Lý thuyết hệ điều hành

Giảng viên: TS. Hà Chí Trung

Bộ môn: Khoa học máy tính

Khoa: Công nghệ thông tin

Học viện Kỹ thuật quân sự

Email: hct2009@yahoo.com

Mobile: 01685.582.102

Chương 3. Quản lý bộ nhớ

3.1. Tổng quan về quản lý bộ nhớ

3.2. Kỹ thuật cấp phát bộ nhớ

3.3. Kỹ thuật chống phân mảnh (compaction)

3.4. Tái định vị (Relocation)

3.5. Các dạng cấu trúc chương trình

Chương 3. Quản lý bộ nhớ

3.1. Tổng quan về quản lý bộ nhớ

- **Bộ phận quản lý bộ nhớ** (Memory Management Unit – **MMU**):
  - **Tái định vị** (Relocation): Để một chương trình có thể nạp vào bất cứ vị trí nào trong bộ nhớ (với mã máy khả tái định vị - relocatable object code);
  - **Bảo vệ bộ nhớ** (Protection): các tiến trình khác không xâm phạm vào vùng nhớ dành cho tiến trình khác;
  - **Chia sẻ bộ nhớ** (Sharing);
  - **Tổ chức bộ nhớ logic** (Logical organization);
  - **Tổ chức bộ nhớ vật lý** (Physical organization);

Chương 3. Quản lý bộ nhớ

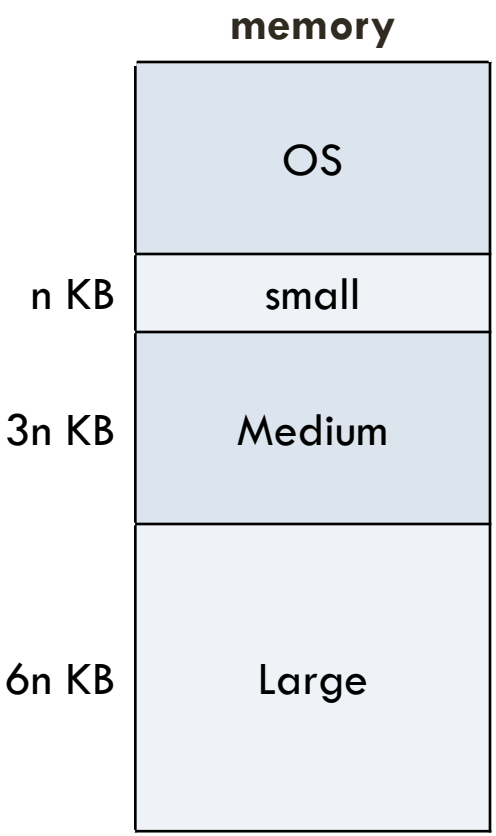
3.2. Kỹ thuật cấp phát bộ nhớ

3.2.1. Kỹ thuật phân vùng cố định (Fixed Partitioning)

3.2.2. Phân vùng cố định kết hợp Swapping

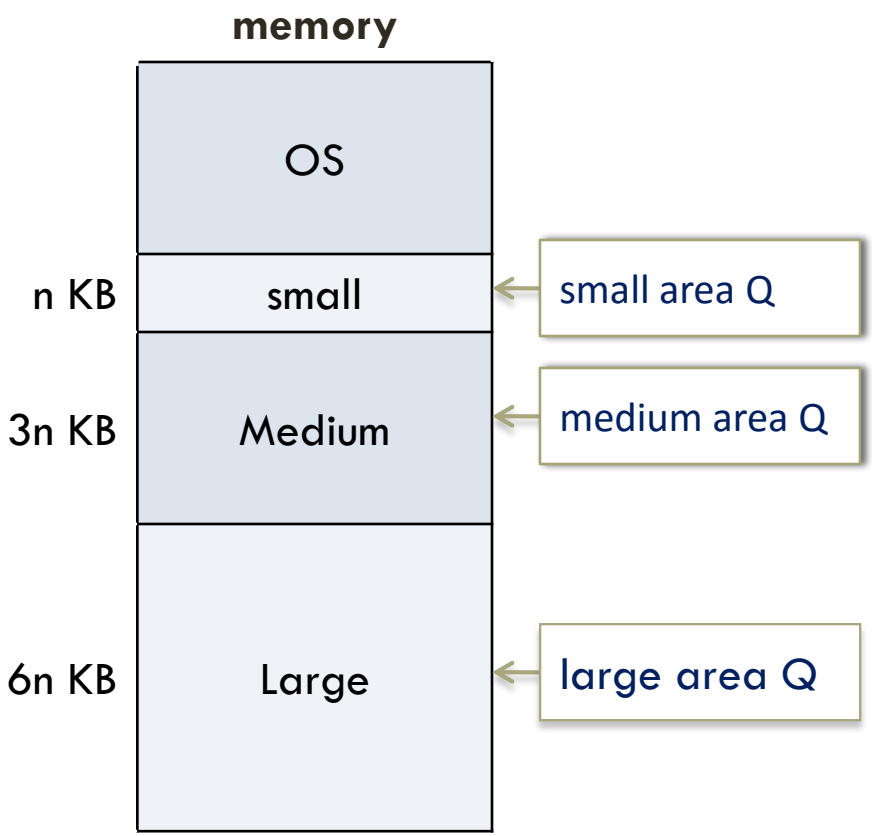
3.2.3. Phân vùng động (Dynamic Partitioning)

3.2.1. Kỹ thuật phân vùng cố định



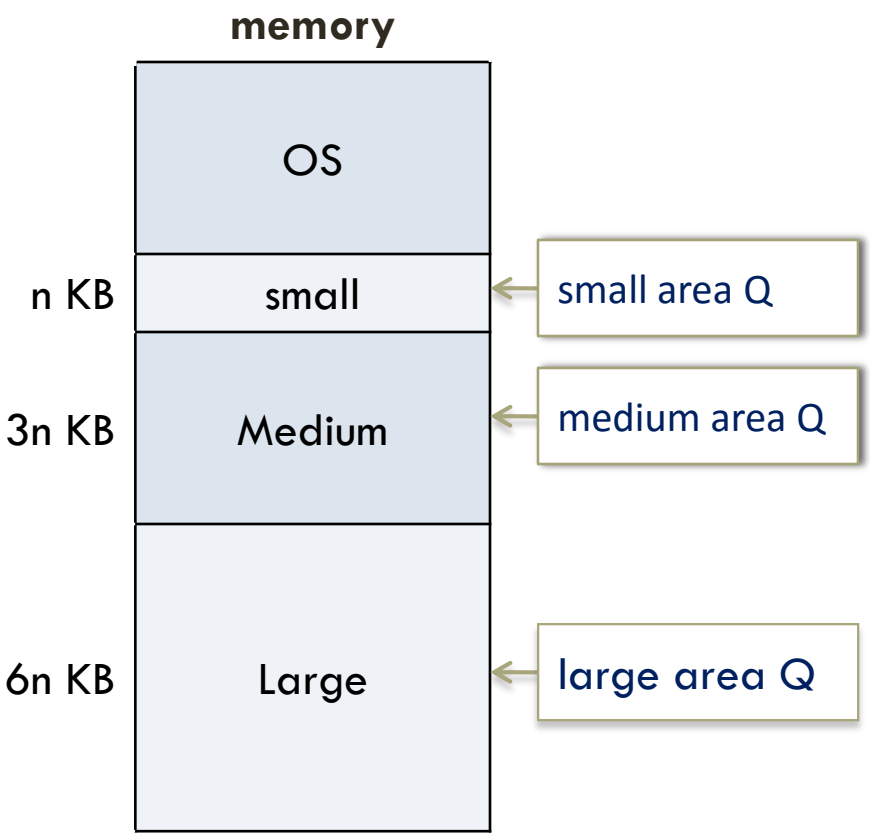
- Trong phương pháp này, bộ nhớ được chia ra thành các phân vùng có kích cỡ cố định;
- Hệ điều hành được nạp vào những vùng nhớ đầu tiên;
- Không cần thiết tái định vị các tiến trình.

3.2.1. Kỹ thuật phân vùng cố định



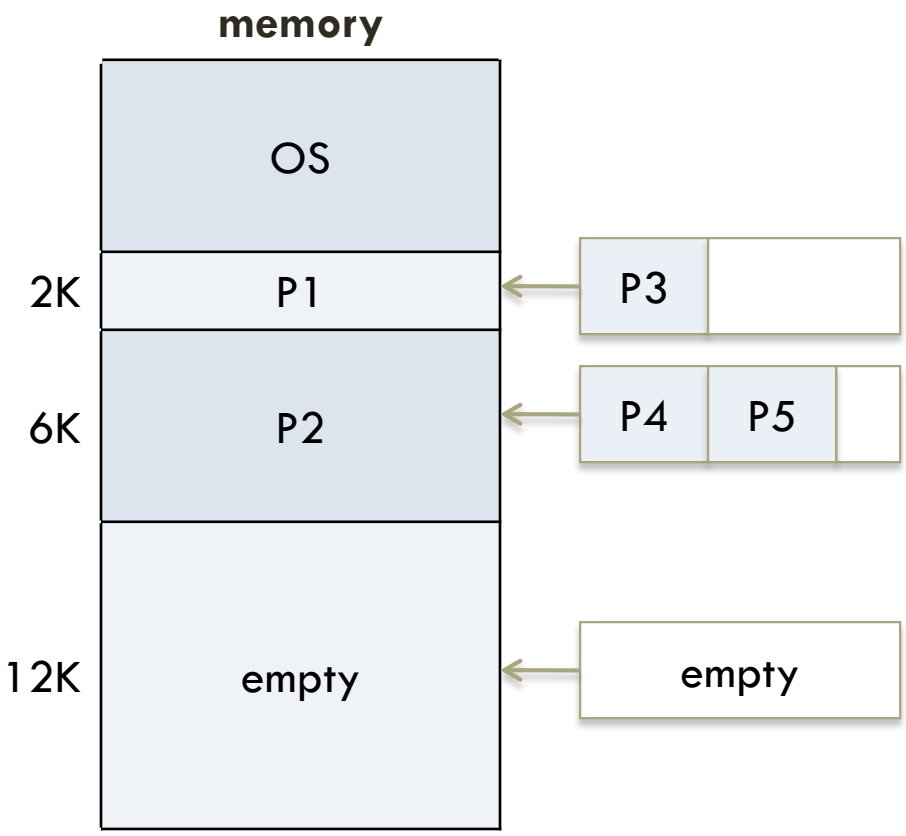
- Tiến trình được phân loại để đưa vào hệ thống theo yêu cầu bộ nhớ của chúng;
- Như vậy với mỗi phân lớp cần có hàng đợi riêng (*Process Queue - PQ)*;
- Mỗi hàng đợi có vùng nhớ riêng -> không có sự tranh chấp bộ nhớ giữa các hàng đợi.

3.2.1. Kỹ thuật phân vùng cố định



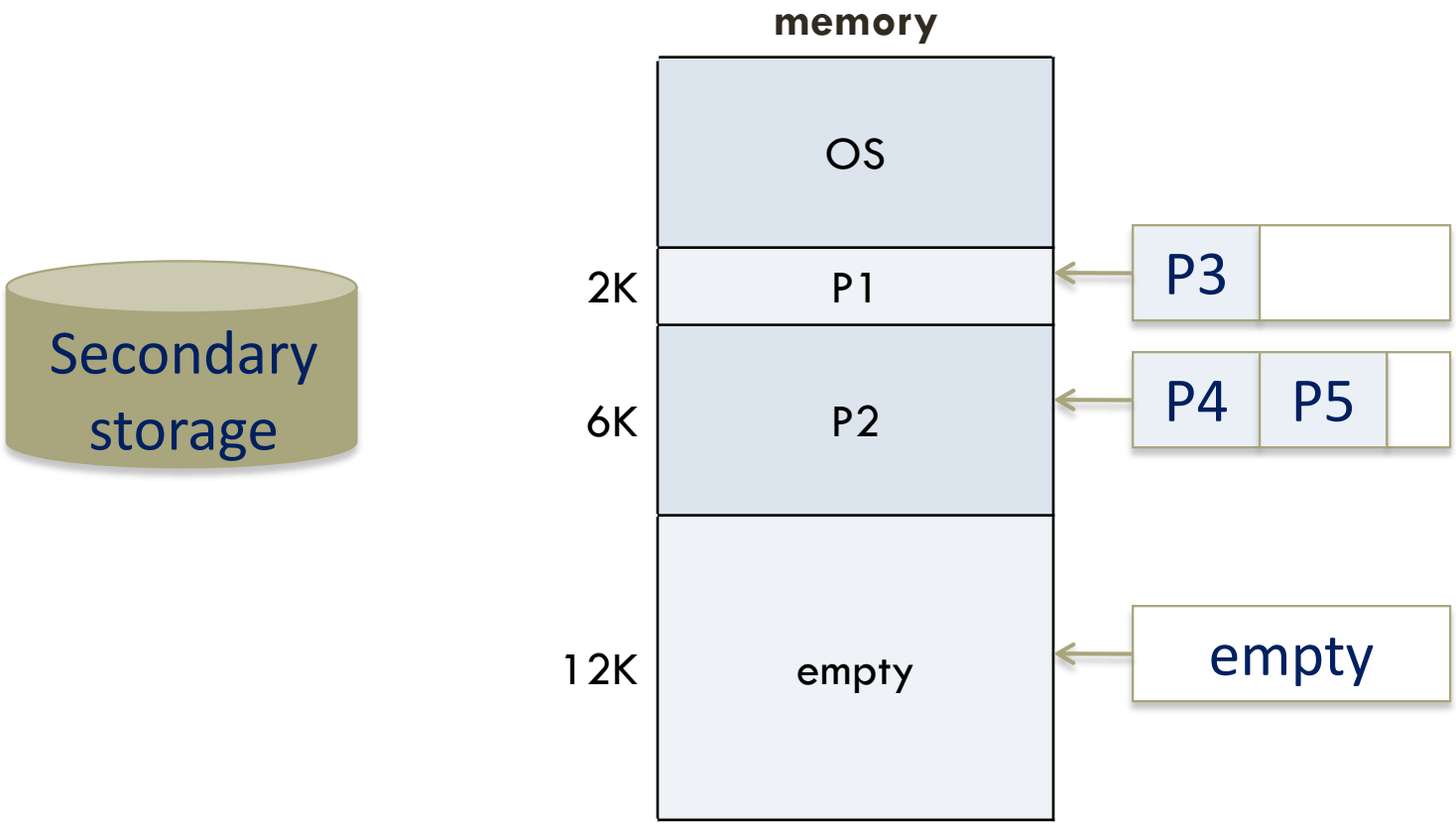
- Nếu một tiến trình được lựa chọn để cấp phát bộ nhớ, nó chiếm dụng bộ nhớ và chờ chiếm dụng processor;
- Số lượng phân vùng quy định mức độ song song hóa các tiến trình;
- Vấn đề chính: làm thế nào để xác định ***số phân vùng*** và ***kích cỡ*** của chúng?

