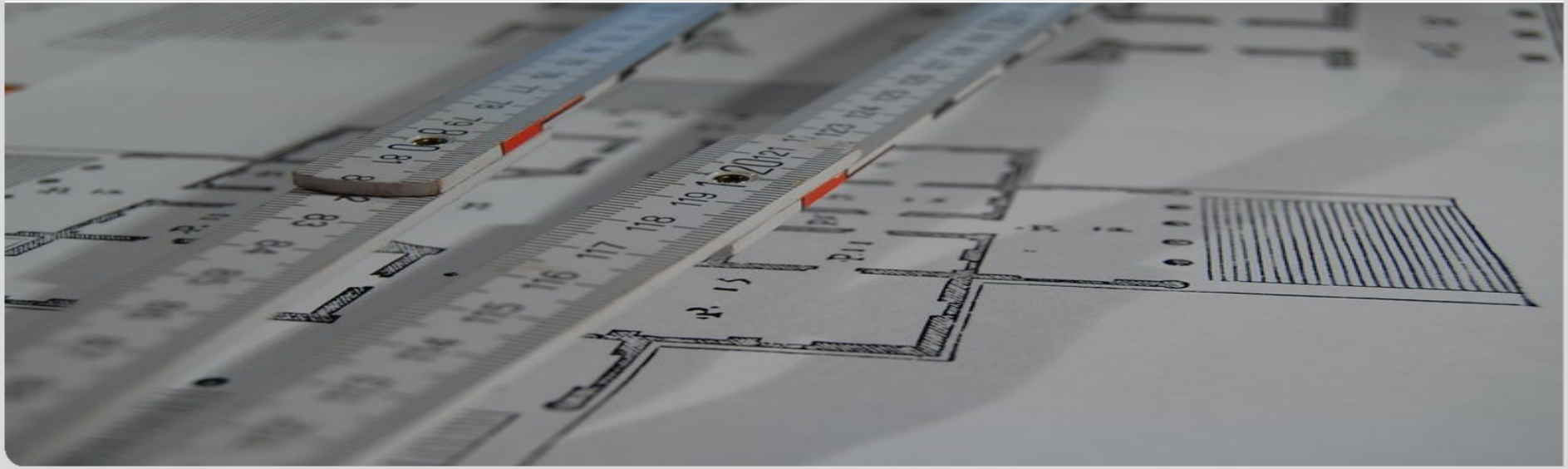
Karlsruhe Institute of Technology

Memory Models

Frederik Zipp <frederik.zipp@student.kit.edu>

Seminar: Programmiersprachen für Parallele Programmierung (SS 2010)

Fakultät für Informatik - IPD Snelting



Überblick

- Motivation
  - Wozu ein Memory Model?
  - Probleme ohne Memory Model (Beispiele C++)
- Memory Models
  - Definition Memory Model
  - Sequentielle Konsistenz
  - Java Memory Model
- Zusammenfassung
- Fragen

Parallele Programmierung

- Vorteile
  - Ausnutzung von Multi-Core-Prozessoren
  - Leistungssteigerung
- Nachteile
  - Fehleranfällig
  - Race Conditions

Threads per Bibliothek

- C und C++:
- Threads nicht Teil der Sprachspezifikation
- Stattdessen Thread-Unterstützung über Bibliotheken mit Synchronisierungs-Primitiven (Locking)
  - *pthreads* (Posix Threads)

```
pthreads_mutex_lock (…)
…
pthreads_mutex_unlock (…)
```

Probleme

Optimierungen durch Compiler und Hardware können unerwartetes Verhalten hervorrufen.

Zum Beispiel durch:

- Umsortierung von Instruktionen (Reordering)
- Implizites Schreiben von nebeneinandergelegenen Speicherbereichen

Beispiel 1

x und *y* initial 0

Thread A: `x = 1; r1 = y;`
Thread B: `y = 1; r2 = x;`

Ist *r1* == *r2* == 0 ein mögliches Ergebnis nach Ende der Threads?

Beispiel 1: Reordering

Thread A: `x = 1;` `r1 = y;`
Thread B: `y = 1;` `r2 = x;`

Reordering durch Compiler oder CPU

Thread A: `r1 = y;` `x = 1;`
Thread B: `r2 = x;` `y = 1;`

r1 == r2 == 0 nun mögliches Ergebnis!

