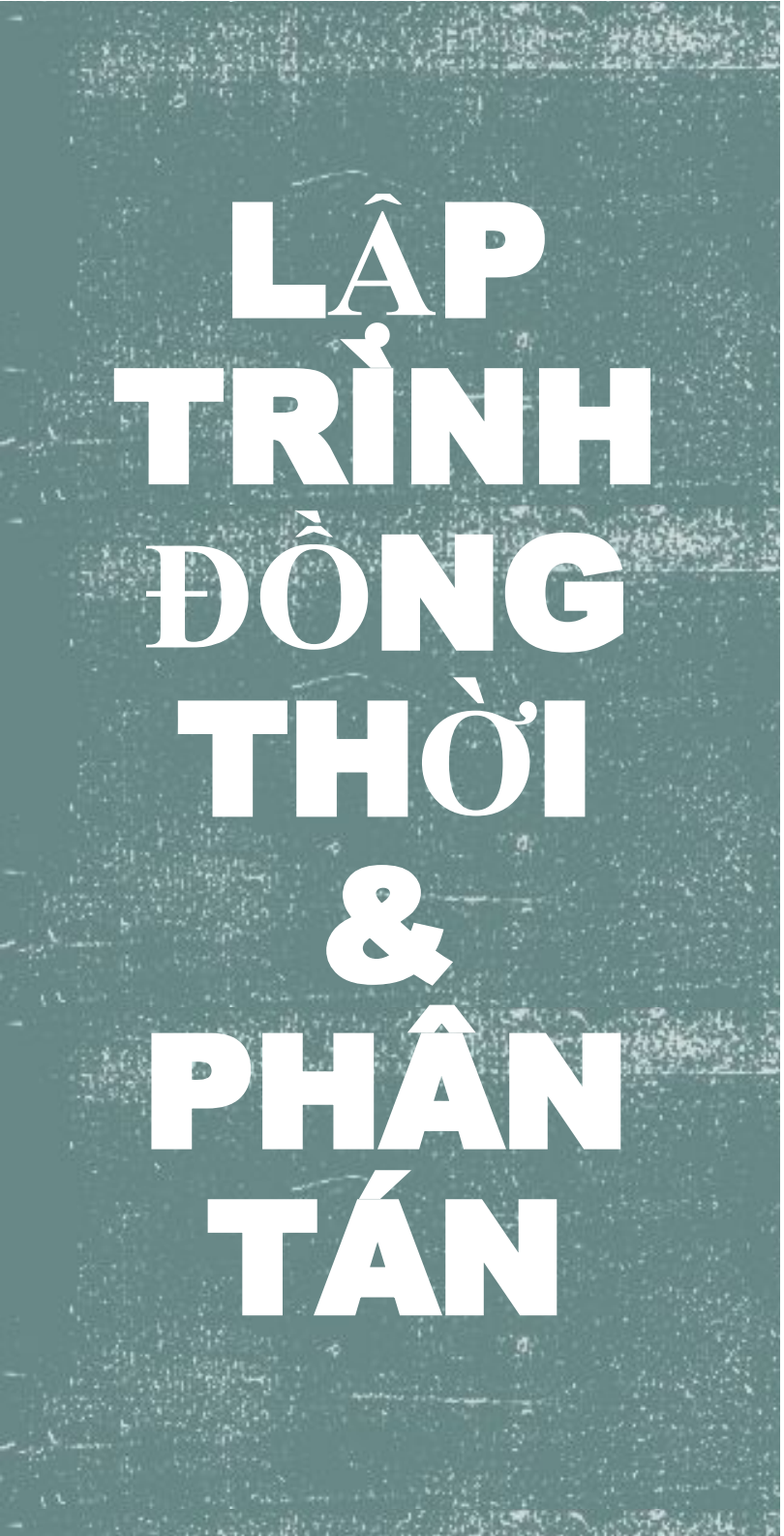
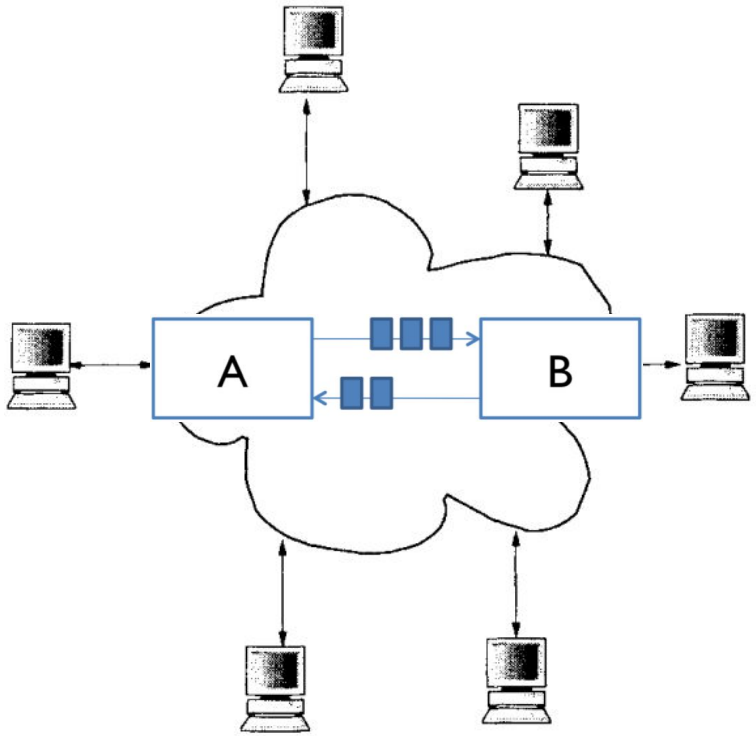


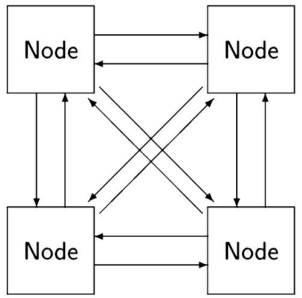
BÀI 4:
CÁC KỸ THUẬT LẬP TRÌNH PHÂN TÁN

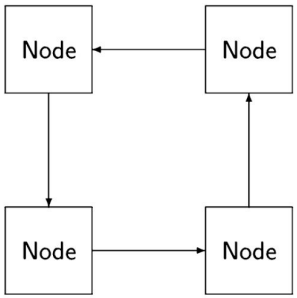
Giảng viên: Lê Nguyễn Tuấn Thành
Email: thanhlnt@tlu.edu.vn

Nhắc lại:
Hệ thống phân tán

- Bao gồm nhiều máy tính kết nối với nhau
  - Không có các biến chia sẻ
  - Trao đổi thông qua các kênh truyền thông để gửi và nhận thông điệp
- Thường sử dung lý thuyết đồ thị để minh hoạ những hệ thống phân tán







NỘI DUNG

- Socket: java networking classes
- RMI: remote method invocation
- Message oriented middleware

1. Các mô hình truyền thông điệp
2. Giao thức truyền thông điệp: UDP, TCP
3. Lập trình truyền thông điệp:
   - **Ở mức thấp**
     - **SOCKET**
     - MPI (Message Passing Interface)
   - **Ở mức cao:**
     - **RMI** (Remote Method Invocations)
     - MOM (Message-Oriented Middleware)

