Memory

Virtual Memory

Where are these page tables?

Memory

Virtual Memory

Where are these page tables?

In memory, of course: and a link to the table is kept in the process's PCB

Memory

Virtual Memory

Where are these page tables?

In memory, of course: and a link to the table is kept in the process's PCB

But it sounds like, *on every memory access*, we have to do (a) a memory read of a page table to find the V to R mapping and then (b) a calculation to get the physical memory location and then (c) a memory access to the physical address we wanted

Memory

Virtual Memory

Where are these page tables?

In memory, of course: and a link to the table is kept in the process's PCB

But it sounds like, *on every memory access*, we have to do (a) a memory read of a page table to find the V to R mapping and then (b) a calculation to get the physical memory location and then (c) a memory access to the physical address we wanted

- every data read

Memory

Virtual Memory

Where are these page tables?

In memory, of course: and a link to the table is kept in the process's PCB

But it sounds like, *on every memory access*, we have to do (a) a memory read of a page table to find the V to R mapping and then (b) a calculation to get the physical memory location and then (c) a memory access to the physical address we wanted

- every data read
- every data write

Memory

Virtual Memory

Where are these page tables?

In memory, of course: and a link to the table is kept in the process's PCB

But it sounds like, *on every memory access*, we have to do (a) a memory read of a page table to find the V to R mapping and then (b) a calculation to get the physical memory location and then (c) a memory access to the physical address we wanted

- every data read
- every data write
- every execute of an instruction

Memory

Virtual Memory

Where are these page tables?

In memory, of course: and a link to the table is kept in the process's PCB

But it sounds like, *on every memory access*, we have to do (a) a memory read of a page table to find the V to R mapping and then (b) a calculation to get the physical memory location and then (c) a memory access to the physical address we wanted

- every data read
- every data write
- every execute of an instruction

This is clearly not sensible as it would be very slow

Memory

Virtual Memory

So, to be practically useful, this is supported by a piece of hardware called the *translation lookaside buffer* (TLB), part of the memory management unit (MMU)

Memory

Virtual Memory

So, to be practically useful, this is supported by a piece of hardware called the *translation lookaside buffer* (TLB), part of the memory management unit (MMU)

The TLB maintains its own copy of *a few* of the virtual-physical mappings from the page table of the current process and can translate very quickly between them

Memory

Virtual Memory

To repeat that: the table in the TLB is a *small subset* of the OS's page table mappings of the current process

Memory

Virtual Memory

To repeat that: the table in the TLB is a *small subset* of the OS's page table mappings of the current process

Only a small subset as TLB memory is very limited since it is very expensive to make memory that runs fast enough to make this mechanism useful: it contains perhaps just a few dozens of the virtual to physical mappings

Memory

Virtual Memory

To repeat that: the table in the TLB is a *small subset* of the OS's page table mappings of the current process

Only a small subset as TLB memory is very limited since it is very expensive to make memory that runs fast enough to make this mechanism useful: it contains perhaps just a few dozens of the virtual to physical mappings

Note (again): the TLB contains copies of the page *mappings*, not pages

Memory

Virtual Memory

The Intel Nehalem architecture has a 64 entry data TLB (and a 512 entry level 2 TLB); and a separate 64 entry instruction TLB

