

# **CS241 System Programming Memory Management (II)**

Klara Nahrstedt

Lecture 29

4/5/2006



# Content

- Fragmentation
- Virtual Memory
- Paging

# Administrative

- MP4 is posted, due April 17, 2006
- Quiz 8 is April 7, 2006

# Review

- Memory Manager
  - Monitor used and free memory
  - Allocate memory to processes
  - Reclaim (De-allocate) memory
  - Swapping between main memory and disk
- Mono-programming memory management
  - Overlay
- Multi-programming memory management
  - Fixed partition, Variable partition
  - Relocation and protection

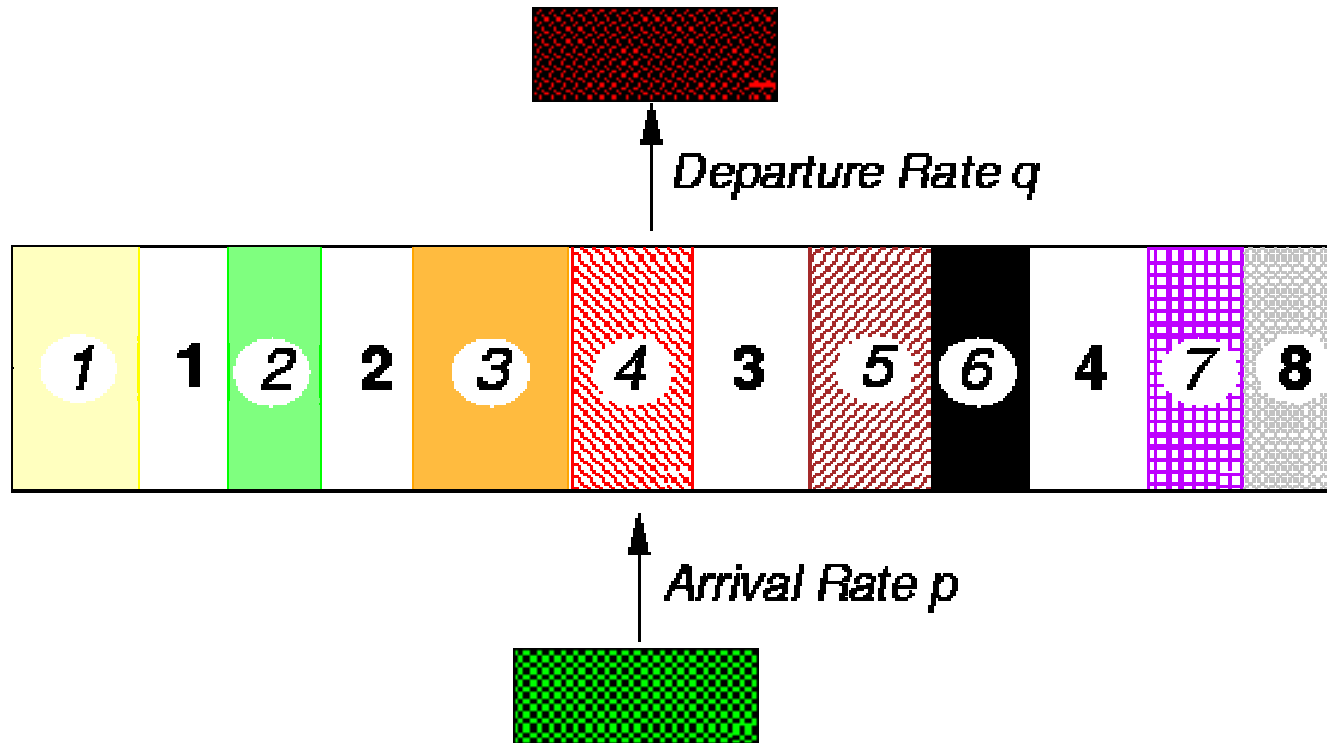
# How Bad Is Fragmentation?

- Statistical arguments - Random sizes
- First-fit
- Given  $N$  allocated blocks
- $0.5*N$  blocks will be lost because of fragmentation
- I.e. 33% of memory may be unusable!!!
- **Known as 50% RULE**
- **Let's look at the derivation of the 50% rule**

# Fragmentation

- consider a system in equilibrium
  - the insertion rate is the same as the deletion rate.
- what can be said about memory usage?

