

APPLE FILE SYSTEM - APFS

TI0146 - SISTEMAS OPERACIONAIS

SUMÁRIO

Esquema Geral

03

Implementação de arquivos

04

Implementação de diretórios

05

Arquivos compartilhados

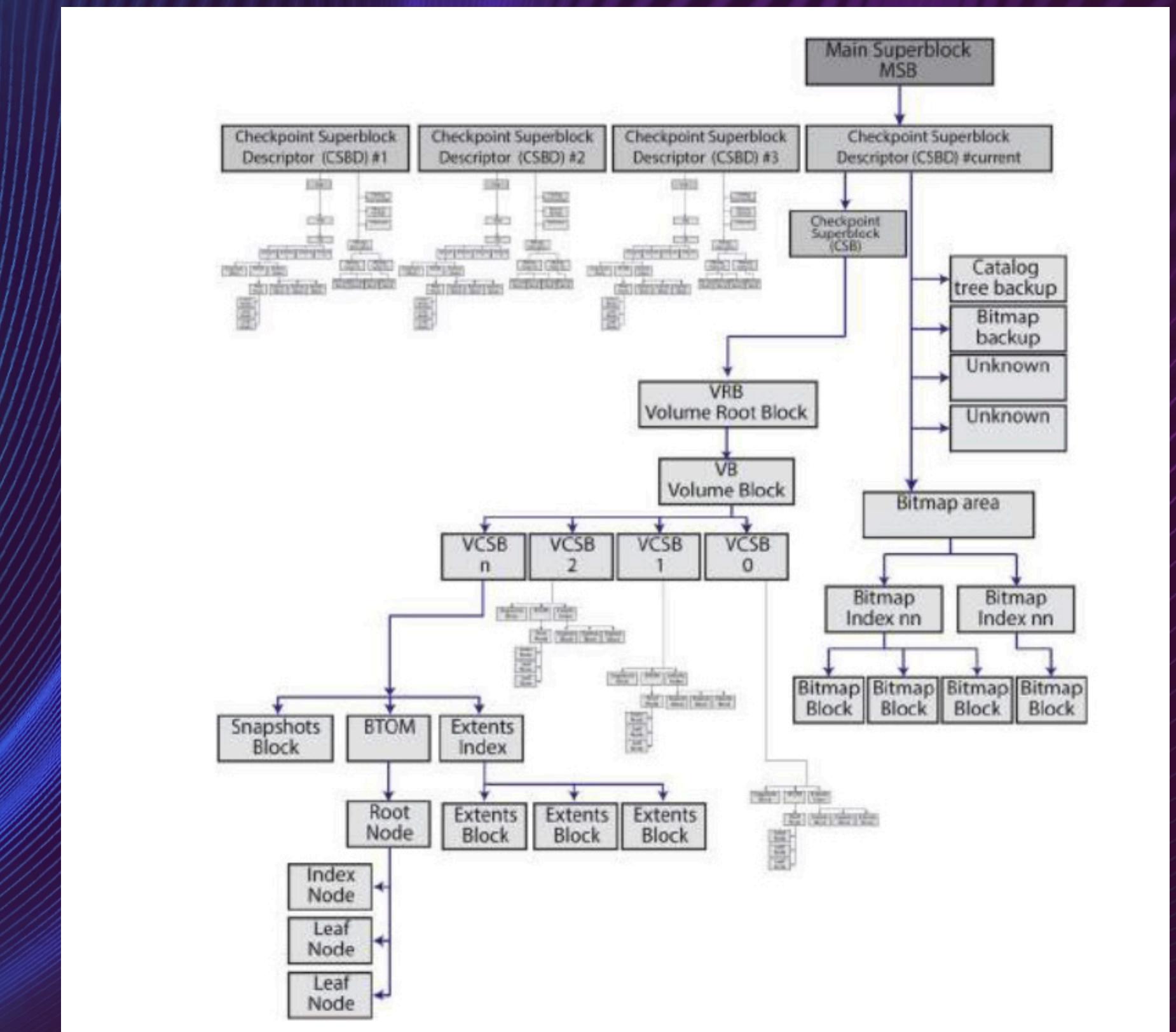
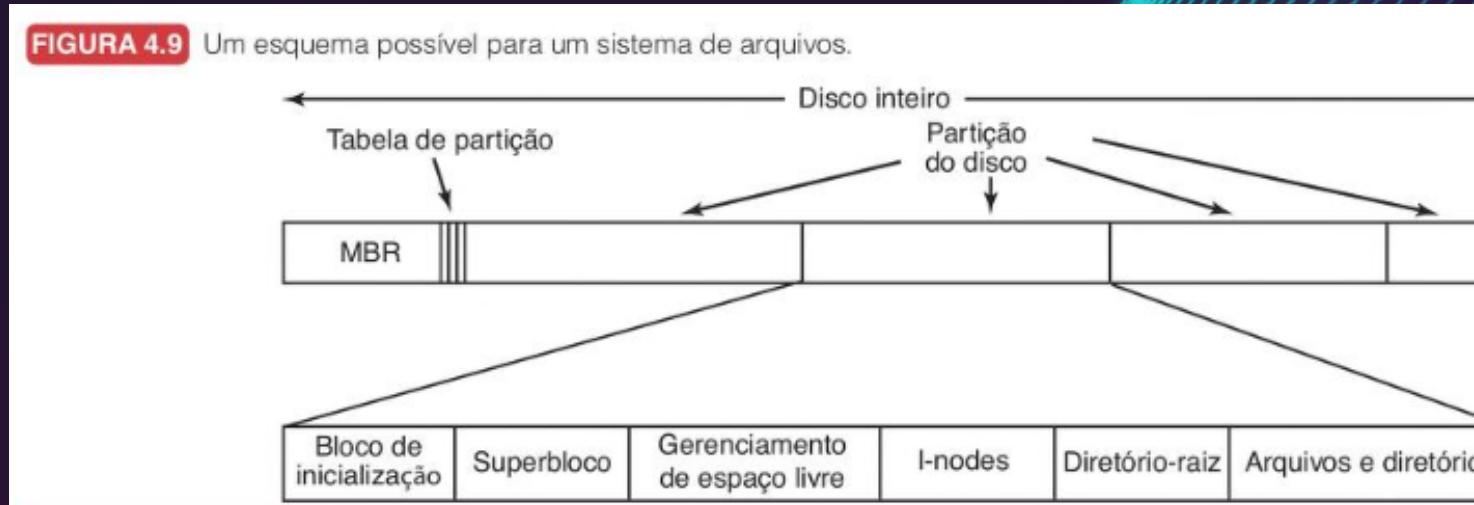
07

