Manajemen Memori

(model awal)

Pendahuluan

- Manajemen memori utama sangat penting
- Kinerja keseluruhan sistem sangat tergantung pada dua hal:
  - Seberapa banyak memori tersedia
  - Optimasi memori selama pemrosesan job
- Bab ini membahas:
  - Memory manager
  - 4 tipe skema alokasi memori
    - Sistem dengan satu pengguna
    - Fixed partitions / Partisi tetap
    - Dynamic partitions / Partisi dinamis
    - Relocatable dynamic partitions / Partisi dinamis yang bisa direlokasi

Pendahuluan

- Alamat yang dibuat CPU merujuk ke sebuah alamat logik.
- Alamat yang dilihat oleh memori adalah alamat yang dimasukkan ke register di memori, merujuk pada alamat fisik
- Pada waktu compile dan waktu penempatan alamat logik dan alamat fisik sama.
- Sedangkan pada waktu eksekusi menghasilkan alamat fisik dan logik yang berbeda.
- Pemetaan dari virtual ke alamat fisik dilakukan oleh Memory-Management Unit (MMU)

Single-User Contiguous Scheme

- Mulai digunakan komersil tahun 1940an dan 1950an
- Keseluruhan program di load kedalam memori (hanya bisa untuk 1 program) => single-user
- Memori dialokasikan sebanyak yang dibutuhkan secara berurutan
- Jobs diproses sekuensial
- Tugas memory manager sedikit.
  - Register untuk menyimpan alamat base
  - Accumulator untuk melacak ukuran pogram

Single-User Contiguous Scheme (cont.)

- Kekurangan
  - Belum mendukung multiprogramming atau networking
  - Tidak hemat biaya
  - Untuk bisa dieksekusi, ukuran program harus lebih kecil dari ukuran memori.

Fixed Partitions

- Mulai digunakan secara komersial tahun 1950an dan 1960an
- Memori utama di-partisi
  - *Saat startup sistem*
  - Satu partisi (berdekatan) per job
- Mendukung multiprogramming
- Ukuran partisi tetap
  - Untuk di konfigurasi ulang, komputer harus di shut down
- Perlu:
  - Menjaga space memori yang digunakan job-job
  - Mencocokan ukuran job dengan ukuran partisi

Fixed Partitions (cont.)

- Memory manager bertugas mengalokasikan ruang memori untuk job – job.
  - Menggunakan tabel

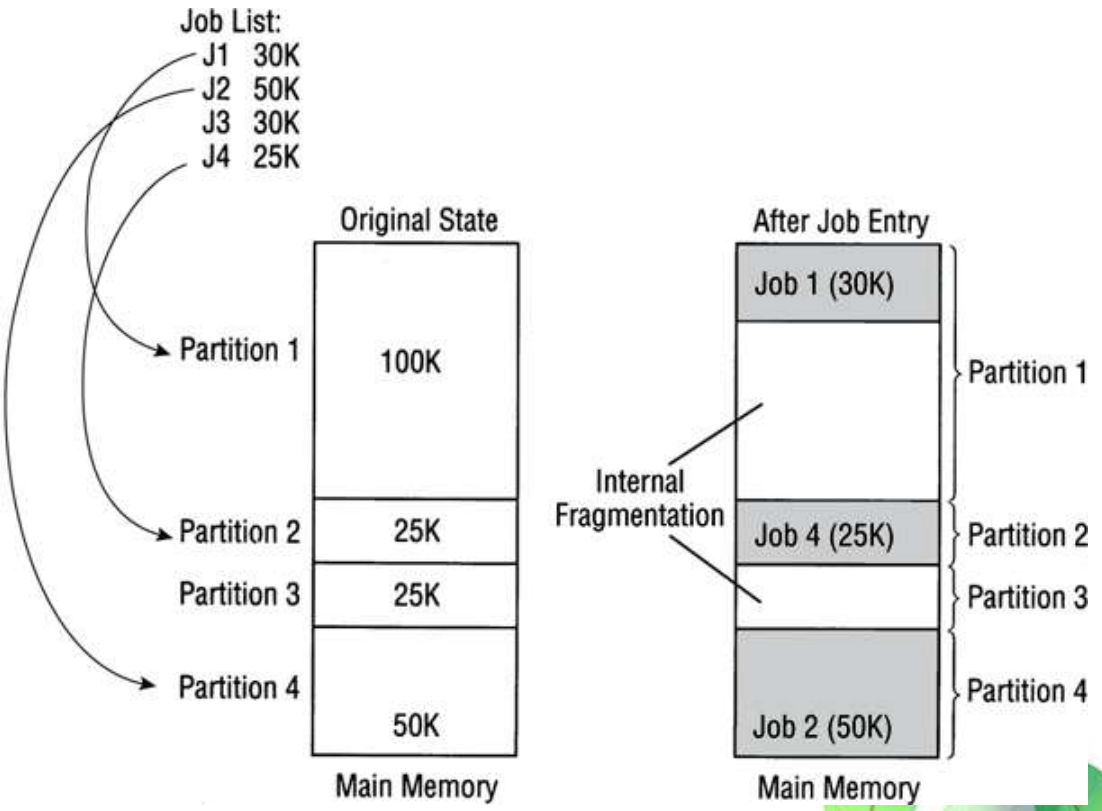
(table 2.1) *A simplified fixed-partition memory table with the free partition shaded.*

Partition Size	Memory Address	Access	Partition Status
100K	200K	Job 1	Busy
25K	300K	Job 4	Busy
25K	325K		Free
50K	350K	Job 2	Busy

Fixed Partitions (continued)

(figure 2.1)

Main memory use during fixed partition allocation of Table 2.1. Job 3 must wait even though 70K of free space is available in Partition 1 where Job 1 only occupies 30K of the 100K available. The jobs are allocated space on the basis of "first available partition of required size."

