http://inst.eecs.berkeley.edu/~cs152

CS 152/252A Computer Architecture and Engineering

Sophia Shao

Lecture 10 – Virtual Memory

Ep 11: Future of AI Computing and How to Build & Nurture Hardware Teams with Jim Keller, Tenstorrent

2 days ago

Jim Keller is the CTO of Tenstorrent, and a veteran computer architect. Prior to Tenstorrent, he has held roles of Senic Vice President of Autopilot at Tesla, Vice President and Chief Architect at AMD, and at PA Semi which was acquired b successful silicon designs over the decades, from the DEC Alpha processors, to AMD K7/K8/K12, HyperTransport and Apple A4/A5 processors, and Telsa's self-driving car chip.

https://comparchpodcast.podbean.com/e/episode-11-future-of-ai-computing-and-how-to-build-nurture-hardware-teams%c2%a0with-jim-keller-tenstorrent/

Jim Keller: Work Experience

AnandTech		Company	Title	Important Product
1980s	1998	DEC	Architect	Alpha
1998	1999	AMD	Lead Architect	K7, K8v1 HyperTransport
1999	2000	SiByte	Chief Architect	MIPS Networking
2000	2004	Broadcom	Chief Architect	MIPS Networking
2004	2008	P.A. Semi	VP Engineering	Low Power Mobile
2008	2012	Apple	VP Engineering	A4 / A5 Mobile
8/2012	9/2015	AMD	Corp VP and Chief Cores Architect	Skybridge / K12 (+ Zen)
1/2016	4/2018	Tesla	VP Autopilot Hardware Engineering	Fully Self-Driving (FSD) Chip
4/2018	6/2020	Intel	Senior VP Silicon Engineering	?
2021		Tenstorrent	President and CTO	TBD

https://www.anandtech.com/show/16762/an-anandtech-interview-with-jim-keller-laziest-person-at-tesla

Last time in Lecture 8

- Protection and translation required for multiprogramming
  - Base and bounds was early simple scheme
- Page-based translation and protection avoids need for memory compaction, easy allocation by OS
  - But need to indirect in large page table on every access
- Address spaces accessed sparsely
  - Can use multi-level page table to hold translation/protection information, but implies multiple memory accesses per reference
- Address space access with locality
  - Can use “translation lookaside buffer” (TLB) to cache address translations (sometimes known as address translation cache)
  - Still have to walk page tables on TLB miss, can be hardware or software talk
- Virtual memory uses DRAM as a “cache” of disk memory, allows very cheap main memory

Modern Virtual Memory Systems

Illusion of a large, private, uniform store

Protection & Privacy

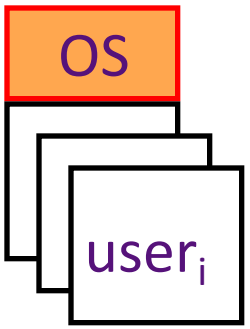
several users, each with their private address space and one or more shared address spaces

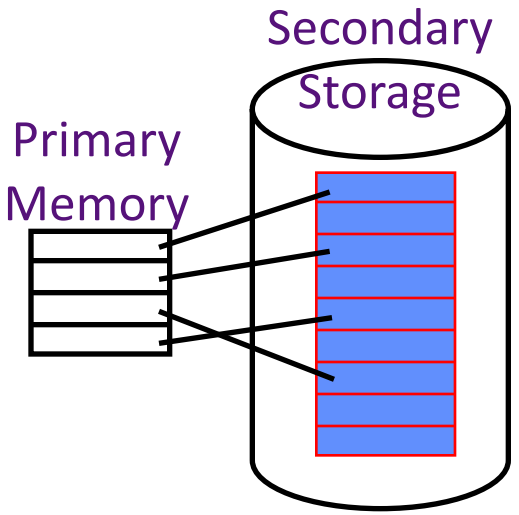
Demand Paging

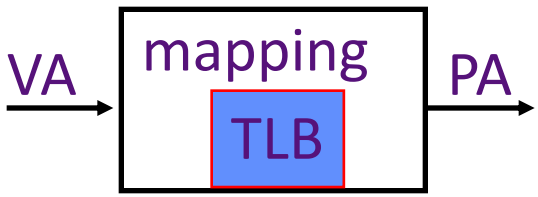
Provides the ability to run programs larger than the primary memory

Hides differences in machine configurations

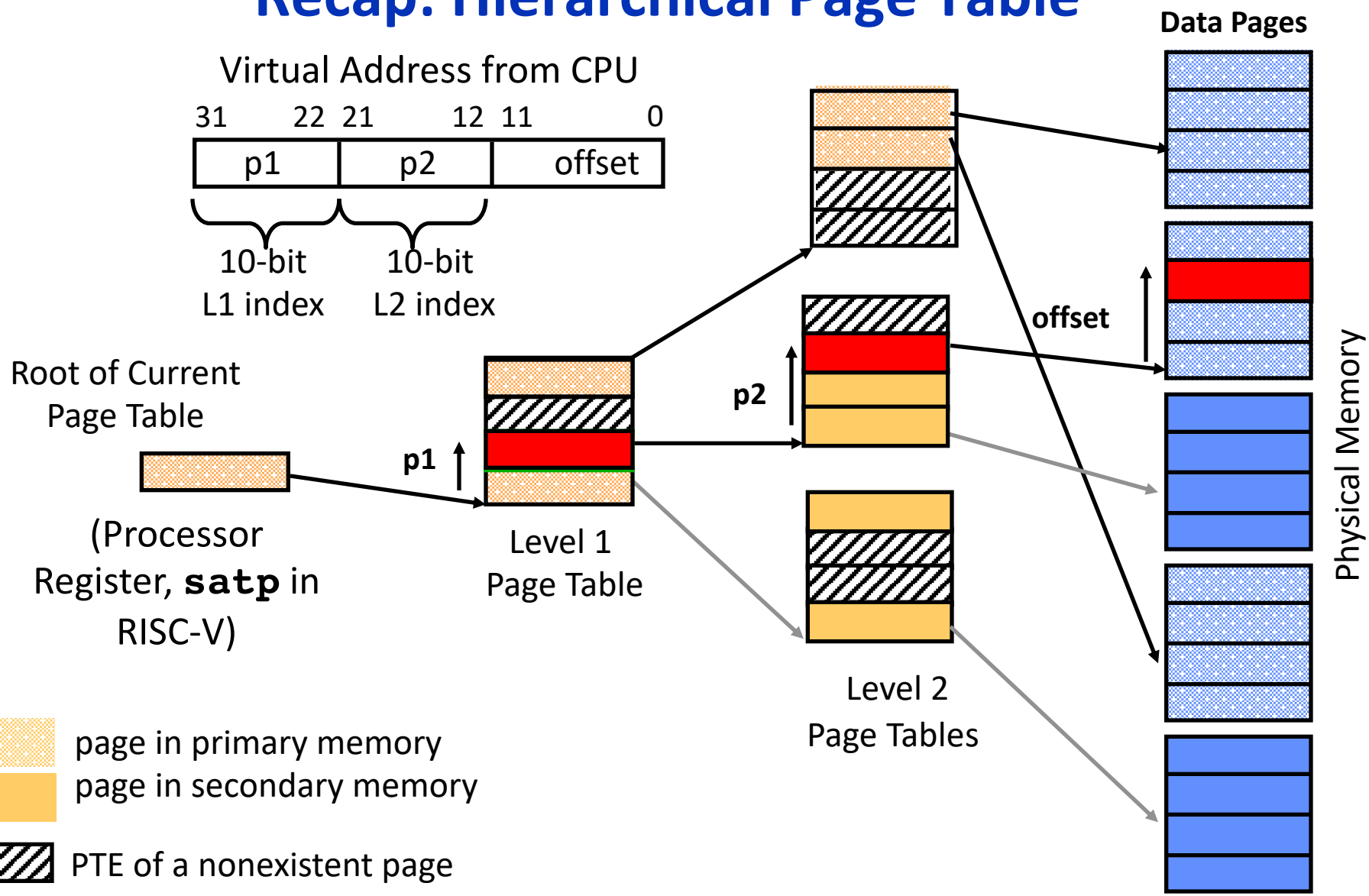
The price is address translation on each memory reference





