Principy počítačů a operačních systémů

Operační systémy
Správa paměti

Zimní semestr 2010/2011

Správa paměti

OS jako správce paměti

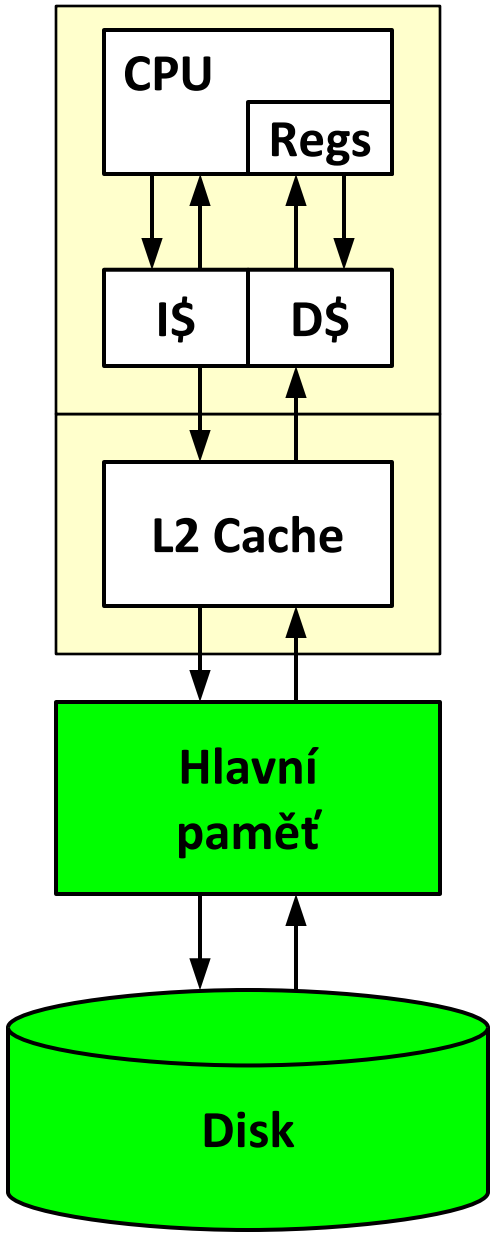
- specializovaný subsystém OS
- spravuje hlavní paměť systému
  - přidělování paměti procesům
  - informace o obsazenosti paměti
  - ochrana paměti mezi procesy
  - využití sekundární paměti pro zvýšení kapacity
- snaha uspokojit požadavky všech procesů
  - nesmí vést k zablokování – preempce

Virtuální paměť

Jak mohou aplikace sdílet paměť?

Aplikace si myslí, že systém neomezené množství paměti...

Virtuální paměť



Virtualizace paměti

- oddělení adresového prostoru fyzické paměti a adresového prostoru procesu
- mechanizmus pro převod adres z virtuálního adresového prostoru (procesu) do fyzického

Přesun dat mezi vrstvami řídí OS

- výpadky stránek, strategie pro výběr oběti

Hlavní výhody

- ochrana paměti (oddělené adr. prostory)
- jednoduchý model paměti (bez relokace)
- snížení nebezpečí uváznutí
  - možnost odebrat přidělenou paměť
- skrytí skutečné velikosti paměti (větší/menší)

