Scientific Computing WS 2019/2020

Lecture 2

Jürgen Fuhrmann

juergen.fuhrmann@wias-berlin.de

Me

- Name: Dr. Jürgen Fuhrmann (no, not Prof.)
- Affiliation: Weierstrass Institute for Applied Analysis and Stochastics, Berlin (WIAS);
  Deputy Head, *Numerical Mathematics and Scientific Computing*
- Contact: **juergen.fuhrmann@wias-berlin.de**
- Course homepage:
  `http://www.wias-berlin.de/people/fuhrmann/teach.html`
- Experience/Field of work:
  - Numerical solution of partial differential equations (PDEs)
  - Development, investigation, implementation of finite volume discretizations for nonlinear systems of PDEs
  - Ph.D. on multigrid methods
  - Applications: electrochemistry, semiconductor physics, groundwater...
  - Software development:
    - WIAS code pdelib (`http://pdelib.org`)
    - Languages: C, C++, Python, Lua, Fortran, Julia
    - Visualization: OpenGL, VTK

Admin stuff

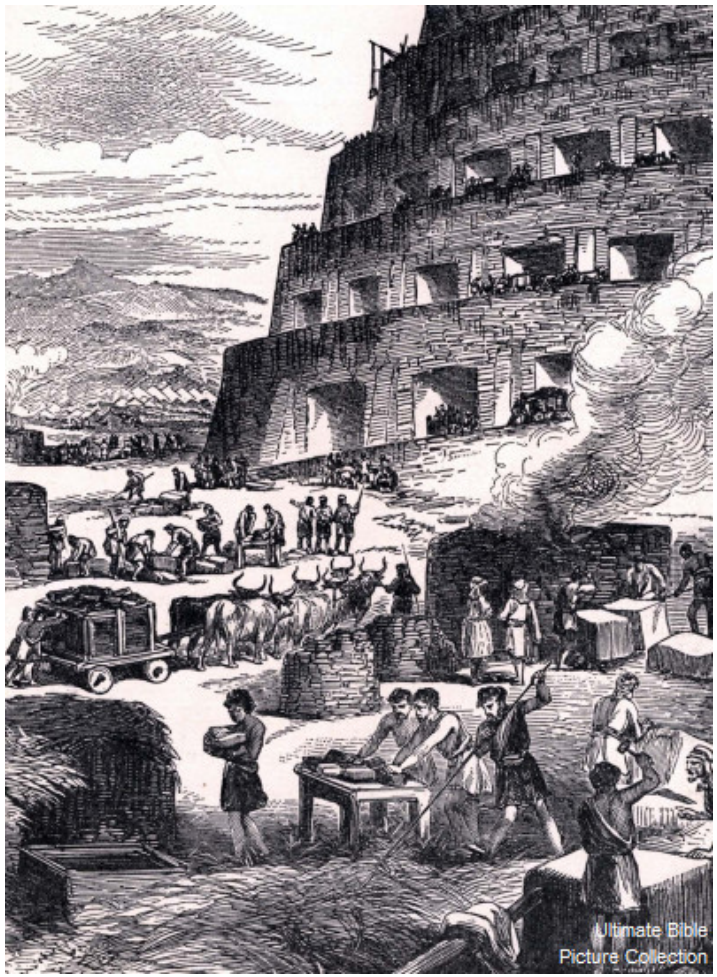
- Lectures: **Tue 10-12 FH 311**, **Thu 10-12 MA269**
- Consultation: **Thu 12-13 MA269**, more at WIAS on appointment
- There will be coding assignments in Julia
  - Unix pool
  - Installation possible for Linux, MacOSX, Windows
- Access to examination
  - Attend $\approx$ 80% of lectures
  - Return assignments
- Slides and code will be online, a script is being developed from the slides.
- See course homepage for literature, specific hints during course

UNIX Pool

- **Please sign up to the account list provided by the admins**
  - Working groups of two students per account/computer
  - Once the administrators open the accounts, you will be able to log in and enter a new password
- **Please check and correct the attendance list**
- All examples during this course will be available on UNIX pool systems.
  - More information is on `https://www.math.tu-berlin.de/iuk/lehrrechnerbereich/v_menue/lehrrechnerbereich/`
  - All homework can be done on UNIX pool machines as well (Room MA241 outside of course hours)
  - Class examples can be executed on the Jupyter server `https://www-pool.math.tu-berlin.de/jupyter/` (use your WIR account) (This possibility is new to both the admins and myself, so we are still exploring...)

