

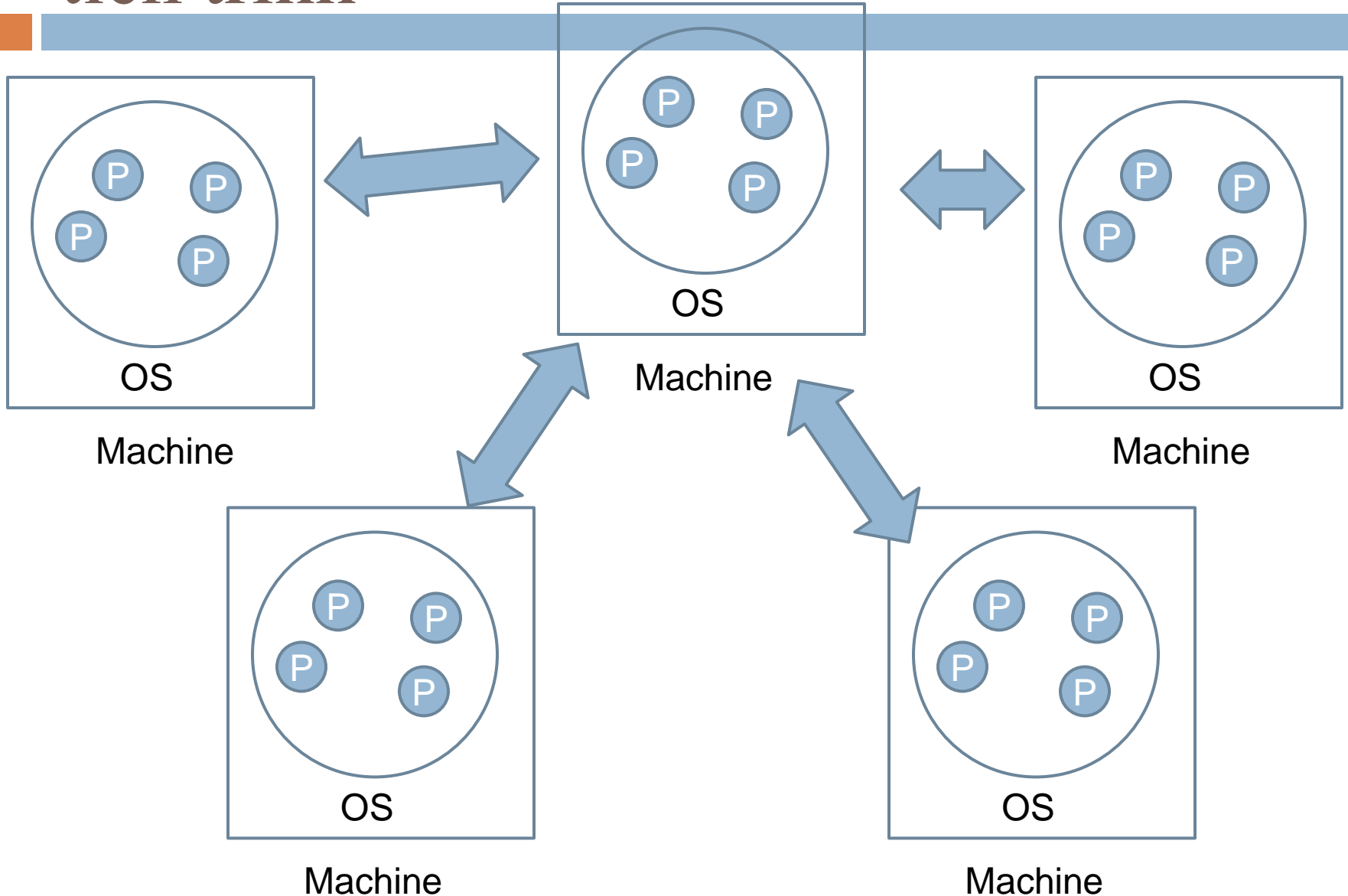


# CHƯƠNG 3: QUẢN LÝ TIẾN TRÌNH VÀ LUỒNG

TS. Trần Hải Anh

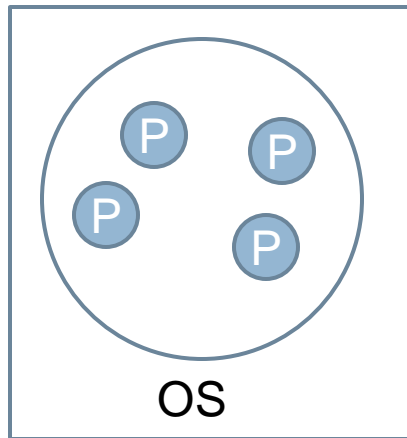
# Vai trò của hệ điều hành trong quản lý tiến trình

2

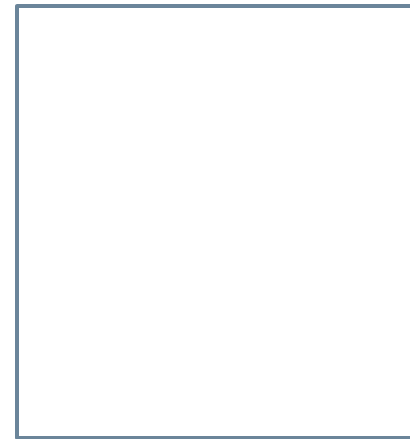


# Ảo hóa

3



Machine A



Machine B

# Nội dung

4

1. Tiến trình và luồng
2. Khái niệm ảo hóa
3. Clients
4. Servers
5. Di trú mã

# 1. Tiến trình và luồng

1.1. Khái niệm

1.2. Luồng trong hệ thống tập trung

1.3. Luồng trong hệ thống phân tán

# 1.1. Tiến trình và luồng

6

- Tiến trình
  - ▣ Chương trình đang hoạt động
  - ▣ Tài nguyên:
    - Virtual Processor
    - Virtual Memory
  - ▣ Trong suốt tương tranh
  - ▣ Quá trình tạo 1 tiến trình
  - ▣ Chuyển ngữ cảnh giữa các tiến trình

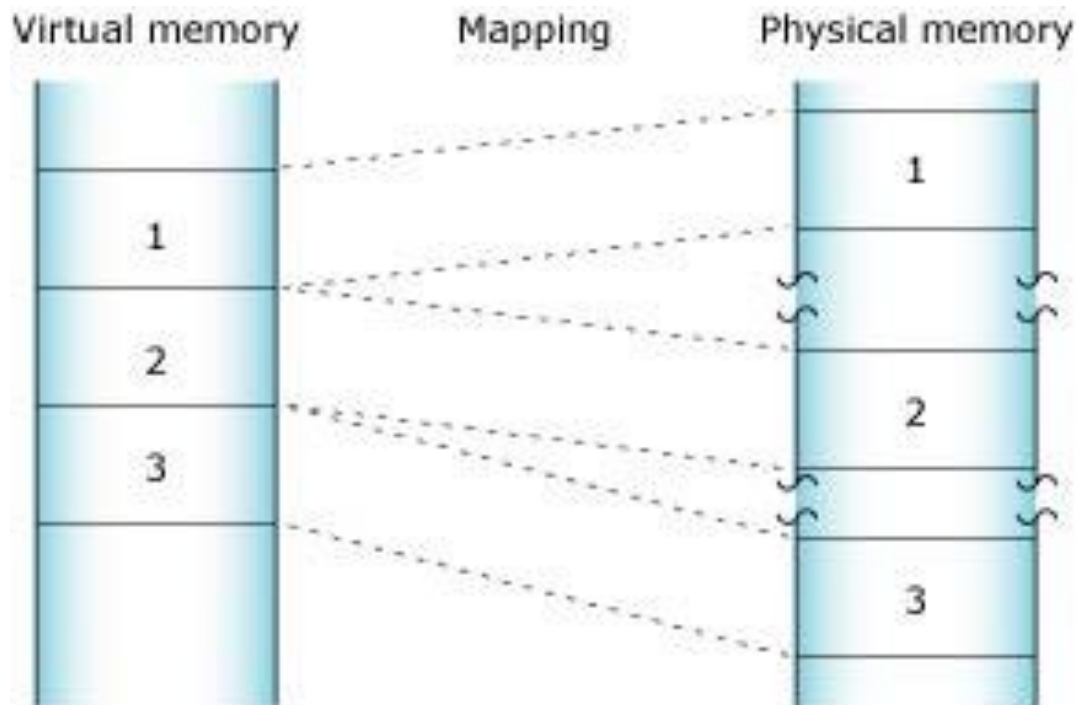
# Luồng

7

- Là một luồng thực thi của tiến trình.
- Tiến trình có nhiều luồng thực thi → Tiến trình đa luồng
- Các luồng của tiến trình dùng môi trường thực hiện chung của tiến trình: trạng thái của CPU
- Trao đổi thông tin giữa các luồng thông qua các biến chia sẻ
- An toàn và hợp lý của tương tác luồng do lập trình viên quyết định
- Luồng=> hiệu năng+chi phí lập trình

