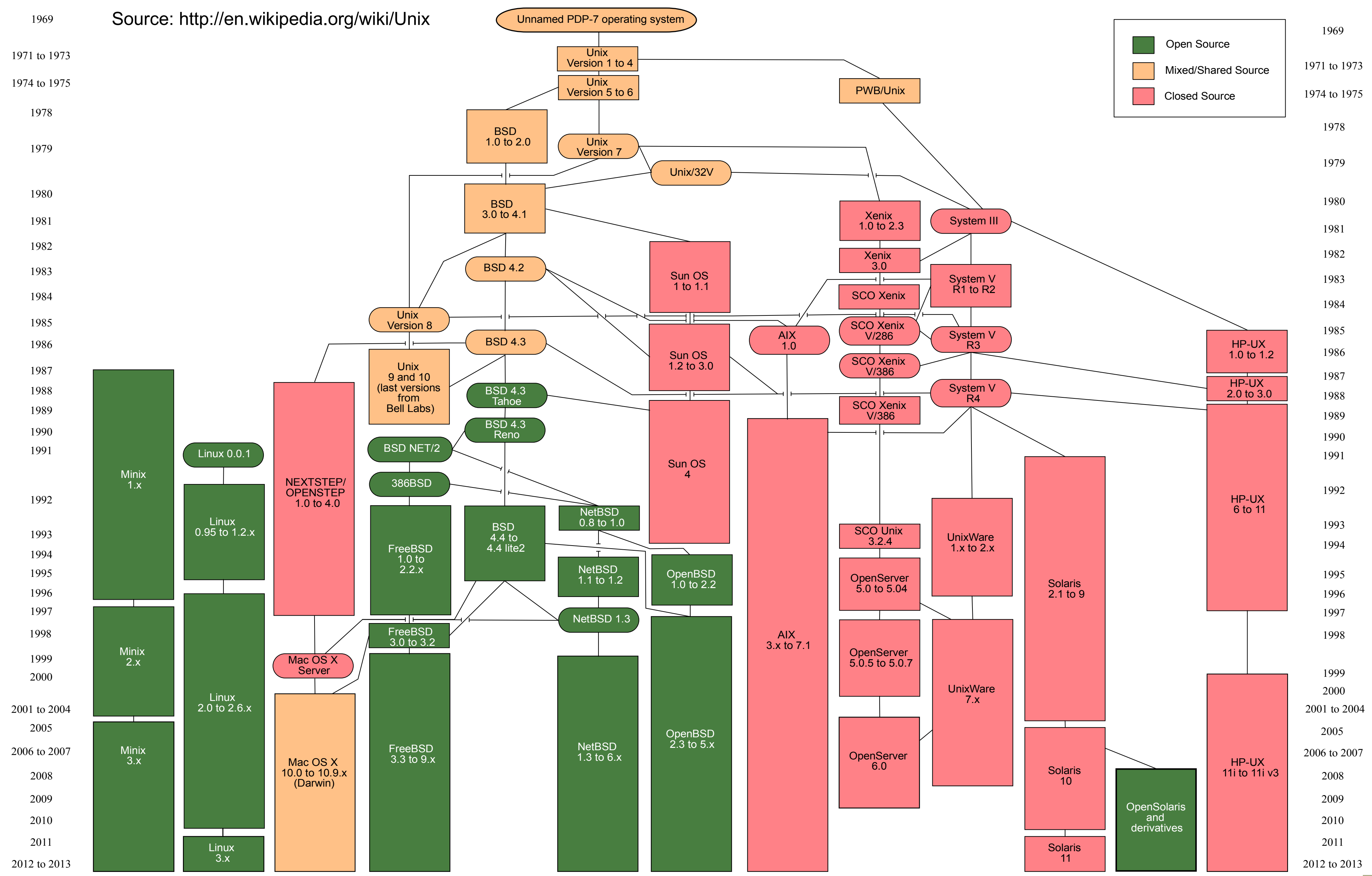
Unix structure, history, and properties.

