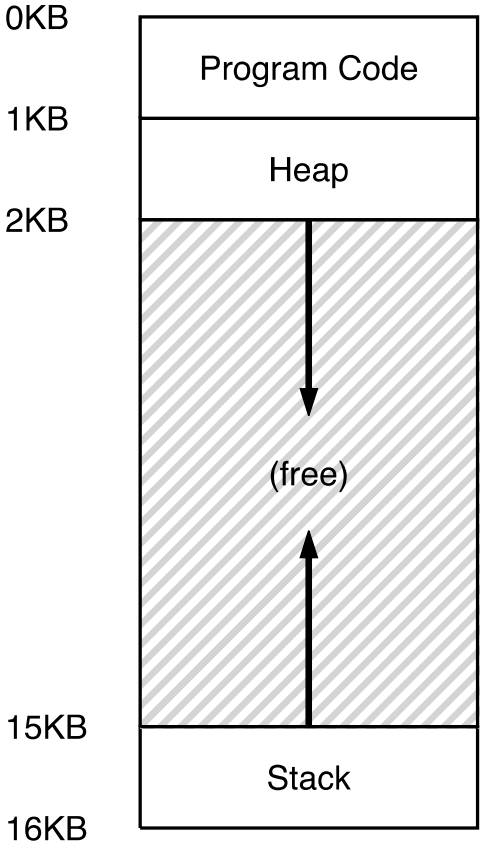
# MEMORY MANAGEMENT: SEGMENTATION AND PAGING

