Chapitre 3
Mécanismes de communication dans les systèmes distribués

Plan

- Passage de messages
  (Message passing)
- Appel de procédure à distance
  (PRC: remote procedure call)
- Appel de méthodes distantes
  (RMI: Remote Method Invocation)
- Communication par flux
  (Stream oriented communication)

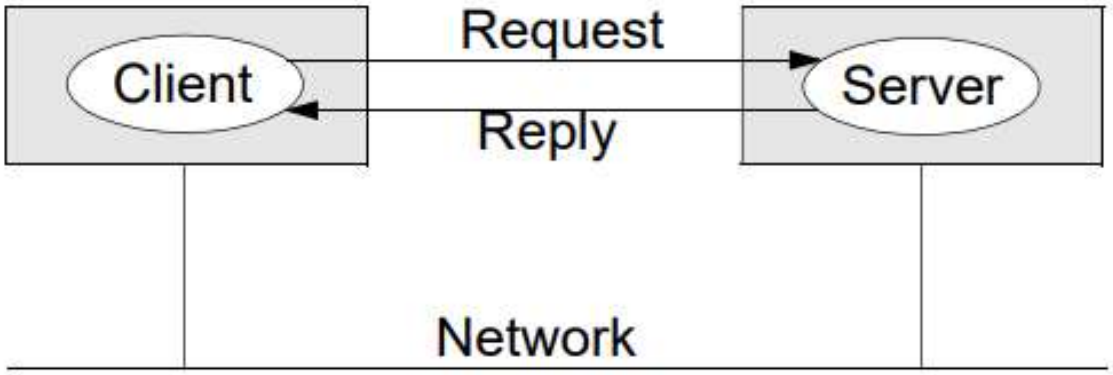
Communication par message

- La communication entre les processus et les objets dans un système distribué est effectuée par passage de messages.

- Dans un scénario typique (par exemple, modèle client-serveur), une telle communication passe par des messages de demande et de réponse.

- Le système est structuré comme un groupe de processus (objets), appelés serveurs, qui fournissent des services aux clients.

Communication par message



The client:

- - - - - - - - - - - - - - - - - -

send (request) to server;
receive (reply);

-- - - - - - - - - - - - - - - - -

The server:

- - - - - - - - - - - - - - - - - -

receive (request) from client;
execute requested operation
send (reply) to client;

- - - - - - - - - - - - - - - - - -

Communication par message

Communication client serveur en utilisant les Sockets

Client

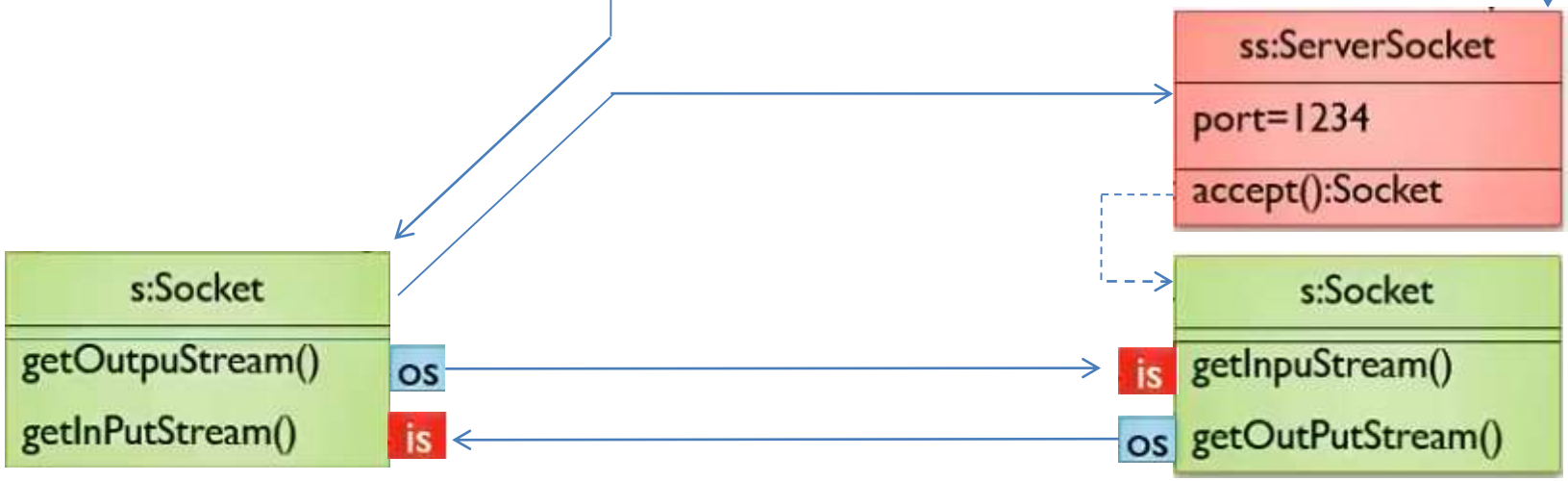
```
Socket s=new Socket("192.168.1.23",1234)

InputStream is=s.getInputStream();
OutputStream os=s.getOutputStream();
os.write(23);
int rep=is.read();
System.out.println(rep);
```

Serveur

```
ServerSocket ss=new ServerSocket(1234);

InputStream is=s.getInputStream();
OutputStream os=s.getOutputStream();
int nb=is.read();
int rep=nb*2;
os.write(rep);
```