# Virtual Memory

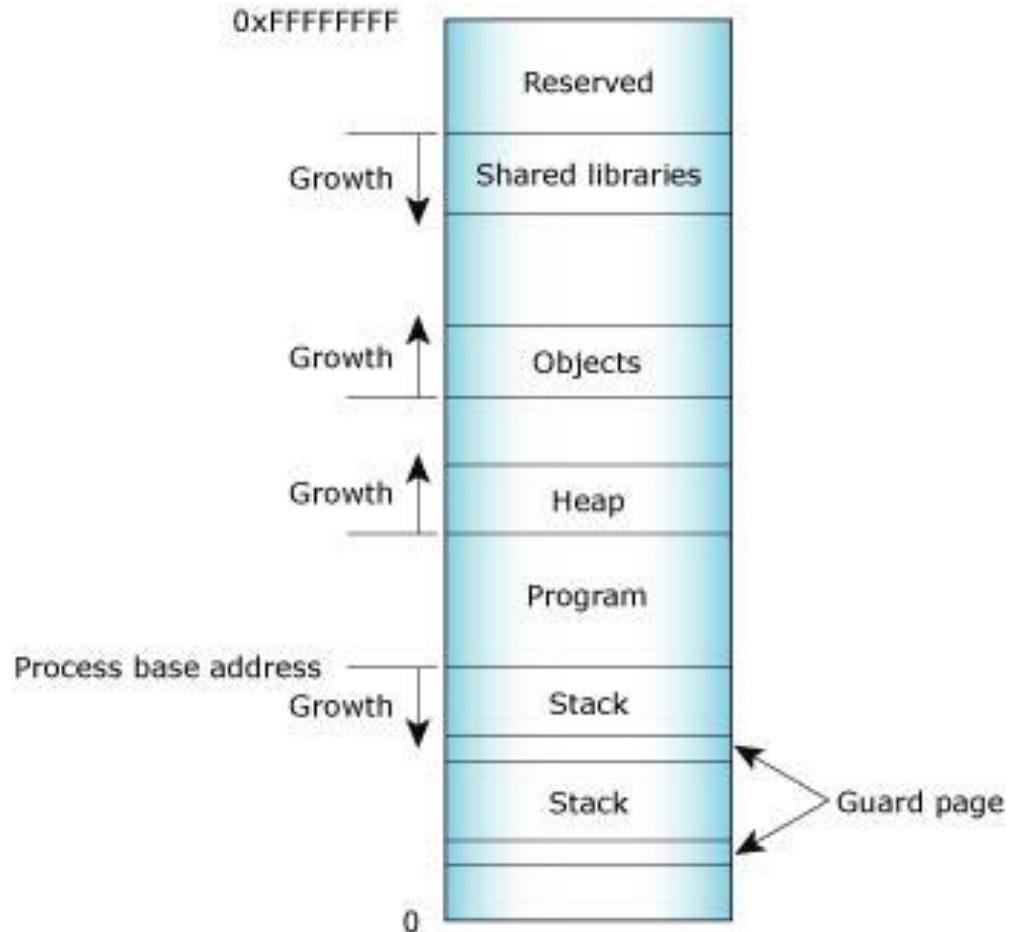
8





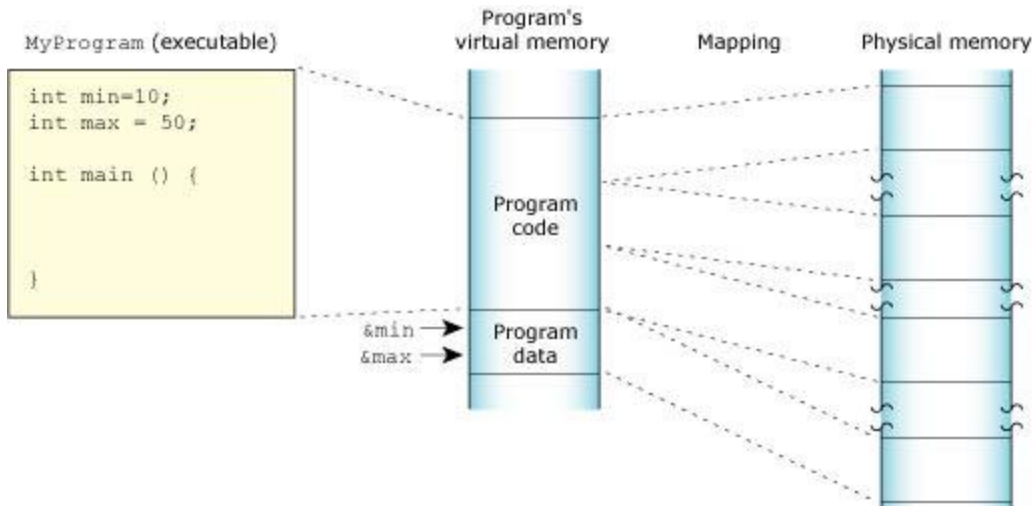
# Process Memory layout

9

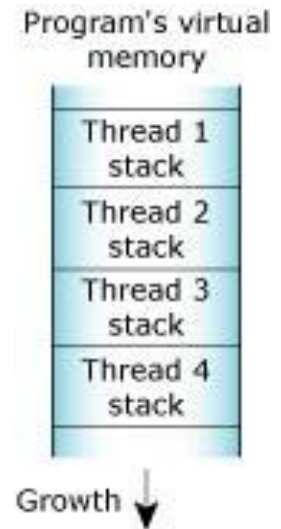


# Program and Stack memory

10



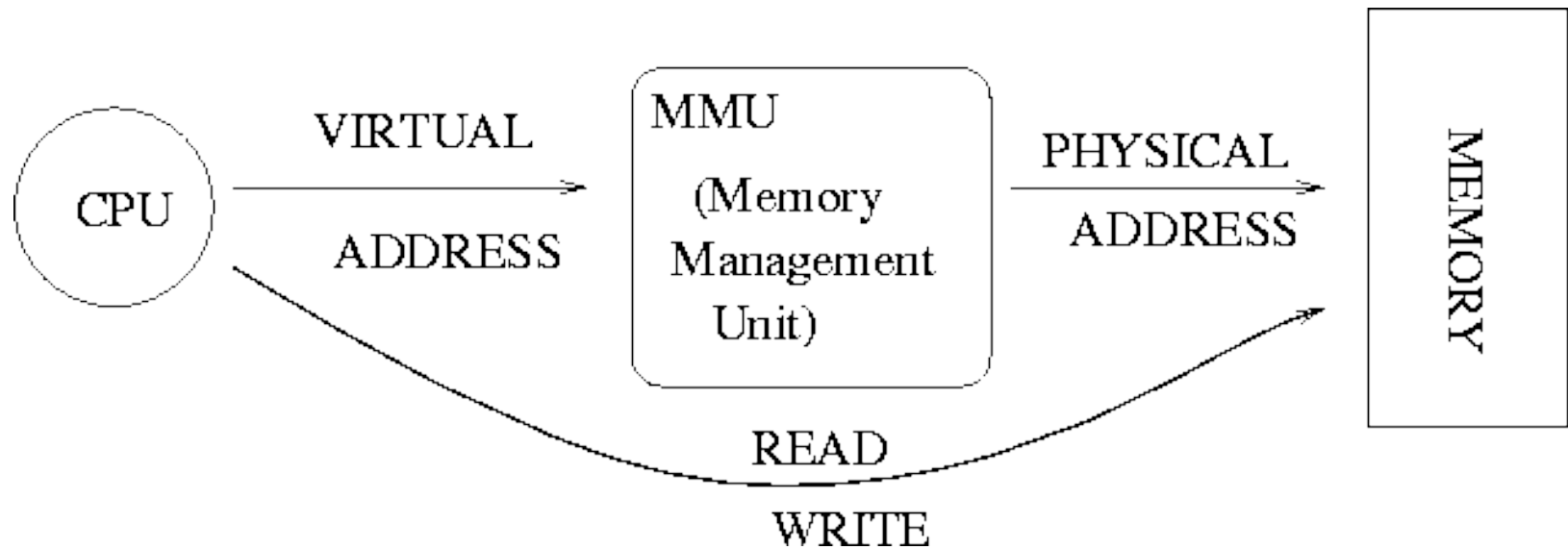
**Program memory**



**Stack memory**

# Cơ chế ánh xạ

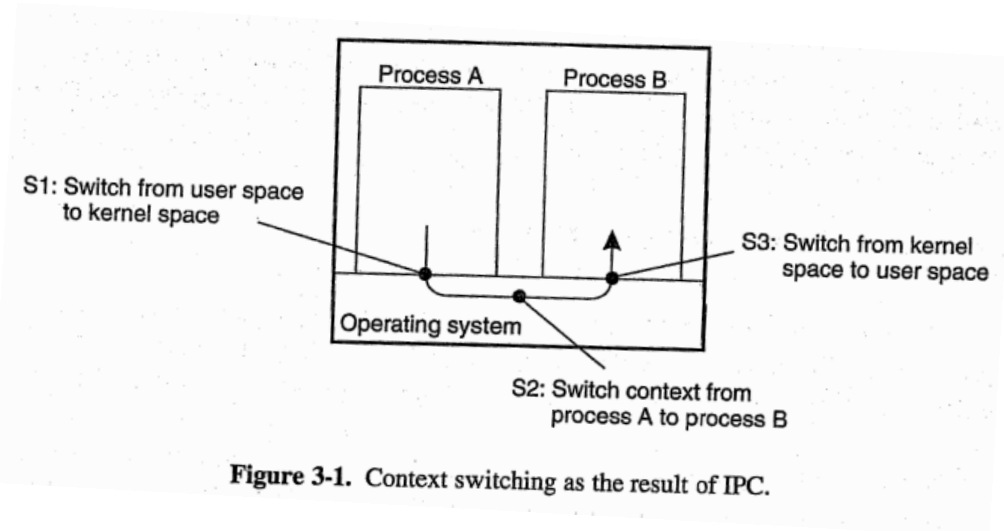
11



# 1.2. Luồng trong các hệ thống tập trung

12

- Lợi ích của tiến trình đa luồng:
  - ▣ Không bị dừng chương trình với các lời gọi hệ thống dừng
  - ▣ Tính toán song song
  - ▣ Đối với các chương trình lớn (nhiều module nhỏ) → đa luồng



# Cài đặt luồng

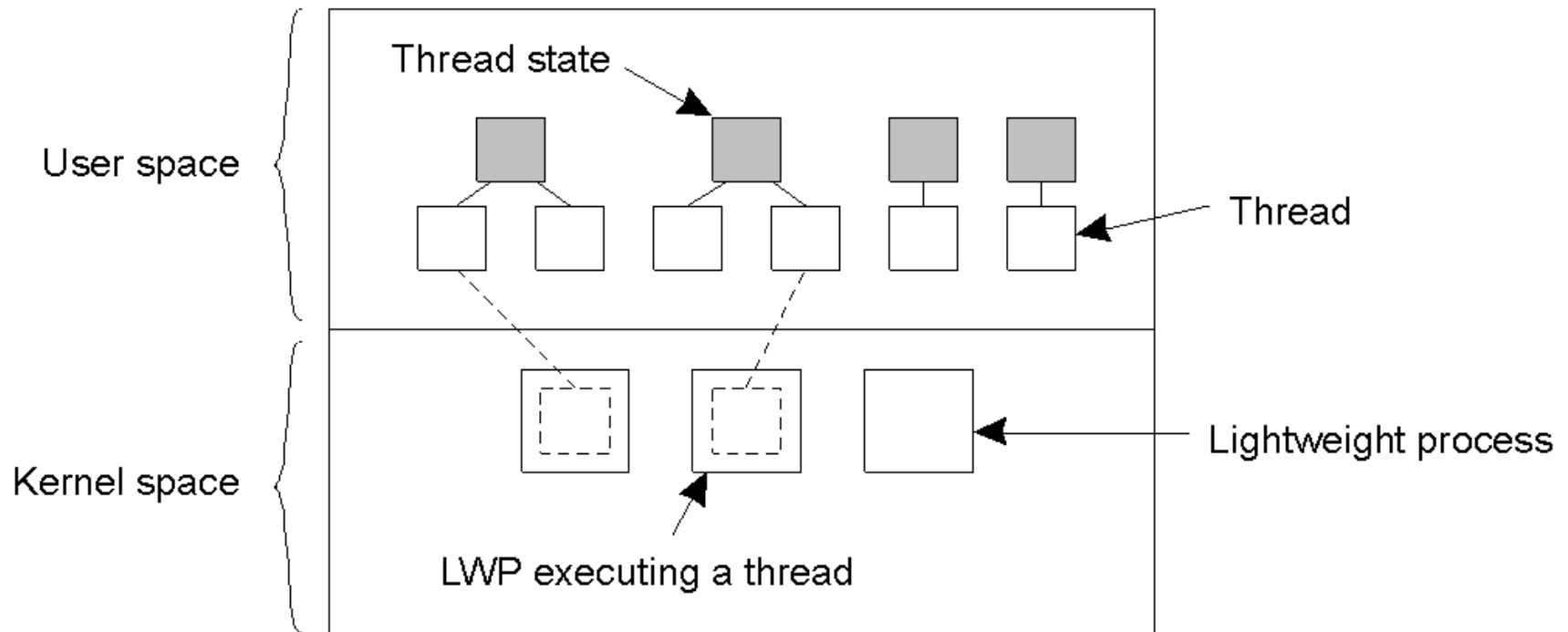
13

- Được quản lý bởi gói luồng (Thread package)
  - ▣ Khởi tạo luồng (1)
  - ▣ Giải phóng luồng (2)
  - ▣ Đồng bộ các luồng (3)
- (1), (2), (3) có thể thực hiện dưới chế độ NSD hoặc nhân
  - ▣ Chế độ NSD: có thao tác vào ra-> block cả tiến trình
  - ▣ Chế độ nhân: Tồn kém

# Cài đặt luồng: các tiến trình nhẹ

14

- Combining kernel-level lightweight processes and user-level threads.



# LINUX triển khai các luồng

15

- ❑ Không có sự phân biệt thread và process, tất cả chỉ là task
- ❑ Luồng ở mức user được xây dựng theo chuẩn POSIX (Portable Operating System Interface for uniX)
- ❑ Chạy ở 2 không gian thực thi phân biệt:
  - ▣ User space: sử dụng thư viện pthread
  - ▣ Kernel: các LWPs
- ❑ Ánh xạ 1-1 từ mỗi thread và 1 LWP
- ❑ Thay vì dùng fork(), LINUX dùng clone().

# Quản lý ID

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

void * function1(void *arg)
{
    pthread_t tid=pthread_self();
    printf("In thread %u and process %u\n",tid,getpid());
}

void * function2(void *arg)
{
    pthread_t tid=pthread_self();
    printf("In thread %u and process %u\n",tid,getpid());
}

int main()
{
    void *status;
    pthread_t tid1,tid2;
    pthread_attr_t attr;

    if(pthread_create(&tid1,NULL,function1,NULL)){
        perror("Failure");
        exit(1);
    }

    if(pthread_create(&tid2,NULL,function2,NULL)){
        perror("Failure");
        exit(2);
    }

