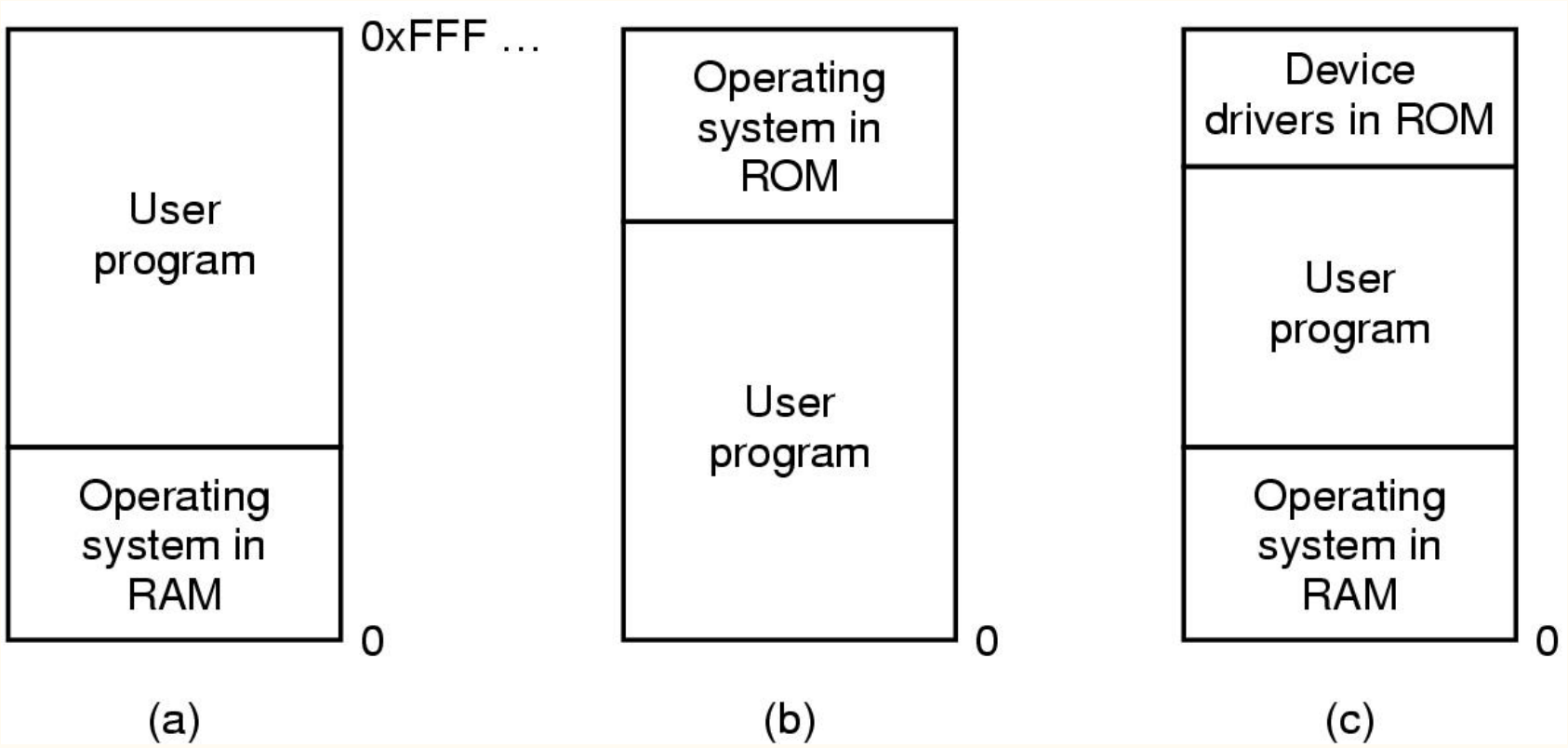
Jednoduchá správa pamäte

- Spravuje len fyzickú pamäť (nepoužíva virtualizáciu)
  - Monoprogramming
  - Multiprogramming s fixným rozdelením pamäte
  - Mutliprogramming s variabilným rozdelením pamäte
  - Swapping
  - Algoritmy jednoduchej správy pamäte

