Virtual and Physical Addresses

Physical addresses are provided by the hardware:

- one physical address space per machine;
- valid addresses are usually between 0 and some machine-specific maximum;
- not all addresses have to belong to the machine's main memory; other hardware devices can be mapped into the address space.

Virtual (or *logical*) addresses are provided by the OS kernel:

- one virtual address space per process;
- addresses may start at zero, but not necessarily;
- space may consist of several *segments* (i.e., have gaps).

Address translation (a.k.a. *address binding*) means mapping virtual addresses to physical addresses.

Stefan Buettcher
<sbuettch@uwaterloo.ca>

A Simple Address Translation Mechanism

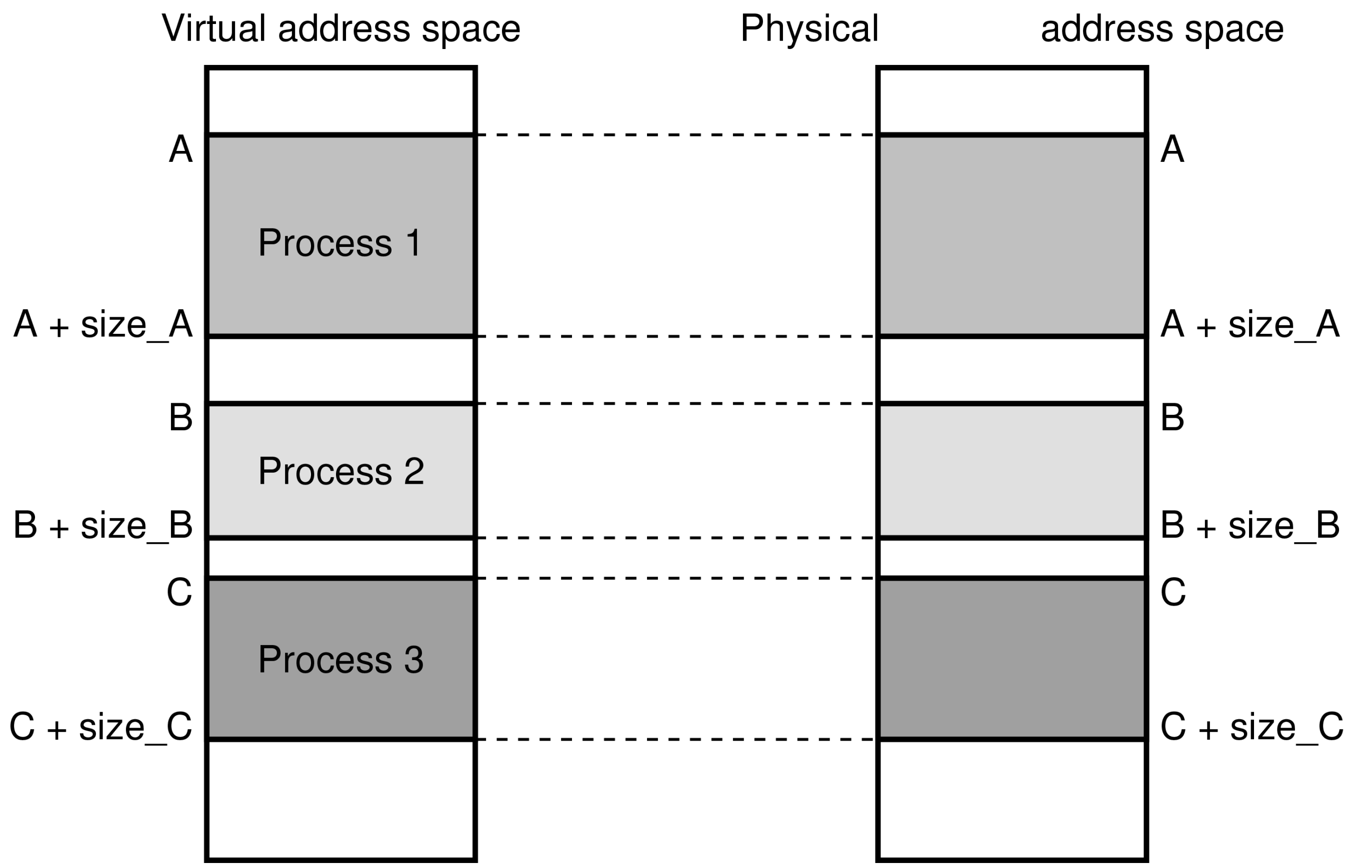
- OS divides physical memory into partitions. Different partitions can have different sizes.

- Each partition can be given to a process as virtual address space.

- Properties:
  - virtual address == physical address;
  - changing the partition a program is loaded into requires recompilation or relocation (if the compiler produces relocatable code);
  - number of processes is limited by the number of partitions size of virtual address space is limited by the size of the partition.

