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

Systems Software &
Architecture Lab.

Seoul National University

Spring 2019

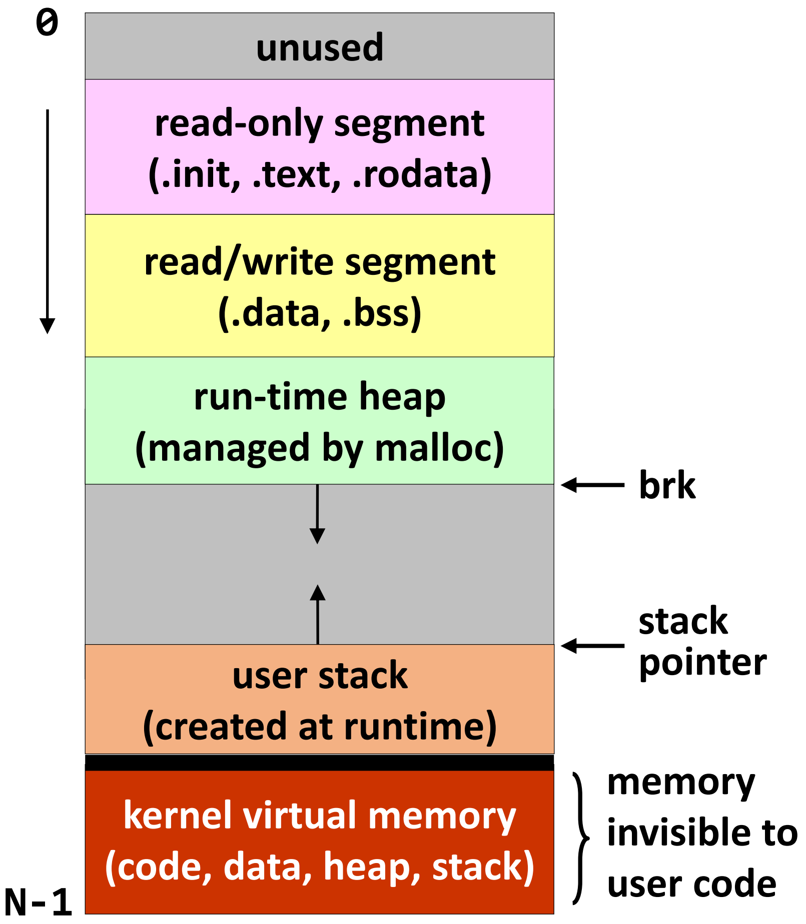
Virtual Memory

Virtual Memory: Goals

- Transparency
  - Processes should not be aware that memory is shared
  - Provides a convenient abstraction for programming (a large, contiguous space)
- Efficiency
  - Minimizes fragmentation due to variable-sized requests (space)
  - Gets some hardware support (time)
- Protection
  - Protect processes and the OS from another process
  - Isolation: a process can fail without affecting other processes
  - Cooperating processes can share portions of memory

(Virtual) Address Space

- Process' abstract view of memory
  - OS provides an illusion of private address space to each process
  - Contains all of the memory state of the process
  - Static area
    - Allocated on exec()
    - Code & Data
  - Dynamic area
    - Allocated at runtime
    - Can grow or shrink
    - Heap & stack

