CS 378 (Spring 2003)

Linux Kernel Programming

Yongguang Zhang

(ygz@cs.utexas.edu)

Copyright 2003, Yongguang Zhang

Last Lecture

- Linux Memory Modeul
  - Demand Paged Virtual Memory
- Address Translation
  - 3-Level Page Table: pgd_t, pmd_t, pte_t
  - i386 Architecture: two-level
    - Page size: 4KB (12-bit)
    - Total addressable space: 4GB (32-bit)
    - User space: 3GB (0x00000000-0xbfffffff)
    - Kernel space: 1GB (0xc0000000-0xffffffff)

© 2003 Yongguang Zhang

This Lecture

- Managing Physical Memory
- Managing Pages
- Managing Kernel Dynamic Memory
- Managing Process Address Space
- Paging/Swapping

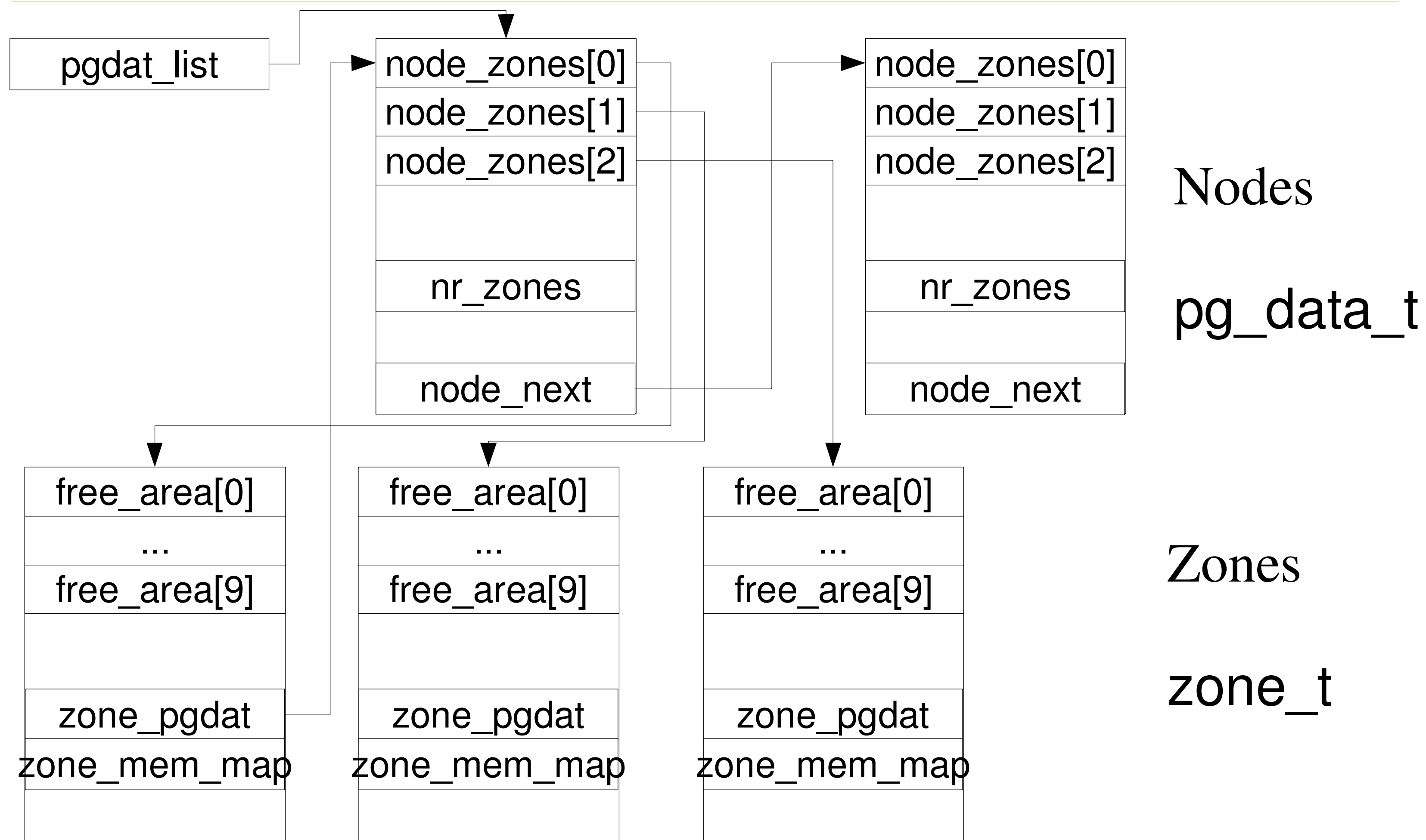
© 2003 Yongguang Zhang

Managing Physical Memory

- NUMA (Non-Uniform Memory Access)
  - Memory in a machine is divided into “nodes”
  - Each node has the same access time
  - Data structure: pg_data_t (include/linux/mmzone.h)
- Memory Zones
  - Each node has several “zones”
  - Each zone is different type of memory
  - Data structure: zone_t (include/linux/mmzone.h)
- Pages (Frames)
  - Data structure: struct page (include/linux/mm.h)