Virtualizace paměti stránkováním

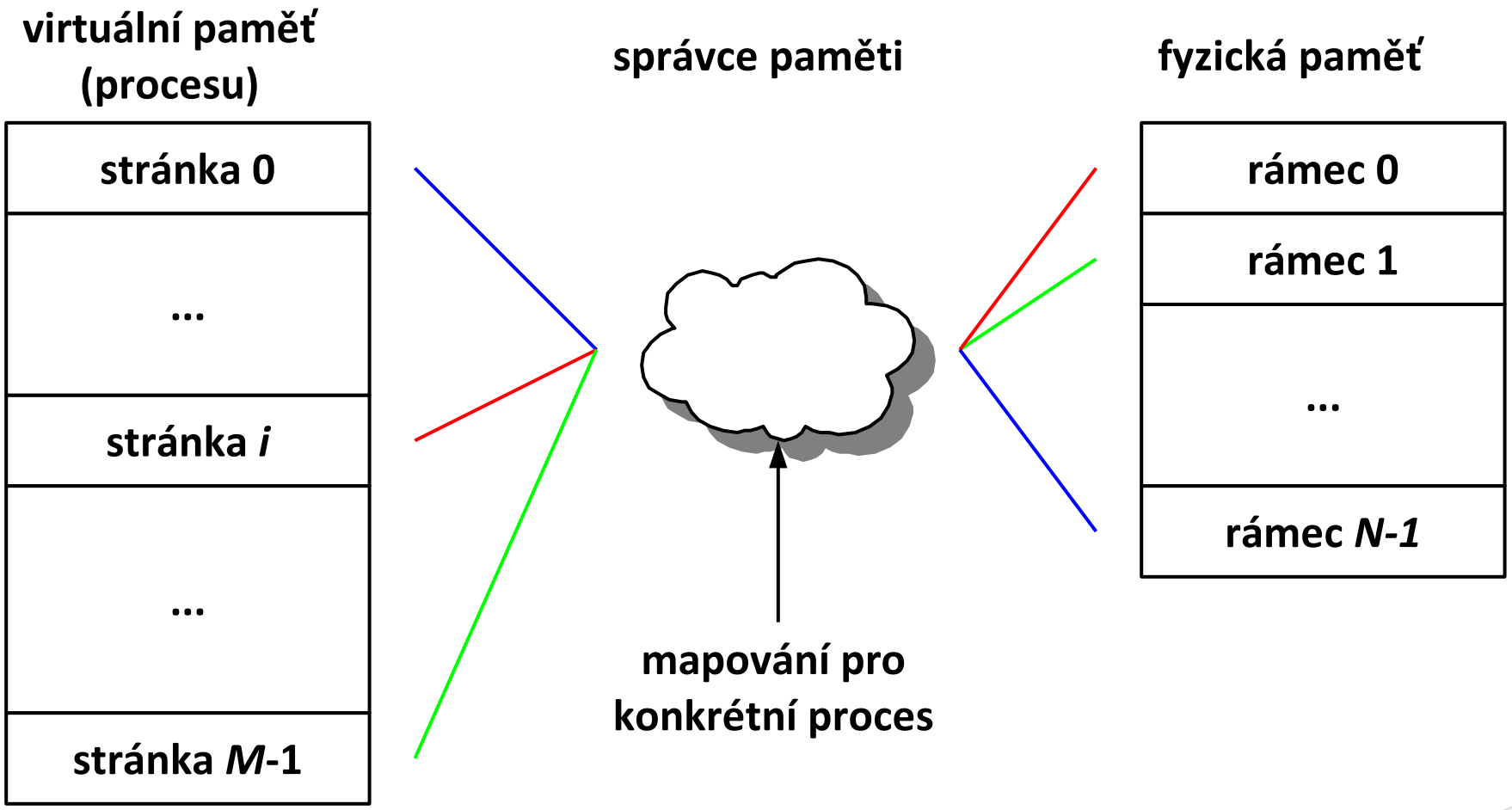
Princip

- VAP i FAP rozdělen na bloky stejné velikosti
  - bloky VAP se nazývají stránky (pages)
  - bloky FAP se nazývají rámace (frames)
- stránky VAP se mapují na rámce FAP
  - obecně nelineární, lineární pouze v rámci bloku
- VA se za běhu překládají na FA

Paměťový model

- 1 virtuální adresový prostor $[0, 2^{\text{počet bitů adresy}})$
- logické části programu rozmístěny ve VAP

Stránkování



Stránkování

Překlad VA na FA

- kód programu používá virtuální adresy
- při přístupu se VA rozloží na složky (page, offset)
- mechanismus překladu adres převede číslo stránky (page) na odpovídající číslo rámce (frame)
- pokud je mapování page → frame definováno, výsledkem je FA tvořená dvojicí (frame, offset)
  - tj. překládá se page → frame, offset se přenáší
- jinak dojde k výpadku stránky (page fault)

Stránkování

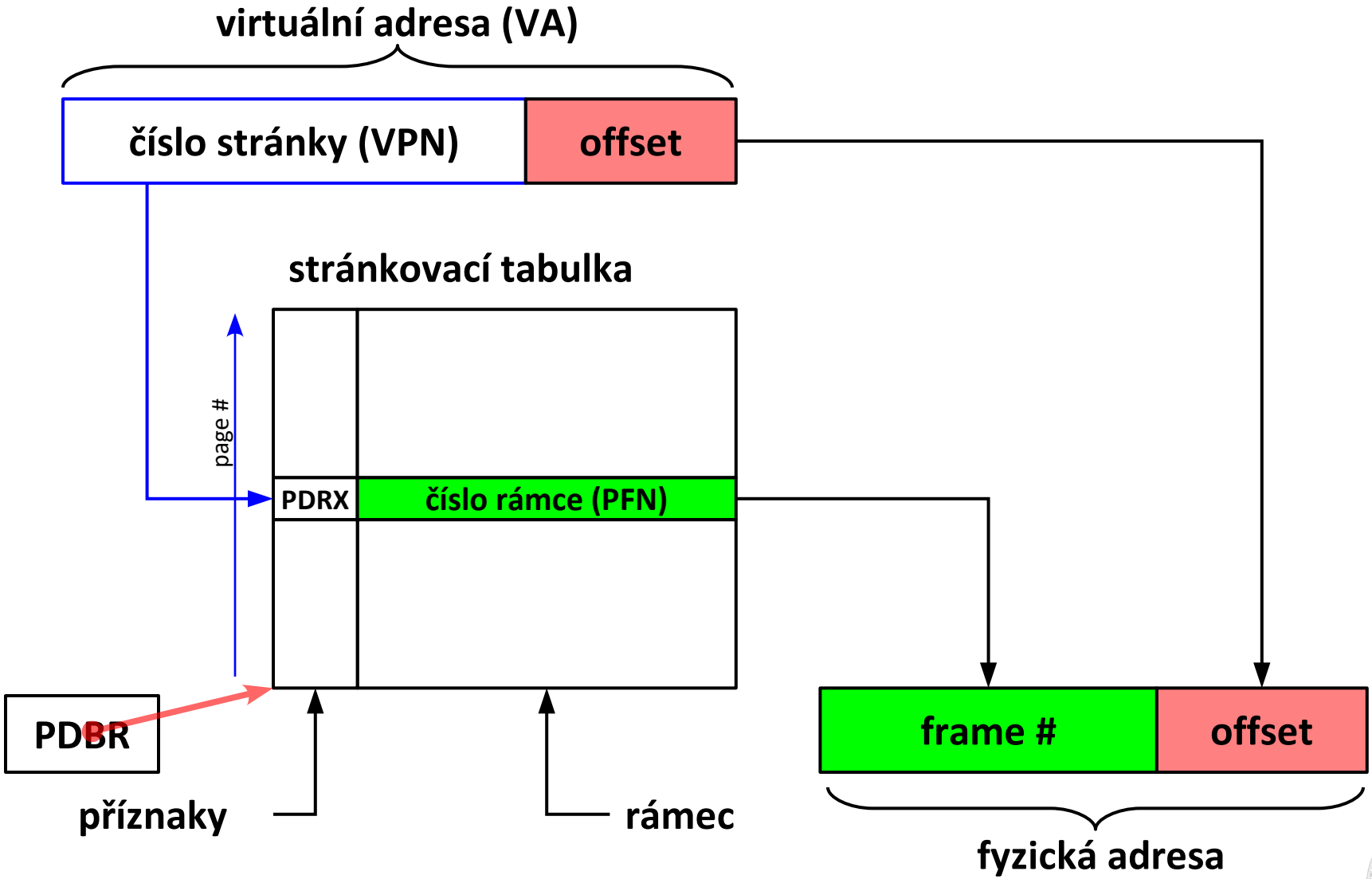
Překlad adres

- mapování definuje operační systém
  - každý proces má vlastní mapu virtuální paměti
- mechanizmus překladu definuje procesor
  - HW může vyžadovat specifické datové struktury