Stefan Buettcher
<sbuettch@uwaterloo.ca>

A Simple Address Translation Mechanism



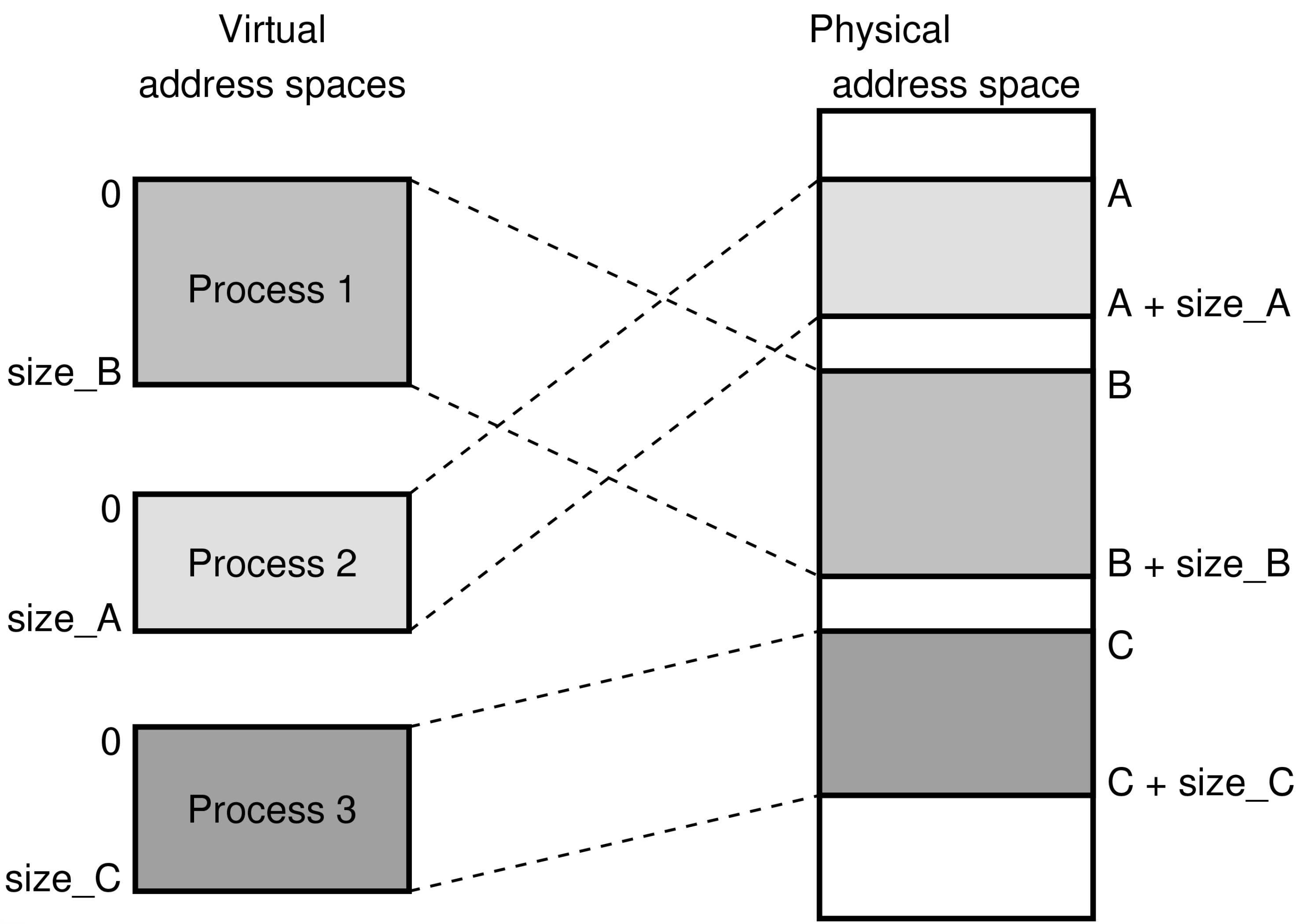
This is really not a good solution!

