Operační systémy a databáze

Petr Štěpán, K13133

KN-E-129

stepan@fel.cvut.cz

Téma 5. Správa paměti

Základní fakta

- FAP – fyzická adresa je adresa vnitřní paměti počítače
  - Rozsah FAP je dán architekturou počítače
    - kolik „drátů“ má adresní sekce sběrnice
  - Velikost vnitřní paměti bývá i menší než je rozsah FAP
    - záleží na tom, kolik peněz jsme ochotni za paměť dát a kolik paměti se fyzicky do počítače vejde, případně na paměťových řadičích apod.
- LAP – logická adresa je adresa hypotetické paměti
  - Její obsah je buď uložen ve fyzické paměti nebo na vnější paměti (případně nebyla ještě použita a proto ani neexistuje)
  - Rozsah LAP je dán architekturou CPU
    - dán „šířkou“ dat, s nimiž je CPU schopen pracovat
- Program se přetváří do formy schopné běhu v řadě kroků
  - V jistém okamžiku se musí rozhodnout **kde** bude kód a **kde** budou data ve vnitřní paměti
  - Cíl: **Vazba adres** instrukcí a dat na skutečné adresy v operační paměti
- Správa paměti je nutně předmětem činnosti OS
  - Aplikační procesy nemají přístup k prostředkům pro řízení paměti
    - Privilegované akce
  - Nelze ji svěřit na aplikačnímu programování
    - Bylo by to velmi neefektivní a nebezpečné

Požadavky na správu paměti

- Potřeba relokace programů
  - Procesu může být dynamicky přidělována jiná (aspoň zdánlivě) souvislá oblast paměti – relokace
    - Odkazy na paměť uvedené v programu (v LAP) se musí dynamicky překládat na skutečné adresy ve FAP
- Potřeba ochrany paměti
  - Procesy nesmí odkazovat na paměťová místa přidělená jiným procesům nebo OS. Požadavek na relokaci způsobí, že adresy nemohou přiřazovány během překladu či sestavování (➔). Odkazy do paměti se musí kontrolovat za běhu hardwarem (softwarové řešení je příliš pomalé)
- Programy jsou sady sekcí s různými vlastnostmi
  - Sekce s instrukcemi jsou „*execute-only*“
  - Datové sekce jsou „*read-only*“ nebo „*read/write*“
  - Některé sekce jsou „soukromé“ (*private*), jiné jsou „veřejné” (*public*)
  - OS a HW musí podporovat práci se sekcemi tak, aby se dosáhlo požadované ochrany a sdílení
- Požadavky na sdílení
  - Více procesů může sdílet společné úseky (sdílené struktury), aniž by tím docházelo k narušení ochrany paměti
    - Sdílený přístup ke společné datové struktuře je efektivnější než udržování konzistence násobných kopií vlastněných jednotlivými procesy

Počítače bez správy paměti

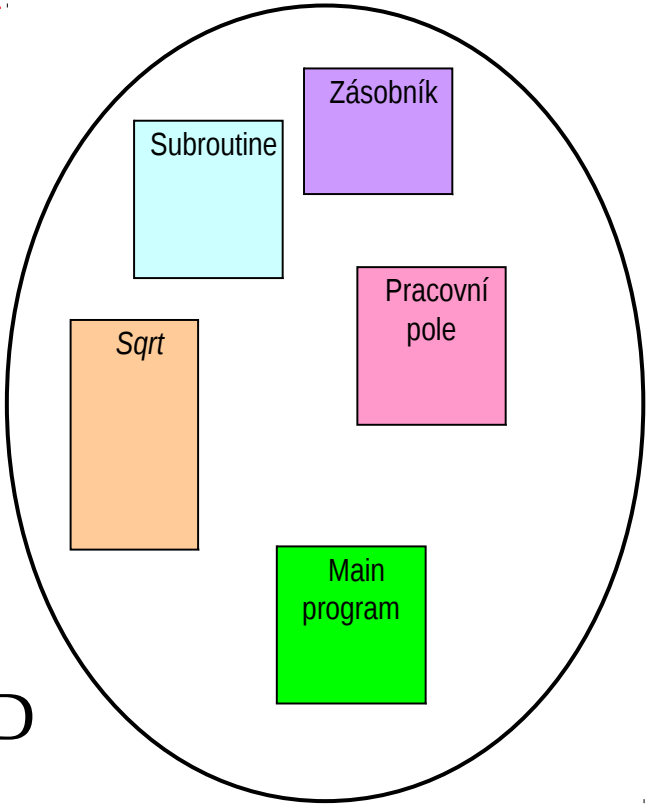
- Výhody systému bez správy paměti
  - Rychlost přístupu do paměti
  - Jednoduchost implementace
  - Lze používat i bez operačního systému – robustnost
- Nevýhody systému bez správy paměti
  - Nelze kontrolovat přístup do paměti
  - Omezení paměti vlastnostmi HW
- Použití
  - Historické počítače
  - Osmibitové počítače (procesory Intel 8080, Z80, apod.)
    - 8-mi bitová datová sběrnice, 16-ti bitová adresová sběrnice, možnost využít maximálně 64 kB paměti
  - Programovatelné mikrokontrolery
  - Řídicí počítače – embedded
    - v současné době již jen ty nejjednodušší řídicí počítače

Základní principy překladu LA → FA

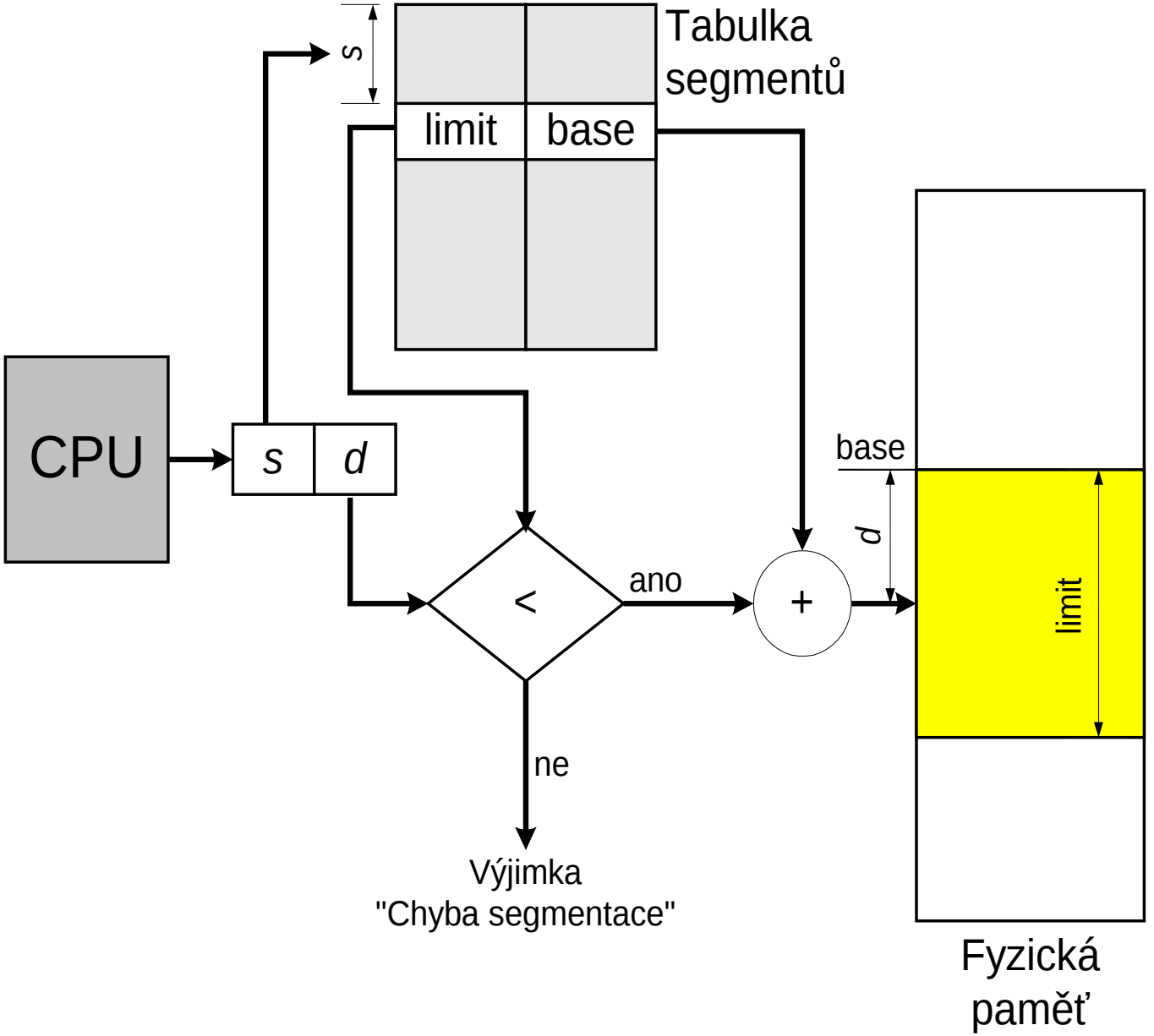
- Jednoduché segmenty – Intel 8086
  - Procesor 8086 má 16 bitů datovou sběrnici a navenek 20 bitů adresy. 20 bitů je ale problém. Co s tím?
  - Řešením jsou jednoduché segmenty
    - Procesor 8086 má 4 tzv. segmentové registry
    - Adresa je tvořena adresou segmentu 16-bitů a adresou uvnitř segmentu (offset) 16-bitů. Výsledná FA se tvoří podle pevného pravidla: (segment<<4)+ offset
      - Problém: Aplikace může měnit segmentové registry, není ochrana paměti
- Prostředek, jak používat větší paměť než dovoluje systém
  - Někdy se hovoří o mapování
    - malý logický adresní prostor se promítá do většího FAP
  - Příkladem je v současnosti využití PAE zvětšení rozsahu adres z 32 bitů na 36 bitů
- Opačná situace nastává, když LAP je větší než FAP
  - 32-bitová CPU generuje 32-bitové logické adresy
  - 32 bity adresovaný FAP je 4 GiB
    - Obvykle je v počítači méně
  - Zde nastupuje potřeba virtualizace ➔
    - Využívá rozšíření FAP o úseky na vnější paměti

Segmentace – obecný princip

- Podpora uživatelského pohledu na LAP
  - Program je kolekce segmentů
  - Každý segment má svůj logický význam: hlavní program, procedura, funkce, objekt a jeho metoda, proměnné, pole, ...
- Základní úkol – převést adresu typu (segment, offset) na adresu FAP
- Tabulka segmentů – Segment table, ST
  - Je uložena v paměti
  - Zobrazení 2-D (segment, offset) LAP do 1-D (adresa) FAP
  - Položka ST:
    - base – počáteční adresa umístění segmentu ve FAP, limit – délka segmentu
  - Segment-table base register (STBR) – umístění ST v paměti
  - Segment-table length register (STLR) – počet segmentů procesu



Hardwarová podpora segmentace



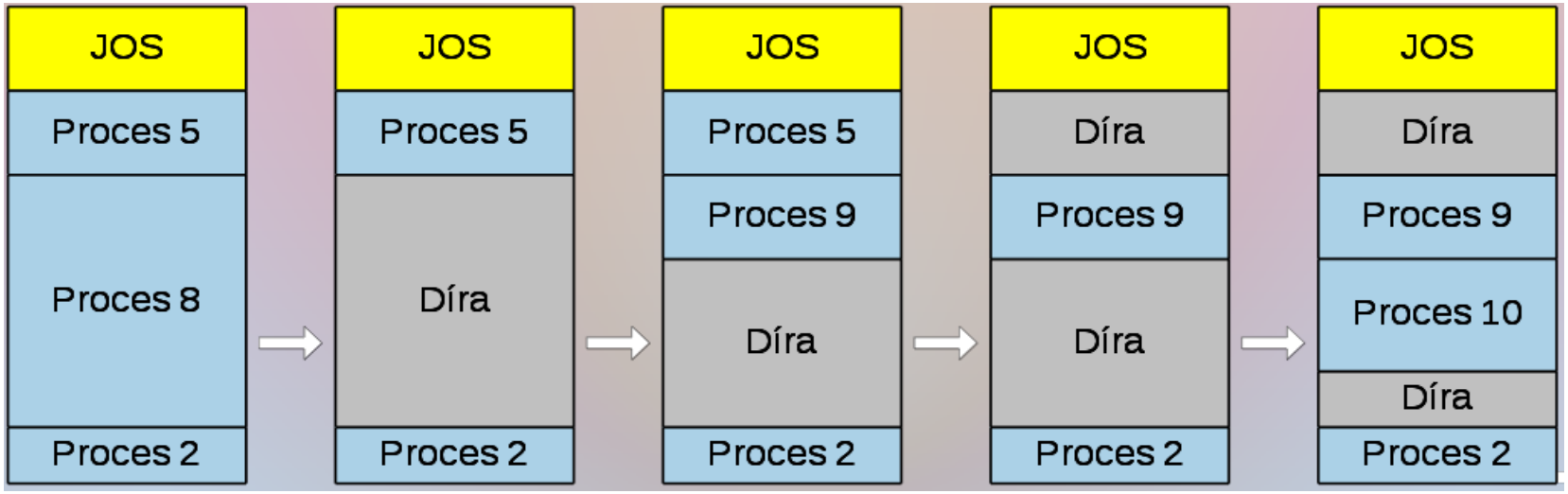
Vlastnosti segmentace

- Výhody segmentace
  - Segment má délku uzpůsobenou skutečné potřebě
    - minimum vnitřní fragmentace
  - Lze detekovat přístup mimo segment, který způsobí chybu segmentace – výjimku typu „*segmentation fault*“
  - Lze nastavovat práva k přístupu do segmentu
    - Operační systém požívá větší ochrany než aplikační proces
    - Uživatel nemůže ohrozit operační systém
  - Lze pohybovat s daty i programem v fyzické paměti
    - posun počátku segmentu je pro aplikační proces neviditelný a nedetekovatelný
- Nevýhody segmentace
  - Alokace segmentů v paměti je netriviální úloha
    - Segmenty mají různé délky. Při běhu více procesů se segmenty ruší a vznikají nové.
  - Problém s externí fragmentací
  - Režie při přístupu do paměti
    - Převod na lineární adresu se opírá o tabulku segmentů a ta je také v paměti
    - Při změně segmentového registru – nutné načíst položku z tabulky
    - Častá změna segmentů (po pár instrukcích) – časově náročná