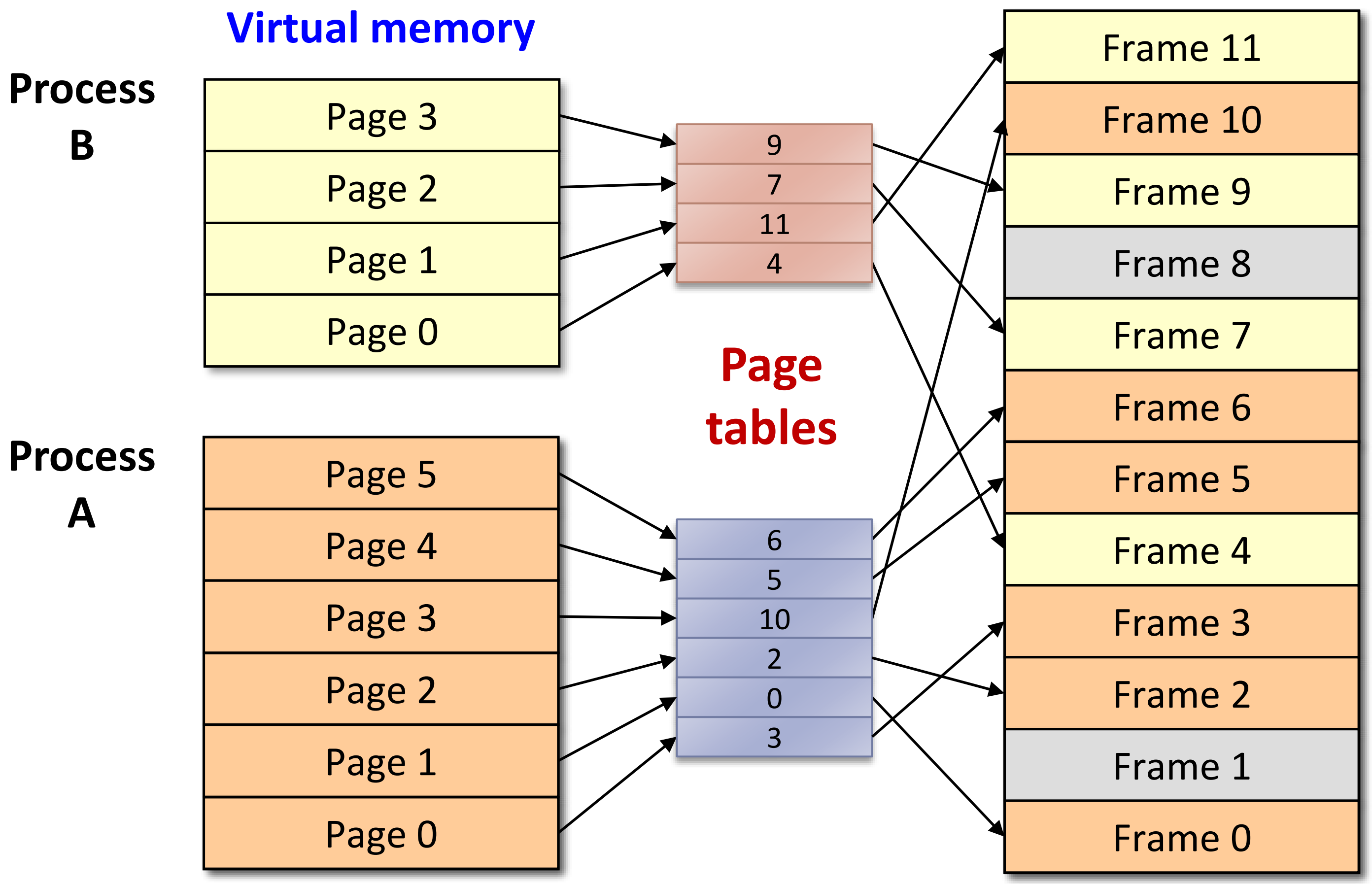


Paging

- Allows the physical address space of a process to be noncontiguous
  - Divide virtual memory into blocks of same size (pages)
  - Divide physical memory into fixed-size blocks (frames)
  - Page (or frame) size is power of 2 (typically 512B – 8KB)

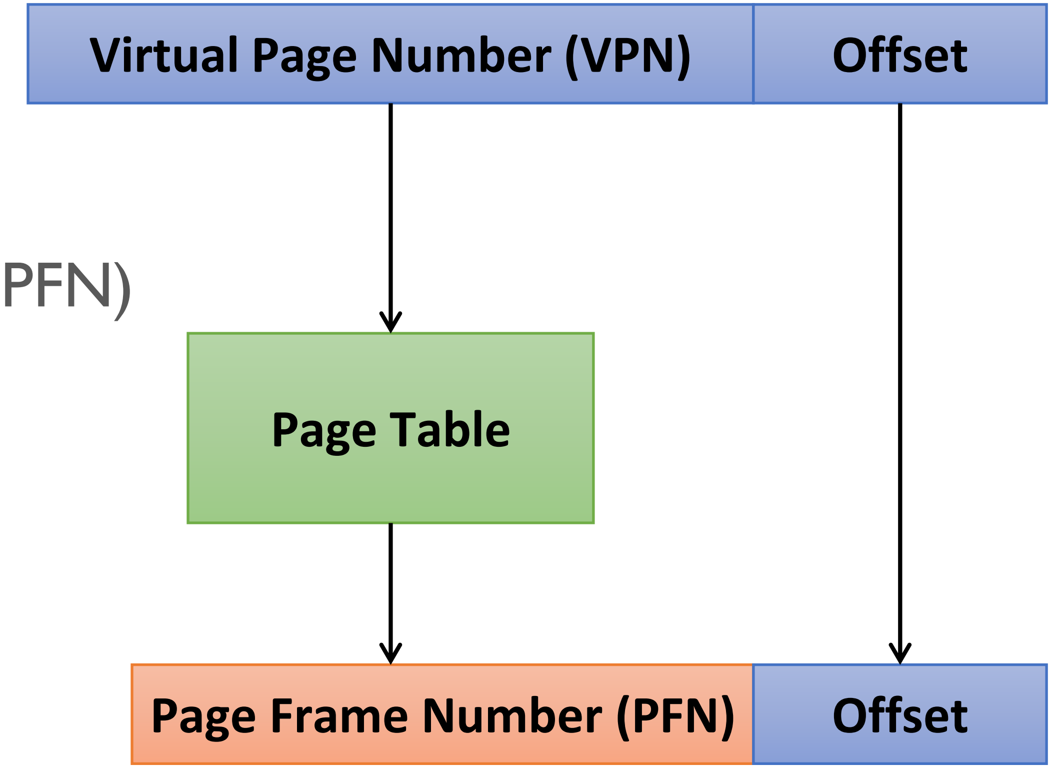
- Eases memory management
  - OS keeps track of all free frames
  - To run a program of size *n* pages, need to find *n* free frames and load the program
  - Set up a page table to translate virtual to physical addresses
  - No external fragmentation

Paging Example



Address Translation

- Translating virtual addresses
  - A virtual address has two parts: <VPN, Offset>
  - VPN is an index into the page table
  - Page table determines Page Frame Number (PFN)
  - Physical address is <PFN, Offset>
  - Usually, |VPN| >= |PFN|



- Page tables
  - Managed by OS
  - Map VPN to PFN
  - One Page Table Entry (PTE) per page in virtual address space

