ISCAS NIST

循序渐进，学习开发一个 RISC-V 上的操作系统

第 8 章 内存管理

汪辰

2021/4

本章内容

- 对内存进一步的管理，实现动态的分配和释放。
- 实现 Page 级别的内存分配和释放。

参考

- 【参考 1】： GNU ld manual on linker scripts: https://sourceware.org/binutils/docs/ld/Scripts.html#Scripts
- 【参考 2】: The RISC-V Instruction Set Manual, Volume II: Privileged Architecture，Document Version 20190608-Priv-MSU-Ratified

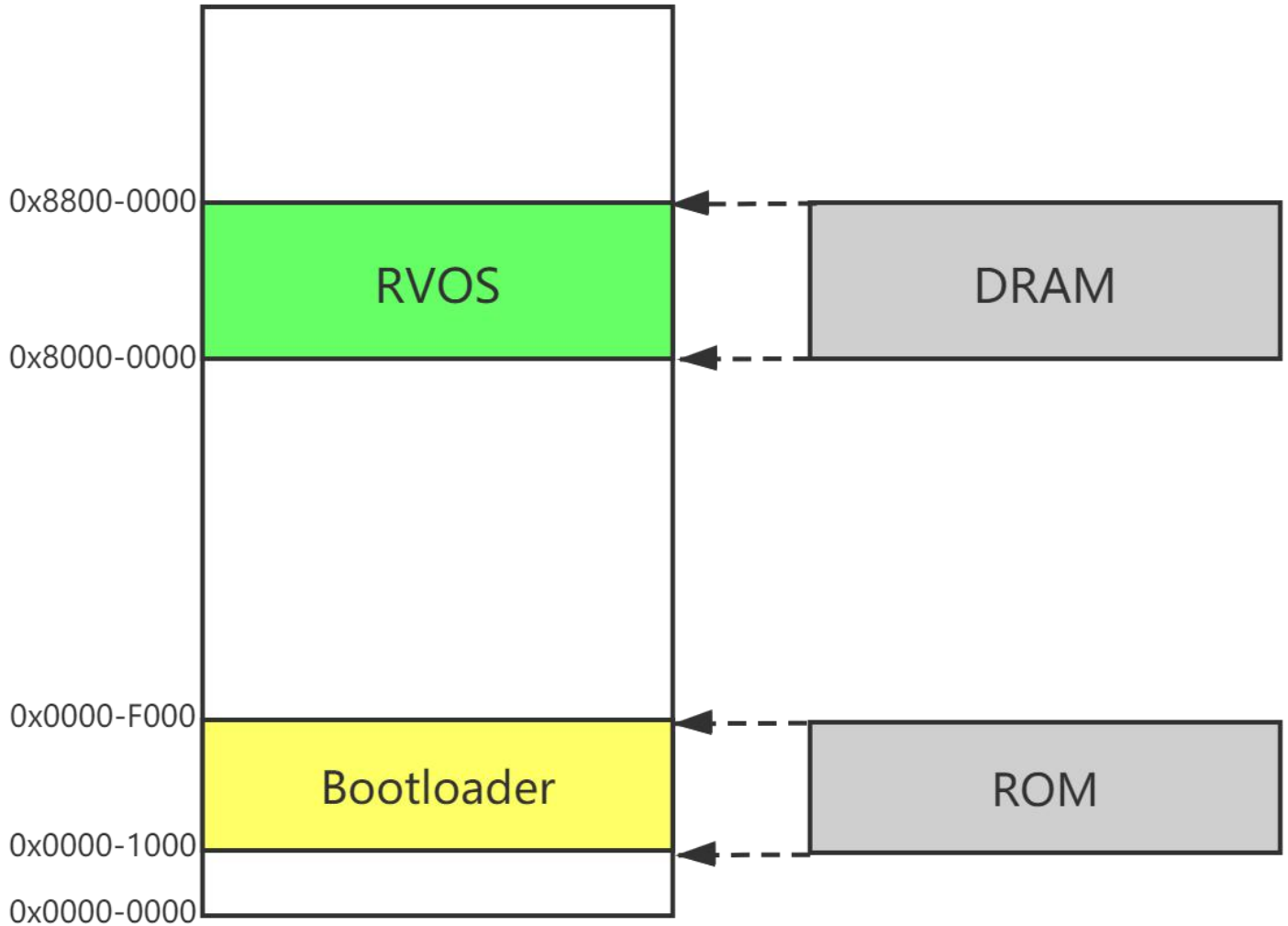
本章内容

- 对内存进一步的管理，实现动态的分配和释放。
- 实现 Page 级别的内存分配和释放。

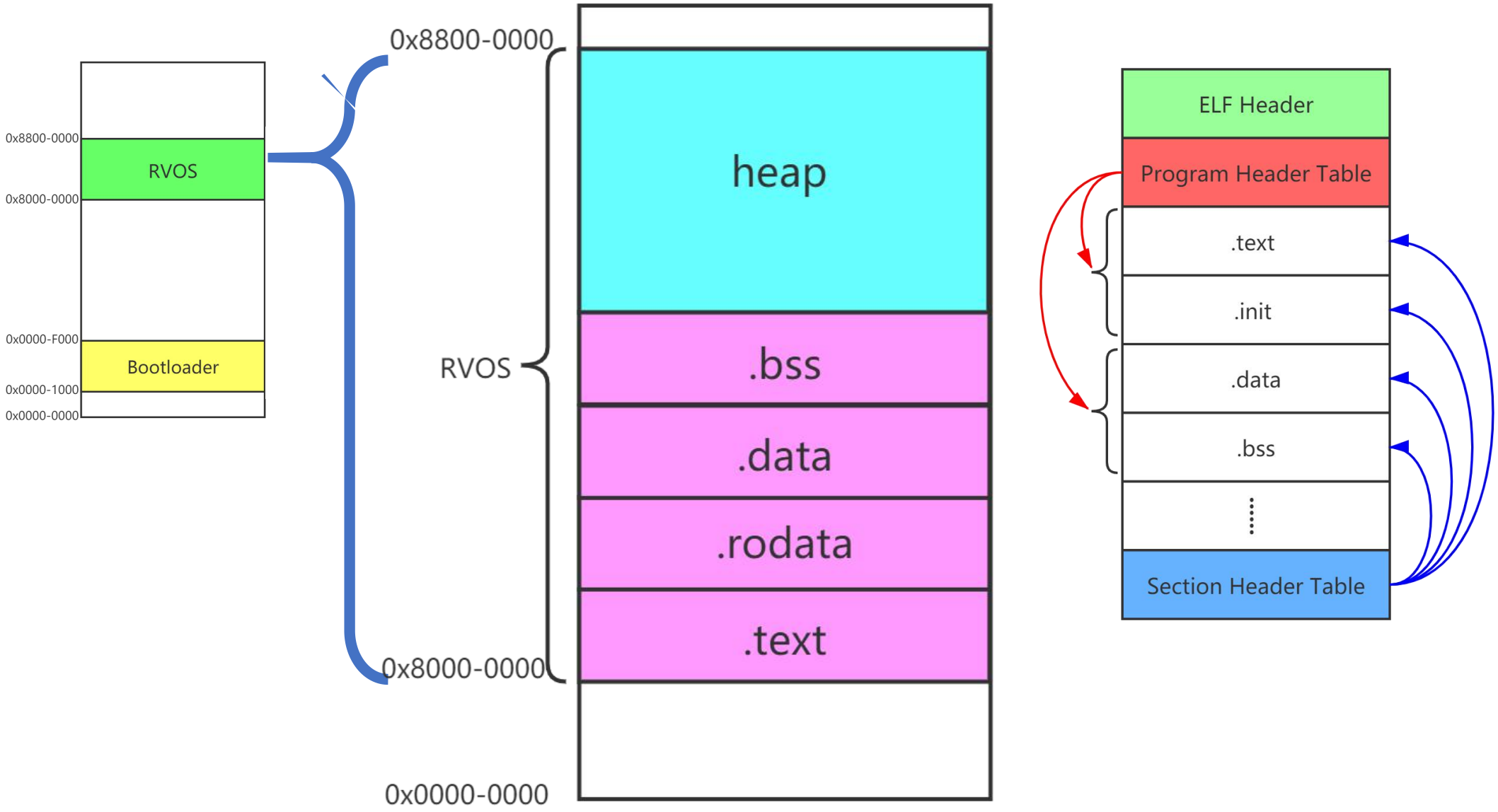
内存管理分类