Stránkovací tabulky

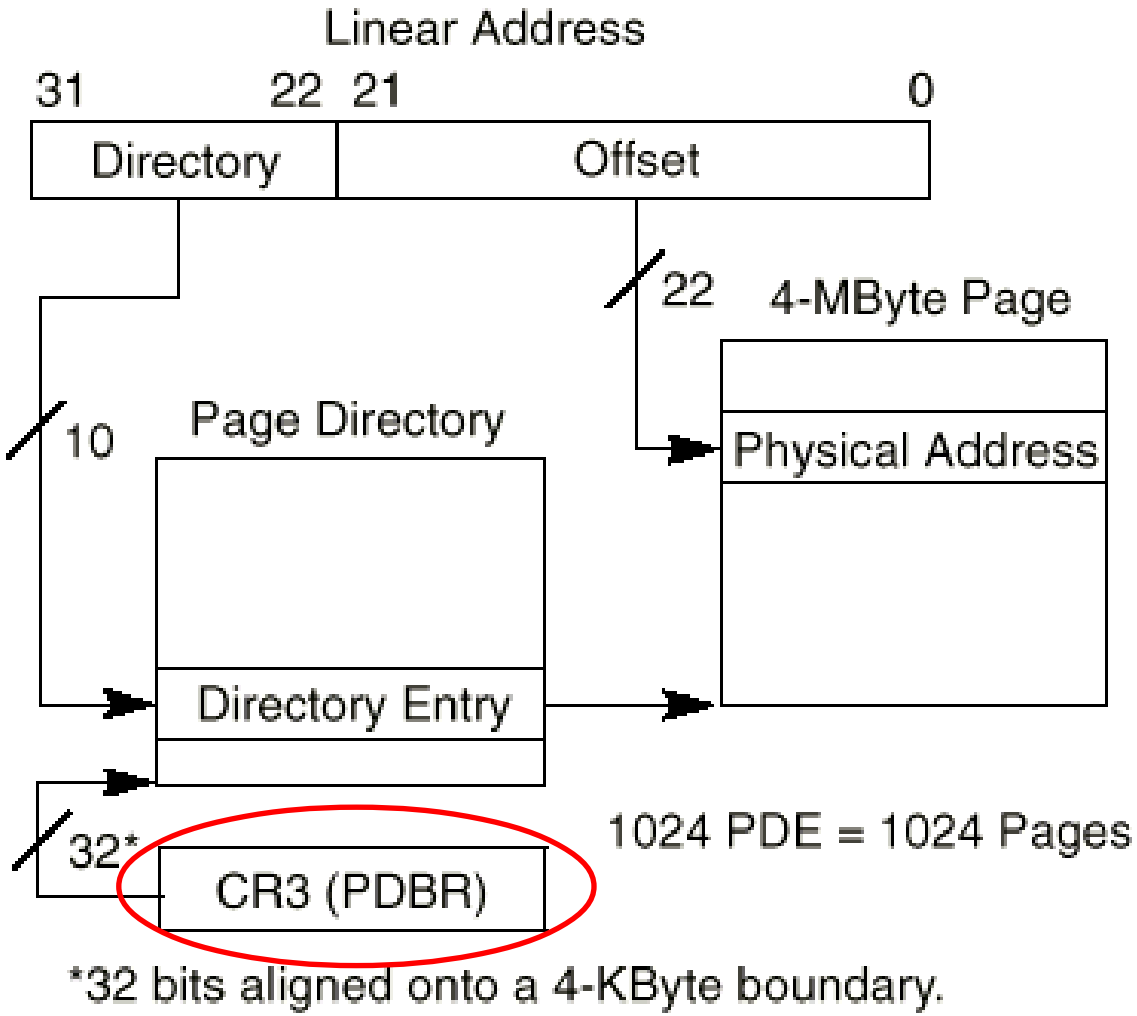
- položka (page table entry) pro každou stránku VA
  - adresa rámce fyzické paměti pro danou stránku
  - příznaky (minimálně platnost mapování)

Překlad adresy pomocí stránkovací tabulky



Příklad: jednoúrovňové stránkování (Intel IA32)

4MiB stránky, 1-úrovňové stránkování



Problémy stránkování

Vnitřní (interní) fragmentace

- v přidělovaných blocích zůstává nevyužité místo
- průměrná režie na souvislý blok ~ 50% velikosti stránky

Velikost stránkovacích tabulek

- 32b adresy → 4GiB paměti
- 4KiB stránky → 1Mi položek stránkovací tabulky
- 4B na položku tabulky → 4MiB paměti na proces

Rychlost přístupu do stránkovacích tabulek

- překlad adresy při každém přístupu do paměti
  - včetně čtení instrukcí programu
- instrukce s paměťovými operandy...

Výběr velikosti stránky

Malé stránky

– malá lokalita referencí (typ. < 256)

\+ menší fragmentace

– velké stránkovací tabulky

Velké stránky

\+ malé stránkovací tabulky

\+ lépe vyhovuje I/O

– vetší fragmentace

Jaké velikosti dávají smysl?

