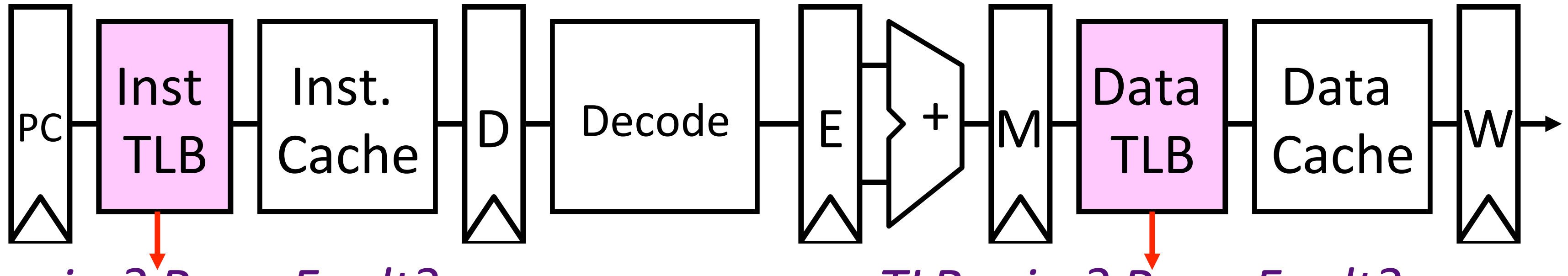
CS252 Graduate Computer Architecture
Fall 2015
Lecture 16: Virtual Memory and Caches

Krste Asanovic
`krste@eecs.berkeley.edu`
`http://inst.eecs.berkeley.edu/~cs252/fa15`

Address Translation in CPU Pipeline

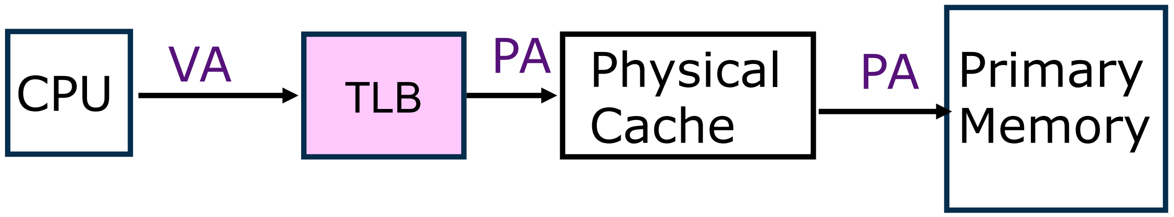


TLB miss? Page Fault? Protection violation?

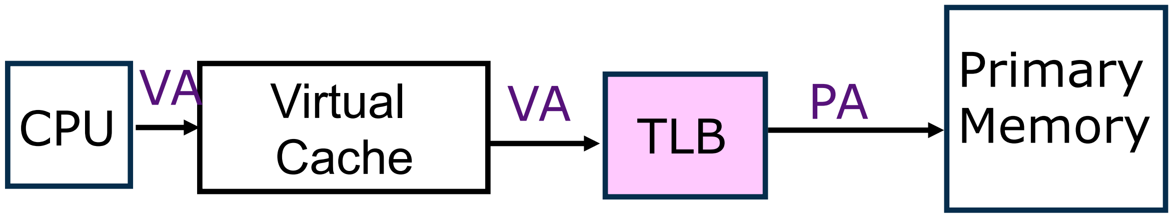
TLB miss? Page Fault? Protection violation?

- Need to cope with additional latency of TLB:
  - slow down the clock?
  - pipeline the TLB and cache access?
  - virtual address caches
  - parallel TLB/cache access

