Java Memory Model

“Was Entwickler *wirklich* darüber wissen sollten”

Abstract

- **Multi-Core:** Performance durch Parallelität
- **Java Memory Model:** Threads und Speicher
- **Einfache Entwicklung:** Effizienz und Effektivität

Disclaimer

Die Inhalte stammen aus verschiedenen Quellen und wurden von mir zusammengestellt. Die Quellen sind am Ende angegeben. Fehler stammen von mir.

Agenda

1. Quiz
2. “Prinzip”
3. Beispiele
4. Exkurs

Quiz

5 Multiple Choice Fragen

Frage 1: “synchronized”

Initialisierung

```
Object lock = new Object();
int x = 0; int y = 0;
int r = 0; int s = 0;
```

Thread 1

```
synchronized (lock) {
    x = 1;
    if (y == 0)
        r = 1;
}
```

Thread 2

```
synchronized (lock) {
    y = 1;
    if (x == 0)
        s = 1;
}
```

Mögliche Ergebnisse

Antwort	r:0 s:0	r:0 s:1	r:1 s:0	r:1 s:1
A	✘	✘	✔	✘
B	✘	✔	✔	✘
C	✔	✔	✔	✘
D	✔	✔	✔	✔

Frage 2: “non-synchronized”

Initialisierung

```
int x = 0; int y = 0;
int r = 0; int s = 0;
```

Thread 1

```
x = 1;
if (y == 0)
    r = 1;
```

Thread 2

```
y = 1;
if (x == 0)
    s = 1;
```

Mögliche Ergebnisse

Antwort	r:0 s:0	r:0 s:1	r:1 s:0	r:1 s:1
A	✘	✘	✔	✘
B	✘	✔	✔	✘
C	✔	✔	✔	✘
D	✔	✔	✔	✔

Frage 3: “volatile”

Initialisierung

```
volatile int x = 0; volatile int y = 0;
int r = 0; int s = 0;
```

Thread 1

```
x = 1;
if (y == 0)
    r = 1;
```

Thread 2

```
y = 1;
if (x == 0)
    s = 1;
```

Mögliche Ergebnisse

Antwort	r:0 s:0	r:0 s:1	r:1 s:0	r:1 s:1
A	✗	✗	✓	✗
B	✗	✓	✓	✗
C	✓	✓	✓	✗
D	✓	✓	✓	✓

Frage 4: “speculative”

Initialisierung

```
int x = 0; int y = 0;
int r = 0; int s = 0;
```

Thread 1

```
if (y != 0)
    x = 1;
if (x == y)
    r = 1;
```

Thread 2

```
if (x != 0)
    y = 1;
if (x != y)
    s = 1;
```

Mögliche Ergebnisse

Antwort	r:0 s:0	r:0 s:1	r:1 s:0	r:1 s:1
A	✘	✘	✔	✘
B	✘	✔	✔	✘
C	✔	✔	✔	✘
D	✔	✔	✔	✔

Frage 5: “independent”

Initialisierung

```
volatile int x = 0; volatile int y = 0;
int r = 0; int s = 0;
```

Thread 1

```
x = 1;
```

Thread 2

```
if (x == 1
 && y == 0)
     r = 1;
```

Thread 3

```
if (y == 1
 && x == 0)
     s = 1;
```

Thread 4

```
y = 1;
```

Mögliche Ergebnisse

Antwort	r:0 s:0	r:0 s:1	r:1 s:0	r:1 s:1
A	✗	✗	✓	✗
B	✗	✓	✓	✗
C	✓	✓	✓	✗
D	✓	✓	✓	✓

acquire

visibility

reordering

happens-before

synchronizes-with

cache refresh

publishing

atomicity

memory barrier

cache flush

race condition

release

memory fence

Sequential Consistency for Data-Race-Free Programs

Sequential Consistency for Data-Race-Free Programs

⑤ Warum ist die Quantifizierung so merkwürdig?

④ Warum taucht "sequentially consistent" hier nochmals auf?

③ Was ist eine "data race"?

"[..] If all sequentially consistent executions are free of data races, [..] then all executions of the program will appear to be sequentially consistent."

① Was bedeutet "will appear to be"?

Java® Language Specification · Java SE 8 Edition ·

② Was heißt "sequentially consistent"?

