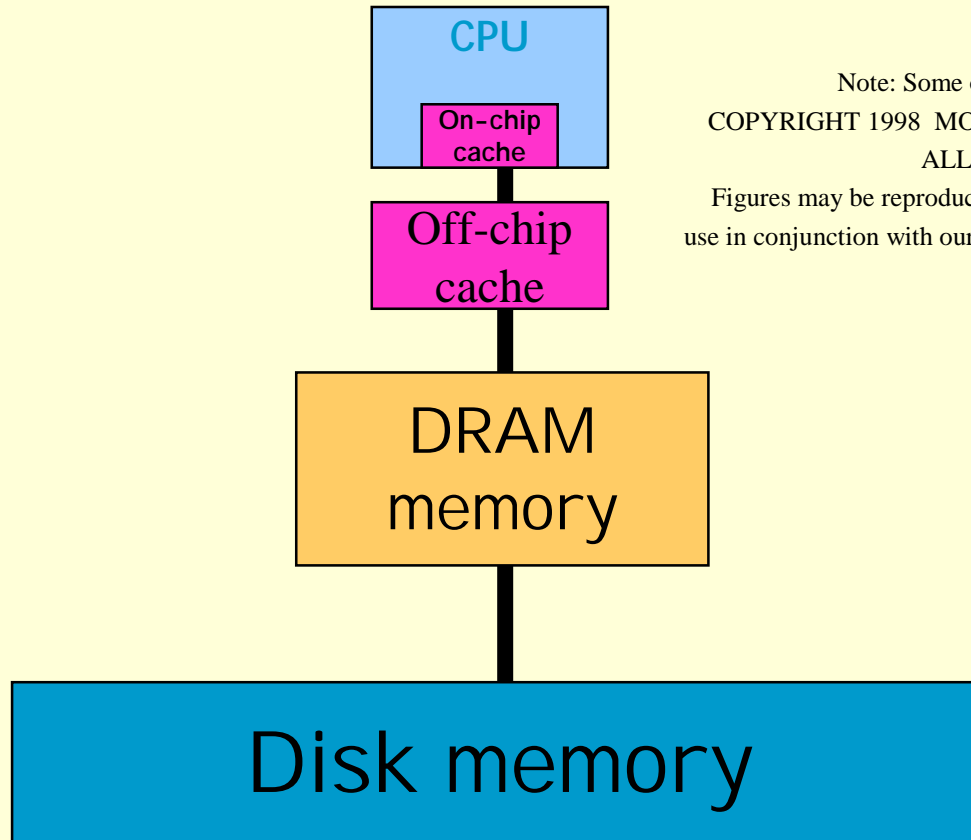


Virtual Memory



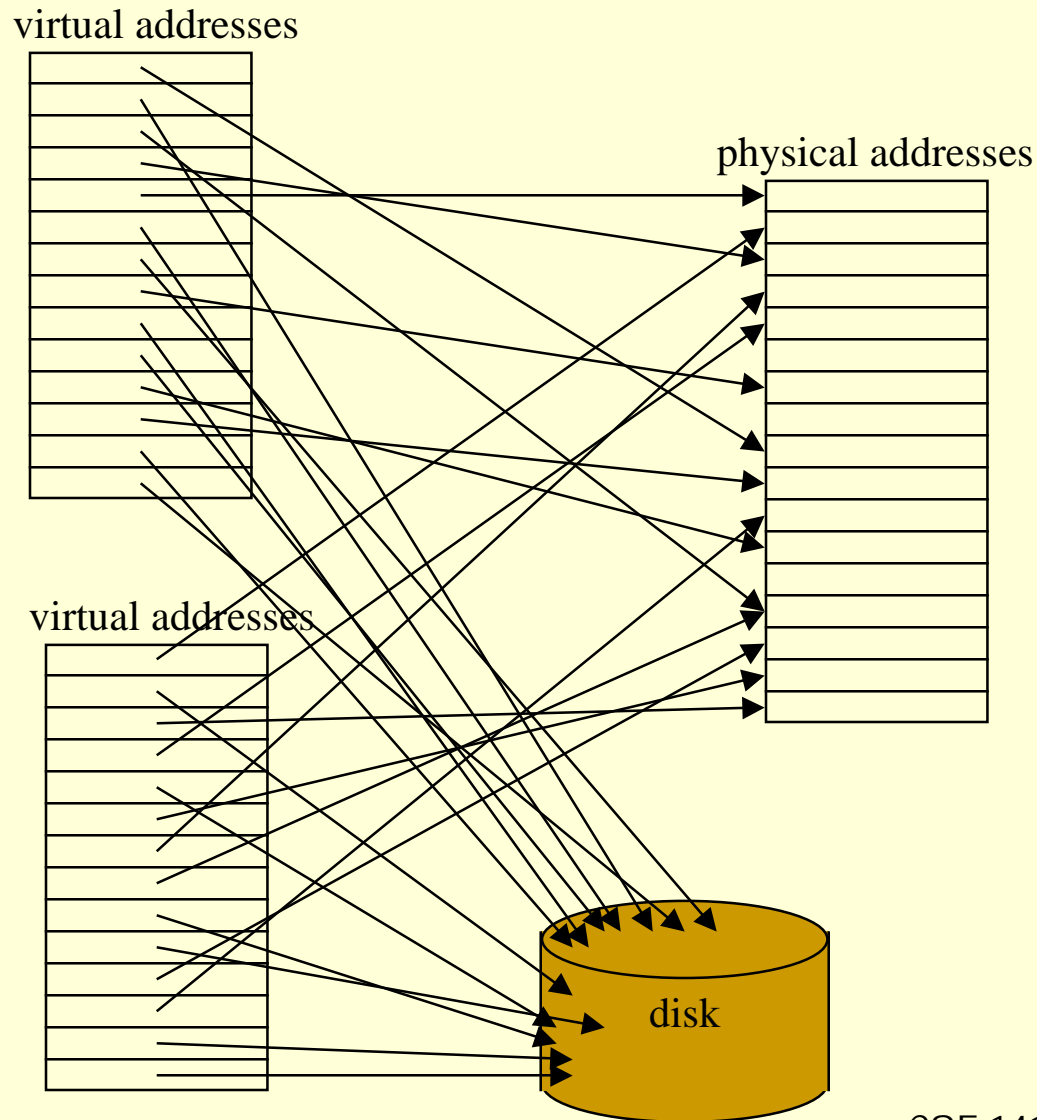
Note: Some of the material in this lecture are
COPYRIGHT 1998 MORGAN KAUFMANN PUBLISHERS, INC.
ALL RIGHTS RESERVED.

Figures may be reproduced only for classroom or personal education
use in conjunction with our text and only when the above line is included.

Virtual Memory

- Goal: every process thinks it has a huge memory ...
 - Virtual address space: addresses 0 to $2^{32}-1$
 - (or 0 to $2^{64}-1$ on machines with 64-bit addressing).
 - even if there is much less physical memory.
