aches and Virtual memory:- Plan

- Caches
- Virtual memory as a cache for the disk.
- Virtual addresses, Pages and page-tables.
- Write-through or Write Back?
- Making Page tables fast :- Translation Look-aside Buffers.

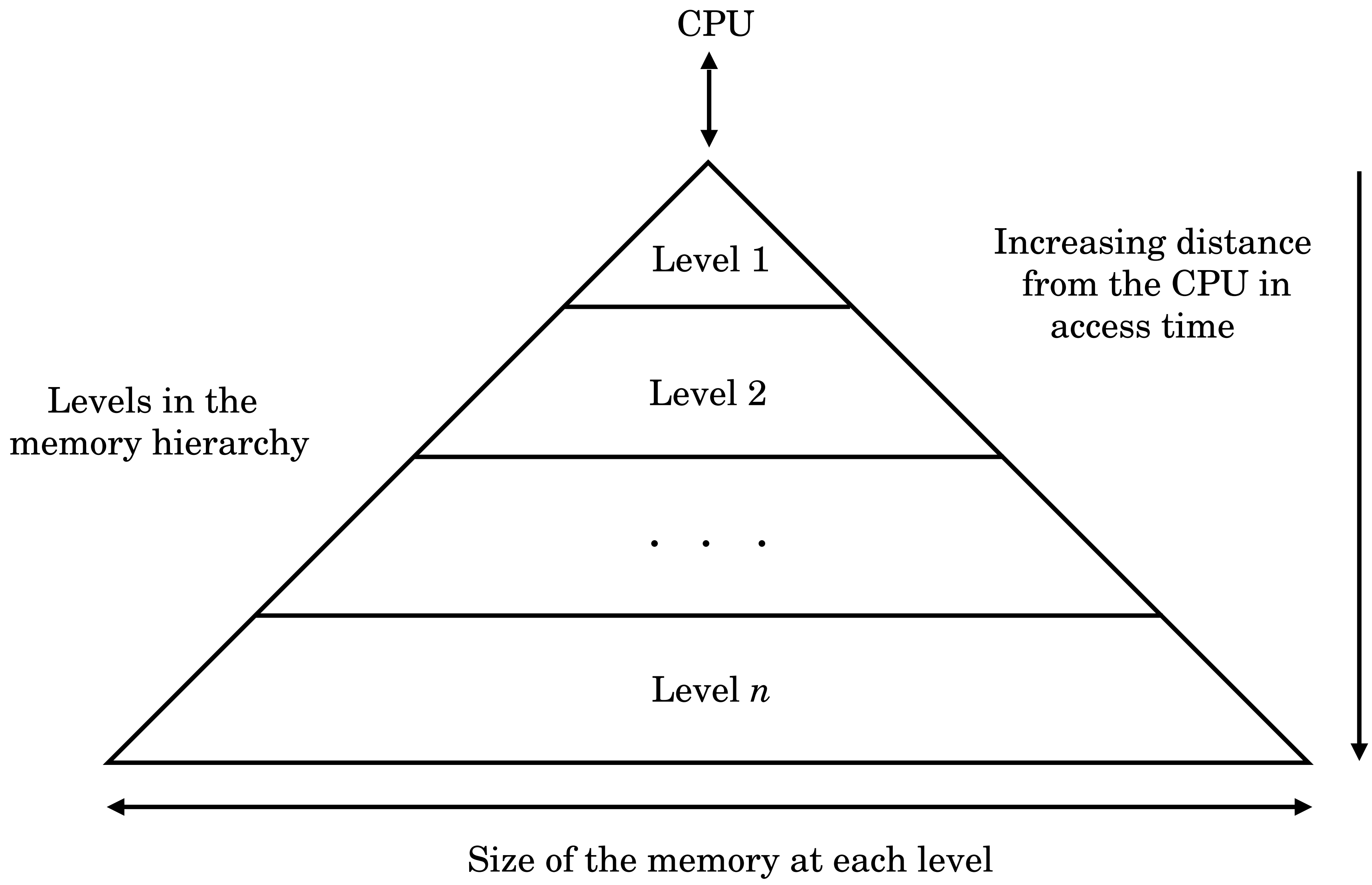
Locality

The principle of locality is an observation of how programs works, it says two things :-

- If you use something now, you'll probably need it soon.
- If you use something in memory you'll probably need the things close by.

Remember this is only an observation that many programs seem to work this way, it is not a universal law (Radix sort).

