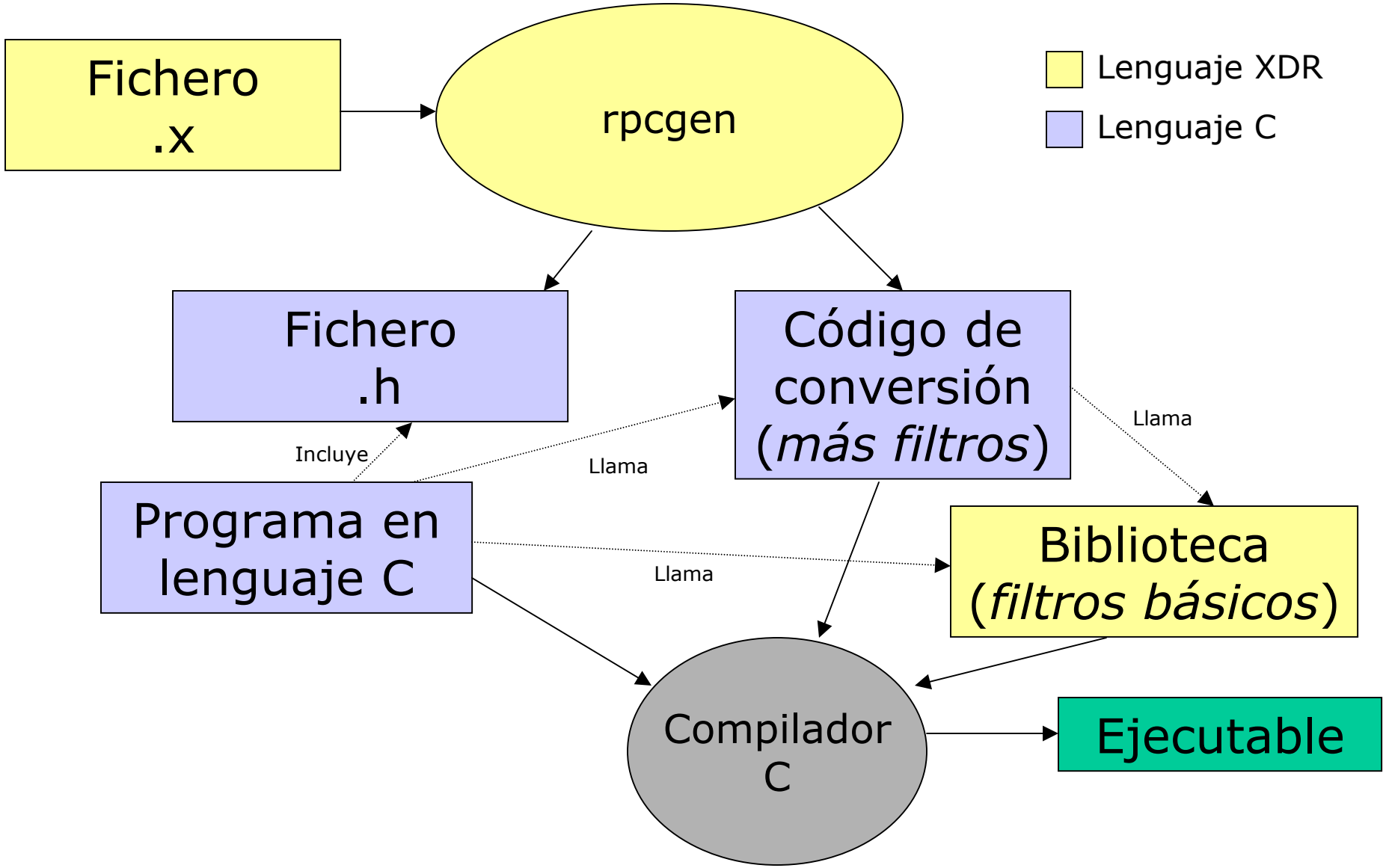
Programación con XDR

(Desde el lenguaje C)

