



Open Source Programování

<http://rtime.felk.cvut.cz/osp/>

Pavel Píša

<pisa@fel.cvut.cz>

<http://cmp.felk.cvut.cz/~pisa>

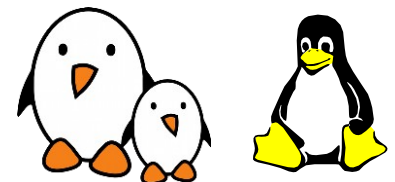
Michal Sojka

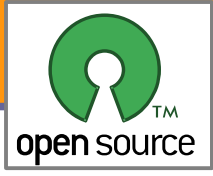
František Vacek

DCE FEL ČVUT



© Copyright 2004-2010, Pavel Píša, Michal Sojka, František Vacek,
Free-Electrons.com, GNU.org, kernel.org, Wikipedia.org
Creative Commons BY-SA 3.0 license Latest update: 24. IV 2013





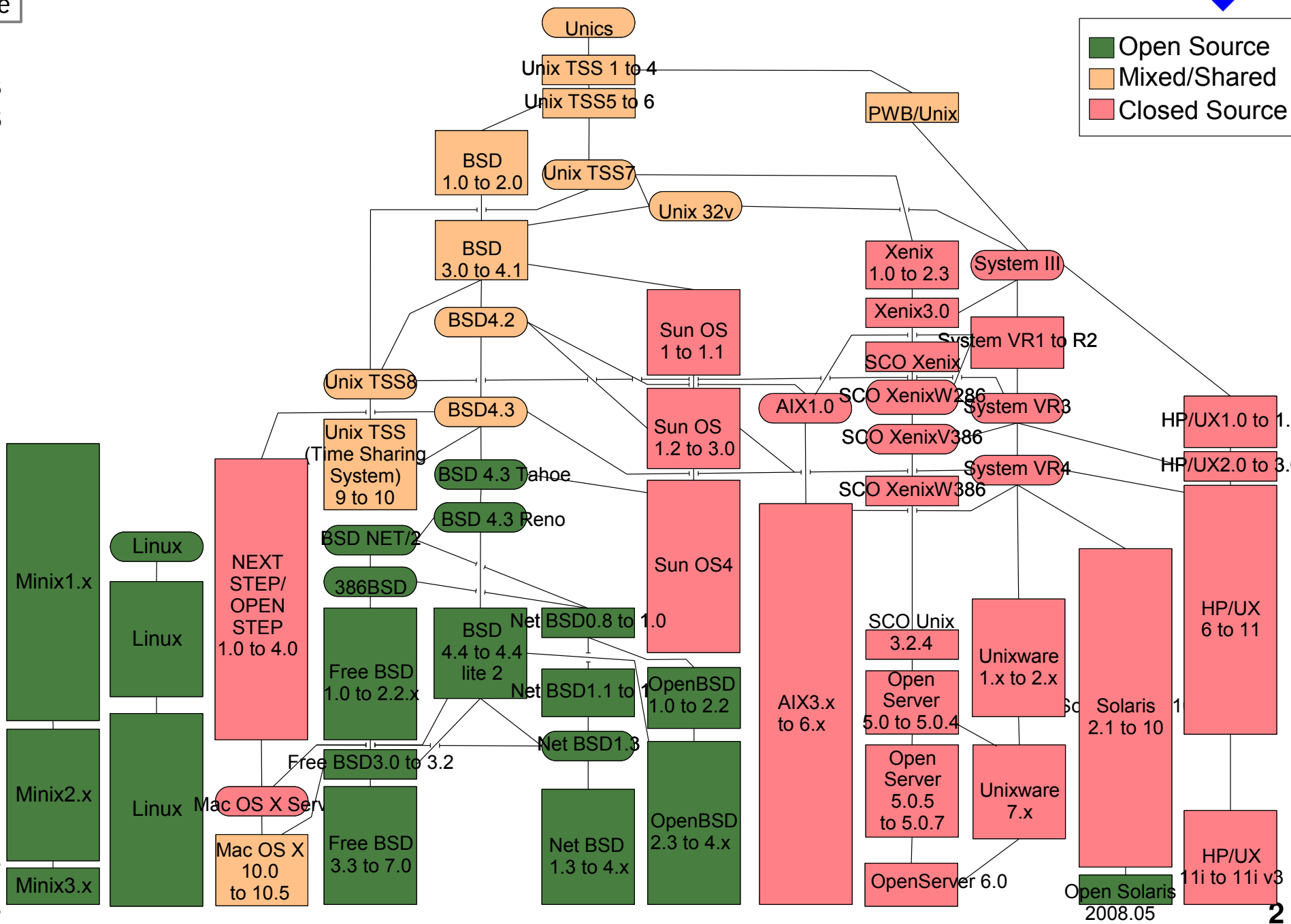
A jak to bylo dál

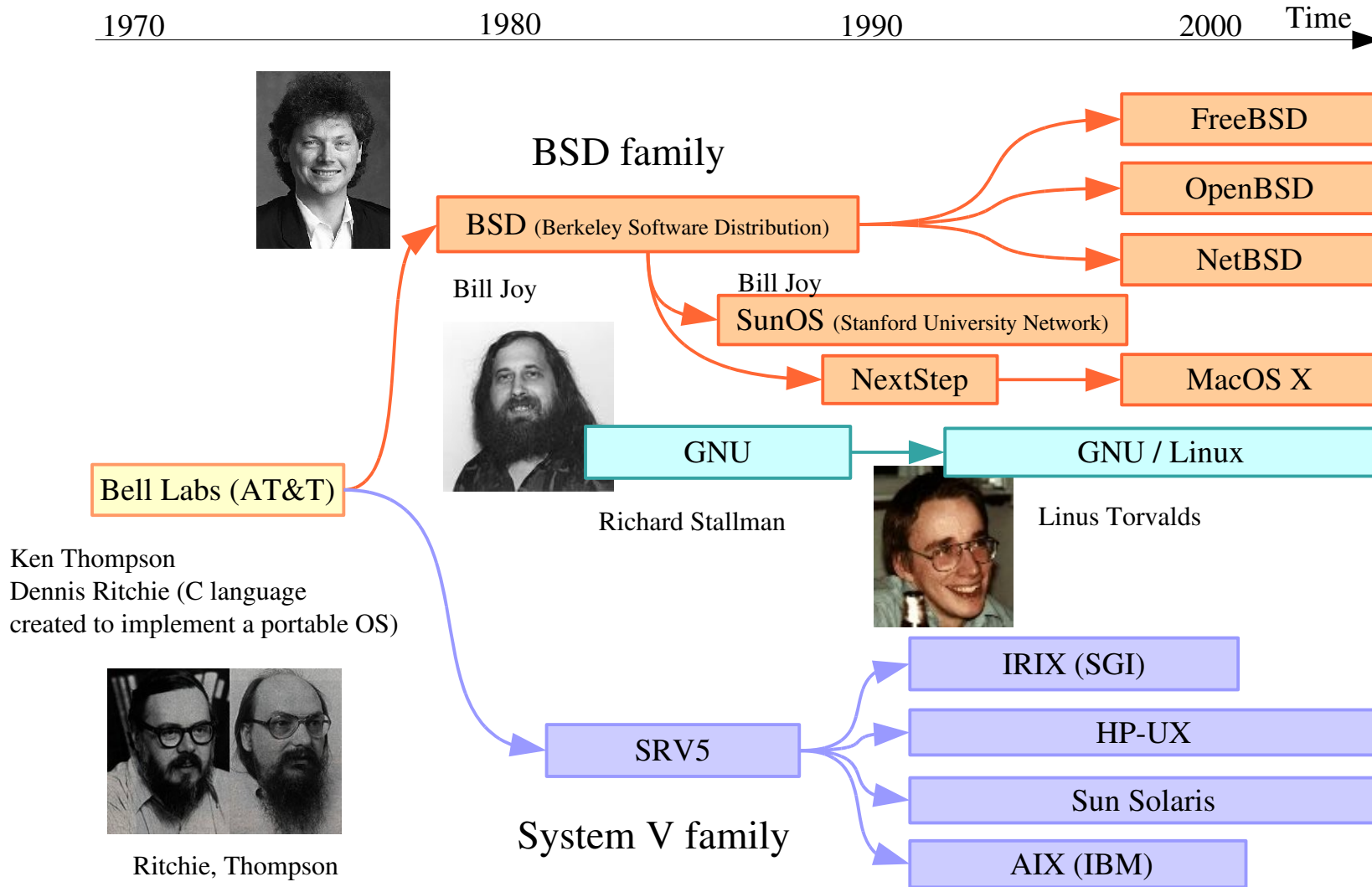


1969
 1971 to 1973
 1974 to 1975
 1978
 1979
 1980
 1981
 1982
 1983
 1984
 1985
 1986
 1987
 1988
 1989
 1990
 1991
 1992
 1993
 1994
 1995
 1996
 1997
 1998
 1999
 2000
 2001 to 2004
 2005
 2006 to 2008

