Správa paměti

Tomáš Vojnar
vojnar@fit.vutbr.cz

Vysoké učení technické v Brně
Fakulta informačních technologií
Božetěchova 2, 612 66 BRNO

22. dubna 2020

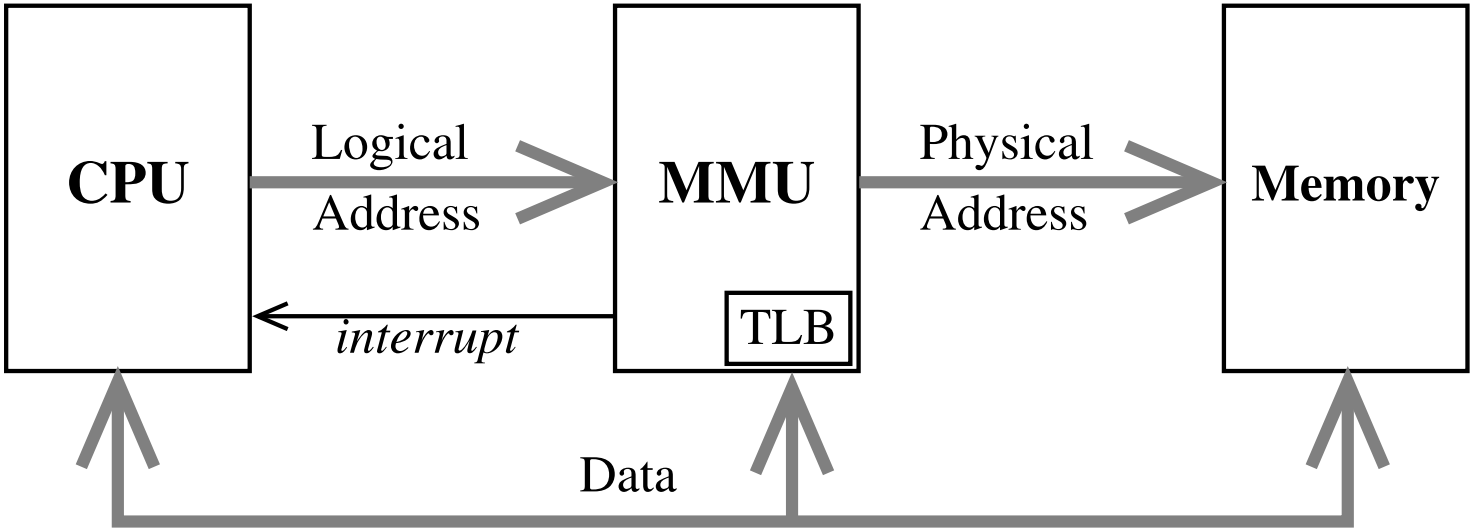
Správa paměti

– Aby program mohl být proveden, musí nad ním být vytvořen proces, musí mu být přidělen procesor a musí mu být přidělena paměť (a případně další zdroje).

– Rozlišujeme:

- **logický adresový prostor** (LAP): virtuální adresový prostor, se kterým pracuje procesor při provádění kódu (každý proces i jádro mají svůj),
- **fyzický adresový prostor** (FAP): adresový prostor fyzických adres paměti (společný pro všechny procesy i jádro).

– **MMU (*Memory Management Unit*)** = HW jednotka pro překlad logických adres na fyzické, běžně součást čipu procesoru:



– MMU využívá speciálních registrů a případě i hlavní paměti systému; pro urychlení překladu může obsahovat různé vyrovnávací paměti (např. TLB).

Přidělování paměti

– Na nejnižší úrovni z hlediska blízkosti HW se můžeme v jádře setkat s přidělováním FAP pro zamapování do LAP, které je konzistentní se způsobem překladu LAP na FAP, jenž je podporován HW daného výpočetního systému:

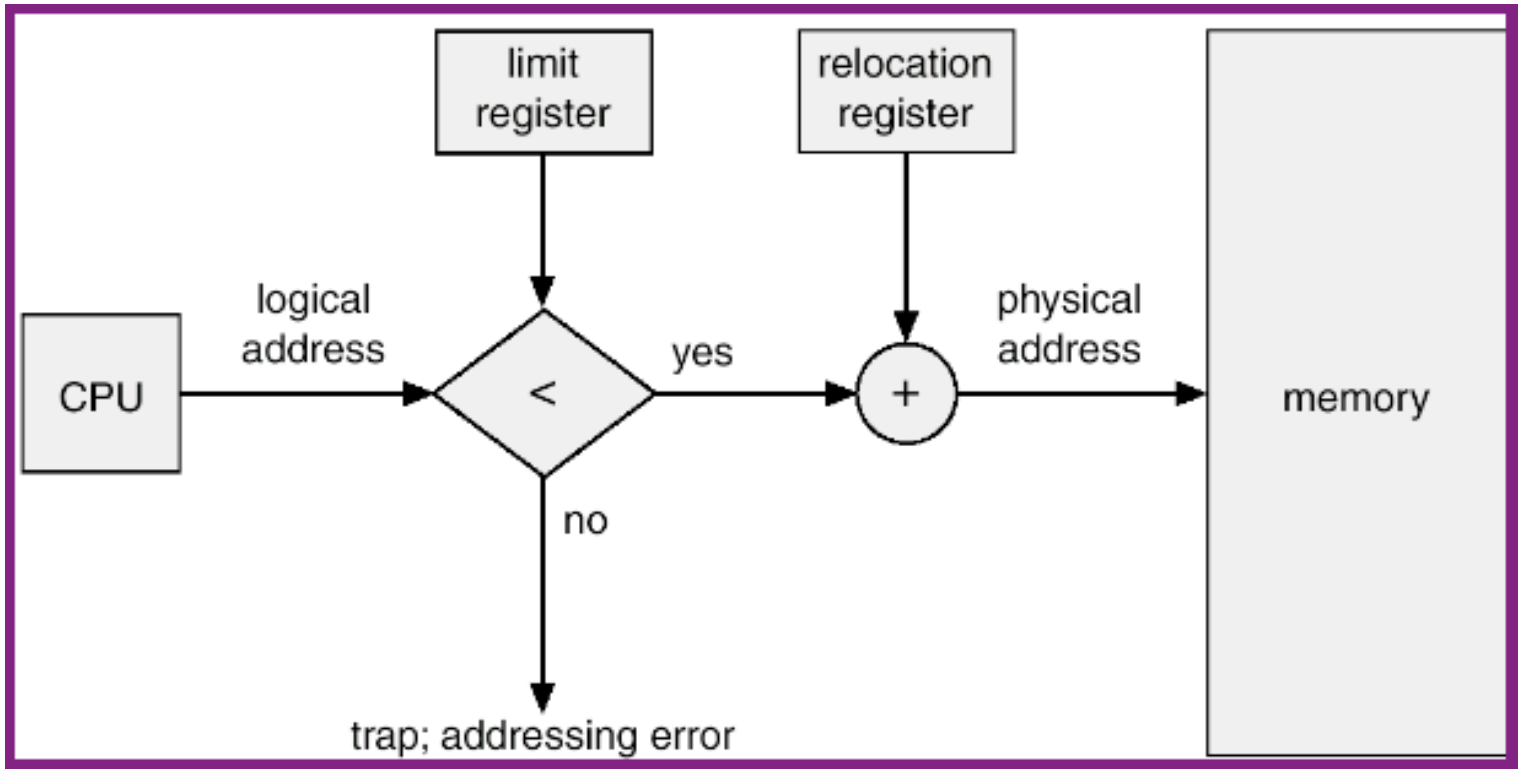
- spojité bloky (*contiguous memory allocation*),
- segmenty,
- stránky,
- kombinace výše uvedeného.

– Na vyšší úrovni se pak používá přidělování LAP pro konkrétní potřeby uživatelských procesů (`malloc`, ... – implementováno mimo režim jádra) i pro běžnou potřebu v jádře (`kmalloc`, `vmalloc`, ...), a to v rámci bloků LAP již zamapovaných do přidělených úseků FAP.

Contiguous Memory Allocation

– Procesům jsou přidělovány spojité bloky paměti určité velikosti.

– Snadná implementace (jak v HW, tak obsluha v OS):



– Významně se projevuje **externí fragmentace** paměti (FAP):

- přidělováním a uvolňováním paměti vzniká posloupnost obsazených a neobsazených úseků paměti různé velikosti způsobující, že volné místo může být nevyužitelné, protože je nespojité,
- minimalizace pomocí různých strategií alokace paměti (přičemž je třeba brát do úvahy také režii spojenou s daným přidělováním) – mimo *first fit* lze užít např. *best fit*, *worst fit*, *binary buddy*, ...,
- problém se zvětšováním přiděleného prostoru,
- dynamická reorganizace paměti (nákladné!).

– Při nedostatku paměti nutno odkládat na disk veškerou paměť procesu: může být zbytečné, pomalé.