- 自动管理内存 - 栈（stack）
- 静态内存 - 全局变量/静态变量
- 动态管理内存 - 堆（heap）

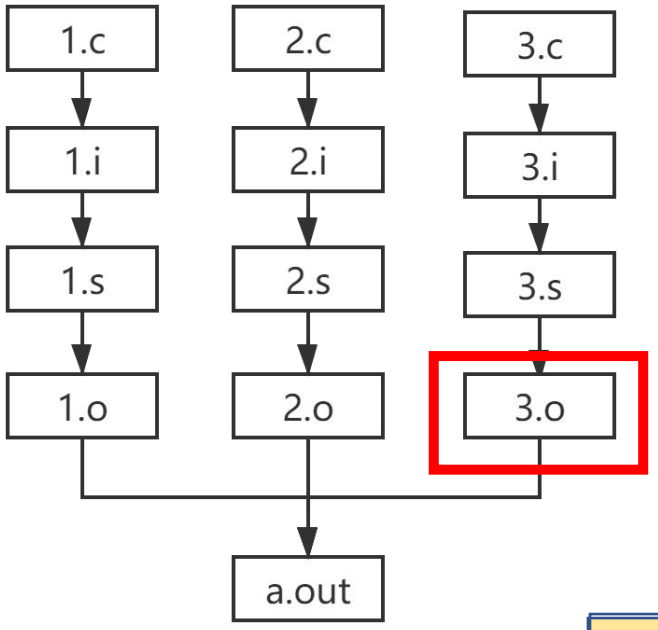
内存映射表（Memory Map）

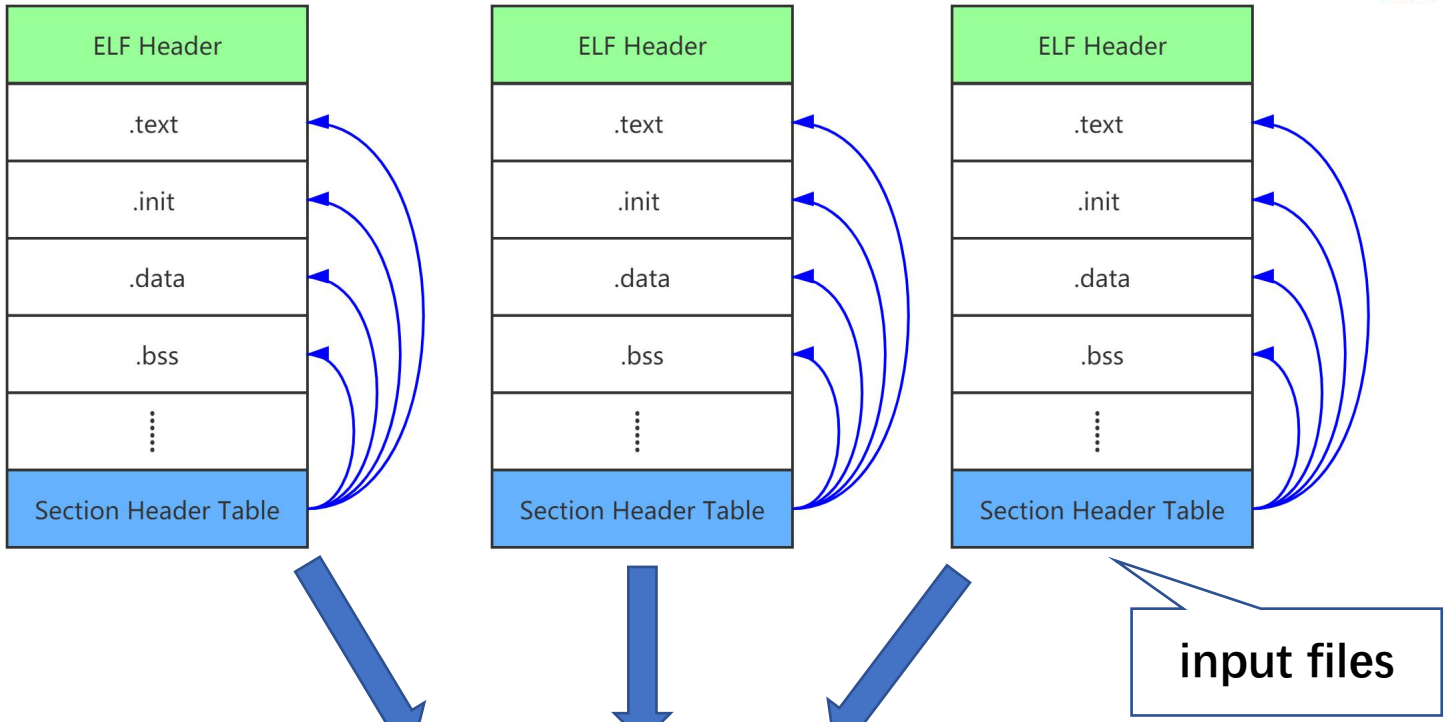


内存映射表（Memory Map）

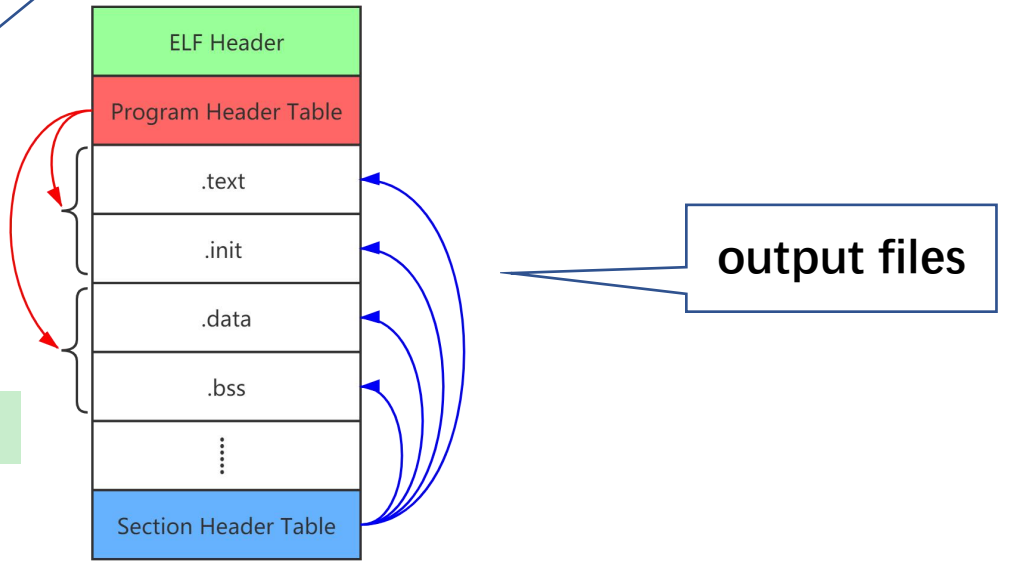


Linker Script 链接脚本





Linker Script (os.ld)




```
${CC} $ ${CC} $(CFLAGS) -T os.ld -o os.elf $^
```

Linker Script 链接脚本

- GNU ld 使用 Linker Script 来描述和控制链接过程。
- Linker Script 是简单的纯文本文件，采用特定的脚本描述语言编写。
- 每个 Linker Script 中包含有多条命令（Command)
- 注释采用 “/*” 和 “*/” 括起来
- gcc -T os.ld ……

