Virtual Memory

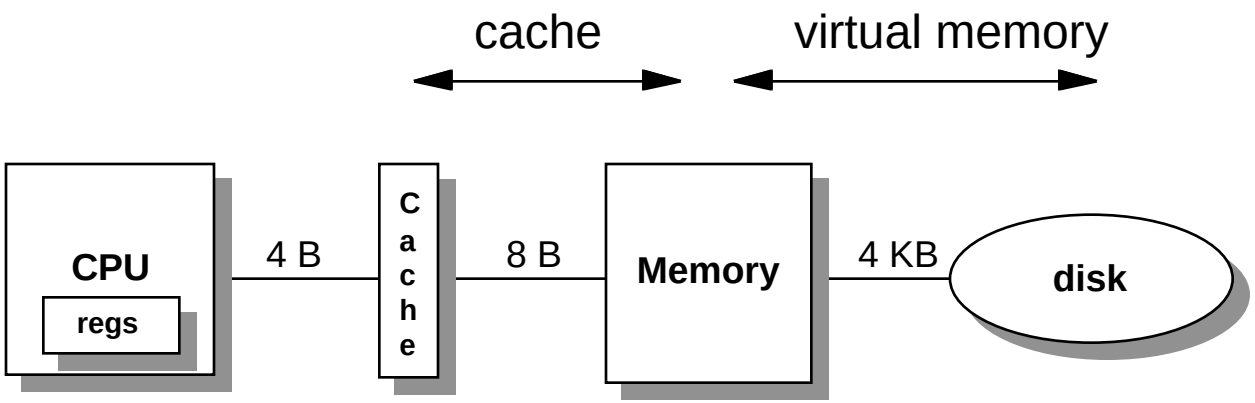
Todd C. Mowry

CS347 Lecture 9
February 10, 1998

Topics

- page tables
- TLBs
- Alpha 21064 memory system

Levels in a Typical Memory Hierarchy

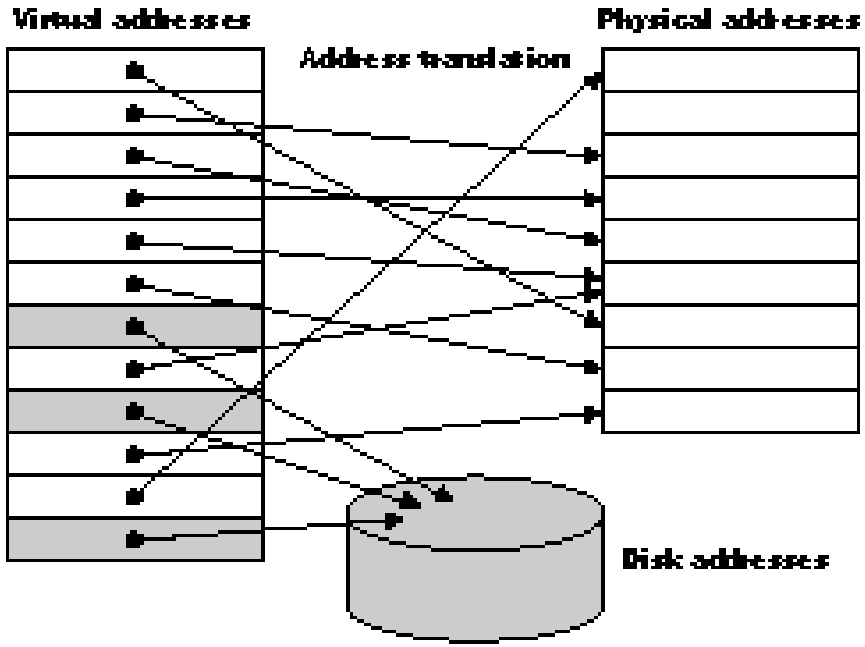


	register reference	cache reference	memory reference	disk memory reference
size:	200 B	32 KB / 4MB	128 MB	20 GB
speed:	3 ns	6 ns	100 ns	10 ms
$/Mbyte:		$256/MB	$2/MB	$0.8/MB
block size:	4 B	8 B	4 KB	

larger, slower, cheaper

Virtual Memory

Main memory can act as a cache for the secondary storage (disk)



Advantages:

- **illusion of having more physical memory**
- **program relocation**
- **protection**

Virtual Memory (cont)

Provides *illusion* of very large memory
– sum of the memory of many jobs greater than physical memory
– address space of each job larger than physical memory

Allows available (fast and expensive) physical memory to be very well utilized

Simplifies memory management (*main reason today*)

Exploits memory hierarchy to keep average access time low.