Legend:

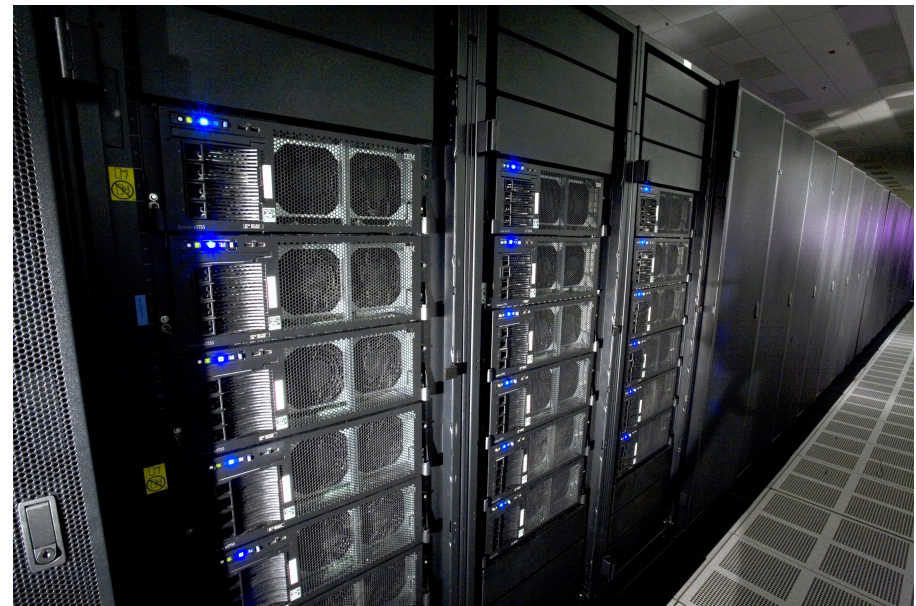
- Open Source (Green box)
- Mixed/Shared (Orange box)
- Closed Source (Red box)