Virtual-Address Caches



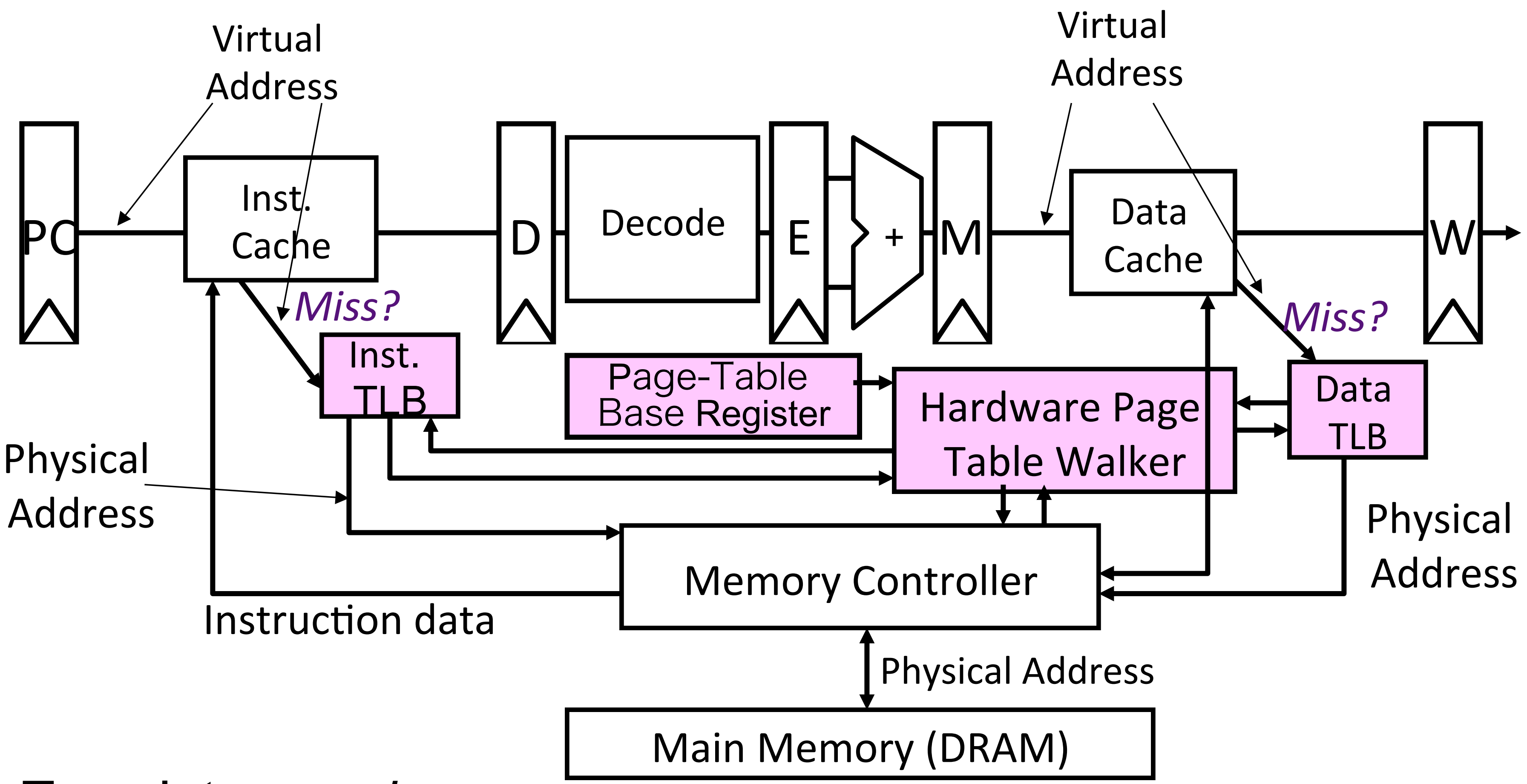
Alternative: place the cache before the TLB



(StrongARM)

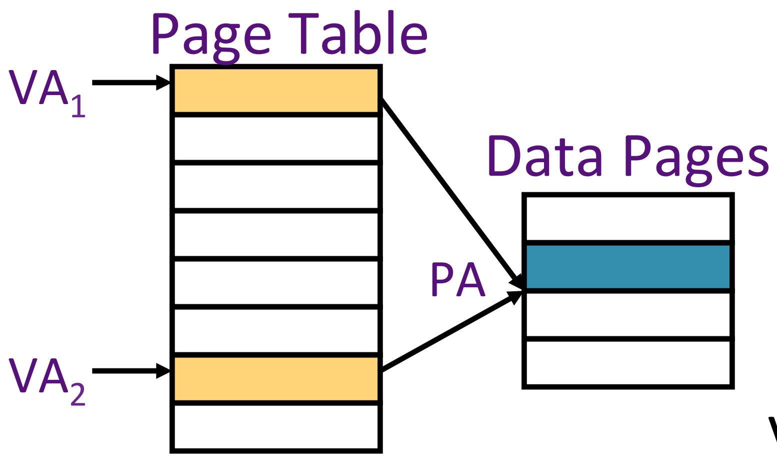
- one-step process in case of a hit (+)
- cache needs to be flushed on a context switch unless address space identifiers (ASIDs) included in tags (-)
- *aliasing problems* due to the sharing of pages (-)
- maintaining cache coherence (-)

Virtually Addressed Cache (Virtual Index/Virtual Tag)