Beispiel 2

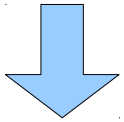
x und *y* initial 0

Thread A: `if (x == 1) ++y;`
Thread B: `if (y == 1) ++x;`

Ist *x* == *y* == 1 ein mögliches Ergebnis?

Beispiel 2

Thread A: `if (x == 1) ++y;`
Thread B: `if (y == 1) ++x;`



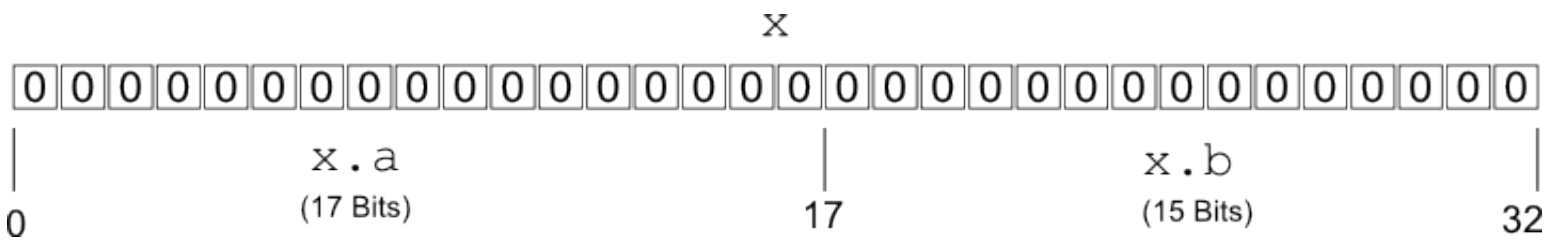
Compiler-Optimierung

Thread A: `++y; if (x != 1) --y;`
Thread B: `++x; if (y != 1) --x;`

`x` == `y` == 1 nun mögliches Ergebnis!

Beispiel 3: Rewriting of Adjacent Data

```
struct { int a:17; int b:15 } x;
```



Zuweisung zu einem der beiden Felder impliziert hardwarebedingt auch Schreiben des anderen Feldes

(CPU arbeitet mit 32-Bit Registern)

