



O Guia Definitivo para Sobreviver a Sistemas Operacionais: Parte I

O Guia Definitivo para Sobreviver aos Sistemas Operacionais

Opala Corp.

Mr Punk da Silva

Table of Contents

Sistemas Operacionais	2
Capítulo 1	3
Operação do Computador	6
Estrutura de Armazenamento	11
Estrutura de Entrada e Saída	19
Arquitetura do Sistema	26
Caching	29
Questões 1	36
Prática 1	43
Hardware	49
Software	50
Respostas	51

Sistemas Operacionais

Bem-vindo ao nosso guia sobre Sistemas Operacionais! Nesta obra, exploraremos os conceitos fundamentais que regem o funcionamento dos sistemas que permitem que nossos dispositivos funcionem de maneira eficaz. Os sistemas operacionais são uma peça crucial da tecnologia moderna, servindo como intermediários entre o hardware e o software, gerenciando recursos, permitindo a execução de aplicativos e garantindo uma experiência de usuário fluida.

Este PDF é estruturado para proporcionar uma compreensão acessível e prática dos sistemas operacionais, abrangendo desde a teoria básica até exercícios práticos que reforçam o aprendizado. Se você é estudante, entusiasta ou profissional na área de tecnologia, este material foi feito para você! Prepare-se para se aprofundar neste universo fascinante, desenvolver novas habilidades e, quem sabe, até se divertir no processo.

Capítulo 1

Introdução

Um **Sistema Operacional** é uma interface que interliga o **hardware** e o **software**, fazendo o gerenciamento dos dois mundos do computador.

i A inexistência, de um **SO** resulta em um impedimento de uma interação natural com o computador.

Como todo grande projeto complexo caímos em uma operação em que deve ser bem definida em como o **SO** irá trabalhar, quais são seus requisitos.

E o **SO** tem algumas funções básicas que é:

- dar partida no sistema
- gerenciar memória
- gerenciar dispositivos E/S

O que os Sistemas Operacionais fazem

Um **sistema computadorizado** ou só computador, pode ser *dividido em quatro partes*:

- Hardware ([Hardware](#))
- Sistema Operacional
- Software ([Software](#))
- Usuários
- Também podemos considerar que um sistema computadorizado é composto por:

```
[0101] |                                     | [  ] -> Finge que é um PC
[010]  |--: Dados                           Hardware :--| |==|
[01]   |                                     |         | ----
```

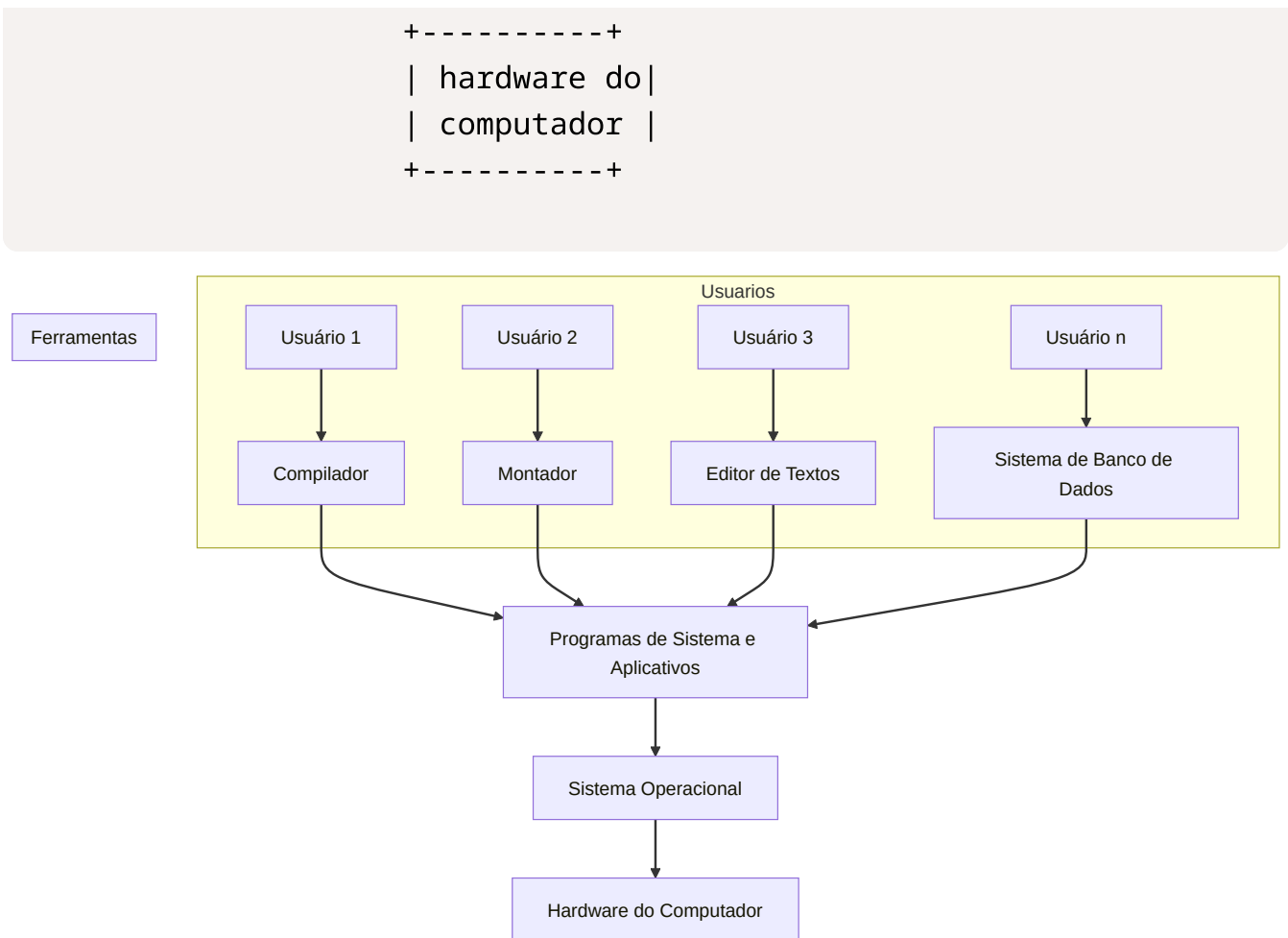
```

-- Software----
    |
    |
  |-----|
  | WIN95  |
  |-----|
  
```

- Exemplo de Funcionamento de um Sistema Operacional

```

+-----+ +-----+ +-----+ +-----+
| usuário | | usuário | | usuário | | usuário |
|   1   | |   2   | |   3   | |   n   |
+-----+ +-----+ +-----+ +-----+
    |           |           |           |
    |           |           |           |
+-----+ +-----+ *-----+ *-----+
| compilador | | montador | | editor de | | sistema |
|           | |           | | textos  | | de banco |
|           | |           | |         | | de dados |
+-----+ +-----+ +-----+ +-----+
    |
    |
    +-----+
    | programas |
    | de sistema |
    | e aplicat- |
    | ivos      |
    +-----+
    |
    |
    +-----+
    | sistema   |
    | operacional|
    +-----+
    |
    |
  
```



Operação do Computador

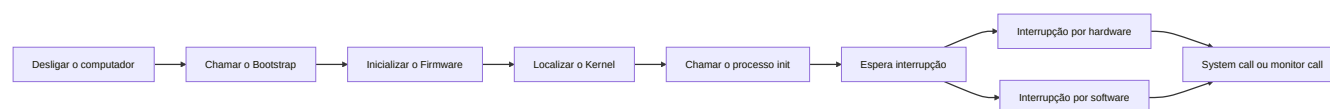
Ao desligar o computador e ligá-lo, o que acontece? Como ele "chama" o Sistema Operacional.

Para o computador começar a funcionar, ele chama um programa básico, chamado de **bootstrap**. Normalmente, este programa está alocado na memória apenas de leitura (**ROM**) ou é salvo na memória de somente leitura apagável programavelmente (**EEPROM**).

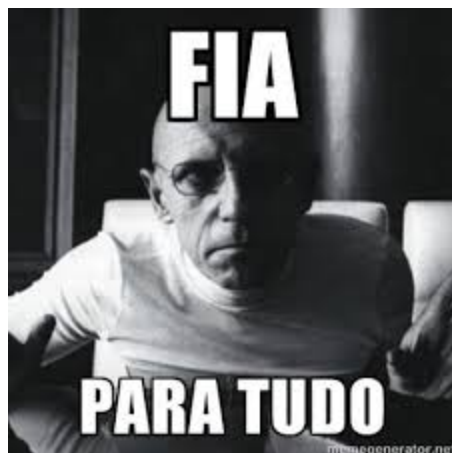
Este programa é conhecido como **Firmware**, pois está instalado diretamente no hardware, assim, ele inicializa todos os aspectos do sistema, desde os registradores da CPU até os dispositivos e o conteúdo na memória.

Para carregar o SO, ele precisa localizar o **Kernel**, que é o núcleo do sistema operacional. Assim que o Kernel é carregado na memória do computador, ele chama um processo chamado **init**, que espera uma interrupção do sistema ou do hardware. Os dois casos são:

- Se for pelo hardware, ele envia uma interrupção por sinal para a CPU, via normalmente o barramento do sistema;
- Se for por software, ele pode fazer de duas maneiras: chamando uma **system call** (chamada do sistema) ou usando um **monitor call** (monitor de chamada). Essas são operações especiais executadas para disparar uma interrupção, enviando um sinal para a CPU.



Quando a CPU recebe uma interrupção, ela para o que está fazendo e executa a rotina de tratamento correspondente:



Meme fia para tudo

A CPU então manda a execução para uma **localização fixa na memória**, onde essa localização contém o **endereço inicial** da rotina para **atender a essa interrupção**.


Essas **interrupções** podem ser tratadas de diferentes maneiras, e cada computador possui seu próprio mecanismo. Um método simples para isso é tratar a transferência chamando uma rotina genérica. Para dar mais enfoque em velocidade pode ser usada uma **tabela de ponteiros a pontando para as interrupções**, já que elas devem ser predefinidas. **Essa tabela é armazenada em memória baixa**, sendo ela a primeira parte ou locação da memória.



Esse **vetor de interrupção** vai ser indexado exclusivamente pelo número do dispositivo, fornecido com a requisição da interrupção para gerar o endereço do tratamento da interrupção:




Interrupção 🔔

CPU manda execução para local fixo na 💾, com endereço da rotina de tratamento. 🏃

Diferentes formas de tratar interrupções, cada 💻 com seu próprio jeito.

Método simples:
Transfere para uma
rotina genérica.  bootstrap

Método rápido:
 Tabela de
ponteiros para
interrupções, em
memória baixa. 