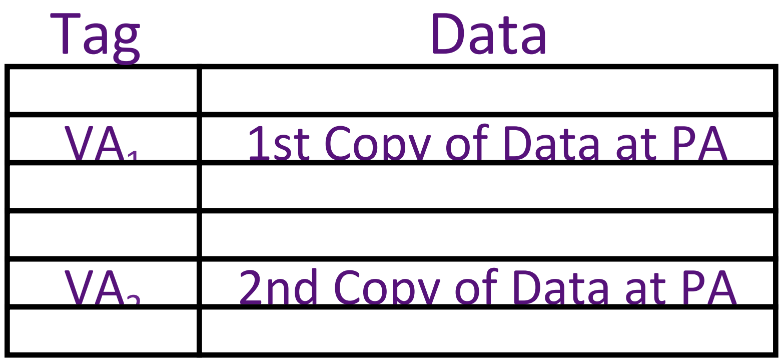


Translate on *miss*

Aliasing in Virtual-Address Caches



Two virtual pages share one physical page



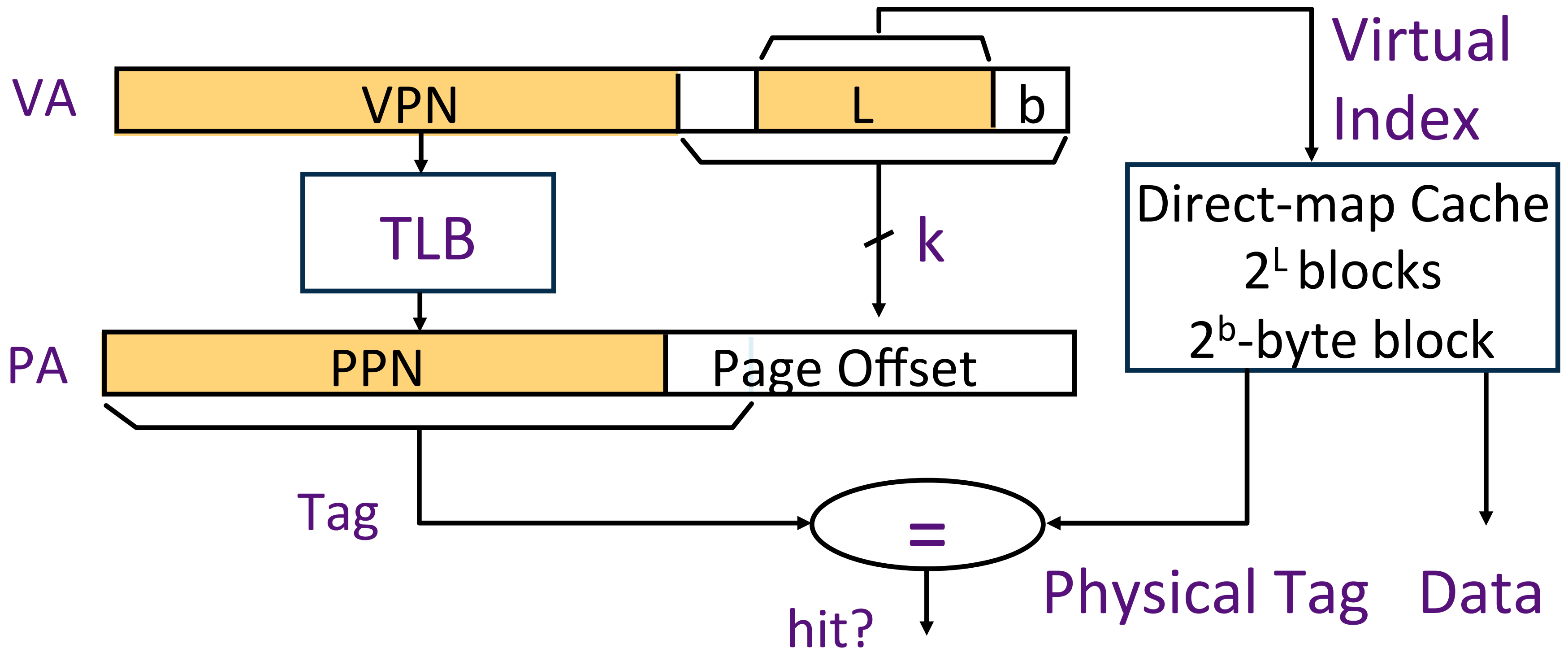
Virtual cache can have two copies of same physical data. Writes to one copy not visible to reads of other!

General Solution: *Prevent aliases coexisting 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 (Virtual Index/Physical Tag)



