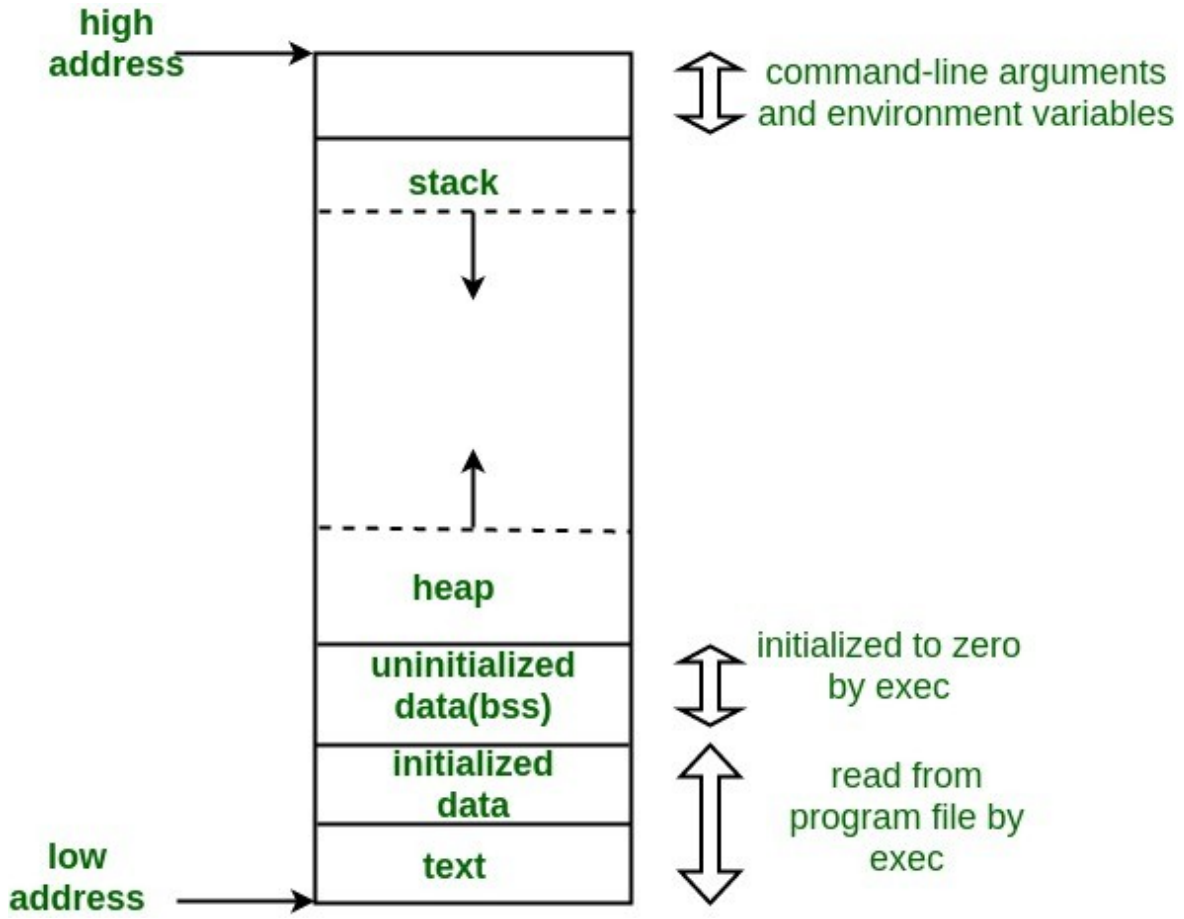
Memory Management and Operating Systems

Dr Sana Belguith

Memory Layout: Refresher

- The stack is the memory set aside as scratch space for a thread of execution. When a function is called, a block is reserved on the top of the stack for local variables and some bookkeeping data. When that function returns, the block becomes unused and can be used the next time a function is called.

- The heap is memory set aside for dynamic allocation. Each thread gets a stack, while there's typically only one heap for the application.



Process as a protection boundary

- OS is responsible for isolating processes from each others.
  - What happened in a process should not affect other processes
- Intra-process communication (between threads) is application responsibility
  - Shared address space.
  - Shared file descriptors.

