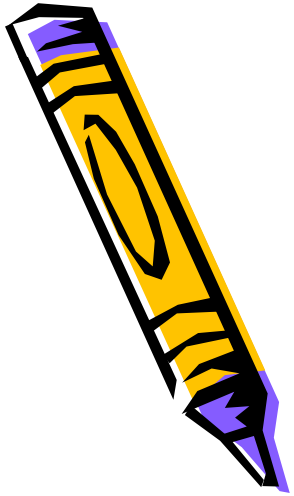
Memori Virtual
(*Virtual Memory*)

Heri Kurniawan

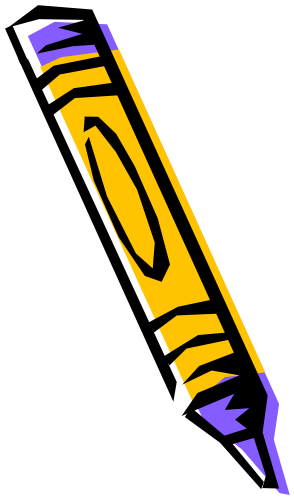
OS-Gasal 2009/2010

Tujuan Pembelajaran



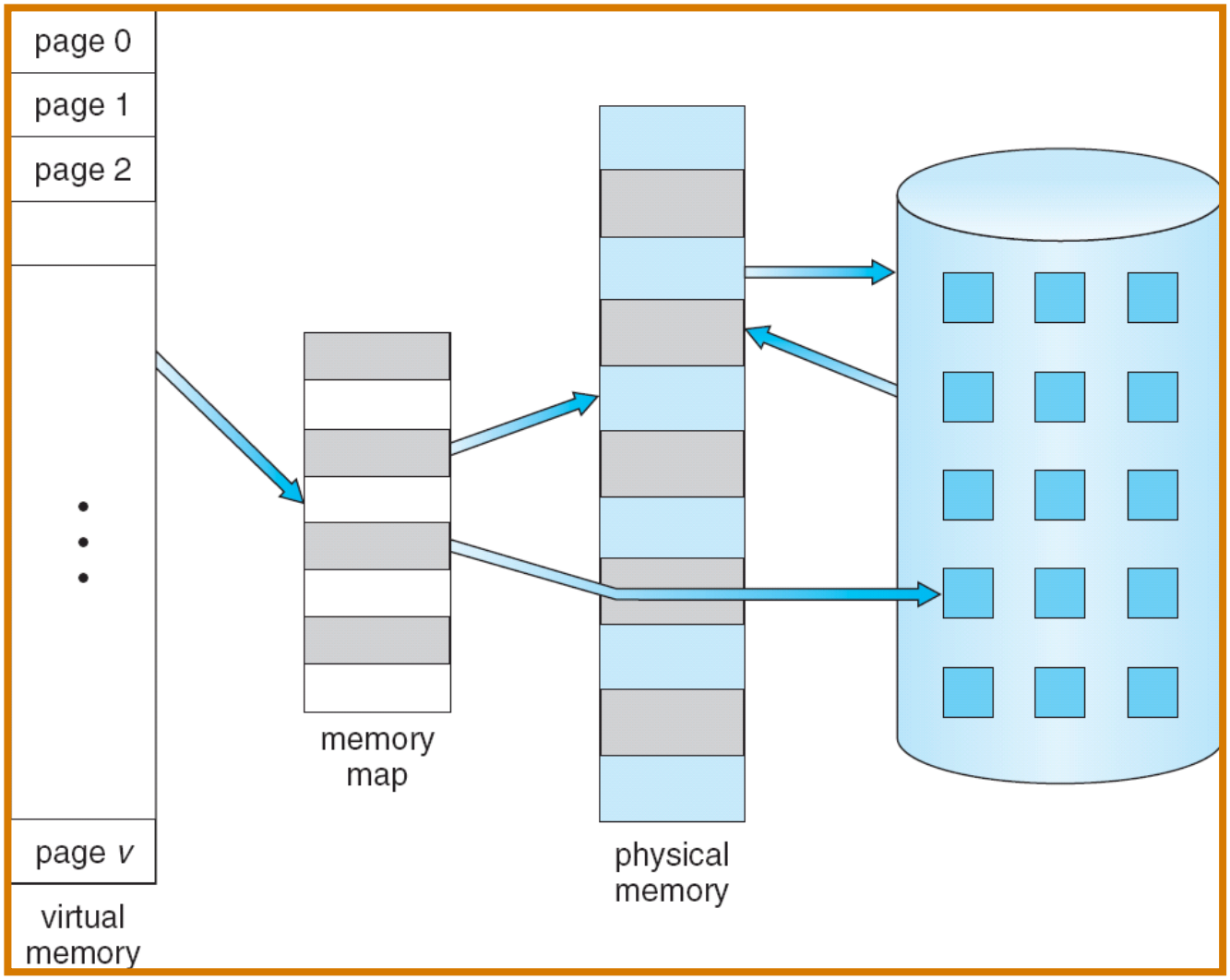
- Memahami manfaat virtual memori
- Memahami bagaimana demand paging bekerja
- Memahami penggunaan *copy-on-write*
- Memahami dasar-dasar pemberian halaman (*page replacement*)

Memori Virtual



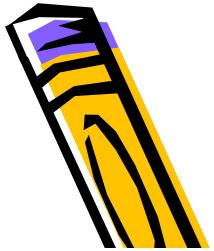
- Program membutuhkan kapasitas yang lebih besar dari kapasitas memori fisik ⇒ gunakan *Virtual Memory* !
- **Virtual memory**
  - Ruang *logical address* dapat lebih besar dari pada ruang *physical address*
  - Proses yang dieksekusi tidak harus seluruhnya berada dimemori.
  - Fakta program di memori:
    - Kode untuk menangani error jarang dieksekusi
    - Alokasi memori untuk variable array, list dan table tidak proporsional
    - Feature dan routine jarang digunakan