Dynamické přidělování více souvislých sekcí

- "Díra" = blok neobsazené paměti
  - Procesu se přiděluje díra, která jeho požadavek uspokojí
  - Díry jsou roztroušeny po FAP
  - Evidenci o sekcích udržuje JOS
  - Kde přidělit oblast délky *n*, když je volná paměť rozmístěna ve více souvislých nesousedních sekcích?
    - First fit – první volná oblast dostatečné velikosti – rychlé, nejčastější
    - Best fit – nejmenší volná oblast dostatečné velikosti – neplýtvá velkými děrami
    - Worst fit – největší volná oblast – zanechává velké volné díry

Nákladné – nutno prohledat celý seznam volných děr



Problém fragmentace

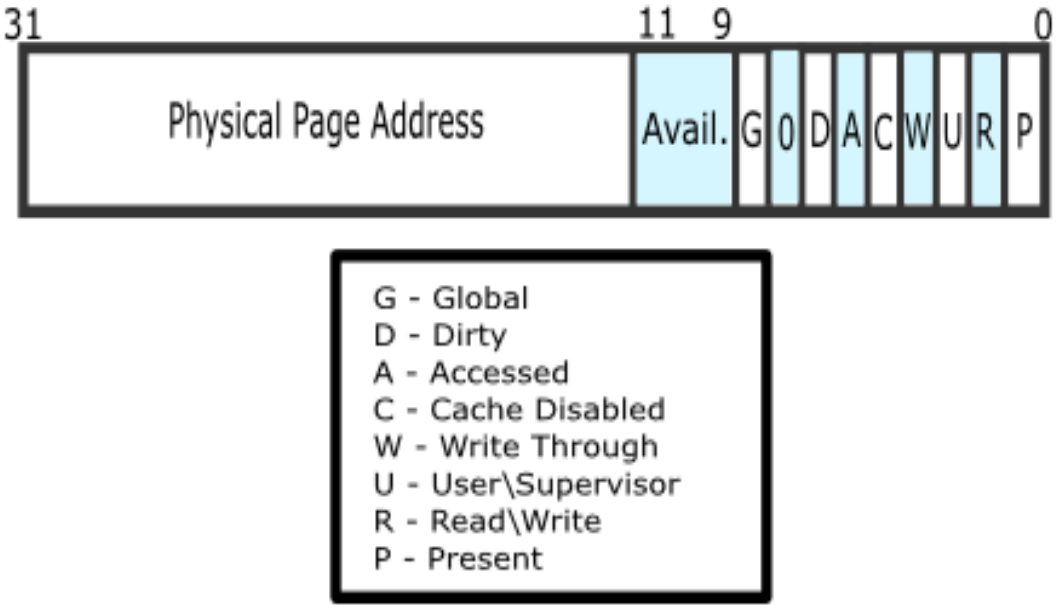
- Obecný problém
- Externí (vnější) fragmentace
  - Celkové množství volné paměti je sice dostatečné, aby uspokojilo požadavek procesu, avšak prostor není souvislý, takže ho nelze přidělit
- Interní (vnitřní) fragmentace
  - Přidělená díra v paměti je o málo větší než potřebná, avšak zbytek je tak malý, že ho nelze využít
- Redukce externí fragmentace pomocí setřásání
  - Přesouvají se obsahy úseků paměti s cílem vytvořit (jeden) velký souvislý volný blok
  - Použitelné *pouze* při dynamické relokaci
    - Při absolutních adresách v paměti by bylo nutno přepočítat a upravit všechny adresy v instrukcích
  - Problém s I/O:
    - S vyrovnávacími pamětmi plněnými z periferií (zejména přes DMA) nelze autonomně hýbat, umisťují se proto do prostoru JOS

Stránkování

- Princip
  - Souvislý LAP procesu není zobrazován jako jediná souvislá oblast FAP
- FAP se dělí na úseky zvané *rámce*
  - Pevná délka, zpravidla v celistvých mocninách 2 (512 až 8.192 B, nejčastěji 4KiB, ale někdy i 4 MiB)
- LAP se dělí na úseky zvané *stránky*
  - Pevná délka, shodná s délkou rámců
- Proces o délce *n* stránek se umístí do *n* rámců
  - rámce ale nemusí v paměti bezprostředně sousedit
- Mechanismus překladu logická adresa → fyzická adresa
  - pomocí tabulky stránek (PT = *Page Table*)
- Může vznikat vnitřní fragmentace
  - stránky nemusí být zcela zaplněny

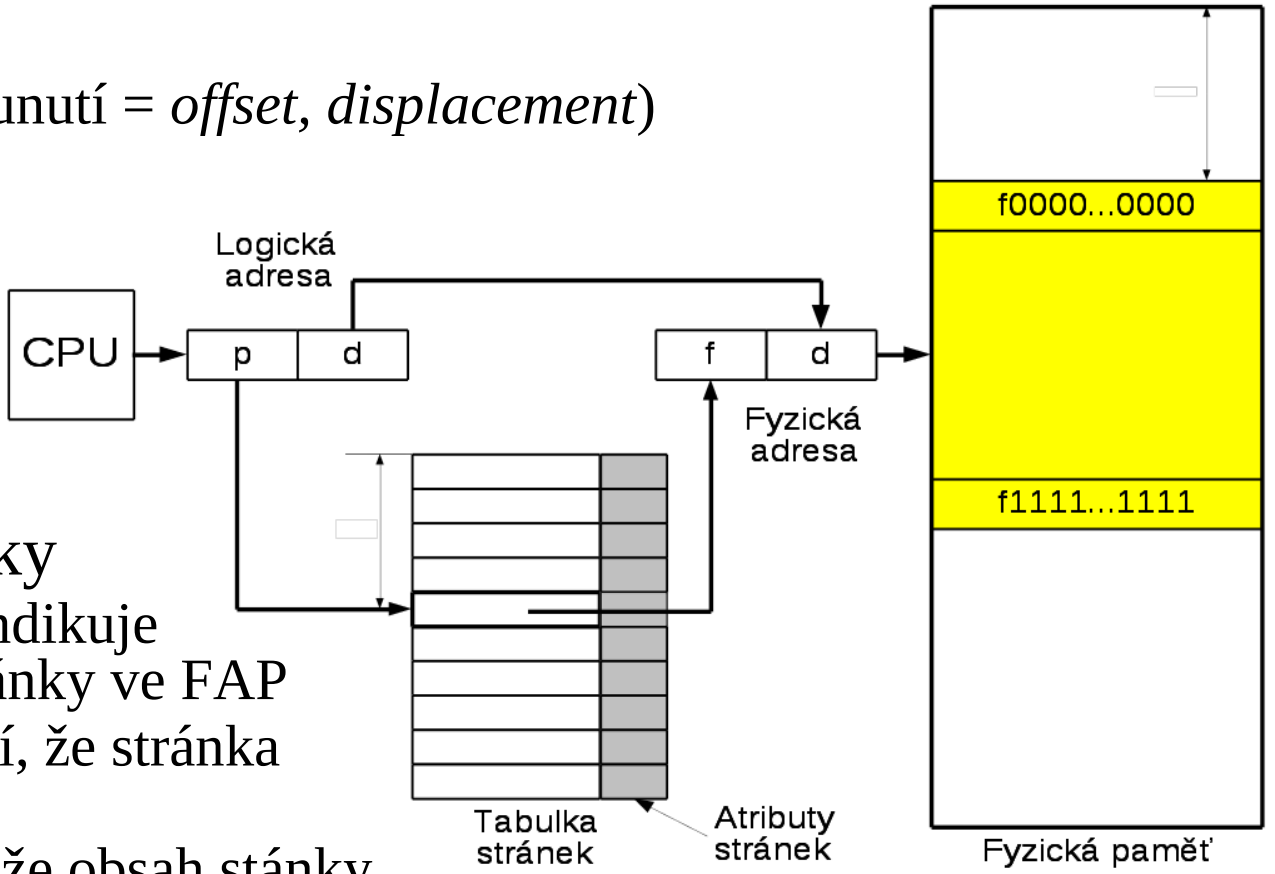
Položka tabulky stránek

- Takto vypadá položka tabulky stránek pro Intel 32 bitovou architekturu a velikost stránky 4 kiB



Stránkování – překlad adres

- Logická adresa (generovaná CPU) se dělí na
  - číslo stránky, *p* (index do tabulky stránek)
    - tabulka stránek obsahuje počáteční adresy rámců přidělených stránkám
  - offset ve stránce, *d*
    - relativní adresa (posunutí = *offset*, *displacement*) ve stránce/v rámci

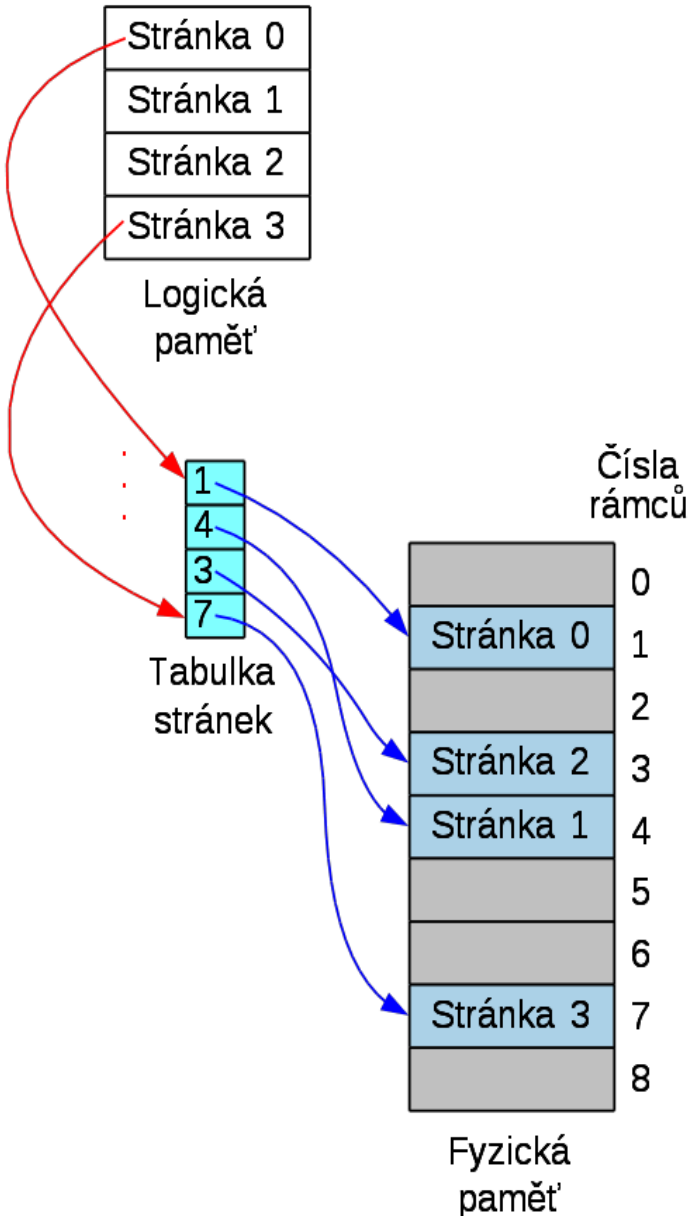


Atributy stránek

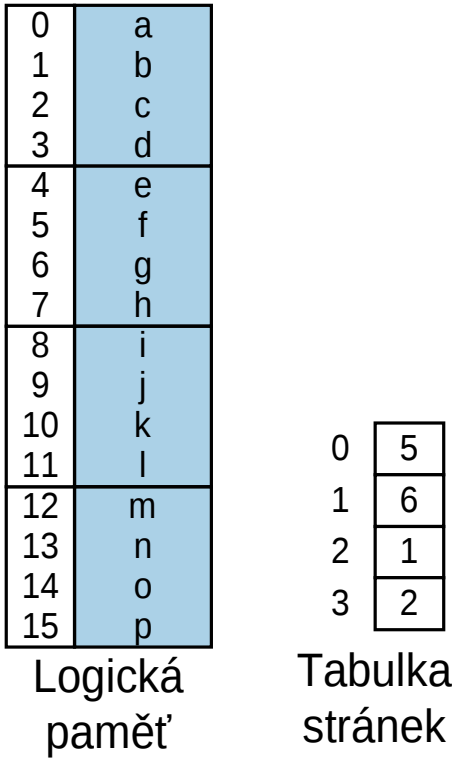
- Ochranné příznaky
  - *r/w* = *read/write*
- Virtualizační příznaky
  - *v/i* = *valid*/*invalid* indikuje přítomnost stránky ve FAP
  - *a* = *accessed* značí, že stránka byla použita
  - *d* = *dirty* indikuje, že obsah stánky byl modifikován