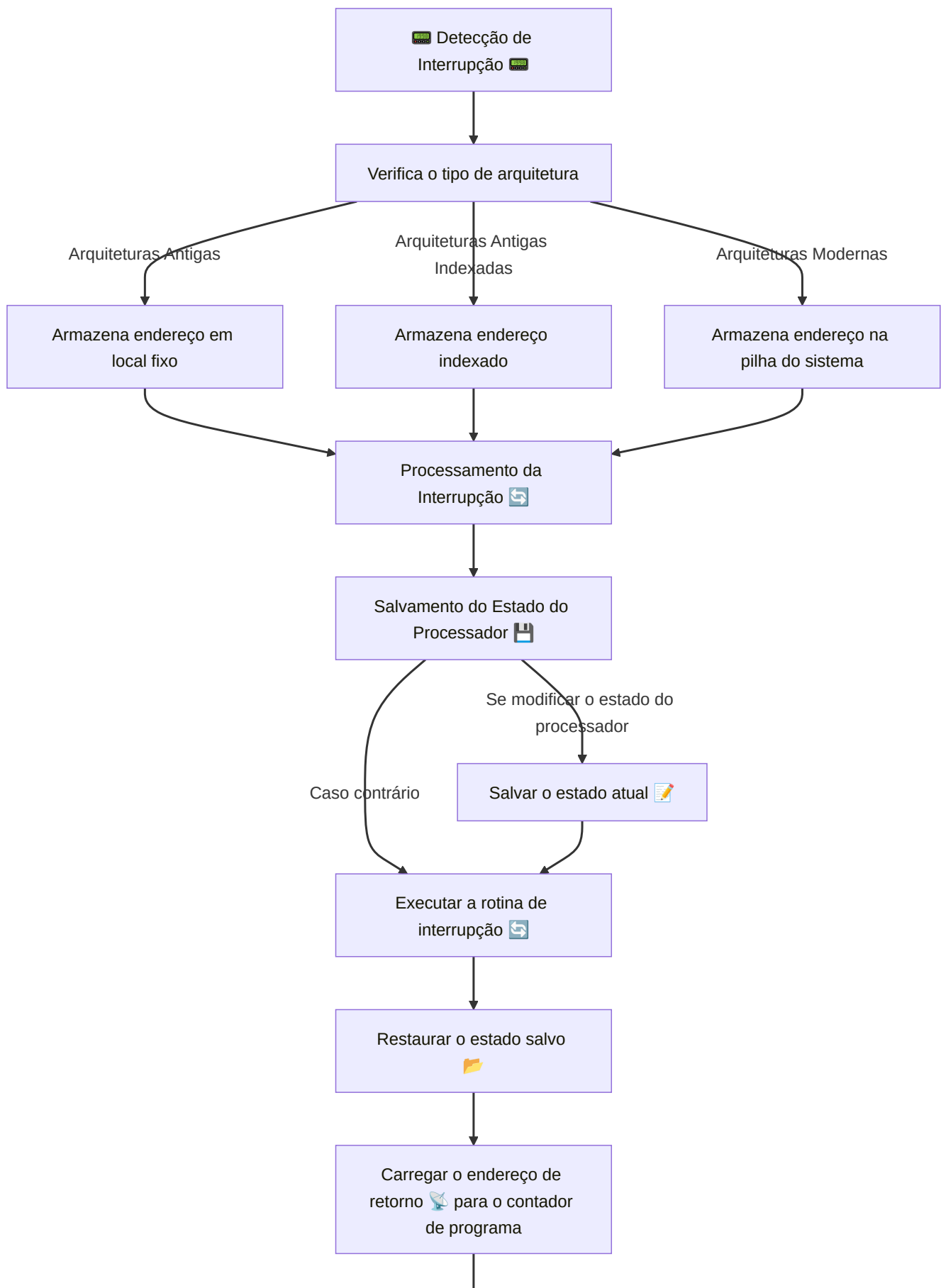
 Vetor usa
 dispositivo
para gerar
endereço do
tratamento. 

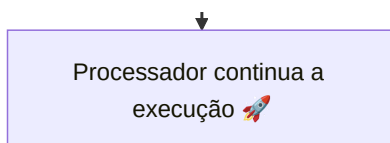
A arquitetura de interrupção **precisa salvar o endereço da instrução interrompida**, em projetos:

- Em alguns antigos armazenam o endereço da interrupção de **maneira fixa ou local indexado** por um numero do dispositivo;
- Em arquiteturas modernas, eles armazenam em **pilhas do sistema**;

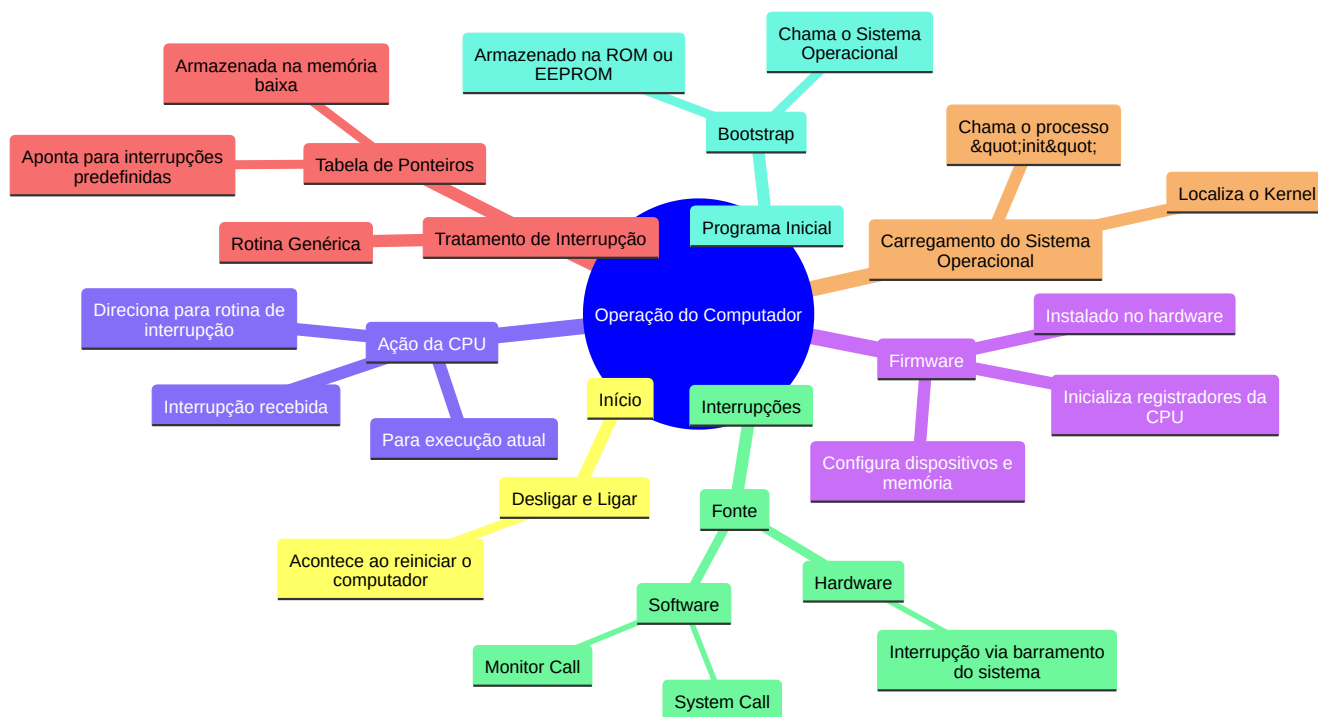
Se a rotina de interrupção precisar modificar algum estado do processador, por exemplo alterando os valores do **registrador**:

- Ela vai **salvar** o estado atual, explicitamente;
- Depois **carregar** e **restaurar** esse estado para depois **retornar**;
- Em seguida será carregado para o **contador de programa** o **endereço do retorno** e o **processador** que foi **interrompido** continua como se nada tivesse acontecido:





Diagrama



Estrutura de Armazenamento

Para os computadores que temos a **CPU** só consegue carregar instruções que vêm diretamente da memória.

- A memória não sendo nada, mas a **Memória Principal** – aquela cujo acesso é randômico, ou seja, desligar o PC não apaga os dados armazenados, que é a memória **RAM**.

Diagrama



[Veja mais sobre tipos de memória em:](#)

A memória RAM é comumente feita numa arquitetura de semicondutores chamada de **Dynamic Random Access Memory (DRAM)** ou, em português, **memória de acesso dinâmica**.

Um outro tipo de memória é aquela que só serve para leitura, assim como a mulher do seu amigo, apenas olhe. As conhecidas são:

- **ROM (Read Only Memory)** ==> normalmente vem nos computadores e é usada para armazenar o programa bootstrap.

- Além disso, é usada por empresas de jogos para guardar os jogos, já que ela possui essa natureza imutável.
- **EEPROM** (Electrically Erasable Programmable Read Only Memory)
 - Por não ser modificado com frequência, essa memória costuma ser usada para armazenar programas padrões de modo estático.
 - Smartphones, por exemplo, utilizam a EEPROM de modo que as fabricantes armazenam nele os aplicativos de fábrica.

Quaisquer destas memórias utilizam **um array de words** ou uma **unidade de armazenamento**.

- Cada *word* possui seu próprio endereço.
- As interações se dão por instruções:
 - **load** - carrega um endereço específico da **memória principal** para um dos **registradores** da CPU.
 - **store** - move um conteúdo de um **registrador da CPU** para a **memória principal**.

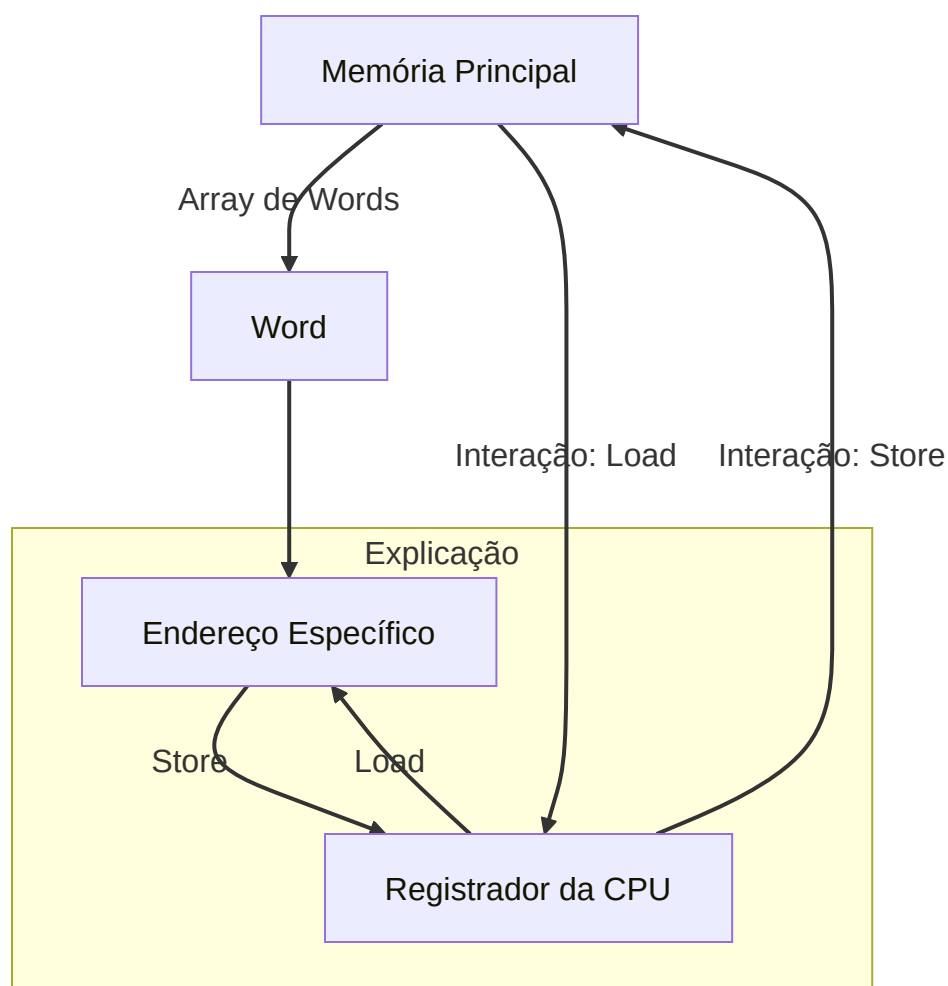


Ilustração de um esquema sobre instruções da CPU (load e store)

i A CPU carrega e armazena essas instruções tanto explicitamente (dizer para ela fazer) como de maneira automática - ela faz sozinha o carregamento da memória principal para serem executadas.

