Abstrakte C-Maschine und Stack

Julian Tobergte

Proseminar "C- Grundlagen und Konzepte", 2013

2013-06-21

Gliederung

Abstrakte Maschine

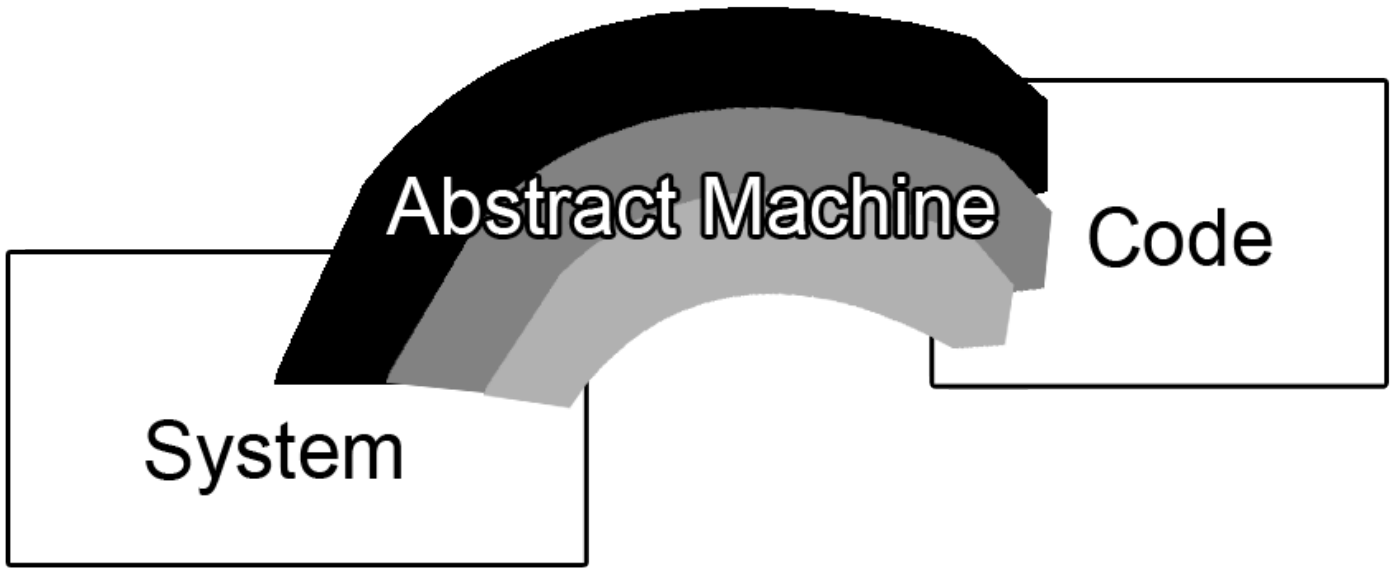
- A schreibt C-Programm D
  - will D auf seinem System ausführen
  - will D seinem Freund B geben

