L’appel de procédure distante ‘ RPC Remote Procedure Call ’

Gérard Florin
- CNAM -
- Laboratoire CEDRIC -

Plan

- Introduction
- Mise en oeuvre de l'appel de procédure distante
- Gestion du contrôle
- Gestion des données
- Transmission des arguments (présentation)
- Désignation / liaison
- Tolérance aux pannes
- Conclusion
- Bibliographie

Introduction

L'approche client-serveur en appel de procédure distante

- **Mode de réalisation d'une interaction client serveur**
  - ou l'opération à réaliser est présentée sous la forme d'une **procédure**
  - que **le client** peut faire exécuter à distance par **le serveur**.
- **Service basique (API d'appel de procédure distante)**

```
/* Coté client : invoque génère l'appel distant et récupère le résultat*/
   invoque (id_client, id_serveur, nom_procedure, paramètres);
/* Coté serveur : reçoit, traite un appel et répond */
   traite (id_client, id_serveur, nom_procedure, paramètres);
```

- **Service intégré objet**

```
/* Coté client : on invoque une procédure localisée à distance*/
   ref_objet_serveur.nom_procedure (paramètres);
/* Coté serveur : on déploie l'objet qui implante la procédure*/
   method nom_procedure (paramètres);
```

Avantage majeur de l'approche client-serveur en appel de procédure distante

- **S'affranchir du coté basique des communications en mode message.**
  - Ne pas avoir à programmer des échanges au niveau réseau en mode message
  - Ne pas utiliser pour construire une application répartie des schémas de contrôle trop simples (affectation dans cohérence, fork)
- **Utiliser une structure familière: l'appel de procédure.**
  - Problème: ne pas ignorer les différences centralisé/réparti.
- **Disposer de mécanismes modernes de programmation.**
  - Vision modulaire des applications réparties (en approche objets répartis ou par composants sur étagères).
  - Réutilisation par délégation en univers réparti.

Les implantations de l'appel de procédure distante (1)

Les approches à RPC traditionnelles

- SUN ONC/RPC
  - Open Network Computing / Remote Procedure Call
- OSF DCE
  - Open Software Foundation - Distributed Computing Environnment
- Systèmes de gestion de bases de données: procédures stockées.

Les implantations de l'appel de procédure distante (2)

Approches à RPC intégrées dans les systèmes d'objets répartis

- OMG CORBA
  - Object Management Group - Common Object Request Broker Architecture
- SUN Java RMI
  - Remote Method Invocation
- Microsoft - DCOM
  - Distributed Component Object Model

Les implantations de l'appel de procédure distante (3)

Approches à RPC intégrées dans les systèmes de composants

- SUN J2EE EJB
  - Java 2 (Platform) Enterprise Edition - Enterprise Java Beans
- OMG CCM
  - Object Management Group - Corba Component Model
- WS-SOAP
  - Web Services - Simple Object Access Protocol

Mise en oeuvre de l’appel de procédure distante

A) Par migration.

B) Par mémoire partagée.

C) Par messages.

D) Par appel léger.

A) Réalisation de l'appel de procédure distante par migration

- **Stratégie de migration:** Le code et les données de la procédure distante sont amenés sur le site appelant pour y être exécutés par un appel local habituel**.**
- Analogie : stratégie de **pré-chargement** en mémoire.

Avantages

- Très efficace pour de nombreux appels.

Inconvénients

- Univers d'exécutions homogènes (ex machine virtuelle).
- Performances selon le volume de codes et de données.
- Problèmes de partage des objets (fermeture d'objets, ...).

B) Réalisation de l’appel de procédure distante en mémoire partagée répartie

- L'appel distant est réalisé en utilisant une **mémoire virtuelle partagée répartie.**
- La procédure est installée pour le client comme pour le serveur dans la mémoire partagée répartie.
- Elle est en fait dans l’espace **réel du serveur**.
- L'appel du client se fait comme si la procédure était locale, provoquant un premier **défaut de page** sur le début du code de la procédure.
- Le code et les données de la procédure distante sont amenés page par page sur le site appelant **selon le parcours du code et des données.**
- Analogie avec une stratégie **page à la demande.**

Réalisation de l'appel de procédure distante en mémoire partagée répartie

Avantages

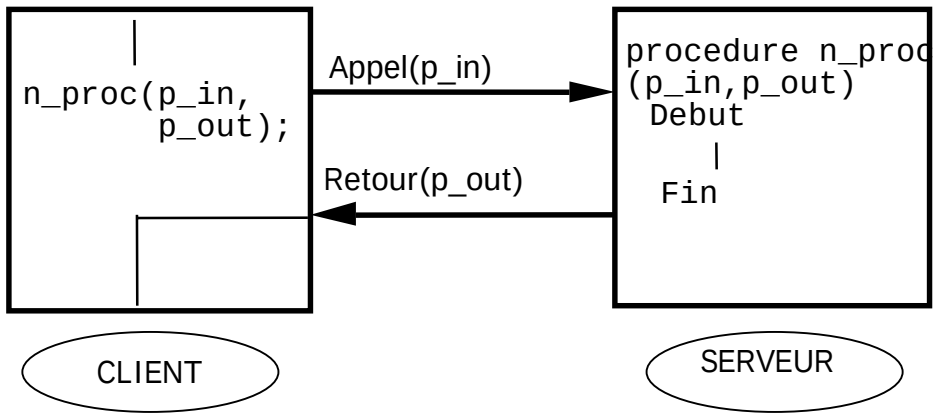
- Efficace en cas de nombreux appels.
- Efficace si tout le code et les données ne sont pas visités.
- Résout le problème de l'utilisation des pointeurs (références d'adresses en mémoire).

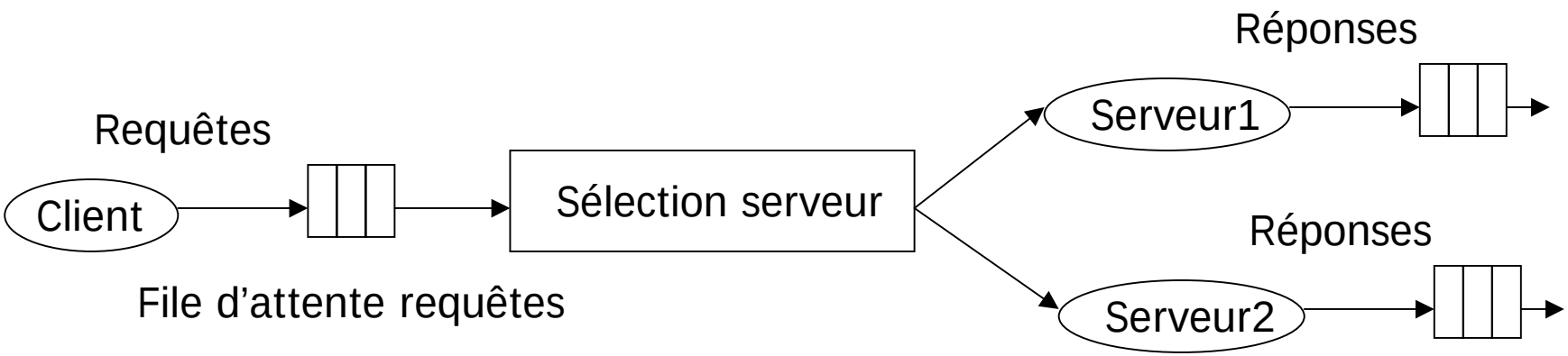
Inconvénients

- Univers de systèmes homogènes.
- Volume de codes et de données à échanger pages par pages.
- Problèmes de partage selon cohérence de la mémoire répartie.

C) Réalisation de l'appel de procédure distante par messages asynchrones

- Deux messages (au moins) échangés: requête et réponse.

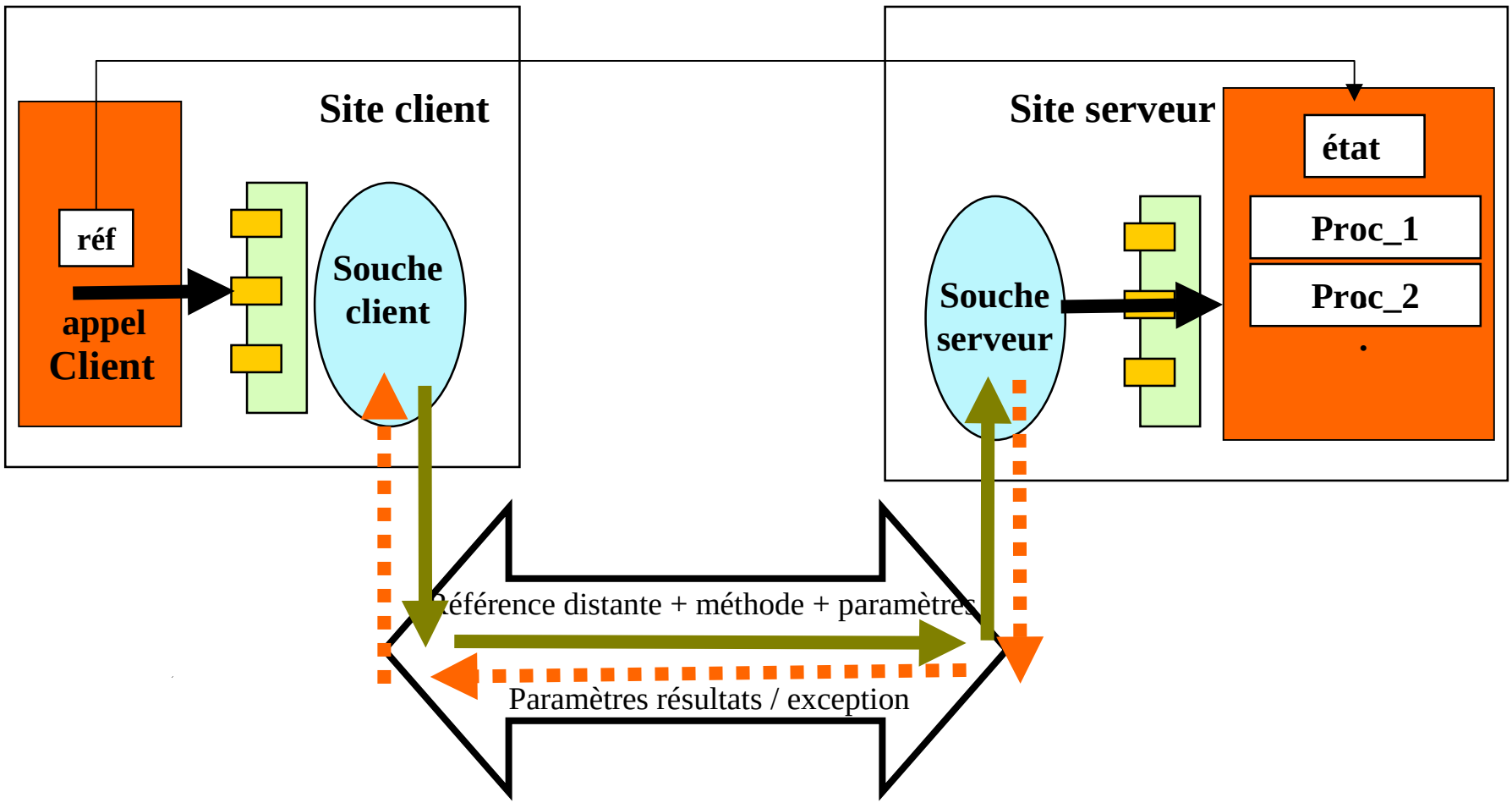




Notion de souches

- **Un mode de réalisation par interception (‘ wrapping ’)**
  - Une procédure intercepteur (‘wrapper’) intercepte l’appel d’un client vers un serveur et modifie le traitement serveur à sa guise.
  - Décomposition en intercepteur coté client et intercepteur coté serveur.
  - Décomposition en traitements avant et après le traitement serveur.
- **Souches: transformation d’un appel local en appel distant.**
- **Objectif de génération automatique** des souches connaissant le profil d ’appel de la procédure distante**.**
- Très nombreuse terminologie dans ce cas : **Souches ("Stubs"), Talons, Squelettes ("Skeletons")...**

