CSE120
Principles of Operating Systems

Prof Yuanyuan (YY) Zhou

Paging

Review

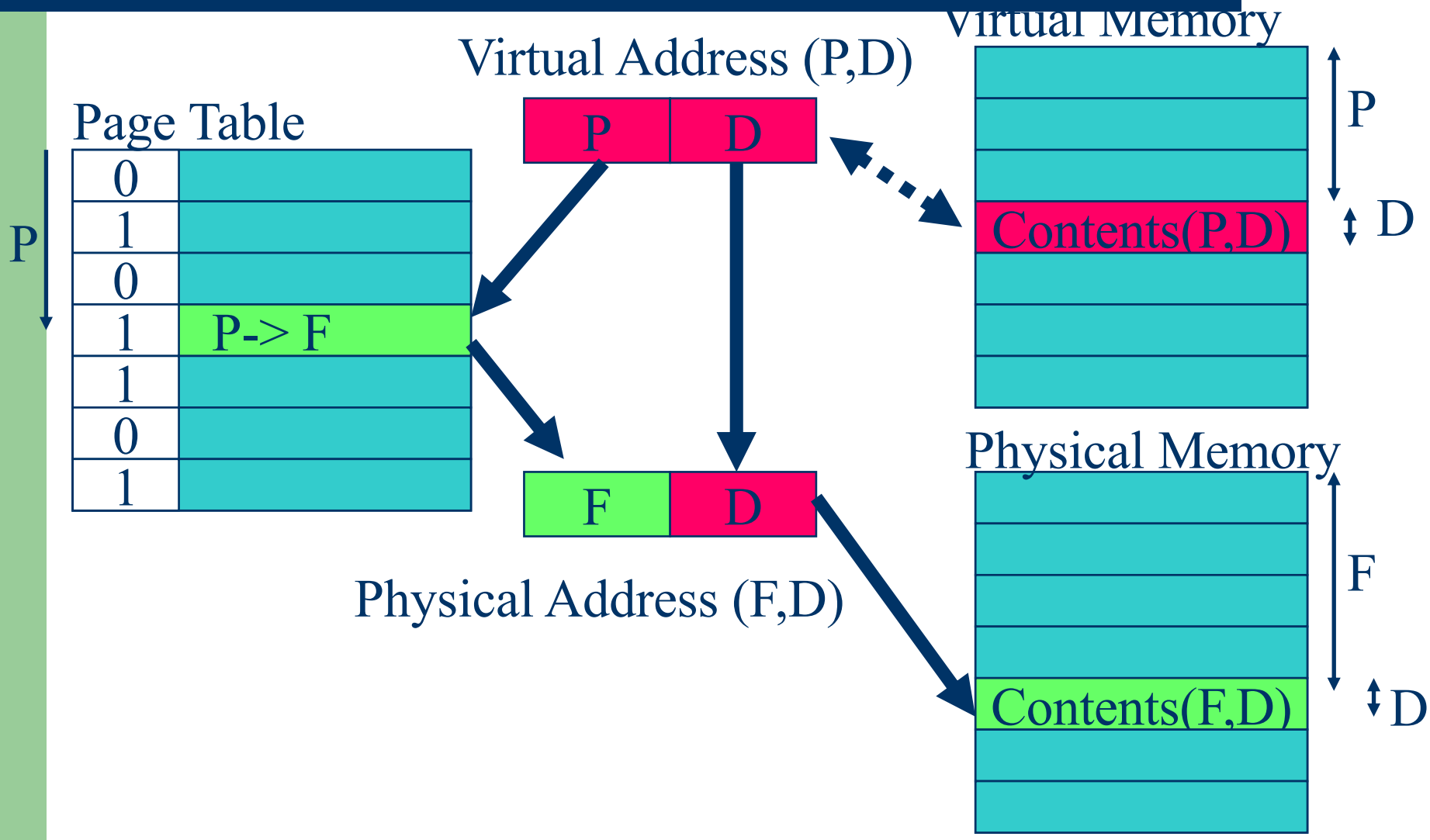
- Fixed and variable size partition
  - Simple to implement (base and limit registers)
  - Internal vs. external fragmentation
  - Best, worst, fast fit
  - Bitmap vs. link-list implementation of free memory regions
- Paging
  - Page size is a power of 2
  - Virtual page number → physical page frame
  - How to calculate VPN from virtual address?
  - How to get physical address?
- Segmentation
  - Segmentation table
- Hybrid: paging+segmentation

Today's Lecture

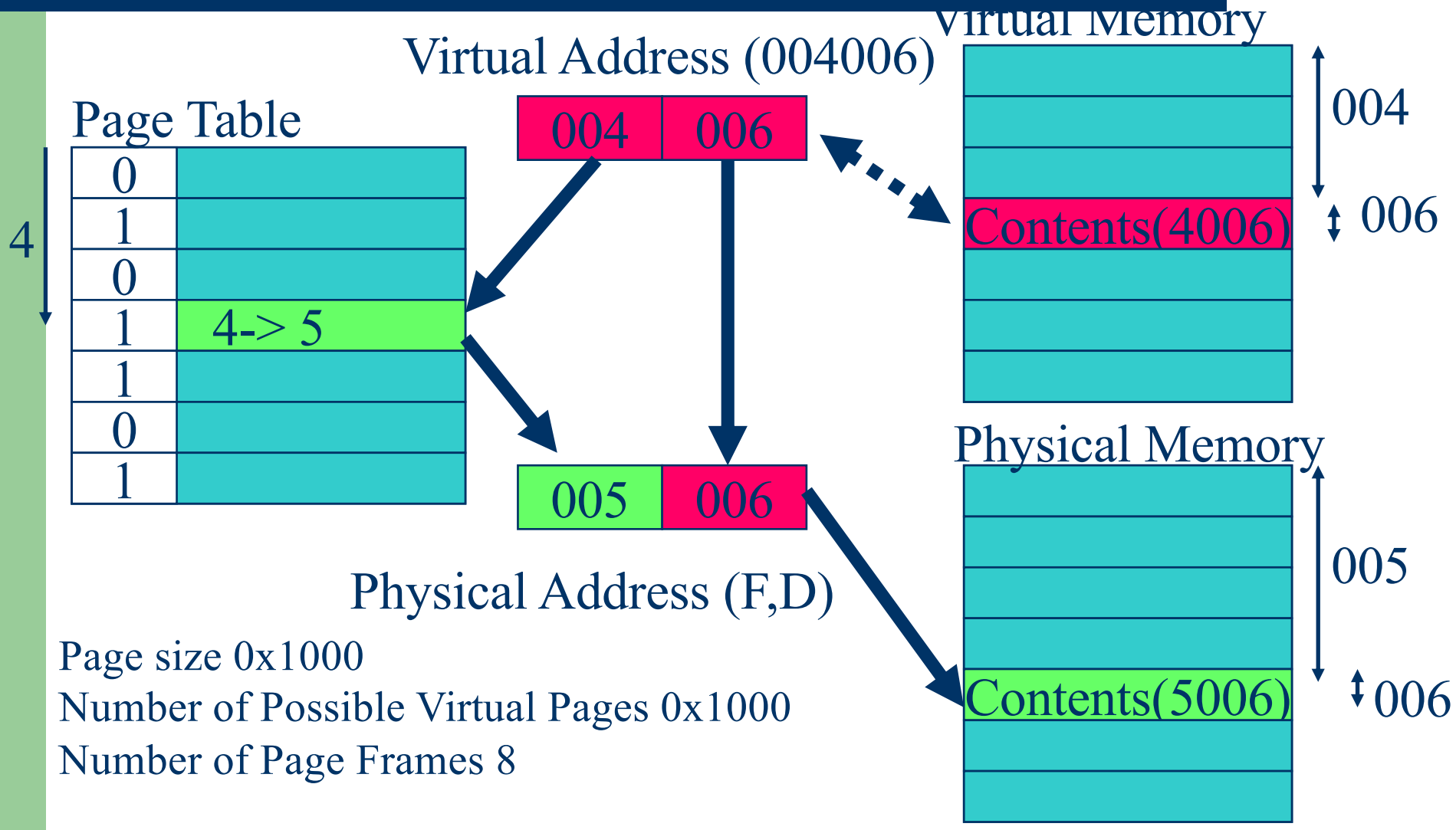
Today we'll cover more paging mechanisms:

- Optimizations
  - Managing page tables (space)
  - Efficient translations (TLBs) (time)
  - Demand paged virtual memory (space)
- Recap address translation

Page Mapping Hardware



Page Mapping Hardware



Paging Issues

- Page size is 2^n
  - usually 512, 1k, 2k, 4k, or 8k
  - E.g. 32 bit VM address may have 2^{20} (1MB) pages with 4k (2^{12}) bytes per page
- Page table:
  - 2^{20} page entries take 2^{22} bytes (4MB)

Managing Page Tables

- The page table for a 32-bit address space w/ 4K pages to be 4MB
  - This is far too much overhead for each process
- How can we reduce this overhead?
  - Observation: Only need to map the portion of the address space actually being used (tiny fraction of entire addr space)

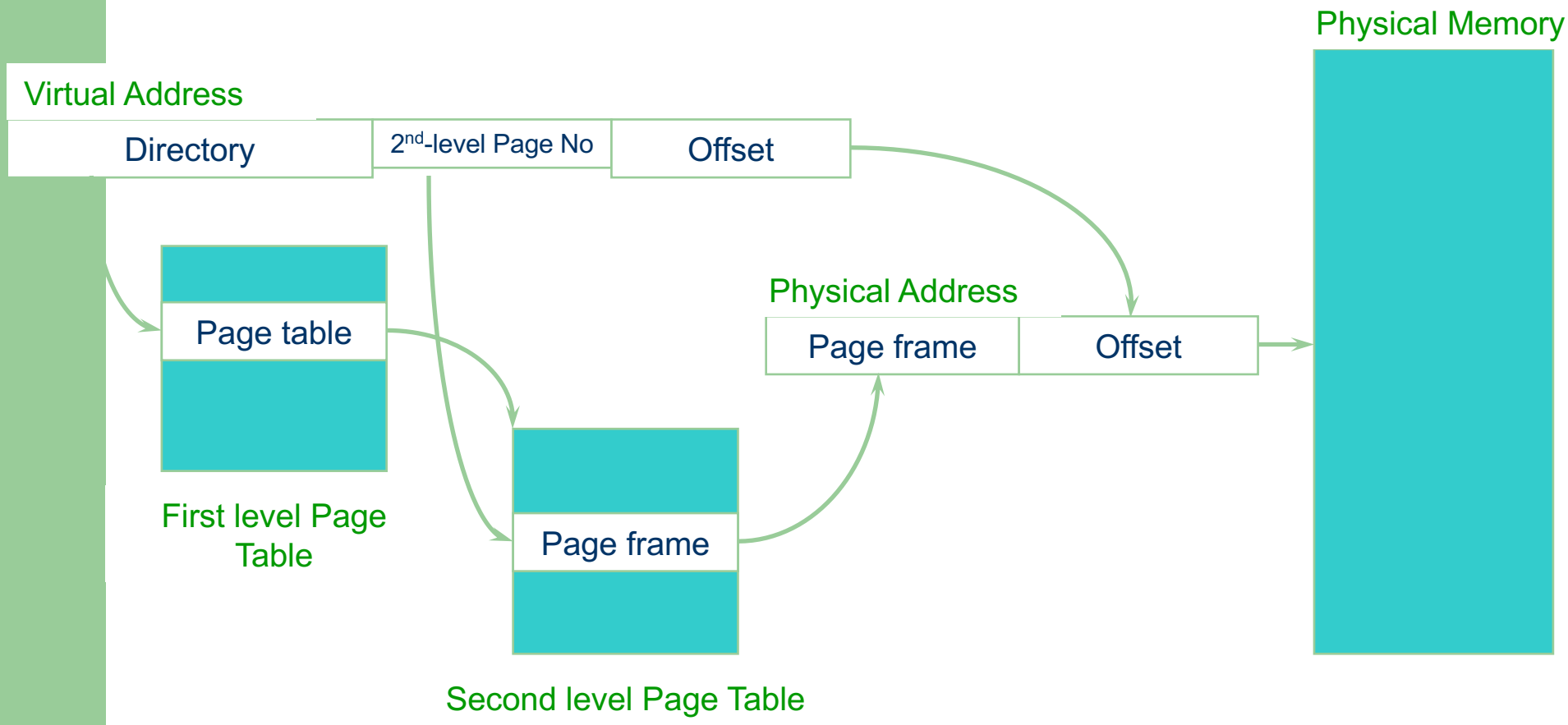
Question?

How to reduce page table size if the virtual pages are sparse?

Two-Level Page Tables

- Two-level page tables
  - Virtual addresses (VAs) have three parts:
    - Directory, secondary page number, and offset
  - Directory maps VAs to secondary page table
  - Secondary page table maps page number to physical page
  - Offset indicates where in physical page address is located
- Example
  - 4K pages, 4 bytes/PTE
  - How many bits in offset? 4K = 12 bits
  - Want master page table in one page: 4K/4 bytes = 1K entries
  - Hence, 1024 secondary page tables. How many bits?
  - Master (1K) = 10, offset = 12, secondary = 32 – 10 – 12 = 10 bits

