SISTEM OPERASI

PERTEMUAN IX : MANAJEMEN MEMORI

PREFACE

- Proses merupakan suatu bagian aktif dari KOMPUTASI
- Suatu komputasi / setiap proses yang akan dijalankan harus melalui memori terlebih dahulu
- CPU mengambil instruksi dari memori sesuai yang ada pada Program Counter
- Memori komputer merupakan salah satu *resource* komputer yang berharga dan perlu dikelola dengan baik oleh Sistem Operasi karena kapasitasnya yang terbatas

PREFACE

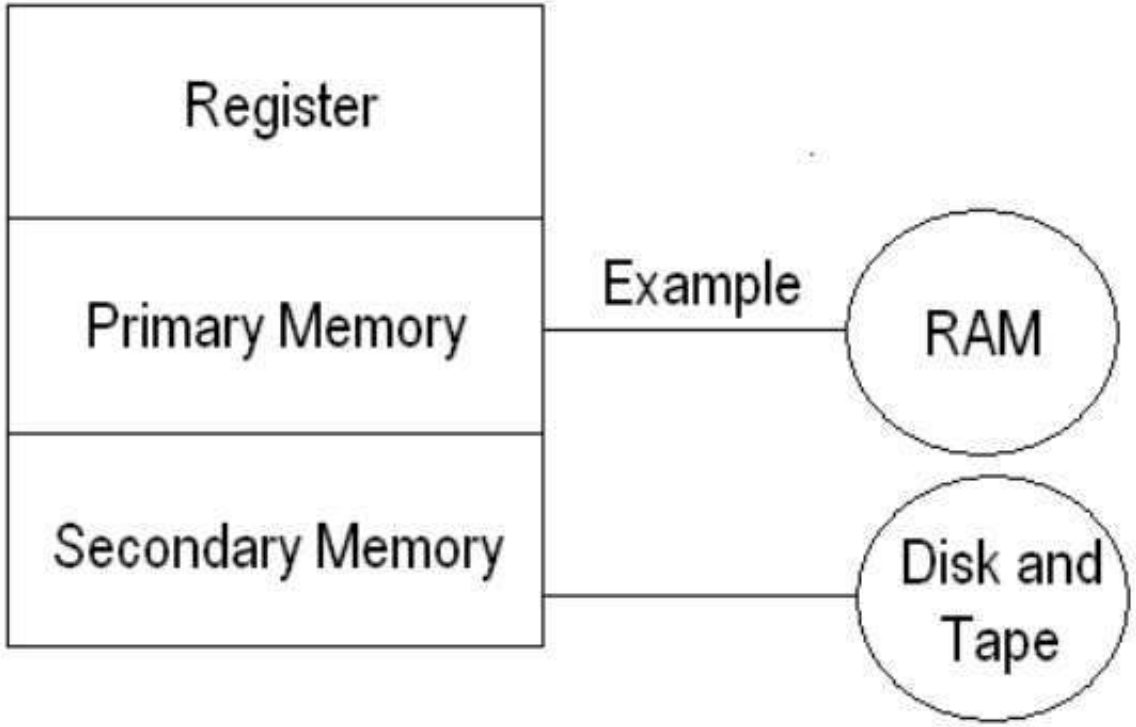
- Kerumitan pengelolaan bertambah seiring dengan adanya fitur *multiprogramming, multitasking* dll dimana lokasi memori tidak hanya diakses oleh satu proses tetapi akan diakses oleh beberapa proses secara konkuren
- Pengelola memori merupakan salah satu komponen penting dari Sistem Operasi