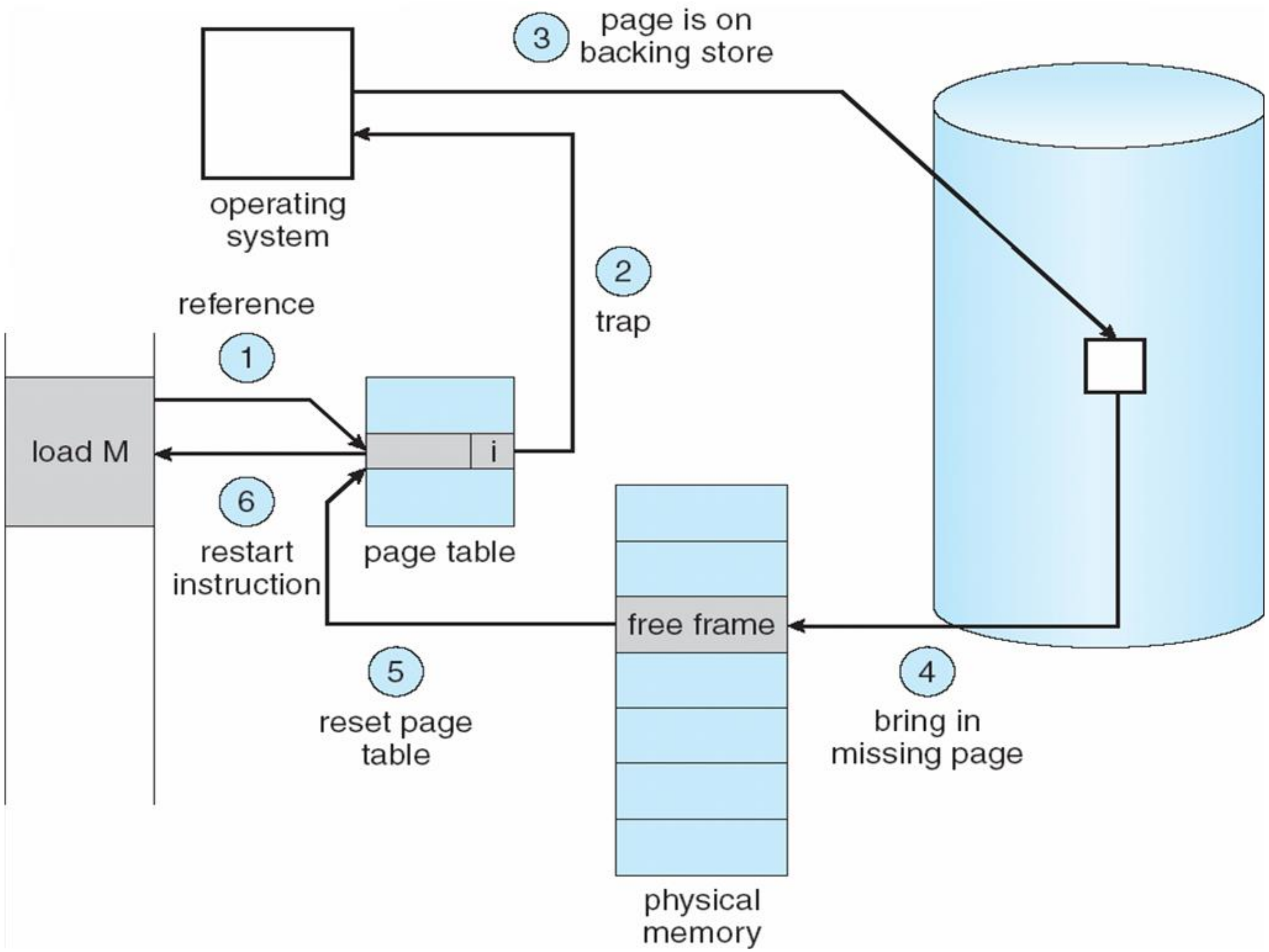
Demand Paging

- OS uses main memory as a (page) cache of all the data allocated by processes in the system
  - Bring a page into memory only when it is needed
  - Pages can be evicted from their physical memory frames
  - Evicted pages go to disk (only dirty pages are written)
  - Movement of pages is transparent to processes
- Benefits
  - Less I/O needed
  - Less memory needed
  - Faster response
  - More processes

Page Fault

- An exception raised by CPU when accessing invalid PTE
- Major page faults
  - The page is valid but not loaded into memory
  - OS maintains information on where to find the contents
  - Require disk I/Os
- Minor page faults
  - Page faults can be resolved without disk I/O
  - Used for lazy allocation (e.g. accesses to stack & heap pages)
  - Accesses to prefetched pages, etc.
- Invalid page faults
  - Segmentation violation: the page is not in use