- ... entirely to itself.
 - Virtual memory also provides protection
 - One process can't corrupt the data of another
 - Another user, running on the same machine, can't see your data

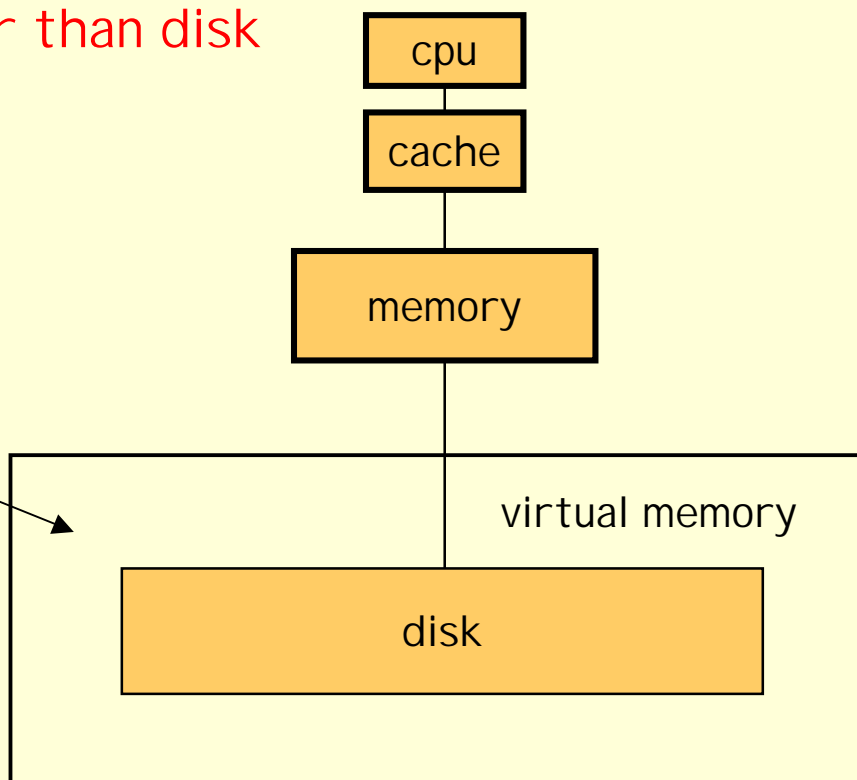
There can be many virtual address spaces



Virtual Memory

- Can be seen as another level in the memory hierarchy
 - Main memory is a cache of a larger memory space.
 - Disk holds data in VM that doesn't fit in cache
 - VM may be bigger than disk

If a process doesn't use the entire virtual memory, there will be addresses that have no storage assigned.



