50.002 Computational Structures

Information Systems Technology and Design

Virtual Memory

Natalie Agus (Fall 2018)

1 Memory Contents

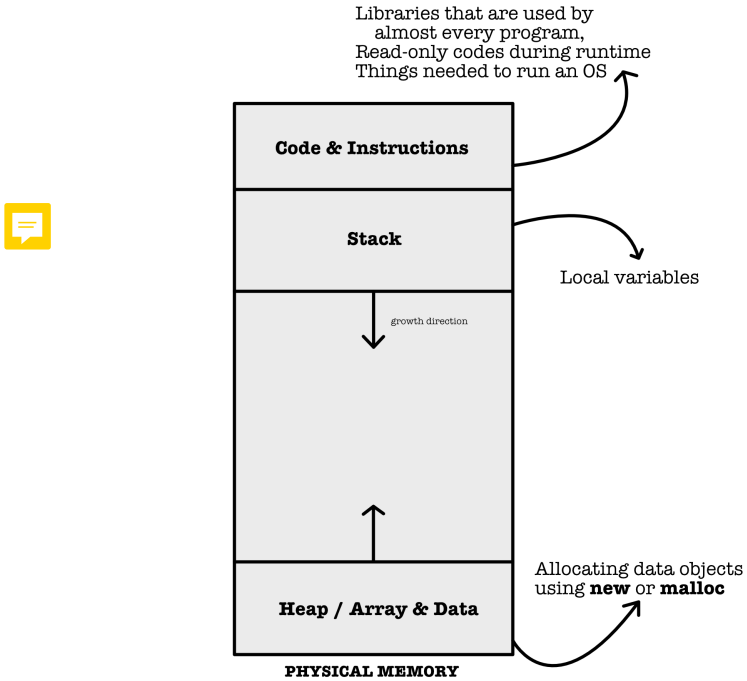

Figure 1

Our physical memory (RAM) contains things shown in Figure 1. Stack and heap can grow during runtime (stack mainly due to recursion and local variables). A physical memory alone definitely is not enough to hold all of our computer data. We need to store the majority of our data in the disk

2 Disk

Disk is cheap, and it gives a big address space, and is **non volatile**, meaning that it retains stored data when unplugged from the power source. Eventually, we will combine DRAM (cache), memory (SRAM), and disk to give the illusion of a computer running at a cache speed with disk-size address space. In other words, we can think of the RAM as the ”cache” to disk.

3 Pages

In a disk, the address space is huge (there's too many memory addresses), so we typically divide them into **pages**,

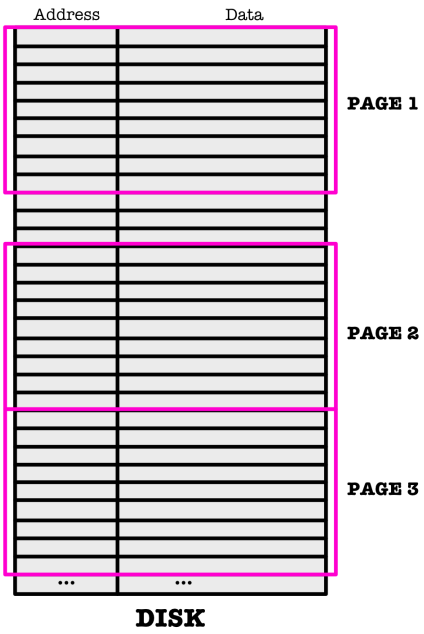

Figure 2

Page: collection of contiguous memory addresses. Good for locality of reference.
So whenever we fetch data from the disk to the RAM, we fetch them by pages. When we want to look for a particular memory location, we now look for their **page number (PPN)** first (i.e: the higher m bits of their physical address), and then their **page offset (PO)** (i.e.: the lower p bits of their physical address).

4 Virtual Memory

A computer can run many programs at a time. These programs ”*do not necessarily know*” the existence of another programs. So for example, all their PCs start from

address 0x0000 within their own program. So we can say each program has their own "**virtual memory**". However physically we only have one RAM, and in that RAM space, each program PC is mapped to different physical addresses, depending on where the instructions for each program codes are stored.

