Constructive Computer Architecture

Virtual Memory: From Address Translation to Demand Paging

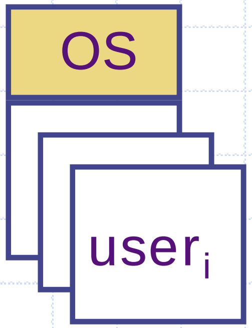
Arvind
Computer Science & Artificial Intelligence Lab.
Massachusetts Institute of Technology

Modern Virtual Memory Systems

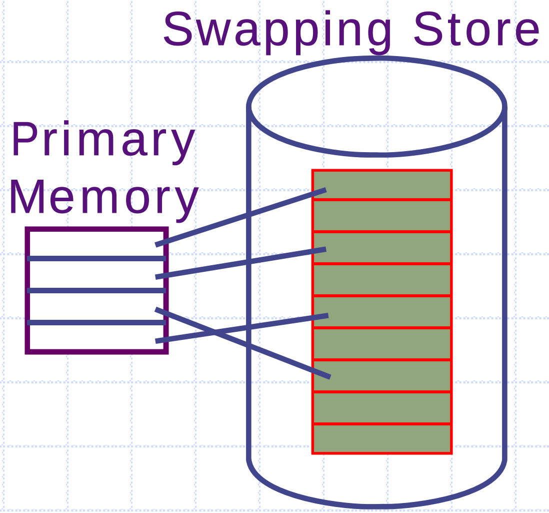
Illusion of a large, private, uniform store

- **Protection & Privacy**
  - Each user has one private and one or more shared address spaces

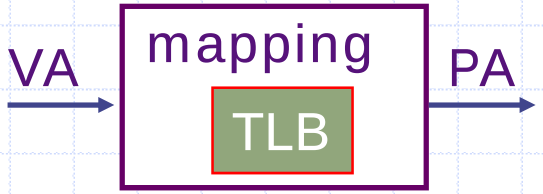
    page table ≡ name space



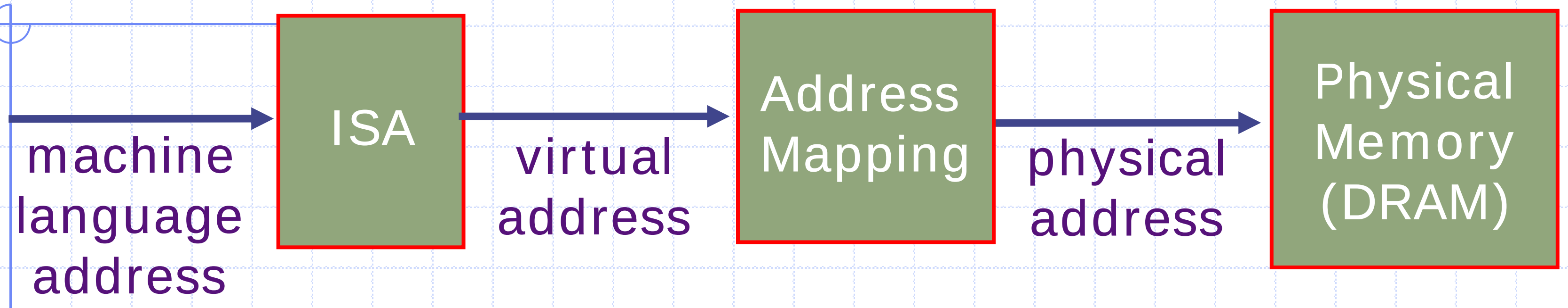
- **Demand Paging**
  - Provides the ability to run programs larger than the primary memory
  - Hides differences in machine configurations



