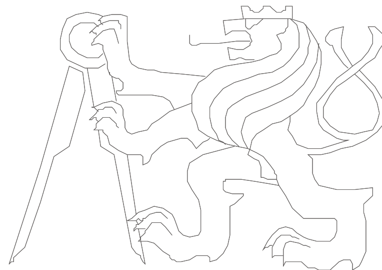
Architektury počítačů

Paměť – část druhá
1. virtuální paměť, celkový pohled

2. sekundární paměť



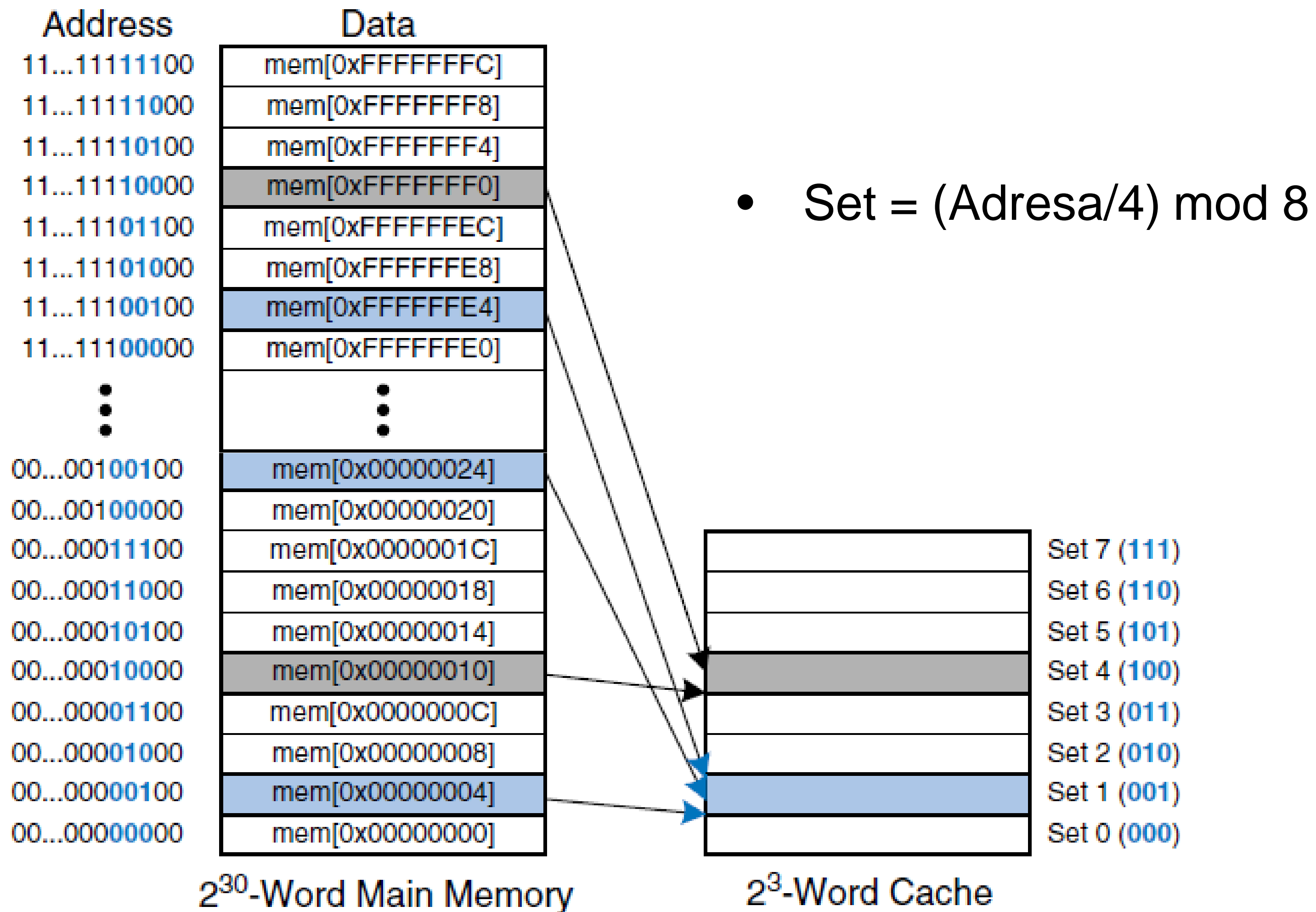
České vysoké učení technické, Fakulta elektrotechnická

Na minulé přednášce…

Příklad

- Mějme cache o velikosti 8-mi bloků. Kam se do ní umístí data z adresy 0xF0000014?
  - Plně asociativní,
  - Přímo mapované, nebo
  - S omezeným stupněm asociativity N=2 (2-cestná, 2-way cache).

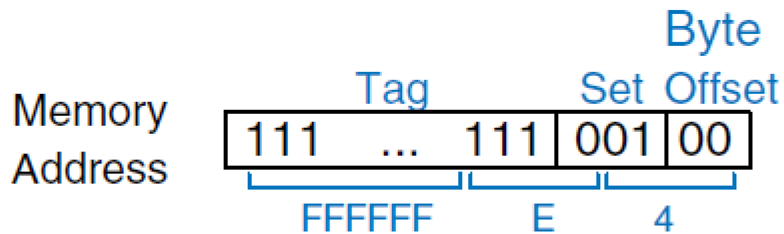
Přímo mapovaná cache



- Set = (Adresa/4) mod 8

Přímo mapovaná cache

přímo mapovaná cache: one block in each set



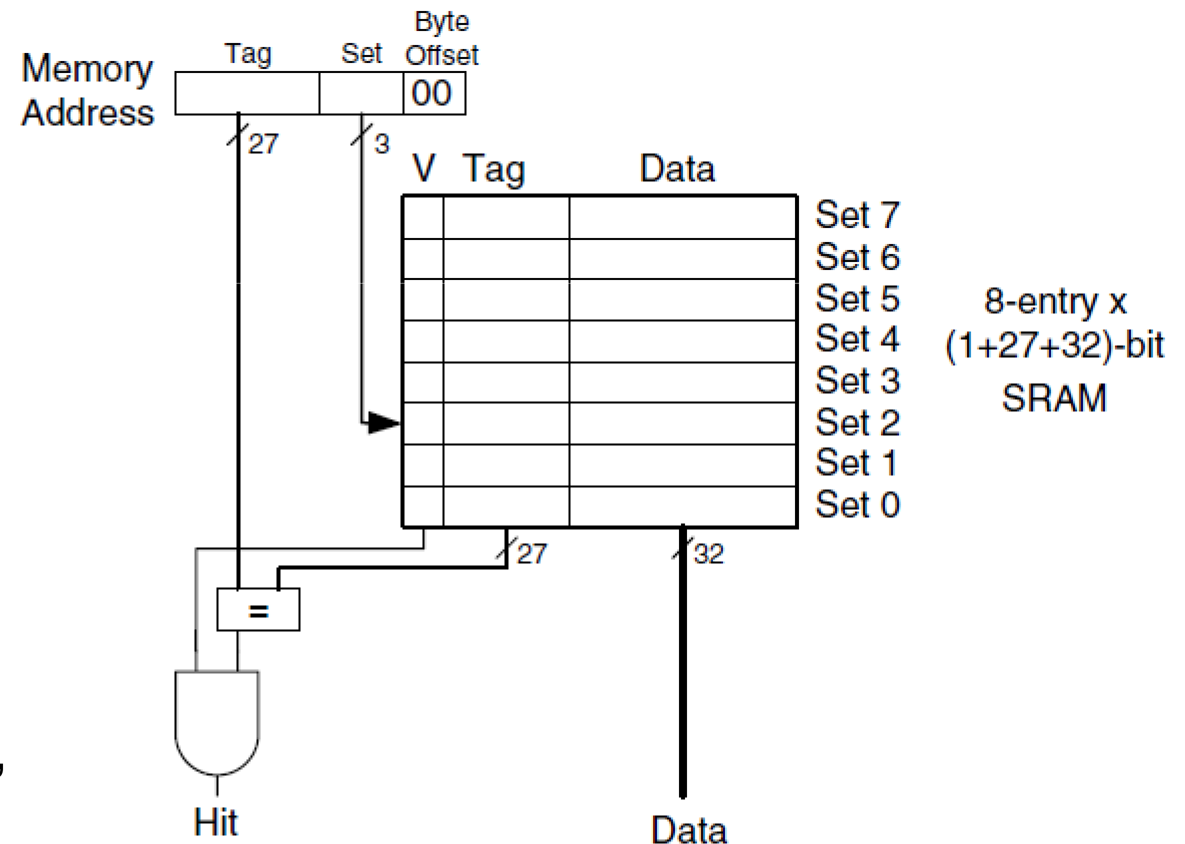
Capacity – C

Number of sets – S

Block size – b

Number of blocks – B

Degree of associativity – N



C = 8 (8 words),

S = B = 8,

b = 1 (one word in the block),

N = 1

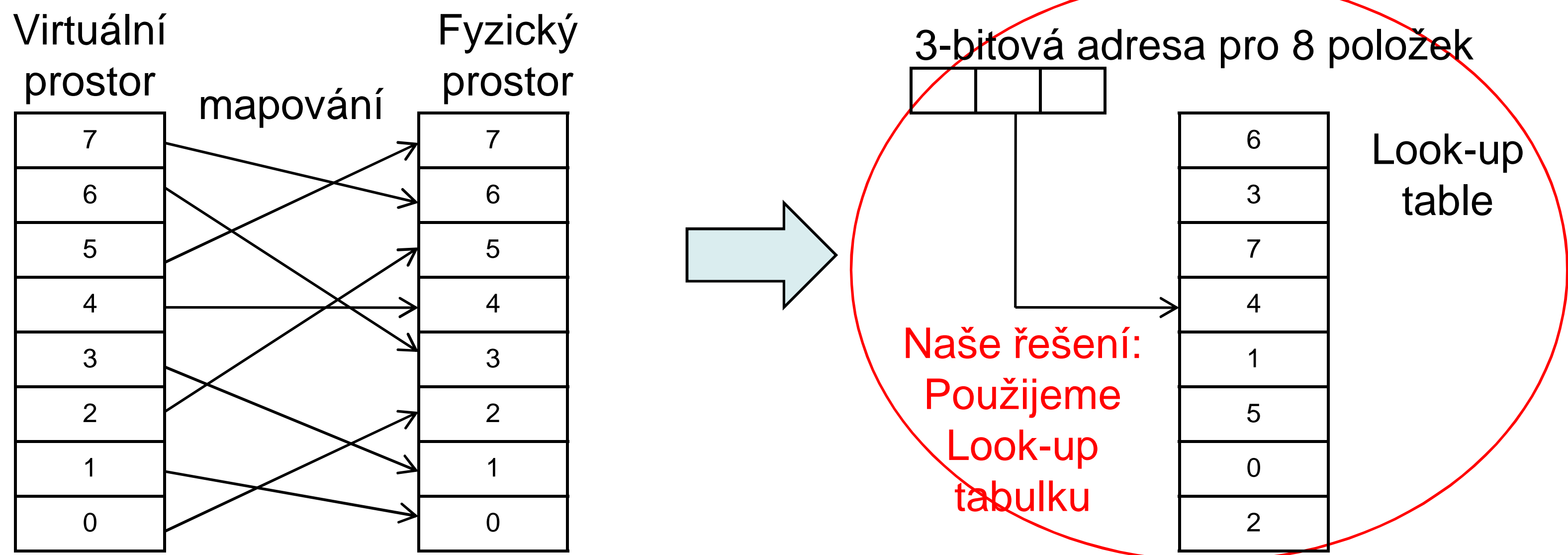
Dnešní téma…

Motivace k virtuální paměti..