Stefan Buettcher
<sbuettch@uwaterloo.ca>

Dynamic Relocation

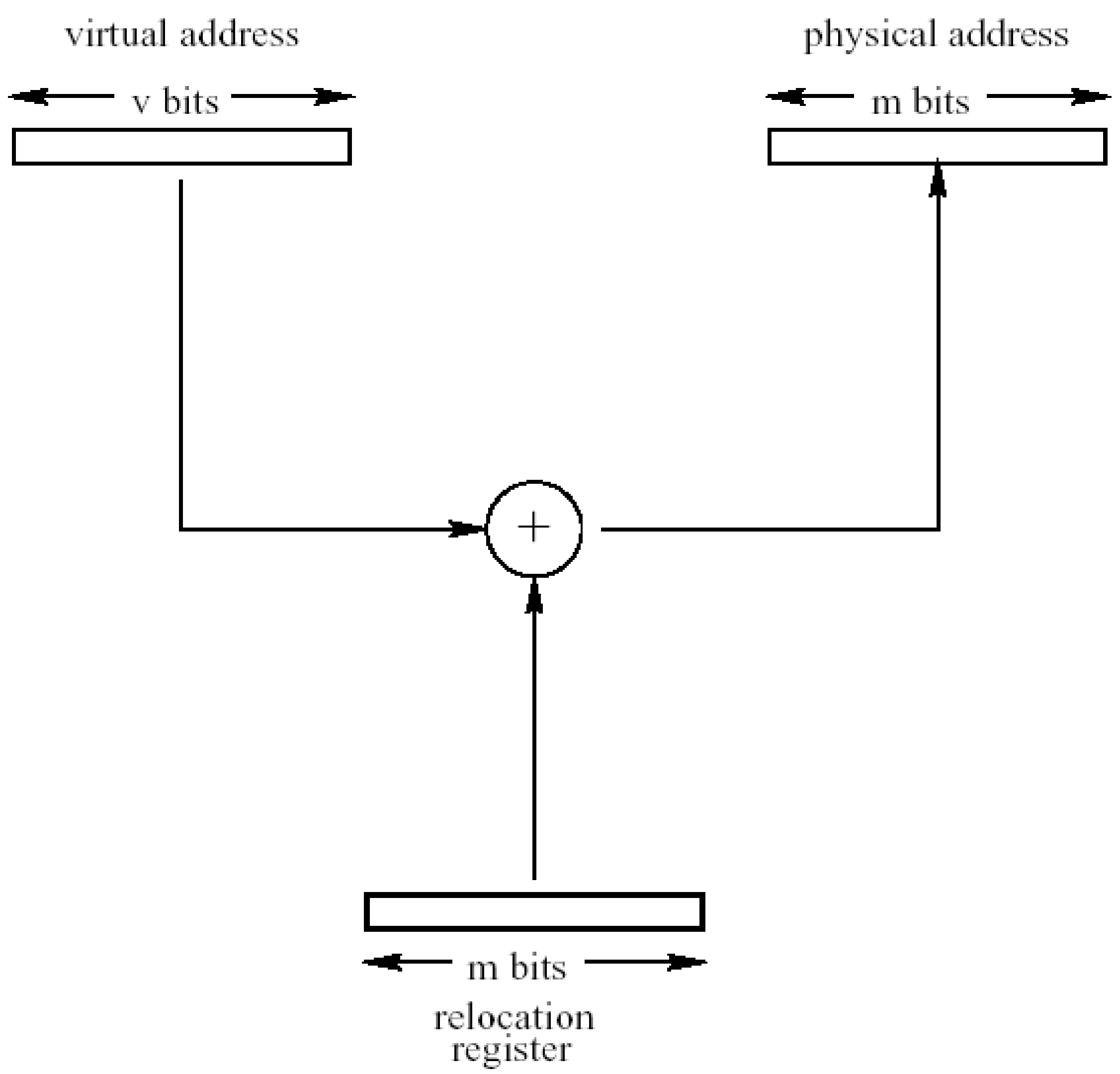
- The memory management unit (MMU) of the CPU contains a relocation register.
- Whenever a thread tries to access a memory location (through a virtual address), the value of the relocation register is added to the virtual memory address – dynamic binding.
- The kernel maintains a separate relocation value for each process (as part of the virtual address space); changes the relocation register at every context switch.
- Properties:
  - all programs can start at virtual address 0;
  - the kernel can relocate a process w/o changing the program;
  - kernel can allocate physical memory dynamically;
  - each virtual address space is still contiguous in physical mem.

Dynamic Relocation



Stefan Buettcher
<sbuettch@uwaterloo.ca>

Dynamic Relocation



Stefan Buettcher
<sbuettch@uwaterloo.ca>

Segmentation

In some systems, a virtual address space can consist of several independent *segments*.

