24

# Summary Dialogue on Memory Virtualization

**Student:** *(Gulps) Wow, that was a lot of material.*

**Professor:** *Yes, and?*

**Student:** *Well, how am I supposed to remember it all? You know, for the exam?*

**Professor:** *Goodness, I hope that's not why you are trying to remember it.*

**Student:** *Why should I then?*

**Professor:** *Come on, I thought you knew better. You're trying to learn something here, so that when you go off into the world, you'll understand how systems actually work.*

**Student:** *Hmm... can you give an example?*

**Professor:** *Sure! One time back in graduate school, my friends and I were measuring how long memory accesses took, and once in a while the numbers were way higher than we expected; we thought all the data was fitting nicely into the second-level hardware cache, you see, and thus should have been really fast to access.*

**Student:** *(nods)*

**Professor:** *We couldn't figure out what was going on. So what do you do in such a case? Easy, ask a professor! So we went and asked one of our professors, who looked at the graph we had produced, and simply said "TLB". Aha! Of course, TLB misses! Why didn't we think of that? Having a good model of how virtual memory works helps diagnose all sorts of interesting performance problems.*

**Student:** *I think I see. I'm trying to build these mental models of how things work, so that when I'm out there working on my own, I won't be surprised when a system doesn't quite behave as expected. I should even be able to anticipate how the system will work just by thinking about it.*

**Professor:** *Exactly. So what have you learned? What's in your mental model of how virtual memory works?*

**Student:** *Well, I think I now have a pretty good idea of what happens when memory is referenced by a process, which, as you've said many times, happens on each instruction fetch as well as explicit loads and stores.*

**Professor:** *Sounds good — tell me more.*

**Student:** *Well, one thing I'll always remember is that the addresses we see in a user program, written in C for example...*

**Professor:** *What other language is there?*

**Student:** *(continuing) ... Yes, I know you like C. So do I! Anyhow, as I was saying, I now really know that all addresses that we can observe within a program are virtual addresses; that I, as a programmer, am just given this illusion of where data and code are in memory. I used to think it was cool that I could print the address of a pointer, but now I find it frustrating — it's just a virtual address! I can't see the real physical address where the data lives.*

**Professor:** *Nope, the OS definitely hides that from you. What else?*

**Student:** *Well, I think the TLB is a really key piece, providing the system with a small hardware cache of address translations. Page tables are usually quite large and hence live in big and slow memories. Without that TLB, programs would certainly run a great deal more slowly. Seems like the TLB truly makes virtualizing memory possible. I couldn't imagine building a system without one! And I shudder at the thought of a program with a working set that exceeds the coverage of the TLB: with all those TLB misses, it would be hard to watch.*

**Professor:** *Yes, cover the eyes of the children! Beyond the TLB, what did you learn?*

**Student:** *I also now understand that the page table is one of those data structures you need to know about; it's just a data structure, though, and that means almost any structure could be used. We started with simple structures, like arrays (a.k.a. linear page tables), and advanced all the way up to multi-level tables (which look like trees), and even crazier things like pageable page tables in kernel virtual memory. All to save a little space in memory!*

**Professor:** *Indeed.*

**Student:** *And here's one more important thing: I learned that the address translation structures need to be flexible enough to support what programmers want to do with their address spaces. Structures like the multi-level table are perfect in this sense; they only create table space when the user needs a portion of the address space, and thus there is little waste. Earlier attempts, like the simple base and bounds register, just weren't flexible enough; the structures need to match what users expect and want out of their virtual memory system.*

**Professor:** *That's a nice perspective. What about all of the stuff we learned about swapping to disk?*

**Student:** *Well, it's certainly fun to study, and good to know how page replacement works. Some of the basic policies are kind of obvious (like LRU, for example), but building a real virtual memory system seems more interesting, like* *we saw in the VMS case study. But somehow, I found the mechanisms more interesting, and the policies less so.*

**Professor:** *Oh, why is that?*

**Student:** *Well, as you said, in the end the best solution to policy problems is simple: buy more memory. But the mechanisms you need to understand to know how stuff really works. Speaking of which...*

**Professor:** *Yes?*

**Student:** *Well, my machine is running a little slowly these days... and memory certainly doesn't cost that much...*

**Professor:** *Oh fine, fine! Here's a few bucks. Go and get yourself some DRAM, cheapskate.*

**Student:** *Thanks professor! I'll never swap to disk again — or, if I do, at least I'll know what's actually going on!*