Memory Management: Paging and Virtual Memory
CS 111
Operating Systems
Peter Reiher

Outline

- Paging
- Swapping and demand paging
- Virtual memory

Paging

- What is paging?
  - What problem does it solve?
  - How does it do so?
- Paged address translation
- Paging and fragmentation
- Paging memory management units
- Paging and segmentation

Segmentation Revisited

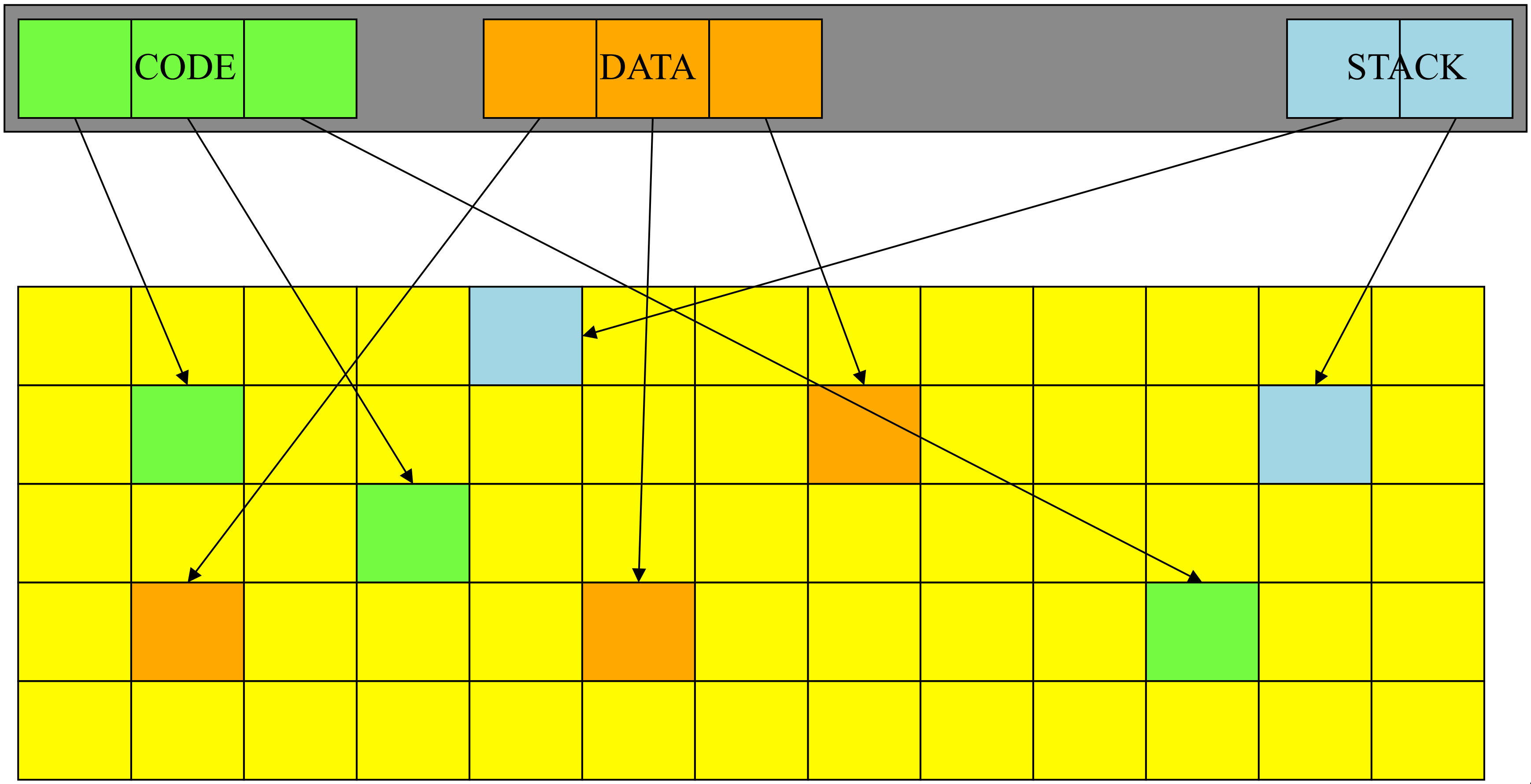
- Segment relocation solved the relocation problem for us
- It used base registers to compute a physical address from a virtual address
  - Allowing us to move data around in physical memory
  - By only updating the base register
- It did nothing about external fragmentation
  - Because segments are still required to be <u>contiguous</u>
- We need to eliminate the "contiguity requirement"