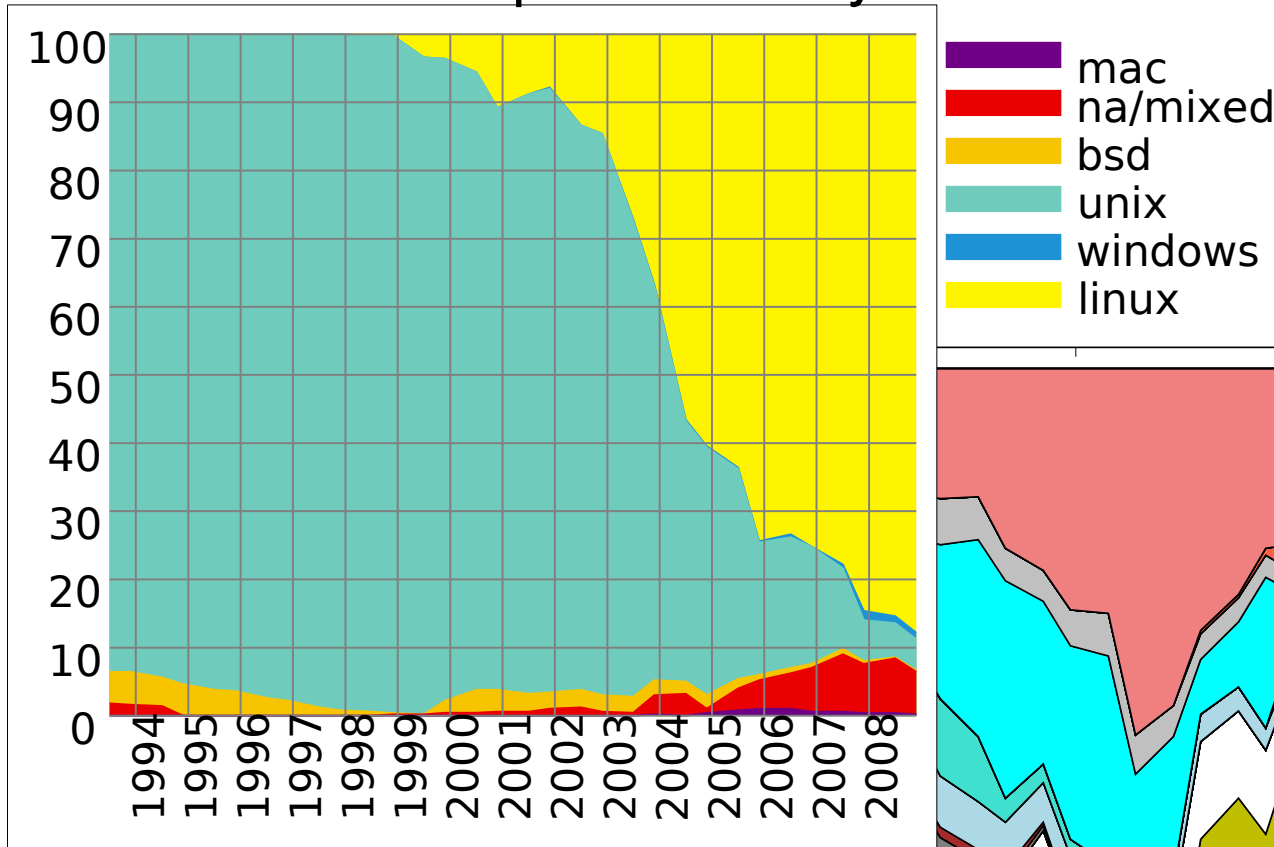


- ▶ 5.1. 1991 Linus Torvalds kupuje IBM PC
- ▶ V srpnu 1991 Linus informace o záměru napsat jádro
- ▶ 1991 – verze 0.01 publikována na internetu
- ▶ 1994 – verze 1.0 – jeden i386 CPU
- ▶ 1996 – verze 2.0 – SMP pro aplikace, BKL pro jádro
- ▶ 1999 – verze 2.2 – spinlock, m68k a PowerPC
- ▶ 2001 – verze 2.4 – ISA PnP, USB, PC Cards, PA-RISC, později Bluetooth, LVM, RAID, ext3
- ▶ 2003 – verze 2.6 – mainline μ Clinux, ARM a další, PAE, ALSA, preemption, Native POSIX Thread Library, Futex, později FUSE, JFS, XFS, ext4, robust mutex, prio inherit mutex, high resolution timers
- ▶ Okolo 2007 – CONFIG_PREEMPT_RT, spinlock \rightarrow RT-mutex, rušení BKL, IRQ \rightarrow thready, preemptible RCU