- **Běžně máme na počítači spuštěno několik desítek/stovek procesů…**
- **Umíte si představit situaci, kdy bychom rozdělili fyzikou paměť (například 1 GB) mezi tyto procesy? Jak veliký kus paměti by pak patřil jednomu procesu? Jak bychom řešili kolize – kdy nějaký program úmyslně (například virus) nebo neúmyslně (chybou programátora – práce s ukazateli) by chtěl zapisovat do kusu paměti, který jsme vyhradili jinému procesu?**
- **Řešením je právě virtuální paměť…**
- **Každému procesu vytvoříme iluzi, že celá paměť je pouze jeho a může se v ní libovolně zcela bezpečně pohybovat.**
- **Dokonce každému procesu dále vytvoříme iluzi, že má k dispozici např. 4GB paměti i když je fyzická paměť mnohem menší. Proces pak nerozlišuje mezi fyzickou pamětí a diskem (disk se mu jeví jako paměť).**
- **Základní idea: Proces adresuje ve virtuální paměti pomocí virtuálních adres. Ty pak musíme nějak přeložit na adresy fyzické.**

Motivace k virtuální paměti..

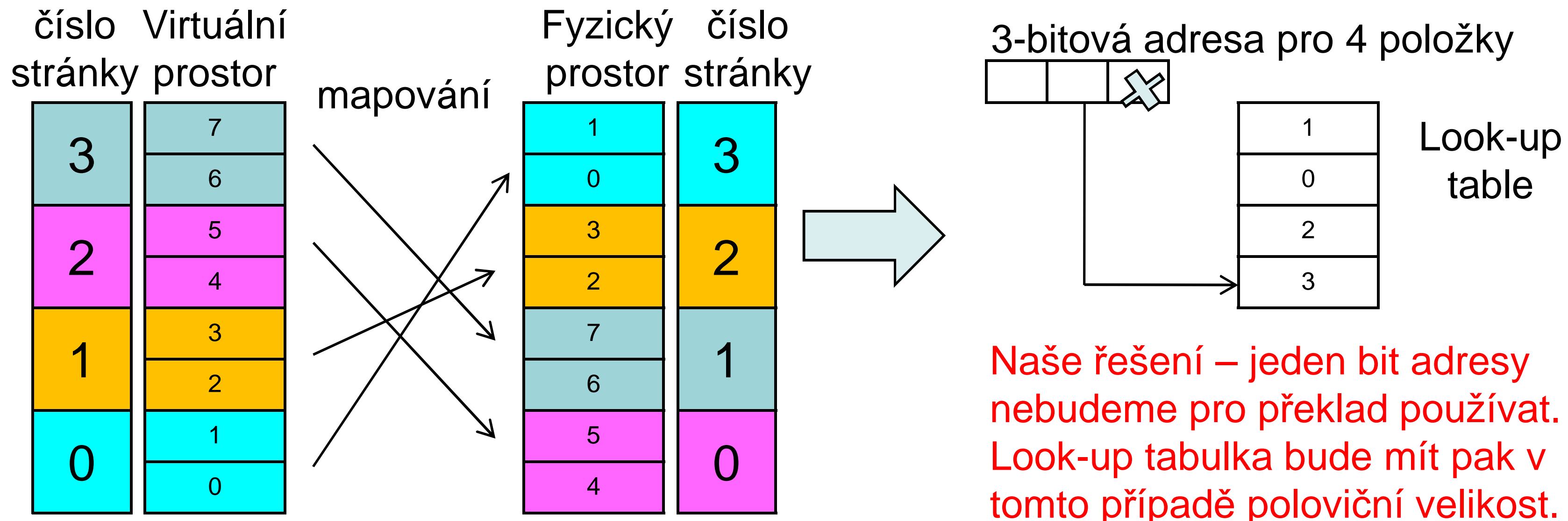
- Představme si, že máme 8B (Bajtů) virtuální prostor a 8B fyzické paměti…
- **Jak zabezpečíme překlad adres?** Předpokládejme **adresaci po bajtech**.
- **Zde je jedno řešení:** Chceme přeložit libovolnou virtuální adresu na libovolnou fyzickou adresu. Máme 3-bitovou virtuální adresu, a tu chceme přeložit na 3-bitovou fyzickou adresu. K tomu stačí tabulka o 8 záznamech, kde jeden záznam bude mít 3 bity, dohromady 8x3=24bitů/proces.



- Problém! Pokud budeme mít 4 GB virtuální prostor, naše Look-up tabulka bude zabírat 2^{32}x32 bitů = 16GB/proces!!! To je poněkud hodně…

Motivace k virtuální paměti.. - Ponaučení z předchozího slide:

- **Mapování z libovolné virtuální adresy na libovolnou fyzickou adresu je prakticky nerealizovatelný požadavek!**
- **Řešení:** Rozdělme virtuální prostor na stejně velké části – virtuální stránky, a fyzickou paměť na fyzické stránky. Ať je velikost virtuální a fyzické stránky stejná. V našem příkladu máme stránku o velikosti 2B.



- Naše řešení tedy překládá virtuální adresy po skupinách… Uvnitř dané stránky se pak pohybujeme za pomocí právě toho bitu, který jsme při překladu ignorovali.. Tím jsme schopni využít celý adresní prostor.

Příklad č.1

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   int a, b[4], *c, d;
   c = (int*)malloc(4*sizeof(int));
   printf("%p %p %p %p\n",&a,&b,&c,&d);
   printf("%p %p %p\n",&b[0],&b[1],&b[2]);
   printf("%p %p %p\n",&c[0],&c[1],&c[2]);
   free(c);
   return 0;
}
```

Co z toho vyplývá?

- Data jsou v poli uložena za sebou.

Otázky, které se nabízí..

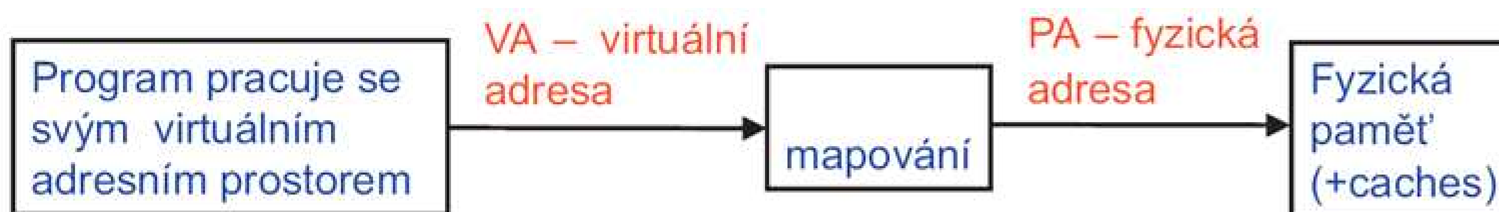
- Ale co je to za adresu?
- Kam do cache se tyto data namapují?

Výpis programu:

```
0028FF1C 0028FF0C 0028FF08 0028FF04
0028FF0C 0028FF10 0028FF14
00801850 00801854 00801858
```

Virtualizace paměti

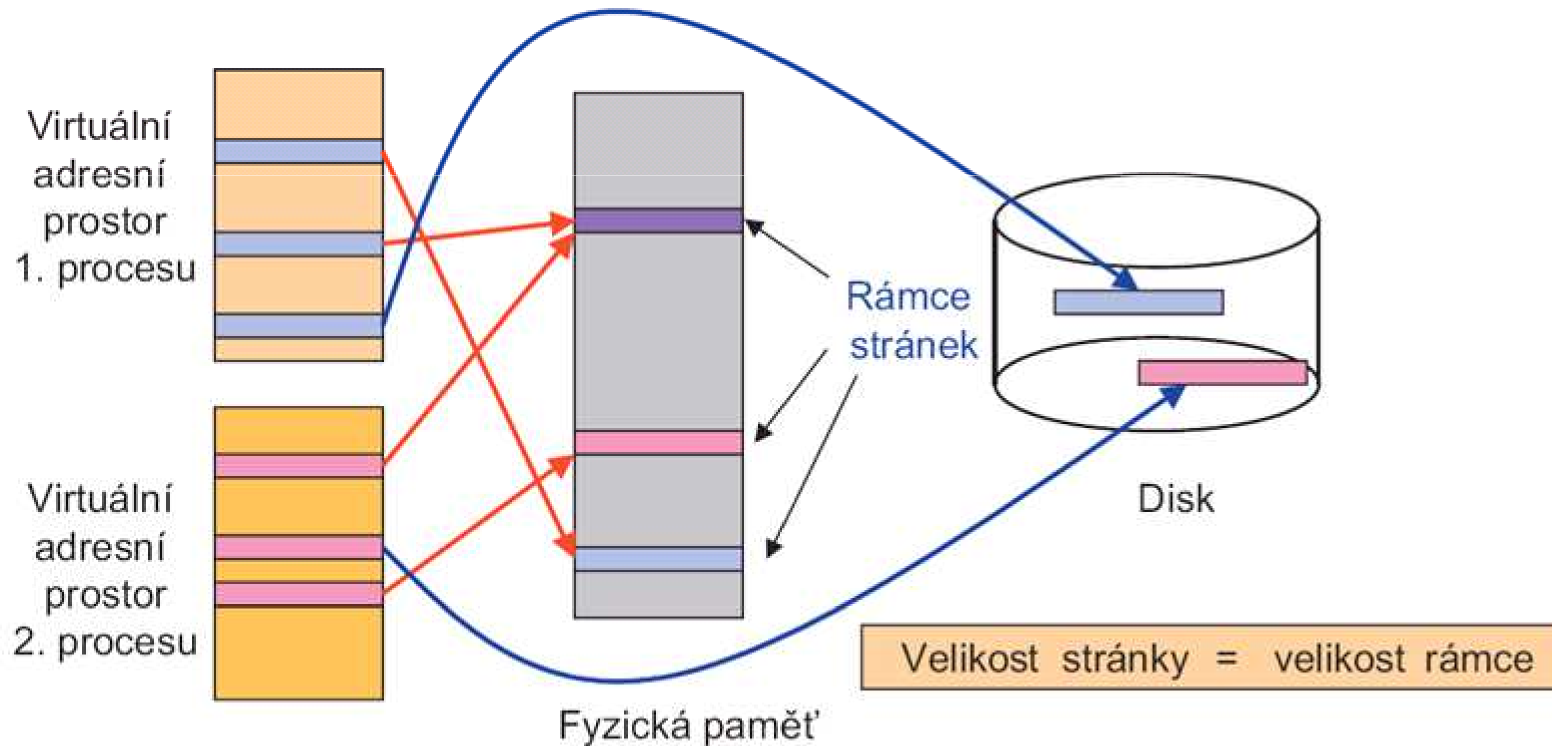
- **VP** je způsob správy operační paměti umožňující běžícímu procesu zpřístupnění paměťového prostoru, který je uspořádán jinak, nebo je dokonce větší, než je fyzicky připojená operační paměť.
- Převod mezi virtuální VA a fyzickou PA adresou může podporovat procesor (HW mapováním TLB, viz dále).
- V současně běžných operačních systémech je virtuální paměť implementována pomocí stránkování paměti spolu se stránkováním na disk, které rozšiřuje operační paměť o prostor na disku.



* R. Lórenc, X36APS, 2005

Virtuální paměť - stránkování

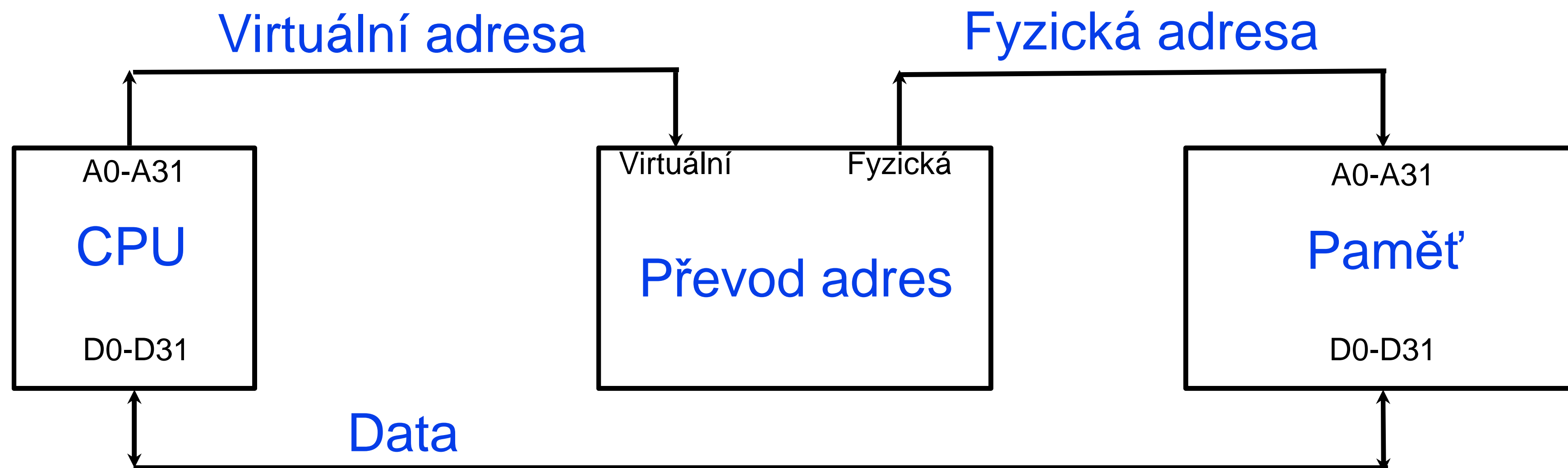
- Virtuální prostor tvoří stejně velké stránky (pages, virtual pages), které se přiřazují jednotlivým běžícím procesům.
- Fyzickou paměť tvoří stejně velké rámce (frames, physical pages).
- Zde jen poznámka: moderní přístupy nevyžadují stejně velké stránky.



Virtuální paměť - stránkování

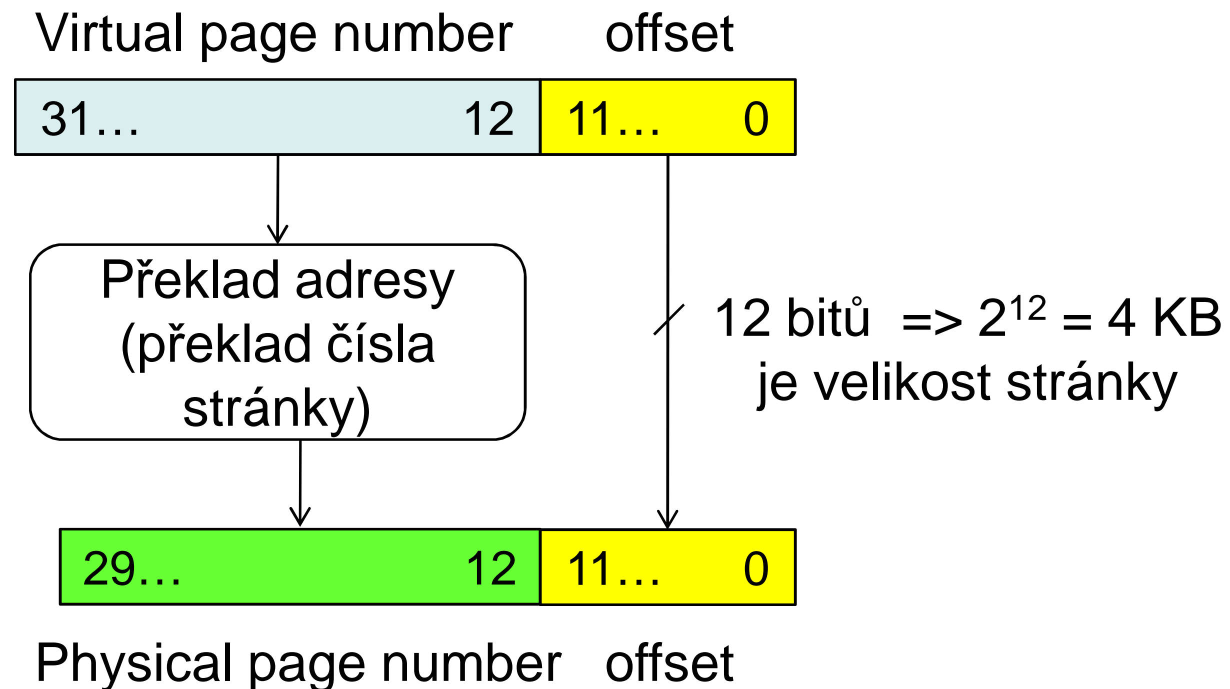
- Každé virtuální stránce může odpovídat nejvýš jedna fyzická stránka, obráceně to neplatí, takže:
- Na jednu konkrétní fyzickou stránku může být namapováno několik virtuálních stránek. Co to přináší?
- Můžeme sdílet paměť napříč různými procesy nebo vlákny (data nebo kód – OS načte sdílené knihovny jenom jednou), můžeme poskytnout jiná oprávnění (přístupová práva).
- Pokud se program snaží přistoupit do stránky způsobem, který neodpovídá jeho oprávněním, CPU generuje *General protection fault*
- handler pro General protection fault – typická reakce je ukončení procesu

Virtuální a fyzické adresování



Virtuální a fyzické adresování - detailněji

- Předpokládejme virtuální adresu o délce 32 bitů, 1GB fyzické paměti a velikost stránky 4 KB



- Jaký **velmi důležitý** praktický důsledek má toto uspořádání překladu (tj. nejnižší bity adresy zůstávají zachovány) ?

Vraťme se k příkladu č.1

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   int a, b[4], *c, d;
   c = (int*)malloc(4*sizeof(int));
   printf("%p %p %p %p\n",&a,&b,&c,&d);
   printf("%p %p %p\n",&b[0],&b[1],&b[2]);
   printf("%p %p %p\n",&c[0],&c[1],&c[2]);
   free(c);
   return 0;
}
```

Výpis programu:

```
0028FF1C 0028FF0C 0028FF08 0028FF04
0028FF0C 0028FF10 0028FF14
00801850 00801854 00801858
```

Vraťme se k příkladu č.1

- Všimli jste si adres, na kterých se nachází proměnné ***a*, c, *d*** a pole ***b***?
- Co když budeme chtít rozšířit náš program třeba o příkazy:

```
a = 1;
b[0] = a+1;
b[1] = b[0]+1;
d = b[2];
```

//b[2] neinicializováno..

Vraťme se k příkladu č.1

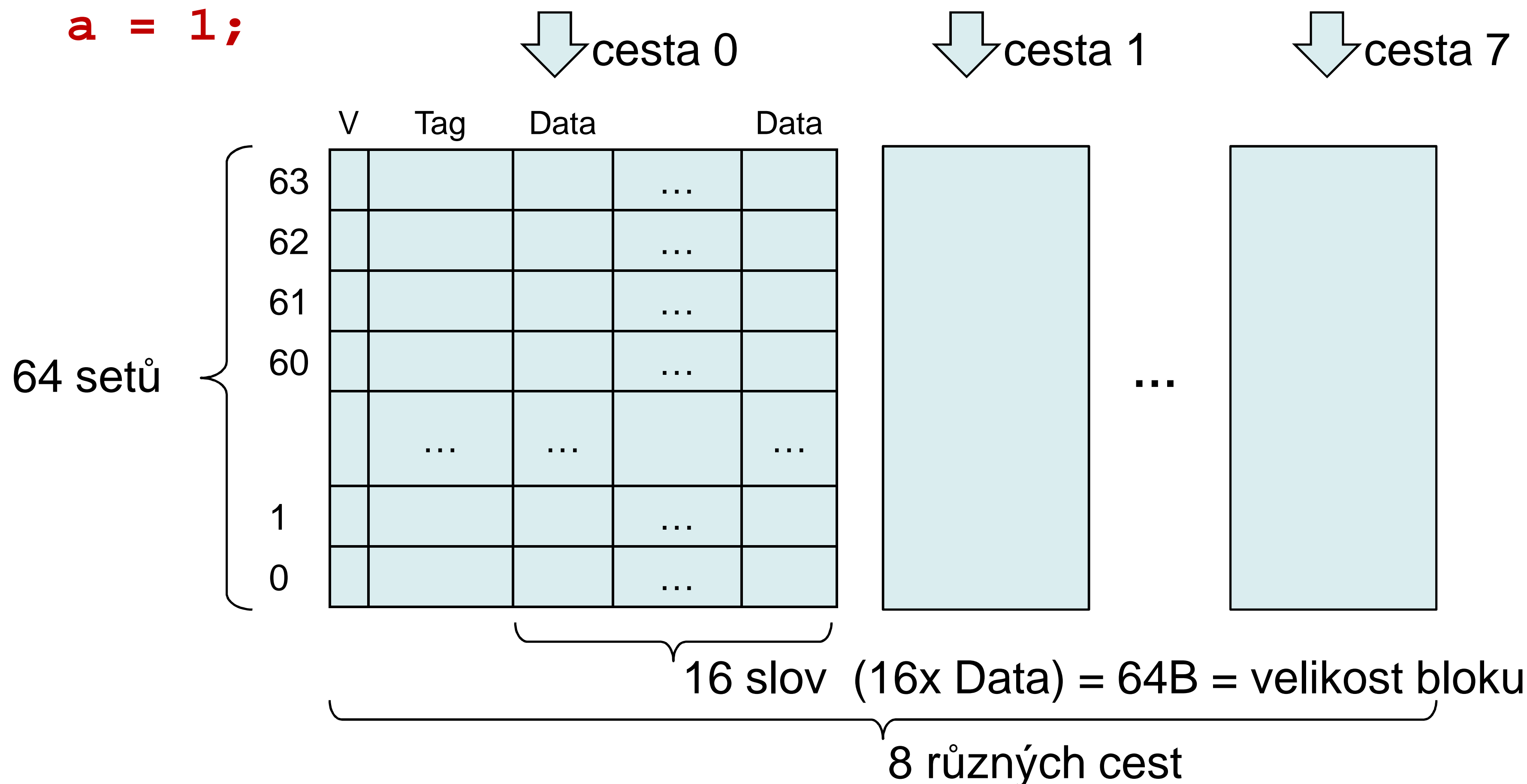
- Předpokládejme L1 datovou cache o velikosti 32kB se stupněm asociativity 8, a velikostí bloku 64B. Cache je na počátku prázdná.
- Co všechno se stane když vykonáme první řádek programu?

```
a = 1;
b[0] = a+1;
b[1] = b[0]+1;
d = b[2];
```

Vraťme se k příkladu č.1

- Předpokládejme L1 datovou cache o velikosti 32kB se stupněm asociativity 8, a velikostí bloku 64B. Cache je na počátku prázdná.
- Co všechno se stane když vykonáme první řádek programu?

