CSCE 5610
Computer System Architecture

Virtual Memory

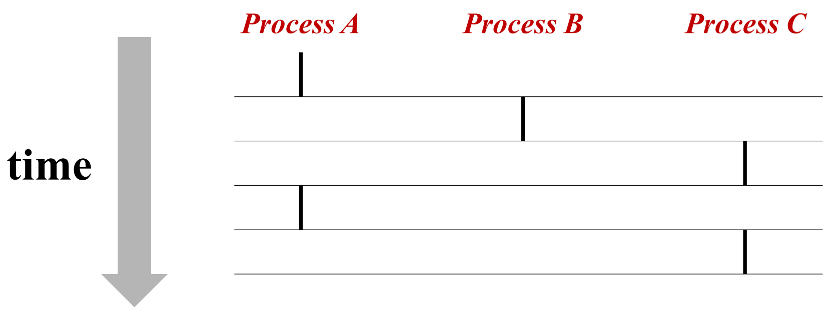
Processes

Definition: A *process* is an instance of a running program

- One of the most important ideas in computer science
- Not the same as “program” or “processor”

Concurrent Processes

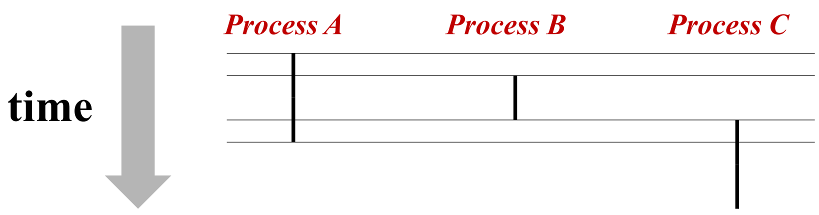
- **Two processes *run* *concurrently* (are concurrent) if their instruction executions (flows) overlap in time**
- **Otherwise, they are *sequential***
- **Examples:**
  - Concurrent: A & B, A & C
  - Sequential: B & C



User View of Concurrent Processes

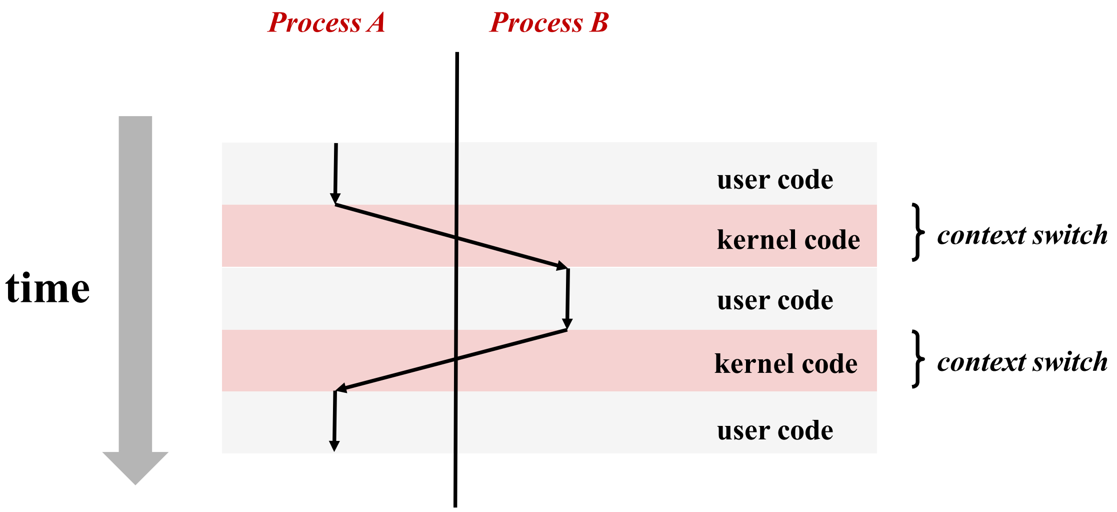
- **Control flows for concurrent processes are physically disjoint in time**

- **However, we can think of concurrent processes as executing in parallel (only an illusion?)**



Context Switching

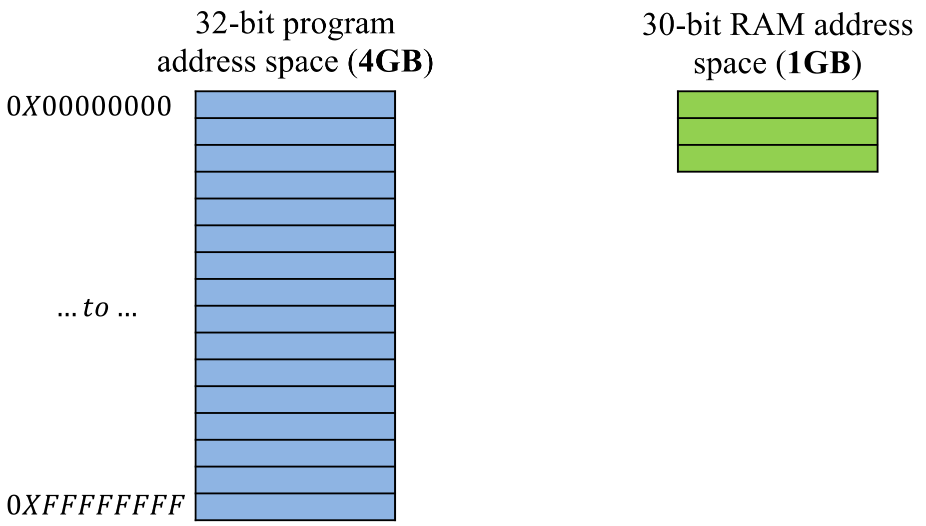
- **Processes are managed by a shared chunk of OS code called the *kernel***
  - Important: the kernel is not a separate process, but rather runs as part of a user process
- **Control flow passes from one process to another via a *context switch*…** *(how?)*



Three problems of memory

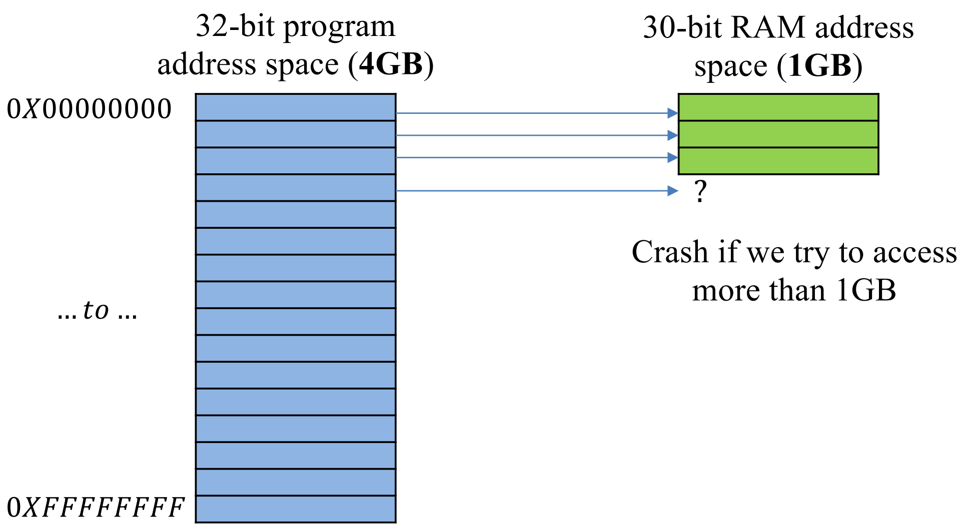
#1: What if we don't have enough memory

- MIPS gives each program its own 32-bit address space
- Programs can access any byte in their 32-bit address space
- How much memory can you access with a 32-bit address?
  - $2^{32} bytes = 4GB$
  - In practice, the OS reserves some of it. So, it is closer to 2GB of usable space.
- What if you don't have 4GB of memory?