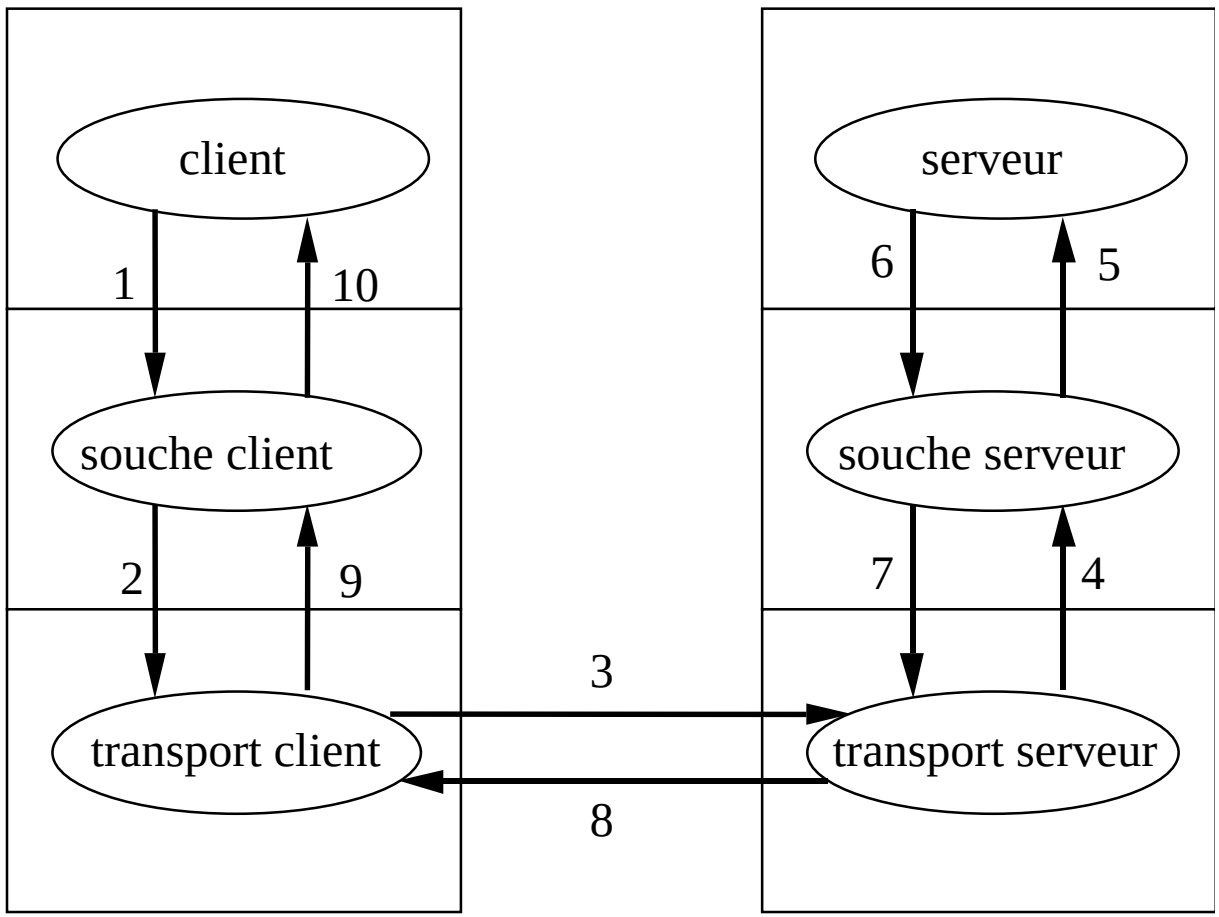
Les souches: diagramme global d'interaction



Souches client et serveur

- **La souche client (“client stub”)**
  - Procédure coté client qui reçoit l’appel en mode local
  - Le transforme en appel distant
  - En envoyant un message
  - Reçoit le message contenant les résultats après l'exécution
  - Retourne les résultats comme dans un retour de procédure.
- **La souche serveur (“server stub”)**
  - Procédure coté serveur qui reçoit le message d ’appel,
  - Fait réaliser l’exécution sur le site serveur par la procédure serveur
  - Récupère les résultats et retransmet les résultats par message.

Les étapes d'un appel de procédure distante par messages



Avantages/inconvénients de l'appel distant réalisé par messages

Avantages

- Volume de code ou de données serveur quelconque.
- Applicable en univers hétérogènes moyennant des conversions.
- Partage d'accès sur le site serveur analogue au transactionnel.

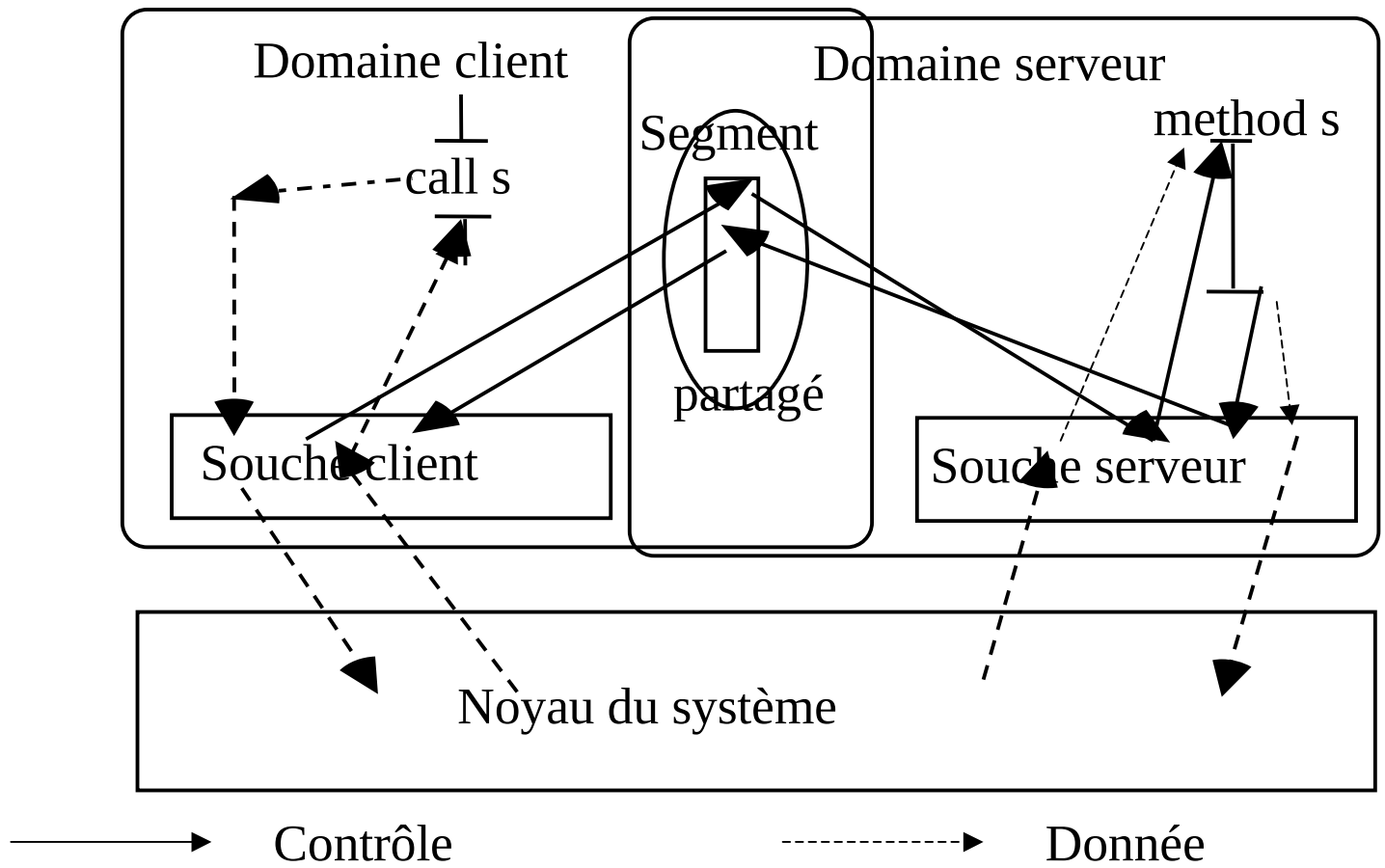
Inconvénients

- Pas d'usage des pointeurs dans les paramètres.
- Échange de données complexes/de grande taille délicat.
- Peu efficace pour de très nombreux appels.

D) Réalisation de l’appel de procédure distante léger (‘ lightweight RPC ’)

- **Problème de performances** : quand on invoque un serveur qui se trouve sur la même machine la traversée des couches réseaux est inutile et coûteuse.
- Si le serveur se trouve **dans le même processus** (même domaine de protection) pas de problème (appel local).
- Si le serveur se trouve **dans un autre processus** (autre domaine de protection)
  - **Solution proposée** : la communication réseau est réalisée par un segment de mémoire partagée entre le client et le serveur qui contient un tas pour les paramètres d'appel et de réponse.

Schéma général de l'appel de procédure distante léger



Avantages Inconvénients: RPC léger

- **Avantages**
  - Transmission d'appel très performant comme mode de RPC local.
- **Inconvénients**
  - Uniquement applicable aux RPC du même site.

Conclusion réalisation de l'appel de procédure distante

- L'appel est d'abord développé en invocation distante par messages.
  - Supporte l'hétérogénéité
  - Finalement le plus simple à réaliser.
  - RPC, DCE, CORBA, RMI, DCOM, SOAP
- Des optimisations peuvent être obtenues par l'usage opportun des autres solutions.
  - Exemple: Chorus a développé les quatre solutions.
  - Exemple: DCOM RPC par messages + RPC léger (aussi prévu en CORBA).

Gestion du contrôle

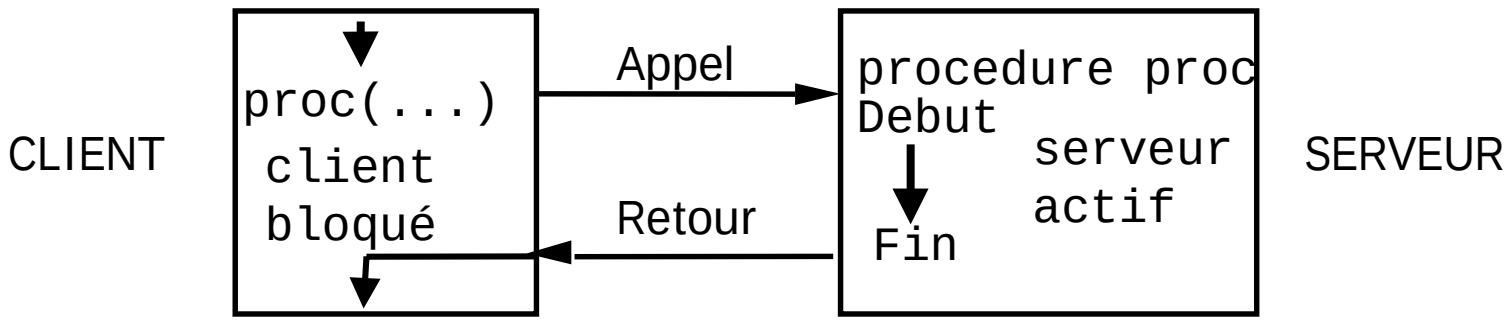
I) Parallélisme chez le client

II) Parallélisme chez le serveur

III) Structures de contrôle réparti par composition d'appels distants

I) Parallélisme chez le client : Appel de procédure distante en mode synchrone

- L'exécution du client est suspendue tant que la réponse du serveur n'est pas revenue ou qu'une condition d'exception n'a pas entraîné un traitement spécifique.

