Communication dans les systèmes distribués

Plan

- Couches de protocoles
- Modèle client-serveur
- Appels de procédures à distance RPC
- La communication de groupe
- Les middlewares
- Objets distribués

Couches de protocoles

Modèle en couches OSI

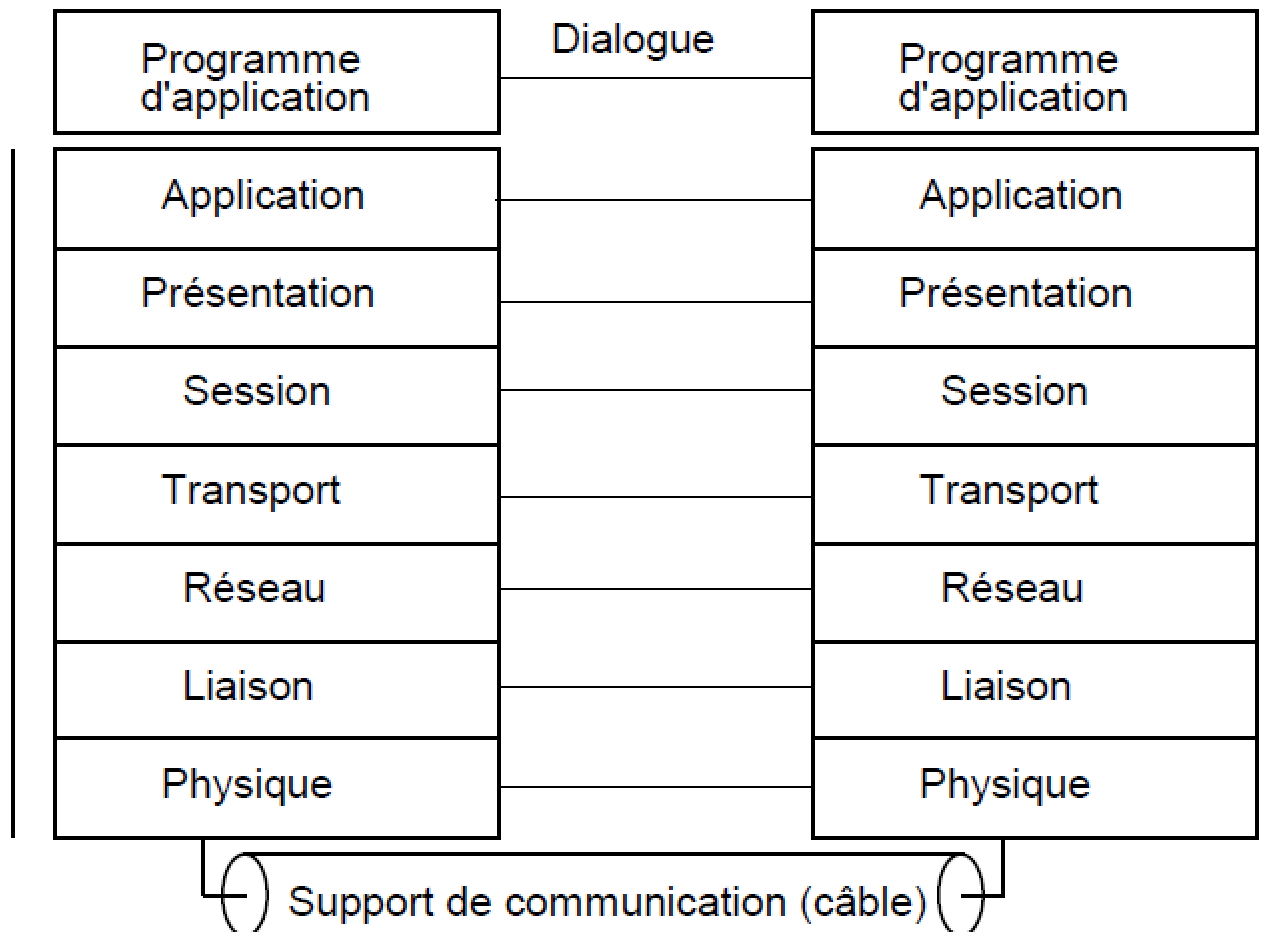
- Une couche à un rôle particulier
- Une couche d'une entité communique avec une couche de même niveau d'une autre entité en respectant un certain protocole de communication
- Pour communiquer avec une autre entité, une couche utilise les services de sa couche locale inférieure
- Données échangées entre 2 couches : trames ou paquets
    - ✓ Données structurées
    - ✓ Taille bornée
    - ✓ Deux parties:

-Données de la couche supérieure à transmettre

-Données de contrôle de la communication entre couches

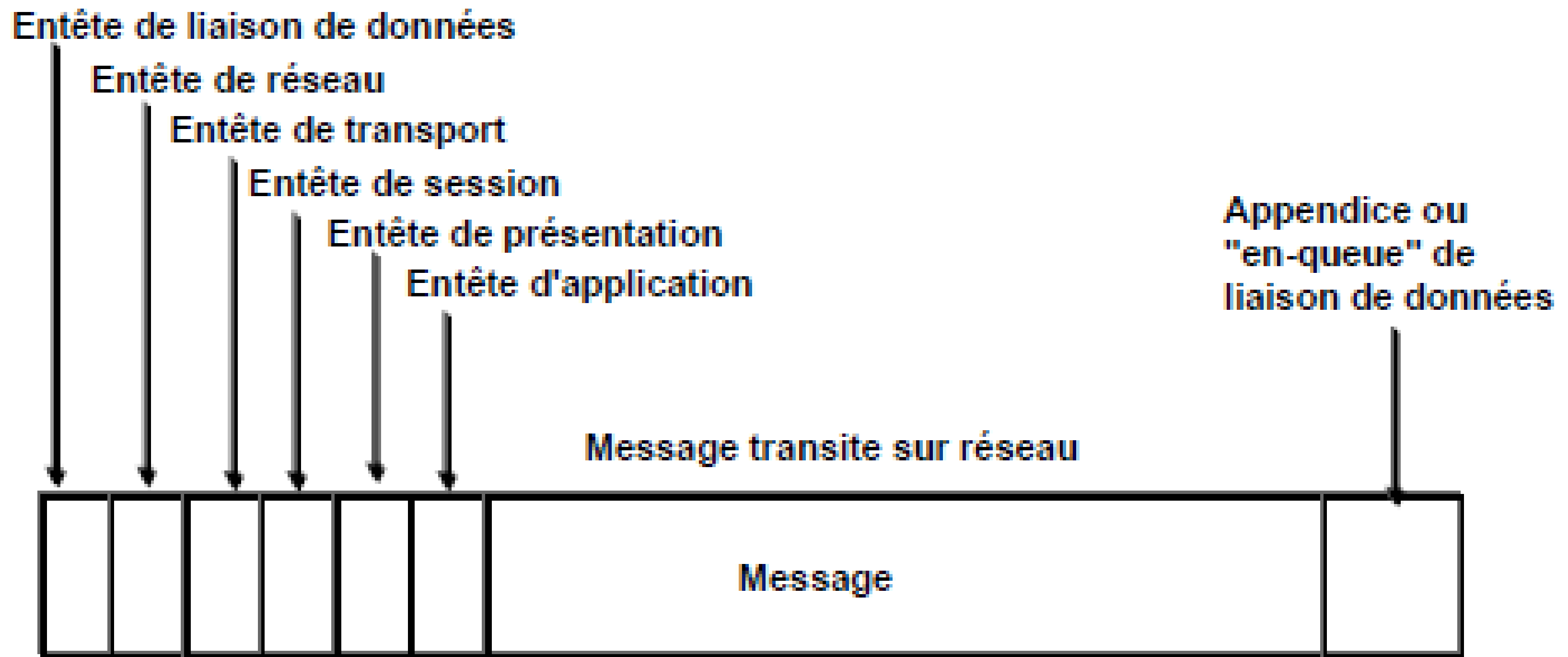
Rappels des protocoles OSI

- **Modèle en couches OSI**

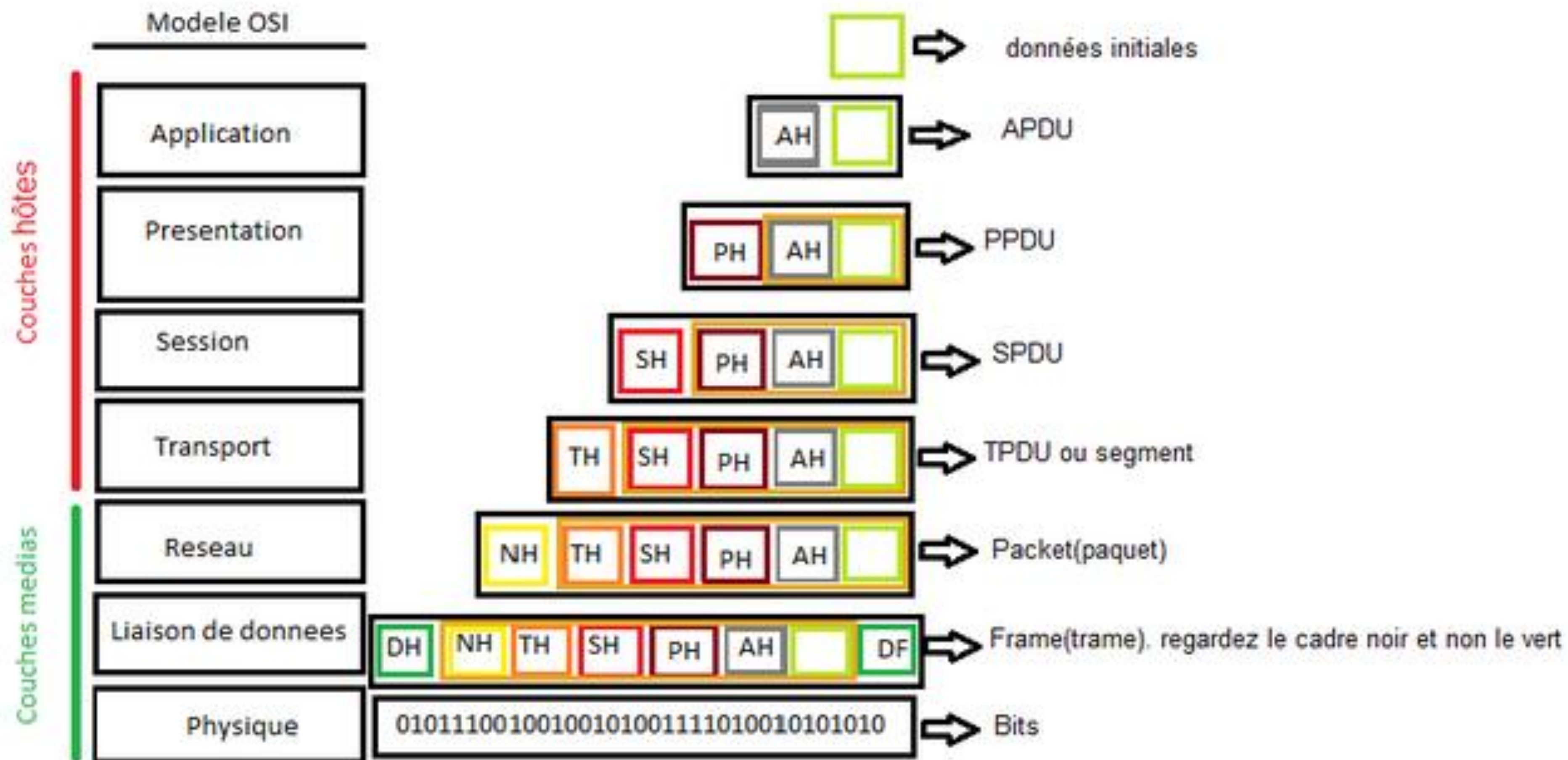


Rappels des protocoles OSI

- **Modèle en couches OSI**



Rappels des protocoles OSI



Rappels des protocoles OSI

architecture en 7 couches

- **Physique** : transmission des données binaires sur un support physique
- **Liaison** : gestion d'accès au support physique, assure que les données envoyées sur le support physique sont bien reçues par le destinataire
- **Réseau** : transmission de données sur le réseau, trouve les routes à travers un réseau pour accéder à une machine distante
- **Transport** : transmission (fiable) entre 2 applications
- **Session** : synchronisation entre applications, reprises sur pannes
- **Présentation** : indépendance des formats de représentation des données (entiers, chaînes de caractères...)
- **Application** : protocoles applicatifs (HTTP, FTP, SMTP ...)

Modèle client-serveur

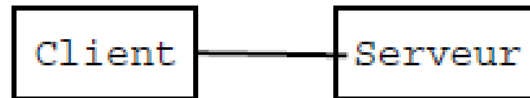
Modèle client/serveur

- Deux rôles distincts
  - ✓**Client** : demande que des requêtes ou des services lui soient rendus
  - ✓**Serveur** : répond aux requêtes des clients

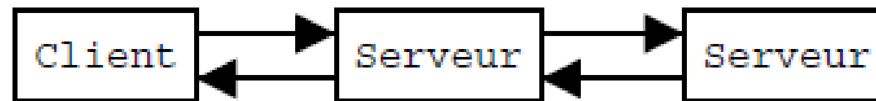
- Interaction **(Requête / Réponse)**
  - ✓Message du client vers le serveur pour faire une requête
  - ✓Exécution d'un traitement par le serveur pour répondre à la requête
  - ✓Message du serveur vers le client avec le résultat de la requête

Types d'architecture client-serveur

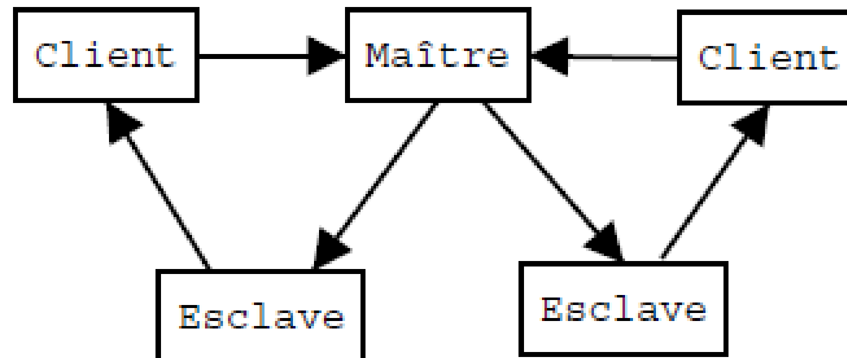
Un client, un serveur :



Un client, plusieurs serveurs :



Plusieurs clients, un serveur :



C/S orienté client ou serveur

Client lourd

- stocke les données et les applications localement. Le serveur stocke les fichiers mis à jour
- Le client effectue une bonne partie du traitement
- Le serveur est plus allégé

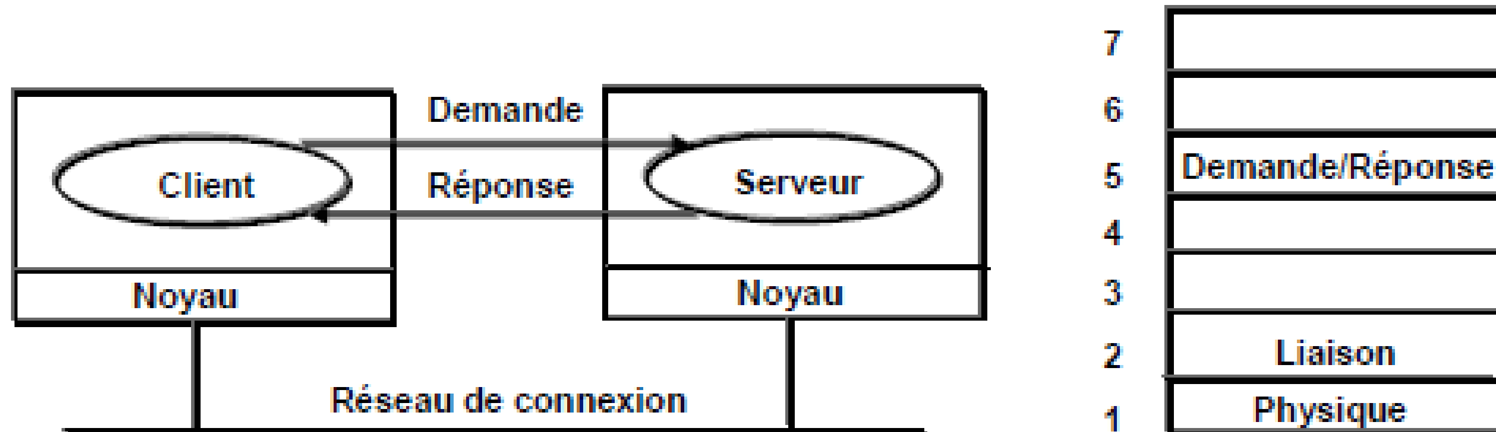
Serveur lourd

- On effectue plus de traitements sur le serveur : transactions, groupware, etc

Client léger

- Client à fonctionnalité minimale (terminaux X, périphérique réseau (Network Appliance), ordinateur réseau (network computer))
- Beaucoup de charge sur le serveur et le réseau

Description

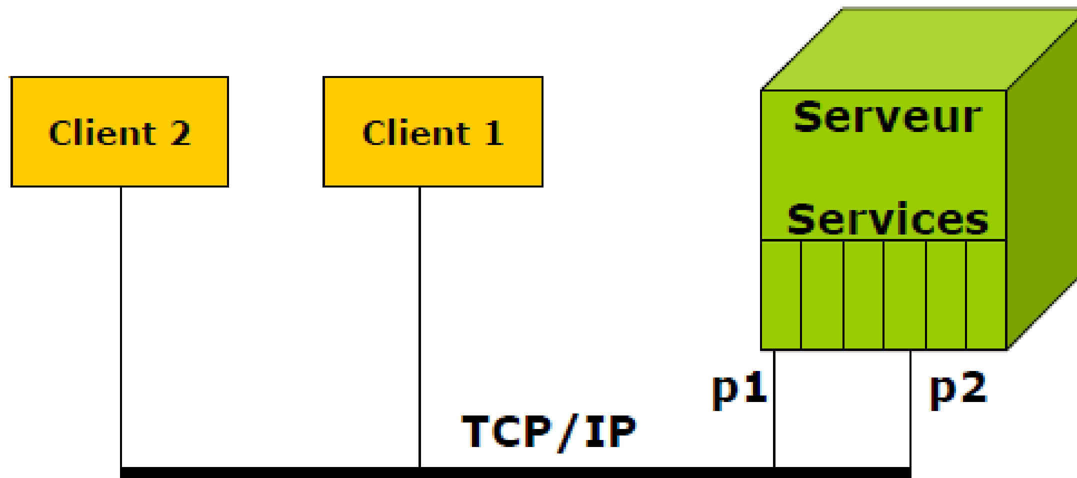


- Protocole simple sans connexion de type Question / Réponse
- Avantage : simplicité - efficacité
- Noyau : assurer l'envoi et réception des messages avec 2 primitives :

  - send (dest, &ptrm)

  - receive (adr, &prtm) (adr : l'adresse d'écoute du récepteur)

Principes

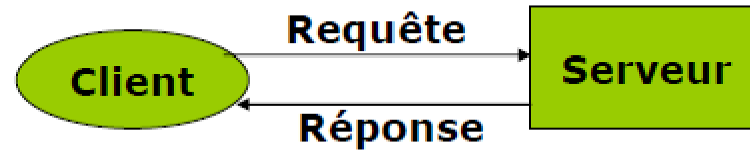


Client :
- émet des requêtes de demandes de service
- mode d'exécution synchrone
-le client est l'initiateur du
-dialogue client - serveur

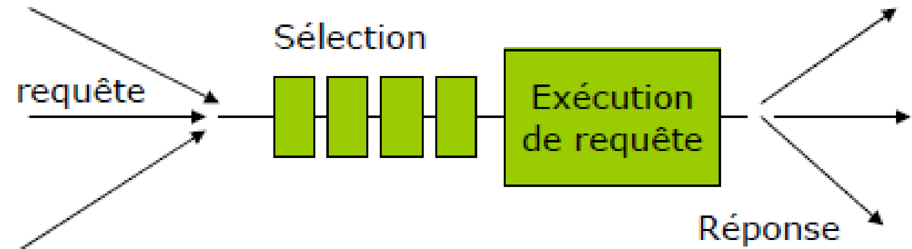
Serveur :
-ensemble de services exécutables
-exécute des requêtes émises par des clients
-exécution (séquentielle ou parallèle) des services

Principes

Client :
-Interface d'utilisateur
-Saisie de requête
-Envoie de requête
- attente du résultat
- affiche du résultat



Serveur :
- Gestion des requêtes (priorité)
- Exécution du service (séquentiel, concurrent)
- Mémorisation ou non de l'état du client



Deux modèles d'exécution :
- Client-Serveur à primitives : SEND et RECEIVE
- Client-Serveur à RPC (Remote Procedure Call)

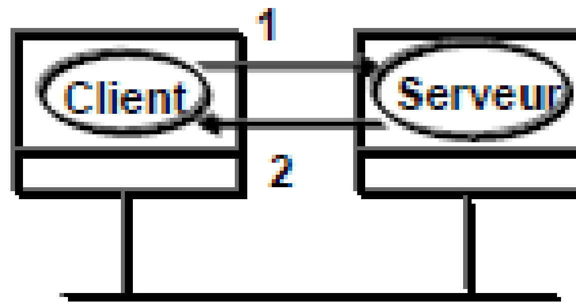
Messages client-serveur

- Trois grands types de message : REQ, REP et ACK
- Autres types possibles : AYA (Are You Alive), BUSY(ordinateur temporairement occupé), ERR (Erreur), etc.

