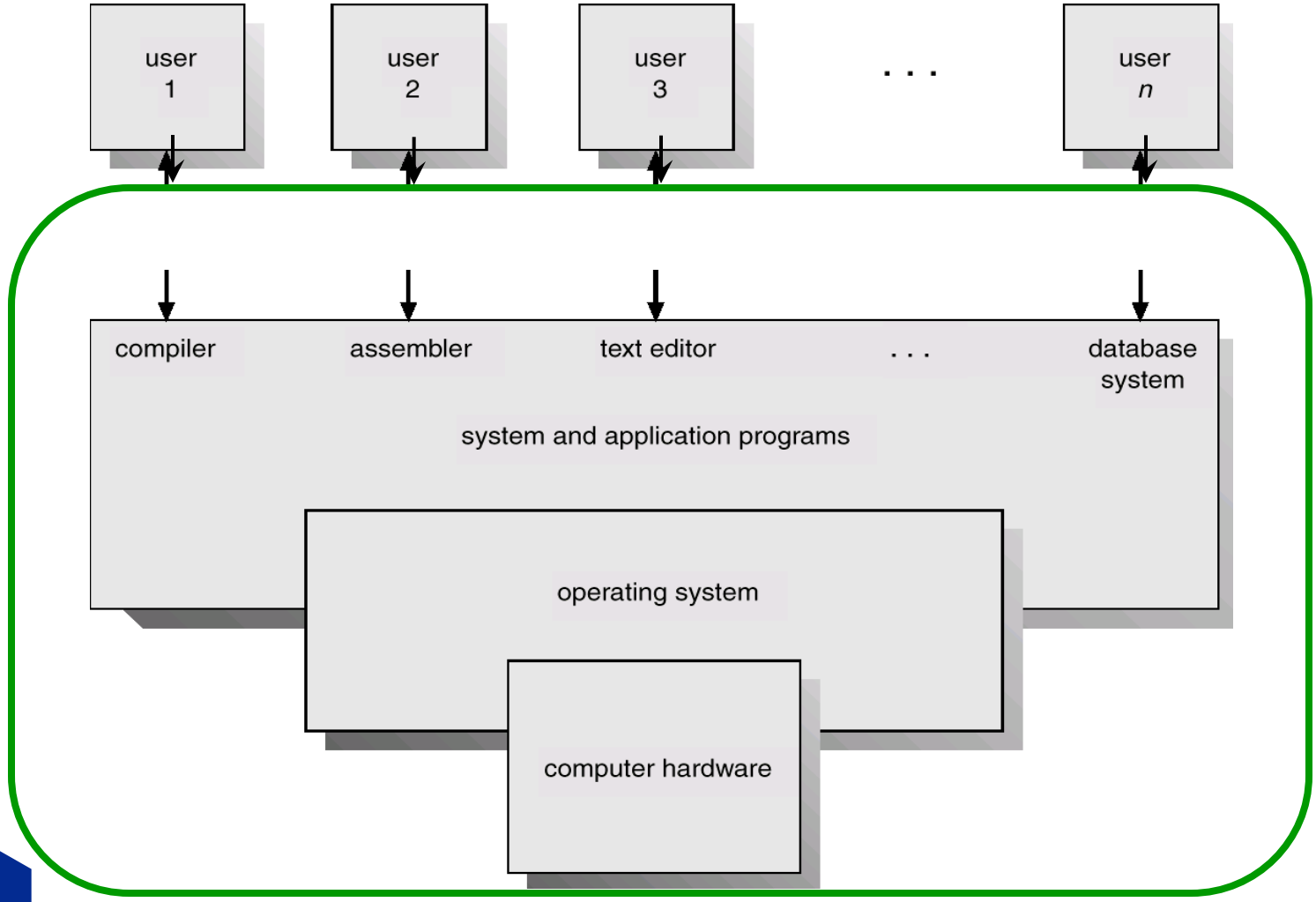
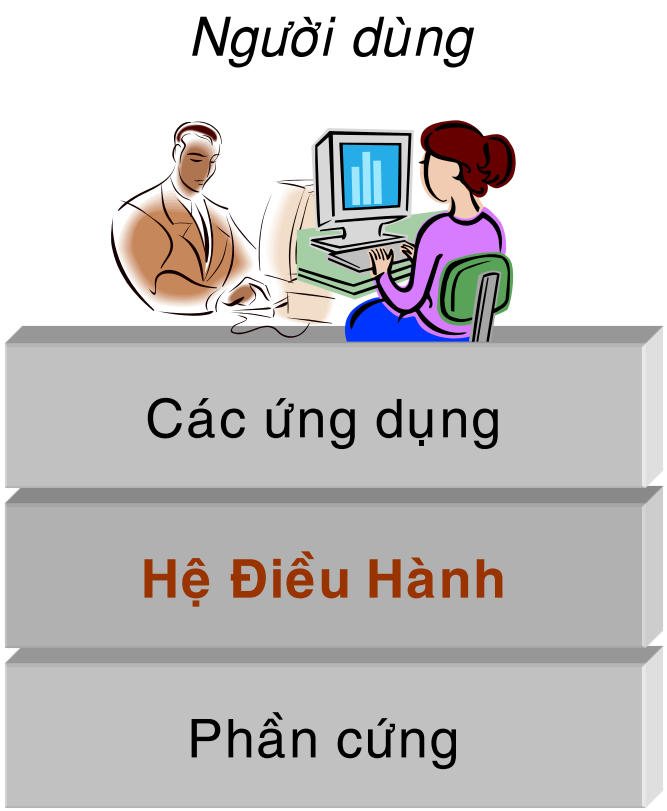
Các thành phần của hệ thống máy tính



Định nghĩa

- Hệ điều hành là gì?
  - *Chương trình* trung gian giữa phần cứng máy tính và người sử dụng, có chức năng *điều khiển phần cứng* và cung cấp các *dịch vụ cơ bản* cho các ứng dụng.
- Mục tiêu
  - Giúp người dùng dễ dàng sử dụng hệ thống.
  - Quản lý và cấp phát tài nguyên hệ thống một cách hiệu quả.



Các chức năng chính của OS

- Phân chia thời gian xử lý trên CPU (định thời)
- Phối hợp và đồng bộ hoạt động giữa các quá trình
- Quản lý tài nguyên hệ thống hiệu quả
- Kiểm soát quá trình truy cập, bảo vệ hệ thống
- Duy trì sự nhất quán của hệ thống, kiểm soát lỗi và phục hồi hệ thống khi có lỗi xảy ra.
- Cung cấp giao diện làm việc thuận tiện cho người dùng

Các thành phần chức năng OS

- *Quản lý quá trình* (process management)
- *Quản lý bộ nhớ chính*
- *Quản lý file*
- *Quản lý hệ thống I/O* (I/O system management)
- *Quản lý hệ thống lưu trữ thứ cấp*

Các dịch vụ OS

- Cung cấp giao diện cho người sử dụng:
  - Giao diện đồ họa
  - Giao diện các nhóm lệnh trình thông dịch lệnh
- Cung cấp tiện ích cho người phát triển:
  - Các thủ tục gọi chức năng hệ thống (system call)
  - Cấp phát tài nguyên
  - Bảo vệ (Protection)
  - An ninh hệ thống (security)

Quá trình

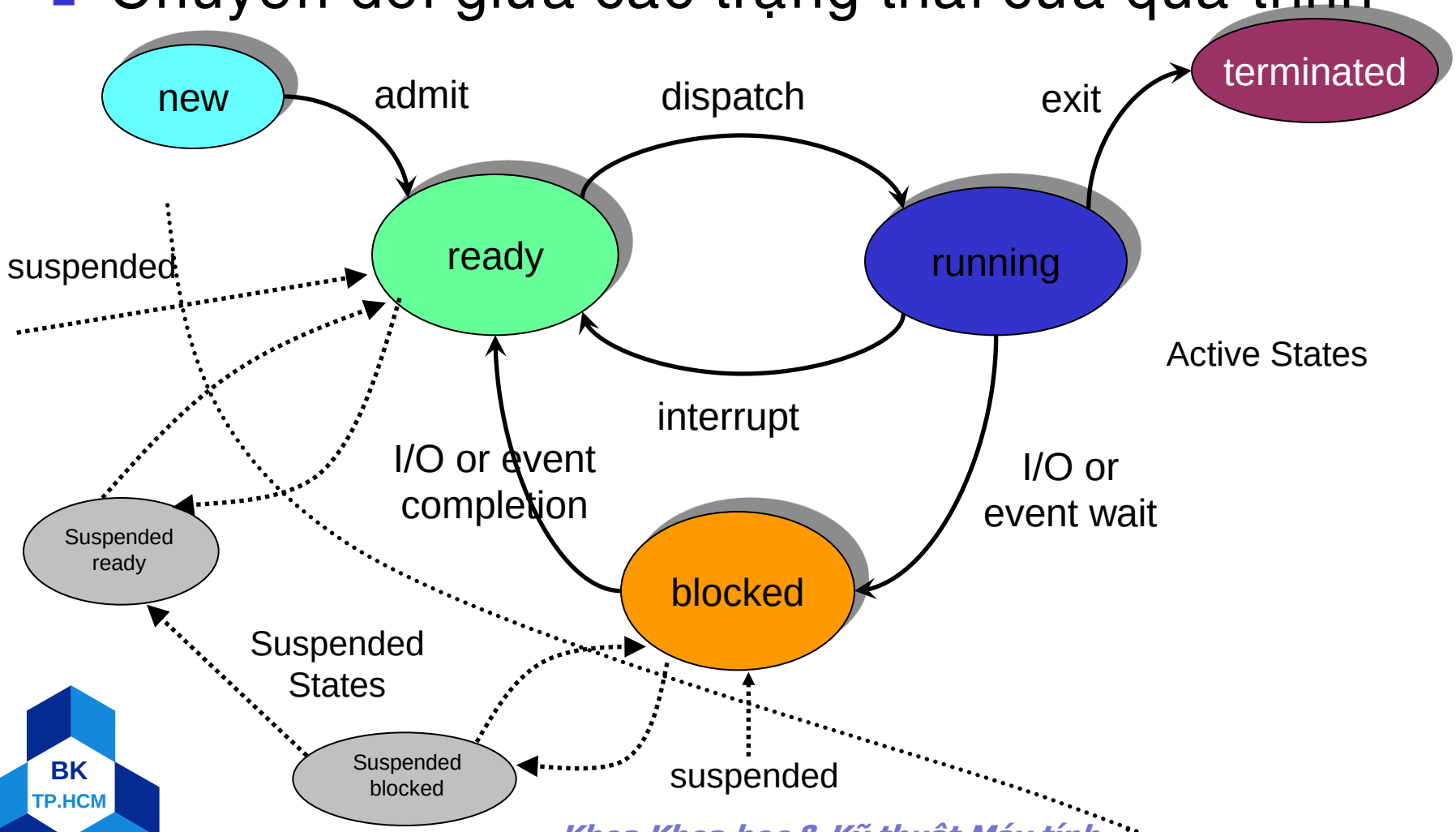
- Hệ thống máy tính thực thi nhiều chương trình khác nhau
  - Batch system: jobs
  - Time-shared systems: user programs, tasks
  - Job ≈ process
- *Quá trình* (process)
  - một chương trình đang thực thi, bao gồm:
    - *Text section* (program code), *data section* (chứa global variables)
    - Hoạt động hiện thời: program counter (PC), process status word (PSW), stack pointer (SP), memory management registers

Khởi tạo quá trình

- Các bước hệ điều hành khởi tạo 1 quá trình
  - Cấp phát *định danh* duy nhất (process number hay process identifier, pid) cho quá trình
  - Cấp phát không gian nhớ để nạp quá trình
  - Khởi tạo khối dữ liệu *Process Control Block* (PCB) cho quá trình
    - PCB là nơi hệ điều hành lưu các thông tin về quá trình
  - Thiết lập các mối liên hệ cần thiết (vd: sắp PCB vào hàng đợi định thời,…)

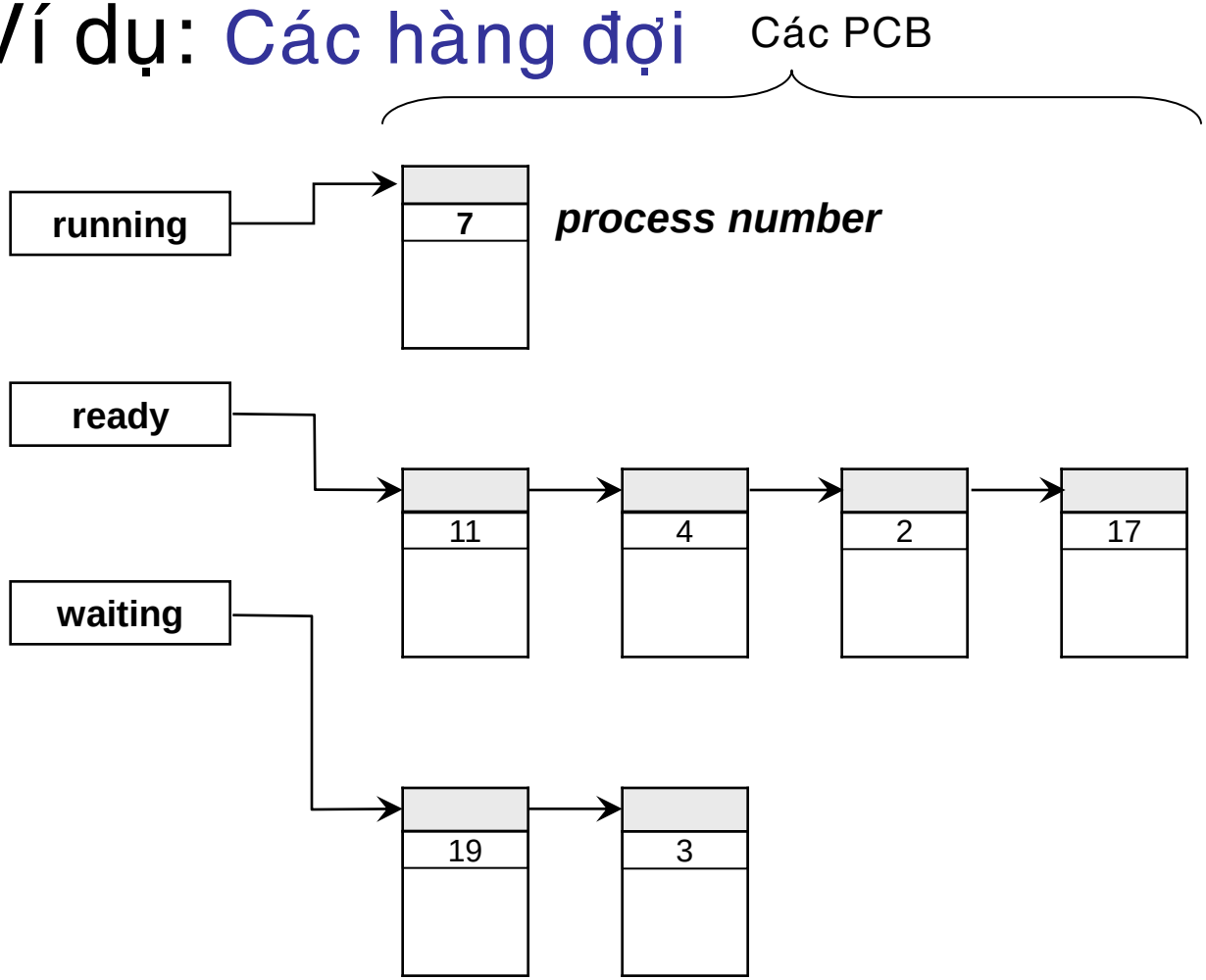
Các trạng thái của quá trình

- Chuyển đổi giữa các trạng thái của quá trình



Quản lý quá trình

- Ví dụ: Các hàng đợi



Cộng tác giữa các quá trình

- Trong quá trình thực thi, các quá trình có thể *cộng tác* (cooperate) để hoàn thành công việc
- Các quá trình cộng tác để
  - Chia sẻ dữ liệu (information sharing)
  - Tăng tốc tính toán (computational speedup)
    - Nếu hệ thống có nhiều CPU, chia công việc tính toán thành nhiều công việc tính toán nhỏ chạy song song
- Sự cộng tác giữa các quá trình yêu cầu hệ điều hành cung cấp cơ chế đồng bộ hoạt động (chương 3) và cơ chế giao tiếp cho các quá trình

Đồng bộ các quá trình

- Critical section
- Các giải pháp dùng lệnh máy thông thường
  - Giải thuật Peterson, và giải thuật bakery
- Các giải pháp dùng lệnh cấm ngắt hoặc lệnh máy đặc biệt
- Semaphore
- Monitor

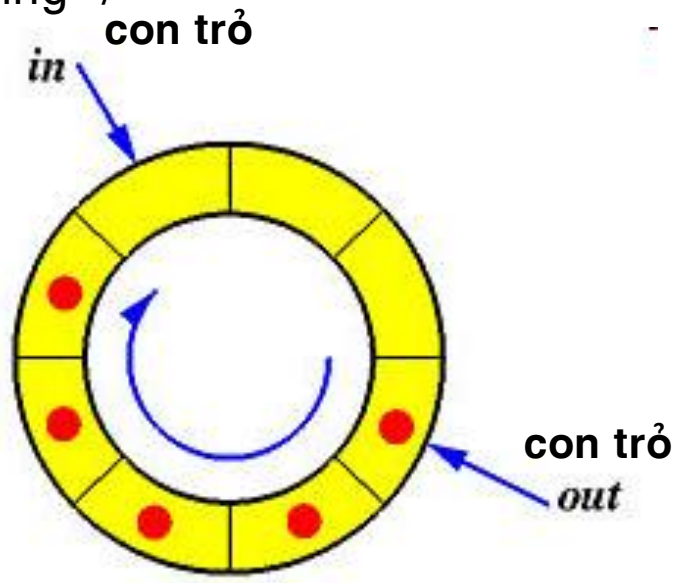
Bài toán Producer-consumer

- **Quá trình Producer**

```
item nextProduced;
while(1) {
    while (count == BUFFER_SIZE); /* do nothing */
    buffer[in] = nextProduced;
    count++;
    in = (in + 1) % BUFFER_SIZE;
}
```

- **Quá trình Consumer**

```
item nextConsumed;
while(1) {
    while (count == 0); /* do nothing */
    nextConsumed = buffer[out];
    count--;
    out = (out + 1) % BUFFER_SIZE;}
```



biến count được chia sẻ giữa producer và consumer

"Critical Section"

- Giả sử có *n* process đồng thời truy xuất dữ liệu chia sẻ.
- Không phải tất cả các đoạn code đều cần được giải quyết vấn đề race condition mà chỉ những đoạn code có chứa các thao tác lên dữ liệu chia sẻ. Đoạn code này được gọi là *vùng tranh chấp* (critical section, *CS*).
- Bài toán loại trừ tương hỗ: phải bảo đảm sự *loại trừ tương hỗ* (mutual exclusion, mutex), tức là khi một process P đang thực thi trong CS của P, không có process Q nào khác đồng thời thực thi các lệnh trong CS của Q.

Cấu trúc tổng quát của quá trình trong bài toán loại trừ tương hỗ

- Giả sử mỗi process thực thi bình thường (i.e., nonzero speed) và không có sự tương quan giữa tốc độ thực thi của các process
- Cấu trúc tổng quát của một process:

```
do {
        entry section
        critical section
        exit section
        remainder section
    } while(1);
```

Một số giả định

- Có thể có nhiều CPU nhưng phần cứng không cho phép nhiều tác vụ truy cập một vị trí trong bộ nhớ cùng lúc (simultaneous)
- Không ràng buộc về thứ tự thực thi của các process
- Các process có thể chia sẻ một số biến chung nhằm đồng bộ hoạt động của chúng
- Giải pháp cần phải đặc tả entry section và exit section

Giải bài toán loại trừ tươnghỗ

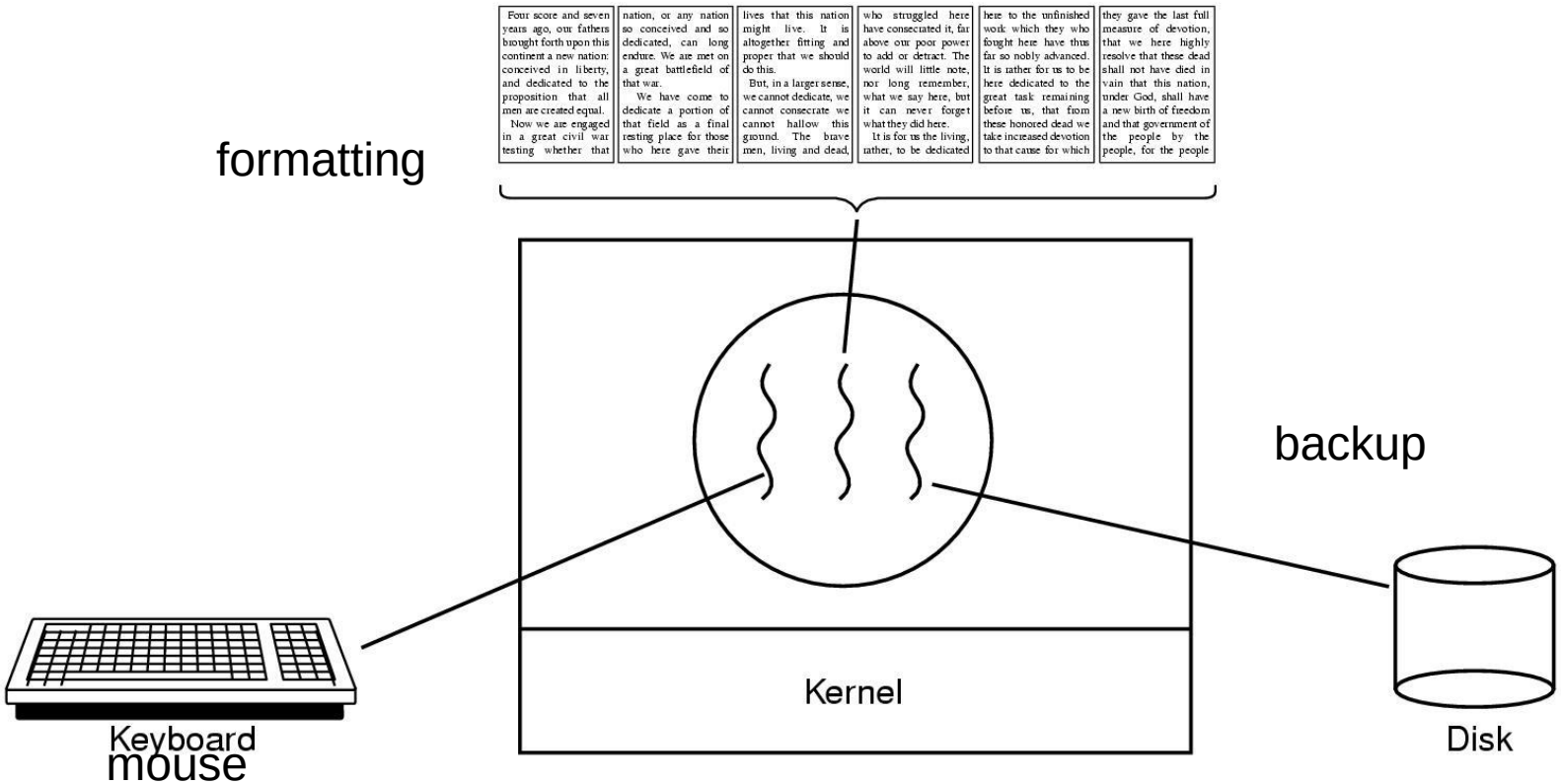
Lời giải phải thỏa ba tính chất:

1. *Mutual exclusion*: Khi một process P đang thực thi trong vùng tranh chấp (CS) thì không có process Q nào khác đang thực thi trong CS.
2. *Progress:*
   - (Progress cho entry section) Nếu ít nhất một process đang trong entry section và không có process nào đang trong critical section, thì một process vào critical section tại một thời điểm sau đó.
   - (Progress cho exit section) Nếu ít nhất một process đang trong exit section, thì một process vào remainder section tại một thời điểm sau đó.
3. *Starvation freedom* (*lockout-freedom*):
   - (cho entry section) quá trình vào entry section sẽ vào CS
   - (cho exit section) quá trình vào exit section sẽ vào remainder section.