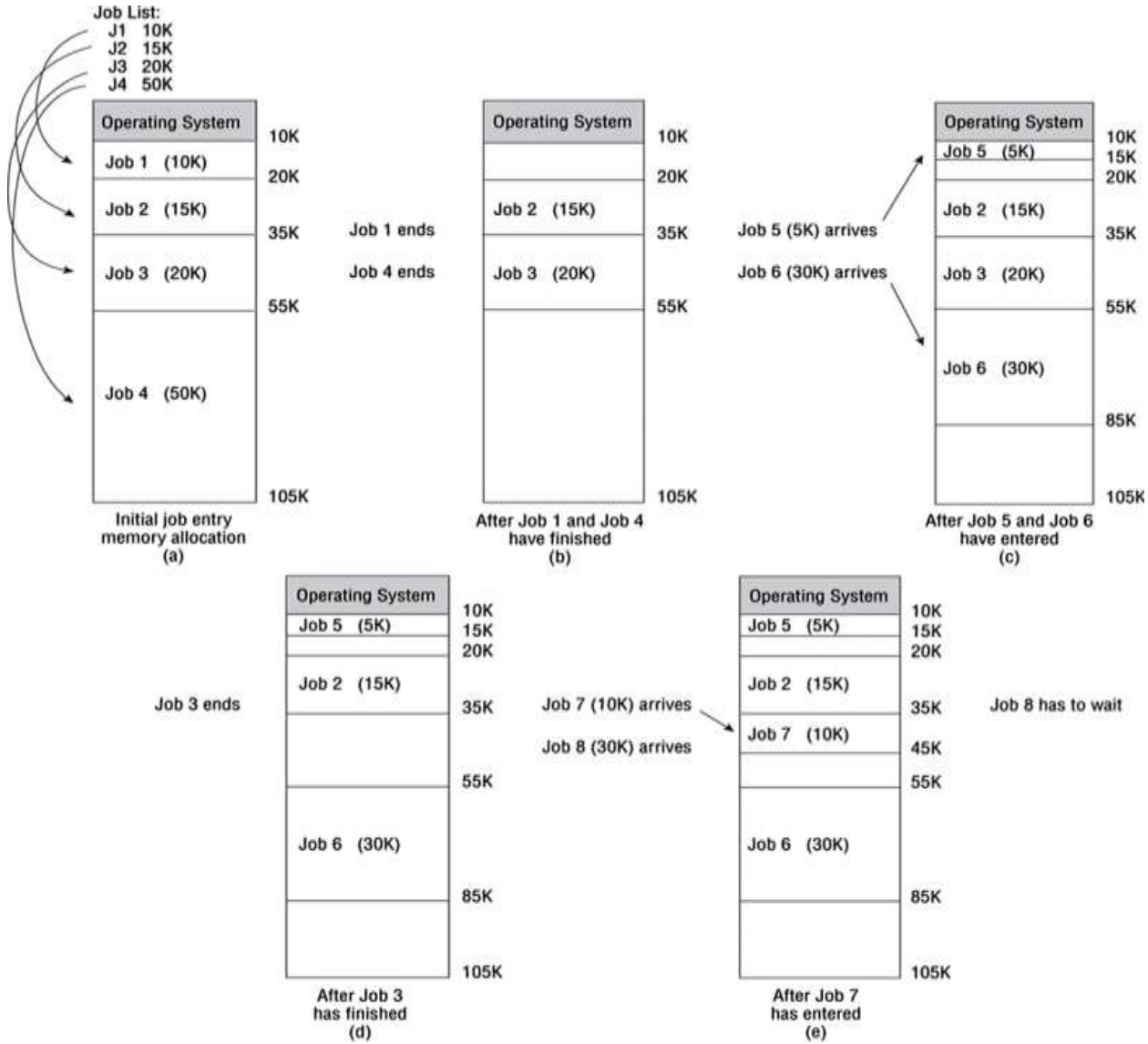


Fixed Partitions (continued)

- Kekurangan
  - Mengharuskan loading program secara contiguous/berdekatan
  - Metode alokasi job
    - Partisi pertama yang tersedia dan ukurannya lebih besar dari ukuran program
  - Untuk dapat bekerja dengan baik:
    - Semua job harus berukuran sama dan ukuran memori diketahui terlebih dahulu
  - Ukuran partisi yang dibuat asal-asalan berakibat buruk
    - Partisi yang terlalu kecil
      - Job-job yang terlalu besar turnaround nya menjadi semakin besar.
    - Partisi yang terlalu besa
      - Buang-buang memori: **internal fragmentation**

Dynamic Partitions

- Memori utama di partisi
  - Memori untuk job dialokasikan ketika di load
  - Satu partisi contiguous per job
- Metode alokasi job
  - **First come, first serve**
  - Keterbuangan memori relatif kecil
- Kekurangan
  - Ulitisasi memori secara penuh hanya ketika job pertama yang di load.
  - Subsequent allocation: memori terbuang
    - External fragmentation: fragments diantara blok



(figure 2.2)

Main memory use during dynamic partition allocation. Five snapshots (a-e) of main memory as eight jobs are submitted for processing and allocated space on the basis of "first-come, first-served." Job 8 has to wait (e) even though there's enough free memory between partitions to accommodate it.