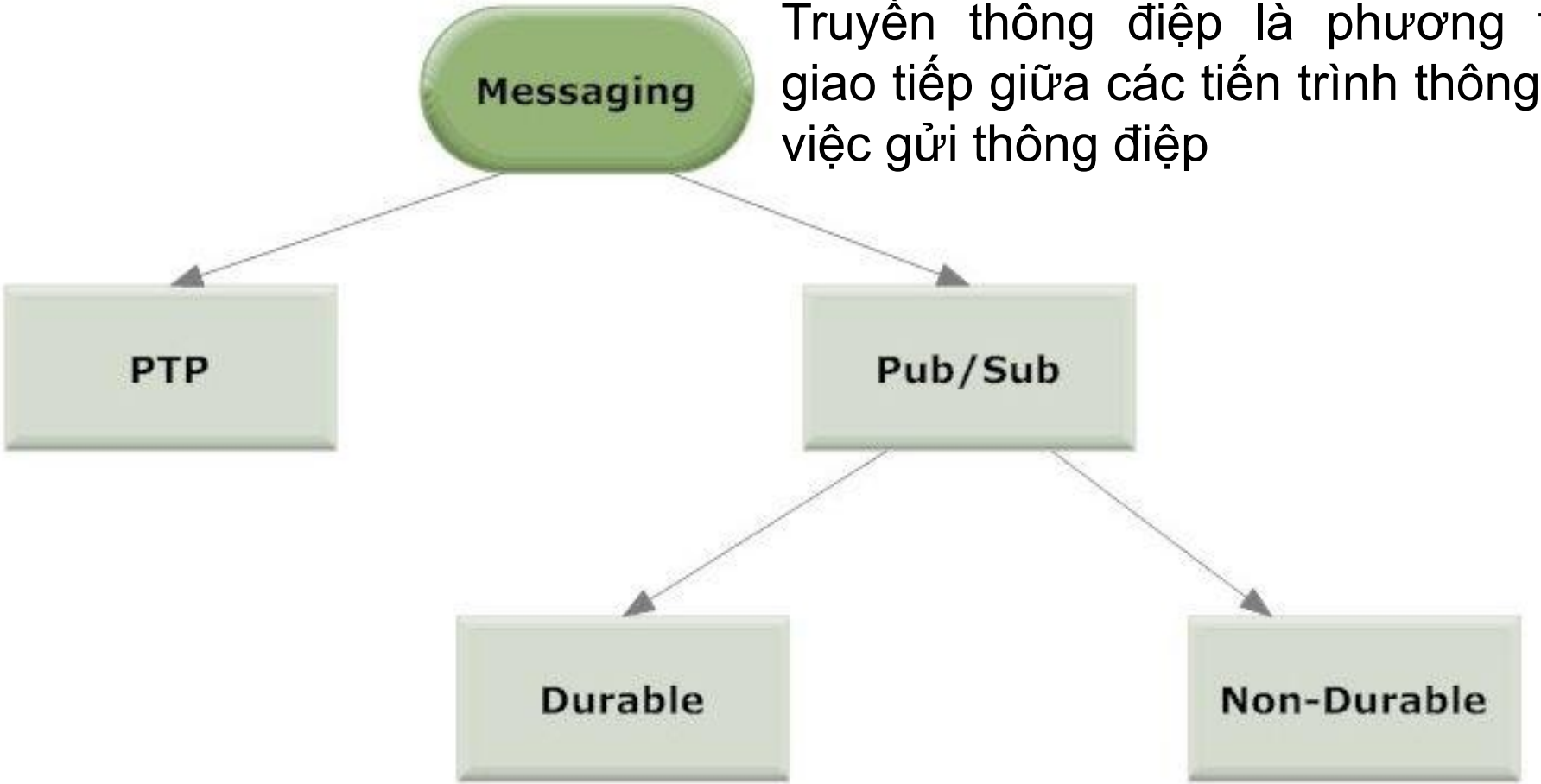
Bài giảng có sử dụng hình vẽ trong cuốn sách "***Concurrent and Distributed Computing in Java***, *Vijay K. Garg, University of Texas, John Wiley & Sons, 2005*"

4

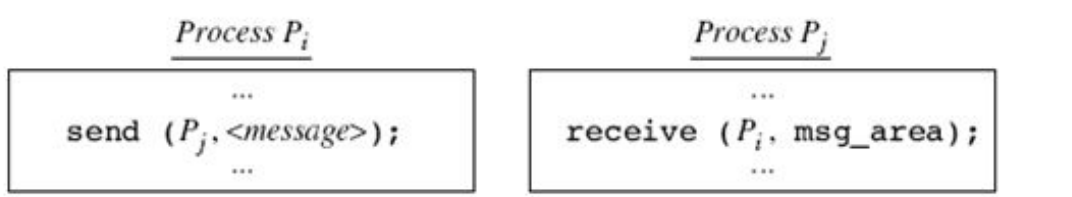
Phần 1. Các mô hình truyền thông điệp

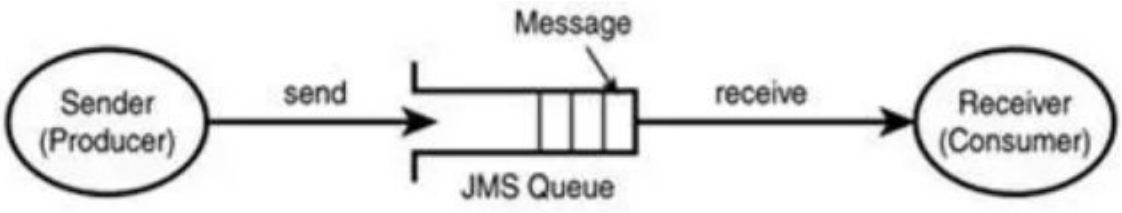
Các mô hình truyền thông điệp

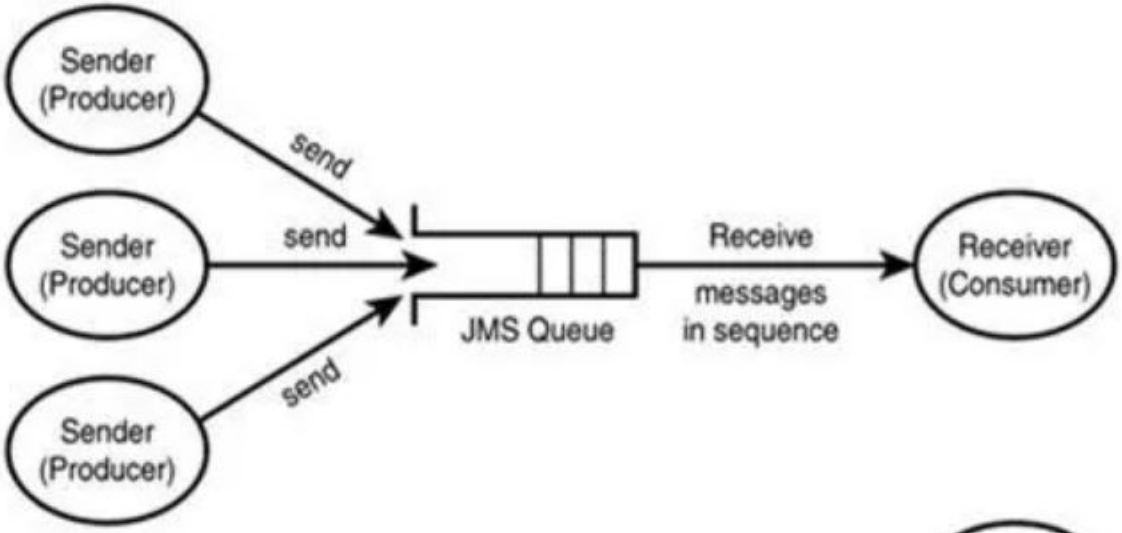
Truyền thông điệp là phương thức giao tiếp giữa các tiến trình thông qua việc gửi thông điệp

