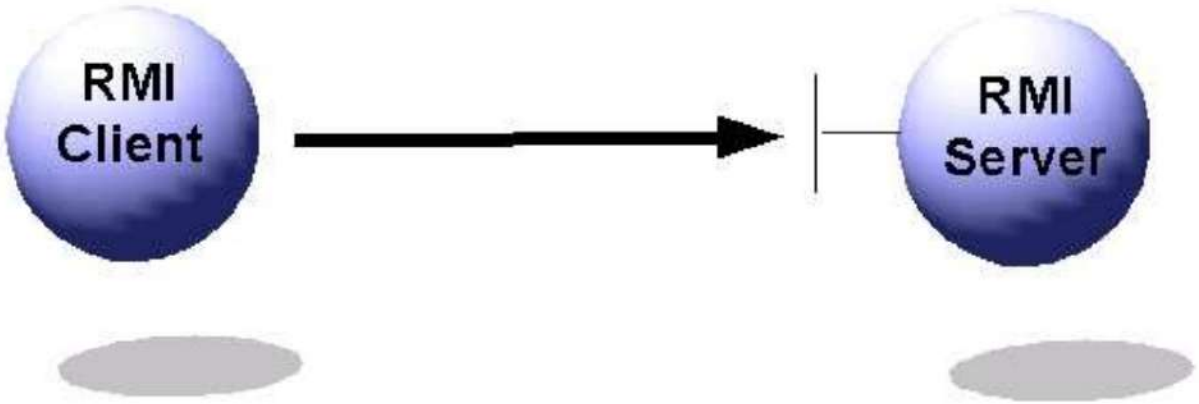
# Systèmes et Applications Distribués

## RPC (Remote Procedure Call)



# Les sockets : primitives de communication

- **On a vu :**
  - Les différentes formes de communication
    - Synchrone
    - Asynchrone
    - Par RDV
  - Et en relation avec elles
    - Intégration des actions de communication dans la sémantique opérationnelle des applications
    - Possibilité de blocages (mort, vivant)
- **Illustration des Sockets sous Java**
  - Ens. de classes java qui permettent de générer des connexions entre les applications et d'échanger un flux de données
  - Une classe pour le serveur : ***`ServerSocket`***
  - Une classe pour définir un canal asynchrone : **`Socket`**
  - => forme de communication par RDV (notion d'entrée)

# Quelques constatations

- **On a considéré implicitement un mode d'interaction avec le serveur**

- **Mode d'interaction : on distingue deux choses**
  - Le protocole (échanges entre clients et serveurs, ex diag. séq. UML)
  - Le contenu (sémantique des données échangées)
  - Exemple d'une messagerie instantanée simple
    - Protocole
      1) Un client C se connecte au serveur S
      2) C → S
      3) C → S et S → C (en parallèle et sans fin)
    - Contenu
      en 2) le contenu du message correspond au pseudonyme de C
      en 3) le contenu du message correspond à un message réél

# Quelques constatations

- **Les données sont échangées sous forme de flux, c'est au récepteur de savoir le type et le sens (interprétation)**

- **=> On ne distingue plus l'utilisation des Socket dans un code C ou Java (même code/utilisation)**

  - On a complètement perdu l'aspect orienté objet aussi ou même toute autre forme de structuration telles que les procedures et les fonctions.

# Le “Web”

- **Le “Web” est un vaste parc de serveurs et de clients on y distribue des ressources sous plusieurs formats.**
  - Image, son, pages html etc.
- **Chaque ressource disponible possède sa propre identification**
  - URI, URL ou URN
- **URL : adresse d'une ressource particulière**
  **On y trouve**
  - l'adresse du serveur qui détient la ressource
  - l'adresse relative de la ressource sur le serveur
  - **le protocole** que le serveur utilise pour fournir cette ressource
- **Protocoles**
  - certain que vous connaissez déjà : SMTP, FTP, HTTP, HTTPS, ...
  - d'autres ...

# Exemple de HTTP

- **Protocole normalisé de communication entre client et serveur**
- **Trois étapes (protocole) :**
  - Établissement d'une connexion TCP avec le serveur sur le port 80 ou autre
  - Envoi d'un message au serveur pour lui réclamer un service (requête)
  - Réception du résultat de la requête (réponse)
- **Requêtes HTTP, deux formes (contenu) :**
  - Réclamer une ressource
    **`Get`** `adresseRelative version_http`
  - Déposer un certain nombre de données pour faire un traitement **`Post`** `formatAttributValeur ressourceDeTraitement`
- **La réponse est un objet multimédia**
  - HTTP, comme FTP et d'autres, se basent sur un typage MIME

# MIME ?

- Protocole de transfert de données
- Système unique d'échange de données typées
  → permet par exemple à un client HTTP (ou Mail) de distinguer une suite de bits correspondant à une image vs. à un fichier pdf
- Le typage MIME est utilisé par le protocole HTTP pour l'encodage des ressources.
- Le protocole HTTP spécifie un attribut pour le type MIME du contenu de la requête ou de la réponse.