Příklady stránkování

Příklad 1



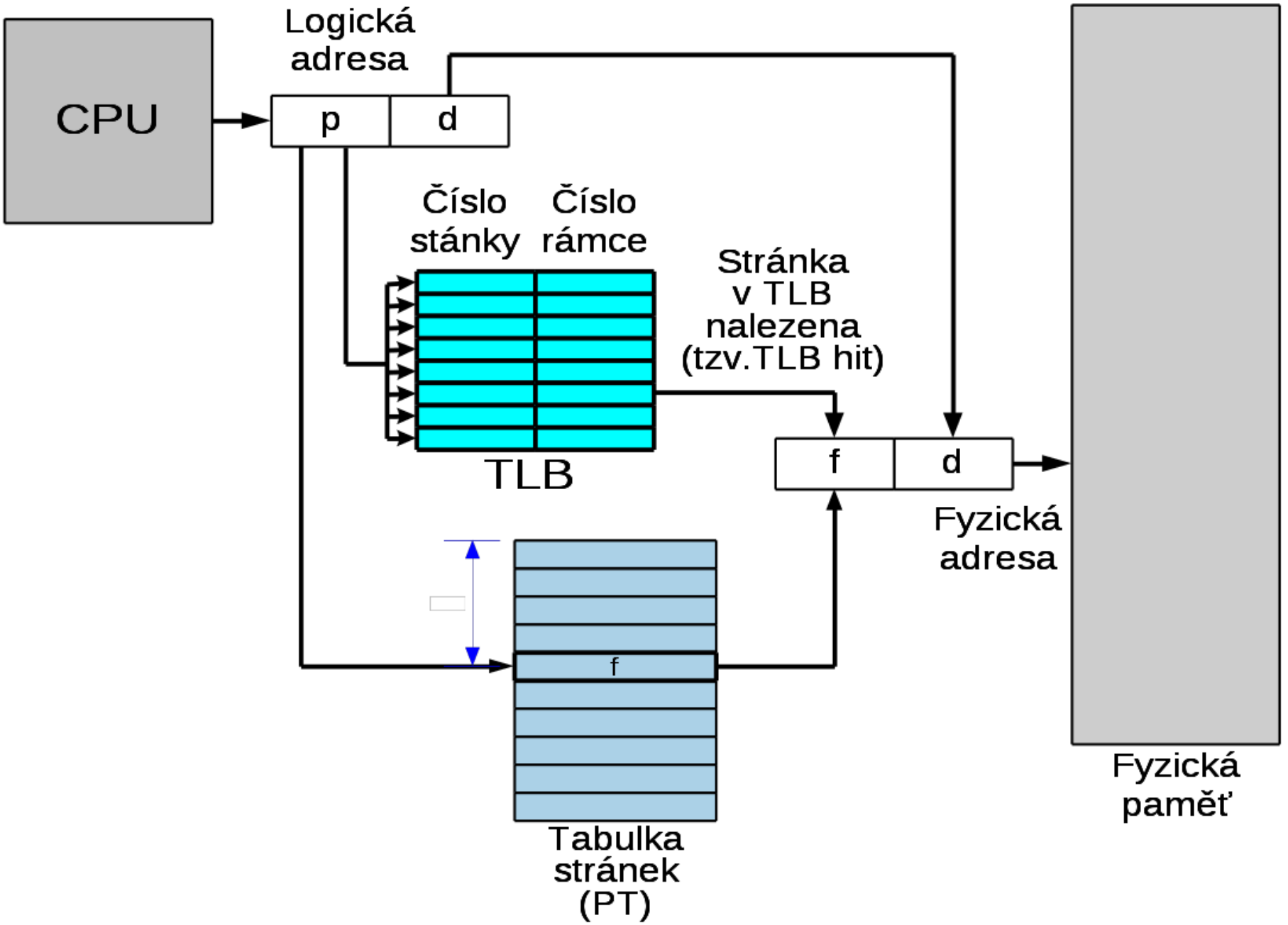
Příklad 2



Implementace tabulky stránek

- Tabulka stránek je uložena v hlavní paměti
  - Začátek odkazován spec. registrem „*Page-table base register*“ (PTBR) a její délka je v registru „*Page-table length register*“ (PTLR)
- Problém:
  - Každý přístup do paměti vyžaduje přístupy dva: přístup do tabulky stránek a vlastní přístup do paměti pro údaj/instrukci – časově náročné
  - řešení speciální rychlou cache pamětí využitím principu lokality
  - asociativní paměť, *Translation Look-aside Buffer* (TLB)
- TLB je asociativní paměť
  - relativně malá kapacita, vysoká rychlost, obvodová složitost
  - Překlad $p \rightarrow f$: Jestliže se p nachází v asociativní paměti, z paměti se získá hodnota f, jinak se f vyhledá v tabulce stránek (PT) v paměti a obsah TLB se zaktualizuje (HW nebo SW prostředky)
  - Omezení: cena

Stránkování s TLB



Zrychlení přístupu do paměti s TLB

- Skutečná přístupová doba – *Effective Access Time* (EAT)
  - Přístupová doba do fyzické paměti = t
  - Přístup to TLB = ε
  - Úspěšnost „Hit ratio“ α – pravděpodobnost, že se stránka nalezne v TLB

$$EAT = (\varepsilon + t)\alpha + (\varepsilon + 2t)(1 - \alpha)$$
$$= (2 - \alpha)t + \varepsilon$$

Příklad pro t = 100 *ns*

Jen *PT* bez *TLB*		*EAT* = 200 *ns*	Snížení rychlosti na polovinu oproti případu bez stránkování
ε = 10 *ns*	α = 60 %	*EAT* = 150 *ns*	*TLB* způsobila významné zrychlení průměrného přístupu do paměti
ε = 10 *ns*	α = 80 %	*EAT* = 130 *ns*	
ε = 10 *ns*	α = 98 %	*EAT* = 112 *ns*	

Vliv TLB

- Velikost TLB 8-4096 položek
- Moderní procesory mají více-úrovní TLB – podobně jako úrovně cache
- Intel Core i7 má 64 položek L1 první úrovně, 128 položek L1 druhé úrovně a L2 512 položek
- I pro malé TLB je úspěšnost nalezení položky 99%-99.99% (souvisí s principem lokality tj. prostorovou závislostí programů)
- Řešení nenalezení položky v TLB může být SW nebo HW(Intel).
- Problém TLB je při změně procesu a tím i změně tabulky stránek
- Intel umožňuje speciálními instrukcemi ponechat stránku v TLB

Možné struktury tabulky stránek (PT)

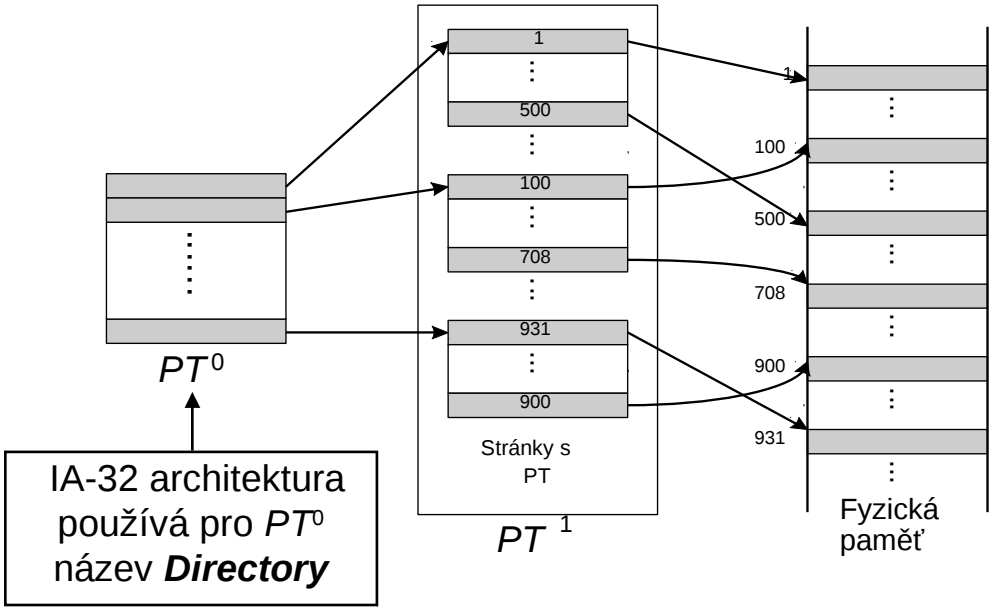
- Otázka velikosti PT
  - Každý proces má svoji PT
  - 32-bitový LAP, 4 KiB stránky → PT má 1 Mi položek, tj. 4 MiB
  - 64-bitový LAP, 8 KiB stránky – PT má 8 Pi (peta) položek, tj. 64 PiB
    - musí být stále v hlavní paměti
- Hierarchické stránkování
  - Zobrazování LAP se dělí mezi více PT
  - Pro 32-bitový LAP typicky dvouúrovňové PT
  - PT^0 obsahuje definice (odkazy) vlastních tabulek PT^1
  - „Vlastní“ tabulky stránek PT^1 mohou podléhat odkládání
    - v RAM lze zobrazovat jen skutečně využité stránky s „vlastními” PT
- Hašovaná PT
  - Náhrada přímého indexování číslem ***p*** v PT hašovací funkcí *hash*(*p*)
- Invertovaná PT
  - Jediná PT pro všechny koexistující procesy
  - Počet položek je dán počtem fyzických rámců
  - Vyhledávání pomocí hašovací funkce *hash*(*pid*, *p*)

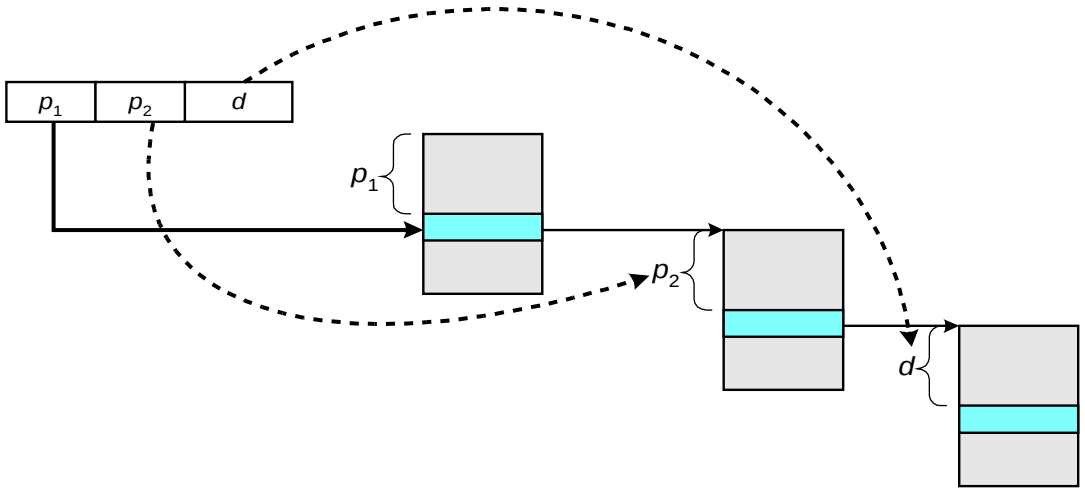
Víceúrovňové stránkování

- 64-bitový procesor se stránkou o velikosti 8 KiB
  - 51 bitů na číslo stránky → 2 Peta (2048 Tera) stránek
- Problém i pro víceúrovňové stránkování:
  - Každá další úroveň znamená další přístup do paměti a zpomalení
- UltraSparc – 64 bitů ~ 7 úrovní → neúnosné
- Linux – 64 bitů
  - Trik: používá se pouze 39/43/48 bitů ostatní ignoruje
    - Počet bitů závisí na architektuře CPU x86_64 – 48 bitů (9+9+9+9+12), IA64 (9+9+9+12), Alpha (10+10+10+13)
    - Velikost LAP je sice „pouhých“ 0.5/8/256 TB
    - 3 nebo 4 úrovně tabulek po 9 nebo 10 bitech
    - 12 nebo 13 bitů offset ve stránce
    - ještě únosná režie
    - velký význam TBL 4 urovně, znamená 5 čtení z paměti, tedy zrychlení z 500ns na 110 ns