Abstrakte Maschine

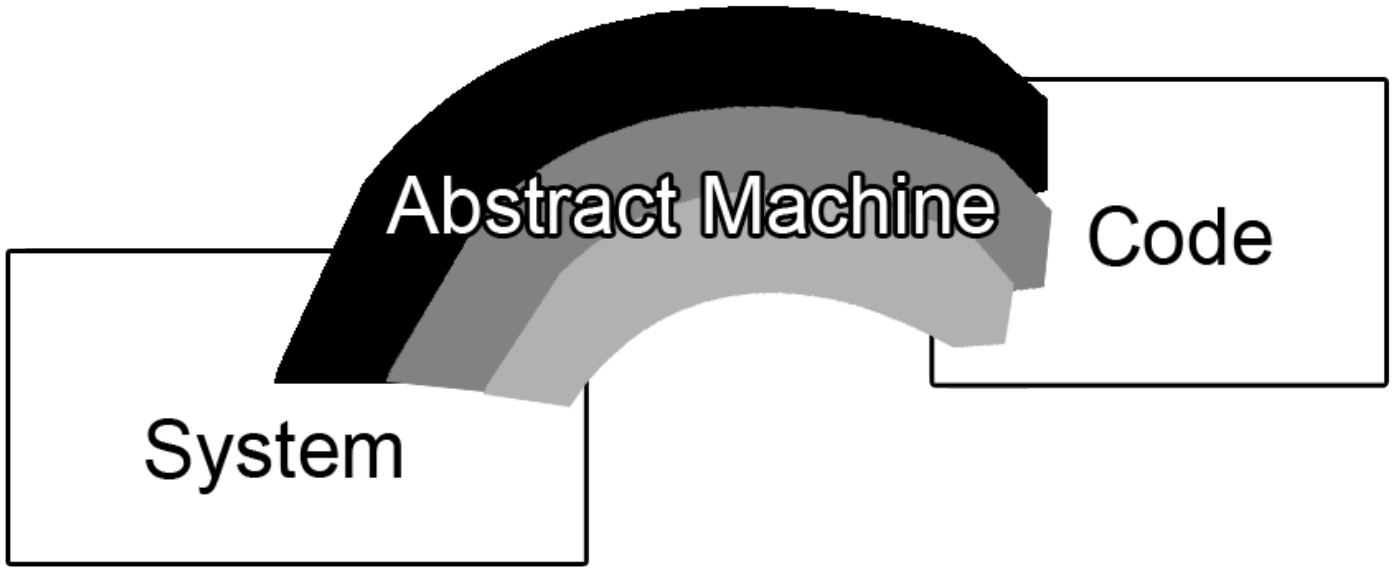
- A schreibt C-Programm D
  - will D auf seinem System ausführen
  - will D seinem Freund B geben
- Problem:

  ?

Was ist das?

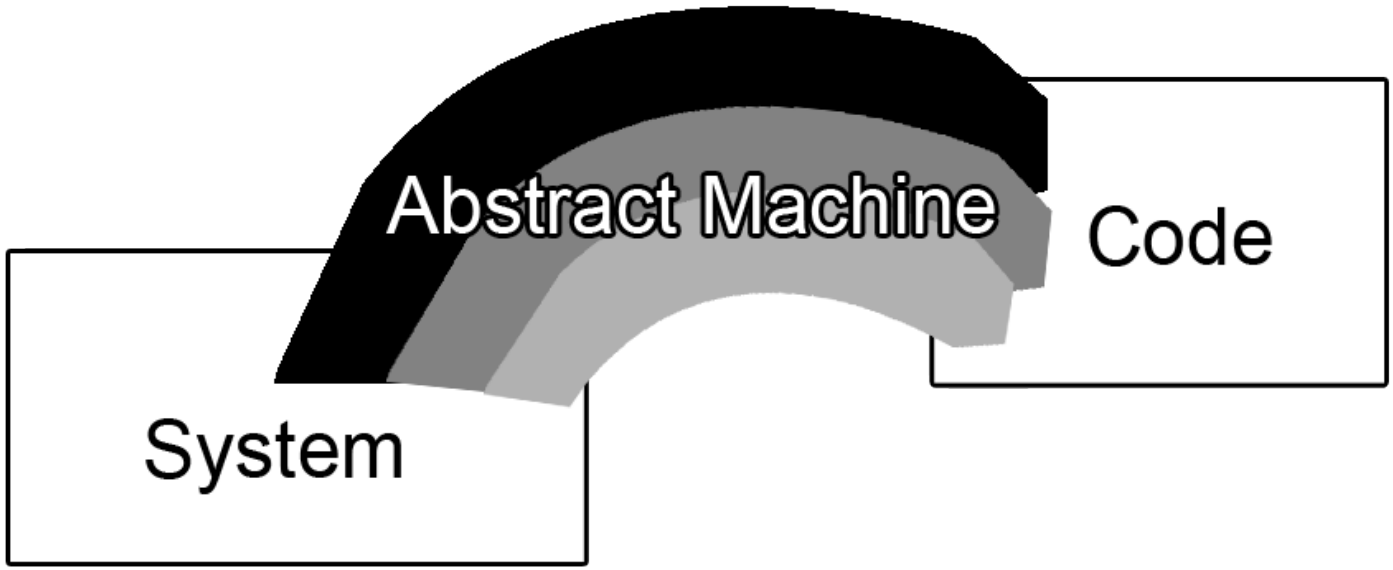


Was ist das?



