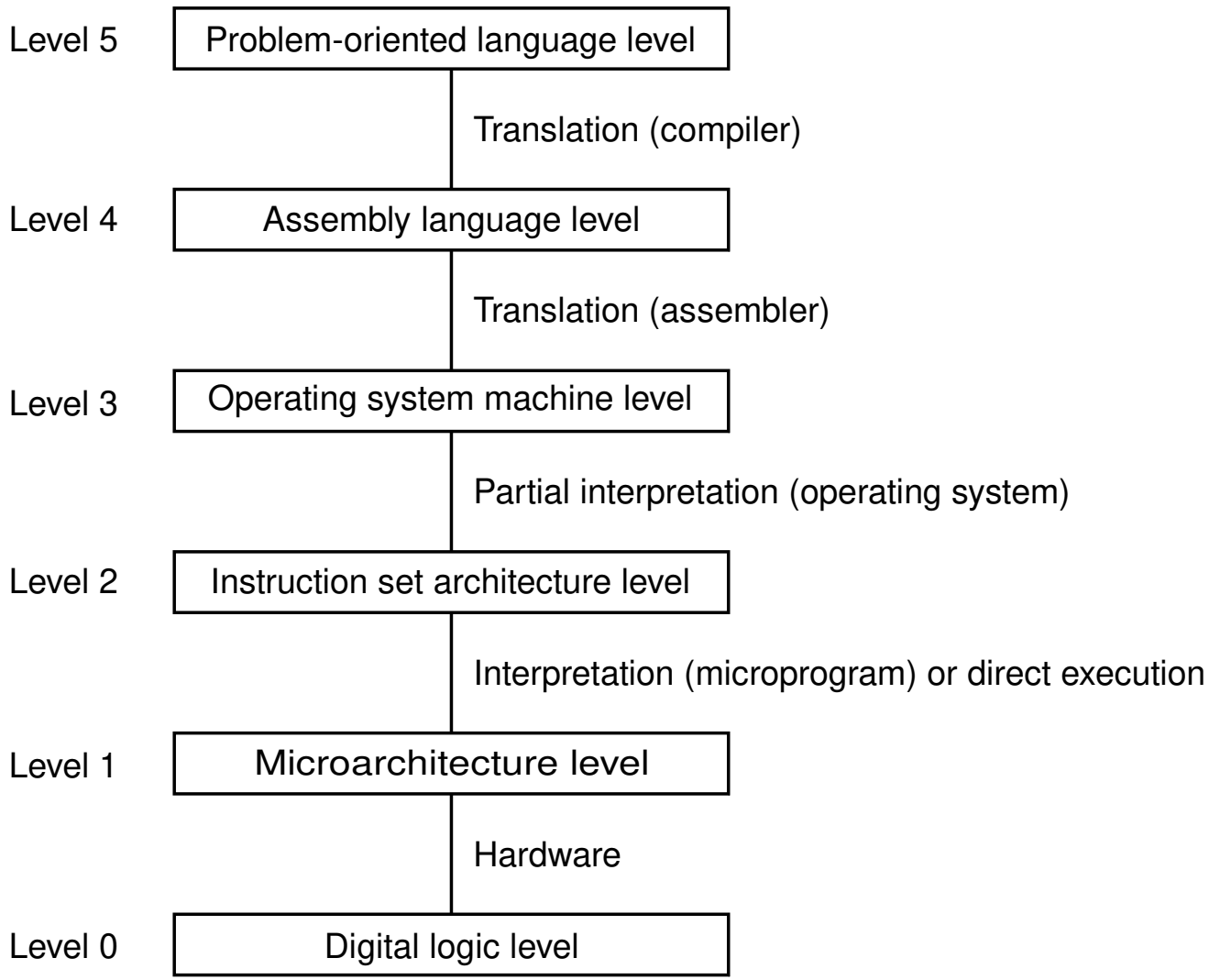
Revisão: Sistemas Operacionais

Jorge Correia

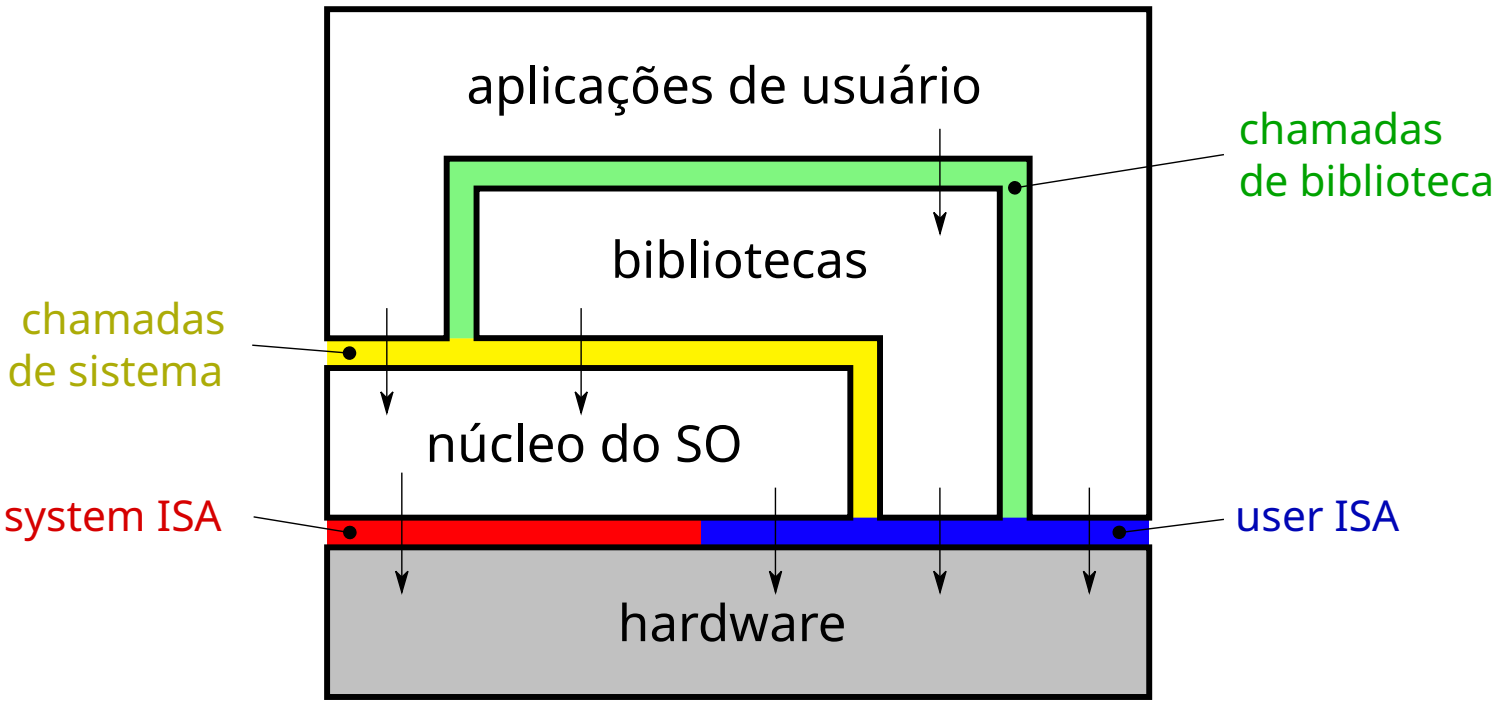
Segurança Computacional
Universidade Federal do Paraná

2024

Vamos começar do começo...



Vamos começar do começo...



Vamos começar do começo...

Quando compilamos um código em C, quais interfaces são utilizadas?

Vamos começar do começo...

Quando compilamos um código em C, quais interfaces são utilizadas?
Depende do alvo de compilação:

Vamos começar do começo...

Quando compilamos um código em C, quais interfaces são utilizadas?
Depende do alvo de compilação:

- Biblioteca / Sistema Operacional / Processador
- Sistema Operacional / Processador
- Processador

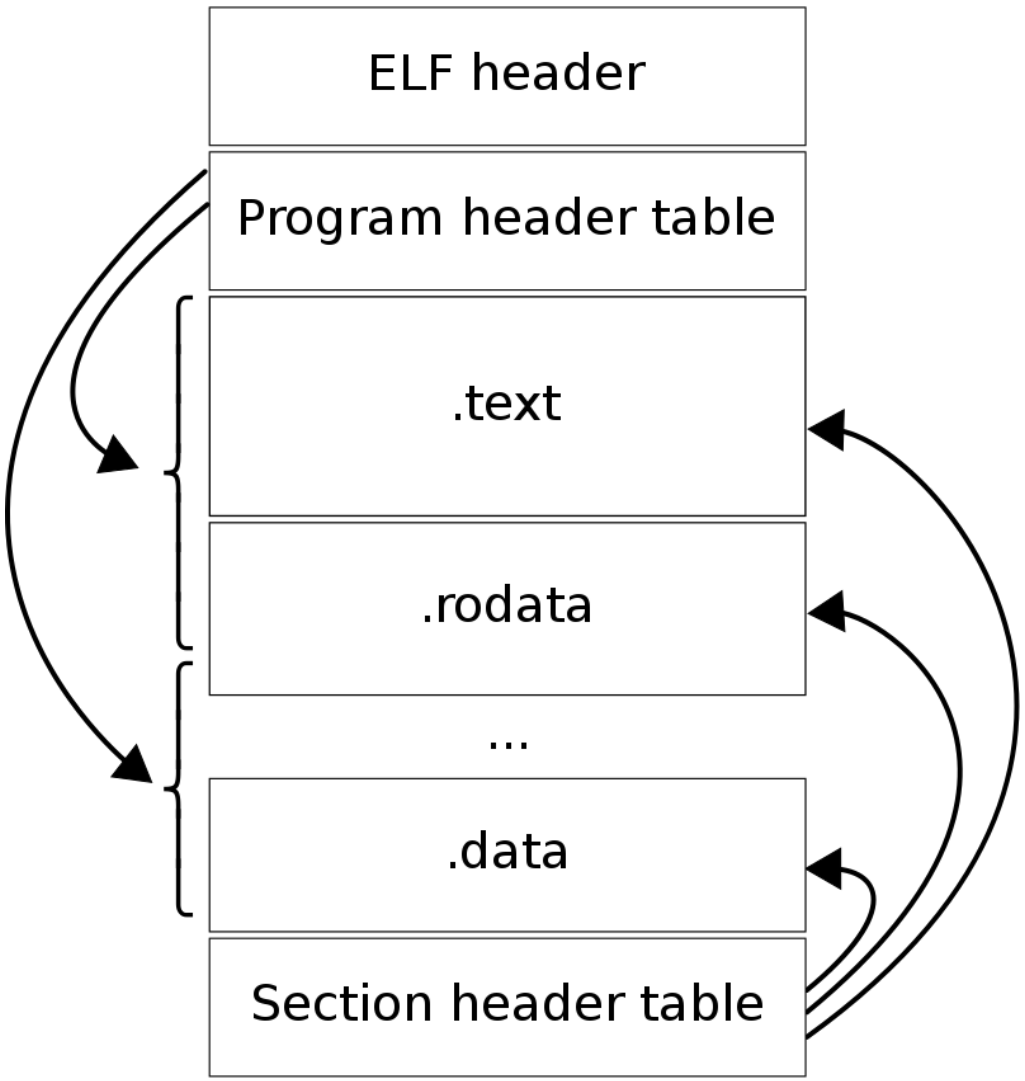
Vamos começar do começo...