Memori Virtual



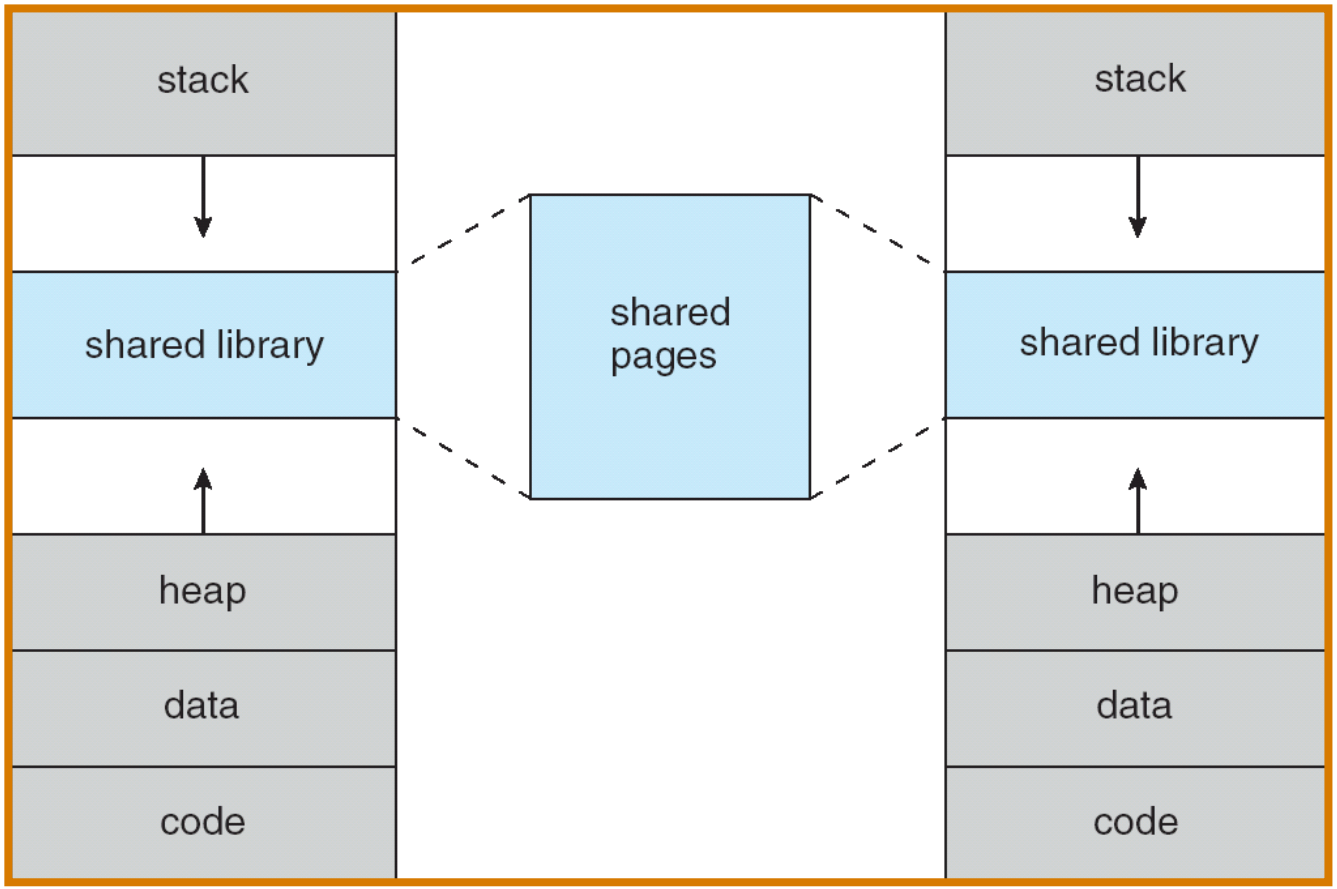
Memori virtual lebih besar dibanding memori fisik

Memori Virtual



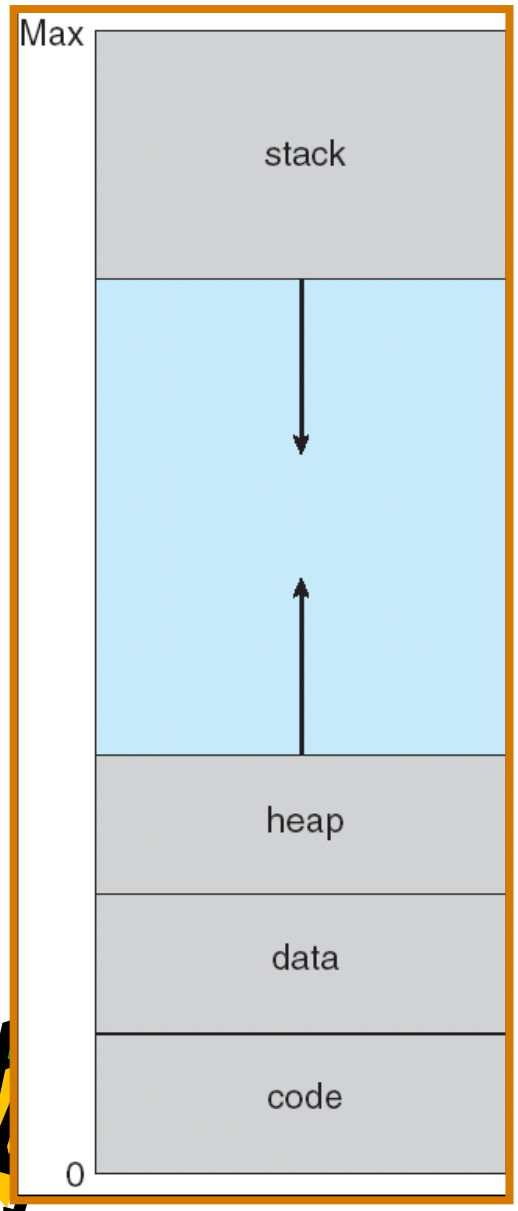
- Eksekusi program secara parsial dimemori:
  - Memudahkan programmer
    - Tidak ada batasan memori
  - Menambah utilisasi dan throughput CPU
  - Aktifitas I/O untuk swapping masing-masing program user berkurang
- Melalui memori virtual, page dapat di*share* oleh beberapa proses
  - System library dapat di share ke beberapa proses
  - Proses dapat membuat *shared* memory untuk komunikasi
  - Page child & parent dapat di*share* setelah fork() dieksekusi
- **Virtual address space**
  - Proses pada ruang alamat logika dapat berurutan
  - Pada memori fisik *page* nya tersebar (melalui MMU)

