Aula 18 22.10.2018

Sumário:

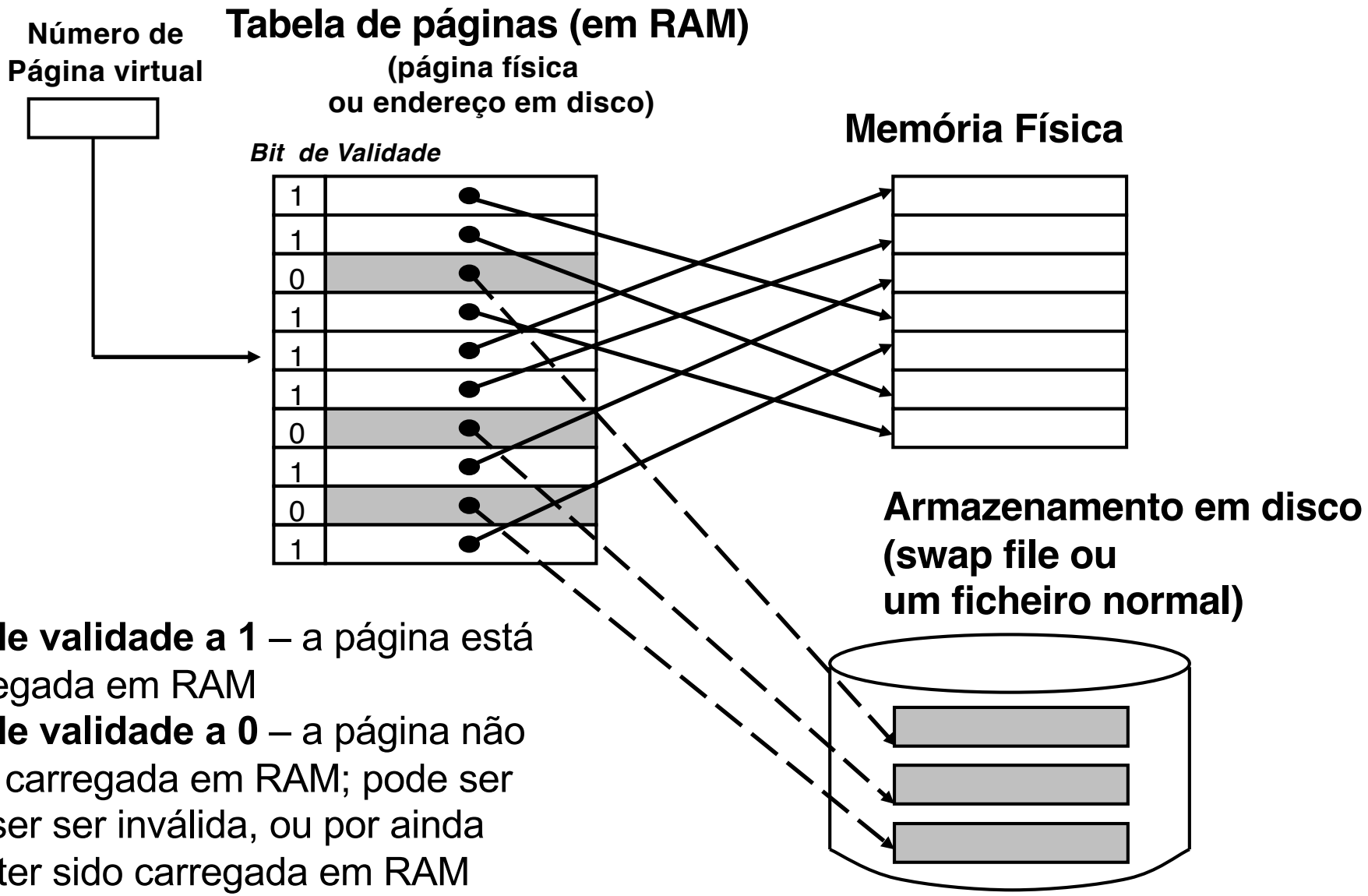
Partilha de páginas entre processos

Mapeamento de ficheiros na memória virtual

Ligação dinâmica

OSTEP caps 21, 22 e 23

Tabela de páginas



Bit de validade a 1 – a página está carregada em RAM

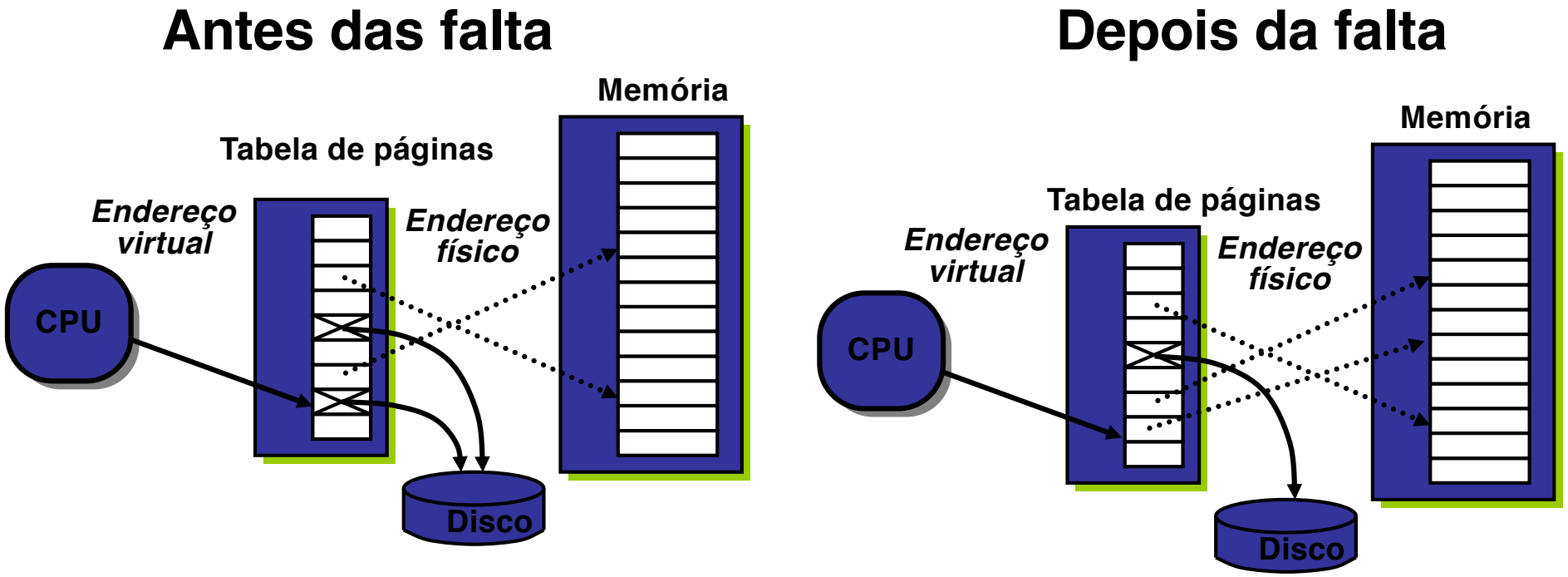
Bit de validade a 0 – a página não está carregada em RAM; pode ser por ser ser inválida, ou por ainda não ter sido carregada em RAM

Falta de página

– Tabela de páginas indica que o conteúdo referenciado pelo endereço virtual não está em RAM

– É invocado um procedimento de excepção para trazer os dados para memória

- o proceso corrente é suspenso, outros podem continuar
- o SO tem controlo completo sobre a localização das páginas



