

# Le client/serveur

## L'API socket

## Java RMI

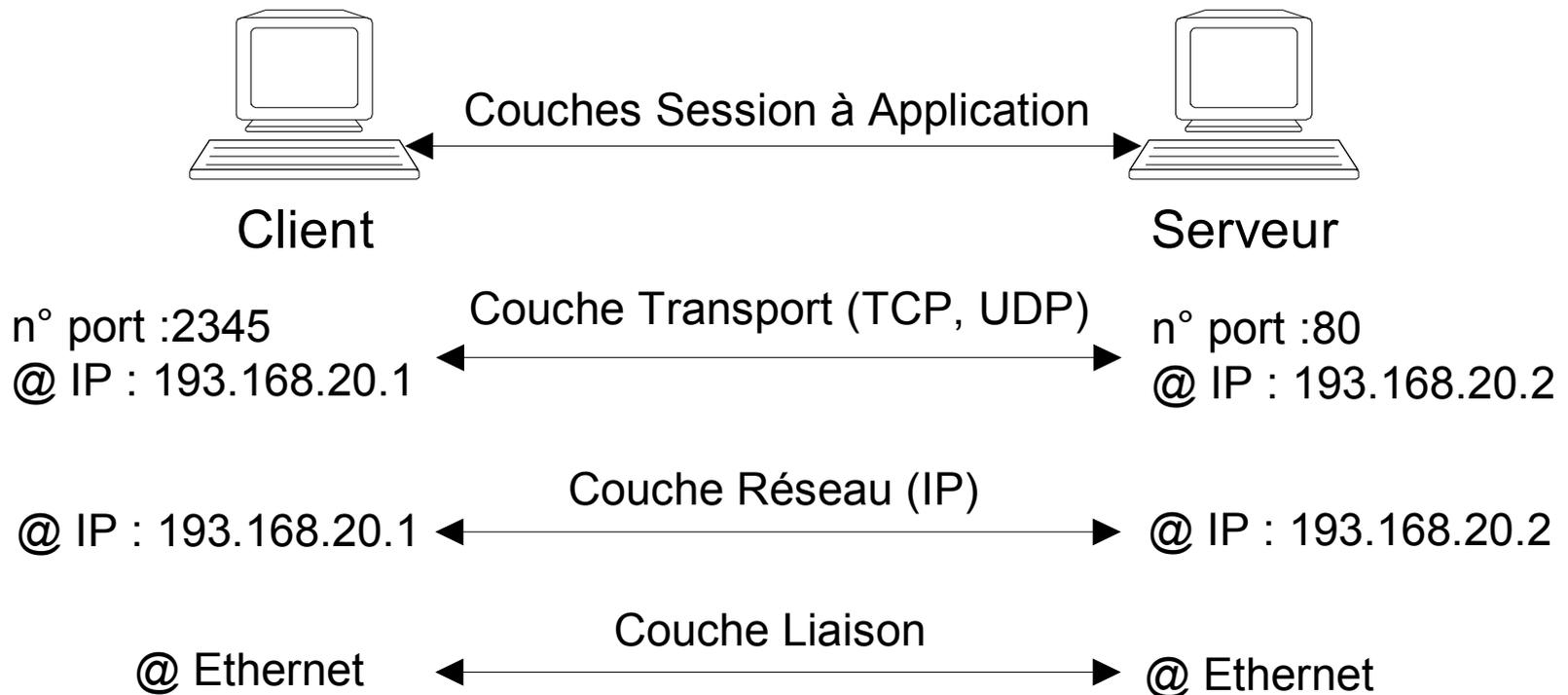
[Noel.DePalma@inrialpes.fr](mailto:Noel.DePalma@inrialpes.fr)

# Mode connecté/non connecté

- Mode connecté (TCP) :
  - problèmes de communications gérés automatiquement,
  - primitives simples d'émission et de réception,
  - gestion de la connexion coûteuse,
  - pas de délimitation des messages dans le tampon.
- Mode non connecté (UDP) :
  - consomme moins de ressources systèmes,
  - permet la diffusion,
  - gestion de toutes les erreurs à la main : il faut réécrire la couche transport !!!

# Les sockets

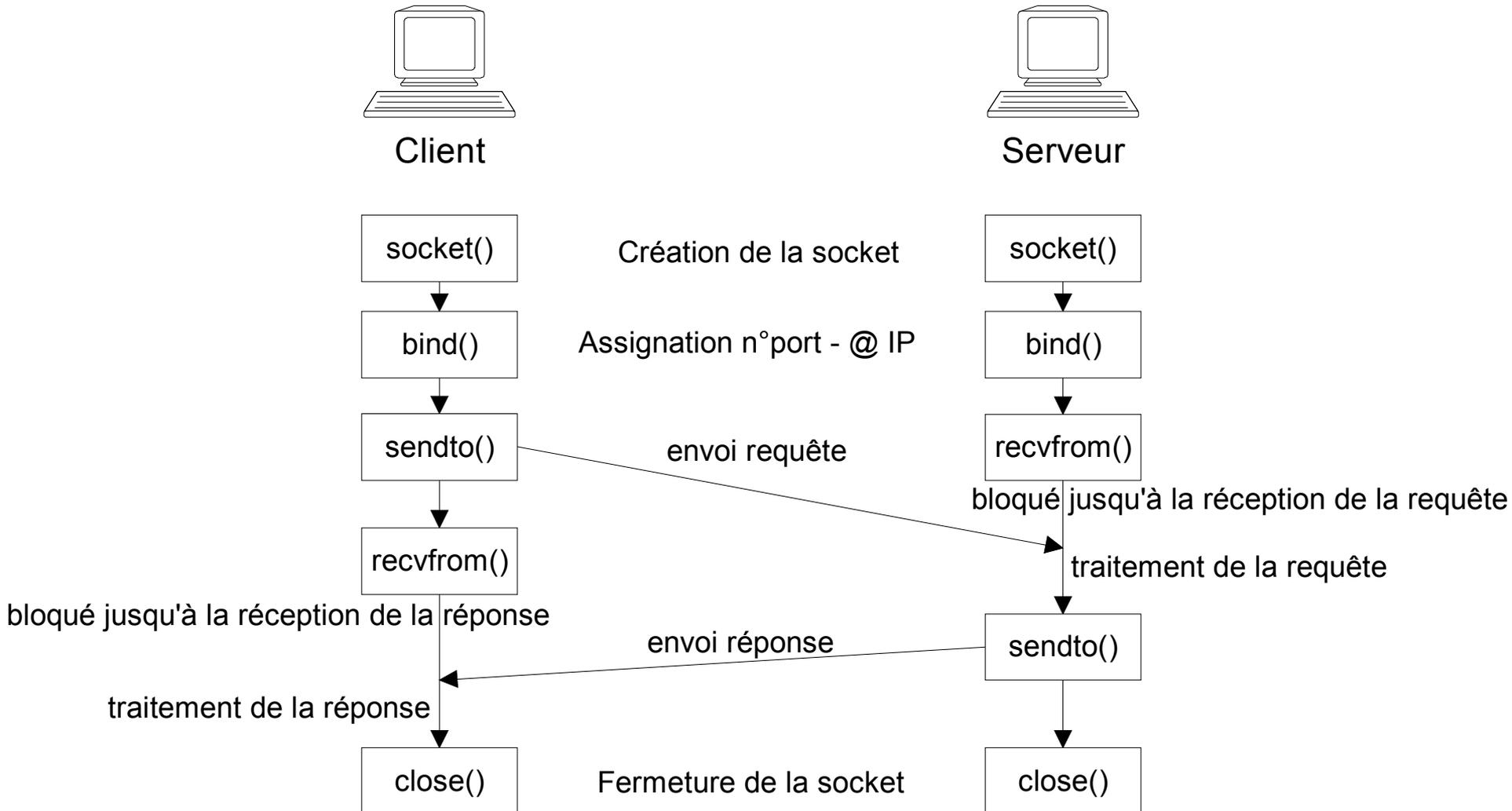
- Interface d'accès au réseau
- développé dans Unix BSD
- n° port, @ IP, protocole (TCP, UDP, ...)



