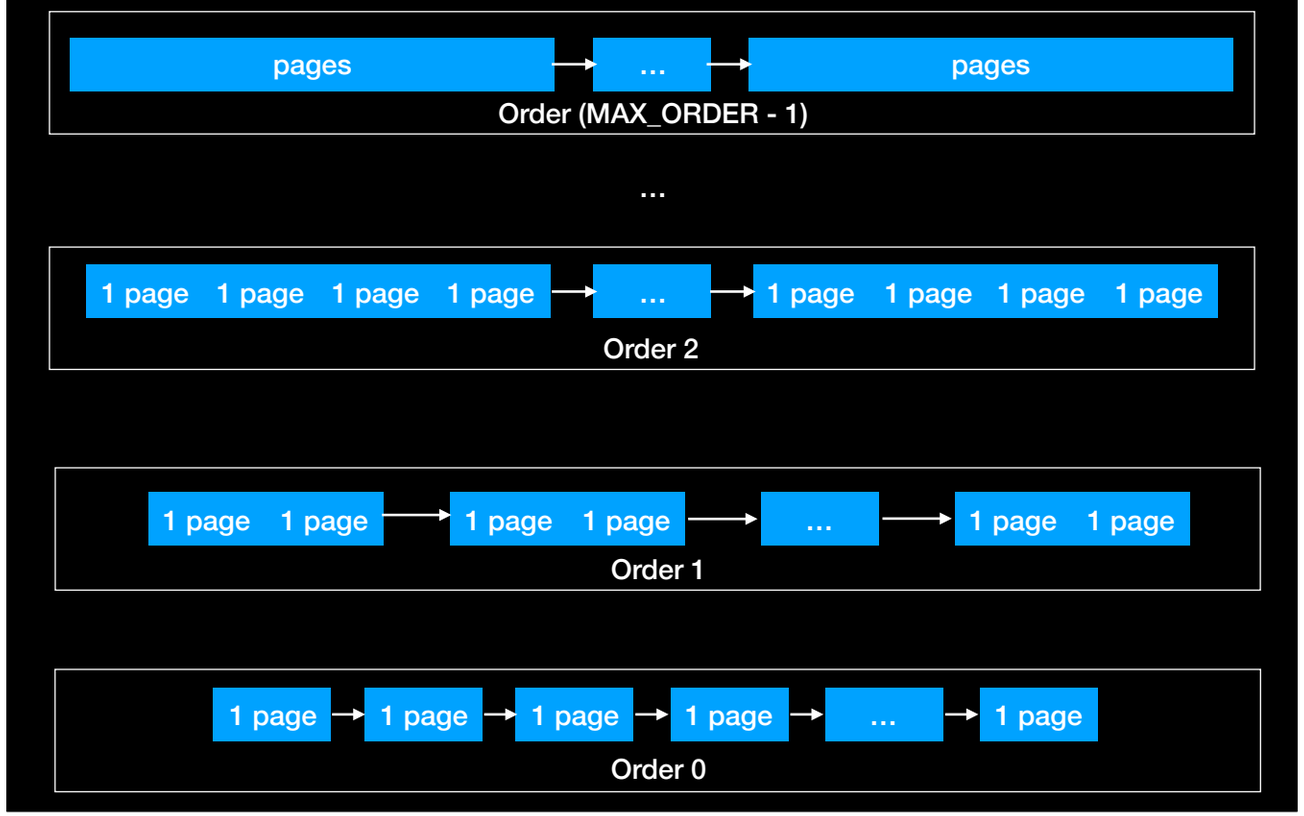
The implementations of anti pages fragmentation in Linux kernel

