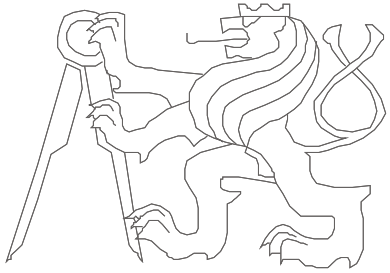
Architektura počítačů

Technické a organizační prostředky
(vnější události, výjimky, reálný čas)



České vysoké učení technické, Fakulta elektrotechnická

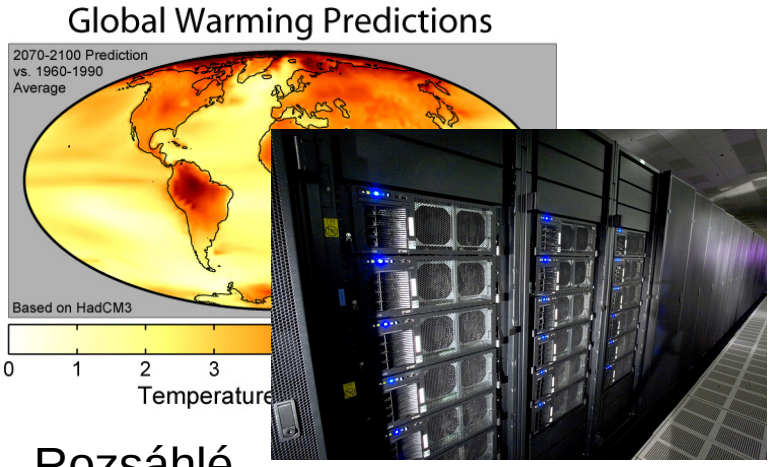
Stavební prvky (zopakování)

- Centrální procesorová jednotka (CPU)
- Paměti pro data a program v hierarchii
  - Registry (zápisníková paměť), cache (vyrovnávací), hlavní, vnější (disk)
- Sběrnice
  - ISA, PCI, PCIexpress

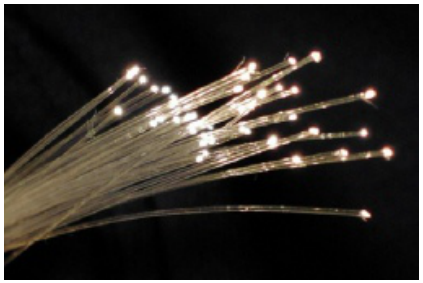
K čemu jsou tyto stavební díly použitelné

Zábava, hry, video

Firemní aplikace, účetnictví, bankovní operace, sklady a online obchody



Rozsáhlé matematické výpočty a modelování (globální klima, jaderná fúze, atd)

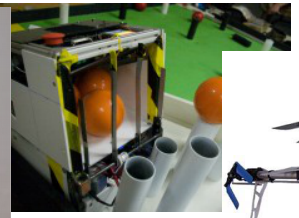


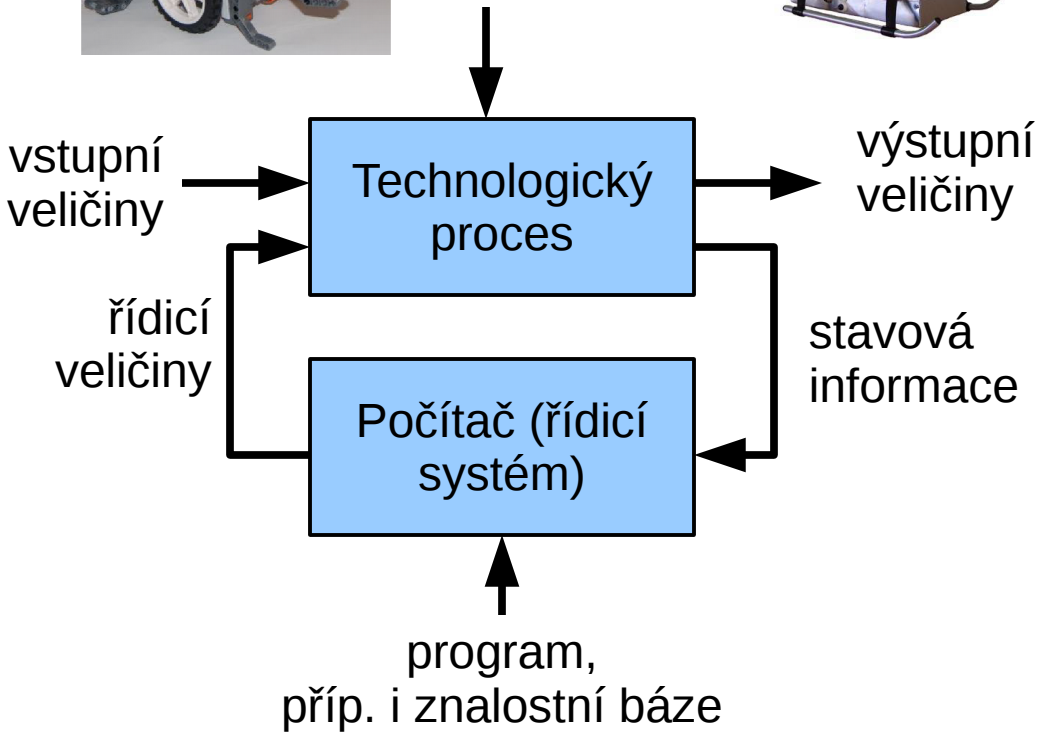
Komunikace, ať již jako samostatné aplikace nebo pro všechny další obory

A mnoho dalších oblastí ...

