Nízkoúrovňové programování

http://d3s.mff.cuni.cz

Martin Děcký

decky@d3s.mff.cuni.cz

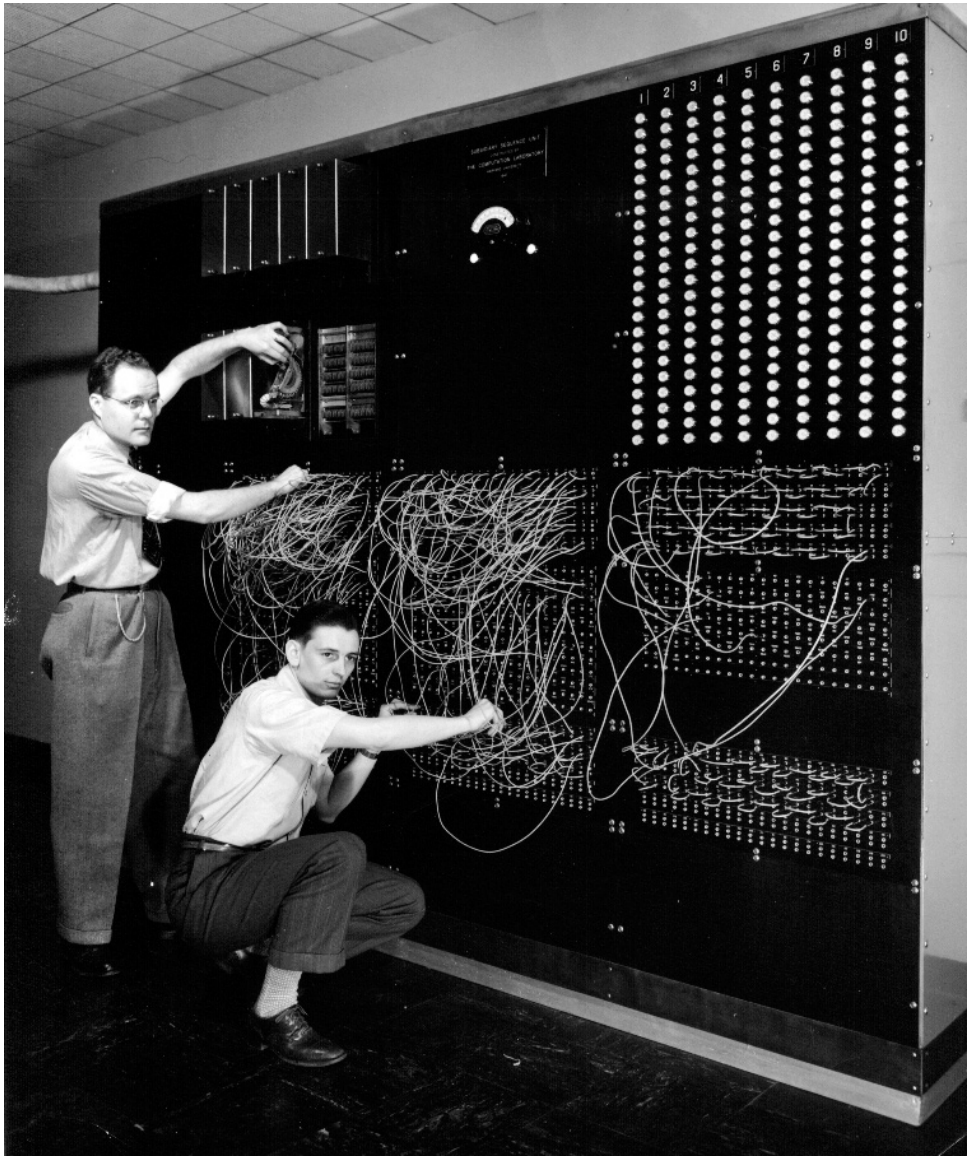
CHARLES UNIVERSITY IN PRAGUE

Faculty of Mathematics and Physics

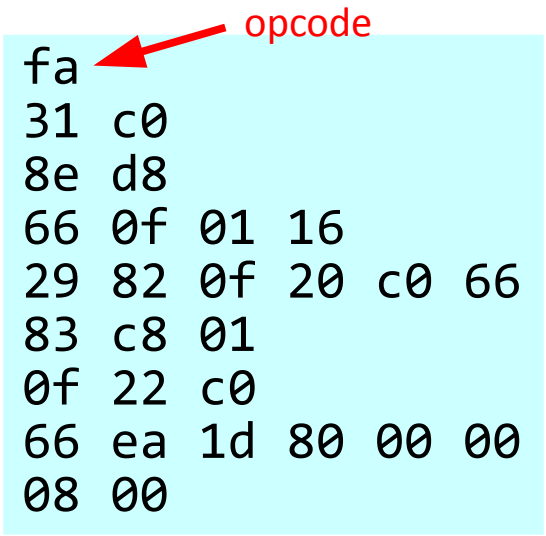
Department of Distributed and Dependable Systems

D3S

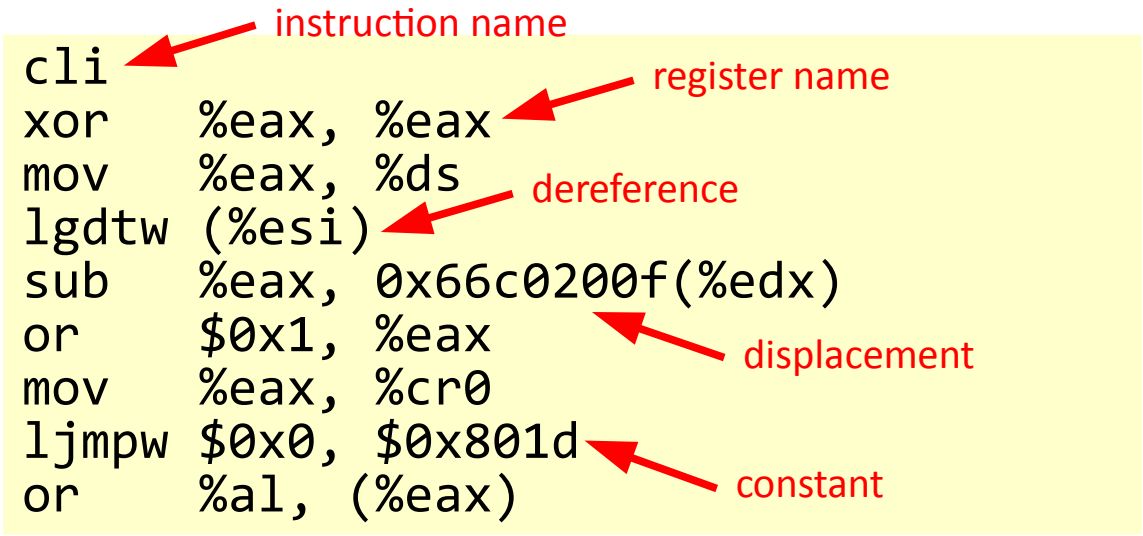
Nízkoúrovňové programování kdysi



Nízkoúrovňové programování dnes



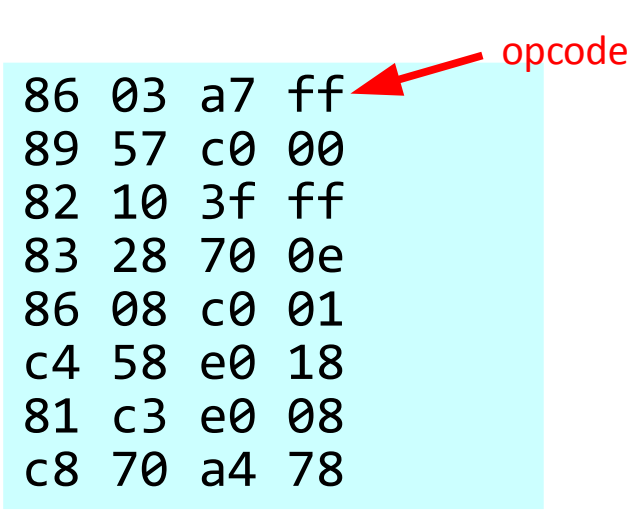
strojový kód na IA-32
Posloupnost bytů uložená v paměti kódující instrukce prováděné procesorem

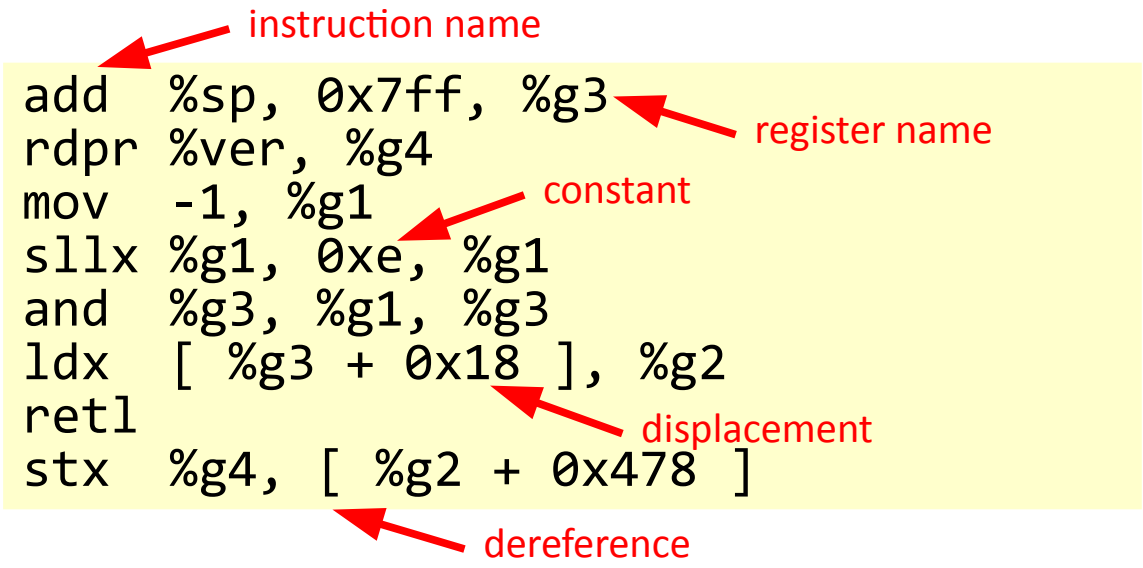


instrukční mnemonika na IA-32 (AT&T syntaxe)
Kód napsaný ručně nebo vyrobený překladačem vyššího programovacího jazyka