Virtual Address (VA): addresses generated by programs as it runs in the CPU
Physical address (PA): real address wiring in the memory chips

Every VA can be mapped into every PA, but not all VA has corresponding PA at a time in the RAM (it may be in the HDD). The mapping between VA to PA is done by the MMU (memory management unit):

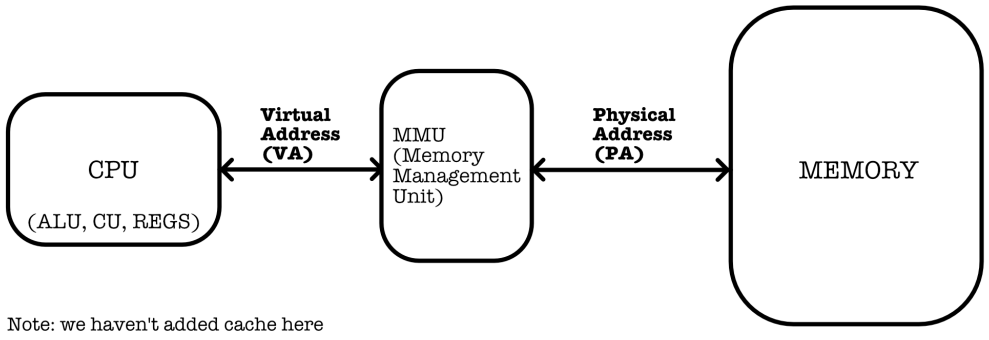

Figure 3

5 Demand Paging

This section explains how the system in Figure 3 works. This entire procedure is called *demand paging*, which is what computers do when it asks for a certain data / instruction from the Memory.

1. Initially, everything is on disk (when a computer just started up)
2. The MMU Pagetable is initially empty, the RAM (Memory) is also empty
3. In the disk, we have a nicely organized array of pages on disk
4. Organized means we know how to get them. The VA generated by the programs upon first launch correspond to the PA **on disk**
5. We bring necessary pages into RAM as program executes
6. Slowly filling up the RAM, we update the MMU so that VA corresponds to PA **in RAM**
7. If RAM runs out fo space, we need to replace pages in RAM (write it to disk if its been changed), and write the new page in from disk to RAM

6 Page-Map Design

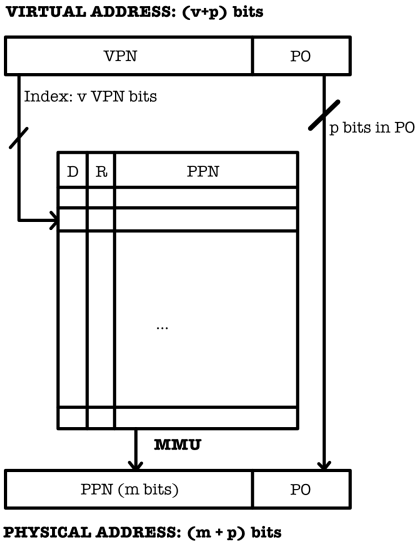

Figure 4

The MMU maps the higher v bits of VA (called the VPN) to a corresponding PPN. The PPN is the m higher order bits of the PA[1]. **1 entry in the MMU is needed for every virtual page**. There are two extra bits D, and R in the MMU:

1. **R: Residence Bit**:
   (a) If $R == 1$, then the data is in the RAM
   (b) If $R == 0$, then it is a **page-fault** exception (no PA in RAM, because data not in RAM). Either get it from Disk or it hasn't been allocated at all.
2. **D: Dirty Bit**: if $D == 1$, then the data has to be written to the Disk before it is removed from the RAM

7 The Virtual Memory

Each program has to refer to the MMU to find out the corresponding PA for some of their VA. The VA address space is much larger than the address space of PA in RAM [2]. Therefore if a page-fault is met, one has to (1) find a page to replace from the RAM, (2) write it to disk if it is dirty, (3) fetch the requested page from disk, (4) write it to RAM, and (5) update the MMU. With this in place, the program had the "illusion" that it has a huge address space with average speed of a RAM. This is called **having a virtual memory**.

[1] due to locality of reference
[2] Whenever we say PA, it refers to the address in RAM, and not Disk

7.1 Pseudocode

```
Given VPN and PO,
        if(R[VPN] == 0),
                PageFault(VPN) /* Go to page fault handler */
        else return (PPN[VPN] << p) | PO /* Concatenate PPN with PO */

If page-fault, /* this is done in software level */
         /* Make space in RAM */
         find LRU page i
         if dirty,
                write to disk, then set R[i] = 0
         otherwise, set R[i] = 0

         /* Fetch data from disk */
         Then PPN[VPN] = PPN[i] /* set the PPN[VPN] to be
                                   the allocated cell in RAM */
         ReadPage(DiscAddr[VPN], PPN[i]) /* get from disk and write
                                            it into memory */
         R[VPN] = 1
         D[VPN] = 0
```

8 MMU Details

8.1 Page-Map Arithmetic

The physical representation of how PPN and PO points to a specific address in the page within memory is shown in the figure below.

