# Translation buffers

To translate a virtual address into a physical address, we need to look up a page table.

A translation for one memory address requires another memory reference. Without special hardware, there is a slowdown factor of 2 at the very least.

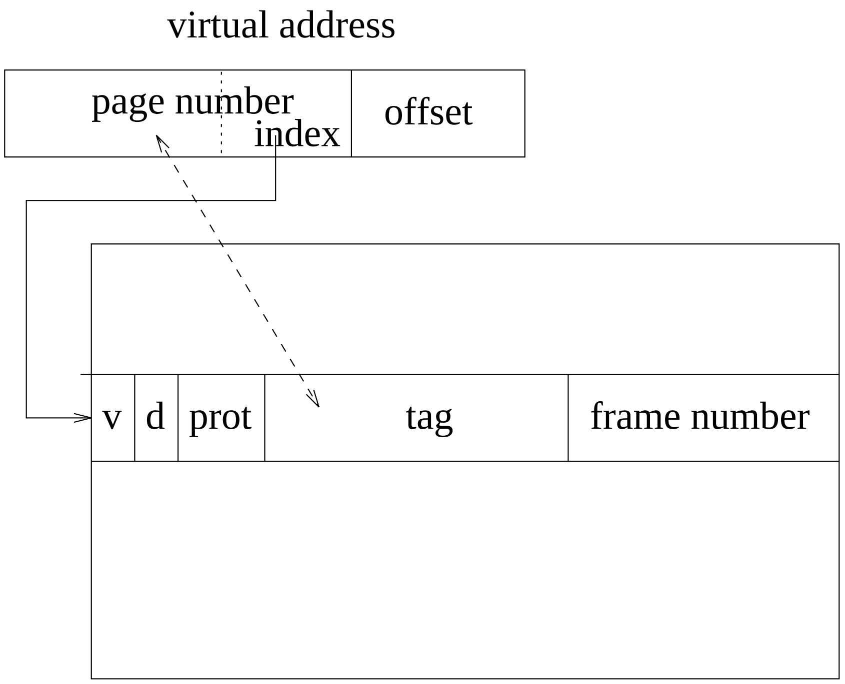
Hence the need for “caching” the page tables: this is done via dedicated “caches” called *translation buffers* (sometimes called *translation lookaside buffers* or TLB’s)

# Translation buffer Organizations

TLB's are organized as caches.

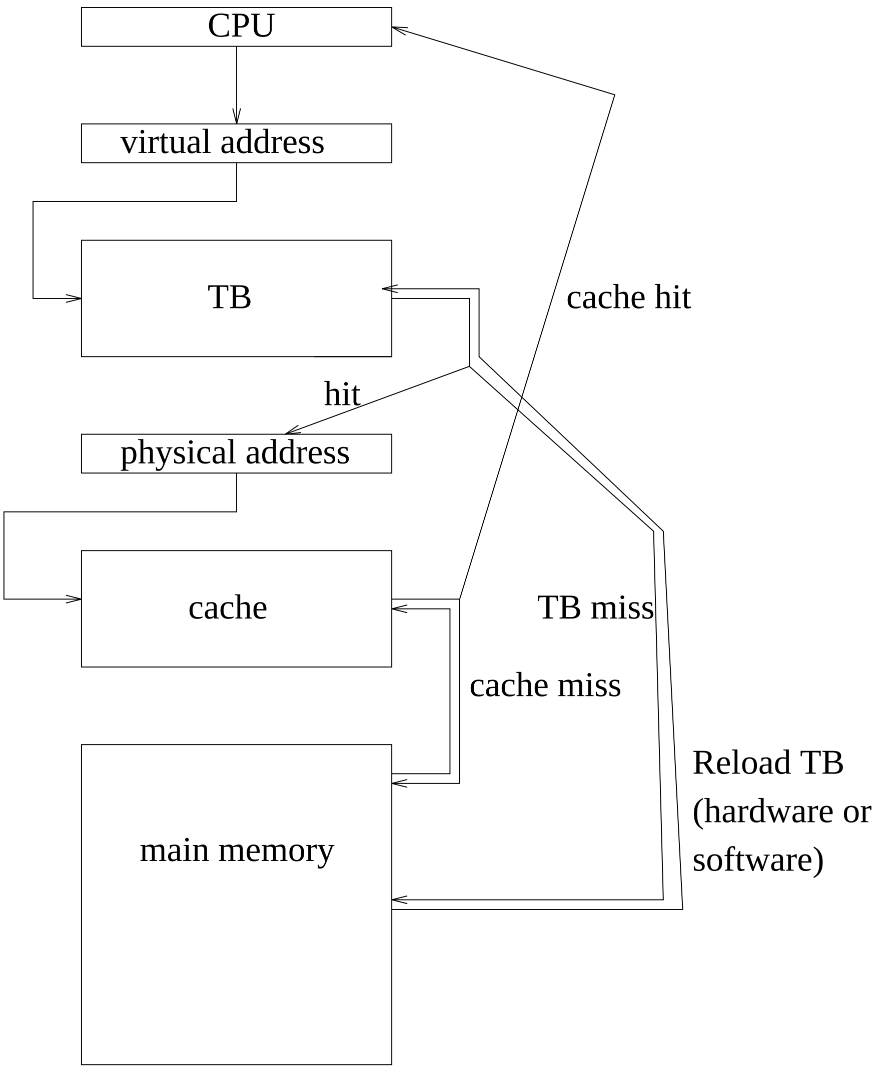
An entry in the TLB has a tag (similar to a cache tag but of course not necessarily of the same length) and information (data) which is the contents of a page table entry (PTE) (instead of the "real" data found in a cache entry).

