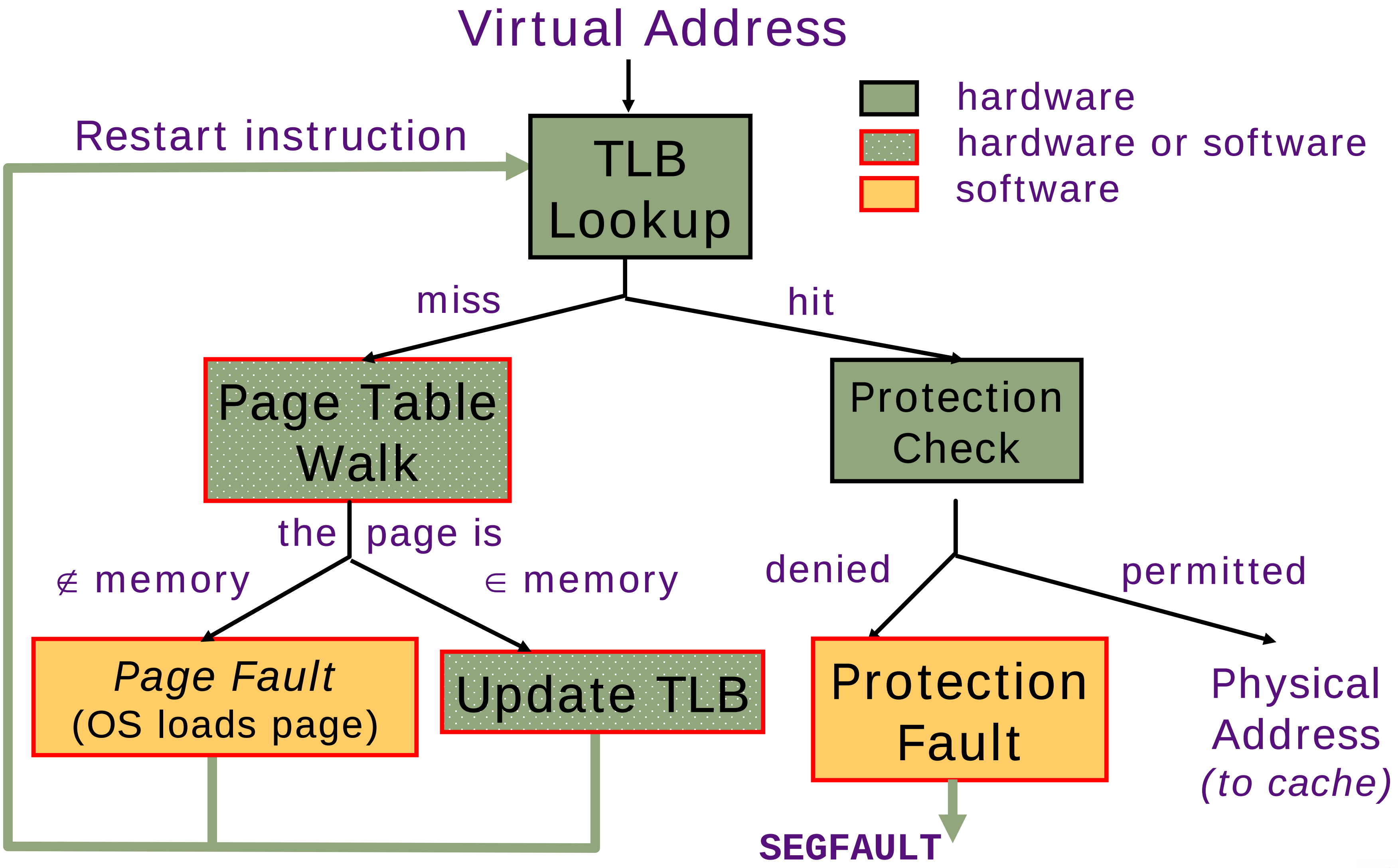
# Modern Virtual Memory Systems

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

*Based on the material prepared by*
*Arvind and Krste Asanovic*

# Address Translation:
## *putting it all together*



Virtual Address

- 🟩 hardware
- ▩ hardware or software
- 🟧 software

Restart instruction

TLB Lookup

miss

hit

Page Table Walk

Protection Check

the page is

$\notin$ memory

$\in$ memory

denied

permitted

*Page Fault*
(OS loads page)

Update TLB

Protection Fault

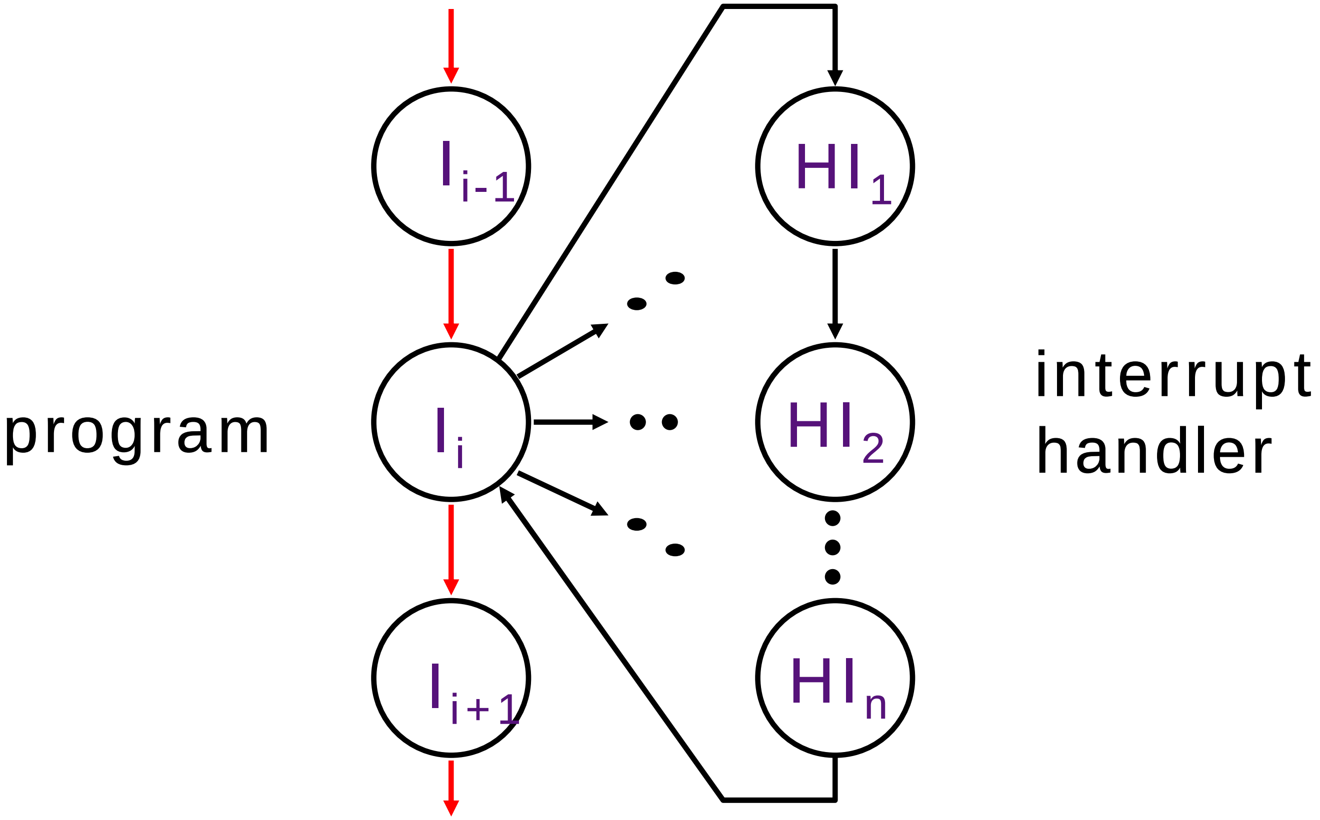
Physical Address
*(to cache)*

**SEGFAULT**

# 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 exceptions)*
  - 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

