

The background features abstract, colorful swirls in shades of purple, green, and blue, interspersed with yellow starburst shapes. The text is centered over this pattern.

# **Sistem Operasi**

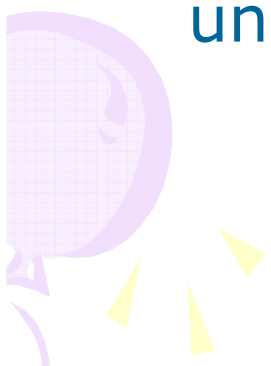
## **9**

**“Virtual Memory”**

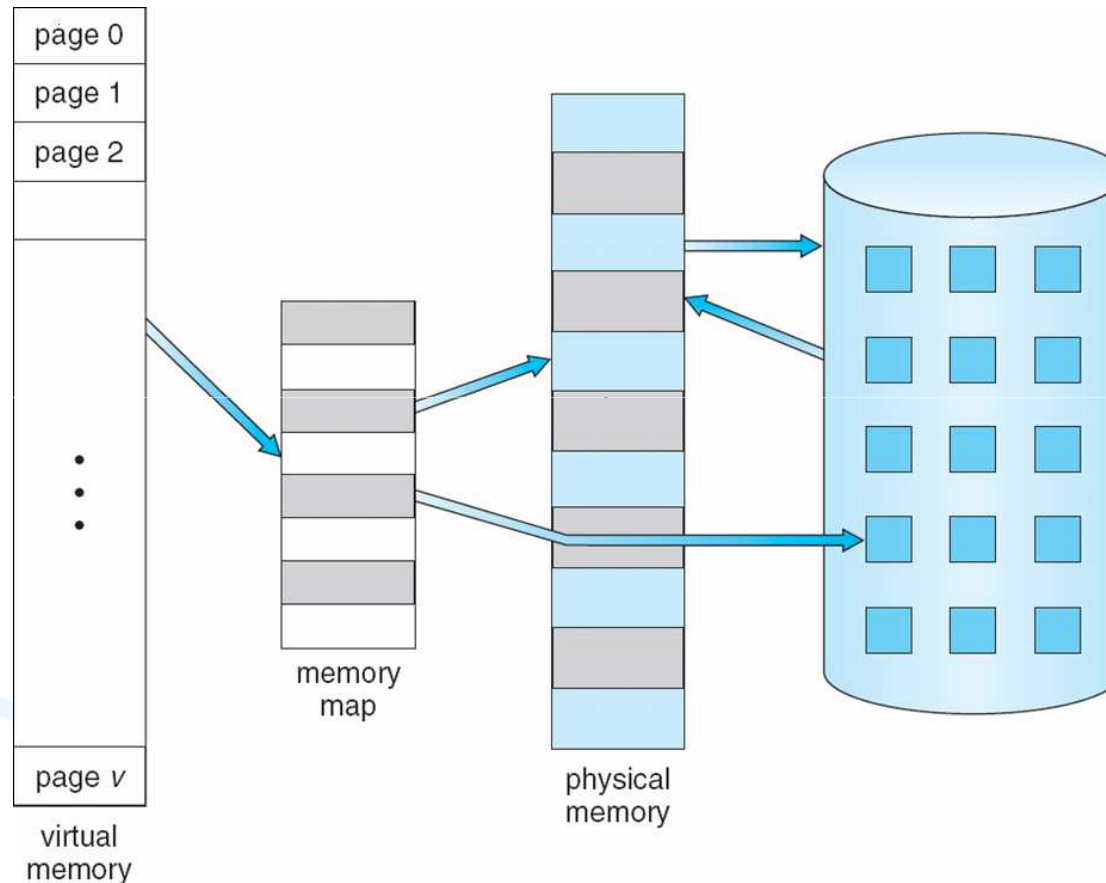
**Antonius Rachmat C, S.Kom,  
M.Cs**



# Virtual Memory

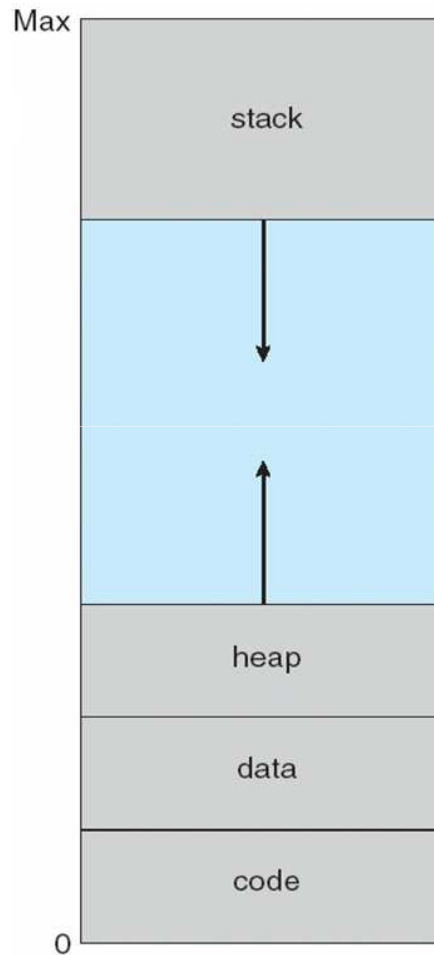
- **Tidak bisa** semua memory logic dipetakan ke memory fisik, walau **dynamic loading** bisa melakukannya
  - **Memori virtual** merupakan suatu teknik yang memisahkan antara memori logis dan memori fisiknya.
  - Hanya bagian dari program yg **perlu** saja, berada di memory fisik untuk eksekusi
  - Logical address space dapat berukuran **lebih besar** daripada physical address space
  - Memperbolehkan virtual address spaces pada VM untuk **disharing** oleh beberapa processes
- 

# Virtual Memory That is Larger Than Physical Memory



**Virtual address space**

# Virtual-address Space



Virtual Address Space yang memiliki **Hole** disebut **Sparse Address Space**



# Program yg tidak perlu berada di Memory Utama

- Program-program (kode2) yg digunakan sbg **error handling**, yg jarang digunakan karena jarang terjadi
- Array, list, atau tabel yg kapasitasnya tidak **terpakai semuanya**
- Fungsi-fungsi yg tidak dipakai secara **bersamaan**
- Program-program yang tidak digunakan secara **real time**