Gerenciamento de espaço livre

09

ESQUEMA GERAL

- Bloco de inicialização: não possui bloco de boot dedicado, mas cada volume do container possui um arquivo “Boot.efi” e as informações de boot necessárias.
- Superbloco: possui no início a estrutura de “superbloco de container” com as informações sobre os volumes e espaço livre.
- Gerenciamento de espaço livre: compartilhamento dinâmico entre volumes conforme necessário usando B-tree’s(árvores平衡adas).
- Blocos de metadados: aloca os metadados nas B-tree’s, de forma semelhante aos inodes.
- Blocos de dados: usa alocações “extent-based”, na qual aloca dinamicamente blocos contíguos para um arquivo conforme necessário.



IMPLEMENTAÇÃO DE ARQUIVOS

B-TREE'S PARA
METADADOS E
EXTENTS

COPY-ON-WRITE
(COW)

SNAPSHOTS

CRIPTOGRAFIA
NATIVA

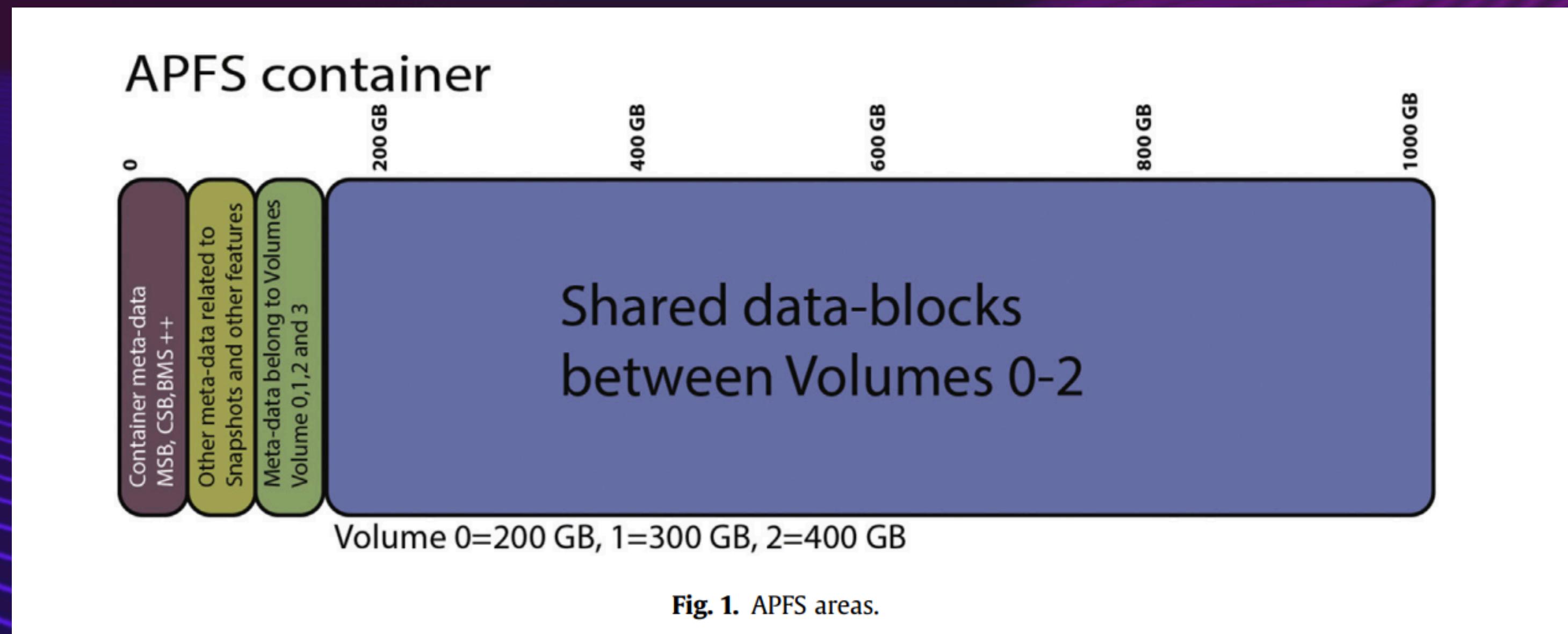
IMPLEMENTAÇÃO DE DIRETÓRIOS

HIERARQUIA DE
DIRETÓRIOS

B-TREES DE
DIRETÓRIOS

MAPEAMENTO DE
ARQUIVOS

IMPLEMENTAÇÃO DE DIRETÓRIOS



ARQUIVOS COMPARTILHADOS (LINKS)

HARD LINKS

LINKS
SIMBÓLICOS

CLONAGEM DE
ARQUIVOS

ARQUIVOS COMPARTILHADOS (LINKS)

