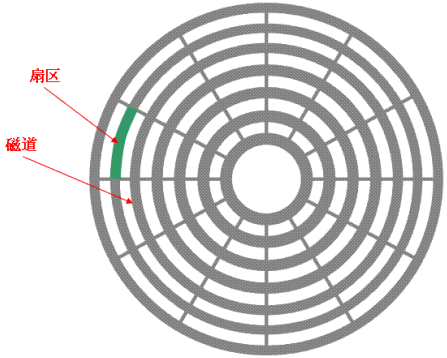
动手实践 – 编写一个简单的块设备驱动

笨叔叔

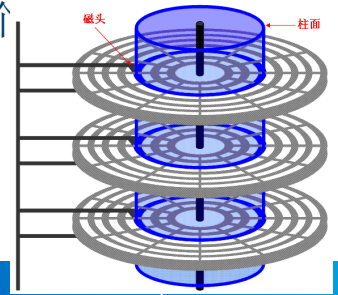
什么是块设备？



- 块设备的特点：
  - ✓ 字符设备是以字节流为传输单位的
  - ✓ 块设备是以块（block）为传输单位的
  - ✓ 字符设备按照字节流顺序来访问的
  - ✓ 块设备可以随机访问

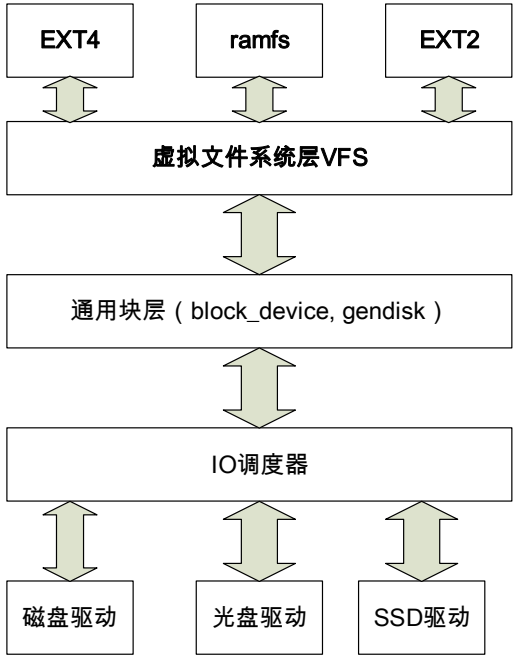
- 常见的块设备：机械硬盘
  - ✓ 磁头(head)： 磁头固定在可移动的机械臂上，用于读写数据
  - ✓ 磁道(track)：每个盘面都有 n 个同心圆组成，每个同心圆称之为一个磁道
  - ✓ 柱面(cylinder)： n 个盘面的相同磁道（位置相同）共同组成一个柱面
  - ✓ 扇区(sector)：从磁盘中心向外画直线，可以将磁道划分为若干个弧段。每个一个弧段被称之为一个扇区

- 其他常见块设备：光盘，SSD等



块设备驱动架构

- 块设备驱动架构
  - ✓ 文件系统层：包含了常用的文件系统如ext4
  - ✓ 通用块层：
  - ✓ IO 调度器：
  - ✓ 具体块设备驱动：



block_device数据结构

- block_device数据结构用来抽象和描述一个块设备。
- block_device把虚拟文件系统VFS和块设备子系统关联起来

```
struct block_device {
      dev_t           bd_dev;
      struct inode *       bd_inode;
      struct super_block * bd_super;
      struct gendisk *     bd_disk;
      struct request_queue *  bd_queue
};
```

gendisk数据结构

- 磁盘类设备的一个抽象
- 可以表示一个已经分区的磁盘，也可以表示一个未分区的磁盘

```
struct gendisk {
        int major;
        int first_minor;
        int minors;
        char disk_name[DISK_NAME_LEN];
        struct disk_part_tbl __rcu *part_tbl;
        const struct block_device_operations *fops;
        struct request_queue *queue;
};
```

