

# CS 110

# Computer Architecture

## *Virtual Memory*

Instructor:

Sören Schwertfeger and Chundong Wang

<https://robotics.shanghaitech.edu.cn/courses/ca/20s>

School of Information Science and Technology SIST

ShanghaiTech University

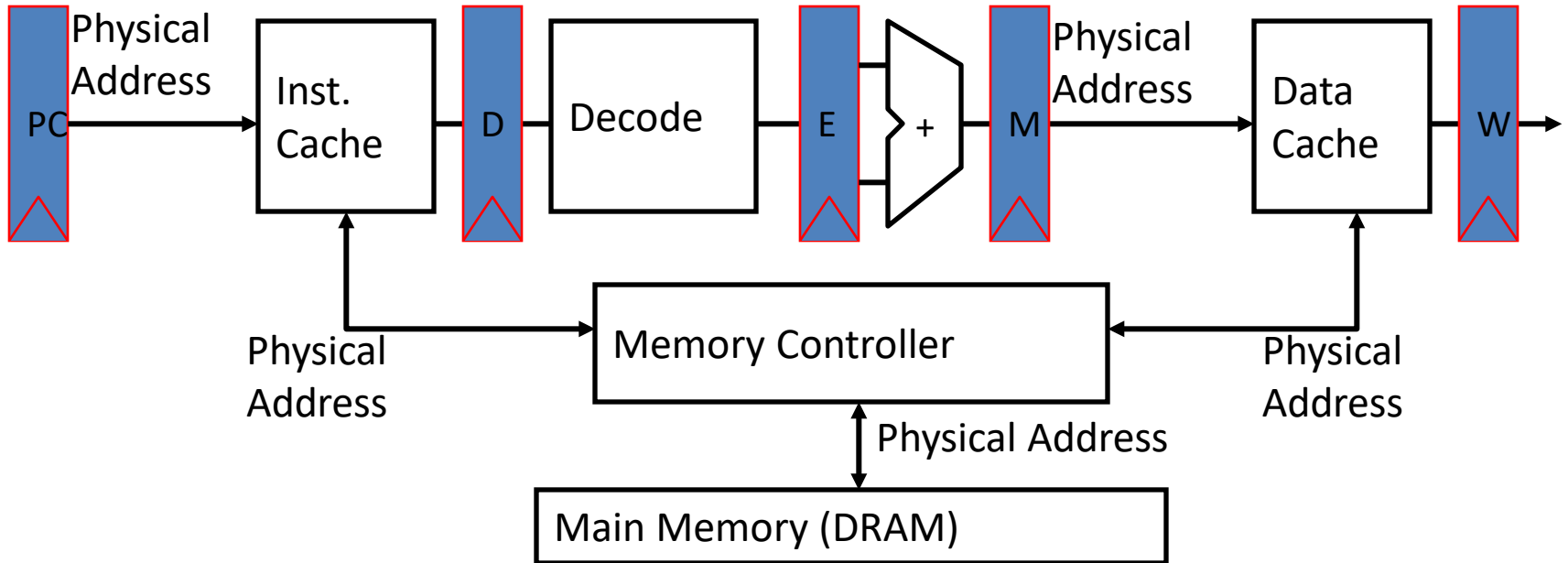
Slides based on UC Berkeley's CS61C

# Review

- Booting a Computer
  - BIOS, Bootloader, OS Boot, Init
- Supervisor Mode, Syscalls
- Memory-mapped I/O
- Polling vs. Interrupts, interrupt vs. exception
- Intro to VM
- Base and Bound registers for VM
  - Simple, but doesn't give us everything we want

# Virtual Memory

# “Bare” 5-Stage Pipeline

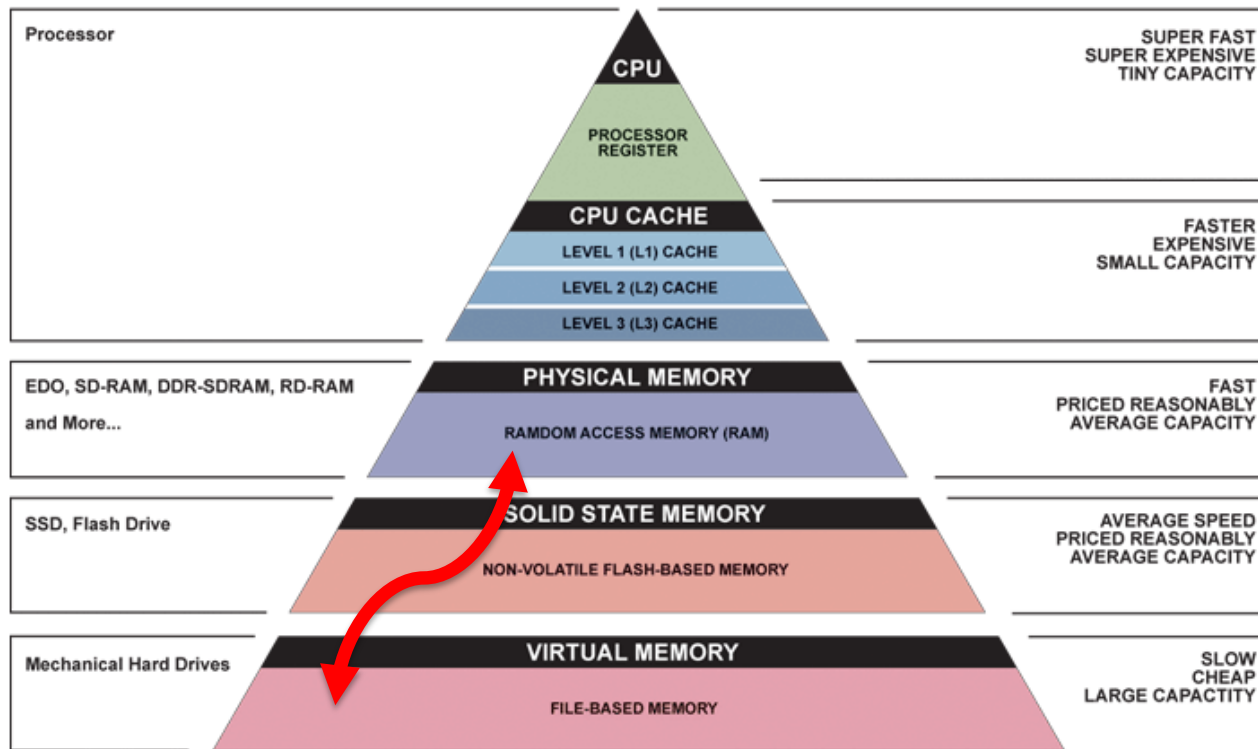


- In a bare machine, the only kind of address is a physical address

# What do we need Virtual Memory for?

## Reason 1: Adding Disks to Hierarchy

- Need to devise a mechanism to “connect” memory and disk in the memory hierarchy

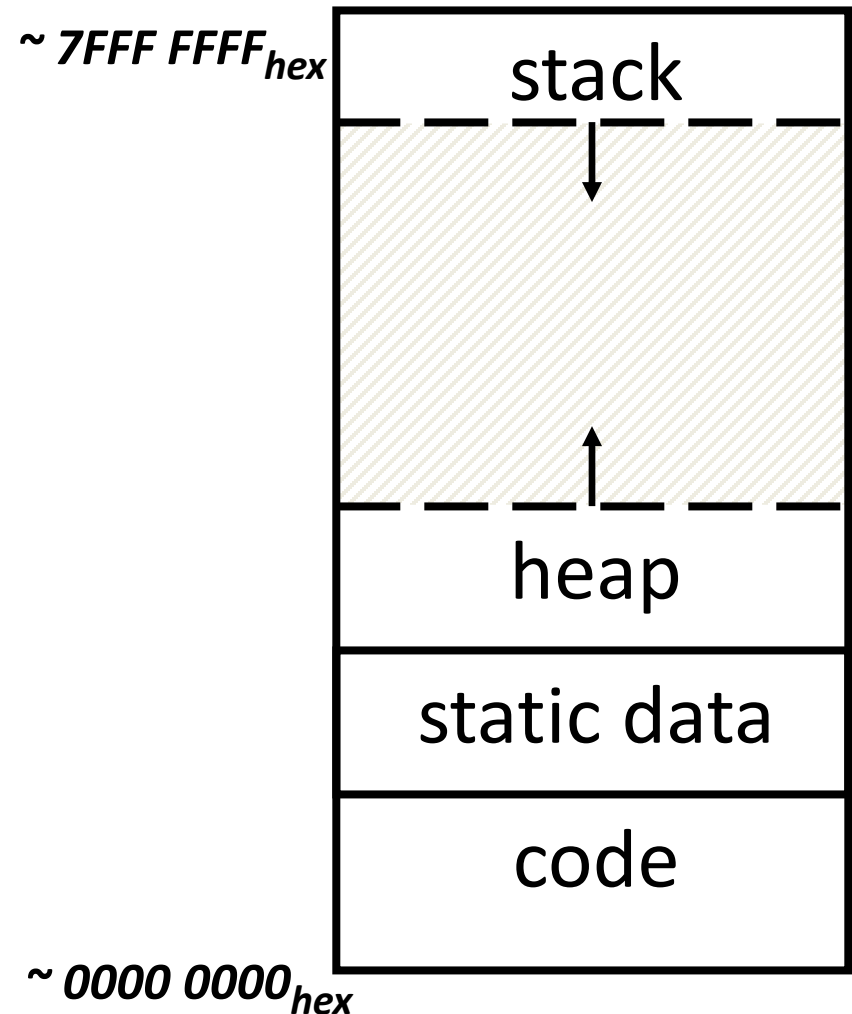


▲ Simplified Computer Memory Hierarchy  
Illustration: Ryan J. Leng

# What do we need Virtual Memory for?

## Reason 2: Simplifying Memory for Apps

- Applications should see the straightforward memory layout we saw earlier ->
- User-space applications should think they own all of memory
- So we give them a **virtual** view of memory