Involves at least two storage levels: *main* (RAM) and *secondary (disk)*

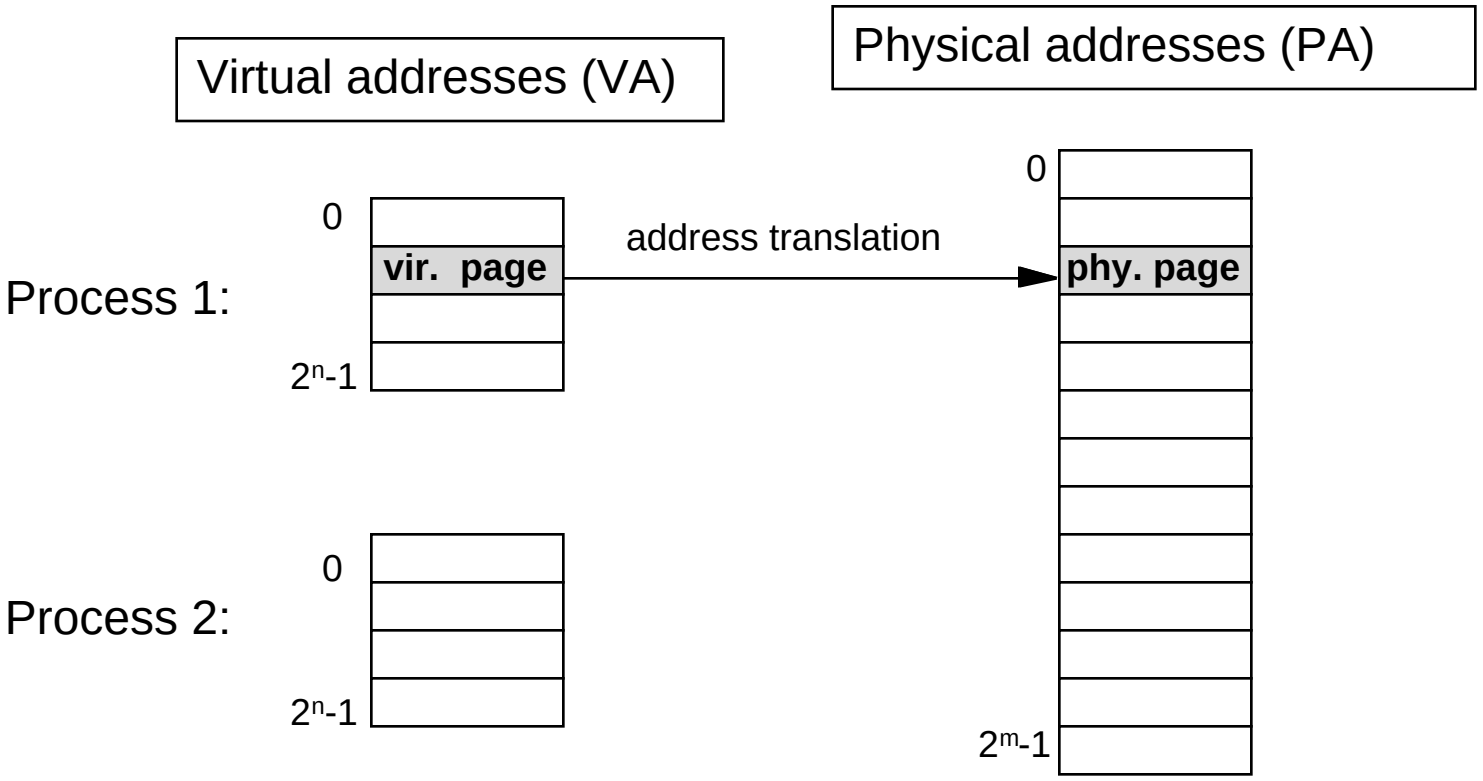
***Virtual Address* -- address used by the programmer**

***Virtual Address Space* -- collection of such addresses**

***Physical Address* -- address of word in physical memory**

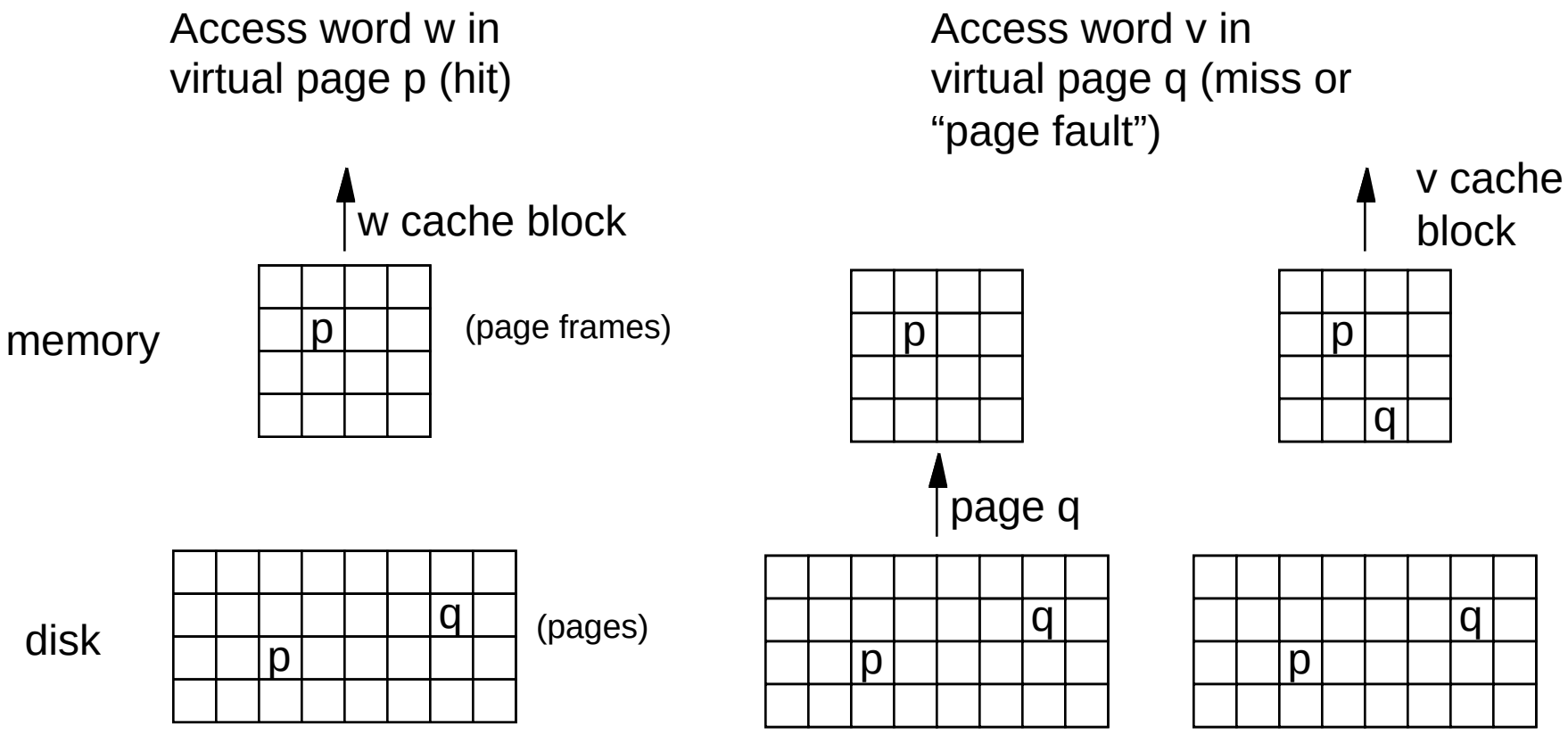
Virtual Address Spaces

Key idea: virtual and physical address spaces are divided into equal-sized blocks known as “virtual pages” and “physical pages (page frames)”



What if the virtual address spaces are bigger than the physical address space?