| Offset | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|--------|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0000 | 0B | 3D | 7F | 89 | C6 | C2 | 5A | 1C | 0F | 04 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0010 | 13 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 02 | 00 | 00 | 00 | 0E | 00 | 00 | 00 |
| 0020 | 01 | 00 | 01 | 00 | 0F | 00 | 00 | 00 | 00 | 00 | 80 | 00 | 98 | 01 | 10 | 0D |
| 0030 | 28 | 00 | 20 | 00 | FF | FF | 00 | 00 | 00 | 00 | 18 | 00 | 08 | 00 | 08 | 00 |
| 0040 | 50 | 01 | 19 | 00 | 70 | 00 | 08 | 00 | 18 | 01 | 19 | 00 | 10 | 00 | 08 | 00 |
| 0050 | 70 | 01 | 27 | 00 | 18 | 00 | 08 | 00 | 40 | 00 | 0A | 00 | 20 | 00 | 08 | 00 |
| 0060 | 50 | 00 | 27 | 00 | 28 | 00 | 08 | 00 | 78 | 00 | 0A | 00 | 30 | 00 | 08 | 00 |
| 0070 | 88 | 00 | 12 | 00 | 38 | 00 | 08 | 00 | 18 | 00 | 0A | 00 | 40 | 00 | 08 | 00 |
| 0080 | D0 | 00 | 27 | 00 | 48 | 00 | 08 | 00 | F8 | 00 | 0A | 00 | 50 | 00 | 08 | 00 |
| 0090 | B8 | 00 | 12 | 00 | 58 | 00 | 08 | 00 | A0 | 00 | 0A | 00 | 60 | 00 | 08 | 00 |
| 00A0 | 38 | 01 | 12 | 00 | 68 | 00 | 08 | 00 | 08 | 01 | 0A | 00 | 78 | 00 | 08 | 00 |
| 00B0 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | | | | | | | | |
| 00B0 | | | | | | | | | 01 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 00C0 | 09 | 01 | 0C | 00 | 70 | 72 | 69 | 76 | 61 | 74 | 65 | 2D | 64 | 69 | 72 | 00 |
| 00D0 | 35 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 06 | 01 | 00 | 00 | 00 | 00 | 00 | 00 |
| 00E0 | 38 | 00 | 10 | 00 | 00 | 00 | 00 | 00 | 06 | 01 | 00 | 00 | 00 | 00 | 00 | 00 |
| 01F0 | 51 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 08 | 01 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0200 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 02 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0210 | 09 | 01 | 0D | 00 | 43 | 49 | 4D | 47 | 30 | 31 | 35 | 32 | 2E | 4A | 50 | 47 |
| 0220 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 18 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0230 | 04 | 01 | 1B | 00 | 63 | 6F | 6D | 2E | 76 | 6D | 77 | 61 | 72 | 65 | 2E | 62 |
| 0240 | 61 | 63 | 6B | 75 | 70 | 52 | 65 | 65 | 6E | 61 | 62 | 6C | 65 | 64 | 00 | 00 |
| 0250 | ... | | | | | | | | | | | | | | | |
| 0F60 | 40 | 04 | 00 | 00 | 00 | 00 | 00 | 00 | 3D | 04 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0F70 | 3C | 04 | 00 | 00 | 00 | 00 | 00 | 00 | 39 | 04 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0F80 | 35 | 04 | 00 | 00 | 00 | 00 | 00 | 00 | 31 | 04 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0F90 | 2C | 04 | 00 | 00 | 00 | 00 | 00 | 00 | 25 | 04 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0FA0 | 22 | 04 | 00 | 00 | 00 | 00 | 00 | 00 | 1F | 04 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0FB0 | 1C | 04 | 00 | 00 | 00 | 00 | 00 | 00 | 19 | 04 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0FC0 | 16 | 04 | 00 | 00 | 00 | 00 | 00 | 00 | 12 | 04 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0FD0 | 13 | 04 | 00 | 00 | 00 | 00 | 00 | 00 | 10 | 04 | 00 | 00 | 00 | 10 | 00 | 00 |
| 0FE0 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 27 | 00 | 00 | 00 | C0 | 00 | 00 | 00 |
| 0FF0 | 9B | 01 | 00 | 00 | 00 | 00 | 00 | 00 | 10 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |

private-dir
Q
CIMG0152.JPG
com.vmware.b
ackupReenabled
@ =
< 9
5 1
, %
' Á

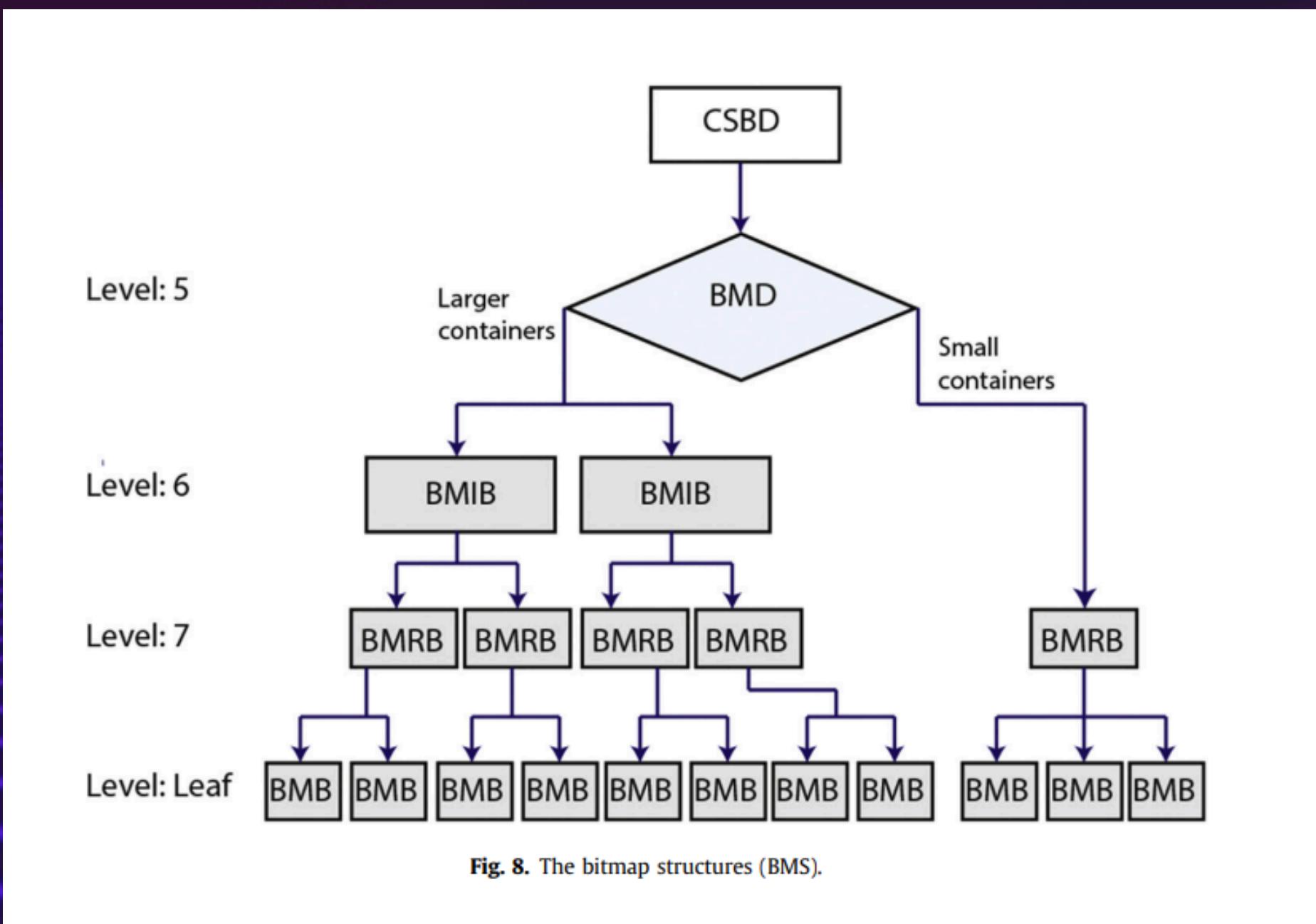
Fig. 17. The B-tree Catalog root node (BTRN).