# Analyse du cas du "Web"

- **On a des clients (navigateurs) et des serveurs.**
- **Les serveurs peuvent rendre des services plus ou moins complexes.**
- **Quel sont les éléments essentiels pour connecter deux applications ?**
- **Cas de HTTP, mise en place :**
  - d'un protocole adapté à la nature de l'échange (message requête-réponse)
  - d'une convention de format de message et significations
  - d'une convention d'encodage de données échangées (typage MIME)
  - d'un mécanisme d'identification d'application distante (URI, ...)

# Pour une application donnée

- **Forme de service offert ?**
  - Cas du “Web” : des ressources
  - Cas des langages procéduraux : une procédure
  - Cas de l'orienté objet : les méthodes d'un objet.
- **Nature de l'échange pour réaliser le service ?**
  - Protocole : séquence plus contenu
- **Une méthode de transfert des artéfacts locaux**
- **Un mécanisme de localisation des services offerts**
- **Un mécanisme d'amorçage qui permet une transparence**

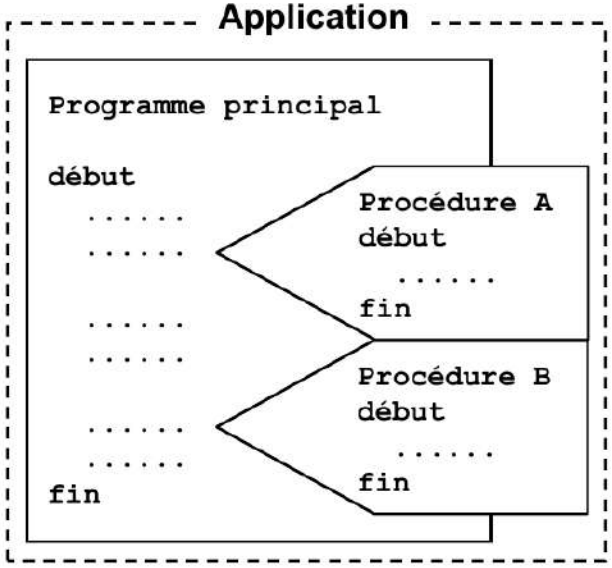
# Objectif de ce cours

***Etudier les mécanismes de mise en place des services et operations (middleware, intergiciel) permettant la connexion entre applications***

- **Cas des langages procéduraux**
  - RPC : Remote Procedure Call
- **Cas des langages orientés objet (Java)**
  - RMI: Remote Method Invocation

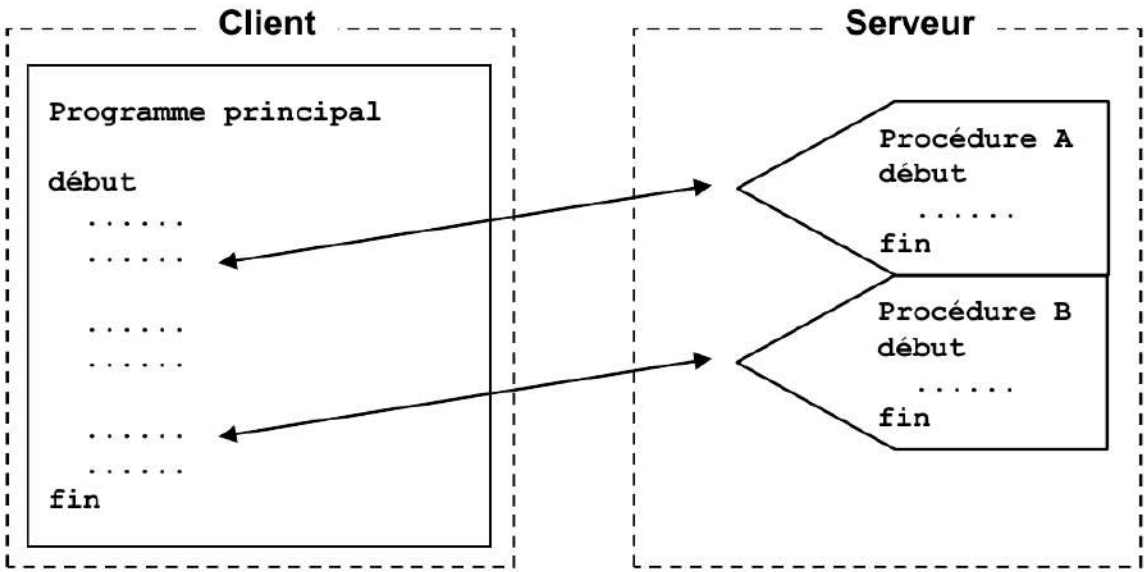
# Programmes et appel de procédures

- **Applications structurées en 2 parties**
  - le programme principal
  - un ensemble de procédures
- **Principe de fonctionnement**
  - programme principal et procédures sont compilés et liés
  - au moment de l'éxécution
    - le programme principal appelle les procédures en transmettant des paramètres d'entrée
    - les procédures s'exécutent et retournent leurs résultats dans les paramètres de sortie



# Analogie avec le client-serveur

- le programme principal se comporte comme un client
- l'ensemble des procédures est assimilable à un ensemble de services disponibles sur un serveur
  - interface d'un service = signature de la procédure
  - interface du serveur = ensemble des signatures des procédures



