

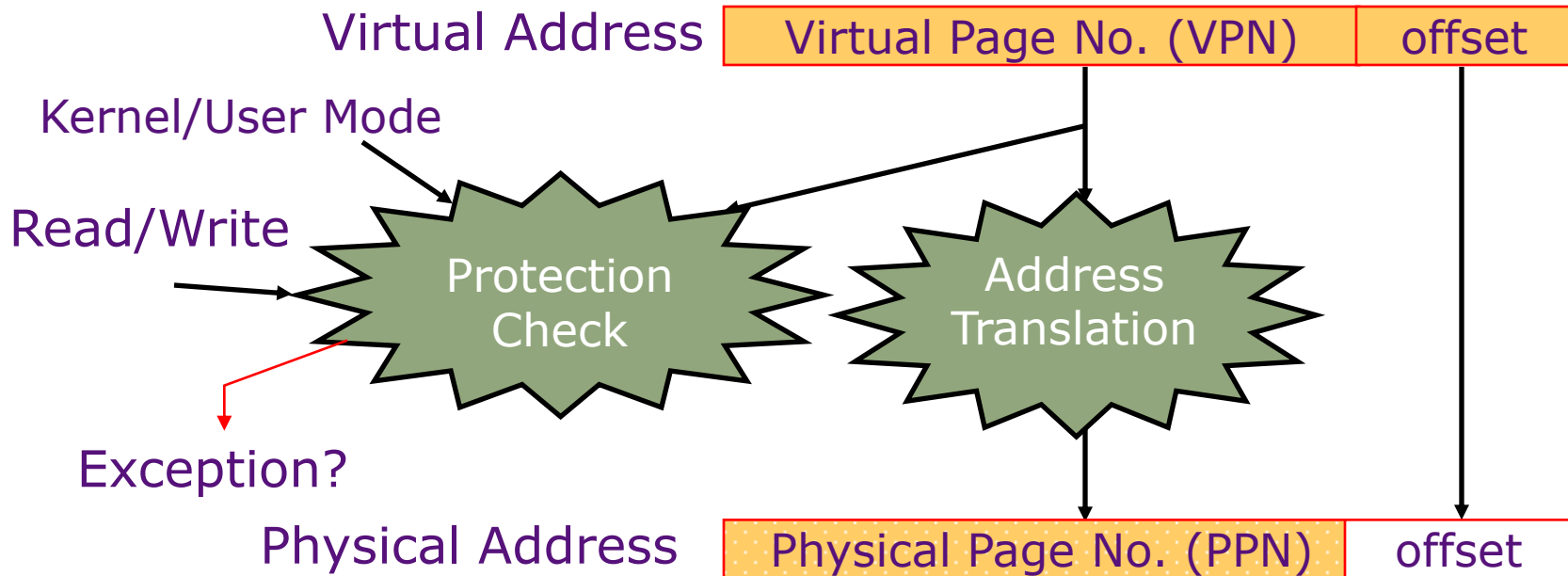
# Modern Virtual Memory Systems

*Daniel Sanchez*

Computer Science and Artificial Intelligence Laboratory  
M.I.T.

<http://www.csg.csail.mit.edu/6.823>

# 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*

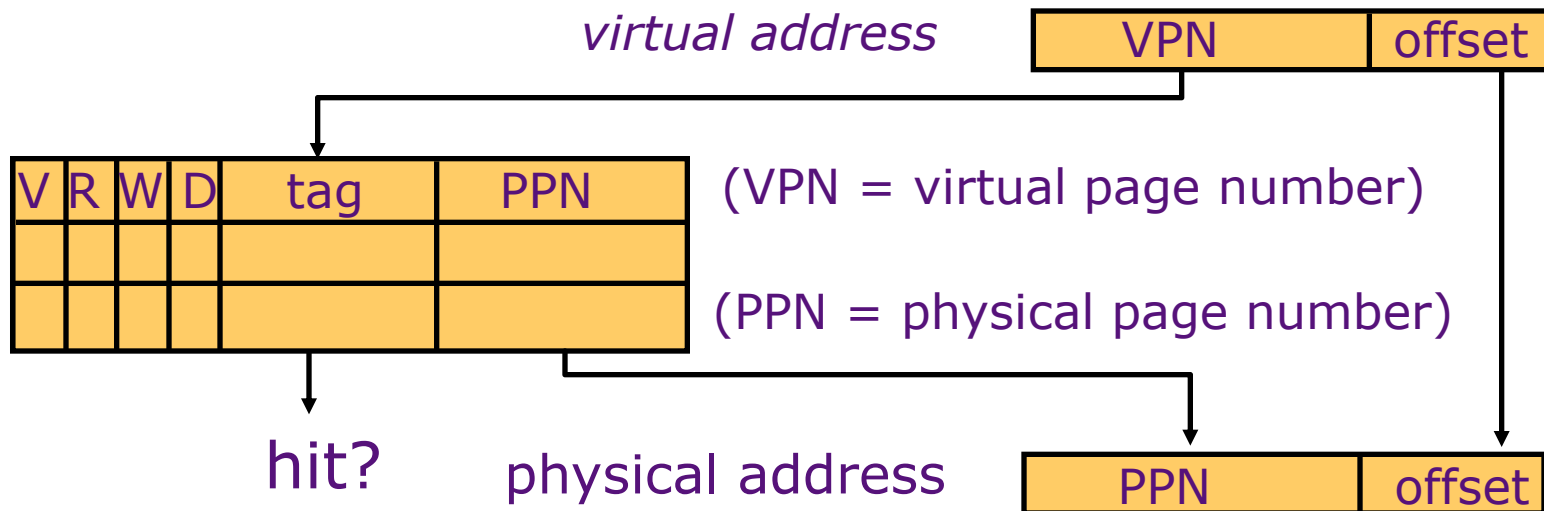
# Translation Lookaside Buffers

Address translation is very expensive!

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

Solution: *Cache translations in TLB*

TLB hit           ⇒ *Single-cycle Translation*  
 TLB miss         ⇒ *Page Table Walk to refill*



# TLB Designs

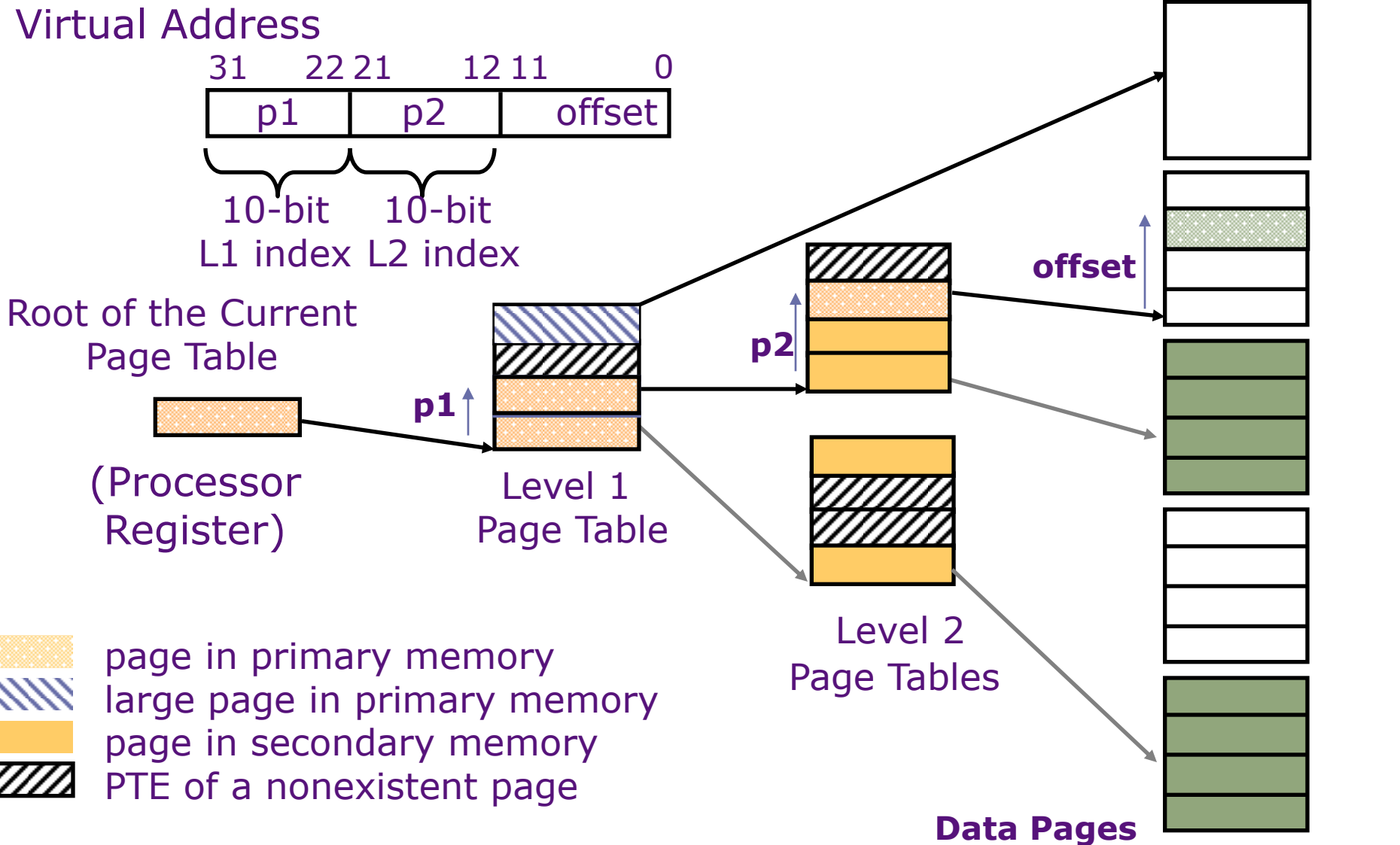
---

- 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
- Random or FIFO replacement policy
- No process information in TLB?
- 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

