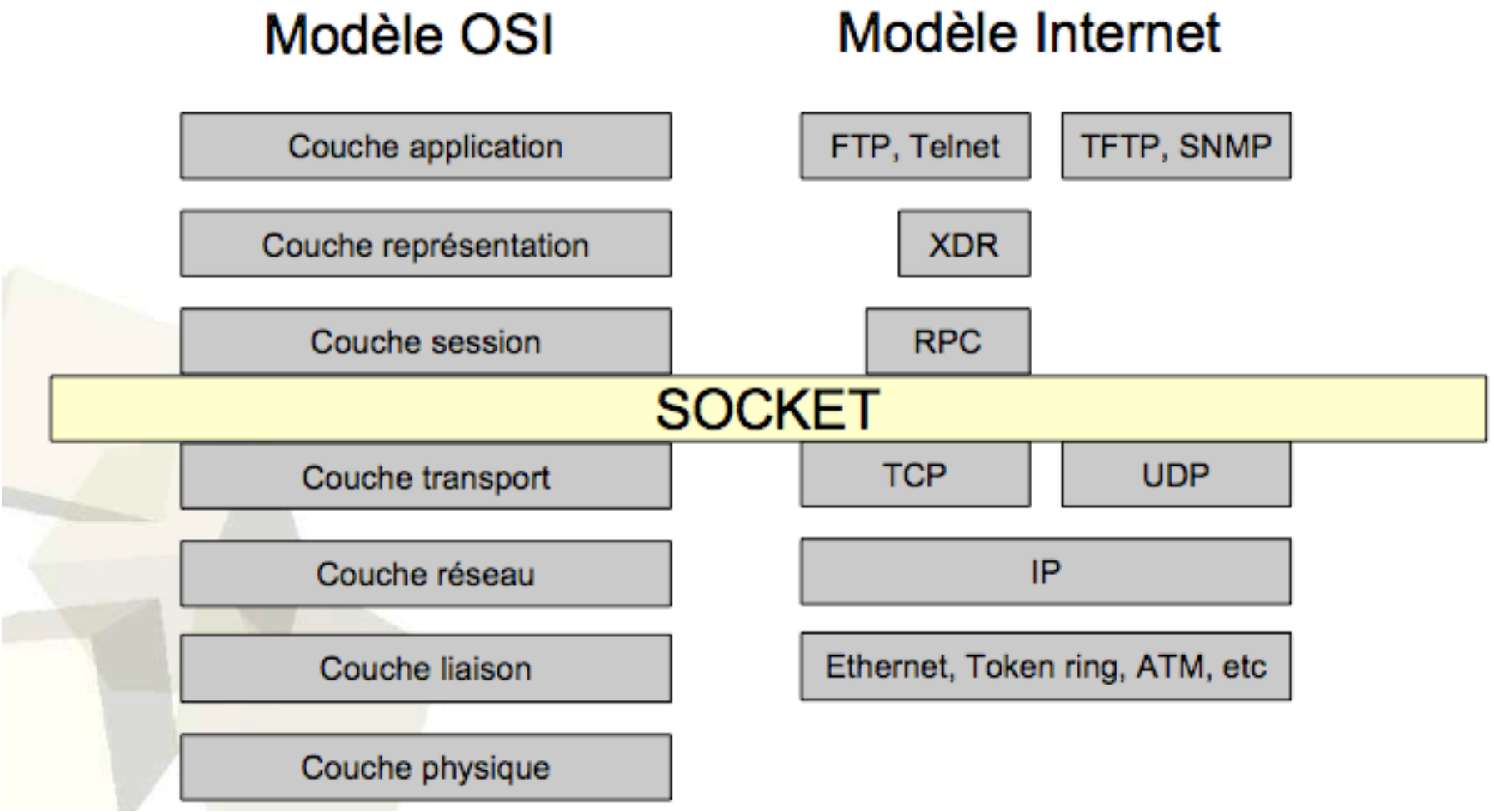
OS Réseaux et Programmation Système - C5