Quando compilamos um código em C, quais interfaces são utilizadas?
Depende do alvo de compilação:

- Biblioteca / Sistema Operacional / Processador
- Sistema Operacional / Processador
- Processador
- Cross-compiler
- https://wiki.osdev.org/GCC_Cross-Compiler

E os formatos de arquivos executáveis (ELF / PE)?

E os formatos de arquivos executáveis (ELF / PE)?



Resumindo

- Existem vários níveis de linguagem
  - ISA (user e kernel)
  - chamadas de bibliotecas
  - syscalls
- Os arquivos executáveis não contém somente códigos assembly
  - informações de carregamento e execução do programa

E quando executamos um arquivo executável?

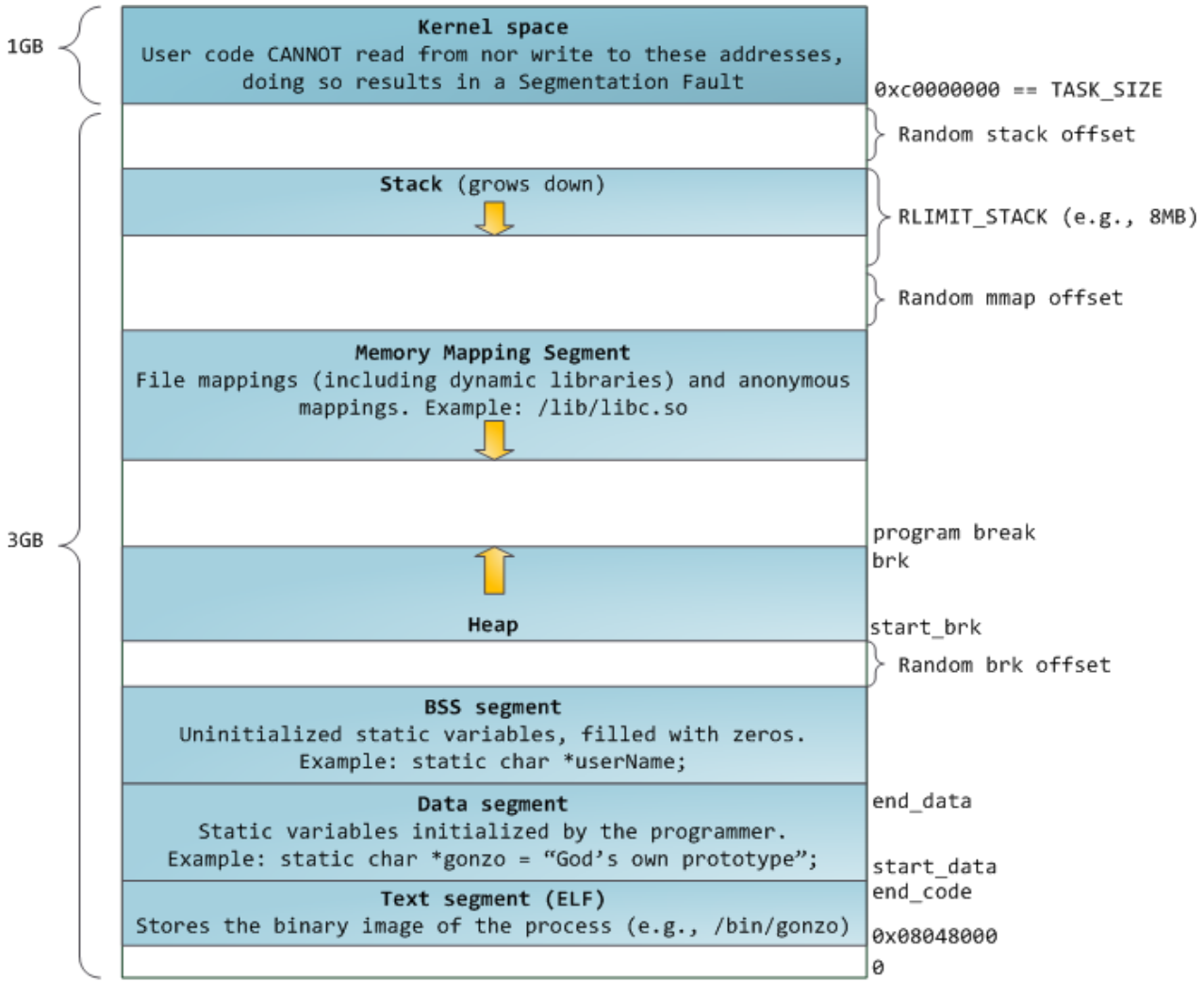
O programa se torna um processo!

E quando executamos um arquivo executável?

O programa se torna um processo!

- Todo processo possui um espaço de endereçamento que possui uma estrutura bem definida.

Espaço de endereçamento de um processo (32 bits)



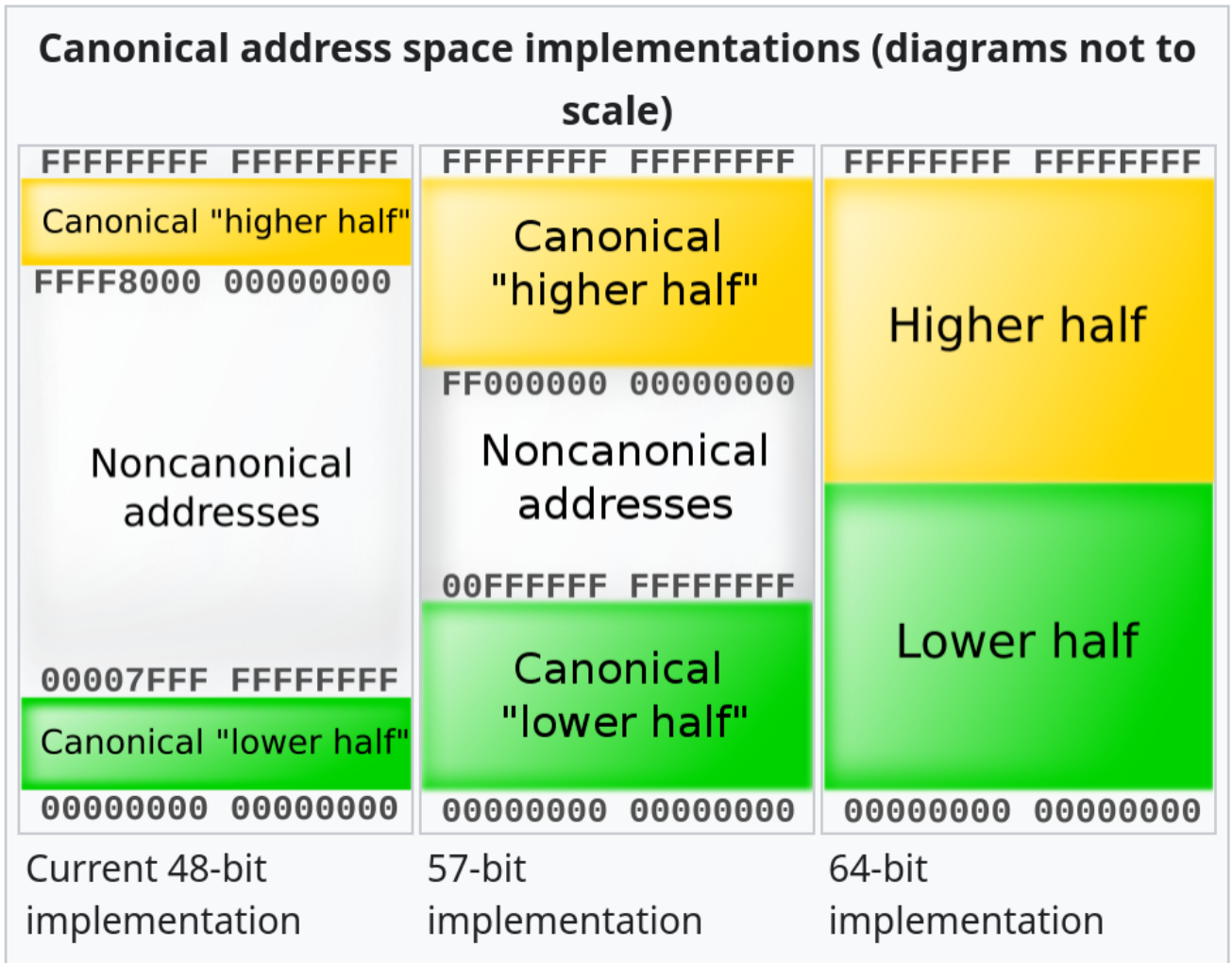
E o espaço de endereçamento de um processo em uma arquitetura 64 bits?

- Os processadores não usam 64 bits de endereçamento (16EiB)
- Seria aproximadamente 4 bilhões de vezes maior do que o espaço de endereçamento de 32 bits

E o espaço de endereçamento de um processo em uma arquitetura 64 bits?

- Os processadores não usam 64 bits de endereçamento (16EiB)
- Seria aproximadamente 4 bilhões de vezes maior do que o espaço de endereçamento de 32 bits
- Complexidade da implementação de um gerenciamento de memória
- Custo na tradução de endereços

E o espaço de endereçamento de um processo em uma arquitetura 64 bits?



