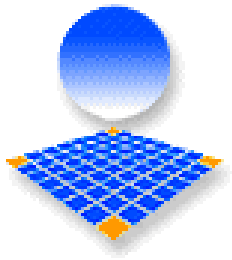
Java Remote Method Invocation (RMI)

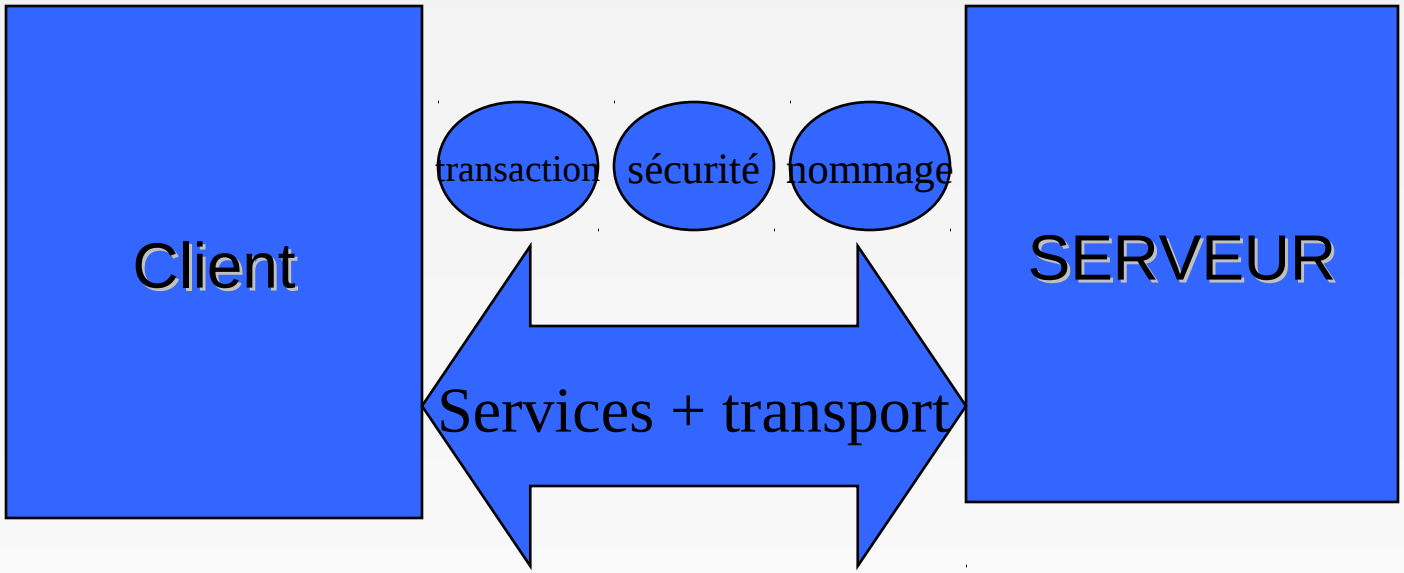
Clémentine Nebut

LIRMM

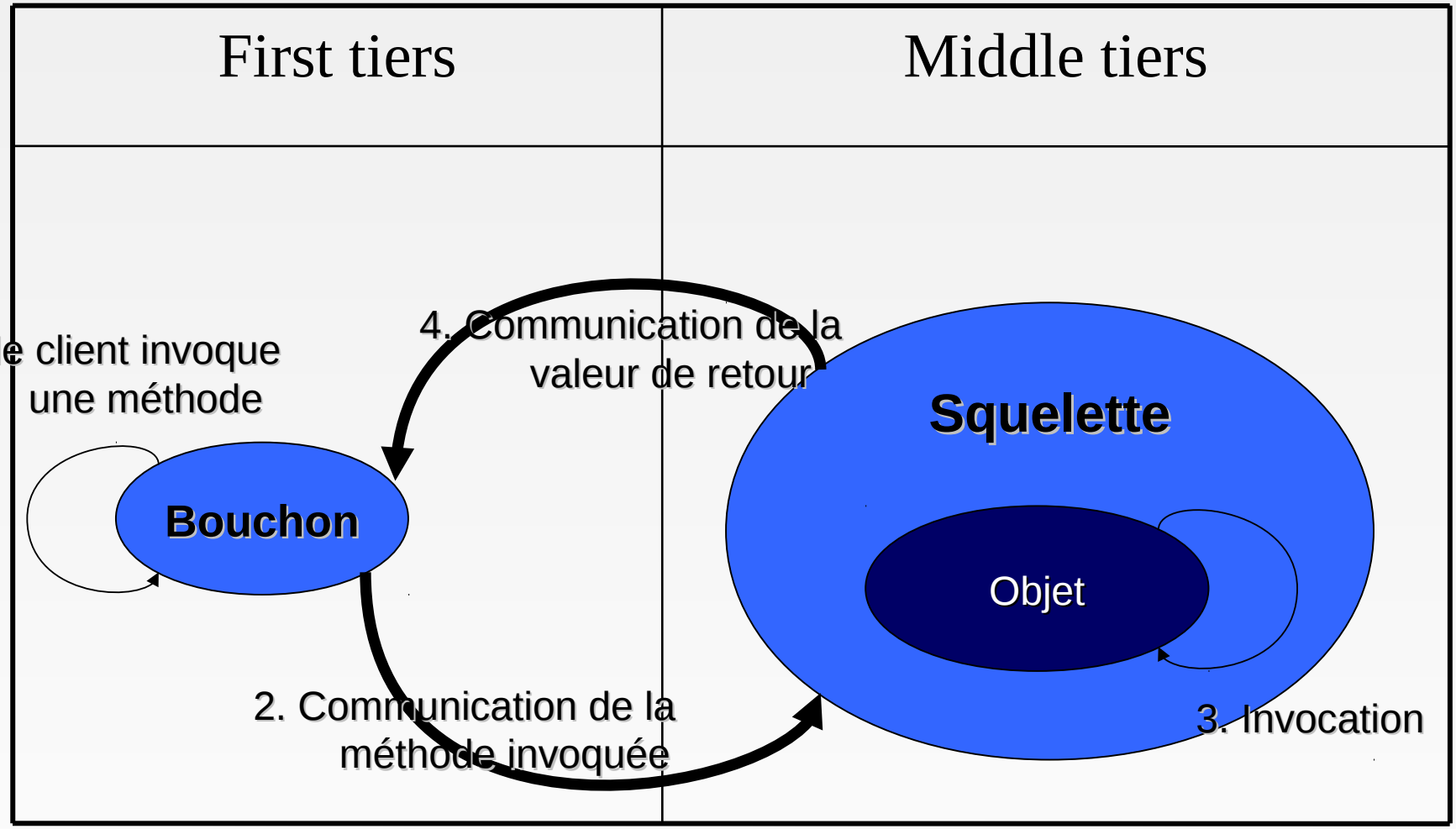
Clementine.nebut@lirmm.fr



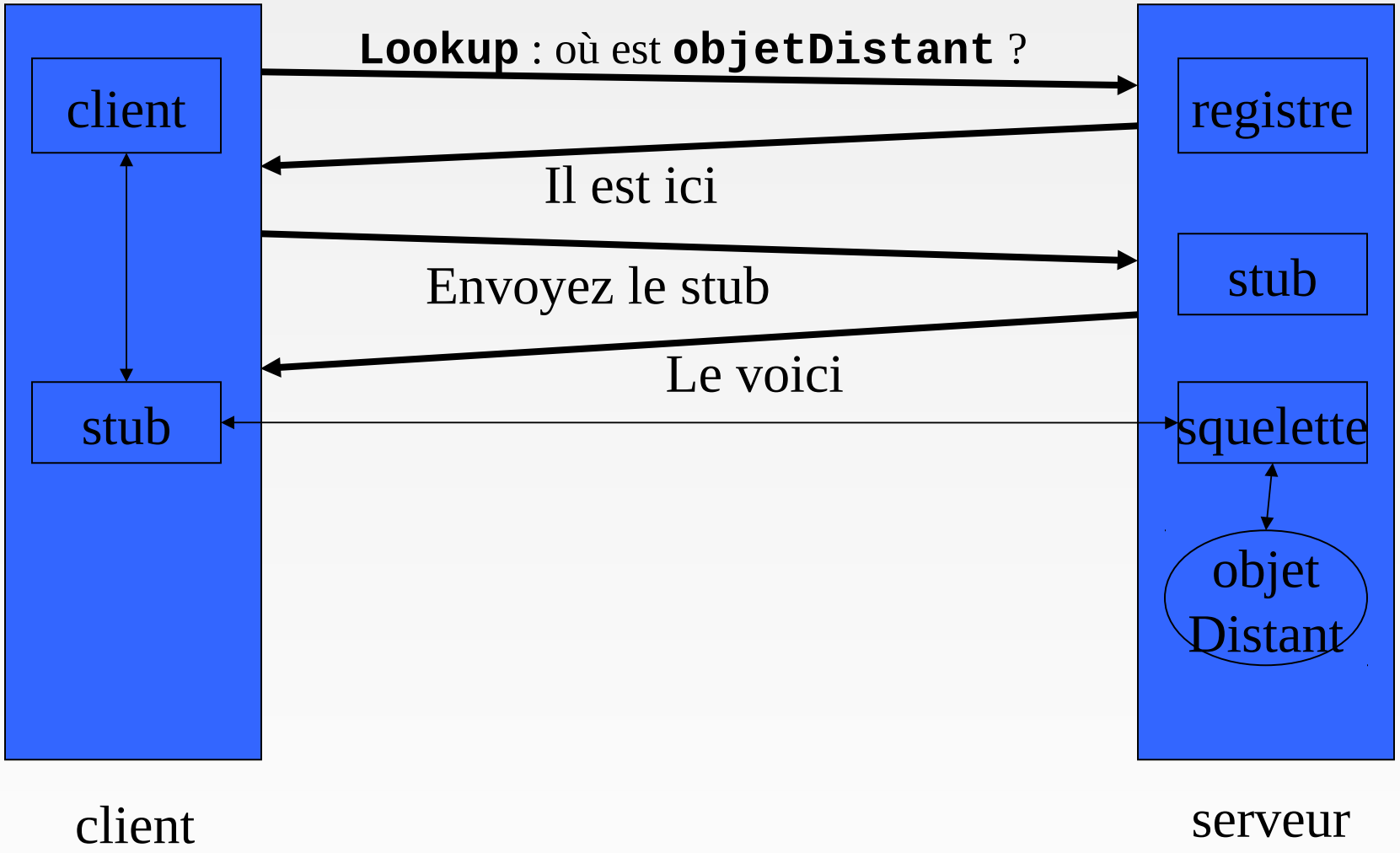
Quelle architecture pour les protocoles à objets distribués ?



Les protocoles à objets distribués : architecture



Interactions



Les protocoles à objets distribués : architecture

- Le bouchon (stub)
  - Implémente une interface identique à celle de l'objet, mais ne contient pas de logique métier
  - Implémente les opérations réseau à effectuer pour transmettre la requête à l'objet
- Le squelette (skeleton)
  - analyse les messages reçus en provenance d'un bouchon
  - Invoque la méthode métier correspondante sur l'objet
- Le registre : annuaire

Rôle du client

- Invoquer les services dont il a besoin par envoi de requêtes
- Accès à l’objet destinataire par une référence à son implémentation par l’interface

Rôle de l’infrastructure

- Administre les implémentations, la création et la destruction d’objets
- Réceptionne les requêtes, localise le serveur, vérifie son état et celui du destinataire
- Active au besoin le serveur, lui envoie les données de la requête
- Ramène les résultats au client
- Doit être informée de l’arrêt d’un serveur
- Doit gérer la persistance

Rôle du serveur

- Administrer un flot de requêtes pour un ou plusieurs objets dont il a la responsabilité
- Ordonnancer la séquence des opérations de réponses à une requête

Rôle du serveur d'objets

- Active si besoin l'objet destinataire
- Recherche et exécute la méthode
- Passe le résultat à l'infrastructure
- Plusieurs requêtes peuvent arriver simultanément
- Arrêt du serveur : désactiver tous les objets et enregistrer leur état

Les différentes technologies

- Corba
- DCOM
  - multi langages, plateforme win32 essentiellement, propriétaire
- .net remoting
- RMI / EJB
- Web services SOAP