A logical address then consists of two parts:

(segment ID, address within segment)

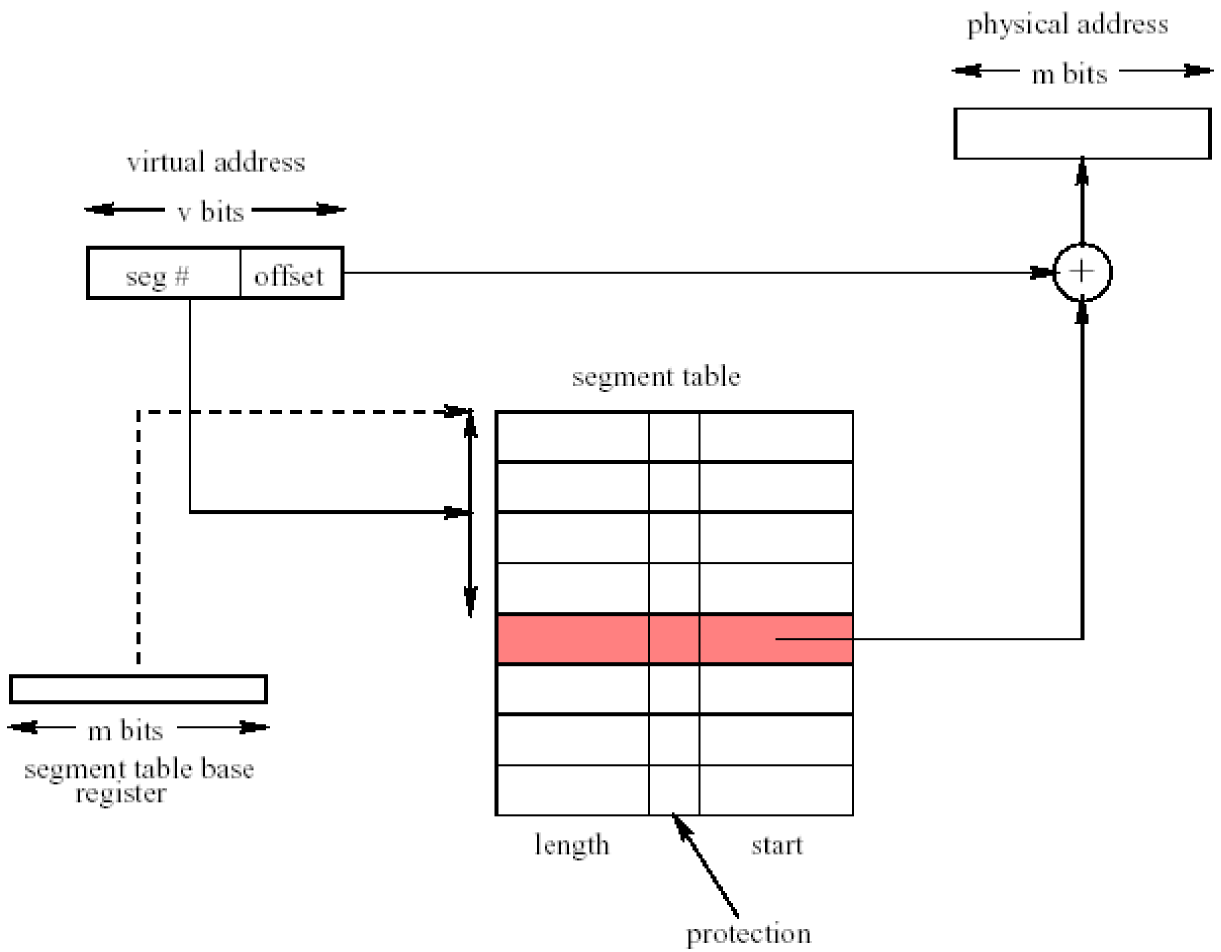
Each segment

- can grow or shrink independently of the other segments in the same address space;
- has its own memory protection attributes.

A process may have separate segments for code, data, stack.