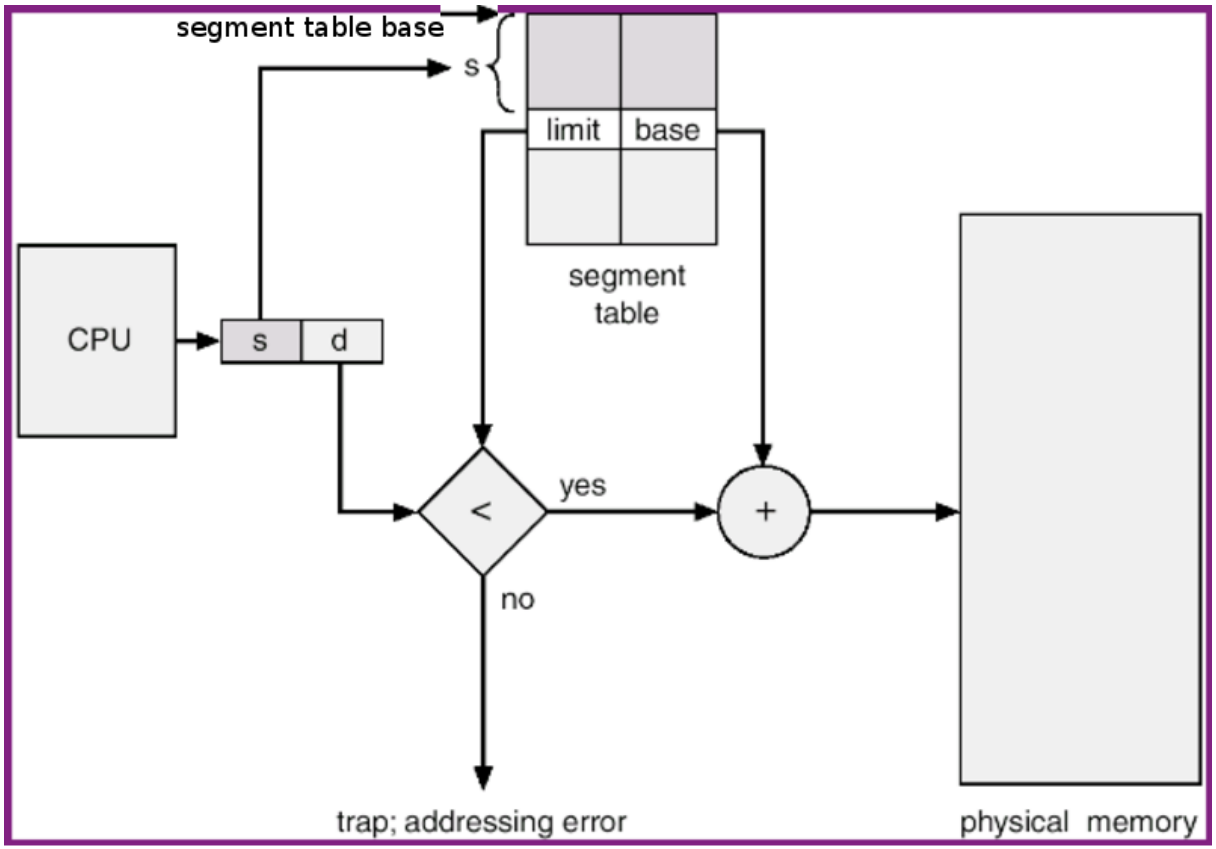
– Není možné jemně řídit přístupová práva, není možné sdílet části paměti.

– Nemusí způsobit **interní fragmentaci**, ve strukturách popisujících obsazení paměti je pak ale nutné pracovat s úplnými adresami. Pro usnadnění evidence přidělené/volné paměti a odstranění možnosti vzniku velmi malých neobsazených úseků se může přidělovat v násobcích určitých bloků, což způsobí interní fragmentaci (toleruje se, vzniká i u ostatních mechanismů přidělování paměti).

Segmentace paměti

– LAP rozdělen na kolekci segmentů. Různé segmenty mohou být přiděleny překladačem (programátorem) jednotlivým částem procesu (např. procedurám, částem dat, zásobníku, ...).

– Každý segment má číslo a velikost; logická adresa sestává z čísla segmentu a posunu v něm:



– Segmenty mohou být využity jako jednotka ochrany, odkládání a/nebo sdílení paměti (segmenty jen pro čtení, sdílené segmenty, ...).

– Implementace je stále poměrně jednoduchá.

– Paměť přidělována po segmentech; zmírnění dopadů externí fragmentace a jemnější odkládání než u přidělování jediné oblasti – ale problém přetrvává.

– Jemnější řízení přístupu a sdílení.

– Segmentace je viditelná procesu: komplikace při překladu (tvorbě programů), možnost chyb.

Stránkování

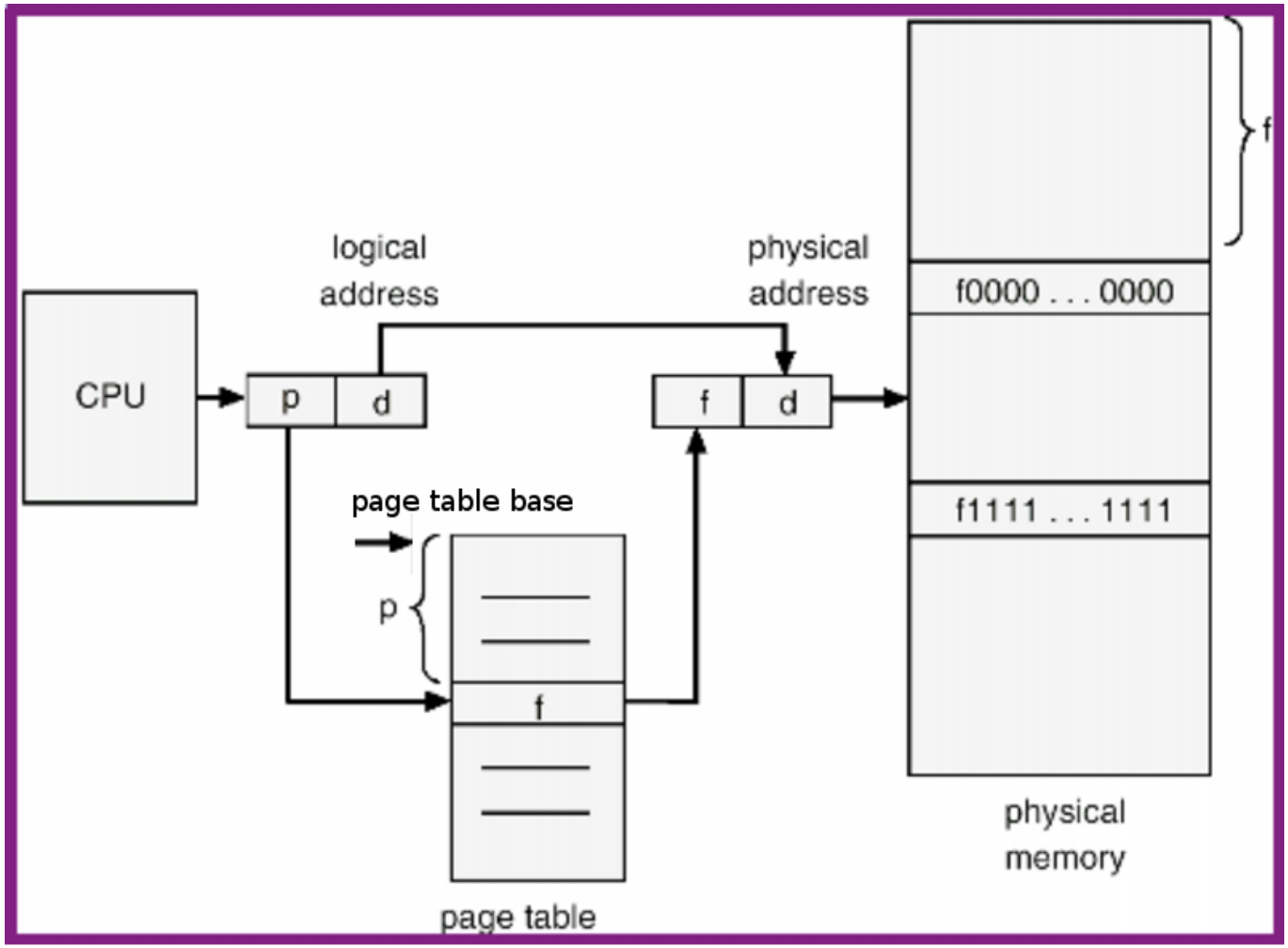
– LAP rozdělen na jednotky pevné velikosti: **stránky** (*pages*).

– FAP rozdělen na jednotky stejné velikosti: **rámce** (*frames*).

– Vlastnosti:

- paměť přidělována po rámcích,
- neviditelné pro uživatelské procesy,
- minimalizovány problémy s externí fragmentací,
  - nevzniká nevyužitelný volný prostor,
  - možné snížení rychlosti přístupu do paměti (větší počet kolizí v různých vyrovnávacích pamětech) a alokace/dealokace (delší práce se strukturami popisujícími aktuální obsah paměti),
  - proto v praxi je snaha přidělovat paměť pokud možno po spojitých posloupnostech rámců: např. pomocí algoritmu „binary buddy“,
- jemná jednotka ochrany (r/rw, user/system, možnost provádění – tzv. NX bit) a/nebo sdílení,
- jemná kontrola odkládání po stránkách,
- složitější implementace, větší režie,
- interní fragmentace.

