Jin-Soo Kim
(*jinsoo.kim@snu.ac.kr*)

Systems Software &
Architecture Lab.

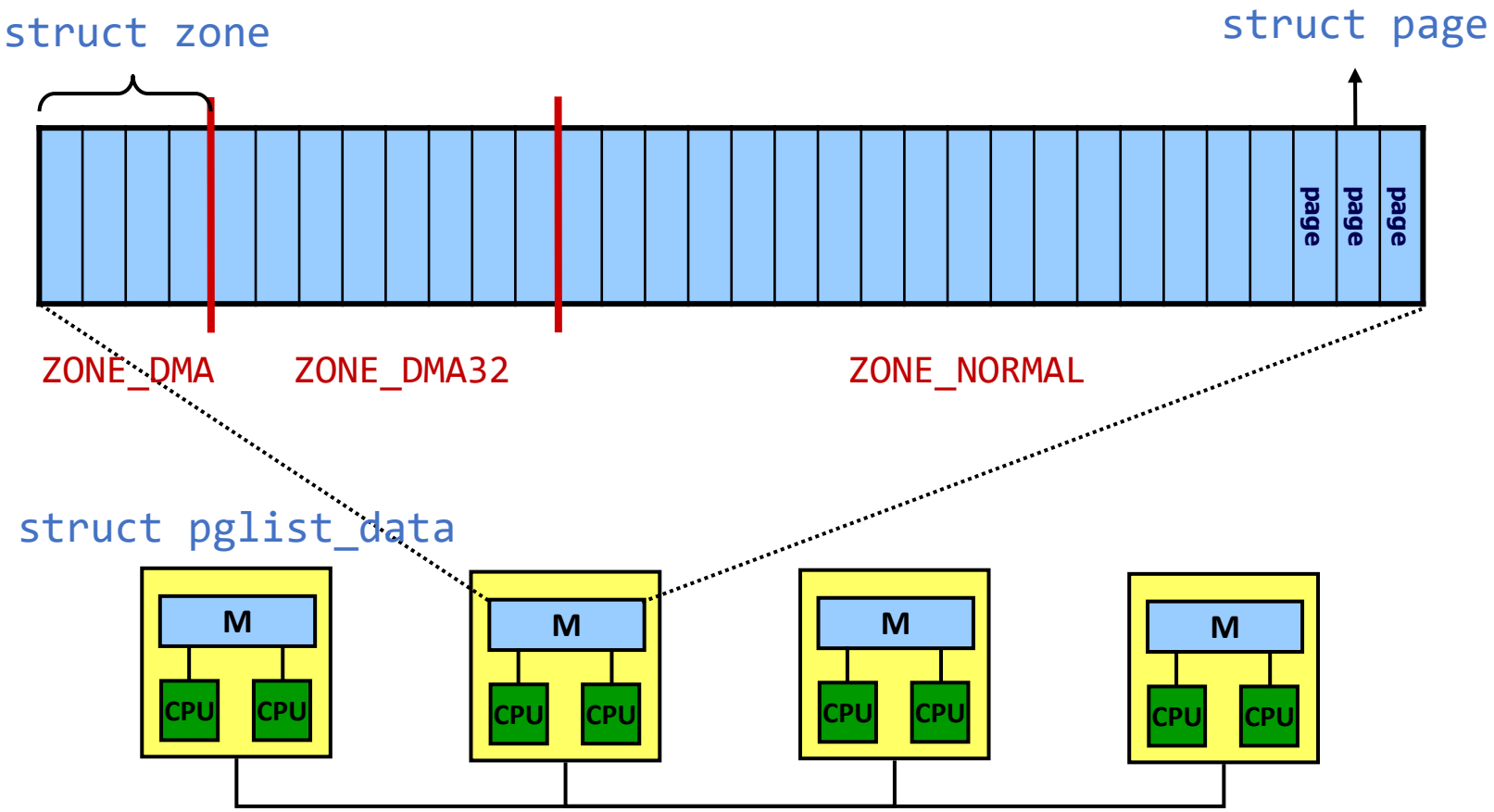
Seoul National University

Fall 2020

Linux
Memory Management

Physical Memory Management

The Big Picture



Pages

- `struct page` (@ `./include/linux/mm_types.h`)
  - One entry (min. 56 bytes) for every physical frame
  - All page descriptors are stored in the `mem_map` array

`unsigned long`	`flags;`	Page flags (defined in ./include/linux/page-flags.h>
`struct list_head`	`lru;`	Next & prev links for LRU list
`struct address_space *`	`mapping;`	How the page is used? (file-backed, anonymous, etc.)
`pgoff_t`	`index;`	Offset within the mapping
`unsigned long`	`private`	Private data
`atomic_t`	`_mapcount;`	Count of PTEs mapped
`atomic_t`	`_refcount;`	The number of references to this page

Zones

- `struct zone` (@ `./include/linux/mmzone.h`)
  - Some hardware devices are capable of performing DMA to only certain memory addresses
  - Some architectures are capable of physically addressing larger amounts of memory than they can virtually address

ZONE_DMA	DMA-able pages (for old ISA devices)	< 16MB
ZONE_DMA32	DMA-able pages (for devices with 32-bit DMA capability)	< 4GB
ZONE_NORMAL	Normally addressable pages (directly mapped to kernel virtual address space)	
ZONE_HIGHMEM	Dynamically addressable pages (should be mapped to kernel virtual address space prior to access)	> 896MB (for 32-bit CPU)

Per-Zone Watermarks

- Reserved page frames
  - Memory allocation requests can be satisfied immediately without memory reclaiming
  - Reduces the change of failure in case of atomic memory allocation requests (GFP_ATOMIC)
- For ZONE_DMA(32) and ZONE_NORMAL:
  - `min_free_kbytes` = `sqrt(directly_mapped_memory * 16)` (KB)
    - 128KB ~ 64MB (e.g., 4MB for 1GB, 8MB for 4GB, 16MB for 16GB)
  - `zone->pages_min` = this zone's contribution to `min_free_kbytes` / pagesize;
  - `zone->pages_low` = `zone->pages_min * 1.25`
  - `zone->pages_high` = `zone->pages_min * 1.5`

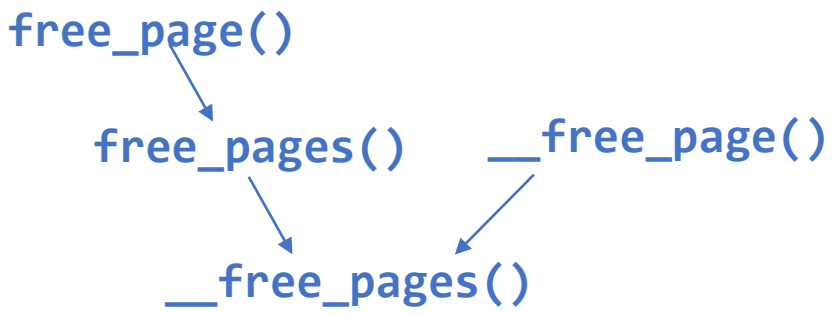
/proc/zoneinfo

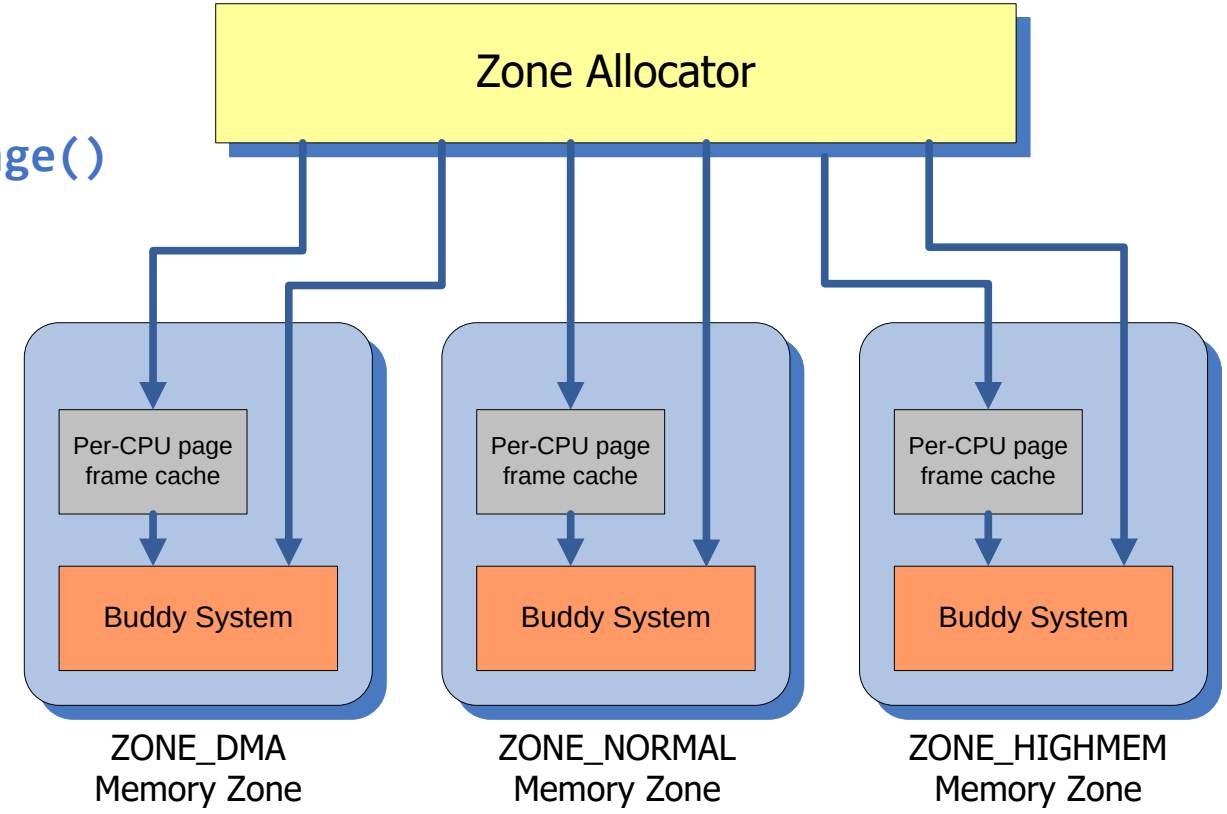
```
Node 0, zone      DMA
  pages free     1971
        min      11
        low      14
        high     17
        spanned  4095
        present  3998
        managed  3977
        ...
Node 0, zone    DMA32
  pages free     5777
        min      1409
        low      1910
        high     2411
        spanned  520176
        present  520176
        managed  505111
        ...
Node 0, zone   Normal
  pages free     0
        min      0
        low      0
        high     0
        spanned  0
        present  0
        managed  0
        ...
```