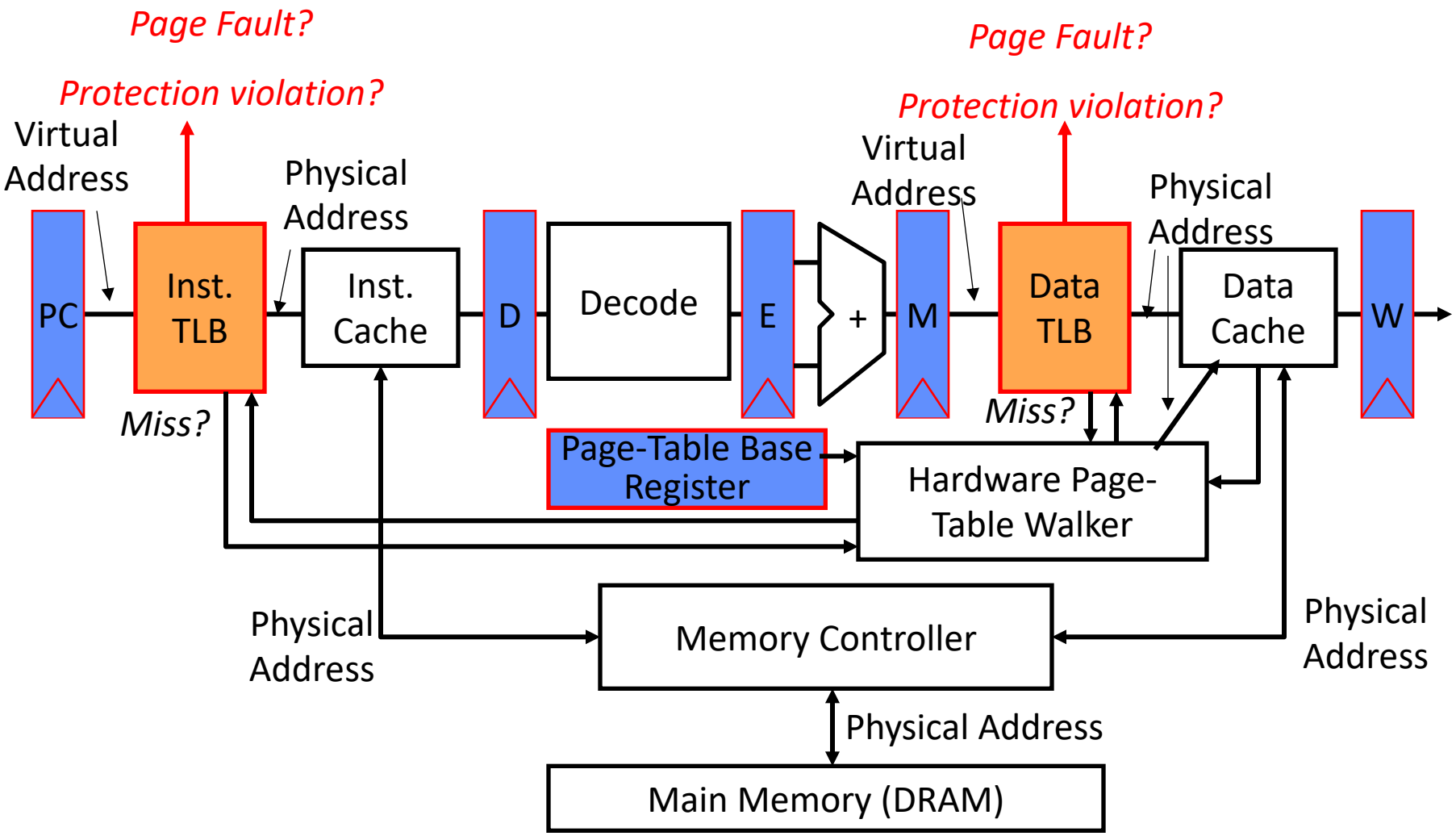


Recap: Hierarchical Page Table

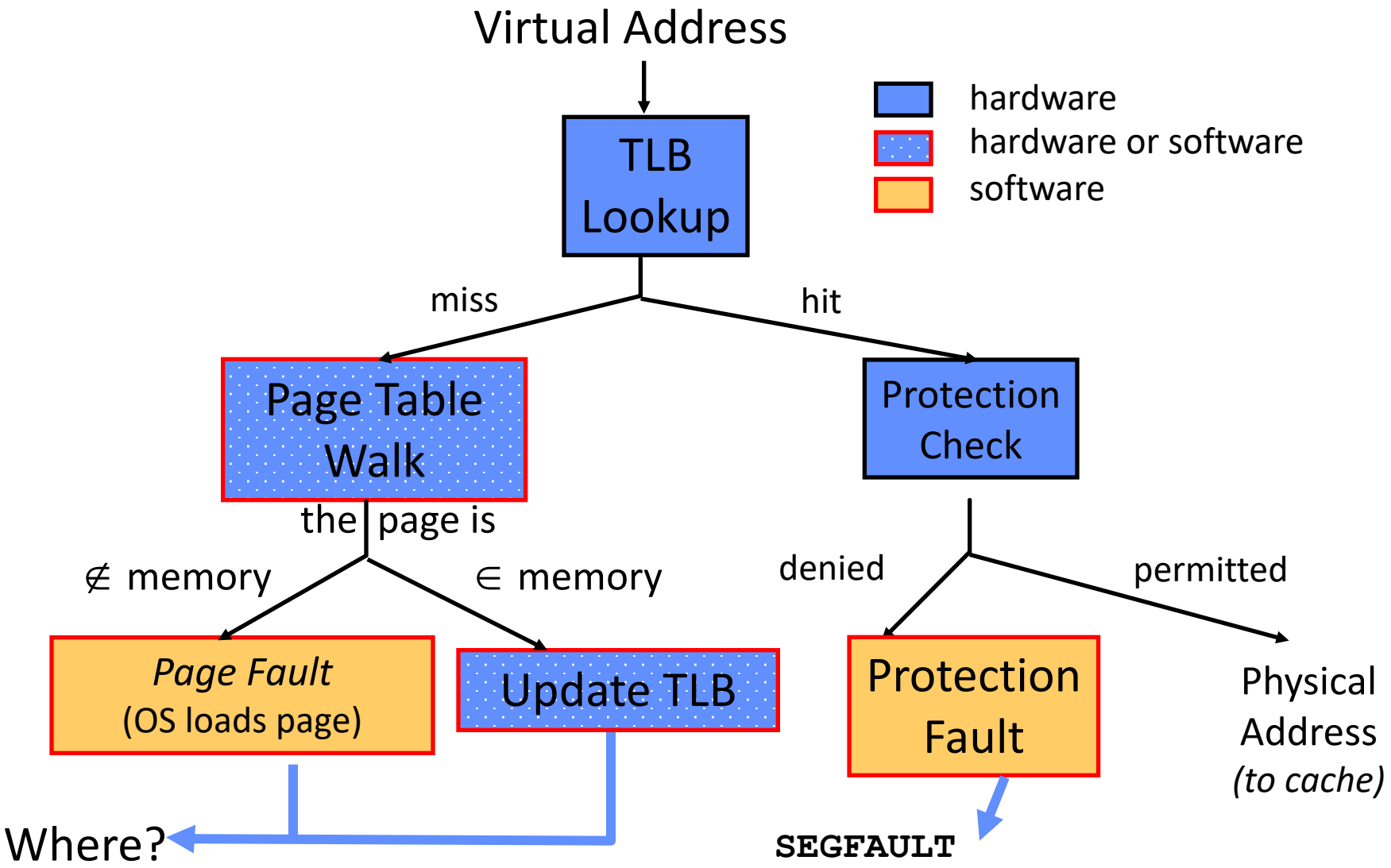


RISC-V Sv32 Virtual Memory Scheme

Recap: Page-Based Virtual-Memory Machine
(Hardware Page-Table Walk)



