We acknowledge and pay our respects to the Kaurna people,
the traditional custodians whose ancestral lands we gather on.

We acknowledge the deep feelings of attachment and relationship of the Kaurna people to country and we respect and value their past, present and ongoing connection to the land and cultural beliefs.

COMP SCI 3004 Operating Systems

Week 3 – Segmentation & Paging

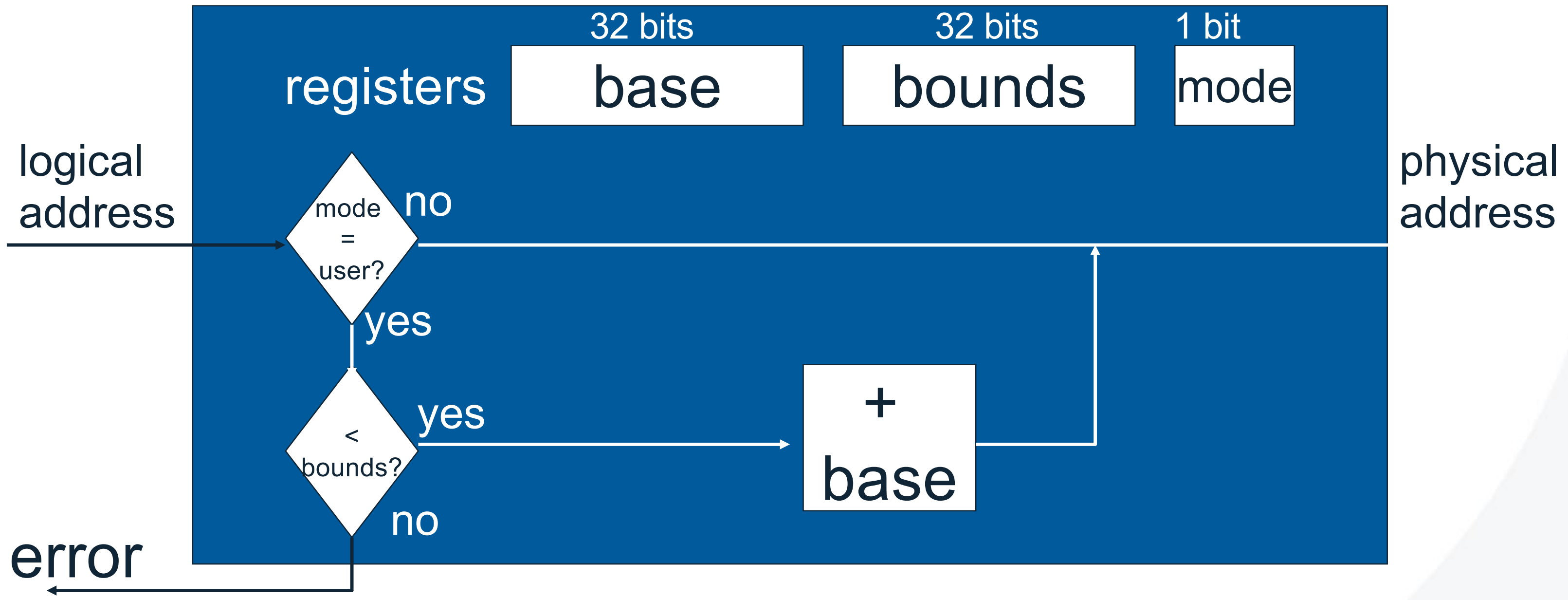
make history.

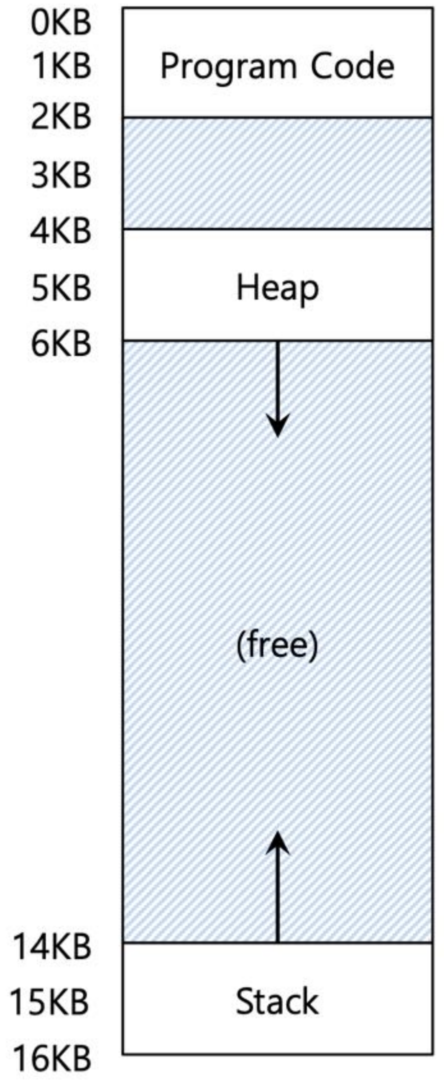
THE UNIVERSITY of ADELAIDE

Recall: Base+Bounds

- **Idea**
  - limit the address space with a bounds register
- **Base register**
  - smallest physical addr (or starting location)
- **Bounds register**
  - size of this process's virtual address space
  - Sometimes defined as largest physical address (base + size)
- **OS kills process if process loads/stores beyond bounds**

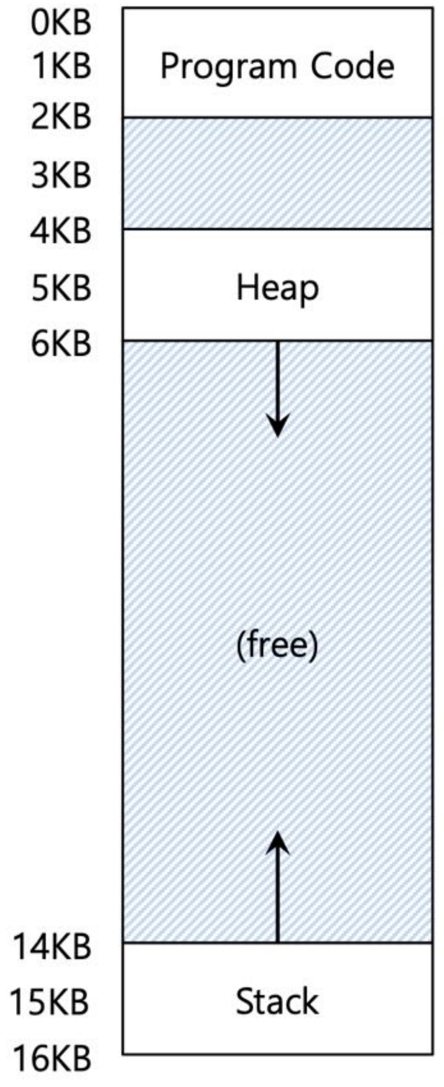
Recall: Base+Bounds





Inefficiency of the Base and Bound Approach

- Big chunk of “free” space
- “free” space takes up physical memory.
- Hard to run when an address space does not fit into physical memory