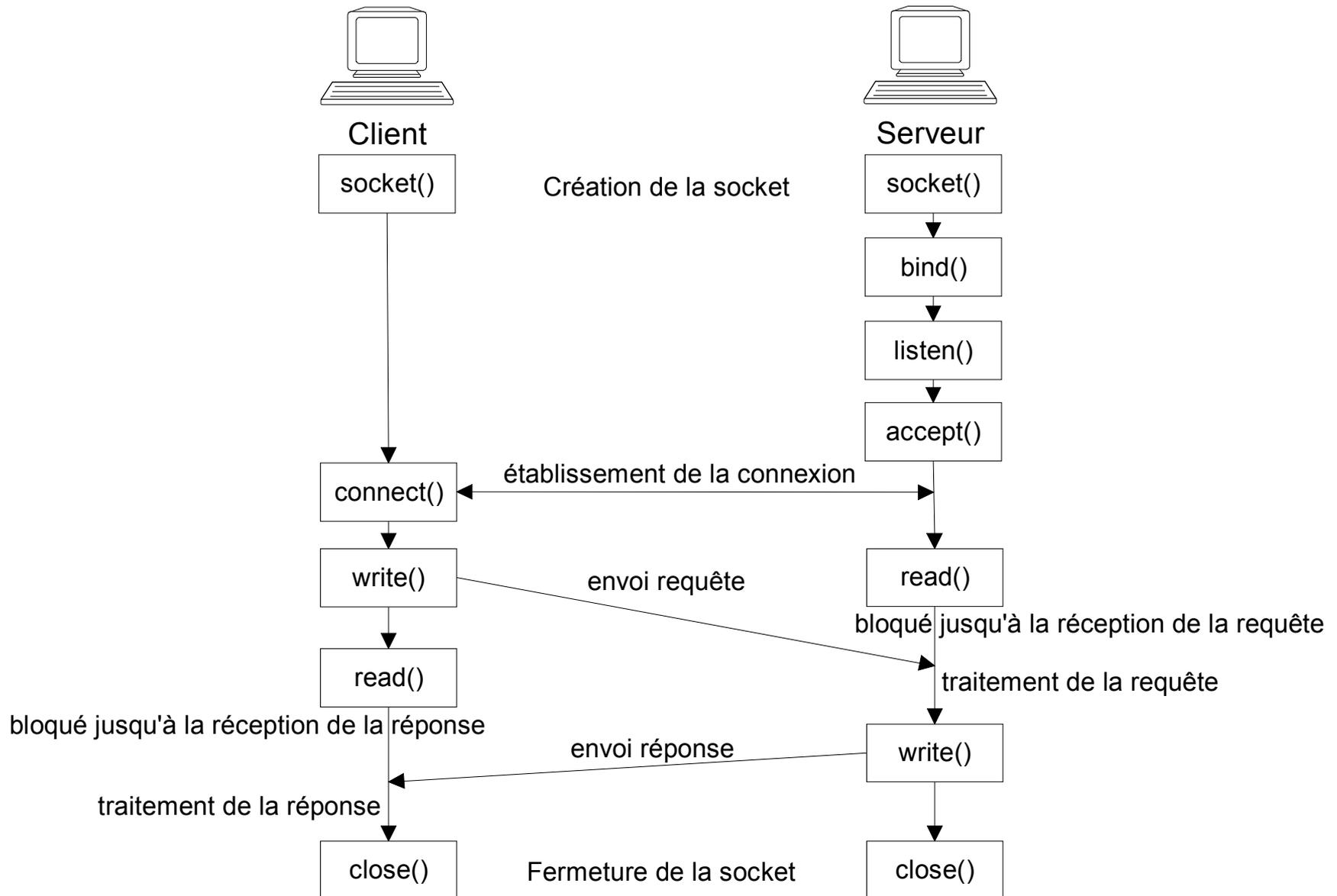
# L'API socket

- Création de socket : `socket(family, type, protocol)`
- Ouverture de dialogue :
  - client : `connect(...)`
  - serveur : `bind(..)`, `listen(...)`, `accept(...)`
- Transfert de données :
  - mode connecté : `read(...)`, `write(...)`, `send(...)`, `recv(...)`
  - mode non connecté : `sendto(...)`, `recvfrom(...)`,  
`sendmsg(...)`, `recvmsg(...)`
- Clôture du dialogue :
  - `close(...)`, `shutdown(...)`

# Client/Serveur en mode non connecté



# Client/Serveur en mode connecté



# Modèle de serveur

- Simple
- Maître/esclave
  - Creation de processus/threads à la demande
  - Pool de processus/threads
- Dupliqué
  - Aiguilleur de requêtes
  - Replication primaire/secondaires
  - Replication active

# Programmation Socket en Java

- **package java.net**
  - **InetAddress**
  - **Socket**
  - **ServerSocket**
  - **DatagramSocket / DatagramPacket**

# Socket cliente et connexion TCP

```
try {  
    Socket s = new Socket  
        ("www.inria.fr", 80);  
    InputStream is = s.getInputStream();  
    ...  
    OutputStream os = s.getOutputStream();  
    ...  
}  
catch (Exception e) {  
    System.err.println(e);  
}  
finally {s.close(); }
```

# Sockets serveur TCP

```
try {  
    ServerSocket serveur = new ServerSocket(port) ;  
    Socket s = serveur.accept() ;  
    OutputStream os = s.getOutputStream() ;  
    InputStream is = s.getInputStream() ;  
    ...  
}  
catch (IOException e) {  
    System.err.println(e) ;  
}  
finally { s.close() ; serveur.close() ; }
```

# Un exemple complet : TCP + serialisation

## Passage d'objets par valeur à l'aide de la serialization

*L'objet à passer à passer au serveur :*

```
public class voiture implements Serializable {  
    public String type;  
    public int imm;  
  
    public voiture(String type, int imm) {  
        this.type = type;  
        this.imm = imm;  
    }  
    public String toString() {  
        return this.type+" "+this.imm;  
    }  
}
```

# Un exemple complet : TCP + serialisation

*Le client*

```
import java.io.*;
import java.net.*;

public class client {
    public static void main (String[] str) {
        try {

            Socket CNX = new Socket("efate",6666);
            ObjectOutputStream p = new
                ObjectOutputStream (CNX.getOutputStream());
            p.writeObject(new voiture("peugeot",171838));
            CNX.close();
        }
        catch (Exception e) {
            System.out.println("An error has occurred ...");
        }
    }
}
```

# Un exemple complet : TCP + serialisation

*Le serveur*

```
public static void main (String[] str) {  
    ServerSocket ss;  
    int port = 6666;  
    ss= new ServerSocket(port);  
    System.out.println("Server ready ...");  
    while (true) {  
        slave rh = new slave(ss.accept());  
        rh.start();  
    }  
}
```

# Un exemple complet : TCP + serialisation

*L'esclave :*

```
public class slave extends Thread {
```

```
    Socket currentRq;
```

```
    public slave(Socket currRq) {  
        this.currentRq = currRq;  
    }
```

```
    public void run() {  
        ObjectInputStream p = new ObjectInputStream(currentRq.getInputStream());  
        voiture v = (voiture)p.readObject();  
        System.out.println("voiture recu : "+ v);  
        currentRq.close();  
    }  
}
```