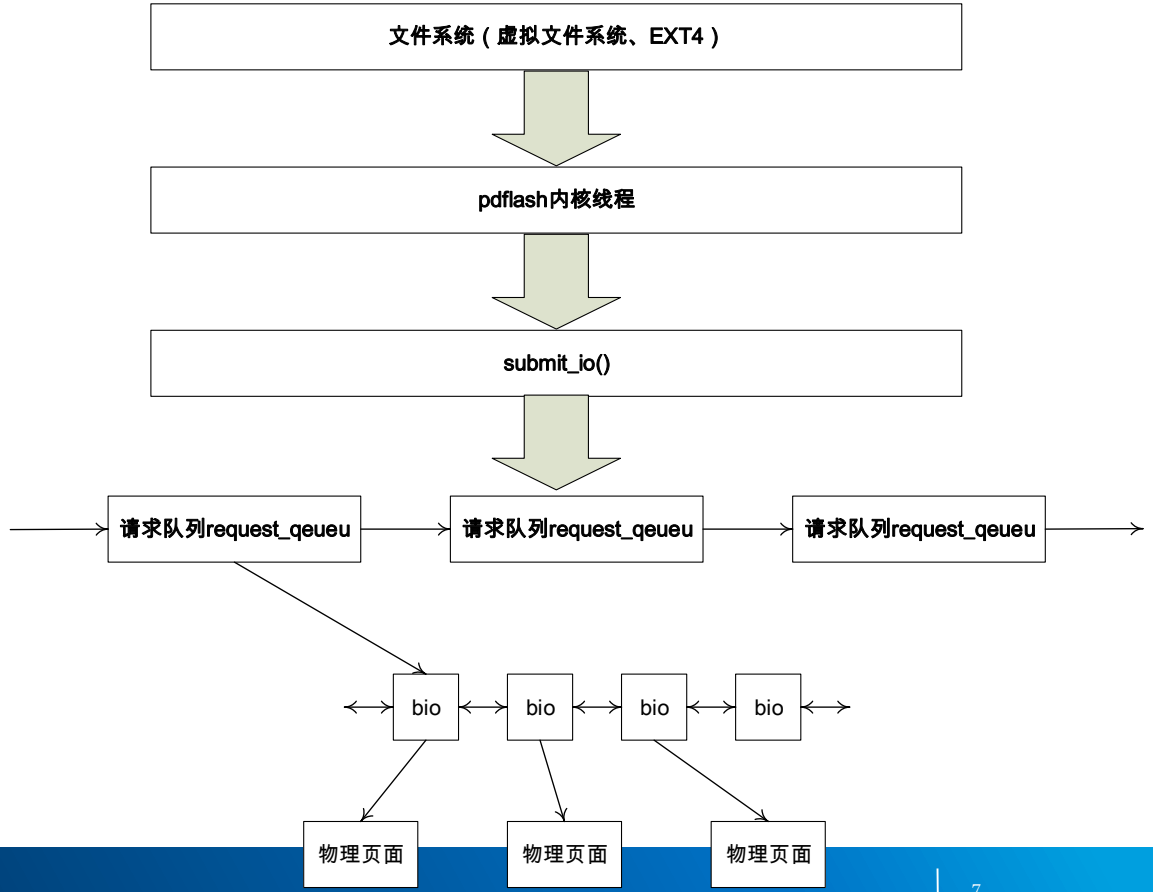
block_device_operations操作方法集

- 块设备的操作方法集，类似字符设备的file_opereations操作方法集

```
struct block_device_operations {
        int (*open) (struct block_device *, fmode_t);
        void (*release) (struct gendisk *, fmode_t);
        int (*rw_page)(struct block_device *, sector_t, struct page *, int rw);
        int (*ioctl) (struct block_device *, fmode_t, unsigned, unsigned long);
        int (*compat_ioctl) (struct block_device *, fmode_t, unsigned, unsigned long);
        long (*direct_access)(struct block_device *, sector_t,
                              void **, unsigned long *pfn, long size);
        unsigned int (*check_events) (struct gendisk *disk,
                              unsigned int clearing);
        int (*media_changed) (struct gendisk *);
        void (*unlock_native_capacity) (struct gendisk *);
        int (*revalidate_disk) (struct gendisk *);
        int (*getgeo)(struct block_device *, struct hd_geometry *);
        /* this callback is with swap_lock and sometimes page table lock held */
        void (*swap_slot_free_notify) (struct block_device *, unsigned long);
        struct module *owner;
};
```

