Manajemen Memori

MANAJEMEN MEMORI

- ❖ Konsep Managemen Memori
- ❖ *Swap* dan Alokasi Memori
- ❖ Konsep *Paging*
- ❖ Struktur *Paging*
- ❖ Konsep Segmentasi
- ❖ Pengantar Memori Virtual; *Demang Paging*
- ❖ Aspek *Demand Paging*; Pembentukan Proses
- ❖ Konsep *Page Replacement*
- ❖ Algoritma *Page Replacement*
- ❖ Strategi Alokasi *Frame*
- ❖ Aspek-aspek lain dari Memori Virtual

Latar Belakang

- ❖ Memori adalah pusat kegiatan pada sebuah komputer, karena setiap proses yang akan dijalankan, harus melalui memori terlebih dahulu.
- ❖ Sistem Operasi bertugas untuk mengatur peletakan banyak proses pada suatu memori.
- ❖ Memori harus digunakan dengan baik, sehingga dapat memuat banyak proses dalam suatu waktu.

Pemberian Alamat

- Sebelum masuk ke memori, suatu proses harus menunggu. Hal ini disebut ***Input Queue*.**

Pemberian Alamat (2)

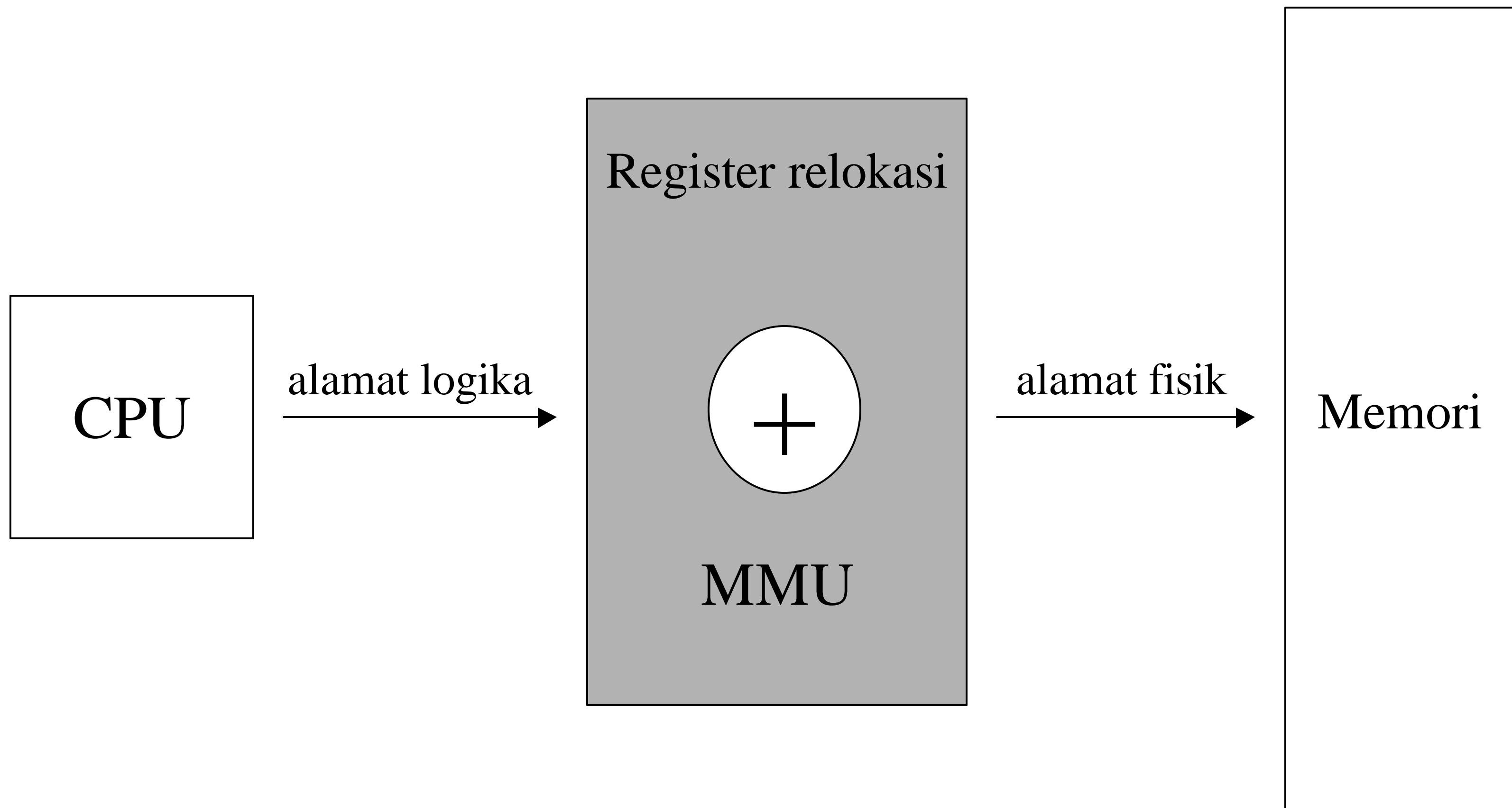
- Penjilidan alamat dapat terjadi pada 3 saat, yaitu :

Compile Time	: pada saat proses di-*compile,* menggunakan kode absolut.
Load Time	: pada saat proses dipanggil, menggunakan kode yang direlokasi.
Execution Time	: pada saat proses dijalankan, memerlukan perangkat keras tersendiri.

Ruang Alamat Logika & Fisik

- Alamat Logika adalah alamat yg dibentuk di CPU, disebut juga alamat virtual.
- Alamat fisik adalah alamat yang terlihat oleh memori.
- Untuk mengubah dari alamat logika ke alamat fisik diperlukan suatu perangkat keras yang bernama ***MMU (Memory Management Unit)*.**
- Pengubahan dari alamat logika ke alamat fisik adalah pusat dari manajemen memori.

MMU (*Memory Management Unit*)



Pemanggilan Dinamis

- Memanggil ***routine*** yang diperlukan untuk menjalankan suatu proses.
- ***Routine*** yang tidak diperlukan, tidak akan dipanggil.
- Tidak memerlukan bantuan sistem operasi.

Penghubungan Dinamis dan Kumpulan Data Bersama

- Menghubungkan semua ***routine*** yang ada di kumpulan data.
- Tidak membuang-buang tempat di disk dan memori.
- Kumpulan data yang ada dapat digunakan bersama-sama.
- Membutuhkan bantuan sistem operasi.

Penghubungan Statis

- Menghubungkan seluruh *routine* yang ada ke dalam suatu ruang alamat.

- Setiap program memiliki salinan dari seluruh kumpulan data.

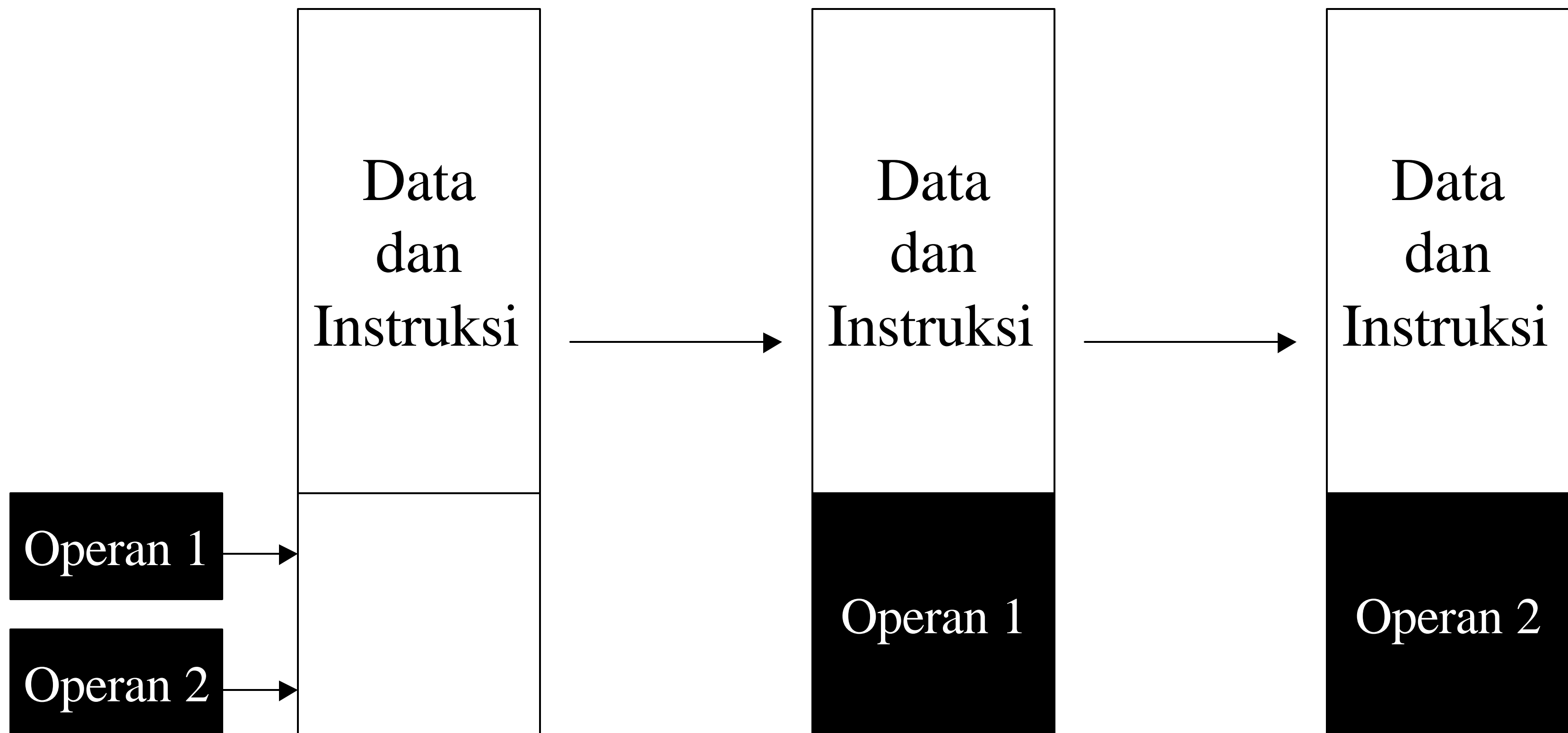
Overlays (1)

- ❖ Untuk memasukkan suatu proses yang membutuhkan memori lebih besar dari yang tersedia.
- ❖ Data dan instruksi yang diperlukan dimasukkan langsung ke memori.
- ❖ *Routine*-nya dimasukkan ke memori secara bergantian.
- ❖ Memerlukan algoritma tambahan untuk melakukan *overlays*.