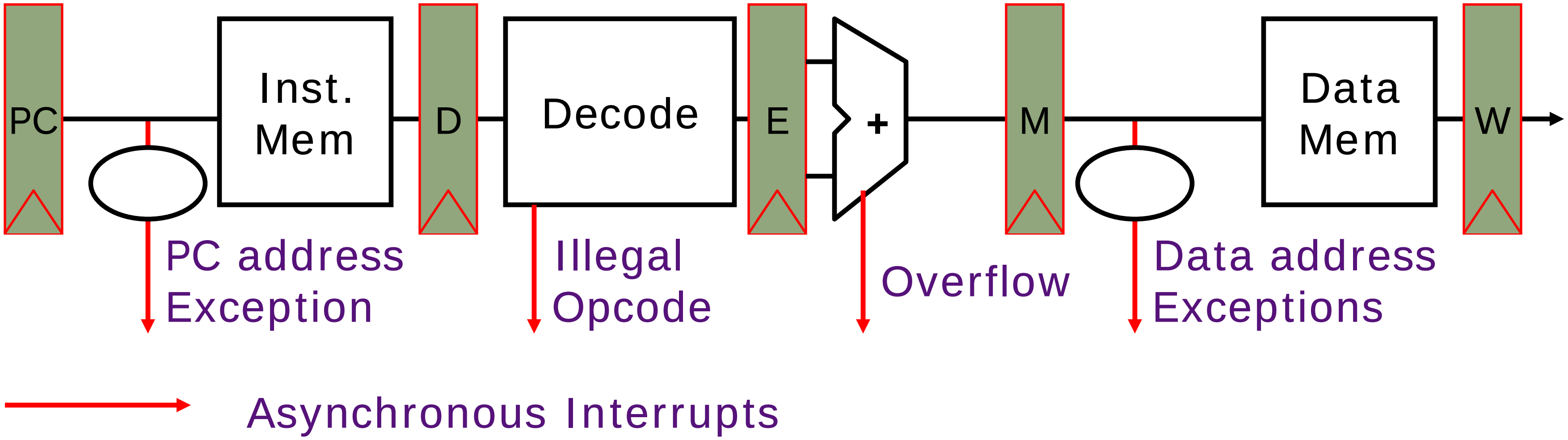
# Interrupt Handler

- Saves EPC before enabling interrupts to allow nested interrupts ⇒
  - 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*) which
  - enables interrupts
  - restores the processor to the user mode
  - restores hardware status and control state

# Synchronous Interrupts

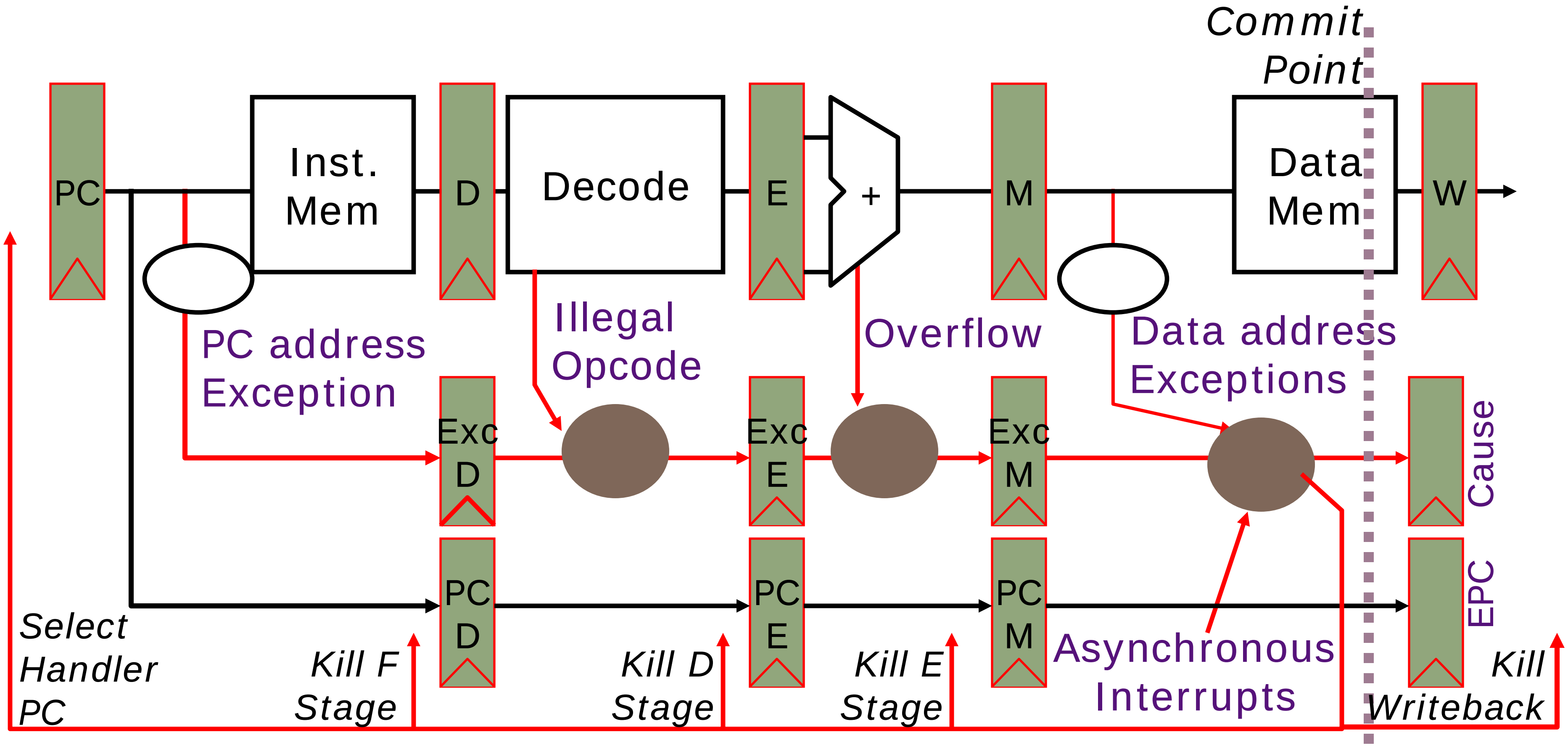
- A synchronous interrupt (exception) is caused by a *particular instruction*
- In general, the instruction cannot be completed and needs to be *restarted* after the exception has been handled
  - requires undoing the effect of one or more partially executed instructions
- In case of a trap (system call),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

# Exception Handling 5-Stage Pipeline



- How to handle multiple simultaneous exceptions in different pipeline stages?
- How and where to handle external asynchronous interrupts?