Autor: Peter Tomcsányi
Niektoré práva vyhradené v zmysle licencie Creative Commons
http://creativecommons.org/licenses/by-nc-sa/3.0/

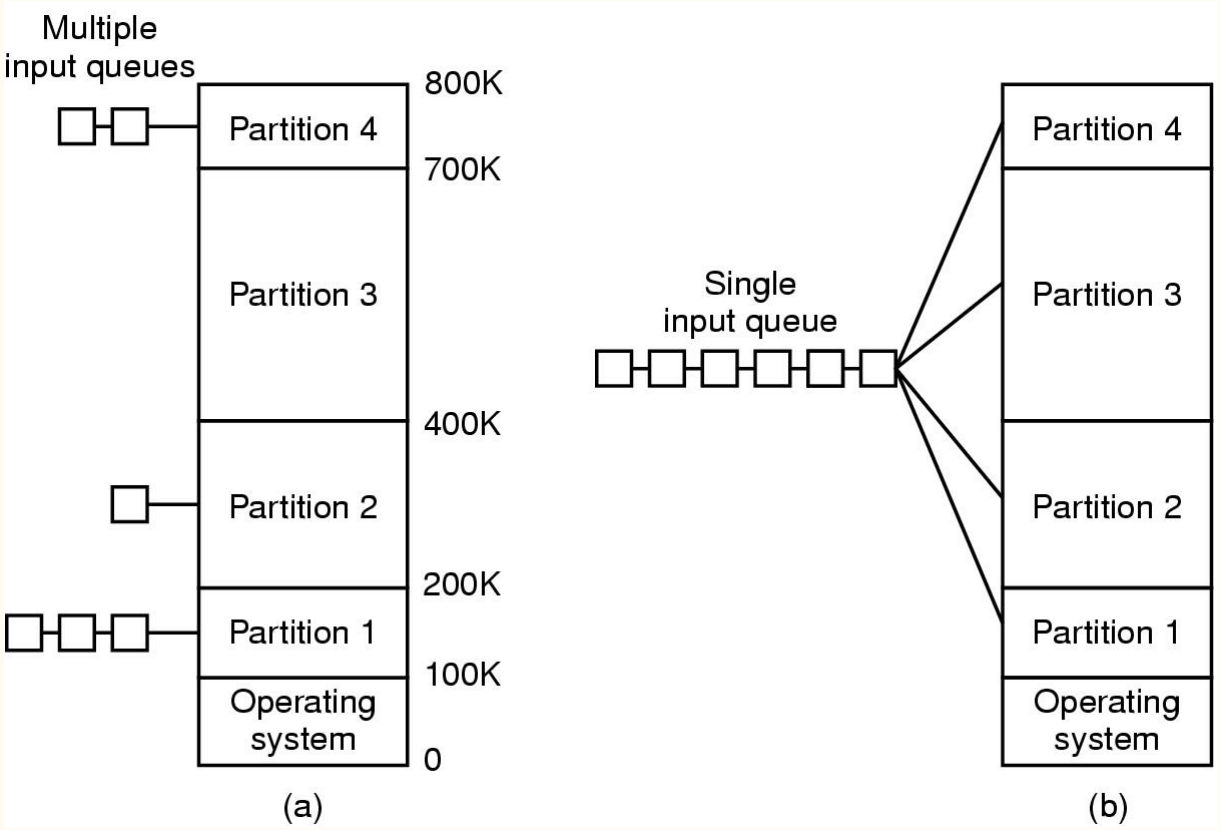
Použité obrázky z učebnice:
Andrew. S. Tanenbaum, Structured Computer Organization
http://www.cs.vu.nl/~ast/books/