RPC : Remote Procedure Call

RPC : Remote Procedure Call

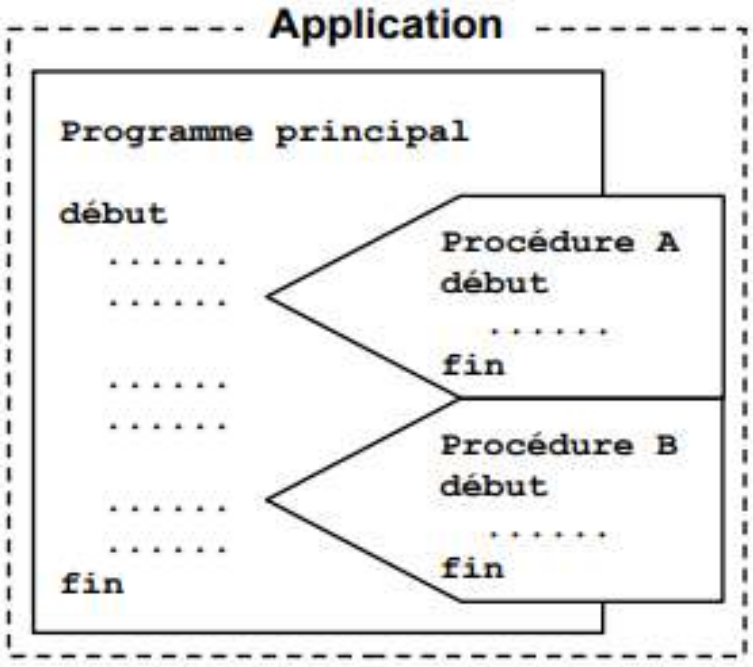
Programmes et appel de procédures

❖ **Applications structurées en 2 parties**

- ➢ Le programme principal
- ➢ Un ensemble de procédures

❖ **Principe de fonctionnement**

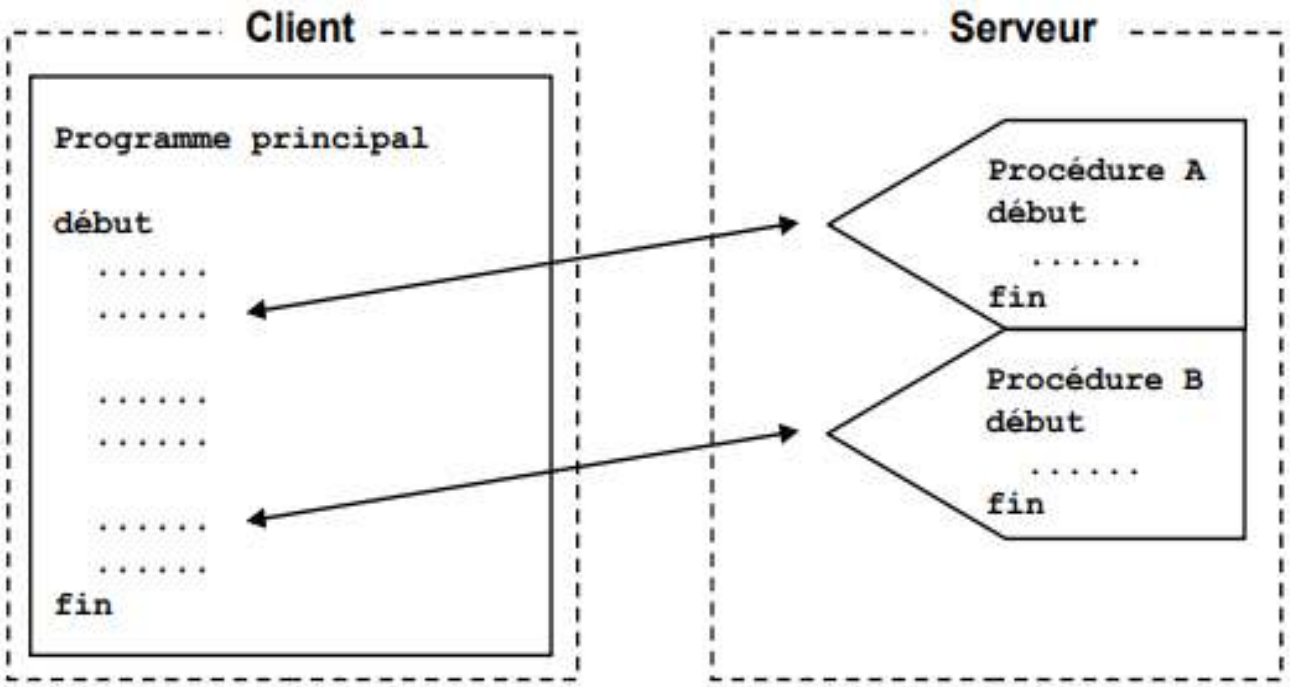
- ➢ Programme principal et procédures sont compilés et liés
- ➢ Durant l'exécution
  - ✓ le programme principal appelle les procédures en transmettant des paramètres d'entrée
  - ✓ les procédures s'exécutent et retournent leurs résultats dans les paramètres de sortie



RPC : Remote Procedure Call

Analogie avec le client-serveur

- Le programme principal se comporte comme un client
- L'ensemble des procédures est assimilable à un ensemble de services disponibles sur un serveur

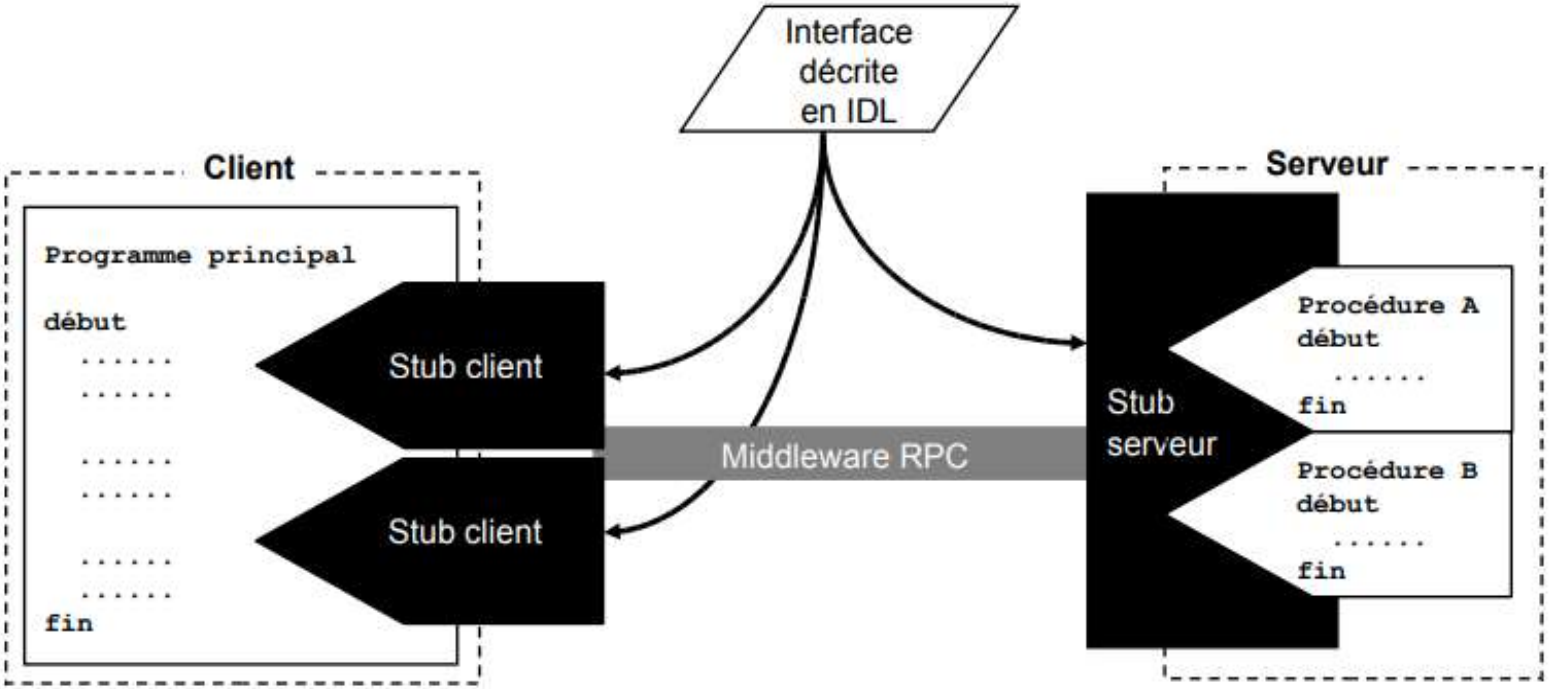


RPC : Remote Procedure Call

Middleware par appel de procédures distantes

Assure la transparence de localisation

- Le client appelle les procédures comme si elles étaient locales
- Le middleware assure la communication avec le serveur
- L'interface du serveur est décrite en IDL (Interface Development Language)
- Le code de préparation d'une requête (stub client) est généré automatiquement à partir de la description IDL



RPC : Remote Procedure Call / Middleware

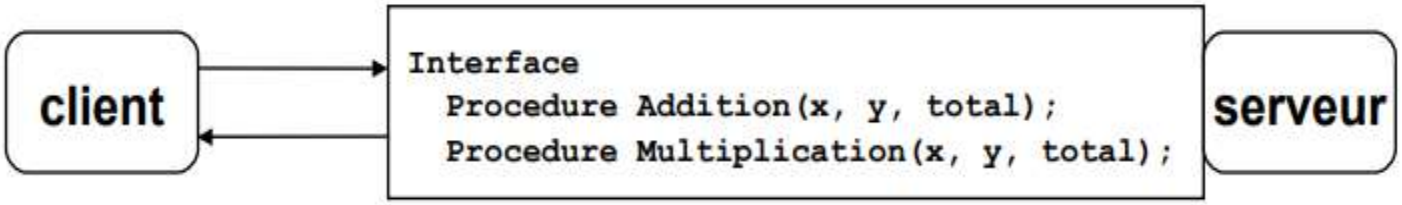
Exemple

```
Programme A;
début
  ....
  appel procédure Addition(1, 2, résultat);
  ....
  appel procédure Multiplication(6, 3, résultat);
  ....
fin
```

```
Programme Addition(x, y, total);
début
  total := x + y;
fin
```

```
Programme Multiplication(x, y, total);
début
  total := x * y;
fin
```

client | serveur



Le client RPC

- Procédure qui porte le nom de la vraie procédure qu'il remplace
- Donne l'illusion au programme principal que la procédure est bien locale
- Remplace le code de la vraie procédure par un autre code
  - ✓gère la connexion avec le bus middleware
  - ✓transmet les paramètres vers la machine où se trouve la procédure
  - ✓récupère le(s) résultat(s)

