Memory

Limitations of Swapping

- Problems with swapping
  - Process must fit into physical memory (impossible to run larger processes)
  - Memory becomes fragmented
    - External fragmentation
      - Lots of small free areas
    - Compaction
      - Reassemble larger free areas
  - Processes are either in memory or on disk: half and half doesn’t do any good

Virtual memory

- Basic idea
  - Allow the OS to hand out more memory than exists on the system
  - Keep recently used stuff in physical memory
  - Move less recently used stuff to disk
  - Keep all of this hidden from processes
- Process view
  - Processes still see an address space from 0 – max address
  - Movement of information to and from disk handled by the OS without process help

Benefits of Virtual Memory

- Especially helpful in multiprogrammed system
  - CPU schedules process B while process A waits for its memory to be retrieved from disk
- Use secondary storage($)
  - Extend DRAM($$$) with reasonable performance
- Protection
  - Programs do not step over each other

Benefits of Virtual Memory

- Convenience
  - Flat address space
  - Programs have the same view of the world
  - Load and store cached virtual memory without user program intervention
- Reduce fragmentation
  - Make cacheable units all the same size (page)

Paging

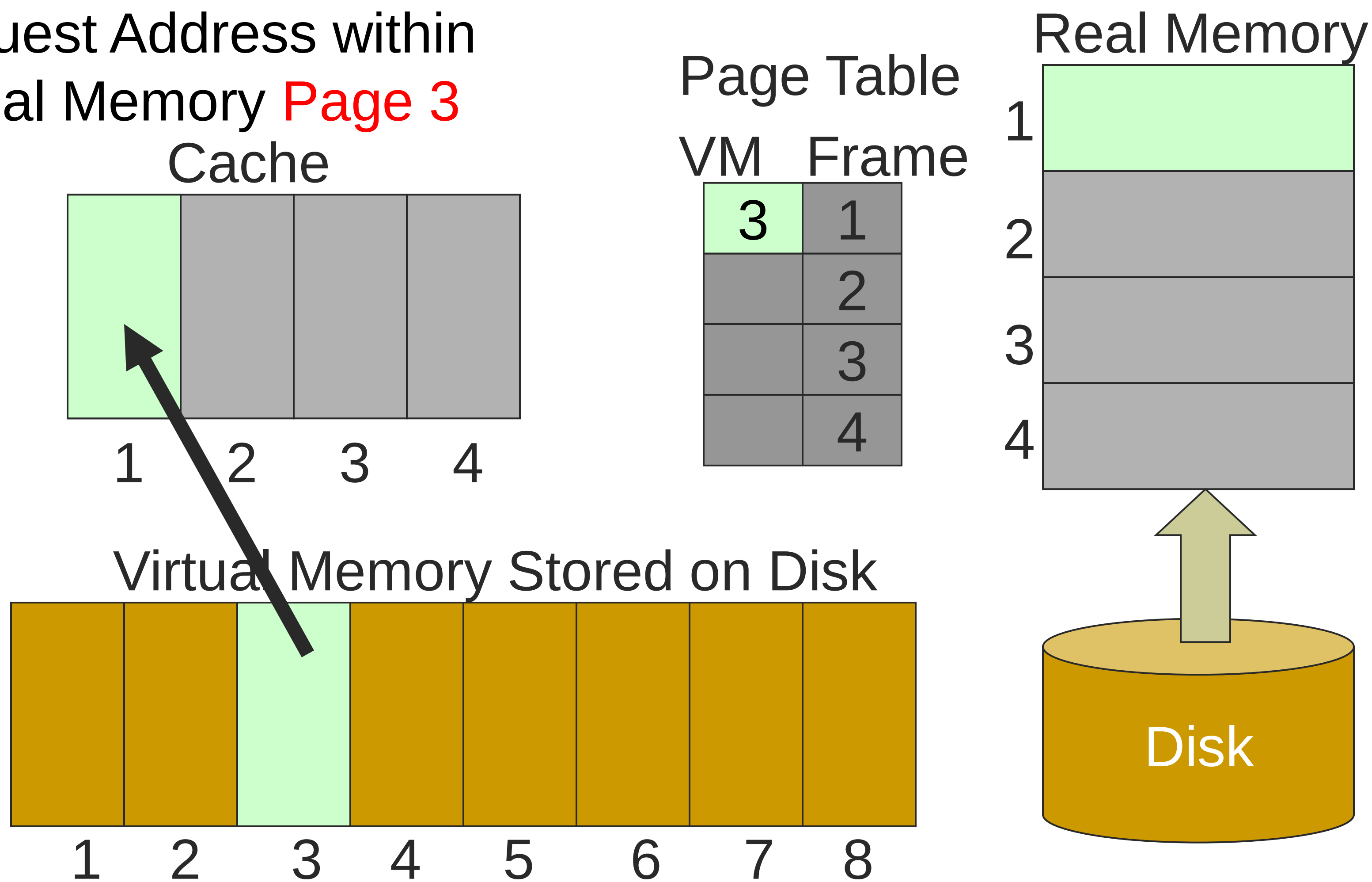
- Paging is how an OS achieves VM
- Goal
  - Provide user with virtual memory that is as big as user needs
- Implementation
  - Store virtual memory on disk
  - Cache parts of virtual memory being used in real memory
  - Load and store cached virtual memory without user program intervention

Page Faults

- What happens when a program accesses a virtual page that is not mapped into any physical page?
  - Hardware triggers a page fault
- Page fault handler
  - Find any available free physical page
  - If none, evict some resident page to disk
  - Allocate a free physical page
  - Load the faulted virtual page to the prepared physical page
  - Modify the page table