Monoprogramming



- Časť pamäte zaberie operačný systém
- Zvyšok je k dispozícii jedinému procesu

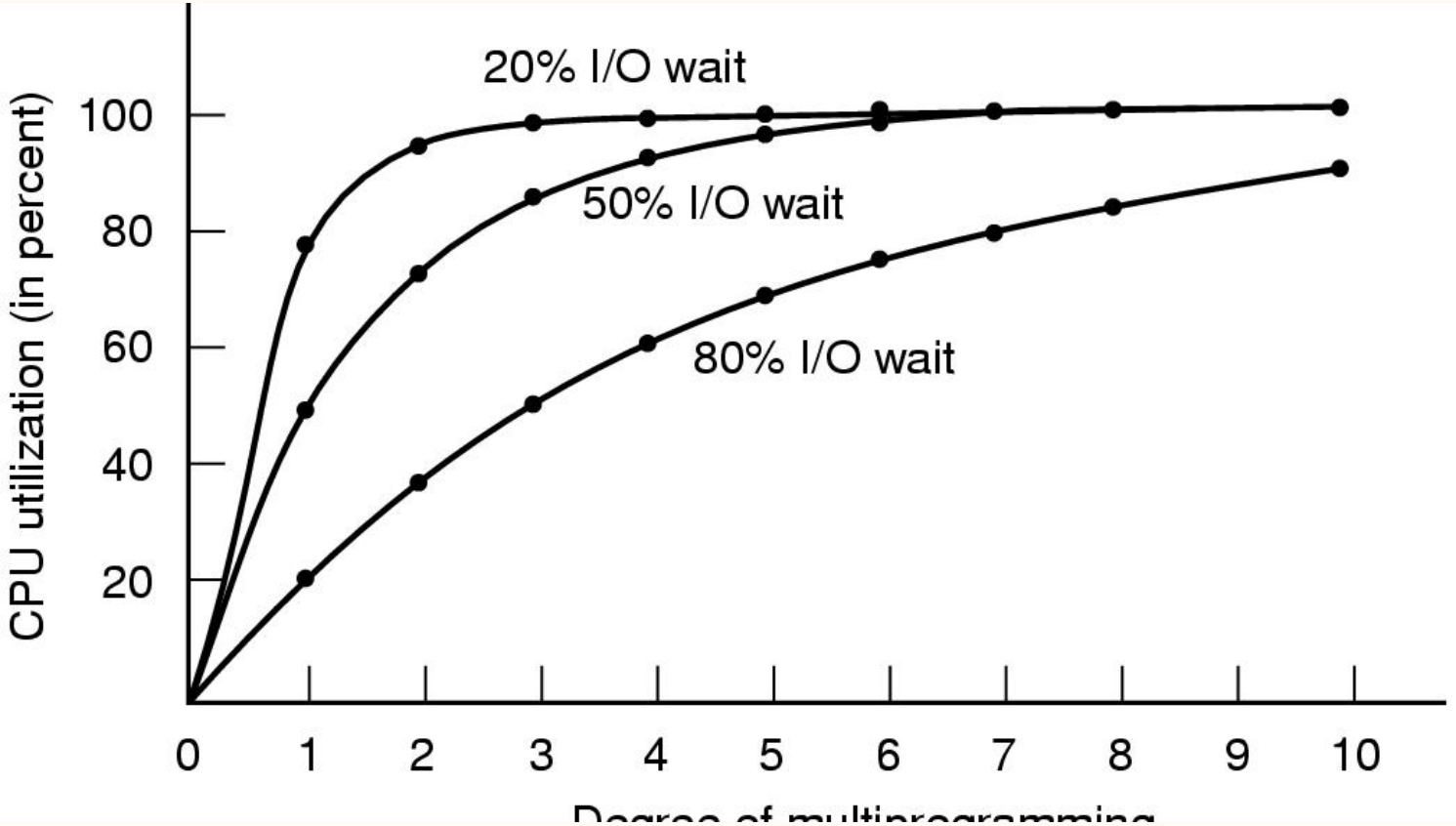
Multiprogramming s pevným rozdelením pamäte