Rabie Ben Atitallah

rabie.benatitallah@univ-valenciennes.fr

RPC - XDR

Rappel

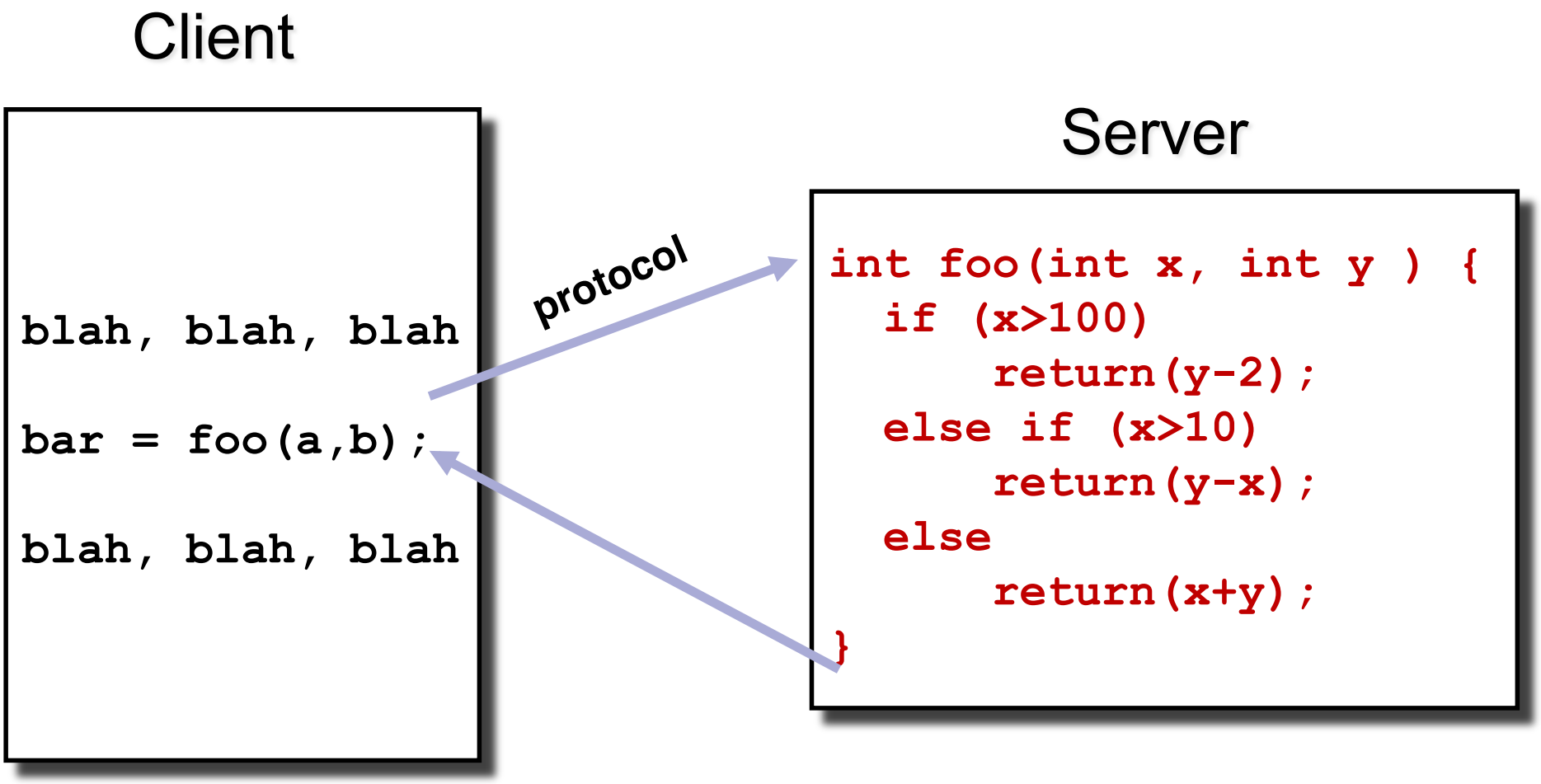

Modèle OSI
Modèle Internet
Couche application
Couche représentation
Couche session
Couche transport
Couche réseau
Couche liaison
Couche physique
FTP, Telnet
TFTP, SNMP
XDR
RPC
SOCKET
TCP
UDP
IP
Ethernet, Token ring, ATM, etc