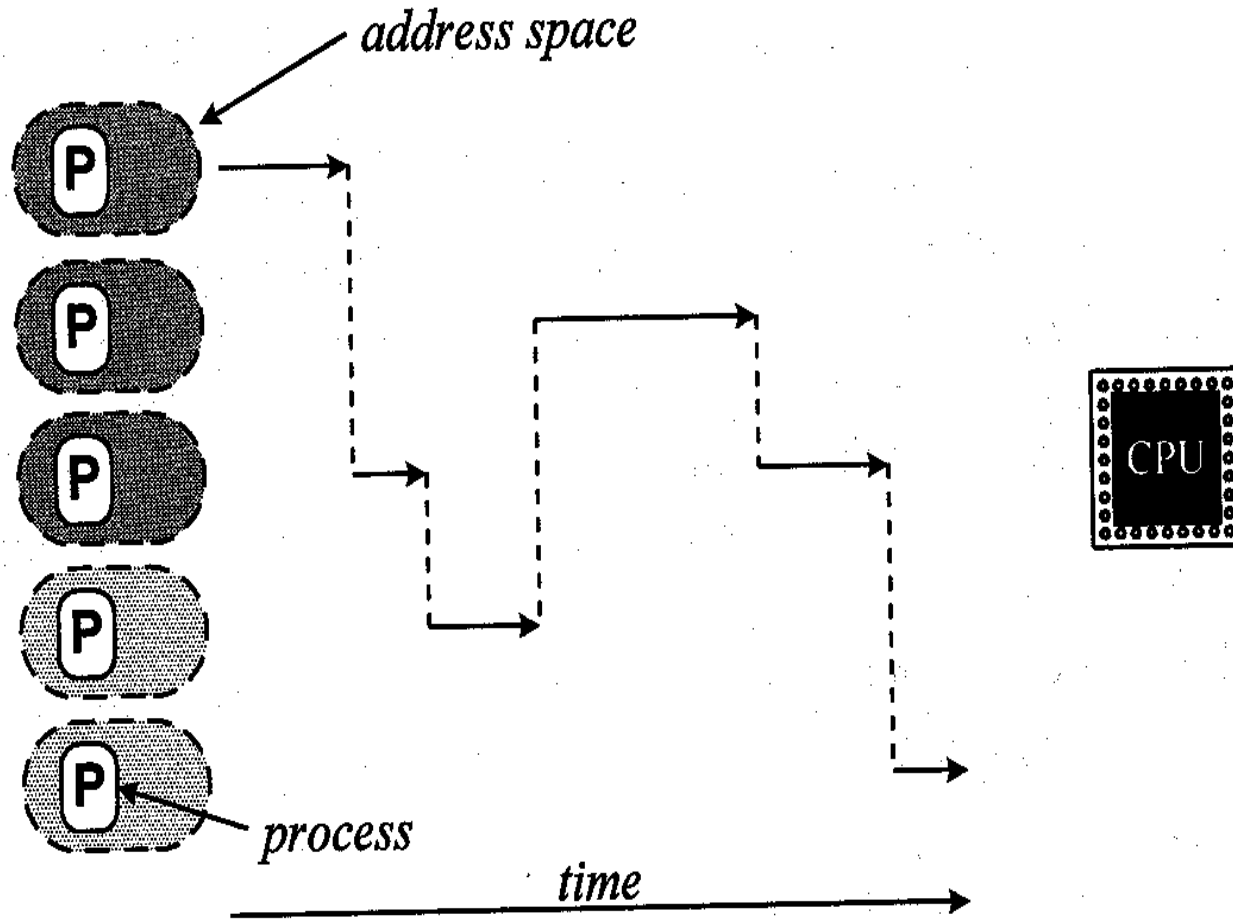
    pthread_join(tid1,NULL);
    pthread_join(tid2,NULL);
    printf("In main thread %u and process %u\n",pthread_self(),getpid());
}
```



In thread 3086625680 and process 5480  
In thread 3076135824 and process 5480  
In main thread 3086628544 and process 5480

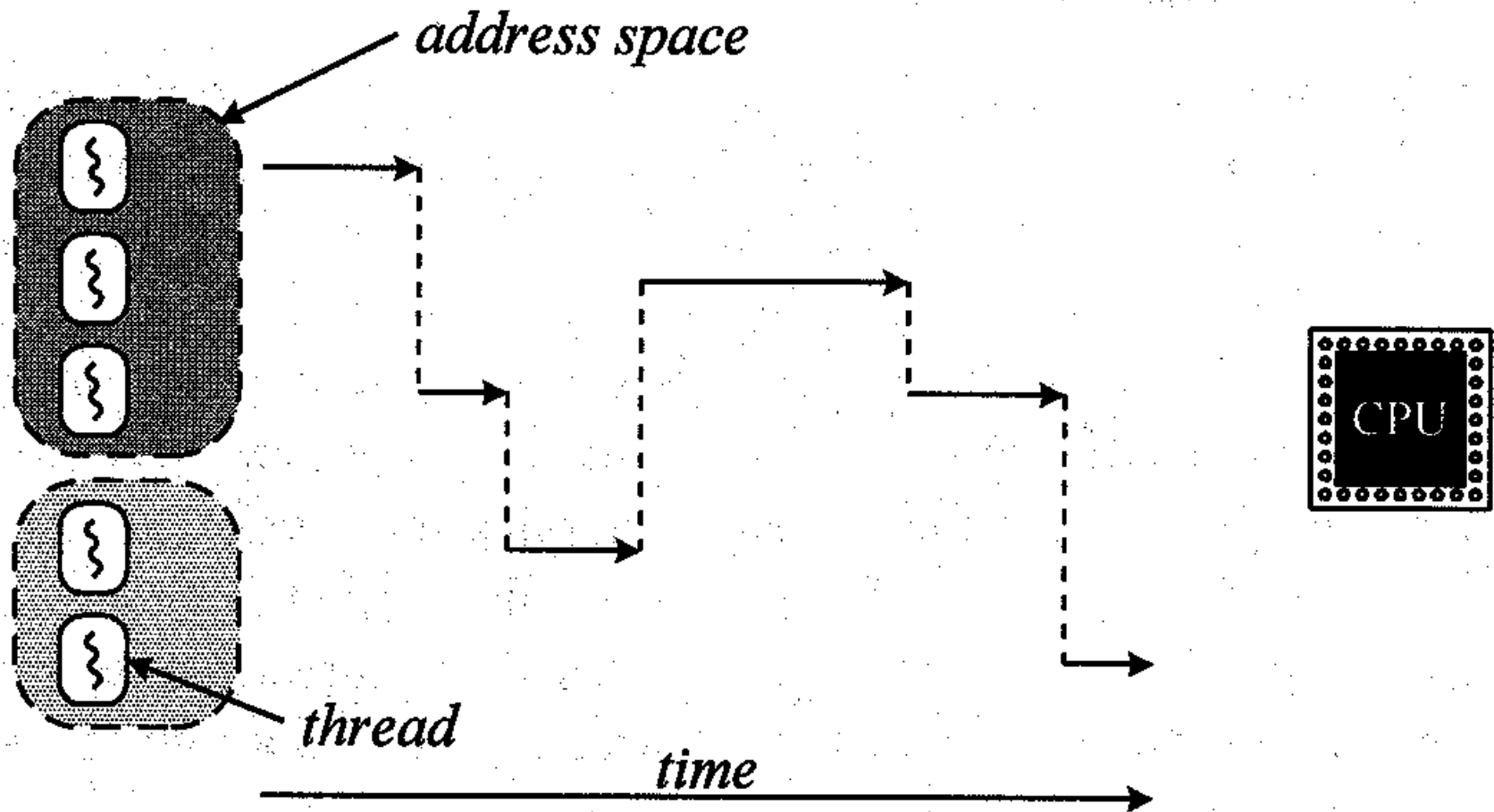


# Traditional UNIX system



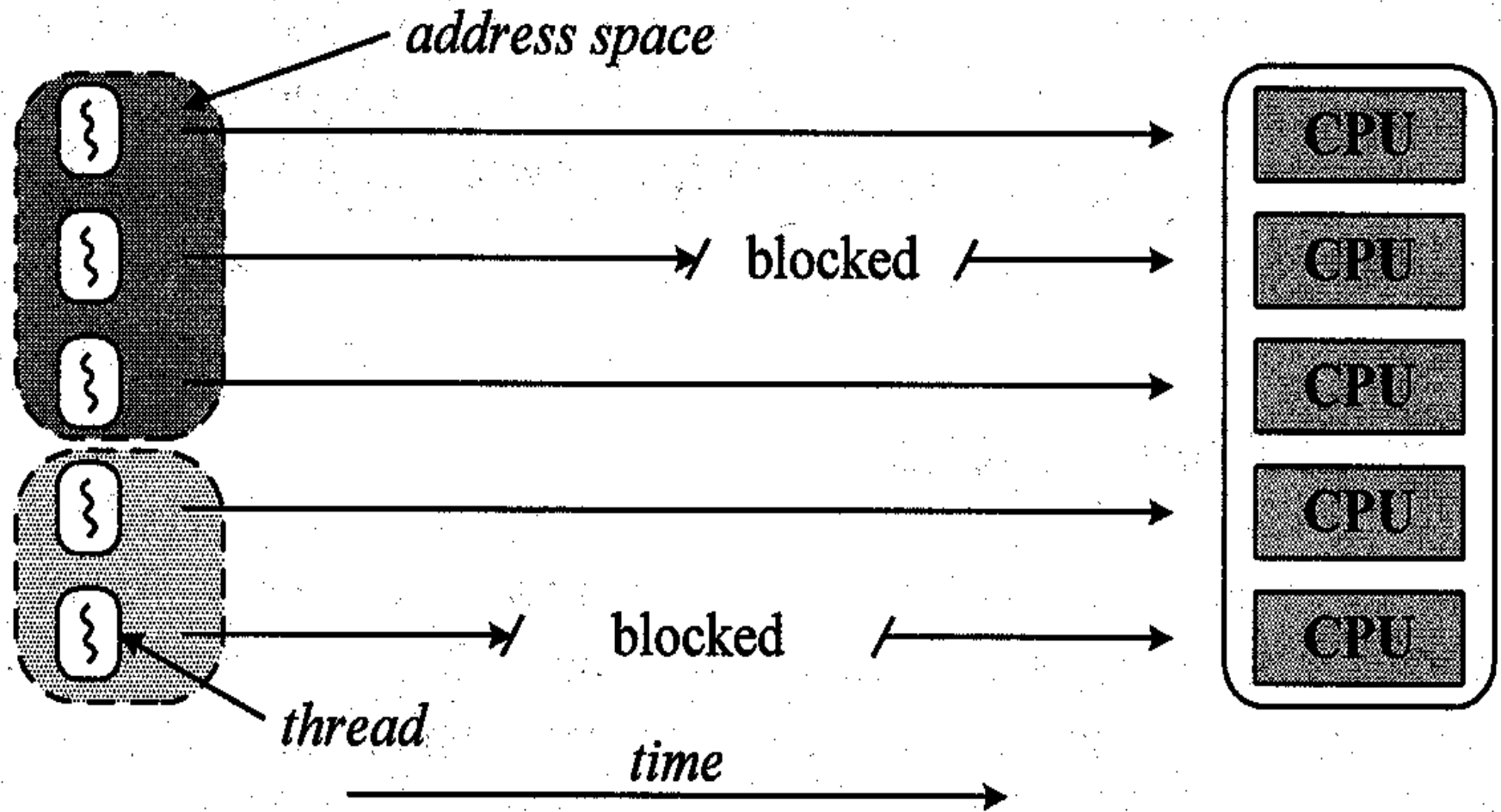
**Figure 3-1.** Traditional UNIX system—uniprocessor with single-threaded processes.

# Multithreaded Processes



**Figure 3-2.** Multithreaded processes in a uniprocessor system.

# Multiprocessor



**Figure 3-3.** Multithreaded processes on a multiprocessor.

# 1.3. Luồng trong các hệ thống phân tán

20

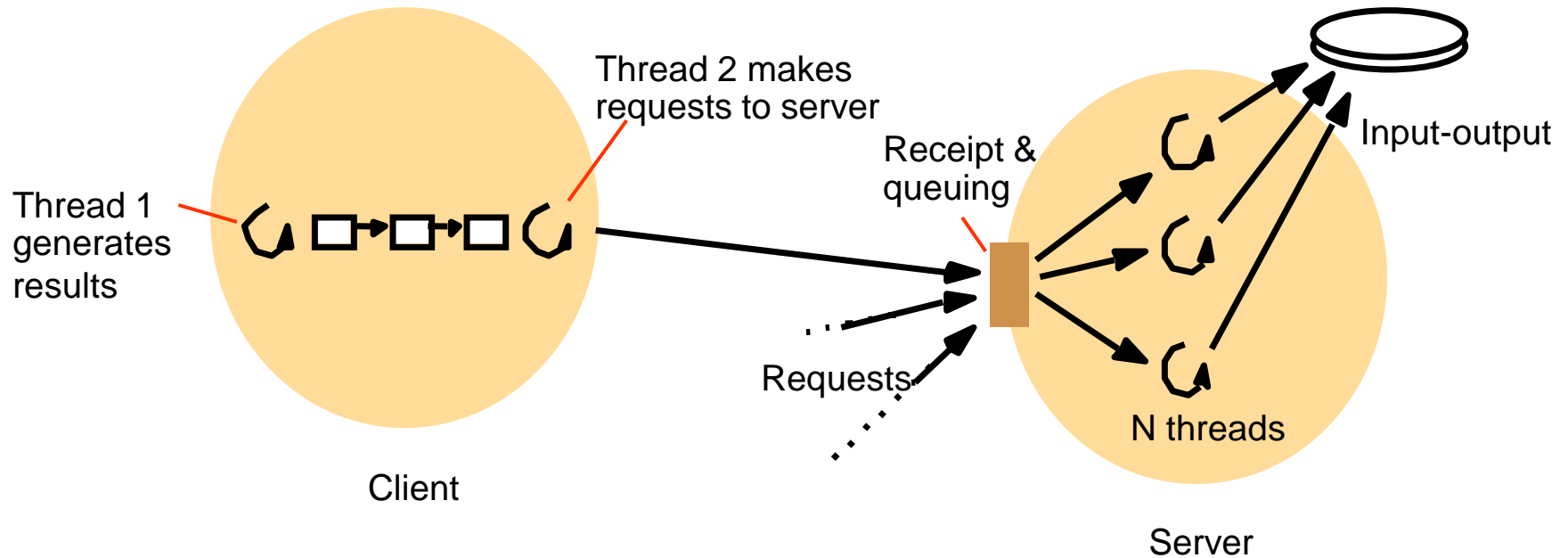
- Server đơn luồng
  - ▣ Chỉ xử lý được một yêu cầu tại một thời điểm
  - ▣ Các yêu cầu có thể được xử lý tuần tự
  - ▣ Các yêu cầu có thể được xử lý bởi các tiến trình khác nhau
  - ▣ Không đảm bảo tính trong suốt



**Server đa luồng**

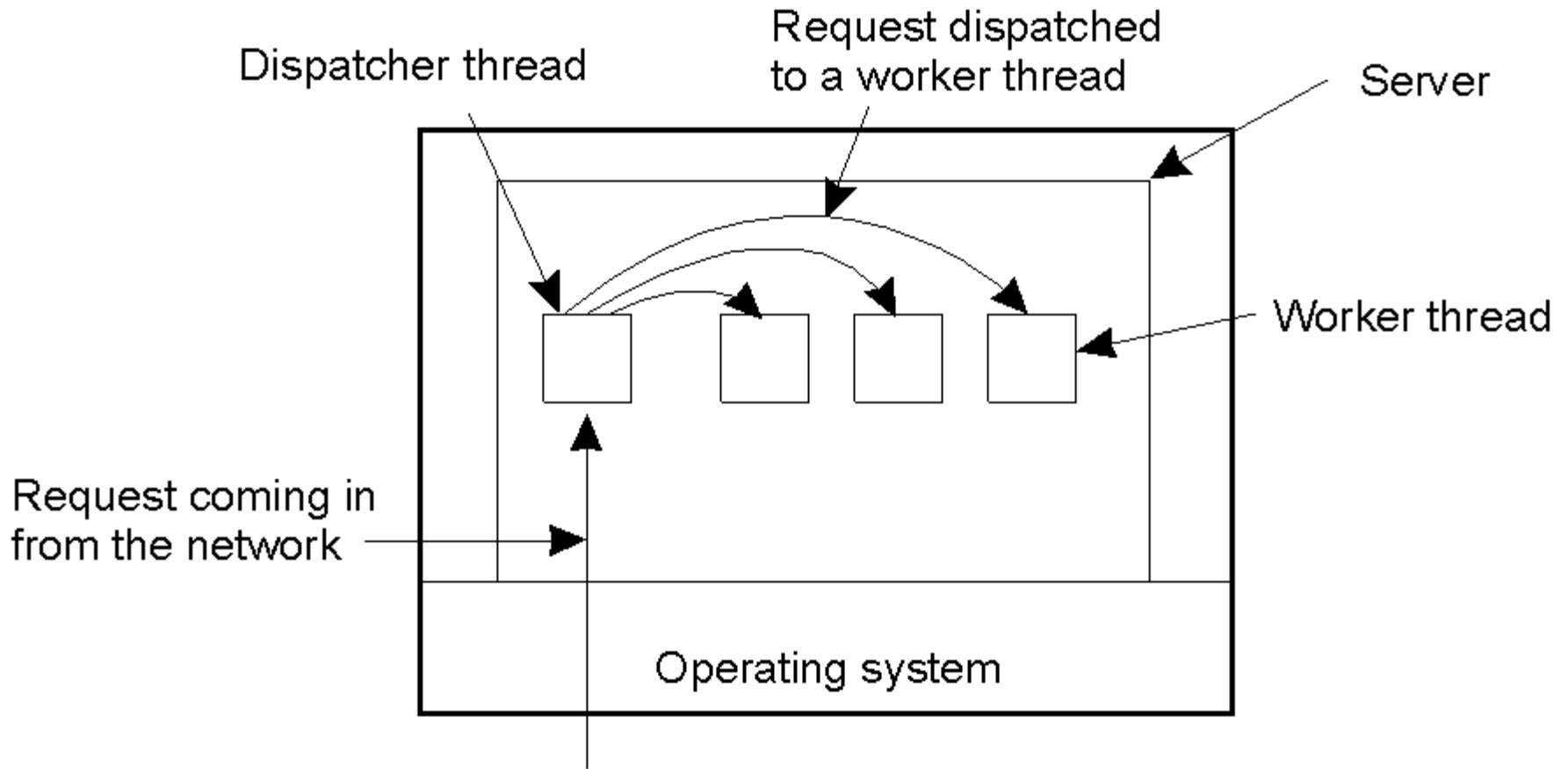
# Client và server đa luồng

21



# Mô hình server dispatcher

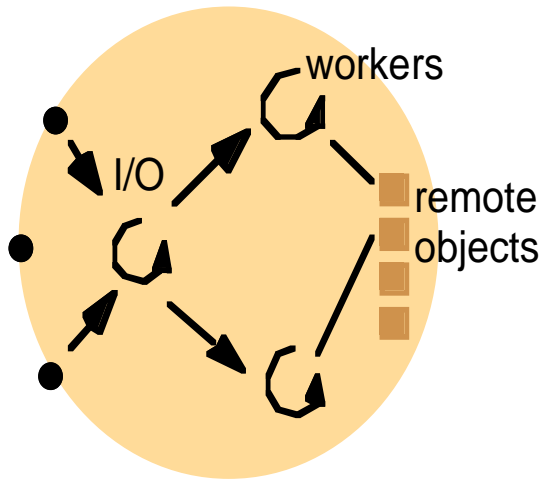
22



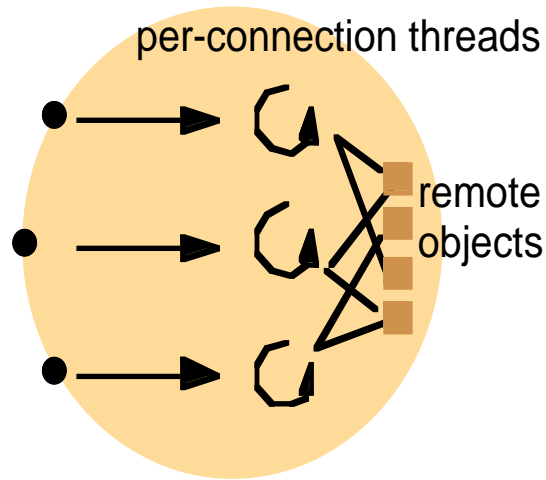