(a) zvláštny front pre každý oddiel

(b) jeden front

Multiprogramming a využitie CPU



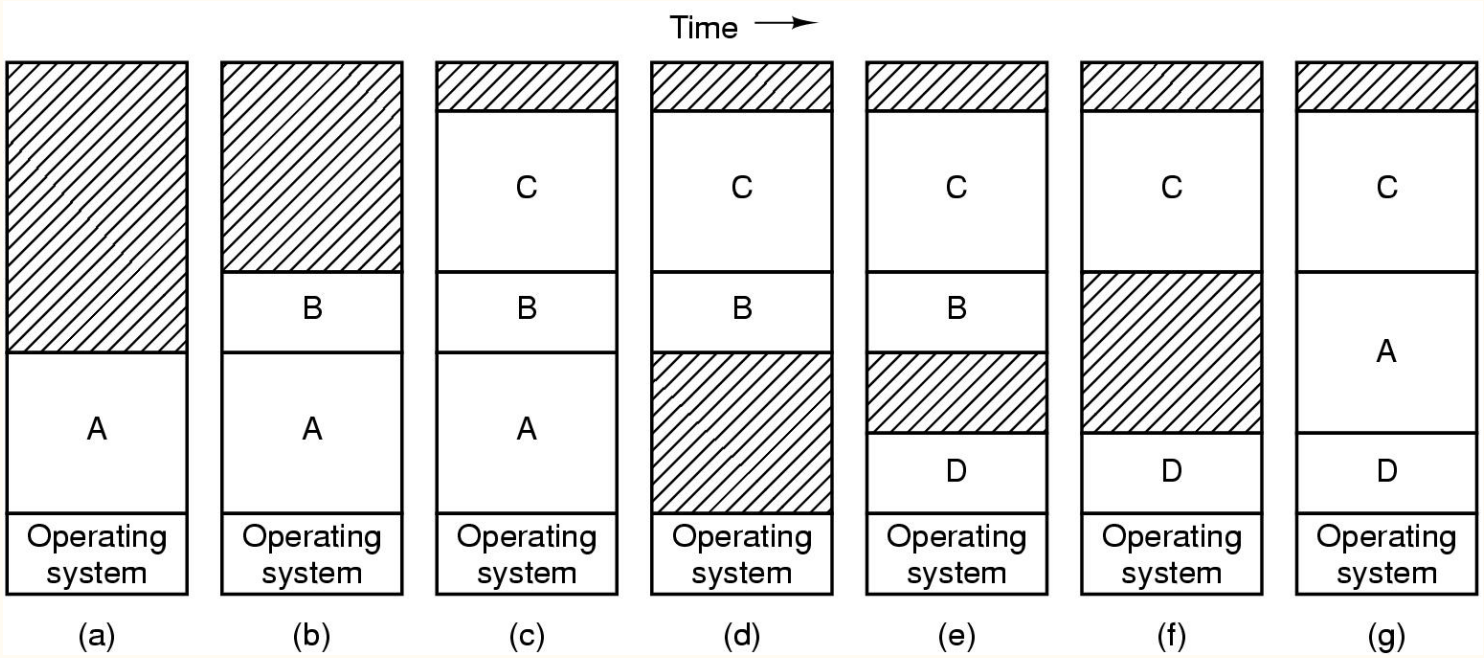
Na osi X je Počet procesov

Relokácia a ochrana

- **Relokácia** - Program môže byť umiestnený pri behu v inej časti pamäte než pri kompilácii, navyše sa môže počas behu v pamäti premiestniť (v prípade swappingu, viď ďalšiu stranu). Program by preto nemal byť fixovaný na pevné adresy
  - Dá sa realizovať ak existuje na úrovni strojového kódu adresovanie typu Bázový register+posun
  - Alebo musíme v súbore s programom mať aj informáciu o tom, na ktorých miestach sú adresy alebo smerníky a tie potom pri zavedení programu do pamäte upraviť. Také sú napríklad súbory typu .exe v MS-DOS aj vo Windows.
- **Ochrana** - Pamäť procesu by mala byť chránená pred neoprávneným prístupom z iného procesu
  - Staršie procesory mali metódu zámok-kľúč
  - Iná metóda je dvojica registrov na uloženie začiatku a konca pamäte práve bežiaceho procesu
- Virtuálna pamäť rieši oba problémy oveľa elegantnejšie