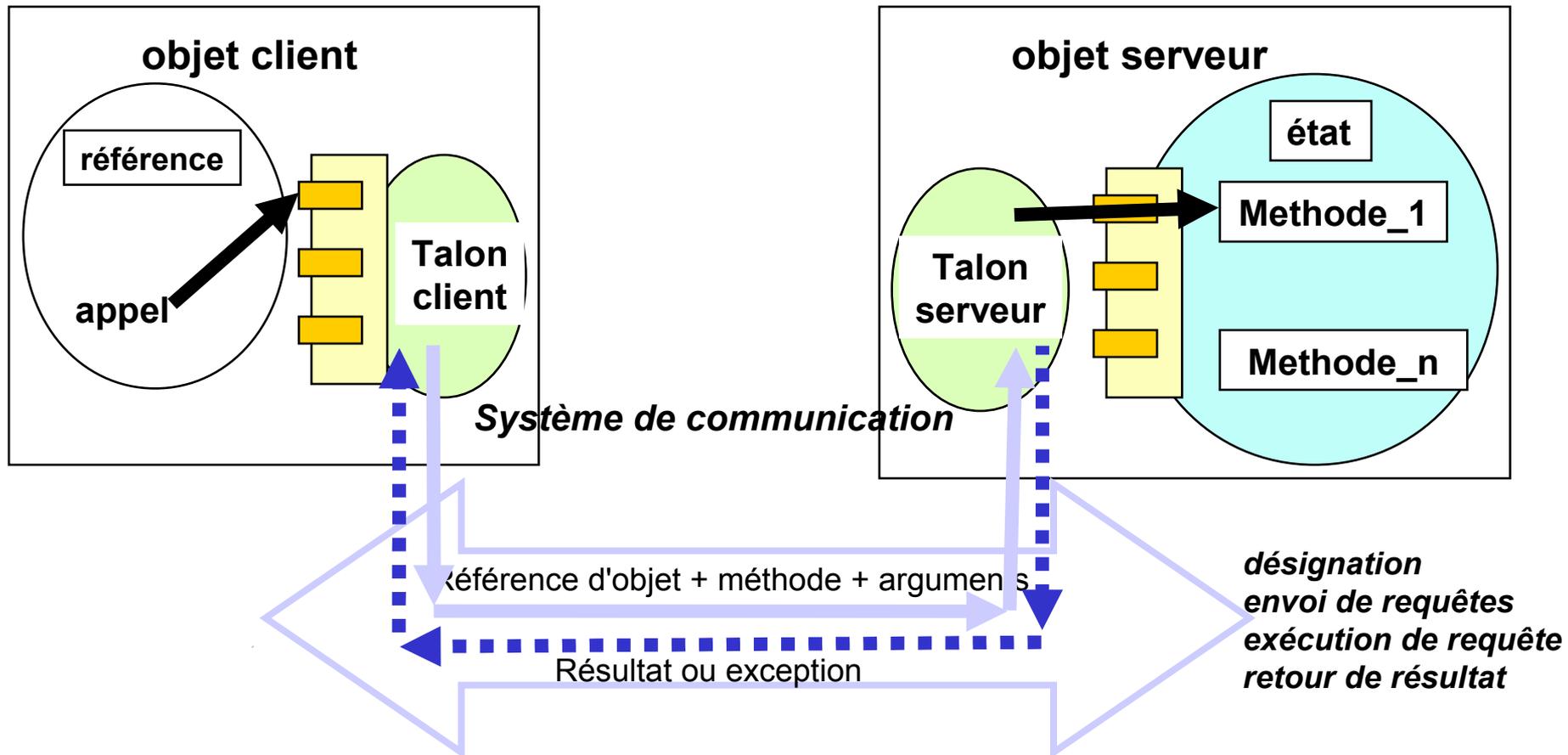
# RPC

## Java RMI

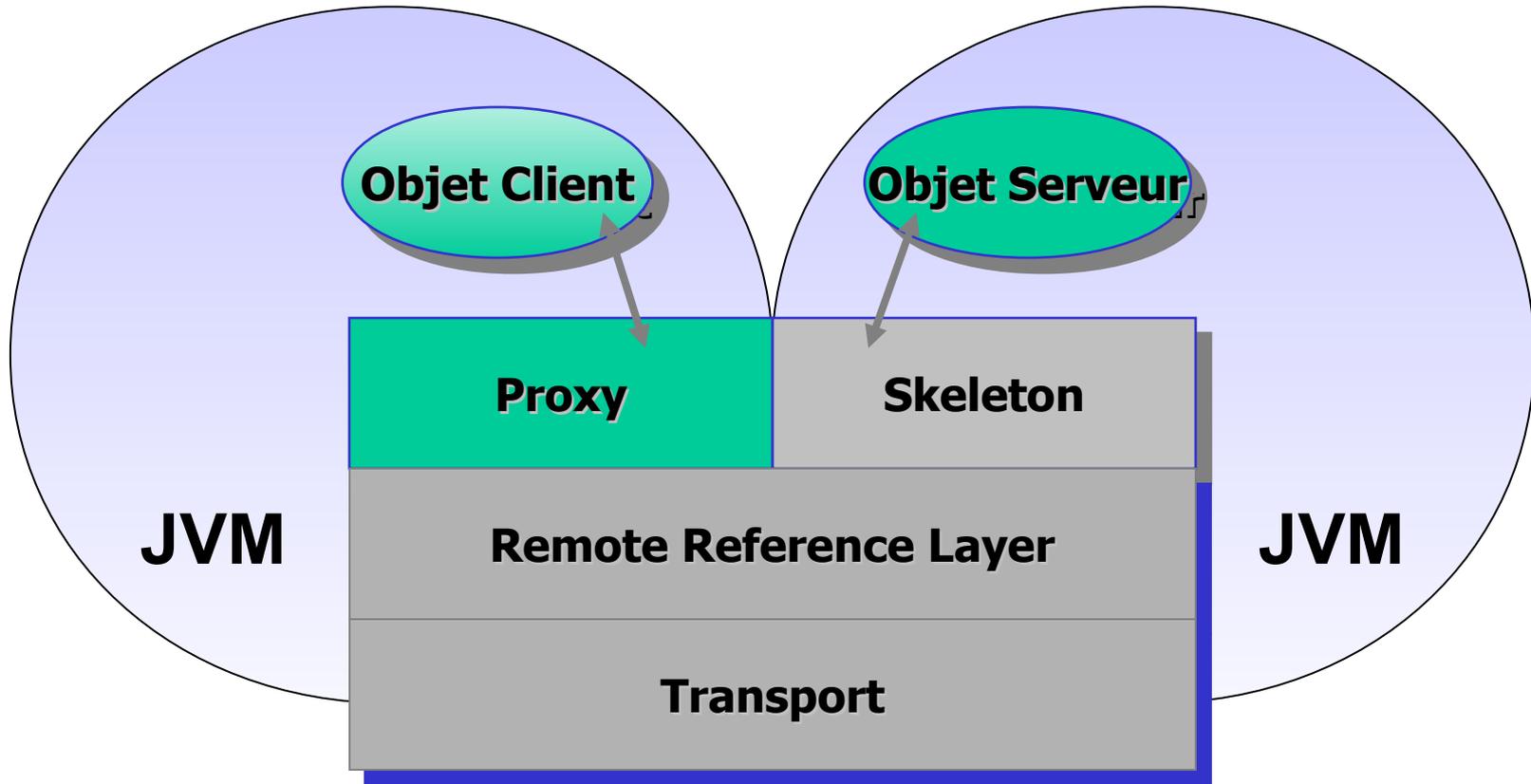
- Un RPC objet intégré à Java
- Interaction d'objets situés dans des espaces d'adressage différents sur des machines distinctes
- Simple à mettre en œuvre : un objet distribué se manipule comme tout autre objet Java