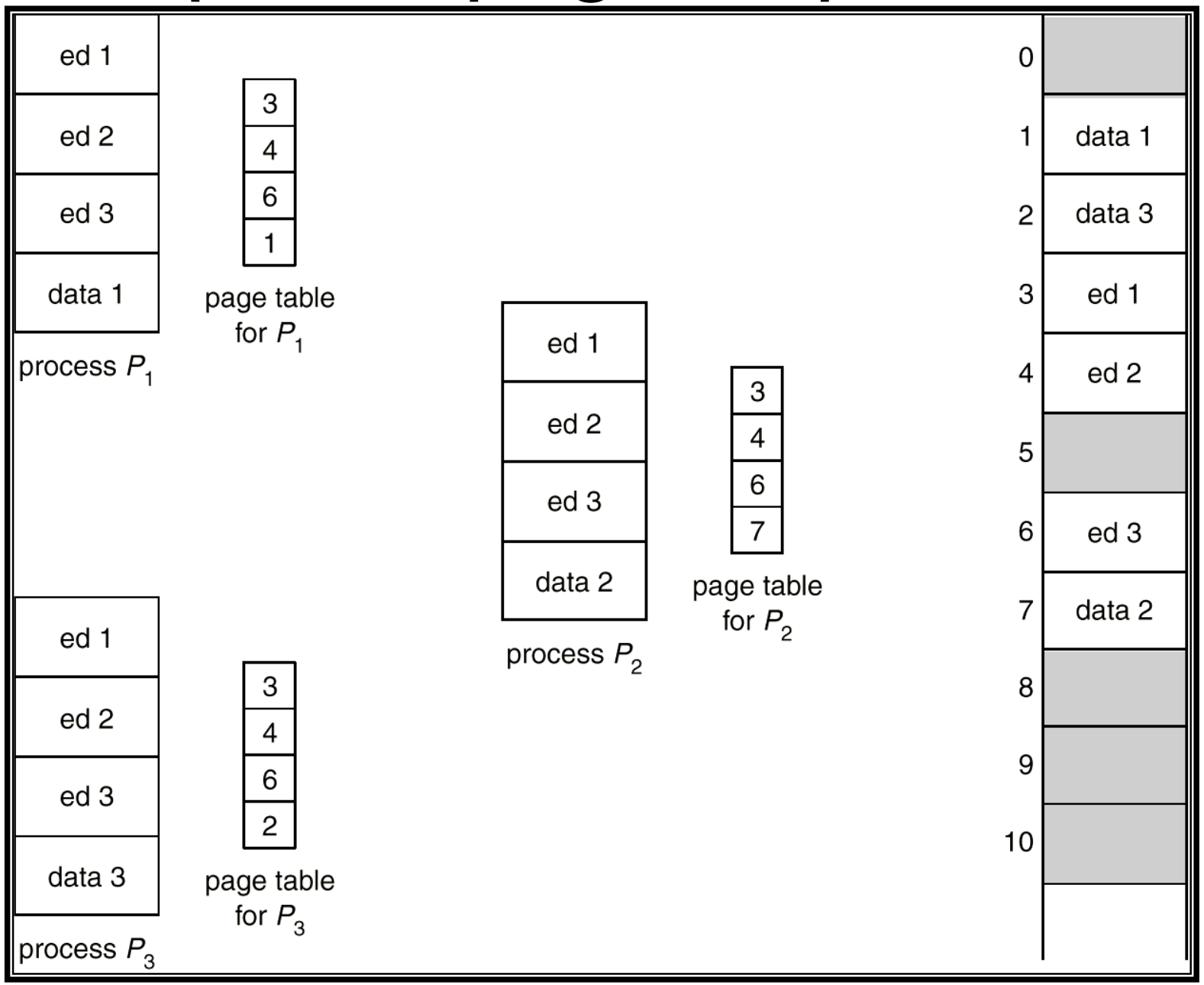
Partilha de páginas

- Crescimento do espaço de endereçamento
- “Memory-mapping” de ficheiros.
- Ligação dinâmica

Páginas partilhadas

- Código partilhado
  - Uma cópia do código (reentrante e “read-only”) partilhado entre vários processos até de utilizadores diferentes (i.e., editores de texto, compiladores, sistemas de janelas).
- Dados e pilha de execução privados
- Comunicação entre processos usando memória partilhada
  - Necessidade de estabelecer secções críticas

Exemplo de páginas partilhadas



Gestão das páginas dos processos

- A memória virtual e a paginação a pedido permitem maior eficiência na gestão da memória central:
  - Código e dados inicializados (não precisam de estar na zona de paginação – pode-se usar o ficheiro executável)
  - Dados não inicializados podem apontar para uma página inicializada com zeros, quando se escreve é atribuída uma nova página
  - Áreas de “stack” e “heap” vão recebendo mais páginas sempre que é preciso

Copy-on-Write

- Quando de um fork() a técnica Copy-on-Write permite que o pai e o filho *partilhem inicialmente* todas as páginas em memória (incluindo os dados e a pilha).
- Só quando um dos processos modifica uma página partilhada é que se procede à sua duplicação.
- O Copy-on-Write permite uma criação mais eficiente de processos uma vez que todas as páginas são iguais inicialmente.

“Memory mapping” de ficheiros

- A função mmap() cria uma nova região no espaço de endereçamento virtual do processo e associa-lhe um ficheiro.

```
void *mmap(void *start, int len, int prot, int
  flags, int fd, int offset)
```

Retorna um apontador para o início da região. O apontador é sempre para o início de uma página (múltiplo de 4096)

start – normalmente 0

len – tamanho da zona. Normalmente o comp. do ficheiro

prot – OU de PROT_NONE, PROT_READ, PROT_WRITE, …

flags – MAP_PRIVATE (privativo do processo), MAP_SHARED

fd – nº do canal associado ao ficheiro por um open() anterior

offset –deslocamento dentro do ficheiro (normalmente 0)

Ficheiros “Memory-Mapped”

- O uso de ficheiros “Memory-mapped” permite que as operações read/write sejam transformadas em acesso à memória. A cada bloco do ficheiro corresponde uma página em RAM.
- O ficheiro é trazido para memória baseado em paginação a pedido. O 1o acesso ao bloco traz a página para RAM; os acesso seguintes são acessos normais a memória.
- Isto simplifica o acesso aos ficheiros uma vez que o acesso aos ficheiros se pode fazer por operações tipo memcpy() ou bcopy() em vez das chamadas ao sistema **read()** e **write()**.
- Vários processos podem incluir o mesmo ficheiro no seu espaço de endereçamento, partilhando as mesmas páginas físicas.
- Usado intensivamente para suporte de “shared libraries”

Exemplo mmap(): cópia de ficheiros

```
#include <unistd.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

/* mmap.c - a program that uses
 mmap to copy itself to stdout*/
int main() {
  struct stat stat;
  int i, fd, size;
  char *bufp;

  /* open the file and get its
   size*/
  fd = open("./mmap.c",
   O_RDONLY);
  fstat(fd, &stat);
  size = stat.st_size;
```

```
/* map the file to a new VM
   area */
  bufp = mmap(0, size,
   PROT_READ,
     MAP_PRIVATE, fd, 0);

  /* write the VM area to
   stdout */
  write(1, bufp, size);
}
```