- Dispatcher (điều phối viên)/worker (người xử lý)

# Server đa luồng

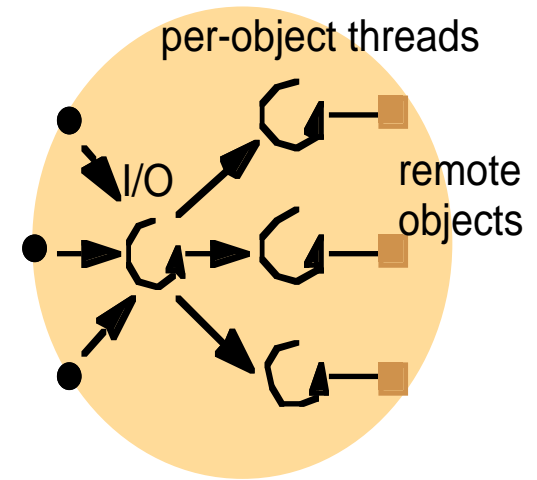
23



a. Thread-per-request



b. Thread-per-connection



c. Thread-per-object

# Mô hình máy trạng thái

24

- ❑ Các yêu cầu từ client và xử lý được sắp hàng
- ❑ Tại một thời điểm server thực hiện thao tác trong hàng
- ❑ Không cần đa luồng
- ❑ Các lời gọi xử lý là các lời gọi “không dừng”



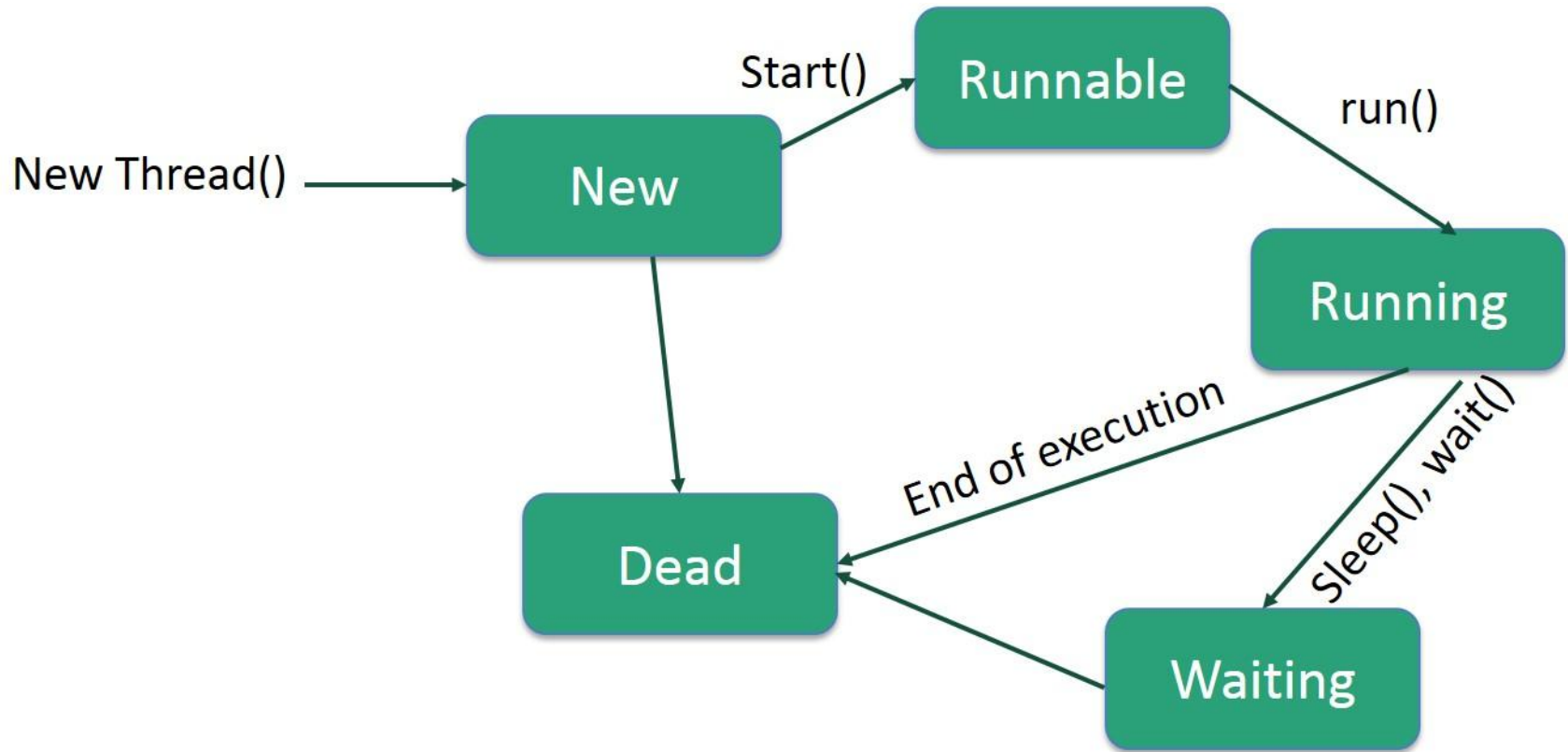
# Client đa luồng

25

- ❑ Tách biệt giao diện người sử dụng và xử lý
- ❑ Giải quyết vấn đề các thao tác chờ đợi lẫn nhau
- ❑ Tăng tốc độ khi làm việc với nhiều server khác nhau
- ❑ Che giấu các chi tiết cài đặt
- ❑ Ví dụ: Tải trang web

# Multithreading in Java

26



- Tạo thread có thể thực hiện bằng 2 cách:
  - ▣ Kế thừa lớp Thread
  - ▣ Triển khai interface Runnable
- Các phương thức
  - ▣ getName(): It is used for Obtaining a thread's name
  - ▣ getPriority(): Obtain a thread's priority
  - ▣ isAlive(): Determine if a thread is still running
  - ▣ join(): Wait for a thread to terminate
  - ▣ run(): Entry point for the thread
  - ▣ sleep(): suspend a thread for a period of time
  - ▣ start(): start a thread by calling its run() method

# Multithreading in Java

28

