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。

# 设计思路

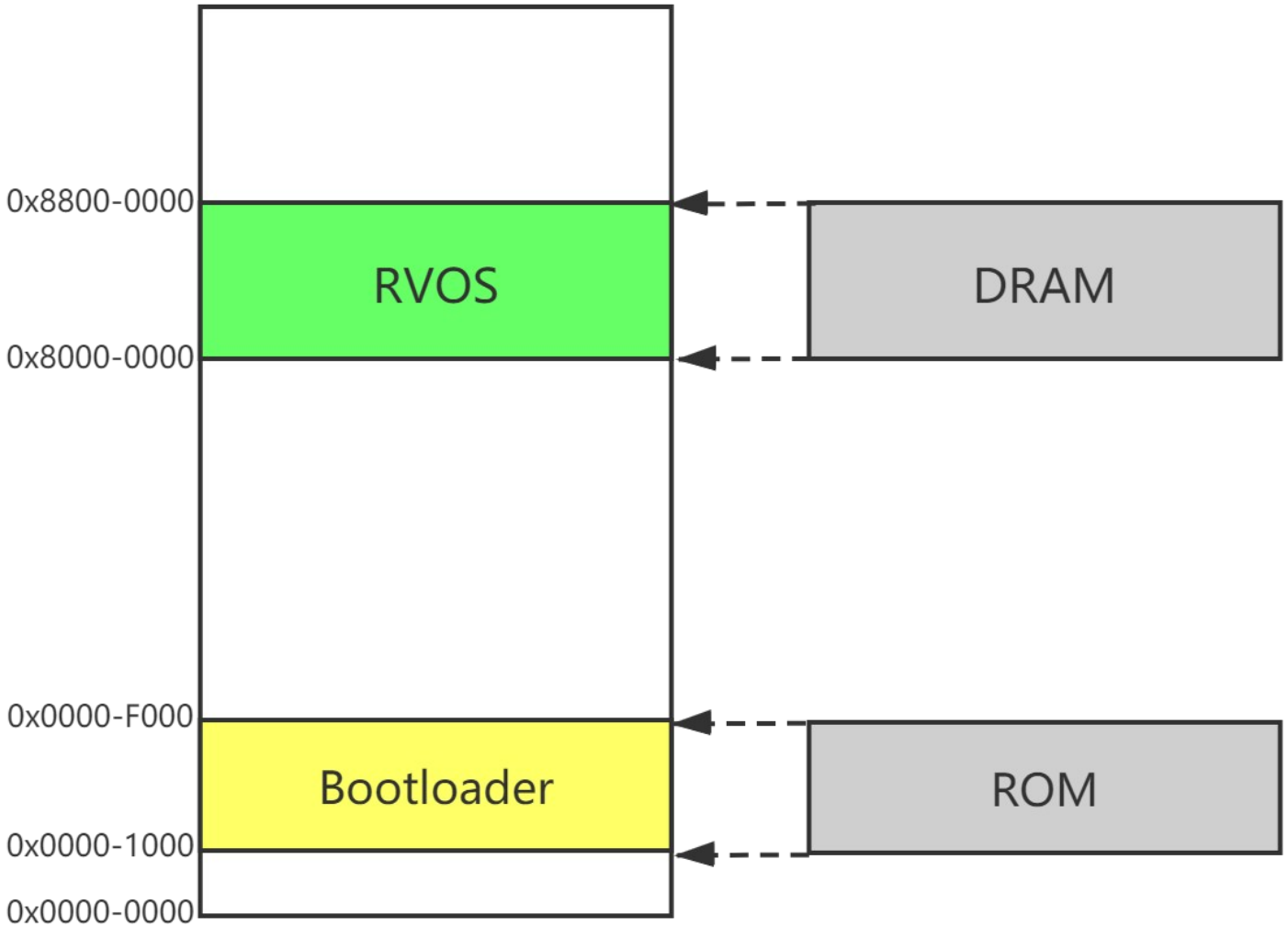
- **建立内存分配表，找出可以动态分配和释放的物理内存区域和大小。**
  - 建立内存分配表
  - 定义和内存区域对应的符号变量
  - 链接脚本文件

