



PB153

# Operační systémy a jejich rozhraní

## Virtuální paměť



# Základní principy

- Virtuální paměť
  - separace LAP a FAP
  - ve FAP se mohou nacházet pouze části programů nutné pro bezprostřední řízení procesů
  - LAP může být větší než FAP
  - adresové prostory lze sdílet mezi jednotlivými procesy
  - lze efektivněji vytvářet procesy
- Techniky implementace
  - Stránkování na žádost, Demand Paging
  - Segmentování na žádost, Demand Segmentation

# Běh procesů ve virtuální paměti

- Část procesu umístěnou ve FAP nazýváme „rezidentní množinou“ (resident set)
- Odkaz mimo rezidentní množinu způsobuje přerušení výpadek stránky (page fault)
- OS označí proces za „čekající“
- OS spustí I/O operace a provede nutnou správu paměti pro zavedení odkazované části do FAP
- Mezitím běží jiný proces (jiné procesy)
- Po zavedení odkazované části se generuje I/O přerušení
  - OS příslušný proces umístí mezi připravené procesy
- Překlad adresy LAP na adresu FAP se dělá indexováním tabulky PT/ST pomocí hardware CPU

# Virtuální paměť – vlastnosti

- Ve FAP lze udržovat více procesů najednou
  - čím více je procesů ve FAP, tím je větší pravděpodobnost, že stále bude alespoň jeden připravený
- Lze realizovat procesy požadující více paměti než umožňuje FAP
  - aniž se o řešení tohoto problému stará programátor / kompilátor
  - jinak bychom museli použít překryvy (overlays)

# Příklad: Linux

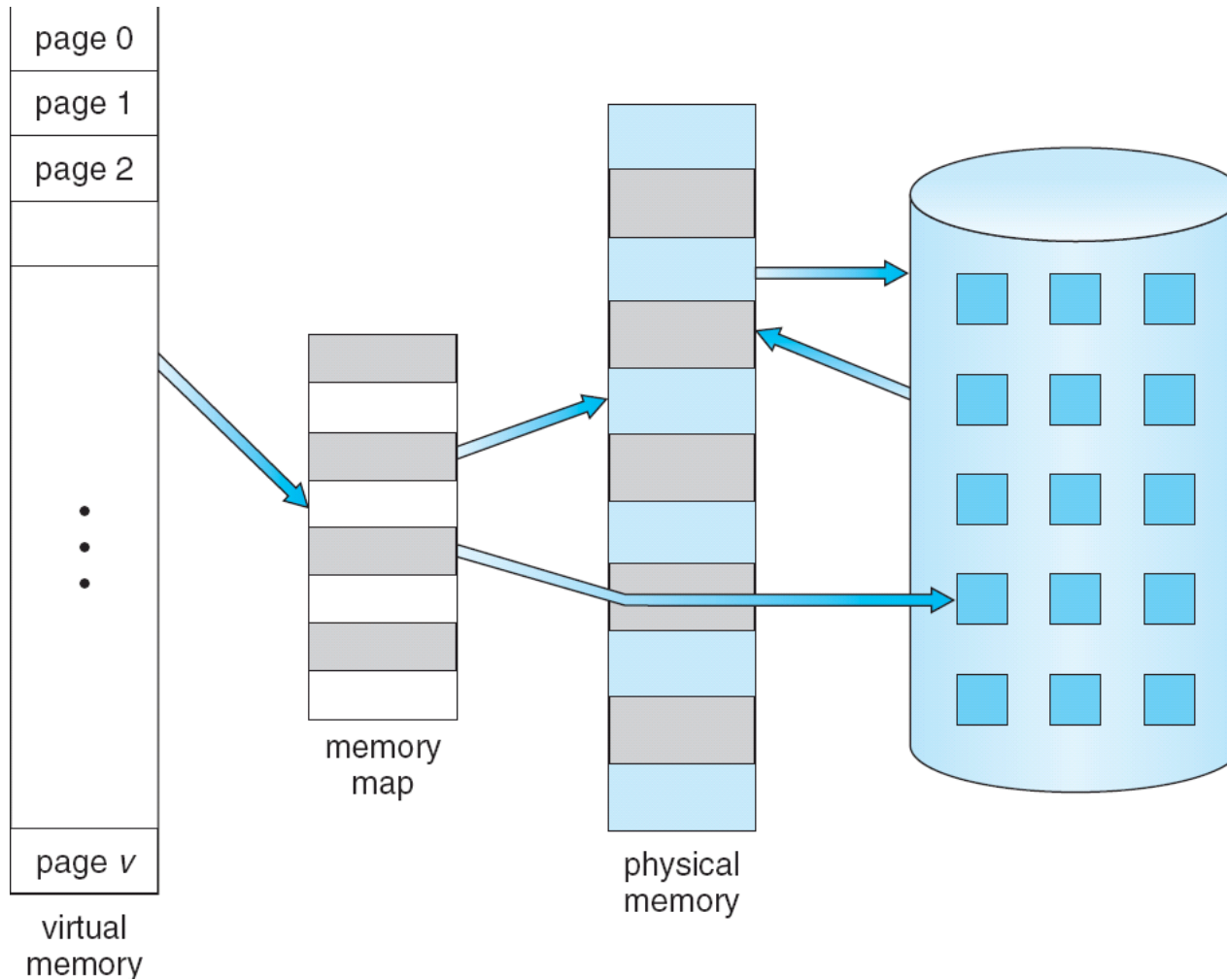
PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
4261	brandejs	17	0	149m	14m	2180	S	4.9	0.0	0:00.15	crond
4268	brandejs	19	0	41164	8072	1836	S	2.9	0.0	0:00.09	execute.pl
4135	zriha	15	0	12560	2160	808	R	2.6	0.0	0:00.58	top
31185	root	16	0	67436	15m	3052	S	2.3	0.0	0:04.82	dovecot-auth
3048	kas	15	0	103m	14m	2184	R	1.9	0.0	0:09.82	vim
9842	root	15	0	0	0	0	S	1.3	0.0	110:03.59	nfsd
6267	root	15	0	14556	532	396	S	1.0	0.0	525:51.34	mcstransd
1	root	15	0	10348	648	548	S	0.6	0.0	79:51.28	init
4236	gruzicka	15	0	47664	2192	1720	S	0.6	0.0	0:00.06	imap
9427	nscd	15	0	486m	179m	3000	S	0.6	0.1	120:59.69	nscd
13943	named	21	0	640m	39m	2212	S	0.6	0.0	929:05.56	named
97	root	10	-5	0	0	0	S	0.3	0.0	0:47.25	events/23
1708	root	15	0	0	0	0	S	0.3	0.0	66:48.99	pdflush
1777	root	15	0	50392	1484	580	S	0.3	0.0	10:44.24	dovecot
2403	root	10	-5	0	0	0	S	0.3	0.0	113:31.64	kjournald
3095	hales	16	0	153m	5408	3832	S	0.3	0.0	0:00.54	vim.Linux

- VIRT - Virtual Image (kb)
  - The total amount of virtual memory used by the task. It includes all code, data and shared libraries plus pages that have been swapped out.
- RES - Resident size (kb)
  - The non-swapped physical memory a task has used.
- SHR - Shared Mem size (kb)
  - The amount of shared memory used by a task. It simply reflects memory that could be potentially shared with other processes.