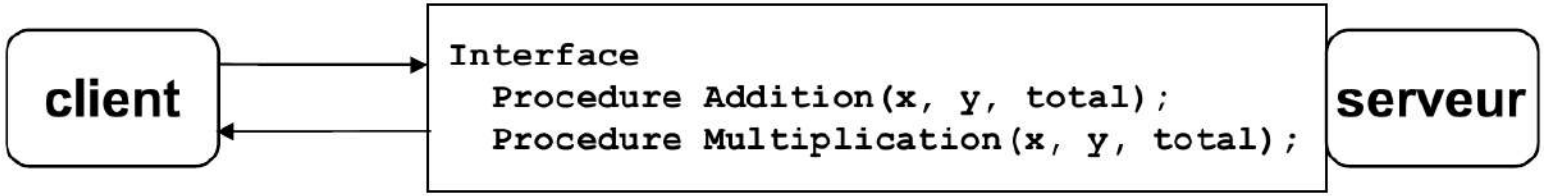
# Exemple : calcul de fonction mathématiques

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

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

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

client | serveur



# Le client RPC

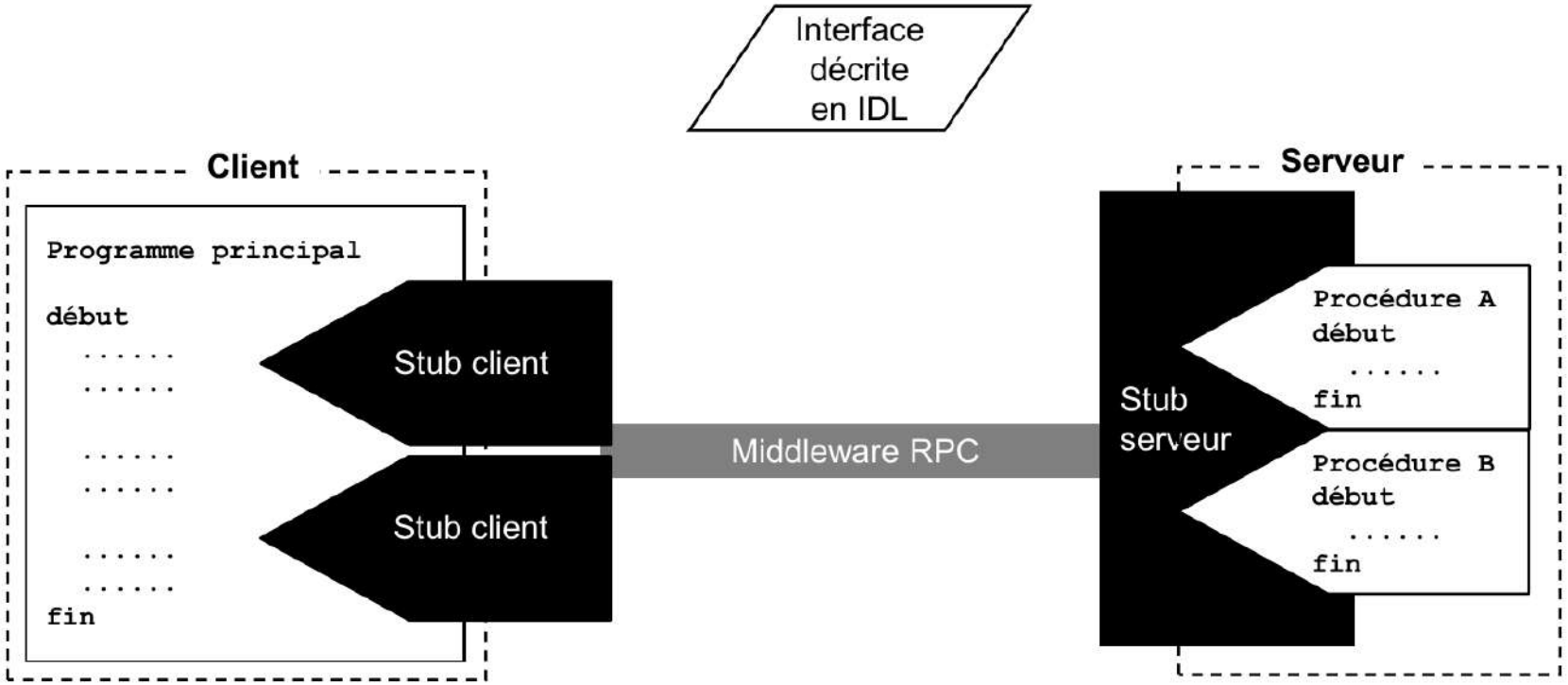
- **Lors de la compilation et de l'édition des liens...**
  - erreur !
    car les procédures ne sont pas présentes
  - procédures émulées par 2 fausses procédures
    appelées *stubs client*
- **Le *stub client***
  - procédure qui porte le nom de la vraie procédure qu'il remplace
  - donne l'illusion au programme principal que la procédure est locale
  - remplace le code de la vraie procédure par un autre code
    - gère la connexion avec le bus middleware
    - transmet les paramètres vers la machine où se trouve la procédure
    - récupère le(s) résultat(s)

# Le serveur RPC

- **Impossible d'avoir un programme exécutable composé uniquement de procédures**
  - nécessité d'un programme principal appelé *stub serveur*
- **Le *stub serveur***
  - permet de créer un exécutable contenant les procédures du serveur
  - gère la communication avec les *stubs client*
    - active la procédure désignée en lui transmettant les paramètres d'appel
    - retourne les valeurs de résultat au stub client