- Brücke zwischen Ausführendem System und Programm-code
  - "Abstrakte" Version der Hardware

Was ist das?



- Brücke zwischen Ausführendem System und Programm-code
  - "Abstrakte" Version der Hardware
- Implementiert die Befehle der Sprache

Compiler / Interpreter ?

- Interpreter
  - Implementation der Abstrakten Maschine
  - verarbeitet den Code zur Laufzeit
  - z.B. Python

Compiler / Interpreter ?

- Interpreter
  - Implementation der Abstrakten Maschine
  - verarbeitet den Code zur Laufzeit
  - z.B. Python
- Compiler
  - Übersetzt Code für jeweilige Plattform
  - keine Abstrakte Maschine

Compiler / Interpreter ?

- Interpreter
  - Implementation der Abstrakten Maschine
  - verarbeitet den Code zur Laufzeit
  - z.B. Python
- Compiler
  - Übersetzt Code für jeweilige Plattform
  - keine Abstrakte Maschine
- Beides
  - Intermediate Language

Warum will ich das haben?

- Effizient
  - nur 1 Quellcode
  - keine Sorgen
- Zukunft
  - Code auch ausführbar
  - muss / kann ihn nicht anpassen
- Zusätzliche Funktionen

Funktionsweise 1/2

- Spezifikation der Sprache
  - bezieht sich auf eine Abstrakte Maschine
  - unabhängig von konkreten Systemen

Funktionsweise 1/2

- Spezifikation der Sprache
  - bezieht sich auf eine Abstrakte Maschine
  - unabhängig von konkreten Systemen
- Programmierer
  - schreibt Code für diese Abstrakte Maschine

Funktionsweise 1/2

- Spezifikation der Sprache
  - bezieht sich auf eine Abstrakte Maschine
  - unabhängig von konkreten Systemen
- Programmierer
  - schreibt Code für diese Abstrakte Maschine
- Compiler
  - bringt den Code auf einem konkreten System zum laufen

Funktionsweise 2/2

- Observable effects
  - As-If -Regel
  - Compiler / Interpreter dürfen alles

Funktionsweise 2/2

- Observable effects
  - As-If -Regel
  - Compiler / Interpreter dürfen alles
- Deshalb:
  - ungenaue Spezifikation erwünscht
  - Flexibilität

Beispiele 1/2

Beispiele 1/2

- Sprache OBIS

Beispiele 1/2

- Sprache OBIS
- Unterstützt Variablen

Beispiele 1/2

- Sprache OBIS
- Unterstützt Variablen
- Datentyp int

Beispiele 1/2

- Sprache OBIS
- Unterstützt Variablen
- Datentyp int
- Enthält eine Add-funktion

Beispiele 1/2

- Sprache OBIS
- Unterstützt Variablen
- Datentyp int
- Enthält eine Add-funktion
- Enthält Print-funktion

