ververica

Understanding Java Memory Model

A Journey from Hardware Guarantees to Happens-Before Edges

Muhammet Orazov

Muhammet Orazov

Software engineer experienced in database, distributed systems and starting streaming journey

- github.com/morazow
- twitter.com/morazow
- muhammet@ververica.com

https://www.ververica.academy

Agenda

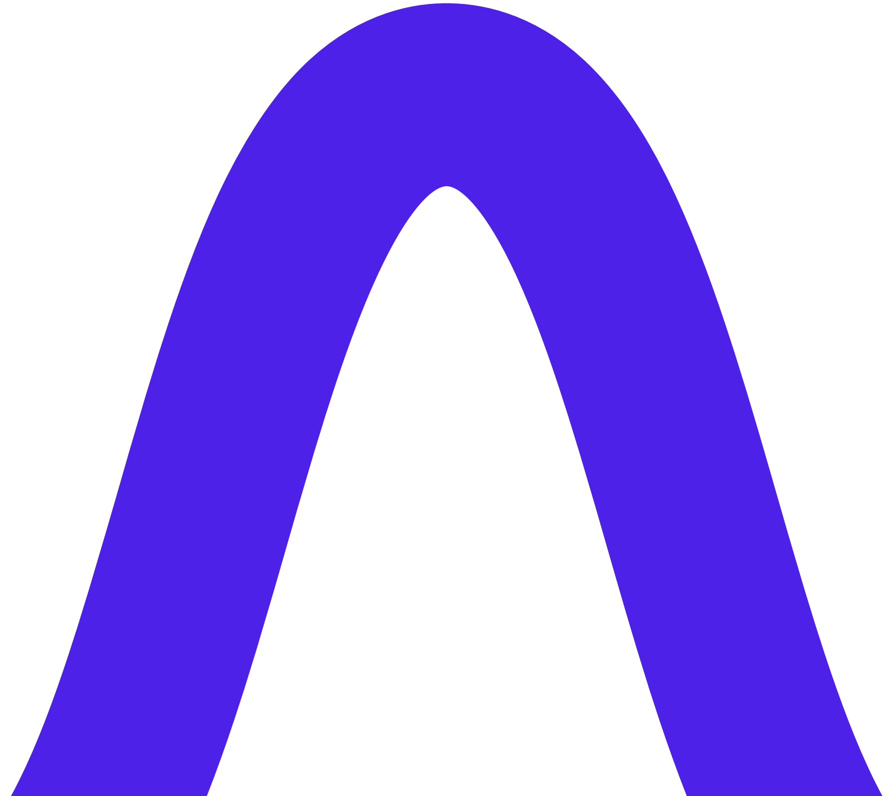
01 Memory Models

02 Hardware Guarantees

03 Java Memory Model

04 Litmus Tests

05 Conclusion



Memory Models

Thread I

```
x = 1;
done = 1;
```

Thread II

```
while (done == 0) { /** **/ }
System.out.println(x);
```

- Variables are set to zero initially
- Both threads run on own dedicated processor
- Can we reason about the output of the program?

IT DEPENDS!