– V nejjednodušším případě tzv. jednoduchých (či jednoúrovňových) tabulek stránek, OS udržuje **informaci o volných rámcích** a pro každý proces (a jádro) **tabulku stránek** (*page table*):



– Tabulka stránek obsahuje popis mapování logických stránek do FAP a příznaky platnosti mapování, přístupu, modifikace, přístupová práva (r/rw, user/system, možnost provádění), příznak globality (neodstraňováno automaticky z TLB při přepnutí kontextu), ...

– Tabulky stránek jsou udržovány v hlavní paměti; speciální registr MMU (CR3 u x86) obsahuje adresu začátku tabulky stránek pro aktuální proces.

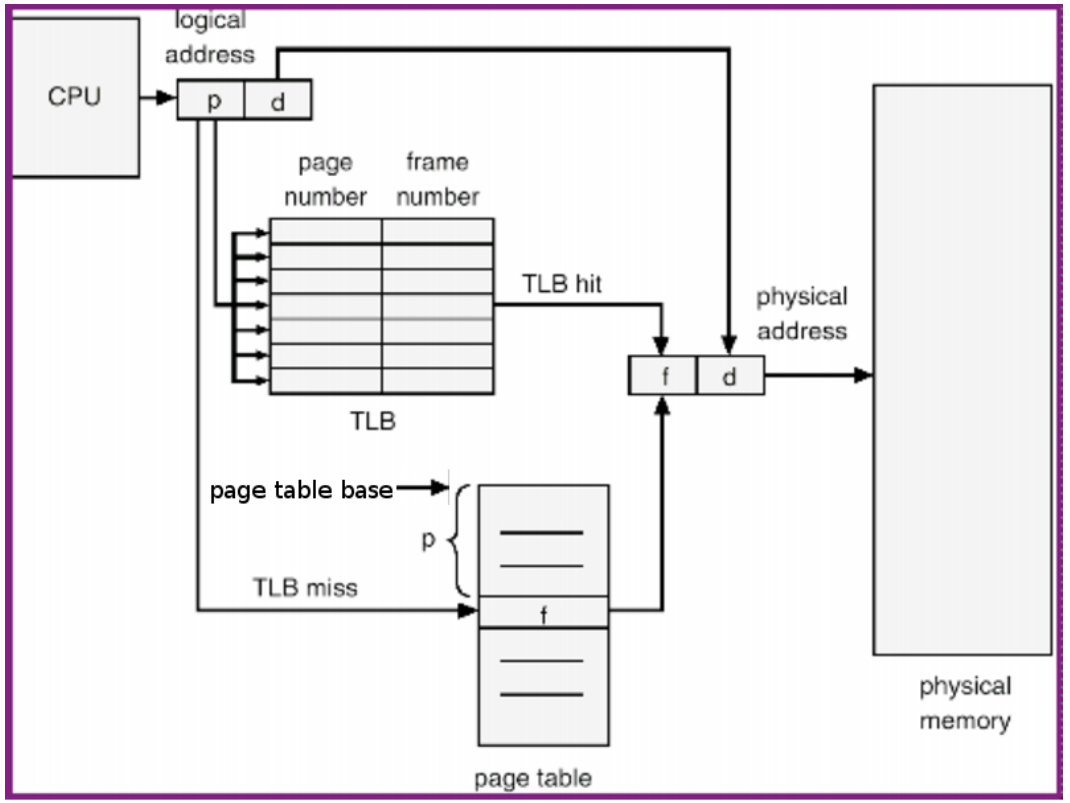
– Každý odkaz na data/instrukce v paměti vyžaduje (u jednoduché tabulky stránek) dva přístupy do paměti: do tabulky stránek a na vlastní data/instrukci.

– Urychlení pomocí rychlé hardwarové asociativní vyrovnávací paměti **TLB (*Translation Look-aside Buffer*)**.

– TLB obsahuje dvojice (číslo stránky,číslo rámce) + některé z příznaků spojených s daným mapováním v tabulkách stránek (přístupová oprávnění, příznak modifikace, příp. další).

– POZOR! V TLB nejsou celé stránky či rámce!

– TLB se prohledává paralelně na základě čísla stránky, a to buď plně (viz obrázek níže), nebo částečně (dojde k indexaci skupiny buněk TLB dle části čísla stránky a pak k paralelnímu dohledání – např. by mohly být užity 2 bity z čísla stránky k rozlišení 4 množin dvojic stránek prohledávaných již tak, jak je znázorněno níže).



– POZOR! K TLB miss může dojít jak při čtení instrukce, tak při čtení jejich operandů, a to u instrukce i operandů i vícenásobně (nezarovnaná instrukce, nezarovnaná data, data delší než jedna stránka).

– Po TLB miss:

- U HW řízených TLB HW automaticky hledá v tabulce stránek.
- U SW řízených TLB (např. MIPS, SPARC) musí v tabulce stránek hledat jádro a patřičně upravit obsah TLB.

– Někdy může být použito více TLB: zvlášť pro stránky obsahující kód a data a/nebo hierarchie více úrovní TLB (různá rychlost, kapacita, cena, spotřeba).

– Při přepnutí kontextu nutno obsah TLB invalidovat. Optimalizace:

- použití globálních stránek označených zvláštním příznakem v tabulce stránek a TLB či
- spojení záznamu v TLB s identifikací procesu (např. PCID u Intelu – viz dále).

– Invalidace TLB nutná samořejmě i po změně obsahu tabulek stránek. Může-li ovlivněné záznamy používat více procesorů, nutno invalidovat TLB na všech procesorech.

– Některé procesory mohou dopředu nahrávat do TLB překlad pro odhadované dále prováděné instrukce.

– **Efektivnost stránkování** velmi silně závisí na úspěšnosti TLB:

- Efektivní přístupová doba: $(\tau+\varepsilon)\alpha+(2\tau+\varepsilon)(1-\alpha)$, kde
  - τ: vybavovací doba RAM
  - ε: vybavovací doba TLB
  - α: pravděpodobnost úspěšných vyhledání v TLB (*TLB hit ratio*)
- Např. pro $\tau = 100ns$, $\varepsilon = 20ns$ a $\alpha = 0.98$ dostaneme průměrné zpomalení o 22%.
- TLB hit ratio významně závisí na lokalitě odkazů programu.

– Výše uvedený vztah je sestaven za předpokladu, že po TLB miss a překladu přes tabulky stránek se získaná adresa ihned použije. V praxi se často získaný překlad nejprve vloží do TLB a pak se opakuje překlad přes TLB – pak je ve výše uvedeném vztahu nutno při TLB miss připočíst další přístup do TLB a také čas pro úpravu TLB.

– **Lokalita odkazů** = vlastnost programu – míra toho, kolik různých shluků adres (odpovídajících typicky adresám v různých stránkách) bude proces potřebovat v krátkém časovém úseku.

Příklad:

– Při ukládání matic po řádcích je

```
for (j = 0; j < MAX; j++)
  for (i = 0; i < MAX; i++)
    A[i,j] = 0;
```

mnohem méně výhodné než

```
for (i = 0; i < MAX; i++)
  for (j = 0; j < MAX; j++)
    A[i,j] = 0;
```

– Rozdíl se projeví ještě výrazněji při virtualizaci paměti a odkládání stránek na disk.