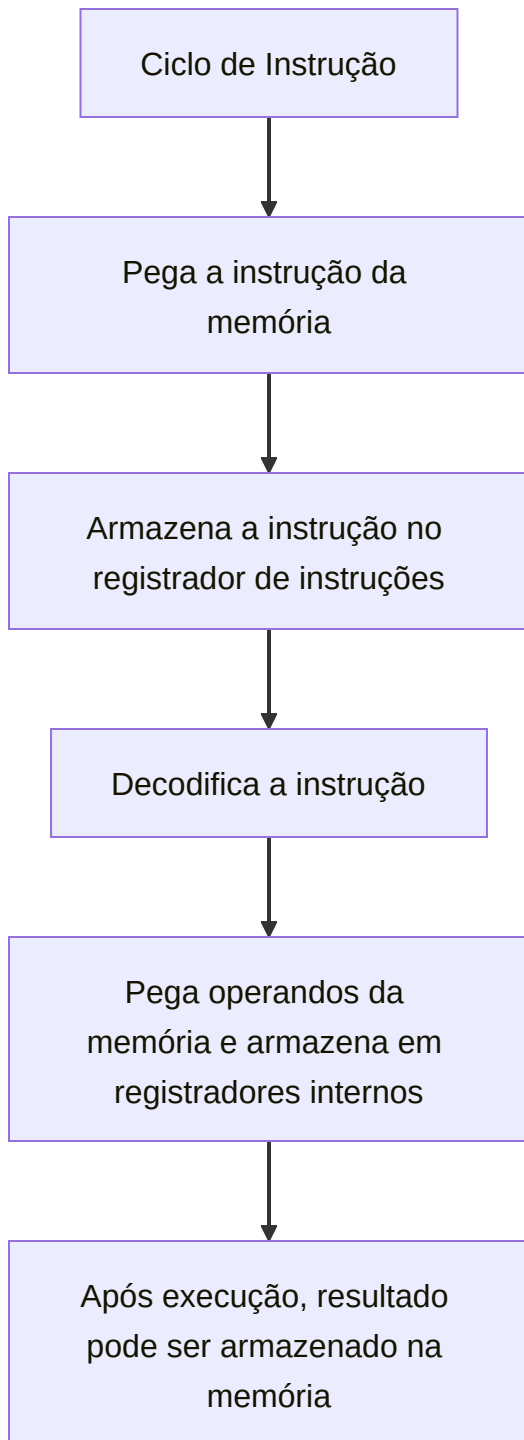
A arquitetura mais usada nos computadores modernos é a de **Von Neumann**. Essa arquitetura funciona da seguinte forma:

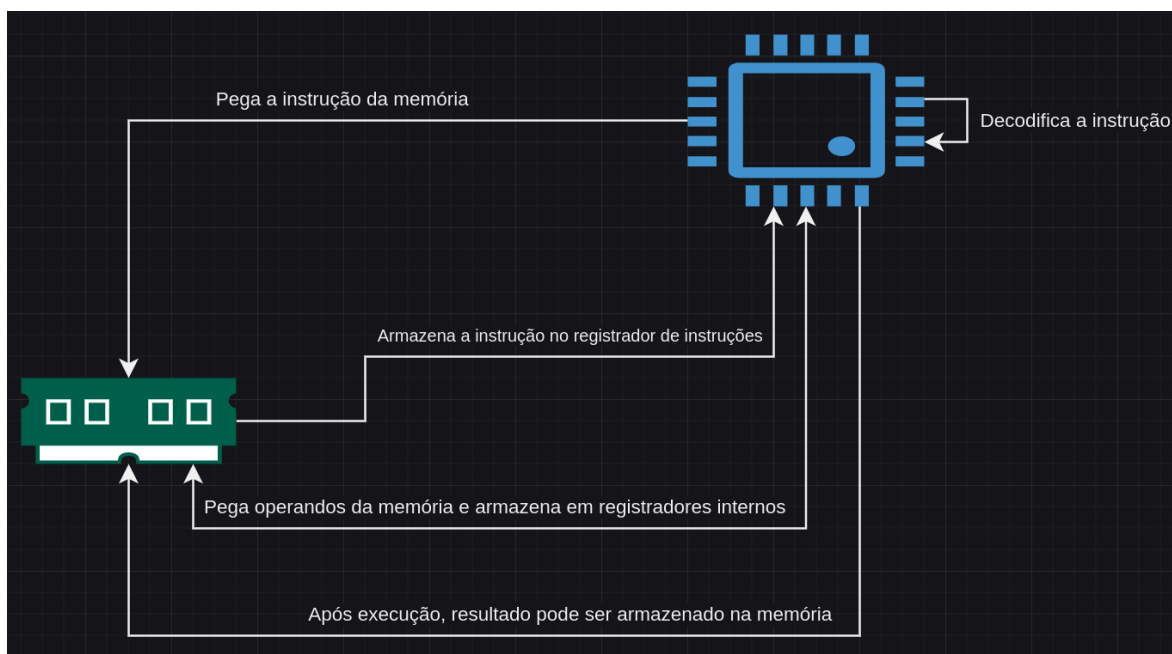
- Programas e dados são armazenados na memória principal.
- A CPU gerencia a memória principal.

Vamos para um ciclo de execução - quando uma instrução é dada:

1. Pega a instrução da memória.
2. Armazena essa instrução no **registrador de instruções**.
3. Essa instrução é então decodificada.
 1. Pode pegar operandos da memória e armazená-los em registradores internos.
4. Após a execução dos operandos, o resultado pode ser armazenado na memória.

Diagramas de Execução de Instrução





003 - Estrutura de Armazenamento

i A unidade de memória só consegue ver um fluxo de endereços de memória. Ela não sabe:

- Como são gerados (Gerados por contador de instruções, indexação, endereços literais e etc)
- Para que servem
- Se são instruções ou dados.

Seria bom, mas a vida não é um morango, a memória principal não consegue armazenar todos os dados e programas. Entretanto, não temos isso, já que:

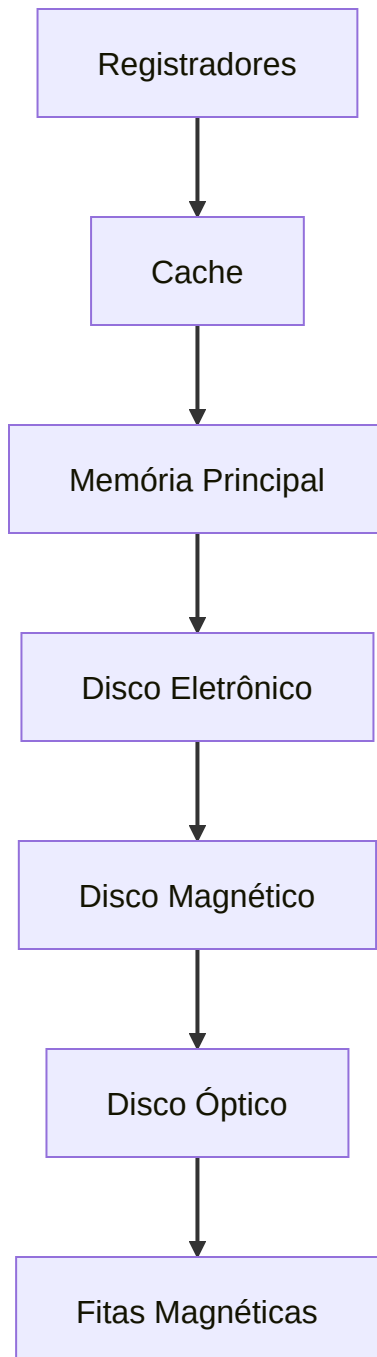
- **A memória principal é volátil**, ela perde os dados assim que a máquina é desligada.
- A memória principal possui um **armazenamento irrisoriamente pequeno** para armazenar todos os programas e dados.

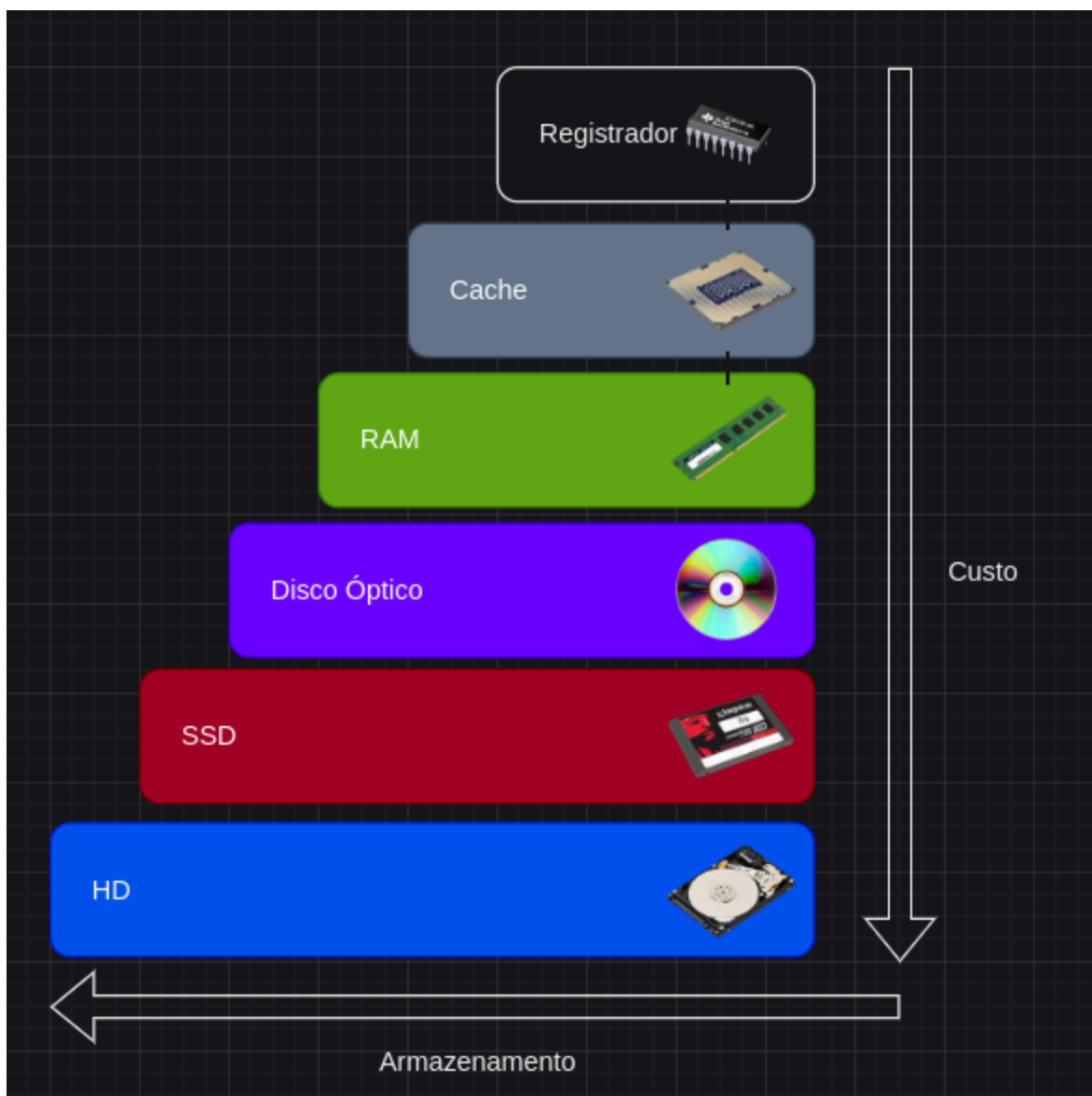
Assim, precisamos de outro tipo de memória chamado **memória secundária**, que tem o propósito de armazenar dados e programas de maneira permanente.

Um bom exemplo de memória secundária é o HD (Disco Rígido) e também temos outro tipo que está se tornando mais popular no mercado, o SSD (Disco de Estado Sólido).

No entanto, não há apenas dispositivos de armazenamento nessa hierarquia. Também podemos fazer uma hierarquia desses dispositivos, que é assim:

Diagramas de Dispositivos de Armazenamento:





003 - Estrutura de Armazenamento Hierarquia Dispositivos De Armazenamento

Estrutura de Entrada e Saída

Os dispositivos de Entrada e Saída (ou E/S), são um dos grandes pontos importantes para um Sistema Operacional, como podemos notar no armazenamento que possui grande importância para ser um dispositivo de E/S.

- Um outro ponto importante é que grande parte do código do SO é pensado para E/S;
 - Tanto por causa da **confiabilidade** como **desempenho**.

i Um sistema computadorizado para uso geral, consiste em:

- CPU
- Diversos tipos de controladores de dispositivos conectados por um barramento comum
- Cada controlador possui um tipo específico de dispositivo

Por exemplo, para o controlador SCSI (Small Computer-System Interface) podemos ter sete ou até mais dispositivos conectados ao mesmo controlador.

Cada controlador armazena **buffer local** e um **conjunto de registradores de uso especial**.

Os controladores tem duas funções básicas, que se baseiam:

- **Move** os dados para os dispositivos periféricos que controla.
- **Gerencia** o uso do buffer local.