The price of VM is address translation on each memory reference



Names for Memory Locations



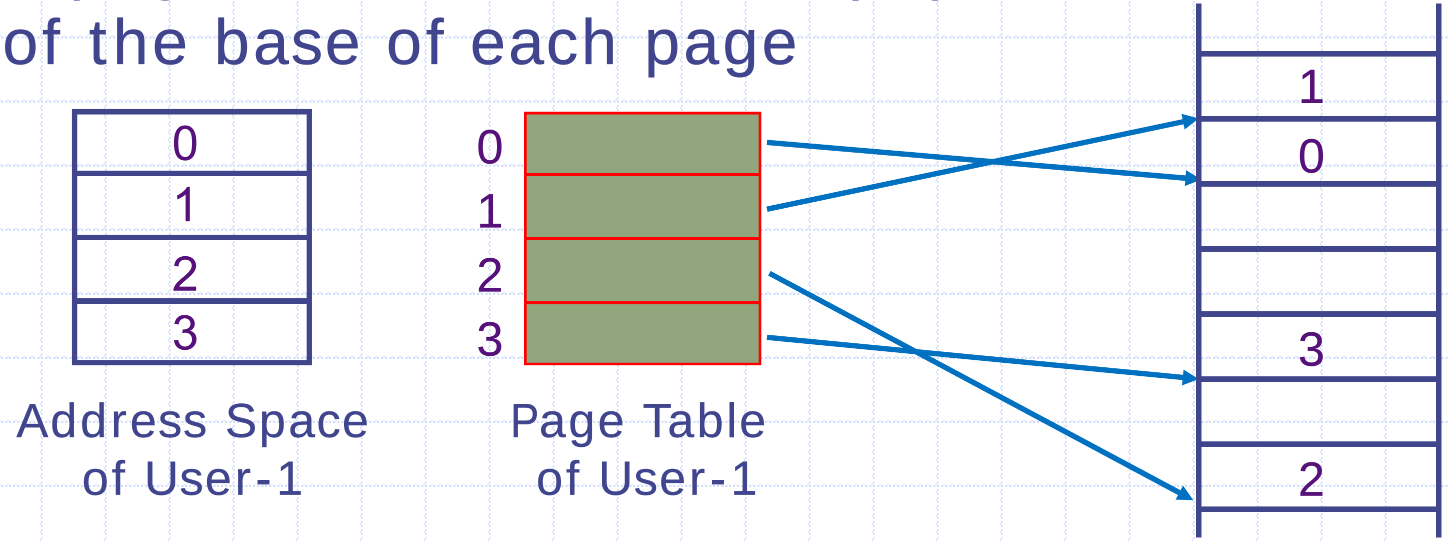
- Machine language address
  - as specified in machine code
- Virtual address
  - ISA specifies translation of machine code address into virtual address of program variable (sometime called *effective* address)
- Physical address
  - operating system specifies mapping of virtual address into name for a physical memory location