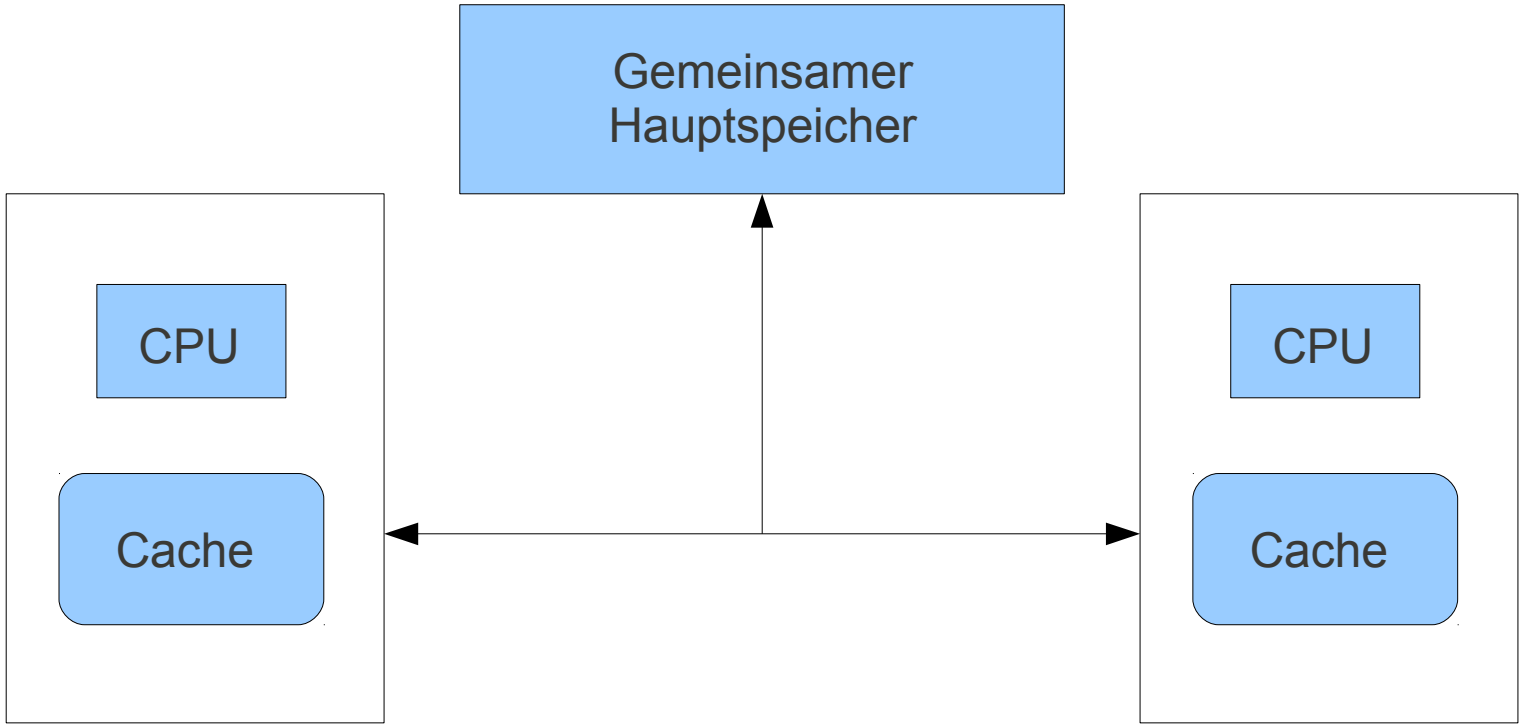
Memory Model

- Threading kann nicht allein über Bibliotheken umgesetzt werden.
- Spezifikation auf Sprachebene notwendig
  => *Memory Model*
  - Spezifiziert, wie Speicheroperationen (z.B. Lese- und Schreibzugriffe) in einem Programm dem Programmierer gegenüber in Erscheinung treten und welchen Wert jeder Lesezugriff auf einen Speicherbereich zurückliefern kann.
  - Bestimmt die Transformationen, die das System (Compiler, virtuelle Maschine oder Hardware) an einem Programm durchführen kann.

Sequentielle Konsistenz

- Einfachstes Memory Model
- Threads führen Operationen in sequentieller Reihenfolge aus
- Threads, die später drankommen sehen Speicheränderungen der Threads die vorher dran kamen
- Keine Optimierungen

Java Memory Model



Java Memory Model

- Synchronisation
  - Schlüsselwort `synchronized`
  - Locking / Unlocking
- Volatile Variablen
  - Schlüsselwort `volatile`
  - Für atomare Typen

Java Memory Model

- *Atomizität (Atomicity):* Welche Operationen sind atomar, d.h. werden nicht durch andere Threads unterbrochen?
- *Sichtbarkeit (Visibility):* Wann werden Modifikationen im Speicher anderen Threads sichtbar gemacht?
- *Reihenfolge (Ordering):* In welcher Reihenfolge passieren die Aktionen?

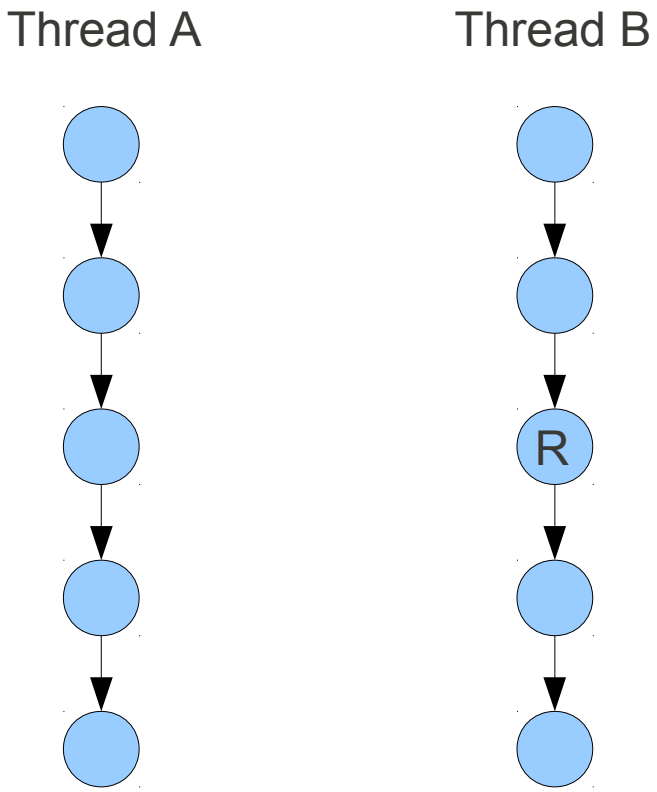
Atomare Operationen

- Zugriff auf Variablen von primitivem Typ (außer *long* und *double*) und Referenzen
- Zugriff auf *volatile* Variablen (inkl. *long* und *double*)
- Operationen auf Variablen von Typen aus `java.util.concurrent.atomic`

