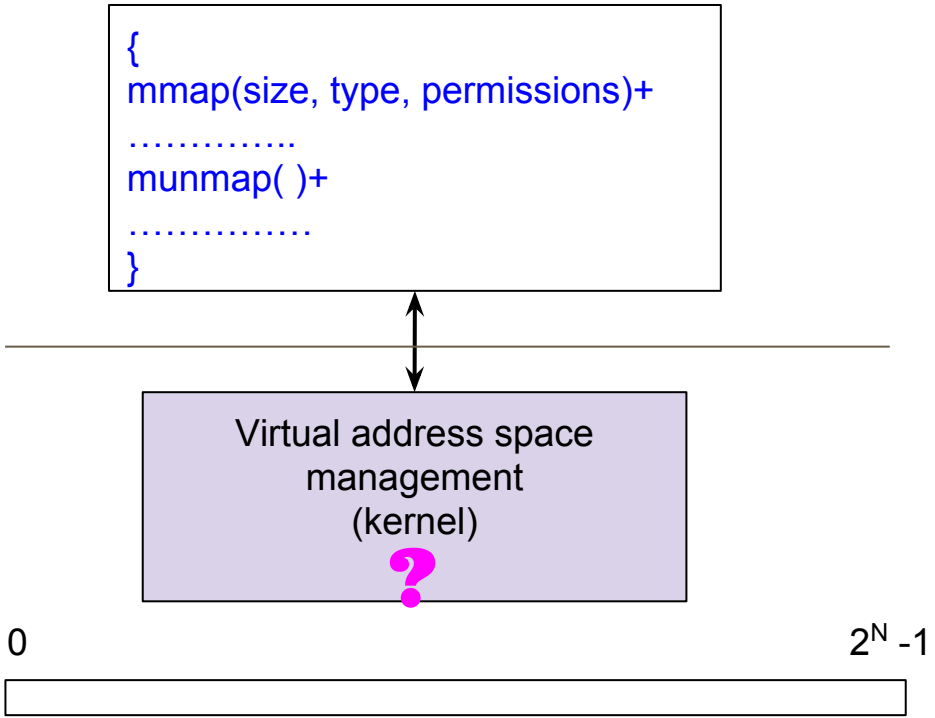
Linux Memory Management

Virtual memory layout: a closer look

- Applications require memory with different properties
  - access permissions
  - sharing
  - file backed vs. anonymous
  - dynamically sized
- /proc/{pid}/maps and mmap() system call

- Why OS should worry how user-space virtual addresses are managed?
  - let a user-space library handle it
  - only virtual to physical translation is managed by OS
  - possible?

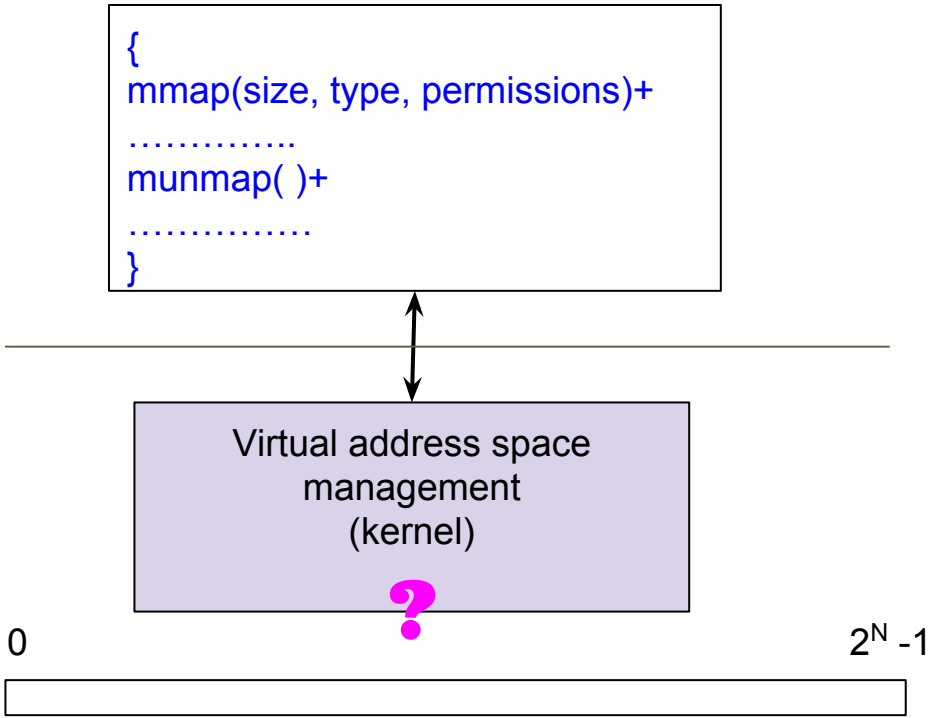
Managing virtual memory (user space address)