Implementace tabulek stránek

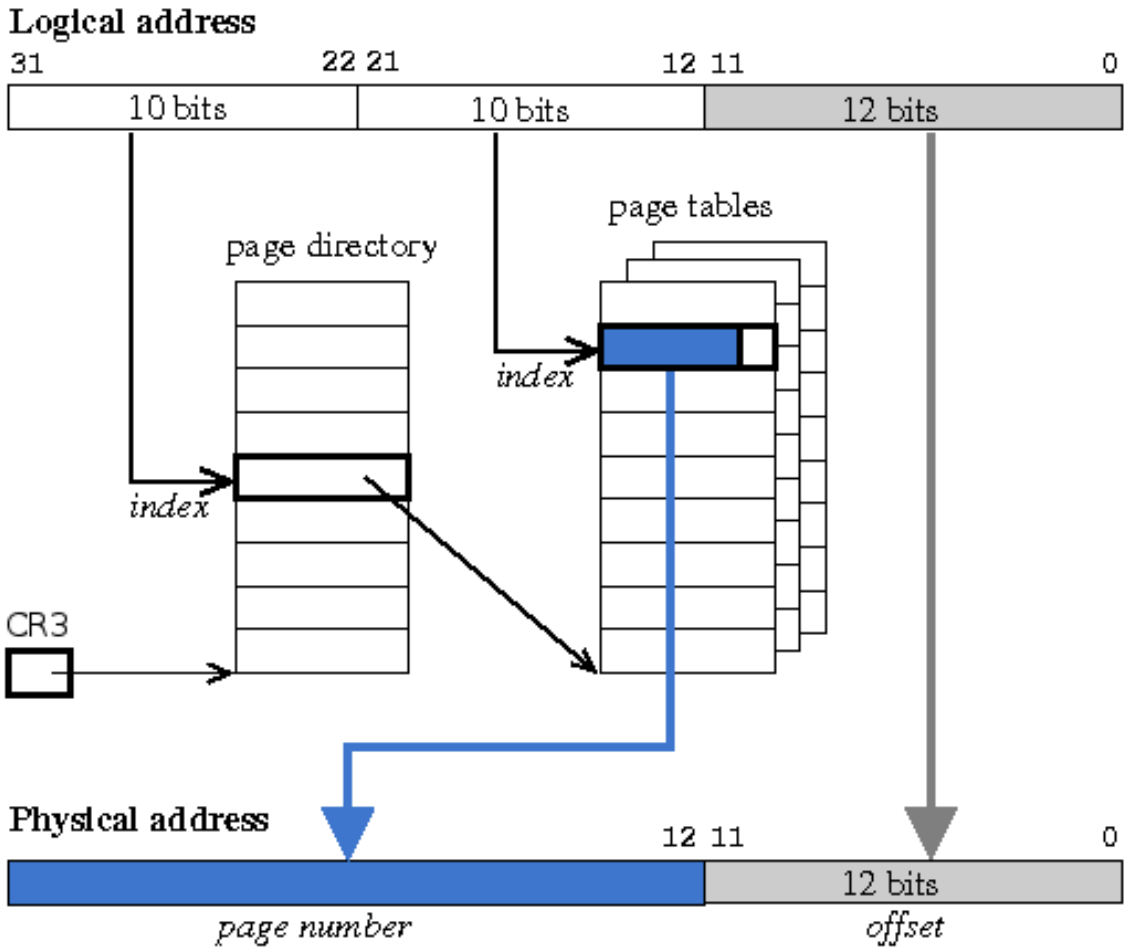
– Tabulky stránek mohou být značně rozsáhlé.

– Pro 32b systém se stránkami o velikosti 4KiB (2^{12}) má tabulka více než milión položek (přesně $2^{32}/2^{12} = 2^{20} = 1,048,576$ položek). Má-li položka tabulky stránek 4B, dostaneme 4MiB na tabulku stránek pro každý proces, což je příliš na spojitou alokaci (pro 100 procesů 400MiB jen pro tabulky stránek!).

– Pro 64b systémy problém exponenciálně roste: jednoúrovňová tabulka pro 4KiB stránky by měla $2^{64}/2^{12} = 2^{52} = 4,503,599,627,370,496$ položek).

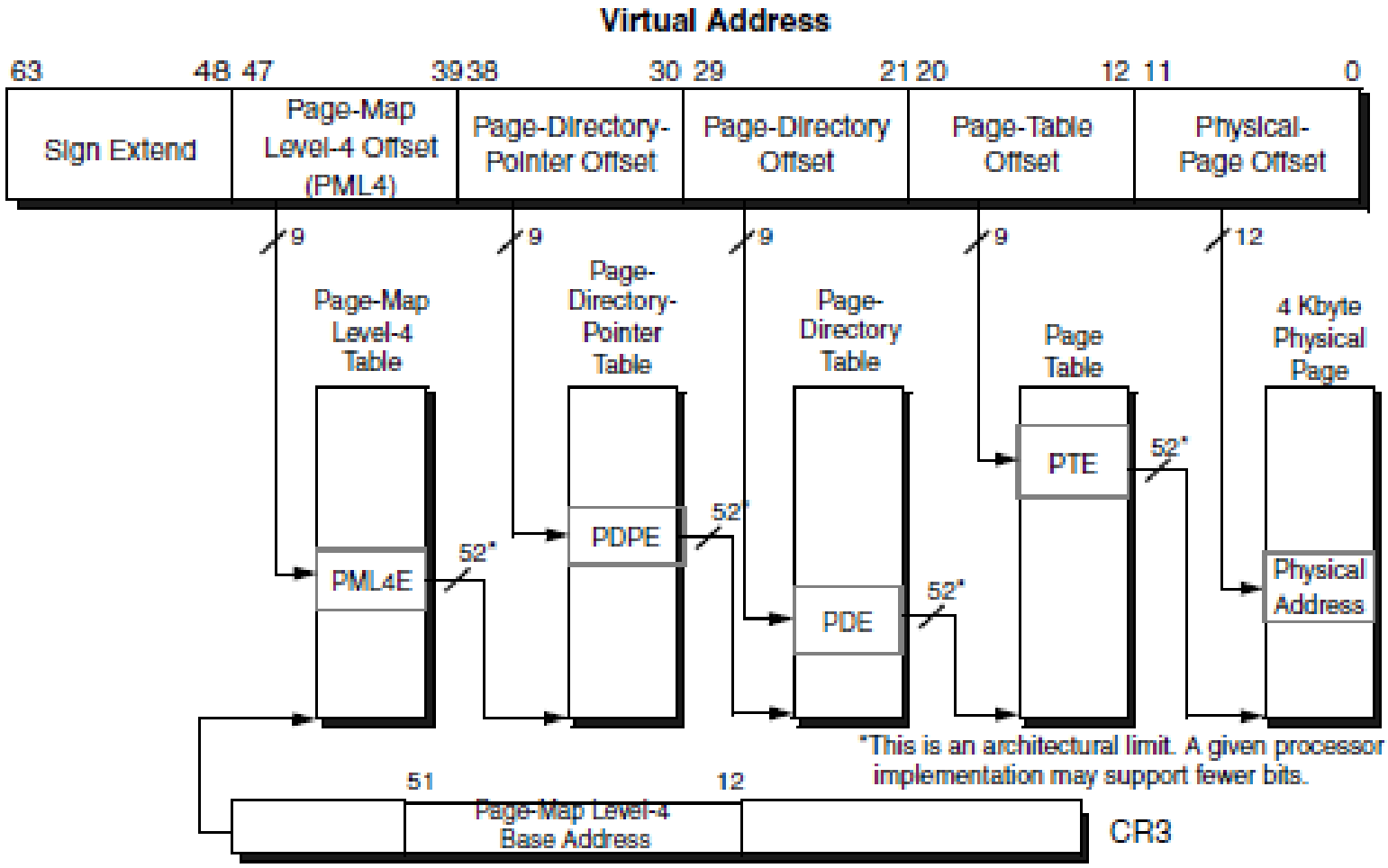
– **Hierarchické tabulky stránek**: tabulka stránek je sama stránkována, vznikají tabulky tabulek stránek, ... (což ovšem dále zpomaluje přístup).

– **Příklad:** dvouúrovňová tabulka stránek procesorů i386+



– Příznak v položkách adresáře stránek určuje, zda je použita i nižší úroveň stránkování. Tímto způsobem je možno pracovat se stránkami o velikosti 4 KiB (2^{12} B) a 4 MiB (2^{22} B).

– 4-úrovňová **tabulka stránek na x86-64**:



– Příznak v položkách tabulek PDPE a PDE určuje, zda je použita i nižší úroveň stránkování. Tímto způsobem je možno pracovat se stránkami o velikosti 4 KiB (2^{12} B), 2 MiB (2^{21} B) a u některých procesorů i 1 GiB (2^{30} B).

– U hierarchických tabulek stránek dále roste zpoždění přístupu do paměti a **roste význam TLB**:

- větší velikost, složitější organizace – více úrovní, oddělení TLB pro překlad adres dat a kódu,
- např. Intel Core i7: zvlášť datová a instrukční TLB 1. úrovně, společná TLB 2. úrovně.

