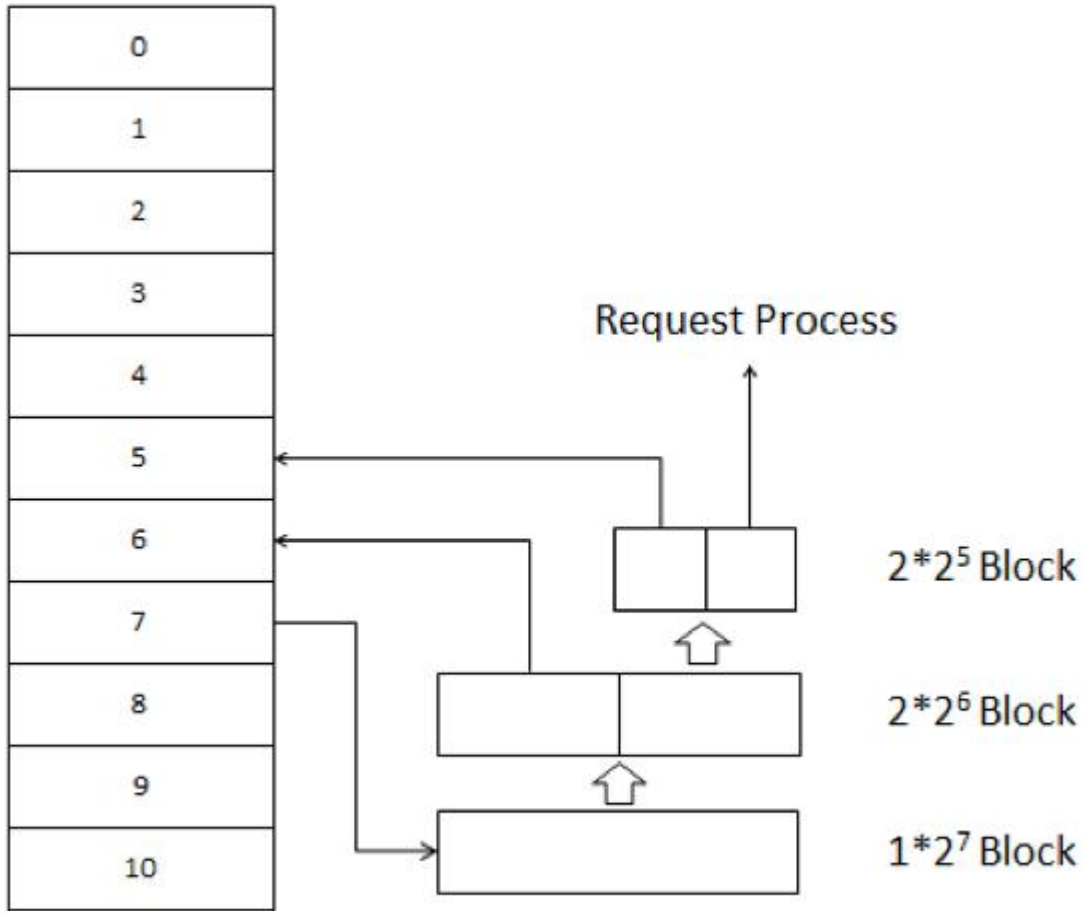
页框和伙伴算法



定义：内核使用struct page结构体描述每个物理页，也叫页框。

场景：内核在很多情况下，需要申请连续的页框，而且数量不定，比如4个，5个，9个等。

实现：Linux把所有的空闲页框分组为11个块链表，每个链表上的页框块是固定的。