- **实现 Page 级别的内存分配和释放的接口。**

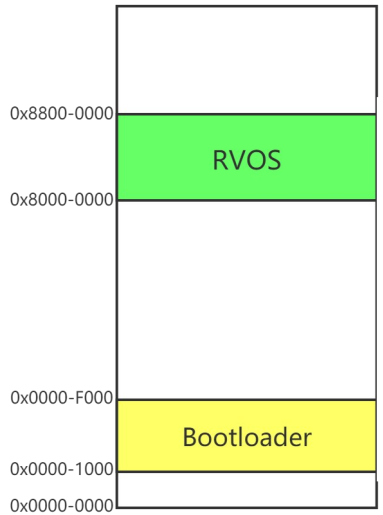
# 动态内存管理

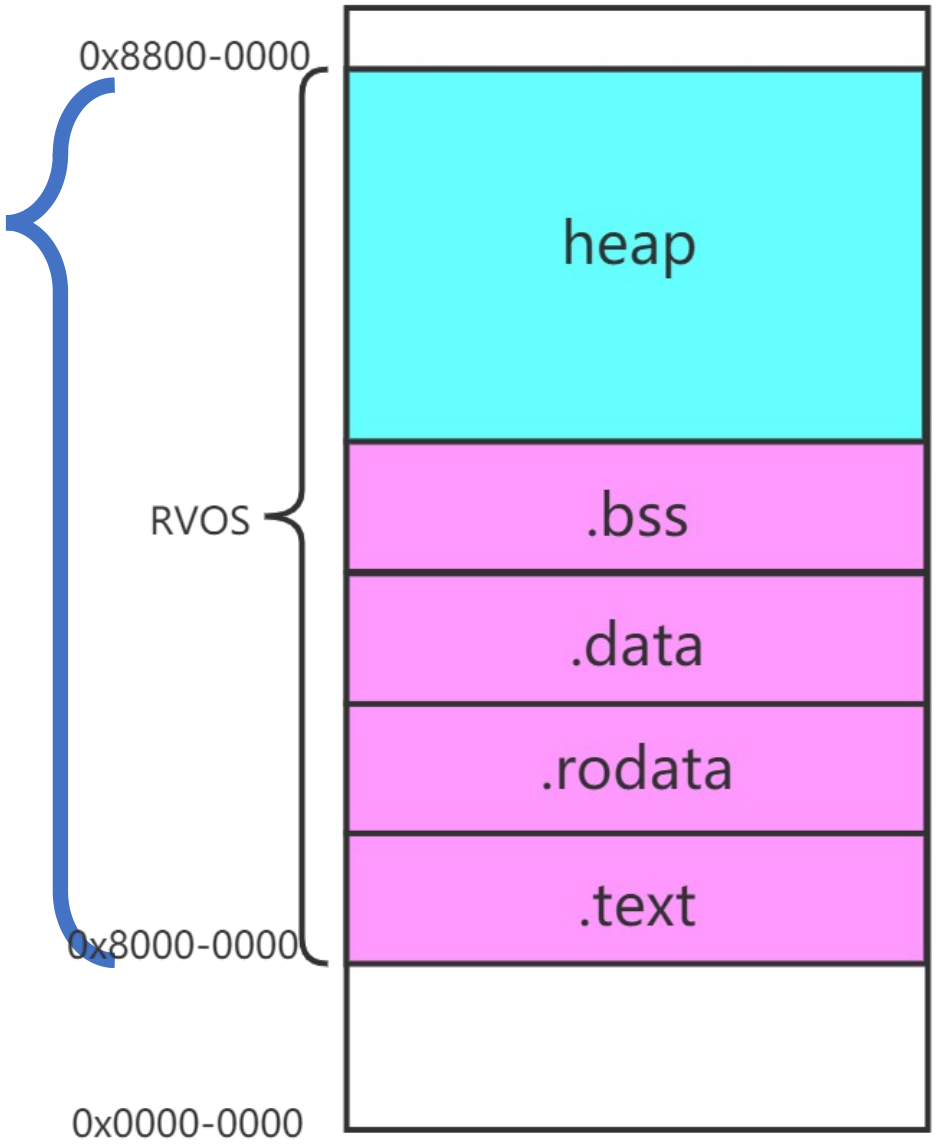
- 静态内存 - 全局变量
- 自动管理内存 - 栈（stack）
- 动态管理内存 - 堆（heap）