# Exception Handling 5-Stage Pipeline



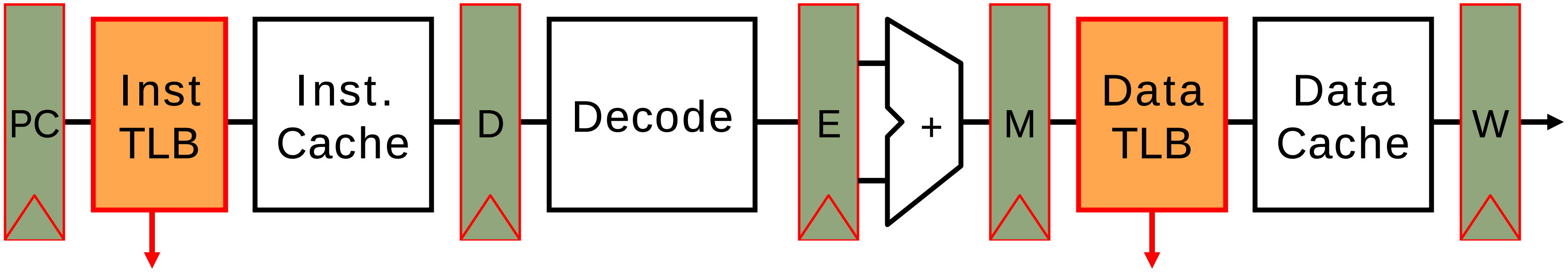
# Exception Handling 5-Stage Pipeline

- Hold exception flags in pipeline until commit point (M stage)
- Exceptions in earlier pipe stages override later exceptions *for a given instruction*
- Inject external interrupts at commit point (override others)
- If exception at commit: update Cause and EPC registers, kill all stages, inject handler PC into fetch stage

# Topics

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

# Address Translation in CPU Pipeline

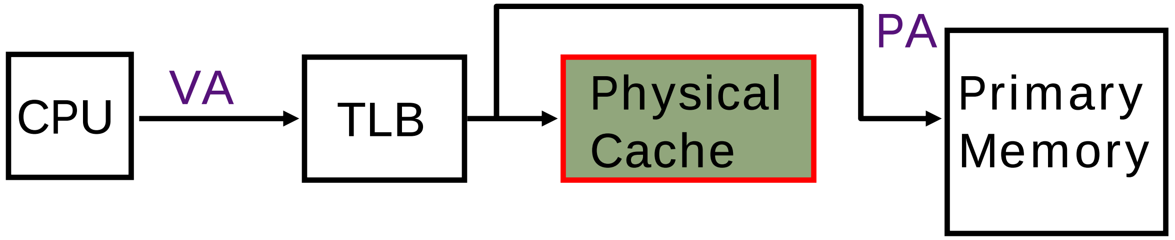


*TLB miss? Page Fault?*
*Protection violation?*

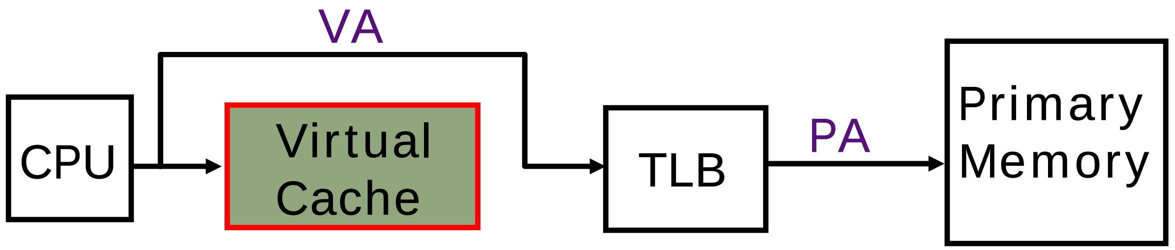
*TLB miss? Page Fault?*
*Protection violation?*

- 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

# Virtual Address Caches



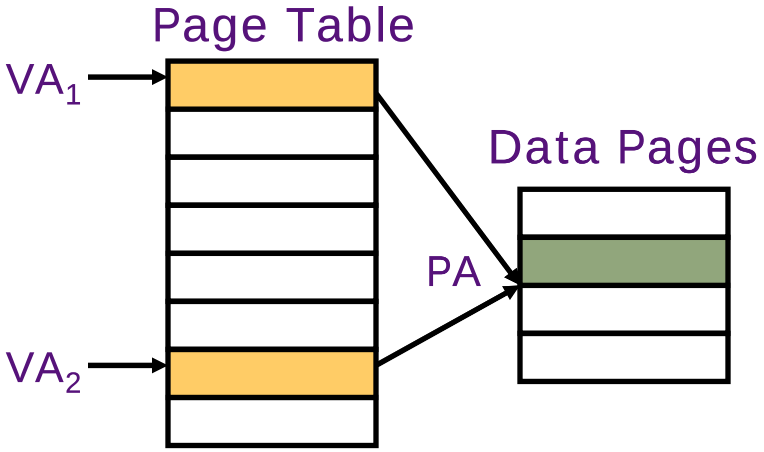
*Alternative: place the cache before the TLB*



*(StrongARM)*

- 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 |
|---|---|
| | |
| $VA_1$ | 1st Copy of Data at PA |
| | |
| | |
| $VA_2$ | 2nd Copy of Data at PA |
| | |

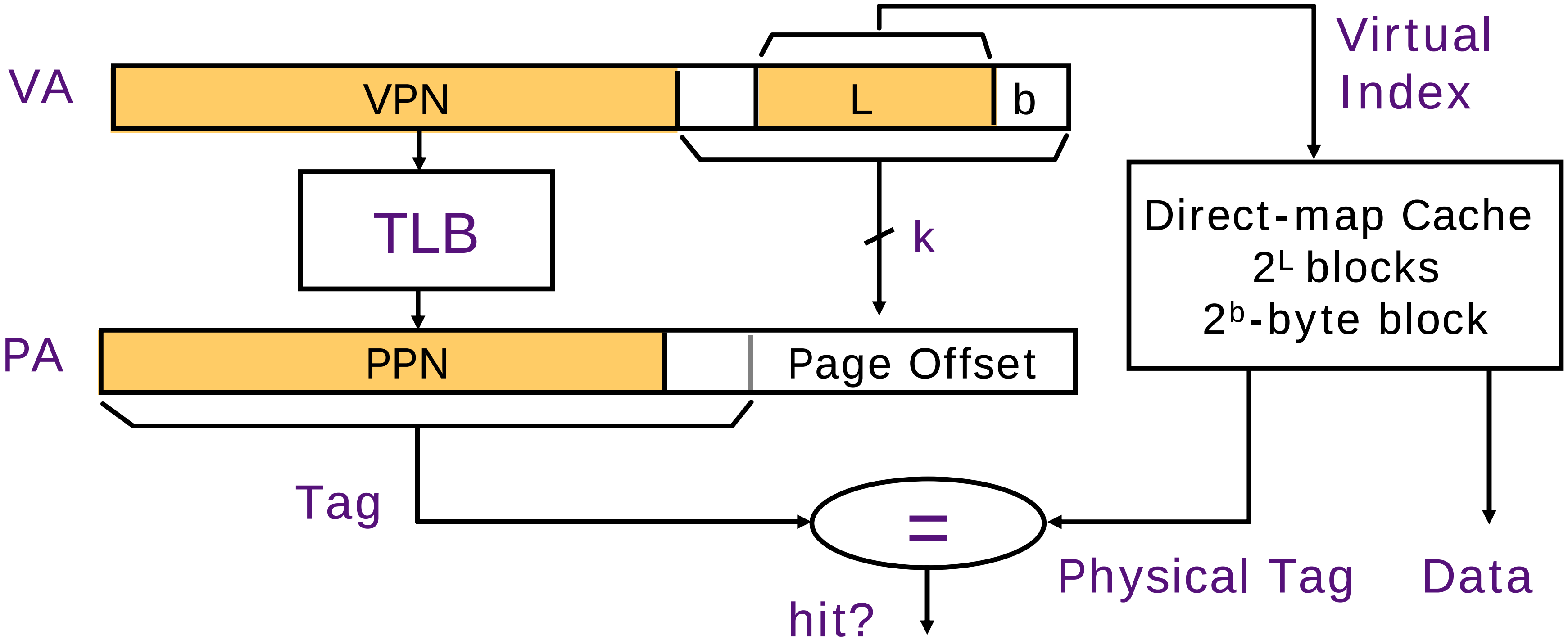
Virtual cache can have two copies of same physical data. Writes to one copy not visible to reads of other!

