CS 423
Operating System Design: Segmentation & Paging
Feb 05

Ram Kesavan

Acknowledgement: Prof. Shivaram Venkataraman (Wisconsin)

Logistics

MP0: Due on 02/09 11:59pm CT

MP1:

- Will get posted by tonight
- Walkthrough by Gabriella on 2/10

4Cr: Linux APIs paper; due by 2/10 2pm CT

- Unsubmitted reviews will hurt your participation grade

Google Apps @Illinois problems

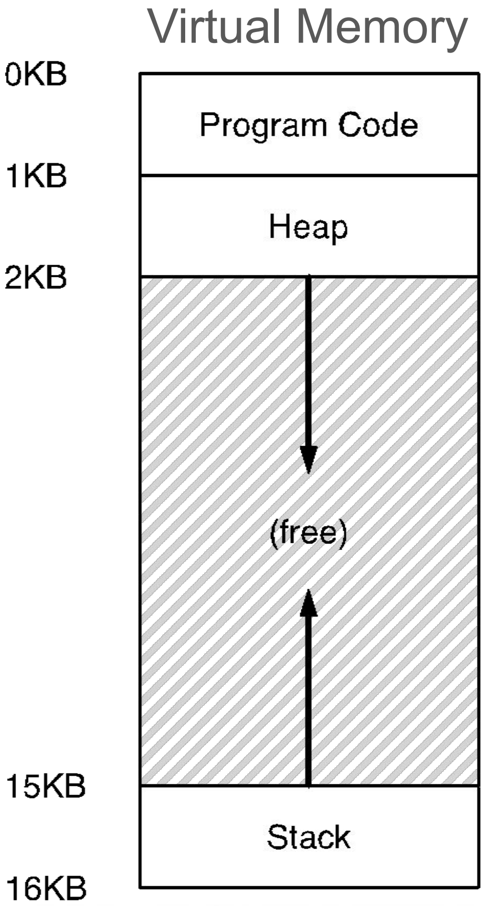
AGENDA / LEARNING OUTCOMES

Memory virtualization

- Segmentation: technique used by older systems
- Pros and cons?
- Paging & modern systems

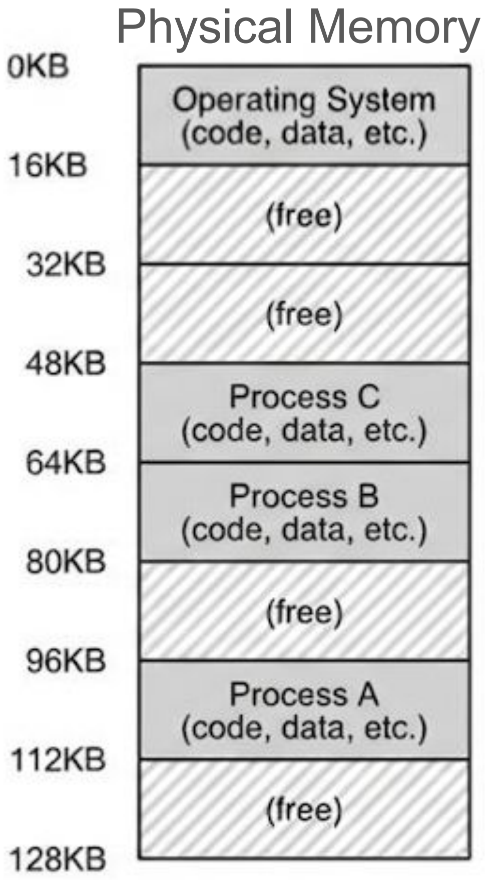
RECAP

ABSTRACTION: ADDRESS SPACE



Virtual Address Space: Each process has its own address range

OS provides that Illusion by mapping to physical memory


Physical Memory
0KB
Operating System (code, data, etc.)
16KB
(free)
32KB
(free)
48KB
Process C (code, data, etc.)
64KB
Process B (code, data, etc.)
80KB
(free)
96KB
Process A (code, data, etc.)
112KB
(free)
128KB

HOW TO VIRTUALIZE MEMORY

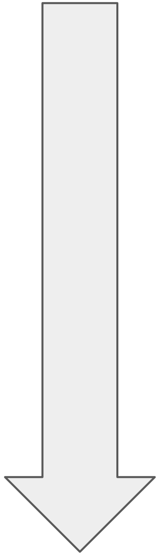
Problem: Addresses are “hardcoded” into process binaries

How to avoid collisions?