Memori Virtual



Shared library dengan menggunakan memori virtual

Virtual address space

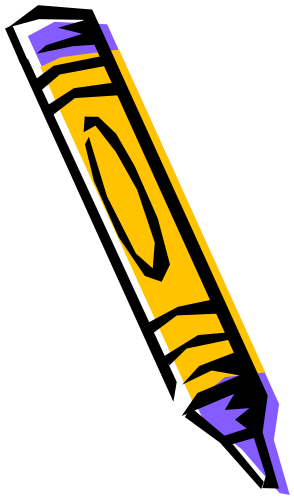


Hole / large bank space dapat digunakan jika ruang logika *stack* dan *heap* bertambah -> *dynamic memory allocation*

Virtual address space yang mempunyai hole disebut *sparse address space*

hole /
large bank space

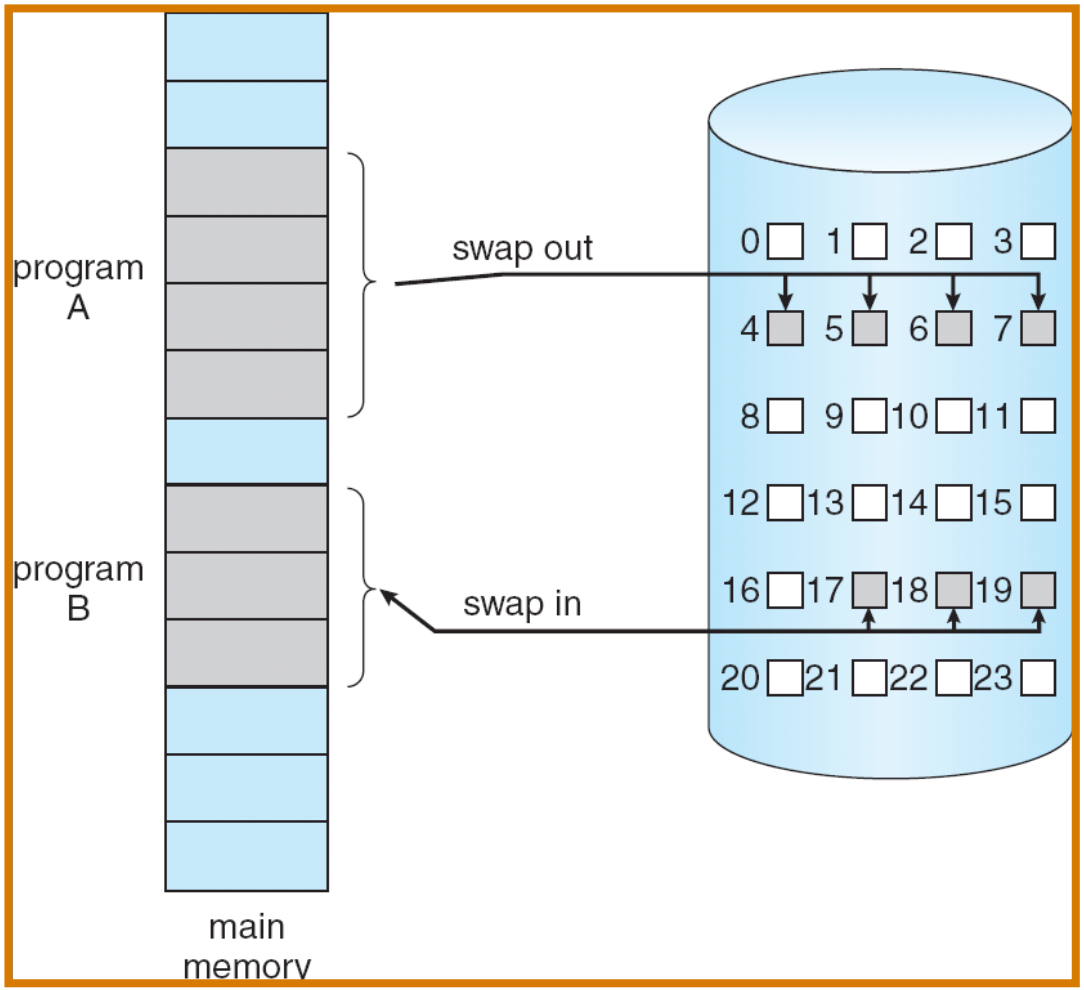
Demand Paging



- Page diload ke memory hanya ketika dibutuhkan
  - Aktifitas I/O lebih sedikit
  - Ruang memori yang dibutuhkan lebih sedikit
  - Respon menjadi lebih cepat
  - Lebih banyak proses user dalam memori