Handling Page Fault



Paging: Pros

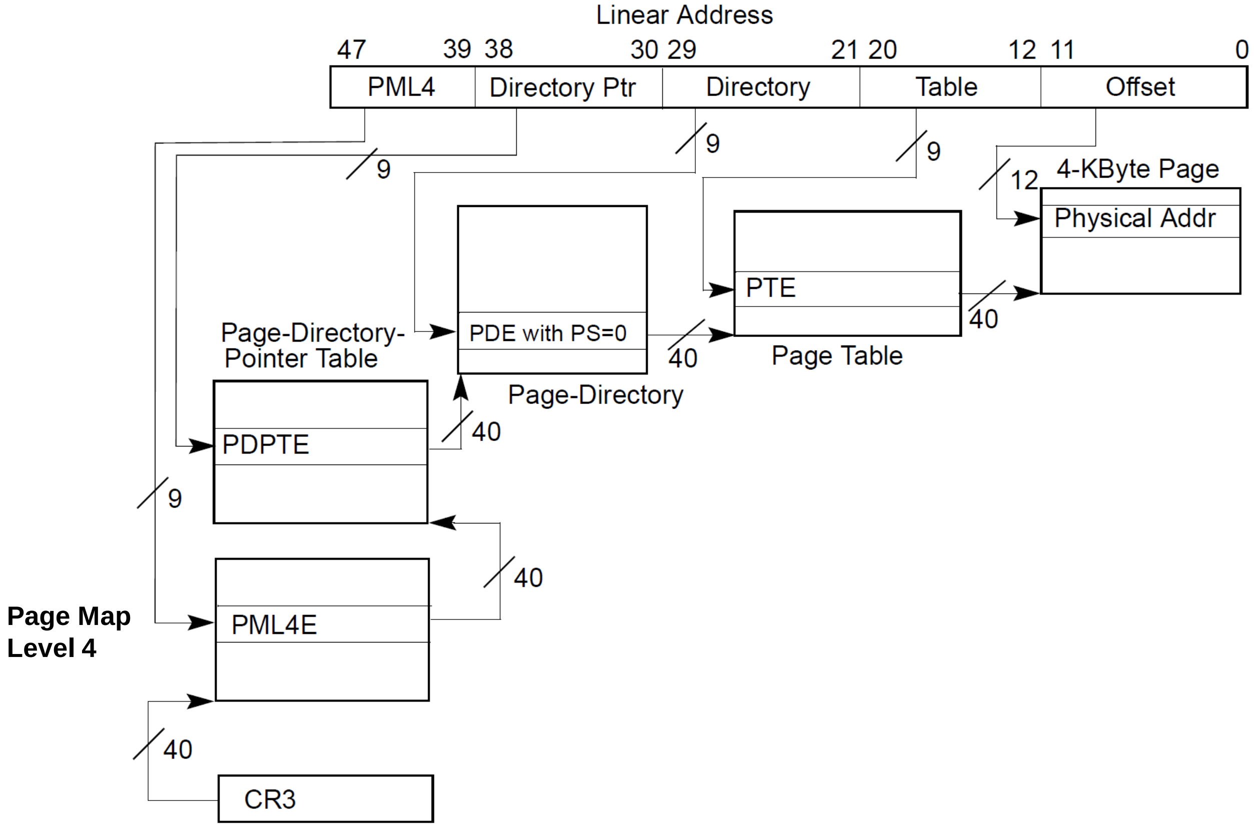
- No external fragmentation
- Fast to allocate and free
  - A list or bitmap for free page frames
  - Allocation: no need to find contiguous free space
  - Free: no need to coalesce with adjacent free space
- Easy to "page out" portions of memory to disk
  - Page size is chosen to be a multiple of disk block sizes
  - Use valid bit to detect reference to "paged-out" pages
  - Can run process when some pages are on disk
- Easy to protect and share pages

Paging: Cons

- Internal fragmentation
  - Wasted memory grows with larger pages
- Memory reference overhead
  - Page table stored in memory
  - Address translation increases latency
  - Solution: get hardware support (TLBs)
- Storage needed for page tables
  - Needs one PTE for each page in virtual address space
  - 32-bit virtual address space with 4KB pages: 4MB per page table
  - Page table for each process
  - Solution: store valid PTEs only or page the page table