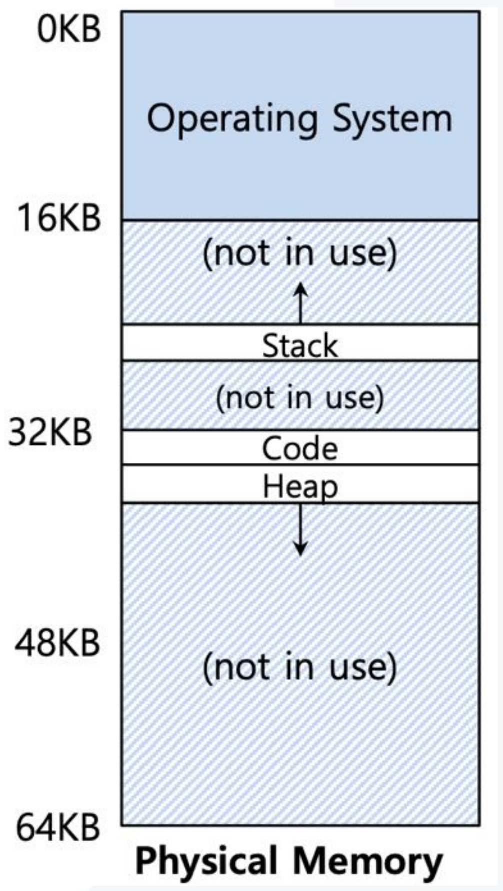


Segmentation

- **Divide address space into logical segments**
  - Each segment corresponds to logical entity in address space
    - code, stack, heap
- **Each segment can independently:**
  - be placed separately in physical memory
  - grow and shrink (separate base+bounds)
  - be protected (separate read/write/execute protection bits)

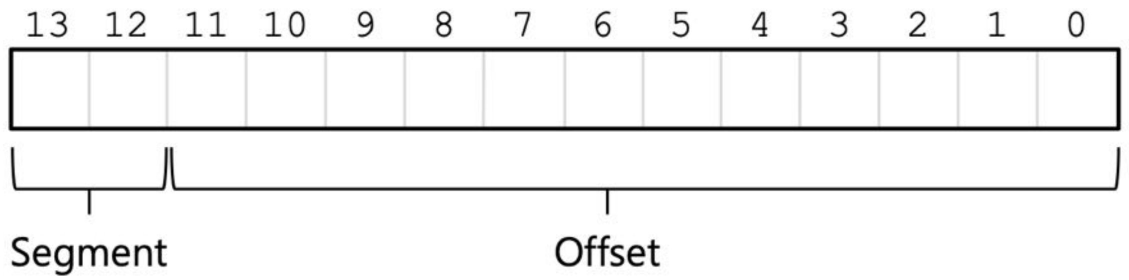
Segmented Addressing

- **Process now specifies segment and 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
- **What if small address space, not enough bits?**
  - Implicitly by type of memory reference
  - Special registers



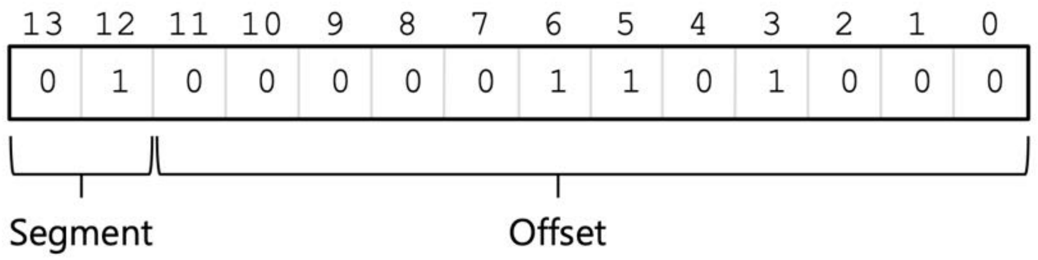
Referring to Segment: Explicit approach

- Chop up the address space into segments based on the top few bits of virtual address.

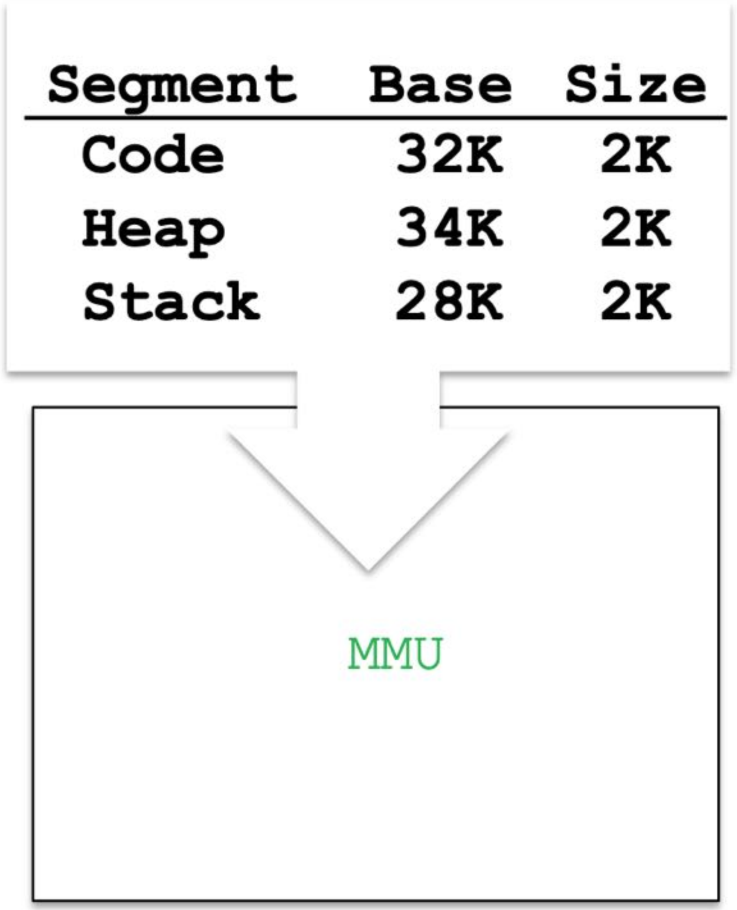


- Example: virtual address 4200 (01000001101000)

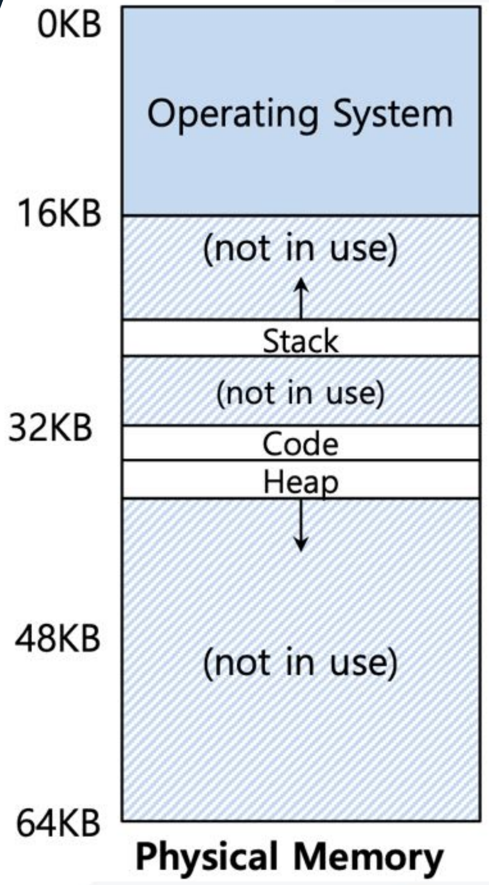
Segment	bits
Code	00
Heap	01
Stack	10
-	11