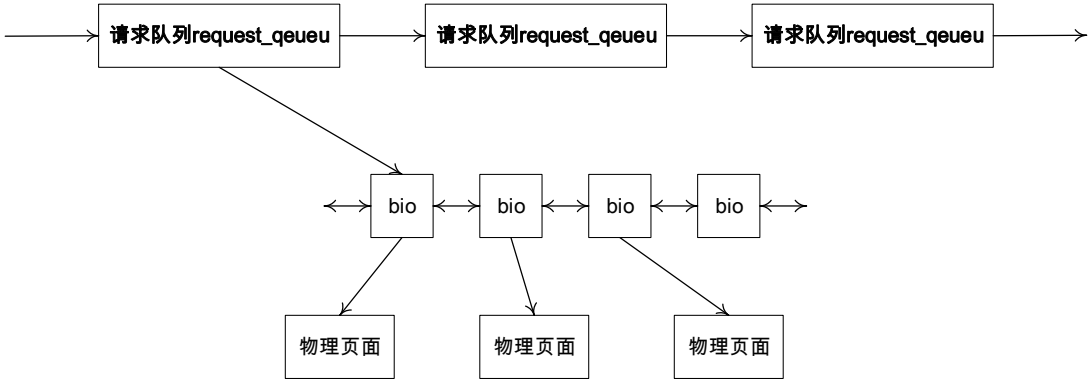
请求队列request_queue

- request_queue数据结构来抽象和描述请求队列
- 当文件系统这样的高层代码有新的请求就会加入到请求队列中。只要请求队列不为空，那么队列中对应中的块设备驱动程序就会从请求队列中获取request，然后送到对应的块设备驱动中。