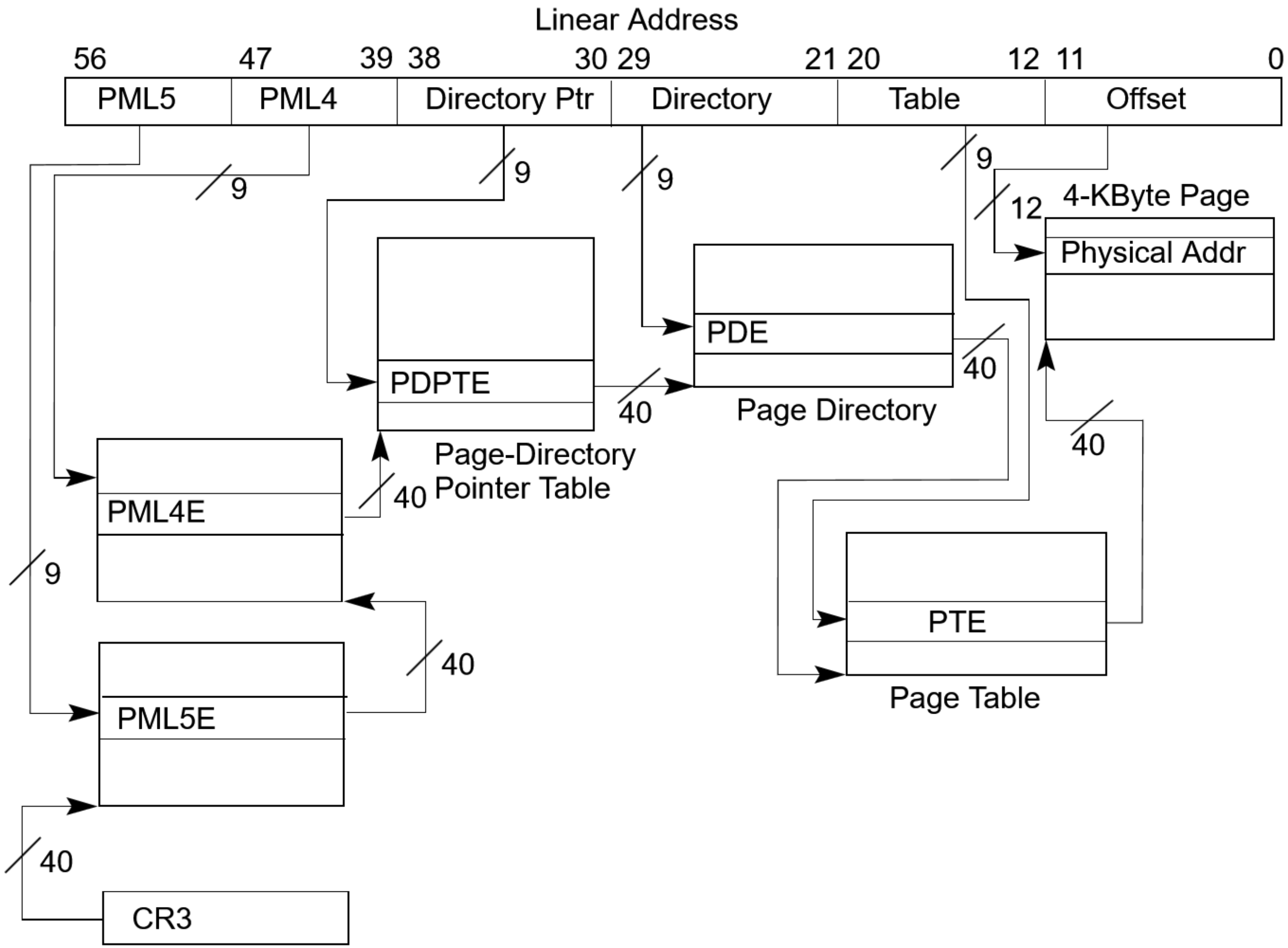
Four-level Page Table

- IA-32e paging mode in Intel 64
  - 48-bit "*linear*" address → 52-bit physical address (4KB page)

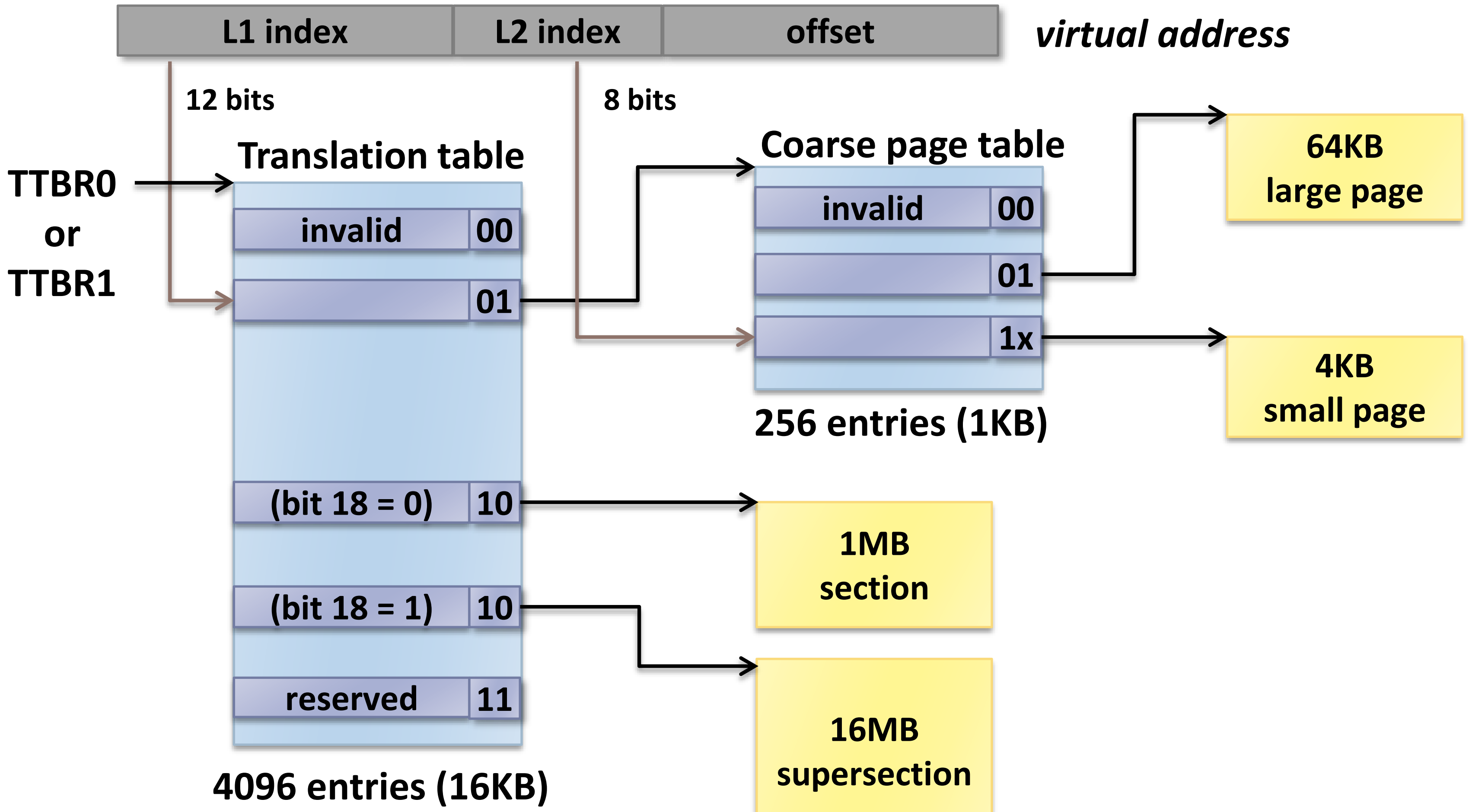


Five-level Page Table (planned)

- 57-bit virtual address space
- Supported by Linux since 4.14



ARMv7 Page Tables (without LPAE)



TLB

- Translation Lookaside Buffer
  - A hardware cache of popular virtual-to-physical address translations
  - Essential component which makes virtual memory possible

- TLB exploits locality
  - Temporal locality: an instruction or data item that has been recently accessed will likely be re-accessed soon
    - Instructions and data accesses in loops, …
  - Spatial locality: if a program accesses memory at address *x*, it will likely soon access memory near *x*
    - Code execution, array traversal, stack accesses, …

TLB Organization

- TLB is implemented in hardware
  - Processes only use a handful of pages at a time
    - 16~256 entries in TLB is typical
  - Usually fully associative
    - All entries looked up in parallel
    - But may be set associative to reduce latency
  - Replacement policy: LRU (Least Recently Used)
  - CPU knows where page tables are in memory (PTBR)
    - e.g. CR3 (or PDBR) register in x86
  - TLB actually caches the whole PTEs, not just PFNs

Valid	Tag (VPN)	Value (PTE)				
1	0x1000	V	R	M	Prot	PFN 0x1234
1	0x2400	V	R	M	Prot	PFN 0x8800
0	-	-				

TLB on Context Switch

- Flush TLB on each context switch
  - TLB is flushed automatically when PTBR is changed in a hardware-managed TLB
  - Some architectures support the pinning of pages into TLB
    - For pages that are globally-shared among processes (e.g. kernel pages)
    - MIPS, Intel, etc.