更多语法见 【参考1】

```
SECTIONS
{
  . = 0x10000;
  .text : { *(.text) }
  . = 0x8000000;
  .data : { *(.data) }
  .bss : { *(.bss) }
}
```

Linker Script 链接脚本

➢ ENTRY

语法	ENTRY(symbol)
例子	ENTRY(_start)

- ENTRY 命令用于设置“入口点 (entry point)”，即程序中执行的第一条指令。
- ENTRY 命令的参数是一个符号（symbol）的名称。

Linker Script 链接脚本

➢ OUTPUT_ARCH

语法	OUTPUT_ARCH(bfdarch)
例子	OUTPUT_ARCH("riscv")

- **OUTPUT_ARCH 命令指定输出文件所适用的计算机体系架构。**

Linker Script 链接脚本

➢ MEMORY

语法	MEMORY
	{
	name [(attr)] : ORIGIN = origin, LENGTH = len
	.....
	}

例子	MEMORY
	{
	rom (rx) : ORIGIN = 0, LENGTH = 256K
	ram (!rx) : org = 0x40000000, l = 4M
	}

- **MEMORY 用于描述目标机器上内存区域的位置、大小和相关**

Linker Script 链接脚本

➢ SECTIONS

语法	例子

SECTIONS
{
sections-command
sections-command
……
}
	SECTIONS
	{
	. = 0x10000;
	.text : { *(.text) }
	. = 0x8000000;
	.data : { *(.data) }
	.bss : { *(.bss) }
	} >ram

- **SECTIONS 告诉链接器如何将 input sections 映射到 output sections，以及如何将 output sections 放置在内存中。**
- **section-command 除了可以是对 out section 的描述外还可以是符号赋值命令等其他形式。**

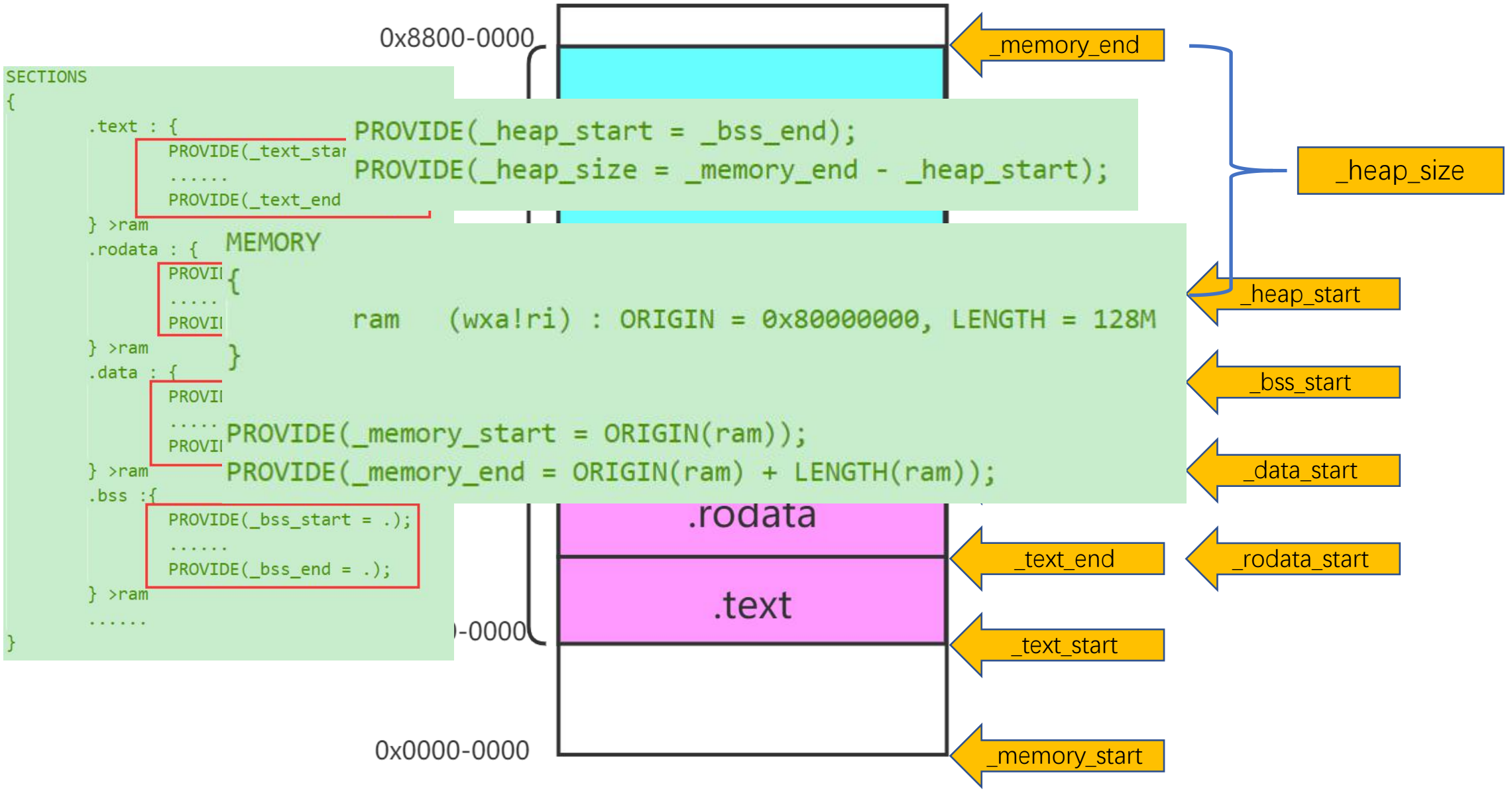
Linker Script 链接脚本

➢ PROVIDE

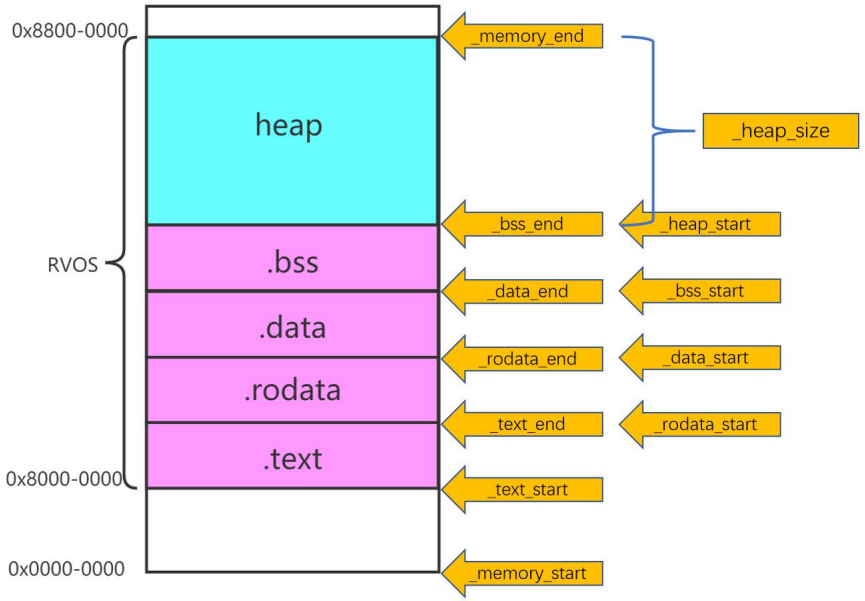
语法	PROVIDE(symbol = expression)
例子	PROVIDE(_text_start = .)