Dvouúrovňové stránkování – příklad

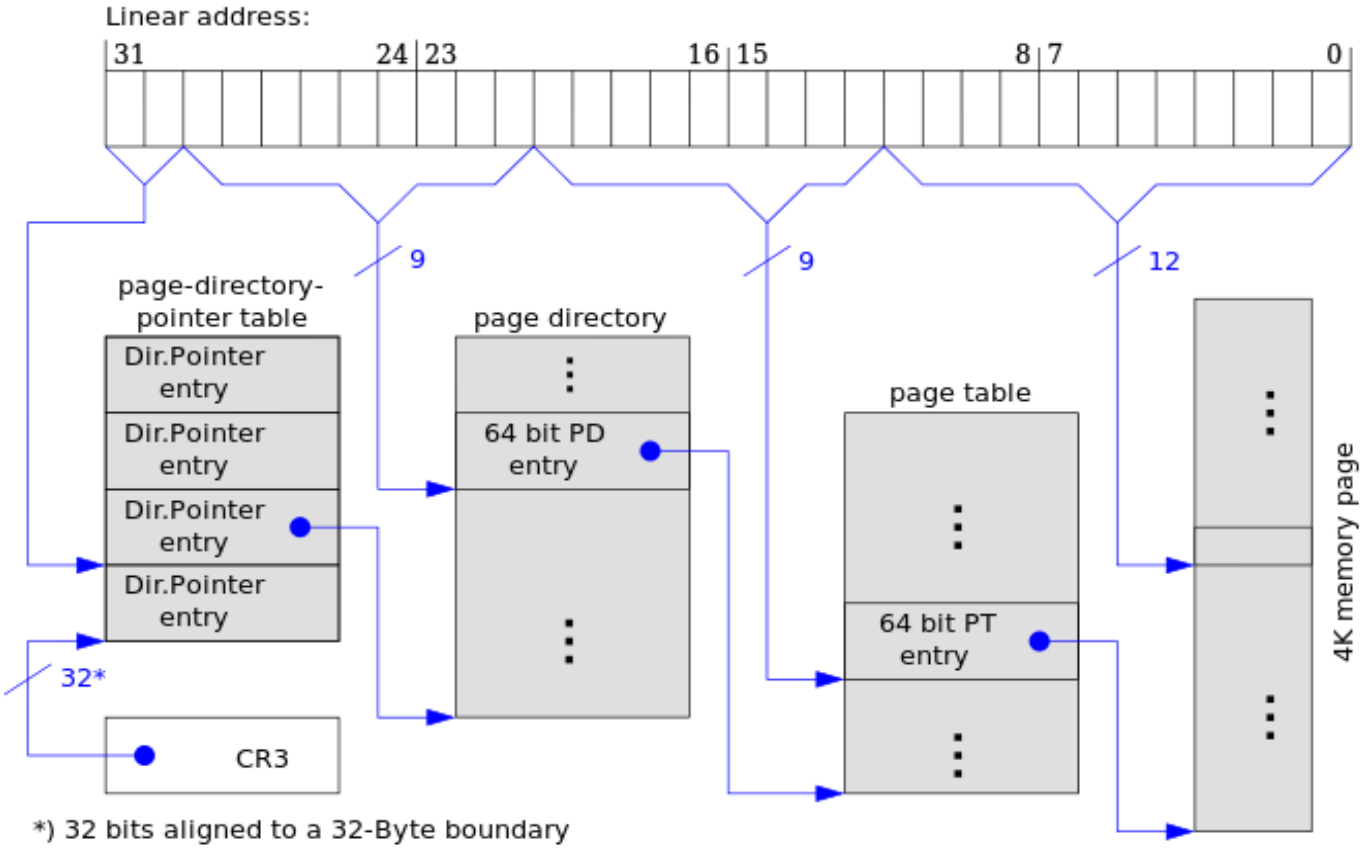
- 32-bitový stroj se stránkou o velikosti 4 KB
  - Např. IA-32 (Pentium): stránka 4 KiB, tj. 12 bitů adresy ve stránce
  - 20 bitů na číslo stránky → 2^{20} stránek
- Logická adresa 32 bitů
  - 20 bitů čísla stránky a 12 bitů adresy ve stránce (offset)
  - Číslo stránky se dále dělí na
    - Index do PT^0 – tzv. „vnější PT“ – 10 bitů (p_1) a
    - Offset v PT^1 – 10 bitů (p_2)





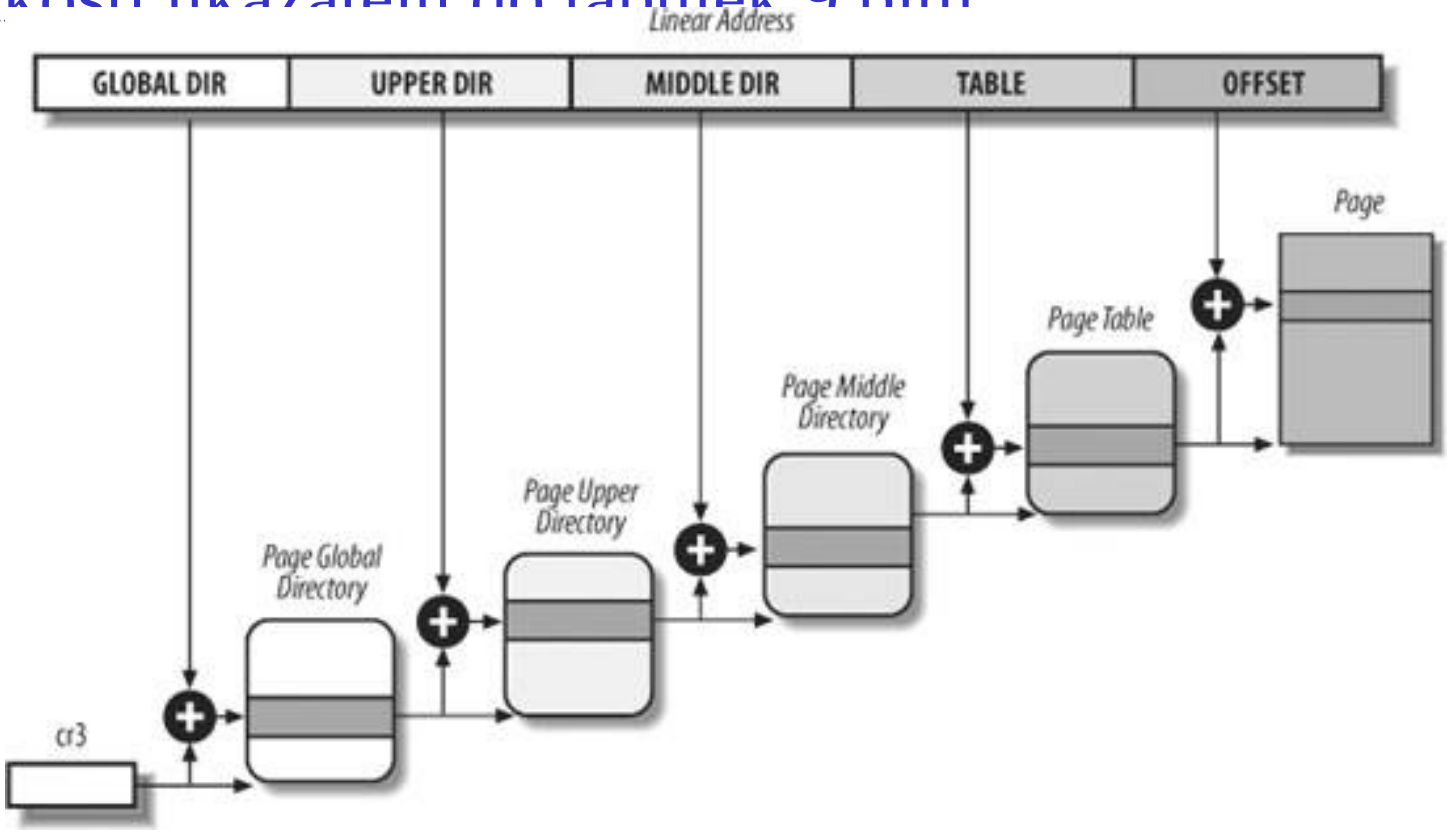
Víceúrovňové stránkování s PAE

- PAE - Physical Address Extension – rozšíření adresového prostoru z 32 bitů na 36(Intel)/52(AMD)
- Poprvé se objevilo v roce 1995
- Nová hierarchie stránkovacích tabulek



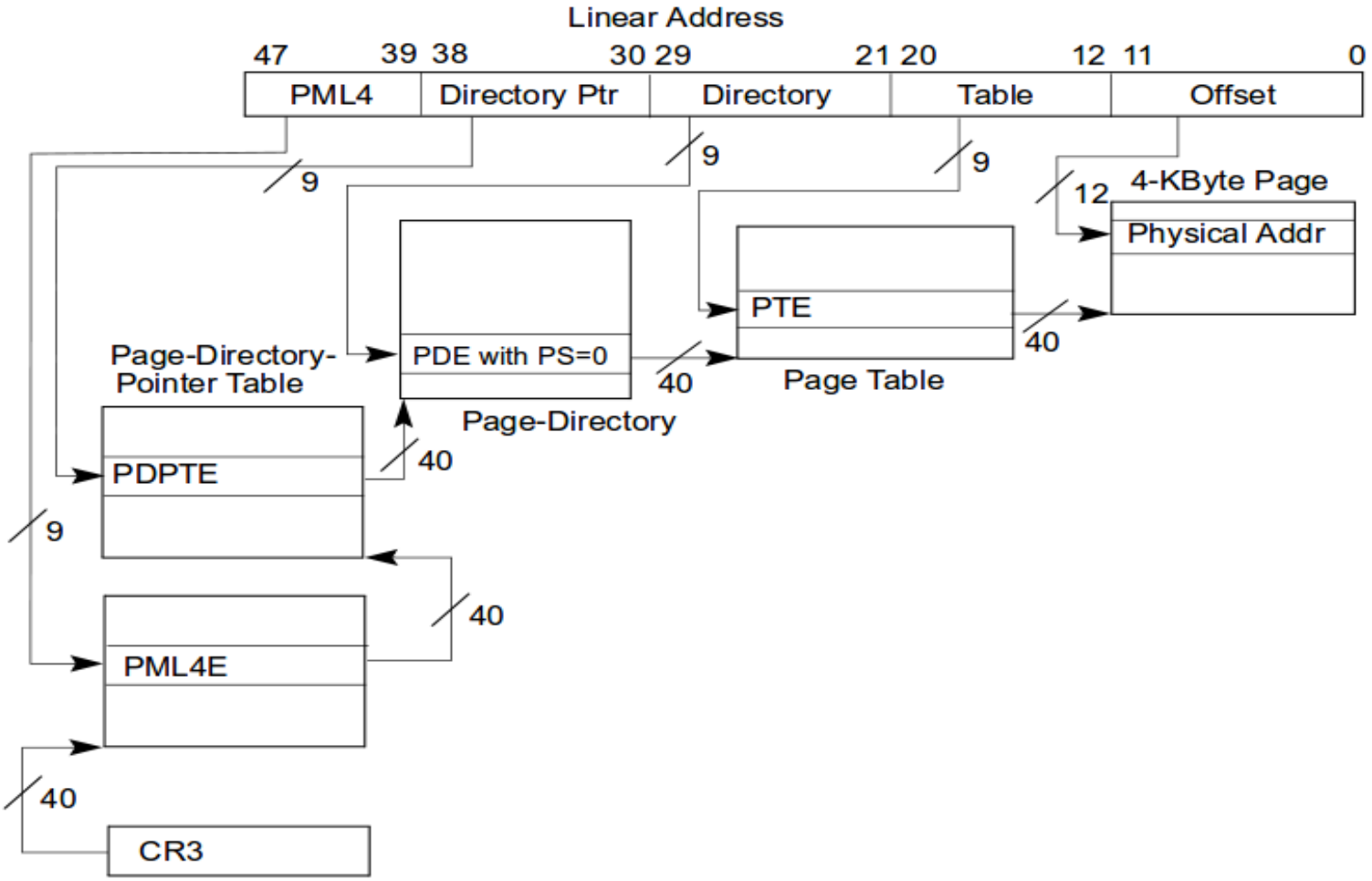
Stránkování 64 bitů

- 4-úrovňové stránkování v Linuxu pro procesory Intel
- Offset 12 bitů
- Velikosti ukazatelů do tabulek 9 bitů



