

OS – Správa paměti

Tomáš Hudec

Tomas.Hudec@upce.cz

<http://fei-as.upceucebny.cz/usr/hudec/vyuka/os/>

Operační paměť

- jeden z nejdůležitějších prostředků spravovaných operačním systémem
- procesy pro svůj běh potřebují paměť
- efektivní správa nutnosti
- ochrana paměti
 - jádra OS před procesy
 - paměť procesu před ostatními procesy

Požadavky na operační paměť

- **relokace** paměti procesu
- **ochrana** paměti
- možnost **sdílení** paměti mezi procesy
- logická a fyzická organizace
 - logické členění paměti (procesu)
 - adresace (logické adresy → fyzické adresy)

Relokace paměti procesu

- přemístění paměti procesu na jiné adresy
- program nesmí pracovat s absolutními adresami fyzické operační paměti
 - programátor nemůže vědět, na které adresy je program do paměti zaveden
 - proces může být pozastaven a jeho paměť odložena na disk (swap)
- proces musí pracovat s logickými adresami
 - za běhu se převedou na skutečné fyzické adresy

Ochrana paměti

- procesy nesmějí přímo přistupovat k paměti
 - jádra OS
 - ostatních procesů
- kontrola přístupu musí být prováděna v době běhu procesu (run-time access control)
 - nelze provést podle kódu programu
- ochrana může mít různé stupně
 - čtení, zápis, provádění

Sdílení paměti mezi procesy

- procesy mohou potřebovat sdílet paměť
 - nutné pro efektivnější provádění procesů
 - soubory nebo jiné mechanismy výměny dat mohou být pomalé
- výhodné je sdílet paměť mezi procesy, které provádějí stejný program (knihovny)
 - kód procesu (program) je v paměti sdílen
 - vede k výrazné úspoře paměti
 - méně časté odkládání paměti na disk (swap)

Logická organizace paměti

- programy jsou obvykle psány modulárně
 - např. program, knihovny
- moduly vyžadují různý stupeň ochrany paměti
 - pouze pro čtení, zápis, pouze provádění
- moduly lze pro zvýšení efektivity sdílet
 - dynamicky připojované knihovny

Fyzická organizace omezeného množství paměti

- fyzická (skutečná) paměť nemusí vždy stačit
- techniky pro umožnění běhu procesů, které vyžadují více paměti, než je dostupné
 - **překrývání (overlaying)**
 - různé moduly programu nejsou vyžadovány současně
 - moduly používají stejnou fyzickou oblast paměti
 - **swapping**
 - odkládání procesu z operační paměti na sekundární
 - využití levné sekundární paměti (disku)

Techniky přidělování paměti

- historické
 - pevné dělení (fixed partitioning)
 - paměť je rozdělena na pevně definované oblasti
 - dynamické dělení
 - paměť je přidělována dle požadavků procesů
- moderní
 - lze alokovat paměť pouze pro část procesu
 - segmentace (segmentation) – víceméně na ústupu
 - stránkování (paging)

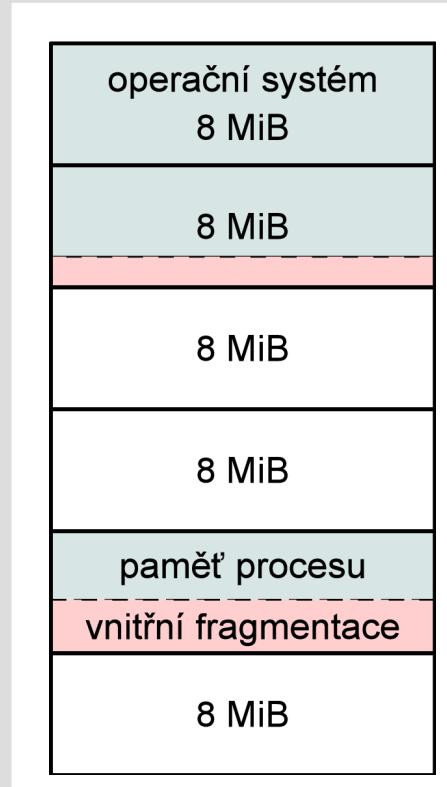
Pevné dělení paměti

- fixed partitioning
- dostupná paměť je rozdělena do oblastí (partitions) s pevnými hranicemi
- stejná velikost všech oblastí
 - proces je zaveden do libovolné volné oblasti
 - jsou-li všechny oblasti obsazeny, lze některý proces odložit na disk (swap out)
 - nevejde-li se proces do oblasti, musí programátor použít techniku překrývání (overlays)

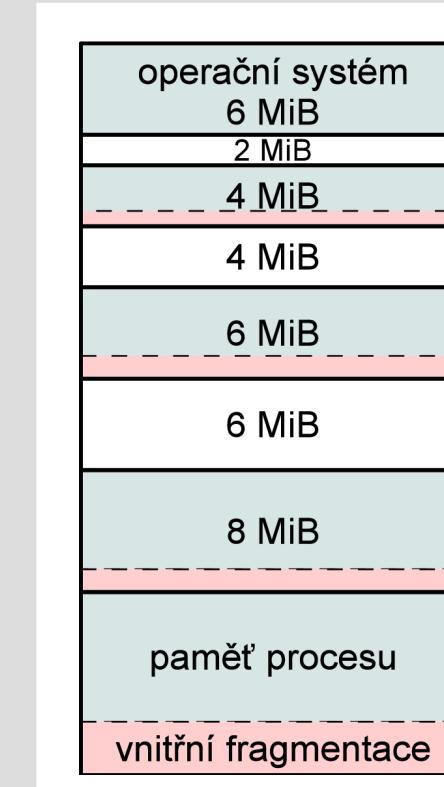
Pevné dělení paměti

- stejná velikost všech oblastí
 - jakýkoli malý proces zabere celou oblast
 - neefektivní využití paměti
 - **vnitřní fragmentace** (internal fragmentation) – proces nevyužije veškerou přidělenou paměť
- nestejná velikost oblastí
 - redukuje se vnitřní fragmentace
 - ale neodstraňuje se úplně

Pevné dělení (obrázek)



stejná velikost oblastí



nestejná velikost oblastí

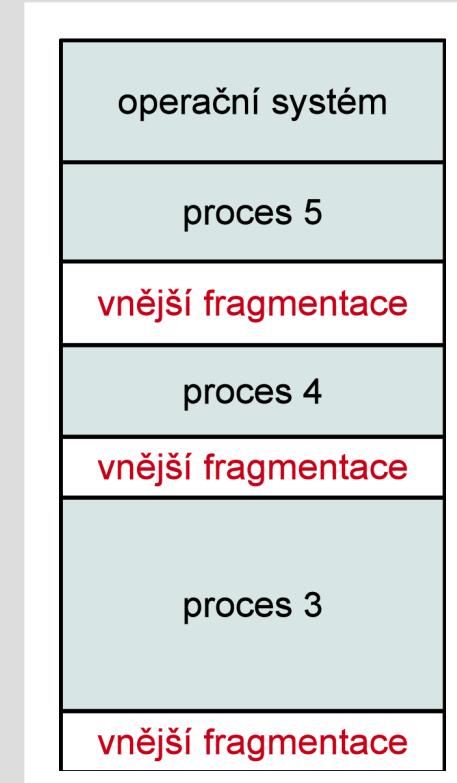
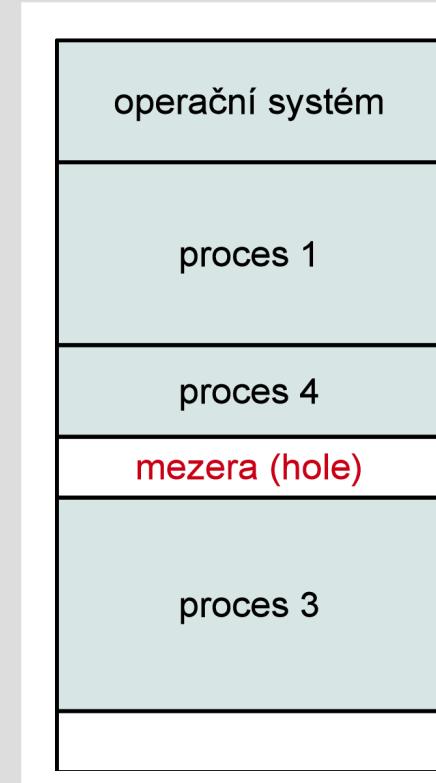
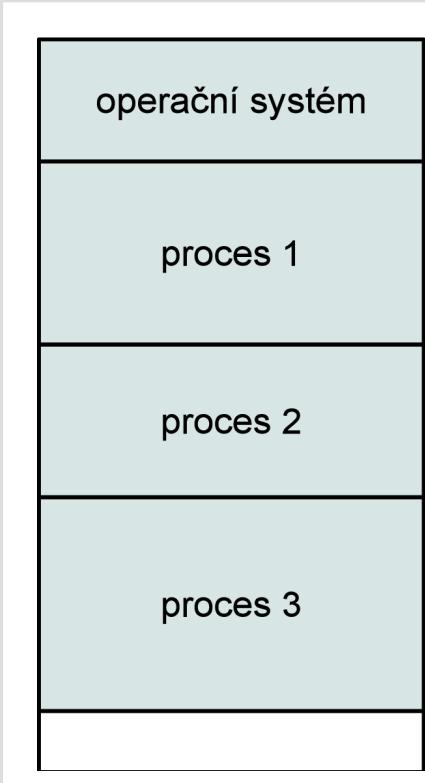
Umist'ovací algoritmus

- stejná velikost oblastí – libovolná volná oblast
- nestejná velikost oblastí
 - procesy čekající na přidělení paměti tvoří frontu
 - fronta samostatná pro každou velikost oblasti
 - fronta se pro proces vybírá tak, aby se minimalizovala vnitřní fragmentace – nejmenší oblast, kam se vejde
 - blokuje procesy, které by mohly být ve větší volné oblasti
 - fronta jediná
 - přidělí se nejmenší volná oblast, do které se proces vejde
 - rychlejší, ale za cenu větší vnitřní fragmentace

Dynamické dělení paměti

- proměnný počet oblastí i jejich velikost
- proces dostane tolik paměti, kolik potřebuje
 - **odstraněna vnitřní fragmentace**
- po ukončení procesu vznikají v paměti **mezery**
 - sem lze umístit jen proces, který se tam ještě vejde
 - při umístění procesu do mezery obvykle zůstane mnohem menší mezera – ještě obtížněji využitelná
 - **vnější fragmentace** (external fragmentation)
 - odstranit lze defragmentací – relokačí paměti procesů tak, aby vznikla souvislá volná oblast – **setřesení**

Dynamické dělení (obrázek)



alokace paměti pro
procesy 1, 2 a 3

ukončení procesu 2,
alokace pro proces 4

ukončení procesu 1,
alokace pro proces 5

Umisťovací algoritmus best-fit

- nejlépe padnoucí oblast – **best-fit**
 - vybere stejnou nebo nejmenší volnou oblast, do které se proces vejde
 - nejméně výkonná metoda
 - nejmenší možná fragmentace
 - vždy je použita nejmenší vyhovující oblast
 - fragmenty jsou malé, ale rychle přibývají
 - nutnost častého provádění setřesení (compaction) obsazené paměti

Umist'ovací algoritmus first-fit

- první padnoucí oblast – **first-fit**
 - paměť se prohledává vždy od začátku
 - vybere první volnou oblast, do které se proces vejde
 - rychlejší než best-fit
 - prohledávání zpomaluje výskyt velkého počtu obsazených oblastí na začátku paměti
 - tato část paměti se vždy zbytečně prohledává

Umist'ovací algoritmus next-fit

- následující padnoucí oblast – **next-fit**
 - paměť se prohledává vždy od oblasti, do které se naposledy umisťovalo
 - vybere první volnou oblast, do které se proces vejde
 - je-li po umístění procesu do volné oblasti zbytek oblasti ještě dostatečně velký, umístí se další proces sem
 - nejčastěji se umisťuje na konci paměti
 - obvykle tam je nejvíce volného místa
 - tendence dělit velké oblasti paměti na menší
 - nejrychlejší metoda

Umisťovací algoritmus worst-fit

- největší padnoucí oblast – exact-or-worst-fit
 - vybere stejně velkou volnou oblast jako proces, pokud existuje, jinak největší volnou oblast
 - paměť se prohledává do nalezení stejně velké oblasti
 - při nenalezení stejně velké oblasti prohledáváme celou paměť
 - tendence dělit velké oblasti paměti na menší
 - může mít za následek nemožnost přidělení paměti velkému procesu
 - nejhorší využití paměti – lowest memory utilization

