# Lecture 1 - Logistics, Introduction to VMs

**Office hours**: proposed 3:05 - 4:35PM

### Course Overview and Structure

- Lectures will not be recorded
- Lectures will primarily be on the board, rarely slides
- No homeworks, no exams
- I will occasionally post readings. These are for your own good. You should get the book (Smith & Nair)
- This course is purely project based. Your grade will be based on projects and on participation
- 4 Projects
    - Emulation
    - HLL VMs
    - Hypervisor (System VMs)
    - Container Engine
- If you're a PhD student (or in other exceptional cases), the last project (containers) can be substituted for a research project
- After each project we will have short presentations/code review
- Discussion of prereqs: C, OS, assembly, hopefully computer architecture
- Course is still in development, so feedback welcome

### Virtual Machines Intro

What is meant by **virtualization?** Recall from your OS class, e.g. virtual memory

- *Illusion* of a (sometimes unlimited) physical resource with certain properties
- We will see that for us to properly virtualize some resource (e.g. memory, a CPU, a physical machine, or an abstract machine), we must *adhere to an interface*
- Examples:
    - With virtual memory, we adhere to the interface of loads/stores with an address. We make it appear as if each process has it's own address space -> requires hardware support (paging)
    - With a virtual CPU, the interface we adhere to is the ISA
    - With a virtualized *system*, we must also provide for the familiar interface of chipsets, devices, bussees, etc.
- Adhering to this interface allows us to achieve *equivalence*, something we'll come back to later
- How we *implement* the interface can differ widely depending on our technique
    - **Emulation**: the visible effects are equivalent to what the user/program would

have normally seen, but we don't really care about *how* that is achieved
    - **Simulation**: we are still implementing the same interface, but in simulation we also care about the physical manifestation of the implementation. For example, a virtual hard disk might actually model the timing of disk rotations and head seeks, or we might actually simulate the gate delays on circuits used to implement some other solid state device
        - In practice it is very difficult to make simulation efficient, limiting its real-time applications
    - **Virtualization:** really includes both of the above. In some cases we can achieve virtualization without *either* emulation or simulation if we have hardware support (although you could argue that even that is a higher-order emulation)
    - Important to note that in any case, we are not necessarily mimicking real hardware, we could also emulate non-existent or prototype hardware, or indeed abstract hardware (as we'll see)

**Theoretical Underpinnings**

- Virtual machines go all the way back to Turing Machines
- Recall what a Turing machine is (formal definition)
- UTMs: TM which can read the transition function and input tape of another TM and perform the associated actions
- Any turing complete computer, we can simulate (virtualize) as long as the machine we're doing it on is also turing complete! This means that virtualization is very powerful and general. If we wanted to, we could build a hypervisor in a finite Turing Machine!

**VM Taxonomy**

- Language VMs: emulate an *abstract* (not real) instruction set, implementing a language
- Process VMs:  VMs that runs as a processes on a host OS (in userspace). Language VMs and System VMs can be Process VMs. Examples: QEMU, Java
- Full System VMs: VMs which present the virtualization of an entire system (i.e. not just a CPU, real or abstract)
- Hypervisor: a virtual machine monitor (manager)