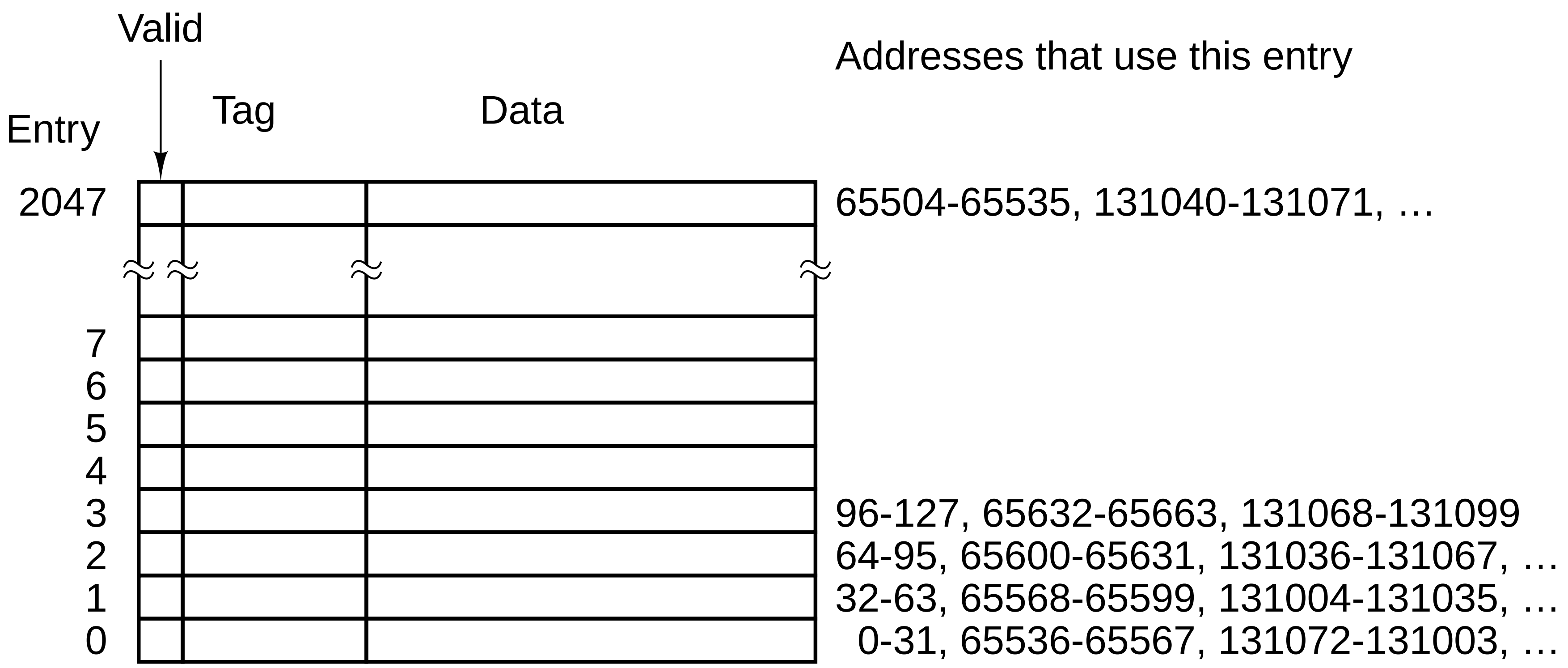
Caches


CPU
Level 1
Level 2
. . .
Level n
Levels in the memory hierarchy
Increasing distance from the CPU in access time
Size of the memory at each level

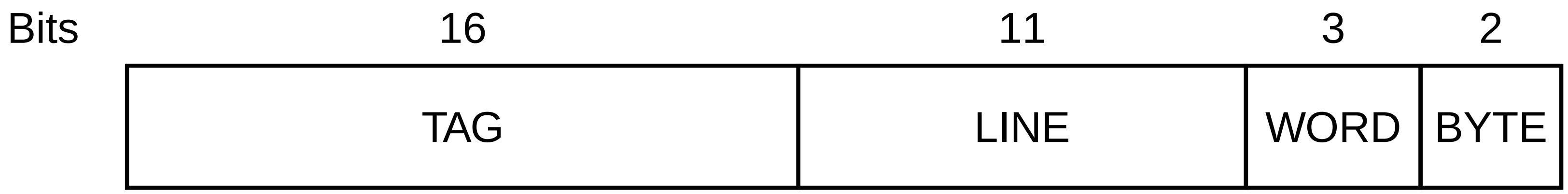
Caches

- A cache is a small fast memory near the processor, it keeps local copies of locations from the main memory.
- The simplest case type of Cache is a direct mapped Cache.
- Each cache entry consists of three parts
  - A valid bit, tells you if the data is valid.
  - A Tag field, tells you where the data came from.
  - Data field, the actual data.

Direct Mapped Caches



(a)



(b)

Caches

- A cache is a small fast memory near the processor, it keeps local copies of locations from the main memory.

Cache hit The item you are looking for is in the cache.

Cache miss the item you are looking for is not in the cache, you have to copy the item from the main memory.

Write Through/Write Back

What happens if you change the value of an item in the cache?

You have to update the value in main memory as well. There are two strategies:

Write Through Change the value in the memory when you change the value in the cache

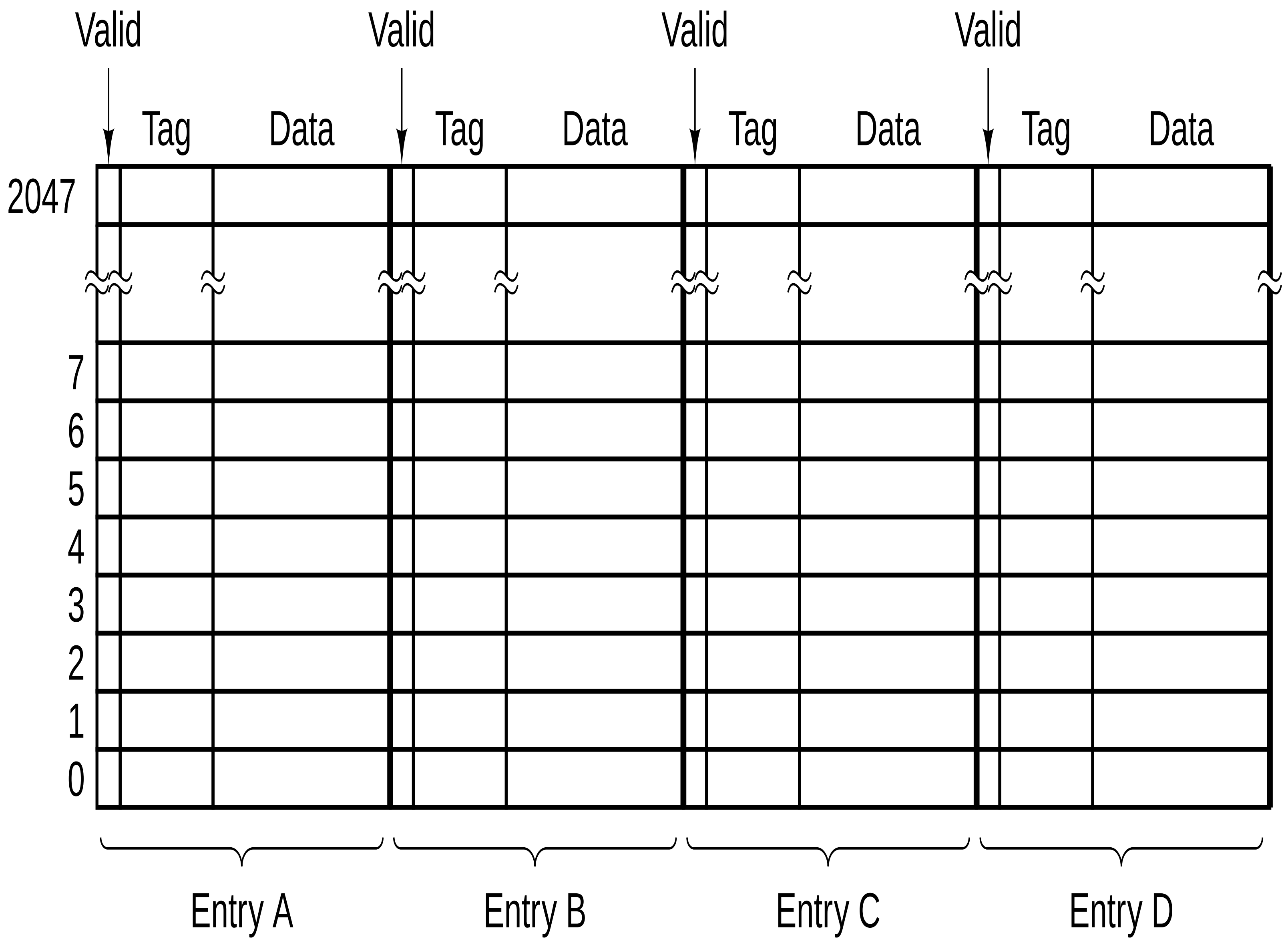
Write Back Only update the value in the main memory when you you remove an item from the cache.