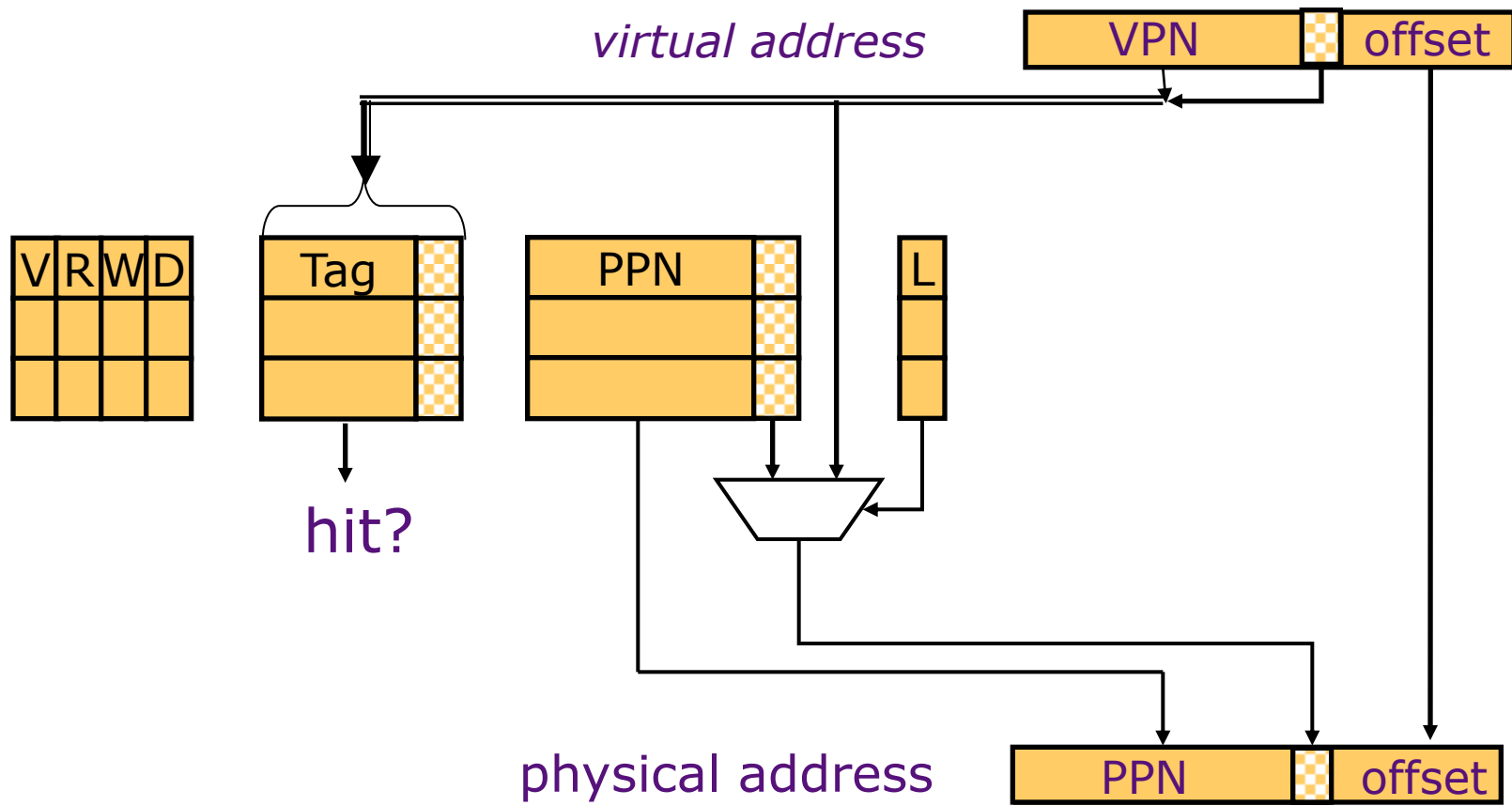
TLB Reach = 64 entries \* 4 KB = 256 KB (if contiguous) ?

# Variable-Sized Page Support



# Variable-Size Page TLB

Some systems support multiple page sizes.



# Handling a TLB Miss

---

## Software (MIPS, Alpha)

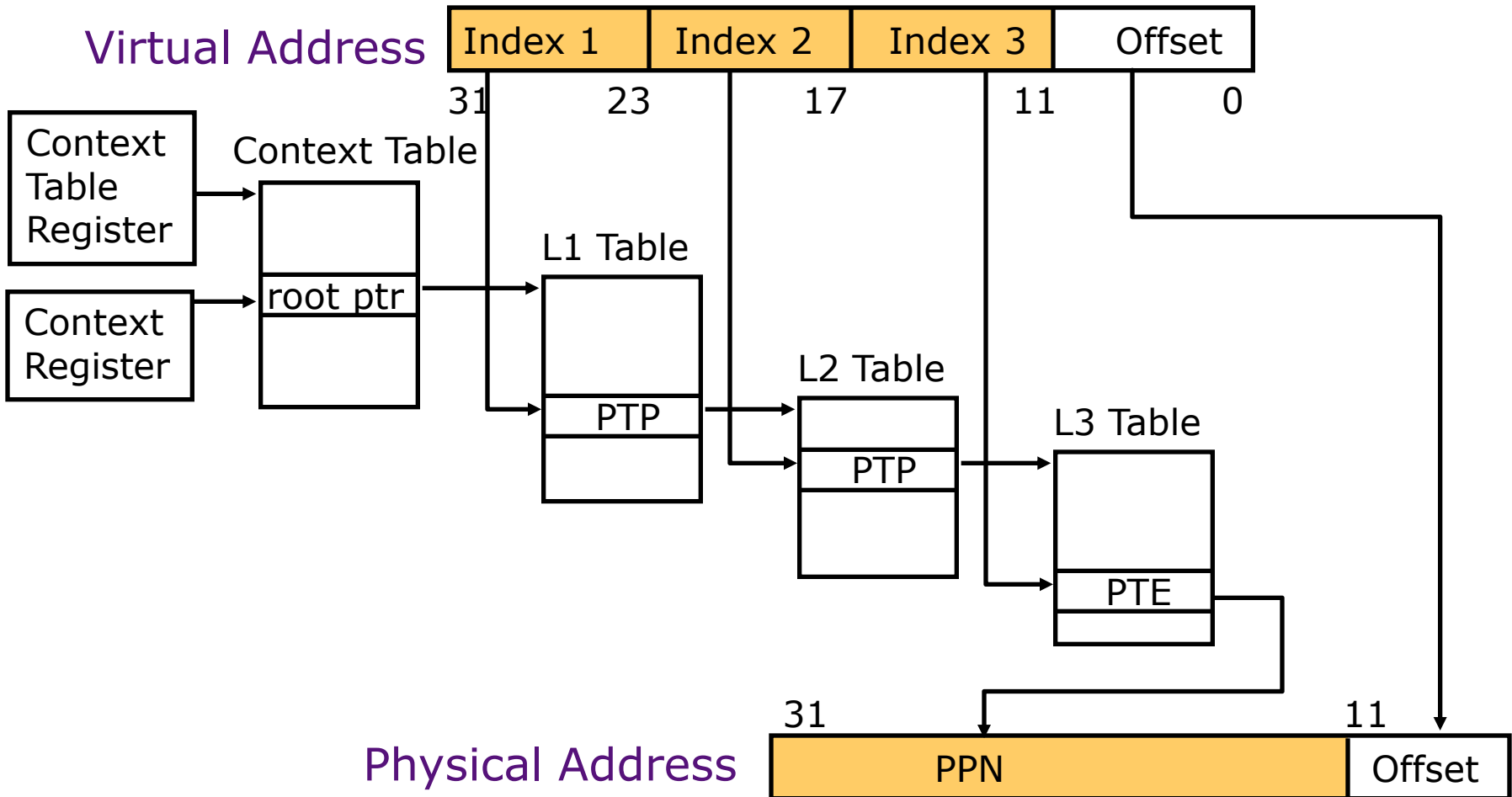
TLB miss causes an exception and the operating system walks the page tables and reloads TLB. *A privileged "untranslated" addressing mode used for walk*

