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

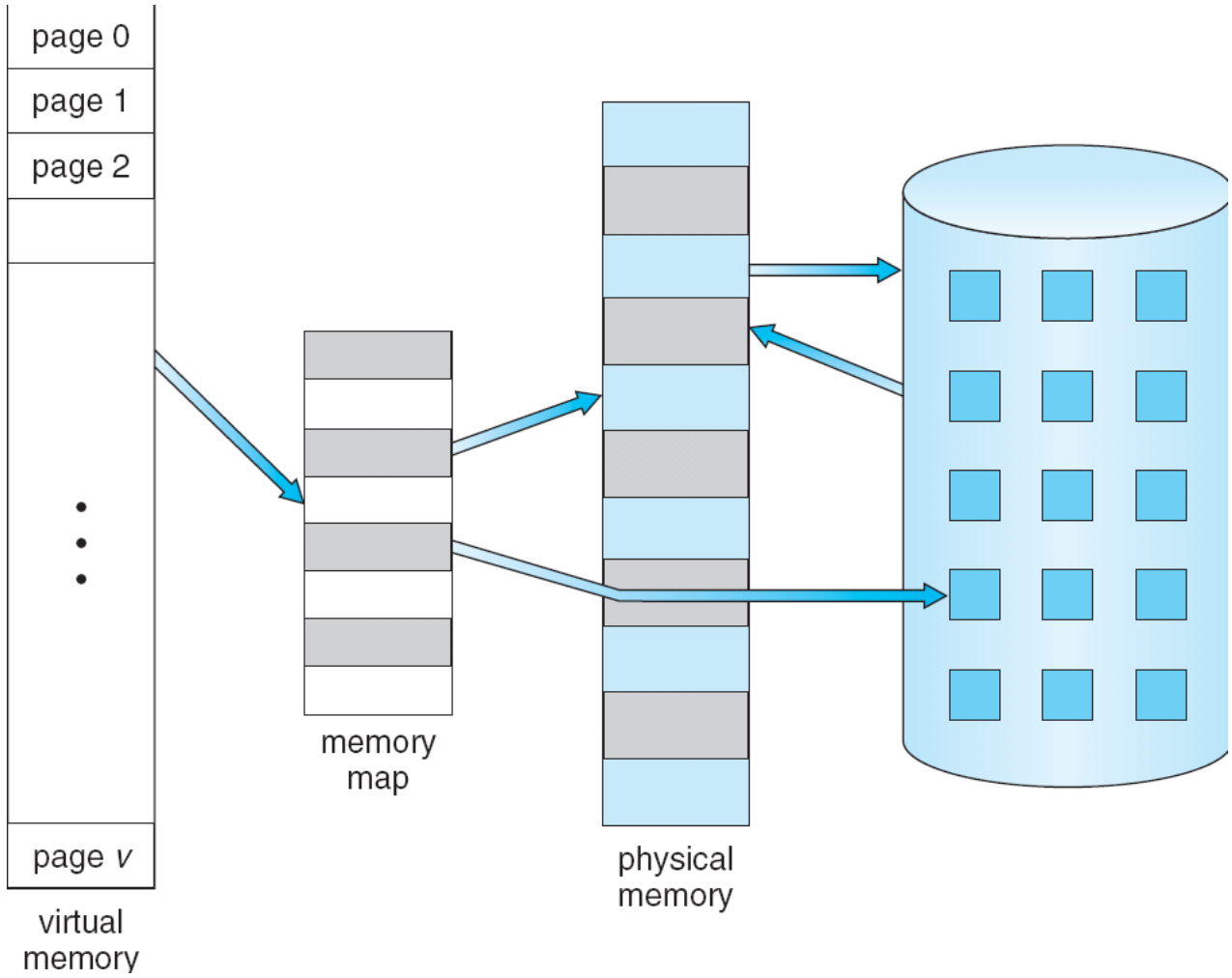
Virtuální paměť – vlastnosti

- 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)

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 pamětí

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 („sousedi“)
  - 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?