Most caches use write through, simple to implement, memory transfers can go in parallel, the added complexity of implementing Write-Back doesn't always pay off.

Set-Associative Caches

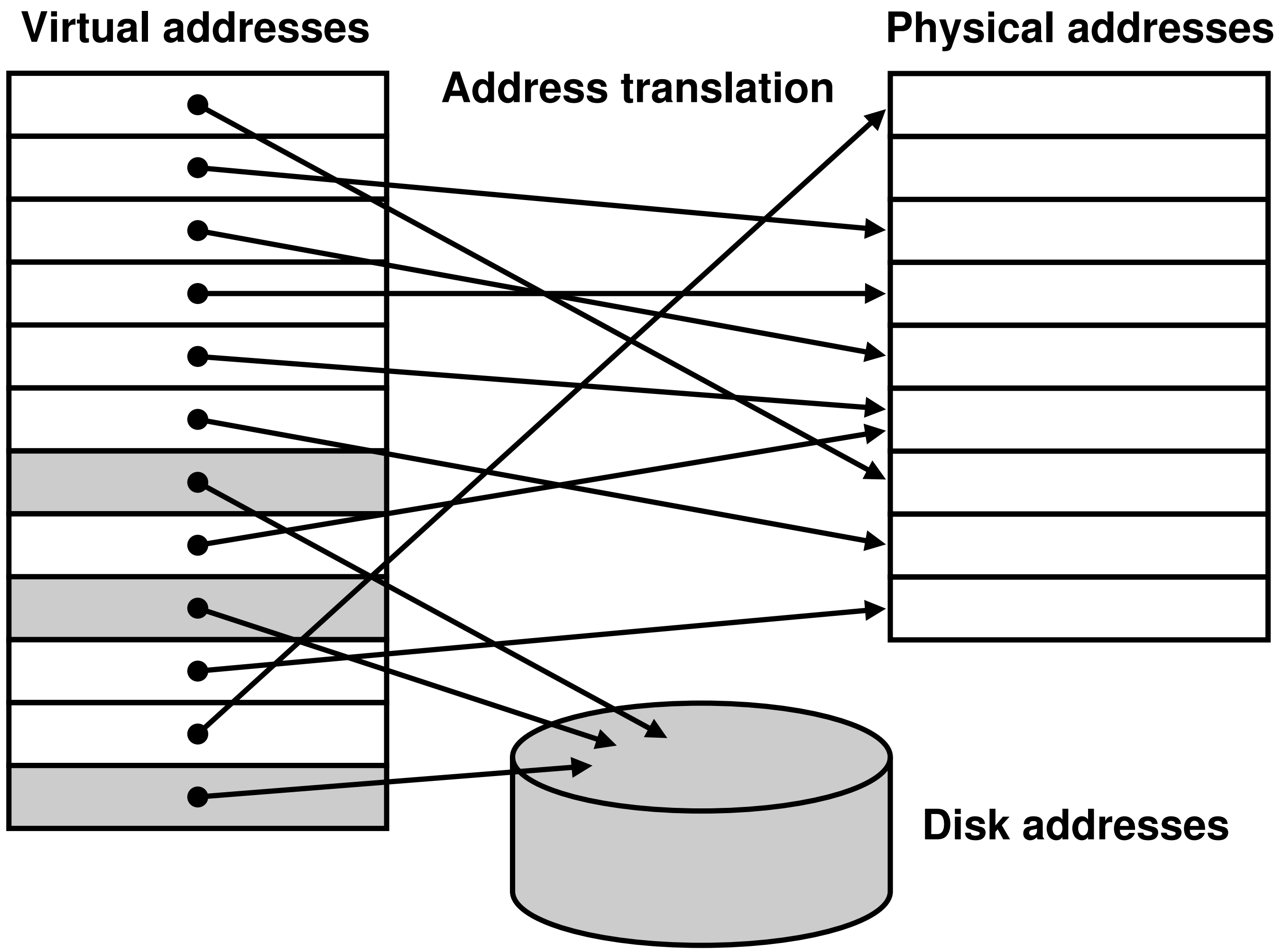
- The problem is that many differnt lines in memory compute for the same cache slots.
- A solution is to have n places for each slot.
- The cache is more complicated but there is less chance of data that you need being overwritten.

Set-Associative Cahces



Virtual Memory

First way of thinking of Virtual Memory as using the main memory as a cache for the external storage.



Virtual Memory

What does virtual memory give us?