Stefan Buettcher
<sbuettch@uwaterloo.ca>

Segmentation



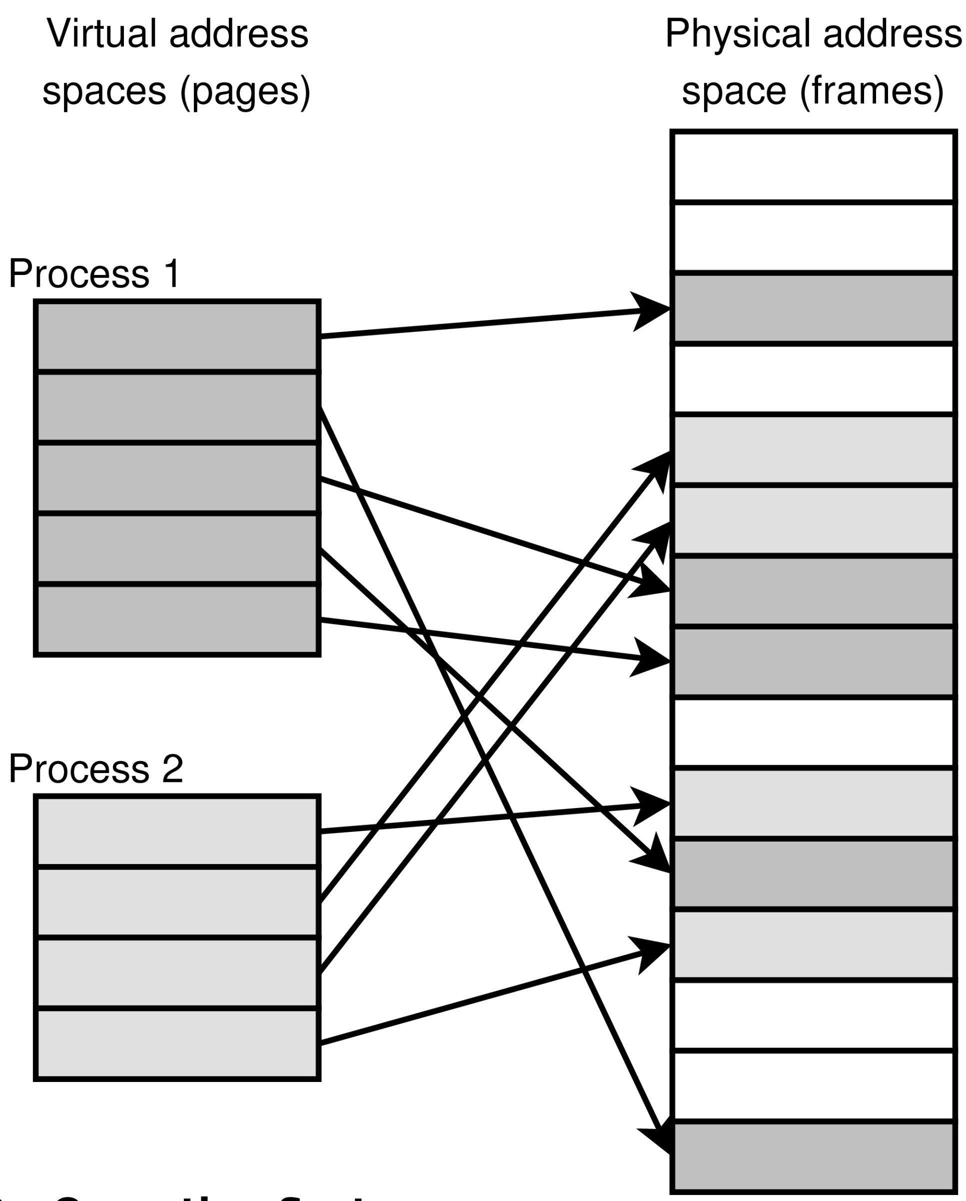
This translation mechanism requires physically contiguous allocation of segments.

Paging

- Each virtual address space is divided into fixed-size chunks called *pages*.
- The physical address space is divided into fixed-size chunks called *frames*.
- Pages have same size as frames.
- The kernel maintains a *page table* (or *page-frame table*) for each process, specifying the frame within which each page is located.
- The CPU's memory management unit (MMU) translates virtual addresses to physical addresses *on-the-fly* for every memory access.
- Properties:
  - relatively simple to implement (in hardware);
  - virtual address space need not be physically contiguous.