Overlays (2)

- Tidak memerlukan bantuan dari sistem operasi.
- Sangat sulit untuk dilakukan.
- Dapat dilakukan di komputer mikro.

Perakit dengan 2 operan



Swapping

- Sebuah proses harus berada di dalam memori untuk dapat dijalankan.

- Sebuah proses dapat di-*swap* sementara keluar memori ke sebuah **penyimpanan cadangan** untuk kemudian dikembalikan lagi ke memori.

- ***Roll out, roll in*** adalah penjadualan *swapping* berbasis pada prioritas (proses berprioritas rendah di-*swap* keluar memori agar proses berprioritas tinggi dapat masuk dan dijalankan di memori.

Pengalokasian Memori (1)

- Salah satu tanggung jawab dari Sistem Operasi adalah mengontrol akses ke sumberdaya sistem. Salah satunya adalah **memori.**
- ***Contiguous Memory Allocation****:* alamat memori diberikan kepada proses secara berurutan dari kecil ke besar.
- **Keuntungan *Contiguous* daripada *Non-contiguous*:** sederhana, cepat, mendukung proteksi memori.
- **Kerugian *Contiguous* daripada *Non-contiguous*:** jika tidak semua proses dialokasikan di waktu yang sama, akan menjadi sangat tidak efektif dan mempercepat habisnya memori.

Pengalokasian Memori (2)

- Ada **2** tipe *contiguous memory* allocation: **partisi tunggal** dan **partisi banyak.**
  - →**Partisi tunggal** adalah alamat pertama yang dialokasikan untuk proses adalah yang berikutnya dari alamat yang dialokasikan untuk proses sebelumnya.
  - →**Partisi banyak** adalah dimana Sistem Operasi menyimpan informasi tentang semua bagian memori yang tersedia untuk digunakan (disebut ***hole****)*.

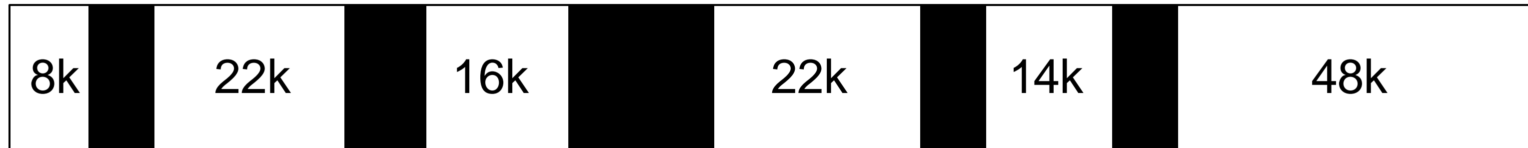
- Proses yang akan dialokasikan dimasukkan ke dalam antrian dan algoritma penjadualan digunakan untuk menentukan proses mana yang akan dialokasikan berikutnya.

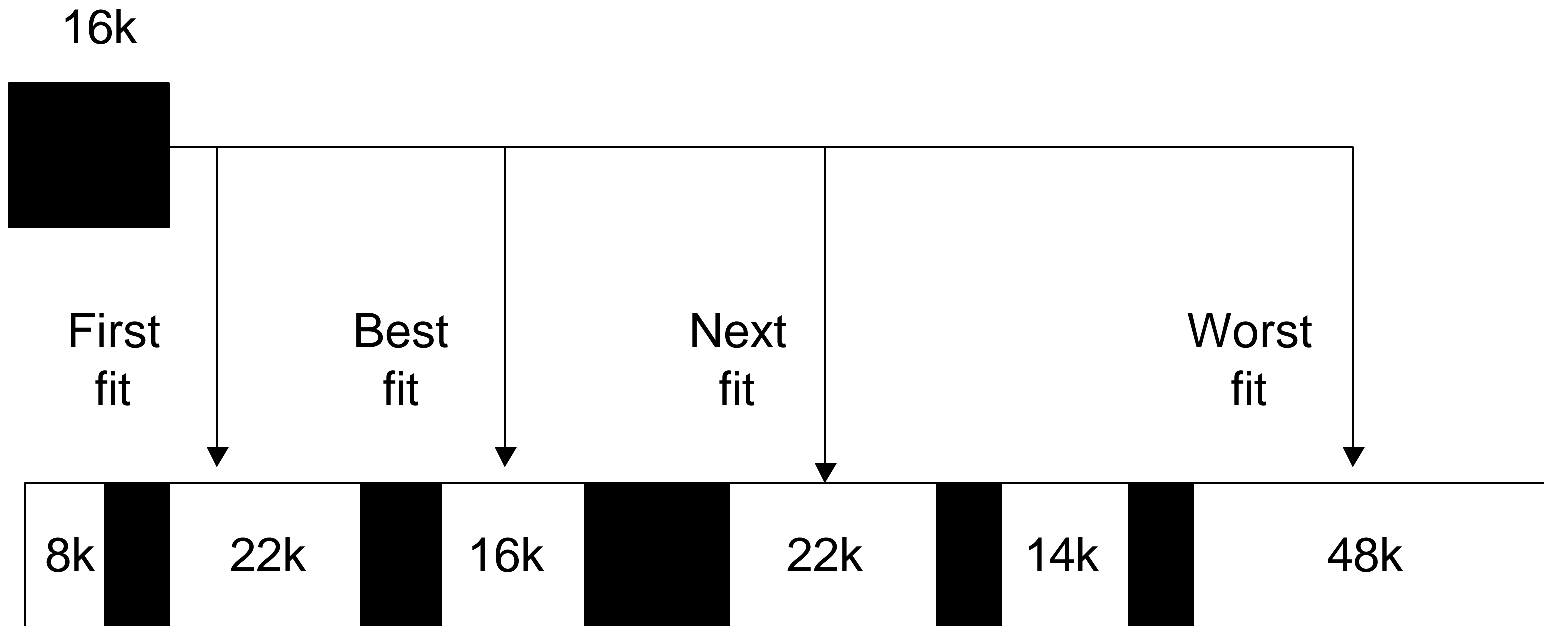
Pengalokasian Memori (3)

- ❖ Ada **2** cara pengaturan partisi pada sistem partisi banyak: **partisi tetap,** dan **partisi dinamis.**
  - → **Partisi tetap** adalah apabila memori dipartisi menjadi blok-blok yang ukurannya ditentukan dari awal. Terbagi lagi atas **partisi tetap berukuran sama,** dan **partisi tetap berukuran berbeda.**
  - → **Partisi dinamis** adalah memori dipartisi menjadi bagian-bagian dengan jumlah dan besar yang tidak tentu.

Algoritma Pengalokasian Memori dengan Partisi Dinamis

- ***First fit*** : Mengalokasikan *hole* pertama yang besarnya mencukupi. Pencarian dimulai dari awal.
- ***Best fit*** : Mengalokasikan *hole* terkecil yang besarnya mencukupi.
- ***Next fit*** : Mengalokasikan *hole* pertama yang besarnya mencukupi. Pencarian dimulai dari akhir pencarian sebelumnya.
- ***Worst fit*** : Mengalokasikan *hole* terbesar yang tersedia.


8k
22k
16k
22k
14k
48k
Blok terakhir yang dialokasikan


16k
First fit
Best fit
Next fit
Worst fit
8k
22k
16k
22k
14k
48k

Fragmentasi (1)

- **Fragmentasi** adalah munculnya *hole-hole* yang tidak cukup besar untuk menampung permintaan dari proses.

- Apabila terdapat dalam bentuk banyak hole yang berukuran kecil, disebut **Fragmentasi Eksternal.** Sedangkan apabila terdapat di dalam blok memori yang sudah teralokasi, disebut **Fragmentasi Internal.**

- Mengatasi Fragmentasi Ekstern: ***compactation,*** yaitu mengatur kembali isi memori agar memori yang kosong diletakkan bersama di suatu bagian yang besar.

- ***Compactation*** hanya dapat dilakukan apabila relokasi bersifat dinamis dan pengalamatan dilakukan pada saat proses dijalankan.

Fragmentasi (2)

- Solusi lain untuk fragmentasi ekstern adalah ***paging,*** dan ***segmentasi*.**

- Partisi tetap berukuran berbeda lebih baik dalam meminimalisasi fragmentasi intern daripada partisi tetap berukuran sama.

Proteksi Memori

- ❖ Proteksi memori dapat berarti melindungi Sistem Operasi dari proses yang sedang dijalankan oleh pengguna komputer, atau melindungi suatu proses dari proses lainnya.
- ❖ ***Swapping*** dan ***Compaction*** dapat menyebabkan suatu proses menempati lokasi memori yang berbeda selama proses tersebut dijalankan.
- ❖ Salah satu penyelesaiannya adalah dengan **relokasi.**

Paging

- Suatu metode yang memungkinkan suatu alamat memori fisis yang tersedia dapat tidak berurutan.