End of Recap

Mechanisms for Virtualization

1. Time Sharing
2. Static Relocation
3. Base
4. Base+Bounds
5. Segmentation



Limited practicality, has many problems

More practical, still has some problems

x86 and Linux: Paging, TLB, Efficient Page Tables

SEGMENTATION

Divide VM address space into logical segments

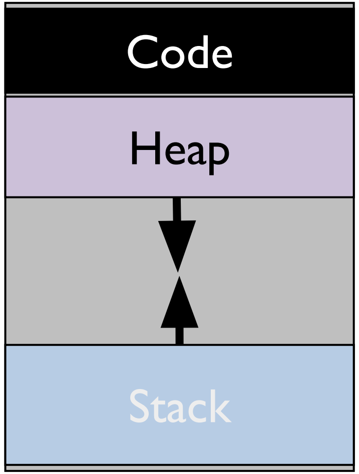
- ○ Each segment corresponds to logical entity in address space (code, stack, heap)

Each segment with its own base + bounds register

Each segment can independently:

1. Be placed in physical memory
2. Grow and shrink (not code & static data)
3. Be protected (read/write/exec)

VM layout



SEGMENTED ADDRESSING

Process should specify segment and offset within segment

How does process designate a particular segment?

- Use portion of logical/VM address
  - MSBs of logical address => segment
  - LSBs of logical address => offset within segment

Segmentation Implementation

MMU contains Segment Table (per process)

- Each segment has own base and bounds, protection bits
- Example: 16KB VM address space = 14 bit logical address with 4 segments

How many bits for segment?

How many bits for offset?

Segment	Base	Bounds	R W
00(code)	32KB	2KB	1 0
01(heap)	34KB	3KB	1 1
11(stack)	28KB	2KB	1 1
10(unused)	0x0000	0x000	0 0

Example Translations

Segment	Base	Bounds	R W
0	32KB	2KB	1 0
1	34KB	3KB	1 1
3	28KB	2KB	1 1
2	0x0000	0x000	0 0

Translate logical (in hex) to physical

0x1108:

Bottom 12 bits =

Offset =

Is Offset < Bounds?

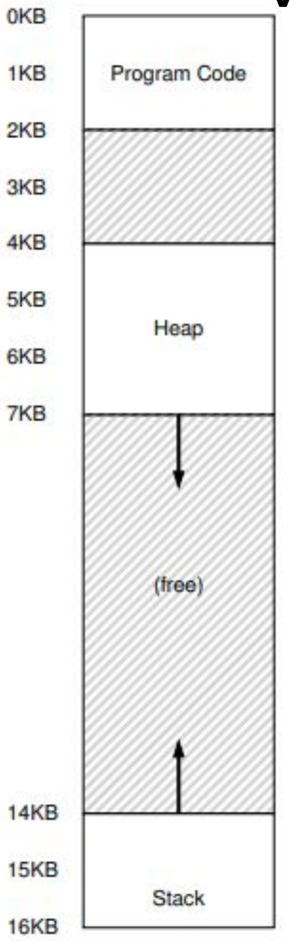
Top 2 bits =

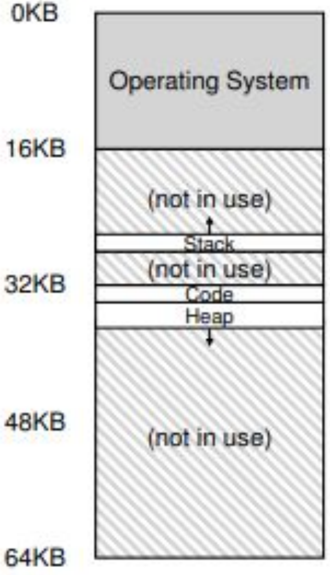
Segment number:

Physical addr:

base + offset =

VIRTUAL->PHYSICAL TRANSLATION





Heap Segment

Virtual address range : 4 KB to 7 KB

Physical address range: 34 KB to 37 KB

Virtual address 6 KB (hex: 0x1800)

Which segment?