# Keuntungan Virtual Memory

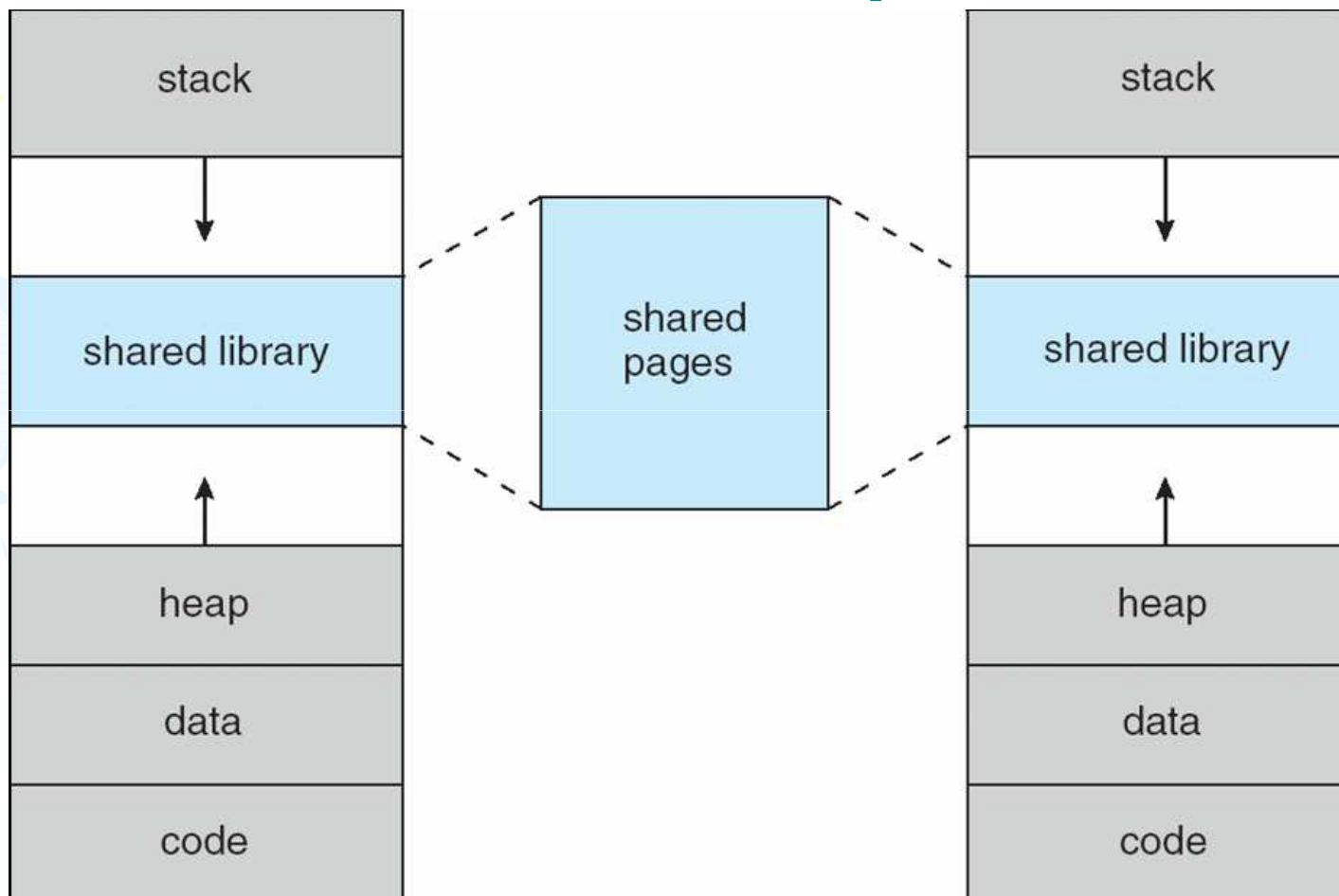
- Berkurangnya total memori fisik yang dibutuhkan.
- Meningkatnya respon, karena tidak deadlock.
- Bertambahnya jumlah user yang dapat dilayani.
- Memori virtual bisa melebihi daya tampung dari memori utama yang tersedia.



# Implementasi Virtual Memory

- Virtual Memory digunakan pada:
  - Multiprogramming
    - Banyak program dapat dijalankan dalam satu waktu
- Memori virtual dapat dilakukan dengan cara:
  - Demand paging
  - Demand segmentation (tidak dibahas)

# Shared Library Using Virtual Memory



Sharing antar proses bisa diciptakan dgn **fork()**

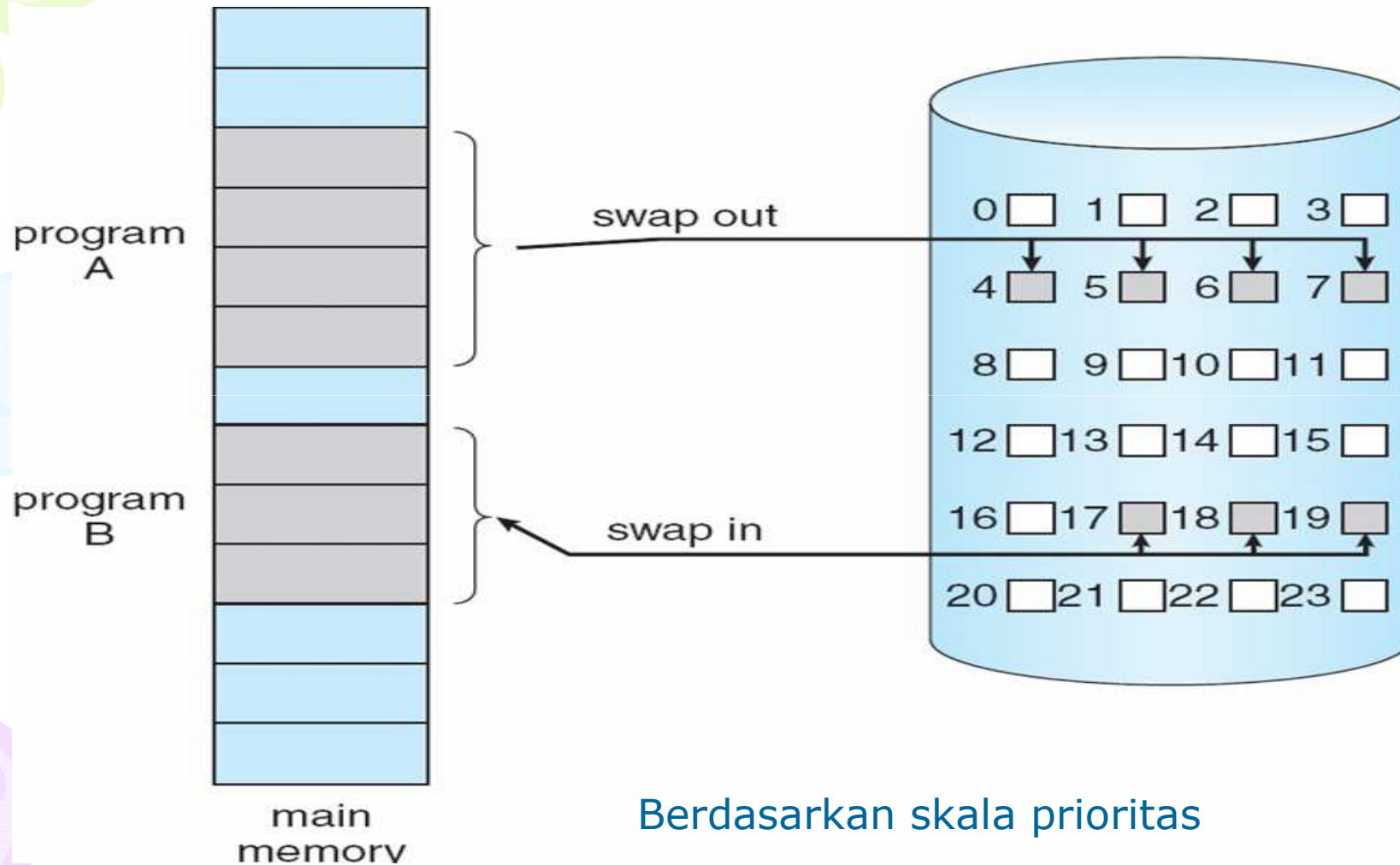




# Demand Paging

- Tidak semua program harus **diload** semua ke memory fisik
  - Hanya **page** yg diperlukan saja yang diload ke memory
- Permintaan pemberian **page** menggunakan teknik **swapping**.
- Page hanya akan di-swap ke memori utama jika **benar-benar** diperlukan.
- Program swapper yg digunakan:
  - **Lazy swapper** – tidak pernah swap page kedalam memory sampai page benar-benar diperlukan

# Transfer of a Paged Memory to Contiguous Disk Space



Berdasarkan skala prioritas



# Demand Paging

- **Demand Paging** hanya akan men-swap in dan out page yg dibutuhkan saja, tidak semuanya!
- Jadi, jika page dibutuhkan  $\Rightarrow$  referensikan, tapi blm tentu semua di-load ke memory fisik
  - invalid reference  $\Rightarrow$  abort
  - Just not-in-memory  $\Rightarrow$  bring to memory
- Butuh dukungan perangkat keras, yaitu:
  - Page-table: “valid-invalid bit”
    - Valid (“1”)  $\rightarrow$  pages berada di memori **fisik semuanya**, atau **sebagian pages** ada di memory fisik, tapi tidak semuanya, sebagian masih berada di disk.
    - Invalid (“0”)  $\rightarrow$  pages **tidak ada** di memory fisik
  - Memori sekunder, untuk menyimpan proses yang belum berada di dalam memori fisik.
- Jika proses mengakses lokasi page yg valid, maka proses akan berjalan normal.
- Jika mengakses yg invalid, maka perangkat keras akan menjebaknya (**trap**) ke Sistem Operasi (**page fault**).