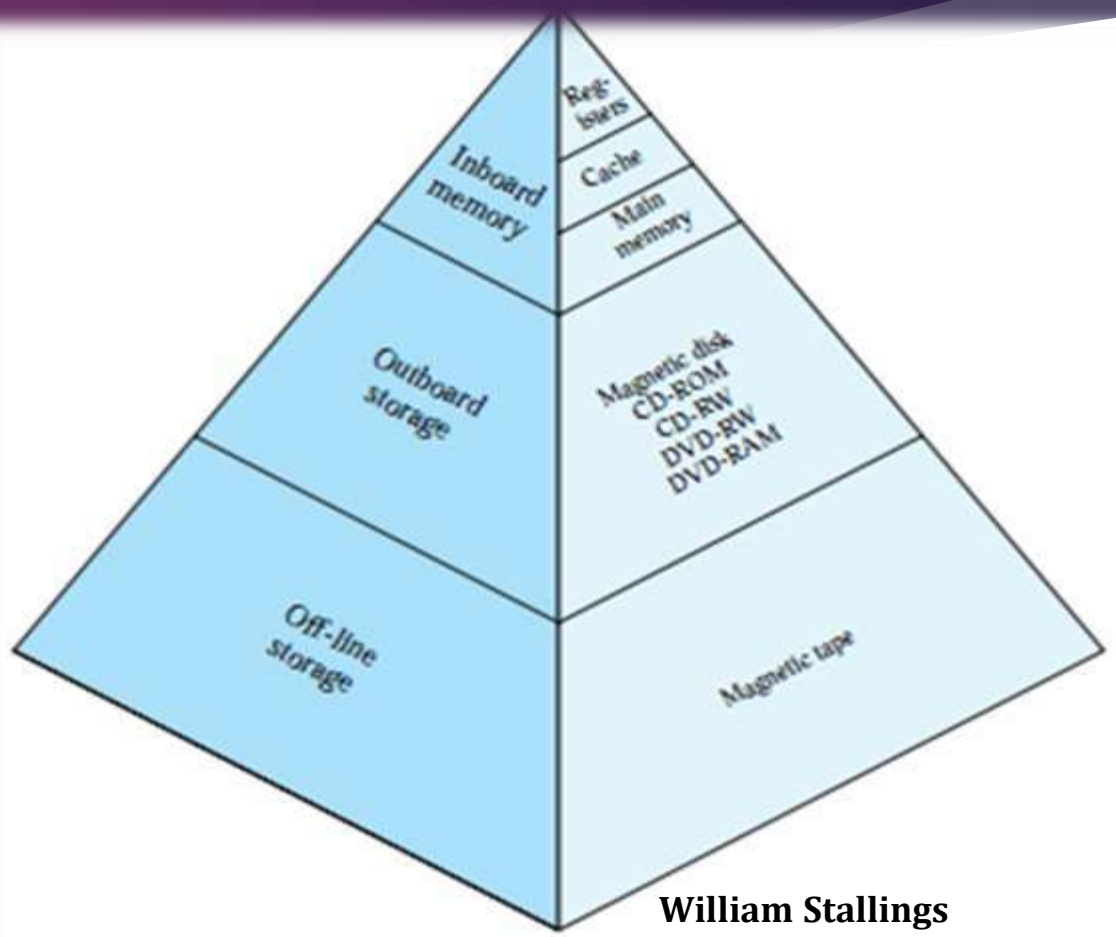
PREFACE

Sebagian besar komputer memiliki hirarki memori yang terdiri atas tiga level, yaitu:

- **Register** di CPU, berada di level teratas. Informasi yang berada di register dapat diakses dalam satu clock cycle CPU.
- **Primary Memory** (executable memory), berada di level tengah. Contohnya, RAM. Primary Memory diukur dengan satu byte dalam satu waktu, secara relatif dapat diakses dengan cepat, dan bersifat volatile (informasi bisa hilang ketika komputer dimatikan).
- **Secondary Memory**, berada di level bawah. Contohnya, disk atau tape. Secondary Memory diukur sebagai kumpulan dari bytes (block of bytes), waktu aksesnya lambat, dan bersifat non-volatile (informasi tetap tersimpan ketika komputer dimatikan).

PREFACE





William Stallings

Figure 1.14 The Memory Hierarchy

PREFACE

- Bagian dari sistem operasi yang mengatur hirarki memori disebut dengan ***memory manager.***
- Di era *multiprogramming* ini, ***memory manager*** digunakan untuk :
  - mencegah satu proses dari penulisan dan pembacaan oleh proses lain yang dilokasikan di primary memory,
  - mengatur *swapping* antara memori utama dan disk ketika memori utama terlalu kecil untuk memegang semua proses.
  - Mengalokasi dan mendealokasi memori

PREFACE

- Setiap proses mempunyai ***memory allocator***
- ***Memory allocator*** mengalokasikan ruang kosong dari memori kepada suatu proses .
- Ketika suatu proses kekurangan memori, ***memory allocator*** ini yang berperan untuk meminta kepada *sistem operasi* untuk diberikan *space* memori kosong dari proses lain.

MANAJEMEN MEMORI

Tujuan manajemen memory :

- Meningkatkan utilitas CPU
- Data dan instruksi dapat diakses dengan cepat oleh CPU
- Efisiensi dalam pemakaian memori yang terbatas
- Transfer dari/ke memori utama ke/dari CPU dapat lebih efisien

MANAJEMEN MEMORI

5 syarat Manajemen Memori :

1. ***Relocation***
2. ***Protection***
3. ***Sharing***
4. ***Logical Organization***
5. ***Physical Organization***

MANAJEMEN MEMORI

1. Relocation

e.g : *swap in swap out*

2. Protection

e.g : *interference by other processes*

Perlindungan atau proteksi ini disediakan oleh perangkat keras. Perangkat keras menyediakan dua register, yaitu **base register** dan **limit register**.