Espaço de endereçamento na prática

- /proc/<process>/maps

Espaço de endereçamento na prática

- /proc/<process>/maps
- E se eu tentar ler um espaço de memória não mapeado?

Espaço de endereçamento na prática

- /proc/<process>/maps
- E se eu tentar ler um espaço de memória não mapeado?
- E se eu tentar escrever em um espaço de memória sem permissão?

E o espaço de endereçamento de um processo em uma arquitetura 64 bits?

Existem algumas arquiteturas que utilizam os bits de alta ordem para prover auxílios de segurança:

- ARM Memory Tagging Extension (MTE)

E o espaço de endereçamento de um processo em uma arquitetura 64 bits?

Normalmente, a metade baixa é usada para o espaço de usuário e a metade alta é usada para o kernel

- https://www.kernel.org/doc/html/latest/arch/x86/x86_64/mm.ht

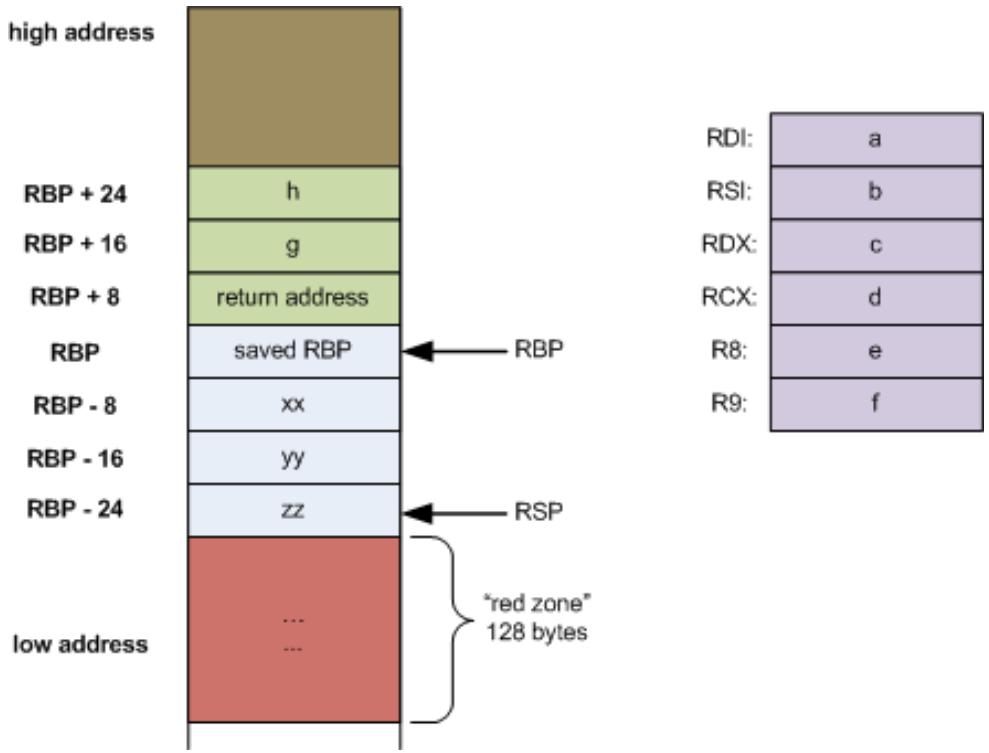
Stack

Espaço de memória para armazenar informações temporárias

- Endereços de retorno de funções
- Parâmetros de funções
- Variáveis locais

Registro de ativação

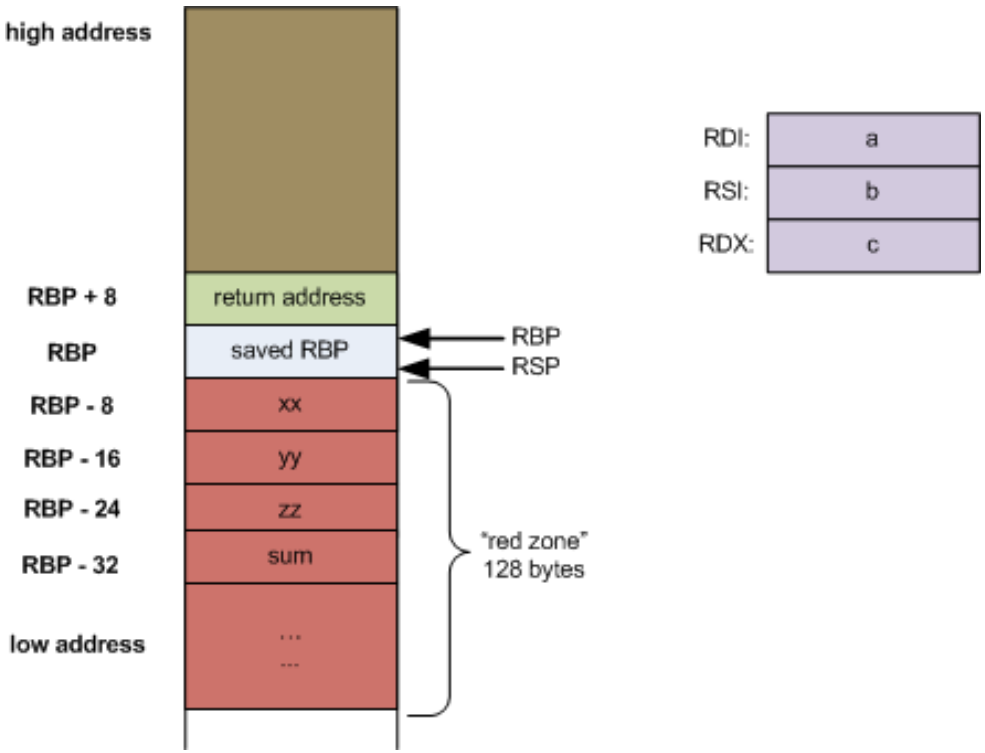
```
long myfunc(long a, long b, long c, long d,
            long e, long f, long g, long h)
{
    long xx = a * b * c * d * e * f * g * h;
    long yy = a + b + c + d + e + f + g + h;
    long zz = utilfunc(xx, yy, xx % yy);
    return zz + 20;
}
```



Registro de ativação

```
long utilfunc(long a, long b, long c)
{
    long xx = a + 2;
    long yy = b + 3;
    long zz = c + 4;
    long sum = xx + yy + zz;

    return xx * yy * zz + sum;
}
```



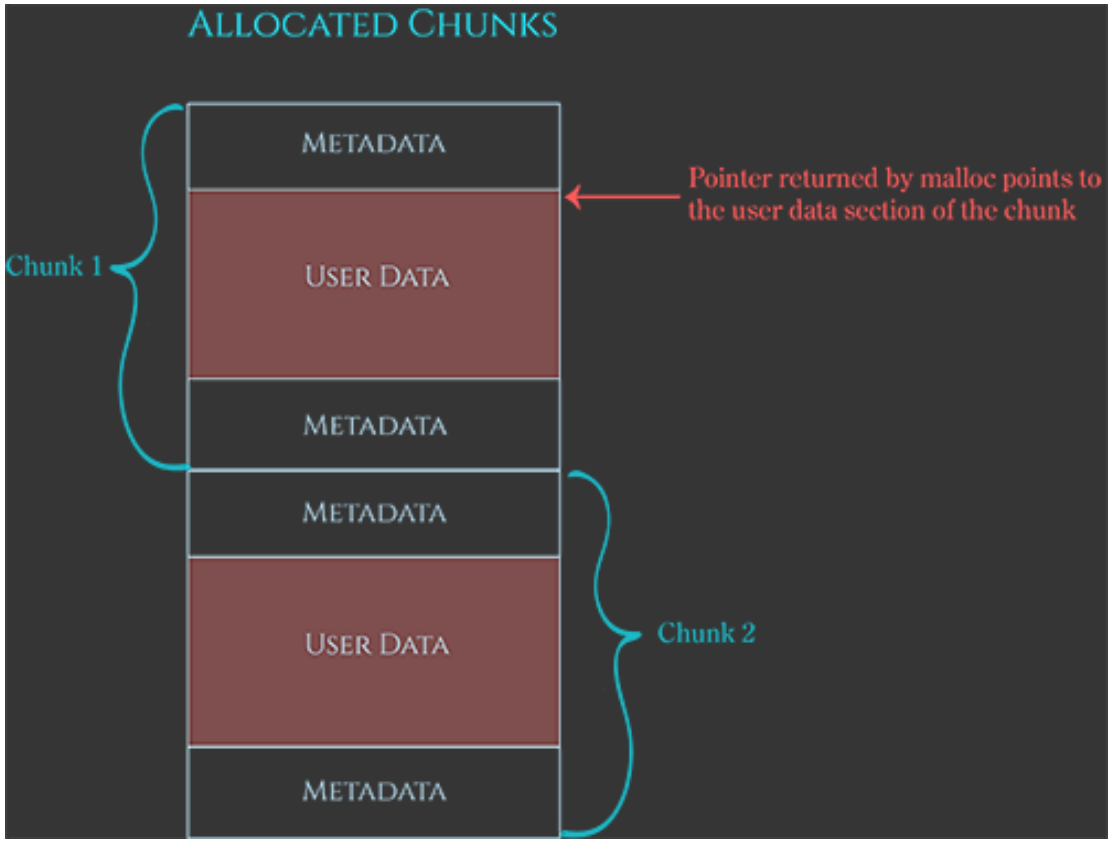
- Calling conventions modificam a forma de dados serem colocados na stack.
- https://wiki.osdev.org/Calling_Conventions

- Calling conventions modificam a forma de dados serem colocados na stack.
- https://wiki.osdev.org/Calling_Conventions
- E se eu executar um memset() de tamanho maior do que o de uma variável da stack?

Alocação dinâmica de memória

- Espaços de memória constantes entre diferentes registros de ativação

Heap


ALLOCATED CHUNKS
METADATA
USER DATA
METADATA
METADATA
USER DATA
METADATA
Chunk 1
Chunk 2
Pointer returned by malloc points to the user data section of the chunk

Heap

- Cada biblioteca C implementa um gerenciamento de heap diferente

Heap

- Cada biblioteca C implementa um gerenciamento de heap diferente
- https://azeria-labs.com/heap-exploitation-part-1-understanding-the-glibc-heap-implementation/

E o SO?