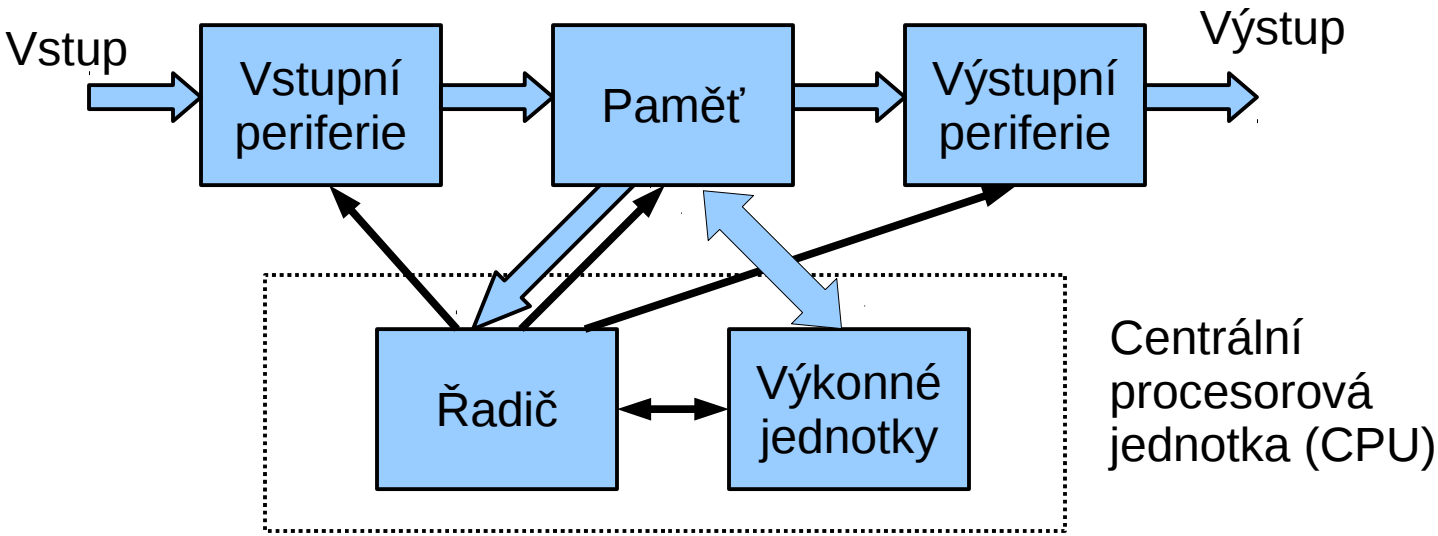
Počítač jako prostředek řízení

1. složitý proces (rychlost výp.)
2. levnost výpočetní techniky
3. flexibilita nasazení (program)
4. hierarchická stavba
5. přesnost výpočtů (zobrazení)
6. složité algoritmy (ext. vel.)





Model průchodu dat počítačem



Různé způsoby řízení zpracování dat

- Dávkové zpracování (úloha si řídí přístup dat tak, jak je zpracovává)
- Interaktivní (řízené událostmi ať od uživatele nebo podle příchodu vnějších požadavků nebo událostí)
- Řízení v reálném čase, kdy po příliš pomalém vyhodnocení/dosažení i správných výsledků nejsou výsledky použitelné

Vstupně výstupní (I/O) podsystém

- Pouze vstupní periferie
  - Běžné: klávesnice, myš, kamera
  - Logické vstupy, převodníky fyzikálních veličin většinou nejdřív na analogový elektrický signál a pak A/D převodníkem na číslo na vstupní bráně a další senzory
- Pouze výstupní periferie
  - Výstup obrazu (2D, 3D + akcelerace), výstup zvuku
  - Výstup s hmatatelnou fyzikální podobou, 3D tisk (rapid prototyping), řízení technologických veličin přes převodníky (D/A, PWM) a další aktuátory
- Obousměrné
  - Disk, komunikační rozhraní
  - I většina výše uvedených „jednosměrných“ periferií ve skutečnosti vyžaduje obousměrnou komunikaci

Metody přenosu dat z/na periferie

- Programový kanál (polling)
  - Procesor ve smyčce čeká na informaci o dostupnosti dat / volné místo ve výstupním registru
- Programový kanál s přerušením (interrupt driven)
  - Na vstupu inicializační kód programu/OS pouze konfiguruje periferii. Při příchodu dat dojde k vyvolání asynchronní události (přerušení). V její obsluze je slovo/blok dat z periferie vyčten
  - Procesor zapíše slovo nebo blok do výstupní periferie a provedení přenosu je potvrzeno přerušením
- Přímý přístup do paměti (direct memory access – DMA)
  - Vlastní přenos dat realizuje specializovaná jednotka
- Autonomní kanál (inteligentní periferie, bus master DMA)

Programový kanál (polling)

```
DoSomethingWithData:
    Wait4Device:
        in( dx, al );
        test( 1, al );
        jnz Wait4Device;
     << Do something with the Data>>
     jmp DoSomethingWithData;
```

Příklady: Randall Hyde (randyhyde_at_earthlink.net) e-mail z 14. června 2004

- Nejméně výhodné řešení, procesor čeká na data ve smyčce (busy wait)
- I pokud opravdu není možné nalézt pro CPU jinou užitečnou činnost (viz dále sdílení času – time sharing, multi processing, threading, user), tak je toto řešení minimálně neekologické

Programový s přerušením (interrupt driven)

```
InterruptServiceRoutine:
      << Get data and move to a shared memory location >>
      mov( 1, DataAvailable );
      iret();

MainThreadLoop:
    << Tell I/O device we want data >>
    Wait4Data:
         OptionalHALT or OtherDataProcessing;
         test( 1, DataAvailable );
         jnz Wait4Data;
     <<Do Something With Data >>
      jmp MainThreadLoop;
```

- Periferie se sama stará o oznámení přítomnosti nových dat – vyvolá výjimku/přerušení
- Výše uvedený příklad celkově situaci nezlepšuje, po přidání plánování úloh je však možné současnou úlohu pozastavit, přeplánovat na jinou úlohu a úlohu opět aktivovat po příchodu dat

Linux kernel: Čekání na událost s přeplánováním

```
static DECLARE_WAIT_QUEUE_HEAD(foo_wq);
volatile int event_pending;

irqreturn_t foo_irq_fnc(int intno, void *dev_id)
{
  <<read device status, store what can be lost and stop/mask IRQ>>
  event_pending = <<indicate even arrival>>;
  wake_up_interruptible(&foo_wq);
  return IRQ_HANDLED;
}

static ssize_t foo_read(struct file *fp, char __user *buf,
                        size_t len, loff_t *off)
{
  wait_event_interruptible_timeout(foo_wq, event_pending != 0);
  << check error state etc. signal_pending(current) >>
  << process event_pending and event_pending = 0 >>
  err = copy_to_user(buf, internal_buffer, len);
  return len;
}
```

RTEMS: Čekání na událost s přeplánováním

```
rtems_isr mmcsd_irq_handler(rtems_irq_hdl_param data)
{
  MMCSD_Dev *device=(MMCSD_Dev *)data;
  rtems_event_send(device->waiter_task_id, MMCSD_WAIT_EVENT);
}

static int mmcsd_read(MMCSD_Dev *device, rtems_blkdev_request *req)
{
  rtems_status_code status;
  rtems_event_set   events;
  rtems_interval    ticks;
  rtems_id          self_tid;

  rtems_task_ident(RTEMS_SELF, 0, &self_tid);
  device->waiter_task_id = self_tid;
  status=rtems_event_receive(MMCSD_WAIT_EVENT | MMCSD_EVENT_ERROR,
                             RTEMS_EVENT_ANY|RTEMS_WAIT, ticks, &events);
  << process event fill sg = req->bufs - List of scatter/gather buffers >>
  req->req_done(req->done_arg, RTEMS_SUCCESSFUL, 0);
  return 0;
}
```

- Příklad je velmi zjednodušený. Dočasná registrace TID vlákna do struktury driveru se nepoužívá. O obsluhu zařízení se stará **worker thread** spuštěný při inicializaci driveru.