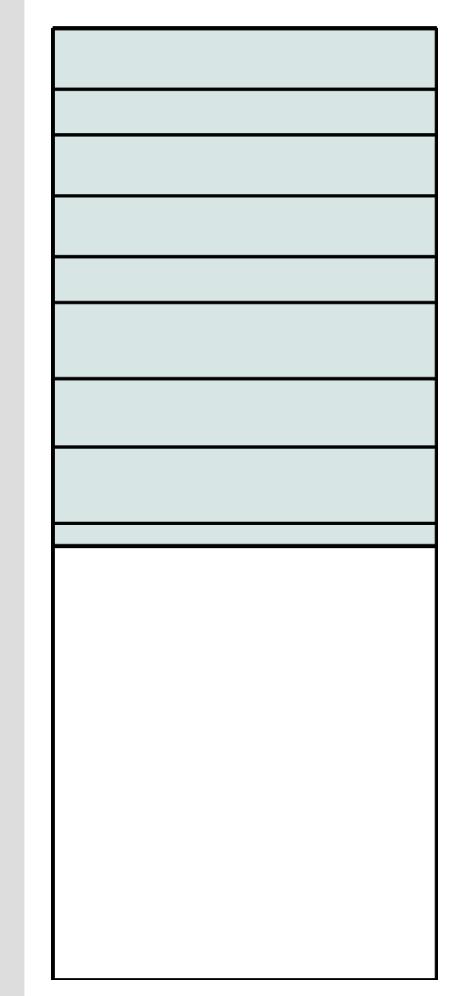
Umíst'ovací algoritmy (obrázek)



před umístěním



po umístění



po setřesení

Možné optimalizace algoritmů

- algoritmy můžeme zrychlit
 - evidence volných oblastí v setříděném seznamu
 - prohledává se pak seznam místo celé paměti
 - rychlosť vyhledání volné oblasti je pak stejná
 - pro metody first-fit a best-fit
 - metoda next-fit je bezúčelná
 - při uvolnění paměti přiléhající k volné oblasti je třeba sousední volné oblasti sloučit v jednu
- rychlé nalezení – **quick-fit**
 - vedeme další seznam běžně alokovaných velikostí

Typy adres

- fyzická adresa
 - používá ji Memory Management Unit (**MMU**)
 - absolutní adresa
- logická adresa
 - používá ji **CPU**
 - virtuální adresa – je nutno ji převést na fyzickou
 - absolutní adresa
 - vzhledem k logickému adresovému prostoru (procesu)
 - relativní adresa
 - vzhledem k počátku/konci oblasti (např. vrchol zásobníku)

Virtuální paměť

- skutečná paměť
 - fyzická operační paměť (FOP) v počítači (RAM)
- virtuální paměť
 - CPU je obvykle schopné adresovat větší množství paměti, než je skutečně instalováno
 - logický adresový prostor zahrnuje
 - skutečnou fyzickou operační paměť – RAM
 - část sekundární paměti (disku) – swap

Dělení adresového prostoru

- rozdělíme-li adresový prostor procesu na části, stačí, aby pouze některé z nich byly v RAM
 - zbytek adresového prostoru může být na disku
 - proces může běžet, i když nemá v RAM celý svůj adresový prostor
 - adresový prostor procesu může být větší než RAM
 - lze provádět více procesů současně
 - je-li více rozpracovaných procesů, je větší pravděpodobnost, že bude některý ve stavu připraven, a tudíž je procesor lépe využit

Dělení adresového prostoru

- **resident set**
 - část adresového prostoru procesu, která je v RAM
- **swap**
 - sekundární paměť (disk), kam lze odložit (dočasně) nepožívané části adresového prostoru procesu
 - přistupuje-li proces k místu (data, kód), které není v RAM, generuje se přerušení (a skok do OS)
 - OS pak proces blokuje, dokud nenahraje příslušnou část jeho adresového prostoru do RAM
 - po dokončení čtení je generováno přerušení a proces je převeden do stavu připravený

Princip lokality odkazů

- proces má tendenci přistupovat k okolí svého adresového prostoru, kam přistupoval nedávno
 - týká se dat (proměnné ve stejné datové části)
 - i kódu (zejména při provádění cyklů)
- lze obvykle odvodit, kterou část paměti bude proces v nejbližší budoucnosti používat
 - **virtuální paměť může** tedy **fungovat efektivně**
 - z disku se může dopředu načíst do RAM více částí paměti procesu z okolí právě vyžadované oblasti

Thrashing

- vzniká při špatné organizaci odkládání částí paměti procesu na disk
 - část paměti procesu je odložena na disk těsně před tím, než ji proces potřebuje
 - tuto část paměti je pak třeba následně nahrát zpět do RAM
 - proces pak zbytečně čeká v blokovaném stavu
- může být následkem nedostatečného dodržení principu lokality odkazů
 - neoptimalizovaného překladu (kompilátorem)
 - neefektivního programování (programátorem)

HW podpora pro virtuální paměť

- procesor musí umět pracovat s logickou (virtuální) adresou
 - CPU předává virtuální adresu paměťové jednotce
- MMU (Memory Management Unit) převádí virtuální adresu na adresu skutečnou (v RAM)
- nenachází-li se adresovaná část v RAM, je třeba generovat přerušení a předat řízení OS
 - OS vydá příkaz pro nahrání příslušné části paměti z disku do RAM

Stránkování paměti

- fyzická operační paměť **RAM** je rozdělena na oblasti malé velikosti – **rámce** (frames)
- logický adresový prostor **procesu** se rozdělí na stejně velké části jako RAM – **stránky** (pages)
- OS udržuje pro každý proces **tabulku** přiřazení stránek k rámcům
- logická adresa se skládá z **čísla stránky** a **offsetu**
 - offset je relativní adresa vzhledem k začátku stránky
 - číslo stránky = logická adresa / velikost stránky

Přidělení stránek (obrázek)

rámeč
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14

alokace rámců

rámeč
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14

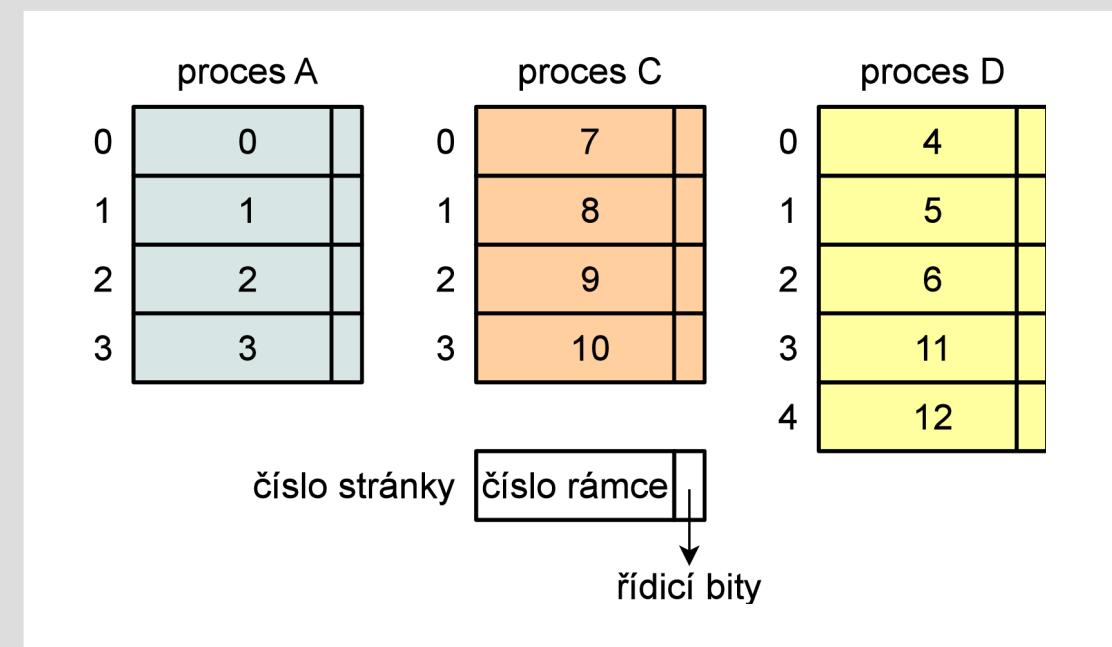
ukončení procesu B

rámeč
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14

alokace pro proces D

Stránkové tabulky (obrázek)

rámeč	
0	proces A – stránka 0
1	proces A – stránka 1
2	proces A – stránka 2
3	proces A – stránka 3
4	proces D – stránka 0
5	proces D – stránka 1
6	proces D – stránka 2
7	proces C – stránka 0
8	proces C – stránka 1
9	proces C – stránka 2
10	proces C – stránka 3
11	proces D – stránka 3
12	proces D – stránka 4
13	
14	