Já vimos como cada processo funciona separadamente, mas e o SO?

O que é um Sistema Operacional (SO)?

O que é um Sistema Operacional (SO)?

- Software de baixo nível que:
  - abstrai recursos do sistema
  - gerencia recursos do sistema
  - provê segurança

O que é um Sistema Operacional (SO)?

- Software de baixo nível que:
  - abstrai recursos do sistema
  - gerencia recursos do sistema
  - provê segurança
- SO provê serviços para as aplicações
  - System Calls

O que é o kernel de um SO?

O que é o kernel de um SO?

- kernel é a parte de código do SO que executa com o processador em modo privilegiado

SO == kernel?

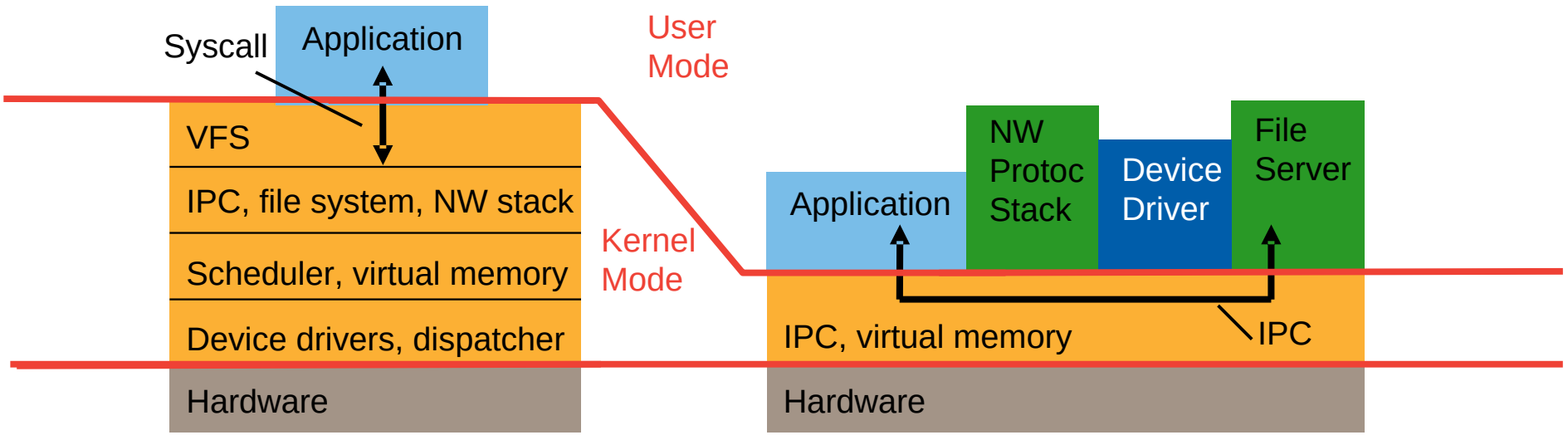
SO == kernel?

- serviços (daemons)

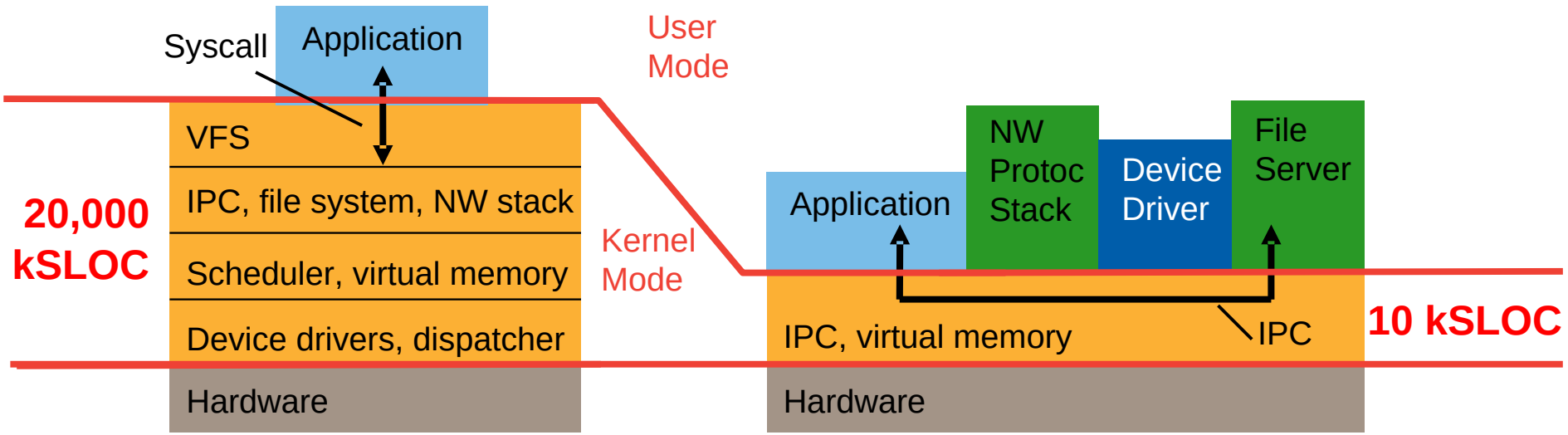
SO == kernel?

- serviços (daemons)
- arquitetura

Kernel monolítico X Microkernel



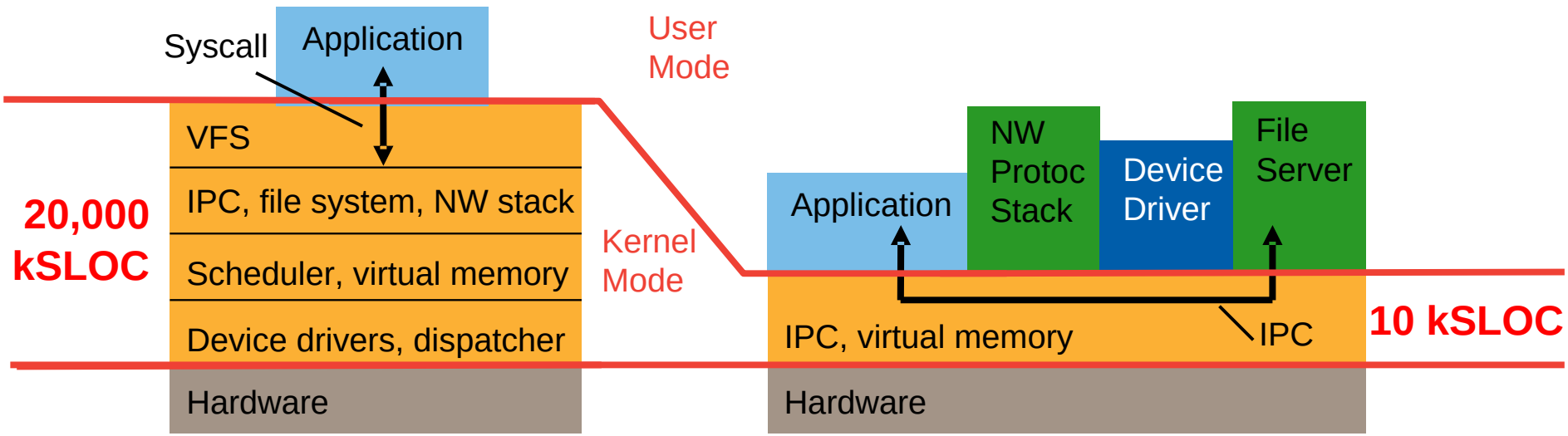
Kernel monolítico X Microkernel



Kernel monolítico X Microkernel

- Novas funcionalidades são adicionadas ao kernel
- Novas políticas são adicionadas ao kernel
- Complexidade do kernel aumenta

- Funcionalidades são adicionadas ao espaço de usuário
- Políticas são adicionadas ao espaço de usuário
- Complexidade do kernel é estável



Alguns mecanismos importantes para segurança

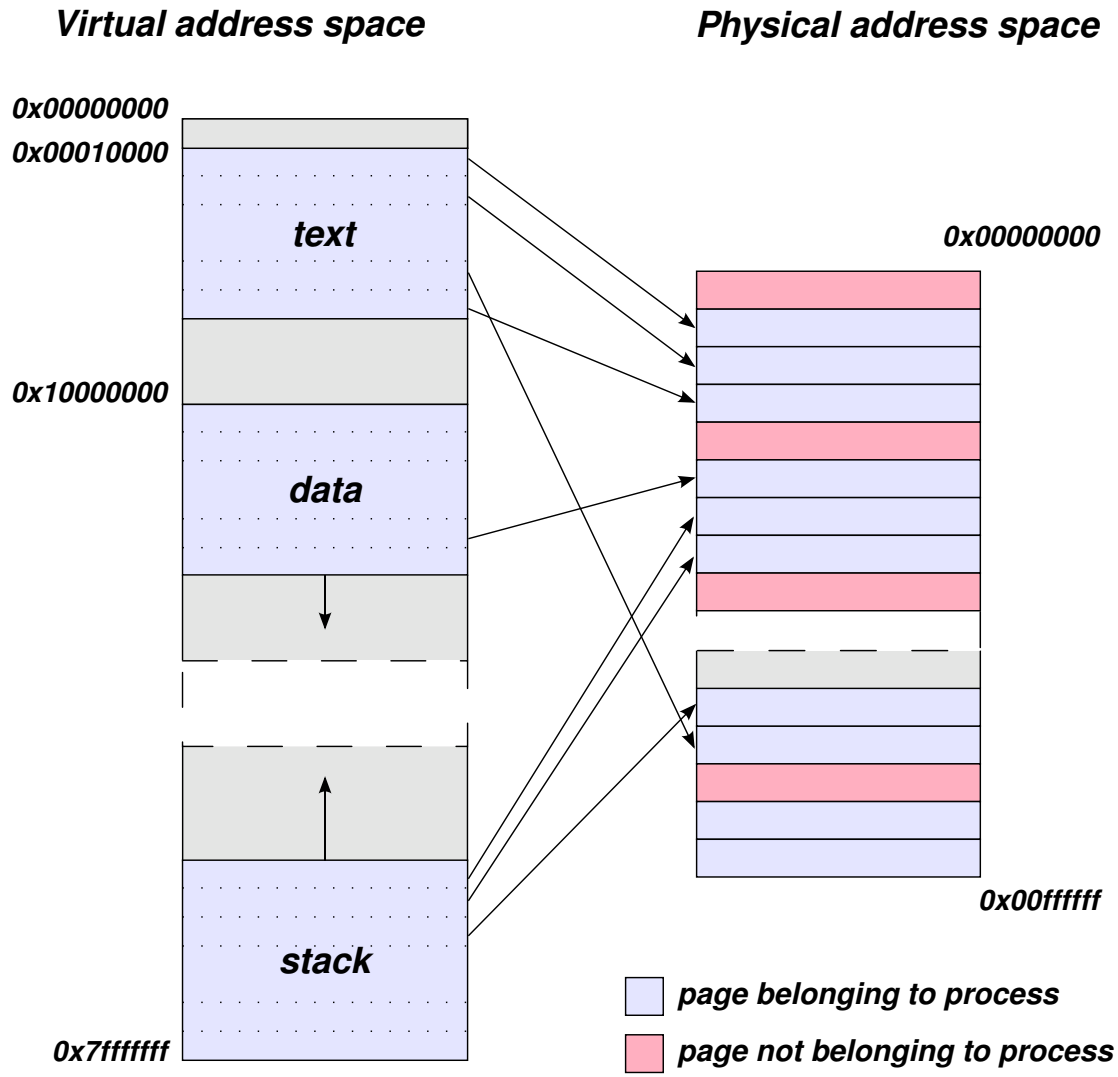
- Virtual Memory
- Scheduler / Dispatcher
- Network
- File system
- IPC
- Drivers