Assembler překládá mnemonika na strojový kód

Nízkoúrovňové programování dnes (2)





strojový kód na SPARC V9

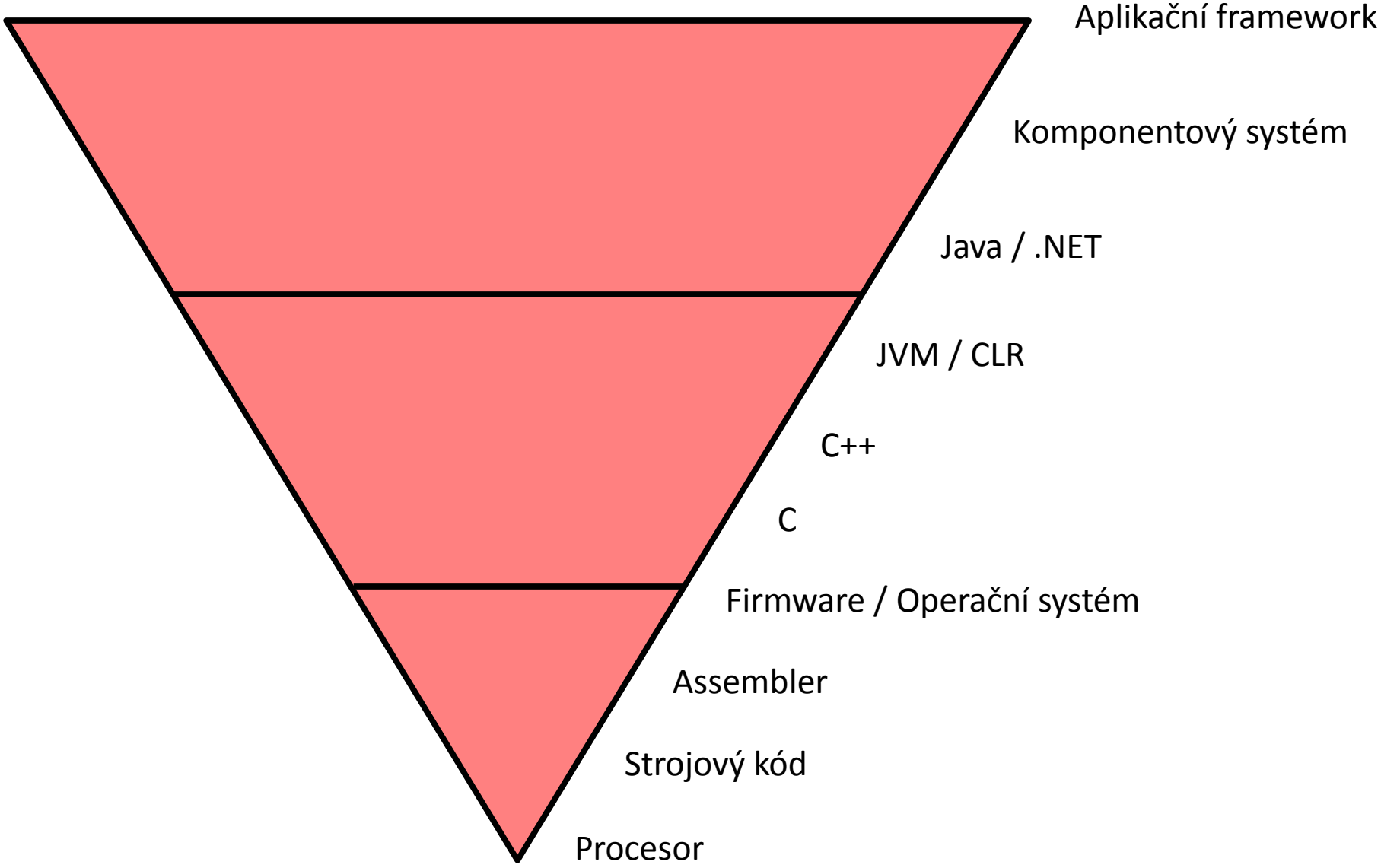
Všechny instrukce mají délku 4 byty

instrukční mnemonika na SPARC V9

K čemu mi to může být dobré?

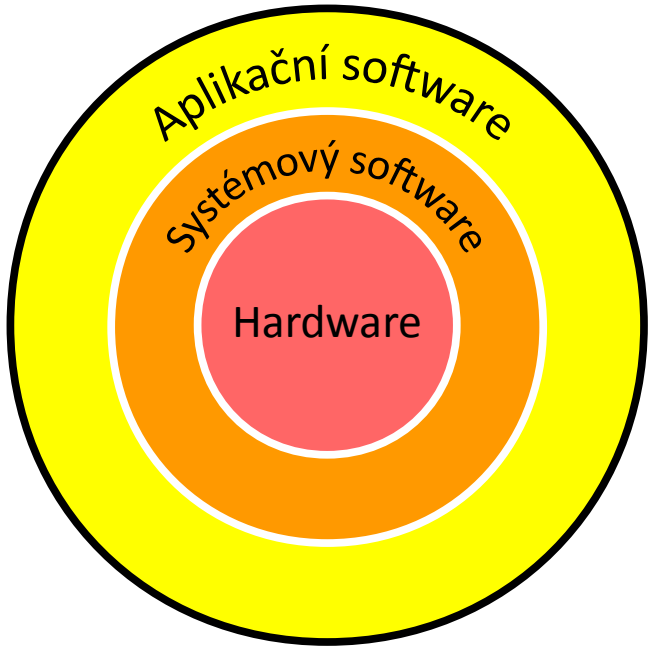
Vždyť přece budu stejně celý život programovat v Javě, C♯, Pythonu nebo PHP!


Disney
TRON
LEGACY
IN 3D DEC 2010
DISNEY.COM/TRON
©DISNEY

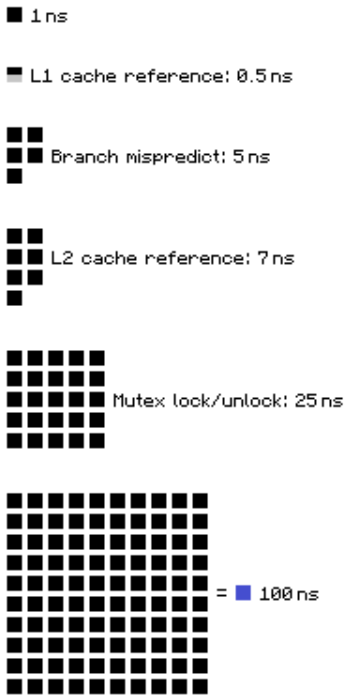

Aplikační framework
Komponentový systém
Java / .NET
JVM / CLR
C++
C
Firmware / Operační systém
Assembler
Strojový kód
Procesor

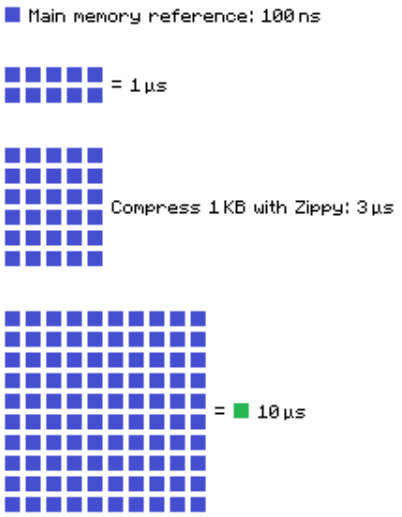
Stovky tisíc řádků kódu

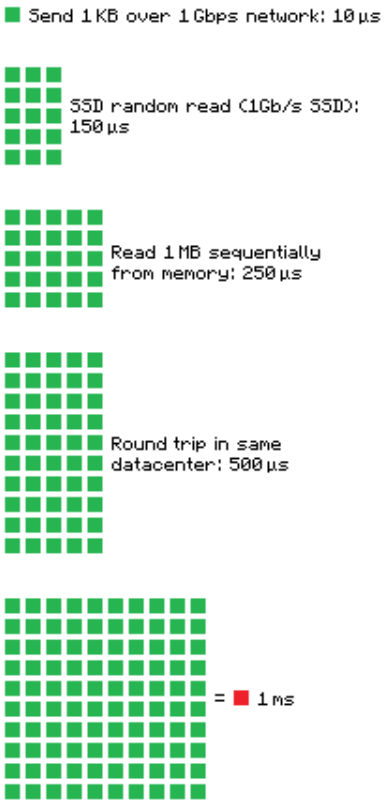
- **Aplikační software**
  - Např. textový editor
  - Knihovny pro uživatelské rozhraní
- **Systémový software**
  - Operační systém
    - Vstupně/výstupní operace
    - Alokace paměti a úložného prostoru
    - Sdílení prostředků
  - Firmware
- **Hardware**
  - Procesor, paměť, periferie