Recap from last time

Confusio Linguarum

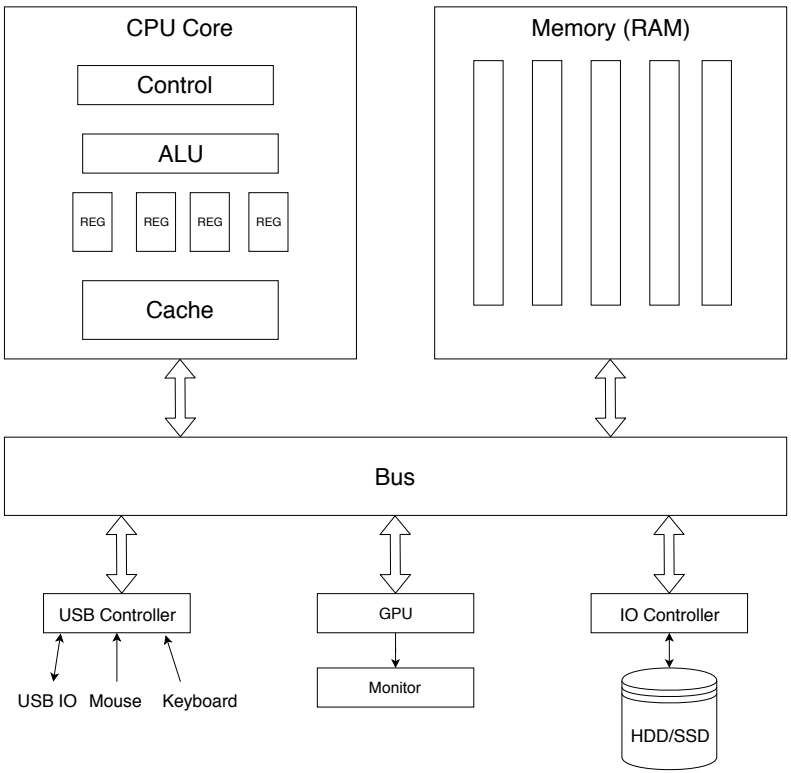


"And the whole land was of one language and of one speech. ... And they said, Go to, let us build us a city and a tower whose top may reach unto heaven. ... And the Lord said, behold, the people is one, and they have all one language. ... Go to, let us go down, and there confound their language that they may not understand one another's speech. So the Lord scattered them abroad from thence upon the face of all the earth." (Daniel 1:1-7)

Intended aims and topics of this course

- Indicate a reasonable path within this labyrinth
- Introduction to Julia
- Software management skills (version control)
- Relevant topics from numerical analysis
- Focus on partial differential equation (PDE) solution
  - Solution of large linear systems of equations
  - Finite elements
  - Finite volumes
  - Mesh generation
  - Linear and nonlinear solvers
  - Parallelization
  - Visualization

von Neumann Architecture



- Data and code stored in the same memory $\Rightarrow$ encoded in the same way, stored as binary numbers
- Instruction cycle:
  - Instruction decode: determine operation and operands
  - Get operands from memory
  - Perform operation
  - Write results back
  - Continue with next instruction

Machine code

- Detailed instructions for the actions of the CPU
- Not human readable
- Sample types of instructions:
  - Transfer data between memory location and register
  - Perform arithmetic/logic operations with data in register
  - Check if data in register fulfills some condition
  - Conditionally change the memory address from where instructions are fetched $\equiv$ "jump" to address
  - Save all register context and take instructions from different memory location until return $\equiv$ "call"
- Instructions are very hard to handle, although programming started this way ...

```
534c 29e5 31db 48c1 fd03 4883 ec08 e85d
feff ff48 85ed 741e 0f1f 8400 0000 0000
4c89 ea4c 89f6 4489 ff41 ff14 dc48 83c3
0148 39eb 75ea 4883 c408 5b5d 415c 415d
415e 415f c390 662e 0f1f 8400 0000 0000
f3c3 0000 4883 ec08 4883 c408 c300 0000
0100 0200 4865 6c6c 6f20 776f 726c 6400
011b 033b 3400 0000 0500 0000 20fe ffff
8000 0000 60fe ffff 5000 0000 4dff ffff
```