Two-Level Page Tables



Youtube Video

- https://www.youtube.com/watch?v=Z4kSOv49GNc

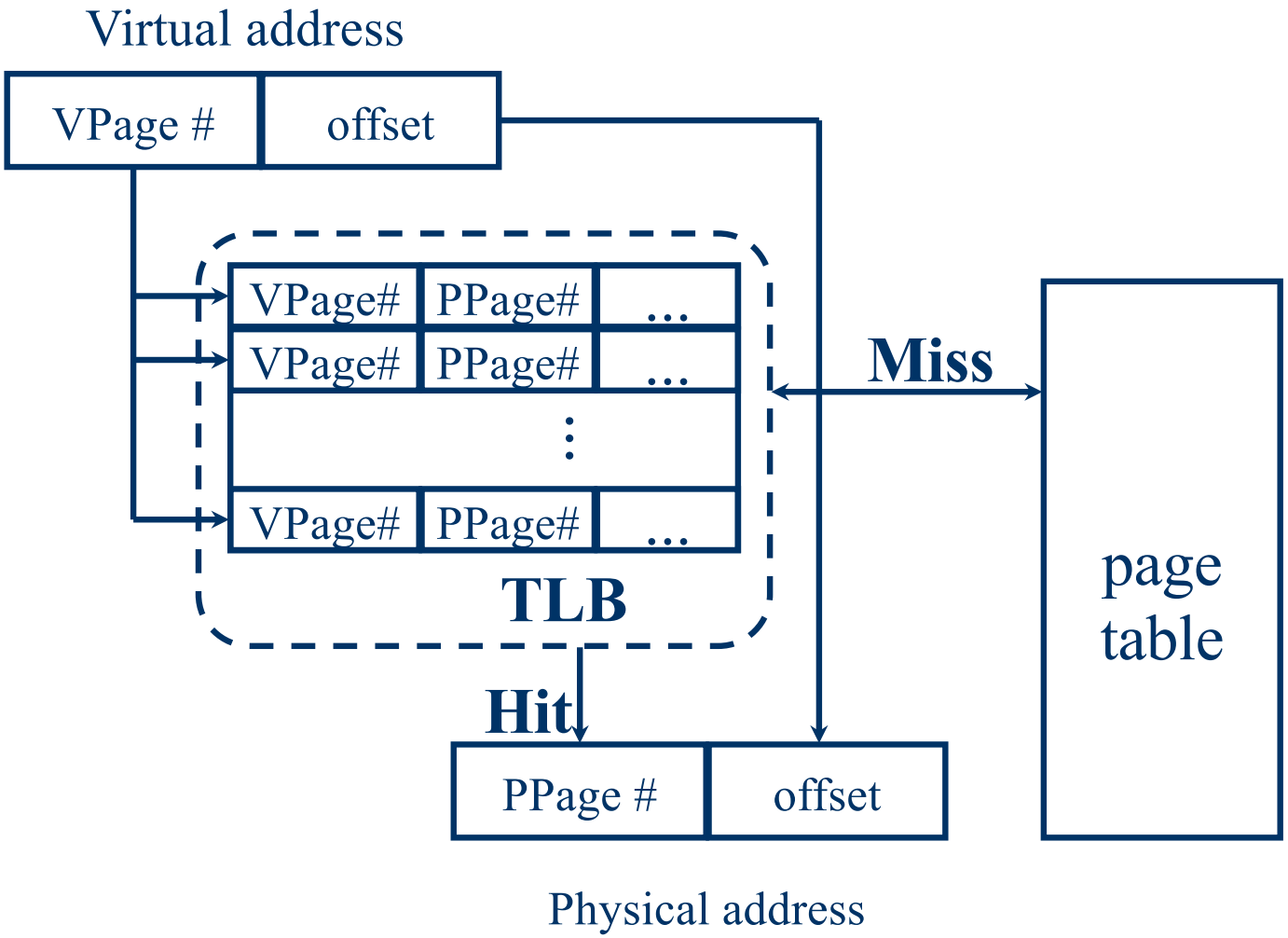
So what is the problem with two-level page table?

- Hints:
  - Programs only know virtual addresses
  - Each virtual address must be translated
    - So, each program memory access requires several actual memory accesses

Efficient Translations

- Our original page table scheme already doubled the cost of doing memory lookups
  - One lookup into the page table, another to fetch the data
- Now two-level page tables triple the cost!
  - Two lookups into the page tables, a third to fetch the data
  - And this assumes the page table is in memory
- How can we use paging but also have lookups cost about the same as fetching from memory?
  - Cache translations in hardware
  - Translation Lookaside Buffer (TLB)
  - TLB managed by Memory Management Unit (MMU)

Translation Look-aside Buffer (TLB)