Virtual Memory

- For historical reasons, it uses different terminology

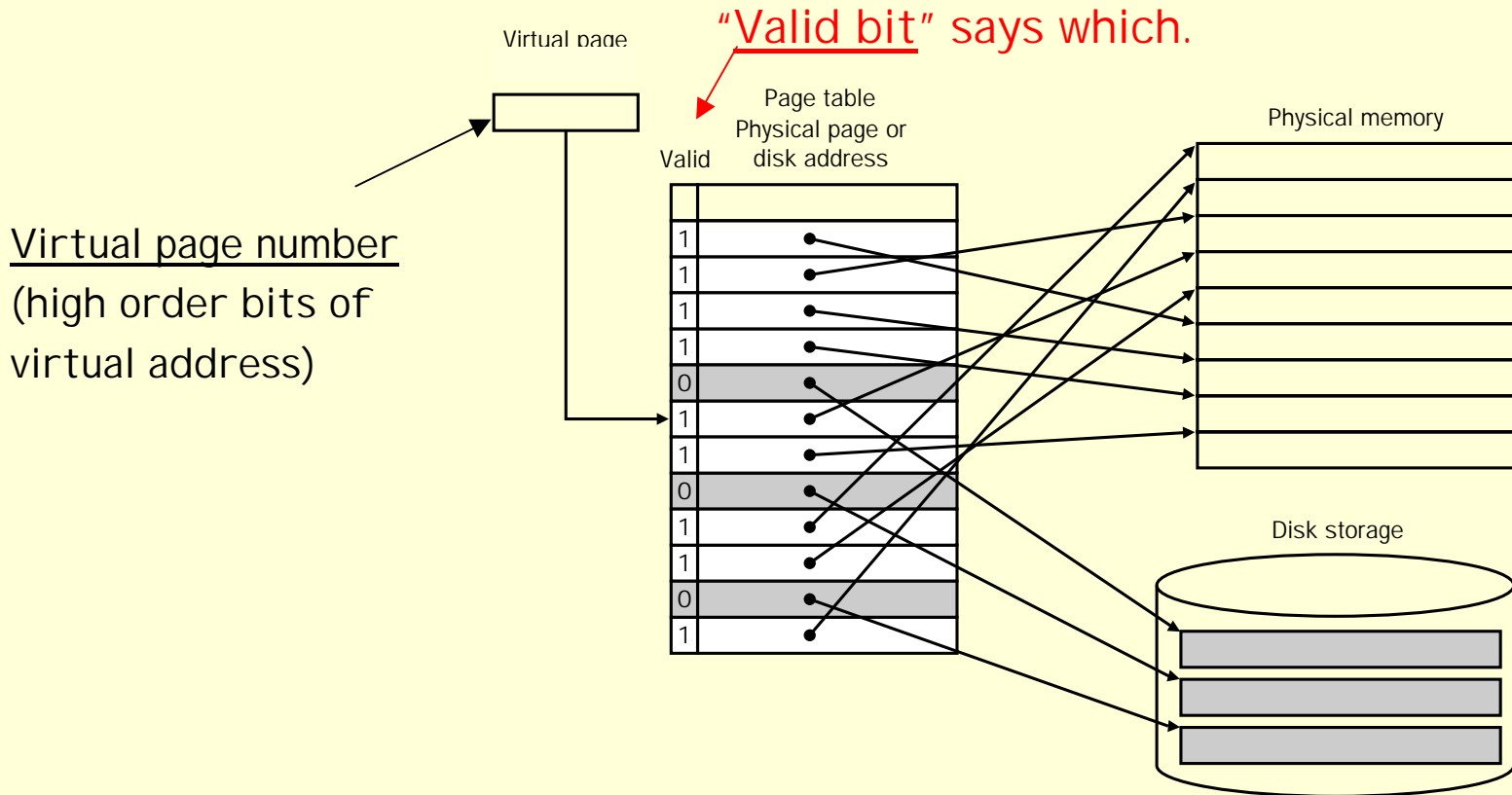
Cache	VM
block (line)	page
cache miss	page fault
address	virtual address

How VM differs from memory caches

- **MUCH** higher miss penalty (millions of cycles)!
Therefore:
 - large pages [equivalent of cache line] (4 KB to MBs)
 - associative mapping of pages (typically fully associative)
 - software handling of misses (but HW handles hits!!)
 - *write-through* never used, only write-back
- Many pages
 - With 64 MByte memory, 4KBytes/pages, have 16K pages
 - It's not practical to have 16K comparitors
 - Nor to have software do many comparisons
 - How can we get virtual memory with full associativity???

