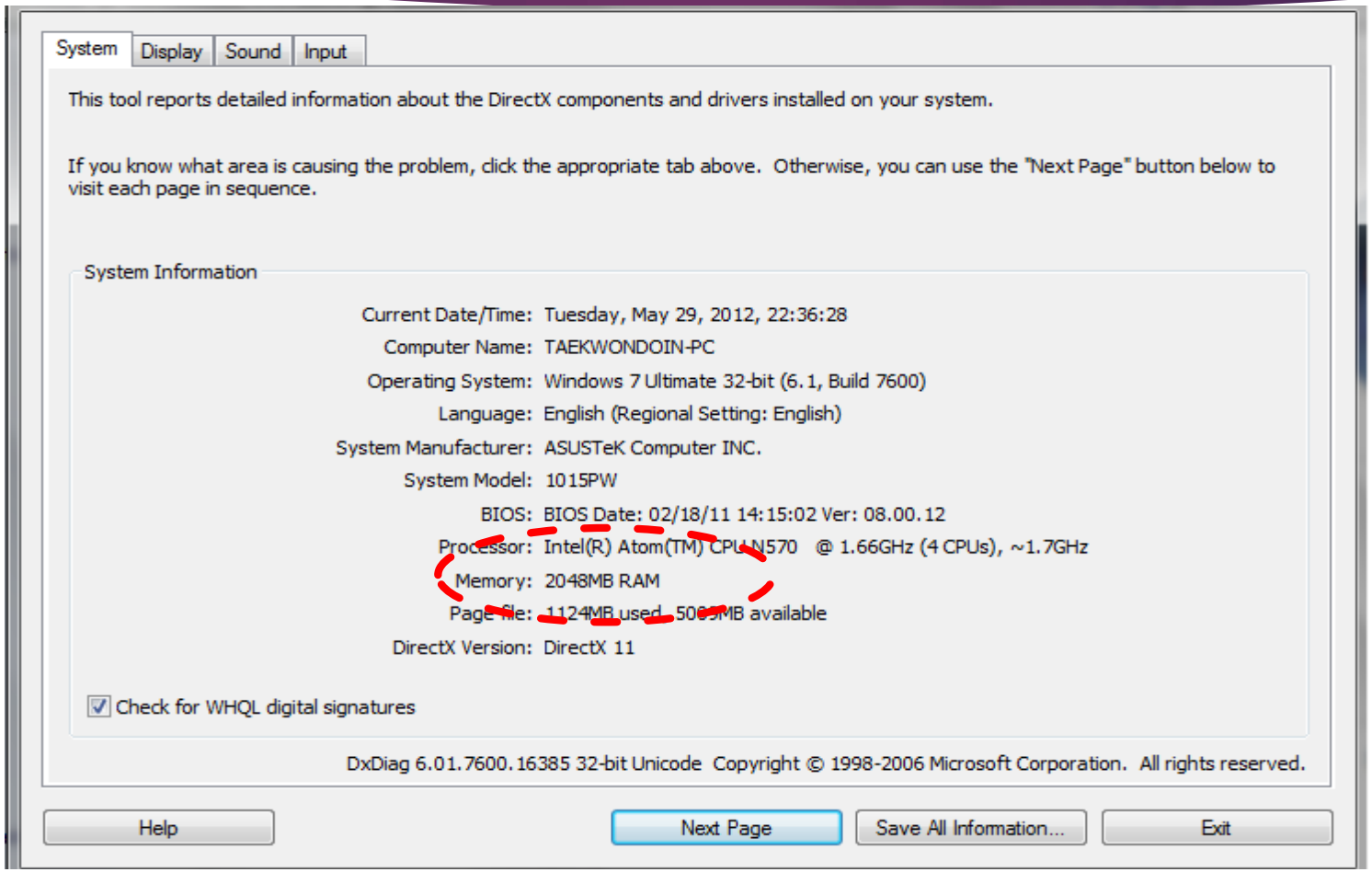
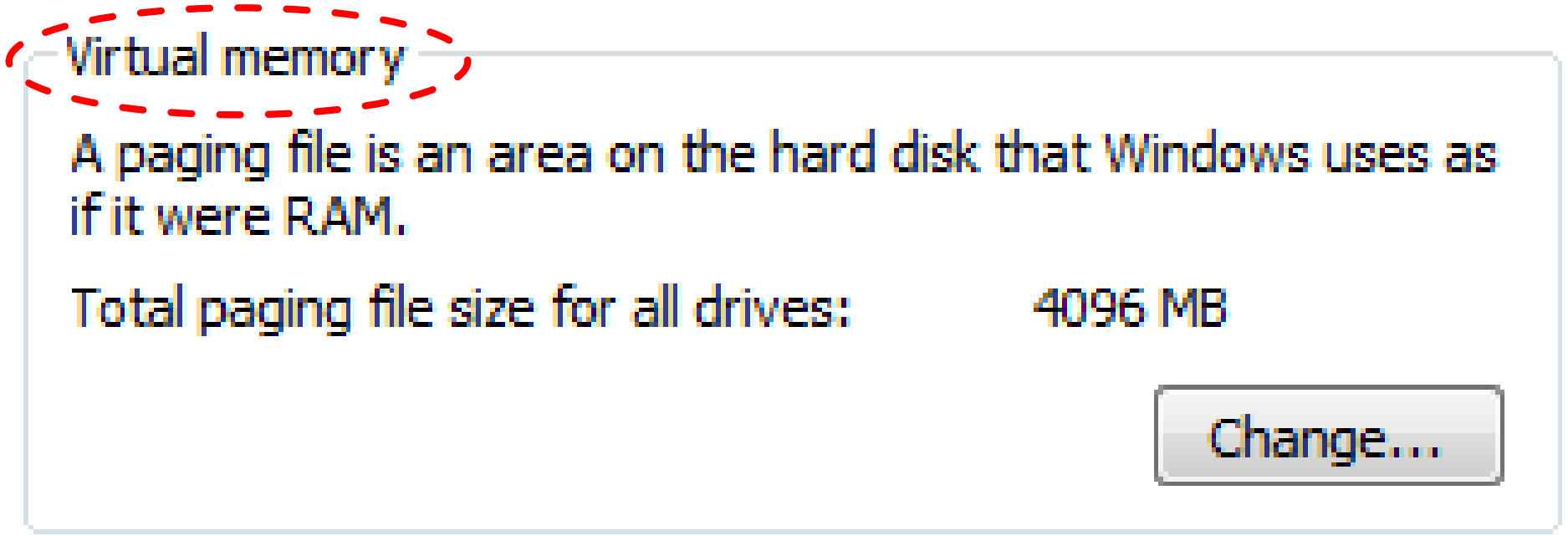
SISTEM OPERASI