# Appel de méthode à distance

## *Remote Method Invocation (RMI)*



# Java RMI Architecture

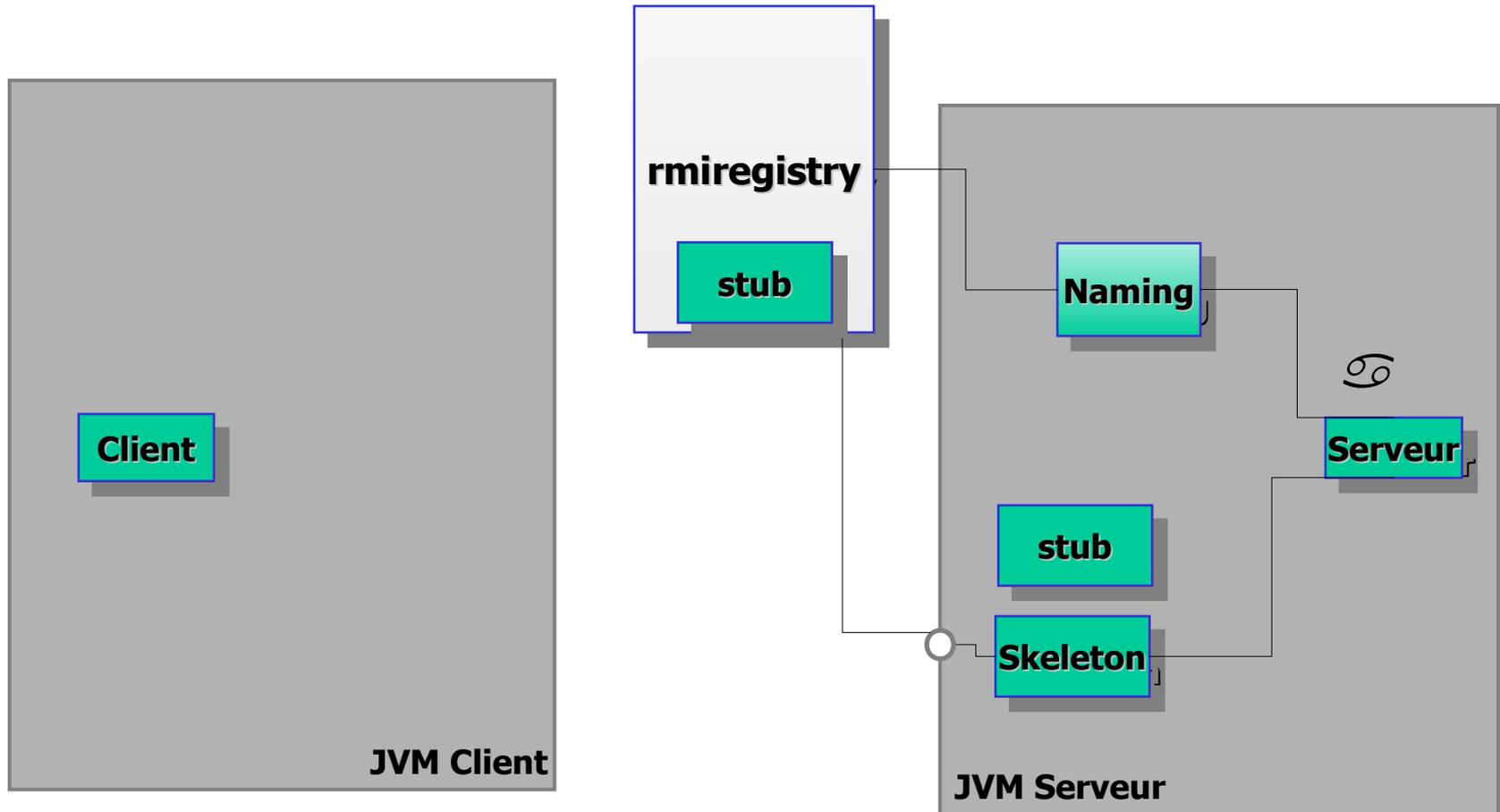


# Java RMI

## Mode opératoire coté serveur

- 0 - A la création de l'objet, un stub et un skeleton (avec un port de communication) sont créés coté serveur
- 1 - L'objet serveur s'enregistre auprès du Naming de sa JVM (méthode *rebind*)
- 2 - Le Naming enregistre le stub de l'objet (sérialisé) auprès du serveur de noms (rmiregistry)
- 3 - Le serveur de noms est prêt à donner des références à l'objet serveur

# Java RMI Architecture

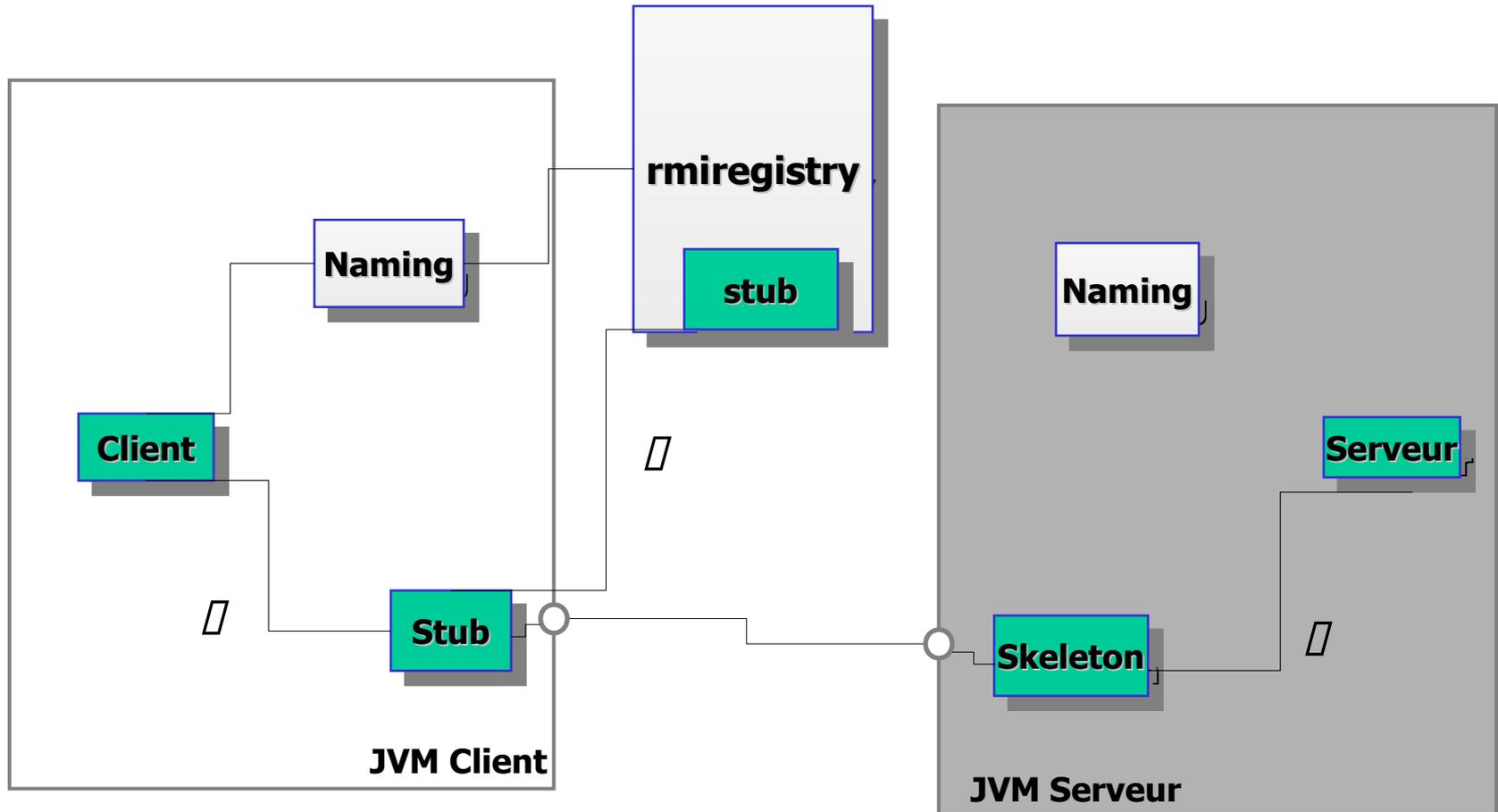


# Java RMI

## Mode opératoire coté client

- 4 - L'objet client fait appel au Naming pour localiser l'objet serveur (méthode *lookup*)
- 5 - Le Naming récupère le stub vers l'objet serveur, ...
- 6 - installe l'objet Stub et retourne sa référence au client
- 7 - Le client effectue l'appel à l'objet serveur par appel à l'objet Stub