# Middleware par appel de procédures distantes

- **Assure la transparence de localisation**
  - le client appelle les procédures comme si elles étaient locales
  - le middleware assure la communication avec le serveur
  - l'interface du serveur est décrite en IDL
  - le code de préparation d'une requête (stub client) est généré automatiquement à partir de la description IDL



# Caractéristiques

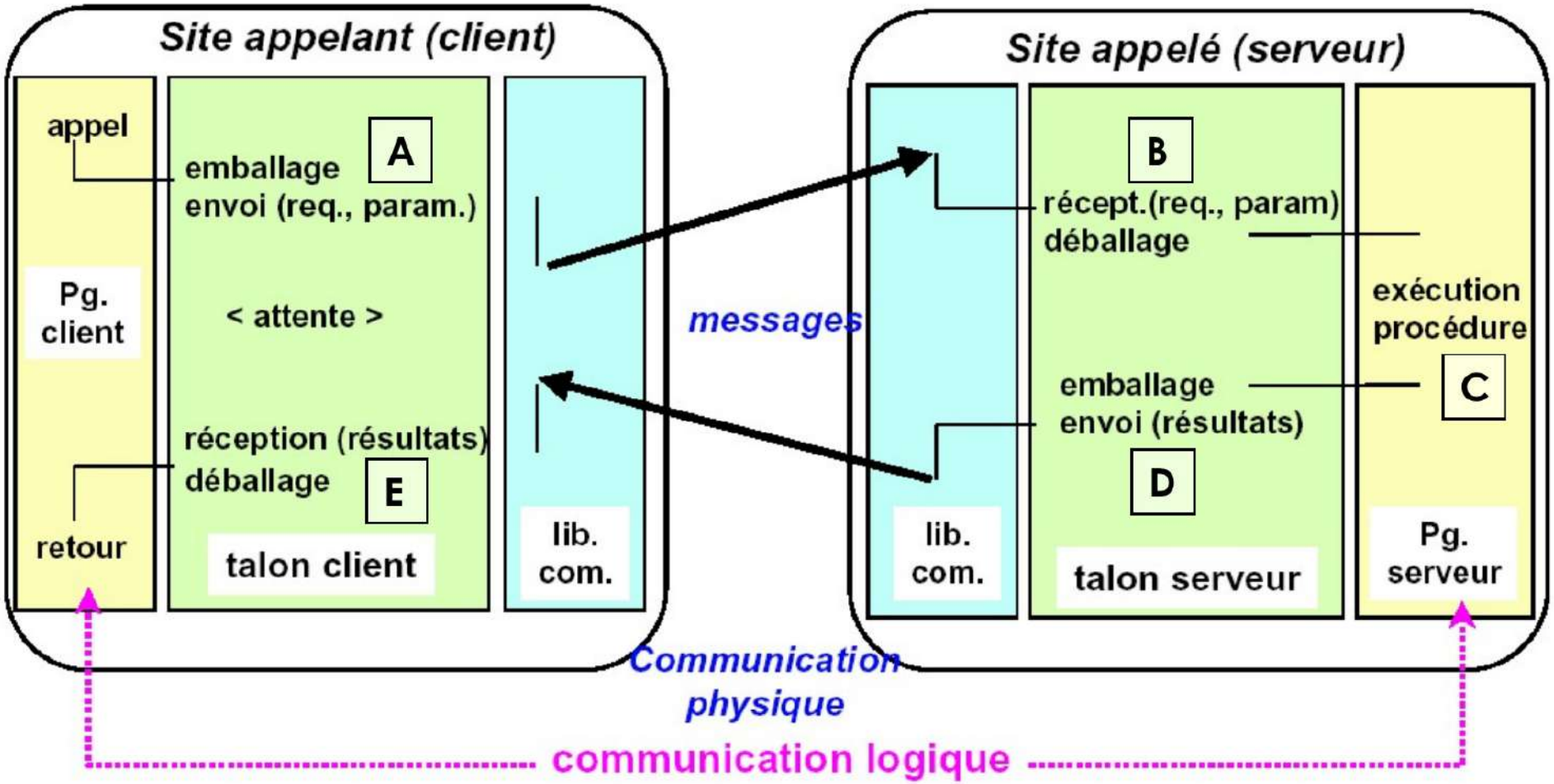
- codes client et serveur indépendants du système de communication
  le client ne sait pas si la procédure est locale ou distante

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

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

- structure de communication construite au moment de la compilation
- communication synchrone → le client attend la réponse à son appel de procédure avant de continuer son traitement
- technologie RPC entièrement standardisée
  (inclue IDL + services nécessaires à la communication)

# RPC [Birrel et Nelson 84] : principe de réalisation



# RPC (A) : principe de fonctionnement

- **Côté de l'appelant**
  - le client réalise un appel procédural vers la procédure talon client
    - transmission de l'ensemble des arguments
  - au point A
    - le talon collecte les arguments et les assemble dans un message (empaquetage - parameter marshalling)
    - un identificateur est généré pour le RPC
    - un délai de garde est armé
    - le talon transmet les données au protocole de transport pour émission sur le réseau. Mais où??
    - pb : détermination de l'adresse du serveur