- Track which entries are for which process
  - Tag each TLB entry with an ASID (Address Space ID)
  - A privileged register holds the ASID of the current process
  - MIPS / ARMv7-A support 8-bit ASID
  - ARMv8-A supports 8-bit/16-bit ASID
  - Intel 64 supports 12-bit PCID (Process Context ID) – Since Westmere (2010)

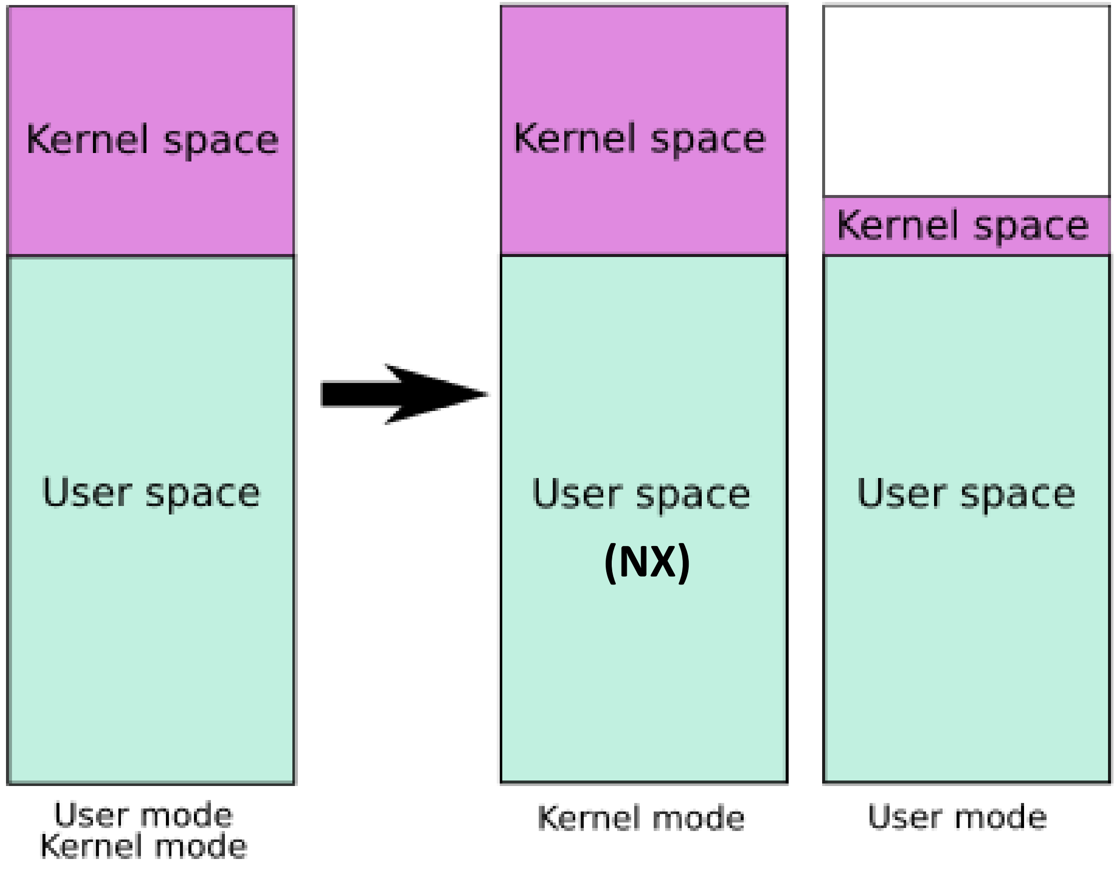
TLB Performance

- TLB is the source of many performance problems
  - Performance metric: hit rate, lookup latency, …
- Increase TLB reach (= # TLB entries * Page size)
  - Increase the page size: e.g. 2MB/1GB in Intel 64, 4KB/16KB/64KB in ARMv8
  - Increase the TLB size
- Use multi-level TLBs
  - e.g. Intel Haswell (4KB pages):
    L1 ITLB 128 entries (4-way), L1 DTLB 64-entries (4-way) +
    L2 STLB 1024 entries (8-way)
- Change your algorithms and data structures to be TLB-friendly

Kernel Page-Table Isolation (KPTI)