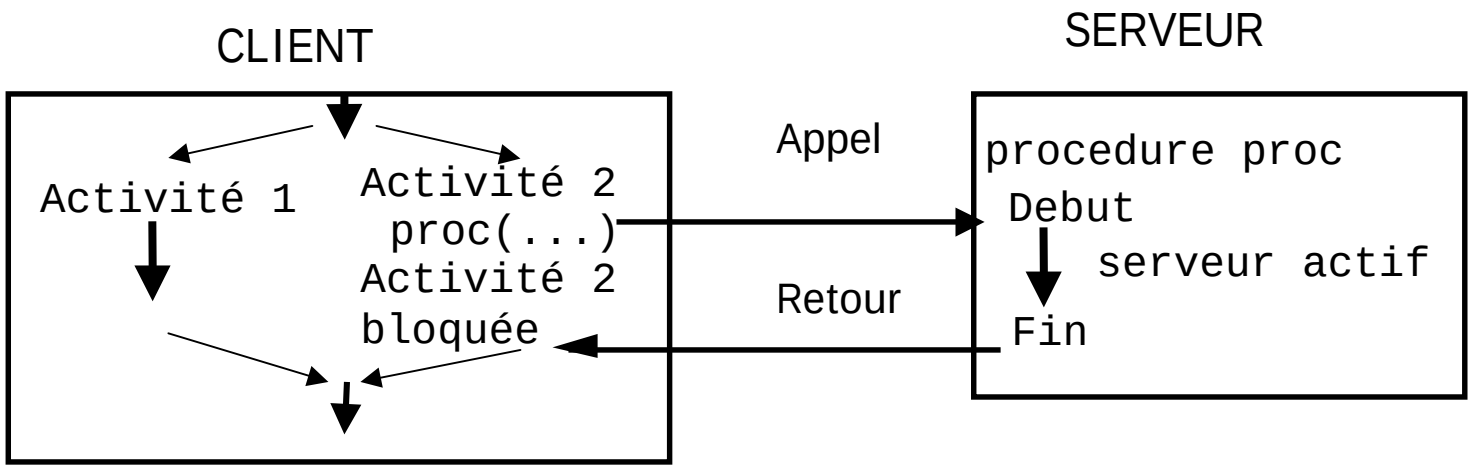


- **Avantage:** le flot de contrôle est le même que dans l'appel en mode centralisé
- **Inconvénient**: le client reste inactif.

Une solution au problème de l'inactivité du client: utilisation des activités

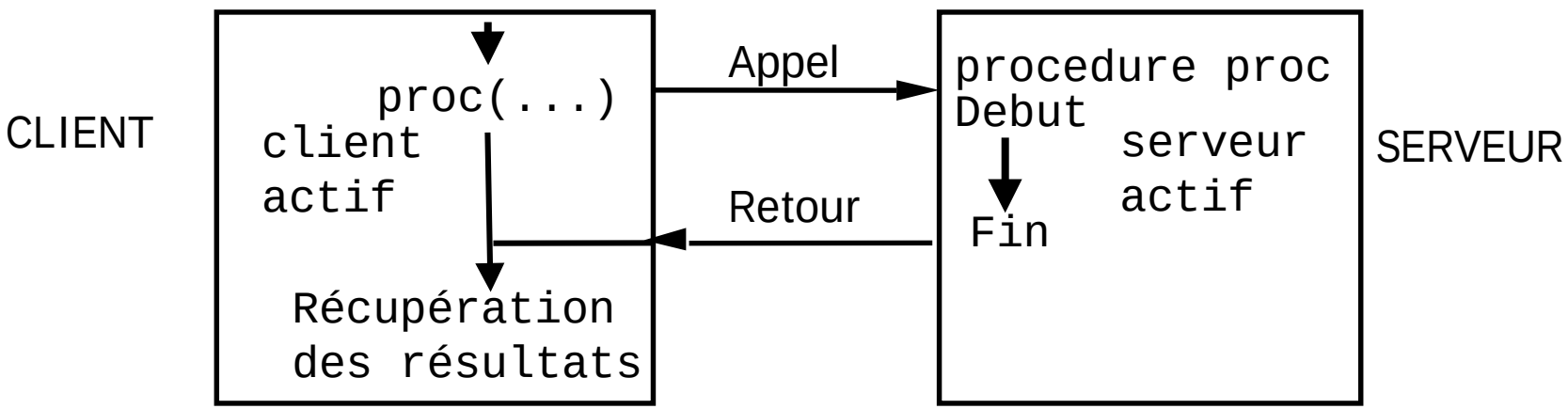
Création de (au moins) deux activités (‘ threads ’) sur le site client

- **L'une occupe le site appelant** par un travail à faire.
- **L'autre gère l'appel en mode synchrone** en restant bloquée: Le fonctionnement est exactement celui d'un appel habituel.



Appel de procédure distante en mode asynchrone

- Le client poursuit son exécution **immédiatement après** l'émission du message porteur de l'appel.
- La **procédure distante s'exécute en parallèle** avec la poursuite du client.
- **Le client doit récupérer les résultats** quand il en a besoin (primitive spéciale).



Récupération des résultats en mode asynchrone: notion de futurs explicites

- Un **futur** : une structure de donnée (un objet) permettant de récupérer des résultats.

  - **Notion de futur explicite**

- Les structures de données sont définies par le client avant l'appel (le serveur les connaît et y dépose les résultats).
- Exemple: En ACT++ une boite à lettre définie par le client sert de moyen de communication pour les paramètres résultats.
- BAL := factorielle.calculfact(n)
- resultat := BAL.prélever()

Récupération des résultats en mode asynchrone: notion de futurs implicites

- **Invocation asynchrone à futur implicite**

- Les structures de données de communication pour les réponses (ex boite à lettre) sont **implicitement créés** par le prestataire du service d'APD asynchrone.
- Approche générale: **défaut d'information** (analogie défaut de page en mémoire virtuelle).
- La lecture de la structure de donnée résultat **bloque le client** s'il accède à la réponse et que celle-ci n'est pas parvenue.
- L'usager ne s'aperçoit de rien (si le résultat est parvenu ou s 'il n'est pas parvenu).

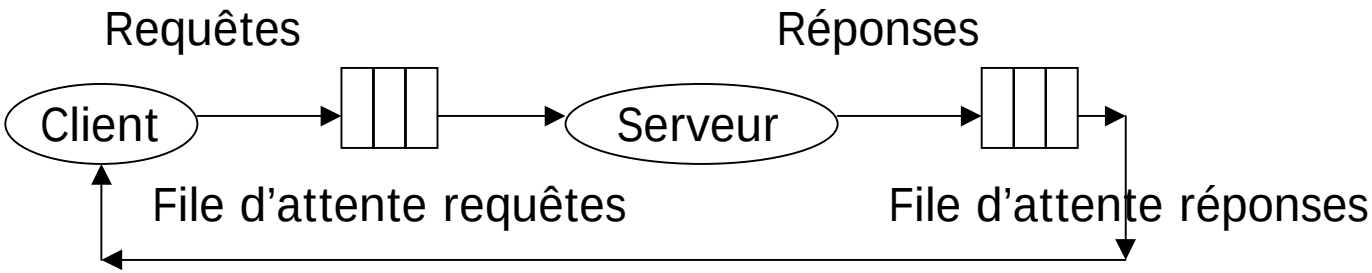
Cas particulier du mode asynchrone: invocation asynchrone à sens unique

Invocation asynchrone sans réponse (autre terminologie, "peut-être", "oneway" , "maybe")

- Invocation asynchrone utilisé pour déclencher une procédure qui ne retourne pas de résultats. Pour obtenir un dialogue il faut prévoir d'autres procédures en sens inverse.
- **Avantage**: Utilisation d'un mode appel de procédure pour des communications sont en fait de mode message.
- **Inconvénients** : Uniquement possible en l'absence de retour de résultat, pas d'informations sur la terminaison du travail demandé.
  - Exemples: CORBA oneway.

II) Parallélisme chez le serveur : Exécution séquentielle des appels

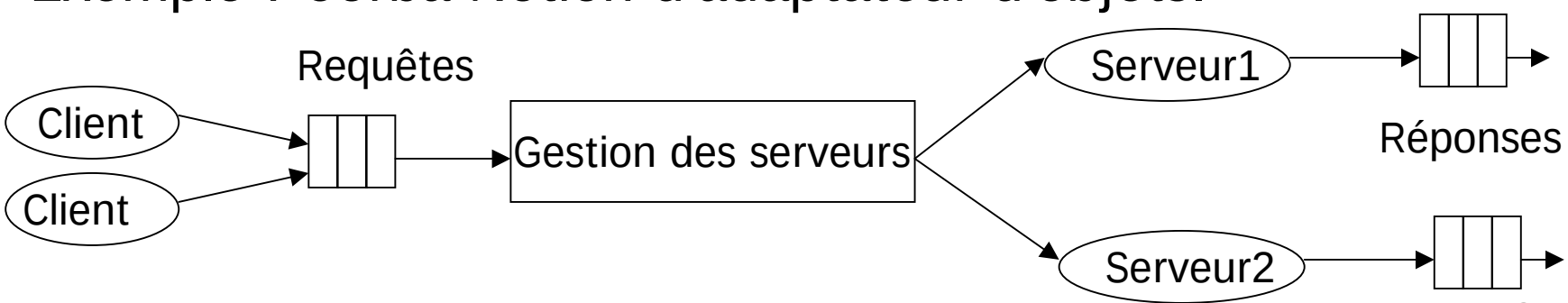
- Les requêtes d'exécution sont traitées l'une après l'autre par le serveur: **exclusion mutuelle entre les traitements**.
- Si la couche transport assure la livraison en séquence et que l'on gère une file d'attente premier arrivé premier servi, on a un traitement **ordonné** des suites d'appels.