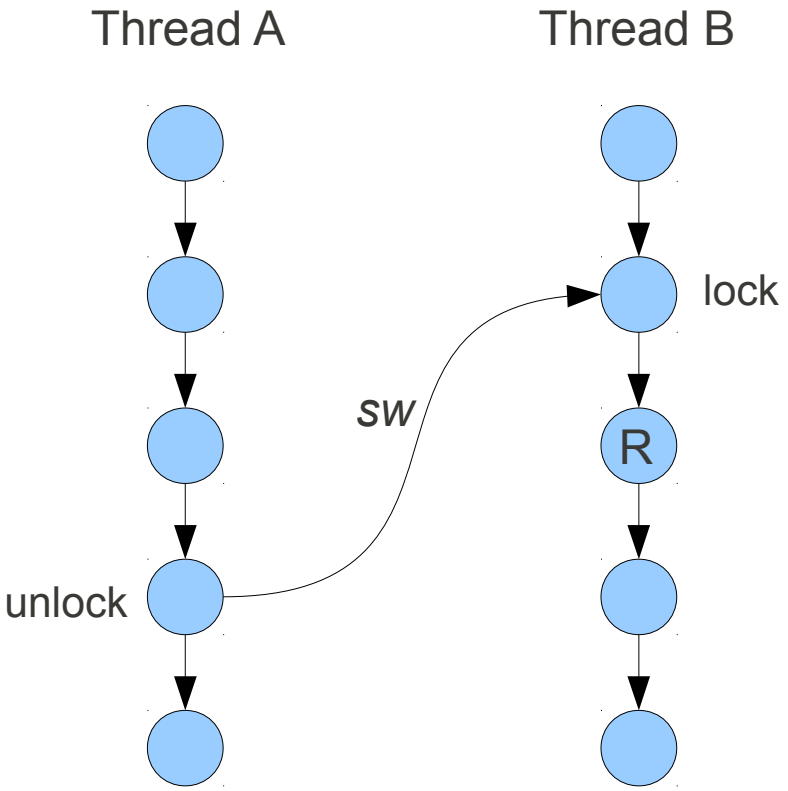
Sichtbarkeitsregeln

- Refresh / Flush bei
  - Threadstart und Threadende
  - Synchronisation
  - Lese- und Schreibzugriff auf *volatile*-Variablen
  - Lese- und Schreibzugriff auf atomare Variablen (`java.util.concurrent.atomic`)
  - Erstmaliger Lesezugriff auf final-Variablen

Happens-Before-Graph



Happens-Before-Graph

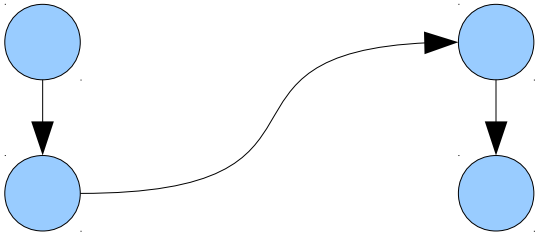


sw = synchronizes-with

Beispiel mit *volatile*

`x == 0; ready == false`
`ready` ist *volatile*

Thread A	Thread B
`x = 1;` `ready = true;`	`if (ready)` `  r1 = x;`

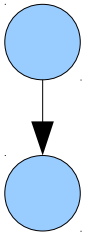


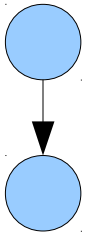
x == 1, wenn ready == true

Beispiel ohne *volatile*

```
x == 0; ready == false
```

Thread A	Thread B
`x = 1;` `ready = true;`	`if (ready)` `  r1 = x;`





sowohl x == 1 als auch x == 0 sind möglich

Zusammenfassung

- Sicheres Threading kann nicht allein als Bibliothek implementiert werden
- Programmiersprache muss Regeln für Speicherzugriffe festlegen (Memory Model)
- Compiler/Laufzeitsystem müssen dafür sorgen, dass die High-Level-Sprachkonstrukte gemäß den Regeln in Low-Level-Operationen umgesetzt werden

Fragen? Kommentare?

- Links
  - http://www.hpl.hp.com/personal/Hans_Boehm/c++mm/
  - http://java.sun.com/docs/books/jls/third_edition/html/memory.html
  - http://www.cs.umd.edu/~pugh/java/memoryModel/