Phân loại giải pháp cho loại trừ tương hỗ

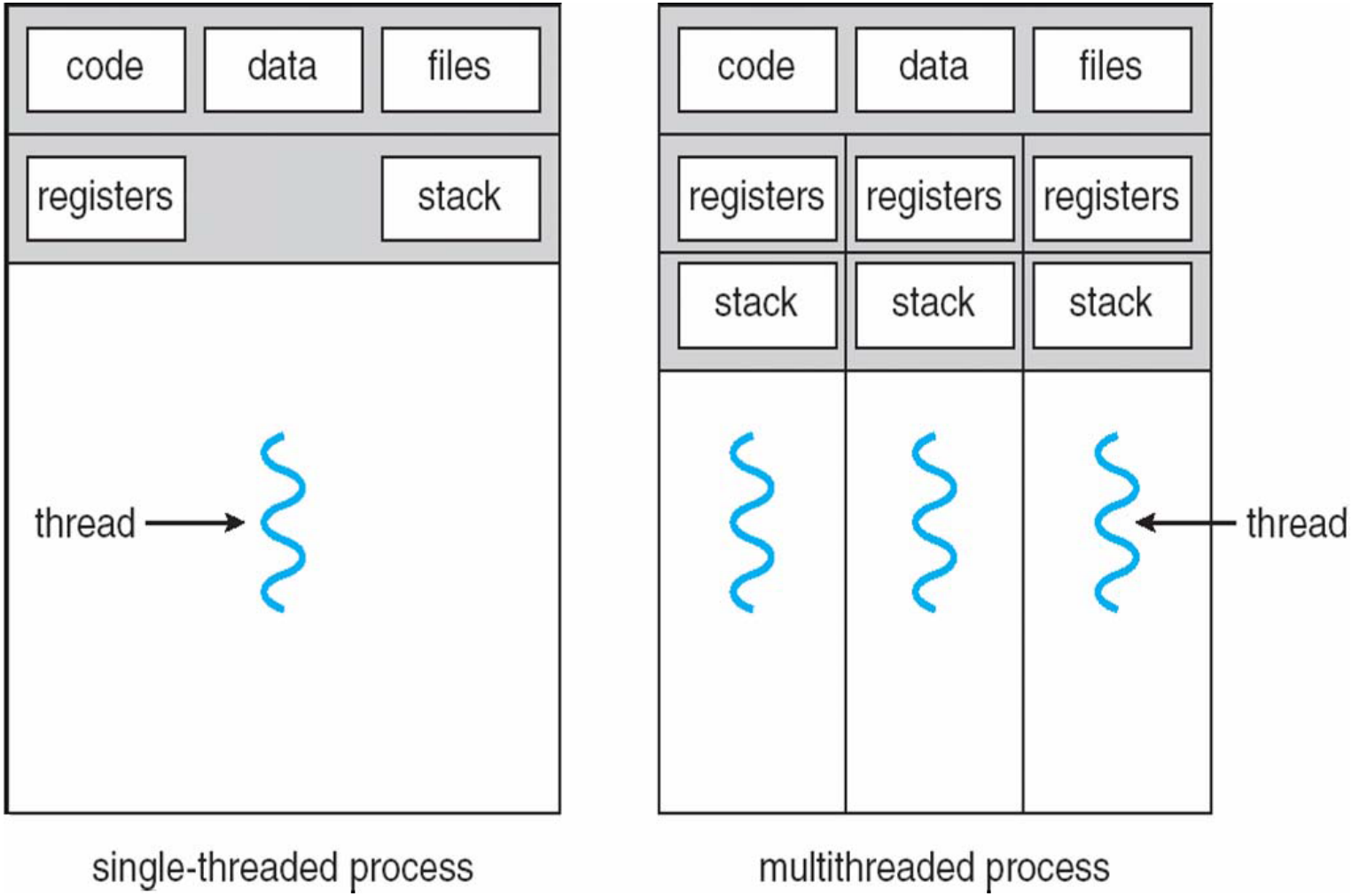
- Giải pháp dùng lệnh máy thông thường
- Giải pháp dùng lệnh cấm ngắt hay lệnh máy đặc biệt
  - Lệnh Disable interrupt
  - Lệnh máy đặc biệt như
    - TestAndSet

Sử dụng thread



A word processor with three threads

Quá trình đơn & đa luồng

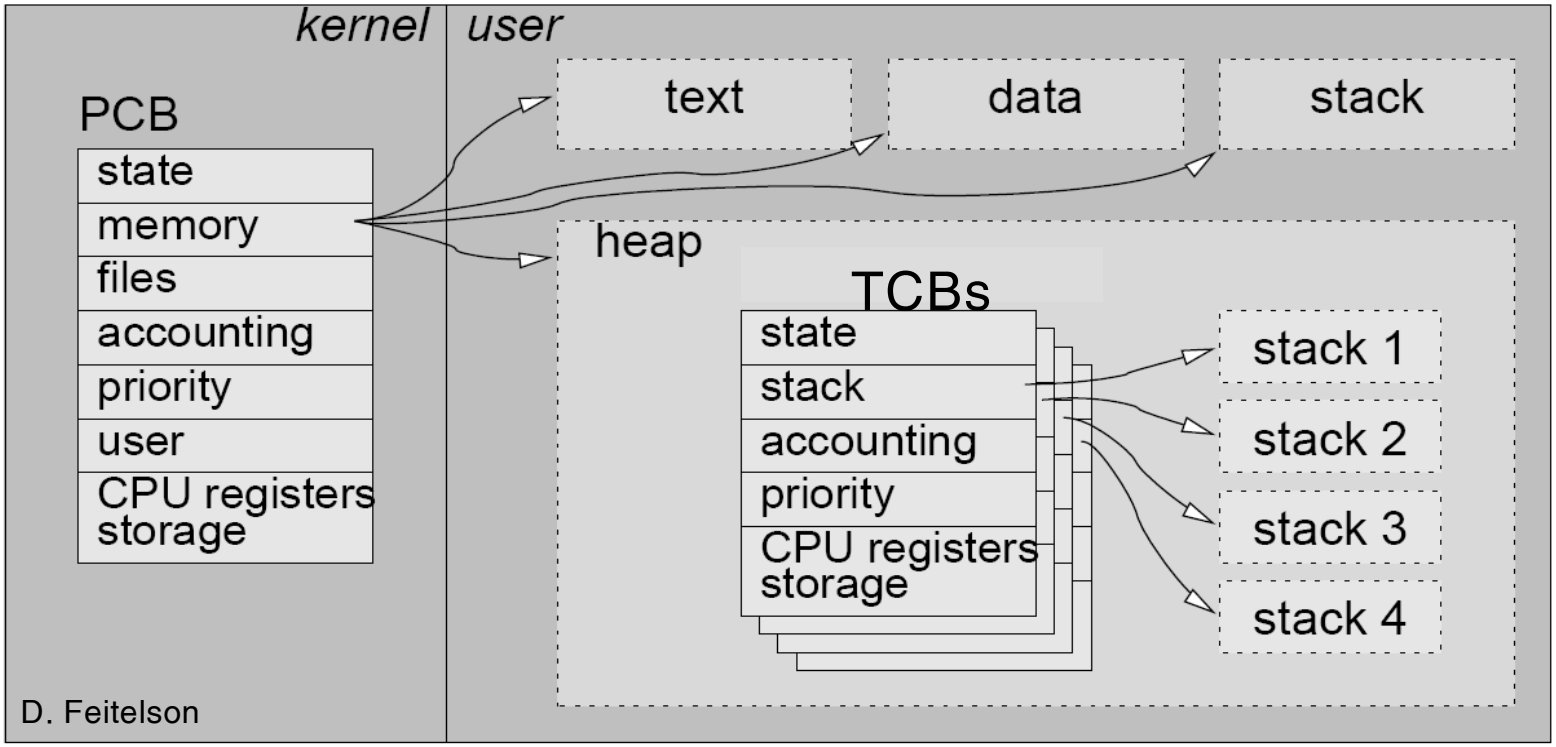


Ưu điểm của thread

- **Tính đáp ứng** (responsiveness) cao cho các ứng dụng tương tác multithreaded
- **Chia sẻ** tài nguyên (resource sharing)
  - ví dụ memory
- **Tiết kiệm** chi phí hệ thống (economy)
  - Chi phí tạo/quản lý thread nhỏ hơn so với quá trình
  - Chi phí chuyển ngữ cảnh giữa các thread nhỏ hơn so với quá trình
- **Tận dụng** được đa xử lý (multiprocessor)
  - Mỗi thread chạy trên một processor riêng, do đó tăng mức độ song song của chương trình.

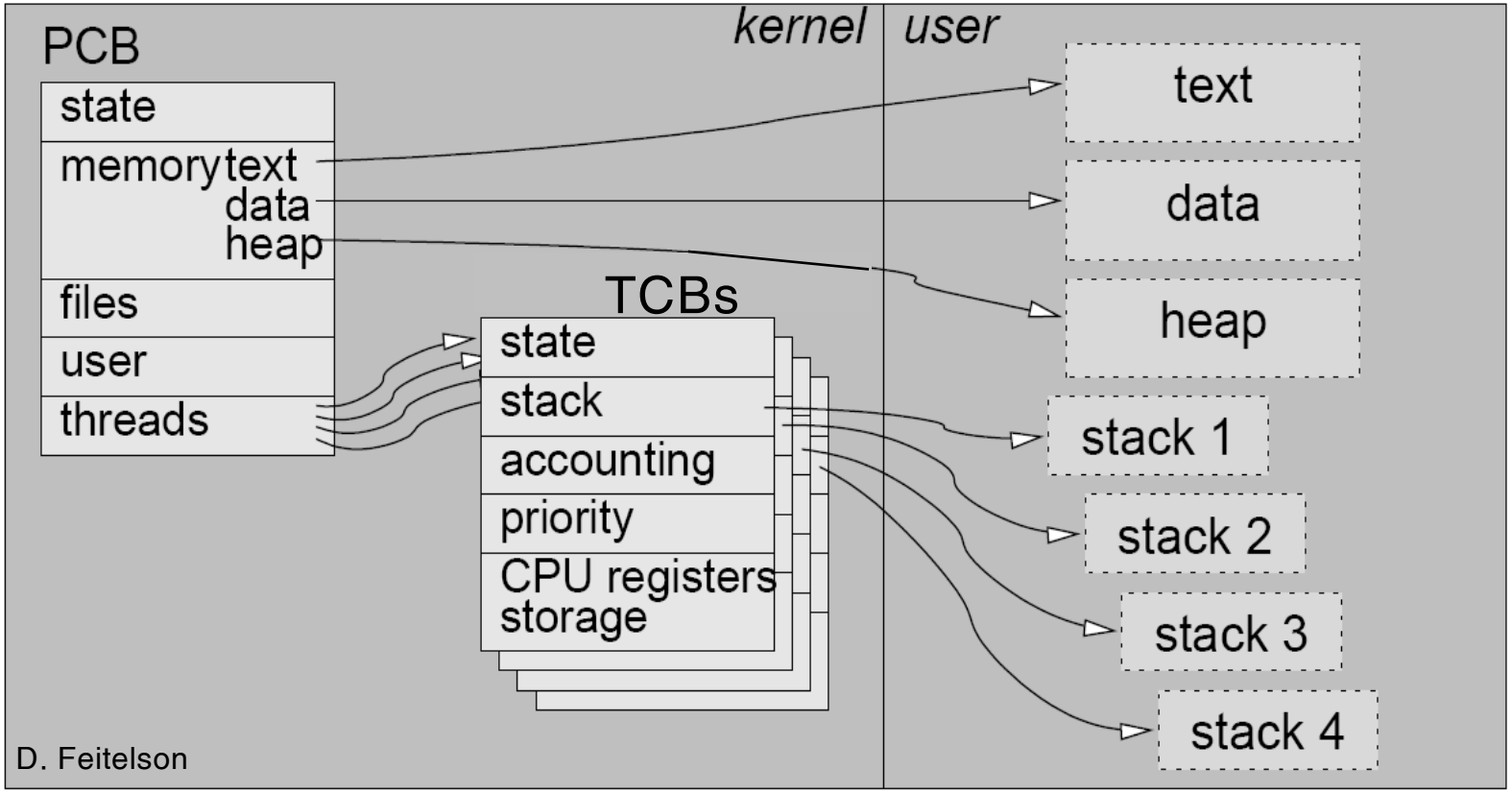
User thread (tt.)

- Cấu trúc dữ liệu và memory layout để hiện thực user thread



Kernel thread (tt.)

- Cấu trúc dữ liệu và memory layout để hiện thực kernel thread



Mô hình thread

- Multithreading có thể hiện thực theo một trong các mô hình sau
  - Mô hình *many-to-one*
  - Mô hình *one-to-one*
  - Mô hình *many-to-many*

Tắc nghẽn (Deadlock)

- Định nghĩa
- Phương pháp giải quyết nghẽn
  - Chống (Ngăn) nghẽn
  - Tránh (avoidance) nghẽn
  - Phát hiện nghẽn
  - Phục hồi nghẽn

Điều kiện cần để xảy ra nghẽn

Bốn điều kiện cần (necessary conditions)

1. *Mutual exclusion:* ít nhất một tài nguyên được giữ theo nonsharable mode (ví dụ: printer; ví dụ sharable resource: read-only file).
2. *Hold and wait:* một process đang giữ ít nhất một tài nguyên và đợi thêm tài nguyên do quá trình khác đang giữ.
3. *No preemption:* (= no resource preemption) không lấy lại tài nguyên đã cấp phát cho process, ngoại trừ khi process tự hoàn trả nó.
4. *Circular wait:* tồn tại một tập $\{P_0,\ldots,P_n\}$ các quá trình đang đợi sao cho

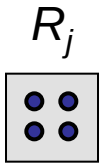
   P_0 đợi một tài nguyên mà P_1 đang giữ
   P_1 đợi một tài nguyên mà P_2 đang giữ
   …
   P_n đợi một tài nguyên mà P_0 đang giữ

Resource Allocation Graph (tt.)

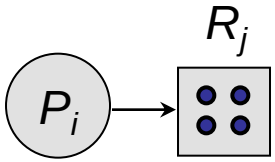
Ký hiệu

- Process:

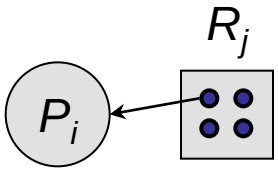
- Loại tài nguyên với 4 thực thể:



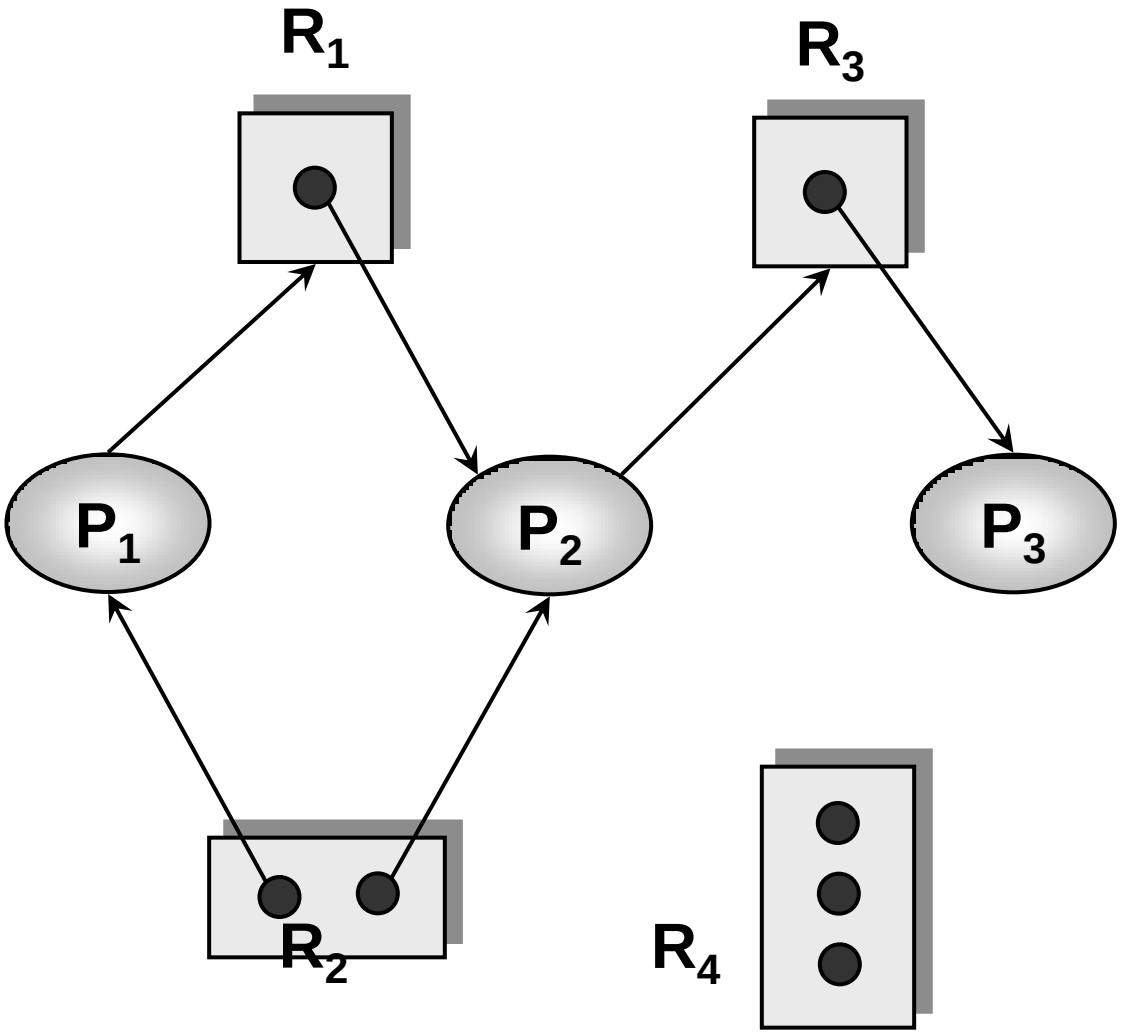
- P_i yêu cầu một thực thể của R_j :



- P_i đang giữ một thực thể của R_j :



RAG



Ngăn deadlock

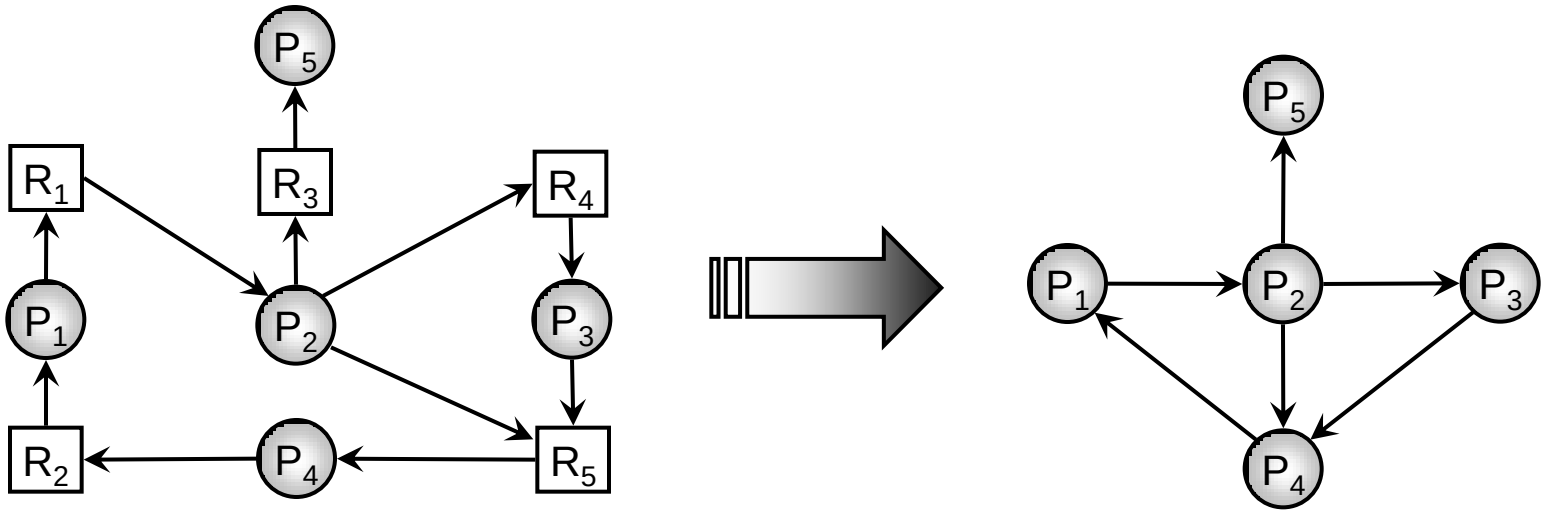
1. Ngăn mutual exclusion
2. Ngăn Hold and Wait
3. Ngăn No Preemption
4. Ngăn Circular Wait

Tránh (avoidance) Nghẽn

- Deadlock prevention sử dụng tài nguyên không hiệu quả.
- Deadlock avoidance vẫn đảm bảo hiệu suất sử dụng tài nguyên tối đa đến mức có thể.
- Yêu cầu mỗi process khai báo số lượng tài nguyên tối đa cần để thực hiện công việc
- Giải thuật deadlock-avoidance sẽ điều khiển *trạng thái cấp phát tài nguyên* (resource-allocation state) để bảo đảm hệ thống không rơi vào deadlock.
  Trạng thái cấp phát tài nguyên được định nghĩa dựa trên số tài nguyên còn lại, số tài nguyên đã được cấp phát và yêu cầu tối đa của các process.

Phát hiện Deadlock

- Sử dụng *wait-for graph*
  - Wait-for graph được dẫn xuất từ RAG bằng cách bỏ các node biểu diễn tài nguyên và ghép các cạnh tương ứng:
    - Có cạnh từ P_i đến $P_j \Leftrightarrow P_i$ đang chờ tài nguyên từ P_j



- Gọi định kỳ một giải thuật kiểm tra có tồn tại chu trình trong wait-for graph hay không. Giải thuật phát hiện chu trình có thời gian chạy là $O(n^2)$, với n là số đỉnh của graph.

Phục hồi khỏi deadlock

- Các giải pháp khi phát hiện deadlock
  - báo người vận hành (operator), người này sẽ xử lý tiếp

  hoặc

  - hệ thống tự động phục hồi bằng cách phá deadlock:
    - Giải pháp chấm dứt quá trình

  hoặc

    - Giải pháp lấy lại tài nguyên

Định thời ngắn hạn

- Xác định process nào được thực thi tiếp theo, còn gọi là *định thời CPU*

- Được kích hoạt khi có một sự kiện có thể dẫn đến khả năng chọn một process để thực thi
  - Ngắt thời gian (clock interrupt)
  - Ngắt ngoại vi (I/O interrupt)
  - Lời gọi hệ thống (operating system call)
  - Signal

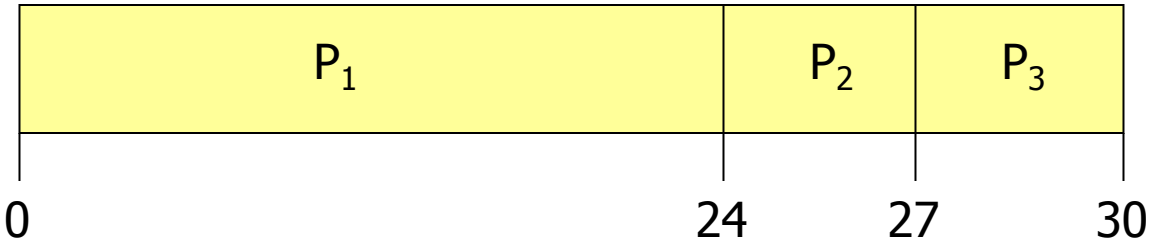
Tiêu chí định thời (tt.)

- Độ lợi CPU – giữ CPU càng bận càng tốt (Cao nhất)
- Thông năng – số lượng process kết thúc việc thực thi trong một đơn vị thời gian (Nhiều nhất)
- Turnaround time – thời gian kể từ lúc đưa vào (submission) đến lúc kết thúc (Ngắn nhất)
- Thời gian chờ – thời gian một process chờ trong hàng đợi ready (Ngắn nhất)
- Thời gian đáp ứng – thời gian từ khi đưa yêu cầu đến khi có đáp ứng đầu tiên (Nhanh nhất)

First Come First Served (FCFS) (tt.)