# 建立内存分配表

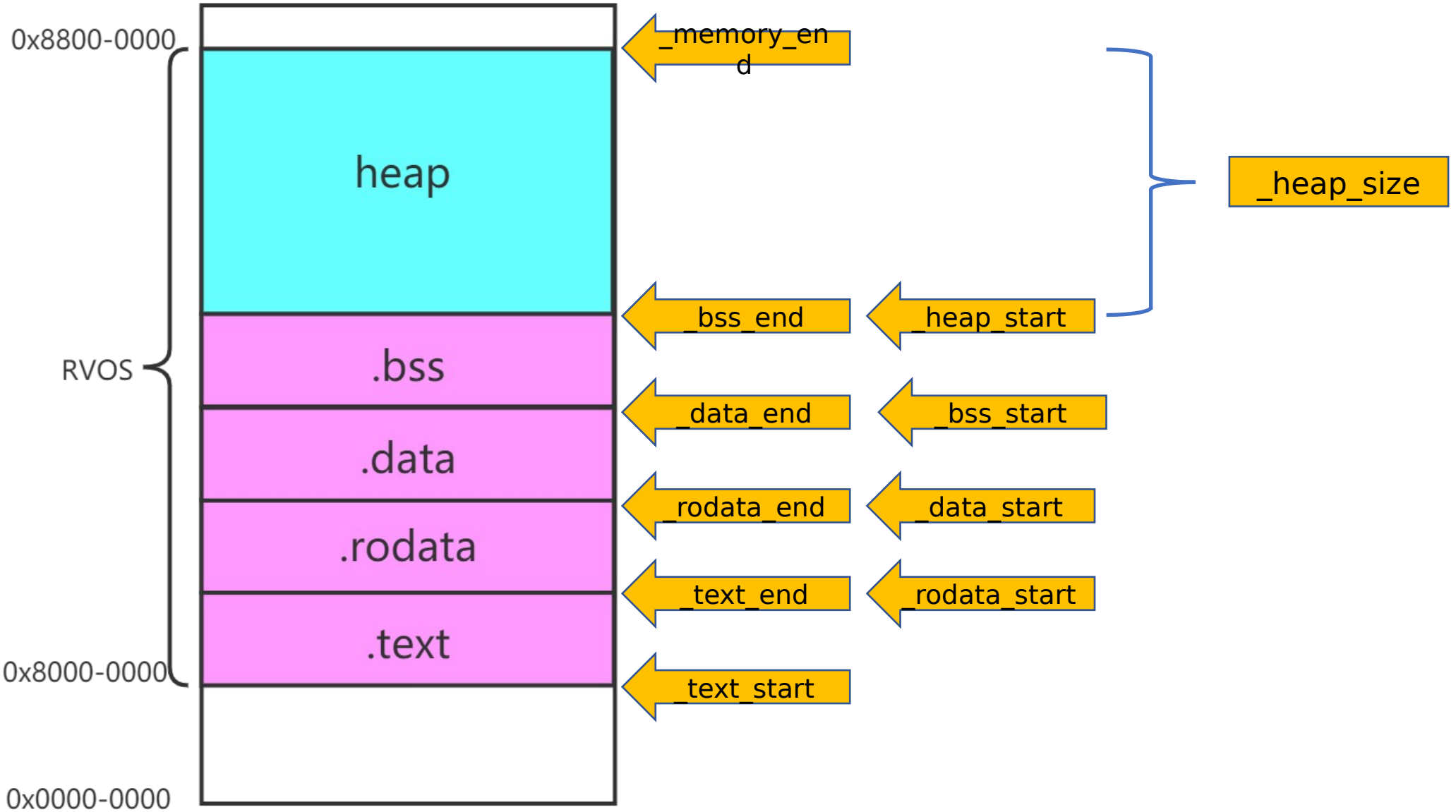

0x8800-0000
RVOS
0x8000-0000
DRAM
0x0000-F000
Bootloader
0x0000-1000
0x0000-0000
ROM

# 建立内存分配表


0x8800-0000
RVOS
0x8000-0000
0x0000-F000
Bootloader
0x0000-1000
0x0000-0000

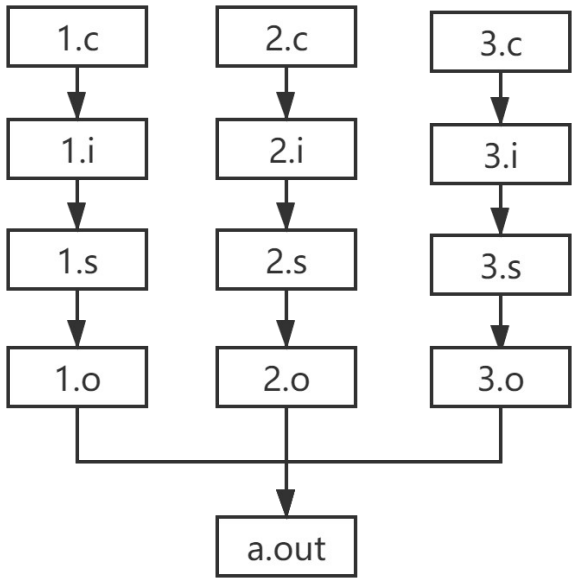

0x8800-0000
heap
RVOS
.bss
.data
.rodata
.text
0x8000-0000
0x0000-0000

# 定义和内存区域对应的符号变量



# Linker Script 链接脚本文件

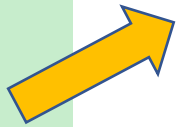
- Linker Script 是简单的纯文本文件
- 每个 Linker Script 中包含有多条命令（Command)
- gcc -T os.ld ……
- 一个简单的 Linker Script 的例子，只包含一条 SECTIONS 命令。



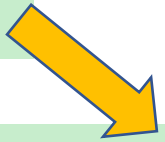

```
SECTIONS
{
   . = 0x10000;
   .text : { *(.text) }
   . = 0x8000000;
   .data : { *(.data) }
   .bss : { *(.bss) }
}
```

# 集成

```
.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
DATA_START: .word _data_start

.global DATA_END
DATA_END: .word _data_end

.global RODATA_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;
```



```
struct Page *page = (struct Page *)HEAP_START;
for (int i = 0; i < _num_pages; i++) {
        _clear(page);
        page++;
}

_alloc_start = _align_page(HEAP_START + 8 * PAGE_SIZE);
_alloc_end = _alloc_start + (PAGE_SIZE * _num_pages);

printf("TEXT:   0x%x -> 0x%x\n", TEXT_START, TEXT_END);
printf("RODATA: 0x%x -> 0x%x\n", RODATA_START, RODATA_END);
printf("DATA:   0x%x -> 0x%x\n", DATA_START, DATA_END);
printf("BSS:    0x%x -> 0x%x\n", BSS_START, BSS_END);
printf("HEAP:   0x%x -> 0x%x\n", _alloc_start, _alloc_end);
```