Physical Memory

- Physical addresses are P bits long
  - Maximum amount of addressable physical memory is 2^P
- OS161's MIPS is 32 bits
  - 2^{32} physical addresses
  - Maximum of 4GB memory
- Modern CPU support large amount of addressable memory
  - X86_64
  - Physical 52 bits
  - Virtual 48 bits

Physical Memory

- Is finite
- Need to be shared between all processes
- Need to be carefully managed to avoid processes stepping on each other toes

Classic OS solution: **hide complexity through an abstraction**

Virtual Memory

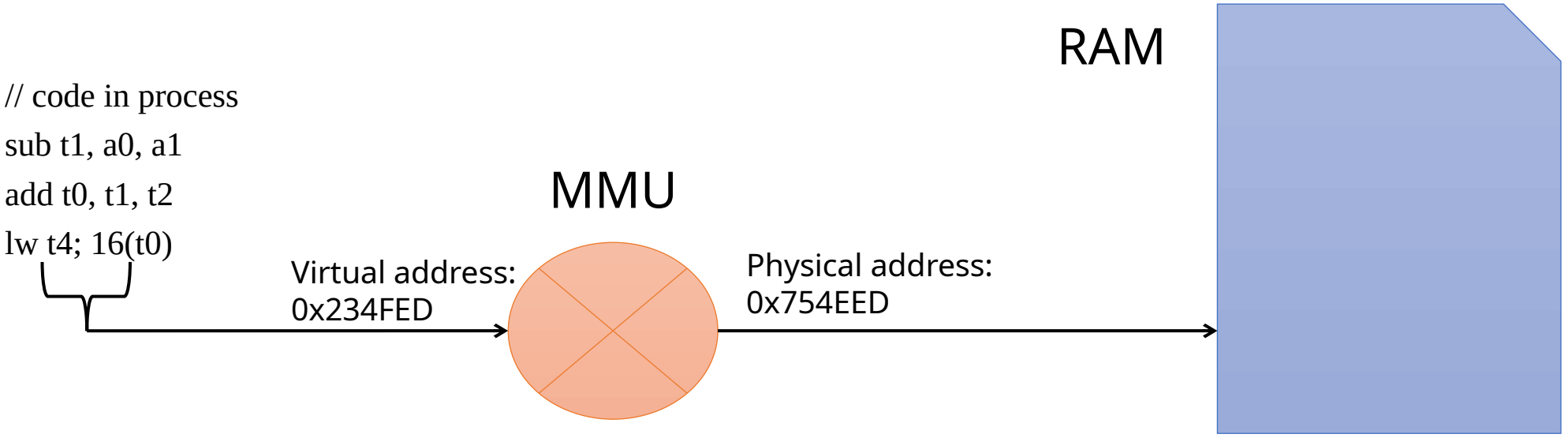
- The kernel provide a virtual memory for each process
- Virtual memory hold code, data and stack(s) for a process
- If virtual memory addresses are V bits
  - Amount of addressable virtual is 2^V
  - On OS161/MIPS V=32
- Running processes see **only** virtual memory
- Each process is **isolated** in its virtual memory and **cannot address** other processes virtual memory

Why virtual memory?

- Isolate processes from each other
- Potentially to support virtual memory larger than physical memory
- Total size of virtual memories can be greater than the physical memory
  - Provide greater support for multiprocessing