# Virtuální paměť – vlastnosti

- Uložení obrazu LAP (virtuální paměti) v externí paměti
  - nepoužívá se standardní systém souborů OS, používají se speciální metody, optimalizované pro tento účel
  - speciální partition/slice disku
- Stránkování / segmentaci musí podporovat hardware správy paměti
  - stránkování na žádost
  - segmentování na žádost
  - žádost – dynamicky kontextově generovaná indikace nedostatku
- OS musí být schopen organizovat tok stránek / segmentů mezi vnitřní a vnější paměti

# LAP větší než FAP



# Kdy zavádět stránku

- Stránkování na žádost (Demand paging)
  - stránka se zobrazuje do FAP při odkazu na ni, pokud ve FAP není již zobrazena
  - počáteční shluky výpadků stránek
- Předstránkování (Prepaging)
  - umístění na vnější paměti sousedních stránek v LAP bývá blízké („sousedí“)
  - princip lokality
  - zavádí se více stránek než se žádá
  - vhodné při inicializaci procesu



# Kam stránku zavést?

- Segmentace
  - best-, first-, worstfit
- Stránkování
  - nemusíme řešit
- Kombinace segmentování + stránkování
  - nemusíme řešit



# Kterou stránku nahradit?

- Uplatňuje se v okamžiku, kdy je FAP plný
- Typicky v okamžiku zvýšení stupně multitaskingu
- Kterou stránku „obětovat“ a vyhodit z FAP (politika výběru oběti)?
- Politika nahrazování
  - určení oběti
    - politika přidělování místa
      - kolik rámců přidělit procesu?
      - intra/extra (local/global) množina možných obětí
        - oběti lze hledat jen mezi stránkami procesu, který vyvolal výpadek stránky?
        - oběti lze hledat i mezi stránkami ostatních procesů (např. podle priorit procesů)?
  - nelze obětovat kteroukoliv stránku
    - někde rámce jsou „zamčeny“
    - typicky I/O buffery, řídicí struktury OS, ...



# Stránkování na žádost

- Stránka se zavádí do FAP jen když je potřebná
- Přínosy
  - méně I/O operací
  - menší požadavky na paměť
  - rychlejší reakce
  - může pracovat více uživatelů
- Kdy je stránka potřebná?
  - byla odkazovaná
  - legální reference
    - Nachází se ve FAP, přeloží se logická adresa na fyzickou
  - nelegální reference (porušení ochrany) → abort procesu
  - reference stránky, která není ve FAP → její zavedení do FAP
    - zprostředkovává OS

# Bit Valid-Invalid

- S každým řádkem PT je spojen bit V/I (valid/invalid)  
(1 → je ve FAP, 0 → není právě ve FAP)
  - iniciálně jsou všechny V/I bity nastaveny na 0
- Při překladu adresy
  - jestliže je bit valid/invalid roven 0, generuje se přerušení typu „page fault“

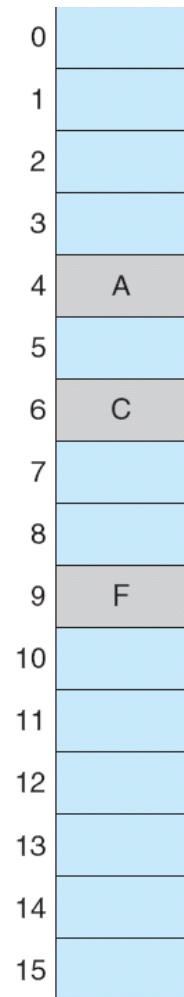
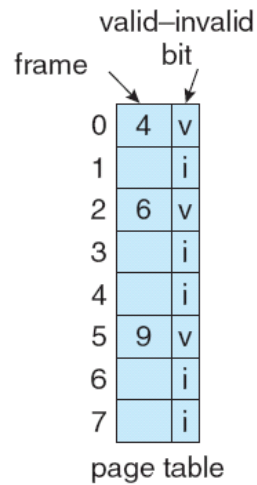
Frame #	valid-invalid bit
	1
	1
	1
	1
	0
⋮	
	0
	0