Code	Type de paquet	source-dest.	Description
REQ	demande	client-serveur	le client désire un serveur
REP	réponse	serveur-client	la réponse du serveur au client
ACK	accusé de réception	noyau-noyau	le paquet précédent est arrivé
AYA	es-tu vivant ?	client-serveur	le serveur fonctionne-t-il ?
IAA	Je suis vivant	serveur-client	le serveur n'est pas en panne
TA	essaye à nouveau	serveur-client	le serveur est saturé
AU	adresse inconnue	serveur-client	aucun processus sur serveur utilise cette adresse

Modèle à primitifves: **Adressage**

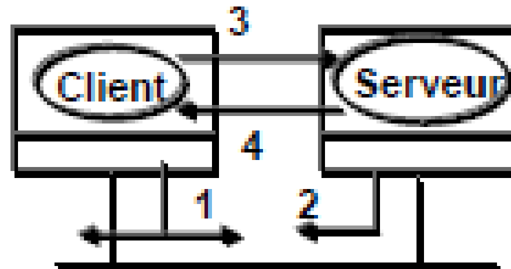
- **Localisation connue**



- tout client doit connaître la localisation et le nom interne de tous les services. Il demande donc explicitement sous format: machine.processus

Modèle à primitifves:Adressage

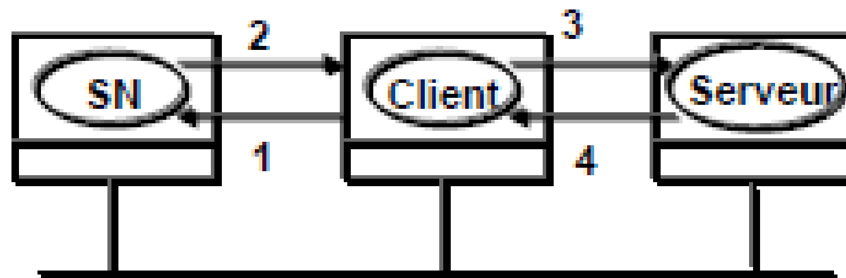
- **Localisation inconnue**



- Chaque processus choisit un nom aléatoire telle sorte qu'il soit unique
- Client diffuse une demande de localisation d'un processus
- Le serveur contenant ce processus donne son adresse machine

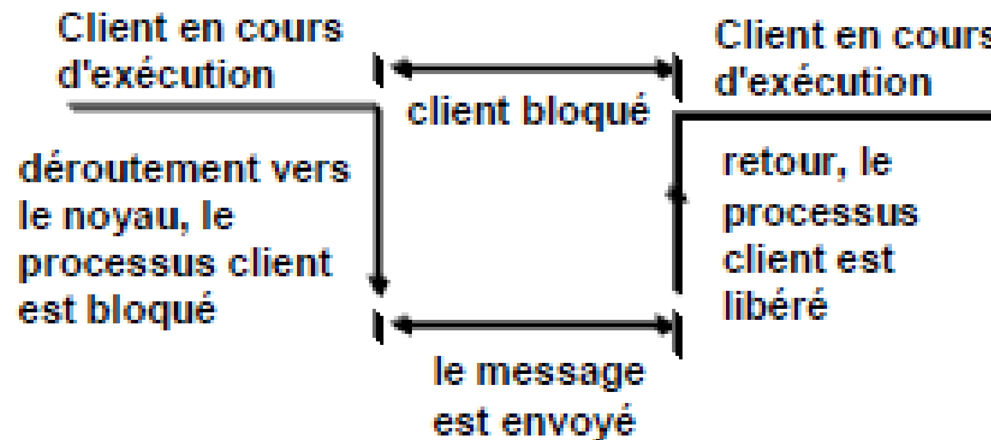
Modèle à primitifves:Adressage

- **Localisation centralisée**



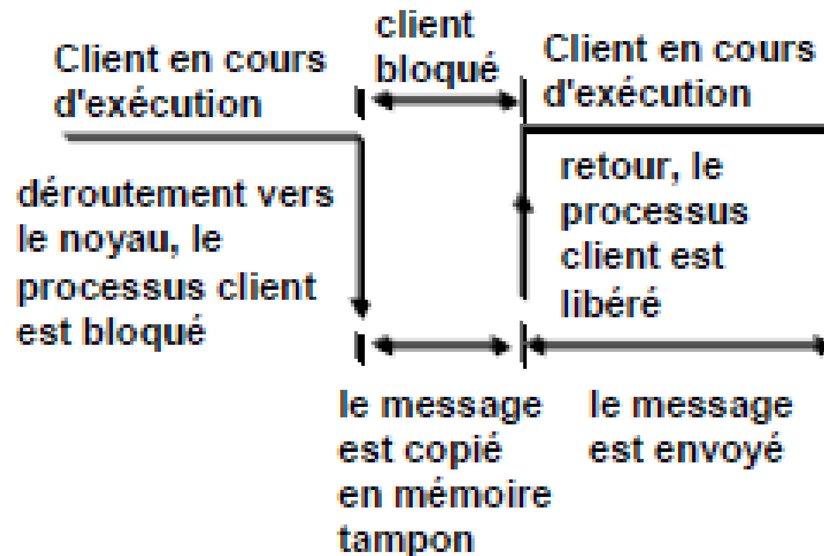
- les serveurs ont leur nom unique en code ascii
- un serveur gère la localisation et le nom interne des machines
- le client connaît seulement l'adresse du serveur de nom. Il lui demande de fournir l'adresse des serveurs à l'exécution

Primitives bloquantes et non bloquantes



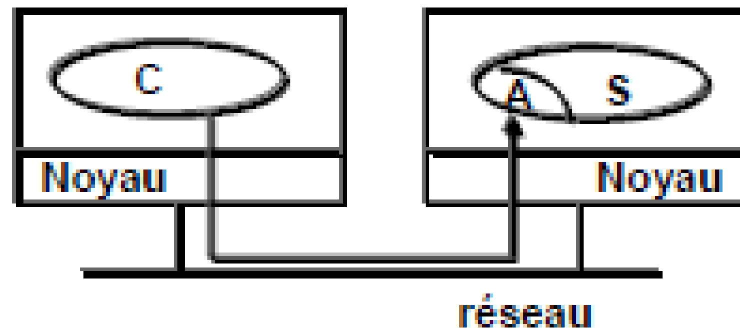
- le processus appelant send ou receive sera bloqué pendant l'émission (ou réception) du message
- l'instruction qui suit send (ou receive) sera exécutée lorsque le message a été complètement émis (ou reçu)
- avantage : simple et claire, facile à la mise en oeuvre
- inconvénient : moins performant

Primitives bloquantes et non bloquantes



- le processus appelant send sera libéré toute de suite, le message sera envoyé en // à l'exécution
- avantage : performant
- inconvénient : perte de mémoire tampon, difficile à réaliser, risque de modif. du tampon

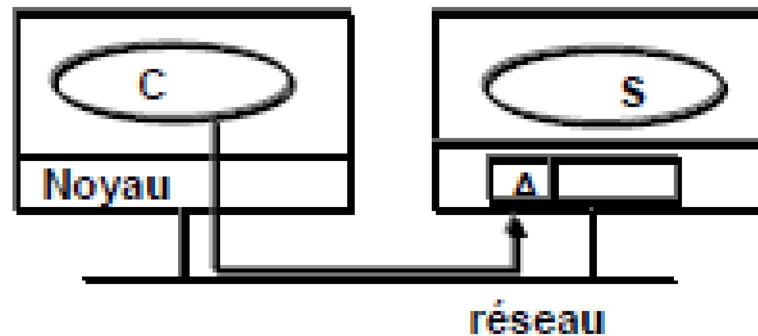
Primitives avec tampon ou sans tampon



- **sans tampon** : le message envoyé par le client sera copié par le noyau du serveur dans la zone ptrm et pour le processus A à l'adresse adr dans l'appel receive(adr,ptrm) déclenché par le processus A

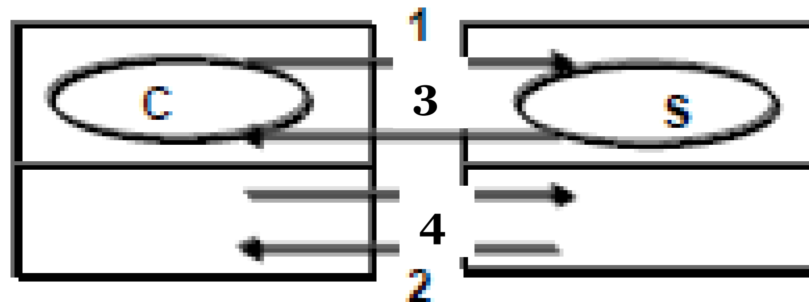
- **problème** : si le processus client a appelé send avant que le processus serveur appelle receive alors, le noyau du serveur n'a ni adresse du processus serveur ni l'adresse de l'endroit où il doit copier le message reçu !!!