Address Translation:
putting it all together



Tagged TLB in MIPS R4000

- When switched to a new process, entries in TLB become invalid
  - TLB needs to be flushed and repopulated with entries for the new process.
- To reduce this overhead, we allow entries from multiple processes to be stored in TLB
  - Tagged TLB to include Address Space ID (ASID)
  - Reduce TLB flushing when switching contexts

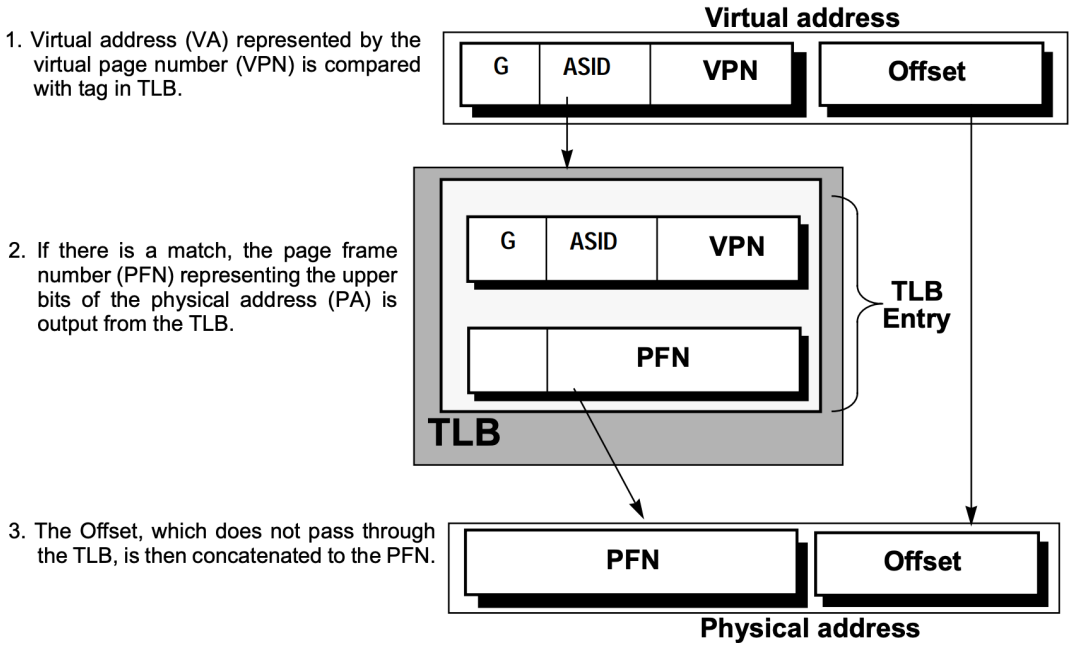
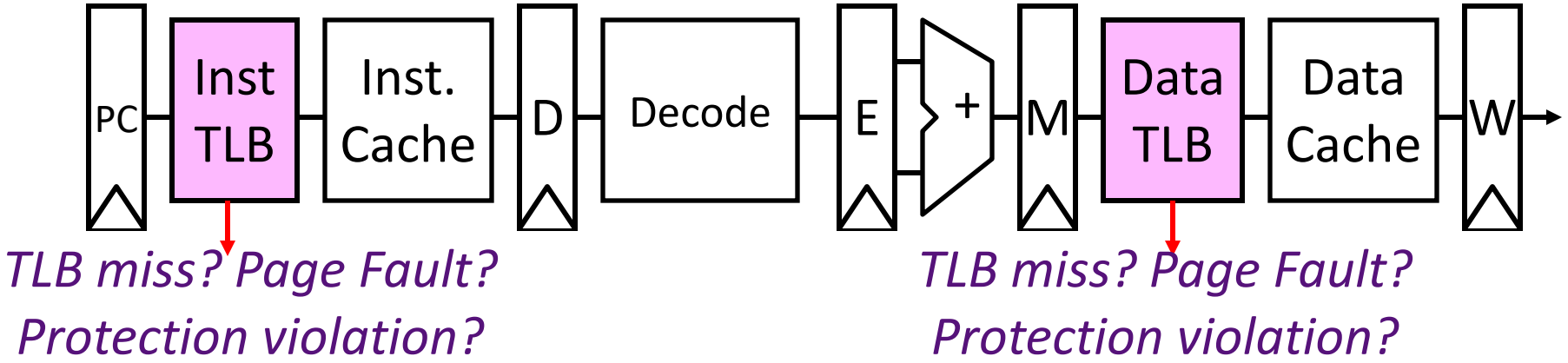


Figure 4-1 Overview of a Virtual-to-Physical Address Translation

Page-Fault Handler

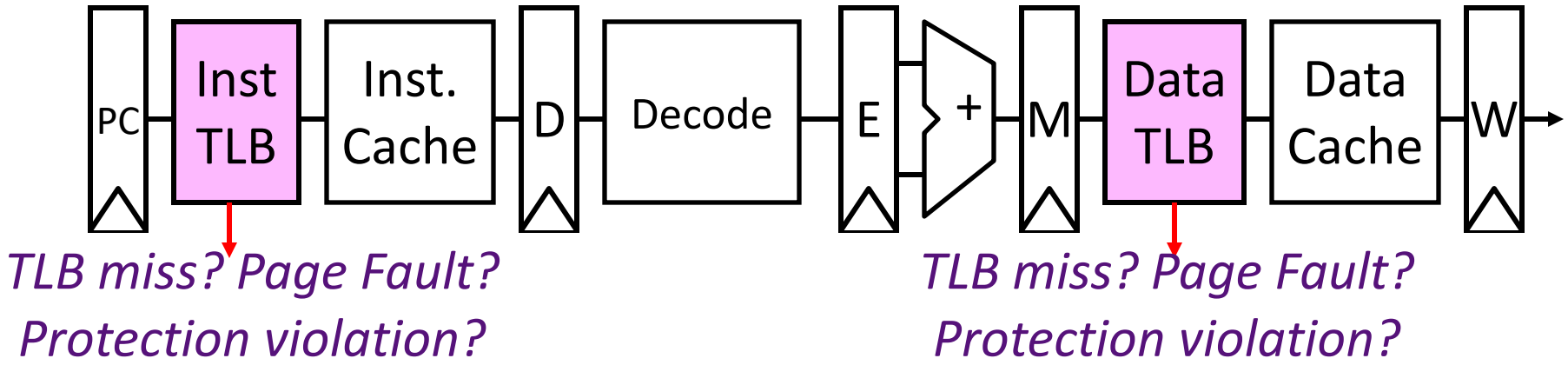
- When the referenced page is not in DRAM:
  - The missing page is located (or created)
  - It is brought in from disk, and page table is updated
    - Another user job may run on CPU while first job waits for the requested page to be read from disk, provided system allows architectural context to be saved and restored
  - If no free pages are left, a page is swapped out
    - Pseudo-LRU replacement policy, implemented in software
    - A set of free pages can be maintained by OS as background activity
- Since it takes a long time to transfer a page (msecs), page faults are handled completely in software by the OS