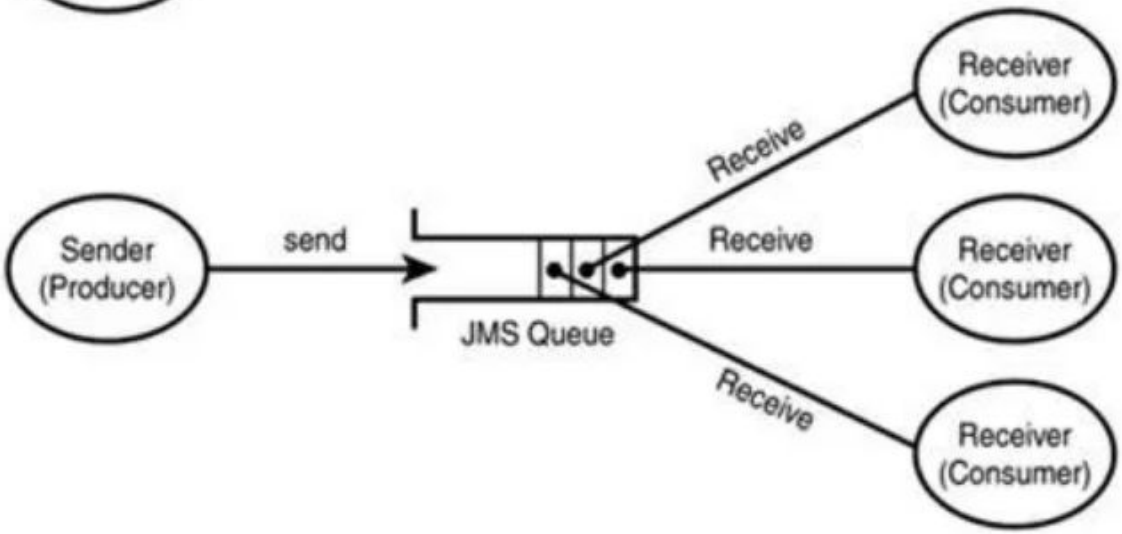


Mô hình Point-to-Point

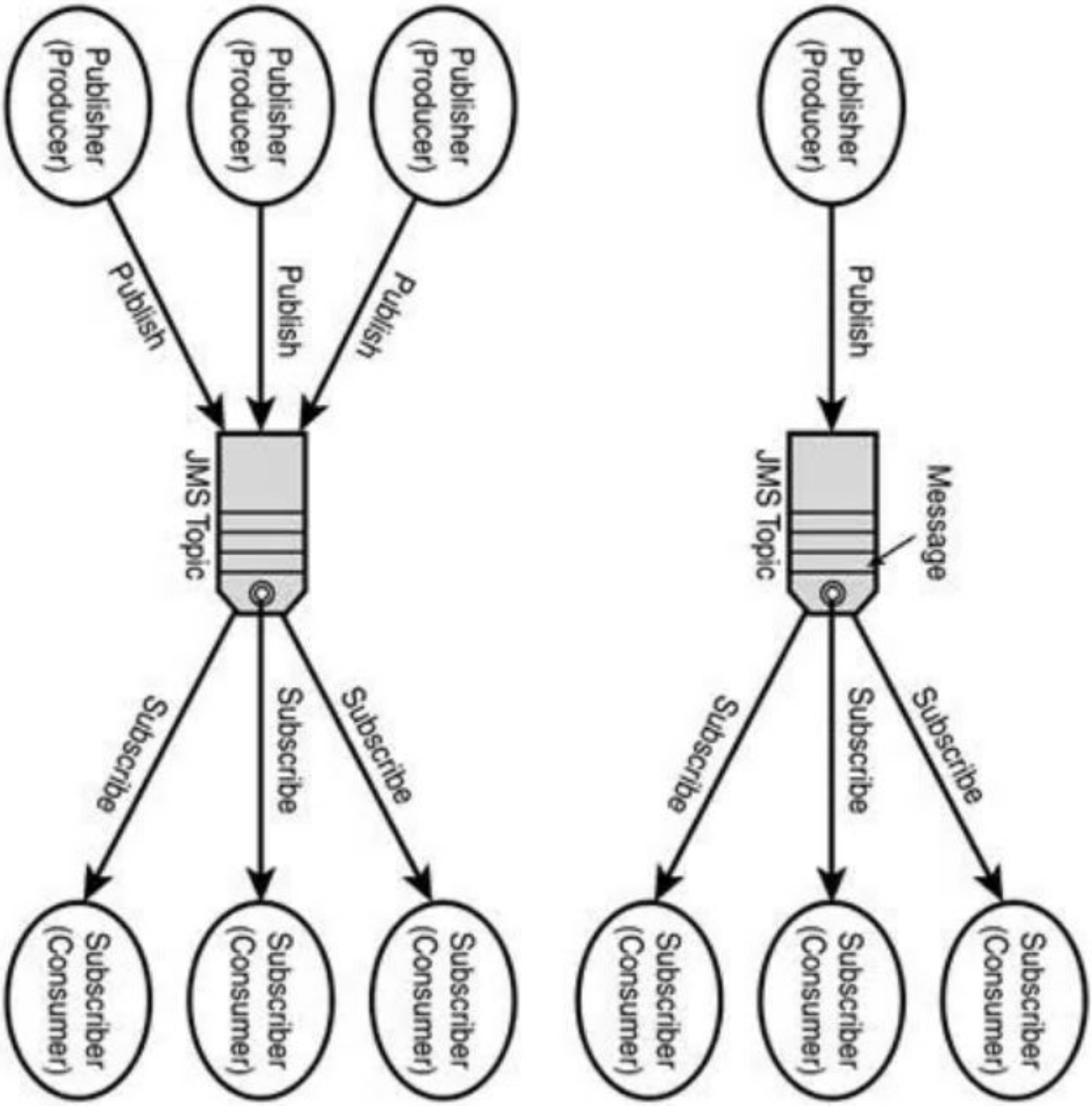








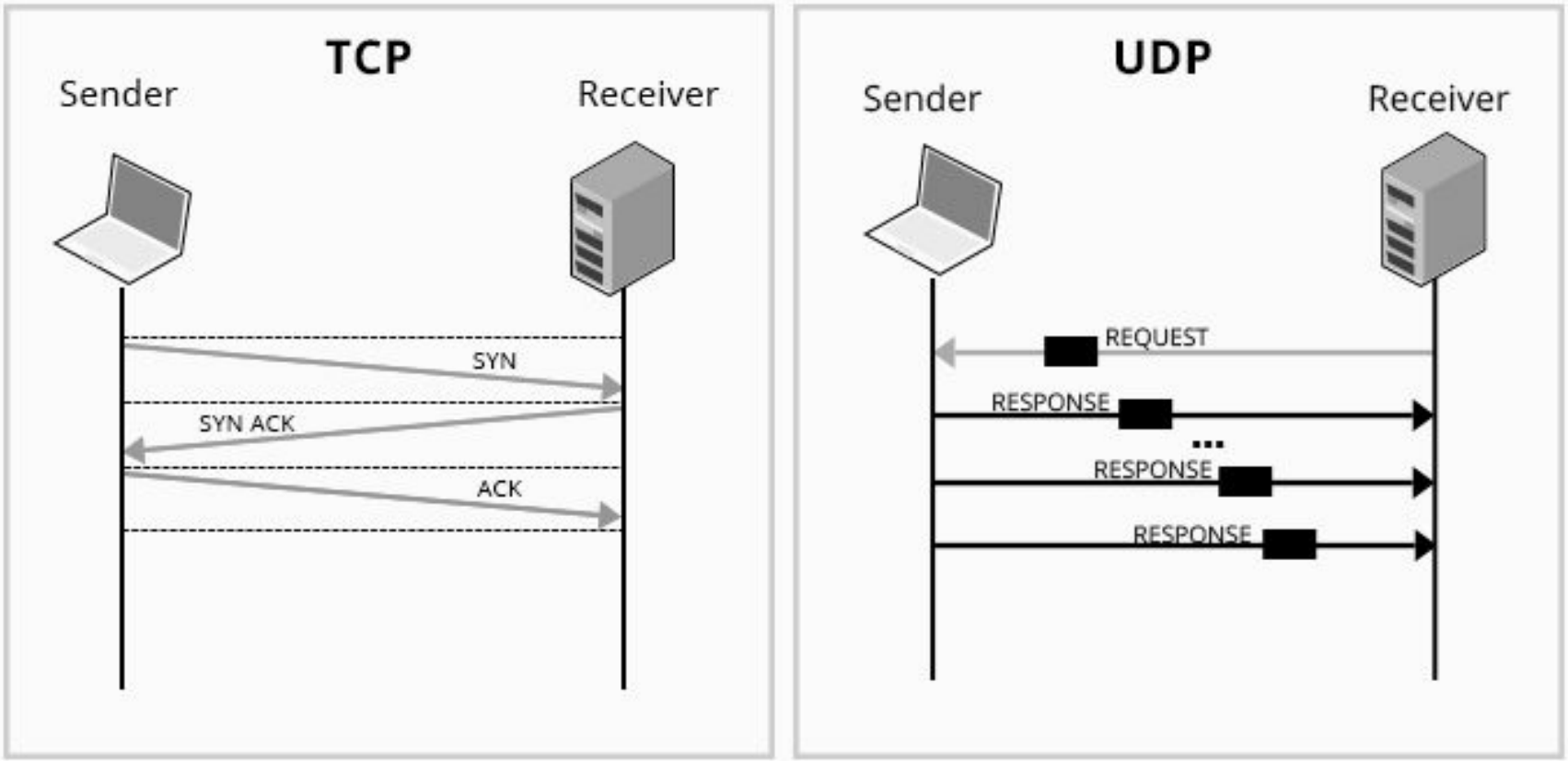
Mô hình Publish&Subscribe

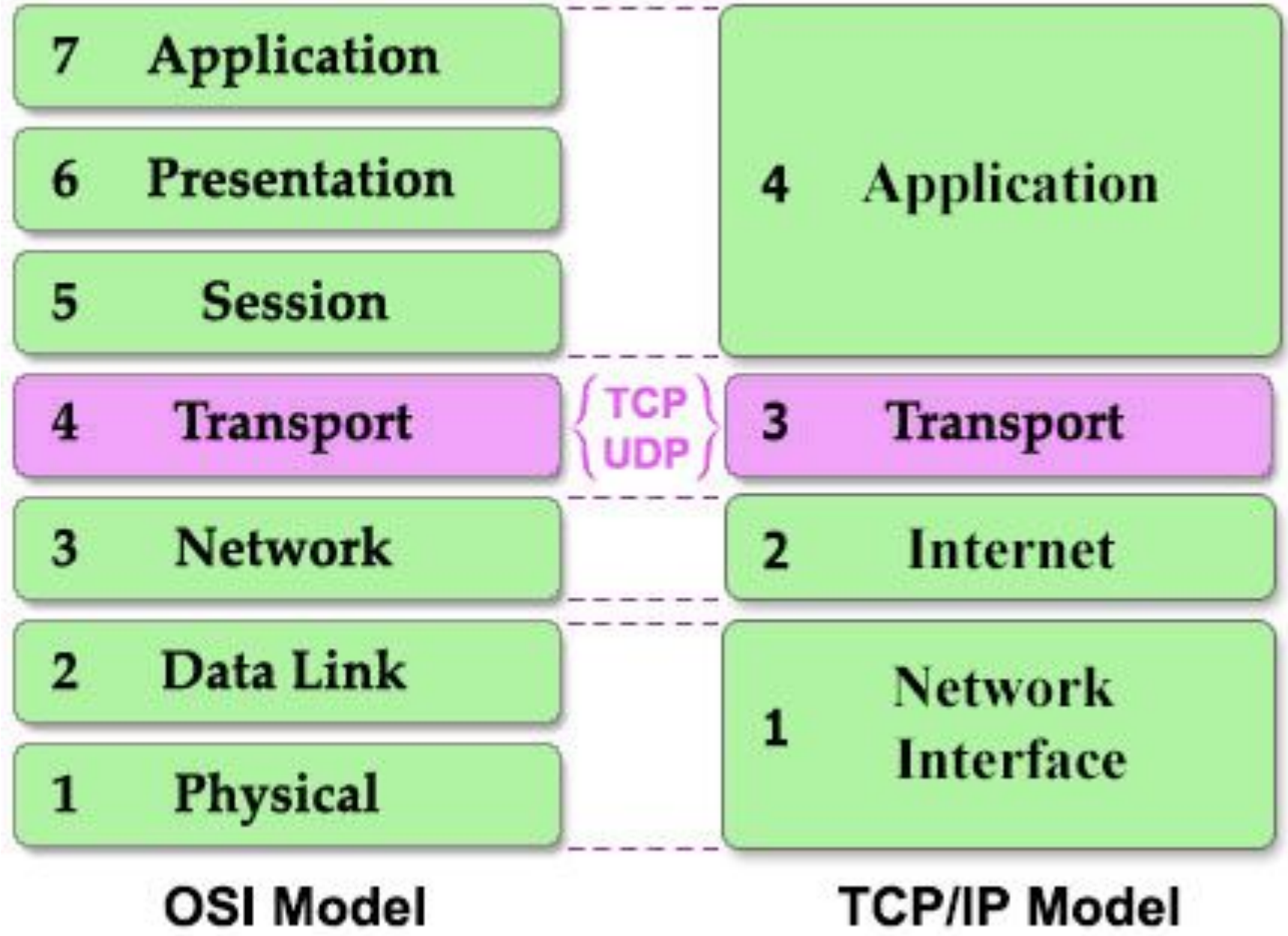


8

Phần 2. Giao thức truyền thông điệp

TCP Vs UDP Communication




7 Application
6 Presentation
5 Session
4 Transport
3 Network
2 Data Link
1 Physical
OSI Model
4 Application
TCP
UDP
3 Transport
2 Internet
1 Network Interface
TCP/IP Model