# Valid-Invalid Bit

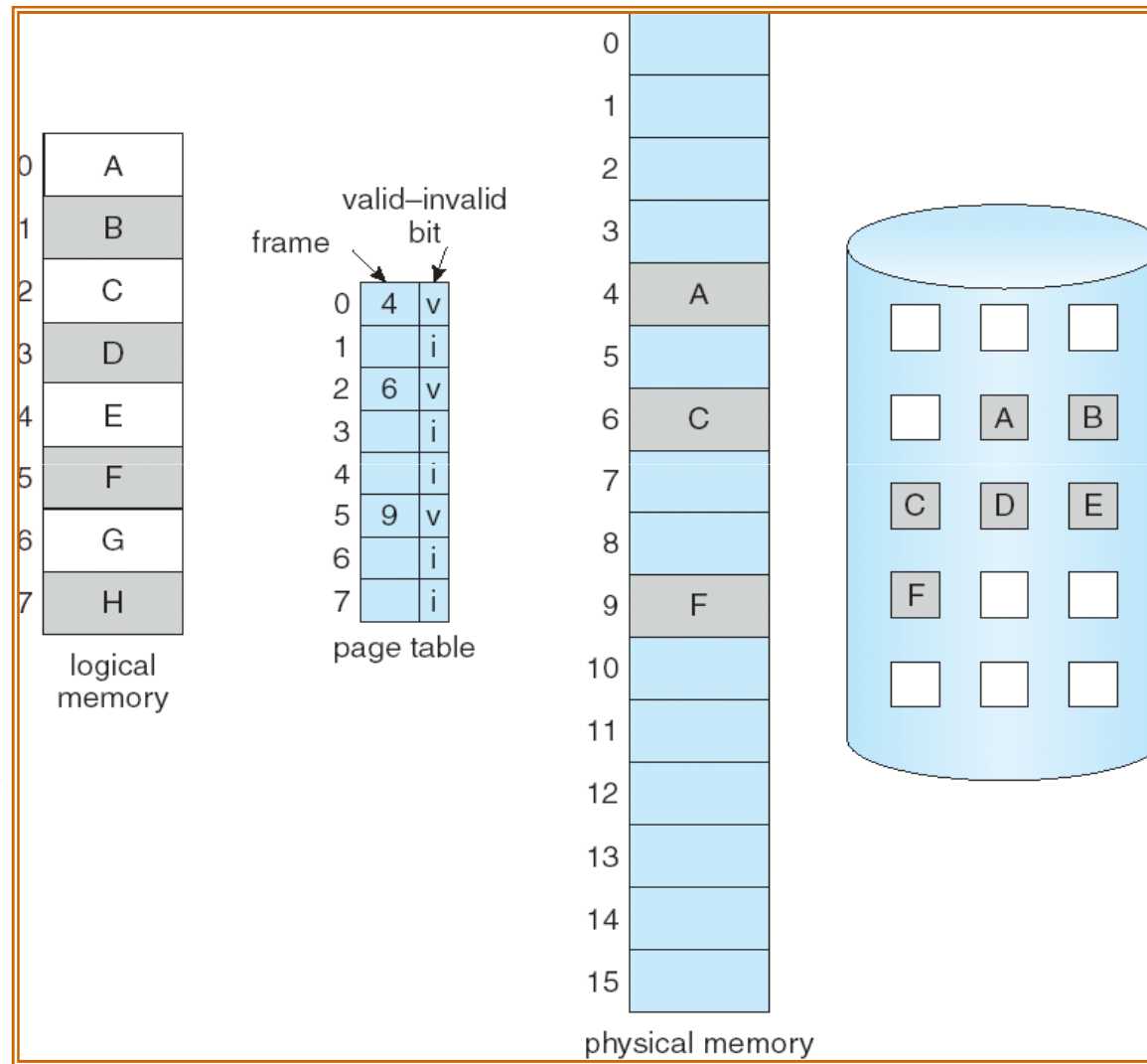
- With each page table entry a valid-invalid bit is associated (**v**  $\Rightarrow$  in-memory, **i**  $\Rightarrow$  not-in-memory)
- Initially valid-invalid bit is set to **i** on all entries
- Example of a page table snapshot:

Frame #	valid-invalid bit
	<b>v</b>
	<b>v</b>
	<b>v</b>
	<b>v</b>
	<b>i</b>
....	
	<b>i</b>
	<b>i</b>

page table

- During address translation, if valid-invalid bit in page table entry is **i**  $\Rightarrow$  page fault

# Page Table When Some Pages Are Not in Main Memory, but in disk





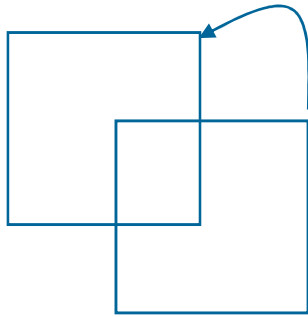
# Page Fault

- Jika ada referensi ke sebuah page, ternyata paganya tidak ada (**invalid**), maka akan ditrap oleh OS, dan menghasilkan: **page fault**
- Untuk menangani page fault menggunakan prosedur berikut:
  - Memeriksa **tabel internal** (biasanya ada dlm PCB) untuk menentukan valid atau invalid
  - Jika **invalid**, proses di **suspend**, jika valid, tapi page belum dibawa ke memory fisik, maka kita bawa page ke memory fisik.
  - Cari sebuah frame bebas (**free frame**).
  - Jadwalkan **operasi sebuah disk** untuk membaca page tersebut ke frame yang baru dialokasikan.
  - Saat pembacaan selesai, **ubah** validation bit menjadi "1" yang berarti page telah ada di memory.
  - **Restart** instruksi program yg ditrap td dari awal sehingga bisa berjalan dgn baik.

