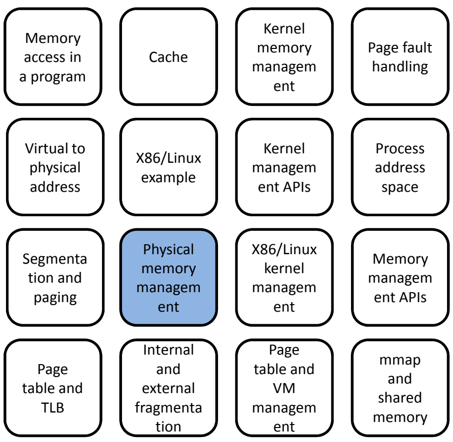
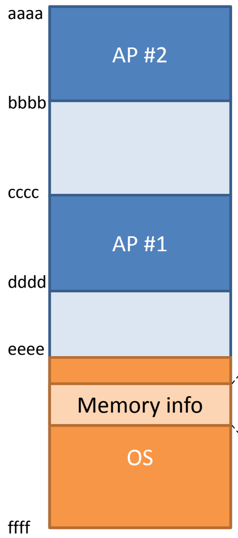
Operating System Design and Implementation

Memory Management – Part II

Shiao-Li Tsao



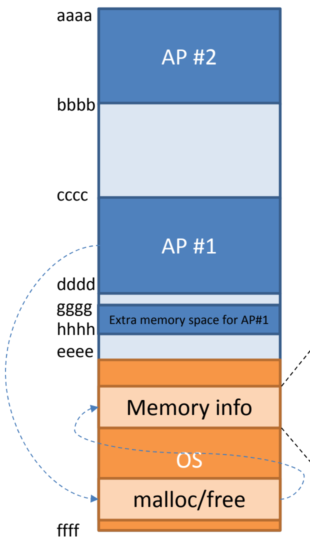
Physical memory management



Assume physical memory

Start	End	Access right
ffff	eeee	OS
dddd	cccc	AP #1
bbbb	aaaa	AP #2

Physical memory management allocation



Assume physical memory

Start	End	Access right
ffff	eeee	OS
dddd	cccc	AP #1
bbbb	aaaa	AP #2
hhhh	gggg	AP#1

Why Physical Memory Management?

- Always required
  - OS has to know how physically memory is used
- Kernel (physical contiguous and logical contiguous addresses are preferred)
  - For better performance
- Without MMU
  - Low cost
  - Better performance
  - Deterministic performance

How to Protect Memory Access without MMU?

- MPU (Memory Protection Units)
  - Low cost solution for multitasking and memory access protection

ARM core	Number of regions	Separate instruction and data regions	Separate configuration of instruction and data regions
ARM740T	8	no	no
ARM940T	16	yes	yes
ARM946E-S	8	no	yes
ARM1026EJ-S	8	no	yes

Source: ARM System Developer's Guide: Designing and Optimizing System Software by Andrew Sloss, Dominic Symes, Chris Wright

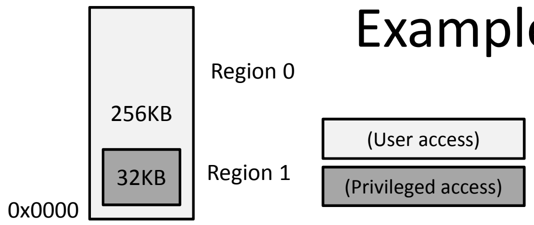
Region Attributes

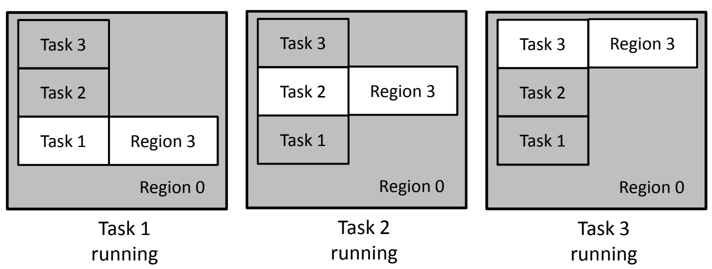
Region attributes	Configuration options
Type	instruction/data
Start address	multiple of size
Size	4 KB to 4 GB
Access permissions	read/write/execute
Cache	copy-back/write-through
Write buffer	enabled/disabled