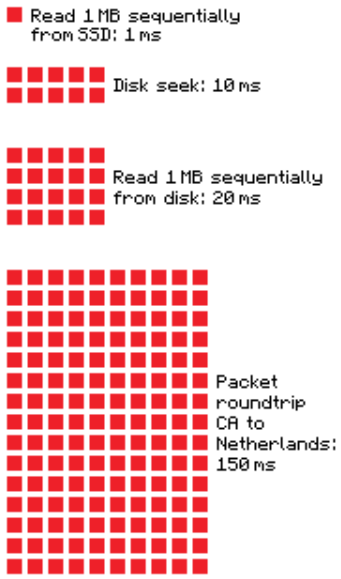


Latency Numbers Every Programmer Should Know





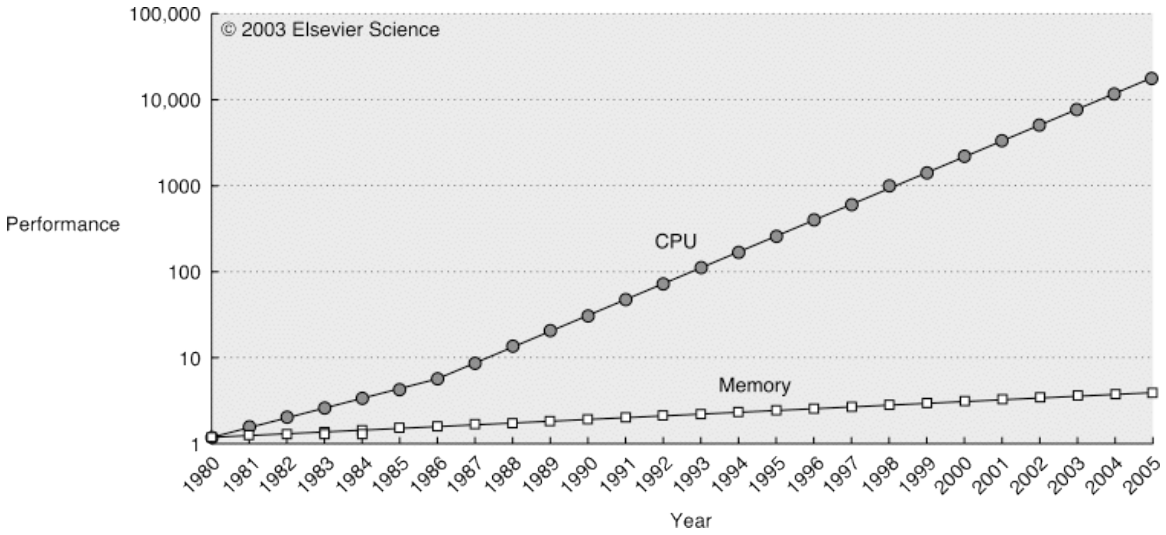




Source: https://gist.github.com/2841832

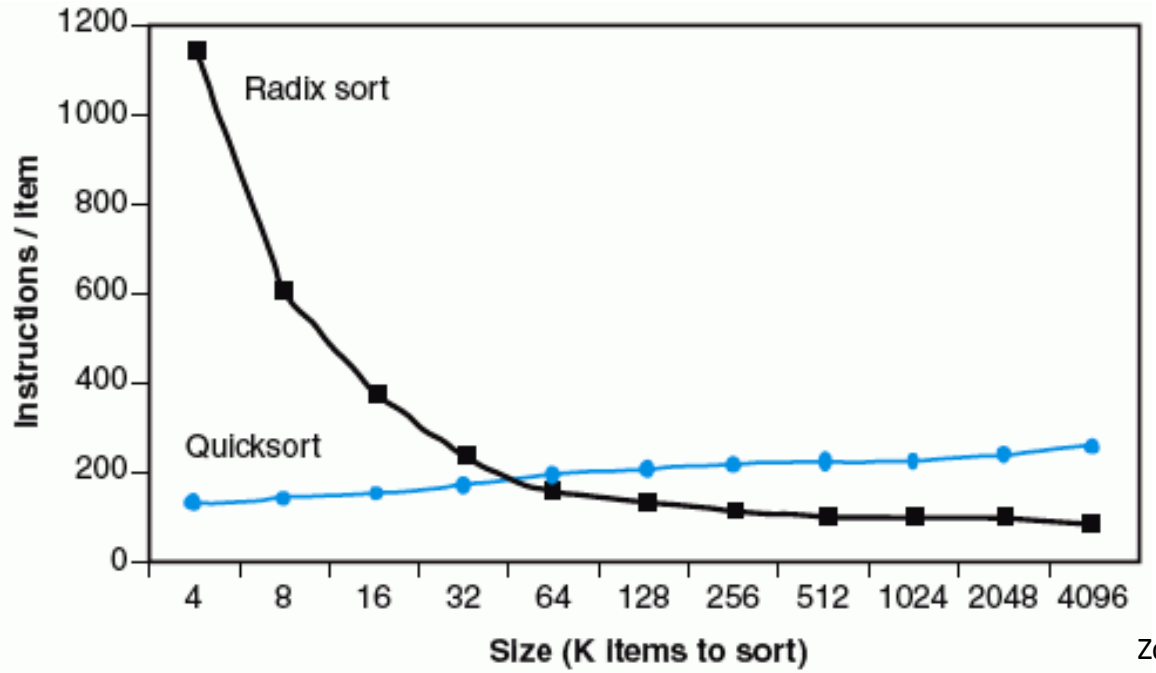
Paměťová zeď

- **Výkon procesorů omezen výkonem pamětí**
  - Výkon procesorů roste rychleji než výkon pamětí
    - Jednoduché operace trvají desetiny ns, přístup do paměti trvá desítky ns
    - Nedosažitelný cíl – kombinace: Paměť stejně rychlá jako procesor, dostatečná kapacita, rozumná cena



Proč je důležité vědět o CPU cache?

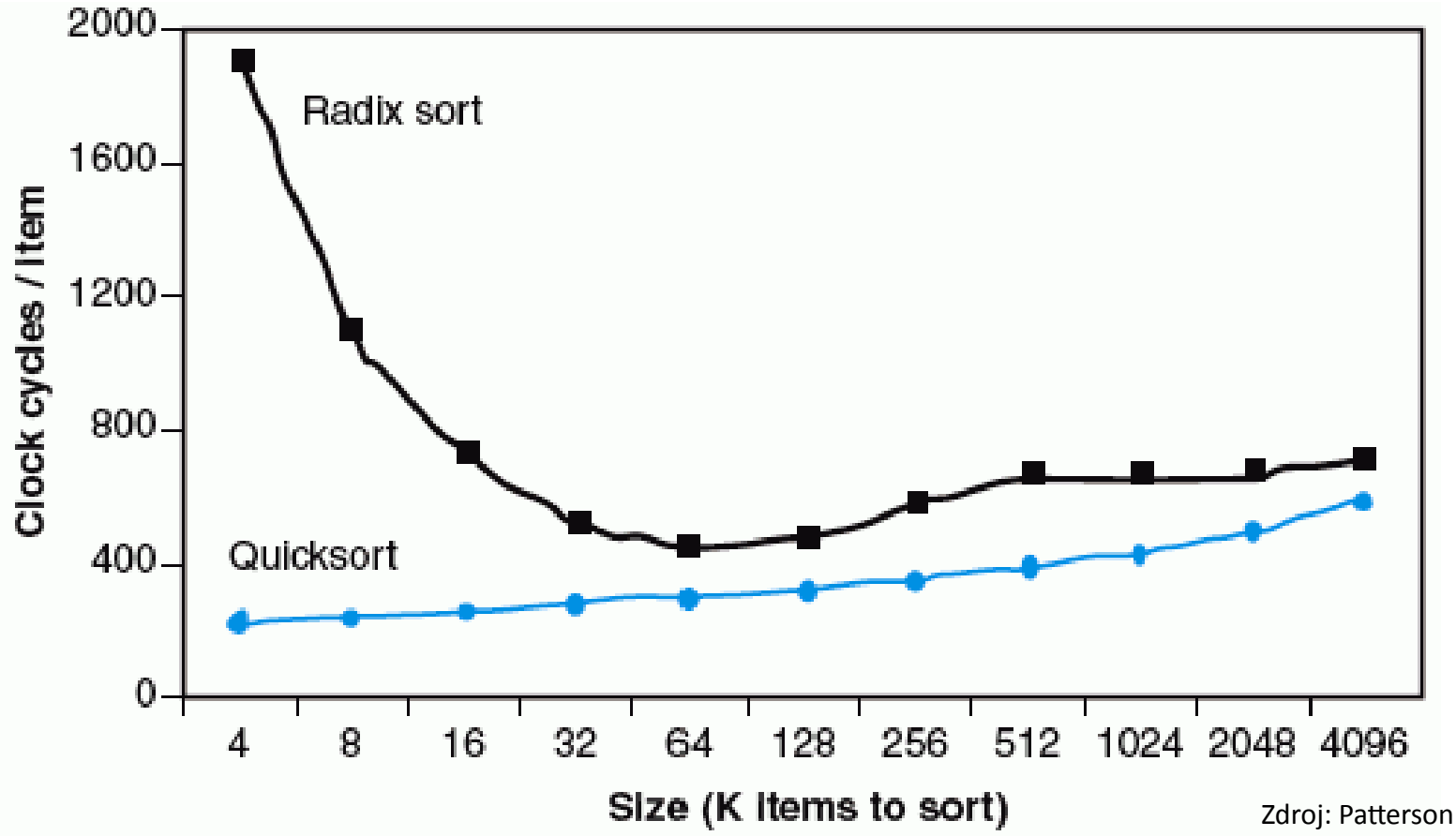
- ***Quick Sort* vs. *Radix Sort***
  - LaMarca, Ladner (1996)
  - **O(*n*×log *n*)** vs. **O(*n*)**
    - Teoreticky „není co řešit“



