LECTURE 12

Virtual Memory

VIRTUAL MEMORY

Just as a cache can provide fast, easy access to recently-used code and data, main memory acts as a “cache” for magnetic disk.

The mechanism by which this is accomplished is known as *virtual memory*. There are two reasons for using virtual memory:

- Support multiple processes sharing the same memory space during execution.
- Allow programmers to develop applications without having to consider the size of memory available.

VIRTUAL MEMORY

Recall that the idea behind a cache is to exploit locality by keeping relevant data and instructions quickly-accessible and close to the processor.

Virtual memory uses a similar idea. Main memory only needs to contain the active, relevant portions of a program at a given time. This allows multiple programs to share main memory as they only use a subset of the space needed for the whole program.

VIRTUAL MEMORY

Virtual memory involves compiling each program into its own address space. This address space is accessible only to the program, and therefore protected from other programs.

Virtual addresses have to be translated into physical addresses in main memory.

VIRTUAL MEMORY TERMS

A lot of the concepts in virtual memory and caches are the same, but use different terminology.

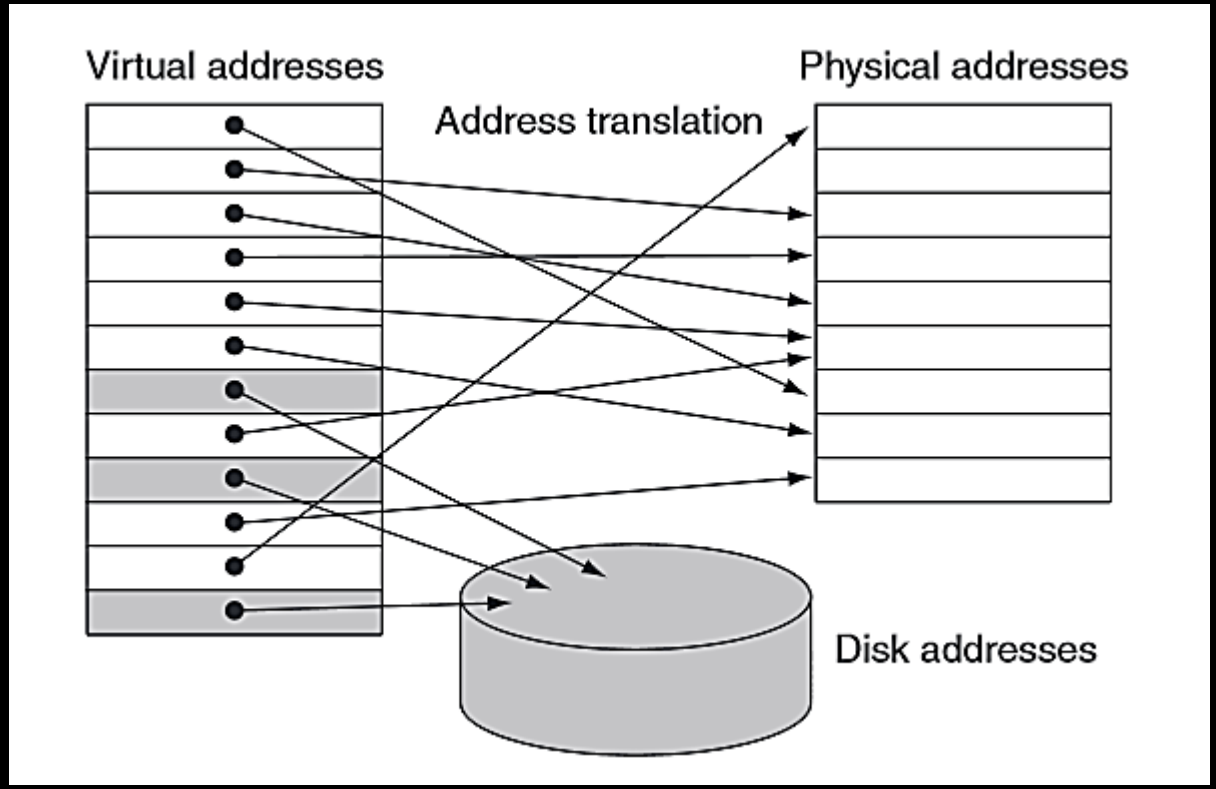
- Just as we have *blocks* in a cache, we have *pages* in virtual memory.
- A miss in virtual memory is known as a *page fault*.
- The processor produces a *virtual address*, which must be translated into a *physical address* in order to access main memory. This process is known as *address mapping* or *address translation*.

ADDRESS TRANSLATION

Pages are mapped from virtual addresses to physical addresses in main memory.

If a virtual page is not present in main memory, it must reside on disk.

A virtual address can only translate to one physical (or disk) address, but multiple virtual addresses may translate to the same physical (or disk) address.



ADDRESS TRANSLATION

The process of *relocation* simplifies the task of loading a program into main memory for execution.

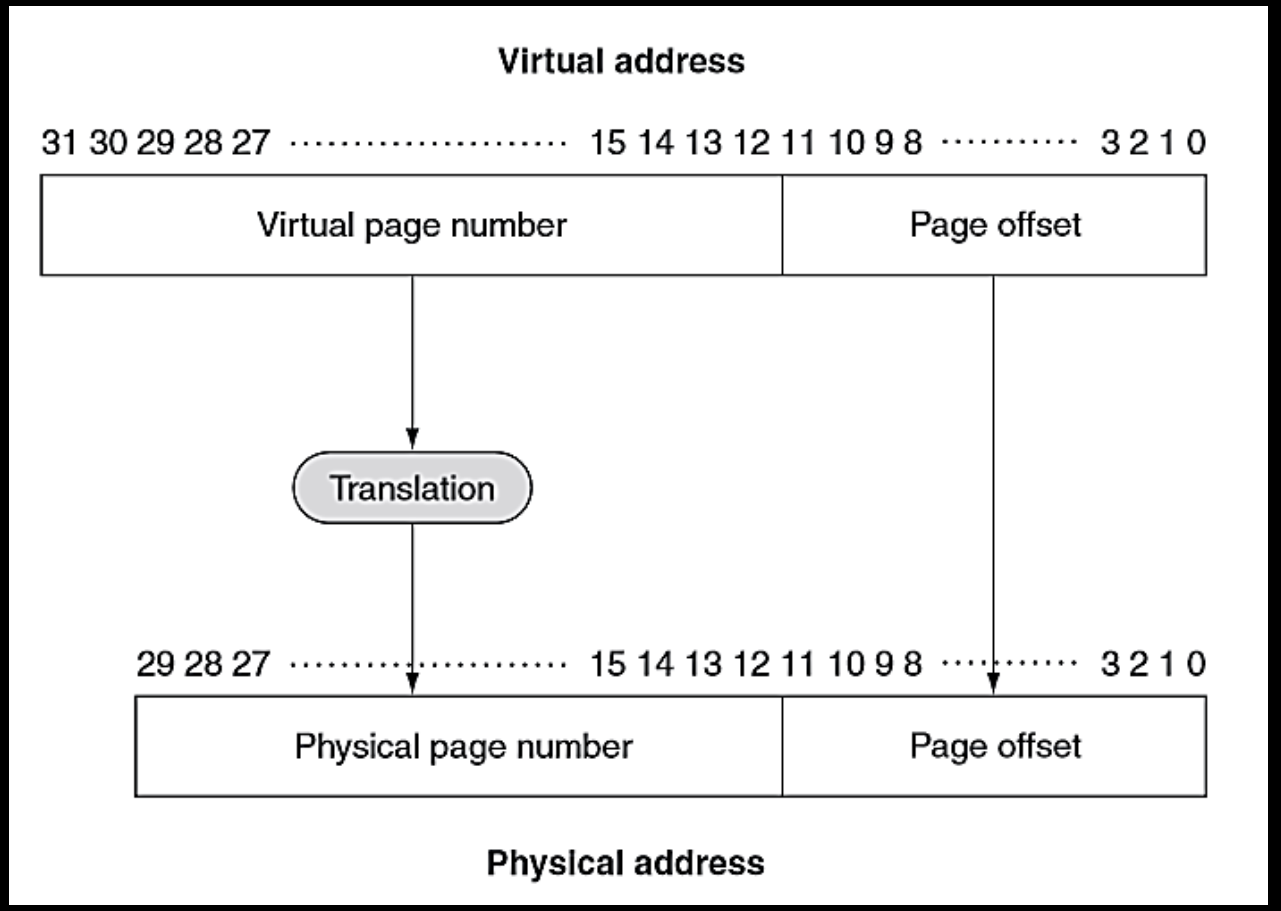
Relocation refers to mapping virtual memory to physical addresses before memory is accessed.