L’ancêtre commun : le RPC

- Remote Procedure Call
- Appel de procédures à distance entre un client et un serveur
  - le client appelle une procédure
  - que le serveur exécute, et en renvoie le résultat
- Outil rpcgen
  - génère la souche d’invocation et le squelette du serveur
  - la souche et le squelette ouvrent une socket et encodent/décodent les paramètres
- Couche de représentation XDR (eXchange Data Representation)

Limitations du RPC

- Pas d'objets
  - Paramètres et valeurs de retour sont des types primitifs
  - Programmation procédurale
  - Pas de "référence distante"

Plan

- Introduction
- Principes de base
- Développement et exécution
- Passage de paramètres
- Les services
  - Le service de nommage
  - L’activation des objets
  - La couche de transport
  - Le ramasse-miettes

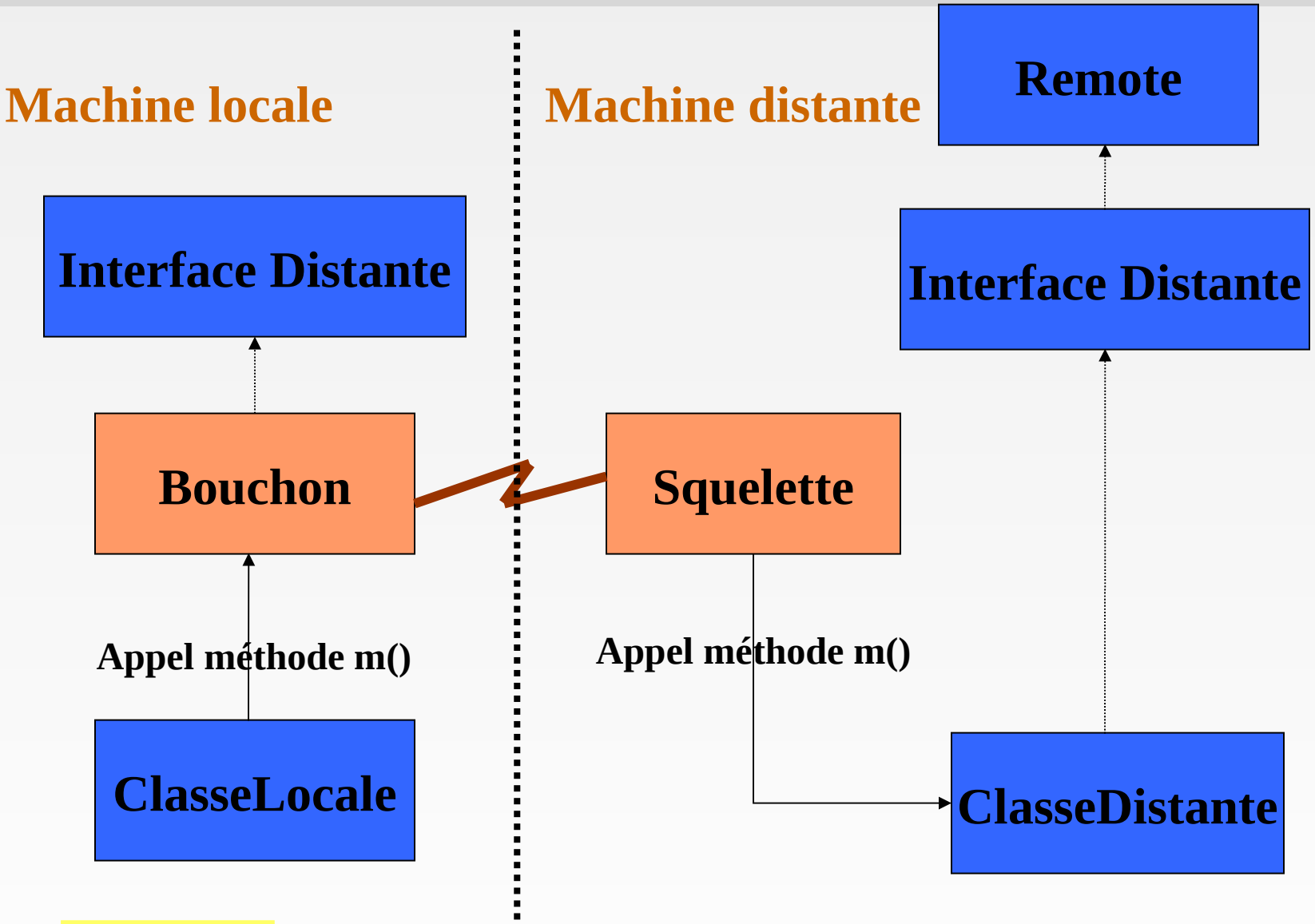
Objectif

- Invocation de méthodes sur des objets distribués
- En cachant au programmeur les détails de connexion et de transport
- Interagir avec un objet distant comme s'il était local i.e. dans la même JVM