Zdroj: Patterson & Hennessy

Proč je důležité vědět o CPU cache? (2)

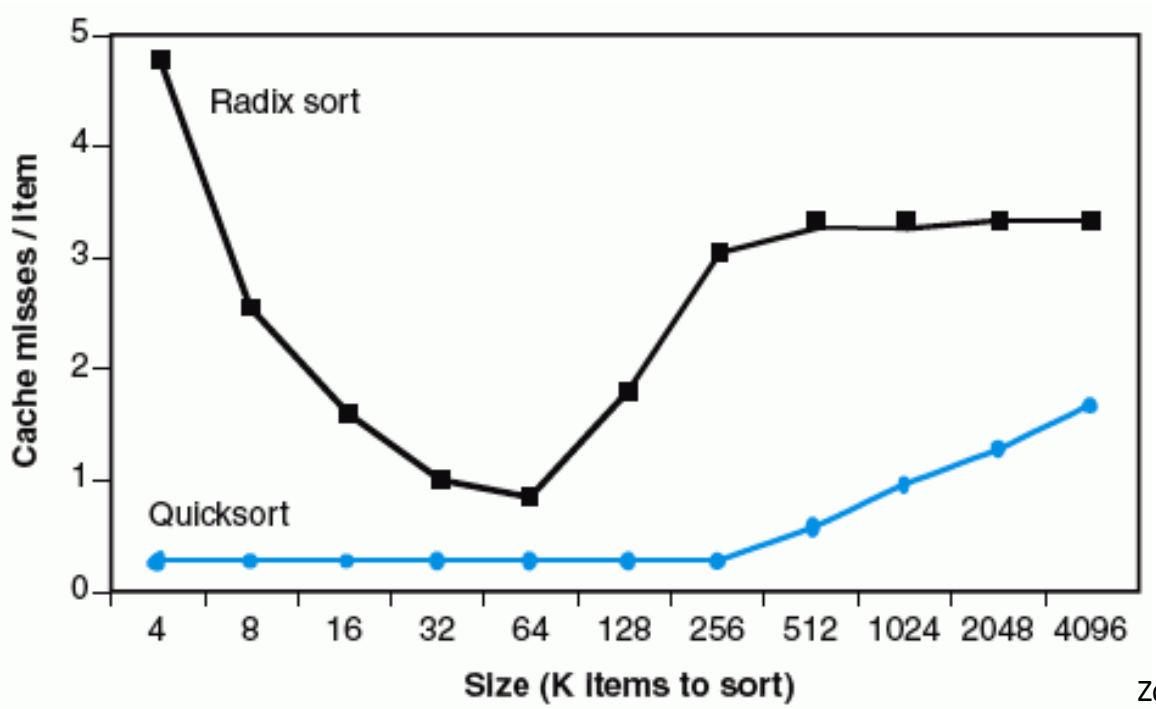
- **Jenže: Quick Sort se pro větší množství dat ukázal rychlejší ...**



Zdroj: Patterson & Hennessy

Proč je důležité vědět o CPU cache? (3)

- **Důvod**
  - Způsob přístupu k datům v implementaci algoritmu *Radix Sort* způsoboval příliš mnoho výpadků cache



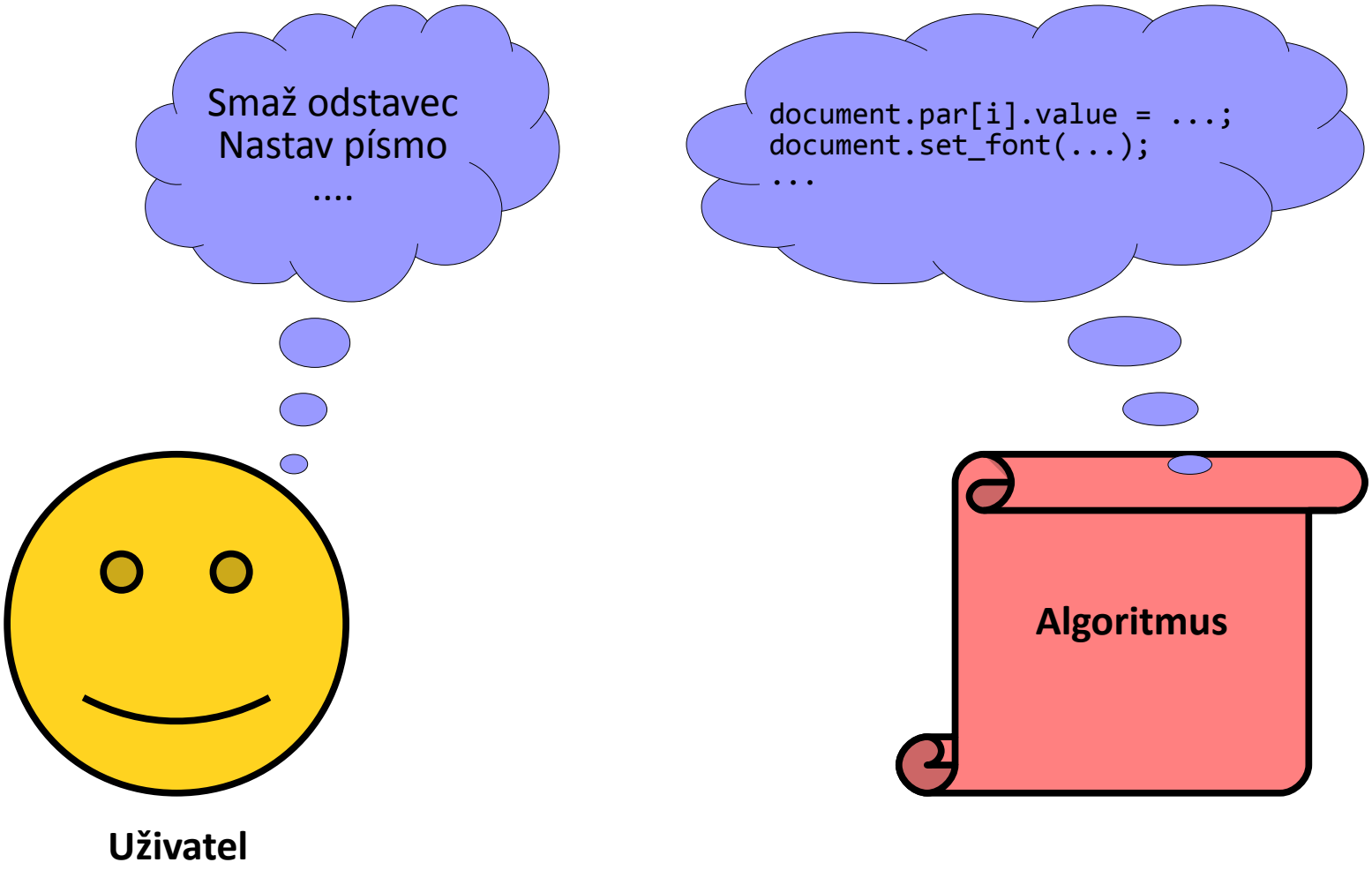
Zdroj: Patterson & Hennessy

Proč je důležité vědět o CPU cache? (4)