page table

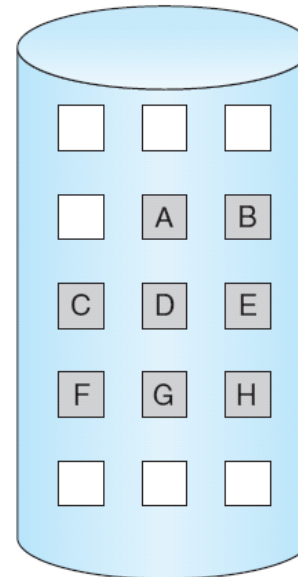
# Příklad: tabulka stránek



logical memory



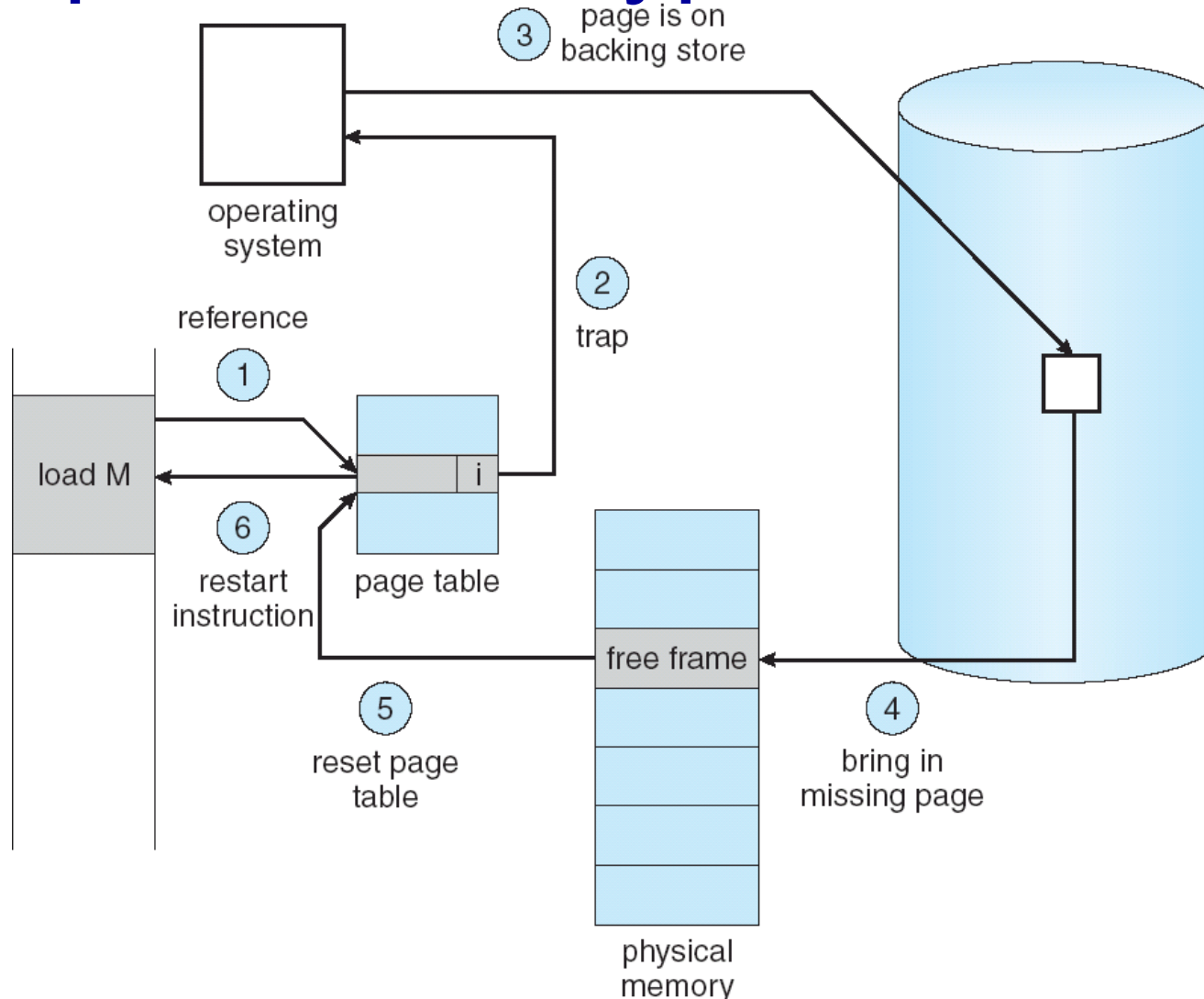
physical memory



# Výpadek stránky

- Pokud se stránka nenachází ve FAP je aktivován OS pomocí přerušení „**page fault**“ (výpadek stránky)
- OS zjistí:
  - nelegální reference → procesu je informován (např. signál SIGSEGV)
  - legální reference, ale stránka chybí v paměti → zavede ji
- Zavedení stránky
  - získání prázdného rámce
  - zavedení stránky do tohoto rámce
  - úprava tabulky, nastaven bit valid/invalid na 1
- Pak se opakuje instrukce, která způsobila „page fault“

# Zpracování výpadku stránky



# Volný rámec

- pokud není žádný rámec aktuálně označen jako volný, musíme nějaký uvolnit
  - nalezneme se „obět“ – nepotřebnou stránku ve FAP (nevíme, co je nepotřebné, můžeme pouze odhadovat)
  - je-li to nutné, před uvolněním stránky ji zapíšeme na disk
    - pokud se do ní nezapsalo, její kopie na disku je aktuální
    - zda se do stránky zapsalo zjistíme v bitu modify (dirty)
    - bit modify nastavuje HW automaticky při zápisu do stránky
- Potřebujeme algoritmus pro hledání „oběti“ a řešení náhrady
  - kritérium optimality algoritmu: nejméně výpadků stránek

