

Bibliotecas partilhadas (*shared objects*)

Bib: Computer Systems: A Programmer's Perspective (cap. 7, .10-.12)

Programação em Sistemas Computacionais

João Pedro Patriarca (joao.patriarca@isel.pt), Gabinete F.0.23 do edifício F

ISEL, ADEETC, LEIC

Agenda

- Motivação, criação de uma biblioteca, utilização e processo de ligação
- Código PIC (*Position Independent Code*)
- Carregamento em tempo de execução

Agenda

- Motivação, criação de uma biblioteca, utilização e processo de ligação
- Código PIC (*Position Independent Code*)
- Carregamento em tempo de execução

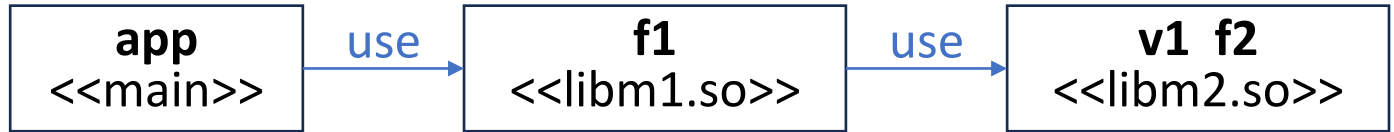
Aspectos negativos das bibliotecas estáticas

- Uma atualização da biblioteca implica ligá-la novamente a todos os programas que pretendam refletir a atualização
- Repetição de código nos executáveis que dependam da mesma biblioteca:
 - Quer no ficheiro executável em disco;
 - Quer em memória RAM quando o programa é executado e carregado em memória
- Aspecto grave se considerarmos a ligação estática da biblioteca standard do C pela maioria dos programas

Bibliotecas partilhadas

- Ficheiros objeto com formato ELF
- Denominadas, igualmente, por *shared objects* (objetos partilhados), com extensão **.so** nos sistemas Linux (**.dll** nos sistemas Windows)
- Partilhados porque:
 - Existe uma única biblioteca no sistema, mesmo sendo usada por múltiplas aplicações;
 - Ligada ao programa e trazida para memória apenas quando é executada a aplicação ou explicitamente pelo programa em tempo de execução;
 - O próprio código da biblioteca pode ser partilhado por múltiplos programas em execução (desde que não seja alterado no carregamento).
- A ferramenta responsável pela ligação chama-se *dynamic linker*
- ✓ Melhor utilização do recurso memória e simplificação de atualizações de bibliotecas
- x A atualização de uma biblioteca pode introduzir erros que não existiam em versões anteriores (*DLL Hell*)

Exemplo base



m1.c

```
#include "m.h"

int f1(void) {
    return v1 + v1 + f2() + f2();
}
```

m2.c

```
int v1 = 10;

int f3() { return v1 * 10; }
int f2() { return v1 + f3(); }
```

m.h

```
#ifndef _M_H
#define _M_H

int f1(void);
int f2(void);
int f3(void);
extern int v1;

#endif/*_M_H*/
```

app.c

```
#include <stdio.h>
#include "m.h"

int main() {
    printf("f1() = %d\n", f1());
    return 0;
}
```

Comandos para criar bibliotecas e aplicação

```
$ gcc -Wall -Og -c app.c
$ gcc -Wall -Og -fpic -shared m1.c -o libm1.so
$ gcc -Wall -Og -fpic -shared m2.c -o libm2.so
$ gcc -o app app.o ./libm1.so ./libm2.so
```

- A opção de ligação **-shared** cria uma biblioteca partilhada em vez de um executável
- A opção de compilação **-fpic** (*position independent code*) é obrigatória no âmbito da geração de uma biblioteca partilhada

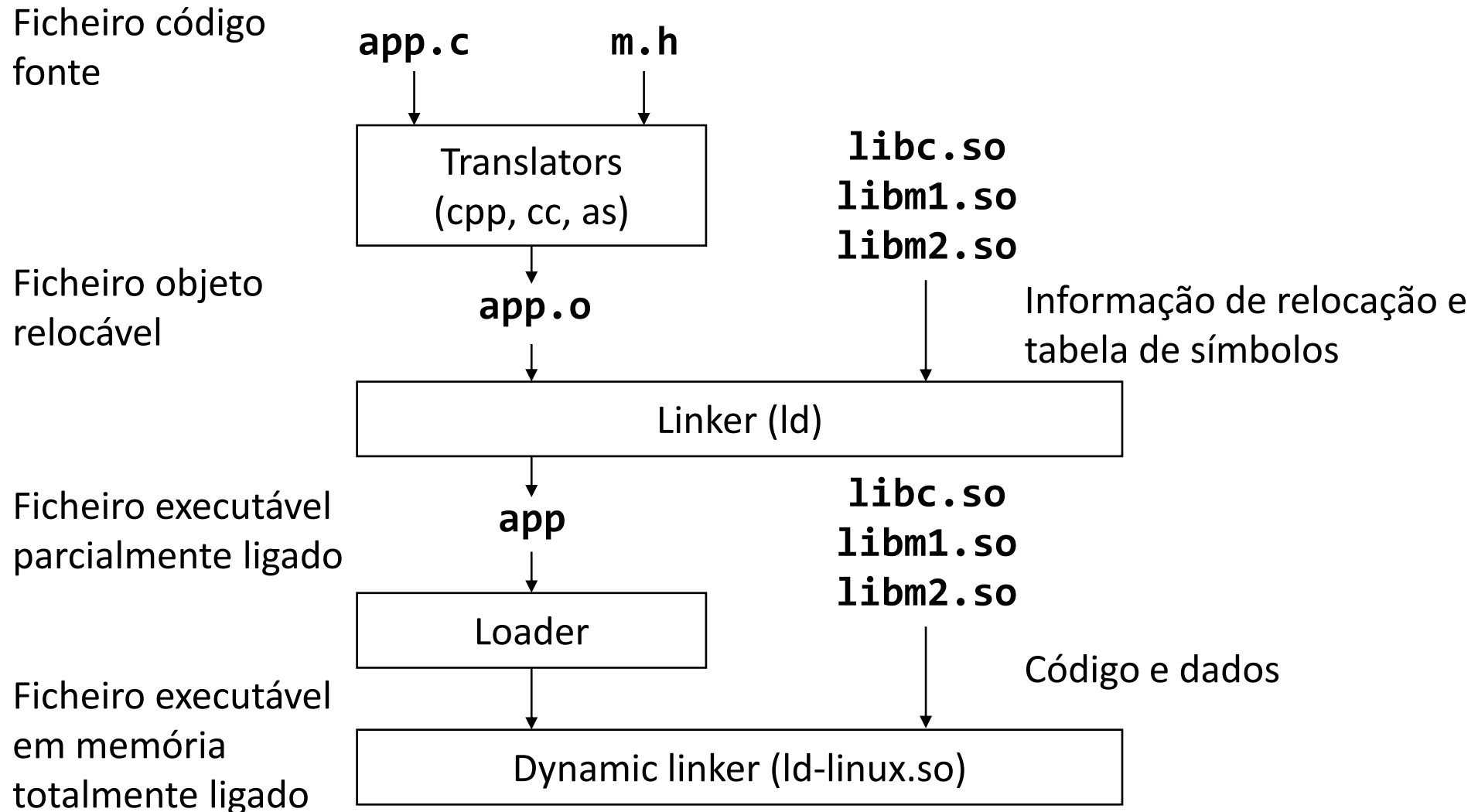
```
$ nm app
...
U f1
0...01169 T main
```

```
$ nm libm1.so
...
0...01119 T f1
U f2
U v1
```

```
$ nm libm2.so
...
0...0112d T f2
0...01119 T f3
0...04010 D v1
```