PERTEMUAN XII : *VIRTUAL MEMORY & PAGE REPLACEMENT*

VIRTUAL MEMORY



VIRTUAL MEMORY



VIRTUAL MEMORY

- **Virtual Memory** ➔ suatu teknik yang memisahkan antara memori logis dan memori fisik dalam eksekusi suatu proses
- **Prinsip Virtual Memory** ➔

  “Kecepatan maksimum eksekusi proses di memori virtual dapat sama, tetapi tidak pernah melampaui kecepatan eksekusi proses yang sama di sistem tanpa menggunakan memori virtual.”

VIRTUAL MEMORY

- Dengan kata lain virtual memory digunakan untuk menampung program dan data yang tidak cukup di memory fisik
- Keuntungan **Virtual Memory :**
  - Ruang menjadi lebih leluasa karena berkurangnya memori fisik yang digunakan.
  - Lebih responsif dan bertambahnya jumlah *user program* yang dapat dilayani.
  - Virtual Memory dapat melebihi daya tampung dari memori utama yang tersedia.

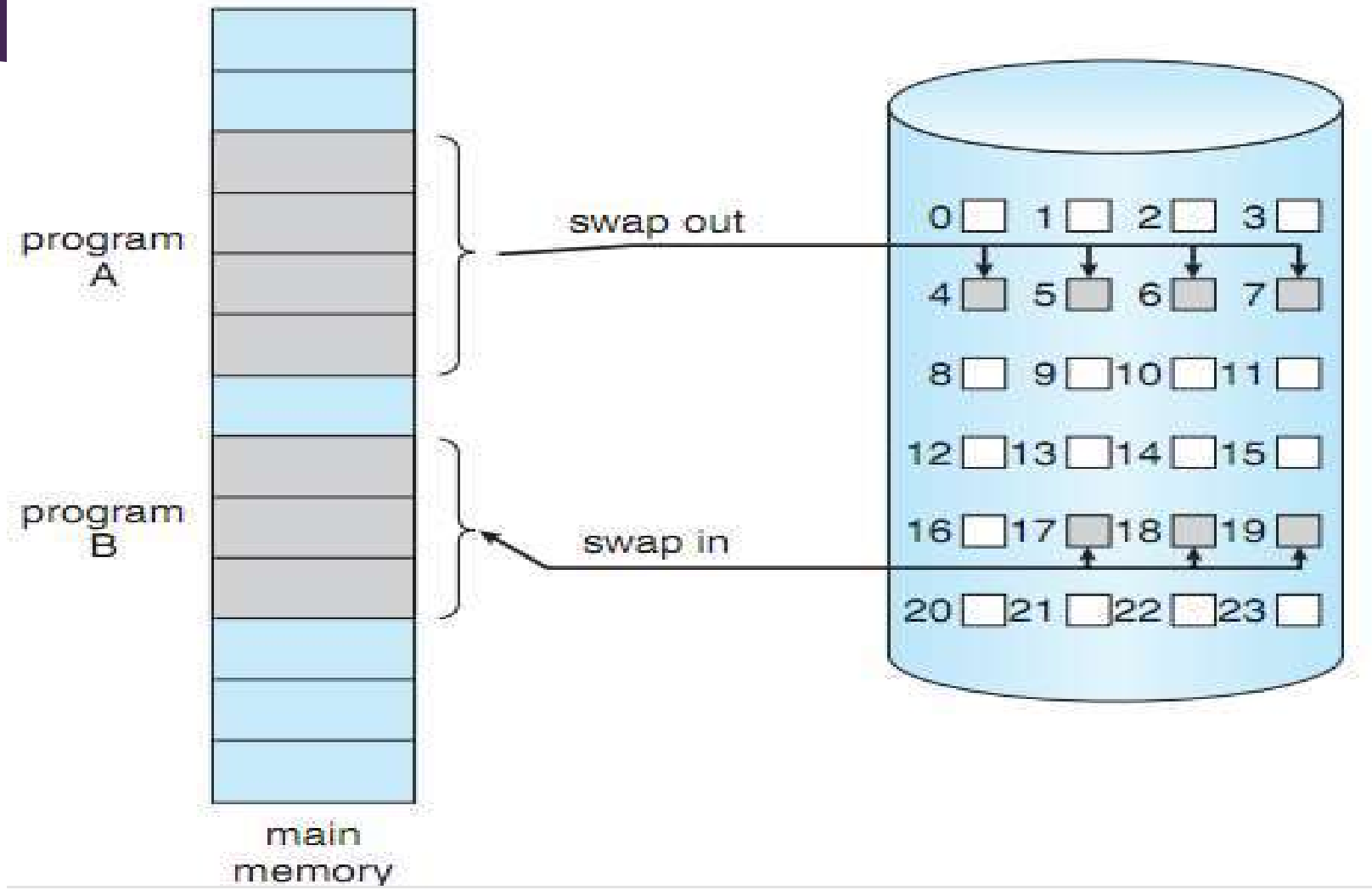
VIRTUAL MEMORY

- Implementasi Virtual Memory dapat dilakukan dengan dua cara :
  - **Demand Paging**
  - **Demand Segmentation**

