# PROGRAMACIÓN DE SISTEMAS DISTRIBUIDOS

Simon Pickin
Alberto Núñez

## gSOAP

# Introducción



- gSOAP nos permite desarrollar Servicios Web en C/C++
- Proporciona
  - Bibliotecas
  - Compiladores
- Para desarrollar el sistema completo es necesario:
  - Las definición del servicio (es decir, la interfaz remota)
  - La implementación de las funciones del servicio en el servidor
  - La implementación del cliente del servicio
  - Un servidor que trata las peticiones del servicio, pasándolas a la implementación de las funciones
    - Servidor independiente (mono-hilo o multi-hilo)
    - Servidor web con CGI

# Definición del servicio



- Vamos de definir como servicio una calculadora
  - Suma
  - Resta
  - Multiplicación
  - División

```
// File calc.h
//gsoap calcns service name: calculador
//gsoap calcns service style: rpc
//gsoap calcns service encoding: encoded
//gsoap calcns service namespace: urn:calcExample
int calcns__add  (int a, int b, int *res);
int calcns__subs (int a, int b, int *res);
int calcns__mult  (int a, int b, int *res);
int calcns__div (int a, int b, int *res);
```

# Definición del servicio



- Las funciones siempre devuelven `int`
  - El resultado del estado de la ejecución
- El último parámetro contiene el resultado
  - El valor que se devuelve al cliente (pasado por referencia)

```
// File calc.h
//gsoap calcns service name: calculador
//gsoap calcns service style: rpc
//gsoap calcns service encoding: encoded
//gsoap calcns service namespace: urn:calcExample
int calcns__add  (int a, int b, int *res);
int calcns__subs (int a, int b, int *res);
int calcns__mult  (int a, int b, int *res);
int calcns__div (int a, int b, int *res);
```

# Definición del servicio



- Generamos los *stubs* a partir de la interfaz remota (fichero: `interfaz.h`)
  - **`soapcpp2 -c interfaz.h`**
    - La opción `-c` genera código en lenguaje C
    - Uso de la opción `-C` generaría solo el código de lado cliente
      - No generaría los ficheros en color marrón de la siguiente página
    - Uso de la opción `-S` generaría solo el código de lado servidor
      - No generaría los ficheros en color azul de la siguiente página
    - En lo siguiente suponemos que el fichero **`interfaz.h`** contiene la directiva gSOAP:
      - **`gsoap <nsPrefijo-elegido> service name: <nombreServicio>`**

      y sigue la convención de empezar las nombres de funciones con:
      - **`<nsPrefijo-elegido>__`**