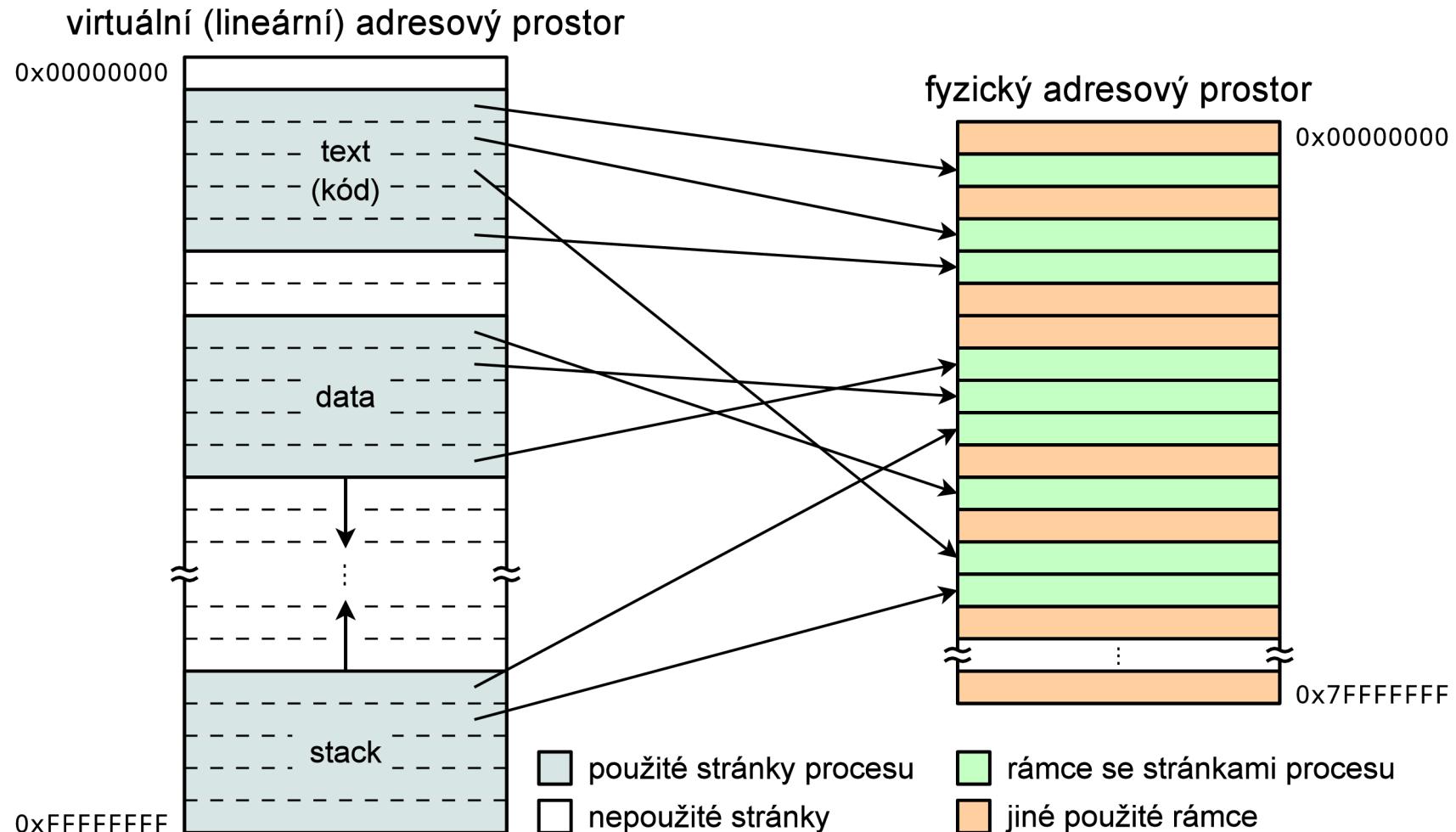


stránkové tabulky procesů
indexem je číslo stránky

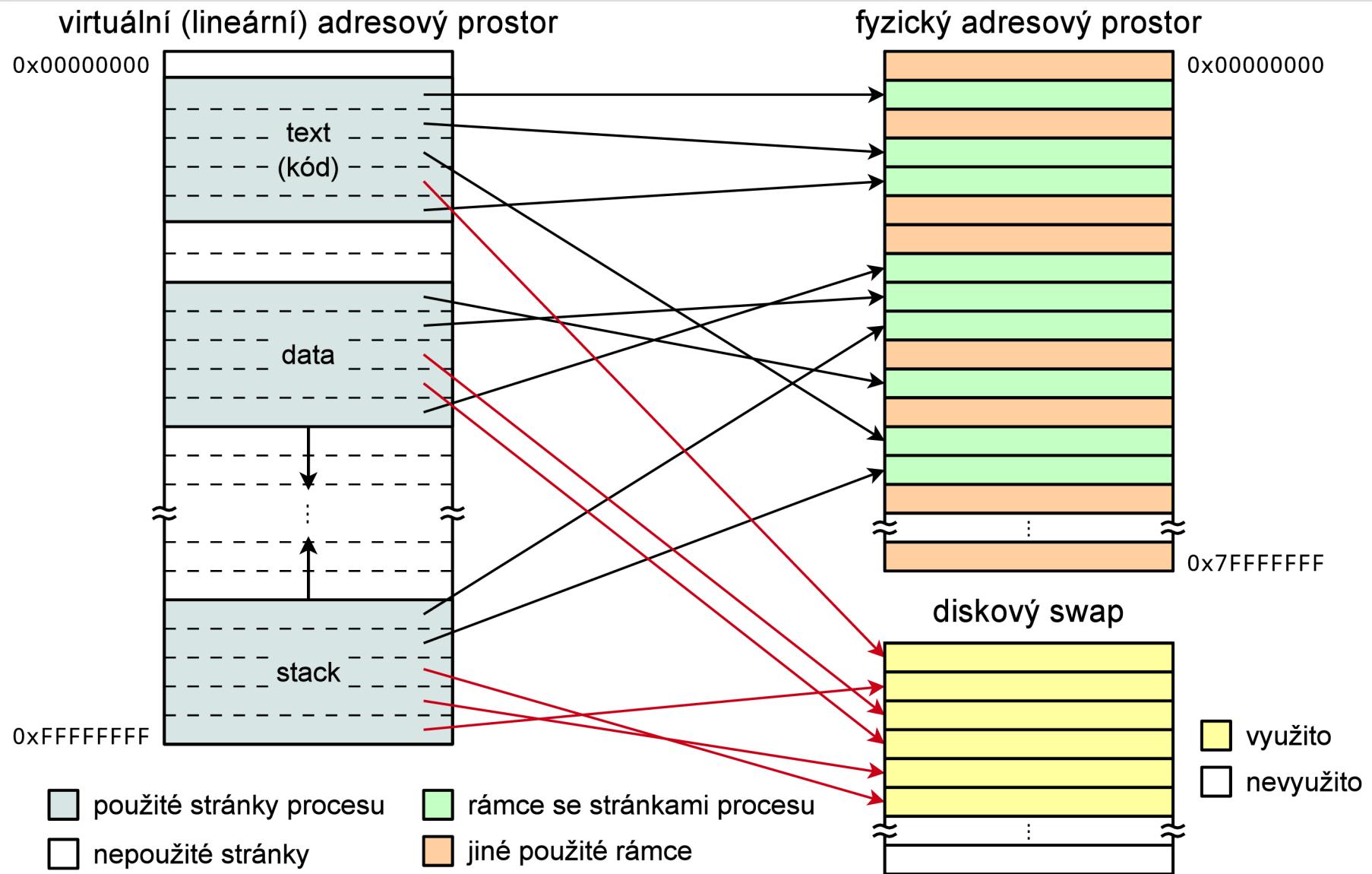
Stránkování paměti

- logický adresový prostor procesu je **lineární**
- skutečné umístění stránek v RAM je **nespojité**
- velikost stránek je daná HW, typicky KiB až MiB
 - IA-32: **4 KiB** a 4 MiB (bez PAE) nebo 2 MiB (s PAE)
- stránková tabulka obsahuje příznaky (**řídicí byty**)
 - **přítomnost** stránky v RAM (present)
 - stránka je **používaná** (referenced, accessed)
 - **modifikovaný** obsah (modified, dirty)
 - nebyla-li stránka v RAM modifikovaná a je již na disku, není třeba ji znova zapisovat při uvolňování rámce

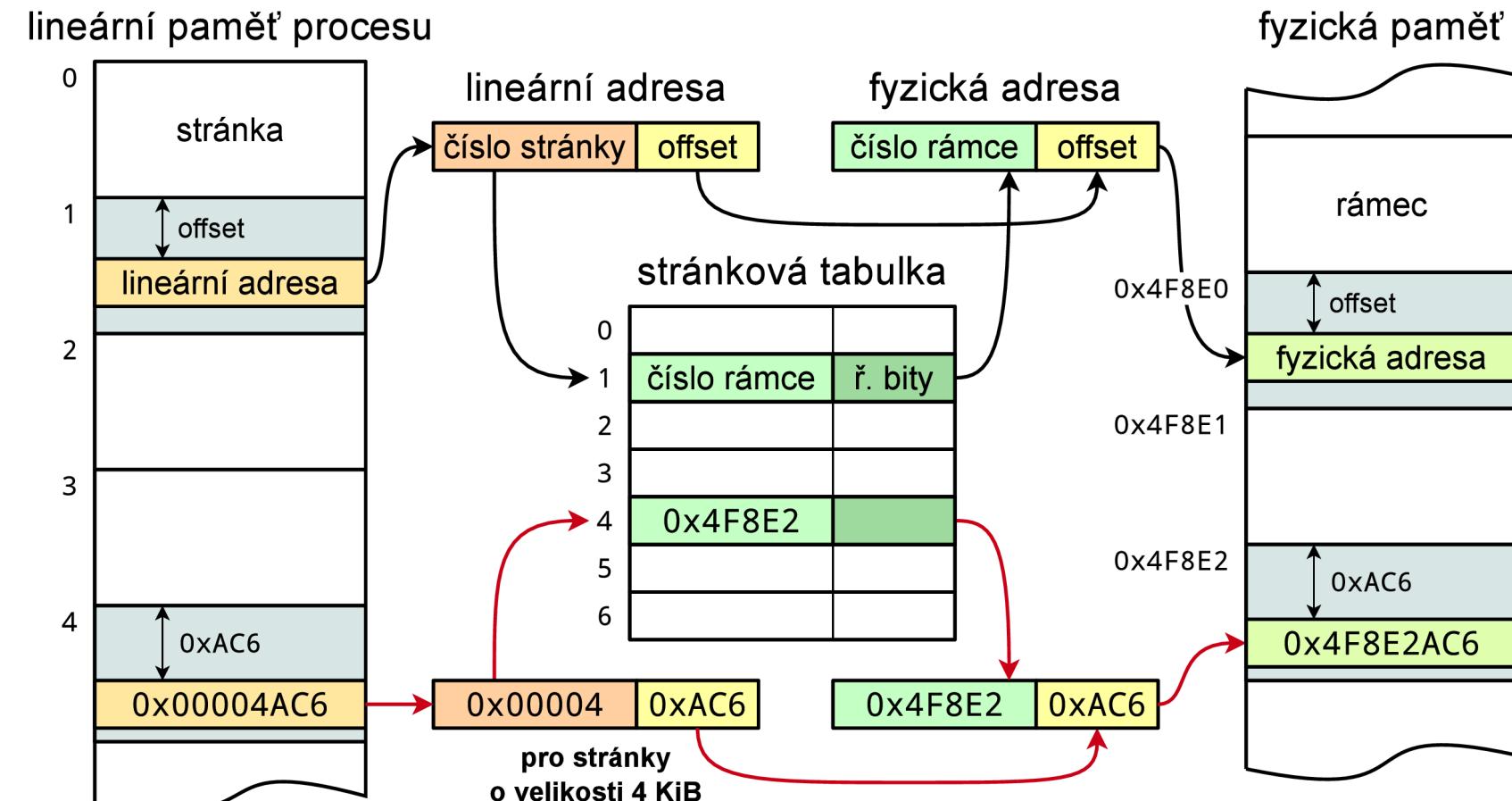
Zobrazení stránek procesu do rámci FOP (obrázek)



Zobrazení stránek procesu do virtuální paměti (obrázek)



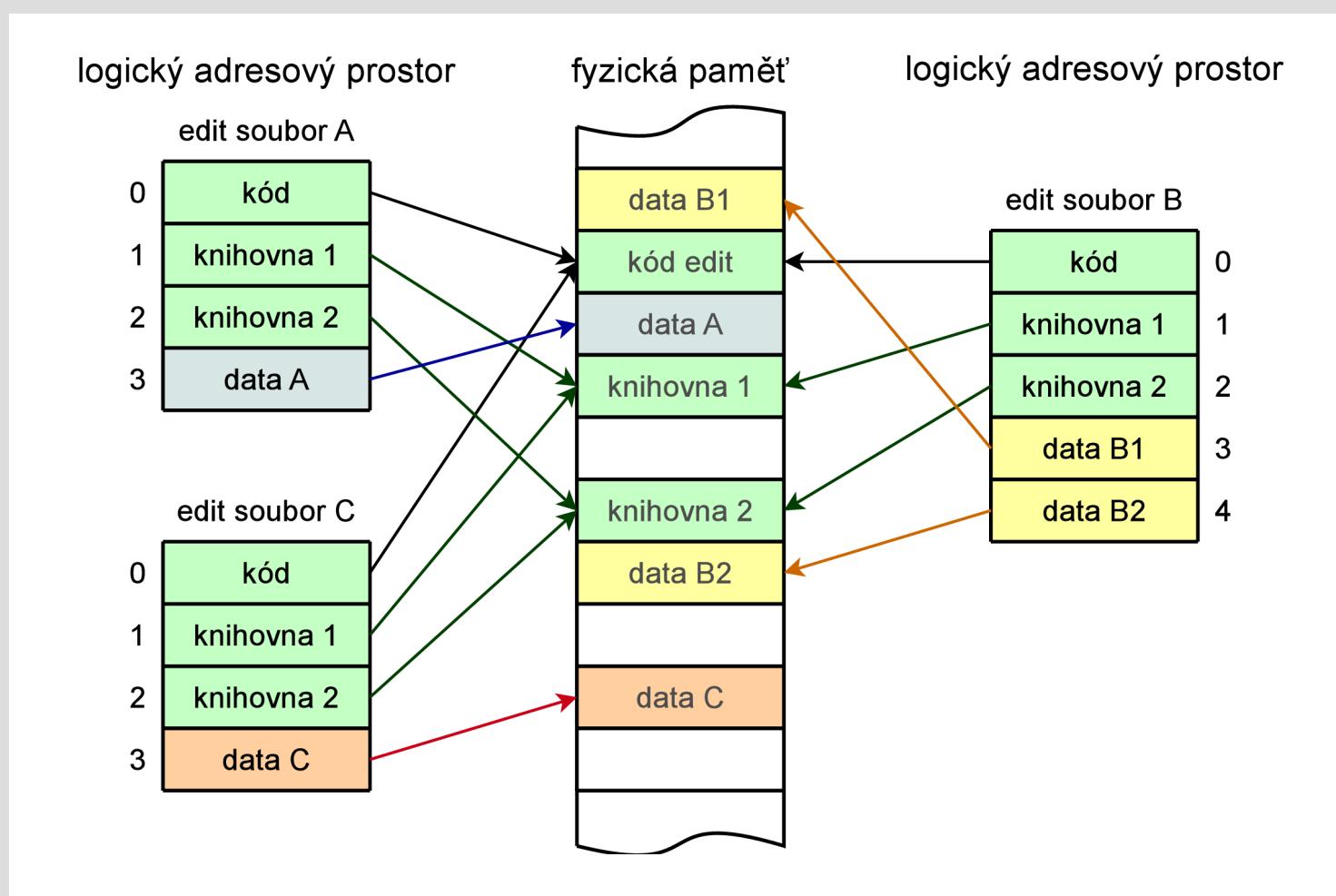
Stránkování – převod virtuální adresy na fyzickou (obrázek)



Sdílení stránek

- stejné stránky různých procesů mohou sdílet rámcem
 - je zbytečné mít v paměti stejně kopie dat
 - je snadné sdílet stránky v režimu read-only (kód)
 - je třeba brát ohled na odkládání obsahu sdílených rámců na swap
- lze sdílet i stránky obsahující stránkové tabulky
 - např. dva procesy vykonávající stejný program mohou sdílet část stránkové tabulky příslušející přiřazení stránek pro kód programu (a knihovny)

Stránkování – sdílení (obrázek)



Sdílení stránek v OS UNIX

- po provedení systémového volání fork(2)
 - rodič i potomek má svou vlastní stránkovou tabulkou
 - stránkové tabulky jsou identické kopie
 - procesy sdílejí celou virtuální paměť
 - všechny stránky jsou označeny read-only
 - při zápisu do stránky je vyvoláno přerušení (porušení ochrany) a předá se řízení OS (TRAP)
 - OS vytvoří kopii stránky (každý proces má nyní vlastní) a nastaví obě kopie do režimu read-write – **copy-on-write**
 - důsledek: většina stránek je sdílena