© 2003 Yongguang Zhang

Nodes and Zones



Nodes

pg_data_t

Zones

zone_t

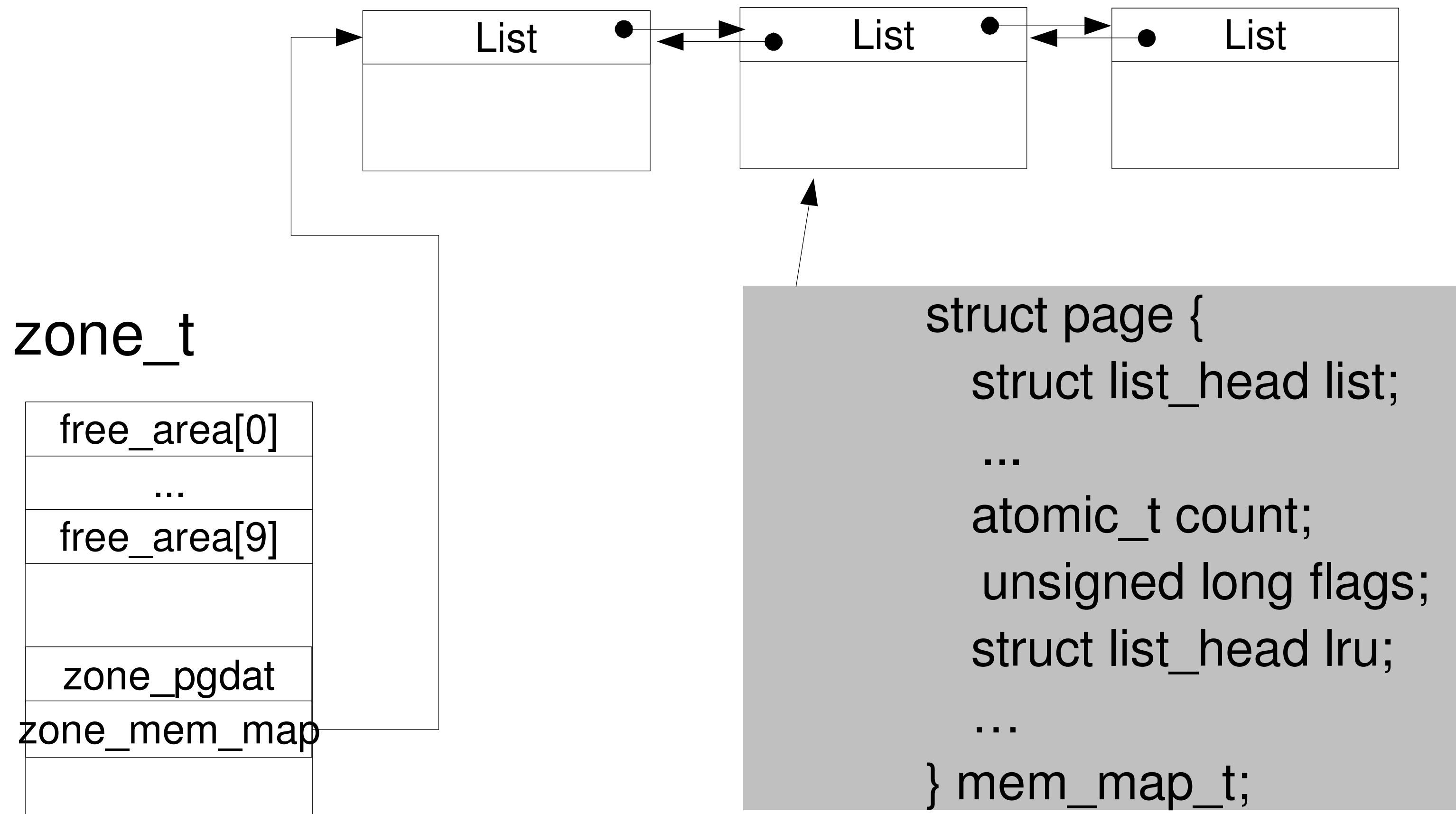
 © 2003 Yongguang Zhang

3 Zones

- ZONE_DMA
  - 0-16M (i386)
  - DMA capability: some device driver need to use this memory for I/O
- ZONE_NORMAL
  - 16M-896M (i386)
  - Normal memory direct mapped by kernel
- ZONE_HIGHMEM
  - >896M (i396)
  - Not used in 64-bit architecture

© 2003 Yongguang Zhang

Physical Pages



© 2003 Yongguang Zhang

Page Allocation

- Contiguous and non-contiguous allocation
- The Buddy System Algorithm
  - Pages are allocated in blocks which are powers of 2 in size (1, 2, 4, 8, 16, 32, 64, 128, 256, 512 pages)
  - Each size with its own free list (called free area)
  - De-allocation: if the adjacent buddy block is also free, combine to form a new free block for the next size block of pages
- Data structure: `free_area_t`

© 2003 Yongguang Zhang

Page Allocation Functions

- Allocate a block of 2^{order} contiguous pages:
  - struct page * alloc_pages(unsigned int gfp_mask, unsigned int order)
- Allocate a single page:
  - struct page * alloc_page(unsigned int gfp_mask)
- Other functions: See include/linux/mm.h
- GFP flags: where to allocate the page(s)
  - GFP_KERNEL, GFP_USER, GFP_DMA, ...