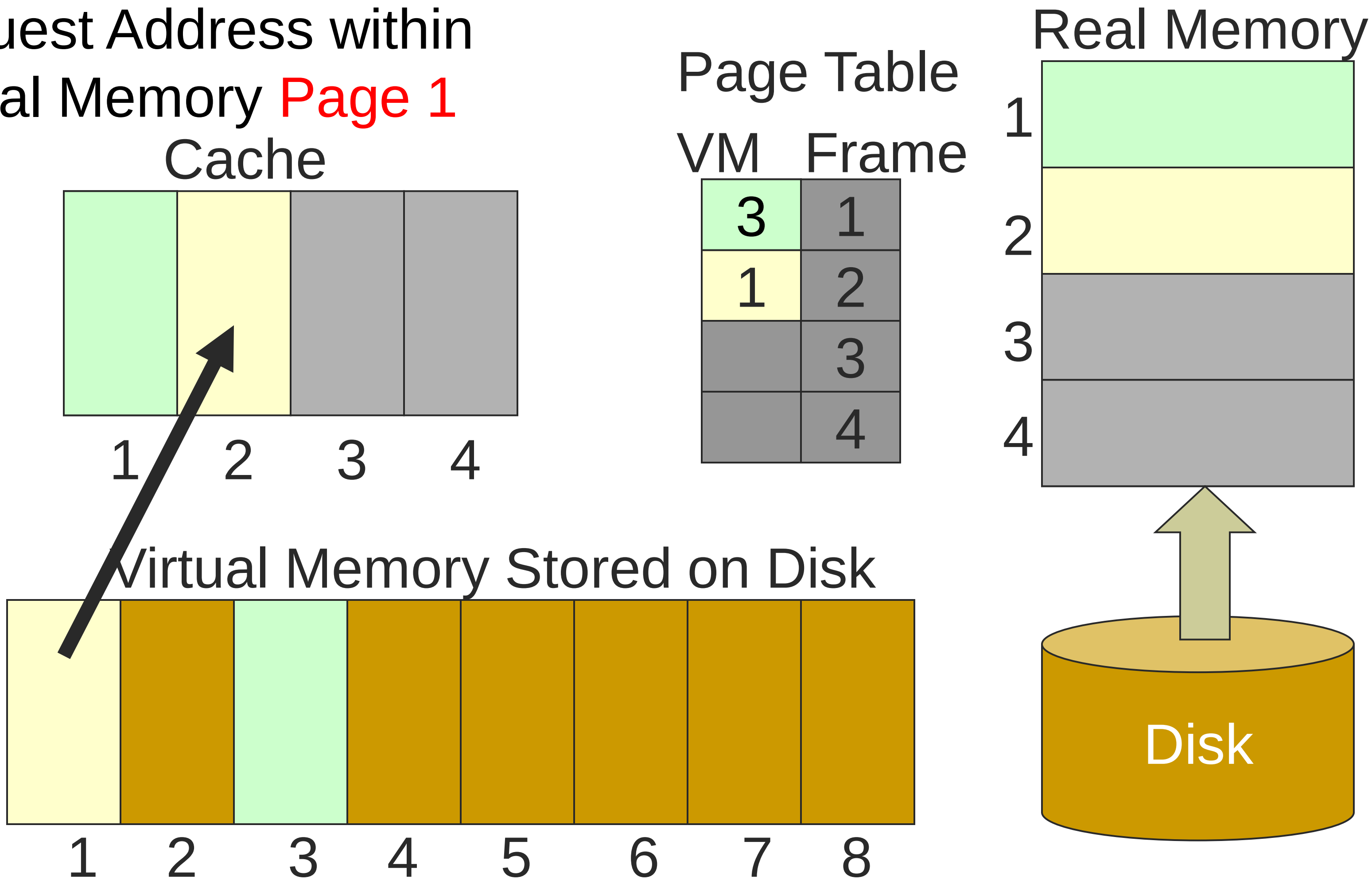
Paging Request

Request Address within Virtual Memory Page 3



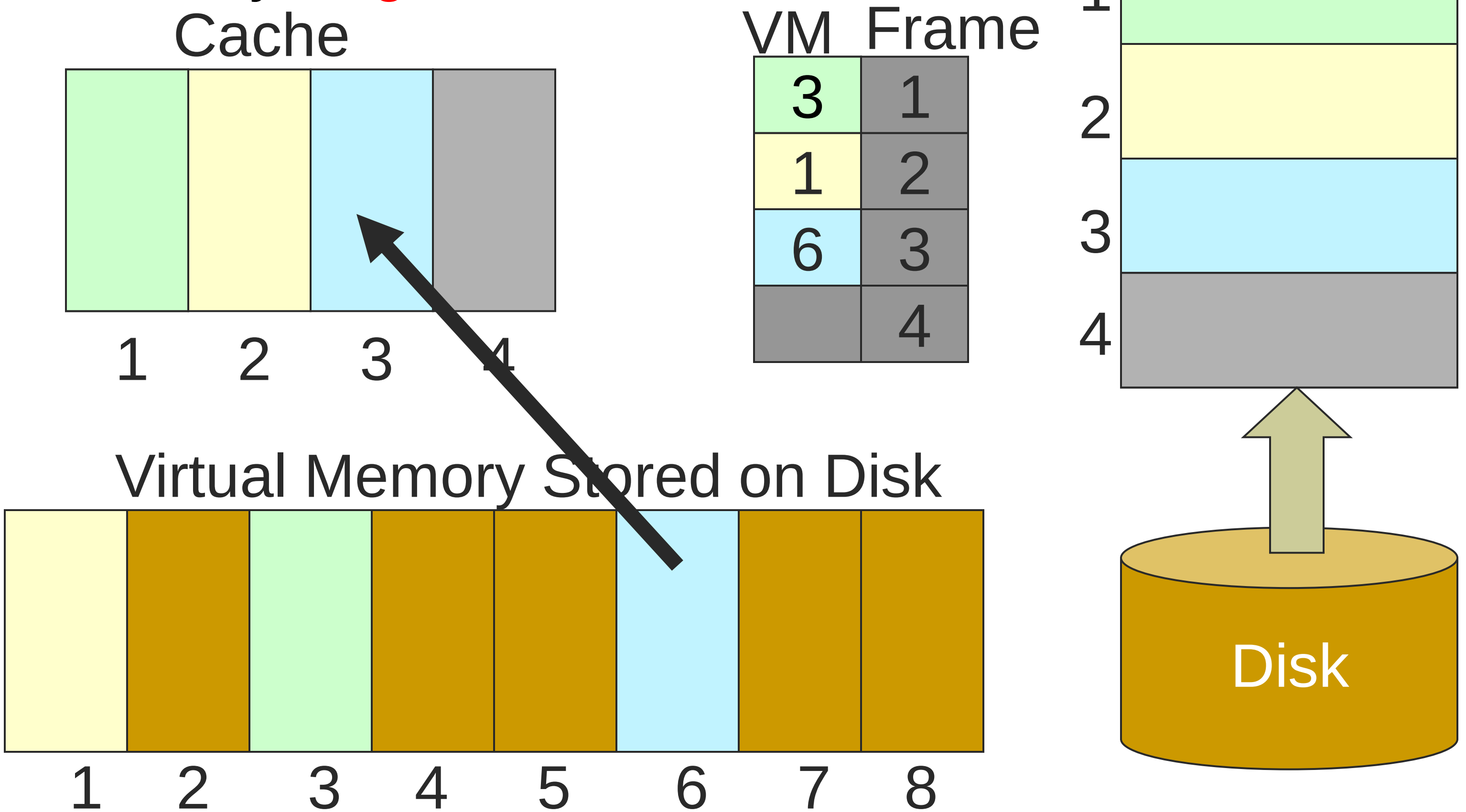
Paging Request

Request Address within Virtual Memory Page 1



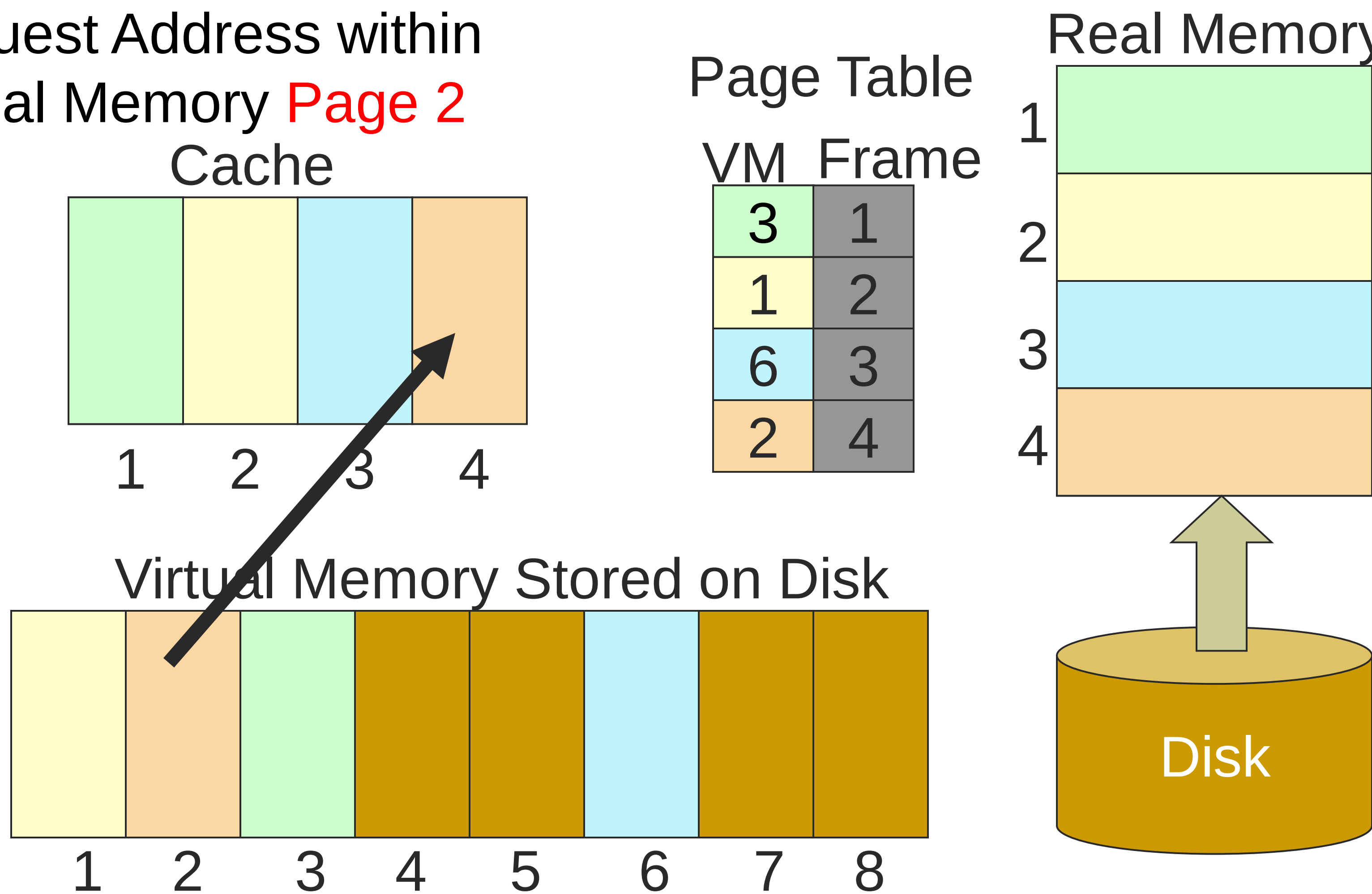
Paging Request