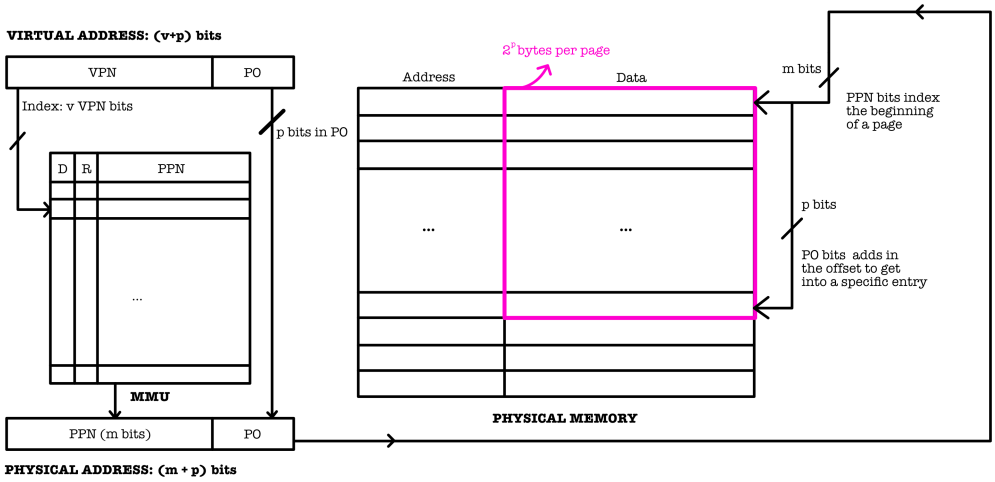

Figure 5

With byte addressing, we have

1. 2^{v+p} bytes of VM
2. 2^{m+p} bytes of actual memory
3. $(m+2)2^v$ bits in MMU Pagetable (2^v rows, each row having $m+2^3$ bits)
4. There are 2^p bytes per page

8.2 MMU Pagetable Location

The MMU Pagetable is actually in the RAM itself, because it has to accommodate every VA and therefore its size is quite large. It is expensive to implement it as SRAM. A portion of the RAM is dedicated to store the MMU Pagetable. The pagetable pointer (**PGTBL Pointer**) points to the first entry of the MMU Pagetable section in the RAM.

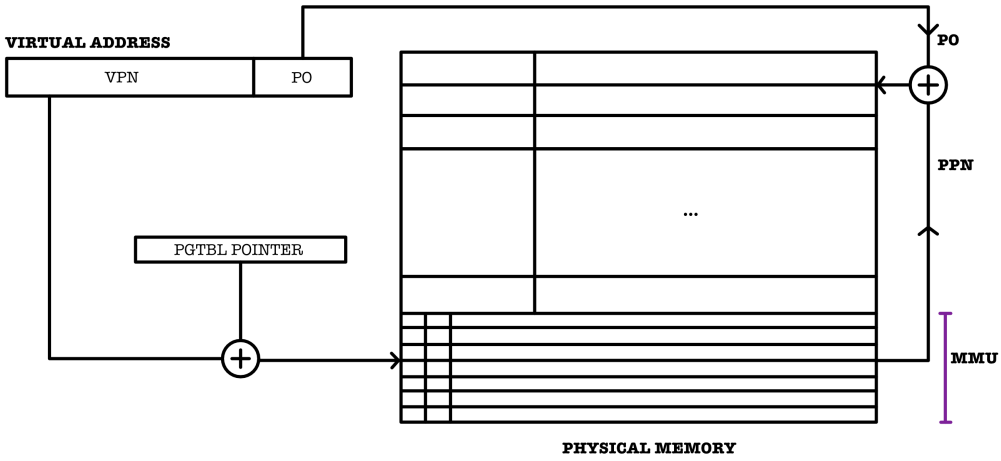

Figure 6

However, this causes us to **access the RAM twice**:

1. Look up Pagetable to get the PA
2. Access the RAM again to get the data with that PA

While it is cheap to put the MMU Pagetable within the RAM, it is too slow. However if we implement the entire MMU Pagetable using SRAM, it is too expensive. The solution is to have a ”cache” for MMU Pagetable, it is called the **Translation Lookaside Buffer** (TLB).