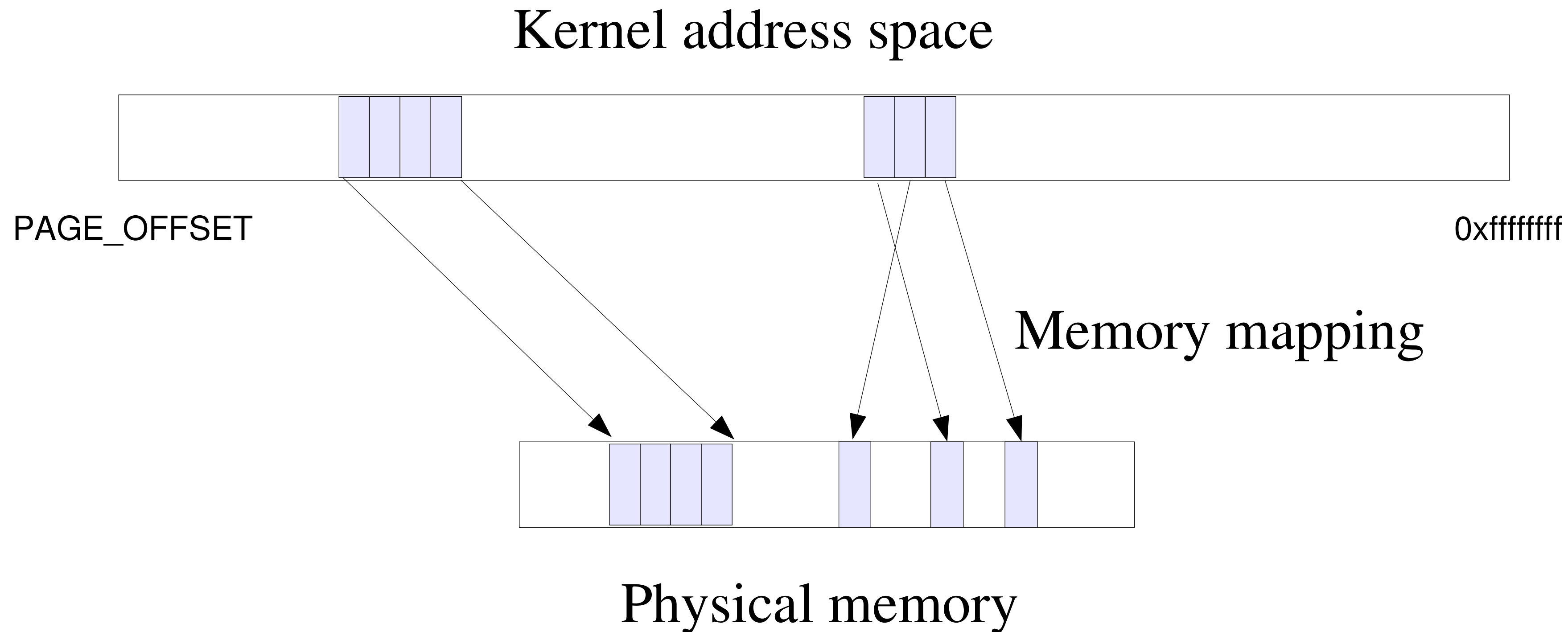
© 2003 Yongguang Zhang

Noncontiguous Memory

- Noncontiguous page frames in contiguous linear address
  - Not all virtual memory maps to the contiguous page frames
  - Make sense for infrequent use
- To allocate (in include/linux/vmalloc.h)
  - void *vmalloc(unsigned long size)
- To release
  - void vfree(const void * addr)

© 2003 Yongguang Zhang

Contiguous vs Non-Contiguous



 © 2003 Yongguang Zhang

Dynamic Kernel Memory