We can load the program anywhere in main memory and, because we perform relocation by page, we do not need to load the program into a contiguous block of memory – we only need to find a sufficient number of pages.

ADDRESS TRANSLATION

A virtual address is partitioned into a virtual page number and a page offset.

To translate a virtual address, we need only translate the virtual page number into the physical page number. The page offset remains the same.

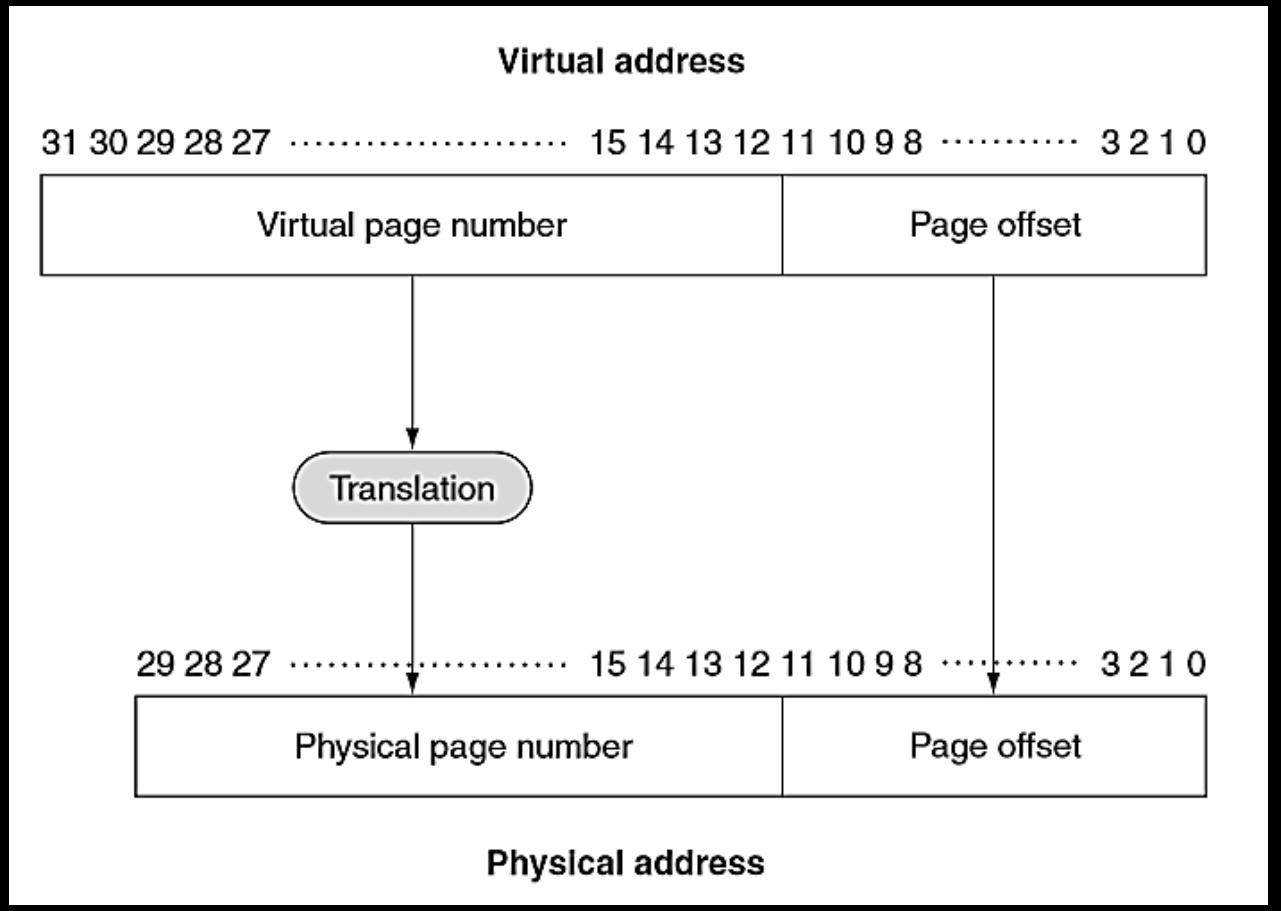


ADDRESS TRANSLATION

This example contains a 12-bit page offset. The page offset determines the page size. This page size is

$$2^{12} = 4096\,Bytes = 4KB$$

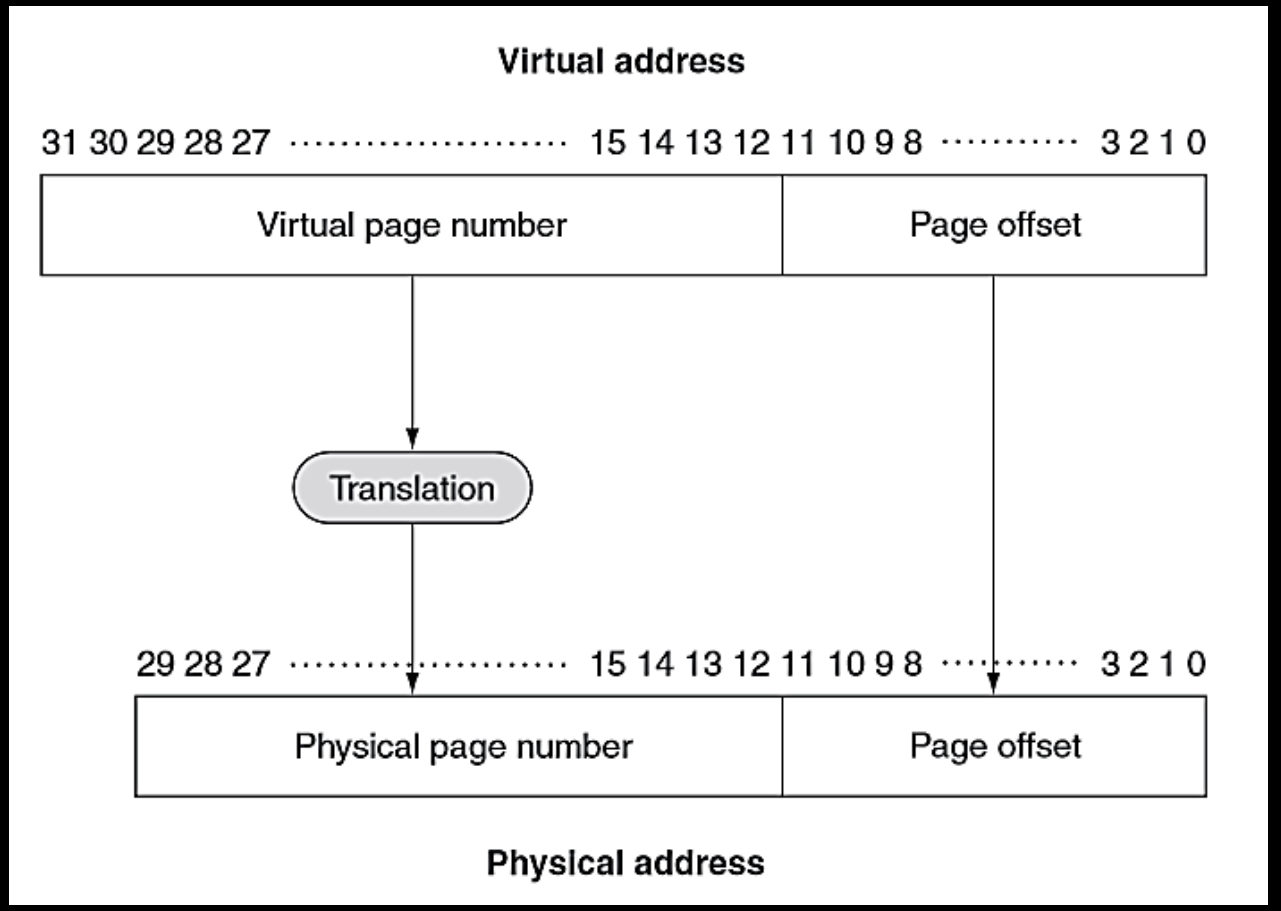
Note that the page number of the physical address has 18 bits, while the virtual page number has 20 bits. How many pages are allowed in physical memory? What about virtual memory?