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



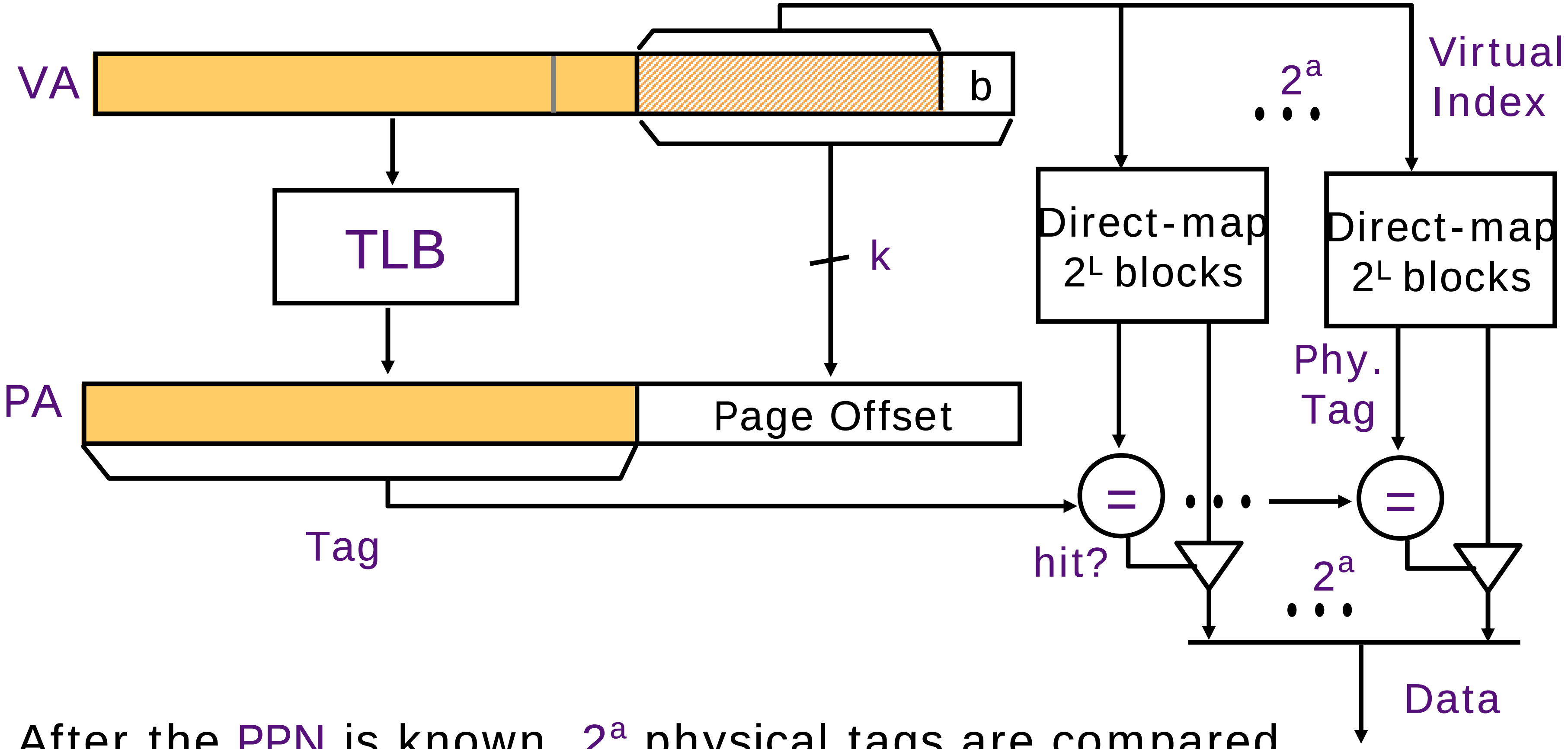
Index L is available without consulting the TLB