- **Exemple RPC SUN** : traitement séquentiel des requêtes mais utilisation de UDP => requêtes non ordonnées (mais mode synchrone le client attend la fin du traitement).
- Autres exemples: les RPC ont un mode séquentiel (ex CORBA)

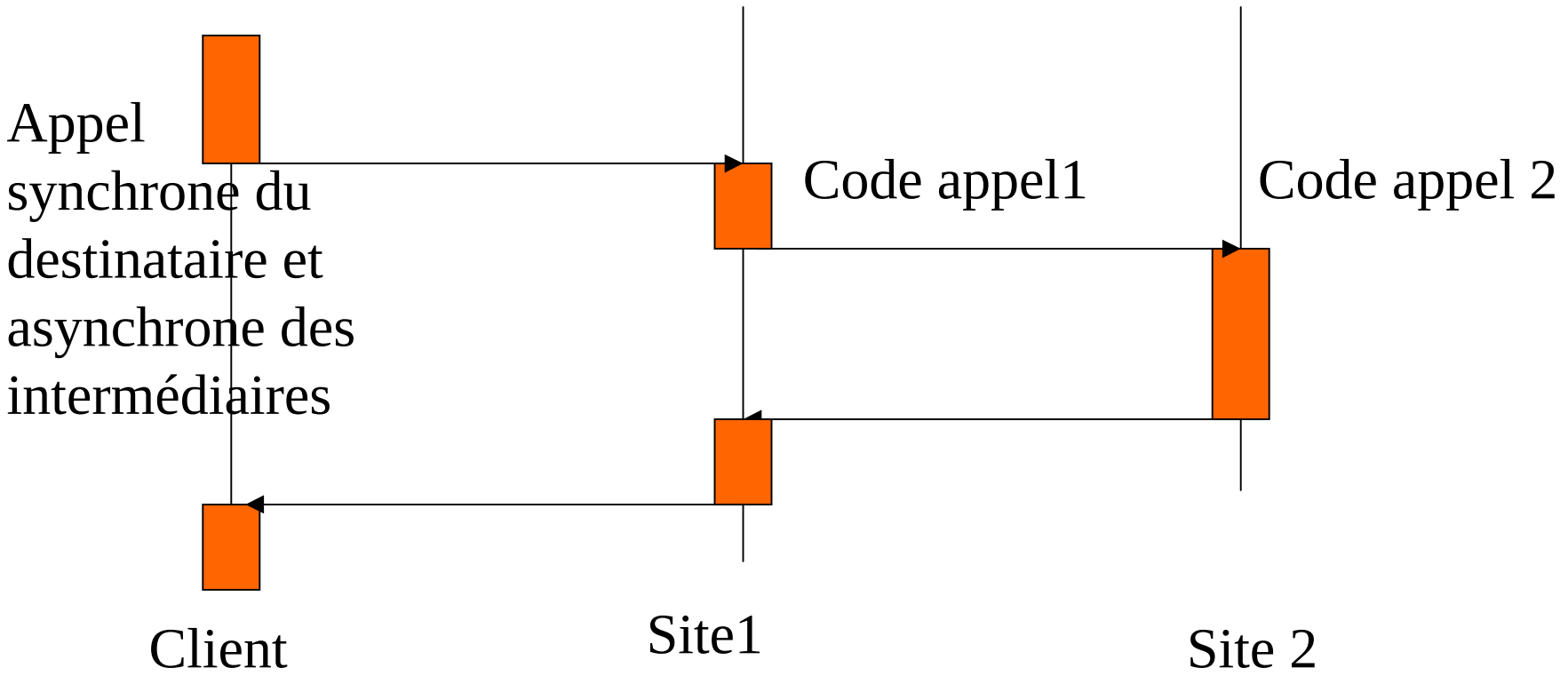
Exécution parallèle des appels

- Dans ce mode le serveur créé un **processus** ou une **activité** (un processus léger ou “thread”) par appel (gestion possible de pool de processus ou d ’activités).
- Les appels sont exécutés **concurremment.**
- Si les procédures manipulent des données globales rémanentes sur le site serveur, le **contrôle de concurrence doit être géré.**
- Exemple : Corba Notion d’adaptateur d’objets.



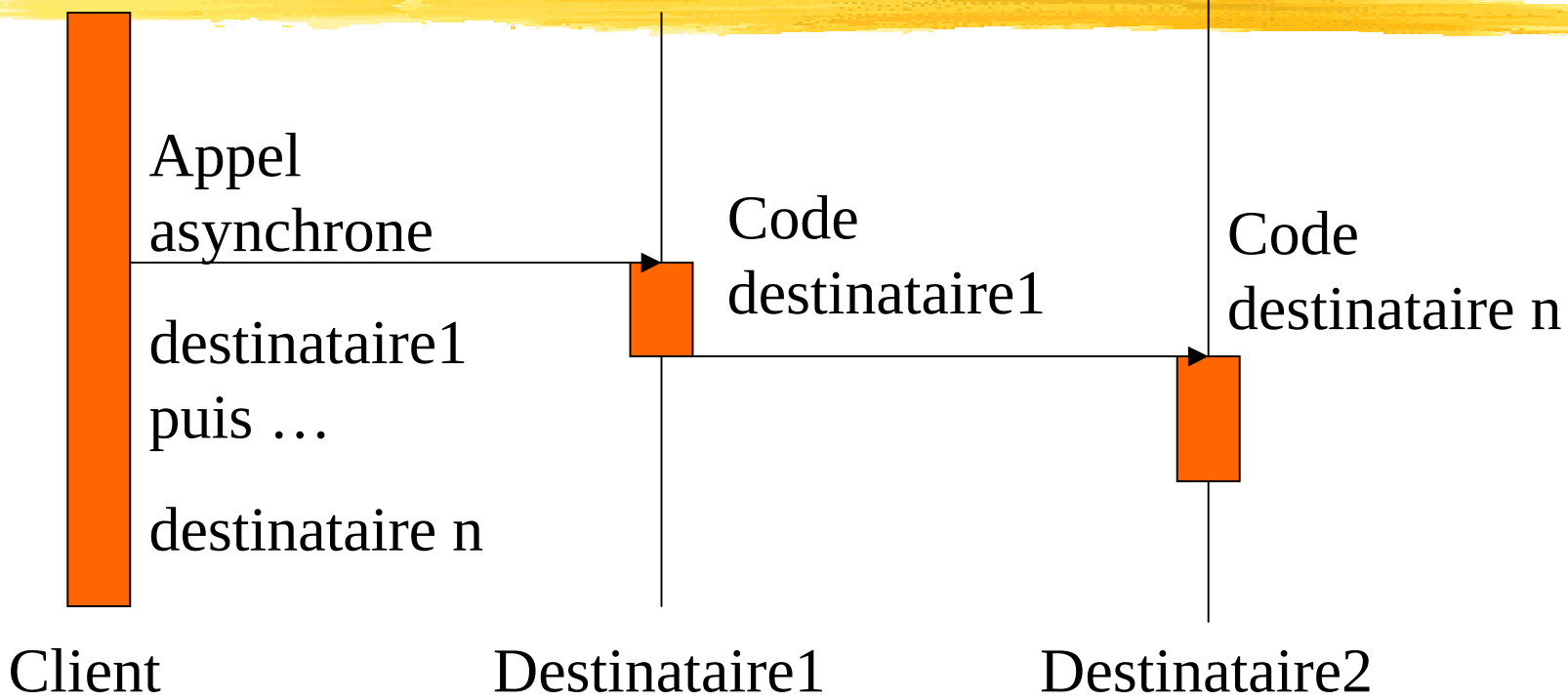
III) Schémas de contrôle: composition d'appels distants en séquence

A) Schéma appel en cascade synchrone



Déduit immédiatement de l'appel de procédure distante synchrone

B) Schéma de continuation asynchrone: appels en cascade asynchrone

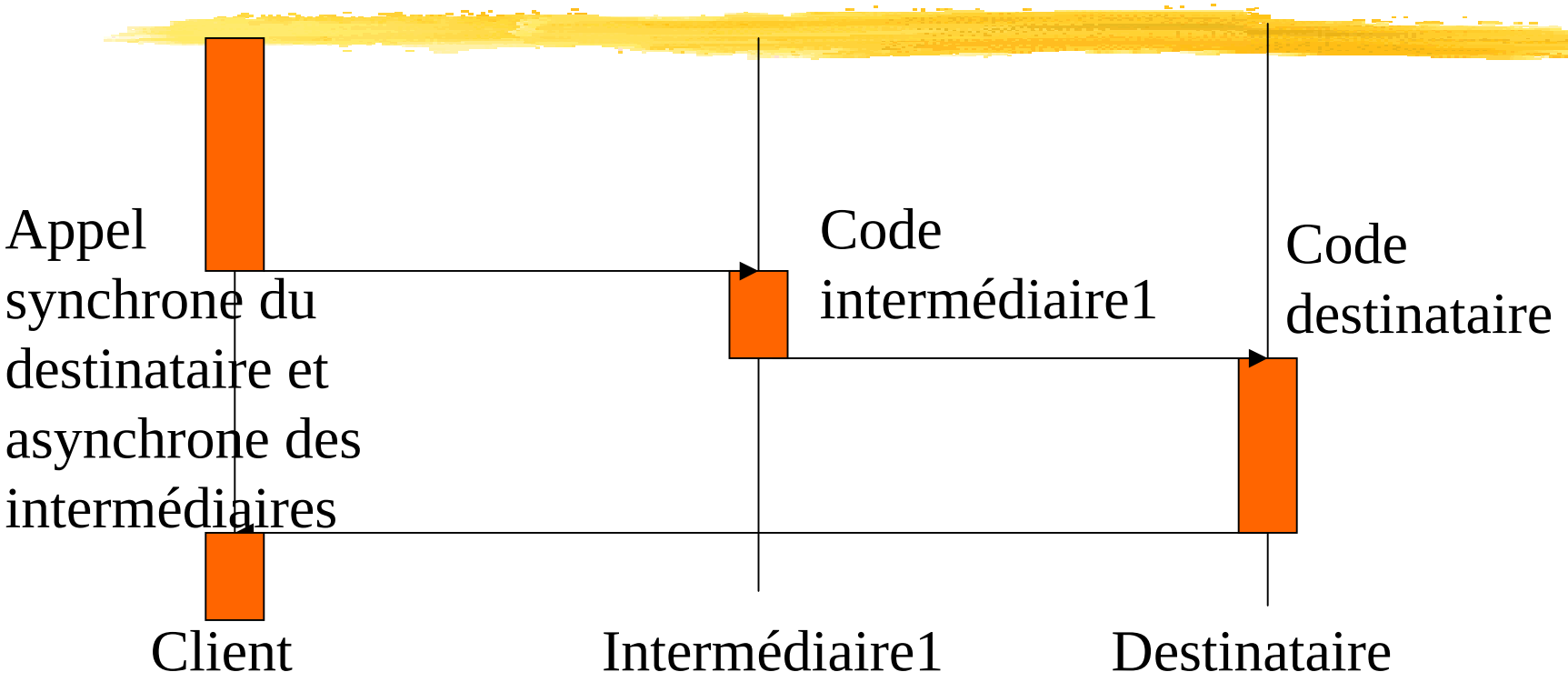


Baptisé schéma à continuation en mode asynchrone. L'émetteur prépare une liste de procédures destinataires à invoquer en mode asynchrone. Le message d'appel visite successivement les destinataires.

Analogie avec le routage par la source en mode message

Implantation: protocole SOAP en mode message.