# Fragmentation

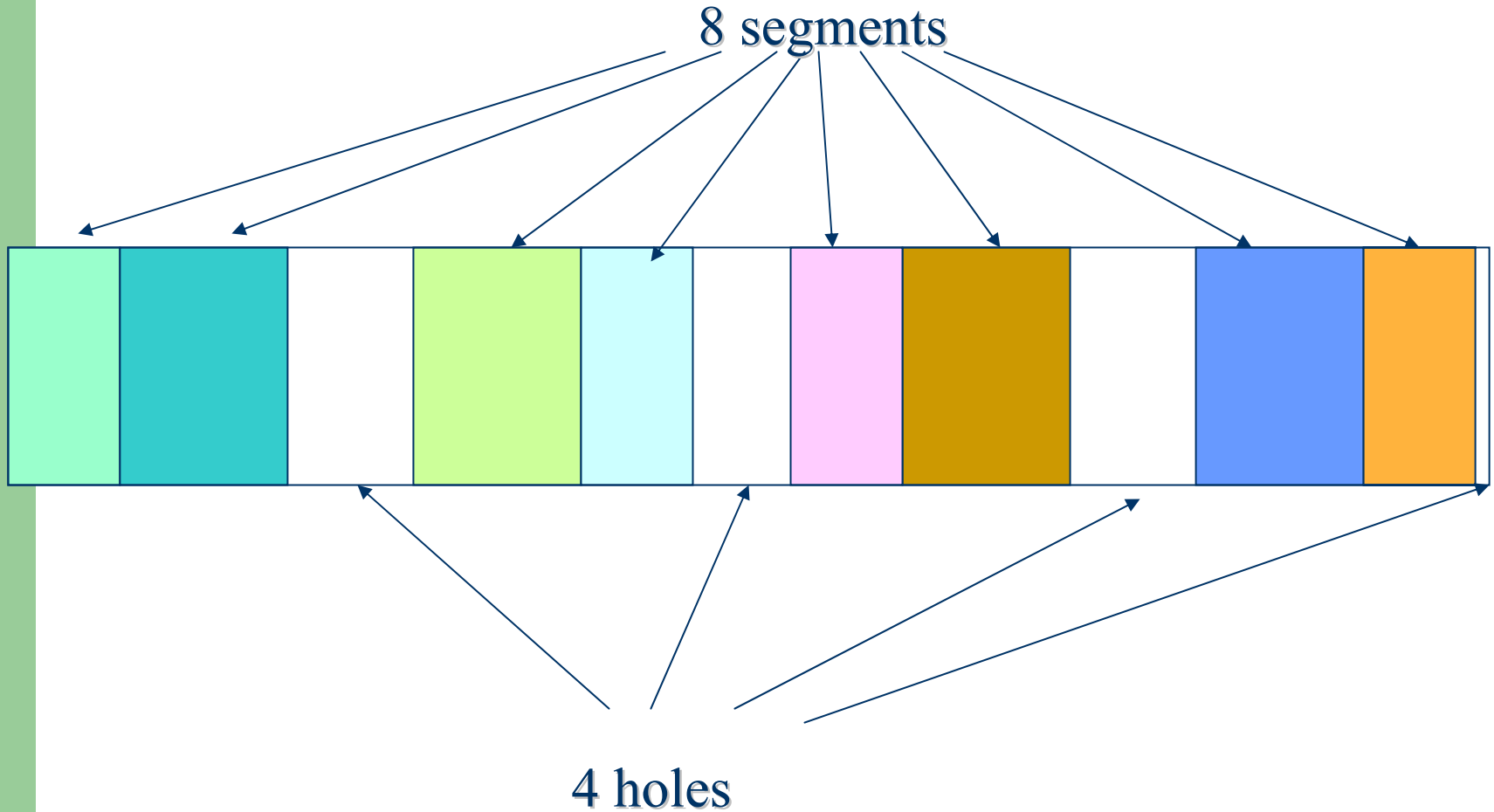


**Under Equilibrium  $p=q$**

**$h = 4$  holes**

**$n = 8$  segments**

# Fragmentation



# Estimates of Fragmentation

- 50% rule
  - if the system has an average of  $n$  segments and  $h$  holes, where  $n$  and  $h$  are large:
$$h \cong \frac{n}{2}$$
  - i.e. there are approximately half as many holes as segments in memory

# Derivation of Unused Memory Rule Estimate

- Let  $f$  be the fraction of memory occupied by holes.
- Let  $S_s$  be the average segment size
- Let the average hole size be at least  $S_h$ .