Víceúrovňové stránkovací tabulky

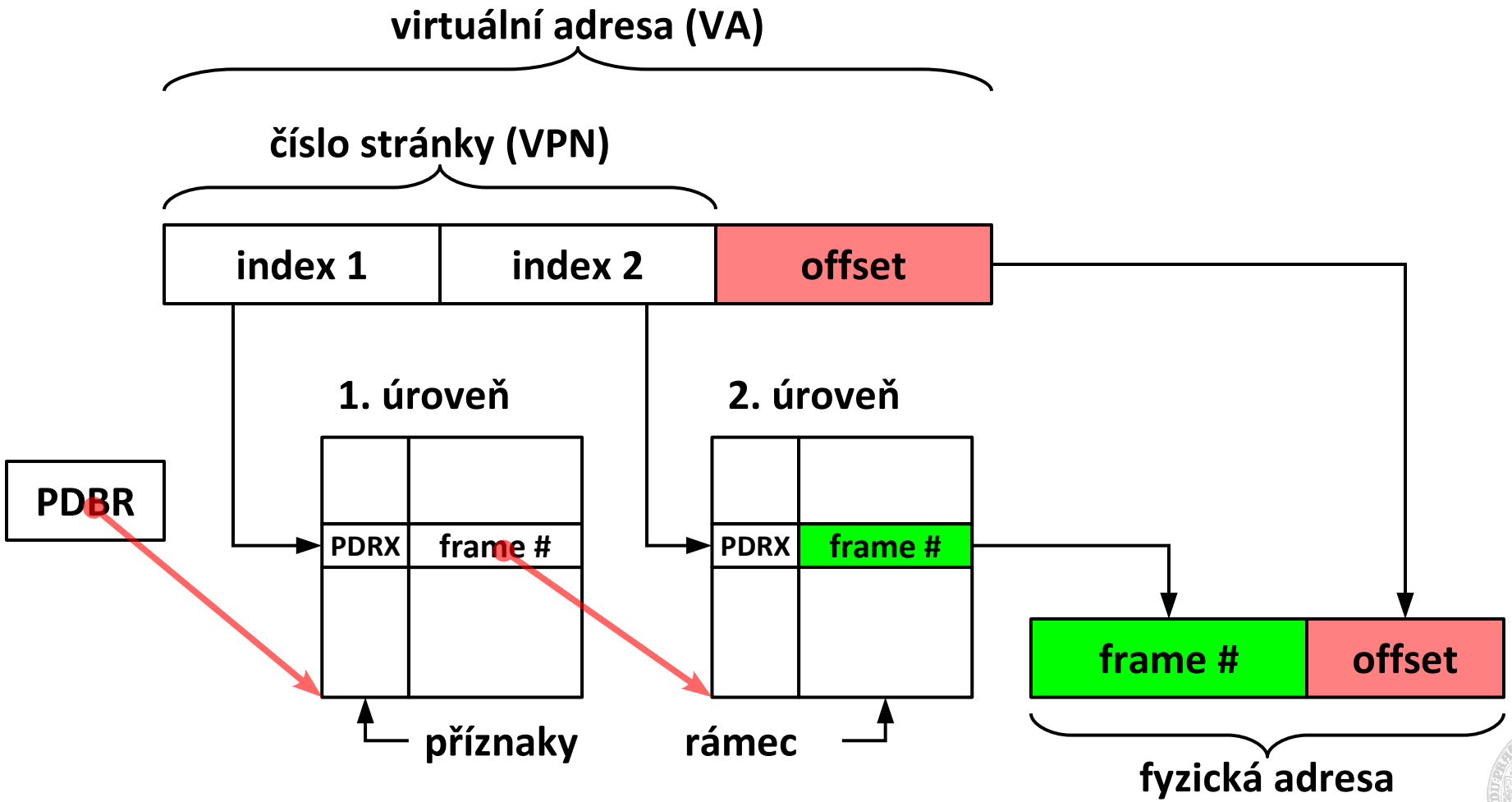
Hierarchická struktura tabulek

- položka odkazuje na rámec s tabulkou další úrovně
- položky listových tabulek odkazují na rámec pro stránku
- V praxi 2-3 úrovně

Řeší problém velikosti tabulek, ale...

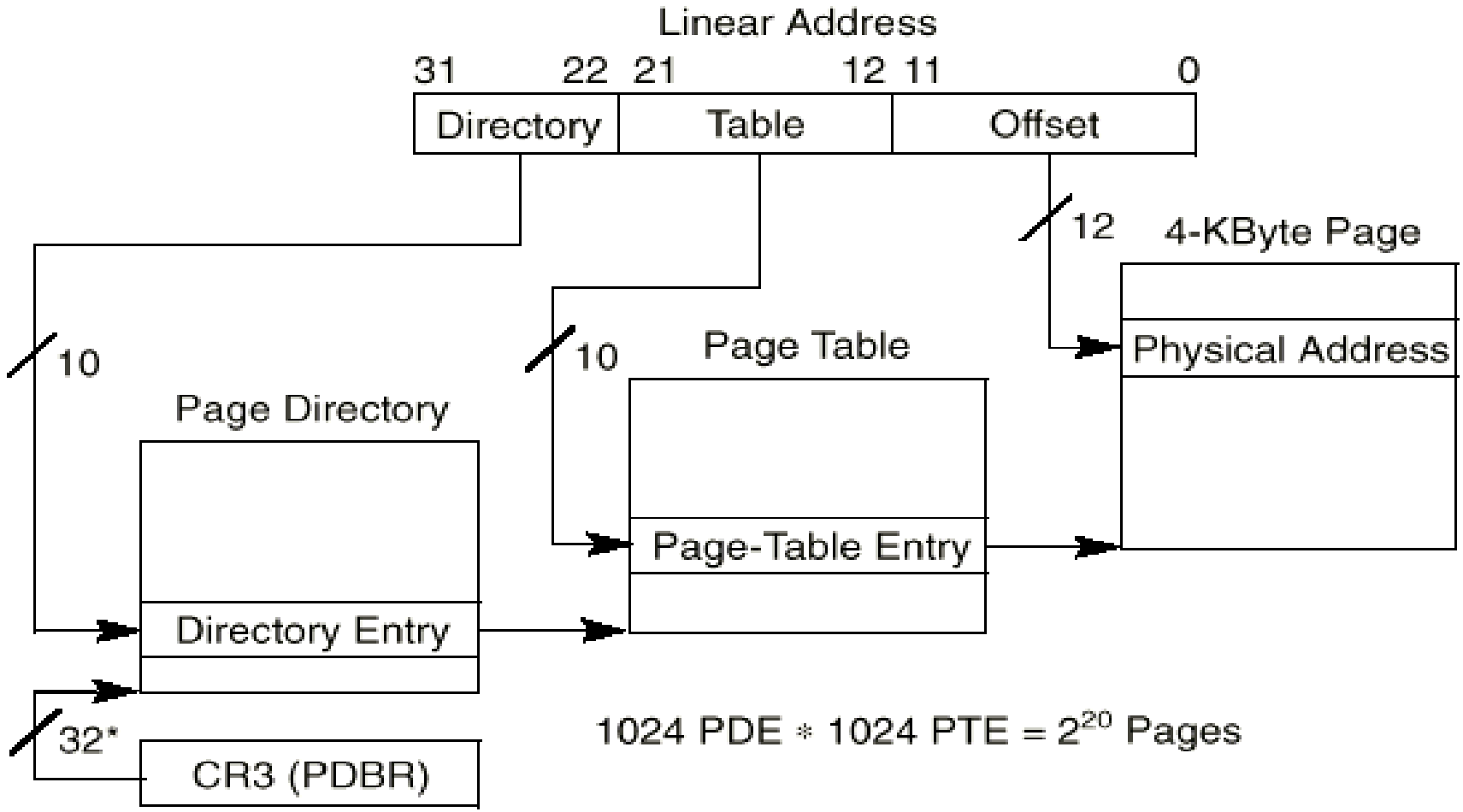
- jak je to s rychlostí překladu při přístupu do paměti?
- co když VAP opravdu velký (např. 56bit)