```
class RunnableDemo implements Runnable {
    private Thread t;
    private String threadName;

    RunnableDemo( String name){
        threadName = name;
        System.out.println("Creating " + threadName );
    }
    public void run() {
        System.out.println("Running " + threadName );
        try {
            for(int i = 4; i > 0; i--) {
                System.out.println("Thread: " + threadName + ", " + i);
                // Let the thread sleep for a while.
                Thread.sleep(50);
            }
        } catch (InterruptedException e) {
            System.out.println("Thread " + threadName + " interrupted.");
        }
        System.out.println("Thread " + threadName + " exiting.");
    }

    public void start ()
    {
        System.out.println("Starting " + threadName );
        if (t == null)
        {
            t = new Thread (this, threadName);
            t.start ();
        }
    }
}
```

# Multithreading in Java

29

```
public class TestThread {  
    public static void main(String args[]) {  
  
        RunnableDemo R1 = new RunnableDemo( "Thread-1");  
        R1.start();  
  
        RunnableDemo R2 = new RunnableDemo( "Thread-2");  
        R2.start();  
    }  
}
```



```
Creating Thread-1  
Starting Thread-1  
Creating Thread-2  
Starting Thread-2  
Running Thread-1  
Thread: Thread-1, 4  
Running Thread-2  
Thread: Thread-2, 4  
Thread: Thread-1, 3  
Thread: Thread-2, 3  
Thread: Thread-1, 2  
Thread: Thread-2, 2  
Thread: Thread-1, 1  
Thread: Thread-2, 1  
Thread Thread-1 exiting.  
Thread Thread-2 exiting.
```

## 2. Ảo hóa

2.1. Vai trò của Ảo hóa

2.2. Kiến trúc của máy ảo

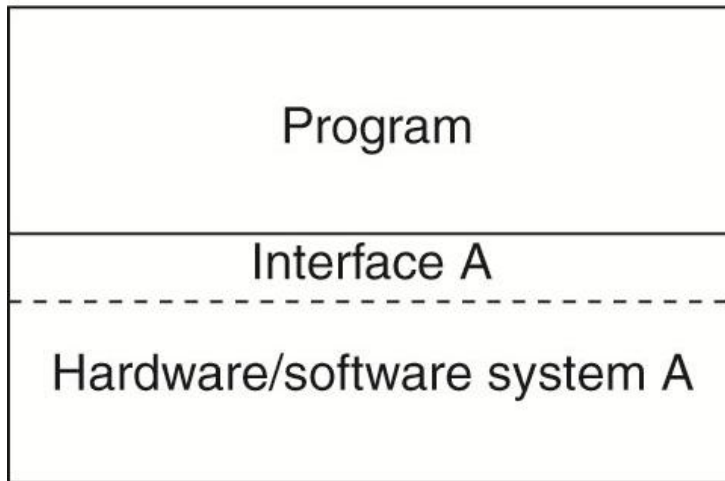
## 2.1. Vai trò của ảo hóa

31

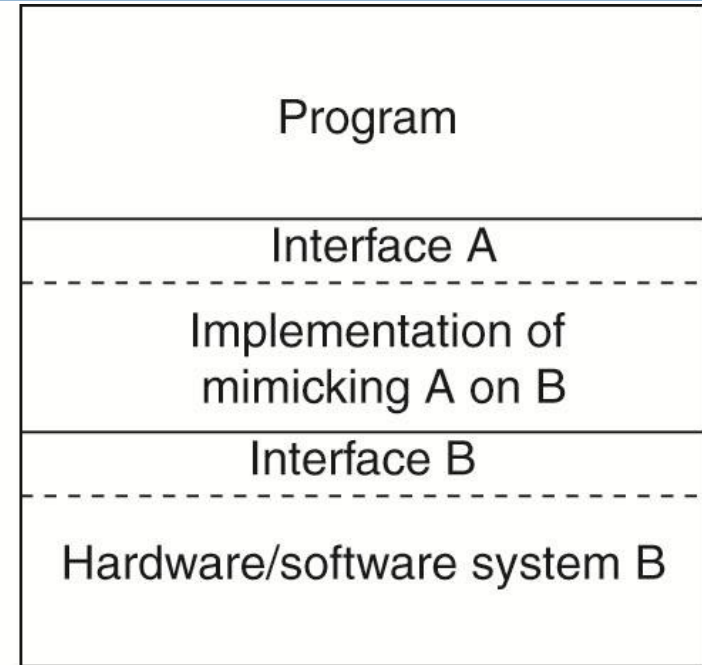
- 1970s: giá thành các máy mainframe cao, cần chạy các phần mềm cũ => sử dụng máy ảo.
- Giá thành phần cứng hạ: sử dụng tối ưu hơn tài nguyên phần cứng