- Na fase de ligação estática não é copiado nem código nem dados das bibliotecas para o executável; apenas é copiada informação relativa a dados de relocação e tabela de símbolos que permitirão a resolução de referências em tempo de carregamento (opções **-R** e **-T** no **objdump**)

Carregamento do executável em memória



Ações durante o carregamento do executável

- Ações do *loader*:
 - Carrega o executável em memória
 - Verifica a existência da secção `.interp` no executável
 - A secção `.interp` identifica o caminho no sistema para o *linker* dinâmico
 - Em vez de passar o controlo para a aplicação, carrega e executa o *linker* dinâmico
- Ações do *linker* dinâmico:
 - Reloca código e dados da biblioteca `libc.so`
 - Reloca código e dados das bibliotecas `libm1.so` e `libm2.so`
 - Resolve as referências indefinidas
 - Passa o controlo para a aplicação principal
- A localização das bibliotecas permanecerá fixa ao longo da execução da aplicação, ou seja, os valores dos símbolos ficam fixos

Secção **.interp**

```
$ objdump -h app
```

```
app:      file format elf64-x86-64
```

```
Sections:
```

Idx	Name	Size	VMA	LMA	File off	Algn
0	.interp	0000001c	0...00318	0...00318	00000318	2**0
		CONTENTS, ALLOC, LOAD, READONLY, DATA				
	...					

```
$ objdump -s -j .interp app
```

```
app:      file format elf64-x86-64
```