VM as part of the memory hierarchy



VM address translation

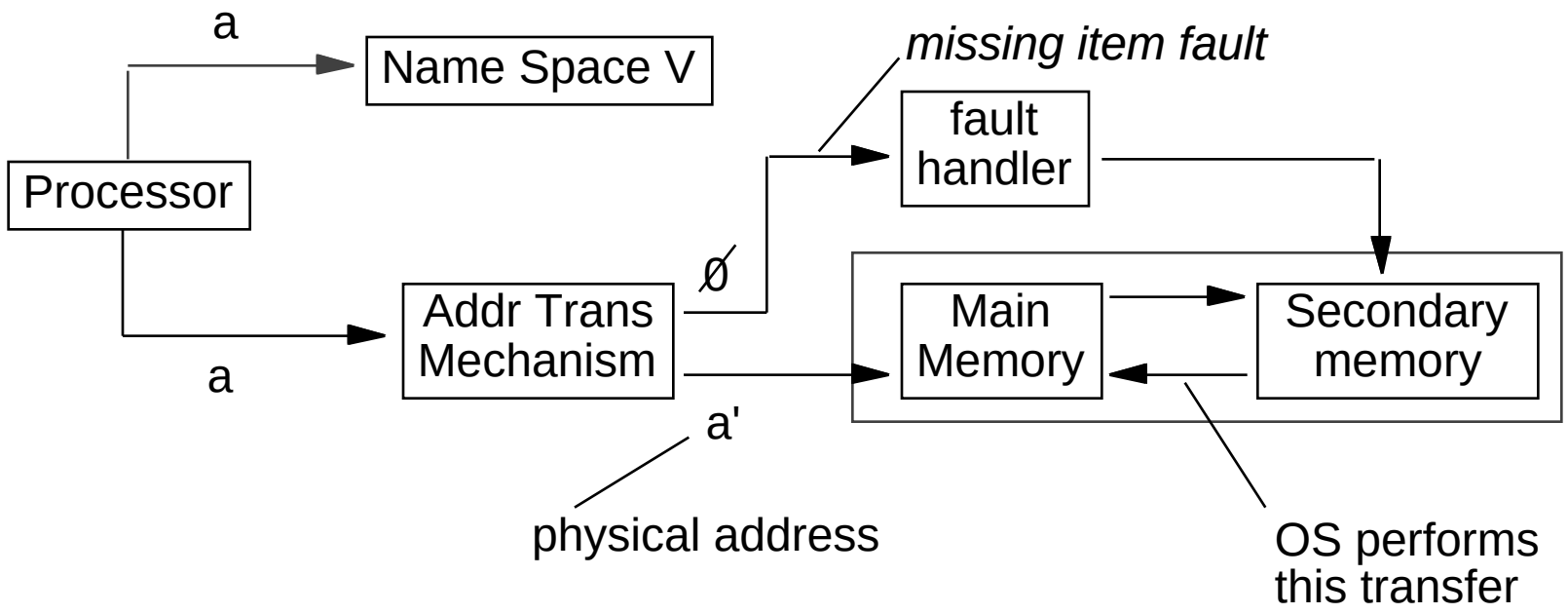
V = {0, 1, . . . , n - 1} virtual address space
M = {0, 1, . . . , m - 1} physical address space

n > m

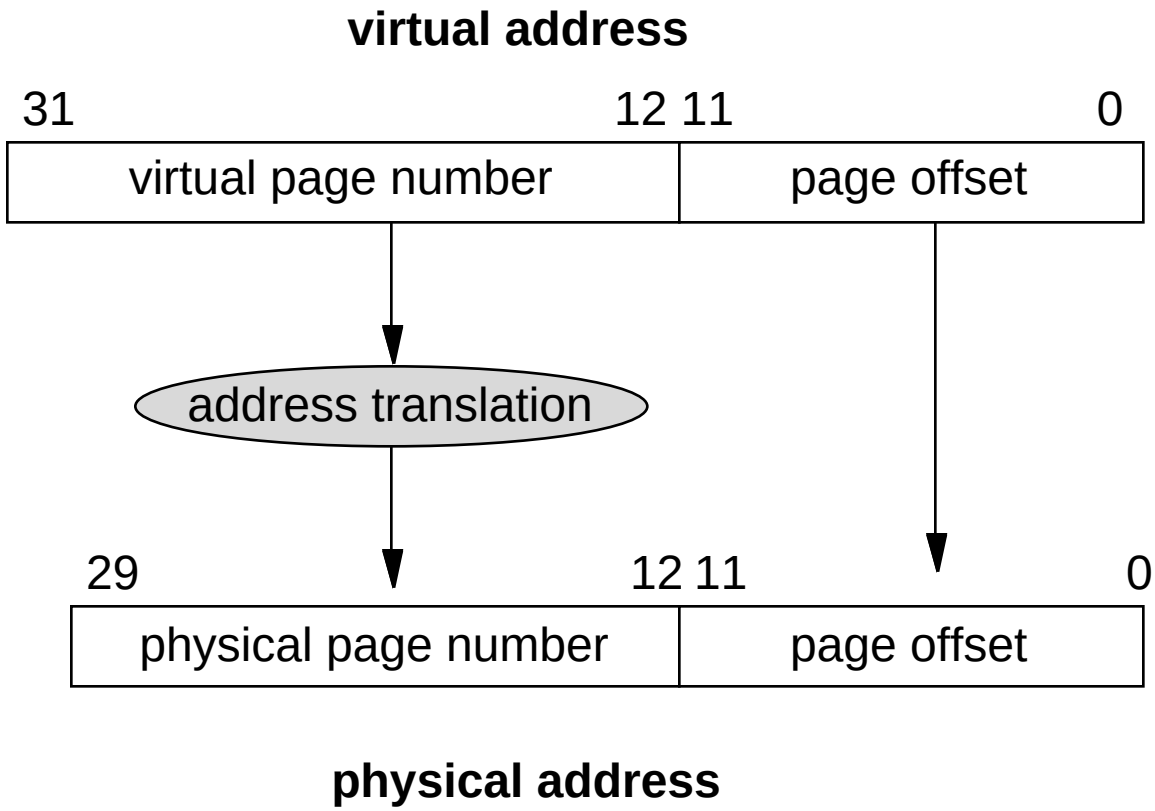
MAP: V --> M U {Ø} address mapping function

MAP(a) = a' if data at virtual address a is present at physical address a' and a' in M