Shell and command-line parsing.

Department of Computer Systems FIT, Czech Technical University in Prague

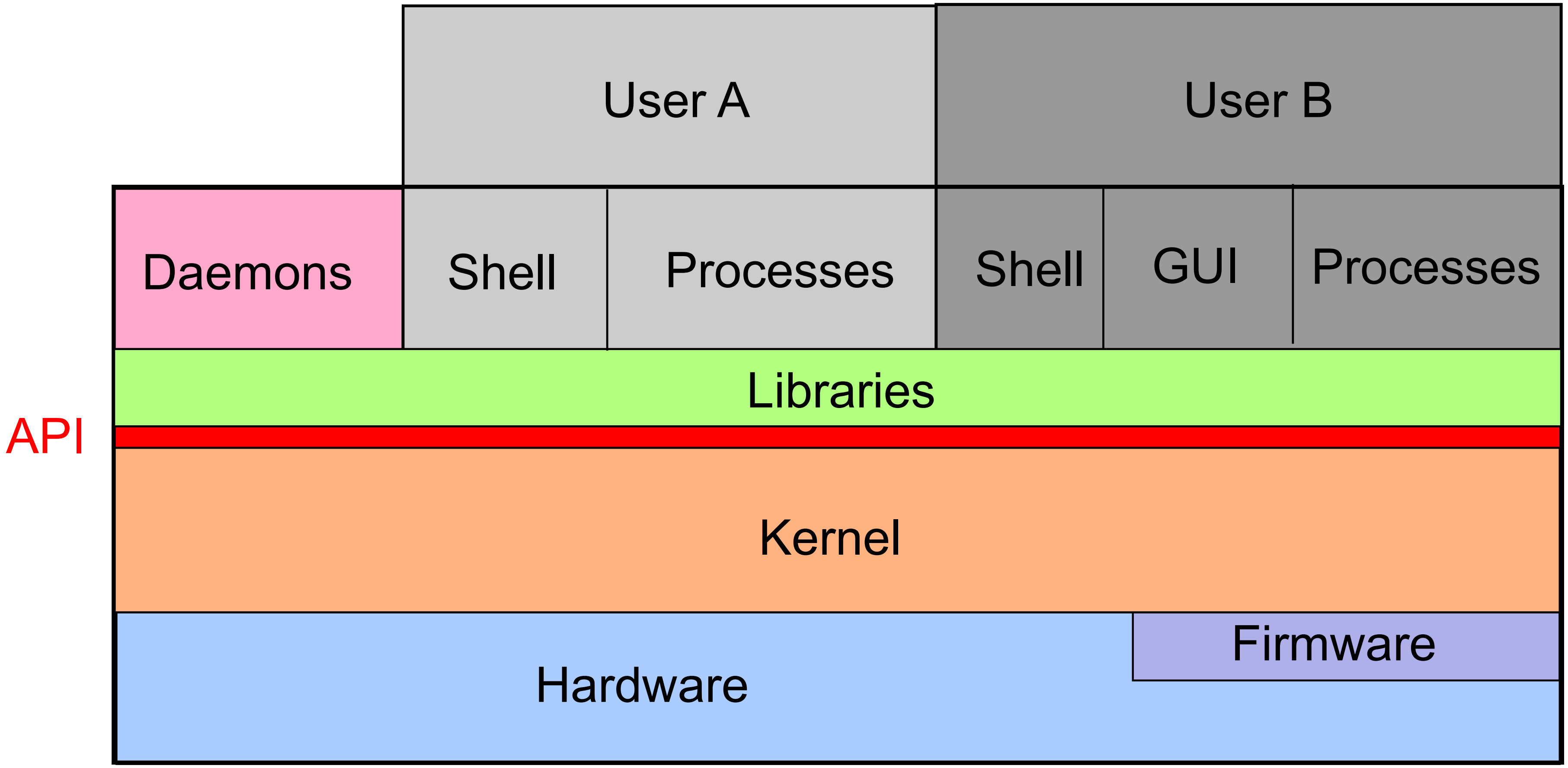
UNIX - history

Source: http://en.wikipedia.org/wiki/Unix



🟩	Open Source
🟧	Mixed/Shared Source
🟥	Closed Source

Unix - structure



UNIX - properties

- **Portable**
  - 90% of kernel is written in C.
- **Multi-user**
- **Multitasking, time-sharing**
- **Multithreading**
- **Symmetric Multi Processing (SMP)**
- **CLI**
- **IO redirection**
- **Hierarchical FS**
- **TCP/IP networking, NFS,…**
- **GUI**
  - X-Windows
  - Window managers - CDE, GNOME, KDE,…