# What do we need Virtual Memory for?

## Reason 3: Protection Between Processes

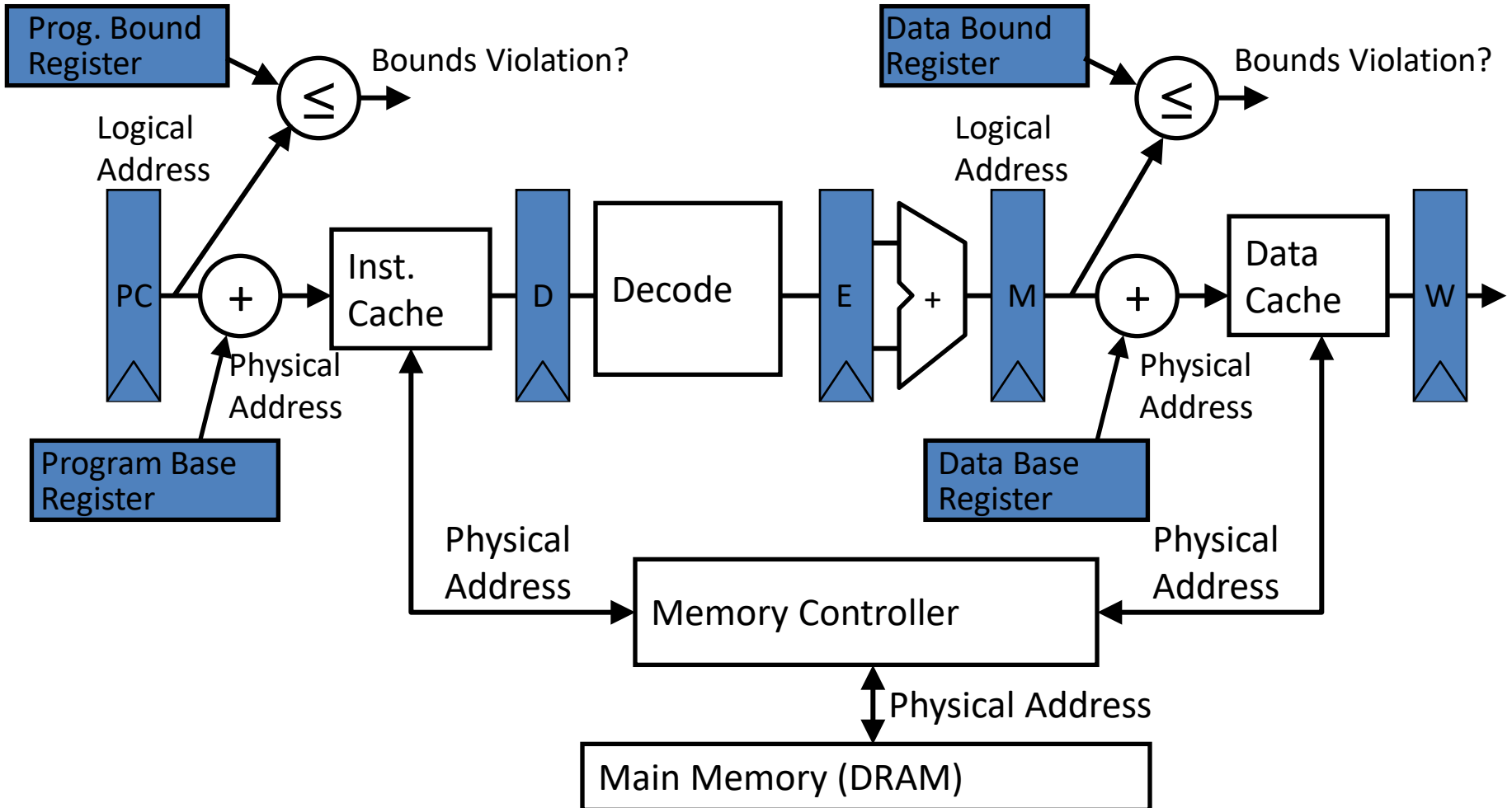
- With a bare system, addresses issued with loads/stores are real **physical** addresses
- This means any program can issue any address, therefore can access any part of memory, even areas which it doesn't own
  - Ex: The OS data structures
- We should send all addresses through a mechanism that the OS controls, before they make it out to DRAM - a **translation mechanism**

# Address Spaces

- The set of addresses labeling all of memory that we can access
- Now, 2 kinds:
  - **Virtual Address Space** - the set of addresses that the user program knows about
  - **Physical Address Space** - the set of addresses that map to actual physical cells in memory
    - Hidden from user applications
- So, we need a way to map between these two address spaces

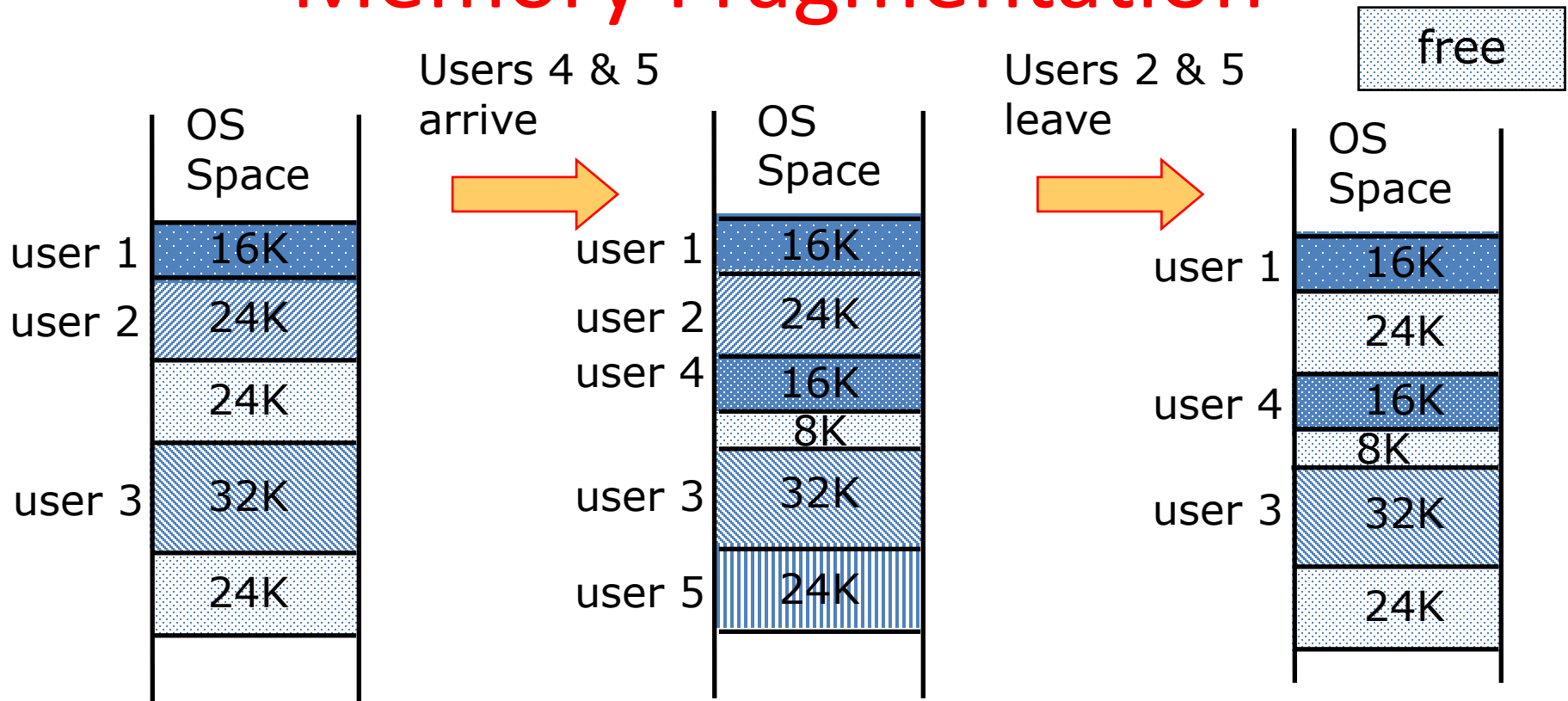


# Base and Bound Machine



[ Can fold addition of base register into (register+immediate) address calculation using a carry-save adder (sums three numbers with only a few gate delays more than adding two numbers) ]