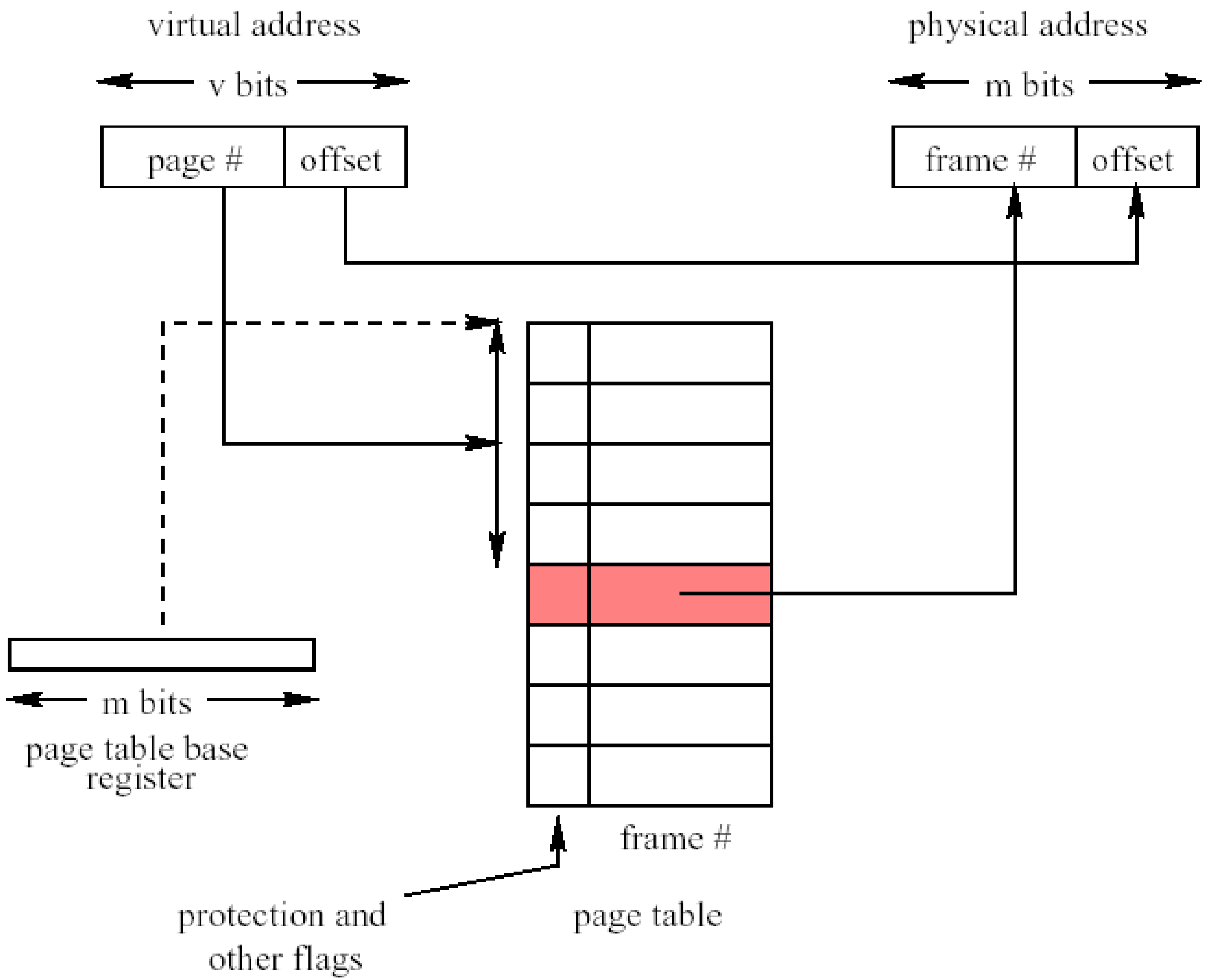
Stefan Buettcher
<sbuettch@uwaterloo.ca>

Paging



Stefan Buettcher
<sbuettch@uwaterloo.ca>

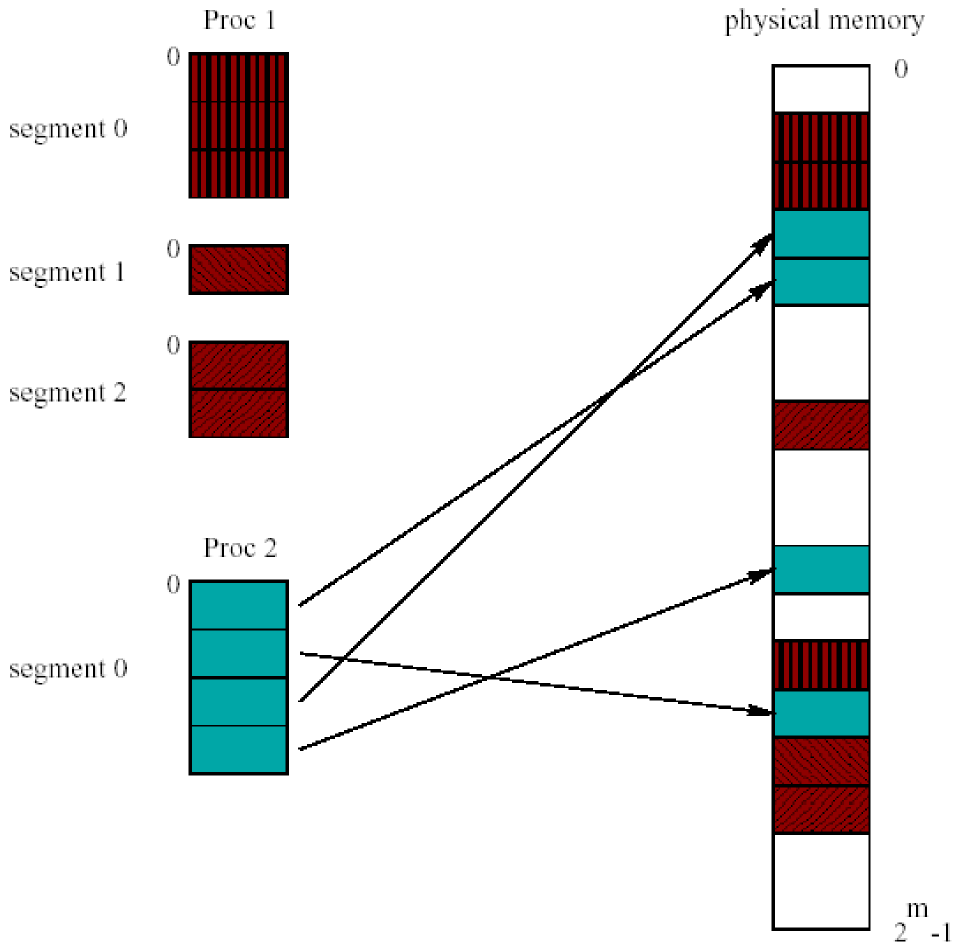
Paging



virtual address
v bits
page #
offset
physical address
m bits
frame #
offset
m bits
page table base
register
frame #
protection and
other flags
page table