MANAJEMEN MEMORI

2. Protection *(Cont…)*

Base register ➔ memegang alamat fisik terkecil yang dilegalkan,

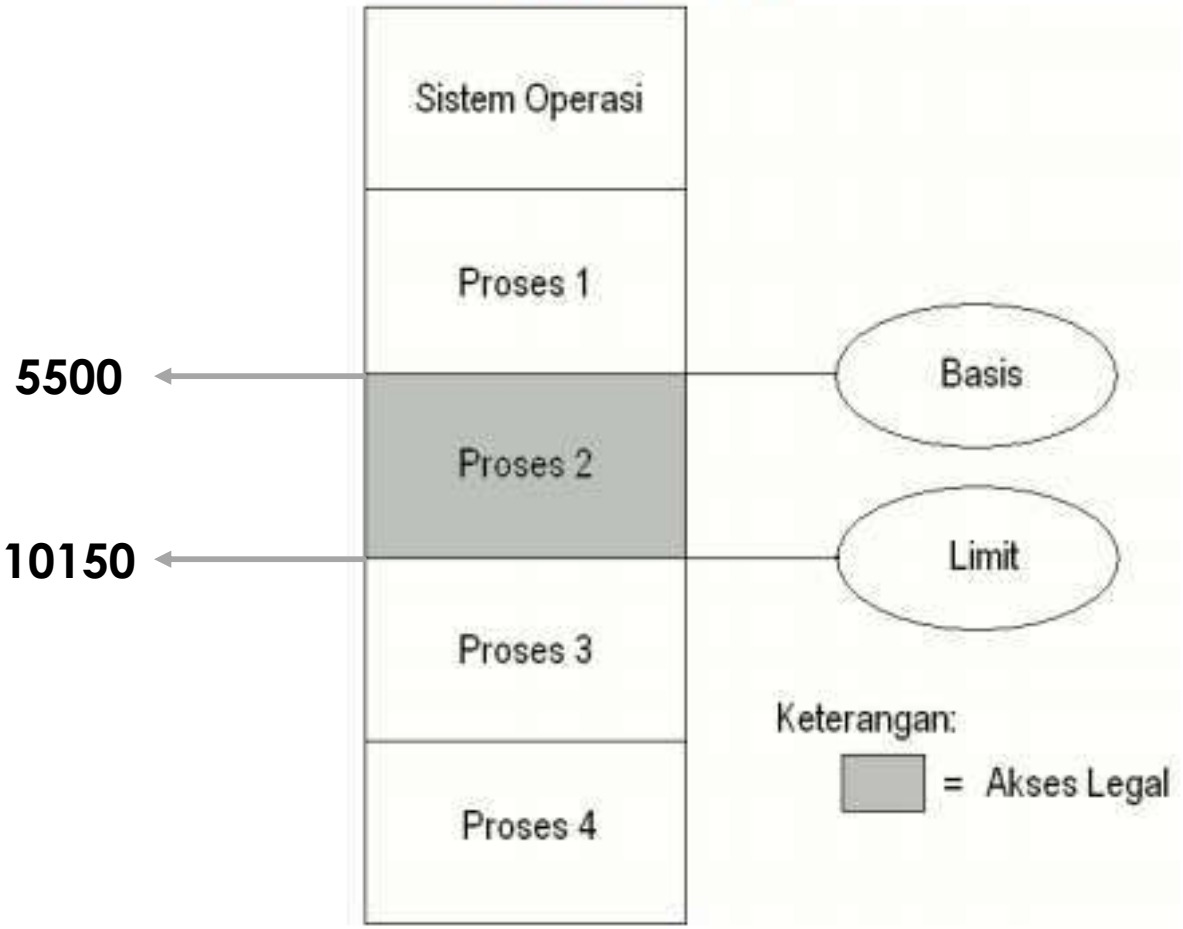
Limit register ➔ menentukan ukuran dari jarak alamat tersebut.

e.g ➔ jika *base register* memegang alamat 5500 dan *limit register* 4650 , maka program bisa mengakses secara legal di semua alamat dari 5500 sampai 10150 (5500+4650).

MANAJEMEN MEMORI

2. Protection

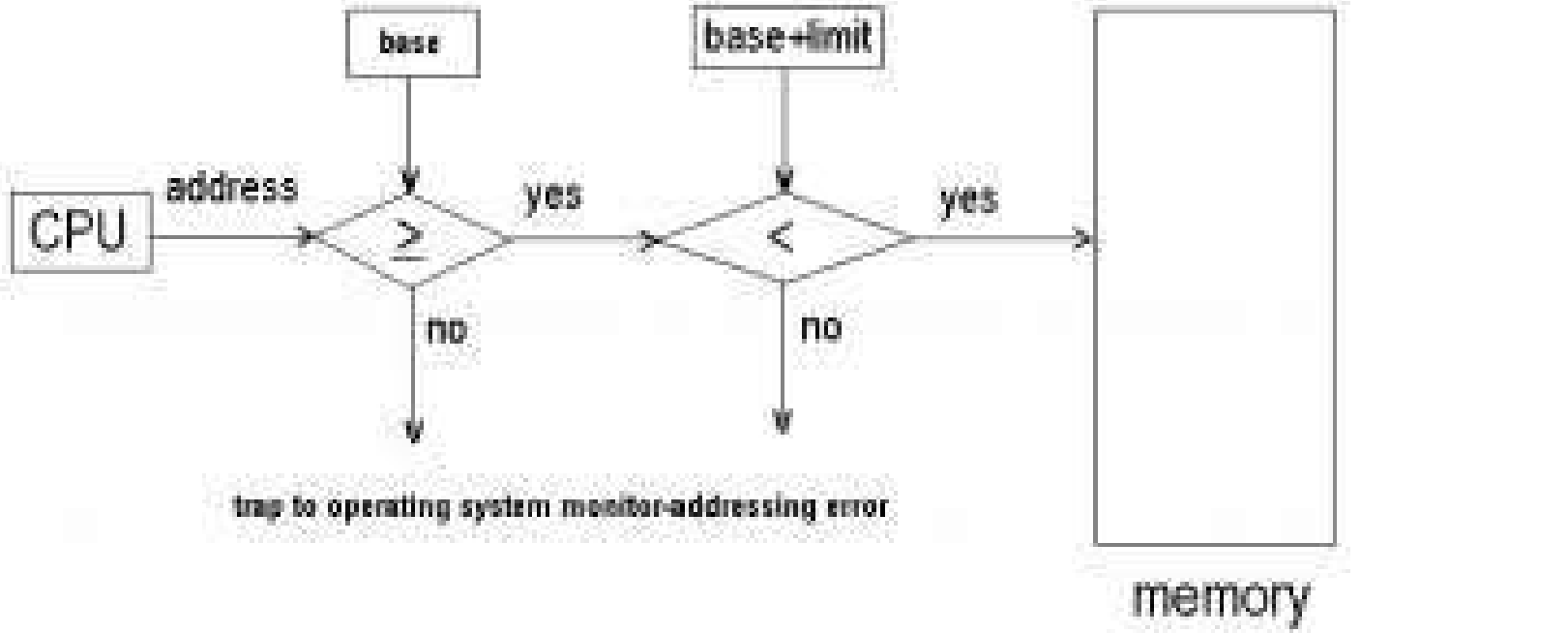
Gambar 1.2. Gambar *Base* dan Limit Register



