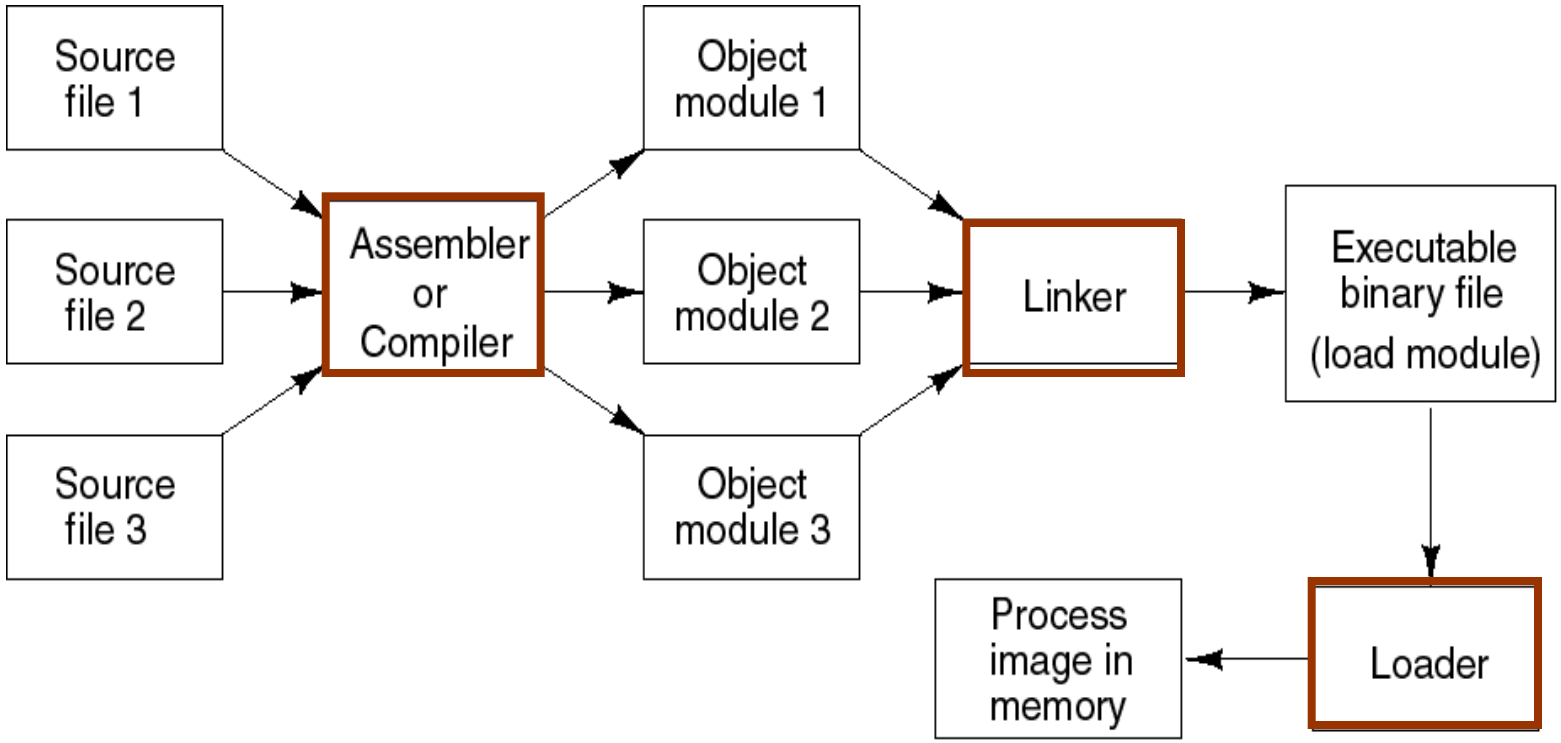
(A) Quản lý quá trình

- Khái niệm cơ bản
- Định thời CPU
- Các tác vụ cơ bản
- Sự cộng tác giữa các quá trình
- Giao tiếp giữa các quá trình

Khái niệm cơ bản

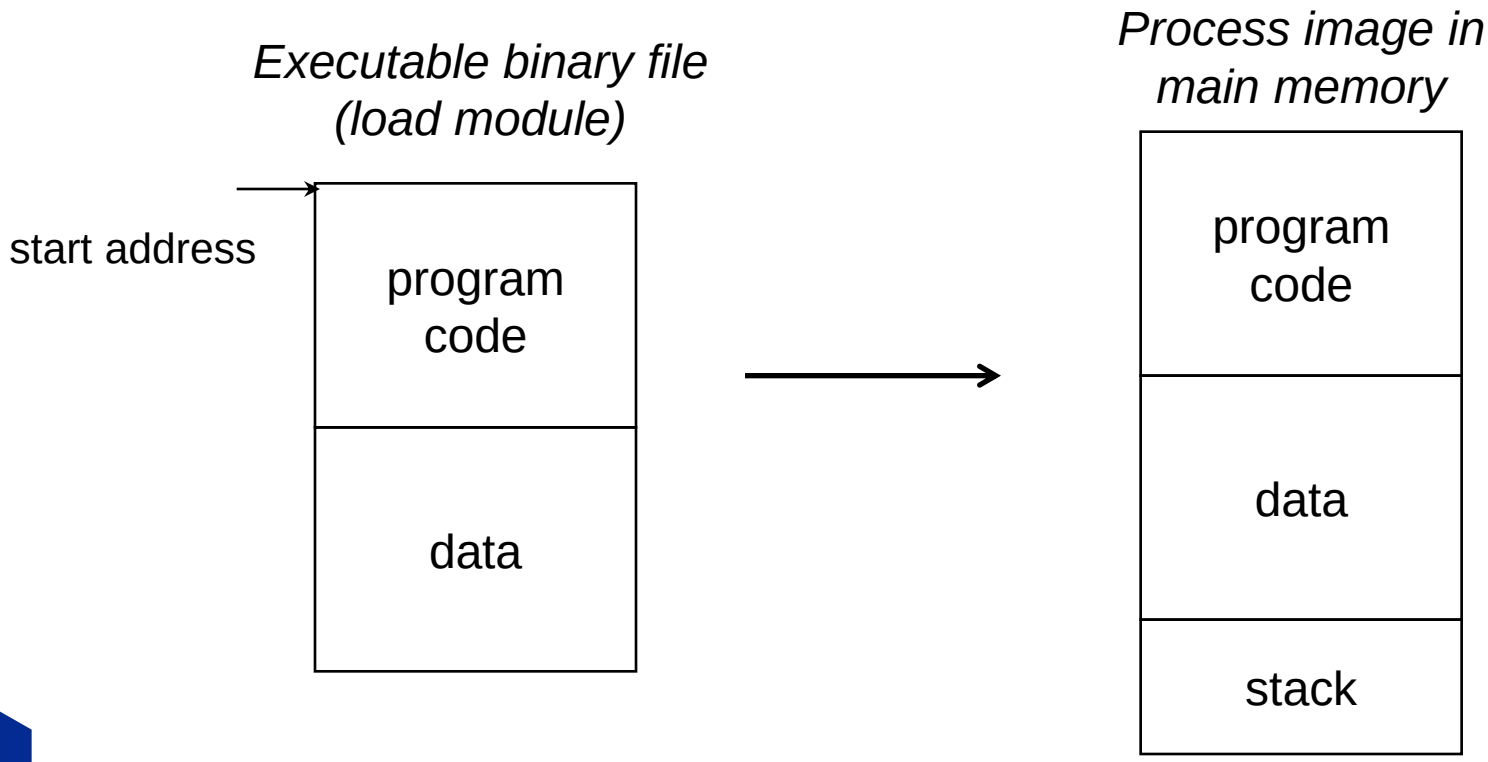
- Hệ thống máy tính thực thi nhiều chương trình khác nhau
  - Batch system: jobs
  - Time-shared systems: user programs, tasks
  - Job ≈ process
- *Quá trình* (process)
  - một chương trình *đang thực thi*, bao gồm:
    - *Text section* (program code), *data section* (chứa global variables)
    - Hoạt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

Thực hiện chương trình



Chương trình → quá trình

- Chương trình thực thi có định dạng *load module* mà trình nạp (loader) "hiểu" được
  - Ví dụ: định dạng elf trong Linux
- Layout luận lý của *process image*