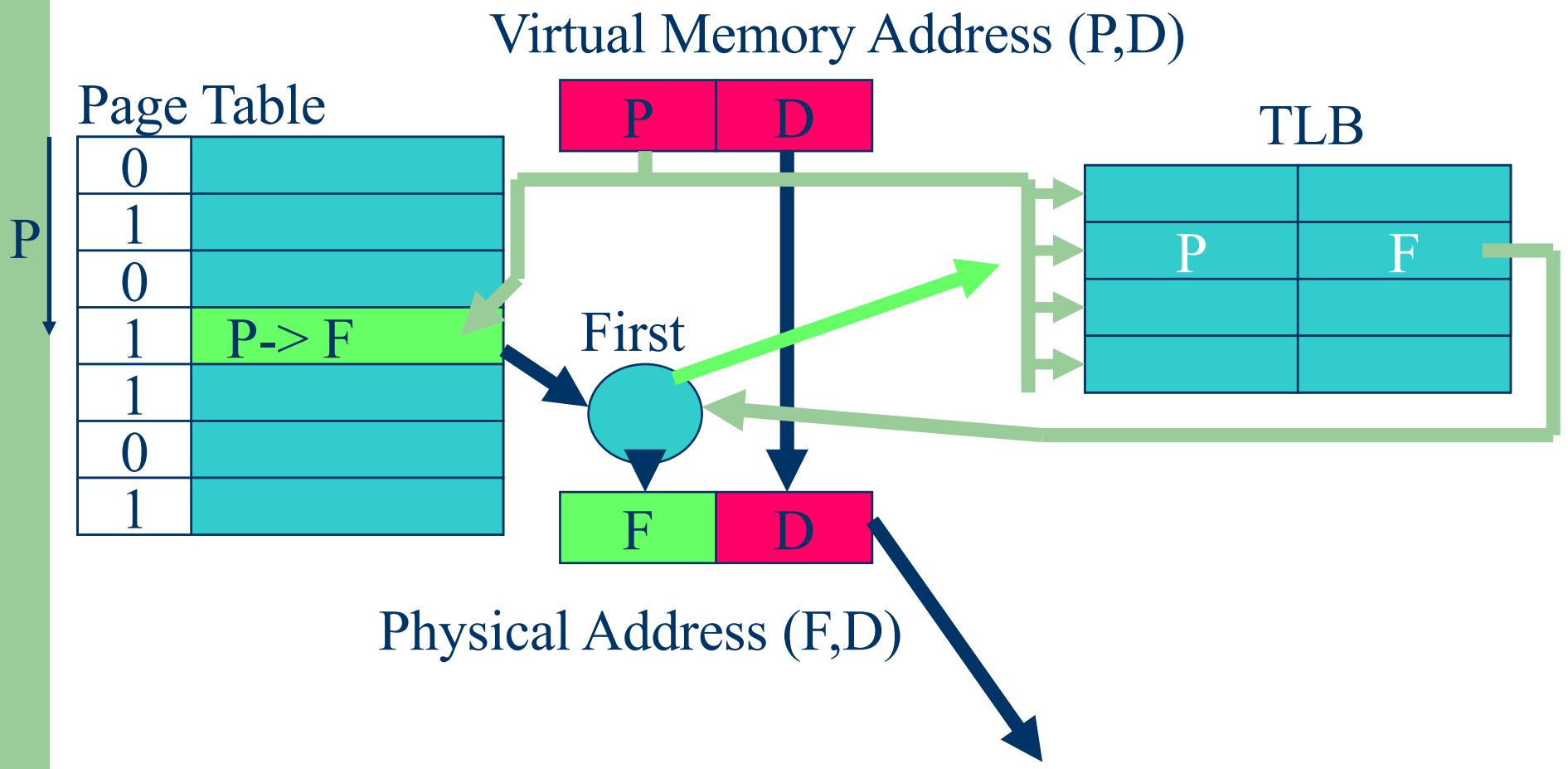
TLBs

- Translation Lookaside Buffers
  - Translate virtual page #s into PTEs (not physical addrs)
  - Can be done in a single machine cycle
- TLBs implemented in hardware
  - Fully associative cache (all entries looked up in parallel)
  - Cache tags are virtual page numbers
  - Cache values are PTEs (entries from page tables)
  - With PTE + offset, can directly calculate physical address
- TLBs exploit locality
  - Processes only use a handful of pages at a time
    - 16-48 entries/pages (64-192K)
    - Only need those pages to be “mapped”
  - Hit rates are therefore very important

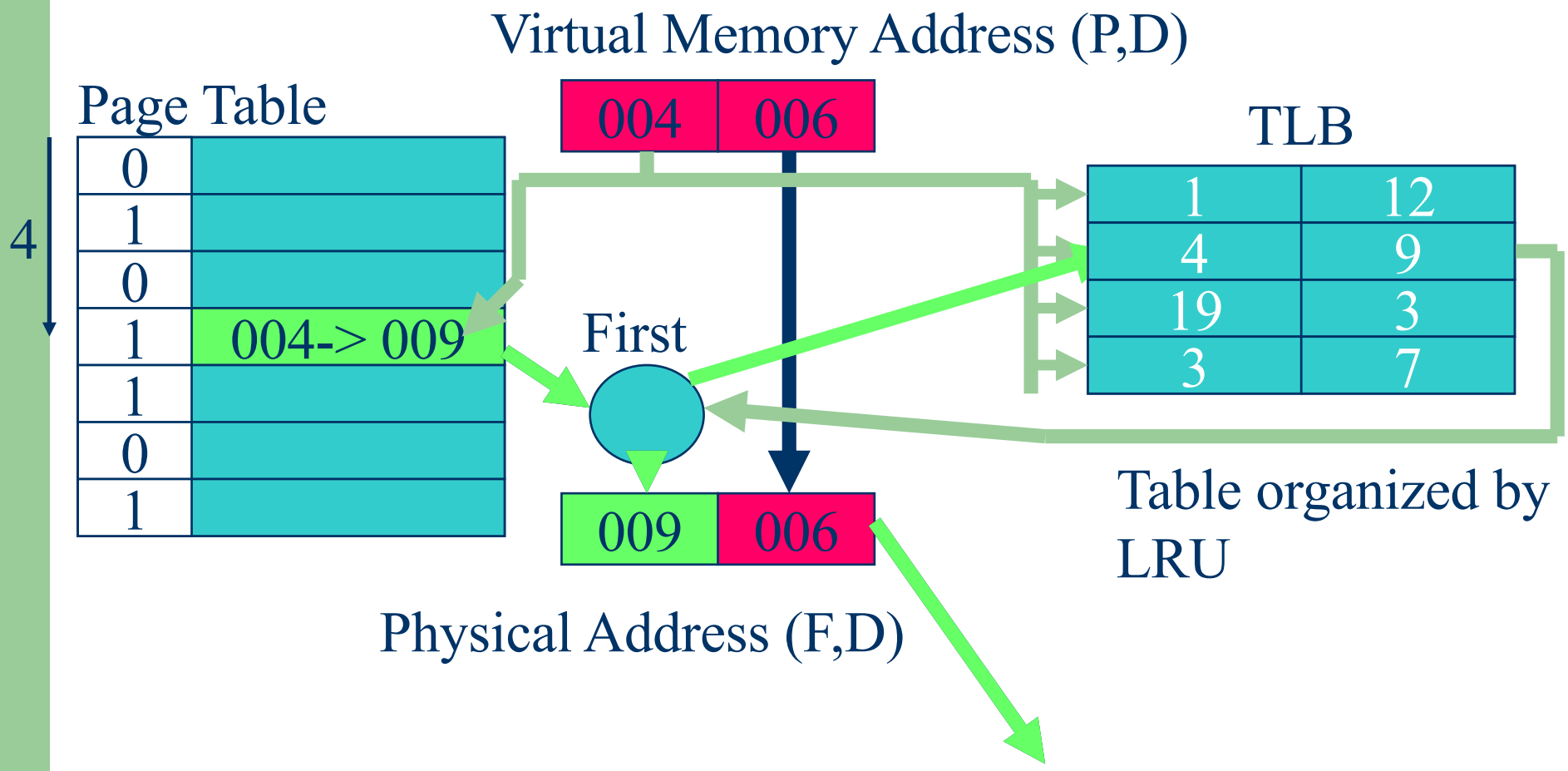
TLB Function

- If a virtual address is presented to MMU, the hardware checks TLB by comparing all entries simultaneously (in parallel).
- If match is valid, the page is taken from TLB without going through page table.
- If match is not valid
  - MMU detects miss and does an ordinary page table lookup.
  - It then evicts one page out of TLB and replaces it with the new entry, so that next time that page is found in TLB.