Best-Fit Versus First-Fit Allocation

- Dua metode untuk alokasi free space memori:
  - **First-fit memory allocation:** partisi pertama yang memenuhi persyaratan
    - Alokasi memori menjadi cepat
  - **Best-fit memory allocation:** partisi terkecil dari yang memenuhi persyaratan
    - Lebih sedikit memori yang terbuang
    - Fragmentasi internal terkurangi, tetapi tetap masih ada

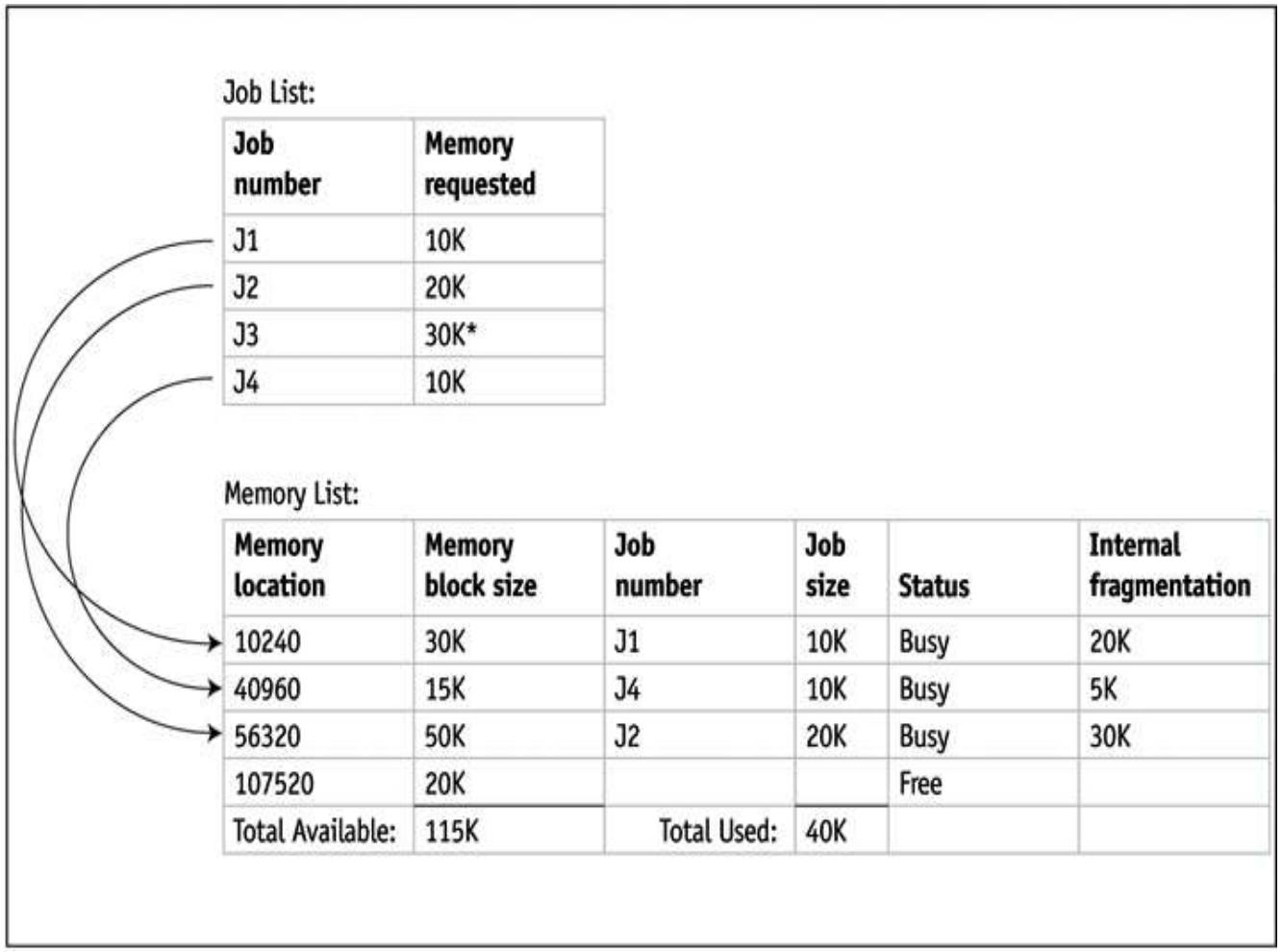
Best-Fit Versus First-Fit Allocation (cont.)

- **First-fit memory allocation**
  - Kelebihan: alokasi memori lebih cepat
  - Kekurangan: Memori yang terbuang lebih banyak
- **Best-fit memory allocation**
  - Kelebihan: memori digunakan secara efisien
  - Kelebihan: alokasi memori makan waktu lebih lama

Best-Fit Versus First-Fit Allocation (cont.)

(figure 2.3)

Using a first-fit scheme, Job 1 claims the first available space. Job 2 then claims the first partition large enough to accommodate it, but by doing so it takes the last block large enough to accommodate Job 3. Therefore, Job 3 (indicated by the asterisk) must wait until a large block becomes available, even though there's 75K of unused memory space (internal fragmentation). Notice that the memory list is ordered according to memory location.