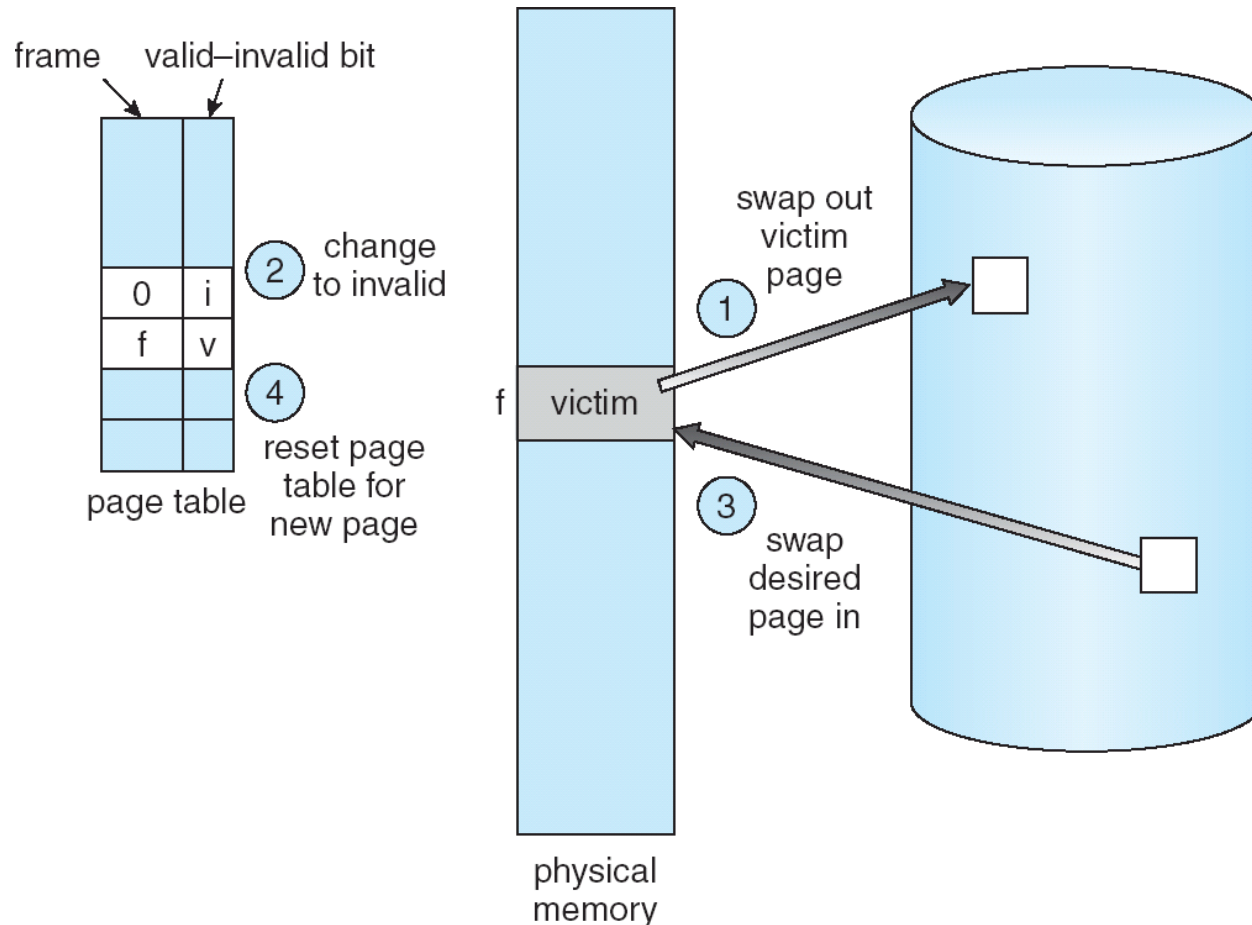




# Princip lokality

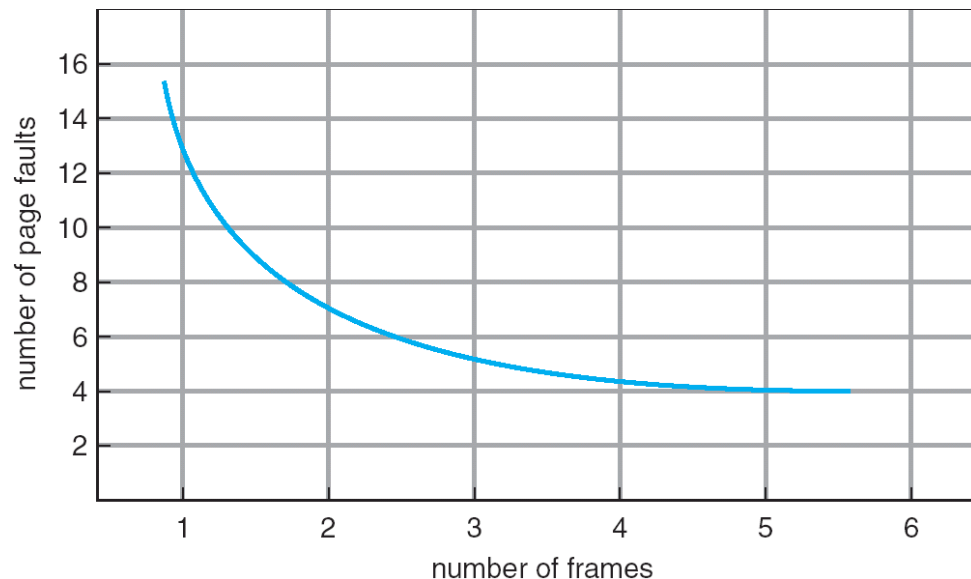
- Odkazy na instrukce programu a na dat mají tendenci tvořit shluky
- Časová lokalita a prostorová lokalita
  - provádění programu je s výjimkou skoků a volání podprogramů sekvenční
  - programy mají tendenci zůstat po jistou dobu v rámci nejvýše několika procedur (nelze jen něco volat a hned se vracet)
  - většina iterativních konstruktů představuje malý počet často opakovaných instrukcí
  - často zpracovávanou datovou strukturou je pole dat nebo posloupnost záznamů
- → lze dělat rozumné odhady o částech programu/dat potřebných v nejbližší budoucnosti
- vnitřní paměť se může zaplnit
  - něco umístit do FAP pak znamená nejdříve něco odložit z FAP

# Náhrada stránky



# Algoritmy určení oběti

- Kritérium optimality
  - nejmenší pravděpodobnost výpadku stránky
- Čím více rámců máme, tím méně často dojde k výpadku stránky



# Algoritmy určení oběti

- Pouze základní typy
  - existují desítky variant
- Optimální algoritmus
  - příští odkazy na oběť je nejpozdější ze všech následných odkazů na stránky
  - generuje nejmenší počet výpadků stránek
  - neimplementovatelný, neboť neznáme budoucnost
  - používá se jen pro srovnání
- Algoritmus LRU (Least Recently Used)
  - oběť = nejdéle neodkazovaná stránka
- Algoritmus FIFO (First-In-First-Out)
  - oběť = stránka nejdéle zobrazená ve FAP
- Algoritmus poslední šance
  - FIFO + vynechávání z výběru těch stránek, na které od posledního výběru bylo odkázáno

# Algoritmus First-In-First-Out

- Posloupnost odkazů stránek:

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

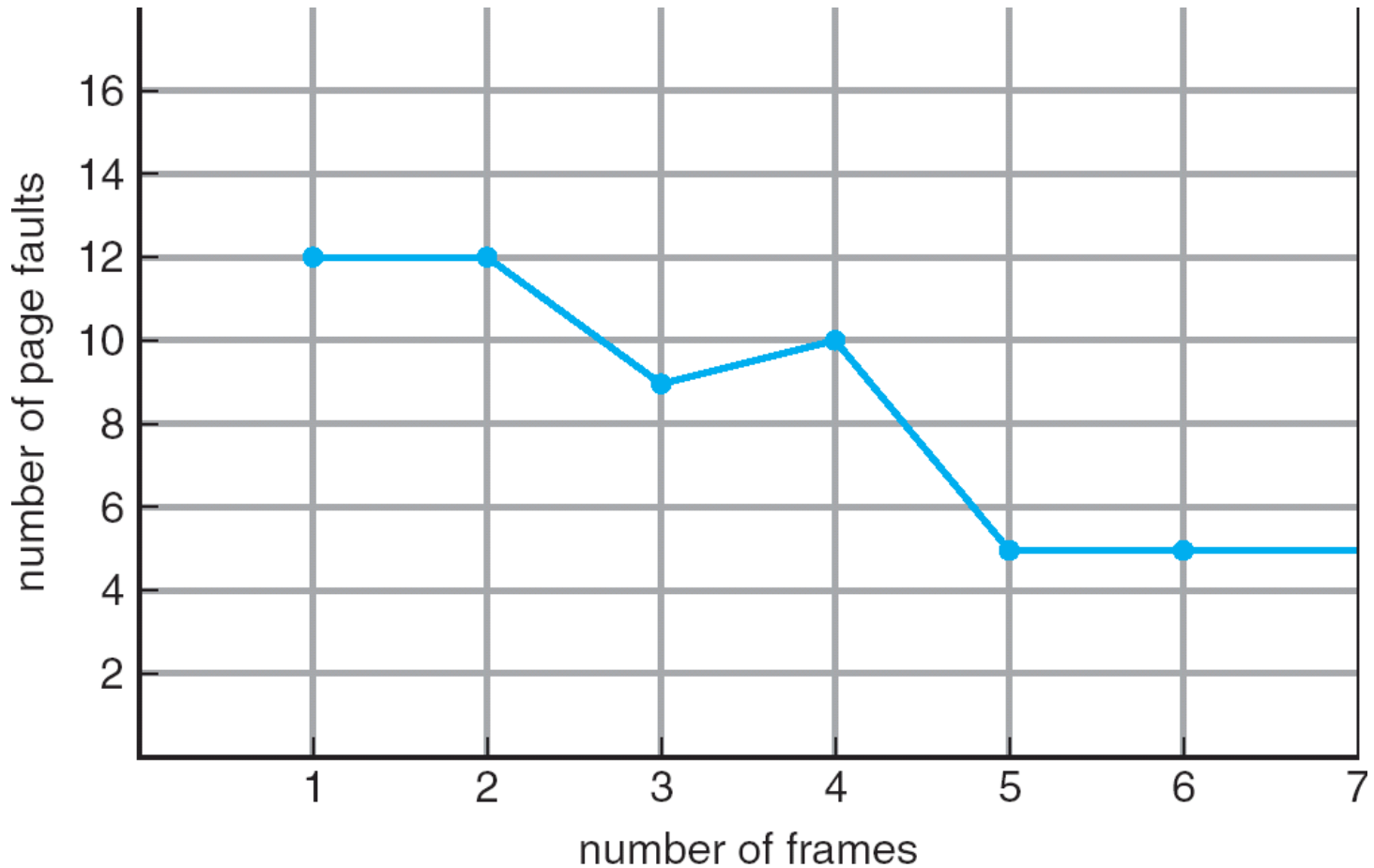
- 5 stránek
- 3 rámce (3 stránky mohou být ve FAP)