ADDRESS TRANSLATION

There are 2^{18} pages allowed in main memory, while 2^{20} pages are allowed in virtual memory.

So, we have way more virtual memory pages than we do physical memory pages – this is alright, however, as we want to create the illusion of having a larger physical address space than we actually have.



DESIGN CHOICES

A lot of the design choices for virtual memory systems are driven by the high cost of a miss, also called a *page fault*. A page fault causes us to have to move a page from magnetic disk into main memory, which can take millions of clock cycles.

- Large page sizes are desirable in order to justify the large miss penalty. Page sizes of 32-64 KB are becoming the norm. x86-64 supports 4 KB, 2 MB, and 1 GB page sizes.
- Fully-associative placement of pages in main memory reduces conflict misses.
- Sophisticated miss and replacement policies are justified because a small reduction in the miss rate still creates a huge reduction in average access time.
- Write-back is always used because write-through takes too long.

PAGE TABLE

As we have discussed, the disadvantage of a fully-associative scheme is that finding an entry in main memory (or the cache) will be slow.

To facilitate the lookup of a page in main memory, we use a page table. A page table is indexed with the virtual page number and contains the corresponding physical page number.

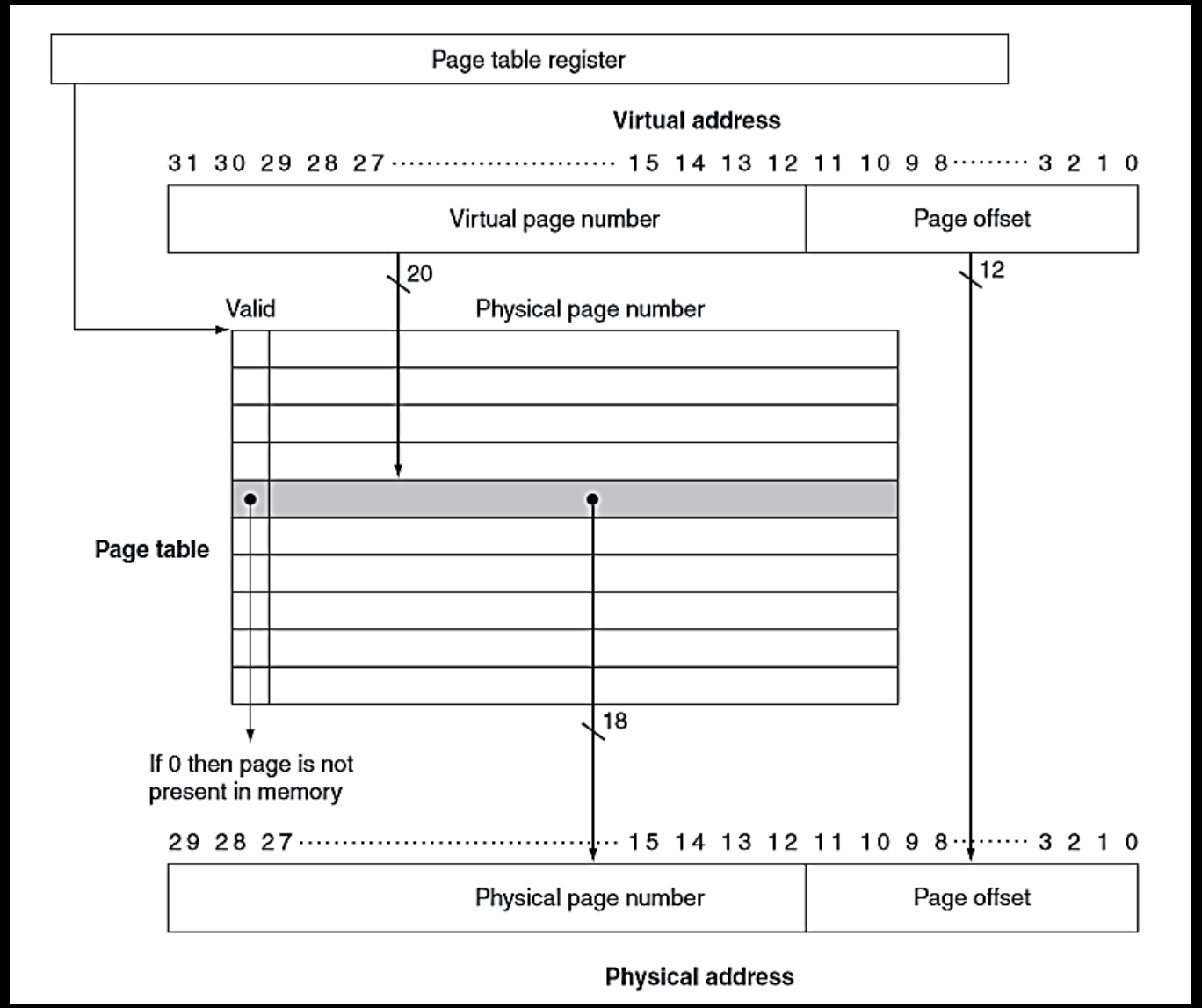
Every process has its own page table that maps its virtual address space to the physical address space.

PAGE TABLE

The address of the first entry in the page table is given by the *page table register*.

The valid bit indicates whether the mapping is legal or not – that is, whether the page is in main memory.

We do not need any notion of a tag because every virtual page number has its own entry.



PAGE FAULTS

When page table entry has a de-asserted valid bit, a page fault occurs.

Since the page must be retrieved from magnetic disk on a page fault, we must also find a way to associate virtual pages with a disk page.

The operating system creates enough space on disk for all of the pages of a process (called a *swap space*) and, at the same time, creates a structure to record the associated disk page for each virtual page.

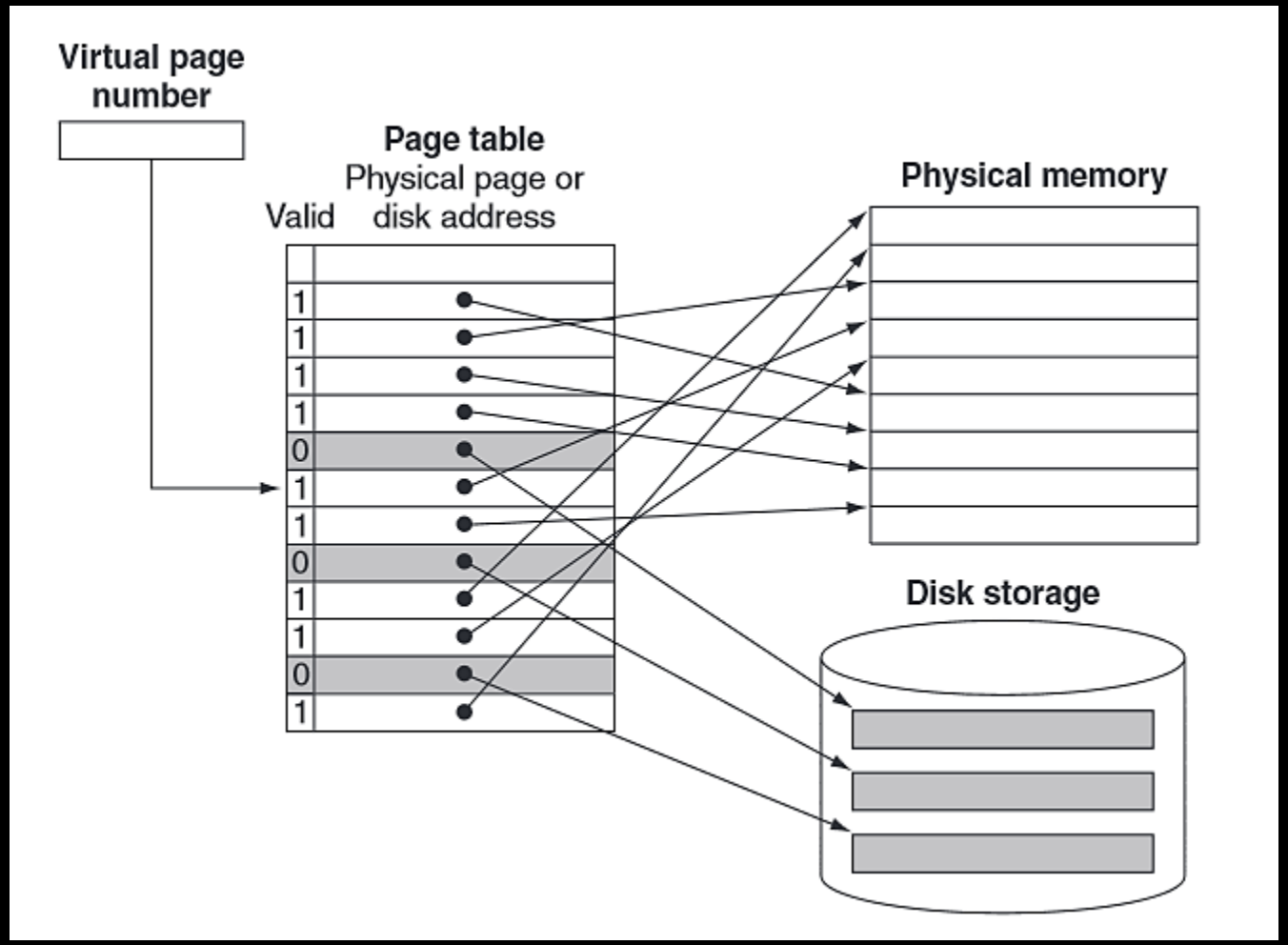
This secondary table may be combined with the page table or left separate.