# RPC (B, C et D) : principe de fonctionnement

- **Côté de l'appelé**
  - le protocole de transport délivre le message au service de RPC (talon serveur/skeleton)
  - au point B
    - le talon désassemble les arguments (dépaquetage - unmarshalling)
    - l'identificateur de RPC est enregistré
    - l'appel est ensuite transmis à la procédure distante requise pour être exécuté (point C). Le retour de la procédure redonne la main au service de RPC et lui transmet les paramètres résultats (point D)
  - au point D
    - les arguments de retour sont empaquetés dans un message
    - un autre délai de garde est armé
    - le talon transmet les données au protocole de transport pour émission sur le réseau

# RPC (E) : principe de fonctionnement

- **Coté de l'appelant**
  - l'appel est transmis au service de RPC (point E)
    - les arguments de retour sont dépaquetés
    - le délai de garde armé au point A est désarmé
    - un message d'acquittement avec l'identificateur du RPC est envoyé au talon serveur (le délai de garde armé au point D peut être désarmé)
    - les résultats sont transmis à l'appelant lors du retour de procédure

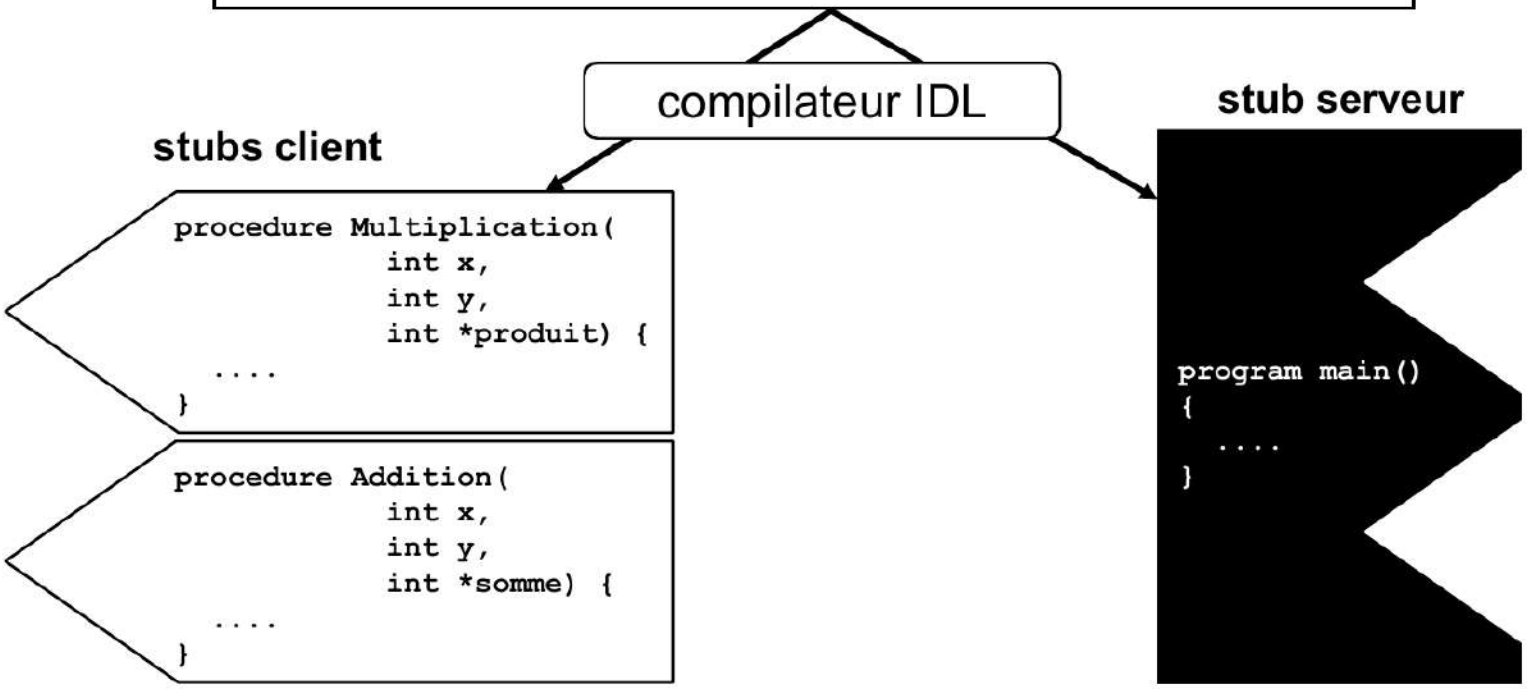
# Notion de contrat entre client et serveur

- **Le contrat**
  - formalise le dialogue entre le client et le serveur
    - permet au client et au serveur d'avoir la même compréhension des échanges effectués
  - répond aux questions
    - que transmet-on ?
    - où envoie-t-on les données ?
    - qui reçoit les données ?
    - comment sait-on que le travail est terminé ?