Generalidades

- Un fichero de extensión .x tiene las declaraciones de tipos XDR.
- La herramienta *rpcgen* genera un .h (declaraciones en C de esos tipos)
- Un dato C se codifica en XDR llamando a un *filtro*.
  - Los filtros de los datos básicos están en una **biblioteca** (**`xdr_int, xdr_float...`**)
  - Los filtros de los datos de usuario son **funciones C** escritas por rpcgen (**`xdr_huevo, xdr_huevera`**)

Esquema de desarrollo (particularizado)



Sintaxis genérica de un filtro

- Los filtros de tipos simples y los de tipos compuestos tienen la misma sintaxis.

```
bool_t xdr_tipo(XDR *xdrs,
                tipo *dato)
```

- El primer parámetro define:
  - El sentido de la conversión (codificación o decodificación)
  - El lugar donde se escribirá o leerá el dato en codificación XDR.
- El segundo parámetro es un puntero al dato en su "codificación C"

Inicialización del parámetro `XDR*`

- Dos posibles destinos/fuentes del dato codificado en XDR:
  - Fichero (o *socket*)
  - Memoria
- Dos funciones para inicializar la estructura **`XDR`**:
  - **`xdrstdio_create`**
  - **`xdrmem_create`**

Sintaxis

- **bool_t xdrstdio_create(**
  XDR *xdrs,
  FILE *fichero,
  const enum xdr_op op
)
- Posibles valores de *op*:
  - **XDR_ENCODE**
  - **XDR_DECODE**
  - **XDR_FREE**

Sintaxis

- **bool_t xdrmem_create(**
  XDR *xdrs,
  const caddr_t *direccion,
  const u_size tamaño,
  const enum xdr_op op
)
- Posibles valores de *op,* igual que en la función anterior.

Ejemplo: escribir datos simples en fichero

```
#include<stdio.h>
#include<rpc/rpc.h>
main() {
 FILE *fichero;
 XDR hxdr;
 int x; float y;

 fichero=fopen("datos.dat", "w");
 xdrstdio_create(&hxdr, fichero,
                        XDR_ENCODE);
 x=0x2548;
 y=12.5;
 xdr_int(&hxdr, &x);
 xdr_float(&hxdr, &y);
}
```

declaraciones de filtros simples y funciones

estructura XDR

Inicialización de la estructura XDR: codificar sobre el fichero

Llamadas a los filtros (escriben codigo XDR en el fichero "datos.dat")