Placing Segment In Physical Memory

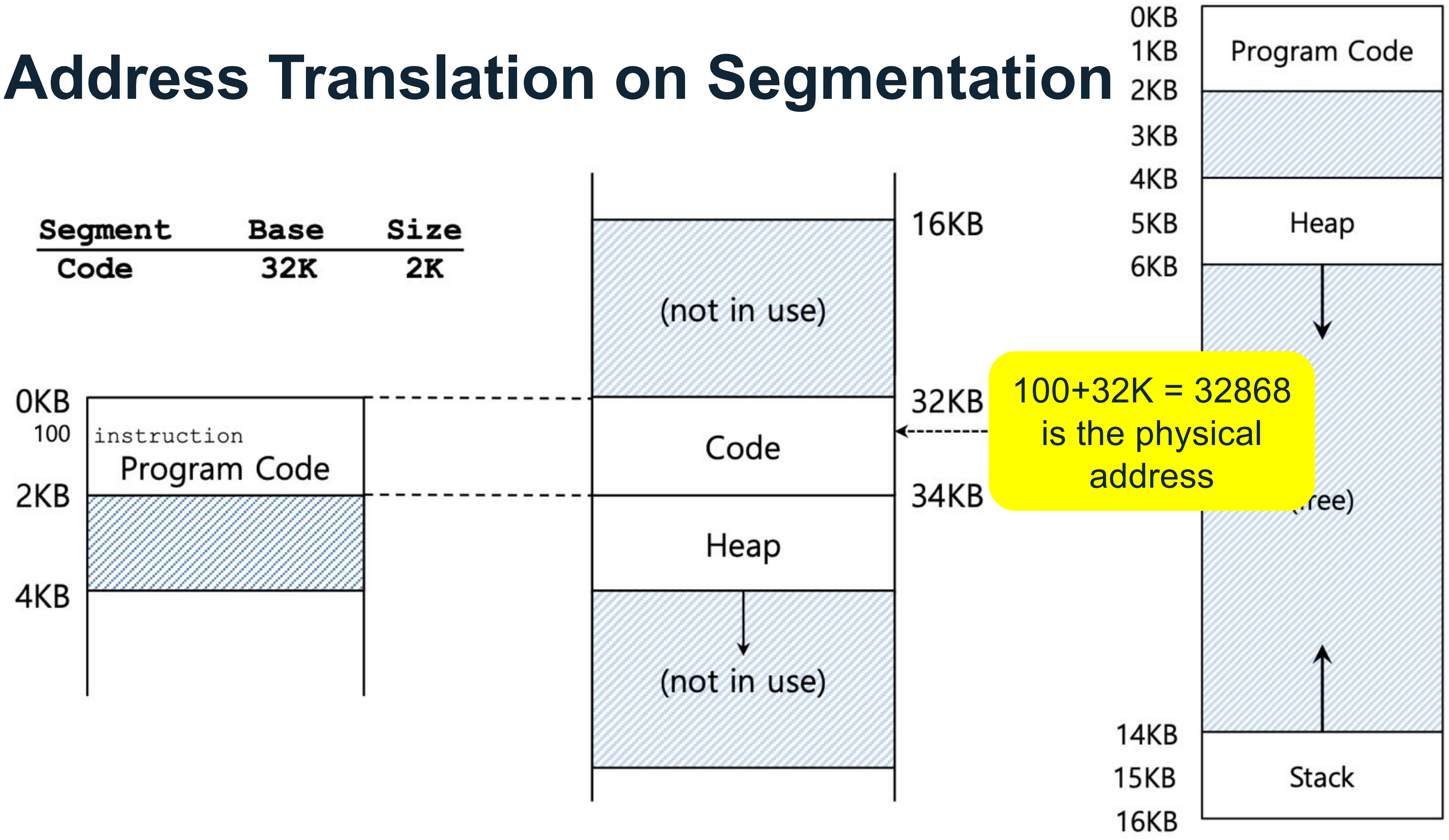


Segment	Base	Size
Code	32K	2K
Heap	34K	2K
Stack	28K	2K



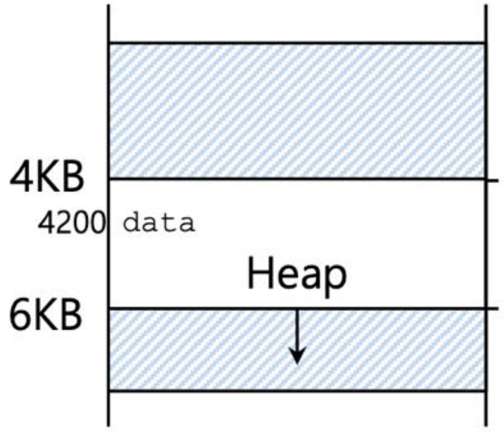
Address Translation on Segmentation

Segment	Base	Size
Code	32K	2K

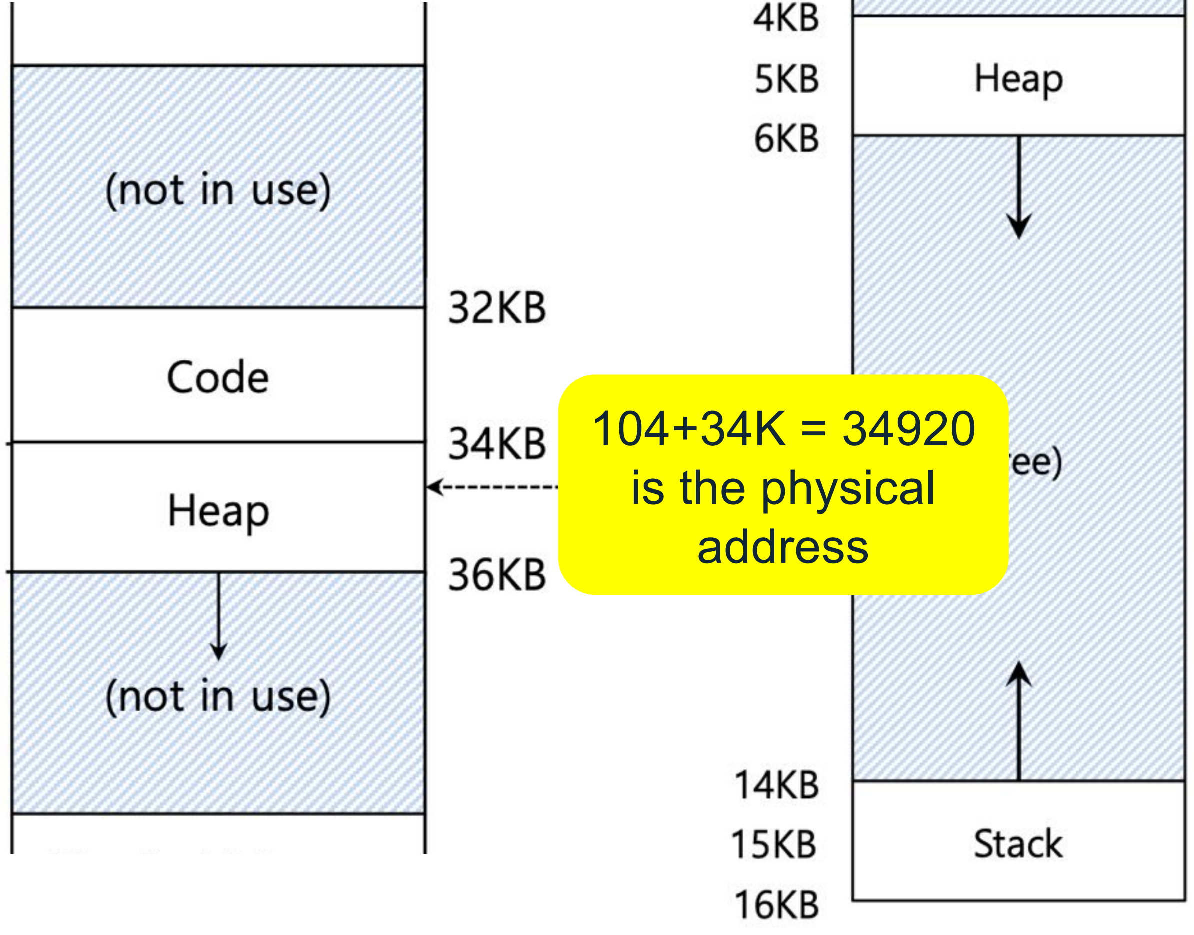


Address Translation on Segmentation

Segment	Base	Size
Heap	34K	2K



The heap segment starts at virtual address 4096.
The offset 4200 is 104.

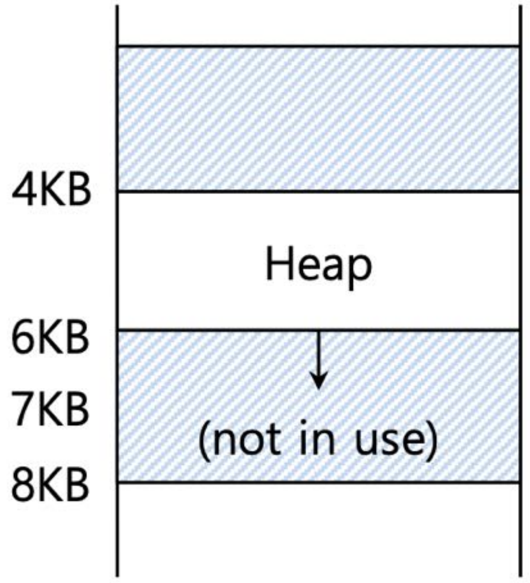


104+34K = 34920 is the physical address

Segmentation Fault or Violation

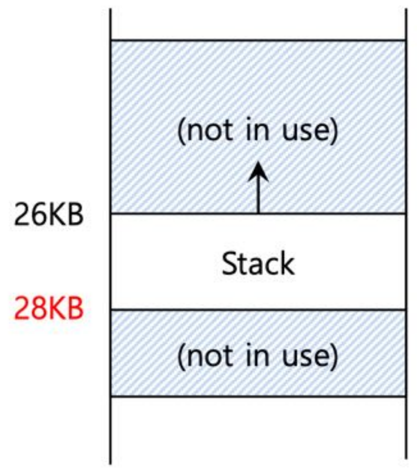
If an illegal address such as 7KB which is beyond the end of heap is referenced, the OS occurs segmentation fault.