# Exemple de contrat

```
/* OSF IDL RPC code */
[uuid (xxxxxxxx-xxxx-xxxx-xxxx) version (N.n)]                    ❶
interface serveur_mathématique                                    ❷
{
      Addition([in]  int x,                                       ❸
               [in]  int y,                                       ❹
               [out] int *somme);                                 ❺

      Multiplication([in]  int x,                                 ❸
                     [in]  int y,                                 ❹
                     [out] int *produit);                         ❺
}
```



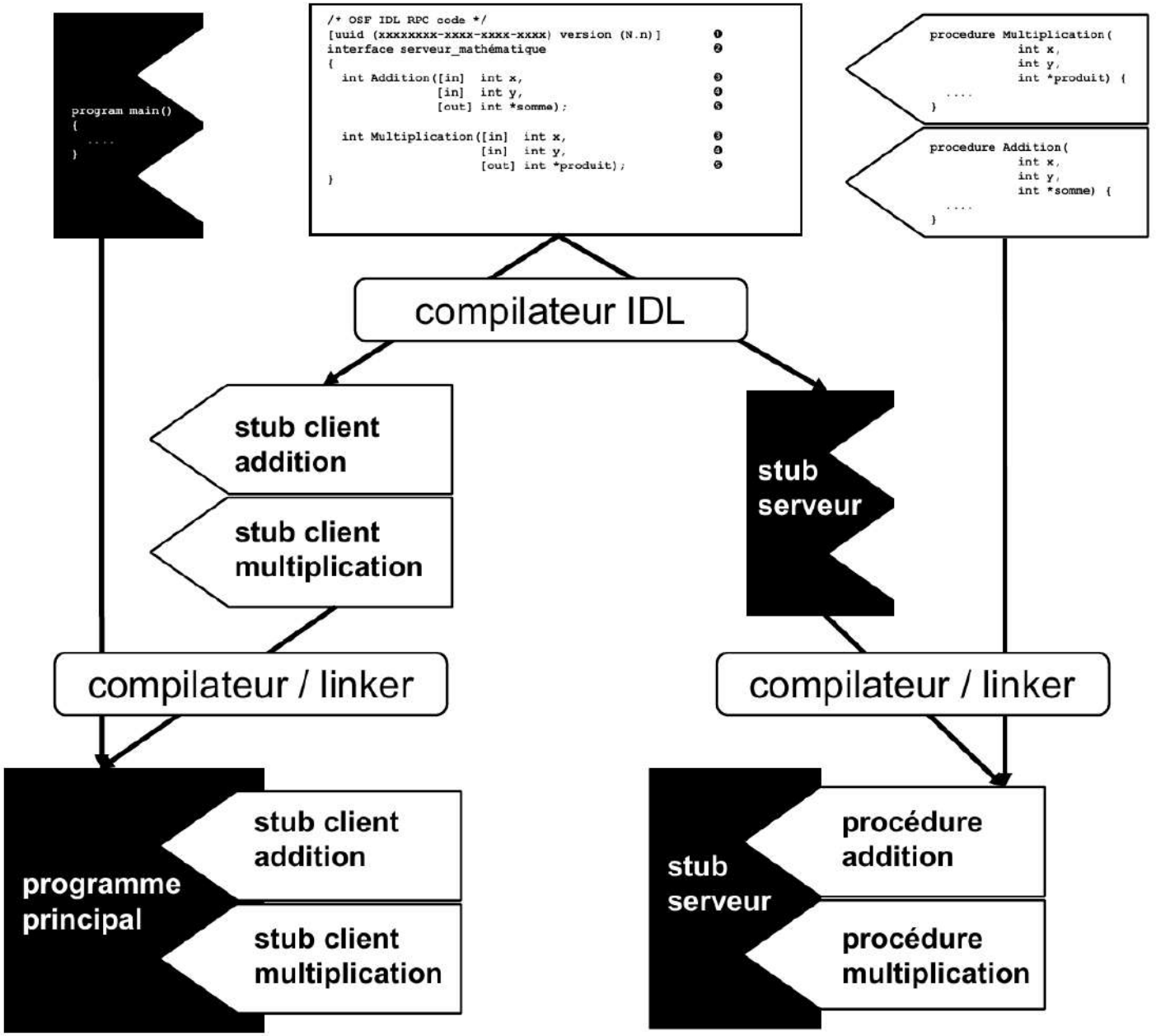
# Construction du client et du serveur

- **Eléments à développer**
  - le programme principal de l'application (le client)
    - n'importe quel langage de programmation
    - le plus souvent en C
  - les procédures composant l'application (le serveur)
  - le contrat décrivant les échanges entre client et serveur
    - écrit en langage IDL
- **Etapes de déploiement**
  - générer les stubs client et serveur
    - compilateur IDL
  - construire l'exécutable client
    - compiler le programme principal et les stubs client
    - lier les deux
  - construire l'exécutable serveur
    - compiler le stub serveur et les procédures
    - lier les deux

# Caractéristiques du contrat

- **Le contrat**
  - ❶ : est repéré par un n° d'identifiant unique (uuid = universal unique identifier)
    - utilisé lors de chaque requête du client
  - ❷ : définit et nomme l'interface offerte par le serveur
    - contient un sous-ensemble des services disponibles sur le serveur
  - ❸ : définit la signature des services
    - nom de la procédure
    - paramètres caractérisés par leur mode d'utilisation
      - paramètres d'entrée [in] (❸ et ❹)
      - paramètres de sortie [out] (❺)
      - paramètres d'entrée et de sortie [in][ou]
  - est rédigé dans un langage spécifique (IDL = Interface Definition Language)
    - plusieurs versions (OSF IDL RPC pour la technologie RPC)
  - **est distribué entre le client et le serveur**
- **Le compilateur IDL**
  - génère les stubs client et serveur (le plus souvent en langage C)