Beispiele 2/2

- Beispielquelltext OBIS:

```
a = 4
b = 2
c = a + b
print c
```

Beispiele 2/2

- Beispielquelltext OBIS:

```
a = 4
b = 2
c = a + b
print c
```

- Benötigt:
  - STORE
  - LOAD
  - ADD
  - PRINT

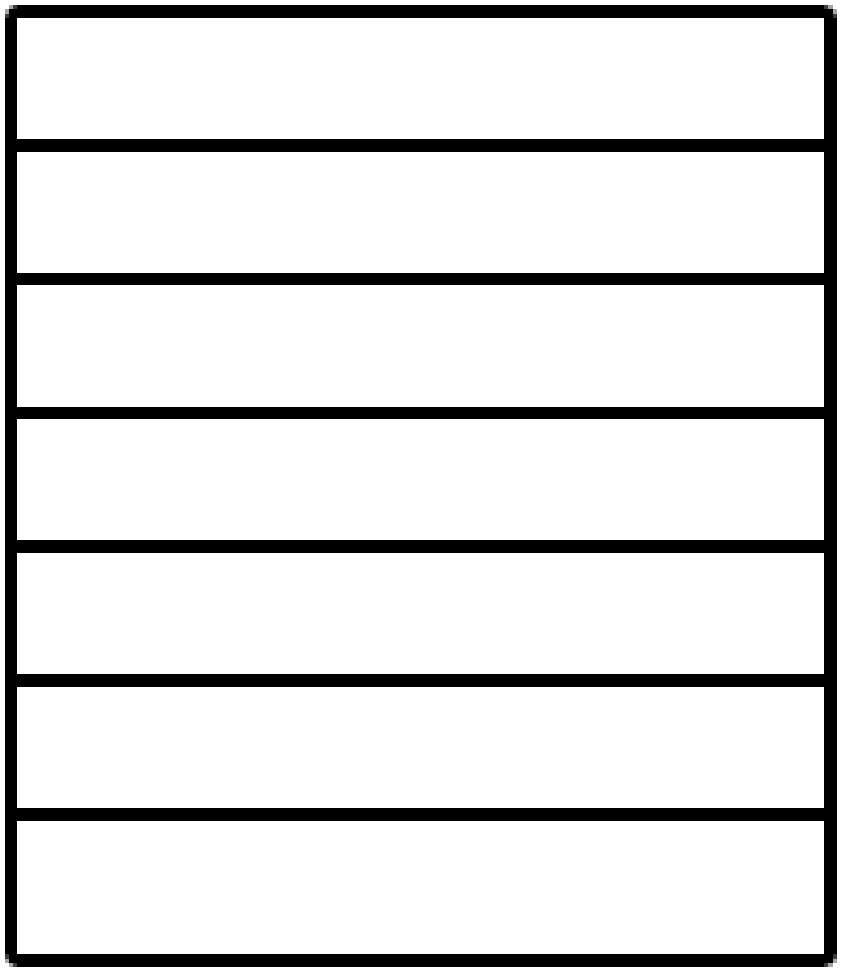
Beispiele 2/2

- Beispielquelltext OBIS:

```
a = 4
b = 2
c = a + b
print c
```

- Benötigt:
  - STORE
  - LOAD
  - ADD
  - PRINT
- Dabei wird nicht spezifiziert wie z.B. ADD intern arbeitet

Stack



Was ist das?

- Stapel
  - LIFO
  - Datenstruktur
  - Häufig von Compilern und Betriebssystemen benutzt
- 3 Operationen:

  `push`

  `pop`

  `peek`

Adressierung von Variablen

- Relativ zum Stack-Frame
  - %ebp Base-pointer : Start des Stack-Frame
  - %esp Stack-pointer : Ende des Stacks
- Beispiel:

  `8(%ebp)`

Funktionsaufruf

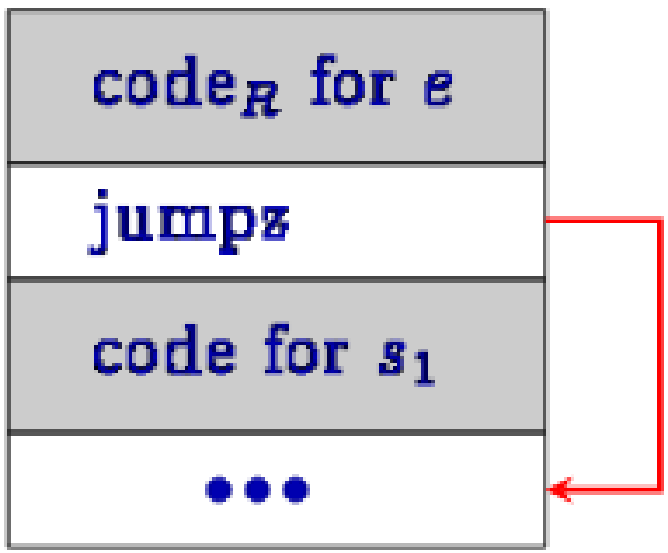
- Alten Base-Pointer zwischenspeichern
  - Rücksprungadresse
- Lokale Variablen auf dem Stack reservieren
- Gut geeignet für Rekursion
- Anfällig für Fehler
  - Falsche Rücksprungadresse
  - Angreifbar (Buffer Overflow)