DEMAND PAGING

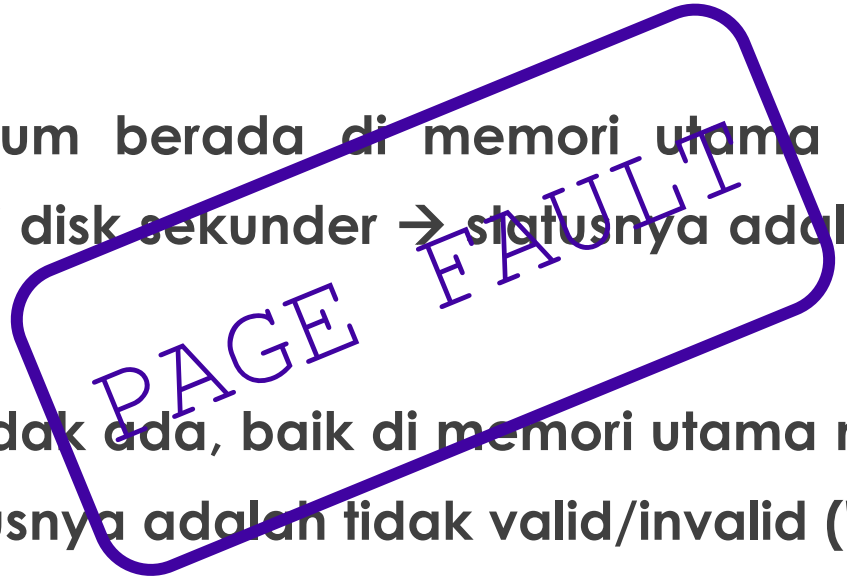
- ***Demand Paging*** ➔ permintaan pemberian halaman *(page)*
- Dalam Demand *Paging* digunakan metode ***lazy swapper***
- ***Lazy swapper*** ➔ tidak pernah menukar sebuah halaman ke dalam memori utama kecuali halaman tersebut diperlukan.
- Sistem *Demand Paging* sama halnya dengan sistem *paging* dengan *swapping*

DEMAND PAGING



DEMAND PAGING

- Dalam melakukan pengecekan pada halaman yang dibutuhkan oleh suatu proses, terdapat tiga kemungkinan kasus yang dapat terjadi, yaitu:
  - **Halaman ada dan sudah langsung berada di memori utama → statusnya adalah valid ("v" atau "1")**
  - **Halaman ada tetapi belum berada di memori utama atau dengan kata lain halaman masih berada di disk sekunder → statusnya adalah tidak valid/invalid ("i" atau "0")**
  - **Halaman benar - benar tidak ada, baik di memori utama maupun di disk sekunder (*invalid reference*) → statusnya adalah tidak valid/invalid ("i" atau "0")**



DEMAND PAGING ➔ Skema Bit Valid - Tidak Valid

- **Skema bit valid - tidak valid** → menentukan halaman mana yang ada dan halaman mana yang tidak ada di dalam memori utama.
- Pengaturan bit dilakukan sebagai berikut:
  - **Bit = 1** berarti halaman **berada** di memori utama
  - **Bit = 0** berarti halaman **tidak berada** di memori utama

"Apabila ternyata hasil dari mengartikan alamat melalui page table menghasilkan bit halaman yang bernilai 0, maka akan menyebabkan terjadinya **PAGE FAULT .**"

DEMAND PAGING ➔ Skema Bit Valid - Tidak Valid

- **Page fault** adalah interupsi yang terjadi ketika halaman yang diminta/dibutuhkan oleh suatu proses tidak berada di memori utama.
- Proses yang sedang berjalan akan mengakses page table (tabel halaman) untuk mendapatkan referensi halaman yang diinginkan.
- Page fault dapat diketahui/dideteksi dari penggunaan skema bit valid-tidak valid ini.
- Bagian inilah yang menandakan terjadinya suatu permintaan pemberian halaman .

DEMAND PAGING ➔ Skema Bit Valid - Tidak Valid

“Jika suatu proses mencoba untuk mengakses suatu halaman dengan bit yang di-set tidak valid maka **PAGE FAULT** akan terjadi. “

dan

“Proses akan dihentikan sementara halaman yang diminta/dibutuhkan dicari didalam disk.”