Khởi tạo quá trình

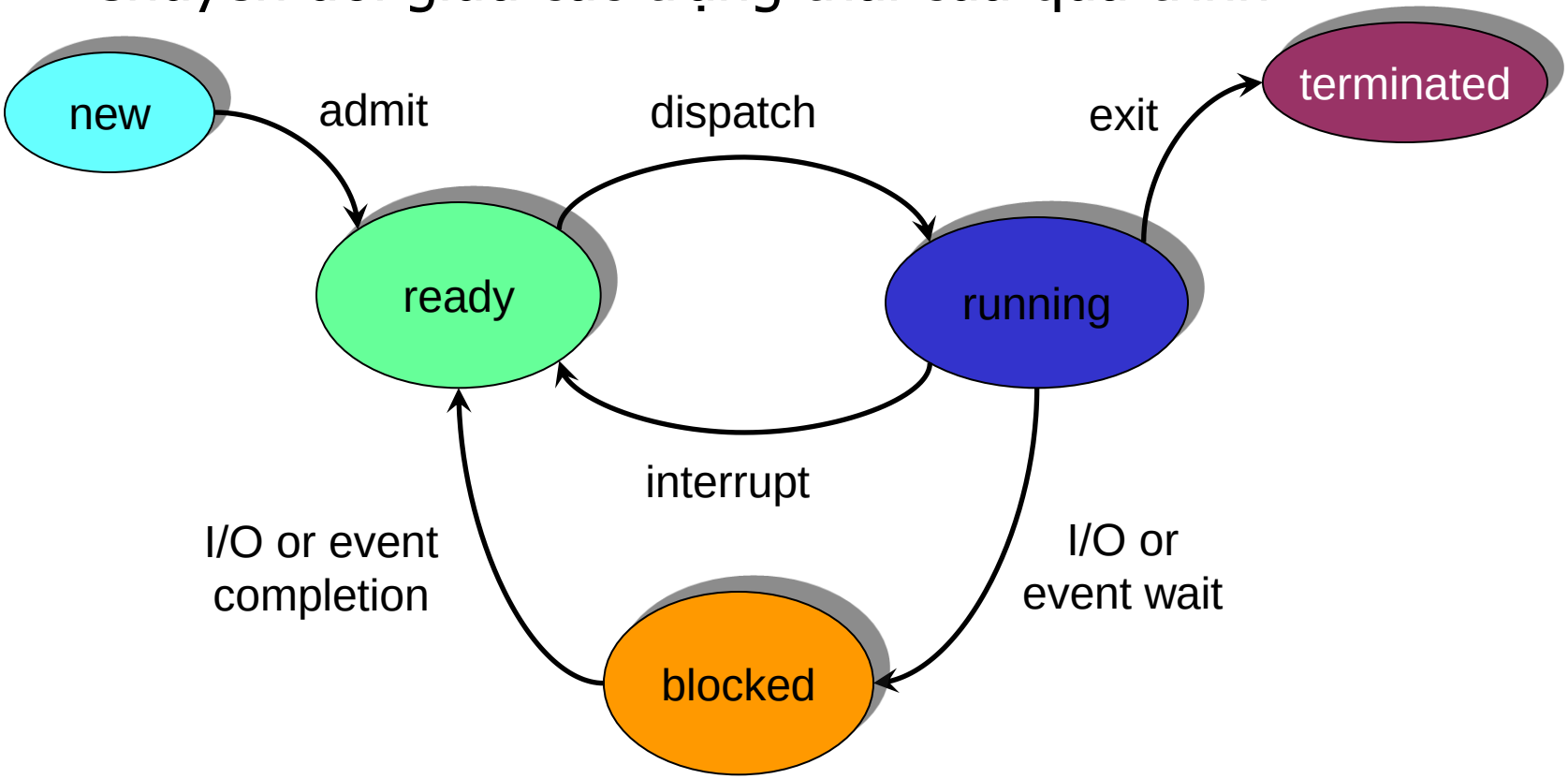
- Các bước hệ điều hành khởi tạo 1 quá trình
  - Cấp phát *định danh* duy nhất (process number hay process identifier, pid) cho quá trình
  - Cấp phát không gian nhớ để nạp quá trình
  - Khởi tạo khối dữ liệu *Process Control Block* (PCB) cho quá trình
    - PCB là nơi hệ điều hành lưu các thông tin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

Trạng thái của quá trình

- Các *trạng thái của quá trình* (process state):
  - *new:* quá trình vừa được tạo
  - *ready:* quá trình đã có đủ tài nguyên, chỉ còn cần CPU
  - *running:* các lệnh của quá trình đang được thực thi
  - *waiting:* hay là *blocked*, quá trình đợi I/O hoàn tất, hay đợi tín hiệu.
  - *terminated:* quá trình đã kết thúc.

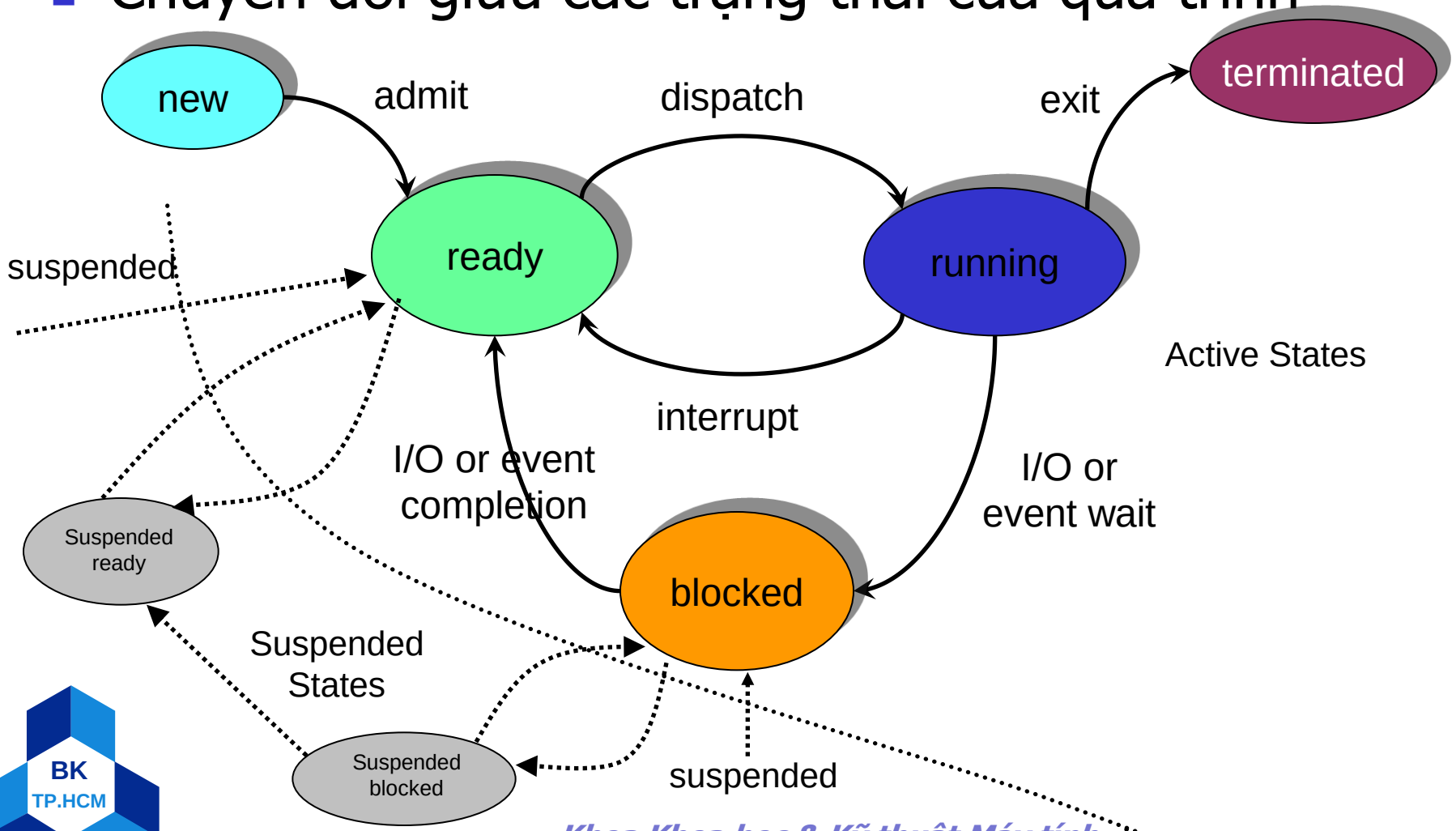
Trạng thái quá trình (tt.)

- Chuyển đổi giữa các trạng thái của quá trình



Trạng thái quá trình (tt.)

- Chuyển đổi giữa các trạng thái của quá trình



Ví dụ về trạng thái quá trình

```
/* test.c */
int main(int argc, char** argv)
{
   printf("Hello world\n");
   exit(0);
}
```

Biên dịch chương trình trong Linux

gcc test.c –o test

Thực thi chương trình test

./test

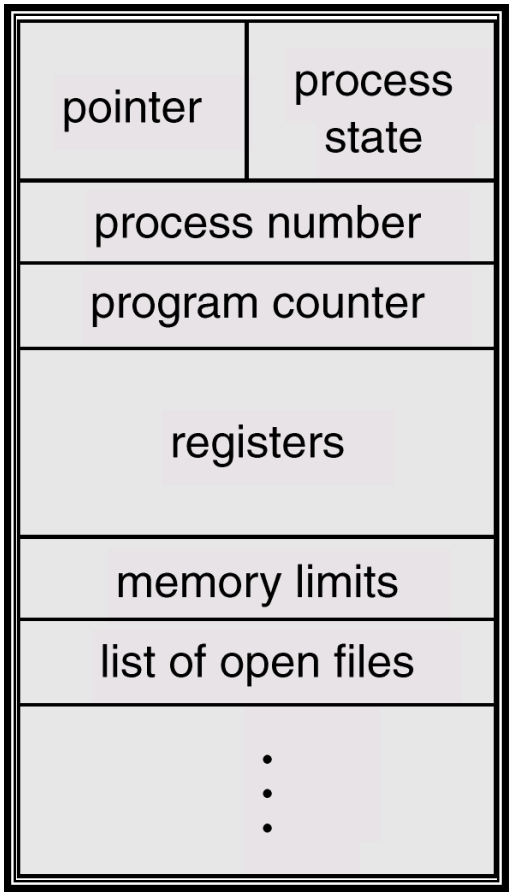
Trong hệ thống sẽ có một quá trình *test* được tạo ra, thực thi và kết thúc.