3.2.2. Phân vùng cố định kết hợp Swapping

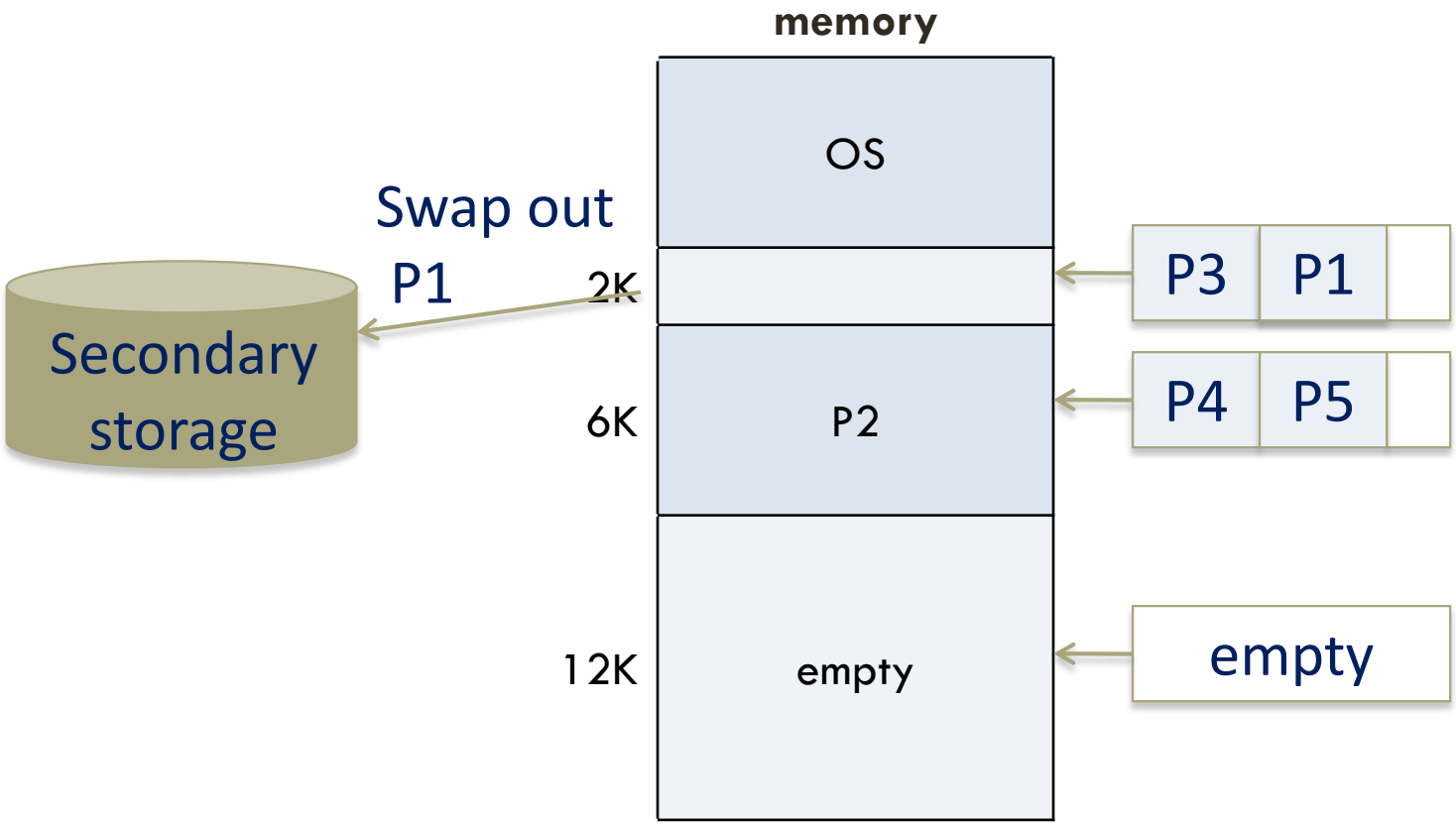


- Phân vùng cố định kết hợp với lượng tử thời gian (time quantum);
- Khi thời gian dành cho một tiến trình hết, nó được đưa ra bộ nhớ phụ (trên đĩa), tiến trình kế tiếp được nạp vào bộ nhớ.

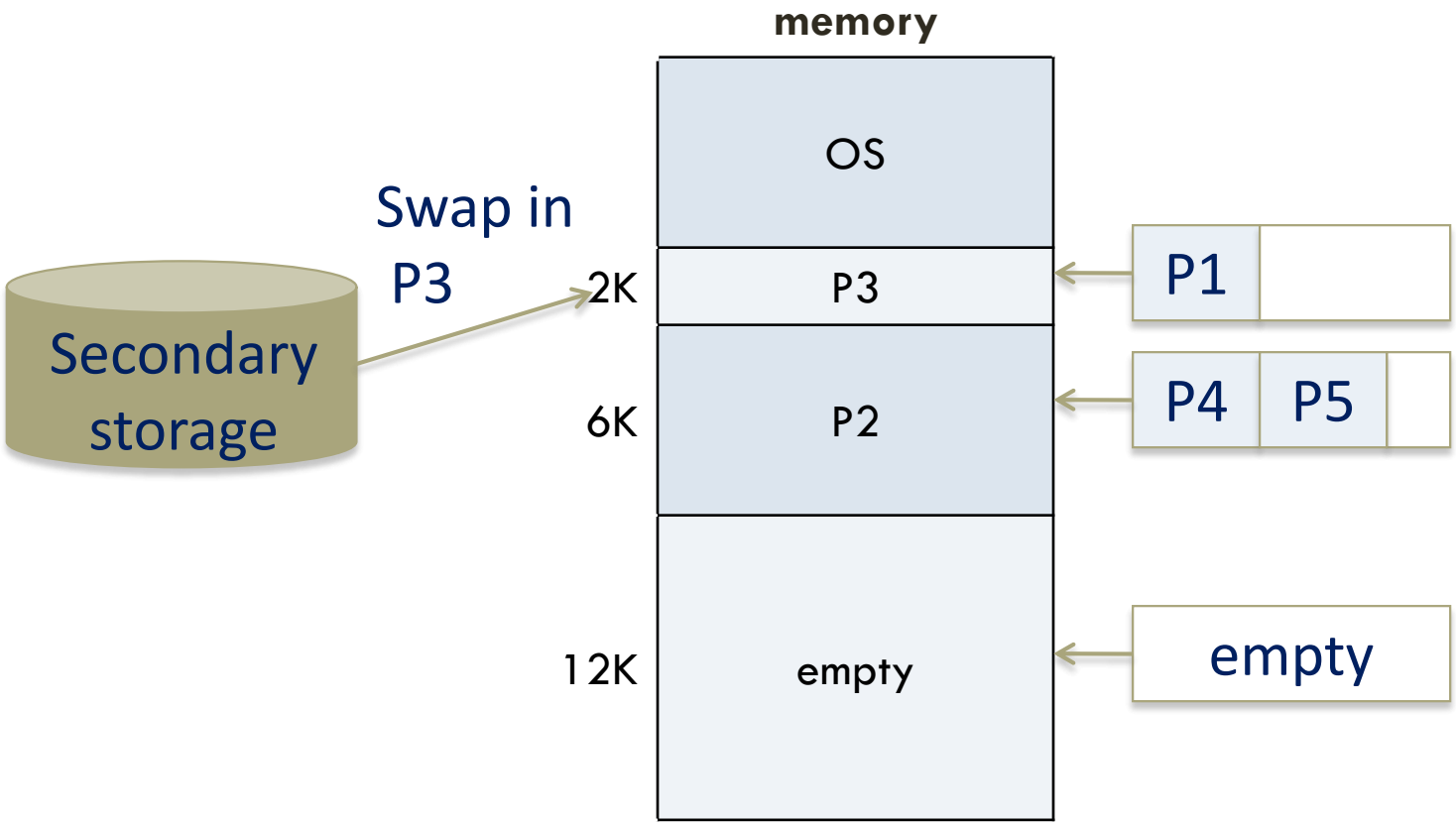
3.2.2. Phân vùng cố định kết hợp Swapping



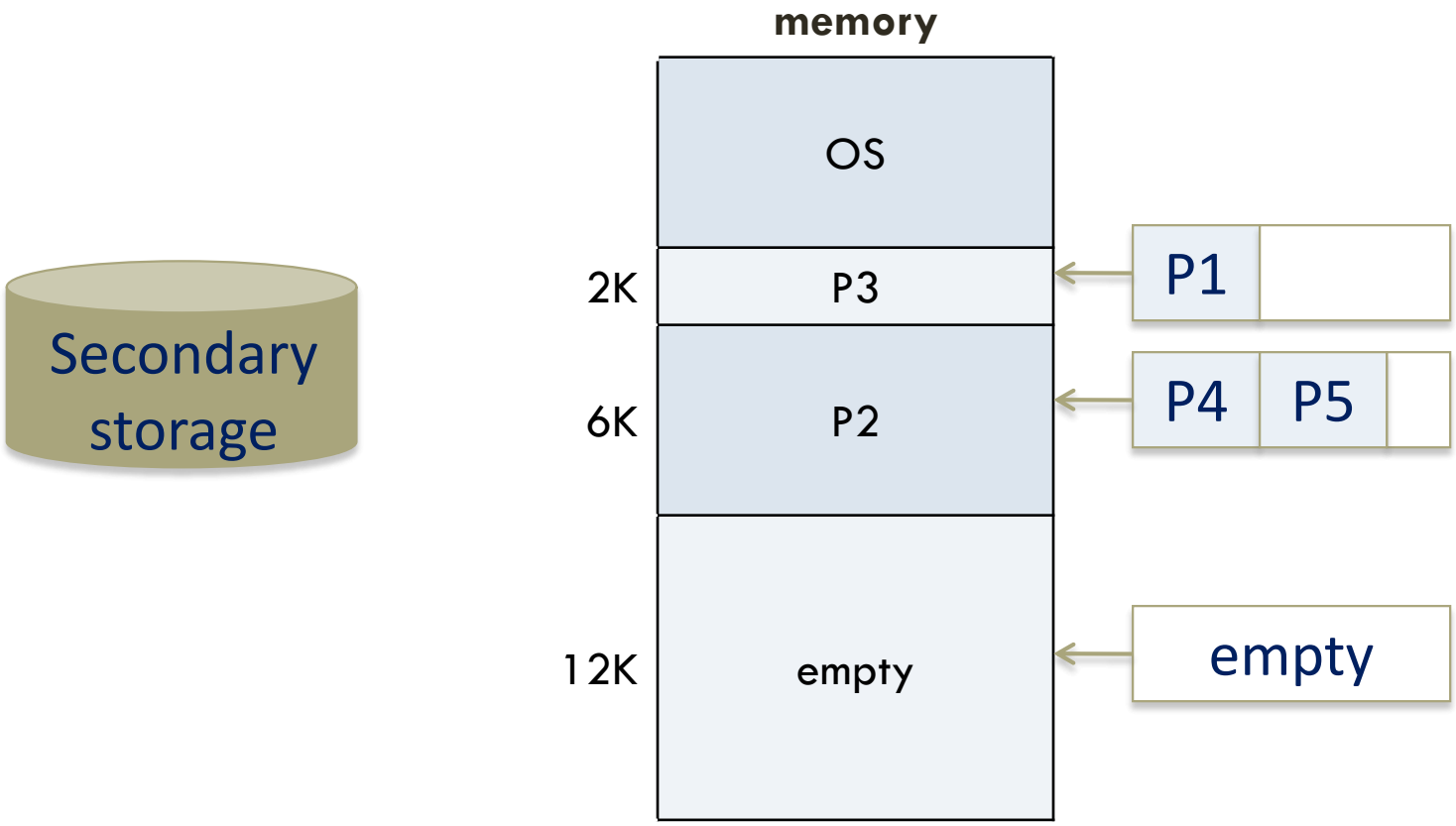
3.2.2. Phân vùng cố định kết hợp Swapping