= Ø if data at virtual address a is not present in M

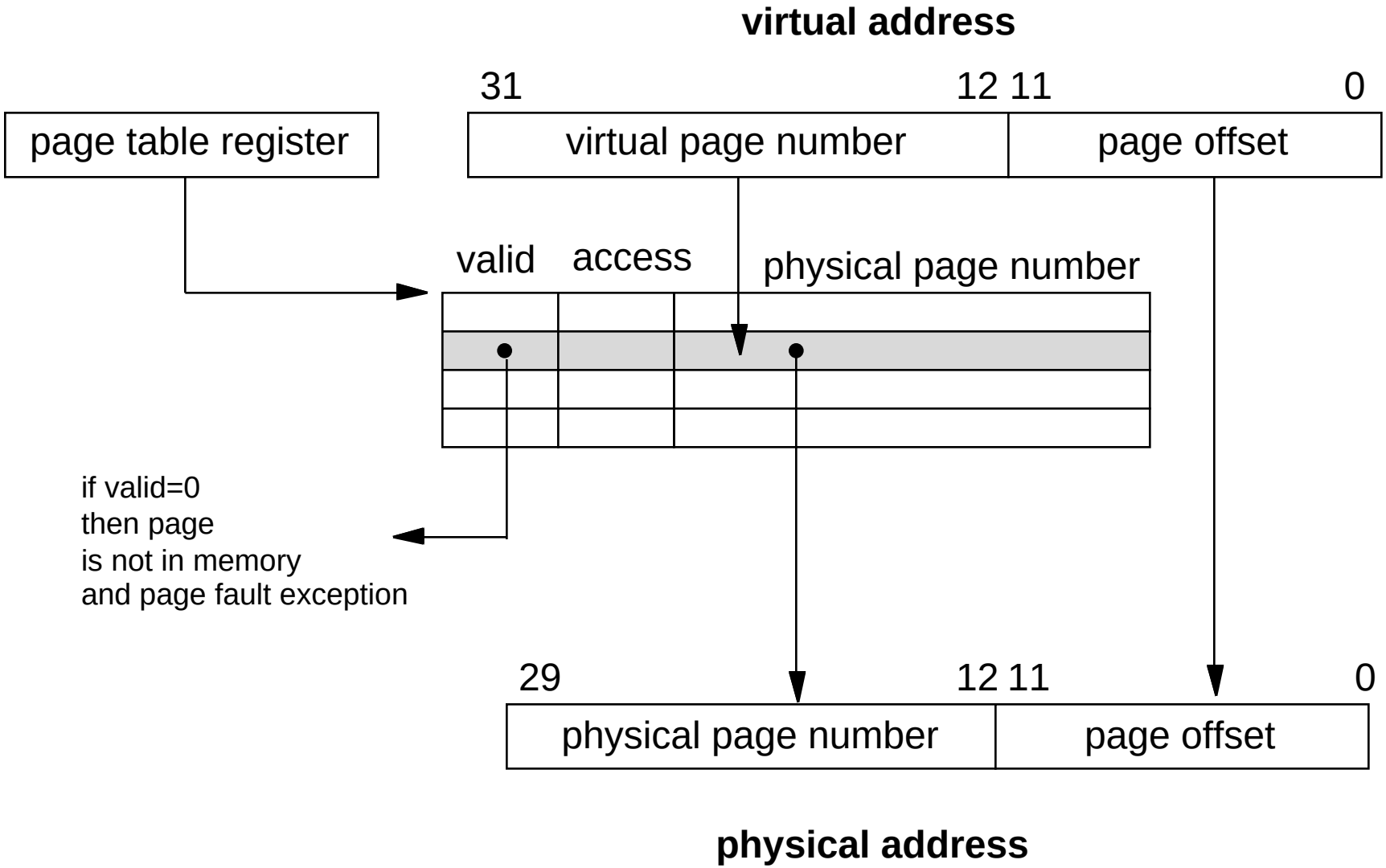


VM address translation

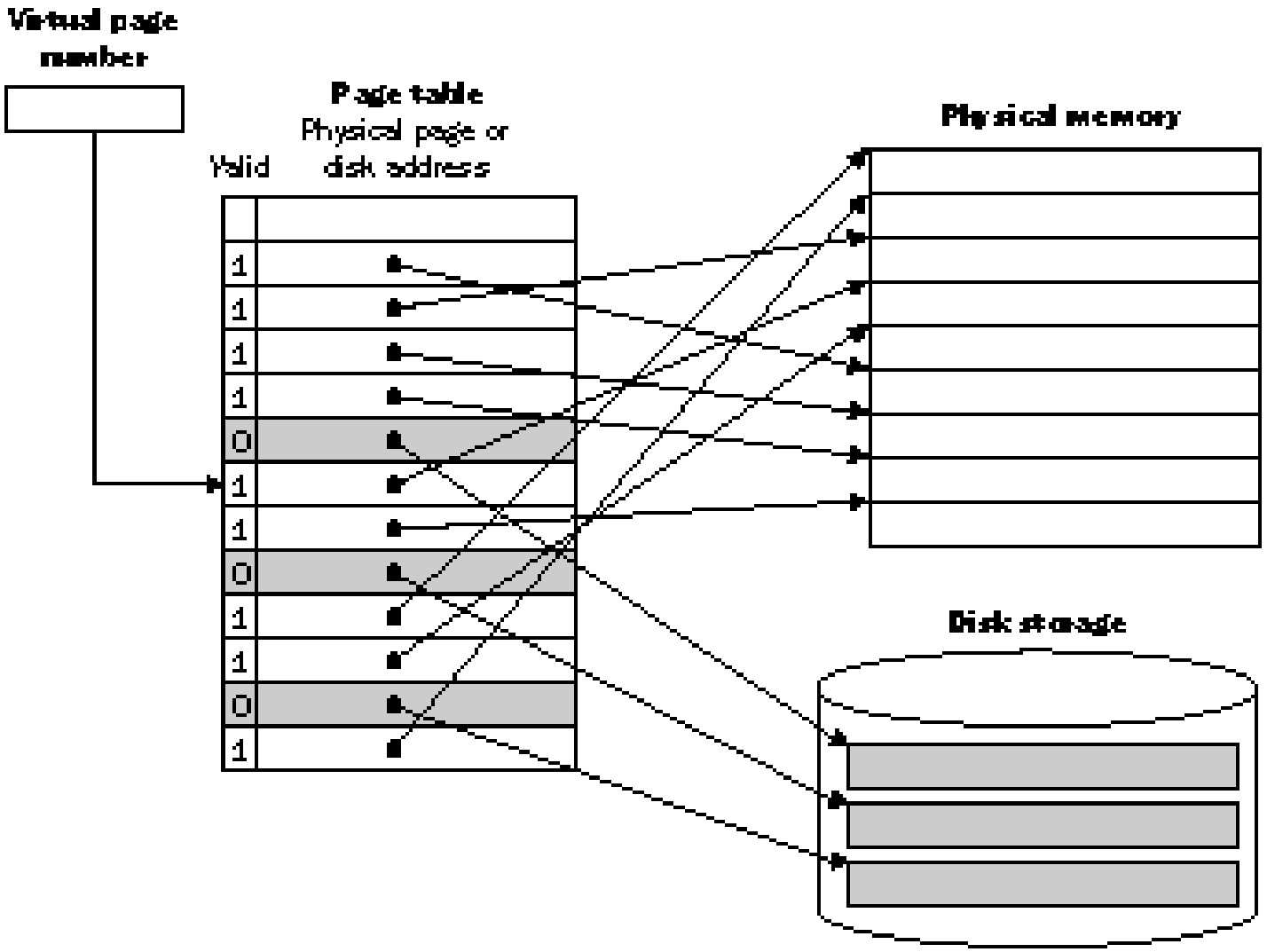


Notice that the page offset bits don't change as a result of translation

Address translation with a page table



Page Tables


Virtual page number
Page table
Physical page or disk address
Valid
1
1
1
1
0
1
1
0
1
1
0
1
Physical memory
Disk storage

Address translation with a page table (cont)

separate page table(s) per process

If V = 1
then page is in main memory at frame address stored in table
else address is location of page in secondary memory

Access Rights
R = Read-only, R/W = read/write, X = execute only

If kind of access not compatible with specified access rights,
then *protection_violation_fault*

If valid bit not set then *page fault*

Protection Fault: access rights violation; causes trap to hardware, microcode, or software fault handler

Page Fault: page not resident in physical memory, also causes trap; usually accompanied by a *context switch:* current process suspended while page is fetched from secondary storage

VM design issues