- 可以在 Linker Script 中定义符号（Symbols）
- 每个符号包括一个名字（name）和一个对应的地址值（address）
- 在代码中可以访问这些符号，等同于访问一个地址。

通过符号获取各个 output sections 在内存中的地址范围



从 Linker Script 到 Code




```
.section .rodata
.global HEAP_START
HEAP_START: .word _heap_start

.global HEAP_SIZE
HEAP_SIZE: .word _heap_size

.global TEXT_START
TEXT_START: .word _text_start

.global TEXT_END
TEXT_END: .word _text_end

.global DATA_START
```

```
extern uint32_t TEXT_START;
extern uint32_t TEXT_END;
extern uint32_t DATA_START;
extern uint32_t DATA_END;
extern uint32_t RODATA_START;
extern uint32_t RODATA_END;
extern uint32_t BSS_START;
extern uint32_t BSS_END;
extern uint32_t HEAP_START;
extern uint32_t HEAP_SIZE;
```

```
_num_pages = (HEAP_SIZE / PAGE_SIZE) - 8;

struct Page *page = (struct Page *)HEAP_START;
......
_alloc_start = _align_page(HEAP_START + 8 * PAGE_SIZE);
_alloc_end = _alloc_start + (PAGE_SIZE * _num_pages);
```

本章内容

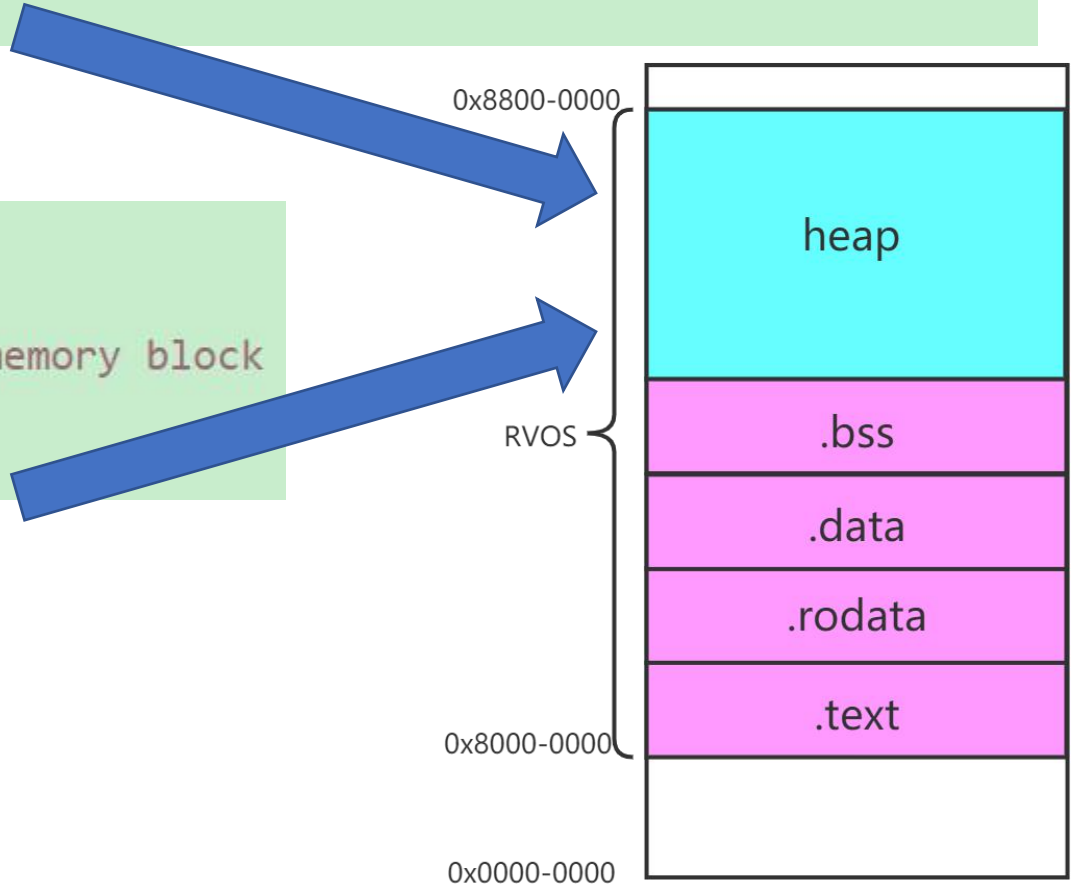
- **对内存进一步的管理，实现动态的分配和释放。**

- **实现 Page 级别的内存分配和释放。**
  - 数据结构设计
  - Page 分配和释放接口设计

需求分析：基于 Page 实现动态内存分配

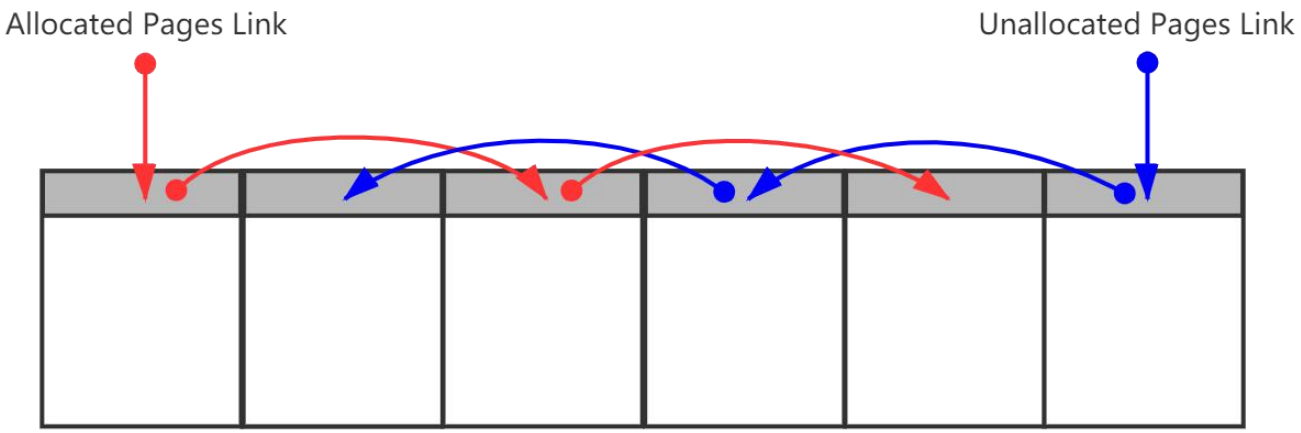
```
/*
 * Allocate a memory block which is composed of contiguous physical pages
 * - npages: the number of PAGE_SIZE pages to allocate
 */
void *page_alloc(int npages)
```

```
/*
 * Free the memory block
 * - p: start address of the memory block
 */
void page_free(void *p)
```