Tais sistemas possuem um **driver de dispositivo** (driver de dispositivo) que serve como ponte entre o dispositivo e o sistema, permitindo que a **entrada dos dispositivos** tenha uma **saída uniforme** para o restante do sistema.

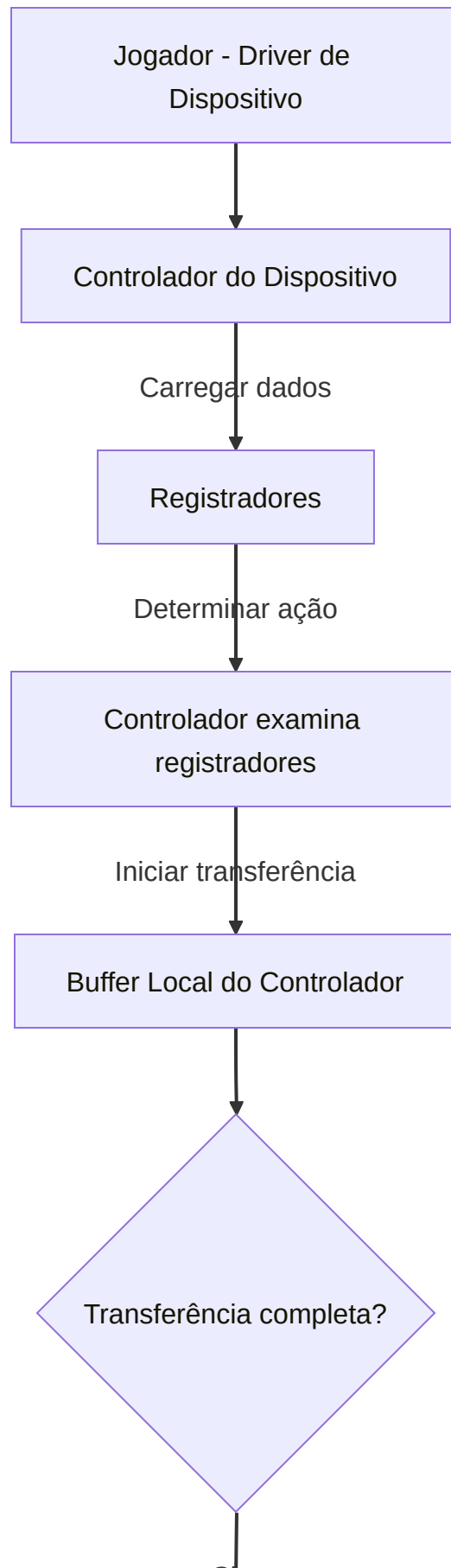
O funcionamento de uma operação de E/S:

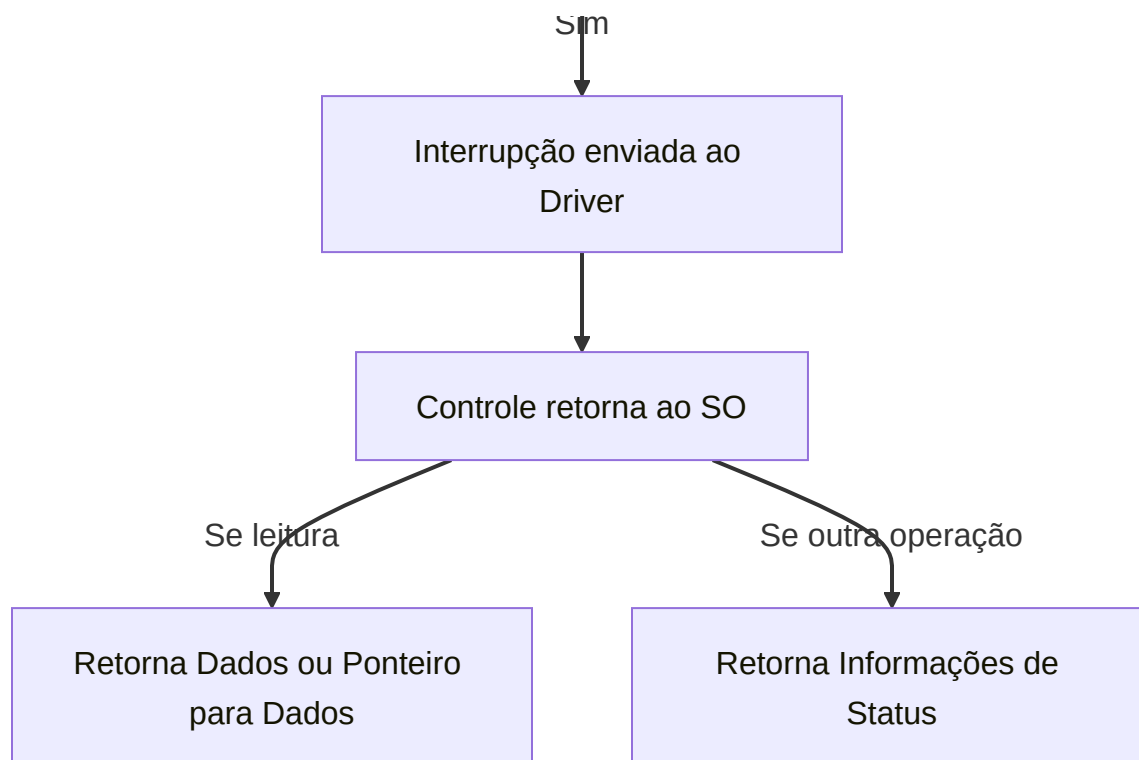
- O **driver de dispositivo** carrega os **registradores** apropriados para dentro do

controlador do dispositivo.

- O **controlador** examina o **conteúdo** que tem nos **registradores**, para determinar que ação deve ser tomada.
- O controlador começa a transferir os dados do dispositivo para o seu buffer local.
- Assim que a transferência está concluída, o **controlador de dispositivo** envia uma **interrupção** para o **driver de dispositivo** informando que a transferência foi concluída.
- O driver de dispositivo então retorna o controle diretamente para o SO, retornando os dados ou um ponteiro para esses dados, possivelmente, caso a operação seja de leitura.
 - Para outras operações, o driver retorna informações de status.

Representação:





i Para pequenas porções de dados, essa arquitetura de E/S por interrupção funciona bem, mas não funciona somente com isso há muito tempo, por isso, se usarmos essa forma para grandes volumes de dados como E/S de disco causa um **overhead** (que é uma sobrecarga).

Com esse grande problema, precisamos então de um outro dispositivo, um que armazene esses dados para que o acesso seja mais rápido, para isso usamos a **DAM** (Direct Access Memory ou Memória de Acesso Direto).

Logo o ciclo se torna assim:

- Depois de configurar buffers, ponteiros e contadores, o dispositivo de E/S, o controlador de dispositivo **move um bloco inteiro de dados** diretamente para ou do seu próprio buffer local para a memória.
- Somente **uma interrupção é feita por bloco**, para que seja avisado ao driver de dispositivo que a **transferência foi concluída**.

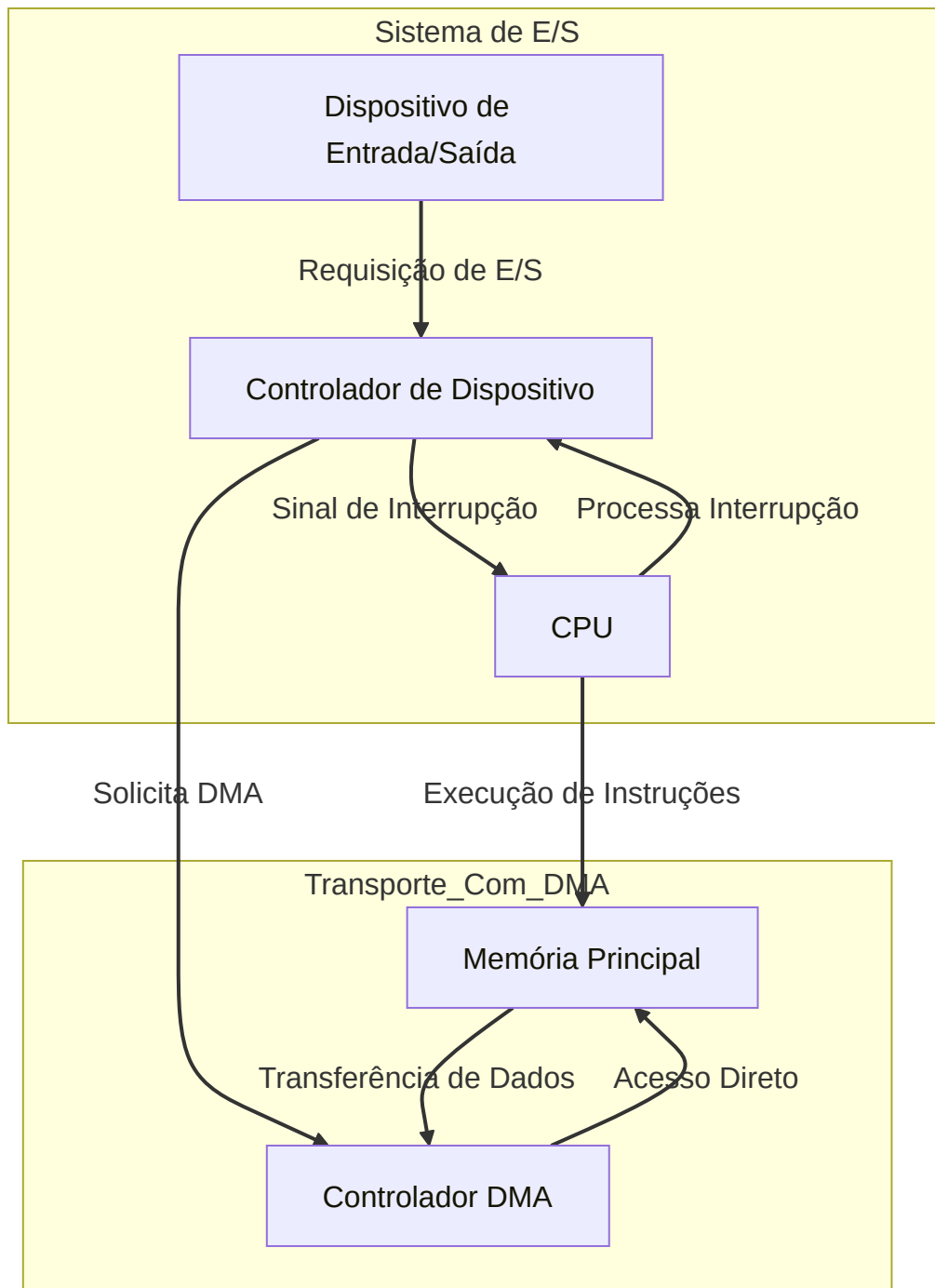
i Nesta etapa de transferência direta não ocorre intervenção da CPU, assim

apenas o controlador de dispositivo cuida dessa tarefa.