⇒ *cache and TLB accesses can begin simultaneously*

Tag comparison is made after both accesses are completed

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

# Virtual-Index Physical-Tag Caches:
## Associative Organization

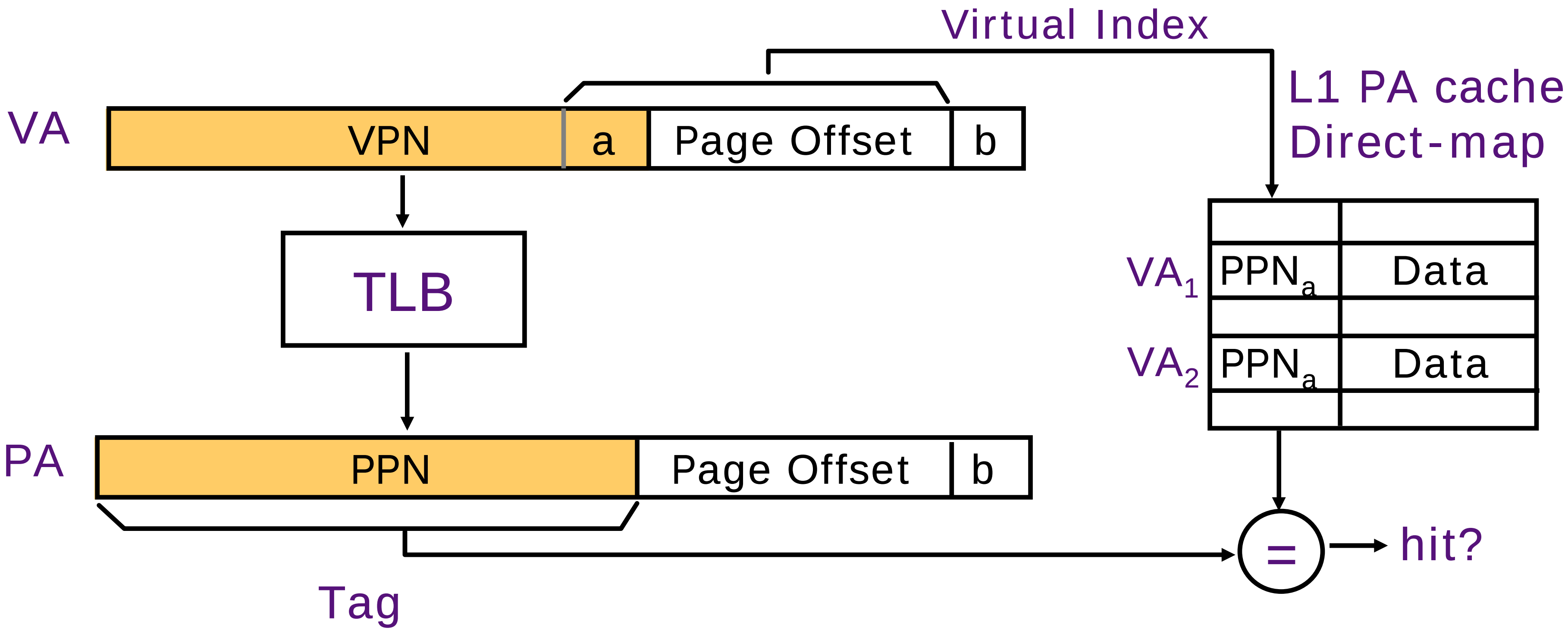


After the PPN is known, $2^a$ physical tags are compared

*Is this scheme realistic?*

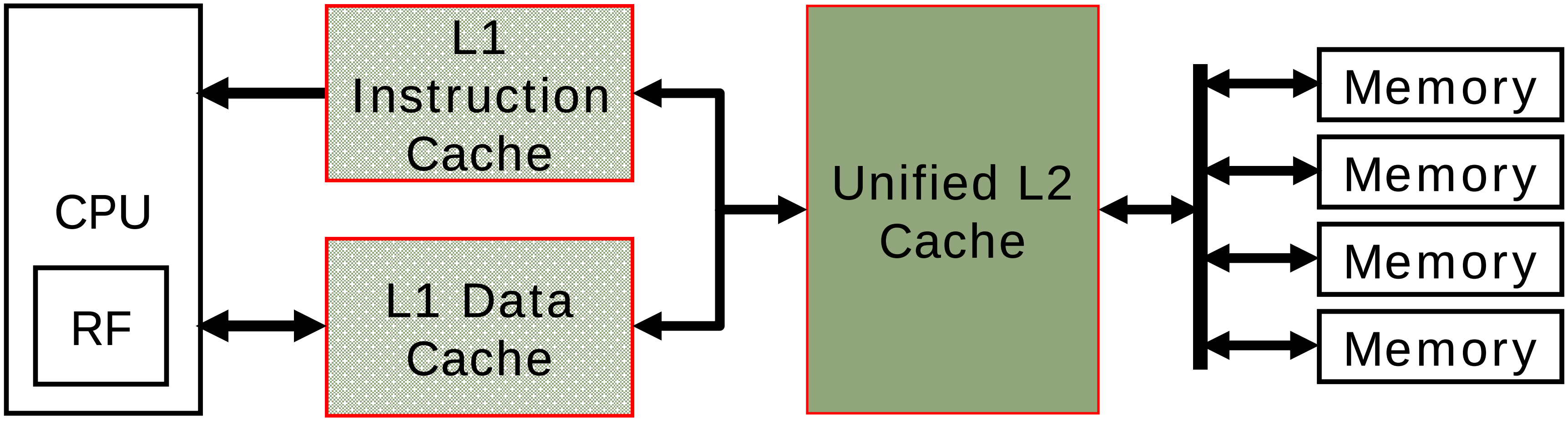
# Concurrent Access to TLB & Large L1

## The problem with L1 > Page size



*Can* VA$_1$ *and* VA$_2$ *both map to* PA *?*

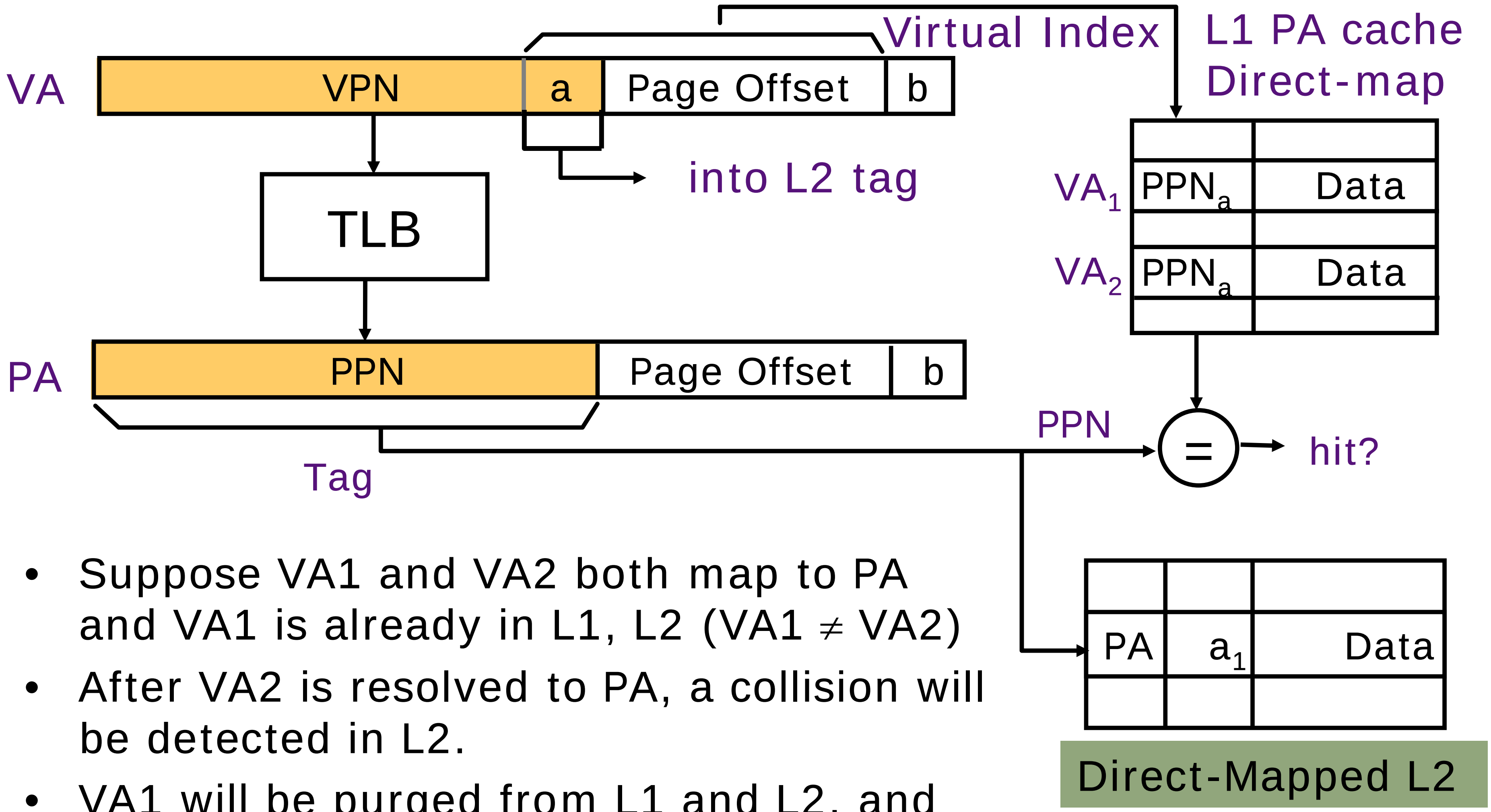
# A solution via Second Level Cache



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

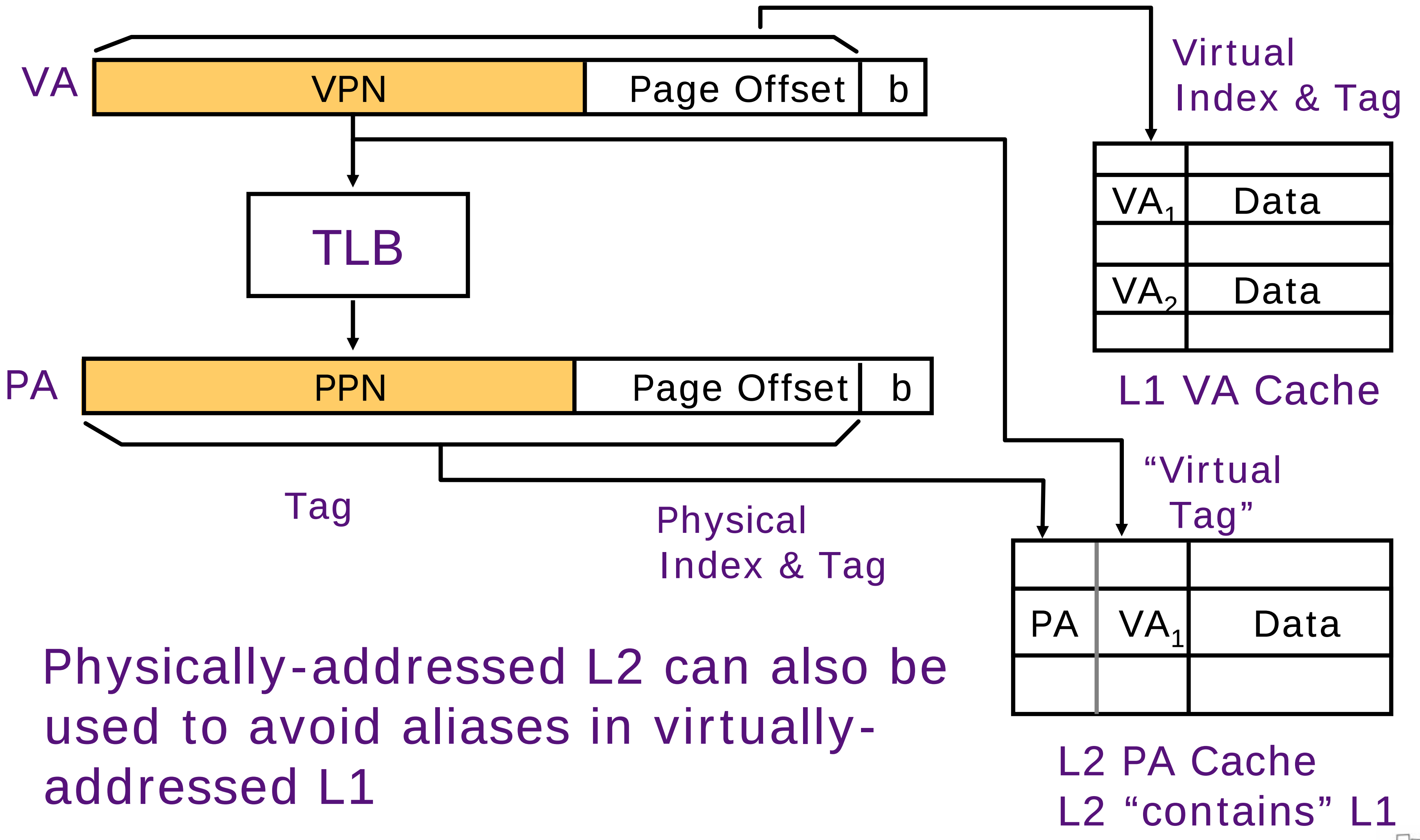
L2 is “inclusive” of both Instruction and Data caches