## 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

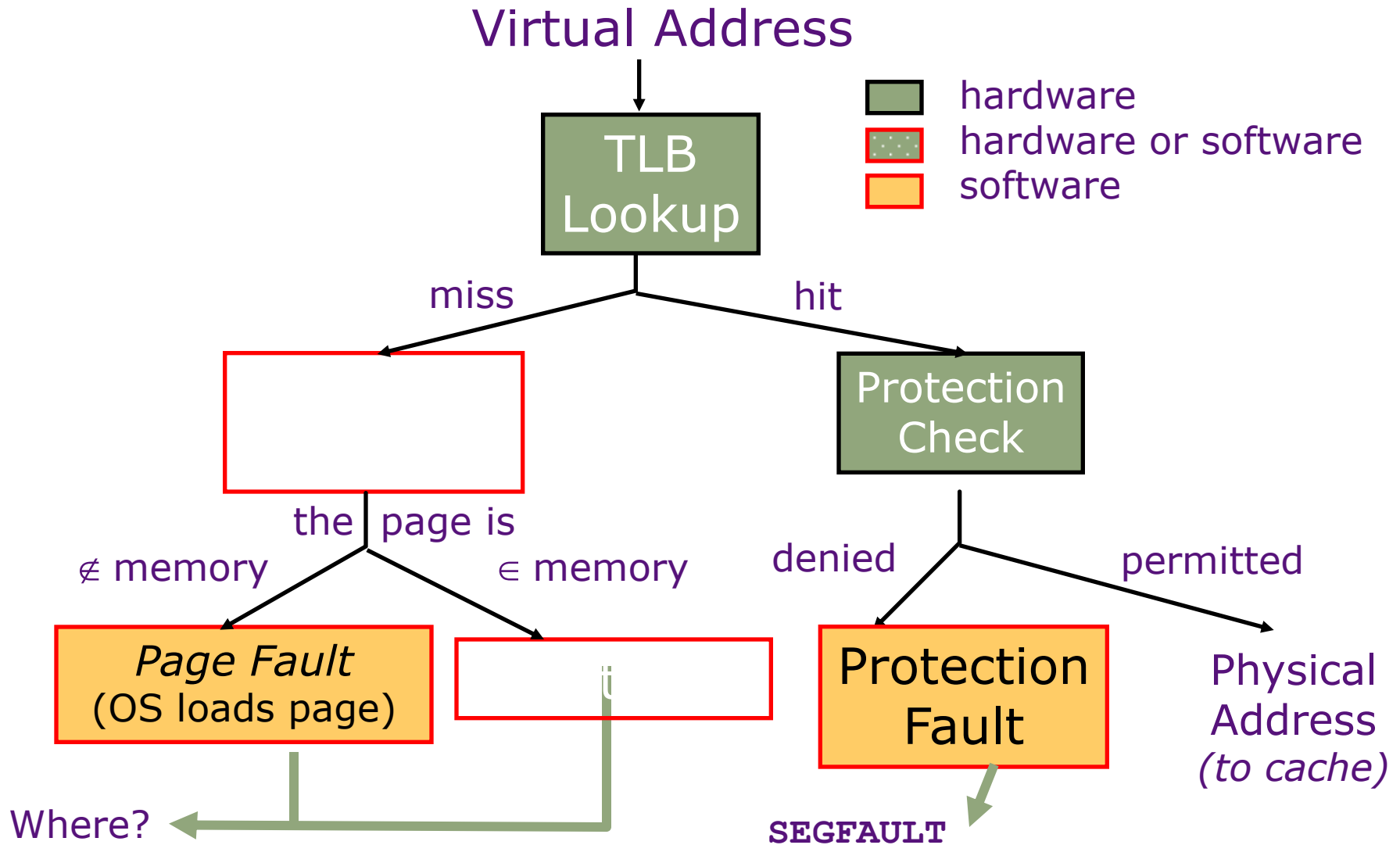
# Hierarchical Page Table Walk: SPARC v8



MMU does this table walk in hardware on a TLB miss



# Address Translation: *putting it all together*



# Topics

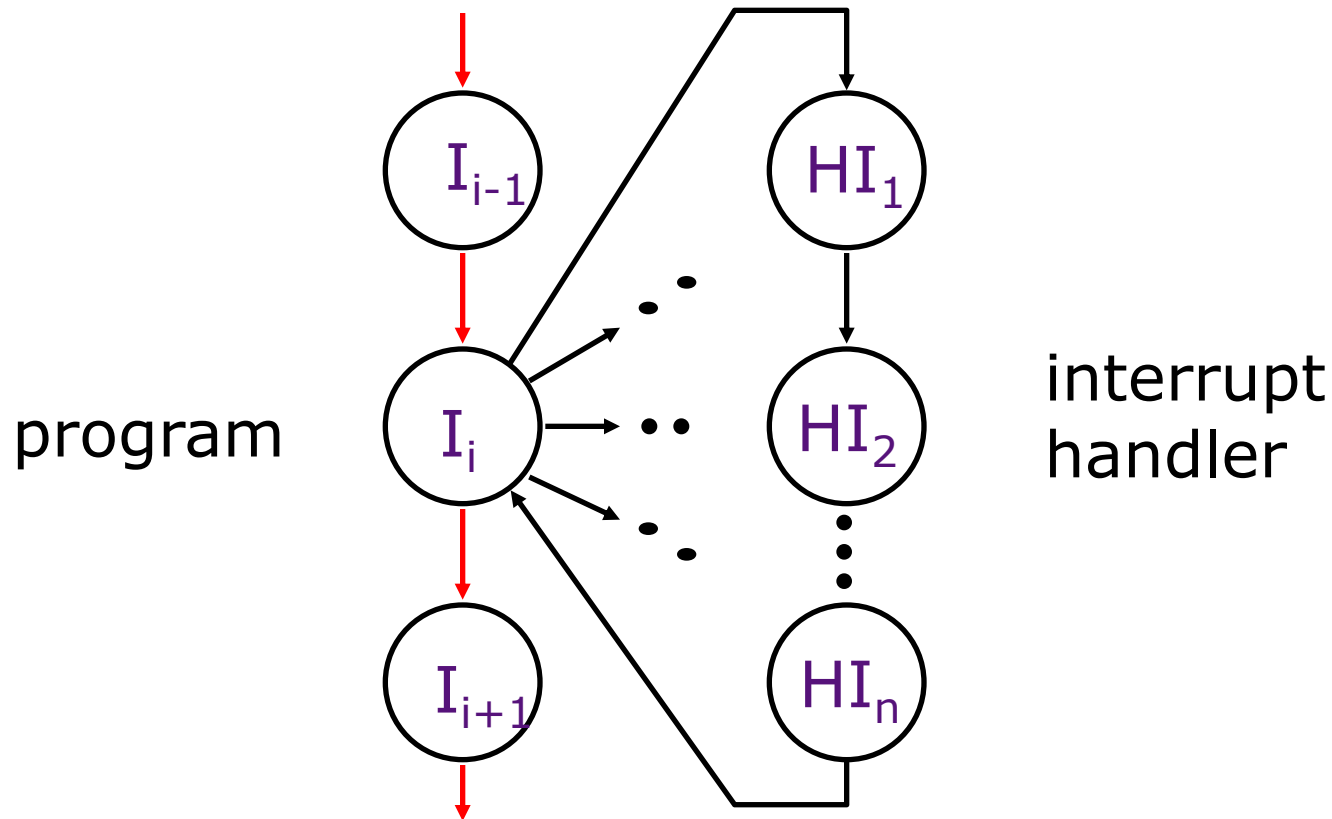
---

- Interrupts
- Speeding up the common case:
  - TLB & Cache organization
- Speeding up page table walks
- Modern Usage

# Interrupts:

altering the normal flow of control

---



An *external or internal event* that needs to be processed by another (system) program. The event is usually unexpected or rare from program's point of view.

# Causes of Interrupts

---

Interrupt: an *event* that requests the attention of the processor

- **Asynchronous: an *external event***
  - input/output device service-request
  - timer expiration
  - power disruptions, hardware failure
- **Synchronous: an *internal event (a.k.a exception)***
  - undefined opcode, privileged instruction
  - arithmetic overflow, FPU exception
  - misaligned memory access
  - *virtual memory exceptions*: page faults, TLB misses, protection violations
  - *traps*: system calls, e.g., jumps into kernel

# Asynchronous Interrupts:

## invoking the interrupt handler

---

- An I/O device requests attention by asserting one of the *prioritized interrupt request lines*
- When the processor decides to process the interrupt
  - It stops the current program at instruction  $I_i$ , completing all the instructions up to  $I_{i-1}$  (*precise interrupt*)
  - It saves the PC of instruction  $I_i$  in a special register (EPC)
  - It disables interrupts and transfers control to a designated interrupt handler running in the kernel mode

# Interrupt Handler

---

- Saves EPC before enabling interrupts to allow nested interrupts  $\Rightarrow$ 
  - need an instruction to move EPC into GPRs
  - need a way to mask further interrupts at least until EPC can be saved
- Needs to read a *status register* that indicates the cause of the interrupt
- Uses a special indirect jump instruction RFE (*return-from-exception*) that
  - enables interrupts
  - restores the processor to the user mode
  - restores hardware status and control state

# Synchronous Interrupts

---

- A synchronous interrupt (exception) is caused by a *particular instruction*
- In general, the instruction cannot be completed and needs to be *restarted* after the exception has been handled
  - With pipelining, requires undoing the effect of one or more partially executed instructions
- In case of a trap (system call), the instruction is considered to have been completed
  - A special jump instruction involving a change to privileged kernel mode

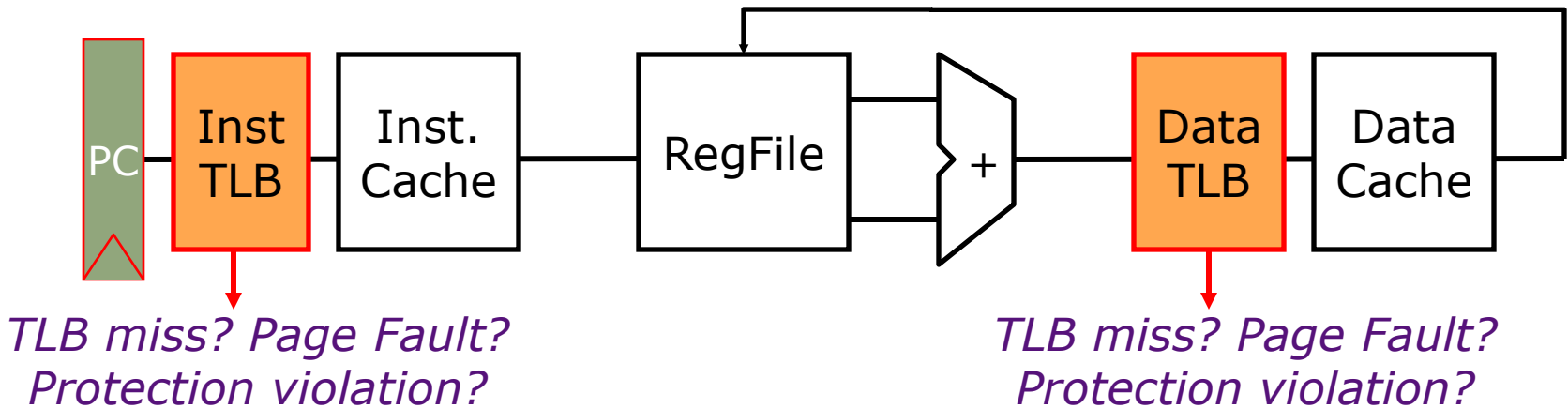
# Topics

---

- Interrupts
- Speeding up the common case:
  - TLB & Cache organization
- Speeding up page table walks
- Modern Usage