Shell – command interpreter

- Interface between user and kernel.
- **Environment setting**
  - Shell variables can define application behavior.
- **CLI**
  - Command-line parsing (e.g. find and replace special symbols)
  - Command execution.
- **Shell scripts**
  - Shell executes commands from file (scripts).
  - Script = Unix commands + control structures (e.g. loops, if/else…)

Bourne shells

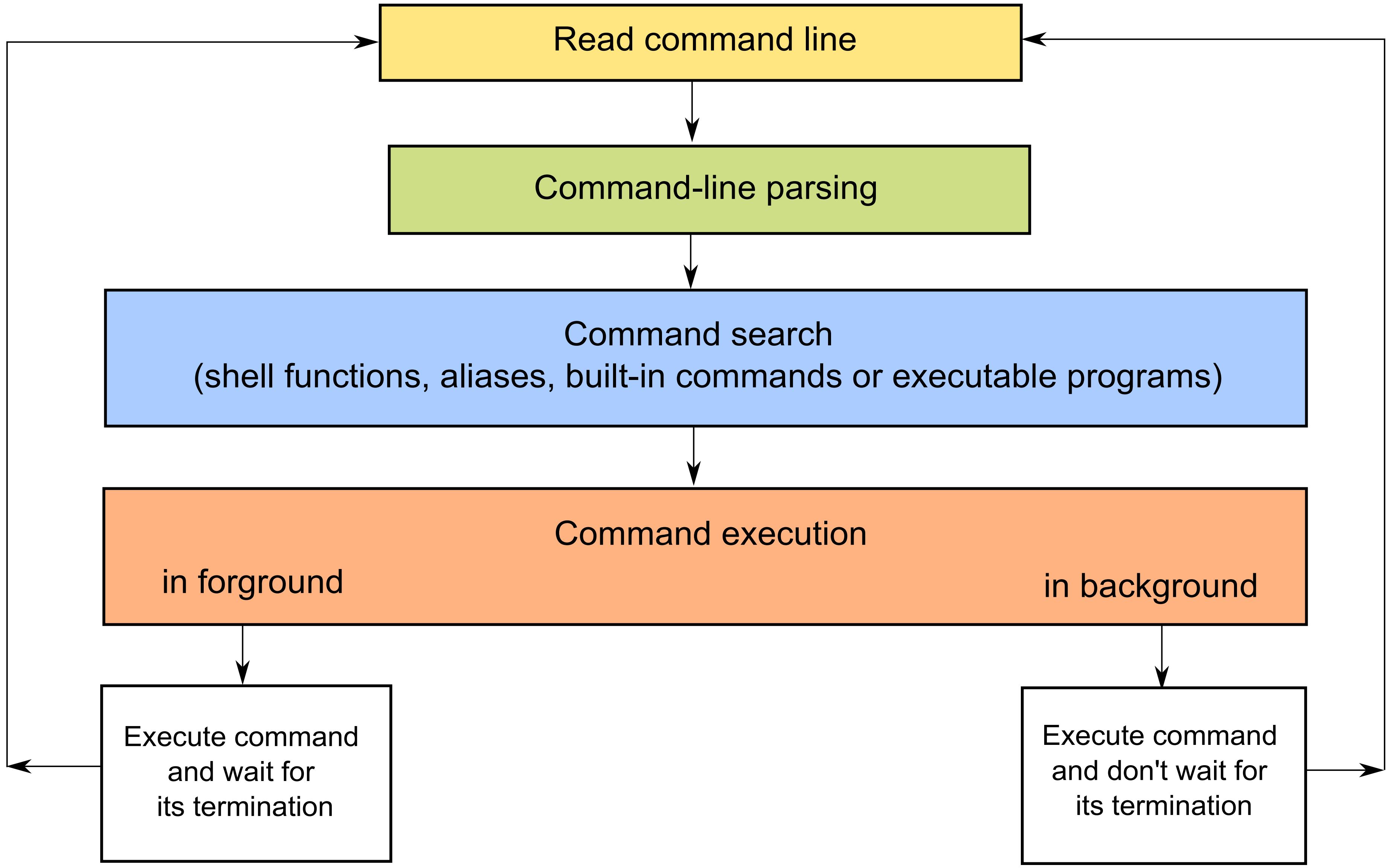
Name	File	Properties
Bourne shell	/bin/sh	basic
Korn shell	/bin/ksh	command history, job control, aliases,…
Bourne again shell	/bin/bash	like ksh but more user friendly