PAGE FAULTS

Here we have a single table holding physical page numbers or disk addresses.

If the valid bit is on, the entry is a physical page number. Otherwise, it's a disk address.

In reality, these tables are usually separate – we must keep track of the disk address for all virtual pages.

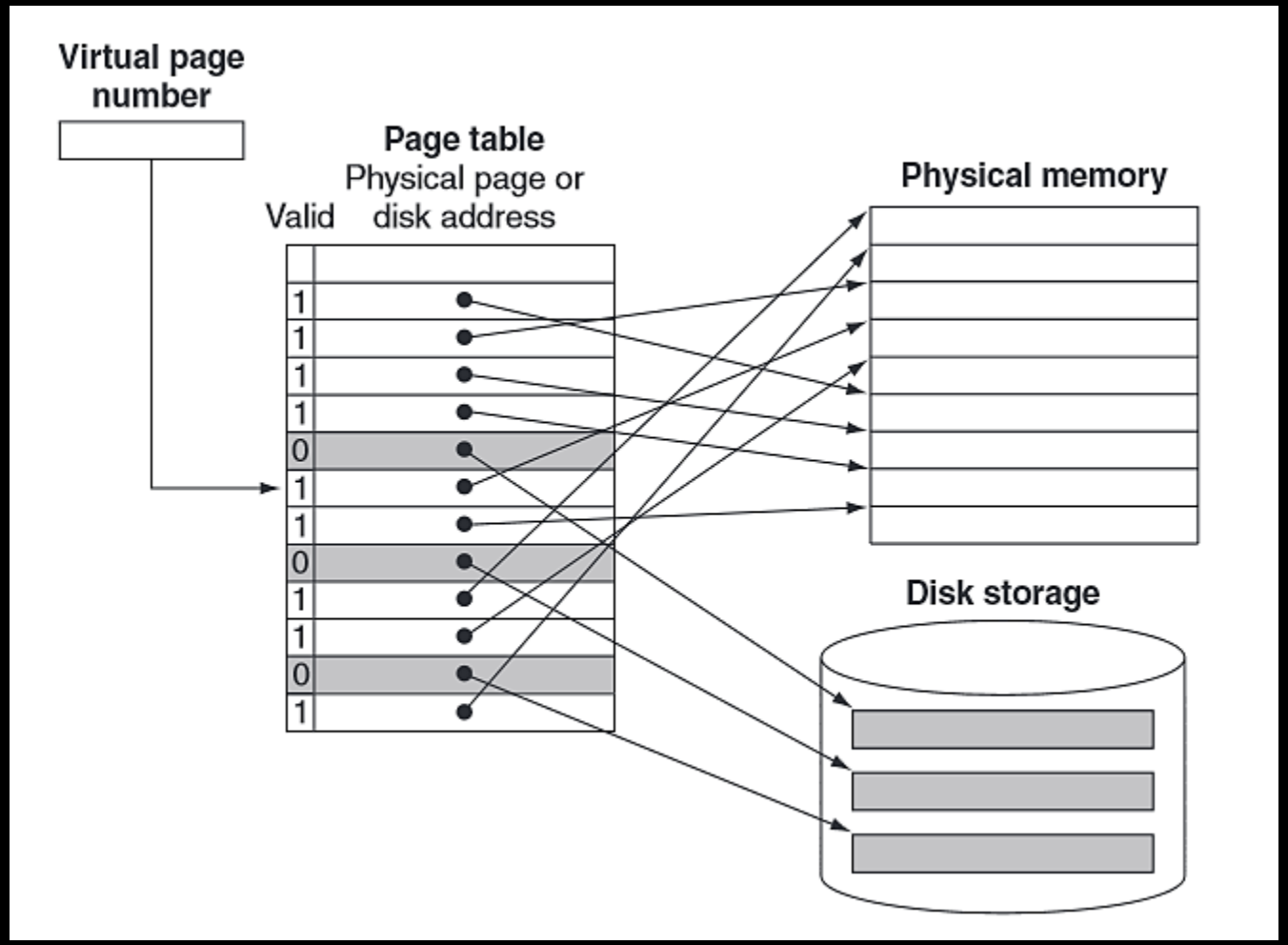


PAGE FAULTS

When a page fault occurs, we must choose a physical page to replace using LRU.

If the evicted page is dirty, it must be written back to disk.

Finally, the desired page is read into main memory and the page tables are updated.



TRANSLATION LOOKASIDE BUFFERS

Because the page table for a process is itself stored in main memory, any access to main memory involves at least two references: one to get the physical address and the other to get the data.

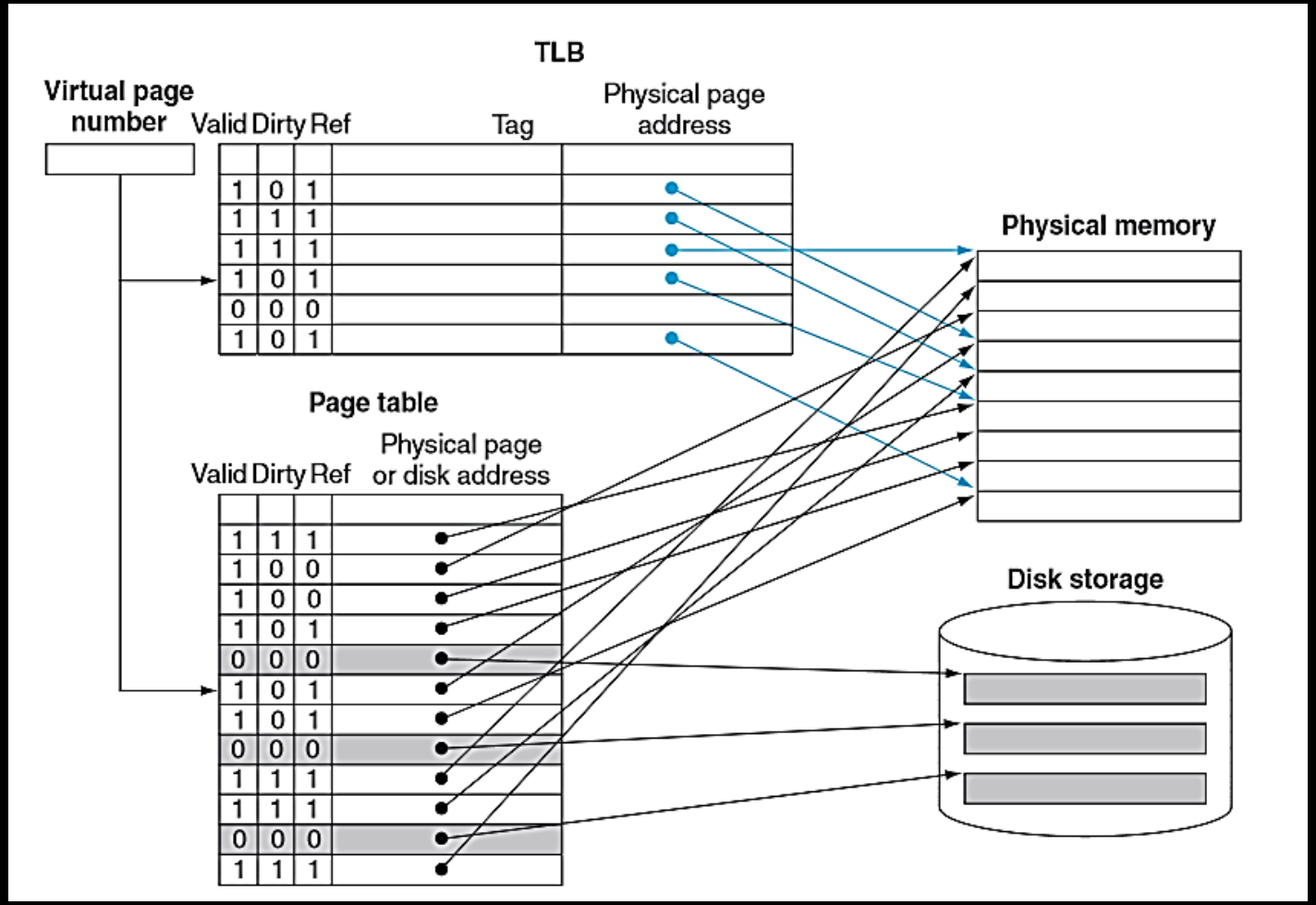
To avoid this, we can exploit both spatial and temporal locality by creating a special cache for storing recently used translations, typically called a *translation lookaside buffer*.