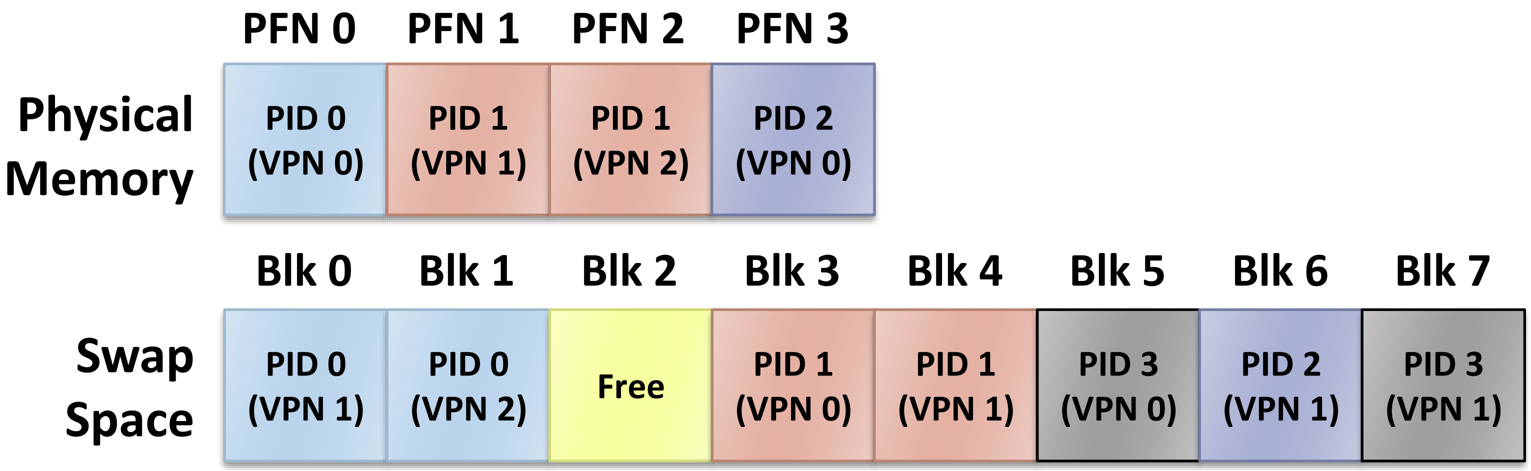
- To mitigate Meltdown vulnerability
- Separate page table for kernel
- Minimal kernel space for syscall, page fault & interrupt handling

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



Swapping: Where to Swap

- Swap space
  - Disk space reserved for moving pages back and forth
  - The size of the swap space determines the maximum number of memory pages that can be in use
  - Block size is same as the page size
  - Can be a dedicated partition or a file in the file system

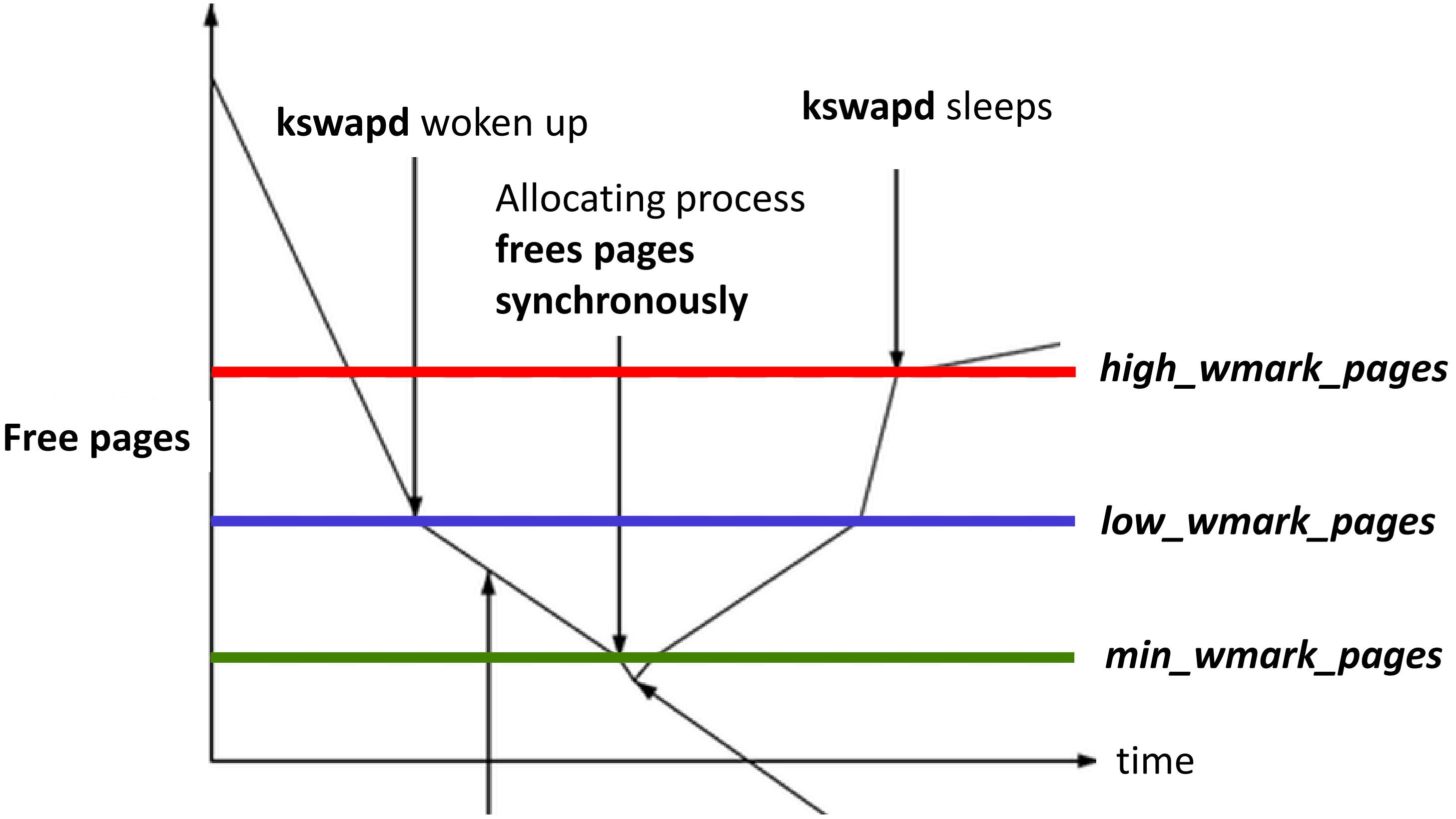


When to Swap

- Proactively based on thresholds
  - OS wants to keep a small portion of memory free
  - Two threshold values:
    HW (high watermark) and *LW* (low watermark)
  - A background thread called swap daemon (or page daemon) is responsible for freeing memory
    - e.g. kswapd in Linux
  - If (# free pages < *LW*), the swap daemon starts to evict pages from physical memory
  - If (# free pages > *HW*), the swap daemon goes to sleep

  - What if the allocation speed is faster than reclamation speed?