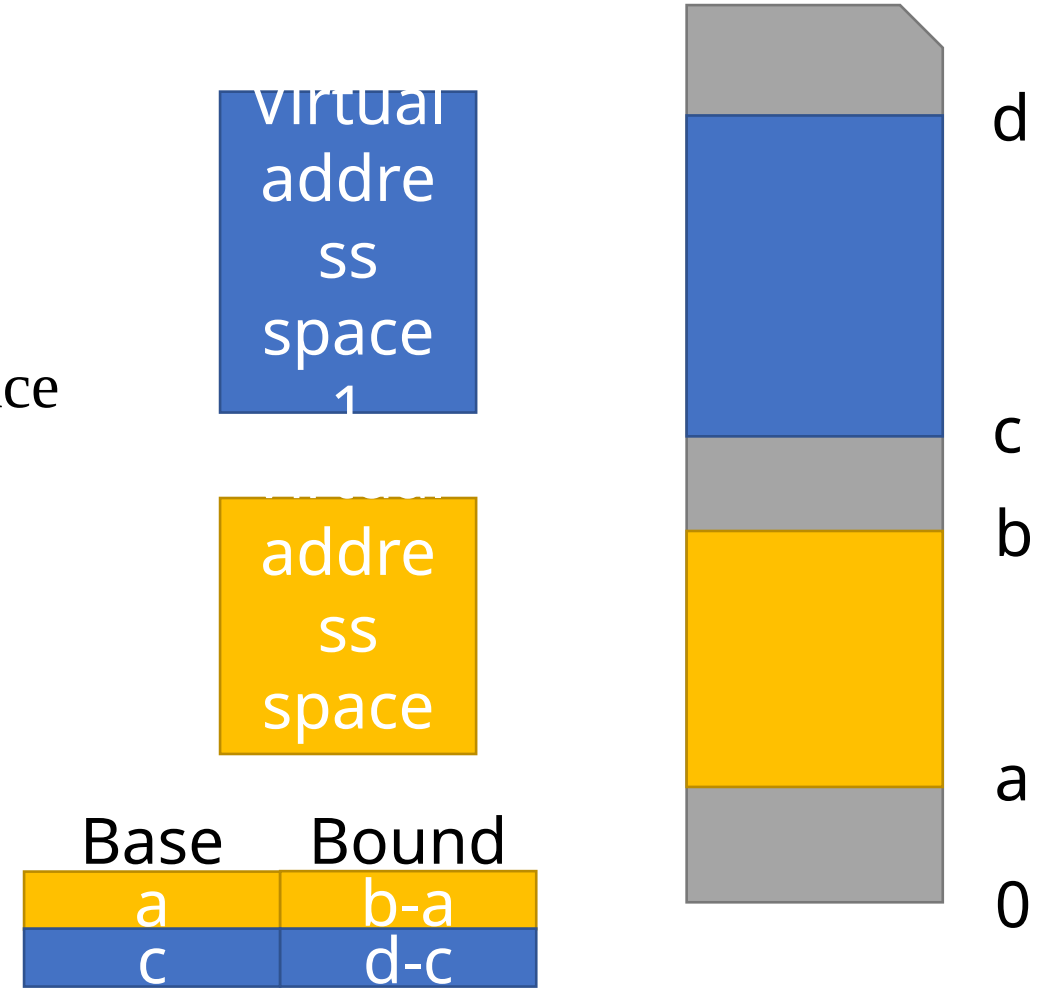
Memory-Mapping Unit (MMU)

- MMU is a piece of hardware
  - Translate virtual addresses to physical addresses
  - Only configurable by a privileged process (i.e. the kernel)
- Virtual addresses are what a process uses
- Physical addresses is what the CPU present to the RAM



Early attempt: base + bound

- Associate virtual address with base and bound register
- Base: where the physical address space start
- Bound: the length of the address space (both virtual and physical)



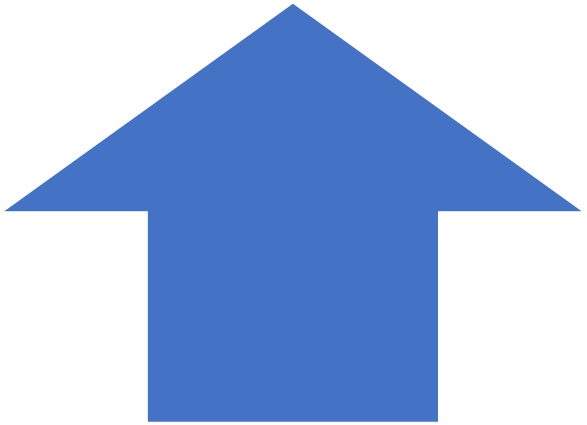
Base + Bound pros and cons

Cons:

- Waste physical memory if the virtual address space is not fully used (i.e. hole between stack and heap)
- Same privilege everywhere read/write/execute
- Sharing memory can only happen by overlapping top and bottom of two spaces (if need to be shared by more than 2?)

Pros:

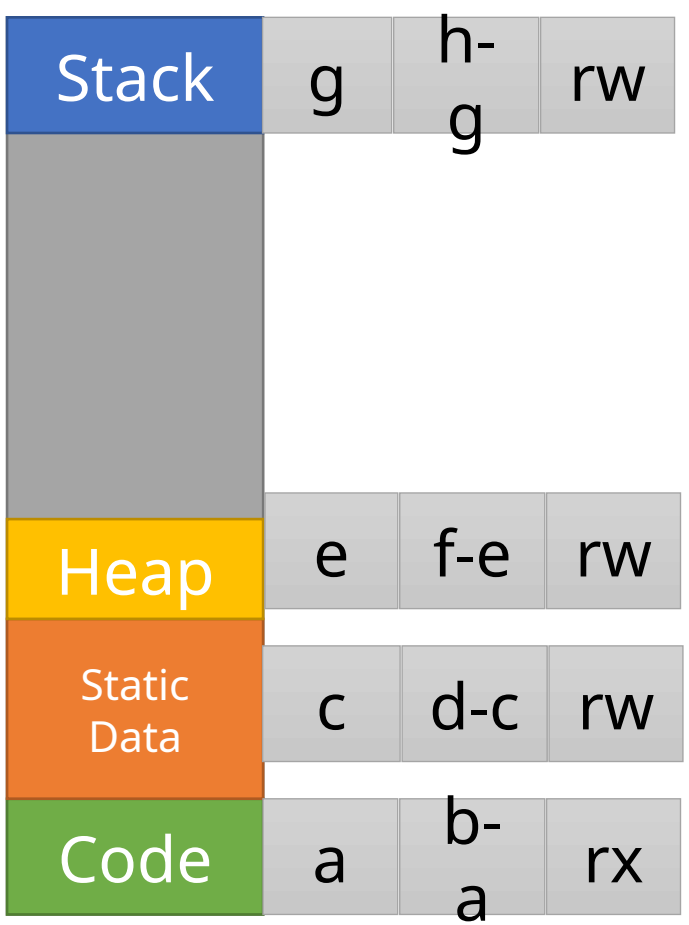
- Allow each virtual address space to be of different size
- Allow each virtual address space to be mapped into any physical RAM of sufficient size
- Straightforward isolation: just ensure no overlap!