在第i条链表中每个页框块都包含2的i次方个连续页。

注意：系统中每个页框块的第一个页框的物理地址是该块大小的整数倍。例如：大小为16个页框的块，其起始地址是16*2^12的倍数。

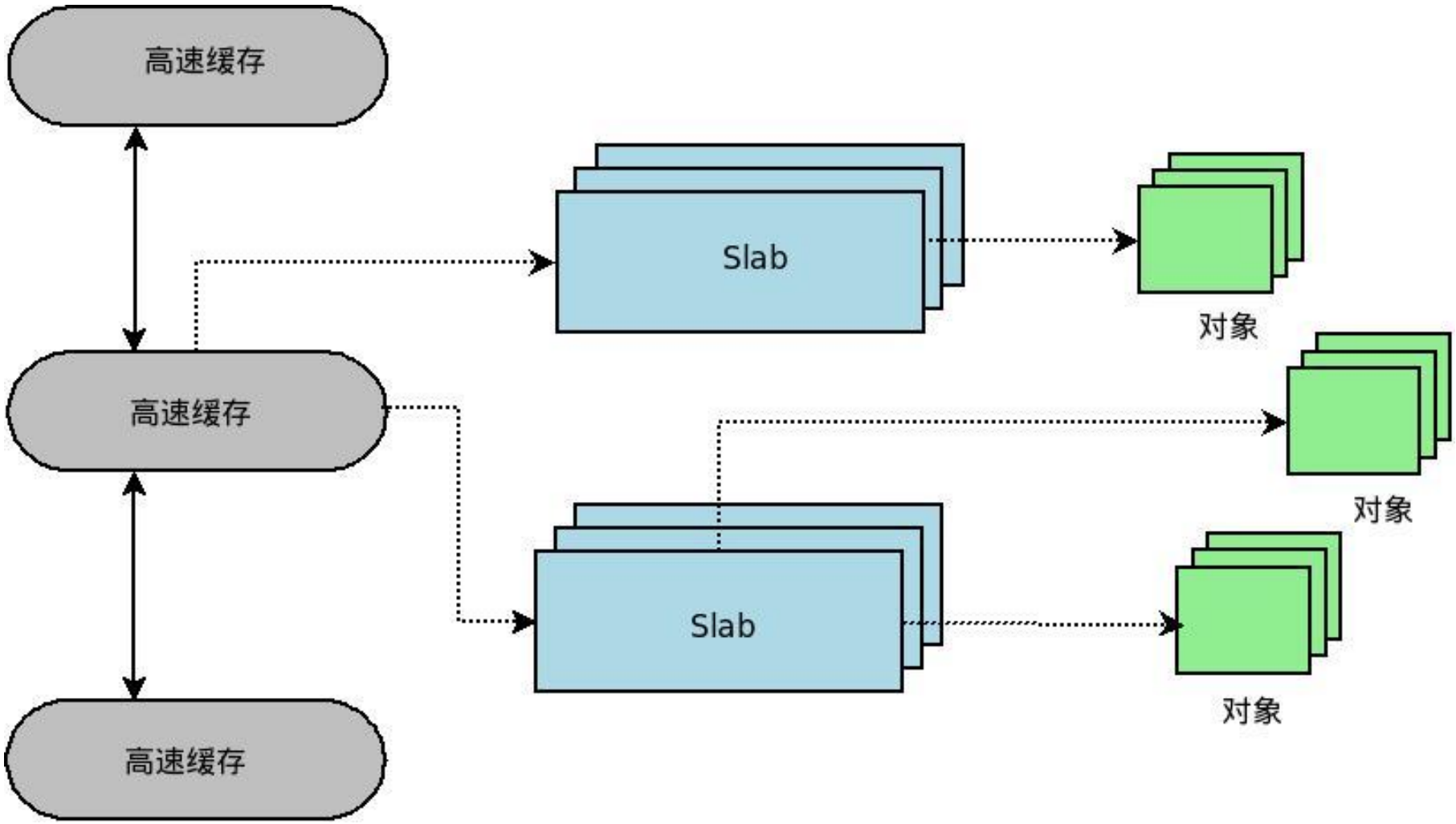
页框操作 alloc_pages(), page_address()