Process	Burst time (ms)
P_1	24
P_2	3
P_3	3

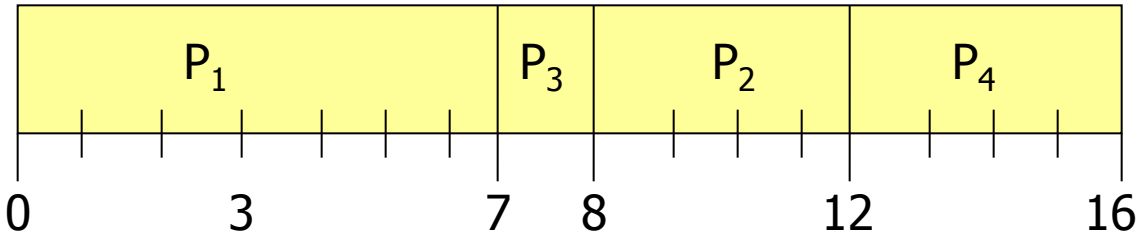
- Giả sử các process đến theo thứ tự P_1 , P_2 , P_3
- *Giản đồ Gantt* cho việc định thời là:
  - Thời gian đợi cho $P_1 = 0$, $P_2 = 24$, $P_3 = 27$
  - Thời gian đợi trung bình: (0 + 24 + 27) / 3 = 17



Shortest Job First (SJF) (tt.)

Process	Thời điểm đến	Burst time (ms)
P_1	0,0	7
P_2	2,0	4
P_3	4,0	1
P_4	5,0	4

- Giản đồ Gantt khi định thời theo SJF



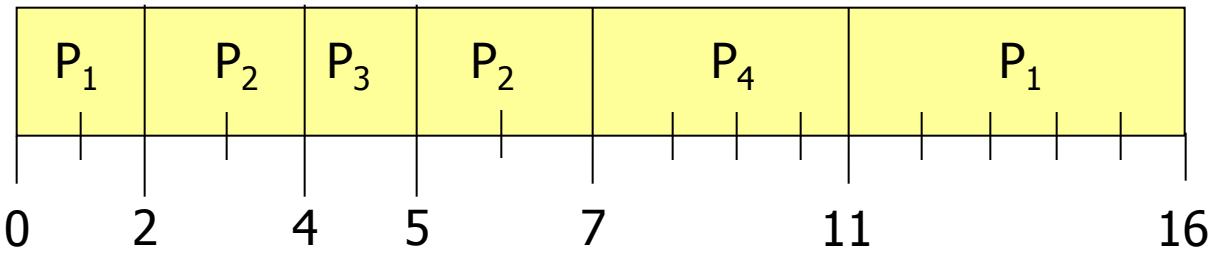
- Thời gian đợi trung bình = (0 + 6 + 3 + 7)/4 = 4

Shortest Remaining Time First (tt.)

(Ví dụ giống vd cho SJF)

Process	Thời điểm đến	Burst time (ms)
P_1	0,0	7
P_2	2,0	4
P_3	4,0	1
P_4	5,0	4

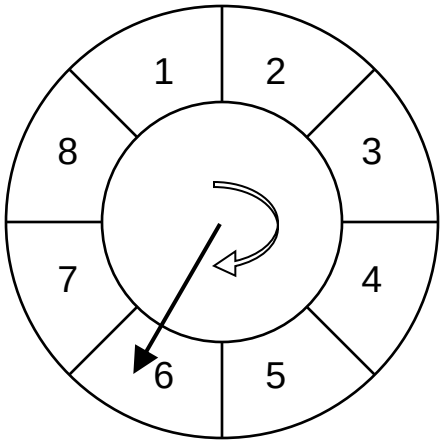
- Giản đồ Gantt khi định thời theo SRTF



- Thời gian đợi trung bình = (9 + 1 + 0 + 2) / 4 = 3
  - Tốt hơn giải thuật SJF

Round Robin (RR)

- Hàm lựa chọn: giống FCFS



Round Robin

Process	Burst time (ms)
P_1	53
P_2	17
P_3	68
P_4	24

- Quantum time = 20 ms
- Giản đồ Gantt:

P_1	P_2	P_3	P_4	P_1	P_3	P_4	P_1	P_3	P_3

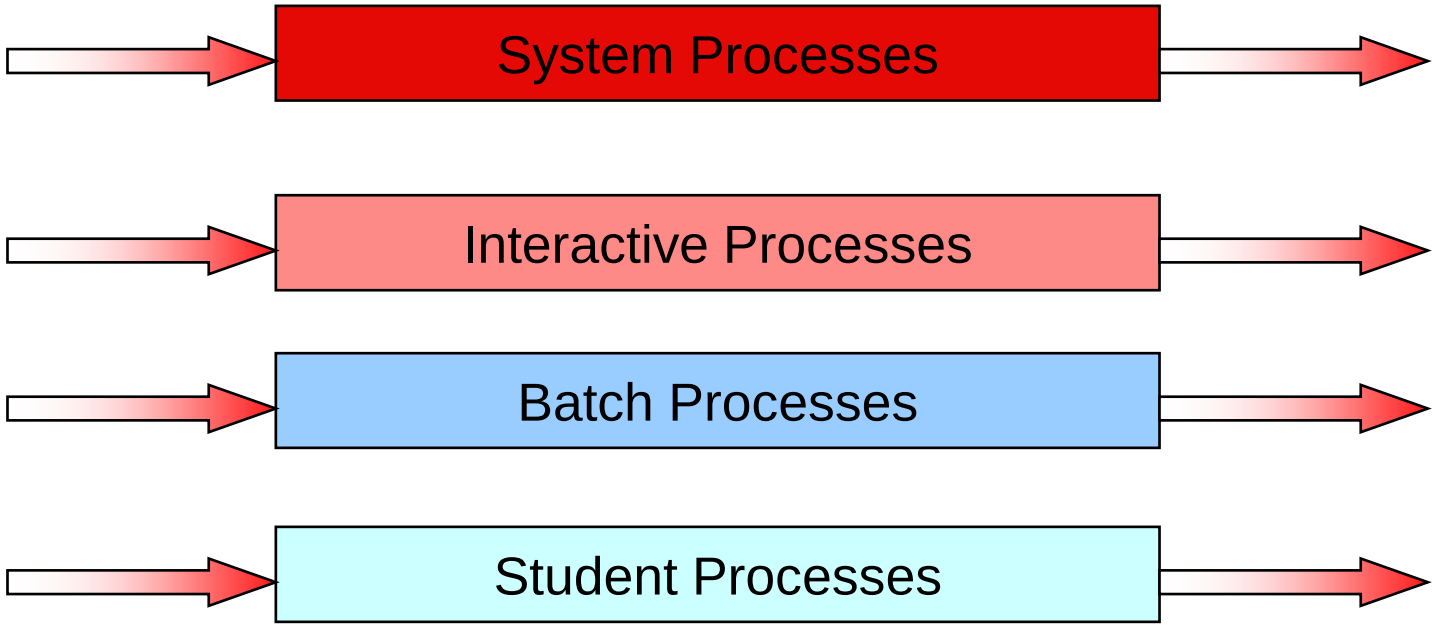
0 20 37 57 77 97 117 121 134 154 162

- Thường có thời gian quay vòng cao hơn SJF, nhưng lại có thời gian đáp ứng tốt hơn

Multilevel Queue Scheduling (tt.)

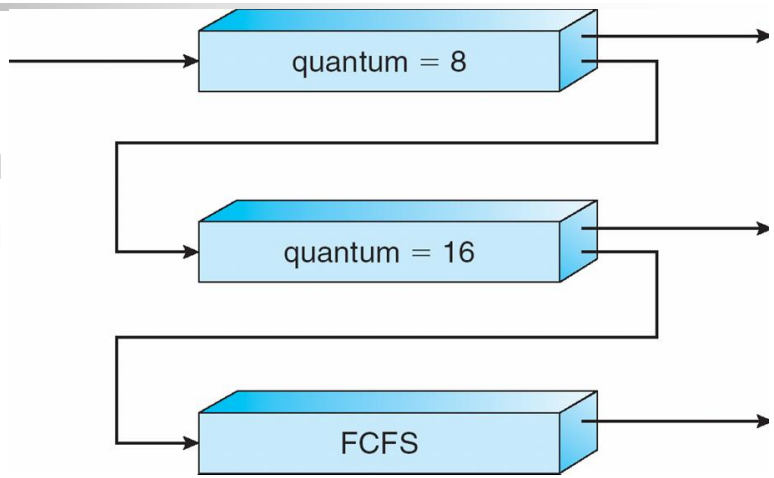
- Ví dụ phân nhóm các quá trình

Độ ưu tiên cao nhất



Độ ưu tiên thấp nhất

Multilevel Feedback Queue (tt.)



- Ví dụ: Có 3 hàng đợi
  - Q_0 , dùng RR với quantum
  - Q_1 , dùng RR với quantum
  - Q_2 , dùng FCFS
- Giải thuật
  - Công việc mới sẽ vào hàng đợi Q_0. Khi đến lượt mình, công việc sẽ được một khoảng thời gian là 8 milli giây. Nếu không kết thúc được trong 8 milli giây, công việc sẽ được đưa xuống hàng đợi Q_1
  - Tại Q_1, tương tự công việc sau khi chờ sẽ được cho một khoảng thời gian thực thi là 16 milli giây. Nếu hết thời gian này vẫn chưa kết thúc sẽ bị chuyển sang Q_2

Quản lý bộ nhớ

- Phân phối và sắp xếp các process trong bộ nhớ sao cho hệ thống hoạt động hiệu quả.
  - Vd: nạp càng nhiều process vào bộ nhớ càng tốt (gia tăng mức độ đa chương)
- Thông thường, kernel chiếm một phần cố định của bộ nhớ, phần còn lại phân phối cho các process.
- Yêu cầu đối với việc quản lý bộ nhớ
  - Cấp phát vùng nhớ cho các process
  - Tái định vị (relocation): khi swapping,…
  - Bảo vệ: phải kiểm tra truy xuất bộ nhớ có hợp lệ không
  - Chia sẻ: cho phép các process chia sẻ vùng nhớ chung
  - Kết gán địa chỉ nhớ luận lý của process vào địa chỉ thực

Mô hình quản lý bộ nhớ

- Trong chương này, mô hình quản lý bộ nhớ là một mô hình đơn giản, không có bộ nhớ ảo.
- Một process phải được nạp hoàn toàn vào bộ nhớ thì mới được thực thi (ngoại trừ khi dùng kỹ thuật overlay).
- Các cơ chế quản lý bộ nhớ sau đây rất ít (hầu như không còn) được dùng trong các hệ thống hiện đại
  - Phân chia cố định (fixed partitioning)
  - Phân chia động (dynamic partitioning)
  - Phân trang đơn giản (simple paging)
  - Phân đoạn đơn giản (simple segmentation)

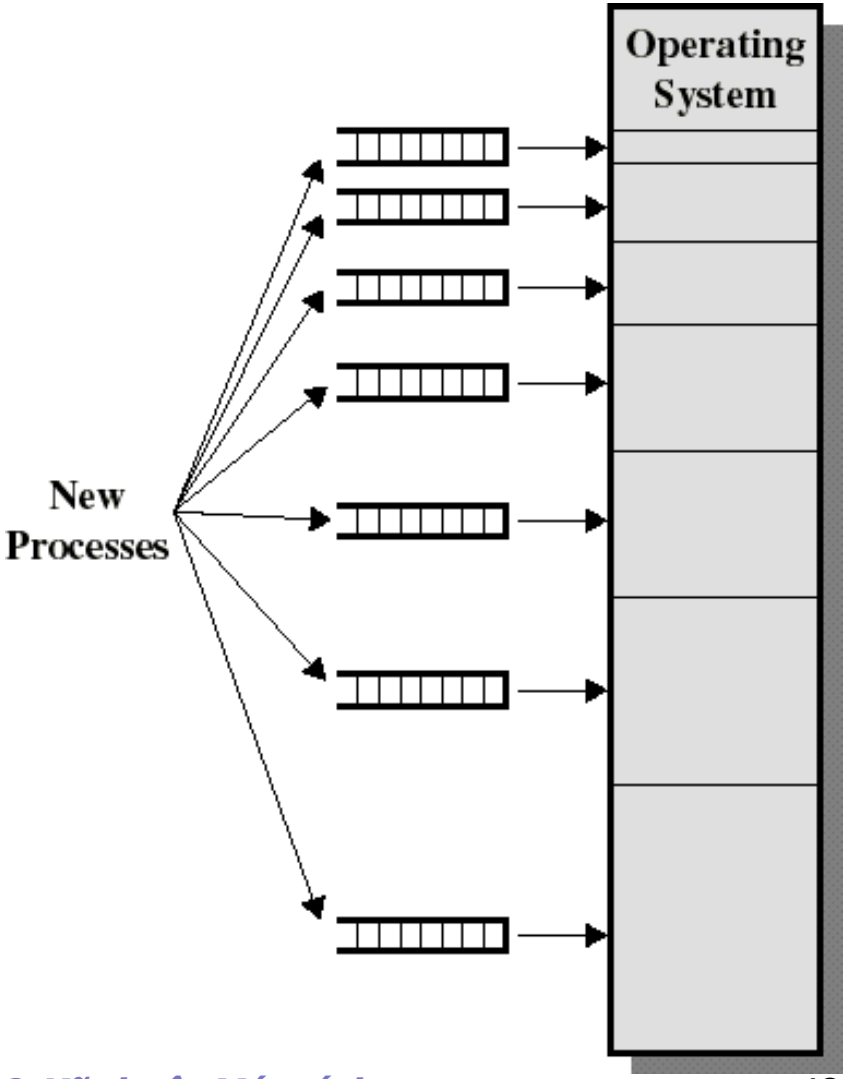
Phân mảnh

- *Phân mảnh ngoại* (external fragmentation)
  - Kích thước không gian nhớ còn trống tuy đủ lớn để thỏa mãn một yêu cầu cấp phát, nhưng không gian nhớ này lại không liên tục
  - Có thể dùng *kết khối* (compacting) để gom lại thành vùng nhớ liên tục.

- *Phân mảnh nội* (internal fragmentation)
  - Kích thước vùng nhớ được cấp phát lớn hơn vùng nhớ yêu cầu.
    - Ví dụ: cấp một khoảng trống 18.464 byte cho một process yêu cầu 18.462 byte.
  - Thường xảy ra khi bộ nhớ thực được chia thành các khối kích thước cố định (fixed-sized block) và các process được cấp phát theo đơn vị khối.

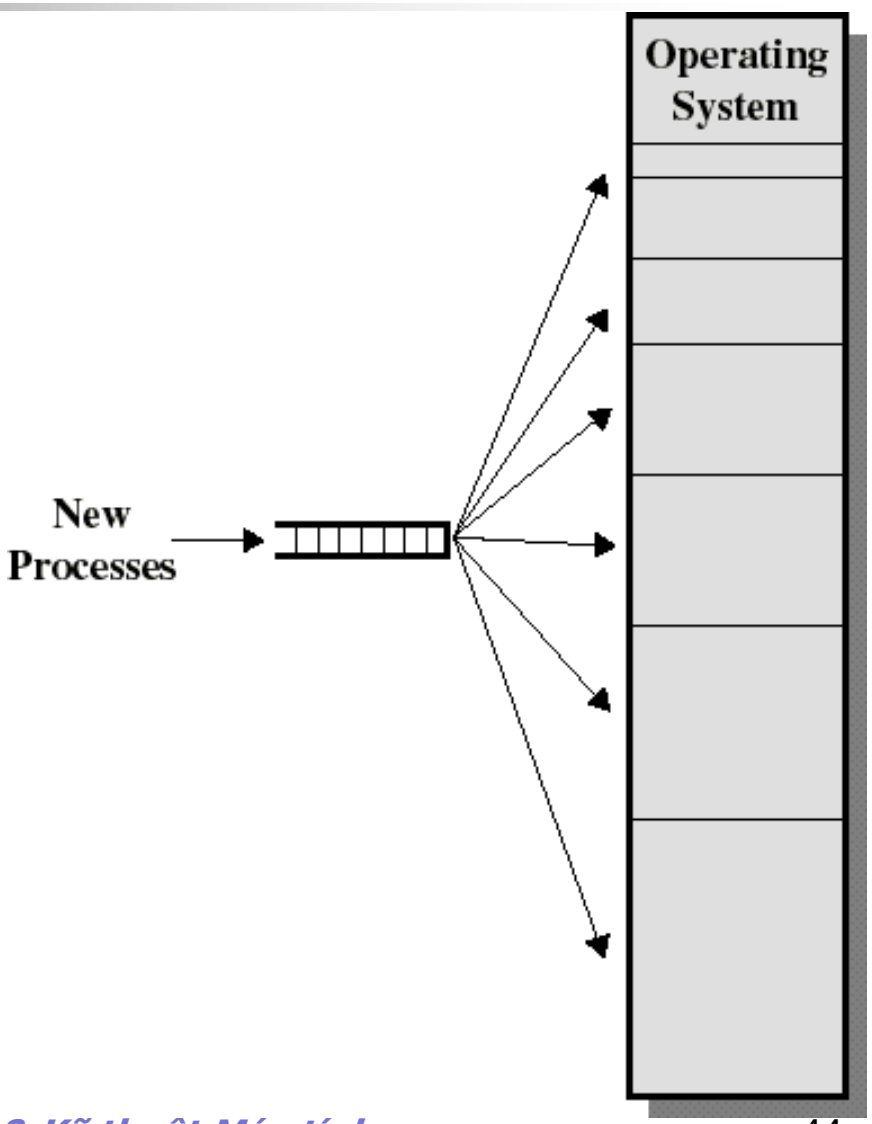
Chiến lược placement (tt.)

- Trường hợp partition có kích thước không bằng nhau: giải pháp 1
  - Gán mỗi process vào partition nhỏ nhất đủ chứa nó
  - Có hàng đợi cho mỗi partition
  - Giảm thiểu phân mảnh nội
  - Vấn đề: có thể có một số hàng đợi trống (vì không có process với kích thước tương ứng) và hàng đợi dày đặc



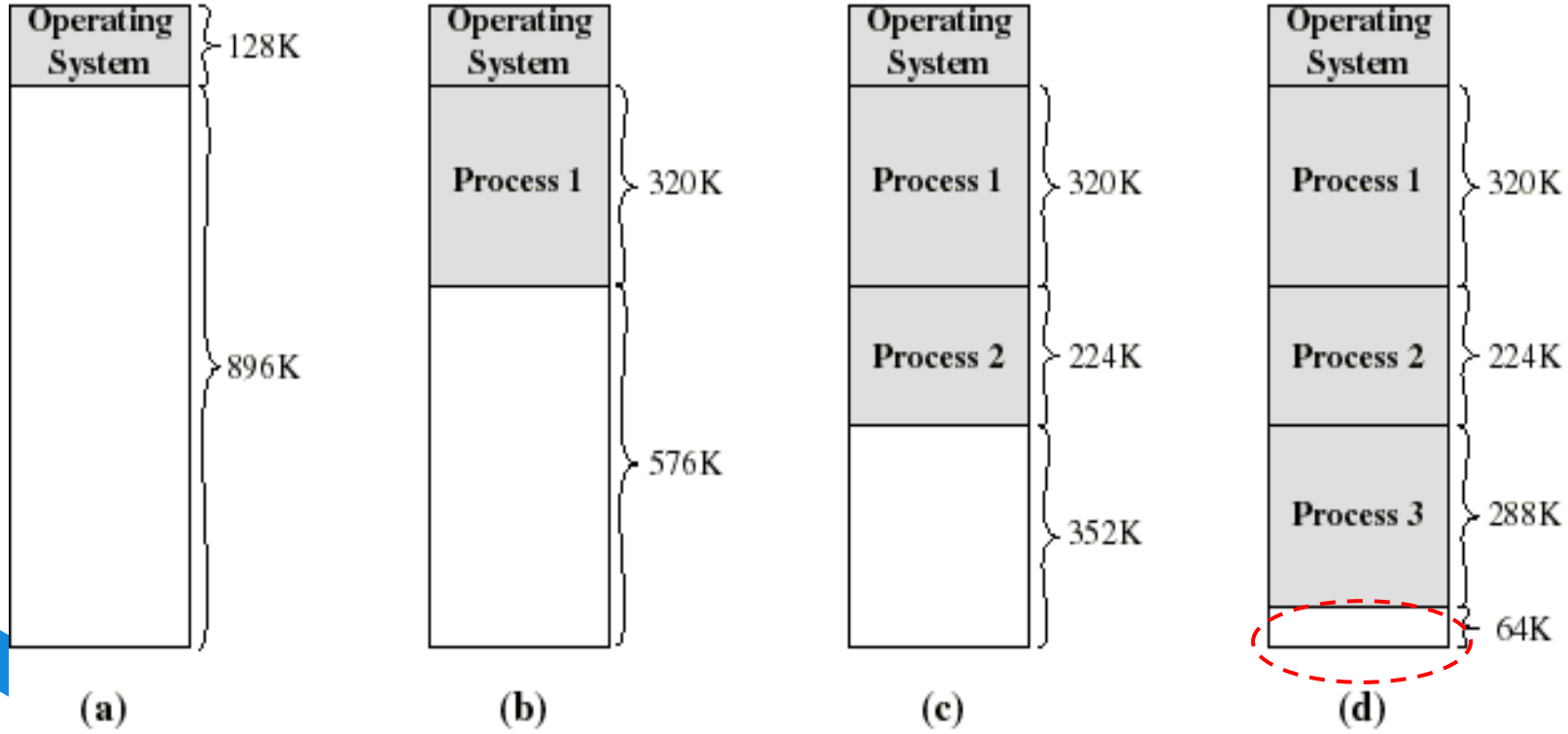
Chiến lược placement (tt.)