The ratio of hole size to segment size

$$k = \frac{\text{holesize}}{\text{segmentsize}} \geq \frac{S_h}{S_s}$$

# Derivation of Unused Memory

$$\begin{aligned} f &= \frac{\text{memoryinholes}}{\text{totalmemory}} = \frac{h \times S_h}{h \times S_h + n \times S_s} \\ &= \frac{h \times \frac{S_h}{S_s}}{h \times \frac{S_h}{S_s} + n \times \frac{S_s}{S_s}} \geq \frac{h \times k}{h \times k + n} = \frac{k}{k + 2} \end{aligned}$$

# Estimates of Fragmentation

$$f \geq \frac{k}{k+2} \quad \text{using the 50\% rule}$$

- it is impossible to use all of central memory,
  - $f > 0$
- simulation indicates  $f$  can be made about 10%

# Memory Utilization



# Questions

- What schemes could be used to overcome fragmentation?
- What does the use of secondary storage for swap space imply about memory organization?

# Virtual Memory

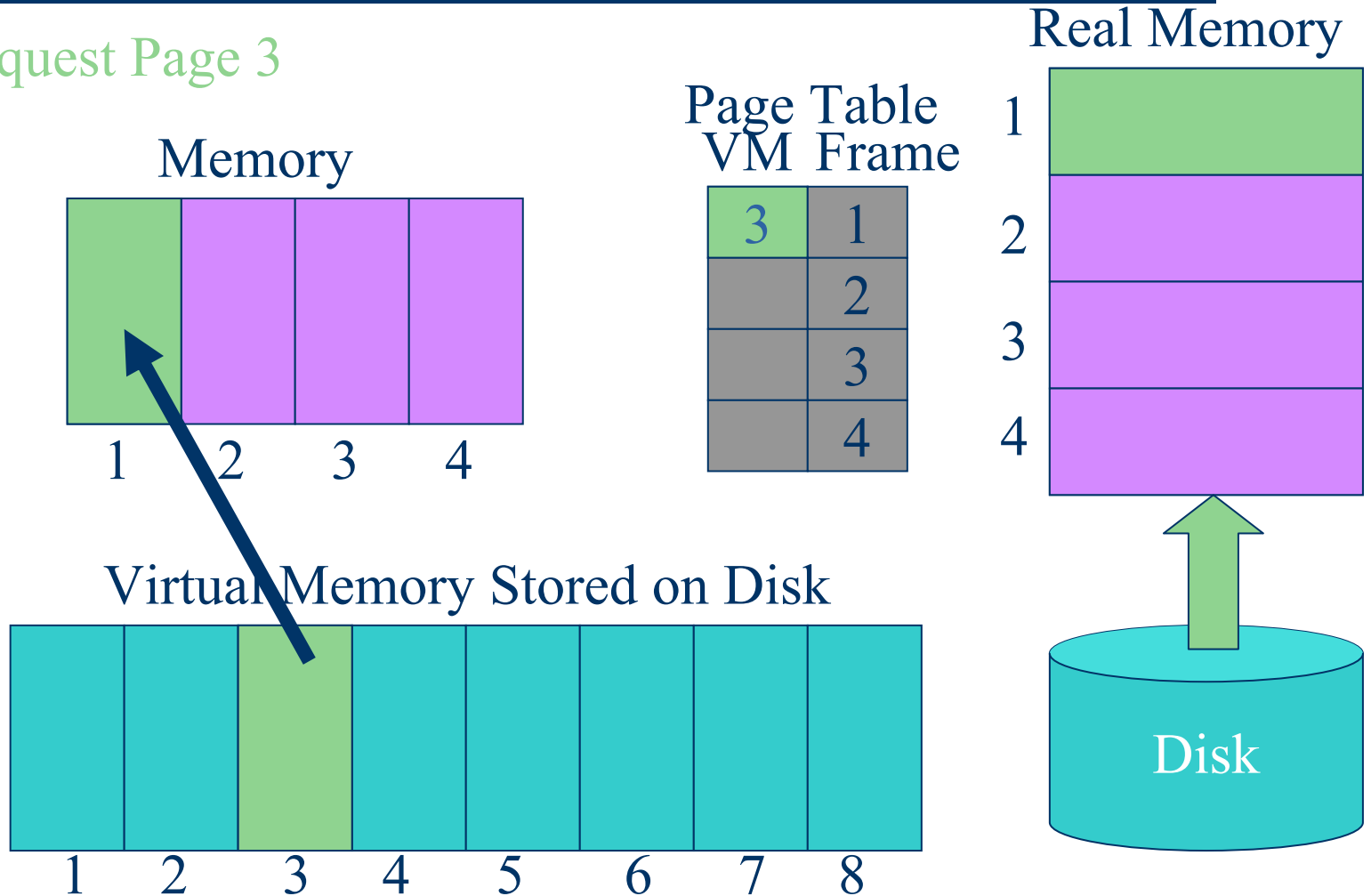
- Provide user with virtual memory that is as big as user needs
- Store virtual memory on disk
- Cache parts of virtual memory being used in real memory
- Load and store cached virtual memory without user program intervention