Paged Memory Systems

- Processor generated address can be interpreted as a pair <page number, offset>

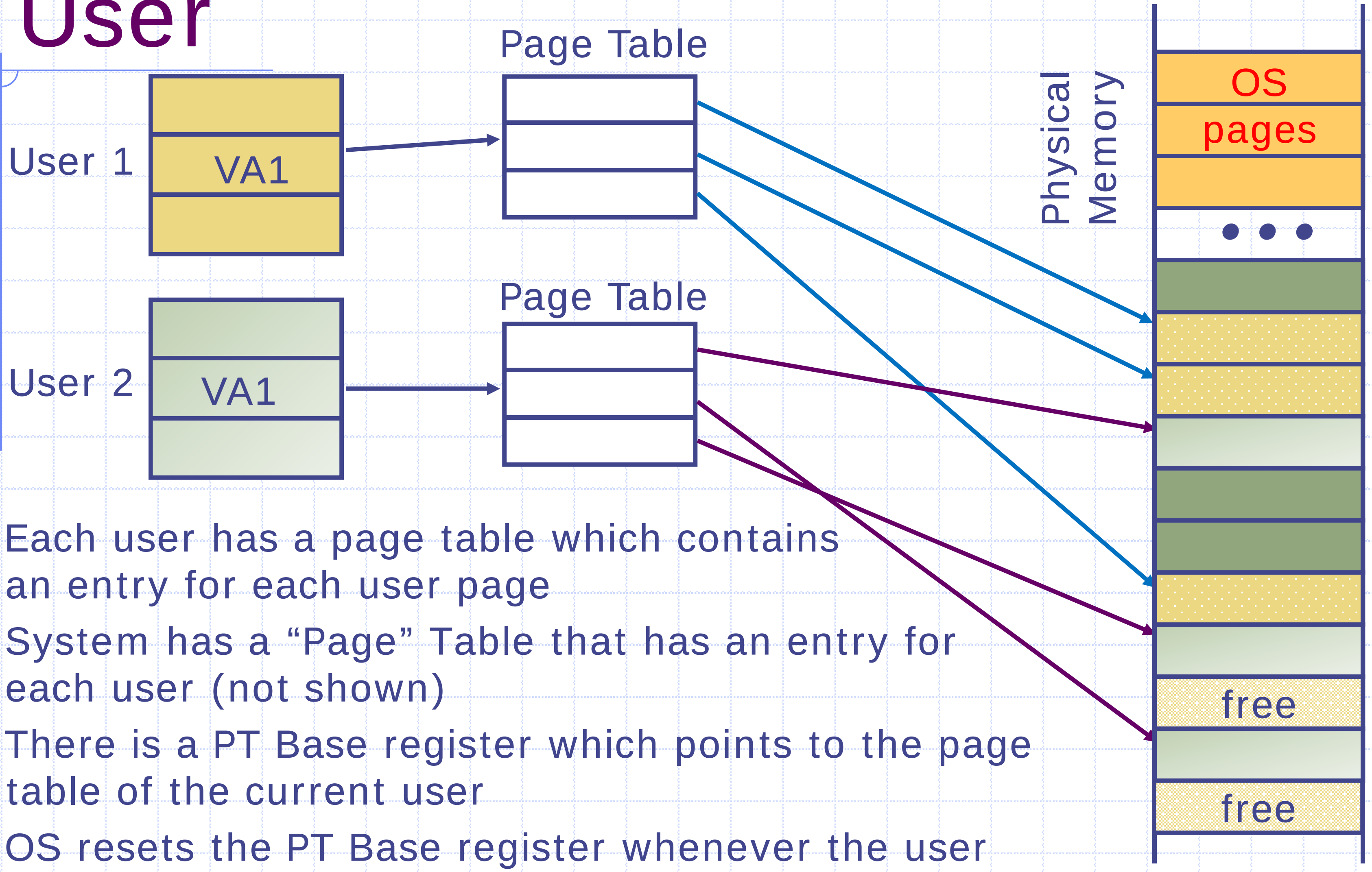
page number	offset

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



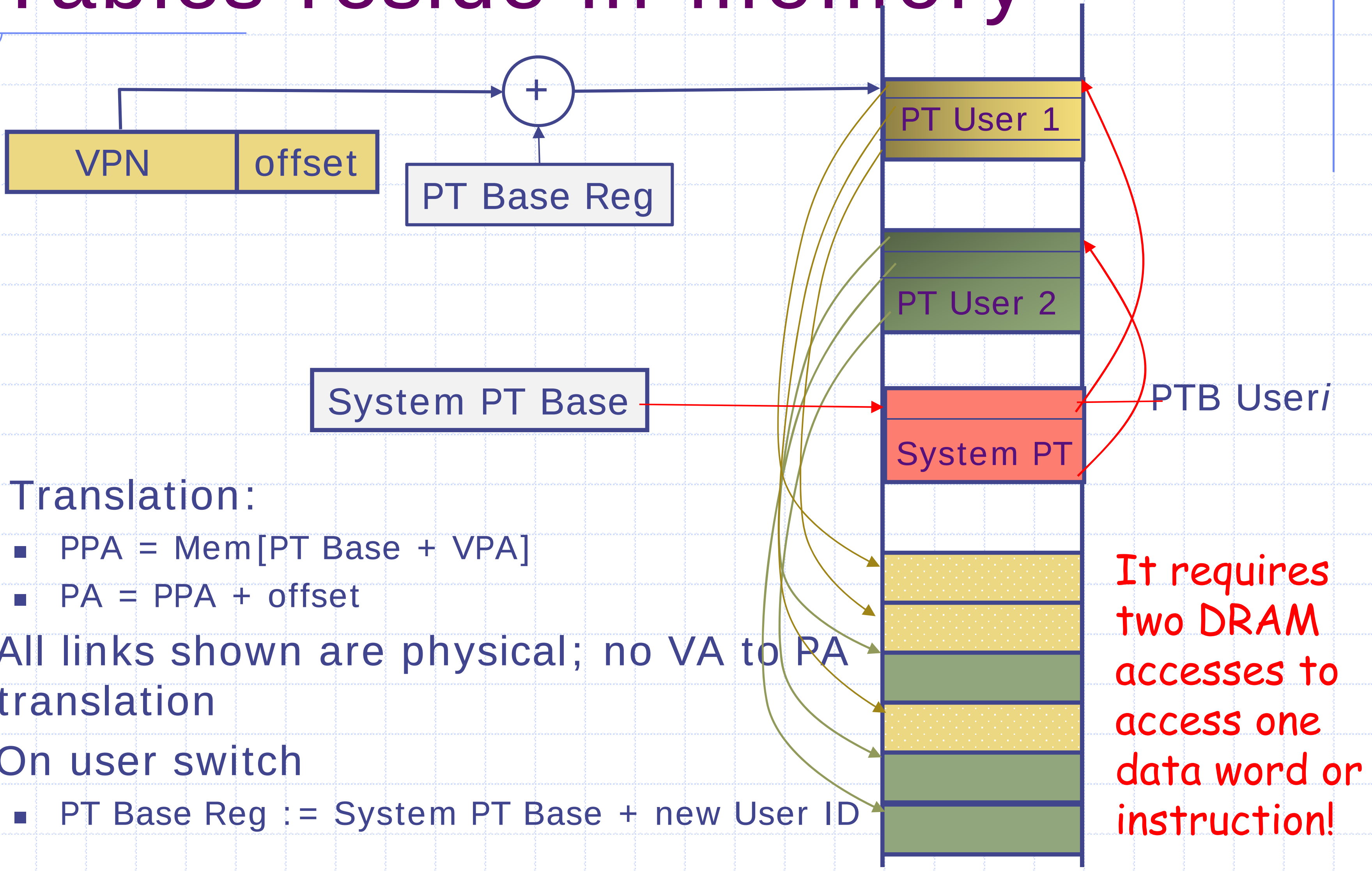
Page tables make it possible to store the pages of a program non-contiguously