- The hardware detects that address is out of bounds.



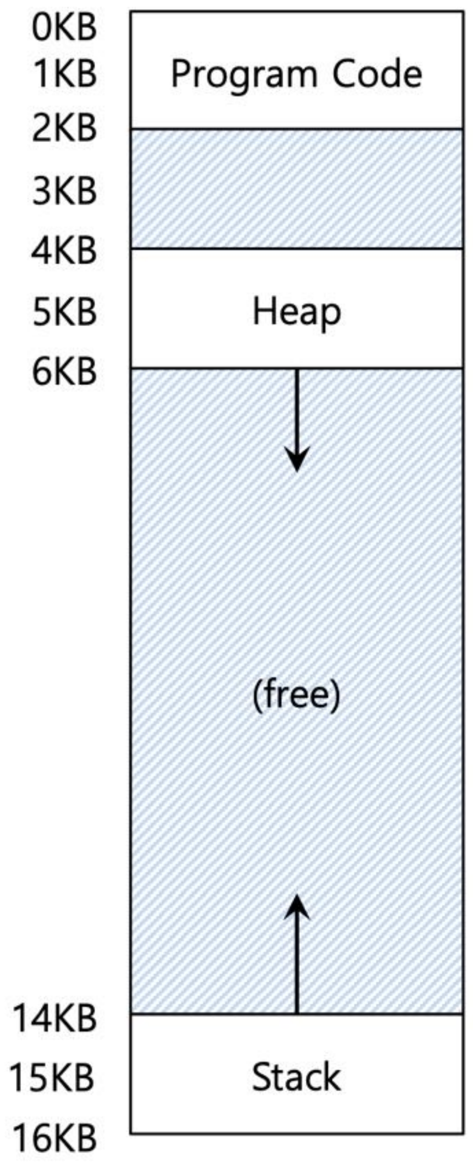
Referring to Stack Segment

- Stack grows backward.
- Extra hardware support is need.
    - The hardware checks which way the segment grows. w
    - 1: positive direction, 0: negative direction



Segment Register(with Negative-Growth Support)

Segment	Base	Size	Grows Positive?
Code	32K	2K	1
Heap	34K	2K	1
Stack	28K	2K	0



Support for Sharing

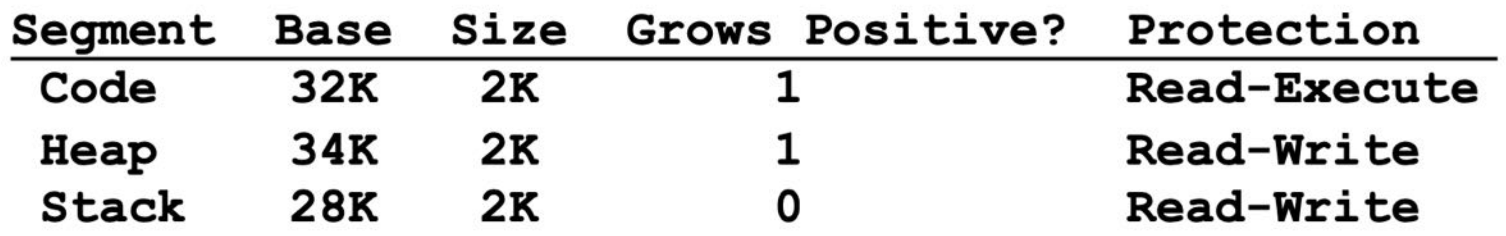
Segment can be shared between address space.

- Code sharing is still in use in systems today.
- by extra hardware support.

Extra hardware support is need for form of Protection bits.

- A few more bits per segment to indicate permissions of read, write and execute.