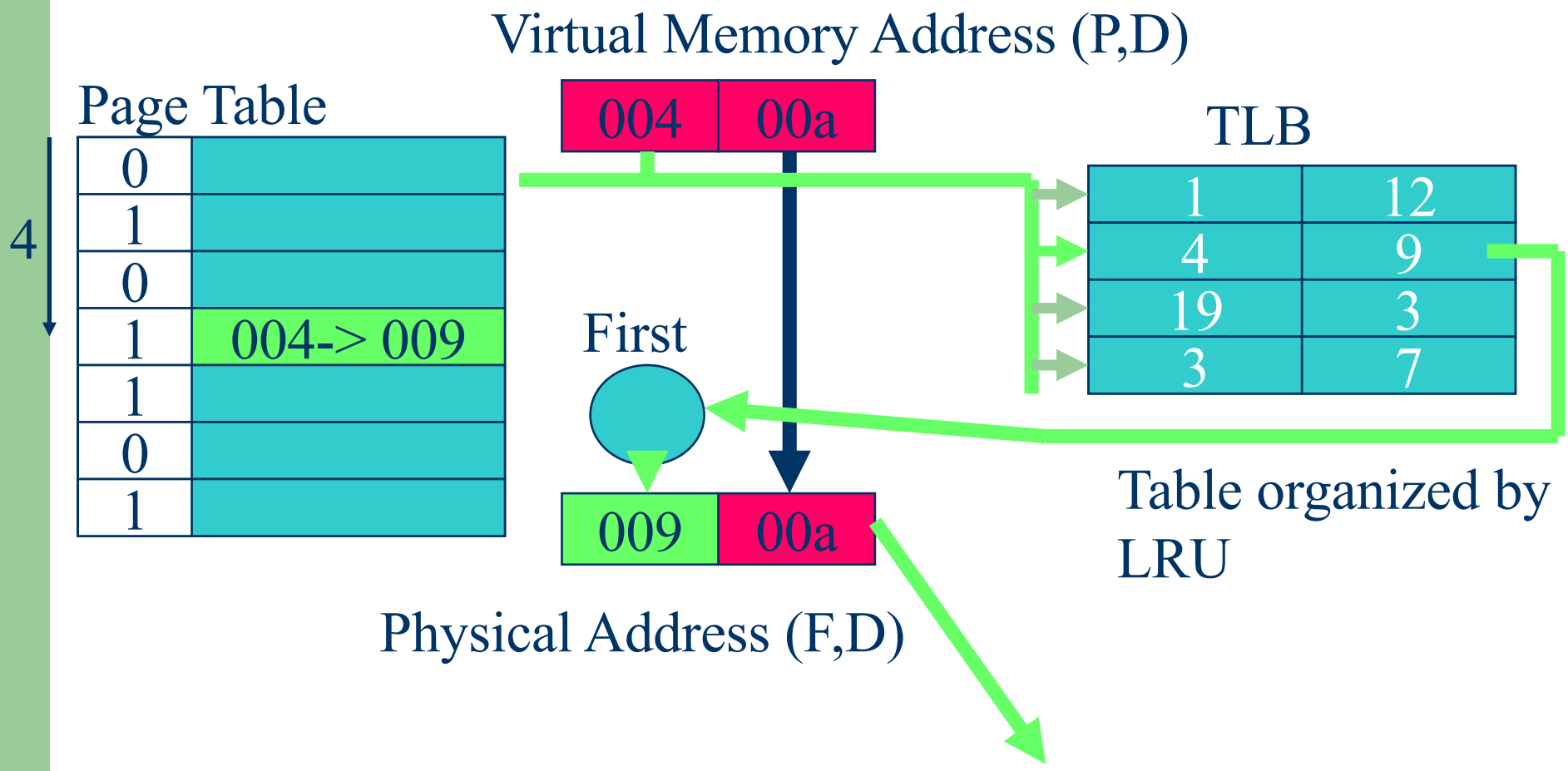
Page Mapping Hardware



Page Mapping Example



Page Mapping Example: next reference



Youtube video

- https://www.youtube.com/watch?v=95QpHJX55bM

Managing TLBs

- Address translations for most instructions are handled using the TLB
  - >99% of translations, but there are misses (TLB miss)…
- Who places translations into the TLB (loads the TLB)?
  - Hardware (Memory Management Unit) [x86]
    - Knows where page tables are in main memory
    - OS maintains tables, HW accesses them directly
    - Tables have to be in HW-defined format (inflexible)
  - Software loaded TLB (OS) [MIPS, Alpha, Sparc, PowerPC]
    - TLB faults to the OS, OS finds appropriate PTE, loads it in TLB
    - Must be fast (but still 20-200 cycles)
    - CPU ISA has instructions for manipulating TLB
    - Tables can be in any format convenient for OS (flexible)

Managing TLBs (2)

- OS ensures that TLB and page tables are consistent
  - When it changes the protection bits of a PTE, it needs to invalidate the PTE if it is in the TLB
- Reload TLB on a process context switch
  - Invalidate all entries
  - Why? What is one way to fix it?
- When the TLB misses and a new PTE has to be loaded, a cached PTE must be evicted
  - Choosing PTE to evict is called the TLB replacement policy
  - If implemented in hardware, often simple (e.g., Last-Not-Used)

Bits in a TLB Entry

- Common (necessary) bits
  - Virtual page number: match with the virtual address
  - Physical page number: translated address
  - Valid
  - Access bits: kernel and user (nil, read, write)
- Optional (useful) bits
  - Process tag
  - Reference
  - Modify
  - Cacheable

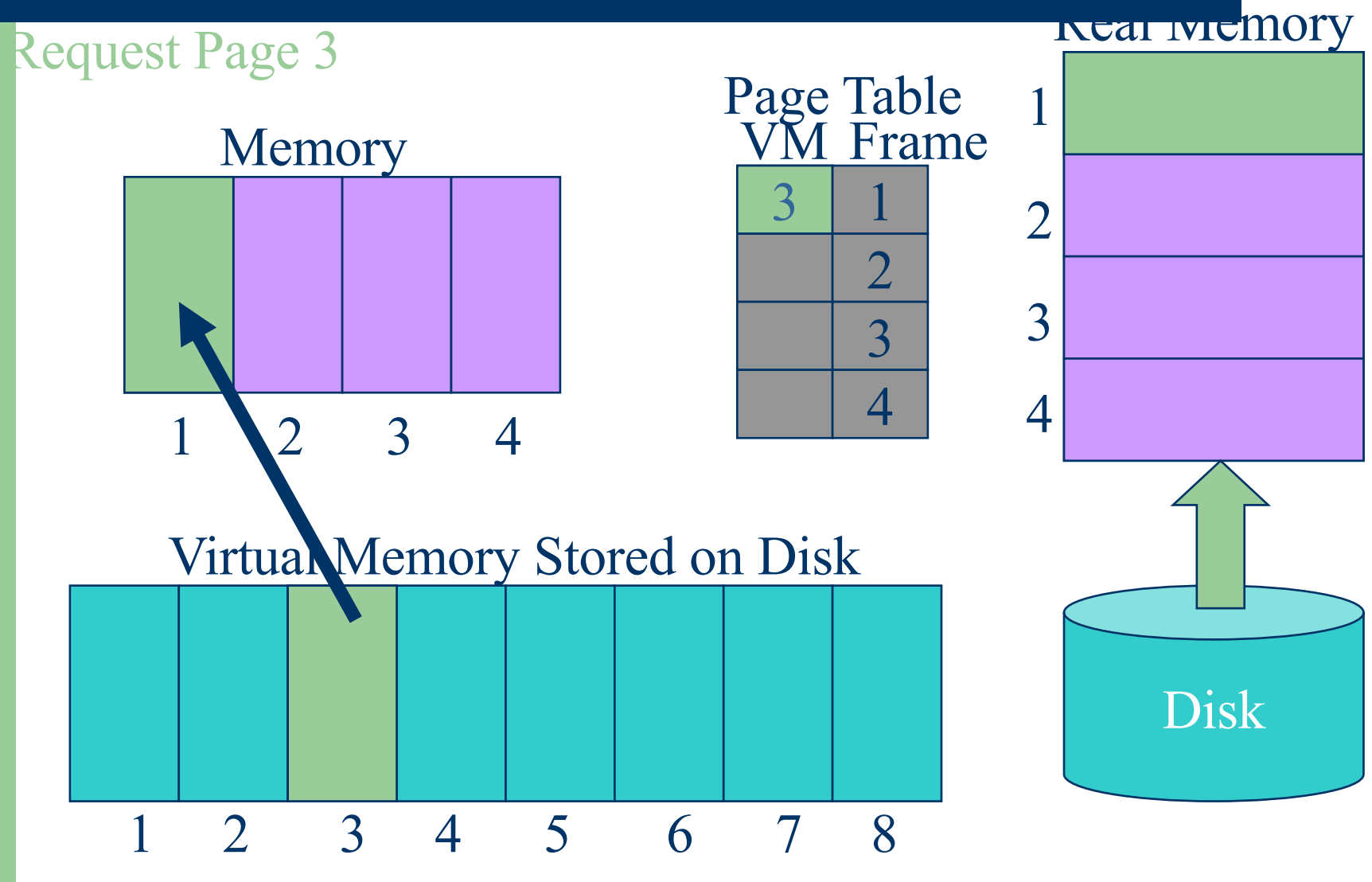
Paging Implementation Issues

- TLB can be implemented using
  - Associative registers
  - Look-aside memory
  - Content-addressable memory
- TLB hit ratio (Page address cache hit ratio)
  - Percentage of time page found in associative memory