2018.06

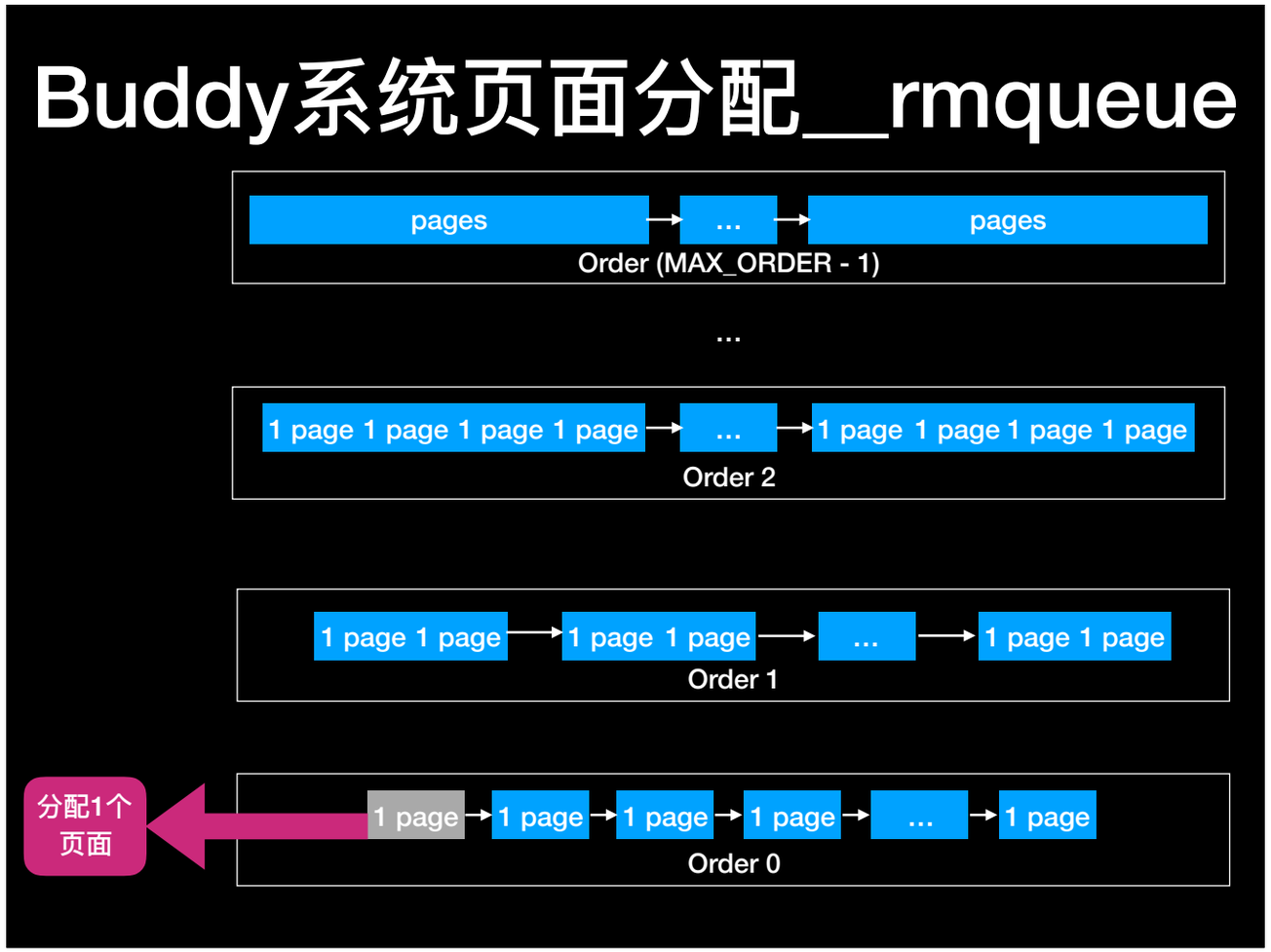
朱辉 teawater@hyper.sh

Buddy系统一个ZONE内空闲页基本结构



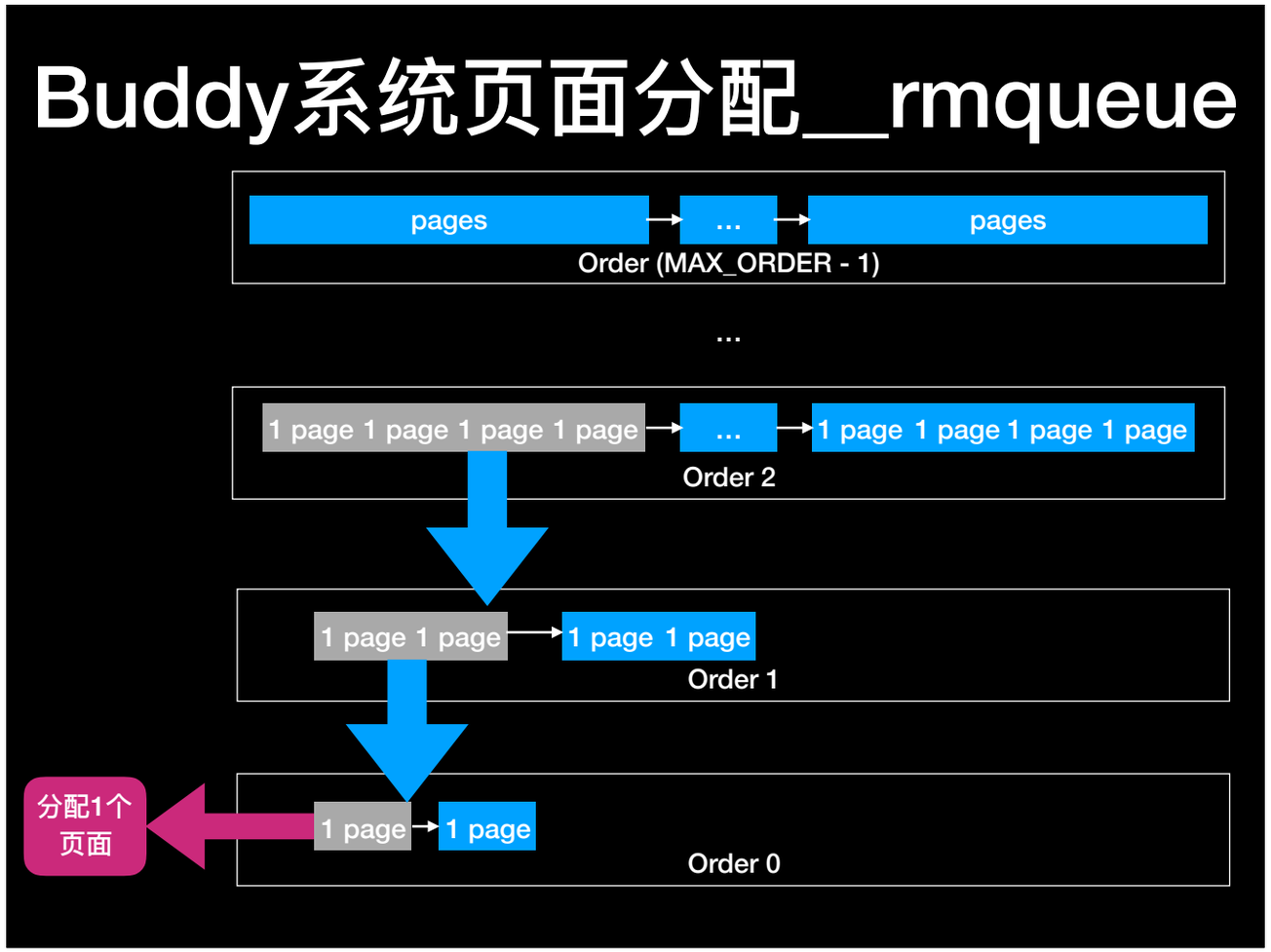
在2.6.23以及其之前版本Buddy系统基本就是这个结构，这个结构用来存储空闲页面，而现在内核也是在这个结构之上进行扩展。
每个内存zone有2的0次方开始到2的MAX_ORDER-1次方的MAX_ORDER个列表，每个列表都串起相应尺寸的空闲PAGE。
MAX_ORDER默认值为11。