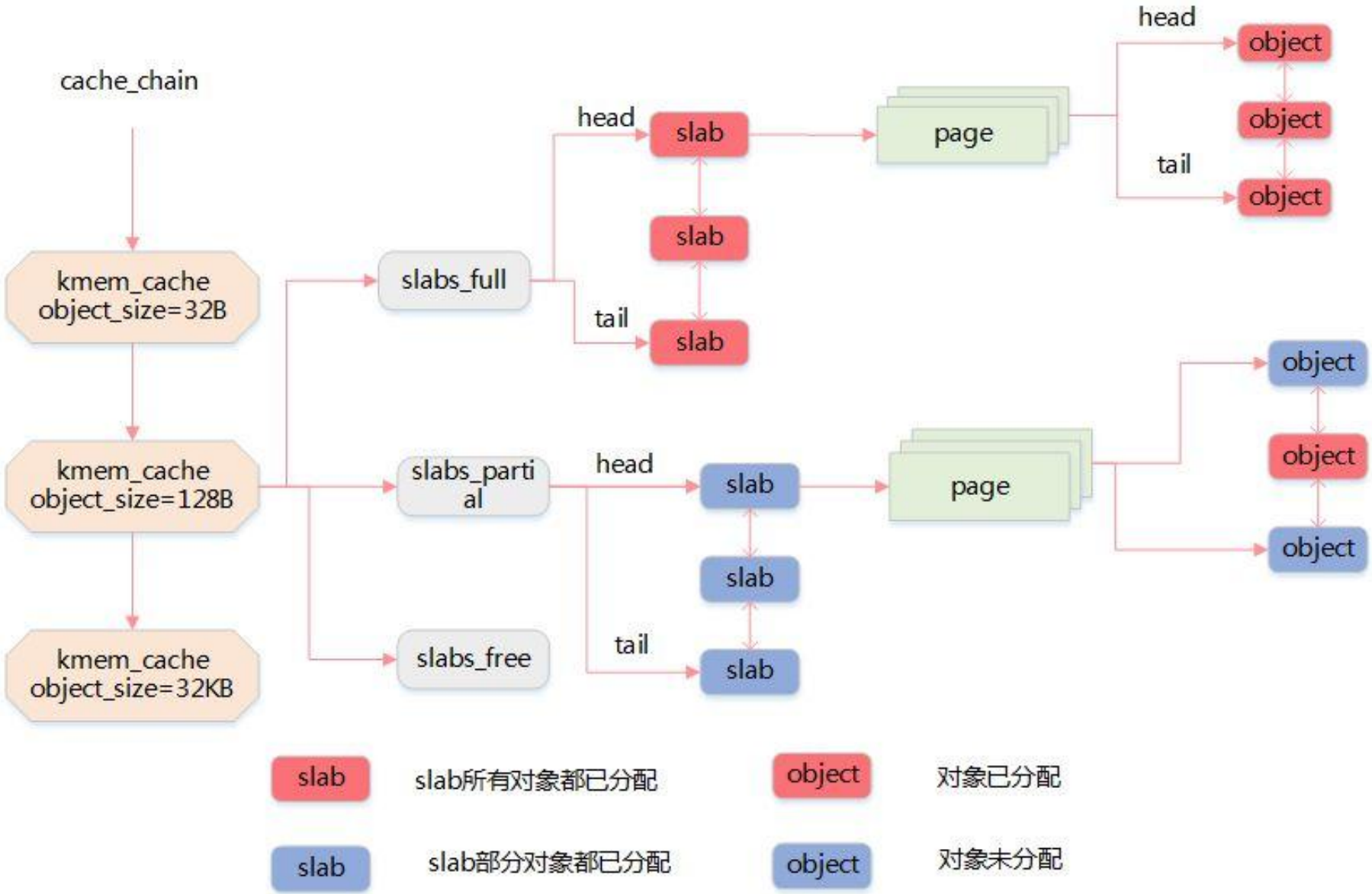
static inline struct page * **alloc_pages**(gfp_t gfp_mask, unsigned int order)
分配2^order个连续的物理页，并返回一个指针，指向第一个页的page结构体

void ***page_address**(const struct page *page)
返回page页面所映射的的虚拟地址

slab实现了内存分配和管理

- slab层把不同的对象划分为所谓的高速缓存（cache）组，其中每个高速缓存都存放不同类型的对象；
- 每种对象类型对应一个高速缓存（cache）；
- 例如一个高速缓存存放task_struct结构体，而另外一个高速缓存存放struct inode结构体；
- slab由一个或者多个物理上连续的页组成。每个高速缓存由多个slab组成。


高速缓存
高速缓存
高速缓存
Slab
Slab
对象
对象
对象


cache_chain
kmem_cache
object_size=32B
kmem_cache
object_size=128B
kmem_cache
object_size=32KB
slabs_full
slabs_parti
al
slabs_free
head
slab
slab
tail
slab
page
head
object
object
tail
object
head
slab
slab
tail
slab
page
object
object
object
slab
slab所有对象都已分配
object
对象已分配
slab
slab部分对象都已分配
object
对象未分配