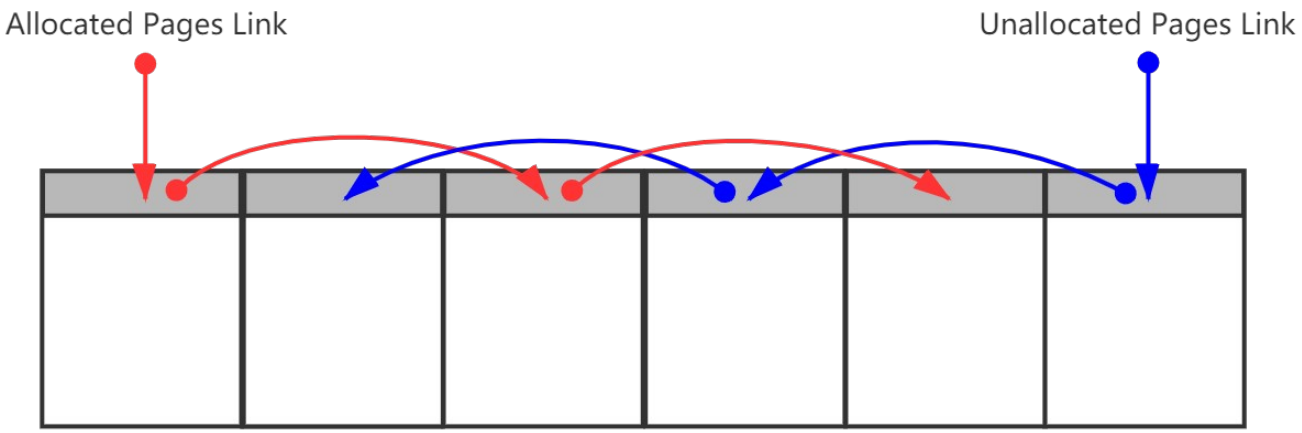
# 设计思路

- **建立内存分配表，找出可以动态分配和释放的物理内存区域和大小。**

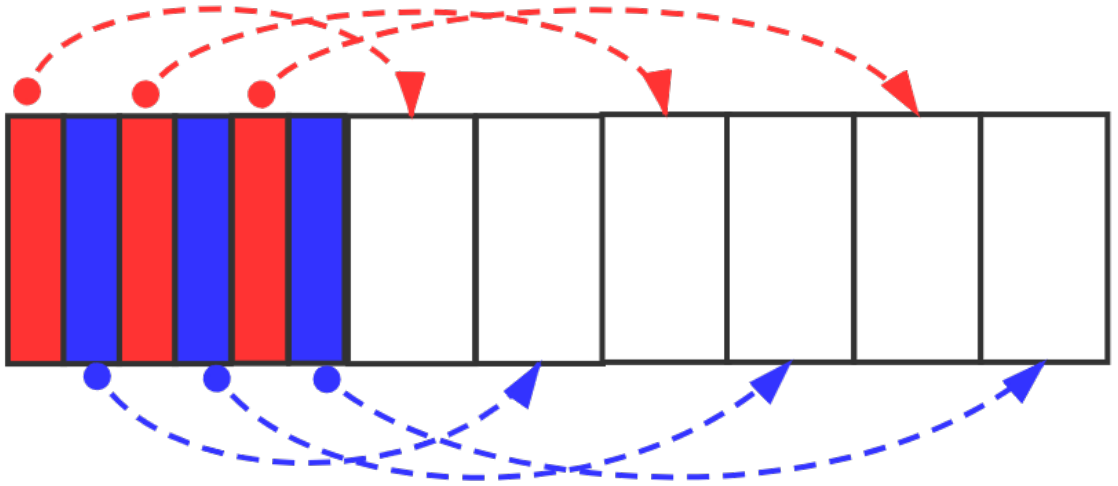
- **实现 Page 级别的内存分配和释放的接口。**
  - 数据结构设计
  - Page 分配和释放接口设计