Private Address Space per User



- Each user has a page table which contains an entry for each user page
- System has a "Page" Table that has an entry for each user (not shown)
- There is a PT Base register which points to the page table of the current user
- OS resets the PT Base register whenever the user changes; requires consulting the System Page Table

Suppose all Pages and Page Tables reside in memory



- Translation:
  - PPA = Mem[PT Base + VPA]
  - PA = PPA + offset
- All links shown are physical; no VA to PA translation
- On user switch
  - PT Base Reg := System PT Base + new User ID

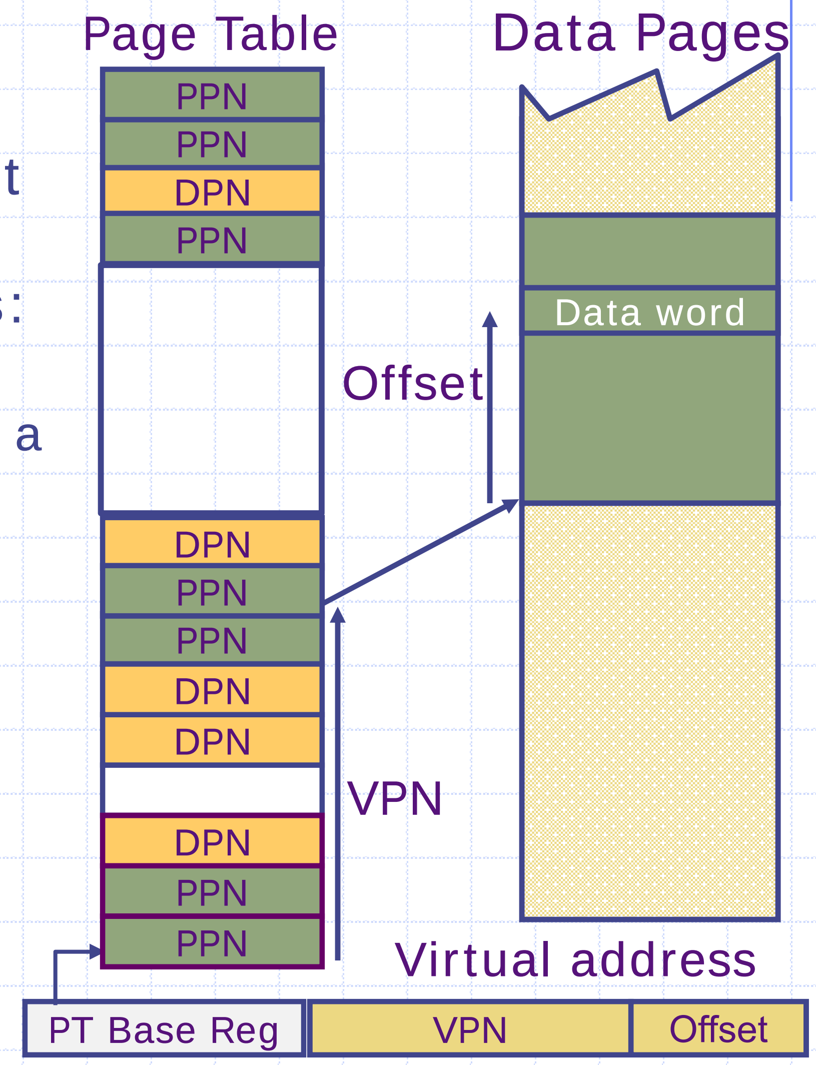
VM Implementation Issues

- How to reduce memory access overhead
- What if all the pages can't fit in DRAM
  - What if the user page table can't fit in DRAM
  - What if the System page table can't fit in DRAM