Handling VM-related exceptions



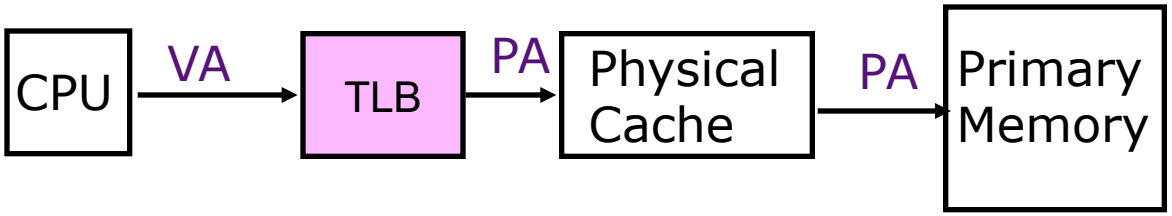
- Handling a TLB miss needs a hardware or software mechanism to refill TLB
- Handling page fault (e.g., page is on disk) needs *restartable* exception so software handler can resume after retrieving page
  - Can be imprecise. but restartable, but this complicates OS software
  - Precise exceptions are easy to restart
- A protection violation may abort process
  - But often handled the same as a page fault

Address Translation in CPU Pipeline

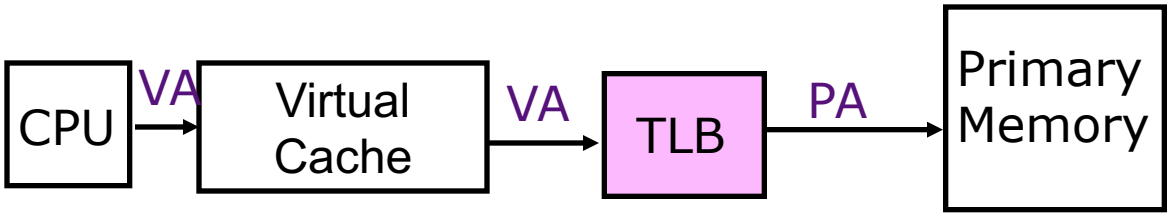


- Need to cope with additional latency of TLB:
  - slow down the clock?
  - pipeline the TLB and cache access?
  - virtual address caches
  - parallel TLB/cache access

Virtual-Address Caches



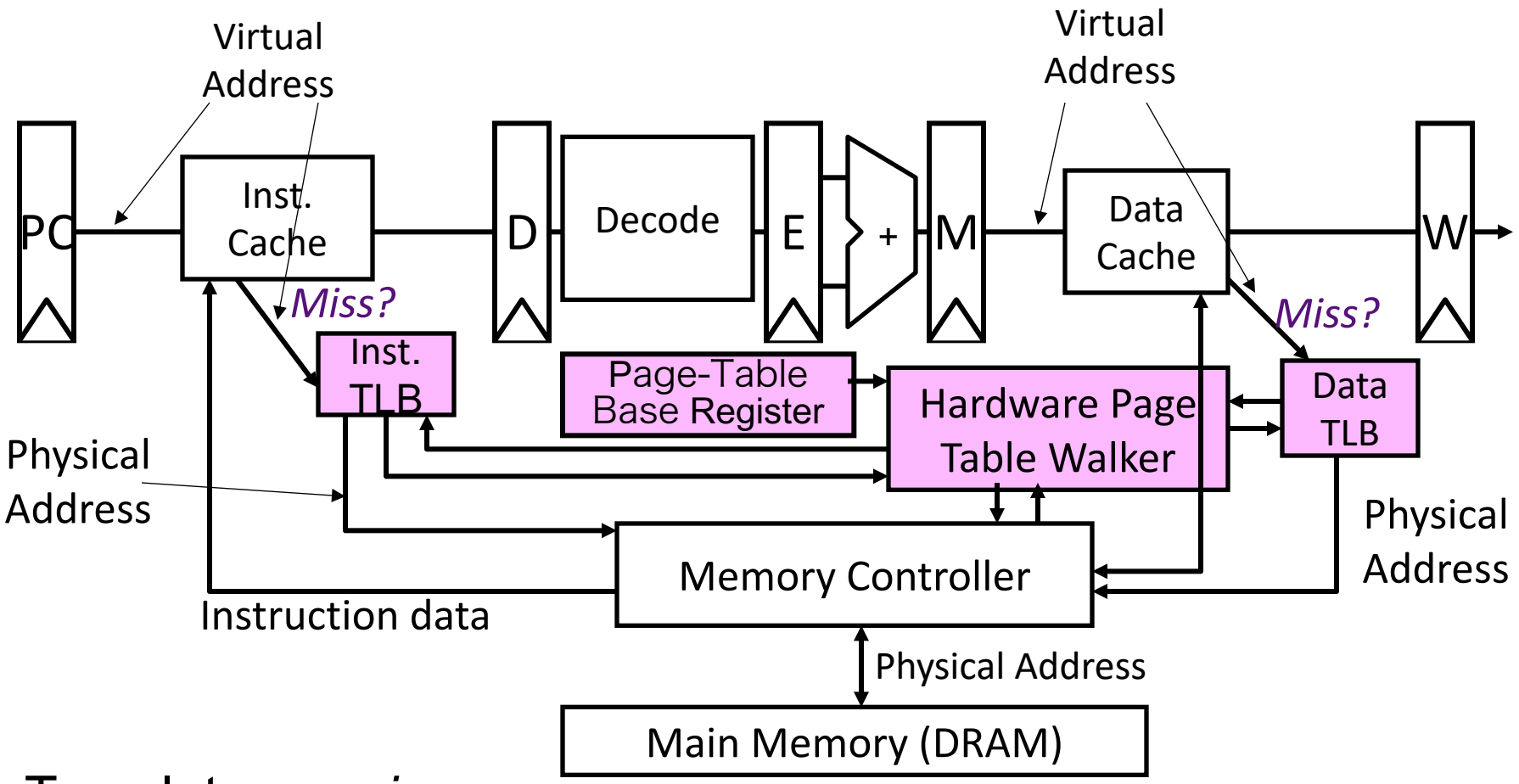
Alternative: place the cache before the TLB



(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*

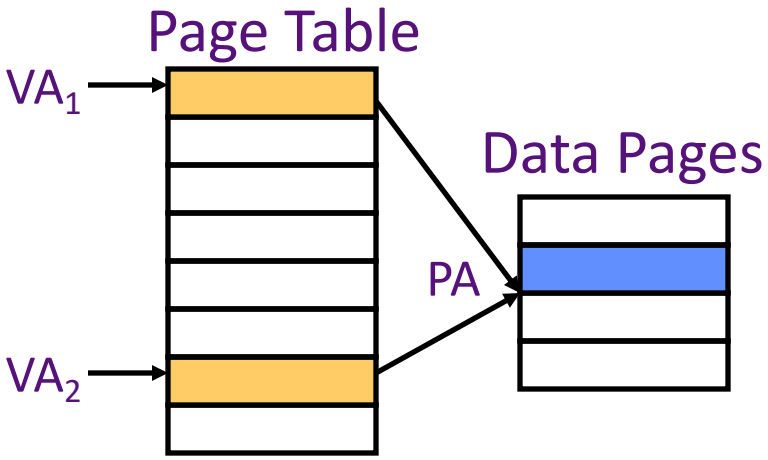
CS152 Administrivia

- HW2 out.
  - Due 2/21
- HW1 graded.
- Lab 2 out this week
  - Due 3/02
- Midterm 1 on Tuesday Feb 28
  - 7-9pm
  - 155 Dwinelle
  - Cover Lectures 1-10 (this week).