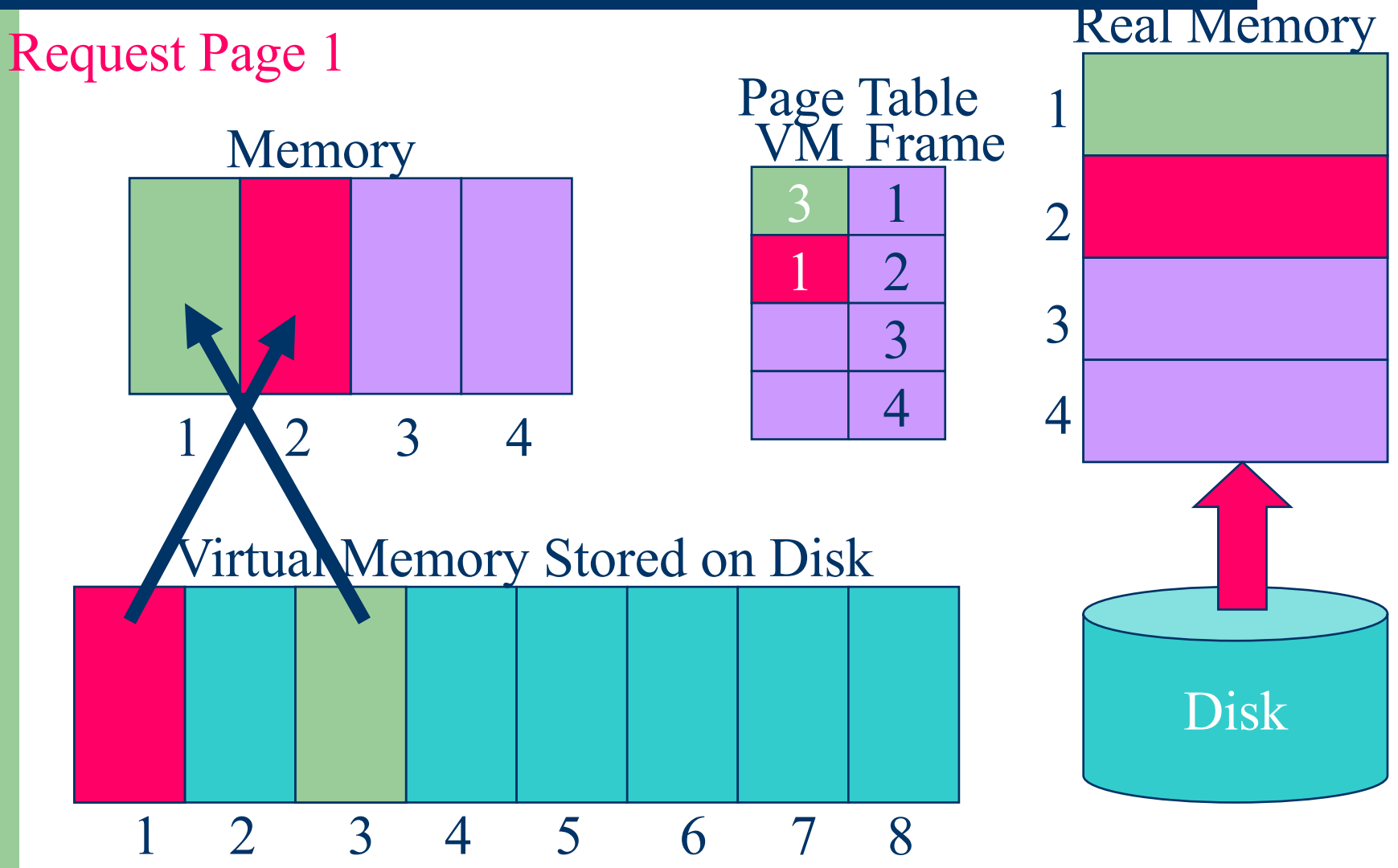
Now we are switching gear

- What if not all virtual memory can fit into physical memory
  - The physical memory is small
  - Too many running processes