# Memory Fragmentation



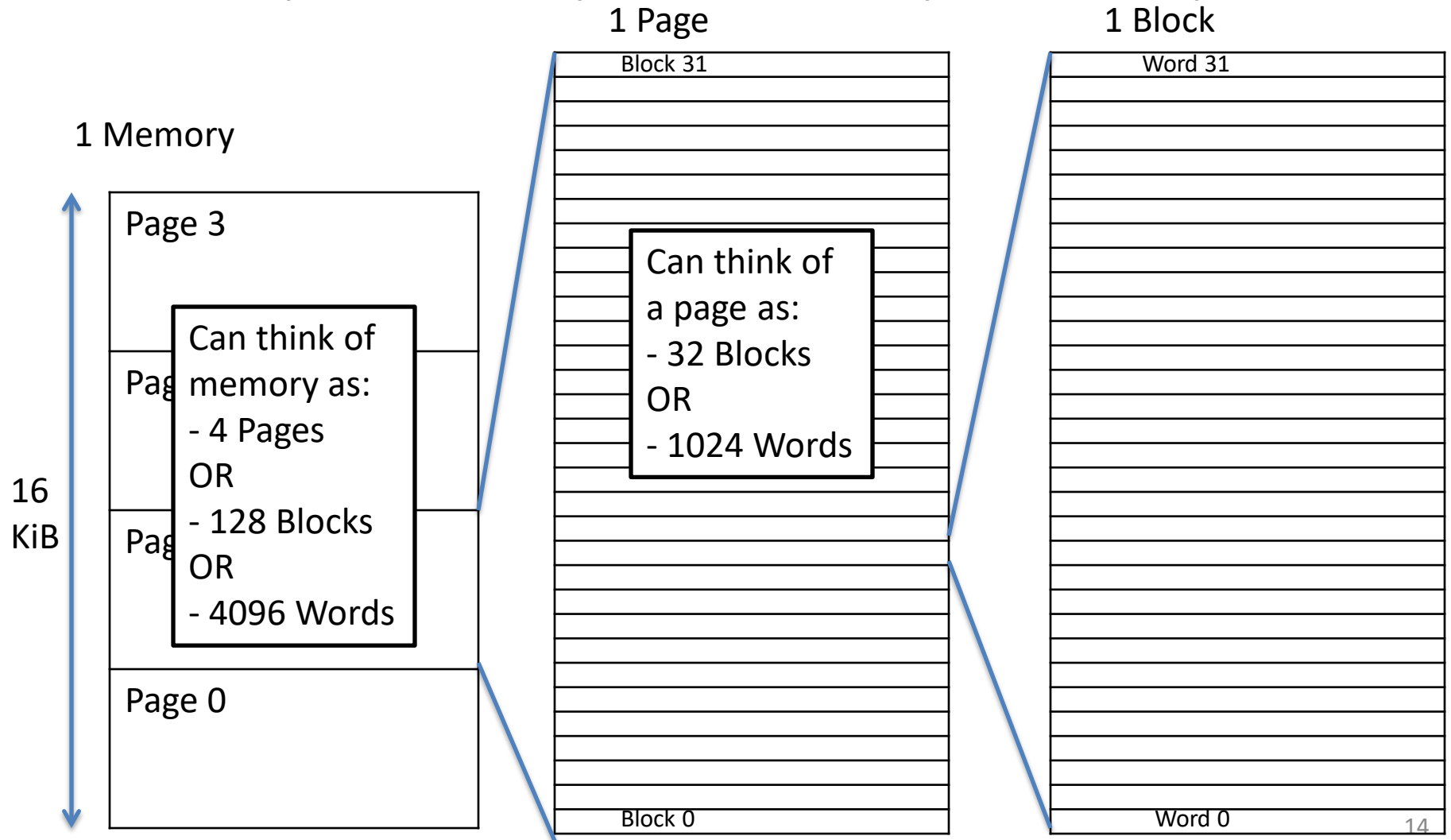
As users come and go, the storage is “fragmented”. Therefore, at some stage programs have to be moved around to compact the storage.

# Blocks vs. Pages

- In caches, we dealt with individual *blocks*
  - Usually ~64B on modern systems
  - We could “divide” memory into a set of blocks
- In VM, we deal with individual *pages*
  - Usually ~4 KB on modern systems
    - Larger sizes also available: 4MB, very modern 1GB!
  - Now, we’ll “divide” memory into a set of pages
- Common point of confusion: Bytes, Words, Blocks, Pages are all just different ways of looking at memory!

# Bytes, Words, Blocks, Pages

Ex: 16 KiB DRAM, 4 KiB Pages (for VM), 128 B blocks (for caches), 4 B words (for lw/sw)



# Address Translation

- So, what do we want to achieve at the hardware level?
  - Take a Virtual Address, that points to a spot in the Virtual Address Space of a particular program, and map it to a Physical Address, which points to a physical spot in DRAM of the whole machine

Virtual Address

Virtual Page Number

Offset

Physical Address

Physical Page Number

Offset

# Address Translation

Virtual Address

Virtual Page Number

Offset

Address  
Translation

Copy  
Bits

Physical Address

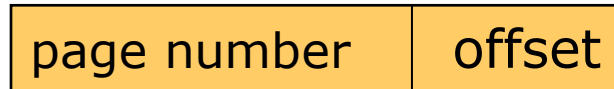
Physical Page Number

Offset

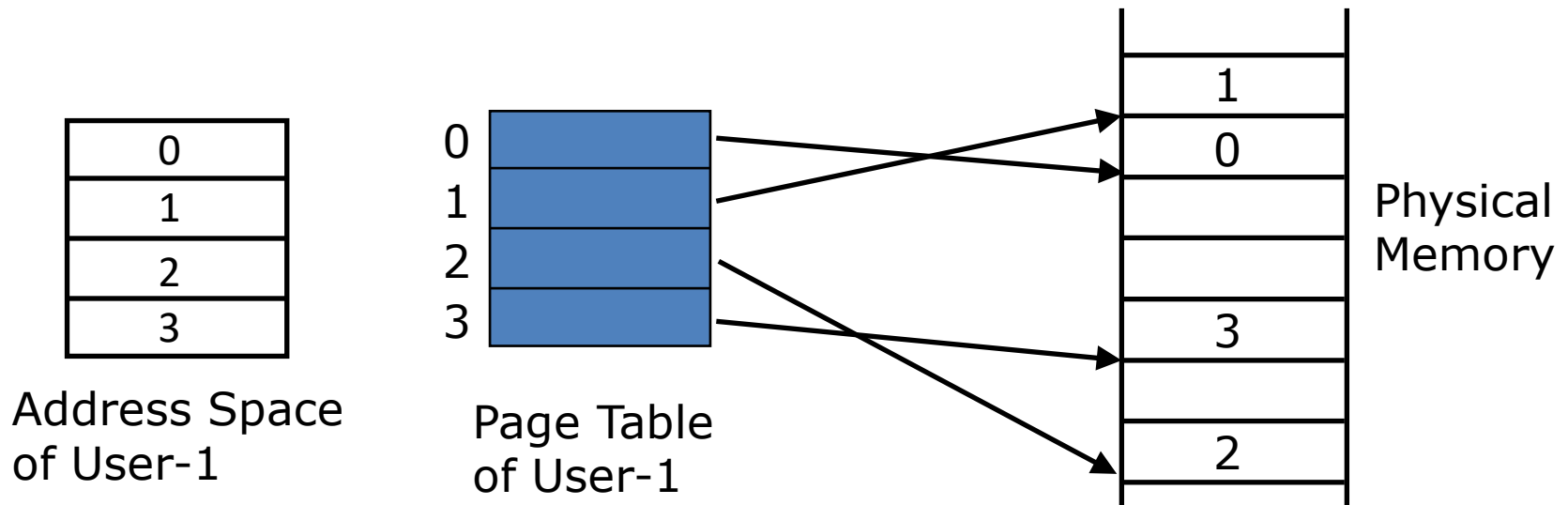
The rest of the lecture is all about implementing

# Paged Memory Systems

- Processor-generated address can be split into:

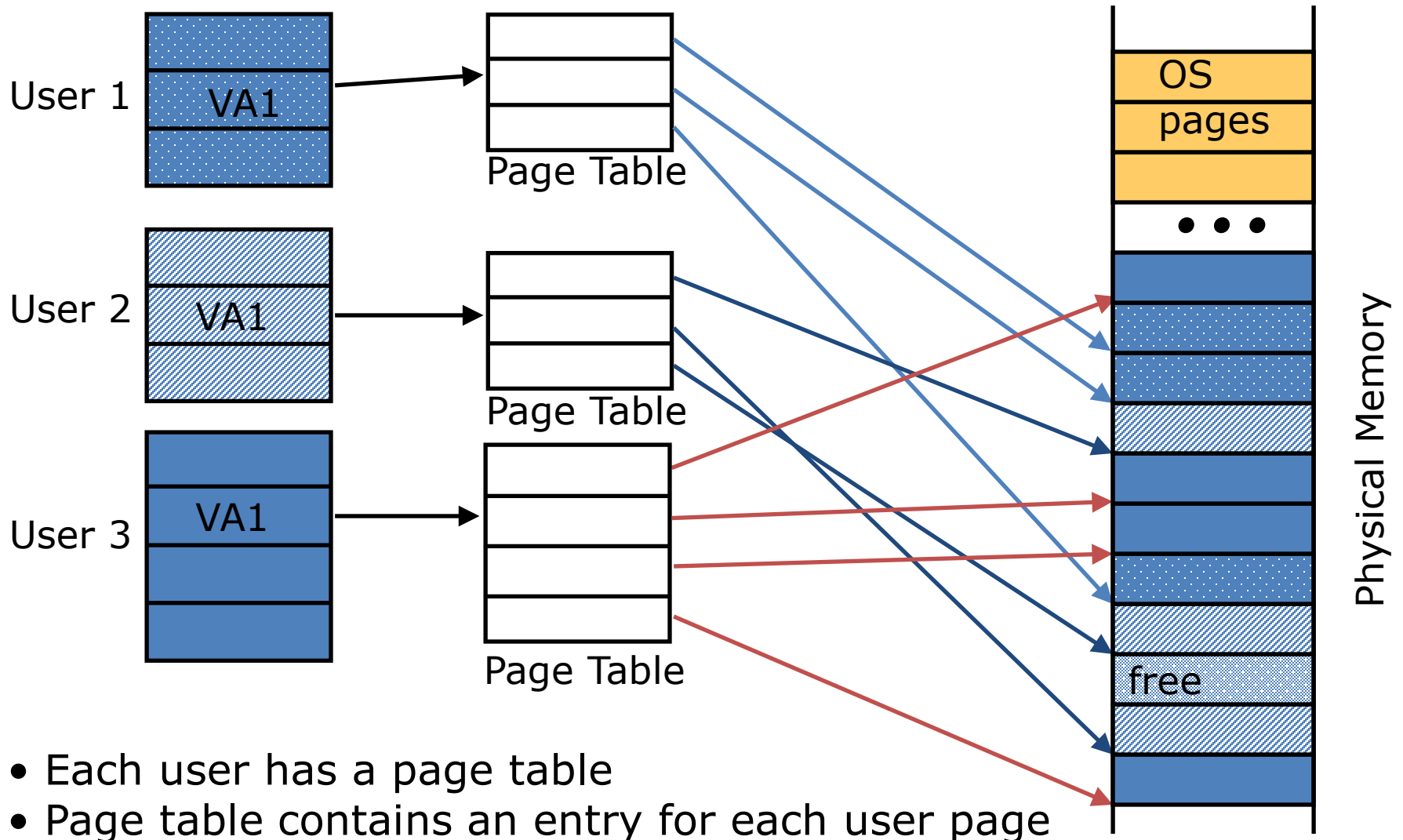


- A page table contains the physical address of the base of each page



*Page tables make it possible to store the pages of a program non-contiguously.*

# Private Address Space per User

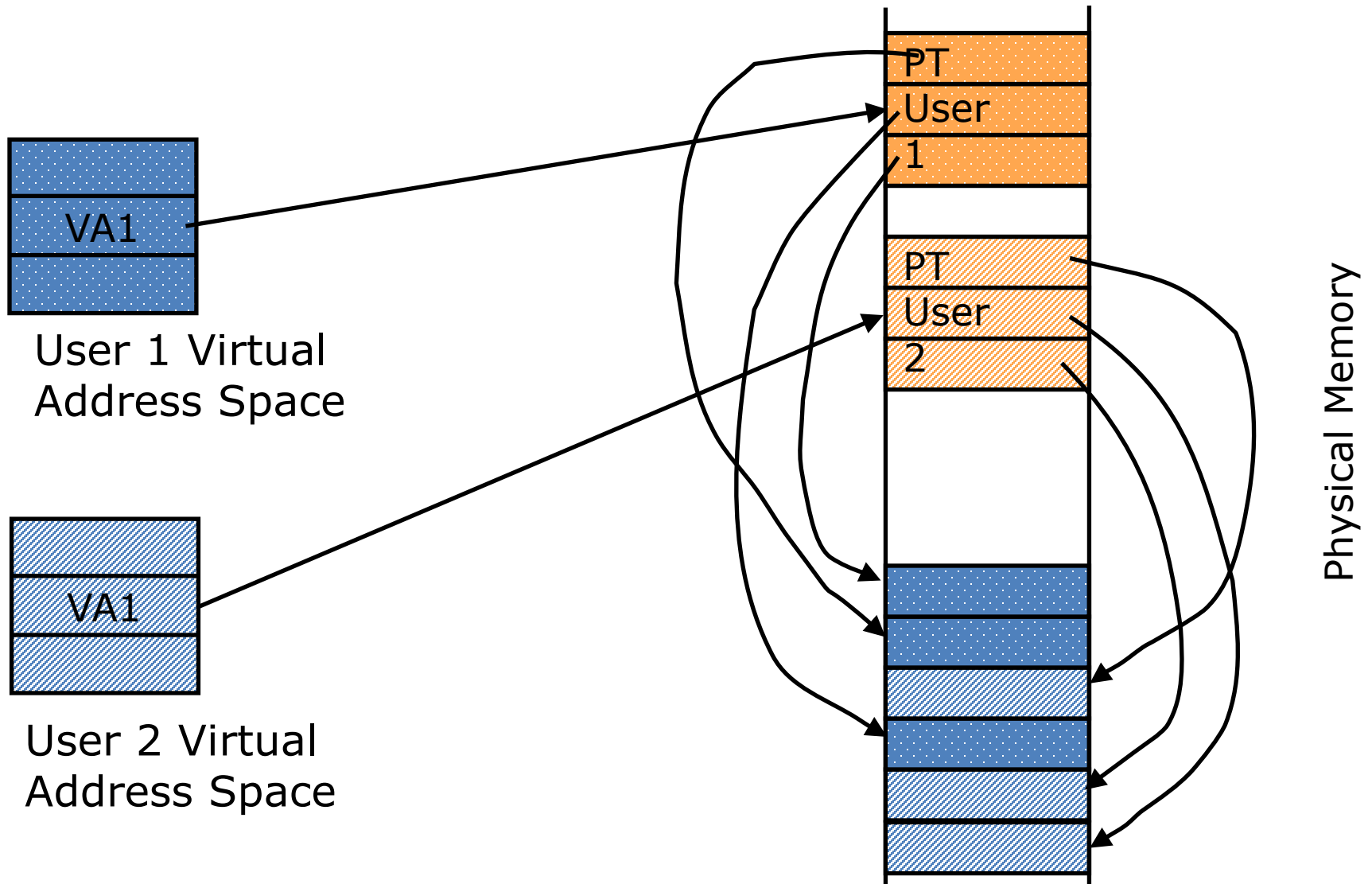




# Where Should Page Tables Reside?

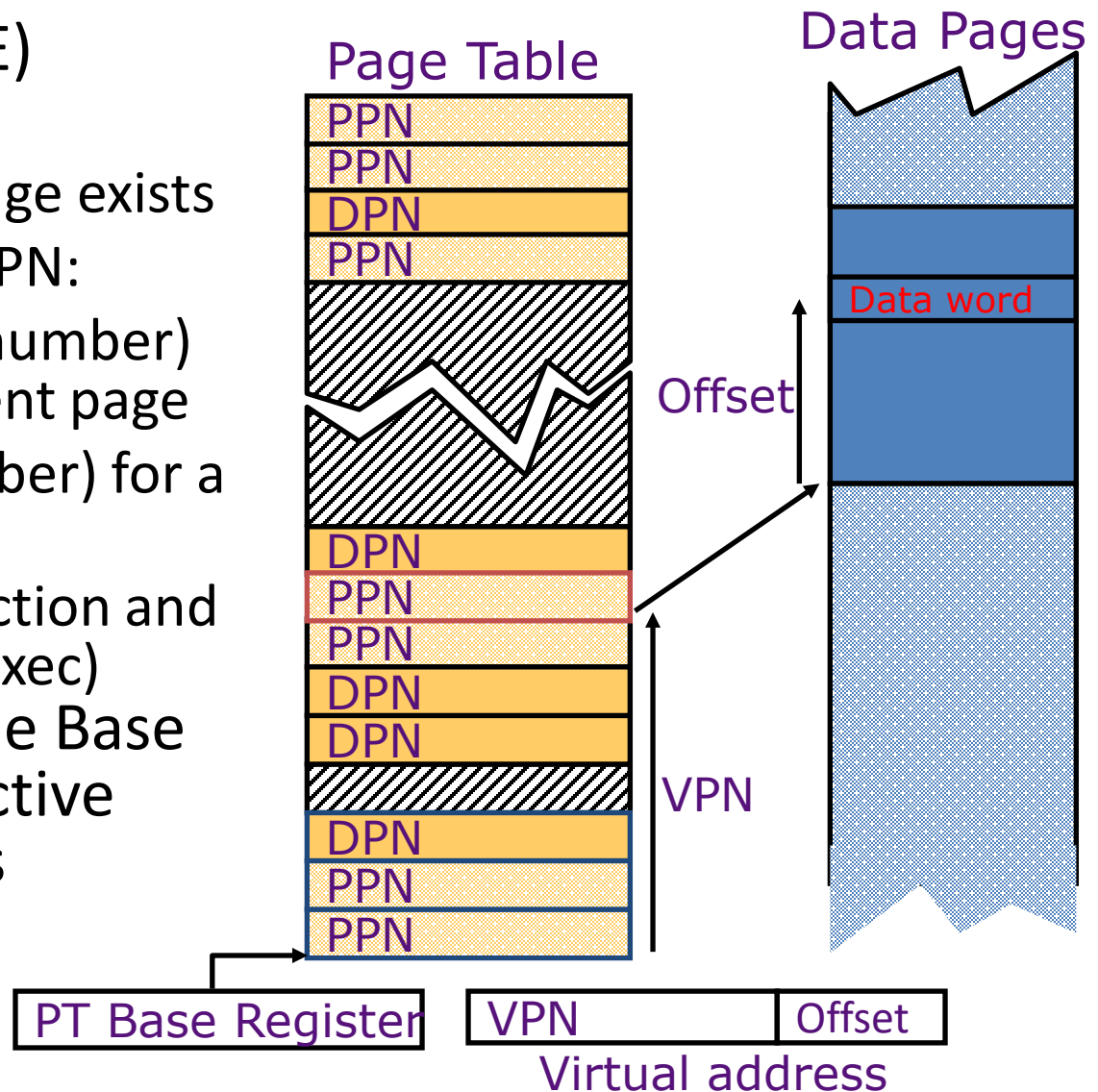
- Space required by the page tables (PT) is proportional to the address space, number of users, ...
  - ⇒ *Too large to keep in CPU registers*
- Idea: Keep PTs in the main memory
  - Needs one reference to retrieve the page base address and another to access the data word
    - ⇒ *doubles the number of memory references!*