- Tốc độ thay đổi nhanh của phần cứng và các phần mềm hệ thống tầng thấp (1990s): phần mềm tầng trên không được hỗ trợ => ảo hóa
- Mạng máy tính phát triển => hệ thống máy/ứng dụng không đồng nhất, rất đa dạng => nhu cầu chia sẻ giữa các máy trong hệ thống => mỗi ứng dụng chạy trên máy ảo của mình, và tất cả lại chạy trên 1 nền tảng chung.

# Khái niệm ảo hóa

32



(a)



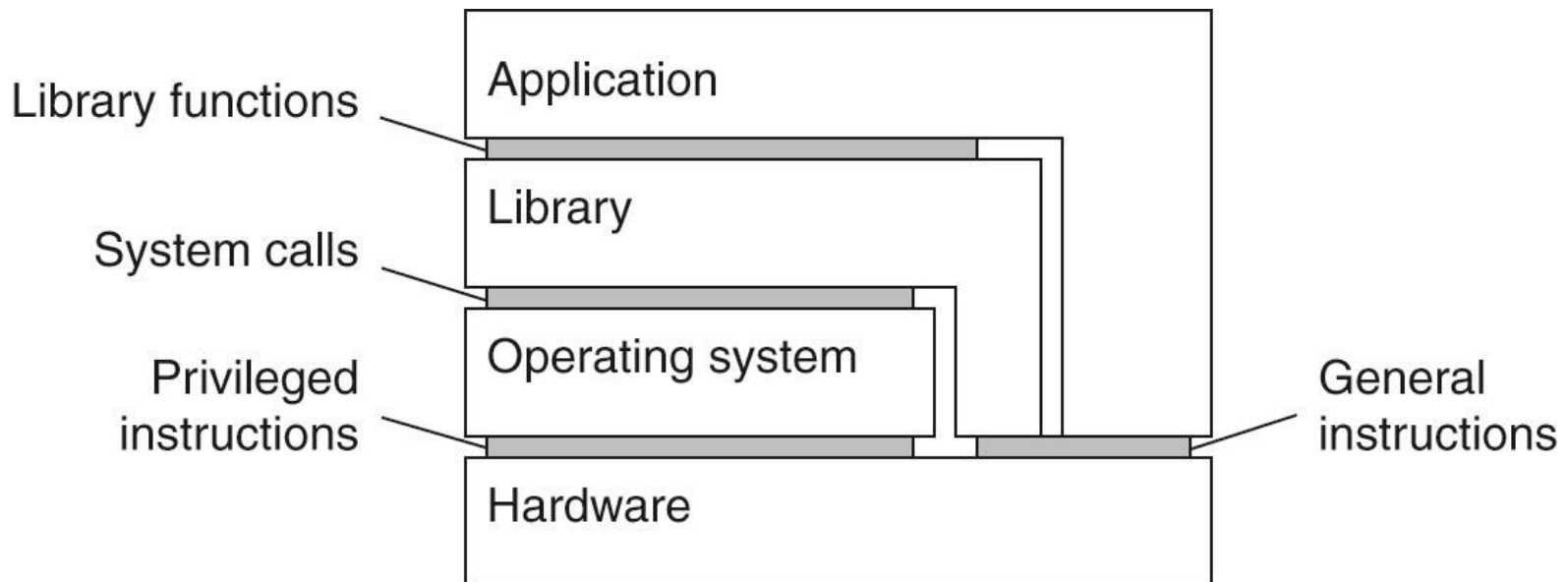
(b)



## 2.2. Các kiến trúc máy ảo

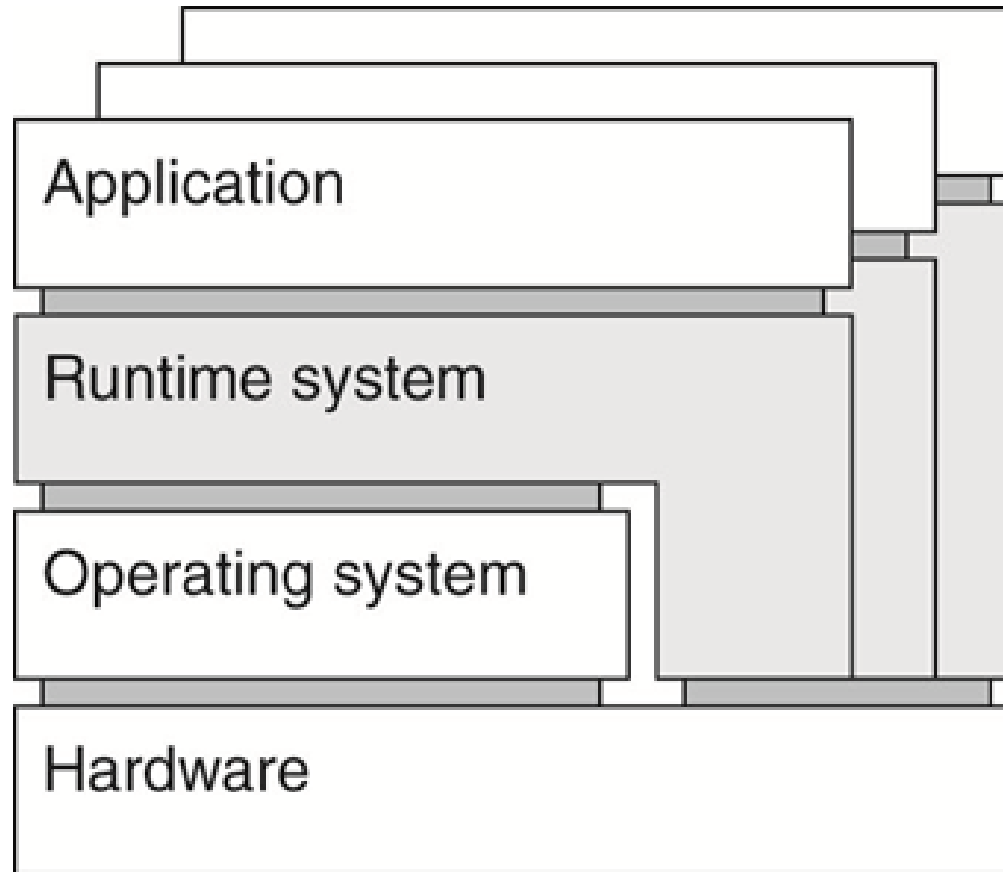
33

- Giao diện với máy tính
  - ▣ Tập lệnh máy cho ứng dụng
  - ▣ Tập lệnh máy cho nhân
  - ▣ Tập lệnh hệ thống
  - ▣ Tập lệnh API



# Kiến trúc mô phỏng hoàn toàn (JVM)

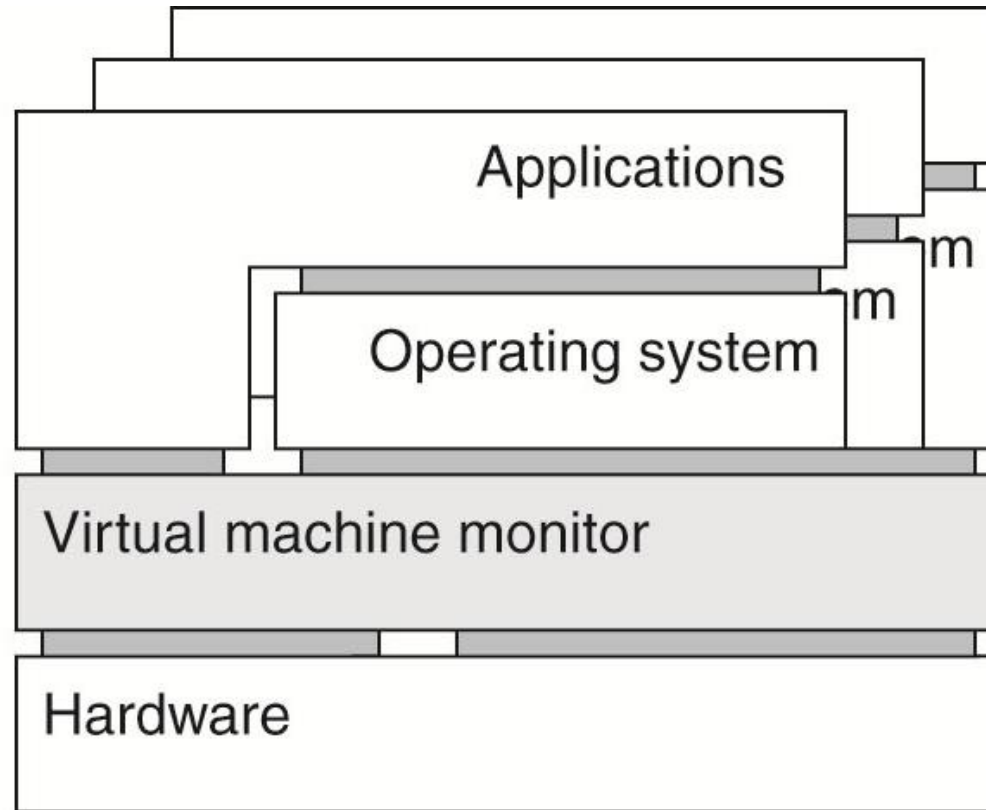
34



(a)

# Kiến trúc kiểm soát

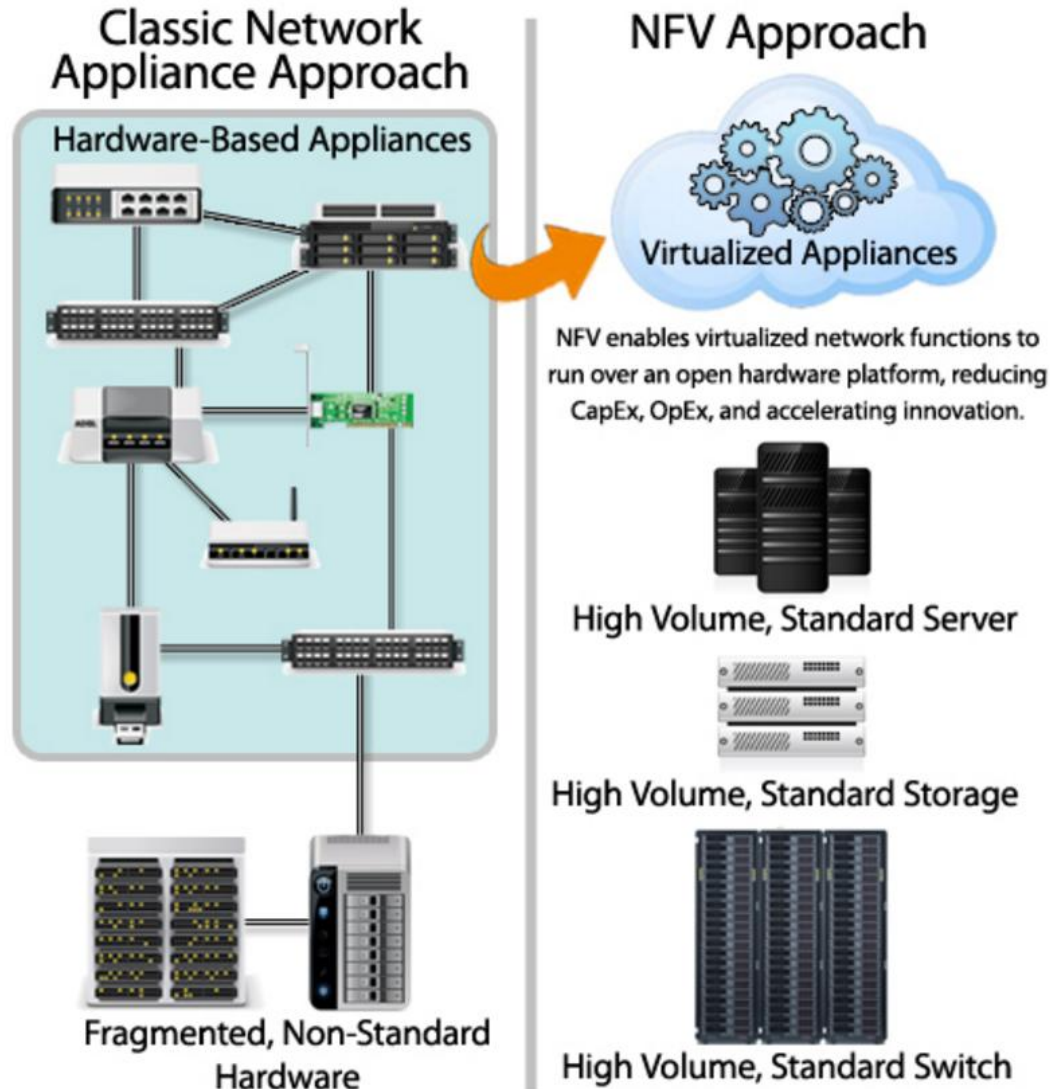
35



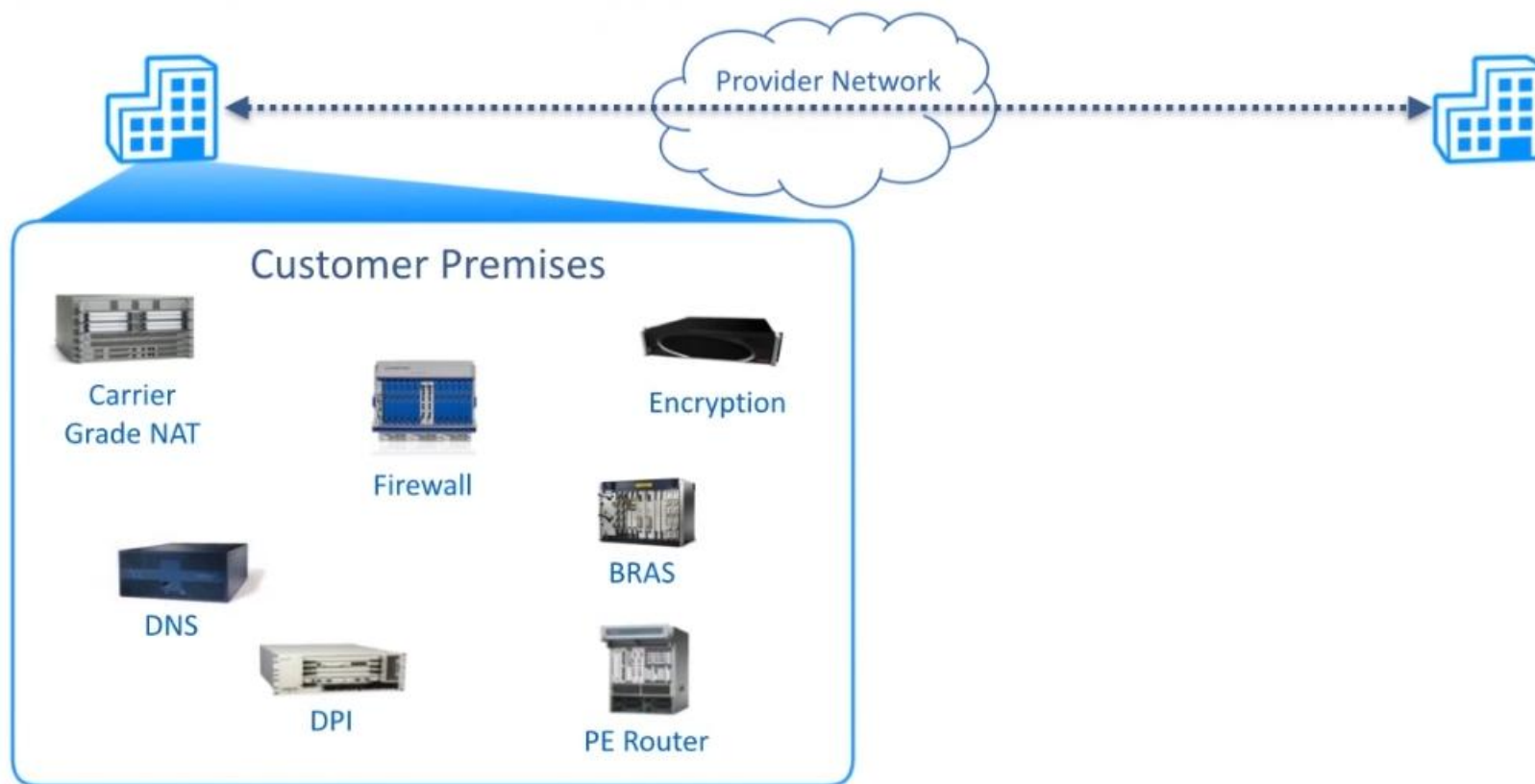
(b)

# Network Function Virtualization

36



# Network Functions



# Network Functions



## Customer Premises



Carrier Grade NAT



Firewall



Encryption



DNS



DPI



BRAS



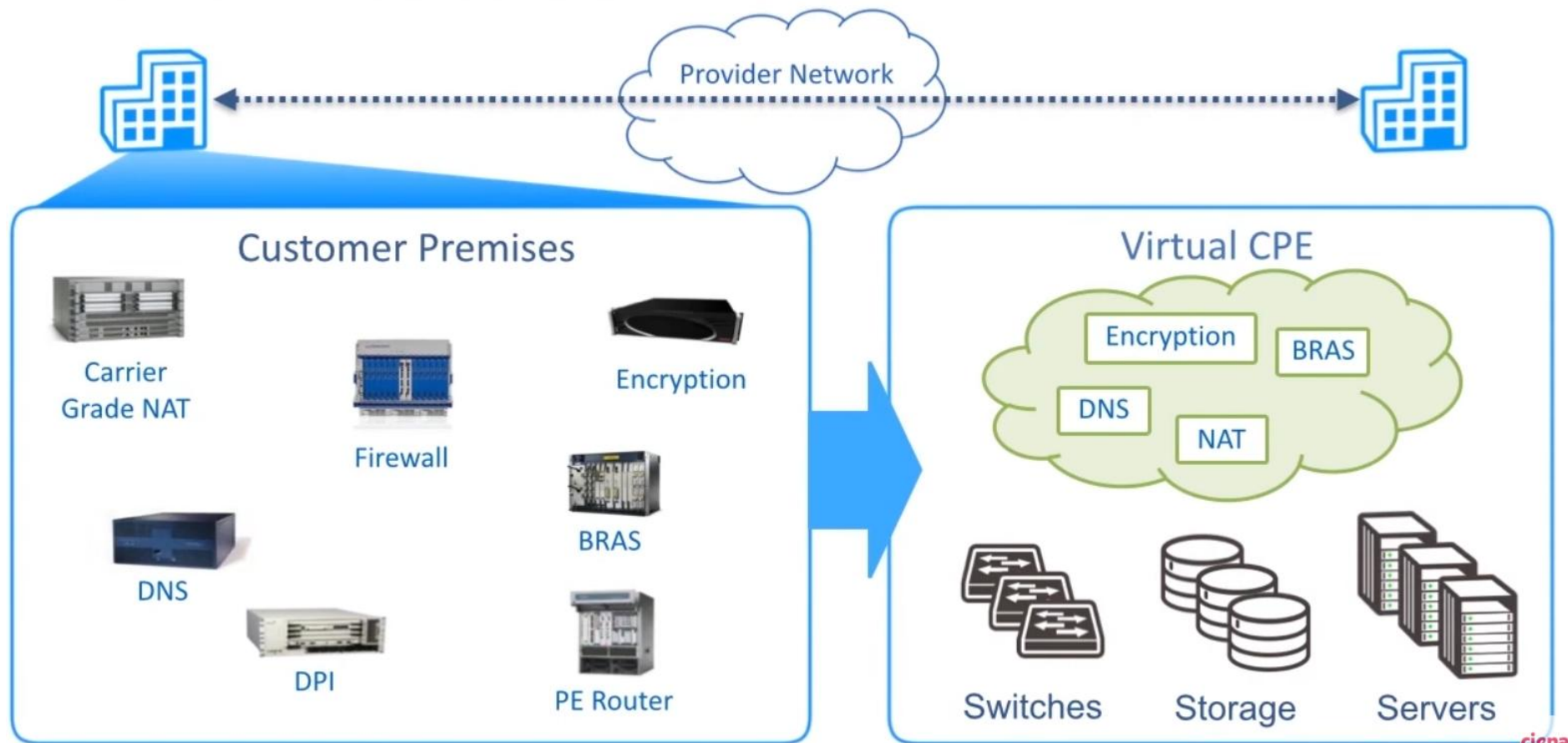
PE Router

- Fragmented, purpose-built hardware
- Physical install per appliance per site
- Difficult upgrades

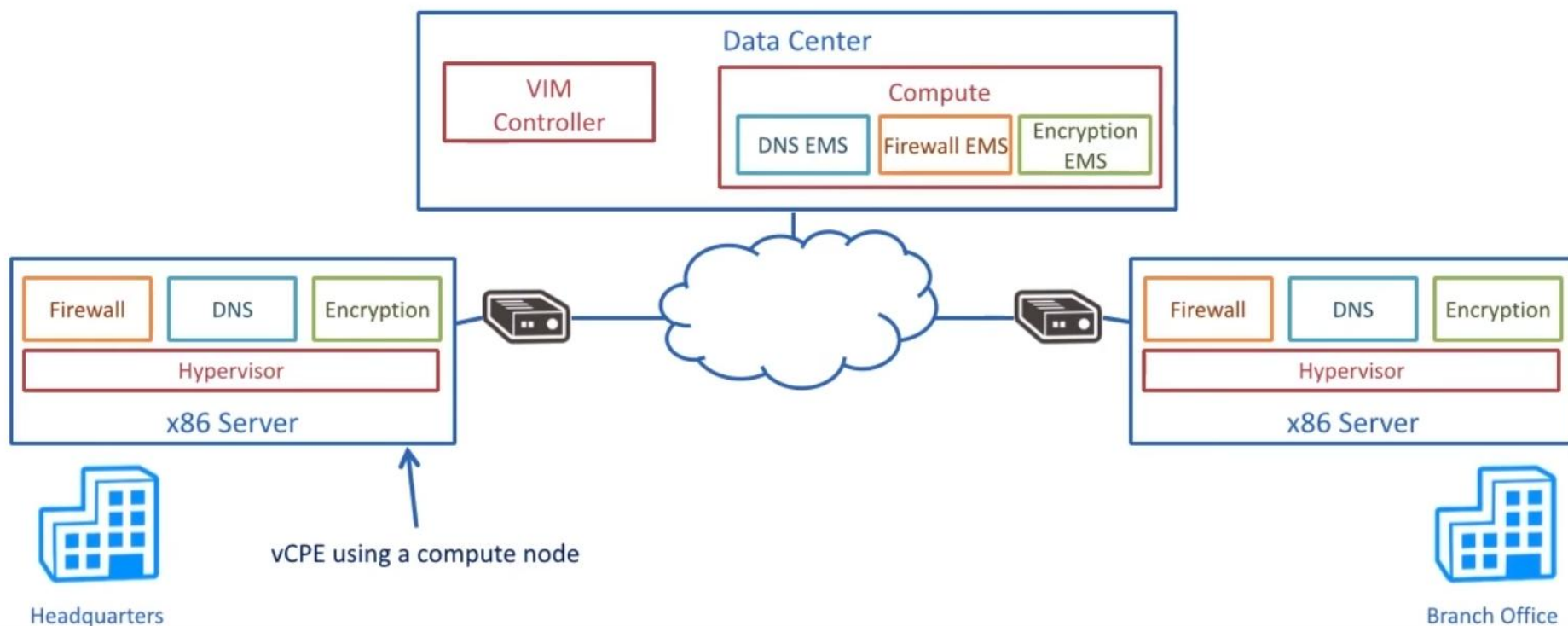


Increased OPEX

# Network Function Virtualization



# Deploying Network Functions





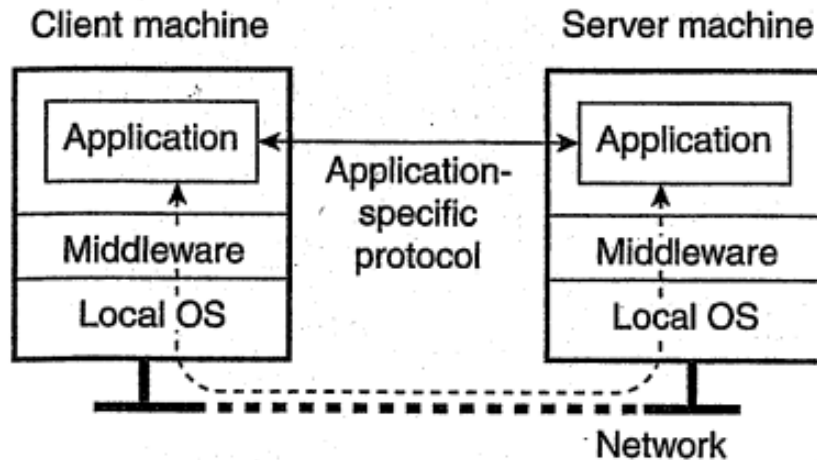
## 3. Clients

Networked User Interfaces

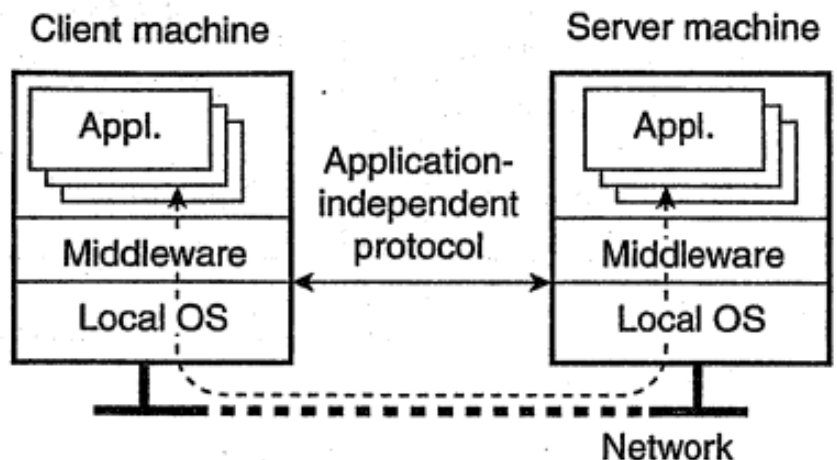
Phần mềm client phục vụ trong suốt phân tán

# 3.1. Networked User Interfaces

42



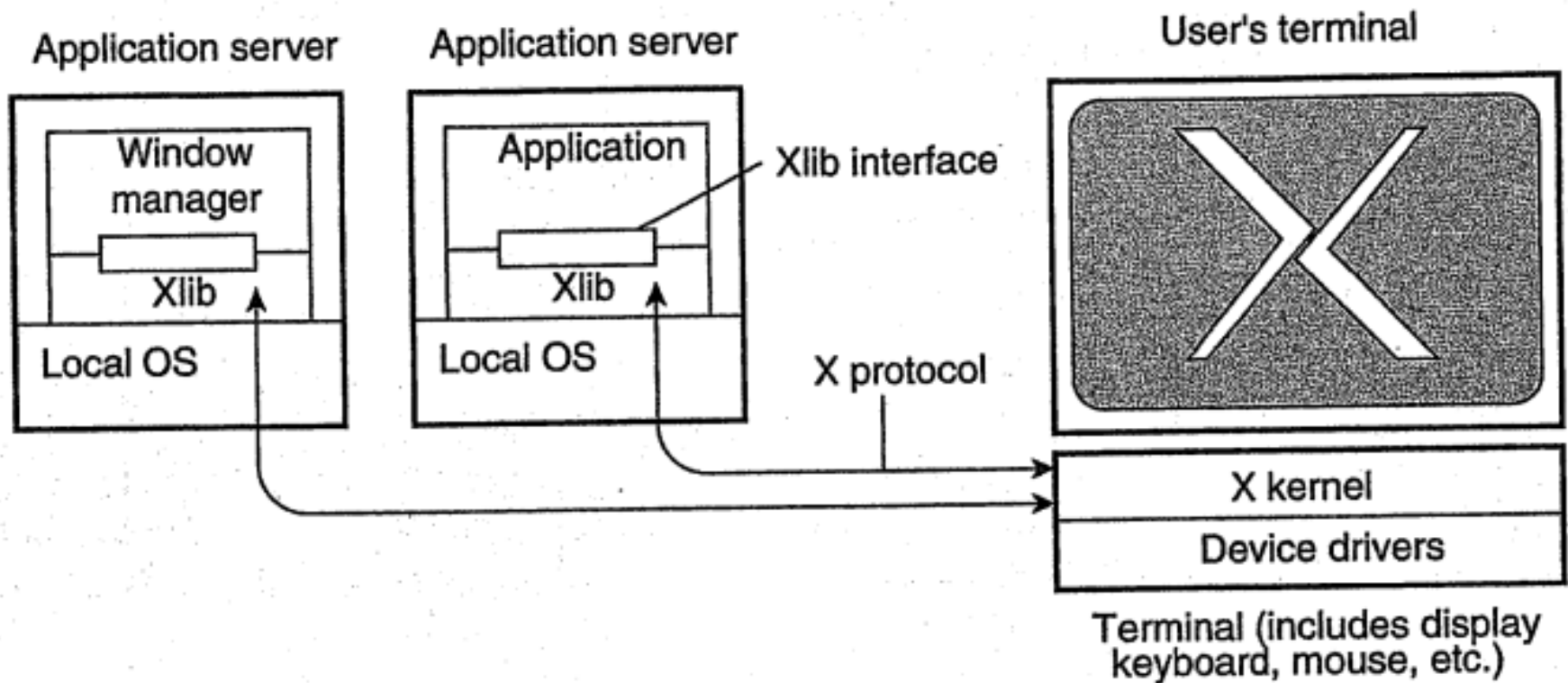
(a)



(b)

# Hệ thống X Window

43



# Thin-client Network Computing

44

- Phân biệt 2 khái niệm: X-client và X-server
- Các ứng dụng điều khiển màn hình bằng các lệnh chuyên dụng (cung ứng bởi X).
- Tách biệt về logic ứng dụng và các câu lệnh giao tiếp người dùng => Không thực hiện được
- Giải pháp: Nén thông điệp

# VD: 1 chương trình X-client sử dụng Xlib

45

[illegible]

# VD: 1 chương trình X-client sử dụng Xlib

46