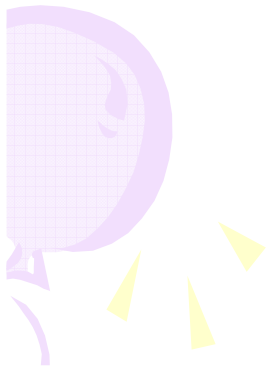


# Page Fault (Cont.)

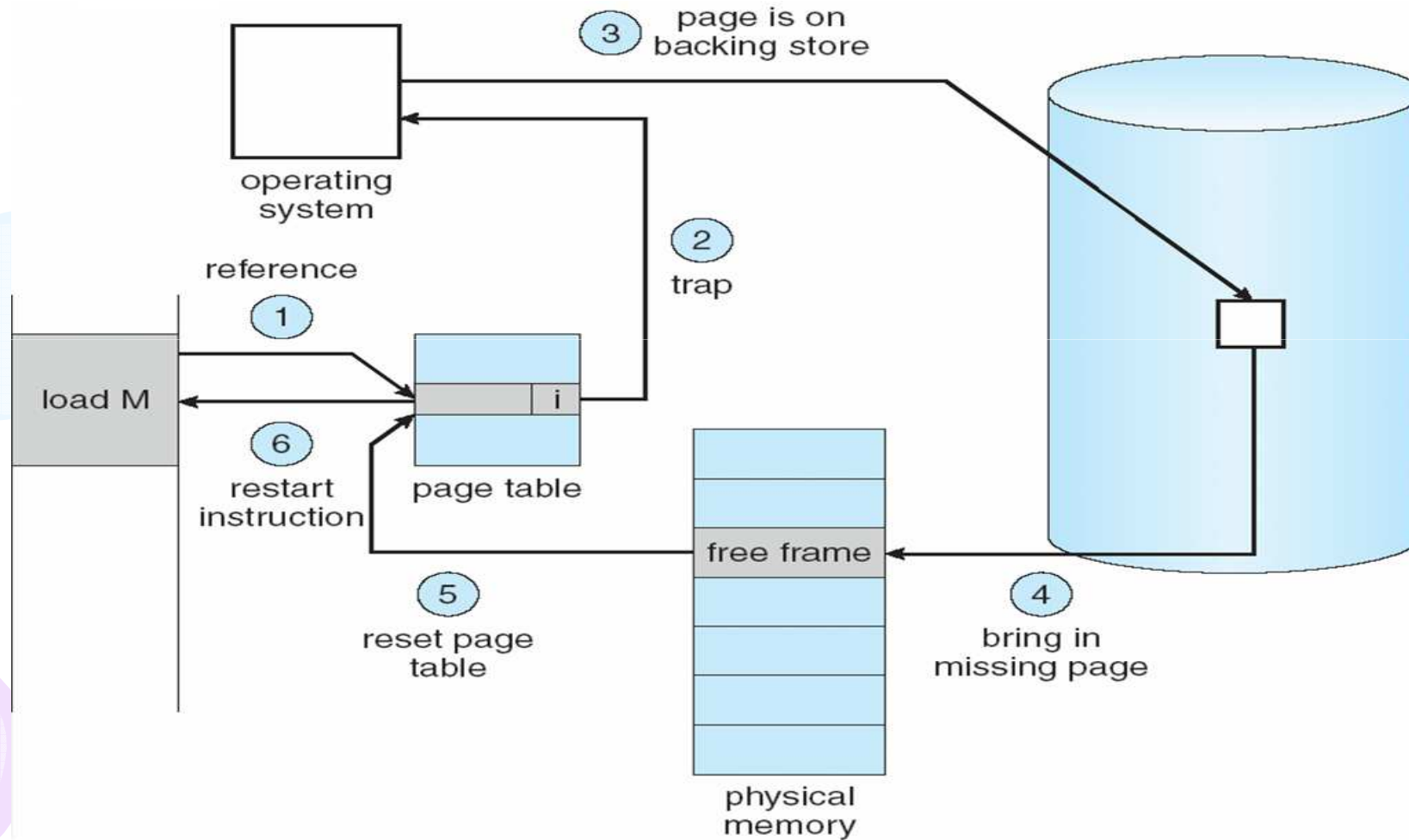
- **Restart instruction**
  - block move



- 
- auto increment/decrement location




# Steps in Handling a Page Fault







# Yg terjadi saat Page Fault

- Ditrap oleh Sistem Operasi.
  - SO menyimpan register user dan proses.
  - Tetapkan bahwa interupsi merupakan **page-fault**.
  - Periksa bahwa referensi page adalah **valid** dan kemudian tentukan lokasi page pada disk.
  - Baca disk, cari **frame kosong**.
  - Selama menunggu pencarian, alokasikan CPU ke proses lain dengan menggunakan penjadwalan CPU.
  - Jika pencarian selesai, terjadi interupsi dari disk.
- 

# Yg terjadi saat Page Fault (2)

- SO menyimpan juga register dan status proses untuk pengguna/proses yang lain.
- Tentukan bahwa interupsi skrng berasal dari **disk**.
- Lakukan pengubahan page table bahwa page telah berada di memory.
- Tunggu CPU selesai dari proses yang lain.
- Kembalikan register user, status proses, page table, dan **resume** instruksi proses yg td interupsi.




# Page fault

- Tidak semua langkah diperlukan pada tiap kasus, ada 3 komponen utama yg pasti terjadi:
  - Melayani interrupt dari page fault
  - Baca dan load page dari disk ke memory
  - Restart proses
- Pada sistem demand paging, sebisa mungkin kita jaga agar tingkat **page-fault nya rendah**.



# VM untuk Process Creation

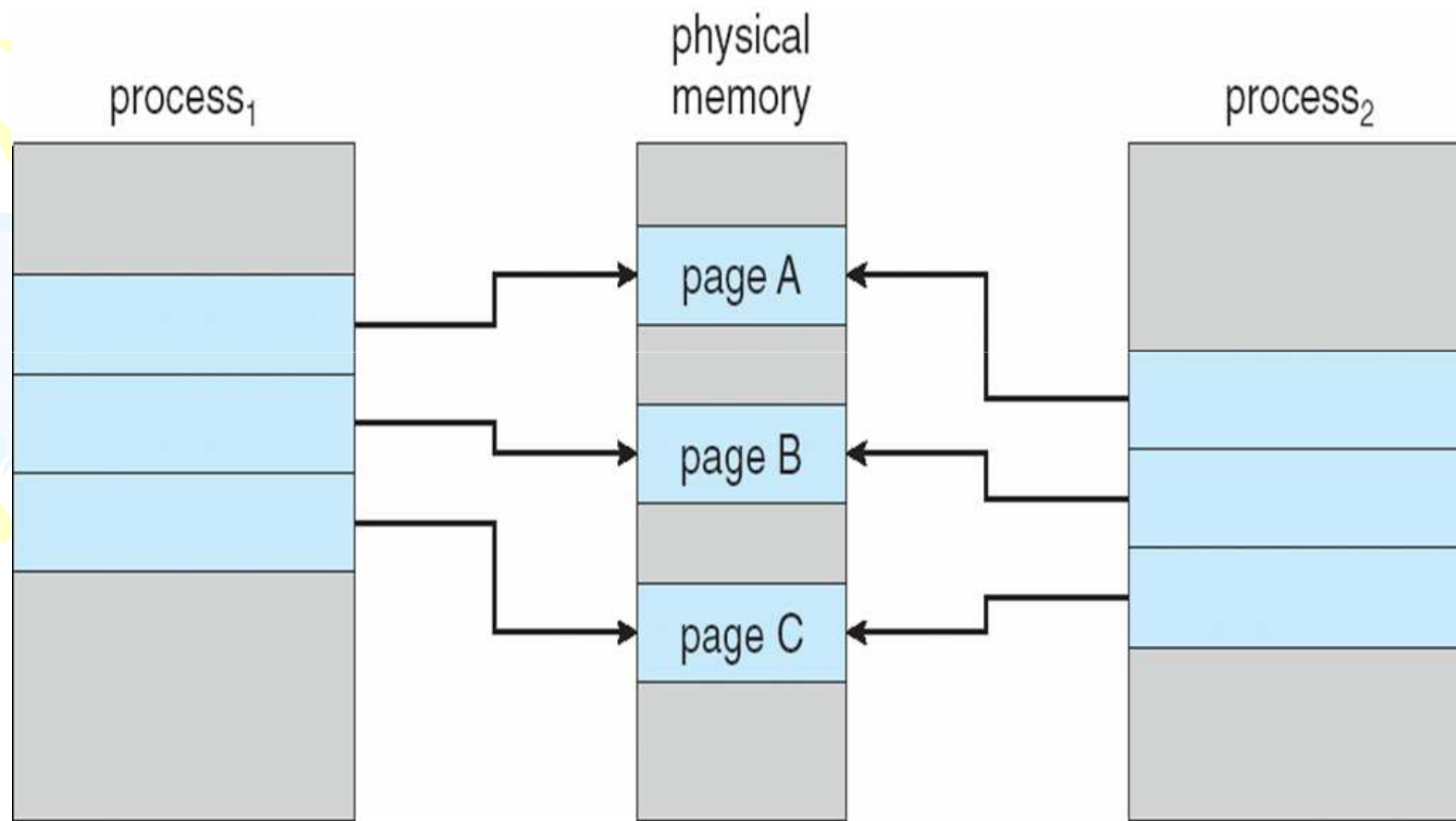
- Karena diperlukan untuk menggandakan proses (**process creation**), maka harus diketahui mana page kosong yang akan dialokasikan.
- Sistem operasi biasanya menggunakan teknik “**zero-fill-on-demand**” untuk mengalokasikan /menginisialisasi page tersebut pada awalnya.