Kai Mast

CS 537

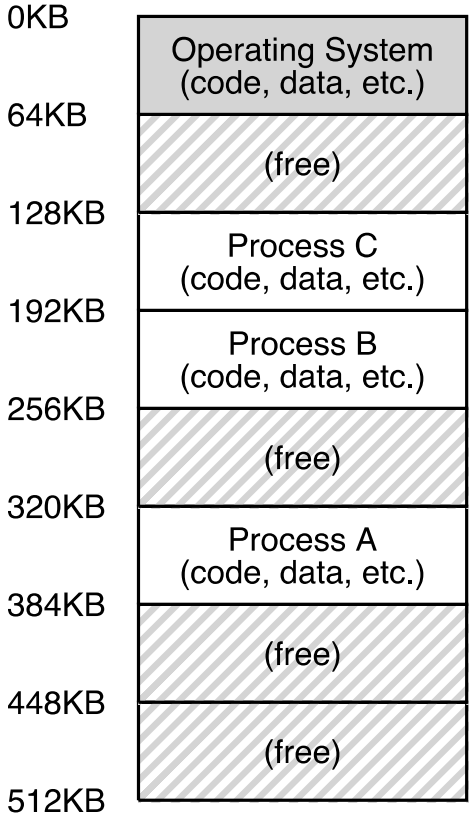
Fall 2022

# RECAP: VIRTUAL MEMORY

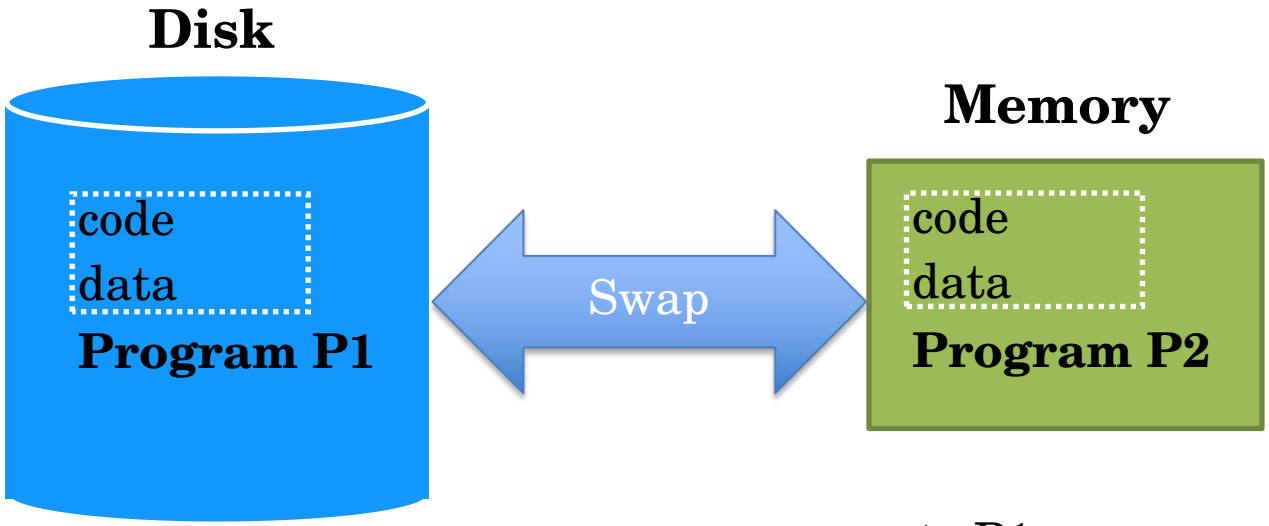



Each process has own set of addresses

How can the OS provide illusion of private (virtual) address space to each process?



# RECAP: TIME-SHARE MEMORY




- create P1
- swap out P1
- create P2
- swap out P2
- swap in P1
- ...

# RECAP: STATIC RELOCATION

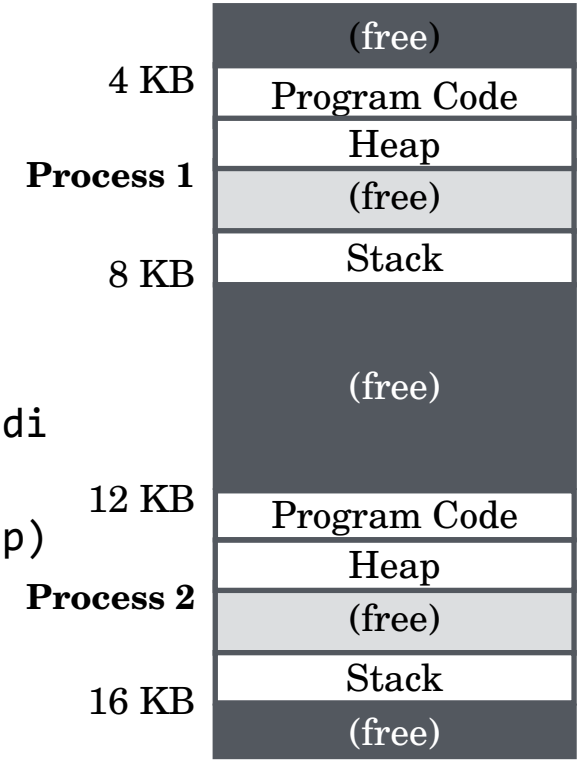
```
0x10:movl 0x8(%rbp), %edi
0x13:addl $0x3, %edi
0x19:movl %edi, 0x8(%rbp)
```