# Definición del servicio

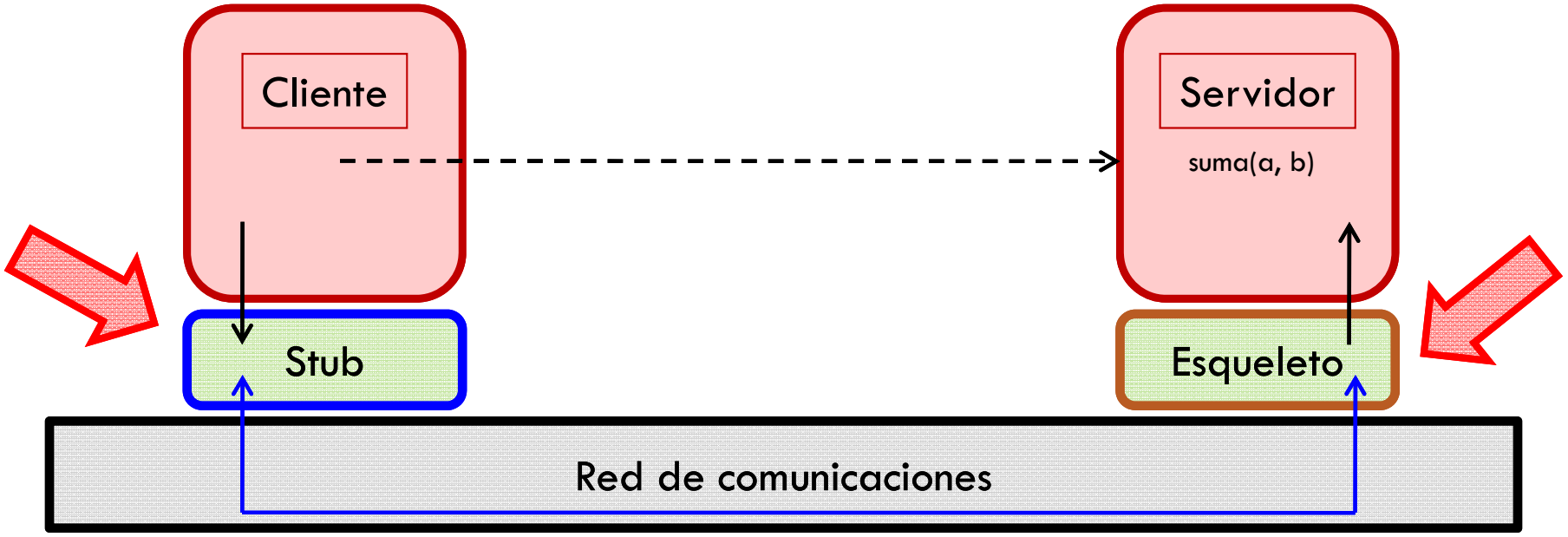


- Se generan los ficheros siguientes (automáticamente):
  - **soapStub.h**: las declaraciones de **`interfaz.h`** junto con otras del stub y del equeleto
  - **soapH.h**: las declaraciones de otras funciones generadas por gSOAP
  - **soapC.c**: el código para SOAP/XML marshalling y unmarshalling de parámetros
  - ***<nombreServicio>*.xsd** : fichero XML Schema del servicio
  - ***<nombreServicio>*.wsdl**: ficheros WSDL del servicio
  - ***<nombreServicio>*.nsmap**: tabla con el mapeo de los namespace utilizados por el servicio
  - **soapClient.c**: el stub del cliente
  - **soapClientLib.c**: la biblioteca cliente (usar en el caso de querer enlazar múltiples clientes)
  - **soapServer.c**: el esqueleto del servidor
  - **soapServerLib.c**: la biblioteca servidor (usar en el caso de querer enlazar múltiples clientes)

# Arquitectura



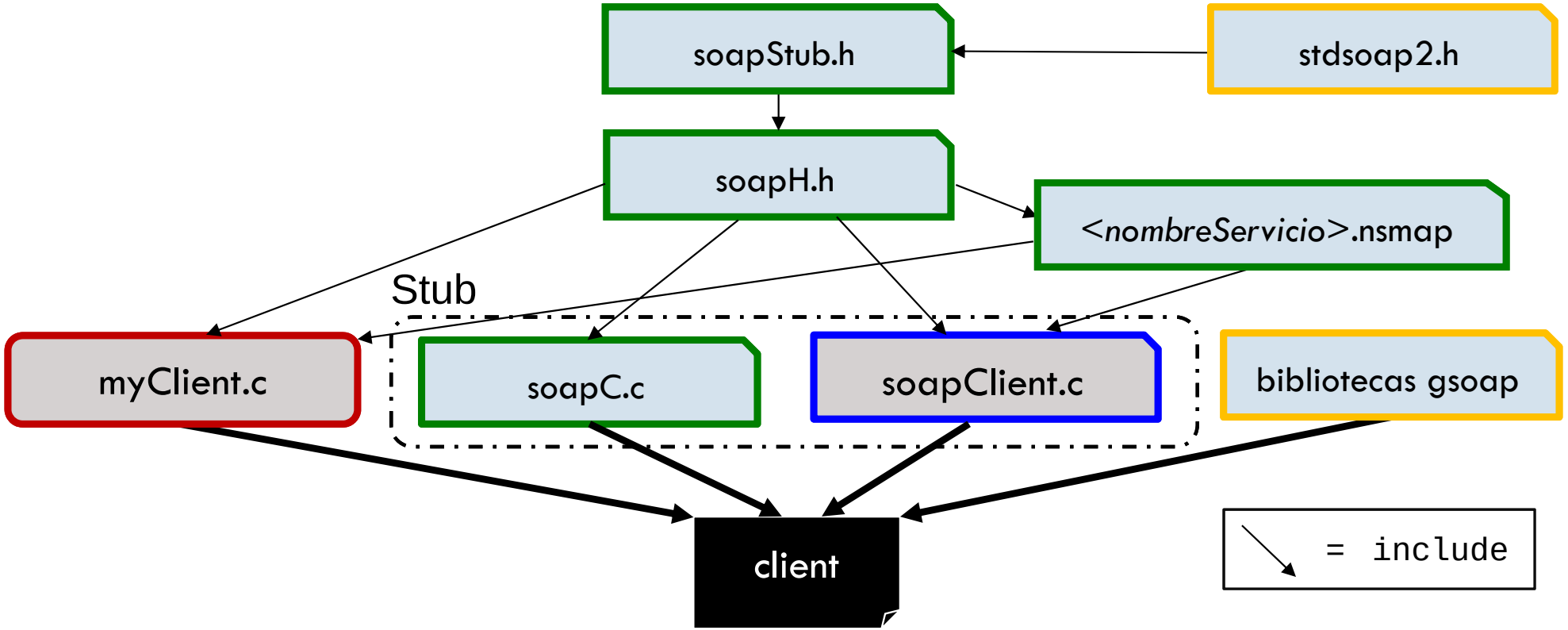
- Ya tenemos la parte azul y marrón de la siguiente figura
  - El stub del cliente y el esqueleto del servidor
  - ¿Cómo enlazamos esto con el cliente/servidor?