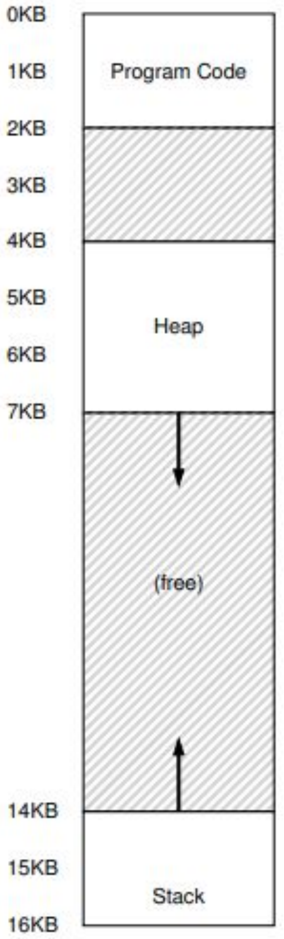
What is the offset?

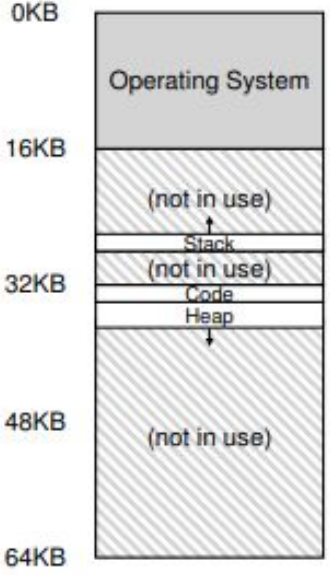
How does the MMU translate that?

Physical address = Base + Offset

=

STACK ADDRESS TRANSLATION





Stack Segment

Virtual address range : 16 KB to 14 KB

Physical address range: 28 KB to 26 KB

Virtual address 15 KB (hex: 0x3C00)

Which segment?

What's the offset?

How does the MMU translate that?

Physical address =?

SEGMENTATION IMPLEMENTATION

MMU contains Segment Table (per process)

- Each segment has own base and bounds, protection bits
- Example: 16KB VM address space = 14 bit logical address with 4 segments

Segment	Base	Bounds	R W	Grows?
00(code)	32KB	2KB	1 0	1
01(heap)	34KB	3KB	1 1	1
11(stack)	28KB	2KB	1 1	0
10(unused)	0x0000	0x000	0 0	0

SEGMENTATION: ADVANTAGES

Enables sparse allocation of address space

Stack and heap can grow dynamically

- Heap: If no data on free list, dynamic memory allocator requests more from OS (e.g., UNIX: malloc lib calls sbrk())
- Stack: OS recognizes reference outside legal segment, extends stack implicitly

Different protection for different segments

- Enables sharing of selected segments (2 processes share code)
- With no write permission for code segment

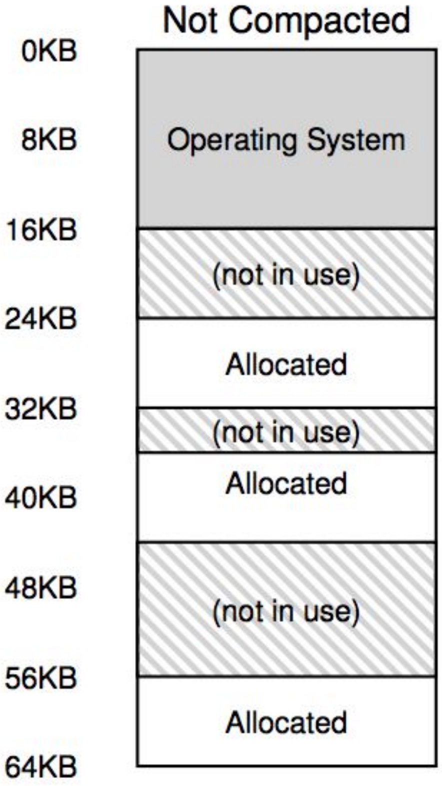
Supports dynamic relocation of each segment