# Anti-Aliasing Using L2: *MIPS R10000*



- Suppose VA1 and VA2 both map to PA and VA1 is already in L1, L2 (VA1 $\neq$ VA2)
- After VA2 is resolved to PA, a collision will be detected in L2.
- VA1 will be purged from L1 and L2, and VA2 will be loaded $\Rightarrow$ *no aliasing !*

# Virtually-Addressed L1:
## Anti-Aliasing using L2



Physically-addressed L2 can also be used to avoid aliases in virtually-addressed L1

Five-minute break to stretch your legs

# Topics

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

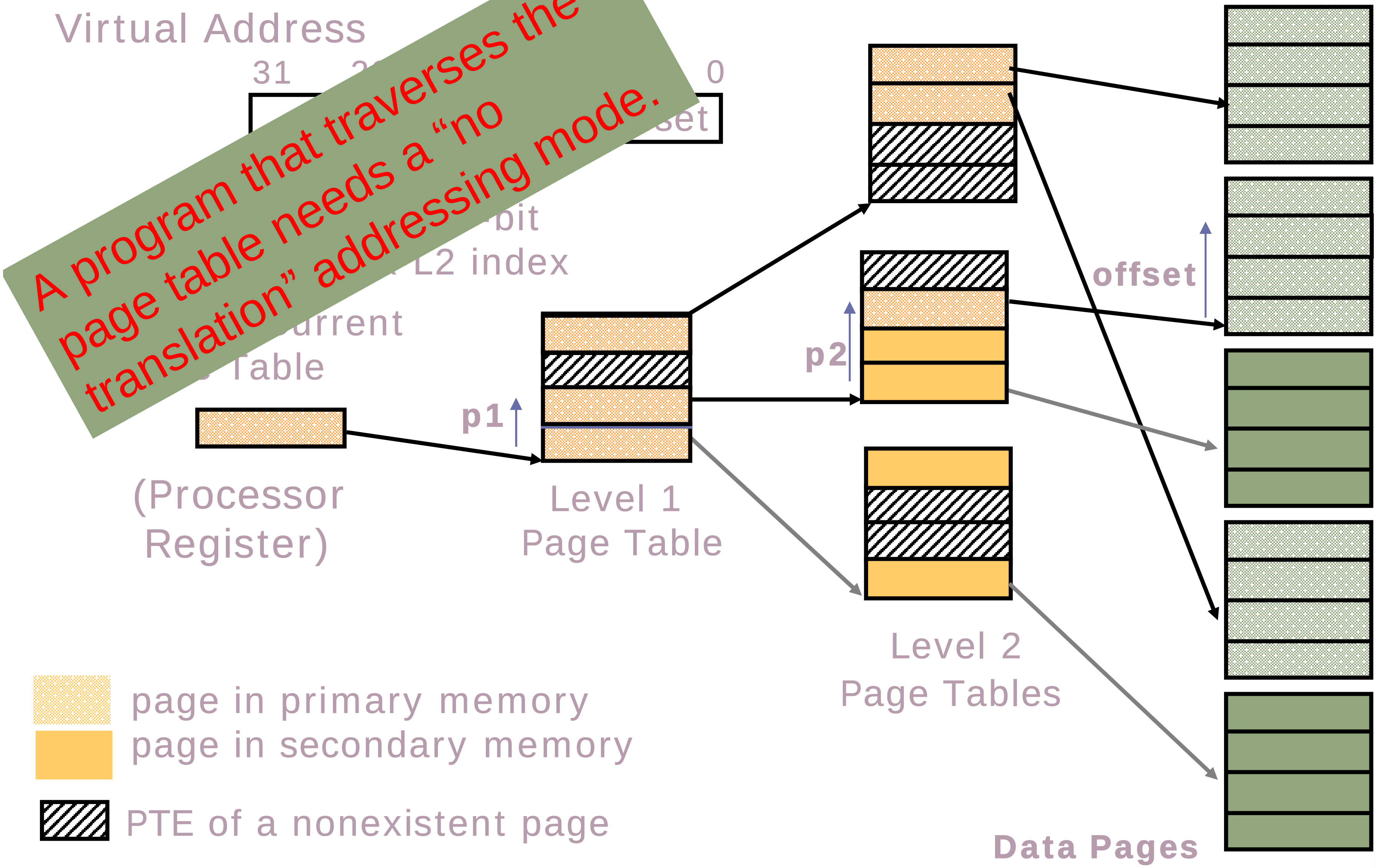
# Page Fault Handler

- 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

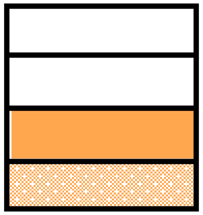
# Hierarchical Page Table



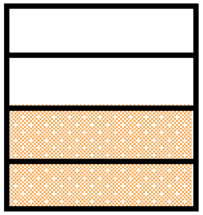
A program that traverses the page table needs a “no translation” addressing mode.