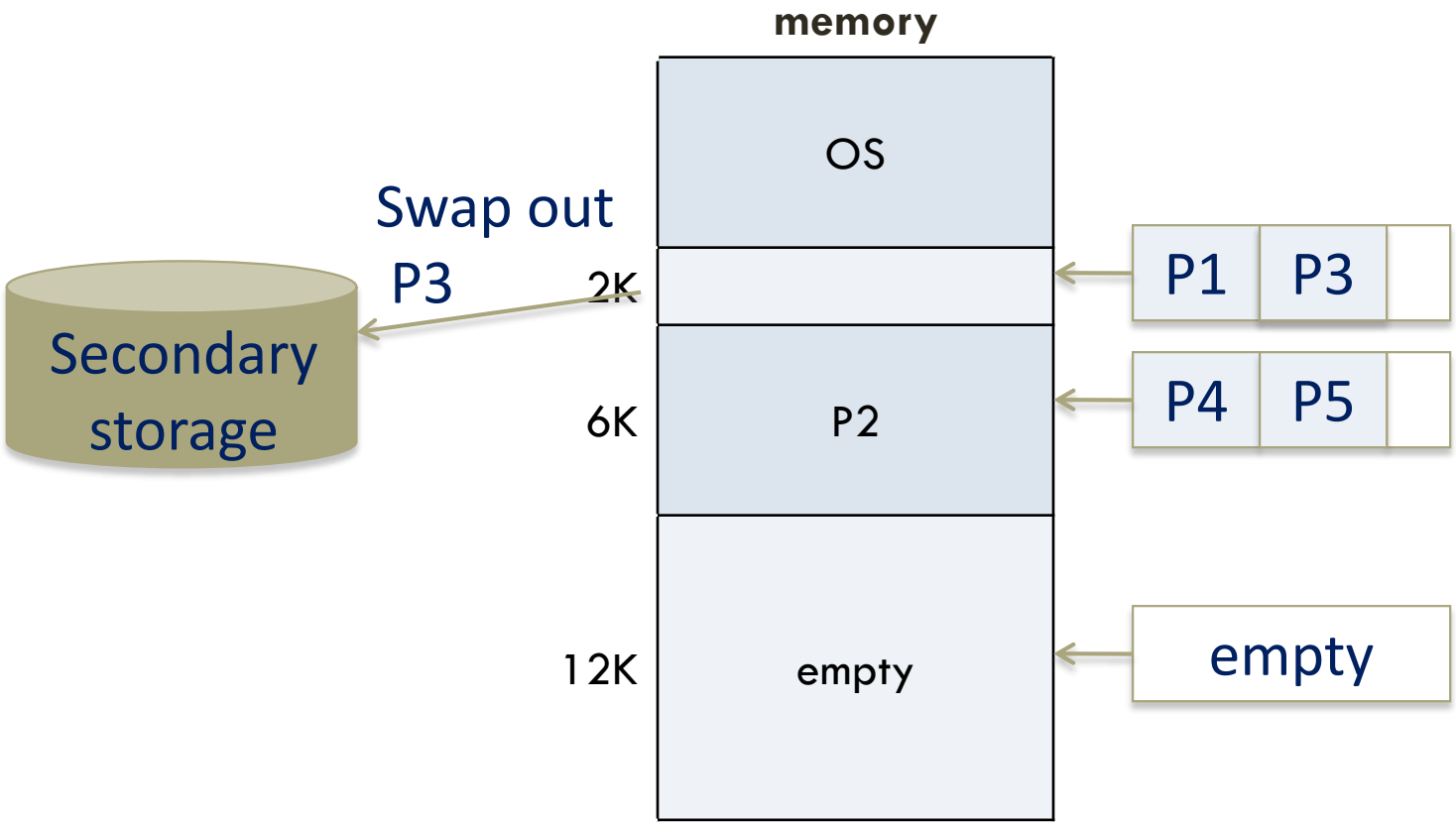
3.2.2. Phân vùng cố định kết hợp Swapping



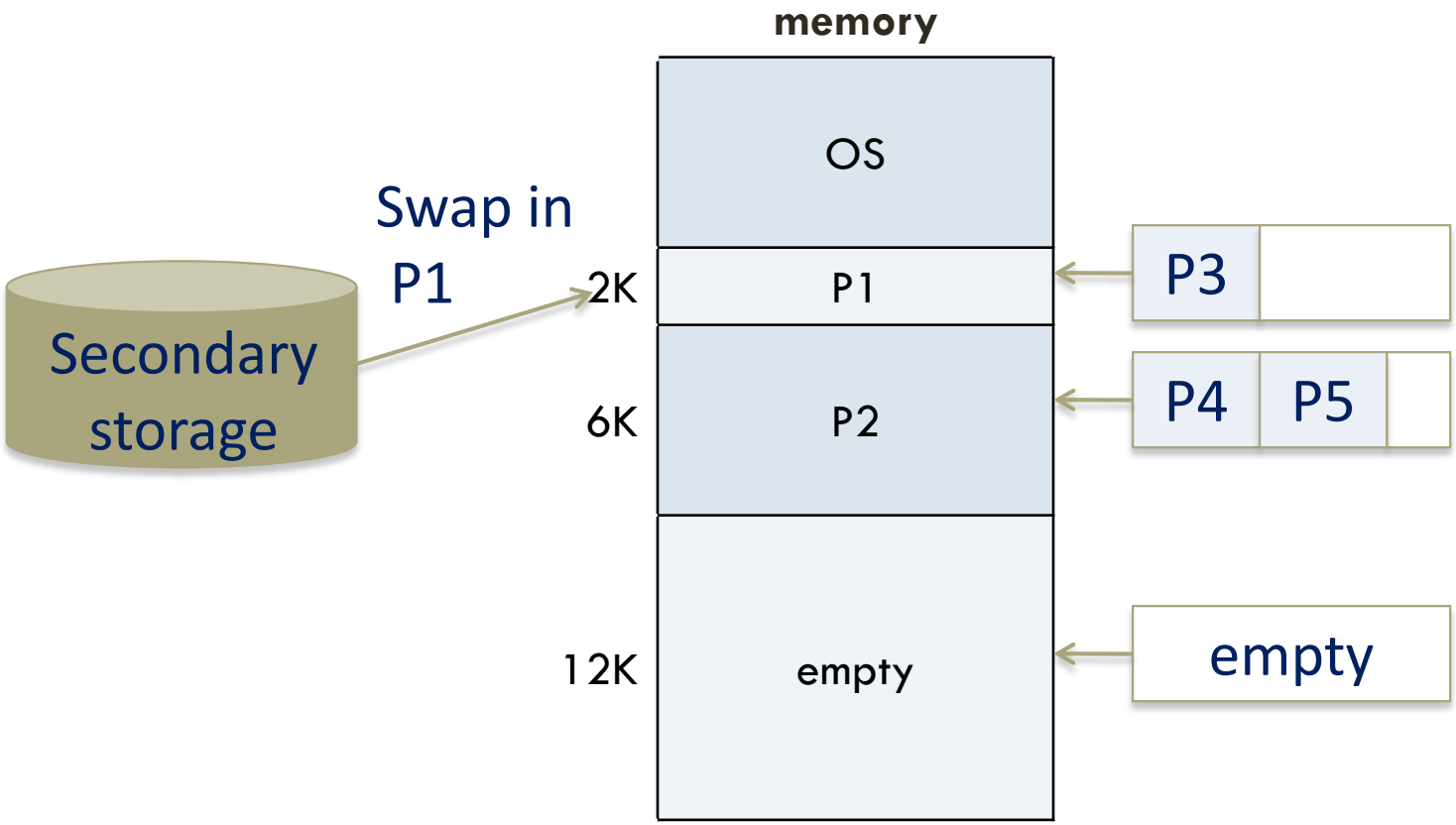
3.2.2. Phân vùng cố định kết hợp Swapping



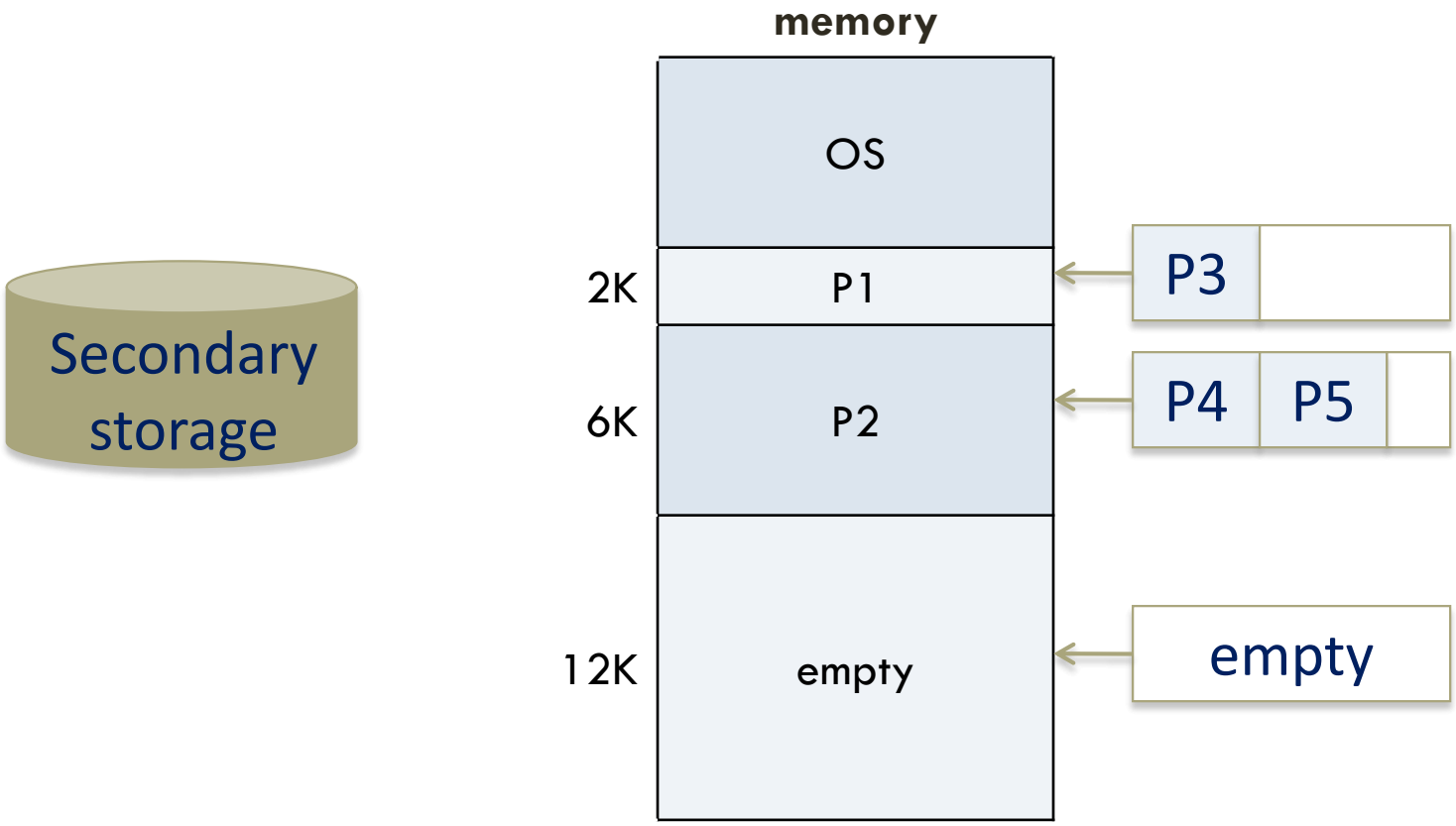
3.2.2. Phân vùng cố định kết hợp Swapping