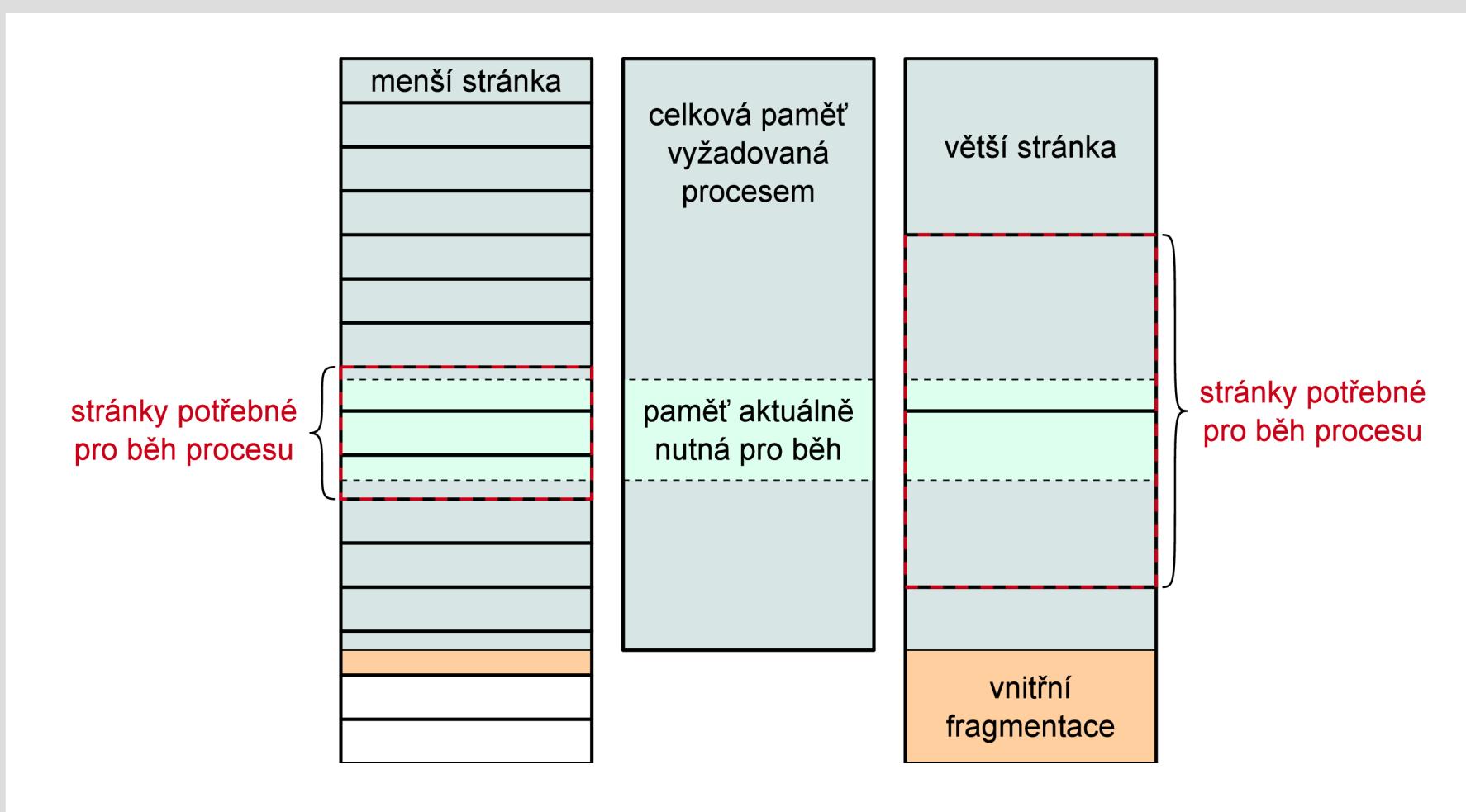
Velikost stránek – menší

- paměť se alokuje po stránkách pevné velikosti
 - vzniká **vnitřní fragmentace**
- menší velikost stránky
 - menší vnitřní fragmentace
 - více stránek na proces
 - větší stránková tabulka – zabírá další paměť
 - více rámců v RAM = lepší hospodaření s RAM
 - v RAM budou stránky z okolí posledních odkazů na paměť a počet neúspěšných přístupů se bude snižovat

Velikost stránek – větší

- větší velikost stránky
 - větší vnitřní fragmentace
 - menší stránková tabulka
 - méně rámců v RAM = horší hospodaření s RAM
 - v RAM budou díky velikosti stránek i bloky paměti, které nejsou aktuálně využívány – princip lokality
 - při nedostatku RAM se bude počet neúspěšných přístupů zvyšovat – hrozí thrashing
 - sekundární paměť je efektivnější při přenosu dat po větších blocích

Velikost stránek (obrázek)



Rozsah stránkové tabulky

- rozsah stránkové tabulky může být velký
 - virtuální adresa 32 bitů, velikost stránky 4 KiB:
 - offset 12 bitů ($2^{12} = 4 \text{ Ki}$), adresový rozsah 4 GiB (2^{32}), pro počet stránek 20 bitů $\rightarrow 2^{32-12} = 2^{20} = 1 \text{ Mi}$ stránek
 - má-li položka stránkové tabulky 32 bitů, pak její celková velikost je **4 MiB** ($1 \text{ Mi} \cdot 4 \text{ B}$) **pro každý proces**
 - proces většinu stránek nepoužije
 - typicky používá stránky z počátku adresového rozsahu (kód a data) a stránky z konce rozsahu (stack)
 - udržovat celou tabulku v paměti je zbytečné
 - 64bitová adresa: velikost tabulky 32 PiB ($2^{64-12} \cdot 8 \text{ B}$)
 - nerealizovatelné!

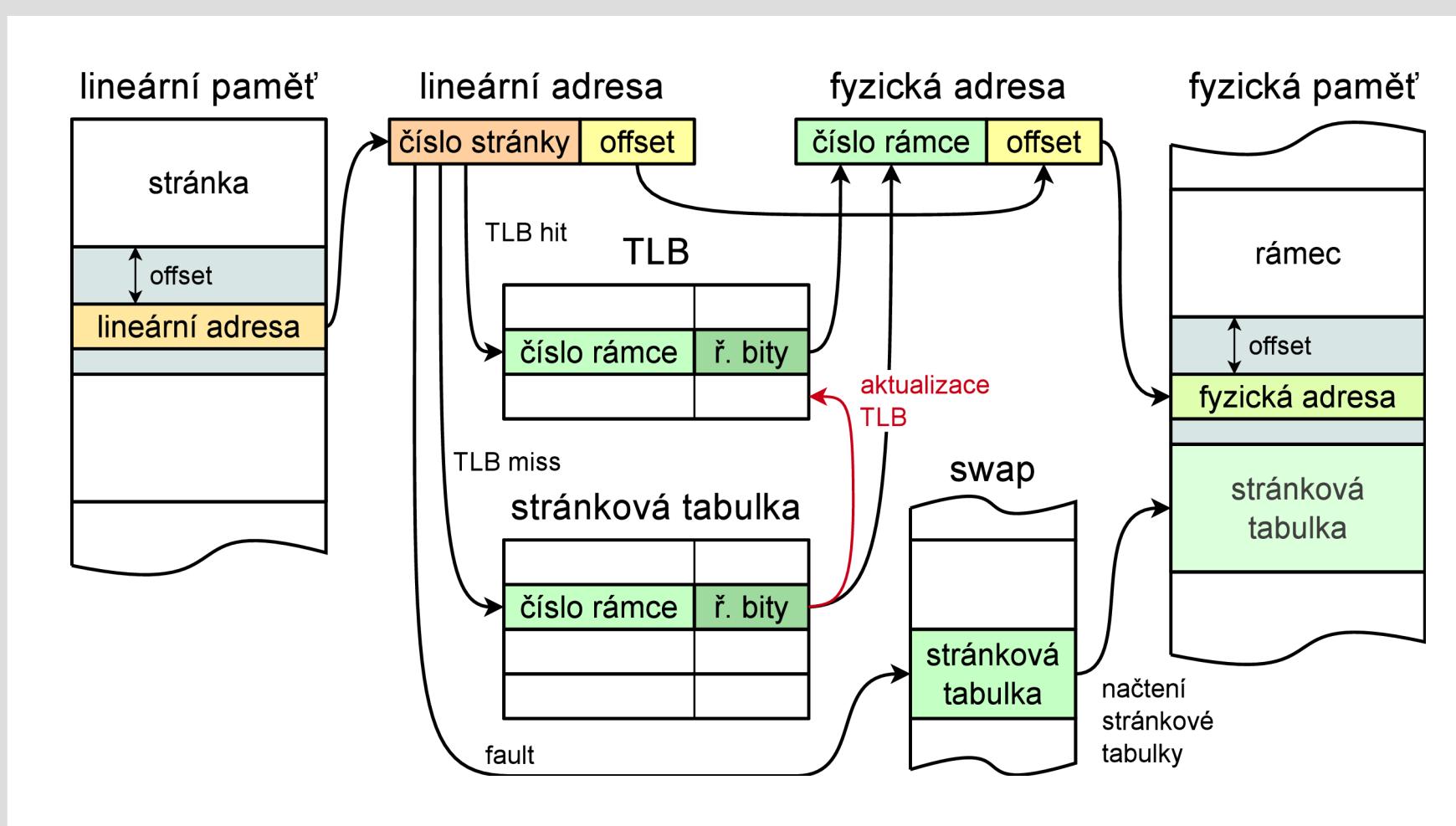
Umístění stránkových tabulek

- stránkové tabulky musí být také v paměti
 - velikost tabulky může být velká, proto se tabulka umisťuje také **do virtuální paměti**
 - v RAM nemusí být tabulka celá
 - část může být na disku (swap)
 - při přístupu procesu na paměťové místo ve stránce, jejíž položka ve stránkové tabulce není v RAM, znamená čtení z disku – zpomalení překladu
 - po načtení položky se může zjistit, že ani stránka není v RAM (**page fault**) – další čtení z disku a zpomalení běhu procesu

Translation Lookaside Buffer (TLB)

- speciální cache pro položky stránkové tabulky
- obsahuje několik posledních použitých položek stránkové (příp. invertované) tabulky
- při hledání položky stránkové tabulky se prohledá nejprve TLB
 - kontrola přítomnosti v TLB může být paralelizována
 - nalezení (hit) – převod virtuální adresy na fyzickou
 - nenalezení (miss) – doplnění položky do TLB
 - doplnění může být z části tabulky v RAM nebo z disku

Použití TLB (obrázek)



Invertovaná stránková tabulka

- místo stránek se v tabulce evidují rámce
 - velikost je závislá na instalovaném množství RAM
 - výrazná úspora: **jediná tabulka pro celý systém**
 - položky obsahují kromě čísla stránky také ID procesu
 - 64bitové položky, 4KiB stránky, 1 GiB paměti RAM:
 - velikost: $1 \text{ GiB} / 4 \text{ KiB} \cdot 8 \text{ B} = 2 \text{ MiB}$ (2^{18} 64b. záznamů)
- převod virtuální adresy má ale vyšší režii
 - tabulka je indexovaná čísly rámců, nikoliv stránek, tj. je třeba tabulku prohledat celou – složitost $> O(1)$
 - částečné zrychlení – použití TLB
 - jiná metoda zrychlení – použití hashovací tabulky

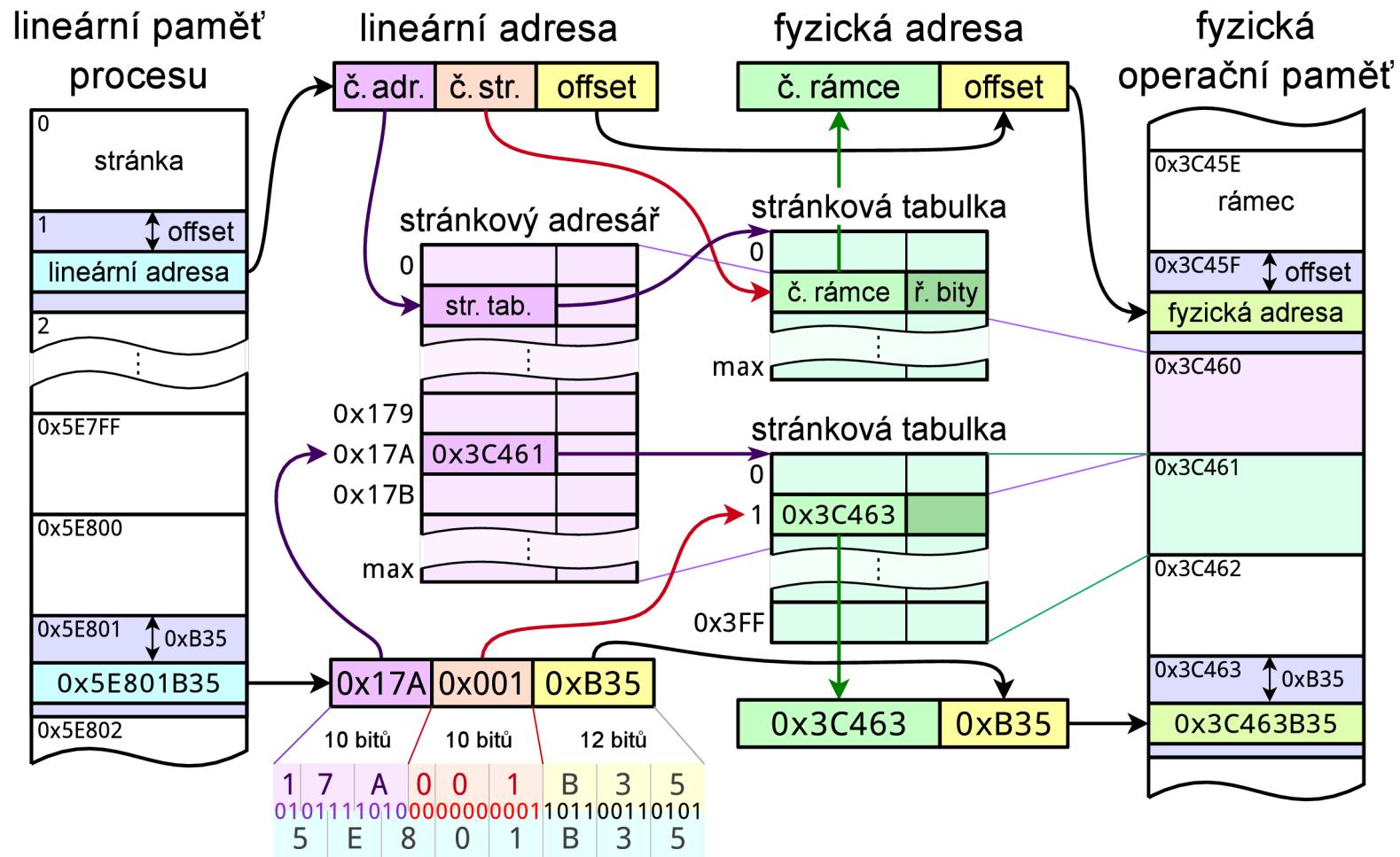
Víceúrovňové stránkové tabulky (1)