DEMAND PAGING ➔ Skema Bit Valid - Tidak Valid

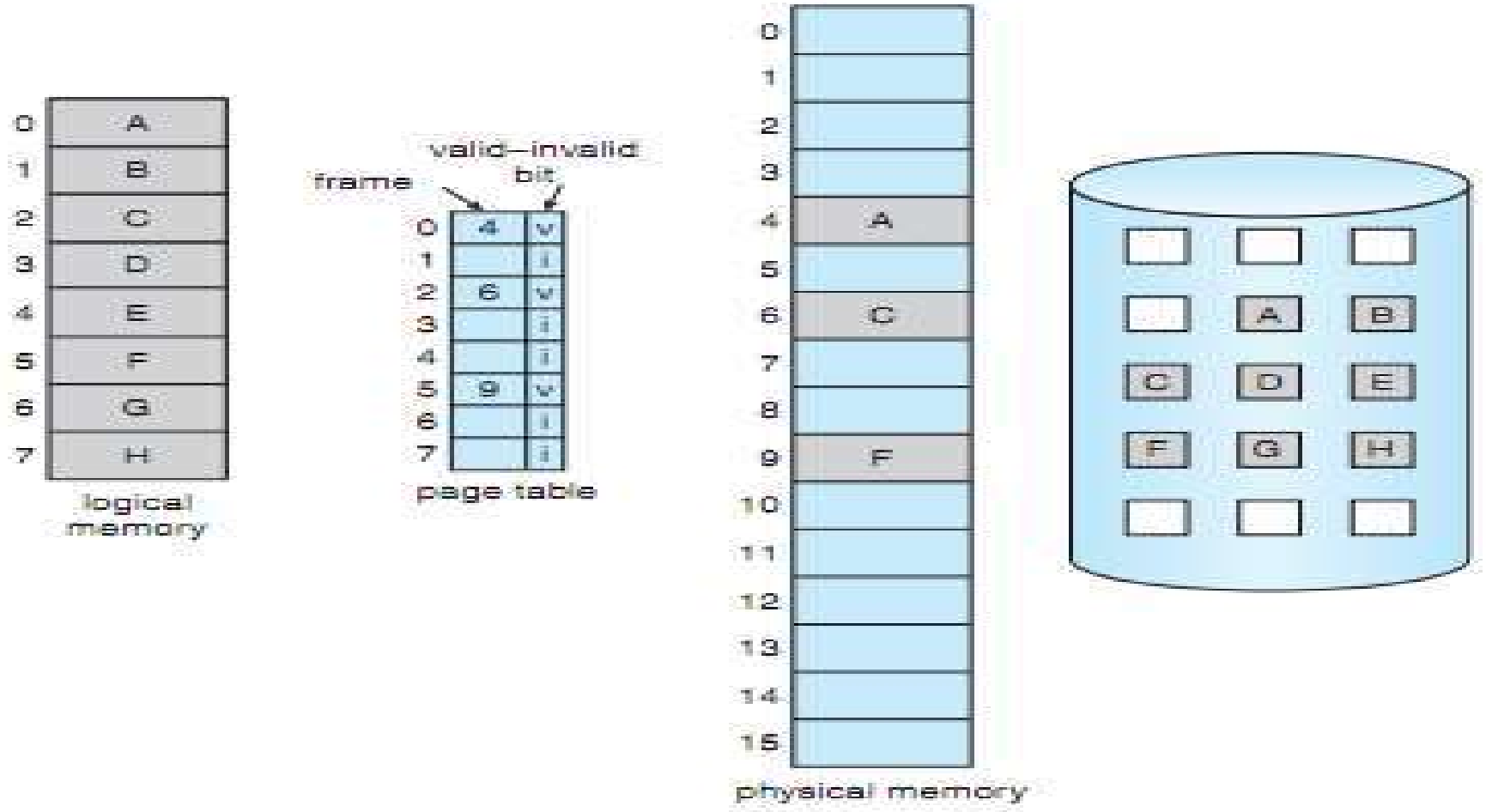


Figure 8.5 Page table when some pages are not in main memory.

DEMAND PAGING ➔ Penanganan Page Fault

- Cara-cara untuk mengatasi *page fault* :
  - CPU Mengecek *page tabel* untuk menentukan bit referensi valid atau invalid
  - Jika referensi invalid, maka proses akan dihentikan, Namun jika referensi alamatnya valid atau legal maka halaman yang diinginkan akan diambil dari disk.
  - *Page Table* (Tabel halaman) akan diatur ulang lagi sesuai dengan kondisi yang baru.
  - Jika tidak terdapat ruang kosong di memori utama untuk menaruh halaman baru, maka akan dilakukan pergantian halaman menggunakan algoritma *Page Replacement*

DEMAND PAGING ➔ Penanganan Page Fault

- Cara-cara untuk mengatasi *page fault* :
  - Setelah halaman yang diinginkan sudah dibawa ke memori utama (fisik) maka proses dapat diulang kembali. Dengan demikian proses sudah bisa mengakses halaman karena halaman telah diletakkan ke memori utama (fisik).

DEMAND PAGING ➔ Penanganan Page Fault

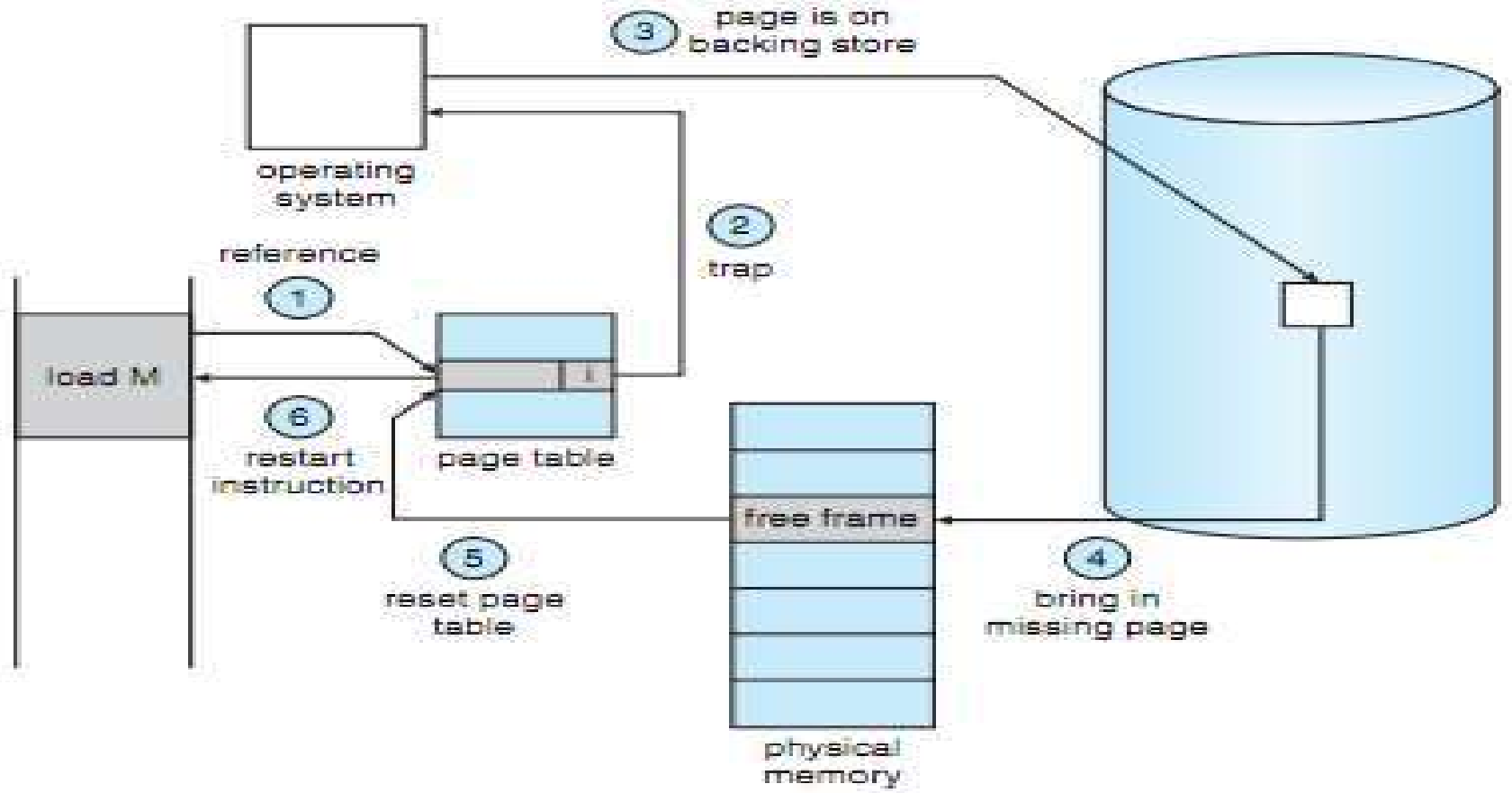


Figure 8.6 Steps in handling a page fault.