Implementation: Page Table

- Page table maps virtual memory addresses to physical memory locations
 - Page may be in main memory or may be on disk.



How big is the page table?

- Answer: Pretty big
 - Example: 512 MB virtual addresses space, 4 KB pages.
 - Then we need $2^{29} / 2^{12} = 2^{17}$ entries in the page table.
 - Suppose each entry is 4 Bytes.
 - Then page table takes 2^{19} Bytes (a half megabyte)
 - ... and each process needs its own page table.
 - So part of page table may not even be in cache.

Page table manager is part of Operating System, but ...

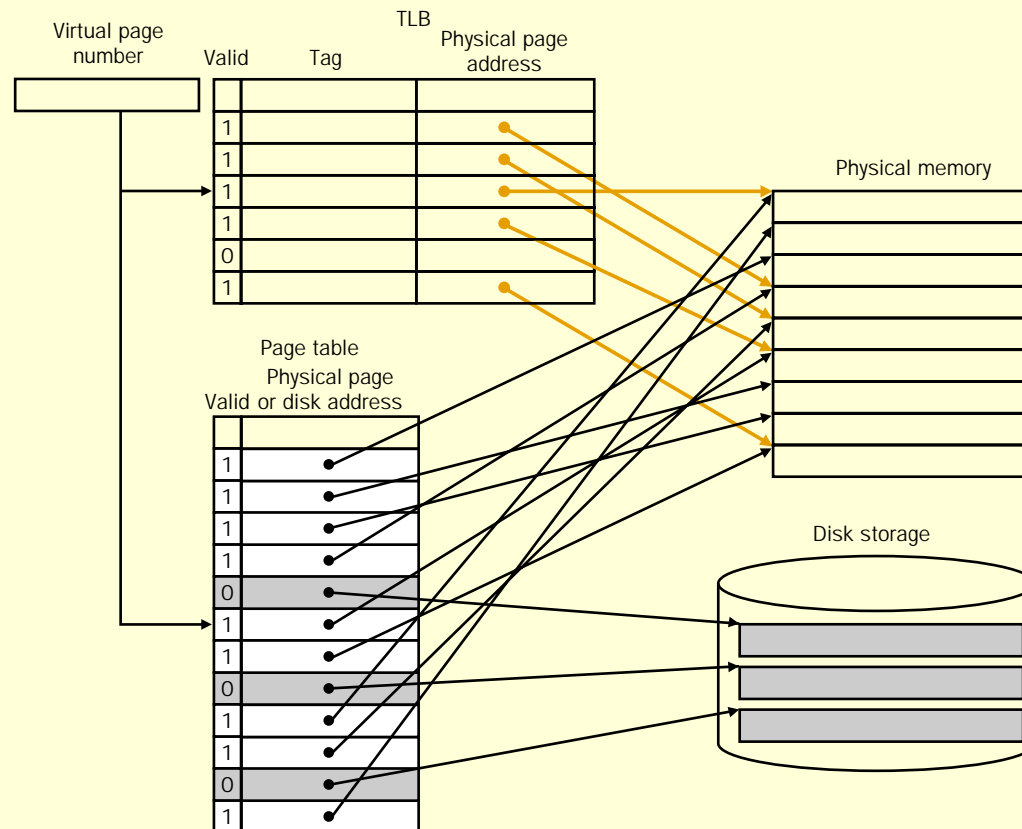
- We care about performance; can you avoid loading a page table entry on every memory reference?
 - Otherwise, VM would slow out-of-cache memory references down by a factor of two.