SEGMENTATION: DISADVANTAGES

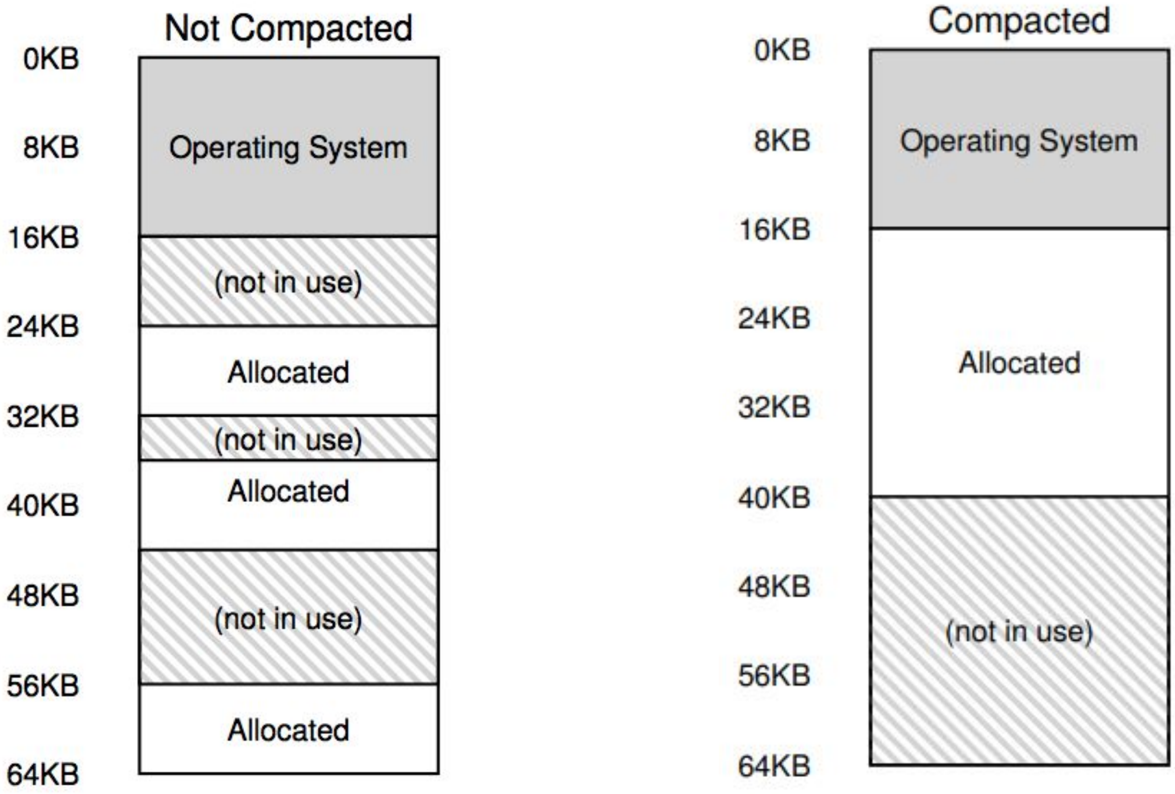
Each segment must be allocated contiguously

Physical memory gets fragmented

May not have sufficient physical memory for large segments?



COMPACT & REARRANGE SEGMENTS



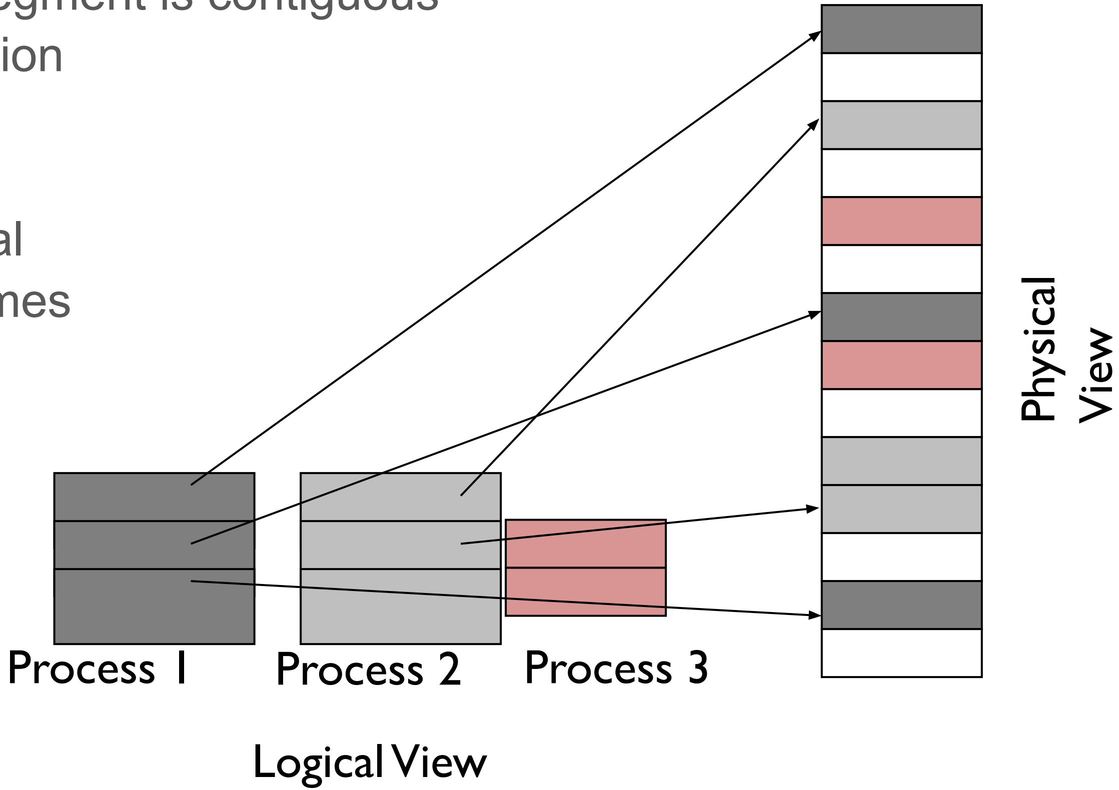
PAGING
(most modern systems including Linux use this)