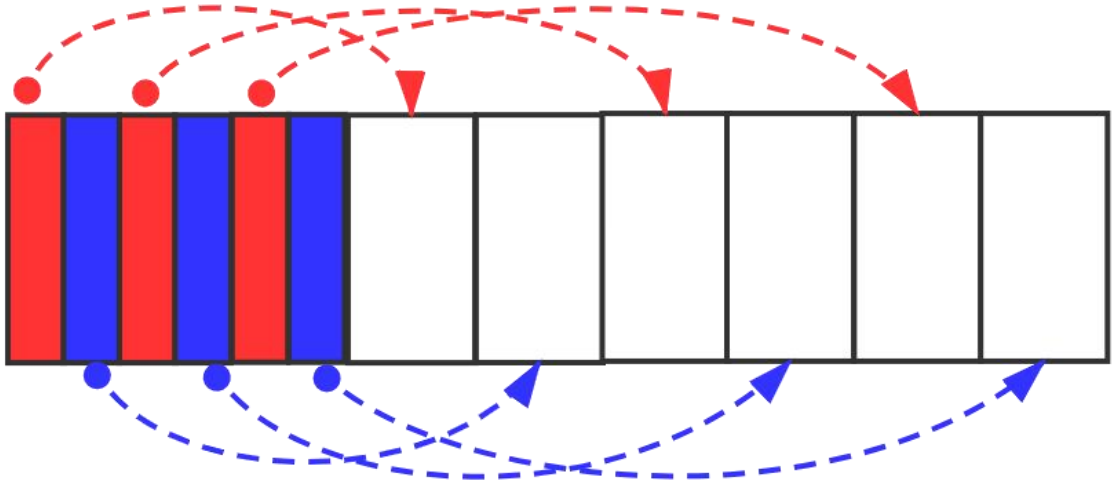


数据结构设计

- 链表方式。

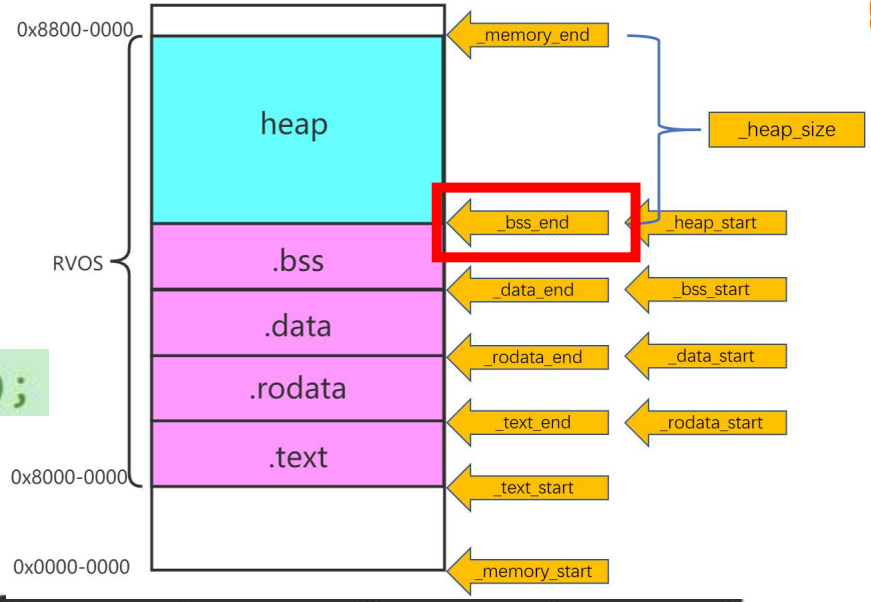


- 数组方式。



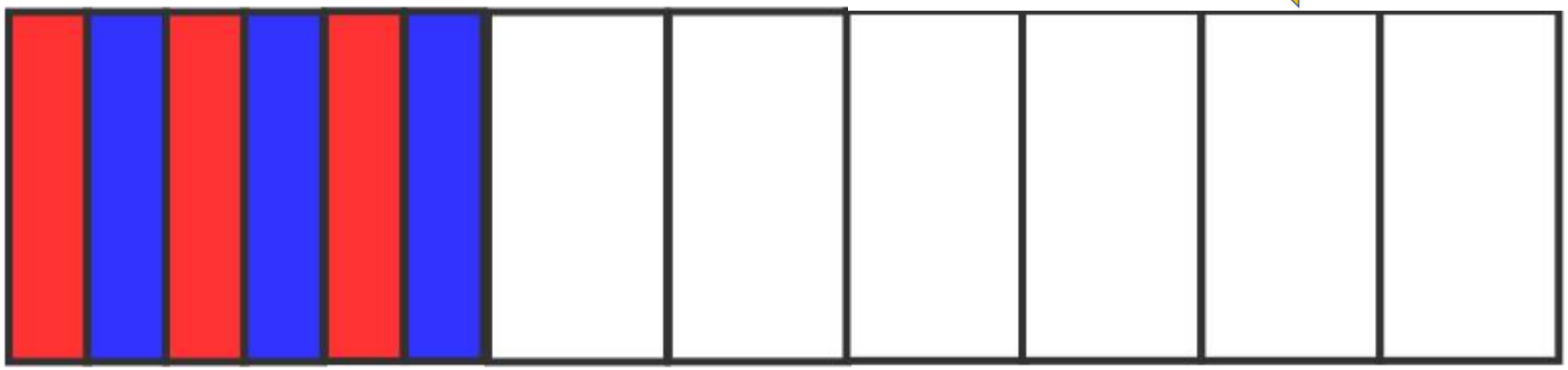
数据结构设计

- **数组方式。**



HEAP_START

```
PROVIDE(_heap_start = _bss_end);
```

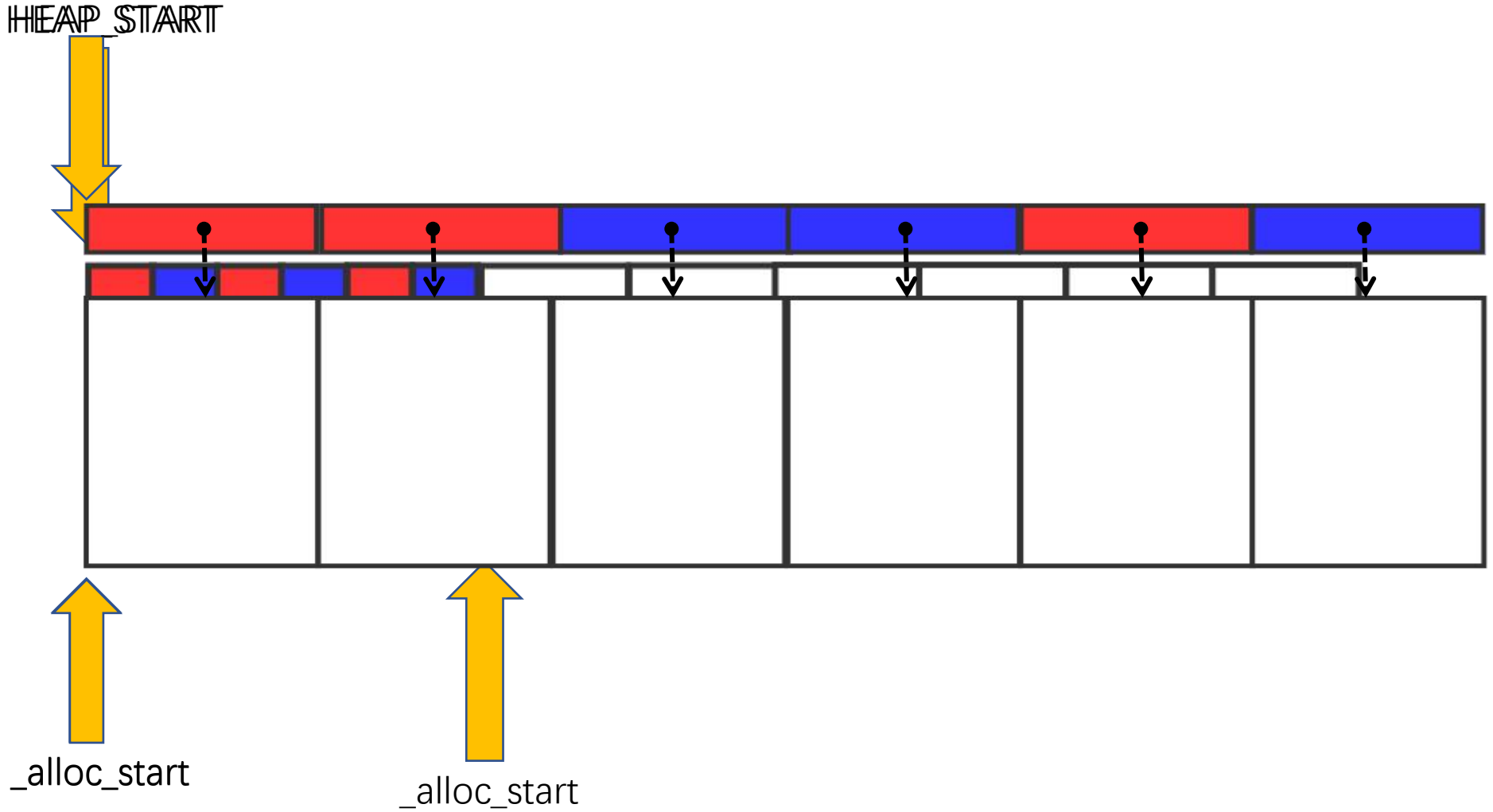


```
_alloc_start = _align_page(HEAP_START + 8 * PAGE_SIZE);
```