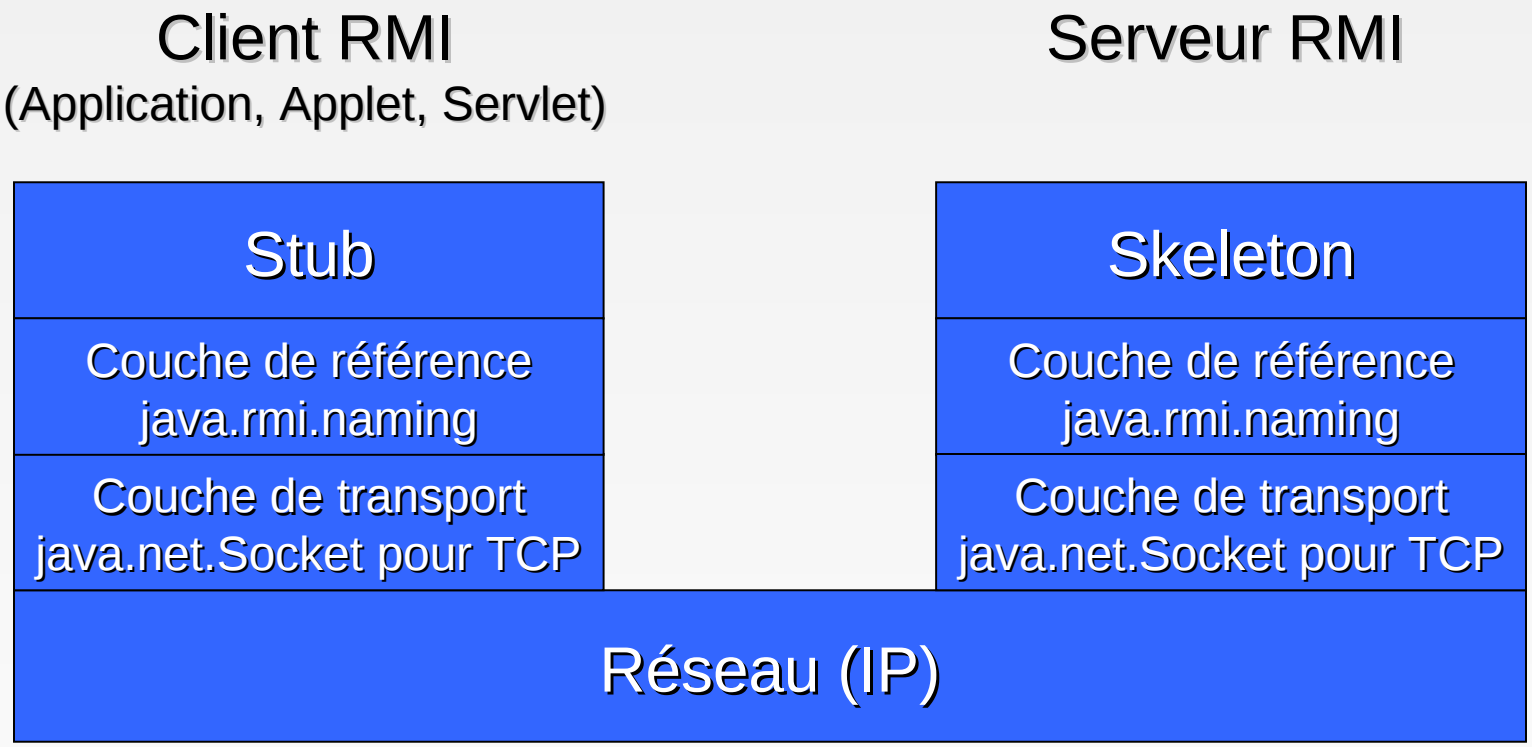
Caractéristiques

- Transmission d’objets par copie ou par référence
- Utilisation de stubs et de squelettes pour masquer le codage et désencodage des données
- Outils pour la génération de stubs et de squelettes, l’activation, l’enregistrement …
- Multi-plateforme

Schéma général



L'architecture



Côté client : le Stub

- Stub pour un objet distant (remote object)
  - joue le rôle d'un proxy (représentant local) pour l'objet distant
  - implémente les mêmes interfaces distantes que celles de l'objet distant
  - masque à l'utilisateur la sérialisation des paramètres (ou un passage de stub) et la communication réseau

Le stub –suite-

- Quand une méthode est appelée sur un stub, celui ci :
  - initie une connexion avec la JVM distante
  - lui transmet les paramètres d'appel
  - attend le résultat
  - récupère la valeur ou l'exception de retour
  - renvoie le résultat à l'appelant

Côté serveur : le squelette

- Responsable de transmettre l'appel à l'implantation du réel objet distant
- Quand un squelette reçoit une invocation de méthode :
  - il lit les paramètres
  - il invoque la méthode sur l'implémentation de l'objet distant
  - il transmet le résultat à l'appelant

Couche des références distantes et de transport

- Couche des références distantes
  - permet l'association stub/objet distant
  - processus tiers : rmiregistry
- Couche de transport
  - écoute les appels entrants
  - gestion des connexions avec les sites distants
  - possibilité d'utiliser différentes classes pour le transport

5 packages

- Java.rmi : accès aux OD
- Java.rmi.server : création d'OD
- Java.rmi.registry : localisation et nommage d'OD
- Java.rmi.dgc : ramasse miettes d'OD
- Java.rmi.activation : activation d'OD

Plan

- Introduction
- Principes de base
- Développement et exécution
- Passage de paramètres
- Les services
  - Le service de nommage
  - L'activation des objets
  - La couche de transport
  - Le ramasse-miettes

RMI pas à pas

- Spécifier et écrire l'interface de l'objet distant
- Ecrire l'implémentation de cette interface
- Générer les stubs et squelettes (versions <1.5)
- Ecrire le serveur (instancie l'objet, exporte son stub, attend les requètes via le squelette)
- Ecrire le client (réclame l'objet distant, importe le stub, invoque une méthode sur l'objet distant via le stub)

Spécifier une interface distante

- Doit être publique
- Hérite de l'interface java.rmi.Remote
- Chaque méthode déclare une java.rmi.RemoteException dans sa clause throws

Implémenter une interface distante

- Implémente une ou plusieurs interfaces distantes
- Hérite de UnicastRemoteObject
- Implémente toutes les méthodes distantes
- Définit le constructeur d’objets distants

Implémentation du serveur d'objets distants

- Création et installation du gestionnaire de sécurité
- Création d'une ou plusieurs instances d'objets distants
- Enregistre au moins un objet distant dans le registre d'objets distants RMI
  - java.rmi.Naming.bind
  - Le registre d'objet peut être créé dans la JVM du serveur (utilisation de la classe LocateRegistry)

Génération des stubs et squelettes

- Appel de l'outil rmic (versions <1.5)
  - génère la classe stub
  - génère la classe squelette
  - à partir de l'implémentation (le .class)
- Génération dynamique (>=1.5)
  - quand un OD est enregistré
  - incompatible avec clients <1.5
- Squelettes pas requis dès Java 2
  - code générique utilisé à la place
  - pas générés par défaut depuis java 1.5

Implémentation d’un client

- Demande un stub auprès de rmiregistry
  - java.rmi.lookup
- Si la classe du stub n’est pas connue, récupération auprès de java.rmi.server.codebase
  - Le téléchargement du code fonctionne fonctionne sur le même mécanisme que les applets
- Invoque des méthodes sur le stub

Côté serveur

- rmiregistry
  - serveur de liaison, sur le port 1099 par défaut
  - expose un objet distant serveur de liaisons (de noms)
  - fait la correspondance entre nom et instance de stub enregistré par le serveur avec Naming.bind()
- Il faut prévoir une procédure d'arrêt avec arrêt des objets distants :
  - public static void Naming.unbind(String name)
  - public static boolean UnicastRemoteObject.unexportObject(Remote,boolean force)