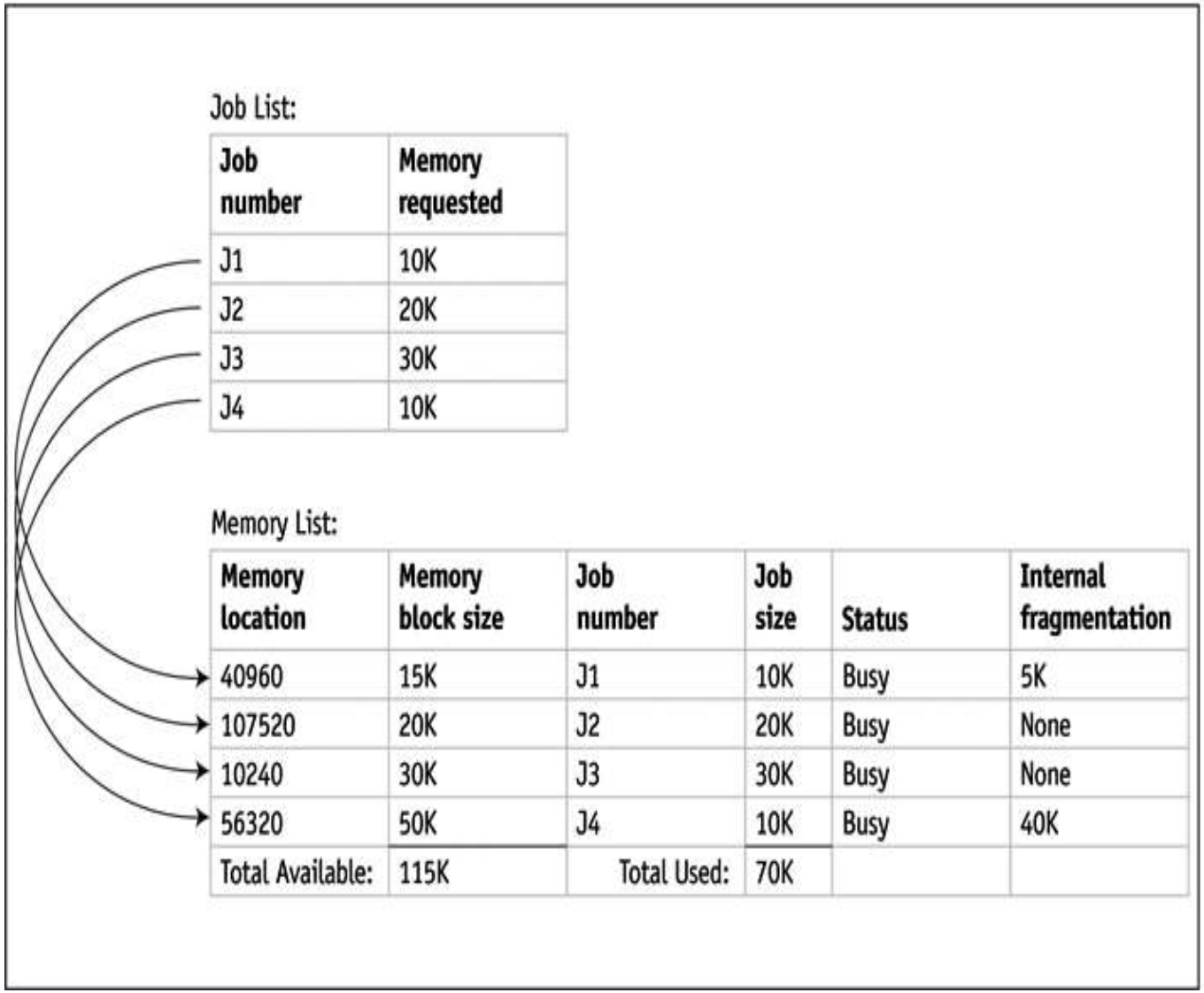
Job List:

Job number	Memory requested
J1	10K
J2	20K
J3	30K*
J4	10K

Memory List:

Memory location	Memory block size	Job number	Job size	Status	Internal fragmentation
10240	30K	J1	10K	Busy	20K
40960	15K	J4	10K	Busy	5K
56320	50K	J2	20K	Busy	30K
107520	20K			Free	
Total Available:	115K	Total Used:	40K		

Best-Fit Versus First-Fit Allocation (continued)



Job List:

Job number	Memory requested
J1	10K
J2	20K
J3	30K
J4	10K

Memory List:

Memory location	Memory block size	Job number	Job size	Status	Internal fragmentation
40960	15K	J1	10K	Busy	5K
107520	20K	J2	20K	Busy	None
10240	30K	J3	30K	Busy	None
56320	50K	J4	10K	Busy	40K
Total Available:	115K	Total Used:	70K		

(figure 2.4)

Best-fit free scheme. Job 1 is allocated to the closest-fitting free partition, as are Job 2 and Job 3. Job 4 is allocated to the only available partition although it isn't the best-fitting one. In this scheme, all four jobs are served without waiting. Notice that the memory list is ordered according to memory size. This scheme uses memory more efficiently but it's slower to implement.

Best-Fit Versus First-Fit Allocation (cont.)