# Benefits of Virtual Memory

- Use secondary storage(\$)
  - Extend DRAM(\$\$\$) with reasonable performance
- Protection
  - Programs do not step over each other
- 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)
- Remove memory deadlock possibilities:
  - permit pre-emption of real memory

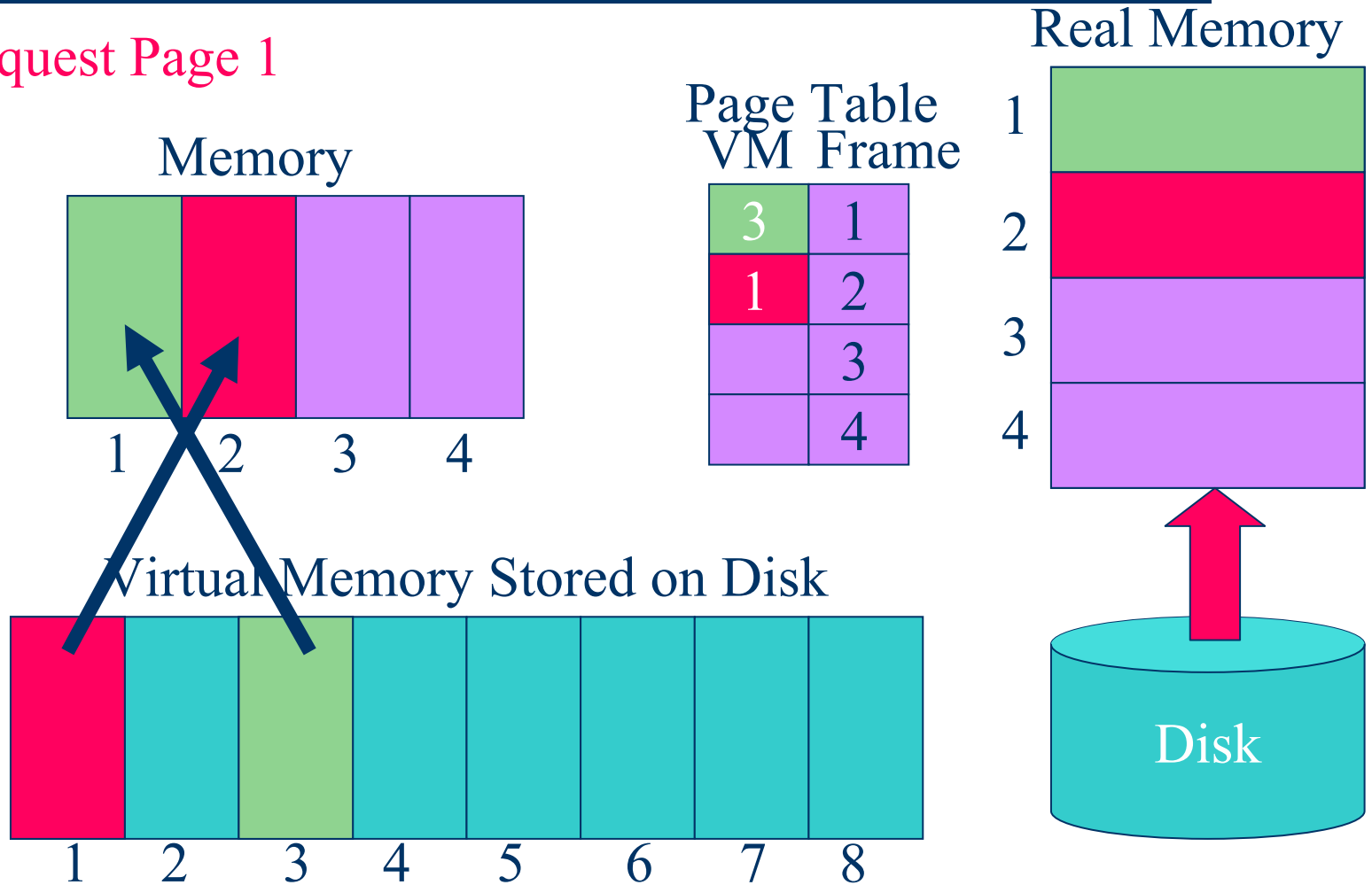
# Paging Request

Request Page 3



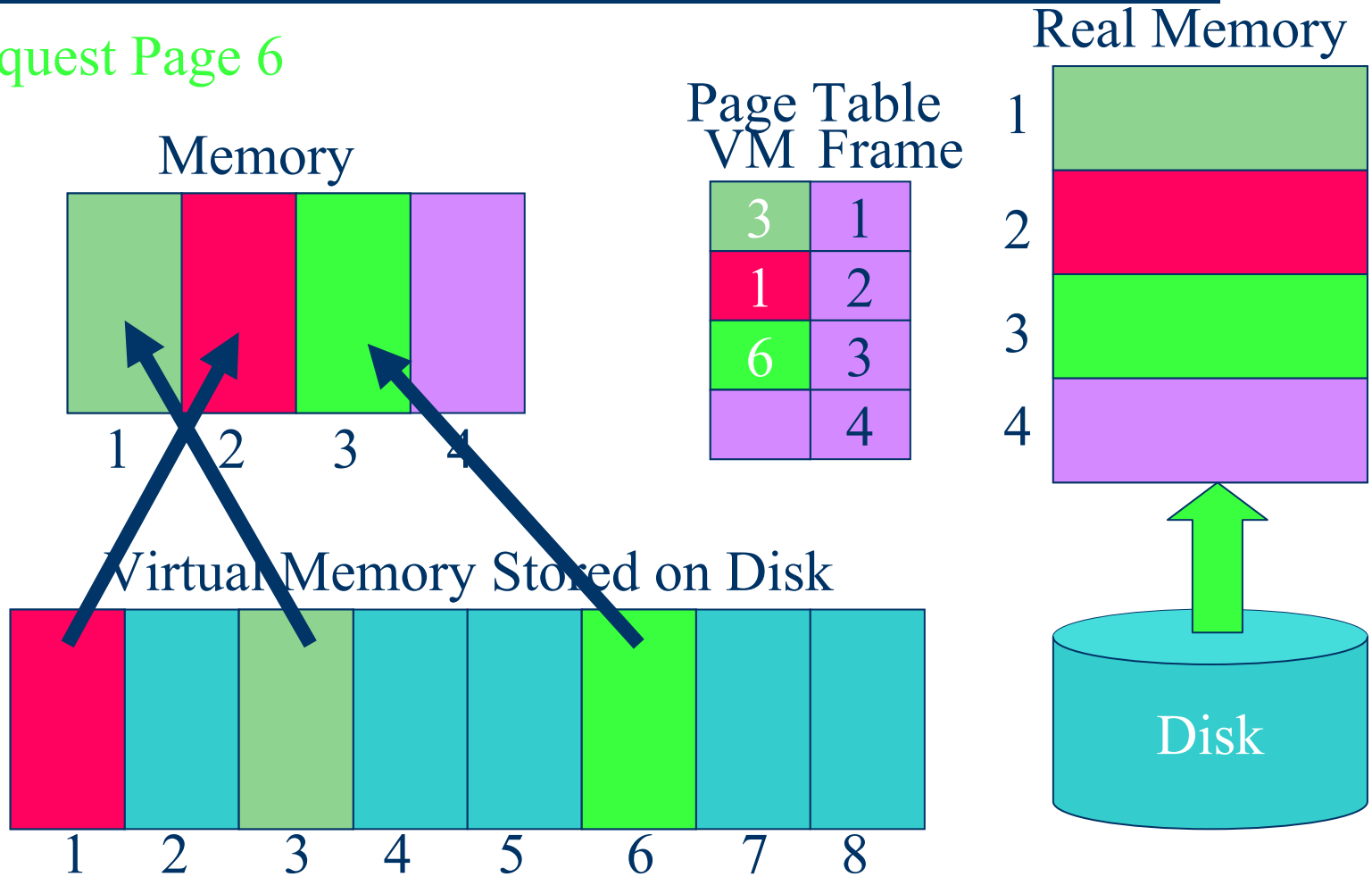
# Paging

Request Page 1