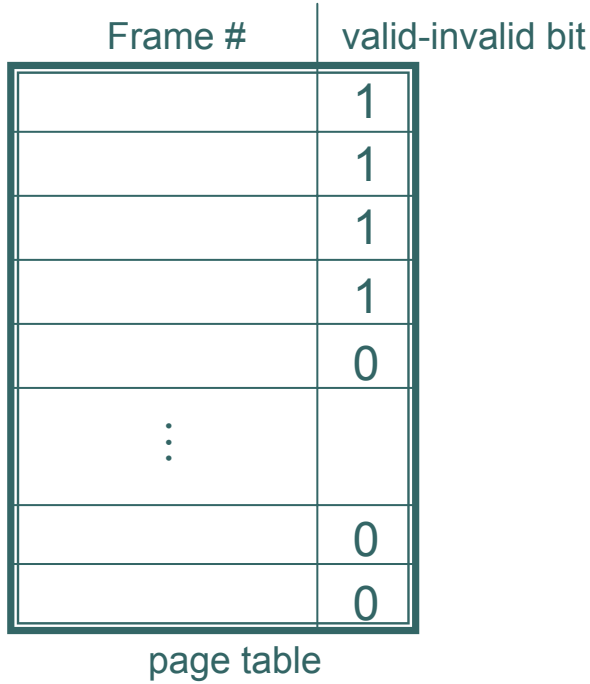
- Uplatňuje se v okamžiku, kdy je FAP plný
- Typicky v okamžiku zvýšení stupně multitaskingu
- Kterou stránku „obětovat“ a vyhodit z FAP (politika výběru oběti)?
- Politika nahrazování
  - určení oběti
    - politika přidělování místa
      - kolik rámců přidělit procesu?
      - intra/extra (local/global) 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

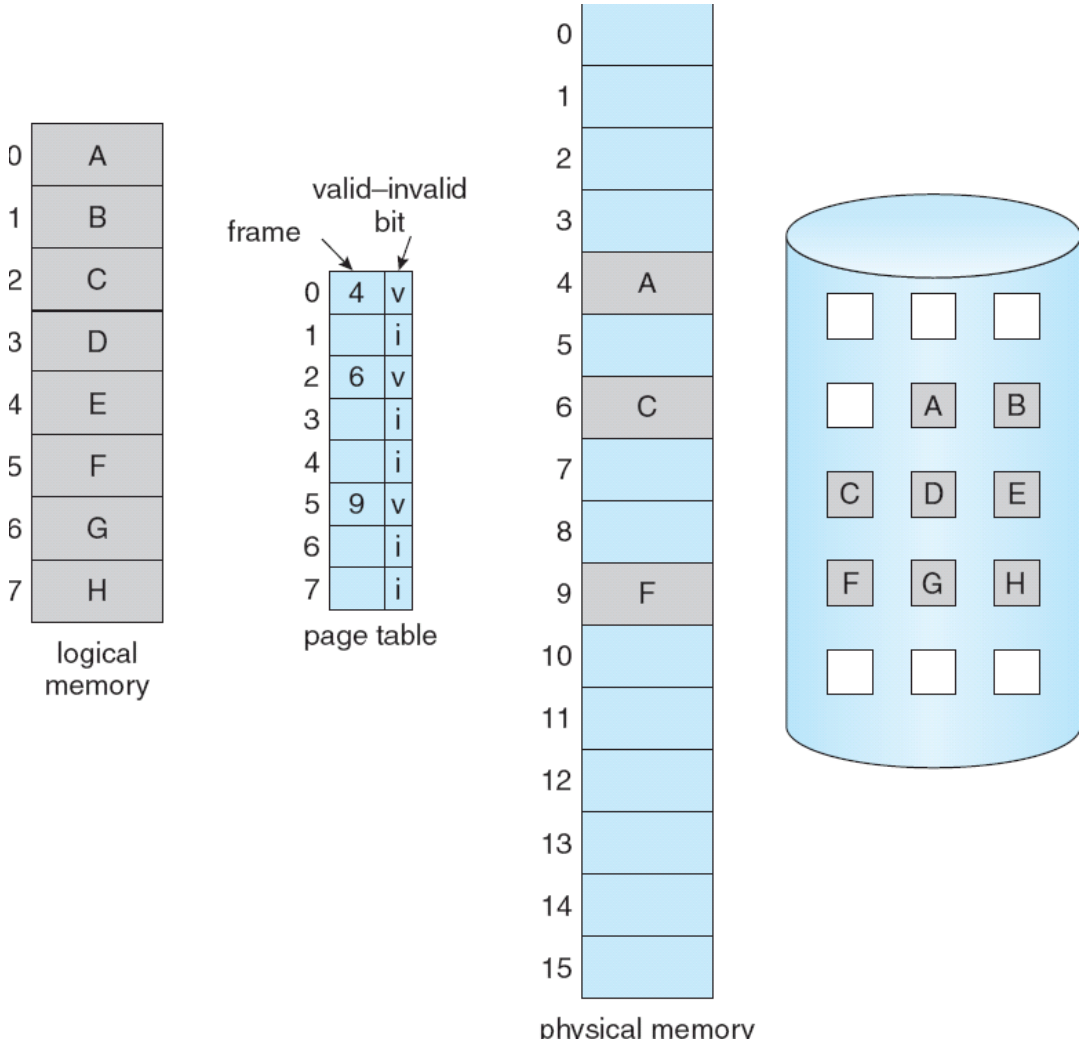
- Stránka se zavádí do FAP jen když je potřebná
- Přínosy
  - méně I/O operací
  - menší požadavky na paměť
  - rychlejší reakce
  - může pracovat více uživatelů
- Kdy je stránka potřebná?
  - byla odkazovaná
  - legální reference
    - Nachází se ve FAP, přeloží se logická adresa na fyzickou
  - nelegální 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“