- **Algoritma untuk first-fit**
  - Asumsikan memory manager memiliki dua daftar:
    - Satu untuk free memory
    - Satu lagi untuk blok memori yang terpakai
  - Loop memeriksa ukuran tiap job terhadap ukuran tiap blok
    - Hingga sebuah blok yang cukup besar untuk job tersebut ditemukan
  - Job di load ke blok memori tersebut
  - Memory Manager keluar dari loop
    - Mengambil job selanjutnya dari antrian

Best-Fit Versus First-Fit Allocation (cont.)

- **Algoritma first-fit (cont):**
  - Jika tidak satupun yang memenuhi
    - Maka job ditempatkan di antrian tunggu
    - Memory Manager mengambil job selanjutnya
  - Proses diulangi lagi

Best-Fit Versus First-Fit Allocation (continued)

(table 2.2)

These two snapshots of memory show the status of each memory block before and after a request is made using the first-fit algorithm. (Note: All values are in decimal notation unless otherwise indicated.)

Before Request		After Request	
Beginning Address	**Memory Block Size**	**Beginning Address**	**Memory Block Size**
4075	105	4075	105
5225	5	5225	5
6785	600	*6985	400
7560	20	7560	20
7600	205	7600	205
10250	4050	10250	4050
15125	230	15125	230
24500	1000	24500	1000

Best-Fit Versus First-Fit Allocation (continued)

- **Algoritma untuk best-fit**
  - Tujuan
    - Menemukan memori blok terkecil dimana job masih muat di dalamnya
  - Keseluruhan tabel diperiksa sebelum alokasi

Best-Fit Versus First-Fit Allocation (cont.)

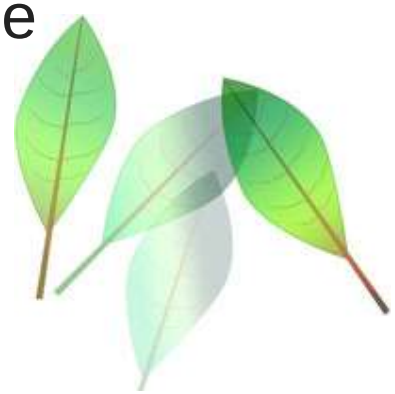
(table 2.3)

These two snapshots of memory show the status of each memory block before and after a request is made using the best-fit algorithm.

Before Request		After Request	
Beginning Address	**Memory Block Size**	**Beginning Address**	**Memory Block Size**
4075	105	4075	105
5225	5	5225	5
6785	600	6785	600
7560	20	7560	20
7600	205	*7800	5
10250	4050	10250	4050
15125	230	15125	230
24500	1000	24500	1000

Deallocation

- **Deallocation:** membebaskan ruang memori yang sudah teralokasikan
- Untuk sistem fixed-partition:
  - Cukup jelas
  - Memory Manager mereset status blok memori sebuah job untuk “membebaskan” nya setelah dipakai
  - Bisa menggunakan berbagai macam kode
  - Misal: 0 = free, 1 = terpakai



Deallocation (continued)

- Untuk sistem partisi dinamis:
  - Algoritma mencoba mengkombinasikan memori yang tidak terpakai
  - Lebih kompleks
- 3 contoh kasus partisi dinamis
  - **Kasus 1:** ketika blok yang akan didealokasi bersebelahan dengan blok lain yang tidak terpakai
  - **Kasus 2:** Ketika blok yang akan didealokasi berada diantara dua blok yang tidak terpakai
  - **Kasus 3:** Ketika blok yang akan didealokasi terisolasi dari blok lain yang tidak terpakai

Kasus 1: Menggabungkan 2 blok yang free

- Blok nya bersebelahan
- Daftar berubah untuk merefleksikan alamat awal blok tak terpakai yang baru
  - Misal: 7600 – Alamat instruksi pertama job yang baru saja membebaskan blok tersebut
- Ukuran blok memori berubah sesuai dengan yang baru
  - Gabungan dua partisi
  - Misal: (200 + 5)

Case 1: Joining Two Free Blocks (continued)

(table 2.4)

This is the original free list before deallocation for Case 1. The asterisk indicates the free memory block that's adjacent to the soon-to-be-free memory block.

Beginning Address	Memory Block Size	Status
4075	105	Free
5225	5	Free
6785	600	Free
7560	20	Free
(7600)	(200)	(Busy)[1]
*7800	5	Free
10250	4050	Free
15125	230	Free
24500	1000	Free

[1]Although the numbers in parentheses don't appear in the free list, they've been inserted here for clarity. The job size is 200 and its beginning location is 7600.

Case 1: Joining Two Free Blocks (continued)

(table 2.5)

Case 1. This is the free list after deallocation. The asterisk indicates the location where changes were made to the free memory block.

Beginning Address	Memory Block Size	Status
4075	105	Free
5225	5	Free
6785	600	Free
7560	20	Free
*7600	205	Free
10250	4050	Free
15125	230	Free
24500	1000	Free

Kasus 2: Menggabungkan 3 blok yang free

- Memori space yang terdealokasi berada diantara 2 blok yang free
- Daftar berubah untuk merefleksikan alamat awal blok baru
  - Misal: 7560, alamat awal yang terkecil
- Ukuran 3 partisi digabung
  - Misal: (20 + 20 + 205)
- Blok yang digabung (yang terakhir) diberi status “null”
  - Misal: 7600

Case 2: Joining Three Free Blocks (continued)

Beginning Address	Memory Block Size	Status
4075	105	Free
5225	5	Free
6785	600	Free
*7560	20	Free
(7580)	(20)	(Busy)[1]
*7600	205	Free
10250	4050	Free
15125	230	Free
24500	1000	Free

[1] Although the numbers in parentheses don't appear in the free list, they have been inserted here for clarity.