Memory Models: Programming Languages

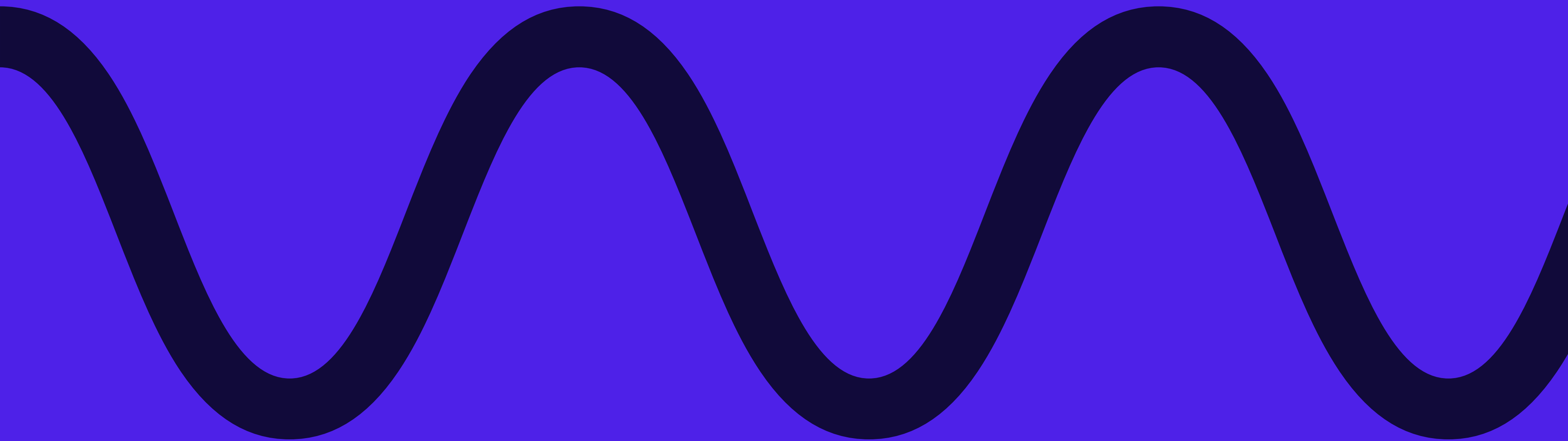
- Describe semantics of multithreaded programs
- Visibility rules of shared memory read updated by multiple writes
- Consistency of changes on a shared memory

ververica | © 2024

Memory Models: Programming Languages

*"These semantics do not prescribe how a multithreaded program should be executed. Rather, they describe the **behaviors** that multithreaded programs are **allowed to exhibit**. Any execution strategy that generates only **allowed behaviors is an acceptable execution strategy**"*

From *Java Language Specification*, Chapter 17



Hardware Guarantees

Sequential Consistency

- Operations of all threads are executed in ***some sequential order***
- Operations of each individual thread ***appear in order specified by*** *its program*

“A multithreaded program satisfying this condition will be called sequential consistent.”

- Leslie Lamport [1]

[1]: How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs

Sequential Consistency

Thread I

```
x = 1;
y = 1;
```

Thread II

```
r1 = y;
r2 = x;
```

Accepted outcomes:

```
x = 1;
y = 1;
                r1 = y;  // 1
                r2 = x; // 1
```

```
r1 = y;  // 0
r2 = x; // 0
                x = 1;
                y = 1;
```

```
x = 1;
                r1 = y;  // 0
y = 1;
                r2 = x; // 1
```

```
x = 1;
                r1 = y; // 0
                r2 = x; // 1
y = 1;
```

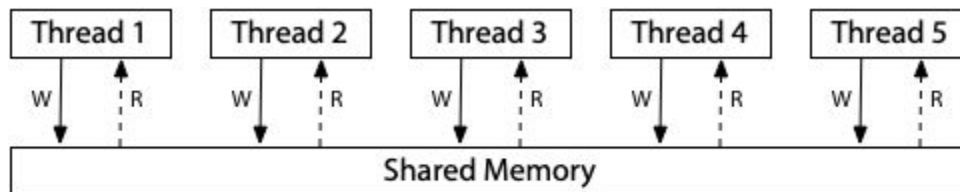
```
r1 = y;  // 0
                x = 1;
                y = 1;
r2 = x; // 1
```

```
                r1 = y; // 0
x = 1;
                r2 = x; // 1
y = 1;
```

r1 = 1, r2 = 0 is not acceptable

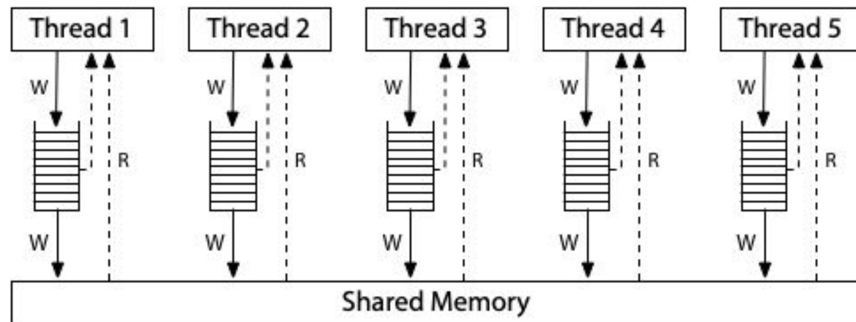
ververica | © 2024

Sequential Consistency



Sequential Consistent Hardware Model. Image © Russ Cox - Hardware Memory Models.