The Paging Approach

- Divide physical memory into units of a single fixed size
  - A pretty small one, like 1-4K bytes or words
  - Typically called a *page frame*
- Treat the virtual address space in the same way
- For each virtual address space page, store its data in one physical address page frame
- Use some magic per-page translation mechanism to convert virtual to physical pages

Paged Address Translation

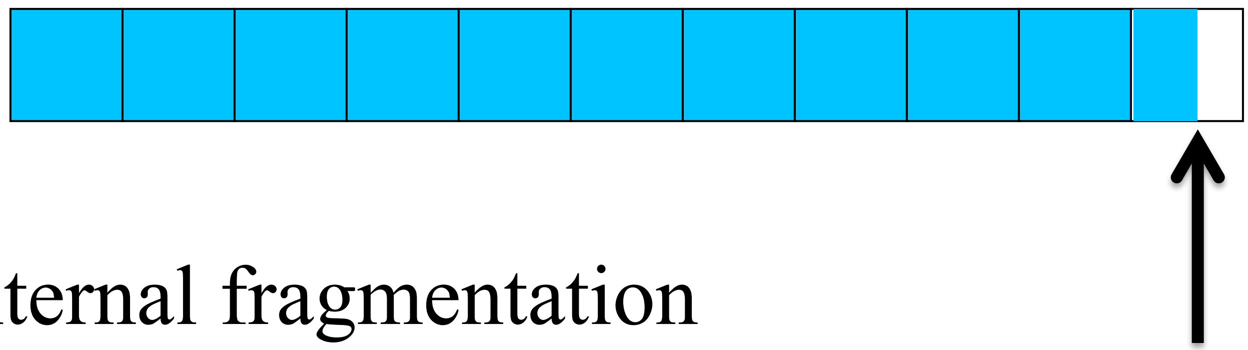
process virtual address space



physical memory

Paging and Fragmentation

- A segment is implemented as a set of virtual pages



- Internal fragmentation
  - Averages only ½ page (half of the last one)
- External fragmentation
  - Completely non-existent
  - We never carve up pages

How Does This Compare To Segment Fragmentation?

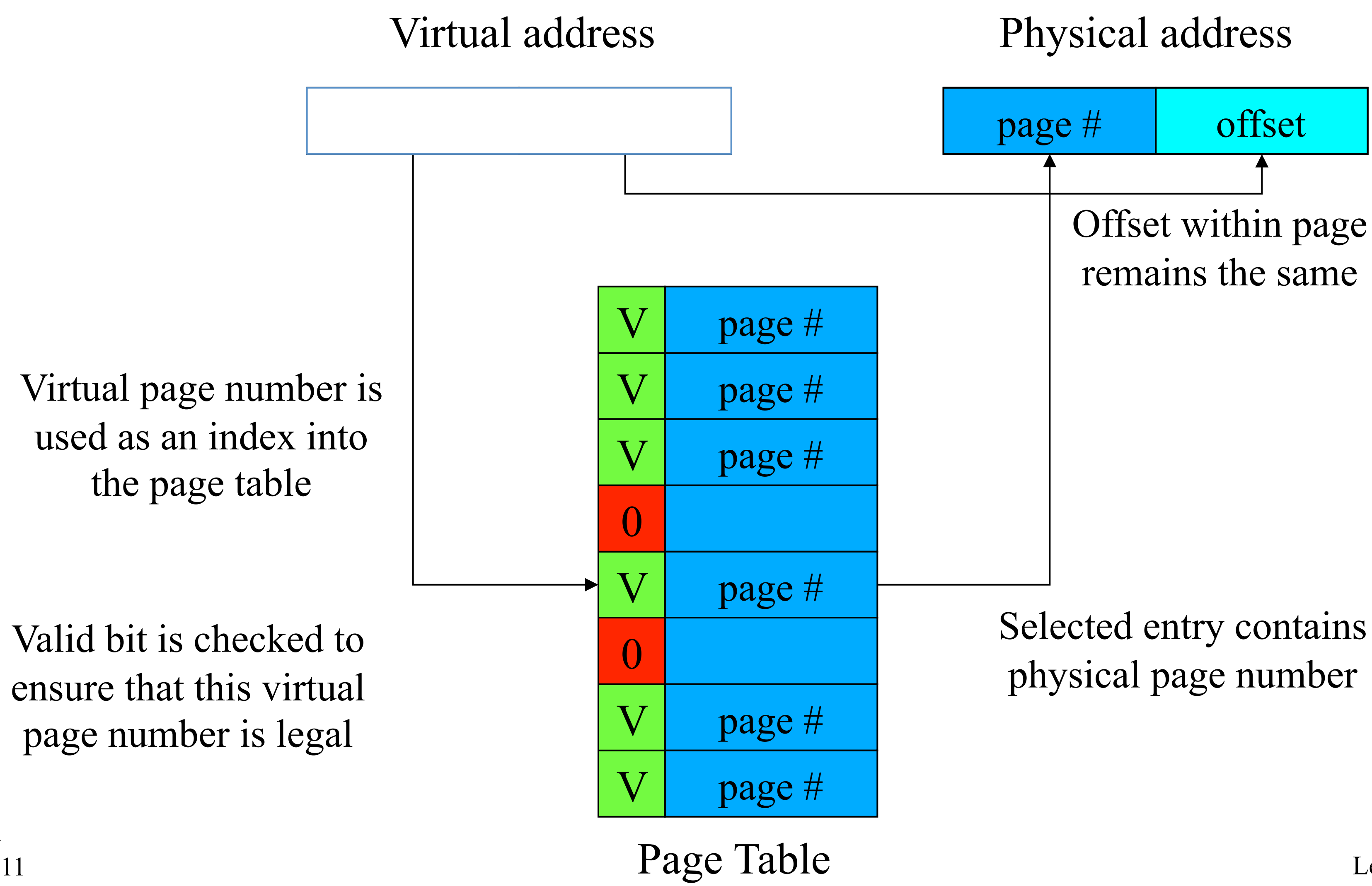
- Consider this scenario:
  - Average requested allocation is 128K
  - 256K fixed size segments available
  - In the paging system, 4K pages
- For segmentation, average internal fragmentation is 50% (128K of 256K used)
- For paging?
  - Only the last page of an allocation is not full
  - On average, half of it is unused, or 2K
  - So 2K of 128K is wasted, or around 1.5%