- It allows programs to be larger than the physical memory.
- It allows programs to have identically looking address spaces, no need for relocation of the code.
- It allows more than one program to be resident at the same time and offers protection so one program can not interfere with another (the same mechanism allows sharing of code between programs).

How is all this done? Page tables and virtual addresses.

Question

Why is ok to allow programs to be larger than the physical memory?

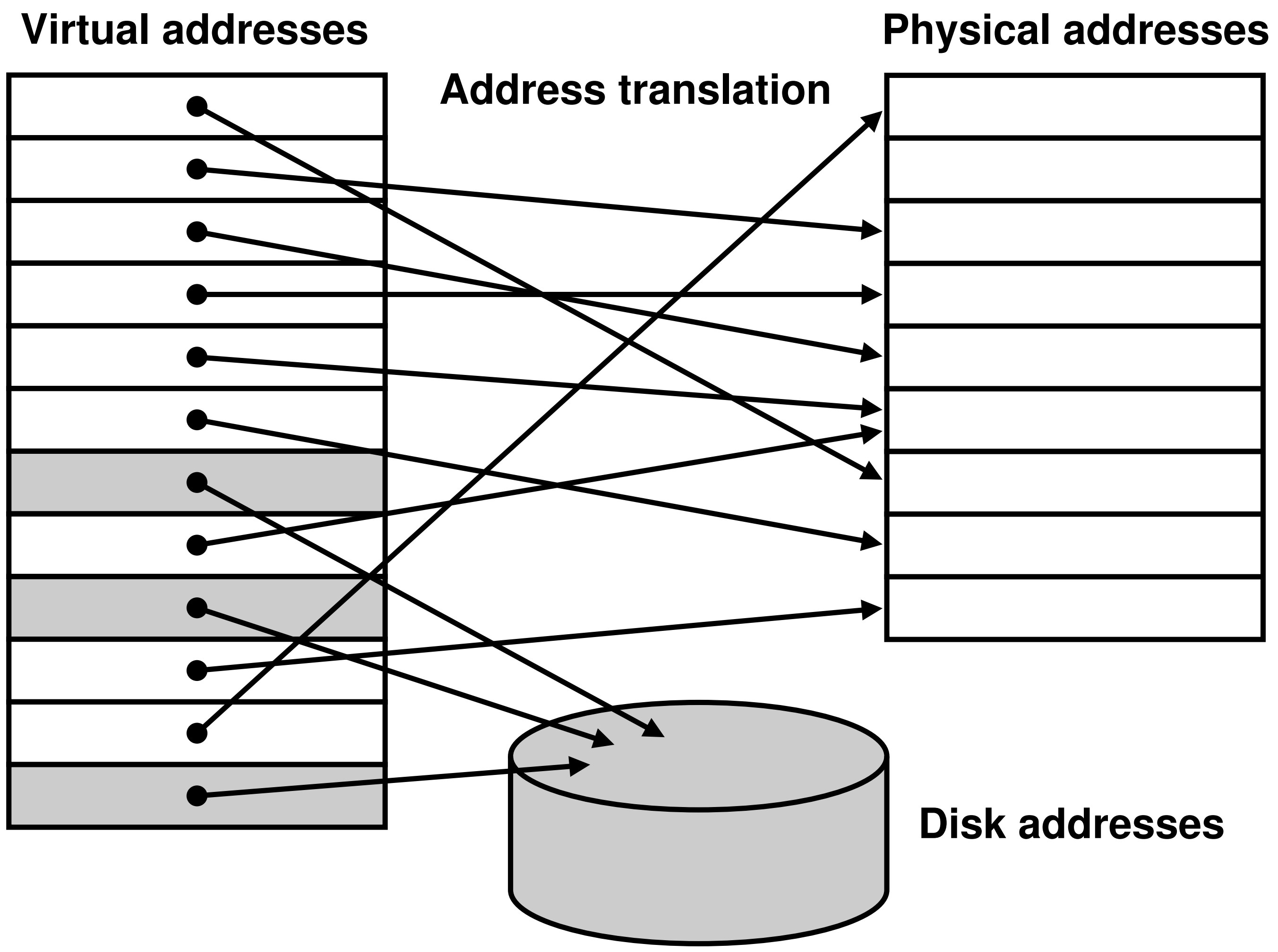
- Think about Locality

Virtual memory also automates overlays.

The Virtual Address Space

- Each process sees a virtual address space and is able to address a larger address space than is available in the physical memory.
- On the MIPS the virtual address space is 2^{32} bytes (4 Gig).
- The job of the virtual memory system is to map virtual memory addresses to physical addresses and to manage which pages be stored on the disc.