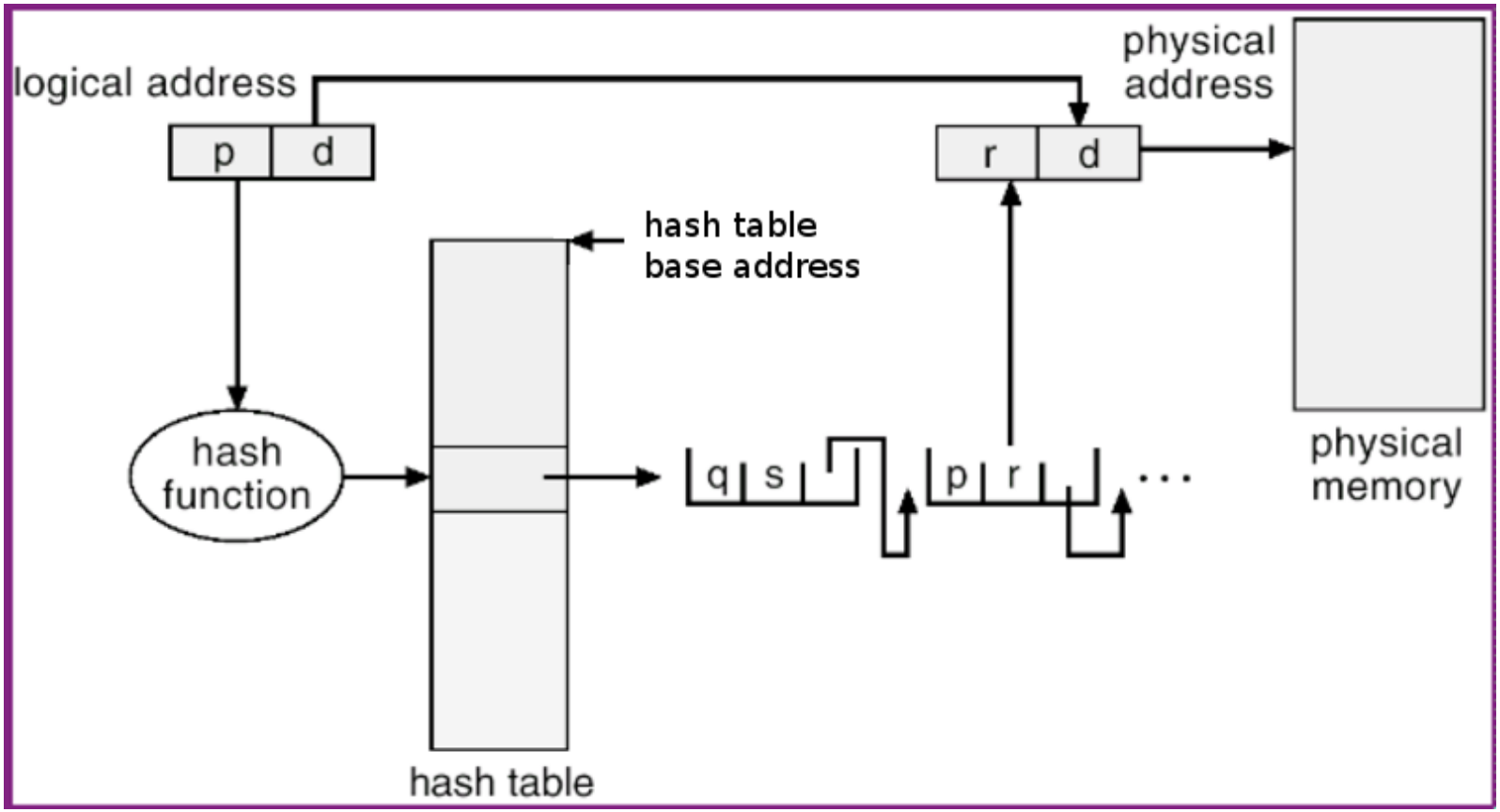
– Další možnosti optimalizace práce s TLB:

- **globální stránky** – záznamy o nich se neodstraňují z TLB při přepnutí kontextu (lze užít např. u stránek jádra, pokud LA jádra je podprostorem LA každého procesu, a to ve fixní části těchto LA),
- **vstupy TLB spojené s identifikátory procesů** – opět není nutno vždy odstraňovat z TLB (např. PCID – process-context ID u procesorů Intel),
- **spekulativní dopředné nahrávání překladu do TLB**,
- využití **specializovaných cache** (mimo TLB) pro ukládání položek (některých úrovní) tabulek stránek.

– **Zanořené hierarchické tabulky stránek** (Intel/AMD):

- podpora virtualizace HW – jedna úroveň 4-úrovňových hierarchických tabulek stránek pro virtuální stroje, další pro překlad „fyzických“ adresových prostorů těchto strojů na opravdu fyzické adresy;
- používá příznak globality stránek a identifikaci virtuálního stroje (Intel: VPID – virtual processor ID, AMD: ASID – adress space ID) v položce TLB pro minimalizaci počtu záznamů TLB invalidovaných při přepnutí kontextu.

– **Hashované tabulky stránek**:



– V překladových položkách ve zřetězeném seznamu může a nemusí být celé číslo stránky (nemusí tam být celé, pokud hash funkce některé bity čísla stránky spolehlivě odliší – např. pokud nikdy nebudou kolidovat čísla stránek s odlišnými n dolními bity apod.).

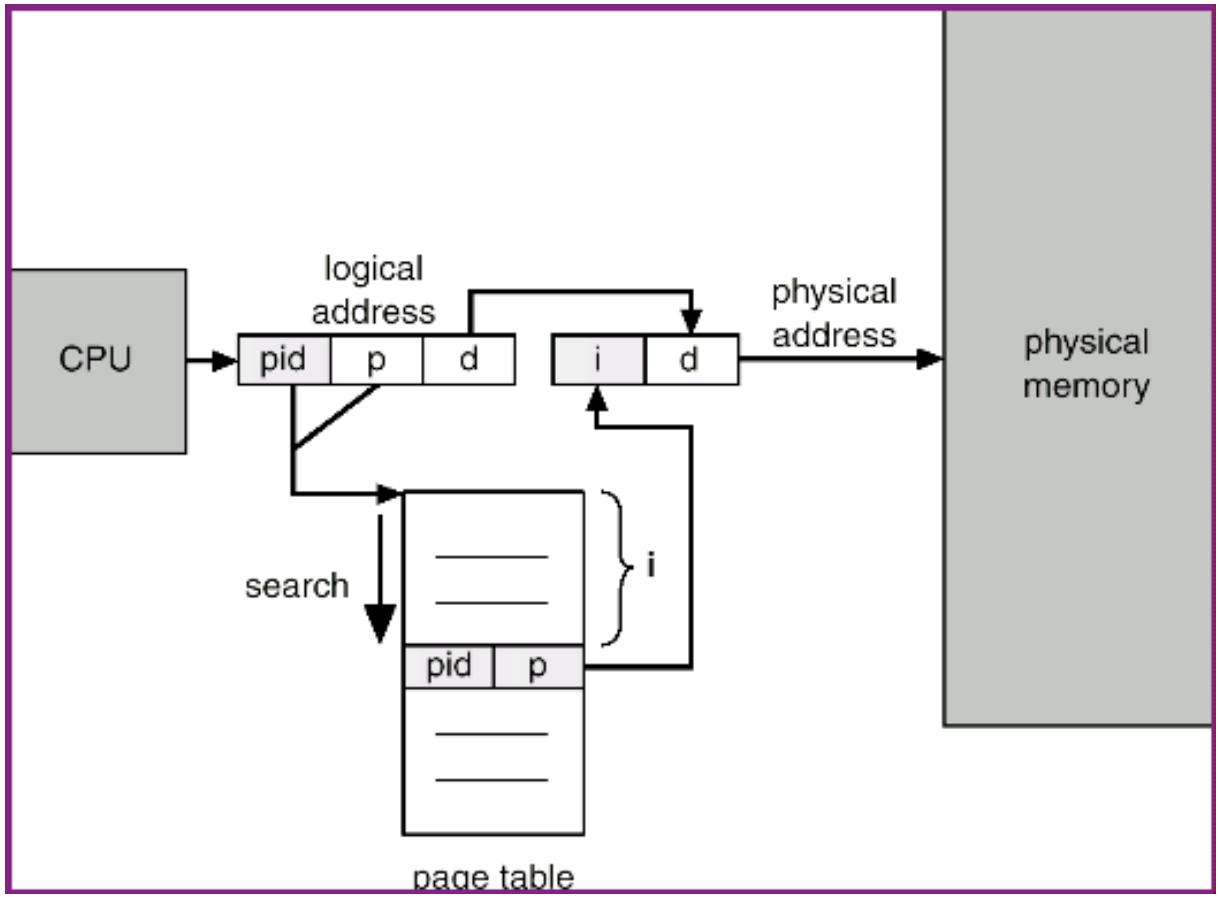
– Obecný zřetězený seznam překladových položek se stejnou hodnotou hash funkce může být nahrazen **fixním počtem překladových položek** ukládaných pro danou hodnotu hash funkce.

- Pokud v takovém případě překlad není nalezen, neznamená to, že stránka není mapována – jádro musí na základě svých pomocných struktur v paměti (čistě programových tabulek stránek, které si jádro musí vést) dohledat překlad a doplnit ho do hashované tabulky stránek namísto některé jiné překladové dvojice.
- Užívá se např. u procesorů PowerPC (několik překladových položek pro jednu hodnotu hash funkce) či Itanium (překladové položky mohou být zřetězeny do seznamu, ale HW tento seznam neprochází, podívá se jen na první položku, zbytek je v režii jádra).

– Hashovaná tabulka stránek **může být sdílená všemi procesy** (čímž se blíží invertované tabulce stránek zmíněné dále: ale nemá jeden řádek pro každý rámec).

- V takovém případě se v překladových položkách mimo čísla stránky a čísla rámce udržuje i číslo procesu (nebo jeho část).
- Namísto čísla procesu zde také může být číslo paměťového regionu (nebo jeho část): LA procesů je rozdělen na regiony, čísla regionů lokální v rámci procesů se převádí pomocí další specializované tabulky na globální čísla regionů (některé regiony mohou takto být sdíleny). Pro hashování se pak používá globální číslo regionu a číslo stránky.
- Tento mechanismus je užit u procesorů PowerPC i Itanium.

– **Invertovaná tabulka stránek** – jediná tabulka udávající pro každý rámec, který proces má do něj namapovánu kterou stránku:



– Možno kombinovat s hashováním (např. IBM RS/6000), kdy hashovací funkce spočte počáteční odkaz do invertované tabulky a její položky pak mohou být zřetězeny do seznamu (díky tomu, že index položek invertované tabulky odpovídá číslu rámce, zde ale není ukládáno explicitně číslo rámce).