RPC : Remote Procedure Call / Middleware

Caractéristiques

✓ codes du client et du serveur indépendants du système de communication; le client ne sait pas si la procédure est locale ou éloignée

✓ le code du client n'a pas à préparer le message ni à localiser le serveur => à la charge du middleware RPC

✓ système de dialogue entièrement externe au client et au serveur décrit dans un langage spécifique (IDL) à partir duquel est généré automatiquement le code nécessaire

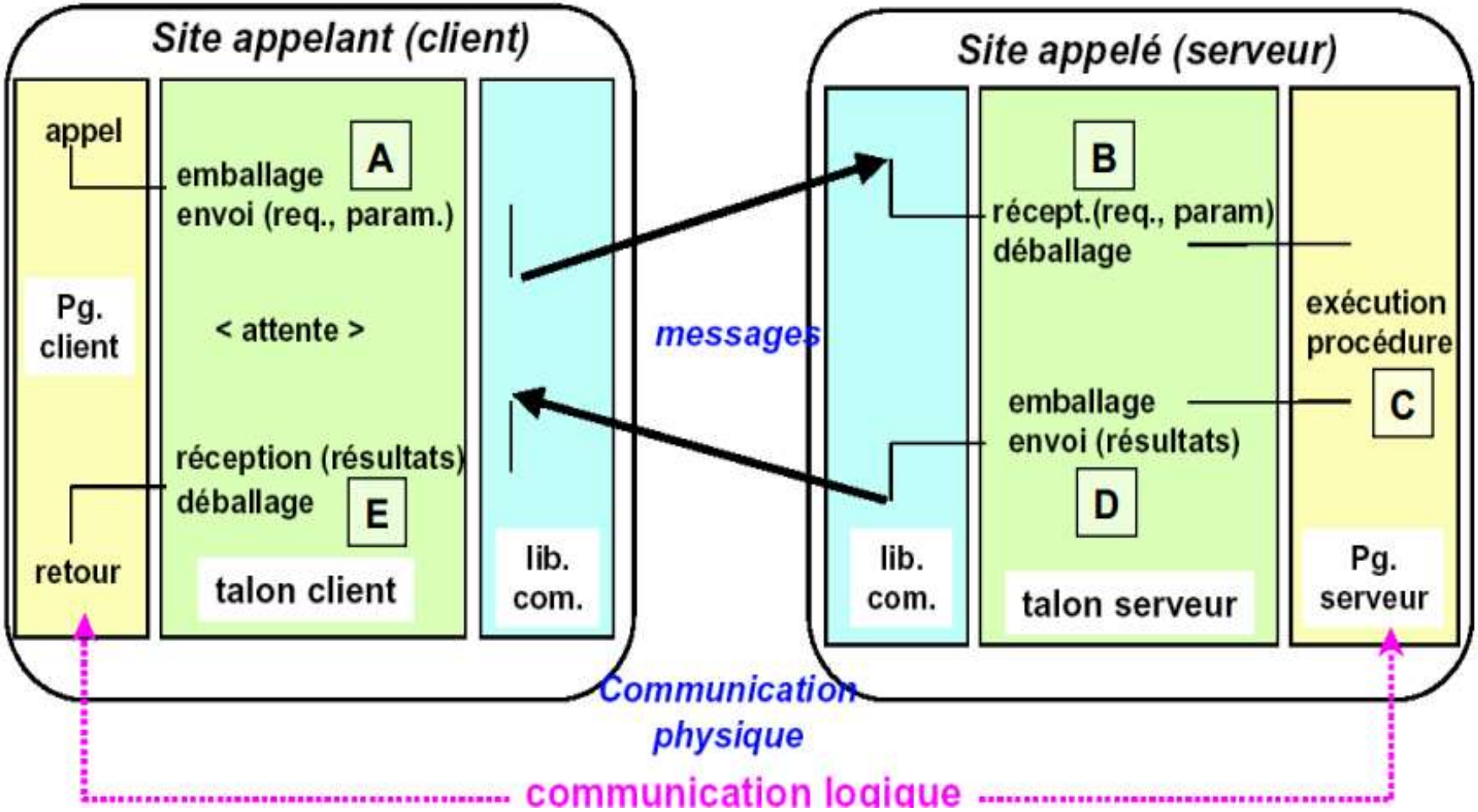
✓ structure de communication construite au moment de la compilation

✓ communication synchrone => le client attend la réponse à son appel de procédure avant de continuer son traitement

✓technologie RPC entièrement standardisée (inclut IDL + services nécessaires à la communication)

Architecture Clients/Serveurs

Principe de réalisation



RPC : principe de fonctionnement

Côté de l'appelant

le client réalise un appel procédural vers la procédure talon client
– transmission de l'ensemble des arguments

au point A

– le talon collecte les arguments et les assemble dans un message (empaquetage - parameter marshalling)
– un identificateur est généré pour le RPC
– un délai de garde est armé
– le talon transmet les données au protocole de transport pour émission sur le réseau. Mais où??
– pb : détermination de l'adresse du serveur

RPC : principe de fonctionnement

Côté de l'appelé

➢ le protocole de transport délivre le message au service de RPC (talon serveur/skeleton)

➢**au point B**

✓ le talon désassemble les arguments (dépaquetage – un marshalling)

✓ l'identificateur de RPC est enregistré

✓ l'appel est ensuite transmis à la procédure distante requise pour être exécuté (point C). Le retour de la procédure redonne la main au service de RPC et lui transmet les paramètres résultats (point D)

➢ **au point D**

✓ les arguments de retour sont empaquetés dans un message

✓ un autre délai de garde est armé

✓le talon transmet les données au protocole de transport pour émission sur le réseau

RPC : principe de fonctionnement

Côté de l'appelé

➢ l'appel est transmis au service de RPC (point E)

✓ les arguments de retour sont dépaquetés

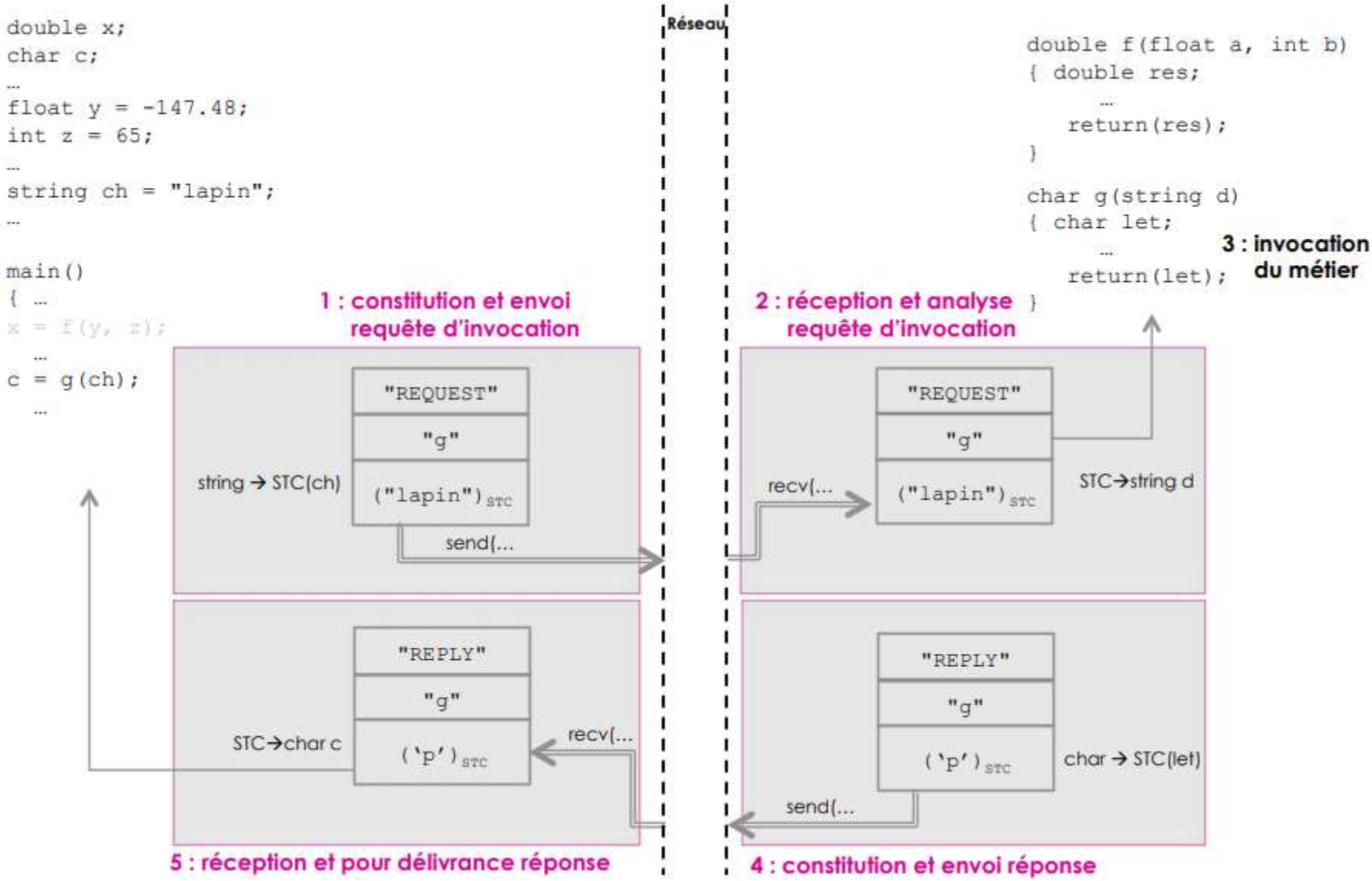
✓ Le délai de garde armé au point A est désarmé