he Virtual Address Space



ages and Virtual Addresses

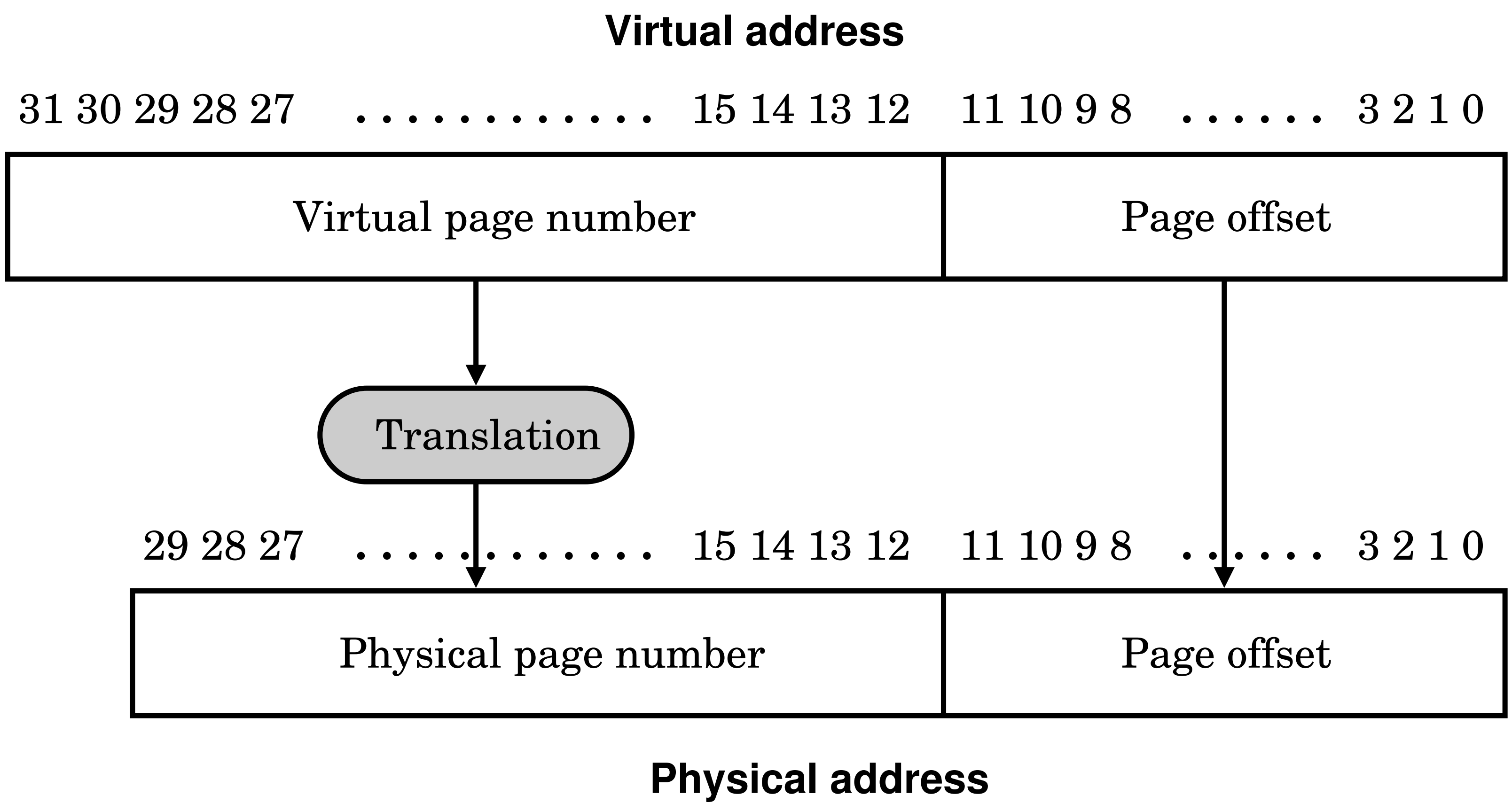
A Cache is divided into a number of lines, when data is loaded in from the main memory more than one location is loaded in to take advantage of spatial locality.

The idea of the virtual memory system system swap in and out data between the disc and the main memory.

Because disc access is much slower than main memory it is better to swap in and out larger chunks than we do with the Cache. Typically the memory is divided into larger chunks, of sizes 4k,8K or larger.

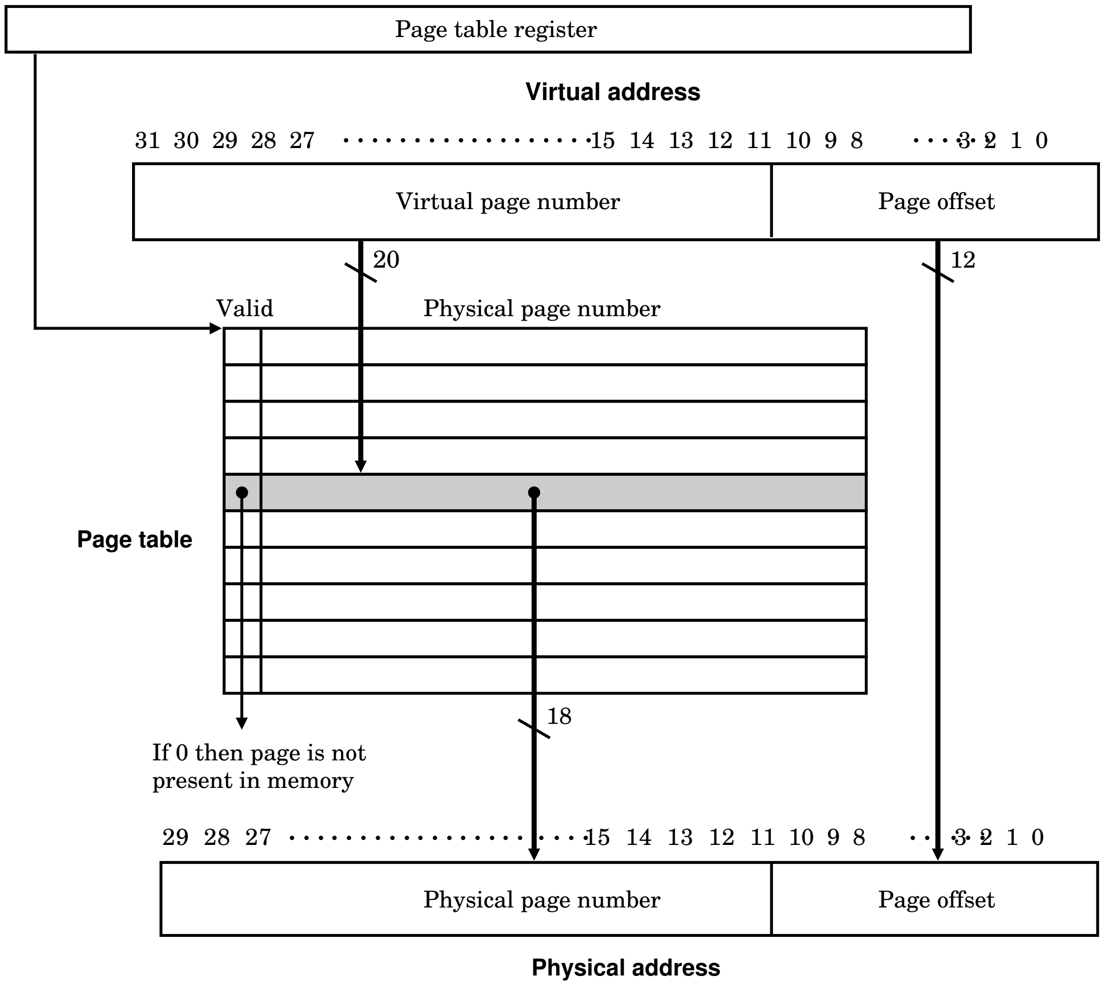
These chunks are referred to as pages.

