PROFESSOR: OK. Welcome. This is great. It almost looks like the beginning of the Fall, at least outside.

So, what we are going to do, is it's going to be a very exciting month. And mainly the reason for doing this course, is I think we are about a software crisis. So there's an interesting quote I want to start with. This is somebody very famous in computer science, who spoke during his student award lecture. His point was, when the computers were small, there's no problem. When computers got bigger, this is basically when software started having problems.

So, the first software crisis, in fact, happened around the '60s and '70s. The problem for the first software crisis was people are writing Assembly language program. When the computers are small, and memory was small, it was completely doable. No problem. But as computers became larger and larger, it became very hard to write Assembly language programs. And what they needed was some kind of abstraction portability. So you should be able to go every two years, to not have to rewrite the programs in there. So you need portability and you need some level of abstraction. So, what that means is instead of writing all Assembly hacking and dealing with every hardware feature, you can erase about that.

And the way that we solved it, in that generation is, we, came up with these high level languages like Fortran and C. And what those did was, they kind of provided a common machine language across these machines. And so what these did was, they looked at all machines those days, the uniprocessors, and kind of abstracted out:

First of all, all the common properties made available for the programmer. So it didn't really lead the programmer too far away from the machine. But it abstracted out all the differences. So by doing that, we avoided, we managed to get these languagese like C and Fortran. Weren't very fast. Pretty efficient. And working through this.

So that's how we avoided the first software crisis. And I think that a lot of really good work happened, and a few people got Turing Awards for doing these kind of things.

And then, we faced a second software crisis in the '80s and '90s. The problem there was, C was great if two guys sit in a corner and write a program. When Microsoft started programming Word, Word suite, with a few hundred programmers, the problem was the bugs never went away. The project was supposed to finish in a year. It lasted three years. And bugs kept coming and coming. And they had no idea when the project was going to finish. Because the way you write in a small piece of code really didn't scale. And, this was a huge crisis for people, who said we will never be able write large programs. Because these programs will never work. They're going to be brittle.

And the problem there is we want to be able to build software that's composable, malleable, and maintainable. So software should be: Multiple people should be able to do it. Compose it. And the requirements say it should be easy to change and easy to maintain.

The interesting thing is, we never cared about high performance at that time. Moore's law. They started to take care of that.

The way we solved that crisis -- Again, a lot of people got very famous for doing that, is came up with things like object-oriented programming languages. C#, C++, and now C# and Java, basically made it possible for, now, hundreds if not thousands of people to develop programs. If you look at something like new Microsoft products coming out, thousands of people are involved. And still they manage to get it out on time.

And, also we did a bunch of better tools. Things like component libraries, and tools like Purify. And we developed a lot of software engineering methodology to develop software. We made it more of a process than an ad hoc art-form. This is great.

So here we are today. We are very happy. A lot of programmers, yes. Unfortunately, programmers are pretty oblivious to what's happening in hardware. We don't care. Programs don't have to know about what things are running. You have these nice virtual machines. And, of course, Moore's law, this great new thing is going to come and take care of all us. If you want speed, just wait six months, the machines will be faster. And that's the thing we have been going on.

The nice thing is, these programs, programs written even in the '70s, still run. Probably two, three, orders of magnitude faster, because processors today have gotten really fast. And this abstraction gave us huge amounts of power. Just say, don't care what's underneath. We are going to sit up there, and just basically program. We were very happy with that.

The problem is, we are running into an issue here. So, we are at the beginning of this third software crisis. What I call the software crisis due to the multicore menace. So, multicore, for hardware people, it's a good opportunity. But for software people, its kind of a menace in there. Problem is, sequential programs are left behind by Moore's law. And, as you see, as probably a lot of you know, that you're not going to get performance on sequential programs, increased performance, without dealing with this

And -- the key thing here is, so why don't we just stop? OK. Good. We have the programs. Machines today run very fast. Isn't that good enough?

The problem is, we want it all. We need to support new features. We need to support things like large datasets in the same programs. And we kind of keep expecting a certain performance gain every year. To sustain portability and malleability, we assume that every few years, the machines' performance will double. Because we believe in that. For example, Microsoft Word at the beginning, ten years ago, it still works. But it didn't have Spellcheck running in the background. We didn't deal with very large images. We didn't deal with XML formats in there. And, then of course, next generation, we probably want to have integrated video, audio, voice recognition, all those things in there. And, we need that power. We rely on that power.

And this is going to be a big problem, because sequential programs are stuck. They're not going to provide that power anymore. And it's critical. This kind of power is critical to keep up the current rate of evolution. And, if we get stuck there, we are going to have a big crisis happening.

So, how is this happening? I talked about Moore's law. So this the obligatory slide of Moore's law. So if you look at it, going from even before the 1970's, we had this amazing, amazing growth in the number of transistors available. So Moore's law doesn't say anything about performance. It says about the number of transistors available on a die. Every 18 months, it doubles. So, this basically -- some people even say -- this is the basis of these last two decades of economic expansion. Because we got all this amazing power, because of this kind of growth in here.

The problem is, if you look at performance, it's flattening out. So, this is spec numbers. Kind of the way you measure performance is to have this set of programs that we can keep running on these machines. And we keep running on these same machines. So even though we are getting twice the amount of transistors, this is kind of flattening out. And then, for the last few years, it's almost constant. We're not getting anything from there. And so, we're still running from the fumes of that amazing growth rates we had in this time. And now we're kind of flattening out in here.