# Java RMI Architecture



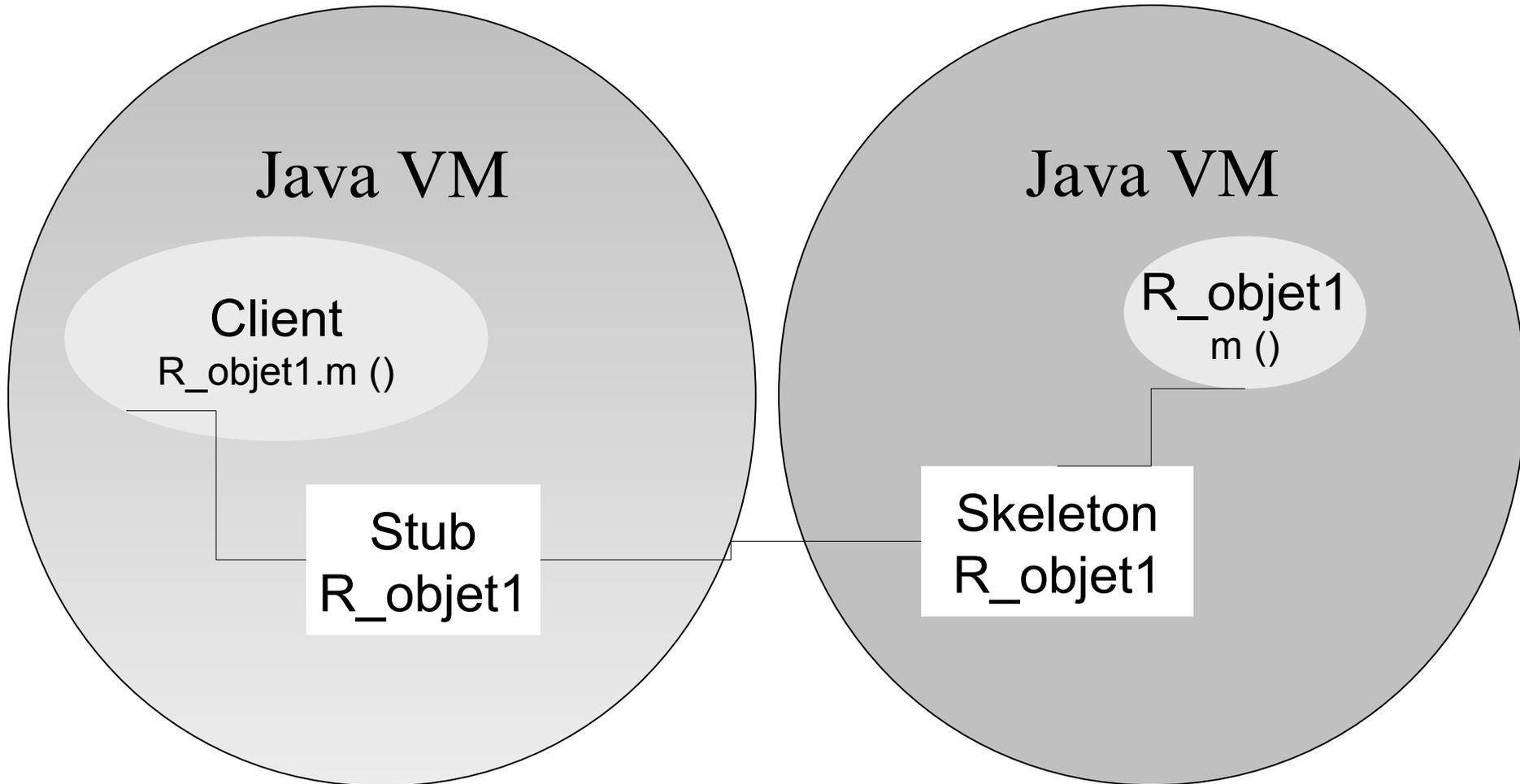
# Java RMI

## Manuel d'utilisation

- Définition de l'interface de l'objet réparti
  - interface : "extends java.rmi.Remote"
  - methodes : "throws java.rmi.RemoteException"
  - paramètres sérializable : "implements Serializable"
  - paramètres référence : "implements Remote"
- Ecrire une implémentation de l'objet serveur
  - classe : "extends java.rmi.server.UnicastRemoteObject"