Beispiel

```
int max(int x, int y)
{
     if (x > y)
        return x;
     else
        return y;
}
```

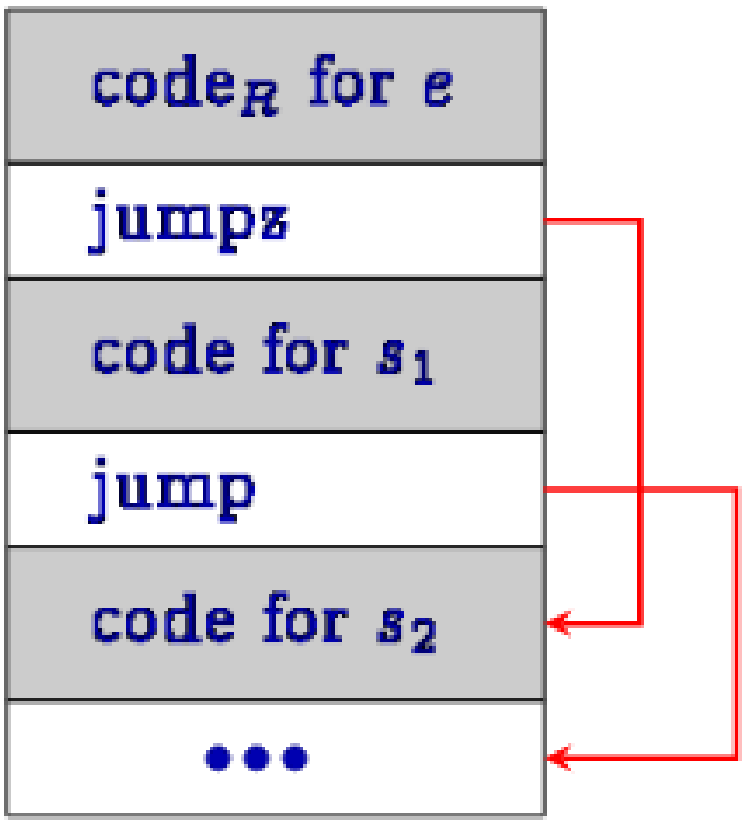
```
_max:
     pushl %ebp
     movl %esp, %ebp

     movl 8(%ebp), %edx
     movl 12(%ebp),%eax
     cmpl %eax, %edx
     jle end
     movl %edx, %eax
end:
     movl %ebp, %esp
     popl %ebp
     ret
```

Beispiele 2/3



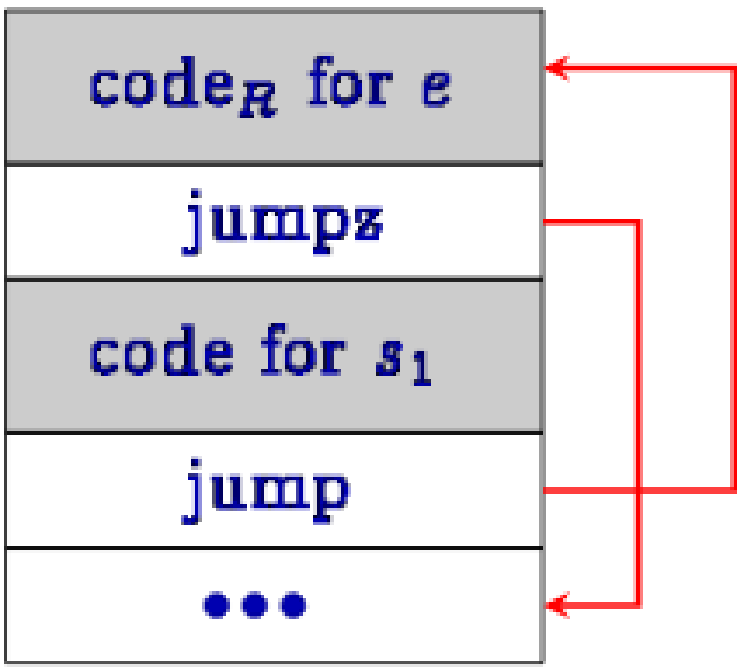
[2]