# 设计思路

- 链表方式。

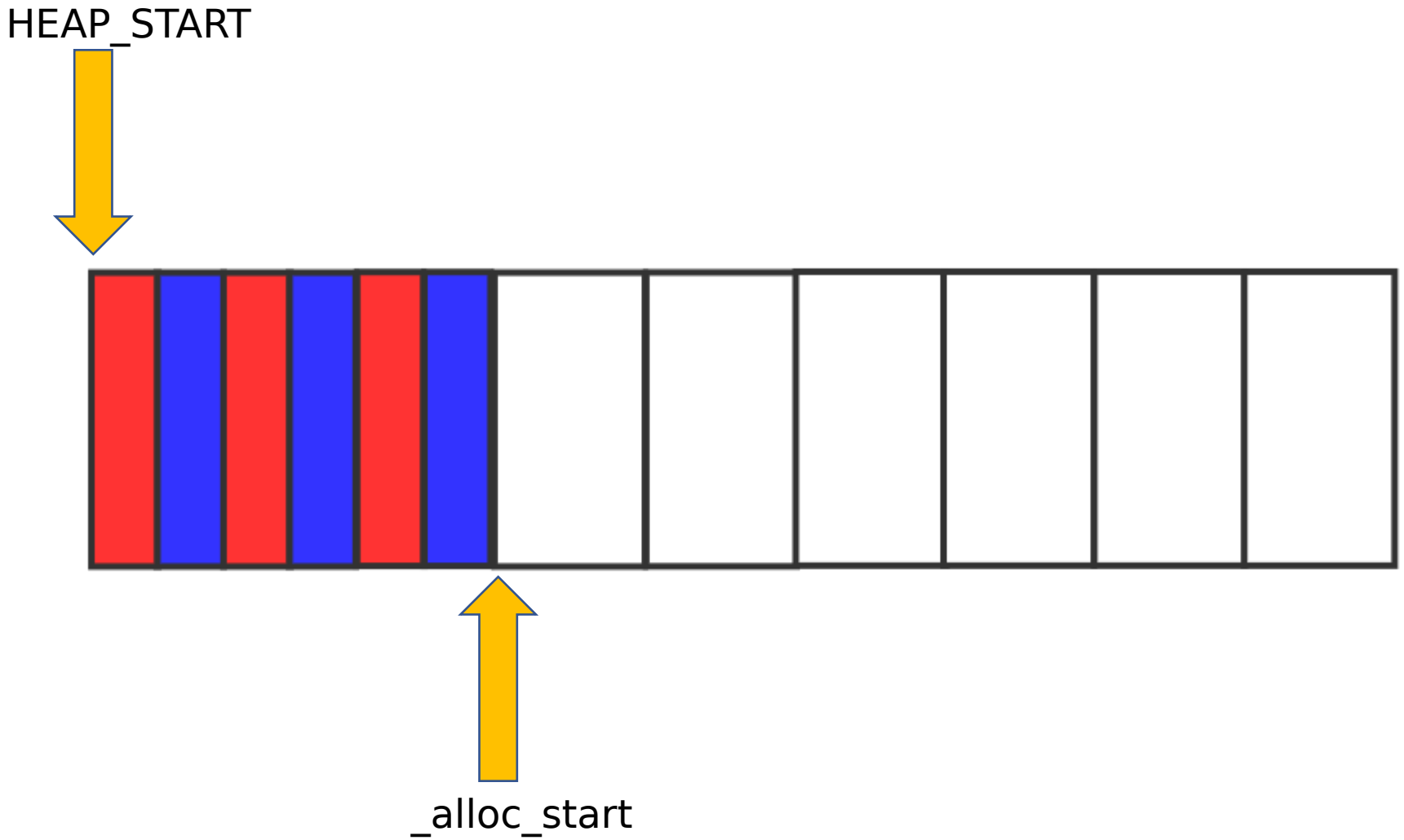



- 数组方式。

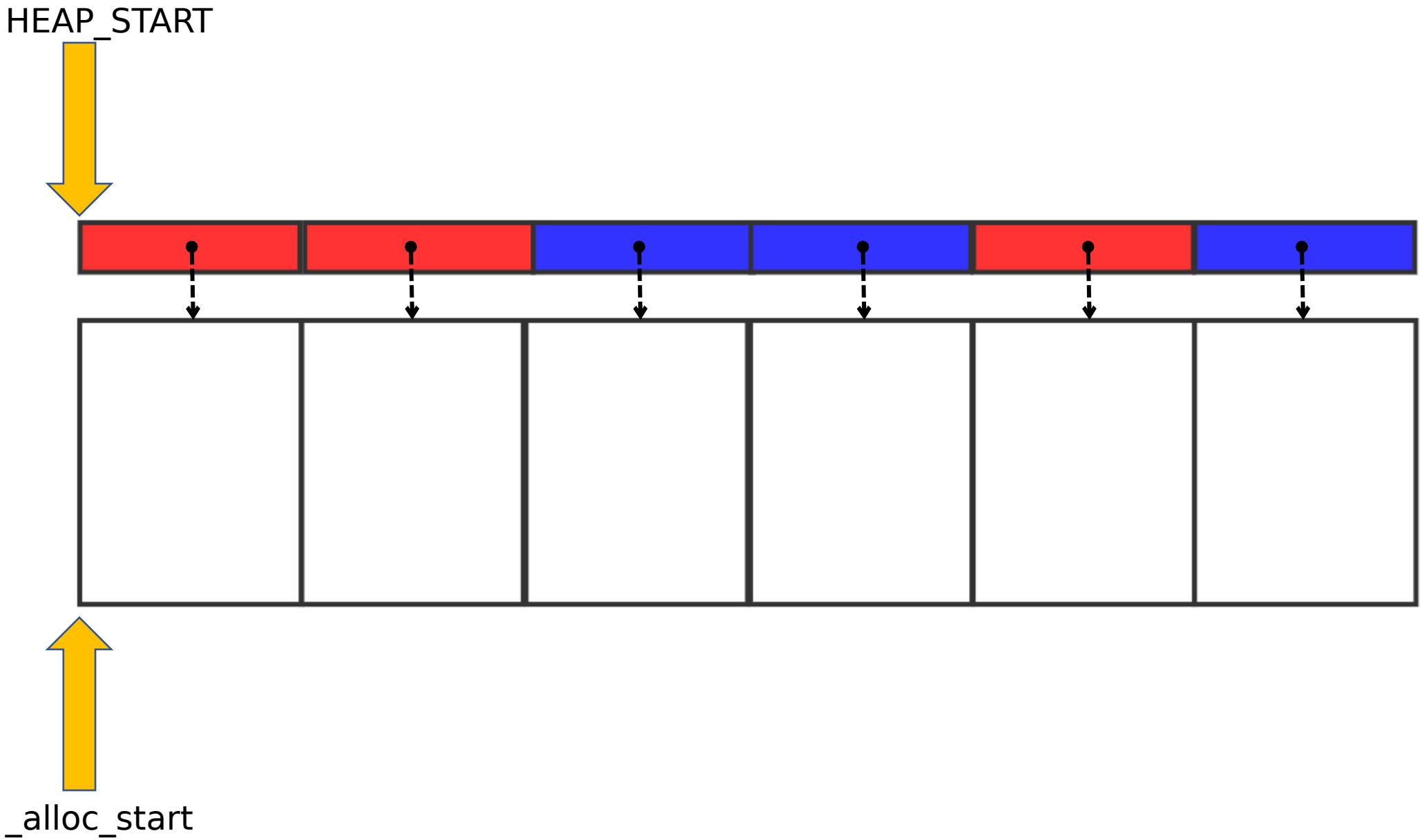


# 设计思路

- 数组方式。



# page_alloc()

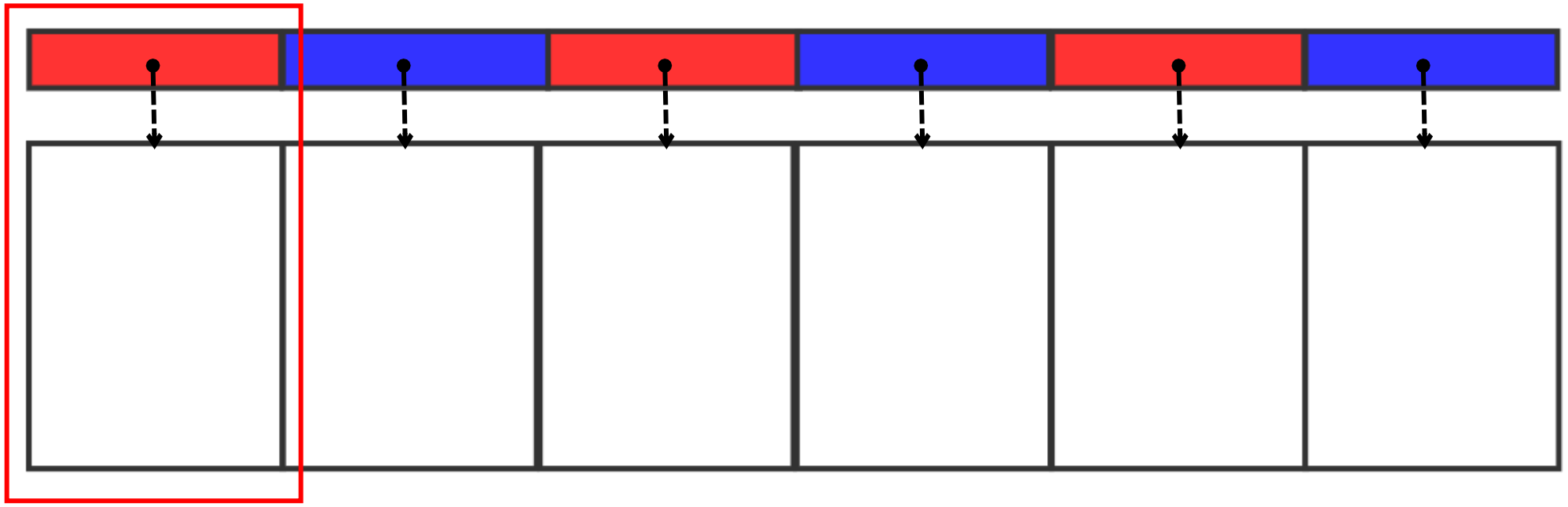