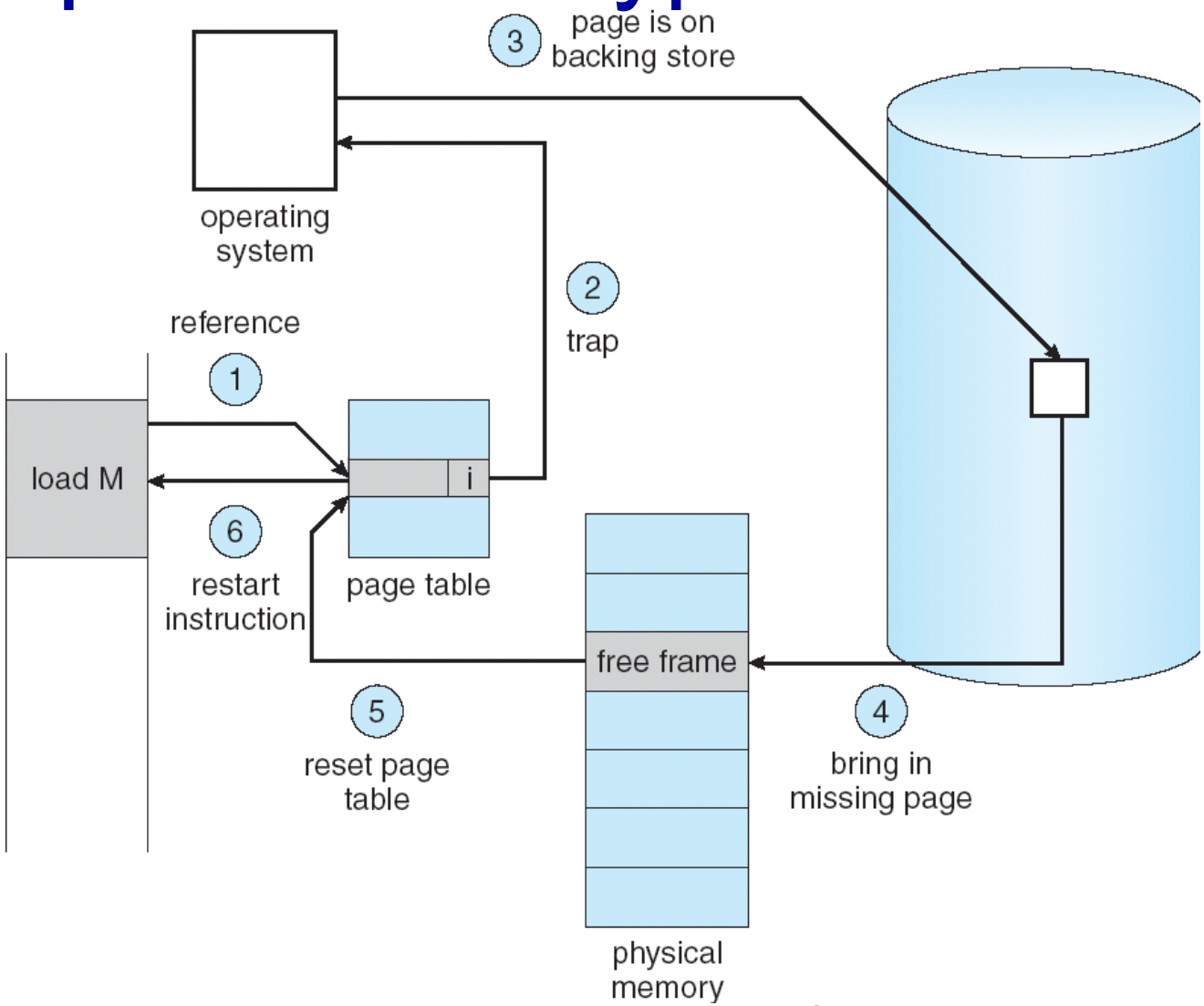
Příklad: tabulka stránek



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



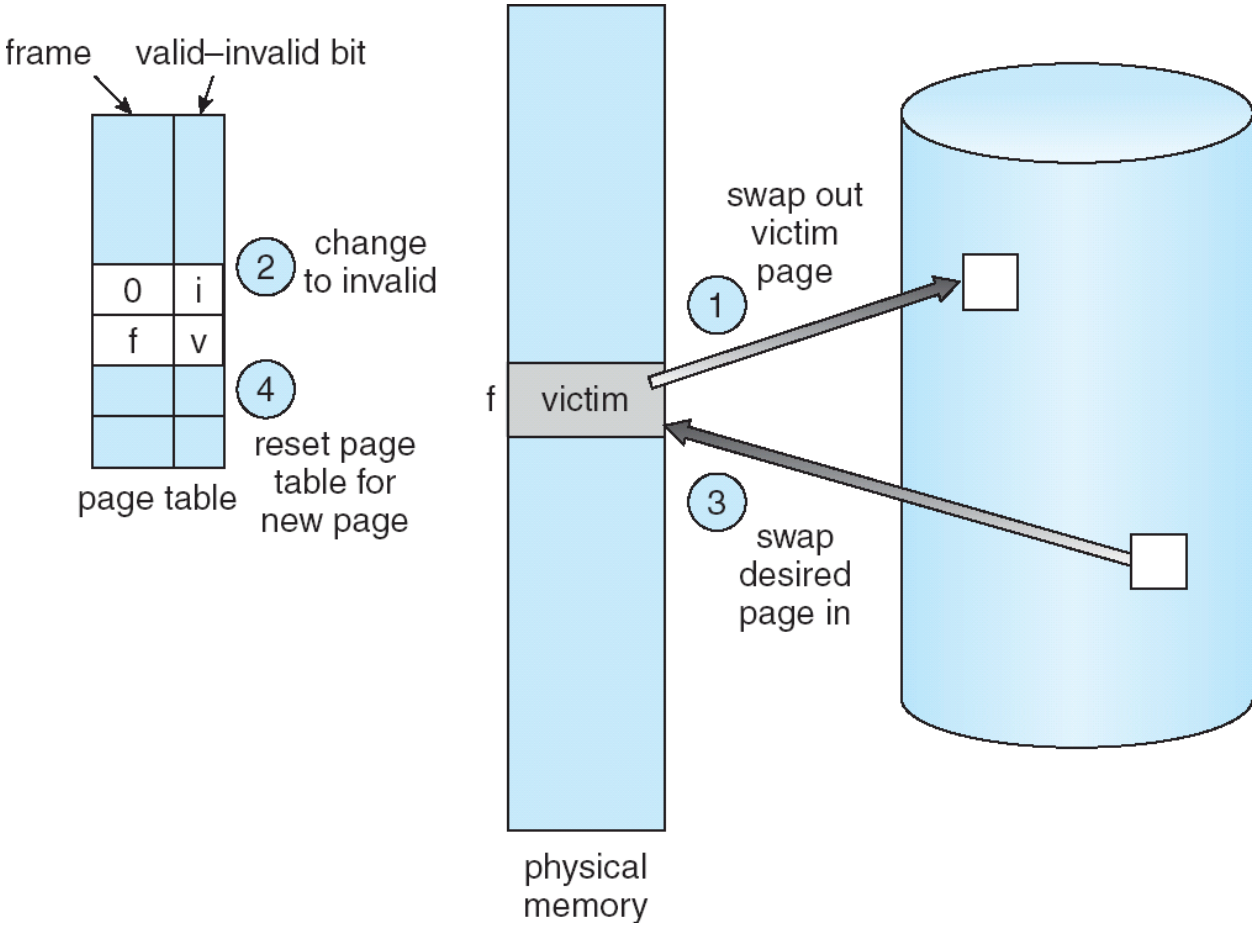
Volný rámec

- pokud není žádný rámec aktuálně označen jako volný, musíme nějaký uvolnit
  - nalezneme se „oběť“ – 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

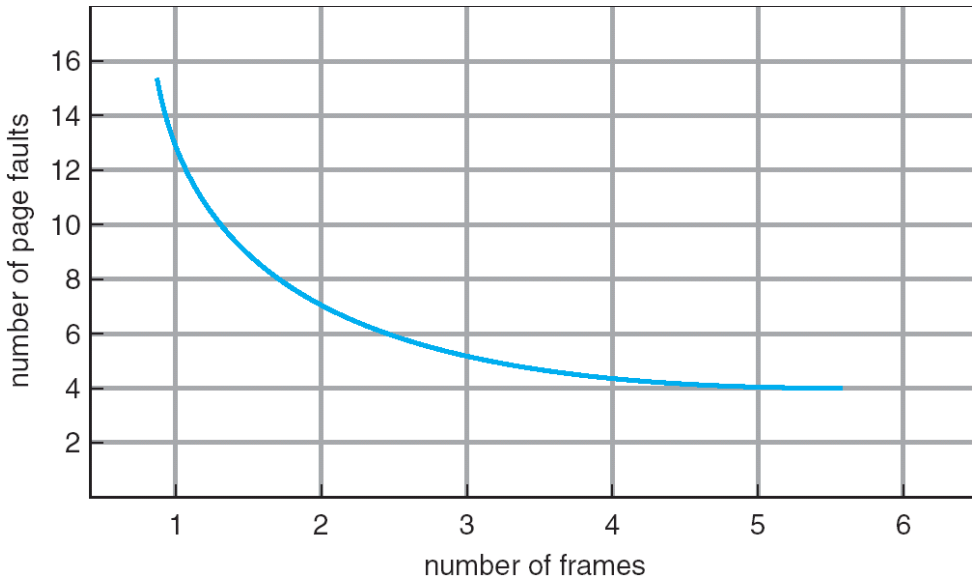
- Odkazy na instrukce programu a na dat mají tendenci tvořit shluky
- Časová lokalita a prostorová lokalita
  - provádění programu je s výjimkou skoků a volání podprogramů sekvenční
  - programy mají tendenci zůstávat po jistou dobu v rámci nejvýše několika procedur (nelze jen něco volat a hned se vracet)
  - 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