#1: What if we don't have enough memory

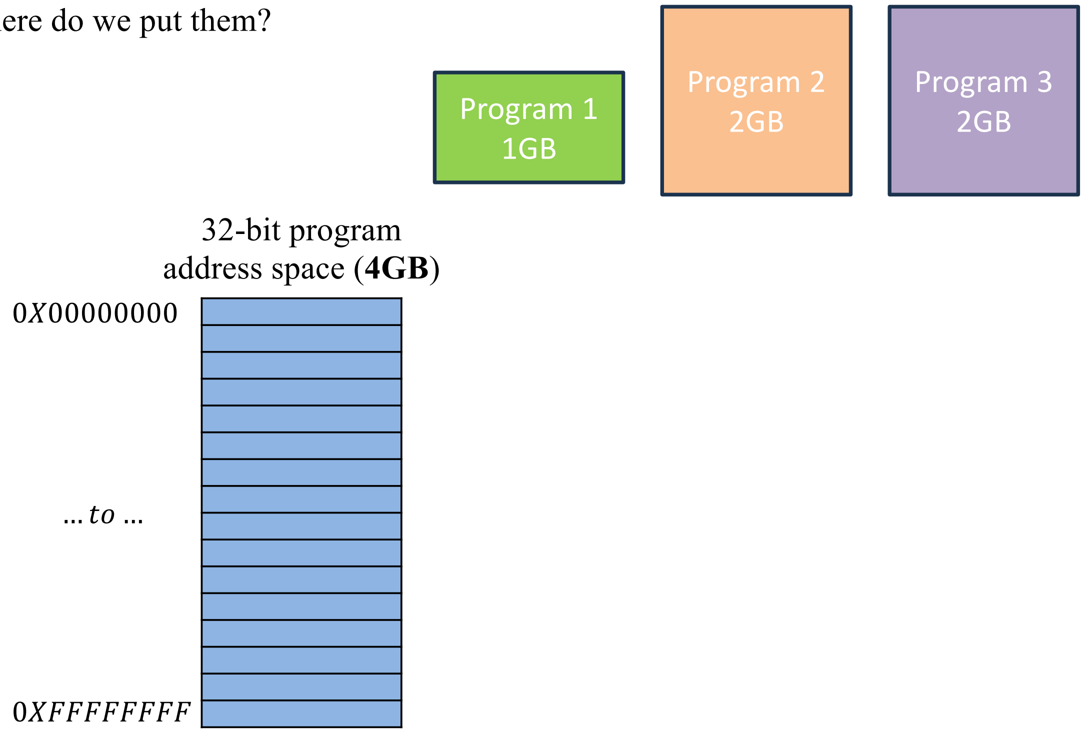
- What if you don't have 4GB of memory?



#2: Holes in our address space

How do programs share the memory?

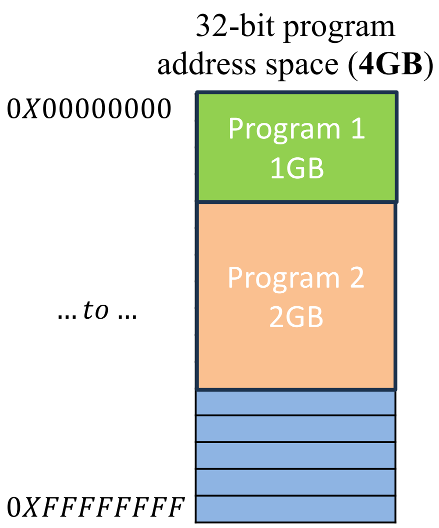
Where do we put them?



#2: Holes in our address space

How do programs share the memory?

Where do we put them?

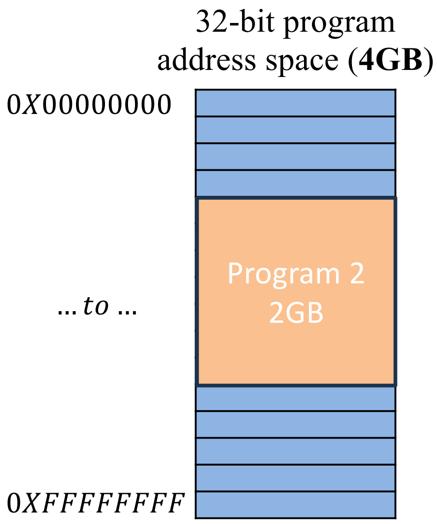


1. Programs 1 and 2 fit
They use 3GB of memory, leaving **1GB free**

#2: Holes in our address space

How do programs share the memory?

Where do we put them?



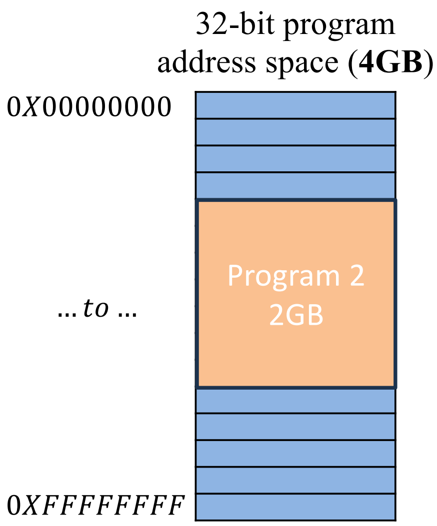
1. Programs 1 and 2 fit
They use 3GB of memory, leaving **1GB free**

2. Quit Program 1
Program 2 uses 2GB of memory, leaving **2GB free**

#2: Holes in our address space

How do programs share the memory?

Where do we put them?



1. Programs 1 and 2 fit
They use 3GB of memory, leaving **1GB free**

2. Quit Program 1
Program 2 uses 2GB of memory, leaving **2GB free**

3. Can't run Program 3
Even though we have enough free space

Memory fragmentation

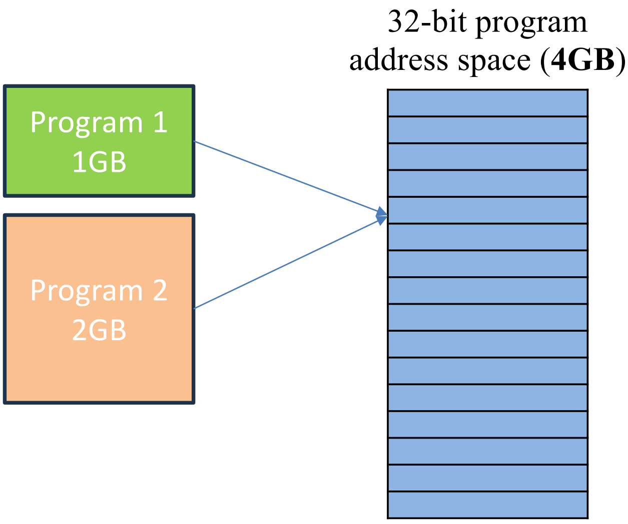
#3: How do we keep programs secure?

Each program can access any 32-bit memory address

What if multiple programs access the same address?

- sw $t0, 1024($0)

They can corrupt or crash each other: security and reliability



Problems with memory

- **If all programs have access to the same 32-bit memory space:**
  - Can crash if less than 4GB of RAM memory in the system
  - Can run out of space if we run multiple programs
  - Can corrupt other programs' data

- **How do we solve this?**

Problems with memory

- **If all programs have access to the same 32-bit memory space:**
  - Can crash if less than 4GB of RAM memory in the system
  - Can run out of space if we run multiple programs
  - Can corrupt other programs' data
- **How do we solve this?**
  - Key to the problem: "Same memory space"

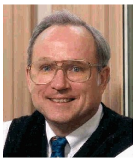
Problems with memory

- **If all programs have access to the same 32-bit memory space:**
  - Can crash if less than 4GB of RAM memory in the system
  - Can run out of space if we run multiple programs
  - Can corrupt other programs' data
- **How do we solve this?**
  - Key to the problem: "Same memory space"
  - Can we give each program it's **own** virtual memory space?
  - If so, we can:
    - Separately **map** each program's memory space to the RAM memory space, and even move it to disk if we run out of memory

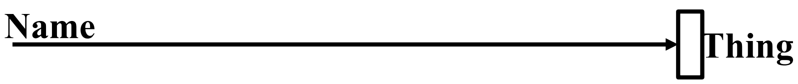
What is Virtual Memory?

Indirection

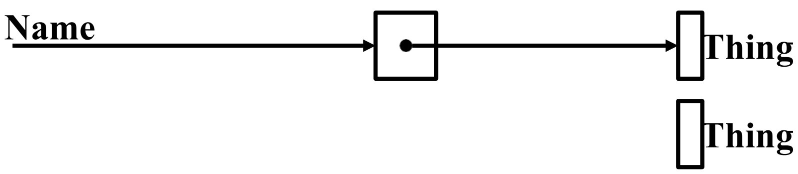
- **“Any problem in CS can be solved by adding a level of indirection”**



- **Without Indirection**



- **With Indirection**



- Examples:

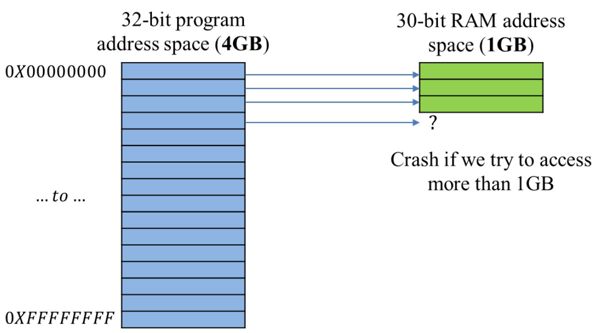
  Pointers, Domain Name Service (DNS) name->IP address, phone system (e.g., cell phone number portability), snail mail (e.g., mail forwarding), 911 (routed to local office), DHCP, call centers that route calls to available operators, etc.

