Transcript

Introduction to Operating Systems VO

Version v1.0

Daniel Kocher

Daniel.Kocher@stud.sbg.ac.at

Salzburg, October 7, 2014

Myth Busting

Myth: I prefer programming language X because it makes it more convinient to implement my application.

Answer: Wrong!

The fundamental problem of programming is to establish functional correctness and adequate performance. Different languages provide different tools of automating the process of establishing correctness and performance. The language should be chosen based on that insight.

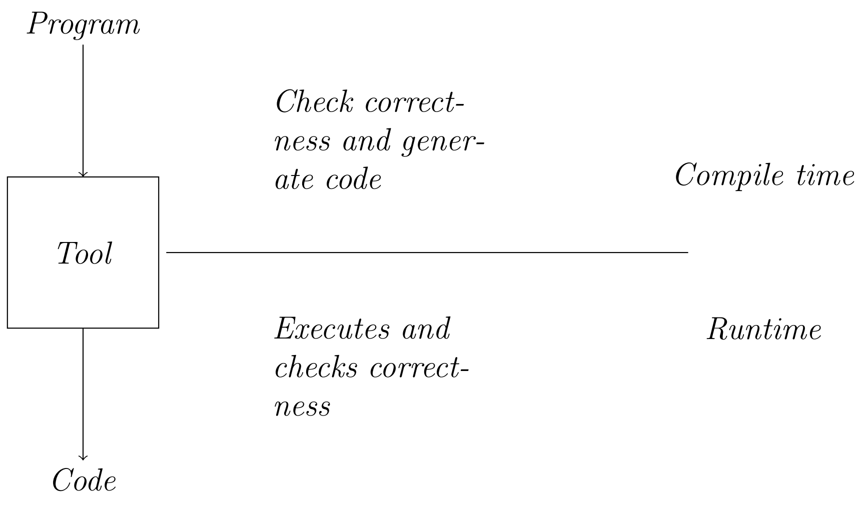
Program
Check correctness and generate code
Compile time
Tool
Executes and checks correctness
Runtime
Code

Figure 1: Process of establishing correctness.

Ultimate goal:
A Compiler checking correctness in such a sense that no exception is thrown while executing in any case. But this is infeasible (mathematical proof possible).

Hardware Exception: Interrupts, a mechanism to stop memory accesses in hardware.

Myth: I like garbage-collected languages like Java because they free me from memory management.

Answer: Wrong!

A garbage collector (GC) provides safe deallocation of unneeded memory but the programmer still needs to say what is unneeded, otherwise the system will run out of memory (memory leak).

General runtime complexity of a garbage collector: the size of the heap.

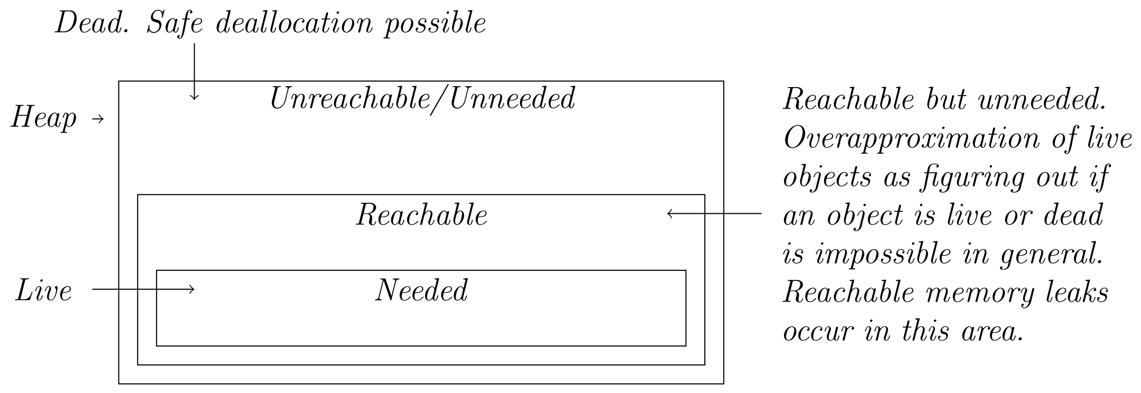
Dead. Safe deallocation possible
Heap
Unreachable/Unneeded
Reachable
Live
Needed
Reachable but unneeded. Overapproximation of live objects as figuring out if an object is live or dead is impossible in general. Reachable memory leaks occur in this area.

Figure 2: Unreachable, reachable and needed set of a program.