Zone Allocator

- Allocation

- Deallocation





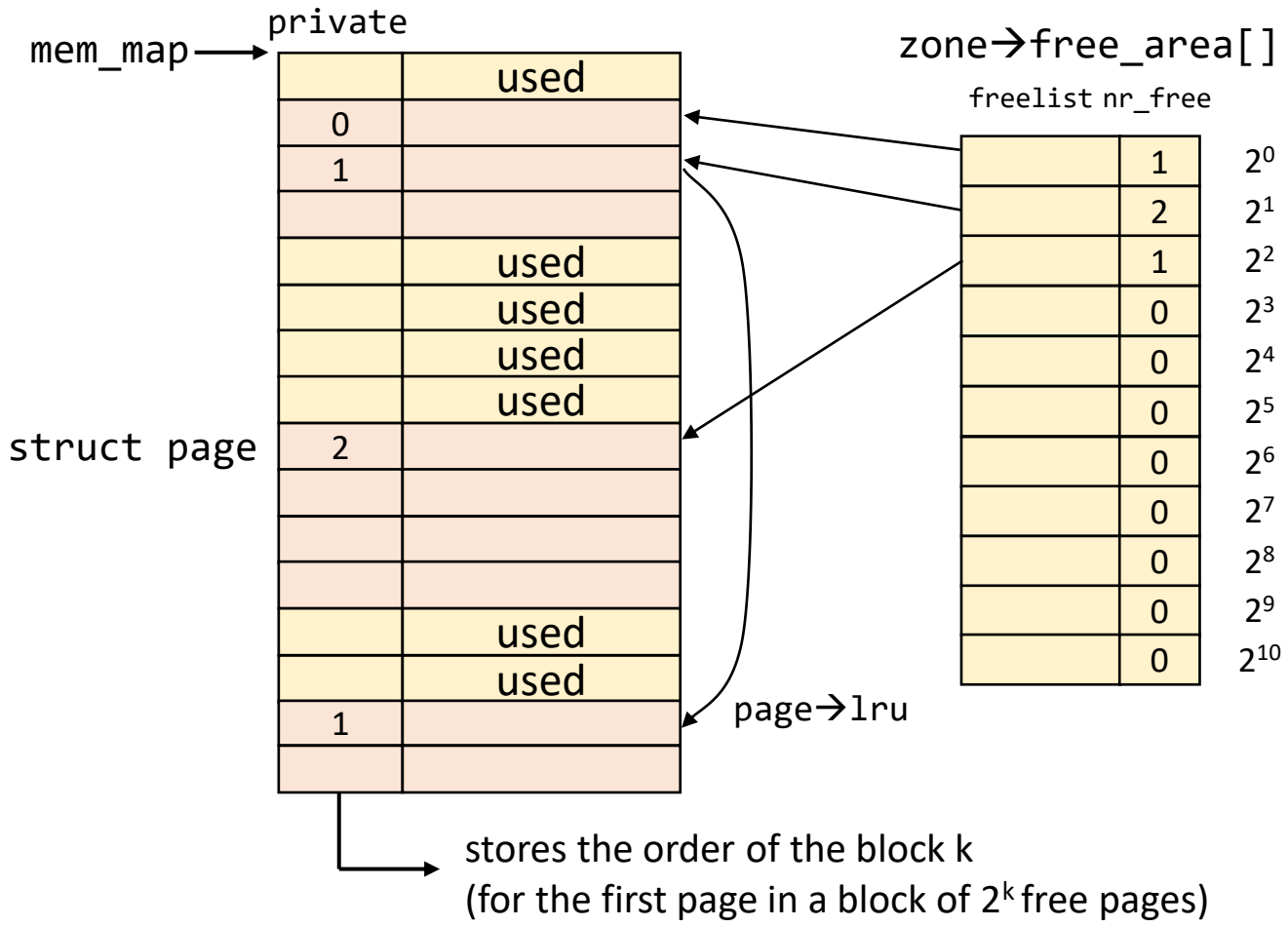
Buddy Algorithm

- Treat physical memory as a collection of 2^n-page-sized blocks aligned on 2^n-page boundaries
- To allocate a block of a given order,
  - If a block is found at the specified order, it is allocated immediately
  - If a block of a higher order must be used,
    - divide the larger block into two $2^{order-1}$ blocks,
    - add the lower half to the appropriate freelist, and
    - allocate the memory from the upper half, executing this step recursively
- When freeing a block,
  - If the block has a free buddy block, combine the two blocks into a single free block
  - This process is performed recursively if necessary

Buddy Allocator Implementation

- @ `mm/page_alloc.c`
- A different buddy system for each zone
- All free page frames are grouped into MAX_ORDER(11) lists of blocks
  - Groups of 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024 contiguous page frames
- `struct page *__rmqueue()`
  - Buddy allocation
- `void __free_one_page()`
  - Buddy deallocation
- `void page_is_buddy()`
  - Checks whether the block is free and it's a buddy

Buddy Allocator



Anti-Fragmentation

- Grouping pages by mobility
  - Buddy allocator suffers from physical memory fragmentation
- Page grouping using pageblock

Unmovable	Mainly used by kernel core (fixed position) A non-movable page cannot be located in the middle of a block of movable pages
Reclaimable	Cannot be moved directly However, after reclaim, it can be used for constructing a larger block
Movable	Always can be moved (e.g., user spaces), When moved, the pte is updated

- ZONE_MOVABLE
  - A portion of ZONE_HIGH is assigned to ZONE_MOVABLE (virtual zone)
  - Movable pages are allocated in ZONE_MOVABLE

Migrate Types

- A zone has a buddy (free_area) list per each migrate type

Types	Description	Fallback order when fails
MIGRATE_UNMOVABLE	Buddy's unmovable free pages	UNM → REC → MOV
MIGRATE_RECLAIMABLE	Buddy's reclaimable free pages	REC → UNM → MOV
MIGRATE_MOVABLE	Buddy's movable free pages	MOV → REC → UNM
MIGRATE_RESERVE	Initialized for urgent page allocation. Used when allocation from other migrate is impossible	
MIGRATE_ISOLATE	Space in which pages are isolated for page migration (compaction or NUMA balancing)	

Per-CPU Page Frame Cache

- The kernel often requests and releases single page frames
- Prepare some pre-allocated page frames to be used for single memory requests issued by the local CPU

- `struct page *__rmqueue_pcplist()`
  - Allocate a page from the per-cpu list
- `void free_unref_page()`
  - Free a 0-order page

Slab Allocator

- Developed by Sun Microsystems for Solaris 2.4 in 1994
- The kernel functions tend to request memory areas of the same type repeatedly
- The slab allocator does not discard the objects that have been allocated and then released but saves them in memory
- When a new object is then requested, it can be taken from memory without having to be reinitialized
- The slab allocator works on top of the buddy system for allocating small objects

Linux Slab Allocators

- SLOB: K&R allocator (1991-1999)
  - Compact, low memory footprint
- SLAB: Solaris-type allocator (1999-2008)
  - Cache-friendly
- SLUB: Unqueued allocator (2008-today)
  - Simple
  - Superior debugging
  - Defragmentation
  - Scalability on many cores

Slab Allocation

- For specific objects
  - `struct kmem_cache *kmem_cache_create()`
  - `void *kmem_cache_alloc()`
  - `void kmem_cache_free()`