```
// We want to get MapNotify events
XSelectInput(dpy, w, StructureNotifyMask);

// "Map" the window (that is, make it appear on the screen)
XMapWindow(dpy, w);

// Create a "Graphics Context"
GC gc = XCreateGC(dpy, w, 0, NIL);

// Tell the GC we draw using the white color
XSetForeground(dpy, gc, whiteColor);

// Wait for the MapNotify event
for(;;) {
    XEvent e;
    XNextEvent(dpy, &e);
    if (e.type == MapNotify)
        break;
}
```

# VD: 1 chương trình X-client sử dụng Xlib

47

```
// Draw the line
```

```
XDrawLine(dpy, w, gc, 10, 60, 180, 20);
```

```
// Send the "DrawLine" request to the server
```

```
XFlush(dpy);
```

```
// Wait for 10 seconds
```

```
sleep(10);
```

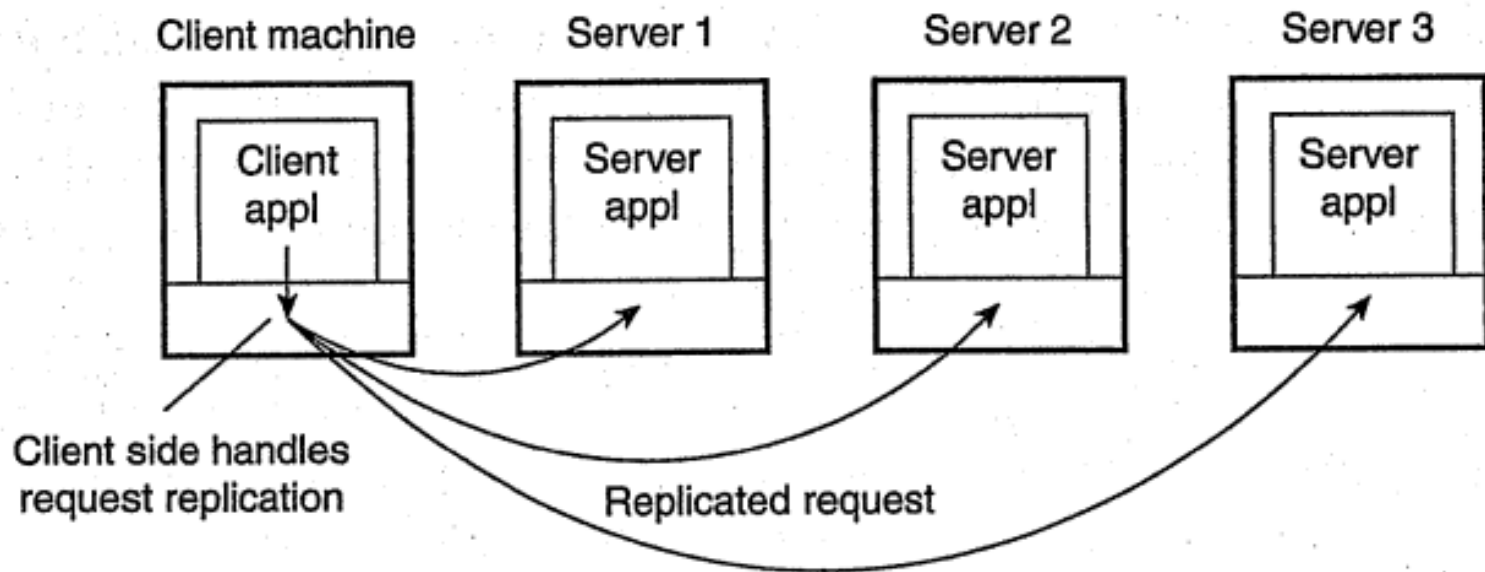
```
}
```

## 3.2. Phần mềm client phục vụ trong suốt phân tán

48

### ❖ Trong suốt phân tán:

- ❖ Trong suốt truy cập
- ❖ Trong suốt di trú
- ❖ Trong suốt sao lưu
- ❖ Trong suốt che giấu lỗi





## 4. Servers

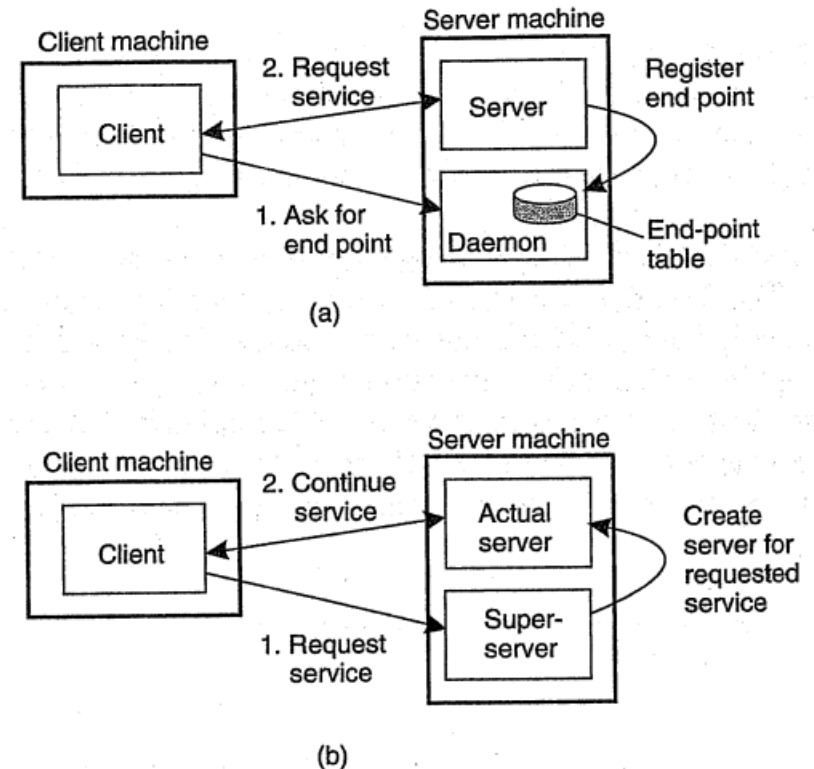
Các vấn đề thiết kế server

Server clusters

# 4.1. Các vấn đề thiết kế server

50

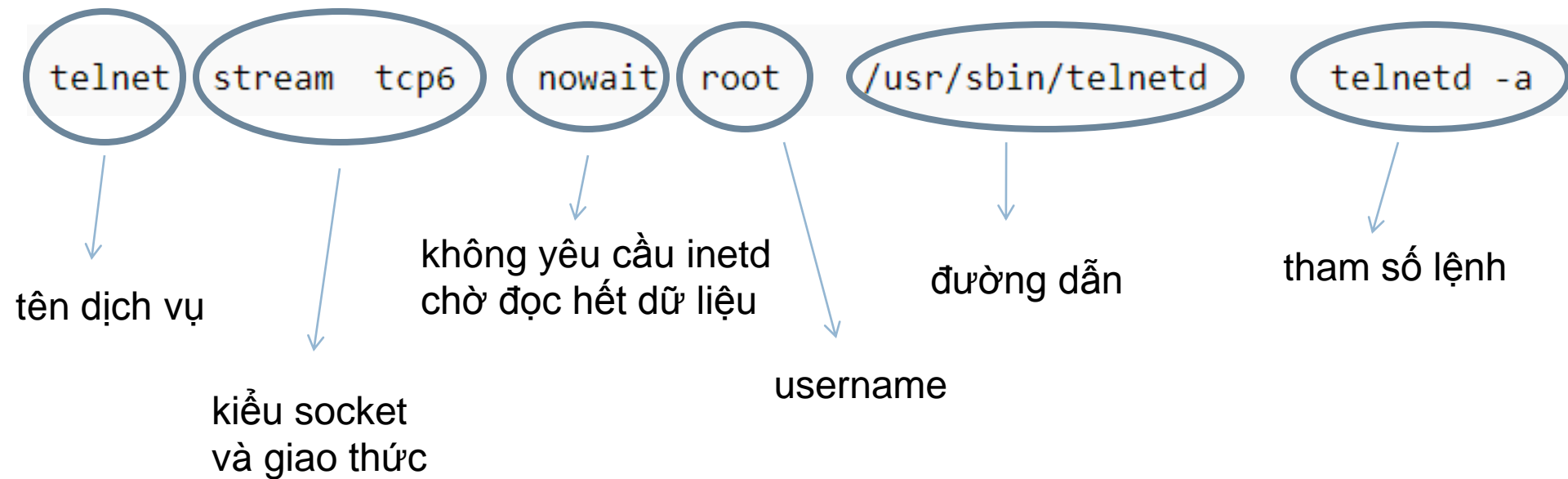
- Tổ chức server
  - ▣ Server lắp
  - ▣ Server đồng thời
- Vấn đề xác định server:
  - ▣ End-point (port)
  - ▣ Deamon
  - ▣ Superserver
- Vấn đề ngắt server
- Stateless & stateful server



# Inetd

51

- Cấu hình của Inetd được lưu trong */etc/inetd.conf*



# Xây dựng dịch vụ cho inetd

52

- Viết 1 chương trình errorLogger.c

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv)
{
    const char *fn = argv[1];
    FILE *fp = fopen(fn, "a+");

    if(fp == NULL)
        exit(EXIT_FAILURE);

    char str[4096];
    //inetd passes its information to us in stdin.
    while(fgets(str, sizeof(str), stdin)) {
        fputs(str, fp);
        fflush(fp);
    }
    fclose(fp);
    return 0;
}
```