RPC: Remote Procedure Call

- Besoin d'un environnement de haut niveau pour le développement d'applications réparties qui :
  - reprend le concept du client/serveur
  - permet d'identifier un très grand nombre de services (> #n de port)
  - conserve les paradigmes habituels d'exécution :
    - l'appel de fonction, passage de paramètres
    - la notion de programme (ensemble de fonctions)

  ➔ **RPC**
- Système de distribution de services proposé par SUN au début des années 80 (utilisé pour le développement de NFS)

Procédures Distantes



La sémantique de l'appel

- Si on tient compte des erreurs (pertes, duplications) pouvant survenir lors des communications, on définit 3 sémantiques possibles pour l'appel de procédures:
  - exactement une fois
  - au moins une fois
  - au plus une fois.
- La sémantique choisie par l'implémentation sous RPC-Sun est *au moins une fois*
- *Il faut s'assurer que l'exécution d'une procédure distante soit idempotent, par exemple en utilisant le numéro de transaction (xid) disponible dans chaque message RPC.*

Les paramètres

- Un seul paramètre est échangé lors de l'appel (RPC call)
  - si l'application requiert l'échange de plusieurs paramètres ils doivent être regroupés au sein d'une seule structure de données.
- Un seul élément peut être échangé lors du retour (RPC reply)
  - à travers la valeur de retour de la fonction

Identification des procédures

- Une procédure distante est identifiée de manière unique par un triplet :
  - #program, #prog_version, #procedure
- Un programme regroupe un ensemble de procédures et possède une version
  - plusieurs versions peuvent être disponibles simultanément
- Certains numéros de programmes sont réservés à certains services :
  - Réservé: 0x00000000 à 0x1FFFFFFFF
  - Public : 0x20000000 à 0x3FFFFFFFF
  - Semi-public : 0x40000000 à 0x5FFFFFFFF
  - Réservé : 0x60000000 à 0xFFFFFFFFF

Fichier */etc/rpc*

- *L'association entre service RPC et numéro de programme est décrit par le fichier/etc/rpc*

```
portmapper      100000  portmap sunrpc
rstatd          100001  rstat rstat_svc rup perfmeter
rusersd         100002  rusers
nfs             100003  nfsprog
ypserv          100004  ypprog
mountd          100005  mount showmount
ypbind          100007
walld           100008  rwall shutdown
yppasswdd       100009  yppasswd
etherstatd      100010  etherstat
rquotad         100011  rquotaprog quota rquota
sprayd          100012  spray
```

Commande *rpcinfo*

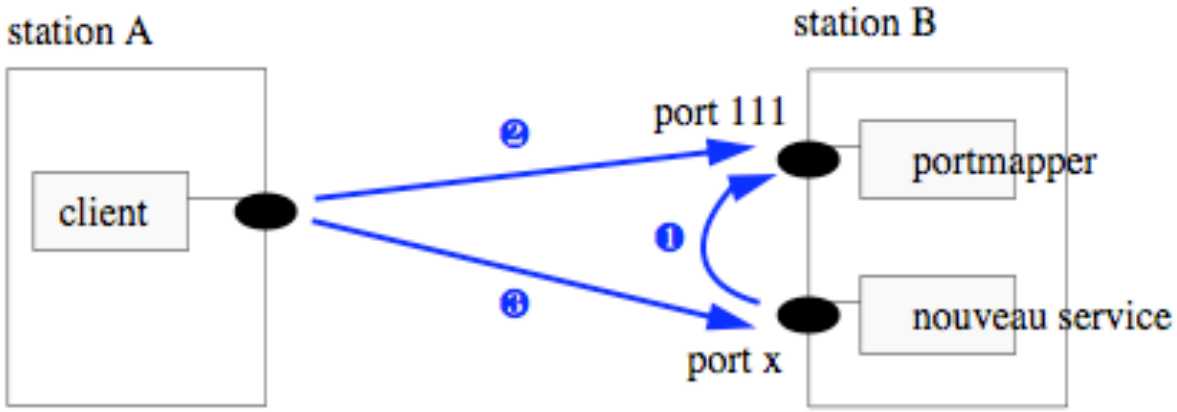
- Liste les programmes, versions et procédures disponibles sur une station :

```
Macintosh-4:rpc rabie$ rpcinfo -p
   program vers proto   port
    100000    2   tcp    111  portmapper
    100000    2   udp    111  portmapper
 536870913    1   udp   1021
 536870913    1   tcp   1023
```