Virtual Addresses



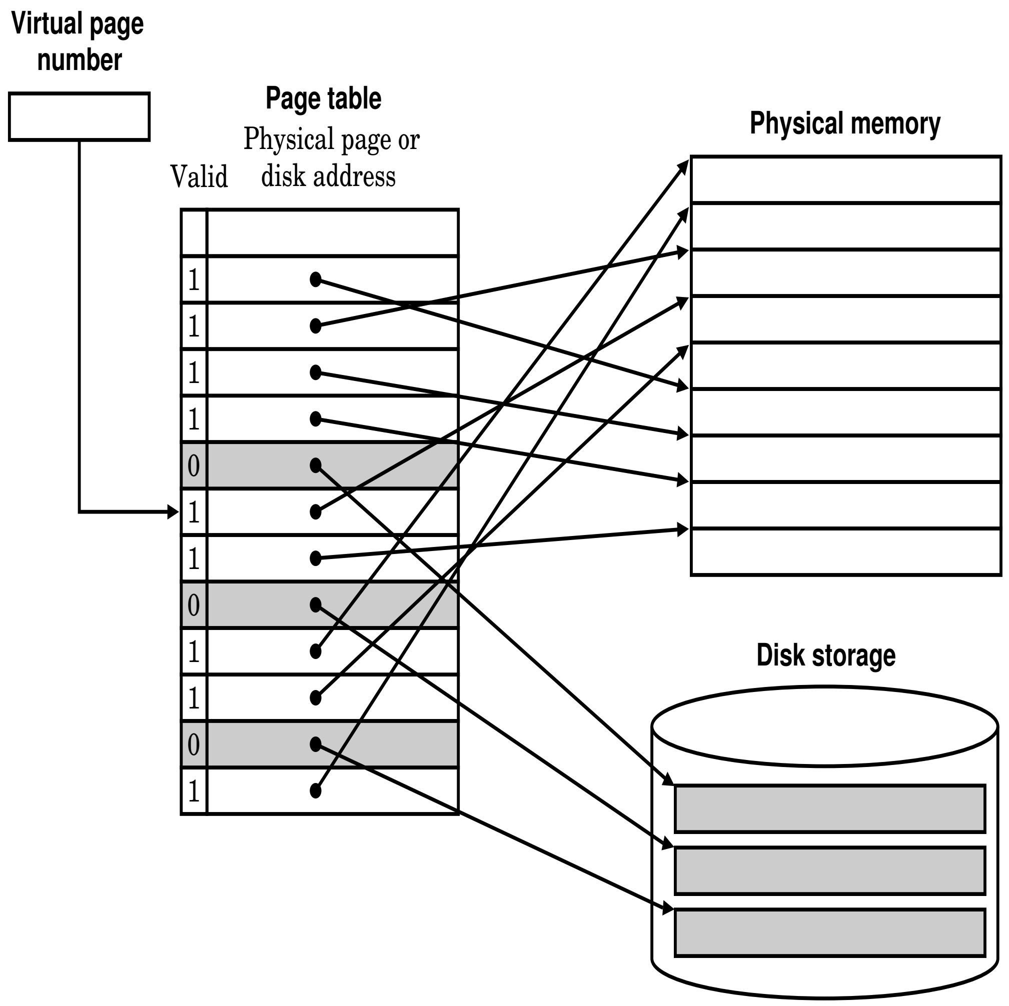
Virtual Addresses

A virtual address is split up into a page number and an offset.



Question: Why do it this way?