# Definición del cliente



- Compilamos la parte cliente (sin la opción de enlazar múltiples clientes)
- La implementación de la aplicación cliente está en `myClient.c`

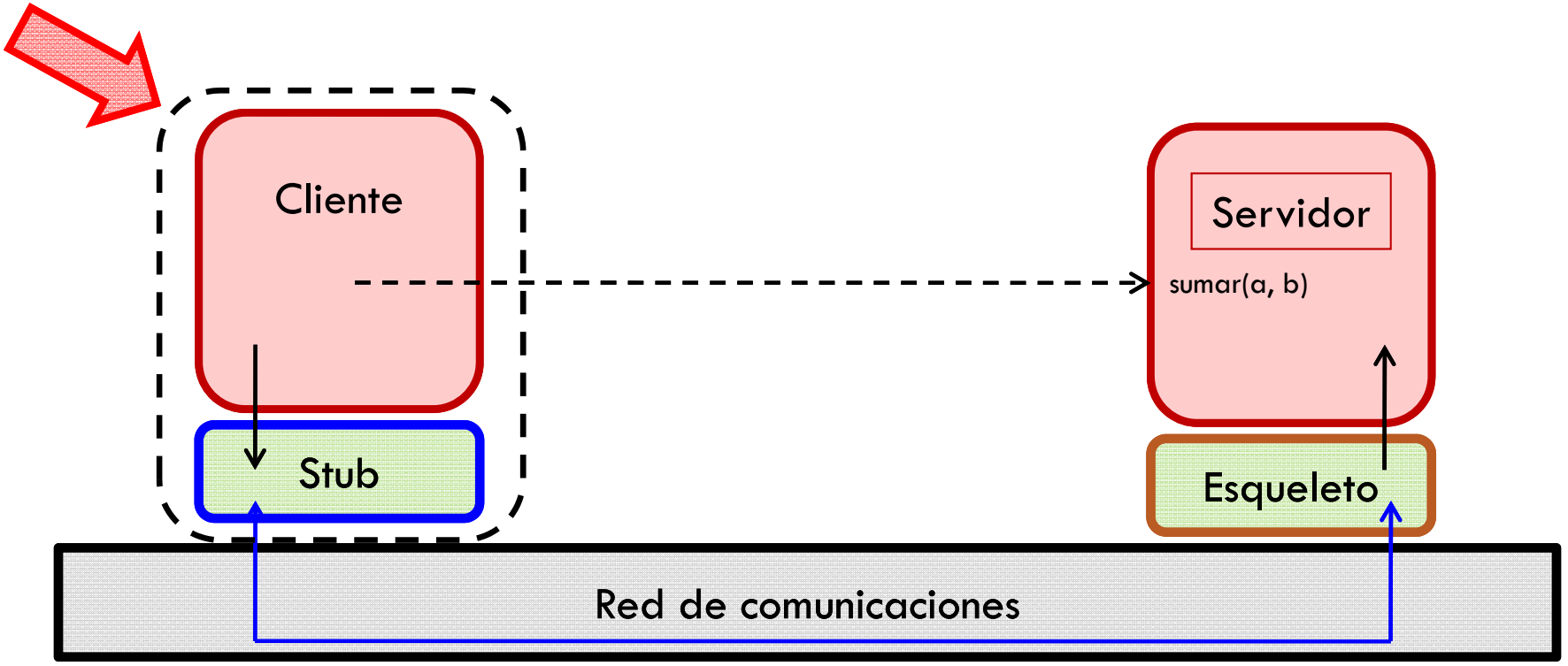



```
gcc -o client myClient.c soapC.c soapClient.c –I{GSOAP_INCLUDE} –lgsoap -L${GSOAP_LIB}
```