TLB's are small – from 8 to 128 entries, can be fullly-associative (for small TLB's) or direct-mapped. There can be unified, or we can have ITLB's and D-TLB's, or user TLB's and system TLB's.



# Memory access (global picture)

This picture assumes that the page being referenced is in main memory, i.e., no page fault



# Translation from virtual to physical address

On each memory reference, the hardware searches the TLB for the translation.

On a TLB hit, the physical address is passed to the cache

On a TLB miss, either the hardware or the software (this depends on implementations) searches the page table of the process. If the PTE is valid, its contents are loaded in the TLB and the address translation resumes.

Note that in the case of a software implementation, the TLB miss resolution is fast - of the order of 10's of instructions.

In either case, there is no need for a context-switch.

# TLB Management

TLB's are organized as caches.

If small, they are often fully-associative (but not the usual case today).

The current trend is for having two TLB's, one for Instructions and one for Data.

TLB's are write-back. The only things that can change are the dirty bit and any other information that might be needed for the page replacement algorithm (see CSE451).

What happens at context-switch? Virtual page translations are no longer valid for the new task. Two possible actions:

- Flush the TLB when moving from one program to another
- Append a Process Id Number (PID) to the virtual address. At context-switch, the O.S. loads a special register with the PID of the program to execute

# MIPS TLB management

The size and organization of the TLB (how many entries, what is in an entry etc.) is defined by the hardware implementation.

The O.S. is responsible for:

- how page tables are organized
- the loading of entries in the TLB
- on a TLB miss, we have a *trap* and the software takes over (but no context-switch)

# MIPS TLB

MIPS R3000 has a 64-entry fully associative TLB

Replacement is "random"

8 entries are "safe", i.e., not replaced, and used by the O.S.

| 20 | 6 | 6 | 20 | 4 | 8 |
|---|---|---|---|---|---|
| virtual page number | PID | | physical frame number | N D V G | |

N don't cache this page

D dirty bit

V valid bit

G global (no PID matching)

# Paging systems: hardware-software interaction

Page tables are built by the O.S.. The first location of the page table is found in a special register that is part of the *state* of the process.

In a paging system, only part of the code/data is in main memory at a given time. Some virtual page might be discarded (or written back to disk) and reloaded in a different physical frame (*relocation*)

The O.S. maintains the data structures necessary to read/write pages from disk (can be embedded in page table or be a different data structure)

When a program attempts to access a location that is not in main memory, we have a *page fault*

# Page fault detection (simplified...)

A page fault is an *exception*.

It will be detected by the hardware (invalid bit in PTE either in TLB or in page table)

To resolve a page fault takes 100,000's of cycles (disk I/O). The program where the page fault occurs must be interrupted (context-switch)

A page fault occurs in the middle of an instruction. In order to restart the program later on, the program *state* must be saved and instructions must be restartable.

The state consists of all registers, including the PC and special registers such as the one indicating the starting location of the page table of the program.

To handle the exception, control is passed (by hardware) to the *page fault handler*, a process in the O.S..

# Page fault handler (schematic)

In general the page fault handler will:

1. Grab a physical frame from a free list (it's the role of the O.S. to be sure that there is always a few free frames)
2. Find out where the faulting page resides on disk
3. Initiate a read from disk of the faulting page to the selected frame
4. Choose a frame to free (if necessary).
5. If the frame is dirty, initiate a write back to disk
6. Context-switch, i.e., switch the processor to another program

When the faulting page has been read from disk (a few milliseconds later), modify the PTE entry to make it become valid. This will imply an *interrupt* (another form of exception) and a context-switch to the O.S. .

The faulting program is now ready to run again.

# Architectural aspects of paging systems (as a contrast with cache organization)

Timing aspects

- A page fault takes milliseconds (100,000's of cycles).
- A TLB miss takes 100's of cycles
- A cache miss takes from 5 to 50 cycles.

Who does what?

- On a page fault, software (O.S.) takes over and there is a context switch
- On a TLB miss either hardware or software takes over. There is no context-switch
- On a cache miss, everything is resolved by hardware

Mapping

- In a paging system the mapping is fully-associative. It is supported by hardware (TLB) and software (page tables)
- Mapping in TLB's can be fully-associative but current trend is towards set-associative
- Caches are direct-mapped or set associative

Page/block sizes

- Page sizes are from 4K to 16K bytes
  - Large page sizes amortize disk I/O time, reduce the number of entries in page tables (less fragmentation) and improve TLB efficiency
  - Small page sizes increase locality, support finer-grain protection, have lower disk transfer latency
- TLB entries map one or two consecutive PTE's, from 8 to 16 bytes.
- Cache block sizes range from 8 to 128 bytes

Write policy

- Pages are always "written-back".
- TLB entries are always "written-back"
- Caches can have write-through or write-back policies

Replacement algorithm

- Important for paging systems (in general some approximation to Least-Recently-Used)
- Not important in caches and TLB's.

# Some optimizations

Speed-up the most common case (TLB hit + cache hit)

- Do TLB look-up and cache look-up in parallel
  - Have index be independent of whether we have a physical or virtual address $\Rightarrow$ cache capacity = page size x associativity
- Have cache indexed by virtual address but with physical tags
- Have cache indexed by virtual addresses with virtual tags

The last two solutions have (solvable) problems because of the presence of *synonyms* (two different virtual addresses mapping to the same physical address)