Buddy系统页面分配__rmqueue



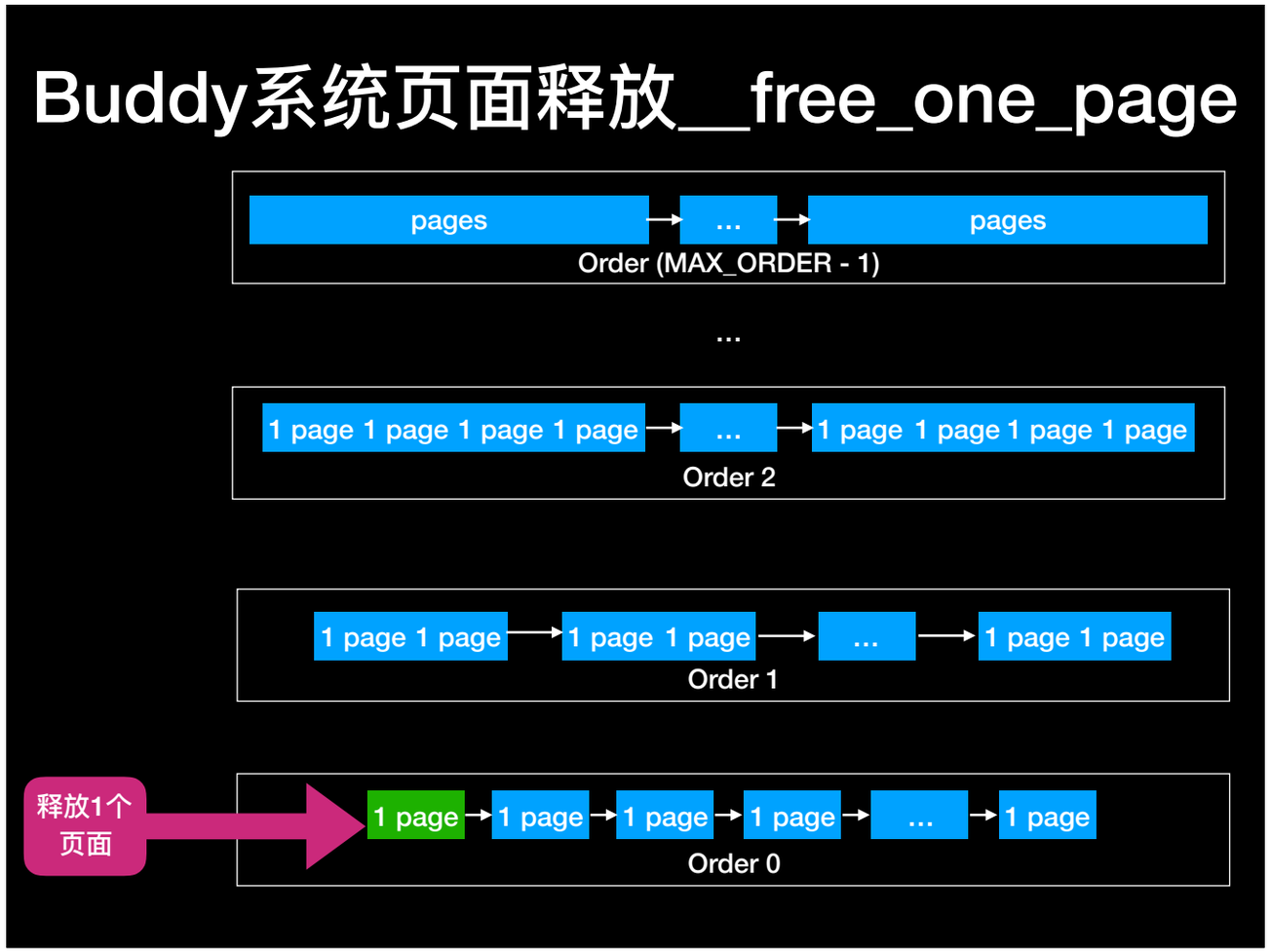
当页面分配的时候，先根据请求页面的大小到相应的链表去申请。

Buddy系统页面分配__rmqueue

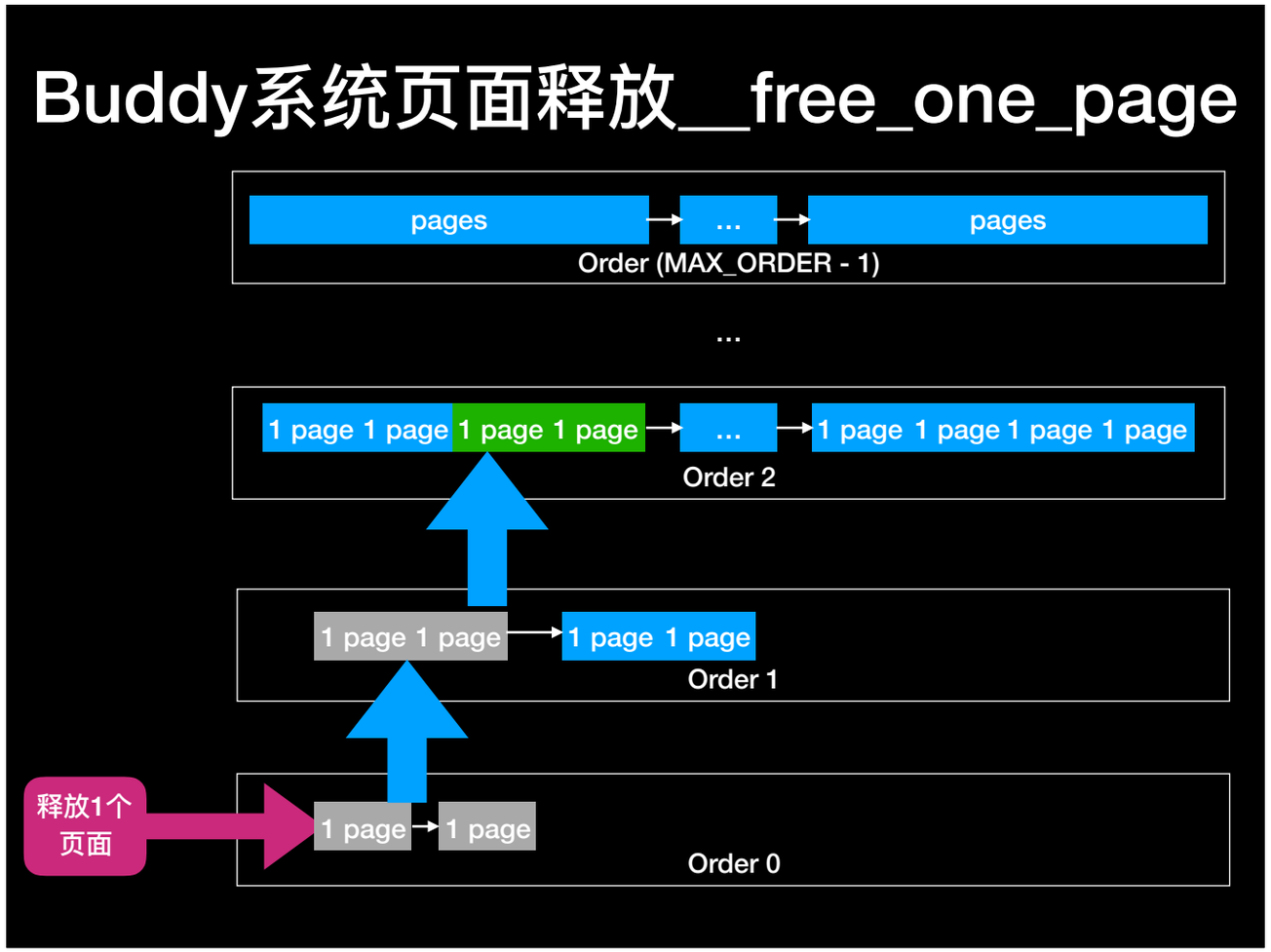


如果在相应尺寸的链表上没有页面，则向更大尺寸的链表去查找。