– OS si pro potřeby správy paměti (odkládání na disk apod.) vede dále i klasické tabulky stránek. Problematická je implementace sdílení stránek (v daném okamžiku je k dispozici mapování jen pro jeden proces – možno řešit mapováním nikoliv pro procesy, ale pro čísla regionů).

Stránkování a segmentace na žádost

– **Virtualizace paměti** umožňuje procesům (a jádru) pracovat s oddělenými lineárními logickými adresovými prostory.

– V případě mapování mezi LAP a FAP pomocí stránek či segmentů je navíc možné zajistit, aby ne všechen využitý LAP byl celý umístěn ve fyzické paměti.

– **Výhodou** je menší spotřeba paměti, rychlejší odkládání na disk a zavádění do paměti (není zapotřebí odložit nebo zavést celý využitý adresový prostor procesu).

– Pro uložení části LAP, které aktuálně nejsou ve FAP, se využívá prostor na disku.

– Z disku se příslušné části LAP zavádí do FAP pouze tehdy, je-li to zapotřebí (nebo v rámci různých optimalizací i spekulativně těsně před jejich odhadovaným použitím).

– Hovoříme pak o:

- **stránkování na žádost** (*demand paging*) a
- **segmentování na žádost** (*demand segmenting*).

Stránkování na žádost

– Stránky jsou zaváděny do paměti jen tehdy, jsou-li zapotřebí (příp. jsou v rámci optimalizací spekulativně zaváděny v předstihu).

– Informace o odložených stránkách, příp. stránkách naalokovaných, ale ještě nenačtených si jádro vede ve svých pomocných strukturách – nejsou uloženy v tabulkách stránek, se kterými pracuje MMU (!).

– Stránka je zapotřebí tehdy, dojde-li k odkazu na ni. V jedno- či víceúrovňové tabulce stránek je příznak, zda příslušné stránce je přidělen rámec. Pokud ne, dojde k **výpadku stránky** (*page fault*).

– U hashované či invertované tabulky se to, že stránce není přidělen rámec, pozná z neúspěšného prohledání příslušného seznamu stránek se stejnou hodnotou hashovací funkce či celé tabulky.

– Výpadek stránky je přerušení od MMU udávající, že nelze převést adresu (není definováno mapování v tabulce stránek).

Obsluha výpadku stránky

– Typická obsluha výpadku stránky v jádře vypadá takto:

1. Kontrola, zda se proces neodkazuje mimo přidělený adresový prostor.
2. Alokace rámce:
   - použijeme volný rámec, pokud nějaký volný je,
   - pokud není,
     - vybereme vhodnou stránku s přiděleným rámcem (*victim page*),
     - byla-li stránka změněna, což udává příznak modifikace v tab. stránek, je odložena na swap,
     - použijeme uvolněný rámec.
3. Inicializace stránky po alokaci závislá na předchozím stavu stránky:
   - první odkaz na stránku:
     kód: načtení z programu,
     inicializovaná data: načtení z programu,
     vše ostatní: vynulování (nelze ponechat původní obsah – bezpečnost),
   - stránka byla v minulosti uvolněna z FAP:
     kód: znovu načte z programu (jen pokud nelze přepisovat kódové stránky),
     konstantní data: totéž co kód,
     ostatní: pokud byla modifikována, je stránka ve swapu a musí se načíst zpět do FAP, jinak je obsah opět vynulován
4. Úprava tabulky stránek: namapování zpřístupňované stránky na přidělený rámec.
5. Proces je připraven k opakování instrukce, která výpadek způsobila (je ve stavu „připravený“).

Výkonnost stránkování na žádost

– Efektivní doba přístupu do paměti: $(1-p)T + pD$, kde

- p: *page fault rate* = pravděpodobnost výpadku stránky,
- T: doba přístupu bez výpadku,
- D: doba přístupu s výpadkem.

– Vzhledem k tomu, že $T << D$, musí být p co nejmenší:

- dostatek paměti a jemu přiměřený počet procesů (s ohledem na jejich paměťové nároky),
- vhodný výběr zaváděných a odkládaných stránek,
- lokalita odkazů v procesech.

Počet výpadků

– K výpadkům může dojít jak při čtení instrukce, tak při práci s každým z jejich operandů, a to u instrukce i u každého z operandů i vícenásobně.

– Vícenásobné výpadky mohou být způsobeny:

- nezarovnáním instrukce,
- nezarovnáním dat,
- daty delšími než jedna stránka,
- výpadky tabulek stránek různých úrovní – a to i vícekrát na stejné úrovni hierarchických tabulek stránek při dotazech na různé dílčí tabulky stránek nacházející se na stejné úrovni.
  - Obvykle alespoň část tabulek stránek je chráněna před výpadkem stránek (zejména se to týká u hierarchických tabulek stránek dílčí tabulky nejvyšší úrovně) mj. proto, aby bylo možno obsloužit výpadky stránek.

– **Příklad**: Jaký je maximální počet výpadků stránek v systému se stránkami o velikosti 4 KiB, 4-úrovňovou tabulkou stránek, u které pouze dílčí tabulka nejvyšší úrovně je chráněná proti výpadku, při provádění předem nenačtené instrukce o délce 4 B, která přesouvá 8 KiB z jedné adresy paměti na jinou?

Odkládání stránek

– K odložení stránky může dojít při výpadku stránky. Může být provedeno odložení:

- **lokální** – tj. v rámci procesu, u kterého došlo k výpadku
  - je zapotřebí vhodný algoritmus alokace rámců pro použití procesy,
- **globální** – tj. bez ohledu na to, kterému procesu patří která stránka.

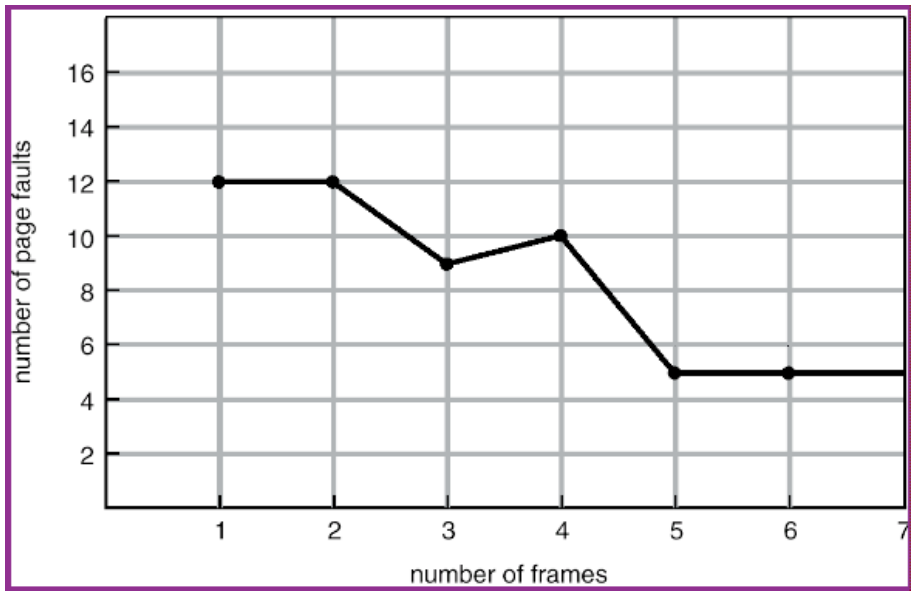
– Typicky je ale neustále udržován určitý počet volných rámců:

- Pokud počet volných rámců klesne pod určitou mez, aktivuje se **page daemon** („zloděj stránek“), který běží tak dlouho, dokud neuvolní dostatečný počet stránek (příp. paměť uvolňuje po částech a dává prostor k běhu ostatním procesům).
- Při výpadku stránky se pak použije rámec z množiny volných rámců.
- Lze doplnit heuristikou, kdy čerstvě uvolněné stránky se okamžitě nepřidělí a zjistí-li se, že byla zvolena nesprávná oběť, lze je snadno vrátit příslušnému procesu k použití.

Algoritmy výběru odkládaných stránek

– **FIFO**:

- Odstraňuje stránku, která byla zavedena do paměti před nejdelší dobou a dosud nebyla odstraněna.
- Jednoduchá implementace.
- Může odstranit „starou“, ale stále často používanou stránku.
- Trpí tzv. Beladyho anomálií – více výpadků při zvětšení paměti.



- Lze užít v kombinací s přemístěním uvolněného rámce do množiny volných rámců, přidělením jiného volného rámce a možností ihned získat zpět právě uvolněný rámec při následném výpadku signalizujícím, že byla zvolena nesprávná „oběť“.