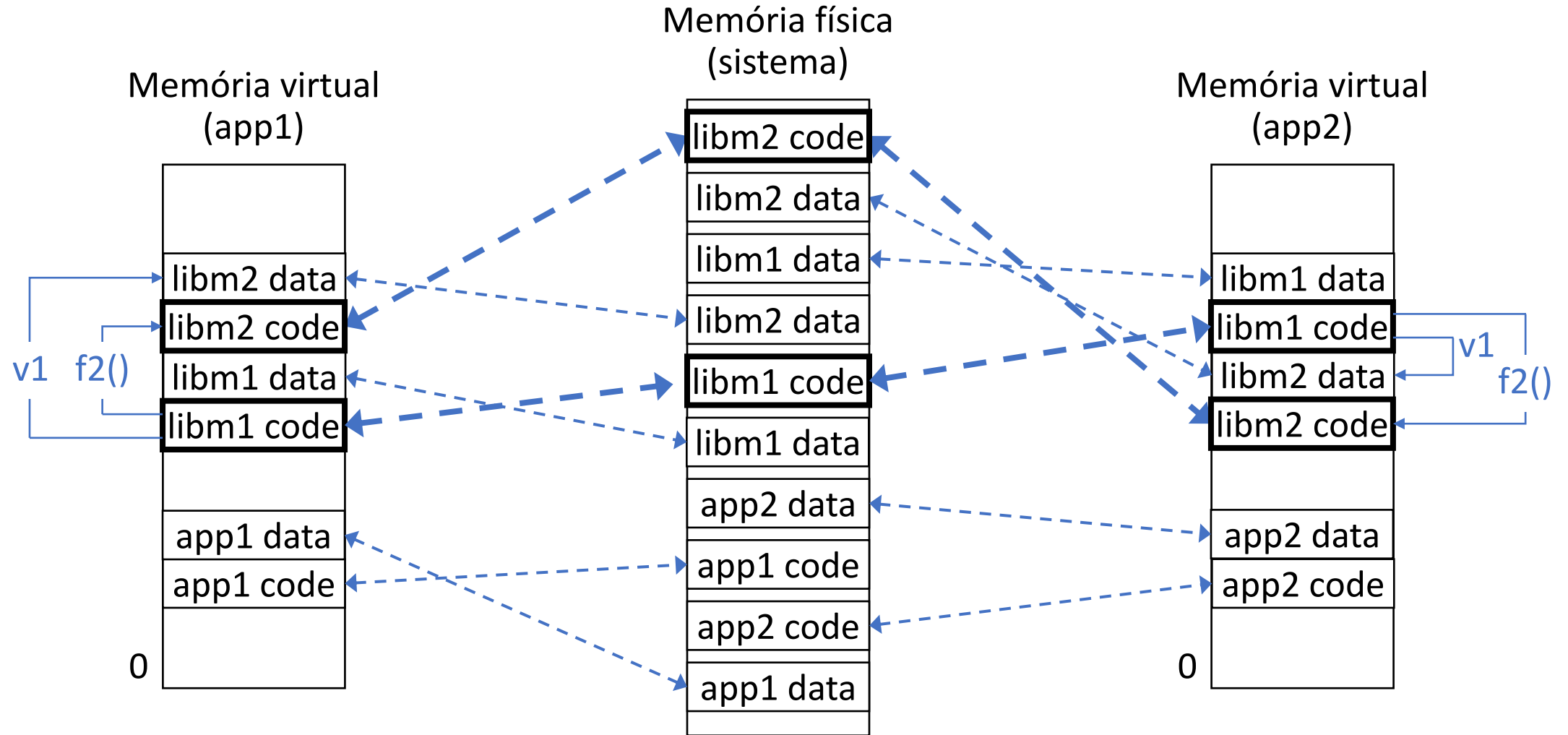
```
Contents of section .interp:
```

0318	2f6c6962	36342f6c	642d6c69	6e75782d	/lib64/ld-linux-
0328	7838362d	36342e73	6f2e3200		x86-64.so.2.

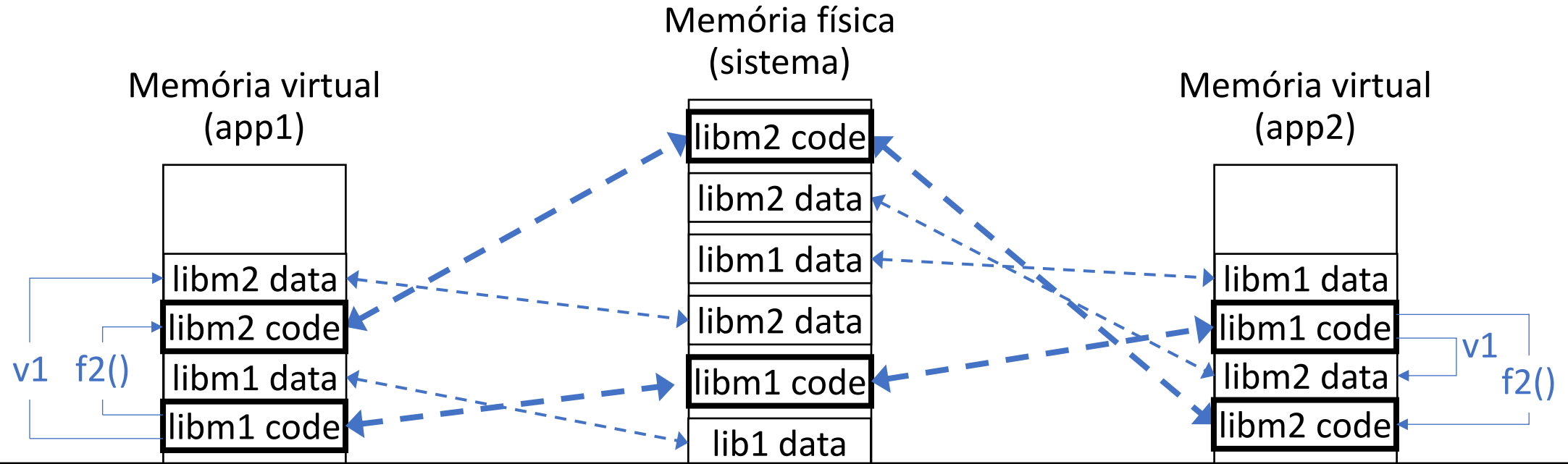
Agenda

- Motivação, criação de uma biblioteca, utilização e processo de ligação
- Código PIC (*Position Independent Code*)
- Carregamento em tempo de execução

Partilha em memória por várias aplicações



Partilha em memória por várias aplicações



Qualquer alteração na fase de ligação dinâmica de uma secção .text, impede que a mesma possa ser partilhada. Sendo a posição relativa de f2 e v1 diferentes em cada aplicação, como é possível resolver as referências em libm1 e ainda assim manter o código de libm1 partilhável?

Hipótese: considerar posições relativas constantes

1. Atribuir a cada biblioteca um endereço predefinido no espaço de endereçamento das aplicações
2. Carregar a biblioteca sempre no endereço predefinido
 - Embora simples, esta solução apresenta vários problemas:
 - x Ineficiente na gestão do espaço de endereçamento porque parte do espaço fica alocado mesmo que a aplicação não utilize a biblioteca
 - x De difícil gestão, uma vez que é preciso garantir que não exista qualquer sobreposição entre bibliotecas
 - x Na alteração de uma biblioteca, é preciso garantir que a nova dimensão não ultrapassa o espaço alocado à biblioteca, caso contrário será necessário alocar novo espaço
 - x Para cada nova biblioteca terá de ser alocado novo espaço
 - Considerando centenas de bibliotecas, rápido se percebe que esta solução não é viável

Código PIC (*Position Independent Code*)

- O código gerado pelo compilador é gerado de forma a que fique independente da localização, ou seja, não seja necessário resolver referências no código
- O módulo tem de ser compilado com a opção **-fpic**
- Referências a símbolos e respetivas definições, exclusivamente dentro da mesma biblioteca, não carecem de resolução porque a posição relativa da referência à respetiva definição é conhecida em tempo de ligação estática
- Referências a símbolos definidos em outras bibliotecas precisam de tratamento especial porque a posição relativa da referência à respetiva definição apenas é conhecida após carregamento das bibliotecas em memória
- Para a compreensão das soluções apresentadas a seguir é importante manter presente:
 - A distância entre secções pertencentes a um mesmo ficheiro objeto, seja executável ou biblioteca, são constantes, independentemente do endereço base onde fiquem mapeados em memória