# Construction du client et du serveur



# Représentation des données

- **Problème classique dans les réseaux**
  - Conversion nécessaire si le site client et le site serveur
    - n'utilisent pas le même codage
      - caractères (ASCII, EBCDIC, Unicode)
      - taille des mots mémoire (16, 32, 64 bits)
      - numérotation des octets dans un mot mémoire (big endian, little endian)
    - utilisent des formats internes différents (type caractère, entier, flottant, ...)
    - solution placée classiquement dans la couche 6 du modèle OSI présentation
  - dans réseau : passage de paramètres par valeur
    - émulation des autres modes

# RPC : les difficultés

- **Appel de procédure local**
  - appelant et appelé sont dans le même espace virtuel
  - même mode de pannes
  - appel et retour de procédure sont des mécanismes internes considérés comme fiables
    - sauf aspects liés à la liaison dynamique de la procédure et à la vérification de la protection
  - dans certains langages
    - mécanisme d'exception pour transmettre les erreurs de l'appelé à l'appelant
- **Appel de procédure à distance**
  - Appelant et appelé sont dans 2 espaces virtuels différents
  - pannes du client et du serveur sont indépendantes
  - possibles pannes du réseau de communication (perte du message d'appel ou de réponse)
  - temps de réponse du serveur long
    - charge du réseau ou du site serveur

# Passage des paramètres

- **Valeur**
  - pas de problème particulier
- **Copie / Restauration**
  - les valeurs des paramètres sont recopiées
  - pas de difficultés majeures
    mais redéfinition des solutions définies pour les réseaux
  - optimisation des solutions pour le RPC
  - bonne adaptation au langage C (faiblement typé)

# Passage des paramètres

- **Reference**
  - utilise une adresse mémoire centrale du site de l'appelant
    …
    … aucun sens pour l'appelé
  - solutions
    - interdiction totale
      - introduit une différence entre procédures locales et procédures distantes
    - simulation en utilisant une copie de restauration
      - marche dans de très nombreux cas
      - mais violation dans certains cas de la sémantique du passage par référence
      - exemple de pb :

```
procedure double_incr (int *x, int *y) {
     x := x+2;
     y := y+1;
}
a := 0 ;
double_incr (a, a)
```

**résultat : a = ? 1 ? 2 ? 3 ?**

- reconstruire l'état de la mémoire du client sur le site serveur
  - solution très coûteuse
- utilisation d'une mémoire virtuelle répartie
  - nécessite un système réparti avec mémoire virtuelle

# Passage des paramètres

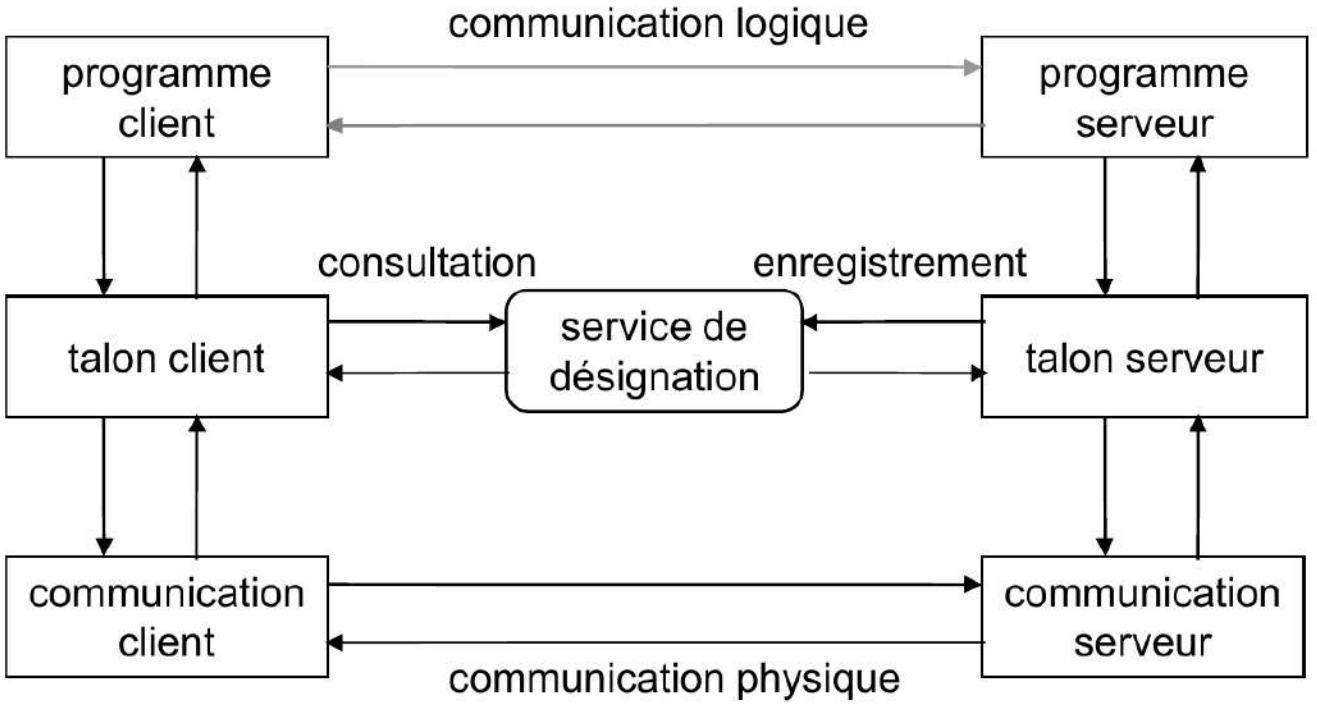
- **Solutions généralement prises**
  - IN : passage par valeur (aller)
  - OUT : passage par valeur (retour)
  - IN-OUT : interdit ou passage par copie-restauration
    - ATTENTION : n 'est pas équivalent au passage par référence

