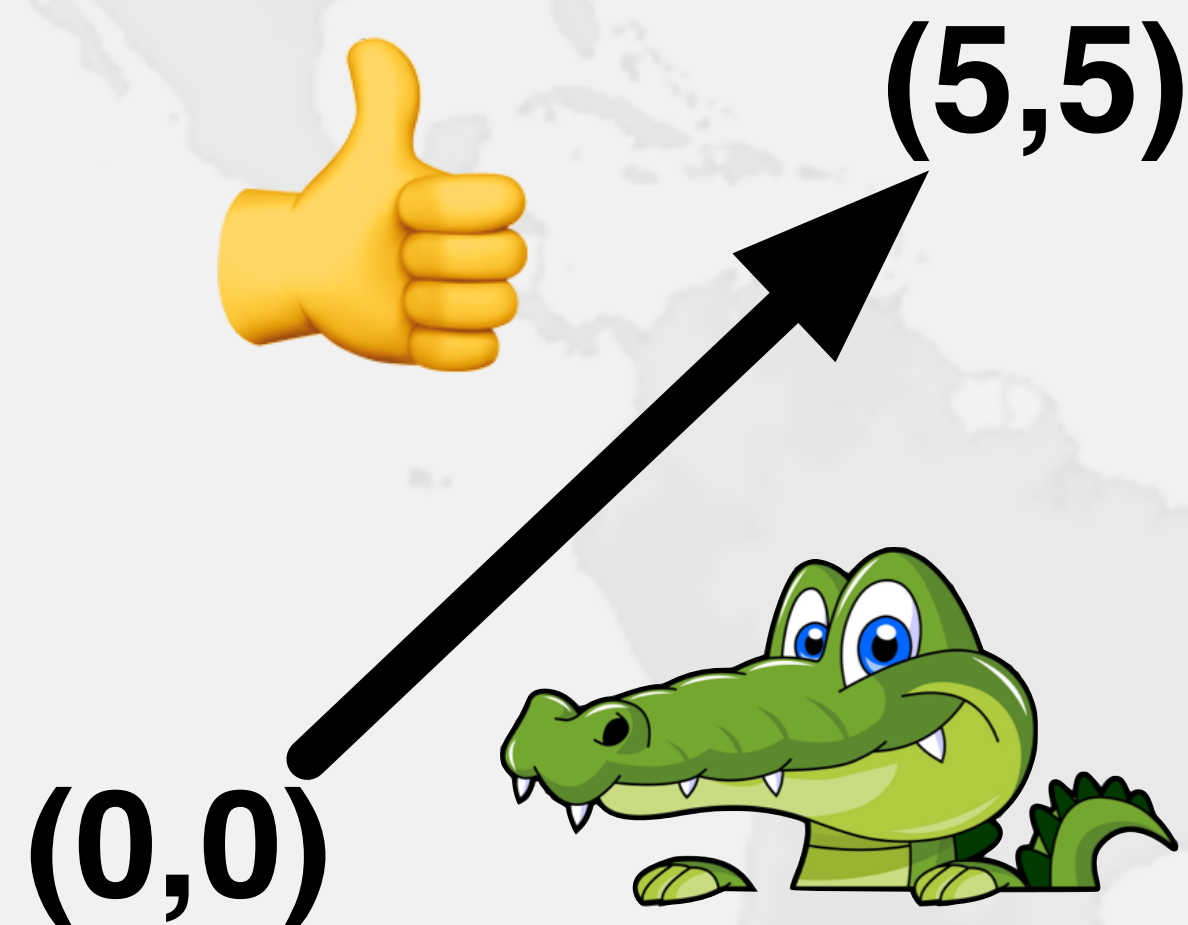

Sifis the Cretan Crocodile (RIP)

- **Poor critter was roaming around Crete**
 - Pet grew too big
 - Or hungry
- **Eventually died in our cold winter months**



Introducing the Position Class

- When moving from $(0,0)$ to $(5,5)$, we want to travel in a diagonal line
 - Don't want to ever see our position at $(0,5)$ or especially $(5,0)$