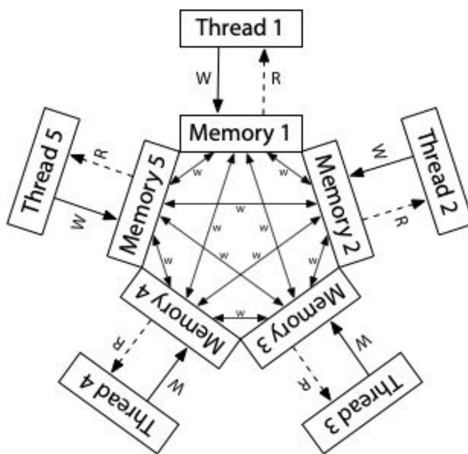
x86 Total Store Order (TSO)



x85 Architecture. Image © Russ Cox - Hardware Memory Models.

- Each write is queued in first-in, first-out (FIFO) order
- Local queue is flushed in FIFO order, keeping program order of writes
- Reads check local queue first, then shared memory

ARM Relaxed Memory Model



ARM Architecture. *Image © Russ Cox - Hardware Memory Models.*

- Each processor has its own copy of memory
- Writes propagated independently, with allowed reordering
- Reads could be delayed until needed or until after a write

ververica | © 2024

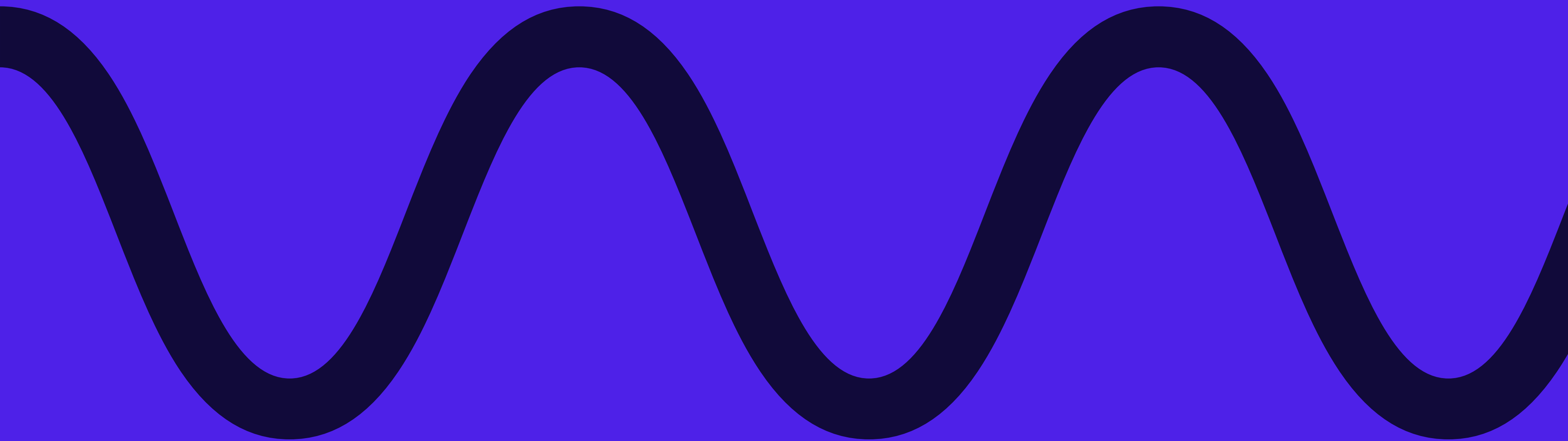
Data Race Free Sequential Consistency (DRF-SC)

- Introduced in 1990 by Sarita Adve and Mark Hill
- Assumes distinct hardware synchronization operations
- Reads and writes can be **reordered** between these synchronization operations
- But, reads and writes **must not be moved** across synchronization operations

A program is **data-race-free** if any two accesses to same memory location from two threads are either both reads or are separated by a synchronization operation that forces one to ***happen before*** the other.