Mutiprogramming s premenlivým rozdelením pamäte, Swapping

Pri swappingu sa v prípade nedostatku fyzickej pamäte odkladá celá pamäť niektorého procesu na disk aby uvoľnila miesto pre pamäť iného procesu



Nový proces A (a), nový proces B (b) a C (c). Nový proces D už nemá miesto, preto sa A odloží (vyswapuje) na disk (d) a proces D dostane časť jeho miesta (e). Proces B skončí (f) a proces A sa vráti do pamäte na iné miesto (g).

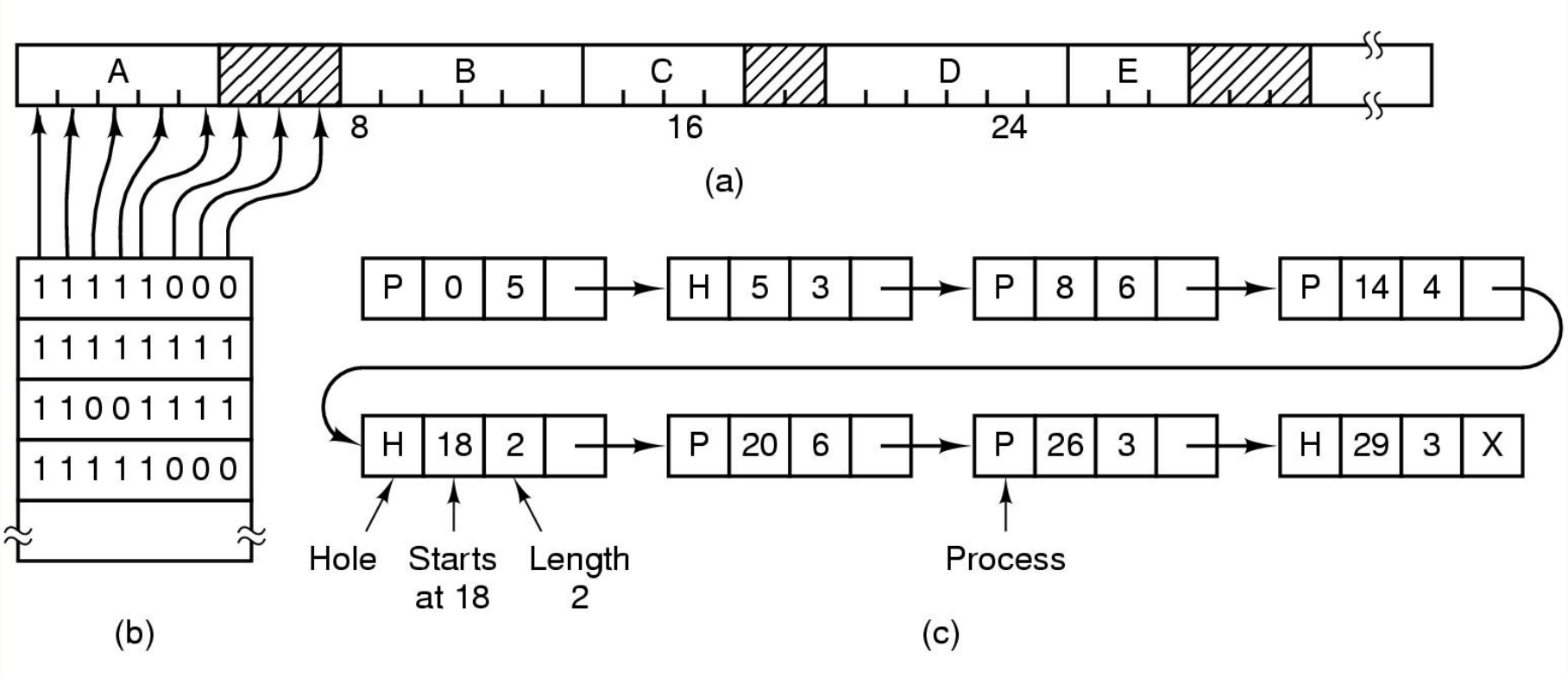
Fragmentácia a kompaktácia

- **Fragmentácia** (vonkajšia) nastáva keď sa voľná pamäť rozdrobí na veľa malých kúskov. Dohromady je veľa voľnej pamäte ale nie je spojitá
- Mnohé systémy s tým už nevedia spraviť nič