PAGING

Goal: Eliminate requirement that segment is contiguous
Eliminate external fragmentation

Idea:
Divide address spaces and physical memory into fixed-sized pages/frames

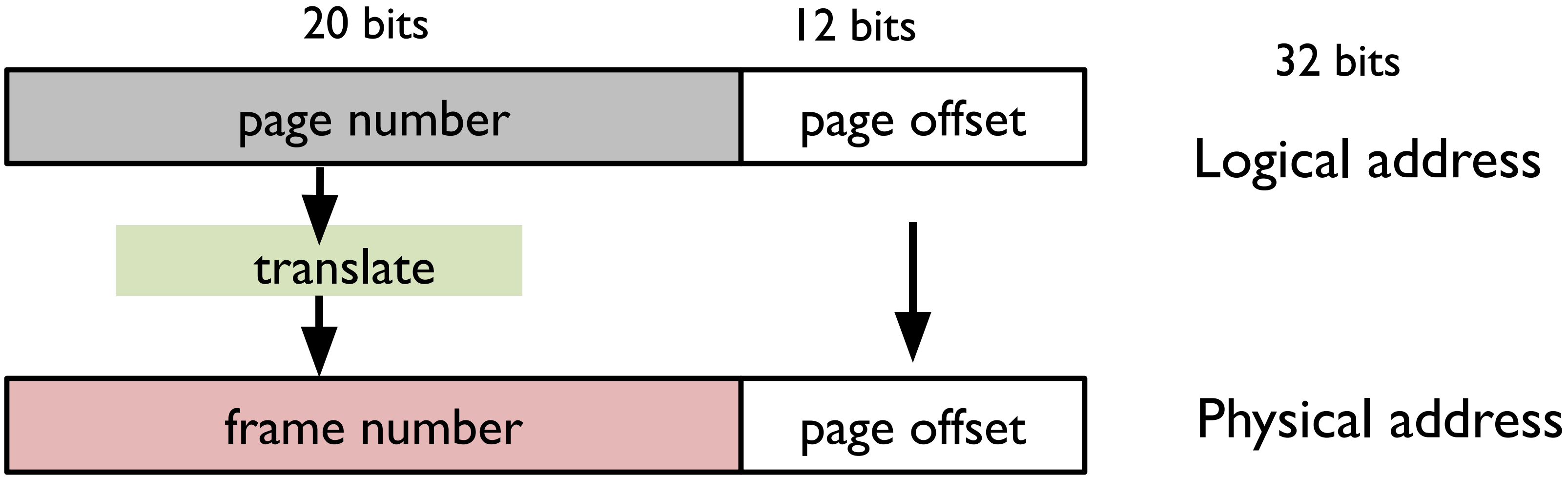
Size: 2^n, Example: 4KB



TRANSLATION of 4KB PAGE ADDRESS

How to translate logical address to physical address?

- MSBs of virtual address => page number
- LSBs bits of virtual address => offset within page