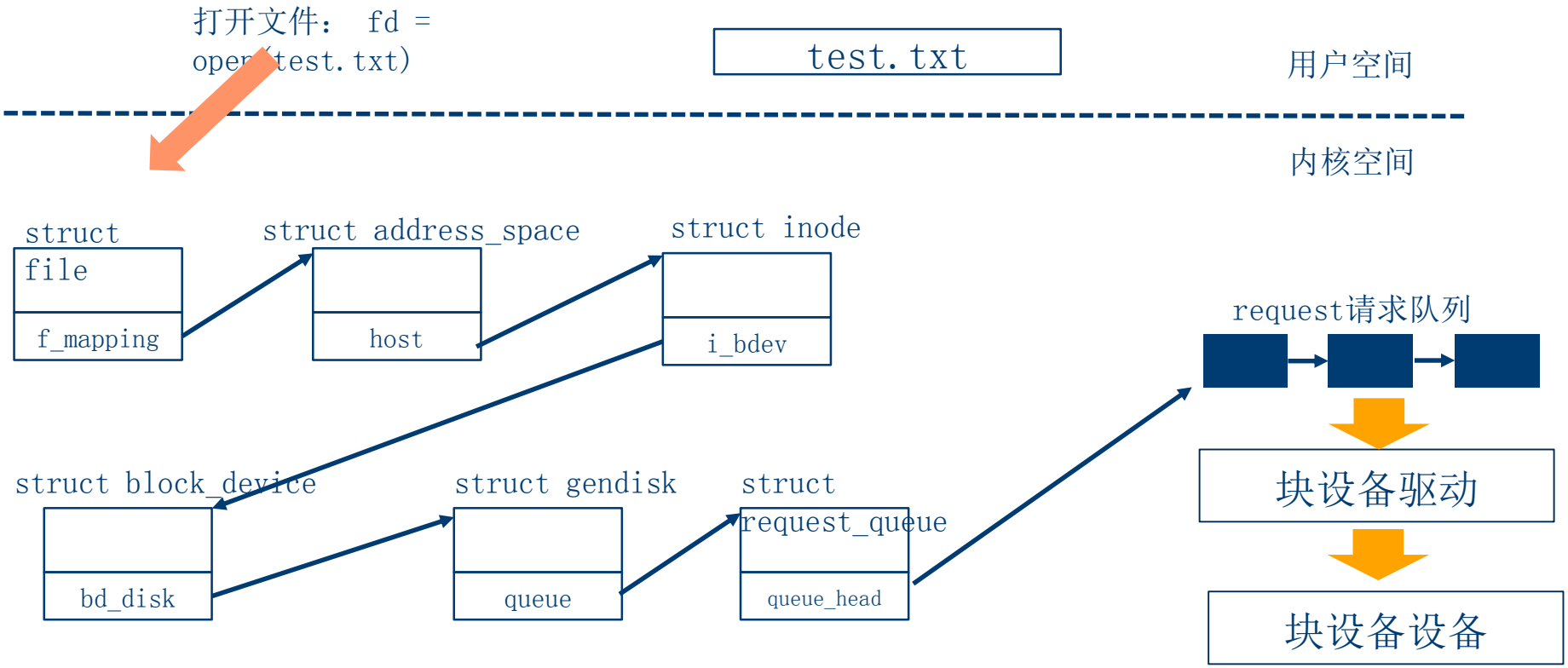


bio数据结构

- 块设备数据传输的数据结构



块设备子系统常用数据结构的关系图



块设备驱动程序常用API

- 注册函数
  - ✓ int register_blkdev(unsigned int major, const char *name)

- 注销函数
  - ➢ void unregister_blkdev(unsigned int major, const char *name)

块设备驱动程序常用API

- 初始化请求队列：
  - ✓ `struct request_queue *blk_init_queue(request_fn_proc *rfn, spinlock_t *lock)`

- 注销请求队列
  - ✓ `void blk_cleanup_queue(struct request_queue *q)`

块设备驱动程序常用API

- 分配gendisk对象：
  - ✓ struct gendisk *alloc_disk(int minors)
- 注册gendisk对象
  - ✓ void add_disk(struct gendisk *disk)
- 注销gendisk对象
  - ✓ void del_gendisk(struct gendisk *disk)

块设备驱动程序常用API

- 初始化一个BIO：
  - ✓ void bio_init(struct bio *bio)

- 提交BIO：
  - ✓ void submit_bio(int rw, struct bio *bio)

实 验

实验：写一个简单的ramdisk设备驱动

- 使用系统的物理内存来实现一个块设备，ramdisk

- 实验要求
  1. 写一个简单的ramdisk设备驱动。可以使用ext4的格式化工具来格式化这个ramdisk。
  2. 在这个设备驱动中，实现HDIO_GETGEO的ioctl命令，可以读出ramdisk的hd_geometry参数，比如有多少磁头，多少个柱面，多少个扇区等信息。写一个简单的用户空间的测试程序来读取hd_geometry参数。