# Page Tables in Physical Memory



# Linear (simple) Page Table

- Page Table Entry (PTE) contains:
  - 1 bit to indicate if page exists
  - And either PPN or DPN:
    - PPN (physical page number) for a memory-resident page
    - DPN (disk page number) for a page on the disk
  - Status bits for protection and usage (read, write, exec)
- OS sets the Page Table Base Register whenever active user process changes



# Suppose an instruction references a memory page that isn't in DRAM?

- We get an exception of type “page fault”
- Page fault handler does the following:
  - If virtual page doesn't yet exist, assign an unused page in DRAM, or if page exists ...
  - Initiate transfer of the page we're requesting from disk to DRAM, assigning to an unused page
  - If no unused page is left, a *page currently in DRAM is selected to be replaced* (based on usage)
  - The replaced page is written (back) to disk, page table entry that maps that VPN->PPN is marked as invalid/DPN
  - Page table entry of the page we're requesting is updated with a (now) valid PPN

# Size of Linear Page Table

With 32-bit memory addresses, 4-KB pages:

- =>  $2^{32} / 2^{12} = 2^{20}$  virtual pages per user, assuming 4-Byte PTEs,
- =>  $2^{20}$  PTEs, i.e, 4 MB page table per process!

Larger pages?

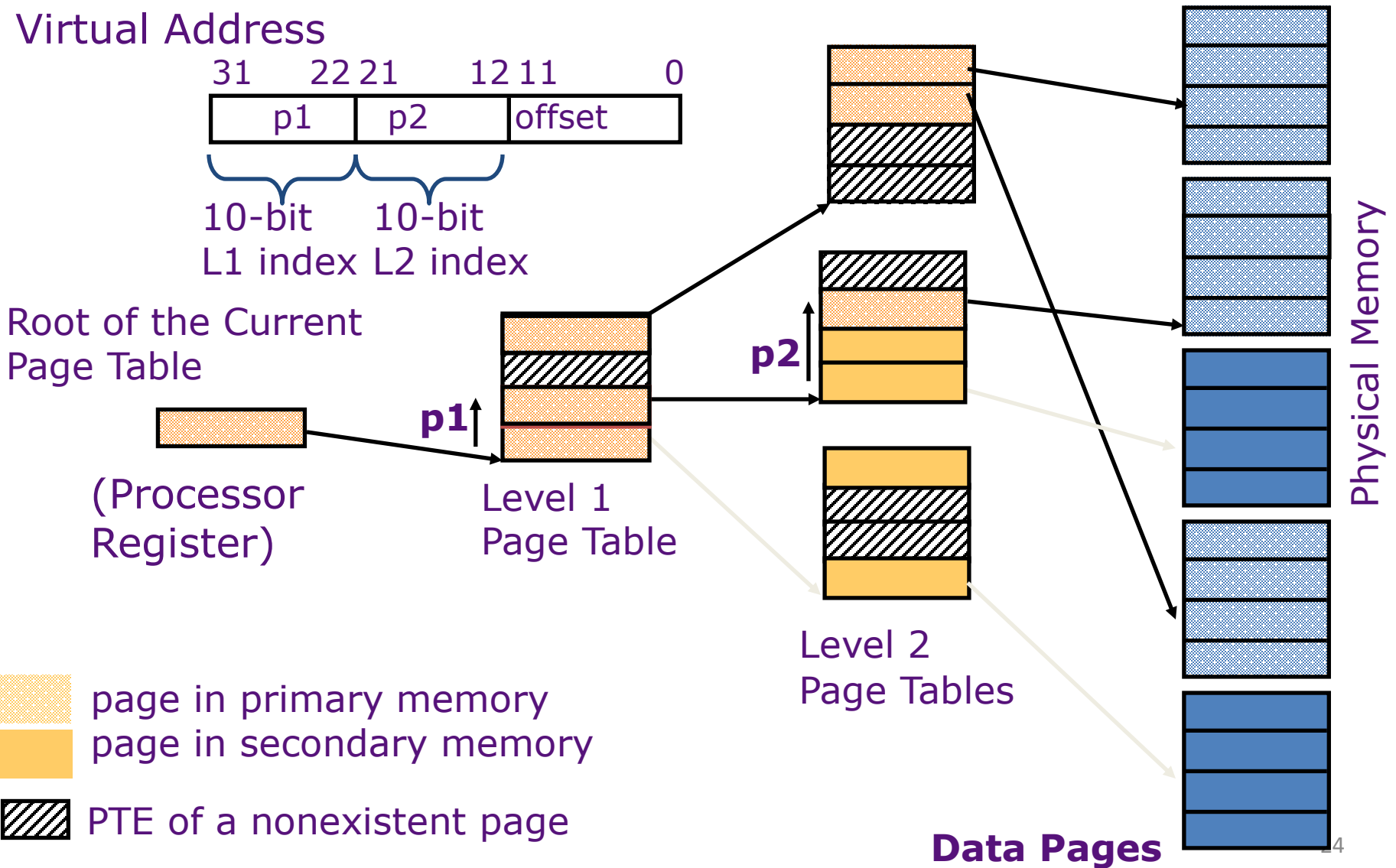
- Internal fragmentation (Not all memory in page gets used)
- Larger page fault penalty (more time to read from disk)

What about 64-bit virtual address space???

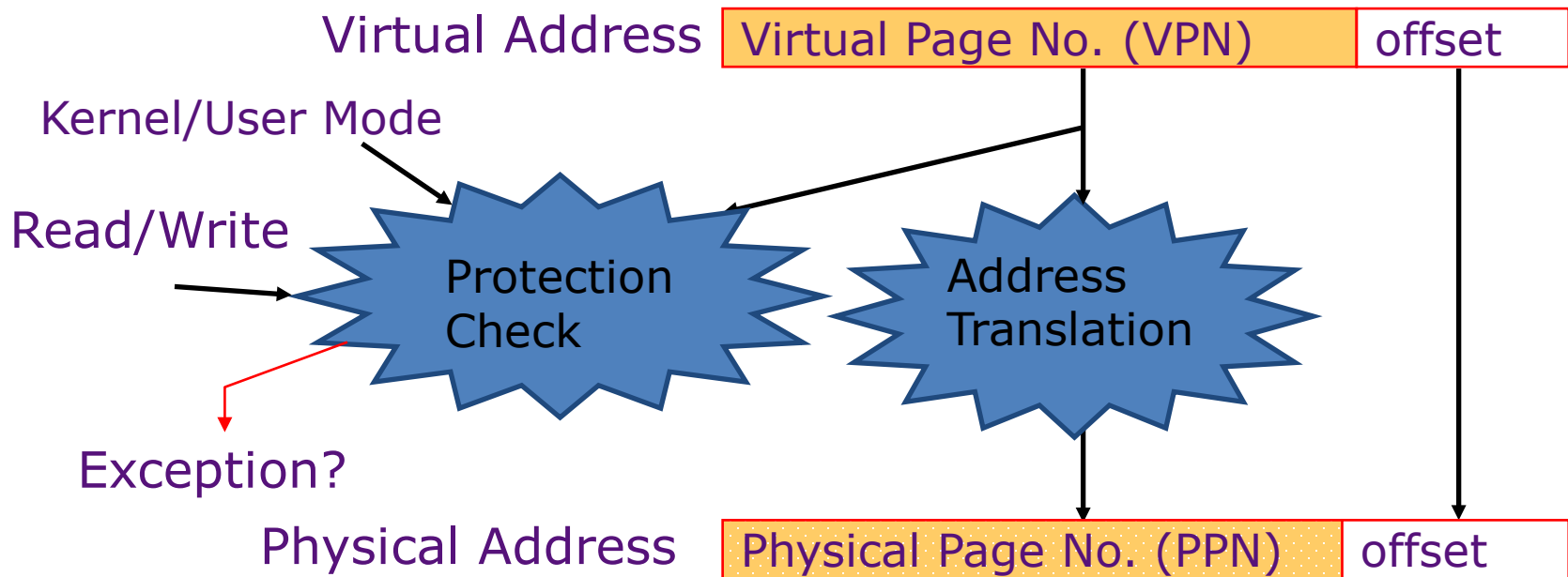
- Even 1MB pages would require  $2^{44}$  8-Byte PTEs (35 TB!)