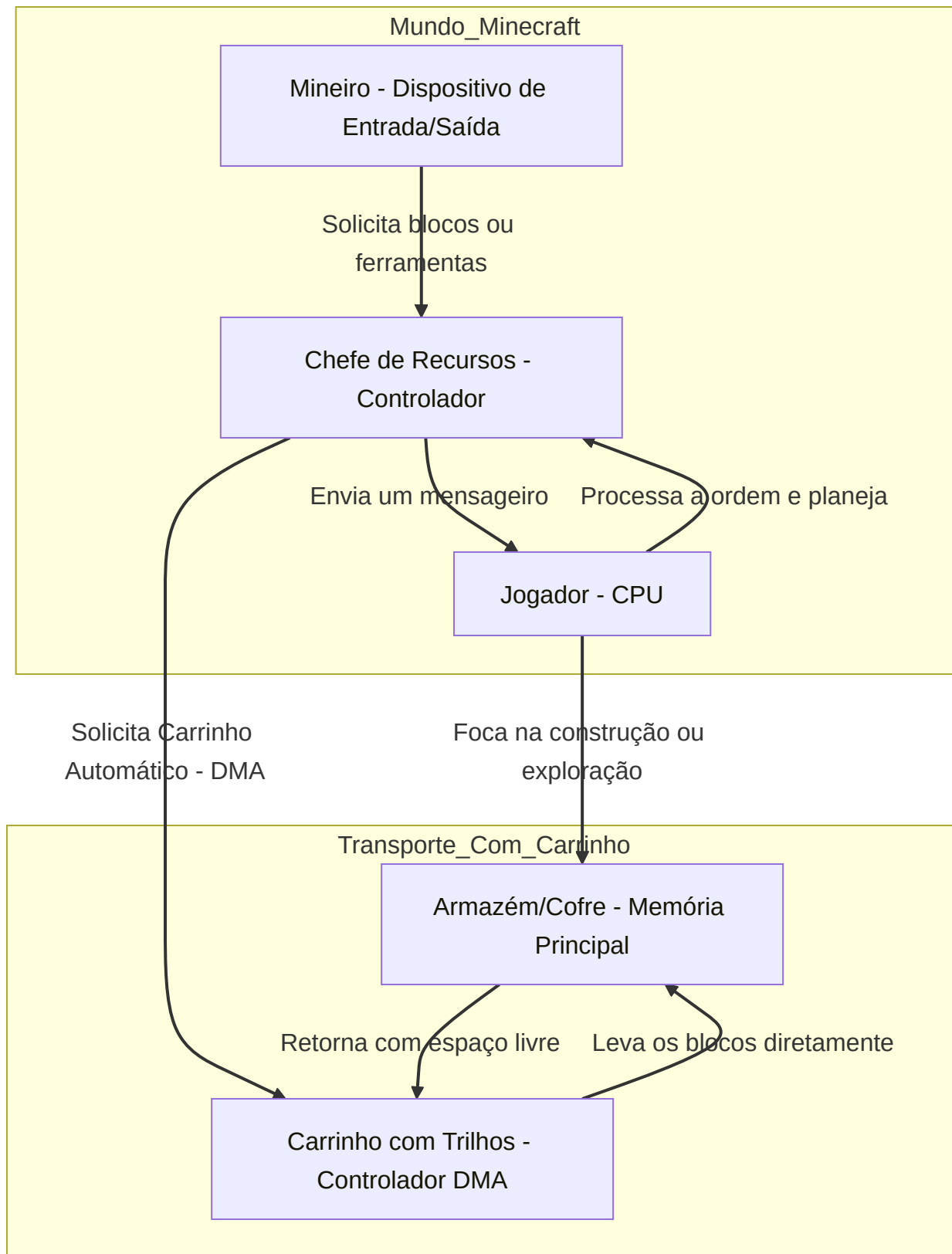
Para alguns sistemas não é utilizado essa arquitetura de barramento e sim de switch:

- Nesse tipo de sistema, os vários componentes do sistema podem interagir entre si ao mesmo tempo.
- Ao invés de competir por ciclos de um barramento compartilhado.
- Assim o **DMA** consegue ser ainda mais eficiente.

Representação da interação dos componentes num sistema:



- **Com Mineiro:**



Arquitetura do Sistema

Agora falaremos sobre a categorização dos sistemas computadorizados, que é feita com base no número de processadores que ele possui, ou seja, estamos nos referindo a computadores de uso geral.

Sistema Monoprocessador

Esses sistemas, como o nome diz, possuem um único processador e foram muito utilizados, desde PDAs até mainframes. Assim, esses sistemas contêm uma única CPU que pode realizar diversas instruções de uso geral, assim como os processos do usuário.

i A maioria dos sistemas utiliza um processador de uso específico, como, por exemplo, para processamento gráfico, com os controladores gráficos, ou nos mainframes, com os processadores de E/S.

Esses processadores específicos não executam processos do usuário e somente realizam instruções limitadas e especializadas.

- Em alguns casos, o sistema operacional controla esse componente, pois o sistema envia informações sobre sua próxima tarefa e monitora seu status.

Exemplo:

- Um processador controlador de disco recebe uma sequência de requisições da CPU principal.
- Implementa sua própria fila de disco e algoritmo de escalonamento.

i Com isso, há um alívio na carga de processamento do escalonamento de disco, que, de outra forma, seria delegado à CPU principal.

O sistema operacional não pode se comunicar diretamente com esses processadores, pois eles operam em um nível mais baixo. Um exemplo disso são os teclados, que

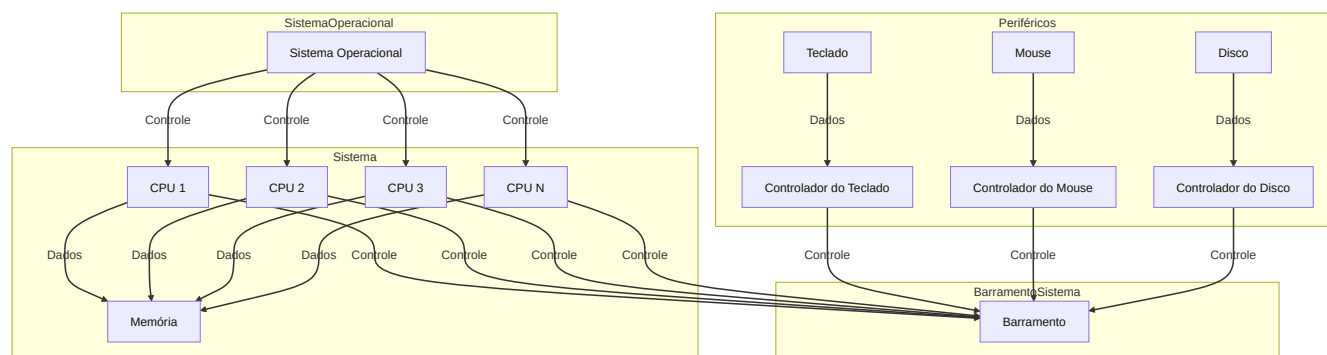
possuem um microprocessador responsável por converter os toques nas teclas em códigos que serão enviados para a CPU principal.

Assim, esses processadores realizam suas tarefas de forma anônima, pois não interagem diretamente com o sistema operacional.

Mesmo com o uso desses processadores específicos, o sistema ainda não é considerado multiprocessado.

Para que um sistema seja classificado como monoprocessador, ele deve possuir uma única CPU de uso geral. Os processadores mencionados anteriormente são de uso específico.

Diagrama



Sistema multi-processador

Esse tipo de sistema em que temos mais de um processador, de uso geral, dentro de um mesmo sistema computadorizado tem ganhado cada vez mais espaço por diversas razões o lugar dos sistema mono processador.

Os sistemas multiprocessados, ou também conhecidos como: **sistemas paralelos** (parallel system) ou **sistema fortemente acoplado** (tightly coupled system) fazem um compartilhamento perfeito de periféricos, relógio do computador, barramento do computador para vários processadores de modo que a comunicação entre eles é perfeita.

Podemos escalar **três grandes vantagens** a cerca desse tipo de arquitetura para sistemas:

- Maior vazão:

- Economia de escala:
- Maior confiabilidade:

Caching

O entendimento de **caching** é fundamental para compreender como os sistemas computadorizados funcionam.

Primeiro, pensemos que as **informações são armazenadas** em algum **dispositivo de armazenamento**, como a memória principal. Conforme são utilizadas, essas informações **são copiadas para uma memória mais rápida de modo temporário** (até mesmo mais rápida que a memória principal). Essa memória é o **cache**.

Como funciona?

Primeiro, ao precisarmos de alguma informação, **o sistema busca no cache**:

- Se a informação estiver disponível, **usamos os dados dali mesmo**.
- Caso não esteja, **o sistema irá carregá-la de uma memória mais lenta** (principal ou até mesmo de massa, secundária) e, em seguida, **fará a cópia temporária para o cache**.
 - Dessa forma, em uma **nova tentativa de acesso ao dado**, **ele poderá ser encontrado no cache**, reduzindo significativamente as consultas lentas que seriam feitas à memória principal.

Indo além, registradores responsáveis pela comunicação com a memória principal, como registradores de índice, **são gerenciados por um algoritmo de alocação e substituição de registradores, implementado pelo programador ou compilador**.

Esses **algoritmos** definem **quais informações serão mantidas nos registradores** e **quais serão enviadas para a memória principal**.

Também existem hardwares projetados para **serem implementados inteiramente no hardware**.

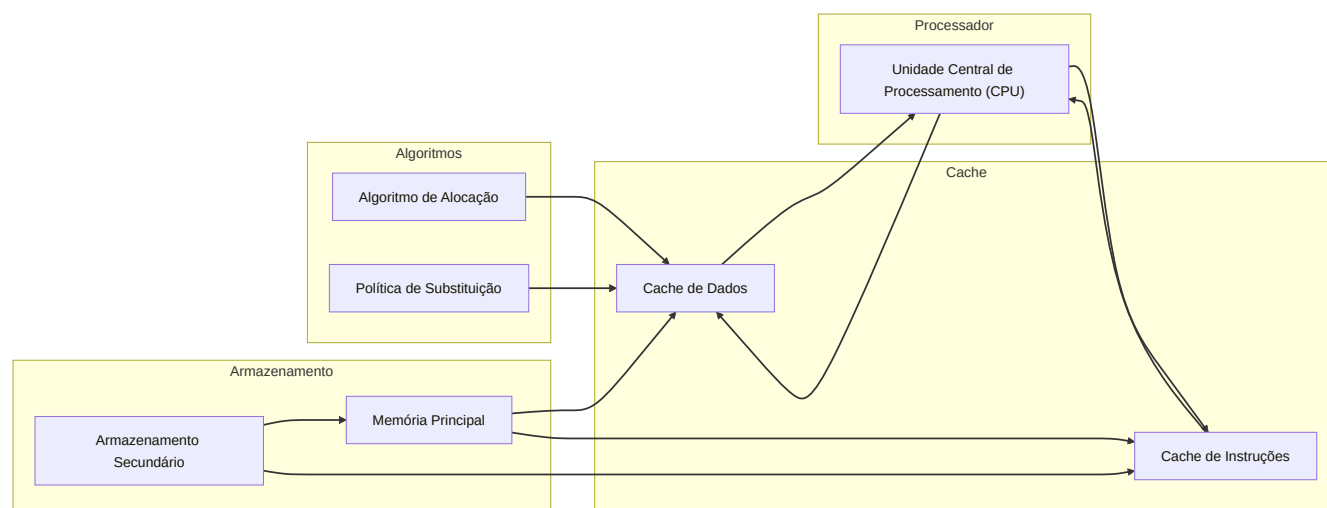
A maioria dos sistemas possui um **cache de instruções**, que serve para **armazenar as próximas instruções a serem executadas**.

Sem isso, a CPU levaria **vários ciclos** para buscar na memória principal a **próxima instrução a ser executada**.

Por essa e outras razões, a maioria dos sistemas possui vários caches de dados de alta velocidade, que estão no **topo da Hierarquia de Memórias**.

Mas, como os caches possuem um **tamanho reduzido**, o **gerenciamento de cache** se torna **fundamental**. Esse gerenciamento envolve alguns aspectos importantes, como:

- Definir o **tamanho do cache**.
- Estabelecer a **política de substituição**.



Esses fatores podem **melhorar o desempenho da memória cache**.

A **memória principal** pode ser vista como um **cache rápido para o armazenamento secundário**, pois os dados precisam ser copiados da memória secundária para a principal antes de serem utilizados.

De forma recíproca, para serem **movidos para a memória secundária**, os dados **precisam estar primeiro na memória principal**, garantindo proteção e integridade.