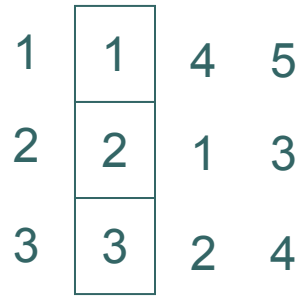


Algoritmy určení oběti

- 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

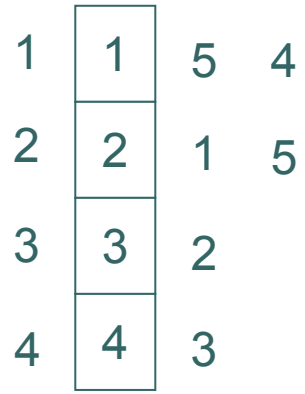
Algoritmus First-In-First-Out

- 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)



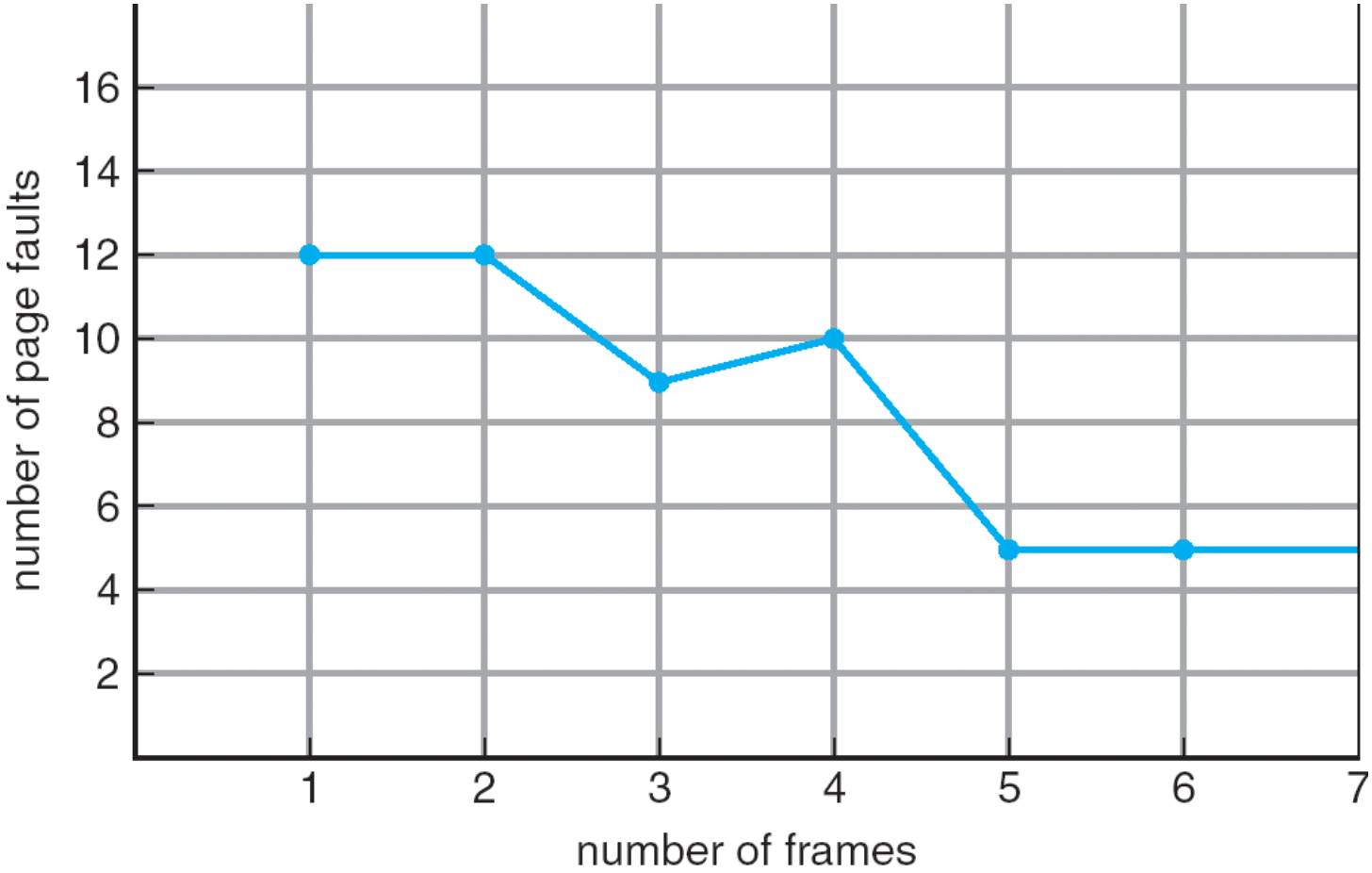
9 výpadků

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