✓ un message d'acquittement avec l'identificateur du RPC est envoyé au talon serveur (le délai de garde armé au point D peut être désarmé)

✓les résultats sont transmis à l'appelant lors du retour de procédure

Architecture Clients/Serveurs

RPC : example



RPC : résumé

Talon client - stub

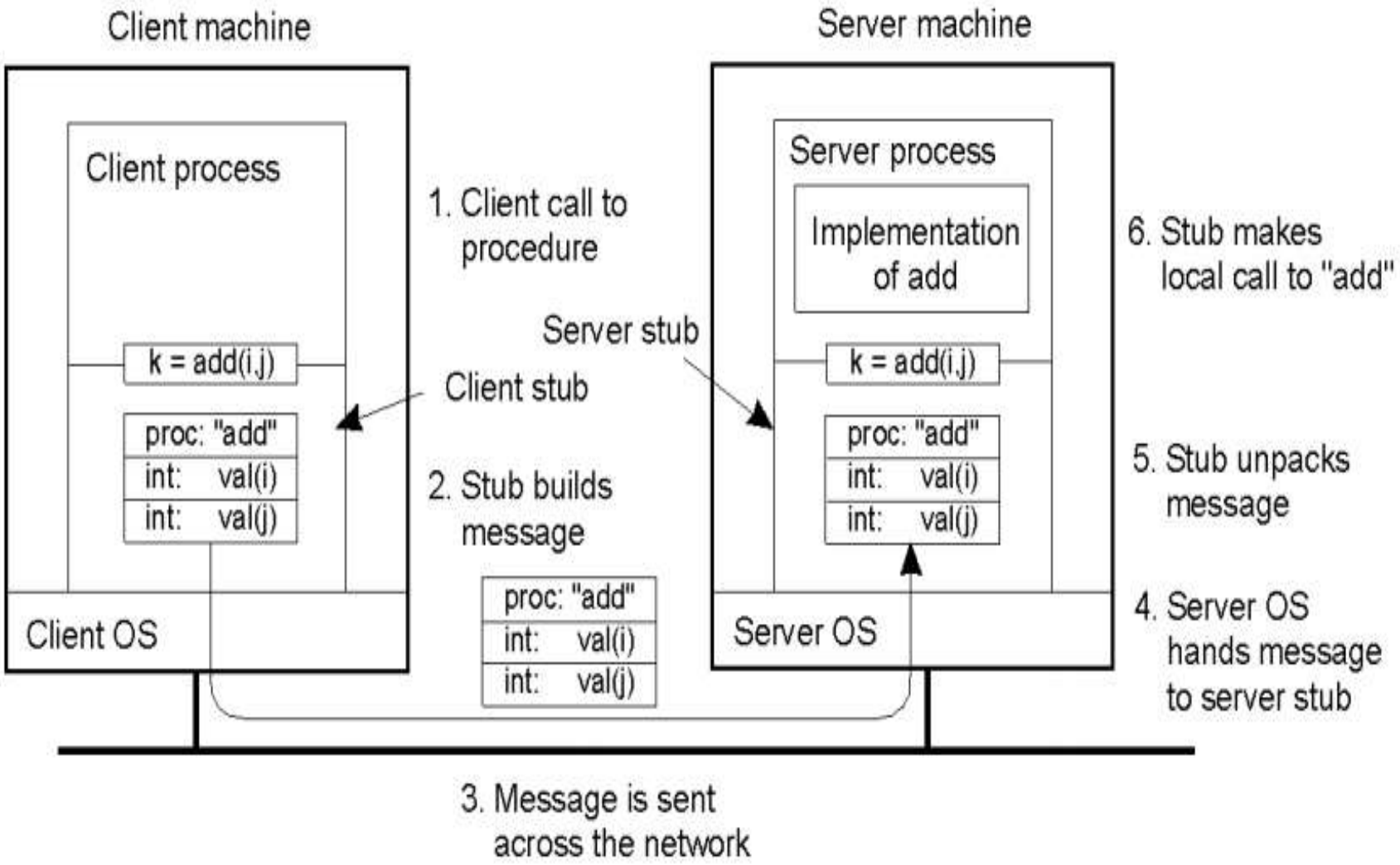
C'est la procédure d'interface du site client
- qui reçoit l'appel en mode local
- le transforme en appel distant en envoyant un message.
- reçoit les résultats après l'exécution
- retourne les paramètres résultats comme dans un retour de procédure.

Talon serveur - skeleton

C'est la procédure sur le site serveur
- qui reçoit l'appel sous forme de message,
- réalise l'appel effectif de la procédure sur le site serveur
- retransmet les résultats sous forme d'un message.

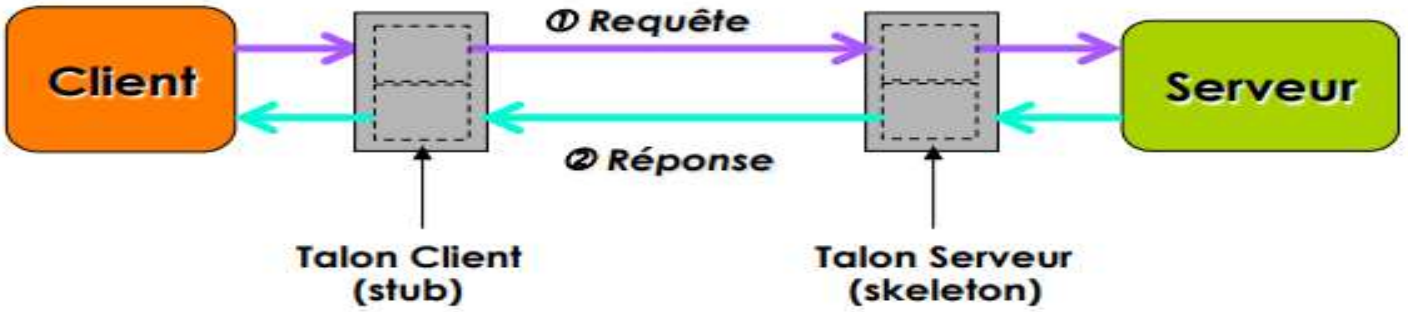
Architecture Clients/Serveurs

RPC : résumé



Architecture Clients/Serveurs

RPC : résumé



➢ Réalisation de l'appel distant (RPC, Remote Procedure Call) via un mandataire appelé talon (ou stub)

➢Rôles des talons:

✓Encodage/décodage des arguments et valeurs de retour (marshal/unmarshal)

✓Adaptation au protocole de communication

✓Appel du service correspondant sur serveur

RPC: les limites

N’assure pas tous les services souhaités d’un bus de communication fiabilité du transfert

- appels perdus si serveur ou réseau en panne
- gestion des erreurs et des pannes à la charge du client concept de transaction diffusion de messages
- communication 1-1
- pas de communication 1-n

Outils de développement

- limités à la génération automatique des talons
- peu d’outils pour le déploiement et la mise au point d'applications réparties

Appel de méthodes distantes
(RMI: Remote Method Invocation)

RMI: Remote Method Invocation

Généralités

RPC

- mécanisme qui permet d'appeler une procédure ou fonction sur un ordinateur distant.
- suit les constructions de langage procédural traditionnelles.
- protocole relativement ancien basé sur le langage C.
- prend en charge uniquement les types de données primitifs, il permet aux objets d'être transmis en tant qu'arguments et de renvoyer des valeurs.
- le programmeur doit fractionner tous les objets composés en types de données primitifs..