Request Address within Virtual Memory Page 6



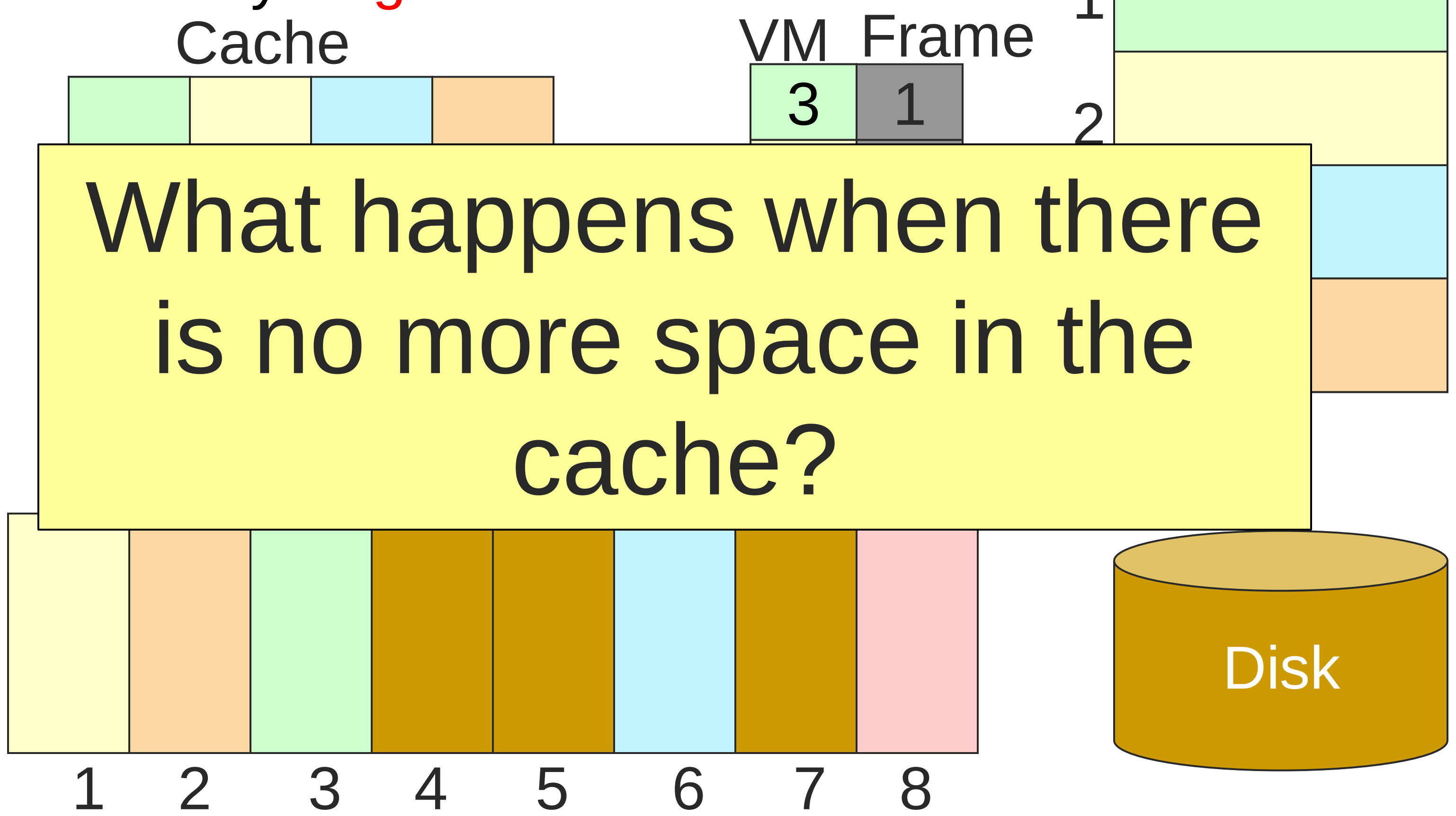
Paging Request

Request Address within Virtual Memory Page 2



Paging Request

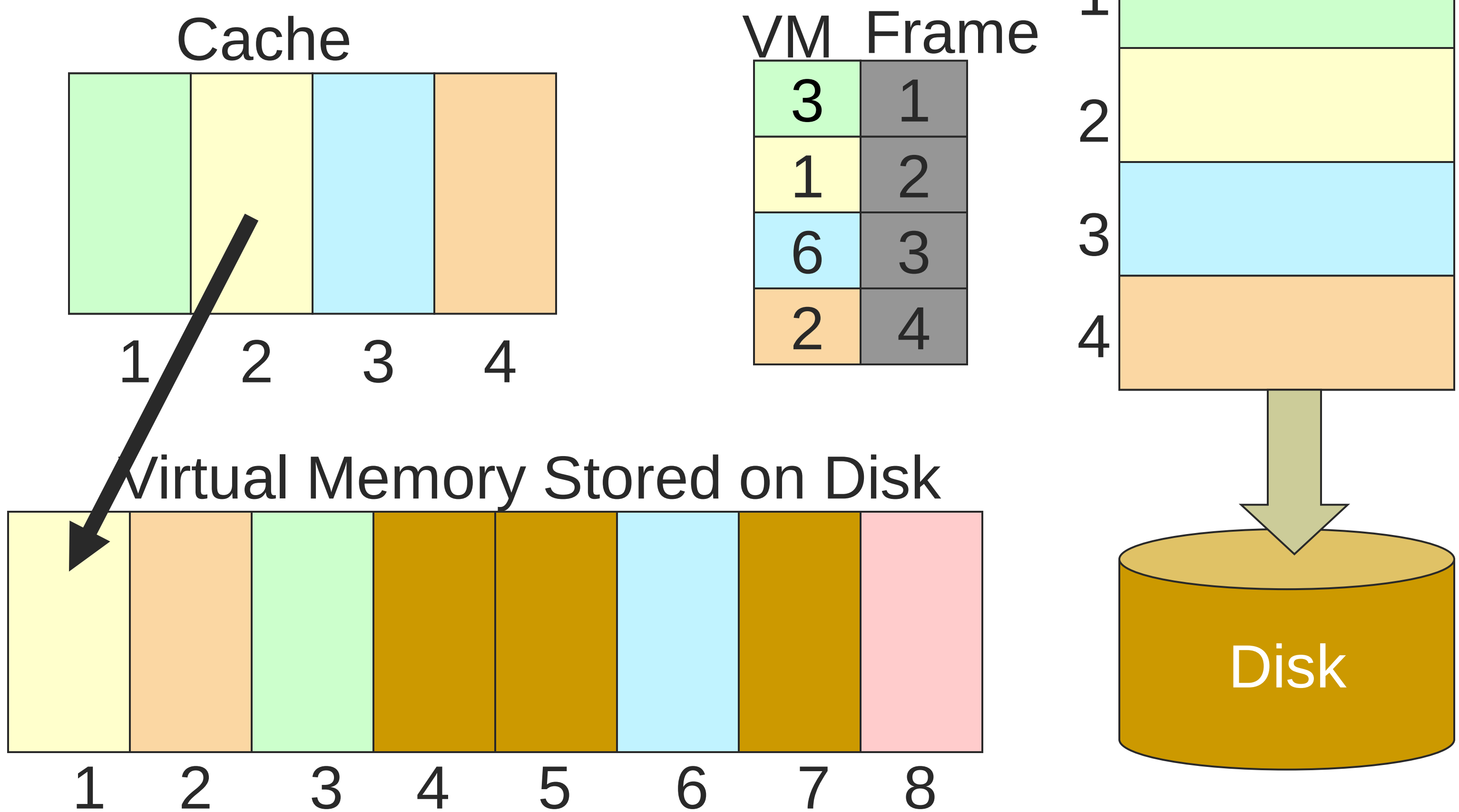
Request Address within Virtual Memory Page 8



What happens when there is no more space in the cache?