Primitives avec tampon ou sans tampon



- **avec tampon** : les messages envoyés par les clients sont déposés dans une boite aux lettres. Le processus serveur A qui appelle un receive va déclencher une recherche par le noyau dans la boite aux lettres. Si le message existe, le noyau va le copier dans la zone de mémoire ptrm et libérer le processus A. Sinon l'appel receive sera bloqué pour l'attente de l'arrivée du message
- **problème** : quand la boite aux lettres est saturée, on doit refuser tous les messages qui arrivent !!!

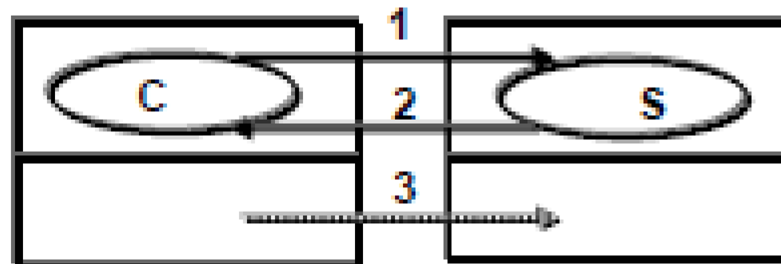
Primitives fiables et non fiables



Message avec accusés de réception individuels

1- Demande (client au serveur)
2- ACK (noyau à noyau)
3- Réponse (serveur à client)
4- ACK (noyau à noyau)

Primitives fiables et non fiables



Message avec la réponse = 1 accusé de réception

1- Demande (client au serveur)

2- Réponse (serveur à client)

3- ACK dans certain cas (noyau à noyau) (si le calcul est cher, on peut bloquer le résultat au serveur jusqu'à qu'il reçoit un ACK du client (3)

Primitives fiables et non fiables

On peut avoir un compromis entre 2 solutions précédentes :

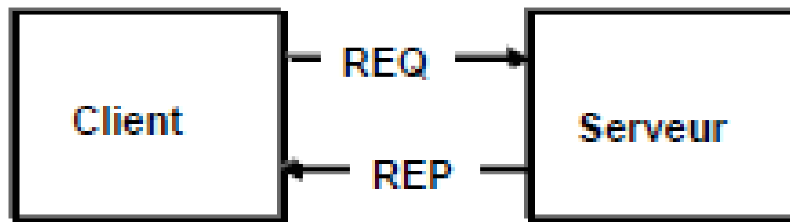
1- Demande (client au serveur)
2- ACK éventuellement si la réponse n'est pas revenue dans certain temps (noyau à noyau)
3- Réponse servant ACK si dans le lapes de temps prévu (serveur à client)
4- ACK éventuellement (noyau à noyau)

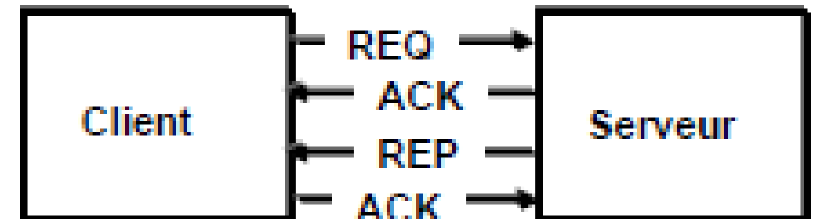
Modele Client/Serveur:Modèle à primitives

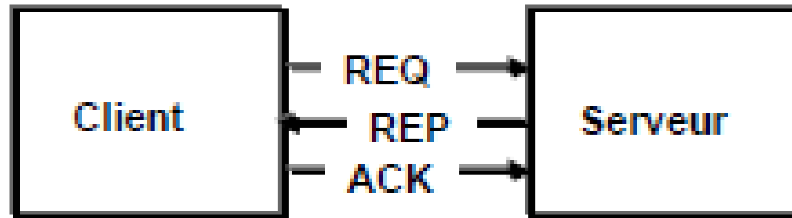
- Résumé

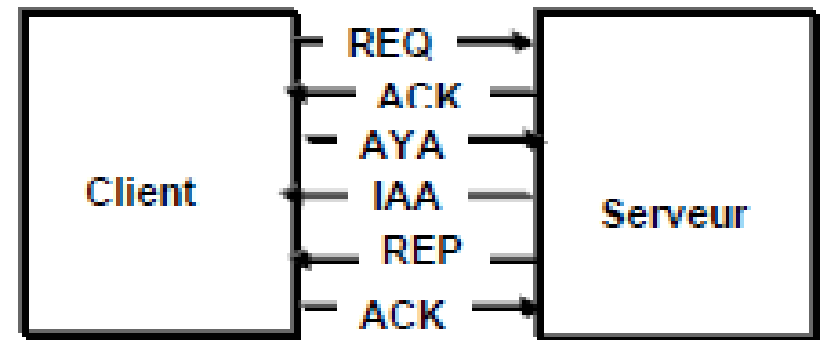
Objet	Option 1	Option 2	Option 3
Adressage	numéro de la machine	adresse choisie par le proc.	nom ASCII avec SN
Blocage	primitives bloquantes	primitives non-bloquantes avec copie vers le noyau	primitives non-bloquantes avec interruption
Mémorisation	pas de tampon	tampon temporaire	boite aux lettre
Fiabilité	non fiable	demande-ACK-réponse-ACK	demande--réponse-ACK

Exemples des échanges des paquets dans le protocole Client/Serveur









Les Sockets

Serveur en gestion multi-clients

- Par principe, les éléments distants communiquants sont actifs en parallèle
- Plusieurs processus concurrents
- Avec processus en pause lors d'attente de messages

Concurrence

Gestion plusieurs clients

- Particularité coté serveur en TCP
- Une socket d'écoute sert à attendre les connexions des clients
- A la connexion d'un client, une socket de service est initialisée pour communiquer avec ce client
- Communication avec plusieurs clients pour le serveur
- Envoi de données à un client

  UDP : on précise l'adresse du client dans le paquet à envoyer

  TCP : utilise la socket correspondant au client
- Réception de données venant d'un client quelconque

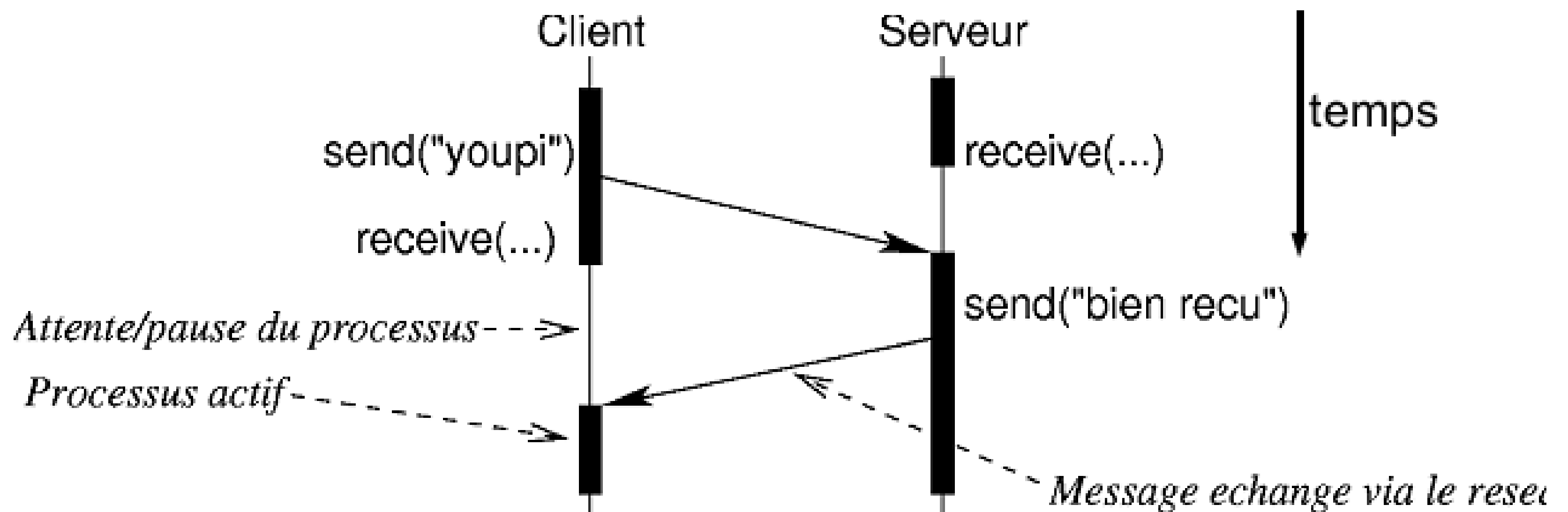
  UDP : se met en attente d'un paquet et regarde de qui il vient