Assembler code

- Human readable representation of CPU instructions
- Some write it by hand ...
  - Code close to abilities and structure of the machine
  - Handle constrained resources (embedded systems, early computers)
- Translated to machine code by a programm called *assembler*

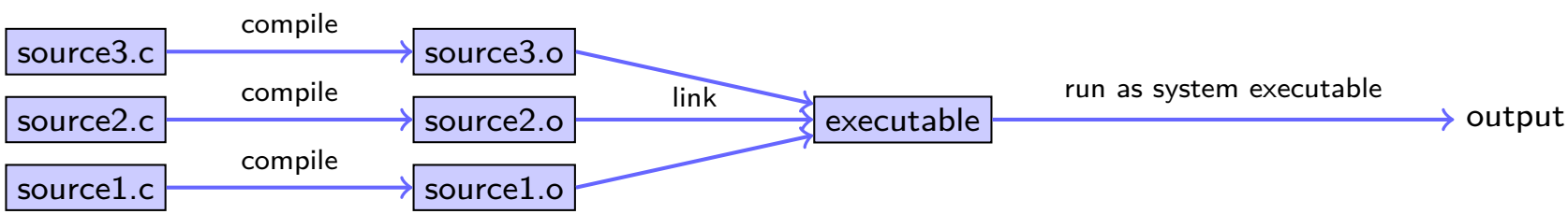
```
.file   "code.c"
.section    .rodata
.LC0:
.string "Hello world"
.text
...
pushq   %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq    %rsp, %rbp
.cfi_def_cfa_register 6
subq    $16, %rsp
movl    %edi, -4(%rbp)
movq    %rsi, -16(%rbp)
movl    $.LC0, %edi
movl    $0, %eax
call    printf
```

Compiled high level languages

- Algorithm description using mix of mathematical formulas and statements inspired by human language
- Translated to machine code (resp. assembler) by *compiler*

```
#include <stdio.h>
int main (int argc, char *argv[])
{
  printf("Hello world");
}
```

- "Far away" from CPU $\Rightarrow$ the compiler is responsible for creation of optimized machine code
- Fortran, COBOL, C, Pascal, Ada, Modula2, C++, Go, Rust, Swift
- Strongly typed
- Tedious workflow: compile - link - run



Compiled languages in Scientific Computing

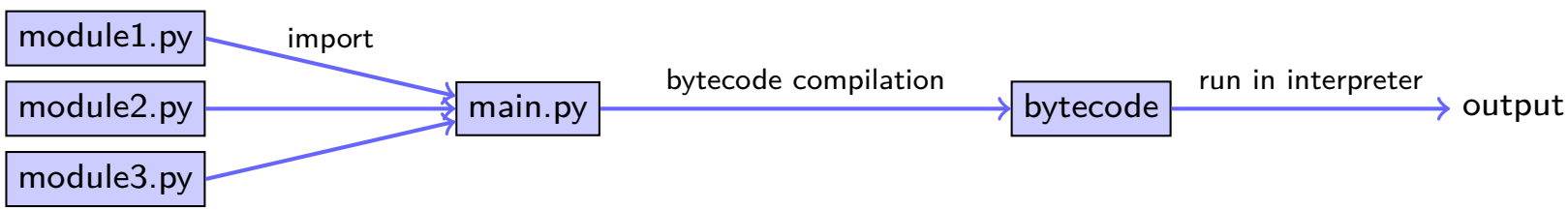
- Fortran: FORmula TRANslator (1957)
  - Fortran4: really dead
  - Fortran77: large number of legacy libs: BLAS, LAPACK, ARPACK ...
  - Fortran90, Fortran2003, Fortran 2008
    - Catch up with features of C/C++ (structures,allocation,classes,inheritance, C/C++ library calls)
    - Lost momentum among new programmers
    - Hard to integrate with C/C++
    - In many aspects very well adapted to numerical computing
    - Well designed multidimensional arrays
- C: General purpose language
  - K&R C (1978) weak type checking
  - ANSI C (1989) strong type checking
  - Had structures and allocation early on
  - Numerical methods support via libraries
  - Fortran library calls possible
- C++: *The* powerful object oriented language
  - Superset of C (in a first approximation)
  - Classes, inheritance, overloading, templates (generic programming)
  - C++11: $\approx$ 2011 Quantum leap: smart pointers, threads, lambdas, initializer lists in standard
  - Since then: C++14, C++17, C++20
  - With great power comes the possibility of great failure...