Exécution côté serveur

- Le stub doit être dans le CLASSPATH ou chargeable via FS ou HTTP
- Un fichier .policy autorise l'usage du accept et du connect sur les sockets
  - java –Djava.security.policy=./hello.policy helloServer hostreg:1099
  - Possibilité d'utiliser une ligne comme : System.setProperty("java.security.policy", "hello.policy");
  - Exemple de fichier (ici : ./hello.policy)

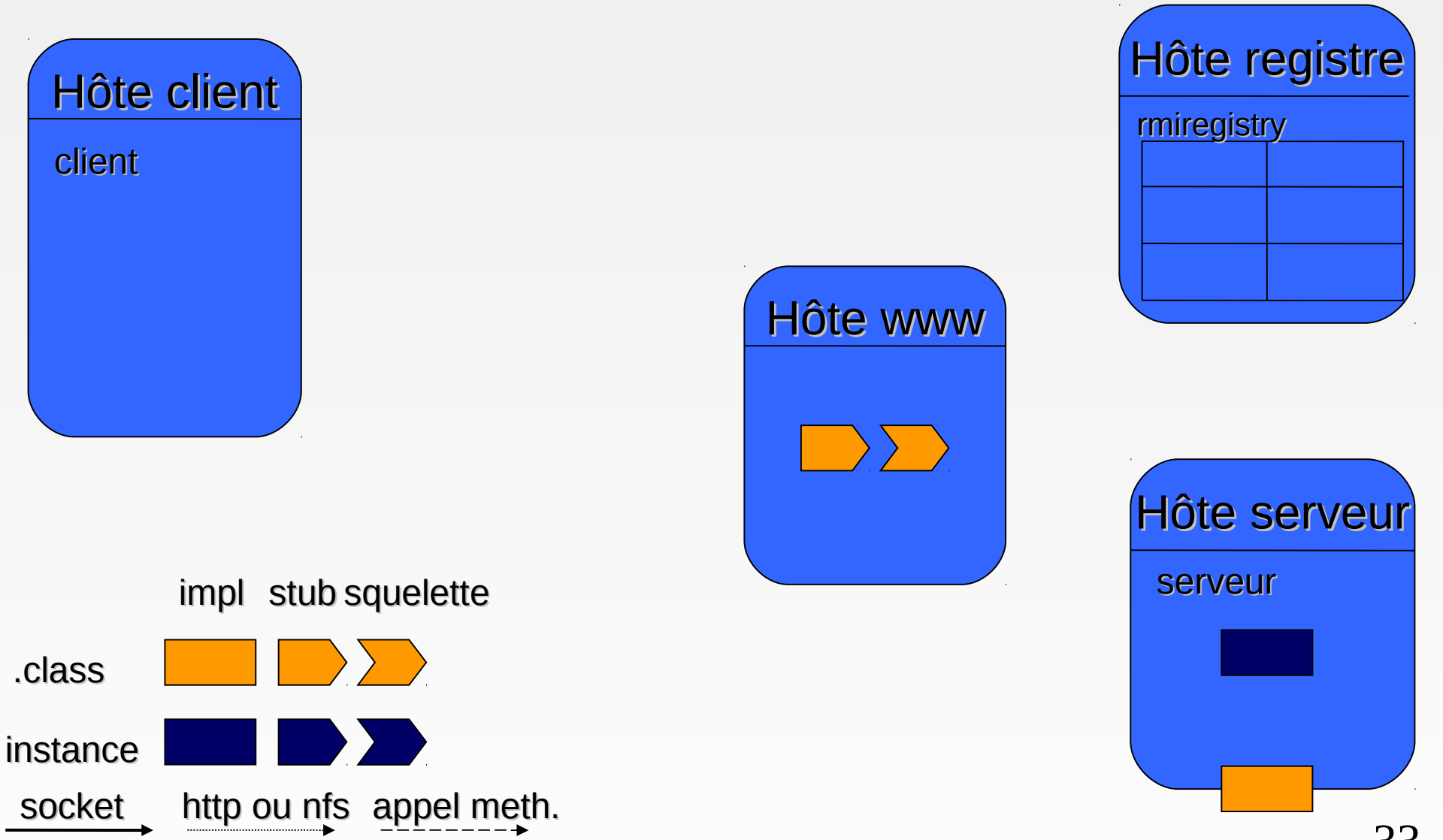
```
grant{
permission java.net.SocketPermission "*:1024-65535","connect,accept";
permission java.net.SocketPermission ":80","connect";
// permission java.security.AllPermission;
}
```