- For general sizes
  - `void *kmalloc()`
  - `void kfree()`

/proc/slabinfo

```
slabinfo - version: 2.1
# name            <active_objs> <num_objs> <objsize> <objperslab> <pagesperslab> : tunables <limit> <batchcount> <sharedf
actor> : slabdata <active_slabs> <num_slabs> <sharedavail>
ext4_groupinfo_4k    336    336    144   28    1 : tunables    0    0    0 : slabdata     12     12      0
ip6-frags              0      0    184   22    1 : tunables    0    0    0 : slabdata      0      0      0
PINGv6                 0      0   1152   14    4 : tunables    0    0    0 : slabdata      0      0      0
RAWv6                 28     28   1152   14    4 : tunables    0    0    0 : slabdata      2      2      0
UDPv6                 12     12   1280   12    4 : tunables    0    0    0 : slabdata      1      1      0
tw_sock_TCPv6          0      0    248   16    1 : tunables    0    0    0 : slabdata      0      0      0
request_sock_TCPv6     0      0    304   13    1 : tunables    0    0    0 : slabdata      0      0      0
TCPv6                 13     13   2368   13    8 : tunables    0    0    0 : slabdata      1      1      0
nf_conntrack           0      0    320   12    1 : tunables    0    0    0 : slabdata      0      0      0
kcopyd_job             0      0   3312    9    8 : tunables    0    0    0 : slabdata      0      0      0
scsi_sense_cache     224    224    128   32    1 : tunables    0    0    0 : slabdata      7      7      0
i915_vma               0      0    576   14    2 : tunables    0    0    0 : slabdata      0      0      0
execute_cb             0      0    128   32    1 : tunables    0    0    0 : slabdata      0      0      0
i915_request           0      0    640   12    2 : tunables    0    0    0 : slabdata      0      0      0
mqueue_inode_cache     9      9    896    9    2 : tunables    0    0    0 : slabdata      1      1      0
nfs_direct_cache       0      0    200   20    1 : tunables    0    0    0 : slabdata      0      0      0
nfs_read_data         36     36    896    9    2 : tunables    0    0    0 : slabdata      4      4      0
nfs_inode_cache        0      0   1048   15    4 : tunables    0    0    0 : slabdata      0      0      0
isofs_inode_cache      0      0    624   13    2 : tunables    0    0    0 : slabdata      0      0      0
fat_inode_cache       11     11    712   11    2 : tunables    0    0    0 : slabdata      1      1      0
fat_cache              0      0     40  102    1 : tunables    0    0    0 : slabdata      0      0      0
jbd2_journal_handle   85     85     48   85    1 : tunables    0    0    0 : slabdata      1      1      0
jbd2_journal_head    952   1088    120   34    1 : tunables    0    0    0 : slabdata     32     32      0
jbd2_revoke_table_s  256    256     16  256    1 : tunables    0    0    0 : slabdata      1      1      0
ext4_inode_cache  136721 136965   1064   15    4 : tunables    0    0    0 : slabdata   9131   9131      0
ext4_allocation_context     32     32    128   32    1 : tunables    0    0    0 : slabdata      1      1      0
```

Noncontiguous Areas

- Noncontiguous memory areas
  - Virtually contiguous, but physically noncontiguous
  - Need to modify kernel page tables
  - Can make use of high memory page frames

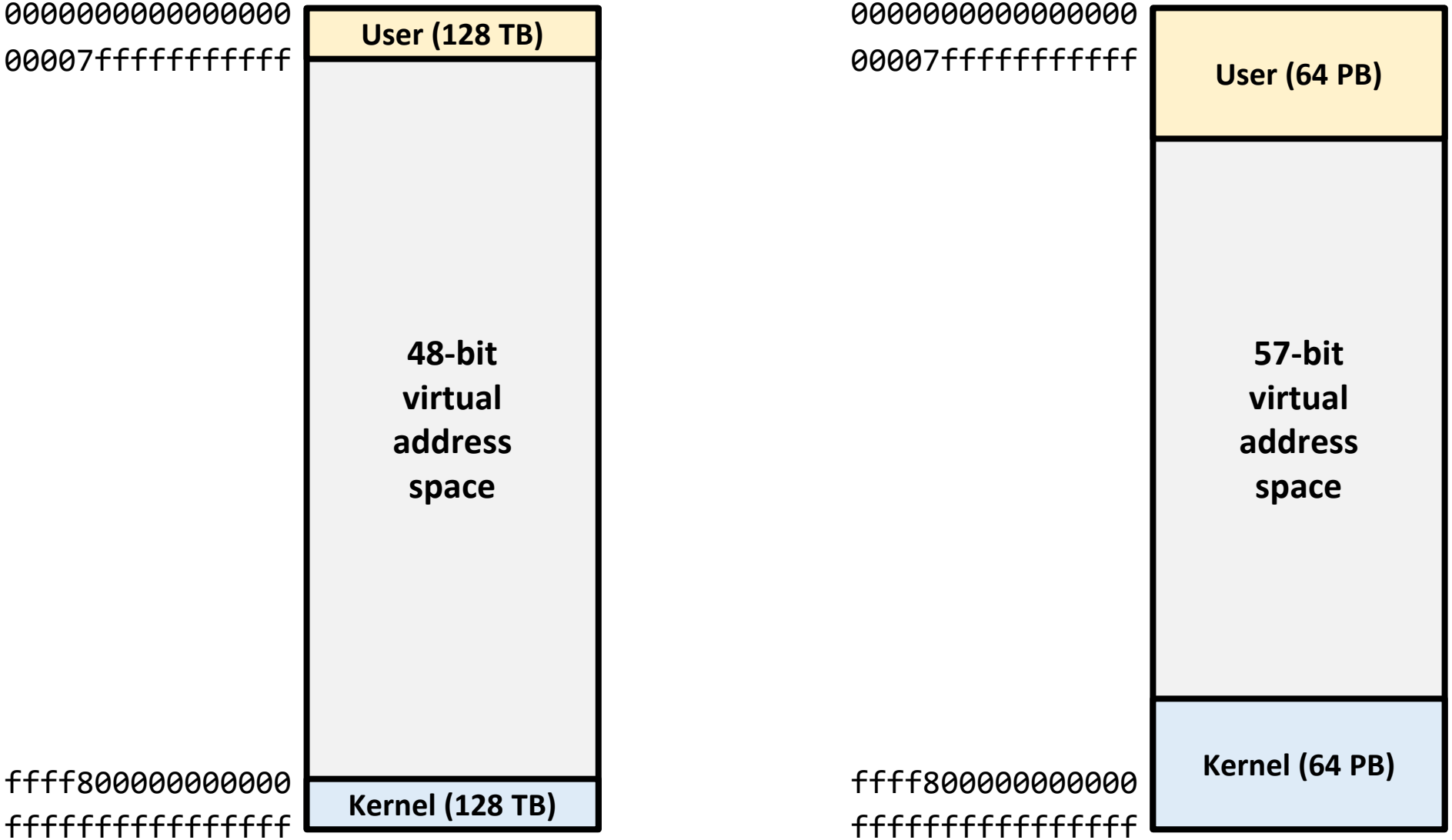
- Using a noncontiguous memory area
  - `void *vmalloc()`
  - `void vfree()`

Memory Allocation in the Kernel

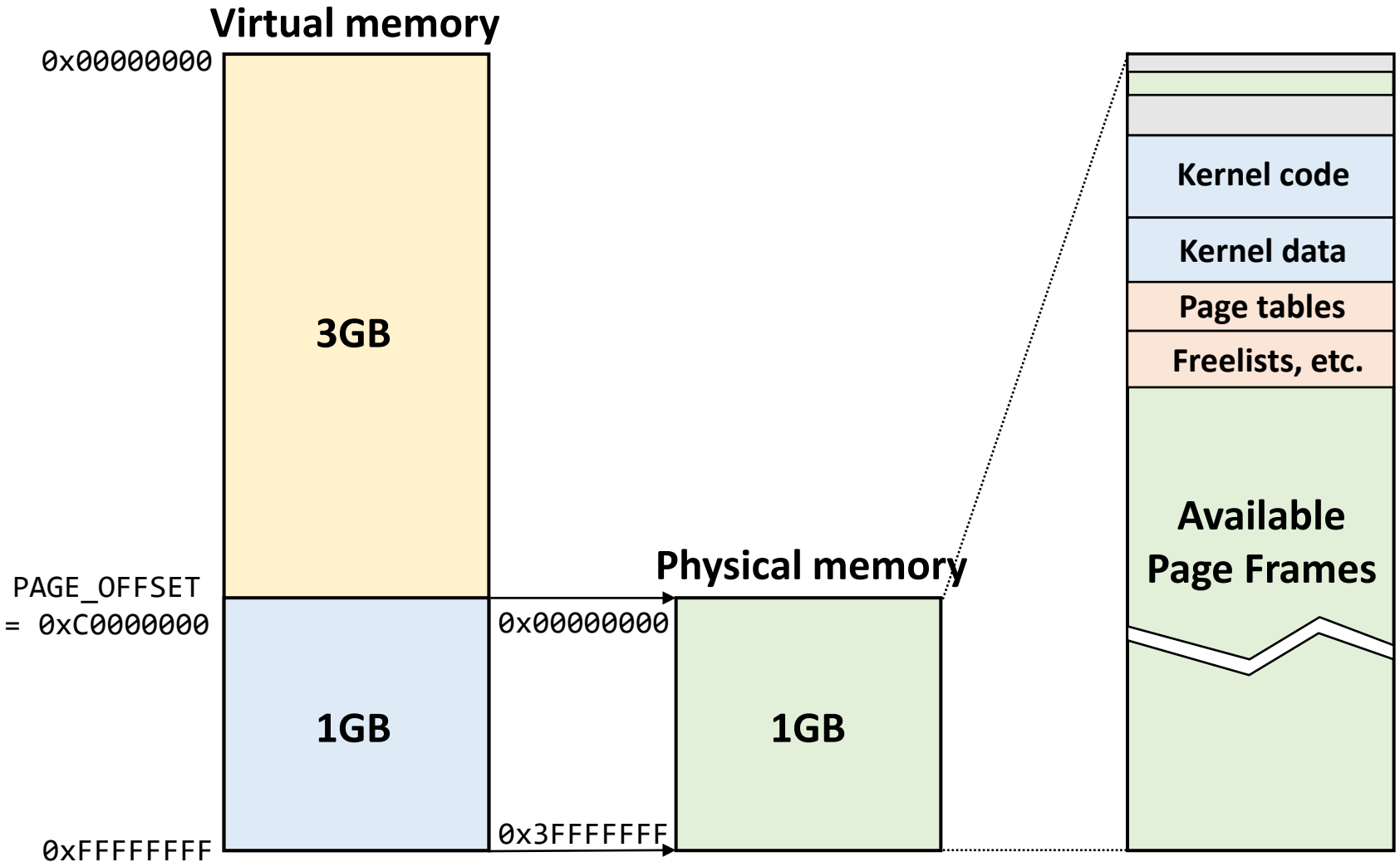
- Buddy allocator
  - Page-aligned
- Slab allocator
  - For various kernel data structures
  - Specific & general
  - On top of the buddy allocator
- Vmalloc
  - For large buffers
  - Virtually contiguous, but physically noncontiguous