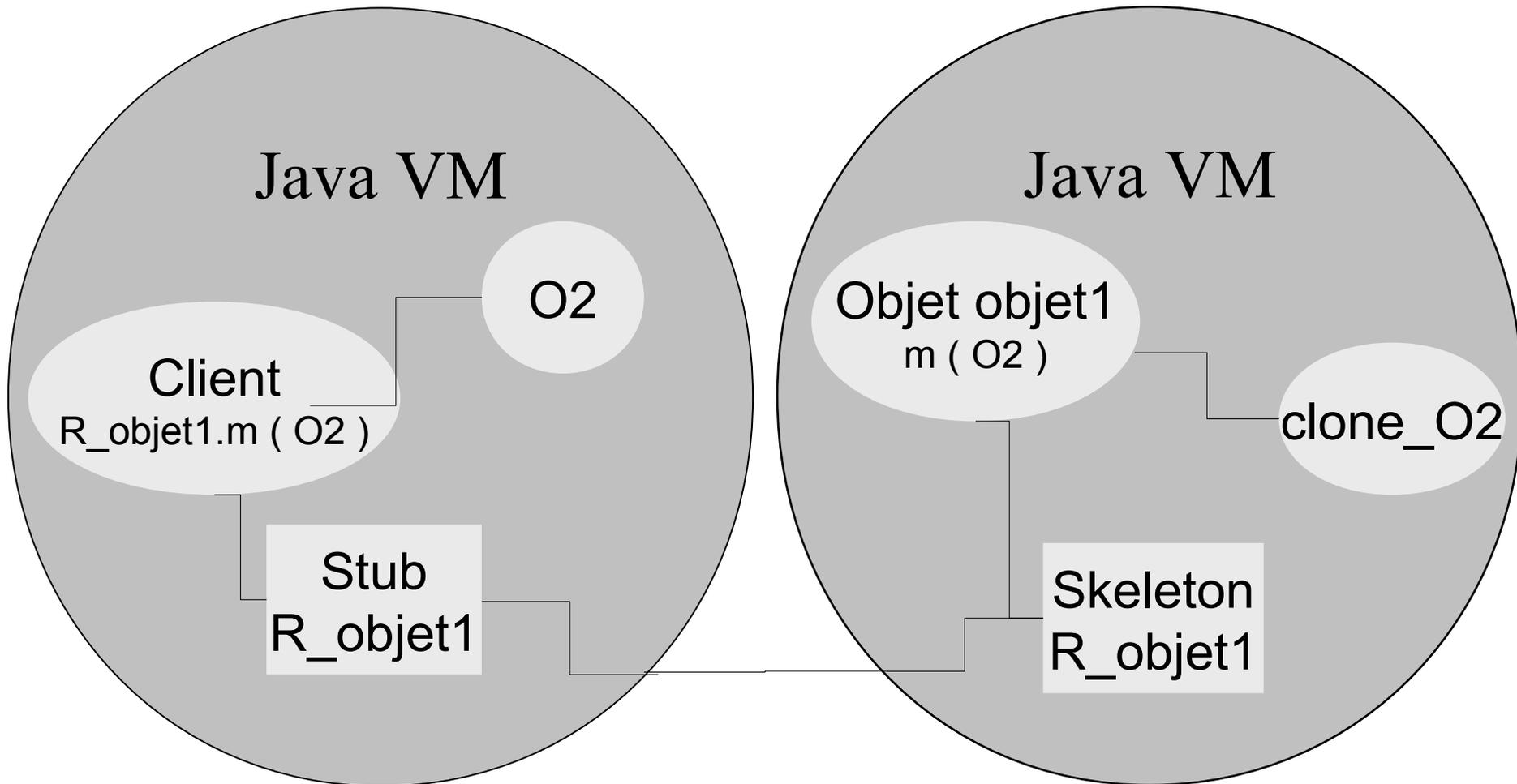
# Java RMI

## Principe de l'appel de procédure



# Java RMI

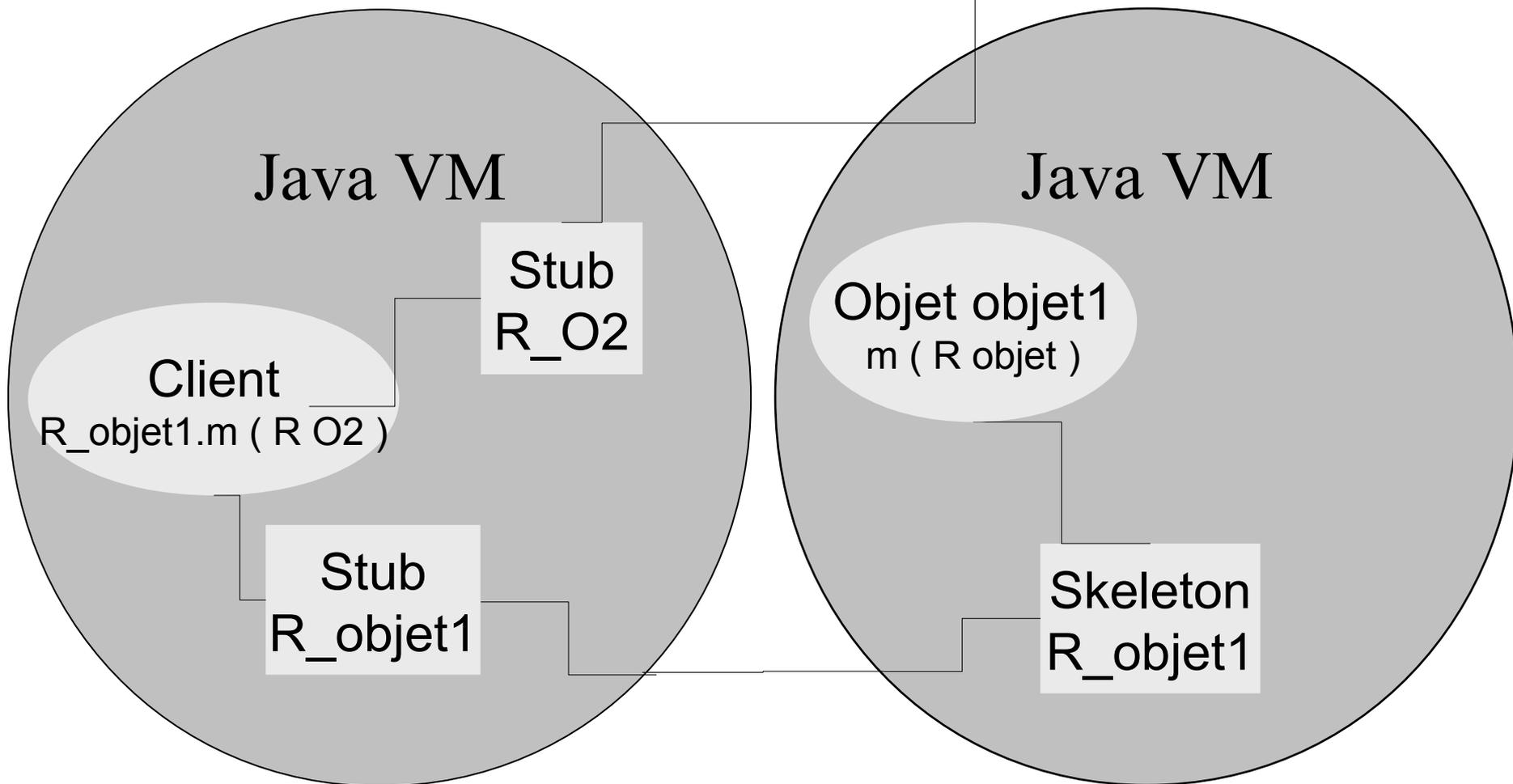
Passage en paramètre d'un objet sérialisable



# Java RMI

Objet O2

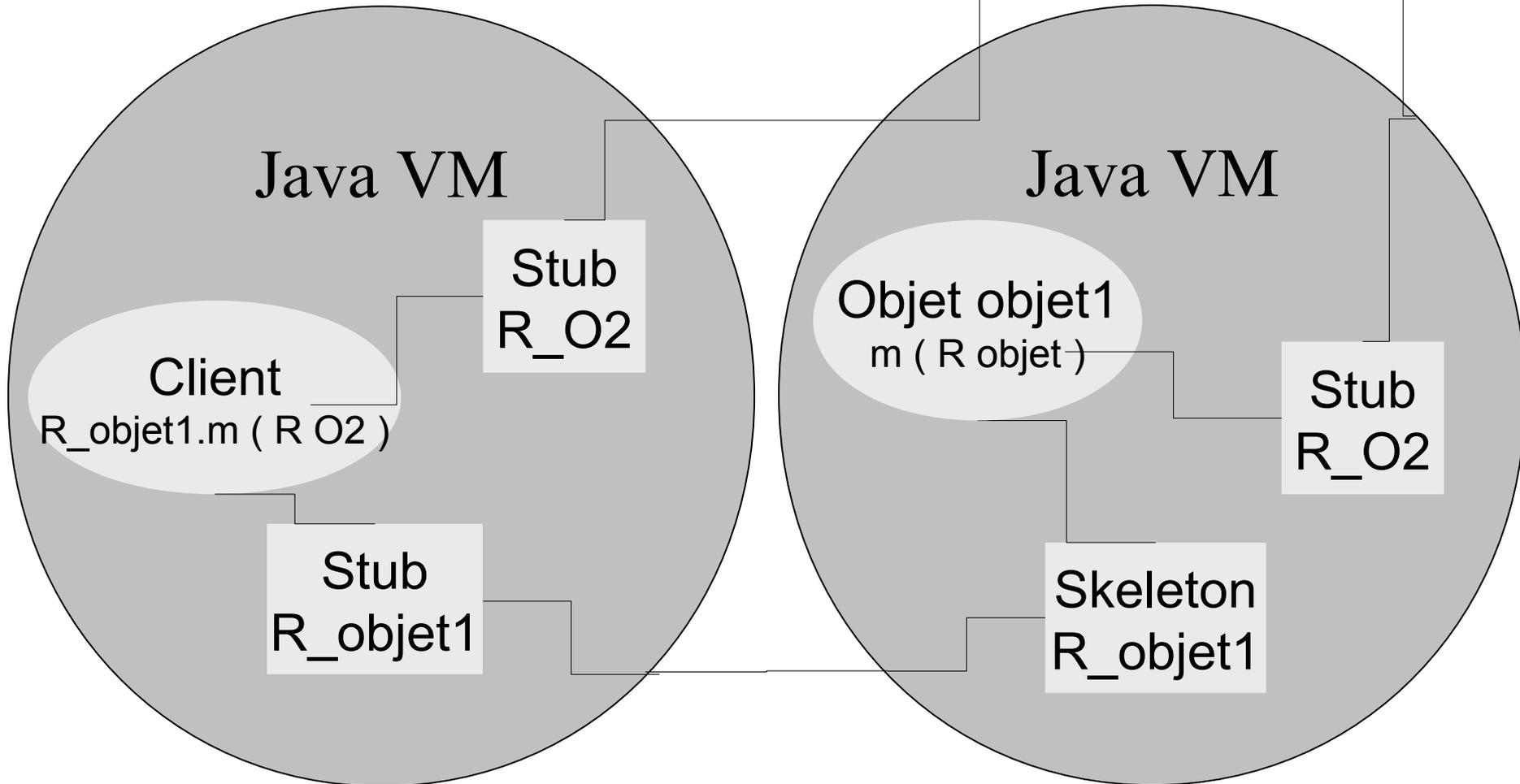
Passage en paramètre d'un objet remote



# Java RMI

Objet O2

Passage en paramètre d'un objet remote



# Java RMI

## Mode opératoire

- codage
  - description de l'interface du service
  - écriture du code du serveur qui implante l'interface
  - écriture du client qui appelle le serveur
- compilation
  - compilation des sources (javac)
  - génération des stub et skeleton (rmic)
- activation
  - lancement du serveur de noms (rmiregistry)
  - lancement du serveur
  - lancement du client

# RPC

## Java RMI : écriture de l'interface

- Mêmes principes de base que pour l'interface d'un objet local
- Principales différences
  - l'interface distante doit être publique
  - l'interface distante doit étendre l'interface `java.rmi.Remote`
  - chaque méthode doit déclarer au moins l'exception `java.rmi.RemoteException`
  - toute référence d'objet (objet "remote" ou "sérialisable") doit être déclaré comme une interface

# Java RMI

## Exemple : Interface

Description  
de  
l'interface

**fichier ConfigManagerItf.java**

```
public interface ConfigManagerItf extends java.rmi.Remote {  
    String getProperty(String name) throws  
        java.rmi.RemoteException;  
  
    void setProperty(String name,String value)  
        throws java.rmi.RemoteException;  
}
```