**Rewrite for P1**

```
0x1010:  movl 0x8(%rbp), %edi
0x1013:  addl $0x3, %edi
0x1019:  movl %edi, 0x8(%rbp)
```

**Rewrite for P2**

```
0x3010: movl 0x8(%rbp), %edi
0x3013: addl $0x3, %edi
0x3019: movl %edi, 0x8(%rbp)
```



# DYNAMIC RELOCATION USING BASE REGISTERS

**Base Registers**

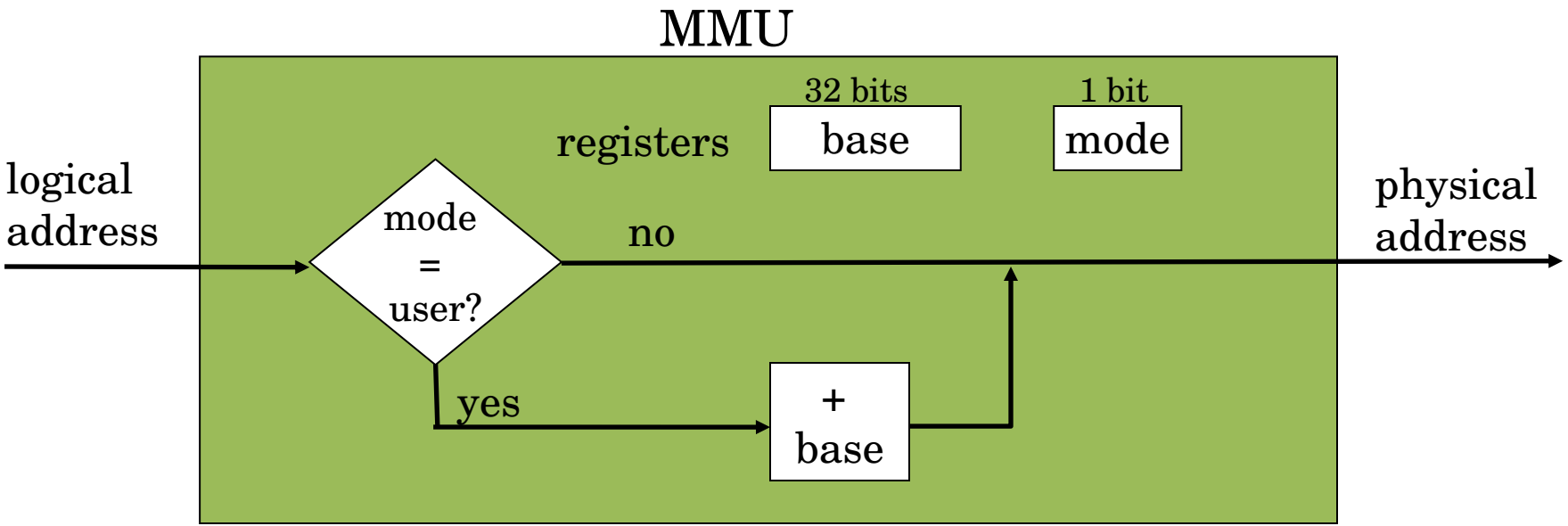
- Stored in the MMU
- Each process has different value in base register
- Set by the OS on context switch

**Address Translation using Base Registers**

- Add base register to virtual address on every memory access
- Dynamic relocation by changing value of base register!

# BASE REGISTERS: IMPLEMENTATION

- Translation on every memory access of user process
- MMU adds base register to logical address to form physical address



# 4) DYNAMIC RELOCATION WITH BASE+BOUNDS

**Idea:** Limit the address space with a bounds register

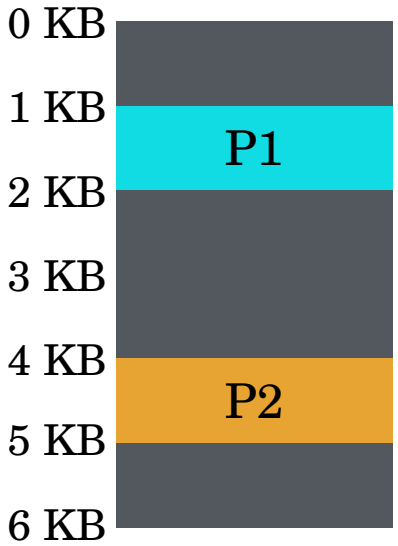
**Base register:** Smallest physical address (or starting location)
**Bounds register:** Size of this process's virtual address space

- Sometimes defined as largest physical address (base + size)

If process loads/stores beyond bounds, what does the OS do? Kill process

What performs the bounds check? MMU

# PROTECTION WITH BASE + BOUNDS




| Process | Base | Bounds |
|---|---|---|
| P1 | 1024 | 1024 |
| P2 | 4096 | 1024 |

**Possible Execution**

| Virtual | Physical |
|---|---|
| P1: load 100, R1 | load 1124, R1 |
| P2: load 100, R1 | load 4196, R1 |
| P2: load 1000, R1 | load 5096, R1 |
| P1: load 1000, R1 | load 2024, R1 |
| P1: store 3072, R1 | **kill P1** |

# MANAGING PROCESSES WITH BASE AND BOUNDS

Track base and bounds registers in PCB (process-control-block)

**Context-Switch Steps**

- OS in privileged mode
- Save base and bounds registers of old process
- Load base and bounds registers of new process
- Change to user mode and jump to new process

What if we do not change base and bounds registers during context switch?