Serait il possible d'avoir un environnement RPC orienté objets?

RMI: Remote Method Invocation

RMI

Est une API (Application Programming Interface) qui implémente RPC en Java pour prendre en charge la nature **orientée objet**. Cela permet d'appeler des méthodes d'un objet Java sur une autre machine virtuelle Java résidant sur le **même ordinateur** ou sur un **ordinateur distant**.

Eléments d'une "invocation"

- Référence d'objet
- Identification d'une méthode
- Paramètres d'appel et de retour

RMI: Remote Method Invocation

Caractéristiques

- Invoquer une méthode d'un objet se trouvant sur une autre machine exactement de la même manière que s'il se trouvait au sein de la même machine

objetDistant.methode();

- utiliser un objet distant (OD), sans savoir où il se trouve, en demandant à un service « dédié » de renvoyer son adresse

objetDistant = ServiceDeNoms.recherche("monObjet");

- pouvoir passer un OD en paramètre d'appel à une méthode locale ou distante

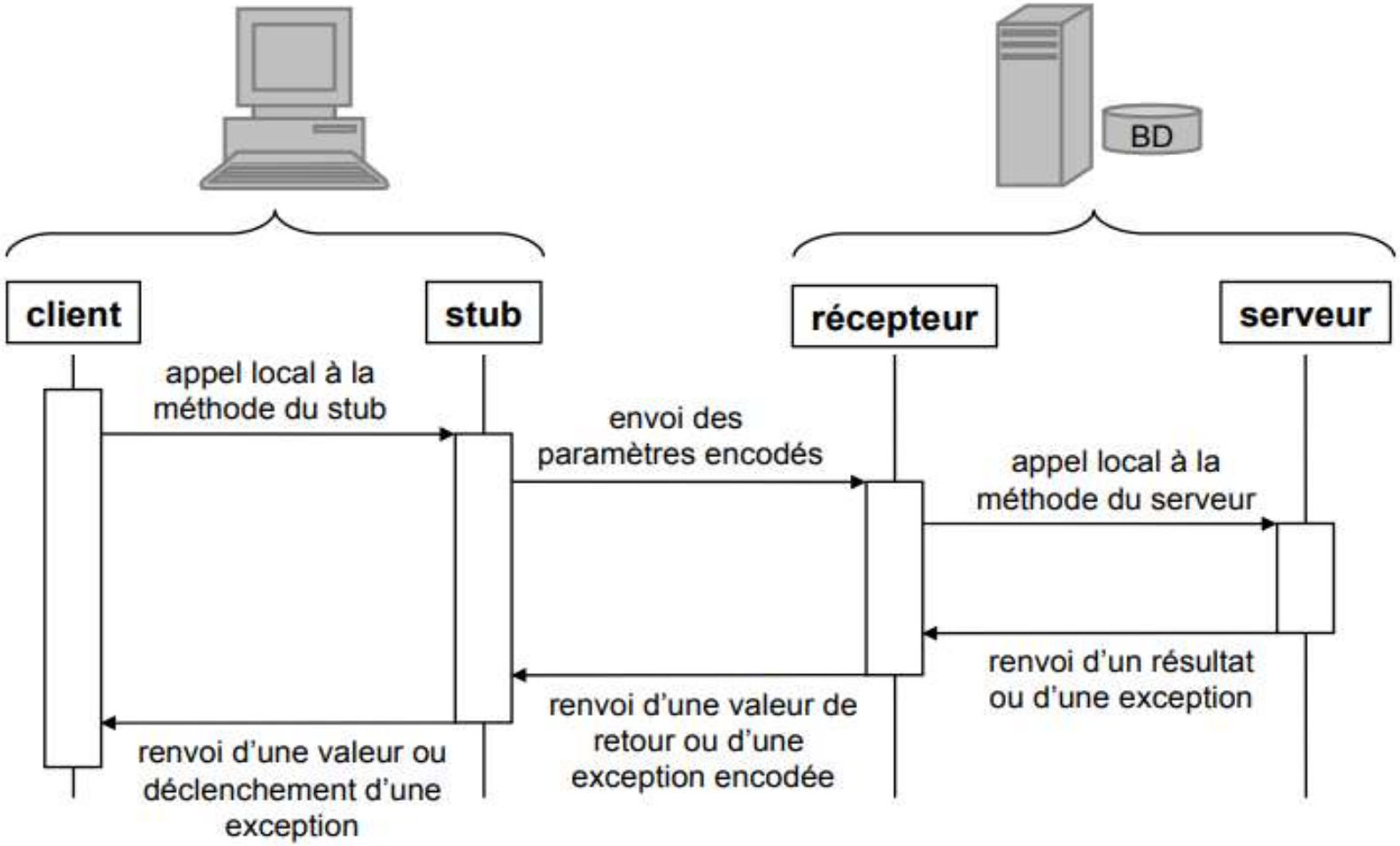
resultat = objetLocal.methode(objetDistant);
resultat = objetDistant.methode(autreObjetDistant);

- pouvoir récupérer le résultat d'un appel distant sous forme d'un nouvel objet qui aurait été créé sur la machine distante

NouvelObjetDistant = ObjetDistant.methode() ;

RMI: Remote Method Invocation

Pour conserver la transparence côté client et serveur, il implémente un objet distant à l'aide de stubs (client) et de squelettes (skeleton serveur)



RMI: Remote Method Invocation

Principe

Mécanisme permettant l'appel de méthodes entre objets Java s'exécutant sur des machines virtuelles différentes (espaces d'adressage distincts), sur le même ordinateur ou sur des ordinateurs distants reliés par un réseau

- Utilise directement les sockets
- Code ses échanges avec un protocole propriétaire : RMP (Remote Method Protocol)

Objectifs

- Rendre transparent l'accès à des objets distribués sur un réseau
- Faciliter la mise en œuvre et l'utilisation d'objets distants Java
- Préserver la sécurité (inhérent à l'environnement Java)
  - ✓RMISecurityManager
  - ✓Distributed Garbage Collector (DGC)

RMI: Remote Method Invocation

Objet distant (objet serveur)