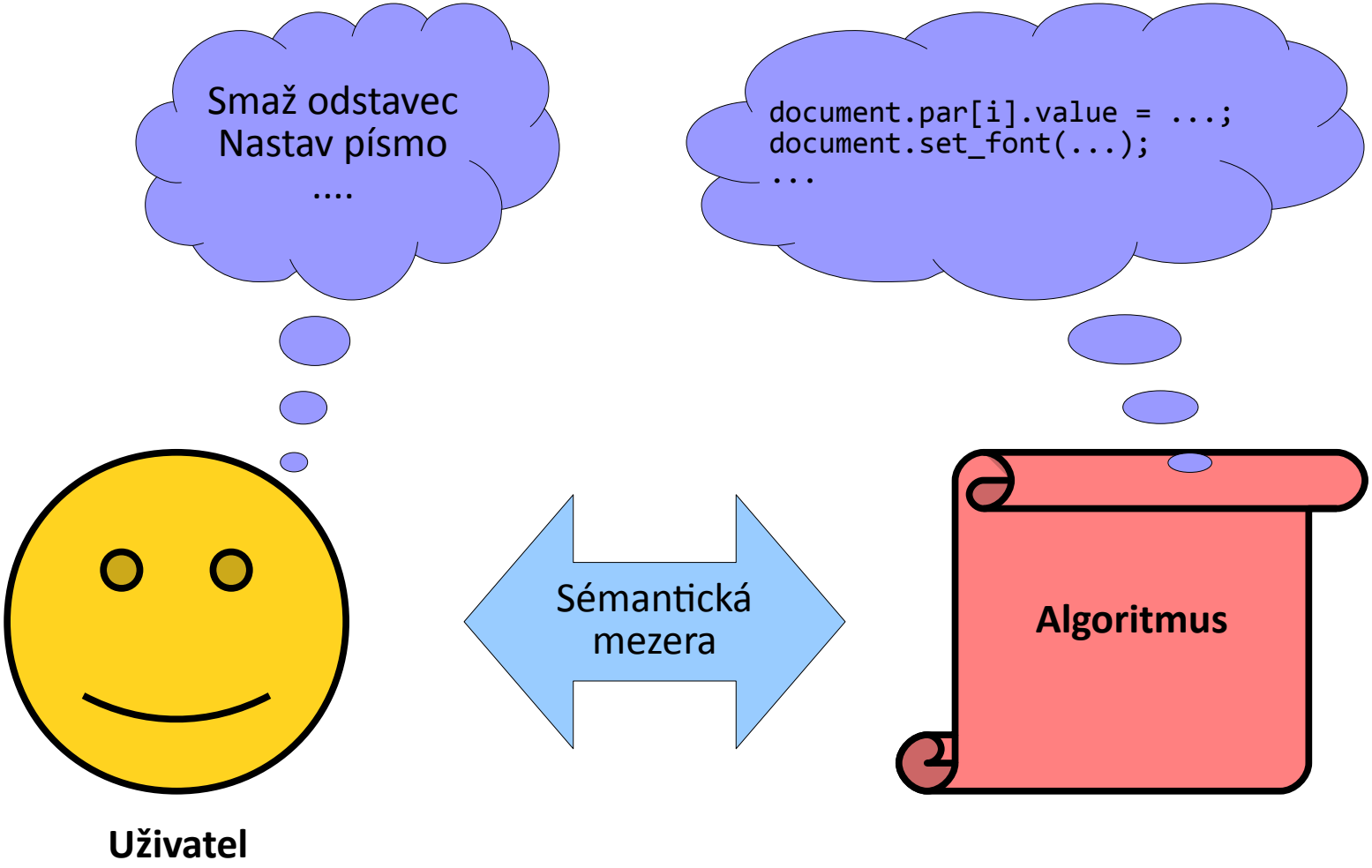
- **Řešení**
  - Úprava implementace algoritmu *Radix Sort*, aby pracoval s daty nejprve v rámci bloku paměti, který je již načtený v cache (řádku cache)