UDP vs. TCP

UDP

(Universal Datagram Protocol)

- Không kết nối
- Không truyền lại
- Không bảo đảm nhận được theo thứ tự đã gửi
- Có thể bị mất, bị trùng lặp

TCP

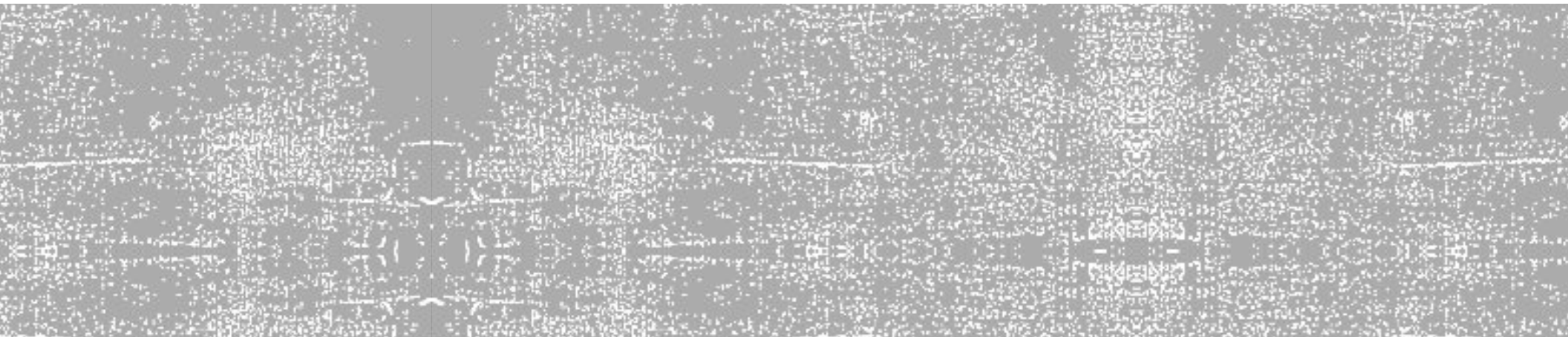
(Transmission Control Protocol)

- Hướng kết nối
- Tin cậy (nhận đủ, không trùng lặp)
- Đảm bảo thứ tự nhận được của các gói tin
- Không bị mất gói tin

Socket

12

Phần 3. Lập trình truyền thông điệp trên Java



Lớp InetAddress (1)

- Bất kỳ máy tính kết nối với Internet (i.e., *host)* có thể được xác định duy nhất bằng một *địa chỉ internet* (i.e., địa chỉ IP)
- Do địa chỉ số này khó nhớ, mỗi host cũng có thêm một tên (i.e., *hostname*) đi kèm
- Máy chủ phân giải hệ thống tên miền (i.e., DNS) giúp cung cấp ánh xạ từ *tên máy* đến *địa chỉ* của nó.

Lớp InetAddress (2)

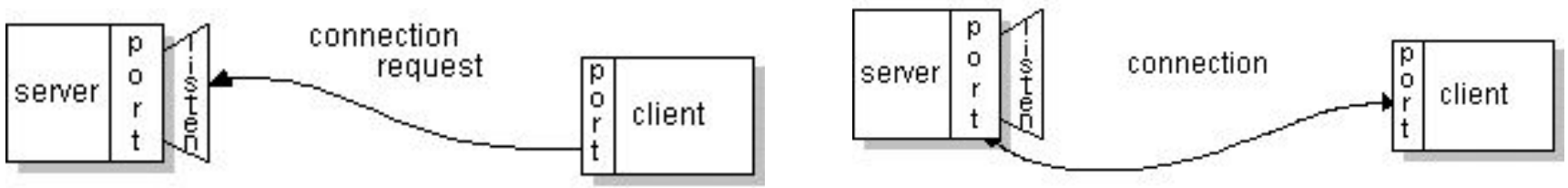
```
public byte[] getAddress()
  Returns the raw IP address of this InetAddress object.
public static InetAddress getByName(String)
  Determines the IP address of a host, given the host's name.
public String getHostAddress()
  Returns the IP address string "%d.%d.%d.%d"
public String getHostName()
  Returns the fully qualified host name for this address.
public static InetAddress getLocalHost()
  Returns the local host.
```

Lập trình Socket trên Java

3.1. Khái niệm Socket

Socket

- Socket là một điểm cuối của liên kết truyền thông hai chiều giữa hai chương trình đang chạy trên mạng.
- Một Socket được liên kết với một số hiệu cổng để có thể xác định ứng dụng đích mà dữ liệu sẽ được gửi đến.
  - Do đó, Socket là sự kết hợp của *địa chỉ IP* và *số hiệu cổng*
  - Ví dụ: *206.62.226.25,p21*
- Hai socket sẽ xác định một kết nối: *socket pair*



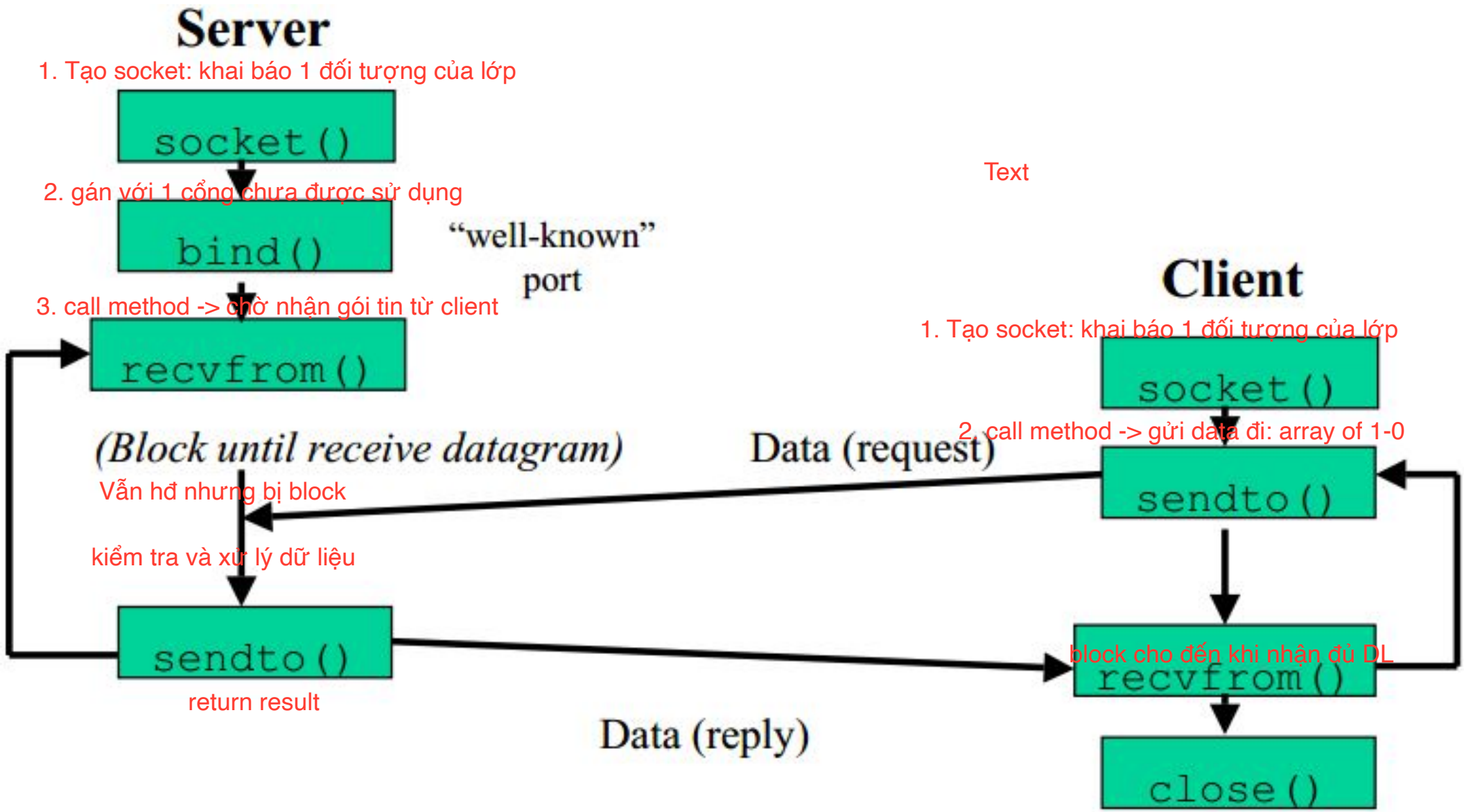
3.2. Lập trình Socket dựa trên UDP

Lớp DatagramSocket (1)