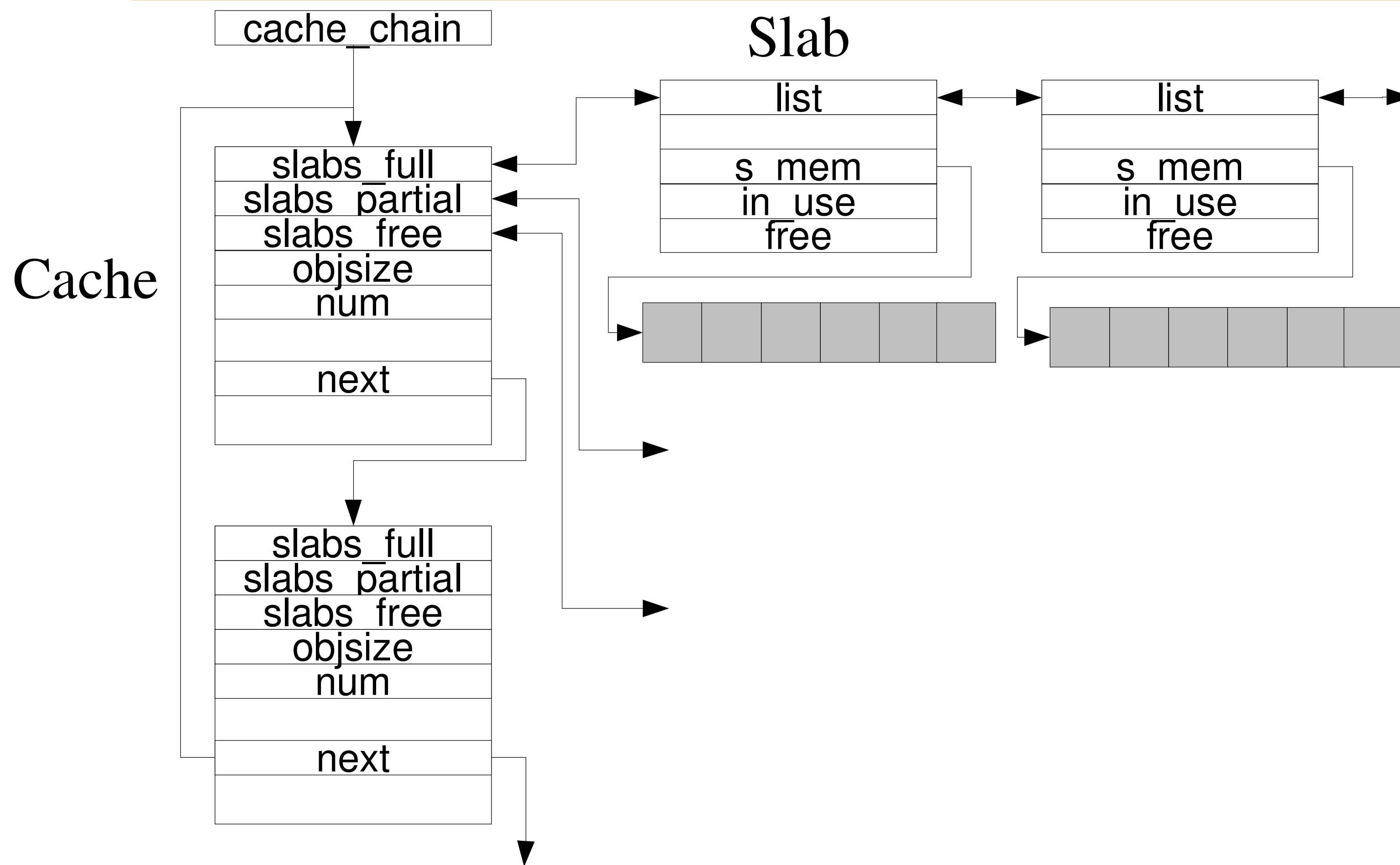
- For small memory use: 32, 64, ..., 131072 bytes
- For kernel data objects that need to be frequently allocated and released
  - Examples: file descriptors, sockets, inodes, …
- Slab Allocator
  - Group memory area (called cache) by kernel data object type
  - For each type, maintain memory cache (free list) for objects that are previously allocated then released
  - Interface with page frame allocator
  - Source code: `mm/slab.c`