Virtual Memory Management

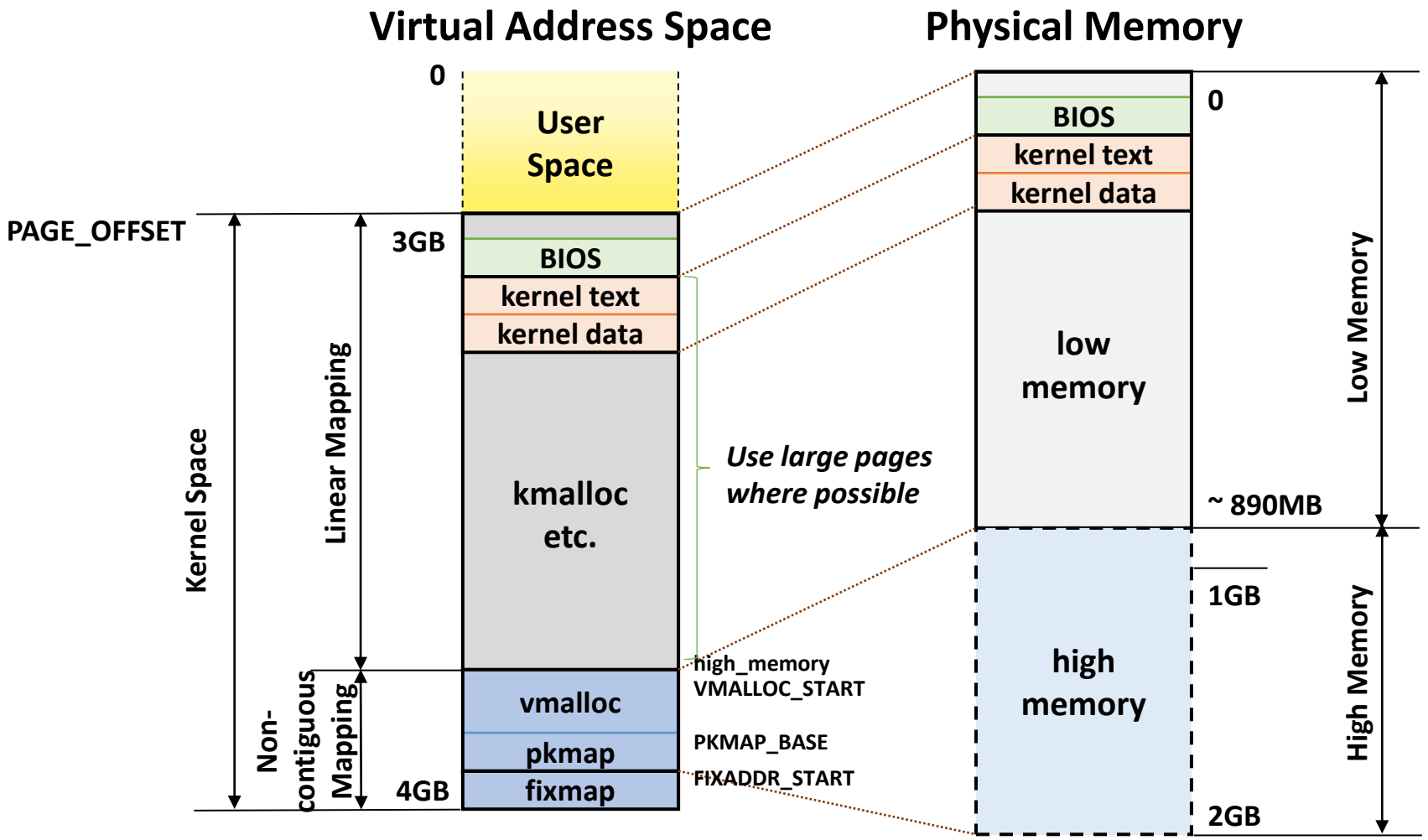
Virtual Address Map



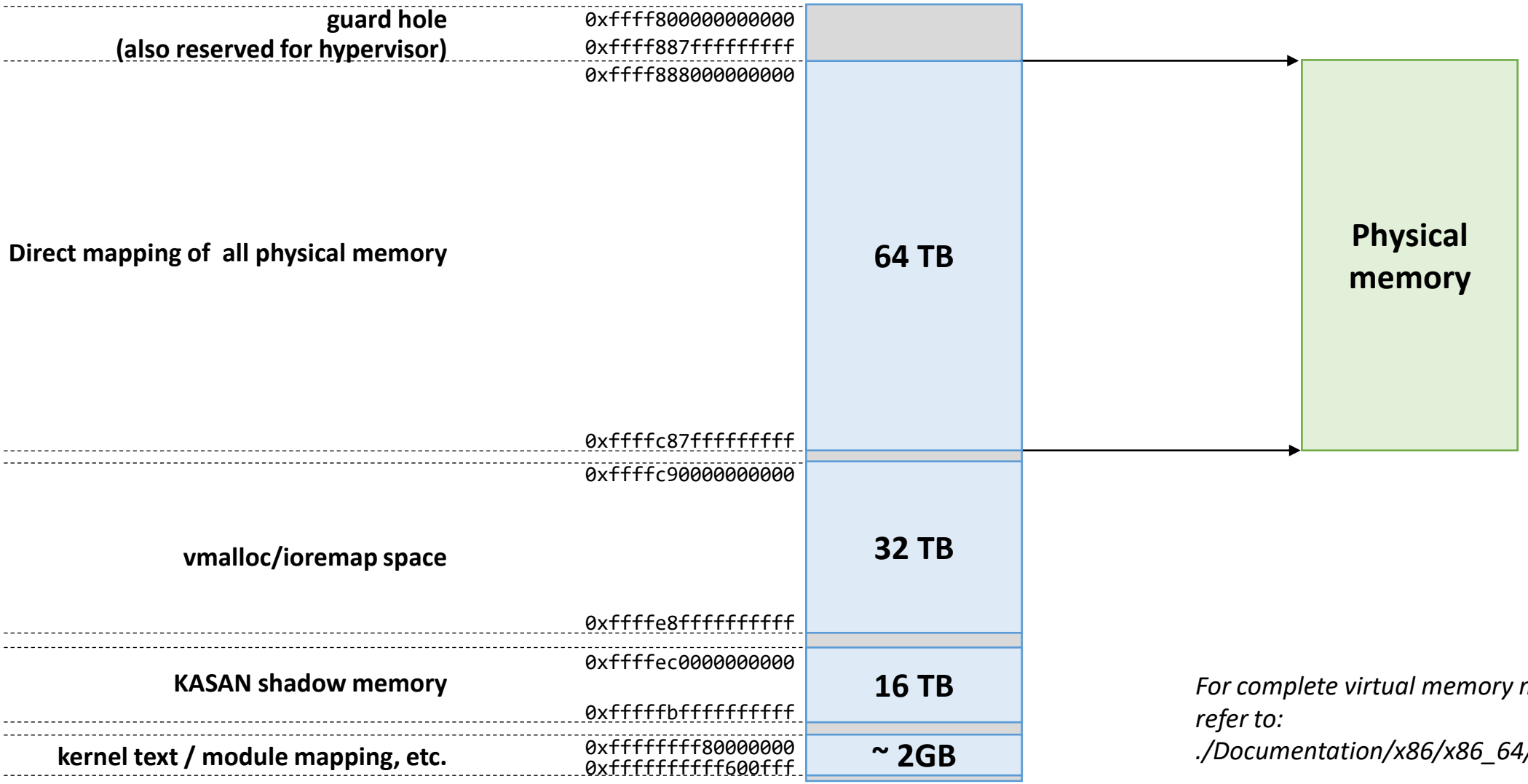
Kernel Address Space (i386, <1GB)



Kernel Address Space (i386)