An agreement between hardware and software!

ververica | © 2024



Java Memory Model

Java Synchronization Instructions

Examples of **synchronization** instructions:

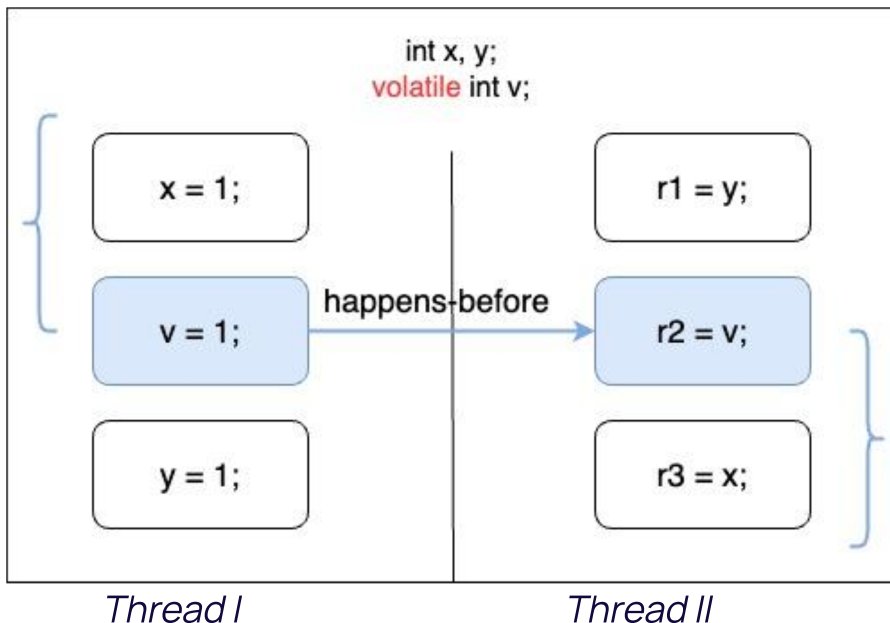
- Unlock of mutex **m** happens before any following lock of **m**
- Write to volatile variable **v** happens before any following read of **v**

These establish a ***happens-before*** relationship between multi-threaded code executions.

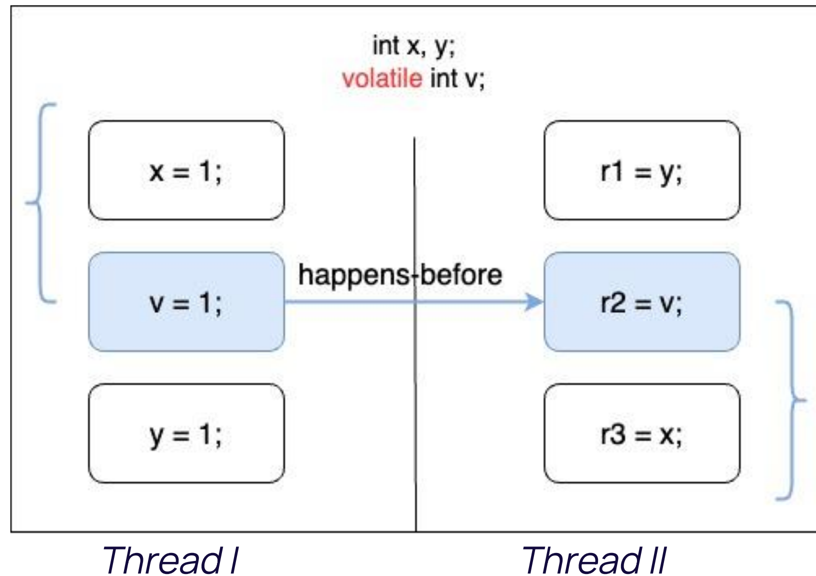
These edges help us write data-race free programs.