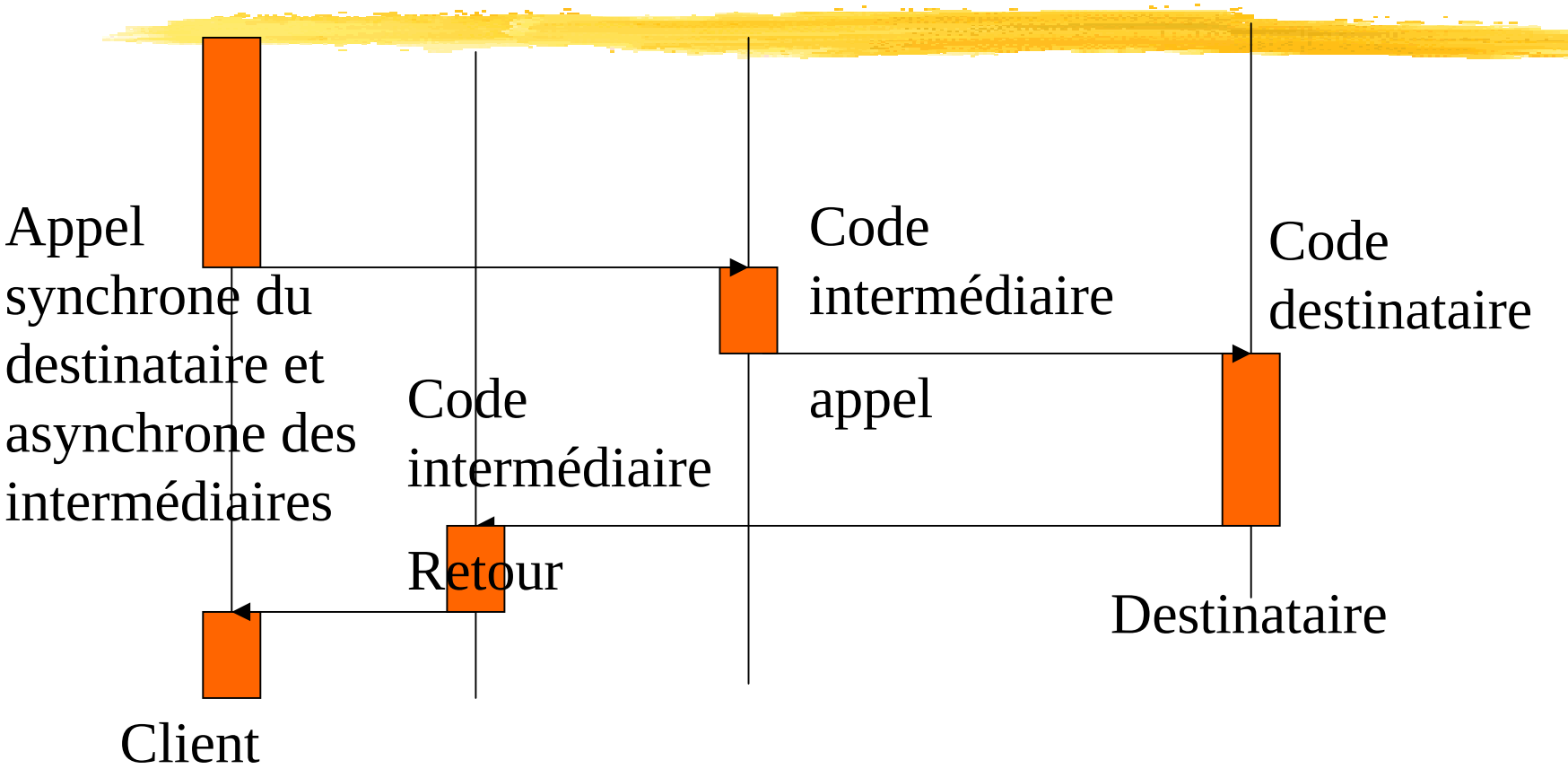
C) Schéma de continuation asynchrone avec appel synchrone pour le client



Le premier schéma à continuation proposé. Une liste d'intermédiaires en mode asynchrone et un destinataire final : le tout en mode synchrone pour le client.

Implantation: protocole SOAP en mode RPC.

D) Schéma de continuation asynchrone en appel et en réponse

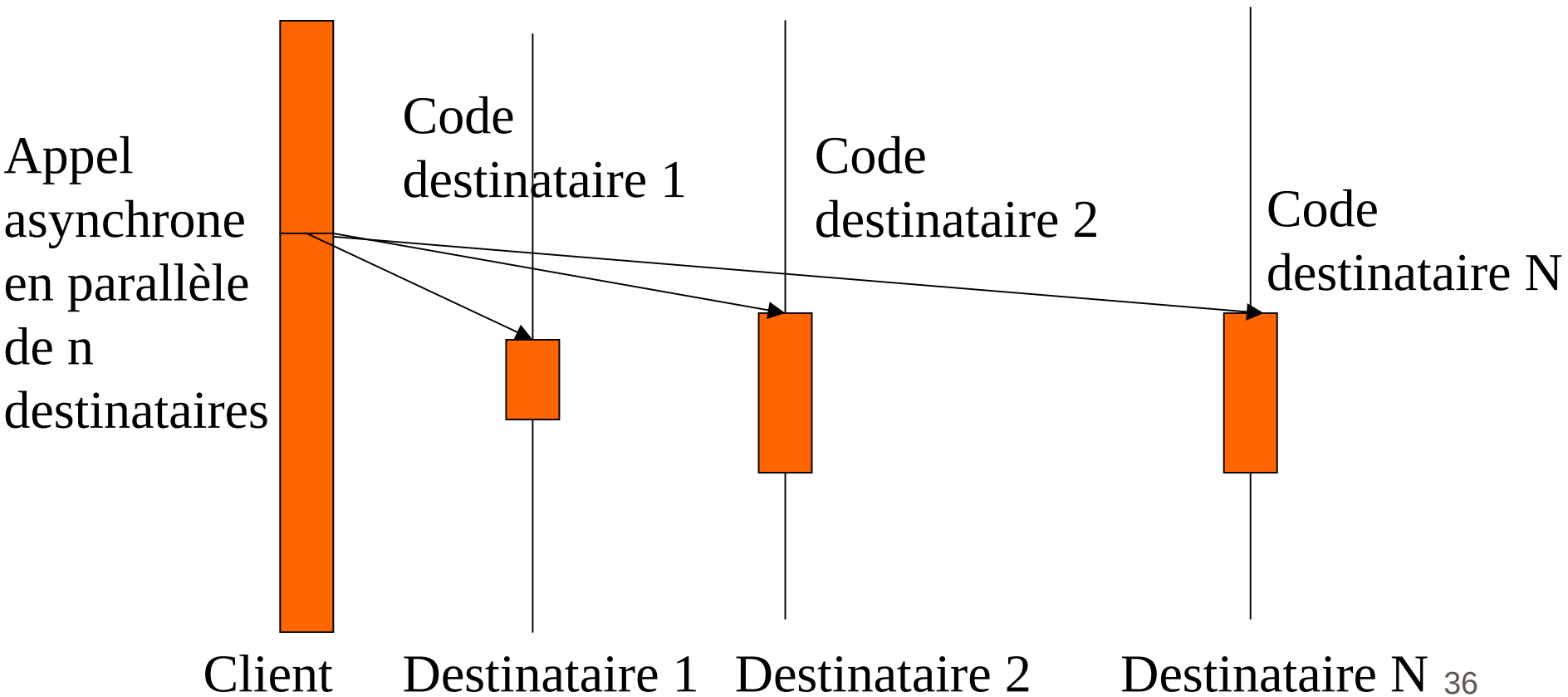


Une liste d'intermédiaires en mode asynchrone est possible à l'appel comme à la réponse : le tout en mode synchrone pour le client.

Implantation: protocole SOAP en mode RPC.

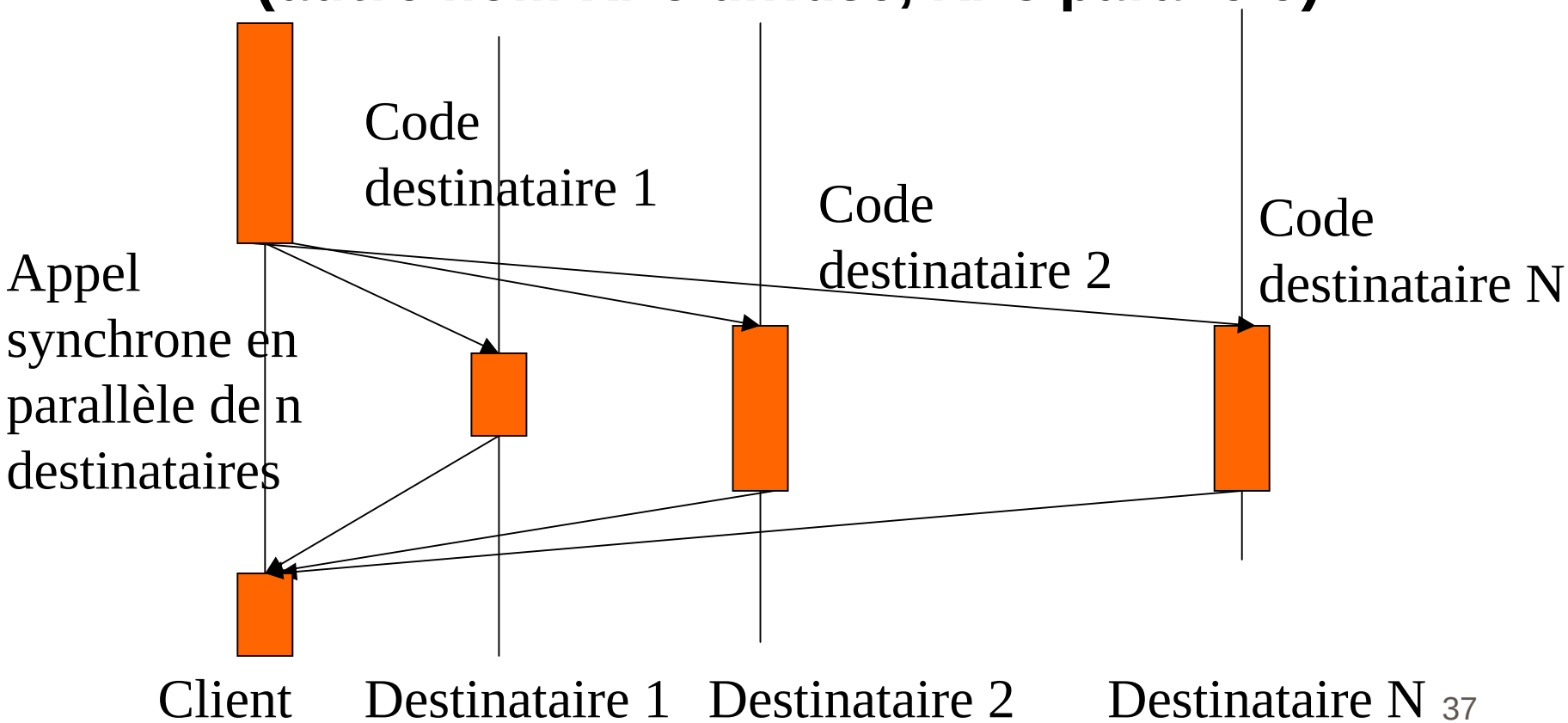
Schémas de contrôle par composition parallèle d'appels distants (1)

Notion de RPC asynchrone sur groupe



Schémas de contrôle par composition parallèle d'appels distants (2)

Notion de RPC synchrone sur groupe
(autre nom RPC diffusé, RPC parallèle)



Propriétés d'ordre dans les communications par RPC

- L'appel de procédure distante peut comporter des **spécifications de propriétés d'ordre**.
- Le respect d'une propriété d'ordre peut porter:
  - sur **le lancement** de la procédure (peut utilisable)
  - sur **la totalité de son exécution**.
- **Ordre local:** Les exécutions pour un client sont réalisées dans l'ordre d'émission.
- **Ordre global:** Les exécutions pour un client sont réalisées dans le même ordre sur tous les destinataires (cas des communications de groupe).
- **Ordre causal:** Les exécutions sont effectuées en respectant la relation de causalité qui existe entre les requêtes.

Gestion des données

I) Gestion des données applicatives persistantes.

II) Gestion des données protocolaires persistantes.

Gestion des données applicatives sans données partagées persistantes

- Données locales à la procédure : pas de problème.
- Données applicatives partagées (variables d'instance, fichiers, bases de données) : problème de persistance.