Caches

- One cache for each type of object
  - Look at /proc/slabinfo
  - size-N and size-N (DMA): general purpose caches
- Data structures
  - Cache: struct kmem_cache_s
  - Slab: struct slab_s
- Functions:
  - To create a cache: kmem_cache_create(...)
  - To allocate an object: kmem_cache_alloc(...)
  - To free (return to cache): kmem_cache_free(...)

© 2003 Yongguang Zhang

Caches and Slabs



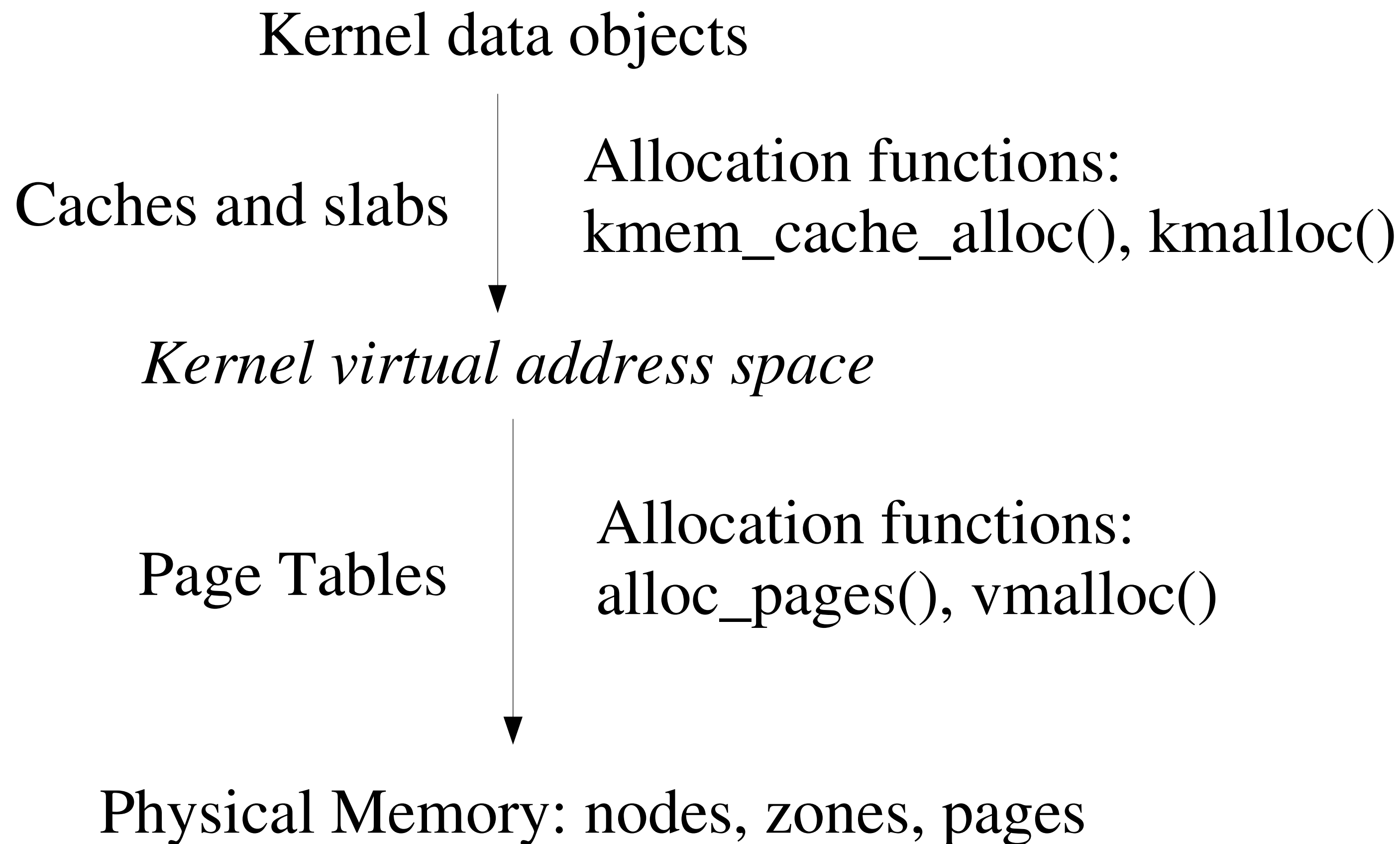
© 2003 Yongguang Zhang

General Purpose Cache