- Chuỗi trạng thái của quá trình test như sau (trường hợp tốt nhất):
  - new
  - ready
  - running
  - waiting (do chờ I/O khi gọi printf)
  - ready
  - running
  - terminated

Process Control Block (PCB)

- Đã thấy là mỗi quá trình trong hệ thống đều được cấp phát một *Process Control Block* (PCB)
- PCB là một trong các cấu trúc dữ liệu quan trọng nhất của hệ điều hành

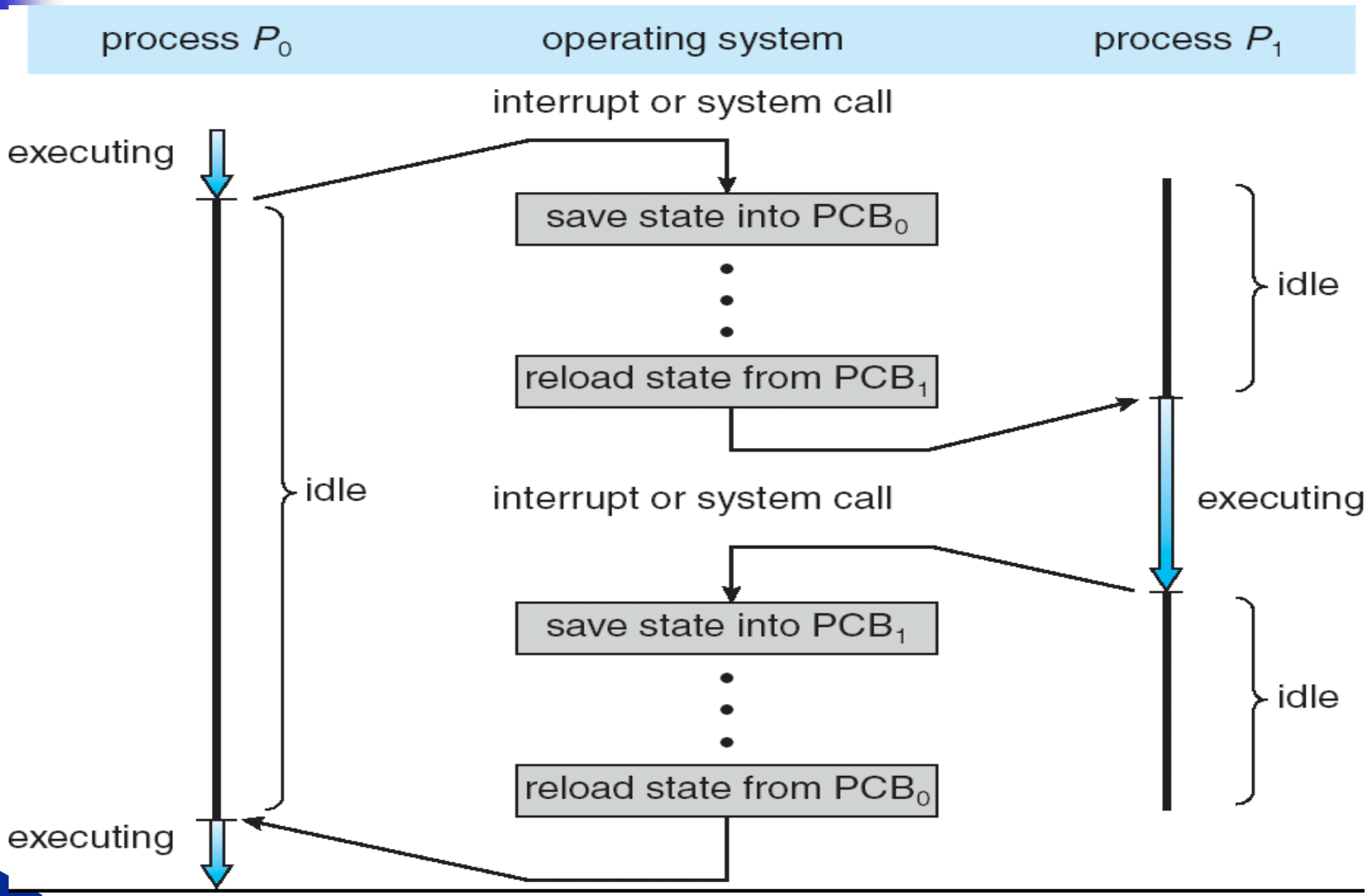
Ví dụ layout của một PCB: (trường pointer dùng để liên kết các PCBs thành một linked list)



Chuyển ngữ cảnh(context switch)

- *Ngữ cảnh* (context) của một quá trình là trạng thái của quá trình
- Ngữ cảnh của quá trình được biểu diễn trong PCB của nó
- *Chuyển ngữ cảnh* (context switch) là công việc giao CPU cho quá trình khác. Khi đó cần:
- lưu ngữ cảnh của quá trình cũ vào PCB của nó
- nạp ngữ cảnh từ PCB của quá trình mới để quá trình mới thực thi

Chuyển ngữ cảnh(tt.)



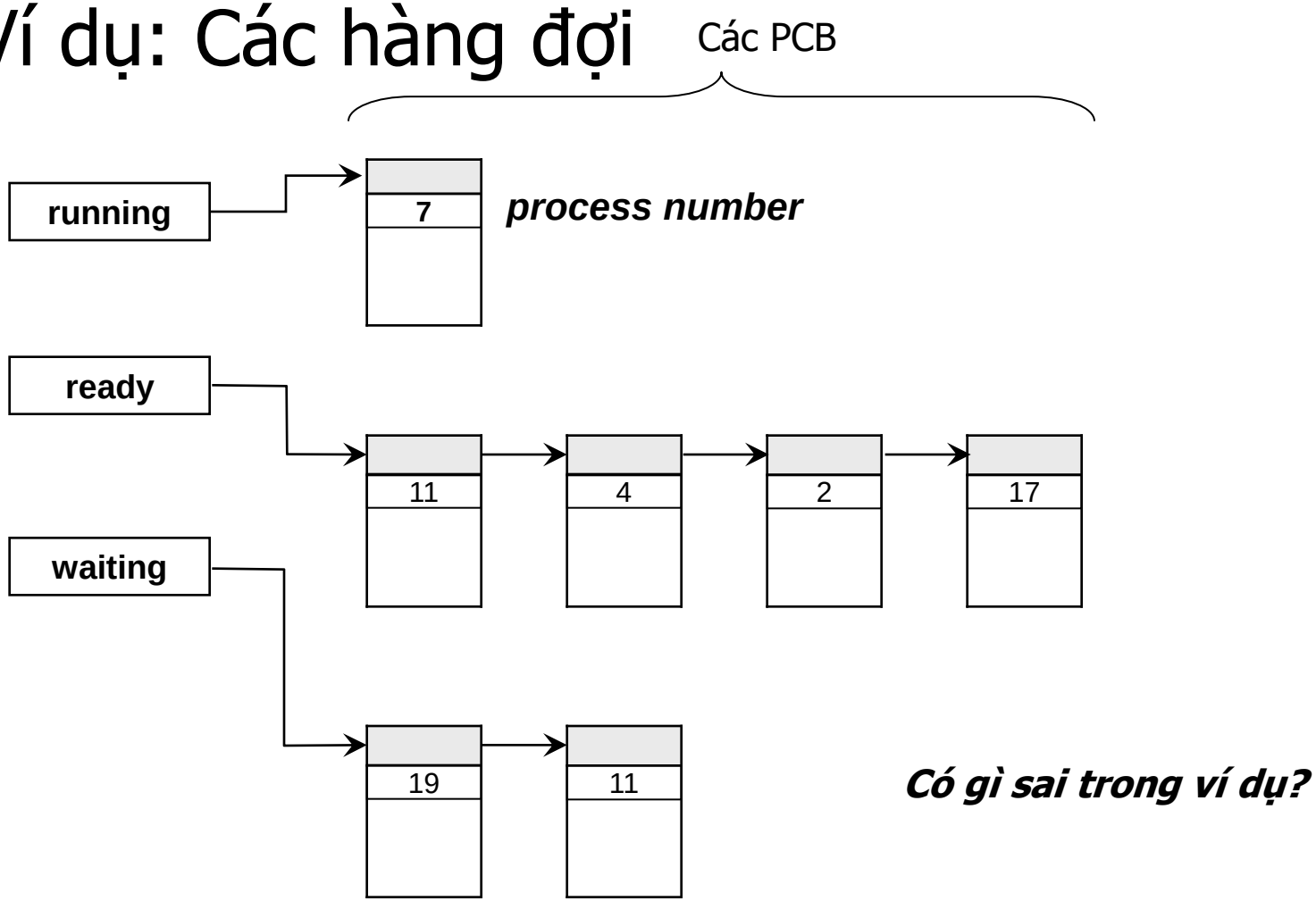
Yêu cầu đối với hệ điều hành

Để quản lý các quá trình, OS cần:

- Hỗ trợ thực thi luân phiên giữa nhiều quá trình
  - Hiệu suất sử dụng CPU
  - Thời gian đáp ứng
- Phân phối tài nguyên hệ thống hợp lý
  - tránh deadlock, trì hoãn vô hạn định,...
- Cung cấp cơ chế giao tiếp và đồng bộ hoạt động các quá trình
- Cung cấp cơ chế hỗ trợ user tạo/kết thúc quá trình