Pages and Virtual Addresses

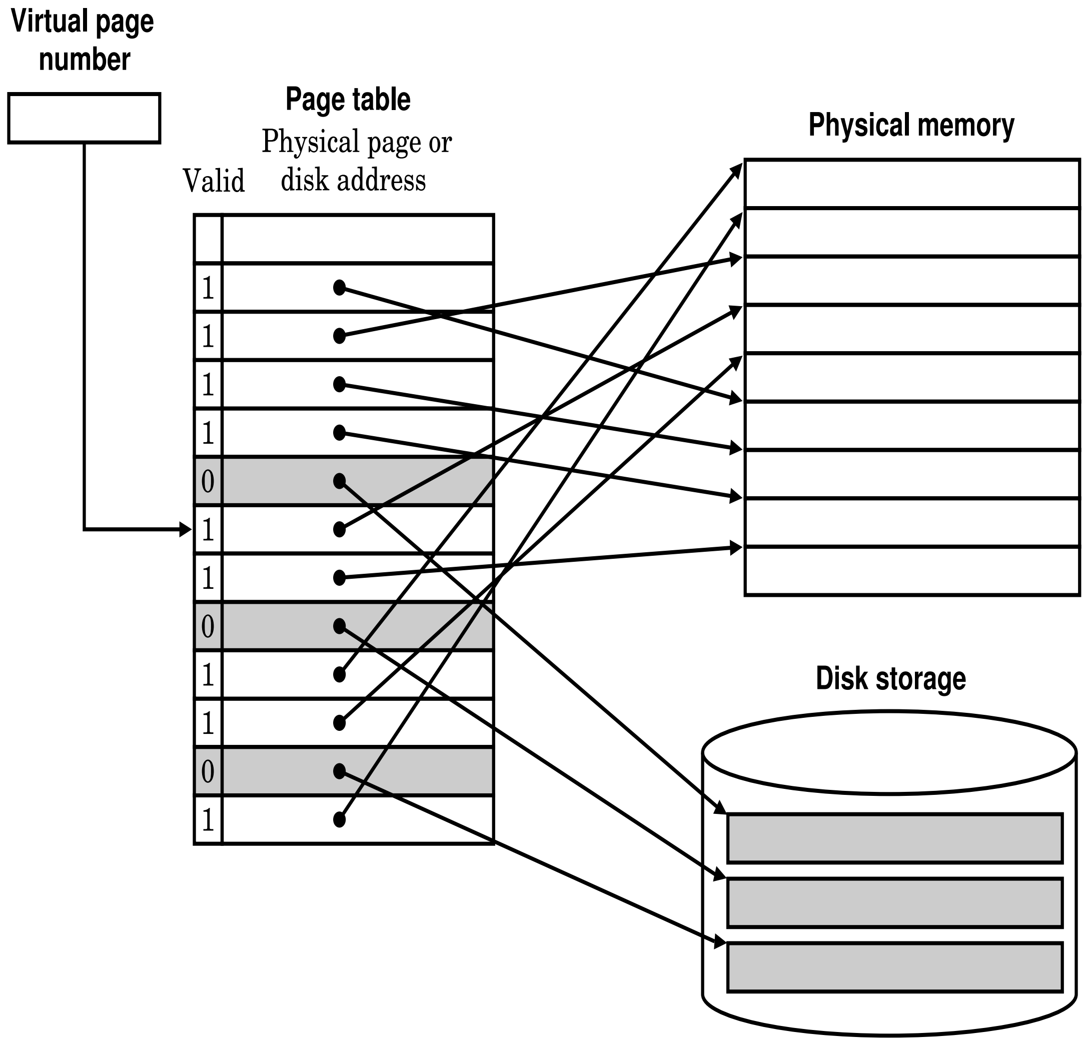


The page table records where each page is stored in memory or on disc.

Pages and Virtual memory, the story so far

- Split the memory up into pages.
- A virtual address has two parts, the page number and the offset in the page.
- Page table translates the virtual address into a physical address or tells you where the page lives on disc.

How do we know if the page is in memory?



The valid bits tells us quickly if the page is in memory.

Page Faults

- What happens if the request page is not in memory?
- We have to load the page in from the disc.
- What happens if the main memory is full?
- We have to chose an page in memory to swap out out to the disc and the load the page in from the disc the new page.

All this is handled by the operating system, this is because loading pages in from the disc takes a relatively long amount of time so the operating system has time to handle this.

Which pages are swapped out has a large impact on the performance of the operating system.

Write through or write back?

With Caches we had two choices on writing to a value in the cache:

Write through Update the item in the main memory at the same time you update the item in the cache;

Write Back Only update the information in the main memory when the copy in the cache gets over written.

What should we do in the virtual memory system and why?

emory protection, relocation and protection

- If each process has it own page table, then you can stop one process overwriting or reading the data of another process. How?
- Programs can be compiled to run at one address and don't need to be relocated when loaded. Why?
- Process can share memory blocks. How?

Large page tables

- Suppose a page is 2^{12} bytes and you have a virtual address space of 2^{32} bytes, how many pages are there?
- There would be
$$\frac{2^{32}}{2^{12}} = 2^{32-12} = 2^{22}$$
pages.
- Suppose each entry in the page table took up 4-bytes (not unreasonable) then the size of a page table for a process would be:
$$2^{22} * 2^2 = 2^{24} = 16\text{Meg}$$

That's a big page table.

arge page tables, what to do?

The are various solutions including

- use a dynamically growing page table guess an initial size and make it grow as needed.
- Hierarchical page tables
- Inverted page tables, store the list of physical pages and their corresponding virtual addresses then use a hash-function to find the entries.

This is not so bad since not all programs need 2^{32} bytes of addressable memory.

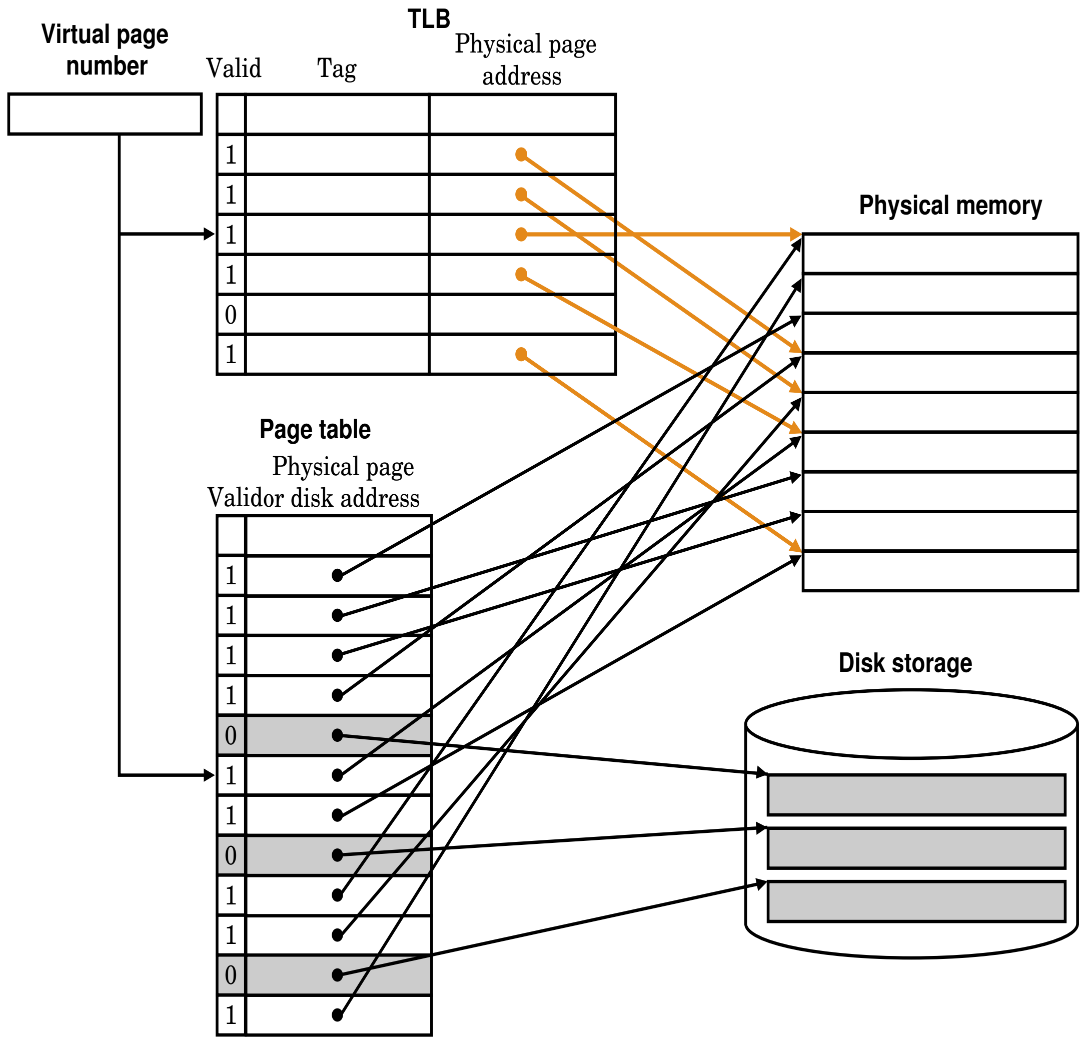
arge page tables, what to do?