Abstrakce: Od uživatele k algoritmu

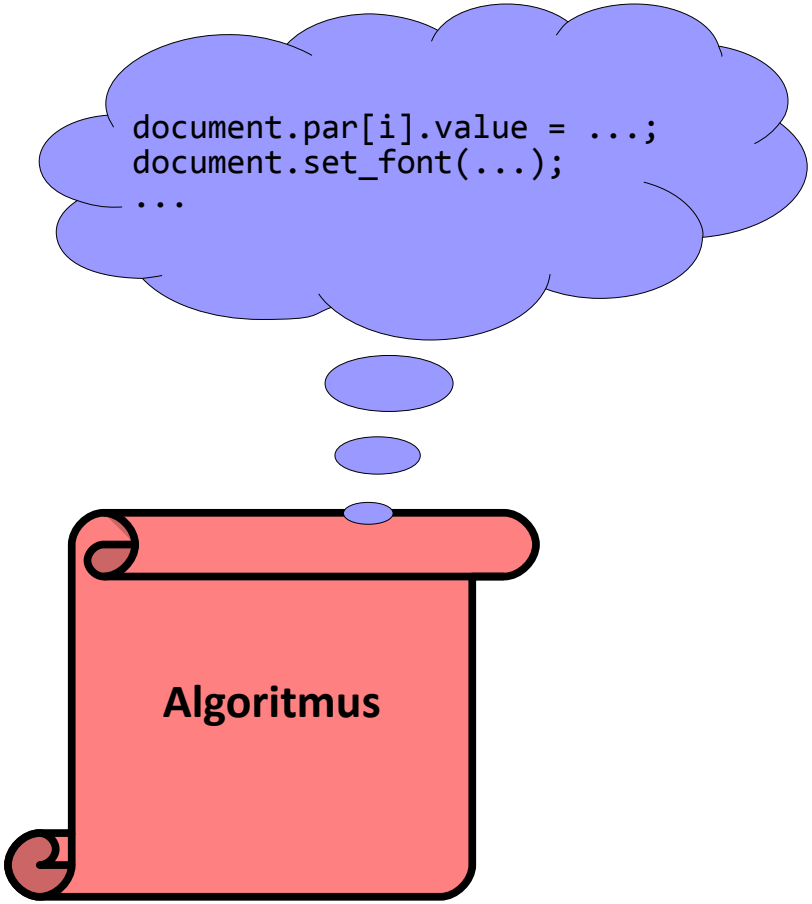
Abstrakce: Od uživatele k algoritmu



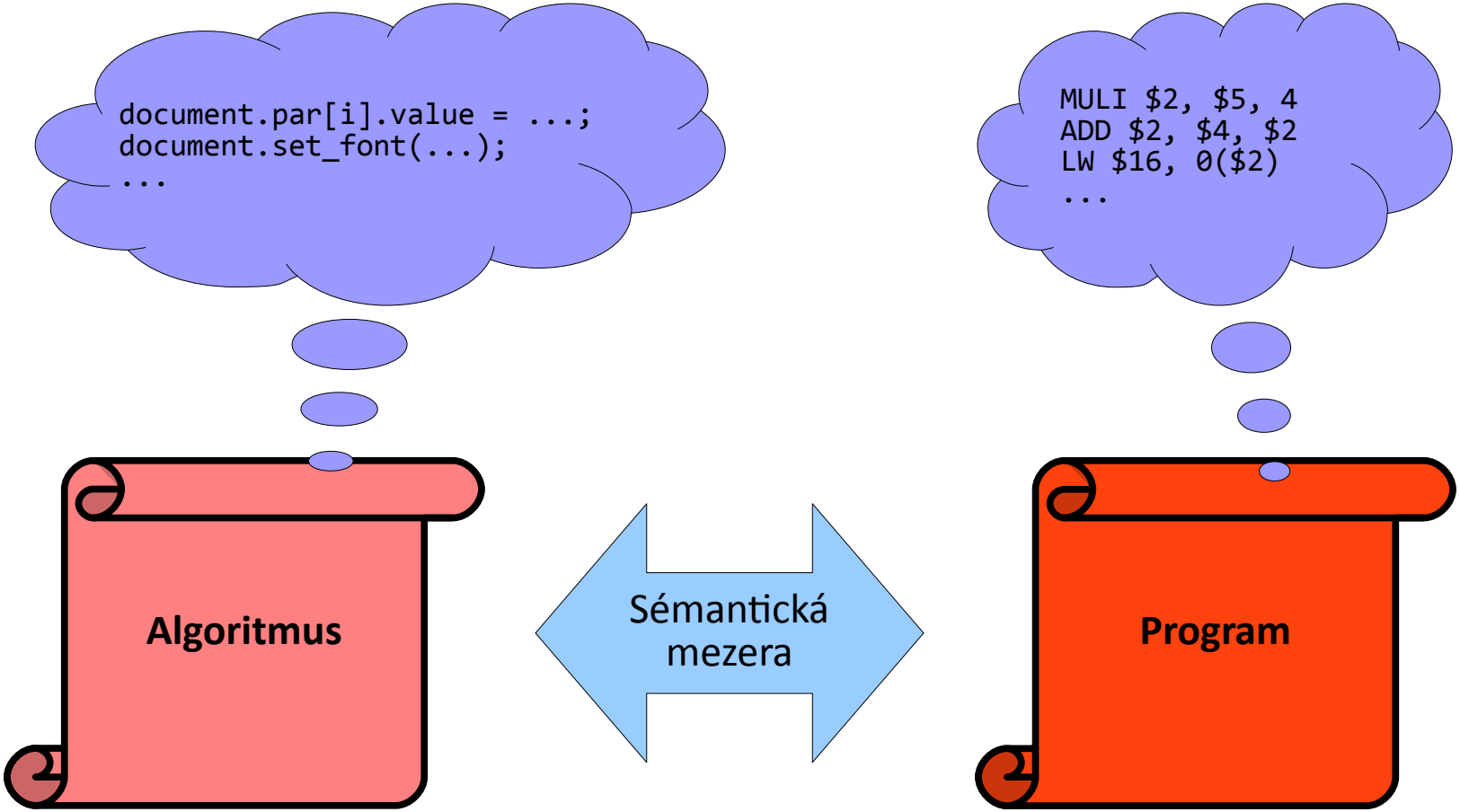
Abstrakce: Od uživatele k algoritmu



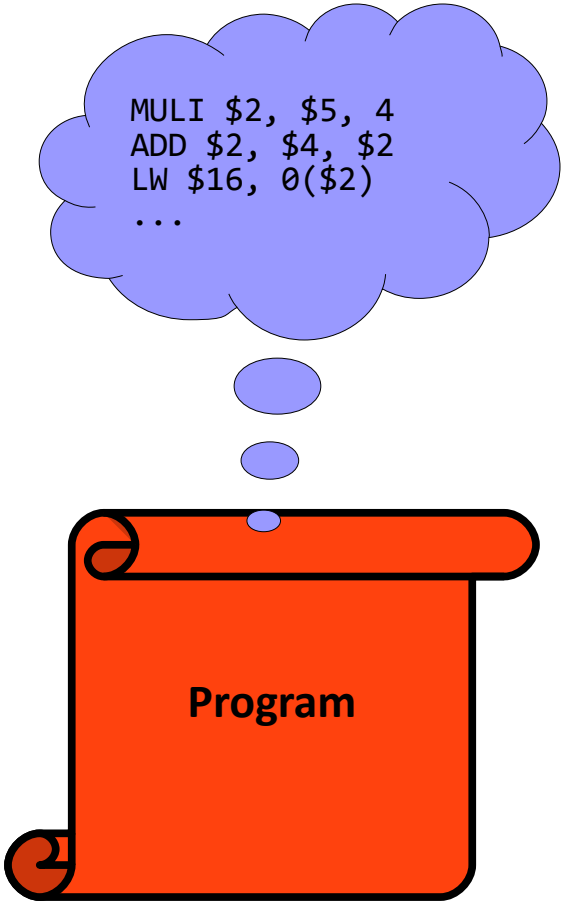
Abstrakce: Od algoritmu k programu



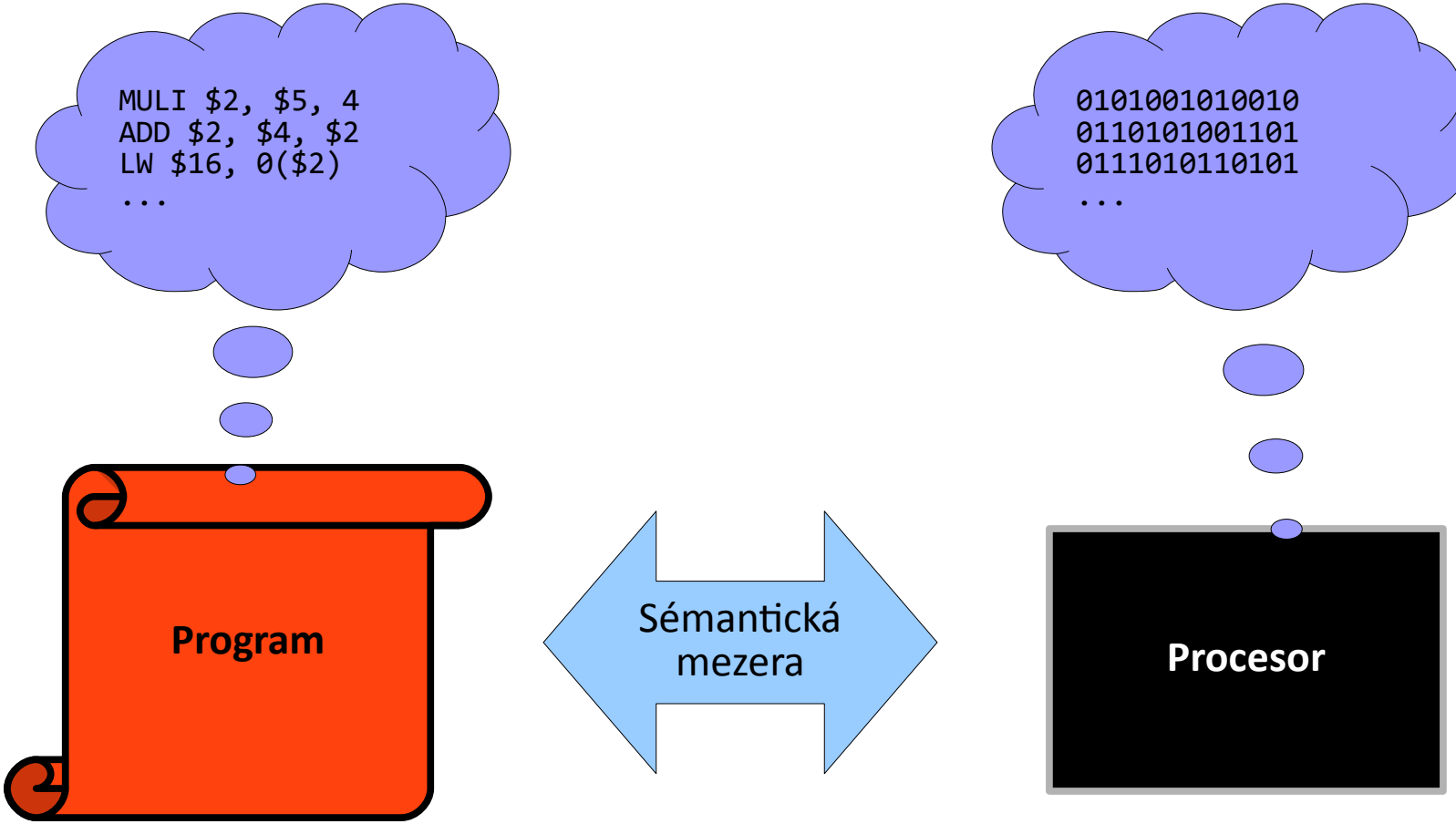
Abstrakce: Od algoritmu k programu



Abstrakce: Od programu ke kódu



Abstrakce: Od programu ke kódu



Abstrakce

- **Překonávání sémantických mezer**
  - Postup od konkrétního (technického) jazyka k abstraktnímu (obecnému) jazyku
  - Ideálně se zachováním přesnosti, ale využití šířeji definovaných pojmů a „zapouzdření“ vnitřních detailů
    - Stručnější a kompaktnější vyjádření
  - *„An abstraction is one thing that represents several real things equally well.“* (Edsger Dijkstra)

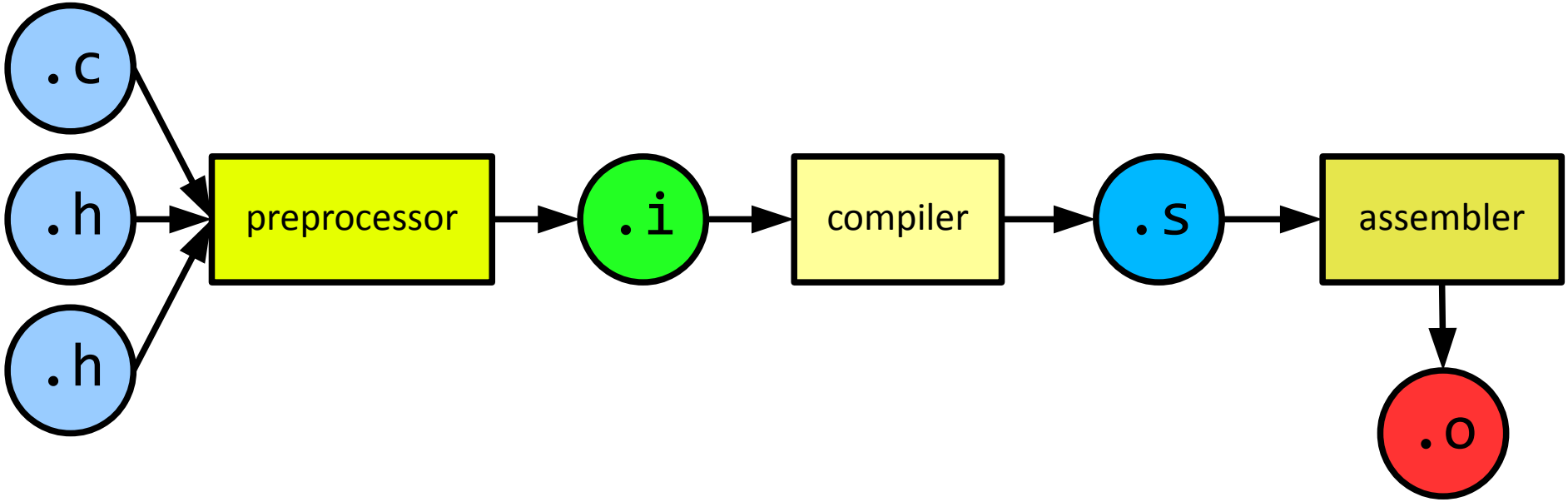
Abstrakce (2)

- **Vhodný jazyk pro danou úroveň abstrakce**
  - Řídící logika procesoru
    - Pomocí jednotky ALU sečti hodnotu z registru A a hodnotu z registru B, výsledek ulož do registru C
  - Strojový kód: instrukce (slova) nad abecedou {0, 1}
    - `1000110010100000`
  - Assembler: symbolický zápis instrukcí programu
    - `add R2, R3, R1`
  - Vyšší programovací jazyk: symbolický zápis algoritmu
    - `fruits := apples + oranges`

Abstrakce (3)

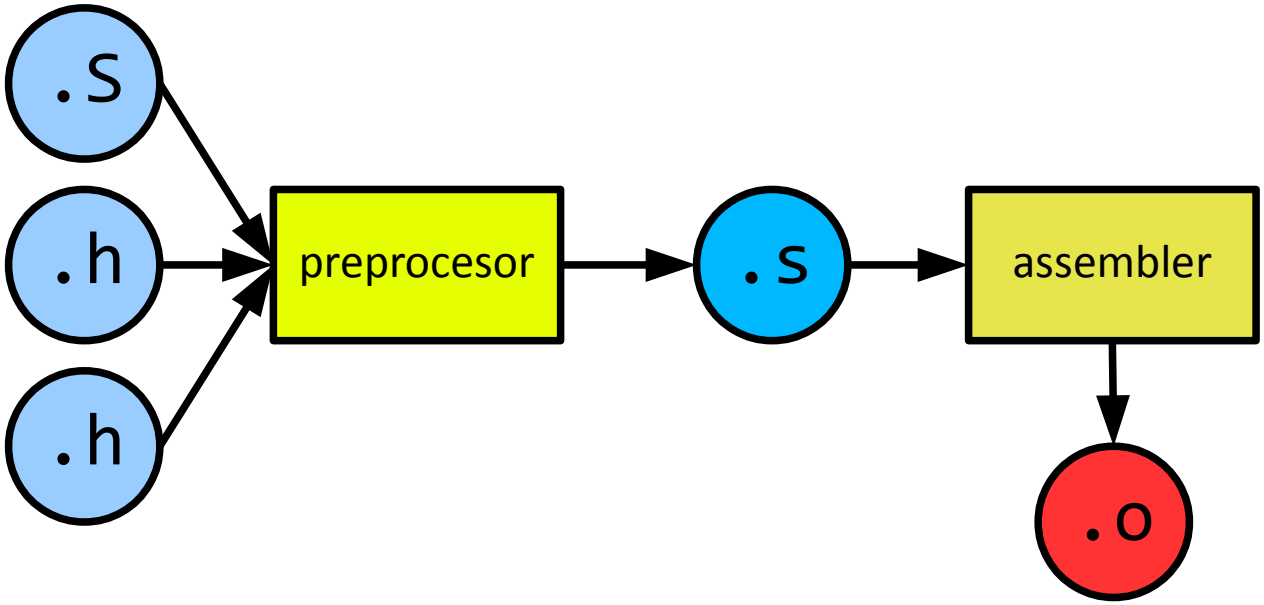
- **Vhodný jazyk pro danou úroveň abstrakce**
  - Překlad mezi jazyky = překonávání sémantické mezery
  - Jazyk vyšší úrovně = vyšší produktivita člověka
    - Doménově-specifické jazyky
  - Jazyk nižší úrovně = vyšší produktivita stroje
    - Strojový kód
  - Překladač
    - Překlad z vyššího programovacího jazyka do jazyka nižší úrovně (často až do symbolického zápisu instrukcí konkrétního procesoru)
  - Assembler
    - Překlad ze symbolického zápisu instrukcí do binárního kódu vykonatelného konkrétním procesorem