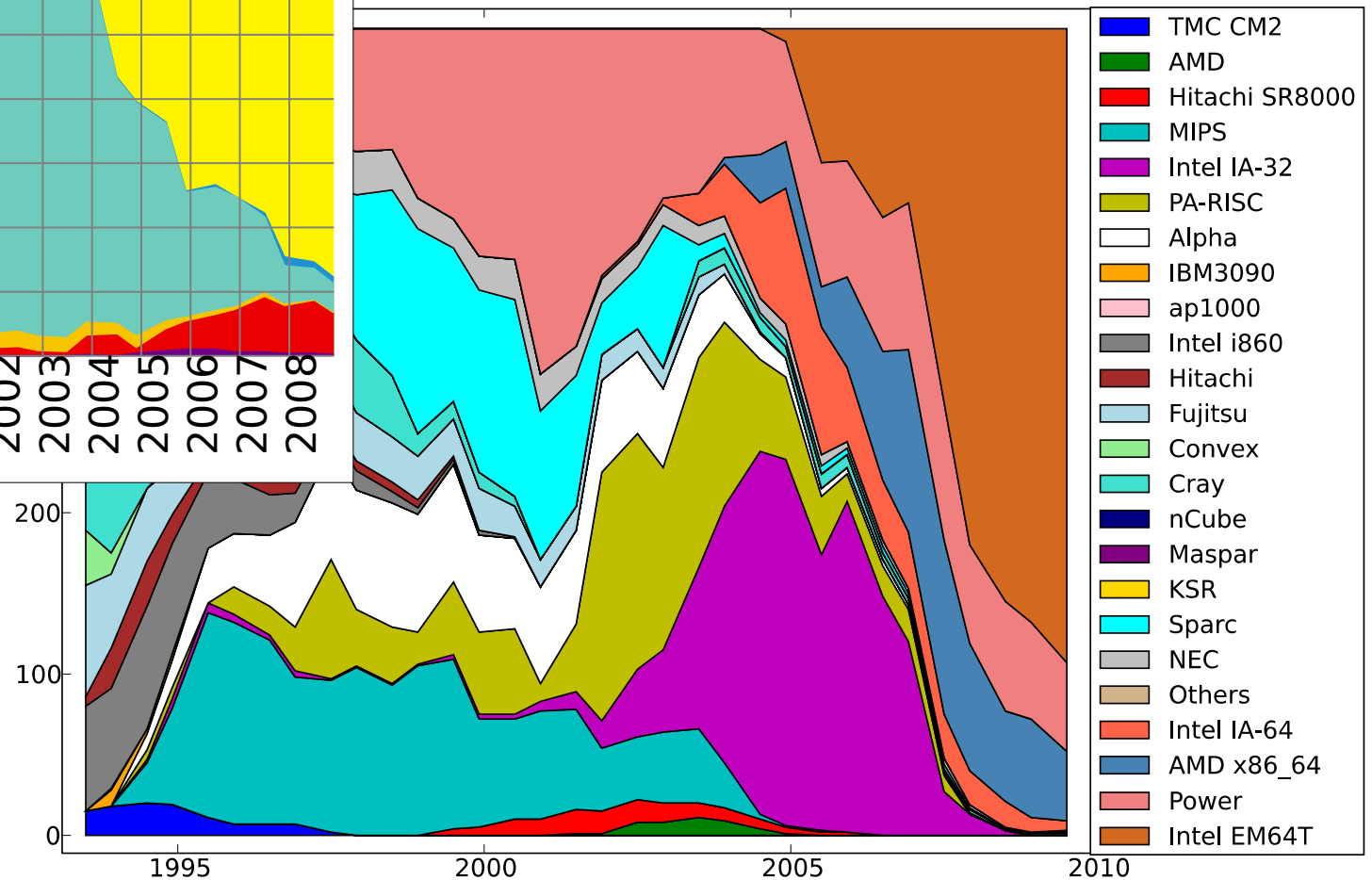
- ▶ IBM Roadrunner, Los Alamos National Laboratory
- ▶ Majitel: National Nuclear Security Administration, USA
- ▶ LINPACK/BLAS (Basic Linear Algebra Subprograms) systém
- ▶ Architektura: 12,960 IBM PowerXCell 8i CPUs,
6,480 AMD Opteron dual-core processors, Infiniband, Linux
- ▶ Systém: Red Hat Enterprise Linux a Fedora
- ▶ Napájení: 2.35 MW
- ▶ Velikost: 296 stojanů
560 m²
- ▶ Paměť: 103.6 TiB
- ▶ Výkon 1.042 petaflops
- ▶ Cena: USD \$125M
- ▶ SGI SSI (single system image) Linux, 2048 Itanium CPU a 4TiB RAM