1	1	4	5	
2	2	1	3	9 výpadků
3	3	2	4	

- 4 rámce
  - Beladyho *anomálie*
  - více rámců → více výpadků

1	1	5	4	
2	2	1	5	10 výpadků
3	3	2		
4	4	3		

# Beladyho anomálie



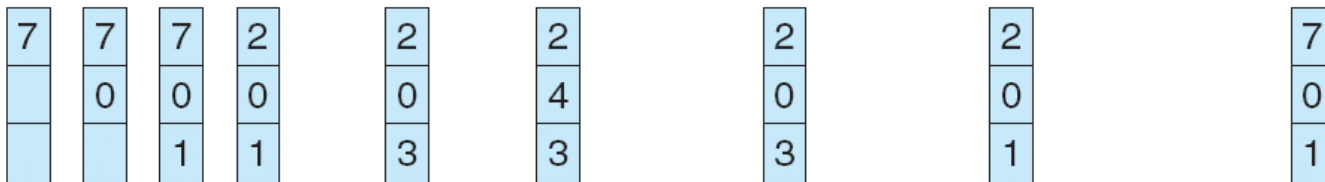
# Algoritmus First-In-First-Out

## Hodnocení FIFO strategie

- často používané stránky jsou v mnoha případech právě ty nejstarší
  - pravděpodobnost výpadku takové stránky je vysoká
  - tj. algoritmus zbytečně odkládá často používané stránky
- jednoduchá implementace
  - stačí udržovat ukazatelem vazbu do cyklického pořadí

reference string

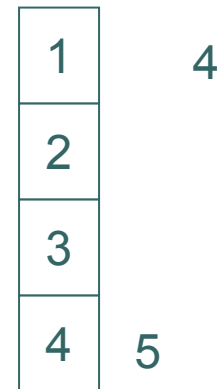
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1



page frames

# Optimální algoritmus

- Obětí je nejpozdější ze všech následných odkazů na stránky
- Potřebuje znát budoucí posloupnost referencí
- Příklad
  - proces s 5 stránkami
  - k dispozici máme 4 rámce
  - Posloupnost přístupů:  
1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
  - celkem 6 výpadků stránky



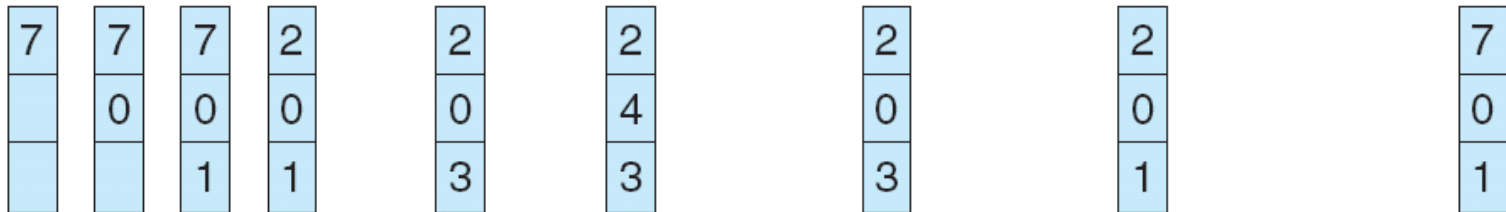


# Optimální algoritmus

## ○ Další příklad:

reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1



page frames

# Algoritmus LRU

- Obětí je nejdéle neodkazovaná stránka
  - princip lokality říká, že pravděpodobnost jejího brzkého použití je velmi malá
  - výkon blízký optimální strategii
- Příklad:
  - proces s 5 stránkami, k dispozici 4 rámce
  - 1, 2, 3, 4, 1, 2, **5**, 1, 2, **3**, **4**, **5**

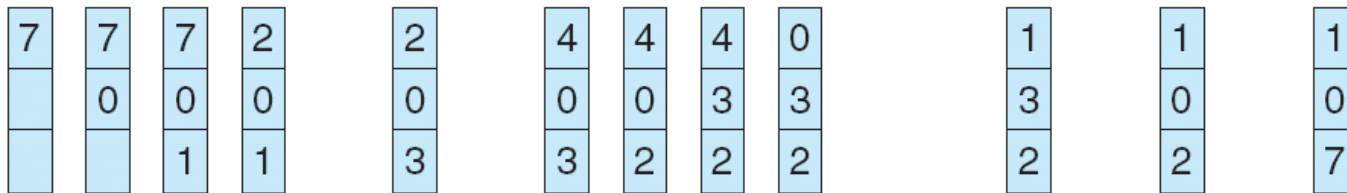
1	1	1	1	<b>5</b>
2	2	2	2	2
3	<b>5</b>	5	<b>4</b>	4
4	4	<b>3</b>	3	3