MANAJEMEN MEMORI

2. Protection

Gambar 1.3. Gambar Proteksi Perangkat Keras dengan *base* dan *limit register*



MANAJEMEN MEMORI

3. Sharing

e.g : *allow several processes to access the same portion of main memory*

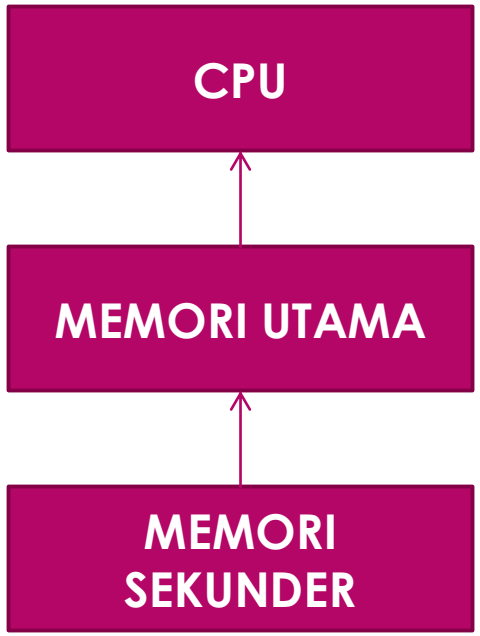
4. Logical Organization

e.g : *pengorganisasian program antara memori logic dan memori fisik* ***(Lebih jelasnya akan dibahas pada bagian Paging dan Segmentasi)***

MANAJEMEN MEMORI

5. Physical Organization

memori diorganisasikan dalam dua tingkatan : **memori utama** dan **memori sekunder.**



Notes

Pengorganisasian antara memori utama dengan memori sekunder sangat diperlukan agar aliran informasi antara dua tingkatan memori ini dapat berjalan dengan lancar

ADDRESS BINDING

"Sebuah prosedur untuk menetapkan alamat fisik yang akan digunakan oleh program yang terdapat di dalam memori utama."

Binding → pemetaan dari satu ruang alamat ke alamat yang lain.

Binding instruksi dan data ke memori dapat dapat terjadi dalam tiga cara yang berbeda:

1. Compilation Time.
2. Load Time.
3. Execution Time.

ADDRESS BINDING

1. ***Compilation Time*** **:** jika lokasi memori telah diketahui sebelumnya, , maka alamat absolut atau alamat fisik dapat dibuat. Kita harus mengkompilasi ulang kode jika lokasi awal berubah.
2. ***Load Time*** **:** harus membangkitkan alamat relocatable (alamat yang dapat dialokasikan ulang) jika lokasi memori tidak diketahui pada saat kompilasi
3. ***Execution Time*** **:** Alamat bersifat relatif, binding akan dilakukan pada saat run time. Pada saat run time dibutuhkan bantuan hardware yaitu MMU (Memory Management Unit)