Segment Register Values(with Protection)

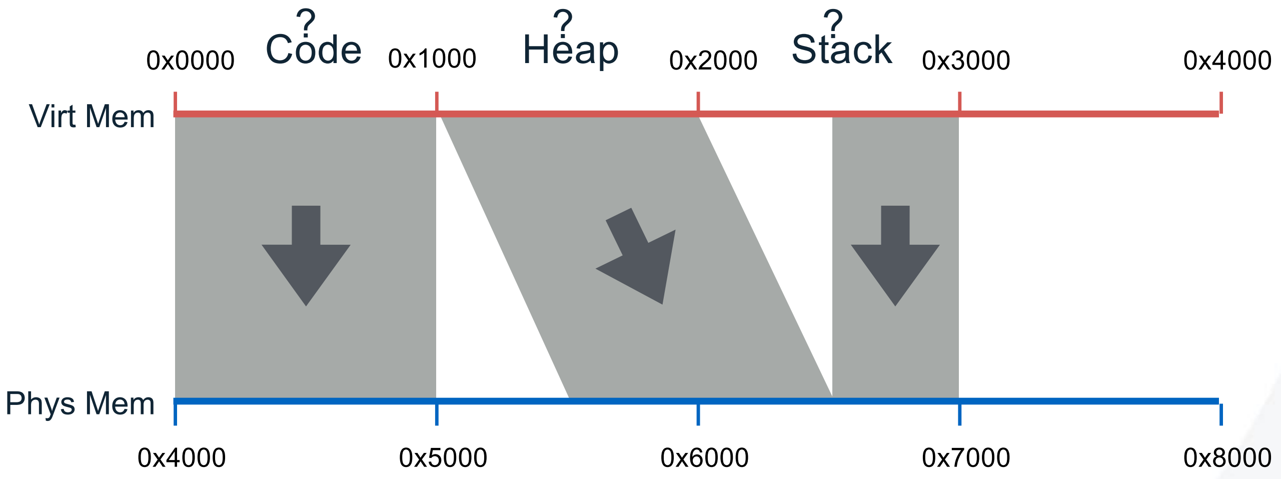


Segment	Base	Size	Grows Positive?	Protection
Code	32K	2K	1	Read-Execute
Heap	34K	2K	1	Read-Write
Stack	28K	2K	0	Read-Write

Summary: Segmentation

Assume 14-bit virtual addresses, high 2 bits indicate segment

Segments:
0=>code
1=>heap
2=>stack.



Seg	Base	Bounds
0	0x4000	0xfff
1	0x5800	0xfff
2	0x6800	0x7ff

Review: Memory Accesses

```
0x0010: movl 0x1100, %edi
0x0013: addl $0x3, %edi
0x0019: movl %edi, 0x1100
```

Seg	Base	Bounds
0	0x4000	0xfff
1	0x5800	0xfff
2	0x6800	0x7ff

Physical Memory Accesses?

1) Fetch instruction at logical addr 0x0010

- Physical addr: 0x4010

Exec, load from logical addr 0x1100

- Physical addr: 0x5900

2) Fetch instruction at logical addr 0x0013

- Physical addr: 0x4013

Exec, no load

3) Fetch instruction at logical addr 0x0019

- Physical addr: 0x4019

Exec, store to logical addr 0x1100

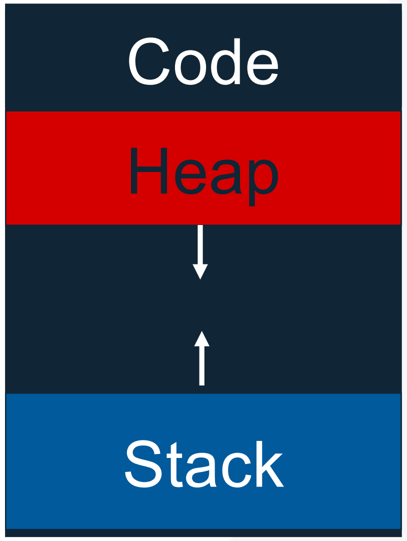
- Physical addr: 0x5900

Total of 5 memory references (3 instruction fetches, 2 movl)

THE UNIVERSITY of ADELAIDE

Advantages of Segmentation

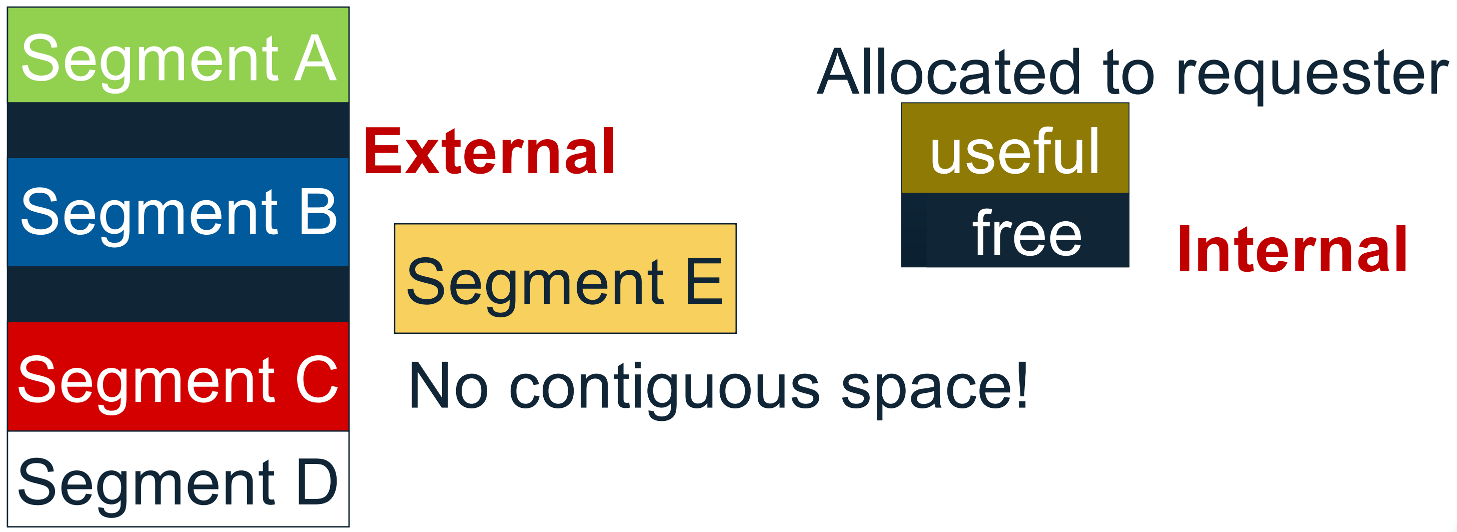
- **Enables sparse allocation of address space**
  - Stack and heap can grow independently
  - Heap: If no data on free list, dynamic memory allocator requests more from OS (e.g., UNIX: malloc calls sbrk())
  - Stack: OS recognizes reference outside legal segment, extends stack implicitly
- **Different protection for different segments**
  - Read-only status for code
- **Enables sharing of selected segments**
- **Supports dynamic relocation of each segment**