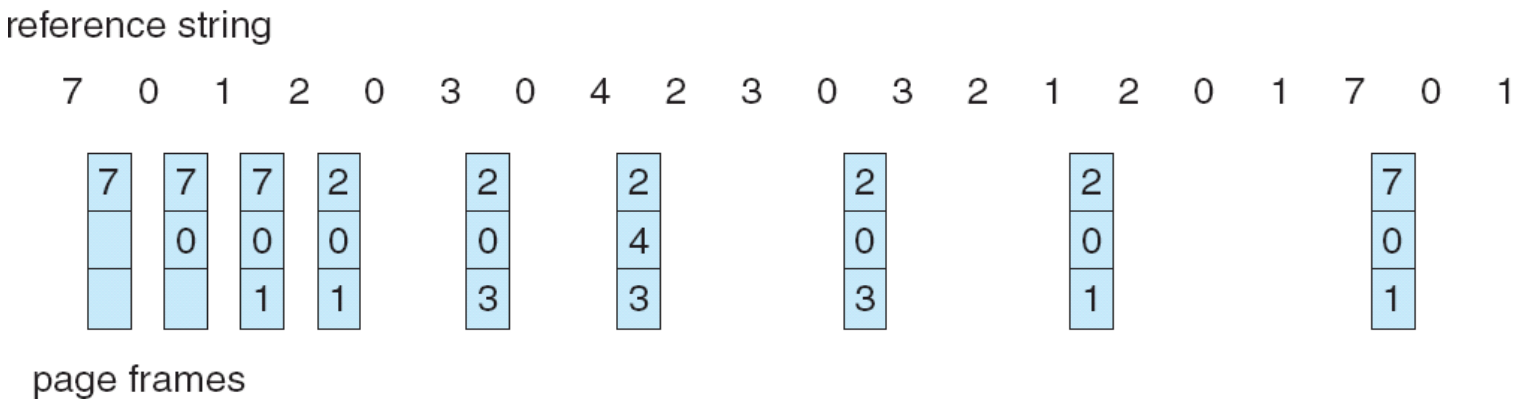
10 výpadků

Beladyho anomálie



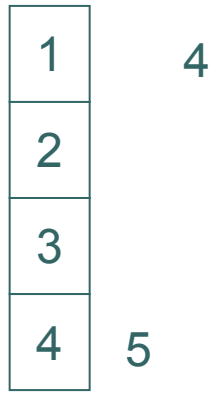
Algoritmus First-In-First-Out

- 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



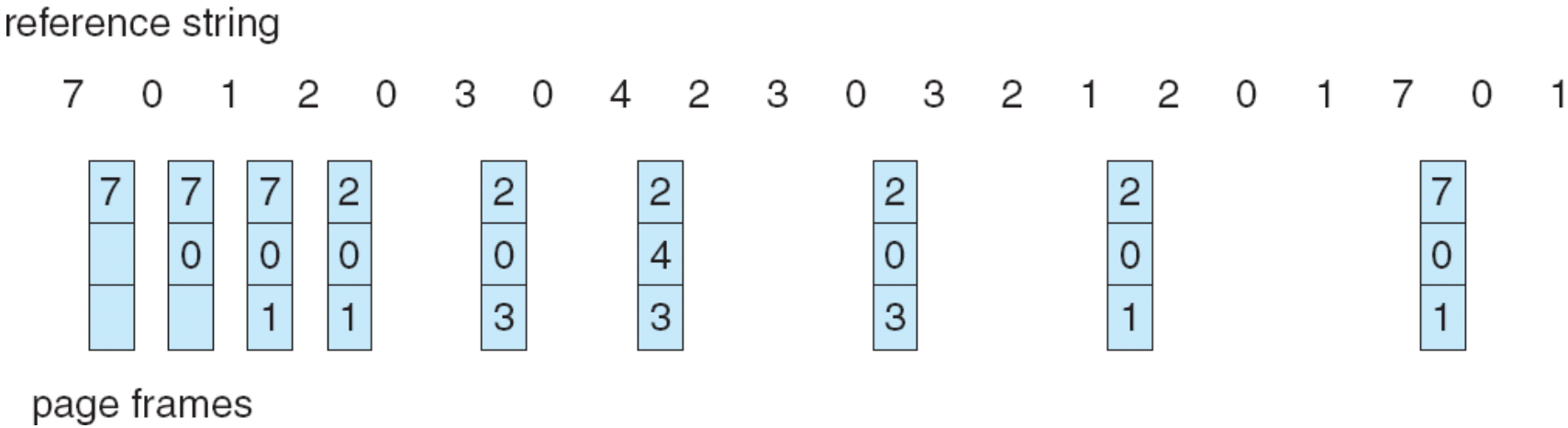
Optimální algoritmus

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