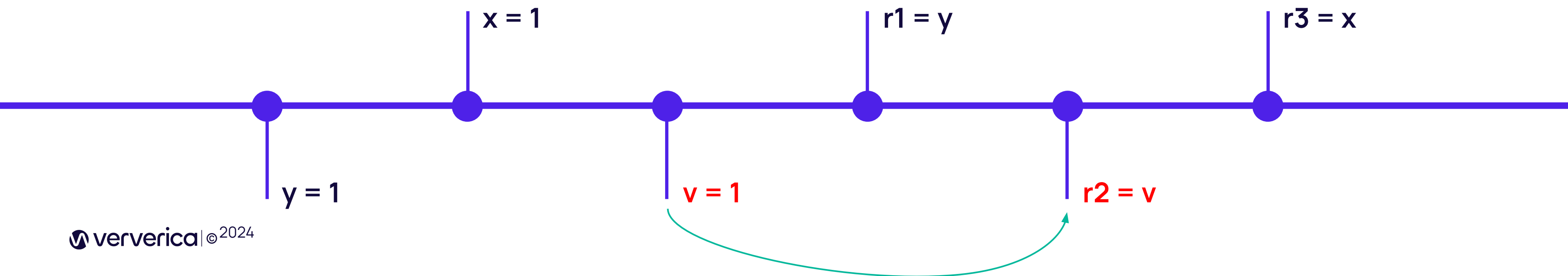
Happens Before

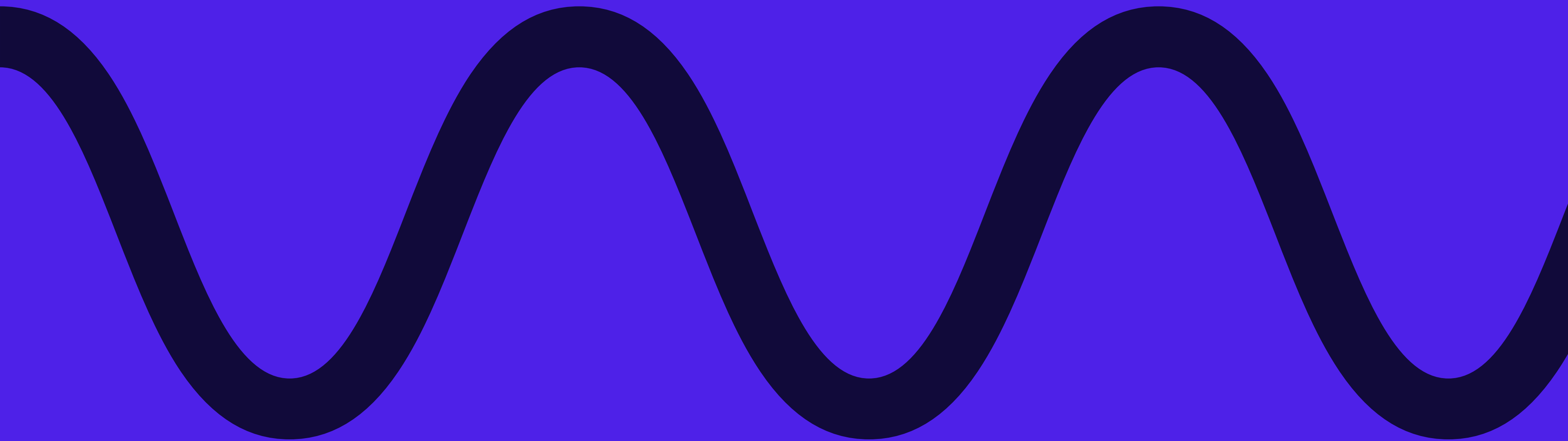
The **happens-before** edges synchronize rest of programs, ordering plain operations across threads



The **happens-before** edges synchronize rest of programs, ordering plain operations across threads