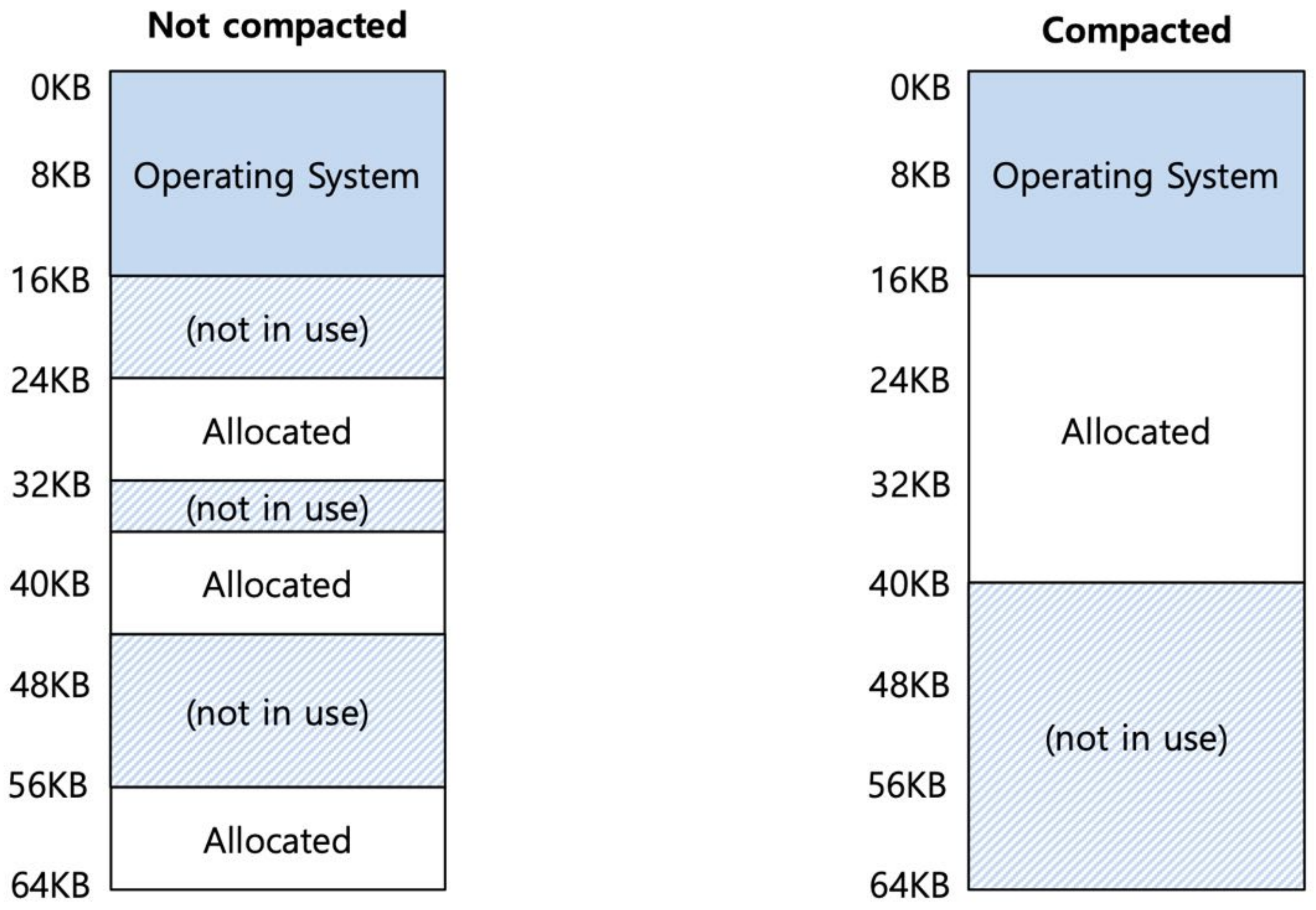
Problem: Fragmentation

- **Definition: Free memory that can not be usefully allocated**
- **Why?**
  - Free memory (hole) is too small and scattered
  - Rules for allocating memory prohibit using this free space
- **Types of fragmentation**
  - External: Visible to allocator (e.g., OS)
  - Internal: Visible to requester (e.g., if must allocate at some granularity)

Problem: Fragmentation



Memory Compaction



THE UNIVERSITY of ADELAIDE

Disadvantages of Segmentation

- **Each segment must be allocated contiguously**
  - May not have sufficient physical memory for large segments

- **Fix with paging…**

Summary

- **HW+OS work together to virtualize memory**
  - Give illusion of private address space to each process
- **Add MMU registers for base+bounds so translation is fast**
  - OS not involved with every address translation, only on context switch or errors
- **Dynamic relocation with segments is good building block**
  - Next: Solve fragmentation with paging

THE UNIVERSITY of ADELAIDE

COMP SCI 3004 Operating Systems

Paging

make history.

THE UNIVERSITY of ADELAIDE

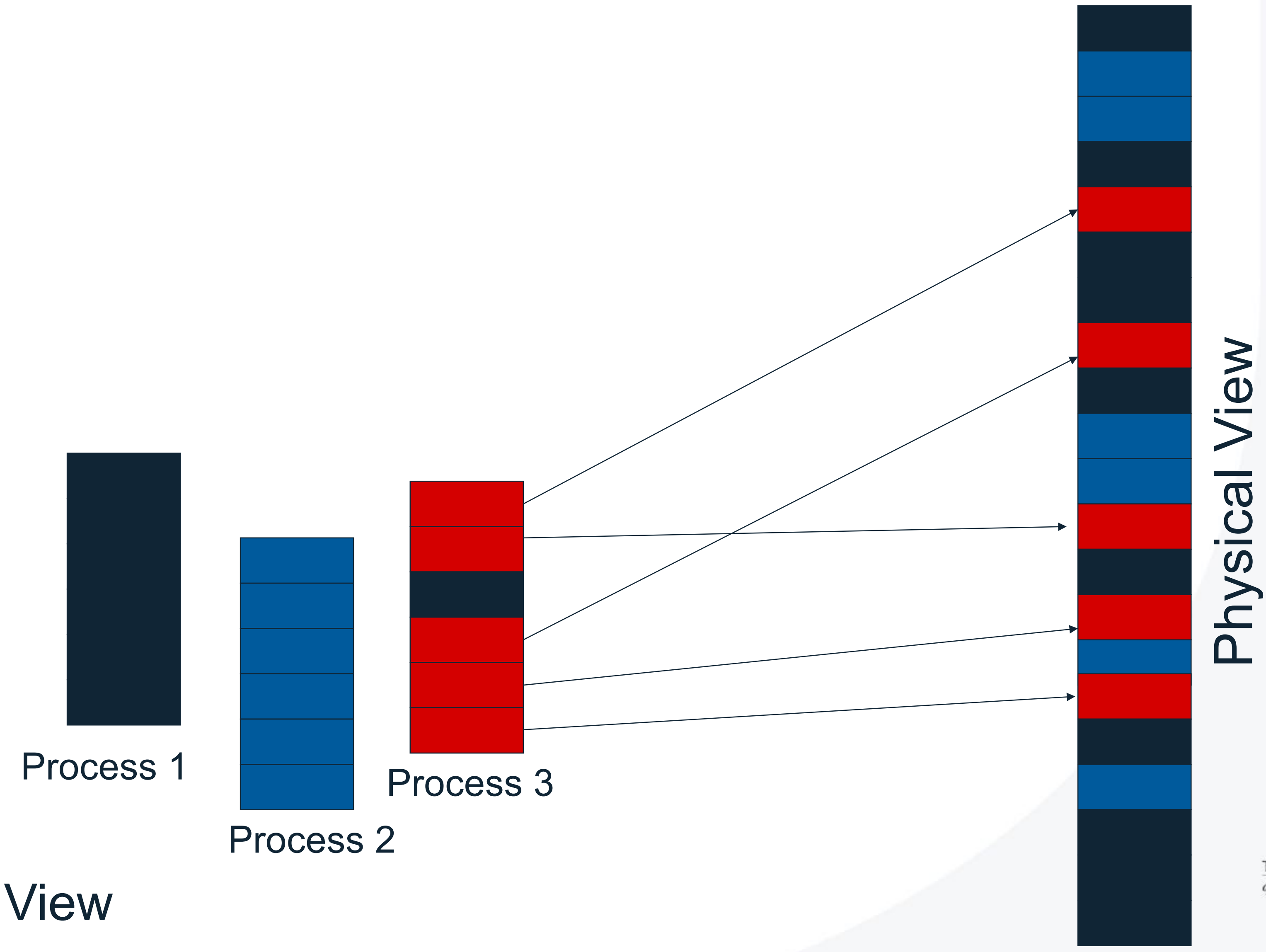
Introduction into paging

- What is paging?
- Where are page tables stored?
- What are advantages and disadvantages of paging?