# Arquitectura



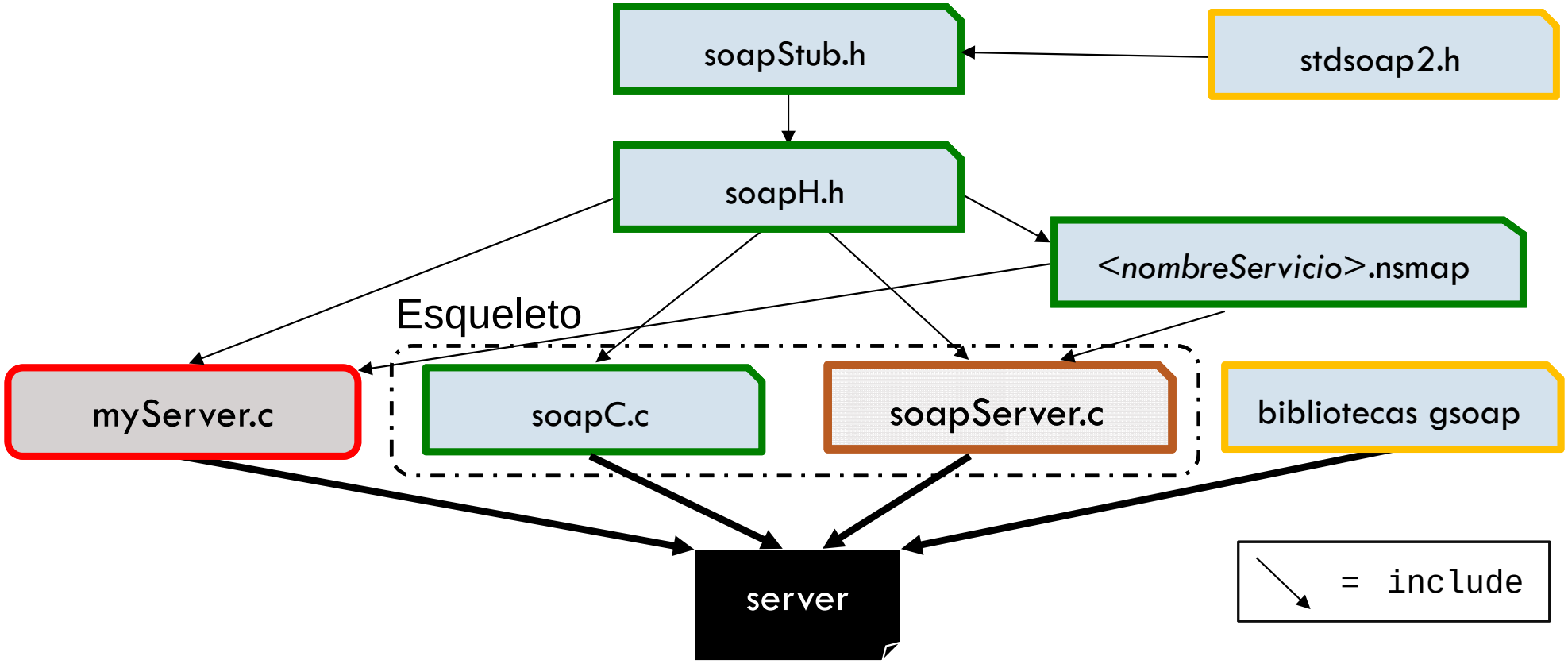
- Ya tenemos la parte cliente



# Definición del servidor



- Compilamos la parte servidor
- La implementación de la aplicación servidor está en `myServer.c`