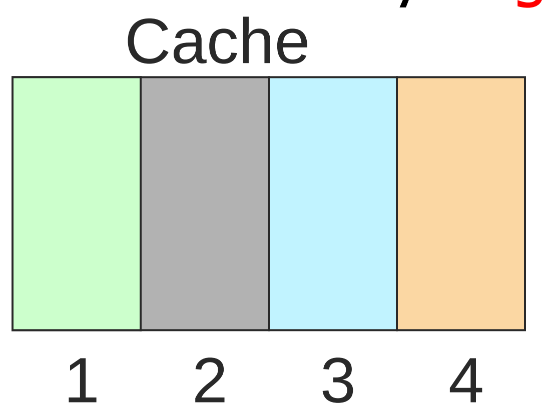
Paging Request

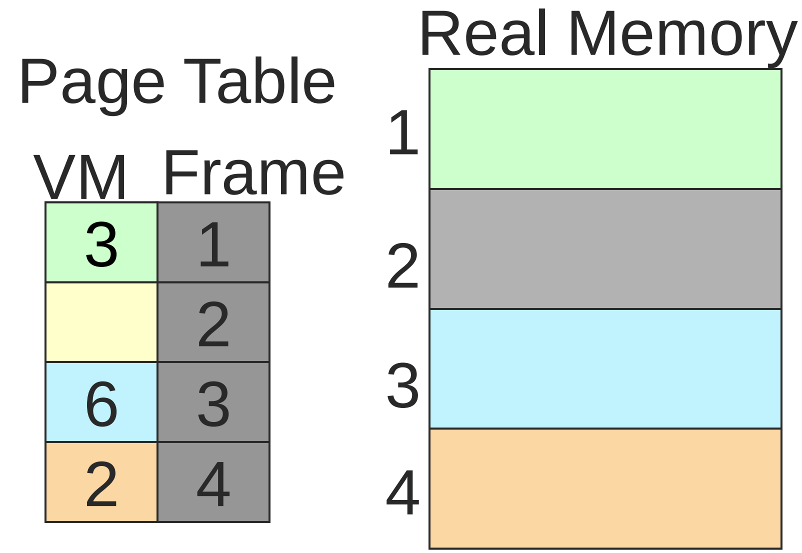
Store Virtual Memory
Page 1 to disk

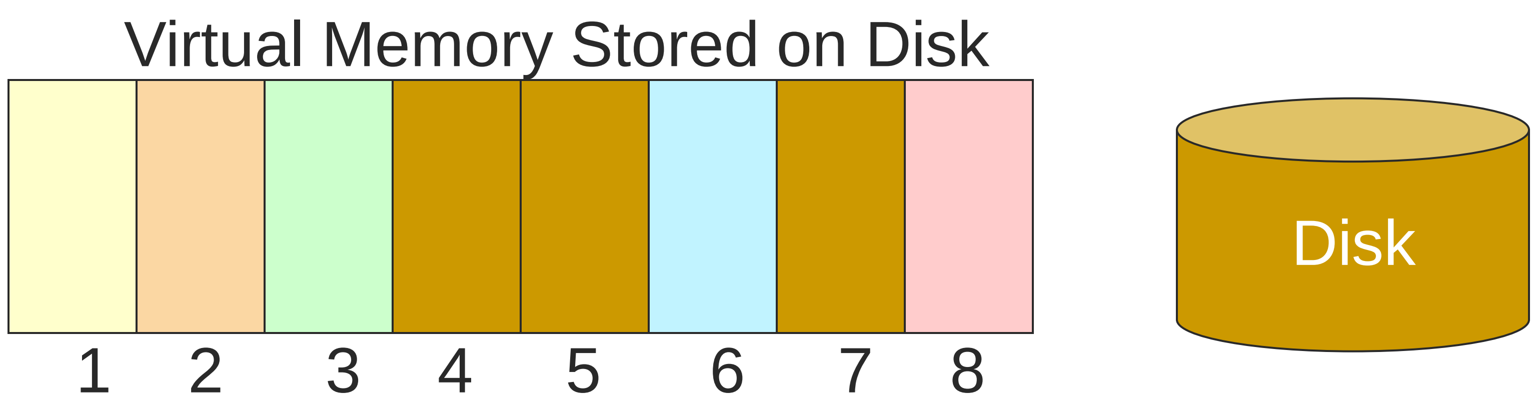


Paging Request

Process request for Address within Virtual Memory Page 8

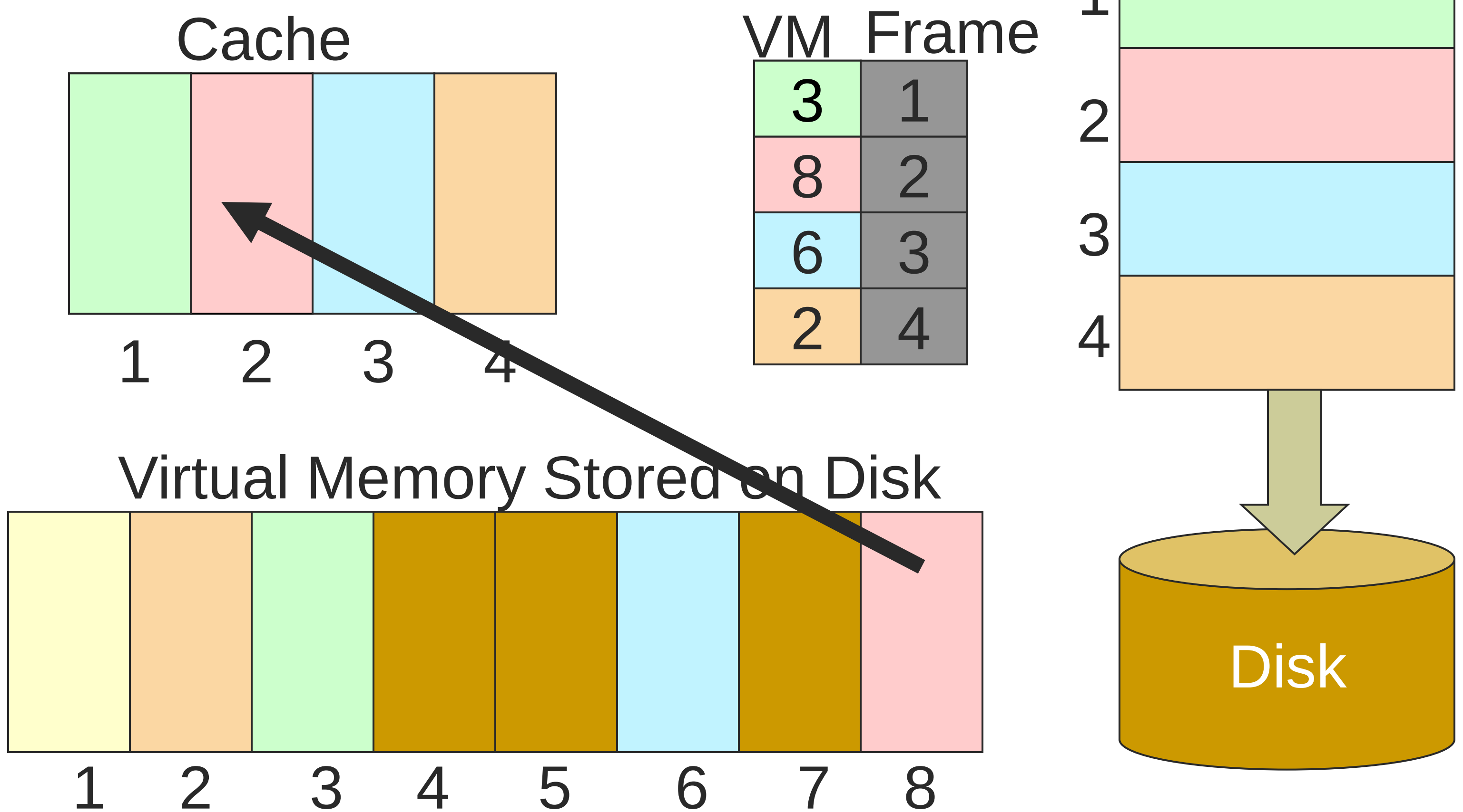






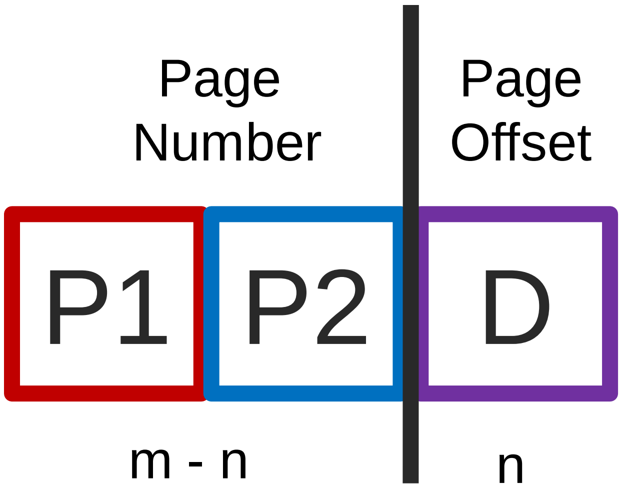
Paging Request

Load Virtual Memory
Page 8 to cache



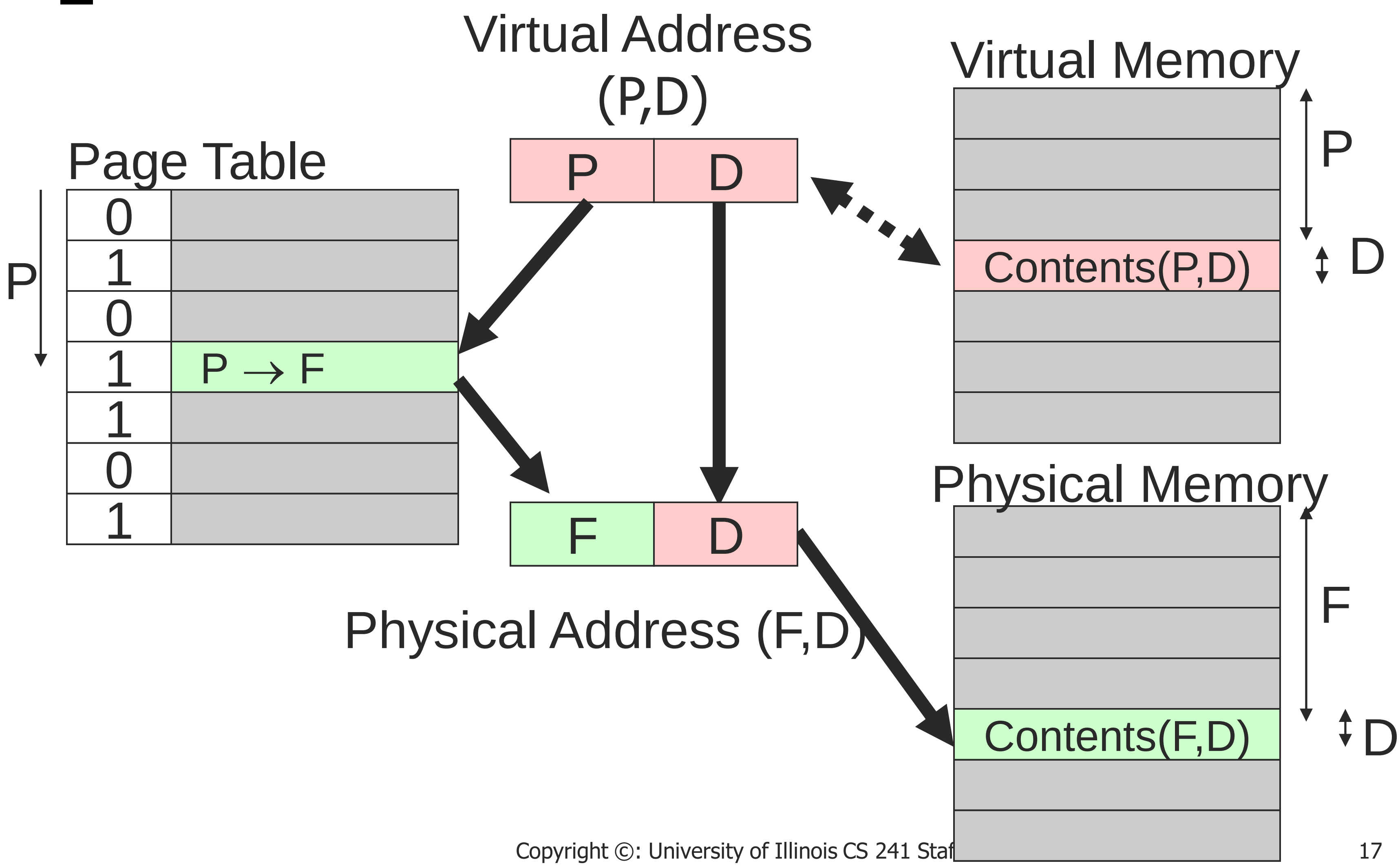
Address Translation Scheme

- Address generated by CPU is divided into
  - Page number (p)
    - An index into a page table
    - Contains base address of each page in physical memory
  - Page offset (d)
    - Combined with base address
    - Defines the physical memory address that is sent to the memory unit

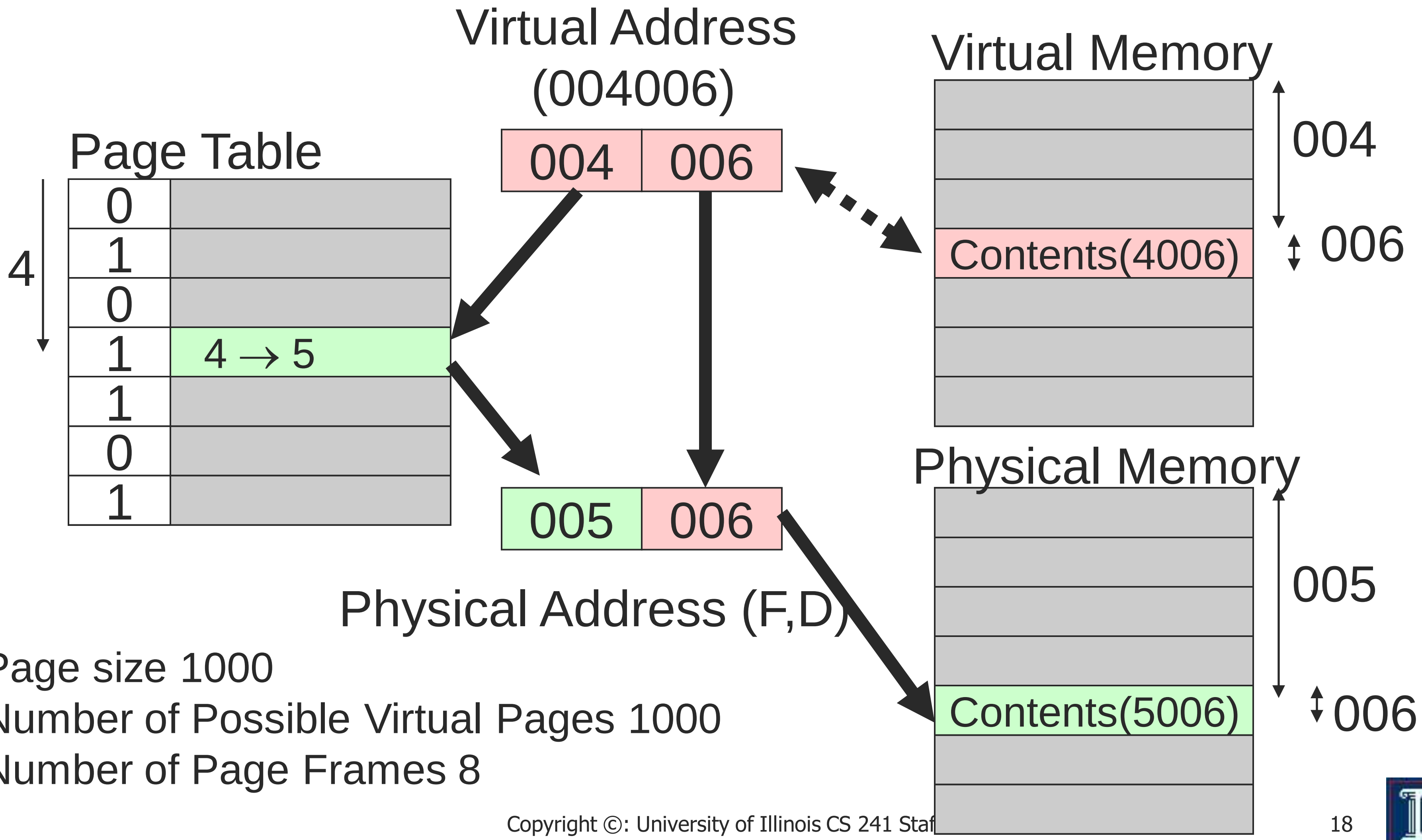


For given logical address space 2m and page size 2n

Page Mapping Hardware



Page Mapping Hardware



Page size 1000
Number of Possible Virtual Pages 1000
Number of Page Frames 8

Paging Issues

- Page size
  - Typically 2^n
    - usually 512, 1k, 2k, 4k, or 8k
  - Example
    - 32 bit VM address may have 2^{20} (1 meg) pages with 4k (2^{12}) bytes per page
    - 2^{20} (1 meg) 32 bit page entries take 2^{22} bytes (4 meg)
  - Page frames must map into real memory

Paging Issues

- Physical memory size: 32 MB (2^{25})
  - Page size 4K bytes
  - How many pages?
    - 2^{13}
- NO external fragmentation
- Internal fragmentation on last page ONLY

Discussion

- How can paging be made faster?
  - Mapping must be done for every reference
  - More memory = more pages!
  - Hardware registers (one per page)
  - Keep page table in memory
- Is one level of paging sufficient?
- Sharing and protections?