- velký rozsah stránkové tabulky lze řešit použitím hierarchických stránkových tabulek
 - virtuální adresa se rozdělí na offset a dva indexy do stránkových tabulek dvou úrovní
 - první index určí položku **stránkového adresáře**, ze které se odvodí **adresa stránkové tabulky** druhé úrovně
 - druhý index určí položku **stránkové tabulky**, která obsahuje **číslo rámce**
 - př.: 4KiB stránky: 12 bitů offset a 2 indexy 2×10 bitů
 - vytvoří se tak 1024 položkové (2^{10}) tabulky stránek
 - stránková tabulka zabírá jeden rámec ($2^{10} \cdot 4 \text{ B} = 4 \text{ KiB}$)

Víceúrovňové stránkové tabulky (2)

- vytvoří se stromová struktura tabulek
- hlavní tabulka (adresář) zůstává v RAM
- tabulky druhé úrovně jsou ve virtuální paměti
 - mohou být odloženy na disk
 - nemusejí existovat – mohou být vytvářeny dynamicky za běhu procesu
 - když proces alokuje další paměť a je potřeba ji umístit do nových stránek
- používáno architekturami IA-32 (2–3 úrovně) a IA-32e (x86_64, 4 a více úrovní)

Stránkování – dvouúrovňové stránkové tabulky (obrázek)



Stránkování na platformě IA-32

- virtuální adresa: 32 bitů
 - proces může mít max. 4 GiB paměti (2^{32})
- velikost stránky: 4 KiB
 - případně 4 MiB (bez PAE) nebo 2 MiB (s PAE)
- fyzická adresová sběrnice: 36 bitů
 - fyzické operační paměti může být max. 64 GiB
 - pro použití více než 4 GiB je nutná podpora PAE
 - bez PAE: dvouúrovňová struktura stránkových tabulek

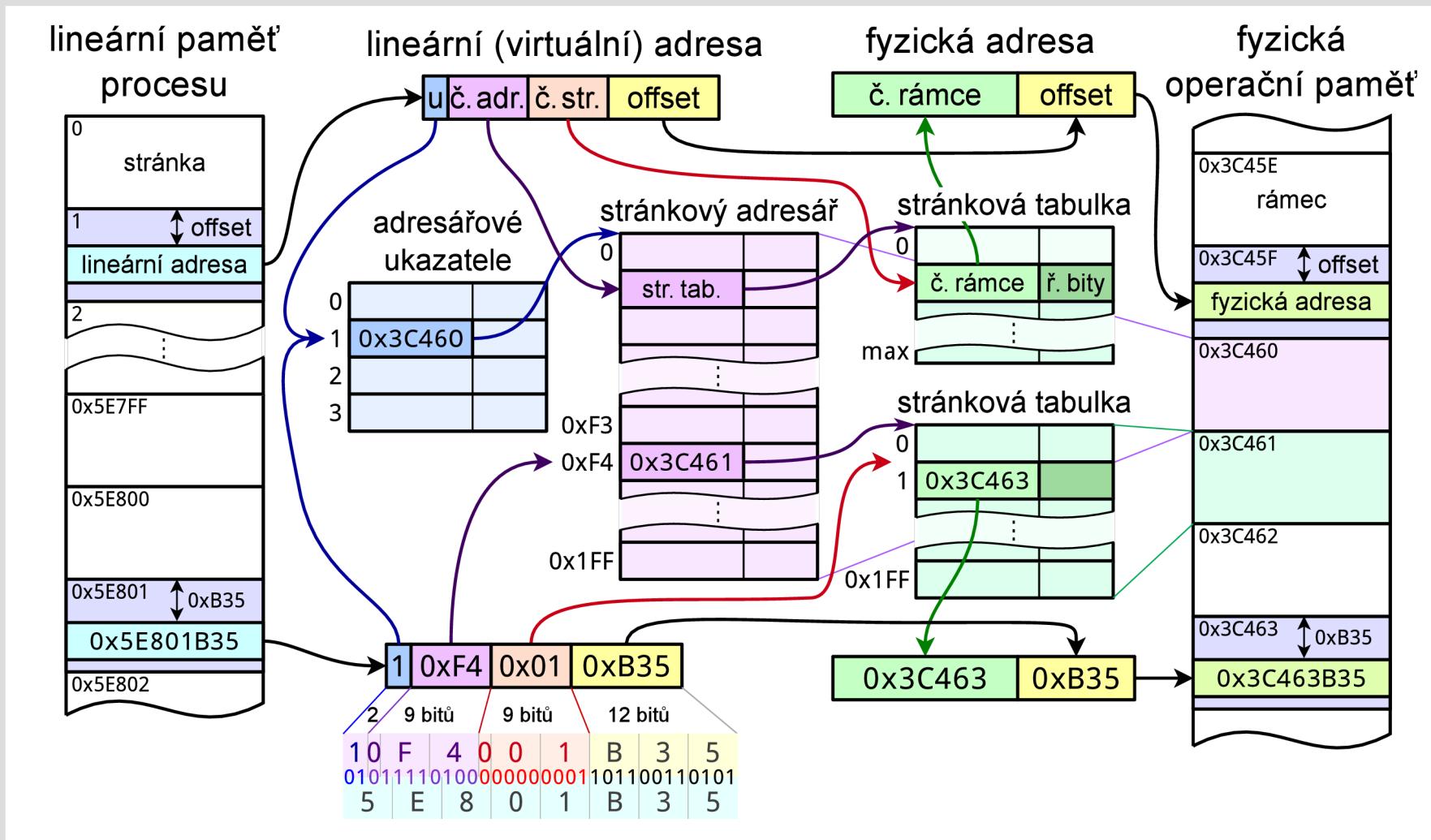
IA-32: řídicí bity v položce stránkové tabulky / adresáře

- 32bitový režim – dolních 12 bitů je řídicích
 - stránková tabulka (PT), stránkový adresář (PD)
 - 0 – **P** – **present** – použitá položka
 - 1 – R/W – zápis povolen
 - 2 – U/S – user/supervisor – přístup v uživatelském režimu
 - 3 – PWT – page-level write-through
 - 4 – PCD – page-level cache disable
 - 5 – **A** – **accessed (referenced)** – stránka byla použita
 - 6 – PT: **D** – **dirty (modified)** – změněna, PD: ignored
 - 7 – PT: PAT/reserved, PD: PS – page size (4KiB/4MiB)
 - 8 – PT: G/ignored – global (podle CR4.PGE), PD: ignored
 - 9–11 – ignored

Režim PAE na platformě IA-32

- PAE (Physical Address Extension)
 - tří- / dvouúrovňové stránkování (4KiB / 2MiB stránky)
 - lineární adresa má 32 bitů, fyzická 36 (až 52) bitů
 - 2 bity: ukazatele na stránkové adresáře (registry PDPTEi)
 - 9 bitů: index do stránkového adresáře
 - 9 bitů: index do stránkové tabulky (pouze pro 4KiB stránky)
 - 12 bitů / 21 bitů: offset ve 4KiB / 2MiB stránce
 - položky stránkové tabulky / adresáře mají 64 bitů
 - číslo rámce: 24–40 bitů (4KiB stránky), 15–31 bitů (2MiB)
 - řídicí bity: dolních 12 bitů, bit 63: XD (execute-disable), pro 2MiB stránky: PAT je bit 12 (bit 7 je PS)

Stránkování – tříúrovňové stránkové tabulky (obrázek)



Adresace IA-32e (64-bit)

- IA-32e – dva podrežimy
 - kompatibilní: virtuální adresace je 32bitová
 - 64bitový: 48 bitů virtuálních → 40–52 bitů fyzických
 - 2^{48} B = 256 TiB pro proces, až 2^{52} B = 4 PiB fyzické RAM
 - 64bitový režim: stránky 4 KiB, 2 MiB nebo 1 GiB
 - čtyř-, tří- a dvouúrovňové stránkové tabulky
 - 9 bitů: PML4 (page map level four)
 - 9 bitů: index do tabulky ukazatelů na stránkové adresáře
 - 9 bitů: index do stránkového adresáře, pouze 4KiB a 2MiB
 - 9 bitů: index do stránkové tabulky, pouze pro 4KiB stránky
 - 12 / 21 / 30 bitů: offset ve 4KiB / 2MiB / 1GiB stránce

Stránkové algoritmy

- algoritmy pro manipulaci se stránkami procesů
- strategie zavádění – **fetch policy**
 - které stránky a **kdy** se zavedou **do paměti**
- strategie umisťování – **placement policy**
 - **kam** se umístí do fyzické paměti (do kterého rámce)
- strategie nahrazování – **replacement policy**
 - které stránky **se odloží** (na swap)
- strategie uklízení (čištění) – **cleaning policy**
 - **kdy se uloží** modifikované stránky na swap

Zavádění stránek do RAM

- strategie zavádění – **fetch policy**
 - určuje, **která** stránka se má **kdy** zavést do RAM
- **demand paging**
 - zavádění stránek jen tehdy, jsou-li potřeba
 - vyskytne-li se odkaz na paměťové místo v dané stránce
 - způsobuje neúspěšné přístupy při zavádění procesu
- **lookahead paging**
 - zavádění stránek předem
 - dle principu lokality odkazů se zavedou stránky z okolí té právě vyžadované (využívá se sekvenční čtení z disku)

Umisťování stránek do RAM

- strategie umisťování – **placement policy**
 - určuje, **do které části RAM** (rámce) se stránka zavede
 - typicky se tím OS nezabývá
 - vybírá obvykle libovolný volný rámec
 - přístupová doba (access time) je stejná pro všechny adresy v RAM
 - skutečnou adresu používá pouze HW

Nahrazování stránek v RAM

- strategie nahrazování – **replacement policy** (RP)
 - určuje, které stránky se z RAM odloží na disk, je-li potřeba načíst jiné stránky
- ideální strategie – odstranění stránky, která bude nejdelší dobu nevyužitá
 - nelze předem přesně určit – pouze se odhaduje
- pro stránky, které nesmí být odstraněny z RAM, se používá **zamykání rámců** (frame locking)
 - jádro OS, V/V buffery, klíčové řídicí struktury OS

RP – Least Recently Used (LRU)

- nahradí stránku, na kterou nebyl nejdelší dobu žádný odkaz
 - dle principu lokality odkazů je to stránka s nejmenší pravděpodobností výskytu odkazu v nejbližší budoucnosti
- režijně náročná strategie
 - u každé stránky je třeba udržovat údaj o čase posledního odkazu na ni (nebo stačí čítač)
 - při nahrazování je třeba najít patřičnou stránku
 - lze řešit pomocí seřazeného seznamu stránek

RP – Least Recently Used (LRU) – možné implementace

- seřazený seznam má vysokou režii
 - při každém přístupu do paměti je třeba aktualizovat
- rozšíření řídicích bitů položky stránkové tabulky
 - při přístupu ke stránce uloží MMU do stránkové položky navýšenou hodnotu globálního čítače
 - při výskytu page-fault se vyhledá nejnižší hodnota
- evidence matice $n \times n$ bitů, kde $n =$ počet rámců
 - při přístupu ke stránce v rámci číslo k nastaví MMU v k -tém řádku jedničky a v k -tém sloupci nuly
 - page-fault → řádek matice s nejnižším číslem (↗)

RP – Not Recently Used (NRU)

- každá stránka má v řídicích bitech položky
 - odkazovaná stránka (referenced bit – R)
 - nastaví se při každém čtení stránky nebo jejím zápisu
 - periodicky se tento bit nuluje
 - změněná stránka (modified bit – M)
 - nastaví se při změně obsahu stránky
- nahradí libovolnou stránku z nejnižší třídy
 1. $R = 0, M = 0$ (neodkazovaná, nezměněná)
 2. $R = 0, M = 1$ (neodkazovaná, změněná)
 3. $R = 1, M = 0$ (odkazovaná, nezměněná)
 4. $R = 1, M = 1$ (odkazovaná, změněná)