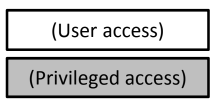
Source: ARM System Developer's Guide: Designing and Optimizing System Software by Andrew Sloss, Dominic Symes, Chris Wright

7

Examples

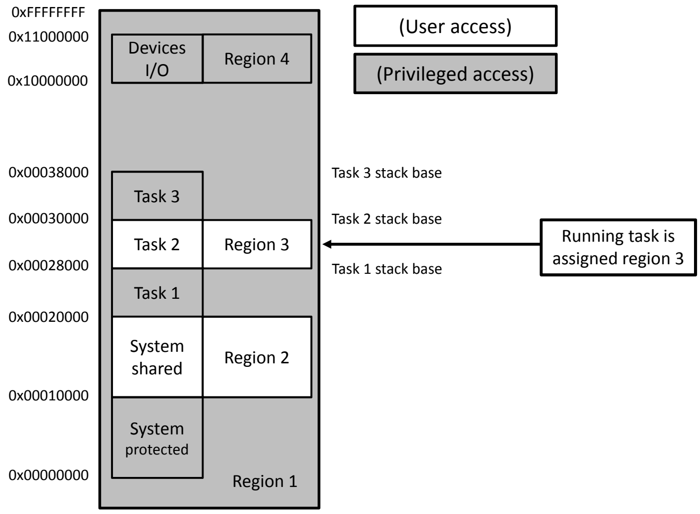






Source: ARM System Developer's Guide: Designing and Optimizing System Software by Andrew Sloss, Dominic Symes,

Region Assignment and Memory Map

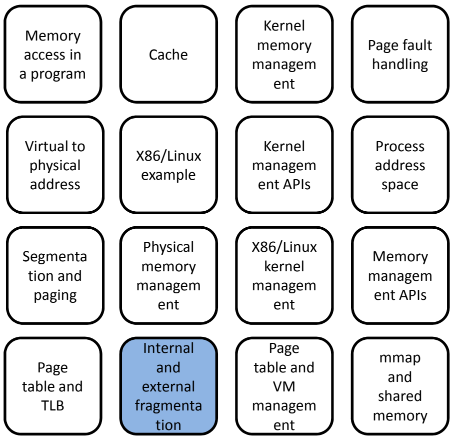


Source: ARM System Developer's Guide: Designing and Optimizing System Software by Andrew Sloss, Dominic Symes, Chris Wright

9

What are the issues for physical memory management?

- External fragmentation
- Internal fragmentation
- Search for empty space



External Fragmentation

Assume physical memory

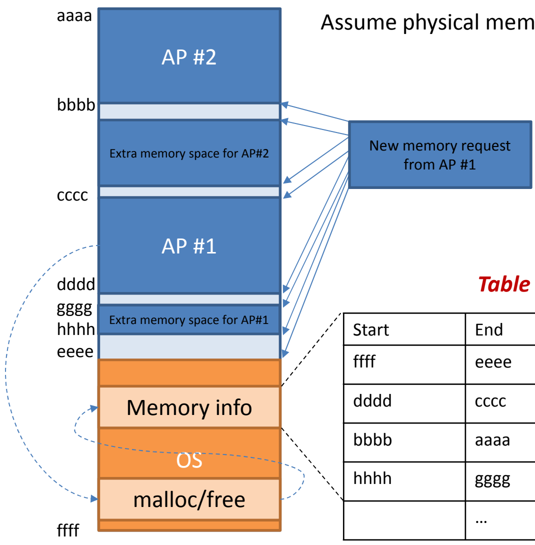
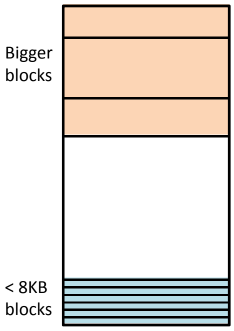


Table size?