CS252 Administrivia

- Project Proposal due Wednesday 02/22.
- Submit through Gradescope.
- Give ~5-minute presentations in class in discussion section time on Wednesday 03/01 and 03/08
- Next week paper reading:
  - Cache and virtual memory

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

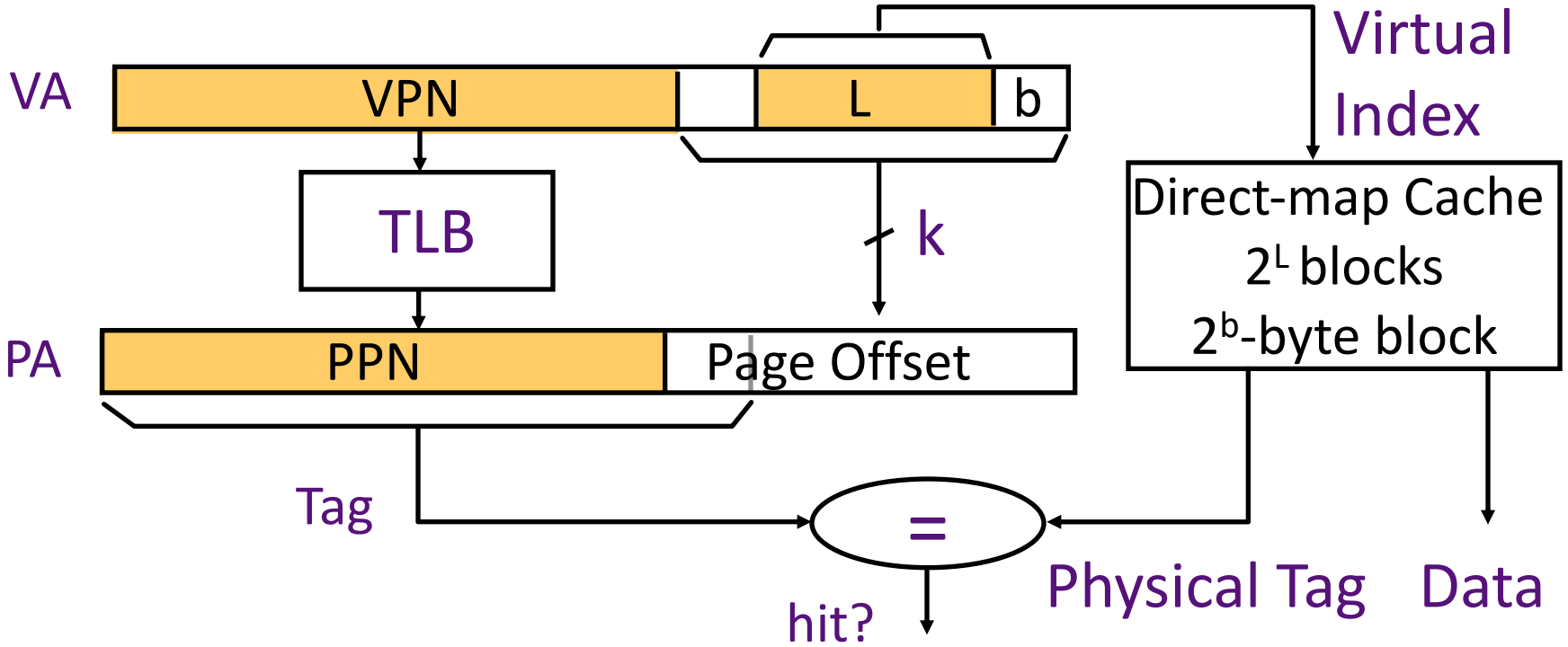
Virtual cache can have two copies of same physical data. Writes to one copy not visible to reads of other!

General Solution: *Prevent aliases coexisting in cache*

One solution: use 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!*

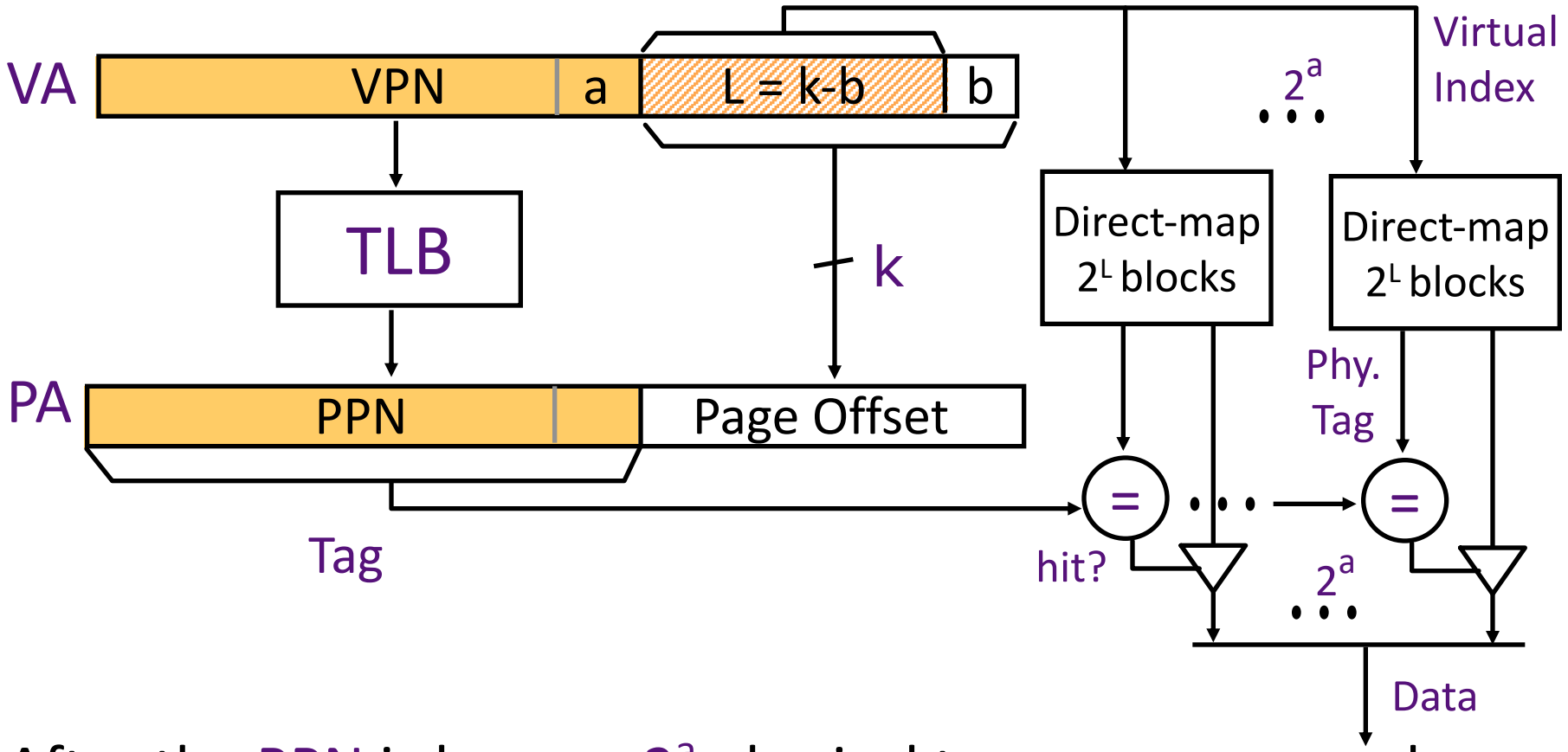
Tag comparison is made after both accesses are completed

Cases: only works when L + b <= k

- *cache can only be as big as page size!*

Virtual-Index Physical-Tag Caches:

Use associativity to further increase cache capacity

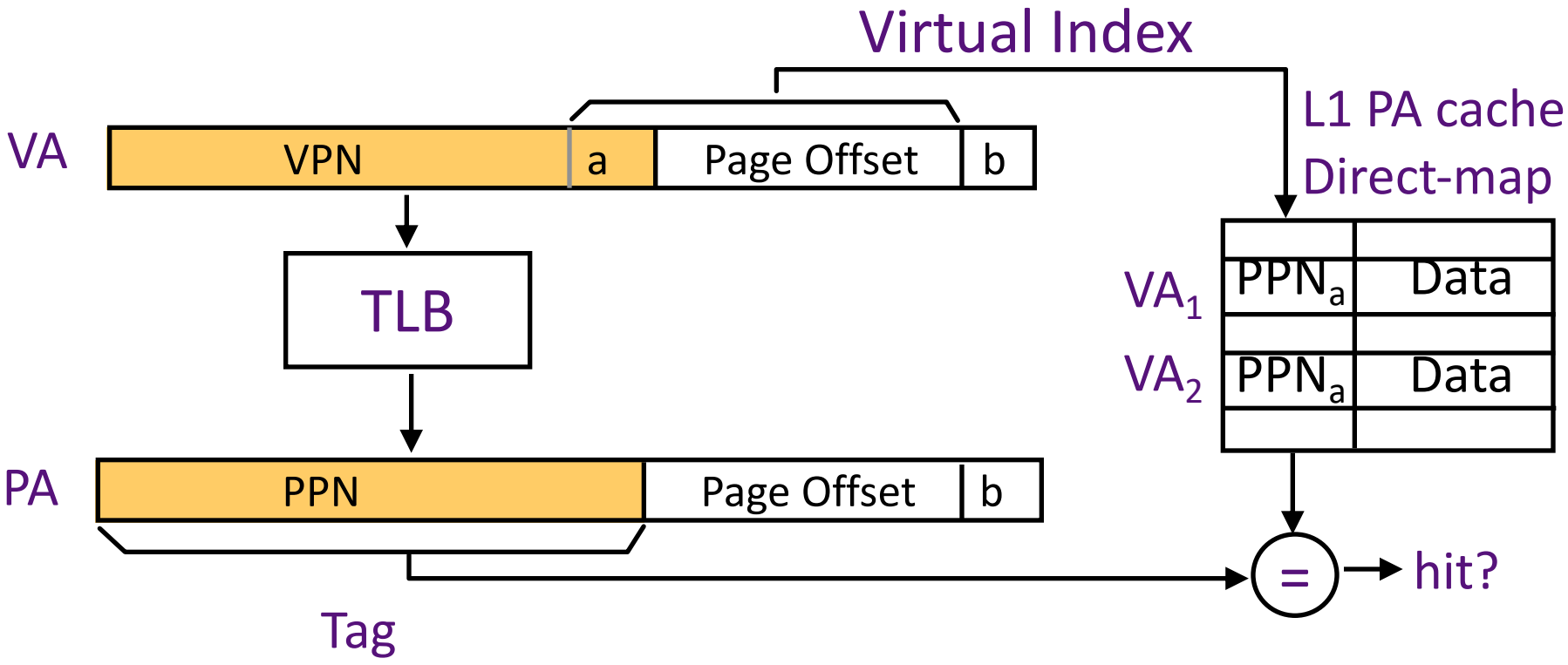


After the PPN is known, 2^a physical tags are compared

*4KB page size * 8-way (2^3) associative = 32KB cache*

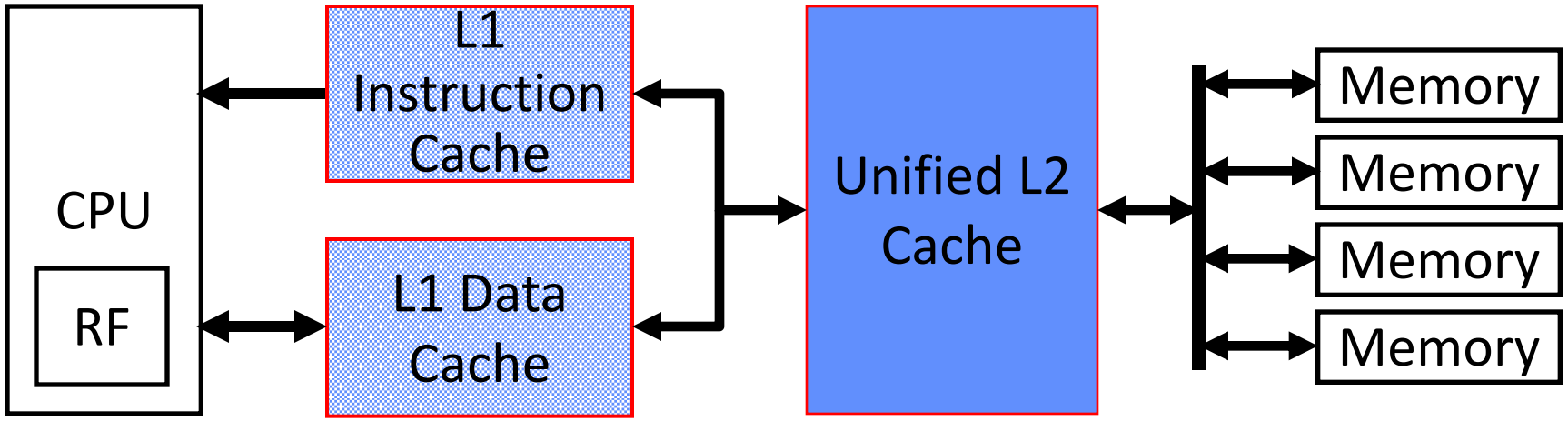
Concurrent Access to TLB & Large L1

The problem with L1 > Page size



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

A solution via Second-Level Cache

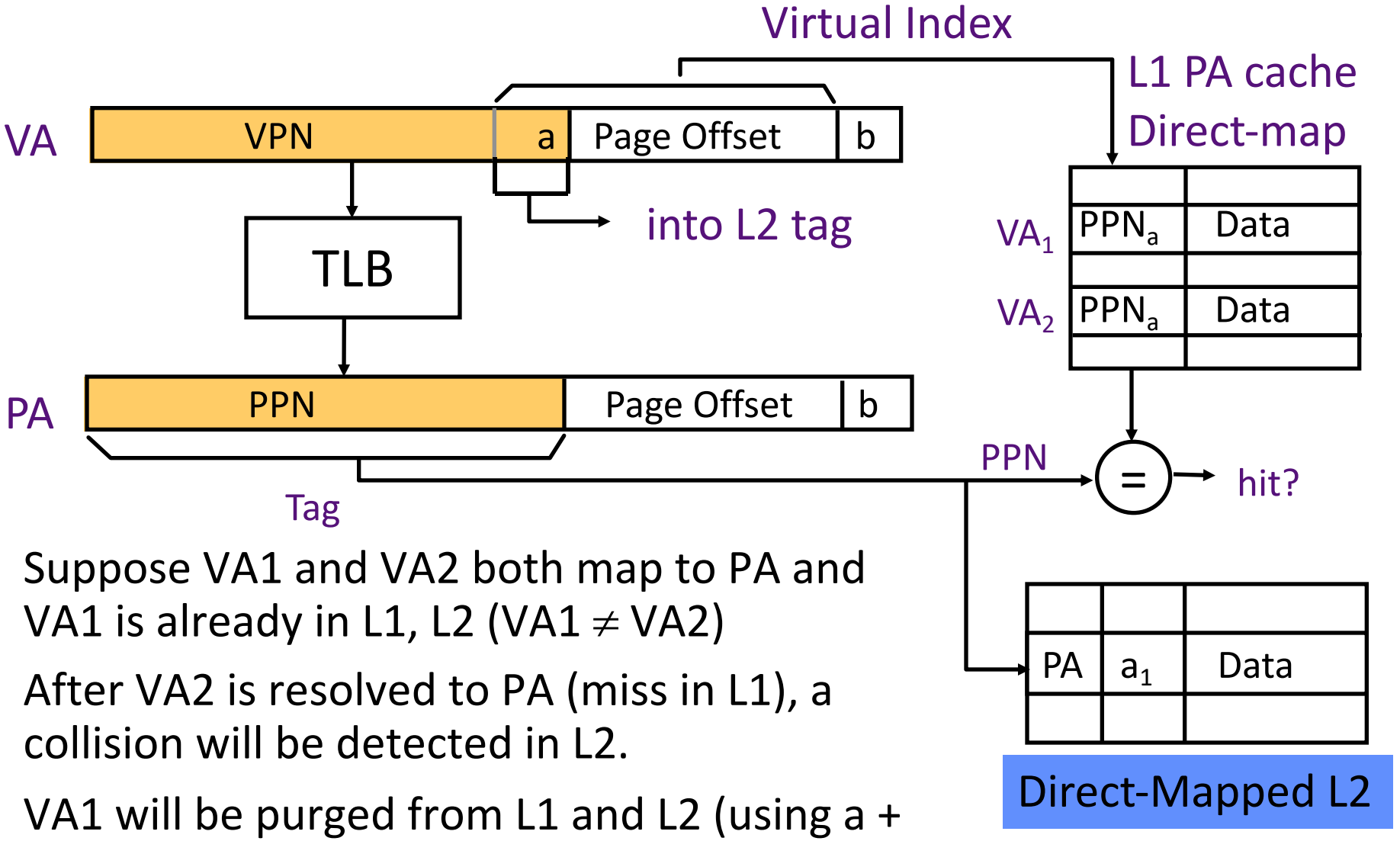


Usually a common L2 cache backs up both Instruction and Data L1 caches

L2 is “inclusive” of both Instruction and Data caches