RTEMS: Využití semaforu pro čekání na interrupt

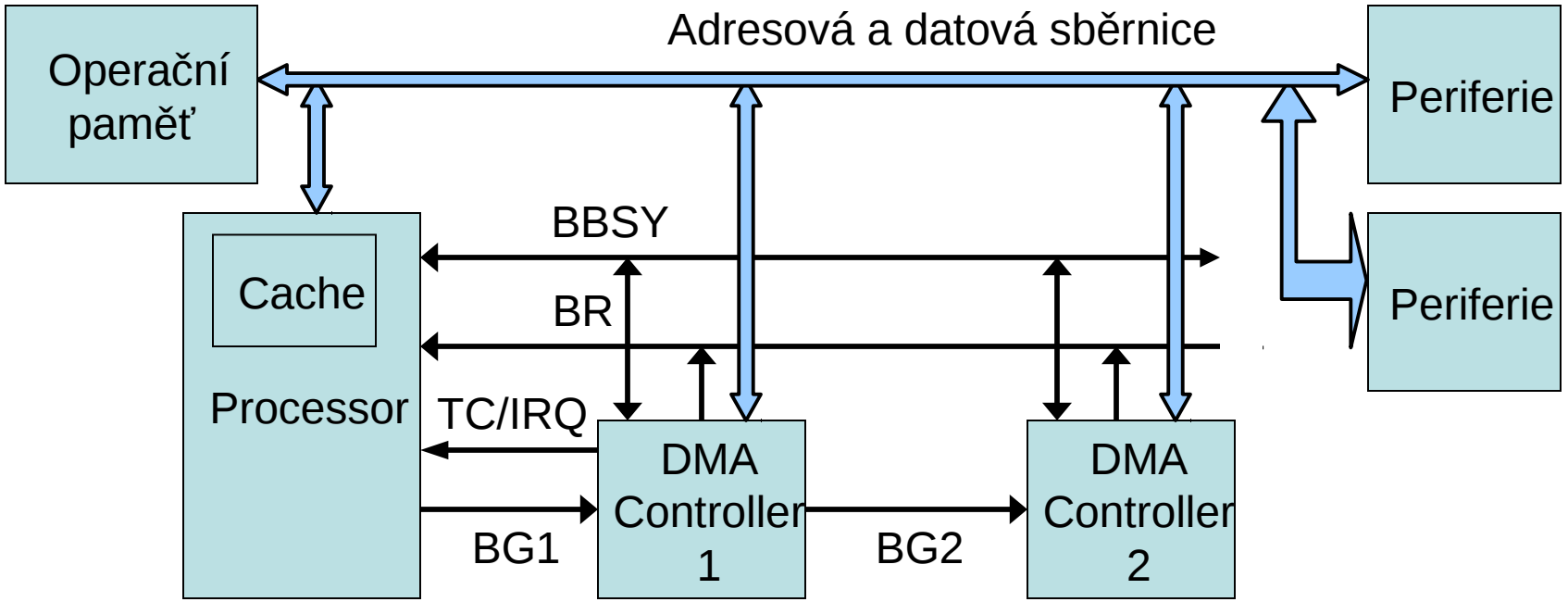
```
static rtems_id my_semaphore;

rtems_isr my_irq_handler(rtems_irq_hdl_param valu)
{
        if (<<check if really from device>>) {
                rtems_semaphore_release(my_semaphore);
        }
}

wait for event
    rtems_semaphore_obtain(semaphore, RTEMS_WAIT, RTEMS_NO_TIMEOUT);

initialize semaphore in the driver init
    rtems_semaphore_create(rtems_build_name('s','e','m','a'),
                    0/*initial value*/, RTEMS_FIFO, 5/*priority*/,
                    &my_semaphore/*location to store new sem ID*/);
```

- Podobně lze použít semafor ve VxWorks nebo jádře Linuxu. Jedná se ale o interní API a implementaci jader, POSIX/ANSI mechanizmy pro obsluhu přerušení nedefineje.

Windows: Interrupt and deferred procedure call

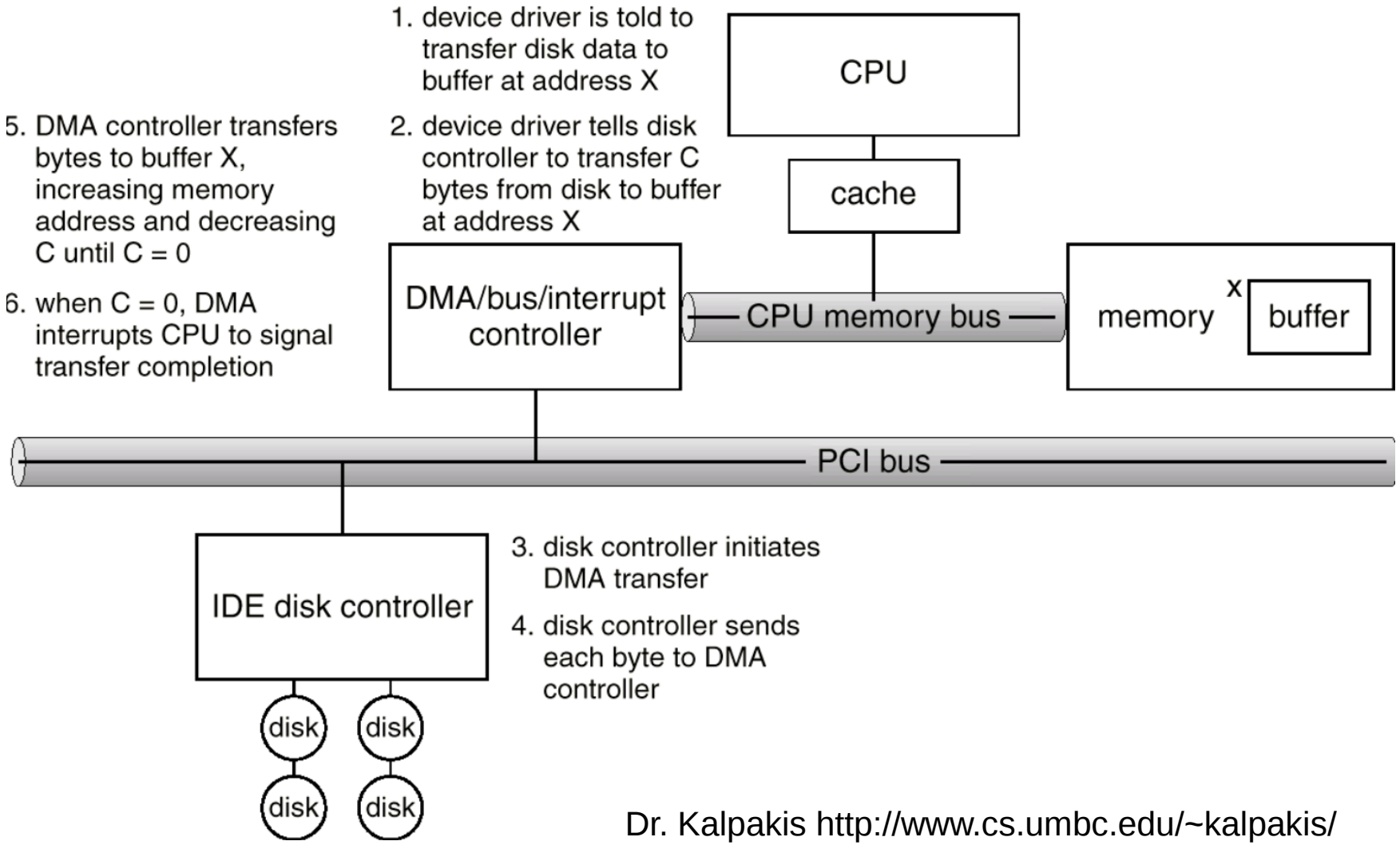
```
VOID NTAPI ulan_bottom_dpc(IN PKDPC Dpc,IN PVOID contex,
                      IN PVOID arg1,IN PVOID arg2);

KSERVICE_ROUTINE InterruptService;
BOOLEAN uld_irq_handler( _In_  struct _KINTERRUPT *Interrupt,
  _In_  PVOID ServiceContext)
{
    …
    KeInsertQueueDpc(&(udrv)->bottom_dpc,NULL,NULL);
    return TRUE;
}

status =
IoConnectInterrupt(&udrv->InterruptObject,
    uld_irq_handler,             // ServiceRoutine
    udrv,                        // ServiceContext
    NULL,                        // SpinLock
    udrv->irq,                   // Vector
    udrv->Irql,                  // Irql
    udrv->Irql,                  // SynchronizeIrql
    udrv->InterruptMode,         // InterruptMode
    TRUE /*FALSE for ISA? */,    // ShareVector
    udrv->InterruptAffinity,     // ProcessorEnableMask
    FALSE);                      // FloatingSave
```