Some portion of the virtual page number is used to index into the TLB. A tag is used to verify that the physical address entry is relevant to the reference being made.

TLB

The TLB contains a subset of the virtual-to-physical page mappings in the page table.

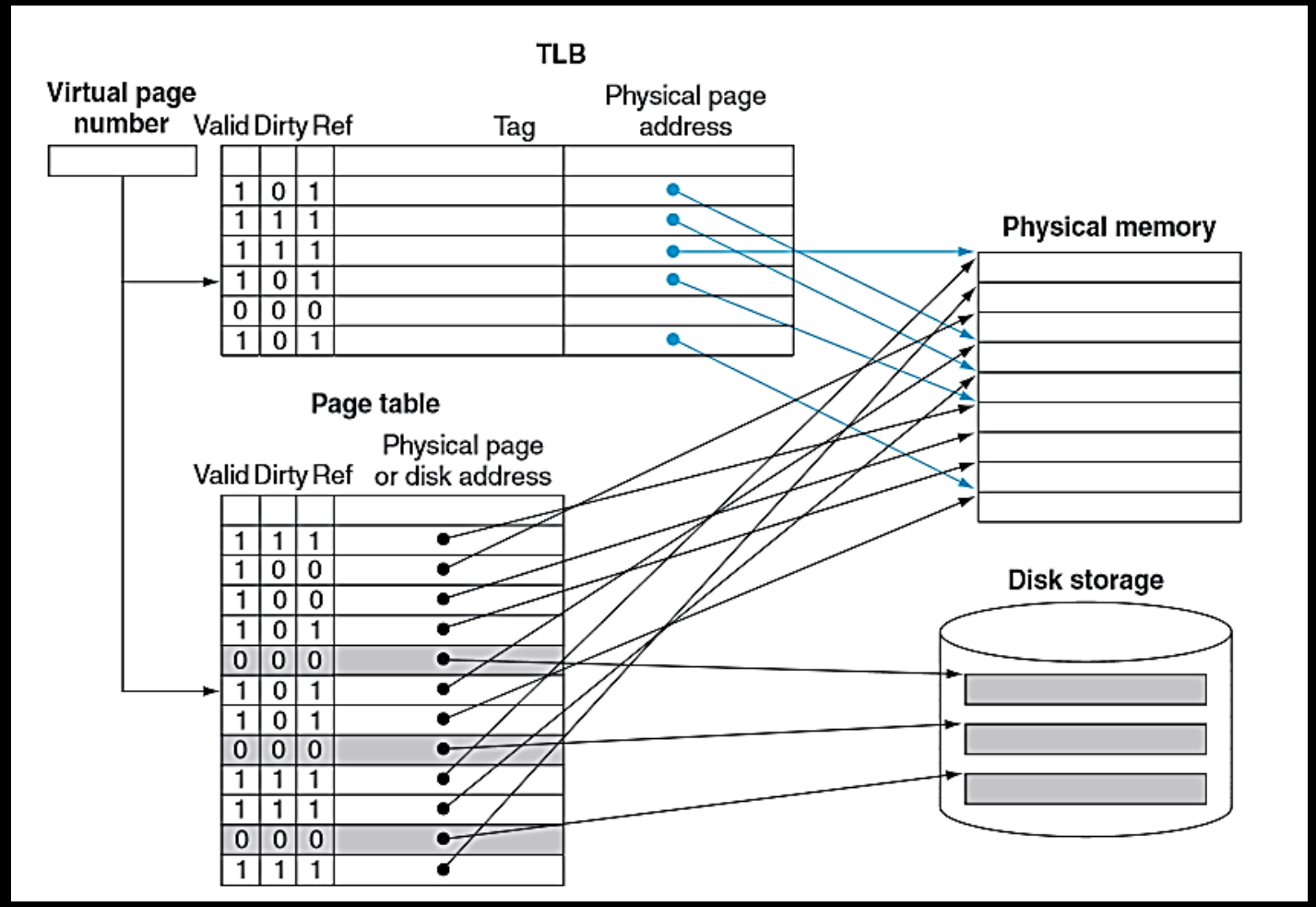
Because not every virtual address has its own entry in the TLB, we index into the TLB using a lower portion of the virtual page number and check the tag against the higher portion.



TLB

If there is no entry in the TLB for a virtual page number, we must check the page table. A TLB miss does not necessarily indicate a page fault.

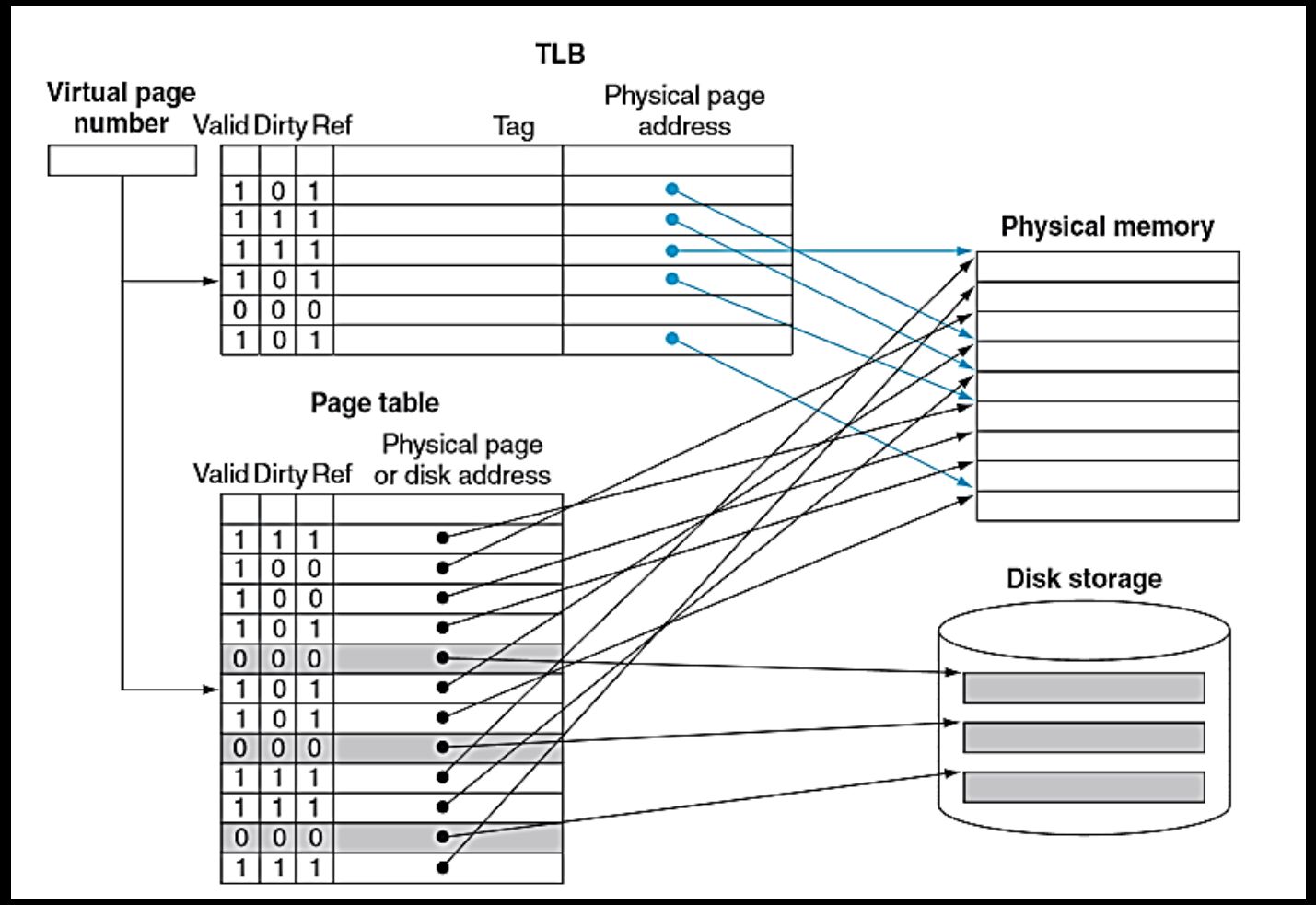
The page table will either provide a physical address in main memory or indicate that the page is on disk, which results in a page fault.