# Page 分配 API 设计

```
/*
 * 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)
```

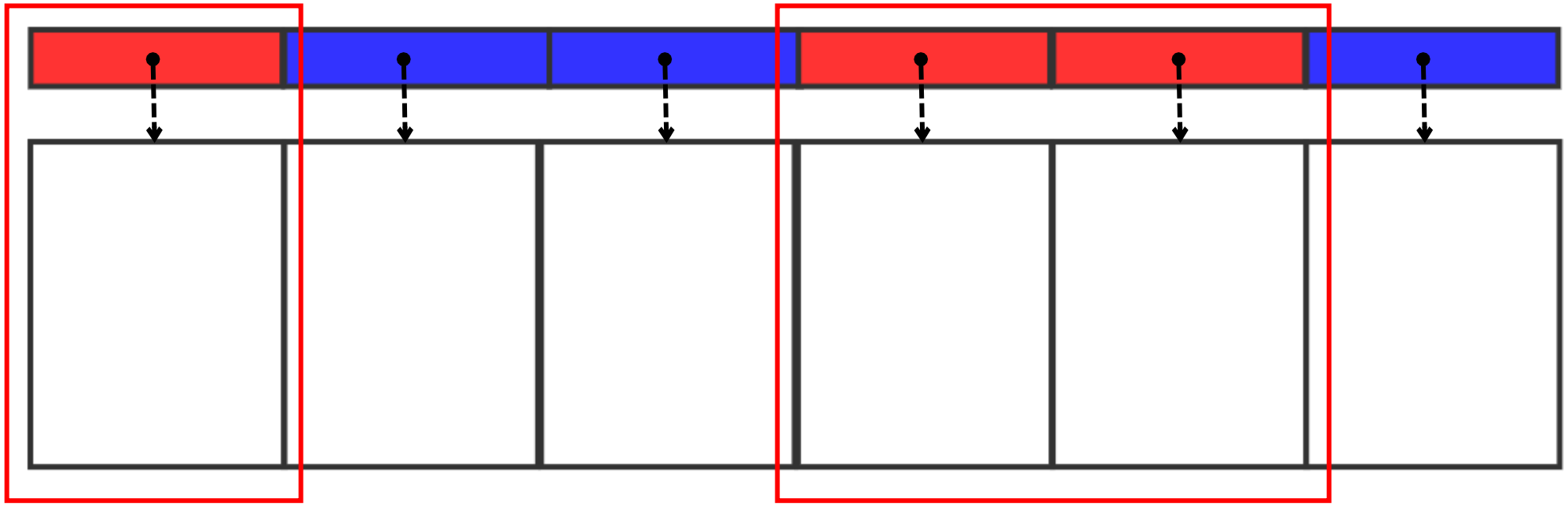
```
/*
 * Free the memory block
 * - p: start address of the memory block