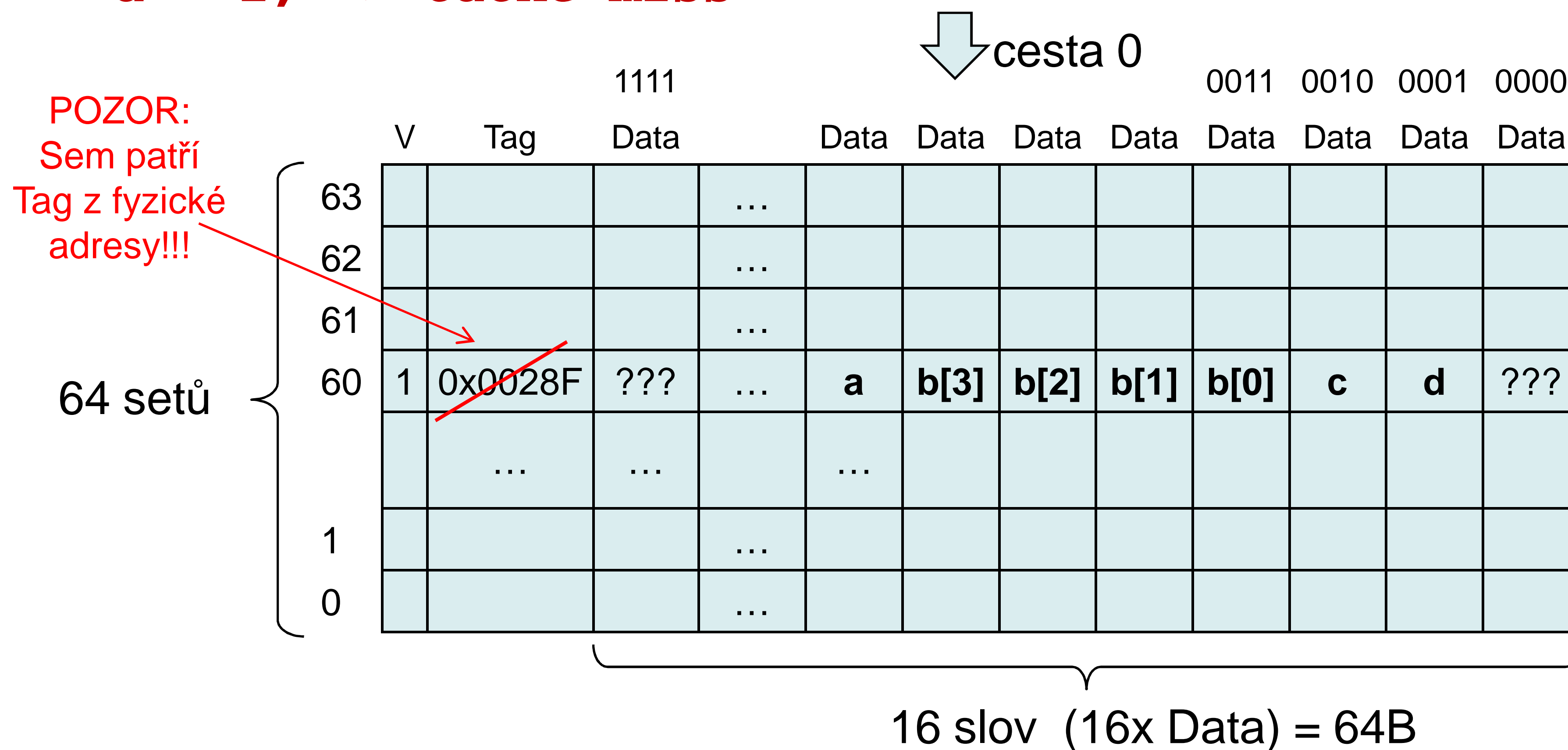
```
a = 1;
```



Vraťme se k příkladu č.1

- Předpokládejme L1 datovou cache o velikosti 32kB se stupněm asociativity 8, a velikostí bloku 64B. Cache je na počátku prázdná.
- Co všechno se stane když vykonáme první řádek programu?

```
a = 1; -> cache miss
```



Vraťme se k příkladu č.1

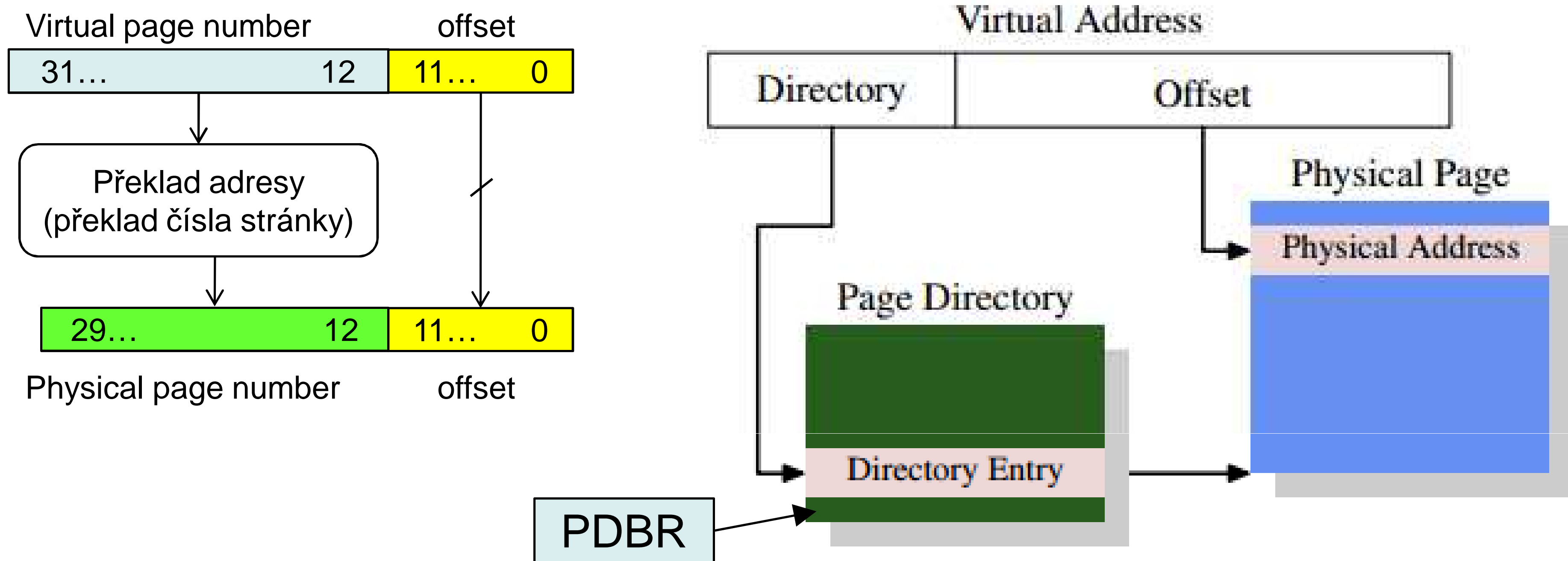
Závěry:

- Stránkování (realizace virtuální paměti) nenarušuje princip prostorové lokality => důležité pro cache.
- **Data na sousedících virtuálních adresách budou uloženy ve fyzické paměti vedle sebe (pokud nepřekročí hranici stránky).**
- Pokud nastane page fault (stránka je na disku) jako důsledek cache miss, pak se celá stránka z disku přesune do paměti a z té se pak celý blok (cache line) přesune do cache. Další cache miss uvnitř stránky již nevyvolá page fault (dokud nebude stránka nahrazena jinou stránkou).

Realizace převodu adres?

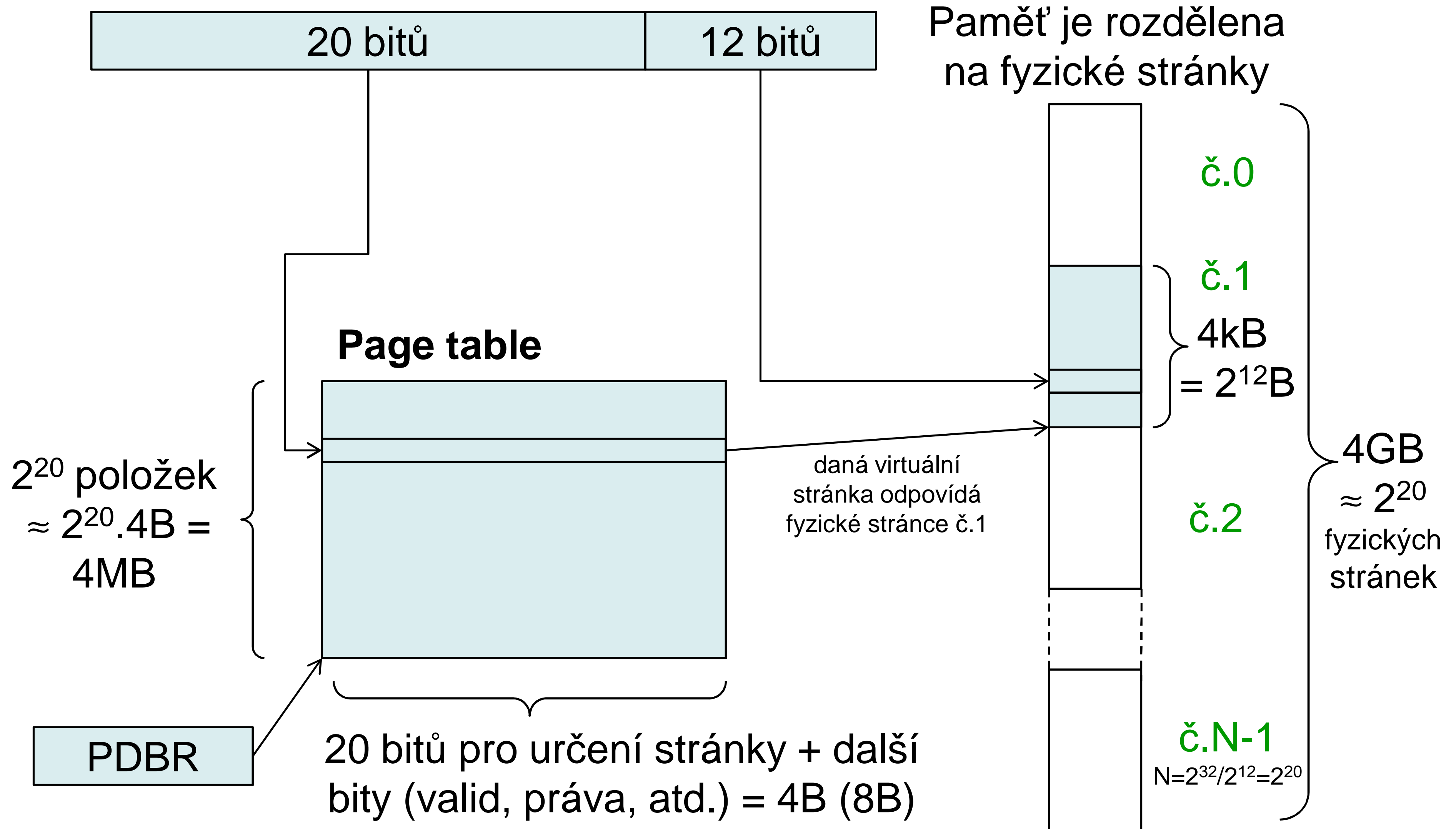
- Tabulka stránek, Page Table.
- Jednotkou mapování jsou stránky,
- Stránka je také jednotkou přenosu mezi vedlejší a hlavní paměti.
- Mapovací funkce se nejčastěji implementuje Look-up Table (vyhledávací tabulkou).
- O překlad virtuálních adres na fyzické se stará **Memory Management Unit** (**MMU**)
- MMU je součástí CPU
- Příklad:

Realizace převodu adres?