  TCP : doit se mettre en attente de données sur toutes les sockets actives

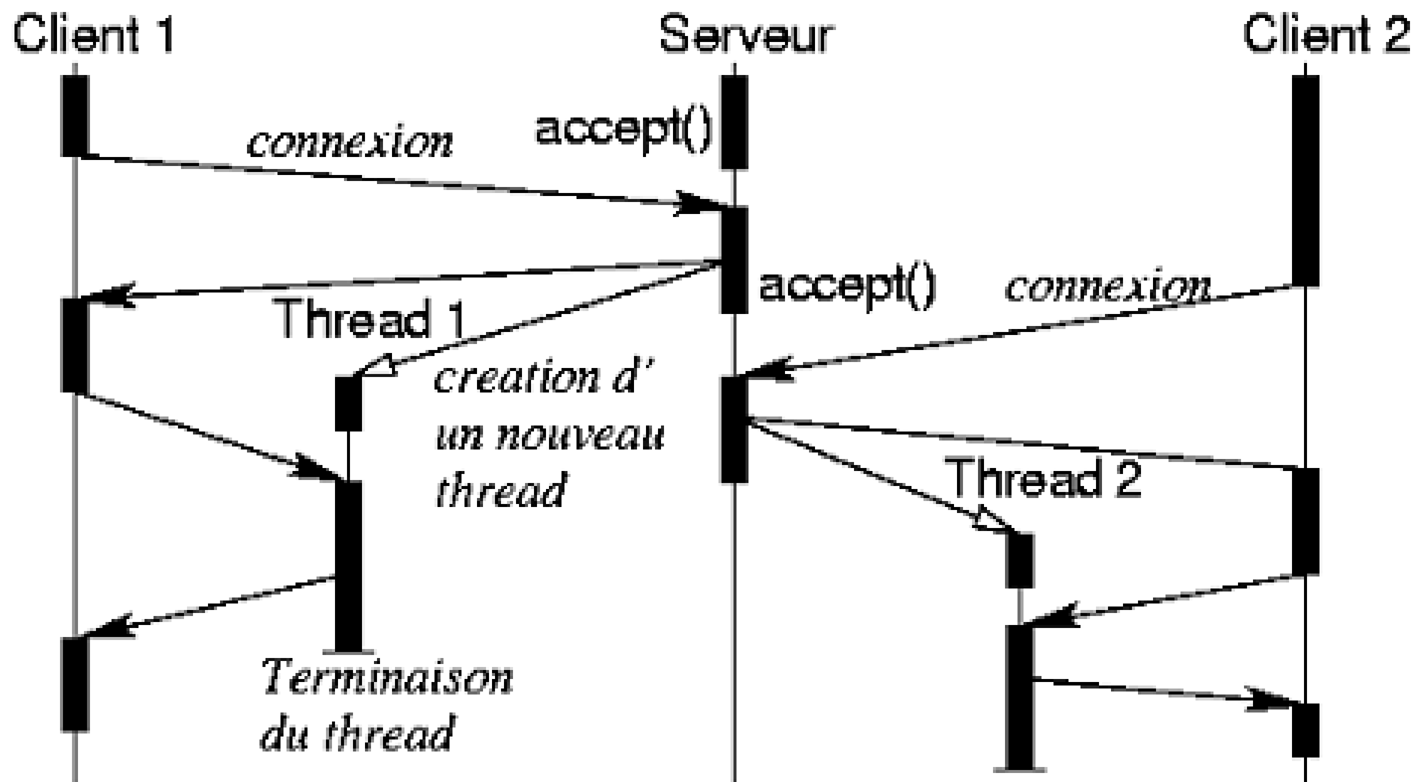
Gestion plusieurs clients

- Contrainte
- Lecture sur une socket : opération bloquante: Tant que des données ne sont pas reçues
- Attente de connexion : opération bloquante: Jusqu'à la prochaine connexion d'un client distant
- Avec un seul flot d'exécution (processus/thread)
- Si ne sait pas quel est l'ordonnancement des arrivées des données des clients ou de leur connexion au serveur: impossible à gérer
- Donc nécessité de plusieurs processus ou threads
- Un processus en attente de connexion sur le port d'écoute
- Nouvelle connexion : un nouveau processus est créé pour gérer la communication avec le nouveau client

Gestion plusieurs clients



Gestion plusieurs clients (suite)



Appels de procédures à distance (RPC)

Modèle RPC :Introduction

- **Problème du modèle Client/Serveur**

Concept basé sur l'échange explicite de données donc orienté Entrées/Sorties qui ne peut pas être le principe clé d'un système distribué

- **Appels de procédures à distance**

Concept introduit par Birrell et Nelson en 1984 : lorque un processus sur la machine A appelle une procédure sur une machine B, le processus sur A est suspendu pendant l'exécution de la procédure appelée qui se déroule sur B. Des informations peuvent être transmises de l'appelant vers l'appelé ou l'inverse par des paramètres

Qu’est ce que RPC?

- Les RPC (Remote Procedure Call : Appels de Procédures Distantes) ont été introduits par SUN. Ils sont basés sur les mécanismes de client-serveur. Ils perme ttent l'appel et l'exécution de procédures sur des machines distantes.

- Les RPCs sont construits au-dessus des sockets dont les mécanismes sont masqués au programmeur, présentant ainsi une interface de plus haut niveau.

Qu'est ce que RPC?

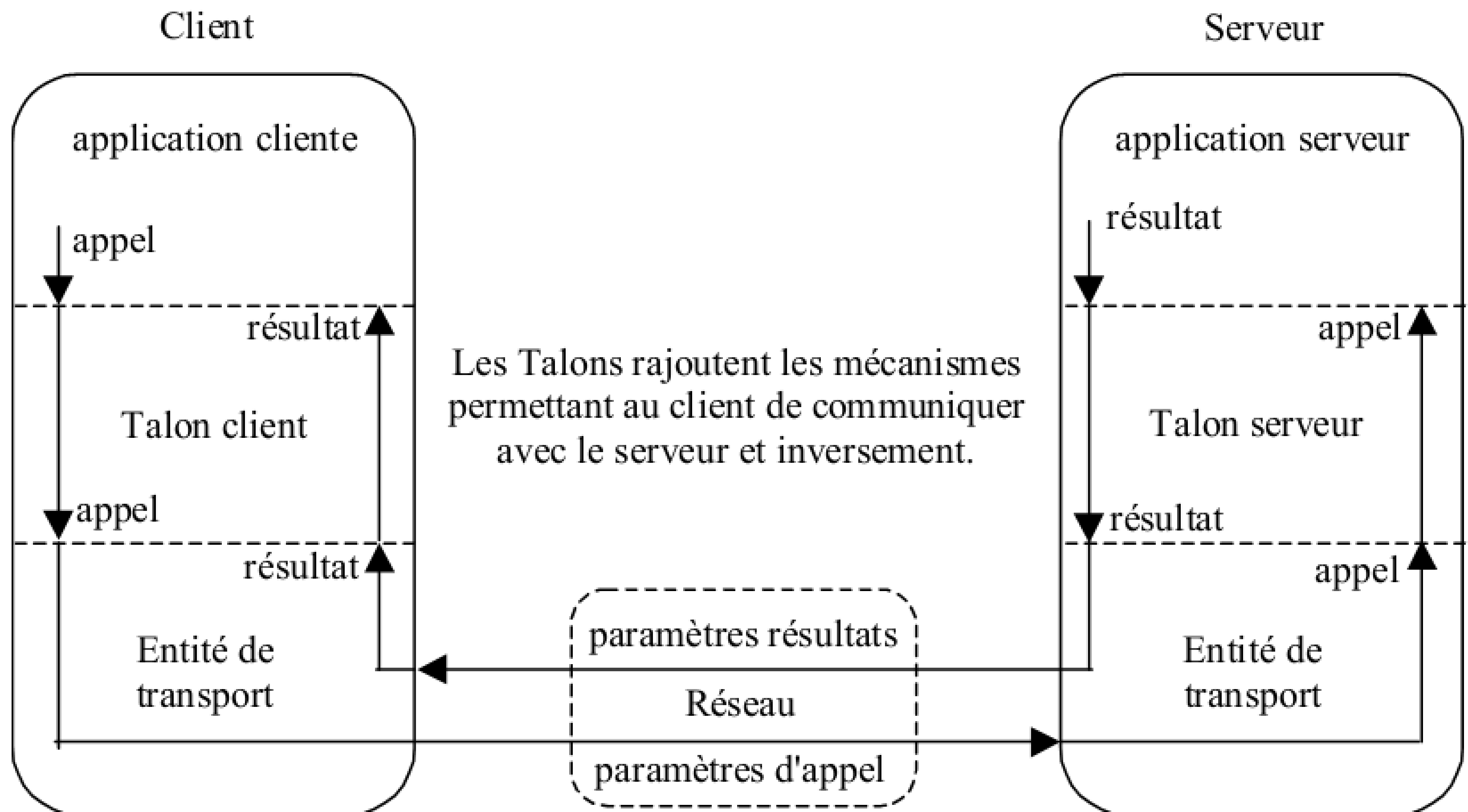
- Vous pouvez utiliser les RPC pour toutes sortes d'applications distribuées, par exemple l'utilisation d'un ordinateur très puissant pour des calculs intensifs (décryptage de données, calculs numériques ...). Cet ordinateur sera donc le serveur. Un autre ordinateur sera le client et appellera la procédure distante pour commander des calculs au serveur et récupérer le résultat

Principe de fonctionnement

- Il existe dans le programme client une fonction locale qui a le même nom que la fonction distante et qui, en réalité, appelle d'autres fonctions de la bibliothèque RPC qui prennent en charge les connexions réseaux, le passage des paramètres et le retour des résultats.

- Côté serveur, il suffira (à quelques exceptions près) d'écrire une fonction comme on en écrit tous les jours, un processus se chargeant d'attendre les connexions clientes et d'appeler votre fonction avec les bons paramètres. Il se chargera ensuite de renvoyer les résultats.

Fonctionnement détaillé d'un RPC



- Le programme client appelle un sous-programme à travers le réseau. Pour le client, tout se passe comme si l'appel était local. En effet le modèle RPC est similaire au modèle d'appel de procédure locale. Dans le cas local, les arguments d'appel sont stockés dans la pile (ou dans des registres) puis le client donne la main à la procédure avec éventuellement un contrôle sur le transfert des données. Ensuite, le résultat de la procédure est extrait de la pile (ou d'un registre) et le client continue son exécution.

- Les RPCs sont similaires : Le processus appelant envoie une requête et attend la réponse. Le message de requête contient entre autres les paramètres d'appel de procédure et la réponse, ceux de retour s'il y a lieu. Côté serveur, le processus est en attente d'une requête. Lors de l'arrivée d'une requête, il extrait les paramètres d'appel, exécute la fonction et récupère le résultat qu'il renvoie. Enfin, il se remet en attente de requête.

Fonctionnement d'une RPC

Les RPCs se décomposent en trois parties principales :

- La partie cliente, dite aussi "l'appelant", est l'application qui utilise une ou plusieurs procédures distantes.
- L'interface réseau, découpée en deux parties souvent appelées des Talons (« stubs » en anglais), est la partie qui gère tous les aspects distribués.
- La partie serveur, dite aussi "l'appelé", contient les procédures distantes.