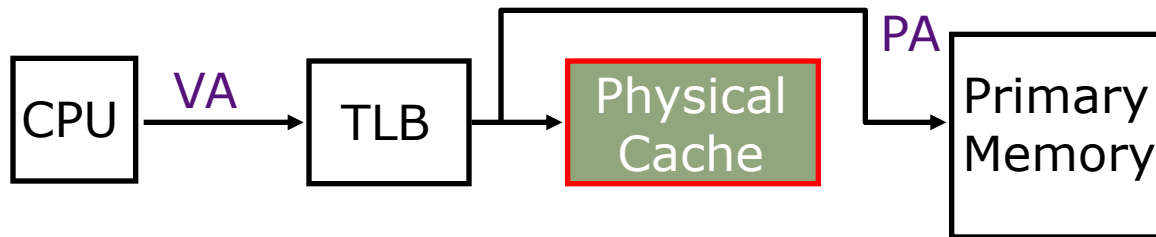


# Address Translation in CPU

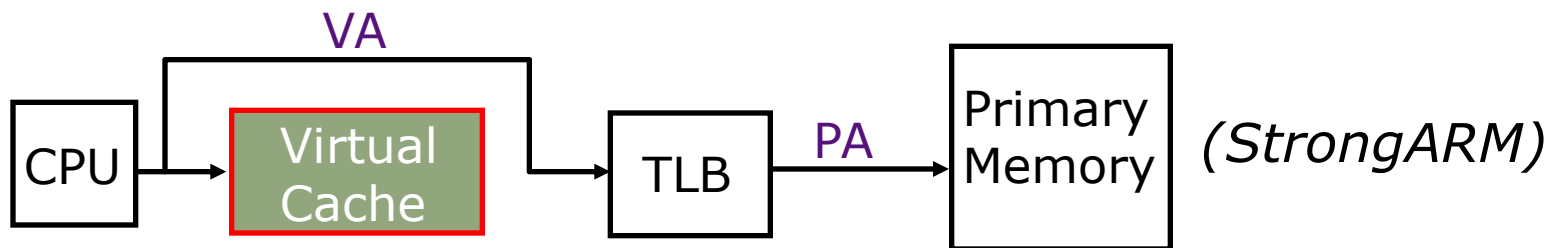


- Software handlers need a *restartable* exception on page fault or protection violation
- Handling a TLB miss needs a *hardware* or *software* mechanism to refill TLB
- Need mechanisms to cope with the additional latency of a TLB:
  - slow down the clock
  - pipeline the TLB and cache access
  - virtual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

# Virtual Address Caches

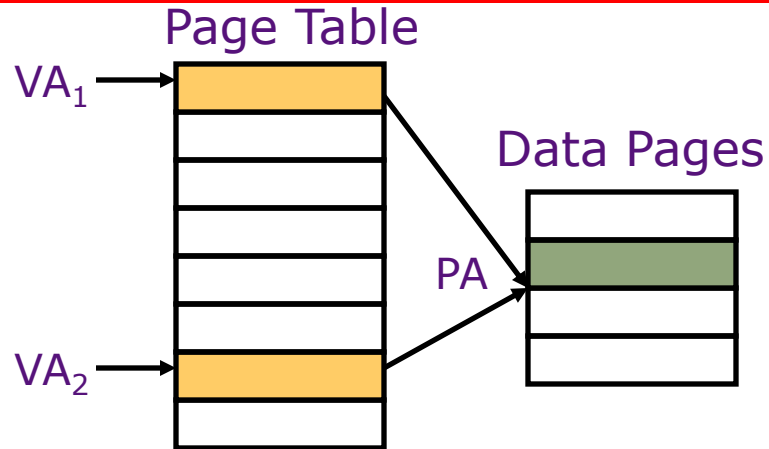


*Alternative: place the cache before the TLB*



- one-step process in case of a hit (+)
- cache needs to be flushed on a context switch unless address space identifiers (ASIDs) included in tags (-)
- *aliasing problems* due to the sharing of pages (-)

# Aliasing in Virtual-Address Caches



Two virtual pages share one physical page

Tag	Data
VA <sub>1</sub>	1st Copy of Data at PA
VA <sub>2</sub>	2nd Copy of Data at PA

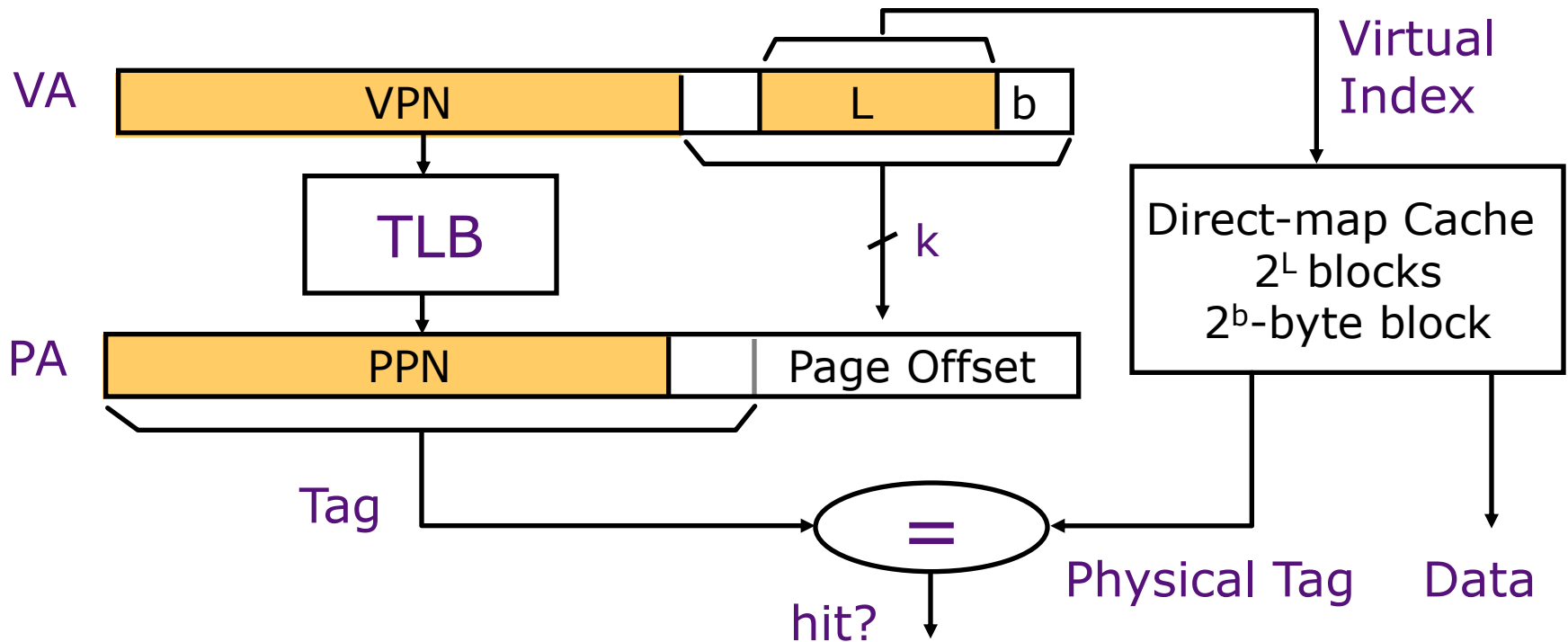
Virtual cache can have two copies of same physical data. Writes to one copy not visible to reads of other!

General Solution: *Disallow aliases to coexist in cache*

Software (i.e., OS) solution for direct-mapped cache

VAs of shared pages must agree in cache index bits; this ensures all VAs accessing same PA will conflict in direct-mapped cache (early SPARCs)