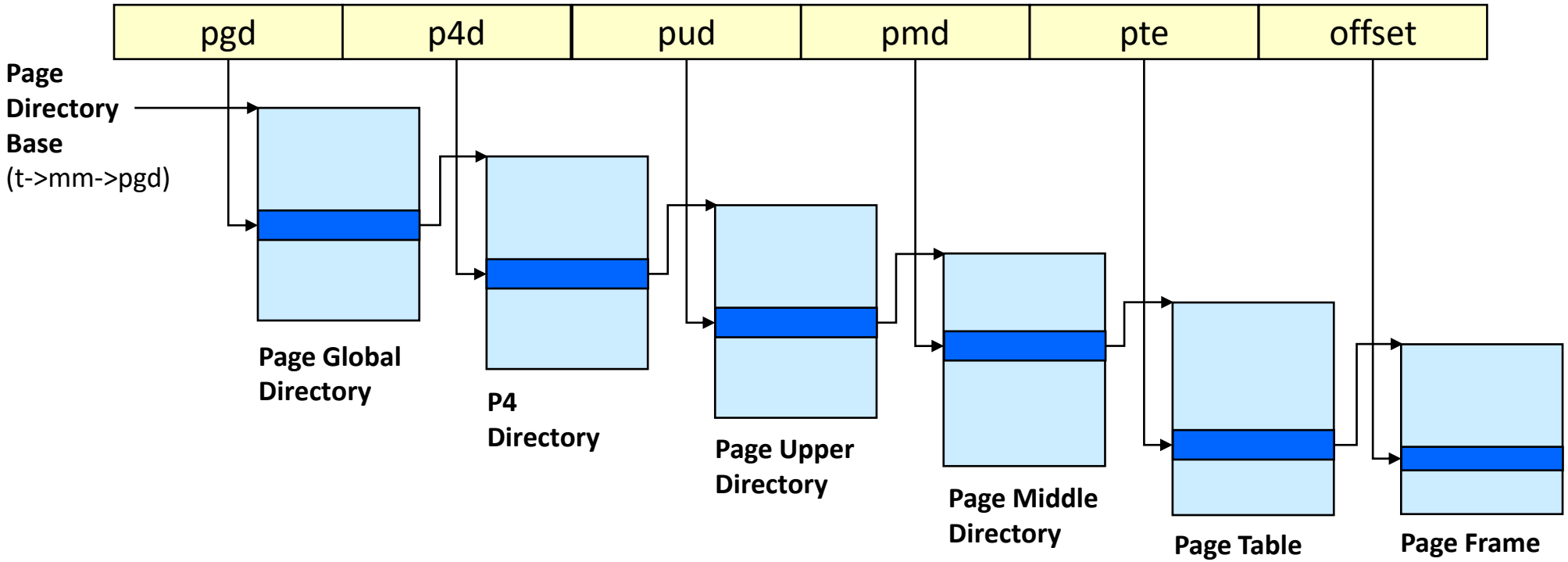
Kernel Address Space (x86_64, 48-bit)



For complete virtual memory map, refer to:
./Documentation/x86/x86_64/mm.rst

Paging in Linux

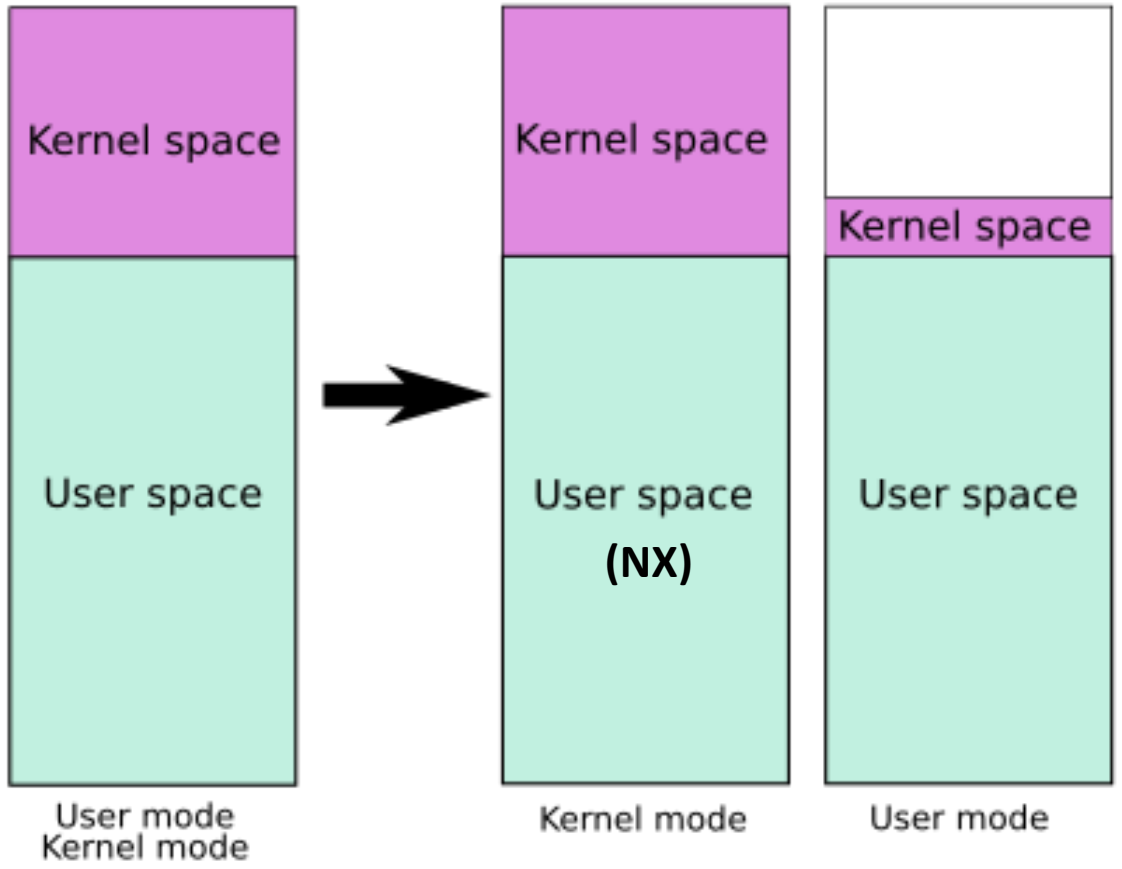
- Five-level address translation
  - 5-level paging for Intel "Ice Lake" processors and beyond (57-bit virtual address)
  - For 48-bit virtual address, the size of P4D is set to 1



Kernel Page-Table Isolation (KPTI)

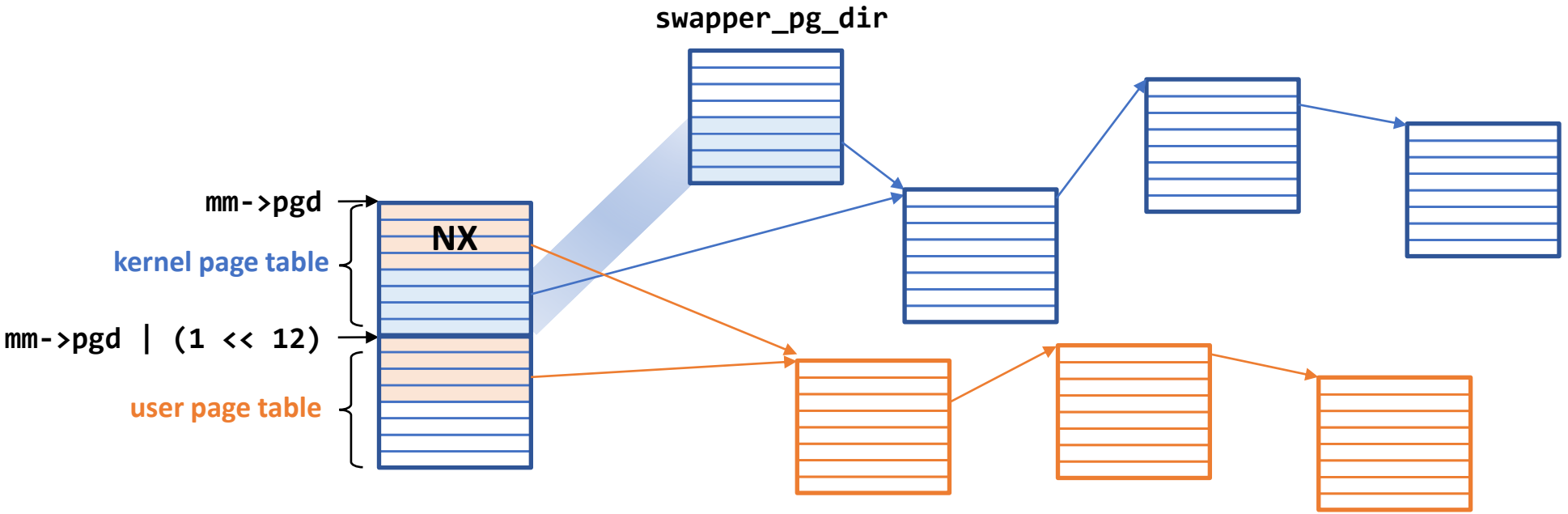
- To mitigate Meltdown vulnerability
- Separate page table for kernel
- Minimal kernel space for syscall, page fault & interrupt handling

- Merged in 4.15
- `CONFIG_PAGE_TABLE_ISOLATION=y`
- Disabled by '`nopti`' at boot time
- ASID becomes critical to the performance



Page Table Allocation

- When KPTI is enabled, two PGDs are allocated (@ `_pgd_alloc()`)
  - PGD_ALLOCATION_ORDER = 1 (8KB in size and 8KB-aligned)
  - One for kernel address space, the other for user address space
  - User portion of the kernel page table is set with the NX bit