Beispiele 2/3



[2]

Beispiele 3/3



[2]

in C

Abstrakte Maschine in C

C Language Specification [4]

- Section 5.1.2.3 3)
  - In the abstract machine, all expressions are evaluated as specified by the semantics. An actual implementation need not evaluate part of an expression if it can deduce that its value is not used and that no needed side effects are produced (including any caused by calling a function or accessing a volatile object).
- Section 5.1.2.3 5)
  - The least requirements on a conforming implementation are:
    * At sequence points, volatile objects are stable in the sense that previous accesses are complete and subsequent accesses have not yet occurred.
    * At program termination, all data written into files shall be identical to the result that execution of the program according to the abstract semantics would have produced.

in C

```
static int a, b;

a = 5;

b = 42;

if (a == 5)
{
    <do stuff>
}
```

in C

```
static int a, b;

a = 5;

b = 42;

if (a == 5)
{
    <do stuff>
}
```

- b könnte nach dem if Block zugewiesen werden

in C

```
static int a, b;

a = 5;

b = 42;

if (a == 5)
{
    <do stuff>
}
```

- b könnte nach dem if Block zugewiesen werden
- b könnte vor a zugewiesen werden

in C

```
static int a, b;

a = 5;

b = 42;

if (a == 5)
{
    <do stuff>
}
```

- b könnte nach dem if Block zugewiesen werden
- b könnte vor a zugewiesen werden
- Andere Reihenfolge im Speicher

in C

Ist das nicht egal?

in C

Ist das nicht egal?

- Nur 1 Thread : ja

in C

Ist das nicht egal?

- Nur 1 Thread : ja
- Mehrere Threads: nein

in C

Ist das nicht egal?

- Nur 1 Thread : ja
- Mehrere Threads: nein
  - Annahme: a wird vor b geschrieben
  - So steht es im Code und das definiert die Sprache
  - Aber: Sprache ist für die Abstrakte Maschine definiert

in C

Ist das nicht egal?

- Nur 1 Thread : ja
- Mehrere Threads: nein
  - Annahme: a wird vor b geschrieben
  - So steht es im Code und das definiert die Sprache
  - Aber: Sprache ist für die Abstrakte Maschine definiert
  - Lösung: Lock-free programming