– **LRU (*Least Recently Used*)**:

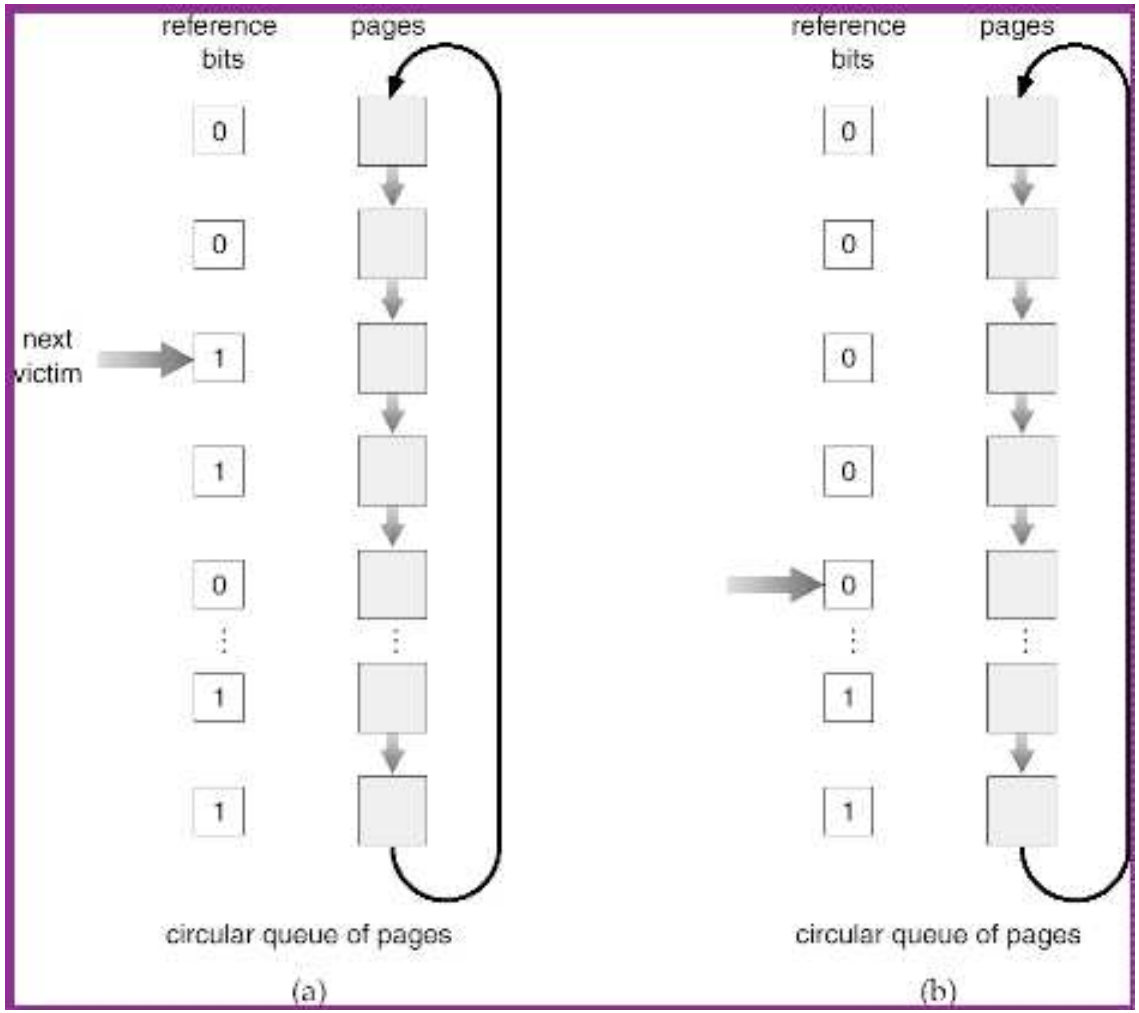
- Odkládá nejdéle nepoužitou stránku.
- Velmi dobrá aproximace hypotetického ideálního algoritmu (tj. algoritmu, který by znal budoucnost a podle budoucích požadavků rozhodoval, co aktuálně odložit tak, aby počet výpadků byl v budoucnu minimální).
  - Někdy se uvádí problémy s cyklickými průchody rozsáhlými poli, spolu se snahou takové přístupy detekovat a řešit zvlášť např. strategií odstranění naposledy použité stránky (most recently used – MRU).
- Problematická implementace vyžadující výraznou HW podporu (označování stránek časovým razítkem posledního přístupu či udržování zásobníku stránek, jehož vrcholem je naposledy použitá stránka).
- Používají se aproximace LRU.

– **Aproximace LRU pomocí omezené historie referenčního bitu stránek** (page aging):

- Referenčí bit stránky je HW nastaven při každém přístupu,
- jádro si vede omezenou historii tohoto bitu pro jednotlivé stránky,
- periodicky posouvá obsah historie doprava,
- na nejlevější pozici uloží aktuální hodnotu referenčního bitu a vynuluje ho,
- oběť je vybrána jako stránka s nejnižší číselnou hodnotou historie.
  - Ukládáme-li 4 bity historie a máme stránky s historií 0110 a 1100, odstraníme první z nich (nejlevější bit je poslední reference).

– **Aproximace LRU algoritmem druhé šance**:

- Stránky v kruhovém seznamu, postupujeme a nulujeme referenční bit, odstraníme první stránku, která již nulový referenční bit má.
- Často používaný algoritmus (též označovaný jako tzv. clock algorithm).



– **Modifikace algoritmu druhé šance**:

- upřednostnění nemodifikovaných stránek jako obětí (modifikované se zapíší na disk a dostanou další šanci),
- dva ukazatele procházející frontou s určitým rozestupem – jeden nuluje referenční bit, druhý odstraňuje oběti (tzv. double-handed clock algorithm),
- Linux:
  - fronty „aktivních“ a „neaktivních“ stránek: stránka zpřístupněná dvakrát během jedné periody nulování referenčních bitů a odkládání neaktivních stránek se přesouvá do fronty aktivních stránek, z aktivní fronty se odstraňuje do neaktivní, z neaktivní se pak vybírají oběti,
  - systém se snaží nejprve odkládat stránky použité pro různé vyrovnávací paměti; při určitém počtu stránek namapovaných procesy přejde na odstraňování jejich stránek: prochází procesy a jejich stránky (propojené ve zvláštních seznamu), snaží se odstranit vždy alespoň určitý počet stránek z procesu, neodstraňuje stránky z aktivního seznamu (nebo alespoň referencované),
  - provádí odkládání po určitých počtech projdených a odložených stránek, agresivita se zvyšuje s rostoucím nedostatkem paměti,
  - tzv. „swap token“: stránky procesu, kterému je tato značka udělena, jsou přeskočeny při výběru obětí,
  - při kritickém nedostatku paměti ukončuje některé procesy.

Alokace rámců procesům (resp. jádru)

– Přidělování rámců procesům (resp. jádru) je významné zejména při lokálním výběru obětí, kde řídí přidělení množiny rámců, v rámci kterých se pak provádí lokální výměny.

– U globálního výběru lze použít pro řízení výběru obětí.

– Je třeba mít vždy přidělen minimální počet rámců pro provedení jedné instrukce: jinak dojde k nekonečnému vyměňování stránek potřebných k provedení instrukce.

– Dále se užívají různé heuristiky pro určení počtu rámců pro procesy (resp. jádro):

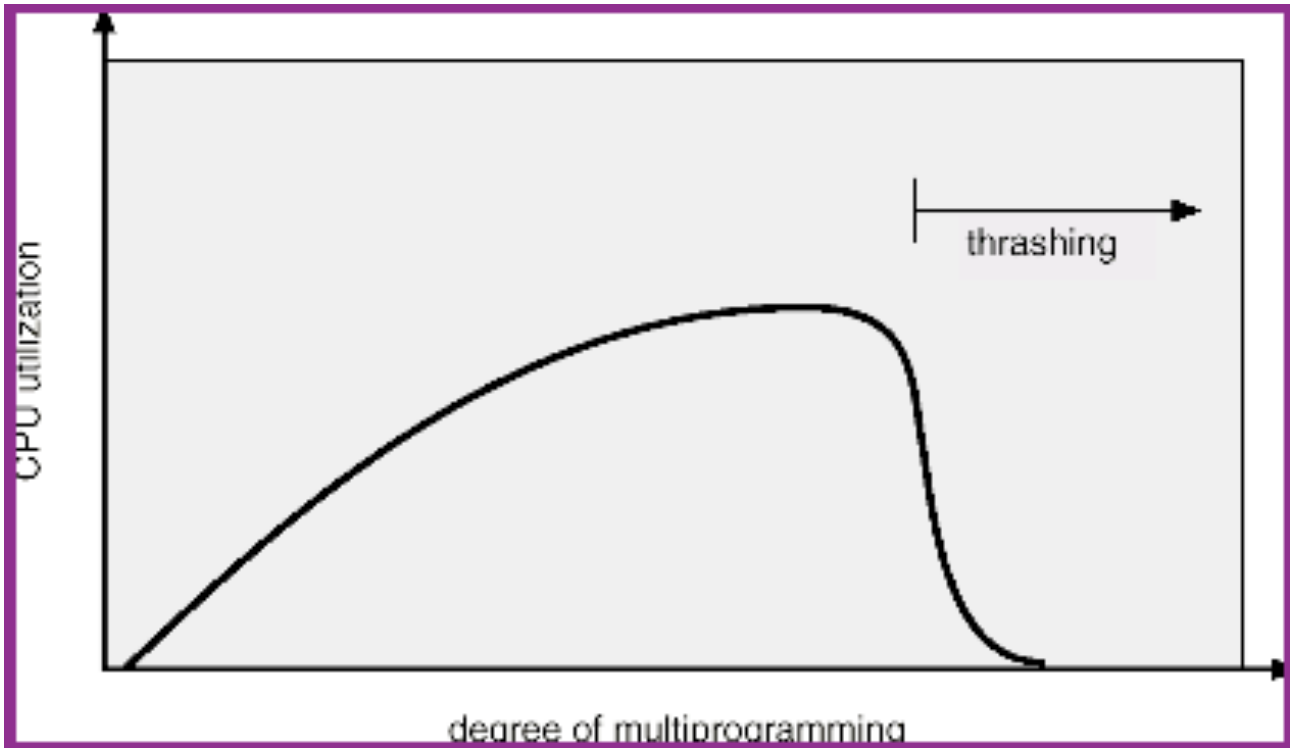
- Úměrně k velikosti programu, prioritě, objemu fyzické paměti, ...
- Na základě **pracovní množiny stránek**, tj. množiny stránek použitých procesem (resp. jádrem) za určitou dobu (aproximace s pomocí referenčního bitu).
- Přímo na základě sledování **frekvence výpadků** v jednotlivých procesech.