# Désignation

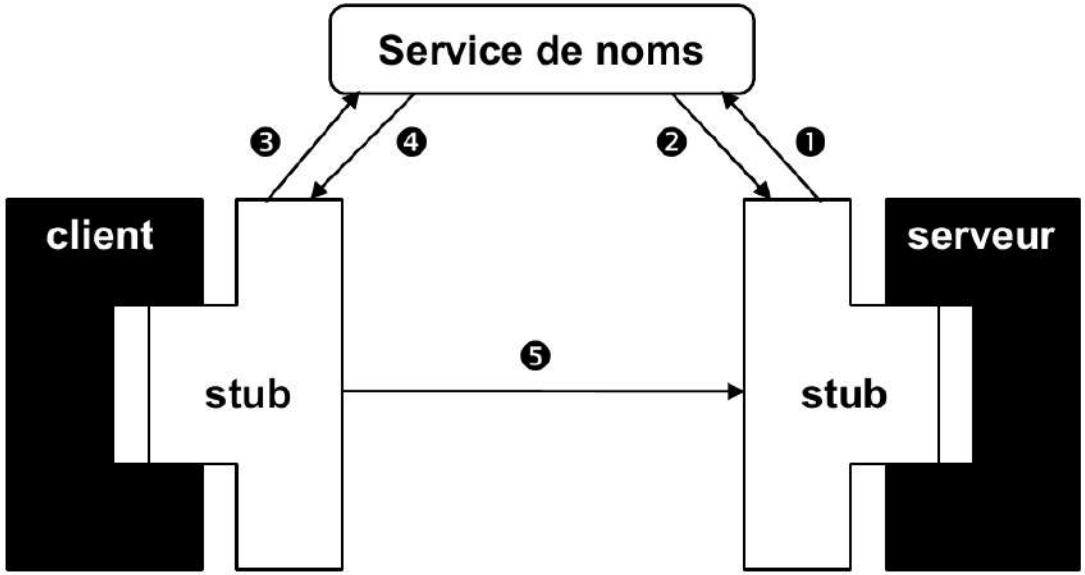
- **Objets à désigner**
  - le site d'exécution + le serveur + la procédure
  - désignation globale indépendante de la localisation
    - possibilité de reconfiguration des services (pannes, régulation de charge, ...)
- **Désignation**
  - statique ou dynamique
    - statique : localisation du serveur connue à la compilation
    - dynamique : non connue à la compilation
      objectifs :
      - séparer la connaissance du nom du service de la sélection de la procédure qui va l'exécuter
      - permettre l'implémentation retardée

# Liaison et fonctionnement

- **Liaison (détermination de l'adresse du serveur)**
  - liaison statique
    (pas d'appel à un serveur de nom ou appel lors de la compilation)
  - liaison au premier appel
    (appel du serveur de nom lors du premier appel)
  - liaison à chaque appel
    (appel du serveur de nom à chaque appel)



# Liaison - solution classique : DNS Internet




**❶ le serveur s'enregistre auprès du serveur de noms**

– adresse, interfaces

**❷ le service de noms confirme au serveur qu'il est connu**

– le serveur se met en attente de connexions

**❸ le client demande au service de noms l'adresse du serveur**

– fournit le nom et le n° d'interface

**❹ le service de noms retourne l'adresse du serveur**

**❺ le client peut établir une connexion avec le serveur**

# RPC : les limites

- **Mécanisme de bas niveau**
  - la notion de procédure n'existe pas dans les méthodes d'analyse
- **N'assure pas tous les services souhaités d'un bus de communication**
  - fiabilité du transfert
    - appels perdus si serveur ou réseau en panne
    - gestion des erreurs et des pannes à la charge du client
  - concept de transaction
  - diffusion de messages
    - communication 1-1
    - pas de communication 1-n
- **Outils de développement**
  - limités à la génération automatique des talons
  - peu d'outils pour le déploiement et la mise au point d'applications réparties

# RPC : les problèmes

- **Traitement des défaillances**
  - congestion du réseau ou du serveur
    - la réponse ne parvient pas avant une date fixée par le client (système temps critique)
  - panne du client pendant le traitement de la requête
  - panne du serveur avant ou pendant le traitement de la requête
  - erreur de communication
- **Problèmes de sécurité**
  - authentification du client
  - authentification du serveur
  - confidentialité des échanges
- **Désignation et liaison**
- **Aspects pratiques**
  - adaptation à des conditions multiples (protocoles, langages, matériels)
  - gestion de l'hétérogénéité