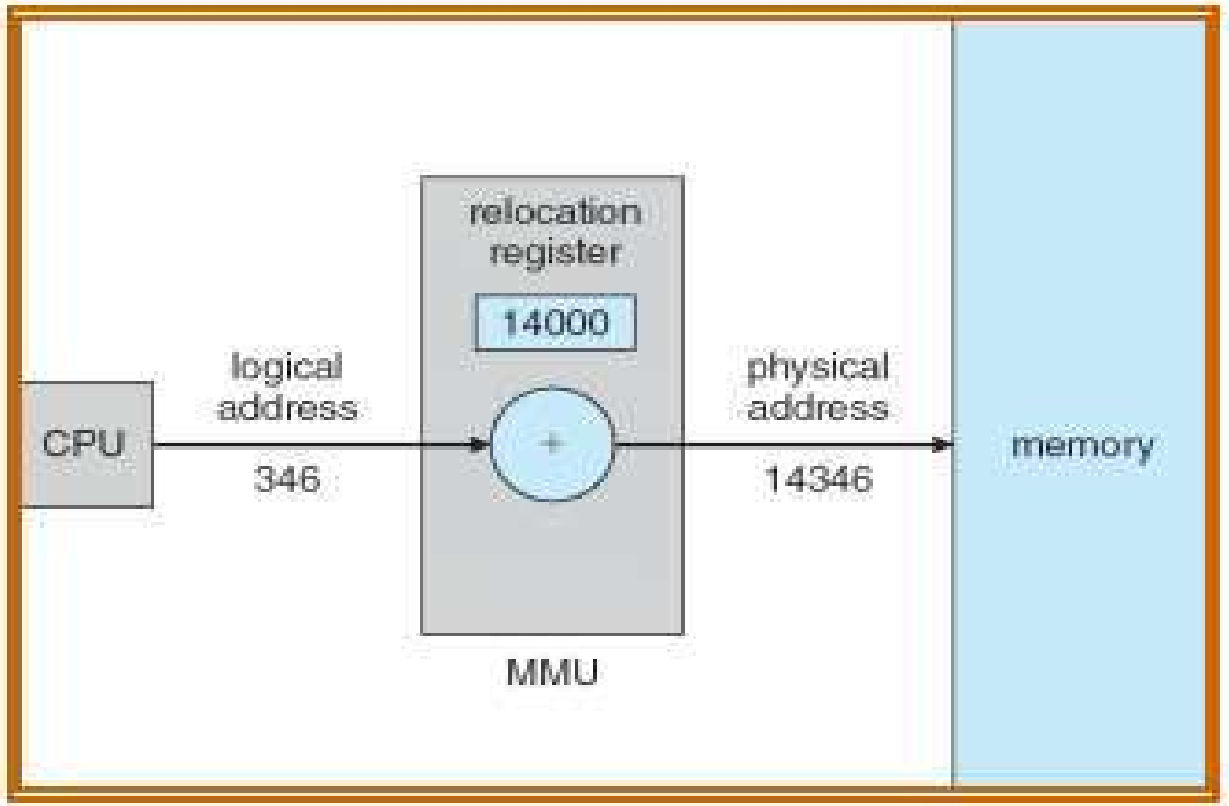
RUANG ALAMAT LOGIKA DAN FISIK

- Alamat yang dihasilkan oleh CPU berupa alamat logika, disebut juga **alamat virtual**
- **Alamat fisik** adalah alamat memori yang sebenarnya.
- Pada saat ***compile time*** dan ***load time***, alamat fisik dan logika identik.
- Sebaliknya, perbedaan alamat fisik dan logika terjadi pada ***execution time***.
- Kumpulan semua alamat logika yang dihasilkan oleh program adalah **ruang alamat logika/ruang alamat virtual.**
- Kumpulan semua alamat fisik yang berkorespondensi dengan alamat logika disebut **ruang alamat fisik.**

RUANG ALAMAT LOGIKA DAN FISIK

- Pada saat program berada di CPU, program tersebut memiliki alamat logika, kemudian **MMU** memetakan menjadi alamat fisik yang akan disimpan di dalam memori.
- ***Memory Management Unit (MMU)*** : Untuk memetakan alamat virtual (alamat logic) ke alamat fisik.
- ***Register utamanya (base register)disebut register relokasi.***
- Nilai pada **register relokasi** akan bertambah setiap alamat dibuat oleh proses pengguna dan pada waktu yang sama alamat ini dikirimkan ke memori.
- Ilustrasinya sebagai berikut, nilai pada register ini akan ditambah dengan setiap alamat yang dibuat oleh user process yang kemudian dikirim ke memori. **Contohnya *register relokasi berada di 14000, alamat logika di 346, maka langsung dipetakan menjadi alamat fisik di 14346.***

RUANG ALAMAT LOGIKA DAN FISIK



Gambar 7.2 Relokasi dinamis menggunakan register relokasi

PEMETAAN MEMORI

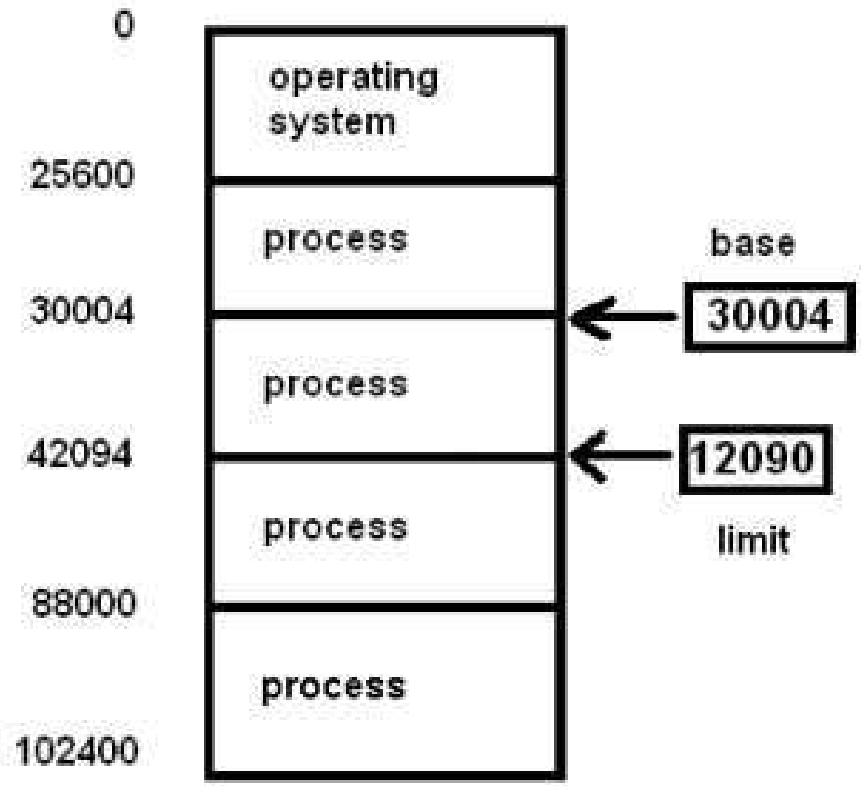
- System operasi dan berbagai macam proses pengguna terletak dalam memori utama.
- Memori ini biasanya dibagi menjadi dua bagian :
  - Sistem operasi dan
  - Proses pengguna
- Pemetaan memori (*memory mapping*) membutuhkan sebuah register relokasi, merupakan *base register* yang ditambahkan ke dalam tiap alamat proses pengguna pada saat dikrimkan ke memori.