Alguns mecanismos importantes para segurança

- Memória Virtual
- Scheduler / Dispatcher
- Rede
- Sistema de arquivos
- IPC
- Drivers

- Vimos que cada processo tem seu espaço de endereçamento próprio
- Um espaço de 48 bits para cara processo
- Mas só existe uma memória física

Memória Virtual



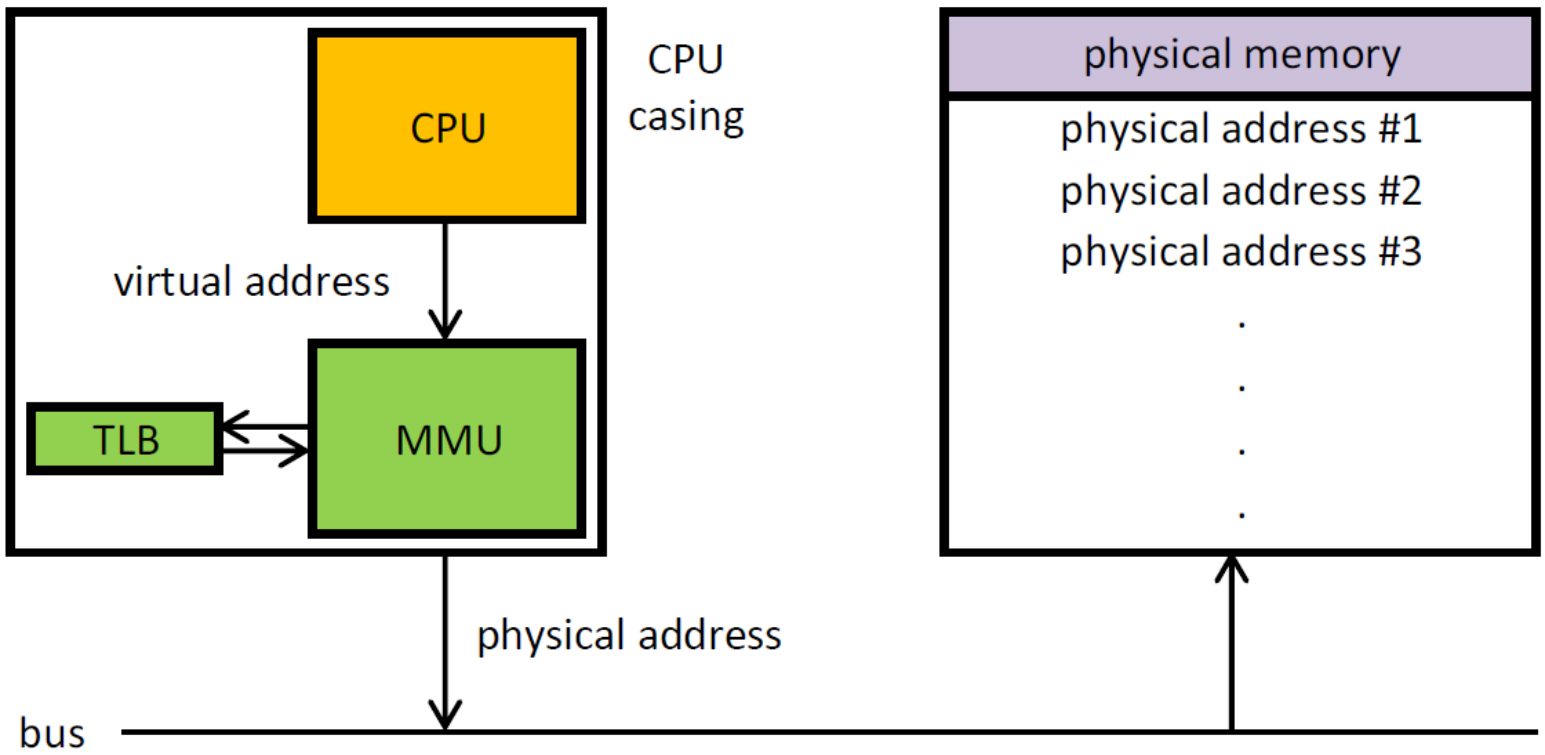
- Cada memória virtual é definida por uma tabela de páginas

- Cada memória virtual é definida por uma tabela de páginas
- Cada processo possui uma tabela de páginas, ou seja, espaço de endereçamento completo e independente

- Cada memória virtual é definida por uma tabela de páginas
- Cada processo possui uma tabela de páginas, ou seja, espaço de endereçamento completo e independente
- Proteção a nível de página

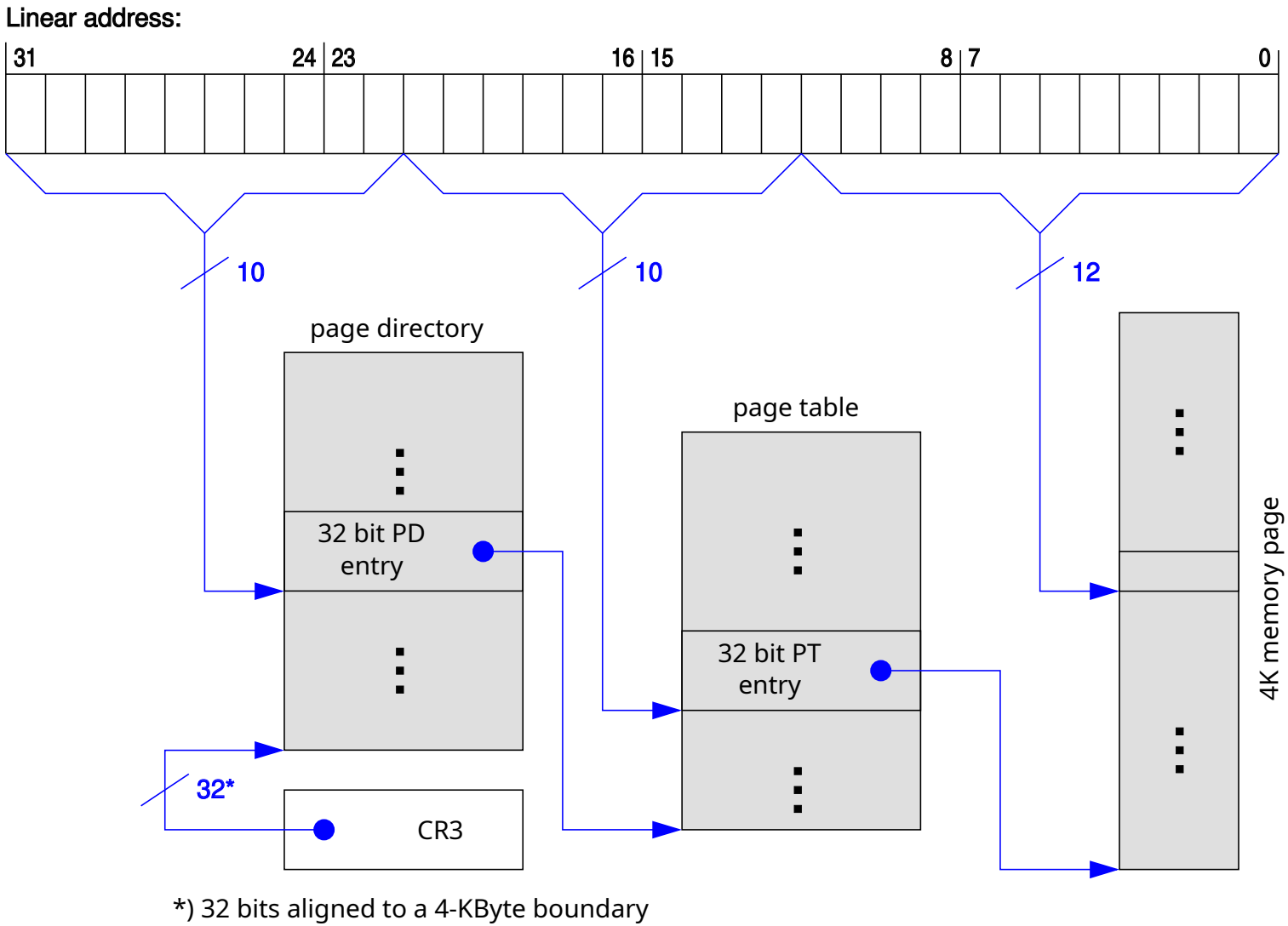
- Quando existe uma troca de contexto, existe uma troca de tabelas de páginas
- Registrador CR3 contém o endereço da tabela de páginas sendo utilizada no momento