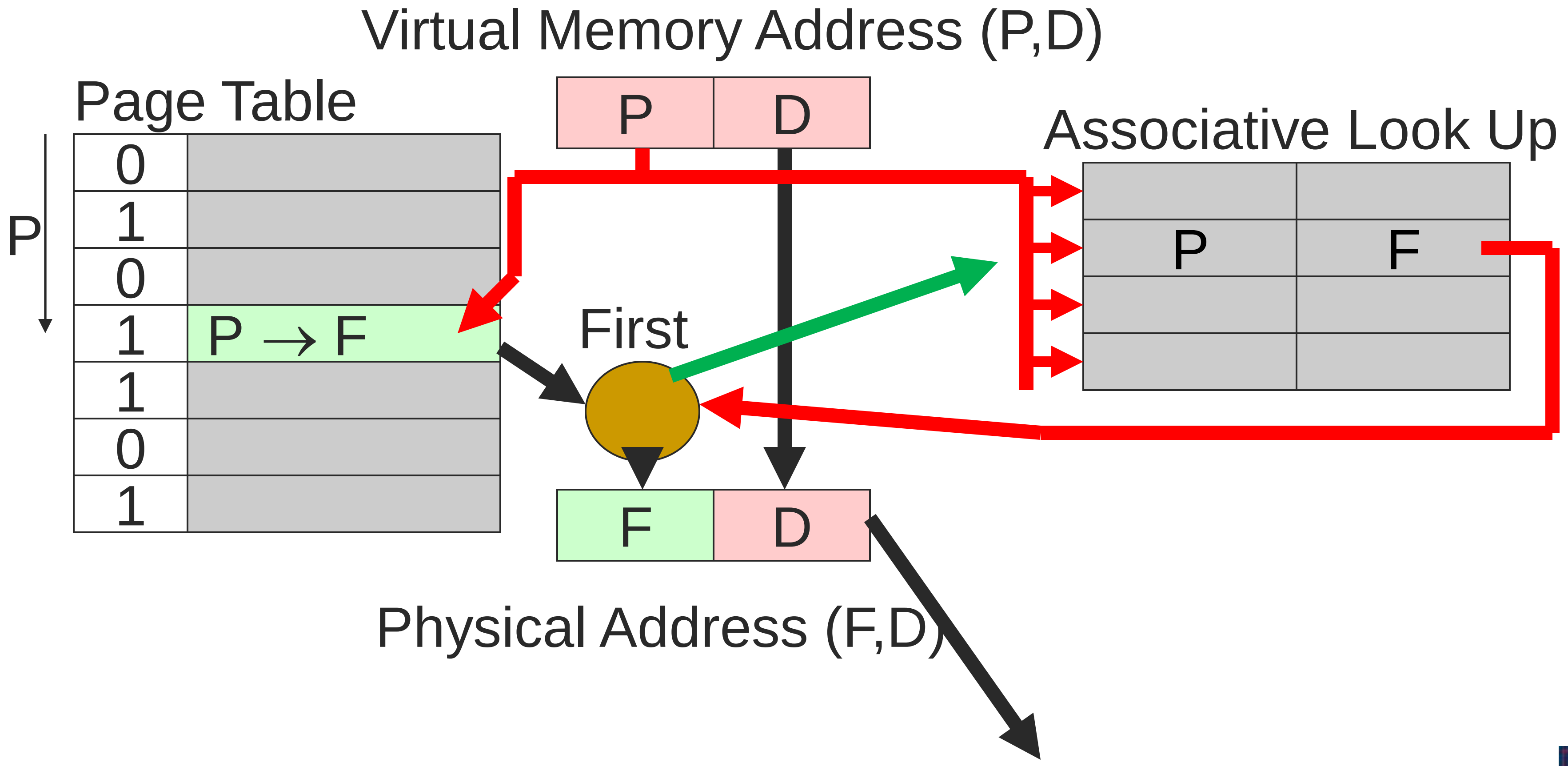
Paging - Caching the Page Table

- Cache page table in registers
- Keep page table in memory
  - Location given by a *page table base register*
- Page table base register changed at context switch time

Paging Implementation Issues

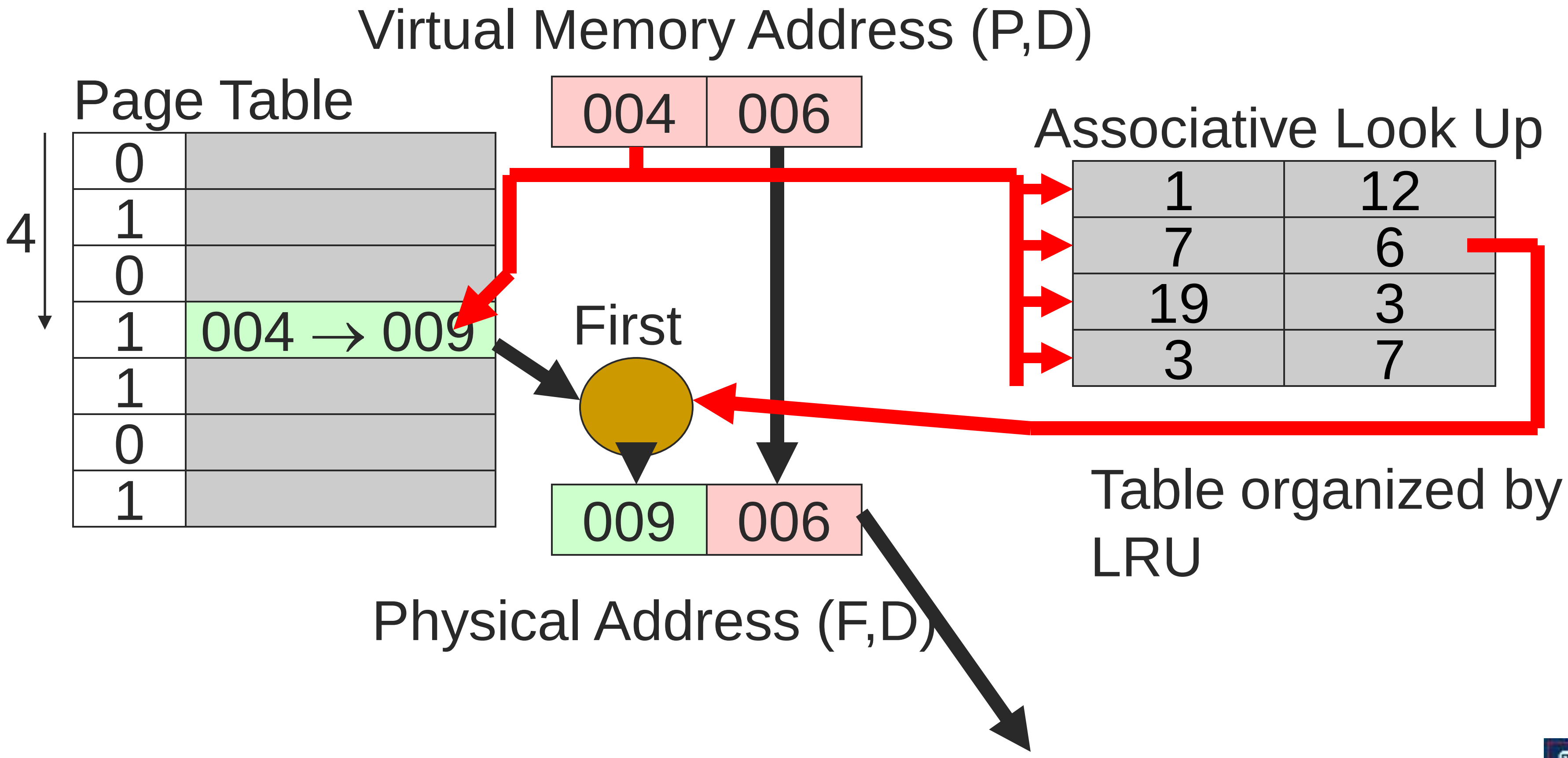
- Caching scheme
  - Associative registers, look-aside memory or content-addressable memory
  - Translation-lookaside-buffer (TLB)
- Page address cache (TLB) hit ratio
  - Percentage of time page found in associative memory
- Cache miss
  - If not found in associative memory, must load from page tables
  - Requires additional memory reference