Quản lý quá trình

- Ví dụ: Các hàng đợi



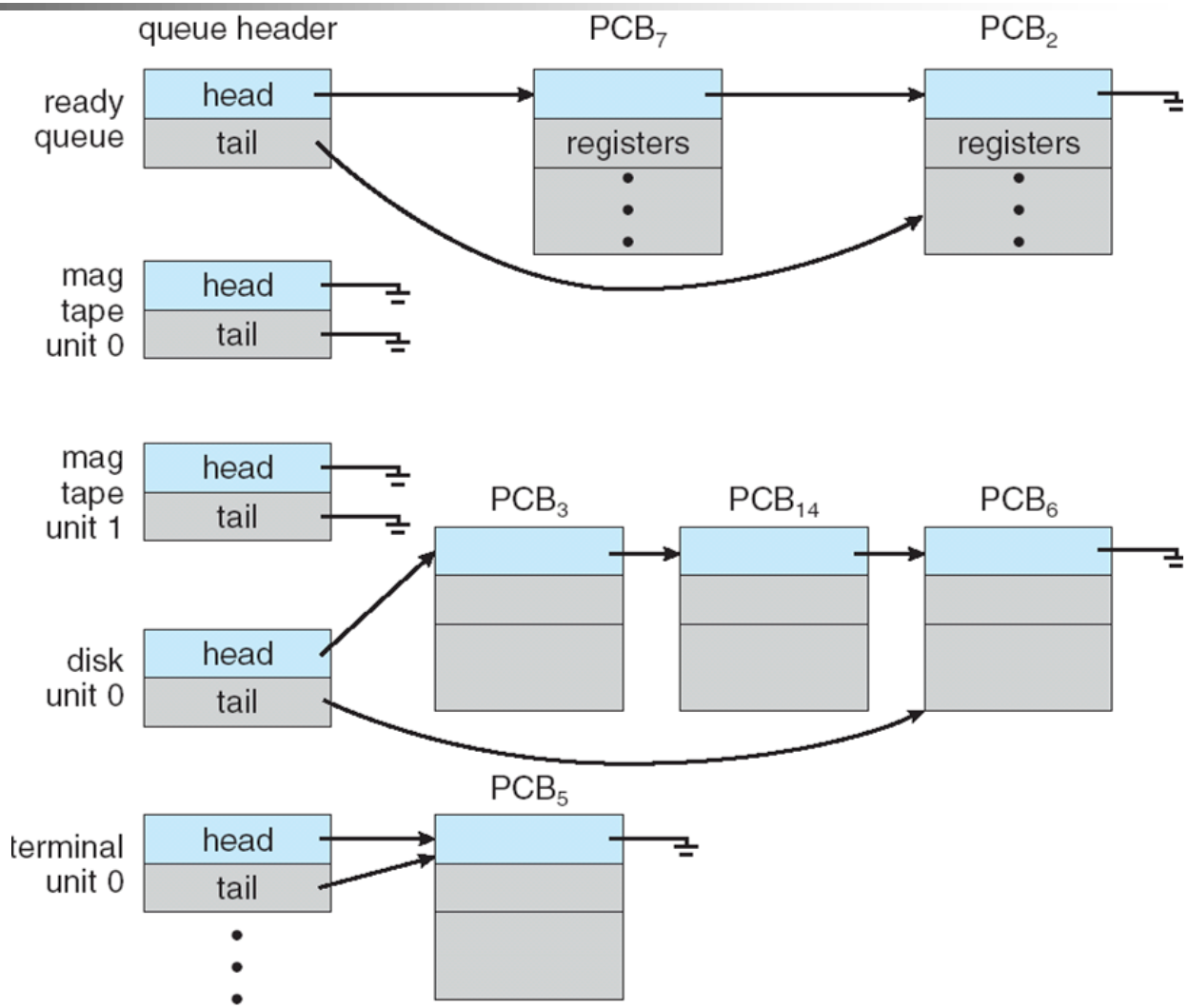
Có gì sai trong ví dụ?

Định thời quá trình

- Tại sao phải định thời?
  - Multiprogramming
    - Có nhiều quá trình phải thực thi luân phiên nhau
    - Mục tiêu: cực đại hiệu suất sử dụng của CPU
  - Time-sharing
    - Cho phép users tương tác với quá trình đang thực thi
    - Mục tiêu: tối thiểu thời gian đáp ứng
  - Các *bộ định thời* (scheduler)
  - Các *hàng đợi định thời* (scheduling queue)

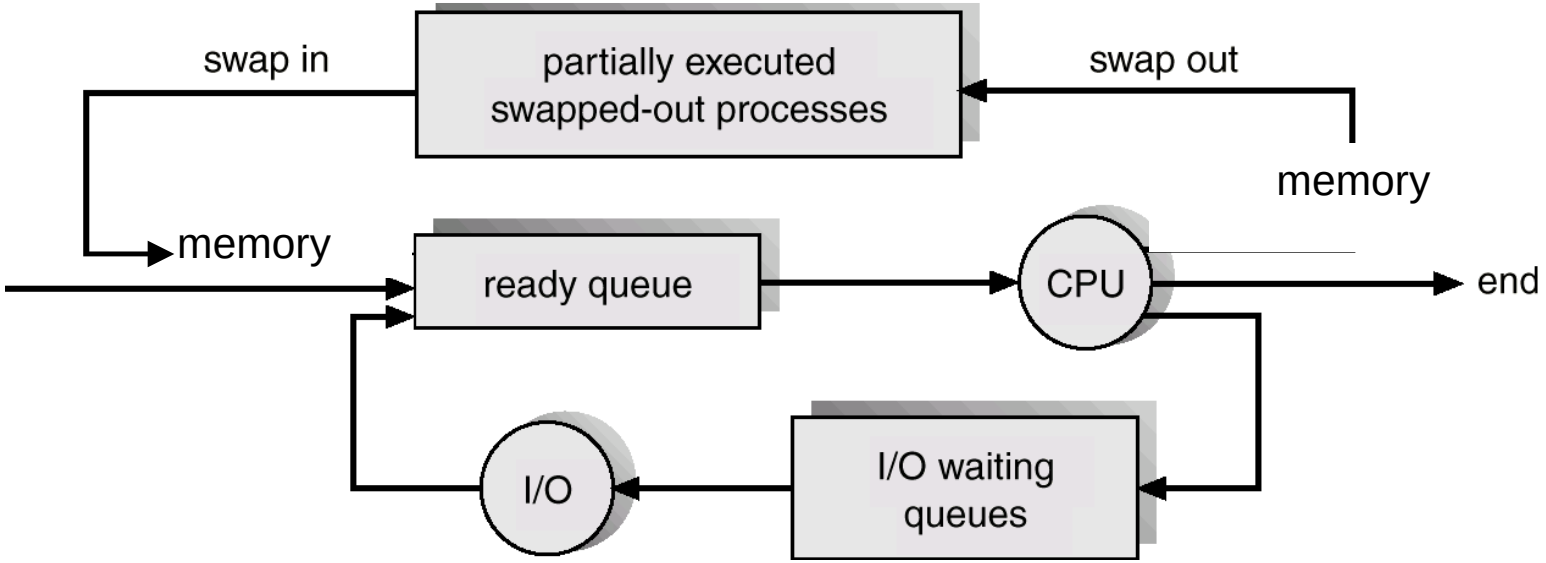
Các hàng đợi định thời

- Job queue
- Ready queue
- Device queues
- ...



Medium-term scheduling

- Đôi khi hệ điều hành (như time-sharing system) có thêm medium-term scheduling để điều chỉnh mức độ multiprogramming của hệ thống
- *Medium-term scheduler*
  - chuyển quá trình từ bộ nhớ sang đĩa (swap out)
  - chuyển quá trình từ đĩa vào bộ nhớ (swap in)



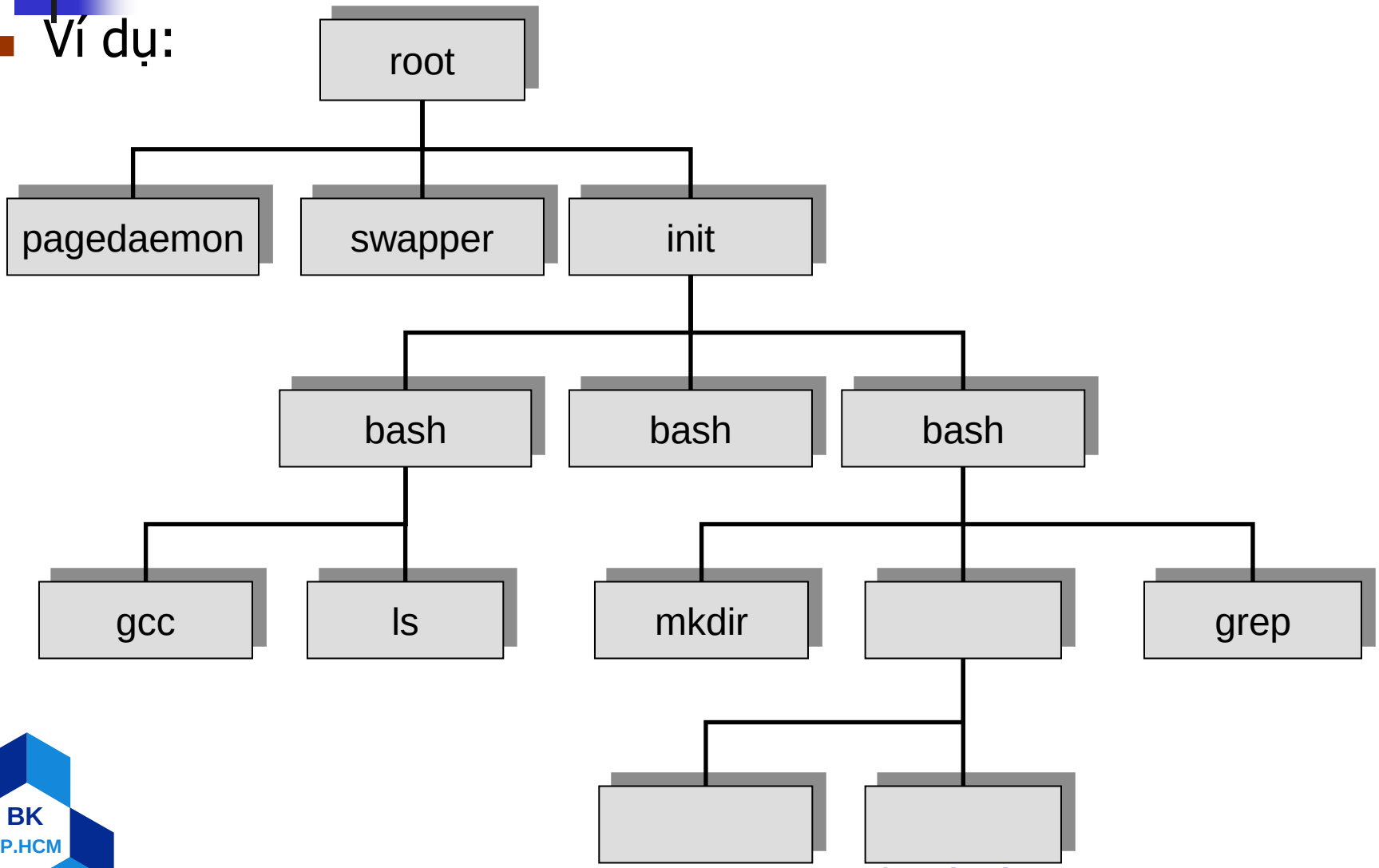
Các tác vụ đối với quá trình

(1) Tạo quá trình mới (process creation)