Some design consideration

- virtual address space is quite large (for 64-bit)
- can not assume virtual address usage size
- efficiency concerns: CPU and Memory
- address space requirements → hardware structures (MMU)

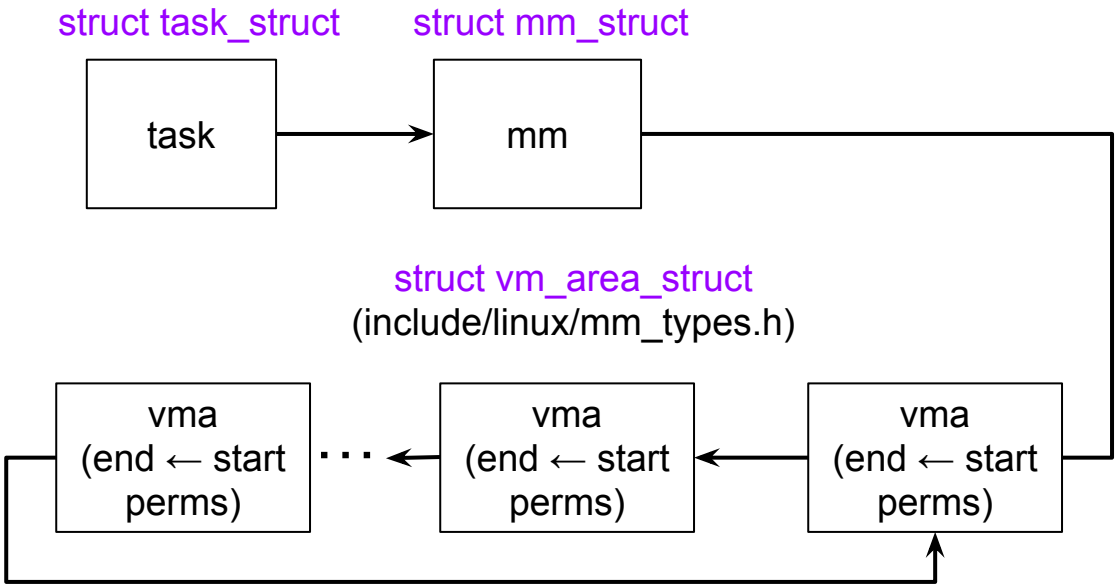
Managing virtual memory (user space address)



Virtual address space management alternatives

- ➔ contiguous allocation based on memory region type
  - inflexible
  - scalability issues
- ➔ sparse allocation
  - sorted list of used ranges
  - scalability issues
    - Can be solved using balanced search trees

How linux does it?



- ➔ start and end never overlaps between two vm areas
- ➔ can merge/extend vmas if permissions match
- ➔ linux maintains both rb_tree and a sorted list (see mm/filemap.c)