▨ page in primary memory
🟧 page in secondary memory
▨ PTE of a nonexistent page

# Swapping a Page of a Page Table



A PTE in primary memory contains
primary or secondary memory addresses



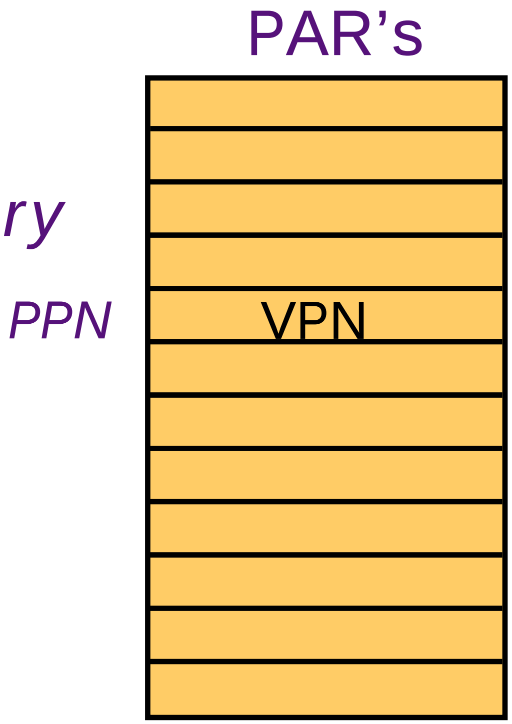
A PTE in secondary memory contains
*only* secondary memory addresses

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

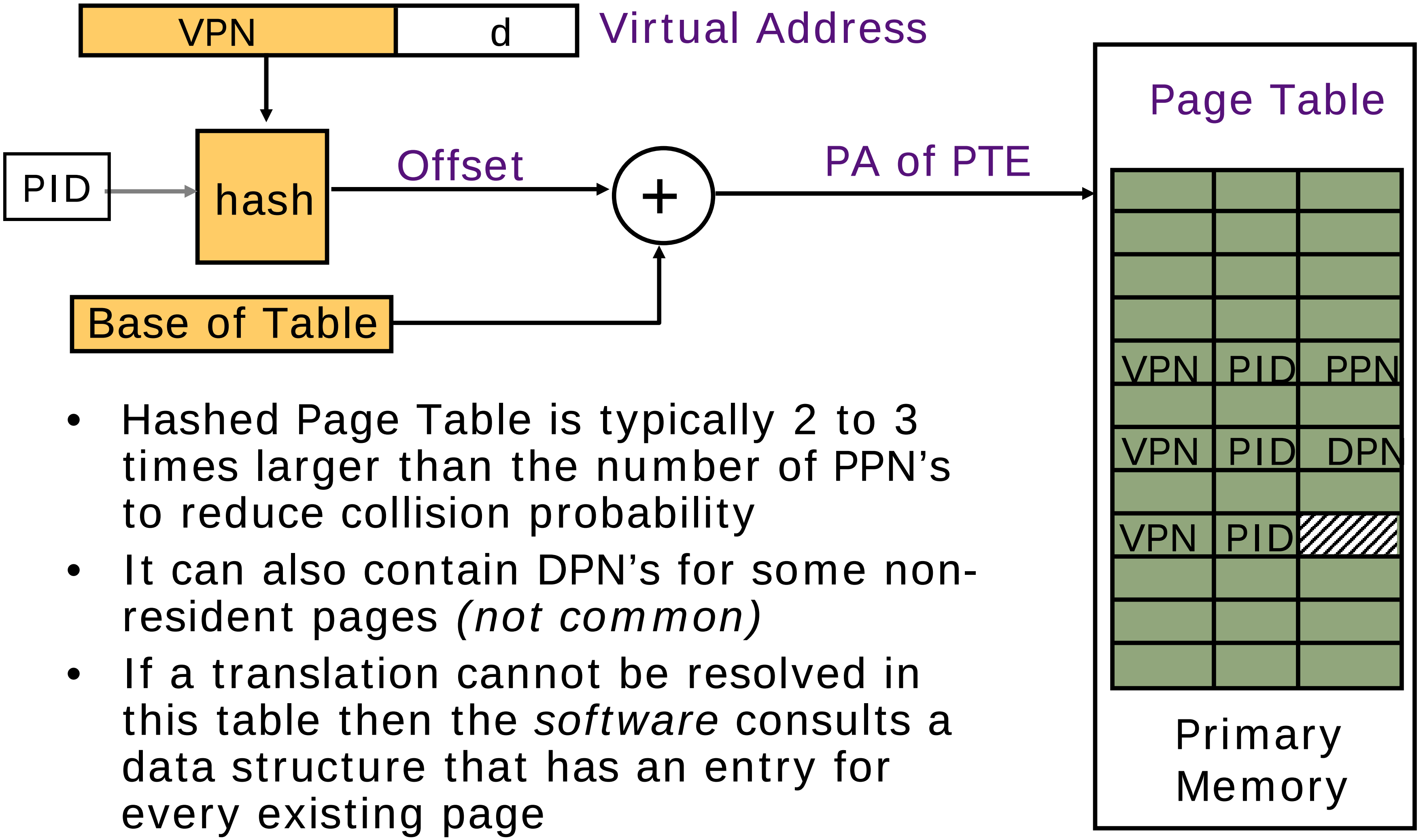
*Why?*____________________________________