Le portmapper

- Le portmapper permet de rediriger un client vers le numéro de port hébergeant le service
- Le portmapper est sur un numéro de port réservé : 111
  - Les clients n'ont besoin de connaître que ce seul numéro de port/service
- Le portmapper est lui-même un service RPC :
  - description des procédures *portmap* en langage RPC
  - description de la structure des messages *portmap* en langage XDR

Le portmapper

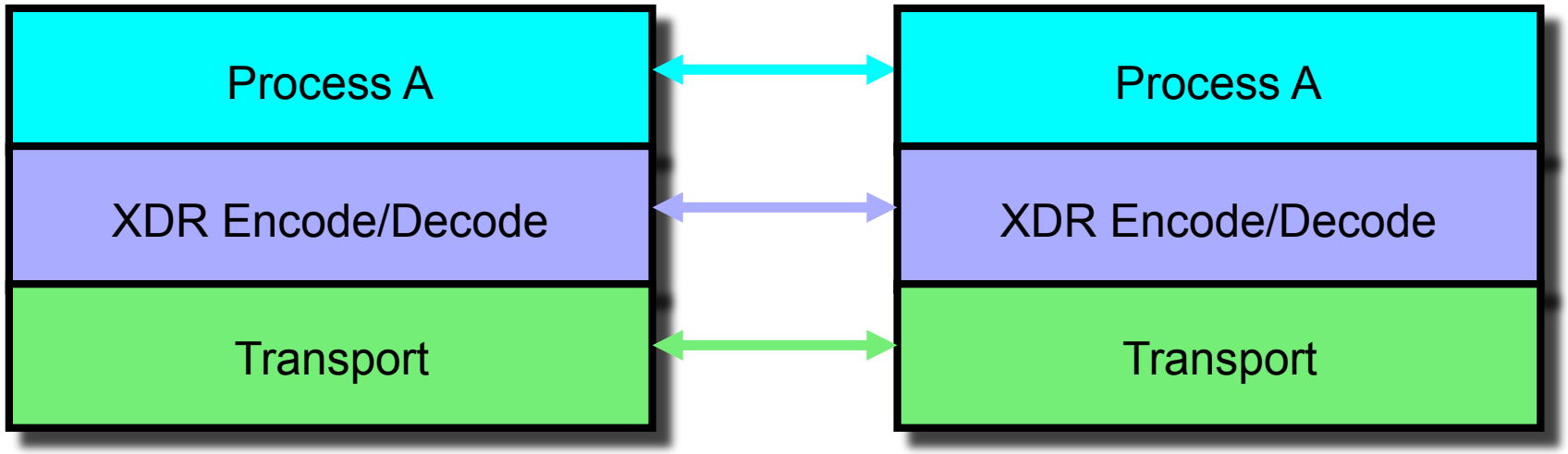


- pmap_set() = 1 ; enregistre un service ❶
- pmap_getport() = 3 : retourne le numéro de port associé au service ❷
- pmap_rmtcall() = 5 : appel d'une procédure distante❸