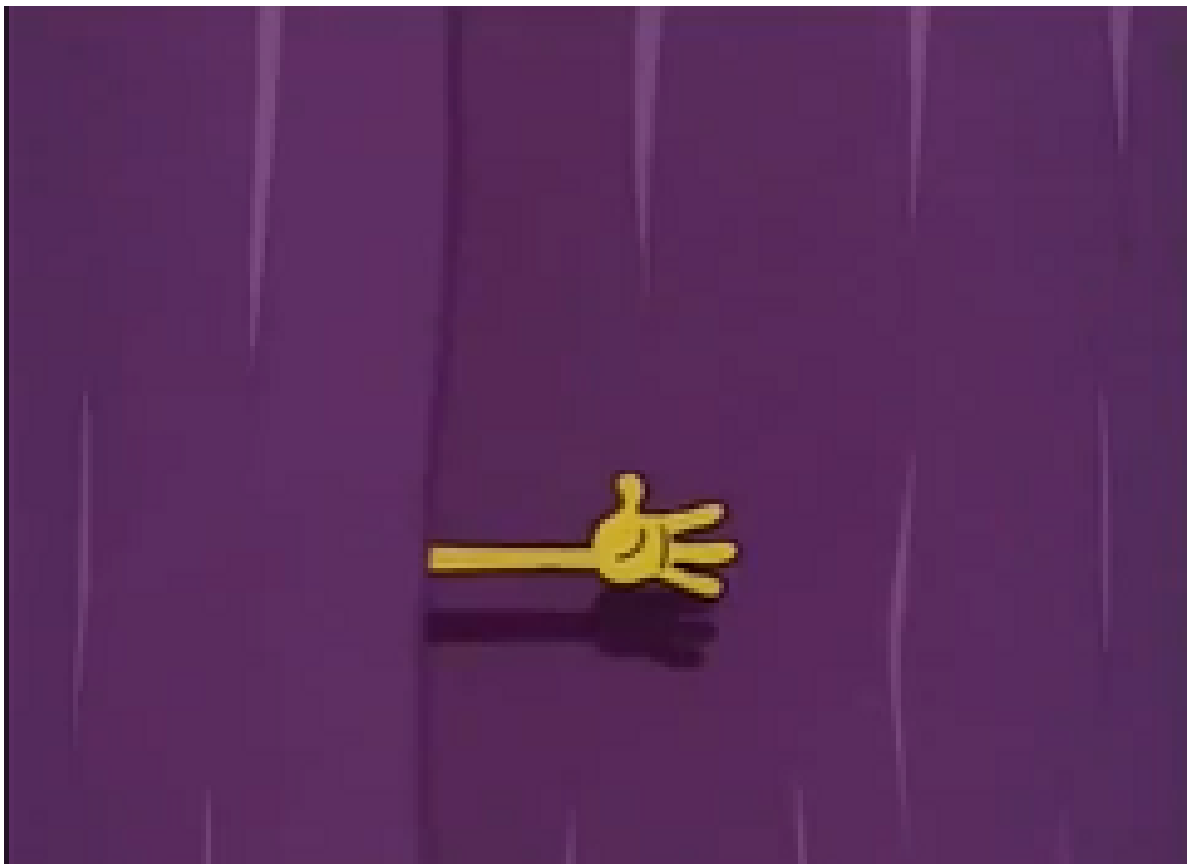
Optional

- C++11
  - "When you are talking about splitting [code] across different cores that's in the standard, we are talking about the memory model. We are going to optimize it without breaking the following assumptions people are going to make in the code," Sutter [3]
  - Multi-threading
- C++98/C++03:
  - Spezifikation: 1 Thread
  - Mehrere Threads möglich, aber kein standard
- Nun Multi-threading "fully portable"
  $\Longrightarrow$ Abstrakte C++11 Maschine dafür designt

Zusammenfassung

- Abstrakte Maschine
  - Ansatz des Interpreters
  - Ansatz des Compilers
  - Verständnis über Funktion / Sinn
- Stack
  - Verständnis über Funktion / Sinn
  - Gefahren
  - Kontrollstrukturen
    - if
    - if else
    - while
- Abstrakte Maschine in C
  - Probleme
  - Möglichkeiten
- C++11
  - Standard Speicher-modell für Multi-threading

Danke für eure Aufmerksamkeit



Danke für eure Aufmerksamkeit

Quellen

- http://en.wikibooks.org/wiki/LaTeX/Presentations
- http://www.cs.ut.ee/∼varmo/TM2008/slides/tm-cma.pdf [2]
- http://www.theregister.co.uk/2011/06/11/herb_sutter_next_c_plus_plus/page2.html [3]
- http://bradmajors.tumblr.com/post/51783938593/me-you
- http://bartoszmilewski.com/2008/12/01/c-atomics-and-memory-ordering/
- http://mortoray.com/2012/06/18/abstract-machines-interpreters-and-compilers/
- http://blogs.msdn.com/b/larryosterman/archive/2007/05/16/the-c-abstract-machine.aspx
- http://stackoverflow.com/questions/6319146/c11-introduced-a-standardized-memory-model-what-does-it-mean-and-how-is-it-g
- http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1124.pdf [4]
- http://www.inf.ed.ac.uk/teaching/courses/lsi/diehl_abstract_machines.pdf
- http://tams.informatik.uni-hamburg.de/lectures/2012ws/vorlesung/rs/index.php?content=01-unterlagen

∀ 06.06.2013