# Concurrent Access to TLB & Cache



Index  $L$  is available without consulting the TLB

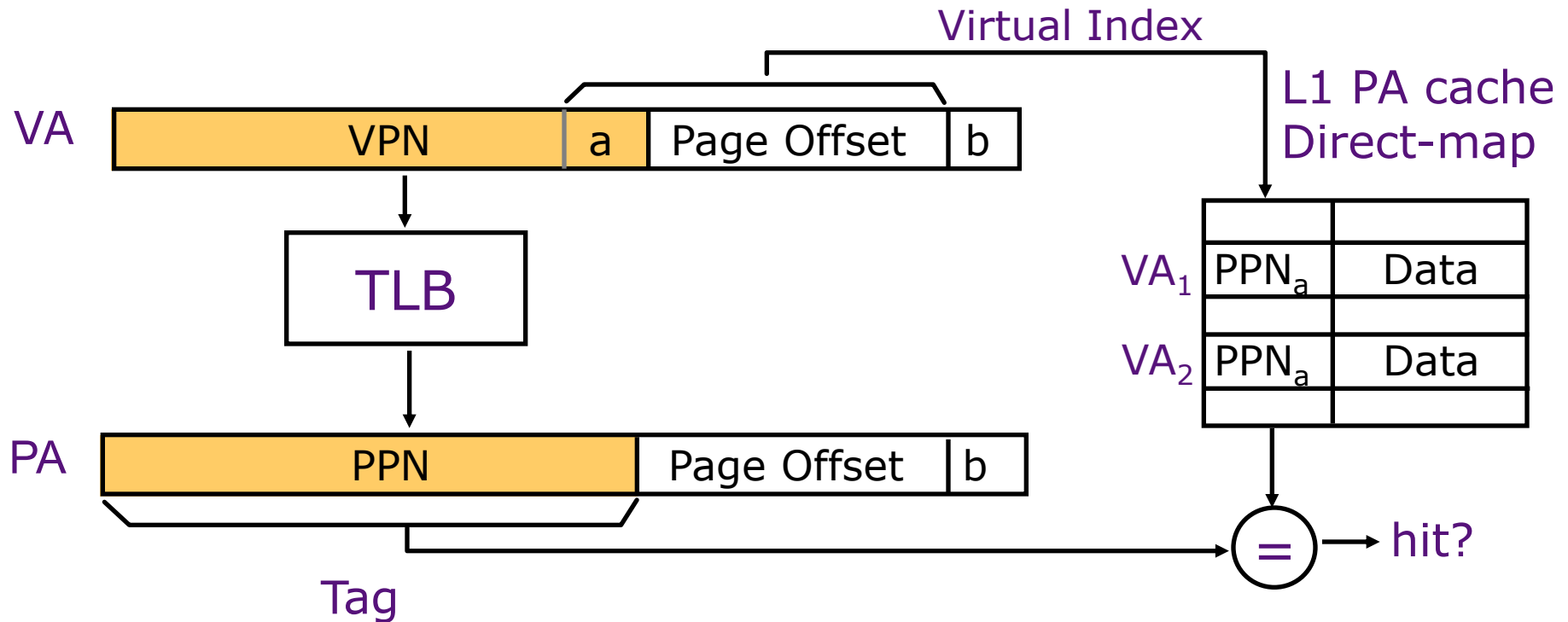
$\Rightarrow$  *cache and TLB accesses can begin simultaneously*

Tag comparison is made after both accesses are completed

*Cases:  $L + b = k$      $L + b < k$      $L + b > k$*

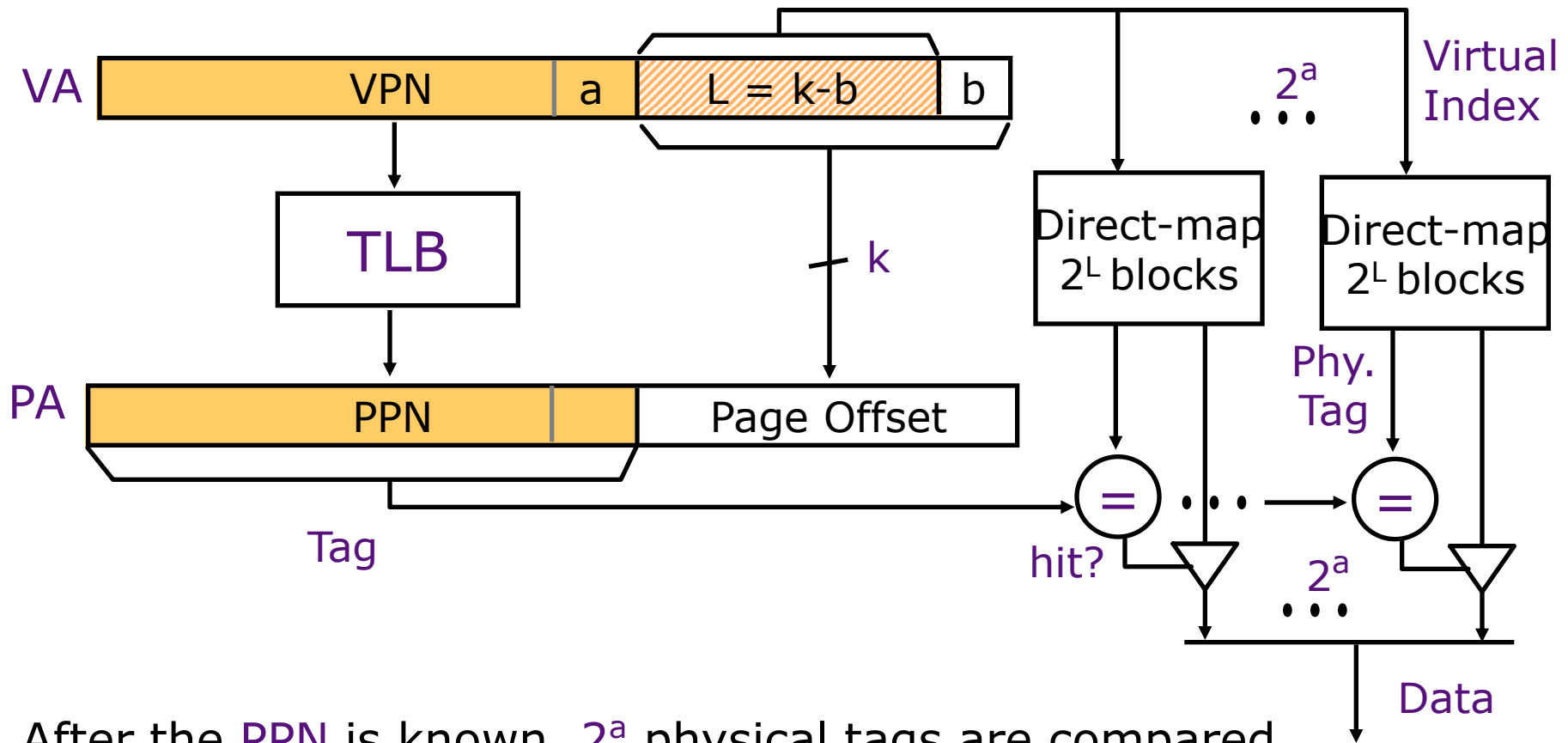
# Concurrent Access to TLB & Large L1

The problem with  $L1 > \text{Page size}$



*Can VA<sub>1</sub> and VA<sub>2</sub> both map to PA?*

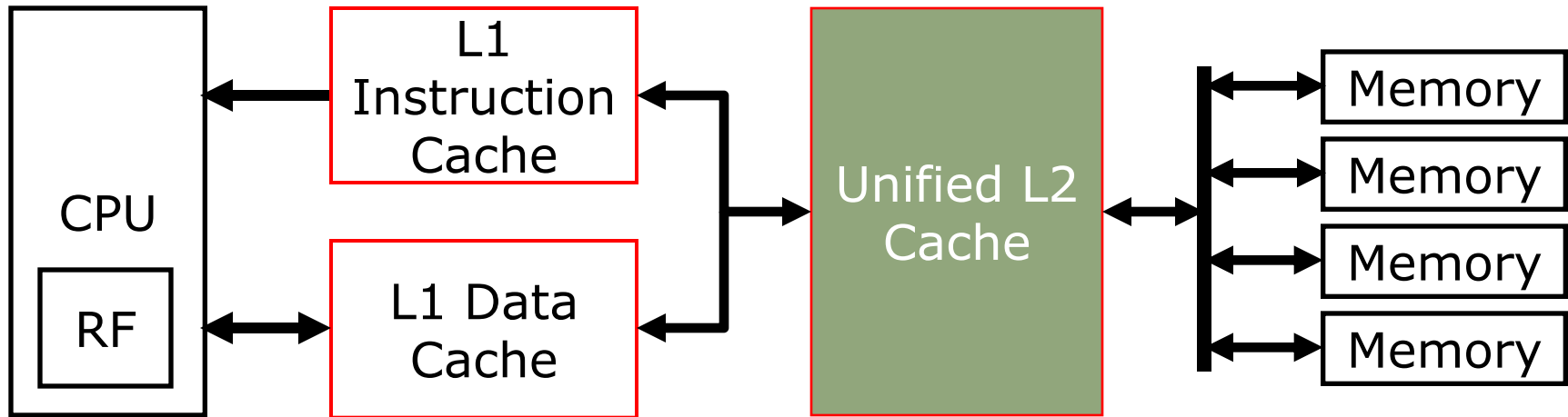
# Virtual-Index Physical-Tag Caches: Associative Organization