Paging

- **Goal: Eliminate requirement that address space is contiguous**
  - Eliminate external fragmentation
  - Grow segments as needed
- **Idea: Divide address spaces and physical memory into fixed-sized pages**
  - Size: 2^n, Example: 4KB
  - Physical page: page frame

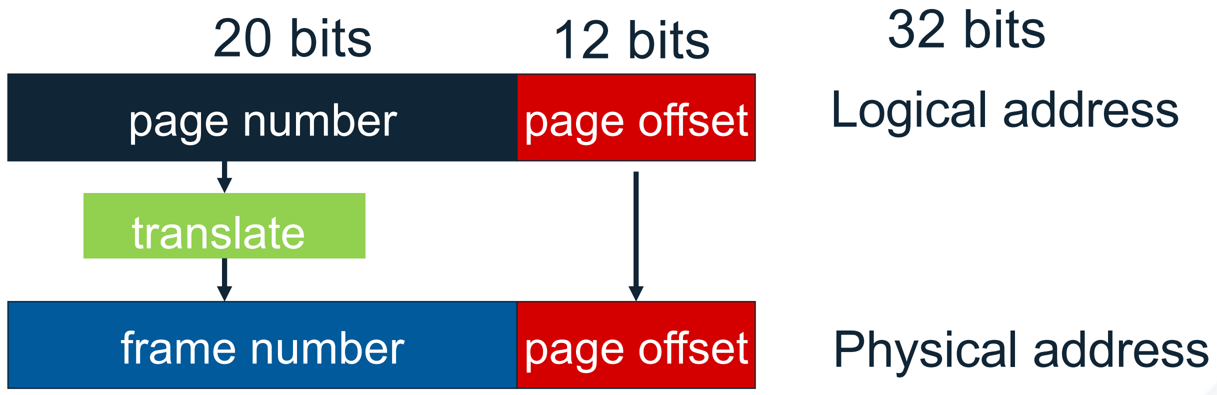
Paging



Logical View

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; just append bits correctly…

- How does format of address space determine number of pages and size of pages?

Example: Address Format

Given the known page size, how many bits are needed in an address to specify offset in a page?

Page Size	Low Bits (offset)
16 bytes	4
1 KB	10
1 MB	20
512 bytes	9
4 KB	12

Example: Address Format

Given number of bits in virtual address and bits for offset, how many bits for virtual page number?

Page Size	Low Bits (offset)	Virt Addr Bits	High Bits (virtual page number)
16 bytes	4	10	6
1 KB	10	20	10
1 MB	20	32	12
512 bytes	9	16	7
4 KB	12	32	20

THE UNIVERSITY of ADELAIDE

Example: Address Format

Given number of bits for vpn, how many virtual pages can there be in an address space?

Page Size	Low Bits (offset)	Virt Addr Bits	High Bits (vpn)	Virt Pages
16 bytes	4	10	6	64
1 KB	10	20	10	1 K
1 MB	20	32	12	4 K
512 bytes	9	16	5	128
4 KB	12	32	20	1 M

THE UNIVERSITY of ADELAIDE

Where Are Pagetables Stored?

How big is a typical page table?
- assume 32-bit address space
- assume 4 KB pages
- assume 4 byte entries

Final answer: 2 ^ (32 - log(4KB)) * 4 = 4 MB

Page table size = Num entries * size of each entry

Num entries = num virtual pages = 2^(bits for vpn)

Bits for vpn = 32– number of bits for page offset

= 32 – lg(4KB) = 32 – 12 = 20

Num entries = 2^20 = 1 M

Page table size = Num entries * 4 bytes = 4 MB

Implication: Store each page table in memory

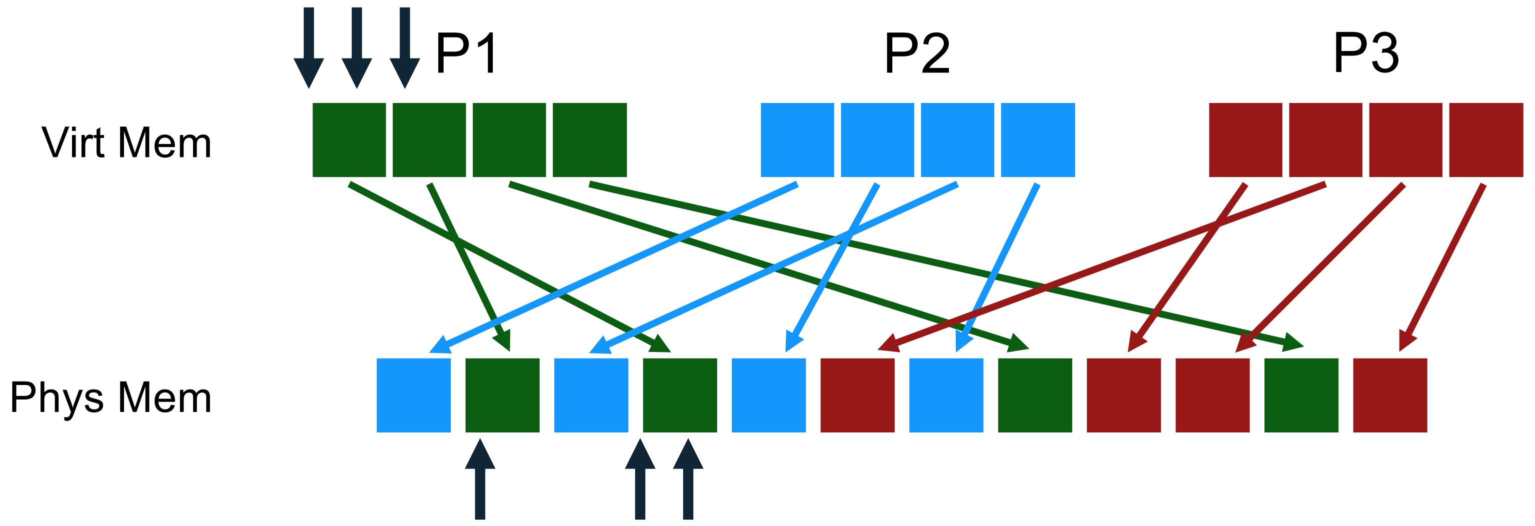
Hardware finds page table base with register

What happens on a context-switch?