Index L is available without consulting the TLB

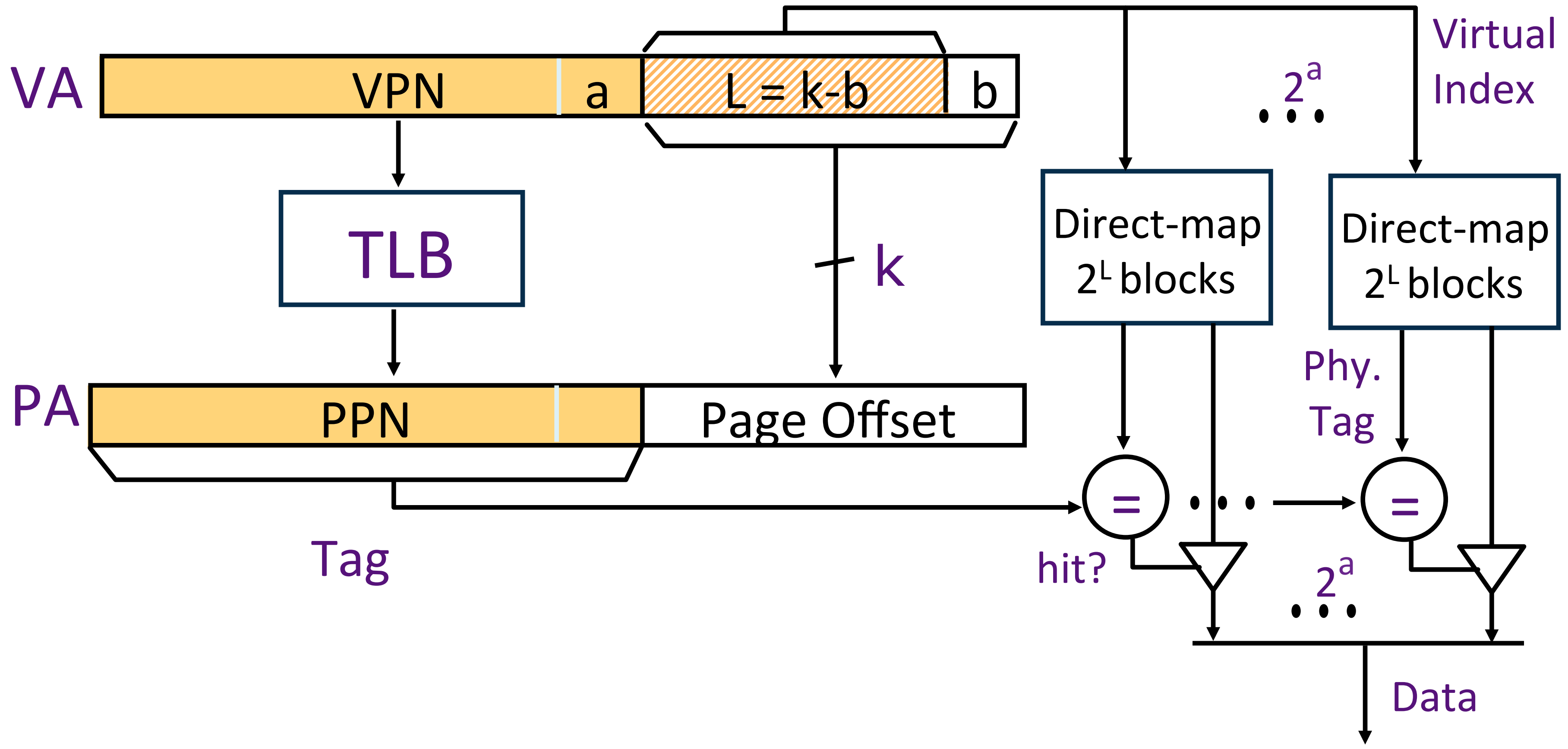
⇒ *cache and TLB accesses can begin simultaneously!*

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

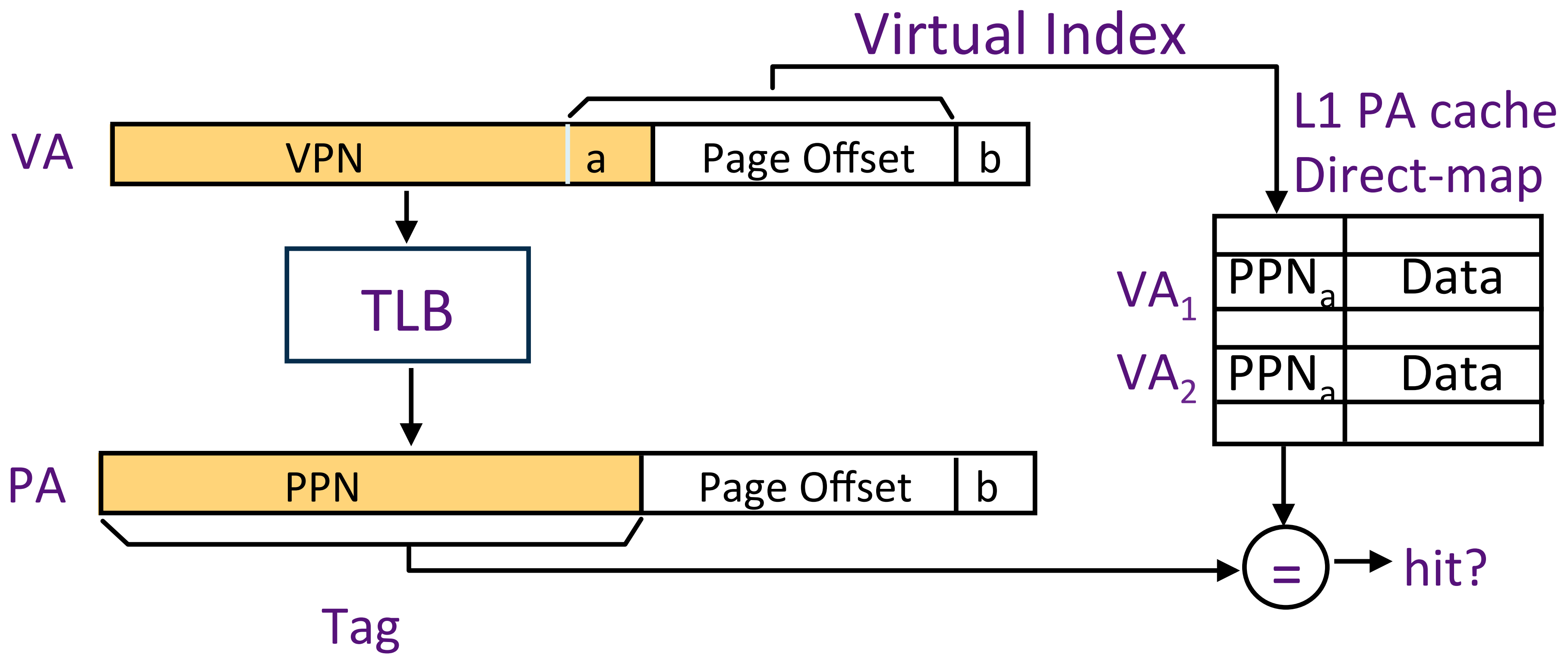


After the PPN is known, 2^a physical tags are compared

How does this scheme scale to larger caches?