PEMETAAN MEMORI

- Pada contoh gambar diatas terlihat sebuah proses memiliki *base register* 30004 dan *limit register* 12090, akan di-*mapping* ke memori fisik dengan alamat awalnya sesuai dengan *base register* (30004) dan berakhir pada alamat (30004 + 12090 = 42094)

- ketika CPU memilih suatu proses untuk dieksekusi, ia akan memasukkan **register relokasi** dan **limit registernya**

Gambar 2.2. *Base dan Limit Register*



base dan limit register

PARTISI MEMORI

- **Partisi tetap**
  - Cara ini memungkinkan pembagian yang tidak sama rata.
- **Partisi dinamis**
  - Menggunakan konsep kumpulan lubang-lubang (*hole* / ruang memori kosong) dalam berbagai ukuran yang tersebar di seluruh memori sepanjang waktu.

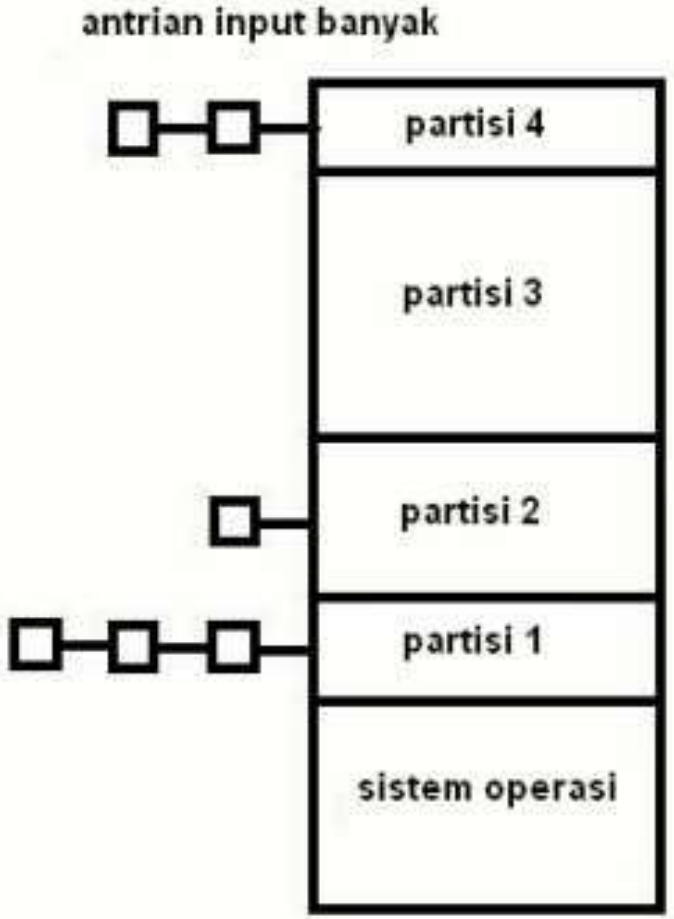
PARTISI MEMORI

- **Partisi tetap**

  "Pada awalnya semua partisi kosong dan dianggap sebagai sebuah blok besar yang tersedia (hole). Ketika sebuah proses datang dan membutuhkan memori, ia akan dicarikan lubang yang cukup besar yang mampu menampungnya. Setelah menemukannya, memori yang dialokasikan untuknya hanyalah sebesar memori yang dibutuhkannya sehingga menyisakan tempat untuk memenuhi kebutuhan proses lain."