- Trường hợp partition có kích thước không bằng nhau: giải pháp 2
  - Chỉ có một hàng đợi chung cho mọi partition
  - Khi cần nạp một process vào bộ nhớ chính ⇒ chọn partition nhỏ nhất còn trống và đủ chứa nó



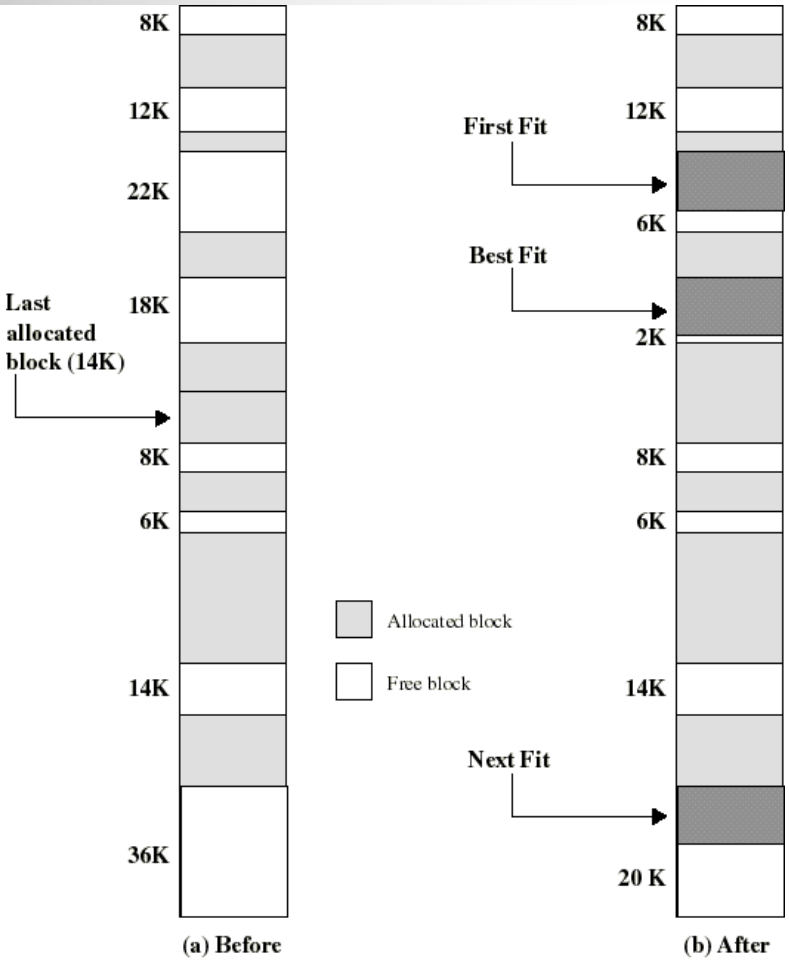
Dynamic partitioning

- Số lượng và vị trí partition không cố định và partition có thể có kích thước khác nhau
- Mỗi process được cấp phát chính xác dung lượng bộ nhớ cần thiết
- Gây ra hiện tượng phân mảnh ngoại



Chiến lược placement khi dynamic partitioning

- Dùng để quyết định cấp phát khối bộ nhớ trống nào cho một process
- Mục tiêu: giảm chi phí compaction
- Các chiến lược placement
  - *Best-fit:* chọn khối nhớ trống nhỏ nhất
  - *First-fit:* chọn khối nhớ trống phù hợp đầu tiên kể từ đầu bộ nhớ
  - *Next-fit:* chọn khối nhớ trống phù hợp đầu tiên kể từ vị trí cấp phát cuối cùng
  - *Worst-fit:* chọn khối nhớ trống lớn nhất

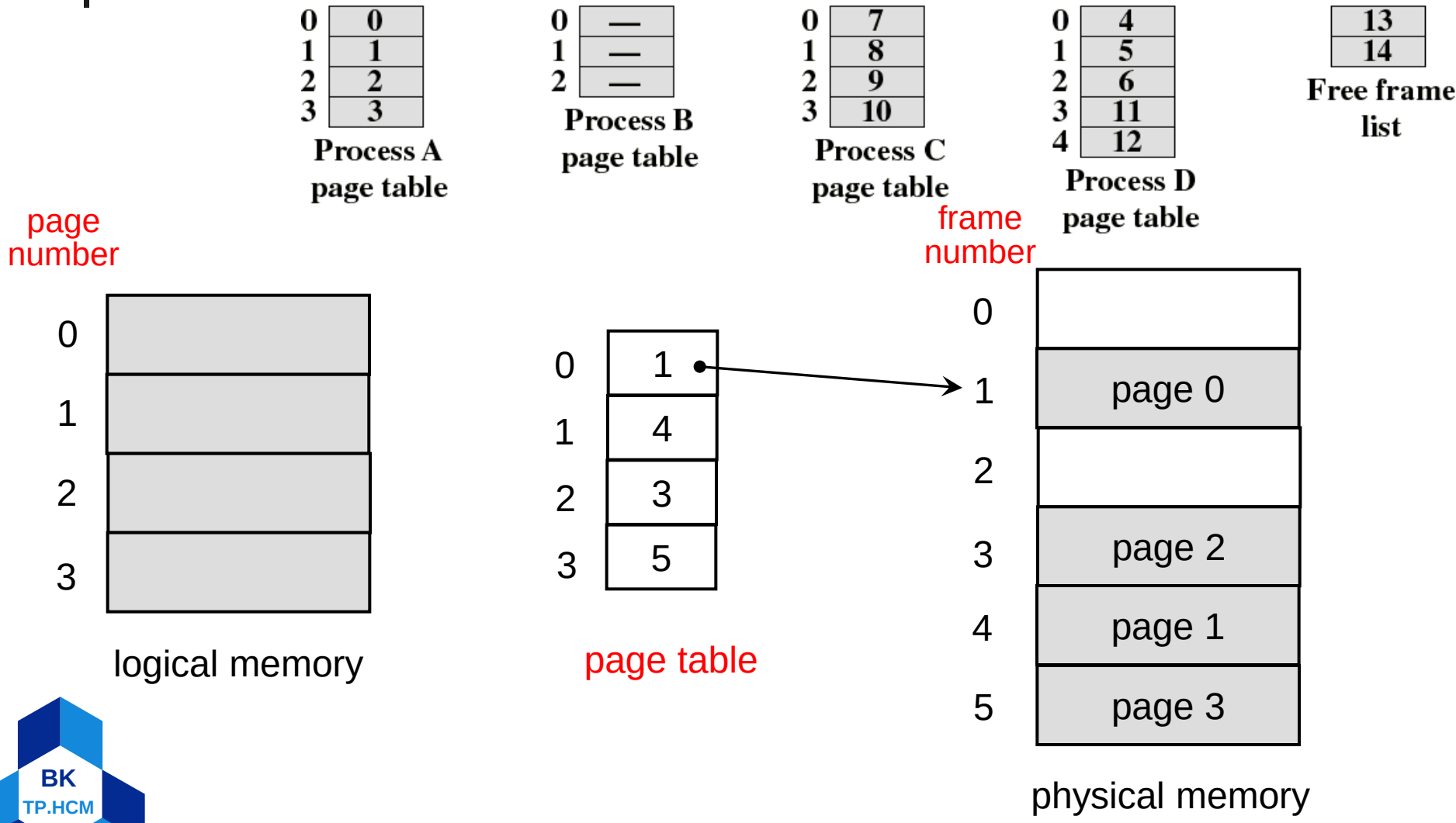


Example Memory Configuration Before and After Allocation of 16 Kbyte Block

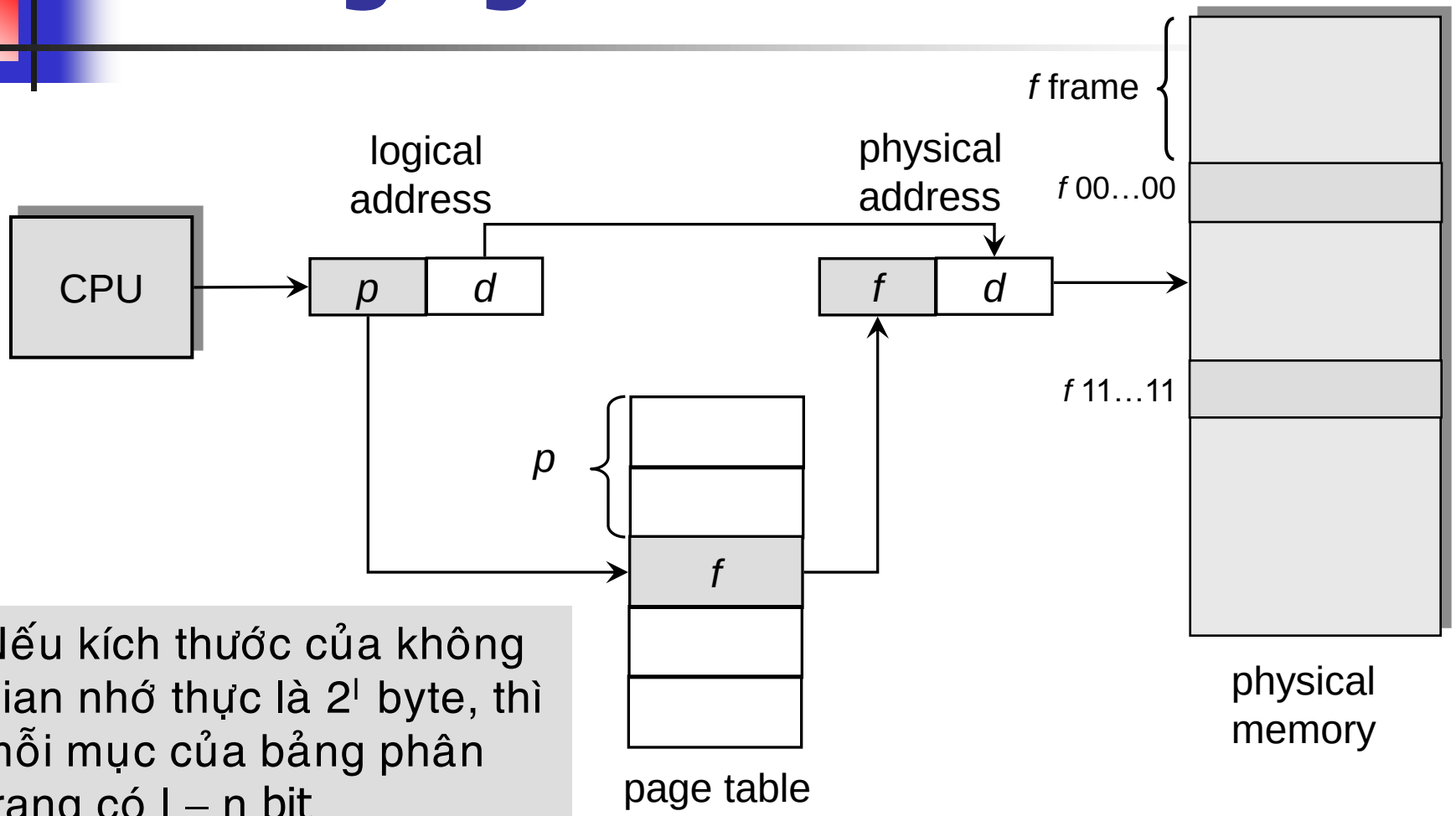
Kỹ thuật phân trang

- Kỹ thuật *phân trang* (paging) cho phép không gian địa chỉ vật lý (physical address space) của một process có thể không liên tục nhau.
- Bộ nhớ thực được chia thành các khối cố định và có kích thước bằng nhau gọi là *frame*.
  - Thông thường kích thước của frame là lũy thừa của 2, từ khoảng 512 byte đến 16 MB.
- *Bộ nhớ luận lý* (logical memory) hay *không gian địa chỉ luận lý* là tập mọi địa chỉ luận lý của một quá trình.
  - Địa chỉ luận lý có thể được quá trình sinh ra bằng cách dùng indexing, base register, segment register,…

Kỹ thuật phân trang (tt.)



Paging hardware

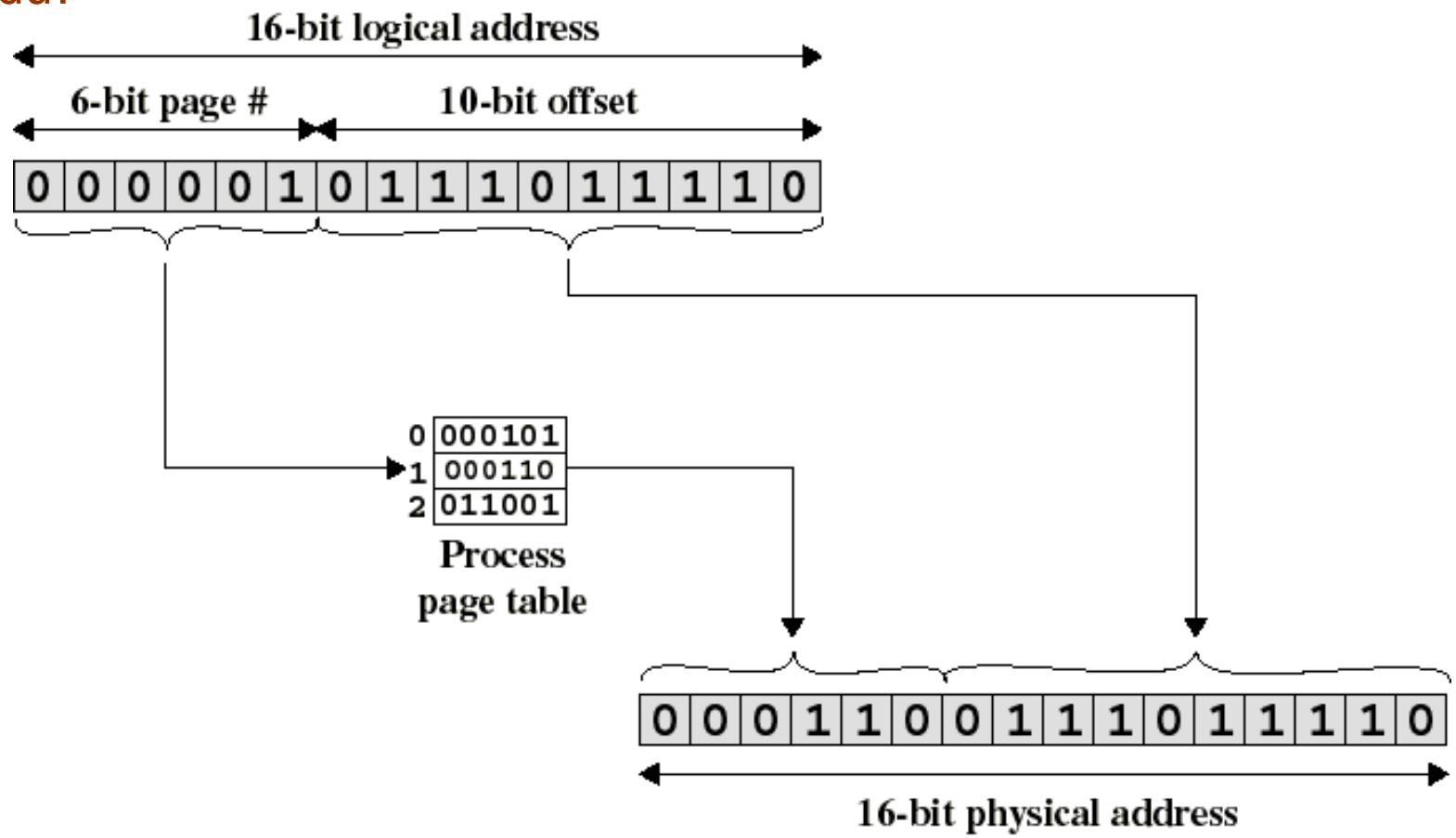


Nếu kích thước của không gian nhớ thực là 2^l byte, thì mỗi mục của bảng phân trang có l – n bit

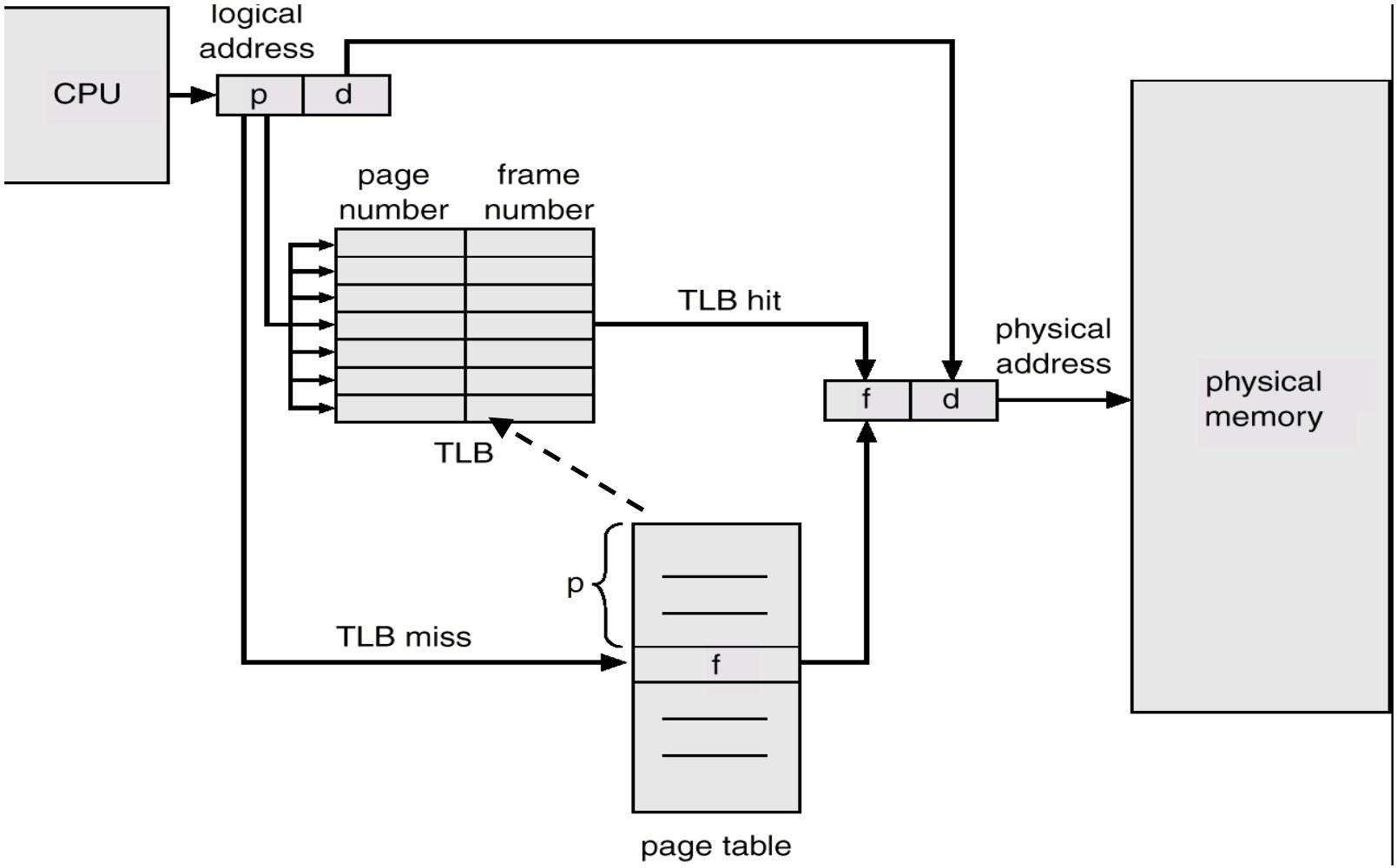
frame number	frame offset
f, *l – n* bit	d, *n* bit

Chuyển đổi địa chỉ nhớ trong paging

- Ví du:

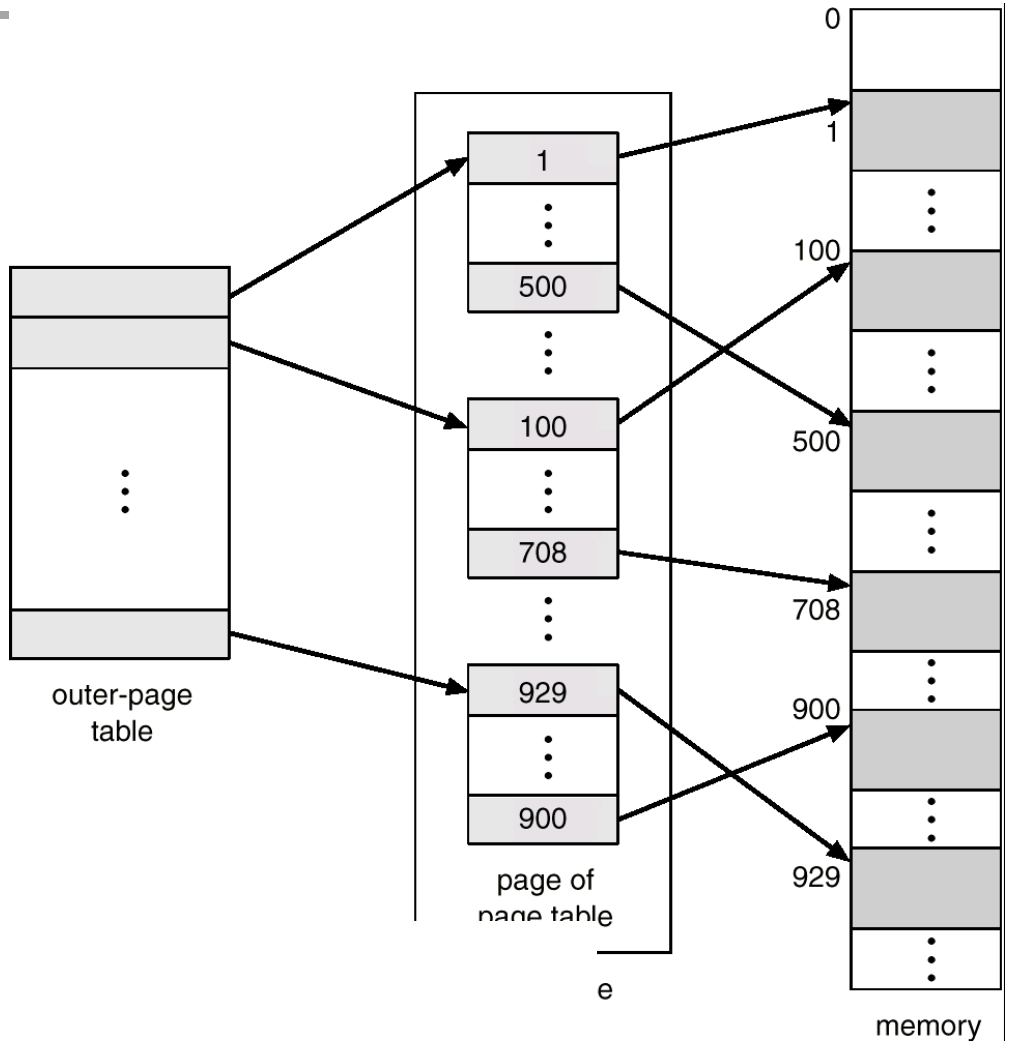


Paging hardware với TLB



Bảng phân trang 2-mức (tt.)

Minh họa



- Có 2^{p1} mục trong bảng phân trang mức 1
- Mỗi bảng phân trang mức 2 chứa 2^{p2} mục

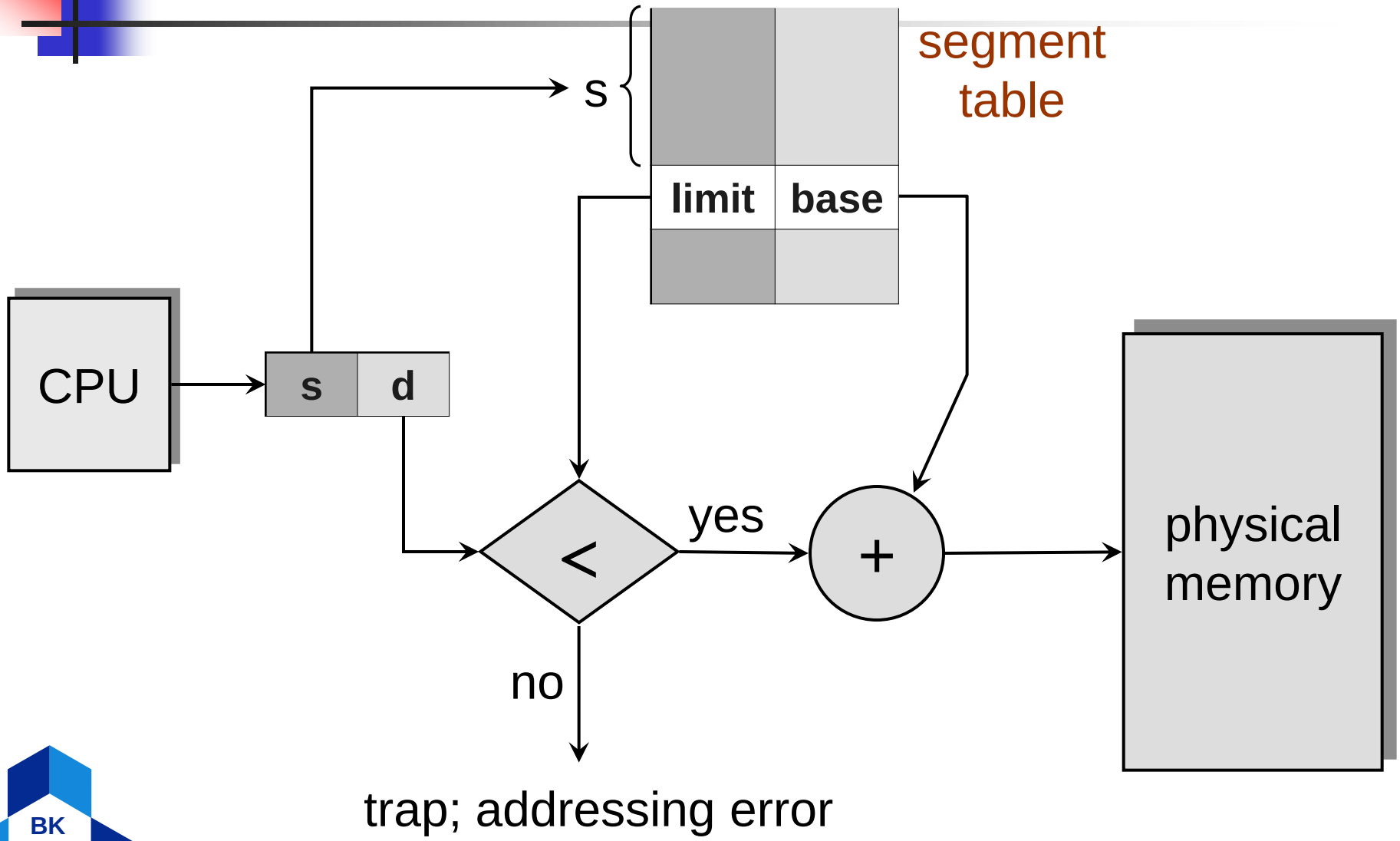
các bảng phân trang mức 2

Phân đoạn

- Nhìn lại kỹ thuật phân trang
  - user view (không gian địa chỉ ảo) tách biệt với không gian bộ nhớ thực. Kỹ thuật phân trang thực hiện phép ánh xạ user-view vào bộ nhớ thực.