- Một quá trình có thể tạo quá trình mới thông qua một system call (vd: hàm fork trong Unix)
  - Ví dụ: (Unix) Khi user đăng nhập hệ thống, một command interpreter (shell) sẽ được tạo ra cho user
- Quá trình được tạo là quá trình *con* của quá trình tạo (quá trình *cha*). Quan hệ cha-con định nghĩa một *cây quá trình*.

Cây quá trình trong Linux/Unix

- Ví dụ:



Các tác vụ đối với quá trình (tt.)

Tạo quá trình mới (tt.)

- Chia sẻ tài nguyên của quá trình cha: hai khả năng
  - Quá trình cha và con chia sẻ mọi tài nguyên
  - Quá trình con chia sẻ một phần tài nguyên của cha
- Trình tự thực thi: hai khả năng
  - Quá trình cha và con thực thi đồng thời (concurrently)
  - Hệ điều hành chỉ cho quá trình cha chạy khi quá trình con kết thúc.

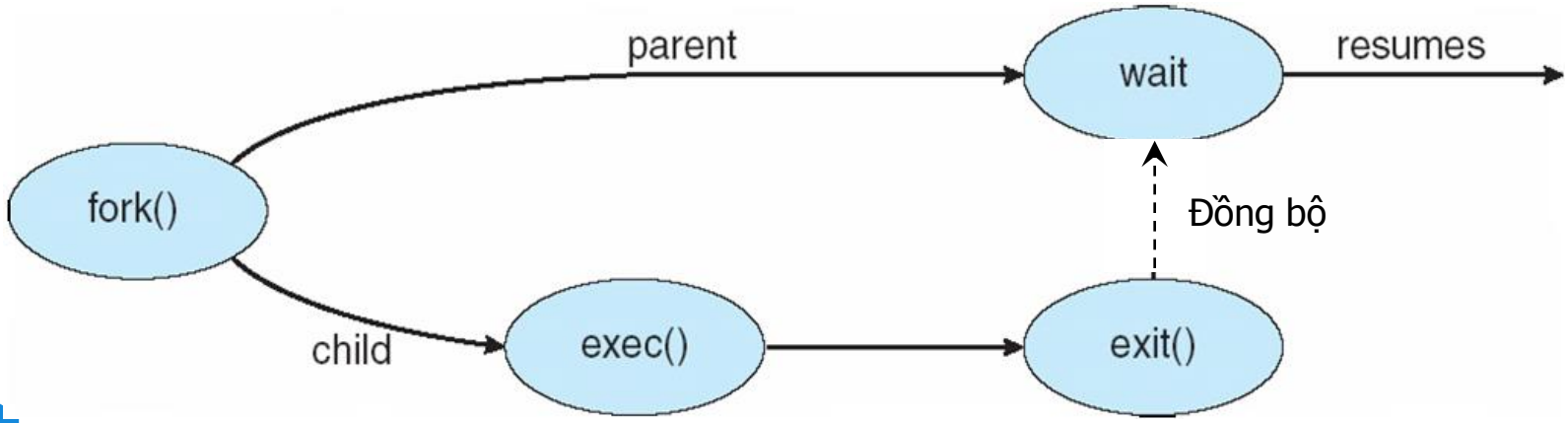
Các tác vụ đối với quá trình (tt.)

Tạo quá trình mới (tt)

- Không gian địa chỉ (address space): hai khả năng
  - Không gian địa chỉ của quá trình con được nhân bản từ cha (con có cùng code và data như cha)
  - Hệ điều hành sẽ cấp cho 1 chương trình mới (qua trình con).

Về quan hệ cha/con

- Ví dụ trong UNIX/Linux
  - Quá trình gọi fork() để tạo một quá trình mới
  - Quá trình mới (con) gọi exec() để nạp và thực thi một chương trình trong không gian nhớ của nó
  - Quá trình cha làm việc khác, ví dụ: gọi wait() để đợi con xong



Ví du: Tạo process với fork()

```
#include <stdio.h>
#include <unistd.h>
int main (int argc, char *argv[]){
   int  return_code;
   /* create a new process */
   return_code = fork();

   if (return_code > 0){
        printf("This is parent process");
        wait(NULL);
        exit(0);
   }
   else if (return_code == 0)
   {
        printf("This is child process");
        execlp("/bin/ls", "ls", NULL);
        exit(0);
   }
   else {
        printf("Fork error\n");
        exit(-1);
   }
}
```

Các tác vụ đối với quá trình (tt.)

(2) Kết thúc quá trình

- Quá trình tự kết thúc
  - Quá trình kết thúc khi thực thi lệnh cuối và gọi system routine exit
- Quá trình kết thúc do quá trình khác (có đủ quyền, vd: quá trình cha của nó)
  - Gọi system routine abort với tham số là pid (process identifier) của quá trình cần được kết thúc
- Hệ điều hành thu hồi tất cả các tài nguyên của quá trình kết thúc (vùng nhớ, I/O buffer,...)

Cộng tác giữa các quá trình

- Trong quá trình thực thi, các quá trình có thể *cộng tác* (cooperate) để hoàn thành công việc
- Module hóa
  - Xây dựng một phần mềm phức tạp bằng cách chia thành các module/process hợp tác nhau
- 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

Bài toán producer-consumer

- Bài toán tiêu biểu về sự cộng tác giữa các quá trình: *bài toán producer-consumer*
  - *Producer* tạo ra các dữ liệu và *consumer* tiêu thụ, sử dụng các dữ liệu đó. Sự trao đổi thông tin thực hiện qua buffer
    - *unbounded buffer:* kích thước buffer vô hạn (không thực tế).
    - *bounded buffer:* kích thước buffer có hạn.
  - Producer và consumer phải hoạt động đồng bộ vì
    - Consumer không được tiêu thụ khi producer chưa sản xuất
    - Producer không được tạo thêm sản phẩm khi buffer đầy.

Interprocess communication (IPC)

- *IPC* là cơ chế cung cấp bởi hệ điều hành nhằm giúp các quá trình giao tiếp với nhau
- Hai mô hình cho IPC
  - Truyền thông điệp (Message passing)
  - Dùng bộ nhớ chung (shared memory)

Message passing system

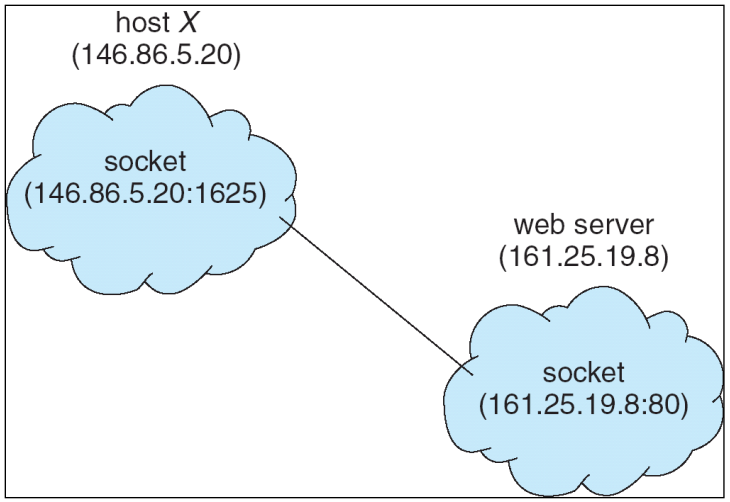
- Làm thế nào để các quá trình giao tiếp nhau? Các vấn đề:
  - *Naming*
    - Giao tiếp *trực tiếp*
      - send(P, msg): gửi thông điệp đến quá trình P
      - receive(Q, msg): nhận thông điệp đến từ quá trình Q
    - Giao tiếp *gián tiếp*: thông qua *mailbox* hay *port*
      - send(A, msg): gửi thông điệp đến mailbox A
      - receive(B, msg): nhận thông điệp từ mailbox B
  - *Synchronization*: blocking send, nonblocking send, blocking receive, nonblocking receive
  - *Buffering*: dùng queue để tạm chứa các message
    - Zero capacity (no buffering)
    - Bounded capacity: độ dài của queue là giới hạn
    - Unbounded capacity: độ dài của queue là không giới hạn

Giao tiếp trong hệ thống client-server

- Sockets
- Remote Procedure Calls (RPC)
- Remote Method Invocation (RMI)