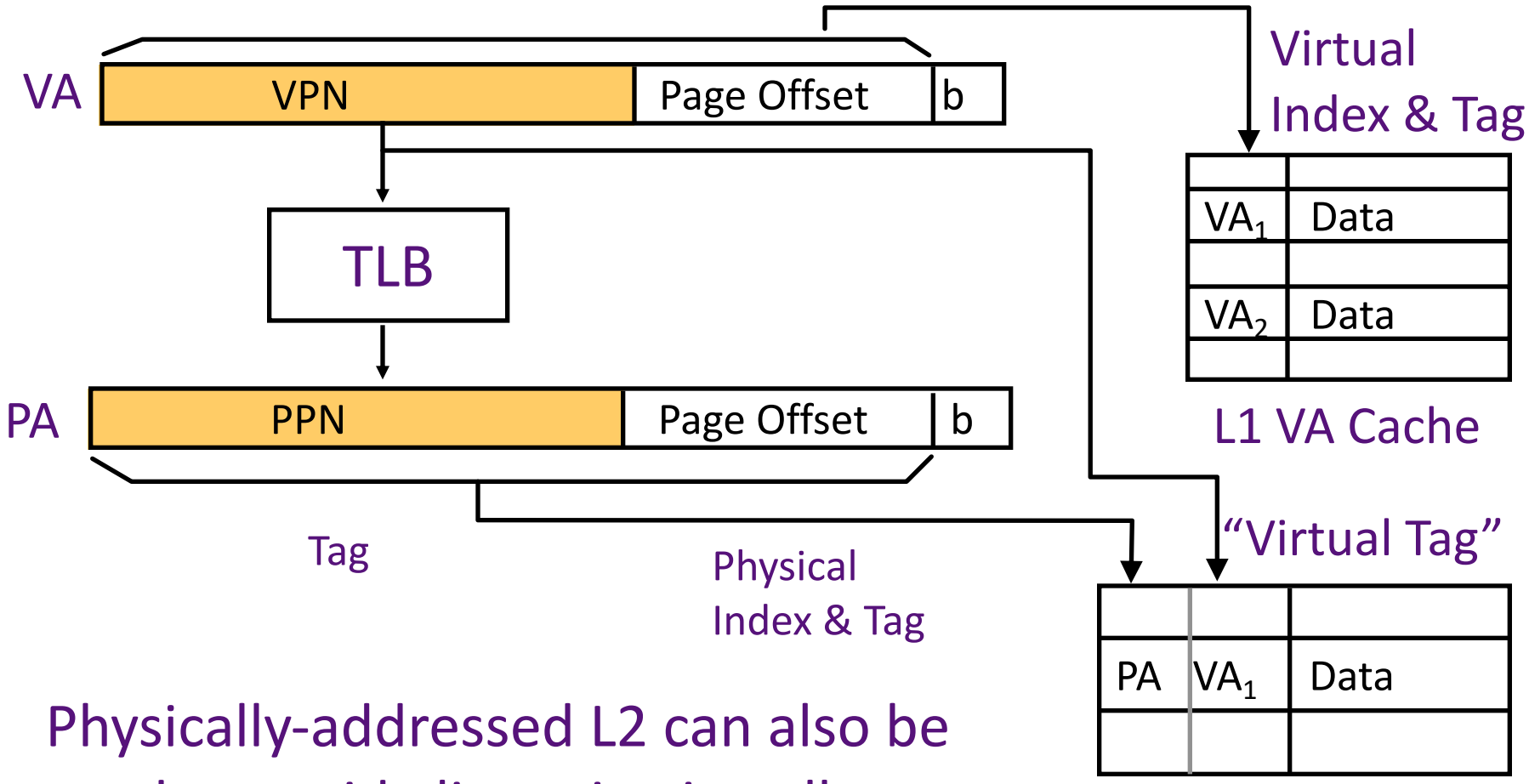
- Inclusive means L2 has copy of any line in either L1

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



- Suppose VA1 and VA2 both map to PA and VA1 is already in L1, L2 (VA1 ≠ VA2)
- After VA2 is resolved to PA (miss in L1), a collision will be detected in L2.
- VA1 will be purged from L1 and L2 (using a + page offset), and VA2 will be loaded ⇒ *no aliasing !*

Anti-Aliasing using L2 for a Virtually Tagged L1

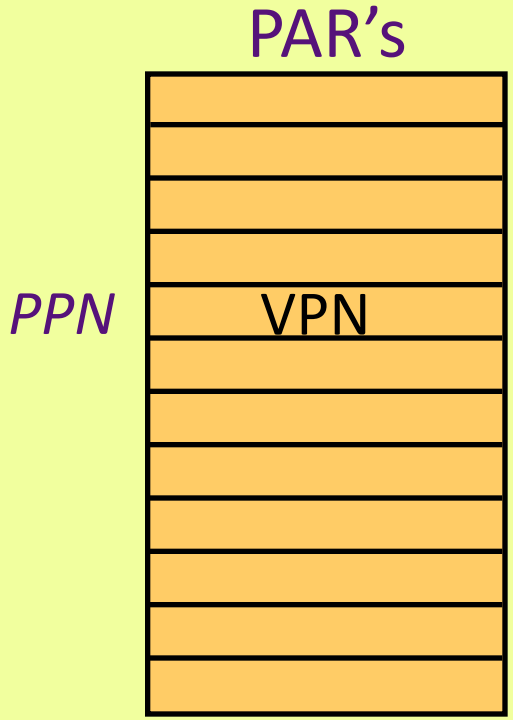


Physically-addressed L2 can also be used to avoid aliases in virtually-addressed L1