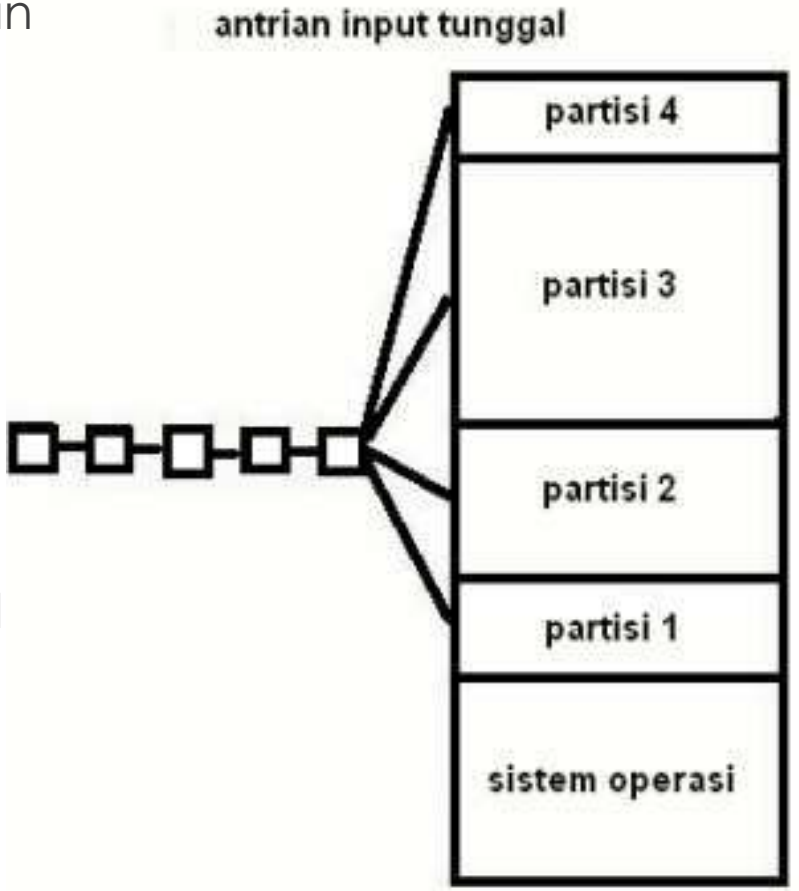
PARTISI MEMORI → Partisi Tetap

- Ketika sebuah proses datang, ia akan diletakkan ke dalam *input queue* (antrian proses pada disk yang menunggu dibawa ke memori untuk dieksekusi) sesuai dengan ukuran terkecil partisi yang mampu menampungnya.
- **Kerugian** : partisi yang besar akan menjadi kosong karena tidak ada proses dengan ukuran sesuai yang diletakkan di partisi tersebut. Dan antrian untuk partisi dengan ukuran kecil sangat padat karena banyaknya proses dengan ukuran yang sesuai.
- **Solusi → Antrian Input Tunggal**



PARTISI MEMORI ➔ Partisi Tetap

- Proses dengan ukuran sesuai partisi tersebut yang terletak di depan antrian dapat dimasukkan lalu dieksekusi.
- **Kerugian** : jika proses yang memasuki partisi yang cukup besar ternyata ukurannya jauh lebih kecil dari partisi itu sendiri.
- **Solusi** → Mencari proses terbesar ke dalam seluruh antrian yang dapat ditampung oleh sebuah partisi pada saat itu
- Dalam partisi tetap ini, sistem operasi menggunakan sebuah tabel untuk mengindikasikan bagian memori mana yang kosong dan mana yang terisi.



PARTISI MEMORI ➔ Partisi Tetap

Notes

- Dalam partisi tetap ini, sistem operasi menggunakan sebuah tabel untuk mengindikasikan bagian memori mana yang kosong dan mana yang terisi.
- Sistem operasi mencatat kebutuhan memori masing-masing proses yang berada dalam antrian
- Sistem operasi juga mencatat jumlah memori yang masih tersedia untuk menentukan proses mana yang harus dimasukkan

PARTISI MEMORI

- **Partisi Dinamis**

  "Sistem operasi akan mencari lubang yang cukup besar untuk menampung memori tersebut. Jika terdapat lubang yang terlalu besar, maka ia akan dipecah menjadi 2, satu bagian digunakan untuk menampung proses tersebut dan bagian lain akan digunakan untuk bersiap-siap menampung proses lain."

PARTISI MEMORI ➔ Partisi Dinamis

- Hal utama yang harus diperhatikan dari alokasi penyimpanan dinamis adalah bagaimana memenuhi permintaan proses berukuran N dengan kumpulan lubang-lubang yang tersedia.
- Digunakan beberapa algoritma :
  - First Fit
  - Best Fit
  - Next Fit
  - Worst Fit

PARTISI MEMORI ➔ Partisi Dinamis

- First Fit

  "Pencarian lubang dimulai dari awal, selanjutnya mengalokasikan lubang pertama ditemukan dan besarnya mencukupi. Pencarian dimulai dari awal. Jika ukuran lubang lebih besar dari ukuran proses maka akan dibagi menjadi 2 bagian : Untuk proses itu sendiri dan untuk proses lain yang membutuhkan."

- Best Fit

  "Mengalokasikan lubang dengan besar minimum yang mencukupi permintaan. best fit mencari lubang dengan ukuran yang hampir sama dengan yang dibutuhkan oleh proses."