# Algoritmus LRU

## ○ Další příklad:

reference string

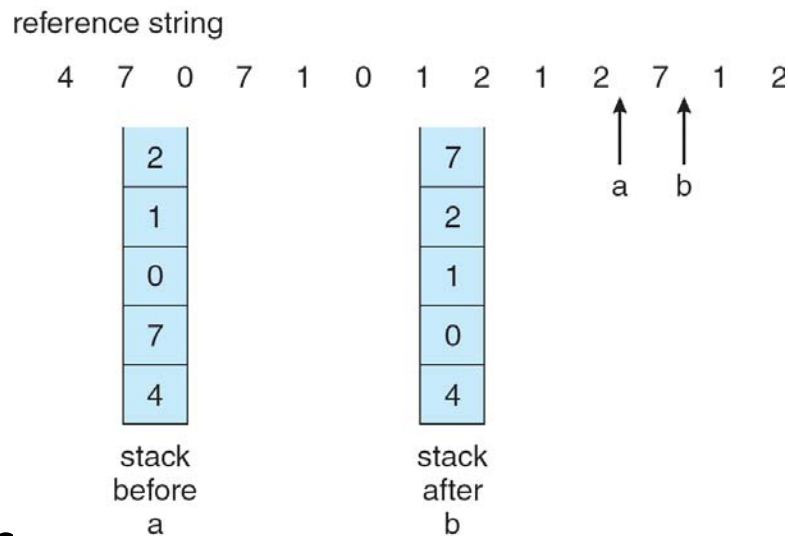
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1



page frames

# Implementace politiky LRU

- V položkách PT se udržuje (HW) čítač (sekvenční číslo nebo logický čas) posledního přístupu ke stránce
  - čítač má konečnou kapacitu
  - problém jeho přečtení
- Zásobník
  - zásobník čísel stránek, při přístupu ke stránce je položka odstraněna ze zásobníku a přemístěna na vrchol (na spodu je „obět“)
- Aproximace: iniciálně je bit nastaven na 0, při každém přístupu je nastaven bit přístupu na 1
  - neznáme pořadí přístupu ke stránkám
  - víme jen které stránky byly použity



# Druhá šance

- Také nazývané „Máš ještě šanci“
- Výběr oběti – cyklické procházení (podobně jako FIFO)
- Odkazem na stránku stránka se nastaví příznak
  - ani násobné odkazy příznak nezvyšují, jen nastavují
- Oběť, která nemá nastaven příznak a je na řadě při kruhovém procházení je nahrazovaná
- Experimenty ukazují, že optimalita algoritmu se blíží skutečnému LRU

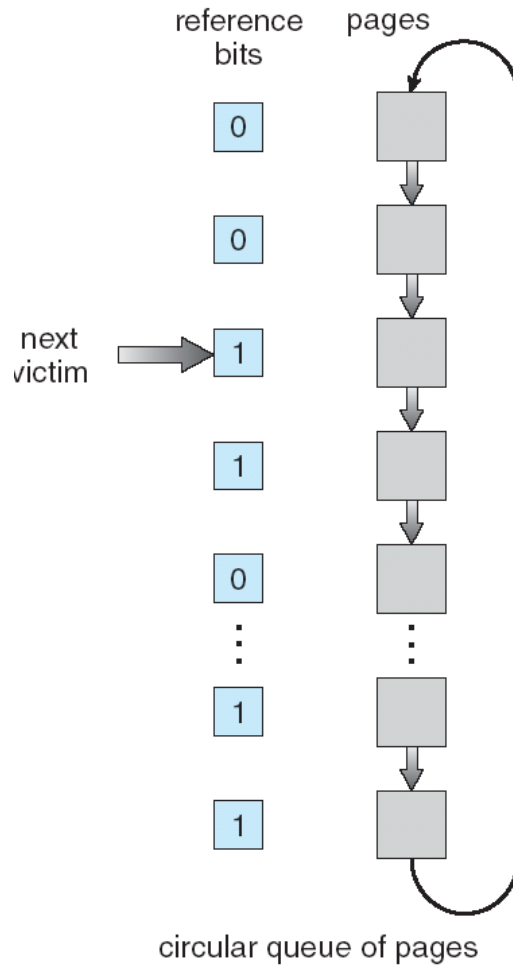


## Druhá šance (2)

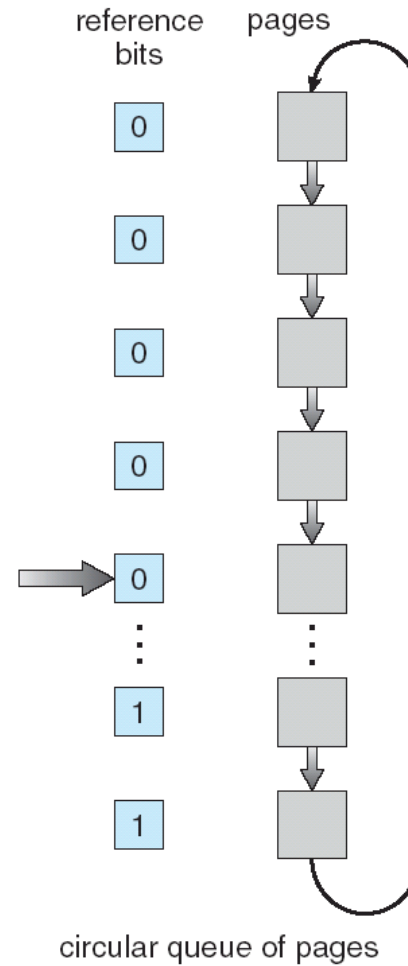
### o Implementace

- Množina rámců kandidujících na nahrazení je organizována jako cyklický buffer
- Když se stránka nahradí, ukazatel se posune na příští rámeček v bufferu
- Pro každý rámeček se nastavuje „use bit“ na 1 když
  - se do rámce zavede stránka poprvé
  - kdykoliv se odpovídající stránka referencuje
- Když se má nalézt oběť, nahradí se stránka v prvním rámečku s „use bit“ nastaveným na 0.
  - Při hledání kandidáta na nahrazení se každý „use bit“ nastavený na 1 nastavuje na 0

# Druhá šance (3)



(a)



(b)

# Srovnání algoritmů

- OPT
  - super, ale nutnost znát budoucnost
- LRU
  - časové čítače u rámců,  $\Delta t + 1$ , nulované odkazem
  - oběť = rámec s nejvyšší hodnotou čítače
  - Implementace má vysokou režii
- FIFO
  - snadná implementace, cyklický seznam
  - špatná heuristika
- Druhá šance
  - upravené FIFO
  - z výběru obětí se vynechává alespoň jednu odkázaná stránka od posledního výběru
  - use\_bit = (počátečně) 0 / po odkazu 1
  - úprava druhá šance
    - use\_bit = 0 / 1, doplněný modified\_bit = 0 / 1
    - pořadí výběru: 00, 01, 1x
    - 0x šetří výpis nemodifikované stránky