Multicore

Amdahl's Law: P represents program parallelism on N cores

- $S(N) = N$ if $P = 100\%$. This is ideal multicore scalability.
- In general: $S(N) = \frac{1}{(1-P)\cdot\frac{P}{N}}$

Sequential vs. Parallelized Code

The bottleneck of parallelized is the memory bus. It is limiting execution even if a problem can be perfectly parallelized without any side effects. Thus a parallelization factor of 100% is infeasible. Any shared resource at any level of an architecture creates a limitation.

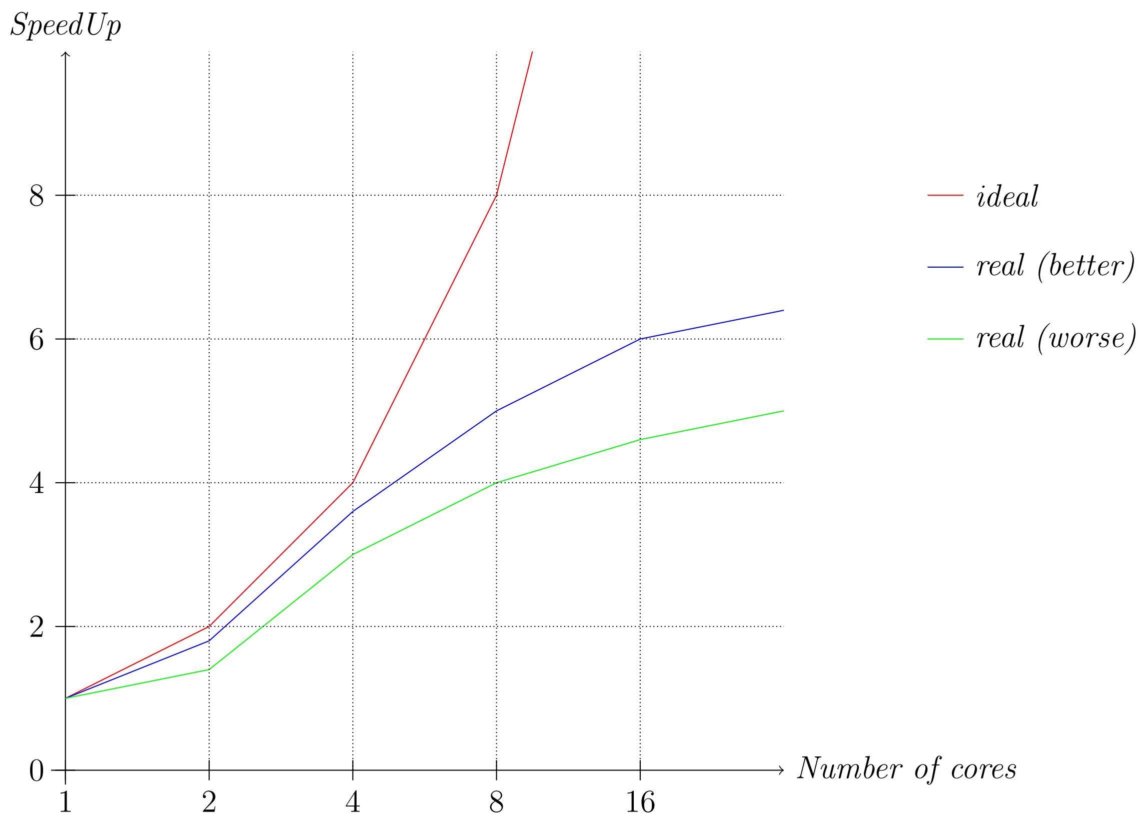

Figure 3: Amdahl's law plot.

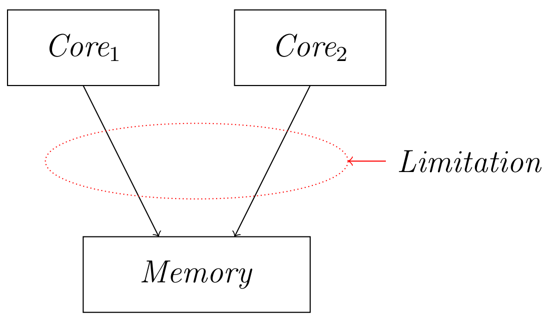

Figure 4: Bottleneck of parallelized code.