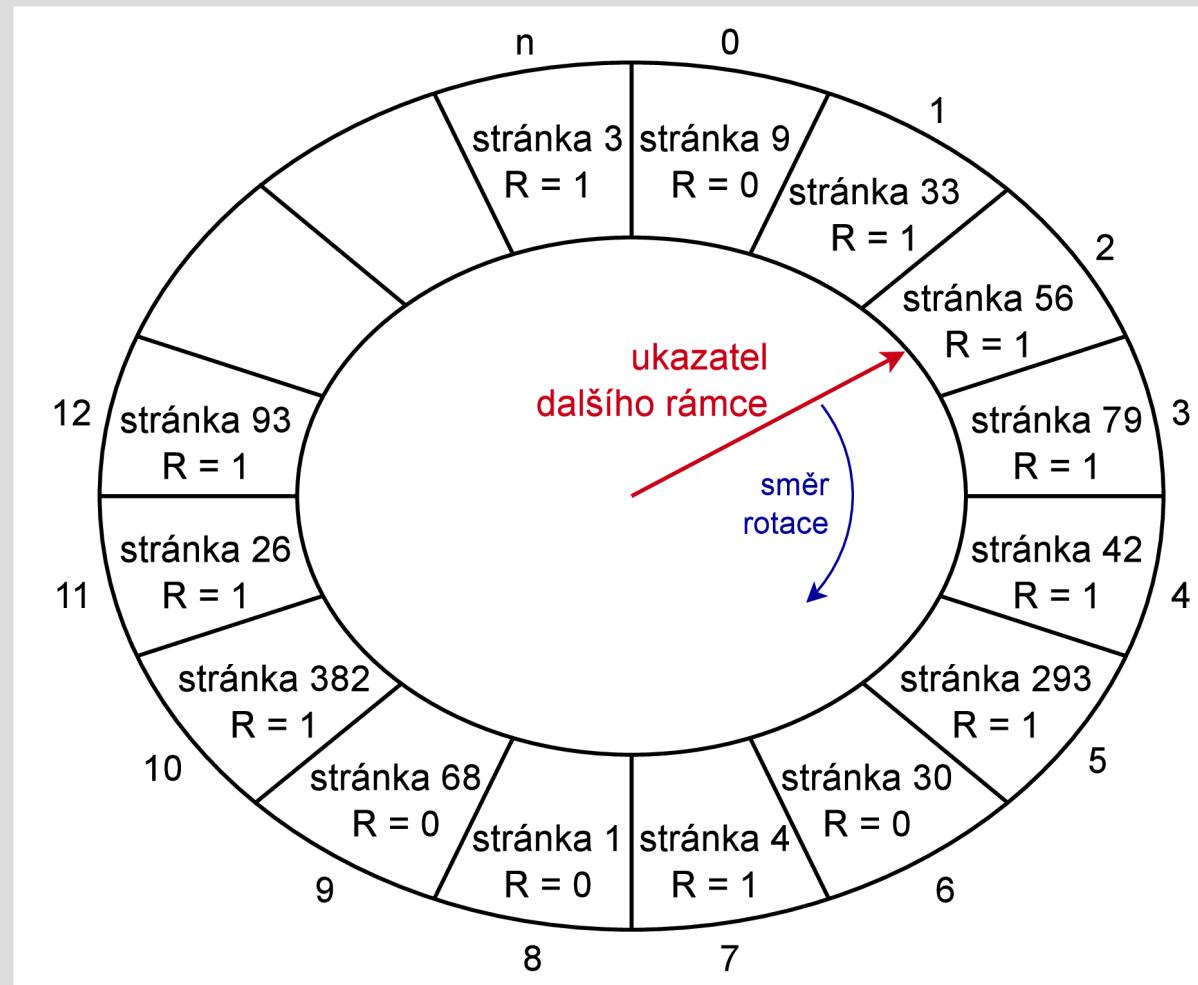
RP – First-In, First-Out (FIFO)

- nahradí stránku, která byla nejdéle ve FOP
 - OS udržuje seznam stránek ve frontě
 - nové položky jsou připojovány na konec fronty
 - první ve frontě je tedy stránka nejdéle používaná
- nenáročná strategie, ale nevýhodná
 - může odstranit stránku, která je právě používaná
- modifikace strategie – **second chance**
 - je-li bit referenced – $R = 1$, je vynulován a stránka je přesunuta na konec fronty

RP – Clock Policy

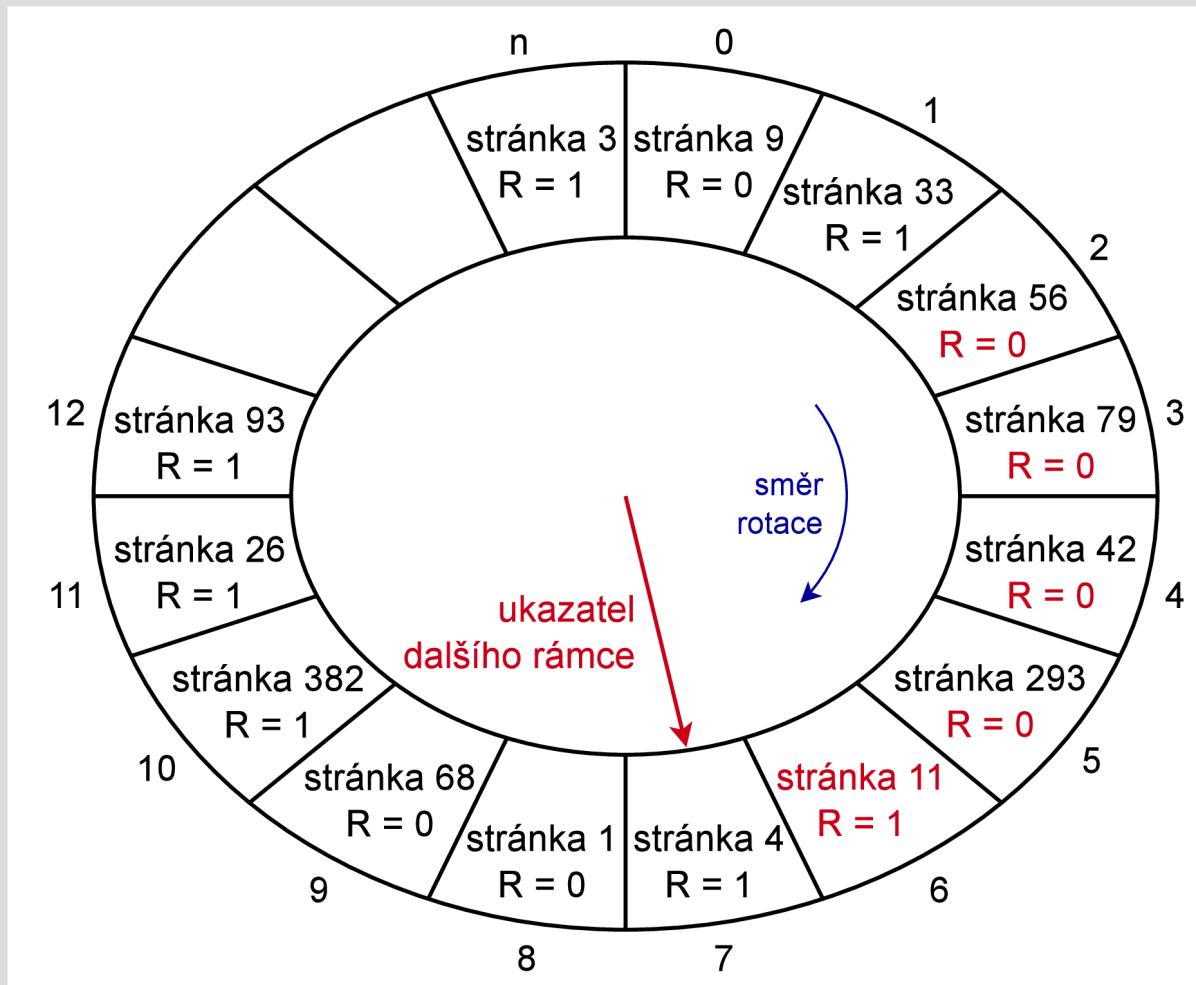
- rámce jsou v kruhovém seznamu
 - index určuje poslední položku
 - není-li volný rámcem a nastane page fault, zvyšuje se index tak dlouho, dokud se nenajde stránka s bitem použití (referenced) $R = 0$
 - tato stránka se nahradí stránkou novou
 - při procházení seznamu se bit R nuluje

Clock Policy (obrázek 1)



stav před umístěním stránky číslo 11

Clock Policy (obrázek 2)



umístění stránky číslo 11

RP – Working Set Clock Policy

- vylepšení nedostatku clock policy
 - při nalezení položky určené pro nahrazení se kontroluje také řídicí bit modifikace (M)
 - má-li nalezená stránka (určená pro nahrazení, $R = 0$) nastaven bit M, musí se uložit na disk, a čekalo by se na dvě diskové operace (zápis, čtení)
 - místo nahrazení změněné stránky proto algoritmus WSClock jen naplánuje diskovou operaci (uložení stránky na disk) a pokračuje ve vyhledávání
 - důsledek: nahrazuje se stránka s nulovými bity R i M
 - zdržení pouze čtením, nikoliv zápisem

Porovnání algoritmů RP

algoritmus hodnocení

ideální

neimplementovatelný

LRU

výborný, ale náročný na režii

NRU

velmi hrubá approximace LRU

FIFO

odstraňuje používané stránky

2nd chance

výrazné vylepšení FIFO

clock

vylepšená implementace 2nd chance

WSClock

efektivní vylepšení clock

Uklízení (čištění) stránek

- strategie uklízení (čištění) – **cleaning policy**
 - určuje, kdy se (modifikované) stránky uloží na disk
- **demand cleaning**
 - ukládání stránky z rámce vybraného pro nahrazení
 - proces čekající po page fault čeká na přenos 2 stránek
- **precleaning**
 - periodické ukládání stránek v dávce
 - může vést ke zbytečným zápisům
 - obsah stránky se po uložení na disk může opět změnit

Buffering stránek (1)

- rámce vybrané pro nahrazení (provádí průběžně algoritmus RP) se vkládají do dvou seznamů
 - seznam volných stránek – **free page list**
 - obsah těchto rámců nebyl od jejich načtení změněn
 - při nahrazení není třeba jejich obsah zapisovat na disk
 - seznam změněných stránek – **modified page list**
 - při nahrazení se obsah těchto rámců musí uložit
 - rámec vybraný pro nahrazení je uložen na konec příslušného seznamu a bit přítomnosti stránky v RAM je ve stránkové tabulce vynulován

Buffering stránek (2)

- nastane-li page fault, hledá se v seznamech, jestli daná stránka ještě není v RAM
 - je-li v některém seznamu, nastaví se zpět bit přítomnosti a stránka se ze seznamu odstraní
 - není-li v žádném seznamu, je stránka načtena do rámce ze začátku seznamu volných stránek a stránka je z tohoto seznamu odstraněna
 - vyprázdní-li se seznam volných stránek, obsah rámců ze seznamu změněných stránek je zapsán na disk a stránky se přeřadí do seznamu volných

Volba velikosti resident set

- pevná alokace – **fixed-allocation policy**
 - procesu je při nahrávání vyhrazen pevný počet rámců RAM (podle kritérií)
 - rovné či proporcionální rozdělení rámců mezi procesy
 - při page fault, se musí uvolnit **rámec stejného procesu**
- proměnná alokace – **variable-allocation policy**
 - počet rámců procesu se průběžně může měnit
 - zvětšuje se při vysoké frekvenci výskytů page fault
 - snižuje se při nízké frekvenci výskytů page fault
 - vyžaduje režii OS při odhadu chování procesů

Segmentace paměti

- paměť procesu je rozdělena na nespojité oblasti
 - obvykle odpovídají modulům
 - např. kód programu, kód knihoven, stack, data, heap
 - umožňuje nezávislou kompliaci modulů
- oblasti mohou být různě velké
- alokace paměti je podobná dynamickému dělení paměti
 - místo alokace pro proces alokujeme pro segment (modul procesu, knihovnu)