# 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 ?*

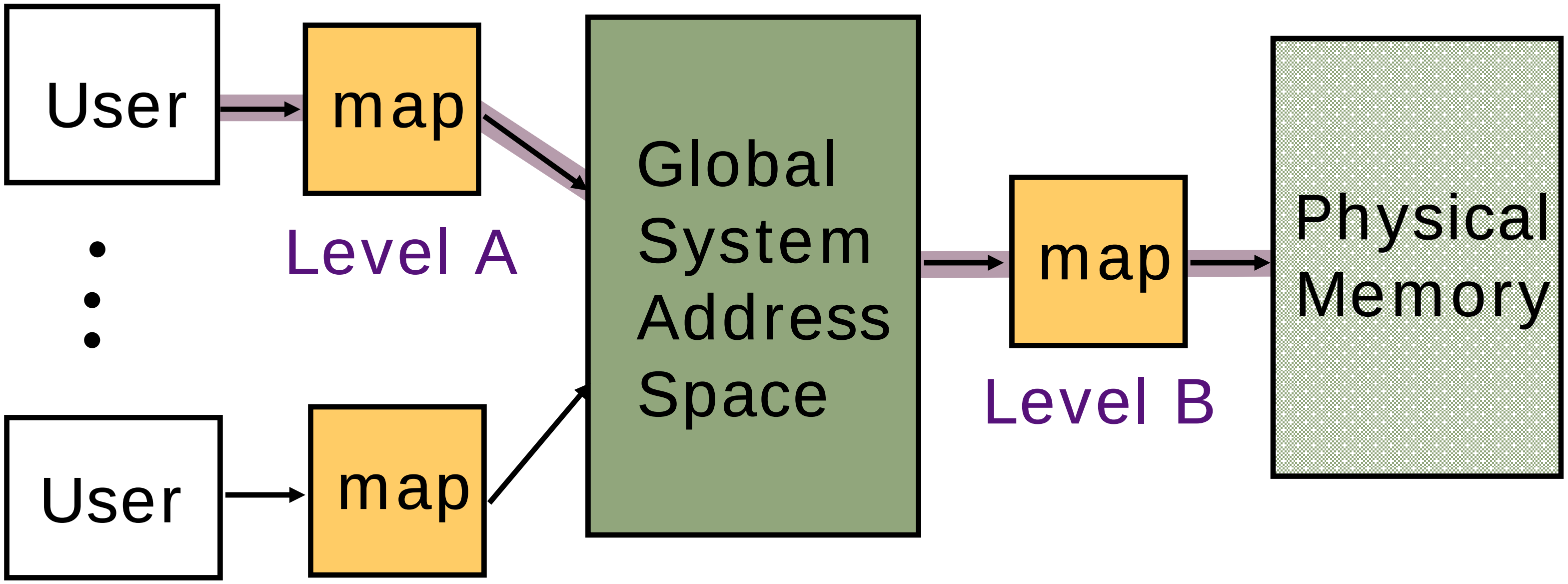


# Hashed Page Table:
## Approximating Associative Addressing



- 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

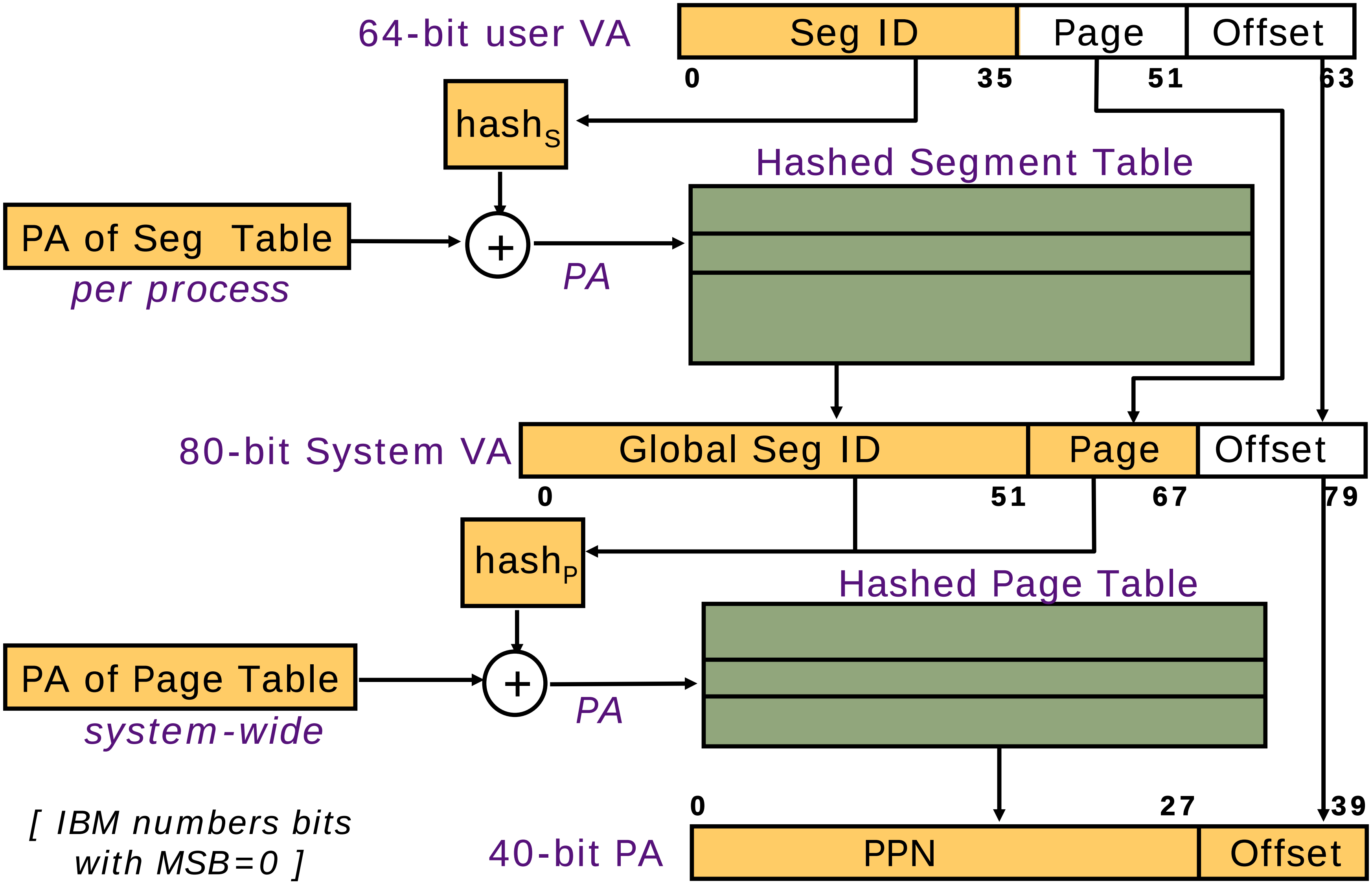
# Global System Address Space



- 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

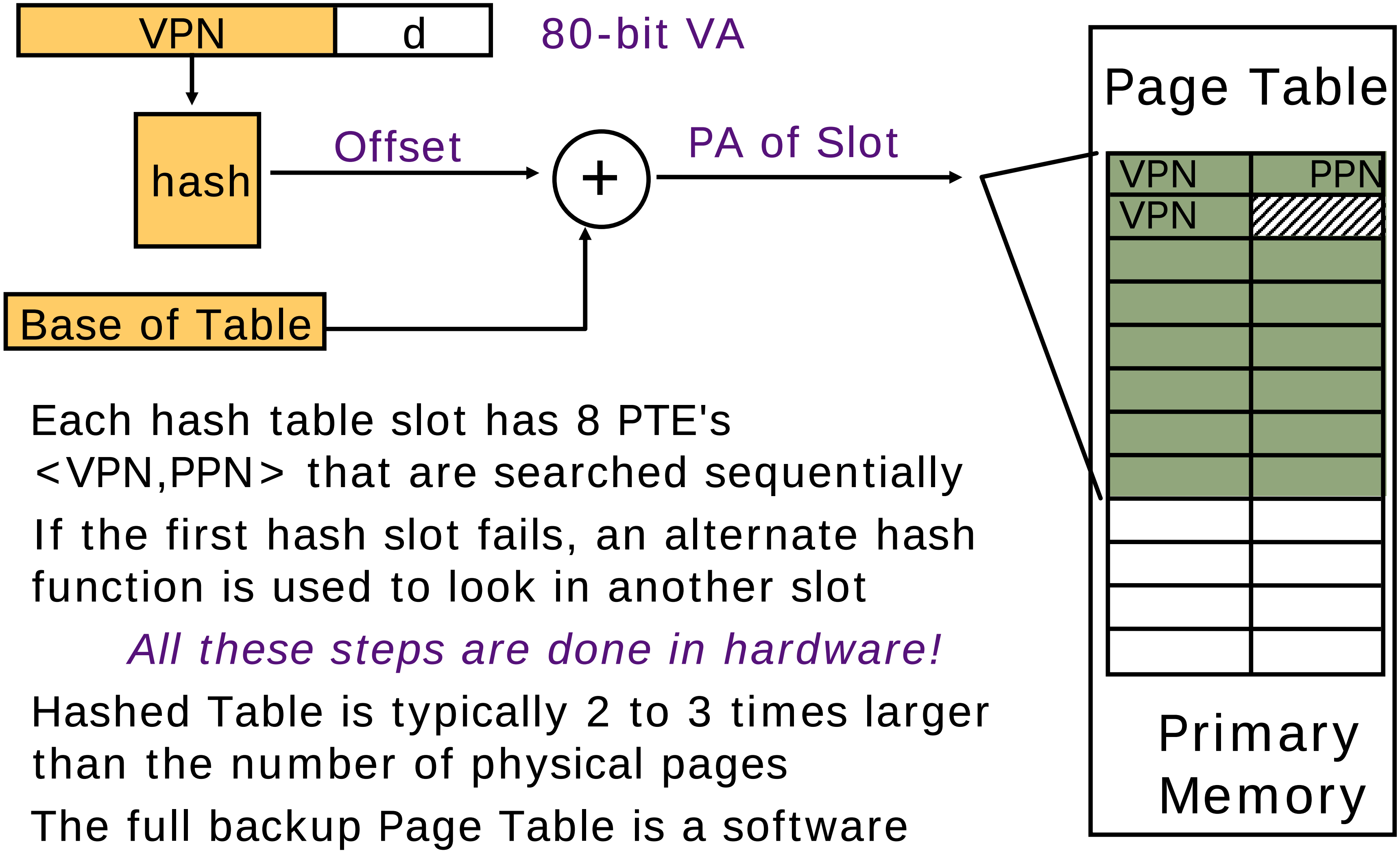
# Hashed Page Table Walk:

## PowerPC Two-level, Segmented Addressing



*[ IBM numbers bits with MSB=0 ]*

# Power PC: Hashed Page Table



- Each hash table slot has 8 PTE's <VPN,PPN> 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

# Virtual Memory Use Today - 1

- Desktops/servers 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 have translation and protection but not demand-paging
  (Crays: base&bound, Japanese: pages)
  - Don’t waste expensive CPU 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

*Given the software demands of modern embedded devices (e.g., cell phones, PDAs) all this may change in the near future!*

*Thank you !*