- Sử dụng giao thức UDP để gửi và nhận thông điệp
  - Ưu điểm: cho phép truyền dữ liệu nhanh
- Một *datagram socket* là điểm gửi hoặc nhận cho một dịch vụ vận chuyển gói tin không kết nối
  - Mỗi gói tin gửi hoặc nhận trên datagram socket được định địa chỉ và định tuyến một cách riêng biệt

```
public DatagramSocket()
public DatagramSocket(int port)
public DatagramSocket(int port, InetAddress laddr)
```

Hoạt động Socket UDP



Source: https://slideplayer.com/slide/4969080/

Lớp DatagramSocket (2)

- public void close()
- public int getLocalPort()
- public InetAddress getLocalAddress()
- **public void receive(DatagramPacket p)**
  - Nhận một gói datagram từ socket
  - Khi trả về từ phương thức này, bộ đệm của p được điền đầy dữ liệu nhận được
  - Phương thức chặn cho đến khi nhận được một datagram
- **public void send(DatagramPacket p)**
  - Gửi một gói datagram từ socket
  - DatagramPacket bao gồm các thông tin về dữ liệu, chiều dài của dữ liệu, địa chỉ IP & cổng của máy đích

Chương trình mẫu: Echo server

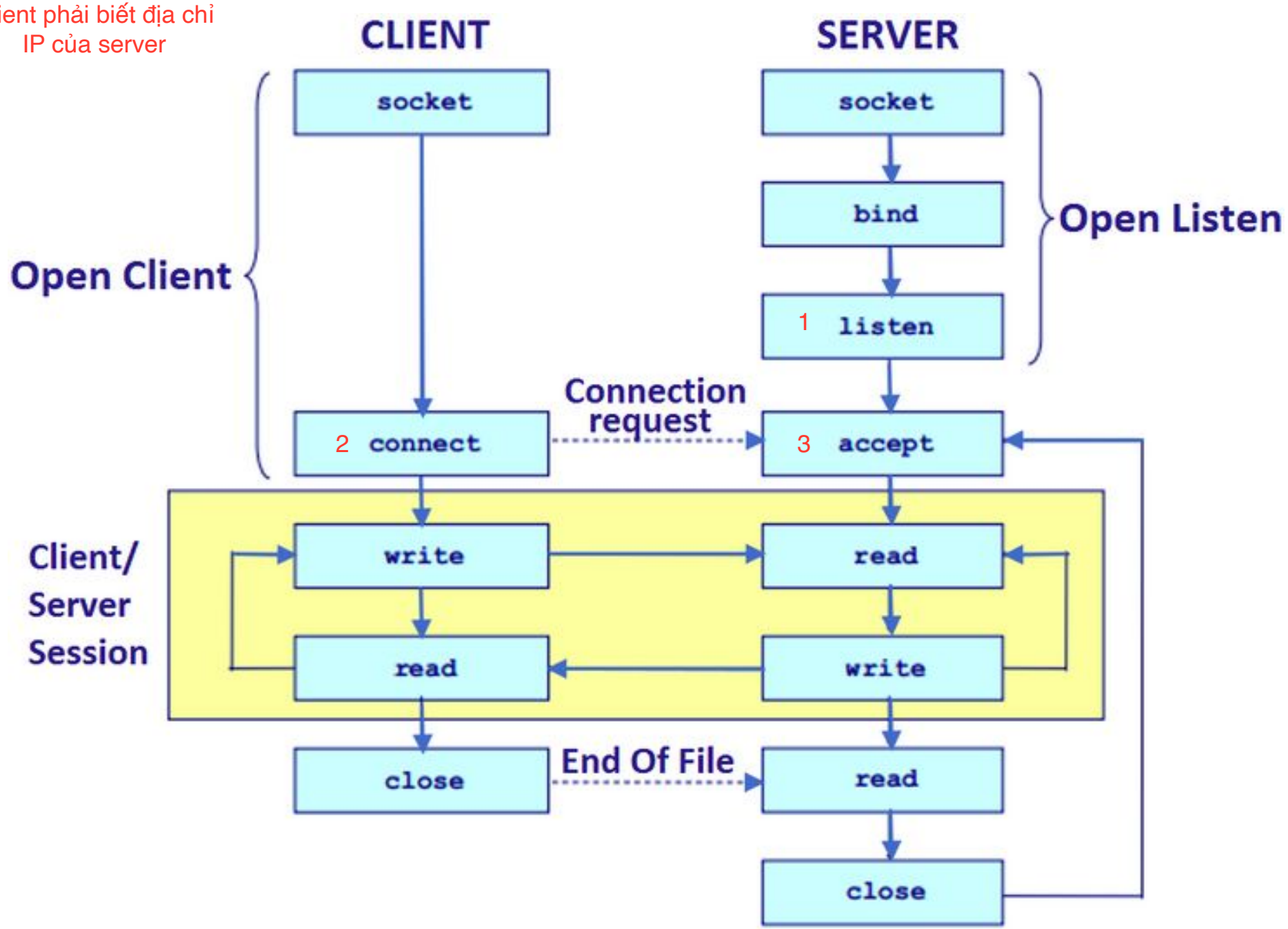
- Hệ thống gồm 2 tiến trình cho:
  - Server
  - Client
- Client đọc dữ liệu đầu vào từ người dùng và gửi nó đến server
- Server nhận được gói tin datagram và sau đó gửi trả lại chính dữ liệu đó cho client
- Client đọc một dòng đầu vào bàn phím
  - Sau đó, client tạo một gói tin datagram và gửi tới server
  - Khi nhận được phản hồi từ server, client sẽ hiển thị thông báo nhận được

3.3. Lập trình Socket dựa trên TCP

Lập trình Socket dựa trên TCP

- Dựa trên khái niệm về *dòng tin* (stream)
- Một kết nối TCP được thiết lập giữa tiến trình gửi và tiến trình nhận
- Cho phép phục hồi lỗi tốt hơn và đảm bảo thứ tự phân phối các gói dữ liệu.
  - Trong một stream, các gói được nhận theo đúng thứ tự như khi chúng được gửi đi.
- Sử dụng lớp Socket cho phía client và ServerSocket cho phía server

Client phải biết địa chỉ IP của server



Source: https://www.safaribooksonline.com/library/view/distributed-computing-in/9781787126992/02dd04be-0dbb-4732-8bc5-1961644e8875.xhtml

Lớp Socket phía client

- **public Socket (String host, int port)**
  - Tạo một stream socket và kết nối nó tới một cổng trên host.
  - Có thể ném ra ngoại lệ **UnknownHostException** và **IOException.**
- **public Socket(InetAddress address, int port)**
- Một số phương thức của lớp Socket:
  - **public InetAddress getInetAddress()**
  - **public InetAddress getLocalAddress()**
  - **public int getport()**
  - **public Inputstream getInputStream()**
  - **public Outputstream getOutputStream()**
  - **public synchronized void close()**

Lớp ServerSocket

- **public ServerSocket(int port)**
- Một số phương thức của lớp *ServerSocket*:
  - public InetAddress getInetAddress()
  - public int getLocalPort()
  - public Socket accept()
  - public void close()

Chương trình mẫu: NameServer

- Một bảng phân giải tên <*name: hostName, portNumber*>
  - Giúp một ánh xạ từ tên chương trình tới địa chỉ và cổng mà tại đó chương trình đó đang chạy
- Giả định rằng kích thước tối đa của bảng phân giải tên là 100
- Hai phương thức:
  - **insert**
  - **search**

3.4. Lập trình RMI

Remote Method Invocations

Khái niệm RMI (1)

- **Ý tưởng chính**: cho phép một luồng T_1 chạy trên máy ảo VM_1 có thể thực hiện lời gọi đến phương thức M_2 của một đối tượng O_2 chạy trên máy ảo VM_2 khác giống như gọi phương thức M_1 của đối tượng O_1 nằm trên cùng một máy ảo với T_1
  - Đối tượng O_2 nằm ở máy ảo VM_2 khác với luồng T_1 được gọi là ***đối tượng ở xa***
  - Luồng tạo lời gọi (T_1) được gọi là *client*
  - Luồng phục vụ yêu cầu (T_2) được gọi là *server*
- Trong RMI, *client* có thể không cần biết về vị trí của đối tượng ở xa !