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*

File vs. Anonymous Pages

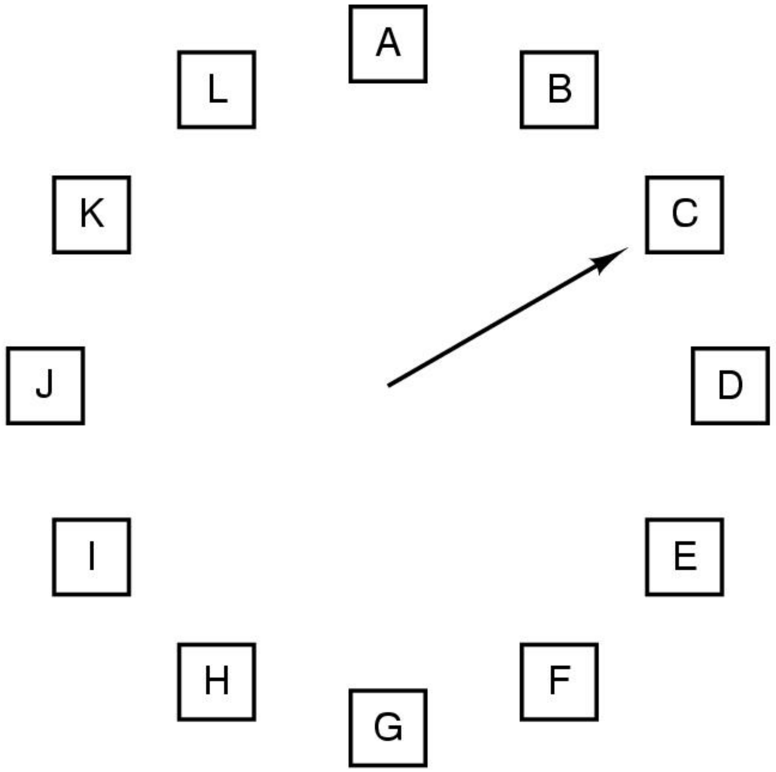
	File Pages	Anonymous Pages
Mapped to User Space	mmap()'ed pages	Stack, Heap pages mmap()'ed pages with MAP_ANONYMOUS COW'ed pages for mmap() with MAP_PRIVATE Shmem pages
Unmapped	Pages in page cache	Pages in swap cache tmpfs pages

What to Swap

- What happens to each type of page frame on low mem.
  - Kernel code → **Not swapped**
  - Kernel data → **Not swapped**
  - Page tables for user processes → **Not swapped**
  - Kernel stack for user processes → **Not swapped**
  - User code pages → **Dropped**
  - User data pages → **Dropped or swapped**
  - User heap/stack pages → **Swapped**
  - Files mmap'ed to user processes → **Dropped or go to file system**
  - Page cache pages → **Dropped or go to file system**
- Page replacement policy chooses the pages to evict

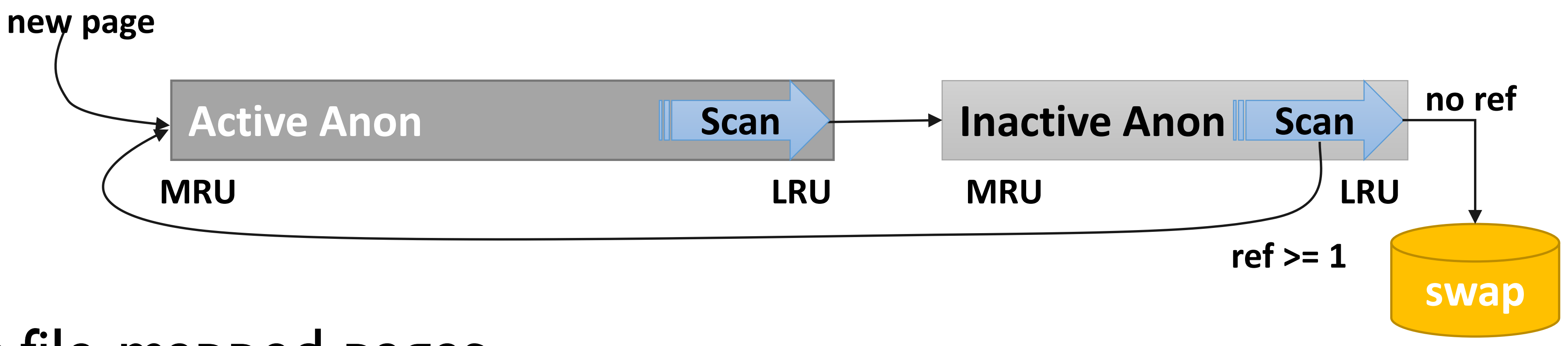
Page Replacement: Clock

- LRU is expensive, why?
- Uses R (Reference) bit in each PTE
- Arranges all of physical frames in a big circle
- A clock hand is used to select a victim
  - If (R == 1), turn in off and go to next page (second chance)
  - if (R == 0), evict the page
  - Arm moves quickly when pages are needed
- If memory is large, "accuracy" of information degrades



Linux Page Replacement (v5.x)

- For anonymous pages



- For file-mapped pages