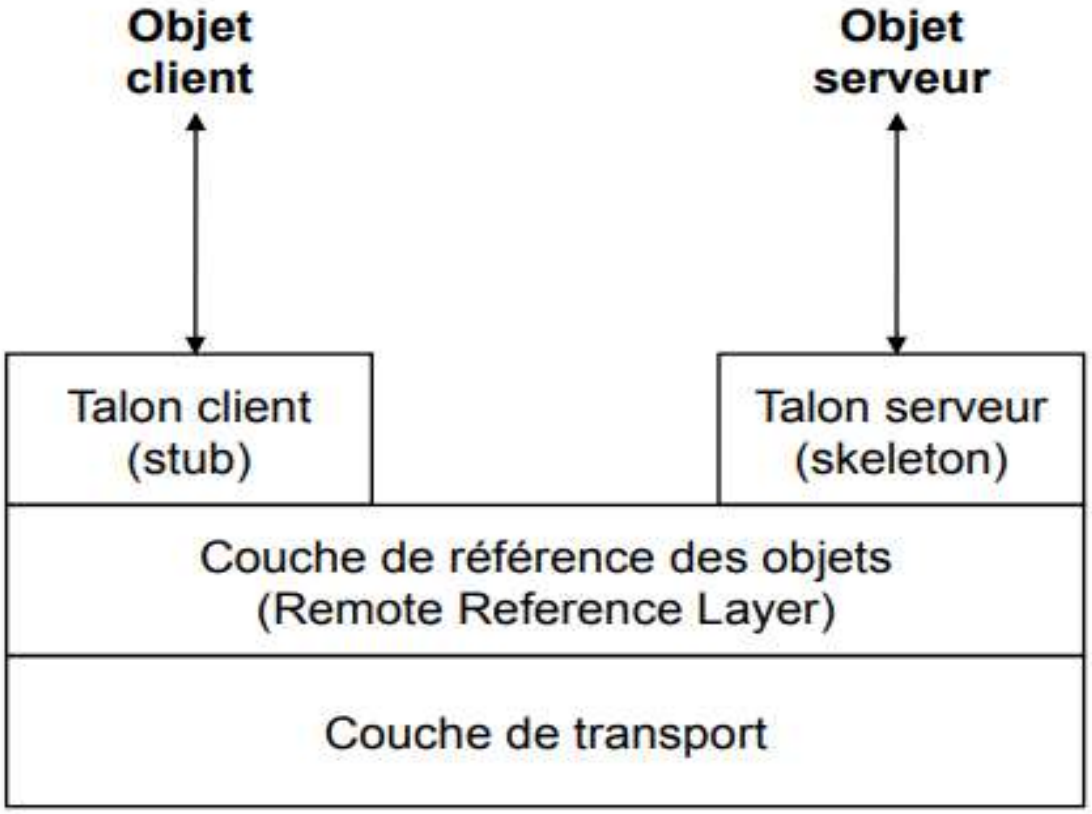
- ses méthodes sont invoquées depuis une autre JVM
  - ✓ dans un processus différent (même machine)
  - ✓ dans une machine distante (via réseau)
- son comportement est décrit par une interface (ou plus) distante Java
  - ✓ déclare les méthodes distantes utilisables par le client
- se manipule comme un objet local

Invocation distante

- action d'invoquer une méthode d'une interface distante d'un objet distant
  - ✓ même syntaxe qu'une invocation sur un objet local