(table 2.6)

Case 2. This is the original free list before deallocation. The asterisks indicate the two free memory blocks that are adjacent to the soon-to-be-free memory block.

Case 2: Joining Three Free Blocks (continued)

Beginning Address	Memory Block Size	Status
4075	105	Free
5225	5	Free
6785	600	Free
7560	245	Free
*		(null entry)
10250	4050	Free
15125	230	Free
24500	1000	Free

(table 2.7)

Case 2. The free list after a job has released memory.

Kasus 3: Membebaskan blok yang terisolasi

- Memori space yang terdealokasi
  - Terisolasi dari area bebas yang lain
- Sistem menentukan status blok memori yang terelease
  - Tidak bersebelahan dengan satupun blok free
  - Diantara 2 area yang terpakai
- Sistem mencari entri ‘null’ di dalam tabel
  - Terjadi apabila ketika blok memori yang terletak diantara 2 blok memori yang terpakai dikembalikan ke free list

Case 3: Deallocating an Isolated Block (continued)

(table 2.8)

Case 3. Original free list before deallocation. The soon-to-be-free memory block is not adjacent to any blocks that are already free.

Beginning Address	Memory Block Size	Status
4075	105	Free
5225	5	Free
6785	600	Free
7560	245	Free
		(null entry)
10250	4050	Free
15125	230	Free
24500	1000	Free

Case 3: Deallocating an Isolated Block (continued)

(table 2.9)

Case 3. Busy memory list before deallocation. The job to be deallocated is of size 445 and begins at location 8805. The asterisk indicates the soon-to-be-free memory block.

Beginning Address	Memory Block Size	Status
7805	1000	Busy
*8805	445	Busy
9250	1000	Busy

Case 3: Deallocating an Isolated Block (continued)

Beginning Address	Memory Block Size	Status
7805	1000	Busy
*		(null entry)
9250	1000	Busy

(table 2.10)

Case 3. This is the busy list after the job has released its memory. The asterisk indicates the new null entry in the busy list.

Case 3: Deallocating an Isolated Block (continued)

Beginning Address	Memory Block Size	Status
4075	105	Free
5225	5	Free
6785	600	Free
7560	245	Free
*8805	445	Free
10250	4050	Free
15125	230	Free
24500	1000	Free

(table 2.11)

Case 3. This is the free list after the job has released its memory. The asterisk indicates the new free block entry replacing the null entry.

Relocatable Dynamic Partitions

- Memory Manager merelokasi program-program
  - Menggabungkan blok-blok yang kosong
- Memadatkan(*compact*) blok yang kosong
  - Membuat satu blok memori menjadi cukup besar untuk mengakomodasikan beberapa atau semua job dalam antrian

Relocatable Dynamic Partitions (continued)

- **Compaction:** mengatur kembali bagian dari memori yang terfragmentasi
  - Semua program di memori harus di relokasi
    - program ditempatkan secara urut.
  - Sistem operasi harus bisa membedakan antara nilai data dan alamat
    - Setiap alamat disesuaikan dengan lokasi program yang baru
    - Data tetap

Relocatable Dynamic Partitions (continued)

```
A       EXP 132, 144, 125, 110      ;the data values
BEGIN:  MOVEI           1,0         ;initialize register 1
        MOVEI           2,0         ;initialize register 2
LOOP:   ADD             2,A(1)      ;add (A + reg 1) to reg 2
        ADDI            1,1         ;add 1 to reg 1
        CAIG            1,4-1       ;is register 1 > 4-1?
        JUMPA           LOOP        ;if not, go to Loop
        MOVE            3,2         ;if so, move reg 2 to reg 3
        IDIVI           3,4         ;divide reg 3 by 4,
                                    ;remainder to register 4
        EXIT                        ;end
        END
```

(figure 2.5)

An assembly language program that performs a simple incremental operation. This is what the programmer submits to the assembler. The commands are shown on the left and the comments explaining each command are shown on the right after the semicolons.

Relocatable Dynamic Partitions (continued)