*Is this scheme realistic?*

# A solution via Second-Level Cache

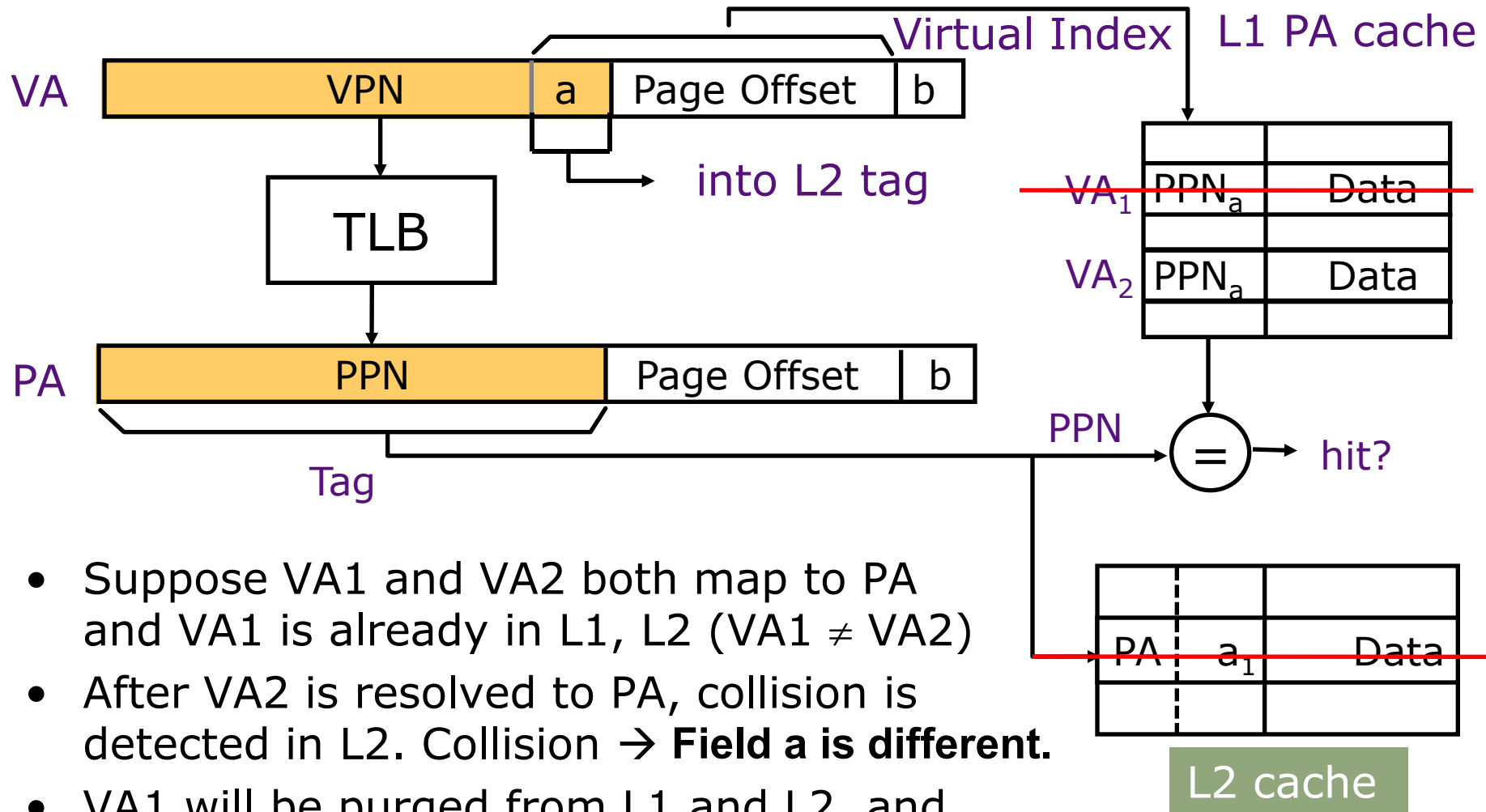
---



Usually a common L2 cache backs up both Instruction and Data L1 caches

L2 is "inclusive" of both Instruction and Data caches

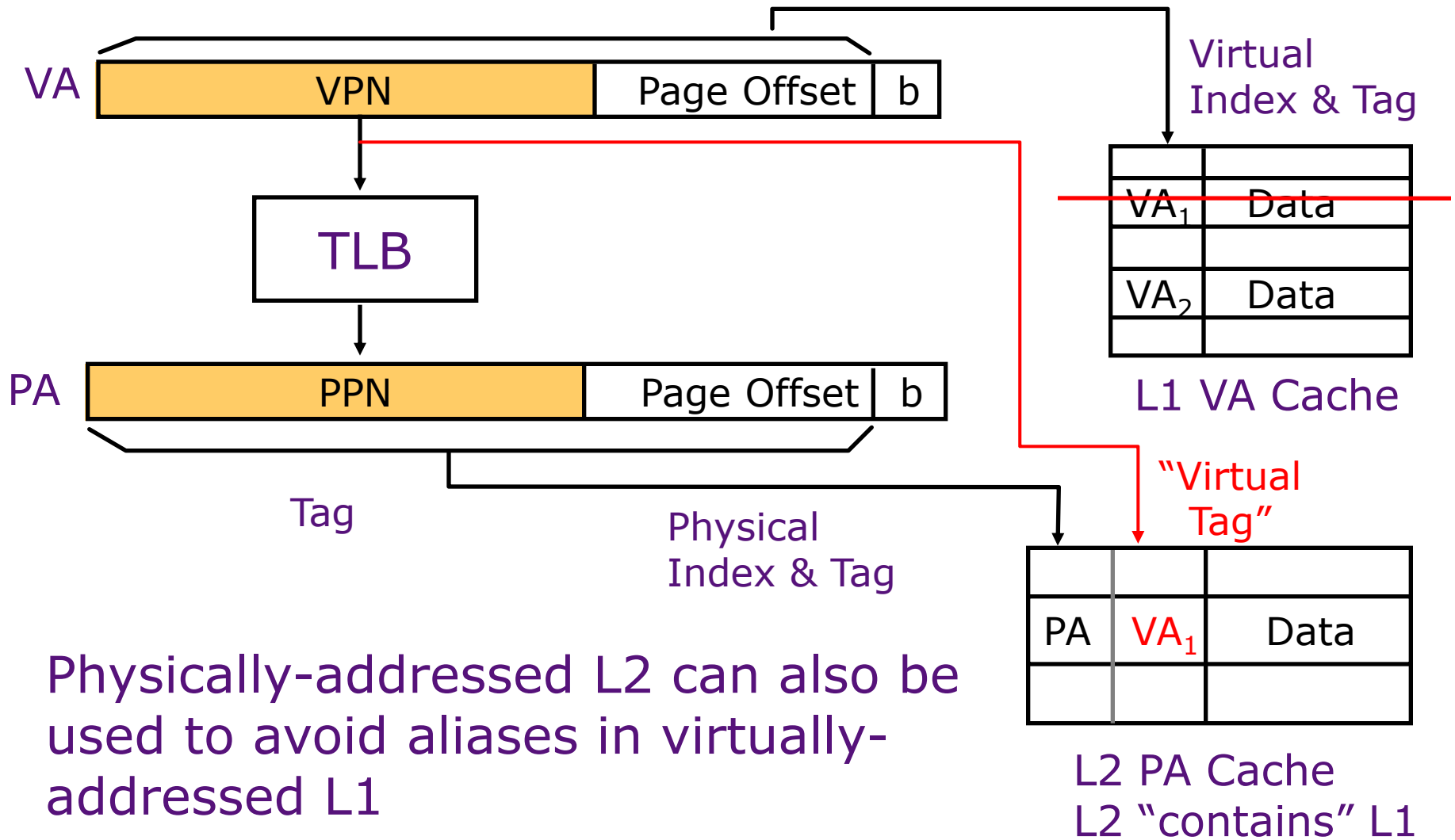
# Anti-Aliasing Using L2: *MIPS R10000*



- Suppose VA1 and VA2 both map to PA and VA1 is already in L1, L2 (VA1 ≠ VA2)
- After VA2 is resolved to PA, collision is detected in L2. Collision → **Field a is different.**
- VA1 will be purged from L1 and L2, and VA2 will be loaded ⇒ *no aliasing!*



# Virtually-Addressed L1: Anti-Aliasing using L2



# Topics

---

- Interrupts
- Speeding up the common case:
  - TLB & Cache organization
- Speeding up page table walks
- Modern Usage

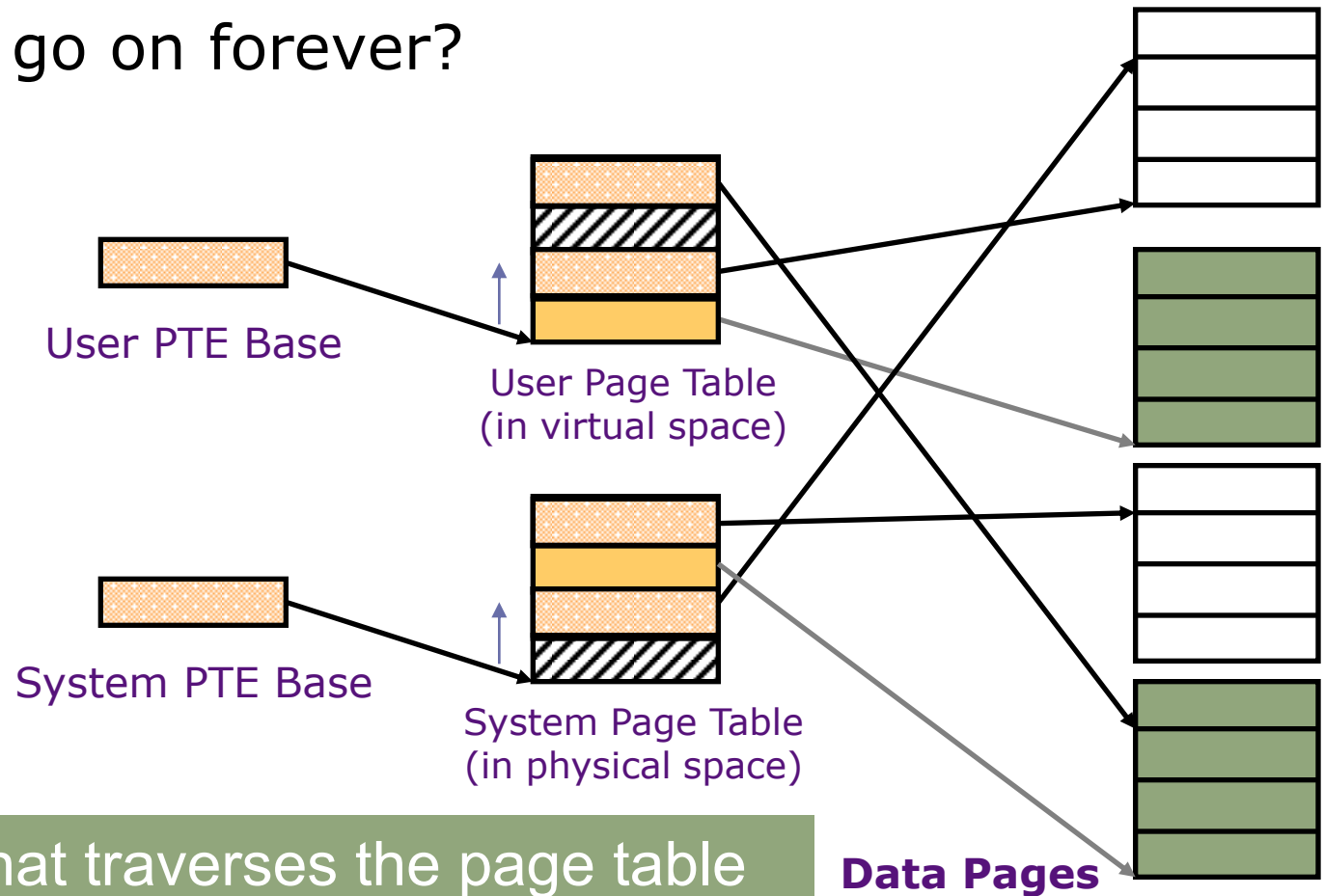
# Page Fault Handler

---

- 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
    - Pseudo-LRU replacement policy*
- Since it takes a long time to transfer a page (msecs), page faults are handled completely in software by the OS
  - Untranslated addressing mode is essential to allow kernel to access page tables

# Translation for Page Tables

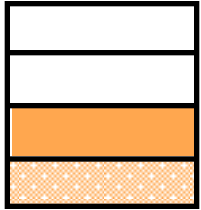
- Can references to page tables cause TLB misses?
- Can this go on forever?