Threads!

**Protection Requirements**

- User process cannot switch to privileged mode (except using traps)
- User process cannot change base and bounds registers

# BASE AND BOUNDS SUMMARY

**Advantages**

- Provides protection (both read and write) across address spaces
- Supports dynamic relocation
- Can place process at different locations initially and move address spaces
- Simple, inexpensive implementation:
  - Few registers, little logic in MMU
- Fast: MMU can add and compare in parallel

**Disadvantages**

- Each process must be allocated contiguously in physical memory
- Must reserve memory that may not be used by process
- No partial sharing between processes
  - Cannot share limited parts of address space

# 5) SEGMENTATION

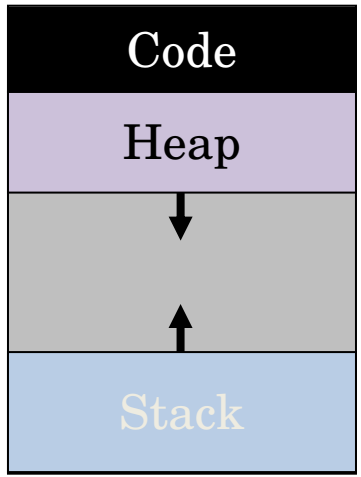
Divide logical address space into logical segments – visible to the OS

- Each segment corresponds to logical entity in address space (code, stack, heap)

Each segment has separate base + bounds register in the MMU

**Advantages:** Each segment can independently

- be placed separately in physical memory
- grow and shrink
- be protected (separate read/write/execute bits)




(Book Chapter 16)

# SEGMENTED ADDRESSING

For virtual address, process specifies 1) segment and
2) offset within segment

How does process designate a particular segment?

- Use part of logical address
  - Top bits of logical address select segment
  - Low bits of logical address select offset within segment

# SEGMENTATION IMPLEMENTATION

MMU contains Segment Table for the current process:

- Each segment has own base and bounds, protection bits
- Example: 14 bit logical address, 4 segments;

How many bits for the segment?

How many bits for the offset?

| Segment | Base | Bounds | R W |
|---|---|---|---|
| 0 | 0x2000 | 0x6ff | 1 0 |
| 1 | 0x0000 | 0x4ff | 1 1 |
| 2 | 0x3000 | 0xfff | 1 1 |
| 3 | 0x0000 | 0x000 | 0 0 |

**Remember:**
A hex digit (4 bits) has 16 possible values

# ADDRESS TRANSLATIONS WITH SEGMENTATION

14 bit logical address, 4 segments;

| Segment | Base | Bounds | R W |
|---|---|---|---|
| 0 | 0x2000 | 0x6ff | 1 0 |
| 1 | 0x0000 | 0x4ff | 1 1 |
| 2 | 0x3000 | 0xfff | 1 1 |
| 3 | 0x0000 | 0x000 | 0 0 |

Remember:
1 hex digit  4 bits

Translate logical (in hex) to physical

0x0240:

0x1108:

0x265c:

0x3002:

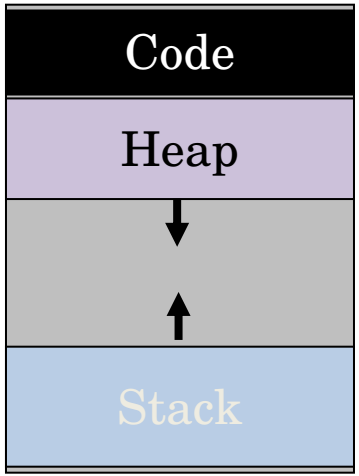
# ADVANTAGES OF SEGMENTATION

Enables sparse allocation of address space

Stack and heap can grow independently

- Heap: If no data on free list, dynamic memory allocator requests more from OS

  (e.g., UNIX: malloc library makes sbrk() system call)
- Stack: OS recognizes reference near outside legal segment, extends stack implicitly