- For general purpose object
- 13 general purpose cache in slab allocator
  - Power of 2: 32, 64, …, 131072
- To allocate memory in this category:
  - include <linux/slab.h>
  - void *kmalloc(size_t size, init flags)
  - Flags: SLAB_ *
- To free memory
  - void kfree(const void *objp)

© 2003 Yongguang Zhang

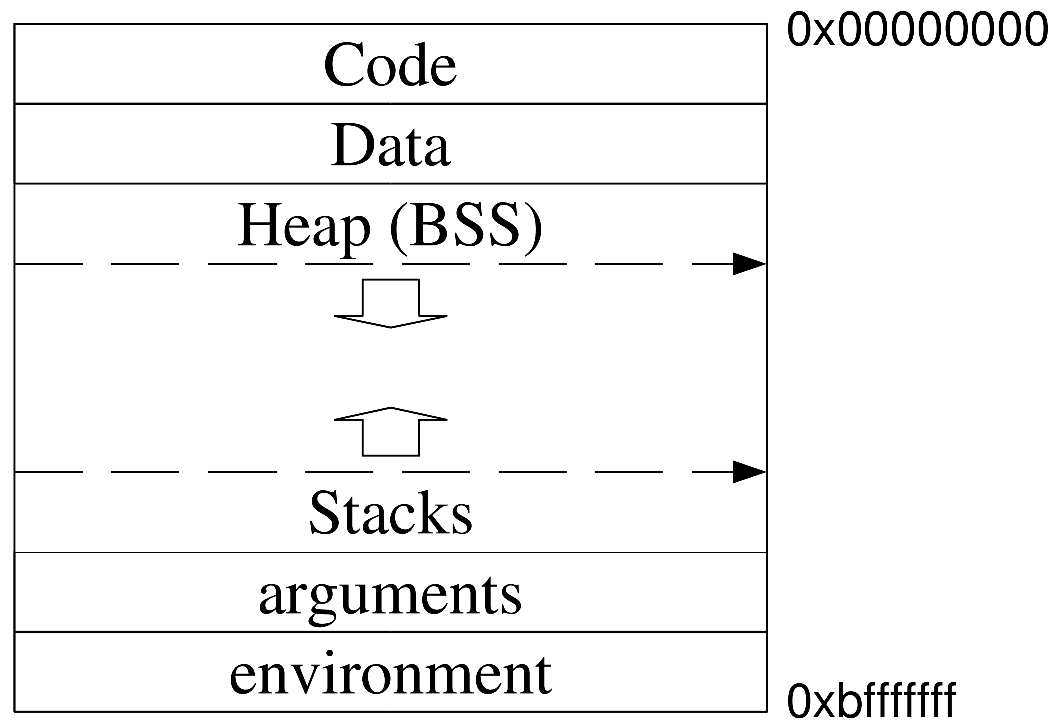
Summary: Kernel Memory



© 2003 Yongguang Zhang

Process Address Space

- Abstract model of user-space memory use:



© 2003 Yongguang Zhang

Address Space in ELF Format

- Program segments
  - Code: start_code to end_code
  - Data: start_data to end_data
  - BSS (Heap): start_brk to brk (growable)
  - Can have more than one segment (program, shared libraries, etc.)
- Run-Time segment:
  - Stack: start_stack (growable)
  - Arguments: arg_start to arg_end
  - Environment: env_start to env_end (0xbfffffff)

© 2003 Yongguang Zhang

Process Memory Descriptor

- One per process (in include/linux/sched.h)

```
struct mm_struct {
    struct vm_area_struct * mmap;
    …
    pgd_t * pgd;
    …
    unsigned long start_code, end_code, start_data, end_data;
    unsigned long start_brk, brk, start_stack;
    unsigned long arg_start, arg_end, env_start, env_end;
    …
};
```


© 2003 Yongguang Zhang

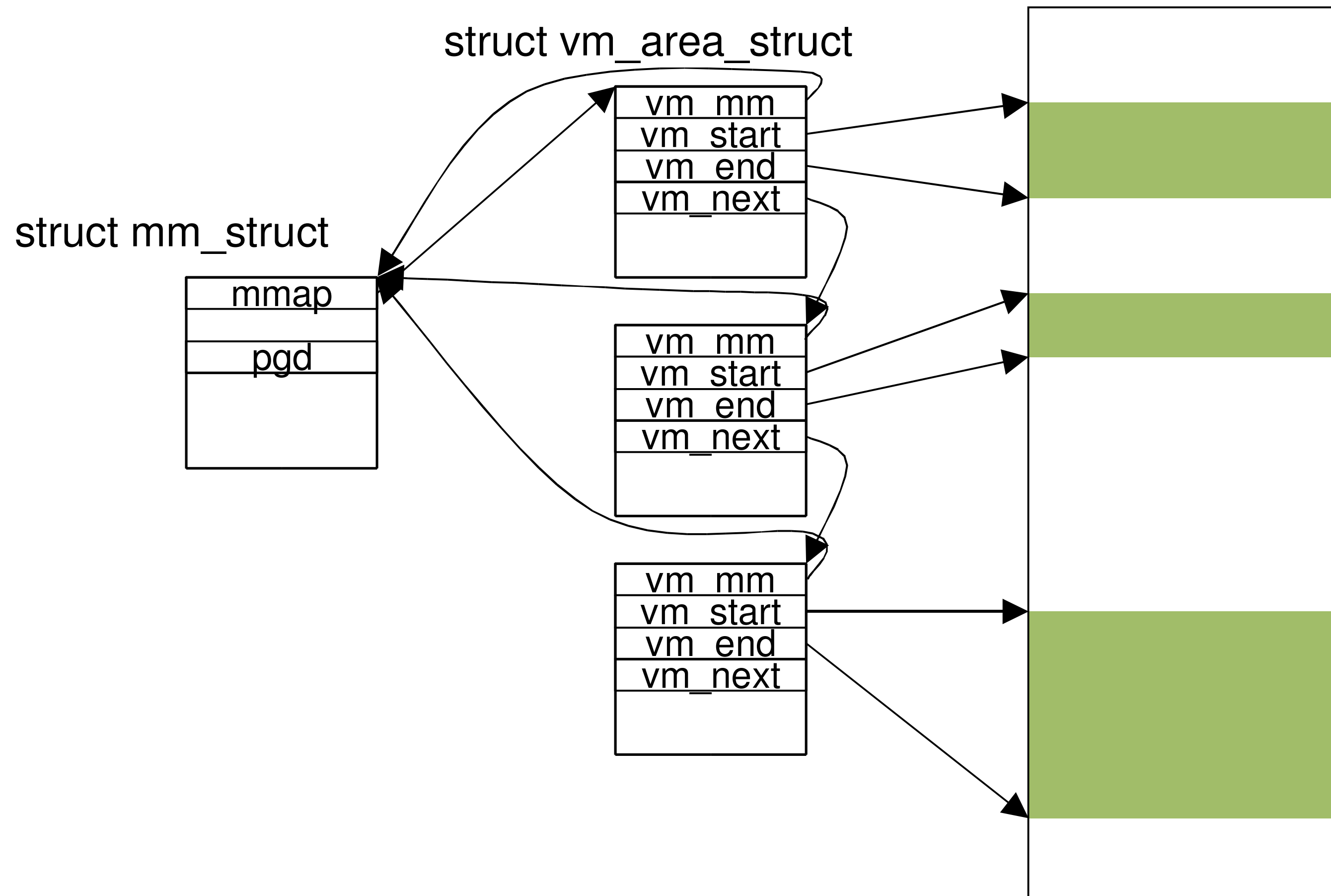
VM Area Descriptor

- Linear Address Interval (in include/linux/mm.h)

```
struct vm_area_struct {
        struct mm_struct * vm_mm;
        unsigned long vm_start;
        unsigned long vm_end;
        struct vm_area_struct *vm_next;
        ….
};
```


© 2003 Yongguang Zhang

VM Area



© 2003 Yongguang Zhang

VM Area Handlers

- To Create VM Area and Do the Mapping
  - do_mmap() (in include/linux/mm.h)
  - Used in system calls execve() , brk()
- To Release a VM Area and Shrink the Address Space
  - do_munmap() (in mm/mmap.c)
  - Used in system calls brk()
- To Find a VM Area that includes a given address
  - find_vma() (in mm/mmap.c)

© 2003 Yongguang Zhang

Swapping and Page Cache

- For Pages in a Process's User Space
  - Swap: Secondary Memory on the disk
  - Page Cache: Main Memory
- Data Structure: 3 sets of lists
  - Active pages, usually mapped by a process's PTE
    - active_list (in mm/page_alloc.c)
  - Inactive, unmapped, clean or dirty
    - inactive_dirty_list (in mm/page_alloc.c)
  - Clean pages, unmapped (one list per zone)