[3]The 2 bits for D and R bits

8.3 TLB

The TLB is serves as a cache to the Pagetable. It is a small, FA cache for mapping certain VPN to PPN. There is locality of reference in address memory reference patterns, therefore there is **super locality** in reference to page-map (hit-rate > 99%).

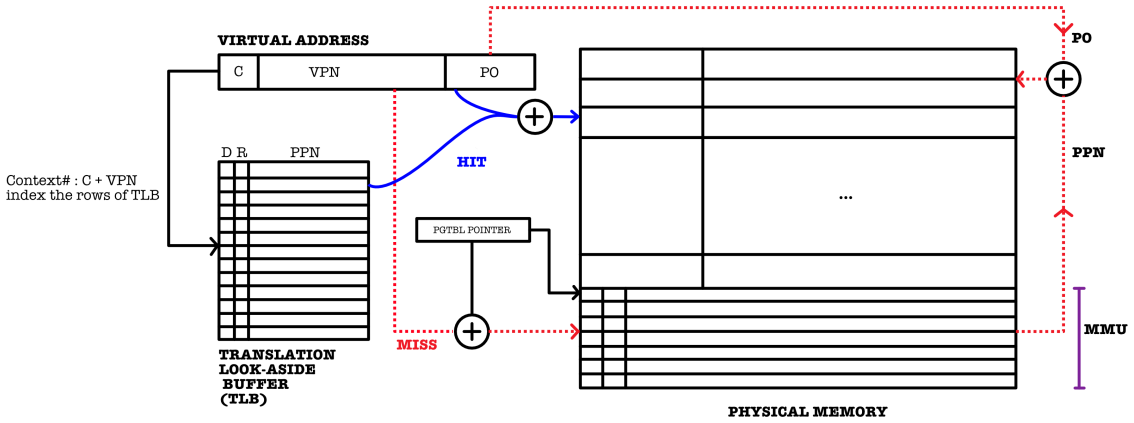

Figure 7

9 Virtual Memory Benefit: Context Switching

In Figure 7, there is an extra field for the VA called *C* (context). A context is a mapping of VA to PA locations as dictated by the Pagetable. Take a look at the figure below:

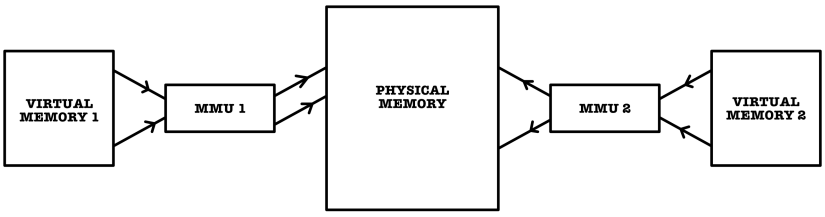

Figure 8

1. Several programs may be loaded to the main memory simultaneously
2. Each program PC starts from 0
3. Each program thinks that they have the entire machine to themselves

4. They can be written as if it has access to all memory without considering other programs
5. The machine does **context switching** to run them ”simultaneously” (i.e: they appear that they run in parallel, but they actually rapidly take turns to run)

9.1 Context Switching

Each program is assigned to a context, with its own Pagetable that translate its specific VA to PA. Context switching between programs is done by adding a register to hold the index of current context, and hence we do not have to ”flush” the TLB whenever we change context. As shown in Figure 7, the context # C and VPN together serve as the index of the rows of the TLB.