Start	End	Access right
ffff	eeee	OS
dddd	cccc	AP #1
bbbb	aaaa	AP #2
hhhh	gggg	AP#1
	…	

uCLinux Design

- Standard Linux allocator: allocates blocks of 2^n size.
  - If 65 KB are requested, 128 KB will be reserved, and the remaining 63 KB won’t be reusable in Linux.
- uClinux 2.4 memory allocator: *kmalloc2* (aka *page_alloc2*)
  - Allocates blocks of 2n size until 4KB
  - Uses 4KB pages for greater requests
  - Stores amounts not greater than 8KB on the start of the memory, and larger ones at the end. Reduces fragmentation
  - Not available yet for Linux 2.6.

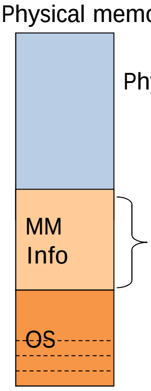


Source: http://free-electrons.com/doc/uclinux_introduction.pdf

Frame Table

- Divide physical memory into frames
  - frame size = page size (why?)
  - Different frame size (why?)

Physical memory

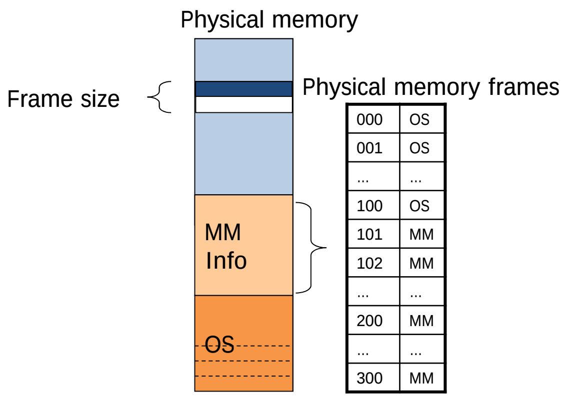


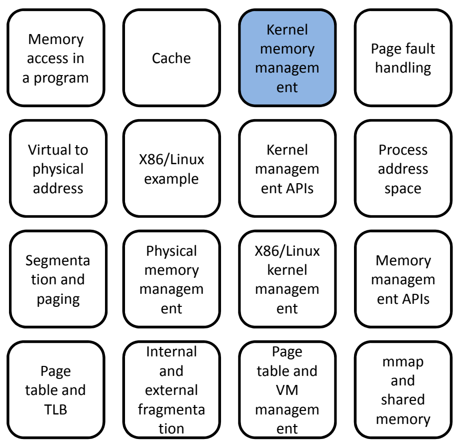
Physical memory frame

000	OS
001	OS
...	...
100	OS
101	MM
102	MM
...	...
200	MM
...	...
300	MM

Internal Fragmentation

- Different frame size (why?)





Kernel Memory Management

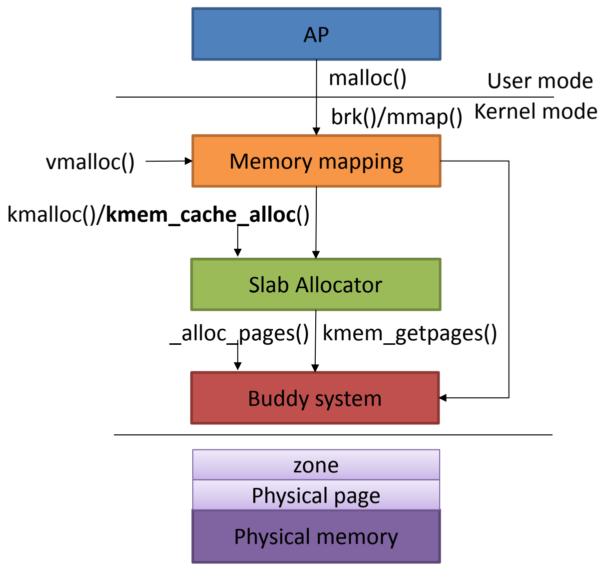
- Requirements
  - Managing memory have to spend memory
    - Balance between management overhead and waste
  - Prefer physical contiguous and logical contiguous addresses
    - For better performance
    - Different from application memory management (logical contiguous but not necessary physical contiguous)
      - Memory utilization is more important
  - Avoid external fragmentation
    - Separate large and small allocations
  - Avoid internal fragmentation
    - Multiplexing more memory block requests into one page

Kernel Memory Management (Cont.)