PARTISI MEMORI ➔ Partisi Dinamis

- Next Fit

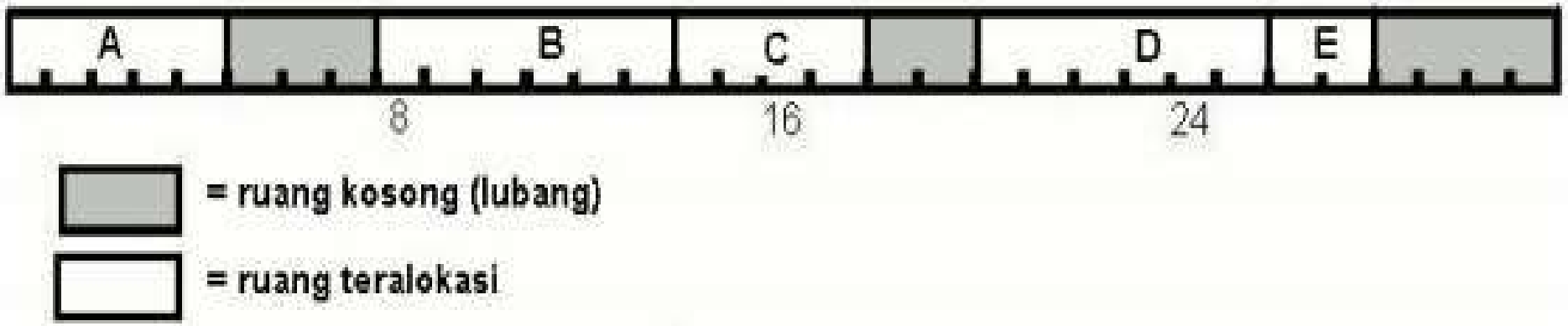
  "Mengalokasikan lubang pertama ditemukan yang besarnya mencukupi. Pencarian dimulai dari akhir pencarian sebelumnya, pendeknya algoritma ini tidak melakukan pencarian dari awal."

- Worst Fit

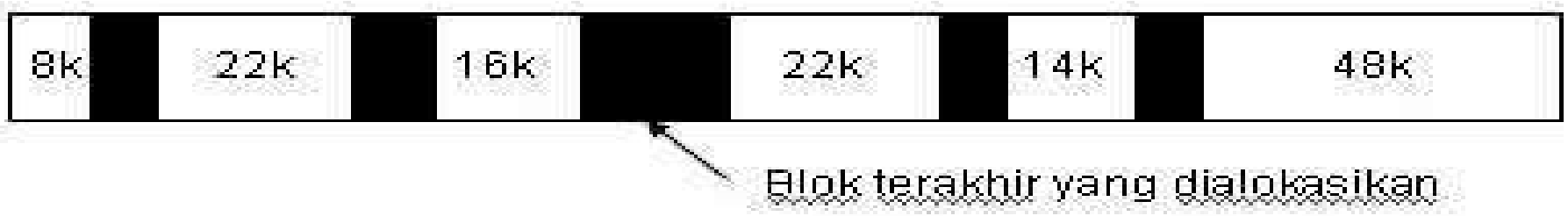
  "Mengalokasikan lubang terbesar yang ada."

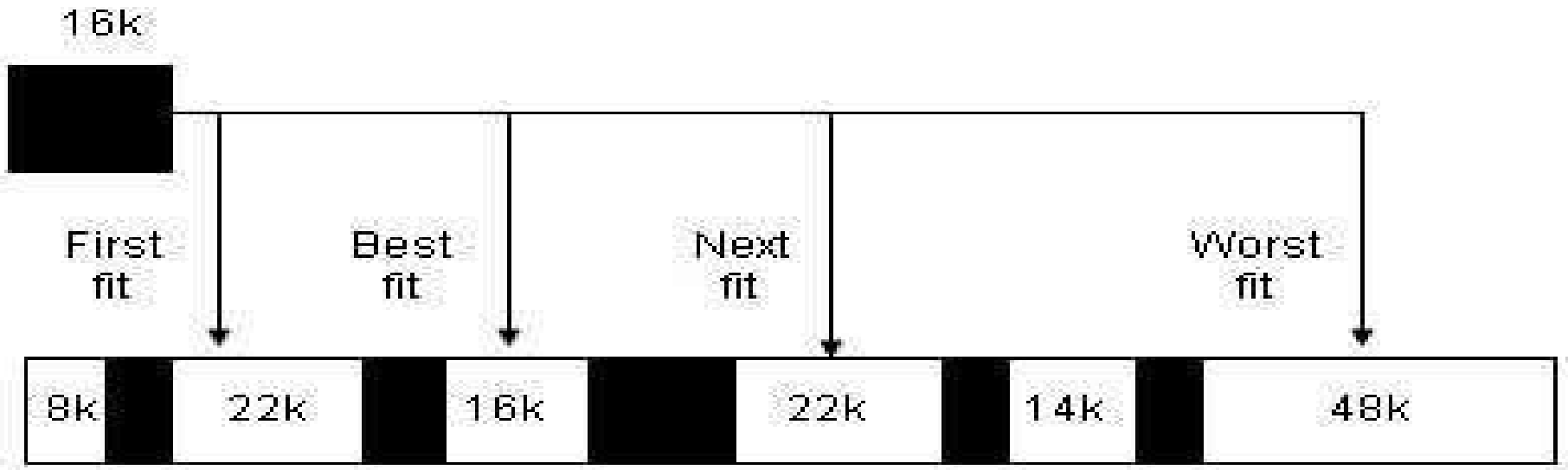
PARTISI MEMORI ➔ Partisi Dinamis

Gambar 2.4. Bagian Memori dengan 5 Proses dan 3 Lubang



PARTISI MEMORI ➔ Partisi Dinamis





PARTISI MEMORI

- Keuntungan dan Kerugian :
  - Menggunakan *first fit, best fit* dan *worst fit* berarti harus selalu memulai pencarian lubang dari awal, kecuali apabila lubang sudah disusun berdasarkan ukuran.
  - Metode *worst fit* akan menghasilkan sisa lubang yang terbesar, sementara metoda *best fit* akan menghasilkan sisa lubang yang terkecil.