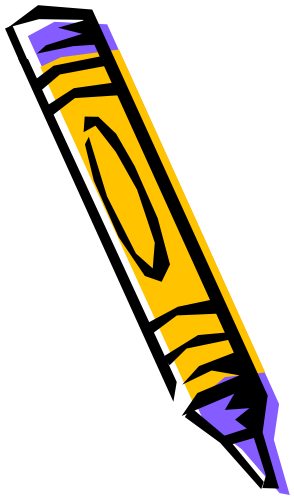
- **Lazy swapper** – tidak melakukan swapping page ke memori hingga page dibutuhkan.
  - Swapper yang berurusan dengan page disebut pager

Demand Paging



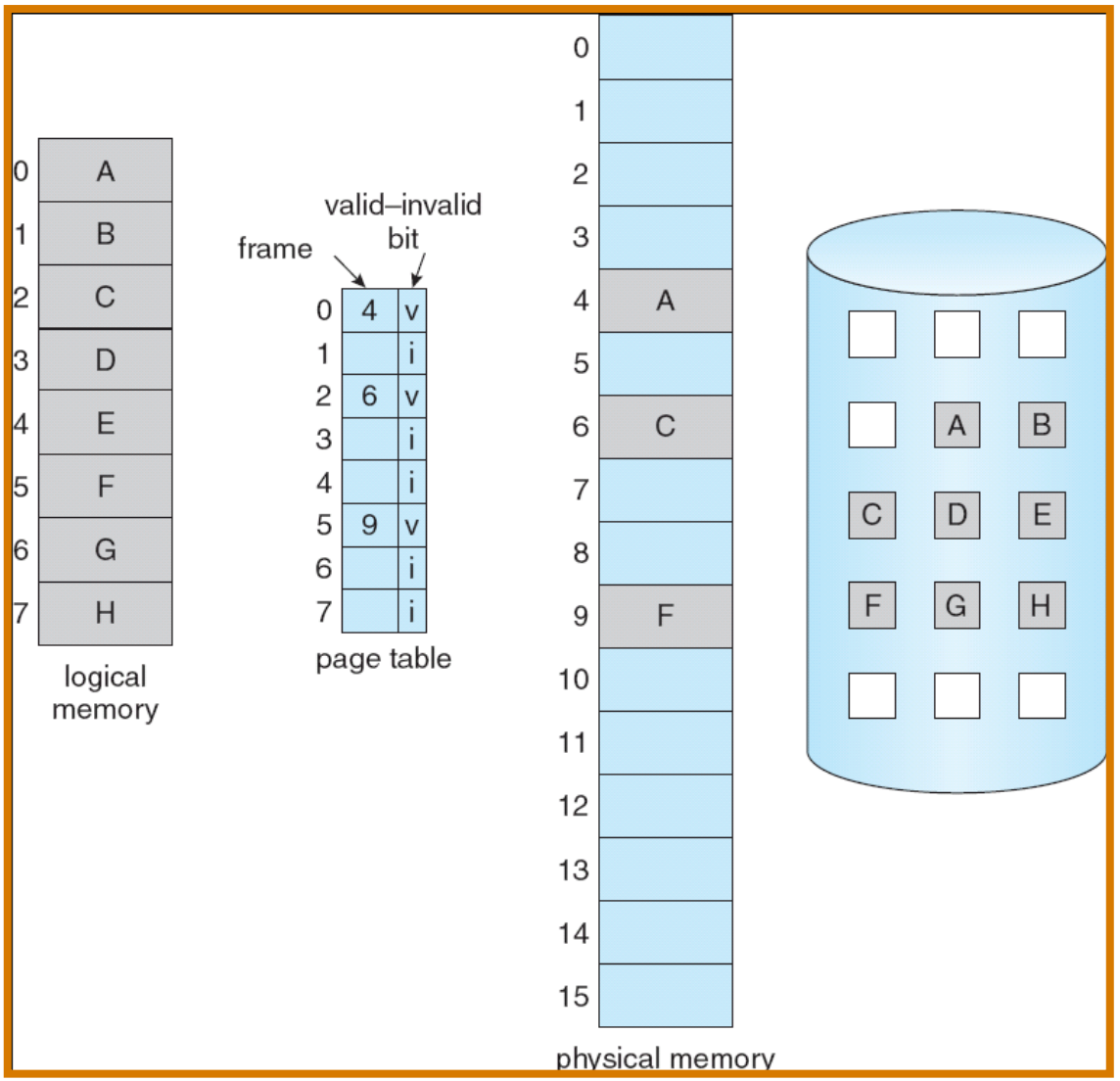
Proses swapping program ke dalam memori

Valid-Invalid Bit



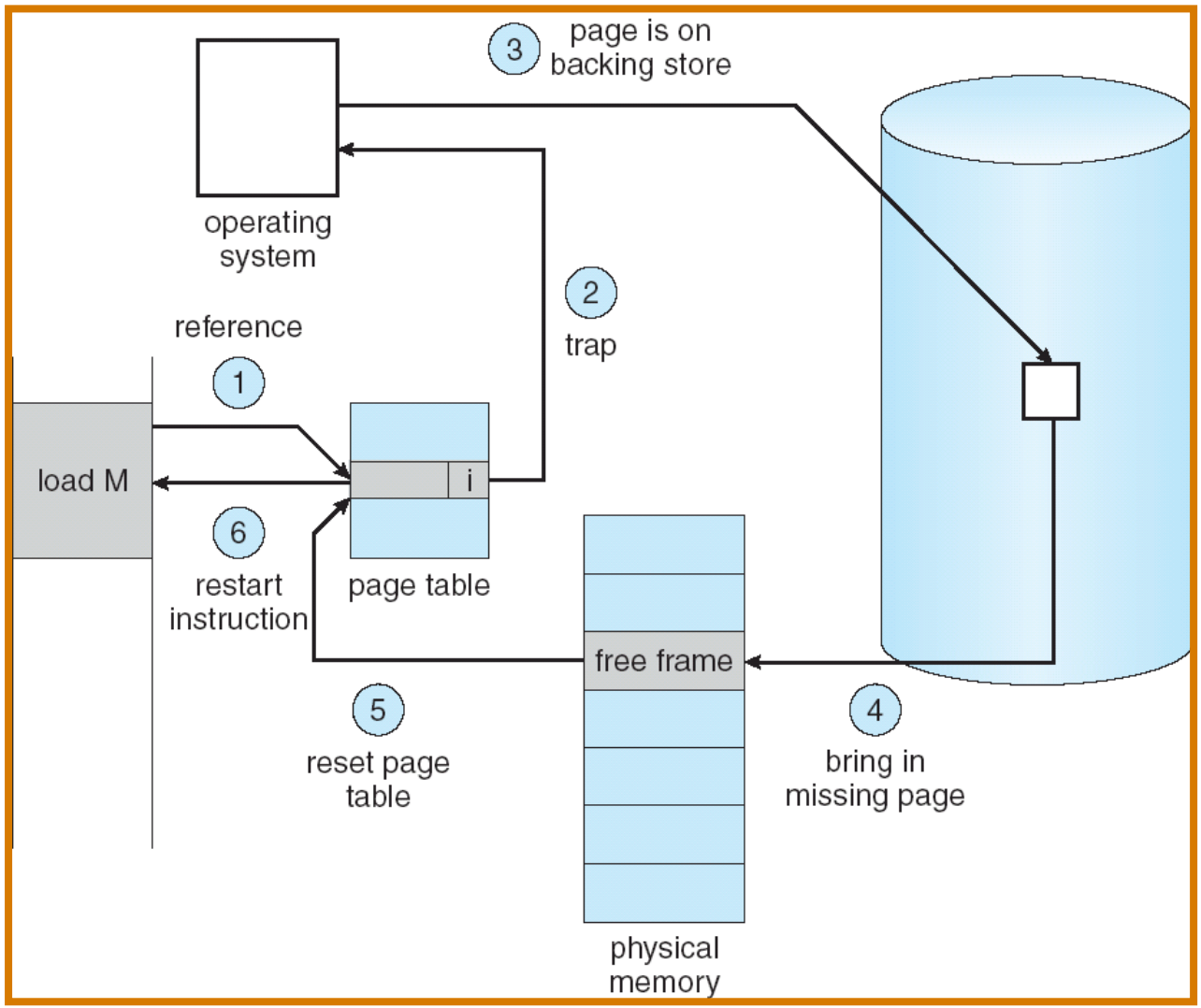
- Cek *page* ada di memori atau tidak? ⇒ gunakan Valid-Invalid bit
  - Setiap entri page table diasosiasikan dengan valid dan invalid bit.
- **v** (valid) ⇒ ada di memori
- **i** (invalid)⇒ punya arti dua :
  1. alamat logika tidak sesuai dengan proses
  2. page ada di disk tapi tidak ada dimemori
- Jika nilai bit **I** tidak pernah diakses, tidak ada pengaruh
- Jika nilai bit **I** diakses, terjadi *page fault-trap*, sistem operasi mengambil page dari disk ke memory

Valid-Invalid Bit



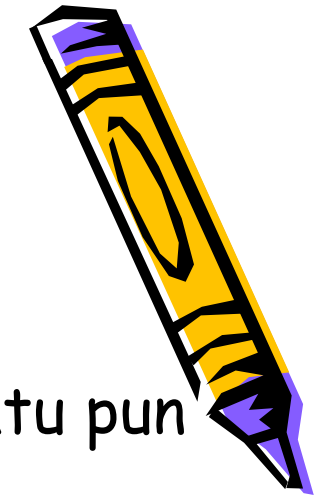
Beberapa page berada dalam disk

Page Fault



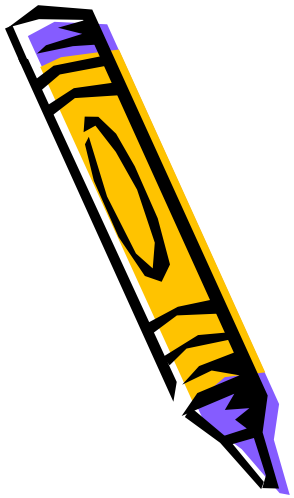
Alur penanganan page fault

Demand Paging



- Pure demand paging ⇒ eksekusi proses tanpa ada satu pun page dimemori