Exécution côté client

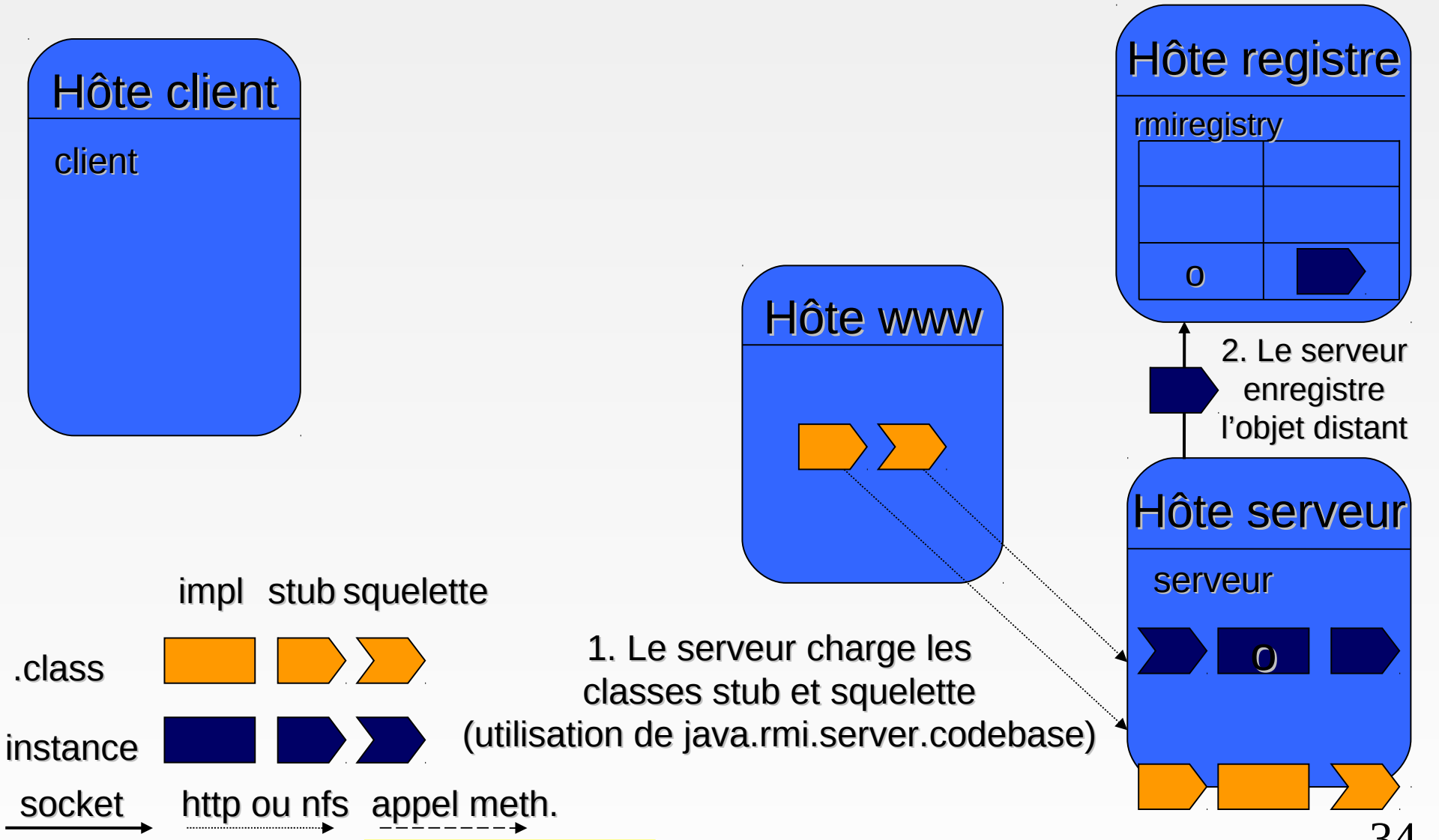
- Le stub doit être dans le CLASSPATH ou chargeable via FS ou HTTP
- Un fichier .policy autorise l'usage du connect sur les sockets
  - java –Djava.security.policy=./hello.policy helloClient hostreg:1099
  - ./hello.policy