Přímý přístup do paměti (Direct Memory Transfer - DMA)



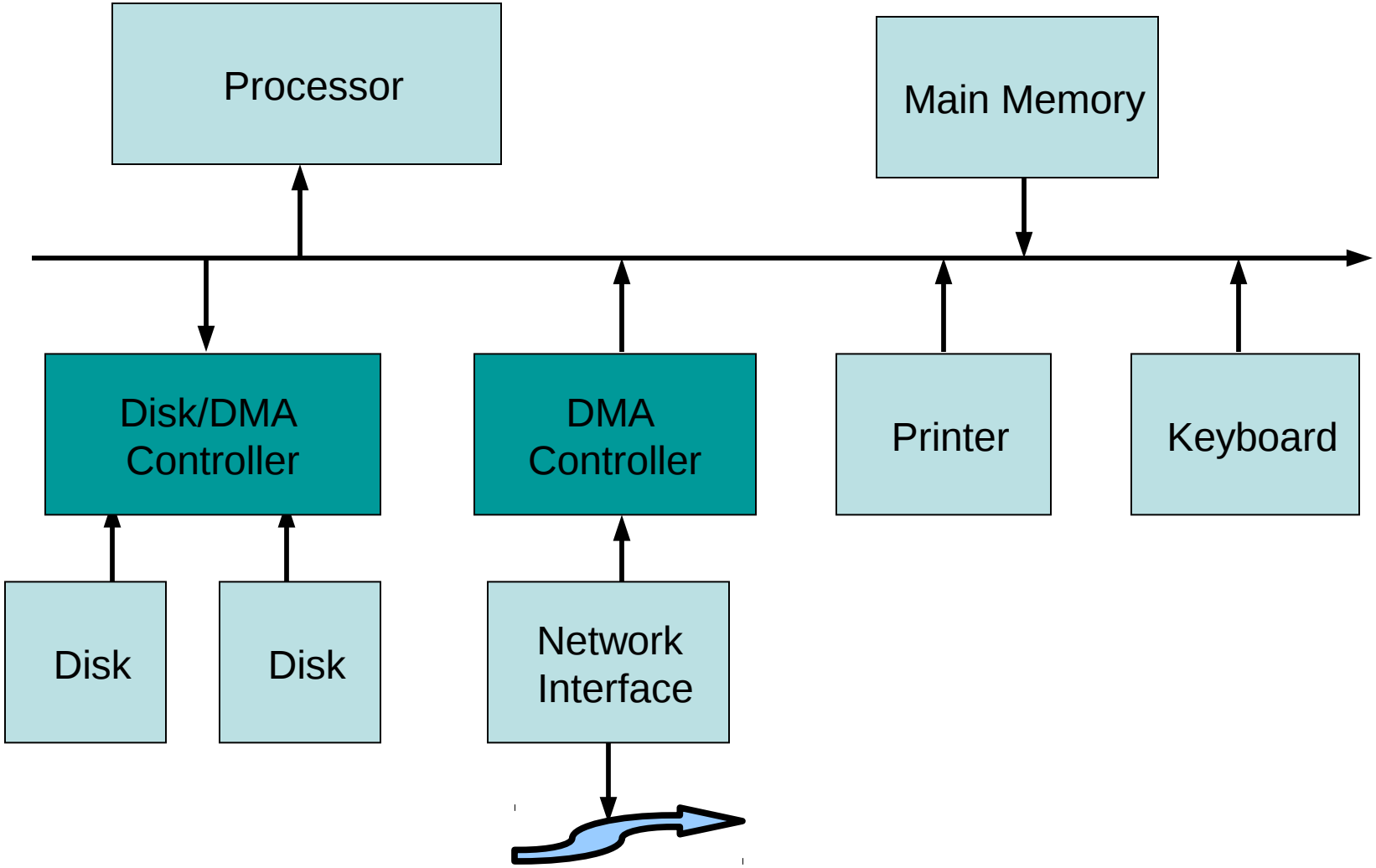
- Počítačový systém je doplněn o specializované jednotky pro určené pro přenos dat
- Velké datové přenosy zbytečně nevytěsňují data z cache
- Program/OS naplánuje parametry přenosu
- Procesor nastaví adresy do DMA řadiče, ten po ukončení operace vyvolá přerušení

Příklad DMA přenosu z disku



Dr. Kalpakis http://www.cs.umbc.edu/~kalpakis/

Decentralizace DMA řadiče – přesun k periferiím



Autonomní kanál (Bus Master DMA a IOP)

- Inteligence se přesouvá do periferie
- Periferie je doplněna vlastním řadičem
  - Konečný automat
  - Vstupně/výstupní procesor (IOP) atd.
- Průběh přenosu
  - Nadřazený/centrální procesor vloží sekvenci datových bloků do paměti
  - Nakonfiguruje nebo přímo naprogramuje řadič periferie a ta provede sekvenci přenosů z/do hlavní paměti
  - Po úplném nebo částečném (přijatý první celý paket) dokončení činnosti informuje CPU vyžádáním přerušení
- Procesor/operační systém zpracuje přerušení a přeplánuje na úlohu, která na data čeká

Kde je zakopaný pes / aneb úskalí DMA a I/O

Paměť, mapované periferie a datová konzistence/koherence

- Vstupně výstupní operace a CPU
  - Pro oblasti adresního prostoru, kam jsou mapované periferie, musí být zakázané použití cache při ukládání a čtení dat
  - Zřetězené zpracování instrukcí (pipelining) samotné nevadí
  - Přeposílání (data forwarding), vynechávání zbytečných čtení a zápisů (bypassing) a případné vykonávání instrukcí mimo pořadí vadí
  - Nutnost zavést specializované instrukce, které zajistí dokončení (všech) přenosů před další operací
    - MIPS IV - **sync** (l*x* a sx se dokončí před dalším l*x*)
    - PowerPC
      - **eieio** (Enforce In-Order Execution of I/O) Instruction
      - **sync** ne jen pro I/O ale i pro paměť
  - Stejně tak je nutné informovat optimalizaci v kompilátoru (volatile, ...)