Virtual memory is a layer of indirection

- "Any problem in Computer Science can be solved by adding indirection"
- Virtual memory takes program addresses and maps them to RAM addresses

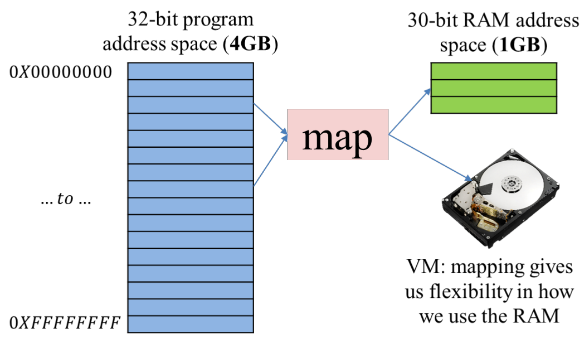
Without Virtual Memory

Program Address = RAM Address



With Virtual Memory

Program Address map to RAM Address

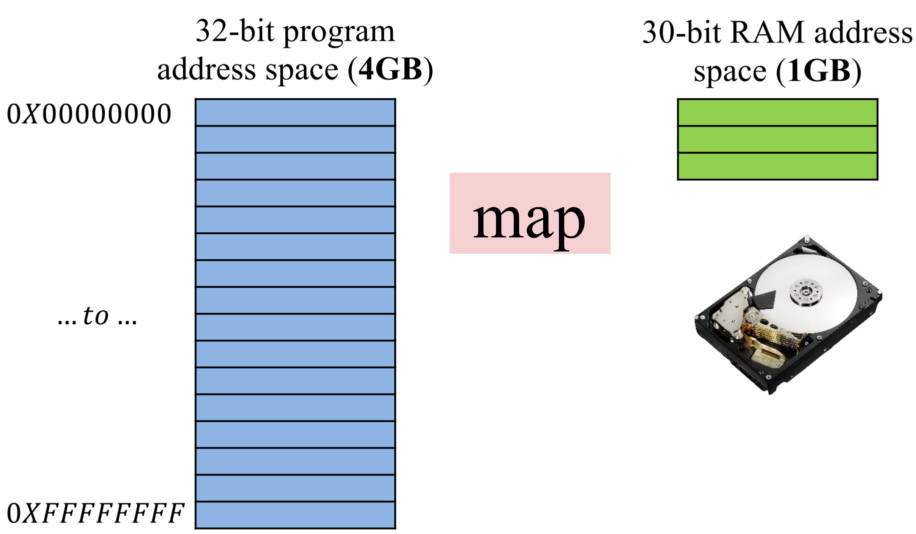


Solving the problems: #1 not enough memory

- Map some of the program's address space to the disk
- When we need it, we bring it into memory

With Virtual Memory

Program Address map to RAM Address

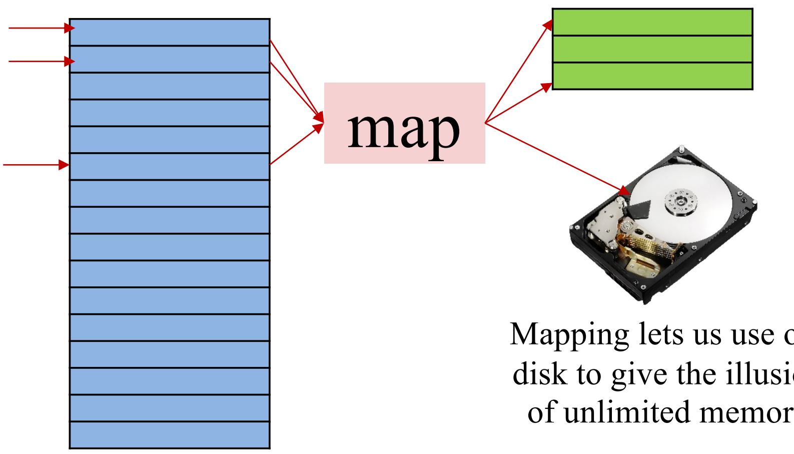


Solving the problems: #1 not enough memory

- Map some of the program's address space to the disk
- When we need it, we bring it into memory

With Virtual Memory

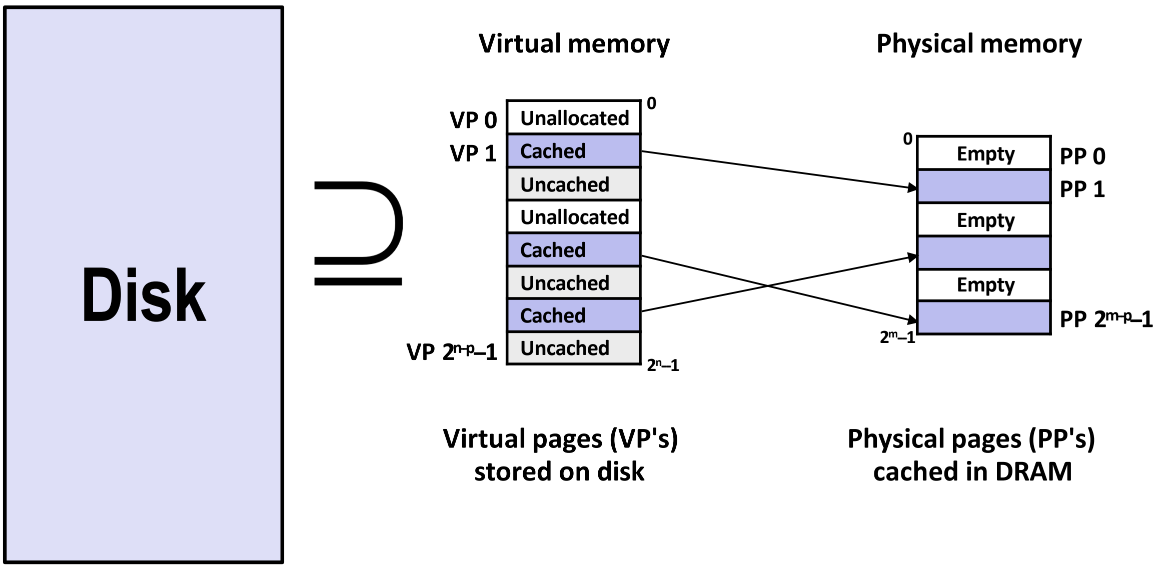
Program Address map to RAM Address



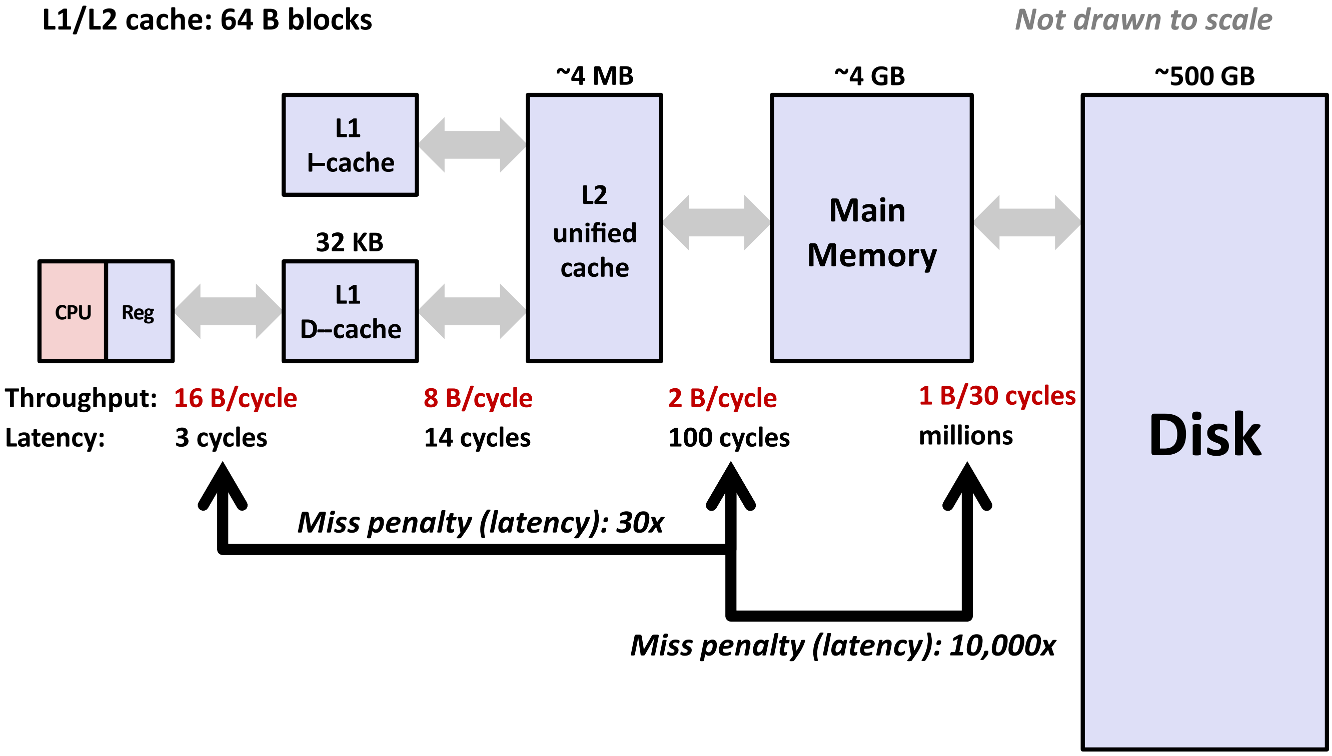
Mapping lets us use our disk to give the illusion of unlimited memory