Sequential Consistency

```
for (;;) {
  var /= 2;
}
```

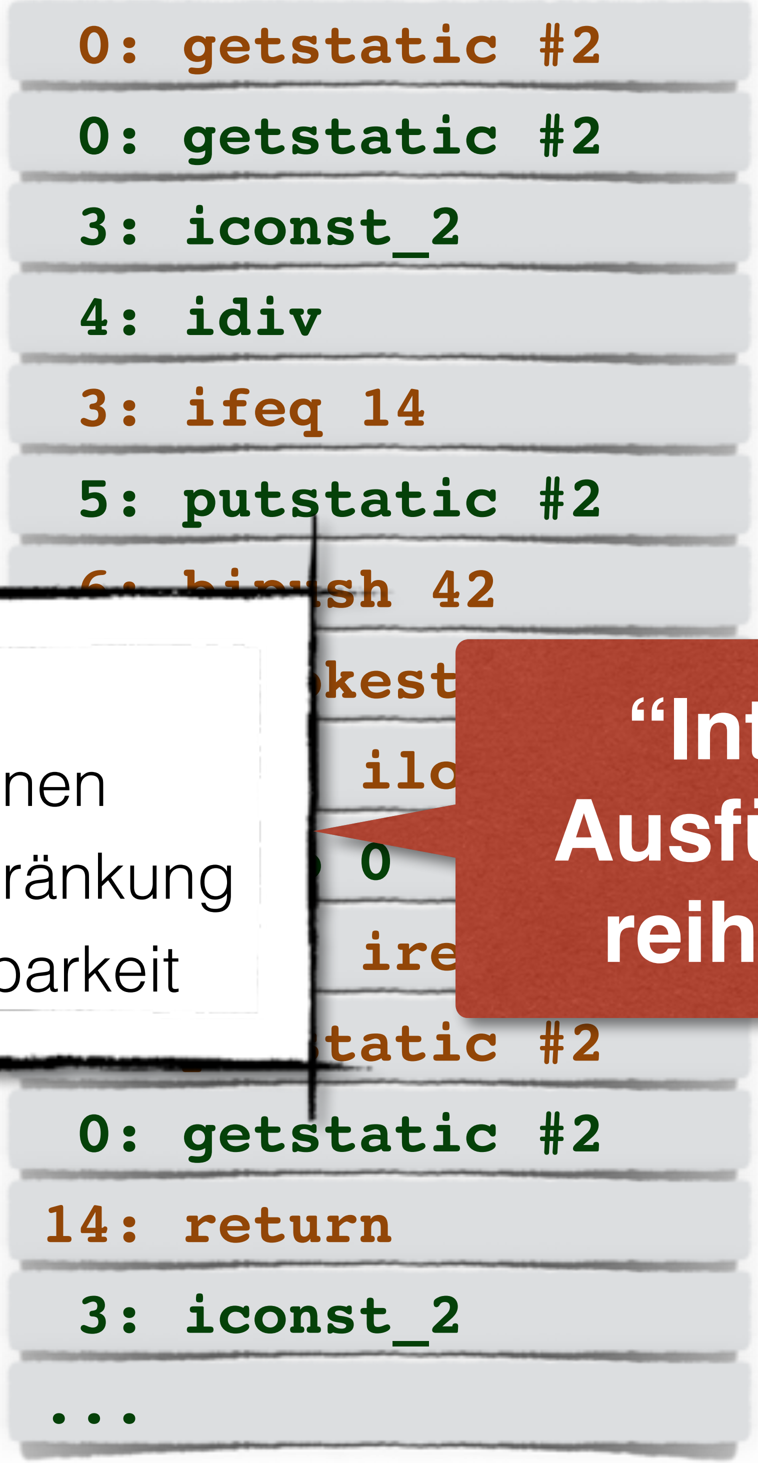
```
0: getstatic #2
3: iconst_2
4: idiv
5: putstatic #2
8: goto 0
0: getstatic #2
3: iconst_2
...
```

```
if (var != 0) {
  var = foobar(42);
}
```

```
 0: getstatic #2
 3: ifeq 14
 6: bipush 42
 8: invokestatic #3
         0: iload_0
         1: ireturn
11: putstatic #2
14: return
```