Imagem de um processo no Linux

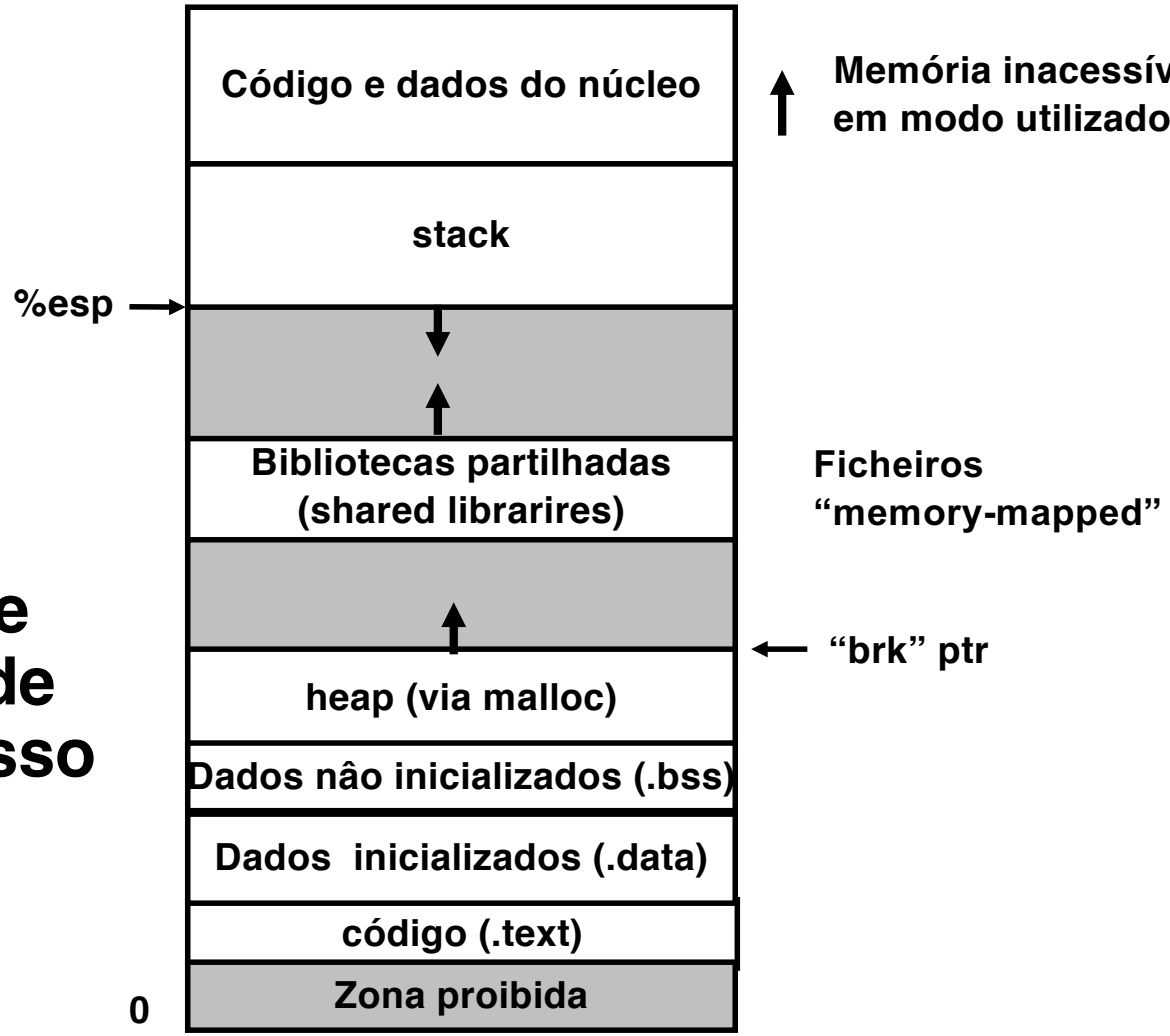


Imagem de memória de um processo Linux/x86

O trabalho prático da semana permite ter uma imagem exacta do mapa de memória do Linux para o kernel do Ubuntu 14.04; noutras versões é diferente