- **Segmentation: 50% waste**
- **Paging: 1.5% waste**

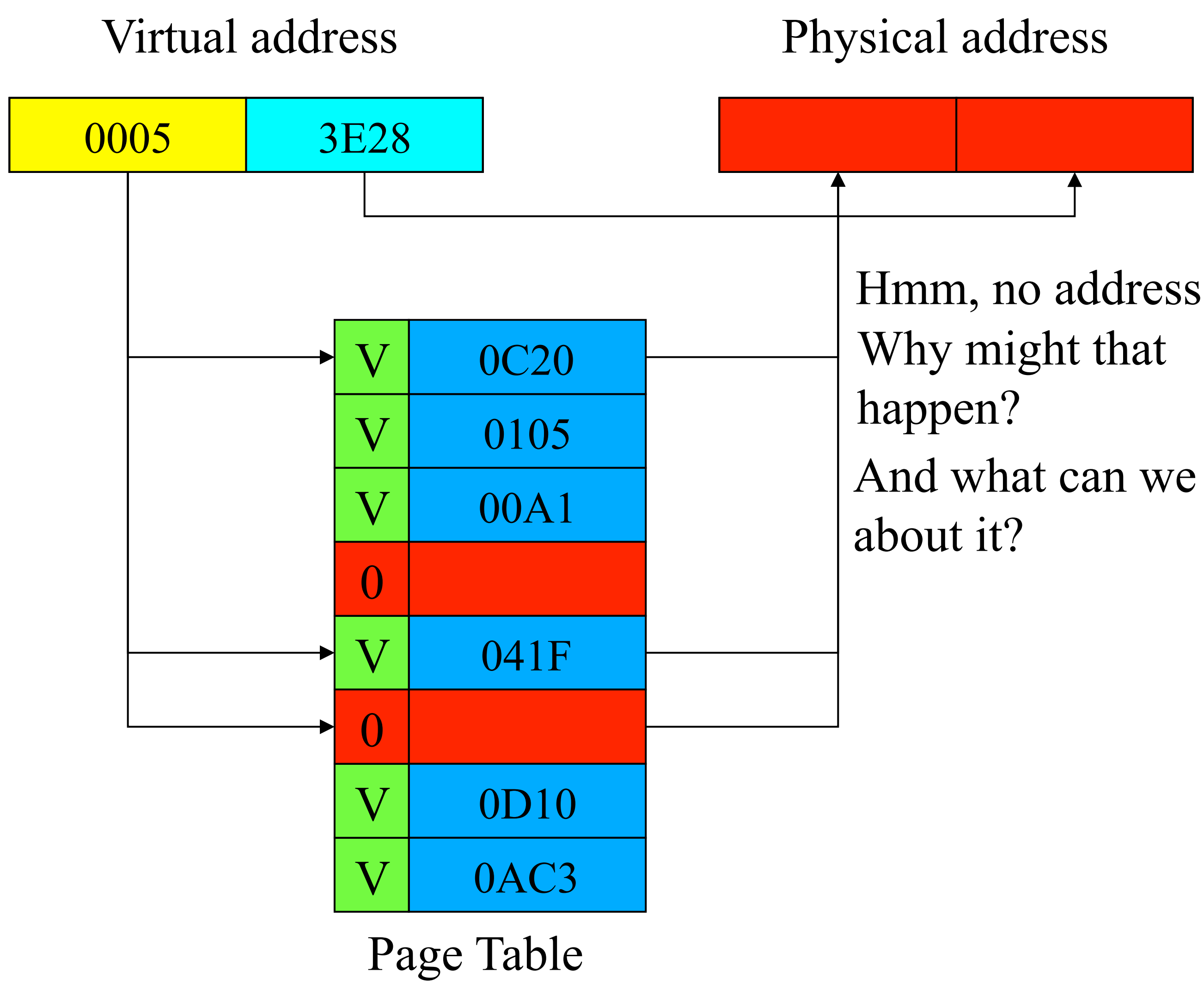
Providing the Magic Translation Mechanism