Socket

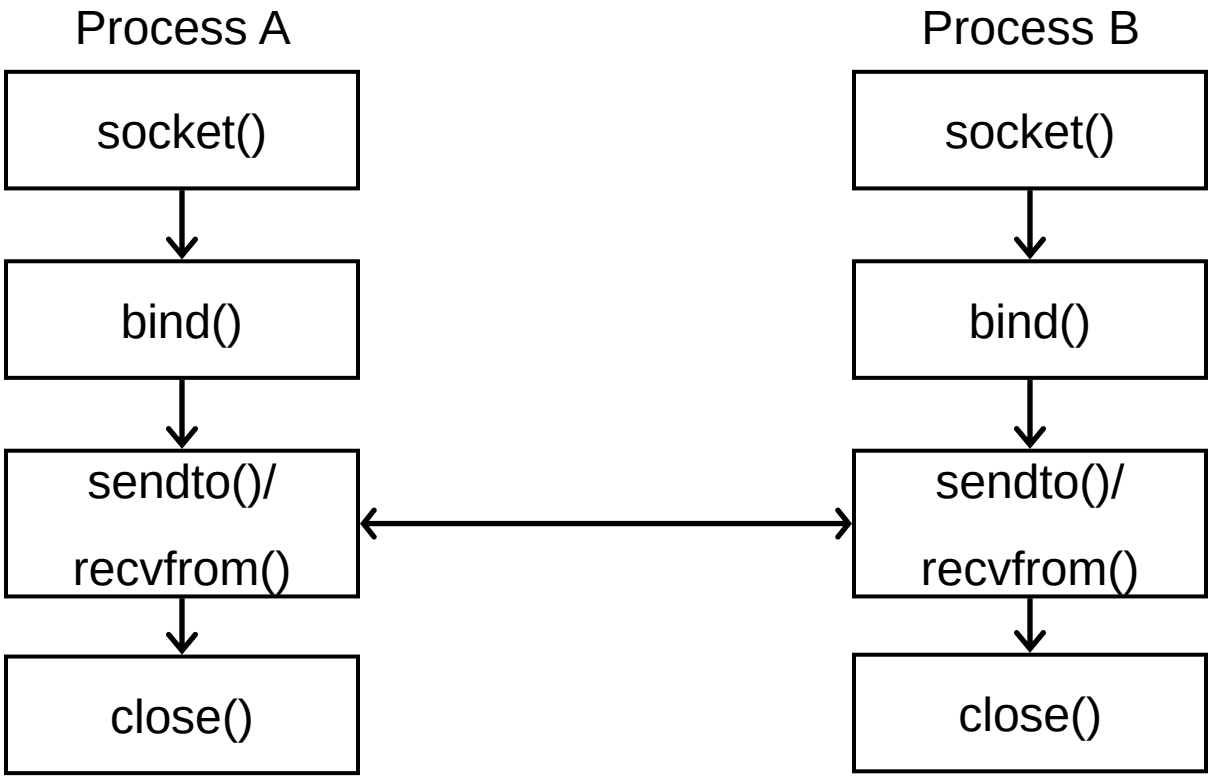
- *Socket*
  - Đầu cuối (endpoint) của một kênh giao tiếp
  - Cơ chế giao tiếp mức thấp (low-level), gửi nhận một chuỗi byte dữ liệu không cấu trúc (unstructured stream of bytes)
  - Giao tiếp qua socket: connectionless và connection-oriented
  - Lập trình socket
    - Berkeley socket (BSD socket), WinSock



Cơ chế gửi/nhận qua socket

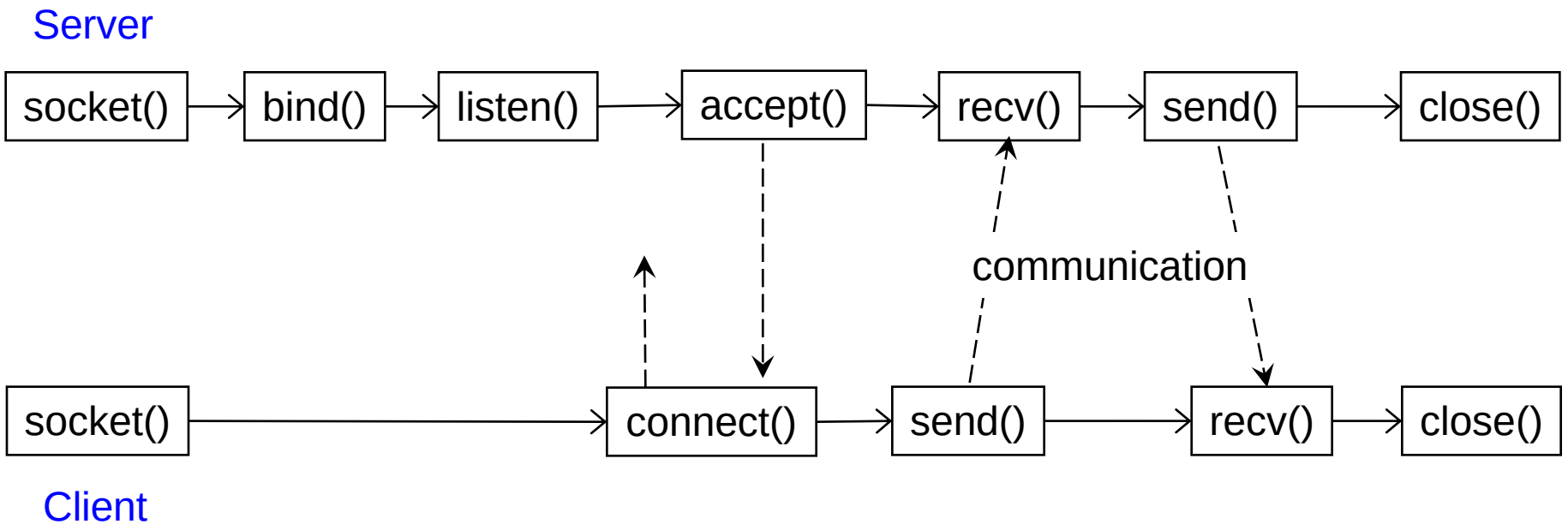
Hàm thư viện	*Diễn giải*
socket()	Tạo một socket
bind()	Gắn địa chỉ cục bộ vào một socket
listen()	Thiết lập độ dài queue
accept()	(server) Chờ kết nối đến từ client
connect()	(client) kết nối đến một server
send() sendto()	Gửi dữ liệu qua kênh giao tiếp đã thiết lập Gửi dữ liệu đến một địa chỉ
recv() recvfrom()	Nhận dữ liệu qua kênh giao tiếp đã thiết lập Nhận dữ liệu đến từ một địa chỉ
close()	Đóng kết nối

Connectionless Transport Service



- sendto(socket, buffer, buffer_length, flags, destination_address, addr_len)
- recvfrom(socket, buffer, buffer_length, flags, from_address, addr_len)

Connection-oriented Transport Service

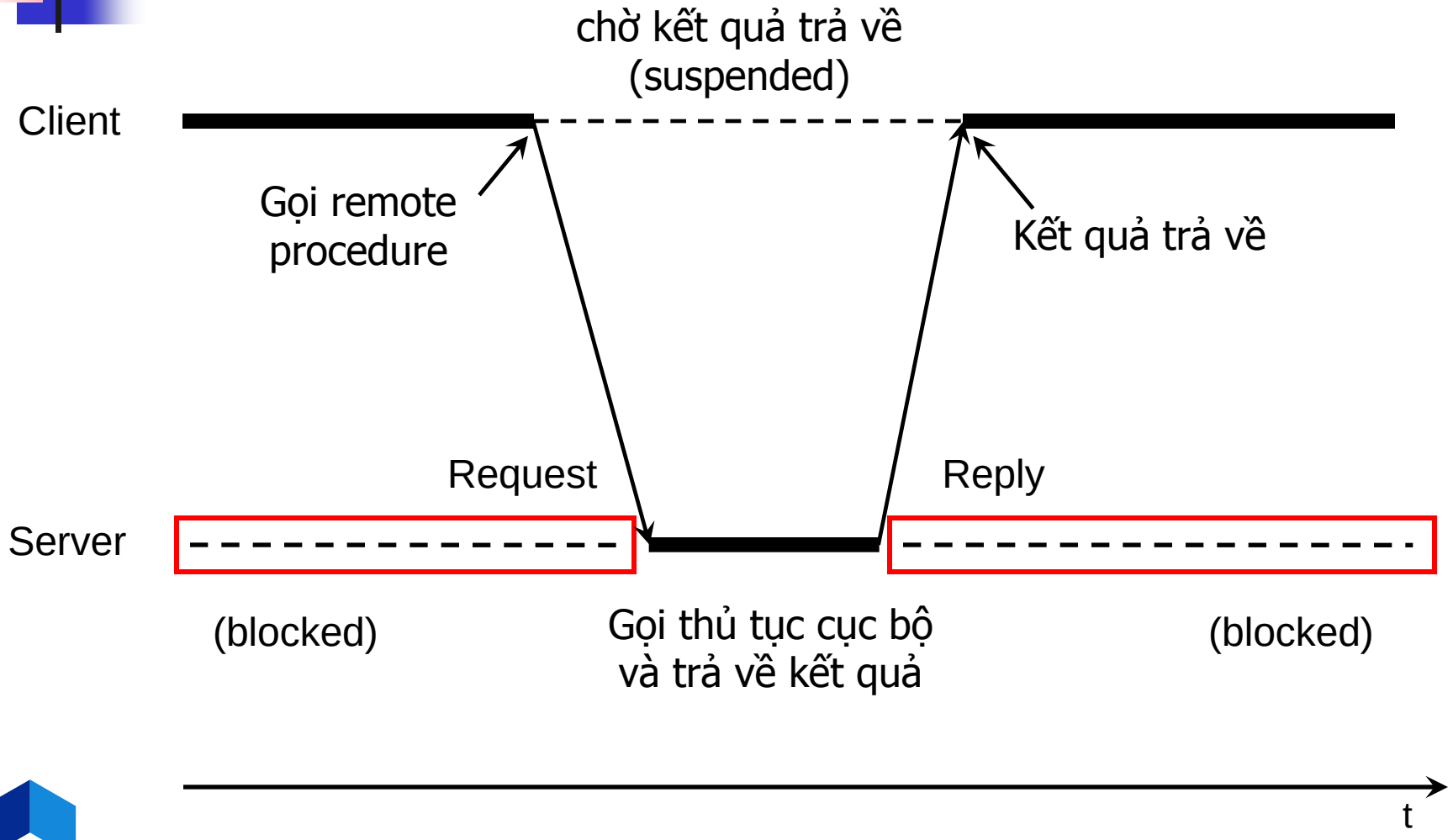


- send(socket, buffer, buffer_length, flags)
- *recv(socket, buffer, buffer_length, flags)*