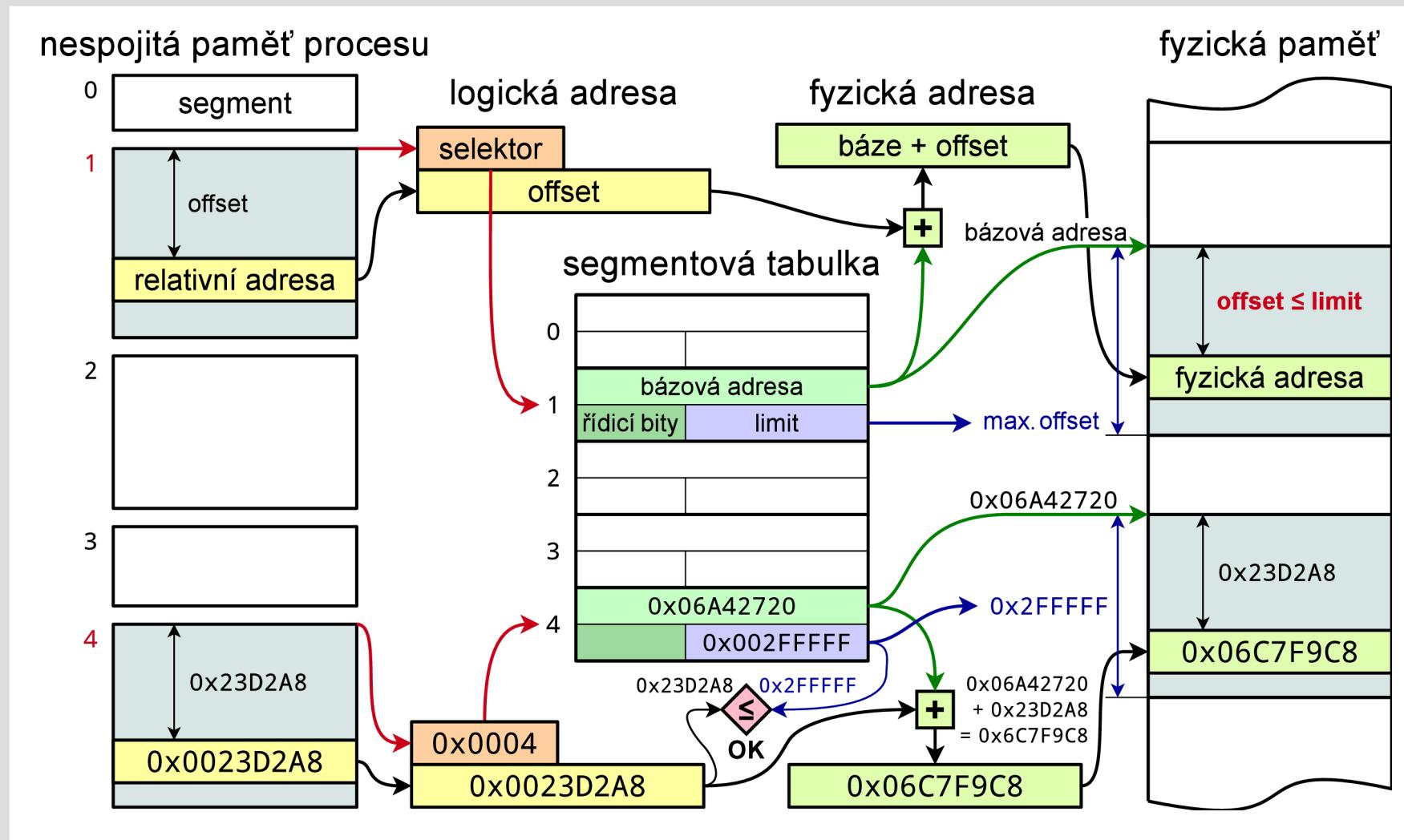
Segmentace paměti – vlastnosti

- zjednodušuje adresaci datových struktur
- zjednodušuje sdílení mezi procesy
 - procesy sdílejí celé segmenty (např. kód)
- logická adresa se skládá z **čísla segmentu** a **offsetu** (relativní adresa vzhledem k začátku)
- OS udržuje pro každý proces tabulku umístění segmentů v paměti společně s jejich délkami
 - implicitně řeší problém ochrany
 - položky tabulky obsahují také řídicí byty

Segmentace – registry CPU (x86)

- bázové (segmentové) registry
 - CS – kódový segment
 - SS – zásobník (stack segment)
 - DS, ES, FS, GS – (extra) datový segment
- offsetové registry – relativní adresa k bázi
 - SP – stack pointer – vrchol zásobníku
 - BP – base pointer – typicky adresuje lokální proměnné na zásobníku
 - SI, DI – source, destination index
 - BX – obecný registr používaný i jako offset

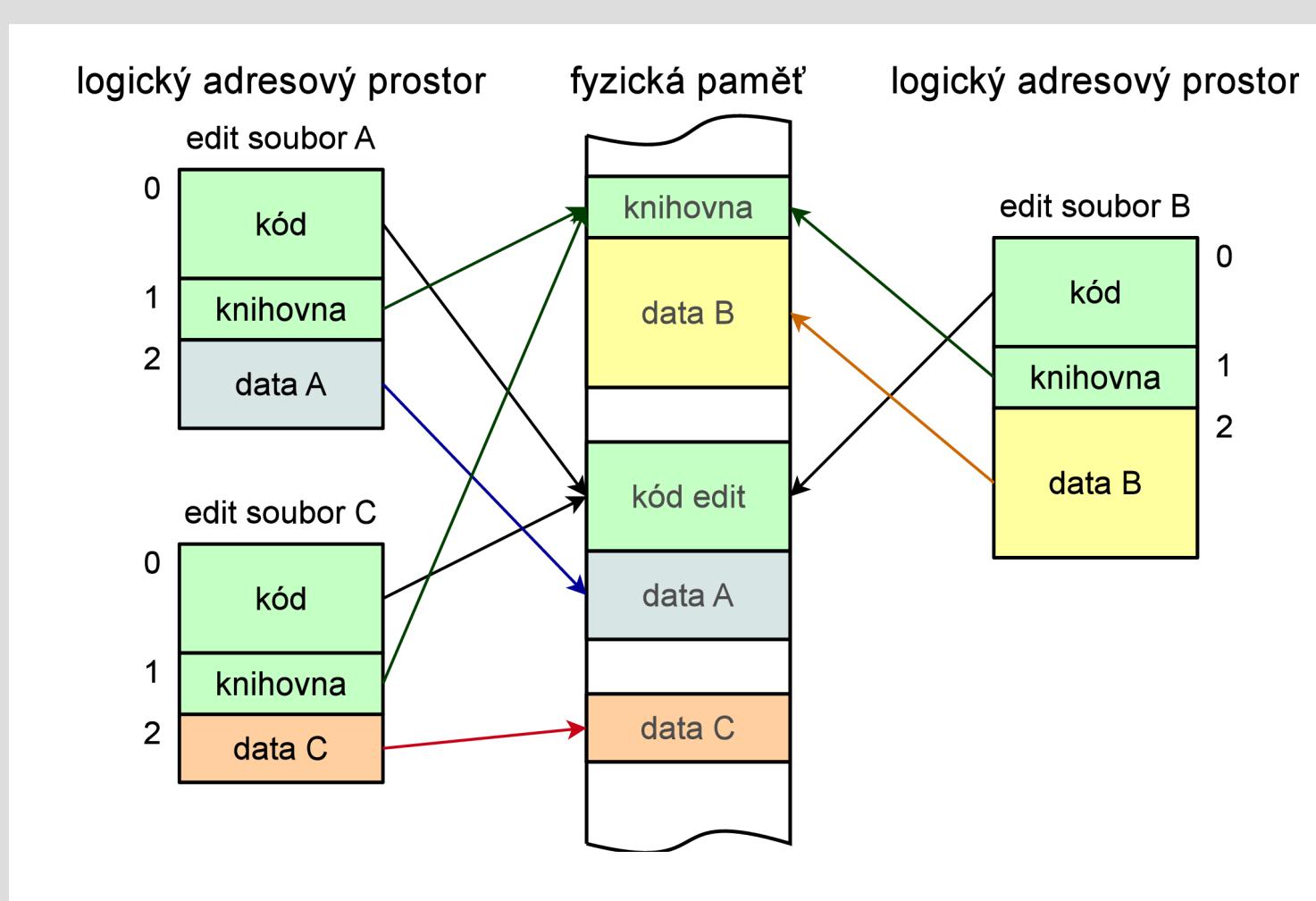
Segmentace – převod virtuální adresy na fyzickou (obrázek)



Segmentace – sdílení segmentů

- moduly procesů mohou být snadno sdíleny
- segmentové tabulky procesů mohou odkazovat na stejné segmenty
- procesy pak sdílejí celý segment
 - výhodné pro sdílení kódu procesů
 - vhodné i pro sdílení dat mezi procesy
 - logičtější než sdílení jednotlivých stránek

Segmentace – sdílení (obrázek)



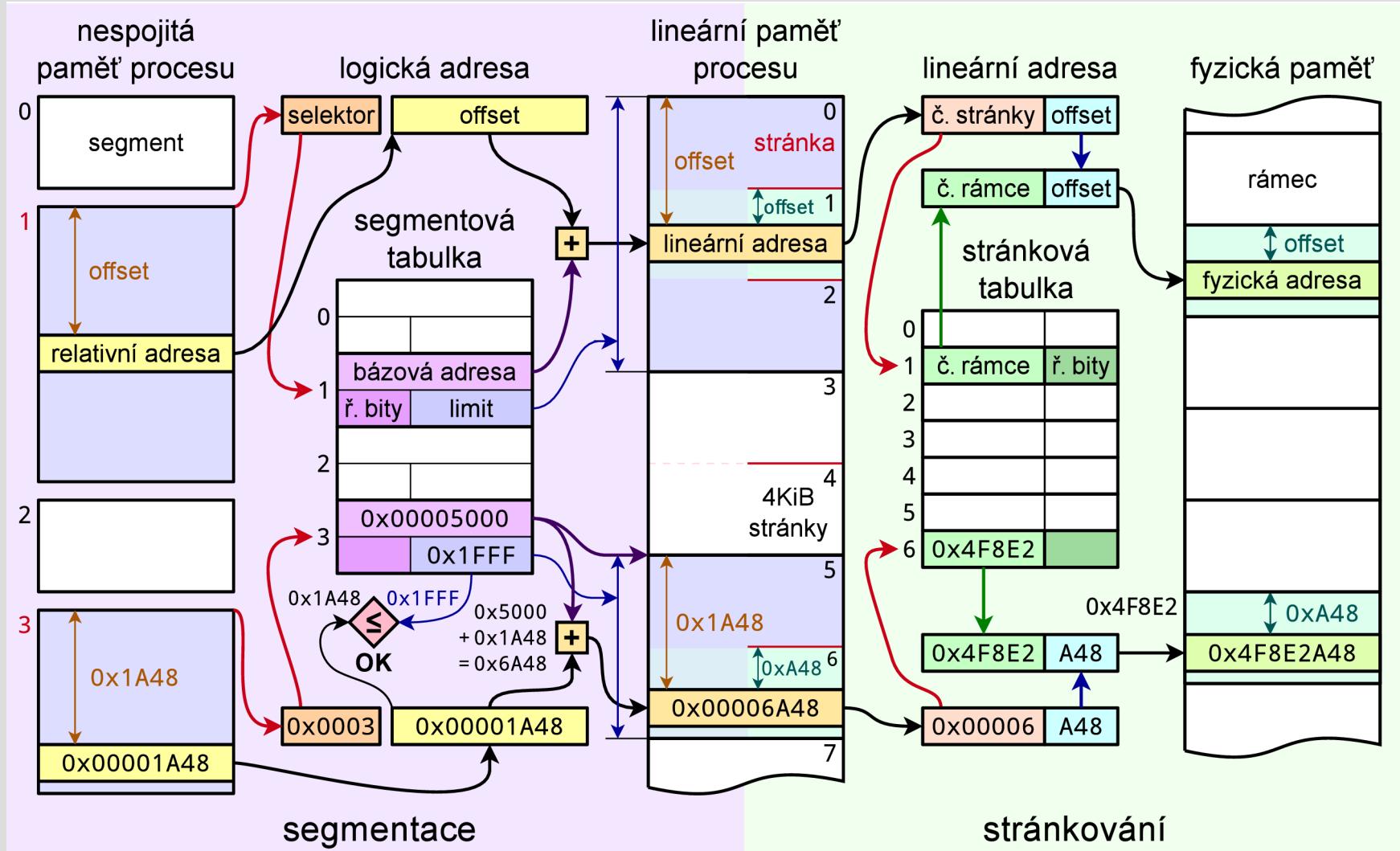
Segmentace – hodnocení

- výhody
 - přirozená alokace paměti – bez vnitřní fragmentace
 - segmenty mohou být zaváděny na vyžádání
 - sdílení segmentů jako logických celků procesu
 - přirozené řešení ochrany paměti
- nevýhody (podobné jako u dynamického dělení)
 - velké segmenty, horší hospodaření s pamětí
 - vnější fragmentace
 - nespojitá paměť – netransparentní pro programátora
 - důsledek: **segmentace je spíše na ústupu**

Kombinace segmentace se stránkováním

- rozdelením segmentů na stránky
 - se odstraní vnější fragmentace
 - segmenty mohou růst bez nutnosti přesunu v RAM
- proces pak má
 - jednu segmentovou tabulkou
 - stránkovou tabulkou
 - buď pro každý segment,
 - nebo jednu společnou
- ochrana a sdílení: na úrovni segmentů / stránek

Segmentace a stránkování – převod virtuální adresy (obrázek)