*What is the “saving grace” ? Most processes only use a set of high address (stack), and a set of low address (instructions, heap)*

# Hierarchical Page Table – exploits sparsity of virtual address space use



# Address Translation & Protection



- Every instruction and data access needs address translation and protection checks

*A good VM design needs to be fast (~ one cycle) and space efficient*

# Translation Lookaside Buffers (TLB)

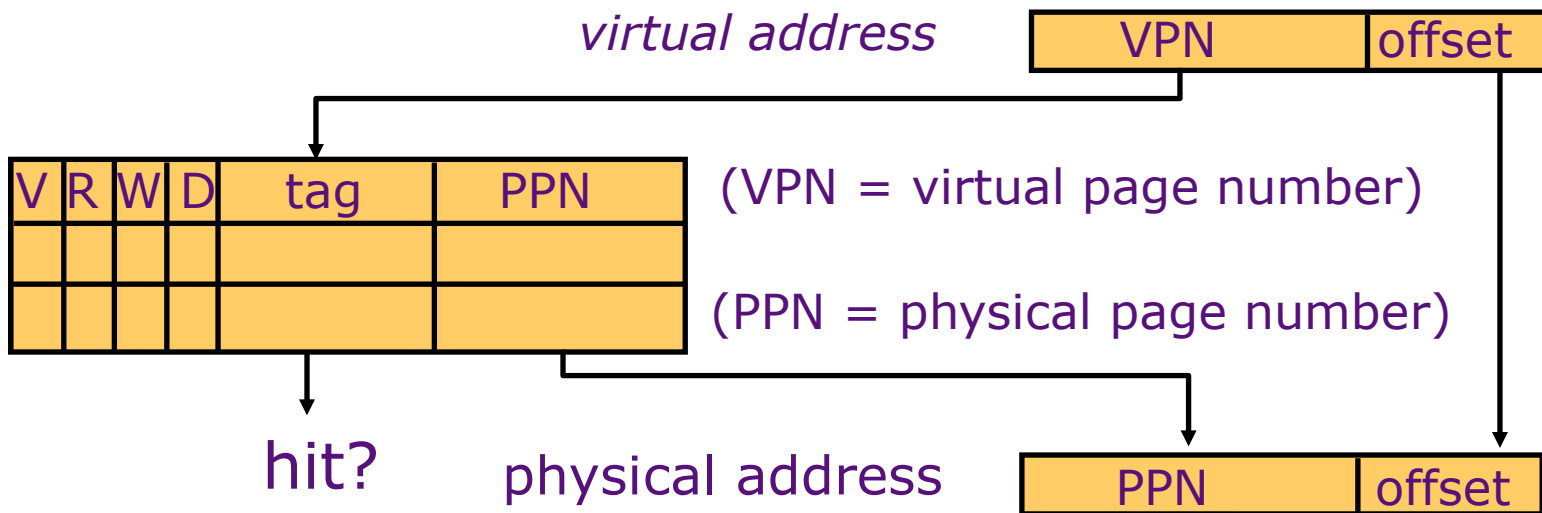
Address translation is very expensive!

In a two-level page table, each reference becomes several memory accesses

Solution: *Cache some translations in TLB*

TLB hit  $\Rightarrow$  *Single-Cycle Translation*

TLB miss  $\Rightarrow$  *Page-Table Walk to refill*

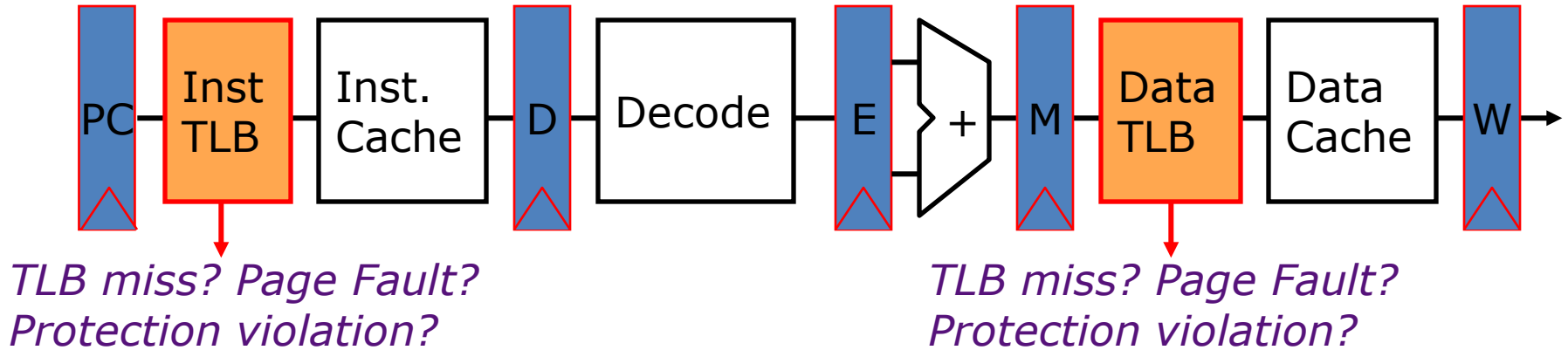




# TLB Designs

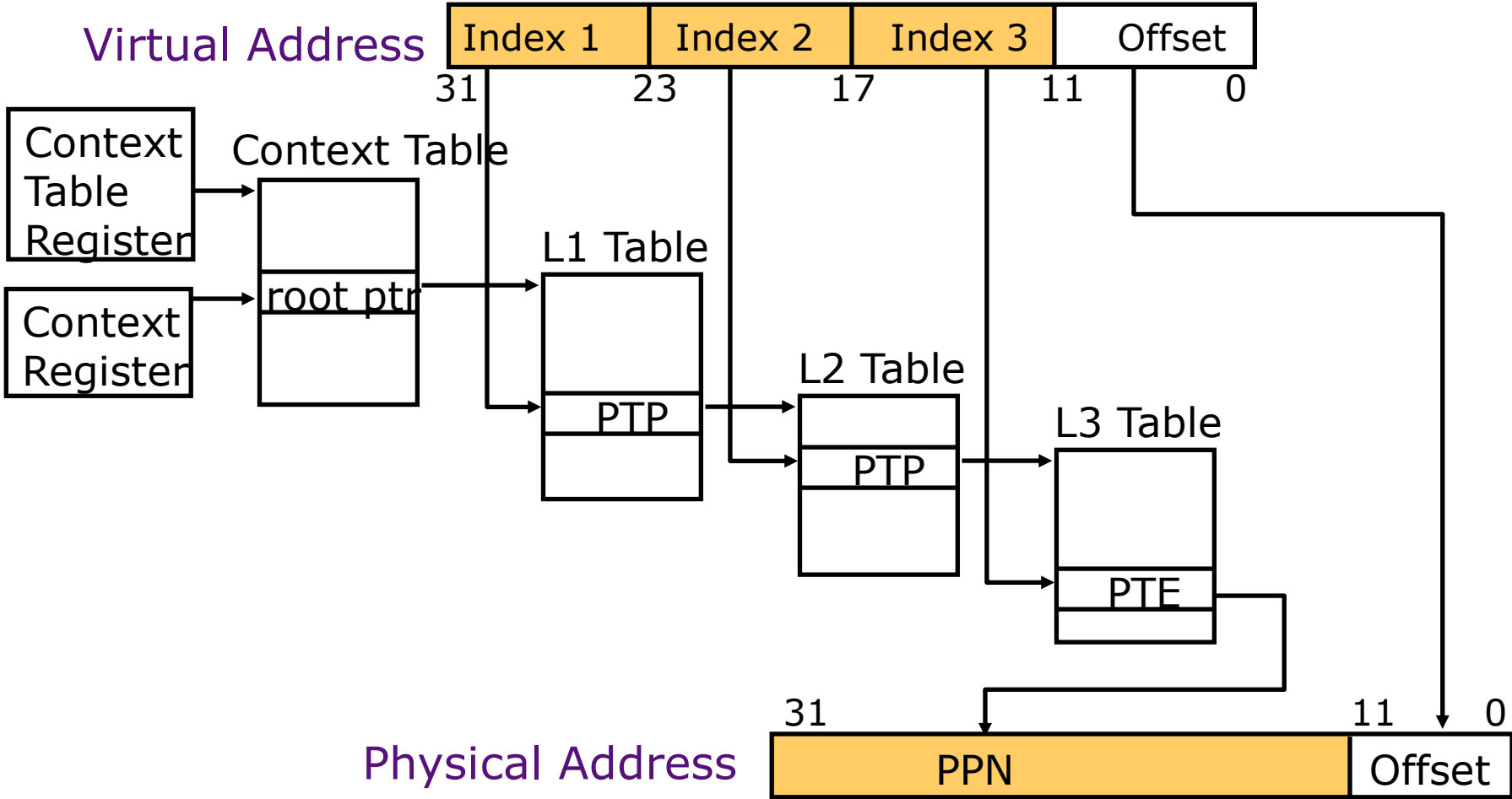
- Typically 32-128 entries, usually fully associative
  - Each entry maps a large page, hence less spatial locality across pages => more likely that two entries conflict
  - Sometimes larger TLBs (256-512 entries) are 4-8 way set-associative
  - Larger systems sometimes have multi-level (L1 and L2) TLBs
- Random or FIFO replacement policy
- Upon context switch? New VM space! Flush TLB
- ...
- “TLB Reach”: Size of largest virtual address space that can be simultaneously mapped by TLB

# VM-related events in pipeline



- Handling a TLB miss needs a hardware or software mechanism to refill TLB
  - usually done in hardware now
- Handling a page fault (e.g., page is on disk) needs a *precise* trap so software handler can easily resume after retrieving page
- Handling protection violation may abort process