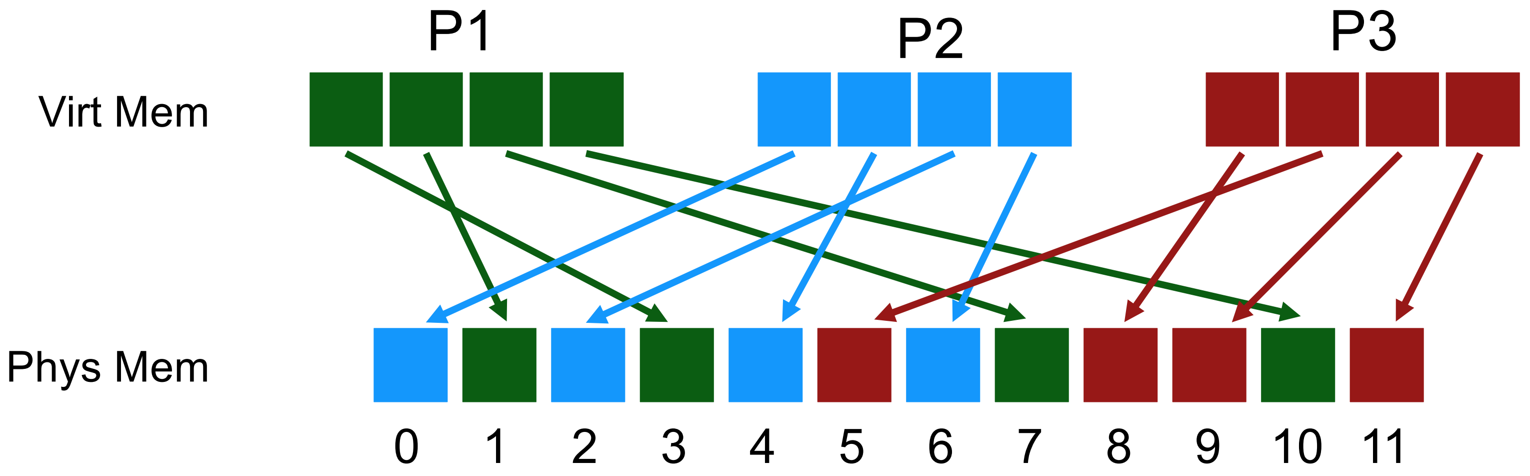
Change contents of page table base register to newly scheduled process

Save old page table base register in PCB of descheduled process

The Mapping



Example: Fill in Page Table



Page Tables:

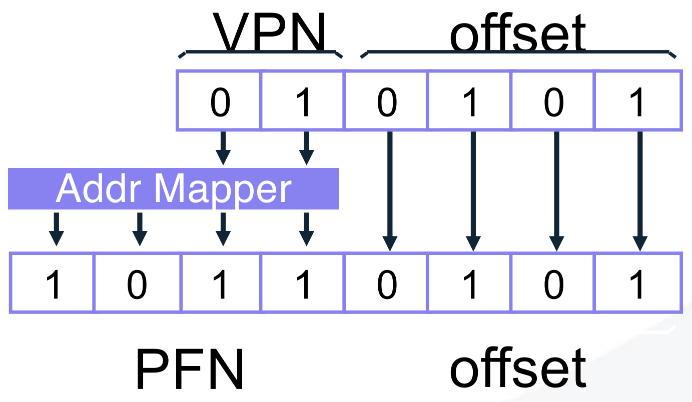
P1	P2	P3
3	0	8
1	4	5
7	2	9
10	6	11

Virtual => Physical PAGE Mapping

- How should OS translate VPN to PFN?
- For segmentation, OS used a formula (e.g., phys addr = virt_offset + base_reg)
- For paging, OS needs more general mapping mechanism
- What data structure is good?

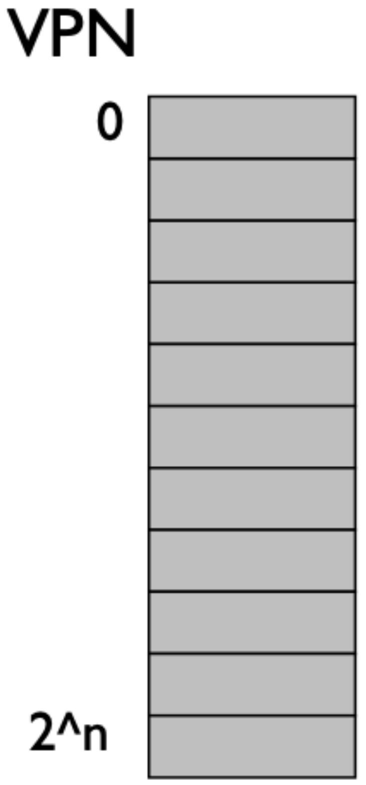
Big array: pagetable

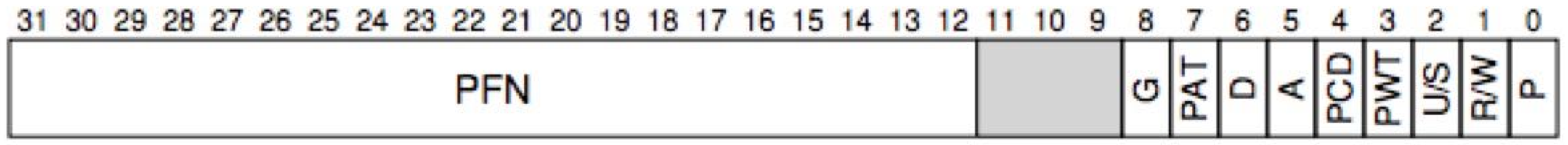
Number of bits in virtual address format does not need to equal number of bits in physical address format



Page Tables

- What is a good data structure?
- Simple solution: Linear page table aka array





Other PT info

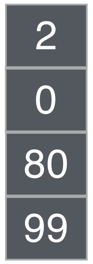
- **What other info is in pagetable entries besides translation?**
  - valid bit
  - protection bits
  - present bit (needed later)
  - reference bit (needed later)
  - dirty bit (needed later)
- **Pagetable entries are just bits stored in memory**
  - Agreement between hw and OS about interpretation

Memory Accesses with Pages

```
0x0010:  movl  0x1100, %edi
0x0013:  addl  $0x3, %edi
0x0019:  movl  %edi, 0x1100
```

Assume PT is at phys addr 0x5000
Assume PTE's are 4 bytes
Assume 4KB pages
How many bits for offset? 12

Simplified view
of page table



Old: How many mem refs with segmentation?
5 (3 instrs, 2 movl)

Physical Memory Accesses with Paging?

1) Fetch instruction at logical addr 0x0010; vpn?
- Access page table to get pfn for vpn 0
- Mem ref 1: 0x5000
- Learn vpn 0 is at pfn 2
- Fetch instruction at 0x2010 (Mem ref 2)

Exec, load from logical addr 0x1100; vpn?
- Access page table to get pfn for vpn 1
- Mem ref 3: 0x5004
- Learn vpn 1 is at pfn 0
- Movl from 0x0100 into reg (Mem ref 4)

Pagetable is slow!!! Doubles memory references

Advantages of Paging

- **No external fragmentation:** Any page can be placed in any frame in physical memory
- **Fast to allocate and free**
  - Alloc: No searching for suitable free space
  - Free: Doesn't have to coalesce with adjacent free space
  - Just use bitmap to show free/allocated page frames
- **Simple to swap-out portions of memory to disk (later)**
  - Page size matches disk block size
  - Can run process when some pages are on disk
  - Add "present" bit to PTE

THE UNIVERSITY *of* ADELAIDE

Disadvantages of Paging

- **Internal fragmentation:** Page size may not match size needed by process. Wasted memory grows with larger pages
- **Additional memory reference to page table.** Very inefficient. Page table must be stored in memory. MMU stores only base address of page table
- **Storage for page tables may be substantial.** Requires PTE for all pages in address space. Entry needed even if page not allocated.

THE UNIVERSITY *of* ADELAIDE

Summary

For-each mem reference:

1. extract VPN (virt page num) from VA (virt addr)
2. calculate addr of PTE (page table entry)
3. read PTE from memory
4. extract PFN (page frame num)
5. build PA (phys addr)
6. read contents of PA