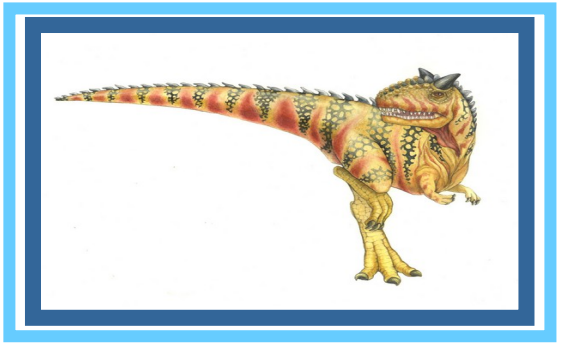
Introdução ao Gerenciamento de Memória



Memória desejada por programadores

- **Grande**
- **Rápida**
- **Não Volátil**
- **Baixo Custo**

A tecnologia atual não comporta tais memórias.

Hierarquia da Memória

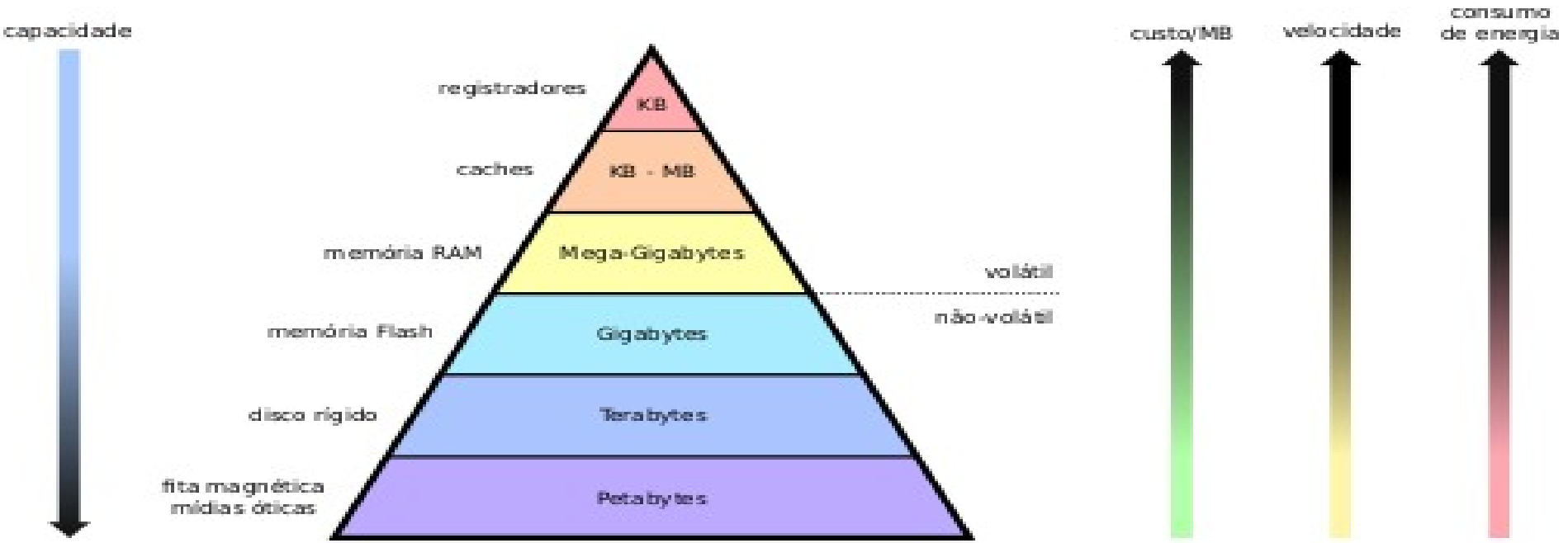


Figura 14.1: Hierarquia de memória.

Tempo de acesso x Taxa de transferência

Meio	Tempo de acesso	Taxa de transferência
Cache L2	1 ns	1 GB/s (1 ns/byte)
Memória RAM	60 ns	1 GB/s (1 ns/byte)
Memória *flash* (NAND)	2 ms	10 MB/s (100 ns/byte)
Disco rígido SATA	5 ms (tempo para o ajuste da cabeça de leitura e a rotação do disco até o setor desejado)	80 MB/s (12 ns/byte)
DVD-ROM	de 100 ms a vários minutos (caso a gaveta do leitor esteja aberta ou o disco não esteja no leitor)	10 MB/s (100 ns/byte)

Tabela 14.1: Tempos de acesso e taxas de transferência típicas [Patterson and Henessy, 2005].