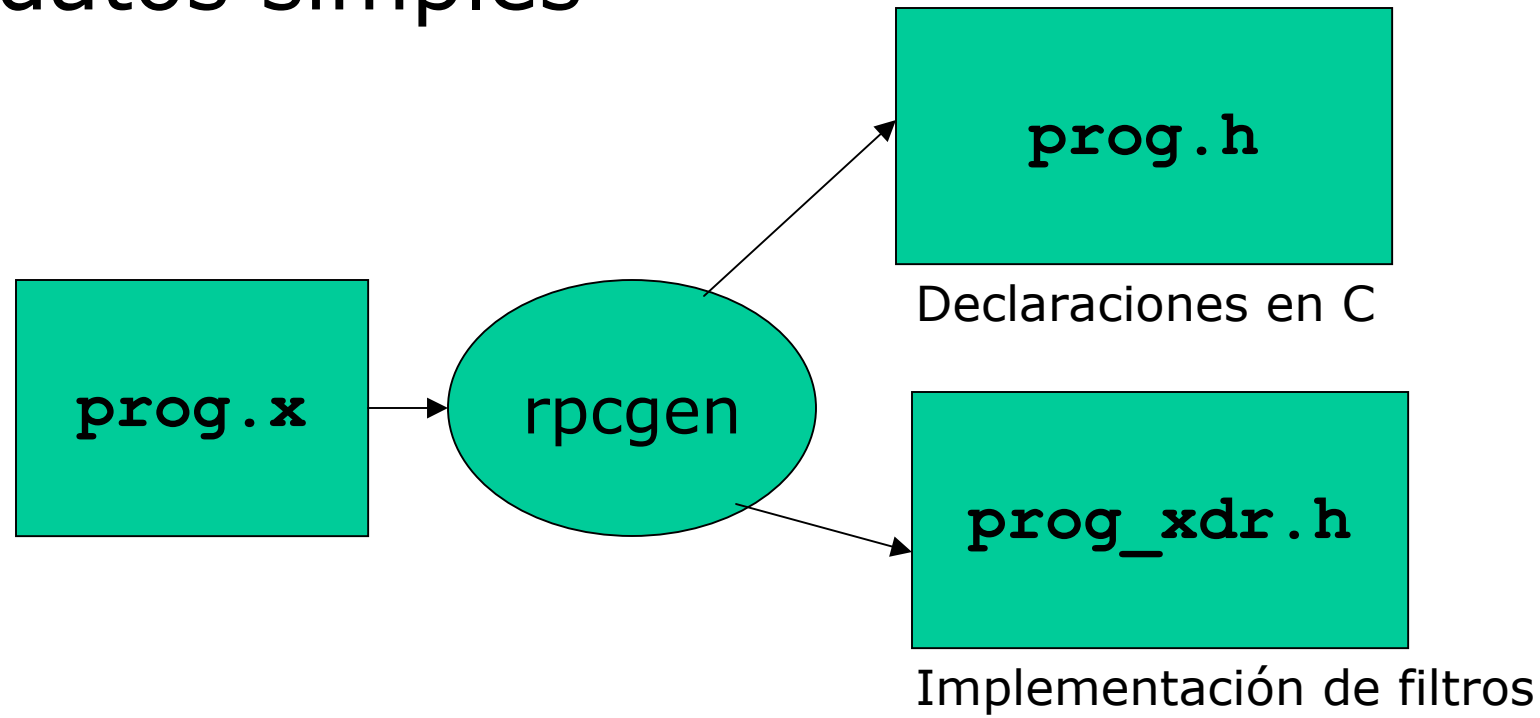
Filtros para datos simples

Tipo XDR	Tipo C	Filtro
`int`	`int`	`xdr_int`
`unsigned int`	`unsigned int`	`xdr_u_int`
`bool`	`bool_t`	`xdr_bool`
`hyper`	`longlong_t`	`xdr_hyper`
`unsigned hyper`	`u_longlong_t`	`xdr_u_hyper`
`float`	`float`	`xdr_float`
`double`	`double`	`xdr_double`

Dependiente de la máquina (Ej, en linux sería `int64_t, uint64_t`)

Filtros para datos compuestos

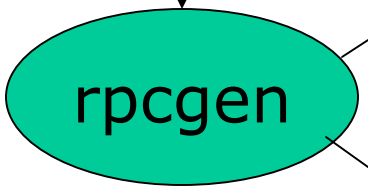
- La herramienta rpcgen genera código C que implementa estos filtros mediante llamadas a los filtros para datos simples



Datos compuestos. Ejemplo

prog.x

```
struct prueba {
   int x;
   float y;
   opaque z<45>;
};
```



prog.h

```
struct prueba {
   int x;
   float y;
   struct {
     u_int z_len;
     char *z_val;
   } z;
}
typedef struct prueba prueba;
extern bool_t xdr_prueba(
                XDR *, prueba *);
```

prog_xdr.c

Implementación de **xdr_prueba** (trasp. siguiente)

Datos compuestos: Ejemplo (cont.)

prog_xdr.c

```
#include "prueba.h"

bool_t
xdr_prueba(XDR *xdrs, prueba *objp)
{
  if (!xdr_int (xdrs, &objp->x))
        return FALSE;
  if (!xdr_float (xdrs, &objp->y))
        return FALSE;
  if (!xdr_bytes (xdrs, (char **)&objp->z.z_val,
                  (u_int *) &objp->z.z_len, 45))
        return FALSE;
  return TRUE;
}
```

Codificar campo x

Codificar campo y

Codificar campo z (longitud máxima 45)

Datos compuestos: Ejemplo (cont.)