Paul E. McKenney: Memory Ordering in Modern Microprocessors

Wikipedia: http://en.wikipedia.org/wiki/Memory_ordering

Atomické operace, kompilátory a STL

- C++ std::atomic_int, std::atomic_intptr_t, …

```
typedef enum memory_order
{
    memory_order_relaxed, memory_order_consume,
    memory_order_acquire, memory_order_release,
    memory_order_acq_rel, memory_order_seq_cst
} memory_order;
```

- C1x

Implementace paměťového modelu podle C++11 v GCC

C++11 memory models

- __ATOMIC_RELAXED – No barriers or synchronization.
- __ATOMIC_CONSUME – Data dependency only for both barrier and synchronization with another thread.
- __ATOMIC_ACQUIRE – Barrier to hoisting of code and synchronizes with release (or stronger) semantic stores from another thread.
- __ATOMIC_RELEASE – Barrier to sinking of code and synchronizes with acquire (or stronger) semantic loads from another thread.
- __ATOMIC_ACQ_REL – Full barrier in both directions and synchronizes with acquire loads and release stores in another thread.
- __ATOMIC_SEQ_CST – Full barrier in both directions and synchronizes with acquire loads and release stores in all threads.

Atomické operace podle normy C++11

- type __atomic_load_n (type *ptr, int memmodel)
  - RELAXED, SEQ_CST, ACQUIRE and CONSUME
- void __atomic_load (type *ptr, type *ret, int memmodel)
- __atomic_store_n (type *ptr, type val, int memmodel)
  - RELAXED, SEQ_CST, RELEASE
- void __atomic_store (type *ptr, type *val, int memmodel)
- __atomic_exchange_n (type *ptr, type val, int memmodel)
  - RELAXED, SEQ_CST, ACQUIRE, RELEASE and ACQ_REL
- void __atomic_exchange (type *ptr, type *val, type *ret, int memmodel)

C++11 Compare and Swap

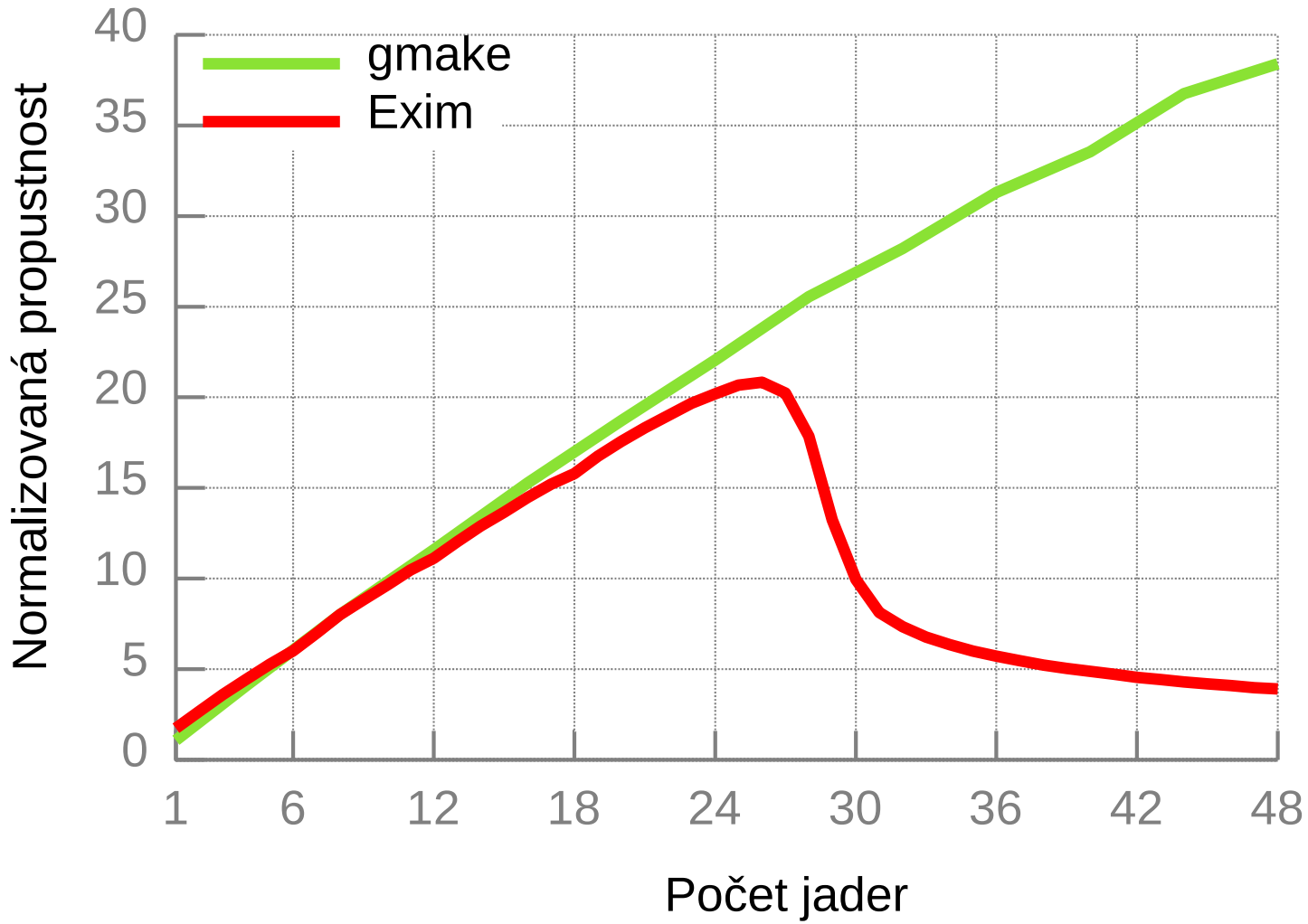
- bool __atomic_compare_exchange_n (type *ptr, type *expected, type desired, bool weak, int success_memmodel, int failure_memmodel)
- bool __atomic_compare_exchange (type *ptr, type *expected, type *desired, bool weak, int success_memmodel, int failure_memmodel)