Architecture

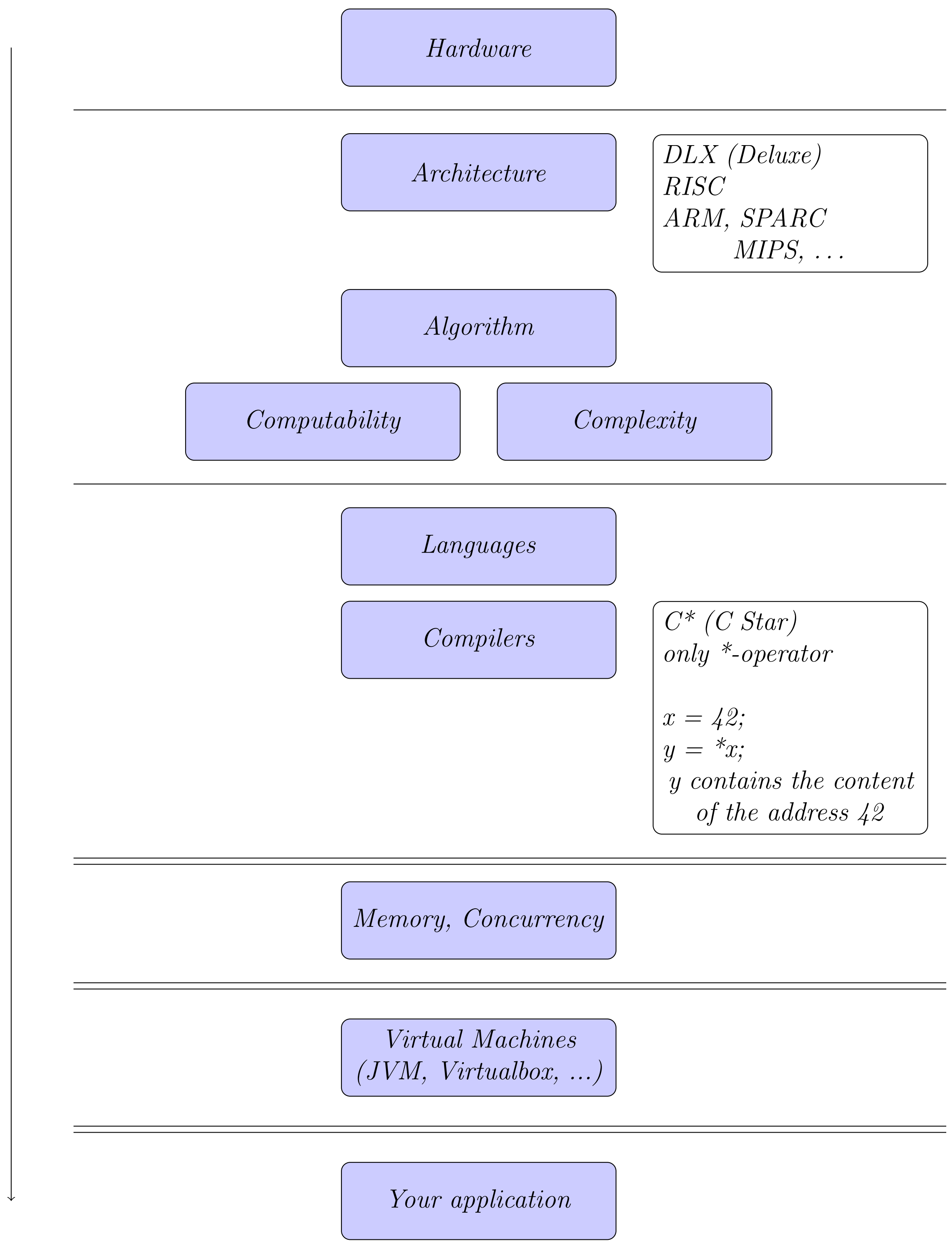

Figure 5: Architecture hierarchy (top-down.

Von Neumann Architecture

Introduced in 1945. This is a stored program computer: data = program. The Fetch-Decode-Execute cycle (see Figure 7) modifies the state of the machine.