Konsep *Paging*

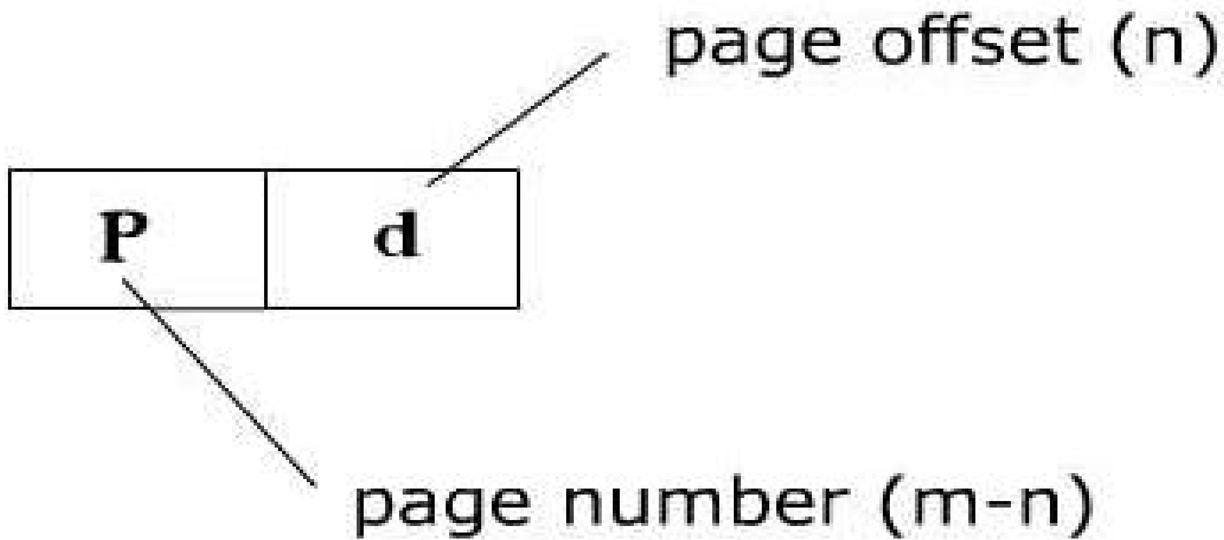
- ❖ Memori virtual dibagi menjadi blok-blok yang ukurannya tetap yang dinamakan page (ukurannya adalah pangkat 2, diantara 512 bytes dan 8192 bytes, tergantung arsitektur).

- ❖ Memori fisik dibagi juga menjadi blok2 yang ukurannya tetap yang dinamakan ***frame*.**

- ❖ Lalu kita membuat suatu page table yang akan menterjemahkan memori virtual menjadi memori fisis.

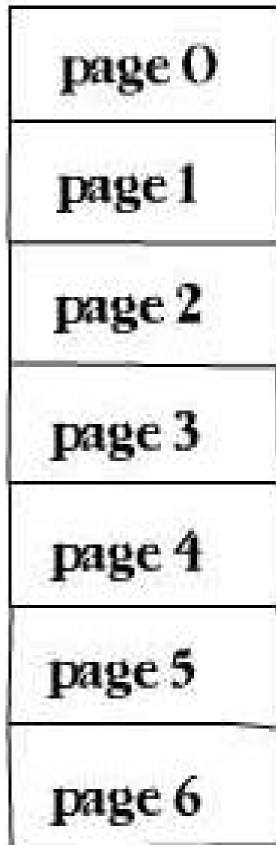
Pages

- Alamat yang dihasilkan oleh CPU(memori logis) akan dibagi menjadi 2 yaitu **nomor *page* (p)** dan ***page offset***(d):
  - → **Nomor *page*** akan menjadi indeks dari ***page table*** yang mengandung alamat dasar dari setiap alamat di memori fisis.
  - → ***page offset*** akan digabung dengan alamat dasar untuk mendefinisikan alamat fisis yang akan di kirim ke unit memori.

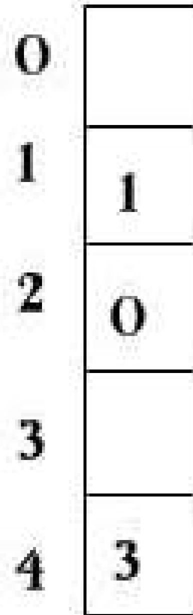

page offset (n)
P
d
page number (m-n)

Frames

- Pada alamat memori fisis akan dibagi menjadi nomor ***frame*** **(f)** yang nantinya akan dicocokan pada ***page table*.**

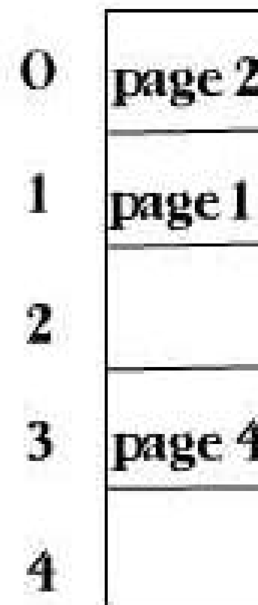


memory logical



frame number

page tabel



memory physical

Penjelasan Konsep

- Setiap alamat yang dihasilkan oleh CPU akan dicocokan nomor ***page-***nya pada ***page table*** lalu akan dicari ***frame*** mana yang sesuai dengan nomor ***page*** tersebut.

Kerugian dan Keuntungan *Paging* (1)

- Jika kita membuat ukuran dari masing-masing ***pages*** menjadi besar:
  - →Keuntungan: akses memori akan relatif lebih cepat.
  - →Kerugian: kemungkinan terjadinya fragmentasi internal yang sangat besar.

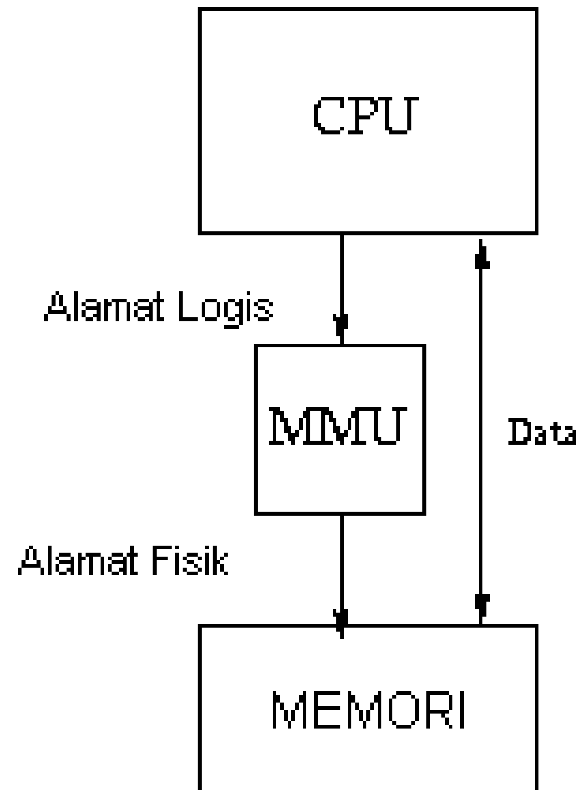
Kerugian dan Keuntungan *Paging* (2)

- ❖ Jika kita membuat ukuran dari masing-masing ***pages*** menjadi kecil:
  - → Keuntungan: akses memori akan relatif lebih lambat.
  - → Kerugian: kemungkinan terjadinya fragmentasi internal akan menjadi lebih kecil.

Struktur Paging

- ❖ Struktur MMU
- ❖ Perangkat Keras Pemberian page
- ❖ Tabel page
- ❖ Skema Tabel page Dua Tingkat
- ❖ Paging secara ***multilevel***
- ❖ Tabel page secara ***inverted***
- ❖ Berbagi *page*.

Struktur MMU



Perangkat Keras Paging

❖ Dikenal dengan ***Unit Manajemen Memori (MMU).***

❖ Jika CPU ingin mengakses memori, CPU mengirim alamat memorinya ke MMU yang akan menerjemahkannya ke alamat lain sebelum mengirim kembali ke unit memori.

❖ Alamat yang dihasilkan oleh CPU disebut **alamat logis.**

❖ Alamat yang didapat setelah melalui MMU disebut **alamat fisis.**

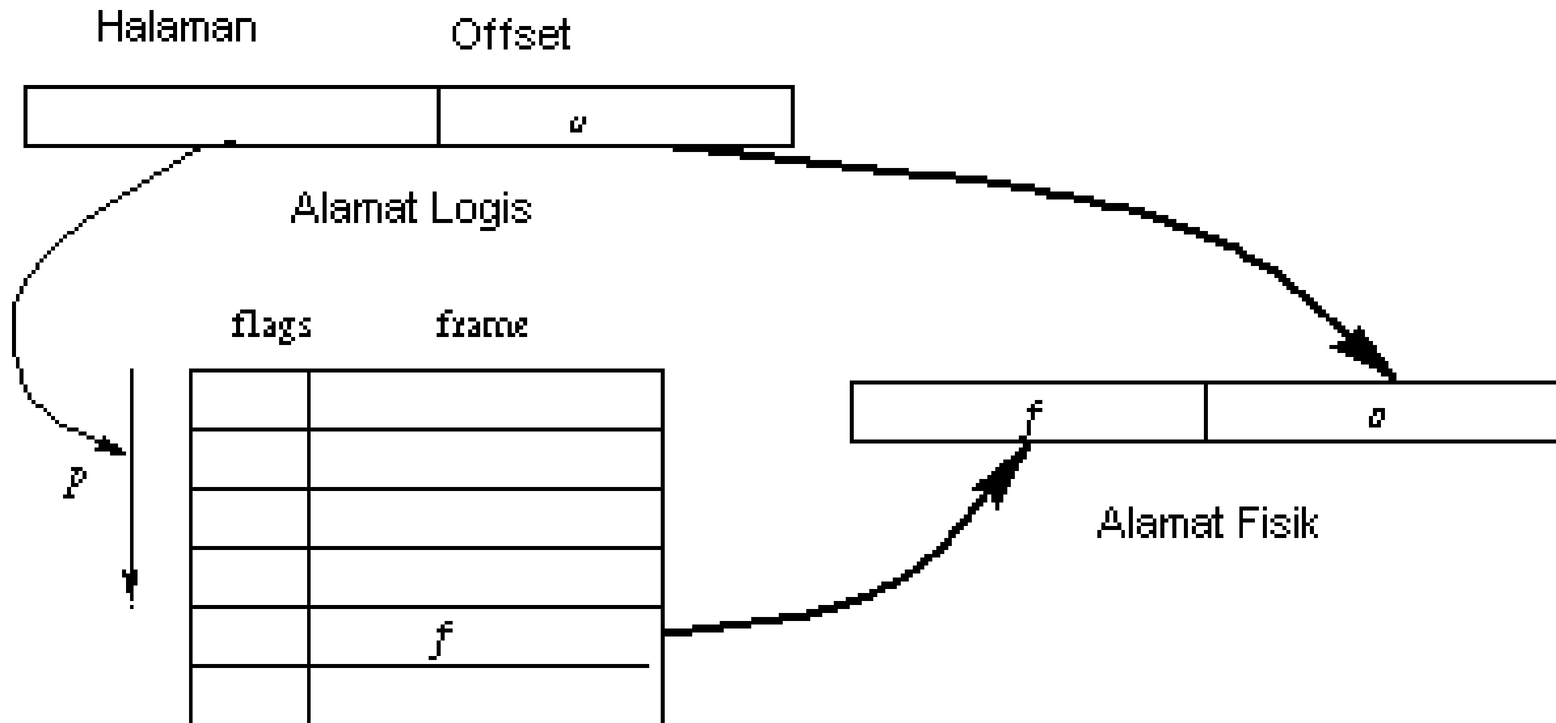
Page Table (1)