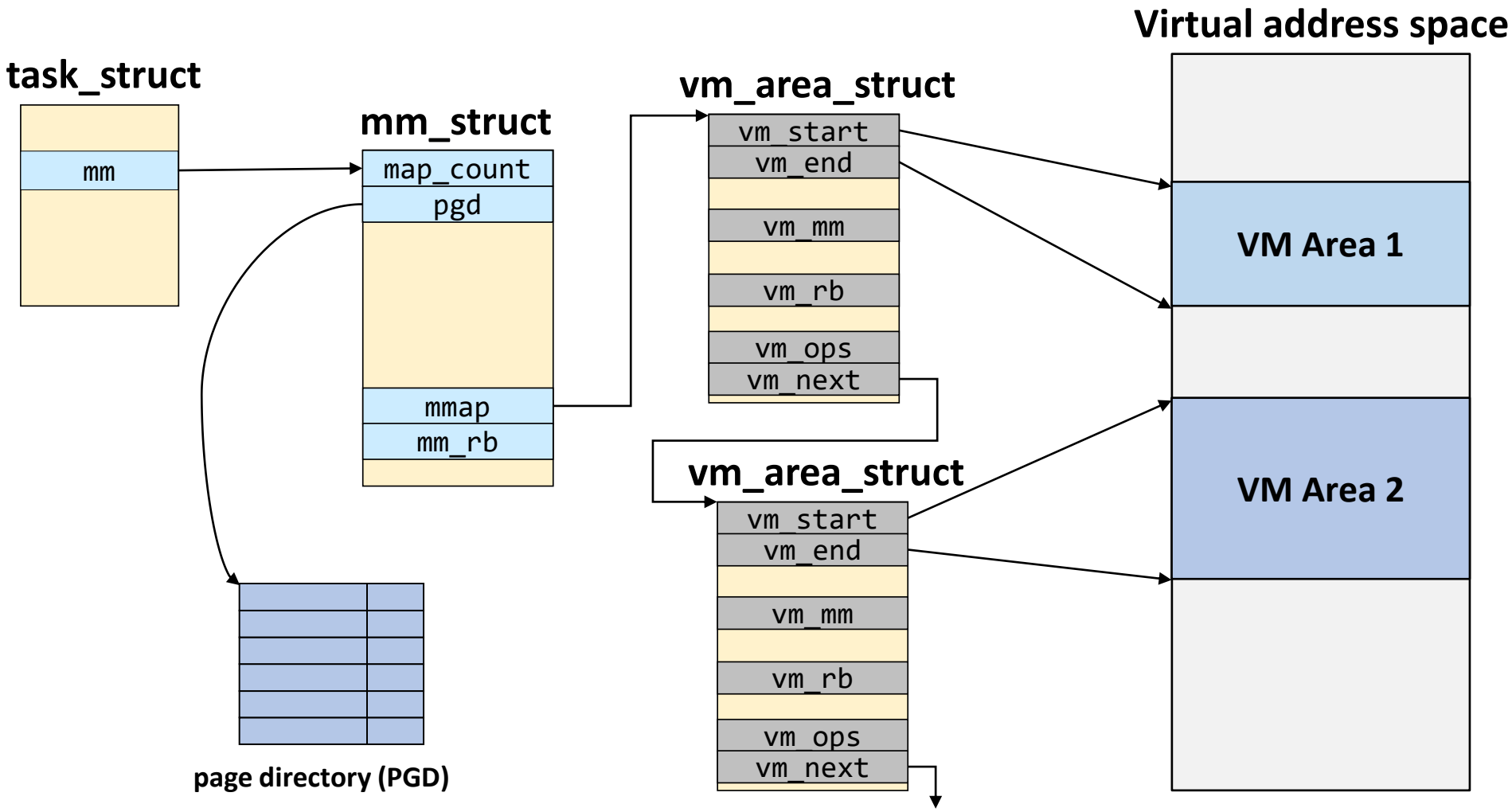
VMA

- Virtual Memory Area
  - A contiguous, page-aligned subset of the virtual address space
  - VMAs are linked with a red-black tree for fast lookup of the region corresponding to any virtual address
  - Described by a `vm_area_struct`
  - Either file-backed or anonymous
  - `/proc/PID/maps`

```
[sys:~-2028] cat /proc/self/maps
562a517de000-562a517e6000 r-xp 00000000 08:01 2228226                    /bin/cat
562a519e5000-562a519e6000 r--p 00007000 08:01 2228226                    /bin/cat
562a519e6000-562a519e7000 rw-p 00008000 08:01 2228226                    /bin/cat
562a52e6a000-562a52e8b000 rw-p 00000000 00:00 0                          [heap]
7f2244bd0000-7f2244db7000 r-xp 00000000 08:01 14286948                   /lib/x86_64-linux-gnu/libc-2.27.so
7f2244db7000-7f2244fb7000 ---p 001e7000 08:01 14286948                   /lib/x86_64-linux-gnu/libc-2.27.so
7f2244fb7000-7f2244fbb000 r--p 001e7000 08:01 14286948                   /lib/x86_64-linux-gnu/libc-2.27.so
7f2244fbb000-7f2244fbd000 rw-p 001eb000 08:01 14286948                   /lib/x86_64-linux-gnu/libc-2.27.so
7f2244fbd000-7f2244fc1000 rw-p 00000000 00:00 0
7f2244fc1000-7f2244fe8000 r-xp 00000000 08:01 14286873                   /lib/x86_64-linux-gnu/ld-2.27.so
7f224500e000-7f2245030000 rw-p 00000000 00:00 0
7f2245030000-7f22451cb000 r--p 00000000 08:01 262169                     /usr/lib/locale/locale-archive
7f22451cb000-7f22451cd000 rw-p 00000000 00:00 0
7f22451e8000-7f22451e9000 r--p 00027000 08:01 14286873                   /lib/x86_64-linux-gnu/ld-2.27.so
7f22451e9000-7f22451ea000 rw-p 00028000 08:01 14286873                   /lib/x86_64-linux-gnu/ld-2.27.so
7f22451ea000-7f22451eb000 rw-p 00000000 00:00 0
7fffdbb92000-7fffdbbb3000 rw-p 00000000 00:00 0                          [stack]
7fffdbbea000-7fffdbbed000 r--p 00000000 00:00 0                          [vvar]
7fffdbbed000-7fffdbbef000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
```

VMA *permission* *offset* *device* *i-node* *mapped file name*

VMA Structure



Lazy TLB Flushing

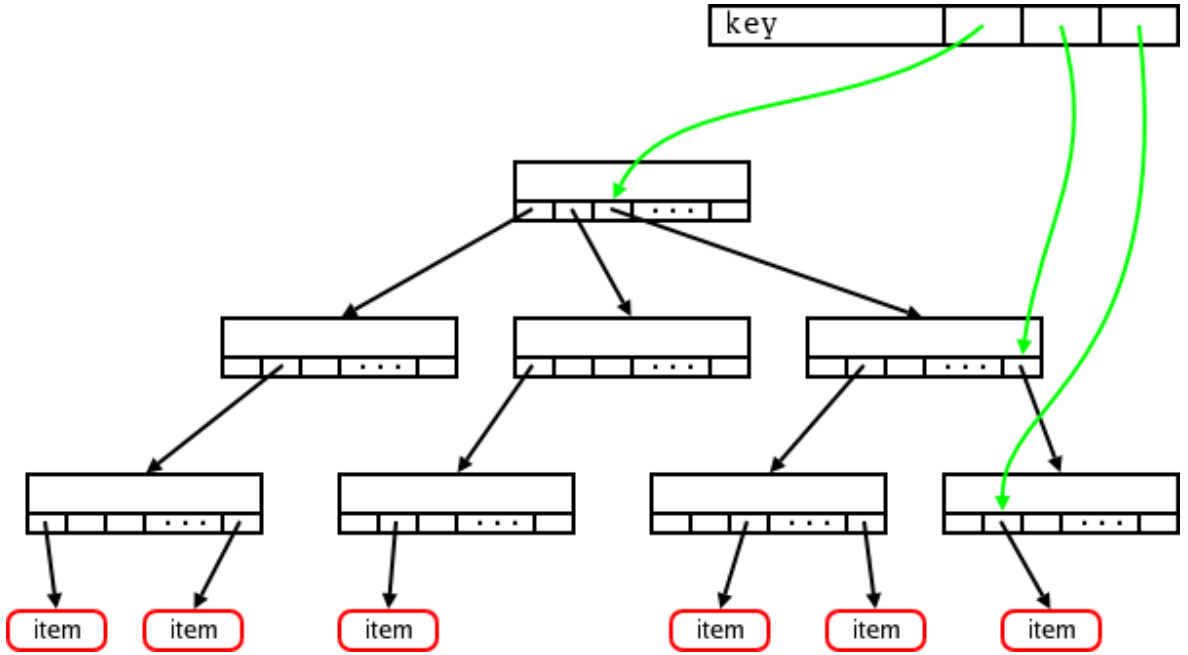
- TLB flush is expensive
- No need to flush TLB for a task without user virtual address space (i.e., kernel threads)

- Implementation
  - (`task_struct *`) `t->mm`: virtual address space of a process
  - (`task_struct *`) `t->active_mm`: the effective mm
  - Normally, `t->active_mm == t->mm`
  - On context switch, `t->active_mm->pgd` is stored into CR3
  - If the next task's `mm` is NULL, use previous task's `active_mm` to avoid TLB flush