DEMAND PAGING ➔ Penanganan Page Fault

Ada tiga komponen yang akan dihadapi pada saat melayani page fault:

- ✓ Melayani interupsi page fault
- ✓ Membaca halaman
- ✓ Mengulang kembali proses

KINERJA *DEMAND PAGING*

- ✓ Kinerja *demand paging* dapat dihitung dengan menggunakan effective access time.

$$\textbf{\textit{effective access time} = (1 - p) x \textit{ma} + p x \textit{page fault time}}$$

  - ma adalah *memory access time*, yang pada umumnya berkisar antara 10 hingga 200 nanosecond.
  - p adalah probabilitas terjadinya *page fault*, yang berkisar antara 0 hingga 1. Jika p sama dengan 0 yang berarti bahwa tidak pernah terjadi *page fault*, maka *effective access time* akan sama dengan *memory access time*. Sedangkan jika p sama dengan 1, yang berarti bahwa semua halaman mengalami *page fault*, maka *effective access time*-nya akan semaikin meningkat.

KINERJA *DEMAND PAGING*

- ✓ Diketahui waktu pengaksesan memori (ma) sebesar 100 ns. Waktu page fault sebesar 20 ms.
- ✓ Maka :

effective access time = (1 - p) x ma + p x page fault time

= (1 - p) x 100 + p x 20000000

= 100 - 100p + 20000000p

= 100 + 19.999.900p nanosecond

KINERJA *DEMAND PAGING*

effective access time = 100 + 19.999.900p nanosecond

- Jika diketahui dalam 1000 *memory access* terdapat 1 *page fault* maka :

effective access time = 100 + 19.999.900 (1/1000)p nanosecond

= 20099.9 nanosecond

= 20.1 microsecond

PAGE REPLACEMENT

"*Page Replacement* diperlukan pada saat dimana proses dieksekusi perlu frame bebas tambahan tetapi tidak tersedia frame bebas"

PAGE REPLACEMENT

Prinsip proses *Page Replacement* sebagai berikut :

- ✓ Proses meminta Halaman tertentu.
- ✓ Jika halaman berada di memori, tidak dilakukan *Page Replacement*.
- ✓ Jika halaman tidak berada di memori, maka :
  - ✓ Jika ada frame kosong, maka halaman tsb di-*load* ke dalam frame yang kosong.
  - ✓ Jika tidak ada frame yang kosong, gunakan algoritma *page replacement* untuk memilih bingkai "korban“
  - ✓ Pindahkan bingkai "korban" tersebut ke disk dan sesuaikan tabel halaman.
- ✓ *Update page table* dan *Restart* Proses.

PAGE REPLACEMENT

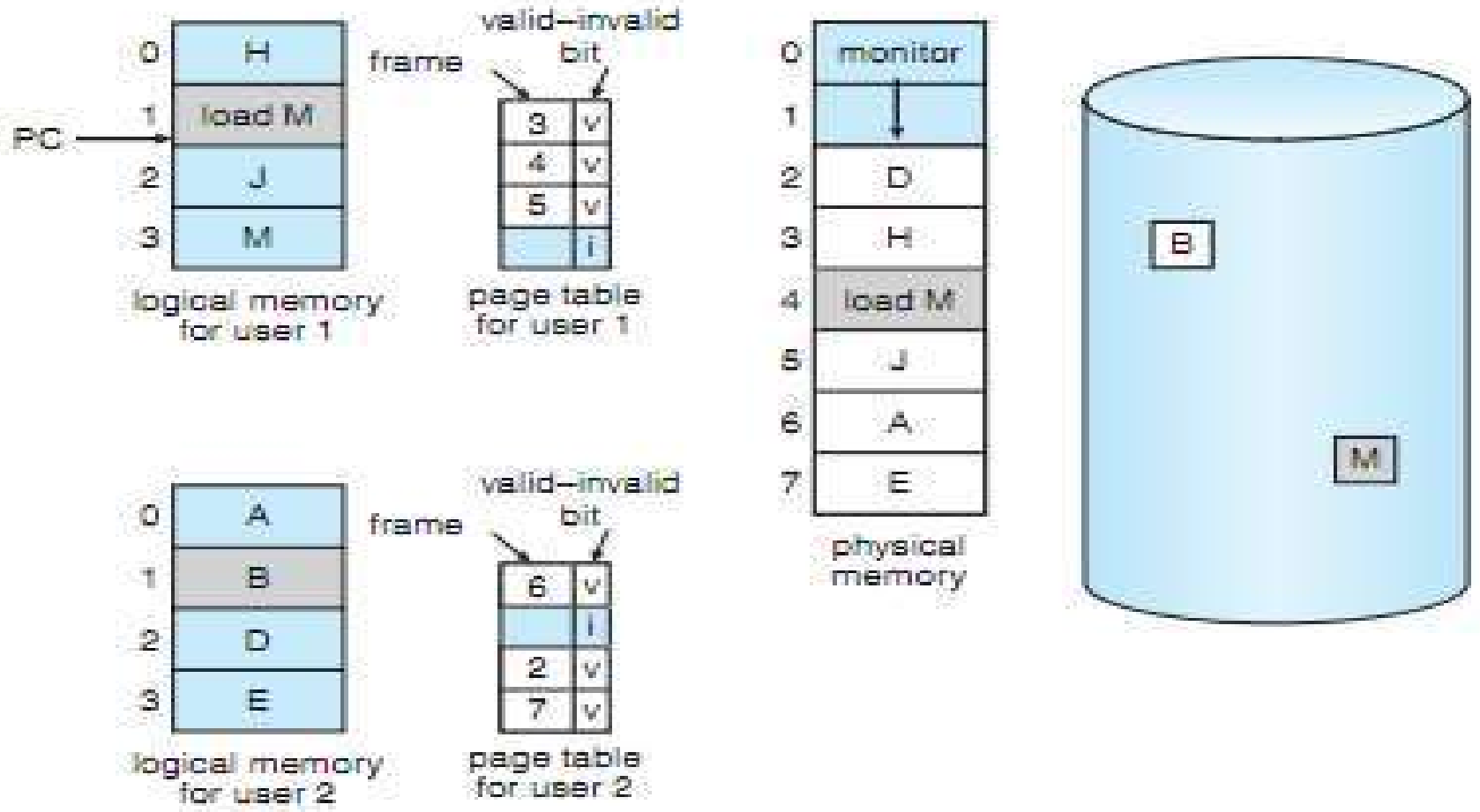


Figure 8.9 Need for page replacement.