C shells

Name	File	Properties
C shell	/bin /csh	like ksh
Toronto C shell	/bin/tcsh	like csh, but more user friendly

- More information about shell we can find in Unix manual (e.g. `man bash`).
- In this modules we concentrate to Bourne shells.

Command-line parsing


Read command line
Command-line parsing
Command search
(shell functions, aliases, built-in commands or executable programs)
Command execution
in forground
in background
Execute command and wait for its termination
Execute command and don't wait for its termination

Command line syntax

- **Variables**

```
<prompt> <variable_name>=<value>
```

`<prompt>`

- Prompt is printed by shell.
- Value of prompt is defined by the shell variable PS1.

`<variable_name>`

- Variable name is identifier.
- No spaces around symbol **`=`**.
- Shell assigns the value to the variable.

`<value>`

- By default it is string.

Command line syntax

- **Simple commands**

```
<prompt> <command_name> <options> <arguments>
```

`<command_name>`

- It defines which program will be executed (which).
- It can be only name or path to the file (relative/absolute).

`<options>`

- They can modify the behavior of command (how).

`<arguments>`

- They specify the input date (what).
-

- Command name, options and arguments are available
  - in script by variables `$#`, `$0`, `$1`, `$2`, …
  - in C program by variables `argc` , `argv[0]`, `arg[1]`, …

Examples

```
ls

ls /etc

ls –la /etc


B=`ypcat passwd | cut -d: -f1`

echo $B

echo "$B"


export LC_TIME=cs_CZ ; /usr/bin/echo "Today is \c" ; date '+%A %d.%m.%Y‘


ypcat passwd | grep "student" | grep -v "docasne konto" | \
    sort -t':' -k3,3n| tail -1 | cut -d: -f 5 | cut -d'  ' -f1,2
```

Is it clear???   Too simple???

Examples

Little bit more complicated?

```
echo PID FD EXEC FILENAME; PID=$(pgrep ''); pfiles $PID | awk
  'BEGIN { fd=-1; }  /^[0-9]/ { if (fd>=0) { print pid, fd,
  exec; fd=-1; }; pid=substr($1,0,length($1)-1); exec=$2; } /
  ^ *[0-9]*: / { if (fd>=0) { print pid, fd, exec; fd=-1; };
  if ($2=="S_IFREG") { fd=substr($1,0,length($1)-1); } } /^
  *\// { fd=-1; }' | while read pid fd exec; do echo $pid $fd
  $exec $(echo 0t$pid ::pid2proc \| ::fd $fd \| ::print
  file_t f_vnode \| ::vnode2path | mdb -k 2>/dev/null); done
```