# Struktura programu

- `int data[128][128];`
- Jeden řádek odpovídá jedné stránce
- Program 1

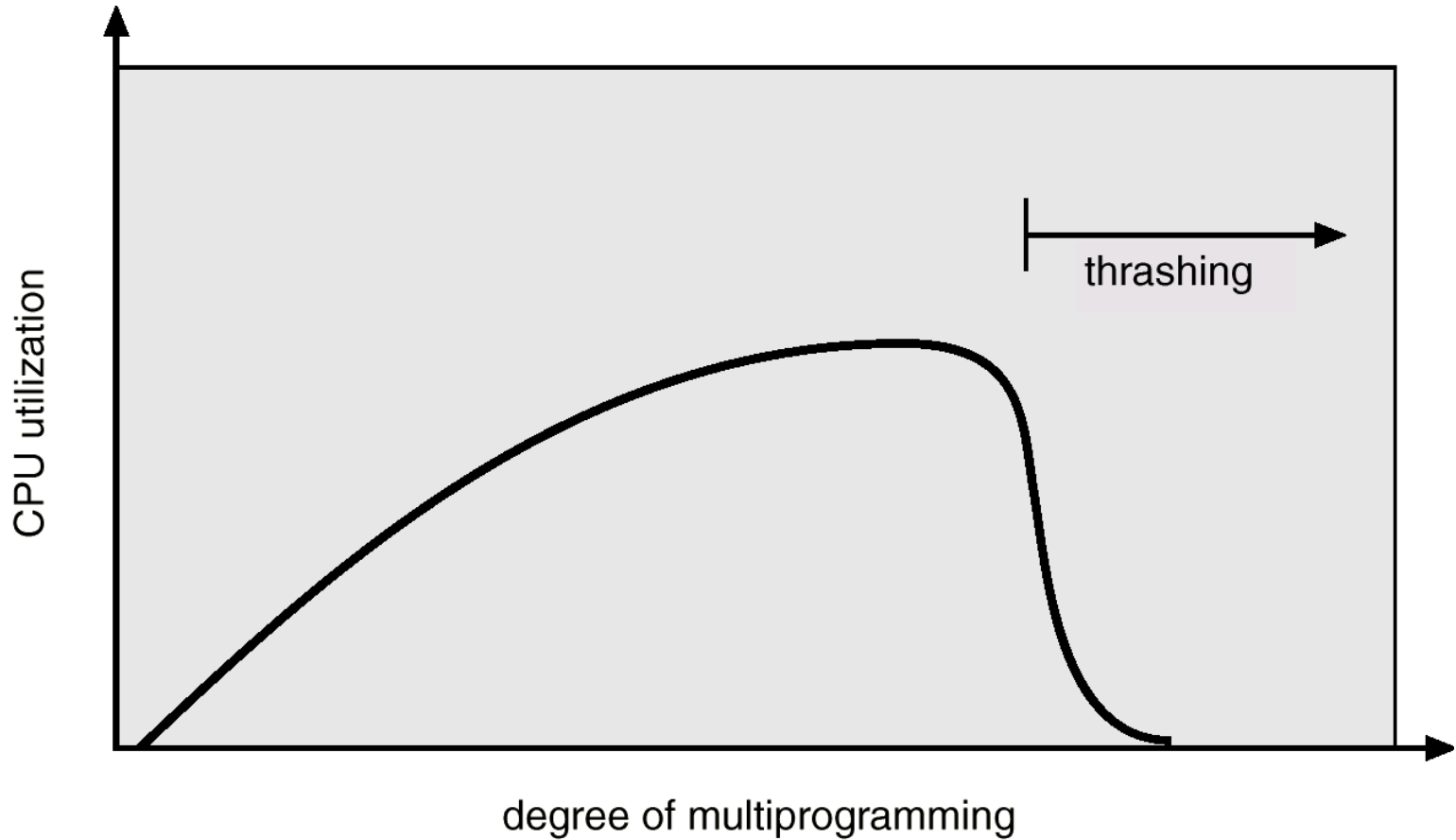
```
for (j = 0; j < 128; j++)  
    for (i = 0; i < 128; i++)  
        data[i,j] = 0;
```

- 128 x 128 = 16,384 výpadků stránky
- Program 2

```
for (i = 0; i < 128; i++)  
    for (j = 0; j < 128; j++)  
        data[i,j] = 0;
```