Sequential Consistency

```
for (;;) {
  var /= 2;
}
```



```
 0: getstatic #2
 0: getstatic #2
 3: iconst_2
 4: idiv
 3: ifeq 14
 5: putstatic #2
 6: bipush 42
    kest
    ilo
    0
    ire
    tatic #2
 0: getstatic #2
14: return
 3: iconst_2
...
```

```
if (var != 0) {
  var = foobar(42);
}
```

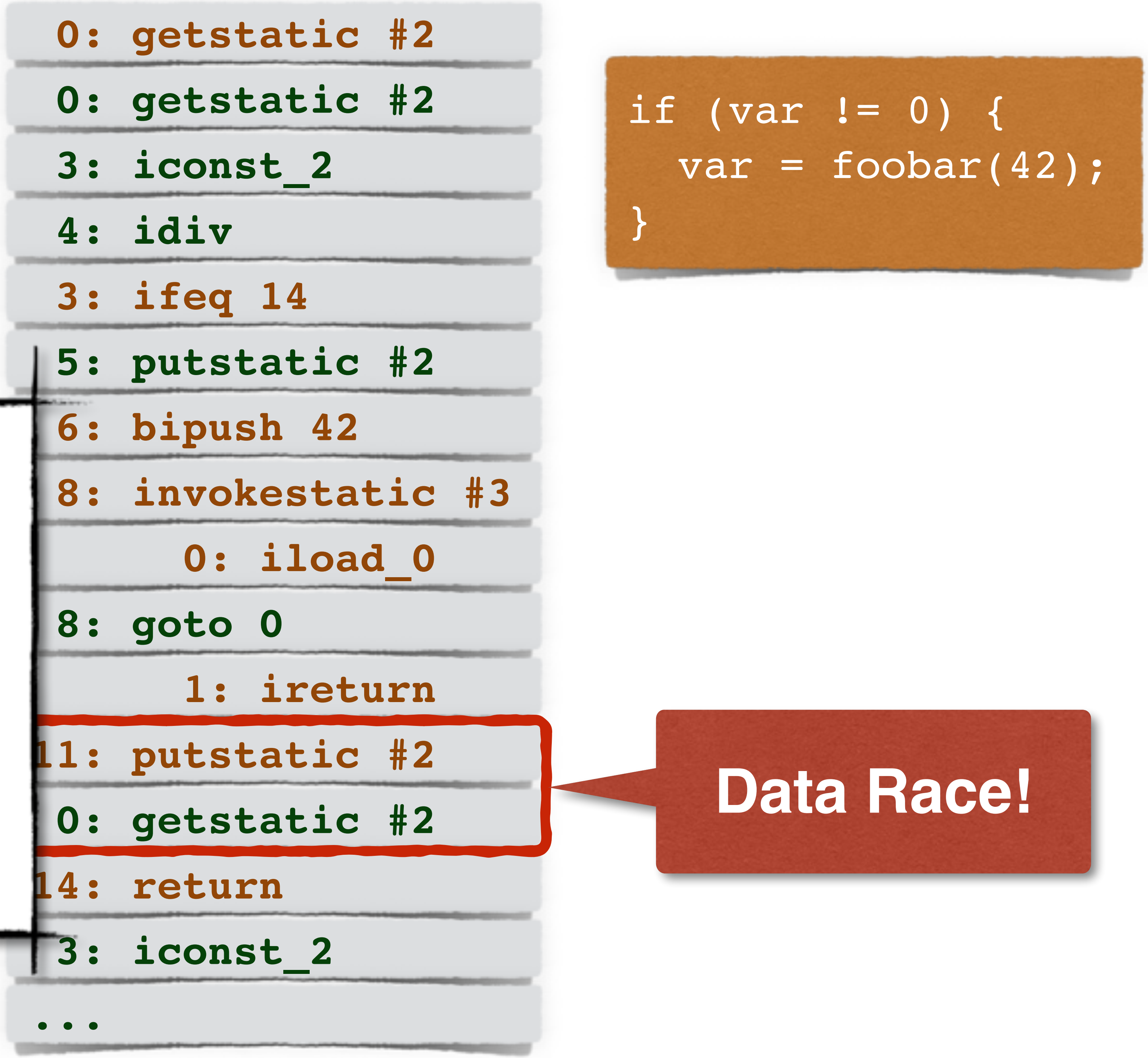
Definition:

- Basis-Operationen
- globale Verschränkung
- sofortige Sichtbarkeit

“Intuitive” Ausführungsreihenfolge

Data Race

```
for (;;) {
  var /= 2;
}
```



```
0: getstatic #2
0: getstatic #2
3: iconst_2
4: idiv
3: ifeq 14
5: putstatic #2
6: bipush 42
8: invokestatic #3
     0: iload_0
8: goto 0
     1: ireturn
11: putstatic #2
0: getstatic #2
14: return
3: iconst_2
...
```

Data Race!

```
if (var != 0) {
  var = foobar(42);
}
```

Definition:

- direkt hintereinander ausgeführte Zugriffe zweier Threads auf selbe Variable
- mindestens eine Schreiboperation
- keine volatile-Variable

Sequential Consistency for Data-Race-Free Programs

⑤ Warum ist die Quantifizierung so merkwürdig?

④ Warum taucht "sequentially consistent" hier nochmals auf?

③ Was ist eine "data race"? ✓

"[..] If all sequentially consistent executions are free of data races, [..] then all executions of the program will appear to be sequentially consistent."

① Was bedeutet "will appear to be"? ✓

Java® Language Specification · Java SE 8 Edition ·

② Was heißt "sequentially consistent"? ✓

Beispiele

Frage 1: “synchronized”

Initialisierung

```
Object lock = new Object();
int x = 0; int y = 0;
int r = 0; int s = 0;
```

Thread 1

```
synchronized (lock) {
    x = 1;
    if (y == 0)
        r = 1;
}
```

Thread 2

```
synchronized (lock) {
    y = 1;
    if (x == 0)
        s = 1;
}
```

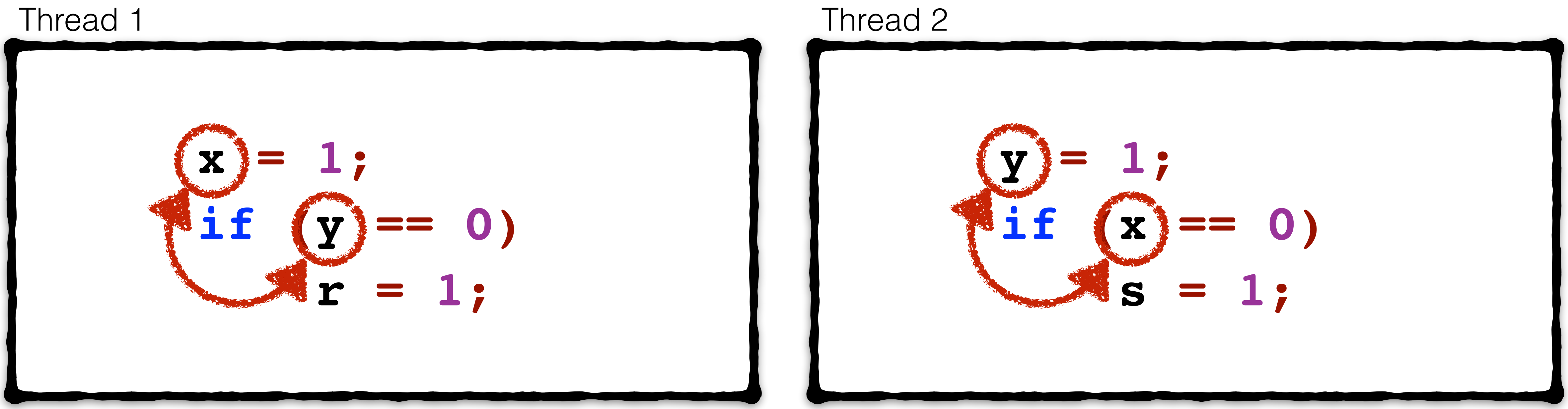
Mögliche Ergebnisse

Antwort	r:0 s:0	r:0 s:1	r:1 s:0	r:1 s:1
A	✘	✘	✔	✘
[✓ B]	✘	✔	✔	✘
C	✔	✔	✔	✘
D	✔	✔	✔	✔

Frage 2: “non-synchronized”

Initialisierung

```
int x = 0; int y = 0;
int r = 0; int s = 0;
```



Thread 1

```
x = 1;
if (y == 0)
r = 1;
```

Thread 2

```
y = 1;
if (x == 0)
s = 1;
```

Mögliche Ergebnisse

Antwort	r:0 s:0	r:0 s:1	r:1 s:0	r:1 s:1
A	✗	✗	✓	✗
B	✗	✓	✓	✗
C	✓	✓	✓	✗
[✓ D]	✓	✓	✓	✓