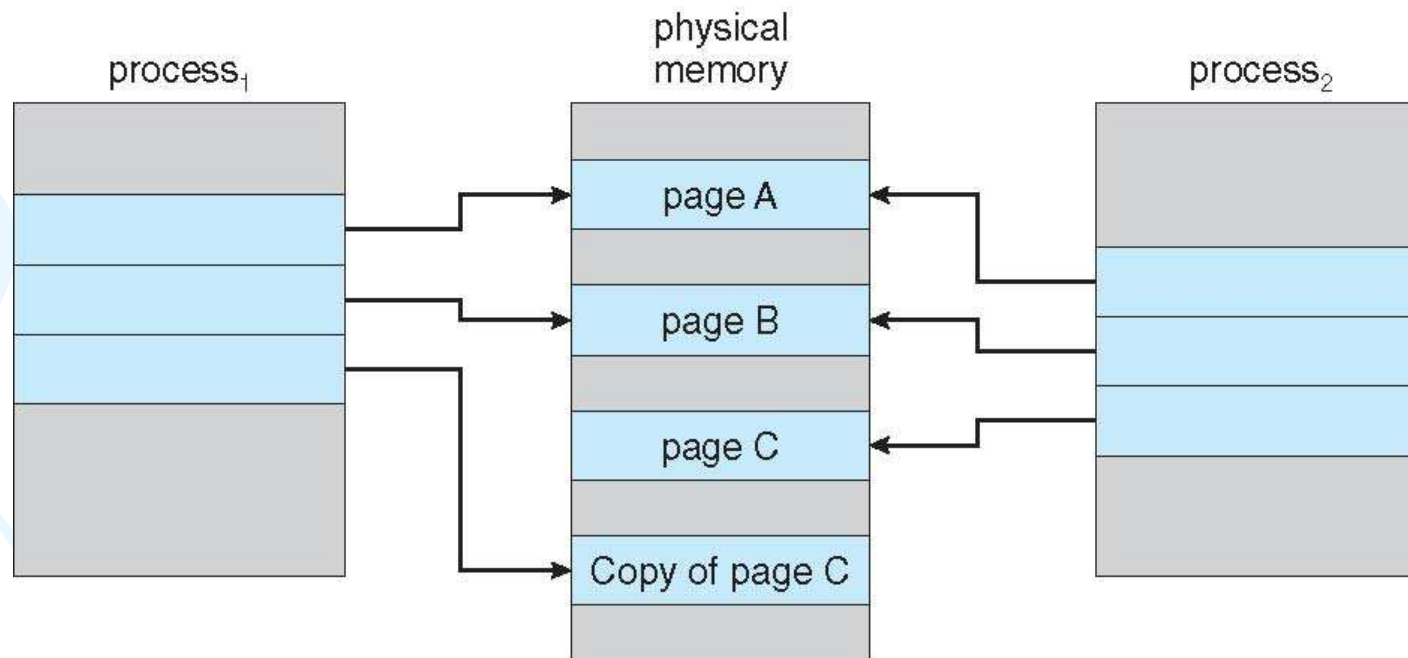
# Metode Process Creation: Copy-on-write

- Pada **copy-on-write**, sistem ini mengizinkan proses parent dan child menginisialisasikan **page yang sama** pada memori.
- Jika proses menulis pada sebuah page yang disharing, maka dibuat juga **salinan** dari page tersebut.
- Dengan menggunakan teknik **copy-on-write**, terlihat jelas bahwa hanya page yang diubah oleh proses child dan parent disalin.
  - Sedangkan semua page yang tidak diubah bisa dibagikan ke proses child dan parent.
- Teknik copy-on-write sering digunakan oleh beberapa sistem operasi saat menggandakan proses. Diantaranya adalah Windows 2000, Linux, dan Solaris 2.