Překladač vyššího jazyka



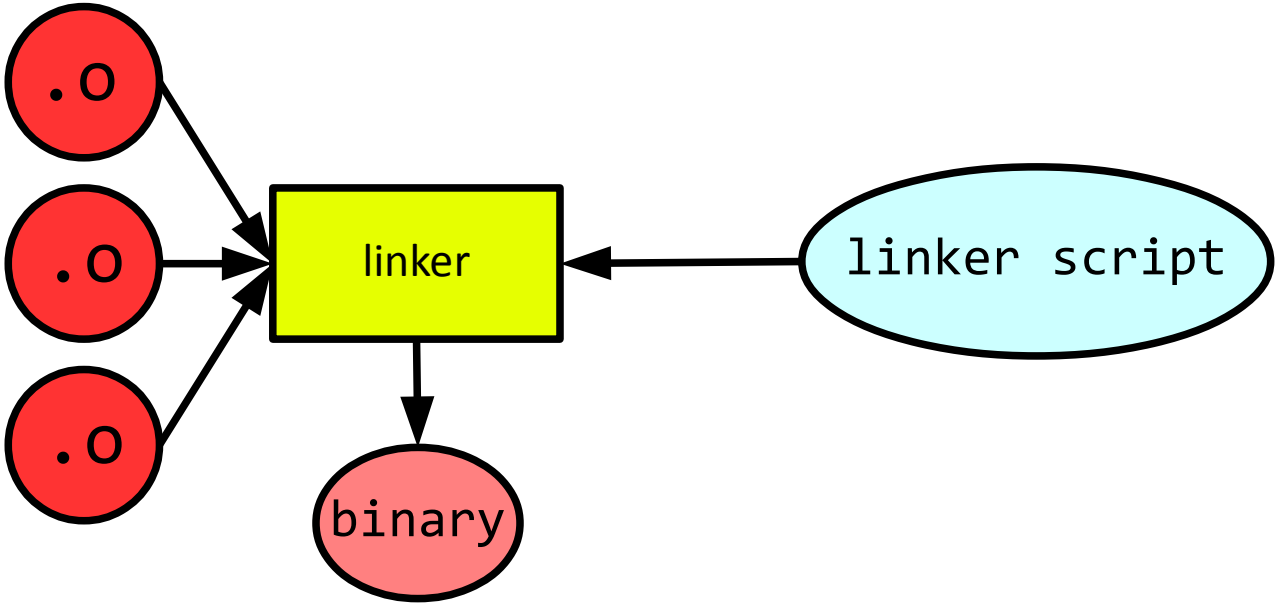

```
cc -c -o output.o input.c
as -o output.o input.s
```

Assembler s preprocessorem




```
cc -c -o output.o input.S
```

Linkování




```
ld -T link.ld -o output.bin input0.o input1.o
```

Linkování (2)

input0.c

```
void global_fnc01() { }
void global_fnc02() { }

int global_int;
void *global_ptr;

int another_symbol __attribute__((section("another_section")));
```

input0.o

```
.text
    global_fnc01
    global_fnc02
.bss
    global_int
    global_ptr
another_section
    another_symbol
```

Linkování (3)

link.ld

```
SECTIONS {
    .output 0x80000000 : {
        *(.text)
        *(.bss)
        *(another_section)
        _output_end = .;
    }
}
```

output.bin

```
.output (displacement: 0x80000000)
    global_fnc01
    global_fnc02
    global_int
    global_ptr
    another_symbol
    _output_end
```

Užitečné přepínače překladače

- **GCC**

  `-ffreestanding`
  - Překlad bez standardní knihovny a funkce `main()`

  `-nostdlib`
  - Bez prohledávání systémové cesty knihoven

  `-nostdinc`
  - Bez prohledávání systémové cesty hlavičkových souborů

  `-fno-builtin`
  - Nepoužívat vestavěné funkce překladače (není-li explicitně použit prefix `__builtin_`)

Užitečné atributy překladače

- **GCC**

`__attribute__((packed))`

- Struktura bez standardního zarovnání prvků

`__attribute__((may_alias))`

- Proměnná daného datového typu může být zároveň jiným datovým typem

`__attribute__((aligned(`*n*`)))`

- Proměnná se zarovnáním v paměti na *n* bytů

`__attribute__((section("`*sekce*`")))`

- Symbol umístěn ve vstupní sekci *sekce*

`__attribute__((returns_twice))`

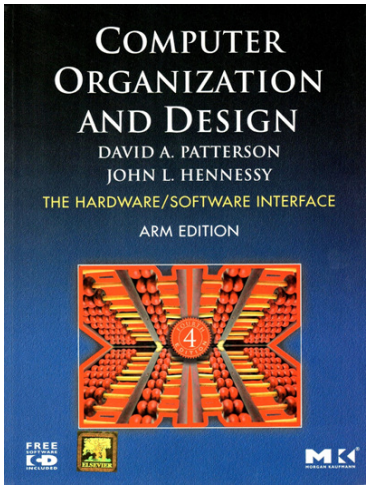
- Pro implementaci funkcí obnovujících kontext procesoru

Užitečné nástroje

- **`objdump`**
  - Výpis informací o objektových a binárních souborech
    - `-x`
      - Výpis všech hlaviček
    - `-d`
      - Disassemblování spustitelných sekcí
    - `-D`
      - Disassemblování všech sekcí
    - `-S`
      - Disassemblování proložené řádky zdrojového kódu
      - Potřebuje debuggovací informace (`gcc -g`)

Kde se to naučit pořádně?

- **Kurzy na MFF UK** (a obdobné na jiných univerzitách)
  - Principy počítačů
  - Architektura počítačů
  - Operační systémy
  - Principy překladačů
  - Crash dump analýza
- **Literatura**
  - David Patterson, John Hennessy: *Computer Organization and Design*
- **On-line**
  - Série článků *Co se děje v počítači* na Root.cz