# Java RMI : écriture du serveur

- **Serveur = la classe qui implémente l'interface**
  - spécifier les interfaces distantes qui doivent être implémentées
    - objets passés par copie (il doivent implémenter l'interface `java.io.Serializable`)
    - objets passés par référence (en fait une référence à un stub)
  - c'est un objet java standard
    - définir le constructeur de l'objet
    - fournir la mise en œuvre des méthodes pouvant être appelée à distance
    - ainsi que celle des méthodes n'apparaissant dans aucune interface implémentée
  - créer au moins une instance du serveur
  - enregistrer au moins une instance dans le serveur de nom (`rmiregistry`)

# Java RMI

## Exemple : Serveur

fichier ConfigServeur.java

```
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;

public class ConfigServeur extends UnicastRemoteObject
    implements ConfigManagerItf {
    private File propFile = ...;

    public void setProperty(String name, String val) throws
        java.rmi.RemoteException {
        propFile.write(name, val);
    }

    public String getProperty(String name) throws
        java.rmi.RemoteException {
        return propFile.read(name);
    }
}
```

Réalisation  
du  
serveur

# Java RMI

## Exemple : Serveur

fichier ConfigServeur.java

```
...  
  
public static void main(String args[]) {  
    try {  
        // Crée une instance de l'objet serveur.  
        ConfigManagerItf obj = new ConfigServeur();  
        // Enregistre l'objet créé auprès du serveur de noms.  
        Naming.rebind("//suldrun/remoteconfig", obj);  
        System.out.println("ConfigServeur " + " bound in registry");  
    } catch (Exception exc) {... }  
}  
}
```

Réalisation  
du  
serveur  
(suite)

ATTENTION : dans cet exemple le serveur de nom doit être activé avant la création du serveur

# Java RMI

Activation du serveur de nom par le serveur

fichier **HelloServeur.java**

```
public static void main(String args[]) {  
    int port; String URL;  
  
    try {                                // transformation d'une chaîne de caractères en entier  
        Integer I = new Integer(args[0]); port = I.intValue();  
    } catch (Exception ex) {  
        System.out.println(" Please enter: Server <port>"); return;  
    }  
  
    try {  
        // Création du serveur de nom - rmiregistry  
        Registry registry = LocateRegistry.createRegistry(port);  
  
        // Création d'une instance de l'objet serveur  
        ConfigManagerItf obj = new ConfigServeur();  
  
        // Calcul de l'URL du serveur  
        URL = "//"+InetAddress.getLocalHost().getHostName()  
+":"+port+"/mon_serveur";  
        Naming.rebind(URL, obj);  
    } catch (Exception exc) { ...
```

# Java RMI

## Exemple : Client

fichier **ConfigClient.java** Réalisation  
du  
client

```
...  
  
public static void main(String args[]) {  
    try {  
        // Enregistre l'objet créer auprès du serveur de noms.  
        ConfigManagerItf s=Naming.lookup("//suldrun/remoteconfig");  
        s.setProperty("prop1", "value");  
    } catch (Exception exc) {... }  
}  
}
```

# Java RMI

## Compilation

- Compilation de l'interface, du serveur et du client
  - javac ConfigManagerItf.java  
ConfigServeur.java ConfigClient.java
- Génération des talons (non obligatoire)
  - rmic ConfigServeur
    - skeleton dans ConfigServeur\_Skel.class
    - stub dans ConfigServeur\_Stub.class.

# Java RMI

## Déploiement

- 1) Activation du serveur de nom
  - start rmiregistry (W95) ou rmiregistry & (Unix)
- 2) Activation du serveur
  - java ConfigServeur
  - java -Djava.rmi.server.codebase=http://suldrun/...
    - path indiquant à quelle endroit la machine virtuelle cliente va pouvoir chercher le code du stub
    - Nécessaire si le client et le serveur ne sont pas sur la même station
- 3) Activation du client
  - java ConfigClient

# Chargement dynamique et sécurité

- Si le code du stub n'est pas présent sur le site local, le protocole RMI prévoit le chargement dynamique du stub en utilisant un serveur web et le protocole HTTP
  - `java -Djava.rmi.server.codebase=http://suldrun/...`
- Si chargement dynamique...
  - RMI impose l'utilisation d'un gestionnaire de sécurité (*SecurityManager*)
    - interdit l'accès à des ressources protégées depuis du code téléchargé
  - créer et installer le "gestionnaire de sécurité"
    - `System.setSecurityManager(new RMISecurityManager());`

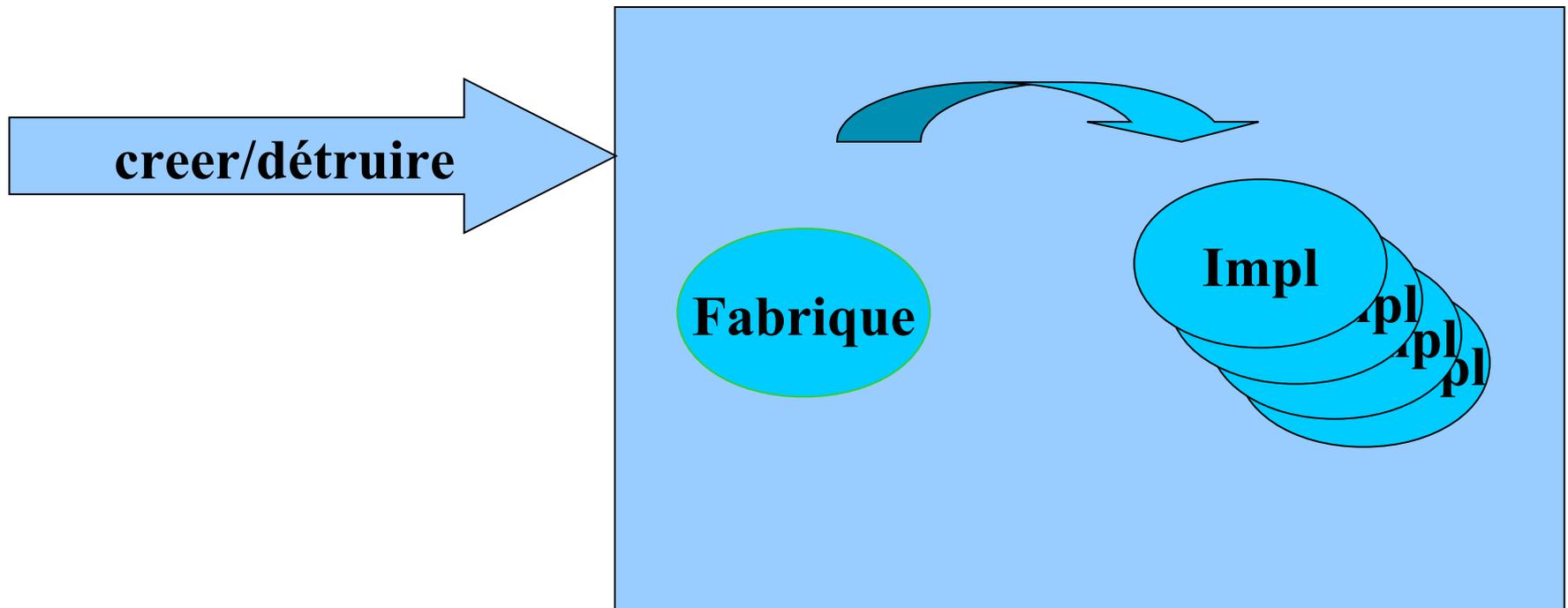
# Pattern de factory

- Creation/destruction d'objet RMI à distance
  - Comment créer d'un objet ConfigServeur à distance et à la demande ?

# Pattern de factory

- Factory

- créer/détruire des objets rmi à distance
- singleton ou non



# Pattern de duplication d'objet rmi

- ConfigServeur RMI dupliqué pour traiter des panne franche de machines
  - Duplication primaire/secondaire
  - Nombres de serveurs statiques
- Architecture et Problèmes ?