PAGE REPLACEMENT

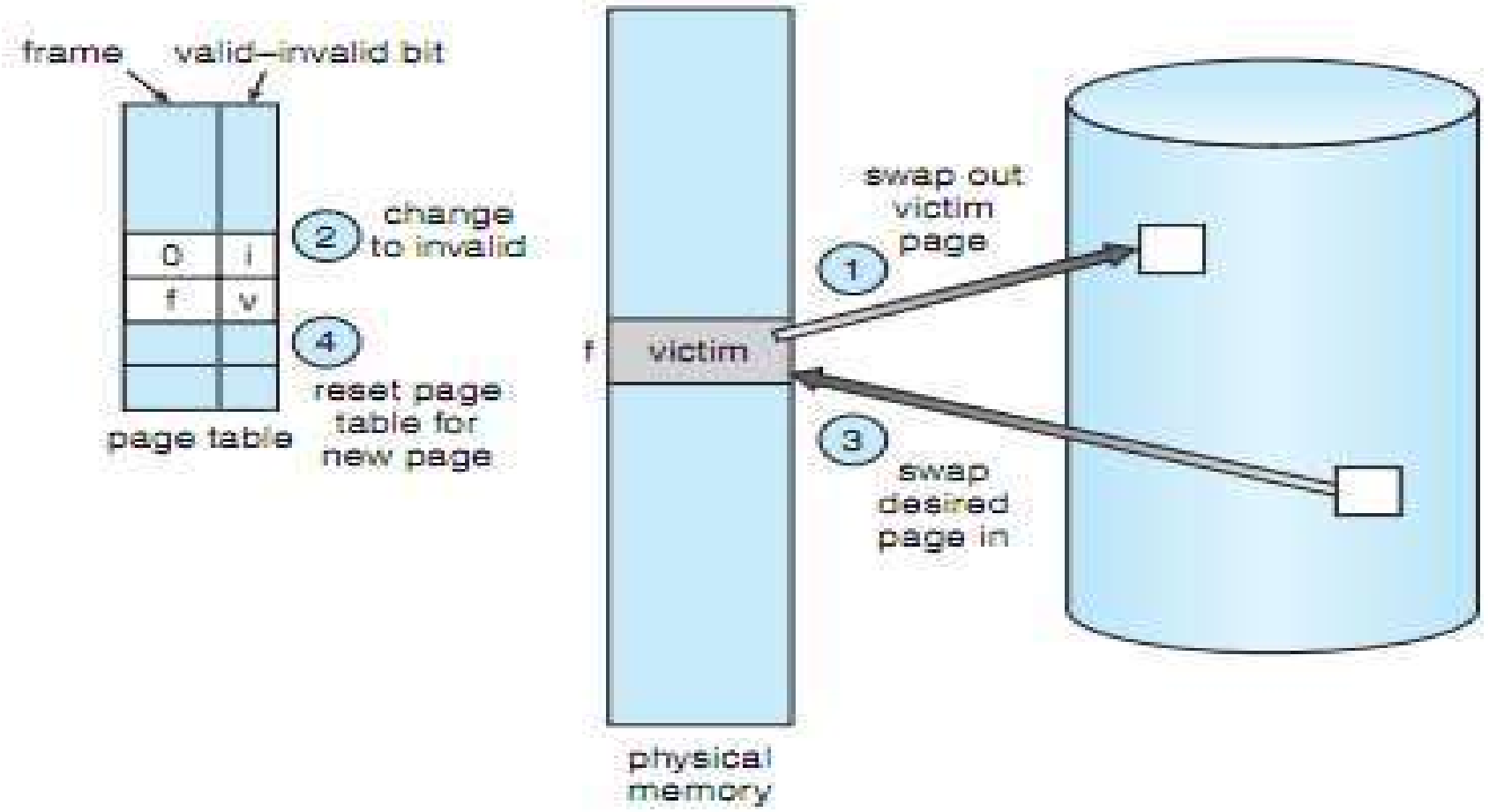


Figure 8.10 Page replacement.