Page Cache

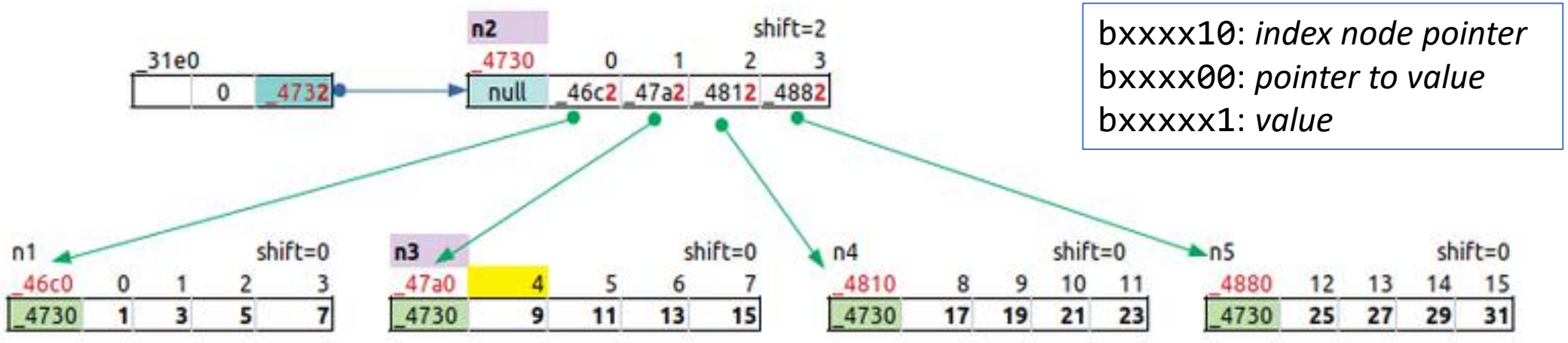
- A cache of pages in RAM
  - From reads and writes of regular filesystem files, block device files, mmap'ed files, …
  - Group cached pages belonging to the same inode

- Page cache lookup
  - Each inode has a unique radix tree
  - Key: <inode, page offset>
  - The radix tree points to the cached page
  - Fanout: 64
    (16 for small system)



Xarray (eXtensible Array)

- An abstract data type which behaves like a very large array of pointers
  - Radix tree is replaced with XArray since 4.19 (by Matthew Wilcox)
  - Can go to the next or previous entry in a cache-efficient manner compared to a hash
  - No need to copy data or change MMU mappings compared to a resizable array
  - More memory-efficient, parallelizable and cache-friendly than a doubly-linked list
  - Perform lookups without locking using RCU



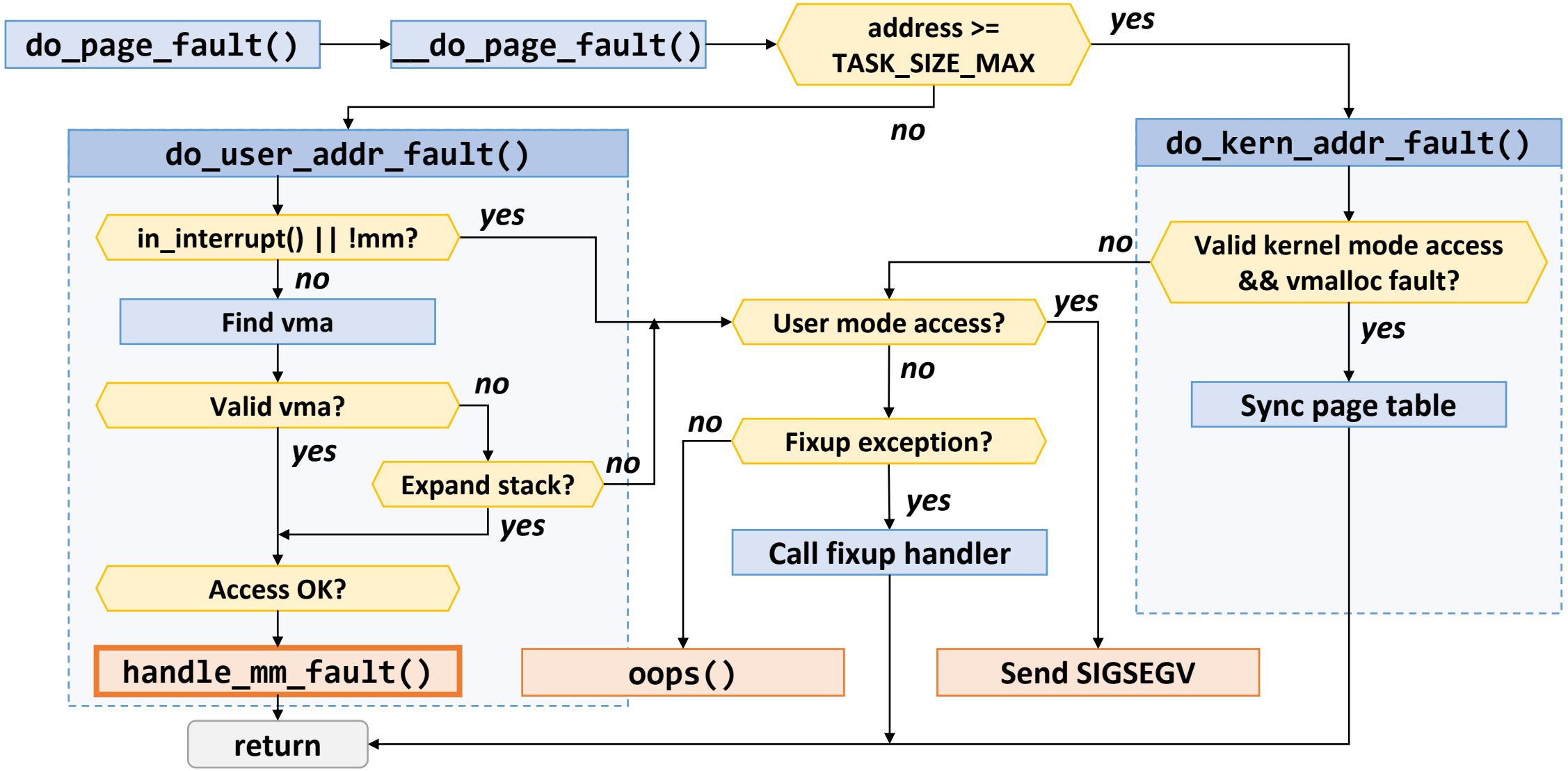
bxxxx10: *index node pointer*
bxxxx00: *pointer to value*
bxxxxx1: *value*

Source: https://www.kernel.bz/boardPost/118679/19

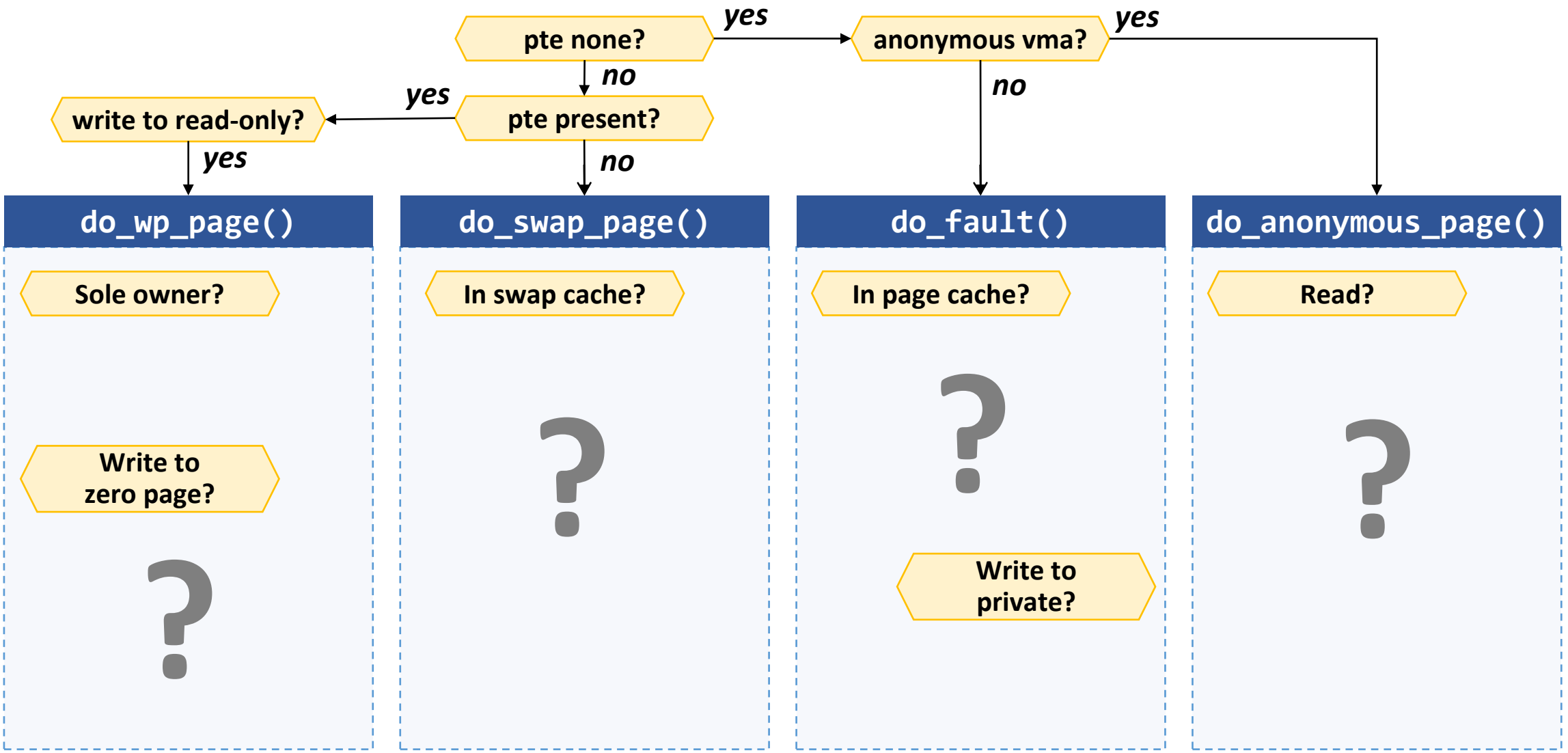
Classifying Page Faults

	User address	Kernel address
User mode	Invalid user address: SIGSEGV Valid user address: Normal page faults	Invalid kernel address: SIGSEGV
Kernel mode	Fixup address: Jump to the fixup handler Kernel bug: OOPS	Inconsistent PGD: Synchronize PGD Fixup address: Jump to the fixup handler Kernel bug: OOPS