Memory

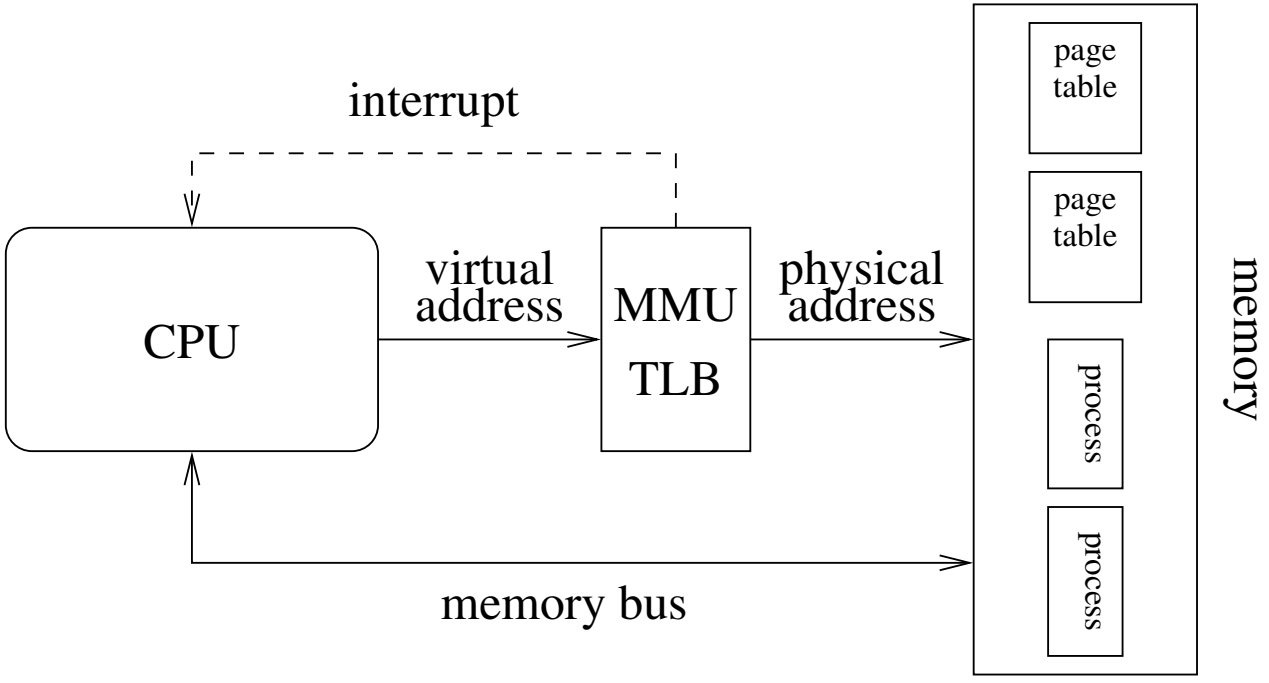
Virtual Memory

The Intel Nehalem architecture has a 64 entry data TLB (and a 512 entry level 2 TLB); and a separate 64 entry instruction TLB

Note that 64 entries typically corresponds to an area of $64 \times$ 4k page $=$ 256k bytes, so while not huge, this isn't so bad as it might seem as first

Memory

Virtual Memory



The MMU and TLB are often physically part of the CPU package, for speed of access

Memory

Virtual Memory

When presented with an address from the CPU the TLB first looks the virtual page up in its table. If it is there is—a *TLB hit*—the memory access goes ahead at full speed using the physical address computed from the real page index found there

Memory

Virtual Memory

When presented with an address from the CPU the TLB first looks the virtual page up in its table. If it is there is—a *TLB hit*—the memory access goes ahead at full speed using the physical address computed from the real page index found there

If there is a *TLB miss* then it has to work a bit harder

Memory

Virtual Memory

When presented with an address from the CPU the TLB first looks the virtual page up in its table. If it is there is—a *TLB hit*—the memory access goes ahead at full speed using the physical address computed from the real page index found there

If there is a *TLB miss* then it has to work a bit harder

There are two popular techniques used

Memory

Virtual Memory

In a *hardware managed* TLB, the CPU/TLB itself stops what it is doing and searches for the page number in the page table (in memory) for the current process: this is called a *page walk*

Memory

Virtual Memory

In a *hardware managed* TLB, the CPU/TLB itself stops what it is doing and searches for the page number in the page table (in memory) for the current process: this is called a *page walk*

If it finds it, it installs it in the TLB table and carries on with the memory access

Memory

Virtual Memory

In a *hardware managed* TLB, the CPU/TLB itself stops what it is doing and searches for the page number in the page table (in memory) for the current process: this is called a *page walk*

If it finds it, it installs it in the TLB table and carries on with the memory access