 */
void page_free(void *p)
```

# Page 描述符数据结构设计



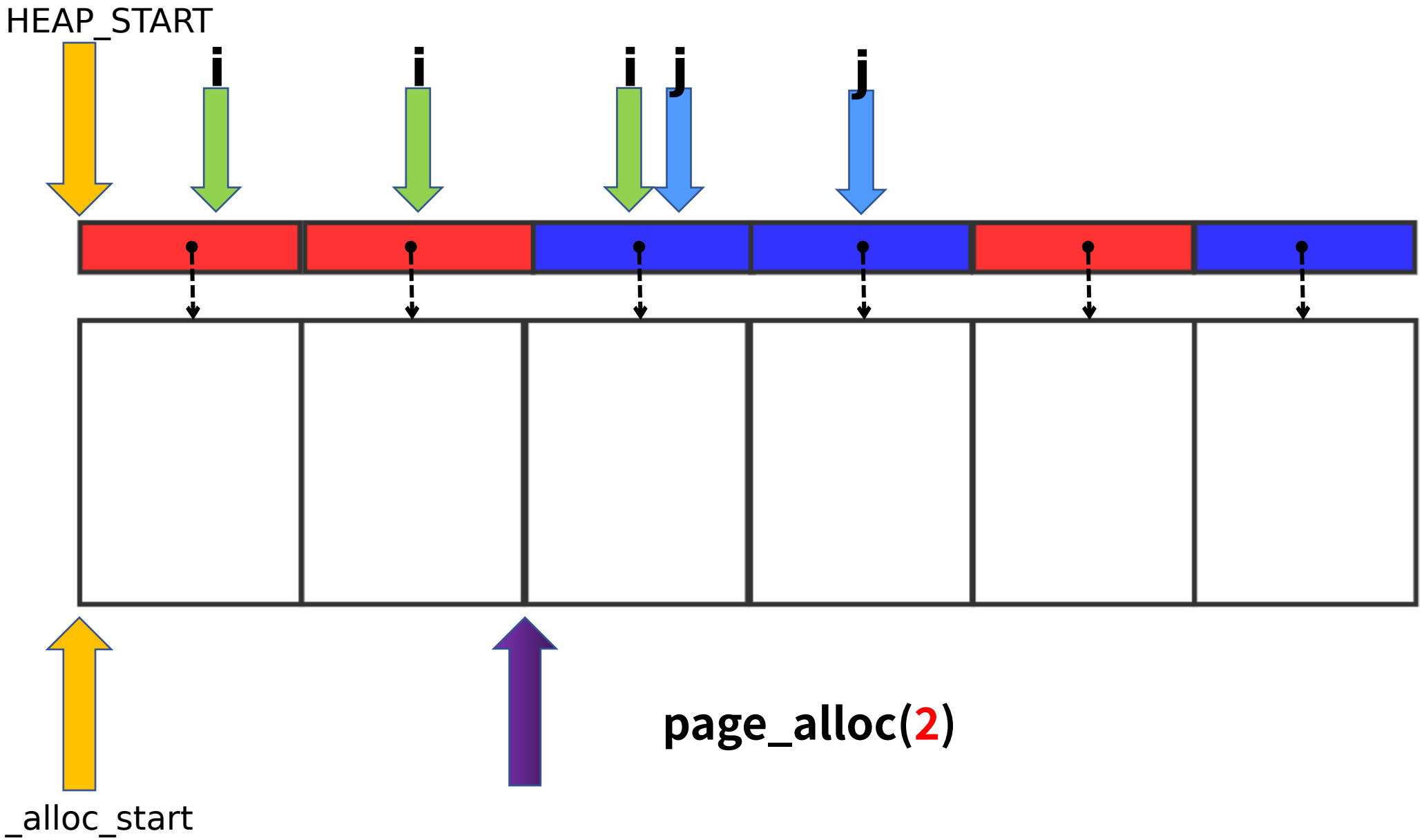
```
#define PAGE_TAKEN (uint8_t)(1 << 0)


struct Page {
        uint8_t flags;
};
```