Překlad pomocí víceúrovňových tabulek



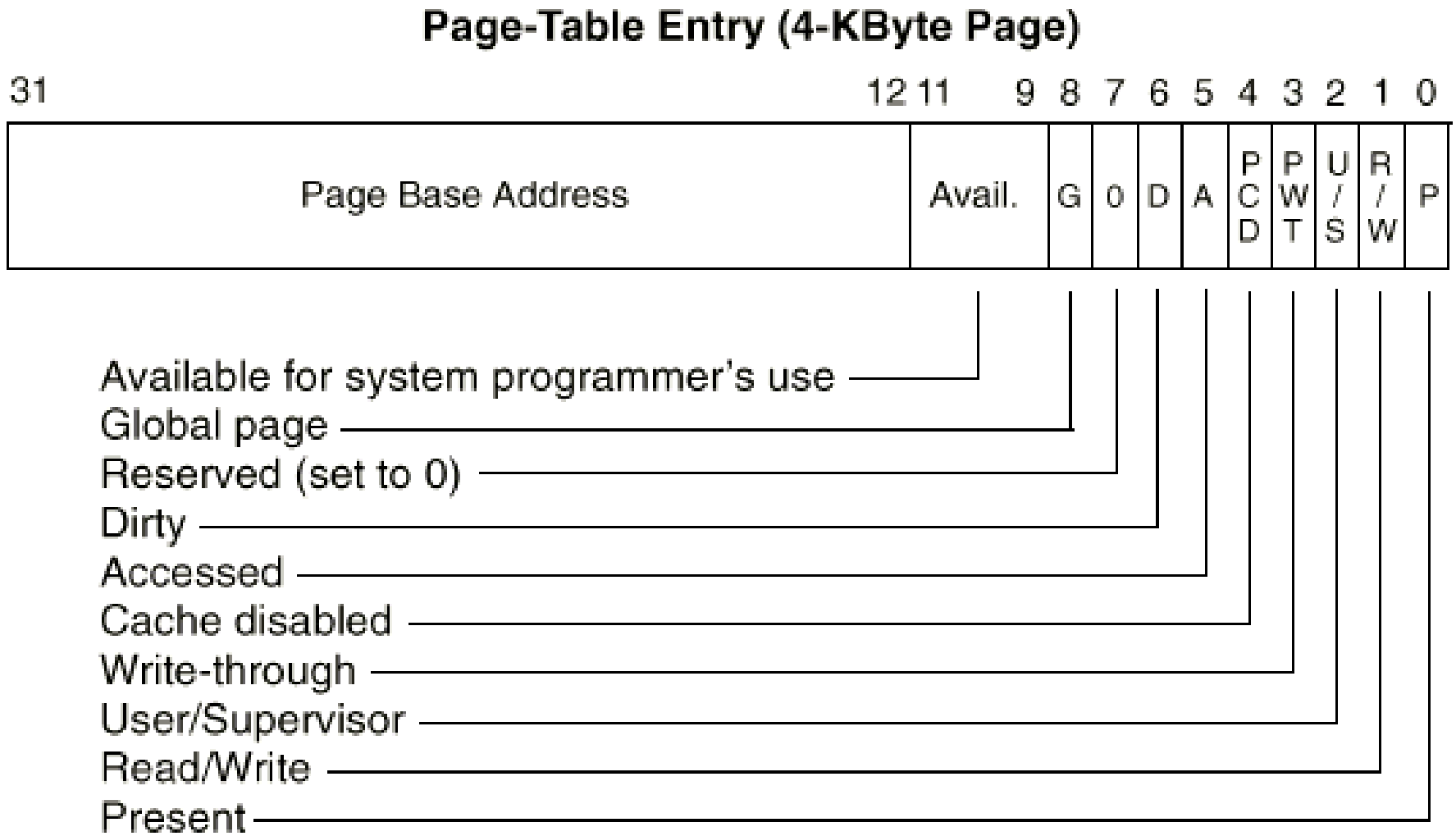
Příklad: víceúrovňové stránkování (Intel IA32)

4KiB stránky, 2-úrovňové stránkování



Příklad: položka stránkovací tabulky (Intel IA32)

Položka stránkovací tabulky



Jak rozdělit číslo stránky na indexy?

Konfigurace

- počet bitů VA, počet bitů FA, velikost stránky

Cíl: maximální využití rámců

- každý rámec obsahuje část stránkovací tabulky
- velikost stránky ⇒ počet bitů na offset a VPN
- počet bitů na VPN ⇒ velikost položky (PTE)
  - počet bitů FA ⇒ počet bitů na číslo rámce
  - spolu s bity na příznaky zarovnáno na délku slova CPU
- velikost položky ⇒ počet položek ve stránce
- počet položek ve stránce ⇒ počet bitů na index
  - indexy stejné, index pro 1. úroveň může být menší

Příklad: 32-bitů VA, 32-bitů FA, stránka 4KiB

Rozdělení VPN na části

- 4KiB stránka ⇒ 12 bitů offset, 20 bitů číslo stránky
- položka stránkovací tabulky – 32 bitů
  - 32-bitů FA ⇒ 20 bitů číslo rámce + 3-9 bitů příznaky
  - zarovnáno na nativní délku slova procesoru (4/8 bajtů)
- 4B na položku ⇒ 1024 položek tabulce (v jednom rámci)
- 1024 položek v tabulce ⇒ 10 bitů na index do tabulky
- 20 bitů číslo stránky ⇒ 2-úrovňové stránkování
  - 10 bitů pro ***index 2***, 10 bitů pro ***index 1***

Příklad: 32-bitů VA, 32-bitů FA, stránka 1KiB

Rozdělení VPN na části

- 1KiB stránka ⇒ 10 bitů offset, 22 bitů číslo stránky
- položka stránkovací tabulky – 32 bitů
  - 32-bitů FA ⇒ 22 bitů číslo rámce + 3-9 bitů příznaky
  - zarovnáno na nativní délku slova procesoru
- 4B na položku ⇒ 256 položek na tabulku v rámci
- 256 položek v tabulce ⇒ 8 bitů na index
- 22 bitů číslo stránky ⇒ 3-úrovňové stránkování
  - 8 bitů index3, 8 bitů index2, 6 bitů index1
  - rámec s tabulkou první úrovně (kořen) není plně využit