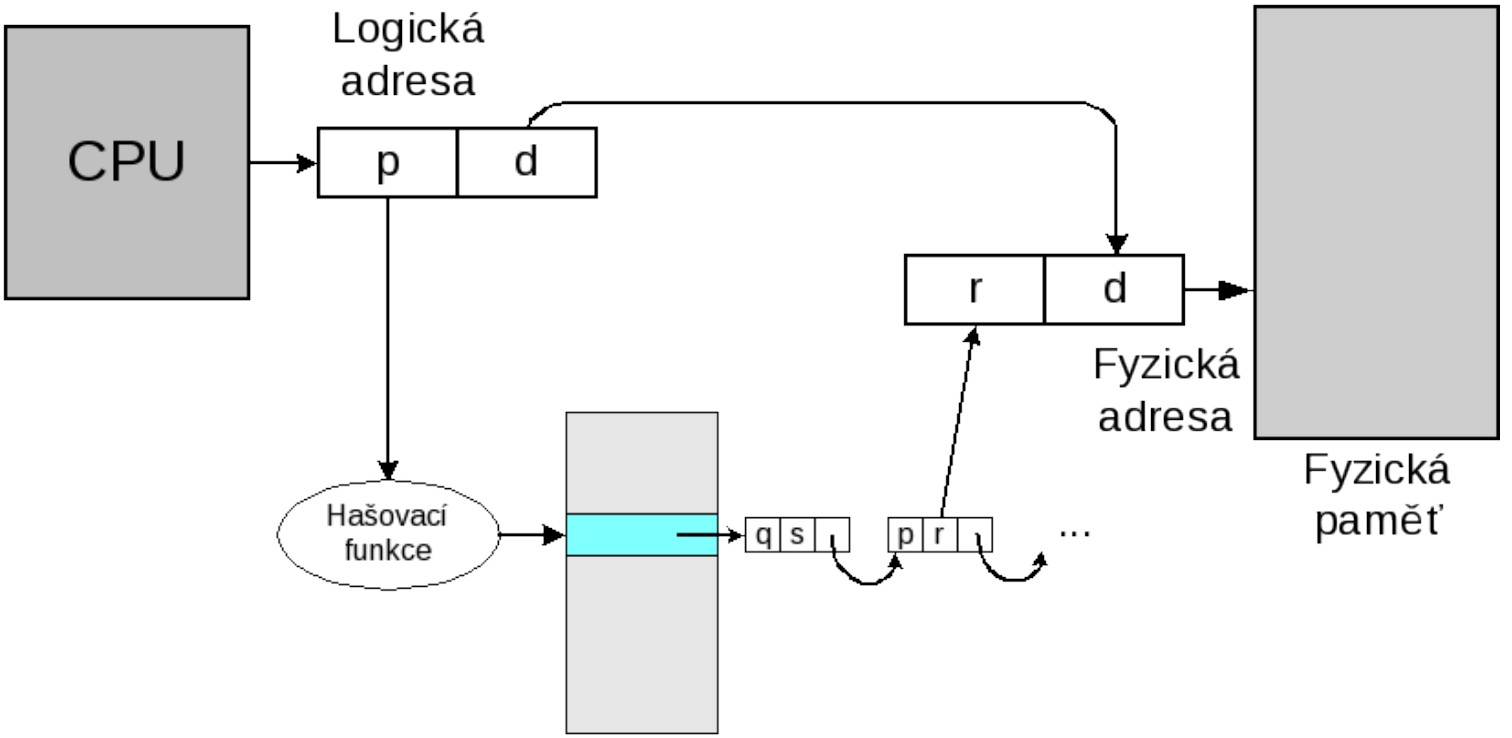
Stránkování IA-32e

- Lineární adresa 48 bits
- Fyzická adresa 52 bits – což je 4 PiB RAM
- Varianty s 2MiB nebo 1GiB stránkami



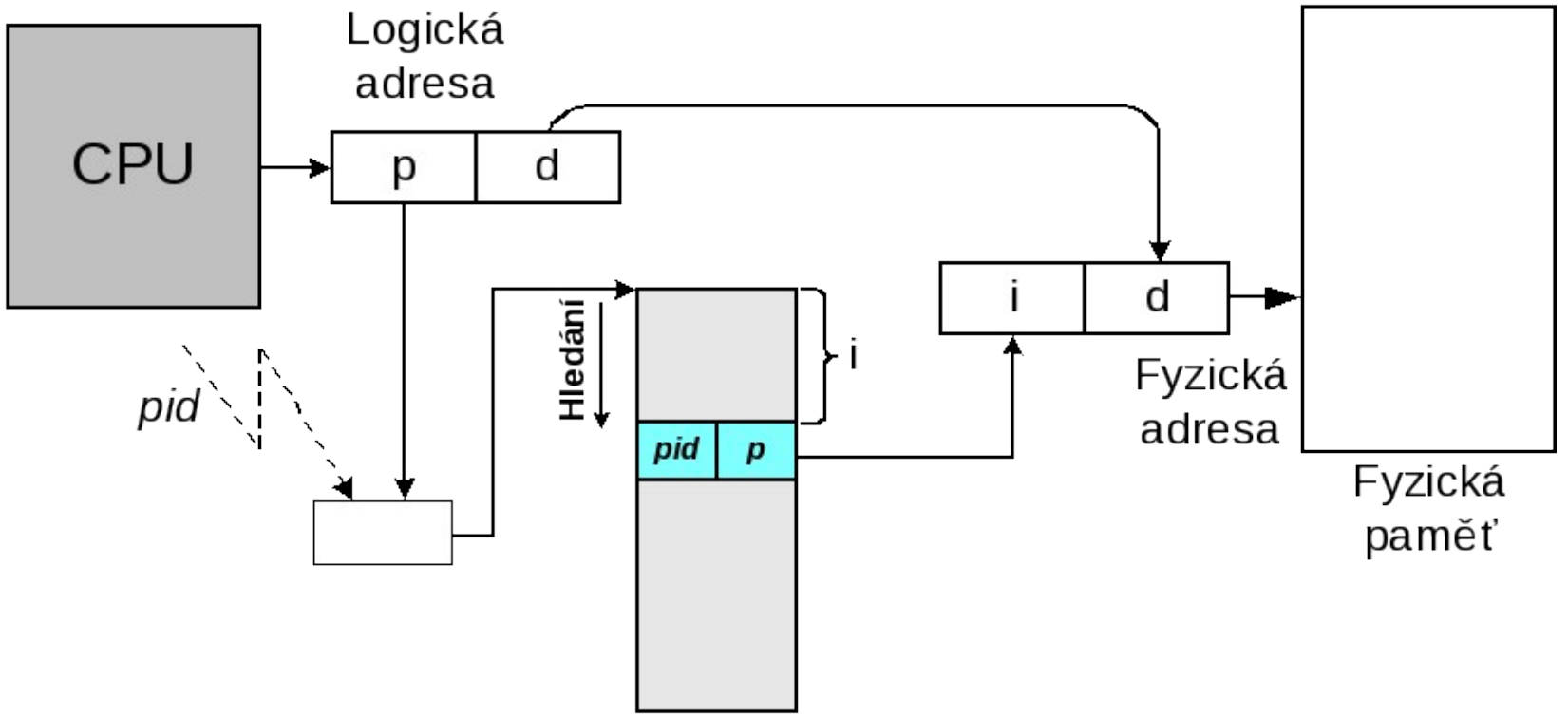
Hašovaná stránkovací tabulka

- Používá se zejména pro veliké adresní prostory
  - šíře adresy více než 32 bitů (např. Intel Itanium 64 bit)
- Číslo stránky se hašuje do tabulky stránek
  - ta obsahuje zřetězené prvky hašované do stejného místa
  - prvek řetězce = (číslo stránky, číslo fyzického rámce)
  - v řetězci se hledá číslo stránky a odtud se získá číslo rámce
  - Hašování i prohledávání řetězce by mělo být realizováno v HW



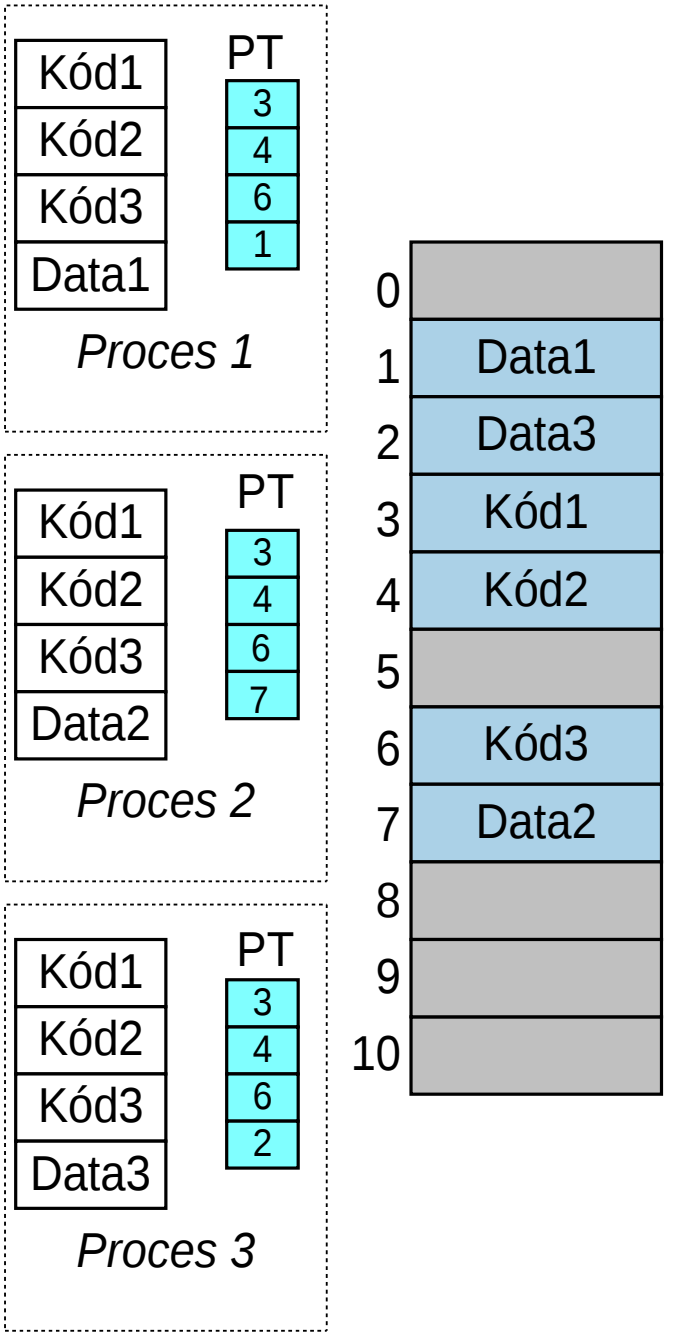
Invertovaná stránkovací tabulka (IPT)

- Index do invertované PT je *číslo rámce* (nikoliv č. stránky)
  - Velikost IPT je dána počtem rámců fyzické paměti
  - Jediná IPT pro všechny procesy, řádek obsahuje ***pid*** a ***p***
  - V položkách IPT jsou skutečně alokované rámce fyzické paměti
    - užito např. v PowerPC
- Šetří se místo v paměti za cenu delšího prohledávání
  - Prohledávání lze opět zrychlit hašováním



Sdílení stránek

- Sdílený kód
  - Jediná *read-only* kopie (reentrantního) kódu ve FAP sdílená více procesy
    - více instancí editoru, shellů, apod.
- Privátní kód a data
  - Každý proces si udržuje svoji vlastní kopii kódu a dat
  - Stránky s privátním kódem a daty mohou být kdekoliv v LAP
- Sdílená data
  - Potřebná pro implementaci meziprocesní komunikace



Segmentace se stránkováním

- Kombinace obou výše uvedených metod
  - Ponechává výhody segmentace, např. možnost přesného omezení paměťového prostoru
  - Přináší výhodu jednoduchého umísťování segmentu do fyzické paměti. Ve fyzické paměti je pouze ta část segmentu, která se používá.
- Tabulka segmentů ST obsahuje místo báze segmentu
  - buď adresu stránkovací tabulky PT
  - nebo tzv. lineární adresu používanou pro přepočet na fyzickou adresu
- Segmentace se stránkováním je používána architekturou IA-32 (např. INTEL-Pentium)