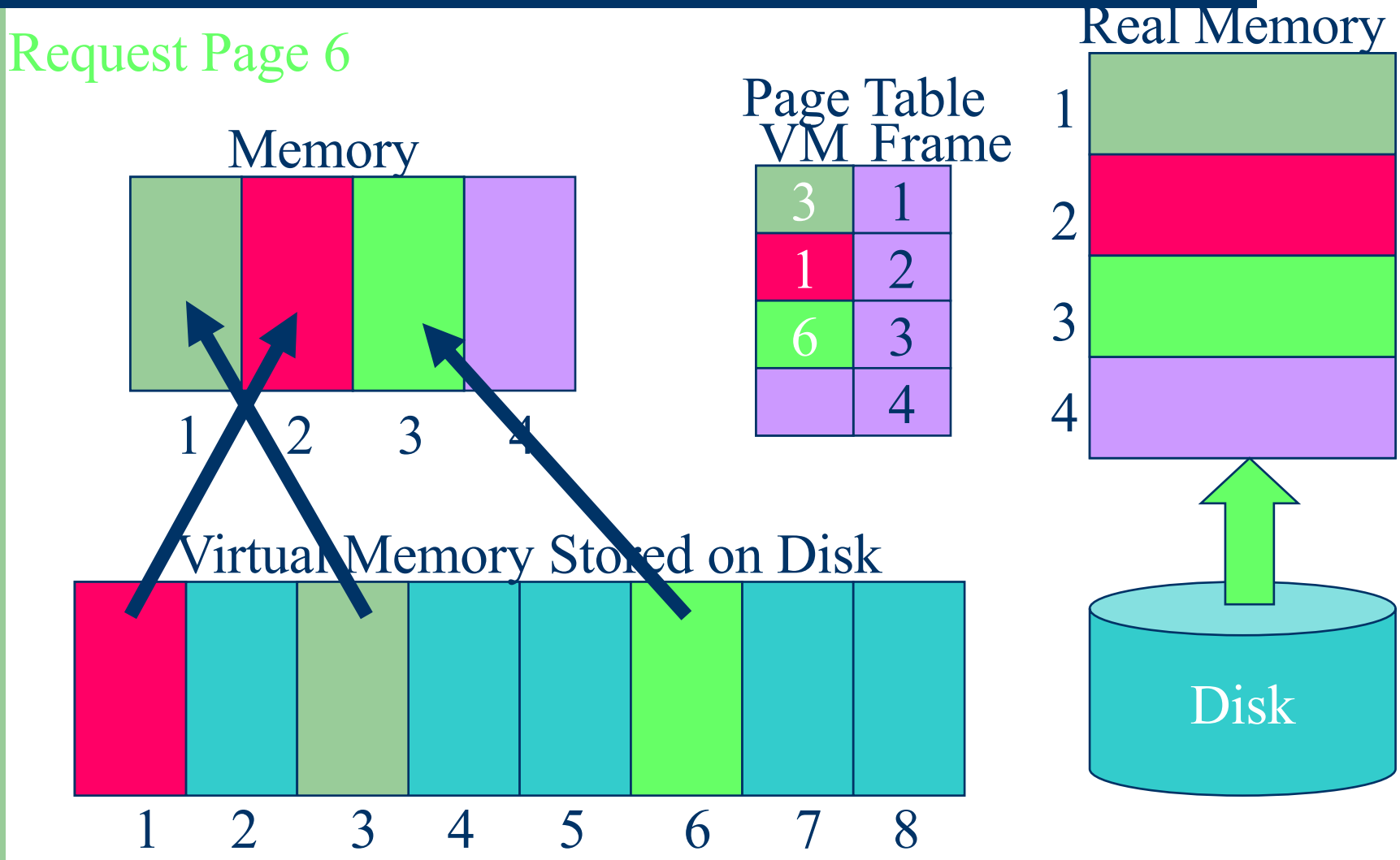
Demand Paging



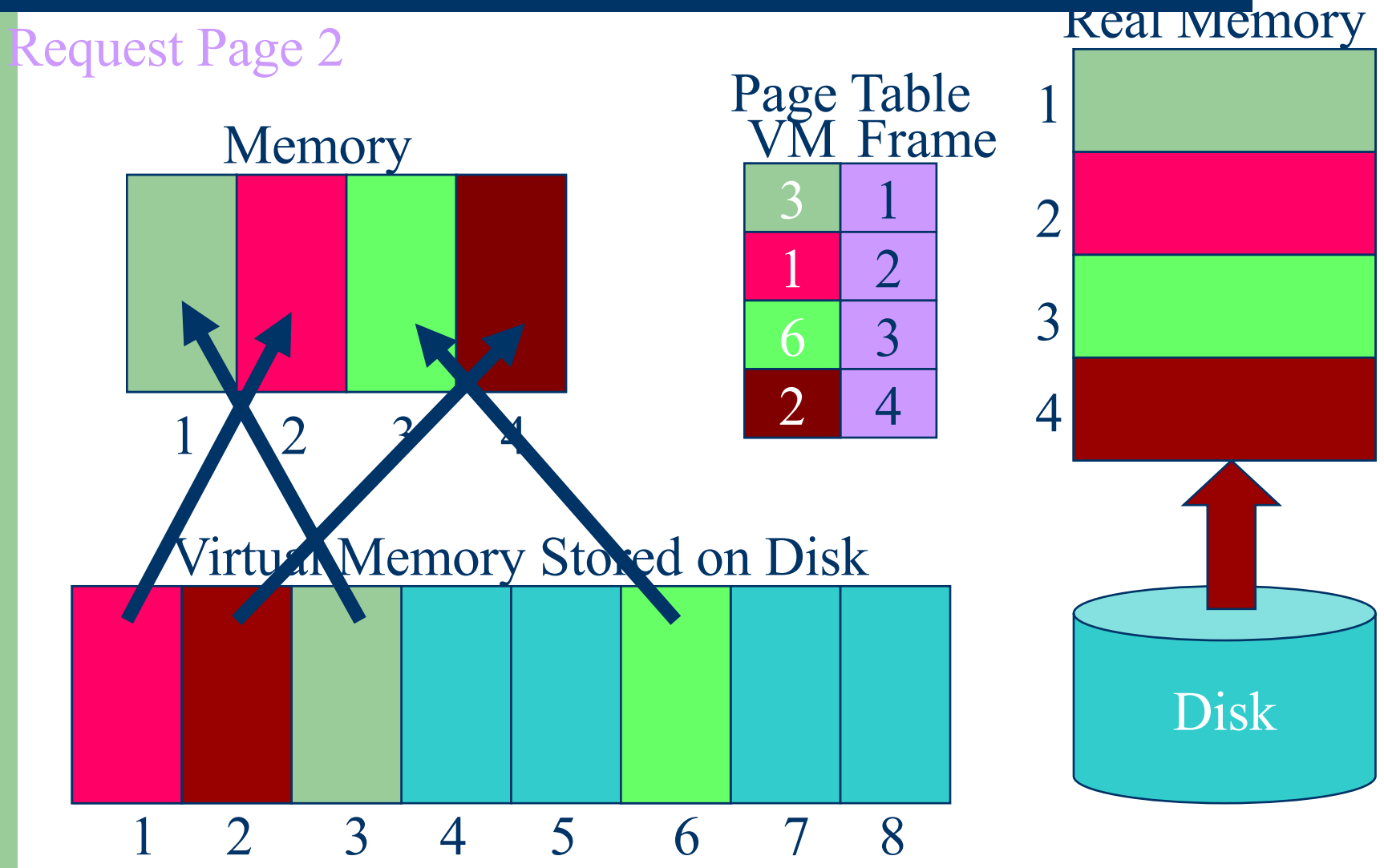
Paging


Request Page 1
Memory
1
2
3
4
Page Table
VM Frame
3 1
1 2
3
4
Real Memory
1
2
3
4
Virtual Memory Stored on Disk
1
2
3
4
5
6
7
8
Disk