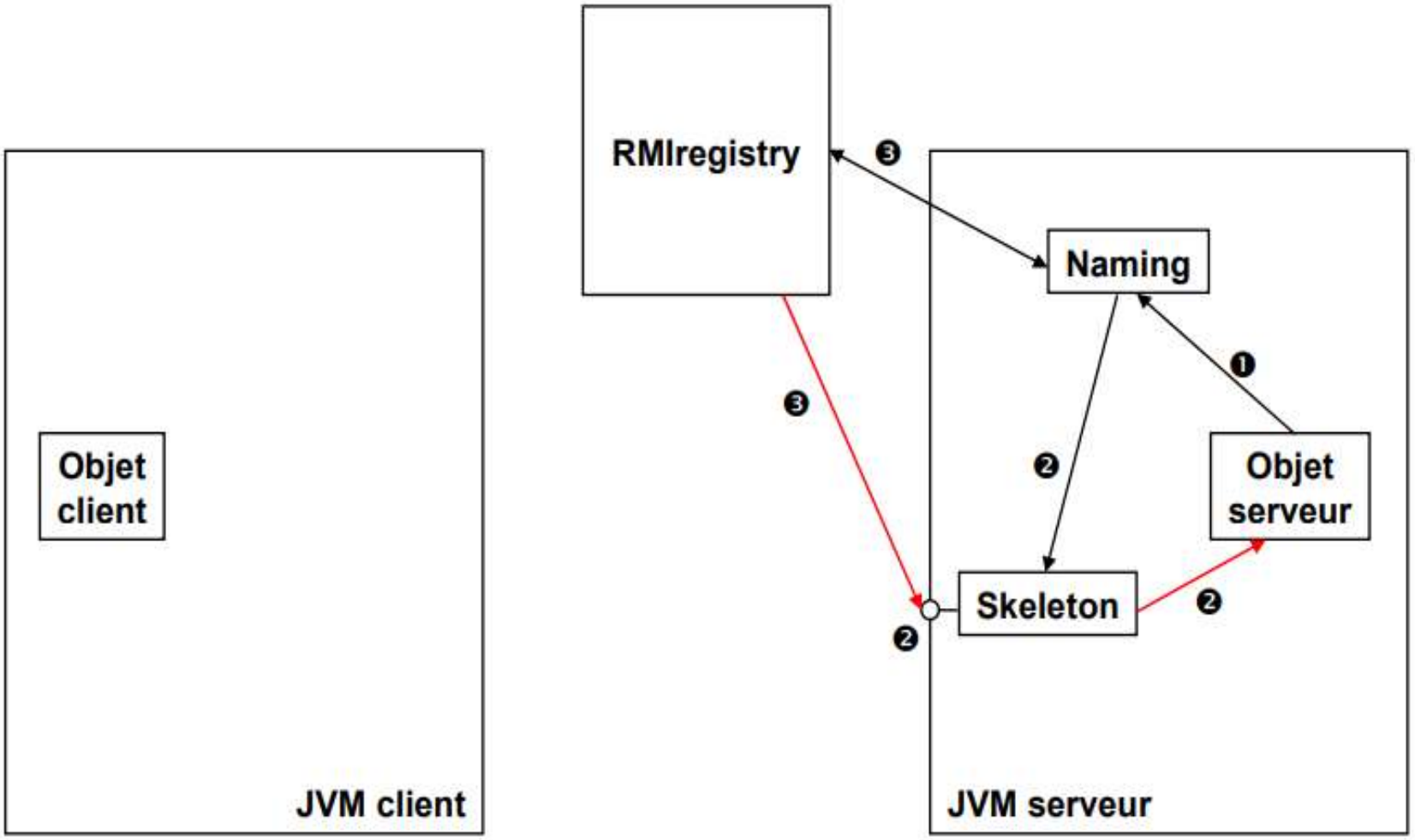
RMI: Remote Method Invocation

Architecture



RMI: Remote Method Invocation

Mode opératoire côté serveur



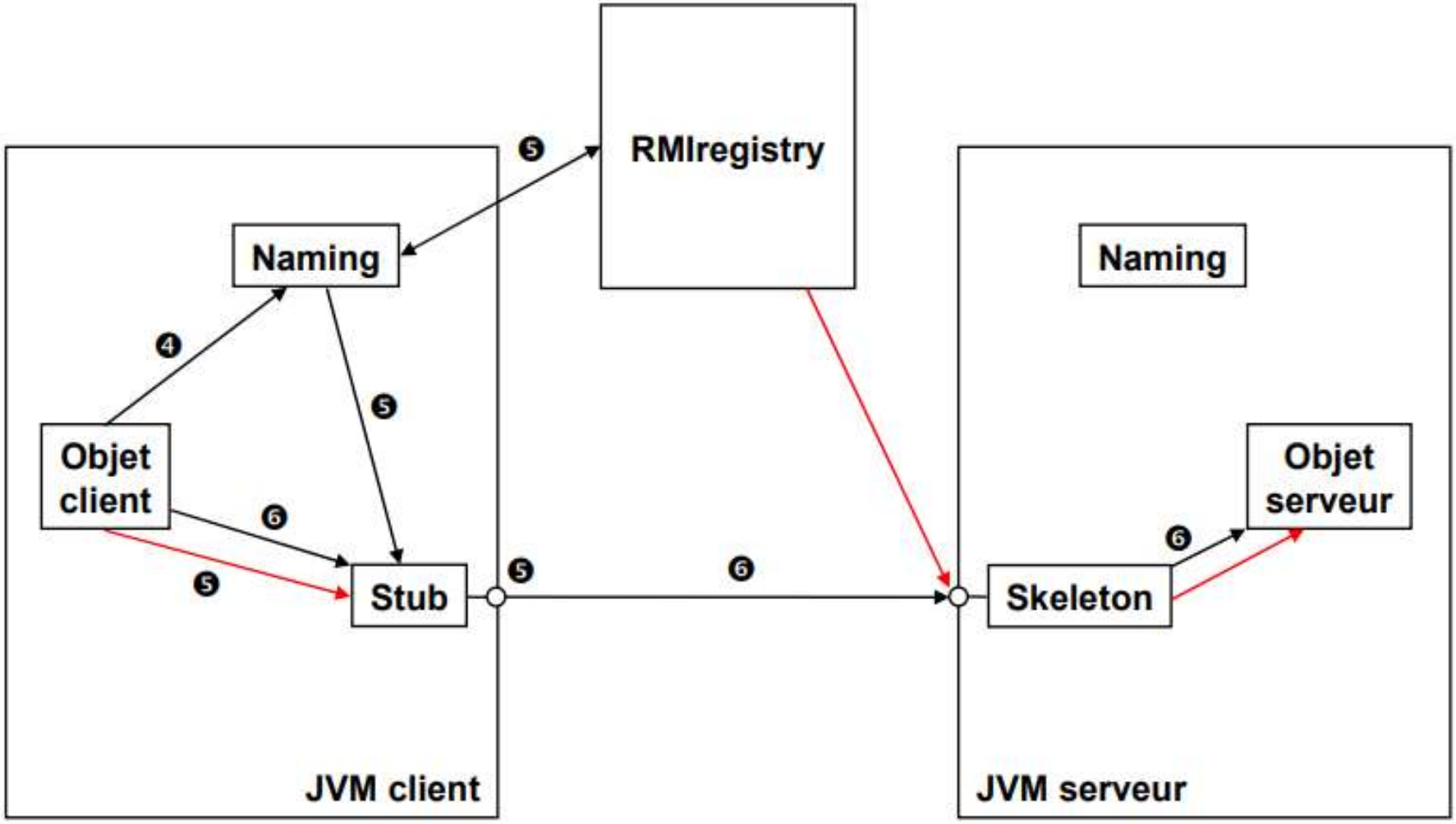
RMI: Remote Method Invocation

Mode opératoire côté serveur

1. L'objet serveur s'enregistre auprès du Naming de sa JVM (méthode rebind)
2. L'objet skeleton est créé, celui-ci crée le port de communication et maintient une référence vers l'objet serveur
3. Le Naming enregistre l'objet serveur, et le port de communication utilisé auprès du serveur de noms
4. L'objet serveur est prêt à répondre à des requêtes

RMI: Remote Method Invocation

Mode opératoire côté client



RMI: Remote Method Invocation

Mode opératoire côté client

4. L'objet client fait appel au Naming pour localiser l'objet serveur (méthode lookup)

5. Le Naming récupère les "références" vers l'objet serveur, crée l'objet Stub et rend sa référence au client

6. Le client effectue l'appel au serveur par appel à l'objet Stub

RMI: Remote Method Invocation

Stub et Skeleton

- Programmes jouant le rôle d'adaptateurs pour le transport des appels distants
  - ✓ réalisent les appels sur la couche réseau
  - ✓ pliage / dépliage des paramètres

- A une référence d'OD manipulée par un client correspond une référence Stub/Skeleton

- Les Stub/Skeleton sont générés par le compilateur d'amorces : rmic

RMI: Remote Method Invocation

Stub

Représentant local de l'OD qui implémente ses méthodes « exportées »

✓ transmet l'invocation distante à la couche inférieure Remote Reference Layer

✓réalise le pliage ("marshalling") des arguments des méthodes distantes

✓dans l'autre sens, il réalise le dépliage ("demarshalling") des valeurs de retour

✓Il utilise pour cela la sérialisation des objets

Skeleton

- ✓ Réalise le dépliage des arguments reçus par le flux de pliage
- ✓ Fait un appel à la méthode de l'objet distant
- ✓ Réalise le pliage de la valeur de retour

RMI: Remote Method Invocation

Les couches « basses » de l'architecture RMI

La couche des références distantes

✓permet l'obtention d'une référence d'objet distant à partir de la référence locale au Stub

✓ce service est assuré par le lancement du programme **rmiregister**

- Ce programme se lancer une seule fois par JVM, pour tous les objets à distribuer
- il représente un service d'annuaire pour les objets distants enregistrés

La couche de transport

✓ connecte les 2 espaces d'adressage (JVM)

✓suit les connexions en cours

✓écoute et répond aux invocations

✓construit une table des OD disponibles

✓ réalise l'aiguillage des invocations

RMI: Remote Method Invocation

Etapes de développement d'une application RMI

1. Définir une interface Java pour un OD
2. Créer et compiler une classe implémentant cette interface
3. Créer les classes Stub et Skeleton (rmic)
4. Créer et compiler une application serveur RMI
5. Démarrer rmiregister et lancer l'application serveur RMI
6. Créer, compiler et lancer un programme client accédant à des OD du serveur

Chargement dynamique

Chargement dynamique des amorces

- ✓ rappel : un objet client ne peut utiliser un objet distant qu'au travers des Stub/Skeleton
- ✓ RMI permet l'utilisation des OD dont les stub ne sont pas disponibles au moment de la compilation
- ✓ à l'exécution, RMI réclamera au serveur l'amorce cliente manquante et la téléchargera dynamiquement (byte code)

Chargement dynamique des classes

- ✓ plus généralement, le système RMI permet le chargement dynamique de classes comme les amorces, les interfaces distantes et les classes des arguments et valeurs de retour des appels distants
- ✓ c'est un chargeur de classes spécial RMI qui s'en charge: **java.rmi.server.RMIClassLoade**r

Chargement dynamique

Côté serveur

- ✓ Le rmiregistry réclame que le stub soit classé dans le classe path
- ✓ Deux façon pour le faire:
  - ➢ Soit le placer dans le classe path
  - ➢ Soit spécifier à la JVM lors du lancement du serveur où aller les chercher pour le binding

    java -Djava.rmi.server.codebase=URL

    exemple java -Djava.rmi.server.codebase=file:// =http://

Coté client

- ✓ Une fois le stub localisé par **lookup** c'est le **RMISecurityManager** qui se charge de chargement dynamique

RMI: Remote Method Invocation

La sécurité

- ✓ RMI n'autorise pas le téléchargement dynamique de classes (avec RMICLassLoader) si l'application (ou l'applet) cliente n'utilise pas de SecurityManager pour les vérifier.
- ✓ dans ce cas, seules les classes situées dans le CLASSPATH peuvent être récupérées
- ✓ le gestionnaire de sécurité par défaut pour RMI est **java.rmi.RMISecurityManager**
- ✓ il doit être absolument utilisé (System.setSecurity()) pour les applications standalone
- ✓ pas de problème pour les applets, c'est l'AppletSecurityManager (par défaut) qui s'en charge

RMISecurityManager

- ✓ vérifie la définition des classes et autorise seulement les passages des arguments et des valeurs de retour des méthodes distantes
- ✓ ne prend pas en compte les signatures éventuelles des classes

RMI: Remote Method Invocation

Aspects avancées : clonage & égalité

- ✓ Passer des objets distant (OD) en paramètre (E/S). C'est possible
- ✓ Ce que vous envoyer ou recevez c'est pas l'objet mais un stub de l'objet (sans passer par le Naming) et donc il faut faire attention à deux choses:
- ✓ Seul les méthodes hérité des interface qui hérite Remote sont accessible dans le stub.
- ✓ Les méthodes hérités de Object et qui peuvent être surchargé dans le l'OD n'ont pas la même signification sur le stub de l'objet.

  Clone()

  Equals() // est celle de RemoteObject signifie égalité de référence sur le serveur entre deux objet stub

RMI: Remote Method Invocation

Aspects avancées : sécurité

Exception: *java.security.AccessControlException: access denied (java.net.SocketPermission 127.0.0.1:1099 connect,resolve)*
C'est une Exception très souvent rencontré par les programmeur. La cause???