- On per page basis, we need to change a virtual address to a physical address
- Needs to be fast
  – So we'll use hardware
- The Memory Management Unit (MMU)
  – A piece of hardware designed to perform the magic quickly

Paging and MMUs



Some Examples



Hmm, no address
Why might that happen?

And what can we do about it?

The MMU Hardware

- MMUs used to sit between the CPU and bus
  - Now they are typically integrated into the CPU
- What about the page tables?
  - Originally implemented in special fast registers
  - But there's a problem with that today
  - If we have 4K pages, and a 64 Gbyte memory, how many pages are there?
  - $2^{36}/2^{12} = 2^{24}$
  - Or 16 M of pages
  - We can't afford 16 M of fast registers

Handling Big Page Tables

- 16 M entries in a page table means we can't use registers
- So now they are stored in normal memory
- But we can't afford 2 bus cycles for each memory access
  - One to look up the page table entry
  - One to get the actual data
- So we have a very fast set of MMU registers used as a cache
  - Which means we need to worry about hit ratios, cache invalidation, and other nasty issues
  - TANSTAAFL

The MMU and Multiple Processes

- There are several processes running
- Each needs a set of pages
- We can put any page anywhere
- But if they need, in total, more pages than we’ve physically got,
- Something’s got to go
- How do we handle these ongoing paging requirements?

Ongoing MMU Operations

- What if the current process adds or removes pages?
  - Directly update active page table in memory
  - Privileged instruction to flush (stale) cached entries
- What if we switch from one process to another?
  - Maintain separate page tables for each process
  - Privileged instruction loads pointer to new page table
  - A reload instruction flushes previously cached entries
- How to share pages between multiple processes?
  - Make each page table point to same physical page
  - Can be read-only or read/write sharing

So Is Paging Perfect?

- Pages are a very nice memory allocation unit
  - They eliminate internal and external fragmentation
  - They require a very simple but powerful MMU
- They are not a particularly natural unit of data
  - Programmers don't think in terms of pages
  - Programs are comprised of, and operate on, segments
  - Segments are the natural "chunks" of virtual address space
    - E.g., we map a new segment into the virtual address space
  - Each code, data, stack segment contains many pages

Paging and Segmentation

- We can use both segments and pages
- Programs request segments
  - Each code, data, stack segment contains many pages
- Requires two levels of memory management abstraction
  - A virtual address space is comprised of segments
  - Relocation & swapping is done on a page basis
  - Segment based addressing, with page based relocation
- User processes see segments, paging is invisible

Relationships Between Segments and Pages

- A segment is a named collection of pages
- Operations on segments:
  - Create/open/destroy
  - Map/unmap segment to/from process
  - Find physical page number of virtual page *n*
- Connection between paging & segmentation
  - Segment mapping implemented with page mapping
  - Page faulting uses segments to find requested page

Segmentation on Top of Paging