VM as a Tool for Caching

- ***Virtual memory:* array of $N = 2^n$ contiguous bytes**
  - think of the array (allocated part) as being stored on disk
- **Physical main memory (DRAM) = cache for allocated virtual memory**
- **Blocks are called pages; size $=2^p$**

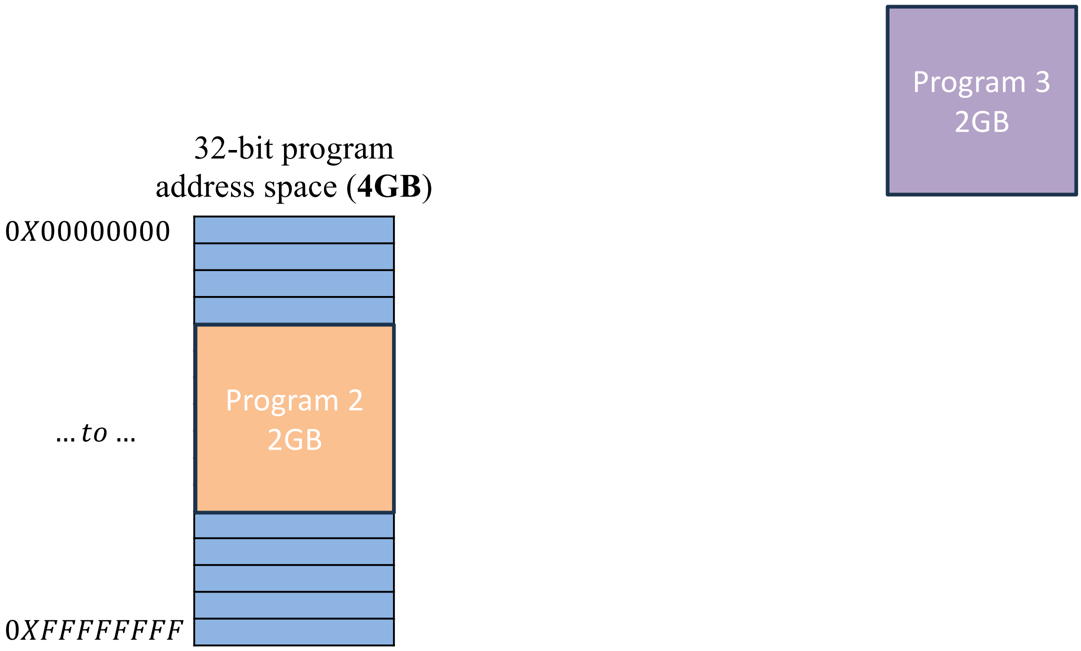


Memory Hierarchy: Core 2 Duo



Solving the problems: #2 holes in address space

- How do we use the holes left when programs quit?
- We can map a program's addresses to RAM addresses however we like

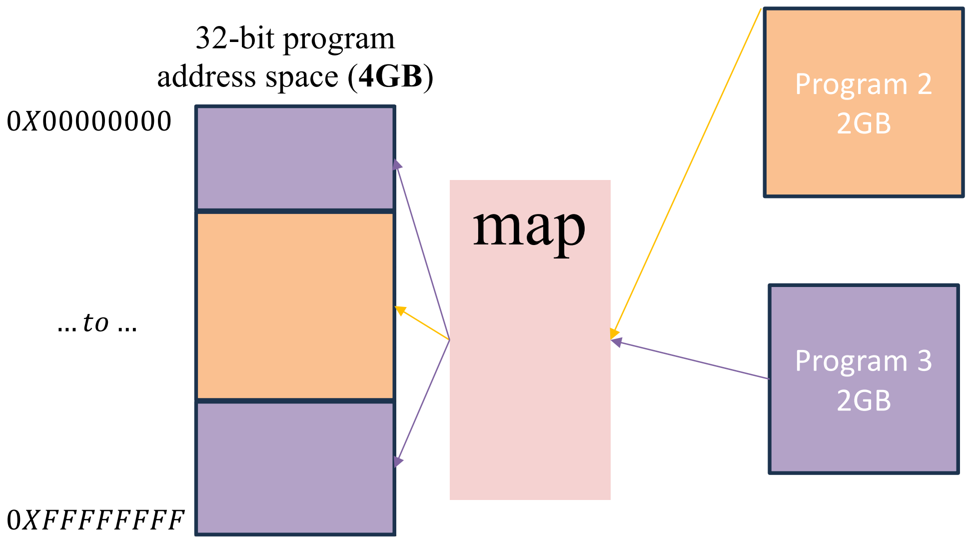


Solving the problems: #2 holes in address space

- How do we use the holes left when programs quit?
- We can map a program's addresses to RAM addresses however we like

With Virtual Memory

Program Address map to RAM Address



Each program has its own mapping. Mappings lets us put our program data wherever we want in the RAM.

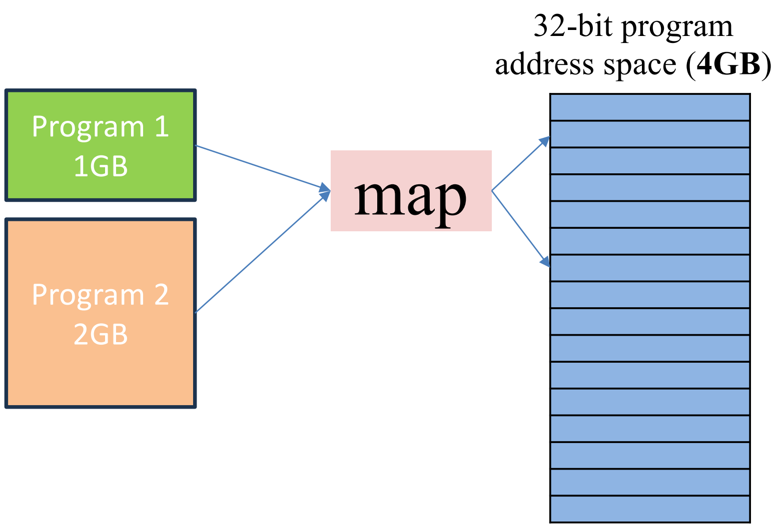
Solving the problems: #3 keeping program secure

- Program 1's and Program 2's addresses map to different RAM addresses

sw $t0, 1024($0)