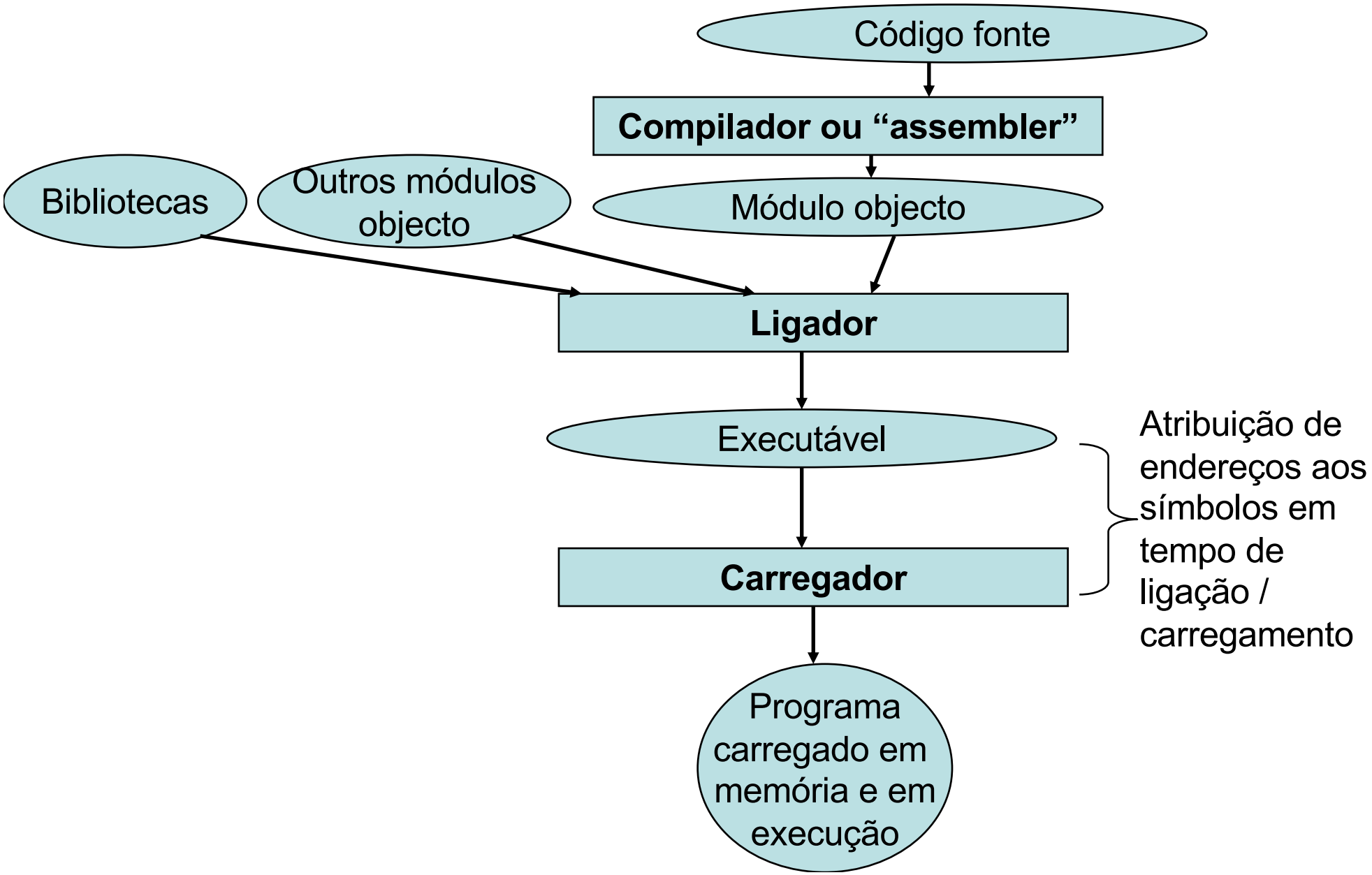
Vantagens do “Memory mapping” de ficheiros

- **Unifica o acesso aos dados de um programa**. Tradicionalmente o acesso aos dados em RAM ou em disco é feita de forma totalmente diferente. O uso de mmap() unifica o modo de acesso; o programador é libertado da tarefa de movimentar dados entre o disco e RAM
- **Exige menos chamadas ao sistema**.
- **As transferências são mais rápidas**: O kernel faz operações de entrada saída directamente entre a memória do utilizador e o disco sem passar pelos “buffers” do sistema.

Ligação estática e dinâmica de executáveis

- **Ligação Estática**. Todo o código é inserido no ficheiro executável. A ligação é feita completamente antes da execução
- **Ligação dinâmica**: O código das bibliotecas não é inserido no ficheiro executável. O estabelecimento dos endereços virtuais das funções das bibliotecas é feita em tempo de execução