Podle operačního systému



Podle CPU architektury



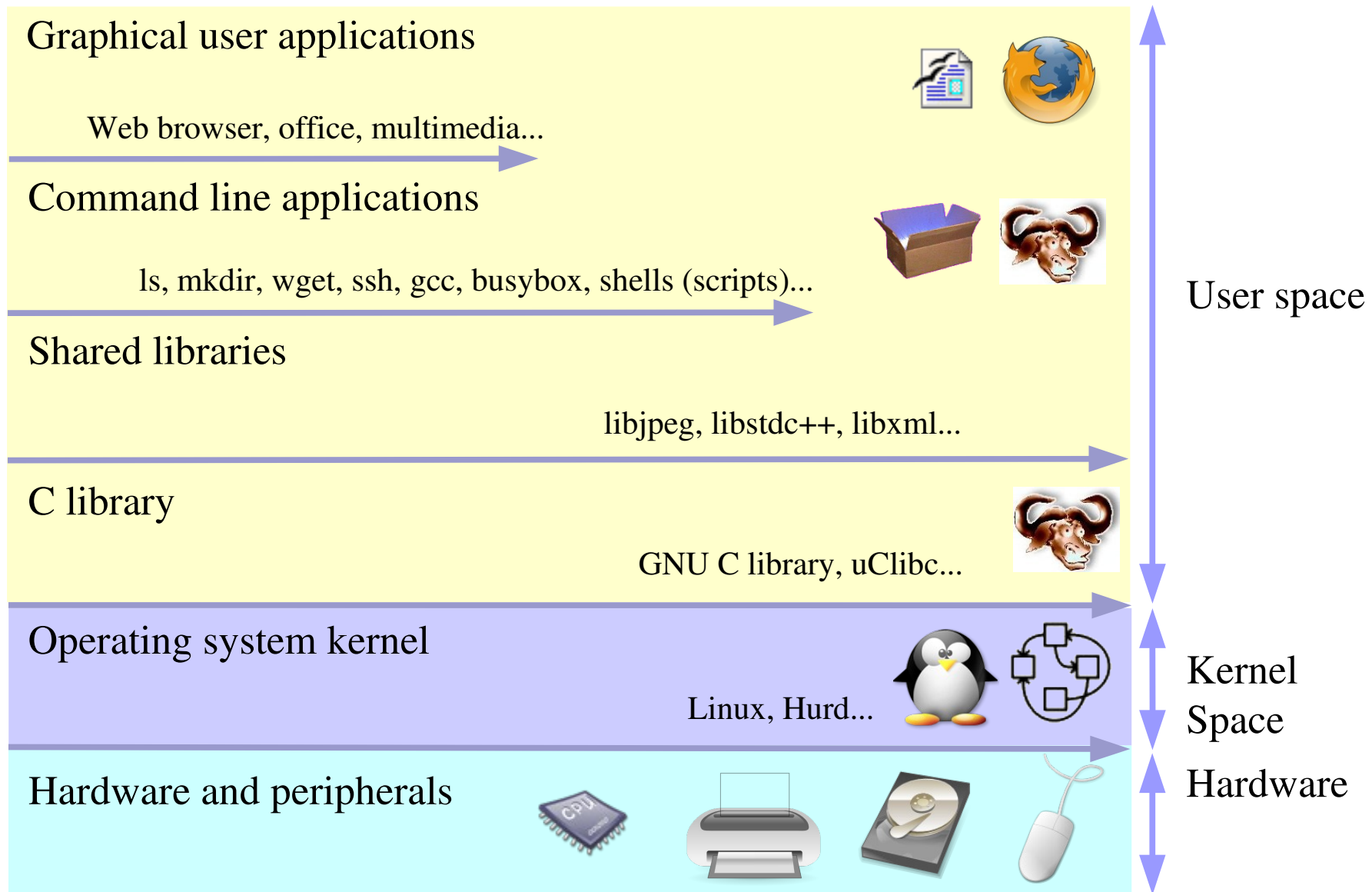
<http://www.top500.org/>

- ▶ Operating system used in the previous project
 - ▶ Proprietary OS: 39%
 - ▶ Free of cost embedded Linux: 29%
 - ▶ Embedded Linux with commercial support: 11%
 - ▶ Home grown OS: 7%
 - ▶ No OS: 11%