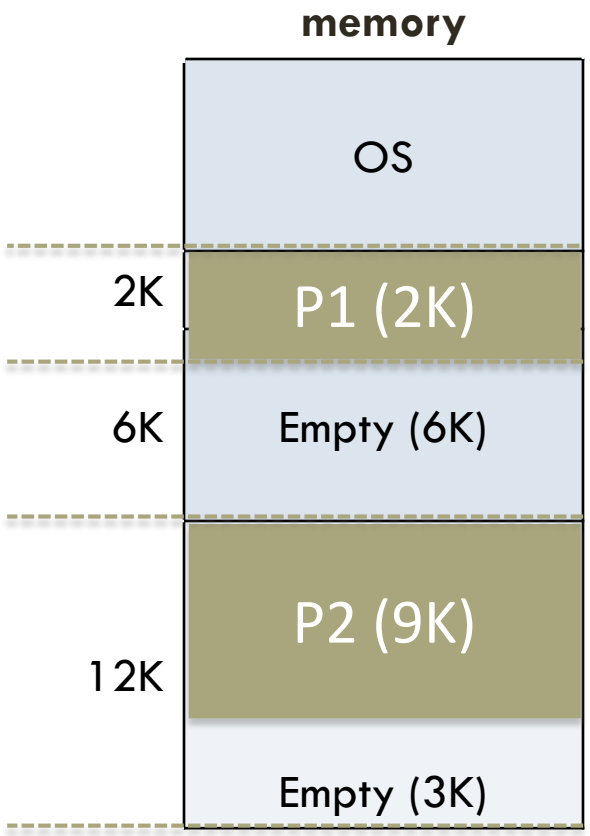
3.2.2. Phân vùng cố định kết hợp Swapping



3.2.2. Phân vùng cố định kết hợp Swapping



3.2.2. Phân vùng cố định kết hợp Swapping



Nếu toàn bộ một phân vùng không sử dụng -> ***(external fragmentation).***

Nếu tiến trình chỉ cần một phần của phân vùng -> ***(internal fragmentation).***

3.2.3. Phân vùng động

- Với phân vùng cố định, chúng ta phải đối phó với vấn đề xác định số lượng và kích thước các phân vùng để giảm thiểu sự phân mảnh nội bộ và bên ngoài.
- Ta có thể sử dụng phân vùng động (**Variable (Dynamic) Partitioning**), kích cỡ phân vùng có thể thay đổi tự động.
- Trong phương pháp này, cần có một bảng (danh sách liên kết) quản lý vùng sử dụng / chưa sử dụng bộ nhớ.

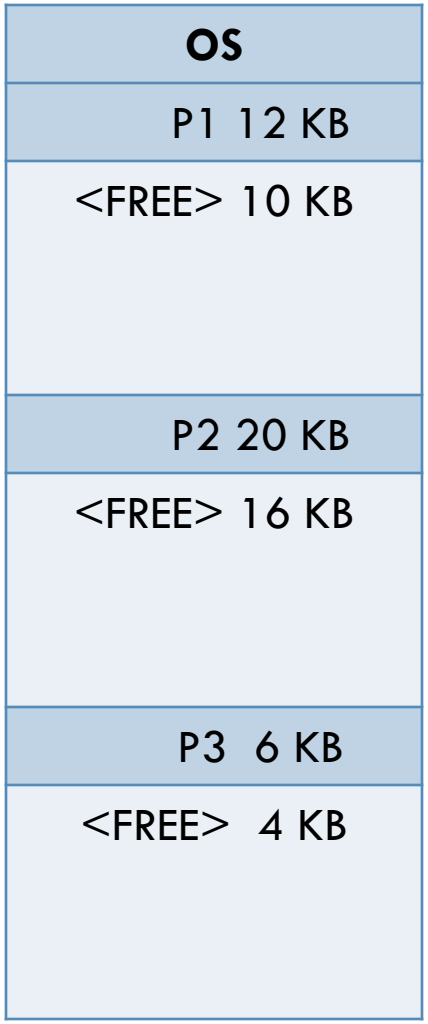
3.2.3. Phân vùng động

- Ban đầu, coi toàn bộ bộ nhớ là một vùng lớn chưa sử dụng;
- Khi có một tiến trình vào hệ thống, HĐH tìm một vùng nhớ chưa sử dụng vừa đủ cho tiến trình này;
- Khi vùng nhớ đó được giải phóng, HĐH cố gắng kết hợp nó với các vùng trống kề bên;
- Có 3 giải thuật cơ bản để tìm ra các vùng trống cho một yêu cầu bộ nhớ cụ thể:
  - First Fit
  - Best Fit
  - Worst Fit

3.2.3. Phân vùng động

First Fit

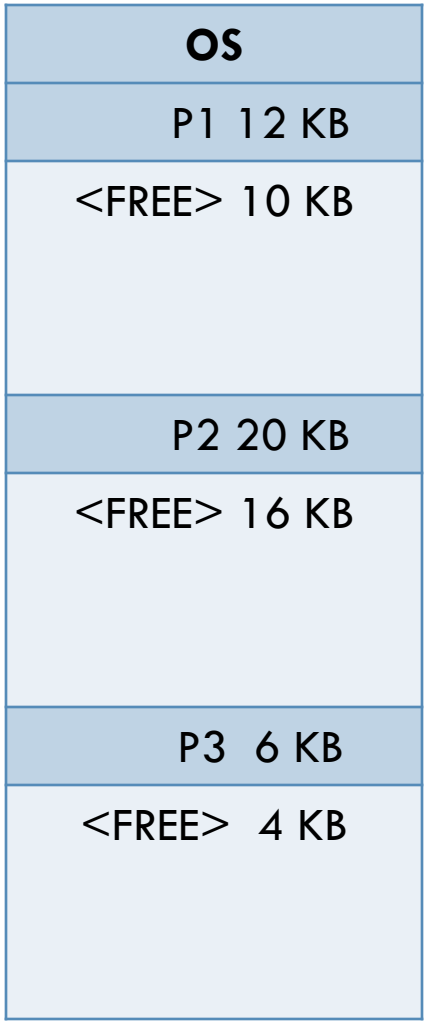
Tình trạng hiện hành của bộ nhớ



3.2.3. Phân vùng động

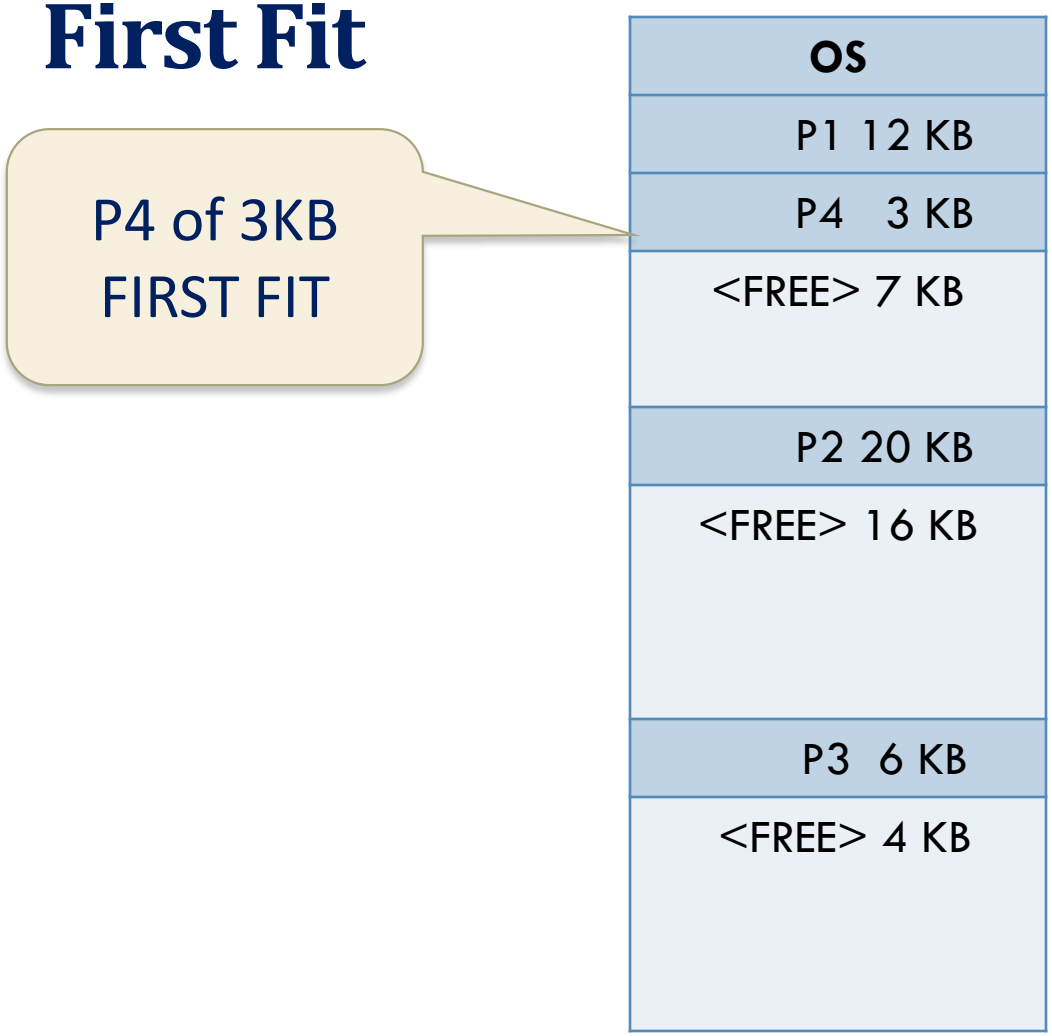
First Fit

P4 of 3KB arrives



3.2.3. Phân vùng động

First Fit



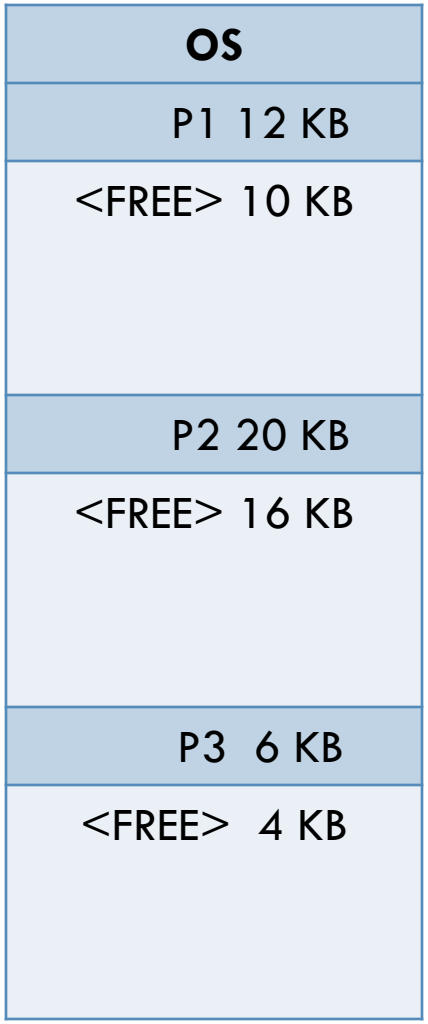
3.2.3. Phân vùng động

- **Best Fit:** Cấp phát vùng nhỏ nhất trong số các vùng đủ lớn cho tiến trình.
  - HĐH tìm ra một danh sách có kích cỡ lớn hơn yêu cầu hoặc lưu trữ một danh sách đã sắp xếp của các phân vùng;
  - Thuật toán giúp hạn chế tối đa phân mảnh nội vi, tuy nhiên đòi hỏi nhiều thời gian cho việc tìm kiếm và sắp xếp;
  - Nếu sử dụng sắp xếp, thì việc kết hợp các vùng nhớ trống kế tiếp trở nên phức tạp hơn.