- Sebuah rangkaian array dari masukan-masukan (***entries***) yang mempunyai indeks berupa nomor page (*p*).

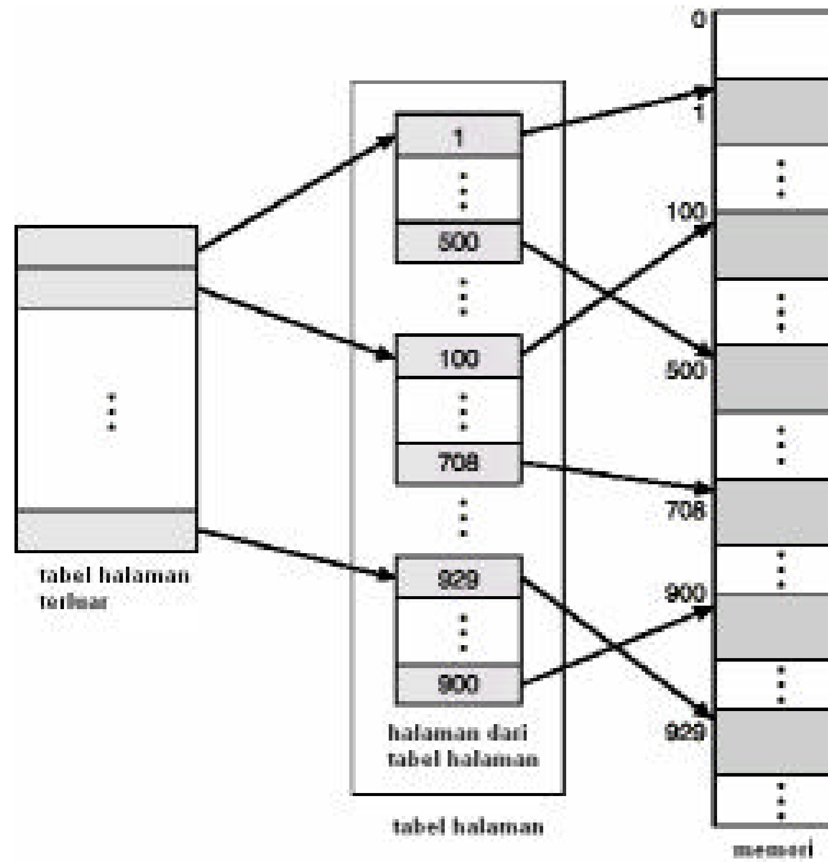
- Setiap masukan terdiri dari ***flags*** (contohnya bit ***valid***) dan nomor ***frame.***

- Alamat fisis dibentuk dengan menggabungkan nomor ***frame*** dengan ***offset.***

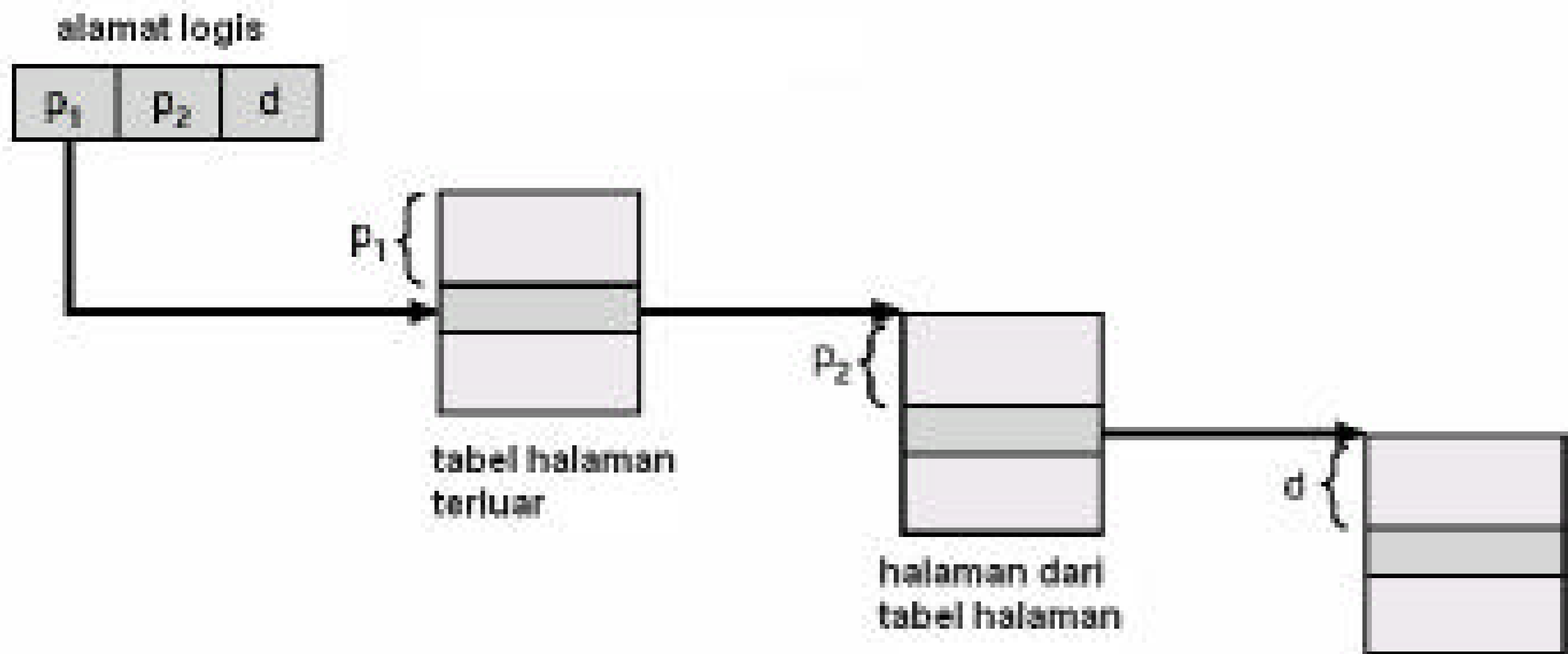
Page Table (2)



Skema *Page Table* Dua Tingkat



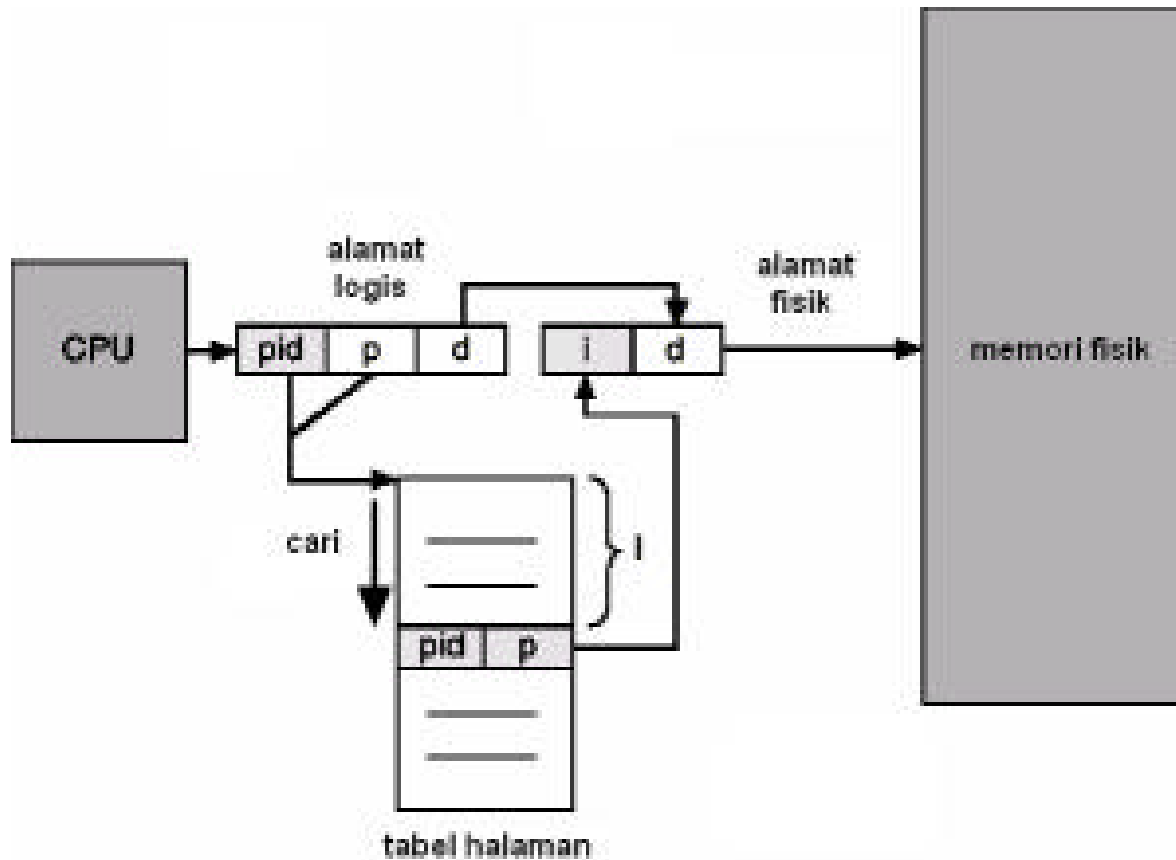
Paging secara *Multilevel*



Inverted Page Table (1)

- Satu masukan untuk setiap *page* dari memori.
- Masukan terdiri dari page di alamat logis yang disimpan di lokasi memori nyata, dengan informasi tentang proses yang dimiliki oleh page tersebut.
- Mengurangi memori yang dibutuhkan untuk menyimpan setiap tabel page, tetapi mengurangi waktu yang dibutuhkan untuk mencari tabel saat page mengalami kerusakan.
- Menggunakan ***hash table*** untuk membatasi mencari satu atau beberapa masukan tabel page.