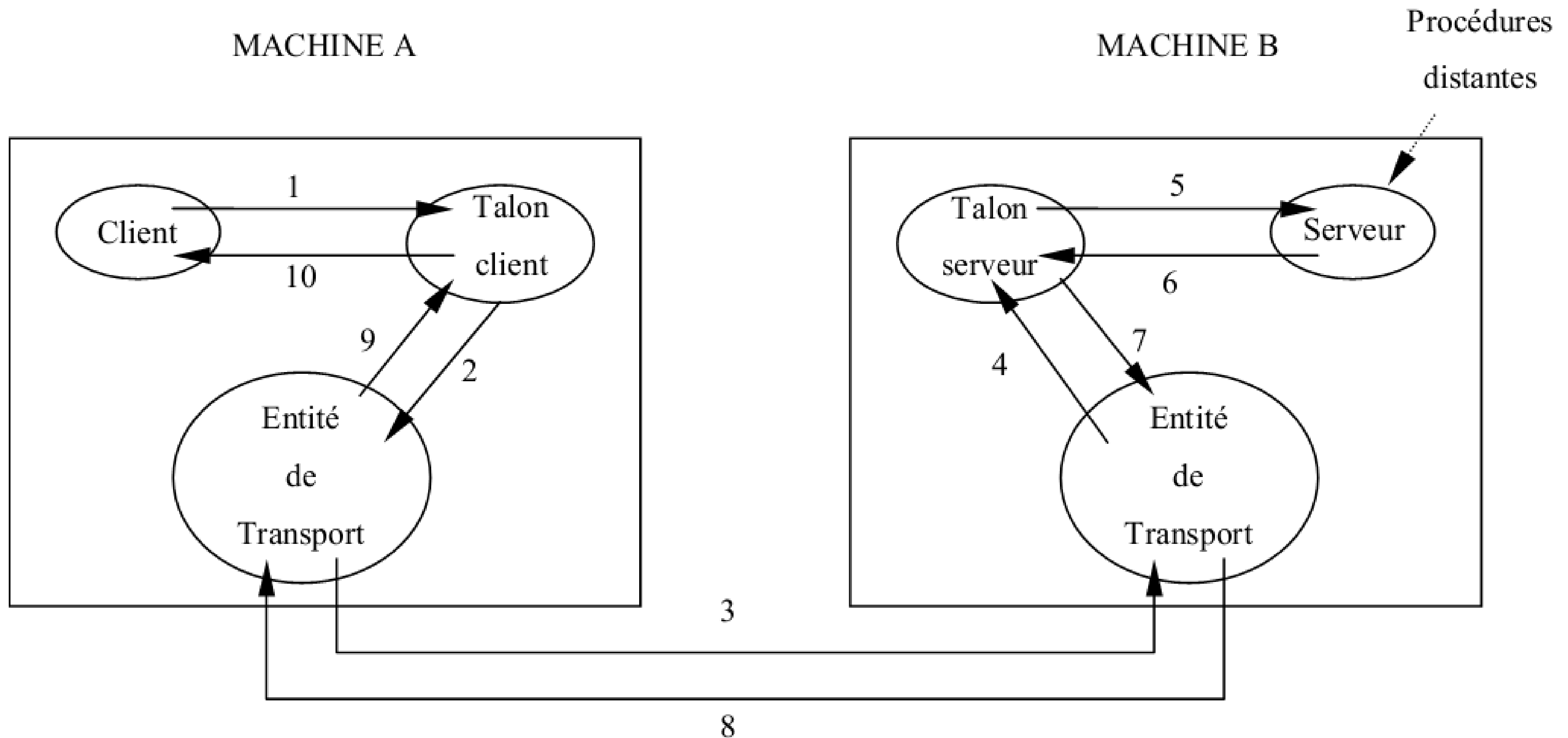
Principe de fonctionnement

- Les fonctions qui prennent en charge les connexions réseaux sont des « **stub** ». Il faut donc écrire un **stub client** et un **stub serveur** en plus du programme client et de la fonction distante.

- Le travail nécessaire à la construction des stubs client et serveur sera automatisé grâce au programme **rpcgen** qui produira du code C qu'il suffira alors de compiler.

- Il ne restera plus qu'à écrire le programme client, qui appelle la fonction et en utilise le résultat (par exemple en l'affichant), et la fonction elle-même.

Fonctionnement détaillé d'un RPC



Fonctionnement détaillé d'un RPC

1. Appel de la procédure Talon_client avec les paramètres (Il peut exister plusieurs procédures Talon,

ce qui permet de différencier les procédures distantes à appeler).

2. La procédure talon construit un message réseau da ns lequel elle place les paramètres utilisateur :

phase de codage.

3. Transmission sur le réseau.
4. La procédure Talon_serveur décode le message.
5. La procédure serveur appelle la fonction demandée avec les para mètres qui viennent d'être extraits

du message.

Fonctionnement détaillé d'un RPC

6. La procédure Talon_serveur récupère le (ou les) résultat(s) de l'exécution de la fonction demandée.
7. La procédure Talon_serveur construit un message dans lequel elle place les éventuels résultats.
8. Transmission sur le réseau.
9. La procédure Talon_client décode le message.
10. La procédure Talon_client retourne les résultats au client qui reprend son exécution.

Le passage de paramètres

- Le passage de paramètres dans le cadre des RPCS n'est pas forcément quelque chose de simple.
- Dans les systèmes répartis, les machines peuvent être différentes. Des problèmes de conversion peuvent apparaître.

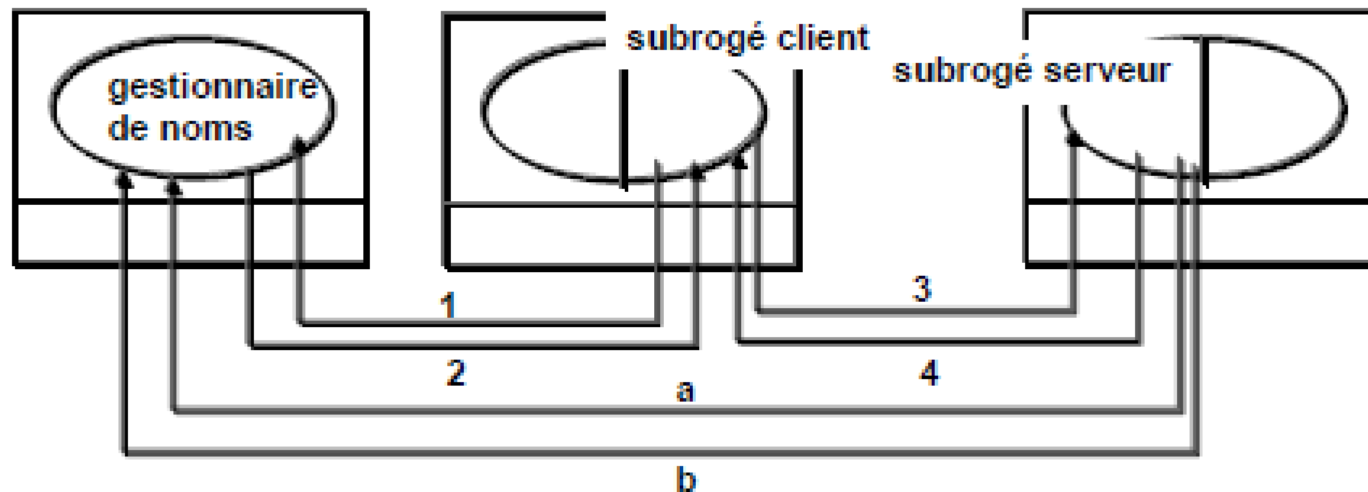
- La problématique va donc être de connaître le format du message et la plateforme cliente :
  - Le premier octet (Byte) du message détermine le format utilisé par le client.
  - Le serveur compare cet octet avec le sien : Si c'est le même aucune transformation n'est nécessaire, sinon il effectue la transformation.

Localisation du serveur

- La localisation d'un serveur peut se faire de différentes manières :
- Statique : Codée en dur l'adresse du serveur dans le client. En cas de changement de l'adresse, le client doit être recompilé (les programmes qui accèdent au serveur).
- Dynamique : lien dynamique (dynamic binding). Ce lien permet de faire dynamiquement connaître les clients et les serveurs.

Nommage dynamique (dynamic binding)

- Problème de la localisation du serveur par le client : écrire dans le code client l'adresse du serveur ou utiliser le mécanisme de nommage dynamique



a : enregistrer un nom (exportation de l'interface)
b : supprimer un nom
1- recherche un nom (importation)
2- retourne l'identificateur et descripteur
3 et 4- RPC

Le lien dynamique

La construction du lien dynamique s'effectue en plusieurs étapes.

Définition de la spécification du serveur qui servira à la génération des souches.

À l'initialisation, le serveur exporte son interface. Il l'envoie à un binder (relieur) pour signaler son existence.

C'est la phase d'enregistrement. Le serveur envoie ses informations :

- nom ;
- version identifiant (unique sur 32 bits en principe) ;
- handle (adresse IP, protocoles, etc.).

Lorsqu'un Client appelle une procédure pour la première fois :

- Il constate qu'il n'est pas relié à un serveur.
- Il envoie un message au relieur pour importer la version de la procédure qu'il désire invoquer.
- Le relieur lui renvoie l'identifiant unique (ID) et le handle.

Le lien dynamique

Avantage :

- Cette méthode d'importation et exportation des interfaces est très flexible.
- Les serveurs peuvent s'enregistrer ou se désenregistrer.
- Le client fournit le nom de la procédure et la version et il reçoit un ID unique et handle.

L’outil jrpcgen

C’est le compilateur qui permet de créer automatiquement les stubs (client et serveur) à partir d’un fichier en langage RPCL

RPCGEN

- Pour éviter d'avoir à programmer la partie réseau de l'application, en particulier celle concernant le serveur, pour faciliter la production des filtres XDR, il existe un utilitaire d'aide au développement des programmes utilisant les RPC.
- Cet outil s'appelle **rpcgen** (cf. man rpcgen).Il s'agit d'un précompilateur qui engendre plusieurs fichiers à partir d'une description synthétique des services, description donnée dans un fichier dont le type est x. Le langage de description ressemble au langage C.