一旦查找到可用的空闲页面则拆成两半，一半存入其order-1链表，一半交给order-2链表来拆开或者分配出去。

Buddy系统页面释放__free_one_page

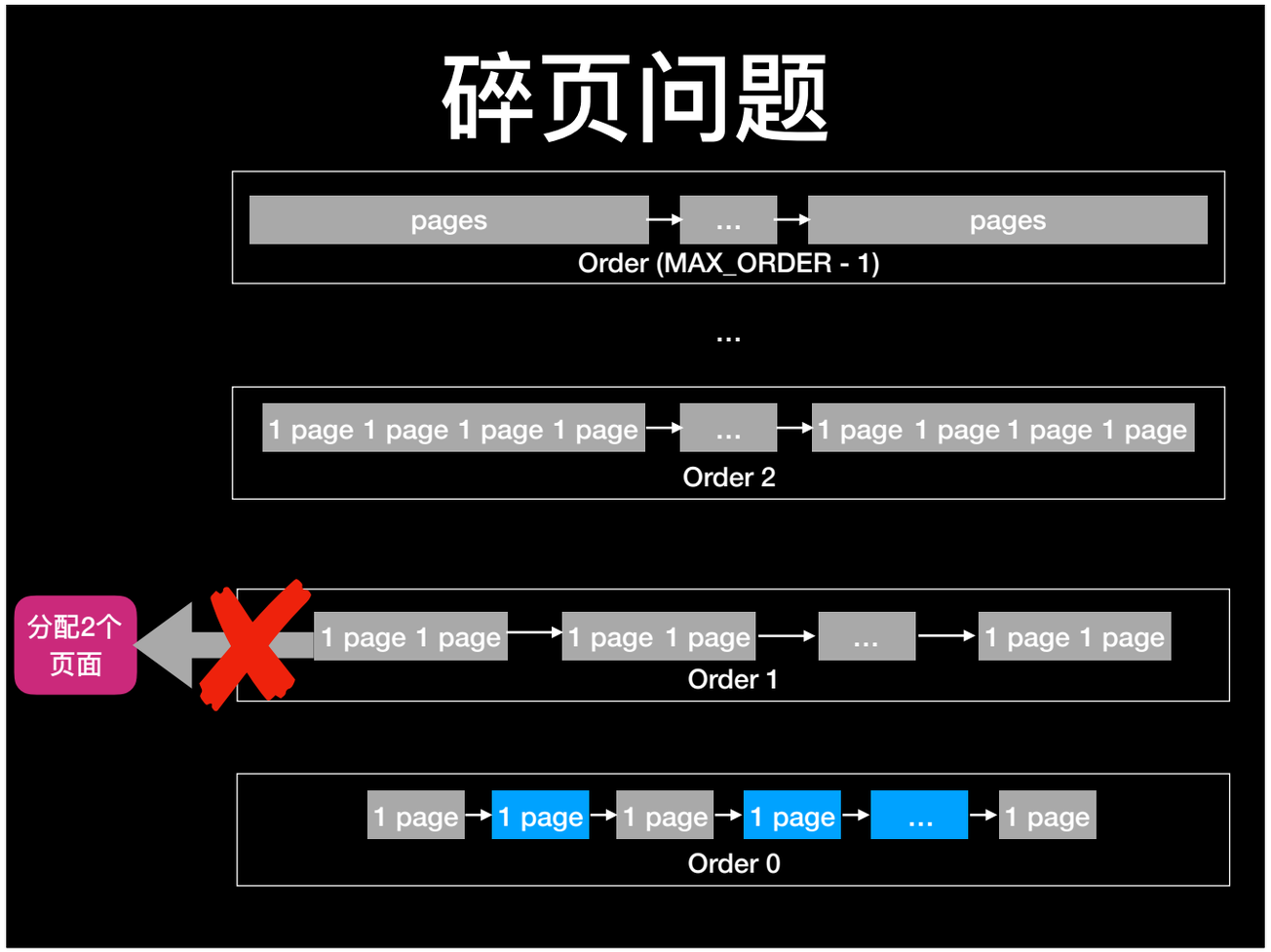


释放页面的时候就直接把页面放回空闲列表，如果之前拆成两半的页面另一半已经被分配出去，则直接返回。
注意这时候内核代码释放页面还没有冷热之分，所以都是放回链表头部。