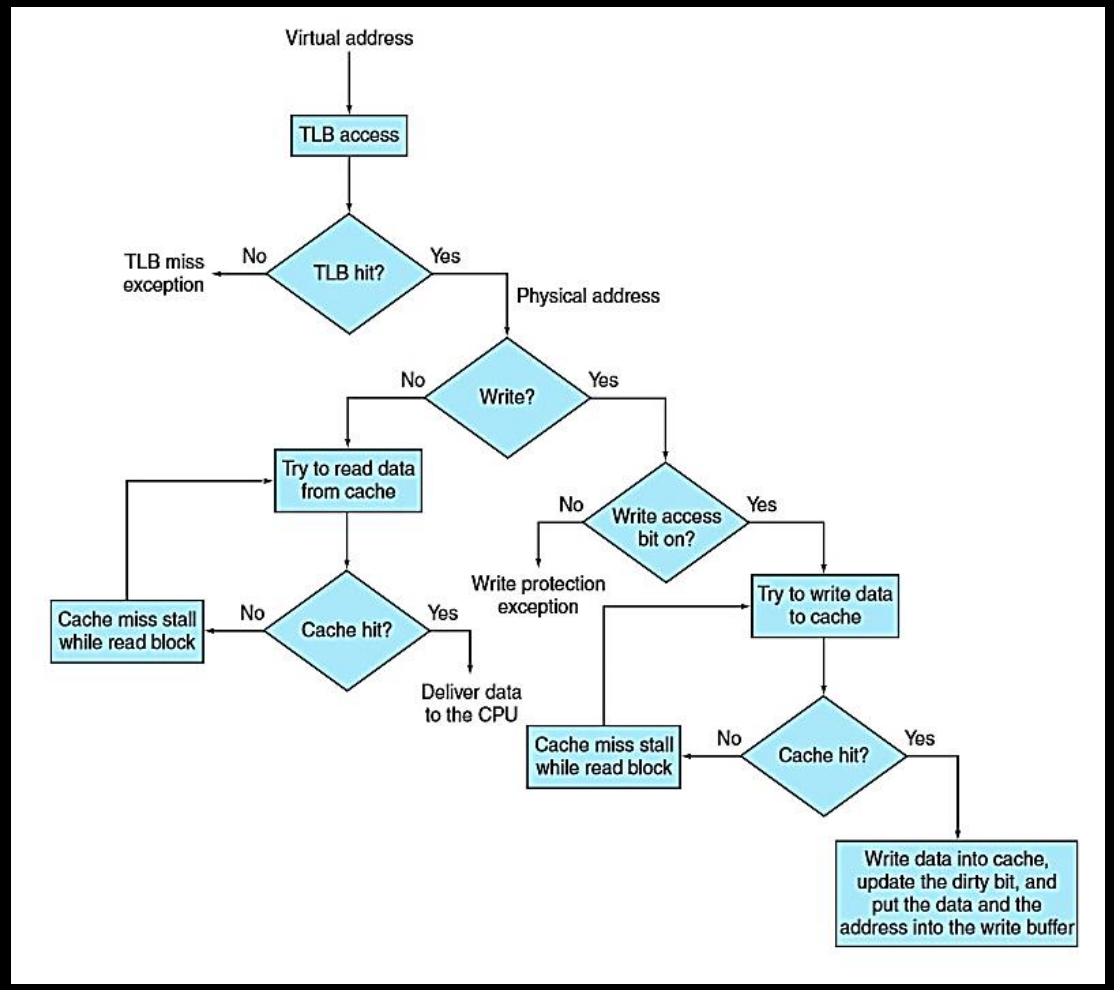
TLB

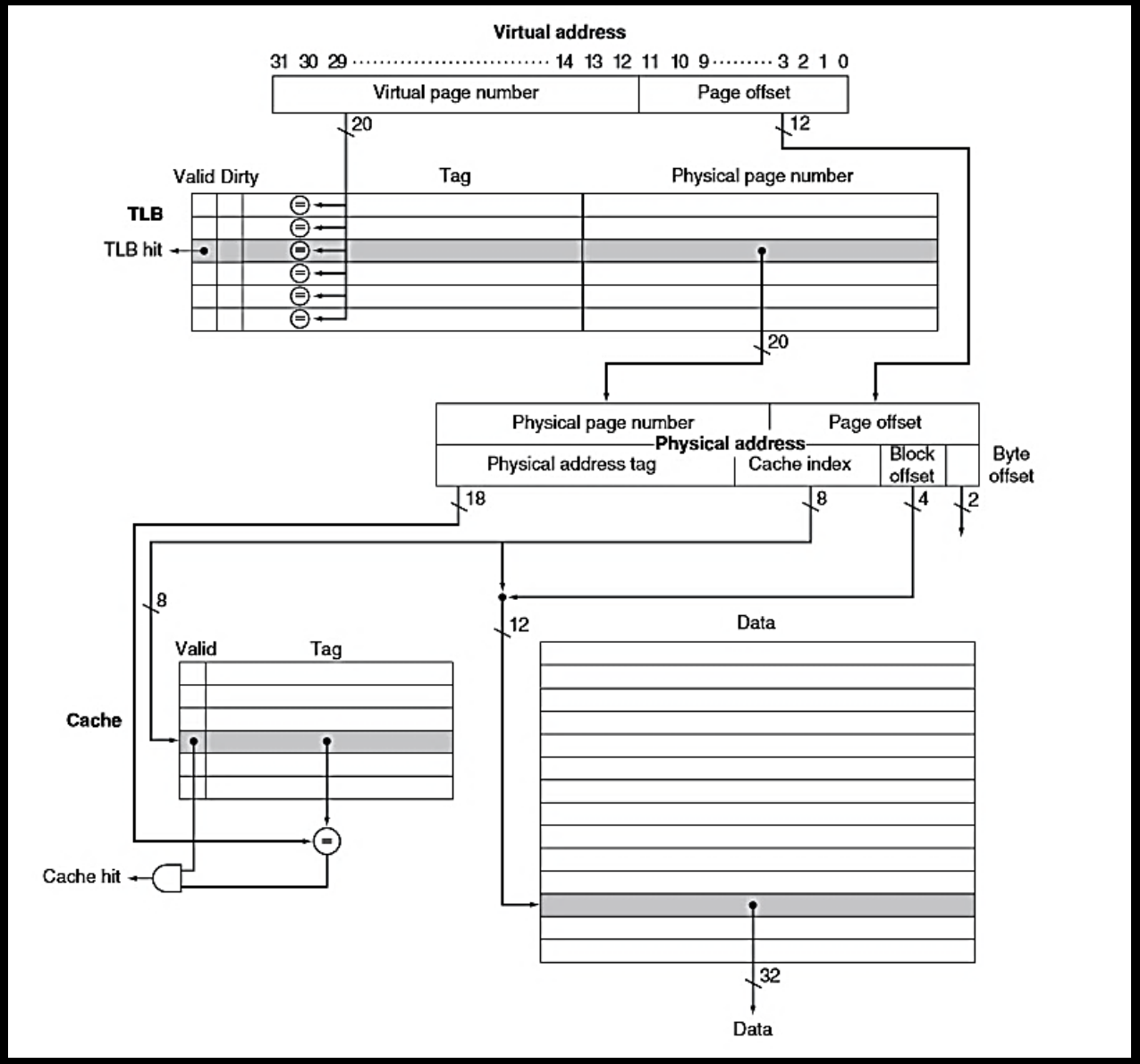
Note the three reference bits:

- Valid – the entry in the TLB or page table is legitimate.
- Dirty – the page has been written and is inconsistent with disk. Will need to be written back upon replacement.
- Reference – a bit indicating the entry has been recently used. Periodically, all reference bits are cleared.