RPCGEN

Exemple:

si le fichier de description s'appelle fichier.x, les fichiers produits par l'exécution de la commande : rpcgen fichier.x seront les suivants :

- un fichier à inclure dans les programmes du serveur et des clients (fichier.h),
- un squelette du code du serveur (fichier_svc.c),
- les procédures réalisant les appels distants pour les clients (fichier_clnt.c),
- les filtres xdr (fichier_xdr.c).
- Il reste donc à écrire deux fichiers : un fichier contenant le code des procédures fournies par le serveur qui complétera fichier_svc.c, lui-même contenant déja la fonction main,
- le code du client qui se servira de fichier_clnt.c, ici on doit écrire la fonction main.

Les problèmes liés aux RPCs dans les systèmes répartis.

1. Le client est incapable de localiser le serveur
2. La demande du client au serveur est perdue
3. La réponse du serveur au client est perdue
4. Le serveur tombe en panne après avoir reçu une demande
5. Le client tombe en panne après avoir envoyé une demande

Les problèmes liés aux RPCs dans les systèmes répartis.

Le serveur est en panne :

- Il faut dissocier la panne avant l'exécution de la requête ou après l'exécution.

Trois écoles pour résoudre ce problème :

1. Attendre que le serveur redémarre et relancer le traitement. « Au moins un traitement est réussi. » (peut-être plus).
2. Abandon du client et rapport de l'erreur. « Au plus un traitement est réussi. » (peut-être aucun).
3. Ne rien dire au client (aucune garantie) Seuls avantages : Facile à gérer et à implémenter. En règle générale, **crash du serveur = crash du client.**

Les problèmes liés aux RPCs dans les systèmes répartis.

Le client est en panne :

Lorsqu'un client tombe en panne alors qu'il a demandé un traitement à un serveur, on dit que l'échange devient orphelin. Cela a pour effet de gaspiller du temps CPU, de bloquer des ressources, lorsqu'il redémarre, il peut recevoir des messages antérieurs à la panne (problème de confusion).

Les problèmes liés aux RPCs dans les systèmes répartis.

Quatre solutions peuvent être envisagées :

1. **L'extermination** : Le client enregistre les appels dans une log. Au redémarrage l'orphelin est détruit. Beaucoup d'écritures disque à chaque RPC.
2. **La réincarnation** : Le client tient compte du temps. Lorsqu'il redémarre, il annule les échanges ente deux époques et reprend les échanges orphelins.
3. **La réincarnation à l'amiable** : Comme précédemment, avec en plus la vérification avec le serveur concerné des échanges orphelins.
4. **Expiration du délai** : Un compteur de temps détermine quand un échange devient orphelin. Les orphelins sont détruits. Un temps T déterminé identique pour tous est donné à 1 RPC. Difficile de choisir un temps T (les RPCs sont sur des réseaux différents).

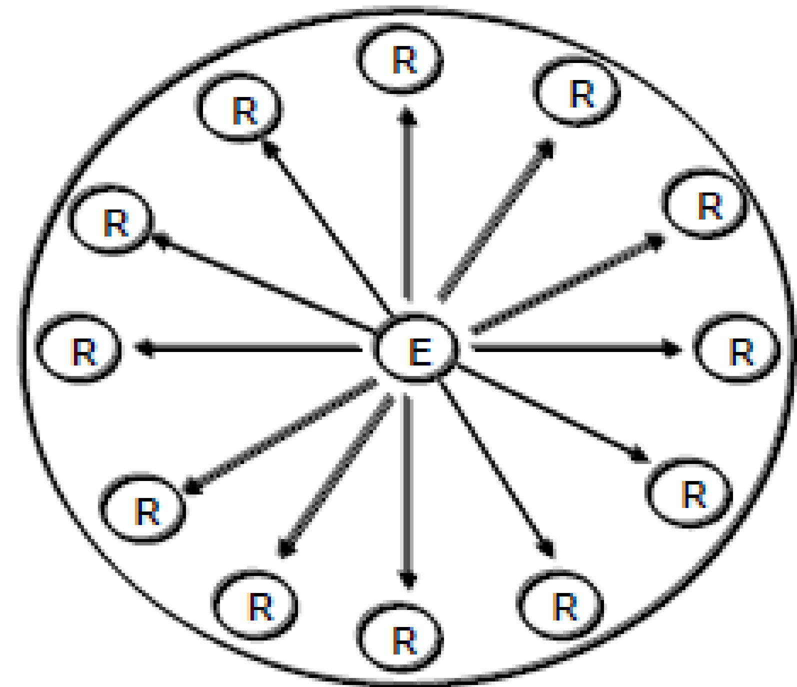
La communication de groupe

Introduction

Point-à-Point

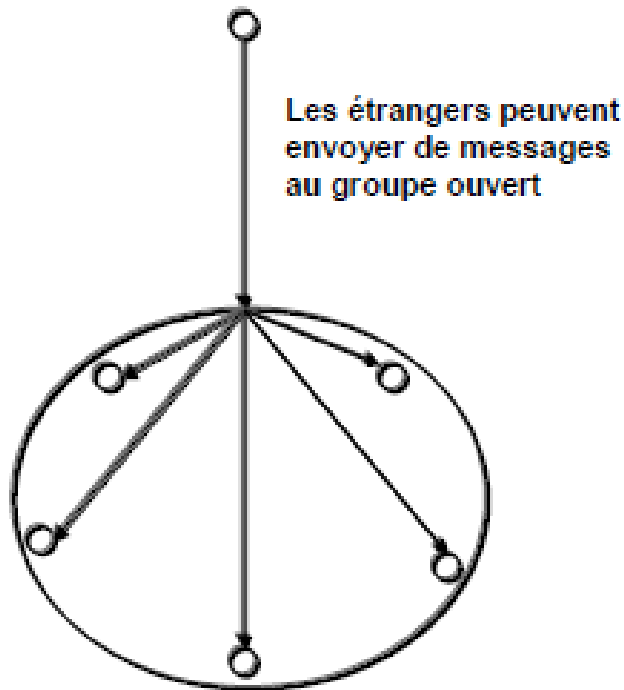
Généralisation du RPC

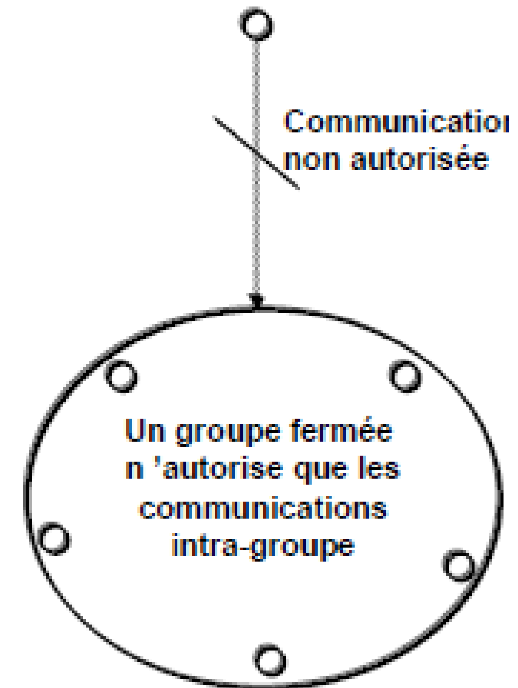
Un émetteur doit communiquer en parallèle à n récepteurs regoupés dans un groupe sémantique



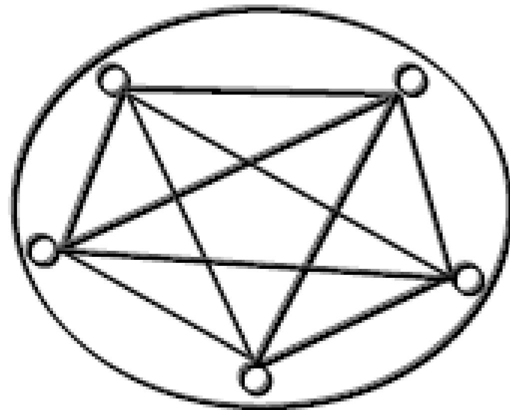
Un-à-Plusieurs

Classement
a) Groupes fermés et groupes ouverts





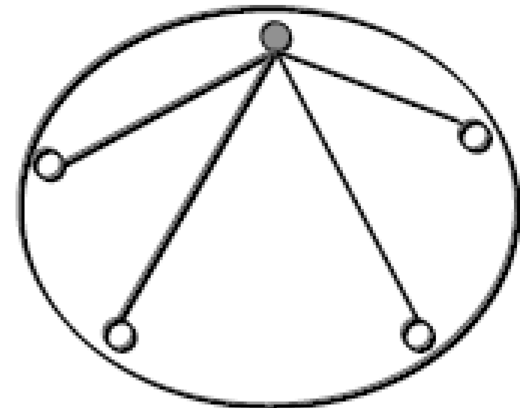
Classement
b) Groupes de paires ou groupes hiérarchisés



Groupe de paires

Coordinateur

Travailleurs



Groupe hiérarchisé

Gestion des groupes

-Création, dissolution d 'un groupe
-Adhésion à un groupe ou retrait d 'un groupe

Gestion centralisée

- Un serveur de groupes qui gère une base de données des groupes et une base de données des membres

- Avantages : efficace, claire et facile à implanter

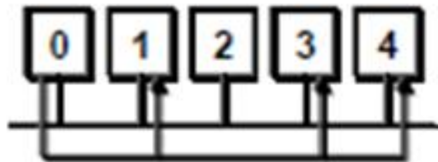
- Inconvénients : fragile devant les pannes

Gestion distribuée

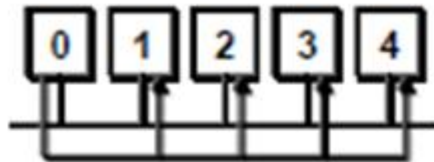
- Un groupe se crée s 'il y a au moins un adhérant
- pour adhérer à un groupe, le processus demandeur doit envoyer sa demande à tous les membres du groupe

- Avantage : Robustesse
- Inconvénient : complexe et coûteuse

Adressage des groupes



Diffusion restreinte
(multicasting)



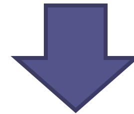
Diffusion globale
(broadcasting)

monodifusion
(point-to-point)

Atomicité

- Propriété de diffusion atomique (atomic broadcast)

Un message envoyé à un groupe doit être reçu par tous ses membres ou par aucun



Algorithme de « relais intra-groupe » (Joseph et Birman 1989)