- Iné môžu pamäť **kompaktovať**, teda posunúť obsadené úseky tak aby vznikol jeden spojitý voľný úsek
- **Kompaktácia** sa dá robiť len ak sú všetky adresy relatívne alebo ak počas nej dokážeme modifikovať všetky adresy skokov a smerníky.

Algoritmy jednoduchej správy pamäte

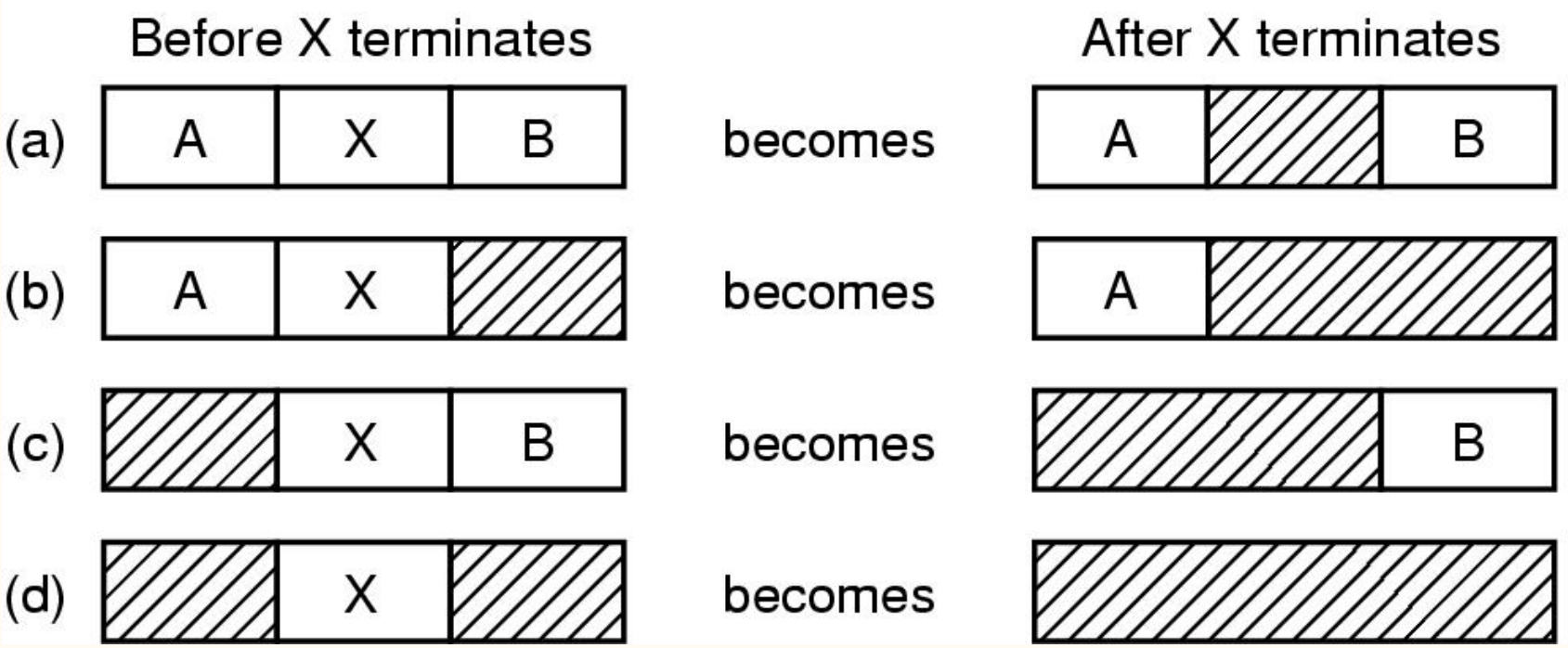
- Ide vlastne o implementáciu dvoch procedúr:
  - GetMem(Pointer,Size);
  - FreeMem(Pointer,Size);
- Algoritmy musia zvoliť ako budú vnútorne reprezentovať prázdne a obsadené úseky pamäte
- Základné metódy reprezentácie:
  - Bitová mapa
  - Spájaný zoznam
  - Kombinácia bitovej mapy, spájaného zoznamu a samotnej prideľovanej pamäte