GERENCIAMENTO DE ESPAÇO LIVRE

COMPARTILHAMENTO
DE ESPAÇO

GERENCIAMENTO
COM B-TREES

GERENCIAMENTO DE ESPAÇO LIVRE



Bootloader

NUCLEO-L476RG

Álvaro Galvão - 537649

Clara Salles - 511201

Danilo Alencar - 508706

Gabriel Moreira - 536708

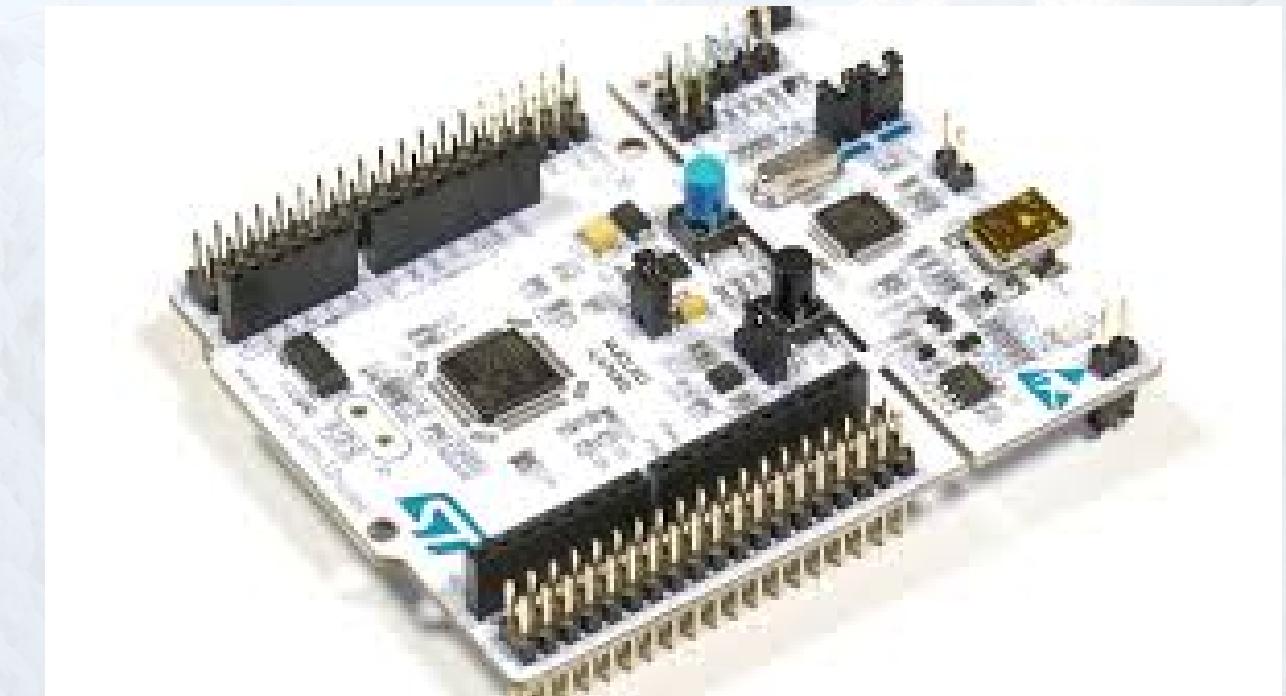
Luís Gustavo - 536164

Sumário

- Visão Geral
- Aplicações
- Como Funciona o Bootloader?
- Demonstração

Visão Geral

- Bootloader para a STM32 NUCLEO-L476RG
- Seleciona entre duas aplicações na memória FLASH do MCU
- Caso não encontre aplicação na região de memória, manda um sinal de SOS



Aplicações

- Aplicação 1:

Usa o freeRTOS para executar uma task que pisca o user LED da placa a cada 500 ms e manda um caractere através do Single Wire Output (SWO). Através do STMCubeIDE, é possível visualizar a mensagem enviada com o Single Wire Viewer.

```
/* USER CODE BEGIN 4 */
void Task_action(char message) {
    ITM_SendChar(message);
    ITM_SendChar('\n');
}

/* USER CODE END 4 */

/* USER CODE BEGIN Header_StartDefaultTask */
/**
 * @brief Function implementing the defaultTask thread.
 * @param argument: Not used
 * @retval None
 */
/* USER CODE END Header_StartDefaultTask */
void StartDefaultTask(void *argument)
{
    /* USER CODE BEGIN 5 */
    /* Infinite loop */
    for(;;)
    {
        Task_action('A');
        HAL_GPIO_TogglePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin);
        osDelay(500);
    }
    /* USER CODE END 5 */
}
```

Aplicações

- Aplicação 2:

A segunda aplicação é um simples Toggle Pin, feito em bare metal, que pisca o user LED da placa a cada 100 ms.

```
/* Initialize all configured peripherals */
MX_GPIO_Init();
/* USER CODE BEGIN 2 */

/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