Litmus Tests

… verified with Java Concurrency Stress (JCStress) framework

Litmus Tests: Message Passing

Can we observe the output *r1 = 1, r2 = 0* ?

```
x = 1;

y = 1;
```

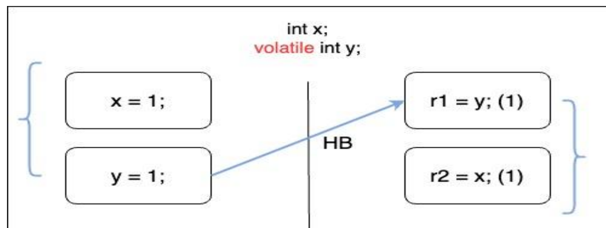
```
r1 = y;

r2 = x;
```

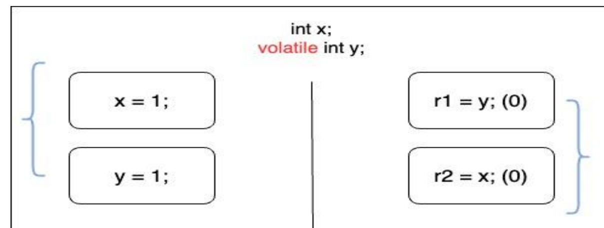
- Sequentially consistent hardware: no
- x86 TSO: no
- ARM: yes
- Java, plain variables: yes
- Java, volatile y: no

Litmus Tests: Message Passing

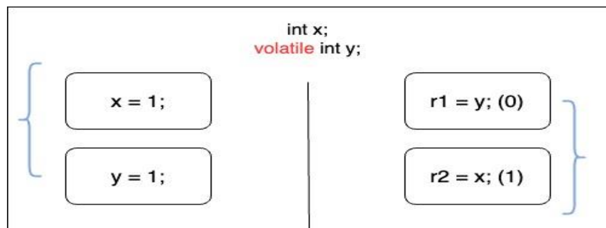
… happens-before edges



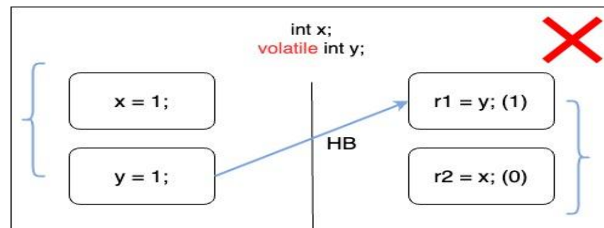
HB consistent, reads observe latest writes on the happens-before edge



HB consistent, reads observe the initial values



HB consistent, it is racy read. There is not happens-before edge between write on x and read on x. It is read via ***race****.*
HB allows observing "unsynchronized" writes via race.



HB inconsistent. We cannot use this particular execution to reason about program outcomes. This outcome is not accepted.

ververica | © 2024

Litmus Tests: Message Passing

```
$ mvn clean package
$ java -jar target/jmm-litmus-tests-1.0.0-assembly.jar -v -t ".*MessagePassingVolatile.*"
```

```
.......... [OK] com.morazow.jmm.LitmusTests.MessagePassingVolatile

  Results across all configurations:

  RESULT    SAMPLES          FREQ      EXPECT        DESCRIPTION
  0, 0      3.213.262.026    40,30%    Acceptable    Thr1 runs after Thr2 finishes.
  0, 1      8.206.437        0,10%     Acceptable    Thr2 is interleaved between writes.
  1, 0      0                0,00%     Forbidden     Observed y write but not x.
  1, 1      4.752.066.525    59,60%    Acceptable    Thr2 runs after Thr1 finishes.
```

Litmus Tests: Store Buffering

Can we observe the output *r1 = 0, r2 = 0* ?

```
x = 1;

r1 = y;
```

```
y = 1;

r2 = x;
```

- Sequentially consistent hardware: no
- x86 TSO: yes
- ARM: yes
- Java, plain variables: yes
- Java, volatile x **and** y: no

Litmus Tests: Load Buffering

Can we observe the output *r1 = 1, r2 = 1*?

```
r1 = x;

y = 1;
```

```
r2 = y;

x = 1;
```

- Sequentially consistent hardware: no
- x86 TSO: no
- ARM: yes
- Java, plain variables: yes
- Java, volatile x **or** y: no

Litmus Tests: Coherence

Can we observe the output *r1 = 1, r2 = 2, r3 = 2, r4 = 1*?

```
x = 1;
```

```
x = 2;
```

```
r1 = x;
r2 = x;
```

```
r3 = x;
r4 = x;
```