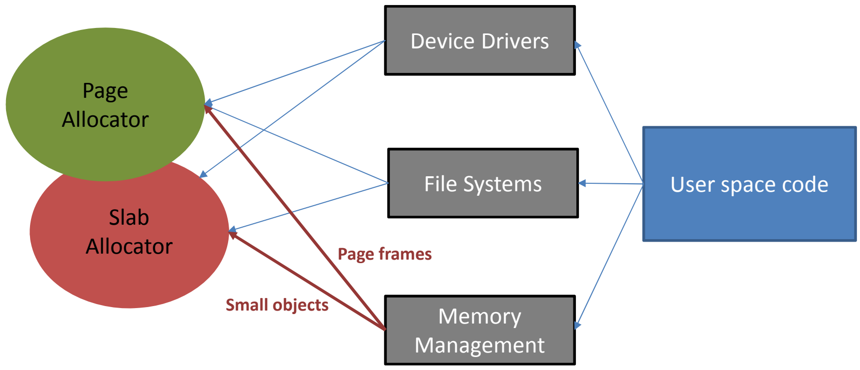
- Requirements
  - Support both large/small blocks allocations and free
    - Large memory block such as DMA
    - Small memory block such as task structure
  - Lower empty space search complexities
    - O(1)
  - Provide APIs for kernel memory allocation/free
    - For drivers, OS codes
  - Provides APIs for realizing application memory allocation/free
    - Allocate logical contiguous but not necessary physical contiguous memory

Memory access in a program	Cache	Internal and external fragmentation	Page fault handling
Virtual to physical address	X86/Linux example	Kernel management APIs	Process address space
Segmentation and paging	Physical memory management	X86/Linux kernel management	Memory management APIs
Page table and TLB	Kernel memory management	Page table and VM management	mmap and shared memory

Overall Architecture

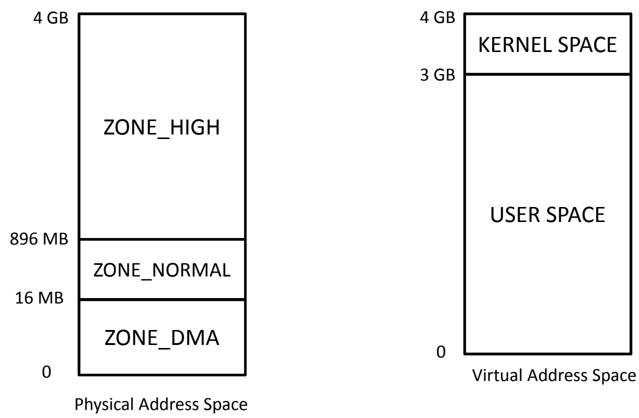


System Components around Slab Allocaters

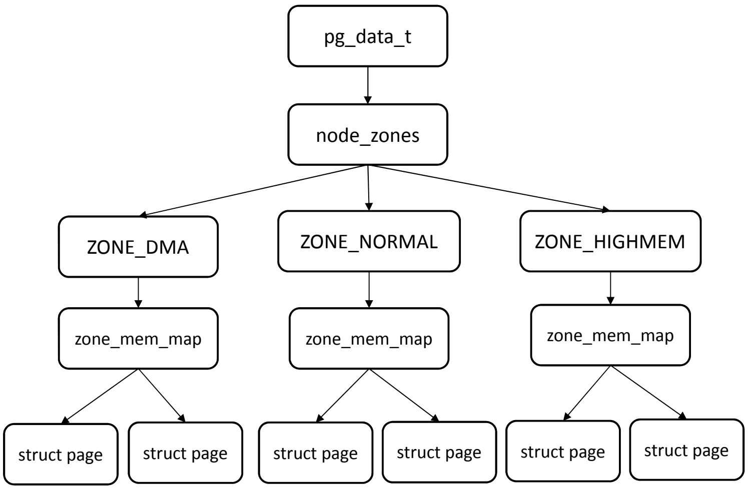


Source: http://events.linuxfoundation.org/sites/events/files/slides/slaballocators.pdf