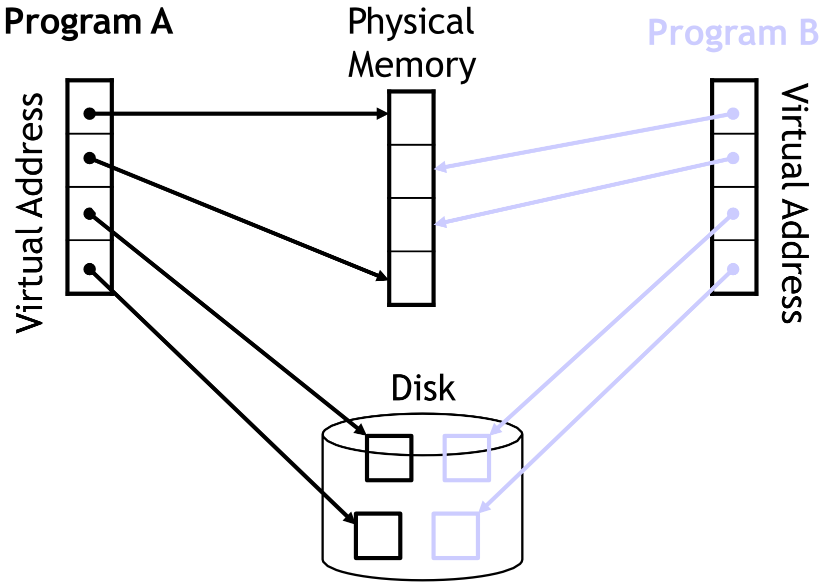
With Virtual Memory

Program Address map to RAM Address

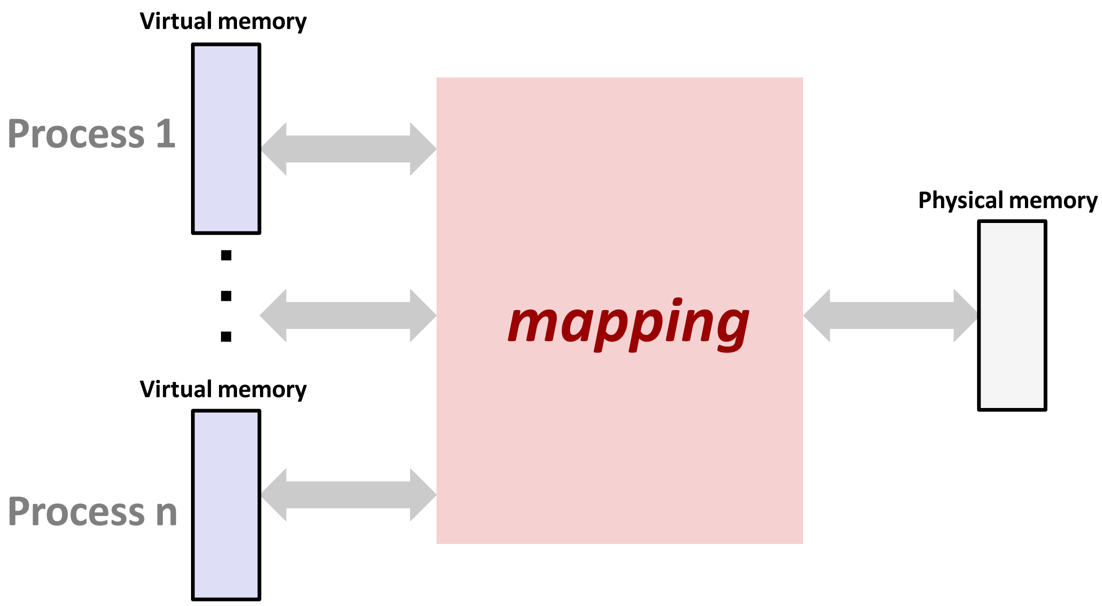


Solving the problems: #3 keeping program secure

- Because different processes will have different mappings from virtual to physical addresses, two programs can freely use the same virtual address.
- By allocating distinct regions of physical memory to A and B, they are prevented from reading/writing each others data.



Solution: Level Of Indirection



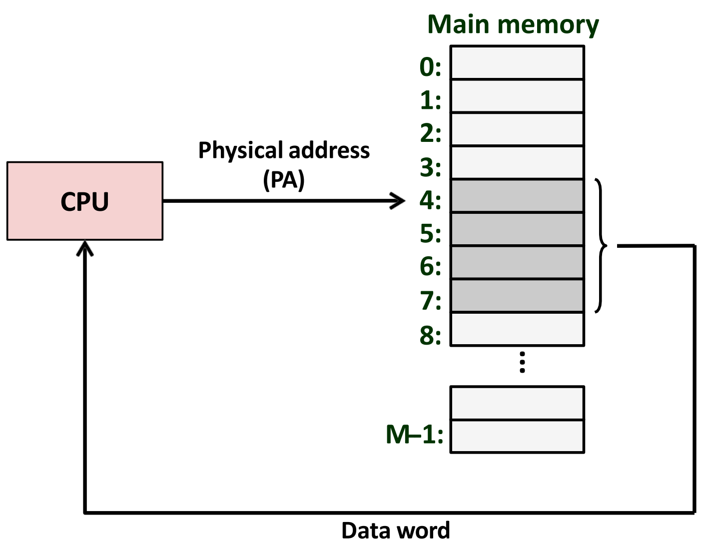
- **Each process gets its own private memory space**
- **Solves the previous problems**

How does VM work?

- Basic idea: separate memory spaces
  - Virtual memory: what the program sees
  - Physical memory: the physical RAM in the computer

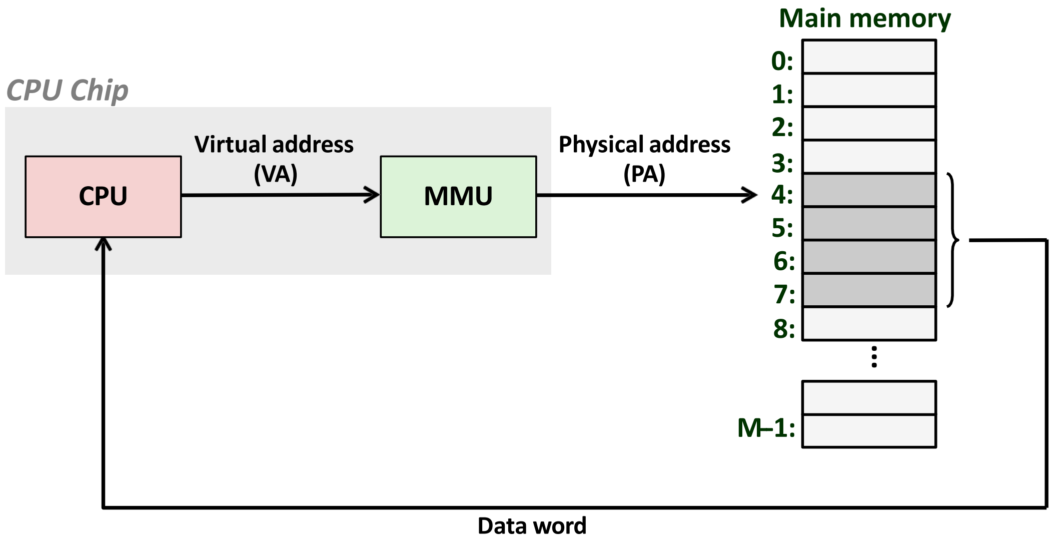
- Virtual Addresses (VA):
  - What the program uses
  - In MIPS, this is the full 32-bit address space: 0 to $2^{32} - 1$
- Physical Address (PA):
  - What the hardware uses to talk to the RAM
  - Address space determined by how much RAM is installed

A System Using Physical Addressing



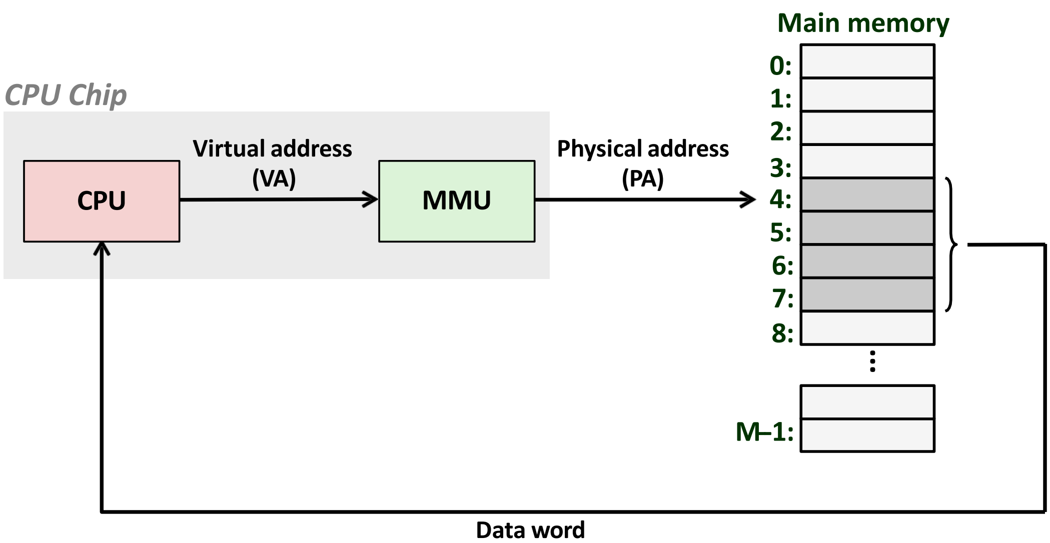
Used in "simple" systems like embedded microcontrollers in devices like cars, elevators, and digital picture frames.

A System Using Virtual Addressing



- Used in all modern desktops, laptops, workstations
- One of the great ideas in computer science

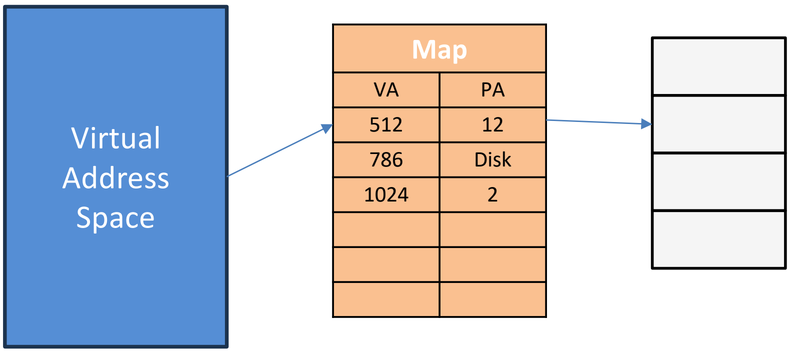
A System Using Virtual Addressing



How would you do the VA -> PA translation?

Page tables

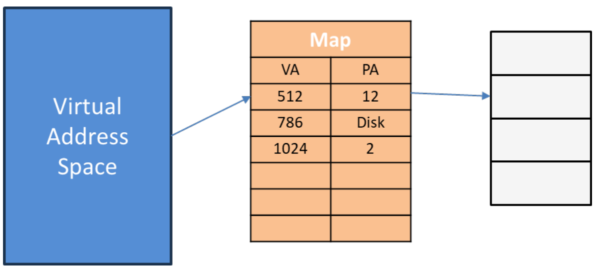
- The map from VA to PA is the page table
- So far we have had one Page Table Entry (PTE) for every Virtual Address
- How many entries do we need in our page table?