Atribuição de endereços

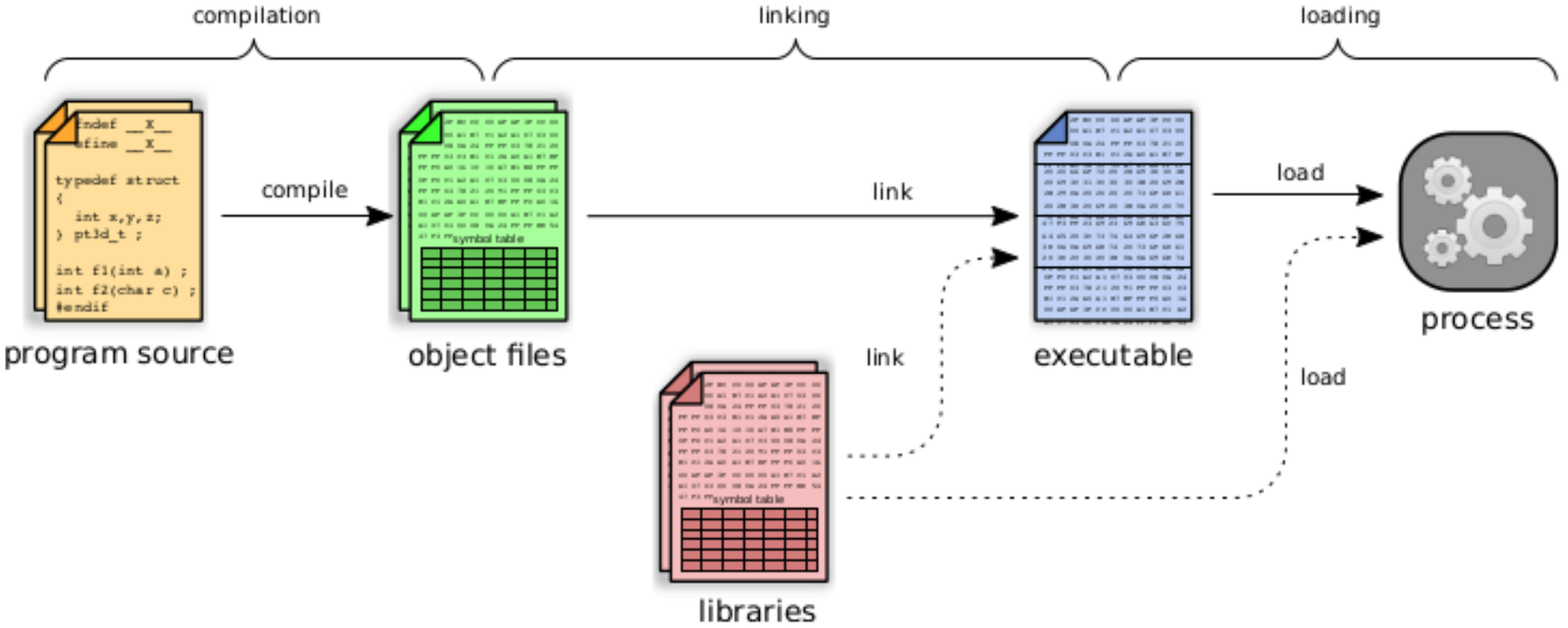
Código Fonte

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

int soma = 0 ;

int main ()
{
   int i ;

   for (i=0; i< 10; i++)
   {
      soma += i ;
      printf ("i vale %d e soma vale %d\n", i, soma) ;
   }
   exit(0) ;
}
```

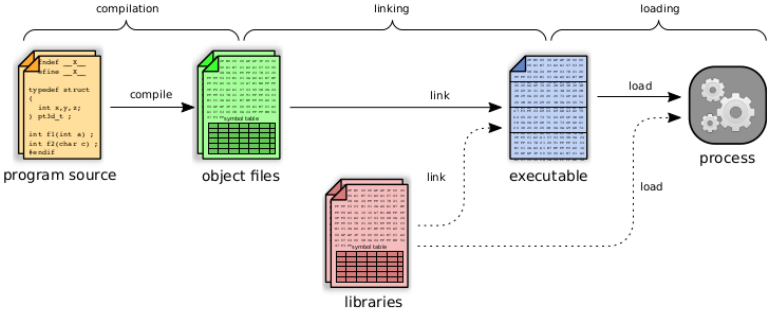
Código Compilado