C++11 aritmetické a logické operace

- type __atomic_add_fetch (type *ptr, type val, int memmodel)

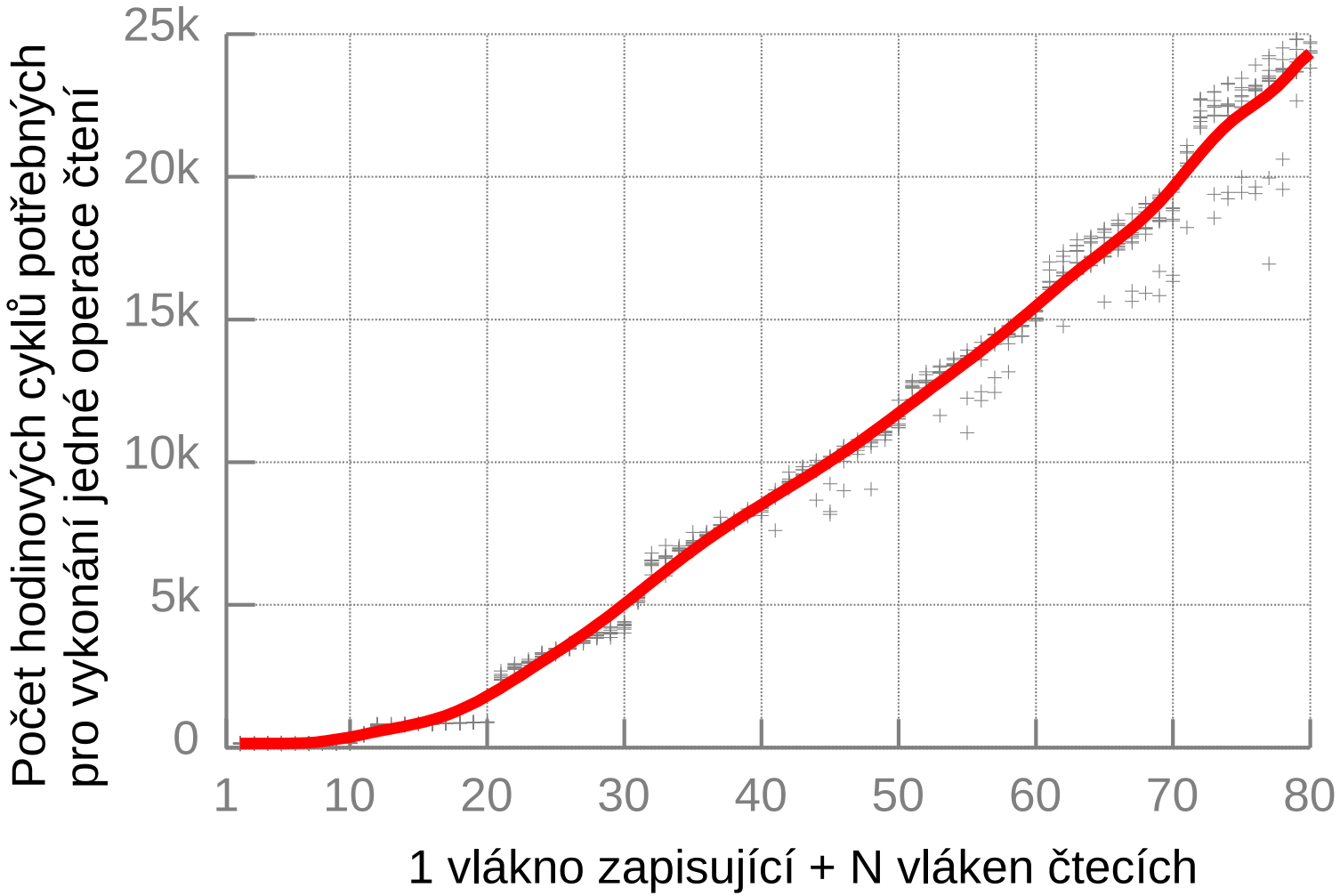
  add, sub, and, xor, or, nand
- type __atomic_fetch_add (type *ptr, type val, int memmodel)
- bool __atomic_test_and_set (void *ptr, int memmodel)
- void __atomic_clear (bool *ptr, int memmodel)
- void __atomic_thread_fence (int memmodel)
- void __atomic_signal_fence (int memmodel)
- bool __atomic_always_lock_free (size_t size, void *ptr)
- bool __atomic_is_lock_free (size_t size, void *ptr)

Úzké hrdlo škálovatelnosti při přístupu k paměti z více jader

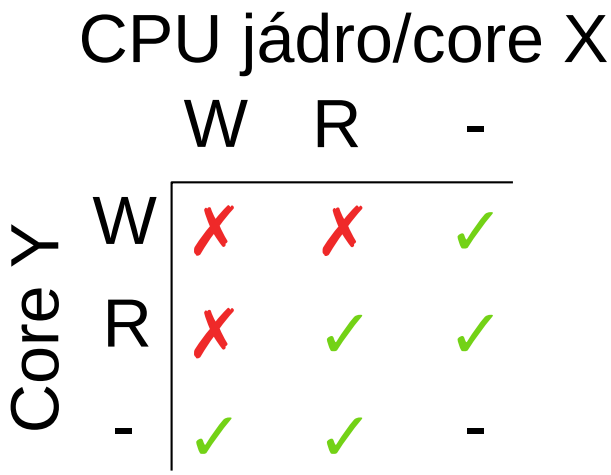


Jak jedna přepisovaná řádka cache může zruinovat propustnost aplikace

Cena kolize v jedné řádce paměti cache



Jaké přístupy a algoritmy jsou škálovatelné?



Source
The Scalable Commutativity Rule: Designing Scalable Software for Multicore Processors by Austin T. Clements

Konstrukce, které jsou škálovatelné pro více threadů

- Úroveň škálovatelnosti: používat škálovatelné datové struktury
  - Lineární pole a radix arrays
  - Hash tables
  - **Nepoužívat** binární/vyvažované stromy pro sdílená data
- Odložení činnosti/úklidu – defer work, reference tracking, read copy update RCU a odložené uvolňování/rušení
- Předcházet pesimistické operaci optimistickou kontrolou
  - Pouze pokud kontrola objektu určí, že je potřeba ho změnit tak provést zápis dat do struktury, souboru, atd.
- Na úrovni práce s operačním systémem použít vždy jen takovou operaci, která je nutná
  - Použít test access(F_OK) místo zjištění existence souboru kontrolou návratového kódu operace otevření a nebo čtení