3.2.3. Phân vùng động

Best Fit

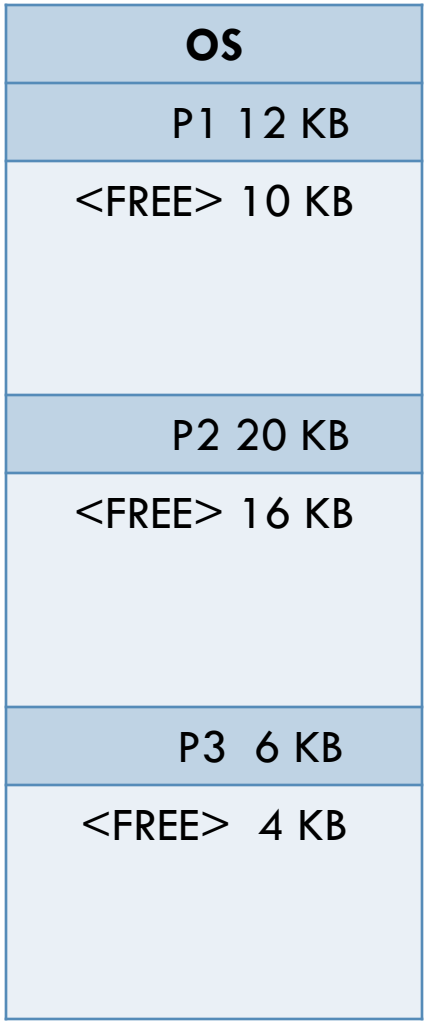
Tình trạng hiện hành của bộ nhớ



3.2.3. Phân vùng động

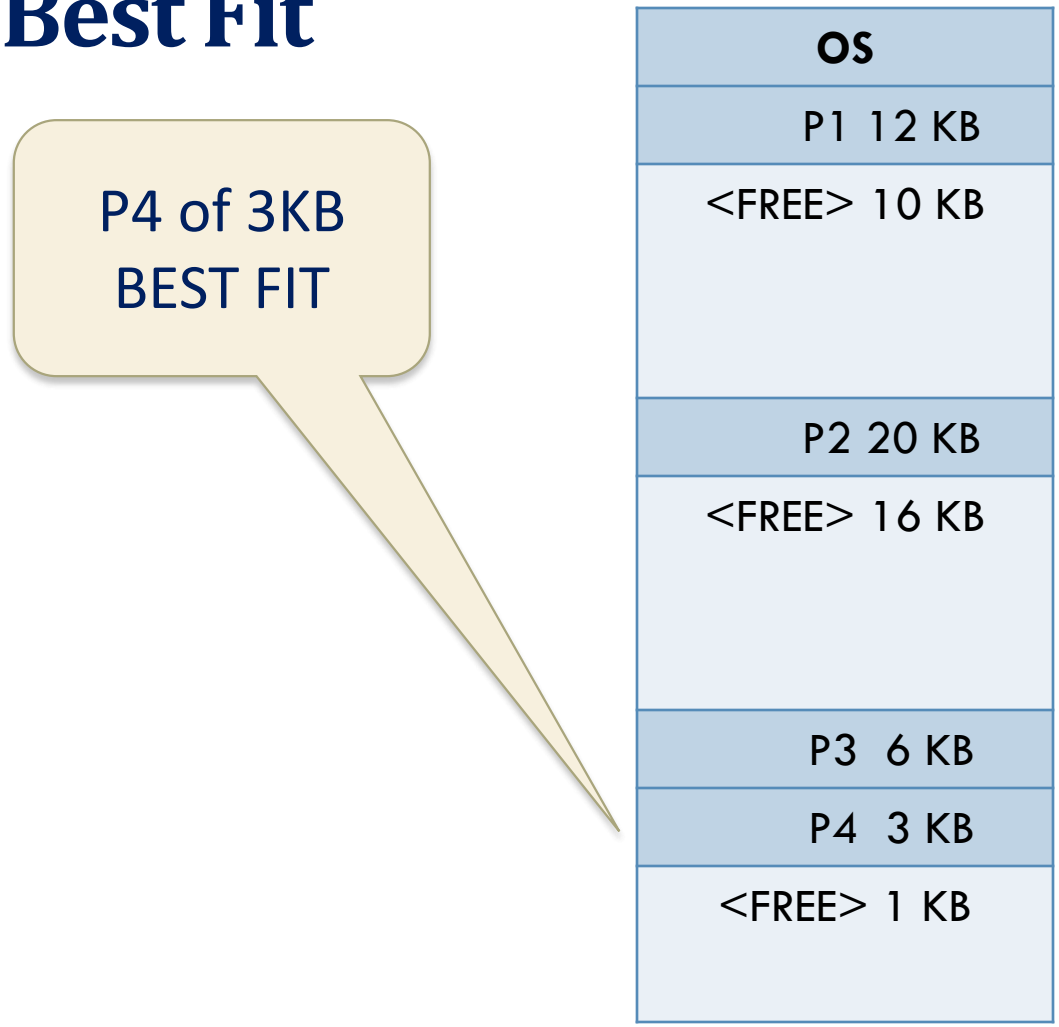
Best Fit

P4 of 3KB arrives



3.2.3. Phân vùng động

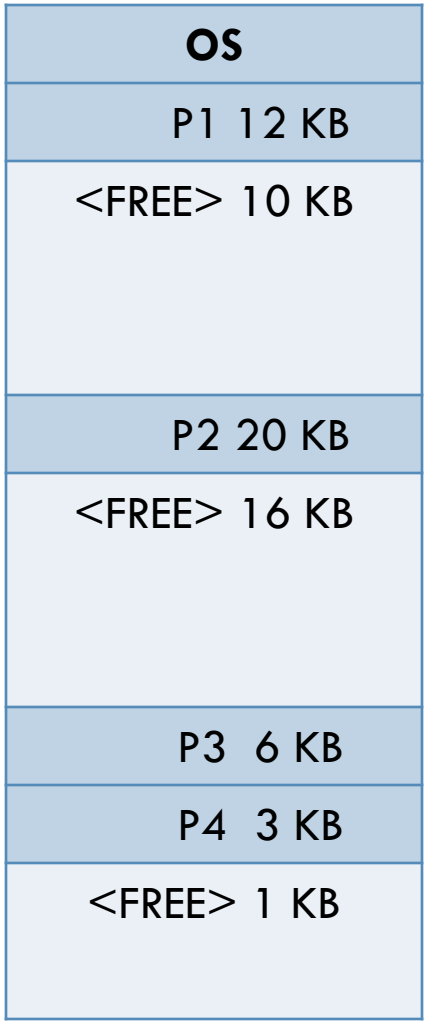
Best Fit



3.2.3. Phân vùng động

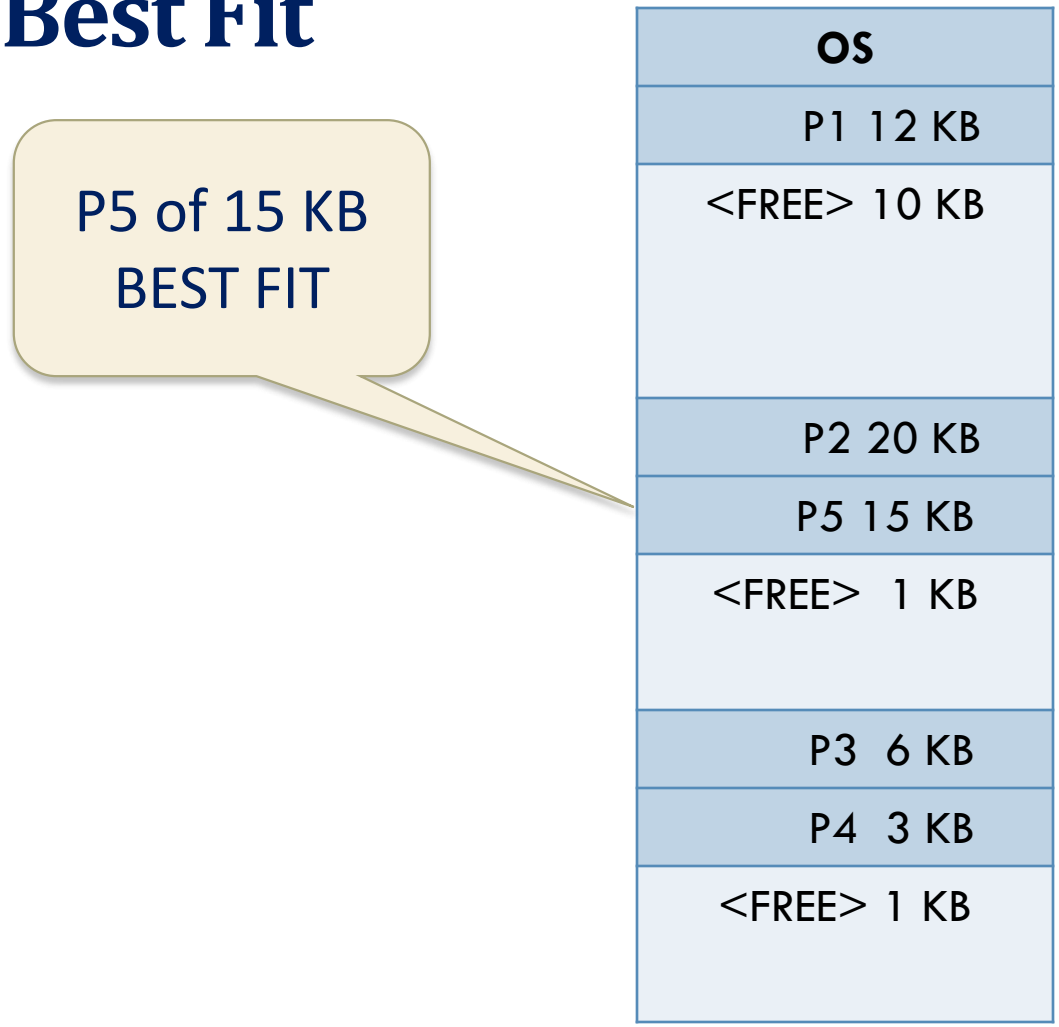
Best Fit

P5 of 15KB arrives



3.2.3. Phân vùng động

Best Fit



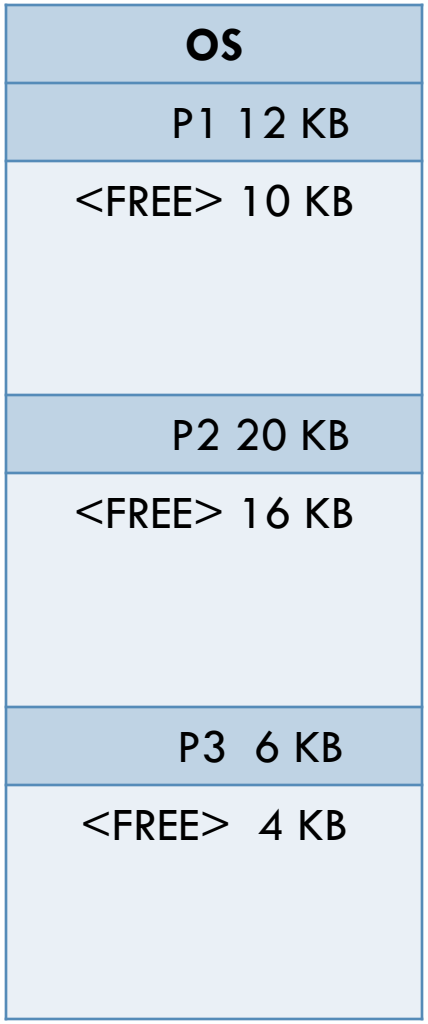
3.2.3. Phân vùng động

- **Worst Fit:** Cấp phát phân vùng còn trống có dung lượng lớn nhất.
  - Đòi hỏi tìm kiếm toàn bộ và/hoặc sắp xếp.
  - Gây lãng phí bộ nhớ do phân mảnh ngoại vi.