Memória Virtual



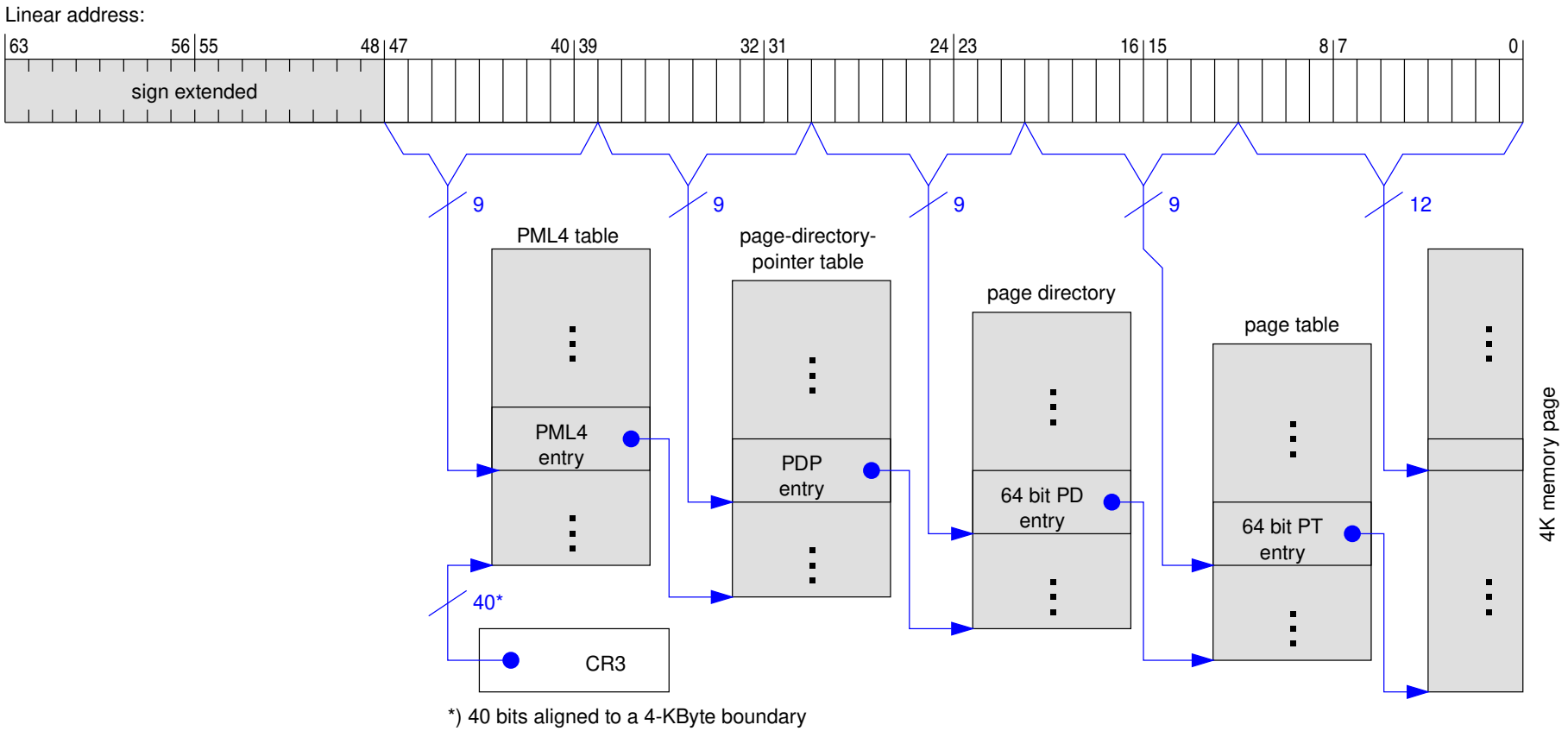
CPU: Central Processing Unit
MMU: Memory Management Unit
TLB: Translation lookaside buffer

Memória Virtual



*) 32 bits aligned to a 4-KByte boundary

Memória Virtual



Parte do SO que:

Parte do SO que:

- gerencia os dados no dispositivo de armazenamento

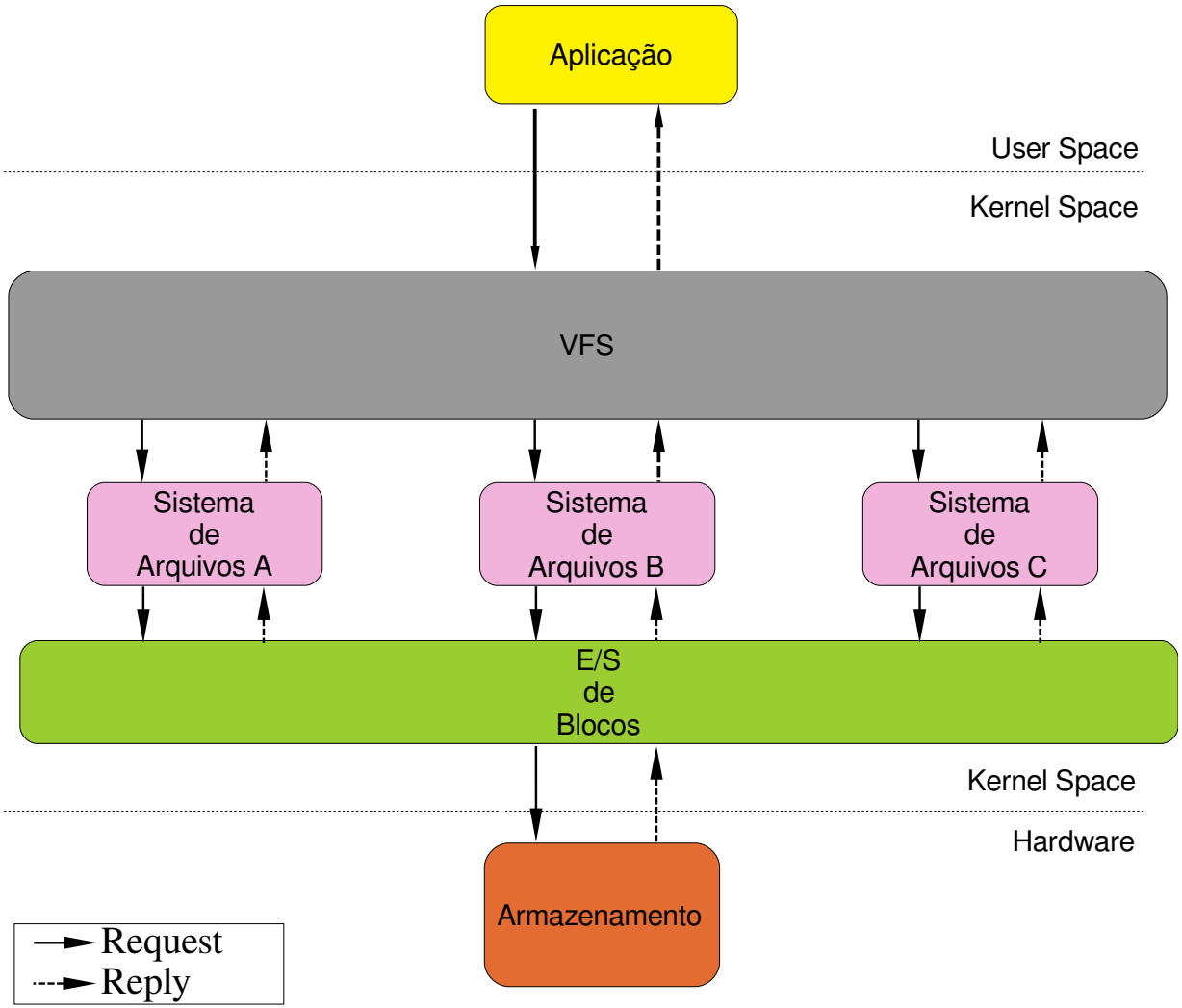
Parte do SO que:

- gerencia os dados no dispositivo de armazenamento
- provê interface para o usuário acessar dados do dispositivo de armazenamento

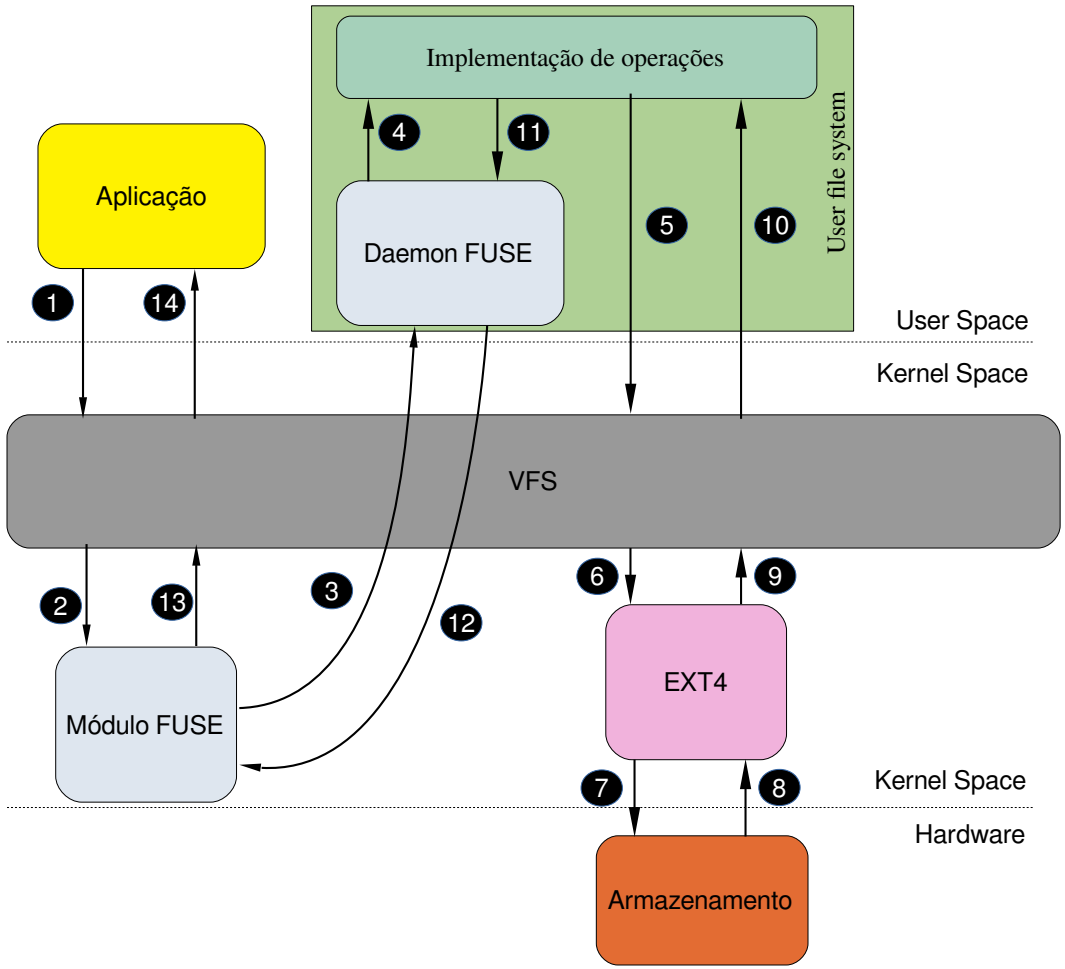
Parte do SO que:

- gerencia os dados no dispositivo de armazenamento
- provê interface para o usuário acessar dados do dispositivo de armazenamento
- provê segurança aos dados

Sistema de arquivos no Linux



Sistema de arquivos em espaço de usuário



"On a UNIX system, everything is a file; if something is not a file, it is a process."