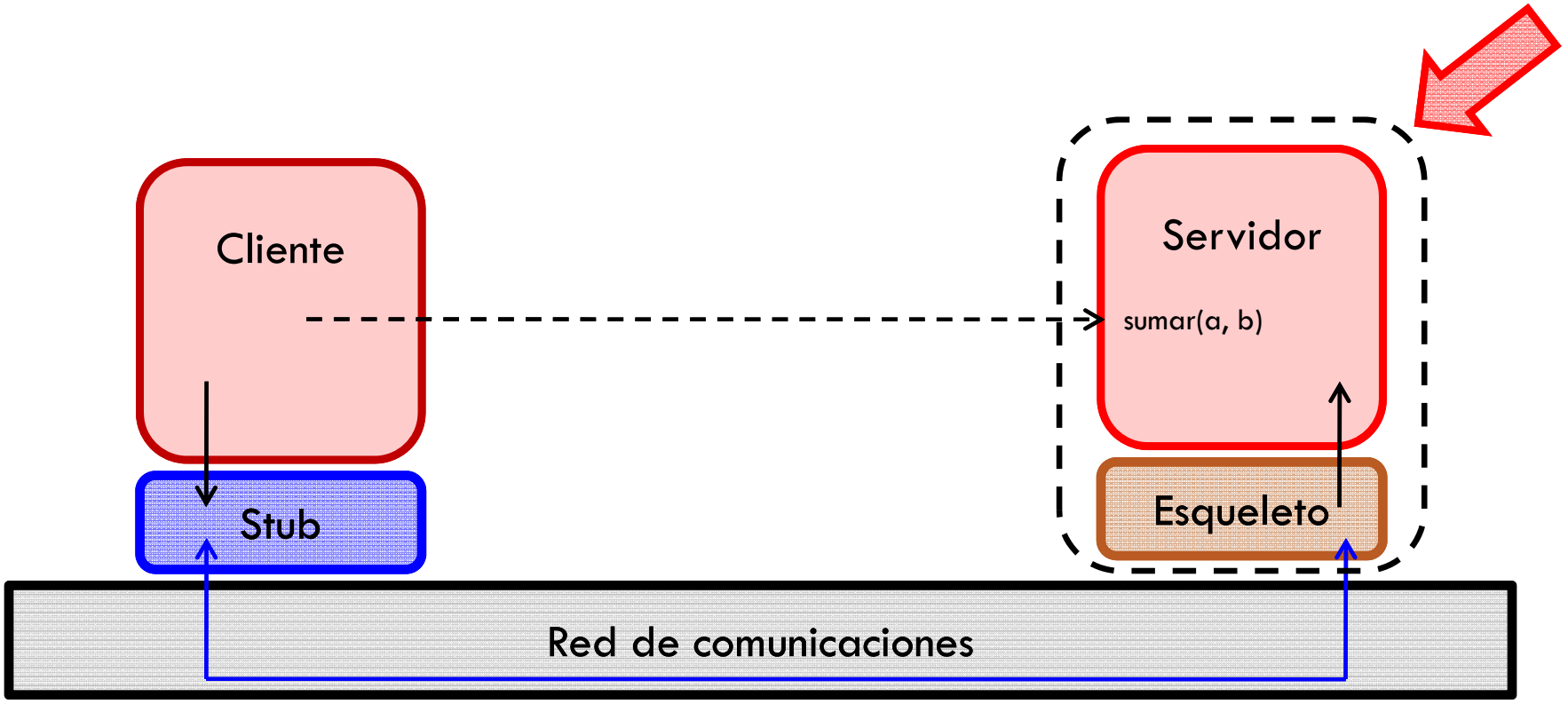



```
gcc -o server myServer.c soapC.c soapServer.c –I{GSOAP_INCLUDE} –lgsoap -L${GSOAP_LIB}
```