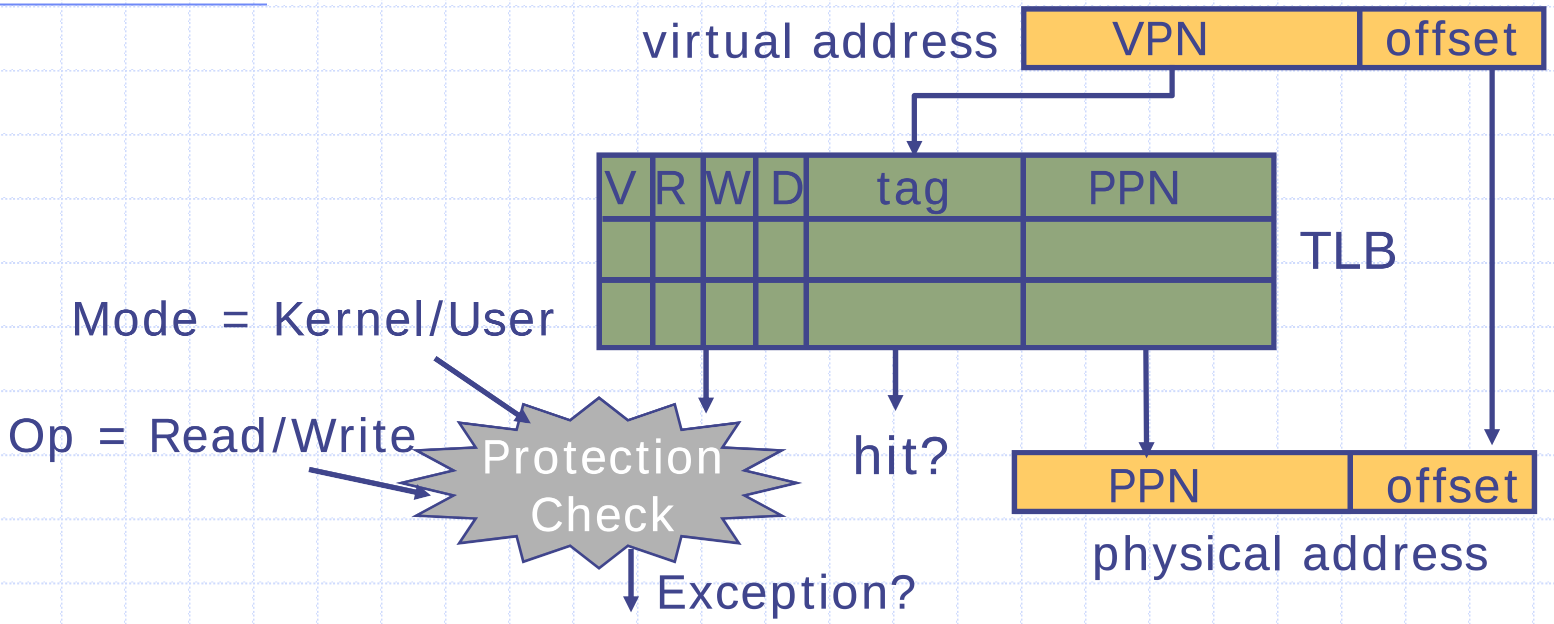
A good VM design needs to be fast and space efficient

Page Table Entries and Swap Space

- DRAM has to be backed up by *swap space* on disk because all the pages of all the users cannot fit in DRAM
- Page Table Entry (PTE) contains:
  - A bit to indicate if a page exists
  - PPN (physical page number) for a memory-resident page
  - DPN (disk page number) for a page on the disk
  - Protection and usage bits



Translation Lookaside Buffers (TLB)



- Keep some of the (VPA,PPA) translations in a cache (TLB)
  - No need to put (VPA, DPA) in TLB
- Every instruction fetch and data access needs address translation and protection checks
- TLB miss causes one or more accesses to the page table

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
- TLB Reach: Size of largest virtual address space that can be simultaneously mapped by TLB

  Example: 64 TLB entries, 4KB pages, one page per entry

  TLB Reach = 64 entries * 4 KB = 256 KB
- Replacement policy? Random, FIFO, LRU, ...
- Switching users is expensive because TLB has to be flushed

  Store User IDs in TLB entries

Handling a TLB Miss

- Software (MIPS, Alpha)
  - TLB miss causes an exception and the operating system walks the page tables and reloads TLB
  - A privileged “untranslated” addressing mode is used for PT walk
- Hardware (SPARC v8, x86, PowerPC)
  - A memory management unit (MMU) walks the page tables and reloads the TLB
  - If a missing (data or PT) page is encountered during the TLB reloading, MMU gives up and signals a Page-Fault exception for the original instruction