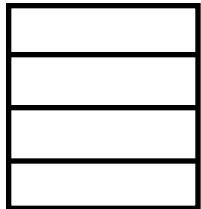
A program that traverses the page table needs a “no translation” addressing mode.

# Swapping a Page of a Page Table

---



A PTE in primary memory contains primary or secondary memory addresses



A PTE in secondary memory contains *only* secondary memory addresses

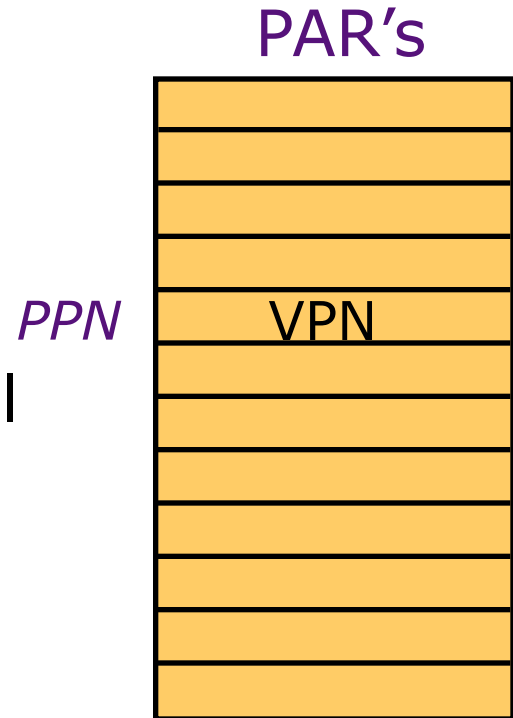
⇒ a page of a PT can be swapped out only if none of its PTE's point to pages in the primary memory

*Why?* \_\_\_\_\_

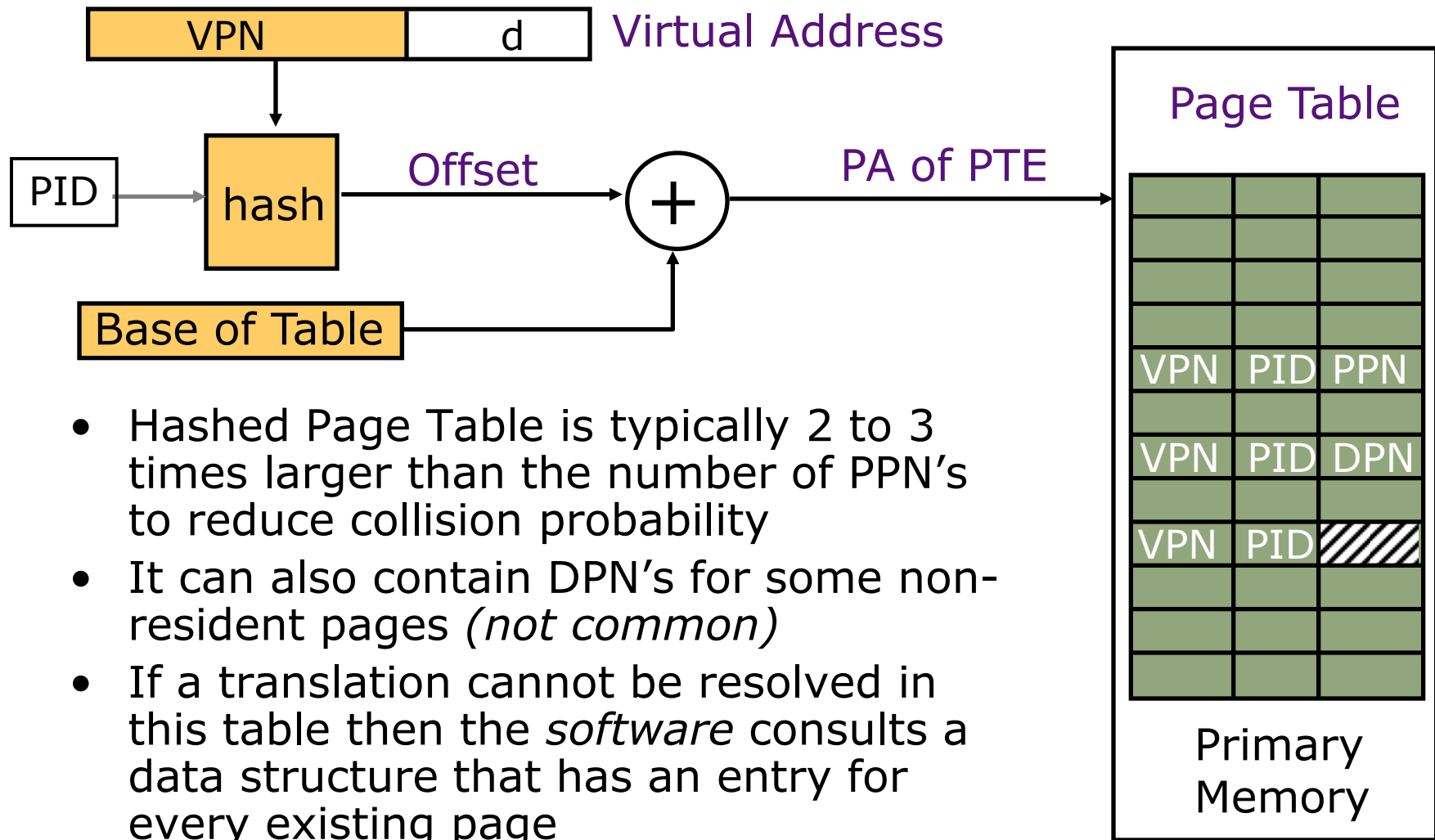
# Atlas Revisited

---

- One PAR for each physical page
- PAR's contain the VPN's of the pages *resident in primary memory*
- *Advantage:* The size is proportional to the size of the primary memory
- *What is the disadvantage?*



# Hashed Page Table: Approximating Associative Addressing



- Hashed Page Table is typically 2 to 3 times larger than the number of PPN's to reduce collision probability
- It can also contain DPN's for some non-resident pages (*not common*)
- If a translation cannot be resolved in this table then the *software* consults a data structure that has an entry for every existing page

# Virtual Memory Use Today - 1

---

- Desktop/server/cellphone processors have full demand-paged virtual memory
  - Portability between machines with different memory sizes
  - Protection between multiple users or multiple tasks
  - Share small physical memory among active tasks
  - Simplifies implementation of some OS features
- Vector supercomputers and GPUs have translation and protection but not demand-paging (Older Crays: base&bound, Japanese & Cray X1: pages)
  - Don't waste expensive processor time thrashing to disk (make jobs fit in memory)
  - Mostly run in batch mode (run set of jobs that fits in memory)
  - Difficult to implement restartable vector instructions



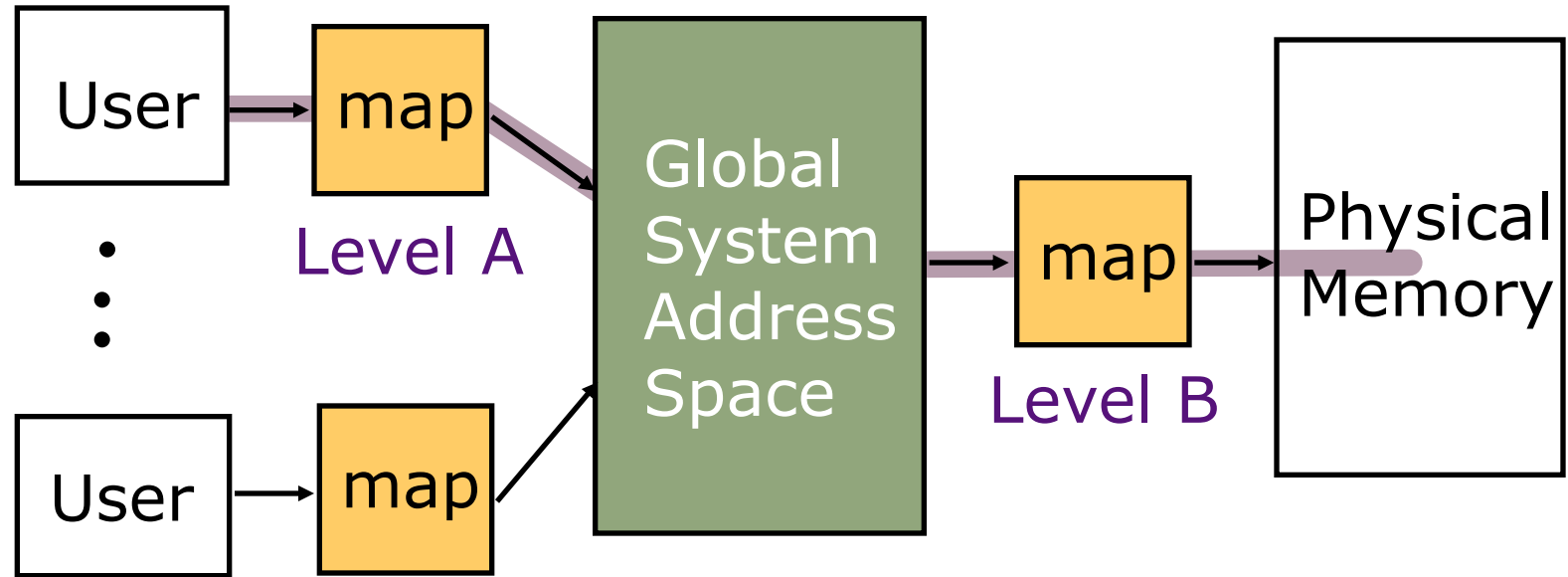
# Virtual Memory Use Today - 2

---

- Most embedded processors and DSPs provide physical addressing only
  - Can't afford area/speed/power budget for virtual memory support
  - Often there is no secondary storage to swap to!
  - Programs custom-written for particular memory configuration in product
  - Difficult to implement restartable instructions for exposed architectures