    HAL_Delay(100);
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
}
/* USER CODE END 3 */

}

/**
 * @brief System Clock Configuration
 * @retval None
 */
```

Como Funciona o Bootloader?

O Bootloader ocupa o primeiro bloco de memória Flash disponível do MCU. Ele fica responsável por gerenciar o deslocamento dentre as aplicações do sistema.

```
void jumpToApp(const uint32_t address)
{
    const JumpStruct* vector_p = (JumpStruct*)address;

    deinitEverything();
    SCB->VTOR = address; // address é o endereço base da aplicação FreeRTOS
    // Atualiza o main stack pointer e faz o jump para a função
    asm("msr msp, %0; bx %1;" : : "r"(vector_p->stack_addr), "r"(vector_p->func_p));
}
```

Como Funciona o Bootloader?

```
void bootloaderInit()
{
    int App = 0;
    BootloaderMode bootloaderMode = JumpMode;

    if(HAL_GPIO_ReadPin(Blue_Button_GPIO_Port, Blue_Button_Pin) == GPIO_PIN_RESET)
        App = App1;
    else
        App = App2;

    if(bootloaderMode == JumpMode)
    {
        if(App == App1)
        {
            //Check if the application is there
            uint8_t emptyCellCount = 0;
            for(uint8_t i=0; i<10; i++)
            {
                if(readWord(APP1_START + (i*4)) == -1)
                    emptyCellCount++;
            }

            if(emptyCellCount != 10)
                jumpToApp(APP1_START);
            else
                errorBlink();
        }
    }
}
```

```
else
{
    //Check if the application is there
    uint8_t emptyCellCount = 0;
    for(uint8_t i=0; i<10; i++)
    {
        if(readWord(APP2_START + (i*4)) == -1)
            emptyCellCount++;
    }

    if(emptyCellCount != 10)
        jumpToApp(APP2_START);
    else
        errorBlink();
}
```

Demonstração