ejemplo.c

```
#include "prueba.h"

main() {
    prueba mi_ejemplo;
    XDR hxdr;
    char mis_bytes[20];

    /* Inicialización de hxdr no mostrada */

    /* Inicialización de array mis_bytes no mostrada */
    mi_ejemplo.x=1234;
    mi_ejemplo.y=12.333;
    mi_ejemplo.z.z_len=20;
    mi_ejemplo.z.z_val=&mis_bytes[0];

    xdr_prueba(&hxdr, &mi_ejemplo); /* Conversión */
}
```

Datos compuestos: Ejemplo (y fin)

- Secuencia de comandos para compilación:

```
$ gcc -c ejemplo.c
$ gcc -c prog_xdr.c
$ gcc -o ejemplo ejemplo.o prog_xdr.o -lbiblioteca
```

- La biblioteca que implementa los filtros básicos depende de la máquina:
  - linux, OSF: libc (no es necesario especificarla)
  - solaris: libnsl (especificar **`-lnsl`**)

Equivalencias entre XDR y C

- No se garantiza que el código generado por **`rpcgen`** en una máquina sea válido para otra.
- No obstante se cumplen una serie de reglas generales:
  - Arrays de longitud fija son lo mismo en C.
  - Estructuras y enumerados se convierten en lo mismo en C, más un **`typedef`** C.
  - Restantes casos: ver transparencia siguiente

Equivalencias entre XDR y C

Tipo XDR	Tipo C
tipos simples	Ver trasparencia 9
opaque	char
string	puntero a char
array de long. variable	Estructura con dos campos: longitud y puntero a datos
unión discriminada	Estructura con dos campos: discriminante y unión C
dato opcional	puntero

Equivalencias: strings

Tipo XDR	Tipo C
`typedef string texto<250>;`	`typedef char *texto;`

Ejemplo de uso en C

```
texto msg; /* lo mismo que char *msg */
...
/* Posibles formas de inicializarlo */
msg="Esto es una prueba"; /* Cuidado! */
msg=malloc(250);
strcpy(msg, "Esto es una prueba");
msg=strdup("Esto es una prueba");
```

Equivalencias: array de longitud variable

Tipo XDR

```
typedef float
   casidoce<12>;
```

Tipo C

```
typedef struct {
   u_int  casidoce_len;
   float  *casidoce_val;
} casidoce;
```

Ejemplo de uso en C

```
casidoce v;
...
v.casidoce_val=malloc(7*sizeof(float));
for (i=0; i<7; i++)
  v.casidoce_val[i]=i/2.0;
v.casidoce_len=7;
```

Equivalencias: unión discriminada

Tipo XDR

```
union ejemplo
switch (int que) {
 case 1: int x;
 case 2: float y;
 default:
   string error<>;
};
```

Tipo C

```
struct ejemplo {
   int que;
   union {
      int x;
      float y;
      char *error;
   } ejemplo_u;
};
typedef struct ejemplo ejemplo;
```

- En C da lugar a una estructura
  - Un campo es el discriminante
  - El otro es una unión, con un nombre terminado en `_u`

Unión discriminada

Ejemplo de uso en C

```
ejemplo ej1;
int a, b; /* Vamos a dividir estos dos enteros
             y guardar el resultado en ej1 */
...
if (b!=0) {
  if (a % b == 0) { /* Cociente entero */
    ejemplo.que=1;
    ejemplo.ejemplo_u.x=a/b;
  } else {          /* Cociente real */
    ejemplo.que=2;
    ejemplo.ejemplo_u.y=((float)a)/((float)b);
  }
} else { /* Division por cero */
   ejemplo.que=-1;
   ejemplo.ejemplo_u.error="Error: división por cero";
}
```

Equivalencias: datos opcionales

Tipo XDR

```
struct lista {
   int dato;
   lista *otro;
};
```

Tipo C

```
struct lista {
    int dato;
    struct lista *otro;
};
typedef struct lista lista;
```