Sans données partagées persistantes

- Situation idéale du cas ou l'appel de procédure s'exécute en fonction uniquement des paramètres d'entrée: en produisant uniquement des paramètres résultats.
- Exemple: fonction scientifique (**EJB session stateless**).
- Il n'y a **pas de modification de données rémanentes** sur le site serveur.
- **Pas de problème** pour la **tolérance aux pannes** et pour le **contrôle de concurrence**

Gestion des données applicatives partagées persistantes

Les **exécutions successives** manipulent **des données persistantes** sur le site serveur.

- **Une exécution modifie le contexte sur le site distant** (un serveur de fichier réparti, de bases de données. Opérations d'écriture de données persistantes, la structure de donnée manipulée par les méthodes d'un objet).

=> Problème de contrôle de concurrence.

=> Problème des pannes en cours d'exécution.

- **Solution**: le couplage d'une gestion transactionnelle avec une approche RPC (ou système d'objets répartis).
- Exemple: **EJB Session stateful** (un seul client), **EJB Entity** (plusieurs clients)

Gestion des données protocolaires: notion de mode avec ou sans état

- Autre aspect de la **rémanence** des données sur le serveur.
- La terminologie avec ou sans état porte sur **l'existence ou non d'un descriptif** pour chaque relation client serveur au niveau du serveur.
- **Notion d'état** : un ensemble de données rémanentes au niveau du protocole pour chaque relation client serveur.
  - Permettrait de traiter les requêtes dans l'ordre d'émission.
  - Permettrait de traiter une requête en fonction des caractéristiques de la relation client serveur (qualité de service).
- En fait une notion identique à celle du **descriptif de connexion** chez le serveur dans une communication en mode connecté.

Mode sans état

- Les appels successifs d'une même procédure s'exécutent **sans liens** entre eux.
- Il peut y avoir modification de données globales rémanentes sur le site serveur mais **chaque opération** du point de vue du protocole s'effectue sans référence au **passé** (indépendamment de toutes celles qui ont précédé).
- Ex: **NFS** ' Network File System ' de SUN système de fichier réparti basé sur RPC sans état. Lecture/Écriture du nième article d'un fichier dont toutes les caractéristiques utiles (nom, droit d'accès) sont passées au moment de l'appel.
- Ex: **HTTP** ' HyperText Transfer Protocol ' protocole d'exécution de méthodes distantes sans état.

Mode avec état

- Les appels successifs s'exécutent en **fonction d'un état de la relation client serveur** laissé par les appels antérieurs.
- Exemple d'utilisation : la gestion de l'ordre de traitement des requêtes, la gestion de caractéristiques du client.
- Exemple système de fichier en RPC: Lecture d'article de fichier sur le pointeur courant.

Mode avec ou sans connexion

- Rappel **gestion des connexions:**
  - Délimitation temporelle des requêtes et des exécutions de procédure entre des opérations d'ouverture et de fermeture.
  - Maintien d'un descriptif de connexion sur le client et sur le serveur pour la gestion de paramètres caractéristiques de la connexion : qualité de service, traitement des pannes, propriétés d'ordre, ...)
  - Définition d'une référence (d'une désignation) de connexion

Appel de procédure distante sans connexion

- Dans ce mode le client peut envoyer des appels au serveur à n'importe quel moment.
- Le client comme le serveur ne gèrent pas de descriptif de la relation client serveur : **absence de mémoire** entre appels successifs.
- Le mode sans connexion est donc un mode orienté
  - Vers un **traitement sans gestion de qualité de service** des appel.
  - Vers le traitement **d'un grand nombre de clients**: la gestion de connexions est jugée trop coûteuse.
- Exemple : tous les RPC

Appel de procédure distante avec connexion

- Dans ce mode le client doit **ouvrir une connexion avec le serveur** pour effectuer des appels puis il doit fermer.
- Exemple : prototypes recherche, pas d'exemple industriel connu.
- Si l'on souhaite gérer des RPC en mode connecté il faut construire cette gestion dans le cadre d'une application donnée.

Transmission des arguments

I) La transmission par valeur

II) Les autres modes de passage

Transmission par valeur: rappels (1)

- Le seul mode de transmission des données dans les messages en réseau.
- **Si le site client et le site serveur utilisent des formats de données différents** : problème syntaxique => une conversion est nécessaire.
- **La solution: notion de présentation des données** au moyen d'une syntaxe de transfert => une **représentation lexicale** des données simples et une **convention d'alignement** des données commune au client et au serveur.
- Dans le domaine des communications en mode message: norme **de représentation lexicale des données simples BER (' Basic Encoding Rules ')**.

Transmission par valeur: rappels (2)

- **Définir une syntaxe abstraite de données pivot**: analogue des langages de description de données des langages évolués, facile à générer pour un développeur d'application (en mode message ASN1 ' Abstract Syntax Notation ' 1).
- A partir de la syntaxe abstraite: **codage / décodage** de la **syntaxe de transfert** (technique de compilation, notion de compilateur de protocole).

Transmission par valeur dans l'appel de procédure distante

- En appel de procédure distante : **génération automatique du code des souches à partir de la syntaxe abstraite**, les souches fabriquent **la syntaxe de transfert** en réalisant l'alignement des paramètres dans les messages (' marshalling ').
- **Insuffisance des normes** de syntaxe abstraite et de syntaxe de transfert en mode message asynchrone: ASN1/BER
- **Définition de nouveaux langages de syntaxe abstraite** adaptés aux appels de procédure distante:

IDL ' Interface Définition Language '

- Pour chaque IDL en général redéfinition **d'une syntaxe de transfert .**

Motivation pour une classe de langages de syntaxes abstraites : IDL

- Être indépendant des langages évolués utilisant le RPC
- Permettre l'invocation distante avec tout langage évolué
  - Définition d'un langage pivot de description de données ayant des fonctionnalités assez riches pour les langages les plus récents.
  - Définition d 'un langage pivot qui permette de corriger les ambiguïtés et les insuffisances des langages anciens (comme par exemple C).
  - Notion de correspondance entre les types retenus dans l'IDL et les types des différents langages existants ("mappings")

Motivation pour une classe de langages de syntaxes abstraites : IDL

- Permettre aux utilisateurs de définir un identifiant unique pour chaque interface afin de l'utiliser .
- Supporter les caractéristiques particulières des langages objets
  - Exemple : permettre de définir des relations d'héritage entre définitions d'interfaces.
- Éventuellement structurer les interfaces en modules.

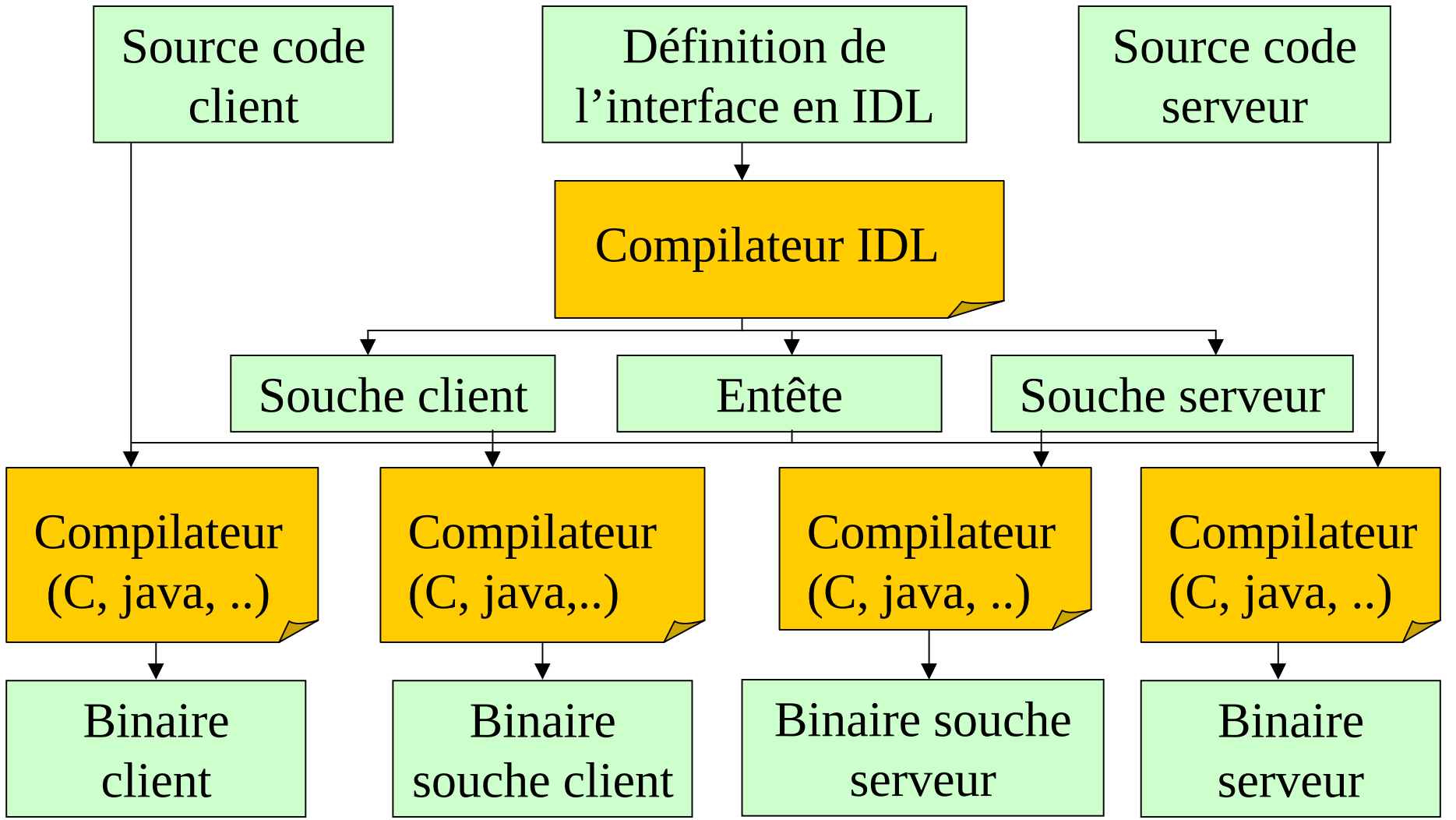
Un exemple en IDL DCE

```
/* Exemple documentation OSF Open Software Foundation */

[uuid (f9f8be80-2ba7-11c9-89fd-08002b13d58d), version(0),
    endpoint("ncadg_ip_udp:[6677]", "dds:[19] »)]

interface bin_op

{
    [idempotent] void bin_add
    {       [in] handle_t h,

            [in] long a,

            [in] long b,

            [out] long *c

    };

}
```

Un exemple en IDL Corba

```
module StockObjects {
  struct Quote {
    string symbol;
    long at_time;
    double price;
    long volume;};
  exception Unknown{};
  interface Stock {
    Quote get_quote() raises(Unknown); // lit la cotation.
    void set_quote(in Quote stock_quote); // écrit la cotation
};
  interface StockFactory {
    Stock create_stock(
      in string symbol,
      in string description
);
};
};
```

Génération des souches



Quelques exemples d'IDL et de format de présentation en RPC

- SUN ONC/RPC
  - XDR eXternal Data Representation
- OSF DCE
  - IDL DCE - Format NDR Network Data Representation
- OMG CORBA
  - IDL Corba - Format CDR Common Data Representation, Protocole IIOP
- SUN Java RMI
  - Langage Java - Protocole JRMP Java Remote Method Protocol
- Microsoft - DCOM
  - MIDL Microsoft IDL - DCOM Protocole ORPC Object RPC Format NDR
- WS-SOAP
  - WSDL Web Services Definition Language - XML

II) Autres modes de transmission : le passage par adresse (pointeurs)

- **Le passage par adresse (par pointeur)** utilise une adresse mémoire centrale du site de l'appelant qui n'a aucun sens sur l'appelé (sauf cas particulier).
- **Indispensable de traiter ce problème.**

A) Interdiction totale des pointeurs

- Dans la plupart des RPC, **pas de passage des paramètres** par adresse ou de structures de données contenant des pointeurs.
- Introduit une **différence** dans le développement de procédures locales et celles destinées à un usage distant.