Impacto da opção **-fpic** no código gerado

m1.c

```
#include "m.h"
int f1(void) {
    return v1 + v1 + f2() + f2();
}
```

Sem -fpic

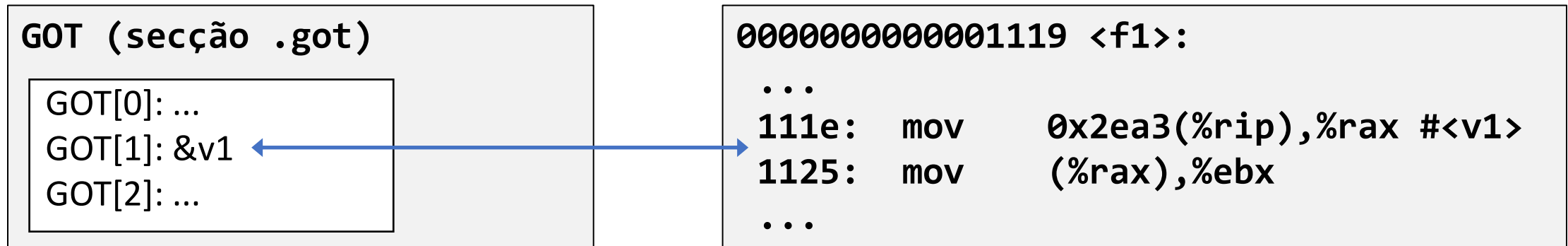
```
0000000000001178 <f1>:
1178: endbr64
117c: push    %rbx
117d: mov     0x2e8d(%rip),%eax #<v1>
1183: lea     (%rax,%rax,1),%ebx
1186: callq   11a7 <f2>
118b: add     %eax,%ebx
118d: callq   11a7 <f2>
1192: add     %ebx,%eax
1194: pop     %rbx
1195: retq
```

Com -fpic

```
0000000000001119 <f1>:
1119: endbr64
111d: push    %rbx
111e: mov     0x2ea3(%rip),%rax #<v1>
1125: mov     (%rax),%ebx
1127: add     %ebx,%ebx
1129: callq   1050 <f2@plt>
112e: add     %eax,%ebx
1130: callq   1050 <f2@plt>
1135: add     %ebx,%eax
1137: pop     %rbx
1138: retq
```