Frage 3: “volatile”

Initialisierung

```
volatile int x = 0; volatile int y = 0;
int r = 0; int s = 0;
```

Thread 1

```
x = 1;
if (y == 0)
    r = 1;
```

Thread 2

```
y = 1;
if (x == 0)
    s = 1;
```

Mögliche Ergebnisse

Antwort	r:0 s:0	r:0 s:1	r:1 s:0	r:1 s:1
A	✗	✗	✓	✗
B	✗	✓	✓	✗
[✓ C]	✓	✓	✓	✗
D	✓	✓	✓	✓

Frage 4: “speculative”

Initialisierung

```
int x = 0; int y = 0;
int r = 0; int s = 0;
```

Thread 1

```
if (y != 0)
    x = 1;
if (x == 1)
    r = 1;
```

Thread 2

```
if (x != 0)
    y = 1;
if (x != y)
    s = 1;
```

Dead code in sequentially consistent executions!

Mögliche Ergebnisse

Antwort	r:0 s:0	r:0 s:1	r:1 s:0	r:1 s:1
A	✗	✗	✓	✗
B	✗	✓	✓	✗
C	✓	✓	✓	✗
D	✓	✓	✓	✓

Frage 5: “independent”

Initialisierung

```
volatile int x = 0; volatile int y = 0;
int r = 0; int s = 0;
```