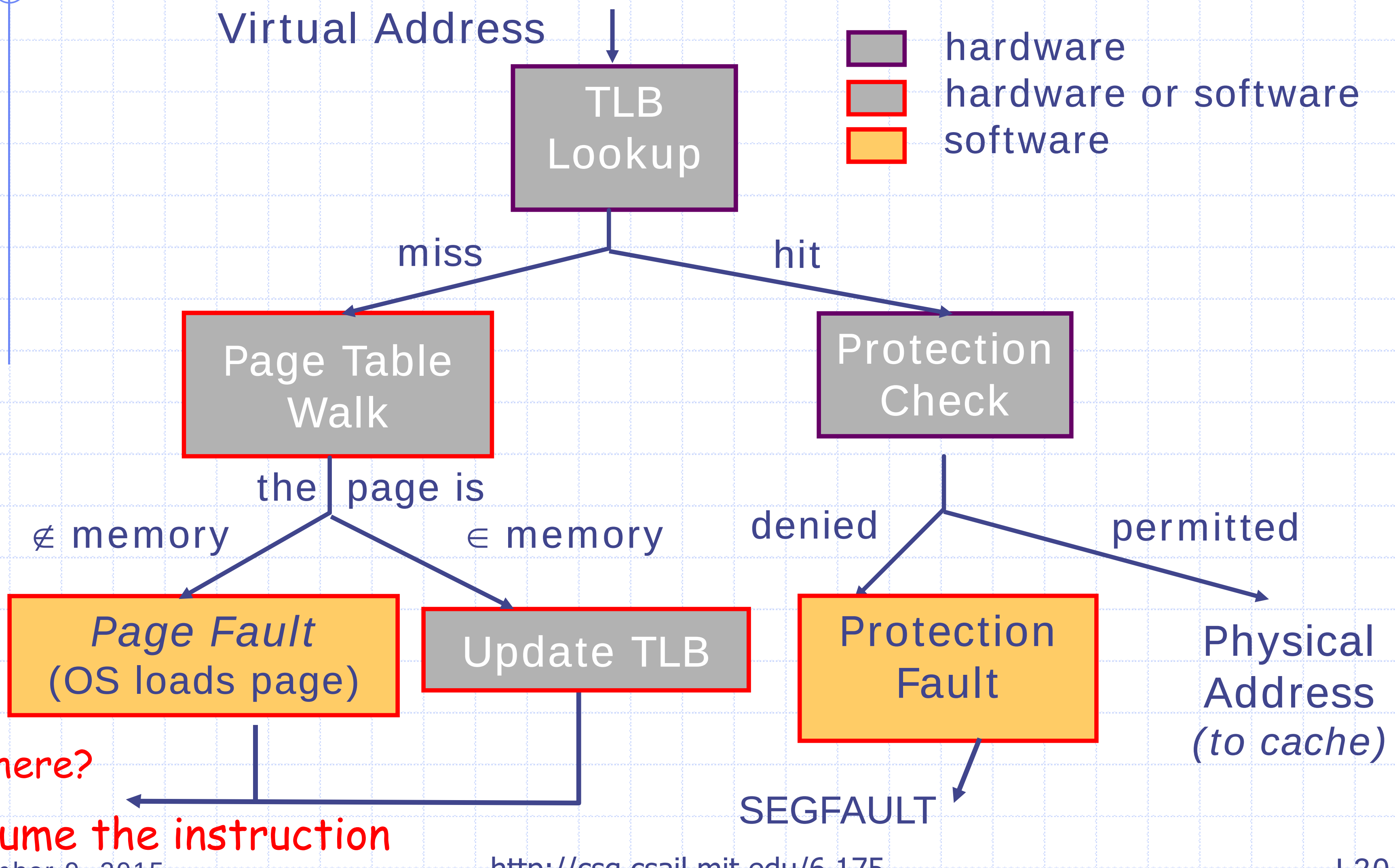
Handling a Page Fault

- When the referenced page is not in DRAM:
  - The missing page is located (or created)
  - It is brought in from disk, and page table is updated

    Another job may be run on the CPU while the first job waits for the requested page to be read from disk
  - If no free pages are left, a page is swapped out

    approximate LRU replacement policy
- Since it takes a long time (msecs) to transfer a page, page faults are handled completely in software (OS)