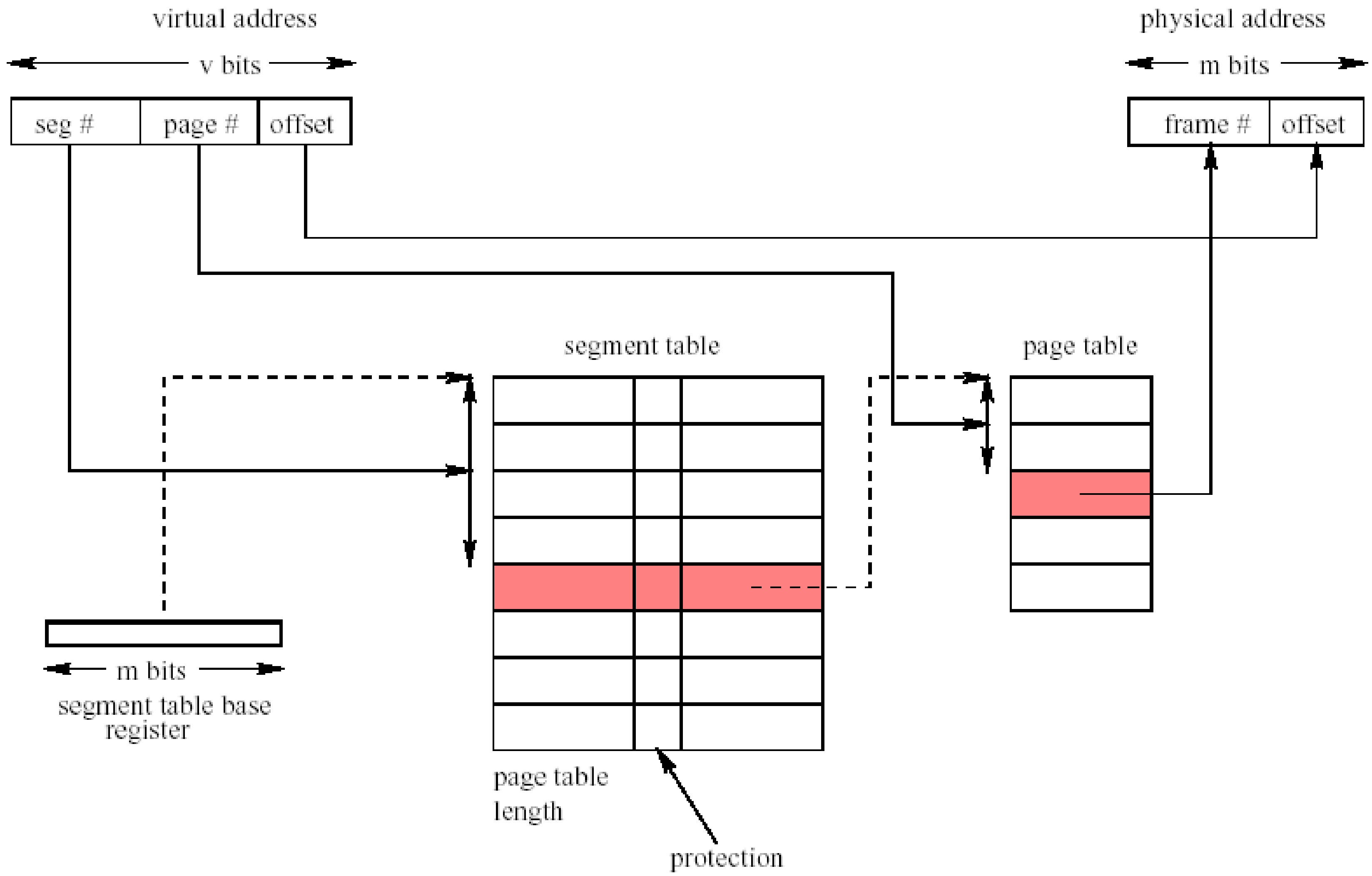
Combining Segmentation and Paging

Segmentation and paging can be combined so that a virtual address space consists of multiple segments, and each segment consists of multiple pages.



Stefan Buettcher
<sbuettch@uwaterloo.ca>

Combining Segmentation and Paging



Stefan Buettcher
<sbuettch@uwaterloo.ca>

Physical Memory Allocation

How to allocate physical memory?