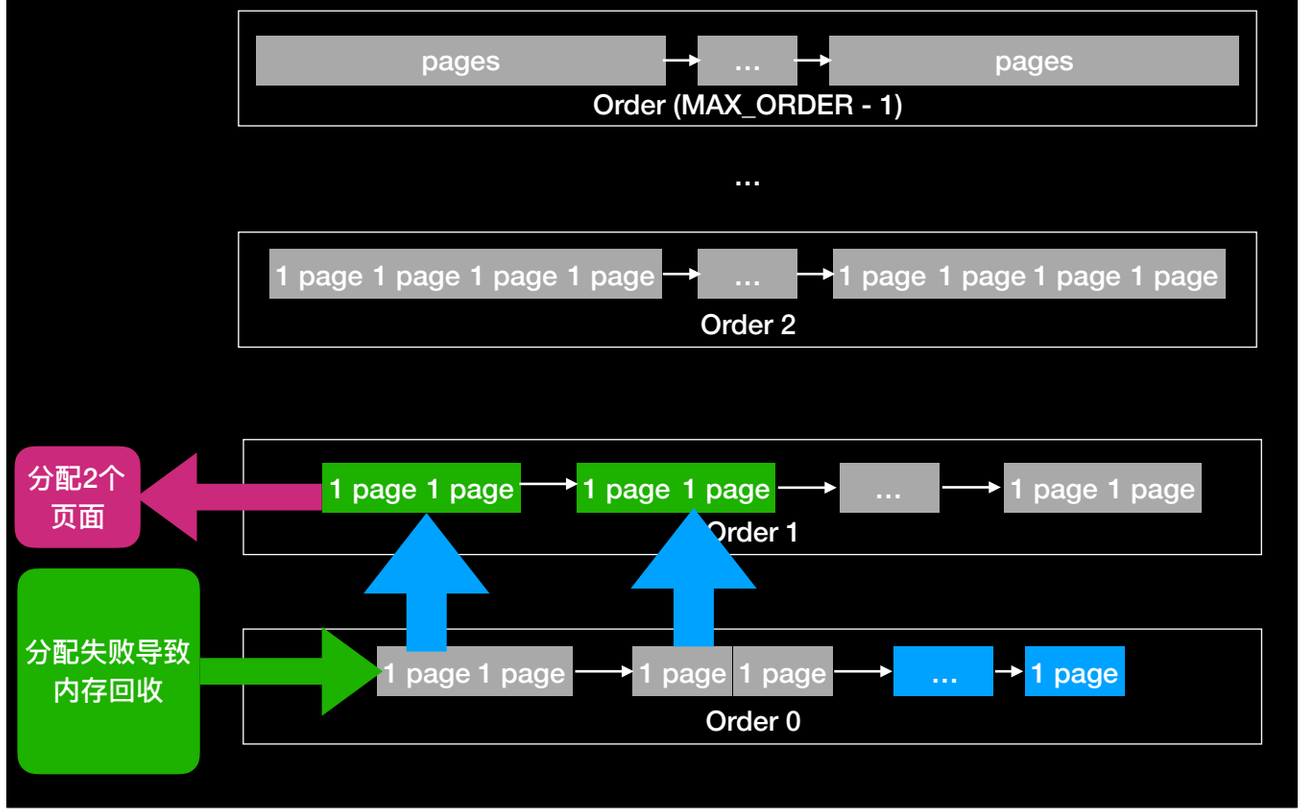
释放页面的时候就直接把页面放回空闲列表，如果其之前拆开的另一半页面也没有被分配，则组合成双倍页面交给更大的列表，然后依次循环，直到不能再循环或者已经到头。

碎页问题



要分配order超过1的页面的时候，虽然系统中有超过order 1数量的页面，但是并没有一个完整的一个相应ORDER的页面可用，这就是碎页问题。
为解决这个问题，系统采取了若干办法，又产生了若干问题，让碎页问题成为了比较综合的一个问题。

通过内存回收解决分配问题



最原始的处理方法。

一些措施

- 因为总体需求不强，碎页问题并不严重。
- 2007年左右开始比较关注碎页问题，因为hugepage用的多?
- 页面预留
- slub_max_order
- 用vmalloc换get_pages

内存回收lumpy reclaim

- 在页面回收isolate_lru_pages也就是选择要回收页面的时候，根据要分配的order，isolate一个页面成功后，将同buddy页面也isolate，最终回收，这样就更容易给系统增加某order的空闲页面。

- 只以位置为回收标准，而不是active和inactive这种更不容易影响性能的标准，对系统影响比较大。
  且因为compaction功能的出现，2012年被去掉。