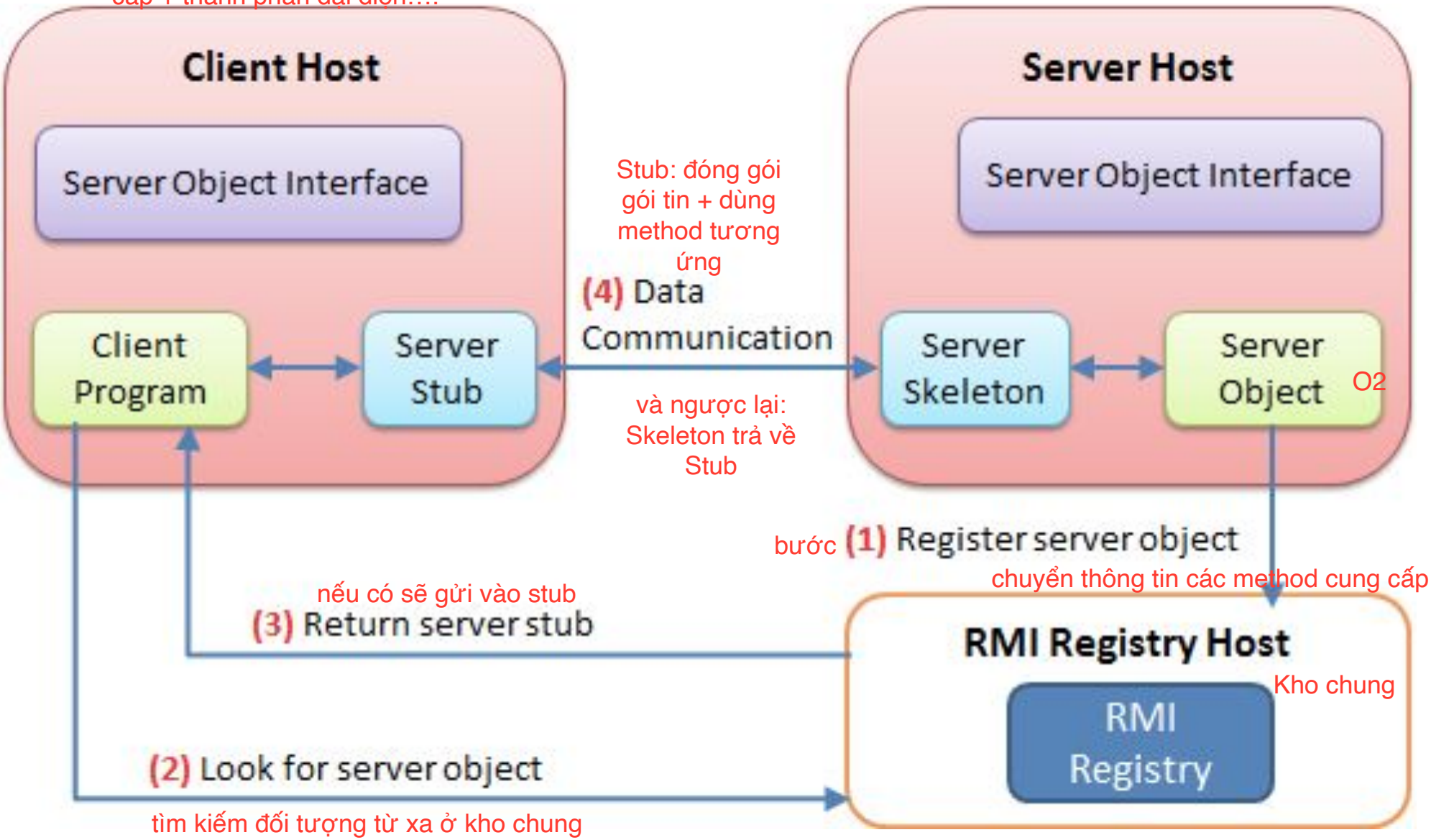
Khái niệm RMI (2)

- Các đối số của phương thức M_2, khi đối tượng ở xa O_2, được gửi thông qua thông điệp
- Tương tự, giá trị trả về của phương thức M_2 được truyền đến nơi gọi thông qua thông điệp
- Tất cả các thông điệp này được ẩn đi đối với lập trình viên
  - Do đó RMI có thể được xem như là một cấu trúc lập trình bậc cao hơn so với gửi hoặc nhận thông điệp bằng socket

Cài đặt RMI

- Với mỗi đối tượng ở xa sẽ phát sinh thêm:
  - một đối tượng liên kết ở phía *client* và
  - một đối tượng ở phía *server*
- Lời gọi tới đối tượng ở xa ở phía client được quản lý bằng cách sử dụng *đối tượng đại diện* cho client, được gọi là *stub*
  - Stub sẽ đóng gói *tên phương thức* và *các đối số* trong một thông điệp và truyền tới phía server
- Thông điệp này được nhận ở phía server bởi đối tượng đại diện, gọi là *skeleton*
  - Skeleton có trách nhiệm tiếp nhận thông điệp, lấy ra các đối số, và cuối cùng gọi phương thức thích hợp ở phía server

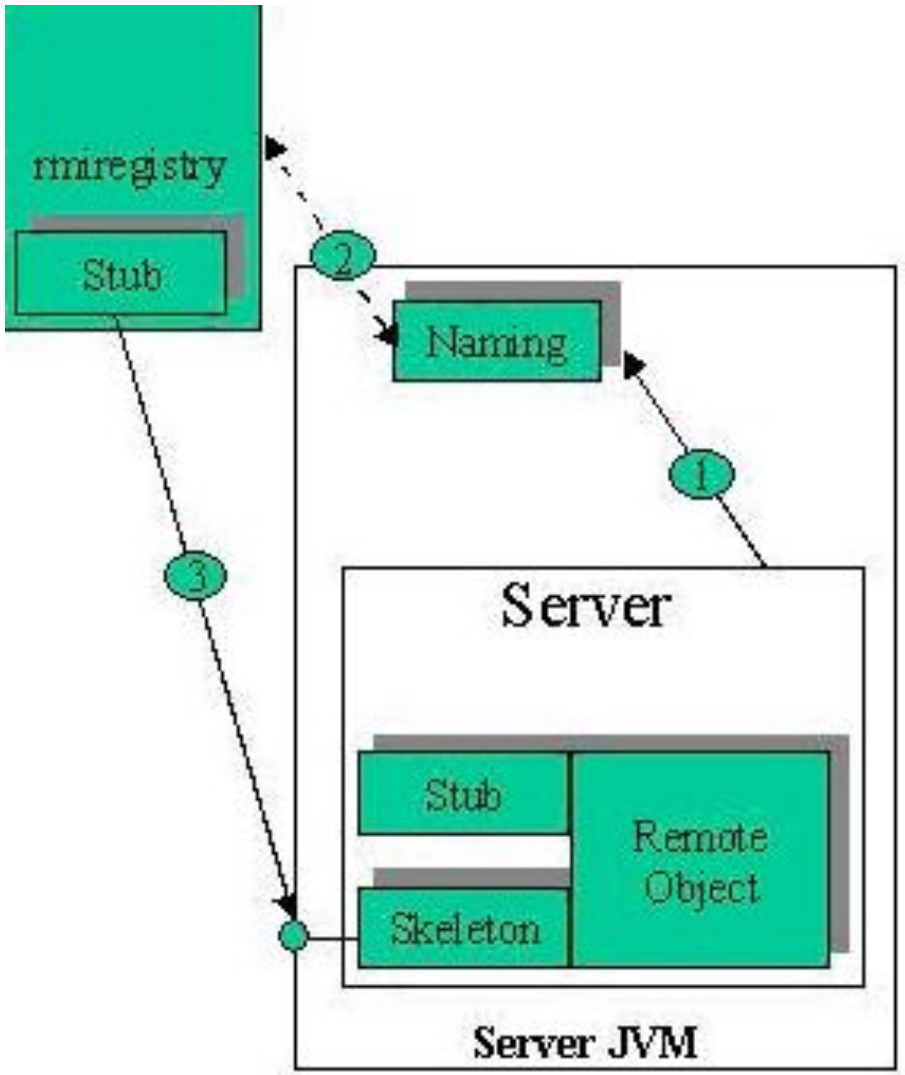
có được các đối tượng mà pthuc cung cấp + thành phần đại diện….