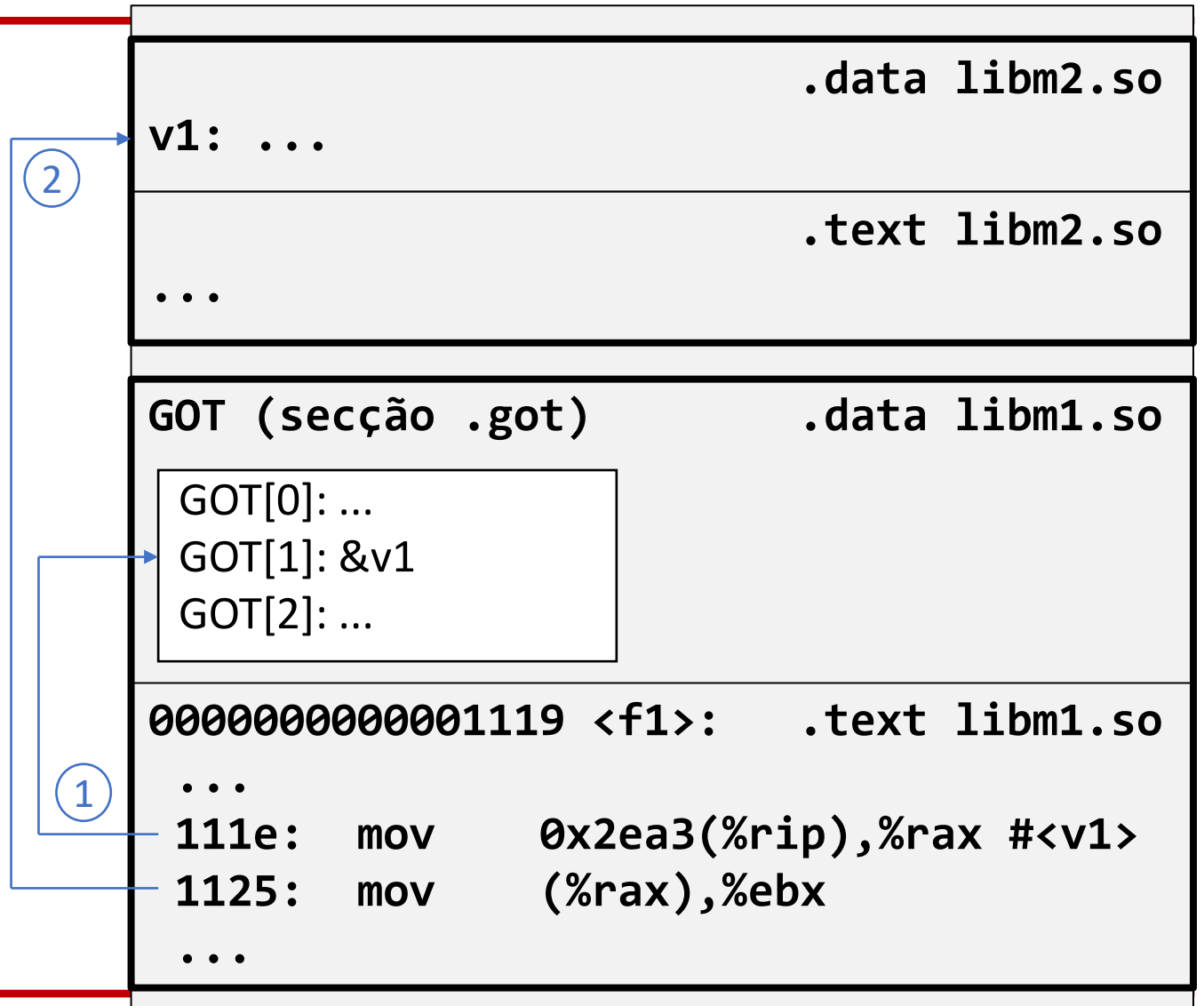

Acesso a variáveis com a opção **-fpic**

- Em tempo de compilação é criada uma tabela GOT (*Global Offset Table*) em cada ficheiro objeto onde sejam feitas referências a variáveis globais com definição externa
 - Cada entrada da tabela tem a dimensão de 8 bytes para suportar um endereço
- Em tempo de compilação é criada, igualmente, uma entrada de relocação por cada entrada GOT (na secção **.rel.text**)
- No carregamento, o *linker* dinâmico inicia a entrada GOT com o endereço absoluto da variável externa (no exemplo, **&v1**)



Acesso a variáveis com a opção **-fpic** (**v1**, no exemplo)

- Existem dois acessos a memória para aceder ao conteúdo de **v1**: (1) para ler o endereço de **v1** com base no endereço relativo a **RIP** (a distância **0x2ea3** é constante independentemente do endereço base onde fica mapeada a biblioteca **libm1** em qualquer aplicação); (2) endereçamento indireto a **v1**

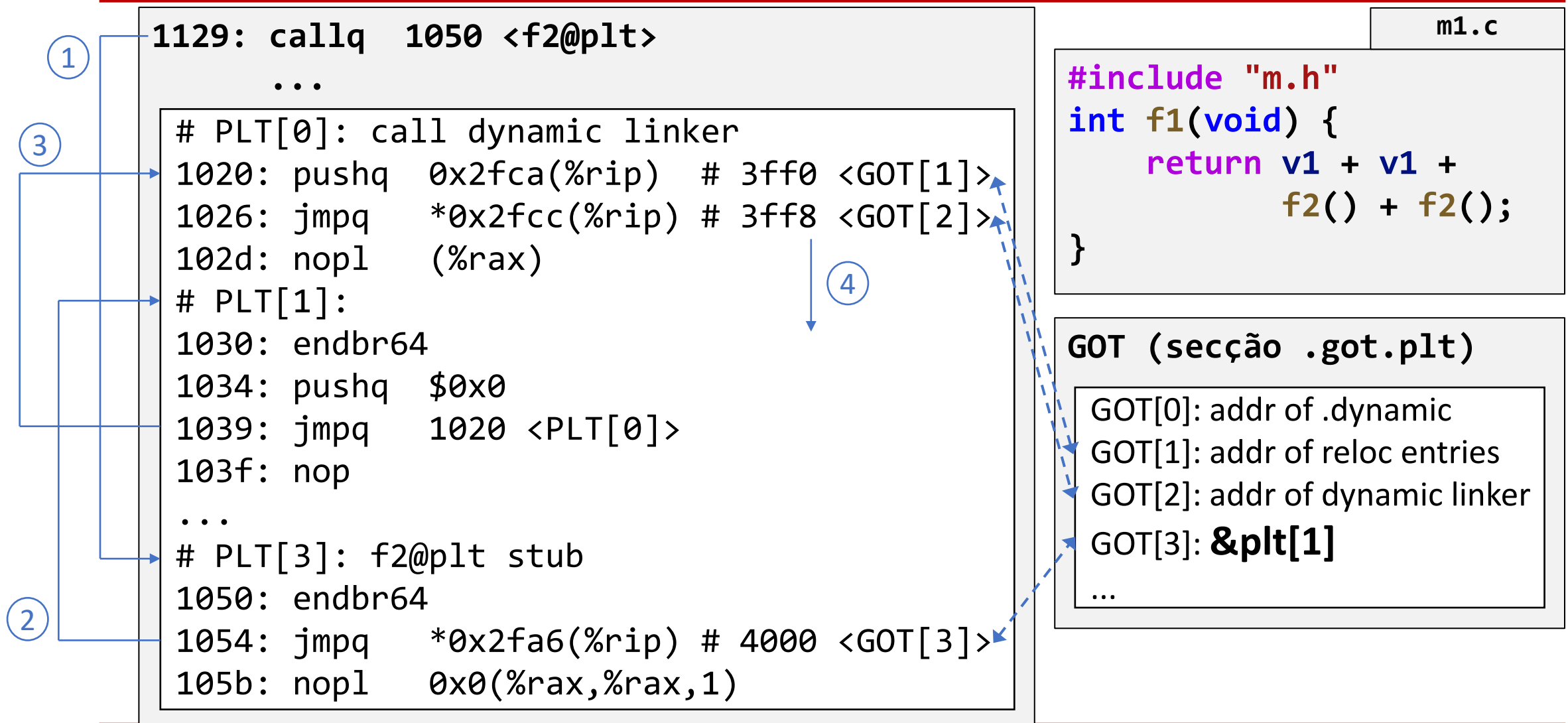


Chamadas a funções com a opção **-fpic**

- Podia ser usada a mesma estratégia que é usada para resolver referências a variáveis globais externas. Não é, porque:
 - Dado o número de funções que uma biblioteca pode exportar, tipicamente, muito superior ao número de variáveis globais externas, o tempo de inicialização da GOT com os endereços de todas as funções durante o processo de carregamento da biblioteca seria penalizante ainda que o código cliente pudesse usar apenas um subconjunto reduzido das funções.
- No âmbito da resolução de chamadas a funções, é introduzido o conceito de *lazy binding* que corresponde a resolver os endereços apenas e somente quando for necessário, ou seja, resolver a ligação apenas na primeira vez em que cada função é chamada
- Para esta solução são usadas duas estruturas de dados no ficheiro objeto onde são feitas referências às funções externas: GOT e PLT (*Procedure Linkage Table*)
 - A GOT está definida numa secção de dados (secção **.got.plt**)
 - A PLT está definida em secções de código (secções **.plt**, **.plt.got** e **.plt.sec**): cada entrada da tabela tem exatamente 16 bytes e corresponde, no código, a um *stub* para função
 - Para cada função definida externamente existe uma entrada em cada uma das duas tabelas: na tabela GOT, o endereço da função depois de resolvida a indefinição; na tabela PLT, o *stub* para a função