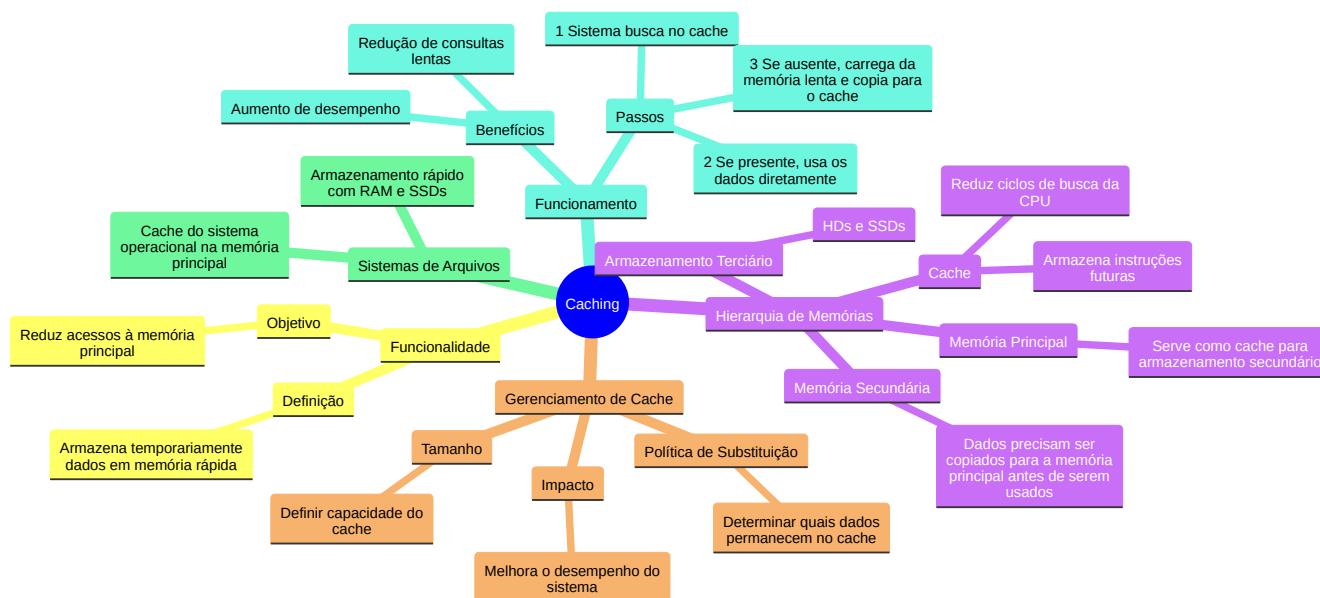
O sistema de arquivos vê os dados permanentemente gravados no armazenamento secundário de forma hierárquica, existindo *diversos níveis na hierarquia*:

- No nível mais alto -> o **sistema operacional** pode manter um **cache do sistema de arquivos na memória principal**.

Também é **possível que memórias RAM, como discos de estado sólido** (ou então discos eletrônicos de RAM), sejam usadas para **armazenamento de alta velocidade**, acessados pela **interface do sistema de arquivos**. Isso significa que a comunicação deve ser feita diretamente com o sistema de arquivos.

Atualmente, a maior parte do **armazenamento terciário** consiste em **HDs ou SSDs**.

Diagrama



Níveis e o Cache

Os **movimentos** de informações entre os **níveis da hierarquia de memórias** podem ser de dois tipos: **explícitos** e **implícitos**. Isso depende da arquitetura do **hardware** e do **software** que controla o sistema operacional.

Podemos exemplificar essa questão:

- A **transferência de dados entre a cache e a CPU e seus registradores**-> ocorre diretamente no **hardware**, sem intervenção do sistema operacional.
- A **transferência de dados do disco para a memória RAM**-> normalmente é controlada pelo **sistema operacional**.

Como, nessa estrutura hierárquica, os mesmos dados podem aparecer em diferentes níveis de armazenamento, vejamos um exemplo:

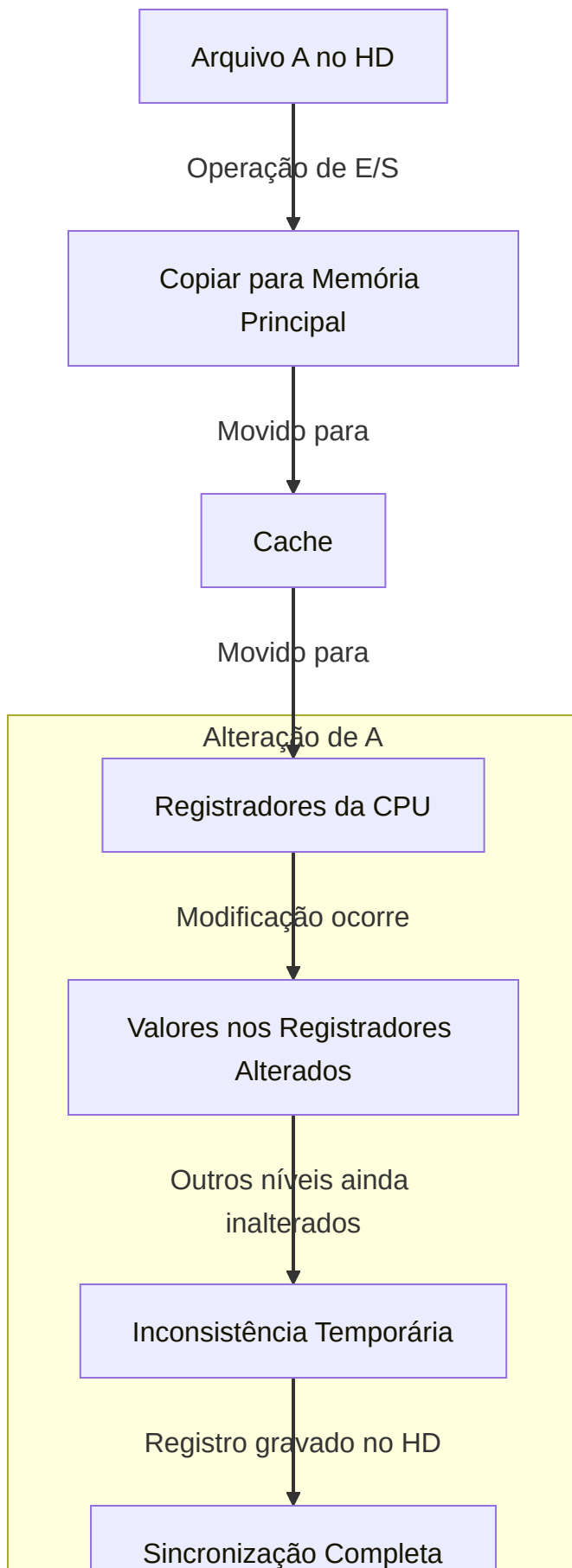
- Suponha que um texto no arquivo **A** precise ser alterado para um outro valor no arquivo **B**, que reside no HD.
- Antes da alteração, o **sistema precisa emitir uma operação de E/S para copiar o bloco de disco contendo A para a memória principal**.

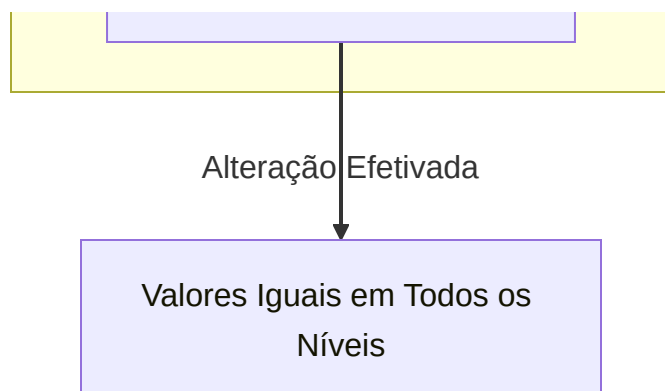
- Em seguida, o arquivo **A** será **copiado para o cache e para os registradores internos da CPU**.
- Assim, a **cópia de A** estará presente em vários níveis, conforme mostrado abaixo:



- Quando a alteração for feita nos registradores internos da CPU, os valores de **A** **serão diferentes nos outros níveis de armazenamento, que permanecerão inalterados**.
- Somente quando o **registrador gravar a mudança no disco rígido** (memória secundária), os valores nos diferentes níveis estarão **sincronizados**, tornando a alteração efetiva.

Diagrama





Threads, Cores e Sistemas Distribuídos

Em um ambiente com **apenas uma thread** (executando uma única tarefa por vez), esse esquema hierárquico funciona perfeitamente. Quando um valor é alterado nos registradores ou acessado no disco, o fluxo ocorre de forma linear, do nível mais alto ao mais baixo, sem conflitos de dados, pois todos os níveis são atualizados sequencialmente.

Entretanto, em **sistemas multicore**, onde múltiplas threads acessam simultaneamente o mesmo arquivo **A**, é necessário um **mecanismo de controle** para garantir que todos os núcleos acessem os valores atualizados de **A**. Caso contrário, podem ocorrer inconsistências, onde diferentes threads terão versões diferentes do mesmo dado, causando falhas no sistema.

Esse problema se torna ainda mais crítico em **sistemas multiprocessadores**, pois, além de registradores internos, cada processador pode possuir caches locais distintos.

Assim, **A** pode existir em diversos caches simultaneamente, e é essencial que a **versão mais recente de A** seja refletida em todos os núcleos.

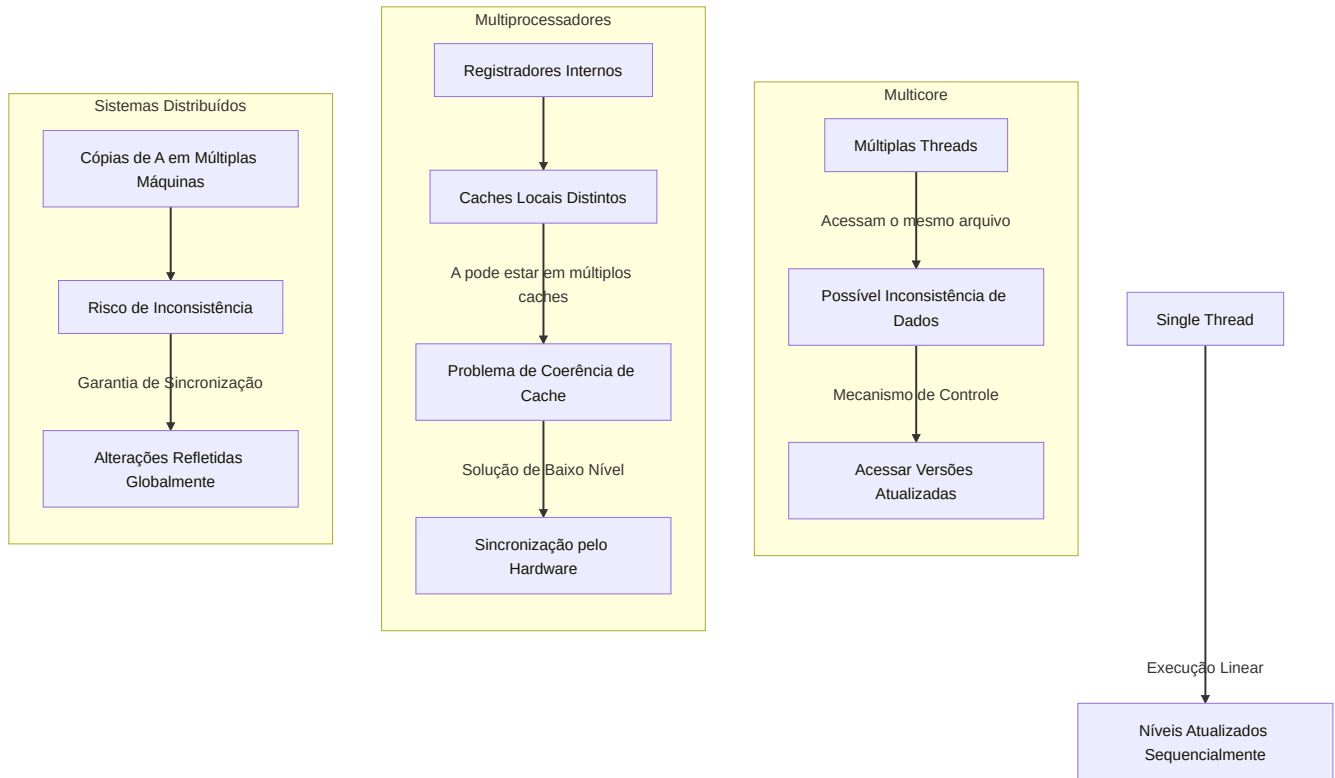
- Esse problema é conhecido como **Coerência de Cache**, e sua solução ocorre em **baixo nível**, diretamente no hardware.