Inverted Page Table (2)

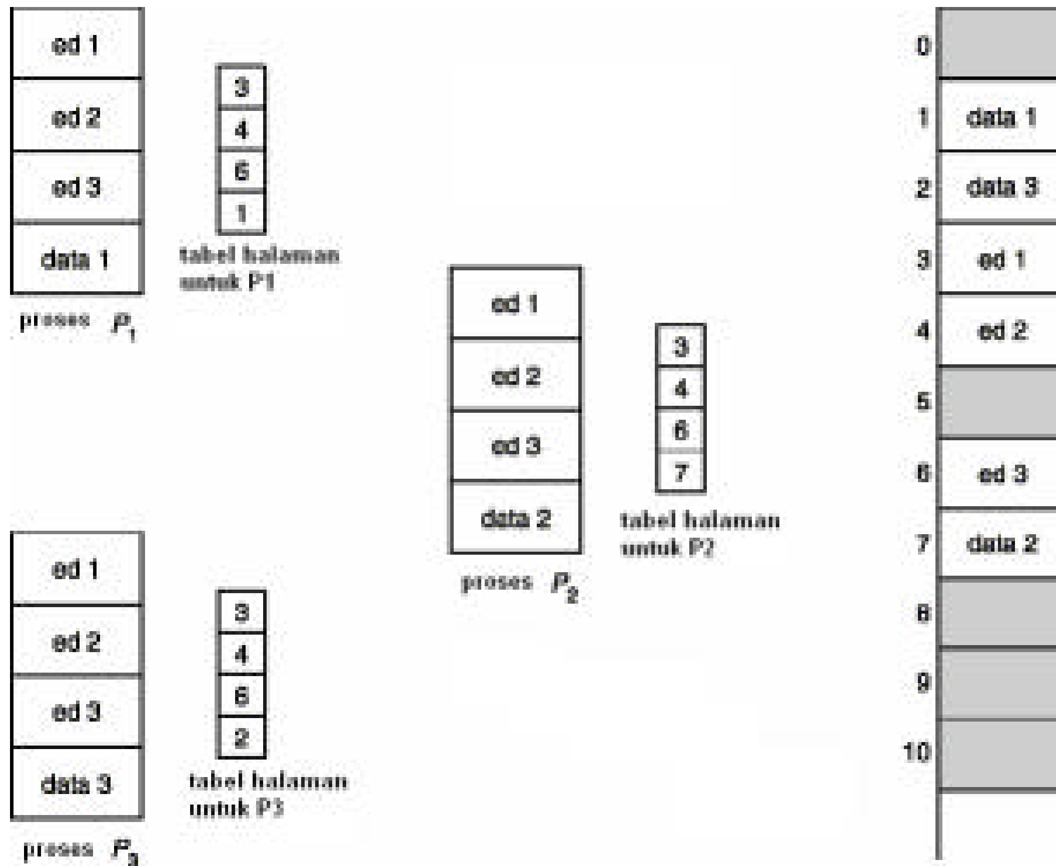


Berbagi *page* (1)

- **Berbagi Kode**
  - → Dibutuhkan suatu kode *read-only* yang dibagi antara proses.
  - →Kode yang dibagi harus berada di lokasi yang sama di alamat logis

- **Kode dan Data Privat**
  - → Setiap proses menyimpan kode dan datanya.
  - → page untuk kode dan data tertutup bisa berada dimana saja dalam ruang di alamat logis.

Berbagi *page* (2)



Segmentasi

- ❖ Salah satu cara untuk mengatur memori dengan menggunakan segmen.
- ❖ Program dibagi menjadi beberapa segmen.
- ❖ Segmen → kumpulan *logical unit*.

Arsitektur Segmentasi (1)

- ❖ Ukuran tiap segmen tidak harus sama.
- ❖ Dapat diletakan di mana saja (di ***main memory,*** setelah program dimasukkan ke memori).
- ❖ **Tabel Segmen** → menentukan lokasi segmen.
- ❖ Alamat logis-nya dua dimensi, terdiri dari : panjang segmen **(*limit*)** dan alamat awal segmen berada **(*base*).**

Arsitektur Segmentasi (2)

- Saling berbagi.
- Adanya proteksi.
- Alokasi yang dinamis.

Masalah dalam Segmentasi

- ❖ Segmen dapat membesar.
- ❖ Muncul fragmentasi luar.
- ❖ Bila ada proses yang besar.

Segmentasi dengan *paging*

❖ Kelebihan *paging*:
  → Tidak ada fragmentasi luar.
  → Alokasi-nya cepat.

❖ Kelebihan segmentasi:
  → Saling berbagi.
  → Proteksi.

Penggunaan Segmentasi

- MULTICS

- INTEL

Pengertian Memori Virtual

- ❖ Memori virtual merupakan suatu teknik yang memisahkan antara memori logis dan memori fisiknya.
- ❖ Menyembunyikan aspek-aspek fisik memori dari user.
  - → Memori adalah lokasi alamat virtual berupa byte yang tidak terbatas.
  - → Hanya beberapa bagian dari **memori virtual** yang berada di **memori logis**.

Prinsip Memori Virtual

- Konsep memori virtual yang dikemukakan Fotheringham pada tahun 1961 pada sistem komputer Atlas di Universitas Manchester, Inggris:

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

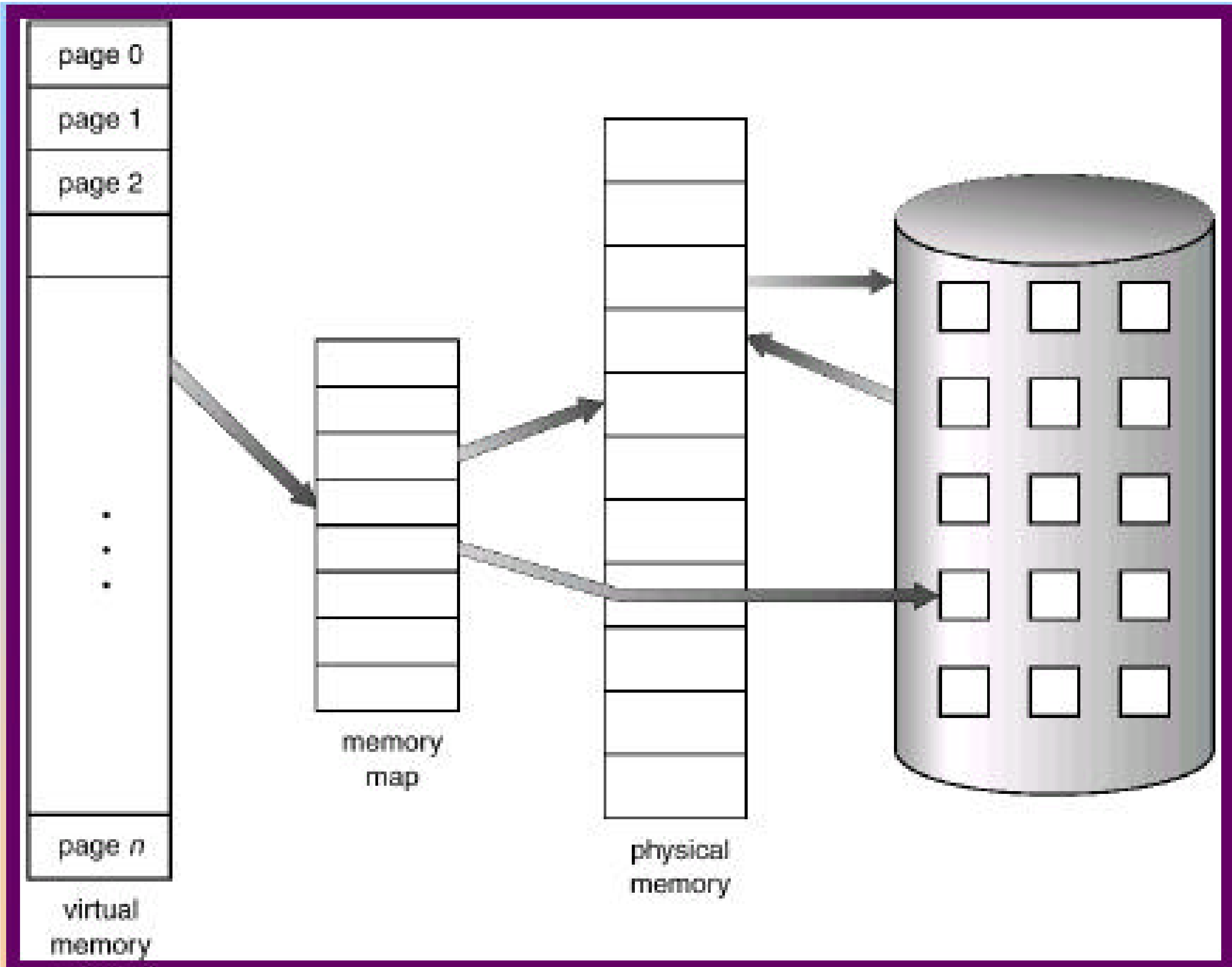
Keuntungan Memori Virtual

- ❖ Lalu lintas I/O menjadi rendah.
- ❖ Berkurangnya memori yang dibutuhkan.
- ❖ Meningkatnya respon.
- ❖ Bertambahnya jumlah ***user*** yang dapat dilayani.
- ❖ Memori virtual melebihi daya tampung dari memori utama yang tersedia.