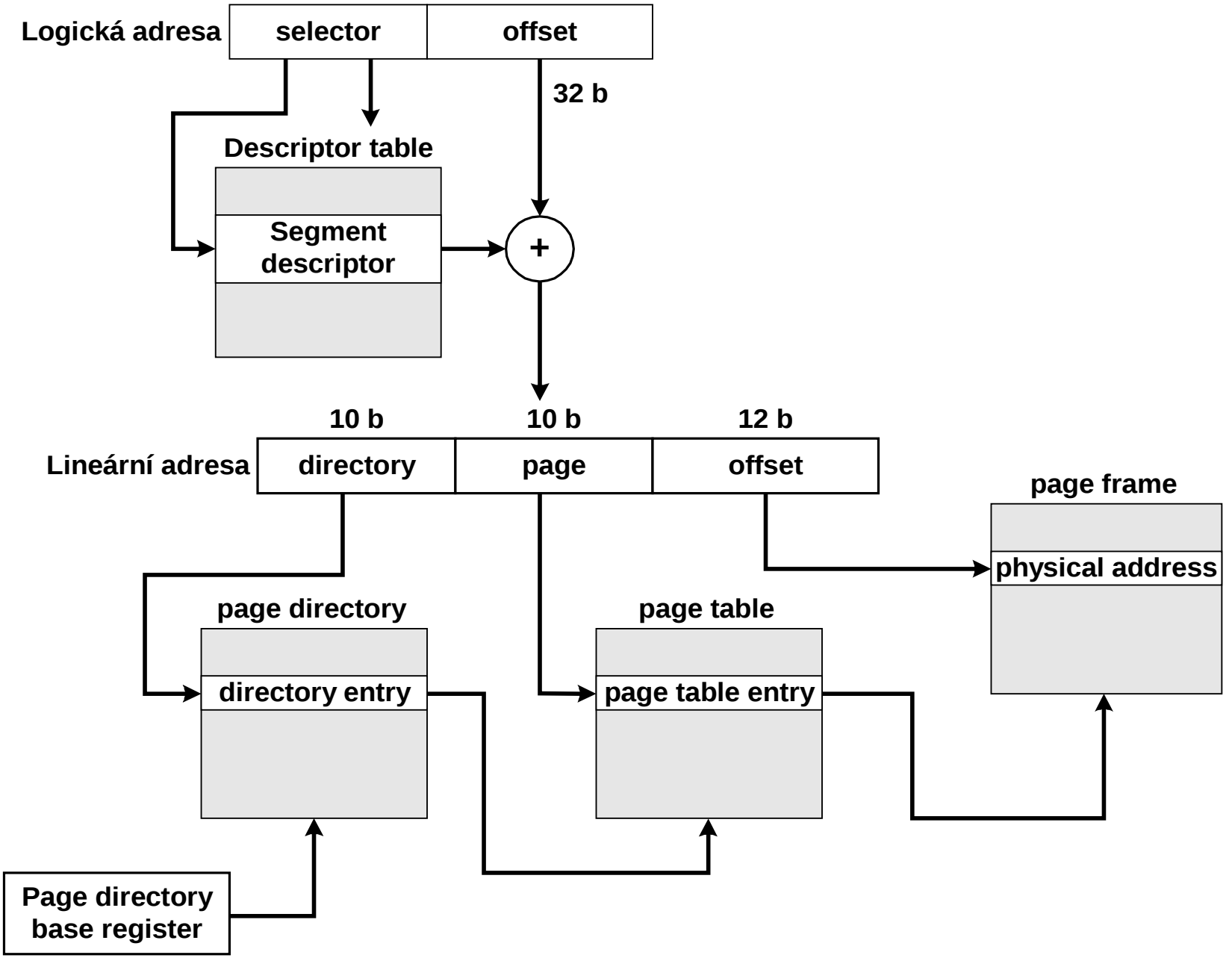
LAP → FAP v IA-32

- IA-32 transformace LAP → FAP
  - LAP lineární (4 GiB), transformace identita
    - používají zpravidla pouze DMA řadiče
  - LAP lineární (4 GiB), stránkování,
    - 1024 oblastí à 4 MiB, délka stránky 4 KiB, 1024 tabulek stránek, každá tabulka stránek má 1024 řádků
  - LAP segmentovaný, segmentace
    - 16 Ki segmentů à 4 GiB ~ 64 TiB
  - LAP segmentovaný stránkovaný, segmentace se stránkováním
    - Segmentace vybírá části LAP, stránkování zobrazuje LAP do FAP
    - Používají Windows, Linux

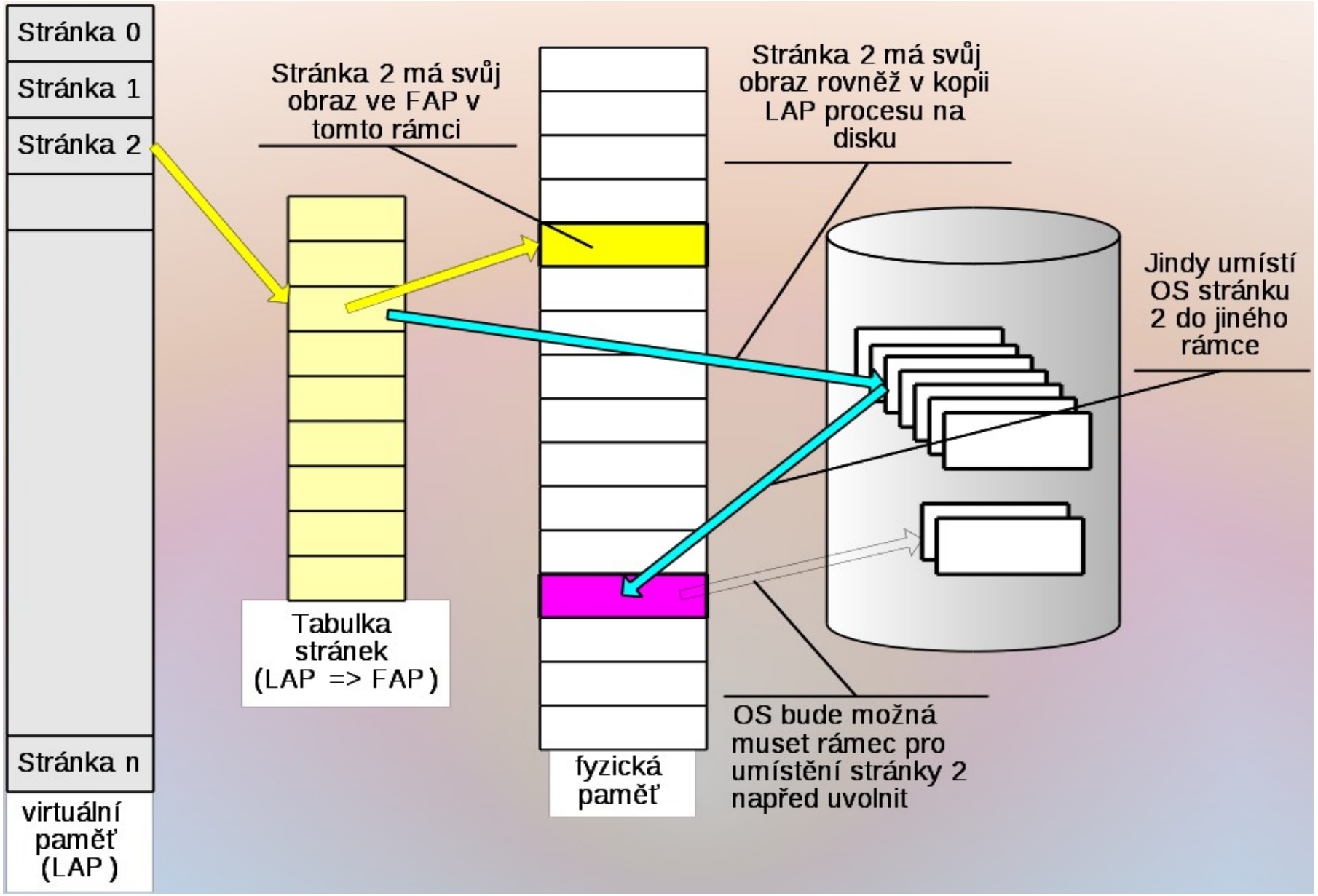
Segmentace se stránkováním IA-32

- LAP: 2x8 Ki segmentů s délkou až 4 GiB každý
- Dva logické podprostory (popisovač TI = 0 / 1)
  - 8 Ki privátních segmentů procesu
    - popisuje tabulka segmentů Local Description Table, LDT
  - 8 Ki segmentů sdílených procesy
    - popisuje tabulka segmentů Global Description Table, GDT
- Logická adresa = (popisovač segmentu, offset)
  - offset = 32-bitová adresa v segmentu, segment je stránkován
  - popisovač segmentu
    - 13 bitů *číslo segmentu*,
    - 1 bit *popisovač TI*,
    - 2 bity *úroveň ochrany*: segment jádra, ..., segment aplikace
    - práva *r*/*w* až na úrovni stránek
- Lineární adresní prostor uvnitř segmentu
  - stránkuje s použitím dvouúrovňového mechanismu stránkování
    - délka stránky 4 KiB, offset ve stránce 12 bitů, číslo stránky 2x10 bitů

Segmentace se stránkováním Pentium

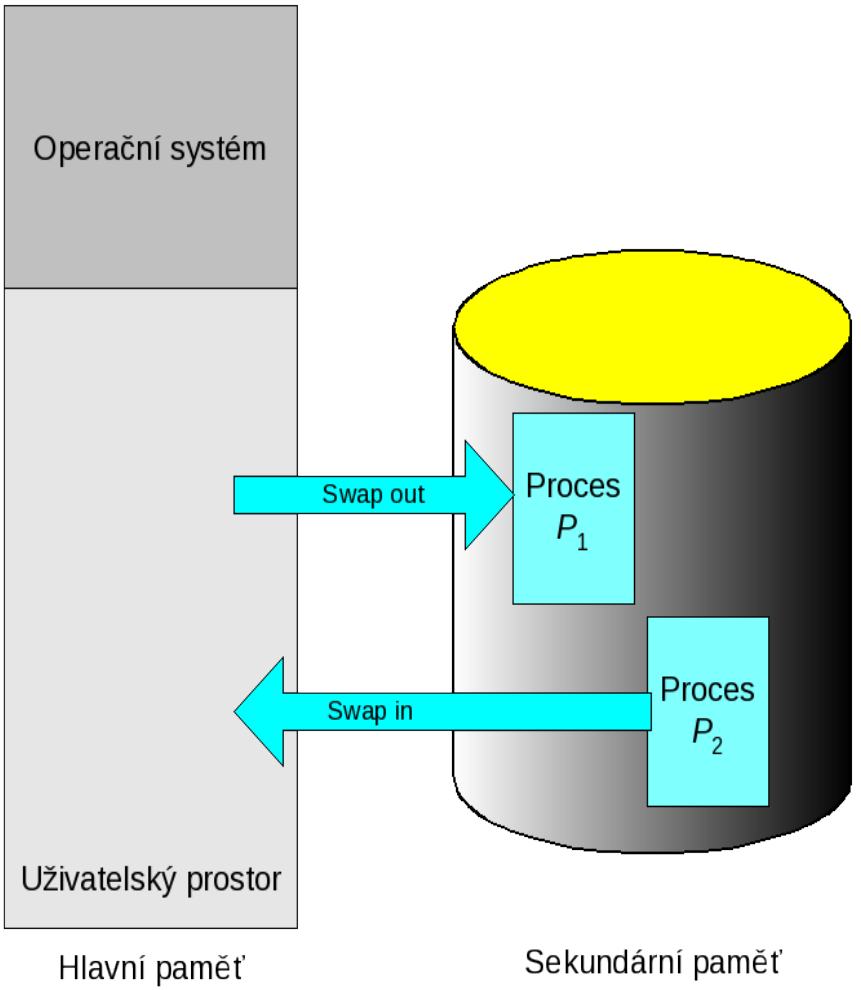


Virtuální paměť je větší než fyzická



Výměny, odkládání (*Swapping*)

- Úsek FAP přidělený procesu je vyměňován mezi vnitřní a vnější (sekundární) pamětí oběma směry
- Výpis na disk, načtení z disku
  - *Swap out, swap in* (*roll out, roll in*)
- Trvání výměn je z podstatné části tvořena dobou přenosu mezi pamětí a diskem
  - je úměrná objemu vyměňované paměti
- Princip používaný v mnoha OS
  - pokud nepodporují virtualizaci, nebo pracují na HW, které nedává podporu pro virtualizaci
- Základní myšlenka virtualizace



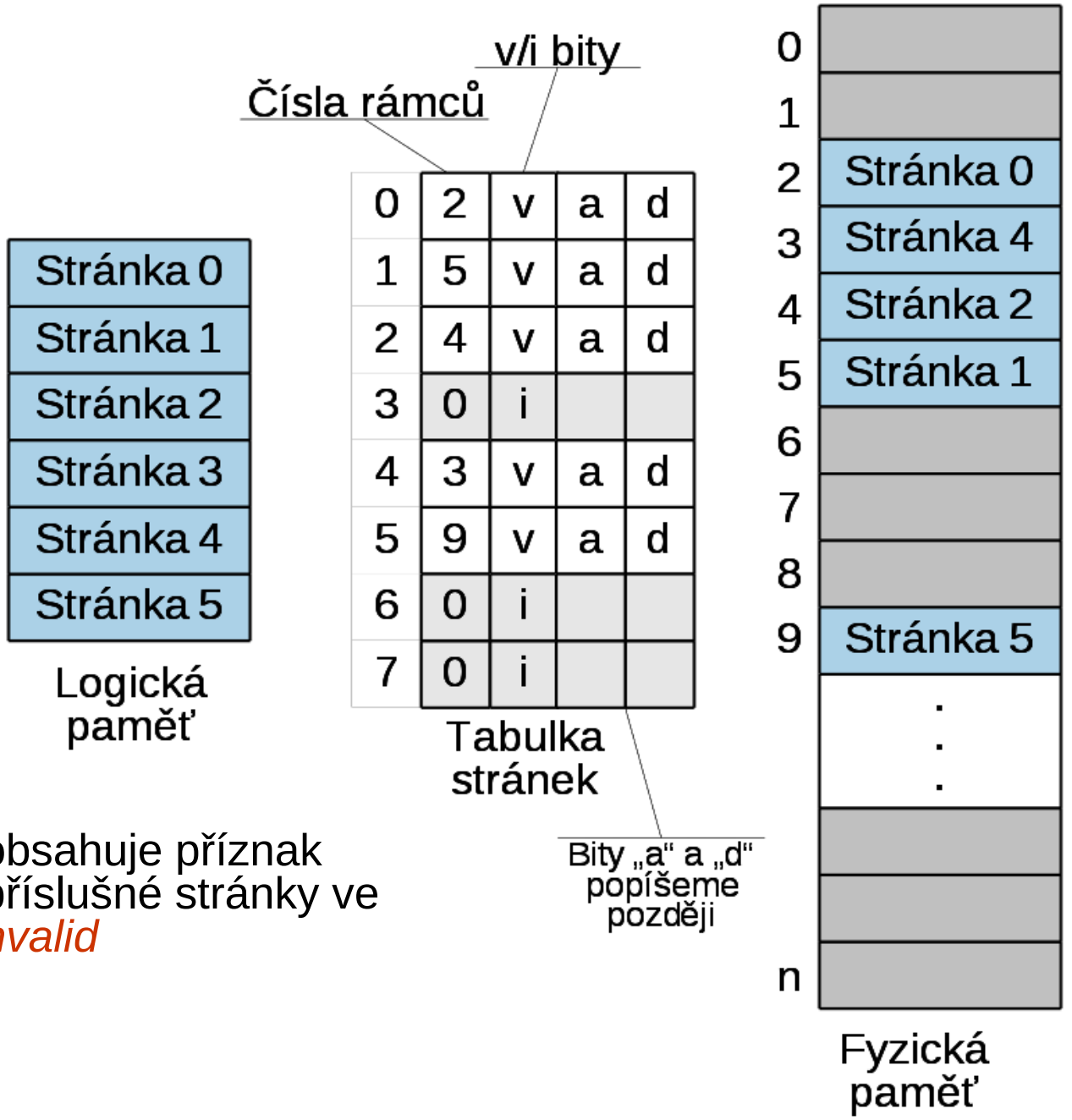
Odkládá se celý proces!