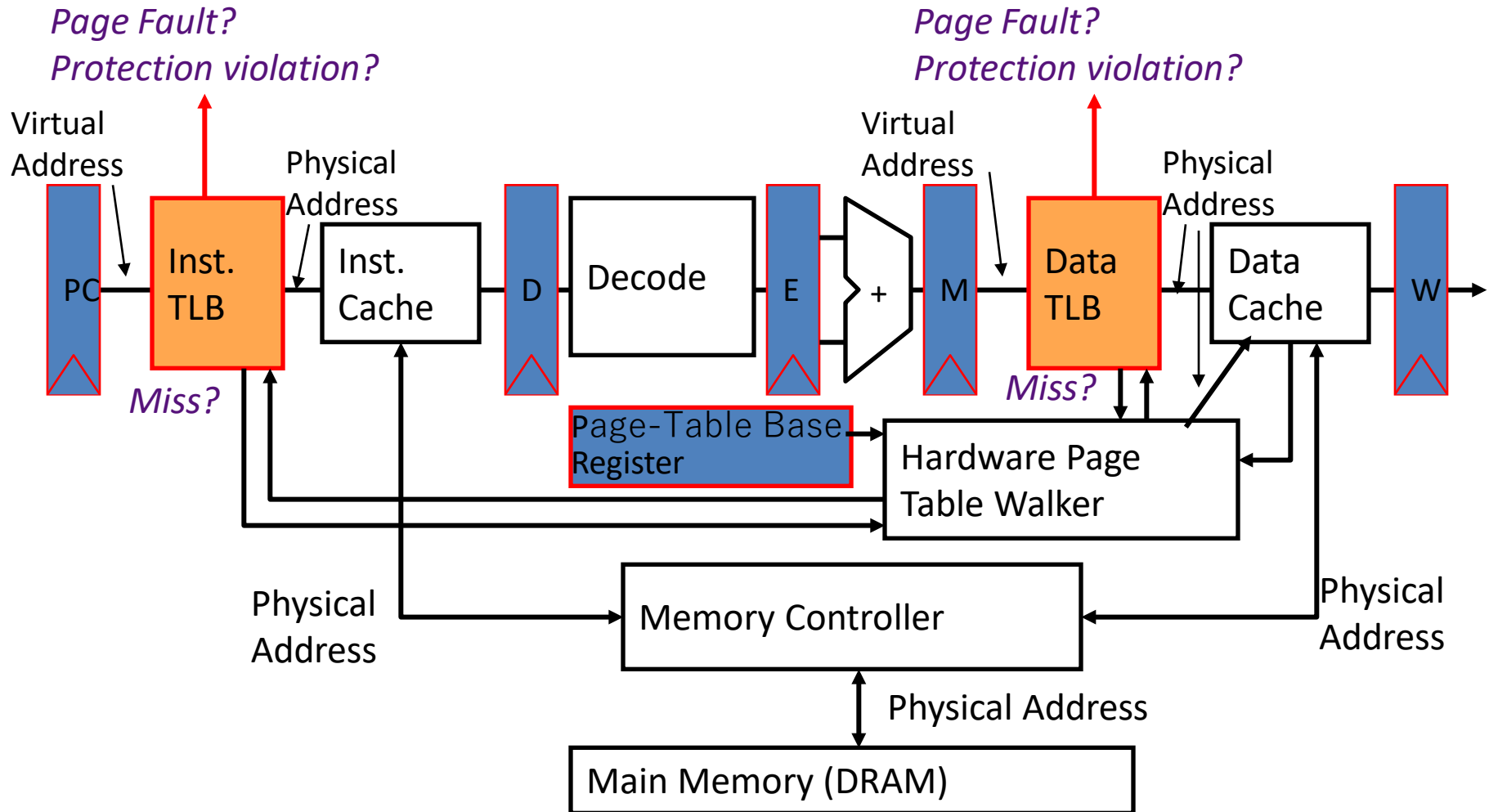
# Hierarchical Page Table Walk: SPARC v8



MMU does this table walk in hardware on a TLB miss

# Page-Based Virtual-Memory Machine

(Hardware Page-Table Walk)

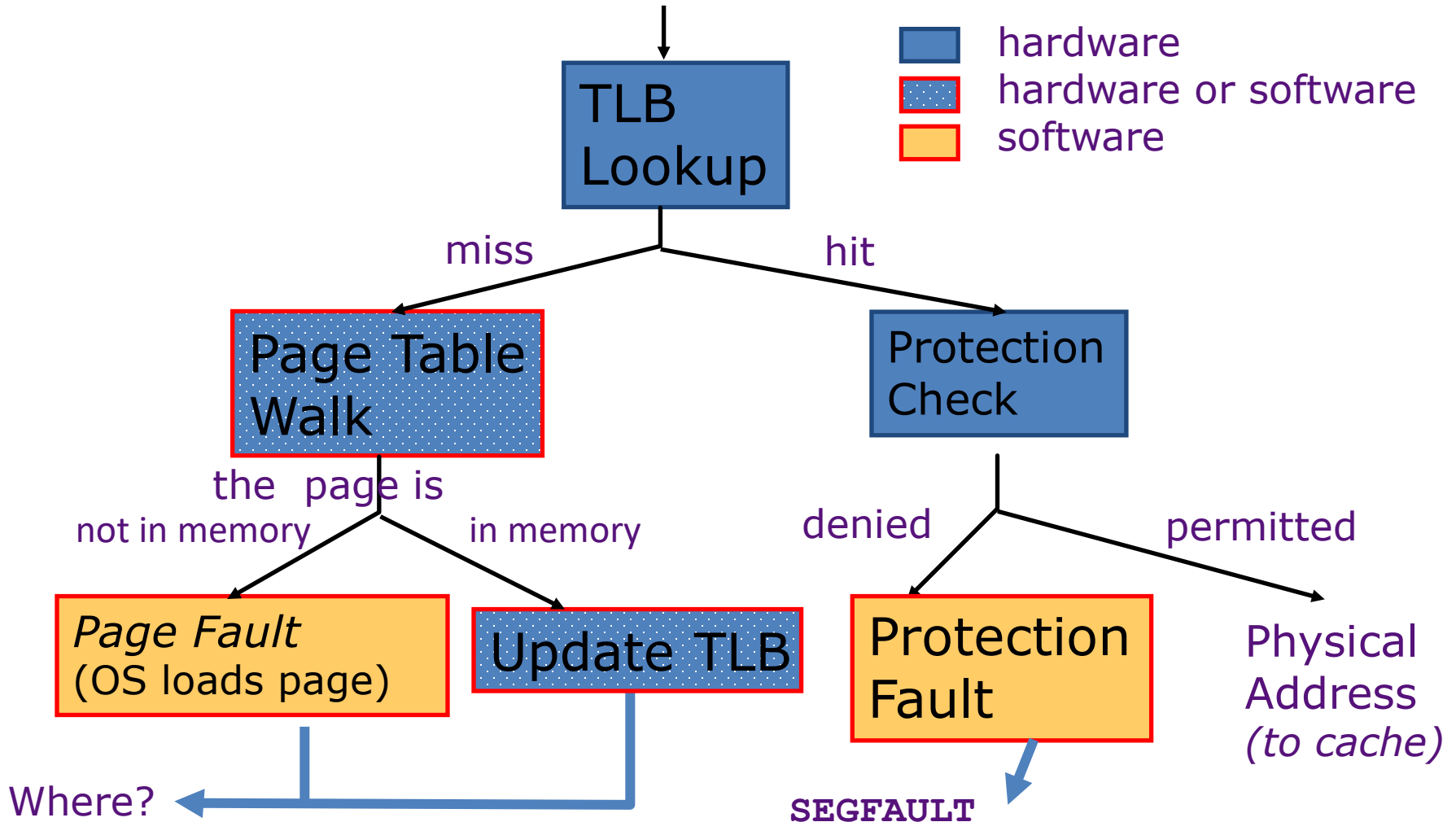


- Assumes page tables held in untranslated physical memory

# Address Translation:

*putting it all together*

Virtual Address

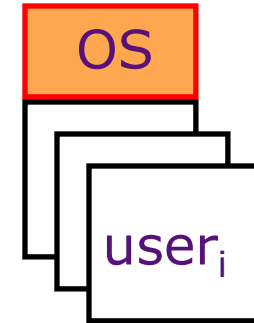


# Modern Virtual Memory Systems

*Illusion of a large, private, uniform store*

## Protection & Privacy

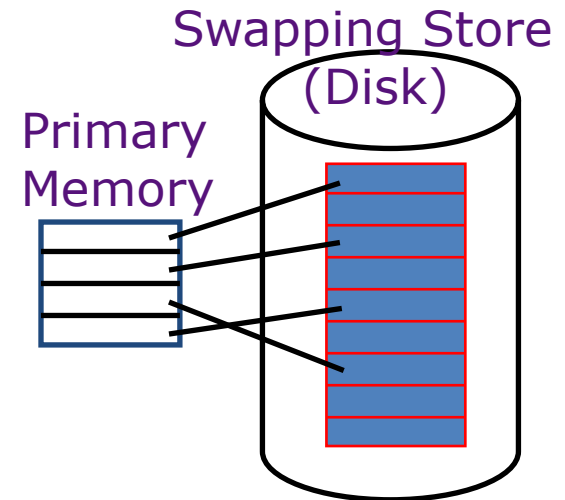
several users, each with their private address space and one or more shared address spaces  
page table = name space



## Demand Paging

Provides the ability to run programs larger than the primary memory

Hides differences in machine configurations



*The price is address translation on each memory reference*



# Unlimited?

```
wangc@64G:~$ ulimit -s
8192
wangc@64G:~$ ulimit -a
core file size          (blocks, -c) 0
data seg size          (kbytes, -d) unlimited
scheduling priority    (-e) 0
file size              (blocks, -f) unlimited
pending signals        (-i) 256820
max locked memory      (kbytes, -l) 16384
max memory size        (kbytes, -m) unlimited
open files             (-n) 1024
pipe size              (512 bytes, -p) 8
POSIX message queues   (bytes, -q) 819200
real-time priority     (-r) 0
stack size             (kbytes, -s) 8192
cpu time               (seconds, -t) unlimited
max user processes     (-u) 256820
virtual memory         (kbytes, -v) unlimited
file locks             (-x) unlimited
wangc@64G:~$
```