Page Cache

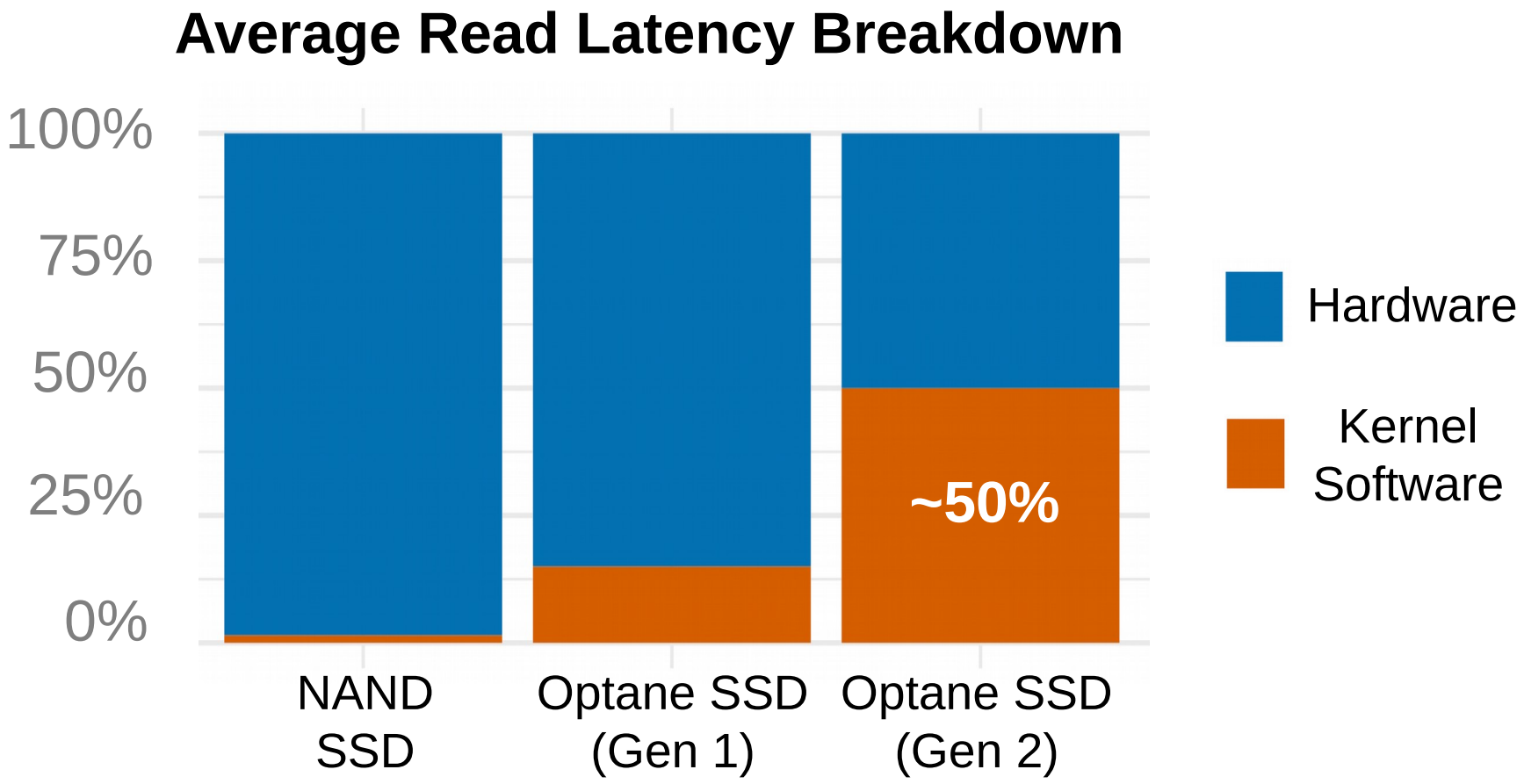
- Conjunto de páginas em memória destinadas a armazenar páginas do dispositivo de armazenamento

Page Cache

- Conjunto de páginas em memória destinadas a armazenar páginas do dispositivo de armazenamento
- Acesso ao dispositivo de armazenamento pode ser ordens de magnitude mais lento que acesso à memória

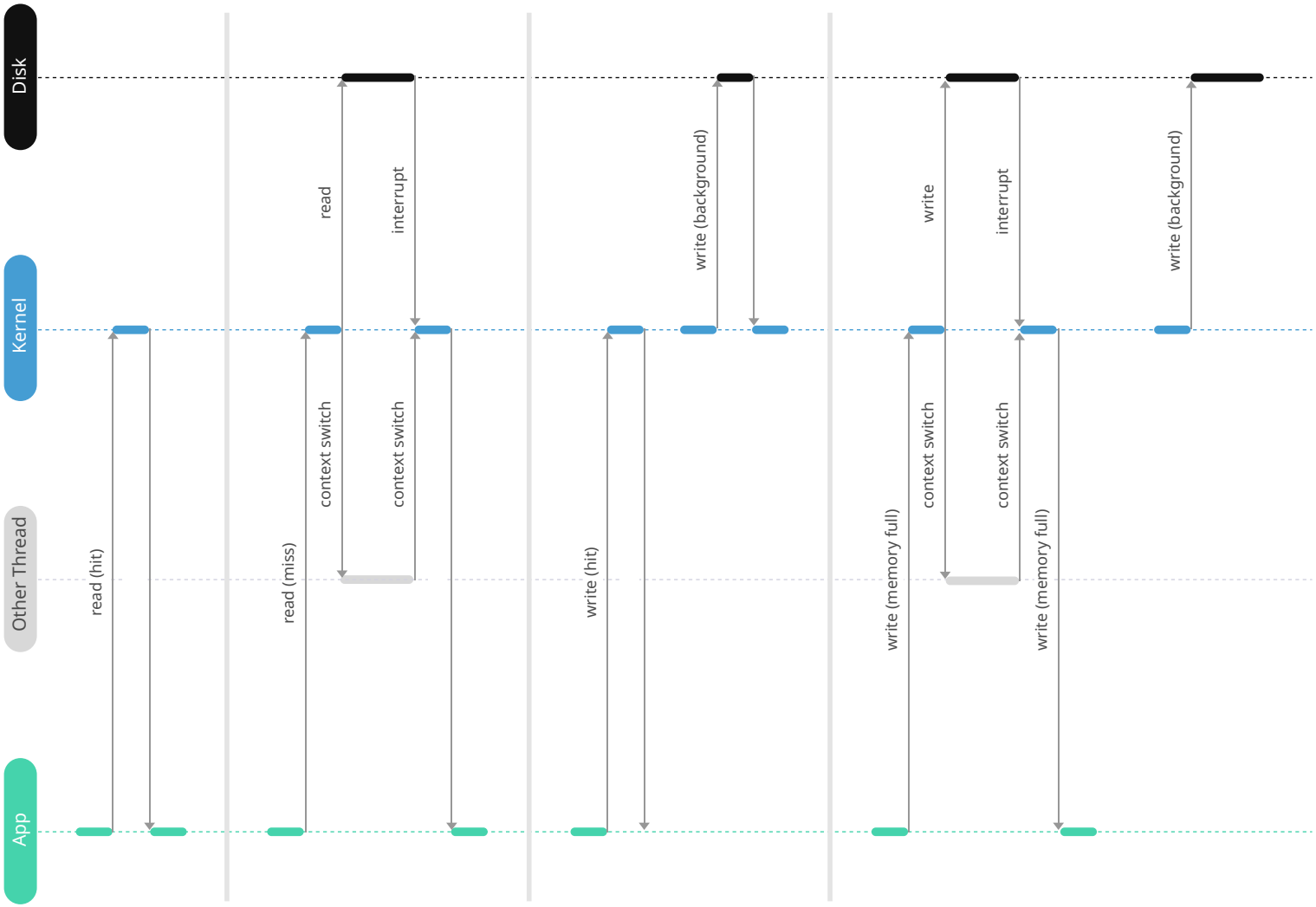
Page Cache

- Conjunto de páginas em memória destinadas a armazenar páginas do dispositivo de armazenamento
- Acesso ao dispositivo de armazenamento pode ser ordens de magnitude mais lento que acesso à memória
- Está mudando com o passar do tempo