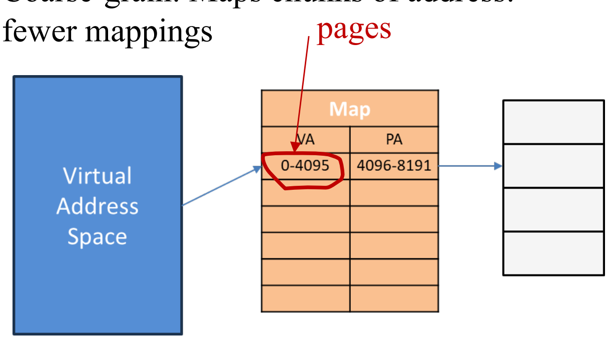
Page tables size

- We need to translate every possible address:
- One program has 32-bit Virtual Memory spaces
  - That's 2^{30} words that need Page Table Entries (1 billion entries!)
  - If they don't have a Page Table Entry then we can't access them because we can't find the physical address.
- How can we make this more manageable?
  - What if we divided memory up into chunks (pages) instead of words?

Fine-grain: Maps each word address: 2^{30} words

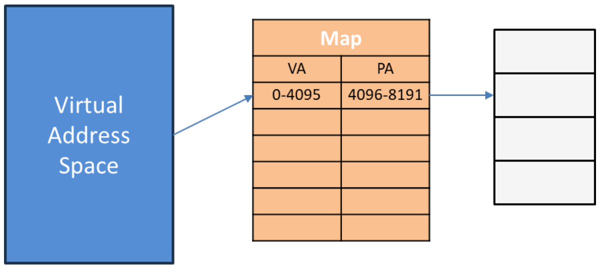


Coarse-grain: Maps chunks of address: fewer mappings



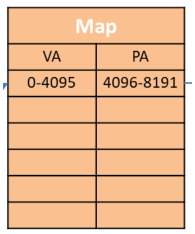
Coarse-grained: pages instead of words

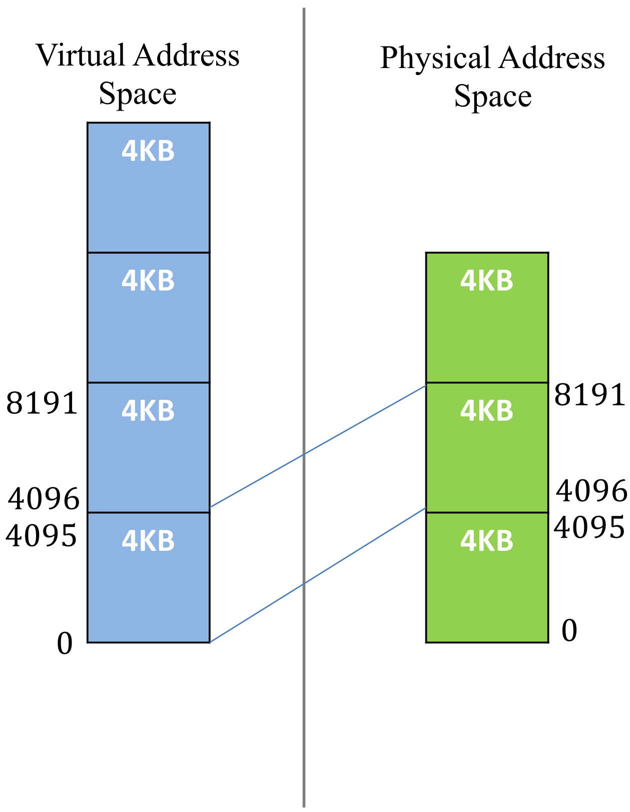
- The page table manages larger chunk (page) of data
  - Fewer Page Table Entries needed to cover the whole address space
  - But, less flexibility in how to use the RAM (have to move a page at a time)
- Today:
  - Typically, 4kB pages (1024 words per page)
    - Question: How many entries do we need in our Page Table with 4kB pages on a 32-bit machine?
  - Sometimes 2MB pages
    - Question: How many entries do we need in our Page Table with 4kB pages on a 32-bit machine?



How do we map address with pages?

Coarse-grain: Maps chunks of address: fewer mappings

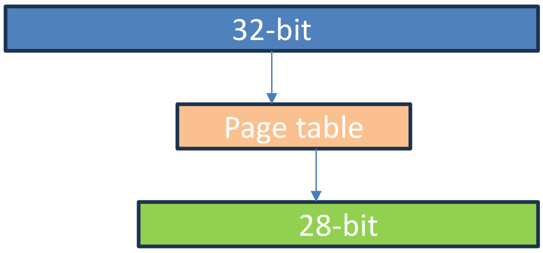




Question: What is the Physical Address for Virtual Address 4?

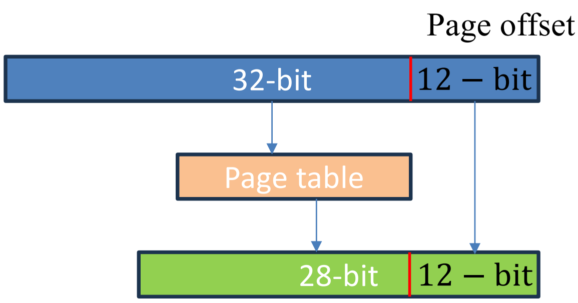
Address translation

- What happens on a 32-bit machine with 256MB RAM and 4kB pages?
  - 32-bit Virtual Address
  - 28-bit Physical Address



Address translation

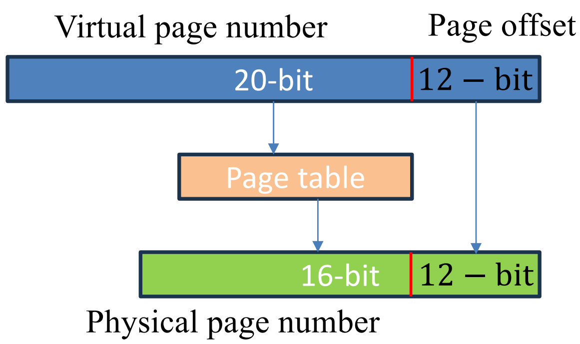
- What happens on a 32-bit machine with 256MB RAM and 4kB pages?
  - 32-bit Virtual Address
  - 28-bit Physical Address



Each PTE contains 4kB of address space…
For every page we have 4096 addresses (12 bits) that don't get translated

Address translation

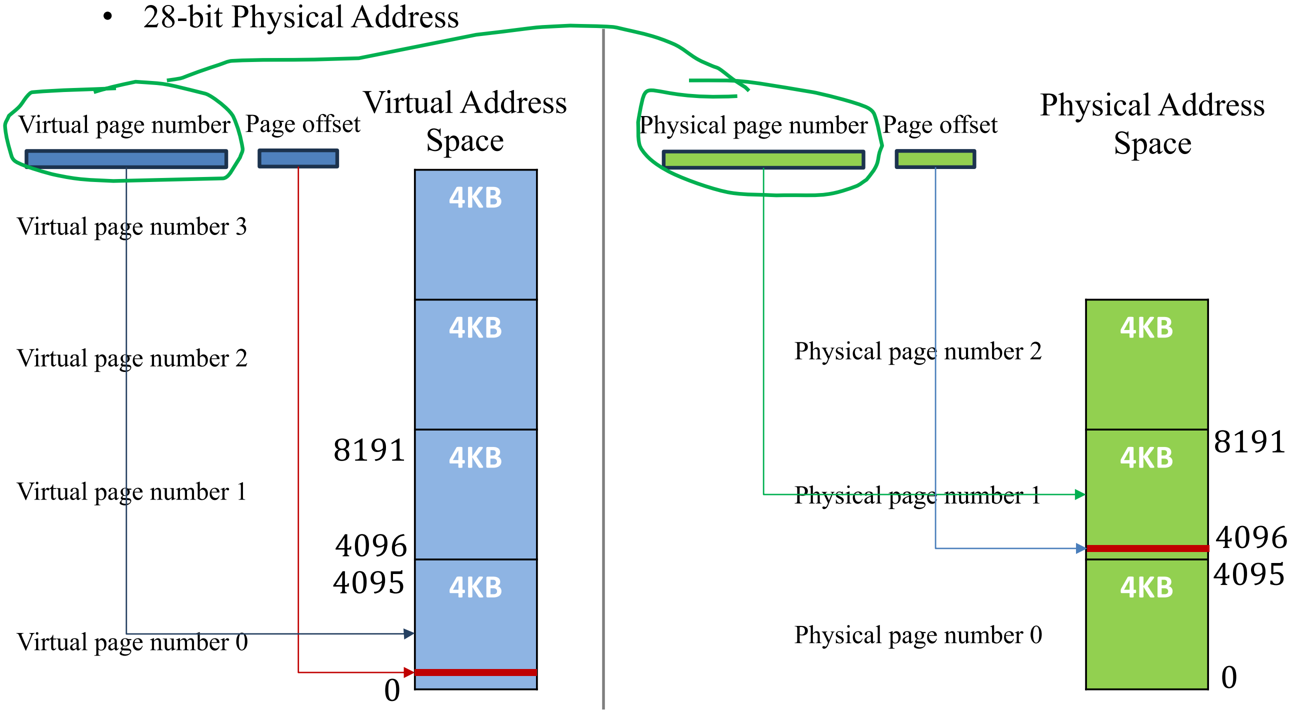
- What happens on a 32-bit machine with 256MB RAM and 4kB pages?
  - 32-bit Virtual Address
  - 28-bit Physical Address