# Remember: Out of Memory

- Insufficient free memory: `malloc()` returns `NULL`

```
1 /*
2     This is a test for CS 110. All copyrights ...
3 */
4
5 #include <stdio.h>
6 #include <stdlib.h>
7
8 int main(int argc, char **argv) {
9     const int G = 1024 * 1024 * 1024;
10    for (int n = 0; ; n++) {
11        char *p = malloc(G * sizeof(char)); // 1GB every time
12        if (p == NULL) {
13            fprintf(stderr,
14                    "failed to allocate > %g TeraBytes\n",
15                    n / 1024.0);
16            return 1;
17        }
18        // no free, keep allocating until out of memory
19    }
20    return 0;
21 }
```

```
wangc@64G:~/TT$ gcc test.c -o t -Wall -O3
wangc@64G:~/TT$ ./t
failed to allocate > 127.99 TeraBytes
wangc@64G:~/TT$
```



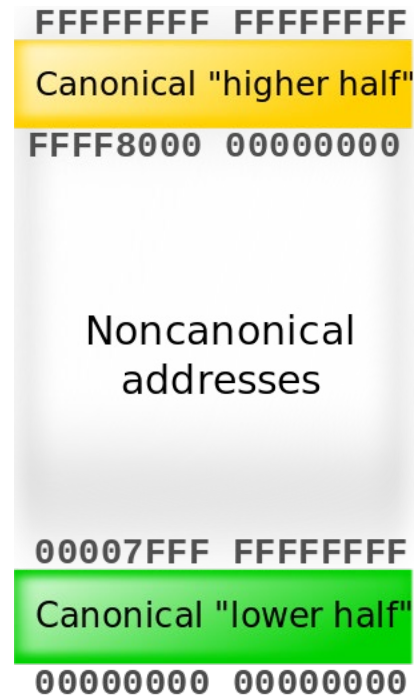
# Limited VM Space with x86-64

- 64-bit Linux allows up to **128TB** of virtual address space for individual processes, and can address approximately 64 TB of physical memory, subject to processor and system limitations.
- For Windows 64-bit versions, both 32- and 64-bit applications, if not linked with “*large address aware*”, are limited to **2GB** of virtual address space; otherwise, **128TB** for Windows 8.1 and Windows Server 2012 R2 or later.

Source: <https://en.wikipedia.org/wiki/X86-64>

# 48bit for address translation only

- Still provides plenty of space!
- Higher bits “sign extended”:  
“canonical form”
- Convention: “Higher half” for  
the Operating System
- Intel has plans (“whitepaper”) for  
56 bit translation – no hardware yet



- [https://en.wikipedia.org/wiki/X86-64#Virtual\\_address\\_space\\_details](https://en.wikipedia.org/wiki/X86-64#Virtual_address_space_details)

# Using 128TB of Memory!?

- A lazy allocation of virtual memory
  - Not used → not allocated
  - Try reading and writing from those pointers: works!
  - Even writing Gigabaytes of memory: works!
- Memory Compression!
  - Take no-recently used pages, compress them => free the physical page
- <https://www.lifewire.com/understanding-compressed-memory-os-x-2260327>

Process Name	Memory	Threads	Ports	PID	User	Compressed M...	Real Mem
a.out	60.51 GB	1	10	22329	schwerti	54.30 GB	6.22 GB

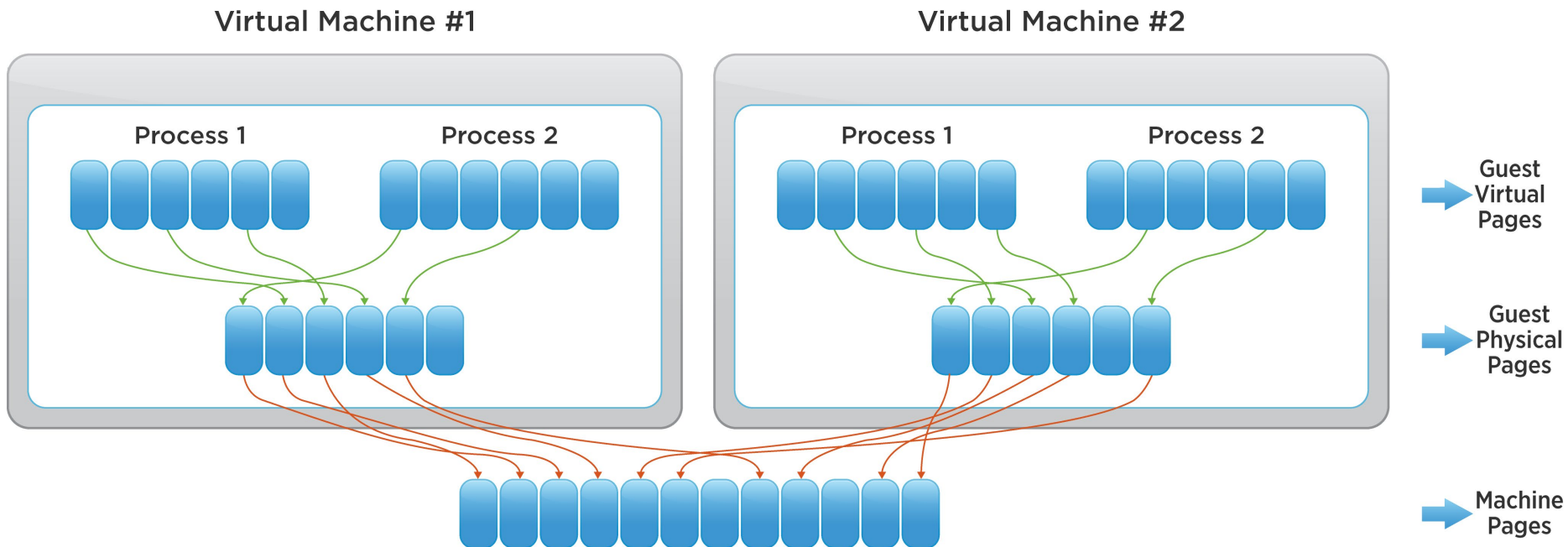
# Virtual Machines

# Virtual Machine

- Virtual Memory (VM) != Virtual Machine (VM)
  - Emulation: Run a complete virtual CPU & Memory & ... - a complete virtual machine in software (e.g. QEMU)
  - Virtual Machine: Run as many instructions as possible directly on CPU, only simulate some parts of the machine) (e.g. VirtualBox)
- Last lecture: Supervisor Mode & User Mode;  
now also: Virtual Machine Mode
  - Host OS activates virtual execution mode for guest OS =>
  - Guest OS thinks it runs in supervisor mode, but in fact it doesn't have access to physical memory! (among other limitations)
- CPUs support it (AMD-V, Intel VT-x), e.g. new Intel instructions: VMPTRLD, VMPTRST, VMCLEAR, VMREAD, VMWRITE, VMCALL, VMLAUNCH, VMRESUME, VMXOFF, and VMXON

# What about the memory in Virtual Machines?

- Need to translate Guest Virtual Address to Guest Physical address to Machine (Host) Physical address: Earlier the Guest part was done (transparently) in software by the Virtual Machine ... now in hardware!



# Kernel Samepage Merging

- Some pages are the same for two guests
  - Say, running two Windows 10 instances in a Ubuntu host
  - They share many pages
- KSM: a way to keep only one copy of pages at the host OS level
  - To save memory, up to 48.0% in some study
  - Not good for protection and privacy
    - Shared pages leading to the leak of information

# Cloud Servers

- Many tech giants provide cloud servers
  - AWS, Microsoft Azure, Aliyun, Tencent Cloud, etc.
  - With flexible and secure computing capabilities
  - Pay-and-rent
- How cloud servers work?
  - Virtualization + distributed storage
    - Resources, e.g., CPU, memory, disk, virtualized to serve multiple users
    - Virtual machine migration between physical servers



# Conclusion: VM features track historical uses

- **Bare machine, only physical addresses**
  - One program owned entire machine
- **Batch-style multiprogramming**
  - Several programs sharing CPU while waiting for I/O
  - Base & bound: translation and protection between programs (not virtual memory)
  - Problem with external fragmentation (holes in memory), needed occasional memory defragmentation as new jobs arrived
- **Time sharing**
  - More interactive programs, waiting for user. Also, more jobs/second.
  - Motivated move to fixed-size page translation and protection, no external fragmentation (but now internal fragmentation, wasted bytes in page)
  - Motivated adoption of virtual memory to allow more jobs to share limited physical memory resources while holding working set in memory
- **Virtual Machine Monitors**
  - Run multiple operating systems on one machine
  - Hardware support for two levels of translation/protection
    - Guest OS virtual -> Guest OS physical -> Host machine physical
  - Kernel samepage merging to reduce memory footprints
  - Cloud servers based on virtual machine instances

# Quiz on TLB

Piazza: “Video Lecture 22 VM”

- TLB Reach: Size of largest virtual address space that can be simultaneously mapped by TLB
- If we increase the page size, say, from 4KB to 16KB, can we increase the TLB reach?
  - A. Yes, of course
  - B. No, TLB reach is fixed given a processor and memory space