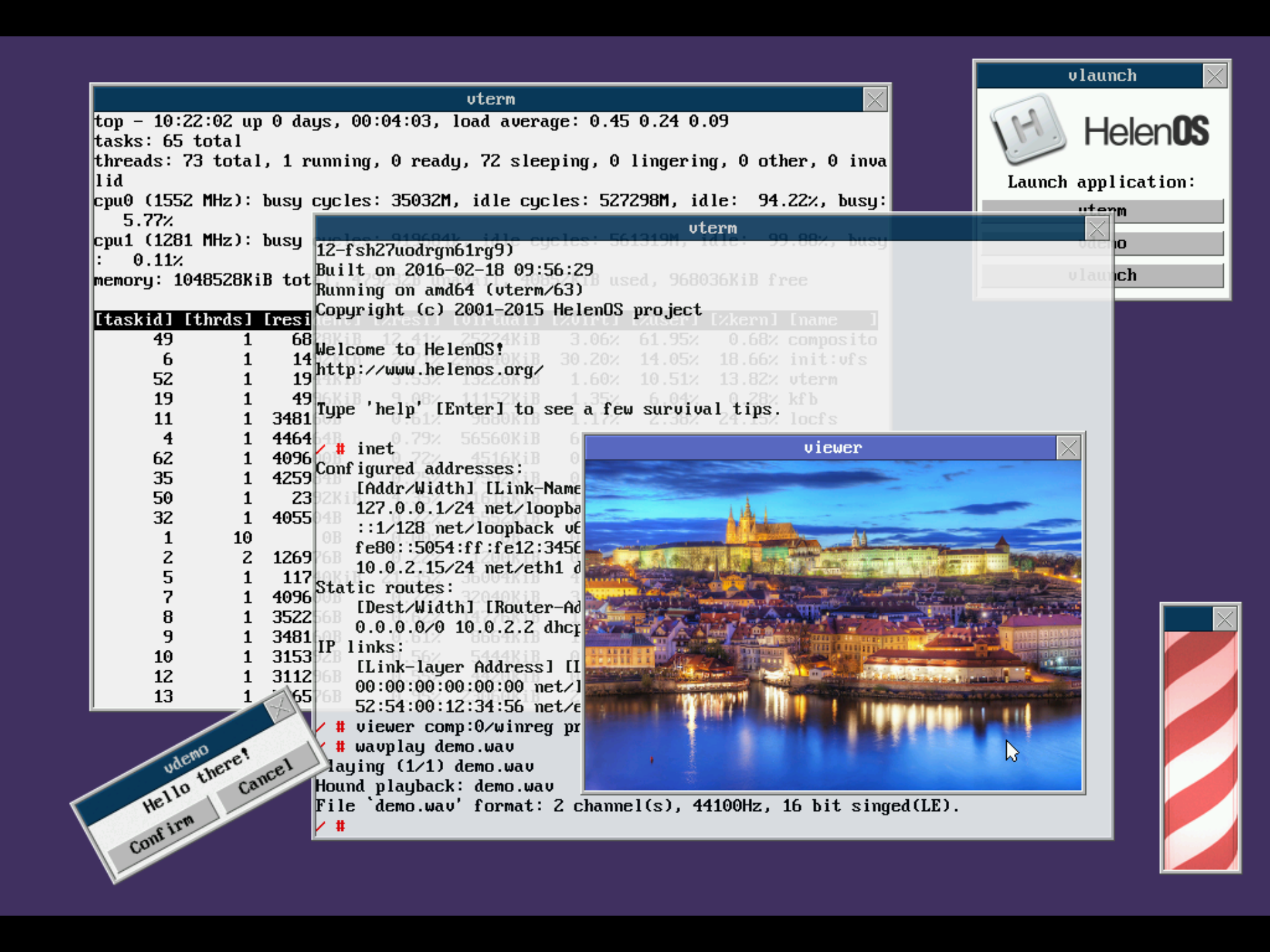


HelenOS

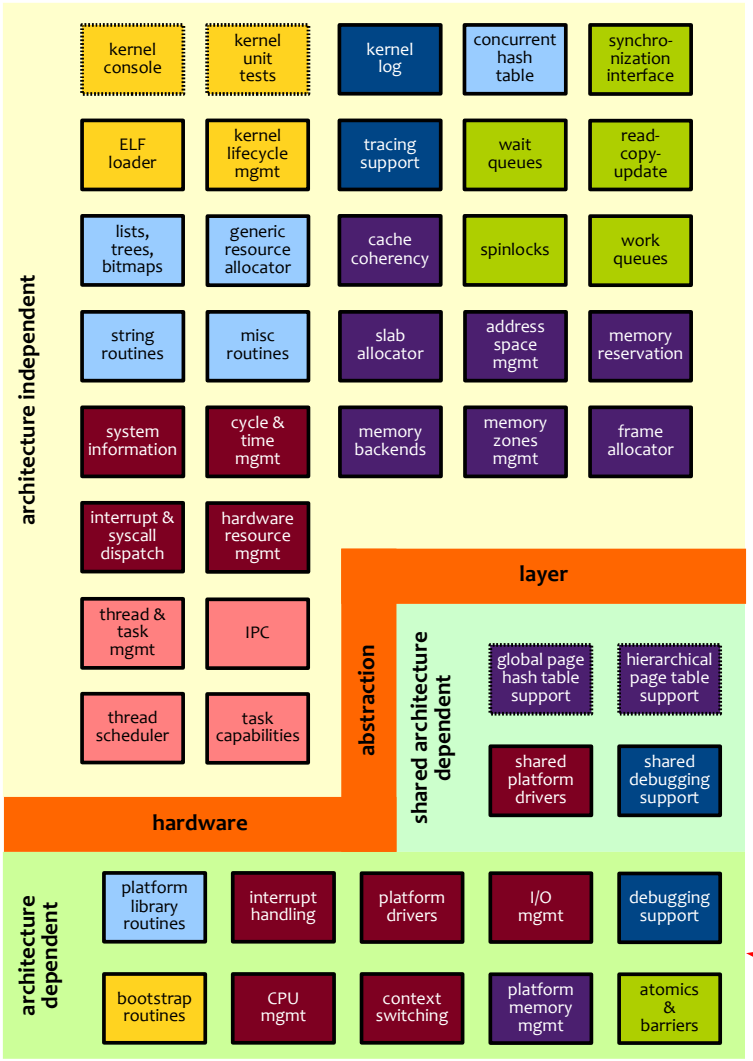
Stručná charakteristika

HelenOS

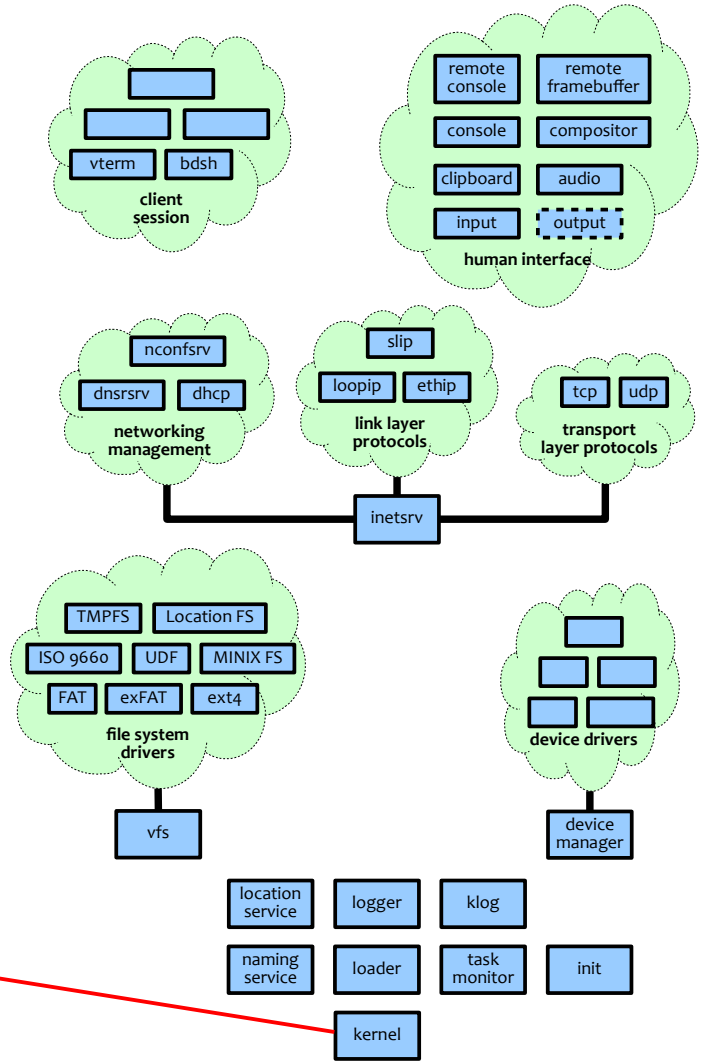
open source general-purpose multiplatform microkernel multiserver operating system implemented from scratch


vterm
top - 10:22:02 up 0 days, 00:04:03, load average: 0.45 0.24 0.09
tasks: 65 total
threads: 73 total, 1 running, 0 ready, 72 sleeping, 0 lingering, 0 other, 0 invalid
cpu0 (1552 MHz): busy cycles: 35032M, idle cycles: 527298M, idle: 94.22%, busy: 5.77%
cpu1 (1281 MHz): busy
: 0.11%
memory: 1048528KiB tot
[taskid] [thrds] [resi
49 1 68
6 1 14
52 1 19
19 1 49
11 1 3481
4 1 4464
62 1 4096
35 1 4259
50 1 23
32 1 4055
1 10
2 2 1269
5 1 117
7 1 4096
8 1 3522
9 1 3481
10 1 3153
12 1 3112
13 1
vterm
12-fsh27uodrgn61rg9)
Built on 2016-02-18 09:56:29
Running on amd64 (vterm/63)
Copyright (c) 2001-2015 HelenOS project
Welcome to HelenOS!
http://www.helenos.org/
Type 'help' [Enter] to see a few survival tips.
/ # inet
Configured addresses:
[Addr/Width] [Link-Name
127.0.0.1/24 net/loopba
::1/128 net/loopback v6
fe80::5054:ff:fe12:3456
10.0.2.15/24 net/eth1 d
Static routes:
[Dest/Width] [Router-Ad
0.0.0.0/0 10.0.2.2 dhcp
IP links:
[Link-layer Address] [I
00:00:00:00:00:00 net/l
52:54:00:12:34:56 net/e
/ # viewer comp:0/winreg pr
/ # wavplay demo.wav
laying (1/1) demo.wav
Hound playback: demo.wav
File `demo.wav' format: 2 channel(s), 44100Hz, 16 bit singed(LE).
/ #
viewer
vlaunch
HelenOS
Launch application:
vterm
vdemo
Hello there!
Cancel
Confirm

HelenOS: Architektura v kostce



Kernel subsystems



System components

Q&A

References

- Slide 2
  - Harvard Mark I, © IBM 1944 (cited under fair use doctrine)
- Slide 6
  - Tron: Legacy, © Walt Disney Studios Motion Pictures 2010 (cited under fair use doctrine)
- Slide 9
  - Data compiled by Jonas Bonér (https://gist.github.com/jboner/2841832), Jeffrey Dean and Peter Norvig; rendering by ayshen (https://github.com/ayshen)
- Slide 10
  - Roth A., Martin M.: *CIS 371 – Computer Organization and Design*, University of Pennsylvania, Department of Computer and Information Science, 2009; data and rendering © Elsevier Science 2003
- Slide 11, 12, 13, 33
  - D. A. Patterson, J. L. Hennessy: *Computer Organization and Design*, Morgan Kaufmann, ISBN 978-0123747501, 2011