Making Address Translation Fast

Translation lookaside buffer (TLB)

A hardware cache for the page table

Typical size: 256 entries, 1- to 4-way set associative



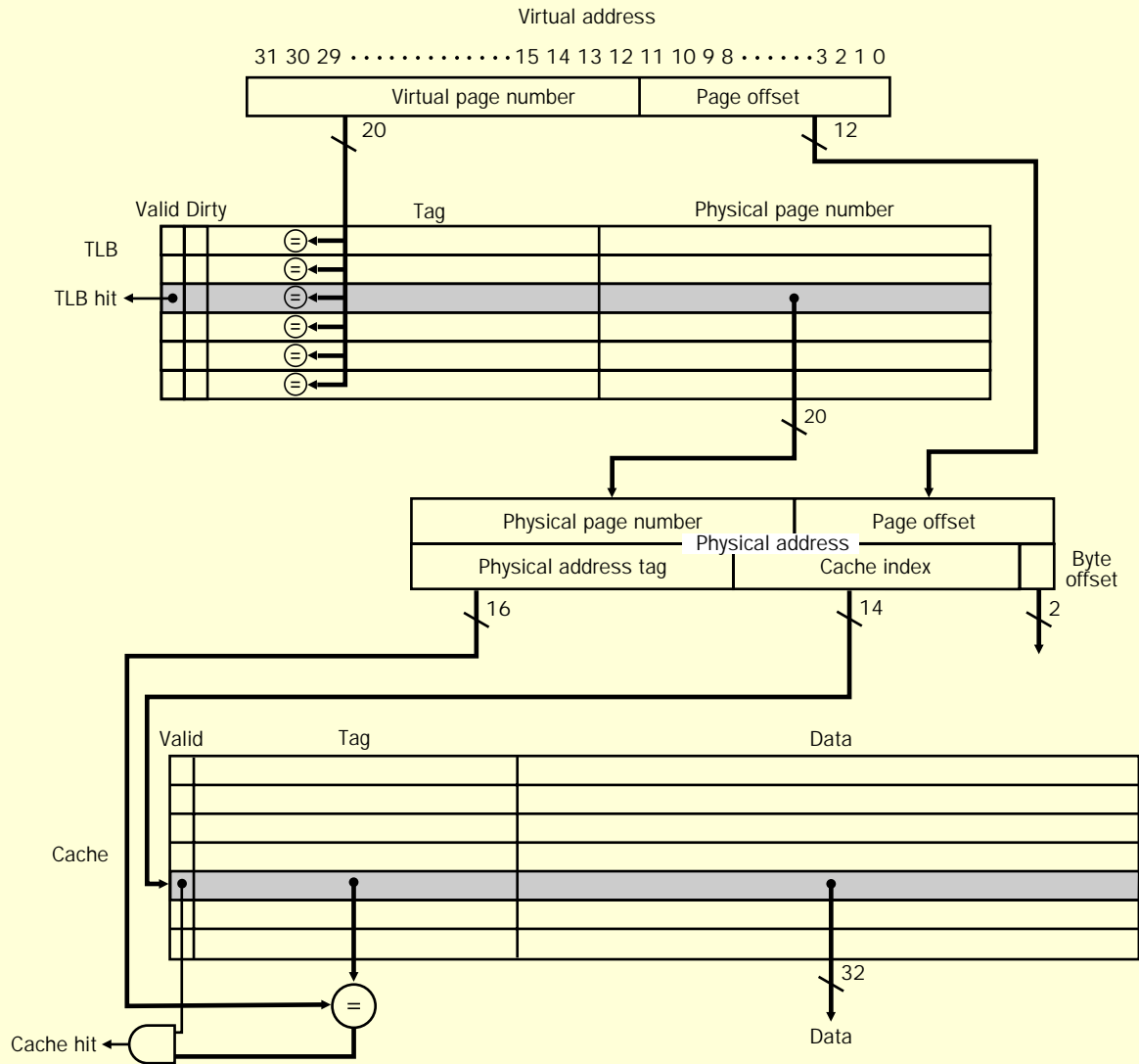
How do cache and VM fit together?

- Cache can be virtually addressed ...
 - the virtual address is broken into tag-index-offset to look up data in cache
- ... or physically addressed
 - the virtual address is first converted to a physical address (using the page table)
 - the physical address is used to find data in cache

Virtually addressed caches are faster, but make sharing data between processes complicated.

One compromise is to use virtual address for the index and physical address for the tag.

TLB and physically addressed cache



Putting it all together

Suppose we have:

- 64-bit virtual addresses
- 36-bit physical addresses
- 8 KB pages
- 256 entry, 2-way set associative TLB
- 32 KB direct mapped virtually addressed cache
- 64 Byte long cache lines

Virtual Memory Key Points

- Virtual memory provides:
 - protection
 - sharing
 - performance
 - illusion of large main memory
- Virtual Memory requires twice as many memory accesses, so we cache page table entries in the TLB.
- Four things can go wrong on a memory access: cache miss, TLB miss (but page table in cache), TLB miss and page table not in cache, page fault.