Výhody stránkování

Mapování VA na FA nemusí být lineární

- stránky „rozházeny“ po hlavní paměti

Obsah některých stránek nemusí být v paměti

- uloženy v sekundární paměti (odkládání)

Pro nevyužité části VAP mapování neexistuje

- umožňuje detekovat některé chyby v programu

Zcela transparentní pro uživatele/proces

- proces si “myslí”, že má celou paměť jen pro sebe

Paměťový kontext procesů je plně izolován

- *v případě potřeby je možné paměť jednoduše sdílet, jak?*

Urychlení překladu při přístupu do paměti

TLB – Translation Lookaside Buffer

- cache položky stránkovacích tabulek
  - klíčem je číslo stránky, kapacita ~ desítky položek
- využití časové a prostorové lokality programu
  - kód se nějakou dobu načítá ze stejné stránky
  - s daty se nějakou dobu pracuje v rámci jedné stránky

0-úrovňové stránkování

- procesor hledá mapování pouze v TLB, zbytek řeší OS
- procesor poskytuje instrukce pro ovládání TLB
- MIPS, UltraSPARC III

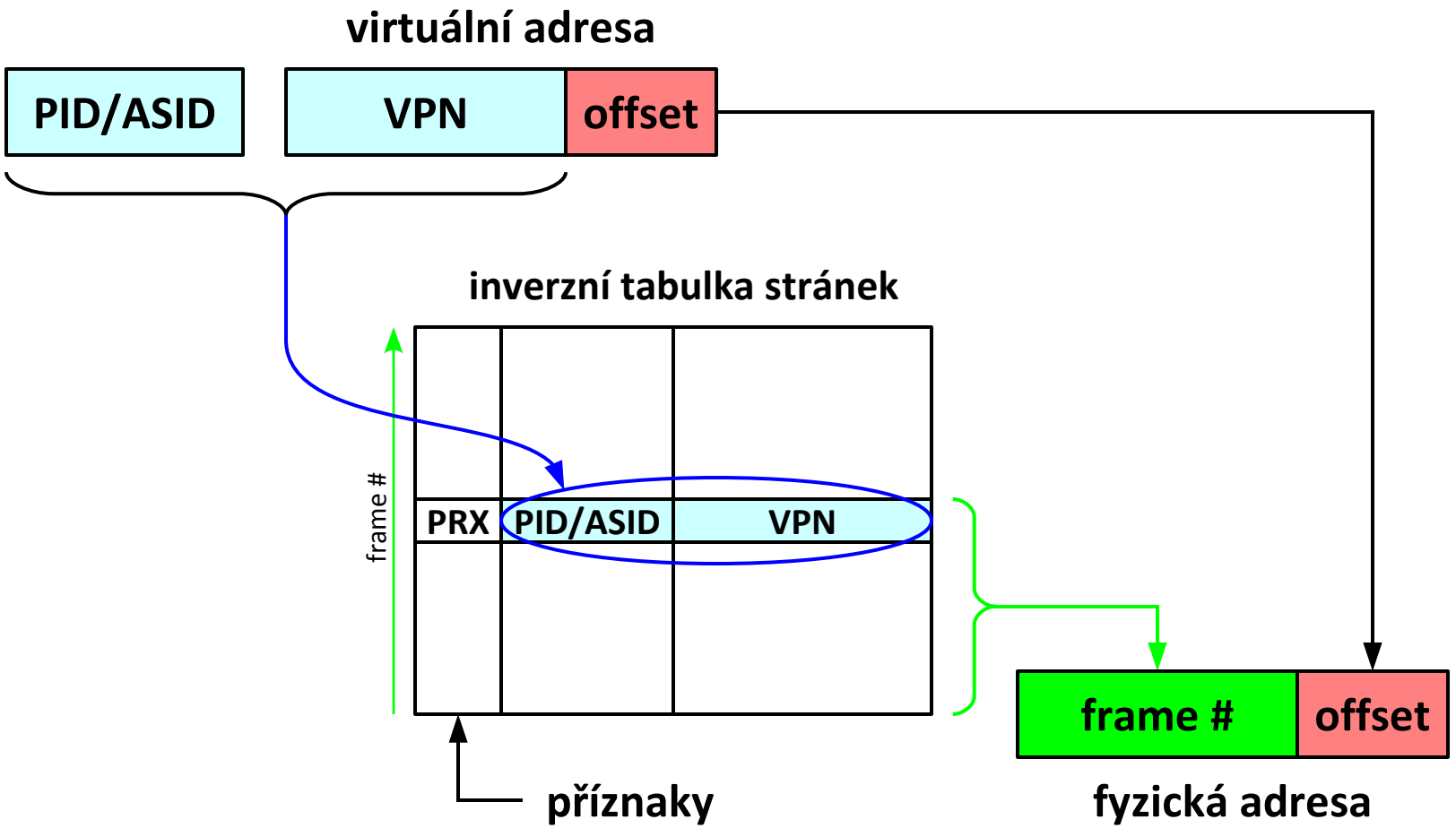
Jak je to s TLB při běhu více procesů?