- Sequentially consistent hardware: no
- x86 TSO: no
- ARM: no
- Java, plain variables: yes
- Java, volatile x: no

```
.......... [OK] com.morazow.jmm.LitmusTests.CoherencePlain

Results across all configurations:

RESULT      SAMPLES    FREQ     EXPECT       DESCRIPTION
1, 2, 2, 1  3.143      <0,01%  Interesting  Thrs see different order of writes on single memory lo...
1, 2, 2, 2  945.661    <0,01%  Acceptable   Reads observer total order of writes on a single memory l...
2, 0, 0, 0  1.372.493  <0,01%  Acceptable   Reads observer total order of writes on a single memory l...
2, 1, 1, 2  3.041      <0,01%  Interesting  Threads see different order of writes on single memory lo...
...
```

Litmus Tests: Independent Reads of Independent Writes

Can this program output *r1 = 1, r2 = 0, r3 = 0, r4 = 1*?

```
x = 1;
```

```
y = 1;
```

```
r1 = x;
r2 = y;
```

```
r3 = y;
r4 = x;
```

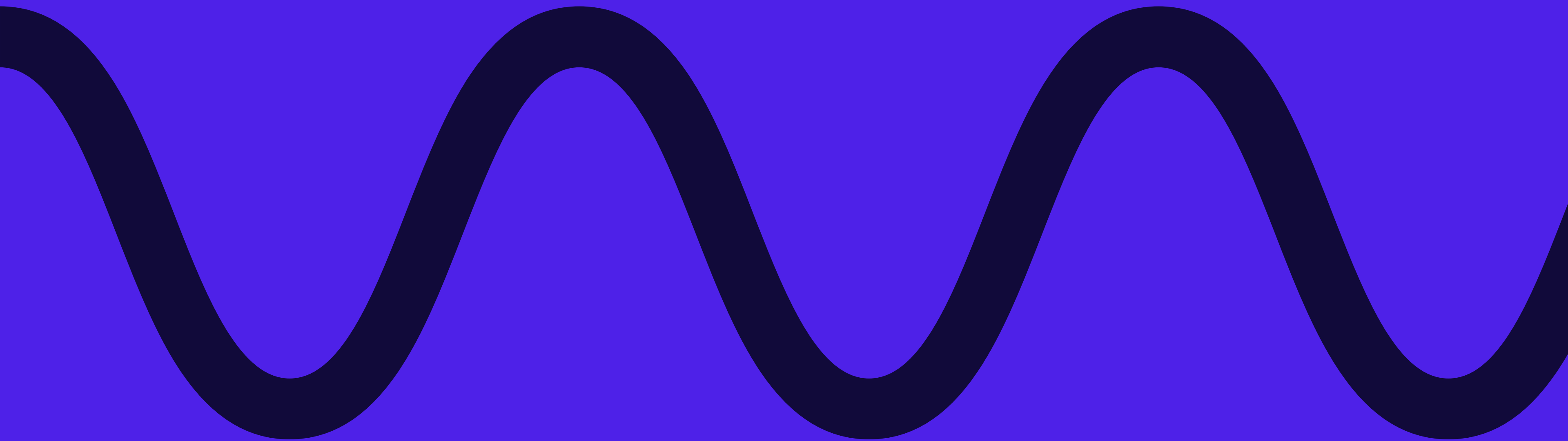
- Sequentially consistent hardware: no
- x86 TSO: no
- ARM: yes
- Java, plain variables: yes
- Java, volatile x **and** y: no

ververica | © 2024

JCStress Tests

All of the litmus tests are validated using the JCStress framework.

You can find the GitHub repository by scanning the QR code or at github.com/morazow/jmm-litmus-tests



Conclusion

Conclusion

We learned about guarantees offered by the different hardware models.

We learned that using proper synchronization mechanisms helps to ensure data-race-free programs.

We looked into the Java Memory Model and learned how happens-before edges are established.

We ran several litmus tests to better understand how different models behave.

References

Russ Cox, *https://research.swtch.com/mm — Memory Models*

Aleksey Shipilëv, *https://shipilev.net/blog/2014/jmm-pragmatics/ — JMM Pragmatics*

Aleksey Shipilëv, *https://shipilev.net/blog/2016/close-encounters-of-jmm-kind/ — Close Encounters of JMM Kind*

Hydra Conference 2021, JCStress Workshop by Aleksey Shipilev

Apache Flink® & Everything Streaming Data

Berlin 2024

Registration Opens Soon!

Organized by

#flinkforward

ververica

Thanks!

Q/A