# Java RMI : bilan

- Très bon exemple de RPC
  - facilité d'utilisation
  - intégration au langage Java et à l'internet
  - utilisation de l'apport de Java
    - Hétérogénéité des plateformes -> machine virtuelle
    - Passage de référence -> sérialisation ou référence à distance
    - Persistance -> sérialisation
    - Absence de talon -> chargement dynamique
    - Désignation -> URL

# Annexe Socket C

# La primitive socket()

- `int socket(int family, int type, int protocol)`
- `family` :
  - `AF_INET` : pour des communications Internet
  - `AF_UNIX` : pour des communications locales
- `type` ou mode de fonctionnement :
  - `SOCK_STREAM` : mode connecté (TCP)
  - `SOCK_DGRAM` : mode déconnecté (UDP)
  - `SOCK_RAW` : accès direct aux couches basses (IP)
- `protocol` :
  - `IPPROTO_UDP` (protocole UDP avec `SOCK_DGRAM`)
  - `IPPROTO_TCP` (protocole TCP avec `SOCK_STREAM`)
  - `IPPROTO_ICMP` (protocole ICMP avec `SOCK_RAW`)
  - `IPPROTO_RAW` (accès direct IP avec `SOCK_RAW`)

# La primitive connect()

- `int connect(int sock_desc, struct sockaddr * @_serveur, int lg_@)`
- `sock_desc` : descripteur de socket retourné par `socket()`
- `@_serveur` : adresse IP et n° de port du serveur distant
- Exemple de client :

```
int sd;
struct sockaddr_in serveur; // @IP, n° port, mode
struct hostent      remote_host; // nom et @IP

sd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
serveur.sin_family = AF_INET;
serveur.sin_port = htons(13);
remote_host = gethostbyname("brassens.upmf-grenoble.fr");
bcopy(remote_host->h_addr, (char *)&serveur.sin_addr,
      remote_host->hlength); // Recopie de l'adresse
connect(sd, (struct sockaddr *)&serveur, sizeof(serveur));
```

# La primitive bind()

- `int bind(int sock_desc, struct sockaddr *my_@, int lg_@)`
- `sock_desc` : descripteur de socket retourné par `socket()`
- `my_@` : adresse IP et n° de port auxquels le serveur veut répondre
- Exemple de serveur :

```
int sd;
struct sockaddr_in serveur; // @IP, n° port, mode

sd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
serveur.sin_family = AF_INET;
serveur.sin_port = 0; // Laisse le système choisir un port
serveur.sin_addr.s_addr = INADDR_ANY;
    // Autorise des connexions de n'importe où
bind(sd, (struct sockaddr *)&serveur, sizeof(serveur));
```

# La primitive listen()

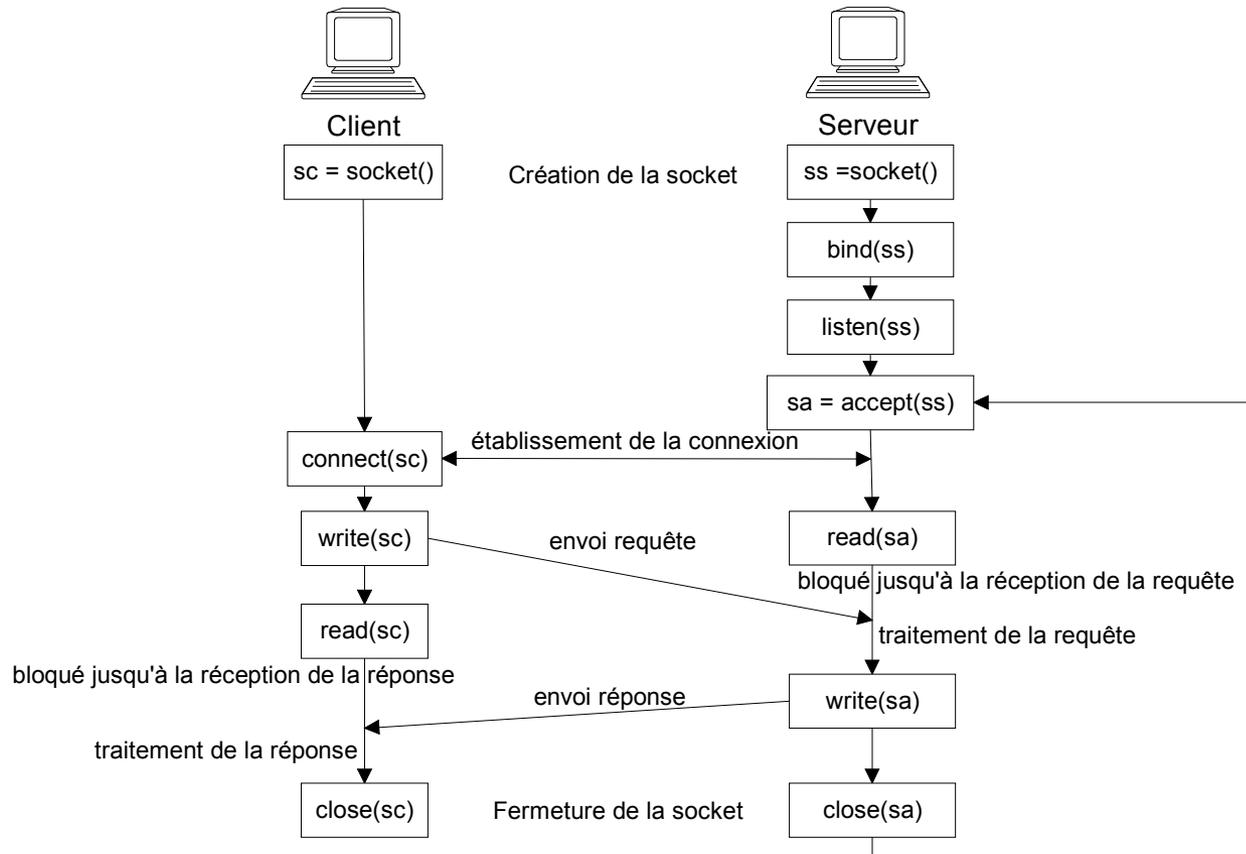
- `int listen(int sock_desc, int backlog)`
- `sock_desc` : descripteur de socket retourné par `socket()`
- `backlog` : nombre maximum de connexions en attente d'être acceptées
- Exemple de serveur :

```
int sd;
struct sockaddr_in serveur; // @IP, n° port, mode

sd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
serveur.sin_family = AF_INET;
serveur.sin_port = 0; // Laisse le système choisir un port
serveur.sin_addr.s_addr = INADDR_ANY;
    // Autorise des connexions de n'importe où
bind(sd, (struct sockaddr *)&serveur, sizeof(serveur));
listen(sd, 5);
```

# La primitive accept()

- `int accept(int sock_desc, struct sockaddr *client, int lg_@)`
- `sock_desc` : descripteur de socket retourné par `socket()`
- `client` : identité du client demandant la connexion
- `accept` renvoie le descripteur de la nouvelle socket créée



# Les primitives d'envoi/réception

- `int write(int sock_desc, char *tampon, int lg_tampon);`
- `int read(int sock_desc, char *tampon, int lg_tampon);`
- `int send(int sock_desc, char *tampon, int lg_tampon, int drap);`
- `int recv(int sock_desc, char *tampon, int lg_tampon, int drap);`
- `int sendto(int sock_desc, char *tampon, int lg_tampon, int drap, struct sockaddr *to, int lg_to);`
- `int recvfrom(int sock_desc, char *tampon, int lg_tampon, int drap, struct sockaddr *from, int lg_from);`
  
- `drap` : options de contrôle de la transmission (consulter le man)

# Un serveur concurrent sous Unix

- Lors du `fork()` le fils hérite des descripteurs du père
- Exemple de serveur :

```
int sd, new_sd;
```

```
...
```

```
sd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
```

```
...
```

```
bind(sd, (struct sockaddr *)&serveur, sizeof(serveur));
```

```
listen(sd, 5);
```

```
while (!fin)
```

```
{
```

```
    nsd = accept(sd, ...);
```

```
    if (fork() == 0)
```

```
    { // C'est le fils !
```

```
        close(sd); // On n'a plus besoin de la socket du père
```

```
        /* On traite ici la connexion avec le client */
```

```
        close(nsd); // Fin de la connexion avec le client
```

```
        exit(0); // Mort du fils
```

```
    }
```

```
    close(nsd); // Le père n'a plus besoin la socket vers le client
```

```
}
```