Stránkování pro rozsáhlé adresové prostory

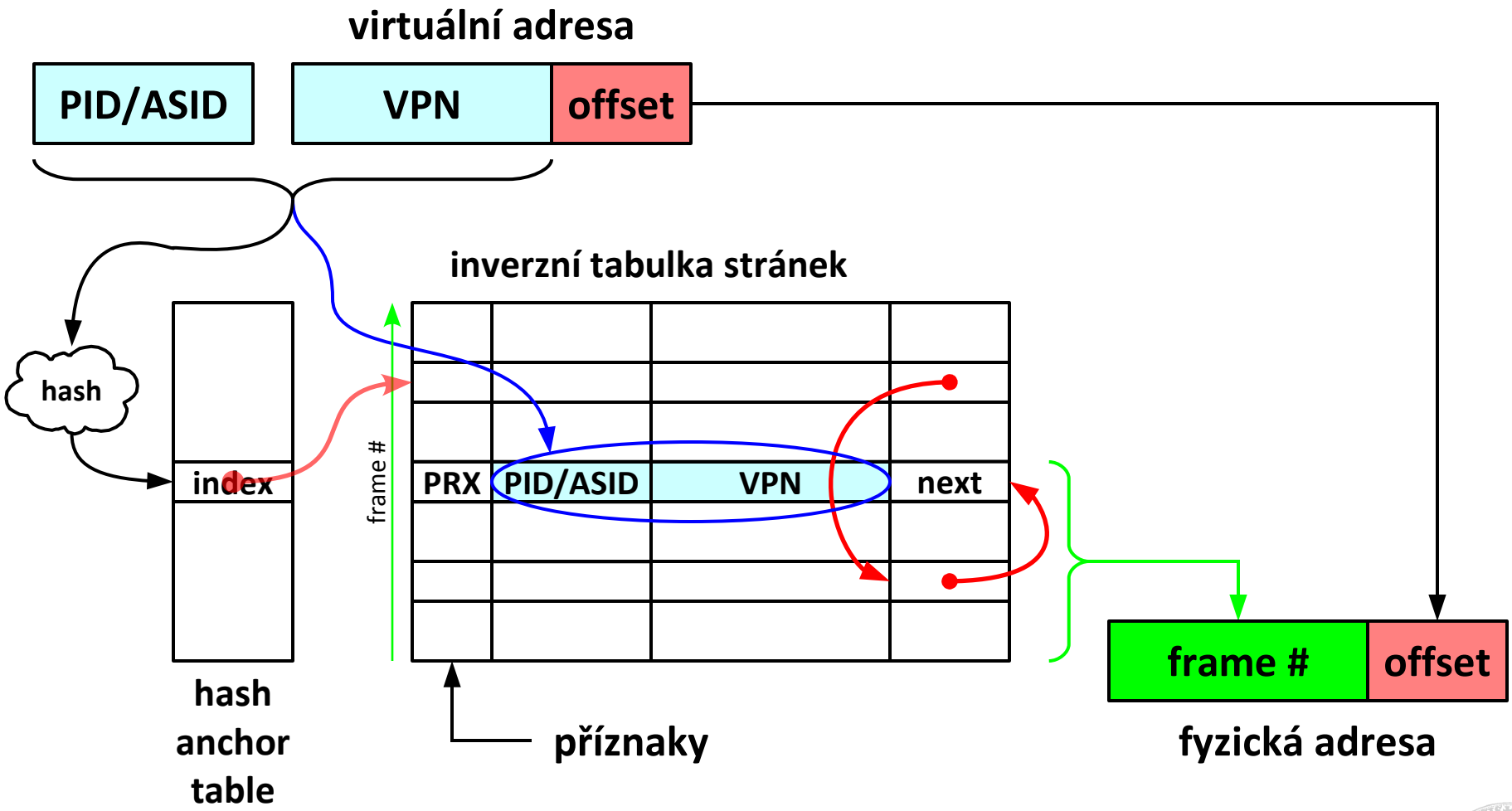
Inverzní stránkovací tabulka (IPT)

- velký VAP zjednodušuje programovací model
  - FAP je typicky menší než VAP
- datová struktura nad rámci, nikoliv stránkami
  - globální pro všechny procesy
- organizace inverzní stránkovací tabulky
  - záznam pro každý rámec, pořadí určuje číslo rámce
  - (ID procesu/adresového prostoru, číslo stránky, ?)
- nalezení položky v IPT
  - sekvenční prohledání, hashování (OS, CPU)

Překlad pomocí inverzní stránkovací tabulky



Překlad pomocí inverzní tabulky a hashování



Stránkování pro rozsáhlé adresové prostory

Nevýhoda oproti klasickým tabulkám

- OS musí mít dodatečné struktury např. pro informaci o tom, kde na disku se nachází rámec

Jak je to se sdílením paměti mezi procesy?

Výpadek stránky a výběr oběti

Výpadek stránky (page fault)

Požadovaná stránka není namapována

- výjimka při provádění programu
- mapování vůbec neexistuje – přístup na špatnou adresu
- mapování existuje, ale obsah stránky není v paměti
  - nutno načíst z disku do nějakého volného rámce
  - namapovat stránku na přidělený rámec (stránkovací tabulka)
- pokud nejsou volné rámce, musí se nějaký uvolnit

Kterou stránku vyhodit z paměti?

- aby se uvolnil rámec, ve kterém byla namapována...

Algoritmus výběru oběti (page replacement policy)

Výběr stránky z množiny potenciálních obětí

- obsah vybrané stránky je zapsán na disk
- původní rámec je použit pro jinou stránku
- stejný princip aplikovatelný i na řádky cache

Working set

- množina "právě používaných" stránek
  - odráží lokalitu při vykonávání programu
- důležité je nevyhazovat stránky z working setů

First In First Out (FIFO)

Vyhodit nejdéle namapovanou stránku

- nejjednodušší (a nejhloupější) strategie
  - seznam namapovaných stránek, oběť je na začátku seznamu
  - nově namapovaná stránka vložena na konec seznamu
- ***ignoruje chování programu***, anomální chování
  - Beladyova anomálie – zvýšení počtu rámců může vést ke zvýšení počtu výpadků
  - množina stránek udržovaná v N rámcích **není** nadmnožinou stránek udržované v N-1 rámcích

FIFO – Beladyova anomálie

Zvýšení počtu rámců zvýší počet výpadků

- jako kdyby zvýšení kapacity cache zvýšilo $\%_{miss}$ cache

Page Refs	3 Page Frames			
	Fault?	Page Contents		
A	yes	A		
B	yes	B	A	
C	yes	C	B	A
D	yes	D	C	B
A	yes	A	D	C
B	yes	B	A	D
E	yes	E	B	A
A	no	E	B	A
B	no	E	B	A
C	yes	C	E	B
D	yes	D	C	E
E	no	D	C	E

Page Refs	4 Page Frames				
	Fault?	Page Contents			
A	yes	A			
B	yes	B	A		
C	yes	C	B	A	
D	yes	D	C	B	A
A	no	D	C	B	A
B	no	D	C	B	A
E	yes	E	D	C	B
A	yes	A	E	D	C
B	yes	B	A	E	D
C	yes	C	B	A	E
D	yes	D	C	B	A
E	yes	E	D	C	B