  - page di pindah kememori hanya saat dibutuhkan
- Hardware untuk demand paging:
  - Page Table
  - Secondary Memory (*high-speed disk*)
- Hal penting dalam demand paging
  - Restart instruksi setelah *page fault* terjadi
  - Restart pada lokasi dan state yang sama

Kinerja Demand Paging



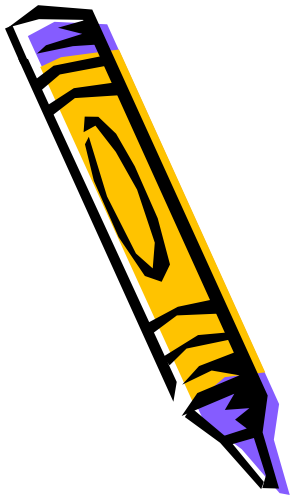
- Kemungkinan terjadinya Page Fault dinotasikan dengan p, dimana p berada dalam range $0 \leq p \leq 1.0$
  - jika $p = 0$, tidak terjadi page fault
  - Jika $p = 1$, page fault pada semua kejadian akses

- Effective Access Time (EAT)

  EAT =((1 – *p*) x memory access) + (*p* x *page fault time*)

- Jika P=0 ⇒ Effective Access Time = Memori Access Time

EAT Demand Paging



- Waktu akses memory = 200 nanosecond
- Rata-rata waktu *page-fault service time* = 8 milliseconds
- 1 ms=10^6 ns
- EAT = ((1 – p) x 200) + (p x (8 milliseconds))

  = ((1 – p) x 200) + (p x 8,000,000)

  = 200 + (p x 7,999,800)
- Jika 1 dari 1.000 kali akses terjadi fault, maka EAT = 8.2 microseconds.
- EAT bertambah menjadi 40 kali lipat dari waktu akses memory !