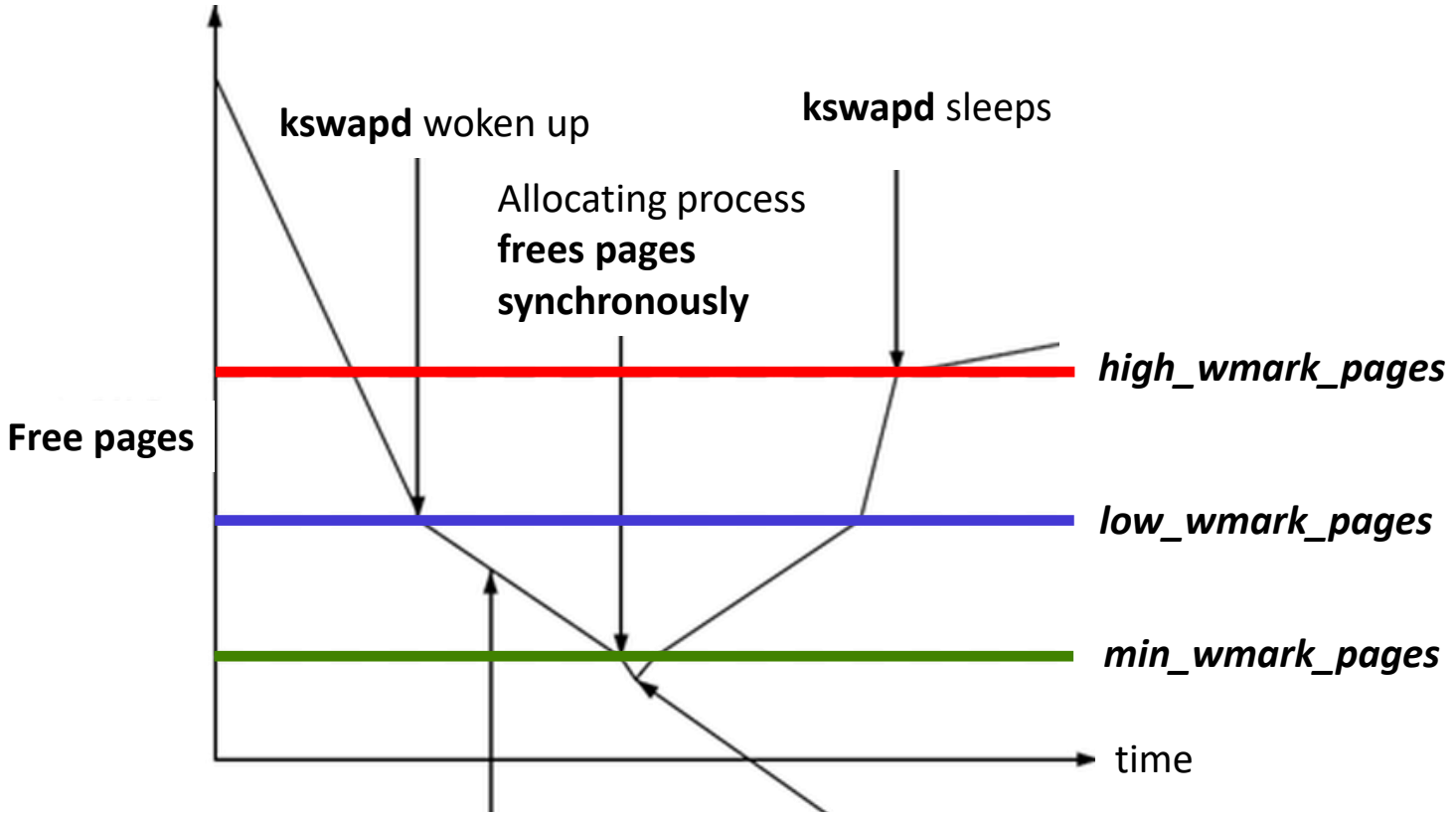
Page Fault Handling



Handling Normal Page Faults



Swapping in Linux



Rate of page consumption is slowed by kswapd but still allocating too fast

GFP_ATOMIC allocation can go below *min_wmark_pages*

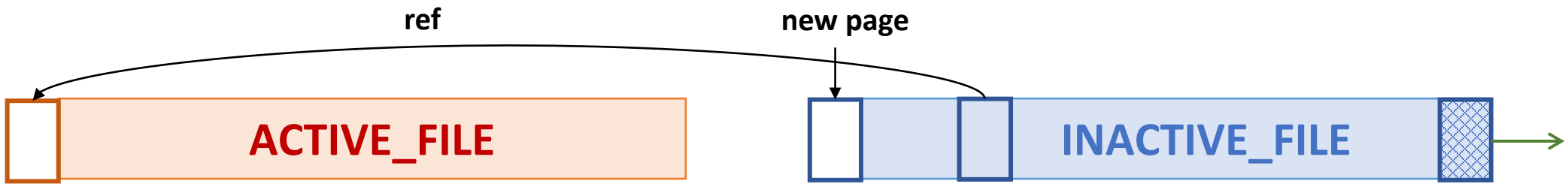
LRU Lists

- `LRU_INACTIVE_ANON`: for inactive anonymous pages
- `LRU_ACTIVE_ANON`: for active anonymous pages
- `LRU_INACTIVE_FILE`: for inactive file-backed pages
- `LRU_ACTIVE_FILE`: for active file-backed pages
- `LRU_UNEVICTABLE`: for unevictable pages (ramfs, locked pages, etc.)

- Why separate lists for ANON and FILE pages?
  - Page cache pages may be hidden behind lots of anonymous pages on the LRU
  - The kernel scans over pages that should not be evicted (e.g., scanning anonymous pages when there is no swap)

Refault Distance

- Inactive_age: total # of pages removed from INACTIVE_FILE
  = # of pages evicted from INACTIVE_FILE +
  # of pages promoted from INACTIVE_FILE to ACTIVE_FILE
- *E* = inactive_age(t_e) when a page is evicted from memory
- *R* = inactive_age(t_r) when the page is fetched to memory again
- Refault distance = *R* – *E*: total # of pages removed from INACTIVE_FILE while the page was outside of memory



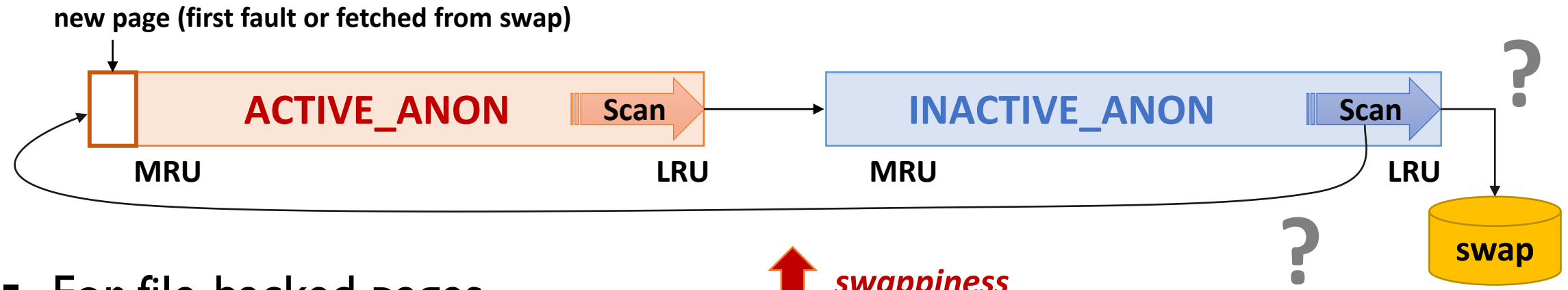
Promotion using Refault Distance

- Minimum access distance D = (R – E) + *nr_inactive*(t_r)
- Check whether D <= *nr_inactive*(t_r) + *nr_active*(t_r) or (R – E) <= *nr_active*(t_r)
- If (R – E) <= *nr_active*(t_r) then the page goes to ACTIVE_FILE
  Otherwise, it goes to INACTIVE_FILE

- Implementation
  - Page cache maintains shadow entries for evicted pages
  - The shadow entry records inactive age (E)
  - On refault, compare (R – E) with *nr_active*

Page Reclaim

- For anonymous pages



- For file-backed pages

swappiness