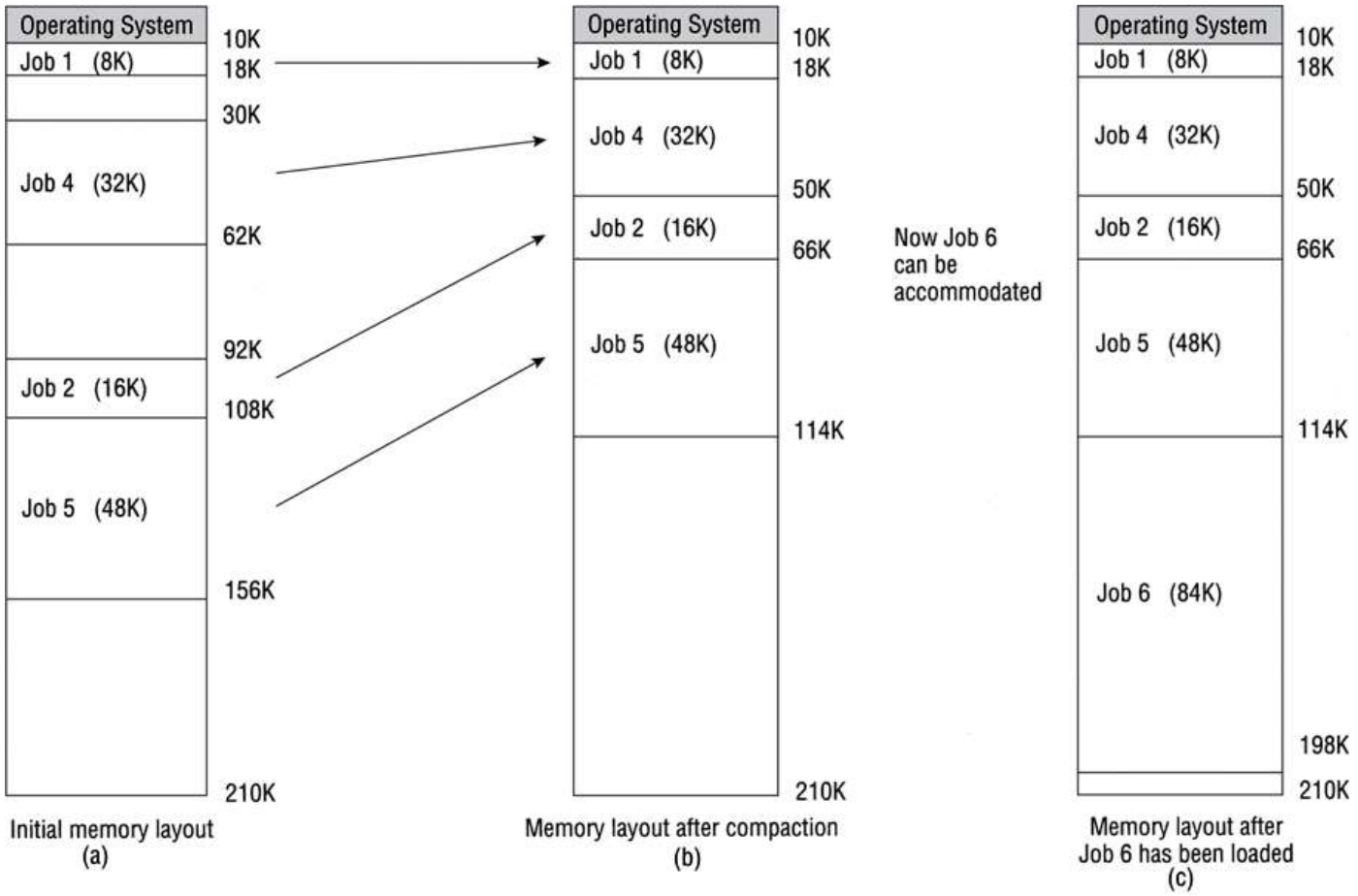
```
000000' 000000 000132          A:       EXP132,144,125,110
000001' 000000 000144
000002' 000000 000125
000003' 000000 000110

000004' 201 01 0 00 000000     BEGIN:   MOVEI    1,0
000005' 201 02 0 00 000000              MOVEI    2,0
000006' 270 02 0 01 000000'    LOOP:    ADD      2,A(1)
000007' 271 01 0 00 000001              ADDI     1,1
000008' 307 01 0 00 000003              CAIG     1,4-1
000009' 324 00 0 00 000006'             JUMPA    LOOP
000010' 200 03 0 00 000002              MOVE     3,2
000011' 231 03 0 00 000004              IDIVI    3,4
000012' 047 00 0 00 000012              EXIT

                  000000                END
```