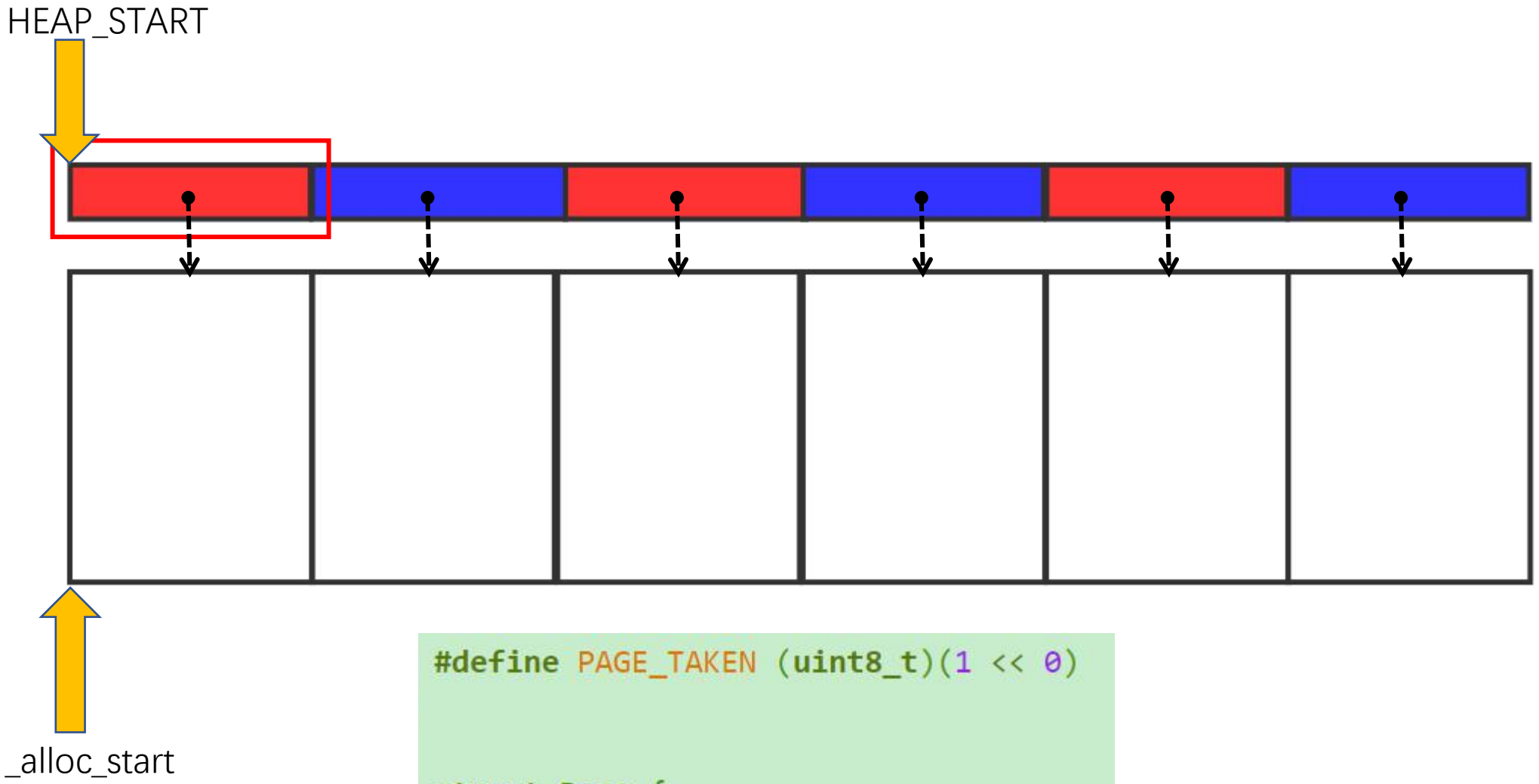
_alloc_start

数据结构设计

- **数组方式。**



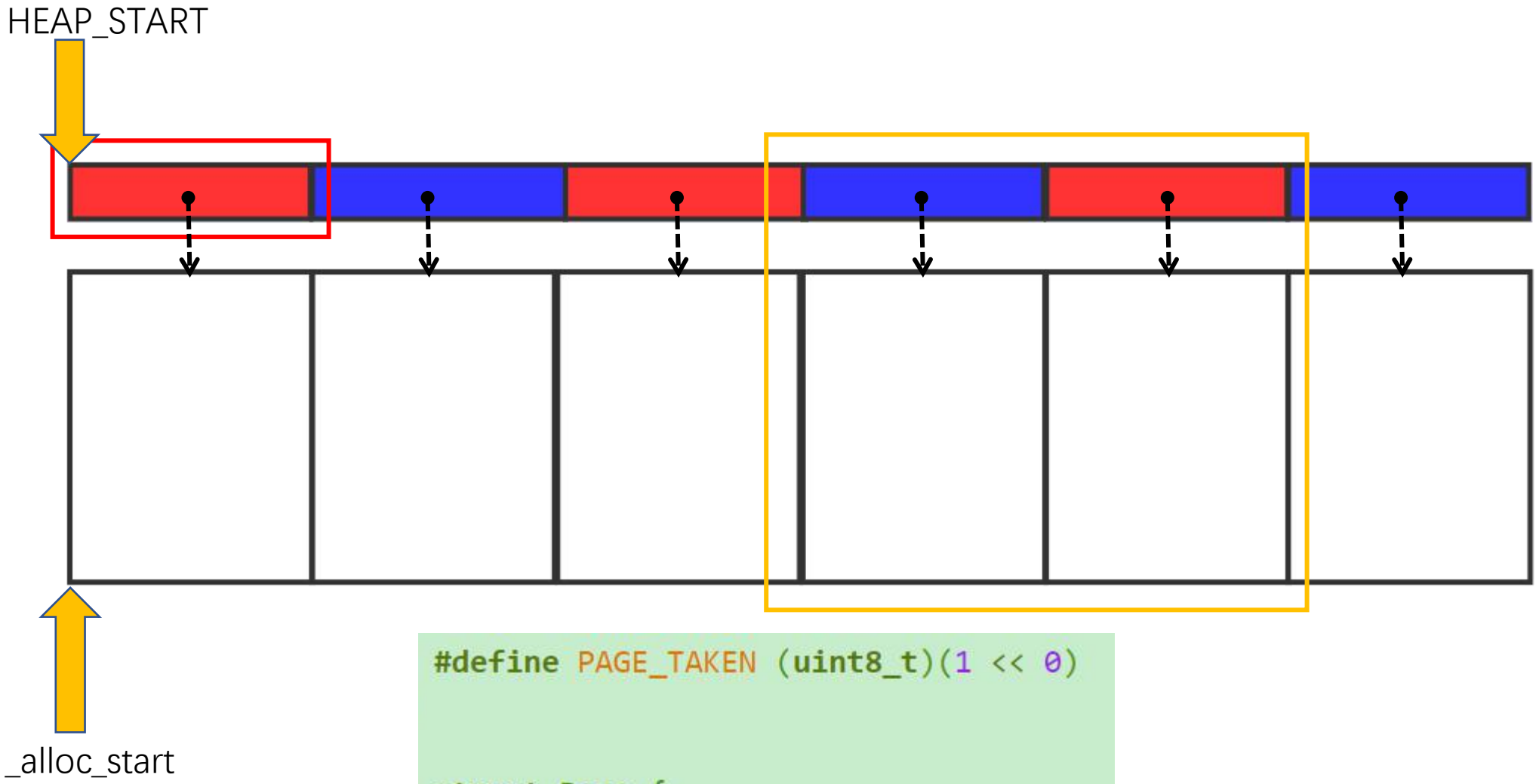
Page 描述符数据结构设计




```
#define PAGE_TAKEN (uint8_t)(1 << 0)


struct Page {
        uint8_t flags;
};
```