# Arquitectura



- Ya tenemos la parte servidor



# Convenciones



- El nombre de cada función de la interfaz se forma como sigue
  - *nsPrefijo-elegido* + __ (2 guiones bajos) + nombre de la función
    - Ejemplo: `int calcns__sumar (int a, int b, int *res)`
- Parámetros y valor de retorno
  - Cada función de la interfaz remota devuelve `int`
    - Indica si la ejecución se hizo con éxito o con error
  - Los n-1 primeros argumentos son de entrada
    - Puede tener 0 argumentos
    - Se pasan por valor
  - El último argumento es de salida
    - Se pasa por referencia
  - Implementación en C de las funciones declaradas en la interfaz
    - Argumento adicional en primera posición: contexto gSOAP

# Directivas gSOAP comunes



- //gsoap *nsPrefijo-elegido* service name: *serviceName*
  - Nombre del servicio
- //gsoap *nsPrefijo-elegido* service location: *url*
  //gsoap *nsPrefijo-elegido* service port: *url*
  - Alternativas para especificar la dirección del servicio (su localización)
- //gsoap *nsPrefijo-elegido* service style: *styleType*
  - Modo de comunicación; hay dos valores posibles: "RPC" o "document" (es decir, no RPC)
- //gsoap *nsPrefijo-elegido* service encoding: *encodingStyle*
  - Si el valor **no** es "literal" (p.ej. "encoded"), hace falta más información que lo que hay en el WSDL para realizar el marshalling y unmarshalling.
- //gsoap *nsPrefijo-elegido* service namespace: *URI-espacioNombresServicio*
  - La URI del espacio de nombres del servicio
- //gsoap *nsPrefijo-elegido* schema namespace: *URI-espacioNombresEsquema*
  - La URI del espacio de nombres del esquema del servicio

# Invocación del servicio en el cliente



- ¿Cómo se invoca cada servicio en C (en C++ es más sencillo)?
  - Devuelve un int como resultado de la invocación
  - soap es una estructura con el entorno gSOAP de ejecución
  - URL es la localización del servicio
  - action son las acciones SOAP especificadas (“” si es no hay)
  - inparam: son los argumentos de entrada
  - outparam es el argumento de salida

```
int soap_call_nsPrefijo-elegido__method_name(struct soap *soap,
                                             char *URL,
                                             char *action,
                                             inparam1..n,
                                             outparam);
```

# Entorno SOAP



- El mismo entorno se puede utilizar para invocar varios servicios
- Se utilizan varios entornos en ejecuciones multi-hilo (multithread)
  - Se garantiza el uso exclusivo de cada thread
- Las funciones más relevantes:
  - `int soap_init(struct soap *soap)`
    - Inicializa el entorno (en la pila)
  - `struct soap* soap_new()`
    - Asignar e inicializar un entorno de ejecución (en el heap)
  - `struct soap* soap_copy(struct soap *soap)`
    - Copia el contenido de un entorno de ejecución en otro

# Entorno SOAP



- Las funciones más relevantes:
  - `int soap_destroy(struct soap *soap)`
    - Borra las instancias (sólo C++)
  - `int soap_end(struct soap *soap)`
    - Finaliza el entorno de ejecución y borra los datos
  - `int soap_free(struct soap *soap)`
    - Desvincula y libera el entorno de ejecución (asignado en el heap)
  - `int soap_done(struct soap *soap)`
    - Desvincula el entorno de ejecución (asignado en la pila)
  - `int soap_serve(struct soap *soap)`
    - Atiende y lanza la invocación del servicio correspondiente

# Estructura del cliente en C



```
#include “soapH.h”
#include “ns.nsmap”

int main() {

     // Entorno de ejecución
     struct soap soap;
     ...

     /* Inicializar entorno de ejecución */
     soap_init (&soap);
     ...

     /* Invocar el servicio correspondiente */
     soap_call_nsPrefijo-elegido__serviceName (&soap, in1, in2, …, output);
     ...


     /* Finalizar entorno de ejecución */
     soap_end (&soap);
     soap_done (&soap);
}
```