Source: http://www.rizzimichele.it/remote-methode-invocation/

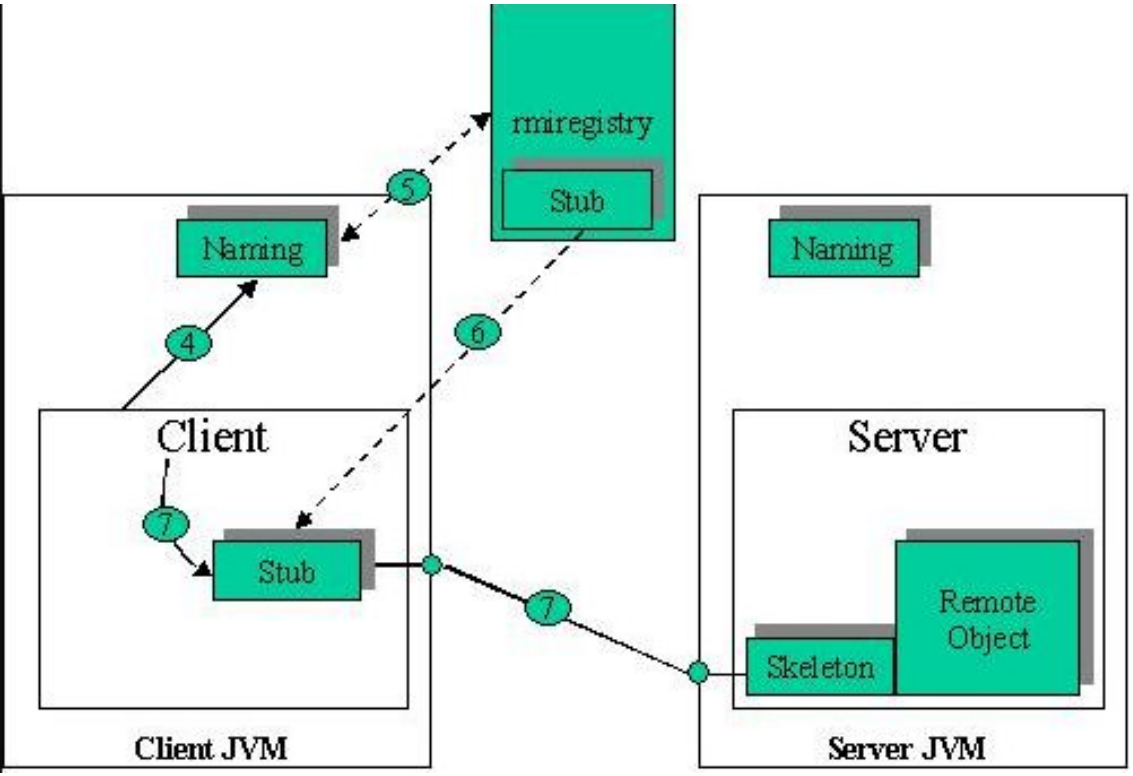
=> Không cần biết địa chỉ IP của server, chỉ cần Stub bên Client biết

Cơ chế vận hành theo RMI (1)



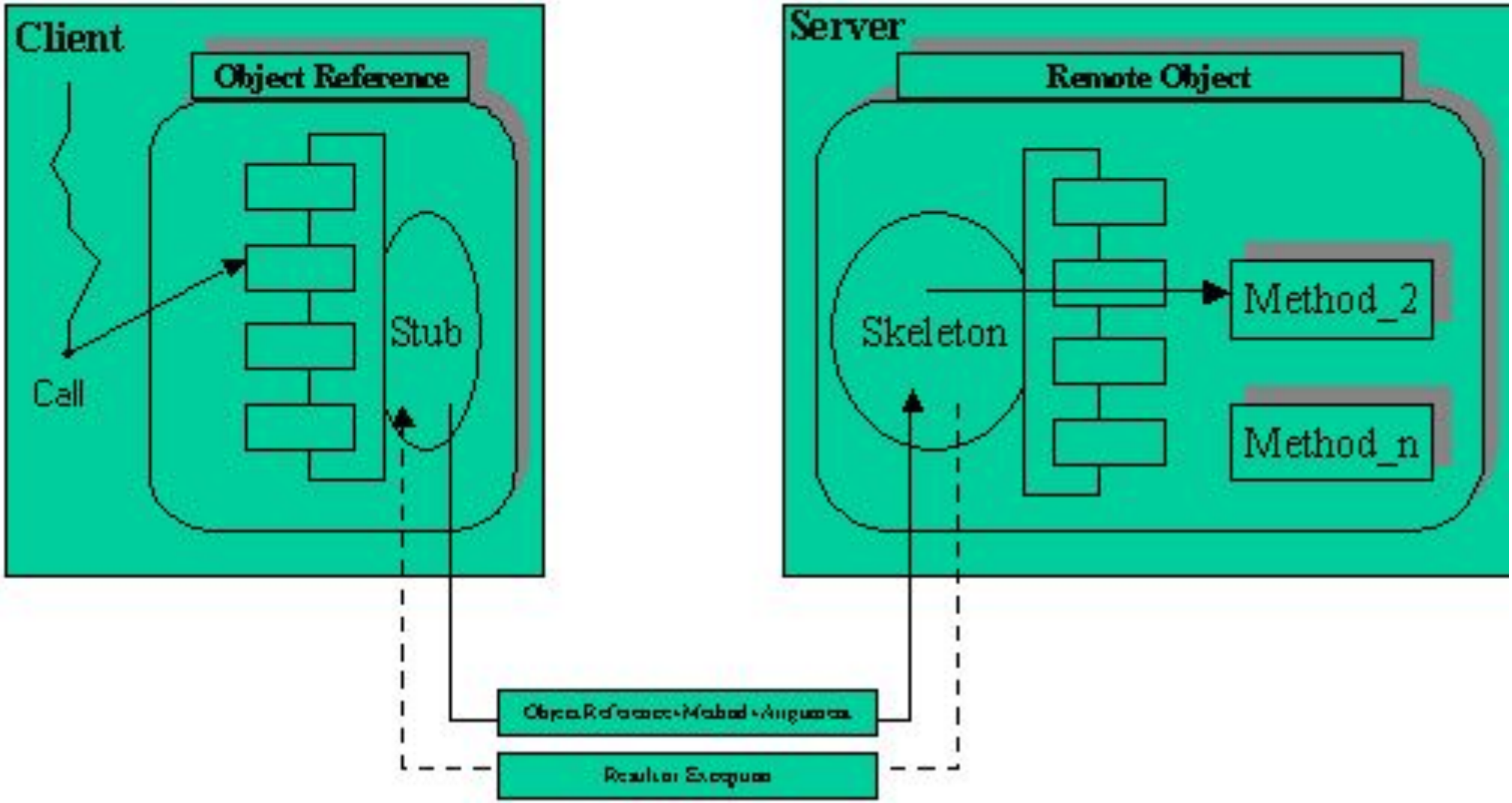
- **Bước 1**: Server tạo ra đối tượng cho phép được gọi từ xa cùng với các Stub và Skeleton tương ứng của đối tượng đó
- **Bước 2**: Server sử dụng lớp Naming để đăng ký tên cho đối tượng từ xa (1)
- **Bước 3**: Naming đăng ký Stub của đối tượng từ xa với RMIRegistry (2)
- **Bước 4**: RMIRegistry sẵn sàng cung cấp tham chiếu đến đối tượng từ xa khi có yêu cầu (3)

Cơ chế vận hành theo RMI (2)



- **Bước 5**: Client yêu cầu Naming định vị đối tượng ở xa qua tên đã được đăng ký (phương thức lookup) với RMIRegistry (4)
- **Bước 6**: Naming tải Stub của đối tượng xa từ RMIRegistry về Client (5)
- **Bước 7**: Cài đặt đối tượng Stub và trả về tham chiếu đối tượng ở xa cho Client (6)
- **Bước 8**: Client thực thi một lời gọi các phương thức ở xa thông qua đối tượng Stub (7)

Con đường kích hoạt một phương thức ở xa



Khai báo đối tượng ở xa

- Đối tượng ở xa được định nghĩa sử dụng một giao diện ở xa, được mở rộng từ lớp java.rmi.Remote

```
import java.rmi.*;
public interface NameService extends Remote {
    public int search(String s) throws RemoteException;
    public int insert(String s, String hostName, int portNumber)
            throws RemoteException;
    public int getPort(int index) throws RemoteException;
    public String getHostName(int index) throws RemoteException;
}
```

Cài đặt RMI Server (1)

1. Để cài đặt rmi server, đầu tiên chúng ta biên dịch file chứa lớp từ xa (vd. **NameServiceImpl.java)**
2. Sau đó, chúng ta cần tạo đối tượng *stub* liên kết với client và đối tượng *skeleton* liên kết với server
3. Tiếp theo đăng ký đối tượng ở xa với RMIRegistry bằng lệnh *rmiregistry*

Các bước:

> **javac NameServiceImpl.java**
> **rmic NameServiceImpl**
> **rmiregistry** &

Cài đặt RMI Server (2)

- Vấn đề bảo mật !
- Tệp tin chính sách bảo mật (file security policy)
- Ví dụ một *file security policy*:

```
grant {
permission java.net.SocketPermission "*:1024-65535",
"connect,accept";
permission java.net.SocketPermission "*:80", "connect";
};
```

Cài đặt RMI Server (3)

- Bây giờ có thể khởi động RMI Server:

```
> java -Djava.security.policy=policy NameServiceImpl
```

Cài đặt RMI Client (1)

- Lớp *java.rmi.Naming* cung cấp những phương thức để lấy được tham chiếu của đối tượng ở xa
  - Một cách để lấy về tham chiếu đến đối tượng ở xa là dựa trên cú pháp URL
  - URL cho một đối tượng ở xa được xác định bằng cú pháp *rmi://host:port/name*
    - **host** là tên của *RMIRegistry*
    - **port** là số cổng của *RMIRegistry*,
    - **name** là tên của đối tượng ở xa.