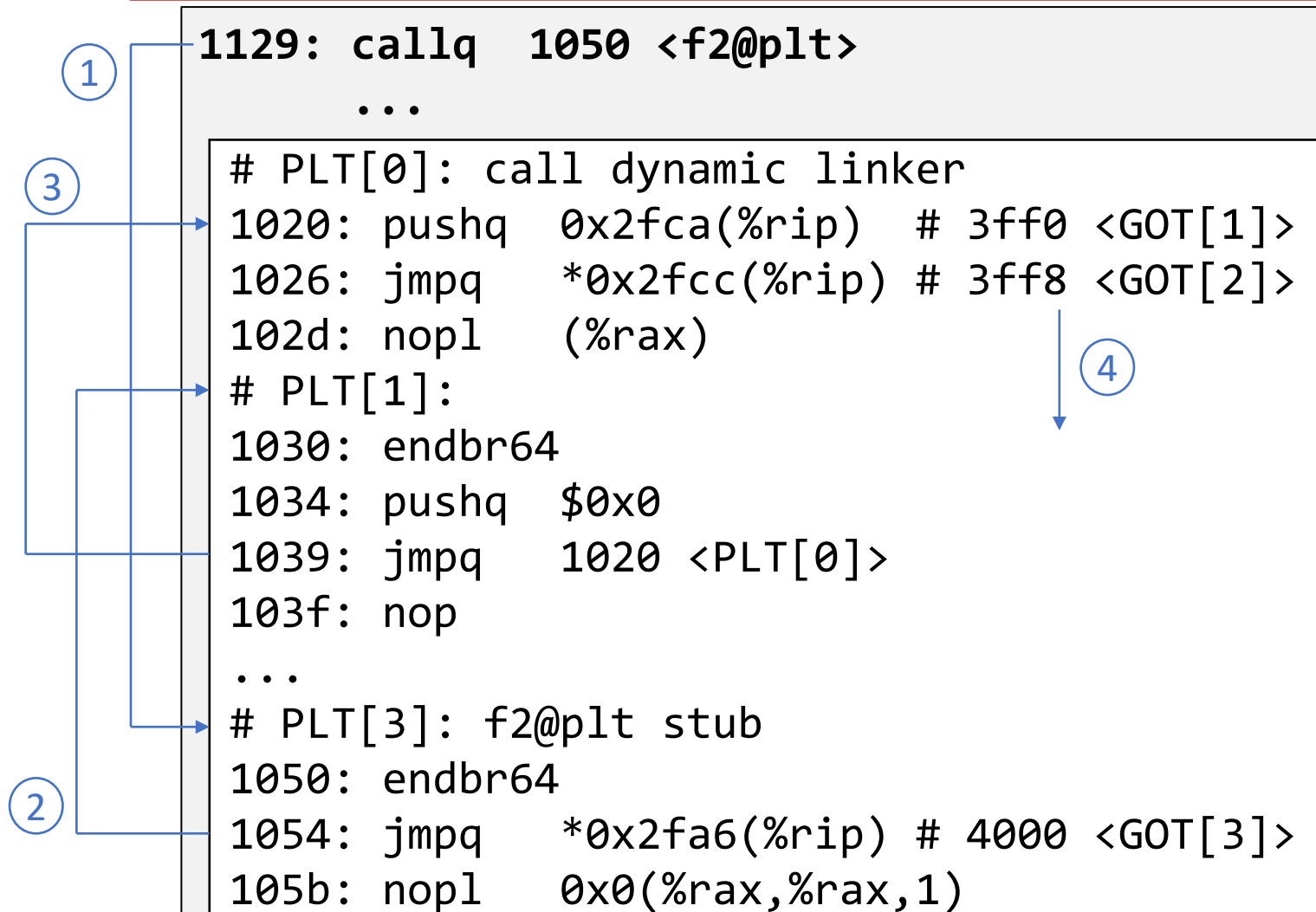
Processo para resolver chamadas a funções com a opção **-fpic**

Primeira chamada



Processo para resolver chamadas a funções com a opção **-fpic**

Primeira chamada



1. A chamada a `f2` provoca o salto para o *stub* associado a `f2` (`PLT[3]`)
2. Salta para o endereço presente na entrada GOT associada a `f2` (`GOT[3]`). Na primeira chamada contém o endereço de `PLT[1]`
3. Empilha identificador da função `f2` e salta para a `PLT[0]` onde volta a empilhar um argumento (entradas para relocação – `GOT[1]`) antes de executar o *linker* dinâmico (`GOT[2]`)
4. Baseado nos dois argumentos, o *linker* dinâmico determina o endereço da função `f2`, atualiza a entrada na GOT associada a `f2` (`GOT[3]`) com esse valor e passa a execução para `f2`