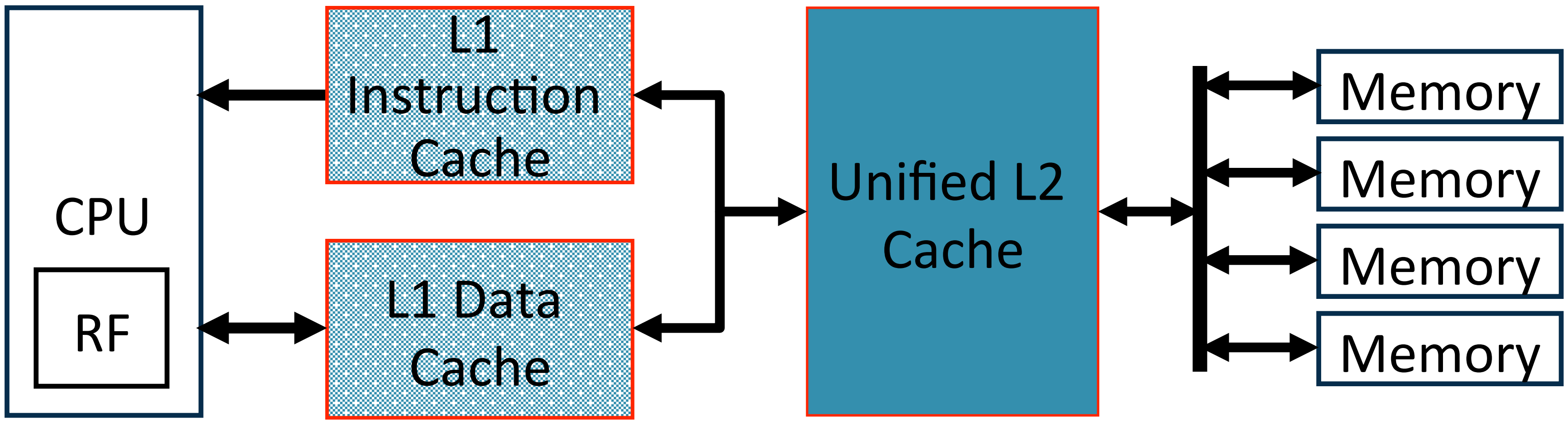
Concurrent Access to TLB & Large L1

The problem with L1 > Page size



Can VA_1 *and* VA_2 *both map to* PA *?*

A solution via Second Level Cache

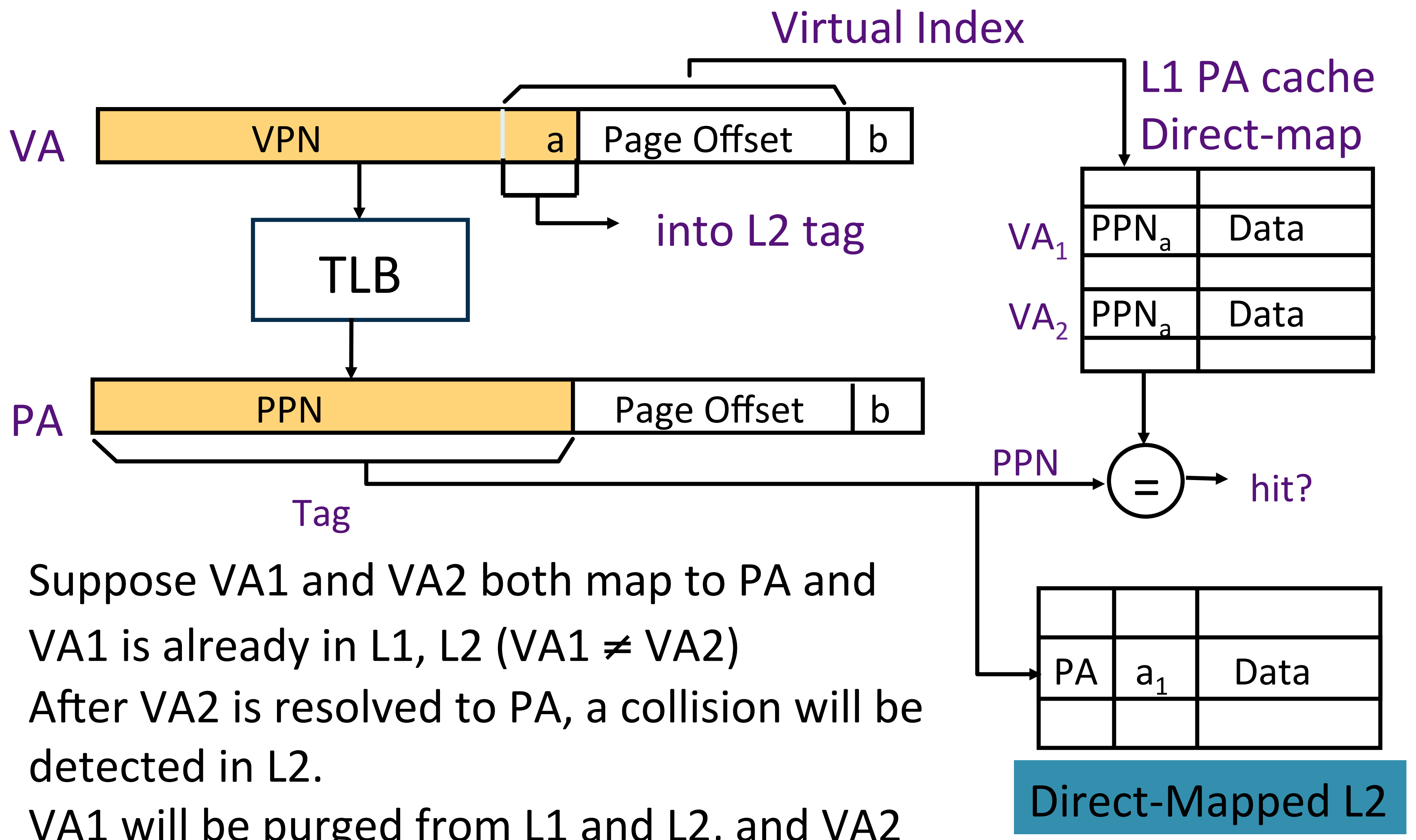