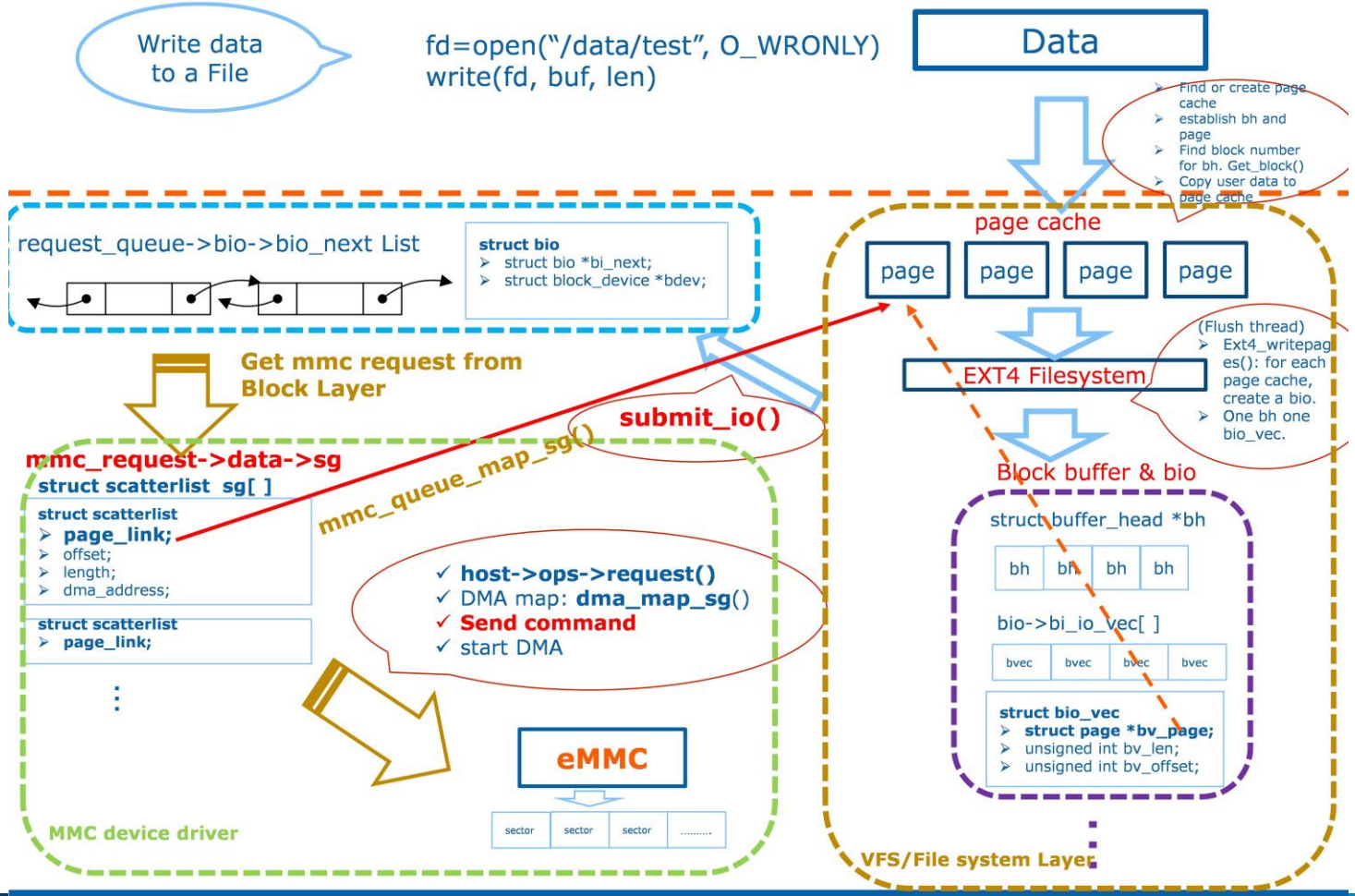
进 阶 思 考

进阶思考题

- 假设系统使用eMMC作为存储介质，文件系统采用ext4。使用C语言的open函数打开一个文件，然后使用write函数来对写入内容，比如字符串“hello world”