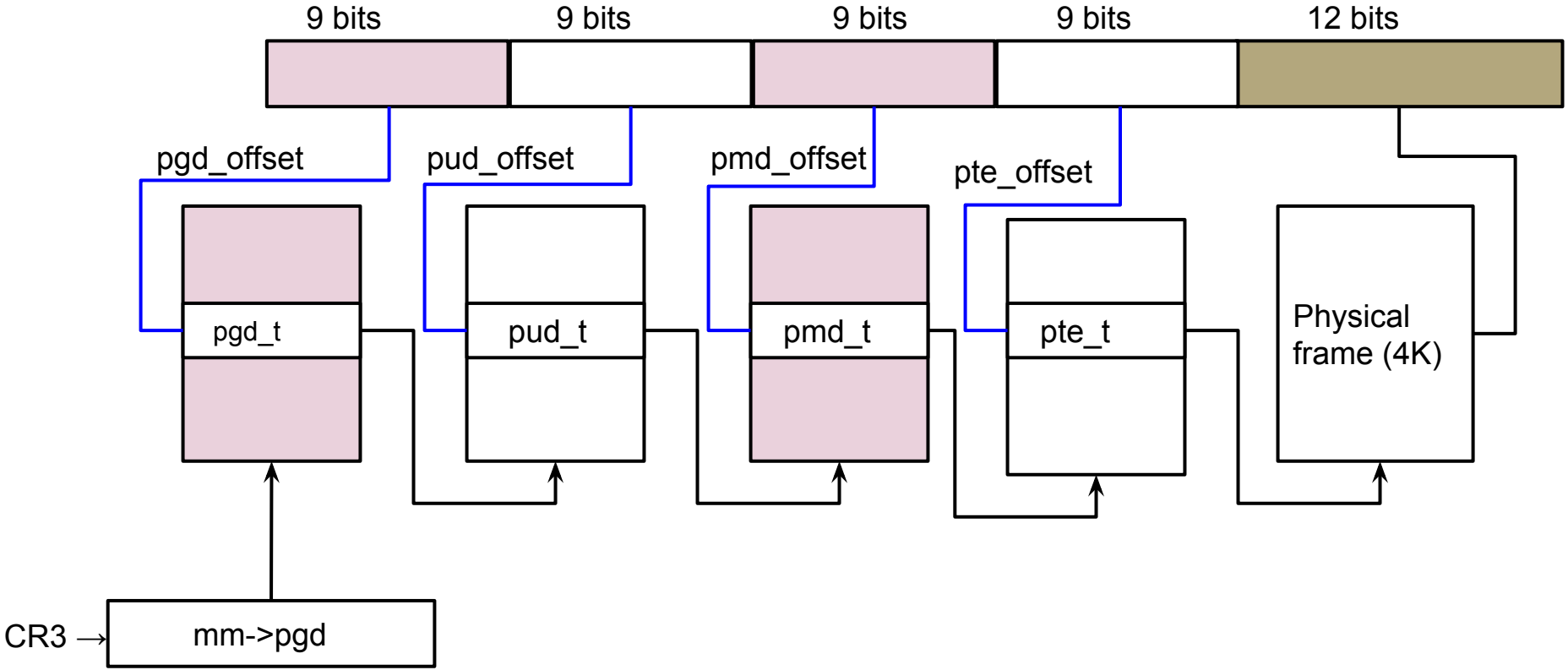
Example usage

- mmap( ), munmap( ), mremap( ) system calls
  - some useful calls: find_vma( ) , get_unmapped_area( ), vma_merge ()
  - can be found in mm/mm.c
- Page fault handler
  - require vm_area access permissions to fix the page fault
  - Ex: fault handling for a read-only vm_area vs. read-write vm_area
- Feature: vma-area specific page fault handler
  - struct vm_operations_struct *vm_ops
  - mechanism to register call backs on page fault (and some other events)

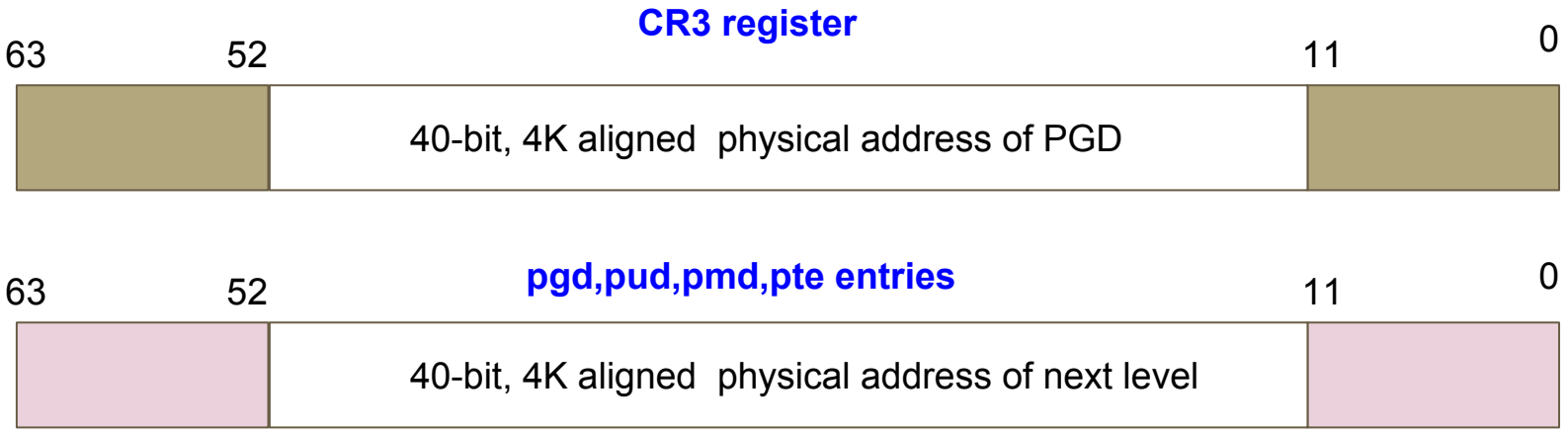
Demand paging: background

- ➔ Why not use physical addressing?
- ➔ Considering application expectation of virtual address space flexibility
  - ◆ Why not use segmentation-only design?
  - ◆ Why use paging?
- ➔ Challenges in paging
  - ◆ Size of translation meta-data
  - ◆ Additional memory accesses during translation

4-level page tables (48-bit virtual address)



X86_64 page table entries (48-bit)



Some important flags

0 (present/absent) 1 (read/write) 2 (user/supervisor), 5(accessed) 7(huge page)
63(execute permissions)

*Source: Intel manual Vol: 3A 4.5