Linux Memory Zones



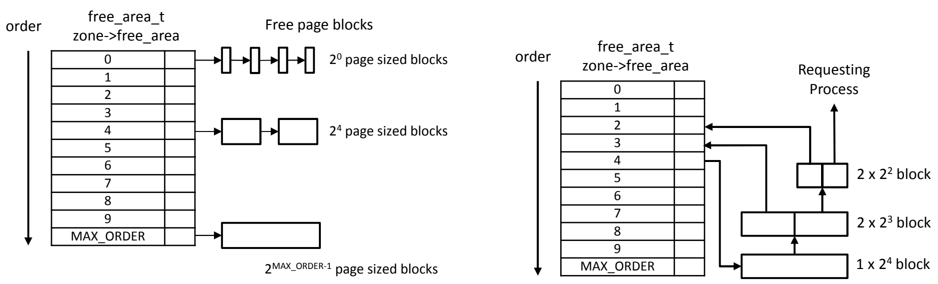
Relationship Between Nodes, Zones and Pages



Source: Understanding The Linux Virtual Memory Manager, Mel Gorman

Linux MM

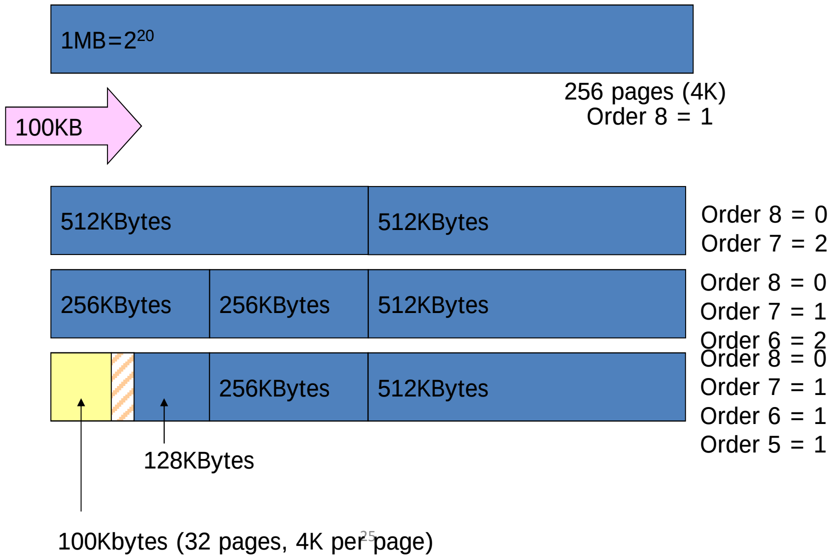
- Buddy allocation
  - Why ?
  - alloc_page



24

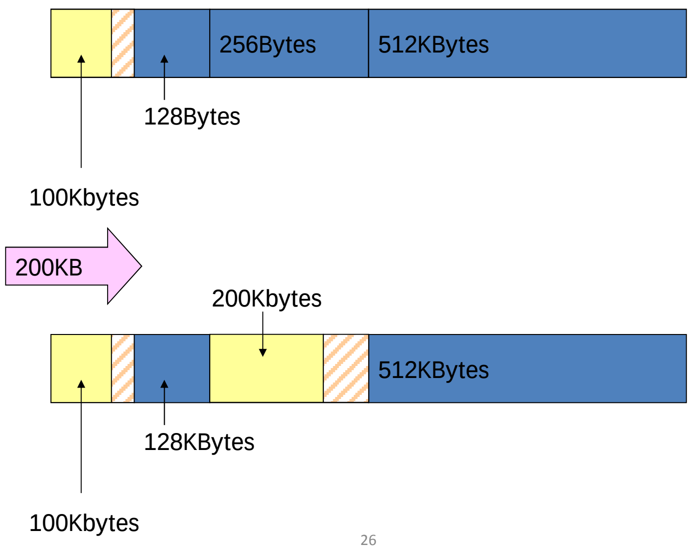
Source: Understanding The Linux Virtual Memory Manager, Mel Gorman