3.2.3. Phân vùng động

Worst Fit

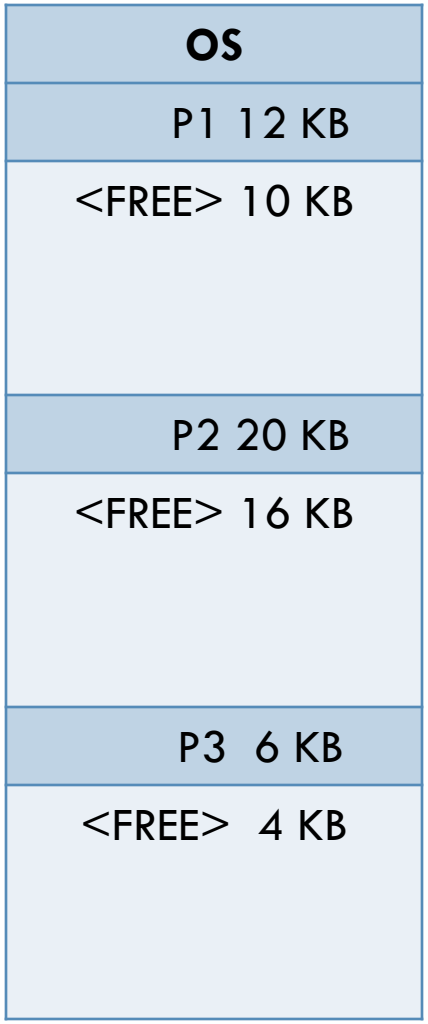
Tình trạng hiện hành của bộ nhớ



3.2.3. Phân vùng động

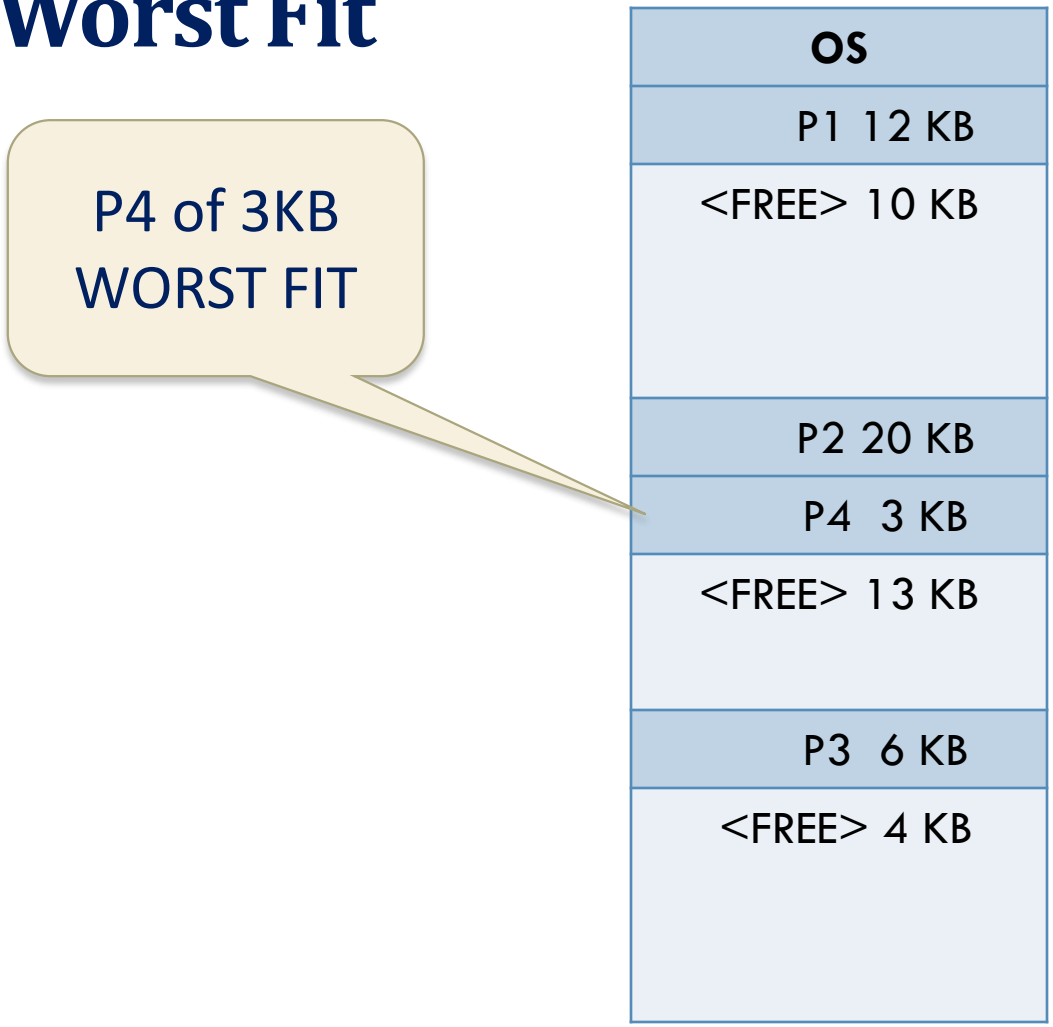
Worst Fit

P4 of 3KB arrives



3.2.3. Phân vùng động

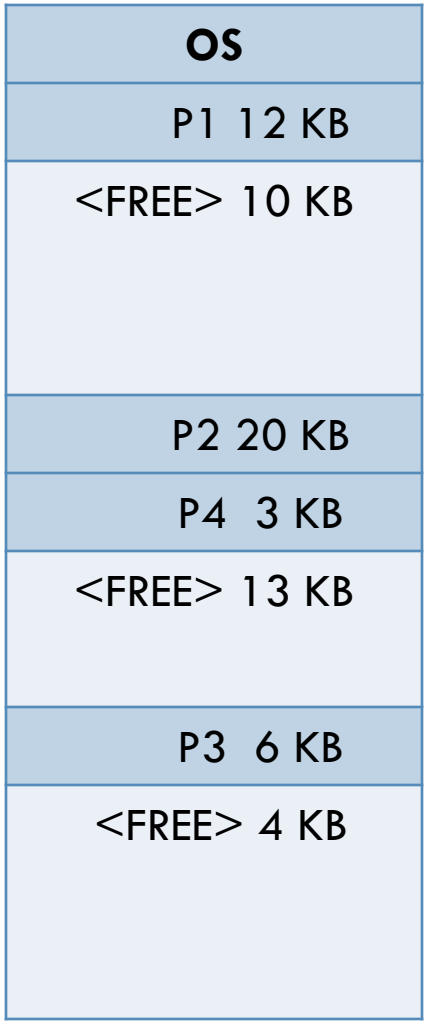
Worst Fit



3.2.3. Phân vùng động

Worst Fit

Không còn chỗ cho P5 - 15K

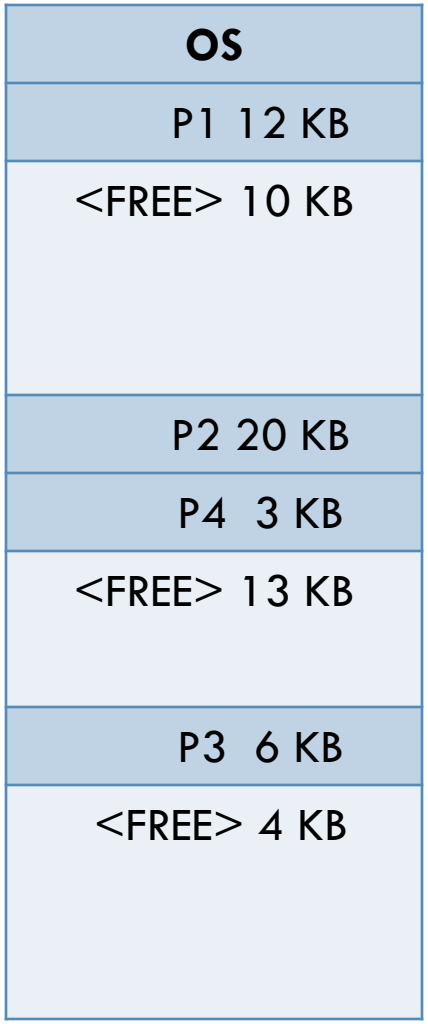


3.2.3. Phân vùng động

Worst Fit

Không còn chỗ cho P5 - 15K

Cần chống phân mảnh!!



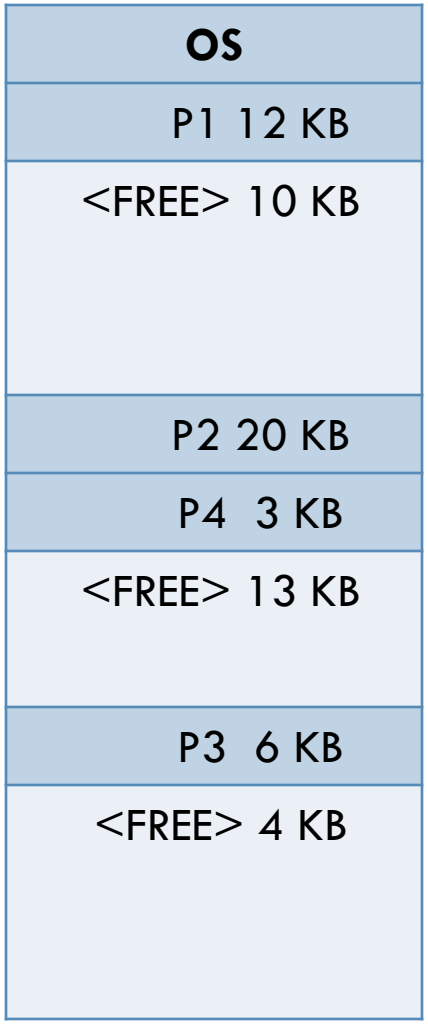
Chương 3. Quản lý bộ nhớ

3.3. Chống phân mảnh (compaction)

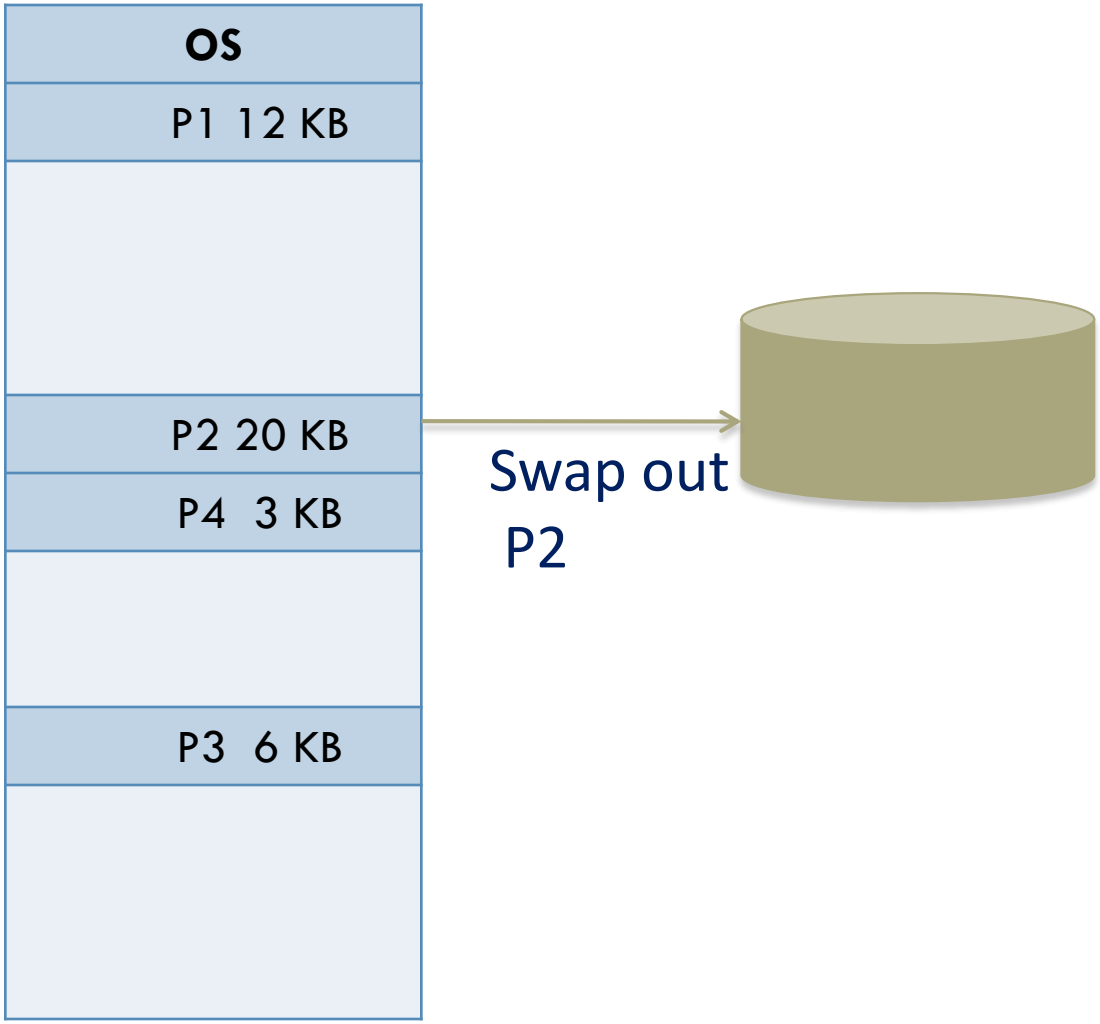
- Chống phân mảnh giúp xử lý vấn đề phân mảnh ngoài bằng cách kết hợp các phân vùng trống liền kề;
- Chống phân mảnh đòi hỏi tái định vị động, việc lựa chọn chiến lược tối ưu là vấn đề phức tạp;
- Một trong các chiến lược là sử dụng swapping giữa bộ nhớ chính và bộ nhớ phụ.