Usually a common L2 cache backs up both Instruction and Data L1 caches

L2 is “inclusive” of both Instruction and Data caches

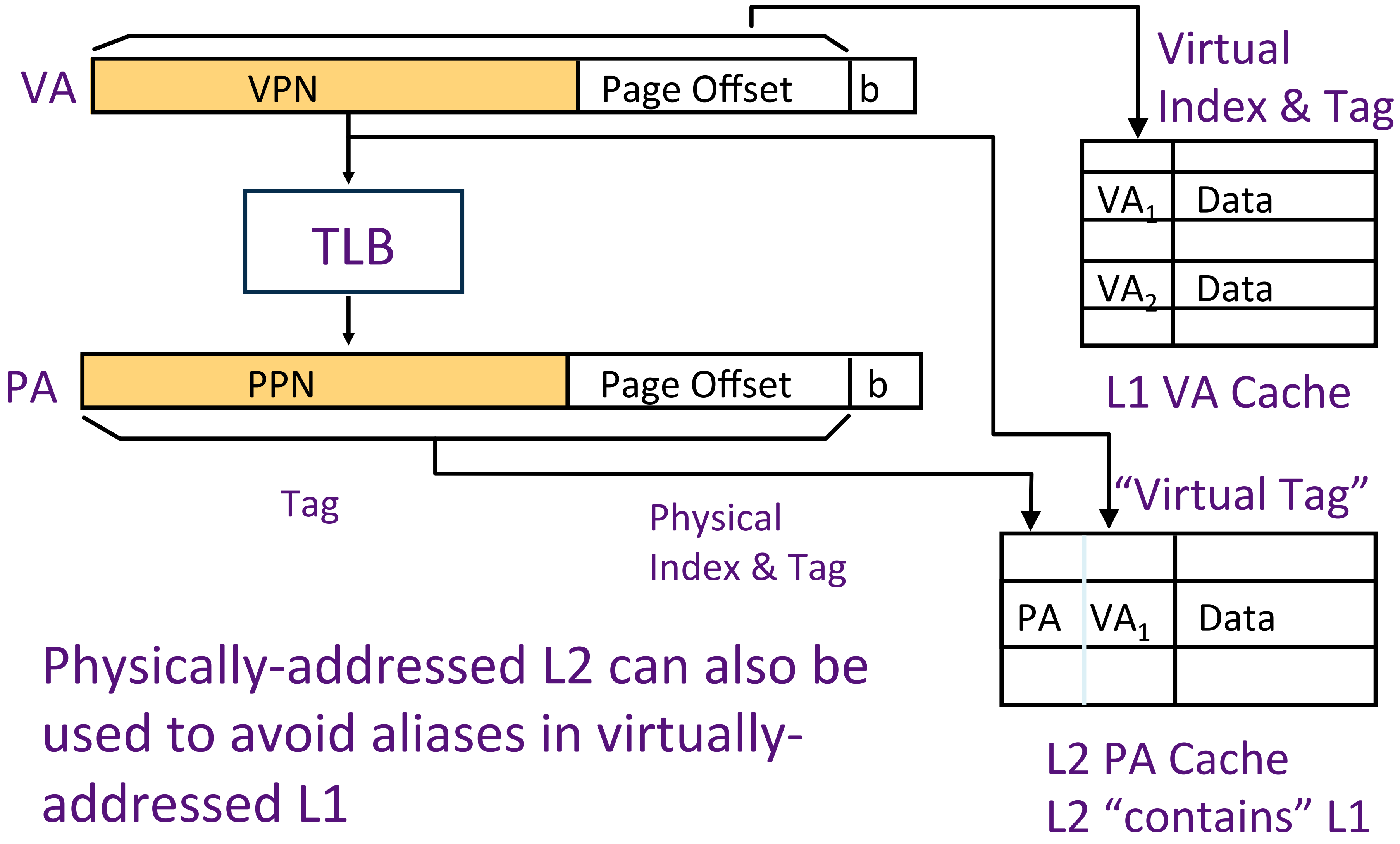
- Inclusive means L2 has copy of any line in either L1