B) Simulation du passage par adresse en utilisant une copie restauration.

- **Marche bien dans beaucoup de cas mais** violation assez inacceptable dans certains cas de la sémantique du passage.
- **Exemple de problème**
  - **procédure** double_incr (x , y) ;
  - x , y : **entier** ;
  - **début**
  - x := x + 1 ;
  - y := y + 1 ;
  - **fin ;**
  - **Séquence d'appel: passage par adresse**
  - a := 0 ; double_incr (a , a) ;
  - Résultat attendu : a = 2
  - **Utilisation d'une copie restauration**
  - Résultat obtenu : a = 1

C) Passage par adresse en mémoire virtuelle partagée répartie

- Faire réaliser des accès mémoire inter-sites : **mémoire virtuelle partagée répartie**
- Solution **très satisfaisante** mais également **très coûteuse**.
  - **Appel**: Défauts d'information mémoire serveur -> mémoire client
  - **Retour**: Défauts d'information mémoire client -> mémoire serveur
- Nécessité d'un **système réparti autorisant la mémoire virtuelle répartie (Chorus, Mach).**
- Ne se conçoit que dans un système réparti de **calculateurs de même type**.

Passage par nom (références ou pointeurs d'objets)

- En approche objets répartis **utilisation des références d'objets.**
- Le passage d'un argument se fait en transmettant une **référence** sur l'objet en univers réparti (en CORBA IOR).
- L'accès s'effectue au **moyen des méthodes implantées par l'objet**. Par exemple accès à un attribut en lecture ou en écriture.

- **Avantages inconvénients**

- Mécanisme universel.
- Coûteux pour des accès fréquents.

Désignation et liaison

Désignation

- Comment sont structurés les noms et références permettant de désigner les services distants:
  - Non symbolique : utilisable au moyen d'un serveur d'annuaire.
  - Référence : une structure de données permettant de réaliser l'invocation.
- Référence : selon l'implantation considérée.
  - Désignation du protocole permettant l'accès distant (TCP)
  - Désignation de l'hôte ou se trouve le serveur (adresse IP).
  - Désignation du point d 'accès de service transport (numéro de port)
  - Désignation locale de l'objet ou se trouve la procédure.
  - Désignation de la procédure.

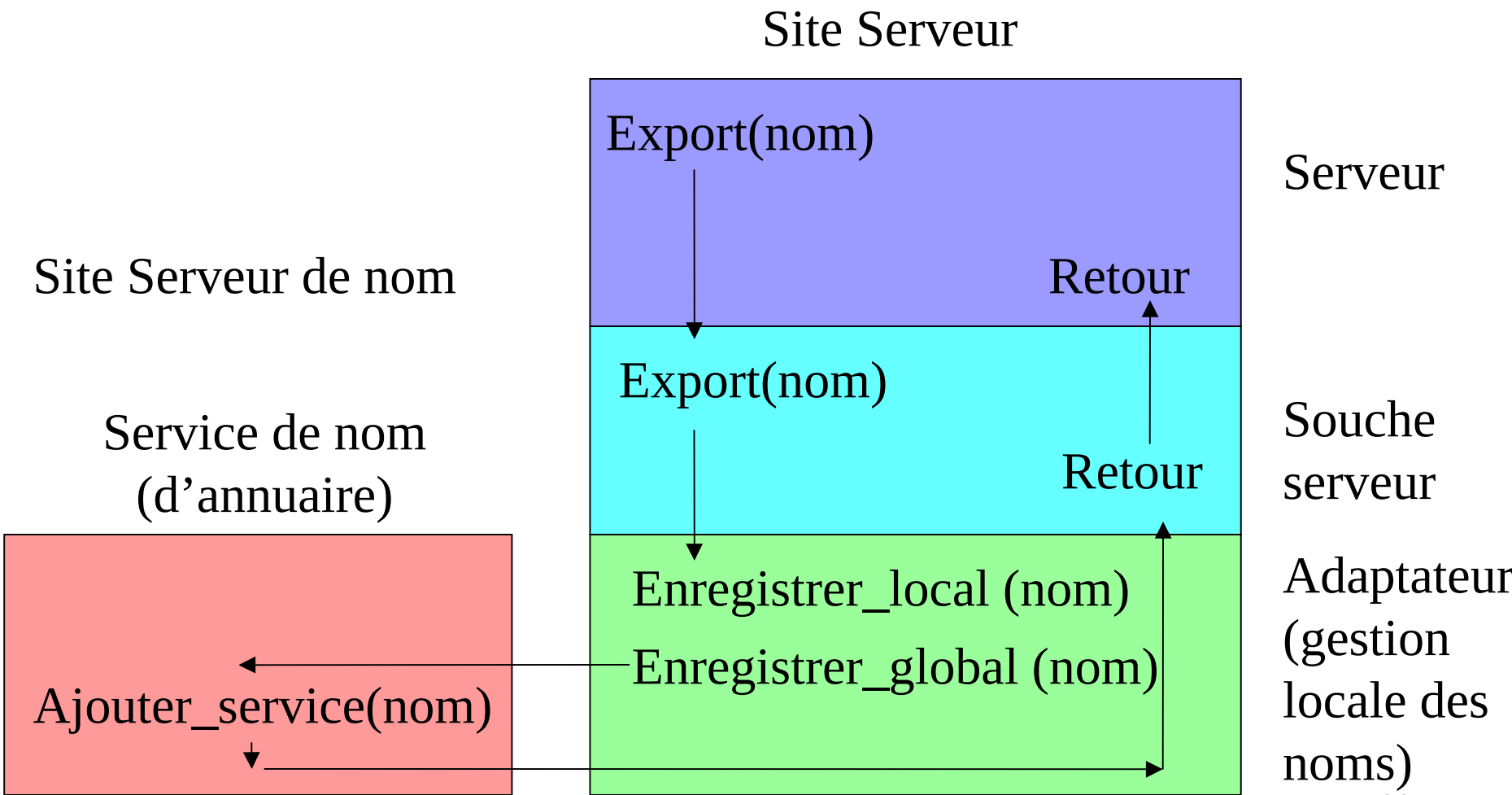
Liaison

- Comment localiser une procédure distante?
  - Au moyen d'un service de gestion de noms (serveur d'annuaire)
- A quel moment effectuer la liaison
  - Le plus tard plus possible -> plus souple, moins performant
  - Le plus tôt possible -> plus performant
- Réalisation de la liaison à la compilation
  - Vision très statique, le serveur ne doit pas bouger.
- Réalisation de la liaison en exécution
  - Au chargement du client
  - Au moment de la première invocation
  - Localiser la procédure (référence), mettre en cache
  - Relocaliser à chaque fois qu'il est nécessaire

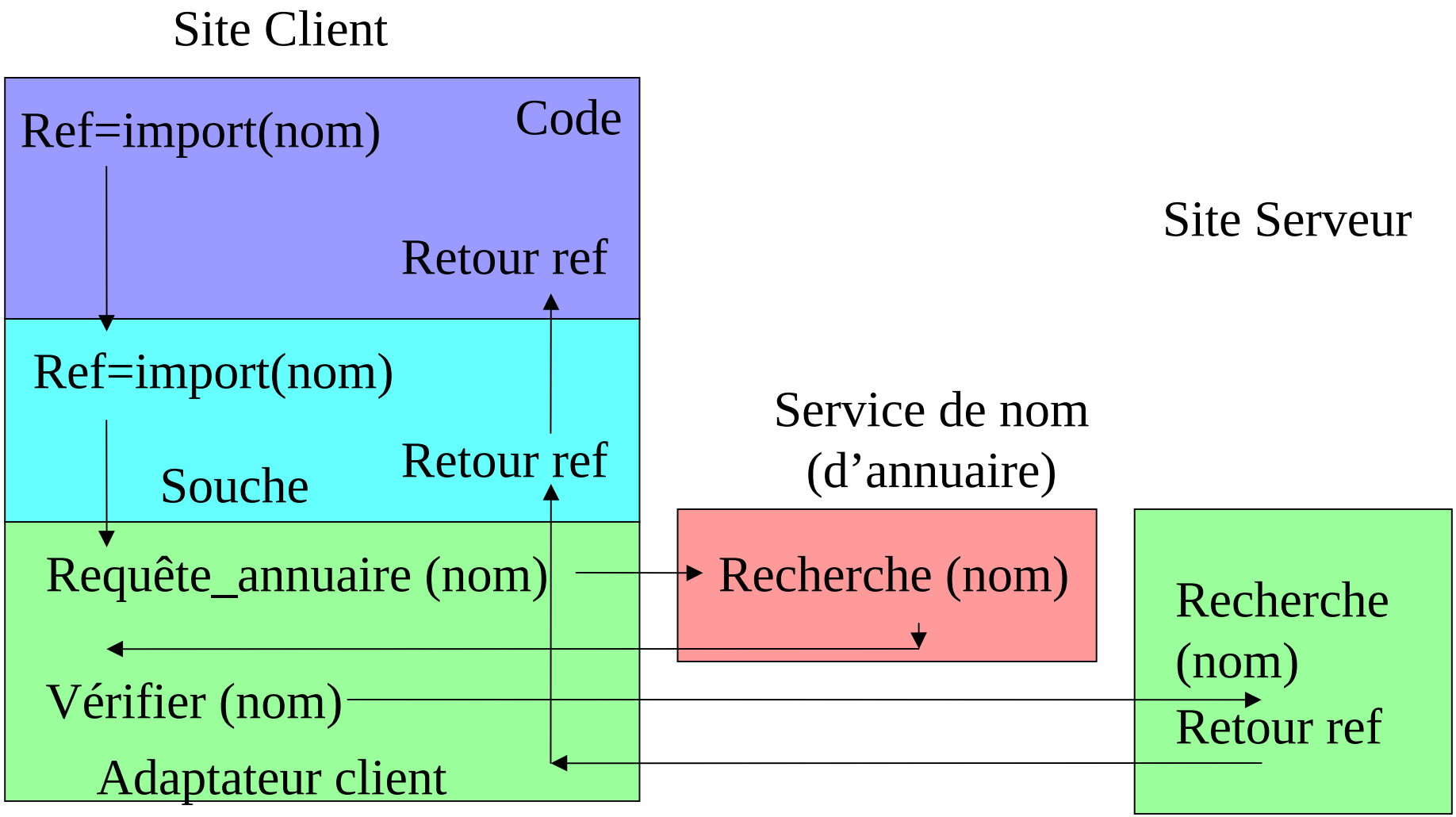
Problèmes de la Liaison

- Le serveur est-il disponible?
  - Solution - fabrique de serveurs
- Est ce que la version de l'interface utilisée est consistante?
  - Solution - gestion d 'identificateurs uniques d'interface (comportant des numéros de version).
- Existence de serveurs multiples
  - Gestion d 'une technique d'équilibrage de charge.
  - Gestion de redondances (temporelles, massives).

Enregistrement d'un serveur



Liaison client serveur



Tolérance aux pannes

Comparaison appel local appel distant

En appel local

- L'appelant et l'appelé s'exécutent sur la même machine: même exécutant => mêmes performances et modes de pannes.
- L'appel et le retour de procédure sont des mécanismes internes considérés comme fiables (sauf systèmes sécuritaires).
- Problème abordé les erreurs => exceptions chez l'appelant.