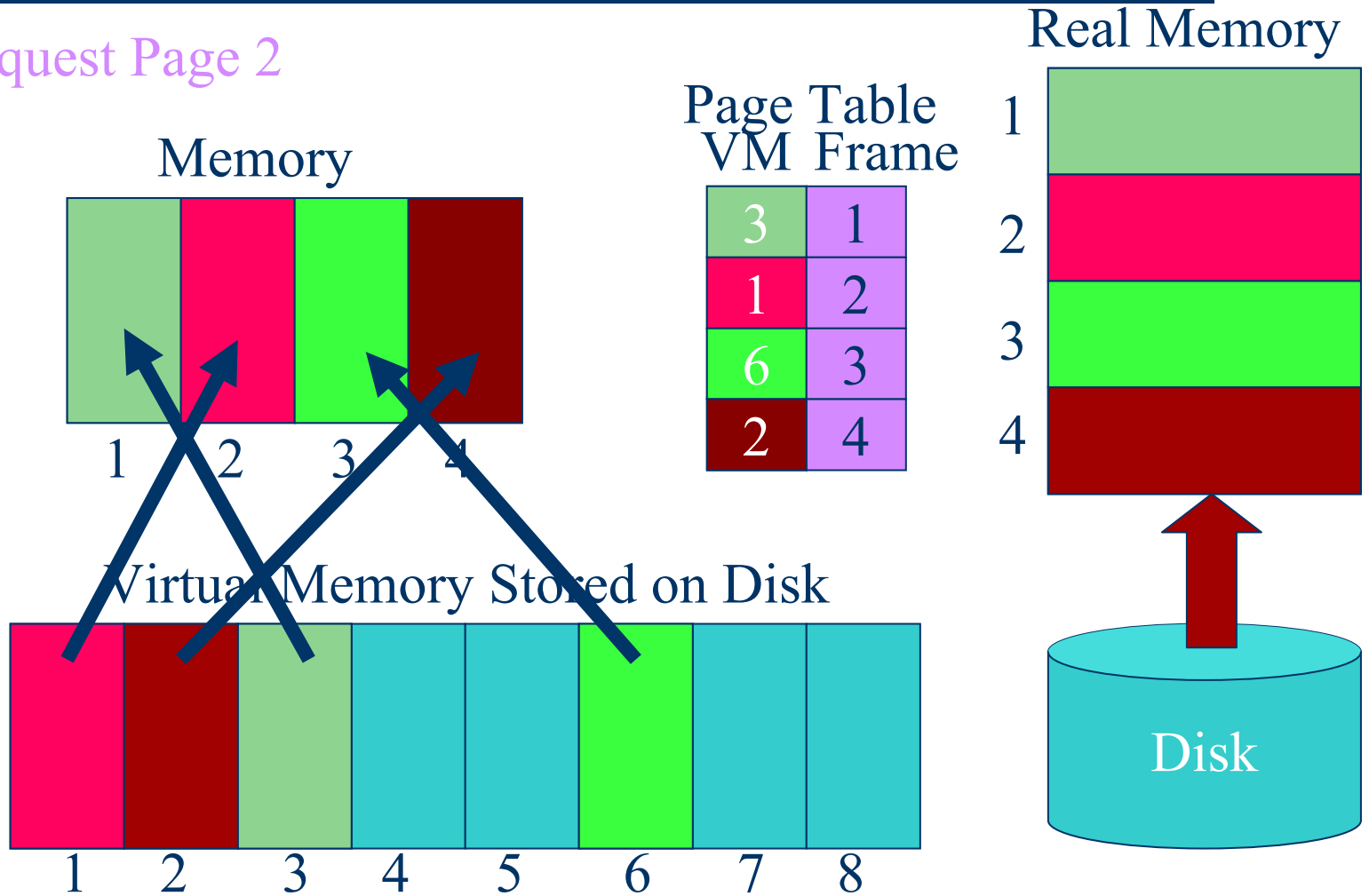
# Paging

Request Page 6



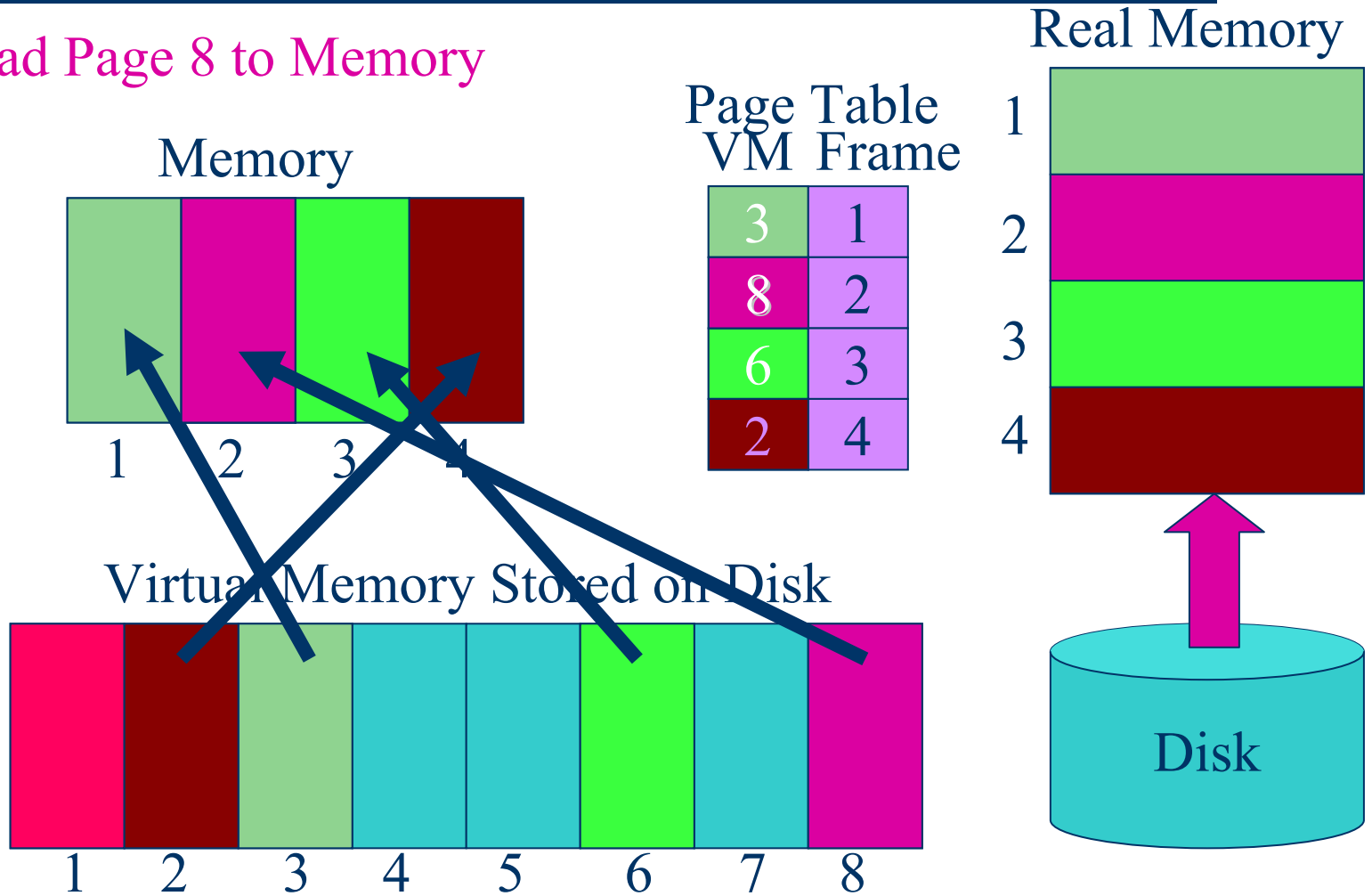
# Paging

Request Page 2

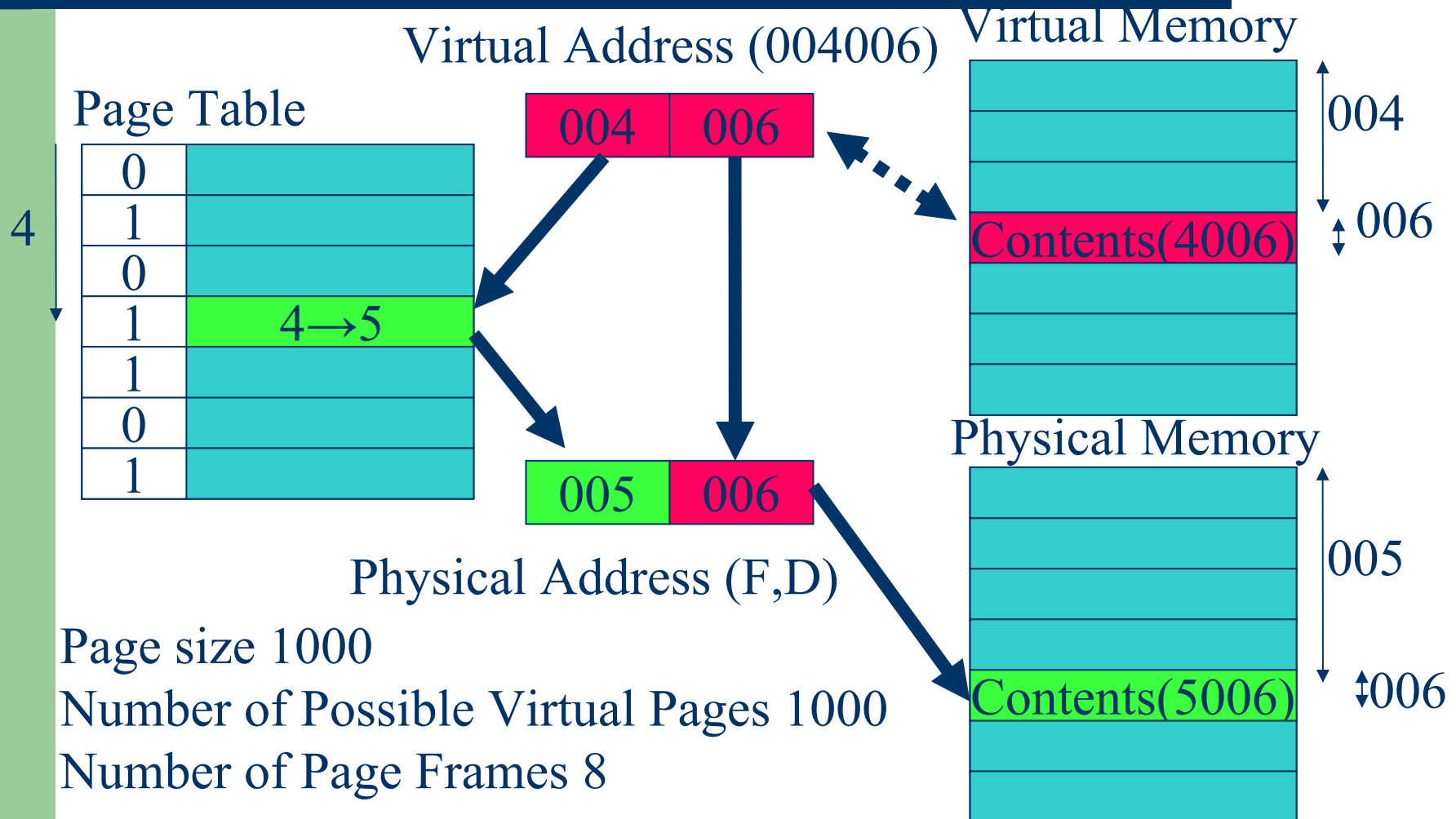


# Paging

Load Page 8 to Memory



# Page Mapping Hardware



# Page Fault

- Access a virtual page that is not mapped into any physical page
  - A fault is triggered by hardware
- Page fault handler (in OS's VM subsystem)
  - Find if there is any free physical page available
    - If no, evict some resident page to disk (swapping space)
  - Allocate a free physical page
  - Load the faulted virtual page to the prepared physical page
  - Modify the page table

# Paging Issues

- Page size is  $2^n$ 
  - usually 512 bytes, 1 KB, 2 KB, 4 KB, or 8 KB
  - E.g. 32 bit VM address may have  $2^{20}$  (1 MB) pages with 4k ( $2^{12}$ ) bytes per page
- Page table:
  - $2^{20}$  page entries take  $2^{22}$  bytes (4 MB)
  - page frames must map into real memory
  - Page Table base register must be changed for context switch
- No external fragmentation; internal fragmentation on last page *only*

# Virtual-to-Physical Lookups

- Programs only know virtual addresses
  - The page table can be extremely large
- Each virtual address must be translated
  - May involve walking hierarchical page table
  - Page table stored in memory
  - So, each program memory access requires several actual memory accesses
- Solution: cache “active” part of page table
  - TLB is an “associative memory”

# Summary

- Fragmentation needs to be minimized
- Virtual memory is a great concept, enlarges the memory space that the users/programs can use
- Pages, Page Tables, Page Table Hardware are important concepts
- Virtual to Physical address translation must be done fast and correctly.