Everything driven by enormous cost of misses:

- hundreds of thousands of clocks.
- vs units or tens of clocks for cache misses.
- disks are high latency, low bandwidth devices (compared to memory)
- disk performance: 10 ms access time, 10 MBytes/sec transfer rate

Large block sizes:

- 4KBytes - 16 KBytes are typical
- amortize high access time
- reduce miss rate by exploiting locality

VM design issues (cont)

Fully associative page placement:

- eliminates conflict misses
- every miss is a killer, so worth the lower hit time

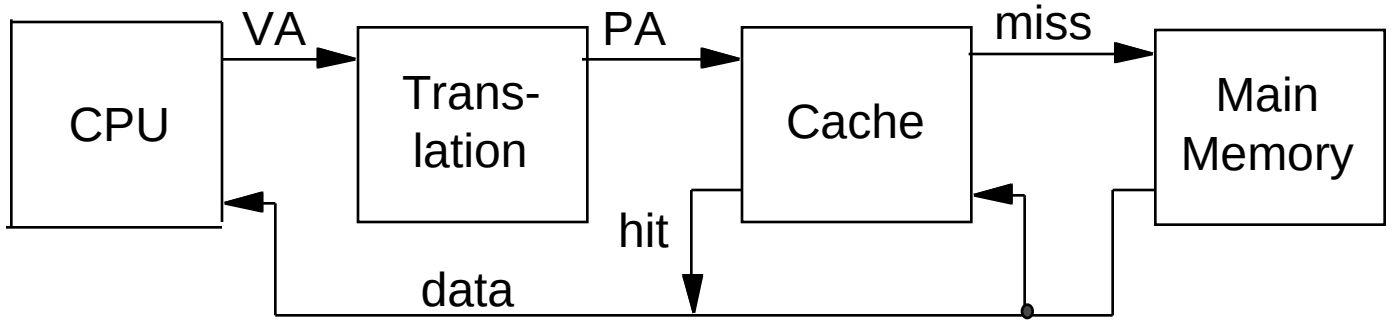
Use smart replacement algorithms

- handle misses in software
- miss penalty is so high anyway, no reason to handle in hardware
- small improvements pay big dividends

Write back only:

- disk access too slow to afford write through + write buffer

Integrating VM and cache



It takes an extra memory access to translate VA to PA. *bummer!*

Why not address cache with VA?

Aliasing problem: 2 virtual addresses that point to the same physical page.
Result: two cache blocks for one physical location

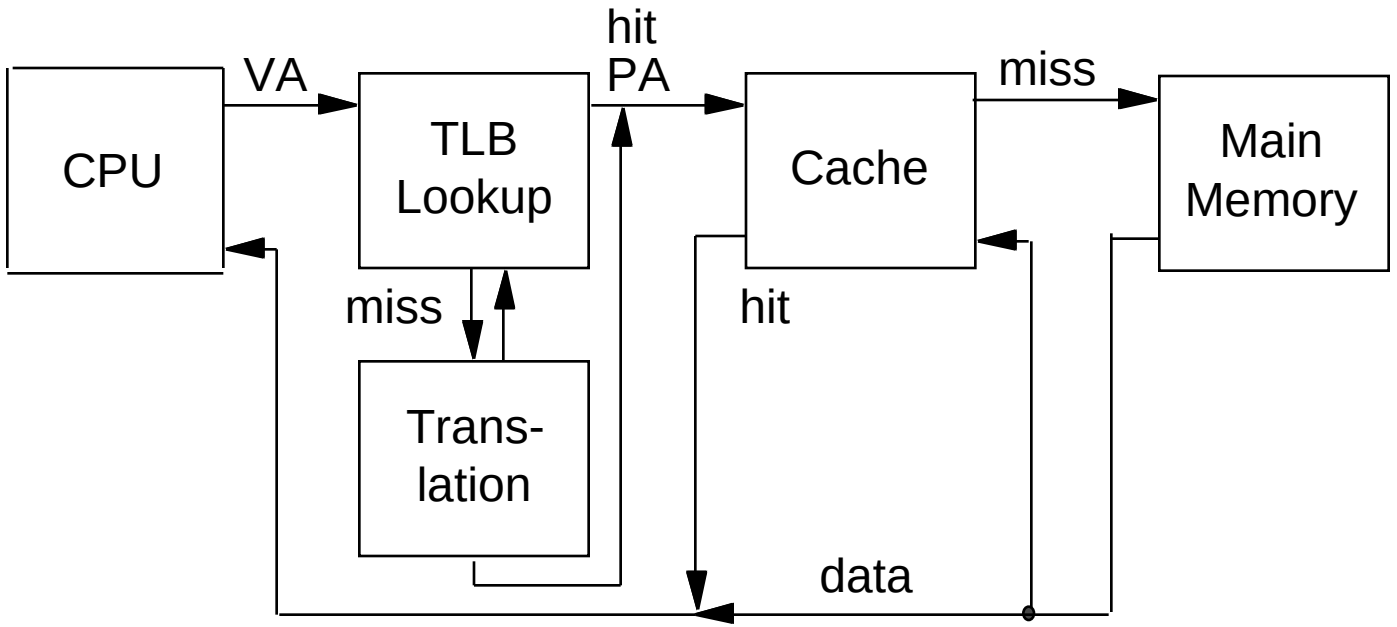
Solutions:

hardware to check for multiple hits and update multiple entries (expensive)

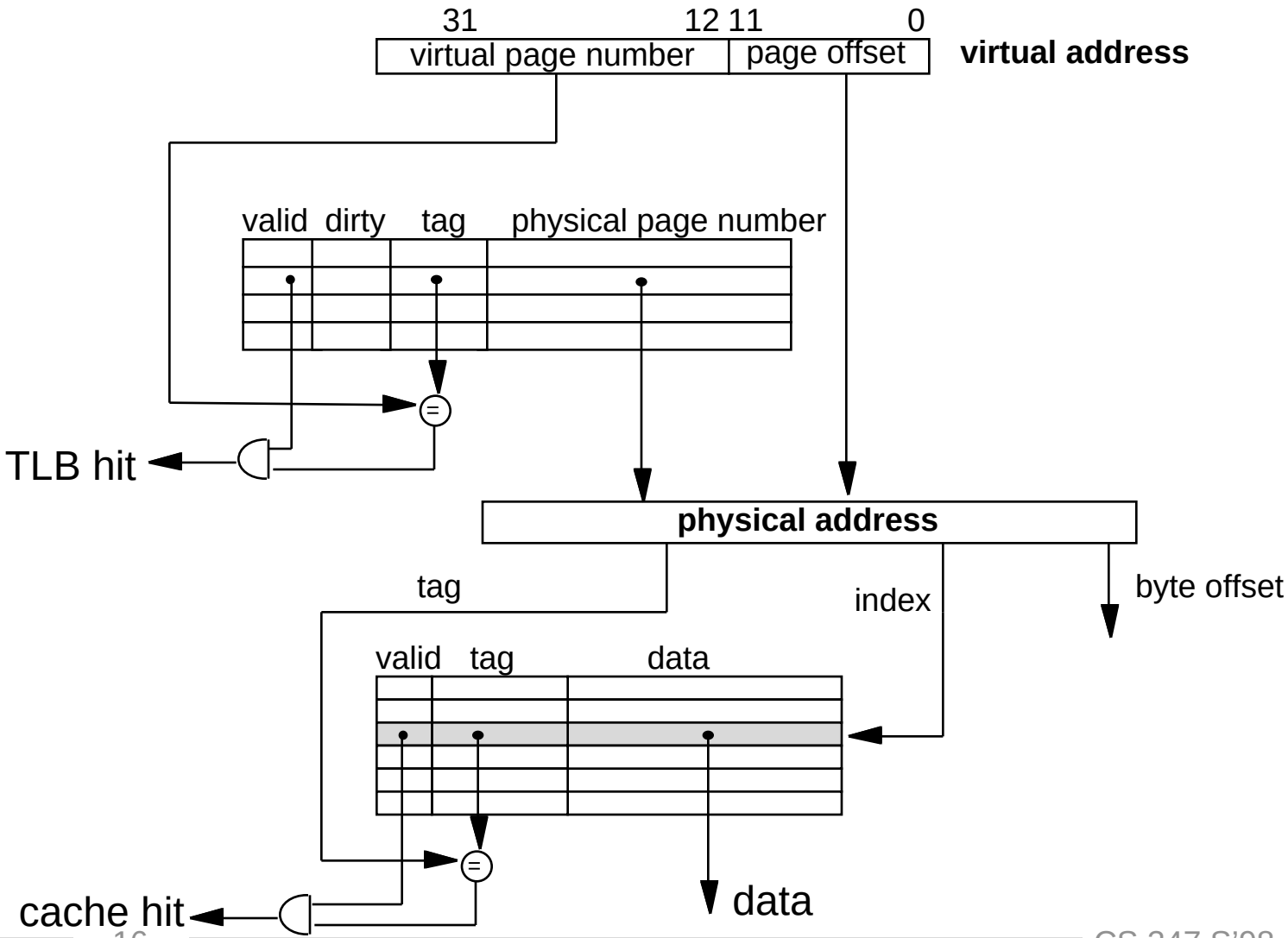
index cache with low order VA bits that don't change during translation. (requires small caches or OS support such as page coloring)

Speeding up translation with a TLB

A translation lookaside buffer (TLB) is a small, usually fully associative cache, that maps virtual page numbers to physical page numbers.