But each PTE contains 4kB of address space…
For every page we have 4096 addresses (12 bits) that don't get translated

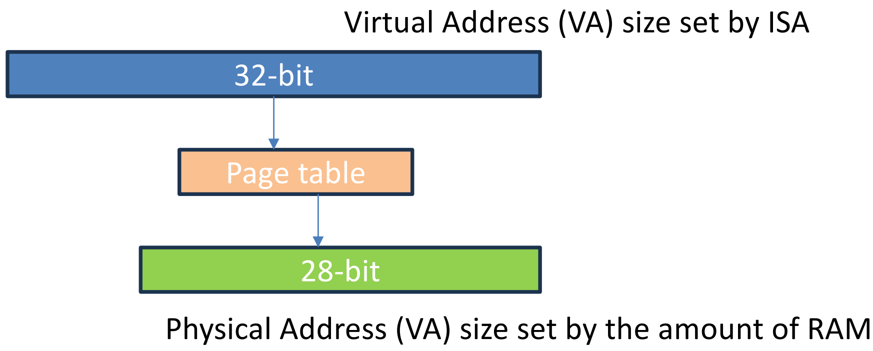
Pages, offsets, and translation

- What happens on a 32-bit machine with 256MB RAM and 4kB pages?
  - 32-bit Virtual Address
  - 28-bit Physical Address



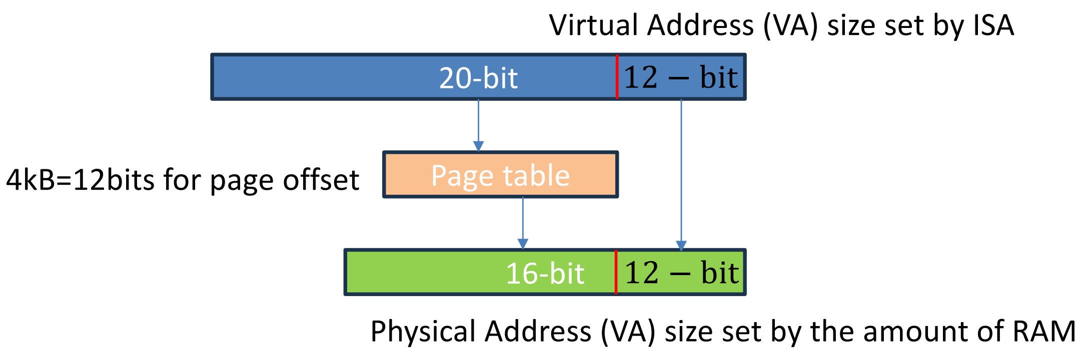
How to do a page table lookup?

- What happens on a 32-bit machine with 256MB RAM and 4kB pages?
  - 32-bit Virtual Address
  - 28-bit Physical Address



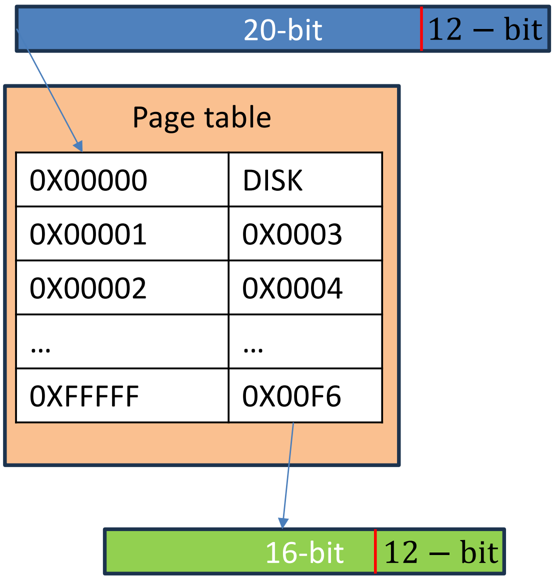
How to do a page table lookup?

- What happens on a 32-bit machine with 256MB RAM and 4kB pages?
  - 32-bit Virtual Address
  - 28-bit Physical Address



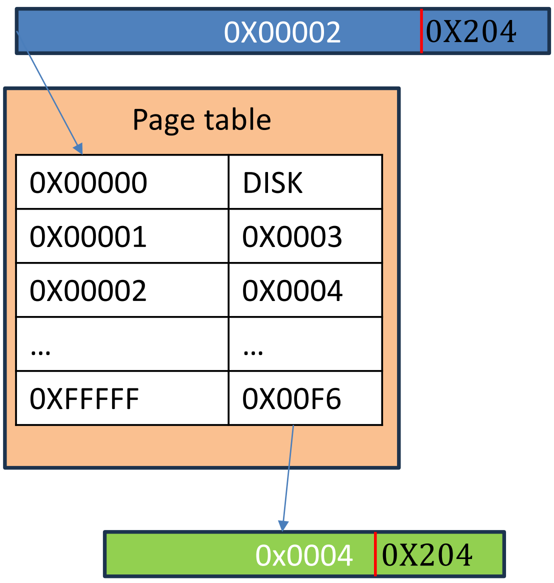
How to do a page table lookup?

- What happens on a 32-bit machine with 256MB RAM and 4kB pages?
  - 32-bit Virtual Address
  - 28-bit Physical Address



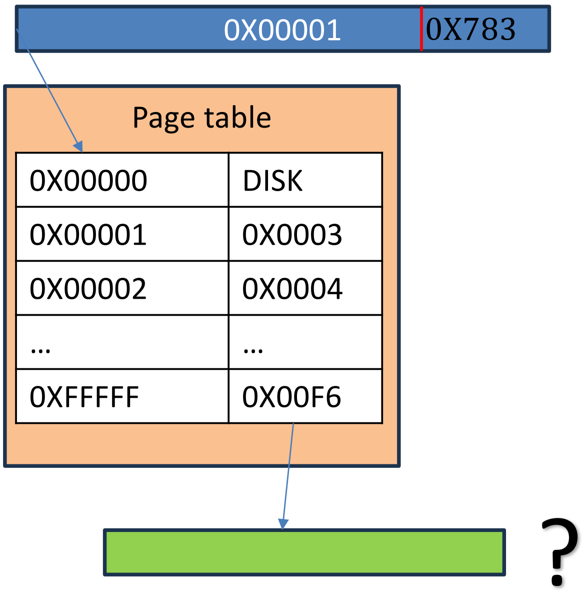
Example Translation

- What happens on a 32-bit machine with 256MB RAM and 4kB pages?
  - 32-bit Virtual Address
  - 28-bit Physical Address



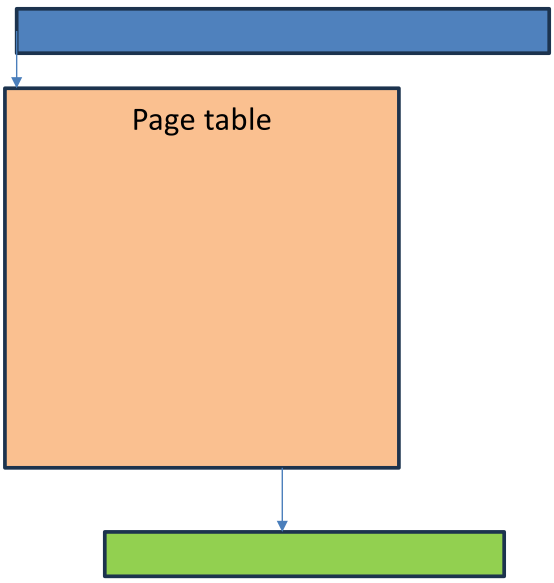
Question

- What happens on a 32-bit machine with 256MB RAM and 4kB pages?
  - 32-bit Virtual Address
  - 28-bit Physical Address



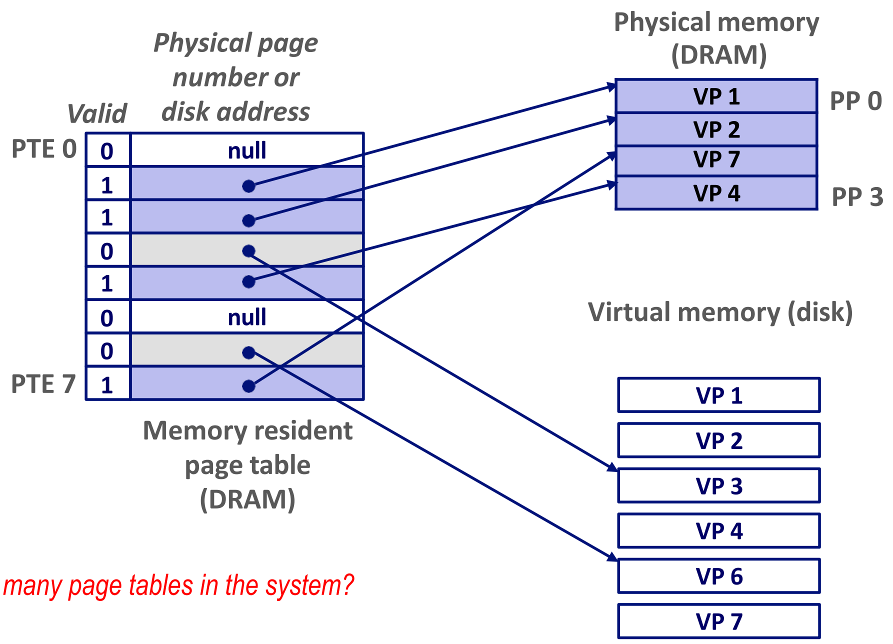
Question

- If we have 64kB pages, how many bits do we use for the page offset?