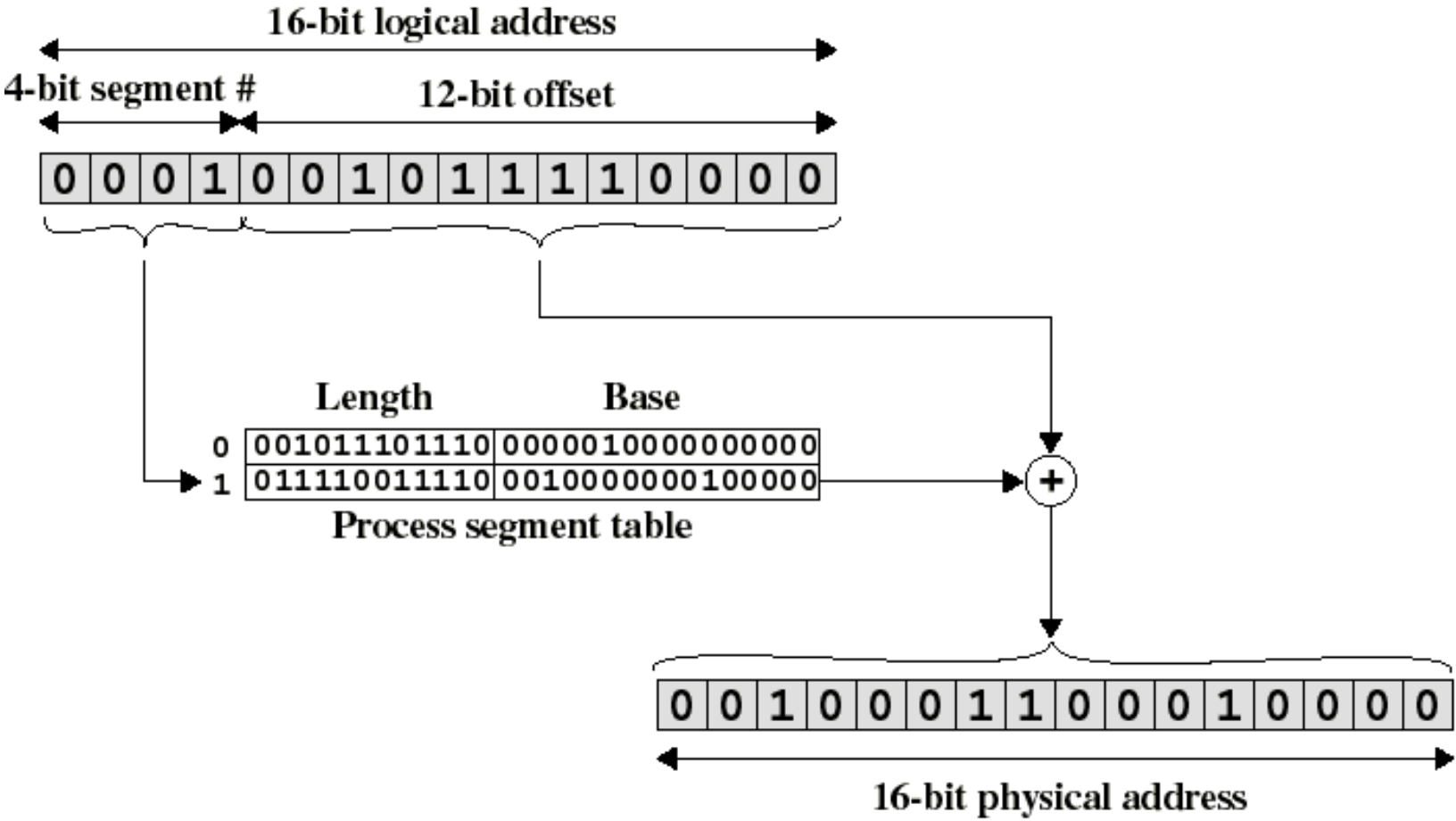
- Trong thực tế, dưới góc nhìn của user, một chương trình cấu thành từ nhiều *đoạn* (segment). Mỗi đoạn là một đơn vị luận lý của chương trình, như
  - main program, procedure, function
  - local variables, global variables, common block, stack, symbol table, arrays,…

Phần cứng hỗ trợ phân đoạn



Chuyển đổi địa chỉ trong kỹ thuật phân đoạn

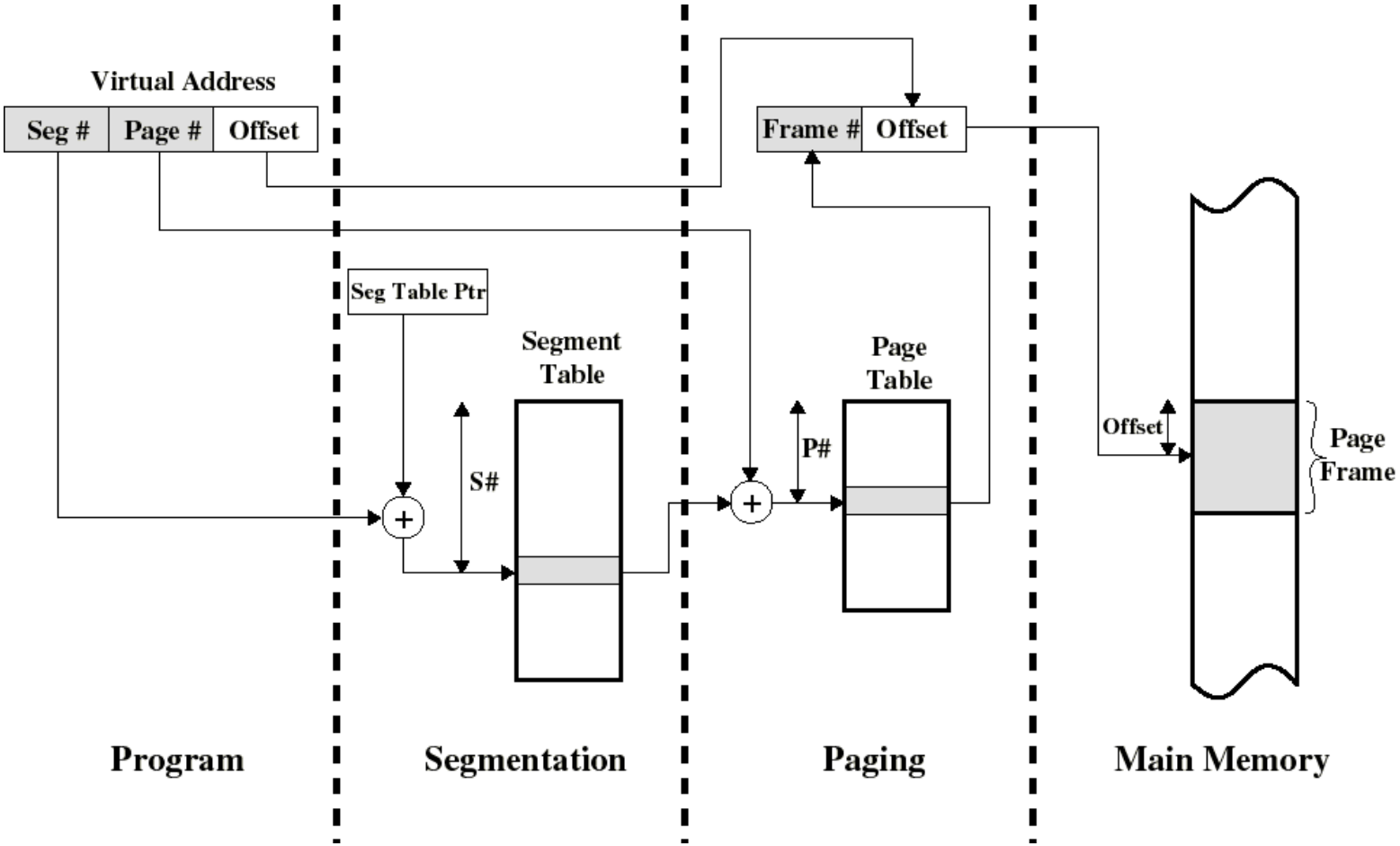
- Ví dụ



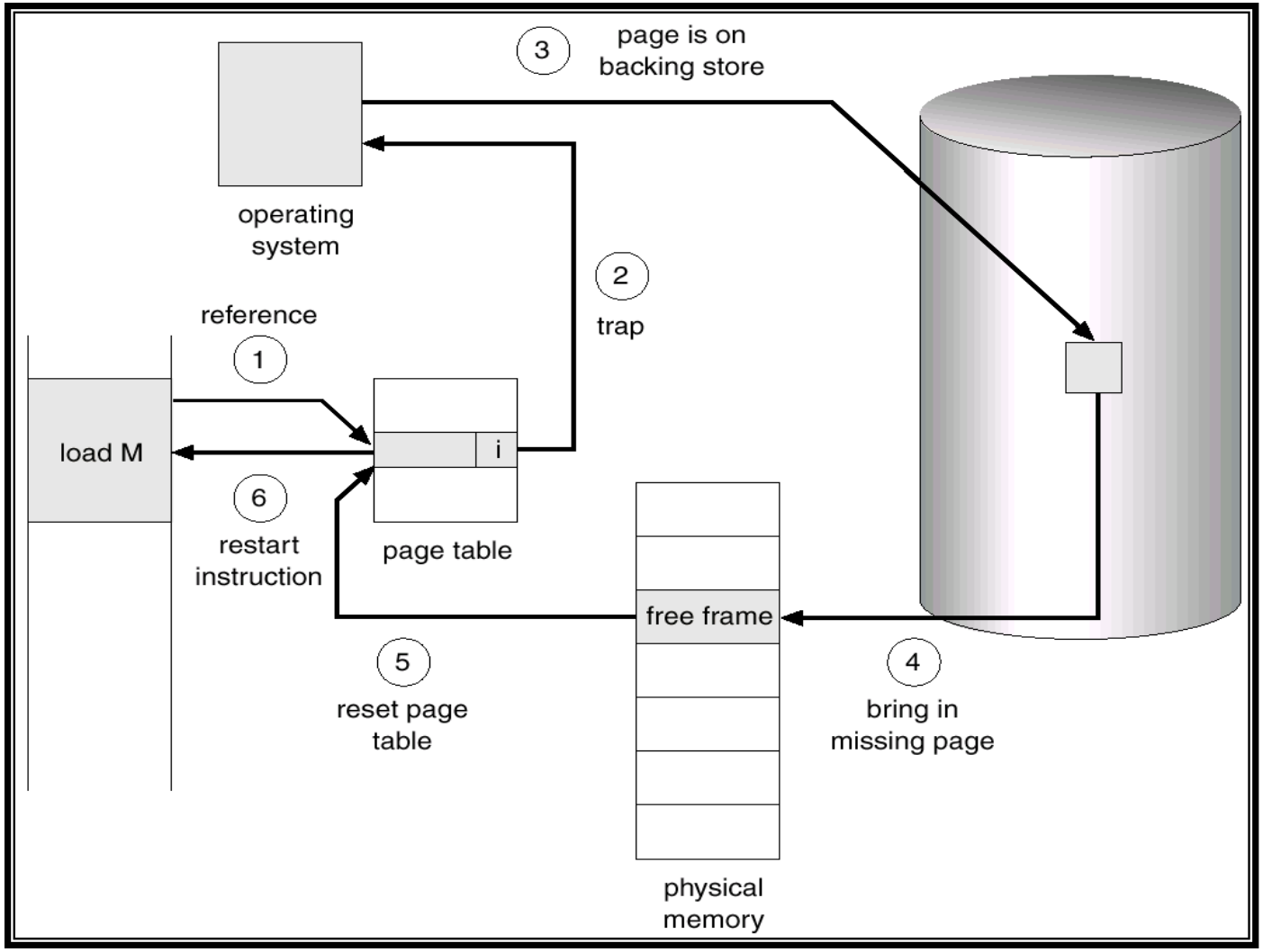
Kết hợp phân trang và phân đoạn

- Kết hợp phân trang và phân đoạn nhằm tận dụng các ưu điểm và hạn chế các khuyết điểm của chúng:
  - Vấn đề của phân đoạn: một đoạn có thể không nạp được vào bộ nhớ do phân mảnh ngoại, mặc dù đủ không gian trống.
  - Ý tưởng giải quyết: paging đoạn, cho phép các page của đoạn được nạp vào các frame không cần nằm liên tục nhau.

Segmentation with paging (tt.)



Page fault và các bước xử lý



Giải thuật thay trang OPT (OPTimal)

- Thay thế trang nhớ sẽ được tham chiếu trong tương lai xa nhất
  - Ví dụ: một process có 5 trang, và được cấp 3 frame

chuỗi tham chiếu trang nhớ

Page address stream	2	3	2	1	5	2	4	5	3	2	5	2
OPT	2	2	2	2	2	2	4	4	4	2	2	2
		3	3	3	3	3	3	3	3	3	3	3
				1	5	5	5	5	5	5	5	5
					F		F			F		

Giải thuật thay trang LRU
(Least Recently Used)

- Thay thế trang nhớ không được tham chiếu lâu nhất
  - Ví dụ: một process có 5 trang, và được cấp 3 frame

chuỗi tham chiếu trang nhớ	2	3	2	1	5	2	4	5	3	2	5	2
OPT	2	2	2	2	2	2	4	4	4	2	2	2
		3	3	3	3	3	3	3	3	3	3	3
				1	5	5	5	5	5	5	5	5
					F		F			F		
LRU	2	2	2	2	2	2	2	2	3	3	3	3
		3	3	3	5	5	5	5	5	5	5	5
				1	1	1	4	4	4	2	2	2
					F		F		F	F		

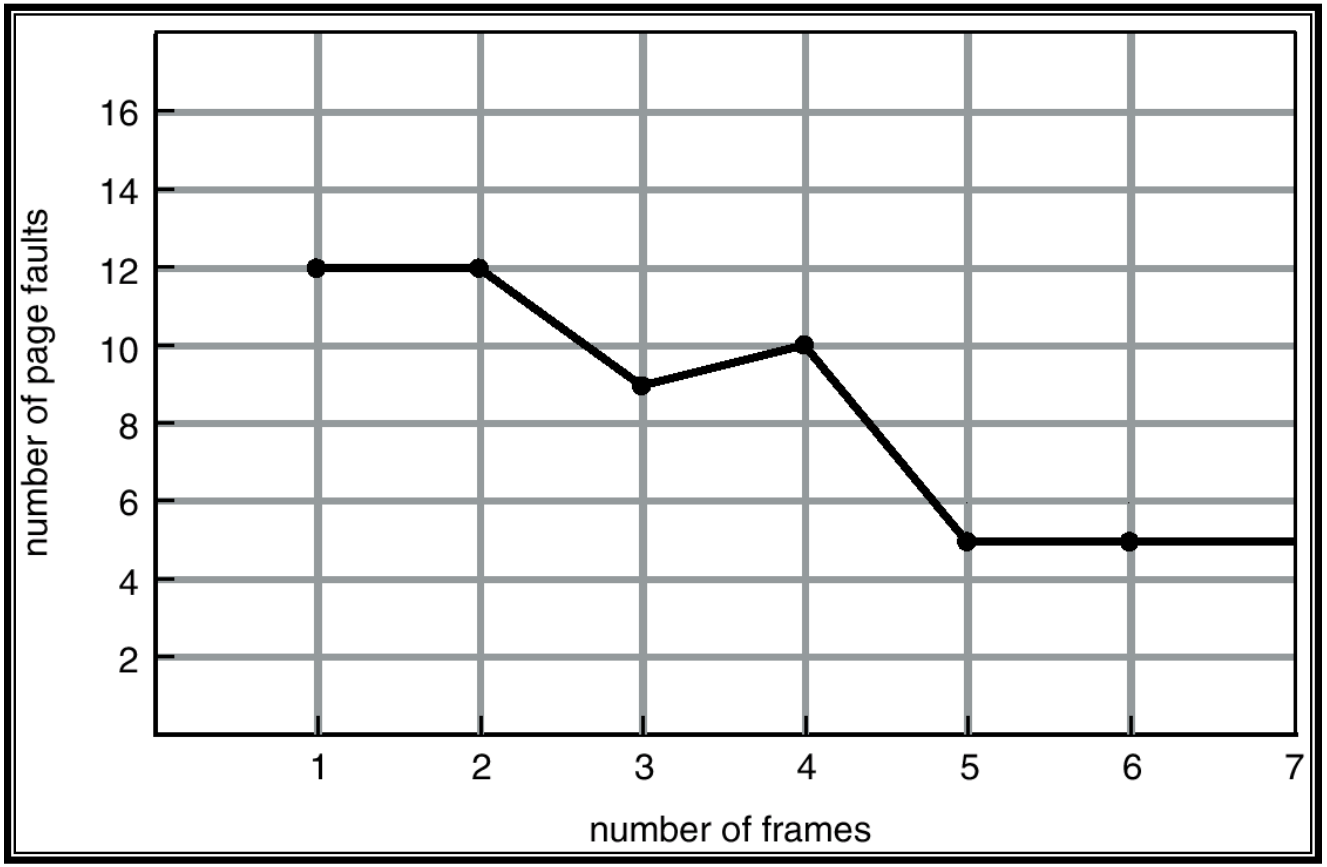
- Mỗi trang được ghi nhận (trong bảng phân trang) thời điểm được tham chiếu. Trang LRU là trang nhớ có thời điểm được tham chiếu nhỏ nhất
  - OS tốn chi phí tìm kiếm trang nhớ LRU này mỗi khi có thay trang. Do vậy, LRU cần sự hỗ trợ của phần cứng. Ít CPU cung cấp đủ sự hỗ trợ phần cứng cho giải thuật LRU.

Giải thuật thay trang FIFO

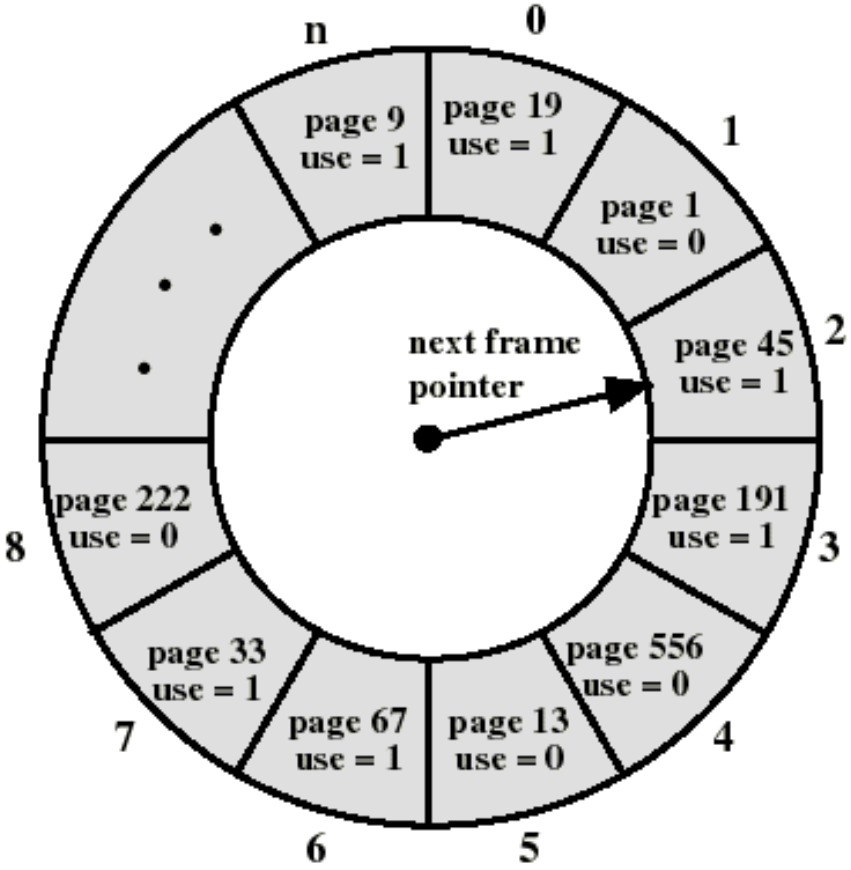
- Xem các frame được cấp phát cho process như là circular buffer
  - Khi bộ đệm đầy, trang nhớ cũ nhất sẽ được thay thế: first-in first-out
  - Một trang nhớ hay được dùng sẽ thường là trang cũ nhất nên hay bị thay thế bởi giải thuật FIFO
  - Hiện thực đơn giản: chỉ cần một con trỏ xoay vòng các frame của process

Giải thuật FIFO: Belady's anomaly

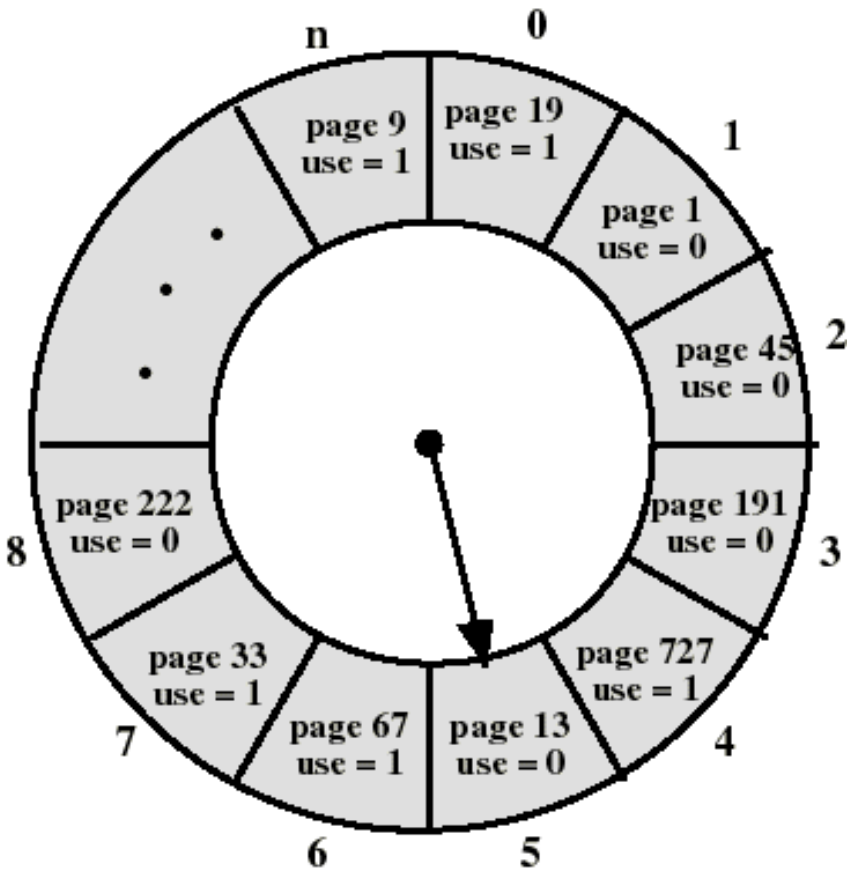
- Số page fault tăng mặc dầu quá trình đã được cấp nhiều frame hơn.



Giải thuật thay trang clock (tt.)