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

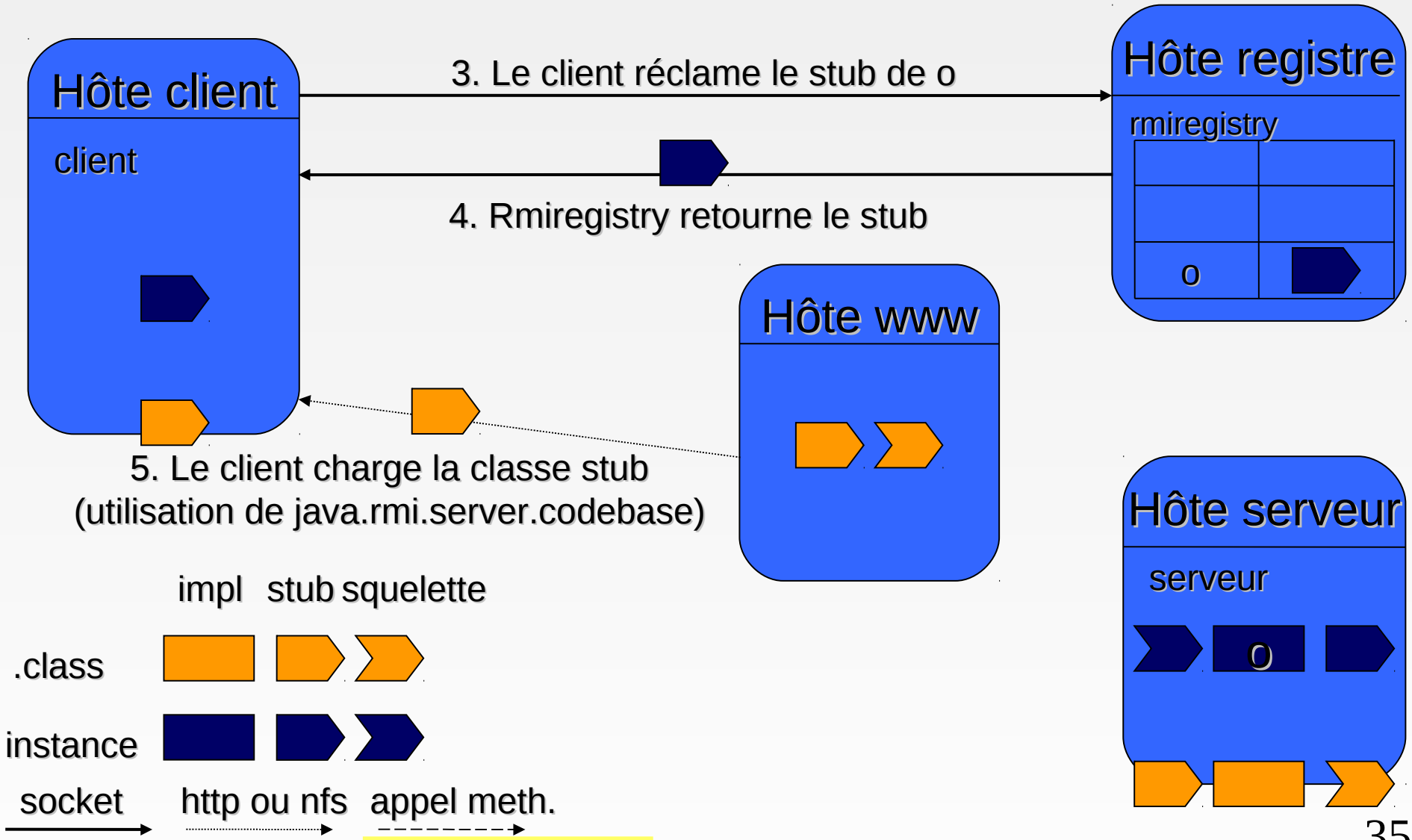
Exécution : illustration



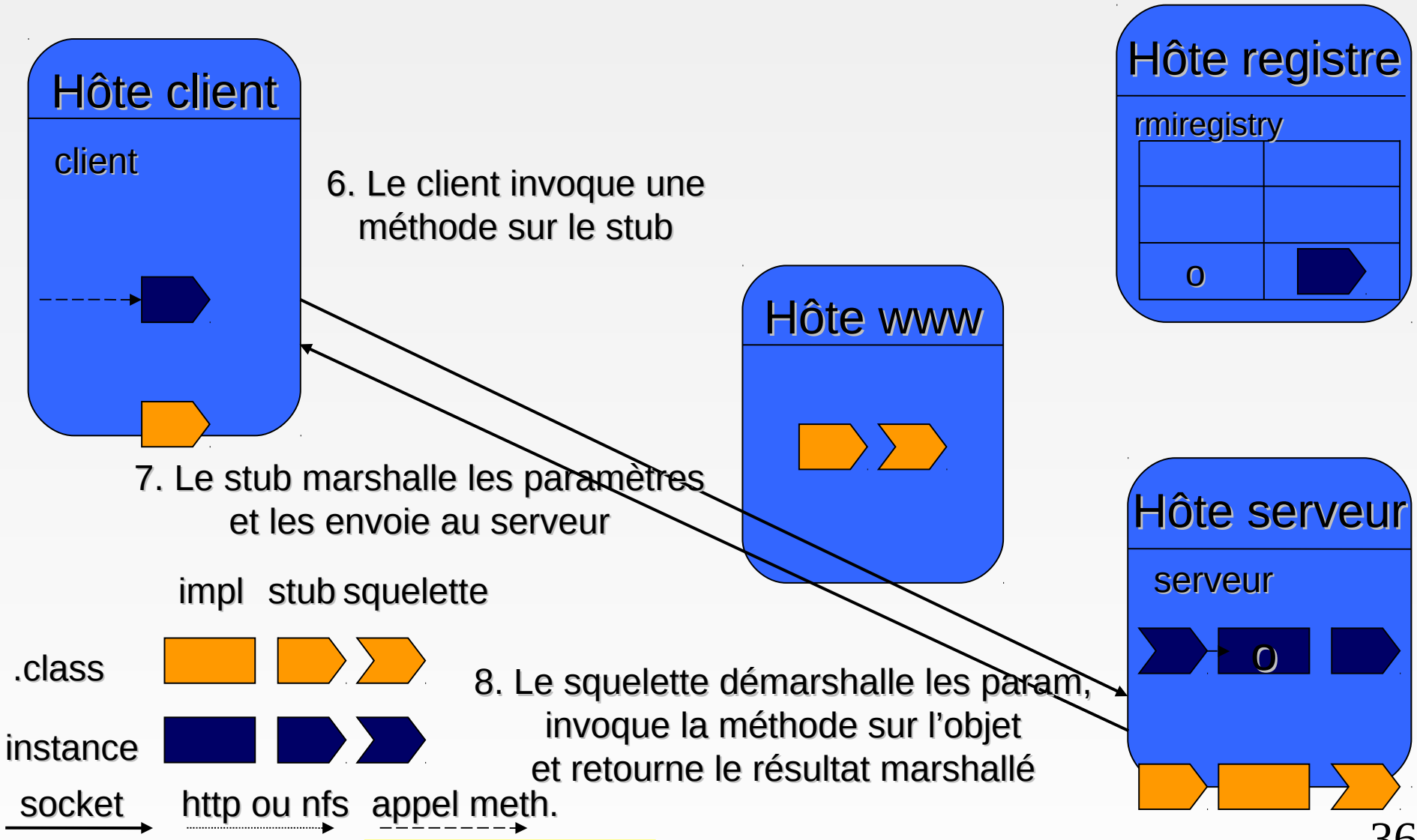
Enregistrement de l'objet



Récupération du stub



Invocation d'une méthode



Plan

- Introduction
- Principes de base
- Développement et exécution
- Passage de paramètres
- Les services
  - Le service de nommage
  - L'activation des objets
  - La couche de transport
  - Le ramasse-miettes