    - zone_t.inactive_clean_list (in include/linux/mmzone.h)

© 2003 Yongguang Zhang

Kernel Swap Daemon

- Implemented as a kernel thread
  - kswapd() (in mm/vmscan.c)
  - Wake up periodically
    - Wake up more frequently if memory shortage
- Check memory and if memory is tight
  - Age pages that have not be used
  - Move pages to inactive lists
  - Write dirty pages to disk
  - Swap pages out if necessary

© 2003 Yongguang Zhang

More kswapd()

- Call swap_out() to scan inactive page lists
  - Removes page reference from process's page table
  - Actual swapping is done independently by file I/O
- Call refill_inactive_scan() to
  - Scan the active_list to find unused page
  - Call age_page_down() to reduce page->age count
  - If page->age is zero, move to inactive_dirty_list
- Call page_launder() to clean dirty pages:
  - Scan inactive_dirty_list for dirty pages, write to disk
  - Move clean pages to the zone's inactive_clean_list

© 2003 Yongguang Zhang

Life Cycle of a Page in Cache

- A page is read into memory from disk
  - Caused by page-fault, or read-ahead
  - Added to page cache: active_list
- Page is "dirty" if written by the process
- If not used, page->age count gradually reduced
  - If page->age is zero, moved to inactive_dirty_list
  - May be moved to an inactive_clean_list later
- Page can be recovered from an inactive list
- Inactive page can be released
  - reclaim_page() to free page for used by others

© 2003 Yongguang Zhang

Memory Management Summary

- Physical Page Management
  - Page Table Structure
  - Memory Zones and Page Allocation
- Kernel Memory Management
  - Slab and Allocation Routines
- Virtual Memory Management (Process Address Space)
  - Memory Layout and VM Area
  - Page Cache and Swapping
  - Page Fault

© 2003 Yongguang Zhang

Summary

- Textbook:
  - LKP §4
  - ULK §2, §7, §8

© 2003 Yongguang Zhang