(a) State of buffer just prior to a page replacement



(b) State of buffer just after the next page replacement

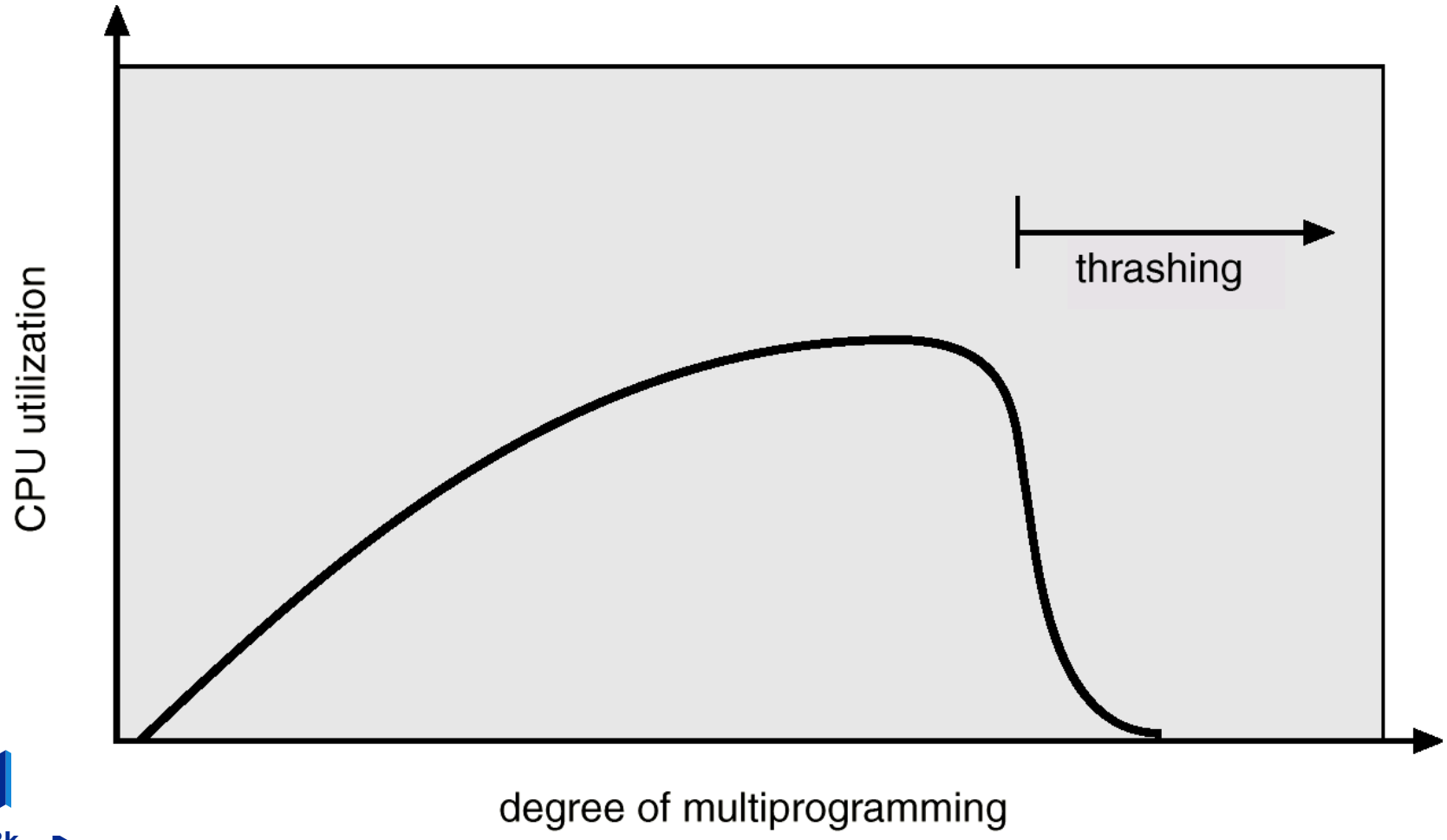
Giải thuật ngẫu nhiên

Not-Used-Recently

- Reference bit = 0 : trang chưa được tham chiếu
  1 : trang đã được tham chiếu
- Modified bit = 0 : trang chưa bị thay đổi
  1 : trang đã bị thay đổi

Group 1	Chưa tham chiếu (0)	Chưa thay đổi (0)
Group 2	Chưa tham chiếu (0)	Thay đổi (1)
Group 3	Tham chiếu (1)	Chưa thay đổi (0)
Group 4	Tham chiếu (1)	Thay đổi (1)

Thrashing diagram



Hệ thống Xuất/Nhập (i/o)

Giao diện I/O cho ứng dụng

- OS cung cấp một giao diện I/O chuẩn hóa, thuần nhất cho các ứng dụng.
  - Ví dụ: một ứng dụng in tài liệu ra máy in mà không cần biết hiệu máy in, đặc tính máy in,…
- Giao diện làm việc là các I/O system call của OS.
- Trình điều khiển thiết bị là "cầu nối" giữa kernel và các bộ điều khiển thiết bị (device controller).

System call yêu cầu I/O

- Blocking: process bị suspended cho đến khi I/O hoàn tất.
  - Dễ dàng sử dụng
  - Không hiệu quả trong một số trường hợp
- Nonblocking: process sẽ tiếp tục thực thi ngay sau lệnh gọi I/O.
  - Ví dụ: data copy (buffered I/O)
  - Thường hiện thực với multithreading
  - Khó kiểm soát kết quả thực hiện I/O
- Asynchronous: process vẫn thực thi trong lúc hệ thống đang thực hiện I/O.
  - Khó sử dụng
  - I/O subsystem báo hiệu cho process khi I/O hoàn tất

Định nghĩa file

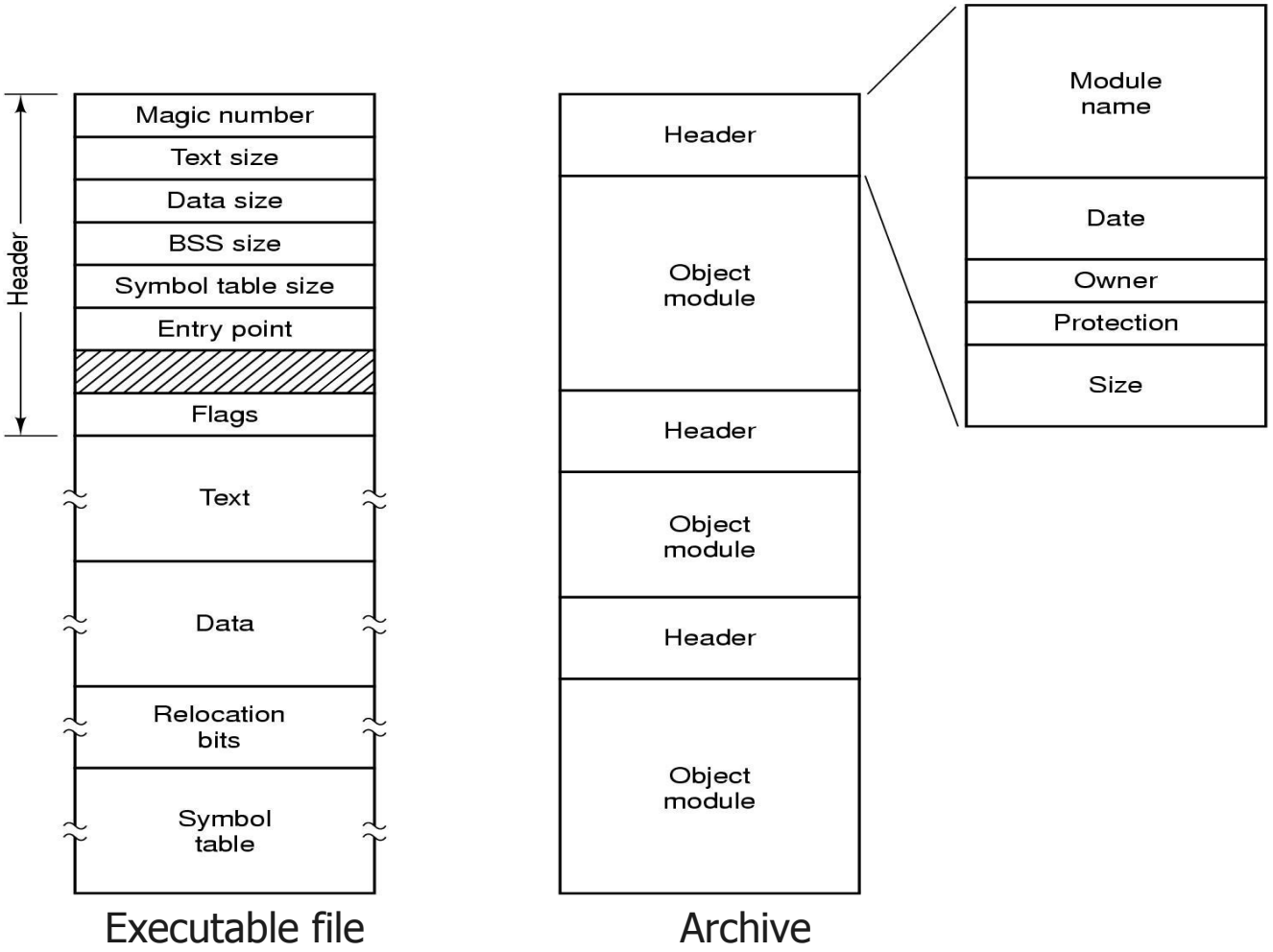
- File là một chuỗi các byte
  - được đặt tên,
  - persistent,
  - các tác vụ lên một file gồm ít nhất là
    - read
    - write

Giao diện I/O cho ứng dụng

- OS cung cấp một giao diện I/O chuẩn hóa, thuần nhất cho các ứng dụng.
  - Ví dụ: một ứng dụng in tài liệu ra máy in mà không cần biết hiệu máy in, đặc tính máy in,…
- Giao diện làm việc là các I/O system call của OS.
- Trình điều khiển thiết bị là "cầu nối" giữa kernel và các bộ điều khiển thiết bị (device controller).

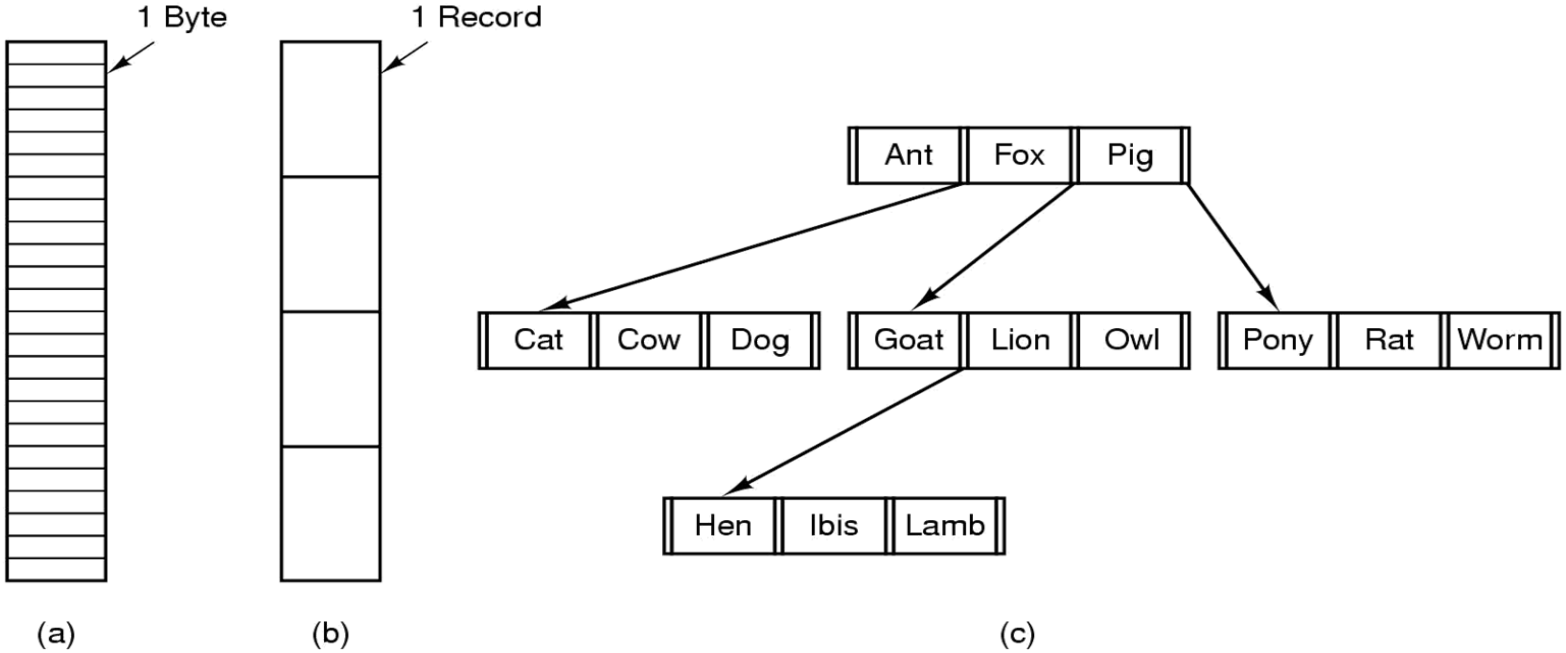
Định dạng file (tt.)

- Ví dụ trong UNIX



Cấu trúc file

- Cấu trúc file: mô hình file để người dùng lập trình
  - Không có cấu trúc: một chuỗi byte (Unix, DOS, Windows), Hình a
  - Cấu trúc record
    - Fixed length, Hình b
    - Variable length: hỗ trợ tìm nhanh chóng một record với key cho trước (IBM mainframe), Hình c



Các tác vụ trên file

- Create
  - Tạo một file mới
- Write
  - Thực hiện tác vụ ghi dữ liệu vào file tại vị trí con trỏ ghi
- Read
  - Thực hiện tác vụ đọc dữ liệu từ file tại vị trí con trỏ đọc
- Reposition
  - Thiết lập con trỏ đọc/ghi đến vị trí do quá trình chỉ định
- Delete
  - Xóa file
- Truncate
  - Giữ lại tất cả các thuộc tính của file, ngoại trừ kích thước file được thiết lập về 0
- Open
  - Quá trình phải mở file trước khi sử dụng
- Close
  - Quá trình phải đóng file sau khi sử dụng

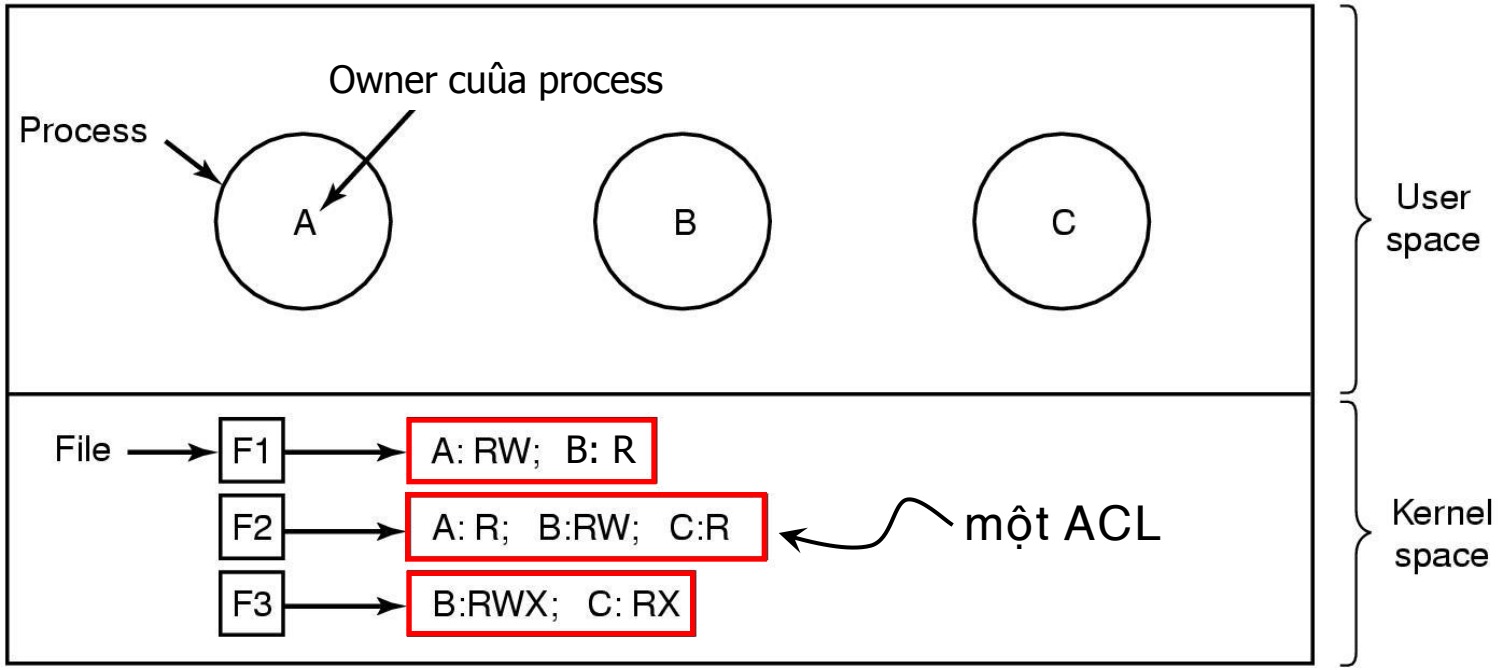
Các tác vụ trên thư mục

- Create
  - Tạo một directory mới
- Delete
  - Xóa một directory trống
- Opendir
  - Quá trình phải mở directory trước khi đọc nó
- Closedir
  - Quá trình phải đóng directory sau khi đọc nó
- Readdir
  - Đọc entry tới của directory
- Link
- Unlink

Bảo vệ (protection)

- Hệ điều hành phải hỗ trợ chủ nhân của file trong việc kiểm soát truy cập file
  - Các tác vụ có thể thực hiện trên file?
  - Những ai được quyền thực hiện thao tác trên file?
- Các quyền truy cập file
  - Read
  - Write
  - Execute, Append, Delete,…
- Cách tiếp cận thông thường
  - Mỗi file có một Access Control List (ACL), gồm các cặp ⟨user, rights⟩
    - Windows NT/2K/XP, Linux
    - user có thể là một nhóm

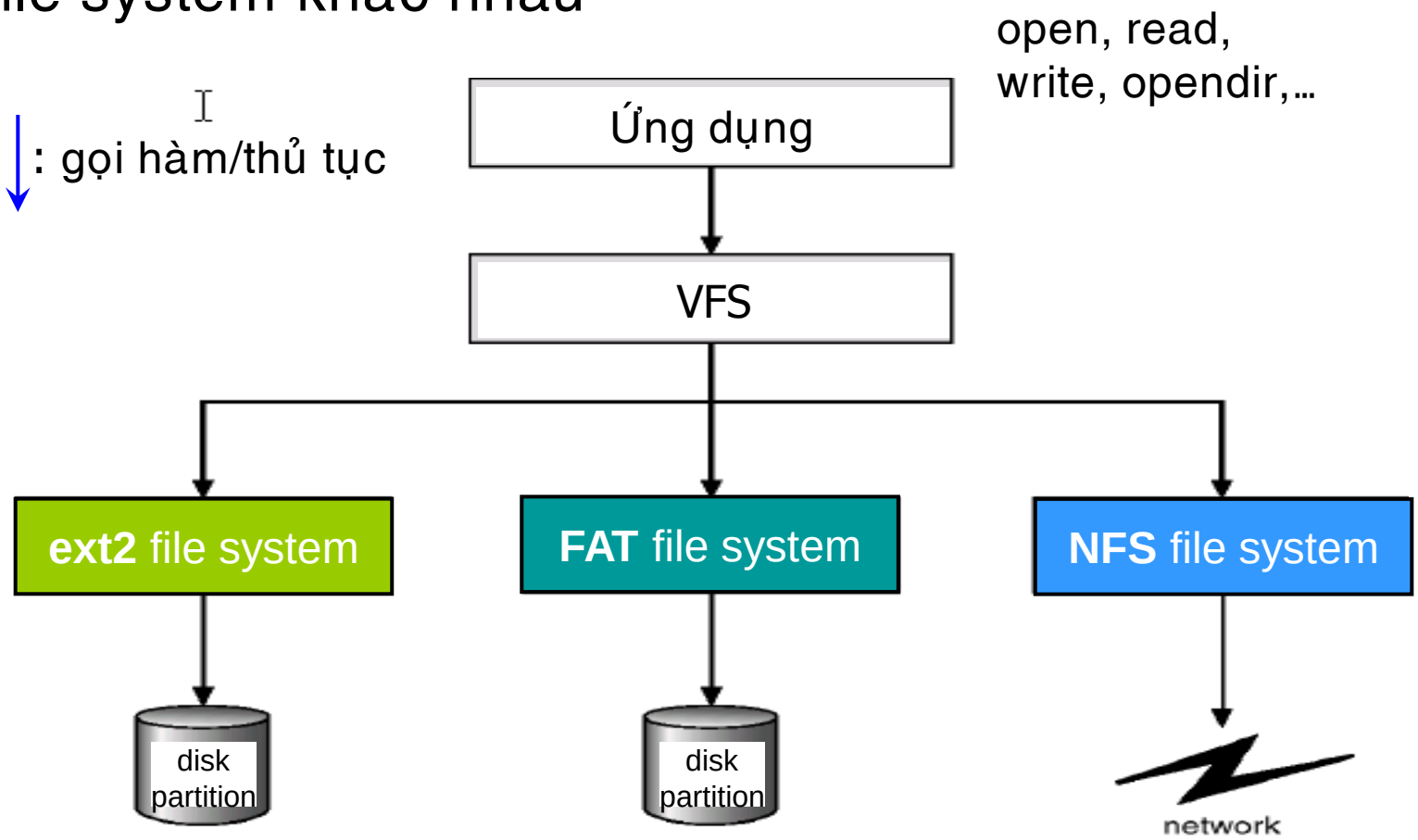
Access Control List



File F1:
- Mọi quá trình của user A có quyền đọc/ghi
- Mọi quá trình của user B có quyền đọc

VFS (Virtual File System)

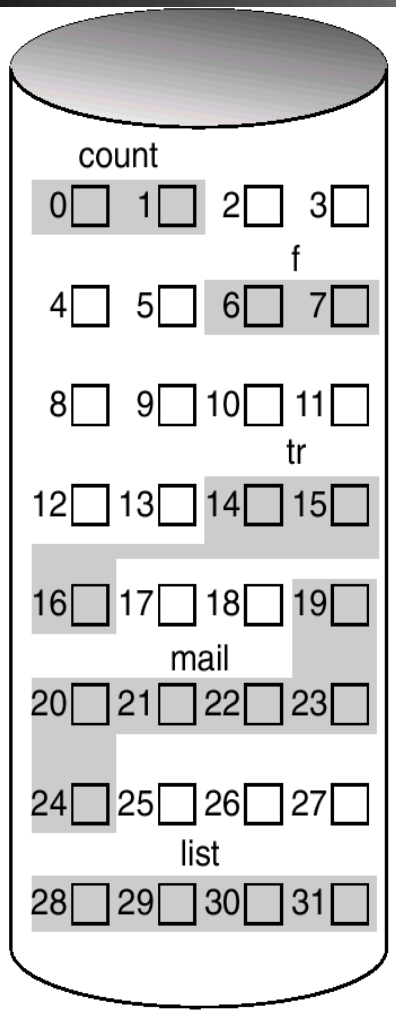
- VFS cung cấp một giao diện đồng nhất đến các loại file system khác nhau



Hiện thực file

- Cấp phát không gian lưu trữ cho file/directory, mục tiêu:
  - sử dụng không gian đĩa hữu hiệu
  - truy cập file nhanh
- Các phương pháp cấp phát phổ biến
  - Cấp phát *liên tục* (contiguous allocation)
  - Cấp phát *theo danh sách liên kết* (linked list allocation)
  - Cấp phát *dùng chỉ mục* (indexed allocation)

Cấp phát liên tục

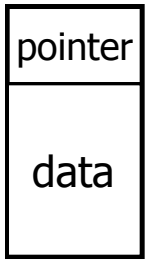


directory

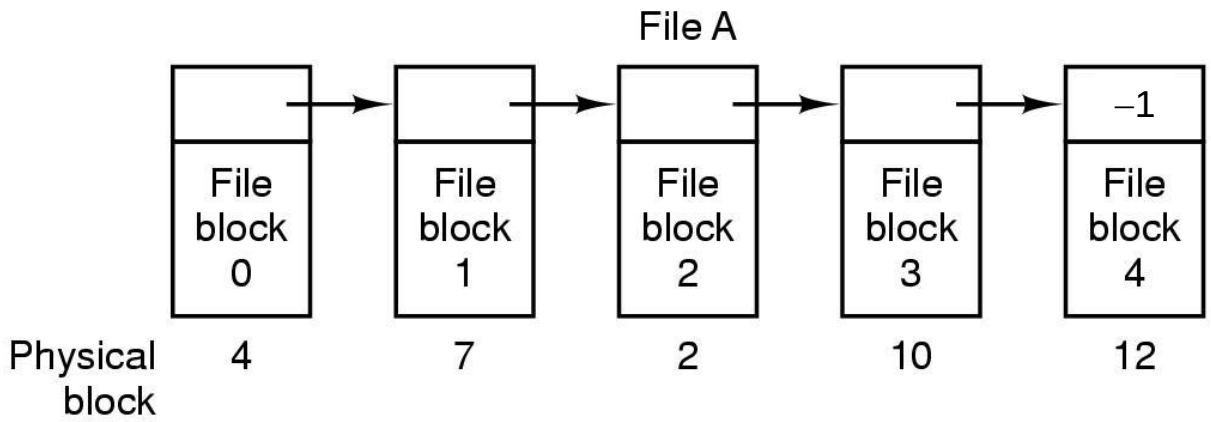
file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