1- L 'expéditeur envoie un message à tous les membres d 'un groupe (des temporisateurs sont armés et des retransmissions faites si nécessaire)

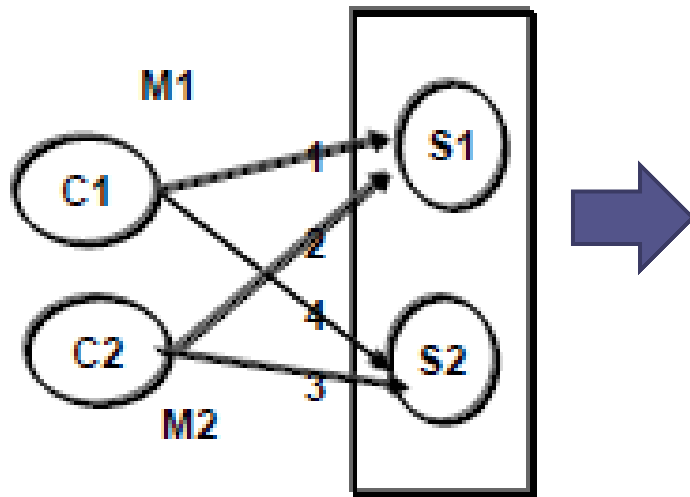
2- Tout membre de ce groupe, qui a reçu ce message pour la première fois, doit l 'envoyer à tous les membres du groupe (des temporisateurs sont armés et des retransmissions faites si nécessaire)

3- Tout membre de ce groupe, qui a reçu ce message plus qu 'une fois, doit simplement le détruire

Séquencement

Propriété de séquencement

Soient A et B 2 messages à envoyer à un groupe de processus G. Si A est envoyé avant B sur le réseau, ils doivent être reçus par tous les membres du groupe G dans le même ordre : A puis B

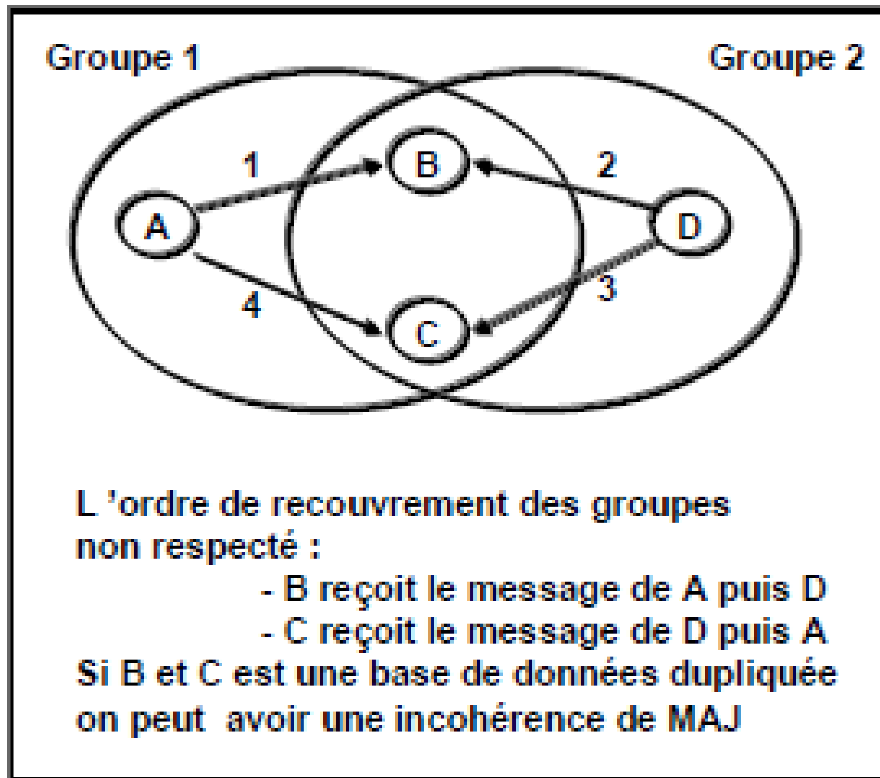


Solution

-Séquencement global (global time ordering) : conserver, en réception, l ’ordre des messages à émission.
Si C1 envoie M1 avant que C2 envoie M2, alors le système doit assurer que tous les membres de G reçoivent M1 avant M2

- Séquencement cohérent (Consistent time ordering) :
Le système établi l ’ordre d ’envoie des messages et assure que cet ordre soit respecté à la réception des messages.

Groupes non disjoints



Solution:

Introduction des mécanisme de séquencement qui tient compte du recouvrement des groupes

Probléme:

La mise en oeuvre est compliquée et coûteuse

Middleware

Middleware ou intergiciel

- **Motivations**

Dans un systeme reparti, meme l'interface fournie par les systemes d'exploitation et de
communication est encore trop complexe pour etre utilisee directement par les applications.

- Heterogeneite
- Complexite des mecanismes (bas niveau)
- Necessite de gerer (et de masquer, au moins partiellement) la repartition

Solution

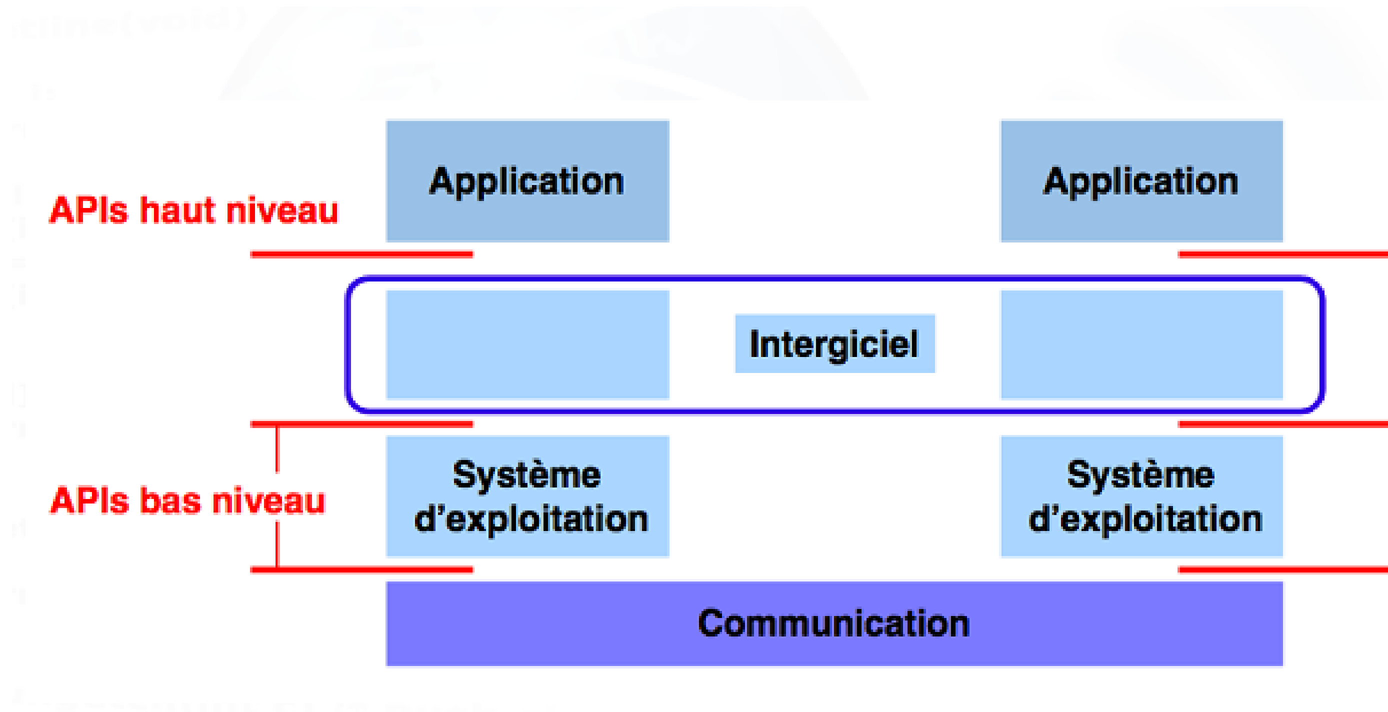
Introduire une couche de logiciel intermediaire (repartie) entre les niveaux bas (systemes et communication) et le niveau haut (applications) : c'est l'intergiciel

L'intergiciel joue un role analogue a celui d'un "super-systeme d'exploitation" pour un systeme reparti

Middleware ou intergiciel

- Couche logiciel
- S'intercale entre le système d'exploitation/réseau et les éléments de l'application distribuée
- Assure les connexions entre les serveurs de données et les outils de développement sur les postes clients
- Ensemble de services logiciels construits au dessus d’un protocole de transport afin de permettre l’échange de requêtes et des réponses associées entre client et serveur *de manière transparente.*

Middleware ou intergiciel



Types de middleware

Général

- Protocoles de communication, répertoires répartis, services d'authentification, service de temps, RPC, etc
- Services répartis de type NOS (Networked OS) : services de fichiers, services d'impression.

Spécifique

- de BD : ODBC, IDAPI, EDA/SQL, etc
- de groupware : Lotus Notes
- d'objets : CORBA, COM/DCOM, .NET

Composantes du middleware

Les canaux

- Services de communications entre composants et applications : RPC (synchrone), ORB (synchrone), MOM (Message Oriented Middleware) (asynchrone) Services de support de communication : SSL, annuaires (LDAP)

Les plate-formes

- Serveurs d'applications qui s'exécutent du côté serveur
- Offrent les canaux de communication
- Assurent la répartition, l'équilibrage de charge, l'intégrité des transactions, etc

Fonctions d’un middleware

L'intergiciel a quatre fonctions principales

- Fournir une interface ou API (Applications Programming Interface) de haut niveau aux applications
- Masquer l'heterogeneite des systemes materiels et logiciels sous-jacents
- Rendre la repartition aussi invisible ("transparente") que possible
- Fournir des services repartis d'usage courant

L'intergiciel vise a faciliter la programmation repartie

- Developpement, evolution, reutilisation des applications
- Portabilite des applications entre plates-formes
- Interoperabilite d'applications heterogenes

Merci pour votre attention!