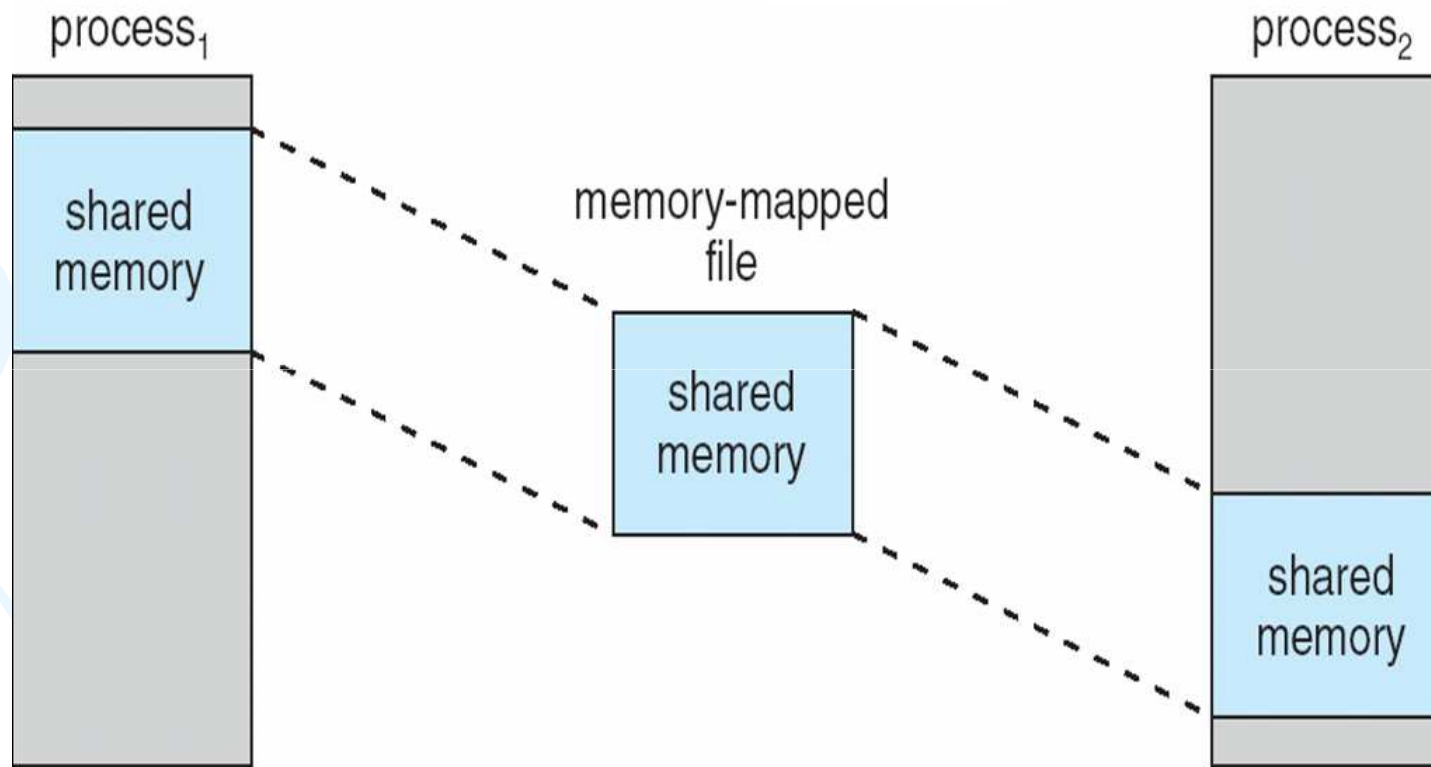
# Before Process 1 Modifies Page C



# After Process 1 Modifies Page C



# Memory-Mapped Shared Memory in Windows





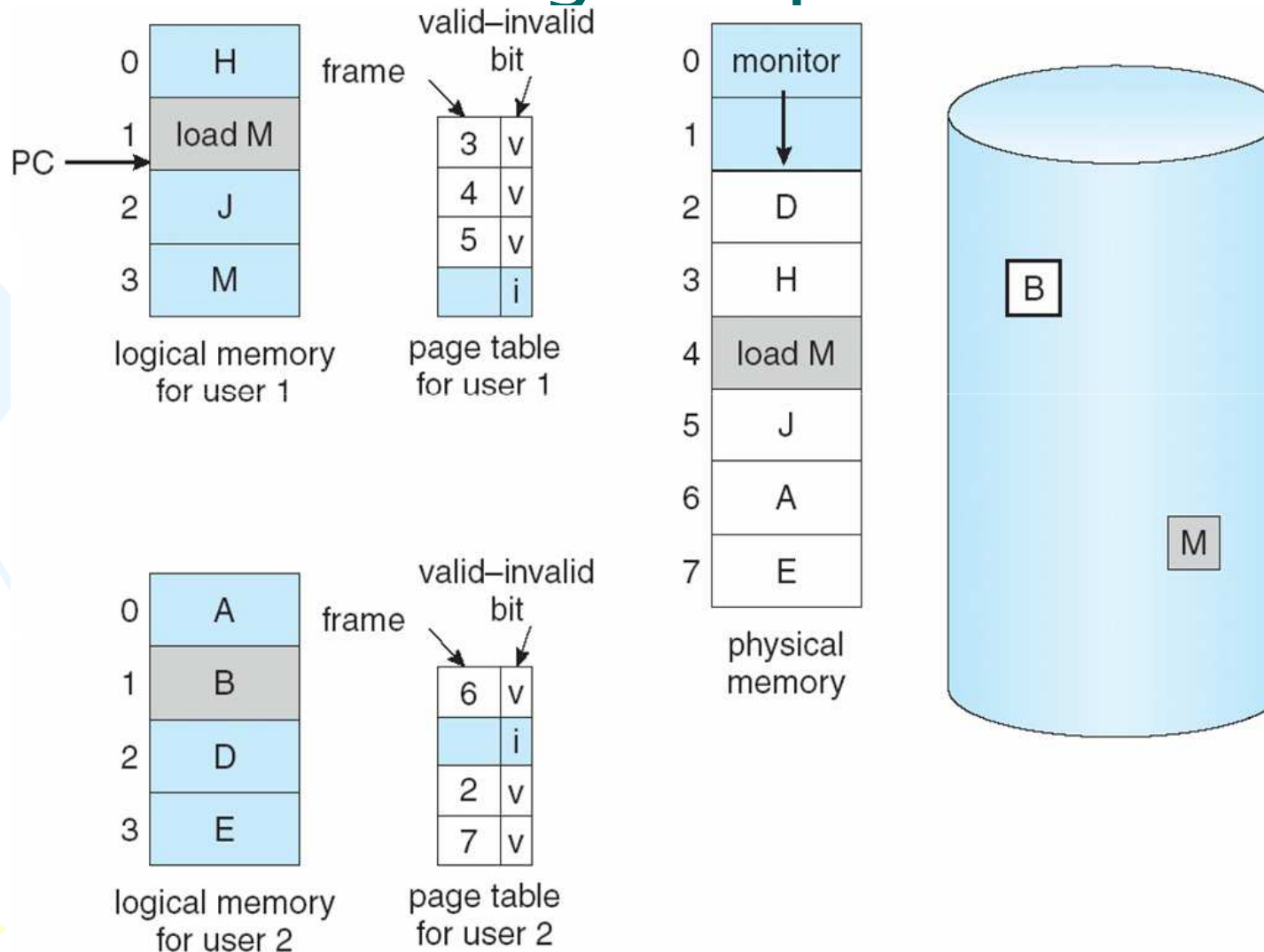
# What happens if there is no free frame?

- **Page Replacement.**

- Pendekatan :

- Jika tidak ada frame yang kosong, cari frame yang tidak sedang digunakan, lalu kosongkan dengan cara menuliskan isinya ke dalam swap space, dan mengubah semua tabel sebagai indikasi bahwa page tersebut tidak akan berada di memori lagi.
- Bagaimana algoritmanya?