ALGORITMA *PAGE REPLACEMENT*

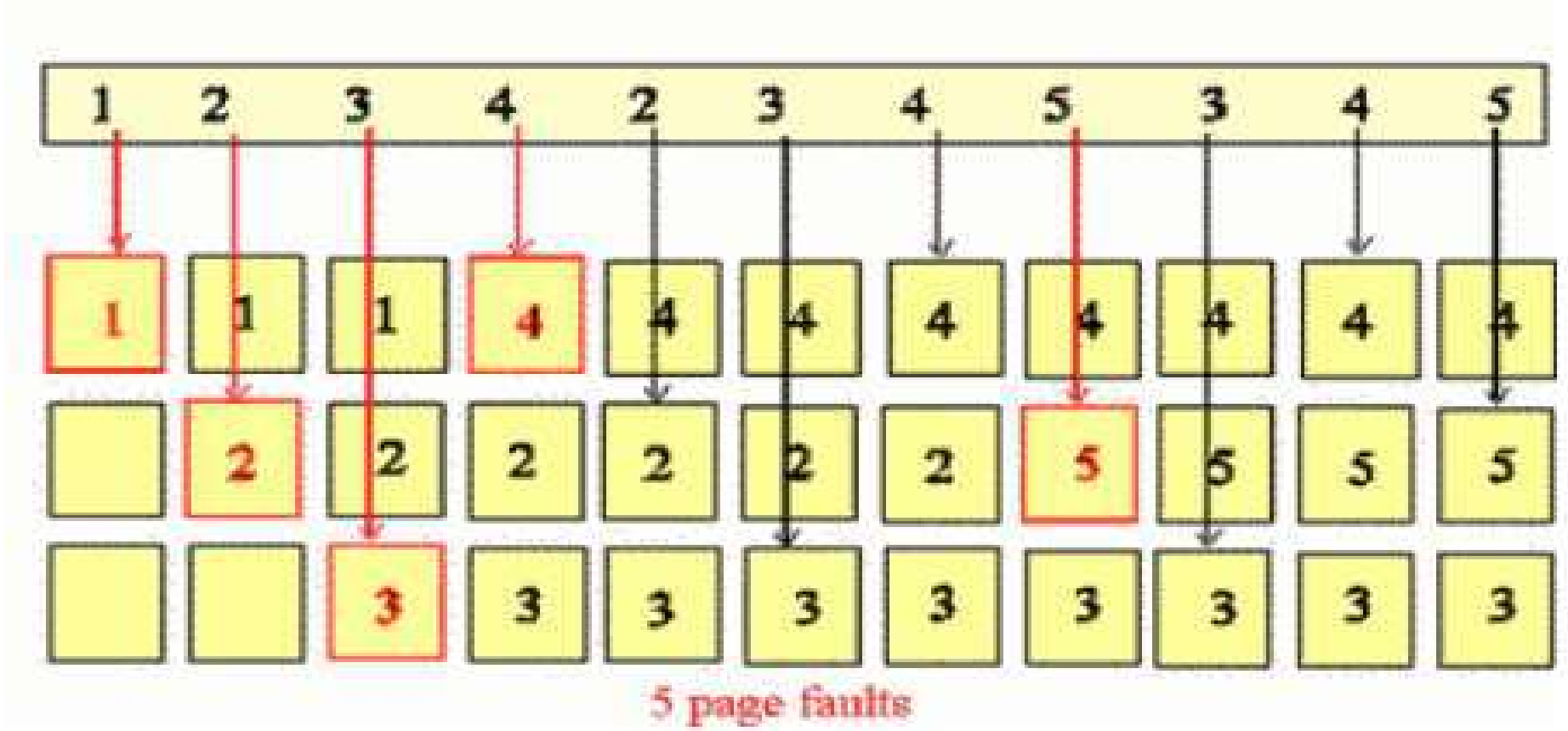
- ✓ Bertujuan untuk mendapatkan Page Fault terendah.
- ✓ Ada beberapa algoritma *page replacement* :
  - ✓ Algoritma FIFO
  - ✓ Algoritma Optimal
  - ✓ Algoritma LRU (Least Recently Used)

ALGORITMA *PAGE REPLACEMENT*

- ✓ Algoritma FIFO
  - ✓ Merupakan algoritma yang paling sederhana
  - ✓ Jika ada suatu *page* yang akan ditempatkan, maka posisi *page* yang paling lama berada di memori yang akan digantikan

Gambar 6.2. Algoritma FIFO

ALGORITMA *PAGE REPLACEMENT*



ALGORITMA *PAGE REPLACEMENT*

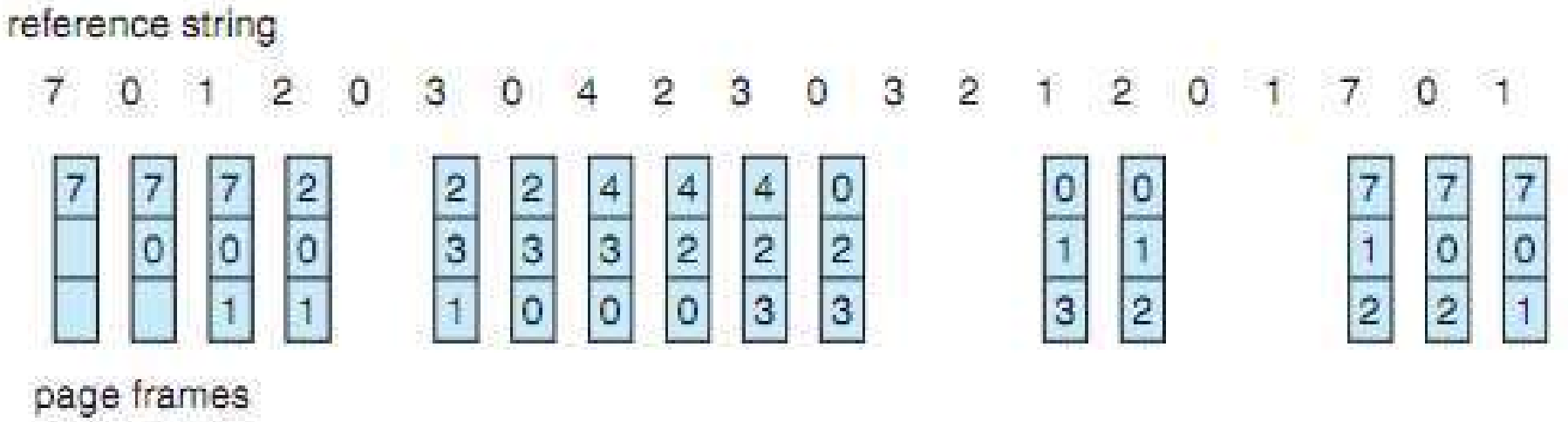
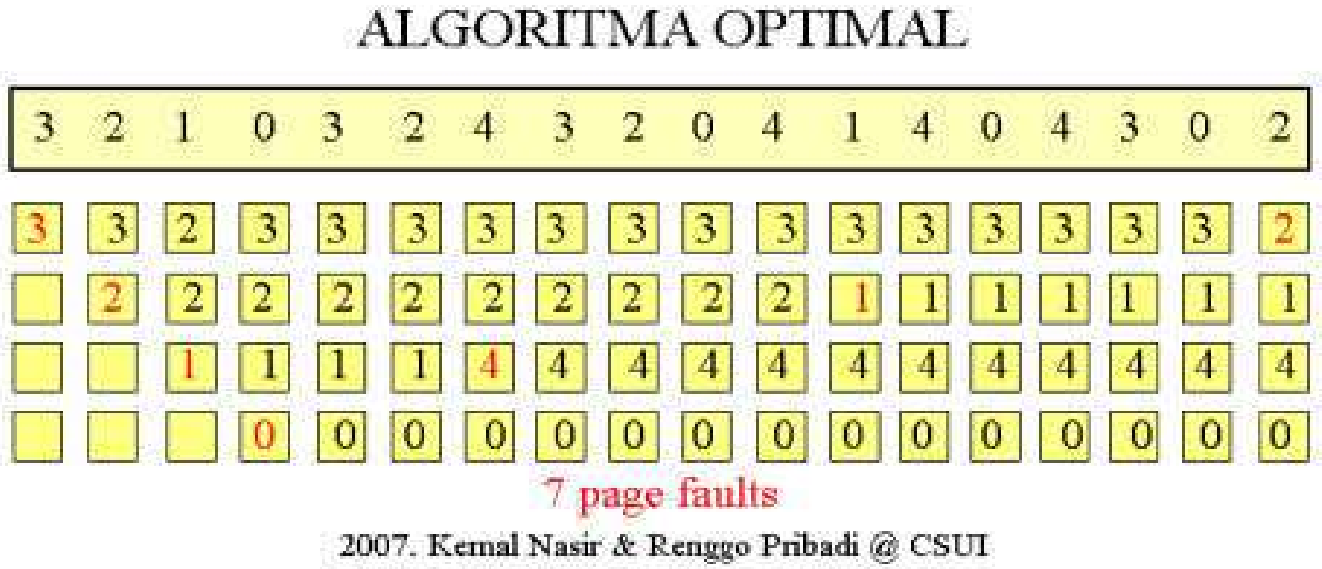