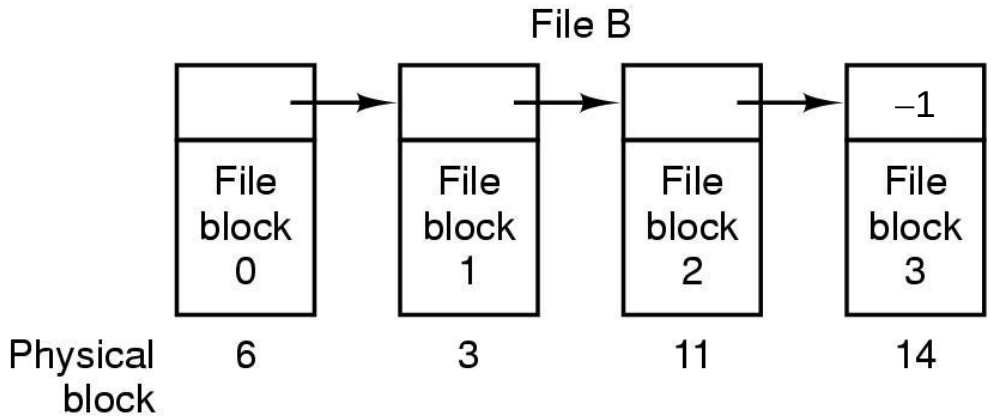
- Seek time? Di chuyển đầu đọc?
- Có thể truy xuất ngẫu nhiên một block của file: block nr = start + block offset
- Phân mảnh ngoại
- Vấn đề khi tạo file mới và khi cần thêm block cho file
- Ứng dụng: ISO-9660 (CDROM)

Cấp phát theo danh sách liên kết



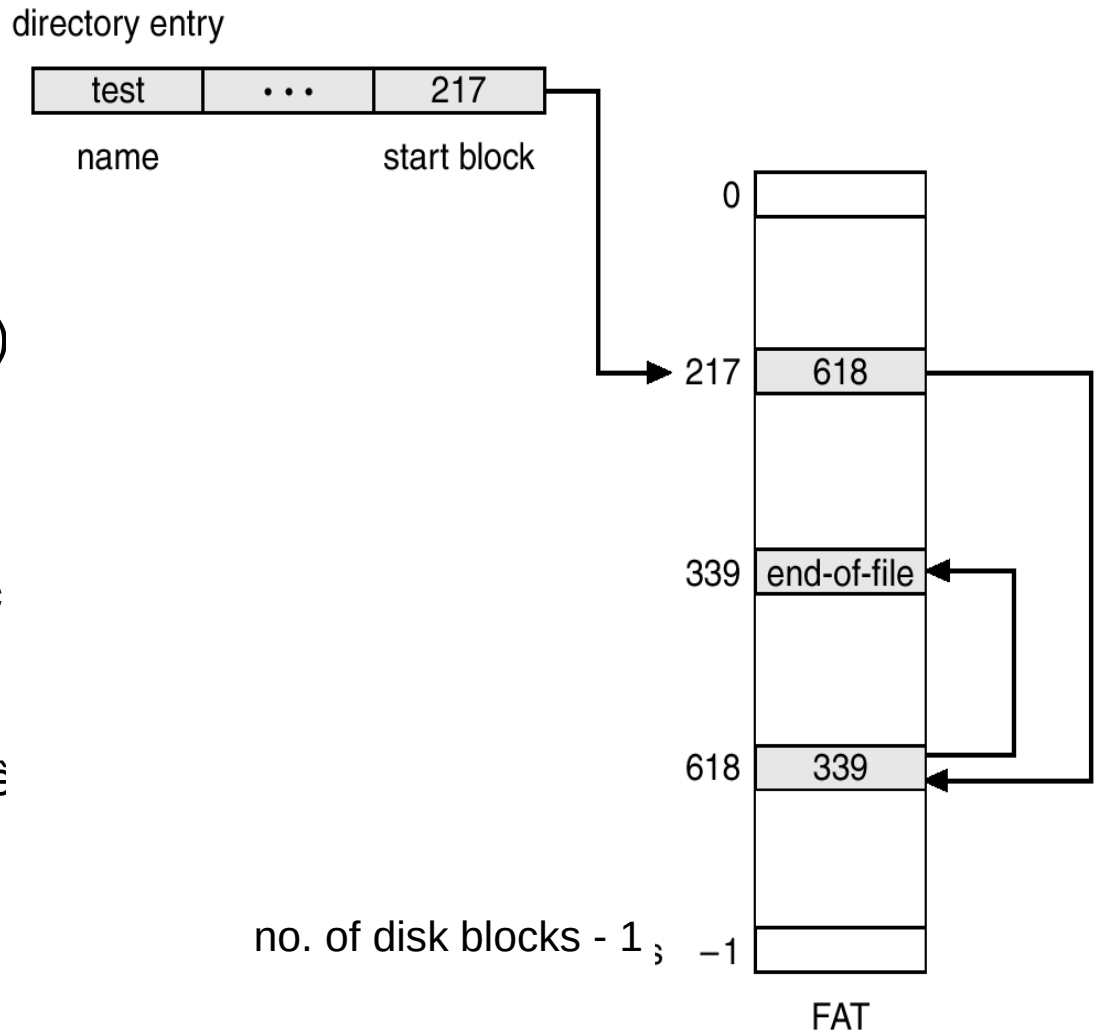
layout cuûa block





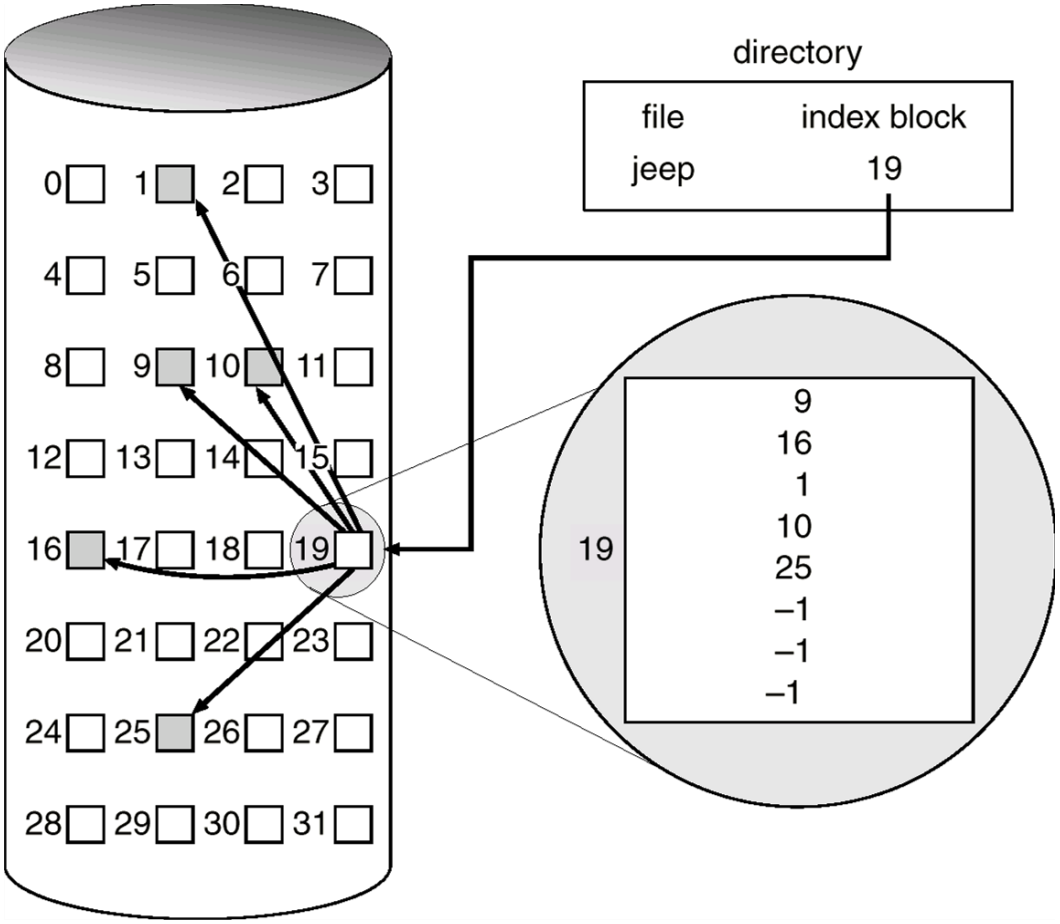
FAT – *một hiện thực cấp phát theo danh sách liên kết:*

- Nhưng không lưu con trỏ đến file block tiếp theo trong block chứa dữ liệu file
- FAT (File Allocation Table)
  - Mỗi block đĩa được tượng trưng bởi một entry trong FAT
  - Block với block nr i được tượng trưng bởi entry với chỉ số (index) i
    - Entry chứa block nr kế tiếp trong file, nếu file gồm nhiều block

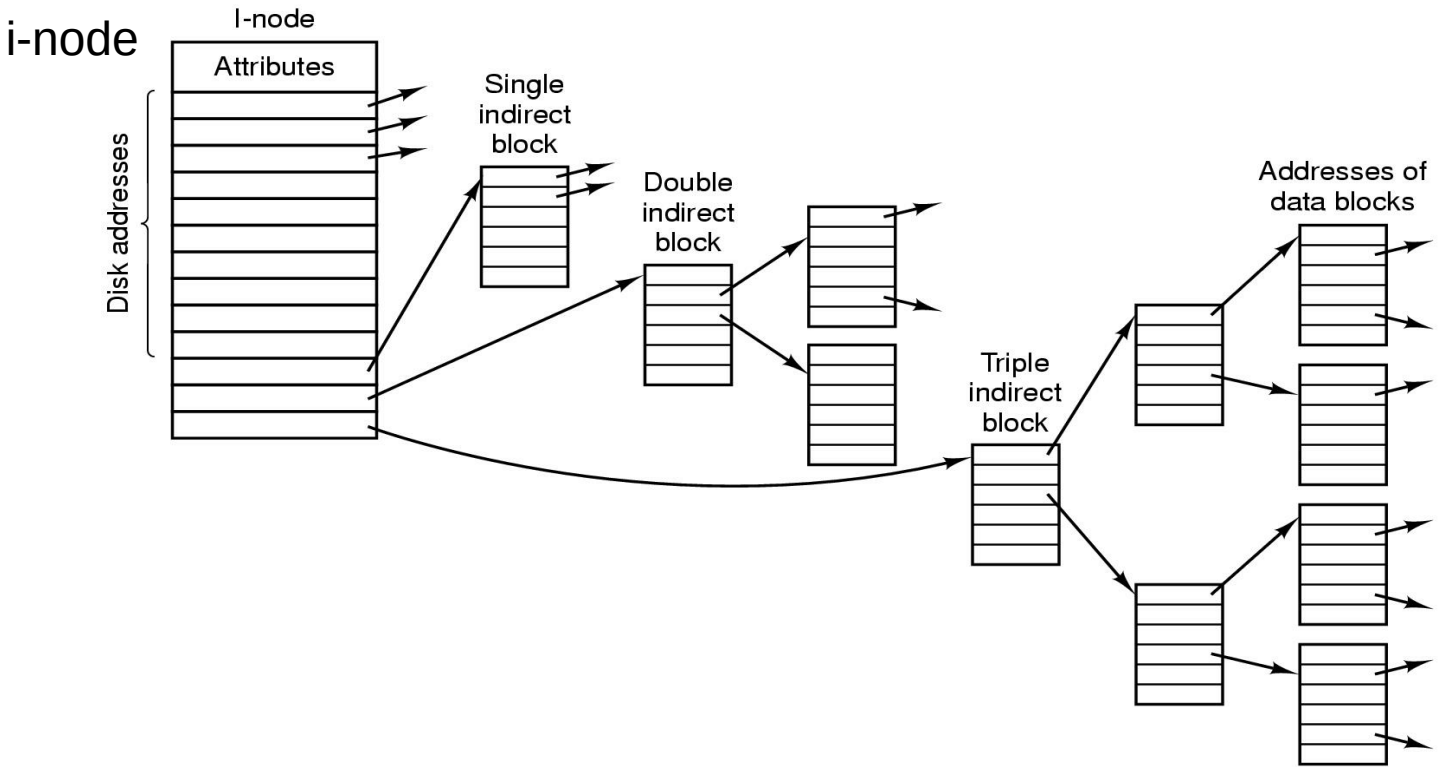


Cấp phát dùng chỉ mục

- Bảng index (index block)
  - chứa địa chỉ các block của file
  - thứ tự các địa chỉ trên trong bảng cũng là thứ tự các block trong file



i-node – một hiện thực của index block



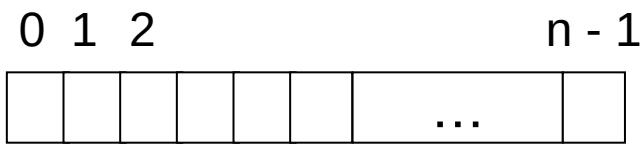
- UNIX v7 i-node: 13 pointers
- Linux ext2 i-node: 15 pointers

Quản lý không gian trống

Các phương pháp

- Bit vector (bit map)
- Linked list
- Grouping
- Counting

Phương pháp bit vector (bit map)



$$bit[\,i\,] = \begin{cases} 0 \Rightarrow \text{block i còn trống} \\ 1 \Rightarrow \text{block i đã được cấp} \end{cases}$$

Ví dụ:

bit vector 00111100…

⇔

block 0, 1 trống

block 2, 3, 4, 5 đã được cấp

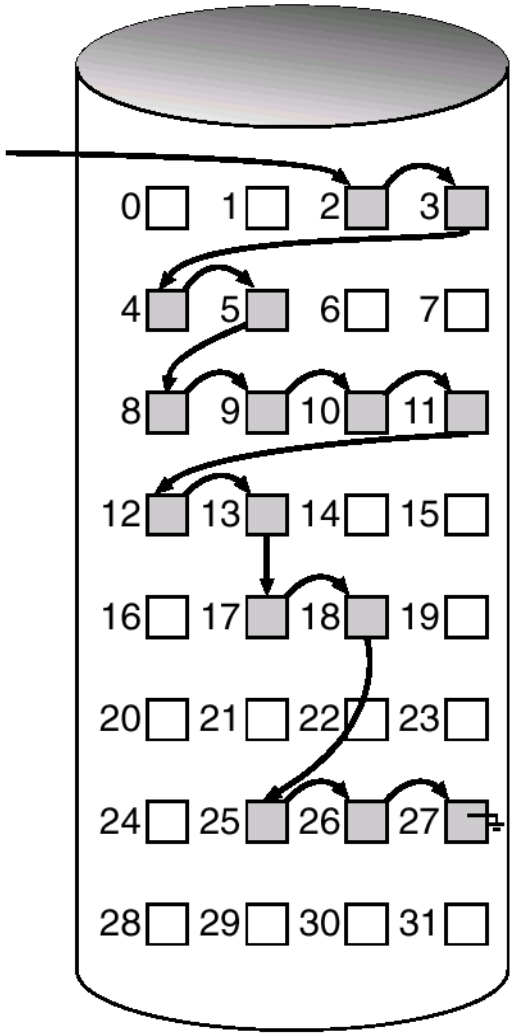
block 6, 7 trống

…

- Ưu: Đơn giản và hiệu quả khi cần tìm khối trống đầu tiên hoặc chuỗi khối trống liên tục
  - Thao tác trên bit
- Khuyết: Cần không gian lưu trữ. Ví dụ
  - Kích thước block = 2^{12} bytes
  - Kích thước đĩa = 2^{30} bytes
  - $n = 2^{30}/2^{12} = 2^{18}$ bit (32KB)

Phương pháp dùng linked list

- Phương pháp
  - Liên kết các khối trống với nhau
  - Chỉ cần giữ con trỏ đến khối nhớ trống đầu tiên trên đĩa hoặc cache trong bộ nhớ chính để tăng tốc
- Ưu: Ít lãng phí không gian đĩa
- Khuyết: Không hiệu quả; trong trường hợp xấu nhất phải duyệt toàn bộ đĩa để tìm không gian trống liên tục

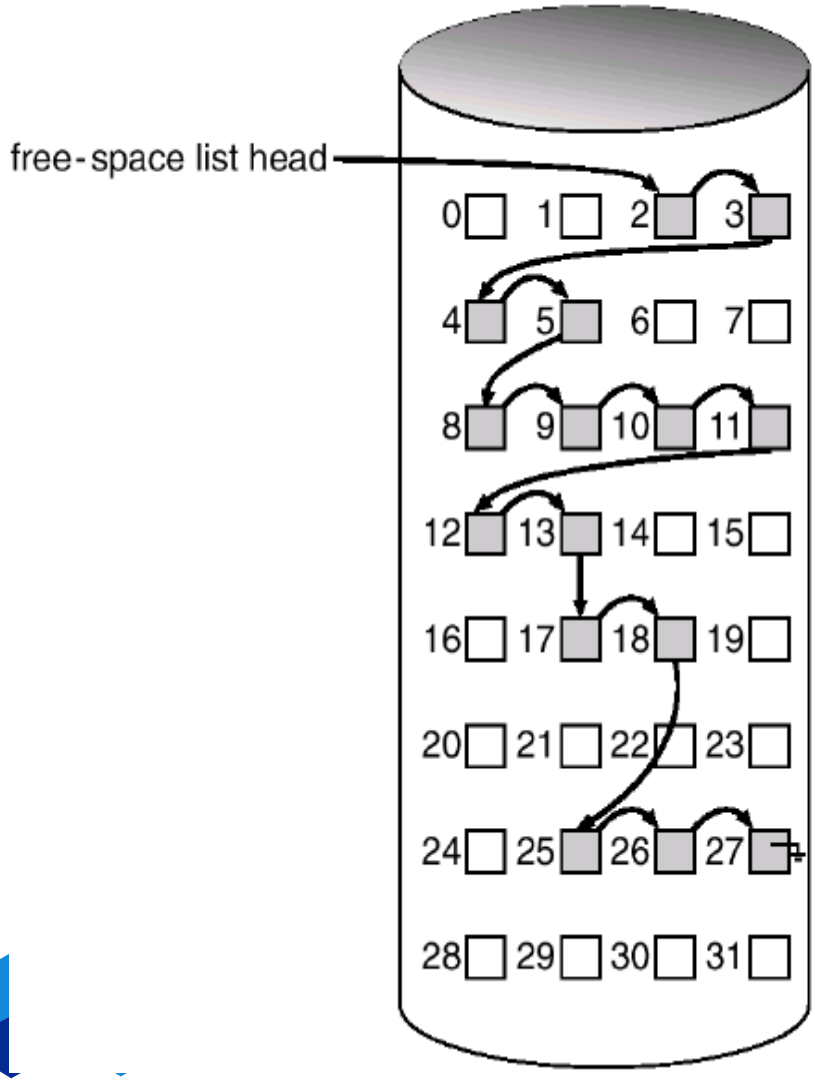


Grouping và counting

- Phương pháp grouping
  - Địa chỉ của n khối trống được lưu trong khối trống đầu tiên.
  - Khối nhớ thứ n chứa địa chỉ của n khối nhớ trống kế tiếp.
- Phương pháp counting
  - Tổ chức bảng chỉ mục
    - mỗi entry: địa chỉ của khối trống đầu tiên trong nhóm khối trống liên tục và một số đếm số lượng khối trống.
  - Có thể cấp phát hoặc thu hồi đồng thời nhiều khối nhớ liên tục.

Grouping và counting (tt.)

- Ví dụ: Phương pháp linked list



- Phương pháp grouping: n = 3

  Block 2 lưu 3, 4, 5
  Block 5 lưu 8, 9, 10
  Block 10 lưu 11, 12, 13
  Block 13 lưu 17, 28, 25
  Block 25 lưu 26, 27

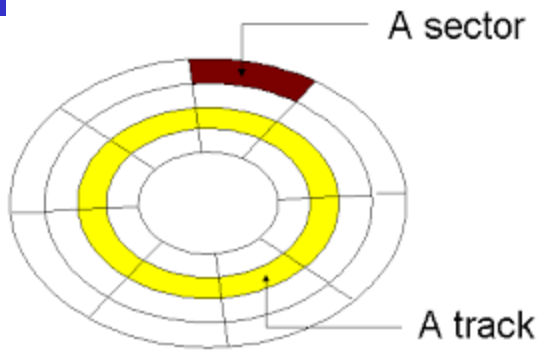
- Phương pháp counting: nội dung index block

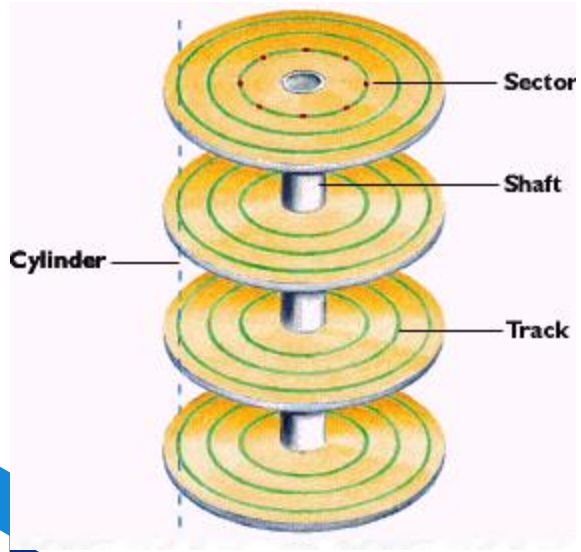
  2 4
  8 6
  17 2
  25 3

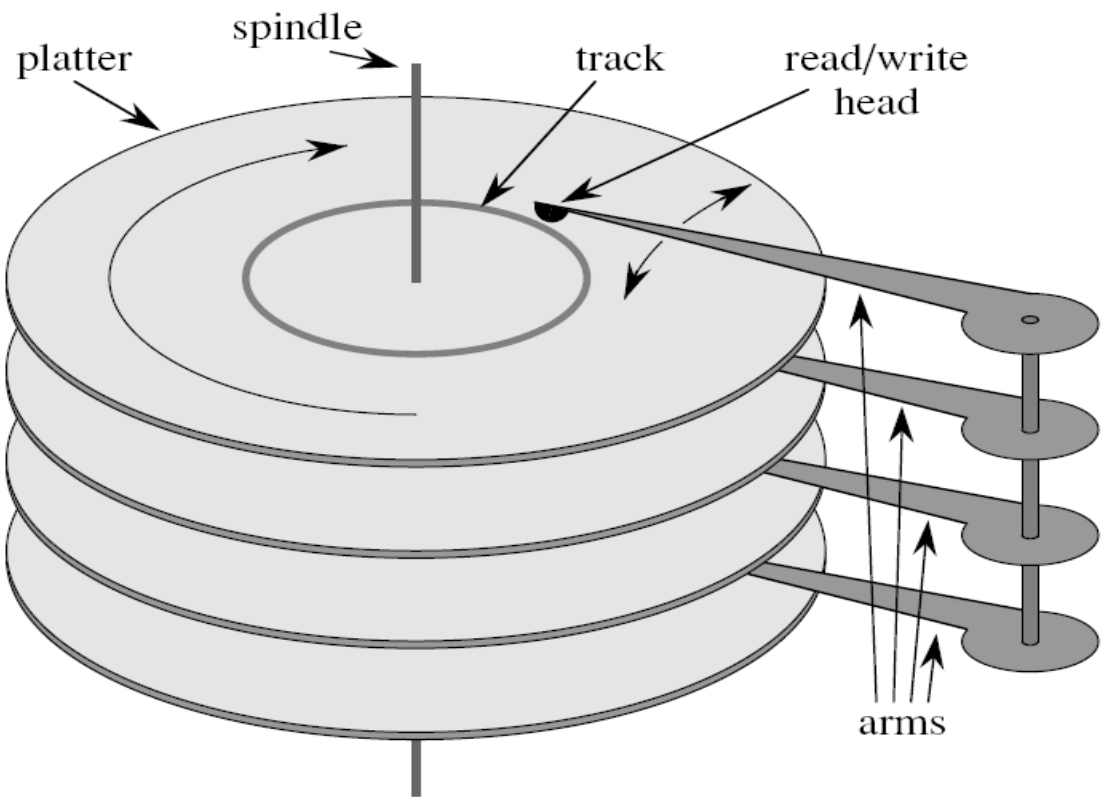
Đĩa cứng: Hệ thống tập tin

- Các giải thuật định thời truy cập đĩa
- Định dạng, phân vùng, raw disk
- RAID (Redundant Arrays of Independent (*Inexpensive*) Disks)