```
0000000000000000 <main>:
   0:   55                          push   %rbp
   1:   48 89 e5                    mov    %rsp,%rbp
   4:   48 83 ec 10                 sub    $0x10,%rsp
   8:   c7 45 fc 00 00 00 00        movl   $0x0,-0x4(%rbp)
   f:   eb 2f                       jmp    40 <main+0x40>
  11:   8b 15 00 00 00 00           mov    0x0(%rip),%edx
  17:   8b 45 fc                    mov    -0x4(%rbp),%eax
  1a:   01 d0                       add    %edx,%eax
  1c:   89 05 00 00 00 00           mov    %eax,0x0(%rip)
  22:   8b 15 00 00 00 00           mov    0x0(%rip),%edx
  28:   8b 45 fc                    mov    -0x4(%rbp),%eax
  2b:   89 c6                       mov    %eax,%esi
  2d:   bf 00 00 00 00              mov    $0x0,%edi
  32:   b8 00 00 00 00              mov    $0x0,%eax
  37:   e8 00 00 00 00              callq  3c <main+0x3c>
  3c:   83 45 fc 01                 addl   $0x1,-0x4(%rbp)
  40:   83 7d fc 04                 cmpl   $0x4,-0x4(%rbp)
  44:   7e cb                       jle    11 <main+0x11>
  46:   bf 00 00 00 00              mov    $0x0,%edi
  4b:   e8 00 00 00 00              callq  50 <main+0x50>
```

Momento da tradução de endereços



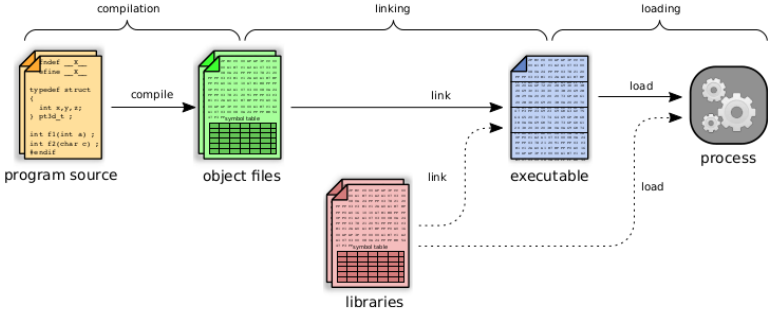
Momento da tradução de endereços



Durante a edição: o programador escolhe o endereço de cada uma das variáveis e do código do programa na memória. Esta abordagem normalmente só é usada na programação de sistemas embarcados simples, programados diretamente em *Assembly*.

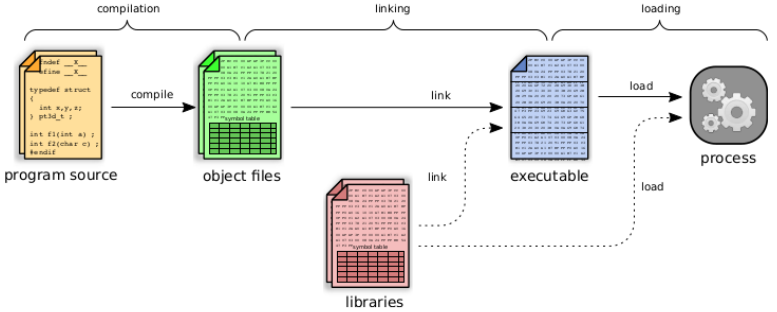
Durante a compilação: ao traduzir o código-fonte, o compilador escolhe as posições das variáveis na memória. Para isso, todos os códigos-fontes necessários ao programa devem ser conhecidos no momento da compilação, para evitar conflitos de endereços entre variáveis em diferentes arquivos ou bibliotecas. Essa restrição impede o uso de bibliotecas precompiladas. Esta abordagem era usada em programas executáveis com extensão `.COM` do MS-DOS e Windows.

Momento da tradução de endereços



Durante a ligação: na fase de compilação, o compilador traduz o código fonte em código binário, mas não define os endereços das variáveis e funções, gerando como saída um *arquivo objeto* (*object file*)[2], que contém o código binário e uma *tabela de símbolos* descrevendo as variáveis e funções usadas, seus tipos, onde estão definidas e onde são usadas. A seguir, o ligador (*linker*) pega os arquivos objetos com suas tabelas de símbolos, define os endereços de memória dos símbolos e gera o programa executável [Levine, 2000].

Momento da tradução de endereços



Durante a carga: também é possível definir os endereços de variáveis e de funções durante a carga do código em memória para o lançamento de um novo processo. Nesse caso, um *carregador* (*loader*) é responsável por carregar o código do processo na memória e definir os endereços de memória que devem ser utilizados. O carregador pode ser parte do núcleo do sistema operacional ou uma biblioteca ligada ao executável, ou ambos. Esse mecanismo normalmente é usado na carga de bibliotecas dinâmicas (DLL - *Dynamic Linking Libraries*).

Durante a execução: os endereços emitidos pelo processador durante a execução do processo são analisados e convertidos nos endereços efetivos a serem acessados na memória real. Por exigir a análise e a conversão de cada endereço gerado pelo processador, este método só é viável com o auxílio do hardware.