- ✓ RMISecurityManager: qui interdit partigement par defaut toutes création de socket sur l'ensemble des port.
- ✓ Pour éviter que les object envoyer en paramètres ou en résultat ne puissent se connecter ....

Pour associer une politique de sécurité à un code JAVA, il faut construire

- ✓ un objet instance de SecurityManager
- ✓ surcharger les fonctions check... dont on veut changer la politique de sécurité
- ✓ invoquer la méthode de la classe System, setSecurityManager, en lui passant cet objet créé plus haut.

Aspects avancées : sécurité

Autre solution

Consiste à passer par fichier de politique de sécurité et lancée l'application en lui précisant le fichier de sécurité à considéré.

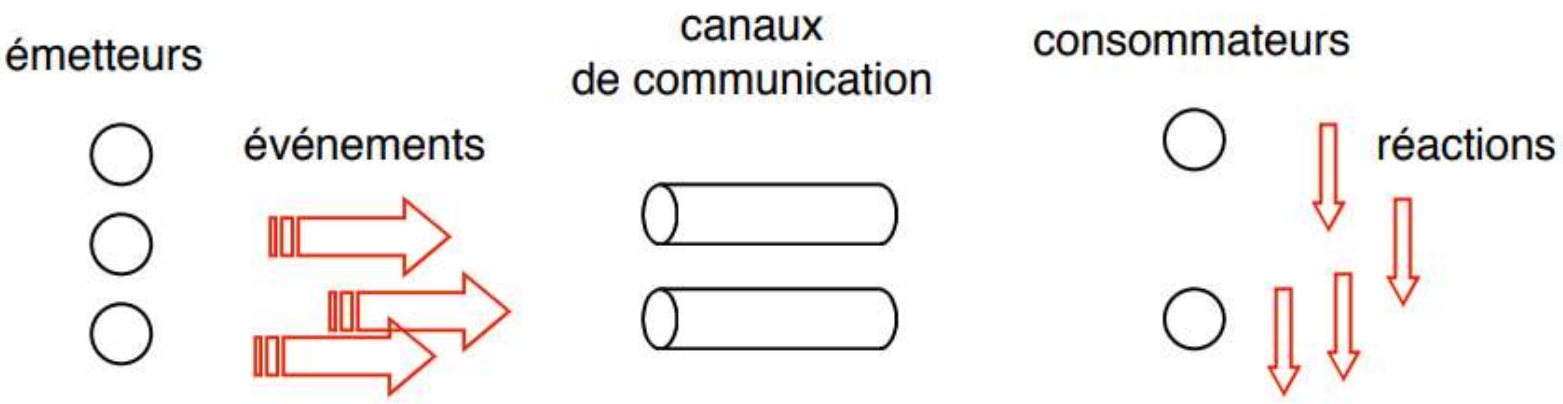
- ✓ java -Djava.security.policy=NomFichier.Policy
- ✓ Pour données l'ensemble de droit a vos application serveur et client il faut lui donner les droit nécessaire.
  - grant {permission java.security.AllPermission;};
  - grant codeBase "file:/home/goubault/-" {permission java.net.SocketPermission "129.104.254.54:1024-", "connect, accept";}

Communication par événements

Communication par événements

Principes de base

- ✓ événement = changement d'état survenant de manière asynchrone (par rapport à ses "consommateurs")
- ✓ réaction = exécution d'une séquence prédéfinie liée à l'événement
- ✓ association dynamique événement-réaction
- ✓ communication anonyme : indépendance entre émetteur et "consommateurs" d'un événement



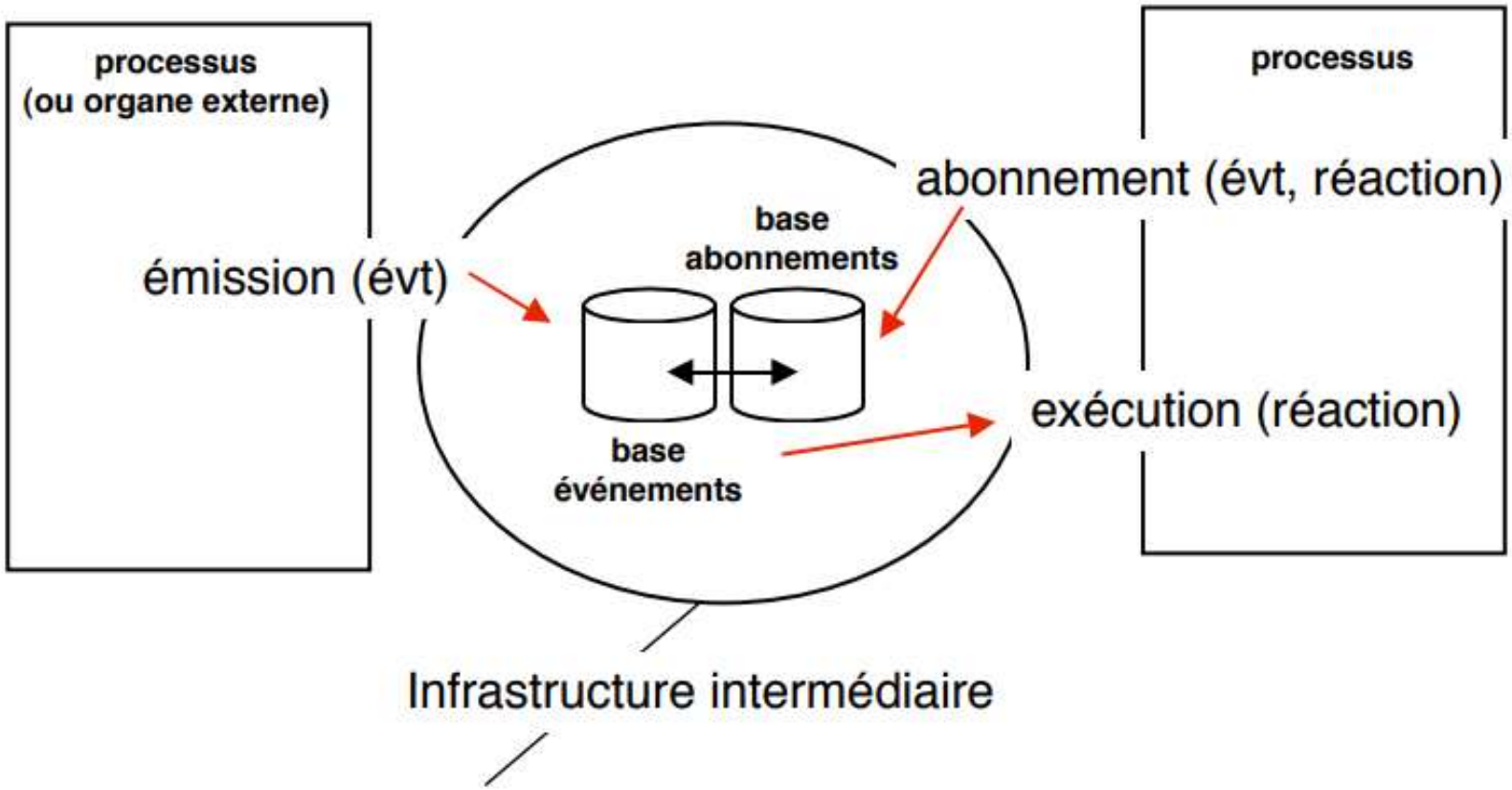
Communication par événements

Interface

Pas d'interface normalisée, donc on donne une idée des opérations usuelles d'un service d'événements (exemples concrets plus loin)

1. abonnement (sujet1, sujet2, ...)
2. désabonnement (sujet1, sujet2, ...)
3. associer (sujet1, réaction1 [paramètres])
4. dissocier (sujet1)
5. déclaration de types d'événements : définition des champs (sujet, attributs, …)
6. créer (événement [paramètres])
7. envoyer (événement)

Communication par événements



Communication par flux
(Stream oriented communication)

Communication par flux