And why is this? The main reason for this is, I will talk about four things. One

is power consumption. Wire delay, and DRAM access time, and diminishing returns. So I'll go a little bit into these details, to kind of give you an overview of why this happened.

The first thing is power. So as we go about, we keep taking more and more power. Right now, if you look at the one Moore trend, that, normally, these one point-some volts that these processors run, this draws more current than any of your houses back home. I mean, that's amazing. The houses probably draw like 150 amps. And in one volt, we are drawing about 150 watts. So there's that much current going to those things. And, heat is a big issue.

So this still shows, OK, power, we are flattening out. But another interesting thing is power efficiency. So, we are trying to use these transistors. We are trying to do more and more out of these transistors. And if you look at how many watts, you are spending to get one of those spec rates, that's basically how much to do the same amount of work, we are spending a lot more power. Because we are building these things that have very marginal return today. That is the problem in building the current super-scaled type things. Because it doesn't go. This power efficiency.

This is why Google is building their shops next to the Grand Coulee dam, because they are sucking in so much power. Because power efficiency is a big issue for them. And, today, microprocessors, most the cost of it is more than purchasing. It's basically feeding it with power and cooling it.

So another problem is wire delay. If you try to build this monolithic, large process. So what happened is, in the good old days, within the clock cycle, in the processor [UNINTELLIGIBLE] this side. You can basically send a wire from one end to another processor, still within one clock cycle. So if you are building this monolithic thing, it's almost like sitting in a room. When I talk, the other end of the room will hear it instantaneously. Because I'm talking slowly, or you're clocking slowly.

And today, we are around this area, So what that means is, in a clock cycle, you can't get a wire for more than a small part of the chip. So it's almost like trying to communicate in this building. I want to tell something to everybody, I can't broadcast. Somebody has to go next-door to tell somebody to get the information through the entire thing. Takes time. And as we go more and more, this is going to be harder and harder and harder.

And what that means is, trying to build large, monolithic, processors like we did to keep the Moore's law kind of performance going, is not working. And we have this problem. Interesting third issue is DRAM access time. So what's happening is microprocessors going like this. Performance gets faster and faster. DRAM gets doubled only about every 10 years. And because of that, there's this interesting mismatch. If you are trying to get to DRAM, if you put things in DRAM, it's almost -- about 10 years ago the kind of latency we had to the disk, is now we are having to DRAM. So the way we look at this activity, the costs of it are going to get big. And it's also a power issue because if you're on chip, it's probably two orders of magnitude cheaper to get to a word on chip then go all the way outside, get to the DRAM and get things out. And so, between power and this, is also creating a huge amount of issue, an impossibility for us to keep this kind of Moore's law type performance to keep going.

The final thing is diminishing returns. So if you look at the `80s, that is the era of superscalars. We had amazing performance improvements. For example, when you started the decade, every CPI speed, clocks per instructions. How many clocks you had to run to get one instruction going? At the beginning of the decade, you had to have about 10 clock cycles to execute one instruction on [UNINTELLIGIBLE]. At the end of the decade, we went about one clock cycle [UNINTELLIGIBLE] we get a new instruction graduated from that. And that gave an amazing performance boost.

And in the '90s, we had an era of diminishing returns. We kind of went from this two-way, superscaled, almost like six-way out of order, out of issue, branch prediction, everything's predicated. And this huge pipeline. Amazingly complex machines. But we went from one clock per instruction to about half a clock per instruction. So we just doubled that kind of performance. And performance was below expectations, and projects got delayed and cancelled.

One thing that failed -- I'm not going to talk about that much. Is, even at this time, you could get that clock frequency higher and higher. And because of power and other physical limitations, we are also hitting a wall in clock frequency. We are not getting -- because at that time, if you realize, everybody [UNINTELLIGIBLE] saying how fast their machine ran. And they have stopped that, because they can't get machines to double their frequencies any more.

So, between those two, your performance kind of stuck. So that is why now we are starting the new era of multicores. We need explicit parallelism. We can't build these large monolithic things that keep giving you performance every few years. That stopped. And we can build a sequential machine where what you get in 10 years' time will almost be same as what you get today.

But, we get more and more silicon. It's doubling. Moore's law still keeps going. So what you do with that silicon is replicate, and build more. And that's where we are going.

So, if you look at what's happening, the unicores are a dying breed. The last big one was Intel had this project that tried to build very fast, amazing, superscalar machines. All those projects had to get cancelled. Because they could not get the performance. Power was way too high, they couldn't get the power in there. And those things got cancelled. And as you see, there's more and more multicores coming in there. In fact, if you look at it, multicores are here in a big way. If you look all these machines, the Pentium line of uniprocessors. From recent time, there had been four cores, eight cores and even machines that have more than 256 cores. Small cores, but we are basically going into -- that's a new trend. And if you look at the trend line, it's an interesting trend line going in there.

So, what I'm going to do now is switch gears. Because, we had Mike [? Brown ?] from IBM. We are very fortunate to have him here to give us a feel about the cell processor. Because we are going to spend this month really getting to understand cell processing. And really taking advantage of that. And he had done huge amounts of work with cells. He's at IBM TJ Watson. And so he's going to, in the next hour, hour and half, he's going to give us the lowdown on cell. And then I will conclude with a little bit more remarks about how we are going to do the class structure. OK. Mike. Hopefully -- he has to catch a flight, so we will kind of go fast. Hopefully you guys [INAUDIBLE] go without too much of a break. So if things get too tired, let us know so we'll see if we can put a break in the middle.