Implementasi Memori Virtual

- ❖ Implementasi dari virtual memori: **multiprograming**

- ❖ Memori virtual dapat dilakukan dengan cara:
  - → ***Demand paging***
  - → ***Demand segmentation***



Silberschatz, Galvin and Gagne ©2002

Demand Paging **(1)**

- **Demand paging**: Permintaan pemberian *page*
- Permintaan pemberian *page* menggunakan ***swapping*.**
- ***Page*** pada permintaan pemberian *page* hanya di-*swap* ke memori jika benar-benar diperlukan.

Demand Paging (2)

- Keuntungan:
  - →Sedikit I/O yang dibutuhkan
  - →Sedikit Memory yang dibutuhkan
  - →Response yang lebih cepat
  - →Lebih banyak melayani user

Demand Paging (3)

- Butuh dukungan perangkat keras, yaitu:
  - → ***Page-table*** “**valid-invalid bit**”
    - Valid (“1”) → *pages* berada di memori.
    - Invalid (“0”) → *pages* berada di ***disk*.**
  - → **Memori sekunder**, untuk menyimpan proses yang belum berada di dalam memori.

- Jika proses mengakses lokasi yang berada di dalam memori, proses akan berjalan normal. Jika tidak, maka perangkat keras akan menjebaknya ke Sistem Operasi ***(page fault)*.**

Penanganan *page-fault*

- Untuk menangani *page* fault menggunakan prosedur berikut:
  - →Memeriksa tabel internal.
  - →Jika invalid, proses selesai, jika valid tapi proses belum dibawa ke *page*, maka kita *page* sekarang.
  - →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.
  - →Ulangi lagi instruksinya dari awal.

Apa yang terjadi pada saat *page-fault?* (1)

- ***Page-fault*** menyebabkan urutan kejadian berikut :
    1. Ditangkap oleh Sistem Operasi.
    2. Menyimpan register user dan proses.
    3. Tetapkan bahwa interupsi merupakan *page-fault*.
    4. Periksa bahwa referensi *page* adalah legal dan tentukan lokasi *page* pada disk.
    5. Kembangkan pembacaan disk ke frame kosong.
    6. Selama menunggu, alokasikan CPU ke pengguna lain dengan menggunakan penjadwalan CPU.
    7. Terjadi interupsi dari disk bahwa I/O selesai.

Apa yang terjadi pada saat *page-fault?* (2)

8. Simpan register dan status proses untuk pengguna yang lain.
9. Tentukan bahwa interupsi berasal dari disk.
10. Betulkan *page table* dan tabel yang lain bahwa *page* telah berada di memory.
11. Tunggu CPU untuk untuk dialokasikan ke proses yang tadi.
12. Kembalikan register user, status proses, *page table*, dan ***resume*** instruksi interupsi.

Apa yang terjadi pada saat *page-fault?* (3)

- Pada berbagai kasus, ada tiga komponen yang kita hadapi pada saat melayani *page-fault* :
  1. Melayani interupsi *page-fault*
  2. Membaca *page*
  3. Mengulang kembali proses

Kinerja *Demand Paging*

- Menggunakan "Effective Access Time (EAT)", dengan rumus :

  EAT = (1-*p*) x *ma* + *p* x waktu page fault

  p : kemungkinan terjadinya page fault ($0 \le p \le 1$)

  $p = 0$ → tidak ada *page-fault*

  $p = 1$ → semuanya mengalami *page-fault*

  ma : waktu pengaksesan memory (***memory access time***)

- Untuk menghitung EAT, kita harus tahu berapa banyak waktu dalam pengerjaan ***page-fault***.

Kinerja *Demand Paging* (2)

- Contoh penggunaan **Effective Access Time**
  - → Diketahui :
    - Waktu pengaksesan memory = *ma* = 100 nanoseconds
    - Waktu page fault = 20 miliseconds
  - → Maka,

$$\begin{aligned} EAT &= (1-p) \times 100 + p \text{ (20 miliseconds)} \\ &= 100 - 100p + 20.000.000p \\ &= 100 + 19.999.900p \text{ (miliseconds)} \end{aligned}$$

Kinerja *Demand Paging* (3)

- Pada sistem *demand paging*, sebisa mungkin kita jaga agar tingkat *page-fault*-nya rendah. Karena bila **Effective Access Time** meningkat, maka proses akan berjalan lebih lambat.

Pembuatan Proses (1)

- Pembuatan proses bisa dilakukan dengan 2 cara: ***copy-on-write*** dan ***memory-mapped files.***

- Pada ***copy-on-write,*** mengizinkan proses **parent** dan **child** menginisialisasikan *page* yang sama pada memori.

- Jika proses menulis pada sebuah *page* yang dibagi, maka dibuat juga salinan dari *page* tersebut.

Pembuatan Proses (2)

- 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.

Pembuatan Proses (3)

- ❖ Karena diperlukan untuk menggandakan proses, maka harus diketahui mana *page* kosong yang akan dialokasikan.
- ❖ Digunakan sebuah **pool** dari *page* kosong yang diminta.
- ❖ Sistem operasi biasanya menggunakan teknik "**zero-fill-on-demand**" untuk mengalokasikan *page* tersebut.

Pembuatan Proses (4)

- Dengan teknik ***memory-mapped-files*** membuat berkas I/O dianggap sebagai **akses memori routine** dengan memetakan satu blok ***disk*** ke sebuah *page* pada memori.

- Sebuah file awalnya dibaca menggunakan ***demand paging***. Sebagian dari ukuran page dibaca dari sistem berkas ke dalam *page* fisis. Urutan membaca dan menulis ke dalam file ditangani sebagai akses memori biasa.

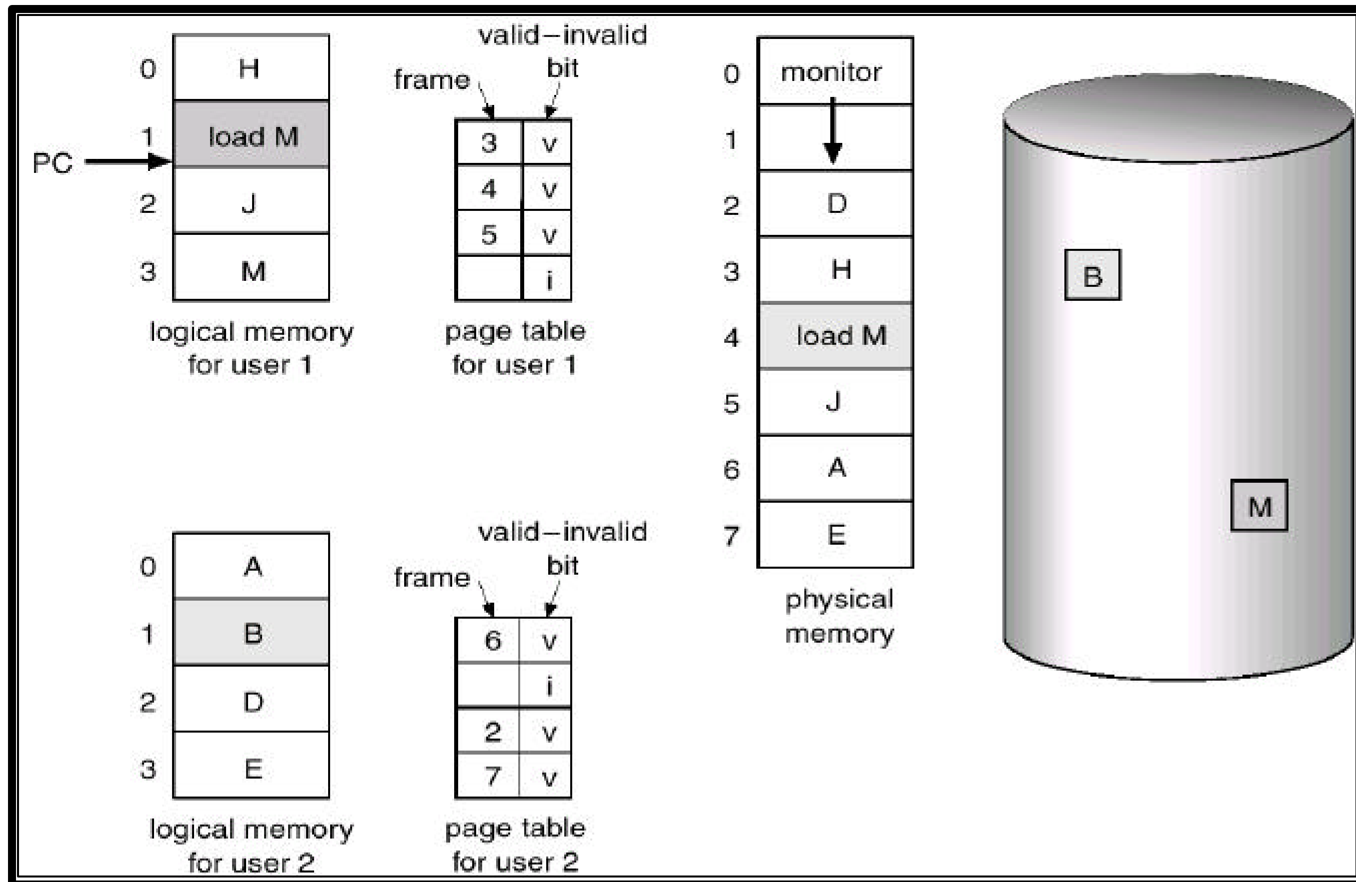
Pembuatan Proses (5)

- ❖ Penyederhanaan pengaksesan dan penggunaan file dengan membolehkan manipulasi file melalui memori lebih dari sekadar sistem pemanggilan read() dan write()
- ❖ Proses yang banyak dapat memetakan berkas yang sama ke dalam memori virtual dari masing-masing file untuk memperbolehkan pembagian data

Page Replacement

- Dasar dari ***demand paging.***
- Berlaku sebagai “**jembatan pemisah**” antara memori logis dan memori fisis
- Memori virtual yang sangat besar dapat disediakan dalam bentuk memori fisis yang kecil.