slab机制要解决的问题

1.减少伙伴算法在分配小块连续内存时所产生的内部碎片；

2.将频繁使用的对象缓存起来，减少分配、初始化和释放对象的时间开销。

3.通过着色技术调整对象以更好的使用硬件高速缓存；

slab/slob/slub

- slab
- slob
- slub

slab -- 高速缓存

struct kmem_cache *
kmem_cache_create(const char *name, size_t size, size_t align,
unsigned long flags, void (*ctor)(void *))

```
root@ubuntu:/home/jinxin/app# cat /proc/slabinfo
slabinfo - version: 2.1
# name            <active_objs> <num_objs> <objsize> <objperslab> <pagesperslab> : tunables <limit> <batchcount> <sharedfactor>
 : slabdata <active_slabs> <num_slabs> <sharedavail>
ext4_groupinfo_4k    224    224    144   56    2 : tunables    0    0    0 : slabdata      4      4      0
UDPLITEv6              0      0   1088   30    8 : tunables    0    0    0 : slabdata      0      0      0
UDPv6                 30     30   1088   30    8 : tunables    0    0    0 : slabdata      1      1      0
tw_sock_TCPv6         58     58    280   58    4 : tunables    0    0    0 : slabdata      1      1      0
request_sock_TCPv6     49     49    328   49    4 : tunables    0    0    0 : slabdata      1      1      0
TCPv6                 15     15   2112   15    8 : tunables    0    0    0 : slabdata      1      1      0
kcopyd_job             0      0   3312    9    8 : tunables    0    0    0 : slabdata      0      0      0
dm_uevent              0      0   2632   12    8 : tunables    0    0    0 : slabdata      0      0      0
cfq_queue              0      0    232   70    4 : tunables    0    0    0 : slabdata      0      0      0
bsg_cmd                0      0    312   52    4 : tunables    0    0    0 : slabdata      0      0      0
mqueue_inode_cache     36     36    896   36    8 : tunables    0    0    0 : slabdata      1      1      0
fuse_request          40     40    400   40    4 : tunables    0    0    0 : slabdata      1      1      0
fuse_inode            42     42    768   42    8 : tunables    0    0    0 : slabdata      1      1      0
ecryptfs_key_record_cache      0      0    576   56    8 : tunables    0    0    0 : slabdata      0      0      0
ecryptfs_sb_cache      0      0   1152   28    8 : tunables    0    0    0 : slabdata      0      0      0
ecryptfs_inode_cache      0      0   1024   32    8 : tunables    0    0    0 : slabdata      0      0      0
ecryptfs_auth_tok_list_item      0      0    832   39    8 : tunables    0    0    0 : slabdata      0      0      0
```

slab -- 从高速缓存申请内存

```
 *
 * Allocate an object from this cache.  The flags are only relevant
 * if the cache has no available objects.
 */
void *kmem_cache_alloc(struct kmem_cache *cachep, gfp_t flags)
void kmem_cache_destroy(struct kmem_cache *s)

static __always_inline void *
slab_alloc(struct kmem_cache *cachep, gfp_t flags, unsigned long caller)
```

kmalloc(), vmalloc()

static __always_inline void ***kmalloc**(size_t size, gfp_t flags)
static inline void ***kzalloc**(size_t size, gfp_t flags) // 内存空间置为0
返回一个指向内存块的指针，其内存块大小至少size大小，所分配的内存在物理上是连续的。

void ***vmalloc**(unsigned long size)
void ***vzalloc**(unsigned long size) // 内存空间置为0
返回一个指向内存块的指针，其内存块大小至少size大小，所分配的内存在物理上无需连续。

linux内存分配函数比较

内核空间		vmalloc/vfree	虚拟连续物理不定	vmalloc区大小限制	页 VMALLOC区域	可能睡眠，不能从中断上下文中调用，或其他不允许阻塞情况下调用。 VMALLOC区域vmalloc_start~vmalloc_end之间，vmalloc比kmalloc慢，适用于分配大内存。
	slab	kmalloc/kcalloc/krealloc/kfree	物理连续	64B-4MB(随slab而变)	2^order字节 Normal区域	大小有限，不如vmalloc/malloc大。 最大/小值由KMALLOC_MIN_SIZE/KMALLOC_SHIFT_MAX，对应64B/4MB。 从/proc/slabinfo中的kmalloc-xxxx中分配，建立在kmem_cache_create基础之上。
		kmem_cache_create	物理连续	64B-4MB	字节大小，需对齐 Normal区域	便于固定大小数据的频繁分配和释放，分配时从缓存池中获取地址，释放时也不一定真正释放内存。通过slab进行管理。
	伙伴系统	__get_free_page/__get_free_pages	物理连续	4MB(1024页)	页 Normal区域	__get_free_pages基于alloc_pages，但是限定不能使用HIGHMEM。
		alloc_page/alloc_pages/free_pages	物理连续	4MB	页 Normal/Vmalloc都可	CONFIG_FORCE_MAX_ZONEORDER定义了最大页面数2^11，一次能分配到的最大页面数是1024。