Ligaão estática



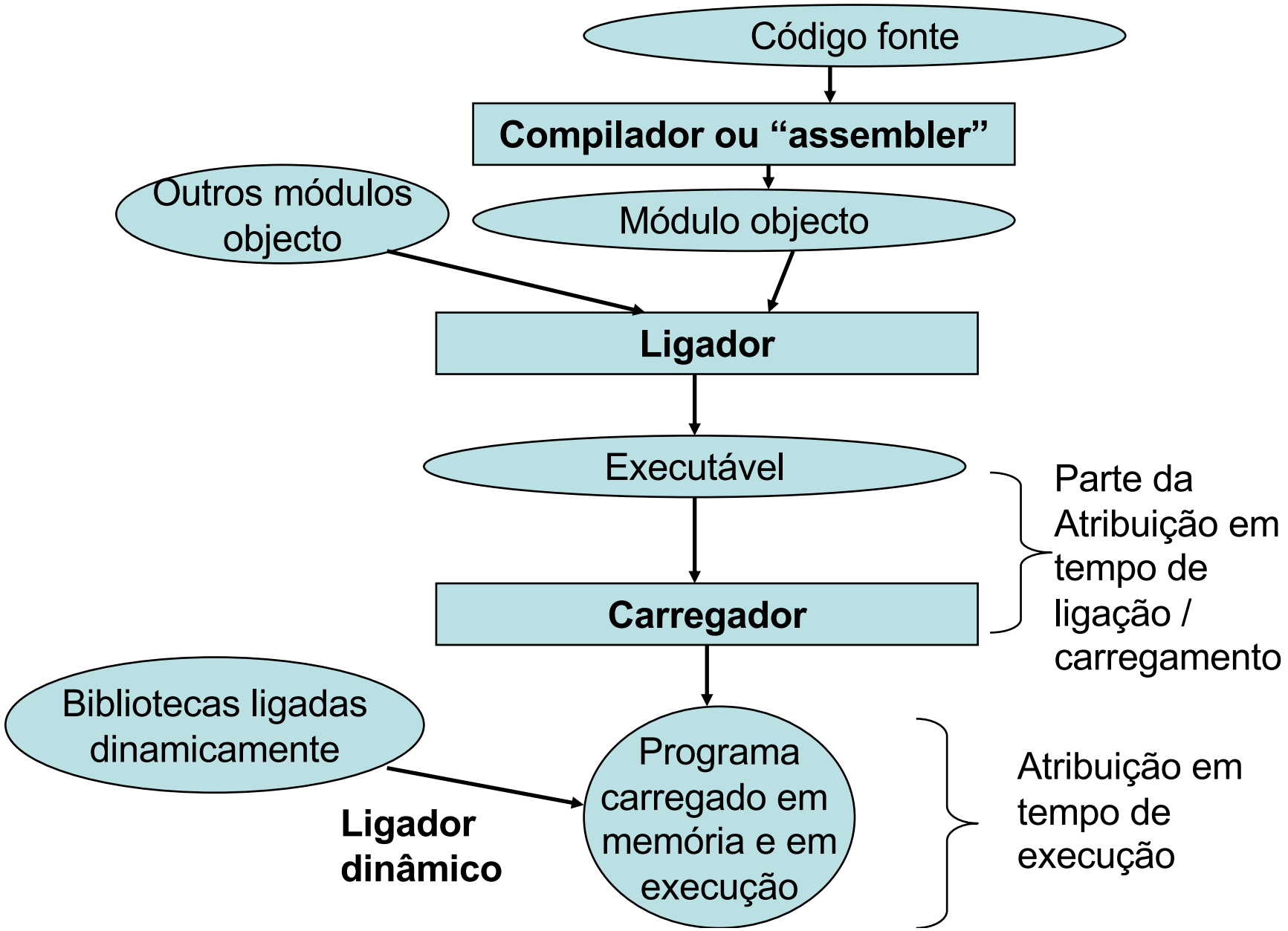
Ligação estática de bibliotecas

- A ligção estáticande bibliotecas tem desvantagens:
  - Potencialmente há muito código comum nos ficheiros executáveis guardados no sistema de ficheiros.
    - e.g., cada programa em C necessita da standard C library
  - Potencialmente há muito código duplicado no espaço de endereçamento dos vários processos.
  - Pequenas modificações nas bibliotecas de sistema obrigam a voltar a ligar todas as aplicações
- Solução:
  - *Bilbiotecas partilhadas shared libraries* (dynamic link libraries, DLLs)