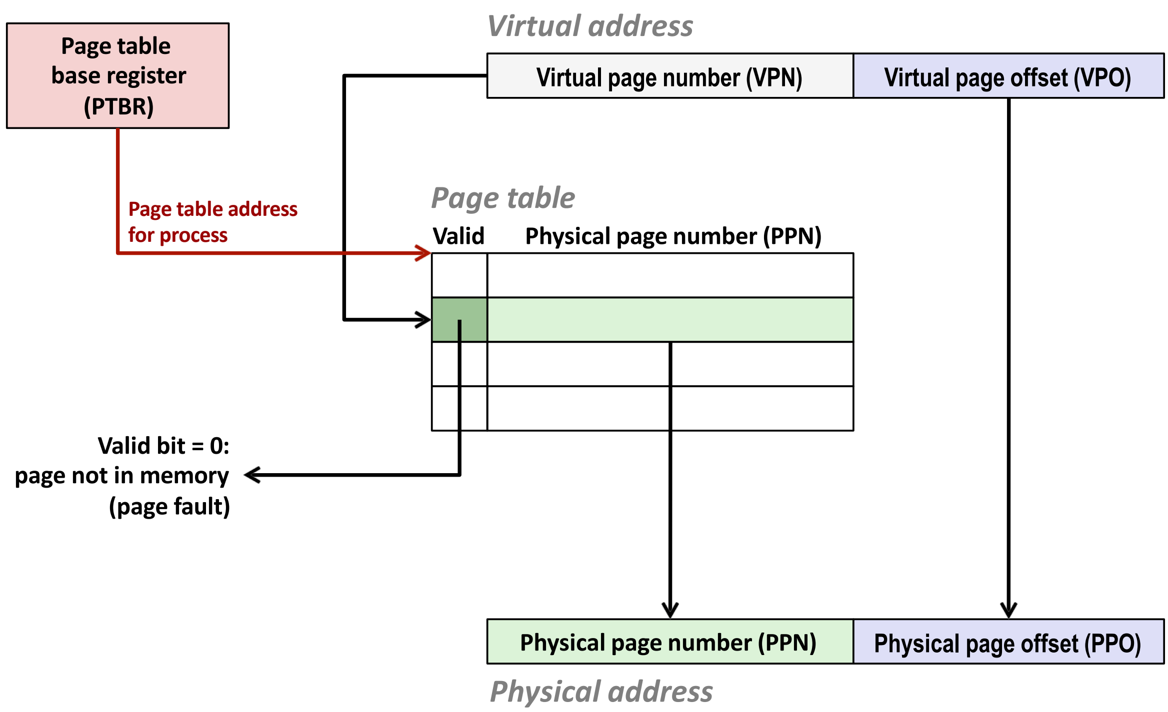
Address Translation: Page Tables

A *page table* is an array of page table entries (PTEs) that maps virtual pages to physical pages. Here: 8 VPs



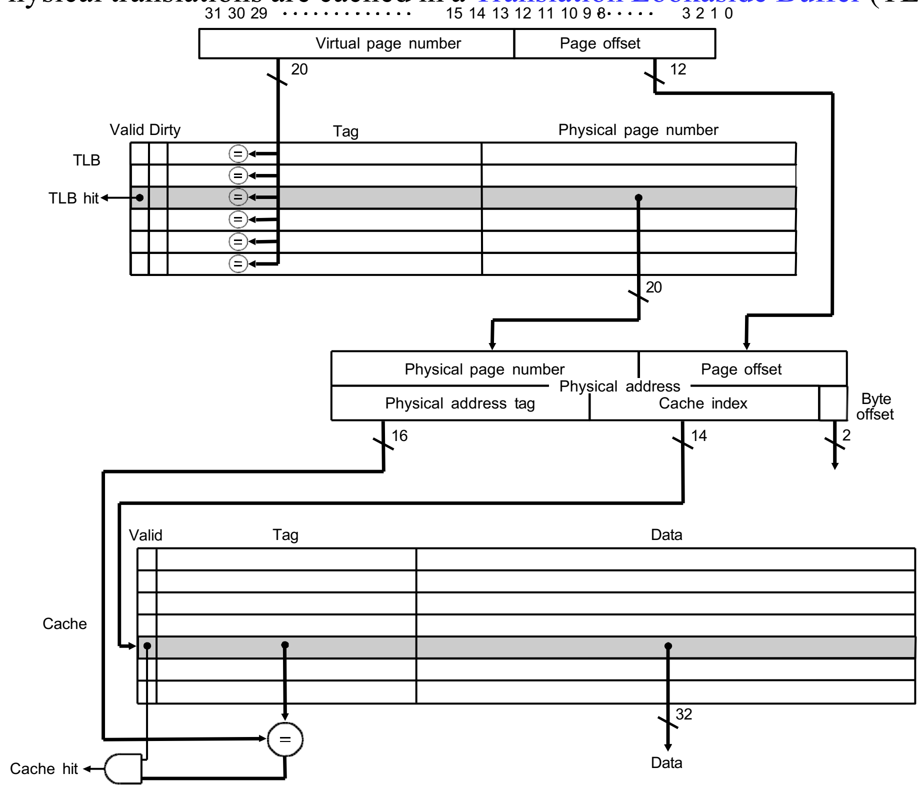
How many page tables in the system?

Address Translation With a Page Table



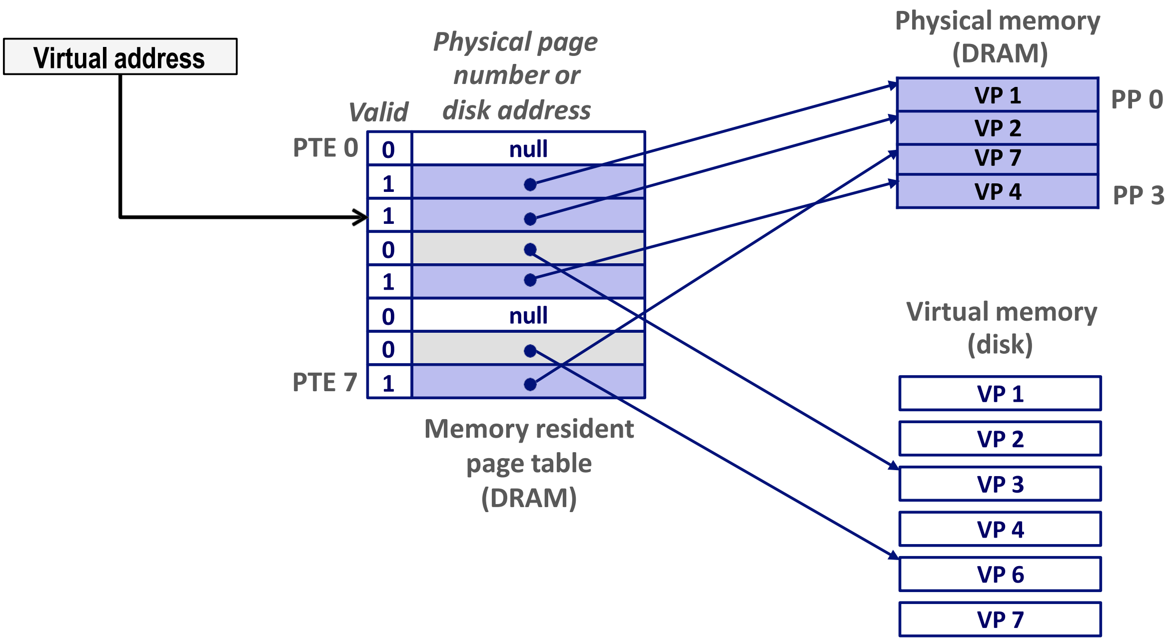
Caching Translations

- Virtual to Physical translations are cached in a Translation Lookaside Buffer (TLB).



Page Hit

Page hit: **reference to VM word that is in physical memory**



Page Miss

Page miss: **reference to VM word that is not in physical memory (shOOt!)**