XDR: eXternal Data presentation

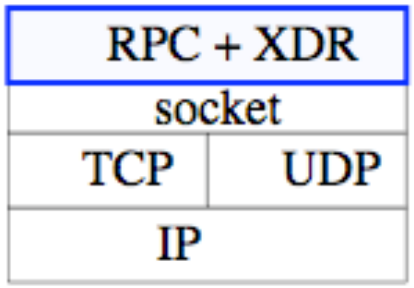
- Masque l'hétérogénéité de représentation des données
  - format standard, fonctions de transcodage
- Utilisation d'un protocole de sérialisation indépendant du client et du serveur
  - Le talon client sérialise en XDR les arguments, envoie la requête au serveur, récupère le résultat et le "désérialise"
  - Le talon serveur "désérialise" les arguments, exécute le code demandé, sérialise en XDR le résultat et l'envoie au client

XDR


Process A
XDR Encode/Decode
Transport
Process A
XDR Encode/Decode
Transport

Environnement

- Un environnement de plus haut niveau que les “sockets” et la transmission de messages.
- Architecture fonctionnelle sur Internet :



- TI-RPC = Transport independent RPC
  - Implémentation des RPC qui permet le développement d’applications indépendamment des éléments logiques et physiques (réseaux, protocoles, etc.) utilisés pour transmettre des données.

Langage XDR/RPC

- ❑ Langage XDR
  - Langage de description de la structure des données échangées lors du RPC
    - Défini par le rfc 1014.
    - Syntaxe issue de celle employée pour la description des données dans le langage C
- Langage RPC
  - Extension du Langage XDR, pour permettre la définition de procédure (programme)

Exemple de description de service

- Décrit en “RPC language”

```
program PING_PROG {
/*Latest and greatest version*/
version PING_VERS_PINGBACK {
        void        PINGPROC_NULL(void) = 0;
        /* Ping the client, return the round-trip time in ms. Returns -1 if the
operation timed out.
       int          PINGPROC_PINGBACK(void) = 1;
} = 2;
/* Original version*/
version PING_VERS_ORIG {
     void           PINGPROC_NULL(void) = 0;
} = 1;
} = 100115;
```

Programmation RPC

- Plusieurs niveaux de programmation :
  - en utilisant une API restreinte (4 fonctions)
  - en utilisant toute l'API (≥20 fonctions), cela permet alors de
    - choisir le protocole de transport
    - faire de l'authentification de client faire des appels de service asynchrones
    - faire des appels de fonction diffusés (broadcastés)
  - **en utilisant le compilateur rpcgen et le langage RPCL**

Exemple de fonctions

- Fonctions du niveau élevé
  - getrpcport() : port associé à la version d'un programme sur une machine donnée
  - rusers() : les utilisateurs connectés sur une machine
  - rnusers() : le nombre d'utilisateurs connectés sur une machine
  - rwall() : envoi un message à tous les utilisateurs d'une machine
- Fonctions de bas niveau pour le serveur
  - svcudp_create() : initialisation de la communication (création d'une socket)
  - svc_register() : enregistrement d'un service, lui associe une fonction de traitement
  - svc_getargs() : décodage des arguments de la procédure
  - svc_getcaller() : origine de la requête
  - svc_freeargs() : libération de l'espace alloué par XDR …