Bibliotecas partilhadas (Shared libraries)

- Os membros de uma *shared library* são carregados dinamicamente para memória e ligados à aplicação em tempo de execução.
  - A ligação dinâmica pode ser feita quando o processo é carregado em memória e executado.
    - É inserido no executável uma função que resolve os endereços en tempo de execução
  - As rotinas que compõem a “shared library” podem ser partilhdas por múltiplos processos.

Ligação dinâmica



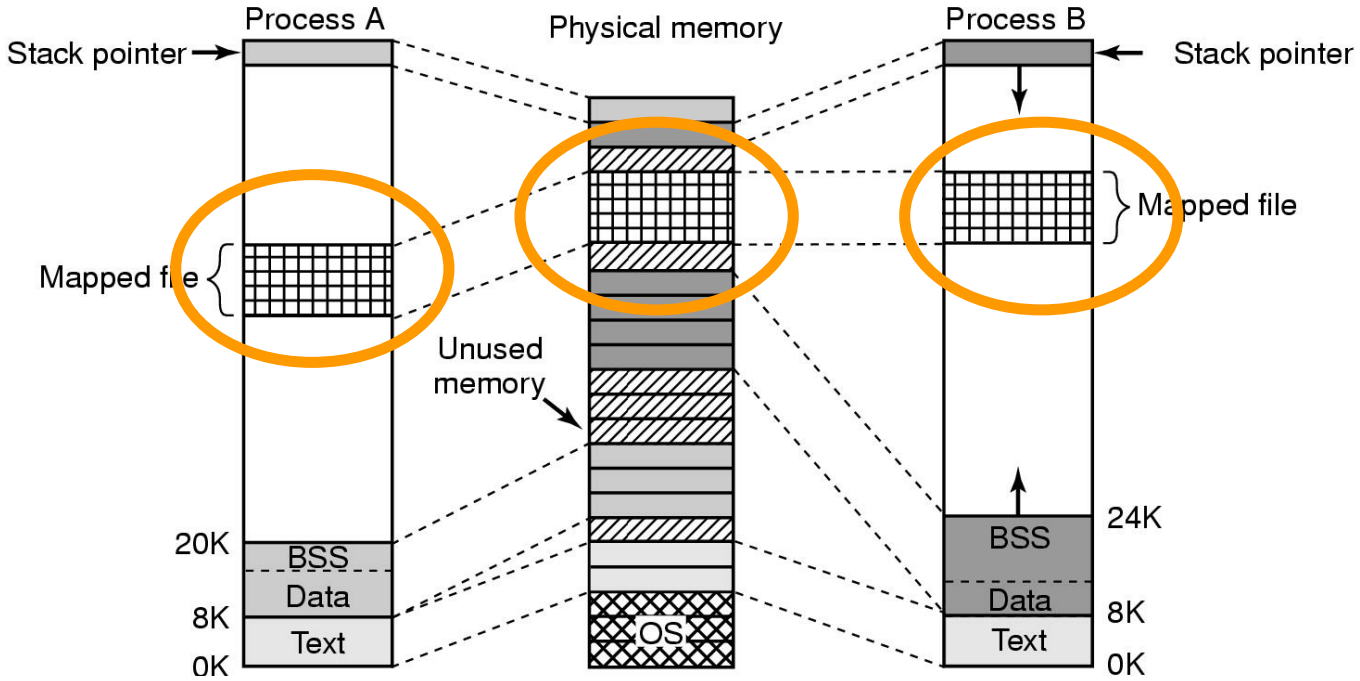
Ligação dinâmica(dynamic linking)