Page Replacement



Konsep *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 lama di memori.

Rutinitas *Page Replacement* (1)

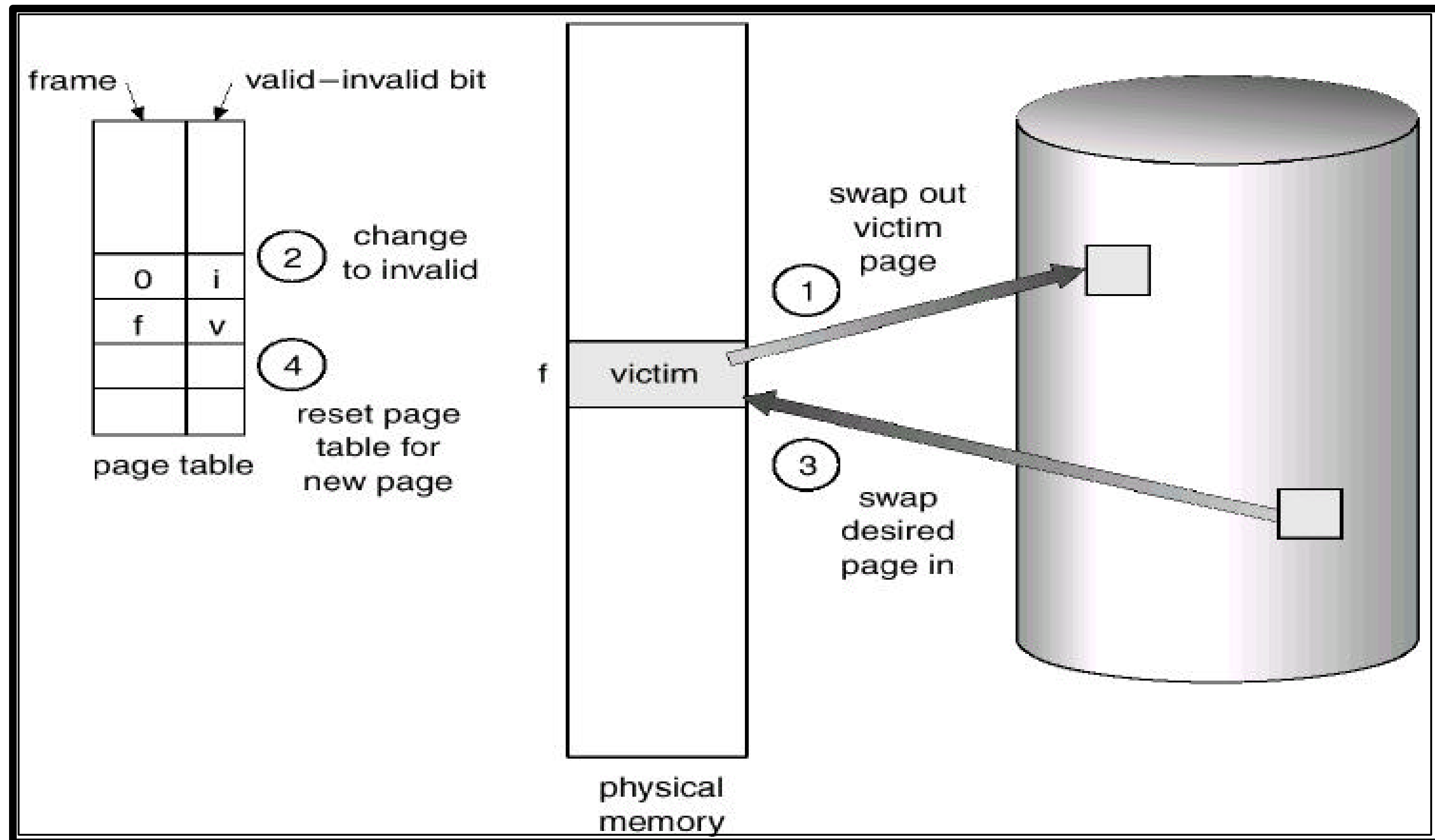
- Mencari lokasi *page* yang diinginkan pada disk.
- Mencari frame yang kosong :
  - →Jika ada, maka gunakan frame tersebut.
  - →Jika tidak ada, maka kita bisa mengosongkan frame yang tidak sedang dipakai. Gunakan algoritma *page-replacement* untuk menentukan frame yang akan dikosongkan.
  - →Tulis *page* yang telah dipilih ke ***disk***, ubah ***page-table*** dan ***frame-table***.

Rutinitas *Page Replacement* (2)

- Membaca *page* yang diinginkan ke dalam ***frame*** kosong yang baru.

- Ulangi ***user process*** dari awal.

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
  - → Algoritma ***Counting***
  - → Algoritma ***Page Buffering***

Algoritma 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*.**

Algoritma Optimal

- *Page* yang diganti adalah *page* yang tidak akan dipakai dalam jangka waktu terlama.
- Sulit diimplementasikan.
- Memiliki *page-fault* terendah.
- Tidak akan mengalami Anomali Belady.

Algoritma LRU (1)

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

Algoritma LRU (2)

- 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 me***reference*** 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.
    - →Lebih mahal.

Algoritma Perkiraan LRU

- Menggunakan bit ***reference.***
- Awalnya semua bit diinisialisasi 0 oleh sistem operasi.
- Setelah page di***reference***, bit diubah menjadi 1 oleh perangkat keras.
- Ada beberapa cara untuk implementasi algoritma ini :
  - →Algoritma ***Additional-Reference-Bits.***
  - →Algoritma ***Second-Chance.***
  - →Algoritma ***Second-Chance*** (yang diperbaiki).

Algoritma *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.
- Contoh *page* yang selalu digunakan setiap periode : 11111111.

Algoritma *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.

Algoritma ***Second-Chance*** (yang diperbaiki)

- Algoritma ini mempertimbangkan 2 hal sekaligus, yaitu bit ***reference*** dan bit modifikasi.

- Ada 4 kemungkinan :
  - → (0,0) tidak digunakan dan tidak dimodifikasi, bit terbaik untuk dipindahkan.
  - → (0,1) tidak digunakan tapi dimodifikasi, tidak terlalu baik untuk dipindahkan karena page ini perlu ditulis sebelum dipindahkan.
  - → (1,0) digunakan tapi tidak dimodifikasi, terdapat kemungkinan page ini akan segera digunakan lagi.
  - → (1,1) digunakan dan dimodifikasi, page ini mungkin akan segera digunakan lagi dan page ini perlu ditulis ke ***disk*** sebelum dipindahkan.

Algoritma *Counting*

- ❖ Menyimpan ***counter*** untuk masing-masing *page*.
- ❖ Prinsip ini dapat dikembangkan menjadi algoritma berikut :
  - → Algoritma LFU

    page yang diganti adalah page yang paling sedikit dipakai (nilai ***counter*** terkecil).
  - → Algoritma MFU

    page yang diganti adalah page yang paling sering dipakai (nilai ***counter*** terbesar).
- ❖ Lebih mahal.

Algoritma *Page Buffering*

- Sistem menyimpan ***pool*** dari *frame* yang kosong.
- Proses dapat mengulang dari awal secepat mungkin.
- Tidak perlu menunggu *page* yang akan dipindahkan untuk ditulis ke ***disk.***
- Teknik ini digunakan dalam sistem VAX/ VMS.

Alokasi *Frame*

- Jumlah ***frame*** minimum.

- Algoritma alokasi.

- Alokasi global lawan lokal.

Jumlah *Frame* Minimum

- Jumlah minimum *frame* yang dapat dialokasikan.
- Jumlah minimum *frame* ditentukan oleh arsitektur set instruksi.
- Setiap proses memerlukan jumlah minimum dari *page*.
- Bertambahnya jumlah *frame* yang dialokasikan ke setiap proses berkurang, tingkat **page-fault** bertambah dan mengurangi kecepatan eksekusi proses.

Algoritma Alokasi

- ***Fixed Allocation***

  Proses dengan prioritas tinggi ataupun rendah diperlakukan sama.

- **Alokasi prioritas**

  Perbandingan ***frame***-nya tidak tergantung pada ukuran relatif dari proses tetapi tergantung pada prioritas proses.

Fixed Allocation

- ***Equal Allocation***

  Memberikan bagian yang sama, sebanyak m/n ***frame*** untuk tiap proses.

- ***Proportional Allocation***

  Mengalokasikan memori yang tersedia kepada setiap proses tergantung pada ukurannya.

  si = ukuran memori virtual dari proses pi

  $S = S si$

  m = jumlah frame yang tersedia

  ai = alokasi untuk pi = mendekati :

  ai = si / S x m

Alokasi Global lawan Lokal

- Penggantian Global
  - → Memperbolehkan sebuah proses untuk menyeleksi sebuah ***frame*** pengganti dari himpunan semua ***frame*** .
  - → Kemungkinan meningkatkan jumlah ***frame*.**

- Penggantian Lokal
  - → Setiap proses boleh menyeleksi hanya dari himpunan ***frame*** yang telah teralokasi pada proses itu sendiri.
  - → Jumlah ***frame*** yang teralokasi pada sebuah proses tidak berubah.

Thrashing

- Proses menghabiskan waktu lebih banyak untuk ***paging*** daripada eksekusi.