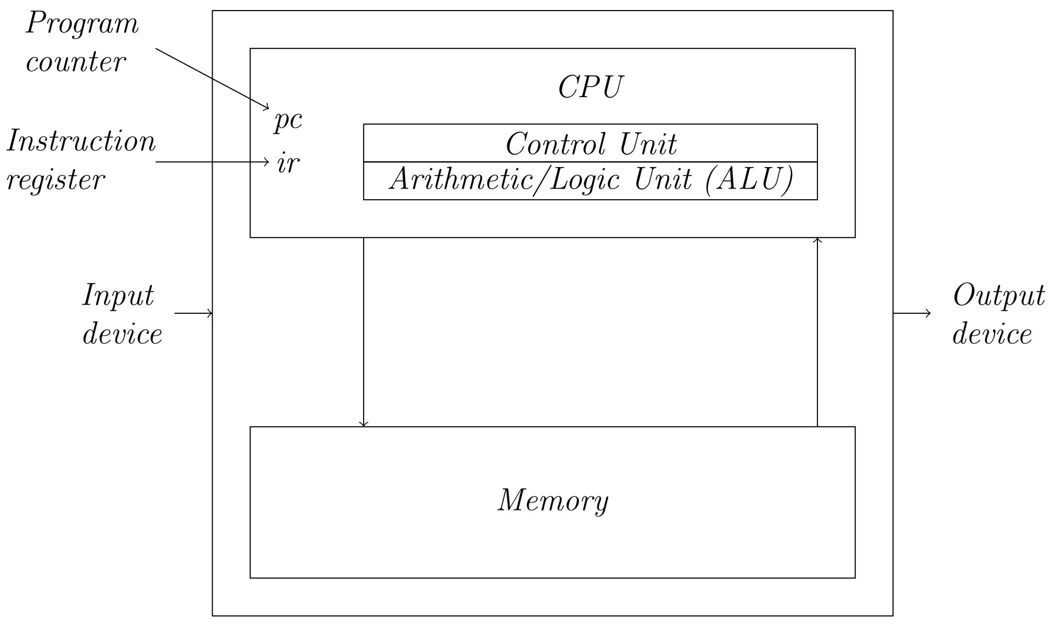

Figure 6: Von Neumann Architecture.

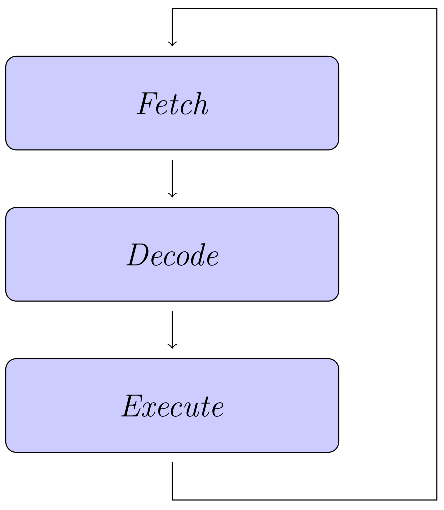

Figure 7: Fetch-Decode-Execute cycle.

DLX Machine

- Control unit:
  Instruction register (ir) and program counter (pc).
- Arithmetic unit:
  32x 32-bit registers; `reg[0]`, `reg[1]`, ..., `reg[31]`. `reg[0]` always contains the value `0` and `reg[31]` is the link register. Both are reserved by convention and must not be used for any other purpose.
- Memory:
  n 32-bit words, byte-addressed (see Figure 8), word-aligned;
  `mem[0]`, ..., `mem[n - 1]`.

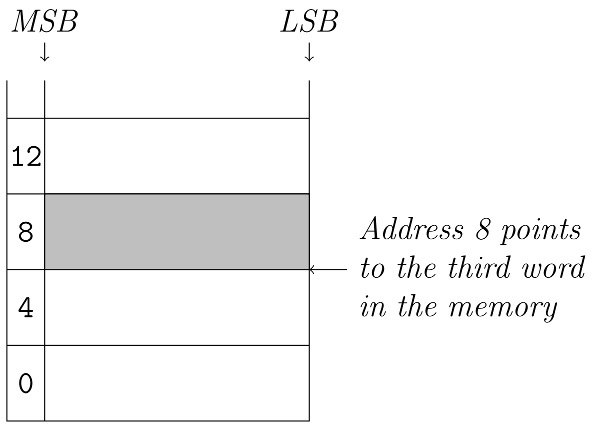

Figure 8: Visualization of a byte-addressed memory of 32-bit words.

Syntax Formats

General syntax of an instruction: `op a, b, c`.

F1

The length of `a` and `b` allow to address all 32 registers. The two's complement is used here because the implementation of arithmetics is easier (in contrast to the one's complement).

op	$0 \leq a \leq 31$	$0 \leq b \leq 31$	$-2^{15} \leq c \leq 2^{15} - 1$
6 bits	*5 bits*	*5 bits*	*16 bits sign-extended to 32 bits*
2^6 *ops*	$2^5 - 1$	$2^5 - 1$	

F2

E.g. `R1 = R2 + R3`

op	$0 \leq a \leq 31$	$0 \leq b \leq 31$	*unused*	$0 \leq c \leq 31$
6 bits	*5 bits*	*5 bits*	*11 bits*	*5 bits*
2^6 *ops*	$2^5 - 1$	$2^5 - 1$		$2^5 - 1$

F3

Absolute addressing in memory.

op	$0 \leq c \leq 2^{26} - 1$
6 bits	*26 bits*
2^6 *ops*	$2^{26} - 1$

Von Neumann Bottleneck: Memory read/write operations limit the performance.

Register Instructions

F1

`ADDI a, b, c`: Add immediate, `c` is data (a constant)
`reg[a] := reg[b] + c; pc := pc + 4;`

`SUBI a, b, c`: Subtract immediate
`reg[a] := reg[b] - c; pc := pc + 4;`