Different protection for different segments

- Enables sharing of selected segments
- Read-only status for code

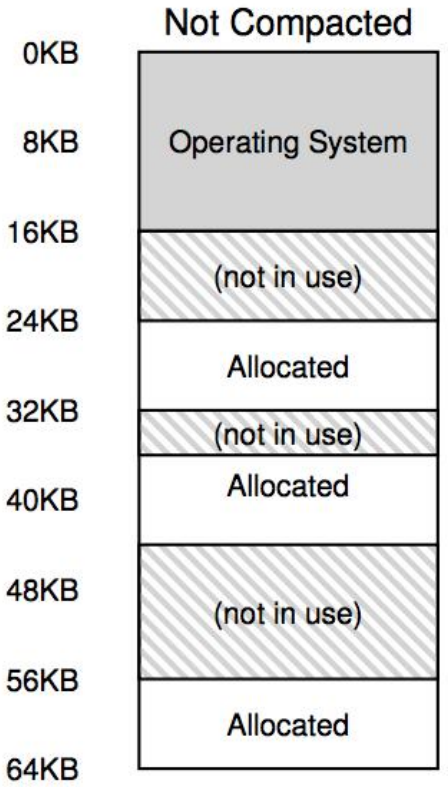
Supports dynamic relocation of each segment

# DISADVANTAGES OF SEGMENTATION

Each segment must be allocated contiguously

May not have sufficient physical memory for large segments?

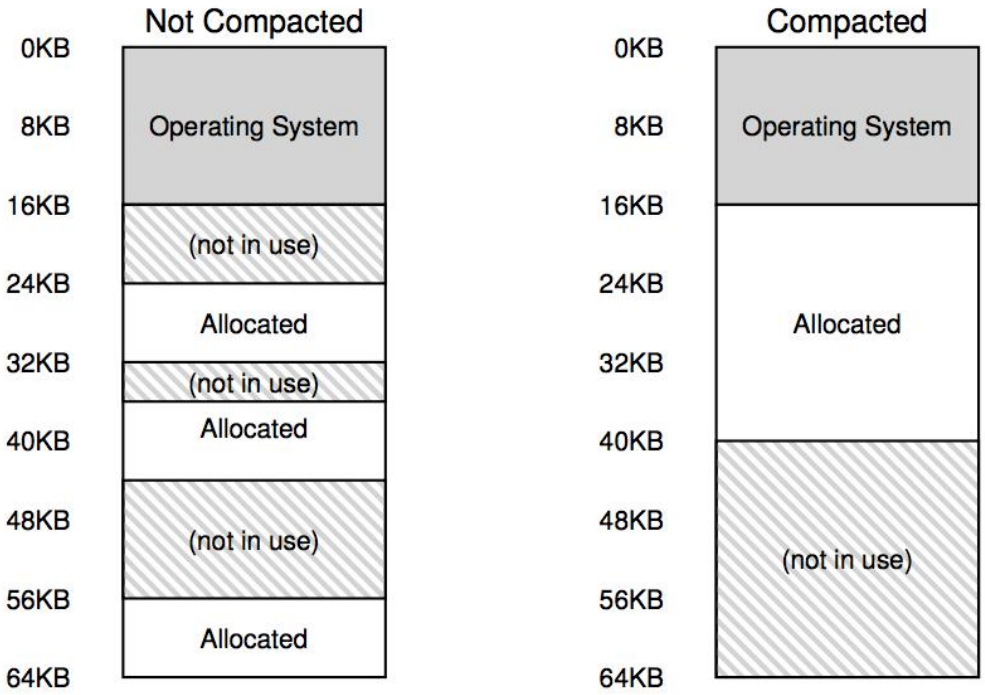
External Fragmentation
(memory fragmentation caused by the OS)