EAT Demand Paging

- Jika ingin EAT tidak lebih dari 220ns (waktu akses memori bertambah 10 %) maka :

220 > 200+7.999.800 x *p*

20 > 7.999.800 x *p*

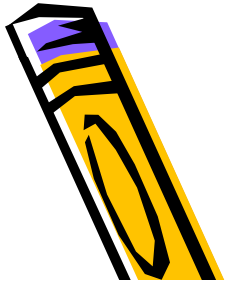
p < 0,0000025, artinya *p* harus lebih kecil dari kejadian *page-fault* sekali dalam 400.000 kali akses

- Tiga komponen waktu utama saat terjadi page fault:
  - *service page fault interrupt* (1-100 microseconds)
  - baca page/*page switch time* (8 millisecond)
    - Rata-rata latency:3 ms, seek:5ms, transfer: 0.05ms
  - restart proses (1-100 microseconds)

Copy-on-Write

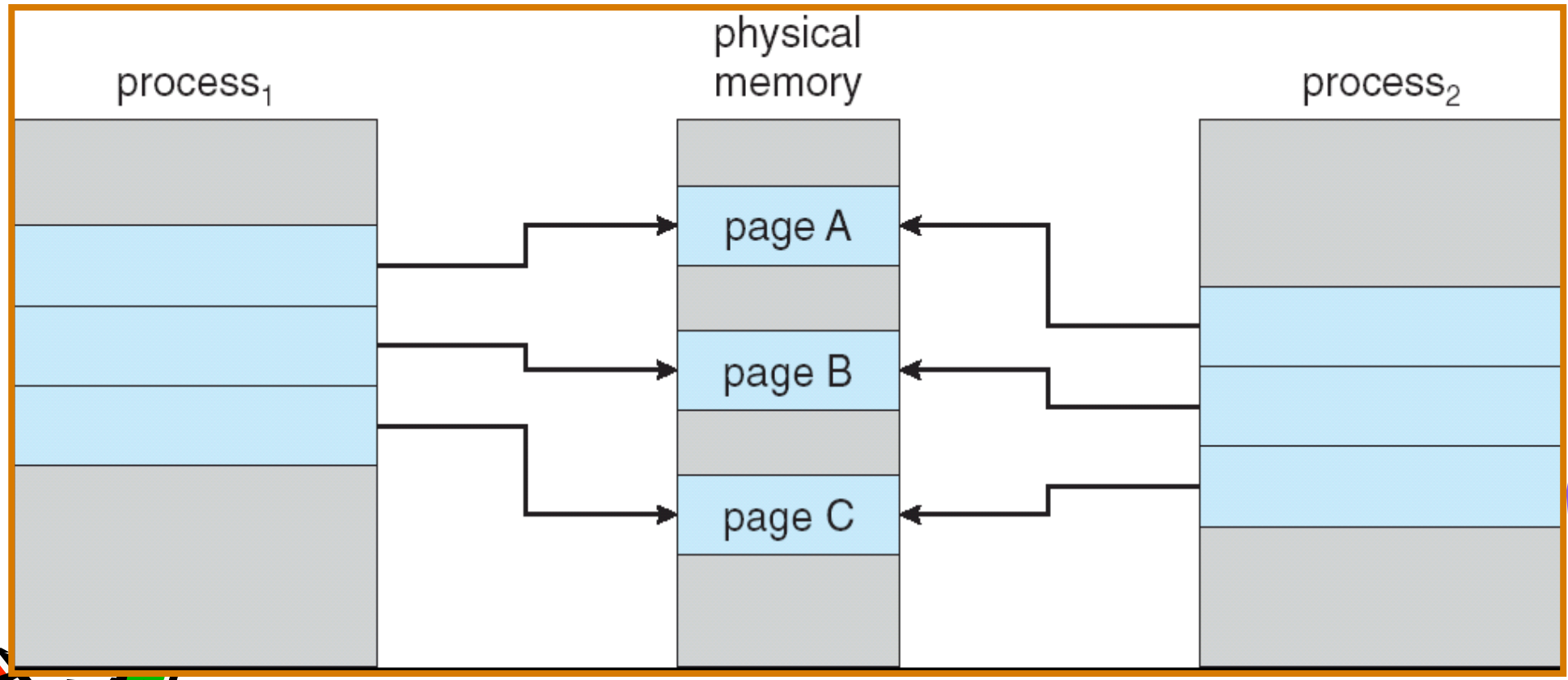
- *Copy-on-Write* (COW) ⇒ Teknik yang memfasilitasi proses *parent* dan *child* untuk *share* page yang sama dalam memory.
- Implementasi → Windows XP, Linux dan Solaris
- Page diset *copy-on-write* agar dapat dimodifikasi
- Jika modifikasi dilakukan, page di duplikasi dan ditempatkan pada posisi berbeda dimemori fisik.
- Sistem operasi menyediakan sebuah pool yang berisikan page-page yang kosong (*pool of free page*).
- Page kosong diberikan kepada:
  - Bagian stack atau heap suatu proses yang terus berkembang
  - Proses yang melakukan *copy-on-write*