# Need For Page Replacement





# Page Replacement

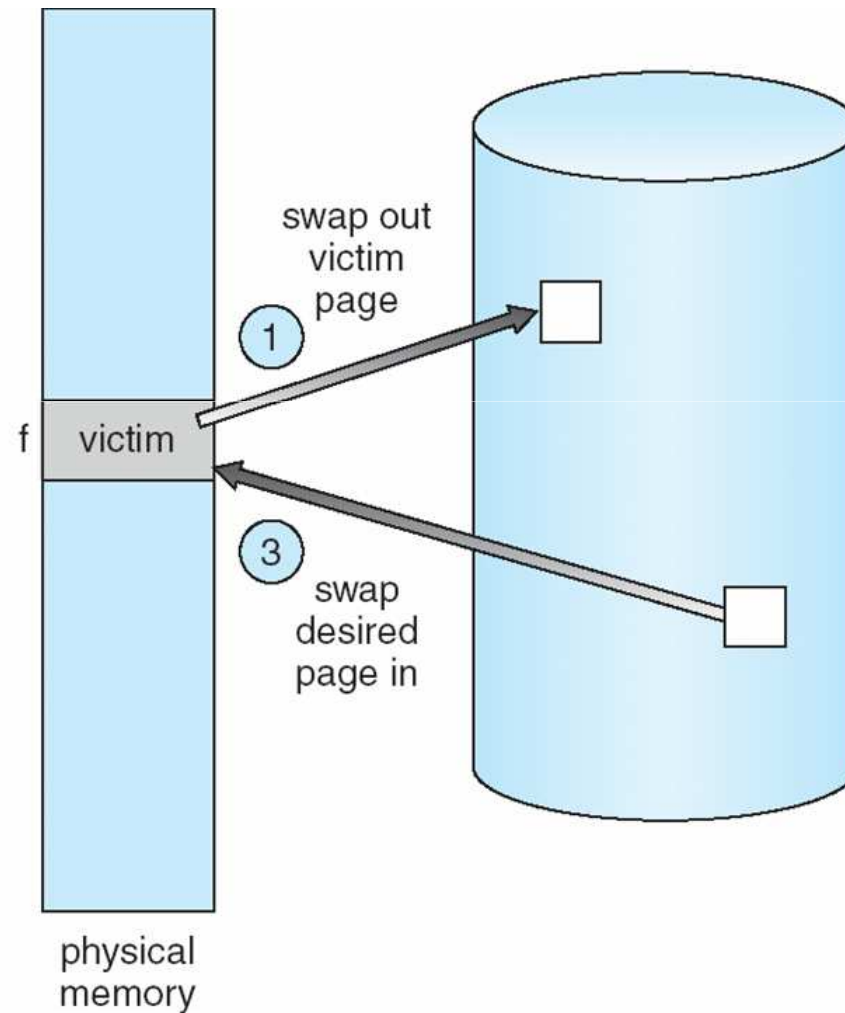
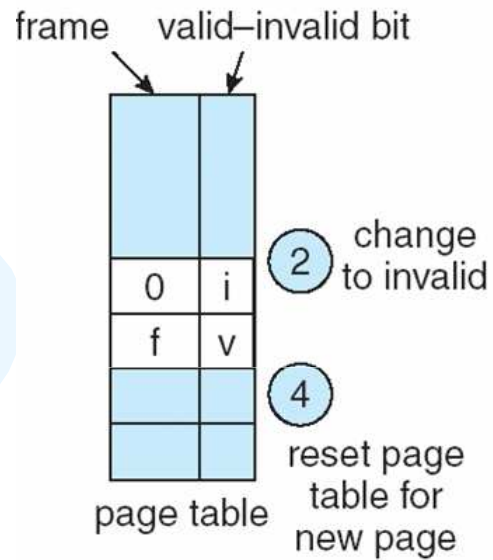
- Use **modify (dirty) bit** to reduce overhead of page transfers – only modified pages are written to disk
- Page replacement completes **separation** between logical memory and physical memory
  - large virtual memory can be provided on a smaller physical memory



# Yang dilakukan saat Page Replacement

- Mencari lokasi page yang diinginkan pada disk.
- Mencari frame yang kosong :
  - Jika ada, maka gunakan frame tersebut.
  - Jika tidak ada, maka kita bisa mengosongkan frame yang tidak sedang dipakai.
    - Gunakan **algoritma page-replacement** untuk menentukan frame yang akan dikosongkan.
  - Tulis page yang telah dipilih ke disk, ubah page-table dan frame-table menjadi **invalid**.
  - Membaca page yang diinginkan lalu di-load ke dalam frame kosong yang baru.
    - Set page-table proses itu menjadi **valid**
  - Ulangi user process dari awal (restart)

# Page Replacement



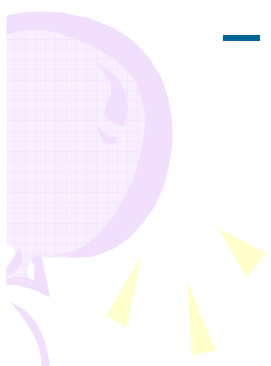


# Algoritma Page Replacement

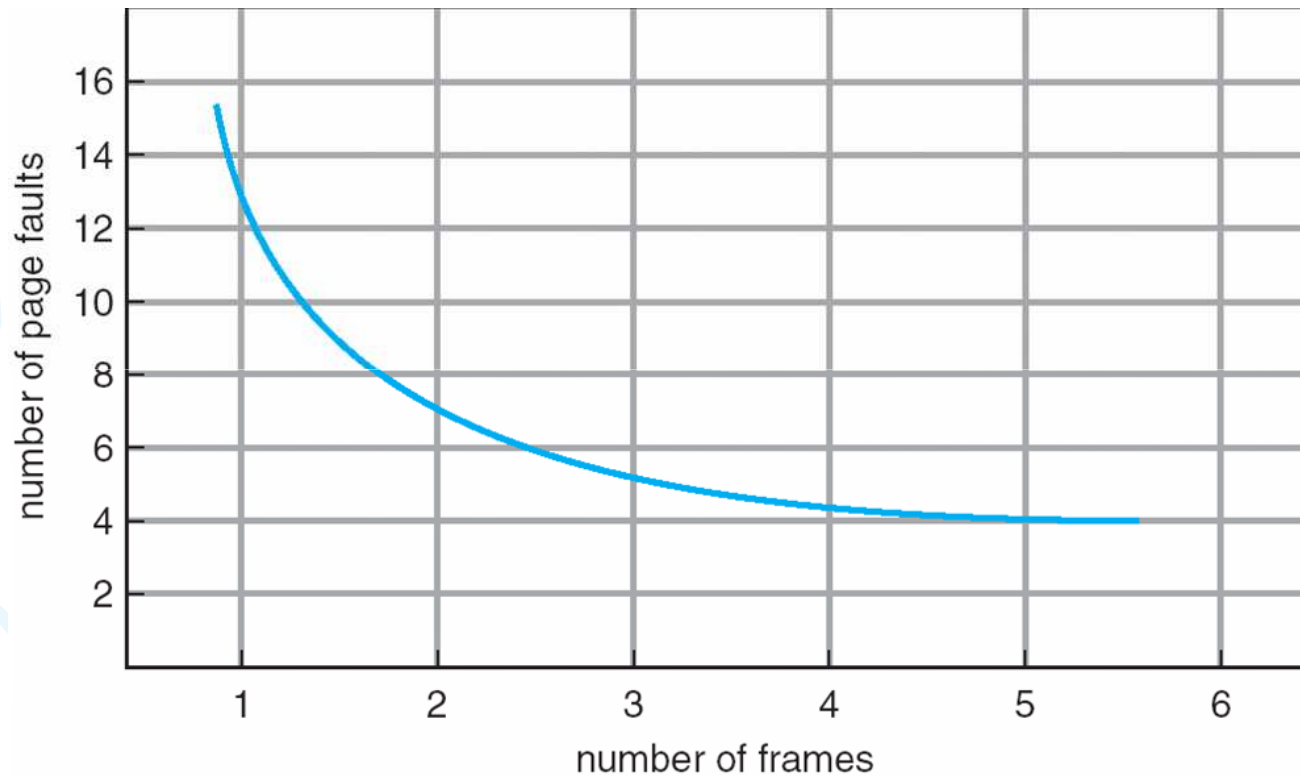
- Bertujuan untuk mendapatkan **page fault terendah.**
- Ada beberapa Algoritma Page Replacement:
  - Algoritma FIFO
  - Algoritma Optimal
  - Algoritma LRU
  - Algoritma Perkiraan LRU



# Alg. FIFO

- Page yang diganti adalah page yang **paling lama** berada di memori.
  - Mudah diimplementasikan.
  - Mudah dimengerti.
  - Bisa mengalami **Anomali Belady**.
    - Page fault rate meningkat seiring dengan meningkatnya jumlah frame.
    - Hanya terjadi pada beberapa Algoritma Page Replacement.
- 

# Graph of Page Faults Versus The Number of Frames



**Anomaly Belady:** kecepatan page fault akan bertambah jika framenya bertambah



# First-In-First-Out (FIFO) Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frames (3 pages can be in memory at a time per process)