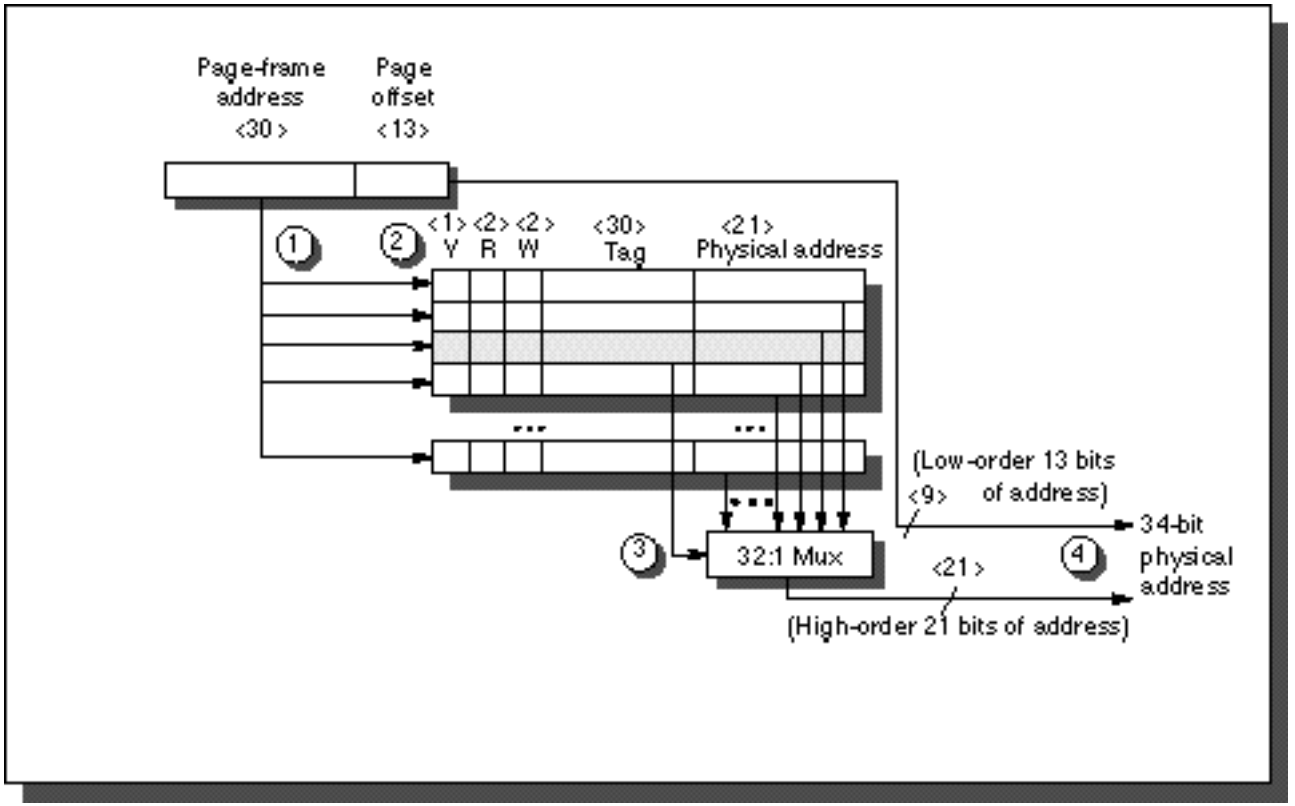


Address translation with a TLB

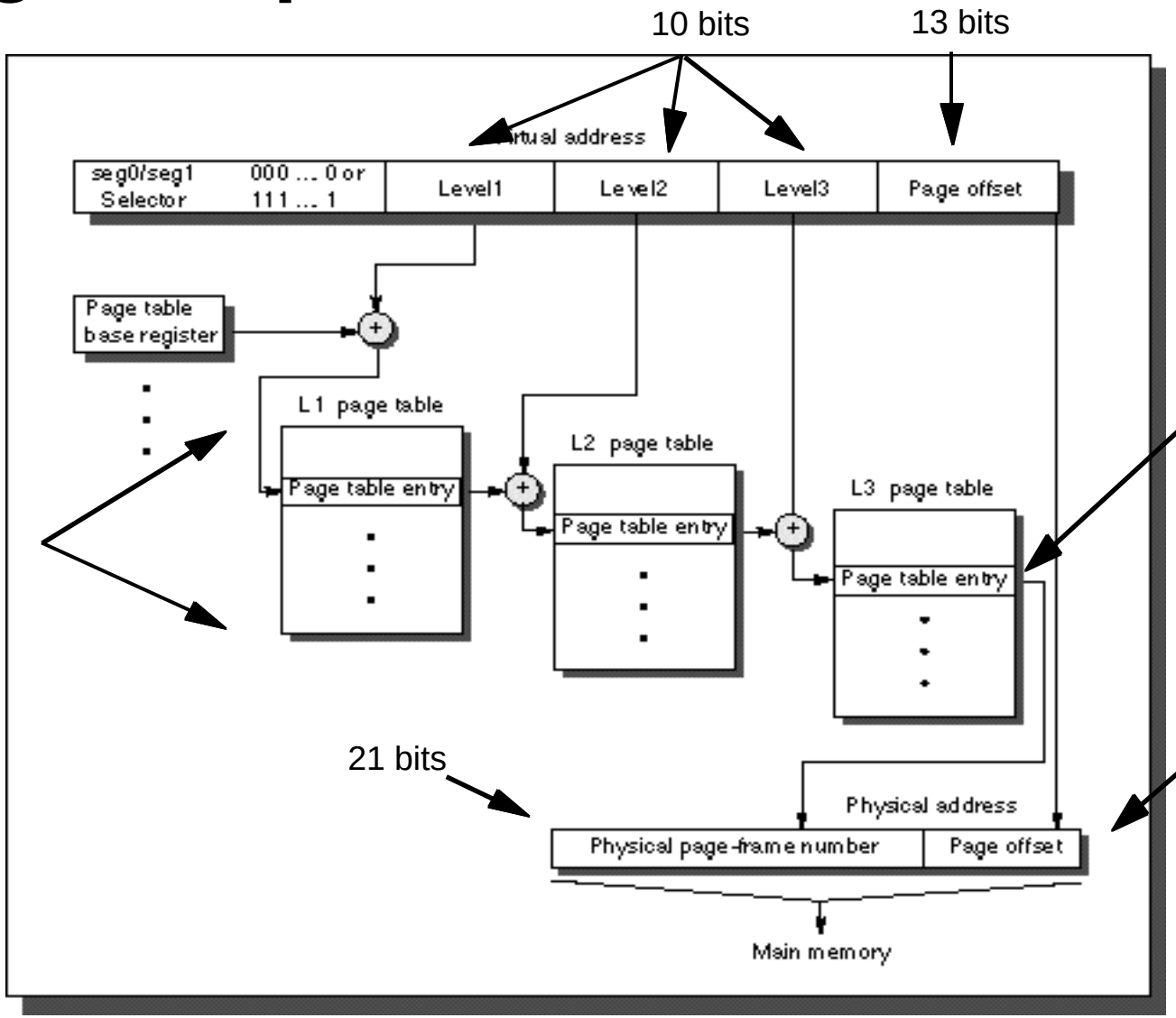


Alpha AXP 21064 TLB

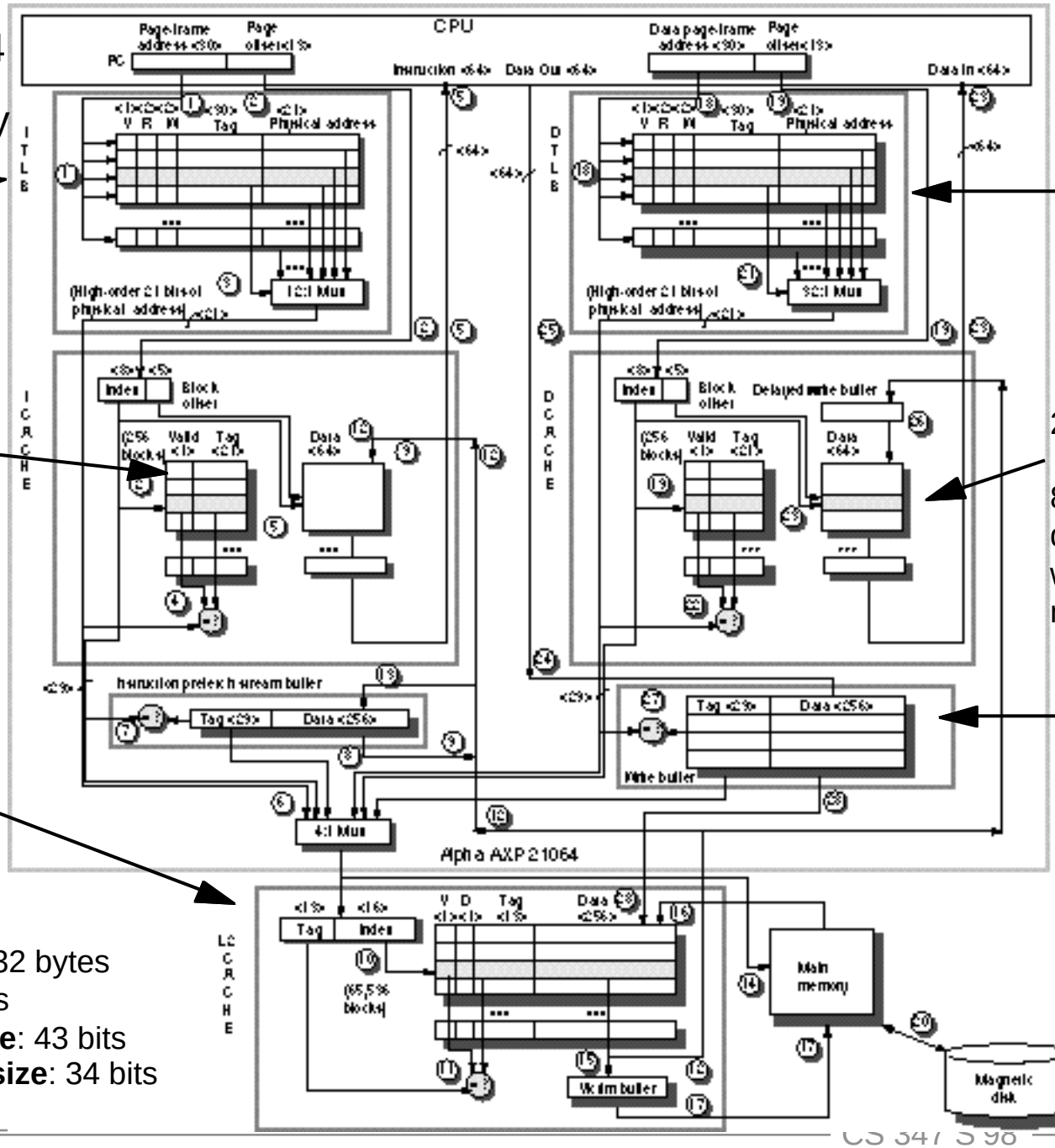
page size: 8KB
block size: 1 PTE (8 bytes)
hit time: 1 clock
miss penalty: 20 clocks
TLB size: ITLB 8 PTEs,
DTLB 32 PTEs
replacement: random(but
not last used)
placement: Fully assoc



Mapping an Alpha 21064 virtual address



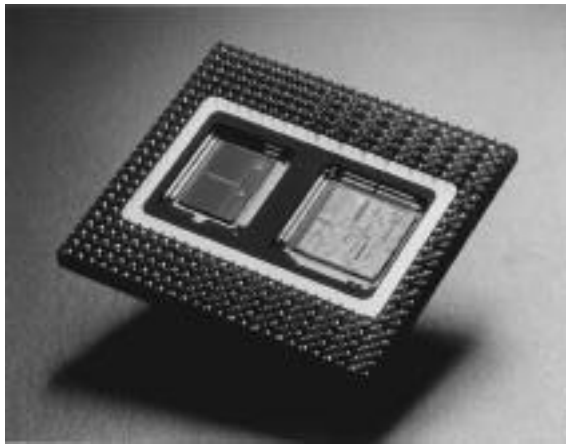
Alpha AXP 21064 memory hierarchy



cache block size: 32 bytes
page size: 8 KBytes
virtual address size: 43 bits
physical address size: 34 bits

Modern Systems

Characteristic	Intel Pentium Pro	PowerPC 604
Virtual address	32 bits	52 bits
Physical addres	32 bits	32 bits
Page size	4 KB, 4 MB	4 KB, selectable, and 256 MB
TLB organizatior	A TLB for instructions and a TLB for data Both four-way set associative Pseudo-LRU replacement Instruction TLB: 32 entries Data TLB: 64 entries TLB misses handled in hardware	A TLB for instructions and a TLB for data Both two-way set associative LRU replacement Instruction TLB: 128 entries Data TLB: 128 entries TLB misses handled in hardware



Characteristic	Intel Pentium Pro	PowerPC 604
Cache organization	Split instruction and data caches	Split intruction and data caches
Cache size	8 KB each for instructions/data	16 KB each for instructions/data
Cache associativity	Four-way set associative	Four-way set associative
Replacement	Approximated LRU replacement	LRU replacement
Block size	32 bytes	32 bytes
Write policy	Write-back	Write-back or write-through