- pozor, nové stránky na začátku a oběti na konci

Optimální strategie (MIN, OPT)

Vyhodit stránku, která nebude nejdéle potřeba

- v principu jde o to mít v paměti vždy celý WS
- za oběť bude vybrána stránka, na kterou přistoupíme za nejdelší dobu
  - least soon needed
- nelze implementovat on-line, lze simulovat

Least Recently Used (LRU)

Vyhodit nejdéle nepoužitou stránku

- dlouho nepoužívané stránky nebudou potřeba
- software implementace
  - seznam stránek, oběť na začátku seznamu
  - při přístupu do stránky ji přesuneme na konec seznamu
  - rychlý výběr oběti, pomalý přístup do paměti
- hardware implementace
  - při přístupu do rámce zapsáno časové razítko
  - při výběru oběti nutno seřadit podle času přístupu
  - pomalý výběr oběti, rychlý přístup do paměti

Problémy LRU

Nevhodné pro stránkování

- vysoké nároky na údržbu datových struktur
  - velký počet rámců, aktualizace při každém přístupu
- realizace v HW obtížná/drahá
  - 64b čítač, který CPU při přístupu uloží do PTE
  - matice n×n, při přístupu do rámce k nastavíme k-tý řádek na 1 a k-tý sloupec na 0. Vybírá se řádek s nejméně 1.
- vhodné pro malý počet/větší granularitu
  - řádky cache v asociativní množině, objekty v HTTP cache
- nerozlišuje četnost přístupů
  - scan: jednorázový sekvenční přístup k bloku paměti

Second Chance

Modifikace FIFO, zohledňuje chování programu

- pokud má oběť nastavený Access bit,
  dostane druhou šanci
  - seznam stránek, potenciální oběť na začátku seznamu
  - pokud má stránka odpovídající oběti nastavený access bit, je vynulován a stránka přesunuta na konec seznamu
  - nově namapovaná stránka zařazena na konec seznamu
- pokud je stránka často používaná, stihne si "opatřit" dříve, než na ni znovu přijde řada

1-Handed Clock

Efektivnější implementace Second Chance

- odstraňuje nutnost upravovat seznam
  - netřeba zamykat
- mapované stránky zařazeny v kruhovém seznamu
  - ukazatel na nejstarší stránku (tj. další kandidát)
- při hledání oběti po výpadku stránky OS u aktuální stránky zkontroluje Access bit
  - pokud Accessed = 0, oběť nalezena
  - pokud Accessed = 1, vynuluji a pokračuji v hledání
  - ukazatel se vždy posune na následující stránku

2-Handed Clock

Flexibilnější varianta 1-Handed Clock

- umožňuje regulovat dobu, po kterou má stránka "druhou šanci", tj. "opatřit" si Access bit
- kruhový seznam stránek, 2 ukazatele
  - ukazatel na nejstarší stránku, posouvá se dokud nenajde stránku s Accessed = 0
  - ukazatel posunutý o konstatní počet stránek před nejstarší, posouvá se spolu s prvním ukazatelem, nuluje Accessed bity "navštívených" stránek
- vzdálenost mezi ukazateli možno měnit v závislosti na zatížení systému

Not Recently Used (NRU)

Snaha nevyhazovat nedávno použité stránky

- stránky mají příznaky Accessed a Dirty
  - příznaky nastavuje CPU v položkách právě aktivních stránkovacích tabulek při přístupu do paměti
  - OS periodicky nuluje příznak Accessed
- při výpadku náhodný výběr stránky ze tříd
  - not accessed, not dirty
  - not accessed, dirty
  - accessed, not dirty
  - accessed, dirty

Co když procesor nepodporuje Accessed a Dirty bity?

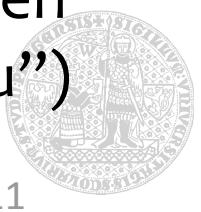
Not Frequently Used (NFU)

Snaha nevyhazovat často používané stránky

- OS udržuje ke každé stránce čítač přístupů
  - k čítači OS periodicky přičítá hodnotu Accessed bitu stránky
- obětí se stane stránka s nejnižší hodnotou čítače
- rychle se "zbavuje" stránek s malou frekvencí přístupů
  - "scan" přístup u LRU vs. NFU

Problémy NFU

- algoritmus nezapomíná
  - často používaná a pak zapomenutá stránka přežívá
- úprava – exponenciální stárnutí (aging)
  - čítač přístupů je posunut vpravo, nejvyšší bit čítače nastaven na hodnotu Accessed bitu (nižší bity automaticky "stárnou")



Problémy při stránkování

Thrashing

- working set běžících procesů překročil počet rámců, které má systém k dispozici
  - není možné držet všechny working sets v paměti
- při výpadku stránky se nutně vyhodí stránka z working setu nějakého procesu
- z definice však k této stránkce bude brzy přistupováno, což opět povede k výpadku stránky
- jediné co bude OS dělat je načítat/zapisovat stránky

Přidělování paměti

Přidělování paměti

Obecný princip – správa adresového prostoru

- evidence volných/obsazených bloků
- nalezení/přidělení bloku požadované velikosti
  - souvislý blok požadované (nebo větší) velikosti
- stejné např. pro diskový prostor (přidělují se sektory)

Datové struktury používané k evidenci

- bloky konstatní velikosti – statické datové struktury
  - konstatní počet alokovatelných bloků
  - bitová mapa obsazenosti
- bloky proměnné velikosti – dynamické datové struktury
  - proměnný počet alokovatelných bloků
  - seznamy, intervalové stromy

Fragmentace paměti

Interní

- přidělený blok paměti není plně využit
- v průměru nevyužito 50% alokační jednotky

Externí

- volné místo tvoří pouze malé bloky
- znemožňuje alokaci souvislých bloků
- možno řešit setřesením alokovaných bloků
  - vyžaduje relokaci programu za běhu

Základní alokační strategie

Best Fit

- vždy je nutné prohledat všechna volná místa
- vytváří velké množství malých děr

Worst Fit

- na rozdíl od Best Fit nevytváří malé díry

First Fit

- menší fragmentace ve srovnání s BF, ale malé díry se shlukují na začátku

Next Fit

- hledání začíná od naposledy obsazeného bloku
- malé díry rovnoměrně rozprostřeny