比如需要4个页面，成功一个就继续把后面3个也isolate出来。

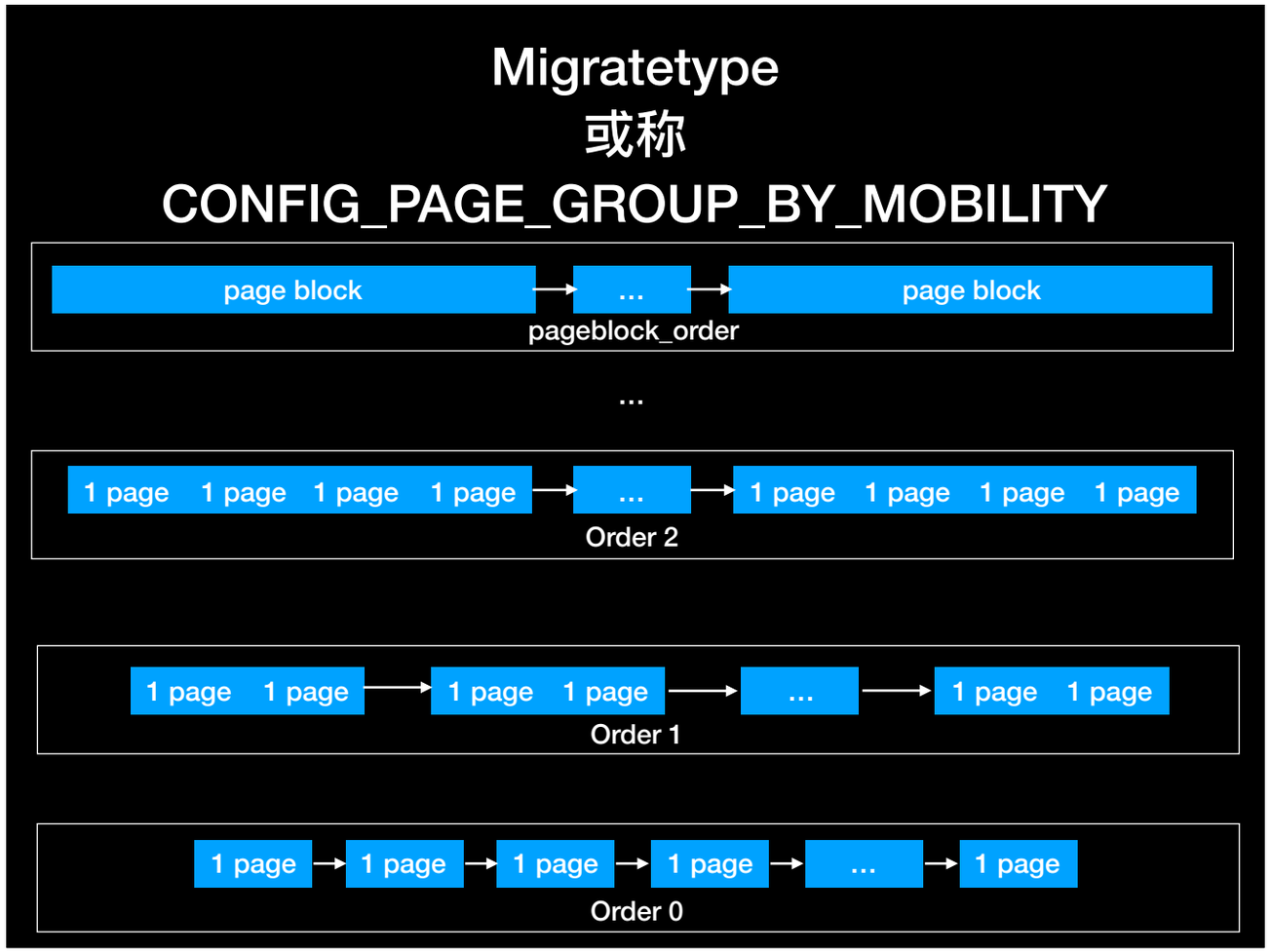
内存回收
PAGE_ALLOC_COSTLY_ORDER

- 一个宏，默认为3。
- 分配超过这个值的order的页面，分配中引起的内存回收就会做更重的操作。
- 给内存申请也提供了标准。
- 和lumpy reclaim同一个commit，commit里都没提，好像买东西随手给的小礼品，但是存活到了现在。

ZONE_MOVABLE

- 虚拟ZONE，在初始化的时候通过kernelcore或者movablecore指定大小。
- 只能被(__GFP_HIGHMEM | __GFP_MOVABLE)申请使用。
- commit上介绍的作用是分开可移动页面和不可移动页面，从而抗碎页。
- 不过感觉更主要的作用是因内存使用者都是100%可迁移的所以更倾向方便memory hotplug。
- 打开hugepages_treat_as_movable后hugepage可从zone_movable分配内存。
  此选项于2018年1月被删除，因为hugepage的不可移动性会影响ZONE_MOVABLE memory hotplug。

hugepage分配大量连续页面导致碎页问题，所以被丢到不太容易有碎页问题的的ZONE里，既减少了unmovable页面的分配的影响，又增加了hugepage的分配成功率。
因为其他抗碎页机制的出现，此功能就更偏向memory hotplug。
当然从这个提交可以感觉到页面分类已经是呼之欲出了。



前面这些其实现在都不太常见，作用不大或者就已经被去掉了，后面开始这些我觉得是比较主流的抗碎页方向。

Migratetype最里面还是之前buddy的结构，最大order中每一个连续页被称为page block。

另外CONFIG_PAGE_GROUP_BY_MOBILITY这个配置选项只存在过很短的时间，出现后很快就被拿掉了。

每个pageblock根据其类型不同分成几个列表，分配内存的时候，根据申请页面类型的不同，向相应的列表中申请页面，而释放的时候也会根据其所属pageblock回到相应列表。

__rmqueue当某个migratetype列表中没有足够页面时__rmqueue_fallback

```
static int fallbacks[MIGRATE_TYPES][4] = {
    [MIGRATE_UNMOVABLE]   = { MIGRATE_RECLAIMABLE, MIGRATE_MOVABLE,   MIGRATE_TYPES },
    [MIGRATE_RECLAIMABLE] = { MIGRATE_UNMOVABLE,   MIGRATE_MOVABLE,   MIGRATE_TYPES },
    [MIGRATE_MOVABLE]     = { MIGRATE_RECLAIMABLE, MIGRATE_UNMOVABLE, MIGRATE_TYPES },
#ifdef CONFIG_CMA
    [MIGRATE_CMA]         = { MIGRATE_TYPES }, /* Never used */
#endif
#ifdef CONFIG_MEMORY_ISOLATION
    [MIGRATE_ISOLATE]     = { MIGRATE_TYPES }, /* Never used */
#endif
};
```