Address Translation:
putting it all together

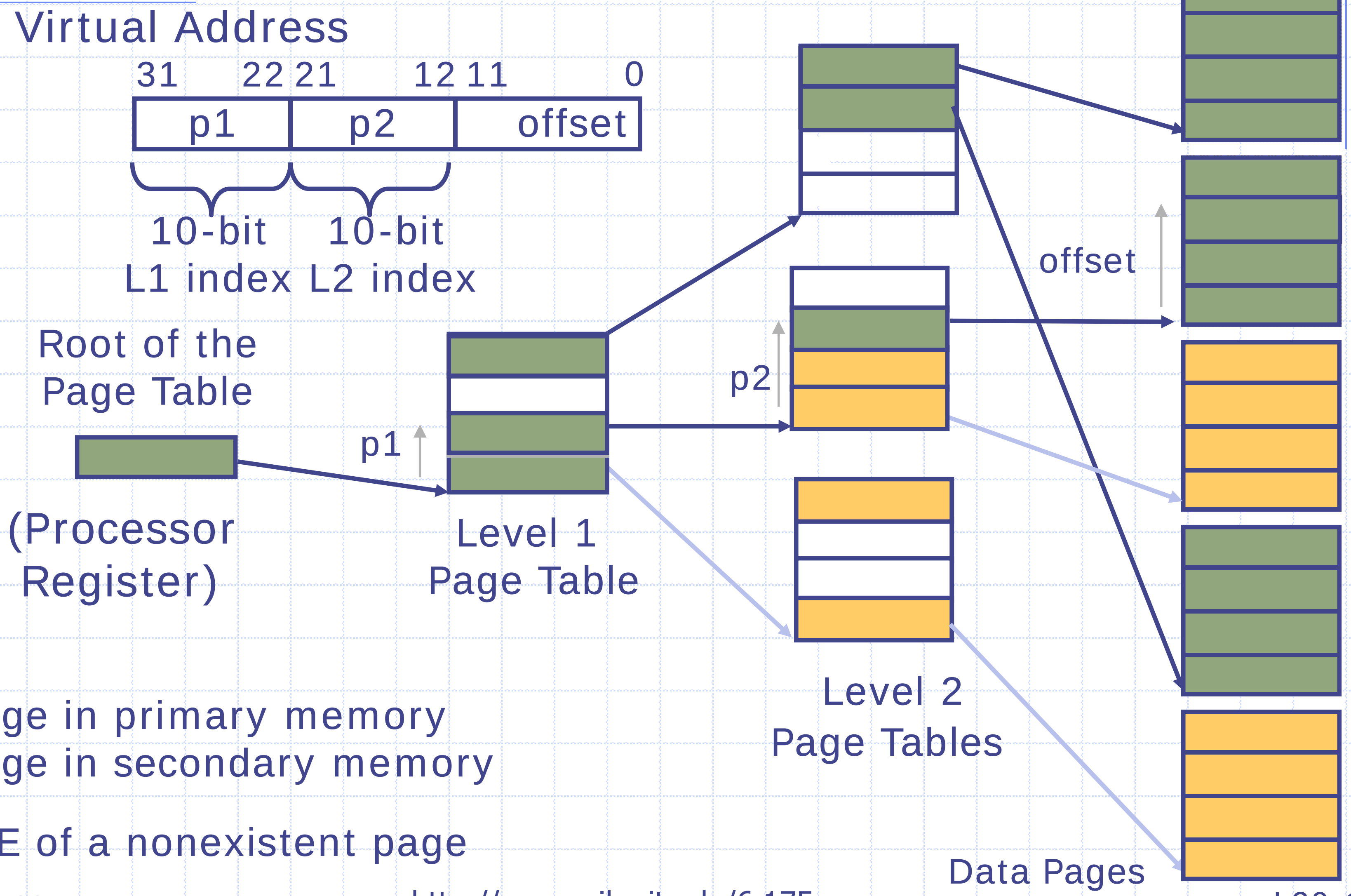


Size of Linear Page Table

- With 32-bit addresses, 4-KB pages & 4-byte PTEs
  - 2^{20} PTEs, i.e, 4 MB page table per user
  - 4 GB of swap space needed to back up the full virtual address space
- Larger Pages can reduce the overhead *but cause*
  - Internal fragmentation (part of memory in a page is not 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!)