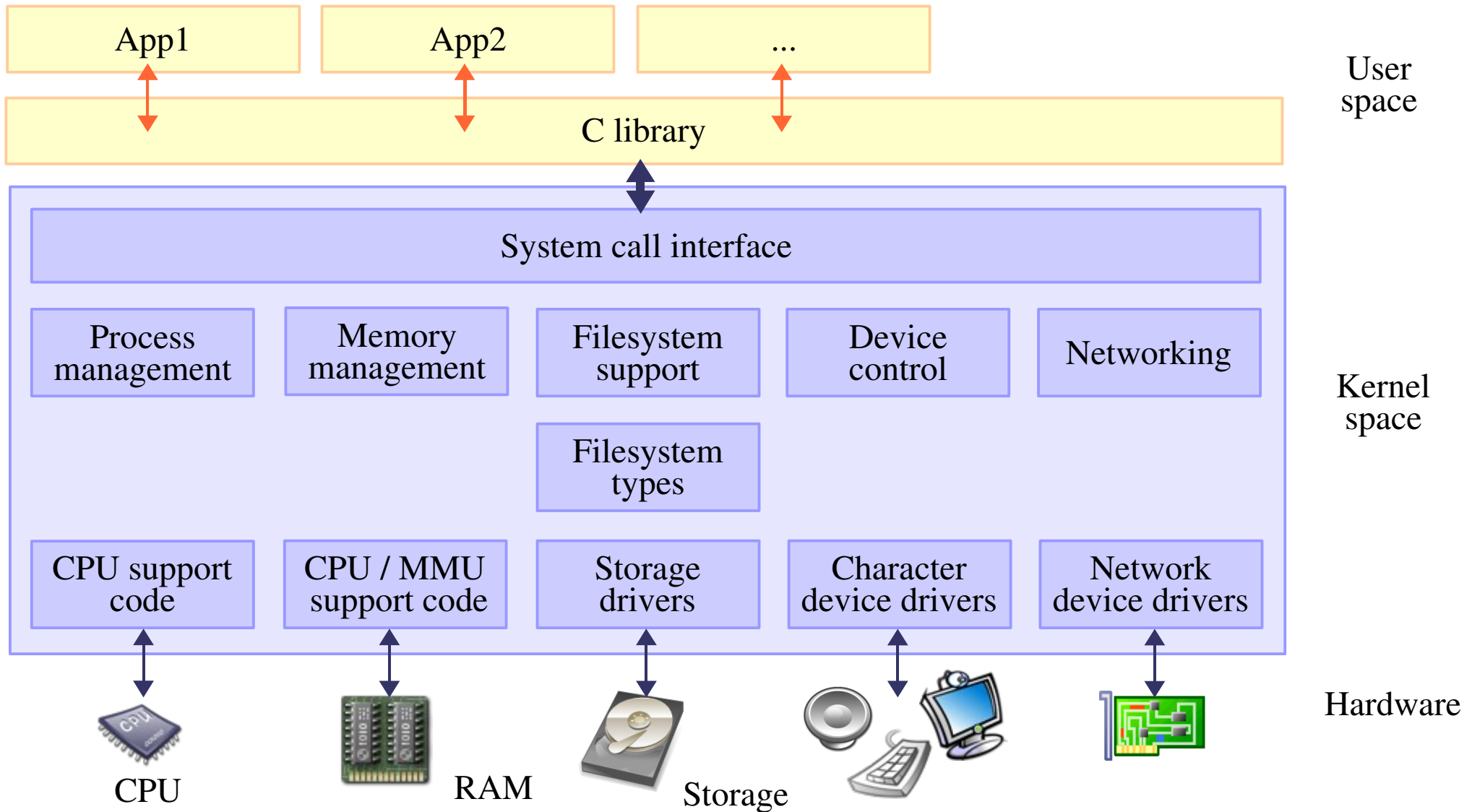
- ▶ Operating system planned for the next project
 - ▶ Free of cost embedded Linux: 71%
 - ▶ Embedded Linux with commercial support: 16%
 - ▶ Proprietary OS: 12%
 - ▶ Home grown OS: 1%

- ▶ Source: Venture Development Corp, October 2007

http://www.vdc-corp.com/_documents/pressrelease/press-attachment-1394.pdf



Převzato z <http://free-electrons.com/docs/freesw> (Michael Opdenacker)



- ▶ Mějme jednoduchý požadavek alokovat a uvolňovat unikátní čísla (např. PID) v určitém rozsahu (alloc_pid)

```
int pid_used[MAX_PID];
int alloc_pid(void)
{ static int pid = 1;
  for(i = 1; i<MAX_PID; i++) {
    if (!pid_used[pid]) {
      pid_used[i] = 1;
      return i;
    } else {
      pid = pid<MAX_PID-1?pid+1:1;
    }
  }
  Return 0;
}
```

Na jednom CPU to jde, ale co cache na 2048 SSI (MSI, MOSI, MOESI si vzalo veškerý výkon)

Kód asi není dobře
Chybí SMP, přidáme zámky
Ó to je pomalé
Zkusíme atomické operace
a bitové pole

```
while (1) {
  if (!test_and_set_bit(...)) {
    atomic_dec(&nr_free);
    return pid;
  }
}
```

- ▶ Ve skutečnosti je to mnohem složitější
- ▶ Protože jádro podporuje i jmenné prostory (lehká virtualizace), potřebujeme celý řetěz PIDů pro každý name space
- ▶ Nemůžeme použít statické pole, max user processes je jen administrátorem nastavitelná konstanta (ulimit -u / -a)
Max 4 milióny PID na 64-bit arch., horní bity pro robust futex na 32-bit max 32 tisíc, CONFIG_BASE_SMALL 8 tisíc
- ▶ Vše tedy musí být dynamické, nelze zabrat tolik paměti zbytečně, zároveň použití PID jako jednoznačného ID je v jádře díky NS nemožné
- ▶ Řešení: `struct pid` – zároveň slouží jako reference na a pro `struct task_struct` , uživatel vidí `pid_t`
- ▶ `PIDMAP_ENTRIES ((PID_MAX_LIMIT+8*PAGE_SIZE-1)/PAGE_SIZE/8)`
- ▶ Ukázka `make cscope, kscope a make htmldocs`