– Přidělování rámců s využitím pracovní množiny a kombinace lokální a globální výměny používají některé systémy **Windows**:

- Procesy a jádro mají jistý minimální a maximální počet rámců (meze velikosti pracovní množiny),
  - při dostatku paměti se jim rámce při potřebě přidávají až po dasažení maxima (příp. se povolí i zvětšení maxima),
  - při dosažení maxima, není-li nadbytek paměti, se uplatní lokální výměna,
  - při výrazném nedostatku volných rámců se do vytvoření patřičné zásoby volných rámců systém snaží (s agresivitou rostoucí s rostoucím nedostatkem paměti) odebírat procesům určitý počet stránek, a to na základě omezené historie přístupů.
- Při výběru obětí dává přednost větším procesům běžícím méně často; vyhýbá se procesům, které způsobily v poslední době mnoho výpadků a procesu, který běží na popředí.
- Počáteční meze pracovních množin odvozuje při startu systému z velikosti fyzické paměti.
- Vybrané oběti se snaží nepřidělit k jinému použití okamžitě, aby bylo možno korigovat chyby při volbě obětí.
- Uvedený mechanismus je doplněn odkládáním (swapováním) celých procesů (vybírá dlouho neaktivní procesy).

Thrashing

– I při rozumné volbě výběru odstraňovaných stránek může nastat tzv. **thrashing**: Proces (v horším případě systém) stráví více času náhradou stránek než užitečným výpočtem.



– Při vážném nedostatku paměti **swapper** (je-li v systému implementován) pozastaví některé procesy a odloží veškerou jejich paměť.

– Jinou možností je ukončení některých procesů.

Poznámky

– **Prepaging**: snaha zavádět do systému více stránek současně (zejména při startu procesu či po odswapování, ale i za běžné činnosti s ohledem na předpokládanou lokalitu odkazů).

– **Zamykání stránek**:

- zabraňuje odložení,
- užívá se např.
  - u stránek, do nichž probíhá I/O,
  - u (částí) tabulek stránek,
  - u (některých) stránek jádra,
  - na přání uživatele (k jeho vyjádření slouží v POSIXu – v rámci přednastavených limitů a oprávnění – volání `mlock()`): citlivá data, (soft) real-time procesy.

Sdílení stránek

– Stránkování umožňuje jemnou kontrolu sdílení paměti.

– **Sdílení stránek**:

- kód programů (procesy řízené stejným programem, sdílené knihovny),
- konstantní data nebo doposud nemodifikovaná data u kopií procesů (technologie copy-on-write),
- mechanismus IPC,
- sdílení paměťově mapovaných souborů.

Sdílené knihovny

– Sdílené knihovny (`.dll`, `.so`): kód, který je v dané verzi v FAP (a na disku) maximálně jednou a může být sdílen více procesy (procesy nemusí být řízeny stejným programem).

- Výhody: menší programy – lepší využití FAP i diskového prostoru, možnost aktualizovat knihovny.
- Nevýhody:
  - závislost programů na dalších souborech a verzích knihoven,
  - možný pomalejší start programu (je nutné dynamicky sestavit; na druhou stanu se ale zase může ušetřit díky nutnosti nezavádět již zavedené stránky),
  - možné pomalejší volání (nepřímé volání přes sestavovací tabulky; je ale možno ušetřit díky lepší lokalitě paměti – méně výpadků, lepší využití cache).

Copy-on-Write

– Při spuštění procesu pomocí `fork` se nevytvoří kopie veškeré paměti procesu.

– Vytvoří se pouze tabulky stránek a stránky se poznačí jako copy-on-write.

– K vytvoření fyzické kopie stránky dojde až při pokusu o zápis jedním z procesů.

– **Poznámka:** `vfork` jako alternativa k `fork` s copy-on-write: paměť je skutečně sdílena.

Sdílená paměť (*shared memory*)

– Forma IPC: Více procesů má mapovány stejné fyzické stránky do LAP.

– `shmget`, `shmat`, `shmdt`, `shmctl`

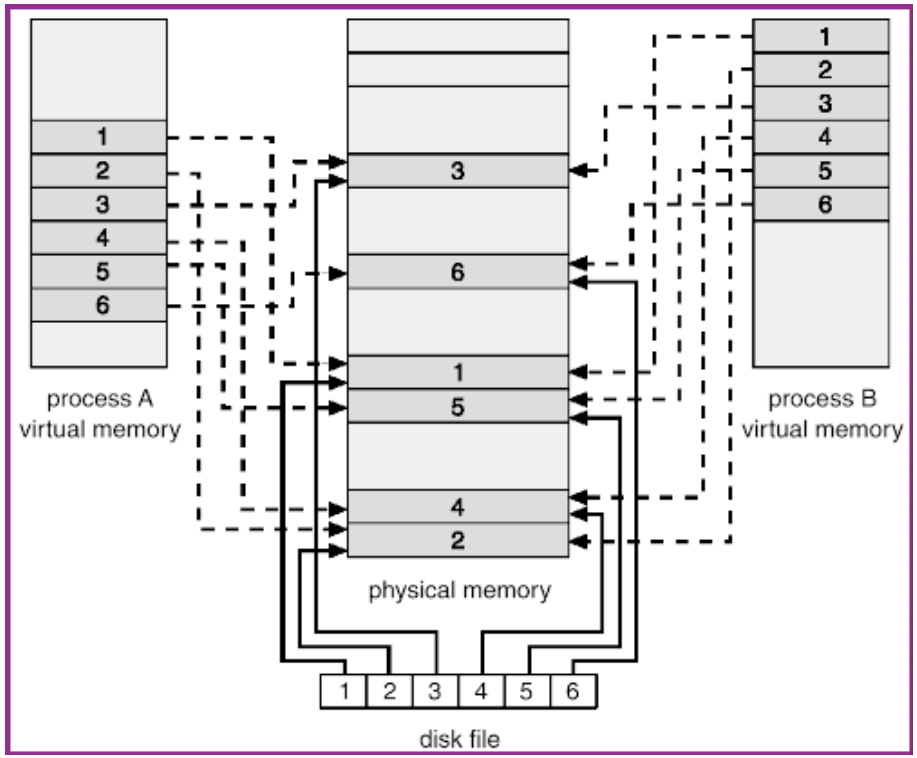
Příklad: GIMP plugin

Paměťově mapované soubory

– Bloky souborů jsou mapovány do stránek v paměti.

– Soubory jsou stránkováním na žádost načteny po stránkách do paměti a dále může být práce se soubory realizována standardním přístupem do paměti namísto použití `read()/write()`.

– Šetří se režie se systémovým voláním, kopírování do bufferu a pak do paměti. Umožňuje sdílený přístup k souborům.



– Dostupné prostřednictvím volání `mmap()`.

Paměťové regiony

– Paměťové regiony jsou jednotkou vyššího strukturování paměti v Unixu (používají se i další: např. v Linuxu tzv. uzly a zóny).

– Jedná se o spojité oblasti virtuální paměti použité za určitým účelem (data – statická inicializovaná data, statická neinicializovaná data, hromada; kód; zásobník; úseky individuálně mapované paměťi – sdílená paměť, paměťově mapované soubory, anonymní mapování).

– Každý proces může mít tabulku regionů procesu udávající pozici regionu v LAP procesu, přístupová práva k regionu a odkazující na systémovou tabulku regionů s dalšími informacemi o regionu, sdílenými mezi procesy – případně má každý proces všechny potřebné údaje přímo ve své tabulce regionů.

– V systémové tabulce regionů je uvedena velikost regionu, typ, i-uzel souboru případně mapovaného do regionu apod.

– Systém může provádět nad regionem jako celkem některé operace: zvětšování/zmenšování (u datového regionu) `brk`, swapování, mapování do paměti aj.

Rozložení adresového prostoru procesu v Linuxu (i386)

(Anatomy of a Program in Memory, G. Duarte)