Thread Safety with Phaser, StampedLock and VarHandle

Refactoring Position

github.com/kabutz/modern-synchronizers
branch talks-20-11-26_JokerConf



Javaspecialists.eu
java training

Newer Idiom for Optimistic Read

```
public double distanceFromOrigin() {
    long stamp = sl.tryOptimisticRead();
    try {
        retryHoldingLock: for (;;) stamp = sl.readLock() {
            if (stamp == 0L) continue retryHoldingLock;
            // possibly racy reads
            double currentState1 = state1;
            double currentState2 = state2; // etc.
            if (!sl.validate(stamp))
                continue retryHoldingLock;
            return process(currentState1, currentState2);
        }
    } finally {
        if (StampedLock.isReadLockStamp(stamp))
            sl.unlockRead(stamp);
    }
}
```


Truly Optimistic, Optimistic Read

```
public double distanceFromOrigin() {
    long stamp = sl.tryOptimisticRead();
    try {
        retryHoldingLock: for (;;) stamp = sl.readLock() {
            if (stamp == 0L) continue retryHoldingLock;
            // possibly racy reads
            double currentState1 = state1;
            double currentState2 = state2; // etc.
            if (!sl.validate(stamp))
                continue retryHoldingLock;
            return process(currentState1, currentState2);
        }
    } finally {
        if (StampedLock.isReadLockStamp(stamp))
            sl.unlockRead(stamp);
    }
}
```

Truly Optimistic, Optimistic Read

```
public double distanceFromOrigin() {
    long stamp = sl.tryOptimisticRead();
    try {
        retryHoldingLock: for (;;) stamp = sl.readLock() {
            // possibly racy reads
            double currentState1 = state1;
            double currentState2 = state2; // etc.
            if (!sl.validate(stamp))
                continue retryHoldingLock;
            return process(currentState1, currentState2);
        }
    } finally {
        if (StampedLock.isReadLockStamp(stamp))
            sl.unlockRead(stamp);
    }
}
```


Thread Safety with Phaser, StampedLock and VarHandle

Refactoring Position x 2

**github.com/kabutz/modern-synchronizers
branch talks-20-11-26_JokerConf**



Javaspecialists.eu
java training

Thread Safety with Phaser, StampedLock and VarHandle

VarHandle

Making your application run even faster!



Javaspecialists.eu
java training

Java 9 VarHandles Instead of Unsafe

- **VarHandles remove biggest temptation for Unsafe**
 - As fast as Unsafe
 - Make sure VarHandle fields are static final
- **Can read and write fields of class**
 - `getVolatile() / setVolatile()`
 - `getAcquire() / setRelease()`
 - `getOpaque() / setOpaque()`
 - `get() / set()` - plain
 - `compareAndSet()`, returning boolean
 - `compareAndExchange()`, returning found value

Thread Safety with Phaser, StampedLock and VarHandle

Refactoring Position to VarHandle

github.com/kabutz/modern-synchronizers
branch talks-20-11-26_JokerConf



Javaspecialists.eu
java training

compareAndExchange()

- Direct support for real compare-and-swap
 - Before it was compare-and-set
- Supported by Atomic classes and VarHandles
- Eliminates one volatile read - might be faster

```
public void move(int deltaX, int deltaY) {  
    int[] current, next = new int[2], swapResult = xy;  
    do {  
        current = swapResult;  
        next[0] = current[0] + deltaX;  
        next[1] = current[1] + deltaY;  
    }  
    while ((swapResult = (int[]) XY.compareAndExchange(  
        this, current, next)) != current);  
}
```


Thread Safety with Phaser, StampedLock and VarHandle

Refactoring Position to VarHandle x 2

github.com/kabutz/modern-synchronizers

branch talks-20-11-26_JokerConf



Javaspecialists.eu
java training

Question Time

- Remember: <https://jgym.io/jokerconf2020>
- Twitter: @heinzkabutz
- Newsletter: <https://www.javaspecialists.eu>
- Email: heinz@javaspecialists.eu