However, Page Tables are sparsely populated and hence hierarchical organization can help

Hierarchical Page Table



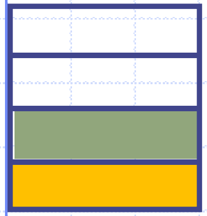
🟩 page in primary memory

🟨 page in secondary memory

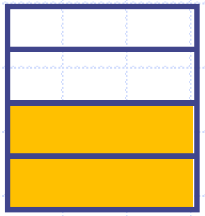
□ PTE of a nonexistent page

Swapping a Page of a Page Table

Hierarchical page tables permit parts of the page table to kept in the swap space



A PTE in DRAM contains DRAM or DISK addresses



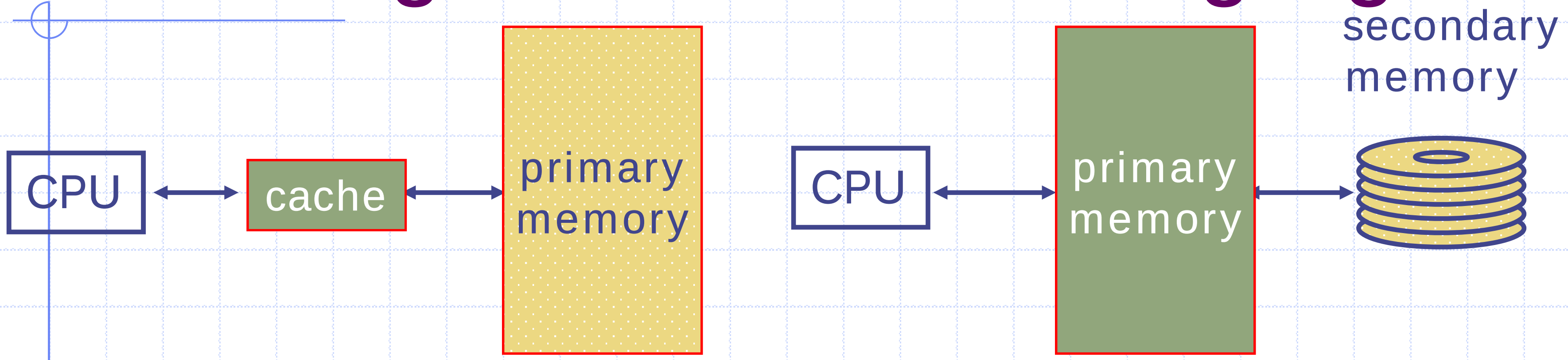
A PTE in DISK contains *only* DISK addresses

⇒ a page of a PT can be swapped out only if none its PTE's point to pages in the primary memory

Why?

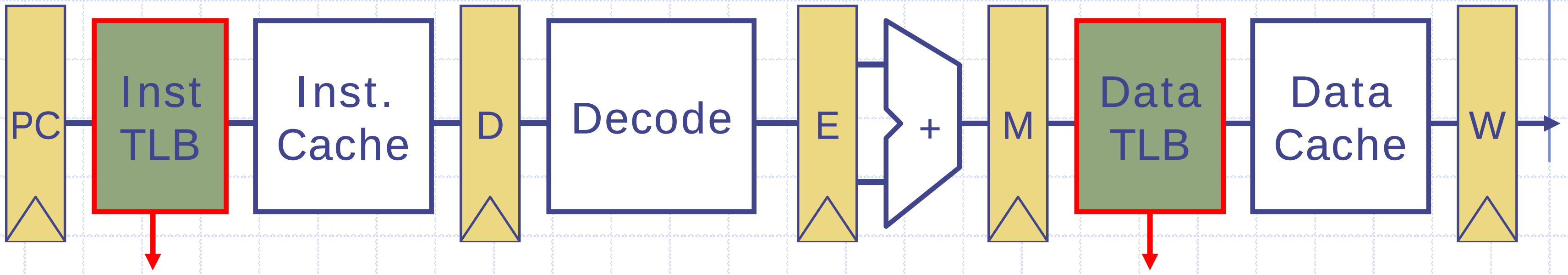
Don't want to cause a page fault during translation when the data is in memory

Caching vs. Demand Paging



Caching	*Demand paging*
cache slot	page frame
cache line (~32 bytes)	page (~4K bytes)
cache miss rate (1% to 20%)	page miss rate (<0.001%)
cache hit (~1 cycle)	page hit (~100 cycles)
cache miss (~100 cycles)	page miss (~5M cycles)
miss is handled in *hardware*	miss is handled mostly in *software*

Address Translation in CPU Pipeline



TLB miss? Page Fault?
Protection violation?

TLB miss? Page Fault?
Protection violation?

- Software handlers need a *restartable* exception on page fault or protection violation
- Handling a TLB miss needs a *hardware* or *software* mechanism to refill TLB
- Need mechanisms to cope with the additional latency of a TLB:
  - slow down the clock
  - √ pipeline the TLB and cache access
  - virtual address caches
  - parallel TLB/cache access

The rest of the slides are optional