DMA, cache a konzistence/koherence dat

- DMA přenos z hlavní paměti, ta je od CPU oddělena cache
- Při přenosu dat do periferie je nutné počkat, až se data dostanou do hlavní paměti (zpožděný zápis/writeback!)
- Před načtením dat z periferie je nutné vyprázdnit (část) cache (zneplatnit nebo zapsat zpět do paměti)
- CPU/jednotku správy paměti doplnit o instrukce/prostředky pro správu řádek cache
  - PowerPC
    - **dcbf** (Data Cache Block Flush), **clcs** (Cache Line Compute Size), **clf** (Cache Line Flush), **cli** (Cache Line Invalidate), **dcbi** (Data Cache Block Invalidate), **dcbst** (Data Cache Block Store), **dcbt** (Data Cache Block Touch), **dcbtst** (Data Cache Block Touch for Store), **dcbz**/**dclz** (Data Cache Block Set to Zero), **dclst** (Data Cache Line Store), **icbi** (Instruction Cache Block Invalidate), **sync** (Synchronize)/**dcs** (Data Cache Synchronize)
  - MIPS – specializovaná instrukce - **cache**

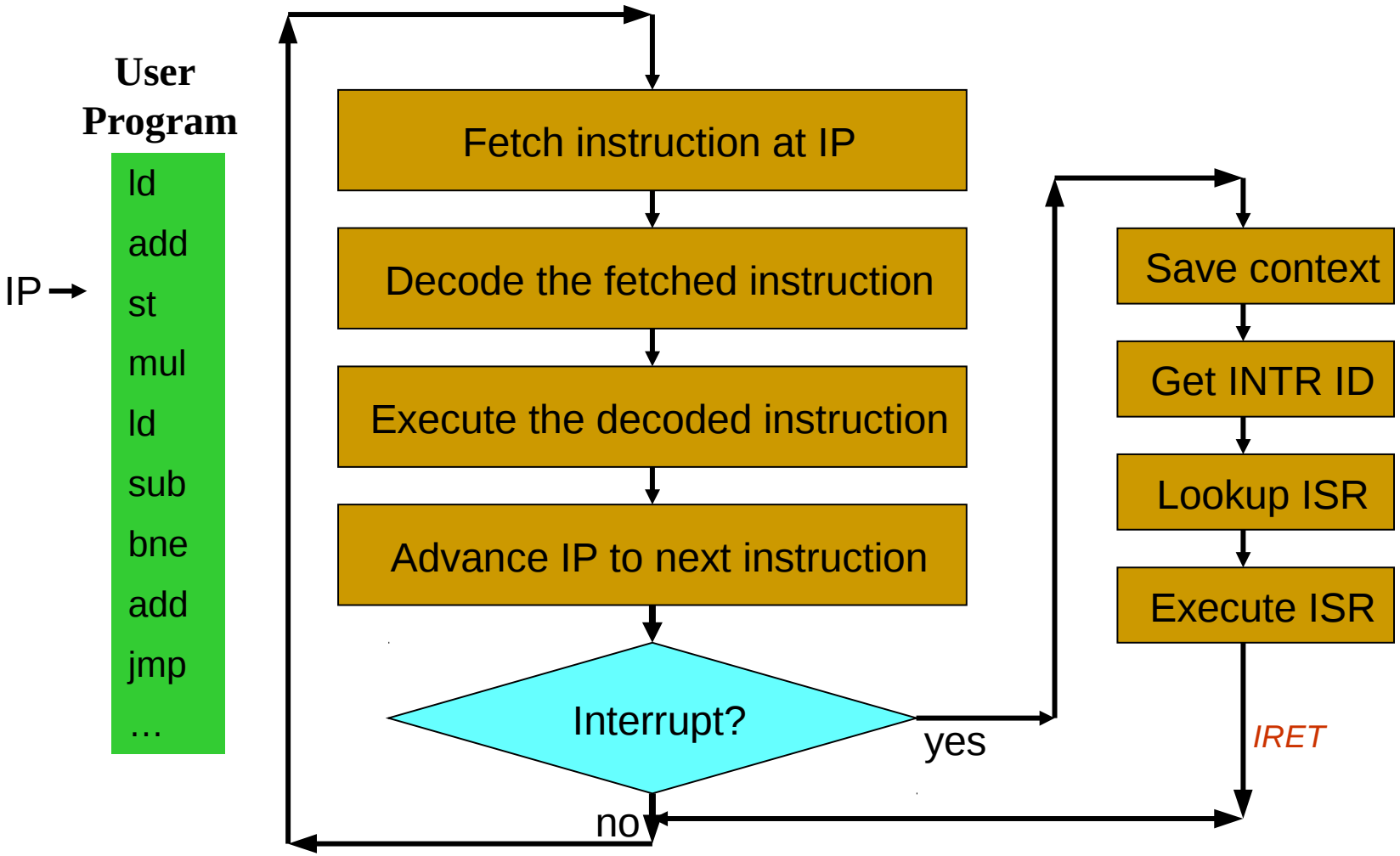
Výjimky a přerušení

- Výjimky – ošetření zvláštních situací, které brání dalšímu vykonávání instrukcí (exceptions)
  - Pro případ procesoru MIPS se jedná především o
    - Matematické přetečení (výsledek instrukce s kontrolou saturace přetekl)
    - Načtena nedefinovaná instrukce (neznámý operační kód instrukce typu IR, nebo neznámá funkce instrukce typu R)
    - Systémové volání (instrukce syscall)
  - Nedostupnost dat nebo selhání zápisu
    - Chybná adresa nebo stránka označená za nevalidní (invalid)
    - Chyba hlášená sběrnicí (parita, ECC, vypršení limitu na potvrzení)
- Asynchronní vnější události – přerušení (interrupts)
  - Maskovatelná, lze je zakázat v stavovém/řídicím slovu CPU, případně řízení priorit (periferie, čítače, časovače)
  - Nemaskovatelná – většinou ošetření HW chyb, hlídací obvod (Watch Dog)

Průběh zpracování výjimky nebo přerušení

- Výjimka typicky přijata vždy, přerušení jen pokud je povolené nebo nemaskovatelné
- Dojde k uložení stavového slova CPU včetně PC (buď na zásobník nebo do specializovaných registrů)
- Čítač instrukcí je nastavený na adresu odpovídající typu výjimky případně i číslu zdroje přerušení
- Z této adresy je vykonávaný kód obslužné rutiny
- Ta uvolní/uloží další registry na zásobník a obslouží periferii/provede nahrání stránky, vyhlásí neopravitelnou chybu úlohy nebo celého systému, atd.
- Obnoví registry
- Příslušnou instrukcí provede návrat CPU do předchozího stavu

Průběh zpracování přerušení graficky



MIPS - registry pro sledování a řízení výjimek

Register name	Register number	Usage
Status	12	Interrupt mask and enable bits
Cause	13	Exception type
EPC	14	Following address of the instruction where the exception occurred

Cause register

Number	Name	Description
00	INT	External Interrupt
01	IBUS	Instruction bus error (invalid instruction)
10	OVF	Arithmetic overflow
11	SYSCALL	System call