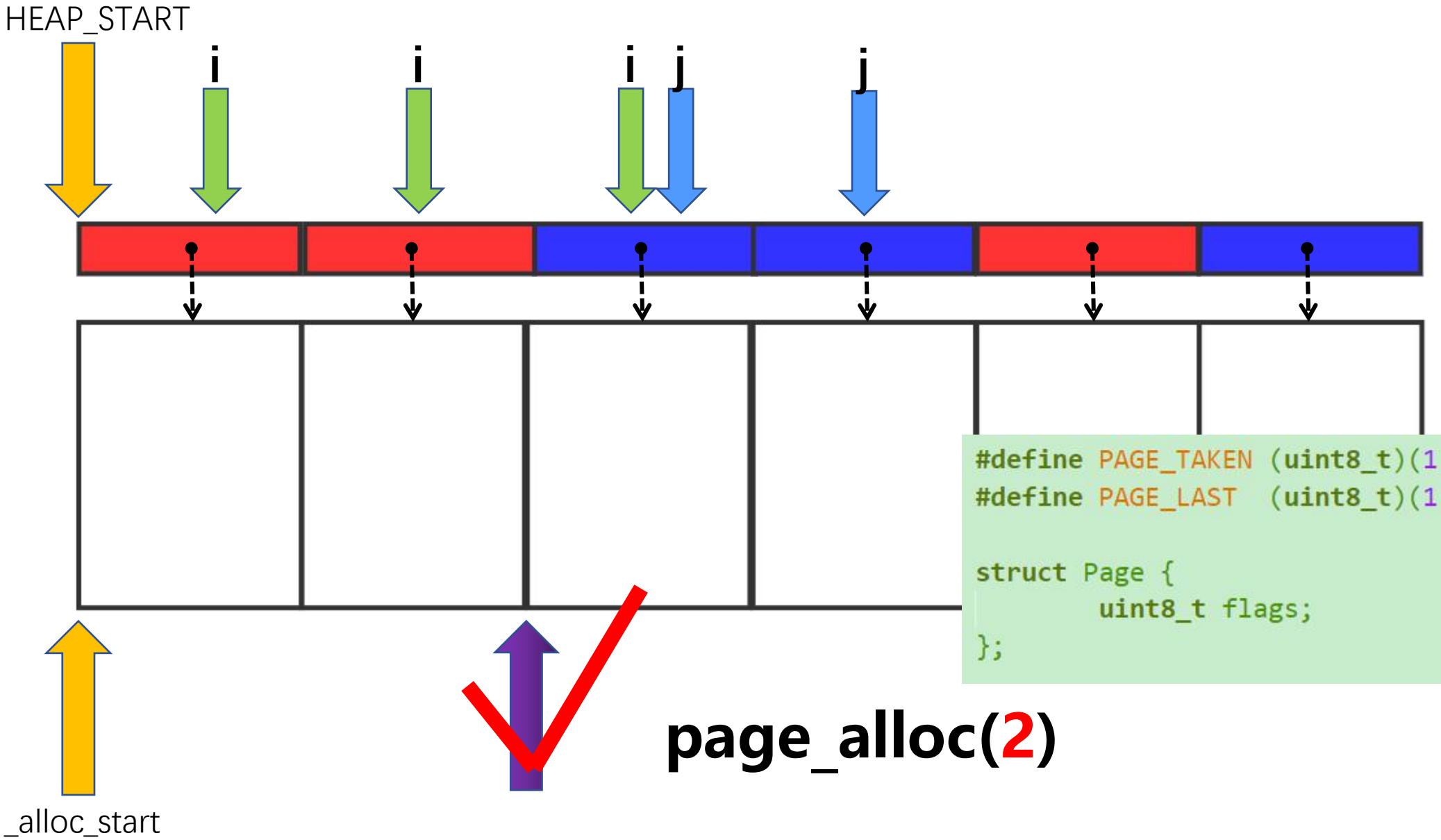
Page 描述符数据结构设计




```
#define PAGE_TAKEN (uint8_t)(1 << 0)


struct Page {
        uint8_t flags;
};
```

void *page_alloc(int npages)

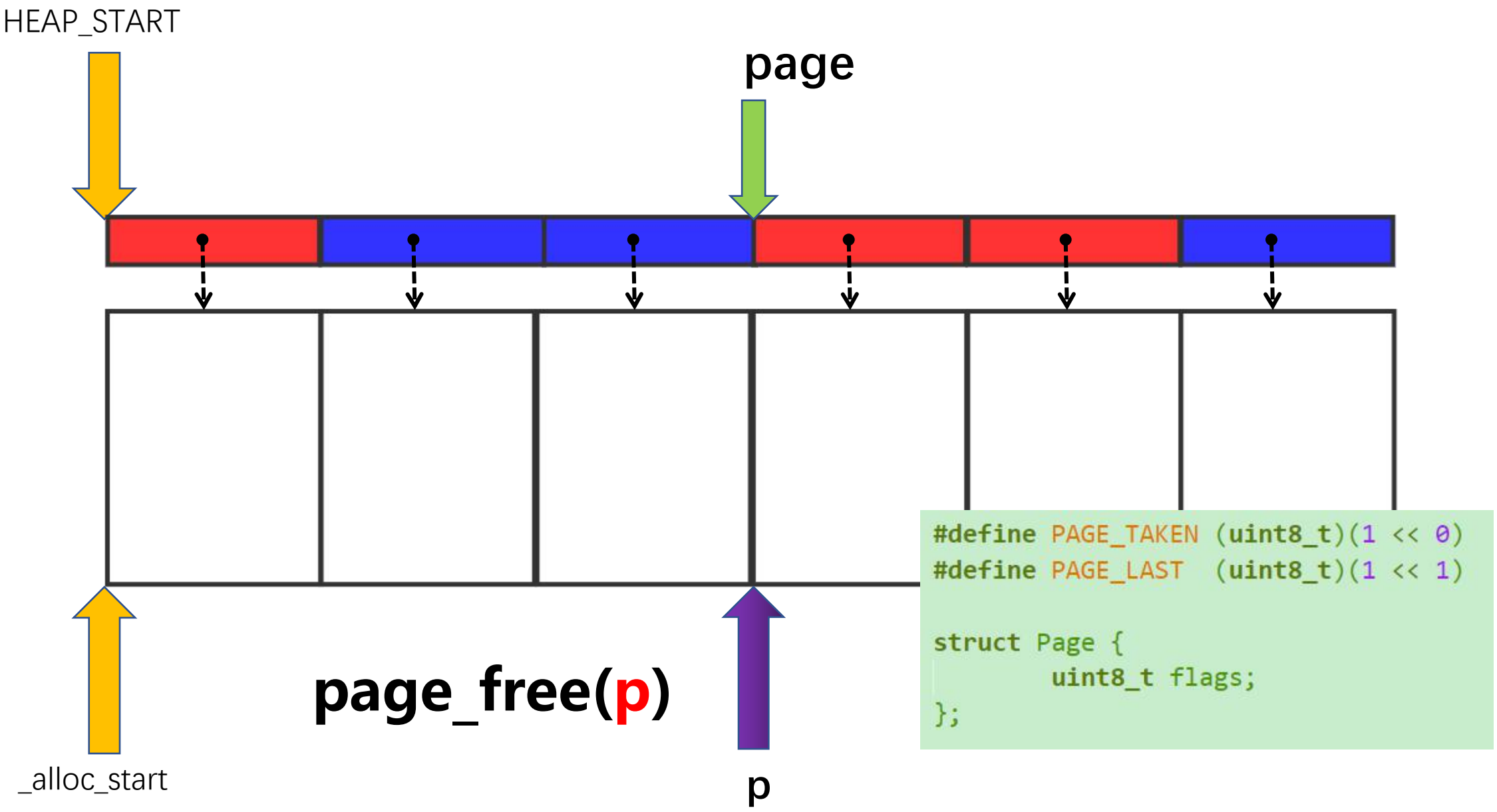



```
#define PAGE_TAKEN (uint8_t)(1 << 0)
#define PAGE_LAST  (uint8_t)(1 << 1)

struct Page {
        uint8_t flags;
};
```