当向自己类型列表申请页面不足的时候，内核的处理用内核代码展示最容易。前面是发生页面不足的类型，后面的列表为当此类型没有可分配的页面的时候，依次扫描这几种类型的列表，找到合适的列表后开始评估：

可能性1，只移动一个页面。

可能性2，全部空闲页面移动要要分配的列表。同时评估其空闲和匹配要分配页面是否超过一半，如果一半则整个页块设置为要分配的类型。

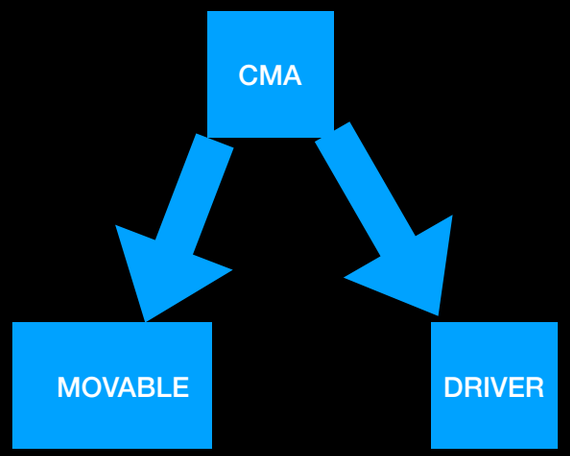
这个移动一个或者若干页的操作被称为steal，这里的实现也是巨坑，后续我会谈到。

Migratetype简介

- MIGRATE_UNMOVABLE，不可migration的页面，一般是内核分配来自己用的页面。
- MIGRATE_MOVABLE，可migration的页面，一般是应用层使用，分配时候会指定__GFP_MOVABLE。
- MIGRATE_RECLAIMABLE，可回收的页面，主要是slab分配的时候指定了SLAB_RECLAIM_ACCOUNT的kmem，指定这个内存都自带shrink_slab接口，内存回收的时候可以被释放掉，比如inode。
- MIGRATE_ISOLATE，其中页面不会被分配，用来帮助isolate页面。isolate页面的时候会将页块先设置为isolate防止其被释放。
- ~~MIGRATE_RESERVE，把min_free_kbytes大小的内存存于特殊的group，作为全部fallback的备份。最终被MIGRATE_HIGHATOMIC反戈一击，被替换掉。~~
- 还有两个类型比较复杂，需要单独来讲。

MIGRATE_CMA

- 用来帮助嵌入式设备驱动分配某个特定位置和大块连续内存。基本原理可以见我2014年的演讲《Buddy和CMA简介》。
- 存在一堆坑人问题，用了很可能不如不用。
- 从2015年就开始喊要弄成类似ZONE_MOVABLE的虚拟ZONE（不能解决其大部分问题），现在还没提交。主要意义是有效阻止了修正问题PATCH的提交。

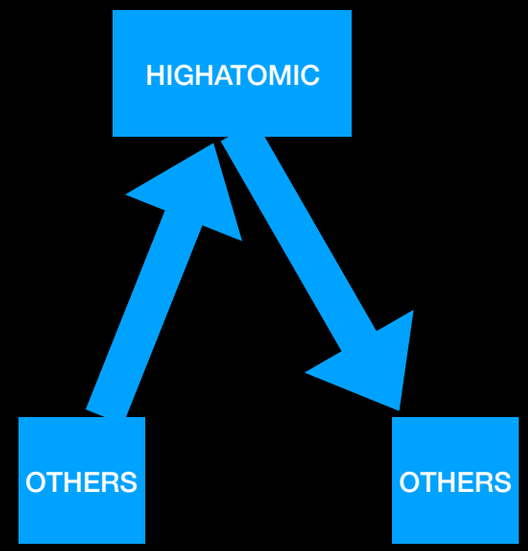


一种特殊的MIGTATE_MOVABLE，当MIGTATE_MOVABLE没有页的时候，从这里分配。不能被转为其他类型。
驱动需要的时候就迁移成isolate页面给驱动使用。

首先说MIGTATE_MOVABLE没有页的问题，这里没有页系统基本处于濒临崩溃状态，这时候再分CMA其实对系统帮助不大。
而如果真的CMA当成MIGTATE_MOVABLE用起来，要分配的时候，因为应用程序用页都是map出来的，关键时刻很可能unmap失败，从而导致分配失败。
为了提高驱动分配成功率还会调用很多很重的回收操作，虽然很可能无效但是会影响系统性能。
同时因为CMA存在，还造成了内存统计数字的不准确，影响内存分配回收的效率。

MIGRATE_HIGHATOMIC

- MIGRATE_HIGHATOMIC，最初版本存在过，存在于普通fallback列表，很快因为被MIGRATE_RESERVE取代。于2015年以更灵活形式回归。
- 当有原子连续页面分配时，自动优先从MIGRATE_HIGHATOMIC分配。
- 以前原子连续页面分配依赖MIGRATE_RESERVE和High-order watermark一起来保证，因为会引起普通连续页面watermark检查失败，所以被替换掉。
- MIGRATE_HIGHATOMIC按需reserve，同时拿掉High-order watermark保证了连续页面分配率。
- reserve_highatomic_pageblock
- unreserve_highatomic_pageblock
- https://lkml.org/lkml/2017/9/26/232