- ▶ Futex – fast userspace mutex
- ▶ Pro shared převod na vma+index+offs → mezi procesy (get_futex_key)
- ▶ Základ pro pthread_mutex_init, _lock, _trylock, _unlock, _destroy
- ▶ Ale i sem_post, sem_wait a veškeré pthread_cond

- ▶ Futex_ + WAIT
- + WAKE
- + FD
- + REQUEUE
- + CMP_REQUEUE
- + WAKE_OP
- + LOCK_PI
- + UNLOCK_PI
- + TRYLOCK_PI
- + WAIT_BITSET
- + WAKE_BITSET
- + WAIT_REQUEUE_PI
- + CMP_REQUEUE_PI

```
static inline int
usema_down(uunlock_t *unlock)
{
    if (!__unlock_down(unlock))
        return 0;
    return sys_unlock_wait(unlock);
}

static inline int
usema_up(uunlock_t *unlock)
{
    if (!__unlock_up(unlock))
        return 0;
    return sys_unlock_wakeup(unlock);
}
```




- ▶ Robust FUTEX od 2.6.17 (release 2006)
- ▶ Priority inheritance pro uživatelské procesy od 2.6.18 (2006)
pthread_mutexattr_setprotocol (... , PTHREAD_PRIO_INHERIT)
- ▶ `glibc/nptl/sysdeps/unix/sysv/linux/i386/lowlevellock.h`
- ▶ Ulrich Drepper: Futexes Are Tricky, Red Hat, Inc. 2011
<http://people.redhat.com/drepper/futex.pdf>
- ▶ Lightweight PI-futexes
<http://lxr.linux.no/source/Documentation/pi-futex.txt>
- ▶ Ingo Molnar a další: Robust Futex
<http://lxr.linux.no/source/Documentation/robust-futexes.txt>
- ▶ MSDN Windows 8, Windows Server 2012
WaitOnAddress, WakeByAddressSingle, WakeByAddressAll
Windows 8 / 2012 Server (žádné PI, robustness, requeue,
chování na SHM nespecifikované)

```
class mutex
{
public:
    mutex () : val (0) { }
    void lock () {
        int c;
        while ((c = atomic_inc (val)) != 0)
            futex_wait (&val, c + 1); }
    void unlock () {
        val = 0; futex_wake (&val, 1); }
private:
    int val;
};
```

Špatně:

- ▶ livelocks caused by the unconditional change of the futex variable must be avoided
- ▶ the futex value must not overflow
- ▶ in case it is known no threads wait on the mutex the futex wake call should be avoided.

```

class mutex2
{
public:
    mutex () : val (0) { }
    void lock () {
        int c;
        if ((c = cmpxchg (val, 0, 1)) != 0)
            do {
                if (c == 2
                    || cmpxchg (val, 1, 2) != 0)
                    futex_wait (&val, 2);
            } while ((c = cmpxchg (val, 0, 2))
                != 0);
    }
    void unlock () {
        if (atomic_dec (val) != 1) {
            val = 0;
            futex_wake (&val, 1);
        }
    }
private:
    int val;
};
    
```

		mutex	mutex2
lock	atomic op	1	1
	futex syscall	0	0
unlock	atomic op	0	1
	futex syscall	1	0

Cena bez kolize

Cena při kolizi

		mutex	mutex2
lock	atomic op	1 + 1	$\frac{2}{3} + \frac{1}{2}$
	futex syscall	1 + 1	1 + 1
unlock	atomic op	0	1
	futex syscall	1	1

```

class mutex3
{
public:
    mutex () : val (0) { }
    void lock () {
        int c;
        if ((c = cmpxchg (val, 0, 1)) != 0) {
            if (c != 2)
                c = xchg (val, 2);
            while (c != 0) {
                futex_wait (&val, 2);
                c = xchg (val, 2);
            }
        }
    }
    void unlock () {
        if (atomic_dec (val) != 1) {
            val = 0;
            futex_wake (&val, 1);
        }
    }
private:
    int val;
};

```

Cena při kolizi

		mutex2	mutex3
lock	atomic op	$\frac{2}{3} + \frac{1}{2}$	$\frac{1}{2} + 1$
	futex syscall	$1 + 1$	$1 + 1$
unlock	atomic op	1	1
	futex syscall	1	1

- ▶ Ulrich Drepper: Futexes are Tricky

<http://people.redhat.com/drepper/futex.pdf>

- ▶ Embedded Linux driver development

<http://free-electrons.com/docs/kernel>