# Xây dựng dịch vụ cho inetd

53

- ❑ Điền thêm vào /etc/services

```
errorLogger 9999/udp
```

- ❑ Điền thêm vào /etc/inetd.conf

```
errorLogger dgram udp wait root  
/usr/local/bin/errlogd errlogd  
/tmp/logfile.txt
```

## 5. Di trú mã

Vì sao phải di trú mã

Các mô hình di trú mã

Di trú mã và tài nguyên cục bộ

Di trú mã trong các hệ thống không đồng nhất

# Vì sao phải di trú mã

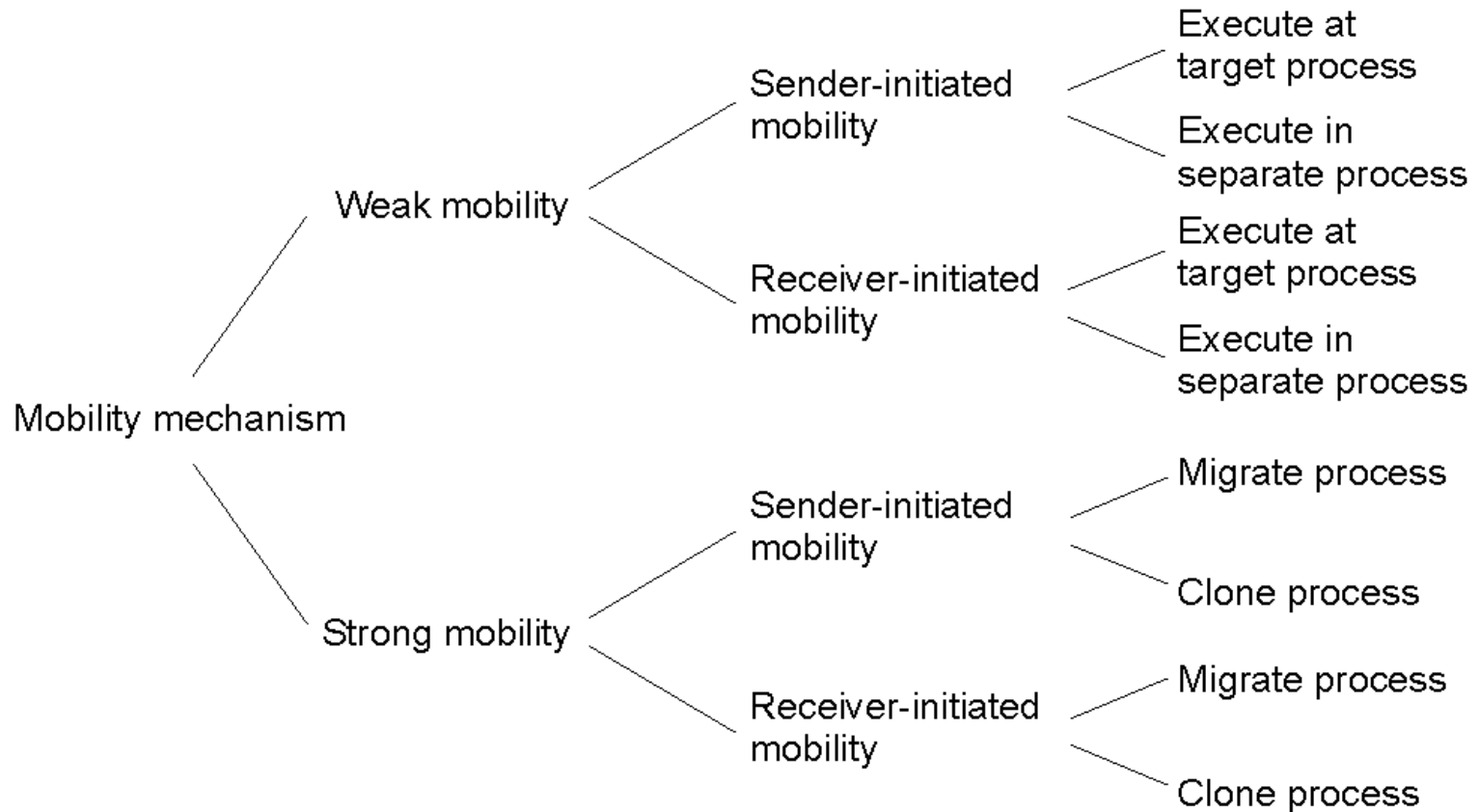
55

- Tăng hiệu năng
  - ▣ Mã server cho client
  - ▣ Mã client cho server
  - ▣ Thực hiện song song một code trên nhiều máy
- Tính mềm dẻo
  - ▣ Tải stub động
  - ▣ Cấu hình động hệ thống

# Mô hình di trú mã

56

## □ Alternatives for code migration.





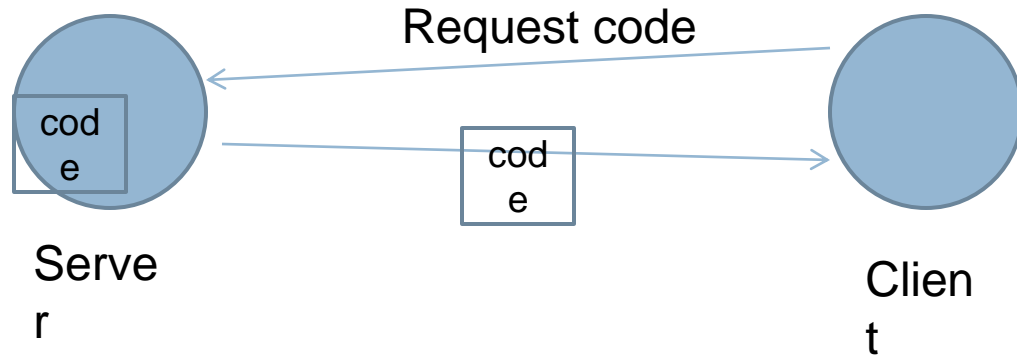
# Mobile Agent

60

- ĐN: những thành phần phần mềm (*mã chương trình, dữ liệu và trạng thái hoạt động*) có thể tự di chuyển từ nơi này sang nơi khác.
- Kích thước nhỏ
- Di trú
- Liên lạc, nhân bản, nhập lại, tổng hợp tính toán
- Khả năng xác định và dùng những tài nguyên trên các máy tính đang chứa nó
- Một số hệ thống Mobile Agent: Aglets, Voyager, Mole, Zeus

# Pull code, Push code & Autonomous code

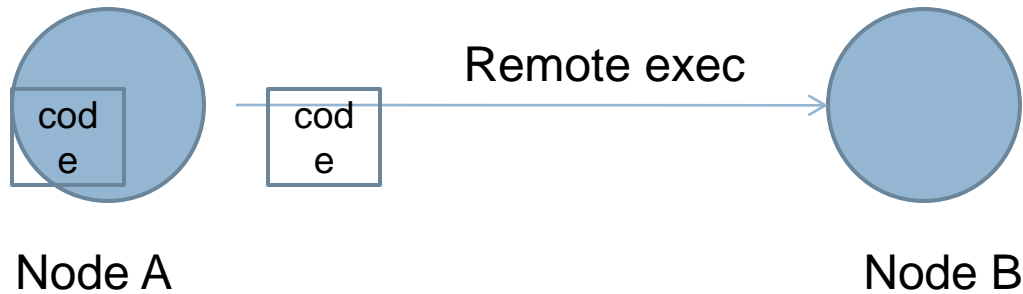
61



**Pull code**

# Pull code, Push code & Autonomous code (2)

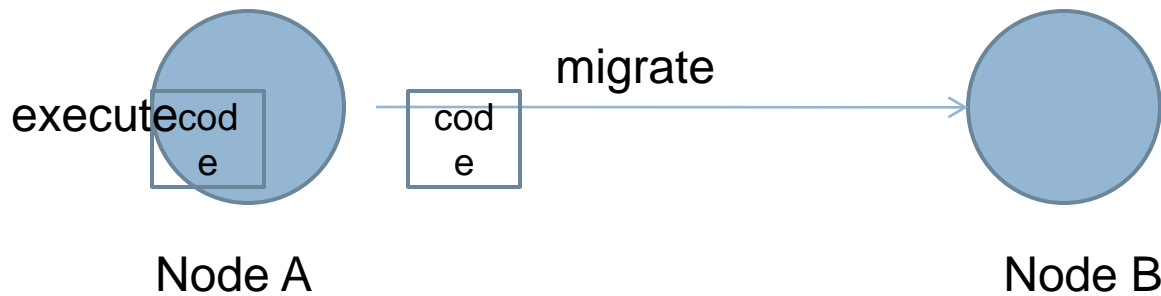
62



**Push code**

# Pull code, Push code & Autonomous code (3)

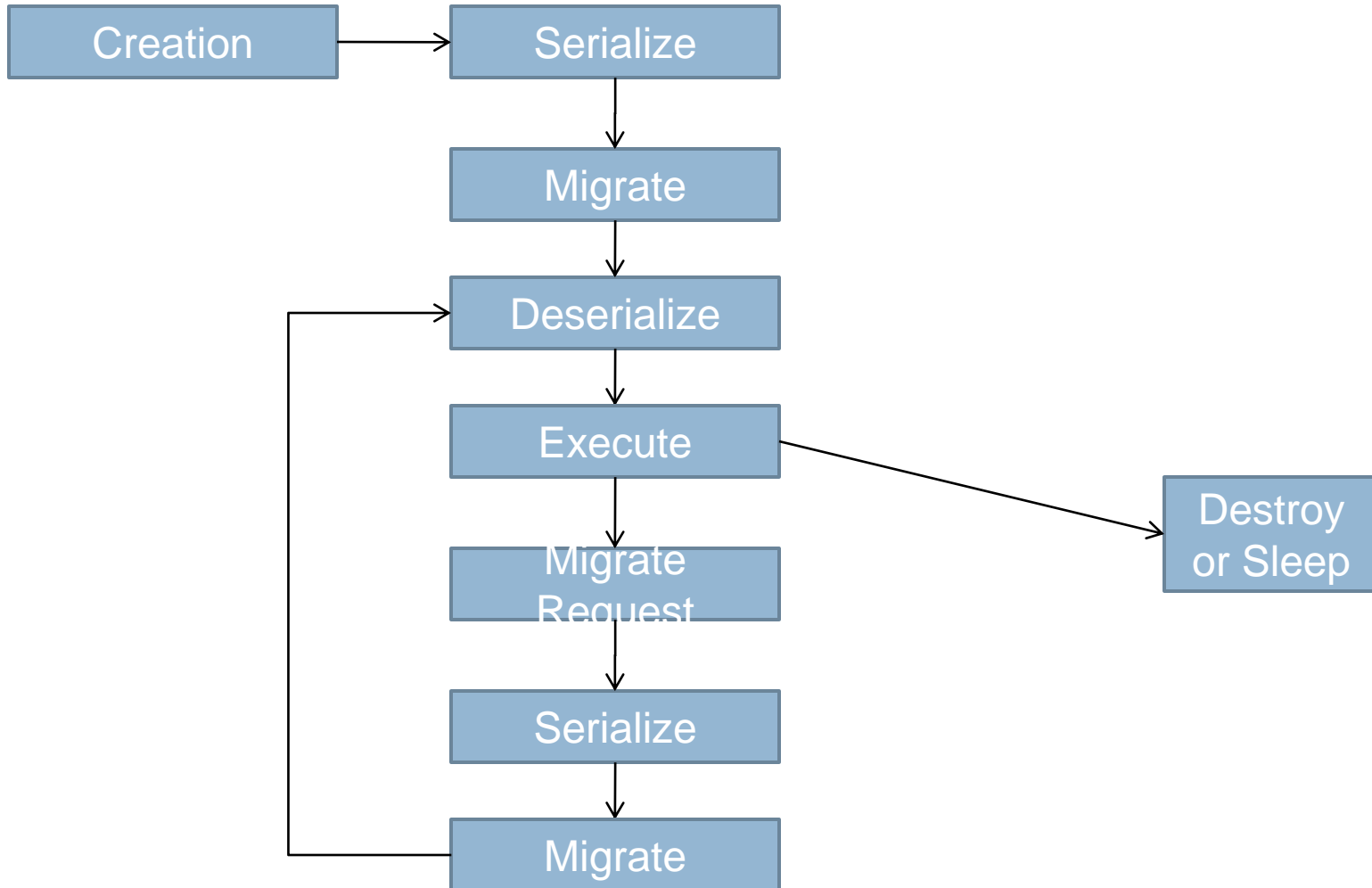
63



**Autonomous  
code**

# Lifecycle

64



# Agent host

65

- AgentOS: tạo ra agent từ đoạn code đã có, thực thi agent, chuyển agent đến host khác, huỷ agent.
- Độc lập platform
- Multithreaded
- Di trú được agent và trạng thái của nó
- Cơ chế cho các agent trong host giao tiếp được
- Có thể mở rộng được

# Ưu điểm

66

- ❑ Giảm băng thông
- ❑ Giảm độ trễ
- ❑ Có thể thực thi khi ngắt kết nối mạng
- ❑ Thực thi bất đồng bộ và tự động
- ❑ Nhanh, giảm thiểu lỗi
- ❑ Khắc phục tình trạng không đồng nhất

# Môi trường ứng dụng

67

- ❑ Thu thập dữ liệu phân tán
- ❑ Theo dõi và thông báo tin cập nhật
- ❑ Giám sát và phân tán thông tin
- ❑ Xử lý song song
- ❑ Thương mại điện tử
- ❑ Quản trị hệ thống mạng
- ❑ Hỗ trợ các thiết bị di động