Anti-Aliasing Using L2 [*MIPS R10000,1996*]



- 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 !*

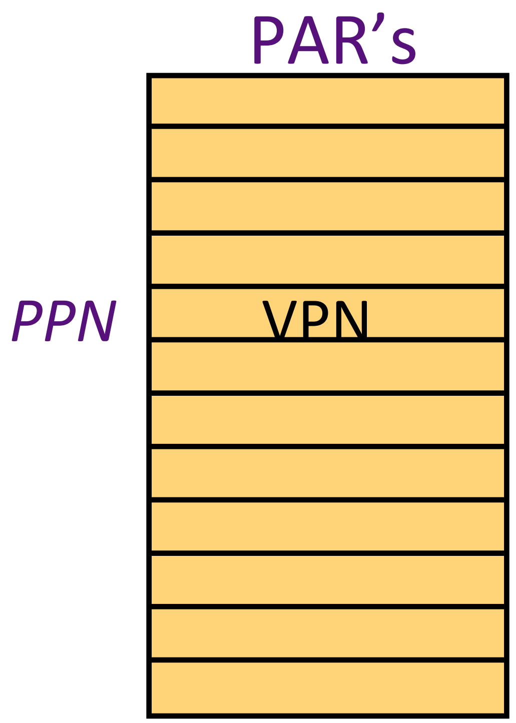
Anti-Aliasing using L2 for a Virtually Addressed L1



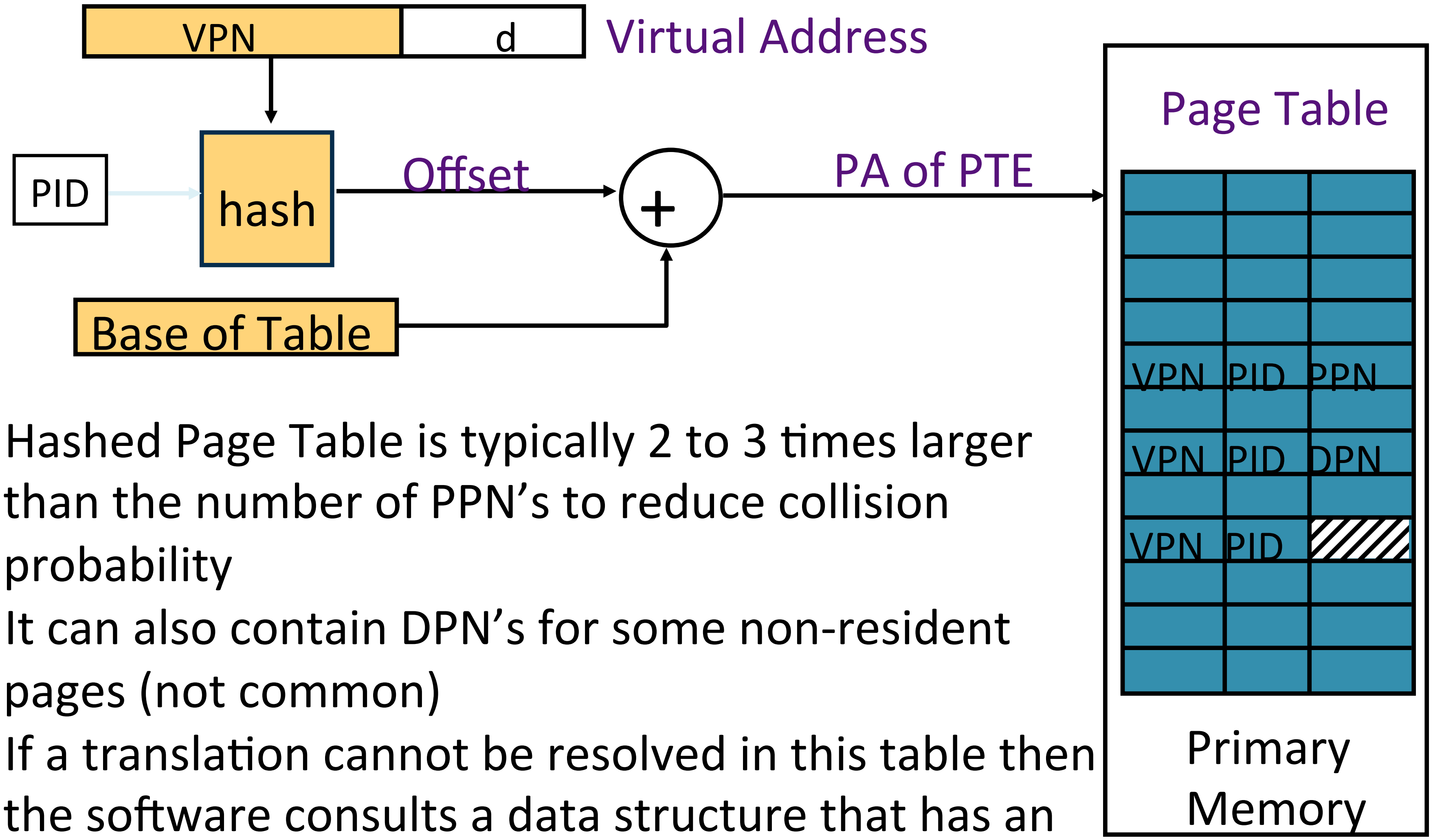
Physically-addressed L2 can also be used to avoid aliases in virtually-addressed L1