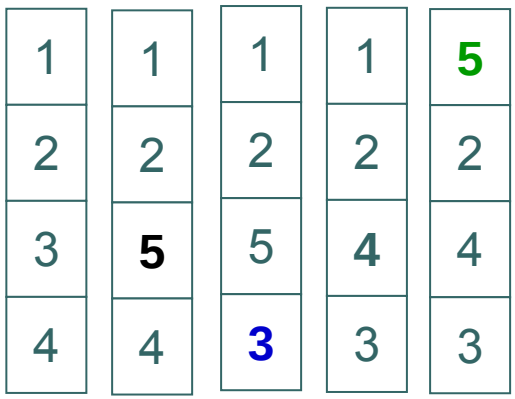
Optimální algoritmus

- Další příklad:



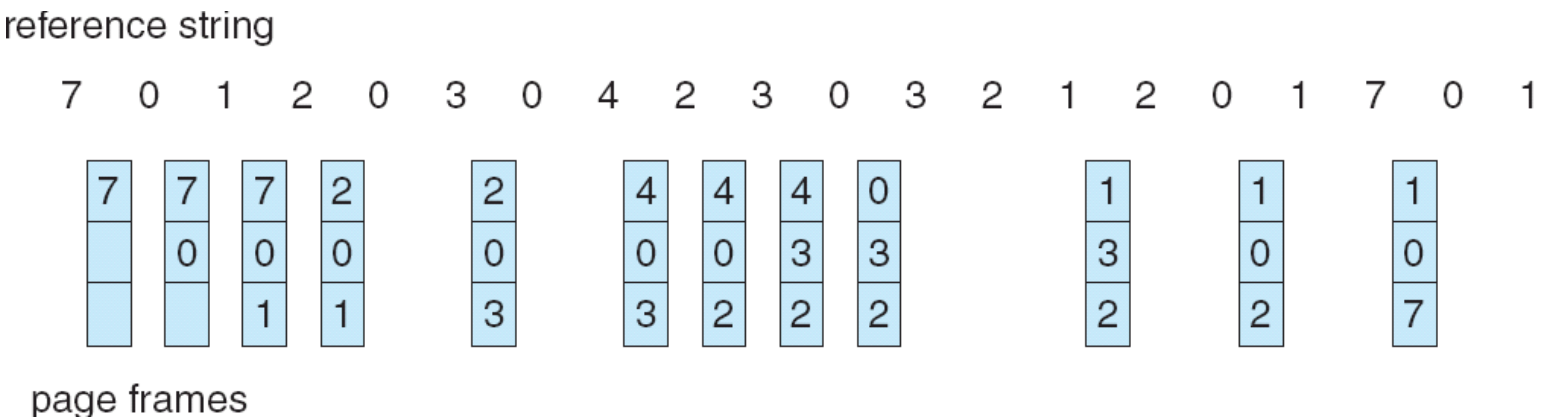
Algoritmus LRU

- 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 3 rámce
  - 1, 2, 3, 4, 1, 2, **5**, 1, 2, **3**, **4**, **5**



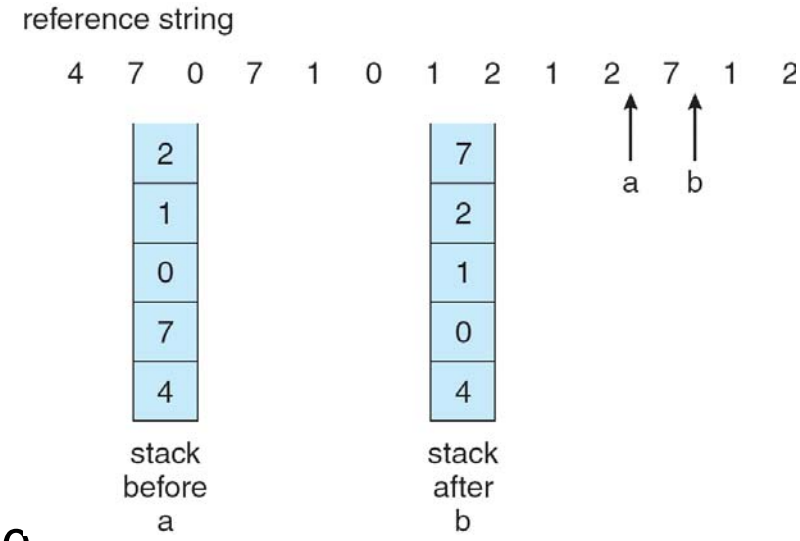
Algoritmus LRU

- Další příklad:



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 „oběť“)
- 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



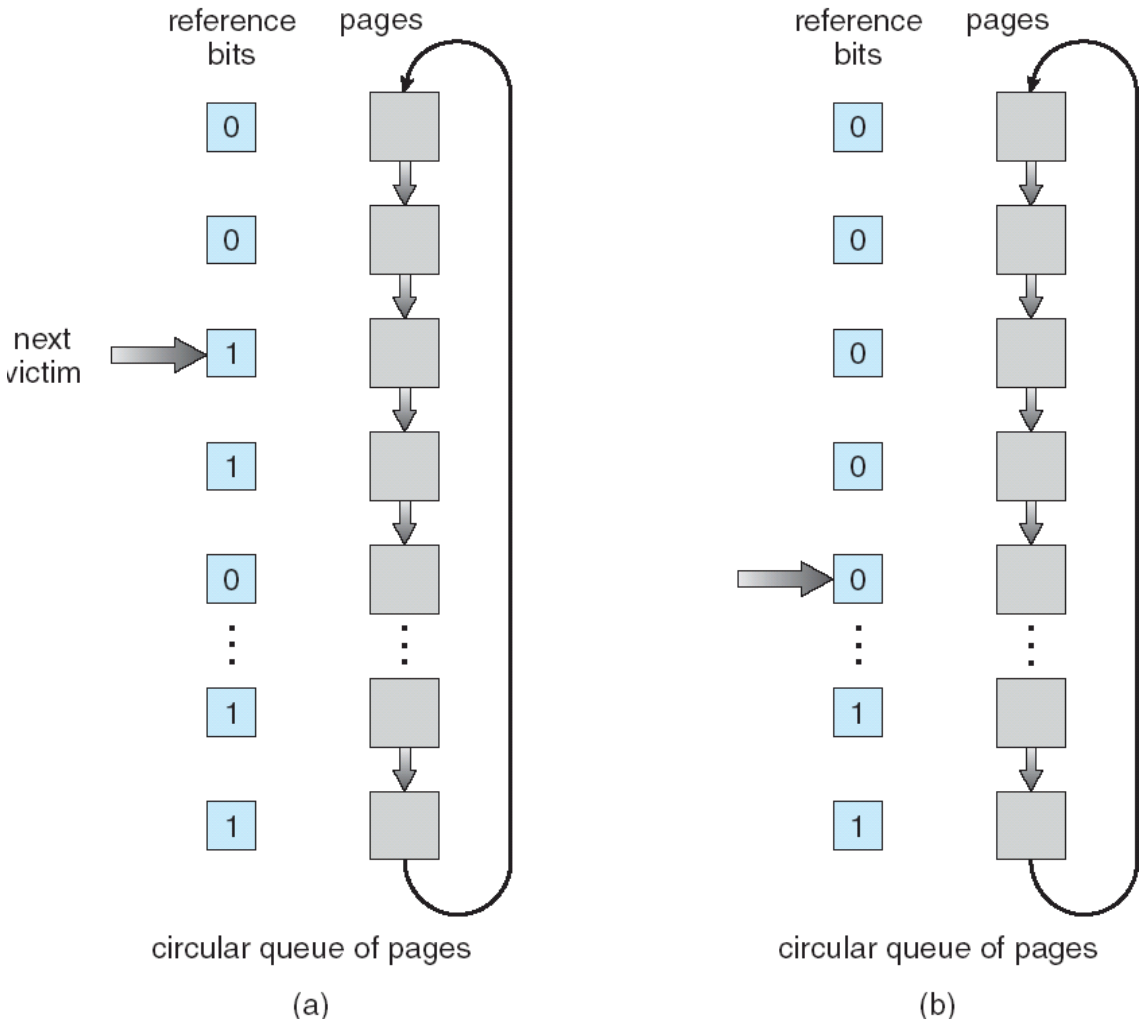
Druhá šance

- 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)

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

Druhá šance (3)



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 jednou odkázaná stránka od posledního výběru
  - use_bit = (počátečně) 0 / po odkazu 1
  - úprava druhá šance
    - use_bit = 0 / 1, doplněný modificated_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