Processo para resolver chamadas a funções com a opção **-fpic**

Segunda chamada e restantes

①

```
1130: callq 1050 <f2@plt>
```

```
...
```

```
# PLT[0]: call dynamic linker
```

```
1020: pushq 0x2fe2(%rip) # 4008 <GOT[1]>
```

```
1026: jmpq *0x2fe3(%rip) # 4010 <GOT[2]>
```

```
102d: nopl (%rax)
```

```
# PLT[1]:
```

```
1030: endbr64
```

```
1034: pushq $0x0
```

```
1039: jmpq 1020 <PLT[0]>
```

```
103f: nop
```

```
...
```

```
# PLT[3]: f2@plt stub
```

```
1050: endbr64
```

```
1054: jmpq *0x2fbd(%rip) # 4018 <GOT[3]>
```

```
105b: nopl 0x0(%rax,%rax,1)
```

②

1. A chamada a **f2** provoca o salto para o *stub* associado a **f2** (PLT[3])
2. Desta feita, o salto indireto com base no conteúdo da GOT[3] provoca a transferência de controlo diretamente para **f2**

GOT (secção .got.plt)

```
GOT[0]: addr of .dynamic
```

```
GOT[1]: addr of reloc entries
```

```
GOT[2]: addr of dynamic linker
```

```
GOT[3]: &f2
```

```
...
```

Agenda

- Motivação, criação de uma biblioteca, utilização e processo de ligação
- Código PIC (*Position Independent Code*)
- Carregamento em tempo de execução

Carregamento em tempo de execução

- A biblioteca é carregada e ligada em tempo de execução do programa
- No código cliente não existe qualquer referência para símbolos (variáveis ou funções) definidos na biblioteca carregada em tempo de execução
- Permite a introdução de novas funcionalidades sem precisar gerar novamente a aplicação ou, inclusive, parar a sua execução

Interface programática para *linker* dinâmico

Necessário incluir ficheiro *header* **dlfcn.h** e ligar com o *linker* dinâmico **-ldl**

```
void *dlopen(const char *filename, int flags);
```

- Carrega biblioteca em memória (pode já estar em memória – incrementa contador)
 - Retorna ponteiro (na forma de *handle*) a usar noutras operações da API
 - Retorna **NULL** em caso de erro
 - O parâmetro **flags** pode valer **RTLD_NOW**, **RTLD_LAZY**, entre outros

```
void *dlsym(void *handle, const char *symbol);
```

- Liga o símbolo passado por argumento
 - Retorna ponteiro para variável ou função em caso de sucesso; retorna **NULL** se não for possível realizar a ligação

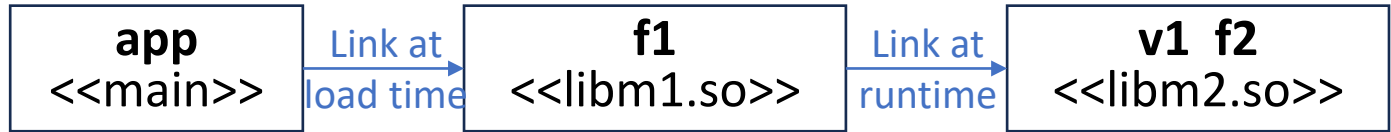
```
int dlclose(void *handle);
```

- Descarrega biblioteca partilhada se respetivo contador atinge o valor 0
 - Retorna 0 em caso de sucesso, -1 em caso de erro

```
char *dlerror(void);
```

- Retorna *string* com o erro mais recente ocorrido na execução das funções anteriores
 - Retorna **NULL** se não aconteceu qualquer erro desde o início ou desde a última chamada a **dlerror**

Exemplo (1 de 7)



app.c

```
#include <stdio.h>
#include "m1.h"

#define PRINT_STEP(s) \
    printf(s ". Press ENTER key to continue."); \
    getchar()

int main() {
    PRINT_STEP("1. Before libm2.so load");
    m1_init();
    PRINT_STEP("2. After libm2.so load");
    printf("f1() = %d\n", f1());
    m1_fini();
    PRINT_STEP("3. After libm2.so unload");
    return 0;
}
```

m1.h

```
#ifndef _M_H
#define _M_H

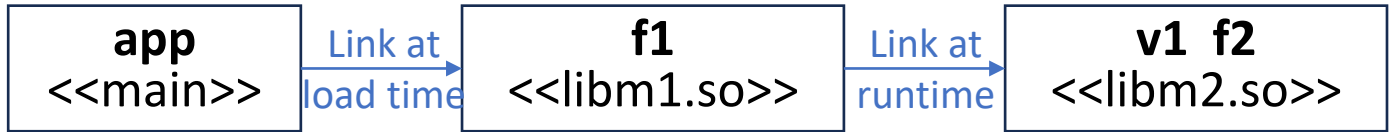
void m1_init();
void m1_fini();
int f1(void);

#endif/*_M_H*/
```

m2.c

```
int v1 = 10;
int f3(void) {
    return v1 * 10; }
int f2(void) {
    return v1 + f3();}
```

Exemplo (2 de 7)



m1.c

```
#include <dlfcn.h>
#include <stdio.h>
#include <stdlib.h>

#define EXIT_IF_NULL(h) \
    if ((h) == NULL) { \
        fprintf(stderr, "%s\n", \
            dlerror()); \
        exit(1); \
    }
#define EXIT_IF_NZERO(h) \
    if ((h) != 0) { \
        fprintf(stderr, "%s\n", \
            dlerror()); \
        exit(1); \
    }

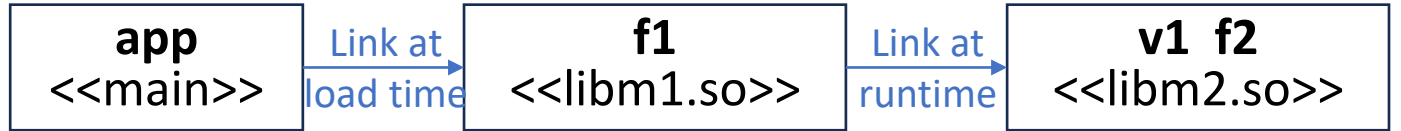
void * hm2;
```

m1.c

```
void m1_init() {
    EXIT_IF_NULL(hm2 = dlopen(
        "./libm2.so", RTLD_NOW));
}
void m1_fini() {
    EXIT_IF_NZERO(dlclose(hm2));
}

int f1(void) {
    int * pv1;
    EXIT_IF_NULL(
        pv1 = (int*)dlsym(hm2, "v1")
    );
    int (*pf2)(void);
    EXIT_IF_NULL(
        *(void**)&pf2 = dlsym(hm2, "f2")
    );
    return *pv1 + *pv1 + pf2() + pf2();
}
```