High level scripting languages

- Algorithm description using mix of mathematical formulas and statements inspired by human language
- Simpler syntax, less "boiler plate"

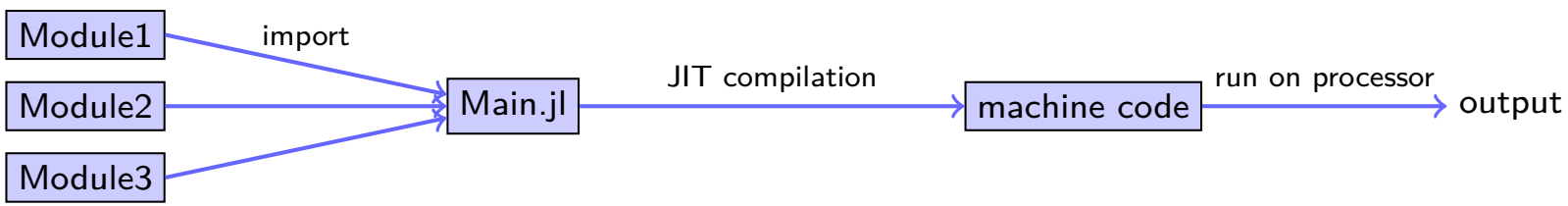
```
print("Hello world")
```

- Need intepreter in order to be executed
- Very far away from CPU $\Rightarrow$ usually significantly slower compared to compiled languages
- Matlab, Python, Lua, perl, R, Java, javascript
- Less strict type checking, powerful introspection capabilities
- Immediate workflow: "just run"
  - in fact: first compiled to *bytecode* which can be interpreted more efficiently



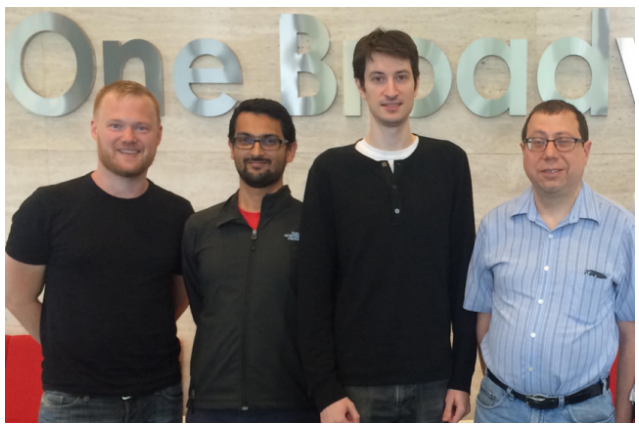
JIT based languages

- Most interpreted language first compile to bytecode wich then is run in the interpreter and not on the processor ⇒ perfomance bottleneck,
  - remedy: use compiled language for performance critical parts
  - "two language problem", additional work for interface code
- Better: Just In Time compiler (JIT): compile to machine code "on the fly"
  - V8 → javascript
  - LuaJIT, Java, Smalltalk
  - LLVM → **Julia** (v1.0 since August, 2018)
  - LLVM Compiler infrastructure → Python/NUMBA
- Drawback over compiled languages: compilation delay at every start, can be mediated by caching
- Advantage over compiled languages: simpler syntax, *tracing* JIT, i.e. optimization at runtime



Julia History & Resources

- 2009-02: V0.1 Development started in 2009 at MIT (S. Bezanson, S. Karpinski, V. Shah, A. Edelman)
- 2012: V0.1
- 2016-10: V0.5 experimental threading support
- 2017-02: SIAM Review: Julia - A Fresh Approach to Numerical Computing
- 2018-08: **V1.0**
- 2018 Wilkinson Prize for numerical software



- Homepage incl. download link: https://julialang.org/
- Wikibook: https://en.wikibooks.org/wiki/Introducing_Julia
- TU Berlin Jupyter server https://www-pool.math.tu-berlin.de/jupyter: you will be able to user your UNIX pool account

Julia - a first characterization

"**Like matlab, but faster**"

"**Like matlab, but open source**"

"**Like python + numpy, but faster and counting from 1**"

- Main purpose: performant numerics
- Multidimensional arrays as first class objects
  (like Fortran, Matlab; unlike C++, Swift, Rust, Go . . .)
- Array indices counting from 1
  (like Fortran, Matlab; unlike C++, python) - but it seems this becomes more flexible
- Array slicing etc.
- Extensive library of standard functions, linear algebra operations
- Package ecosystem