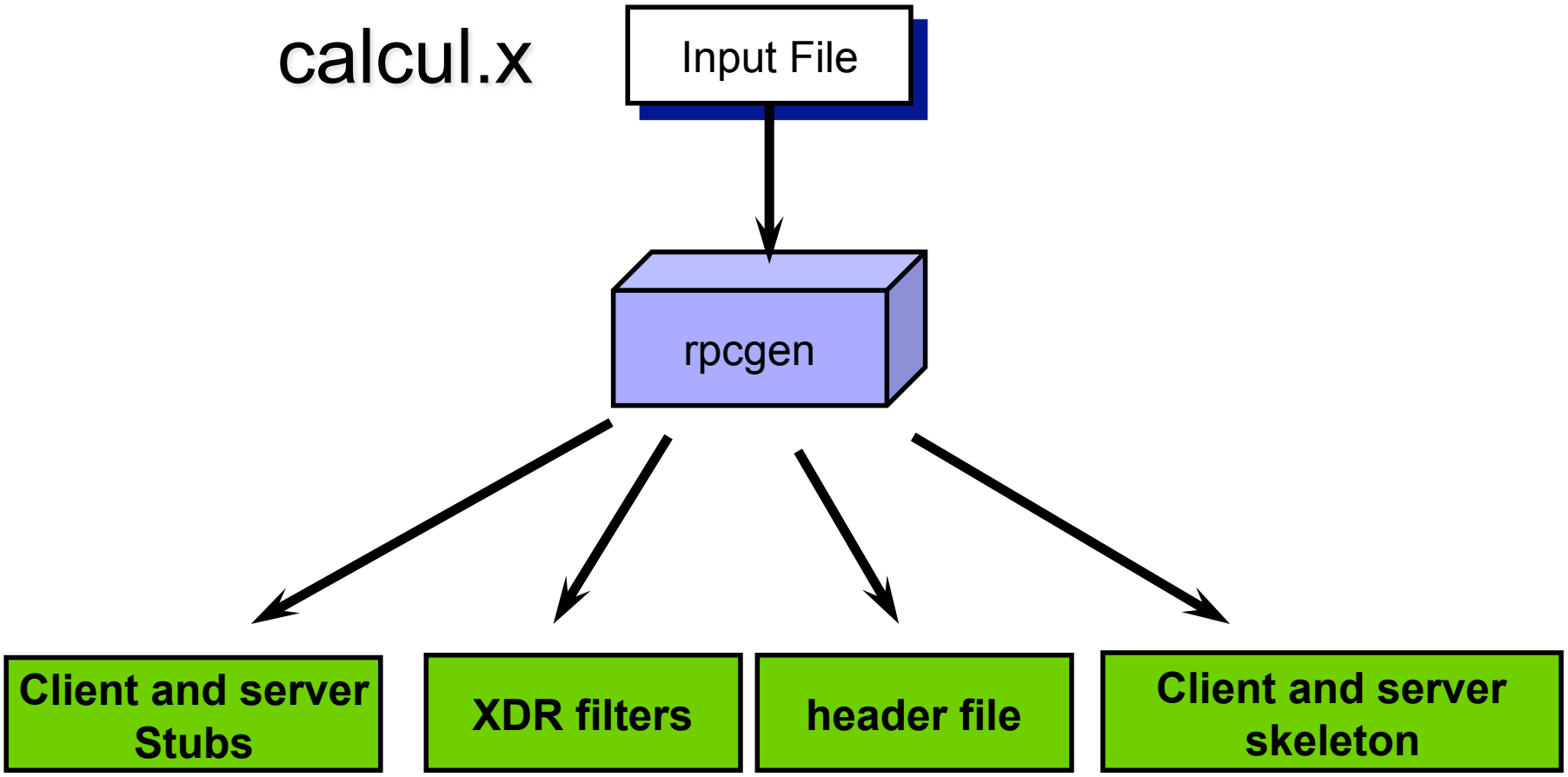
RPCGen

- Utilisation de la description des structures de données (XDR language) et de la définition des services (RPC language) pour générer automatiquement le code des appels aux fonctions de codage :
- rpcgen -a calcul.x
- Le fichier calcul.x contient la description des données et des procédures employées par le service

Exemple

```
struct data {
    unsigned int arg1;
    unsigned int arg2; };
typedef struct data data;
struct reponse {
    unsigned int somme;
    int errno;};
typedef struct reponse reponse;
program CALCUL{
    version VERSION_UN{
        void CALCUL_NULL(void) = 0;
            reponse CALCUL_ADDITION(data) = 1;
    } = 1;
} = 0x20000001;
```

RPCGen


calcul.x
Input File
rpcgen
Client and server Stubs
XDR filters
header file
Client and server skeleton
C Source Code