The OS is not involved in the page walk, it is purely hardware

Memory

Virtual Memory

The second technique, a *software managed* TLB, simply raises a *TLB miss* interrupt on a TLB miss

Memory

Virtual Memory

The second technique, a *software managed* TLB, simply raises a *TLB miss* interrupt on a TLB miss

The OS then has to do the page walk

Memory

Virtual Memory

This deals with the case of when the requested page has already been allocated by the OS to the current process, so there is an entry in the page table for the page walk to find

Memory

Virtual Memory

This deals with the case of when the requested page has already been allocated by the OS to the current process, so there is an entry in the page table for the page walk to find

In either software or hardware case, if the requested virtual page is not yet allocated by the OS to the process and so not in its page table, the OS needs to allocate a page

Memory

Virtual Memory

A hardware managed TLB will now raise a *page fault* interrupt to pass control to the OS

Memory

Virtual Memory

A hardware managed TLB will now raise a *page fault* interrupt to pass control to the OS

A software managed TLB is already running the OS

Memory

Virtual Memory

A hardware managed TLB will now raise a *page fault* interrupt to pass control to the OS

A software managed TLB is already running the OS

The OS allocates a physical page, installs the new page mapping into the page table for that process for that page and writes the relevant page mapping into the TLB

Memory

Virtual Memory

A hardware managed TLB will now raise a *page fault* interrupt to pass control to the OS

A software managed TLB is already running the OS

The OS allocates a physical page, installs the new page mapping into the page table for that process for that page and writes the relevant page mapping into the TLB

(When the process is rescheduled) the memory access can then proceed

Memory

Virtual Memory

Of course, the OS may choose not to allocate a page and it would likely then send a segmentation violation signal to the process

Memory

Virtual Memory

Of course, the OS may choose not to allocate a page and it would likely then send a segmentation violation signal to the process

x86 and ARM processors have hardware managed TLBs

Memory

Virtual Memory

Of course, the OS may choose not to allocate a page and it would likely then send a segmentation violation signal to the process

x86 and ARM processors have hardware managed TLBs

SPARC and MIPS are software managed

Memory

Virtual Memory

Of course, the OS may choose not to allocate a page and it would likely then send a segmentation violation signal to the process

x86 and ARM processors have hardware managed TLBs

SPARC and MIPS are software managed

Terminology warning: a TLB miss when the page is already allocated and indexed in the page table is sometimes called a *minor* or *soft* page fault; while a miss on an unallocated page is a *major* or *hard* page fault

Memory

Virtual Memory

Speed relies crucially on the TLB containing a good proportion of the addresses currently being used: if a process writes wildly all over memory we are guaranteed to get TLB misses and slow memory access: lots of TLB misses and page walks or page fault interrupts

Memory

Virtual Memory

Speed relies crucially on the TLB containing a good proportion of the addresses currently being used: if a process writes wildly all over memory we are guaranteed to get TLB misses and slow memory access: lots of TLB misses and page walks or page fault interrupts

Fortunately, most well-written programs behave sensibly and tend to use the same addresses over and over, meaning lots of TLB hits

Memory

Virtual Memory

Speed relies crucially on the TLB containing a good proportion of the addresses currently being used: if a process writes wildly all over memory we are guaranteed to get TLB misses and slow memory access: lots of TLB misses and page walks or page fault interrupts

Fortunately, most well-written programs behave sensibly and tend to use the same addresses over and over, meaning lots of TLB hits

After a while, the TLB settles down, caching the indices of the pages the process is using, the *working set*

Memory

Virtual Memory

Note that a page fault can cost a lot of time

Register access	1 cycle
(L1 memory cache hit	$\approx$ 2 cycles)
(L3 memory cache hit	$\approx$ 50 cycles)
Main memory access	$\approx$ 200 cycles
TLB miss (page in memory)	$\approx 10,000$ cycles
Page fault (page on disk)	$\approx 1,000,000,000$ cycles

These are very rough figures and are the combined overhead of OS operations and memory architecture