Bitová mapa a spájaný zoznam



- Hľadanie prázdneho úseku (pre GetMem) v bitovej mape môže byť pomalé - treba prejsť všetky bity
- V bitovej mape rýchlejšie nájdeme informáciu o obsadenom úseku podľa adresy (pre FreeMem), čas však stratíme nulovaním všetkých bitov úseku
- Spájaný zoznam má ťažko predvídateľné pamäťové nároky, nevieme koľko miesta mu rezervovať
- Jeden bit bitovej mapy zobrazuje B bajtov (napr 32). Prideľujeme teda celé násobky B aj keď proces žiada menej. Vzniká **vnútorná fragmentácia**

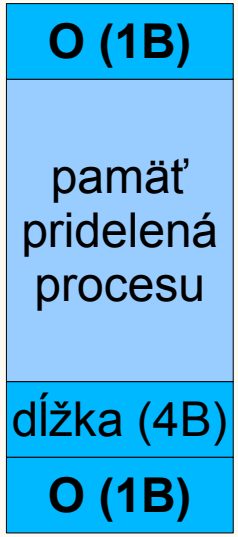
Spájanie úsekov



- Pri implementácii pomocou spájaných zoznamov netreba zabudnúť na spájanie susediacich voľných úsekov
- Preto je častejšie používaný dvojsmerne spájaný zoznam
- Spájanie komplikuje kód (oproti bitovej mape), ale nezhoršuje znateľne časovú zložitosť (spotrebu času)