# Servidor CGI en C



```
#include “soapH.h”
#include “ns.nsmap”

int main() {

     // Entorno de ejecución
     struct soap soap;
     ...
     /* Inicializar entorno de ejecución */
     soap_init (&soap);
     ...
     /* Lanzar un servicio web */
     soap_serve  (&soap);
     ...

}

/* Implementación del servicio */
int serviceName (in1, in2, . . ., output){

     // Hacer algo…

     return SOAP_OK;
}
```

# Comunicación



- Enlazar un entorno de ejecución a una conexión
  - `int soap_bind(struct soap *soap, char *host, int port, int backlog)`
    - Devuelve el socket primario del servidor
- Aceptar la conexión
  - `int soap_accept(struct soap *soap)`
    - Devuelve el socket secundario

# Servidor stand-alone en C



```
#include “soapH.h”
#include “ns.nsmap”

int main(){
      struct soap soap;   /* Entorno de ejecución */
      int m,s;            /* Sockets */

      /* Inicializar el entorno gSOAP */
      soap_init(&soap);

      /* Asociar el entorno gSOAP al socket que escuchará en la dirección IP y puerto dados */
      if ((m=soap_bind(&soap, DIRECCION_DEL_HOST, PUERTO, 100))<0){
            // Tratar el error!
      }

      /* En un servidor multi-hilo, habría un llamada a soap_serve, soap_end y soap_done por cada hilo */
      for(;;){
            /* Aceptar conexiones entrantes */
            if ((s=soap_accept(&soap))<0){
            // Tratar el error
            }
            /* Invocar el servicio correspondiente */
            if (soap_serve(&soap) != SOAP_OK){
                soap_print_fault(&soap, stderr);
                exit(-1);
            }
            /* Borrar datos y cerrar socket */
            soap_end(&soap);
      }
      /* Cerrar socket principal y desvincula el contexto */
      soap_done(&soap);
      exit(0);
}
```

# Instalación de gSOAP



- Bajar la versión gSoap **gSOAP 2.8.24 stable**
  - `http://sourceforge.net/projects/gsoap2/files`
- Copiar el fichero `gsoap_2.8.24.zip` en **$HOME**
- Descomprimir
  - `unzip gsoap_2.8.24.zip`
  - Se generará un directorio `gsoap-2.8`
- Debe tener instalado previamente
  - *Bison, flex, yacc y ssl*
- Configuramos para instalar
  - `./configure`

# Instalación de gSOAP



- Compilamos
  - `make`
- Creamos directorio de instalación
  - `mkdir $HOME/gsoap-linux_2.8.24`
- Instalamos
  - `make install exec_prefix=$HOME/gsoap-linux_2.8.24`
  - Si hay problemas de permisos:
    `sudo make install exec_prefix=$HOME/gsoap-linux_2.8.24`
- Incluimos en `$HOME/.bashrc`
  - `export GSOAP_HOME=$HOME/gsoap-linux_2.8.24`
  - `export GSOAP_LIB=${GSOAP_HOME}/lib`
  - `export GSOAP_INCLUDE=${GSOAP_HOME}/include`
  - `PATH=$PATH:$GSOAP_HOME/bin`

# Compilar ejemplo calculadora



- Bajamos el fichero `calcExample.tgz`
- Descomprimimos
  - `tar -zxvf calcExample.tgz`
- Vamos al directorio creado
  - `cd calcExample`
- Generamos los stubs
  - `soapcpp2 -c calc.h`
- Compilamos
  - `make`

# Ejecutar ejemplo calculadora



- Ejecución del servidor (elegimos un número de puerto encima de 255)
  - `./server 22500`
- Ejecución del cliente
  - `./client http://localhost:22500 5 + 9`
  - `./client http://localhost:22500 5 - 9`
  - `./client http://localhost:22500 5 m 9`
  - `./client http://localhost:22500 5 / 9`