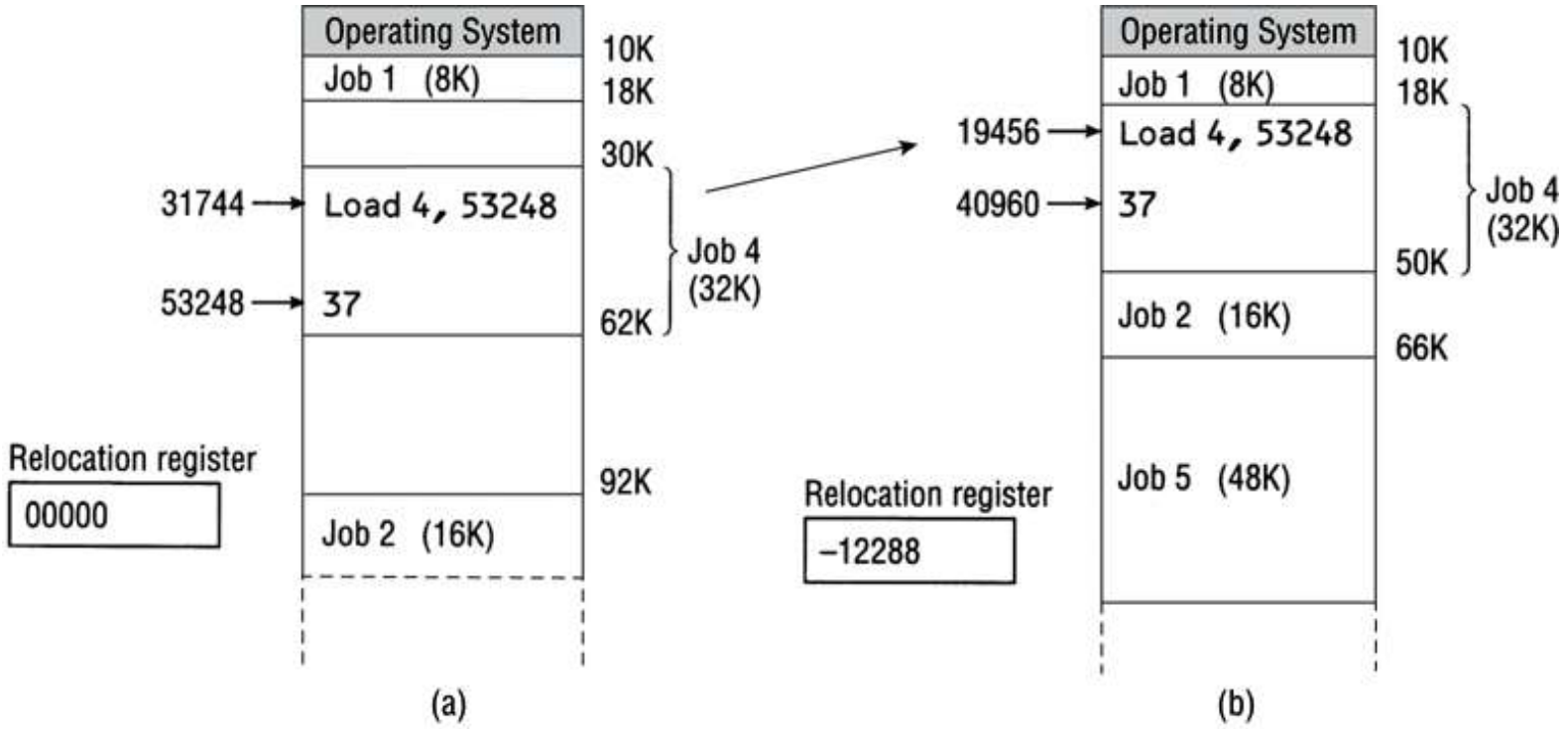
(figure 2.6)

The original assembly language program after it has been processed by the assembler. To run the program, the assembler translates it into machine-readable code (shown on the left) with all addresses marked by a special symbol (shown here as an apostrophe) to distinguish addresses from data values.



(figure 2.7)

Three snapshots of memory before and after compaction. When Job 6 arrives requiring 84K, the initial memory layout in (a) shows external fragmentation totaling 96K of space. Immediately after compaction (b), external fragmentation has been eliminated, making room for Job 6 (c).



(figure 2.8)

Contents of relocation register and close-up of Job 4 memory area (a) before relocation and (b) after relocation and compaction.

Relocatable Dynamic Partitions (cont.)

- Masalah Compaction:
  - Apa yang terjadi dibelakang layar saat relocation dan compaction terjadi?
  - Apa yang melacak seberapa jauh setiap job berpindah dari tempat asalnya?
  - Daftar apa yang harus di update?

Relocatable Dynamic Partitions (cont.)

- **Daftar apa yang harus di update?**
  - Daftar blok memori yang tak terpakai / Free list
    - Harus selalu menunjukkan partisi blok memori tak terpakai yang terbaru
  - Daftar blok memori yang terpakai / Busy list
    - Harus selalu menunjukkan lokasi terbaru jobjob yang sudah dieksekusi
  - Setiap job memiliki alamat baru
    - Perkecualian: kalau lokasinya sudah yang paling rendah

Relocatable Dynamic Partitions (continued)

- Ada register yang dikhususkan untuk relokasi:
  - **Bounds register**
    - Menyimpan lokasi memori tertinggi yang digunakan oleh program
  - **Relocation register**
    - Berisi nilai yang harus ditambahkan ke setiap alamat yang dirujuk dalam program
    - Harus dapat mengakses alamat memori yang benar setelah relokasi
    - Kalau program tidak di relokasi, register ini berisi “zero” value

Relocatable Dynamic Partitions (cont)

- Compacting dan relocating mengoptimasi penggunaan memori
  - meningkatkan *throughput*
- Opsi *timing* untuk *compaction*:
  - Ketika persentase memori yang ‘busy’ mencapai angka tertentu
  - Ketika ada job yang antri untuk dieksekesi
  - Setelah periode waktu tertentu / tergantung setting
- Compaction memiliki konsekuensi ada lebih banyak overhead
- **Goal:** mengoptimasikan waktu pemrosesan dan penggunaan memori sembari menjaga overhead se rendah mungkin

Summary

- 4 teknik manajemen memori
  - Single-user systems, fixed partitions, dynamic partitions, dan relocatable dynamic partitions
- Persyaratan umum 4 teknik manajemen memori
  - Seluruh program disimpan di memori
  - Penyimpanan yang urut / contiguous
  - Memori dipakai/terpakai sampai job selesai
- Keempatnya memiliki batasan dalam ukuran job
- Cukup baik untuk 3 generasi pertama komputer

Summary (continued)

- Trend manajemen memori moderen di akhir 1960an dan awal 1970an
  - Didiskusikan pertemuan mendatang
  - Karakteristik umum skema memori
    - Program tidak disimpan di lokasi memori secara berurutan/bersebelahan/contiguous
    - Tidak semua segmen berada di memori ketika job di eksekusi