En appel distant

- Appelant et appelé peuvent tomber en panne indépendamment.
- Le message d'appel ou celui de retour peuvent être perdus (sauf si l'on emploie un protocole de transport fiable).
- Le temps de réponse peut-être très long en raison de surcharges diverses (réseau, site appelé).

Les différents modes de pannes : les pannes franches du serveur

- 1 - Attente indéfinie par le client d'une réponse qui ne viendra jamais => Abandon du programme sur temps limite
- 2 - Utilisation d'un délai de garde par le site appelant

=> Signalement de l'absence de réponse au client qui décide de la stratégie de reprise.

=> Reprise arrière automatique (si possible)

soit tentative d'exécution sur un autre site

soit reprise sur le même site (si relance)

=> **Plusieurs exécutions successives de la même procédure pour une seule demande.**

Les différents modes de pannes : les pannes franches du client

- Le serveur est dit **orphelin (“orphan”).**
- **Réalisation de travaux inutiles**, utilisation de ressources qui ne sont plus accessibles pour des tâches utiles.
- **Confusion après relance du client** entre les nouvelles réponses attendues et les réponses à d'anciennes requêtes.
- => Nécessité de détruire les tâches serveurs orphelines et de distinguer les requêtes utiles des vieilles requêtes.

Problème des pannes franches

- L’état du client ou du serveur peuvent devenir inconsistants
- => Analyse des modifications de l’état du client et du serveur.

Les différents modes de pannes : les pertes de messages

- Cas facile: pertes traitées par une couche transport fiable.
- Cas difficile: il existe des RPC implantés pour des raisons d'efficacité au dessus d'une communication non fiable (couche réseau sans connexion type UDP) => Mécanisme de traitement des pertes de message à prévoir dans la conception du RPC
  - Ex : La réponse acquitte la demande
  - La prochaine requête acquitte la réponse.
- **Pour tolérer les pertes de messages on peut lancer plusieurs exécutions de la même requête**.

Les différents modes de pannes : les pannes temporelles

- Cas d'un appel de procédure utilisé sous **contraintes de qualité de service temporelle**
  - contraintes d'une application temps réel (contrôle de procédé industriel)
  - contraintes d'une application multimédia.
- **Temps de réponse**: La réponse doit parvenir avant une date limite définie par une échéance.
- **Variation du temps de réponse**: La variation du temps de traitement doit être bornée.
- Notion d'ORB (' Object request Broker ') temps réel en cours d'étude. Gestion de priorités au niveau réparti, gestion d'échéances intégrant les délais réseaux.

Les différents modes de pannes : les pannes quelconques

- Cas d'un appel de procédure utilisé sous **contraintes de criticité (système critique)**
- **Pas de spécificité particulière** des protocoles de RPC
- **Application des techniques habituelles de redondances massives**: pour tolérer les défaillances d'un (ou de plusieurs) des serveurs redondants.

Formalisation des techniques de reprise sur panne franche

- **Procédure F appelée à distance:**

  F

  (C_avant, S_avant) ----> (C_après, S_après)

  État des données État des données

  client serveur avant client serveur après

- **F dans son traitement modifie l'état du serveur.**

  (S_avant) -----> (S_après)

- **Les résultats de l'exécution de F en retour modifient aussi l'état du client.**

  (C_avant) -----> (C_après)

- **Techniques de reprise sur erreur :** en cas de problèmes rééxécution: $F^{(0)}$, $F^{(1)}$,...., $F^{(n)}$ les tentatives successives

A) Reprise arrière complète

- Les exécutions successives du serveur peuvent laisser l'état du serveur indéterminé (panne en cours d'exécution).
- Le client peut être dans un état incertain (panne au moment de la modification en retour des variables persistantes du client).
- Solution maximale => Pour chaque tentative **l'état du client et celui du serveur doivent être restaurés** aux valeurs avant le premier appel:
- $F^{(n)}$
- (C_avant, S_avant) --> (C_après$^{(n)}$, S_après$^{(n)}$)

Validité de la reprise arrière complète

Cas d'une procédure déterministe

- La reprise complète est correcte quelles que soient les pannes.
- Les états après sont identiques puisque les exécutions répétées du code du serveur sur la même donnée produisent des résultats identiques (la procédure produit un **résultat déterministe).**

Cas d'une procédure indéterministe

- Les états **après sont potentiellement différents** puisque les exécutions répétées donnent des résultats qui dépendent du contexte d'exécution.
- La reprise est correcte si les aléas conduisant à l'indéterminisme du fonctionnement sont acceptés.

B) Reprise arrière du serveur seul

- **Hypothèse: Les rééxécutions du serveur sont toujours complètes** ou équivalent à une exécution complète correcte.
  - Par exemple retransmission d'appel sur délai de garde pour un mode d'exécution séquentiel des appels distants
- **Il n'y a pas de perte de cohérence de l'état du serveur** (pas d'exécutions partielles interrompues avec des **variables globales rémanentes laissées incohérentes)**.

Fonctionnement de la reprise

- **Pas de sauvegarde de l'état client:** on peut s'en passer le client réalise seulement l'écriture des paramètres résultats.
- **Pas de sauvegarde de l'état du serveur:** Au moment d'une tentative n l'état du serveur est donc celui qui résulte de la dernière tentative n-1.

Reprise arrière du serveur seul: condition l'idempotence

Condition à respecter

- Les états après sont conservés si des exécutions répétées du code du serveur sur les données résultant de l'exécution précédente produisent des résultats identiques: F est **idempotente F (F (x))=F (x)**.

Exemples relatifs à l'idempotence

- F est idempotente si l'exécution de la procédure ne modifie pas l'état du serveur (Ex :lecture du kième article d'un fichier)
- F est idempotente si l'état du serveur est modifié seulement à la première exécution de F (Ex l'écriture du kième article).
- L'écriture en fin de fichier est une opération non idempotente.
- L'incrémentation d'une variable est non idempotente.

Résumé des techniques de reprise arrière

- **Les trois attitudes concernant les reprises en RPC**

- A) **Reprise arrière complète** : mise œuvre de techniques de type gestion transactionnelle.
- B) **Reprise arrière du serveur seul** : uniquement valable si le serveur est une fonction idempotente sur son état.
- C) **Tentatives de reprise hors des deux cas précédents: interdiction totale.**

Sémantique du RPC du point de vue des pannes serveurs

Sémantique exactement une fois

- **Définition théorique:** Une seule exécution est effectuée et celle-ci réussit toujours.
- **Impossible au sens strict:** des lors qu'une hypothèse de panne est formulée.
- **Comportement souhaité:** le plus voisin possible de celui de l'appel de procédure en mode centralisé.
  - Une suite de n tentatives est effectuée
  - Avec sauvegarde état du client et du serveur avant chaque tentative.
  - Pour une procédure déterministe

Sémantique : Au plus une fois

- **Cas d'une fonction quelconque (non idempotente)**: on ne doit pas l'exécuter deux fois.
- **Solution très répandue**: garantir qu'on n 'exécute pas deux fois une même procédure.
  - Identification des requêtes + exécution effectuée une seule fois.
  - . **Si tout va bien** le résultat est retourné à l'utilisateur.
  - La requête ne sera plus exécutée (modulo le délai de garde de l'historique).
  - . **Si un problème est détecté** Il est signalé à l'utilisateur pour qu 'il puisse éventuellement statuer).
- Aucune tentative de reprise n'est effectuée automatiquement. Elle est laissée entièrement à la charge du client.

Sémantique : Au moins une fois

- On se place dans le cas ou chaque exécution du serveur est réalisée sans sauvegarde de l'état du serveur.
- L'exécution est lancée plusieurs fois si nécessaire (dans une certaine limite) pour obtenir une réponse correcte .
- => La procédure doit être **idempotente**.

- **Variante: La dernière de toutes**
- On fait de plus l'hypothèse que les appels sont numérotés et traités séquentiellement.
- On est capable d'attribuer la réussite de l'exécution à la dernière de toutes les tentatives.

Sémantique du RPC du point de vue des pannes client (1)

Problème des serveurs en cas de panne du client

- Abandonner le plus vite possible l'exécution en cours.
- En laissant un état cohérent sur le site serveur.

Extermination

- **Une approche de journalisation**: La souche client note en mémoire stable tous les appels en cours.
- **Lorsqu'un site client est relancé**: il doit examiner l'historique des appels en cours non terminés et demande la destruction de tous les serveurs orphelins encore actifs.

Sémantique du RPC du point de vue des pannes client (2)

Réincarnation

- **Une technique de numéros de séquence:**en fait **un numéro d'époque** correspondant aux périodes d'activité successives du client.
- Il doit être enregistré en mémoire stable et doit marquer toutes les requêtes.
- Lorsqu'un client est relancé il change d'époque et diffuse sa nouvelle époque à ses serveurs qui détruisent les appels anciens.

Sémantique du RPC du point de vue des pannes client (3)

Expiration

- **Une technique de délais de garde :** L'exécution d'un serveur est toujours associée à **une surveillance périodique** du client : le serveur arme des délais de garde.
- **Si le quantum s'achève,** le serveur **demande a son client s'il est encore opérationnel**:
  - **si le client répond** : armement d'un nouveau délai et poursuite.
  - **si le client est en panne** : abandon cohérent d'exécution.
- Remarque : Cette technique permet à la fois la surveillance du client par le serveur et celle du serveur par le client.

Conclusion

Les avantages de l'appel de procédure distante

- **De plus haut niveau** que les communications en mode message.
- **Une structure de contrôle bien connue, l'appel de procédure** (mode de communication support naturel de l'approche client-serveur).
- **Qui s'intègre à l'univers réparti des concepts modernes de génie logiciel: approche objets, approches composants**
  - Modularité, encapsulation, réutilisation par délégation.

Les impressions trompeuses de l'appel de procédure distante

- Une application en appel distant est une application répartie qui risque de présenter à un moment ou à un autre pratiquement toutes les difficultés systèmes/réseaux:
  - **de conception.**
  - **de désignation et de liaison.**
  - **de présentation** des données échangées.
  - **de synchronisation.**
  - **de contrôle de concurrence**.
  - **de tolérance aux pannes et de sécurité.**
  - **de performances.**
  - **de disponibilité d'outils conviviaux.**

L’appel de procédure distante

- **Pour:** Le mode appel de procédure distante est en développement important pour la conception et la réalisation de tous les types d’applications réparties.
- **Restriction:** Le développement de protocoles RPC intégrés aux approches langages, objets ou composants devrait encore mobiliser longtemps les énergies des chercheurs et des développeurs.

Bibliographie

- A.D. Birell and B.J. Nelson, Implementing remote procedure calls ACM Trans on Comp Syst, vol. 2(1), pp 39-59, feb 84
- M. D. Schroeder and M. Burrows Performance of Firelly RPC ACM Trans. On Comp. Syst., vol. 8(1), pp 1-17, jan 90
- B. Liskov and L. Shira Promises: linguistic Support for efficient Asynchronous Procedure Calls in Distributed Systems Proc of SIGPLAN, pp 260-267, 88
- B.N. Bershad, T.E. Anderson, E.D. Lazowska and H.M. Levy Lightweight remote procedure call ACM Trans. On Comp. Syst., vol. 8(1) pp 37-55, jan 90
- Satyanarayanan, H. Siegel Parallel communication in a large distributed environment ACM Trans. On Comp, vol 39(3), pp 328-348, mar 90
- A.S. Tannenbaum "Distributed Operating Systems" Prentice Hall
- W Rosenberry, D Kenney, G Fisher, Comprendre DCE, Addison Wesley