3.3. Chống phân mảnh (compaction)

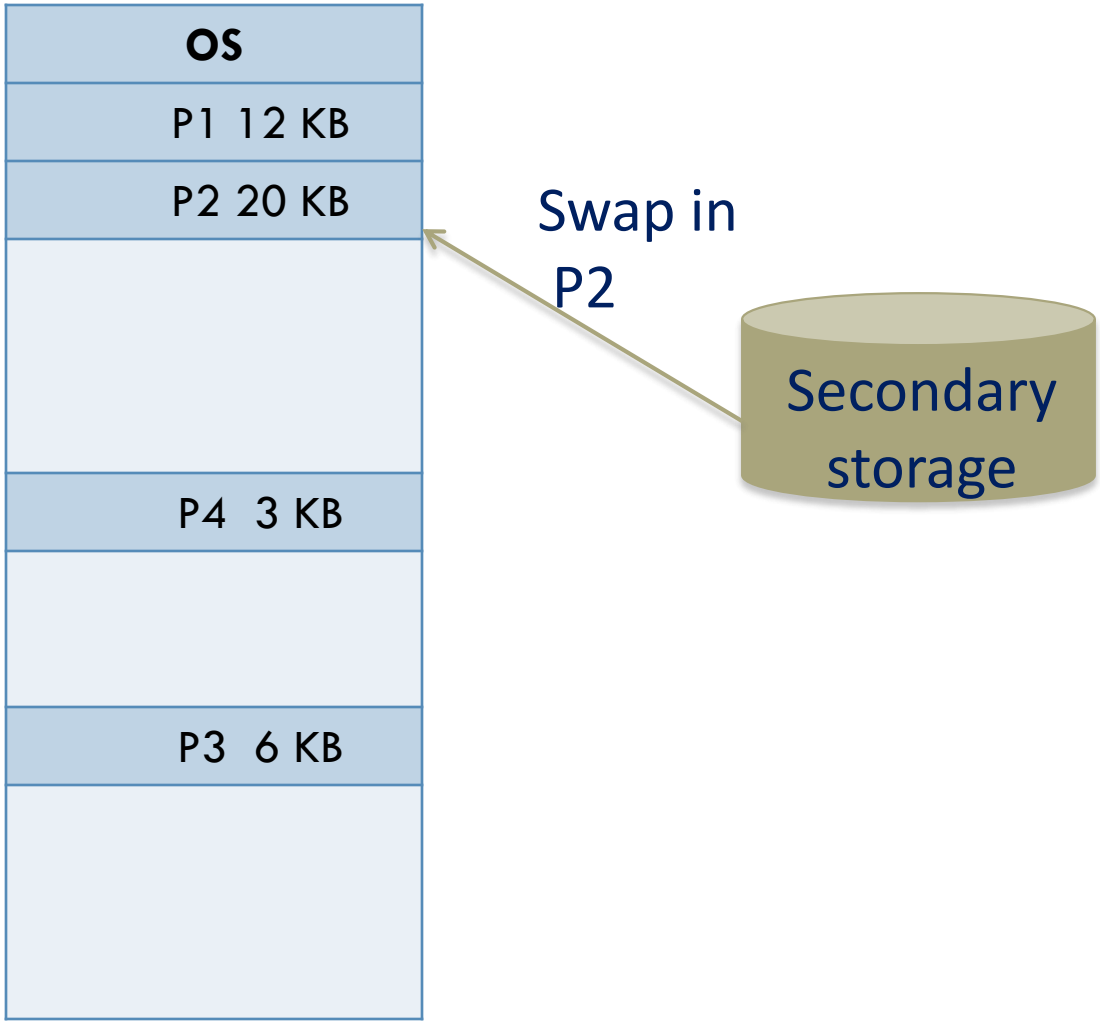
Bộ nhớ bị phân mảnh



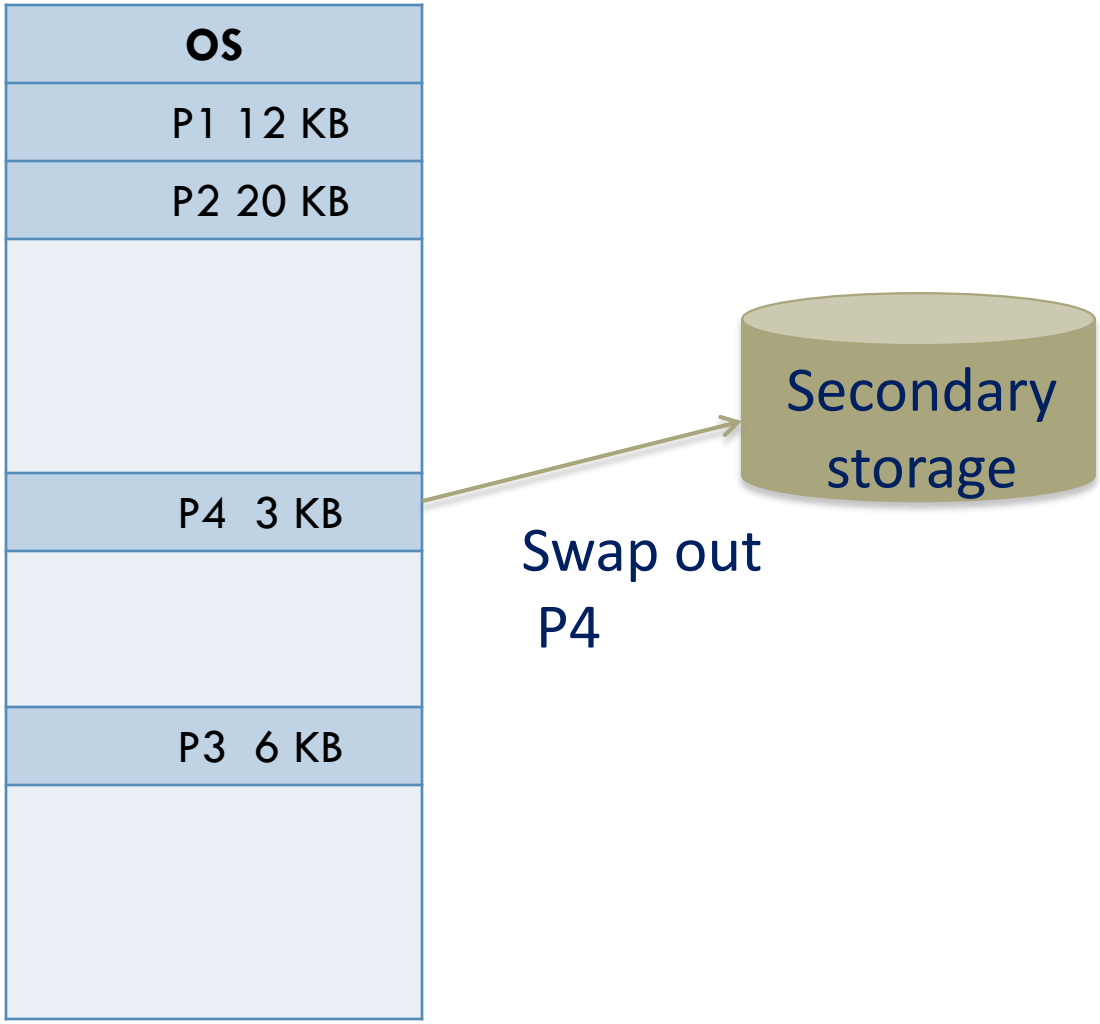
3.3. Chống phân mảnh (compaction)



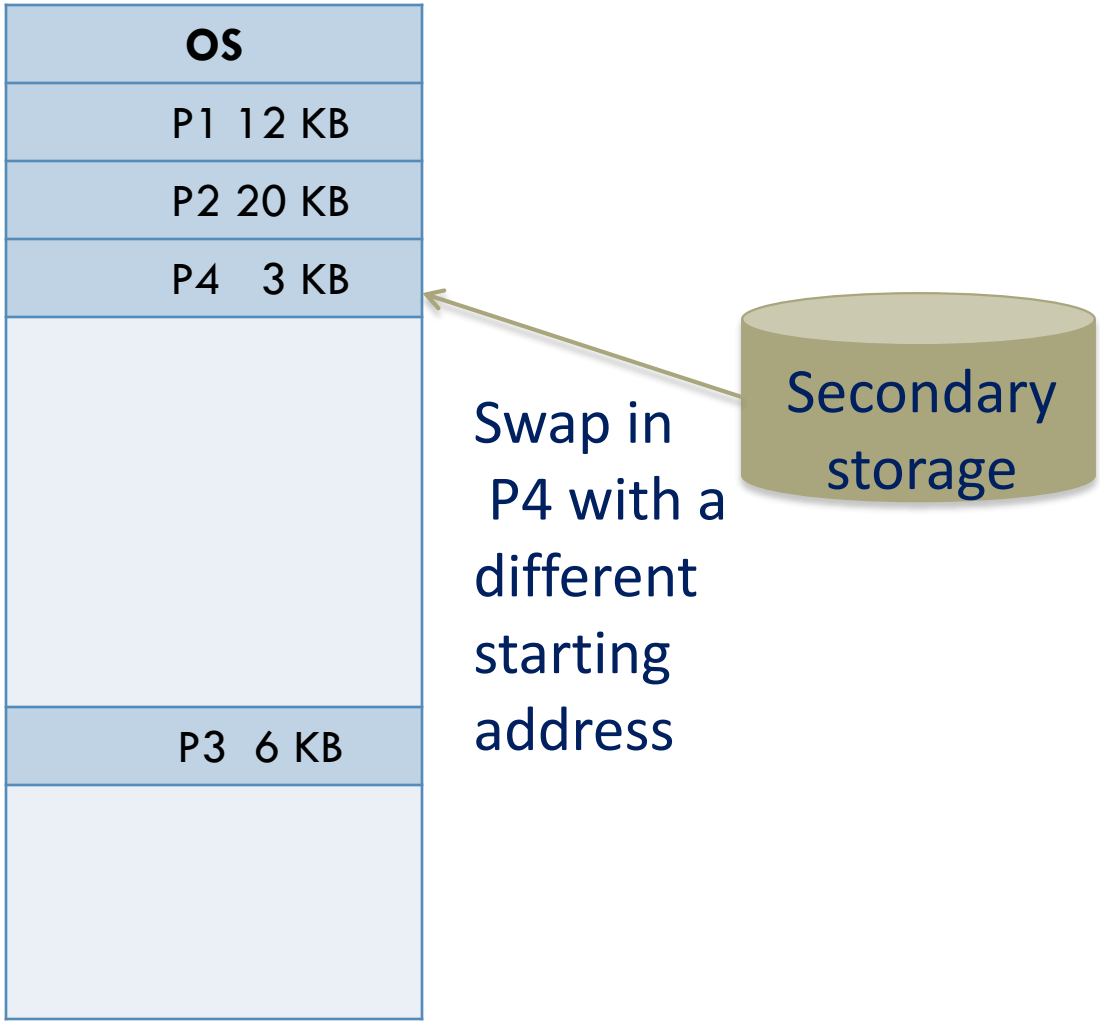
3.3. Chống phân mảnh (compaction)



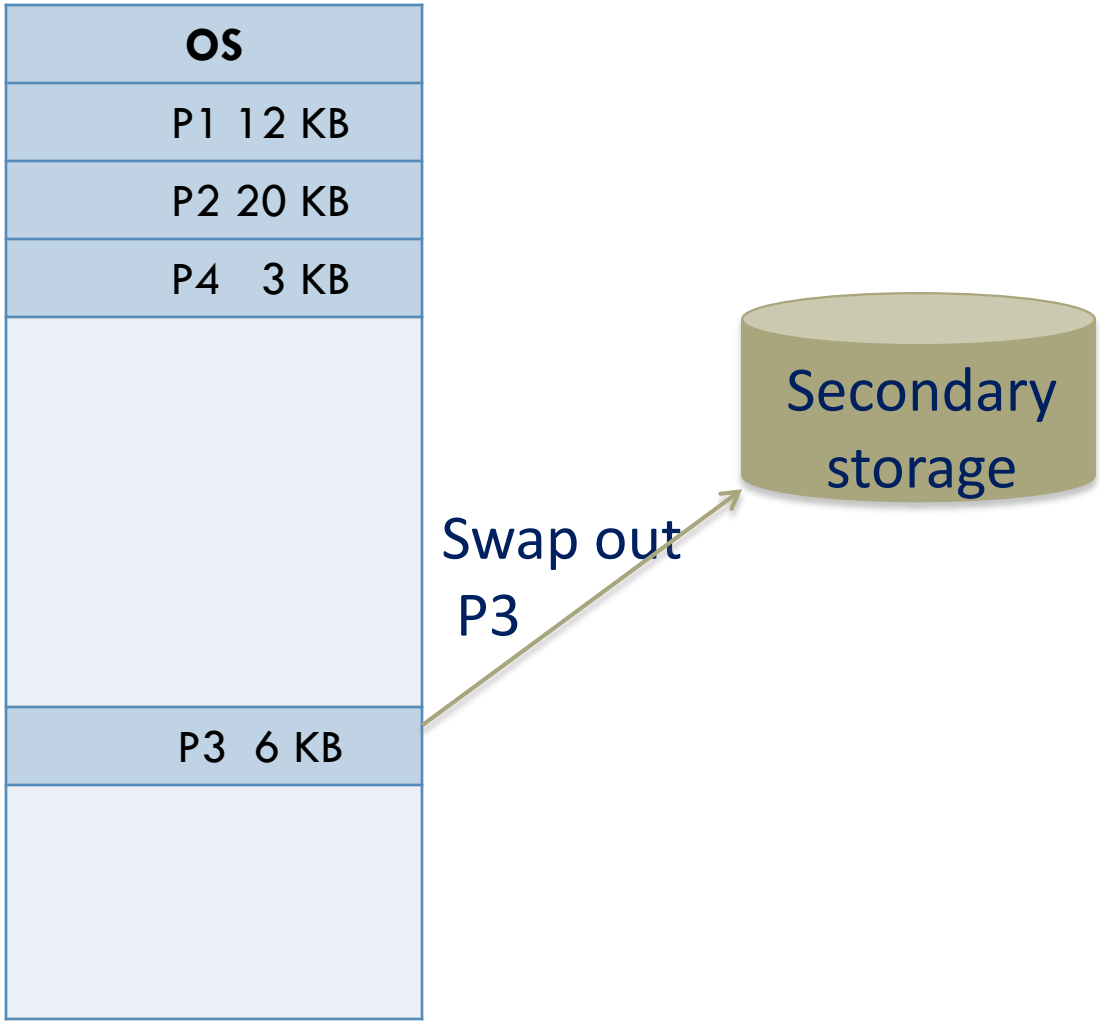
3.3. Chống phân mảnh (compaction)