Copy-on-Write



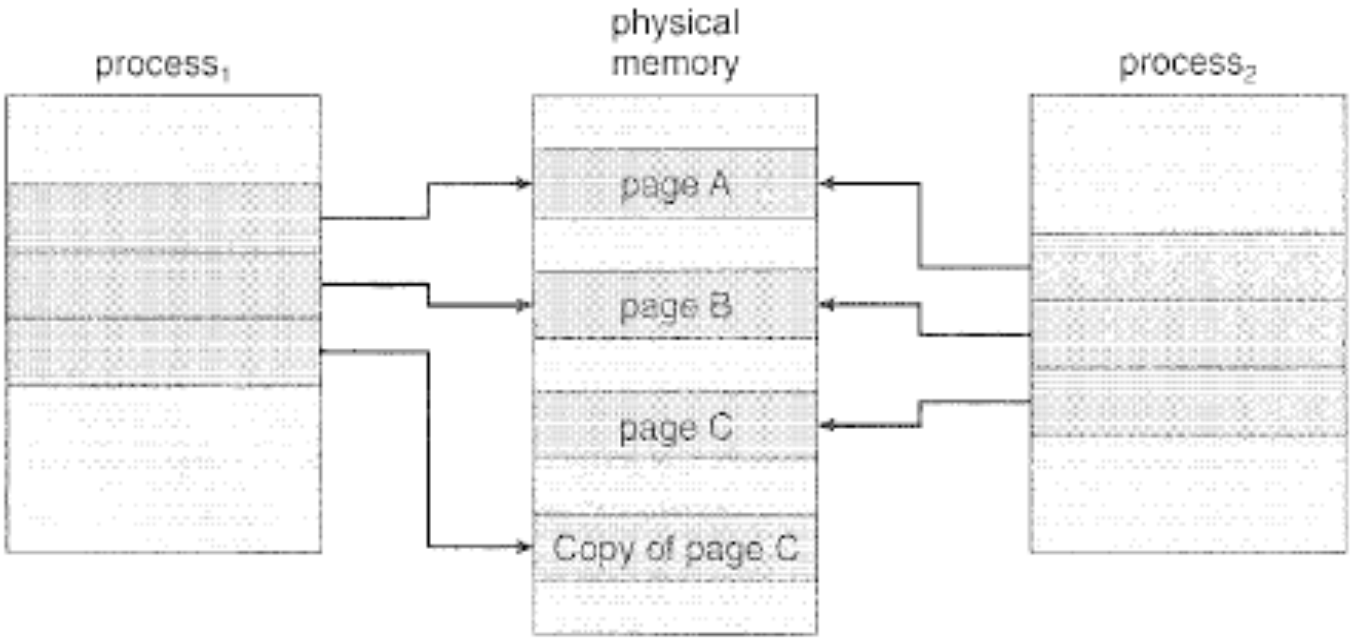
- Alokasi page baru menggunakan teknik **zero-fill-on-demand** (mengkosongkan page yang lama)
- vfork() ⇒ (*Virtual Memory fork*())
  - proses *parent* dihentikan sementara (*suspended*), proses child menggunakan alamat logika parent.
  - Perubahan page si parent, tidak menggunakan *copy-on-write*
  - Pengubahan page terlihat saat proses parent di resume
  - *Caution*: proses child tidak mengubah alamat logika parent
  - Tujuan → efisiensi memori bagi process child yang langsung mengeksekusi exec() setelah vfork()
  - berjalan di varian unix

Copy-on-Write



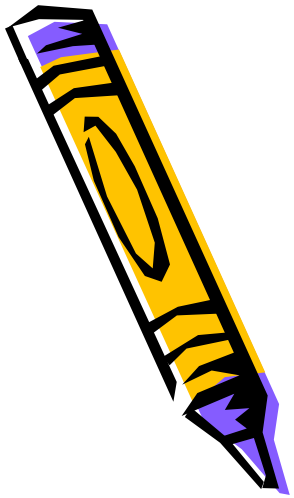
Sebelum proses 1 melakukan modifikasi page c

Copy-on-Write



Setelah proses 1 melakukan modifikasi page c

Pergantian Page (*Page Replacement*)