- Datová struktura pro Page Directory (Page Table) je uložena v hlavní paměti. Úkolem operačního systému je alokovat souvislou oblast paměti a počáteční adresu této oblasti uložit do speciálního registru CPU.
- PDBR - page directory base register – v x86 v registru CR3 – obsahuje fyzickou adresu
- PTBR - page table base register – to samé…

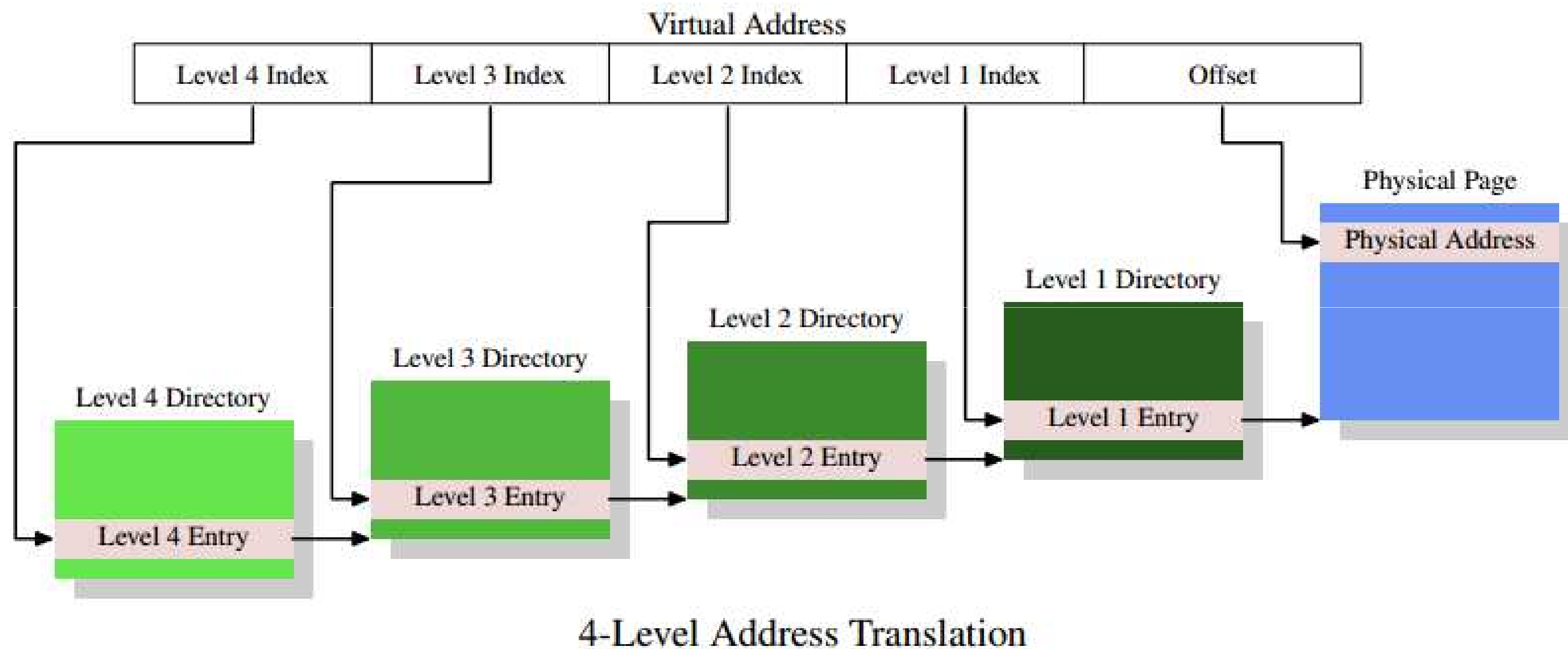
Realizace převodu adres?



Uvažujme…

- Stránka je typicky 4 kB = 2^12
- Když budeme znát adresu stránky, postačuje nám tedy jenom 12 bitů na pohyb (adresaci) v ní. Zbývá 20 bitů (pro 32-bitovou adresu).
- Tudíž Page Directory (Page Table) by měl obsahovat 2^20 položek. To je nepraktické a přináší řadu nevýhod.
- Typický proces/vlákno se v daném „okamžiku“ pohybuje pouze v malé části svého adresního prostoru – princip časové a prostorové lokality…
- Řešením je více-úrovňové stránkování.

Více-úrovňové stránkování



4-Level Address Translation

Více-úrovňové stránkování – 2 úrovně

Více-úrovňové stránkování

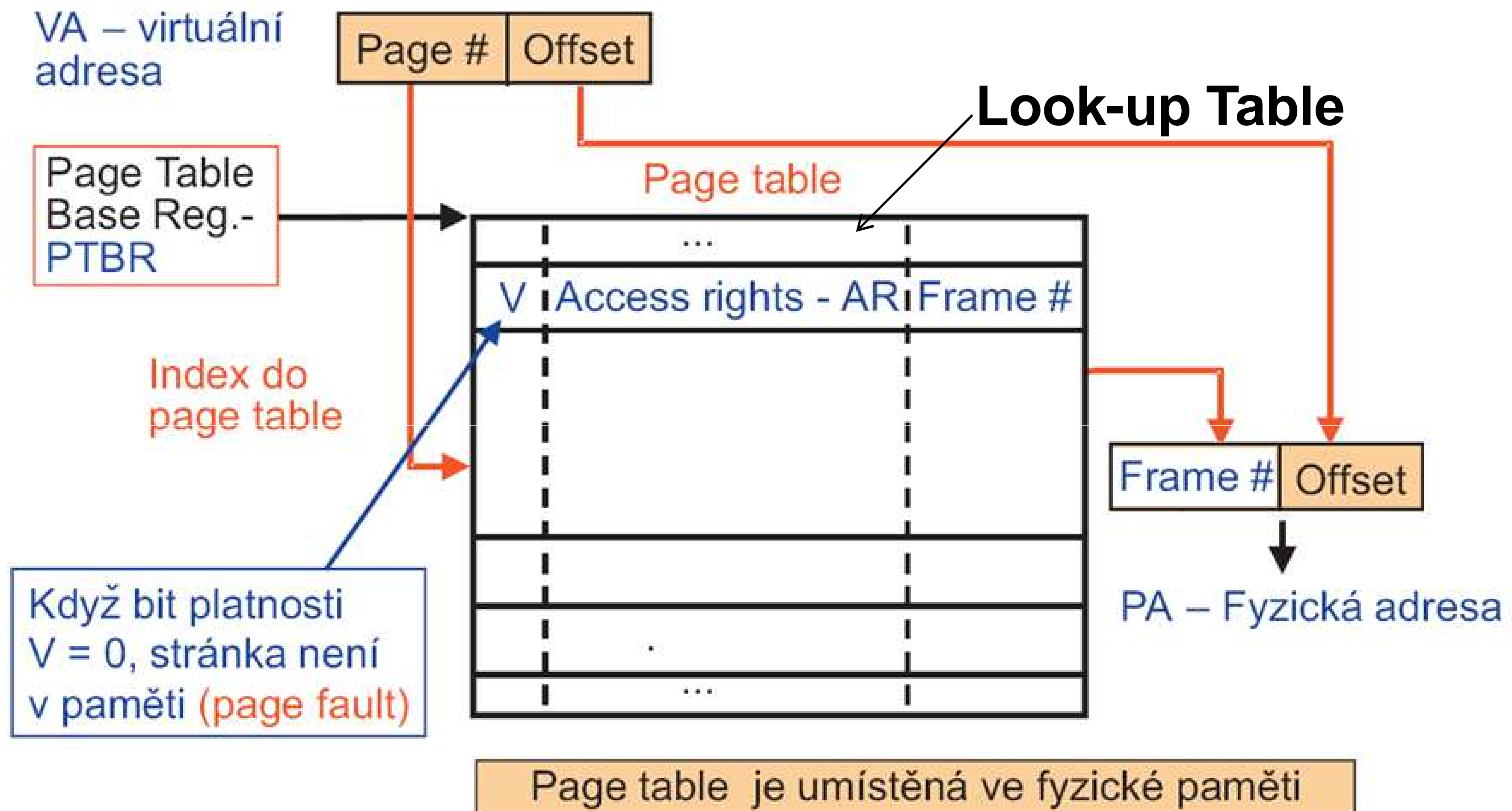
Poznámky k předchozímu slide:

- Ne každý proces využívá celý svůj adresní prostor => není nutné alokovat v druhé úrovni 2^{10} Page tables
- Tabulky stránek můžou být rovněž stránkovány

Obecné poznámky:

- Intel IA32 implementuje 2-úrovňové stránkování
  - Page Table v úrovni 1 označuje jako Page Direcory (10 bitů pro adresaci)
  - Page Table v úrovni 2 pak jako Page Table (10 bitů)
- V případě 64-bitové virtuální adresy je obvyklé používat méně bitů pro fyzickou adresu – například 48, nebo 40.
- Intel Core i7 používá 4-úrovňové stránkování a 48 bitový adresní prostor
  - Page Table v úrovni 1: Page global directory (9 bitů)
  - Page Table v úrovni 2: Page upper directory (9 bitů)
  - Page Table v úrovni 3: Page middle directory (9 bitů)
  - Page Table v úrovni 4: Page table (9 bitů)

Tabulka stránek – jak vypadají položky? Význam položek…



Tabulka stránek – jak vypadají položky? Význam položek…

Podívejme se na položku Page Directory (Page Table na 1.úrovni)

31…	1	0
Dostupné pro Operační Systém		P=0

31… 12	… 6	5	4	3	2	1	0
Bázová adresa Page table	…	A	PCD	PWT	U/S	R/W	P=1

- bit 0: Present bit – určuje zda je stránka v paměti (1), nebo na disku (0) Někdy se tento bit označuje V – valid.
- bit 1: Read/Write: pokud 1 – R/W; pokud 0 – jenom ke čtení
- bit 2: User/Supervisor: 1 – uživatelský přístup; 0 – pouze OS
- bit 3: Write-through/Write-back – zapisovací strategie pro stránku
- bit 4: Cache disabled/enabled – některé periferie jsou namapovány přímo do paměti (memory mapped I/O), čímž je umožněn zápis/čtení do/z dané periferie. Tyto adresy v paměti pak chápeme jako I/O porty. Nekešujeme.
- bit 5: Accessed – zda jsme četli/zapsali – pomáhá rozhodovat o tom, které stránky mají být odstraněny když potřebujeme uvolnit prostor v paměti

Tabulka stránek – jak vypadají položky? Význam položek…

Podívejme se na položku Page Table (Page Table na 2.úrovni)

31…	1	0
Dostupné pro Operační Systém		P=0

31… 12	7	6	5	4	3	2	1	0
Bázová adresa Page	…	D	A	PCD	PWT	U/S	R/W	P=1

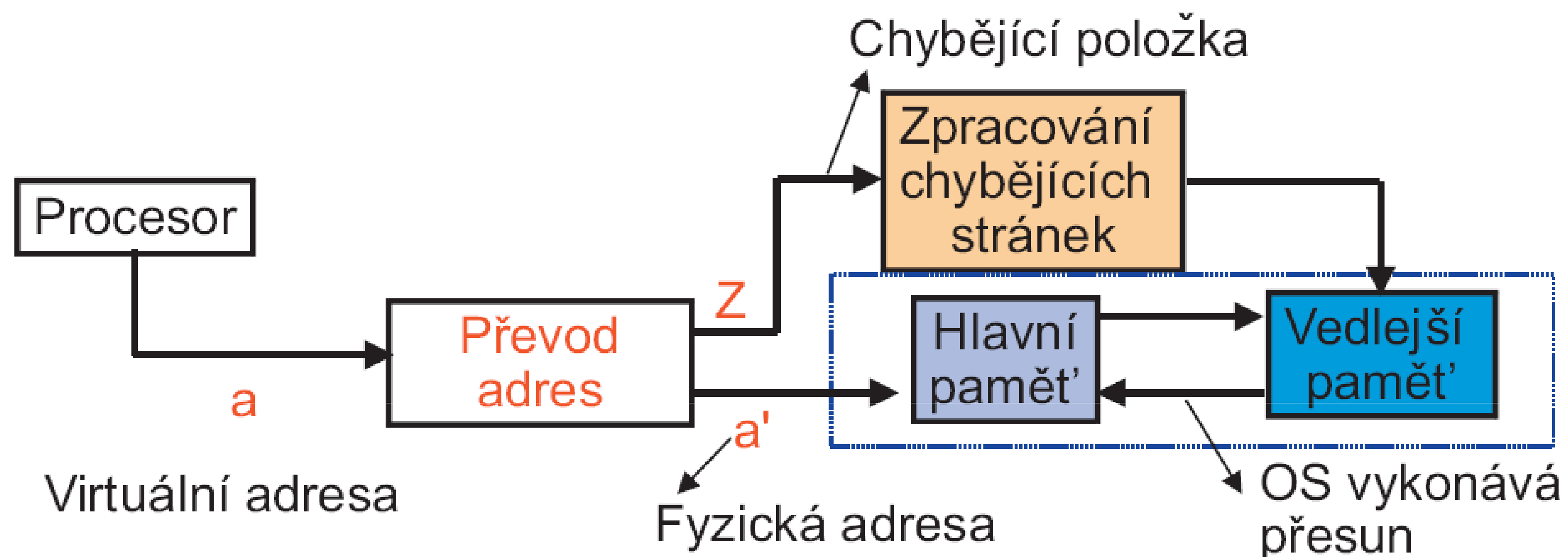
- bit 6: Dirty bit – Je nastaven pokud jsme do stránky zapsali.