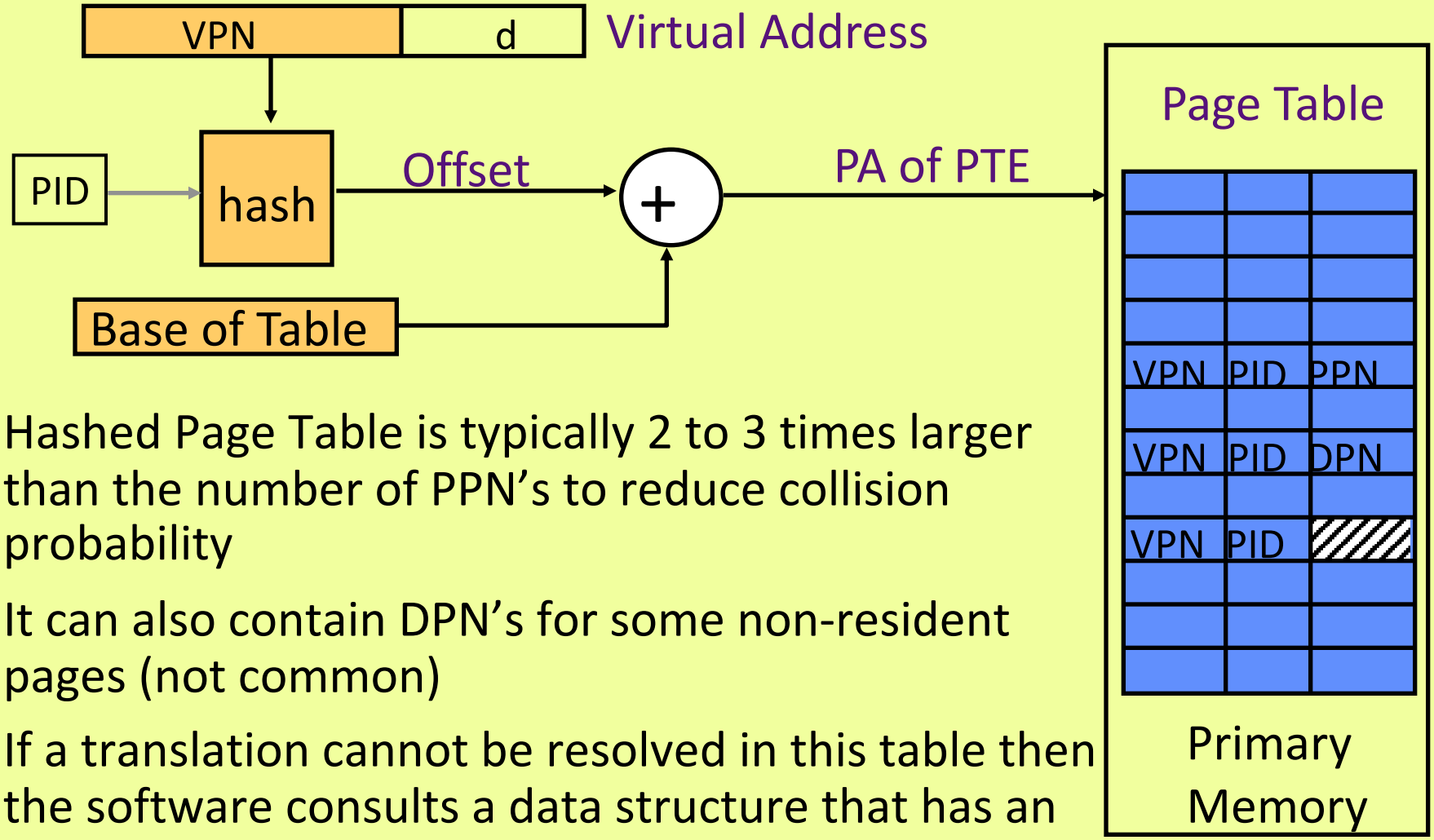
- "virtual tag" will be much bigger.

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 ?*

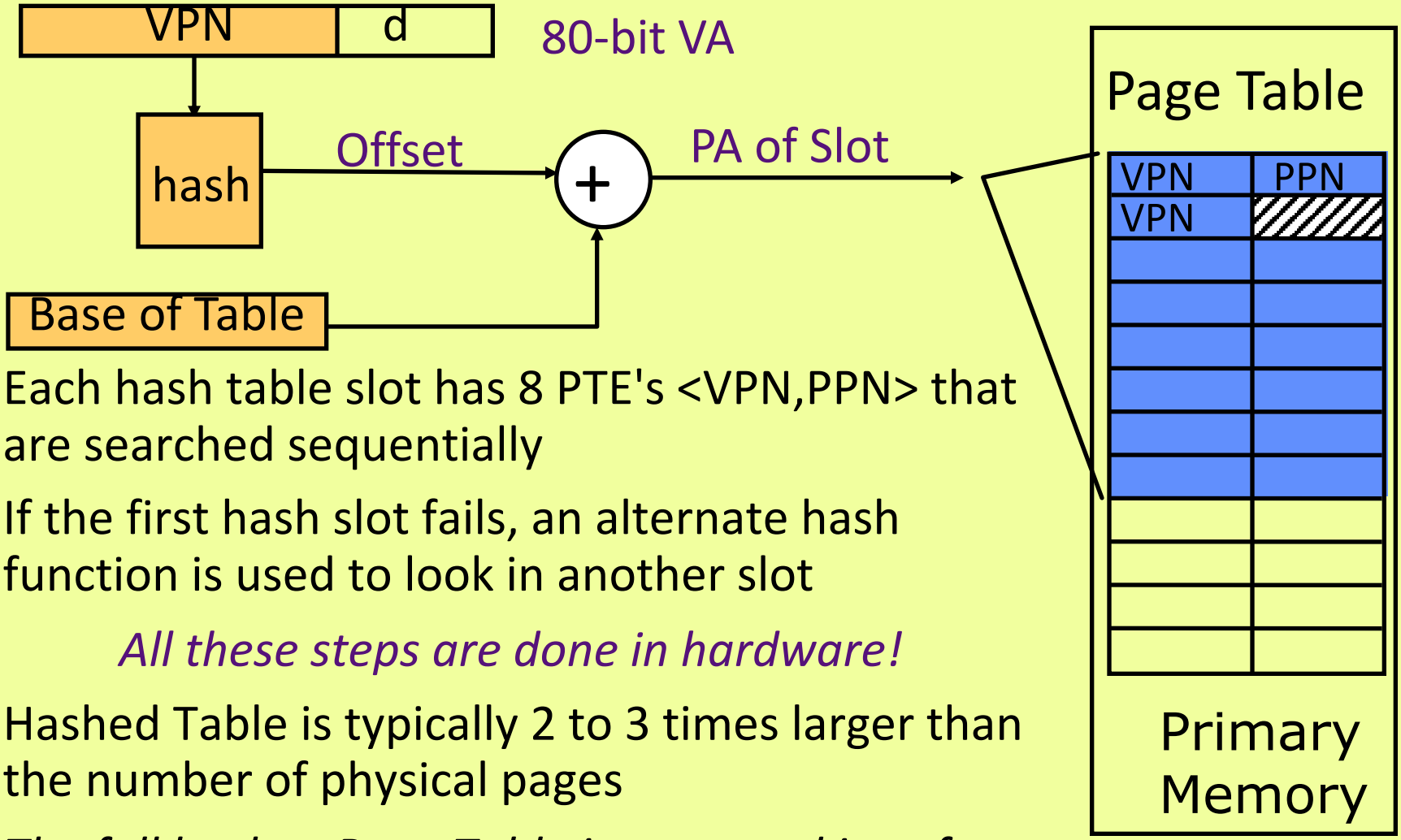


Hashed Page Table: Approximating Associative Addressing



- 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
- Modern GPUs operate similarly to vector supercomputers, with translation and protection but not demand paging

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 precise or restartable exceptions 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)
  - Krste Asanovic (MIT/UCB)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
  - David Patterson (UCB)