10 Using Cache with Virtual Memory

So far we have learned about MMU: consisted of PageTable in the RAM and also a ”cache” to it called the TLB. Given a VA, we refer to TLB (or pageTable if its a miss) to get the PA and then access the RAM. We now combine the concept with *cache* that we learned last week. Recall that a cache stores the memory address and the data straight away (not pages), so there is no need to access the RAM. There are two possible options on where to connect the cache, **before** or **after** the MMU:

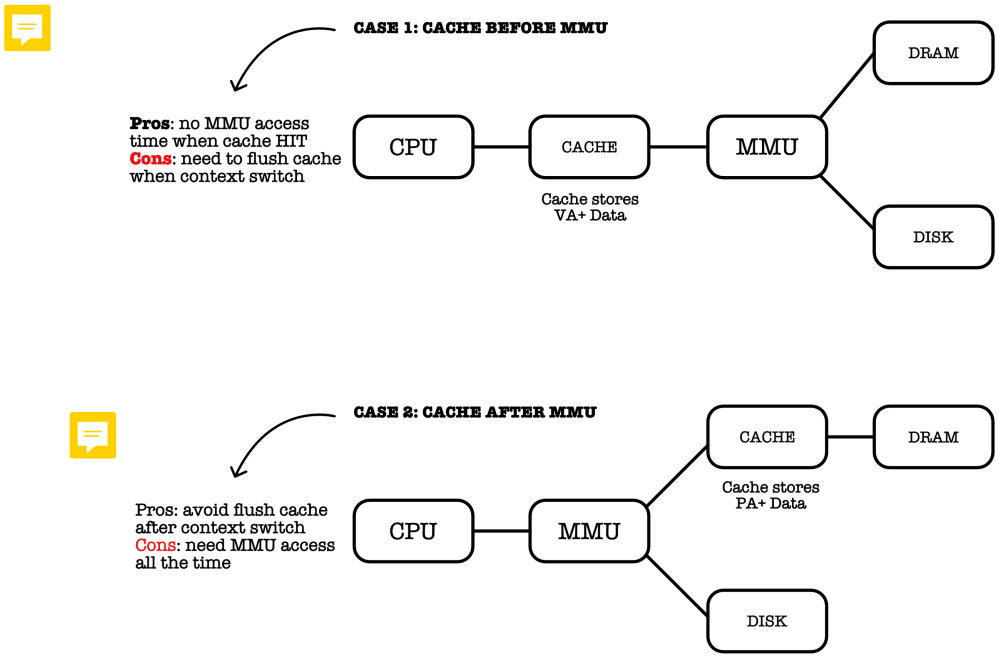

Figure 9

10.1 The best of both worlds

The indexing of cache lines uses PO, while the indexing of the pagetable in MMU uses VPN.

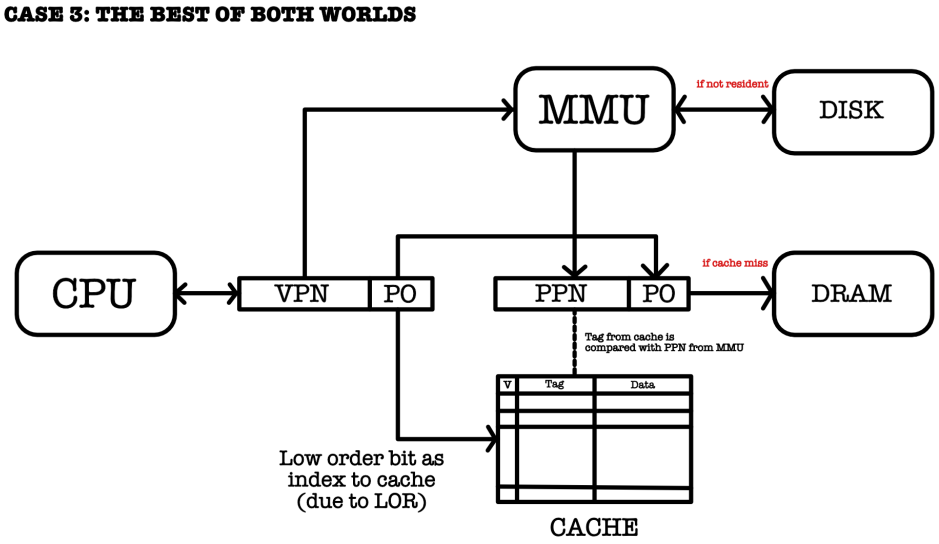

Figure 10

10.1.1 The Cache

Each cache line stores a single data (not pages) and the address of each data. Therefore the index of the tag in the cache is set to be the PO of the VA, due to locality of reference [4]. The content of the tag field in cache is still the PA. The PA from cache is compared with PPN from MMU + PO. **MMU access and cache access can be done in parallel.**

10.1.2 The MMU

If cache miss, then the machine can immediately fetch the data from either memory or disk, and update the MMU and cache at the same time.

[4] If higher order bit is used then you will end up with address contention for DM cache