Bên trong đĩa cứng







Các tham số của đĩa

- Thời gian đọc/ghi dữ liệu trên đĩa bao gồm:
  - **Seek time**: thời gian di chuyển đầu đọc để định vị đúng track/cylinder, phụ thuộc tốc độ/cách di chuyển của đầu đọc
  - **Rotational delay** (latency): thời gian đầu đọc chờ đến đúng sector cần đọc, phụ thuộc tốc độ quay của đĩa
  - **Transfer time**: thời gian chuyển dữ liệu từ đĩa vào bộ nhớ hoặc ngược lại, phụ thuộc băng thông kênh truyền giữa đĩa và bộ nhớ
- Disk I/O time = seek time + rotational delay + transfer time

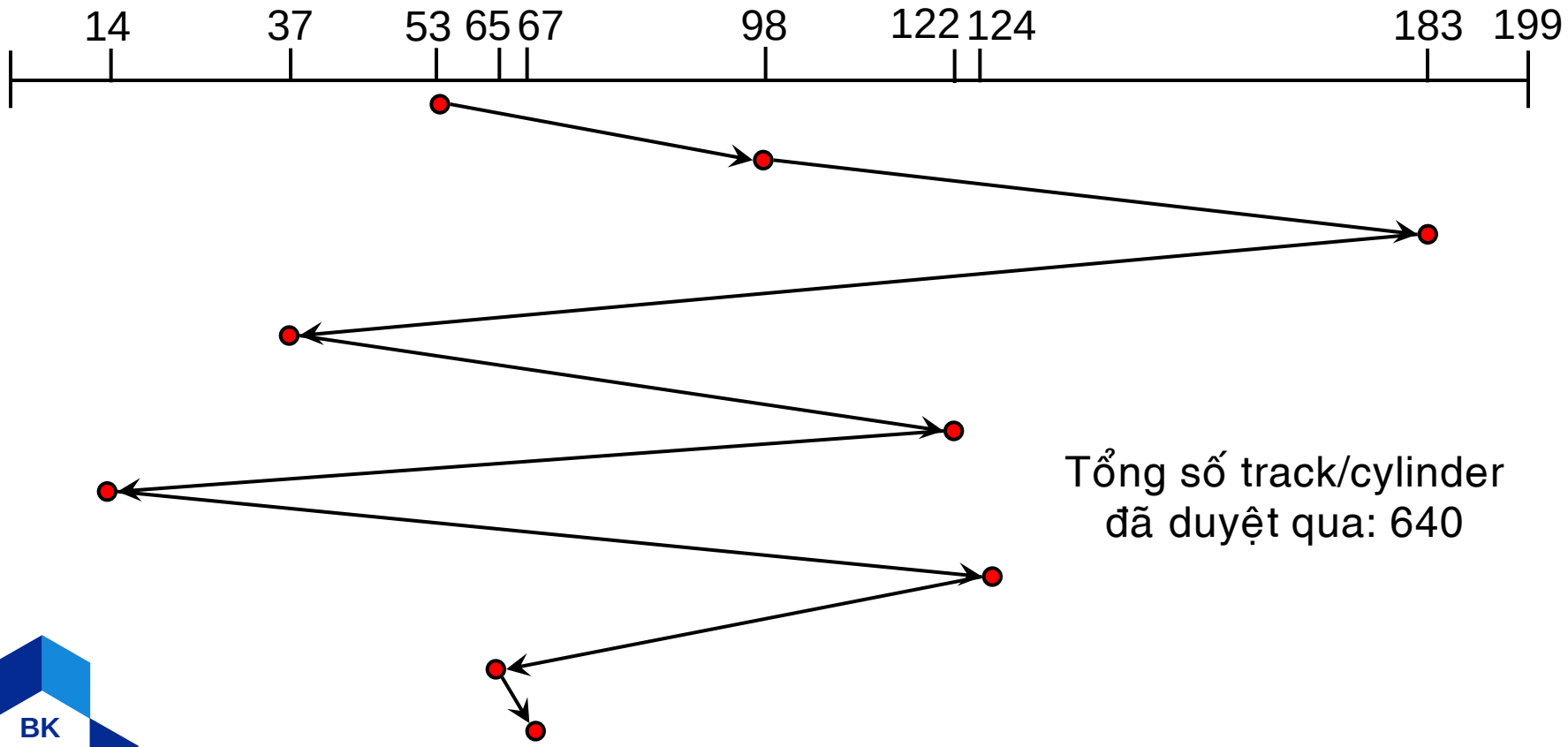
Định thời truy cập đĩa

- Ý tưởng chính
  - Sắp xếp lại trật tự của các yêu cầu đọc/ghi đĩa sao cho giảm thiểu thời gian di chuyển đầu đọc (seek time)
- Các giải thuật định thời truy cập đĩa
  - First Come, First Served (FCFS)
  - Shortest-Seek-Time First (SSTF)
  - SCAN
  - C-SCAN (Circular SCAN)
  - C-LOOK
- Ví dụ: định thời chuỗi yêu cầu đọc/ghi đĩa tại
  - cylinder 98, 183, 37, 122, 14, 124, 65, 67
  - Đầu đọc đang ở cylinder số 53

First Come First Served (FCFS)

Hàng đợi: 98, 183, 37, 122, 14, 124, 65, 67

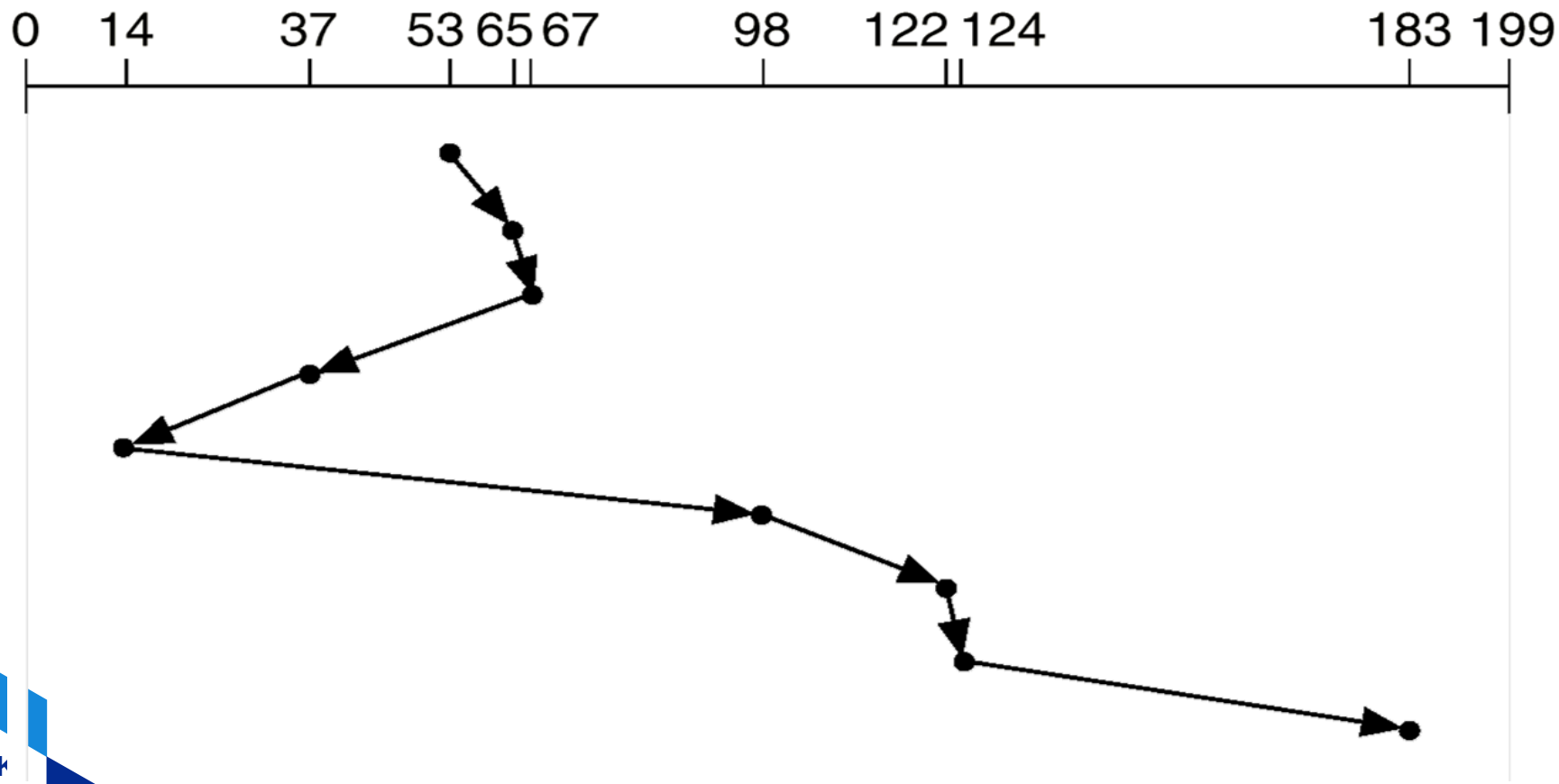
Đầu đọc đang ở cylinder số 53



Tổng số track/cylinder đã duyệt qua: 640

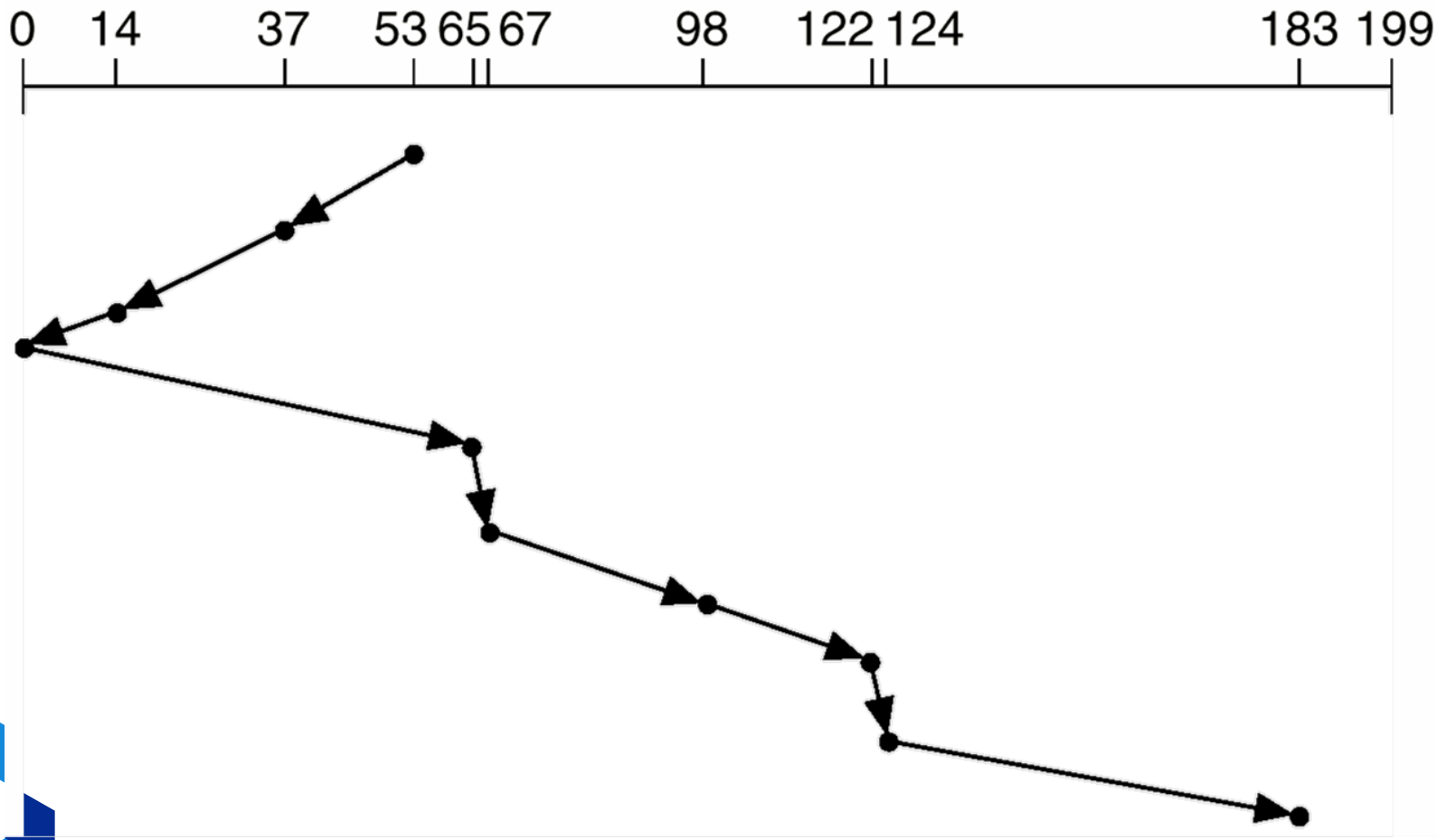
Shortest-Seek-Time First (SSTF)

queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53



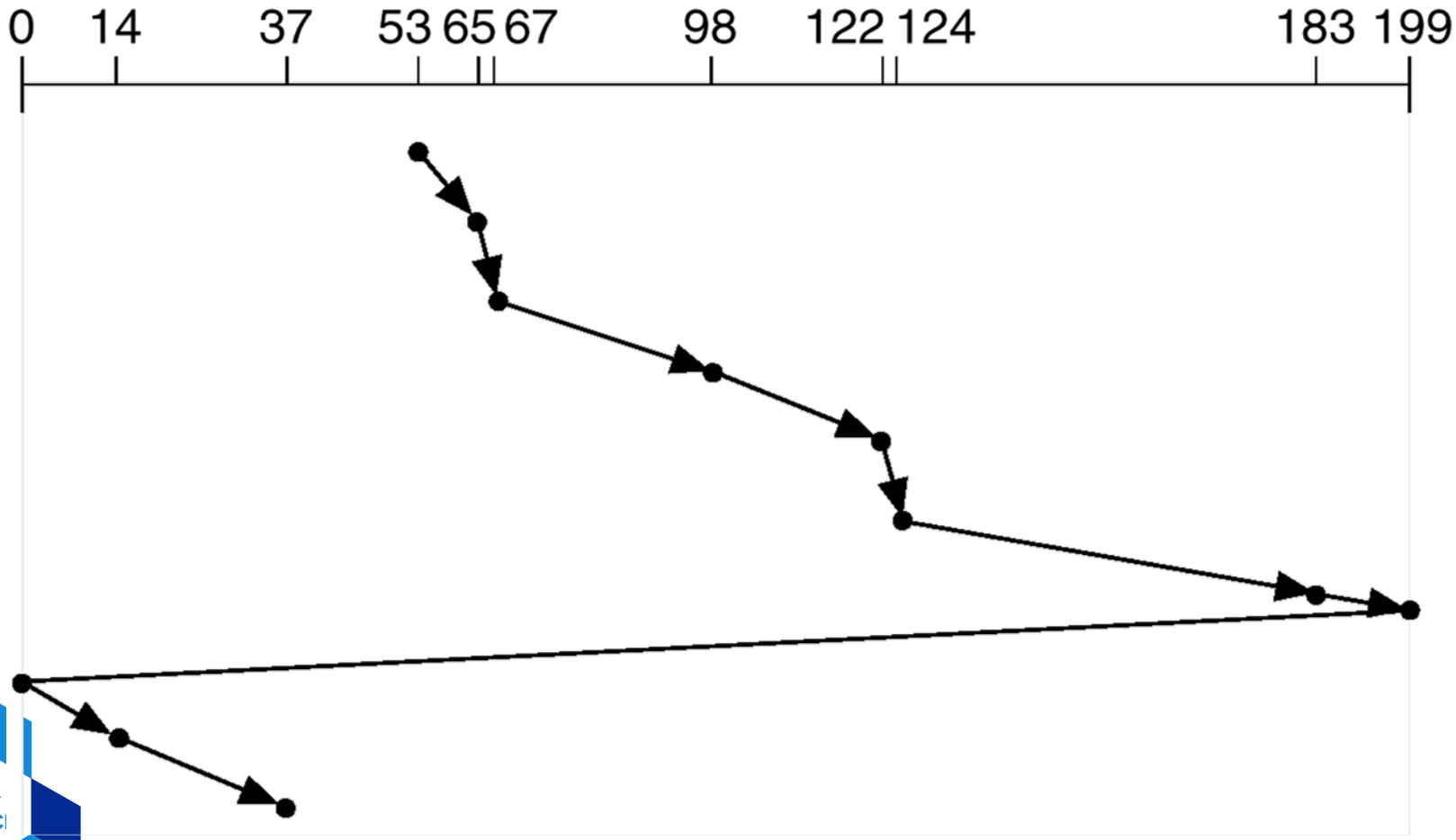
SCAN (elevator algorithm)

queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53



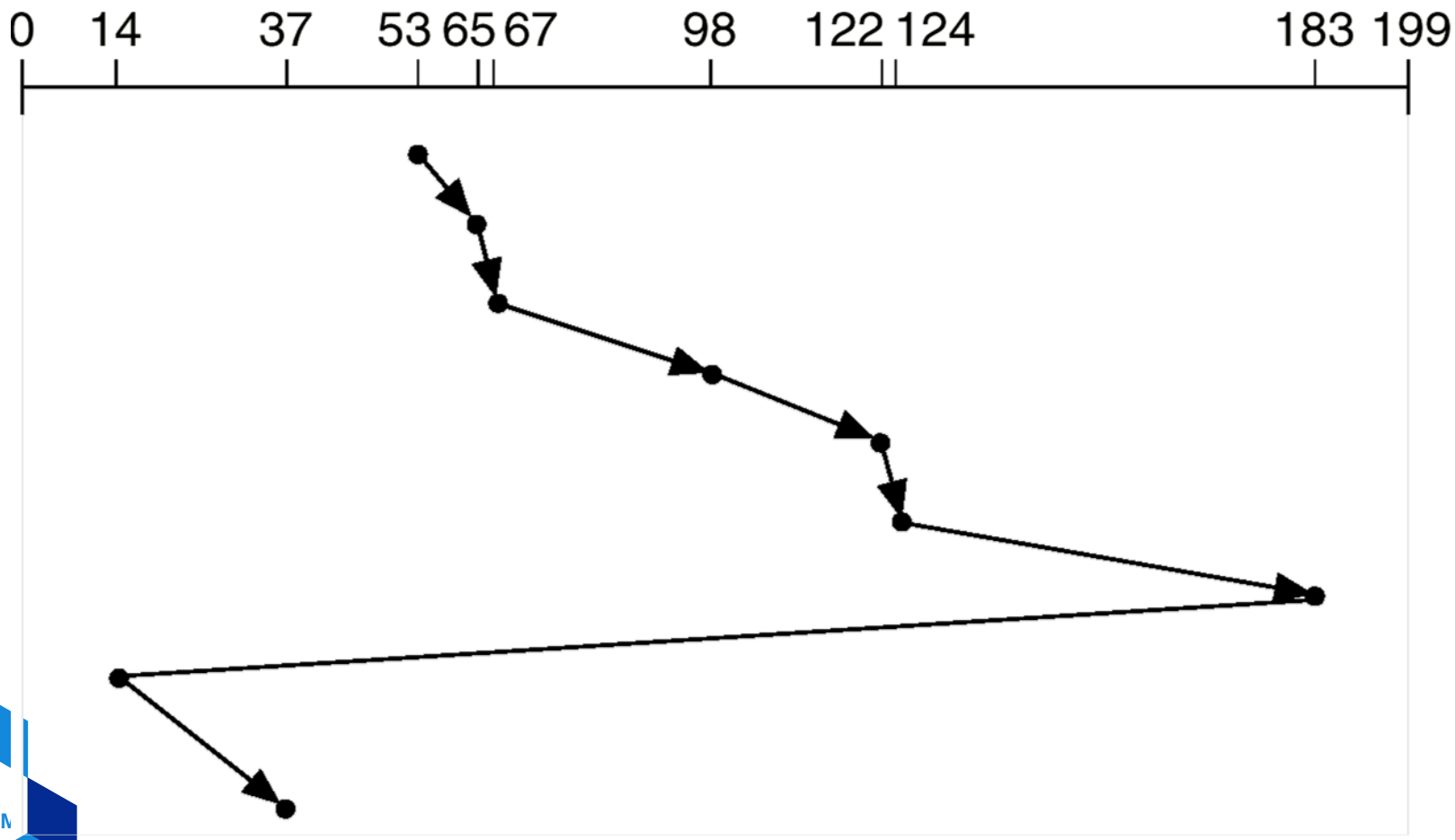
C-SCAN (Circular SCAN)

queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53



C-LOOK

queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53



Quản lý đĩa: Phân vùng (partitioning)

- Phân vùng: chia đĩa thành nhiều vùng (partition), mỗi vùng gồm nhiều block liên tục.
  - Mỗi partition được xem như một "đĩa luận lý" riêng biệt.
- Định dạng luận lý cho partition: tạo một hệ thống file (FAT, ext2,…)
  - Lưu các cấu trúc dữ liệu khởi đầu của hệ thống file lên partition
  - Tạo cấu trúc dữ liệu quản lý không gian trống và không gian đã cấp phát (DOS: FAT, UNIX: inode table)

Quản lý đĩa: Raw disk

- Raw disk: partition không có hệ thống file
- I/O lên raw disk được gọi là raw I/O
  - đọc hay ghi trực tiếp các block
  - không dùng các dịch vụ của file system như buffer cache, file locking, prefetching, cấp phát không gian trống, định danh file, và thư mục
- Ví dụ
  - Một số hệ thống cơ sở dữ liệu chọn dùng raw disk

Quản lý không gian tráo đổi (swap space)

- Swap space
  - không gian đĩa được sử dụng để mở rộng không gian nhớ trong kỹ thuật bộ nhớ ảo
  - Mục tiêu quản lý: cung cấp hiệu suất cao nhất cho hệ thống quản lý bộ nhớ ảo
  - Hiện thực
    - chiếm partition riêng, vd swap partition của Linux
    - hoặc qua một file system, vd file pagefile.sys của Windows
    - Thường kèm theo caching hoặc dùng phương pháp cấp phát liên tục

Quản lý các khối bị lỗi

- Tồn tại một số khối (sectors) bị lỗi:
  - Ngay sau khi xuất xưởng: tự sửa bằng cách thay thế với các sectors, tracks dự trữ.
  - Phát hiện sau một thời gian sử dụng trong hệ thống (OS):
    - Ví dụ:
      - Block 87 (logic block) không truy xuất được
      - Điều khiển đĩa phát hiện EEC không đúng, báo Os
      - Os ghi nhận để lần sau khi reboot thông báo điều khiển đĩa thay thế
      - Sau đó vị trí block 87 đã được cập nhật lại

RAID *(Redudant Arrays of Independent Disk)*

- Khi mật độ yêu cầu truy cập đĩa cao: nghẽn, hoặc "cổ chai" → hạn chế hiệu năng và tính ổn định của hệ thống
- Giải pháp: kết hợp nhiều đĩa (array) truy xuất song hành:
  - Hiệu năng cải thiện: chia mảnh dữ liệu và chứa trên nhiều đĩa (data striping)
  - Reliability is improved through redundancy
- Tăng độ tin cậy: lưu trữ dư thừa thông tin (Redundant Arrays of Independent Disks, or RAID)
- Có nhiều phương pháp để đáp ứng theo tiêu chí lưu dữ thông tin (schemes or levels)