page_alloc(2)

void page_free(void *p)



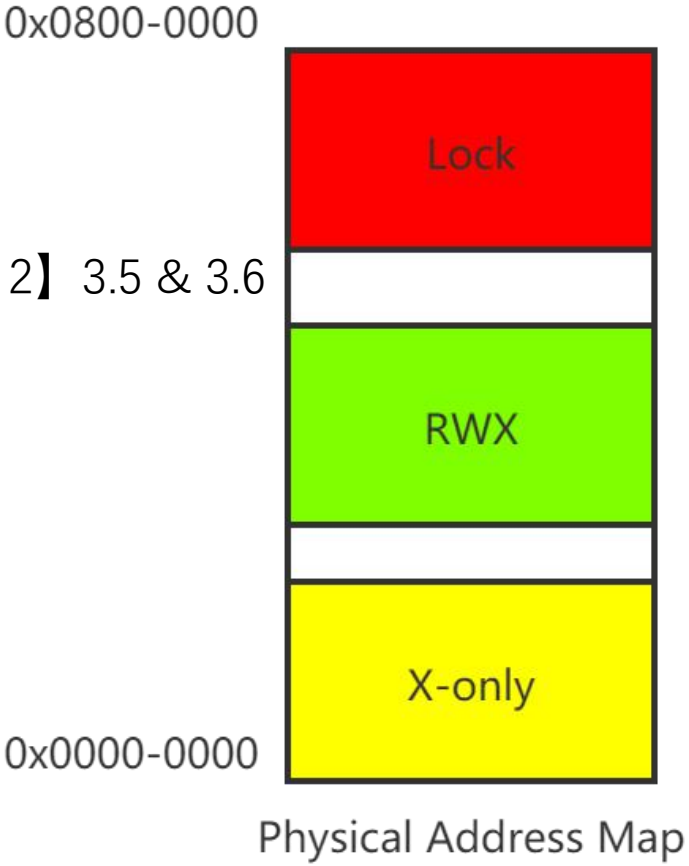

```
#define PAGE_TAKEN (uint8_t)(1 << 0)
#define PAGE_LAST  (uint8_t)(1 << 1)

struct Page {
        uint8_t flags;
};
```

内存管理与保护

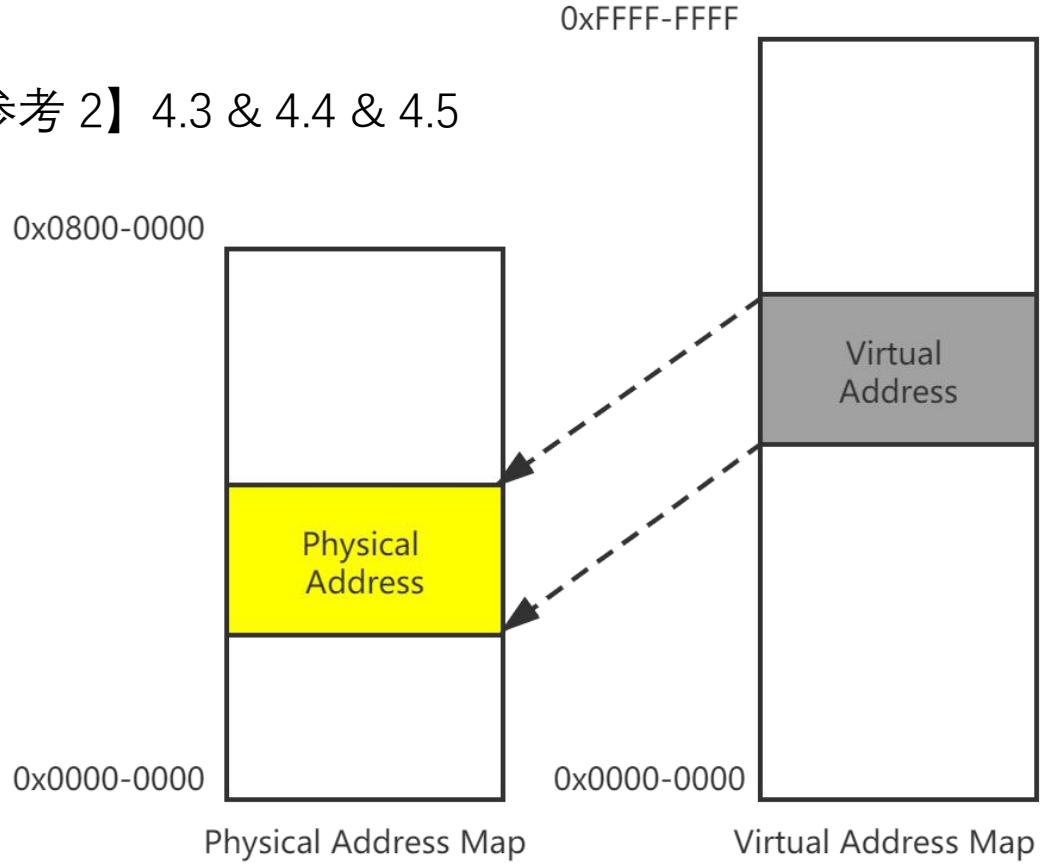
- **物理内存保护（Physical Memory Protection，PMP）**
  - 允许 M 模式指定 U 模式可以访问的内存地址。
  - 支持 R/W/X，以及 Lock

【参考 2】3.5 & 3.6



- **虚拟内存（Virtual Memory）**
  - 需要支持 Supervisor Level
  - 用于实现高级的操作系统特性（Unix/Linux）
  - 多种映射方式 Sv32/Sv39/Sv48

【参考 2】4.3 & 4.4 & 4.5



练习 8-1

谢 谢

欢迎交流合作