Part III: Academic research

List three things that surprised you, about the work itself or things the authors mentioned.

1. First of all, and defiantly most importantly had no idea that that the MMU and TLB process was this complex. Furthermore it amazes my that all of this complexity is implemented in the hardware.
2. I am surprised at the hardware variability between AMD and Intel in their implementation in x86-64 processors. I had never quite realized how different they could be.
3. Building on that last point, it is amazing to me how removed the operating system is from all of these hardware processes.

List three questions the paper raised in your mind.

1. The paper dives extensively into the differences between AMD and Intel implementations of the MMU and TLB, but both of these are x86-64 bit processors. I am curious how virtual memory translations are made on ARM chips, like I have in my M1 mac.
2. This paper came out in 2010 and establishes unified translation cache with a modified LRU replacement scheme as the most efficient caching technique. Since then have Intel and AMD switched its design to this mechanism, or is it even widely recognized that the unified translation cache with a modified LRU replacement scheme is the most efficient.
3. The paper suggests that AMD's MMUs uses a page table cache while Intel's MMUs use translation caches. Does this mean that Intel's MMUs are more efficient even though they do not have this modified least recently used algorithm?

List one thing this paper made you want to learn more about.

1. Hearing a lot about radix trees reminded me of decision tree pruning that I had heard about in a video about a chess AI. This led me down quite a rabbit whole of pretty casual personal research but I would love to continue personally exploring this topic as it is super interesting to me.

Why can't page table caches cover the same amount of a process' address space as translation caches?

- A page table cache is less efficient with storing entries than a translation cache. In order for a page table cache to provide coverage for an entry it needs to hold the information at all levels up until the L2 entry. Meaning it needs to hold the L4, L3, and L2 entries in order for the cache to be able to "find" that L2 entry. This is because the page table cache requires a top to bottom mapping. Contrarily because a translation cache can be accessed in any order it only needs to hold the L2 entry itself, and can have other entries in the place of L3 and L4. Thus meaning that the translation cache has a greater coverage.

What is significant about processes that use large amounts of memory versus those that use small amounts of memory when discussing page table walks?

- A process that uses a large amount of memory as compared to a process only accessing a small amount of memory differ in that the larger process can instead be assigned a "large-page" of virtual memory. Instead of being assigned a normal 4KB at the L1 level a "L2 entry can directly point to a contiguous and aligned 2MB data page instead." This is beneficial in three ways. First the L2 "large-page" support increases the TLB coverage. Similarly, it also reduces the number of page entries required as each entry is much larger. Finally, in decreases the number of memory accesses from four to three needed in order to locate a page.