Exemple

- L'option -a permet de produire un squelette pour notre programme client (calcul_client.c) et un squelette pour la fonction distante (calcul_server.c).
- Avec ou sans cette option, les fichiers suivants sont également produits: calcul.h (entête), calcul_clnt.c (stub client), calcul_svc.c (stub serveur) et calcul_xdr.c (routines XDR).
- Le format XDR (eXternal Data Representation) définit les types utilisés pour l'échange de variables entre le client et le serveur.
  - les types que nous avons nous-mêmes définis nécessitent un filtre XDR, c'est le rôle des fonctions définies dans le fichier calcul_xdr.c
  - gcc -c calcul_xdr.c
- Les stubs client et serveur sont complets et peuvent déjà être compilés
  - gcc -c calcul_clnt.c
  - gcc -c calcul_svc.c

Processus serveur

```
#include "calcul.h”
void * calcul_null_1_svc(void *argp, struct svc_req *rqstp){
    static char* result;
    /* insert server code here   */
     return((void*) &result);
}
 reponse * calcul_addition_1_svc(data *argp, struct svc_req *rqstp){
    static reponse  result;
    /*   insert server code here   */
    return(&result);
}
```

Processus serveur

```
reponse * calcul_addition_1_svc(data *argp, struct svc_req *rqstp){
  static reponse  result;
  unsigned int max;
  result.errno = 0; /* Pas d'erreur */
 /* Prend le max */
 max = argp->arg1 > argp->arg2 ? argp->arg1 : argp->arg2;
/* On additionne */
result.somme = argp->arg1 + argp->arg2;
/* Overflow ? */
  if ( result.somme < max ) {    result.errno = 1;  }
     return(&result);}
```

Processus serveur

- Nous pouvons alors le compiler en faisant:
  - gcc -c calcul_server.c
- Puis pour obtenir le programme serveur complet, il faut lier calcul_svc.o, calcul_server.o et calcul_xdr.o ensemble:
  - gcc -o server calcul_svc.o calcul_server.o calcul_xdr.o
- Macintosh-4:rpc rabie$ sudo ./server &
  Macintosh-4:rpc rabie$ rpcinfo -p
  program vers proto port
  100000 2 tcp 111 portmapper
  100000 2 udp 111 portmapper
  536870913 1 udp 1021
  536870913 1 tcp 1023

Processus client

```
#include <limits.h>
#include "calcul.h”
CLIENT *clnt;
Void test_addition (uint param1, uint param2){
  reponse  *resultat;
  data  parametre;  /* 1. Preparer les arguments */
  parametre.arg1 = param1;
  parametre.arg2 = param2;
   printf("Appel de la fonction CALCUL_ADDITION avec les parametres: %u et
     %u \n", parametre.arg1,parametre.arg2);
  /* 2. Appel de la fonction distante */
    resultat = calcul_addition_1 (&parametre, clnt);
  if (resultat == (reponse *) NULL) {    clnt_perror (clnt, "call failed");
     clnt_destroy (clnt);    exit(EXIT_FAILURE);  }
 else if ( resultat->errno == 0 ) {    printf("Le resultat de l'addition est: %u \n
     \n",resultat->somme);  }
 else {    printf("La fonction distante ne peut faire l'addition a cause d'un
     overflow \n\n");  }}
```

Processus client

```
Int main (int argc, char *argv[])
{
  char *host;
    if (argc < 2) {
    printf ("usage: %s server_host\n", argv[0]);    exit (1);  }
     host = argv[1];
    clnt = clnt_create (host, CALCUL, VERSION_UN, "udp");
  if (clnt == NULL) {    clnt_pcreateerror (host);    exit (1);  }
    test_addition ( UINT_MAX - 15, 10 );
  test_addition ( UINT_MAX, 10 );
    clnt_destroy (clnt);  exit(EXIT_SUCCESS);
}
```

Exécution

- gcc -c calcul_client.c
- > gcc -o client calcul_client.o calcul_clnt.o calcul_xdr.o
- > ./client localhost

  Appel de la fonction CALCUL_ADDITION avec les parametres: 4294967280 et 10

  Le resultat de l'addition est: 4294967290

  Appel de la fonction CALCUL_ADDITION avec les parametres: 4294967295 et 10

  La fonction distante ne peut faire l'addition a cause d'un overflow >