*Next lecture: Pipelining!*

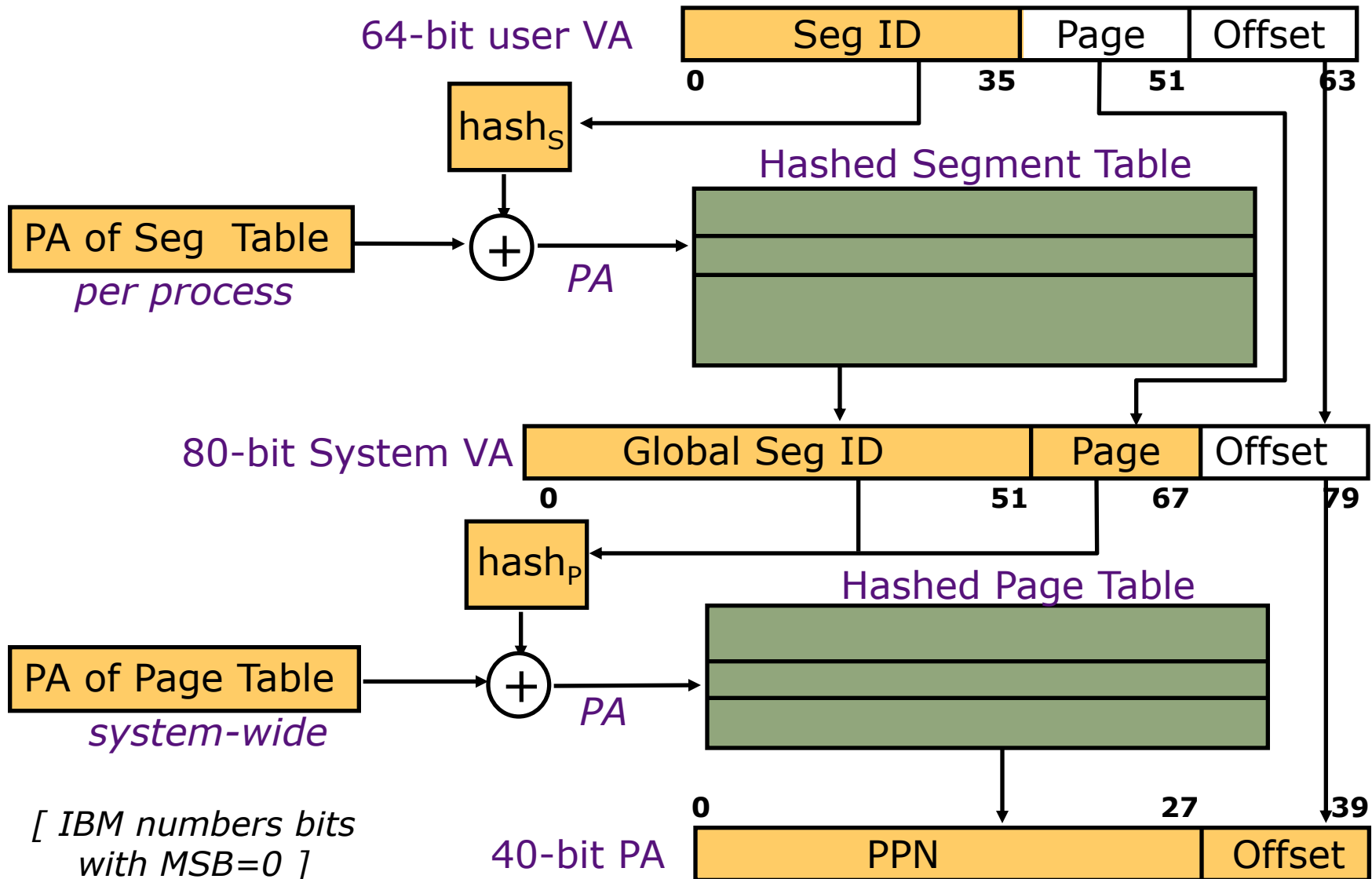
# Global System Address Space



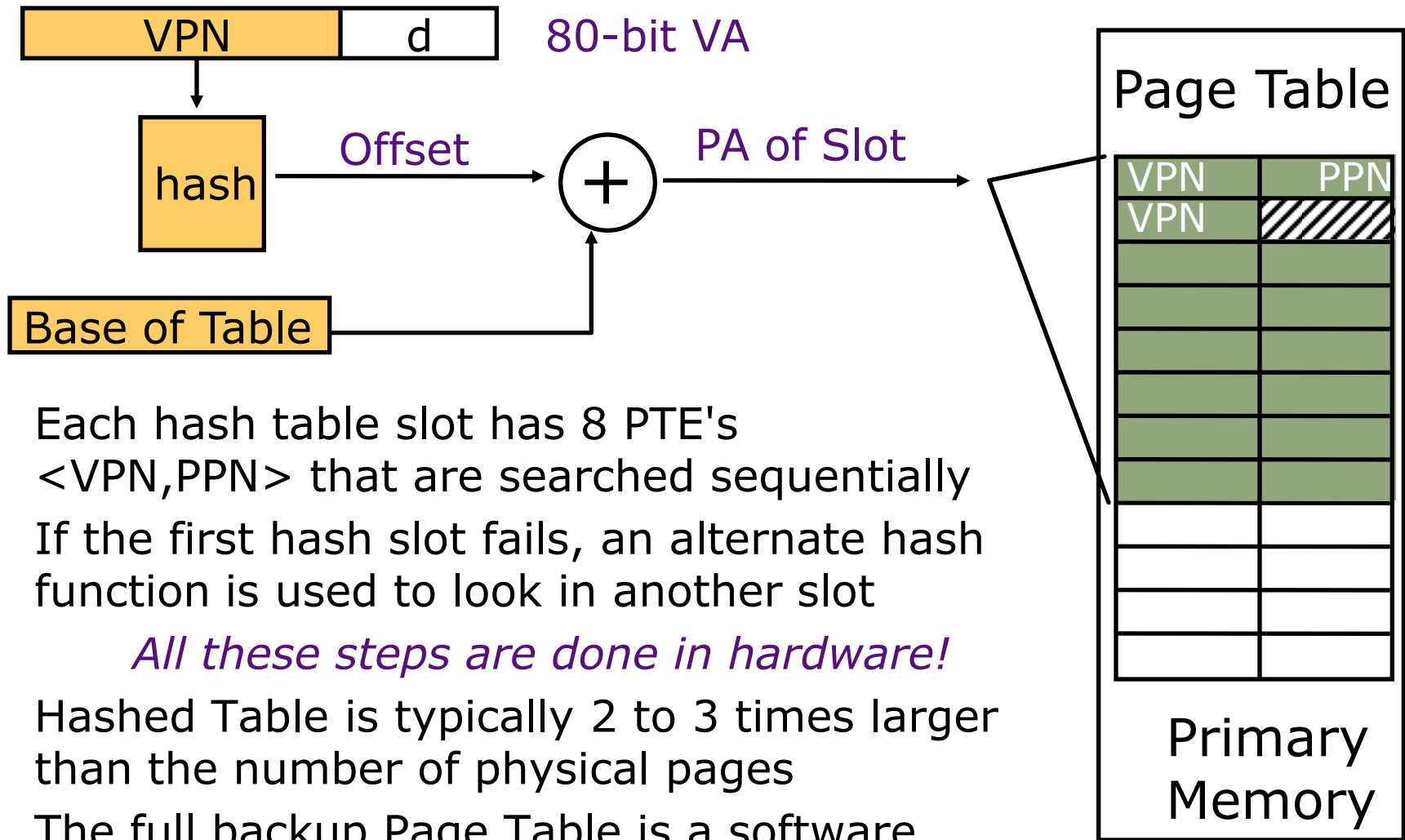
- Level A maps users' address spaces into the global space providing privacy, protection, sharing etc.
- Level B provides demand-paging for the large global system address space
- Level A and Level B translations may be kept in separate TLB's

# Hashed Page Table Walk:

## PowerPC Two-level, Segmented Addressing



# Power PC: Hashed Page Table



- Each hash table slot has 8 PTE's  $\langle \text{VPN}, \text{PPN} \rangle$  that are searched sequentially
- If the first hash slot fails, an alternate hash function is used to look in another slot
- *All these steps are done in hardware!*
- Hashed Table is typically 2 to 3 times larger than the number of physical pages
- The full backup Page Table is a software data structure