- What ever scheme you use to optimise your page table you want to minimise the amount of time you spend translating virtual addresses to physical addresses.
- Remember every address request a process makes in a virtual address request. Processes can only make virtual address request they never have access to the physical addresses.

If virtual address translation is slow, this will be a real bottleneck.

TLBs - Making address translation fast

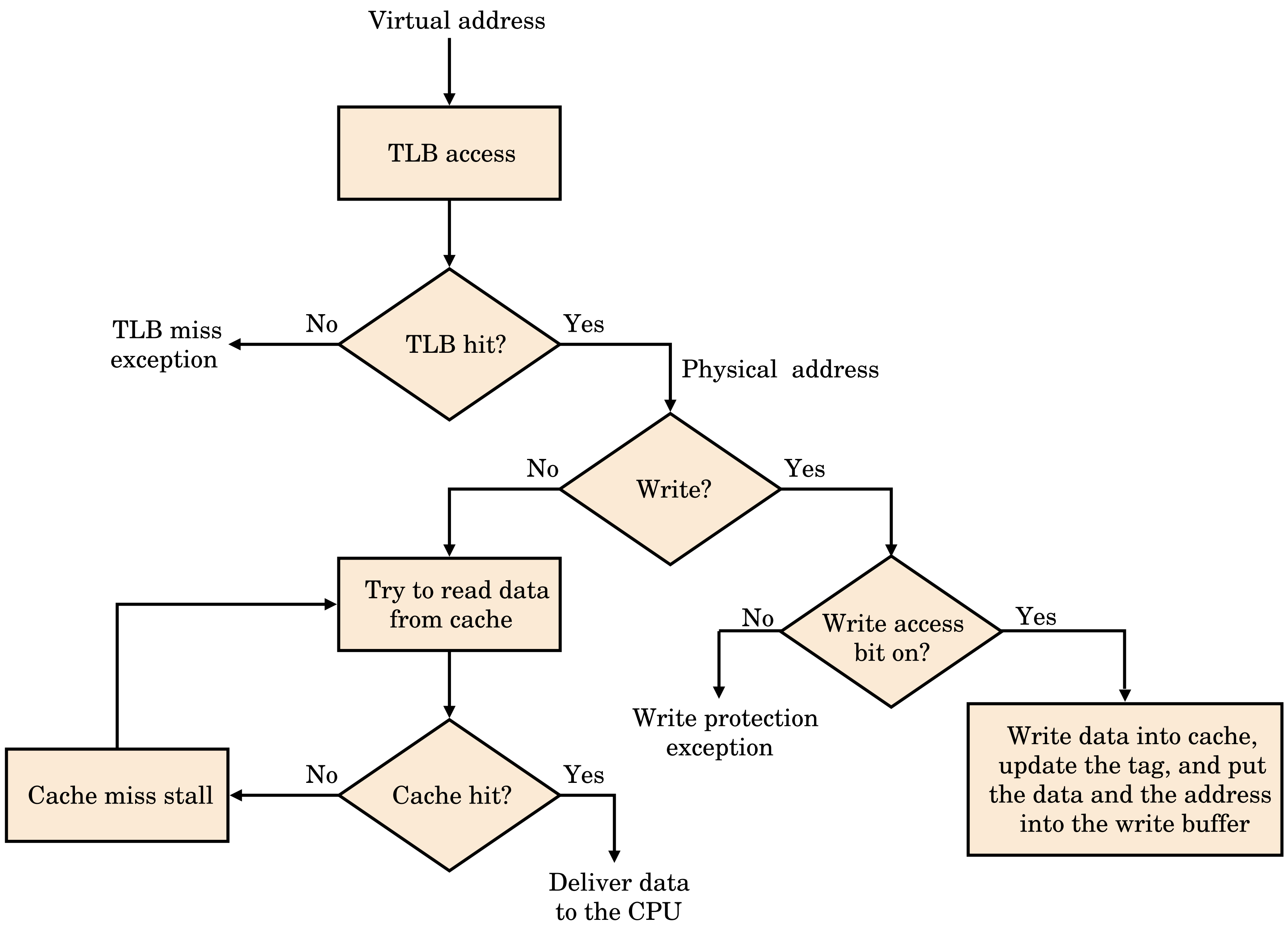
A TLB is like a cache for recently used translations.



TLBs

- If the item is not in the TLB then a lookup is performed in the page-table.
- The TLB is most often implemented in hardware.
- Question: What principle is the designer of a TLB relying on to get good performance?

What happens when a virtual address is requested?


Virtual address
TLB access
TLB hit?
No
TLB miss exception
Yes
Physical address
Write?
No
Yes
Try to read data from cache
Write access bit on?
No
Write protection exception
Yes
Write data into cache, update the tag, and put the data and the address into the write buffer
Cache hit?
No
Cache miss stall
Yes
Deliver data to the CPU