A complexidade aumenta ainda mais em **sistemas distribuídos**, onde diversas cópias de **A** podem estar espalhadas por vários computadores conectados em rede.

Para evitar inconsistências, o sistema precisa garantir que **as alterações no arquivo sejam refletidas nas demais máquinas o mais rapidamente possível**.

- Existem diversas estratégias para resolver esse problema de sincronização em sistemas distribuídos.

Diagrama



Questões 1

Esta seção contém perguntas relacionadas ao assunto em questão, que podem ser usadas como referência para aprender ou revisar conhecimentos.

i Sugerimos que resolva estas questões para auxiliar em um melhor entendimento do conteúdo e assim melhor resolução dos quizzes

Pergunta 1

Dispositivos que utilizam Bluetooth e redes Wi-Fi se comunicam sem fio dentro de uma determinada área, formando uma:

- a) Rede de área metropolitana (MAN)
- b) Rede de área local (LAN)
- c) Rede de pequena área (SAN)
- d) Rede de longa distância (WAN)

Pergunta 2

O que fornece serviços adicionais para desenvolvedores ao intermediar a comunicação entre aplicativos e o sistema operacional?

- a) Virtualização
- b) Middleware
- c) Computação em nuvem
- d) Software de sistema

Pergunta 3

Os sistemas operacionais geralmente operam em dois modos distintos. Quais são eles?

- a) Modo físico e modo lógico
- b) Modo usuário e modo kernel
- c) Modo supervisor e modo secundário

d) Modo protegido e modo comum

Pergunta 4

No contexto do Linux, uma versão personalizada do sistema operacional é conhecida como:

- a) Instalação (Installation)
- b) Distribuição (Distribution)
- c) LiveCD
- d) Virtual Machine

Pergunta 5

Qual das seguintes afirmações sobre dispositivos móveis é falsa?

- a) Eles geralmente possuem menos núcleos de processamento do que desktops.
- b) O consumo de energia é um fator crítico para dispositivos móveis.
- c) Dispositivos móveis sempre possuem mais capacidade de armazenamento que laptops.
- d) Algumas funcionalidades dos dispositivos móveis não estão presentes em desktops.

Pergunta 6

Sistemas embarcados geralmente executam um sistema operacional de:

- a) Tempo real
- b) Rede
- c) Clusterizado
- d) Multiprogramação

Pergunta 7

Quais são dois fatores importantes no design da memória cache?

- a) Consumo de energia e reutilização

- b) Política de tamanho e substituição
- c) Privilégios de acesso e controle
- d) Velocidade e volatilidade

Pergunta 8

De que forma um sistema operacional pode ser comparado a um governo?

- a) Ele executa todas as funções sozinho.
- b) Ele cria um ambiente para que outros programas possam operar.
- c) Ele prioriza as necessidades individuais dos usuários.
- d) Ele raramente funciona corretamente.

Pergunta 9

Sobre instruções privilegiadas, qual das afirmações a seguir é incorreta?

- a) Elas não podem ser executadas no modo usuário.
- b) Apenas podem ser executadas no modo kernel.
- c) São usadas para gerenciamento de interrupções.
- d) Nunca representam riscos ao sistema.

Pergunta 10

A menor unidade de execução dentro de um sistema operacional é chamada de:

- a) Sistema operacional
- b) Timer
- c) Bit de modo
- d) Processo

Pergunta 11

Qual das opções abaixo é um exemplo de um programa de sistema?

- a) Navegador Web

- b) Interpretador de comandos
- c) Planilha eletrônica
- d) Software de edição de imagem

Pergunta 12

Qual chamada de sistema do Windows é equivalente à `close()` do UNIX?

- a) `CloseHandle()`
- b) `close()`
- c) `Exit()`
- d) `CloseFile()`

Pergunta 13

As chamadas de sistema são responsáveis por:

- a) Fornecer uma interface para os serviços do sistema operacional
- b) Gerenciar apenas memória virtual
- c) Proteger exclusivamente processos críticos
- d) Impedir a execução de programas de terceiros

Pergunta 14

O que define o que será feito dentro de um sistema operacional?

- a) Política
- b) Mecanismo
- c) Estratégia
- d) Interface

Pergunta 15

No Windows, a chamada de sistema `CreateFile()` é utilizada para criar arquivos. Qual a chamada equivalente no UNIX?

- a) fork()
- b) open()
- c) createfile()
- d) ioctl()

Pergunta 16

Qual das opções **não** é uma categoria principal de chamadas de sistema?

- a) Segurança
- b) Proteção
- c) Controle de processos
- d) Comunicação

Pergunta 17

Microkernels utilizam ____ para comunicação interna.

- a) Chamadas de sistema
- b) Memória compartilhada
- c) Virtualização
- d) Passagem de mensagens

Pergunta 18

Um bloco de inicialização (bootstrap loader):

- a) É composto por vários blocos de disco
- b) Pode conter múltiplos cilindros de disco
- c) Normalmente é suficiente para carregar e iniciar o sistema operacional
- d) Apenas aponta para a localização do restante do sistema de boot

Pergunta 19

Qual é o sistema operacional utilizado em dispositivos iPhone e iPad?

- a) iOS
- b) UNIX
- c) Android
- d) Mac OS X

Pergunta 20

Para um programa SYSGEN de um sistema operacional, qual das informações abaixo é **menos útil**?

- a) Quantidade de memória disponível
- b) Configurações como tamanho de buffer e algoritmo de escalonamento de CPU
- c) Lista de aplicativos a serem instalados
- d) Arquitetura da CPU

Pergunta 21

Um sistema operacional pode ser classificado como um:

- a) Gerenciador de recursos
- b) Programa de usuário
- c) Firmware de inicialização
- d) Simulador de processos

Pergunta 22

O que é multitarefa em um sistema operacional?

- a) A execução de um único processo por vez
- b) A capacidade de executar múltiplos processos simultaneamente
- c) A execução de tarefas em tempo real sem atraso
- d) A execução de comandos administrativos pelo usuário

Pergunta 23

Qual das seguintes opções descreve um **hipervisor**?

- a) Um software responsável pela virtualização de hardware
- b) Um protocolo de comunicação em redes
- c) Um sistema operacional para servidores
- d) Um método de gerenciamento de arquivos

Pergunta 24

O que acontece quando um processo em execução tenta acessar uma página de memória que não está carregada?

- a) Ele é imediatamente encerrado pelo sistema operacional
- b) O sistema operacional gera uma interrupção e carrega a página necessária
- c) O sistema operacional ignora a solicitação
- d) O processo entra em um estado de loop infinito

Pergunta 25

O que é um deadlock em um sistema operacional?

- a) Um erro crítico no kernel
- b) Um estado onde dois ou mais processos ficam bloqueados indefinidamente
- c) Um método de gerenciamento de arquivos
- d) Uma forma de escalonamento de processos


Prática 1

Nessa prática vamos criar e configurar máquinas virtuais.

Conhecendo ferramentas

Para fazer a virtualização de sistemas operacionais temos alguns programas para isso:

- **Windows:**
 - VirtualBox
 - VMWare
 - Paralles Desktop
- **Linux:**
 - VirtualBox
 - Gnome Boxes
 - Virtual Machine Manager

 Há casos que o VirtualBox falha por sabe se lá quais razões então é melhor ter mais de uma opção na manga.

Agora vamos

Requisitos dos sistemas

Confira os requisitos dos sistemas operacionais que será usado:

- Windows 10 (<https://support.microsoft.com/pt-br/windows/requisitos-do-sistema-do-windows-10-6d4e9a79-66bf-7950-467c-795cf0386715>)
- Ubuntu Desktop (<https://ubuntu.com/server/docs/system-requirements>)

Instalando ferramenta

Vamos usar o VirtualBox.

- Para Windows (<https://download.virtualbox.org/virtualbox/7.1.6/VirtualBox-7.1.6-167084-Win.exe>)
- Para Linux (https://www.virtualbox.org/wiki/Linux_Downloads)

Instalar ISOs (Windows e Ubuntu)

- Windows (<https://www.microsoft.com/pt-br/software-download/windows10ISO>)

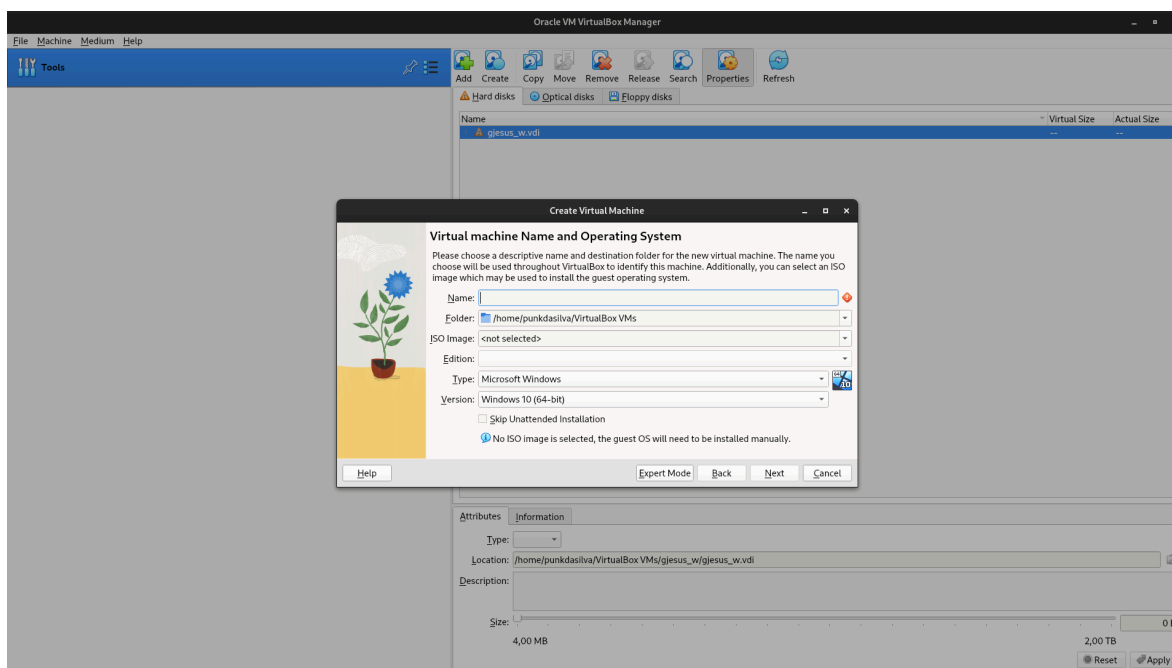
i Na ISO oficial do Windows, o link a cima, vem com todas as versões.