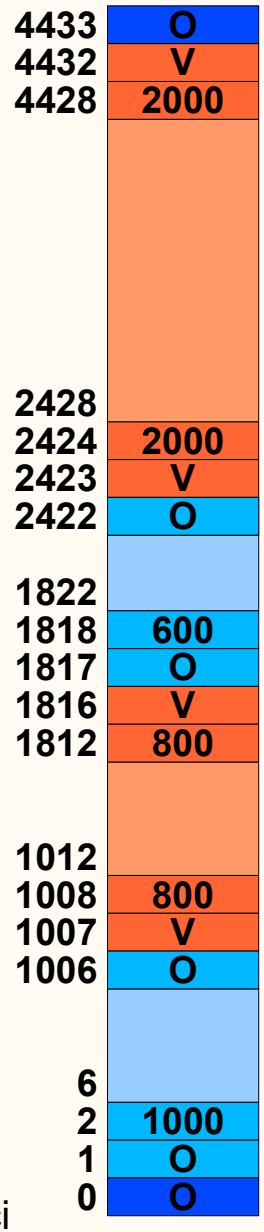
Kombinovaná metóda

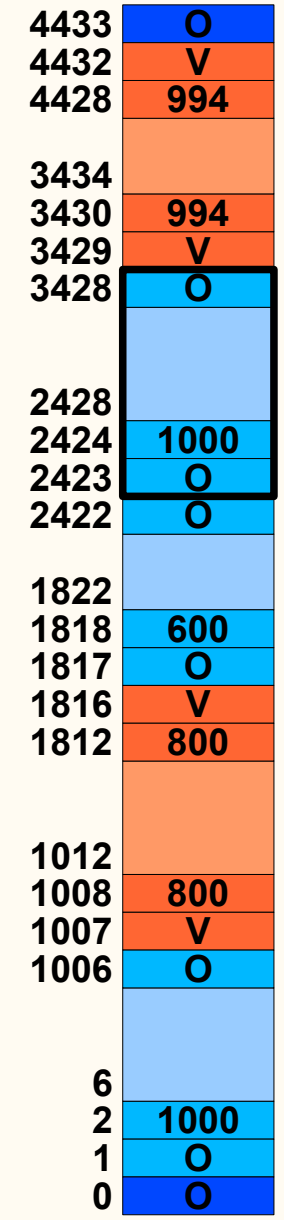
- Kombinuje pomocné dátové štruktúry a prideľovanú pamäť, preto nemusí vyhradiť zvláštnu pamäť na dátové štruktúry
- Nájdenie voľného úseku je tak rýchle ako v spájanom zozname
- Nájdenie informácií o obsadenom úseku podľa adresy je tak rýchle ako v bitovej mape, pritom netreba nulovať bity



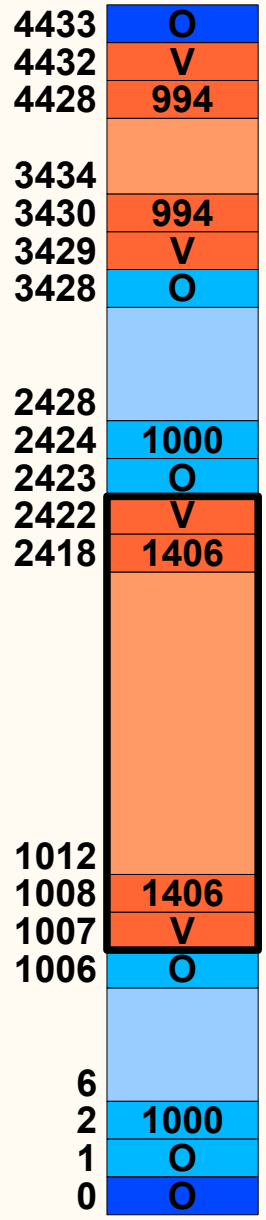
Obsadený úsek
O je jednobajtová značka na začiatku aj na konci

Voľný úsek
Zopakovaná dĺžka na konci je potrebná pri spájaní úsekov





po **GetMem(P,1000)**
P=2428



po **FreeMem(1822,600)**

Zhrnutie metód reprezentácie

N je veľkosť celej pamäte
M je počet úsekov, na ktoré je pamäť práve rozdelená (M << N)
K je veľkosť úseku, ktorý sa uvoľňuje

	bitová mapa	**spájaný zoznam**	**kombinácia**
Trvanie GetMem závisí od	N (lineárne)	M (lineárne)	M (lineárne)
Trvanie FreeMem závisí od	K (lineárne)	M (lineárne)	konštantná
Potrebná zvláštna pomocná pamäť	áno veľkosť známa	áno (nie) veľkosť neznáma	nie
Možnosť vnútornej fragmentácie	áno	nie	áno
Oddelenie prideľovanej pamäte a pomocných údajov	áno	áno/nie	nie

Ktorý úsek je vhodný?

Treba ešte vyriešiť ako nájsť vhodný úsek:

- **First fit** - prideľme prvý úsek, ktorého veľkosť je postačujúca
- **Circular First fit** (**Next fit**) - First fit môže nahromadiť na začiatku pamäte len maličké zvyšky a väčšie voľné úseky sú až ďalej. Hľadajme preto radšej od miesta kde sme posledne pridelili.
- **Best fit** - taký úsek, ktorého veľkosť je najbližšia žiadanej - ostane najmenší zvyšok. Problém sú práve tie maličké zvyšky, ktoré už nikto nechce.
- **Worst fit** - Opak Best fit. V praxi nie je použiteľný
- **Quick fit** (**Fast Fit**) - združovať prázdne úseky podľa dĺžok. Môže byť veľmi efektívny ak procesy žiadajú len niekoľko málo rôznych dĺžok úsekov.