Zoned Buddy Allocator



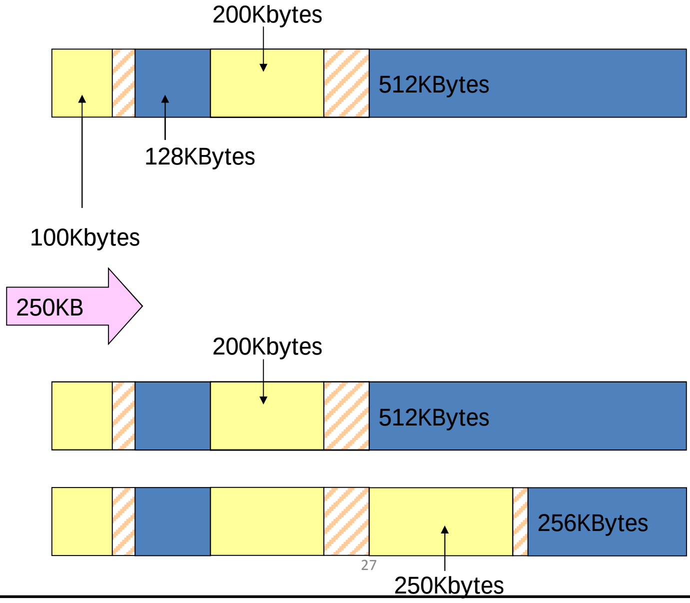
25

Zoned Buddy Allocator (Cont.)



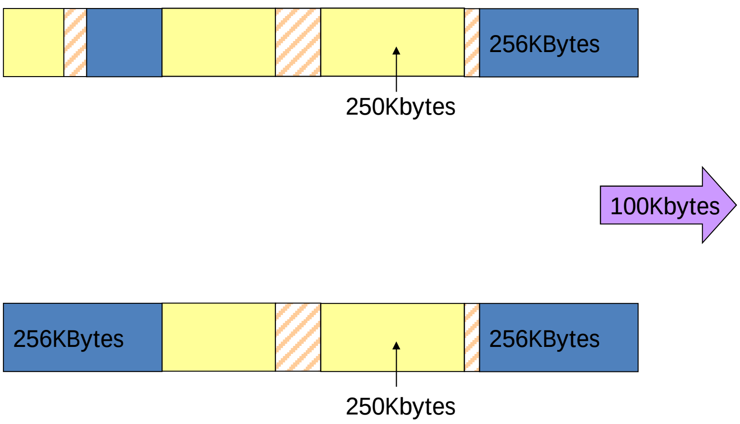
26

Zoned Buddy Allocator (Cont.)



27

Zoned Buddy Allocator (Cont.)



28

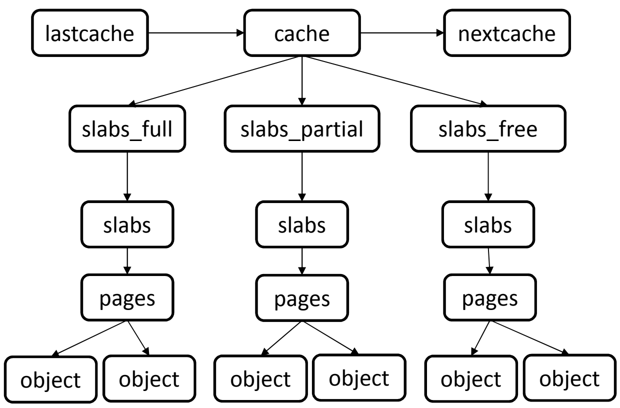
Page Allocation/Free APIs

- **gallopades** (unsigned int gfp_mask)
  - Allocate a single page and return a struct address
- **alloc_pages** (unsigned int gfp_mask, unsigned int order)
  - Allocate 2^{order} number of pages and return a struct page
- **get_free_page** (unsigned int gfp_mask)
  - Allocate a single page, zero it and return a virtual address
- **__get_free_page** (unsigned int gfp_mask)
  - Allocate a single page and return a virtual address
- **__get_free_pages** (unsigned int gfp_mask, unsigned int order)
  - Allocate 2^{order} number of pages and return a virtual address
- **__get_dma_pages** (unsigned int gfp_mask, unsigned int order)
  - Allocate 2^{order} number of pages from the DMA zone and return a struct page
- **__free_pages** (sturct page *page, unsigned int order)
  - Free an order number of pages from the given page
- **__free_page** (sturct page *page)
  - Free a single page
- **free_page** (void *addr)
  - Free a page from the given virtual address