Principy stránkování

- Kdy stránku zavádět do FAP? (*Fetch policy*)
  - stránkování při spuštění
    - Program je celý vložen do paměti při spuštění
    - velmi nákladné a zbytečné, předem nejsou známy nároky na paměť, neužívá se
  - stránkování či segmentace na žádost (*Demand Paging/Segmentation*)
    - Tzv. „líná metoda“, nedělá nic dopředu
  - předstránkování (*Prepaging*)
    - Nahrává stránku, která bude pravděpodobně brzy použita
  - čištění (*Pre-cleaning*)
    - Změněné rámce jsou ukládány na disk v době, kdy systém není vytížen
  - kopírovat při zápisu (*copy-on-write*)
    - Při tvorbě nového procesu není nutné kopírovat žádné stránky, ani kódové ani datové. U datových stránek se zruší povolení pro zápis.
    - Při modifikaci datové stránky nastane chyba, která vytvoří kopii stránky a umožní modifikace

Příznak *v/i* v tabulce stránek

v/i = Valid/Invalid
Příznaky *a* a *d* ➔



Každá položka v PT obsahuje příznak indikující přítomnost příslušné stránky ve FAP – příznak *valid/invalid*

Procesy ve virtuální paměti

- Při startu procesu zavede OS do FAP pouze tu část programu (LAP) kam se iniciálně předává řízení
  - Pak dochází k dynamickému zavádění částí LAP do FAP po stránkách či po segmentech „na žádost”
    - tj. až když je jejich obsah skutečně referencován
- Pro překlad LA → FA
  - Tabulka stránek (PT) a/nebo segmentů (ST)
  - Sada stránek procesu, které jsou ve FAP – rezidentní množina (*resident set*)
  - Odkaz mimo rezidentní množinu způsobuje přerušení výpadkem stránky/segmentu (*page /segment fault*) a tím vznikne „žádost“
    - Proces, jemuž chybí stránka, označí OS jako pozastavený
    - OS spustí I/O operace k zavedení chybějící stránky do FAP (možná bude muset napřed uvolnit některý rámec, viz politika nahrazování dále)
    - Během I/O přenosu běží jiné procesy; po zavedení stránky do paměti se aktualizuje tabulka stránek, „náš“ proces je označen jako připravený a počká si na CPU, aby mohl pokračovat

Stránkování na žádost

- Základní politika stránkování na žádost (*Demand paging*)
  - Při překladu LA → FA se zjistí, že stránka není ve FAP
    - Hardware testuje bit *Valid*/*Invalid* v položce PT
    - Pokud je *Invalid*, generuje se výjimka (přerušení) typu výpadek stránky
    - Při inicializaci procesu jsou všechny bity nastaveny na *Invalid*
    - Stránka se zavádí jako reakce na výpadek stránky
    - Výhoda: Málo I/O operací
    - Nevýhoda: Na počátku běhu procesu se tak tvoří série výpadků stránek a proces se „pomalu rozbíhá“

Princip lokality

- Odkazy na instrukce programu a data tvořívají shluky
- Vzniká časová lokalita a prostorová lokalita
  - Provádění programu je s výjimkou skoků a volání podprogramů sekvenční
  - Programy mají tendenci zůstávat po jistou dobu v rámci nejvýše několika procedur
  - Většina iterativních výpočtů představuje malý počet často opakovaných instrukcí,
  - Často zpracovávanou strukturou je pole dat nebo posloupnost záznamů, které se nacházejí v „sousedních“ paměťových lokacích
- Lze pouze dělat odhady o částech programu/dat, která budou potřebná v nejbližší budoucnosti

Stránkování na žádost – vylepšení

- Předstránkování (*Pre-paging*)
  - Sousední stránky LAP obvykle sousedí i na sekundární paměti, a tak je jich zavádění poměrně rychlé
    - bez velkých přejezdů diskových hlaviček
  - Platí princip časové lokality – proces bude pravděpodobně brzy odkazovat blízkou stránku v LAP. Zavádí se proto najednou více stránek
  - Výhodné zejména při inicializaci procesu - menší počet výpadků stránek
  - Nevýhoda: Mnohdy se zavádějí i nepotřebné stránky
- Čištění (*Pre-cleaning*)
  - Pokud má počítač volnou kapacitu na I/O operace, lze spustit proces kopírování změněných stránek na disk
  - Výhoda: uvolnění stránky je rychlé, pouze nahrání nové stránky
  - Nevýhoda: Může se jednat o zbytečnou práci, stránka se ještě může změnit

Stránkování – Politika nahrazování

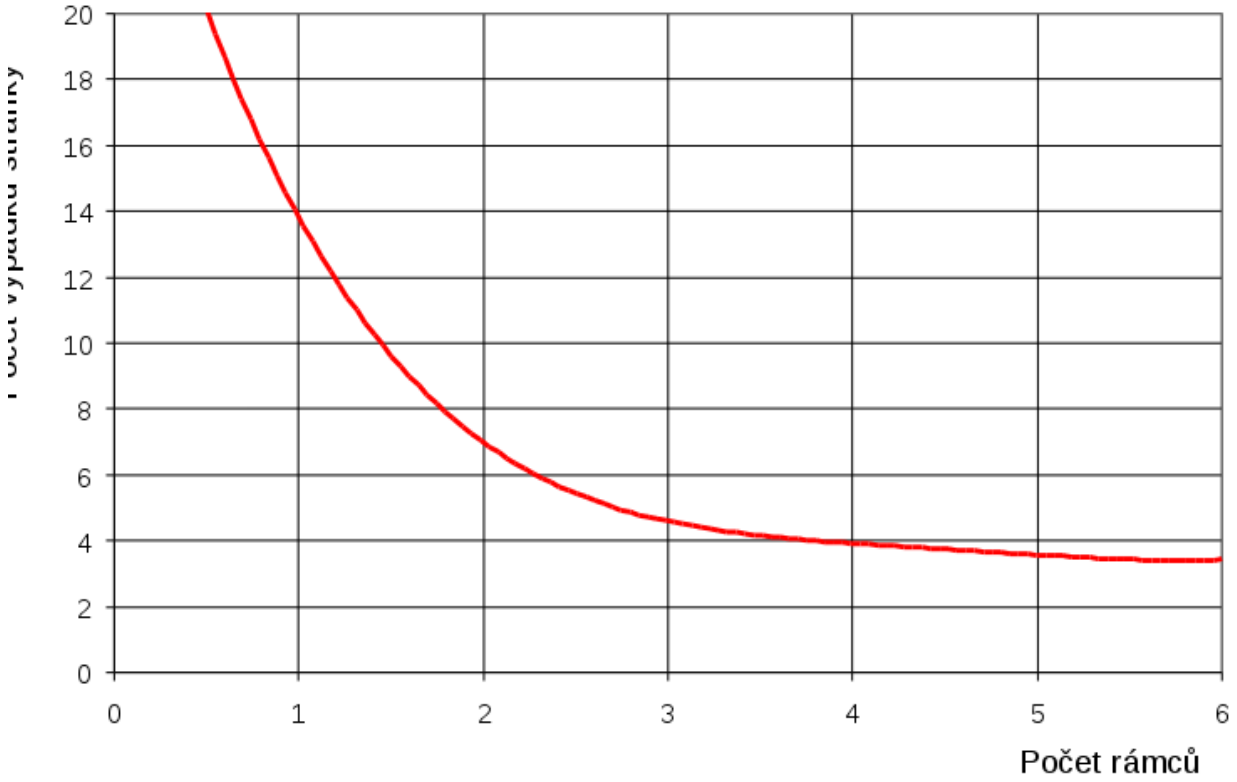
- Co činit, pokud není volný rámec ve FAP
  - Např. v okamžiku zvýšení stupně paralelismu (nový proces)
- Politika nahrazování (*Replacement Policy*)
  - někdy též politika výběru oběti
  - Musí se vyhledat vhodná stránka pro náhradu (tzv. oběť)
  - Kterou stránku „obětovat“ a „vyhodit“ z FAP?
  - Kritérium optimality algoritmu: minimalizace počtu (či frekvence) výpadků stránek
- Určení oběti:
  - Politika nahrazování říká, jak řešit problémy typu
    - Kolik rámců procesu přidělit?
    - Kde hledat oběti? Jen mezi stránkami procesu, kterému stránka vypadla nebo lze vybrat oběť i mezi stránkami patřícími ostatním procesům?
  - Některé stránky nelze obětovat
    - Některé stránky jsou dočasně „zamčené“, tj. neodložitelné
      - typicky V/V vyrovnávací paměti, řídicí struktury OS, ...
  - Je-li to třeba, musí se rámec vypsat na disk („*swap out*“)
    - Nutné, pokud byla stránka od svého předchozího „*swap in*“ modifikována. K tomu účelu je v řádku PT tzv. dirty (*modified*) bit, který je automaticky (hardwarově) nastavován při zápisu do stránky (rámce).

Algoritmy výběru oběti

- Požadujeme minimální frekvenci výpadků stránek
- Volba vhodného algoritmu
  - Algoritmus se vyhodnocuje tak, že se pro zadanou posloupnost referencí na stránky (tzv. řetězec referencí) se modeluje a počítá množství výpadků stránek při daném počtu rámců
- Pro naše ukázky použijeme řetězec referencí do stránek

  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

- Očekávané chování:
  - kvalitativní graf



Algoritmus First-In-First-Out (FIFO)

- Obětí je vždy nejstarší stránka
- 3 rámce (ve FAP mohou být až 3 stránky)

Reference:	1	2	3	4	1	2	5	1	2	3	4	5	
Číslo rámce	Obsahy rámců												Výpadky **tučně**
1	**1**	1	1	**4**	4	4	**5**	5	5	5	5	5	
2		**2**	2	2	**1**	1	1	1	1	**3**	3	3	
3			**3**	3	3	**2**	2	2	2	2	**4**	4	**9 výpadků**

- 4 rámce (ve FAP mohou být až 4 stránky)

Reference:	1	2	3	4	1	2	5	1	2	3	4	5	
Číslo rámce	Obsahy rámců												Výpadky **tučně**
1	**1**	1	1	1	1	1	**5**	5	5	5	**4**	4	
2		**2**	2	2	2	2	2	**1**	1	1	1	**5**	
3			**3**	3	3	3	3	3	**2**	2	2	2	**10 výpadků**
4				**4**	4	4	4	4	4	**3**	3	3	

  - *Beladyho anomálie*
    - oproti očekávání:více rámců – více výpadků
- FIFO – jednoduché, avšak neefektivní
    - I staré stránky se používají často

Optimální algoritmus

- Oběť – stránka, která bude odkazována ze všech nejpozději
  - tj. po nejdelší dobu se do ní odkaz nepovede
  - Budoucnost však neznáme
    - lze jen přibližně predikovat
  - Lze užít jen jako porovnávací standard pro ostatní algoritmy

- Příklad: 4 rámce
  - Díky zadanému řetězci referencí „známe budoucnost“

Reference:	1	2	3	4	1	2	5	1	2	3	4	5	
Číslo rámce	Obsahy rámců												
1	**1**	1	1	1	1	1	1	1	1	1	**4**	4	
2		**2**	2	2	2	2	2	2	2	2	2	2	
3			**3**	3	3	3	3	3	3	3	3	3	**6 výpadků**
4				**4**	4	4	**5**	5	5	5	5	5	Lepšího výsledku dosáhnout nelze

Algoritmus LRU (*Least Recently Used*)

- Predikce založená na nedávné historii
  - Předpoklad: Stránka, která nebylo dlouho odkazována, nebude odkazována ani v blízké budoucnosti
- Oběť – stránka, která nejdelší dobu nebyla odkazována
  - LRU se považuje za nejlepší aproximaci optimálního algoritmu
    - bez věštecké křišťálové koule lze těžko udělat něco lepšího

- Příklad: 4 rámce

Reference:	1	2	3	4	1	2	5	1	2	3	4	5	
Číslo rámce	Obsahy rámců												
1	**1**	1	1	1	1	1	1	1	1	1	1	**5**	
2		**2**	2	2	2	2	2	2	2	2	2	2	**8 výpadků**
3			**3**	3	3	3	**5**	5	5	5	**4**	4	
4				**4**	4	4	4	4	4	**3**	3	3	

  - FIFO 10 výpadků; optimální algoritmus 6 výpadků

Algoritmus LRU – implementace

- Řízení časovými značkami
  - Ke každé stránce (rámci) je hardwarově připojen jeden registr, do nějž se při přístupu do stránky hardwarově okopírují systémové hodiny (*time stamp*)
    - Při hledání oběti se použije stránka s nejstarším časovým údajem
    - Přesné, ale náročné jak hardwarově tak i softwarově – prohledávání časovacích registrů

- Zásobníková implementace
  - Řešení obousměrně vázaným zásobníkem čísel referencovaných stránek
  - Při referenci přesune číslo stránky na vrchol zásobníku
  - Při určování oběti se nemusí nic prohledávat, oběť je na dně zásobníku
  - Problém
    - Přesun na vrchol zásobníku je velmi náročný, hardwarově složitý a nepružný; softwarové řešení nepřichází v úvahu kvůli rychlosti
      - Nutno dělat při každém přístupu do paměti!