INTRINSITY FASTMATH





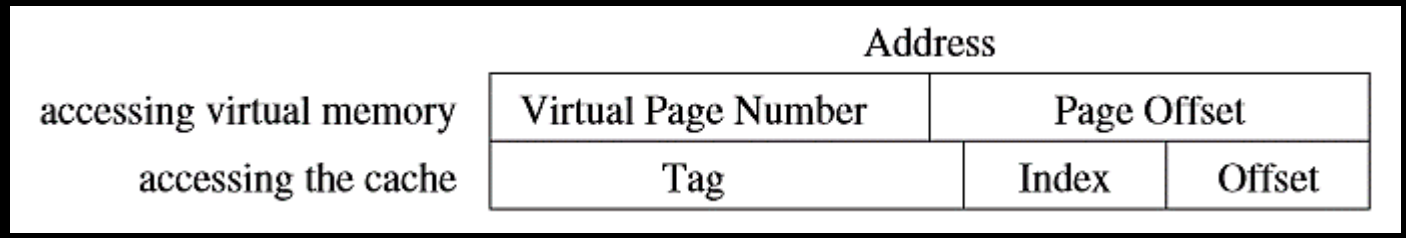
VIRTUAL OR PHYSICAL?

There's something significant to notice about the Intrinsity example: the physical address is used to index into the cache, not the virtual address. So, which should we use? The virtual address or physical address? The answer is that it depends.

- *Physically indexed, physically tagged* (PIPT) caches use the physical address for both the index and the tag.
  - Simple to implement but slow, as the physical address must be looked up (which could involve a TLB miss and access to main memory) before that address can be looked up in the cache.
- *Virtually indexed, virtually tagged* (VIVT) caches use the virtual address for both the index and the tag.
  - Potentially much faster lookups.
  - Problems when several different virtual addresses may refer to the same physical address -- addresses would be cached separately despite referring to the same memory, causing coherency problems.
  - Additionally, there is a problem that virtual-to-physical mappings can change, which would require clearing cache blocks.
- *Virtually indexed, physically tagged* (VIPT) caches use the virtual address for the index and the physical address in the tag.

VIRTUALLY INDEXED, PHYSICALLY TAGGED

- Index into the cache using bits from the page offset.
- Do the tag comparison after obtaining the physical page number.
- Advantage is that the access to the data in the cache can start sooner.
- Limitation is that one size of a VIPT cache can be no larger than the page size.



VIRTUAL MEMORY, TLB, AND CACHE

TLB	Page table	Cache	Possible? If so, under what circumstance?
Hit	Hit	Miss	Possible, although the page table is never really checked if TLB hits.
Miss	Hit	Hit	TLB misses, but entry found in page table; after retry, data is found in cache.
Miss	Hit	Miss	TLB misses, but entry found in page table; after retry, data misses in cache.
Miss	Miss	Miss	TLB misses and is followed by a page fault; after retry, data must miss in cache.
Hit	Miss	Miss	Impossible: cannot have a translation in TLB if page is not present in memory.
Hit	Miss	Hit	Impossible: cannot have a translation in TLB if page is not present in memory.
Miss	Miss	Hit	Impossible: data cannot be allowed in cache if the page is not in memory.

TLB AND PAGE TABLE EXAMPLE

Page size is 512 bytes. TLB is direct-mapped and has 64 sets.

Given virtual address 36831 (1000 1111 1101 1111), what is the physical address?

Given virtual address 4319 (1 0000 1101 1111), what is the physical address?

TLB

Index	Page	Tag	Valid
0	?	?	?
…	…	…	…
7	?	?	0
8	2	0	1
…	…	…	…
63	?	?	?

Page Table

Index	Page	Res	Dirty	Disk Addr	…
0	?	?	?	?	…
…	…	…	…	…	…
71	9	Yes	?	?	…
..	…	…	…	…	…

TLB AND PAGE TABLE EXAMPLE

Page size is 512 bytes. TLB is direct-mapped and has 64 sets.

Given virtual address 36831, what is the physical address? 5087
Given virtual address 4319, what is the physical address? 1247

TLB

Index	Page	Tag	Valid
0	?	?	?
…	…	…	…
7	?	?	0
8	2	0	1
…	…	…	…
63	?	?	?

Page Table

Index	Page	Res	Dirty	Disk Addr	…
0	?	?	?	?	…
…	…	…	…	…	…
71	9	Yes	?	?	…
..	…	…	…	…	…

MULTIPROGRAMMING

- *Multiprogramming* is the ability of several processes to concurrently share a computer.
- A *process* is the code, data, and any state information used in the execution of a program.
- A *context switch* means transferring control of the machine from one process to another.
- On a context switch, we must change the page table register to point to the appropriate page table and either:
  - the TLB must be cleared (could be very inefficient!)
  - or tags in the TLB must be extended to include a PID (Process IDentifier).

PROCESS PROTECTION

- Proper protection must be provided to prevent a process from inappropriately affecting another process or itself.
- Page tables are kept in the address space of the operating system.
- Virtual memory systems keep bits in the page table and TLB entries to indicate the type of access that the process has to each of the pages.
- Two modes of execution are supported:
  - User mode is used by the process.
  - Kernel mode is used by the operating system. This mode can be used for updating page tables and accomplishing I/O.
- A switch from user to kernel mode is accomplished by a system call.
- A switch from kernel to user mode is accomplished by returning from an exception (system call).

DESIGN PARAMETERS

Below are typical design parameters for different memory hierarchy levels in 2012.

Server processors also have L3 caches, which are not shown.

Feature	Typical values for L1 caches	Typical values for L2 caches	Typical values for paged memory	Typical values for a TLB
Total size in blocks	250–2000	2500–25,000	16,000–250,000	40–1024
Total size in kilobytes	16–64	125–2000	1,000,000–1,000,000,000	0.25–16
Block size in bytes	16–64	64–128	4000–64,000	4–32
Miss penalty in clocks	10–25	100–1000	10,000,000–100,000,000	10–1000
Miss rates (global for L2)	2%–5%	0.1%–2%	0.00001%–0.0001%	0.01%–2%