- 128 výpadků stránky

# Thrashing

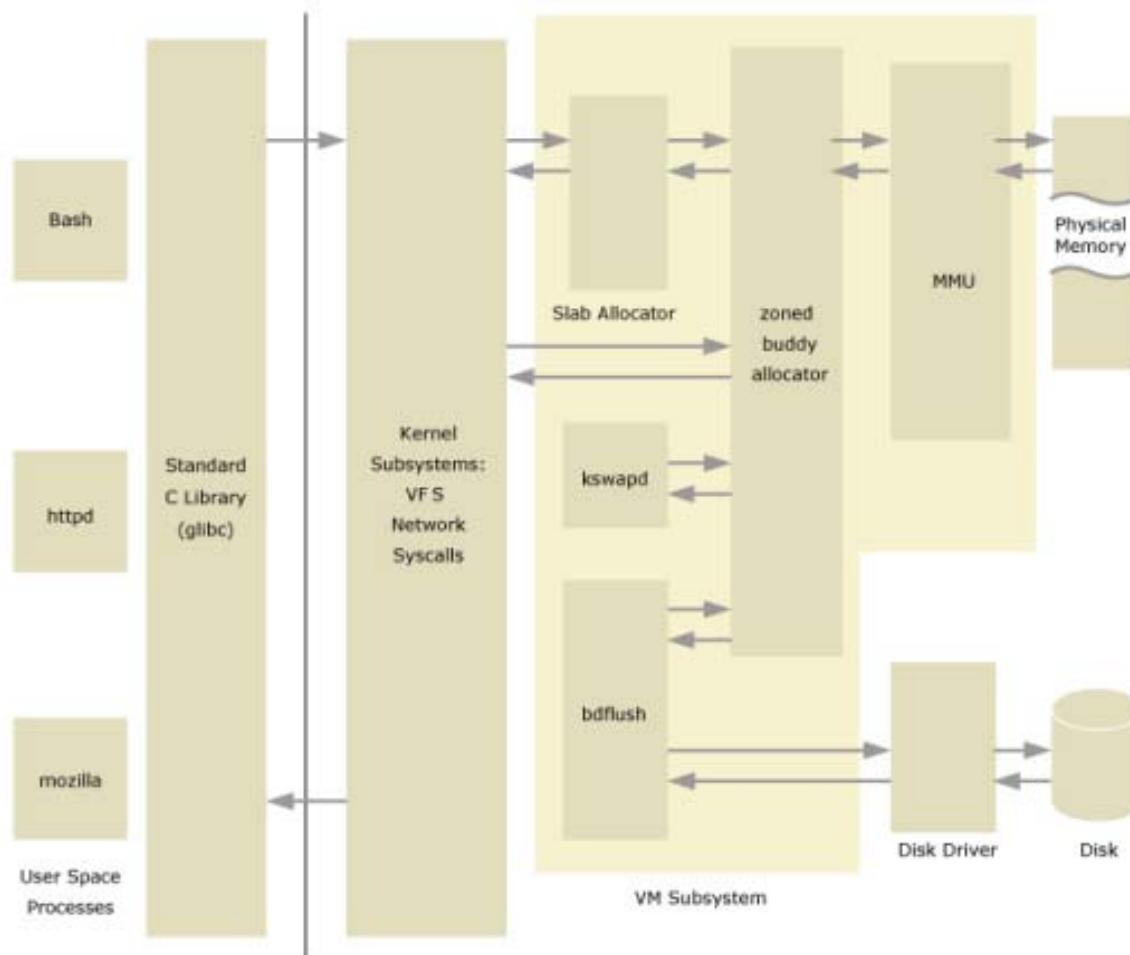


# Buddy alokační algoritmus

	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K
$t = 0$	1024K															
$t = 1$	A-64K	64K	128K		256K			512K								
$t = 2$	A-64K	64K	B-128K		256K			512K								
$t = 3$	A-64K	C-64K	B-128K		256K			512K								
$t = 4$	A-64K	C-64K	B-128K		D-128K		128K	512K								
$t = 5$	A-64K	64K	B-128K		D-128K		128K	512K								
$t = 6$	128K		B-128K		D-128K		128K	512K								
$t = 7$	256K				D-128K		128K	512K								
$t = 8$	1024K															

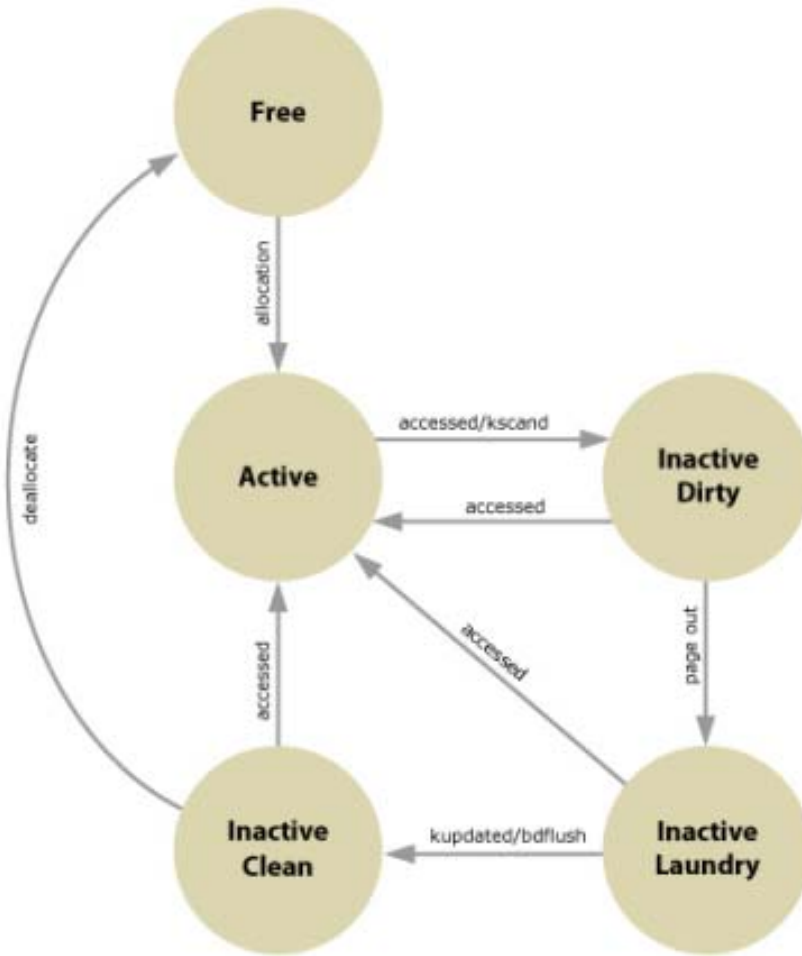
1. A alokuje 64K
2. B alokuje 128K
3. C alokuje 64K
4. D alokuje 128K
5. C uvolňuje paměť
6. A uvolňuje paměť
7. B uvolňuje paměť
8. D uvolňuje paměť

# Příklad: Linux 2.4



Zdroj: RedHat

# Příklad: Linux 2.4



Zdroj: RedHat

# ● ● ● | Příklad: Linux 2.4

- Kscand
- Kswapd
  - Prochází paměť a hledá stránky k uvolnění pokud dochází paměť
- Kupdated [dirty pages; pravidelně] + Bdflush (2.4) [dirty buffers; v případě potřeby]
- Pdflush (2.6) [dirty pages]
  - Zapisuje na disk



# Documentation/sysctl/vm.txt

- Currently (2.6.32), these files are in /proc/sys/vm:
  - `block_dump`
  - `dirty_background_bytes`
  - `dirty_background_ratio`
  - `dirty_bytes`
  - `dirty_expire_centisecs`
  - `dirty_ratio`
  - `dirty_writeback_centisecs`
  - `drop_caches`
  - `hugepages_treat_as_movable`
  - `hugetlb_shm_group`
  - `laptop_mode`
  - `legacy_va_layout`
  - `lowmem_reserve_ratio`
  - `max_map_count`
  - `memory_failure_early_kill`
  - `memory_failure_recovery`
  - `min_free_kbytes`
  - `min_slab_ratio`
  - `min_unmapped_ratio`
  - `mmap_min_addr`
  - `nr_hugepages`
  - `nr_overcommit_hugepages`
  - `nr_pdflush_threads`
  - `nr_trim_pages`
  - `numa_zonelist_order`
  - `oom_dump_tasks`
  - `oom_kill_allocating_task`
  - `overcommit_memory`
  - `overcommit_ratio`
  - `page-cluster`
  - `panic_on_oom`
  - `percpu_pagelist_fraction`
  - `stat_interval`
  - `swappiness`
  - `vfs_cache_pressure`
  - `zone_reclaim_mode`