# EXTERNAL FRAGMENTATION

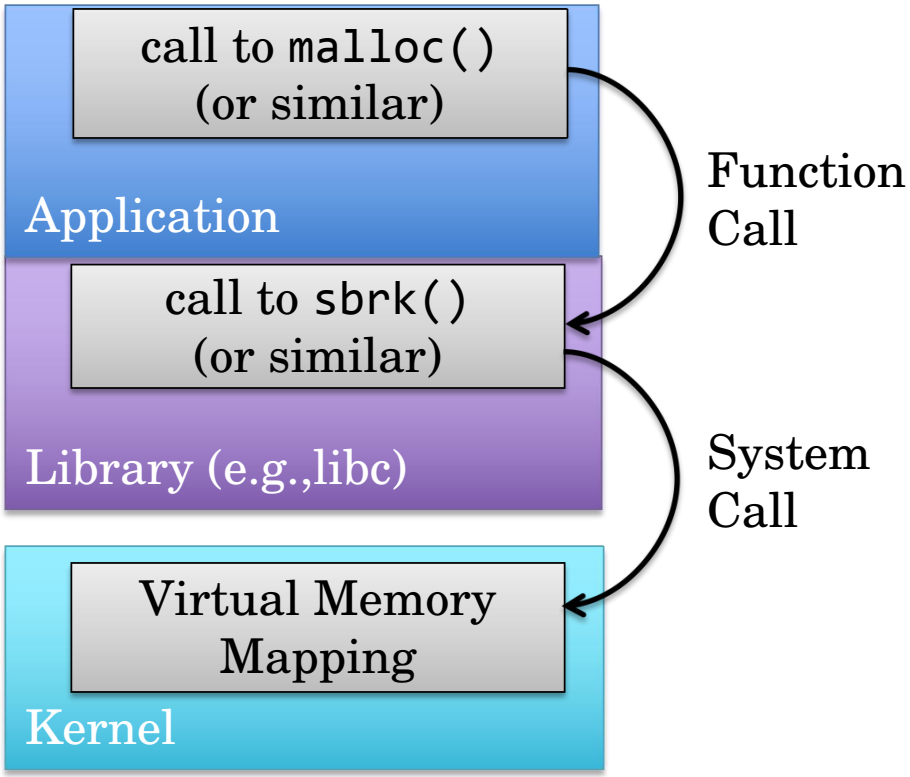
**Why is this bad?**

Small pieces of free memory might not be usable/assignable



# INTERNAL FRAGMENTATION

**What happens when you allocate memory?**




**Why is this additional layer needed?**

- System calls are expensive
- VM is allocated at a coarse granularity (when using pages)

**Why can the heap get fragmented?**

- Allocation patterns are hard to predict
- `malloc()` may be called with vastly different sizes

# REVIEW: MATCH DESCRIPTION

| Description | Name of Approach |
|---|---|
| One process uses RAM at a time | Time Sharing |
| Rewrite code and addresses before running | Static Relocation |
| Add per-process starting location to virtual address to obtain phys address | Dynamic w/ Base |
| Dynamic approach that verifies address is in valid range | Dynamic w/ Base+Bounds |
| Several base+bound pairs per process | Segmentation |

**Candidates:** Segmentation, Static Relocation, Base, Base+Bounds, Time Sharing

# MEMORY MANAGEMENT WITH PAGING

(Book Chapter 18)