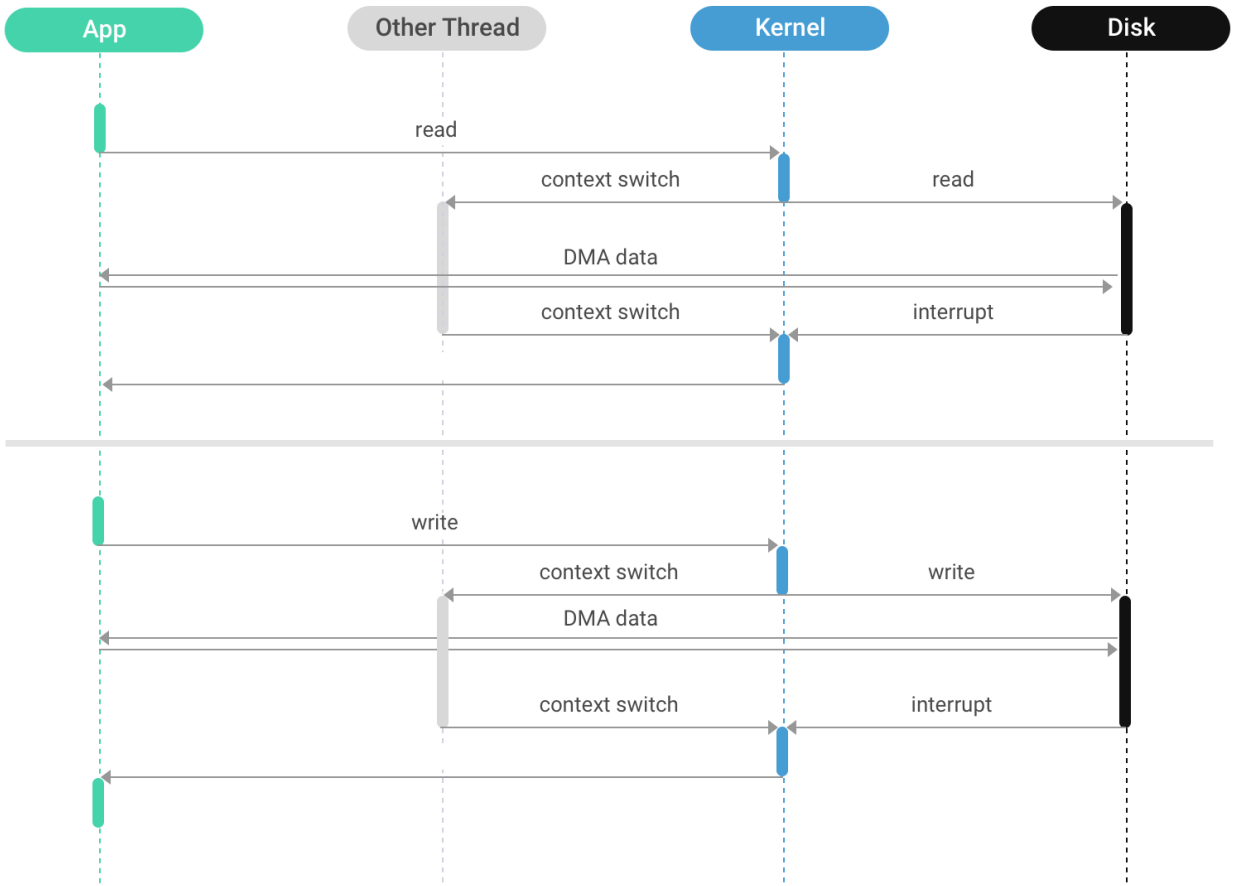


Workload: Random 512B Read

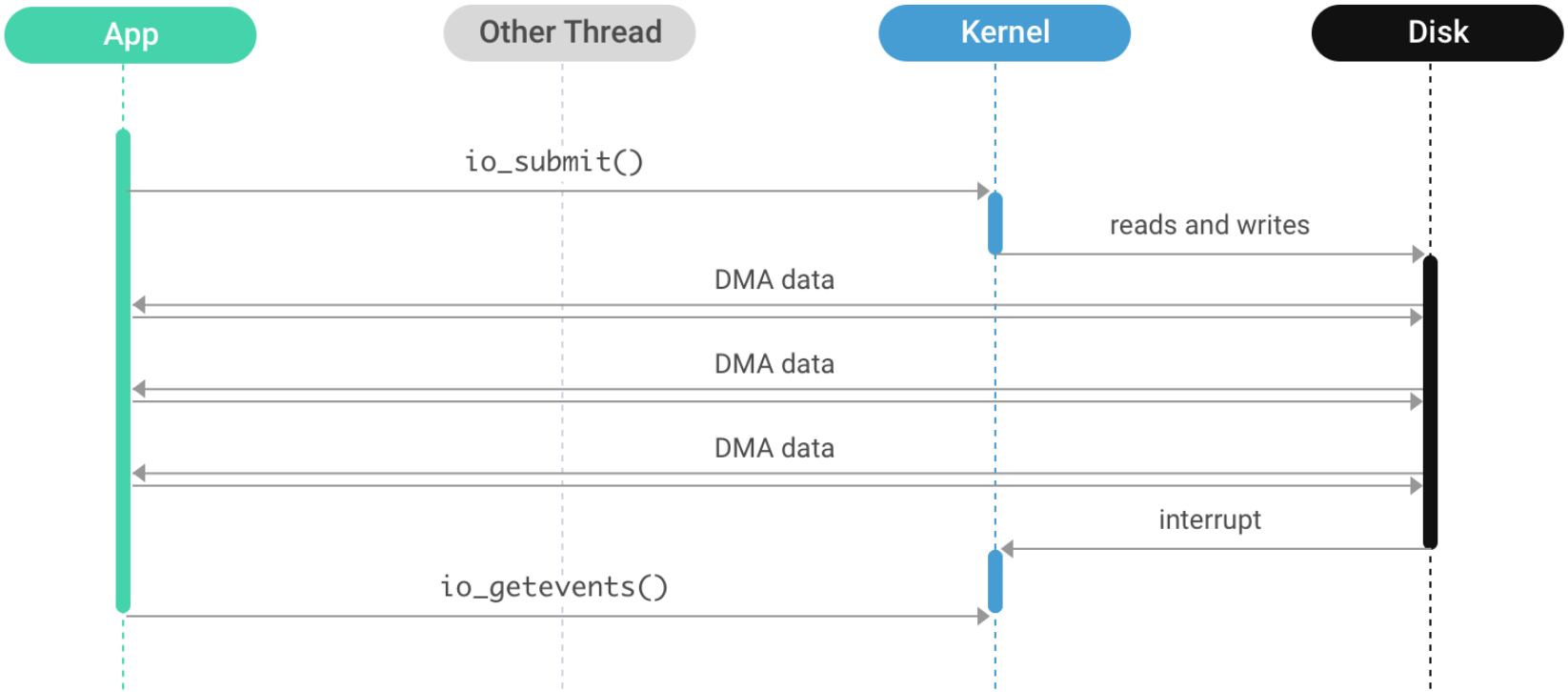
Buffered I/O (tradicional)



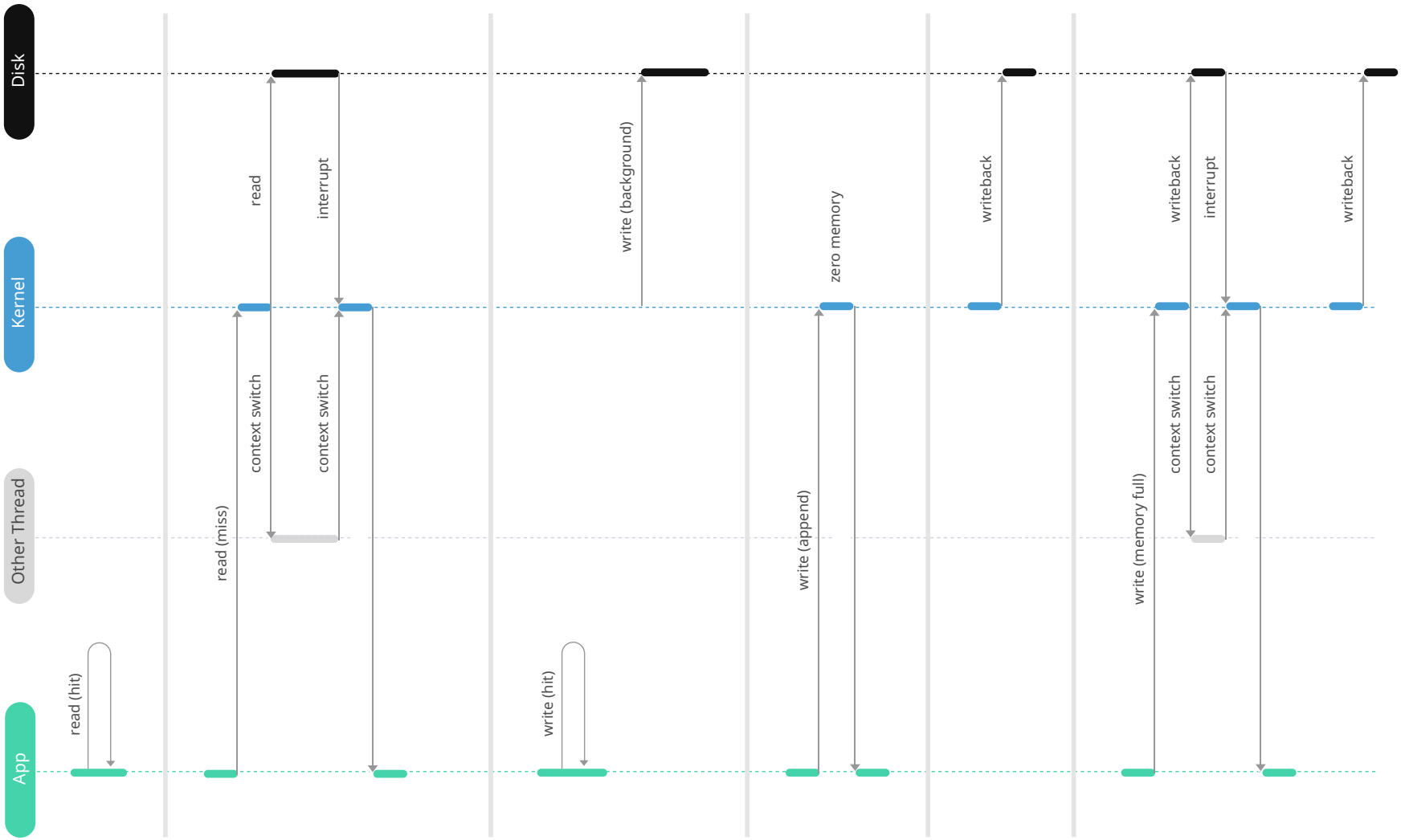
Direct I/O (Bypass da Page Cache)



Asynchronous I/O (AIO)



Mapeamento de Arquivos em Memória (mmap)



IO uring

- https://www.youtube.com/watch?v=AaaH6skUEI8
- https://www.blackhat.com/us-23/briefings/schedule/#bad-io_uring-a-new-era-of-rooting-for-android-32243
- https://www.phoronix.com/news/Google-Restricting-IO_uring