Page Mapping Hardware



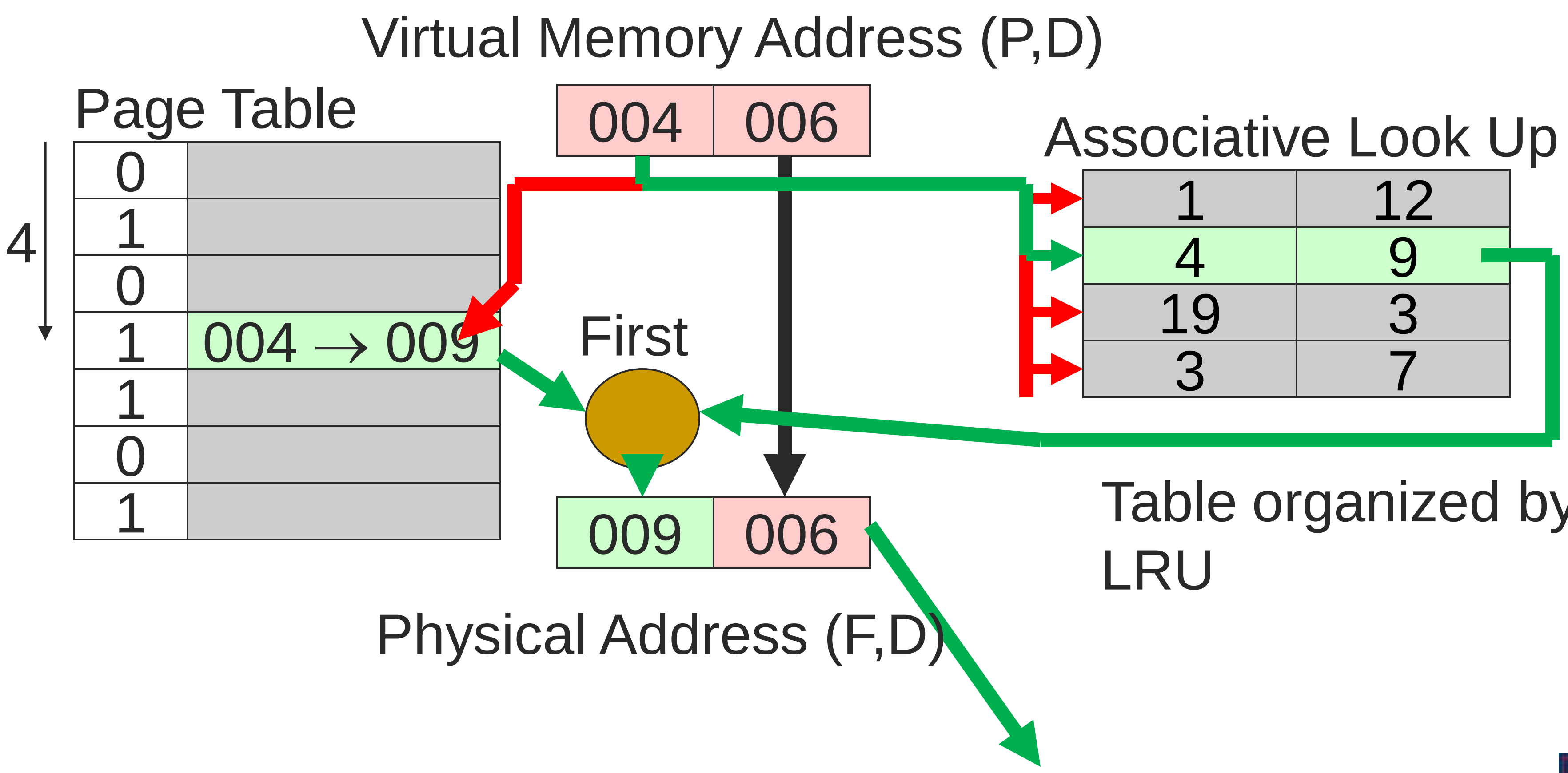
Page Mapping Hardware

First access, retrieve page from page table



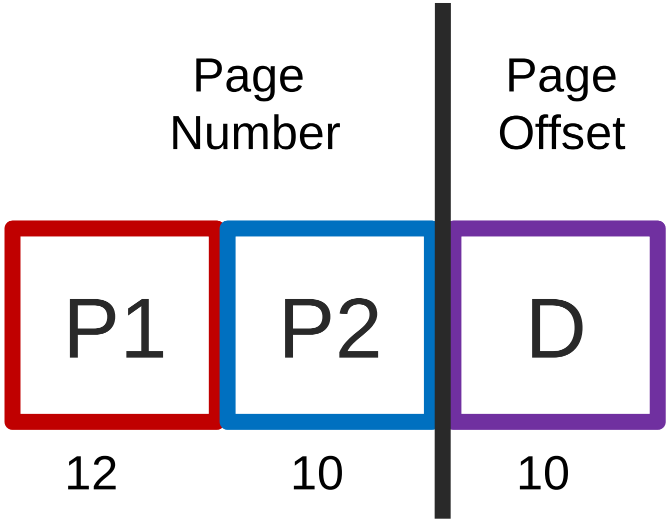
Page Mapping Hardware

Second access, retrieve page from associative registers.

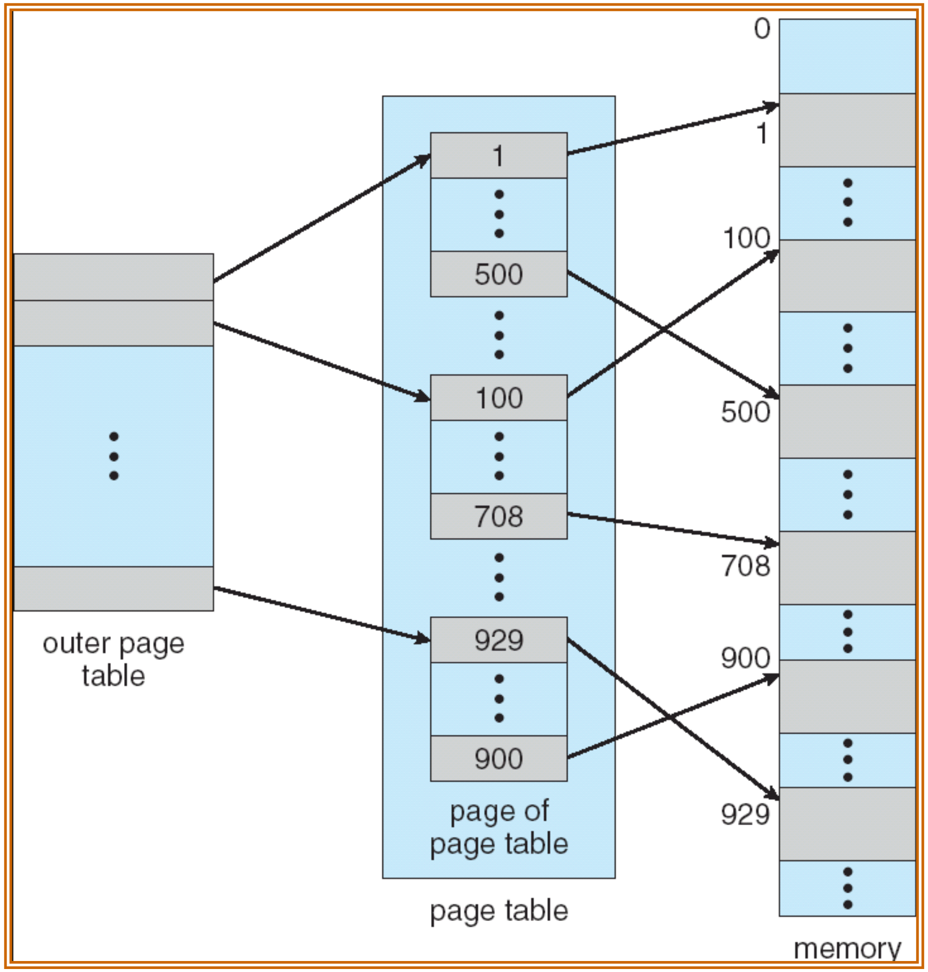


Addressing on Two-Level Page Table

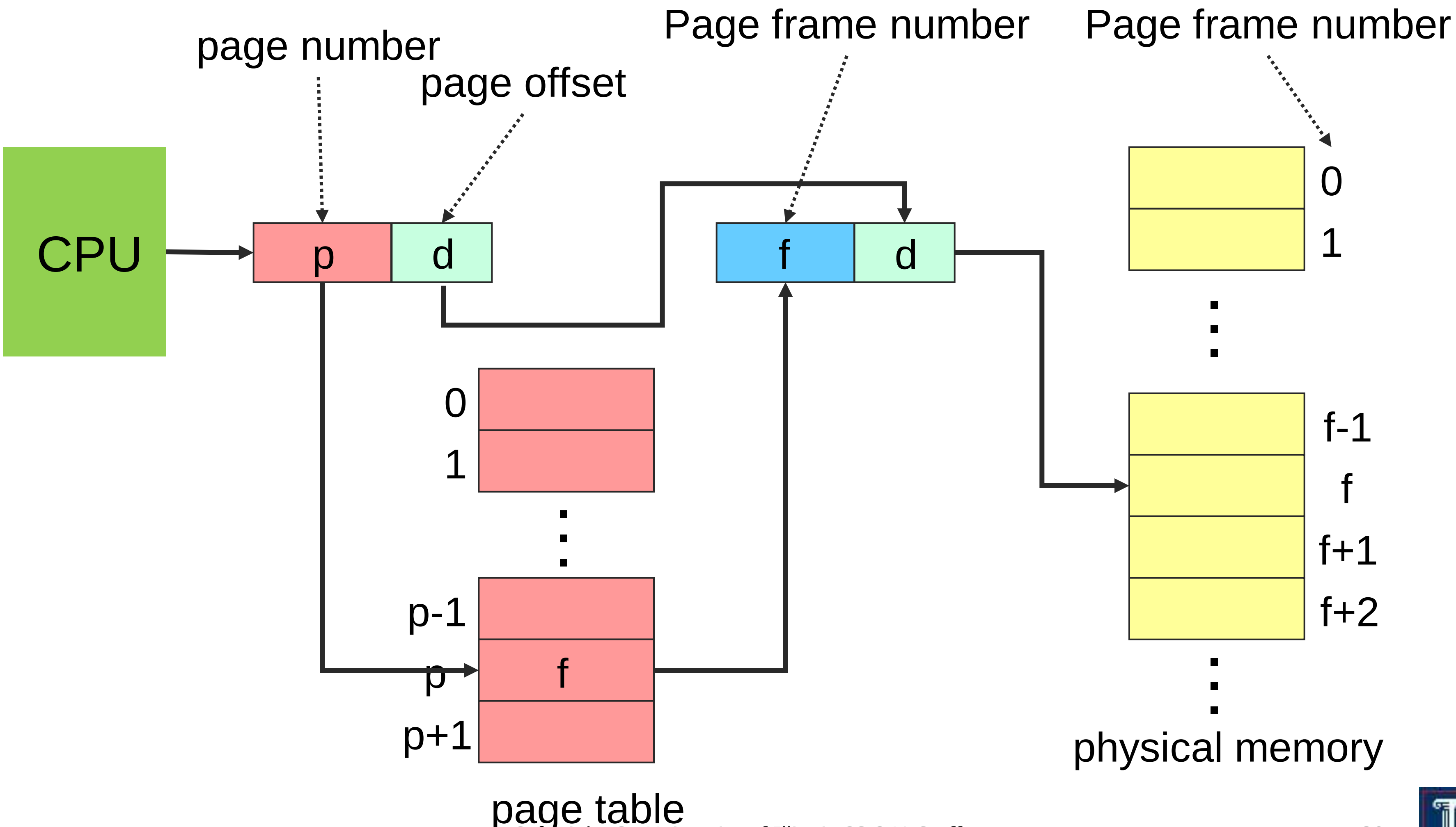
- 32-bit Architecture
  - 4096= 2^{12} B Page
- 4K Page of Logical Memory
  - 4096 addressable bytes
- Page the Page Table
  - 4K pages as well
  - 1024 addressable 4byte addresses