Exemplo (3 de 7)



Makefile

```
CFLAGS = -Wall -pedantic -c -Og
HEADER_FILES = m1.h
ALL = app

all: $(ALL)

# Create app
app: app.o libm1.so libm2.so
    gcc -o $@ $< ./libm1.so -ldl

# Create shared objects
libm1.so: m1.o
    gcc -shared -o $@ $^
libm2.so: m2.o
    gcc -shared -o $@ $^
```

Makefile

```
# Create relocatable objects
m1.o: m1.c
    gcc $(CFLAGS) -fpic $<
m2.o: m2.c
    gcc $(CFLAGS) -fpic $<
app.o: app.c m1.h
    gcc $(CFLAGS) -g $<

# Clear all generated files
clean:
    rm -f *.o $(ALL) *.so
```

Exemplo (4 de 7)

Observação do espaço de endereçamento da aplicação **app**

- Primeiro é preciso conhecer o identificador do processo **app**
- Utilitário **ps** para consultar o PID (*Process ID*)

```
$ ps -all
F S  UID    PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
...
0 S  1000 29897   551   0  80   0 -   638 -          pts/4      00:00:00 app
...
0 R  1000 29946 27395   0  80   0 -  2635 -          pts/2      00:00:00 ps
```

- Depois, consulta-se o ficheiro **maps** associado ao processo **app** que contém o estado atual do espaço de endereçamento da aplicação

```
$ sudo cat /proc/<pid>/maps
```

Exemplo (5 de 7)

Observação do espaço de endereçamento da aplicação **app** (1 de 3)

```
$ ./app
```

```
1. Before libm2.so load. Press ENTER key to continue.
```

```
$ sudo cat /proc/29897/maps
```

561f4b522000-561f4b523000	r--p	0...0	08:10	46883	/home/jpatri/test/lib_shared_exe/app
561f4c21e000-561f4c23f000	rw-p	0...0	00:00	0	[heap]
7fdf75a9f000-7fdf75ac1000	r--p	0...0	08:10	42911	/usr/lib/x86_64-linux-gnu/libc-2.31.so
7fdf75c91000-7fdf75c92000	r--p	0...0	08:10	43181	/usr/lib/x86_64-linux-gnu/libdl-2.31.so
7fdf75ca0000-7fdf75ca1000	r--p	0...0	08:10	46877	/home/jpatri/test/lib_shared_exe/libm1.so
7fdf75ca7000-7fdf75ca8000	r--p	0...0	08:10	38549	/usr/lib/x86_64-linux-gnu/ld-2.31.so
7ffd3a13c000-7ffd3a15e000	rw-p	0...0	00:00	0	[stack]

Exemplo (6 de 7)

Observação do espaço de endereçamento da aplicação **app** (2 de 3)

```
$ ./app
```

1. Before libm2.so load. Press ENTER key to continue.
2. After libm2.so load. Press ENTER key to continue.

```
$ sudo cat /proc/29897/maps
```

561f4b522000-561f4b523000	r--p	0...0	08:10	46883	/home/jpatri/test/lib_shared_exe/app
561f4c21e000-561f4c23f000	rw-p	0...0	00:00	0	[heap]
7fdf75a9f000-7fdf75ac1000	r--p	0...0	08:10	42911	/usr/lib/x86_64-linux-gnu/libc-2.31.so
7fdf75c91000-7fdf75c92000	r--p	0...0	08:10	43181	/usr/lib/x86_64-linux-gnu/libdl-2.31.so
7fdf75c9b000-7fdf75c9c000	r--p	0...0	08:10	46881	/home/jpatri/test/lib_shared_exe/libm2.so
7fdf75ca0000-7fdf75ca1000	r--p	0...0	08:10	46877	/home/jpatri/test/lib_shared_exe/libm1.so
7fdf75ca7000-7fdf75ca8000	r--p	0...0	08:10	38549	/usr/lib/x86_64-linux-gnu/ld-2.31.so
7ffd3a13c000-7ffd3a15e000	rw-p	0...0	00:00	0	[stack]

Exemplo (7 de 7)

Observação do espaço de endereçamento da aplicação **app** (3 de 3)

```
$ ./app
```

```
1. Before libm2.so load. Press ENTER key to continue.
```

```
2. After libm2.so load. Press ENTER key to continue.
```

```
f1() = 240
```

```
3. After libm2.so unload. Press ENTER key to continue.
```

```
$ sudo cat /proc/29897/maps
```

561f4b522000-561f4b523000	r--p	0...0	08:10	46883	/home/jpatri/test/lib_shared_exe/app
561f4c21e000-561f4c23f000	rw-p	0...0	00:00	0	[heap]
7fdf75a9f000-7fdf75ac1000	r--p	0...0	08:10	42911	/usr/lib/x86_64-linux-gnu/libc-2.31.so
7fdf75c91000-7fdf75c92000	r--p	0...0	08:10	43181	/usr/lib/x86_64-linux-gnu/libdl-2.31.so
7fdf75ca0000-7fdf75ca1000	r--p	0...0	08:10	46877	/home/jpatri/test/lib_shared_exe/libm1.so
7fdf75ca7000-7fdf75ca8000	r--p	0...0	08:10	38549	/usr/lib/x86_64-linux-gnu/ld-2.31.so
7ffd3a13c000-7ffd3a15e000	rw-p	0...0	00:00	0	[stack]