Poznámky

- Každý proces má svou Tabulku stránek,
- Tedy i svou hodnotu PTBR (bázového registru).
- To, mimochodem, zajišťuje paměťovou bezpečnost procesů.
- Co chceme abyste si zapamatovali z Formátu položky Tabulky stránek?
  - V – Validity Bit. V=0 Stránka není platná (je na disku).
  - AR – Access Rights. Přístupová práva (Read Only, Read/Write, Executable, apod.),
  - Frame# - číslo rámce (bázová adresa do nižší úrovně),
  - Popřípadě další, např. Modified/Dirty, apod. (budeme dále podle potřeby doplňovat).

V	AR	Frame#

Virtuální paměť: spolupráce HW a SW



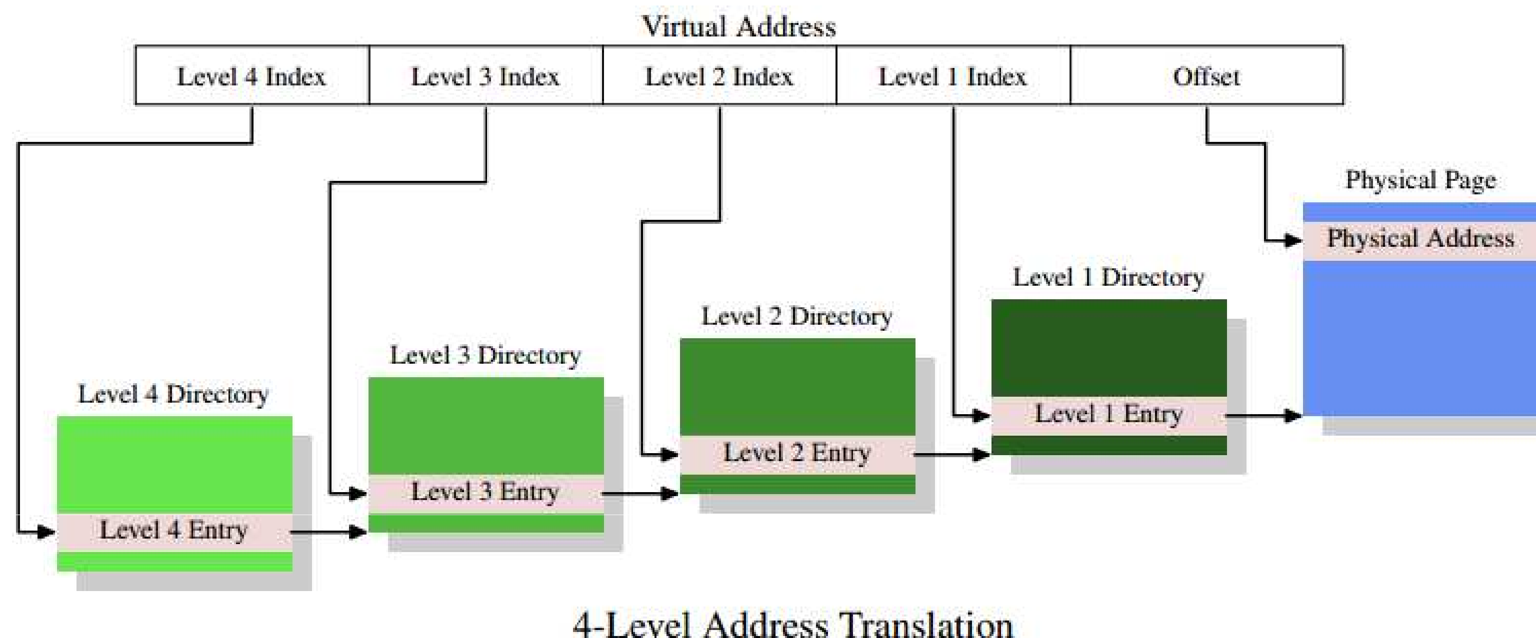
Co dělat, když je výpadek stránky – Page Fault?

- Fyzická paměť je volná, ale
  - Rámec je prázdný, data jsou ve vedlejší paměti (na disku).
  - Požadovaná stránka se „nějak“ (DMA, Direct Memory Access, přímým přístupem do paměti, ale to zde neřešíme) načítá do prázdného rámce. Přepne se na případně čekající proces, který může probíhat.
  - Po dokončení DMA přenosu se vyvolá přerušení, aktualizuje se Tabulka stránek procesu.
  - Přepne se zpět na původní proces.
- Paměti je nedostatek
  - Pomocí LRU najdeme rámec, který můžeme uvolnit.
  - Má-li nastaven Dirty bit, zapíšeme stránku do vedlejší paměti (na disk).
  - Aktualizuje se Tabulka stránek procesu.

Virtuální paměť a soubory na disku…

- Virtuální paměť rozšiřuje fyzickou paměť o prostor na disku tím, že automaticky odkládá/načítá stránky na disk (swapování). Toho lze využít…
- **Načtení programů a knihoven do paměti:**
  - Programy a knihovny jsou uloženy na disku jako binární soubory obsahující instrukce a data
  - Když chceme spustit nový program:
    - Jádro OS alokuje souvislou množinu virtuálních stránek (dostatečně velký prostor pro uchování vlastního programu a dat)
    - Poté OS aktualizuje Page table procesu (Page tables pak odkazují na soubory na disku)
    - Položky Page table jsou označeny jako Valid=0 (na disku)
  - Jakmile program běží, správa virtuální paměti načte program do paměti automaticky…
  - Viz **mmap()** – funkce alokuje virtuální stránky a nastaví položky Page table tak, aby odkazovaly na soubor na disku

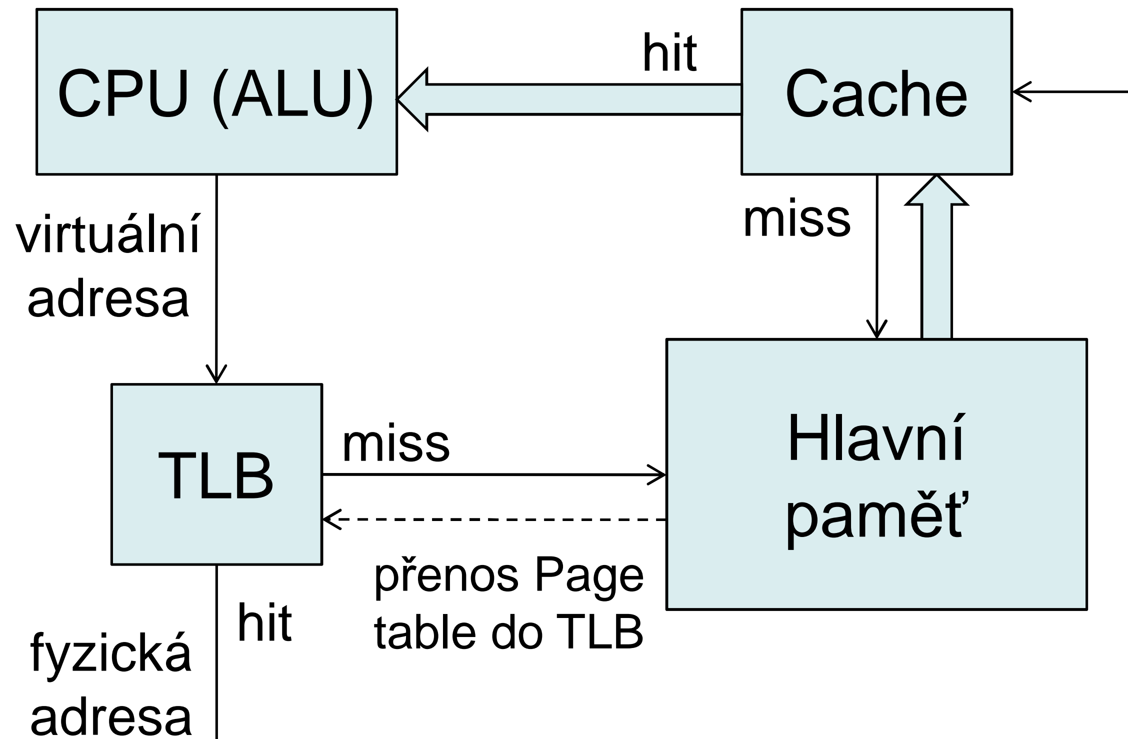
Více-úrovňové stránkování – **Problém rychlosti**



4-Level Address Translation

- Pokud bychom předpokládali, že všechny položky pro výpočet adresy máme již v cache, bude i tak výpočet adresy trvat velmi dlouhou (v závislosti od počtu úrovní – nelze paralelizovat).
- Výhodnější je přímo cachovat „vypočtené" adresy.
- K tomu slouží Translation Look-Aside Buffer (TLB)
- Dnes se používají více-úrovňové TLB

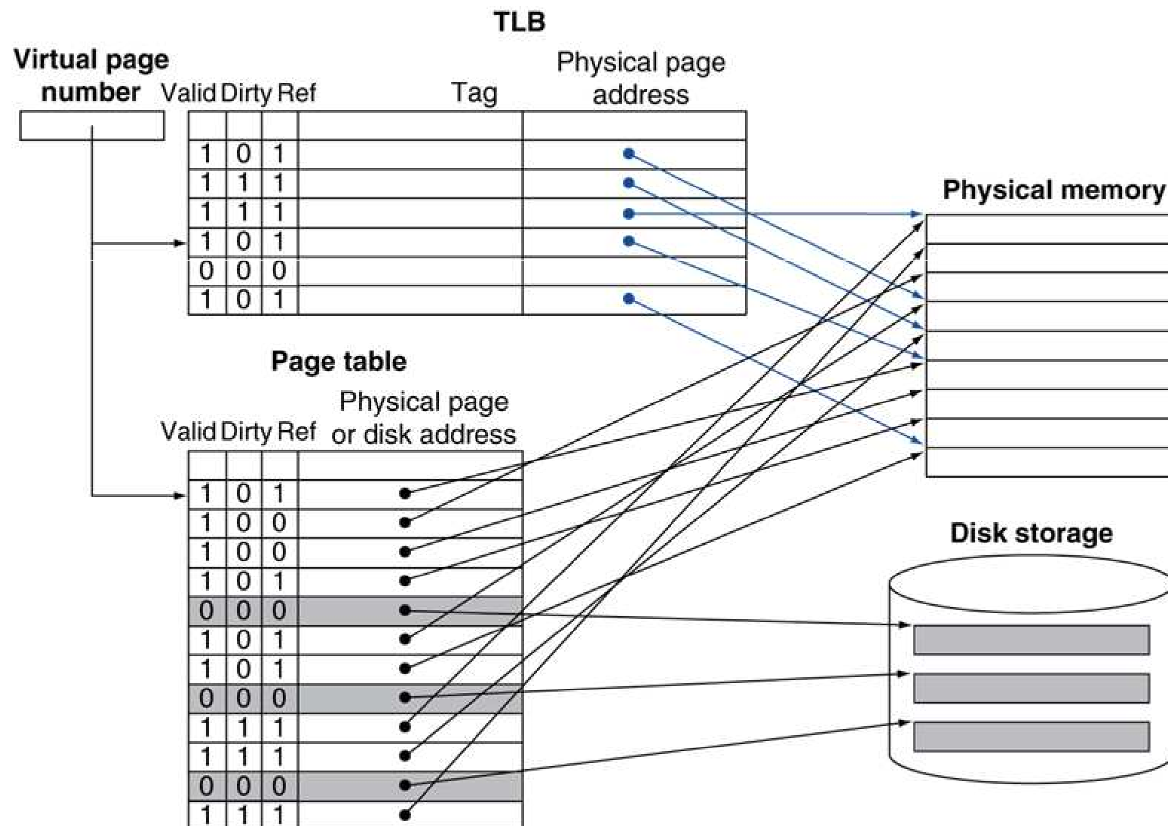
Idealizace překladu adres pomocí TLB - čtení