Thread 1

```
x = 1;
```

Thread 2

```
if (x == 1
 && y == 0)
    r = 1;
```

Thread 3

```
if (y == 1
 && x == 0)
    s = 1;
```

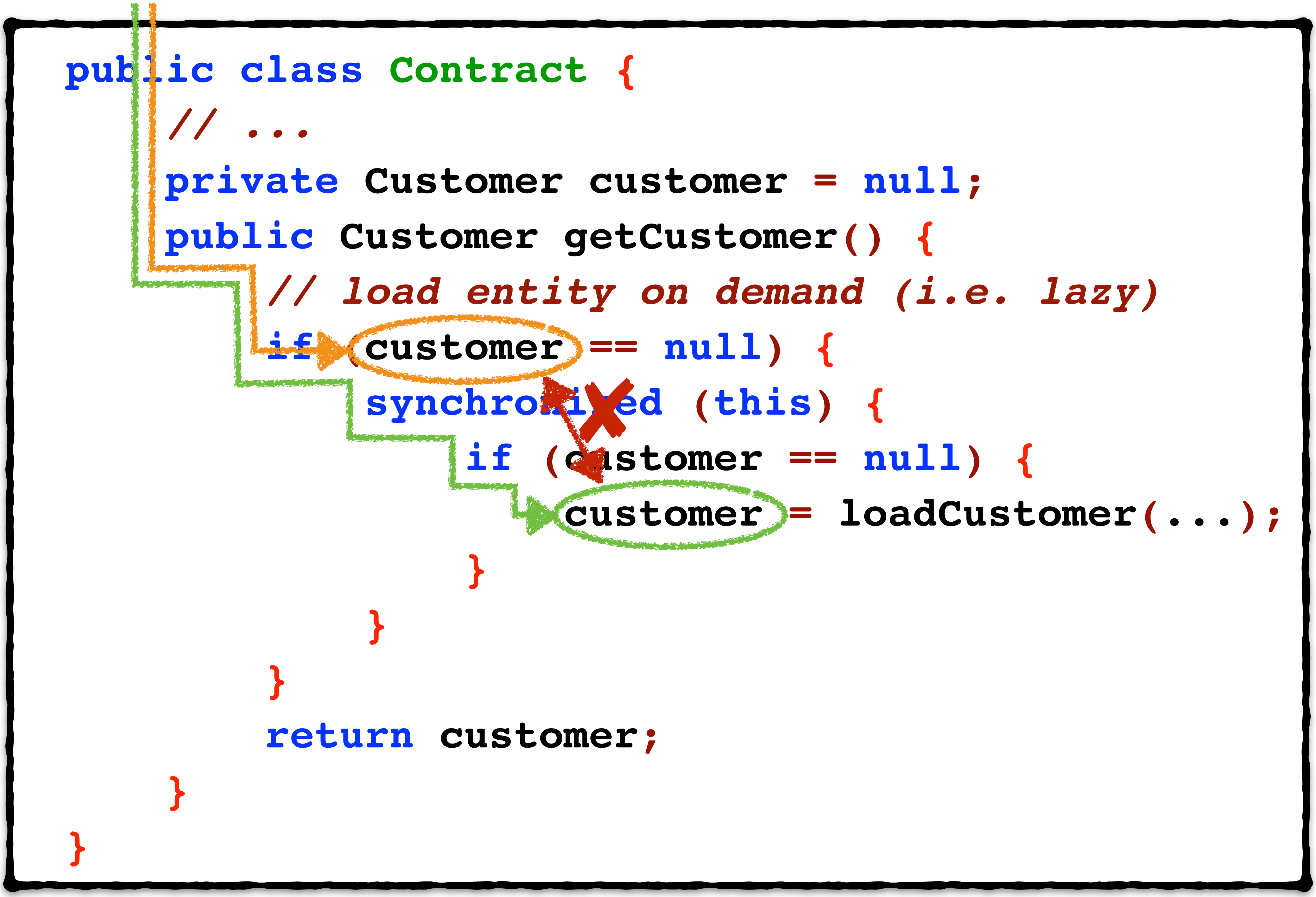
Thread 4

```
y = 1;
```

Mögliche Ergebnisse

Antwort	r:0 s:0	r:0 s:1	r:1 s:0	r:1 s:1
A	✗	✗	✓	✗
B	✗	✓	✓	✗
[✓ C]	✓	✓	✓	✗
D	✓	✓	✓	✓

Double-checked Locking



```
public class Contract {
    // ...
    private Customer customer = null;
    public Customer getCustomer() {
        // load entity on demand (i.e. lazy)
        if (customer == null) {
            synchronized (this) {
                if (customer == null) {
                    customer = loadCustomer(...);
                }
            }
        }
        return customer;
    }
}
```

Exkurs

Wichtige Spezialfälle

- long und double:
  - Sequential Consistency $\Longrightarrow$ atomare Zugriffe
- Arrays:
  - keine Data Races bei unterschiedlichen Indizes

volatile ≠ flush + refresh

```
private volatile boolean flush = false;
private volatile boolean refresh = true;
private boolean running = true;

public void run() {
    while (refresh && running) { }
}

public voi
    running = false;
    flush = true;
}
```

Unterschiedliche Variablen
⇒ Keine Synchronisation!

Use-Case für final

```
Global.s = "/tmp/usr".substring(4);
```

```
String myS = Global.s;
if (myS.equals("/tmp"))
    System.out.println(myS);
```

> “final fields are designed to allow
> for necessary security guarantees,
> [..] if malicious code is using data races
> to pass [..] references between threads.”

–The Java® Language Specification · Java SE 8 Edition · §17.5

Miscompilation

- Enthält das Programm eine *Data Race*, falls zwei Threads die Methode **bar** gleichzeitig aufrufen?

```java
public class Foo {
    private int count;
    // ...
    void bar(int[] vs) {
        for (int v : vs)
            if (v == 42)
                count++;
    }
}
```

```java
public class Foo {
    private int count;
    // ...
    void bar(int[] vs) {
        int reg = count;
        for (int v : vs)
            if (v == 42)
                reg++;
        count = reg;
    }
}
```

Zusammenfassung

“Sequential Consistency for Data-Race-Free Programs”

Intuitive Ausführungsreihenfolge solange auf keine non-volatile Variable gleichzeitig aus zwei Threads zugegriffen wird, wobei es sich bei mindestens einem Zugriff um eine Schreiboperation handelt.

Quellen

- “Java Language and Virtual Machine Specifications” by ORACLE
- “Threads and Shared Variables in C++11” by Hans Boehm
- “How to miscompile programs with ‘benign’ data races” by Hans Boehm
- “Why are two writes to the same variable conflicting in the Java memory model?” by Hubert Schmid (Stack Overflow)
- “Are final fields really useful regarding thread-safety?” by Hubert Schmid (Stack Overflow)
- “Cost of using final fields” by Hubert Schmid (Stack Overflow)
- “The paragraph as letter” by “SvaRoM” is licensed under CC BY-SA 3.0
- “Arcadia pocket watch 1859” by “Suebun” is licensed under CC BY-SA 3.0
- “Question mark made of question marks” by “Khaydock” is licensed under CC BY-SA 3.0

Fragen?