Figure 8.12 FIFO page-replacement algorithm.

ALGORITMA *PAGE REPLACEMENT*

- ✓ Algoritma Optimal
  - ✓ Mengganti *page* yang tidak digunakan dalam waktu dekat
  - ✓ Mempunyai rata-rata *page fault* terendah

Gambar 6.4. Algoritma Optimal



ALGORITMA *PAGE REPLACEMENT*

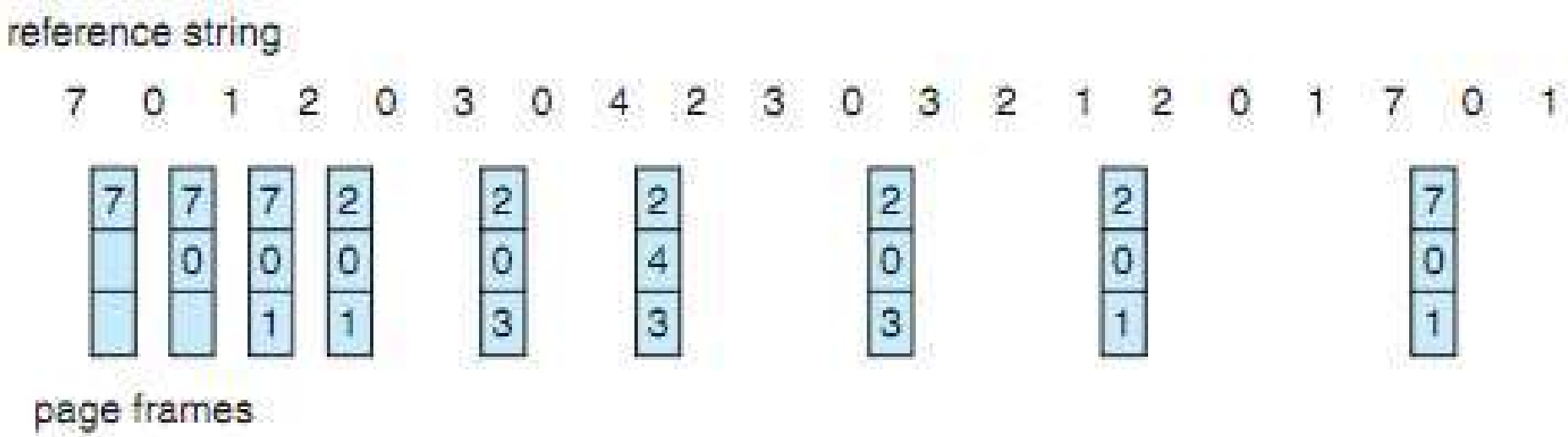


Figure 8.14 Optimal page-replacement algorithm.

ALGORITMA *PAGE REPLACEMENT*

- ✓ Algoritma LRU
  - ✓ Merupakan perpaduan dari FIFO dan Optimal
  - ✓ *Page* yang diganti adalah *page* yang telah lama tidak digunakan.

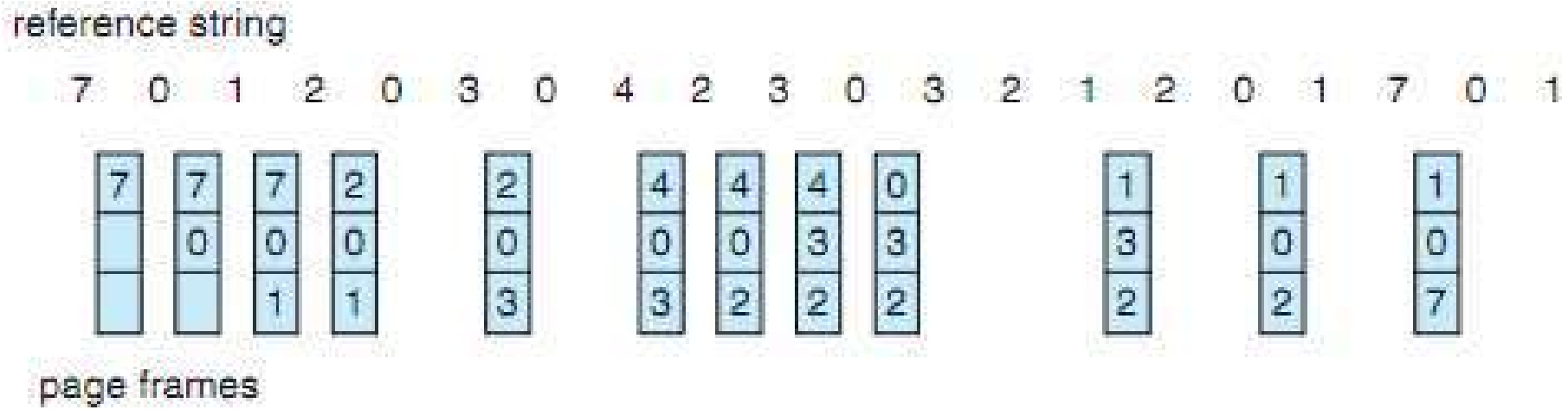


Figure 8.15 LRU page-replacement algorithm.