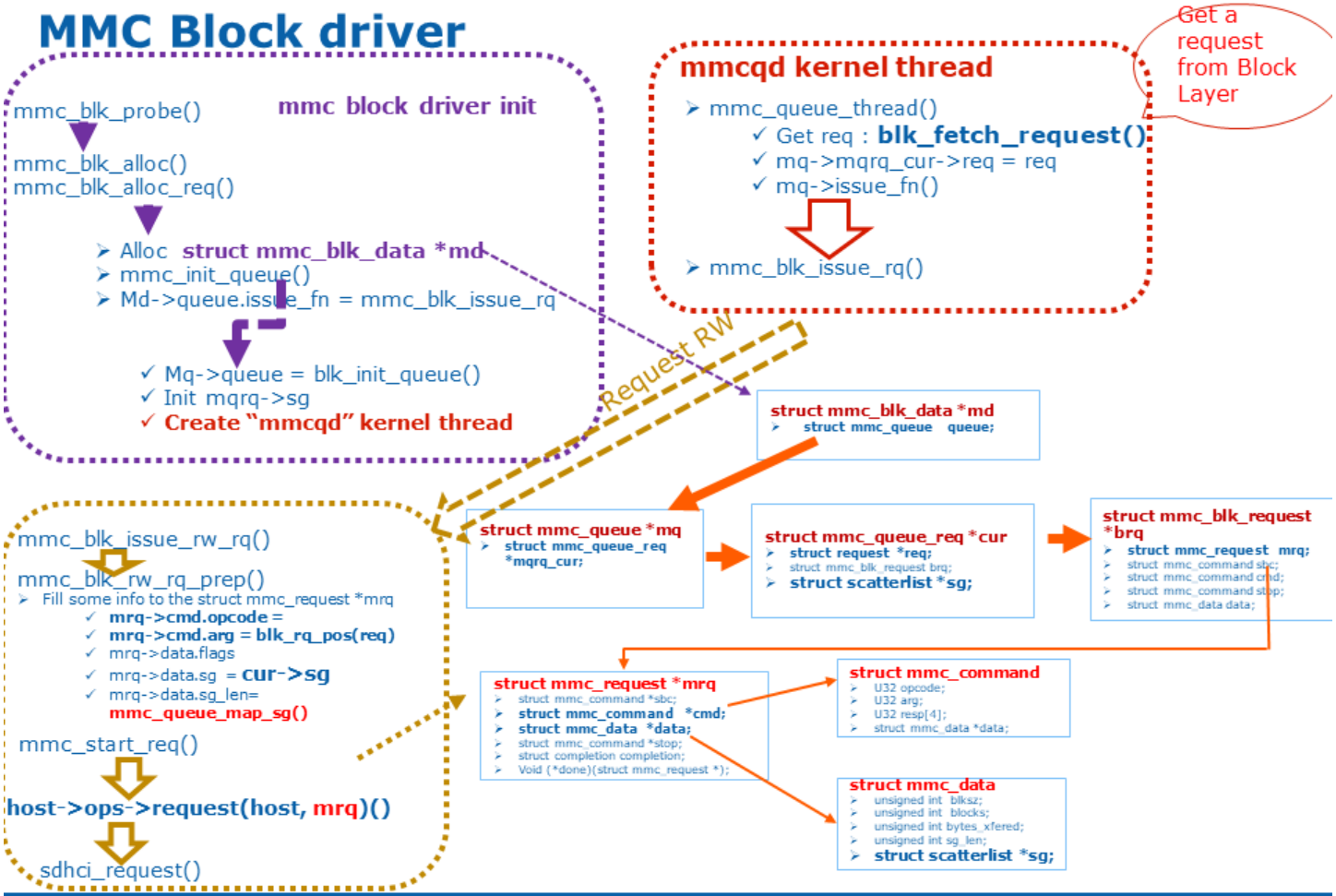
请阅读Linux内核代码，跟踪从用户空间write函数到eMMC存储设备了解写入内容经历了内核中哪些内核模块，期间使用了哪些主要的数据结构，并请画出其代码workflow。

Backup



Figo.zhang <figo1802@126.com>

MMC Block driver



Card Identification (卡检测过程)

When eMMC init

mmc_detect_change()
mmc_rescan()

mmc_rescan_try_freq(): try to scan card by 100kHz/200K/300K/400K

mmc_attach_mmc()

- Send CMD1 get OCR
- Set lowest voltage depend on OCR
- mmc_init_card()
  - Set ORC bit30, sector mode
  - Send CMD2 fetch CID
  - Set card RCA and open drain mode
  - **Send CMD9 fetch CSD, decode it.**
  - **Send CMD8 fetch EXT_CSD, decode it**
  - Active high speed.
    - ✓ Use CMD 6 to Set HS_TIMING= 1, enable high speed mode
    - ✓ mmc_set_timing() => mmc_set_ios() => host->ops->set_ios()
  - Set max clock: ext_csd.hs_max_dtr => mmc_set_clock()
  - Activate wide bus and DDR mode
    - ✓ select PowerClass for the current bus width. Select pwrclass_val by vdd (OCR), set pwrclass_val to EXT_CSD[187]
    - ✓ Set bus width to EXT_CSD[183], and mmc_set_bus_width()
    - ✓ Enable DDR mode: DDR PowerClass, DDR bus width, mmc_set_timing(), mmc_set_bus_width()
- mmc_add_card()
  - ✓ device_add()