# PAGING

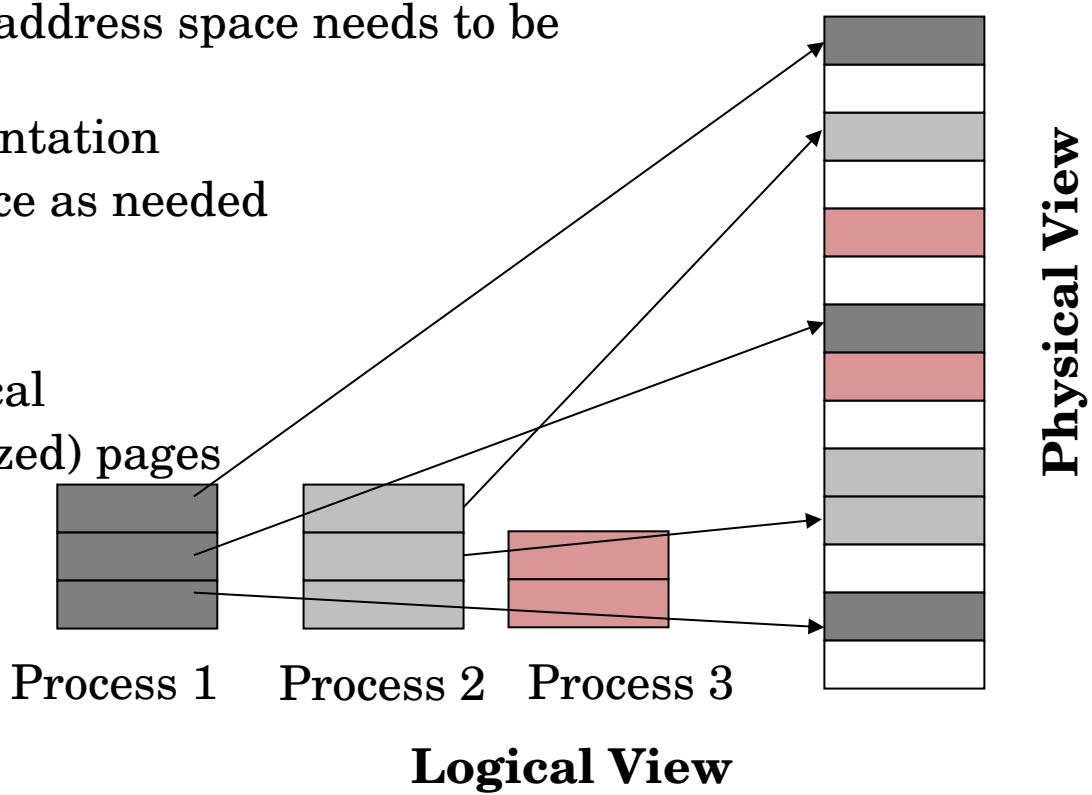
**Goal:** Remove requirement that address space needs to be contiguous

- Eliminates external fragmentation
- Ability to grow address space as needed

**Idea:**

Divide address spaces and physical memory into fixed-sized (same sized) pages

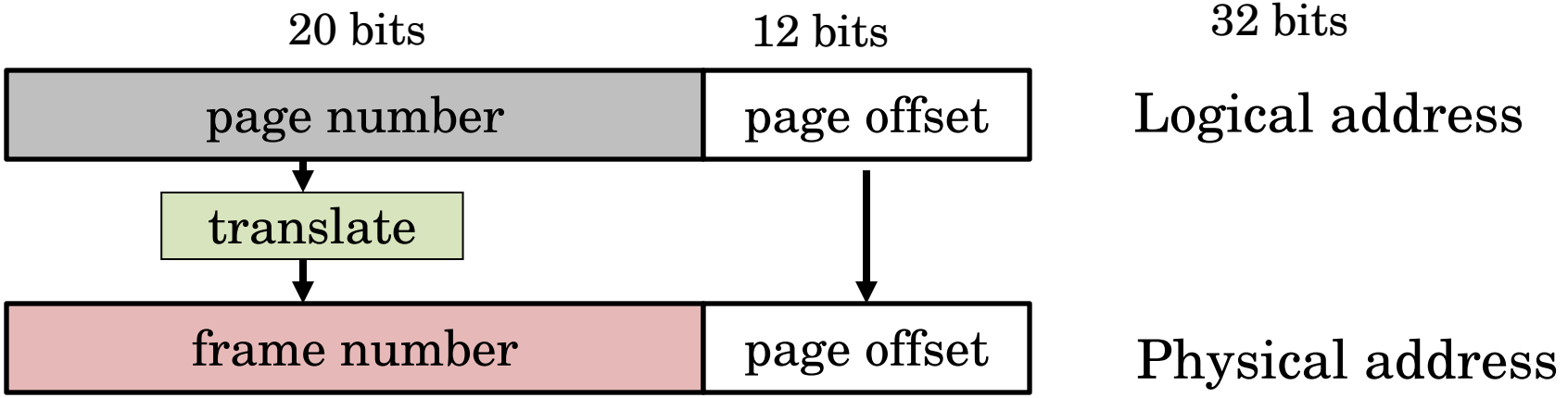
Size: $2^n$, Example: 4KB



# TRANSLATION OF PAGE ADDRESSES

How to translate logical address to physical address?

- High-order bits of address designate page number
- Low-order bits of address designate offset within page




No addition needed (unlike segmentation); just append bits correctly…