- Ubuntu Desktop (<https://ubuntu.com/download/desktop>)

Criando Maquinas Virtuais

Windows

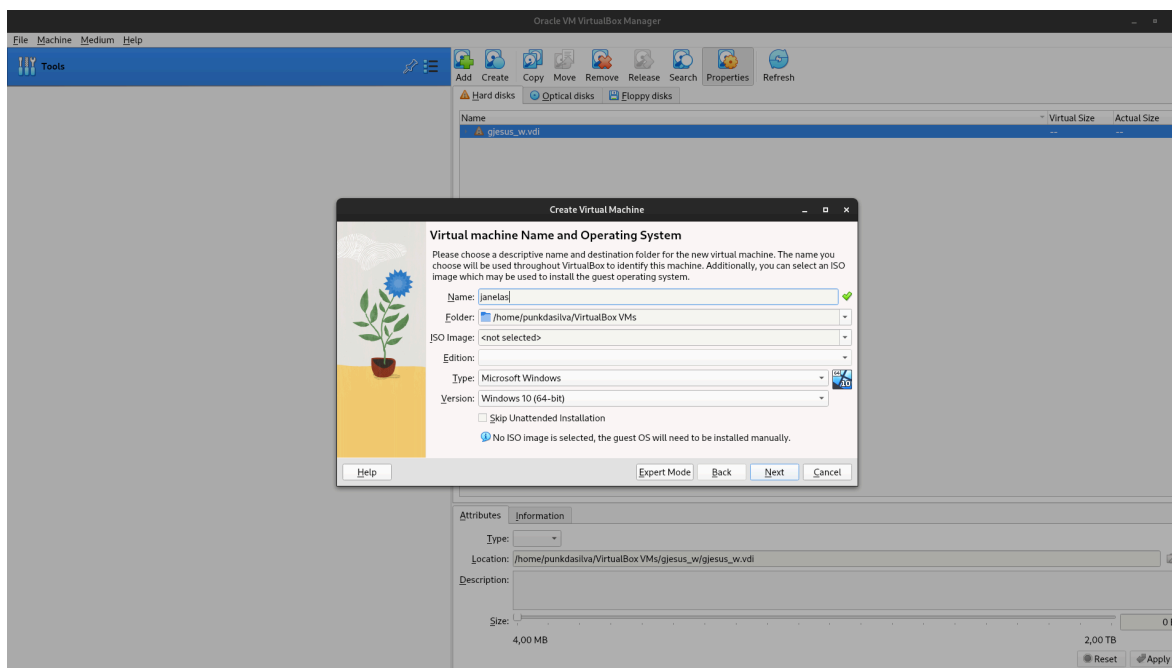
1. Com o VirtualBox aberto aperte: **CTRL** + **N**



Criando uma nova Maquina

i Você pode acessar a opção também por: Machine > Add

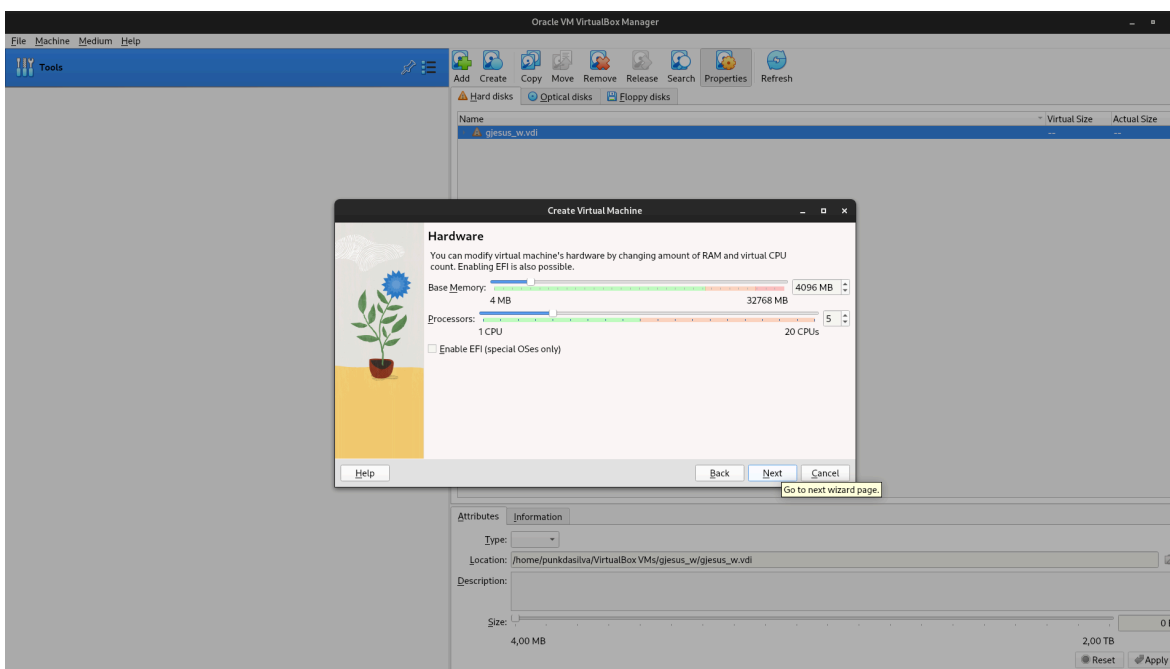
2. Definir nome, sistema, e a versão:



Definindo configurações a VM

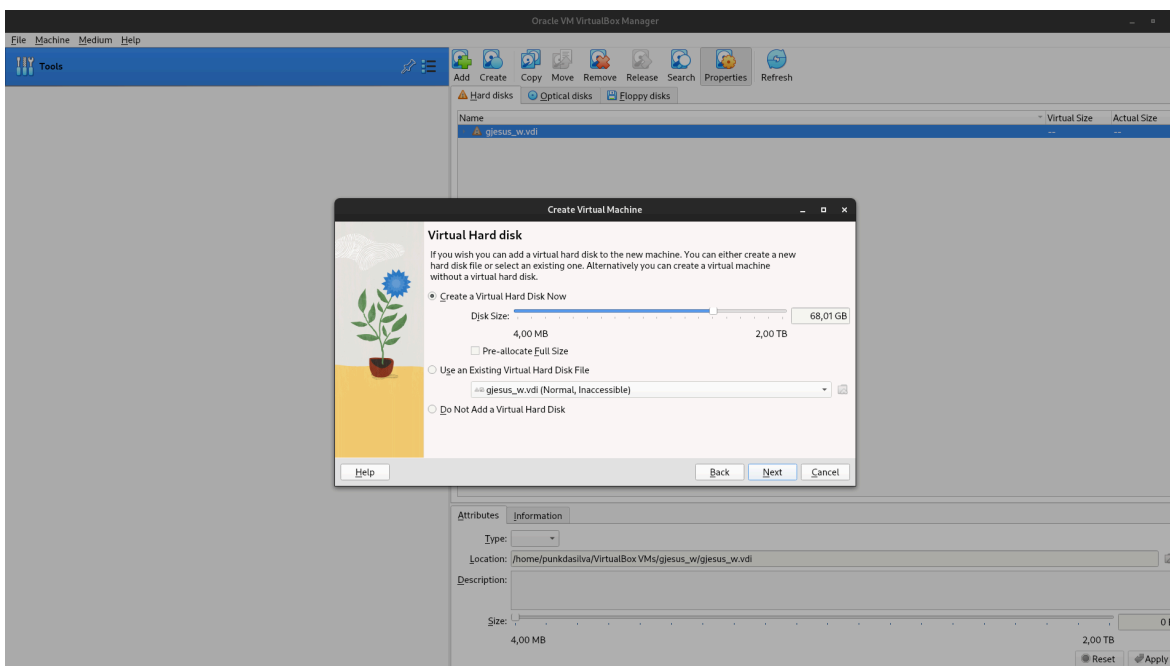
3. Agora vamos definir a memória ram, é bom deixarmos no mínimo 4GB

i Atente-se que computadores não são bons lidando com números ímpares.



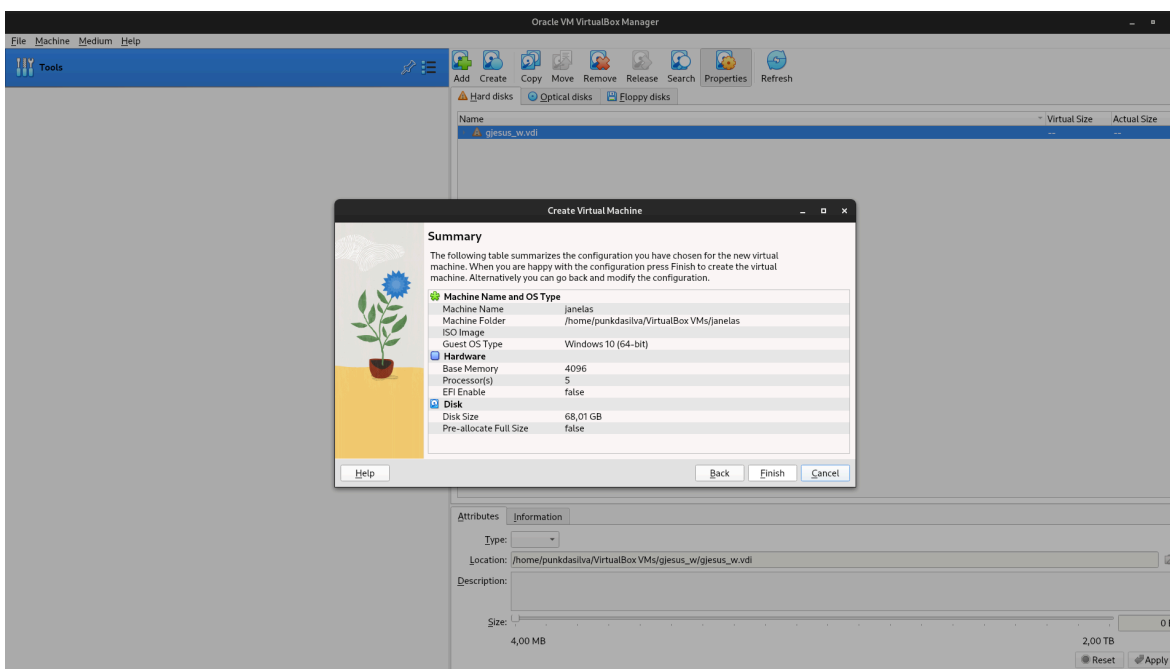
Definindo para a VM memoria RAM

4. Agora definimos o espaço de armazenamento, hard disk:



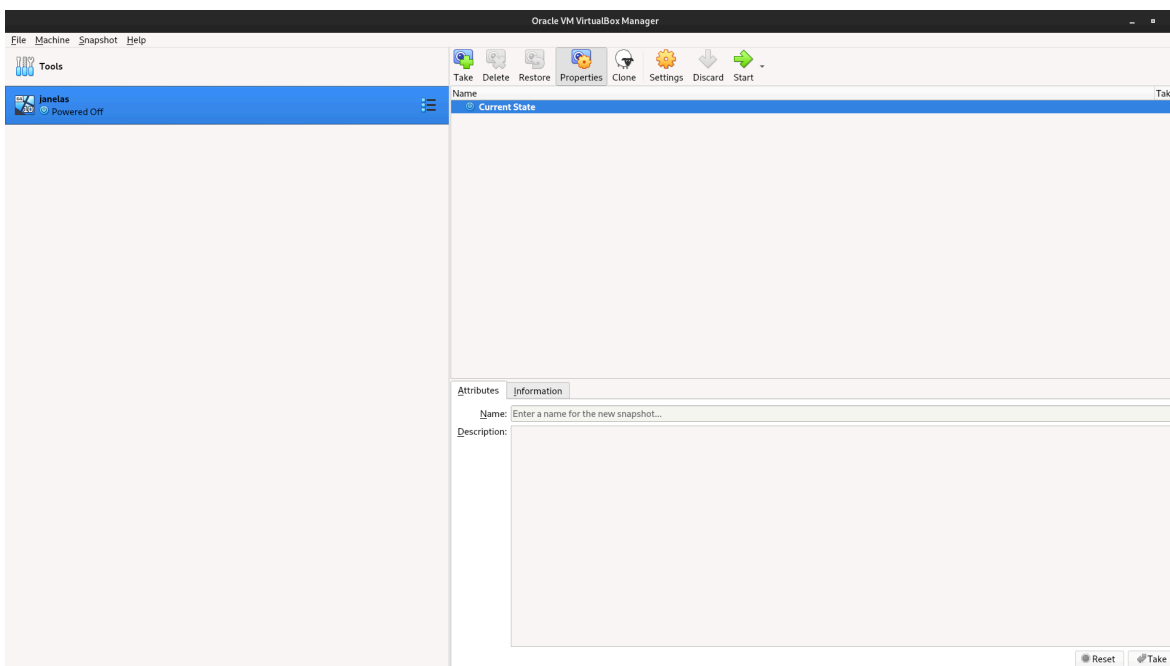
Definindo Hard Disk

5. Com tudo criado basta ir em **Finish**:



Configurações iniciais na primeira VM feitas

6. Temos então nossa primeira maquina virtual criada:



Maquina virtual criada com sucesso

Ubuntu

Logar nas VMs assim que criadas

Configurando VMs para os SOs

Logar nas VMs

Hardware

Start typing here...

Software

Start typing here...

Respostas

#	Resposta Correta
1	b
2	b
3	b
4	b
5	c
6	b
7	d
8	b
9	d
10	d
11	d
12	a
13	d
14	d
15	b

16	a
17	d
18	d
19	a
20	c
21	a
22	b
23	a
24	b
25	b