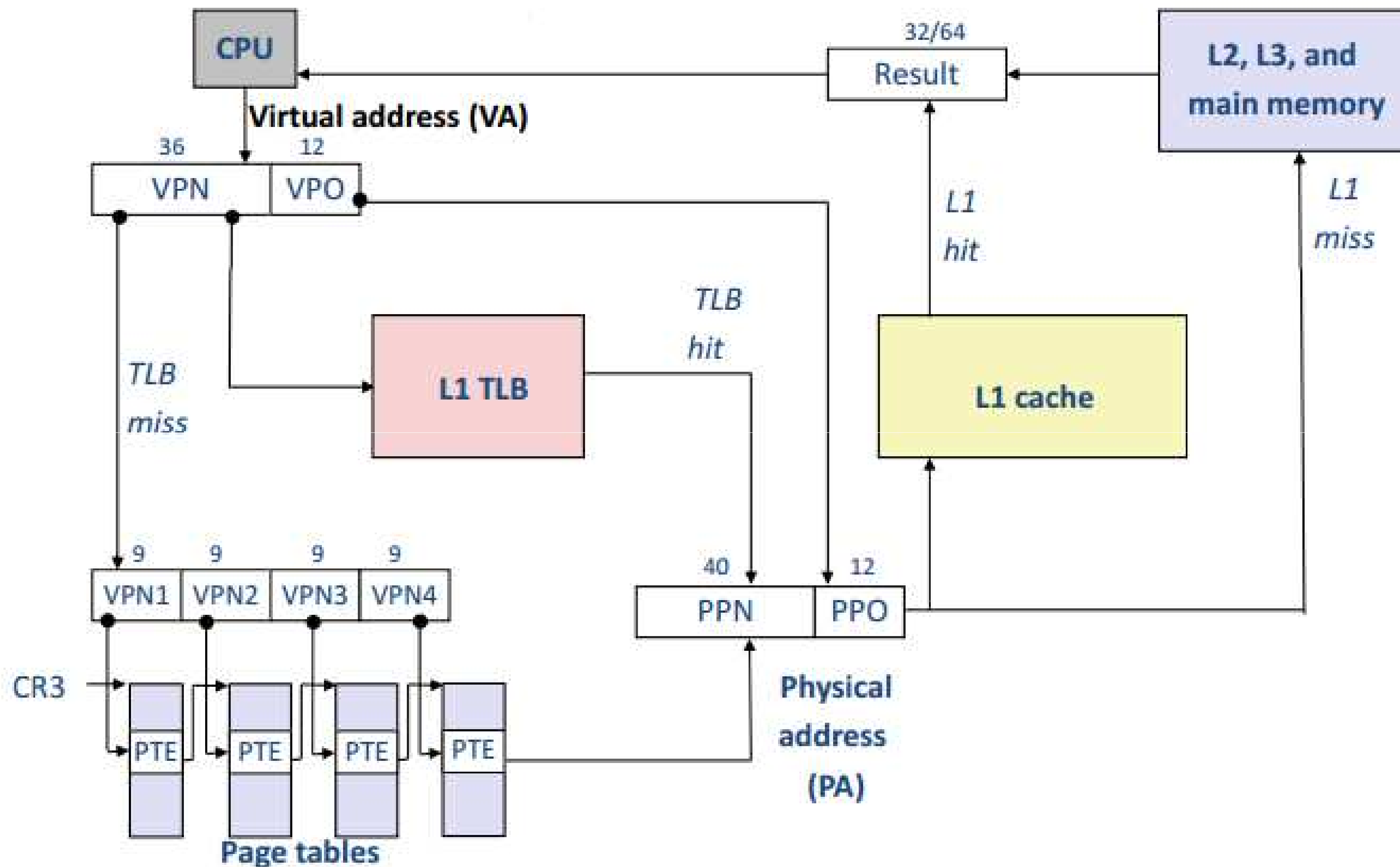
- Všimněte si, že může dojít k miss-u 2x
- Pokud nastane TLB miss, musíme vykonat tzv. *page walk*

Rychlá realizace Tabulky stránek - TLB

- Translation-lookaside Buffer, výstižnější by byl termín překládací keš (Translation Cache).
- Je vlastně SP (keší) adres stránek.

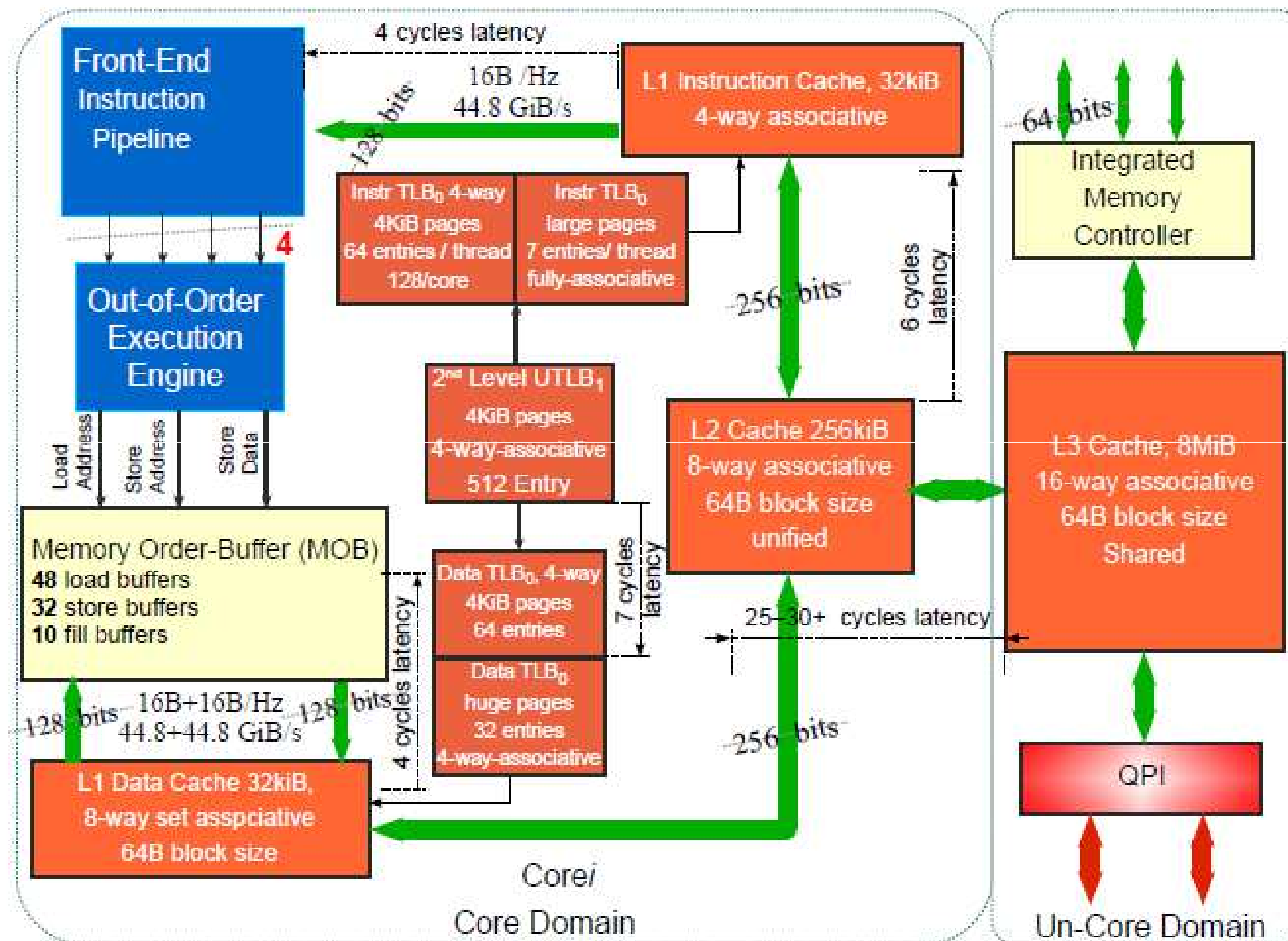


Překlad adres – Intel Nehalem (Core i7)

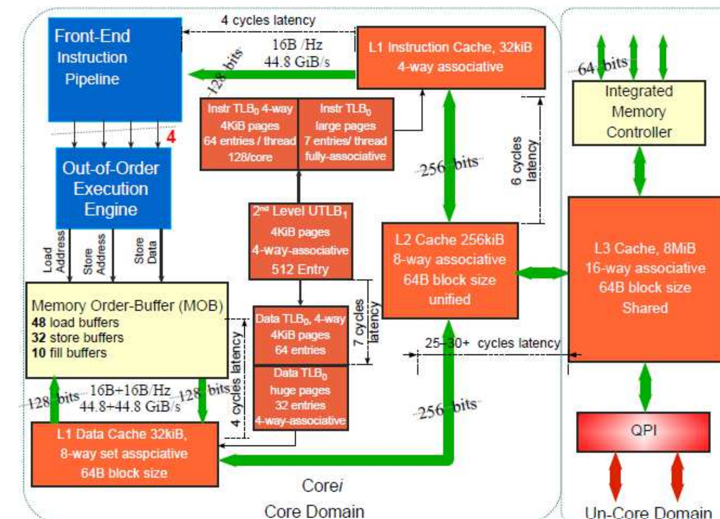


http://cs.nyu.edu/courses/spring13/CSCI-UA.0201-003/lecture18.pdf

Organizace paměti - Intel Nehalem (Core i7)



Organizace paměti - Intel Nehalem – několik poznámek



- Velkost bloku: 64B
- procesor vždy čte řádek cache ze systémové paměti zarovnán na 64B (6 LSb adresy jsou nuly) a nepodporuje částečně plněné řádky
- L1 – Harvard. V SMT sdílená oběma vlákny, Instrukční – 4-way, Datová 8-way.
- L2 – unifikovaná, 8-way, neinkluzivní, WB
- L3 – unifikovaná, 16-way, inkluzivní (řádek obsažen buď v L1 nebo L2 se nachází v L3), WB
- Store Buffers – dočasně uchovávají data pro každý zápis. Netřeba čekat na zápis do cache či paměti. Zajišťují, že zápisy jsou ve správném pořadí a také když je potřeba:
  - výjimka, přerušení, instrukce serializace, lock,..
- Můžete si také všimnout oddělených TLB (Translation Lookaside Buffer)

Pro vaší představu: typické hodnoty

	Typicky pro stránkované paměti	**Typicky pro TLB**
Velikost v blocích	16 000-250 000	40-1024
Velikost	500-1 000 MB	0,25-16 KB
Velikost bloku v B	4 000-64 000	4-32
Miss penalty (v hod)	10 000 000 – 100 000 000	10-1 000
Miss rates	0,00001-0,0001%	0,01-2

Efektivnější používáni paměti – prostředek zrychlení programu

Váš program může brát v potaz velikost stránky a používat paměť efektivněji – jednak zarovnáním alokací na násobek velikosti stránky a pak redukcí interní a externí fragmentace stránek.. (pořadí alokací atd. Viz také *memory pool*)

```
#include <stdio.h>
#include <unistd.h>
int main(void) {
      printf(„Velikost stranky je: %ld B.\n",
             sysconf(_SC_PAGESIZE));
      return 0;
}
```

Akolace paměťově zarovnaného bloku:

```
void * memalign(size_t size, int boundary)
void * valloc(size_t size)
```

windows

```
#include <stdio.h>
#include <windows.h>

int main(void) {
    SYSTEM_INFO s;
    GetSystemInfo(&s);
    printf("Velikost stranky je: %ld B.\n",
       ns.dwPageSize);
    printf("Rozsah adres pro aplikaci (a dll):
       0x%lx - 0x%lx\n",
       s.lpMinimumApplicationAddress,
       s.lpMaximumApplicationAddress);
    return 0;
}
```

Problémy hierarchických pamětí

Některé problémy hierarchických pamětí?