Aproximace algoritmu LRU

- Příznak přístupu (*Access bit, reference bit*) – *a*-bit
  - Spojen s každou stránkou, po „*swap-in*“ = 0, při referenci rámce hardwarově nastavován na 1
  - Jako oběť se volí stránka s $a = 0$ (existuje-li).
- Algoritmus druhá šance
  - Používá *a*-bit, FIFO seznam zavedených stránek a tzv. mechanismus hodinové ručičky
    - Každá reference rámce „nastaví život“
    - Každé ukázání hodinové ručičky způsobí, že rámec „ztratí život“
    - Obětí se stane stránka, na niž ukáže hodinová ručička a rámec nemá „život“, který by mohl ztratit
  - Akce ručičky závisí na hodnotě *a*-bitu:
    - *a*=0: vezmi tuto stránku jako oběť
    - *a*=1: vynuluj *a*, ponechej stránku v paměti a posuň ručičku o pozici dále
  - Jednoduché jako FIFO, při výběru oběti se vynechává stránka aspoň jednou referencovaná od posledního výpadku
  - Numerické simulace – dobrá aproximaci čistého LRU

Modifikovaná „druhá šance“

- Algoritmus označovaný též NRU (*not recently used*)
  - Vedle *a*-bitu se používá i bit modifikace obsahu stránky (*dirty* bit, *d*-bit)
    - nastavován hardwarem při zápisu do stránky
  - Hodinová ručička maže *a*-bity
    - proto je možná i stránka s nastaveným *d*-bitem a nulovým *a*-bitem

d	***a***	**Význam**
0	0	stránka se vůbec nepoužila
0	1	ze stránky se pouze četlo
1	0	stránka má modifikovaný obsah, ale dlouho se k ní nepřistupovalo
1	1	stránka má modifikovaný obsah a byla i nedávno použita

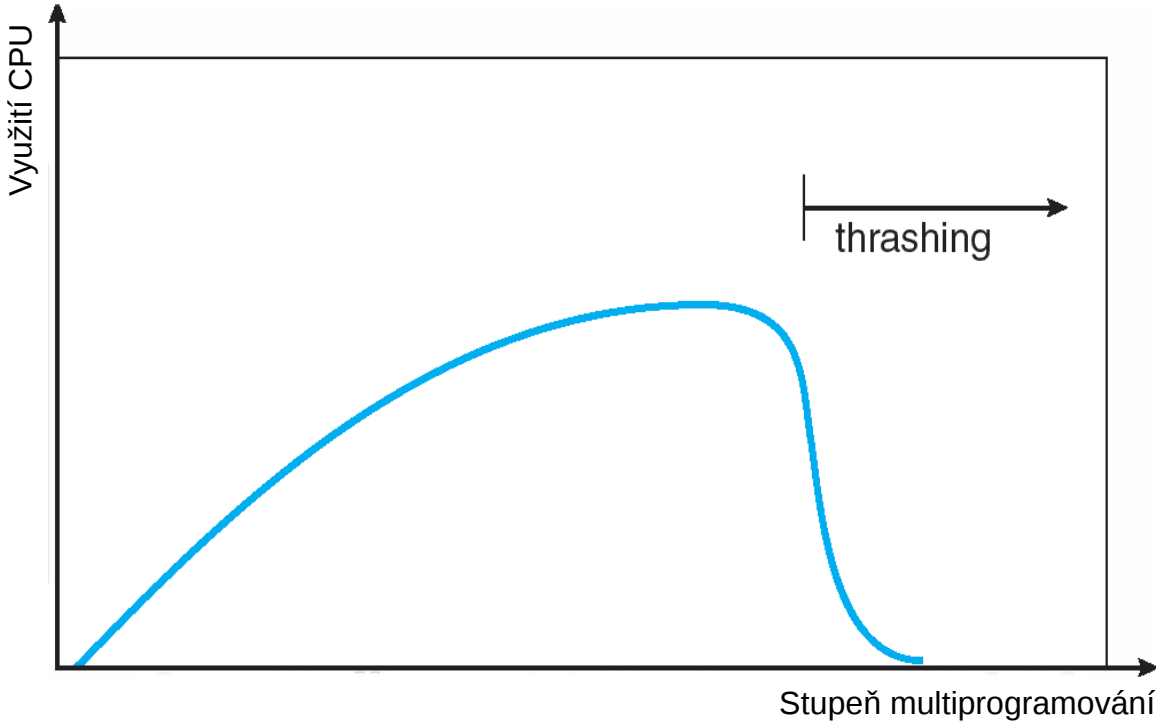
  - Pořadí výběru (*da*): 00, 01, 10, 11
    - Využití *d*-bitu šetří nutnost výpisu modifikované stránky na disk

Přidělování rámců procesům

- Obvyklé politiky
  - Pevné přidělování
    - Procesu je přidělen pevný počet rámců
      - buď zcela fixně, nebo úměrně velikosti jeho LAP
    - Podhodnocení potřebného počtu rámců => velká frekvence výpadků
    - Nadhodnocení => snížení stupně paralelismu
  - Prioritní přidělování
    - Procesy s vyšší prioritou dostanou větší počet rámců, aby běžely „rychleji“
    - Dojde-li k výpadku, je přidělen rámec patřící procesu s nižší prioritou
  - Proměnný počet rámců přidělovaných globálně (tj. z rámců dosud patřících libovolnému procesu)
    - Snadná a klasická implementace, užíváno mnoha OS (UNIXy)
    - Nebezpečí „výprasku“ (*thrashing*)
      - mnoho procesů s malým počtem přidělených rámců → mnoho výpadků
  - Proměnný počet rámců přidělovaných lokálně (tj. z rámců patřících procesu, který způsobil výpadek)
    - Metoda tzv. pracovní množiny (*working sets*) ➔

Problém výprasku, *Thrashing*

- Jestliže proces nemá v paměti dost stránek, dochází k výpadkům stránek velmi často
  - nízké využití CPU
  - OS „má dojem“, že může zvýšit stupeň multiprogramování, protože se stále se čeká na dokončení I/O operací
    - odkládání a zavádění stránek
  - Tak se dostávají do systému další procesy a situace se zhoršuje
- Thrashing – počítač nedělá nic jiného než výměny stránek

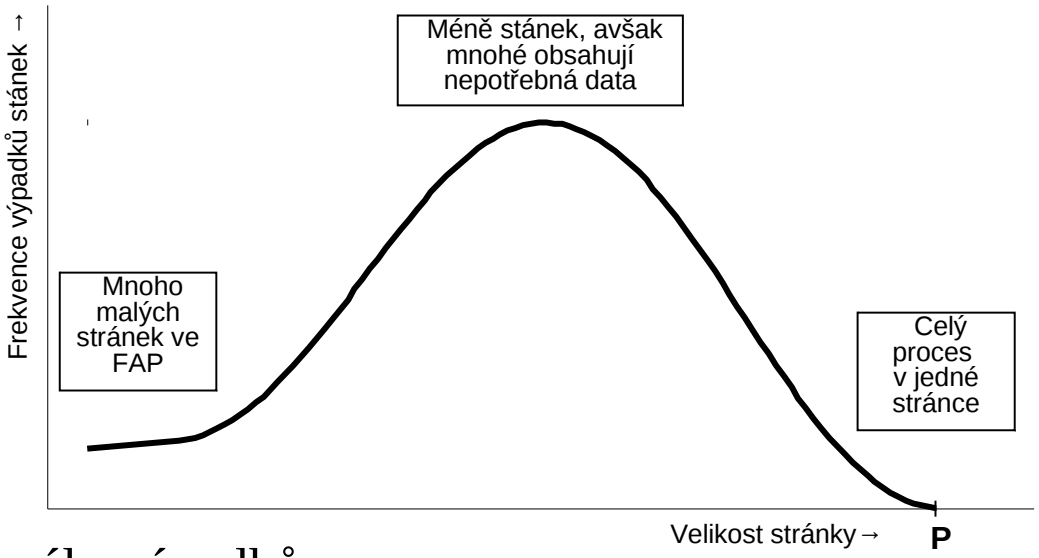


Jak reálně řídit virtuální paměť?

- Model pracovní množiny procesu P_i (*working set*) WS_i
  - Množina stránek, kterou proces referencoval při posledních n přístupech do paměti (n ~ 10.000 – tzv. okno pracovní množiny)
  - Pracovní množina je aproximace prostorové lokality procesu. Jak ji ale určovat?
    - Při každém přerušení od časovače lze např. sledovat a-bity stránek procesu, nulovat je a pamatovat si jejich předchozí hodnoty. Jestliže a-bit bude nastaven, byla stránka od posledního hodinového „tiku“ referencována a patří do WS_i
    - Časově náročné, může interferovat s algoritmem volby oběti stránky, avšak účelné a často používané
    - Pokud suma všech WS_i (počítaná přes všechny procesy) převýší kapacitu dostupné fyzické paměti, vzniká „výprask” (*thrashing*)
      - Snadná ochrana před „výpraskem” – např. jeden proces se pozastaví

Otázka velikosti stránek

- Velké stránky
  - Malý počet výpadků
  - Velká vnitřní fragmentace
  - Pokud délka stránky je větší než délka programu, vše je ve FAP a není potřeba žádná virtualizace
- Malé stránky
  - Velký počet malých stránek
    - Stránka se často najde v paměti → málo výpadků
  - Čím menší stránky, tím je
    - menší vnitřní fragmentace, avšak klesá efektivita diskových operací při výměnách stránek (mnoho přenosů malých bloků)
    - stránek více a roste potřebná velikost tabulky stránek a s tím spojená náročnost vyhledání vhodné oběti při výpadku stránky
  - Veliká tabulka stránek
    - PT trvale (neodložitelně) ve FAP – zabírá mnoho místa a zmenšuje efektivně využitelnou paměť
    - Umístění PT ve virtuální paměti způsobuje až dvojnásobný počet výpadků stránek (samotný přístup do PT může způsobit výpadek!)



Způsob programování a výpadky

- Technika programování aplikací může významně ovlivnit efektivitu

  double data[512][512]; (pole 512x512 prvků)
  - Předpokládáme, že double zabírá 8 bytů
  - Každý řádek pole zabírá 4 KB a je uložen v jedné stránce velké 4 KB

Postup 1:

```
for (j = 0; j <512; j++)
        for (i = 0; i < 512; i++)
                data[i][j] = i*j;
```

Potenciálně až
512 x 512 = 262 144
výpadků

Postup 2:

```
for (i = 0; i <512; i++)
        for (j = 0; j < 512; j++)
                data[i][j] = i*j;
```

Jen 512 potenciálních
výpadků

Je tedy dobře vědět, jak se data ukládají v paměti a účelně k nim přistupovat

- Metoda pracovních množin tento problém redukuje, ale lze tomu pomoci
- Některé překladače to udělají za programátora v rámci optimalizace kódu

Stránkování ve Windows XP

- Stránkování na žádost s použitím „*prepaging*“
  - do paměti se zavádí chybějící stránka a stránky okolní
- Používá se technika pracovních množin
  - Z měření WS se určuje minimální počet stránek, které musí mít proces ve FAP
  - Klesne-li objem volné paměti v systému pod jistý práh, automaticky se přehodnotí WS s cílem obnovit dostatečný objem volné paměti
    - Z FAP se odstraňují stránky procesům, které mají v hlavní paměti více než minimum určené metodou WS
- Přesto se v praxi setkáváme u Windows XP s nedostatkem paměti – „výpraskem“
  - Doporučené minimum fyzické paměti – 128 MB
  - Reálně použitelné minimum – 512 MB

To je dnes vše.

Otázky?