- Ligação é adiada até ao momento da execução.
- Uma pequena rotina (*stub*) é usada para localizar uma rotina que já pode estar carregada em memória. O “stub” substitui-se a si próprio com o endereço da rotina.
- O SO é chamado para reprogramar a MMU de forma a incorporar a referida rotina no espaço de endereçamento do process.
- A ligação dinâmica é particularmente útil para bibliotecas (shared libraries) – DLLs no windows.

Bibliotecas partilhadas (Shared libraries)

- Os membros de uma *shared library* são carregados dinamicamente para memória e ligados à aplicação em tempo de execução.
  - A ligação dinâmica pode ser feita quando o processo é carregado em memória e executado.
    - no Linux é a solução habitual, tratado automaticamente pela biblioteca ld-linux.so.
  - As rotinas que compõem a “shared library” podem ser partilhdas por múltiplos processos.

Ficheiros que contêm o código são "memory-mapped" antes da execução começar



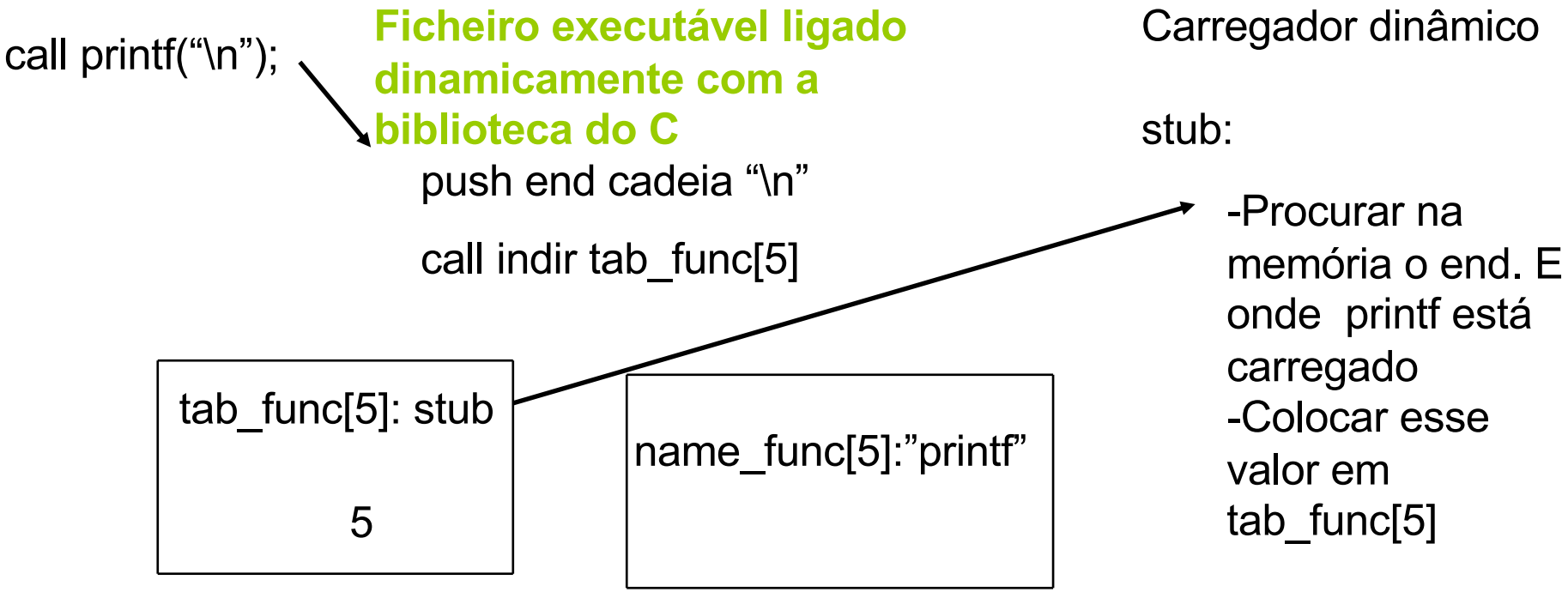
Os ficheiros que contêm o código das bibliotecas vão sendo trazidos para a RAM página a página à medida que as rotinas vão sendo referenciadas