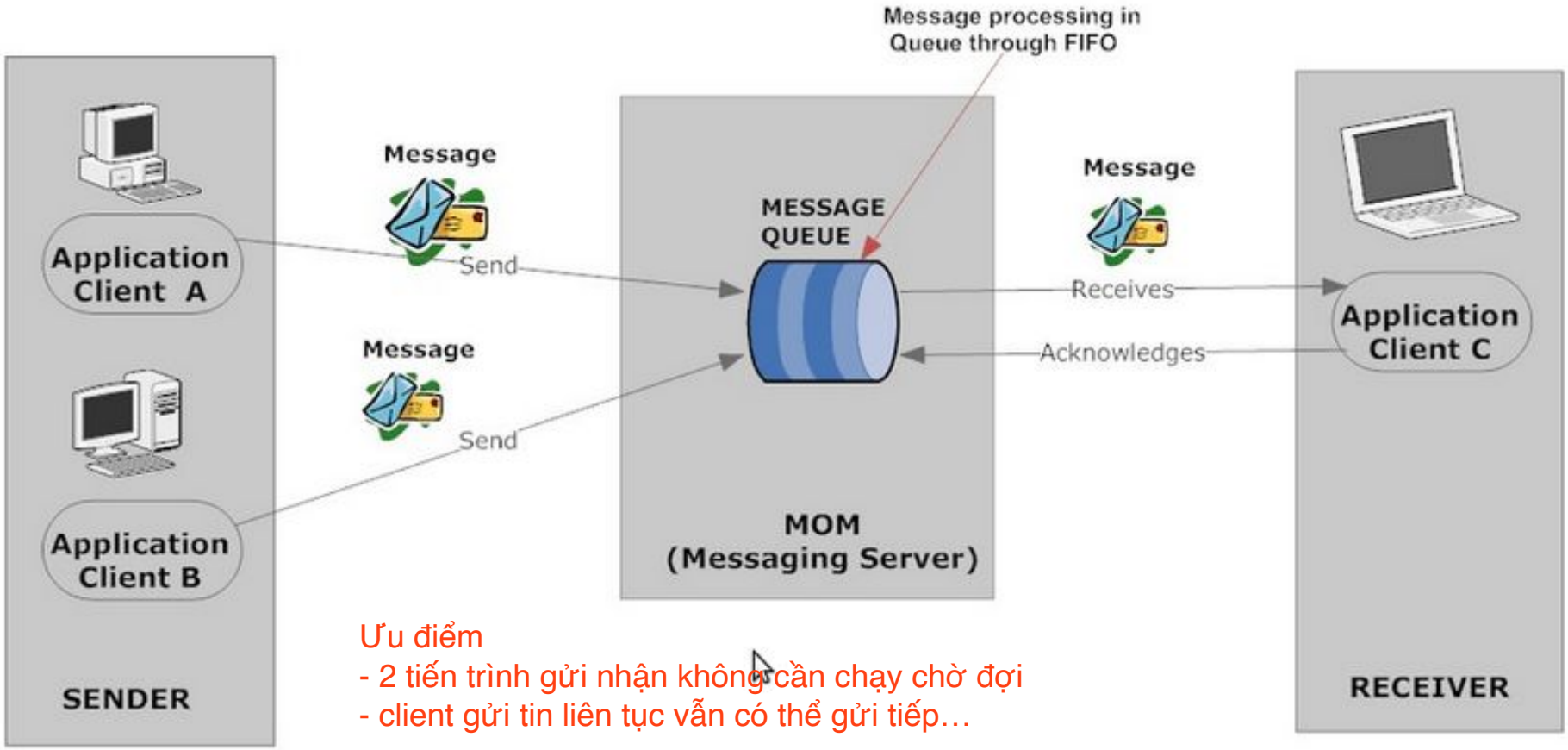
Phương thức chính của lớp Naming

```
bind(String, Remote)
  Binds the name to the specified remote object.
list(String)
  Returns an array of strings of the URLs in the registry.
lookup(String)
  Returns the remote object for the URL.
rebind(String, Remote)
  Rebind the name to a new object; replaces any existing binding.
unbind(String)
  Unbind the name.
```

Cài đặt RMI Client (2)

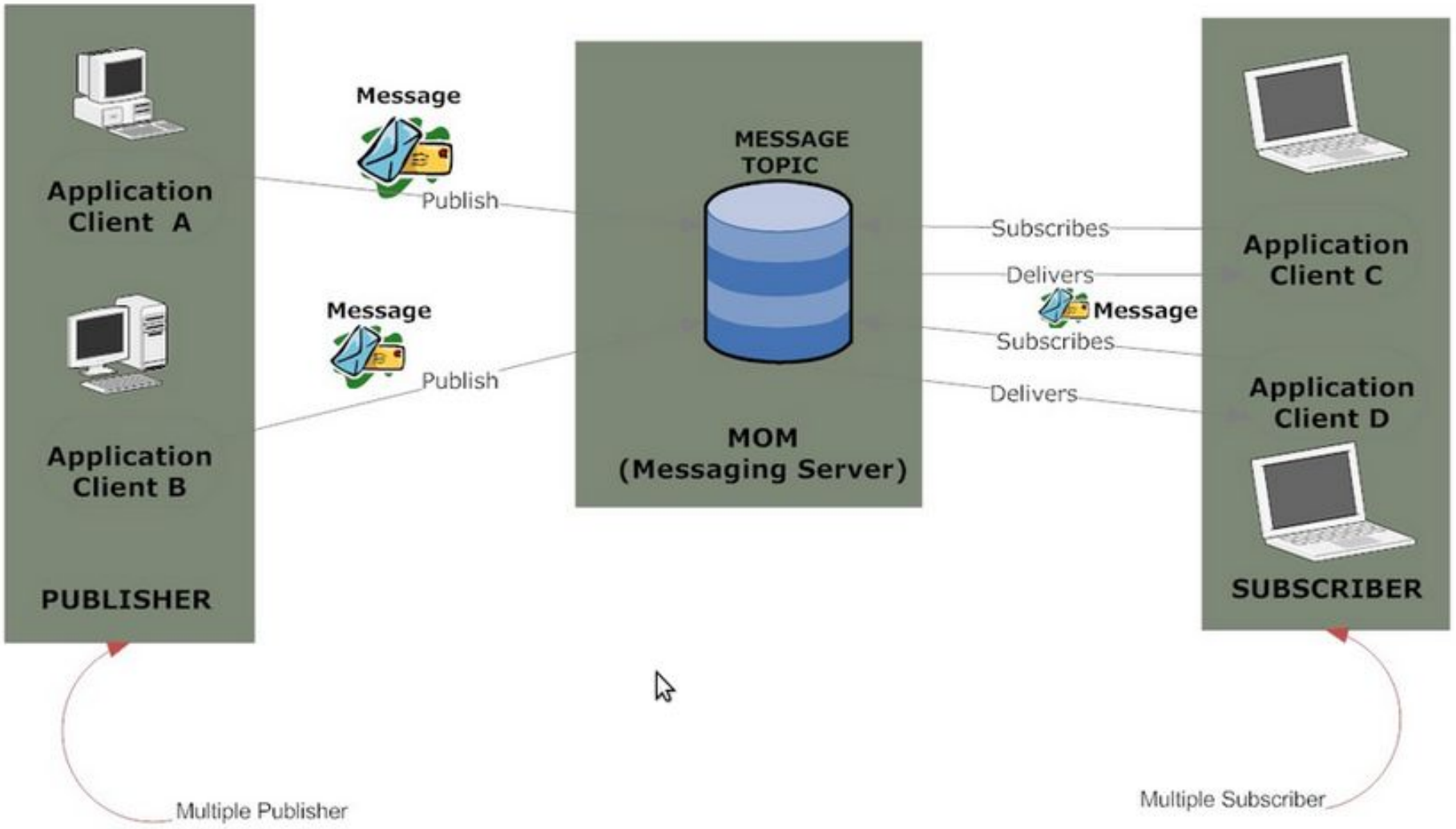
```
import java.rmi.*;
public class NameRmiClient {
    public static void main(String args[]) {
        try {
            NameService r = (NameService)
                            Naming.lookup("rmi://linux02/MyNameServer");
            int i = r.insert("p1", "tick.ece", 2058);
            int j = r.search("p1");
            if (j != -1)
                System.out.println(r.getHostName(j) +":" + r.getPort(j));
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

3.5. Message-Oriented Middleware



Ưu điểm
- 2 tiến trình gửi nhận không cần chạy chờ đợi
- client gửi tin liên tục vẫn có thể gửi tiếp…

thầy chữa:
- MOM: client gửi thông điệp -> không bị chặn lại -> gửi tiếp các gói tin khác dc
- socket RMI: cả 2 tiến trình đều cần sẵn sàng để chạy


Application Client A
Application Client B
PUBLISHER
Message
Publish
Message
Publish
MESSAGE TOPIC
MOM (Messaging Server)
Subscribes
Delivers
Message
Subscribes
Delivers
Application Client C
Application Client D
SUBSCRIBER
Multiple Publisher
Multiple Subscriber

Tài liệu Tham khảo

- *Concurrent and Distributed Computing in Java*, Vijay K. Garg, University of Texas, John Wiley & Sons, 2005
- Tham khảo:
  - *Principles of Concurrent and Distributed Programming*, M. Ben-Ari, Second edition, 2006
  - *Foundations of Multithreaded, Parallel, and Distributed Programming*, Gregory R. Andrews, University of Arizona, Addison-Wesley, 2000
  - *The SR Programming Language: Concurrency in Practice*, Benjamin/Cummings, 1993
  - *Xử lý song song và phân tán*, Đoàn văn Ban, Nguyễn Mậu Hân, Nhà xuất bản Khoa học và Kỹ thuật, 2009