Architektura IA-32

- *Intel 64 and IA-32 Architectures: Software Developer's Manual: Volume 3A: System Programming Guide, Part 1* [online]. Intel, September 2015, c 1997–2015 [cit. 2015-09-28]. Odkaz: <intel.com>.
- podporuje segmentaci
 - poskytuje izolaci (ochranu) modulů
 - v chráněném režimu **nelze vypnout** (Ize resetem)
- podporuje stránkování
 - virtuální paměť typu demand-paged
 - implementuje také izolaci procesů

IA-32 – segmentace

- segmentace poskytuje rozdělení procesorem adresovatelné paměti (**lineární adresový prostor**) do chráněných segmentů
 - segmenty jsou učeny pro kód, data, stack procesů nebo pro systémové struktury (TSS, LDT apod.)
 - každý proces má svou vlastní sadu segmentů
 - segmenty zajišťují izolaci procesů a umožňují nastavit omezení možných operací
 - čtení, zápis, provádění

IA-32 – segmentace – adresa

- **logická adresa** (vzdálený ukazatel, **far pointer**)
 - **selektor** – unikátní identifikátor segmentu (16 bitů)
 - 13 bitů index, 1 bit Table Indicator (GDT/LDT), 2 bity RPL
 - index do globální / lokální tabulky deskriptorů (GDT, LDT)
 - deskriptor – položka GDT / LDT – určuje bázovou adresu segmentu, jeho velikost a práva na něm
 - RPL (Requested Privilege Level) – stupeň ochrany (0–3)
 - **offset** – relativní adresa vzhledem k bázové adrese
 - 32 bitů v režimu IA-32, 64 bitů v režimu IA-32e
 - bázová adresa + offset = lineární adresa
 - logická adresa (nespojitého prostoru procesu) se převádí na lineární adresu (do spojitého prostoru)

IA-32 – basic flat model

- **basic flat model** (plochý model paměti)
 - OS i procesy mají přístup k celé nesegmentované paměti
 - musejí se vytvořit dva popisovače segmentů
 - kódový segment (registrový CS) – nelze zapisovat
 - datový segment (registery DS, SS, ES, FS, GS)
 - oba segmenty mapují celý adresový prostor (4 GiB)
- používáno v 64bitovém režimu IA-32e (x86_64), který de facto nepoužívá segmentaci
 - ignorují se bázová adresa (je vždy 0), limit i atributy

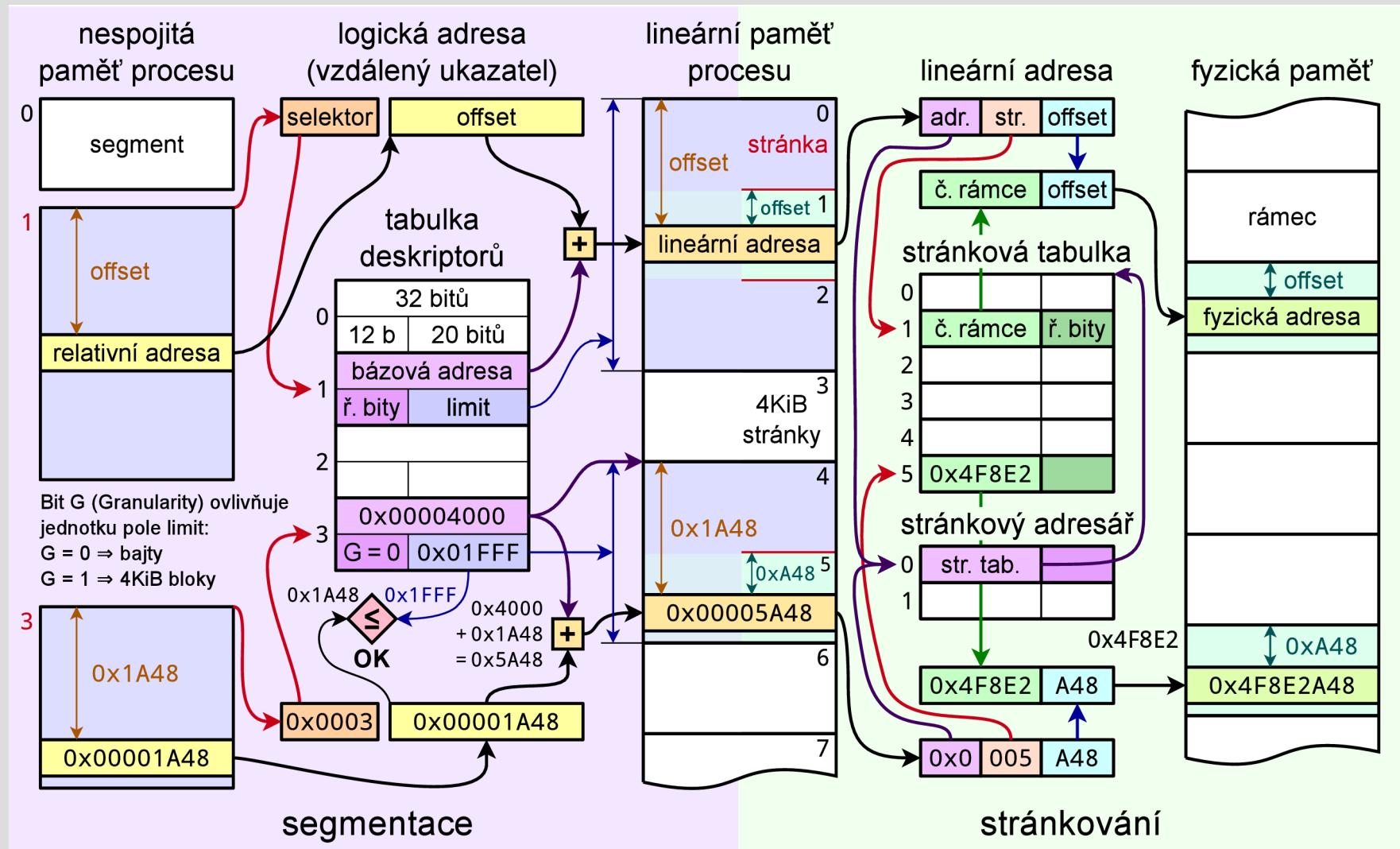
IA-32 – protected flat model

- **protected flat model** (chráněný plochý model)
 - podobné basic flat model, pouze limitní registry jsou nastaveny na velikost skutečně instalované fyzické paměti
 - výjimka ochrany je generovaná při přístupu do neexistující paměti
 - minimální ochrana paměti
 - další mechanismy ochrany paměti a izolaci lze nastavit použitím stránkování
- používáno 32bitovými systémy Linux i Windows

IA-32 – multi-segment model

- **multi-segment model** (vícesegmentový model)
 - plné využívání možností segmentace
 - paměť procesu je rozdělena na několik segmentů
 - každý proces má segmentovou tabulku s informacemi o segmentech (adresa, limit, práva)
 - segmenty mohou mít různá oprávnění včetně provádění určitých operací – stupně ochrany (ring levels)
 - přístup a práva kontroluje hardware
- používáno systémem OS/2

IA-32 – segmentace a stránkování, převod virtuální adresy (obrázek)



Posixové sdílení paměti

- hlavičkové soubory

```
#include <sys/types.h>
#include <sys/mman.h>
#include <sys/stat.h>    // pro konstanty S_I*
#include <fcntl.h>        // pro konstanty O_*
```

- knihovna librt – nutnost linkování
`gcc ... -lrt ...`
- se sdíleným paměťovým objektem se pracuje podobně jako se souborem – `shm_open(3)`
- objekt lze připojit do paměti procesu – `mmap(2)`

Sdílení paměti – vytvoření oblasti

- vytvoření nebo otevření – `shm_open(3)`

```
int shm_open(const char *name, int oflag,  
mode_t mode);
```

- vytvoří sdílený paměťový objekt – podobné `open(2)`
 - `oflag` – způsob otevření (`O_CREAT`, `O_RDWR`, ...)
 - `mode` – přístupová práva (`S_IRWXU`, `S_IRUSR`, ...) – respektuje se nastavení `umask(2)`
- přenositelnost: jméno začíná / (další / neobsahuje)
- vrací **memory descriptor** – ten se používá dále
 - nastavení velikosti nového objektu – `ftruncate(2)`
 - promítnutí objektu do paměti procesu – `mmap(2)`

Sdílení paměti – velikost, zrušení

- nastavení velikosti sdílené paměti – ftruncate(2)

```
#include <unistd.h>
#include <sys/types.h>

int ftruncate(int fd, off_t length);
```

- nastaví velikost objektu daného **fd** na **length**
- nově alokovaná paměť je inicializovaná nulami

- zrušení – shm_unlink(3)

```
int shm_unlink(const char *name);
```

- podobné unlink(2) – odstraní jméno objektu
- po odpojení objektu se uvolní i paměť

Sdílení paměti – promítnutí objektu do paměti procesu (1)

- promítnutí objektu do paměti procesu – mmap(2)

```
#include <sys/mman.h>

void *mmap(void *start, size_t length,
           int prot, int flags, int fd, off_t offset);
```

- vrací adresu, na kterou jádro OS zobrazilo obsah souboru či sdílenou paměť s deskriptorem **fd** o velikosti **length** od pozice **offset**
- pokud **start** není **NULL**, požaduje se tato adresa
 - jádro bere parametr pouze jako vodítko (tip)
- **start** i **offset** musí být na hranici stránky

Sdílení paměti – promítnutí objektu do paměti procesu (2)

- promítnutí objektu do paměti procesu – mmap(2)

```
void *mmap(void *start, size_t length,  
           int prot, int flags, int fd, off_t offset);
```

- **prot** určuje režim přístupu
 - PROT_EXEC, PROT_READ, PROT_WRITE, PROT_NONE
 - více hodnot se nastavuje operací OR
- **flags** nastavuje mj. viditelnost změn jinými procesy
 - MAP_SHARED, MAP_PRIVATE, MAP_FIXED, ...
- při chybě vrací MAP_FAILED a nastaví errno

Sdílení paměti – odpojení objektu z paměti procesu

- odpojení sdílené paměti – munmap(2)

```
#include <sys/mman.h>  
  
void *munmap(void *start, size_t length);
```

- zruší zobrazení objektu do virtuální paměti procesu
 - další odkazy do oblasti jsou neplatné a způsobují chybu
- ukončení procesu ruší zobrazení automaticky
- **uzavření deskriptoru nezruší zobrazení**
- shm_unlink také nezruší zobrazení
 - pouze odstraní jméno objektu

Paměťově orientované vstupně-výstupní operace

- memory-mapped IO – mmap(2)
- umožňuje k blokům souboru přistupovat jako k blokům paměti
 - zjednodušuje přístup k souboru
 - soubor je načítán do paměti jádrem OS metodou demand-paging
 - více procesů může promítnout do své paměti stejný soubor
 - stránky pak mohou být mezi procesy sdíleny

Sdílení paměti System V IPC

- alokace – `shmget(2)`

```
#include <sys/ipc.h>
#include <sys/shm.h>

int shmget(key_t key, size_t size, int shmflg);
```

- vrátí identifikátor sdíleného paměťového segmentu podle klíče `key`, při chybě vrací `-1` (nastaví `errno`)
- nový sdílený segment o velikosti `size` (zvětšený na násobek `PAGE_SIZE`) je vytvořen, pokud
 - `key = IPC_PRIVATE` (lepší název by byl `IPC_NEW`) nebo
 - `key ≠ IPC_PRIVATE`, segment neexistuje a `shmflg` obsahuje `IPC_CREAT`

Sdílení paměti System V – alokace

- alokace (pokračování) – `shmget(2)`

```
int shmget(key_t key, size_t size, int shmflg);
```

- nově alokovaný segment je vyplněn nulami
- struktura `shmid_ds` je naplněna – viz `shmctl(2)`
 - vlastník a skupina podle volajícího procesu, práva a velikost podle parametrů, čas modifikace na aktuální, zbytek parametrů je vynulován (včetně `shm_nattach`)
 - práva lze specifikovat dolními 9 bity v `shmflg`
 - je-li dáno `IPC_CREAT` a `IPC_EXCL` a segment existuje, vrací chybu (`EEXIST`)

Sdílení paměti System V – operace

- operace se sdílenou pamětí – `shmop(2)`

```
#include <sys/ipc.h>
#include <sys/shm.h>

void *shmat(int shmid, const void *shmaddr,
            int shmflg);
```

- připojí segment sdílené paměti s id **shmid**
 - adresu necháme zvolit systémem zadáním **NULL**
- odpojí segment paměti předtím připojený **shmat**
- sníží počet odvolávek – **shm_nattach**

Sdílení paměti System V – ovládání

- ovládání segmentu sdílené paměti – `shmctl(2)`

```
#include <sys/ipc.h>
#include <sys/shm.h>

int shmctl(int shmid, int *cmd,
           struct shmid_ds *buf);
```

- vykoná příkaz `cmd` na sdíleném segmentu
 - `IPC_STAT` (zjištění info), `IPC_SET` (nastavení práv), `IPC_RMID` (nastavení značky pro odstranění segmentu)
- datová struktura `shmid_ds` obsahuje
 - práva, vlastníka, velikost, počet připojení, časy (připojení, odpojení, změny), PID (alokátor, poslední přístup), ...