Status register - for disabling interrupts and exceptions

Bit	Interrupt/exception
3	INT
2	IBUS
1	OVF
0	SYSCALL

MIPS – průběh zpracování přerušení

Přijatý požadavek na přerušení, výjimka, **syscall** instrukce

```
EPC <= PC
Cause <= (cause code for event)
Status <= Status << 4
PC <= (handler address)
```

Začátek obslužné rutiny

- zjištění zdroje s využitím instrukce koprocesoru 0 **mfc0 rd, rt**
- stav lze modifikovat instrukcí **mtc0 rd, rt**
- **rd** je běžný registr, **rt** jeden ze speciálních registrů z koprocesoru 0

Návrat z přerušení instrukcí **rfe**

```
PC <= EPC
Status <= Status >> 4
```

Precizní obsluha výjimek

- Pokud je výjimka/přerušení obslouženo úspěšně, původní úloha pokračuje instrukcí, před kterou byla přerušena a kromě časového zpoždění nemůže přímo přerušení detekovat (nepřímo přes přerušovací rutinou nastavený stav, předaná data, modifikovaný tok instrukcí ovšem ano)
- Poznámka: Precizní obsluha paměťových výjimek je především komplikovaná zpožděným zápisem, který může selhat v době zpracování následujících instrukcí

Určení zdroje výjimky přerušení

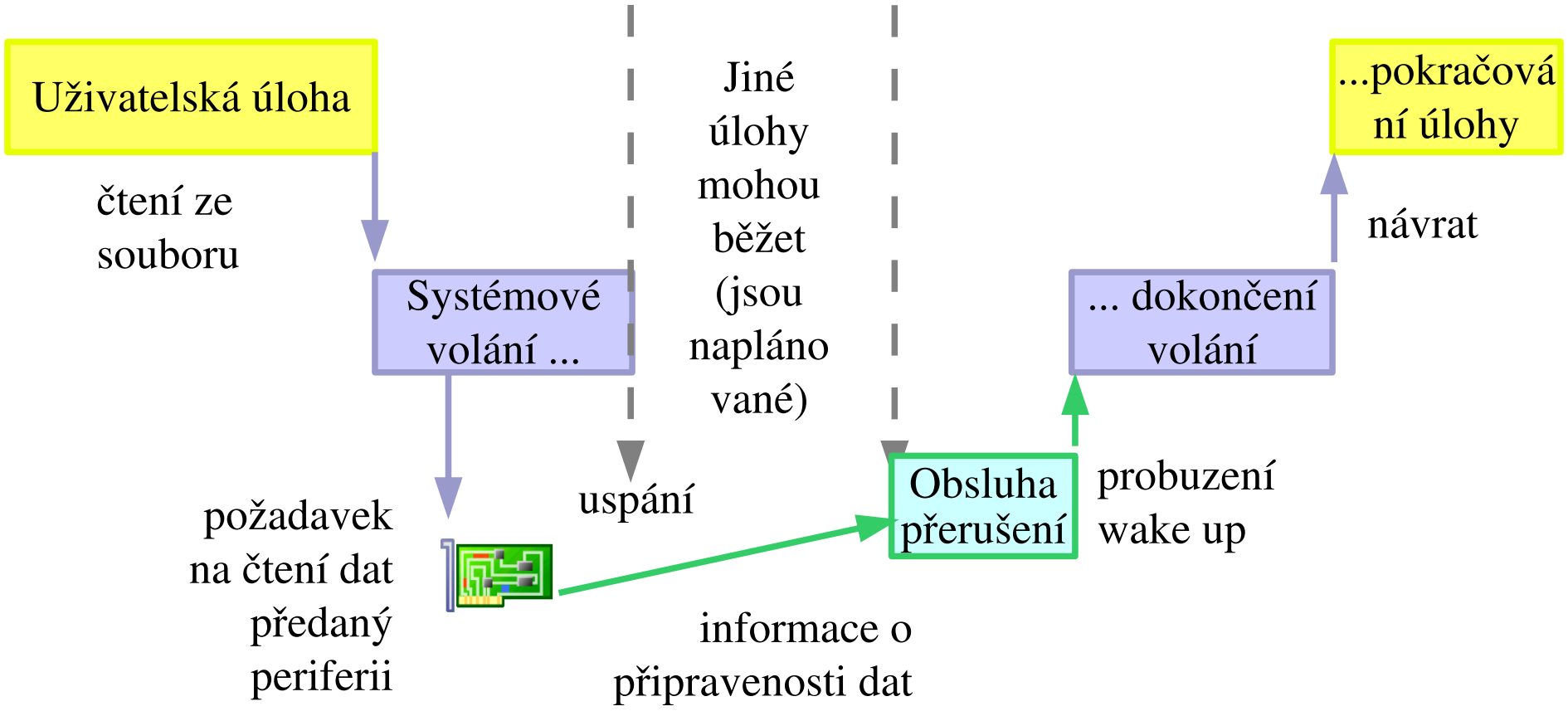
- Softwarové vyhledání (polled exception handling)
  - Veškerá přerušení a výjimky spouštějí rutinu od stejné adresy – např. standardní MIPS, adresa 0x00000004
  - Rutina zjistí důvod ze stavového registru (MIPS: cause registr)
- Vektorová obsluha přerušení (vectored exception handling)
  - Již hardware CPU zjistí příčinu/číslo zdroje
  - V paměti se nachází na pevné/řídicím registrem specifikované (VBR) adrese tabulka vektorů přerušení
  - Procesor převede číslo zdroje na index do tabulky
  - Z daného indexu načte slovo a vloží ho do PC
- Nevektorová obsluha více pevně určených adres podle priorit/důvodu
- Často jsou přístupy kombinované, např. výjimky mají oddělené cílové adresy skoků, využívá tabulku atd. ale veškerá vnější přerušení končí pouze na jednom z vektorů

Asynchronní a synchronní výjimky/přerušení

- Vnější přerušení jsou obecně asynchronní, nejsou vázané na instrukci
  - RESET- základní nulování a nastavení
  - NMI - nemaskované přerušení
  - INT - maskované přerušení
- Naopak synchronní výjimky a přerušení jsou svázané s instrukcí
  - Aritmetické přetečení, dělení nulou
  - TRAP - krokovací režim, přerušení na konci každé instrukce
  - U této skupiny může mít smysl ovlivňovat v obslužné rutině stav programu - registrů (emulace instrukcí, vykonání systémového volání atd.)

Přerušení – obsluha I/O operací na úrovni OS

V době, kdy periferie přenáší data, se úloha uspí (je možné vykonávat jinou činnost). Po příchodu dat je procesor aktivovaný, dokončí přenos a původní úloha pokračuje



Zdroj: Free Electrons: Kernel, drivers and embedded Linux development http://free-electrons.com

Obvody reálného času a hlídací obvody

- hodiny reálného času
  - údaj o reálném (astronomickém) čase
- časovač
  - periodický čas. signál (časový INT), časové funkce
- hlídací obvody
  - hlídání SW a HW výpadků systému včetně výpadku zdroje (watchdog, power fail)