Physical memory can be allocated in different ways.

Variable allocation size:

- always give a process exactly as much memory as it requests
- space tracking and placement are very complex
- placement heuristics are necessary: first fit, best fit, worst fit
- risk of external fragmentation

Fixed allocation size:

- allocate memory in fixed-size chunks
- space tracking and placement are very simple
- risk of internal fragmentation

Stefan Buettcher
<sbuettch@uwaterloo.ca>

Memory Protection

Ensure that each process can only access the physical memory that its virtual memory is bound to.

What if a thread tries to access memory outside its own virtual address space?

MMU limit register is used to check every memory access:

- for simple dynamic relocation, the limit register contains the maximum virtual address of the running process;
- with paging, the limit register contains the maximum page number for the running process.

MMU generates exception when a thread is trying to access a memory address beyond this limit.

(In Nachos: `AddressErrorException`)

Stefan Buettcher
<sbuettch@uwaterloo.ca>

Memory Protection

In addition, access to certain portions of the address space may be restricted:

- read-only memory
- execute-only memory

When paging is used:

- the page table includes flags that define the permitted access modes for each page;
- MMU raises exception when permissions are violated (e.g., thread tries to write to read-only page).

Stefan Buettcher
<sbuettch@uwaterloo.ca>

Memory Management: Roles of OS and MMU

MMU (Memory Management Unit, part of CPU):

- translates virtual addresses to physical addresses;
- checks for protection violations;
- raises exceptions when necessary (e.g., write operation on read-only memory region).

Operating system:

- saves/restores MMU state during context switch (limit register, page tables, ...)
- handles exceptions raised by the MMU
- manages and allocates physical memory

Stefan Buettcher
<sbuettch@uwaterloo.ca>

Address Translation

Executing a single machine instruction may involve one or more memory access operations: One to fetch the instruction; zero or more to fetch the operand(s).

- Simple dynamic relocation with relocation register does not affect the total number of memory operations.
- Address translation through a page table doubles the number of memory operations: Every memory access is preceded by a page table lookup.
  - ⇒ A simple page-table-based address translation scheme can cut the execution speed in half.
  - ⇒ More complex translation schemes might result in an even more severe slowdown.

Solution: Use a cache!

Stefan Buettcher
<sbuettch@uwaterloo.ca>

Translation Lookaside Buffer

- The *Translation Lookaside Buffer* (TLB) is a fast, fully-associative address translation cache in the MMU.
- A TLB hit avoids a memory access due to page table lookup caused by a virtual memory access.
- Each entry in the TLB contains a pair of the form

  (page number, frame number)

  and some additional data, such as protection bits.
- The TLB is on the CPU; a TLB access is much faster than a memory access.
- If the entry for a given page cannot be found in the TLB, the page table has to be queried and an entry in the TLB is replaced.
  - In most systems, this is all done by the MMU; in Nachos, this is done inside the kernel (your code).

Stefan Buettcher
<sbuettch@uwaterloo.ca>

Shared Virtual Memory

Virtual memory allows address spaces to overlap (*shared memory*):

Two or more processes share the same physical memory.

Shared memory:

- allows to use memory more efficiently (e.g., when loading more than one copy of the same program into memory)
- is a mechanism for inter-process communication (IPC).

The unit of sharing can be a page or a segment.

<u>Shared memory in UNIX:</u>
`shmget` (create a new shared memory region or obtain a handle to an existing one); `shmat` (attach to an existing shared mem. Region); `shmdt` (detach), `shmctl` (change attributes, delete)

Stefan Buettcher
<sbuettch@uwaterloo.ca>

Kernel Address Space

There are several possibilities to include the kernel into the bigger memory management picture.

- **Kernel in physical address space –** disable MMU in kernel mode, enable MMU in user mode; to access process data, the kernel must interpret page tables without hardware support; OS must always be in physical memory (*memory-resident*).
- **Kernel in separate virtual address space –** MMU has separate state for user mode and kernel mode; accessing process data is rather difficult; parts of the kernel data may be non-resident.
- **Kernel shares virtual address space with each process –** use memory protection mechanisms to isolate kernel from user processes; accessing process data is trivial; parts of the kernel data may be non-resident.

Stefan Buettcher
<sbuettch@uwaterloo.ca>

Kernel Address Space

Most common solution:
Kernel shares address space with each process.

Disadvantage:
Less space for user space processes (parts of the virtual address space are occupied by the kernel). On 64-bit systems, this is not a problem. On 32-bit systems, it might be.

Under 32-bit Linux, the kernel traditionally gets 1 GB of the total address space; the other 3 GB are for the user process.

When kernel shares address space with user process: Trying to access kernel data does result in protection violation, not in invalid address exception.

Stefan Buettcher
<sbuettch@uwaterloo.ca>