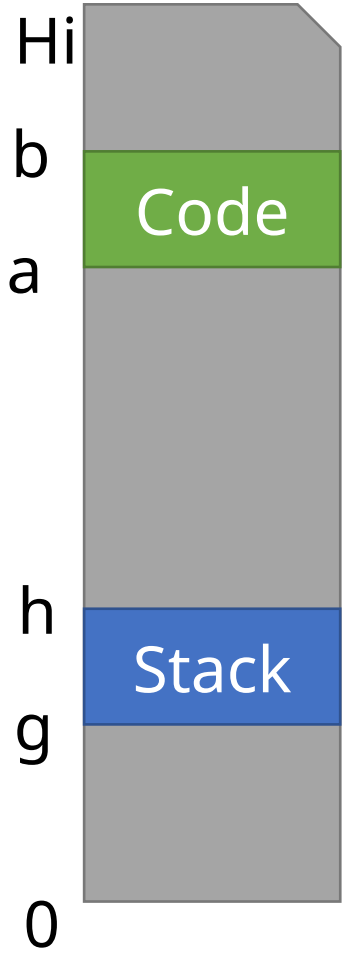


Segmentation

- A single address space has multiple logical segment
  - Code: read/execute, fixed size
  - Static data: read/write, fixed size
  - Heap: read/write, dynamic size
  - Stack: read/write, dynamic size
- Each segment is associated with privilege + base + bound

Segmentation





Segmentation Advantages

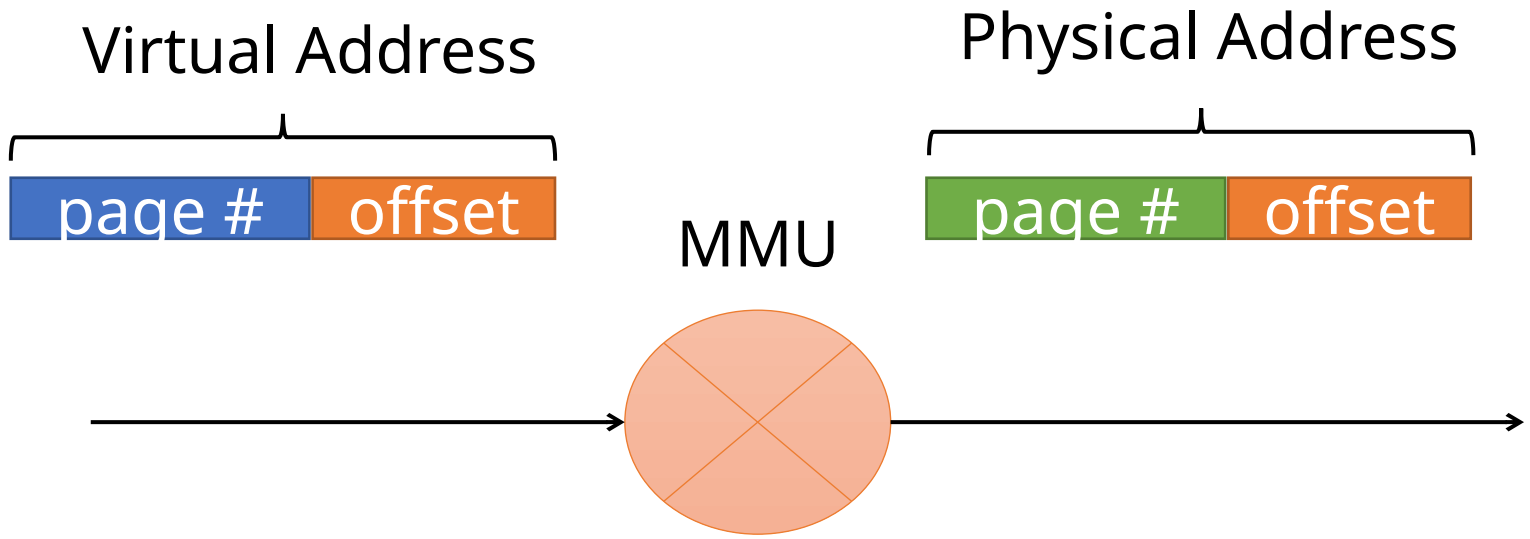
- Shared advantage with base + bound
  - Small address space metadata (few segments, few information about those segments)
  - Isolation is easy just ensure there is no overlap
  - Can map segment in any large enough region of physical RAM
- Advantage over base + bound
  - Can share memory at the segment granularity
  - Waste less memory (i.e. hole between heap and stack doesn’t need to be mapped)
  - Enables segment granularity memory protection