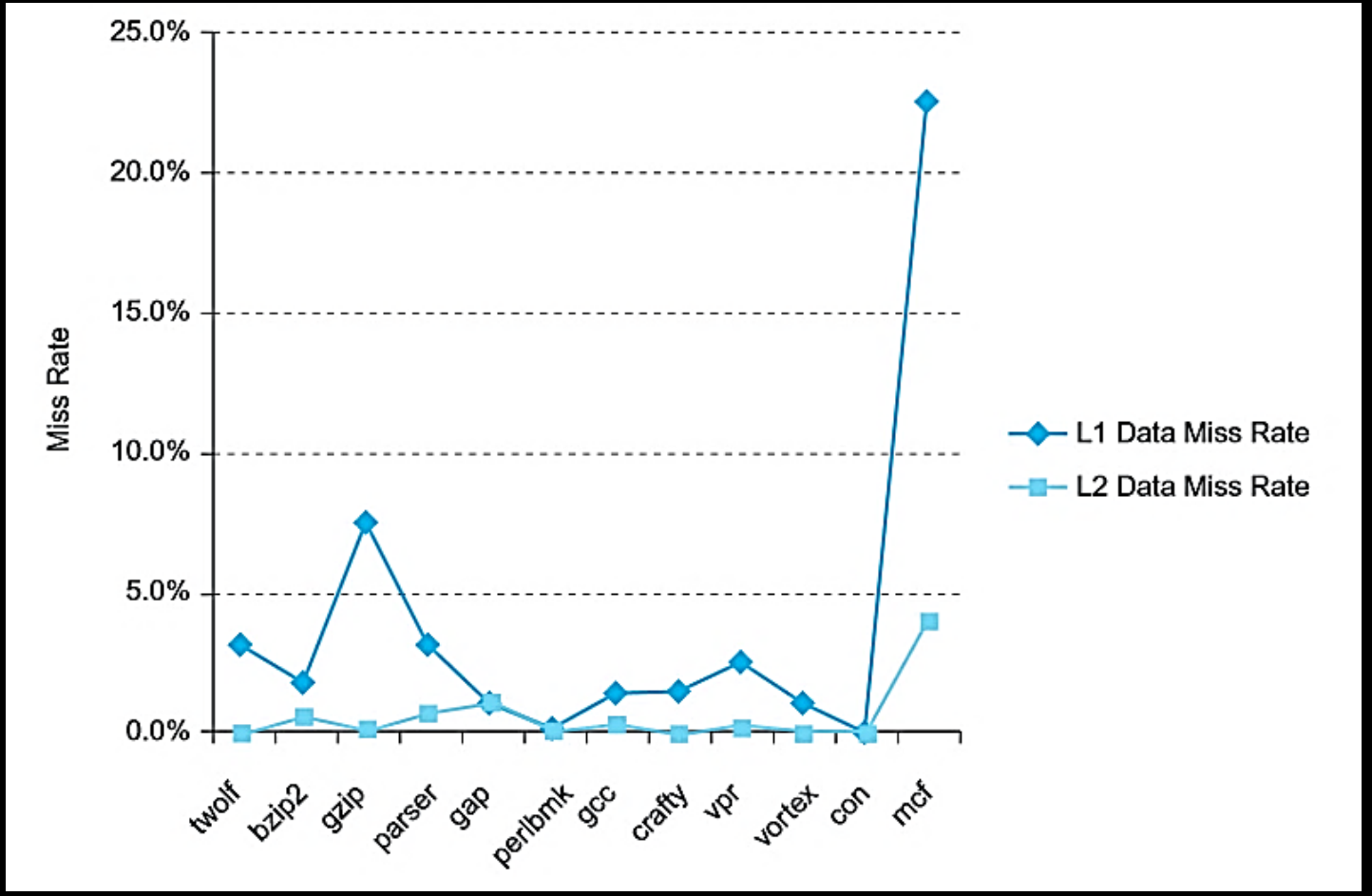
ADDRESS TRANSLATION HARDWARE

Characteristic	ARM Cortex-A8	Intel Core i7
Virtual address	32 bits	48 bits
Physical address	32 bits	44 bits
Page size	Variable: 4, 16, 64 KiB, 1, 16 MiB	Variable: 4 KiB, 2/4 MiB
TLB organization	1 TLB for instructions and 1 TLB for data Both TLBs are fully associative, with 32 entries, round robin replacement TLB misses handled in hardware	1 TLB for instructions and 1 TLB for data per core Both L1 TLBs are four-way set associative, LRU replacement L1 I-TLB has 128 entries for small pages, 7 per thread for large pages L1 D-TLB has 64 entries for small pages, 32 for large pages The L2 TLB is four-way set associative, LRU replacement The L2 TLB has 512 entries TLB misses handled in hardware

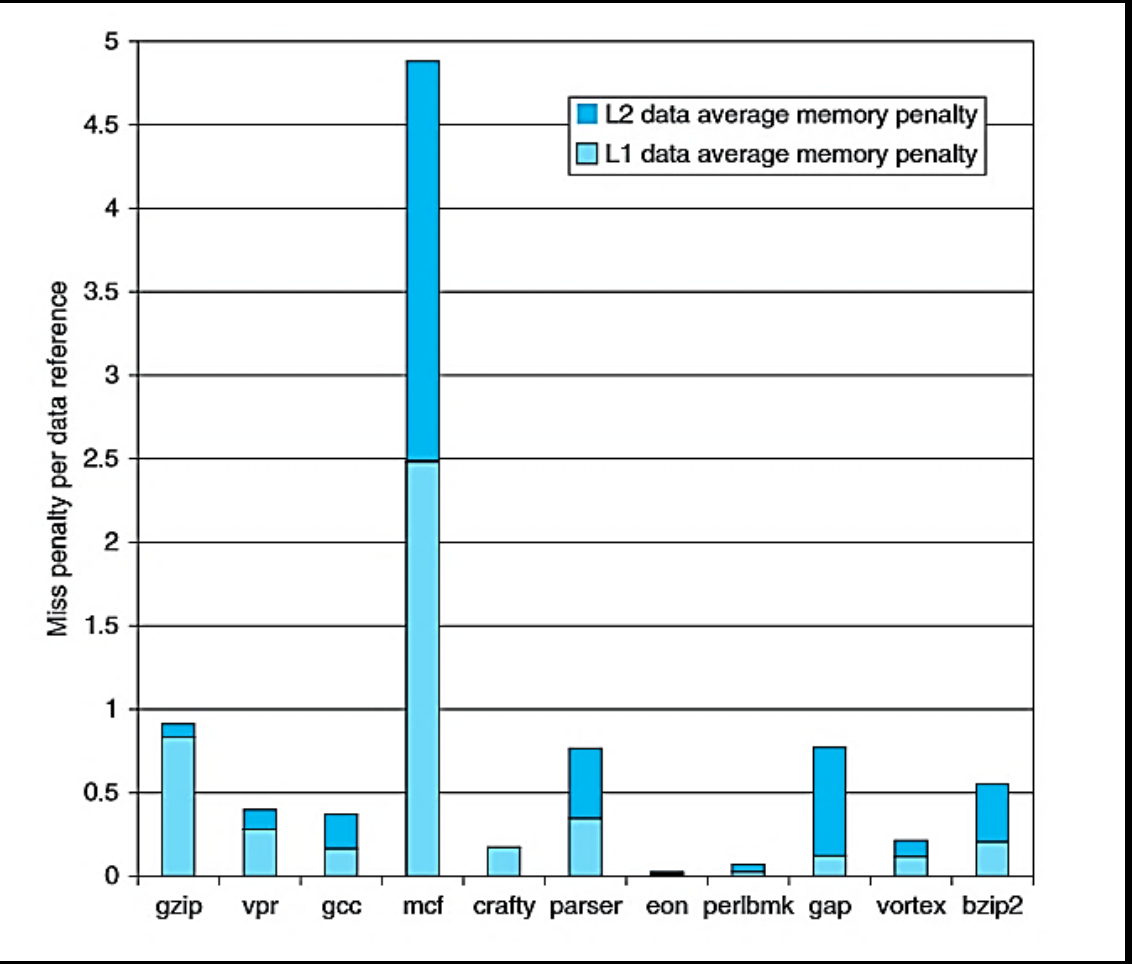
EXAMPLE CACHES

Characteristic	ARM Cortex-A8	Intel Nehalem
L1 cache organization	Split instruction and data caches	Split instruction and data caches
L1 cache size	32 KiB each for instructions/data	32 KiB each for instructions/data per core
L1 cache associativity	4-way (I), 4-way (D) set associative	4-way (I), 8-way (D) set associative
L1 replacement	Random	Approximated LRU
L1 block size	64 bytes	64 bytes
L1 write policy	Write-back, Write-allocate(?)	Write-back, No-write-allocate
L1 hit time (load-use)	1 clock cycle	4 clock cycles, pipelined
L2 cache organization	Unified (instruction and data)	Unified (instruction and data) per core
L2 cache size	128 KiB to 1 MiB	256 KiB (0.25 MiB)
L2 cache associativity	8-way set associative	8-way set associative
L2 replacement	Random(?)	Approximated LRU
L2 block size	64 bytes	64 bytes
L2 write policy	Write-back, Write-allocate (?)	Write-back, Write-allocate
L2 hit time	11 clock cycles	10 clock cycles
L3 cache organization	–	Unified (instruction and data)
L3 cache size	–	8 MiB, shared
L3 cache associativity	–	16-way set associative
L3 replacement	–	Approximated LRU
L3 block size	–	64 bytes
L3 write policy	–	Write-back, Write-allocate
L3 hit time	–	35 clock cycles

DATA CACHE MISS RATES FOR THE ARM CORTEX-A8



AVG MEM ACCESS PENALTY PER ARM DATA REF



CACHE MISS RATES FOR THE INTEL I7 920