Source: Understanding The Linux Virtual Memory Manager, Mel Gorman

Linux MM

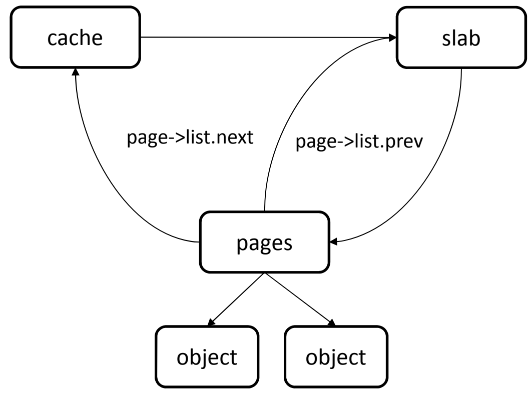
- Slab allocation
  - Why ?



30

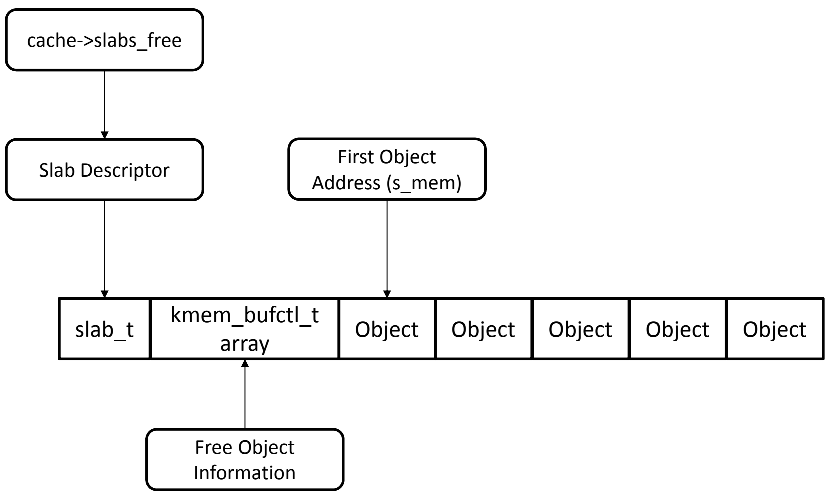
Source: Understanding The Linux Virtual Memory Manager, Mel Gorman

Page to Cache and Slab Relationship



Source: Understanding The Linux Virtual Memory Manager, Mel Gorman

Slab With Descriptor On-Slab



Source: Understanding The Linux Virtual Memory Manager, Mel Gorman

Slab Allocator API for caches

- **kmem_cache_create** (const char *name, size_t size, size_t offset, unsigned long flags, void (*ctor) (void*, kmem_cache_t *, unsigned long), void (*dtor)(void*, kmem_cache_t *, unsigned long))
  – Create a new cache and adds it to the cache chain.
- **kmem_cache_reap** (int gfp_mask)
  – Scans at mose REAP_SCANLEN caches and selects one for reaping all per-cpu objects and free slabs from. Called when memory is tight.
- **kmem_cache_shrink** (kmem_cache_t *cachep)
  – This function will delete all per-cpu objects associated with a cache and delete all slabs in the `slabs_free` list. It returns the number of pages freed.
- **kmem_cache_alloc** (kmem_cache_t *cachep, int flags)
  – Allocate a single object from the cache and return it to the caller.
- **kmem_cache_free** (kmem_cache_t *cachep, void *objp)
  – Free an object and return it to the cache.
- **kmalloc** (size_t size, int flags)
  – Allocate a block of memory from one of th sizes cache.
- **kfree** (const void *objp)
  – Free an block of memory allocated with `kmalloc`.
- **kmem_cache_destroy** (kmem_cache_t *cachep)
  – Destroys all objects in all slabs and frees up all associated memory before removing the cache from the chain.

Source: Understanding The Linux Virtual Memory Manager, Mel Gorman

Linux MM

- kmalloc/vmalloc
  – kmalloc()
    - similar to that of user-space's familiar malloc() routine
    - byte-sized chunks
    - memory allocated is physically contiguous
    - Through slab allocator
  – vmalloc()
    - virtually contiguous and not necessarily physically contiguous
    - user-space allocation function works
    - allocating potentially noncontiguous chunks of physical memory and "fixing up" the page tables to map the memory into a contiguous chunk of the logical address space

34