Segmentation Disadvantages

- Segment may be large
  - Need to map the whole segment into memory even to access a single byte
  - Cannot map only the part of the segment that is utilized
- Need to find free physical memory large enough to accommodate a segment

Paging

- Makes allocation problem trivial
  - Fixed sized units called pages
  - No more bounds !
- Use space efficiently
  - Small fixed size (no need to use large chunk of memory to access a single byte)
  - No more segment, address is divided in a collection of pages



Paging Pros and Cons

- Good
  - Can allocate virtual address space with fine granularity
  - Only need to bring small pages that the process needs into the RAM
- Bad
  - Bookeeping becomes more complex
  - Lots of small pages to keep track of

Swapping pages out

- Physical RAM may be oversubscribed
  - Total virtual pages greater than the number of physical pages
- Swapping is moving virtual pages from physical RAM to a swap device
  - SSD
  - Hard Drive
  - etc.

Page Faults

- When process try to access page not in memory
  - **MMU** detect this and raise an exception
- Attempting to access a page not in RAM is a **page fault**
- The kernel job on page fault is to:
  - Swap the page from secondary storage to memory, evicting another page if necessary
  - Return from the exception so the application can try again

Page Faults are slow!

- Accessing secondary storage is slow
  - Millisecond for harddrive
  - Microsecond for SSD
  - … comparing to nanoseconds for RAM
- Suppose secondary storage is 1000 time slower
  - 1 in 10 access results in page fault -> Average access 100 times slower
  - 1 in 100 access results in page fault -> Average access 10 times slower
  - 1 in 1000 access results in page fault -> Average access 2 times slower
- Goal is to reduce occurrence of page faults
  - Limit the number of processes, so that there is enough RAM
  - Hide latencies by prefetching a page before a process needs them
  - Be clever about which page is kept in physical memory and which page is evicted

Simplest replacement policy: FIFO

- What page to evict?
- First In First Out (FIFO): remove the page that has been in memory the longest

Num	1	2	3	4	5	6	7	8	9
Refs	a	b	c	d	a	b	e	a	b
PP1	a	a	a	d	d	d	e	e	e
PP2		b	b	b	a	a	a	a	a
PP3			c	c	c	b	b	b	b
Fault ?	x	x	x	x	x	x	x		

Optimum replacement policy: MIN

- What page to evict?
- MIN: replace the page that will not be referenced for the longest

Num	1	2	3	4	5	6	7	8	9
Refs	a	b	c	d	a	b	e	a	b
PP1	a	a	a	a	a	a	a	a	a
PP2		b	b	b	b	b	b	b	b
PP3			c	d	d	d	e	e	e
Fault ?	x	x	x	x			x		

Least Recently Used (LRU) replacement policy

- What page to evict?
- LRU : remove the page that has been used the least recently (temporal locality)

Num	1	2	3	4	5	6	7	8	9
Refs	a	b	c	d	a	b	e	a	b
PP1	a	a	a	d	d	d	e	e	e
PP2		b	b	b	a	a	a	a	a
PP3			c	c	c	b	b	b	b
Fault ?	x	x	x	x	x	x	x		

Practical replacement policy: Clock

- What page to evict?
- Add a “used” bit to PTE
  - Set by MMU when page accessed
  - Can be cleared by kernel

victim = 0

while use bit of victim is set

    clear use bit of victim

    victim = (victim + 1) % num_frames

evict victim

Practical replacement policy: Clock

Num	1	2	3	4	5	6	7	8	9
Refs	a	b	c	d	a	b	e	a	b
PP1	A	A	A	D	d	d	e	e	e
PP2		B	B	B	a	a	a	a	a
PP3			C	C	c	b	b	b	b
Fault ?	x	x	x	x	x	x	x		