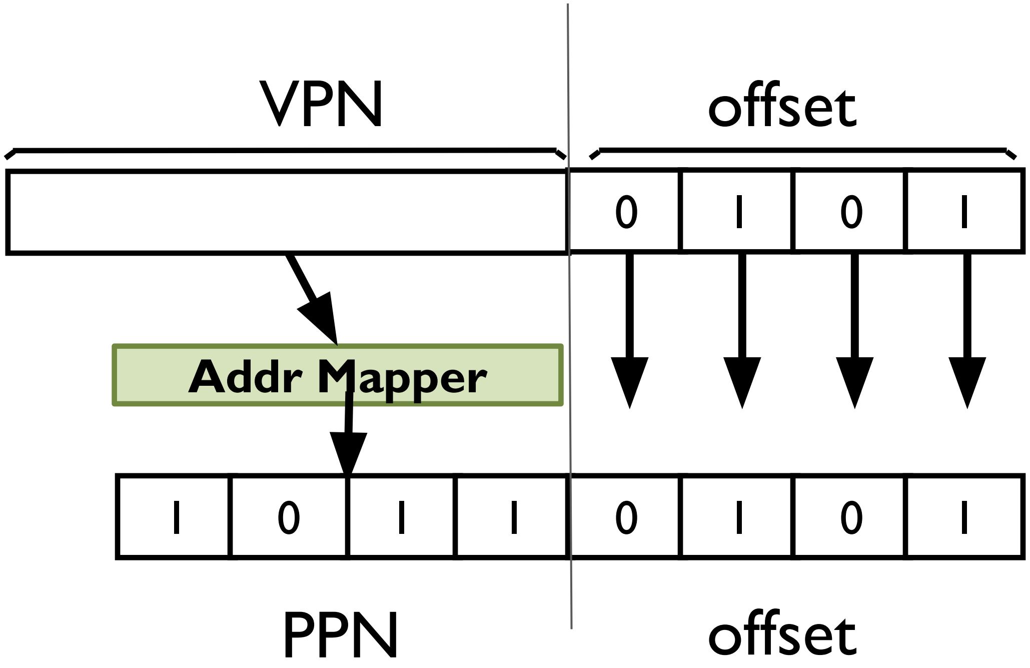


ADDRESS FORMAT

Page Size	Low Bits (offset)	N-Bit Address	High Bits (vpn)	Virt Pages
16 bytes		10		
1 KB		20		
1 MB		32		
512 bytes		16		
4 KB		32		

VIRTUAL -> PHYSICAL PAGE MAPPING

#bits in virtual address need not be equal to #bits in physical address



# bits in VPN

> # bits in PPN?

What does this mean?

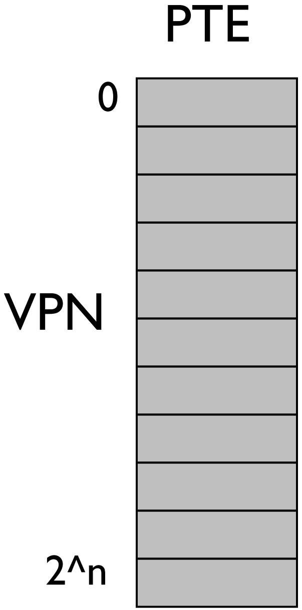
Is inverse possible?

How should OS translate VPN to PPN/PFN?

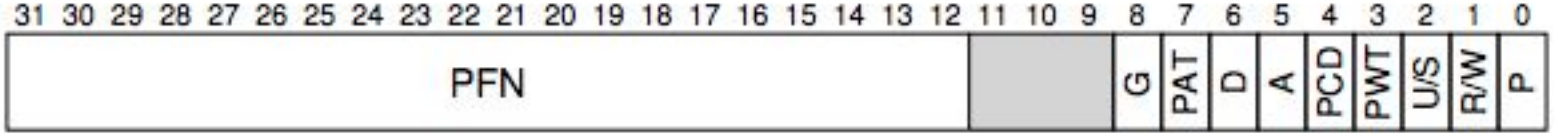
LINEAR PAGE TABLE

What is an obvious data structure?

- Linear page table (array)
- Indexed by VPN
- Yields PFN (Physical Frame Number)



A Single PTE (Page Table Entry):



ADVANTAGES OF PAGING

No external fragmentation

- A page can be placed in any frame in physical memory

Fast to allocate and free

- Alloc: No searching for suitable free space
- Free: No need to coalesce/compact free space

Simple to swap-out portions of memory to disk (later lecture)

- Page size matches disk block size!
- Can run process when some pages are on disk
- Add “present” bit to PTE

HOW BIG IS A PAGE TABLE?

Assume 32-bit address
Assume 4KB pages

Assume 4 byte page table entries (PTE)

How large is PT for each process?

Implications?

IMPLICATIONS

Page tables may be substantial

- Simple page table: requires PTE for all pages in address space
  Entry needed even if page not allocated ?

Additional memory reference to page table

- Very inefficient, so…
- …page table must be stored in memory
- MMU stores only base address of page table

Could choose larger page size

- Leads to (internal) fragmentation

Next Lecture: Paging and TLBs