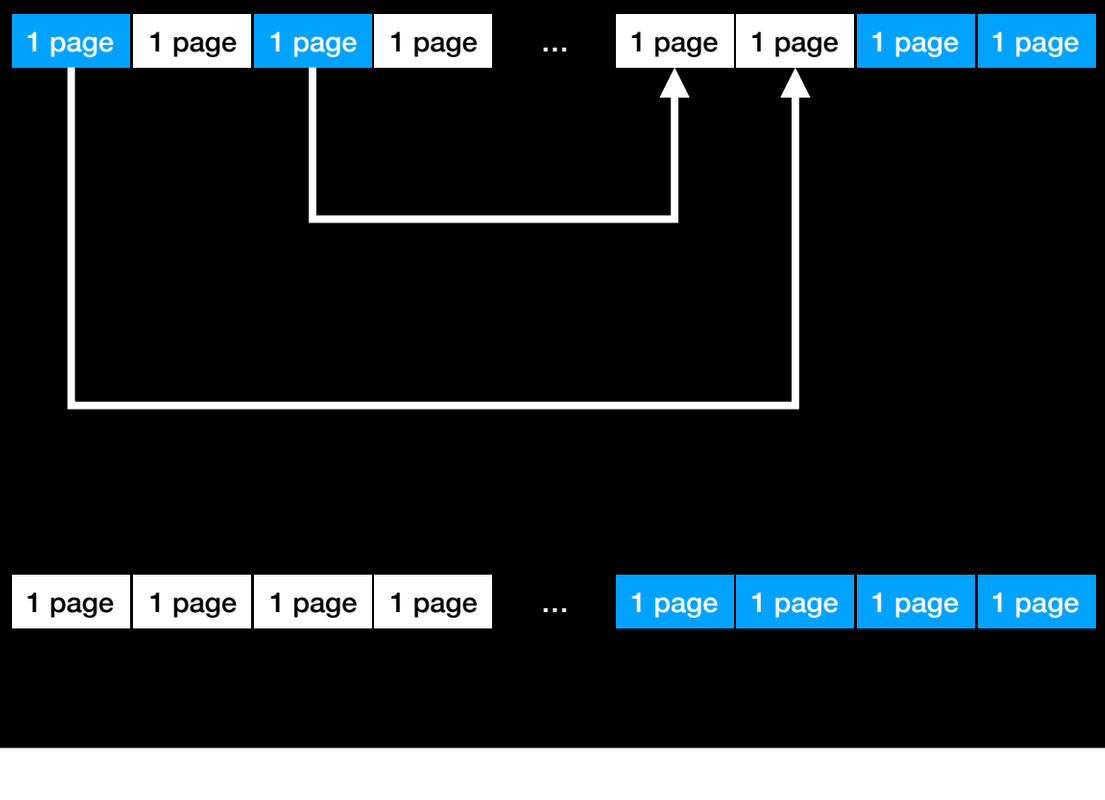


此类型页面的来源不是steal，而是当原子大页面分配成功时，如果是从非MIGRATE_HIGHATOMIC分配，则用reserve_highatomic_pageblock转为MIGRATE_HIGHATOMIC页面。注意系统对MIGRATE_HIGHATOMIC数量是有限制的，为可管理页面数量的百分之一加一个页块。

当系统页面不足导致页面分配失败时，会调用unreserve_highatomic_pageblock将MIGRATE_HIGHATOMIC转为其他类型。

最后这个是我对HIGHATOMIC使用的抗碎页扩展，主要是扩展了对其的使用，对一些问题在本地测试有不错收益。

compaction碎页整理



2010年提交。
在分配页面失败的时候，会被调用到。

基本思路把前面的被用的可迁移页面移动到后面空闲页面，从而产生连续空闲页块。

因为页面中数据被迁移到新位置，所以不会有采取的内存回收导致的那么大的性能问题，而且因为目标更明确，得到的连续页面的成本也更低。
另外ANON页面回收是需要SWAP的，而这里不需要。

compaction碎页整理的一些优化

- 在紧急情况下，来源和目标页块可更多的使用非MIGRATE_MOVABLE页块。
- 在紧急情况下，迁移（isolate_migratepages_block）失败后的处理变严格。（持怀疑态度，建议在用的可以拿掉测试一下）
- kcompacted

这里和前面介绍__rmqueue_fallback，因为其实现的特点，大部分时刻都是把MIGRATE_MOVABLE迁移到MIGRATE_UNMOVABLE上。在内存负载比较高并且内存量又不是很够的系统上，比如安卓上，其运行时间越长MIGRATE_MOVABLE就越少。
而这时因为早期compaction的实现，迁移来源和目标更倾向使用MIGRATE_MOVABLE页块，迁移目标甚至只能接受MIGRATE_MOVABLE页块（因为能迁移的页面都是movable）。这样系统中就会产生大量的碎页无法被处理。
这也是导致系统因为碎页过多反复调用内存回收和碎页整理，影响系统性能。
所以很有一部分非upstream的patch都是努力不让MIGRATE_MOVABLE页块转为MIGRATE_UNMOVABLE的。
我认为处理这个问题最好的办法就是，重启系统。
另外就是在紧急情况下，来源和目标页块可更多的使用非MIGRATE_MOVABLE页块。

紧急情况下迁移一旦有失败就会马上放弃整个页块，commit认为可以减少没价值的扫描次数，这个我持怀疑态度。
小龙虾出肉率才15%，一样成为夜宵之王，至少该加个开关。

将compaction的功能从kswapd里拿出来单做了kcompacted，因为kswapd主要的目标是内存回收，把碎页整理抽出来有利于更好的精准处理碎页整理的问题。

Kernel page movable

- 内存需求量大的内核内功能，会造成碎页。比如A64的安卓上的ZRAM的后端ZSMALLOC。
- 给这部分页面增加相应接口，令其可以migration。这些页面可以分配为MIGRATE_MOVABLE。
- ZSMALLOC，可以见我的演讲《ZRAM那点事》。
  77ff465799c60294e248000cd22ae8171da3304c
  另外ZSMALLOC因为migration成功率高，作为CMA后端也是不错的选择。
- F2FS

VMAP_STACK

- 打开此选项，alloc_thread_stack_node改alloc_pages_node为vmalloc，不再需要连续页面。
- 除了用来侦测内核栈溢出，其实也可以抗碎页。
- 在内存不足，且内核栈巨大的系统上，连续的内核栈分配既是碎页产生的原因，又是碎页问题的受害者。
- 而且VMAP_STACK backporting难度低，比之前介绍的难度都低几个等级。
- 缺点是因为内核栈变成vmalloc出来的，很多驱动会受到影响。

谢谢！ 问题?

我的微信公众号 茶水侃山