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 ?*

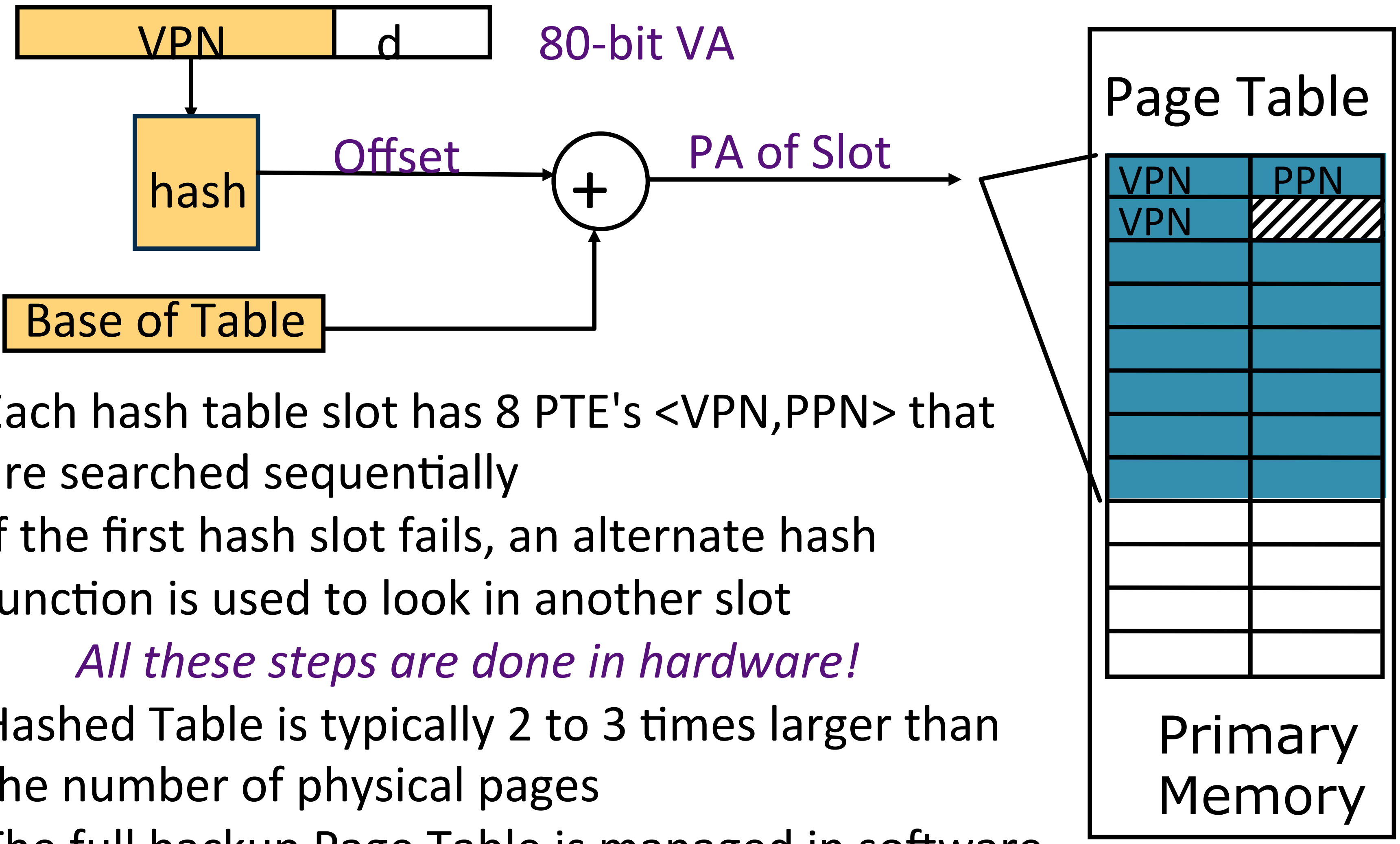


Hashed Page Table: Approximating Associative Addressing



- 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 (e.g., full page table)

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 managed in software

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 (supports *swapping* entire programs but not demand-paged 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
  - Idea from 1970s IBM mainframes, now common on laptops
    - e.g., run Windows on top of Mac OS X
  - Hardware support for two levels of translation/protection
    - Guest OS virtual -> Guest OS physical -> Host machine physical

Virtual Memory Use Today - 1

- Servers/desktops/laptops/smartphones 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 rarely complete demand-paging
- (Older Crays: base&bound, Japanese & Cray X1/X2: 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

Acknowledgements

- This course is partly inspired by previous MIT 6.823 and Berkeley CS252 computer architecture courses created by my collaborators and colleagues:
  - Arvind (MIT)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
  - David Patterson (UCB)