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

9 page faults

- 4 frames

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

10 page faults

- Belady's Anomaly: more frames  $\Rightarrow$  more page faults

# FIFO Page Replacement

reference string

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

7	7	7	2																
	0	0	0																
			1	1															

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

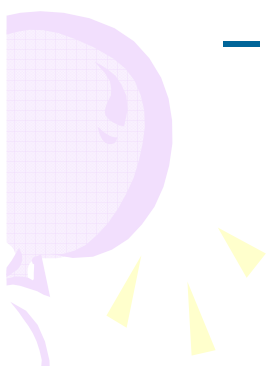
0	0																		
1	1																		
3	2																		

7	7	7																	
1	0	0																	
2	2	1																	

page frames



# Alg. Optimal

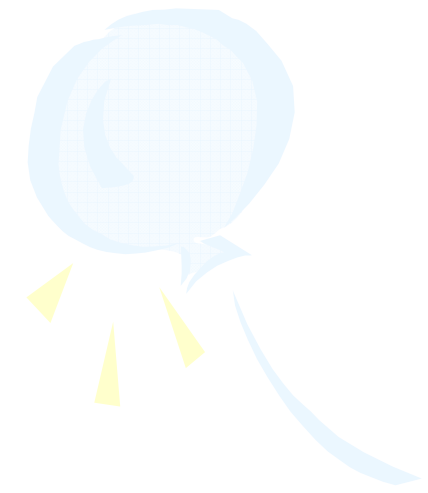
- Page yang diganti adalah page yang **tidak akan dipakai dalam jangka waktu terlama.**
  - Sulit diimplementasikan (krn prediksi sulit dilakukan)
  - Memiliki page-fault terendah.
  - Tidak akan mengalami Anomali Belady:
    - Tidak mengalami : more frames  $\Rightarrow$  more page faults
- 



# Optimal Algorithm


- 4 frames example

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



1	4
2	
3	
4	5

6 page faults

- 
- How do you know this?
  - Used for measuring how well your algorithm performs

# Optimal Page Replacement

reference string

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

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

page frames



# Alg. Least Recently Used

- Page yang diganti adalah page yang **tidak baru saja digunakan**.
- Merupakan perpaduan antara Algoritma FIFO dan Algoritma Optimal.
- Sulit diimplementasikan.
- Tidak akan mengalami Anomali Belady.

# Least Recently Used (LRU) Algorithm

- Reference string: 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

# LRU Page Replacement

reference string

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

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

page frames





# Alg. LRU

- Dapat diimplementasikan dengan 2 cara, yaitu :

- **Counter**

- Menggunakan clock yang nilainya akan ditambah 1 tiap kali melakukan reference ke suatu page.
    - Harus melakukan pencarian.

- **Stack**

- Tiap mereference ke suatu page, page tersebut dipindah dan diletakkan pada bagian paling atas stack.
    - Page yang diganti adalah page yang berada di stack paling bawah.
    - Tidak perlu melakukan pencarian.



# Algoritma Counting

- Menyimpan ***counter untuk masing-masing page.***
- Prinsip ini dapat dikembangkan menjadi algoritma berikut :
- Algoritma LFU (least frequently used)
  - *page yang diganti adalah page yang paling jarang dipakai (nilai **counter terkecil**).*
- Algoritma MFU (most frequently used)
  - *page yang diganti adalah page yang paling sering dipakai (nilai **counter terbesar**).*

# Use Of A Stack to Record The Most Recent Page References

reference string

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

2
1
0
7
4

stack  
before  
a

7
2
1
0
4

stack  
before  
b

↑  
a

↑  
b



# Cara Penghitungan

- 4 7 0 7 1 0 1 2 1 2 7 1 2
- Stack: 4 7 0 7 1 0 1 2 1 2
- Stack before a = 4 7 0 1 2
- Masuk lagi 7, sehingga:
  - Stack = 4 7 0 1 2 7
  - Stack = 4 0 1 2 7
- Dilanjutkan menjadi:
  - Stack = 4 0 1 2 7 1 2
  - Stack akhir: 4 0 7 1 2



# Alg. LRU-Approximation Page


- Menggunakan bit ***reference***
- Awalnya semua bit diinisialisasi 0 oleh sistem operasi.
- Setelah page di***reference, bit diubah menjadi 1*** oleh hardware
- Cara implementasi:
  - Algoritma ***Additional-Reference-Bits.***
  - Algoritma ***Second-Chance.***

# Additional-Reference-Bits

- Setiap page memiliki 8 bit byte sebagai penanda.
- Pada awalnya 8 bit ini diinisialisasi 0 (contoh : 00000000)
- Setiap selang beberapa waktu, ***timer melakukan interupsi*** kepada sistem operasi, kemudian sistem operasi menggeser 1 bit ke kanan.
- *Page yang diganti adalah page yang memiliki **nilai terkecil.***



# ***Second-Chance***

- Dasar algoritma ini adalah Algoritma FIFO.
  - Algoritma ini juga menggunakan ***circular queue.***
  - Apabila nilai bit ***reference-nya 0,*** ***page dapat diganti.***
  - Apabila nilai bit ***reference-nya 1,*** ***page tidak diganti tetapi*** bit ***reference diubah menjadi 0*** dan dilakukan pencarian kembali.
- 



# Alokasi Frame

- Alokasi frame berhubungan dengan mekanisme alokasi pada sejumlah memori fisik yang bebas untuk berbagai proses.
- **Fixed Allocation**
  - Proses dengan prioritas tinggi ataupun rendah diperlakukan sama.
    - **Equal Allocation**: semua sama rata
    - **Proportional Allocation**: sesuai kebutuhan
- **Alokasi prioritas**
  - Perbandingan frame-nya tidak tergantung pada ukuran relatif dari proses tetapi tergantung pada **prioritas proses**.



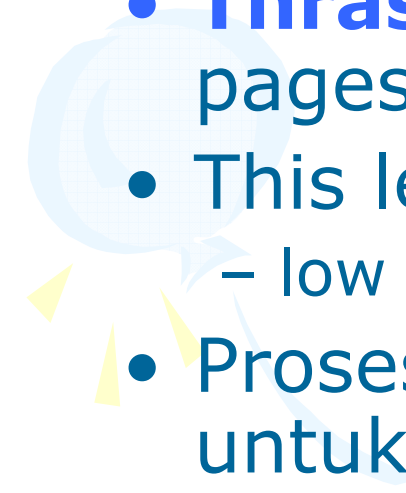
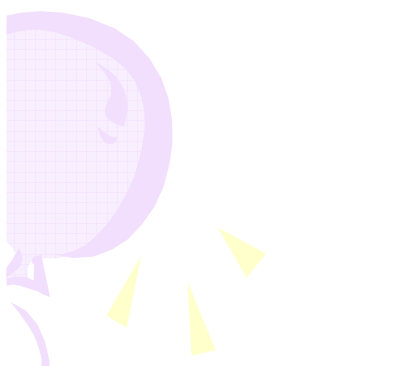


# Jenis Page Replacement

- **Global replacement** memungkinkan suatu proses untuk menyeleksi sendiri suatu frame yang akan digantikan dari sejumlah frame yang ada, meskipun frame tersebut sedang dialokasikan ke proses yang lain.
- **Local replacement**, jumlah frame yang dialokasikan untuk proses sudah ditentukan dari awal (fixed).
  - Setiap proses dapat memilih dari frame-frame yang hanya dialokasikan khusus untuknya.



# Thrashing

- If a process does not have “enough” pages, the page-fault rate is **very high**.
  - **Thrashing**  $\equiv$  a process is **busy** swapping pages in and out
  - This leads to:
    - low CPU utilization
  - Proses menghabiskan waktu lebih banyak untuk paging daripada eksekusi.
- 
- 



# NEXT

- File-System dan Security