Two-Level Page-Table



Addressing on Two-Level Page Table



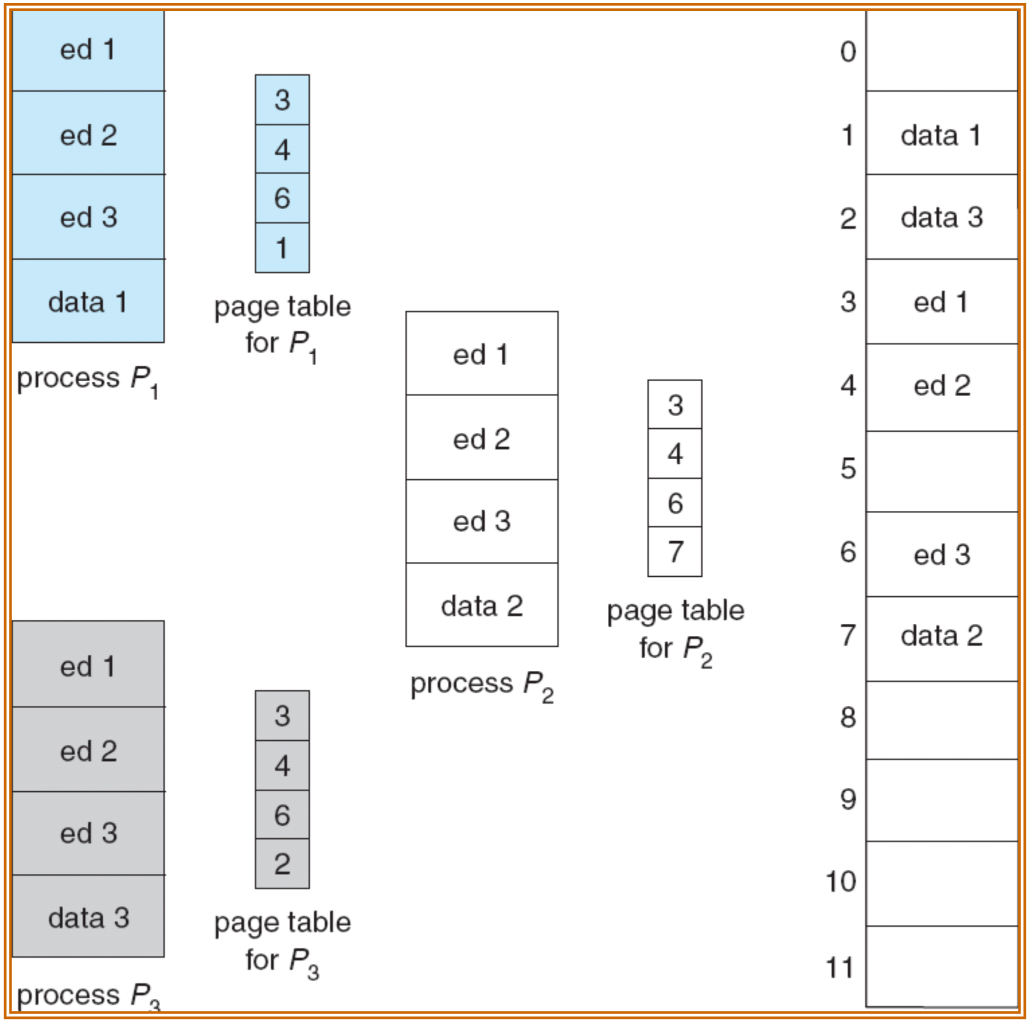
Newer Architectures

- 64-bit Architecture
  - Address space: 2^{64} B
  - Page size: 4096 B
  - Page table size: 2^{52}
  - For 8B entries, need 30 Million GB!
- Approach
  - Have enough entries to match the number of page frames
  - Smaller page table

Sharing Pages

- Shared code
  - One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems).
  - Shared code must appear in same location in the logical address space of all processes
- Private code and data
  - Each process keeps a separate copy of the code and data
  - The pages for the private code and data can appear anywhere in the logical address space

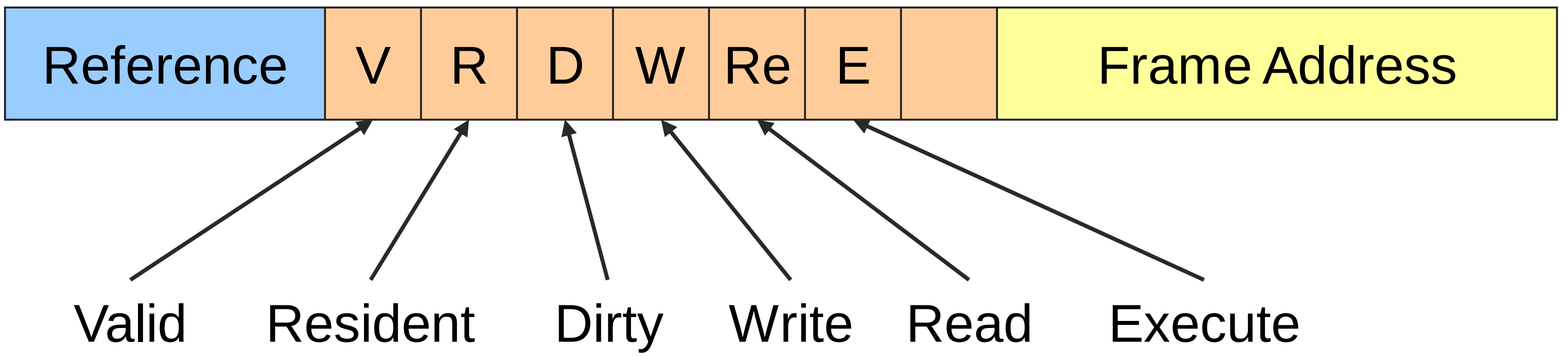
Shared Pages



Page Protection

- Can add read, write, execute protection bits to page table to protect memory
  - Check is done by hardware during access
  - Can give shared memory location different protections from different processes by having different page table protection access bits
- Valid-invalid bit attached to each entry in the page table
  - “valid” indicates that the associated page is in the process’ logical address space
  - “invalid” indicates that the page is not in the process’ logical address space

Page Protection



- Reference — page has been accessed
- Valid — page exists
- Resident — page is cached in primary memory
- Dirty — page has changed since page in

Demand Paging

- Never bring a page into primary memory until its needed
- Fetch Strategies
  - When should a page be brought into primary (main) memory from secondary (disk) storage.
- Placement Strategies
  - When a page is brought into primary storage, where should it be put?
- Replacement Strategies
  - Which page now in primary storage should be removed from primary storage when some other page or segment needs to be brought in and there is not enough room

Issue: Eviction

- Hopefully, kick out a less-useful page
  - Dirty pages require writing, clean pages don't
  - Where do you write? To "swap space"
- Goal: kick out the page that's least useful
- Problem: how do you determine utility?
  - Heuristic: temporal locality exists
  - Kick out pages that aren't likely to be used again

Principal of Optimality

- Definition
  - Each page is labeled with the number of instructions that will be executed before that page is first referenced
  - The optimal page replacement algorithm: choose the page with the highest label to be removed from the memory.
- Impractical: requires knowledge of future references
- If future references are known
  - should use pre paging to allow paging to be overlapped with computation.

Page Replacement Strategies

- Random page replacement
  - Choose a page randomly
- FIFO - First in First Out
  - Replace the page that has been in primary memory the longest
- LRU - Least Recently Used
  - Replace the page that has not been used for the longest time
- LFU - Least Frequently Used
  - Replace the page that is used least often
- NRU - Not Recently Used
  - An approximation to LRU.
- Working Set
  - Keep in memory those pages that the process is actively using.