Acção do ligador dinâmico

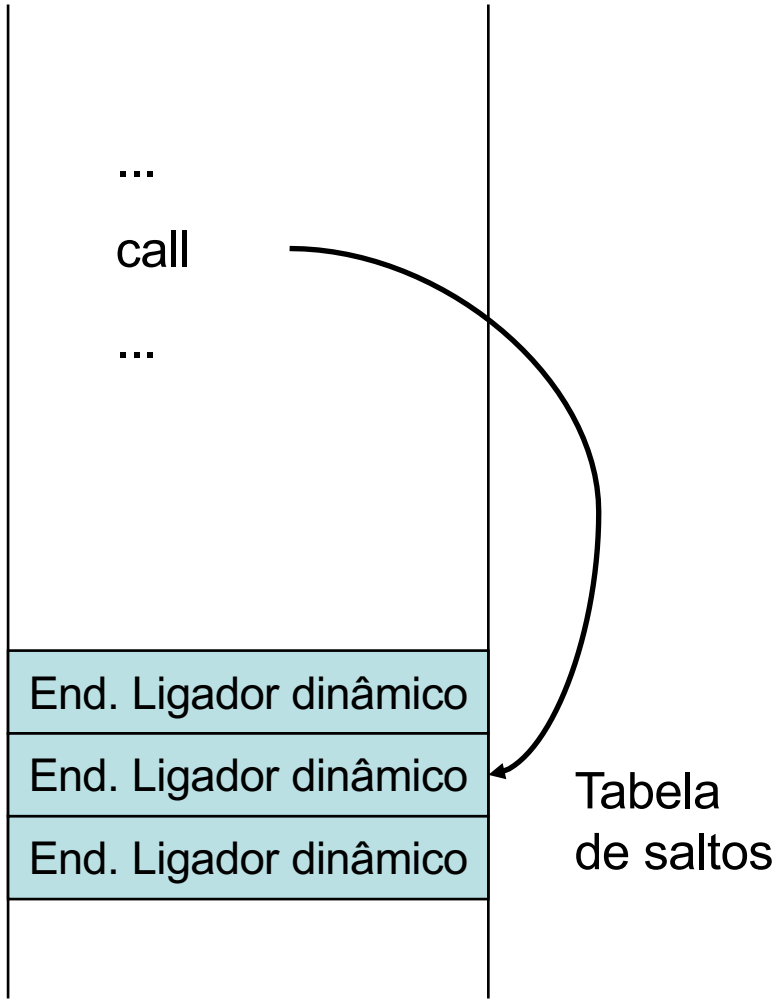
- No ficheiro executável existe uma tabela de saltos em que inicialmente todas as entradas apontam para um endereço onde está um carregador dinâmico (stub)

- Quando a função é chamada pela primeira vez o código que está em stub procura o endereço E do símbolo na memória e preenche a entrada na tabela de saltos

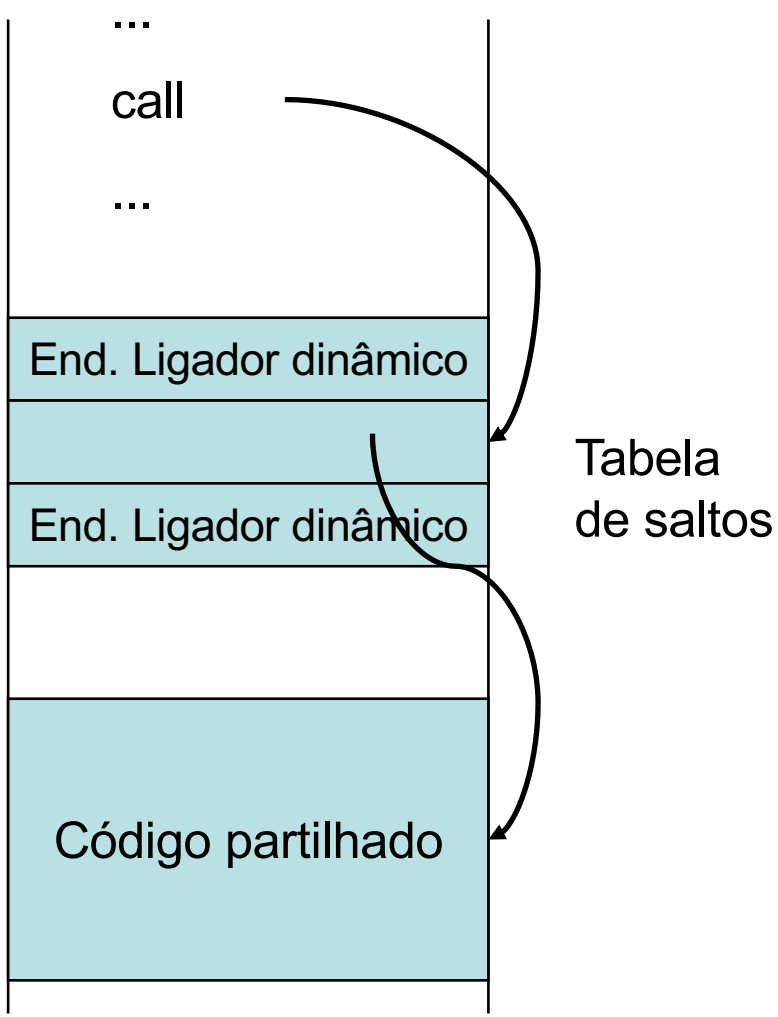
- As próximas referências ao símbolo resolvem directamente



Ligação dinâmica



Antes da 1ª invocação



Depois da 1ª invocação