- Un dato opcional, en C da lugar a un puntero
  - Si el puntero vale NULL, el filtro lo codifica como 0000h
  - Si es distinto de NULL el filtro lo codifica como 0001h seguido de la codificación del dato a que apunta

Dato opcional: Ejemplo

Ejemplo de uso en C (lista con tres elementos)

```
lista p; /* primer elemento de la lista */
...
p.dato=1;
p.otro=malloc(sizeof(lista)); /* Crear segundo */
p.otro->dato=2;
p.otro->otro=malloc(sizeof(lista)); /* Crear tercero */
p.otro->otro->dato=3;
p.otro->otro->otro=NULL;  /* Éste es el último */
...
xdr_lista(&hxdr, &p); /* Codifica la lista completa */
```

Codificación

00 00 00 01	00 00 00 01	00 00 00 02	00 00 00 01	00 00 00 03	00 00 00 00

Nota sobre punteros y decodificación

- Algunos tipos XDR (*strings*, datos opcionales, ...) se convierten en punteros en C
- El valor del puntero influye en el comportamiento del filtro DECODE
  - Si es distinto de NULL, el filtro guarda el dato decodificado en la dirección proporcionada
  - Si es NULL, el filtro reserva memoria para el dato decodificado, y asigna al puntero la dirección reservada

Ejemplo

Decodificación de una lista con N elementos

```
lista p; /* primer elemento de la lista */
...
/* Inicializamos a NULL el campo "otro" */
p.otro=NULL;

...

/* Asumiendo que hxdr ha sido inicializado con
   XDR_DECODE para decodificar la lista completa
   basta llamar al filtro */

xdr_lista(&hxdr, &p);

/* En p.otro tendremos ahora un puntero al segundo
   dato, y en p.otro->otro al tercero, etc. */
```

XDR y sockets

- Un filtro XDR puede volcar/recoger los datos codificados de un fichero
- Un *socket* es un descriptor de fichero
- Por tanto, un filtro XDR puede operar directamente sobre un *socket*.
  - El socket debe ser tipo STREAM
  - **`fdopen`** permite asociar un FILE* a un socket
  - El FILE* obtenido puede usarse con **`xdrstdio_create`**
  - Para asegurar que los datos generados por el filtro son enviados, es necesario usar **`fflush`**

Ejemplo

Enviar un entero por un socket

```
#includes...
main()
{
    int sock, dato; struct sockaddr_in dir;
    FILE *fsock;    XDR hxdr;

    sock=socket(PF_INET, SOCK_STREAM, 0);
    dir.sin_family=AF_INET; dir.sin_port=htons(15000);
    dir.sin_addr.s_addr=inet_addr("156.35.151.2");

    fsock=fdopen(sock, "w+");
    xdrstdio_create(&hxdr, fsock, XDR_ENCODE);
    connect(sock, &dir, sizeof dir);

    dato=1228;
    xdr_int(&hxdr, &dato);

    fclose(fsock); close(sock);
}
```

Envío del dato por el socket

Observaciones, E/S *buffereada*

- Cuando se escribe en un **`FILE*`** los datos quedan almacenados en un *buffer* de la estructura **`FILE`**.
- Por tanto, no son enviados inmediatamente a su destino.
- El *buffer* es vaciado cuando se llena, o cuando se hace **`fclose`** o **`fflush`** sobre el **`FILE*`**
- Al leer y escribir en el mismo **`FILE*`**, los datos pueden mezclarse en el *buffer*, causando problemas

Recomendaciones

- Hacer **`fflush`** tras llamar a cada filtro XDR, para asegurarse de que los datos se envían en ese momento
- Cuando se tiene que leer y escribir en el mismo *socket*, usar un **`FILE*`** para lectura y otro para escritura, para evitar "choques" en el buffer

```
int sock; FILE *f_lect, *f_escr;
XDR hxdr_lect, hxdr_escr;
...
f_escr=fdopen(sock, "w");   f_lect=fdopen(sock, "r");
xdrstdio_create(&hxdr_escr, f_escr, XDR_ENCODE);
xdrstdio_create(&hxdr_lect, f_lect, XDR_DECODE);
```