# Page 描述符数据结构设计

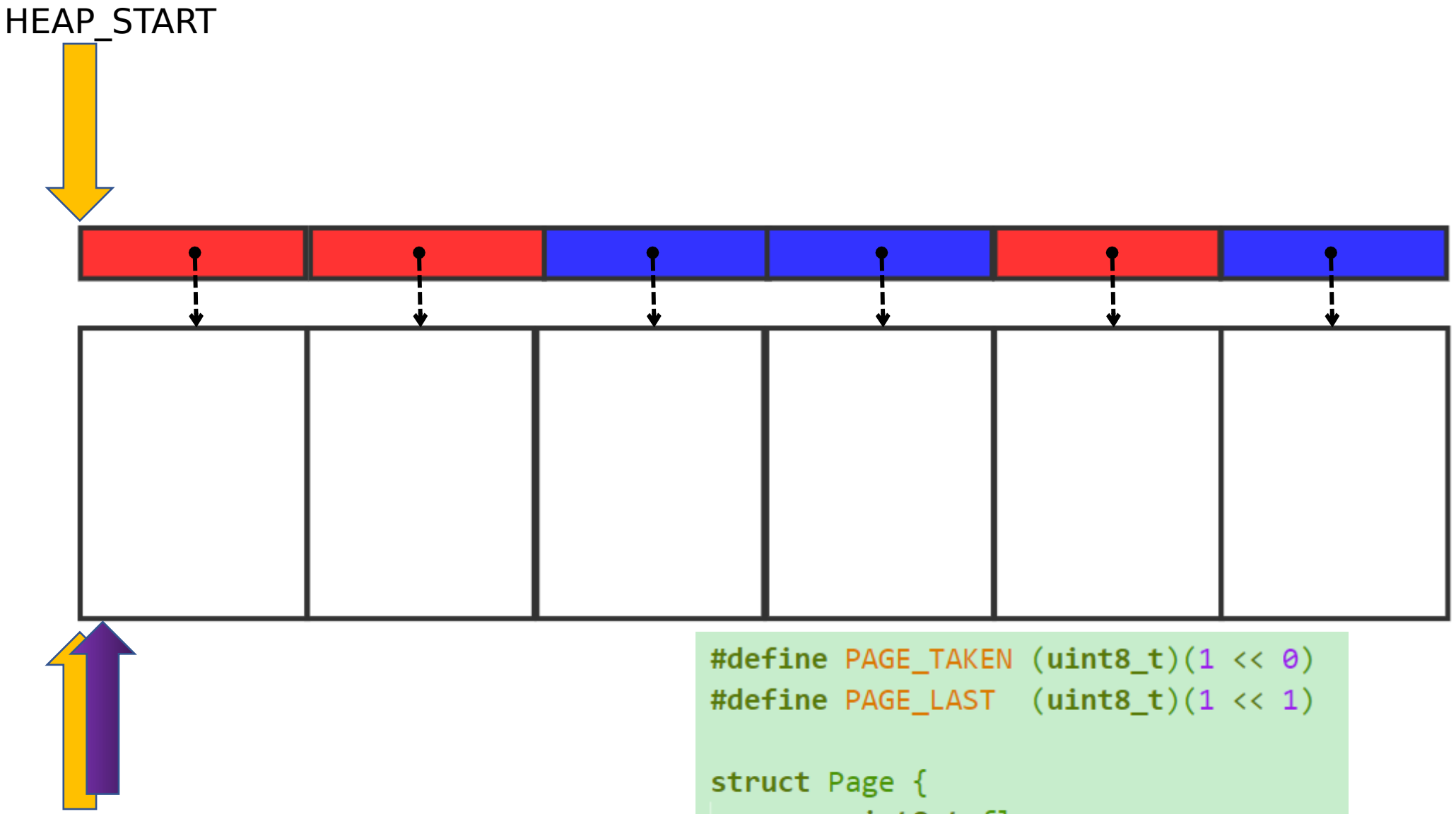


```
#define PAGE_TAKEN (uint8_t)(1 << 0)


struct Page {
        uint8_t flags;
};
```

# page_alloc()



# page_free()




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

struct Page {
        uint8_t flags;
};
```

# 谢 谢

欢迎交流合作