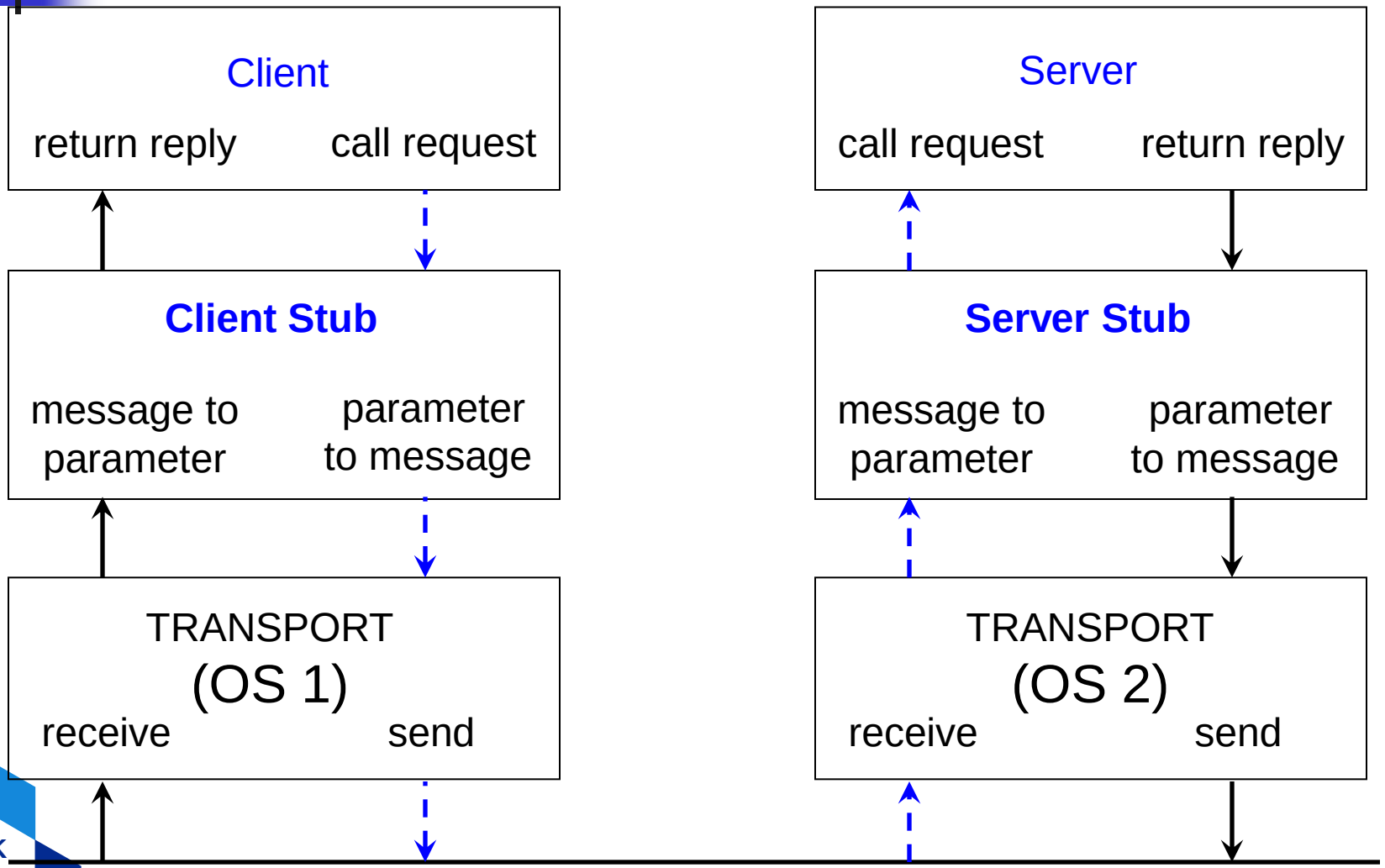
Remote procedure call

- *Remote procedure call* (RPC)
  - Cho phép một chương trình gọi một thủ tục nằm trên máy tính ở xa qua mạng.
- Các vấn đề khi hiện thực RPC
  - Truyền tham số và kết quả trả về của lời gọi thủ tục
  - Chuyển đổi dữ liệu khi truyền trên mạng (data conversion)
  - Kết nối client đến server
  - Bin dịch chương trình
  - Kiểm sóat lỗi
  - Security