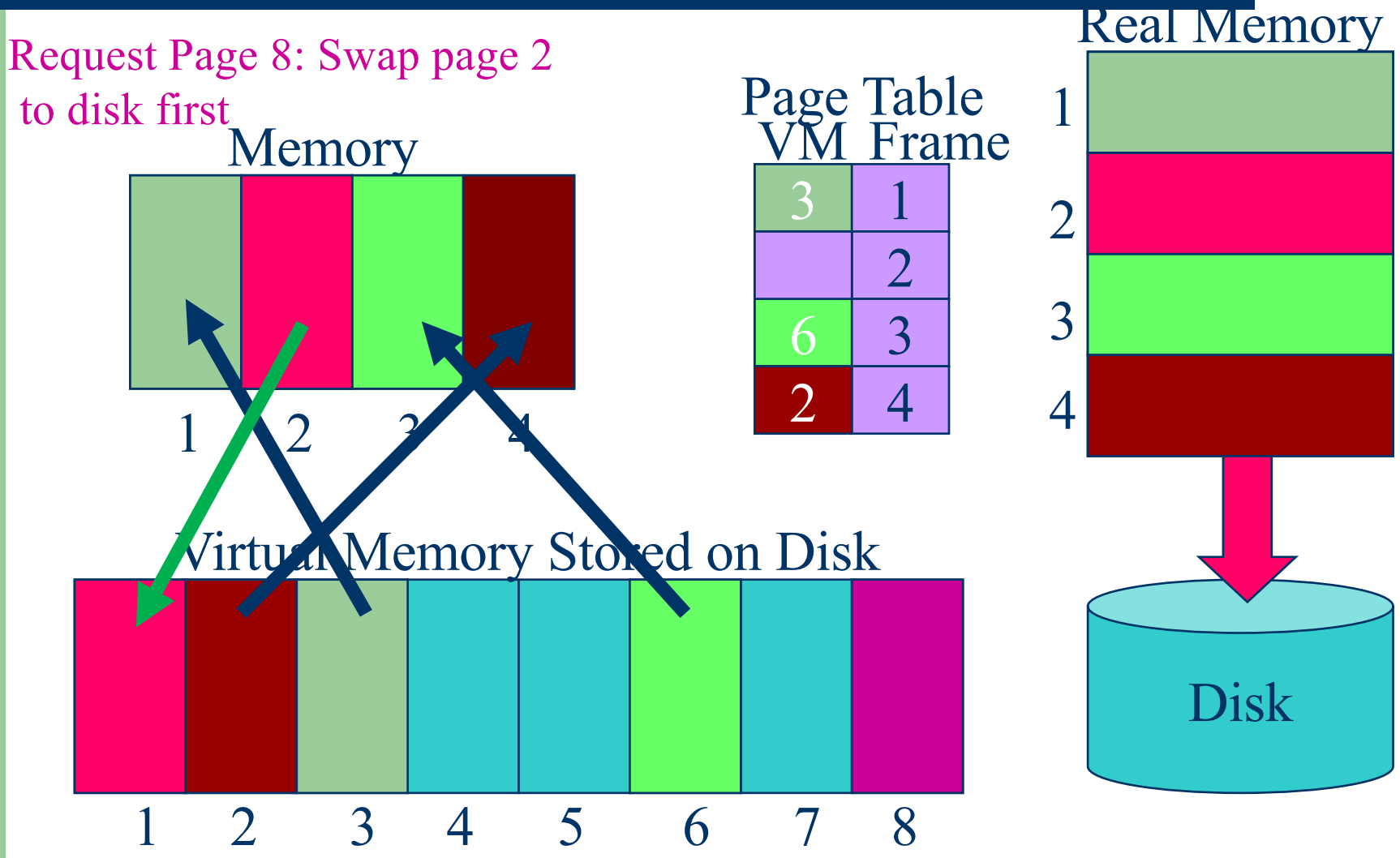
Paging


Request Page 6
Memory
1
2
3
4
Page Table
VM Frame
3 1
1 2
6 3
4
Real Memory
1
2
3
4
Virtual Memory Stored on Disk
1 2 3 4 5 6 7 8
Disk

Paging

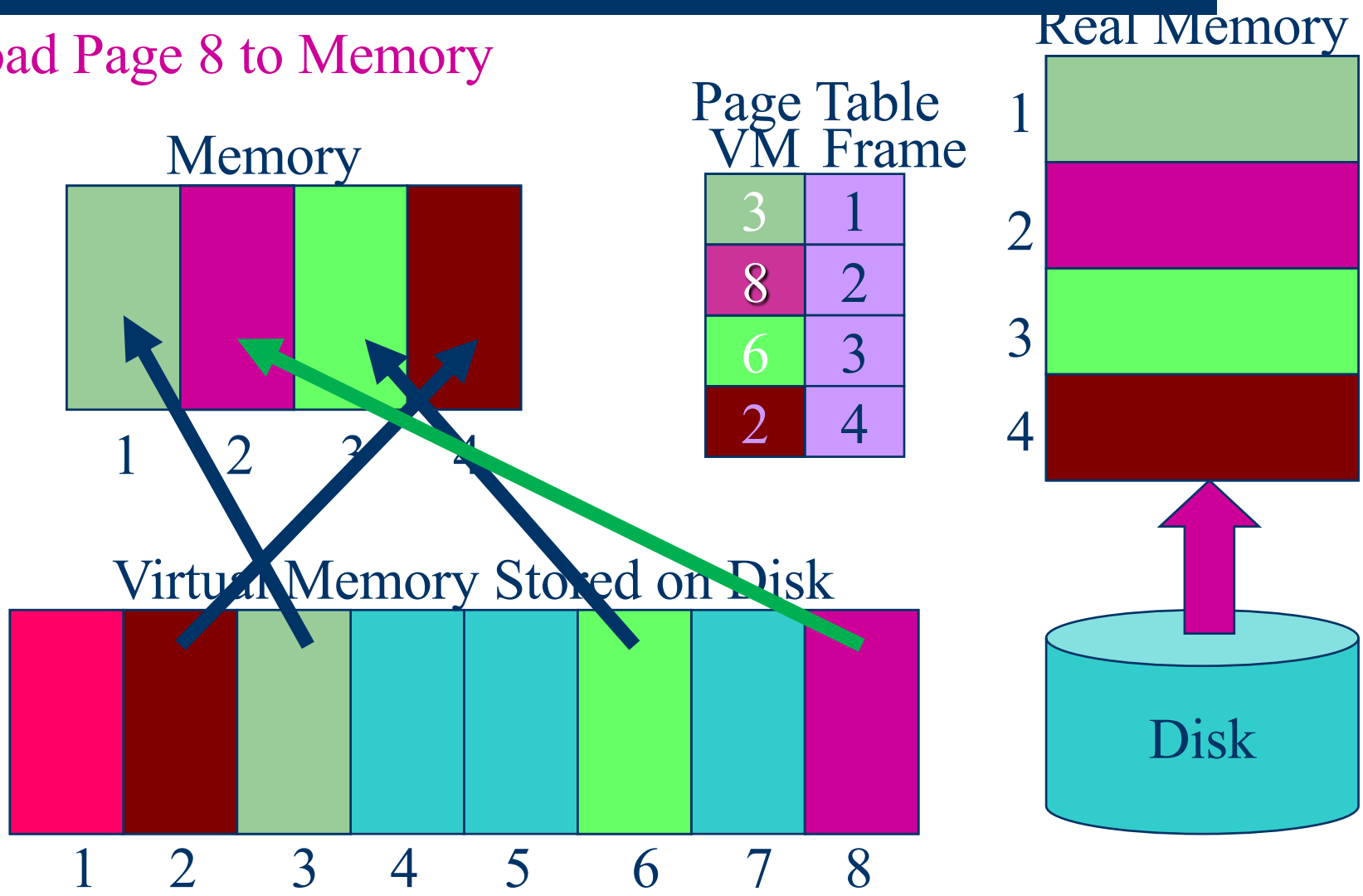

Request Page 2
Memory
Page Table
VM Frame
3 1
1 2
6 3
2 4
Real Memory
1
2
3
4
1 2 3 4
Virtual Memory Stored on Disk
1 2 3 4 5 6 7 8
Disk

Paging


Request Page 8: Swap page 2
to disk first
Memory
1
2
3
4
Page Table
VM Frame
3 1
2
6 3
2 4
Real Memory
1
2
3
4
Disk
Virtual Memory Stored on Disk
1 2 3 4 5 6 7 8

Paging

Load Page 8 to Memory



Summary

Paging mechanisms:

- Optimizations
  - Managing page tables (space)
  - Efficient translations (TLBs) (time)
  - Demand paged virtual memory (space)