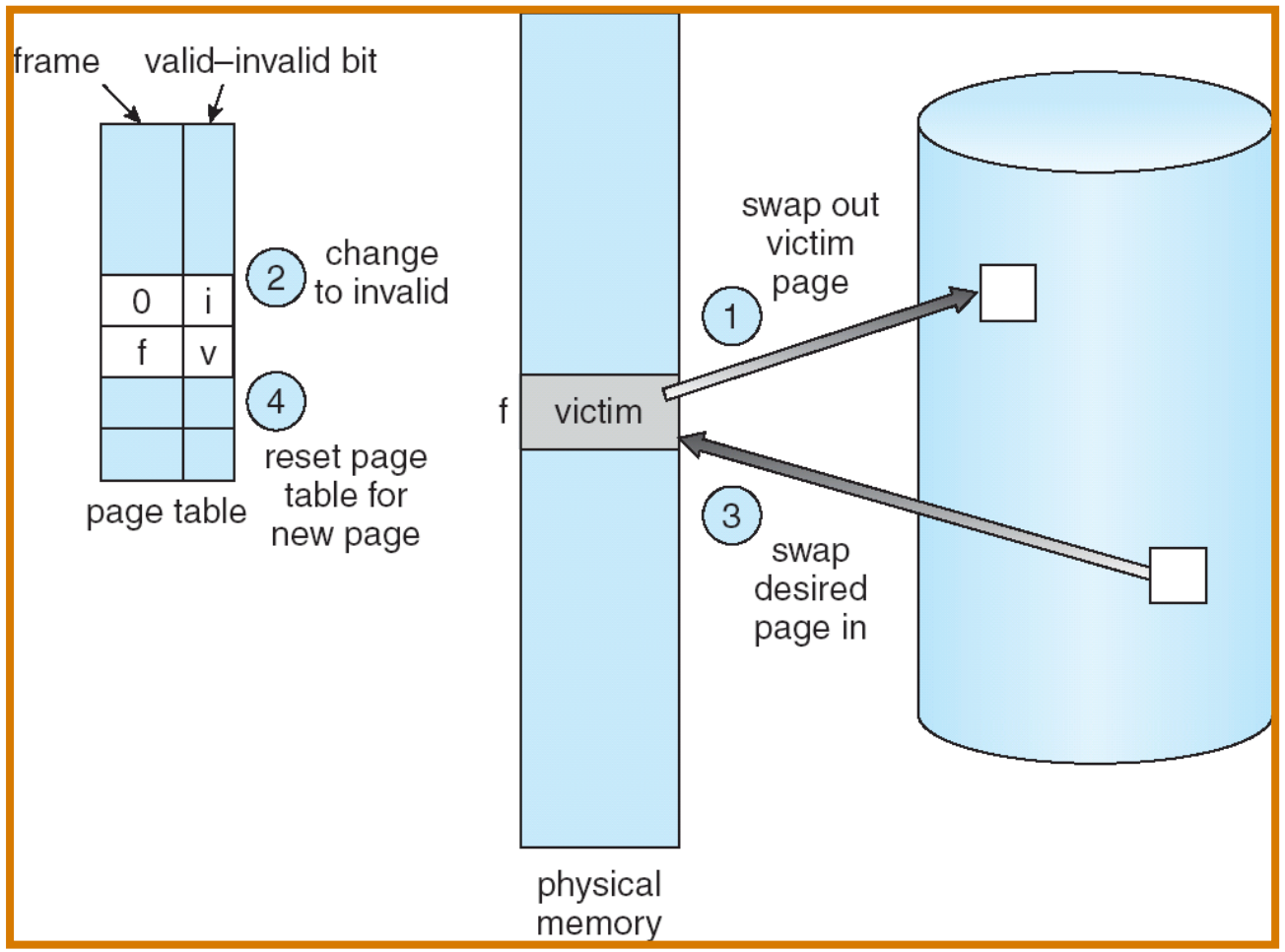


- Fakta :
  - Dari 10 page proses A ⇒ 5 page yang digunakan
  - 40 Frame ⇒ diisi 8 Proses (1 proses @ 5 frame)
  - 40 Frame ⇒ diisi 6 Proses (1 proses @ 10 frame) ? *over-allocating*!
- *Over-allocating⇒ page-fault ⇒ all memory in use, no space available*
- Solusi alternatif ?
  - Hapus proses user
  - *Swap out* proses
  - Pergantian page (*Page replacement*)

Pergantian Page (*Page Replacement*)

- Pergantian page – cari frame dalam memory yang sedang tidak digunakan kemudian lakukan *page out*
  - Frame *A* dipindahkan ke *swap space*
  - Page *A* tidak lagi berada dimemori ⇒ ubah *page table*
  - Gunakan frame kosong *A* untuk page yang baru
- Algoritma Pergantian page diinteragrasikan pada *page-fault service routine*

Pergantian Page (*Page Replacement*)

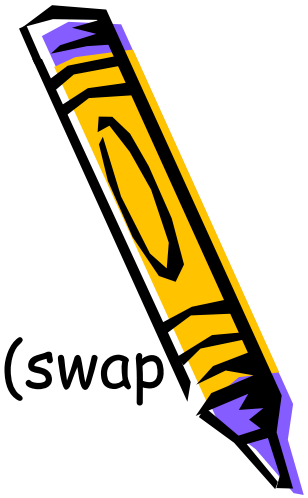


alur pergantian page (*page replacement*)

Pergantian Page (*Page Replacement*)

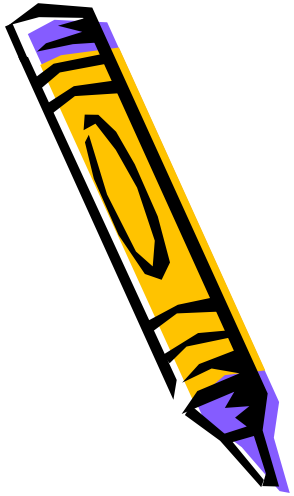
- Langkah *page fault service routine* dengan pergantian page
  1. Cari lokasi page dalam disk
  2. Cari frame kosong di memori
     a) Gunakan frame kosong Jika ada
     b) Jika tidak ada, gunakan algoritma page-replacement untuk memilih *victim frame* yang akan dikeluarkan
     c) Pindahkan *victim frame* ke disk; ubah page dan frame table
  3. Pindahkan page baru dari disk ke frame; ubah page dan frame table
  4. Restart proses user

Pergantian Page (*Page Replacement*)



- *page fault service routine* ⇒ dua kali page transfer (swap out & swap in)
- Agar satu kali transfer? gunakan page table dengan *modify bit* !
- *Modify bit* (*dirty bit*)
  - Jika bit diset, page telah berubah
    - page di memori != page didisk
    - page harus dipindah ke disk
  - Jika bit tidak diset, page tidak berubah
    - page dimemori == page didisk
    - page tidak dipindah ke disk, langsung dihapus!
      Contoh : shared code, binary code

Pertanyaan



- Apa yang dimaksud dengan memori virtual?
- Apa yang dimaksud dengan demand paging
- Apa yang dimaksud dengan Lazy Swapper
- Apa perbedaan pager dan swapper
- Apa yang yang dimaksud dengan copy-on-write?
- Apa perbedaaan vfork() dengan copy-on-write?
- Terangkan alur proses pergantian page?
- Apa manfaat modify/dirty bit bagi proses pergantian page?