Sơ đồ hoạt động của RPC



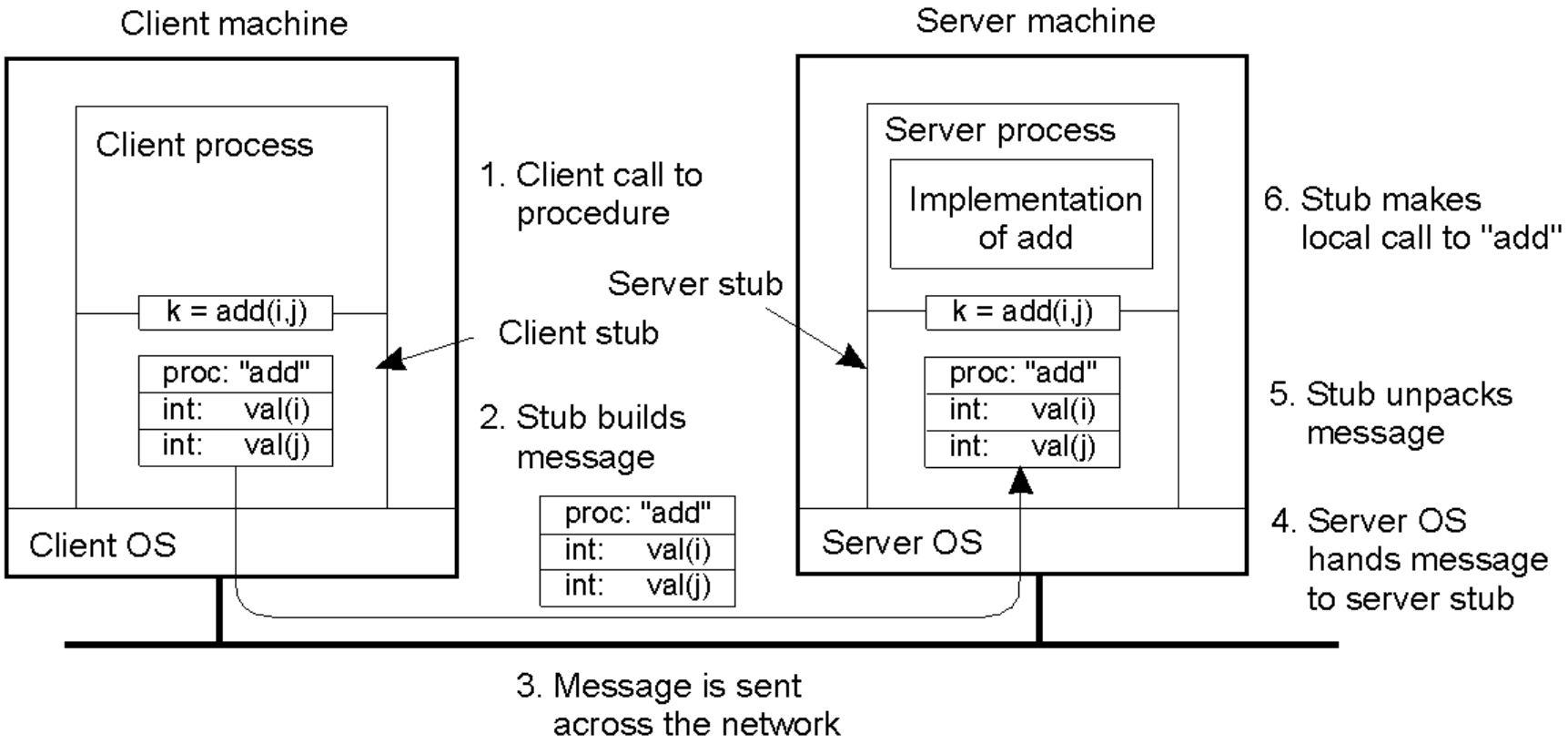
Lưu đồ thực hiện RPC



Truyền tham số trong RPC

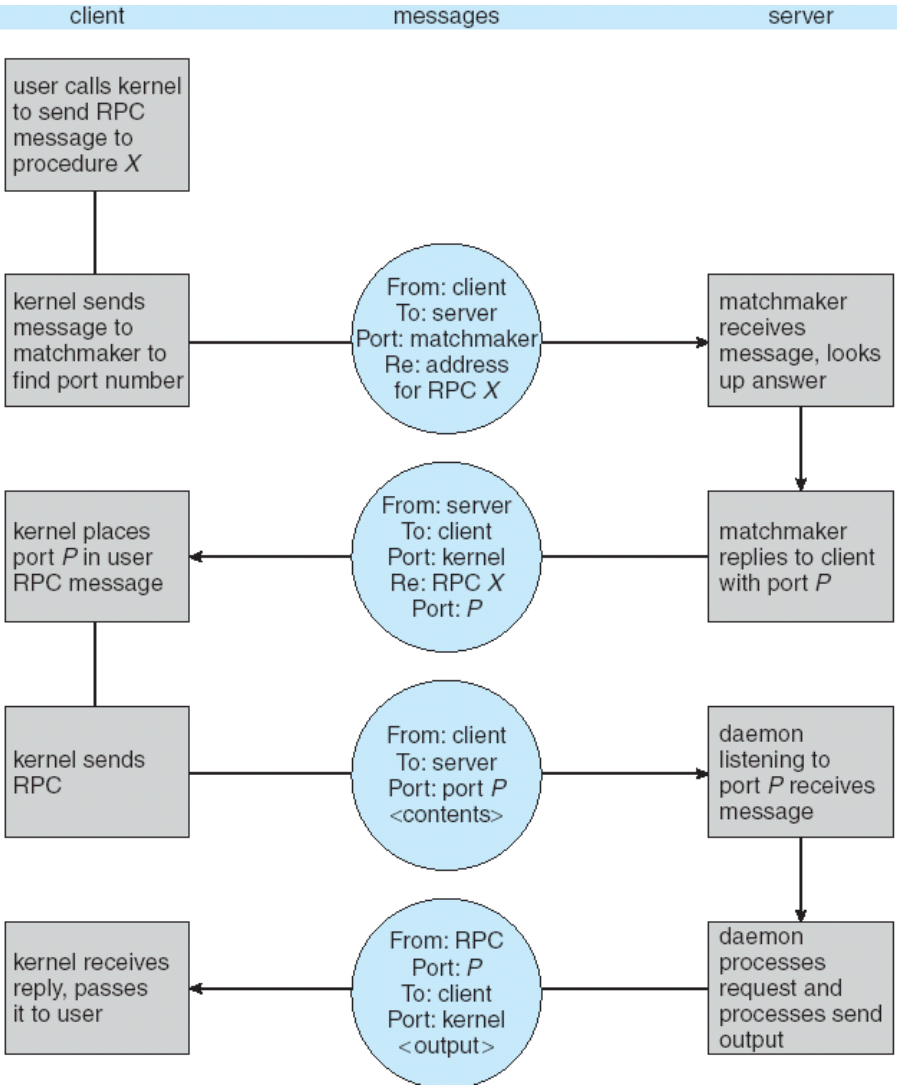
- *Marshalling*
  - qui tắc truyền tham số và chuyển đổi dữ liệu trong RPC bao gồm cả đóng gói dữ liệu thành dạng thức có thể truyền qua mạng máy tính..
- Biểu diễn dữ liệu và kiểm tra kiểu dữ liệu
  - Dữ liệu được biểu diễn khác nhau trên các hệ thống khác nhau
    - ASCII, EBCDIC
    - Ví dụ biểu diễn 32-bit integer trong máy:
      - *big-endian* → most significant byte tại high memory address (Motorola)
      - *little-endian* → least significant byte tại high memory address (Intel x86)
    - Dạng biểu diễn XDR (External Data Representation): biểu diễn dữ liệu machine-independent

Truyền tham số trong RPC (tt.)



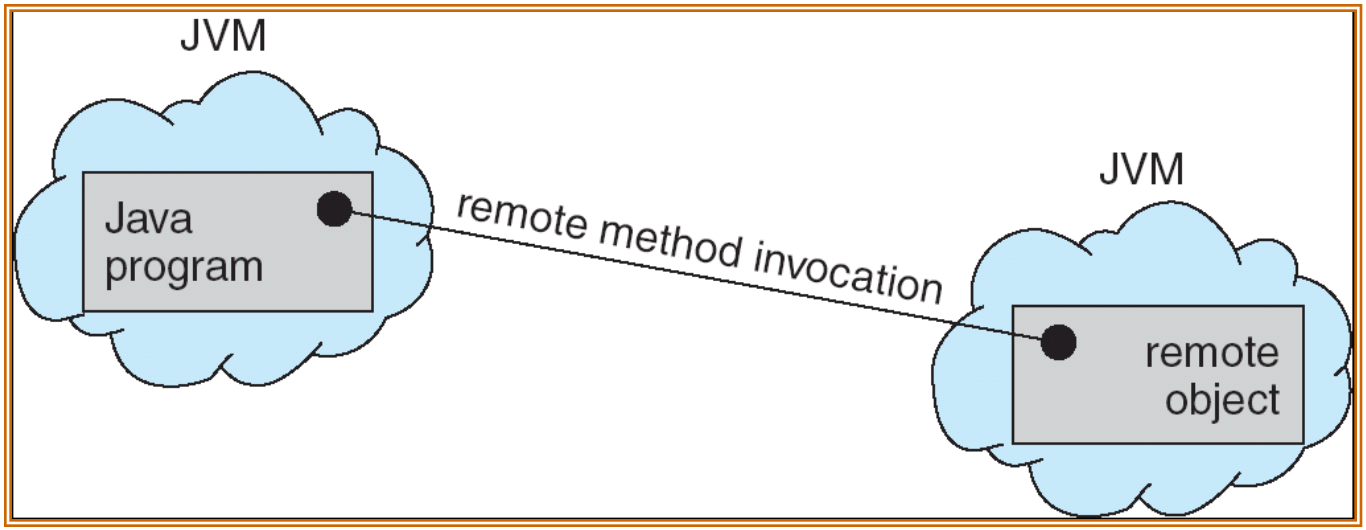
Quá trình thực hiện RPC

Dùng *dynamic binding* để xác định port number của RPC X

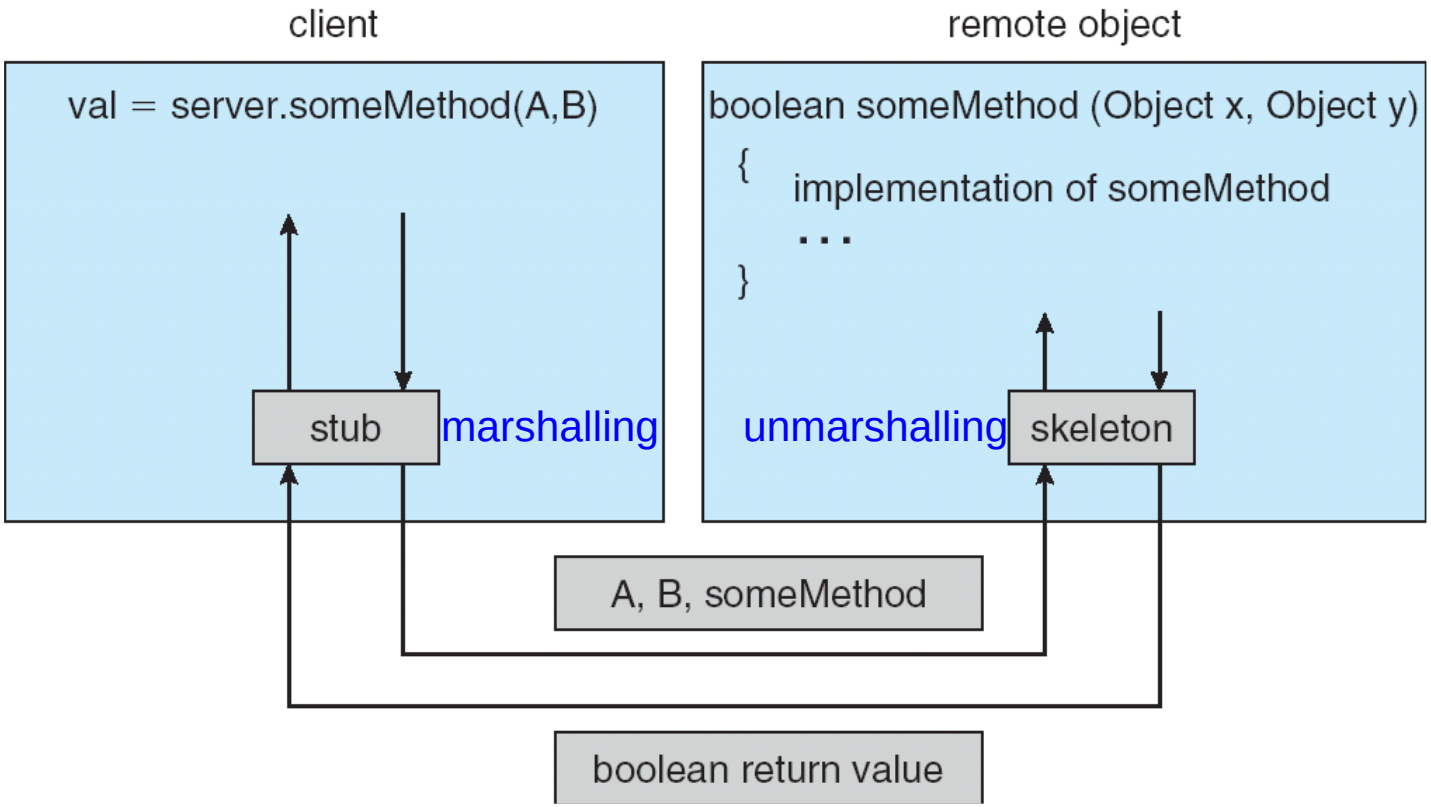


Remote method invocation

- *Remote Method Invocation* (RMI)
  - Cho phép một chương trình Java có thể gọi một phương thức (method) của một *đối tượng ở xa*, nghĩa là một đối tượng ở tại một máy ảo Java khác



Marshalling tham số trong RMI



Phương thức được triệu gọi có dạng sau:

boolean someMethod(Object x, Object y)