- Proses sibuk untuk melakukan ***swap-in swap-out.***

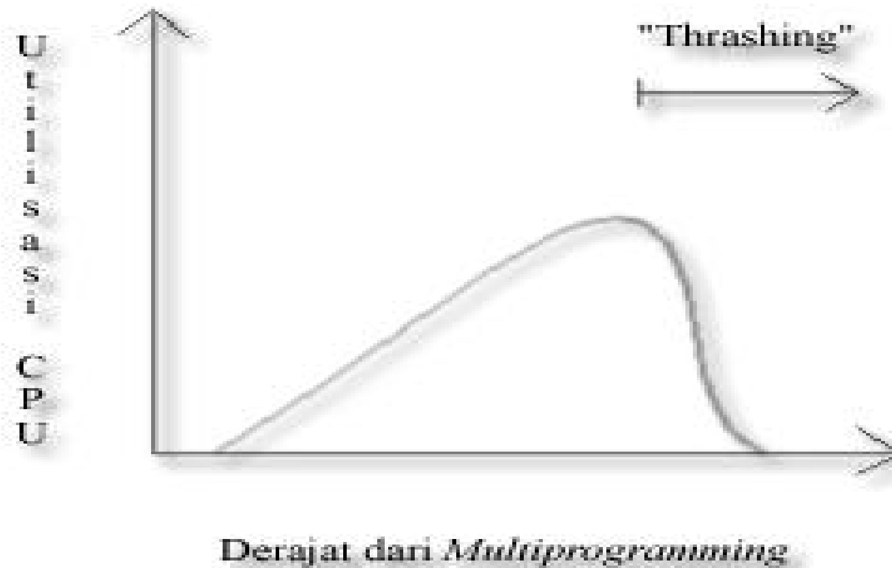
Penyebab *Thrashing*

- ❖ Rendahnya utilitas dari CPU

  Sistem meningkatkan derajat dari ***multiprogramming*** dengan menambahkan proses baru ke sistem.

- ❖ Jika derajat dari ***multiprogramming*** ditambah terus menerus, utilisasi CPU akan berkurang dengan drastis dan terjadi ***thrashing*.**

Derajat dari *Multiprogamming*



Membatasi Efek *Thrashing*

- ❖ Algoritma penggatian lokal atau prioritas
  - → Proses tersebut tidak dapat mencuri ***frame*** dari proses yang lain.
  - →Jika proses ***thrashing***, proses tersebut akan berada di antrian untuk melakukan ***paging*** yang mana hal ini memakan banyak waktu.
- ❖ Menyediakan sebanyak mungkin ***frame*** sesuai dengan kebutuhan suatu proses.

Model *Working Set* (1)

- Menggunakan parameter ? (delta) untuk mendefinisikan jendela ***working set.***
- ***Working set :*** Kumpulan dari ? *page* dengan *page* yang dituju yang paling sering muncul.
- Mempertahankan derajat multiprogramming setinggi mungkin.

Model *Working Set* (2)

❖ Keakuratan ***working set*** tergantung pada pemilihan ? :

→Jika ? terlalu kecil, tidak akan dapat mewakilkan keseluruhan dari lokalitas.

→Jika terlalu besar, akan menyebabkan ***overlap*** beberapa lokalitas.

→Jika tak terhingga, ***Working Set*** adalah kumpulan *page* sepanjang eksekusi proses.

Working Set

- Menghitung ukuran total ***frame*** yang diminta (WWSi)

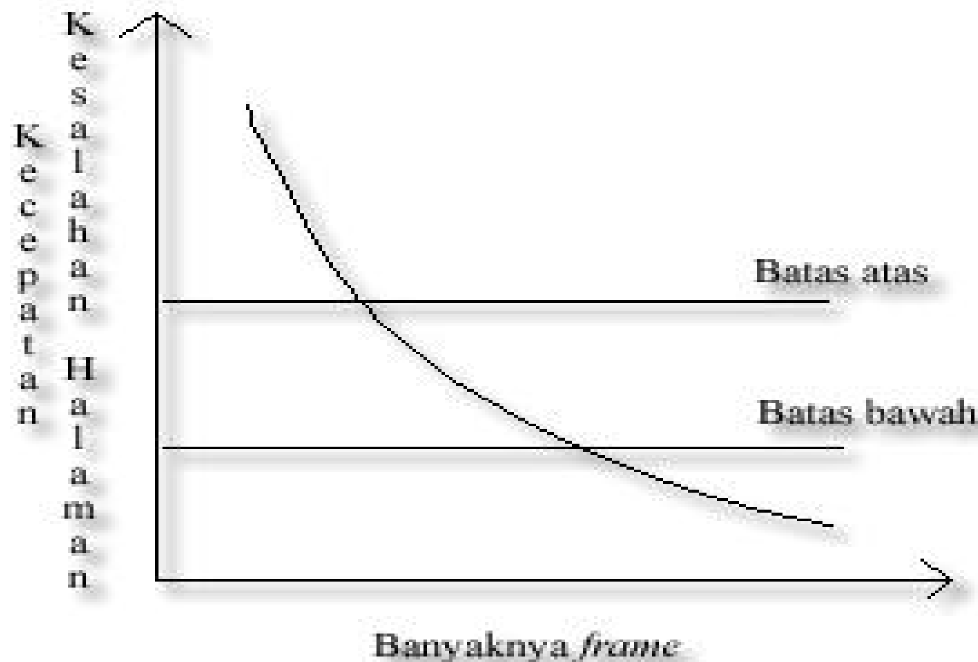
  D = ? WSSi

  jika D < m, ***trashing*** dapat terjadi

- Kesulitan dari model ***Working Set*** adalah Menjaga ***track*** dari ***Working Set*** :
  - → Mendekati model ***Working Set*** dengan ***fixed interval timer interrupt*** dan ***reference*** bit.

Frekuensi *page-fault*

❖ Mengambil pendekatan yang lebih langsung untuk mengontrol ***thrashing*.**



❖ Jika kesalahan page terlalu tinggi, proses membutuhkan ***frame*** lebih.

❖ Jika kesalahan *page* terlalu rendah, maka perlu kita pindahkan ***frame*** dari proses tersebut.

Prepaging

- ***Prepaging***: Membawa dalam satu waktu seluruh ***page*** yang dibutuhkan ke dalam memori. Dapat bermanfaat pada situasi tertentu.
- **s** = jumlah ***page*** yang di ***prepage*.**
- **a** = **%** jumlah ***page*** yang benar2 digunakan
- Jika biaya untuk ***prepaging*** s x (100% – a) *page* yang tidak dibutuhkan lebih kecil dari biaya s x a ***page fault*** yang disimpan, maka ***prepaging*** akan menguntungkan.

Pemilihan Ukuran Page

- ❖ Ukuran tabel → ukuran ***page*** besar
- ❖ Fragmentasi → kecil
- ❖ *I/O overhead* → besar
- ❖ Lokalitas → kecil
- ❖ *Page-fault* → besar

TLB Reach

- ❖ Jumlah memori yang dapat diakses dari ***TLB*.**

- ❖ ***TLB reach*** = (jumlah ***entry***) x (ukuran ***page***).

- ❖ Idealnya, ***working set*** dari sebuah proses disimpan dalam ***TLB***, jika tidak proses tersebut akan menghabiskan waktu yang cukup banyak menangani ***memory references*** dalam ***page table*** daripada ***TLB*.**

Meningkatkan *TLB Reach*

- Menambah ukuran *page*

  Dapat meningkatkan ***fragmentation*** karena tidak semua aplikasi membutuhkan ukuran ***page*** yang besar.

- Menyediakan ukuran *page* yang bervariasi

  Sistem operasi harus mengatur ***TLB*** yang akan menambah biaya pada performa. Akan tetapi, peningkatan ***hit ratio*** dan *TLB reach* dapat menutupi biaya tersebut.

Contoh Program

```
int A[][] = new int [512] [512];

for (int j = 0; j < A.length; j++)
   for (int i = 0; i < A.length; i++)
        A[i][j] = 0;
```

512 x 512 page faults!

```
for (int i = 0; i < A.length; i++)
   for (int j = 0; j < A.length; j++)
        A[i][j] = 0;
```

512 page faults

I/O Interlock

- Saat menggunakan ***demand paging***, kita terkadang harus mengizinkan beberapa page untuk dikunci dalam memori.
- Misalnya jika sebuah proses mengeluarkan permintaan ***I/O***. ***Page*** tersebut harus dikunci agar tidak menjadi korban algoritma ***page replacement.***

Contoh Sistem Operasi (1)

Windows NT

- Mengimplementasi memori virtual menggunakan ***demand paging*** dengan ***clustering***. ***Clustering*** menangani ***page fault*** dengan menambahkan tidak hanya ***page*** yang mengalami ***fault*** tetapi juga ***page-page*** lain yang disekelilingnya.
- Proses diberikan ***working set*** minimum dan ***working set*** maksimum.
- ***Working set*** minimum → jumlah ***page*** minimum yang dijamin akan dimiliki proses tersebut dalam memori.
- Jika memori yang tersedia cukup, proses dapat diberikan ***page*** sebanyak ***working set*** maksimumnya.
- Jika jumlah memori jatuh dibawah batas, manajer memori virtual menggunakan ***automatic working set trimming*** untuk menambah memori agar melebihi batas → membebaskan *page* dari proses yang memiliki ***page*** lebih dari ***working set*** minimumnya.

Contoh Sistem Operasi (2)

Solaris 2

- ❖ ***Kernel*** menjaga sejumlah ***free memory*** yang cukup untuk menangani ***page fault.***
- ❖ ***Lotsfree*** → parameter batas untuk memulai ***paging*** (biasanya 1/64 dari ukuran memori fisik).
- ❖ Jika ***page*** bebas < batas, ***pageout*** dimulai, menggunakan algoritma ***two-handed-clock*****.**
- ❖ ***Scanrate*** bergantung pada jumlah memori bebas.
- ❖ Proses ***page-out*** akan dipanggil lebih sering jika jumlah memori bebas berkurang.

Solaris 2 ***page scanner***