- Koherence pamětí. Definice viz další slajd
- Jednoprocesorové (jednojádrové) stroje.
  - Řeší D-bit a migrační strategie Write-back.
- Multiprocesory se společnou i sdílenou pamětí – řešení je mnohem složitější. Používá se mj.
  - Společná sběrnice: Snooping (s odposlechem, slíděním), MESI protokol,
  - Broadcast (s rozesíláním),
  - Directories (adresáře).
- Je obsahem předmětu A4M36PAP.

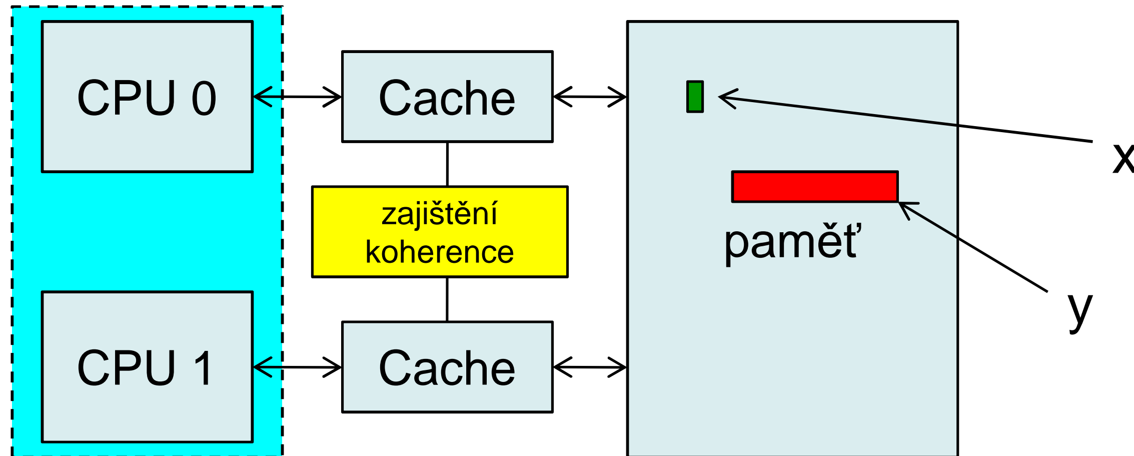
Definice koherence

- Řekneme že multiprocesorový **paměťový systém je koherentní** jestliže výsledek jakéhokoli provádění programu je takový, že pro každé paměťové místo je možné sestavit myšlené sériové pořadí čtení a zápisů k tomuto paměťovému místu a platí

- 1. Paměťové operace k danému paměťovému místu pro každý proces jsou provedeny v pořadí, ve kterém byly spuštěny tímto procesem.

- 2. Hodnoty vracené každou operací čtení jsou hodnotami naposledy provedené operace zápis do daného paměťového místa vzhledem k sériovému pořadí.

Problém koherence



Proto je důležité, aby byl systém paměťově koherentní – viz *cache coherence*

Nicméně i v paměťově koherentním systému může nevhodný programátorský styl vést k značnému zpomalení běhu programu…

Příklad A:

Vlákno 0:	Vlákno 1:
…	…
x=1;	x=3;
…	…
if(x==1)	

Nechť x je sdílená proměnná, y sdílené pole.

Příklad B:

Vlákno 0:	Vlákno 1:
…	…
y[1]=1;	y[0]=0;
…	…
y[3]=3;	y[2]=2;
…	…
y[5]=5;	y[4]=4;

Srovnání VPxSP, nenechte se zmást

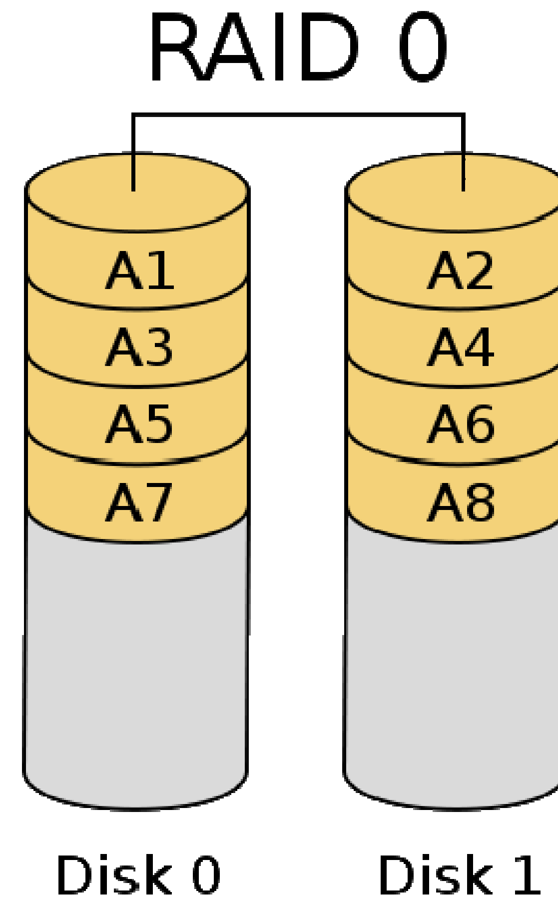
Virtuální paměť	Skrytá paměť
Stránka	Blok/řádek
Page Fault	Read/Write Miss
Velikost stránky: 512 B – 8 KB	Velikost bloku: 8 – 128 B (64B)
Plně asociativní	(DM), N-cestná, plně asociativní
Výběr oběti: LRU	LRU, ARC, CAR
Write Back	Write Back

- Pozn.: TLB virtuální paměti může být plně asociativní, ale pro větší TLB typicky bývá jen 4-cestná.
- Rozumíte pojmům?
  - Co je oběť
- Závěr: každé adjektivum **V**/**S**P vyjadřuje něco jiného…

Nejdůležitější periferií je disk. Tak ho musíme zrychlit…

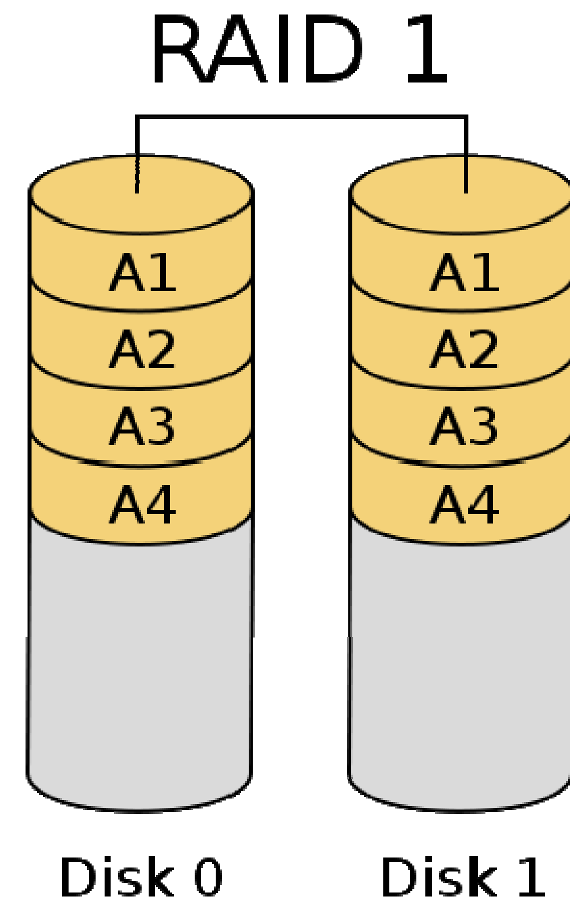
RAID 0

- Pro zvýšení výkonu systému pevných disků.
- tzv. “stripping” (proužkování)



RAID 1

- Pro zvýšení spolehlivosti uložených dat.
- Označuje se jako “Mirroring”.
- Nezrychluje, ale zvyšuje spolehlivost.

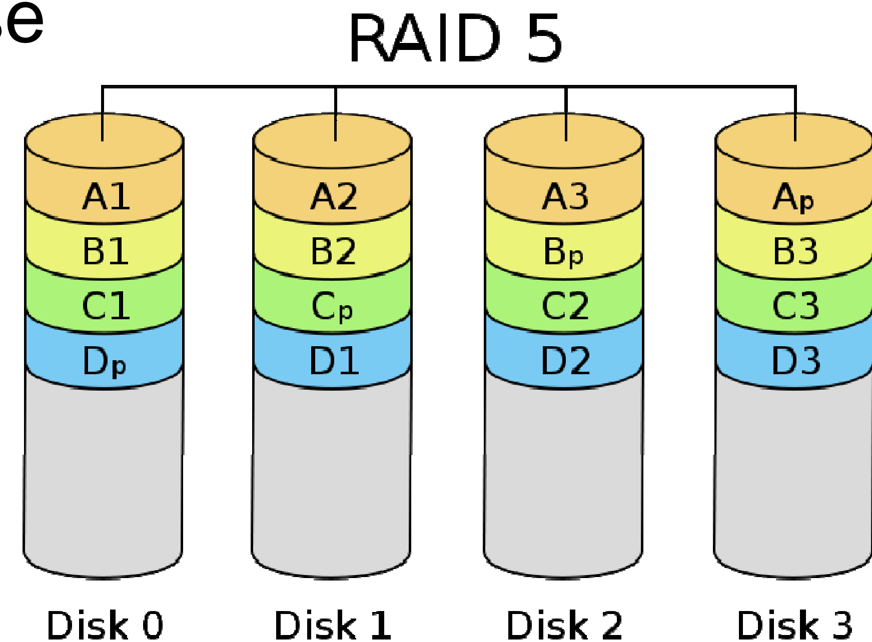


RAID 10

- Kombinace obou výše popsaných.
- Vytvoří se RAID 0 a ten se pak zrcadlí na RAID 1. Výsledkem jsou vlastně dva RAID 0 obsahující identická data.
- RAID 10 zvyšuje jak výkon, tak spolehlivost, musíte ovšem použít nejméně čtyři disky, nejlépe se stejnými parametry.

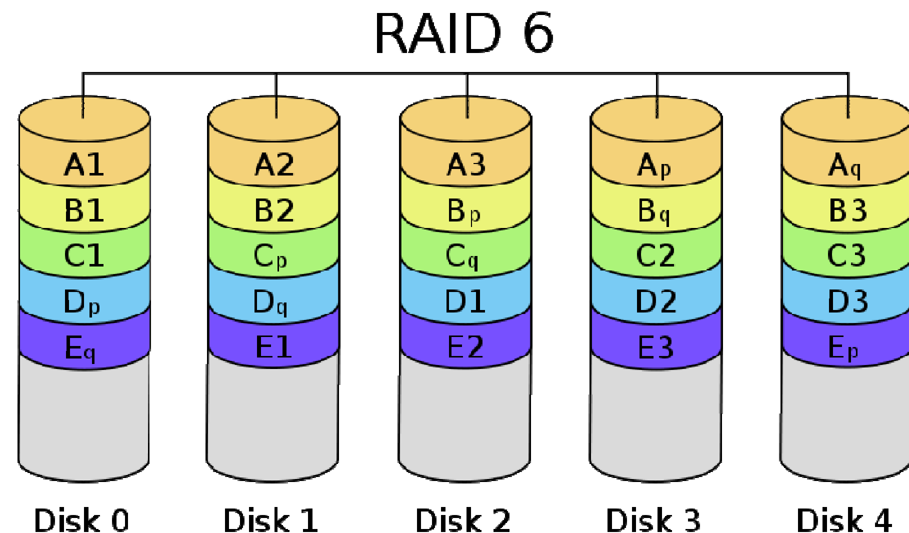
RAID 5

- Ukládá paritní informace, nikoli však na jeden vyhrazený disk.
- V degradovaném režimu se musejí data uložená na vadném disku odvodit z dat zbývajících disků a parity.
- Zrychluje čtení, zpomaluje zápis.



RAID 6

- Obdoba RAID 5, používá dva paritní disky s různě vypočtenou paritou.
- Odolný proti výpadkům 2 disků.
- Rychlost čtení jako RAID 5, zápis ještě pomalejší.



What Every Programmer Should Know About Memory

Přečtěte si:
http://www.akkadia.org/drepper/cpumemory.pdf