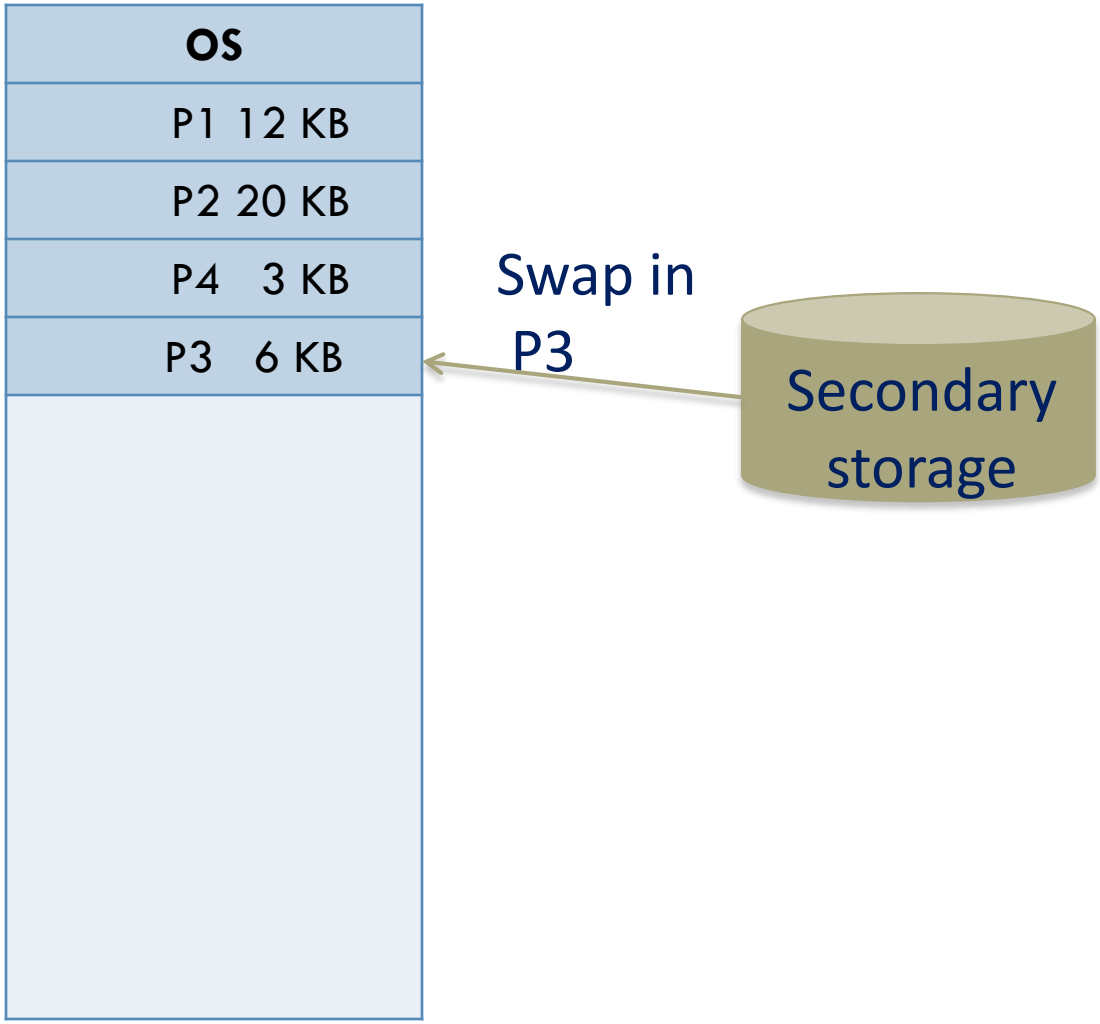
3.3. Chống phân mảnh (compaction)



3.3. Chống phân mảnh (compaction)

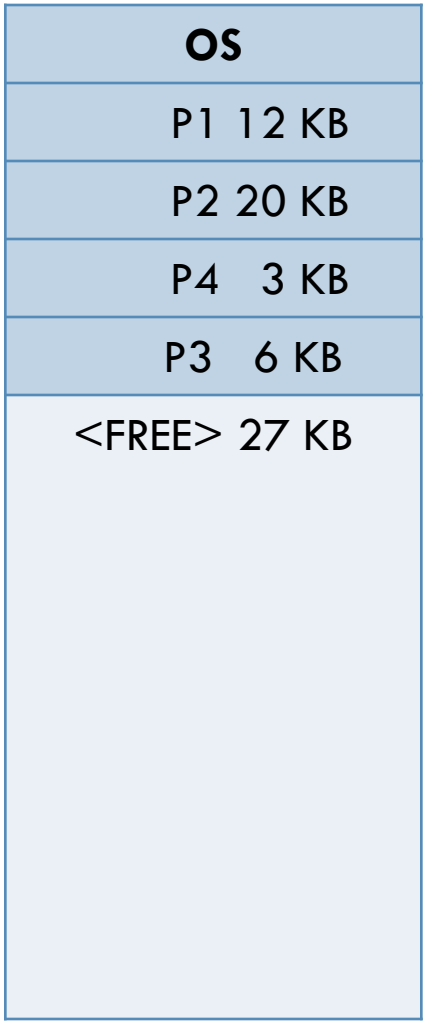


3.3. Chống phân mảnh (compaction)



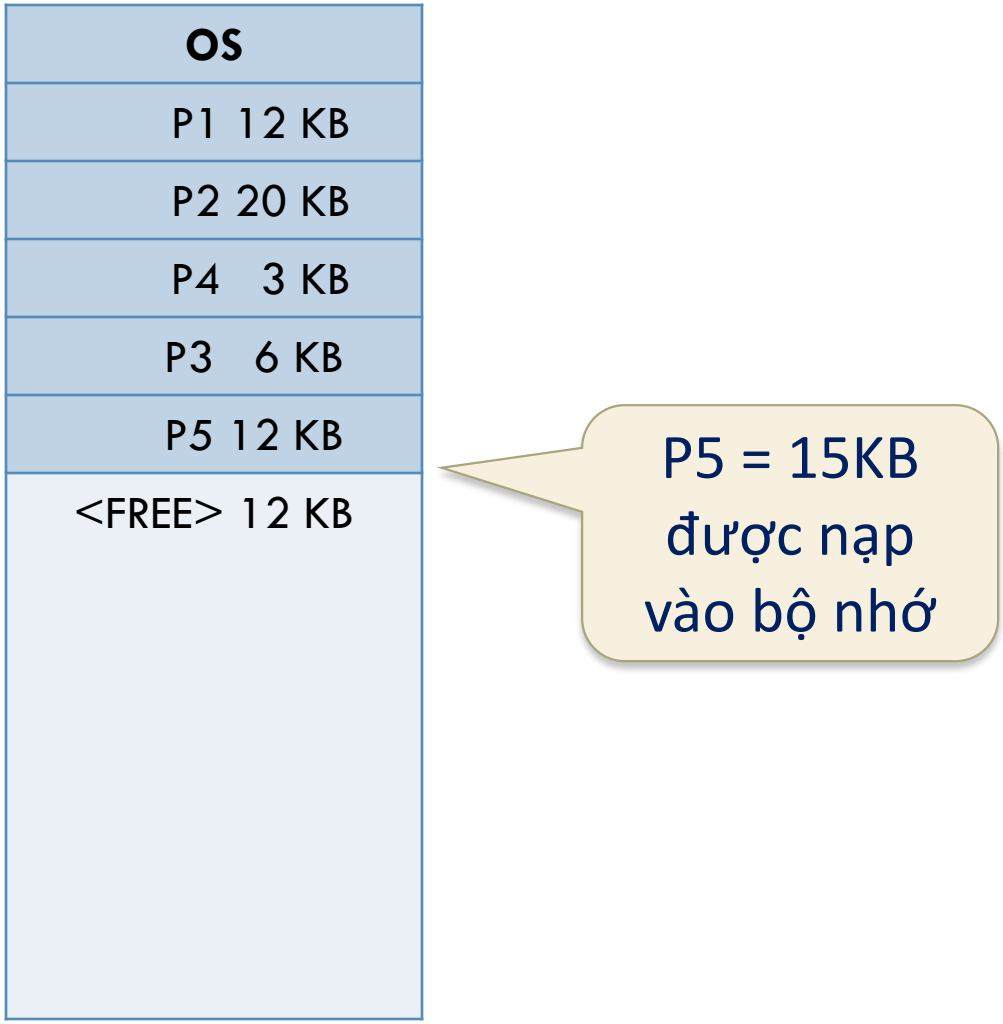
3.3. Chống phân mảnh (compaction)

Bộ nhớ sau khi chống phân mảnh



P5 = 15KB có thể được nạp

3.3. Chống phân mảnh (compaction)



Chương 3. Quản lý bộ nhớ

3.4. Tái định vị (Relocation)

- **Tái định vị tĩnh:** Mỗi tiến trình được nạp vào bộ nhớ, ở mỗi lần nạp có thể bị được đưa vào những vùng địa chỉ khác nhau nhưng không thay đổi trong suốt quá trình hoạt động của nó. Tái định vị tĩnh sử dụng cho phân vùng động;
- **Tái định vị động:** Địa chỉ bắt đầu của tiến trình có thể thay đổi trong quá trình hoạt động. Tái định vị động được sử dụng cho việc chống phân mảnh.

Chương 3. Quản lý bộ nhớ

3.5. Các dạng cấu trúc chương trình

- Cấu trúc chương trình tuyến tính
- Cấu trúc chương trình động
- Cấu trúc chương trình Overlay
- Cấu trúc chương trình phân trang
- Cấu trúc chương trình phân đoạn

3.5. Các dạng cấu trúc chương trình

- **Cấu trúc chương trình tuyến tính**
  - Tất cả các module, thư viện sử dụng trong chương trình khi biên dịch sẽ được biên dịch thành 1 module duy nhất;
  - Khi thực hiện HĐH phải nạp toàn bộ module này vào bộ nhớ;
  - Cấu trúc chương trình này có tính độc lập cao và có tốc độ thực thi cao;
  - Làm lãng phí bộ nhớ vì kích thước chương trình tăng lên khi biên dịch.

3.5. Các dạng cấu trúc chương trình

- **Cấu trúc chương trình động**
  - Chương trình được viết dưới dạng các module riêng rẽ
  - Được biên dịch thành các module riêng rẽ, các thư viện chuẩn của HĐH và của NN lập trình không được tích hợp trong module chính của chương trình
  - Khi thực thi chương trình chỉ 1 module chính được nạp vào bộ nhớ, các module khác khi cần sẽ được nạp vào sau
  - Cấu trúc này tiết kiệm được không gian nhớ nhưng thực thi chậm hơn cấu trúc tuyến tính

3.5. Các dạng cấu trúc chương trình

- **Cấu trúc chương trình Overlay**
  - Chương trình được biên dịch thành các module riêng Các module chương trình được chia thành các mức khác nhau:
    - Mức 0: Chứa module gốc dừng để nạp chương trình
    - Mức 1: Chức các module được gọi bởi mức 0
    - Mức 2: Chức các module được gọi bởi mức 1
    - …
    - Mức i: Chức các module được gọi bởi mức i-1

3.5. Các dạng cấu trúc chương trình

- **Cấu trúc chương trình Overlay(tt)**
  - Các module trong cùng một mức có thể có kích thước khác nhau, kích thước của module lớn nhất trong lớp được xem là kích thước của mức
  - Bộ nhớ dành cho chương trình cũng được tổ chức thành các mức tương ứng với các chương trình
  - Khi thực hiện chương trình HĐH nạp sơ đồ overlay của chương trình vào bộ nhớ sau đó nạp các module cần thiết ban đầu vào bộ nhớ
  - HĐH dựa vào sơ đồ overlay để nạp các module khác nếu cần

3.5. Các dạng cấu trúc chương trình

- **Cấu trúc chương trình phân trang**
  - Các module chương trình được biên dịch thành 1 module duy nhất nhưng sau đó được chia thành các phần có kích thước bằng nhau được gọi là các trang
  - Bộ nhớ phải được phân trang, tức chia thành các không gian nhớ bằng nhau gọi là khung trang
  - HĐH phải xây dựng bộ điều khiển trang (PCT-page control table)

3.5. Các dạng cấu trúc chương trình

- **Cấu trúc chương trình phân đoạn**
  - Chương trình được biên dịch thành nhiều module độc lập, được gọi là các đoạn
  - Bộ nhớ phải được phân đoạn, tức chia thành các không gian có kích thước có thể không bằng nhau tương ứng với kích thước của các đoạn chương trình
  - Khi thực hiện chương trình HĐH có thể nạp tất cả các đoạn hoặc 1 vài đoạn cần thiết vào các phân đoạn nhớ liên tiếp hoặc không liên tiếp
  - HĐH phải xây dựng bộ điều khiển đoạn (SCT-Segment control table).