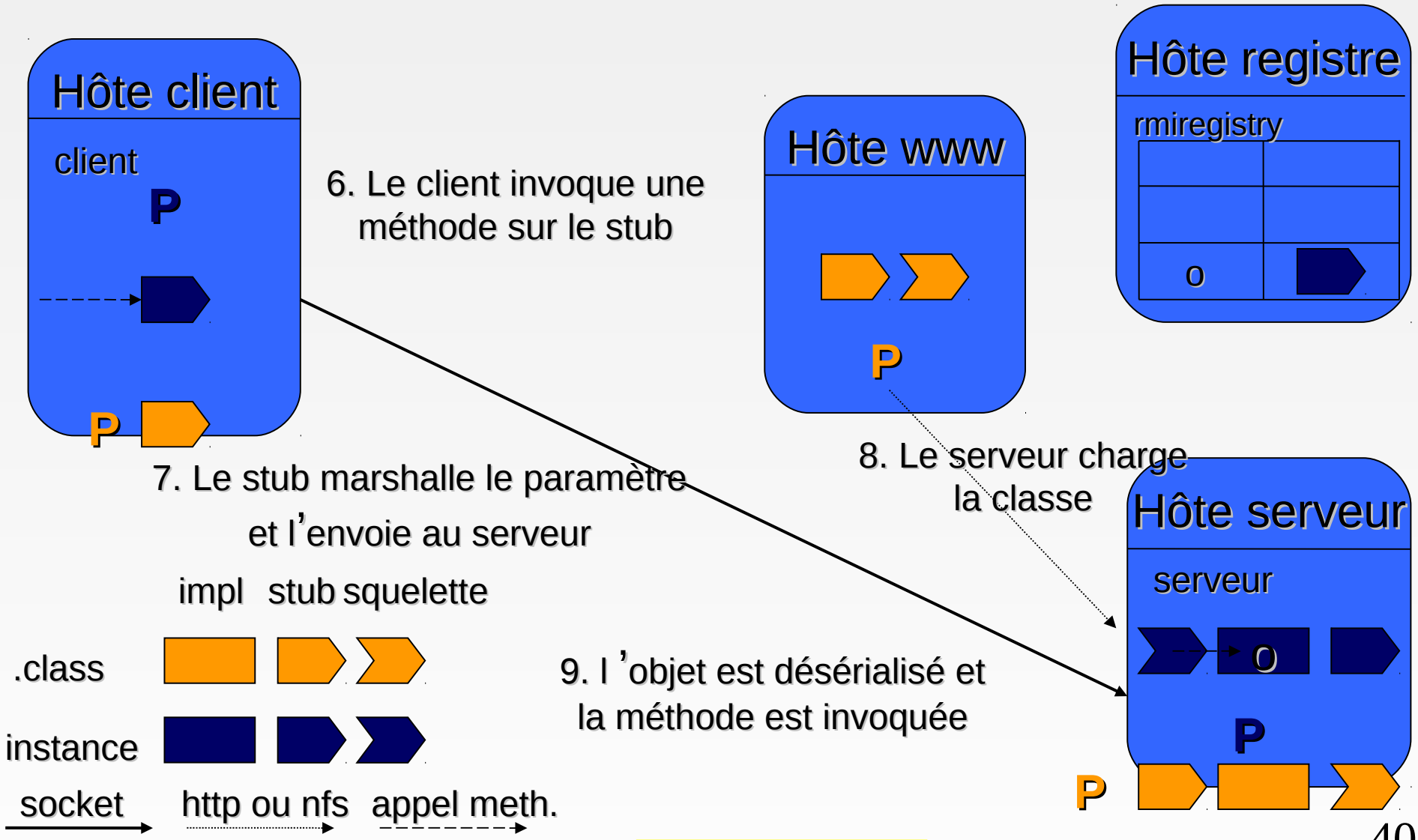
Le passage de paramètres

- Les paramètres des méthodes invoquées sur un objet distant sont soit :
  - une valeur de type primitif
    - passage par valeur
  - un objet d'une classe sérialisable
    - l'objet est sérialisé et envoyé à l'objet distant qui le désérialise avant de l'utiliser
  - un objet d'une classe qui implémente l'interface Remote
    - c'est l'objet stub qui est sérialisé et envoyé à l'objet distant
- Sinon une exception est levée

Passage d’un paramètre de classe inconnue du serveur

- Le client invoque une méthode avec un paramètre inconnu du serveur
- Le Stub sérialise l’instance du paramètre et l’envoie au serveur
- Le serveur charge la classe d’après java.rmi.server.codebase
- L’objet est désérialisé et la méthode est invoquée
- Le squelette retourne le résultat au client

Passage de paramètre inconnu



Retour sur le transport du code

- Dans certains cas, on a besoin de télécharger du code du serveur vers le client (ou l'inverse)
- Mécanisme de « classpath distribué » : codebase
  - Le classpath java : où trouver les classes en local
  - Le codebase : où trouver les classes distantes
- Property java.rmi.server.codebase positionnée :
  - En ligne de commande :

```
java -Djava.rmi.server.codebase=http://mycomputer/arch.jar
```

  - Ou dans le code :

```
System.setProperty( "java.rmi.server.codebase", « http://mycomputer/arch.jar » );
```

Retour sur la sécurité

- Créer un gestionnaire de sécurité
  - System.setSecurityManager (new RMISecurityManager()) ;
- La politique de sécurité
  - Dans un fichier à part
  - Localisée via la property java.security.policy
  -

http://docs.oracle.com/javase/1.4.2/docs/guide/security/permissions.html

Retour sur la sécurité : fichier de politique de sécurité

grant [signedBy "signer_names" ,]

[codeBase "URL"] // ending with / includes all class files in the specified directory; ending with /* includes all class and jar files in the directory; ending with /- includes all class and jar files in the directory tree rooted at the specified dir.

[principal principal_class_name "principal_name",]+

{

[permission permission_class_name ["name",] ["action",]

[signedBy "signer_names"] ;]+

...

};

Exemples :

grant codeBase "file:/home/jones/src/" {

permission java.security.AllPermission;

};

grant codeBase "file:.${/}bin/" { permission java.io.FilePermission "..${/}*", "read" ; };

donne l'autorisation aux fichiers class chargés depuis le sous-répertoire bin à lire les fichiers présents dans le répertoire parent. ${/} permet d'avoir un séparateur de répertoire portable.

Plan

- Introduction
- Principes de base
- Développement et exécution
- Passage de paramètres
- Les services
  - Le service de nommage
  - L’activation des objets
  - La couche de transport
  - Le ramasse-miettes

La classe Naming

- Encapsule le dialogue avec plusieurs objets serveur de liaison
- Méthodes statiques
  - bind(String url, Remote r) – rebind(String url, Remote r) – unbind(String url)
  - Remote lookup(String url)
  - String[] list()

La couche de transport

- Par défaut, TCP est utilisé par la couche de transport RMISocketFactory
  - Dans le cas de firewall/proxies, invocation des méthodes en utilisant un POST HTTP
    - la propriété java.rmi.server.disableHttp=true désactive le tunneling HTTP
- Mais la couche de transport est personnalisable
  - utilisation d'autres classes que Socket et SocketServer basées sur
    - TCP
    - UDP

Personnaliser la couche de transport

- Ecrire deux sous classes de java.rmi.RMIClientSocketFactory et java.rmi.RMIServerSocketFactory
  - qui utilisent 2 autres classes de transport que Socket et SocketServer
  - par exemple CompressionSocket et CompressionServerSocket
- Spécifier les factories dans le constructeur de l'objet distant qui hérite de la classe UnicastRemoteObject

RMI et SSL

- Créer les classes SSLClientSocketFactory et SSLServerSocketFactory
  - qui utilisent SSLSocket et SSLServerSocket
- http://java.sun.com/j2se/1.4.2/docs/guide/rmi/socketfactory/index.html

L’activation d’objets distants

- En JDK1.1, tous les objets distants étaient actifs au démarrage du serveur RMI
- JDK 1.2 introduit le démon rmid
  - Rmid démarre une JVM qui sert l’objet distant au moment de l’invocation d’une méthode

Le ramasse miettes distribué dgc

- Ramasse les objets distants qui ne sont plus référencés
- Basé sur le comptage de références
- Interagit avec les GCs locaux de toutes les JVMs
  - maintient des *weak references* pour éviter le ramassage par le GC local

Conclusions sur RMI

- Protocole de communication entre objets java distants
  - masque à l'utilisateur la couche de transport
  - facile d'utilisation et de mise en place
  - multi plateformes mais mono-langage