How can we access the OS functionality?

* Problem: If a task is protected from getting into the OS code and data, OS functionality are restricted from these tasks
* How does CPU know if certain instruction should be allowed?
* How does OS grant a task access

Kernel Mode vs User Mode

* Processors include a hardware *mode* bit that identifies whether the system is in *user* mode or *supervisor/kernel* mode
  + Requires extra support from the CPU hardware for this OS feature
* Supervisor or kernel mode (mode bit = 0)
  + Can execute all machine instructions, including privileged instructions
  + Can reference all memory locations
  + Kernel executes in this mode
* User Mode (mode bit = 1)
  + Can only execute a subset of non-privileged instructions
  + Can only reference a subset of memory locations
  + All applications run in user mode

Multiple Rings/Modes of Privilege

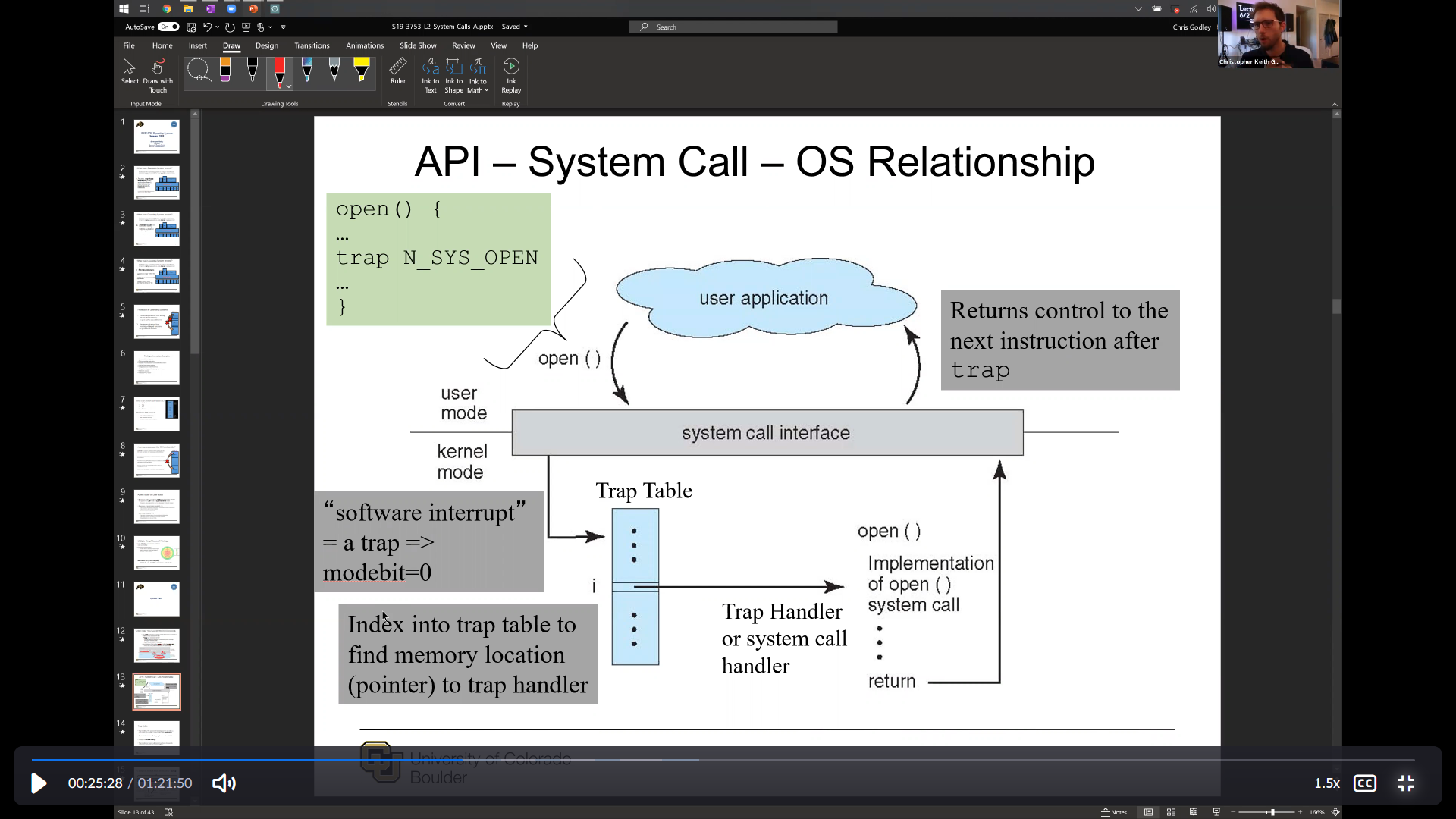
* Intel x86 CPUs support four modes or rings of privilege
* Common configuration:
* OS like Linux or Windows run in ring 0 (highest privilege), Apps run in ring 3 and rings 1-2 are unused
* Virtual Machines (one possible configuration)
* VM’s hypervisor run in ring 0, guest OS runs in ring 1 or 2, apps run in ring 3

System Call

System calls: How apps and the OS Communicate

* The *trap* instruction is used to switch from user to supervisor mode, thereby entering the OS
  + Trap sets the mode bit to 0
  + On x86 use INT assembly instruction (more recently SYSCALL/SYSENTER)
  + Mode bit set back to 1 on return
* Any Instruction that invokes *trap* is called a *system call*
  + There are many different classes of system calls
*  So laying out the steps to run a system call again we we are in a user space program and we're trying to access something that Colonel We need curl privileged to do. So what we do is we ask the colonel nicely through one of these system calls in order to do that operation.

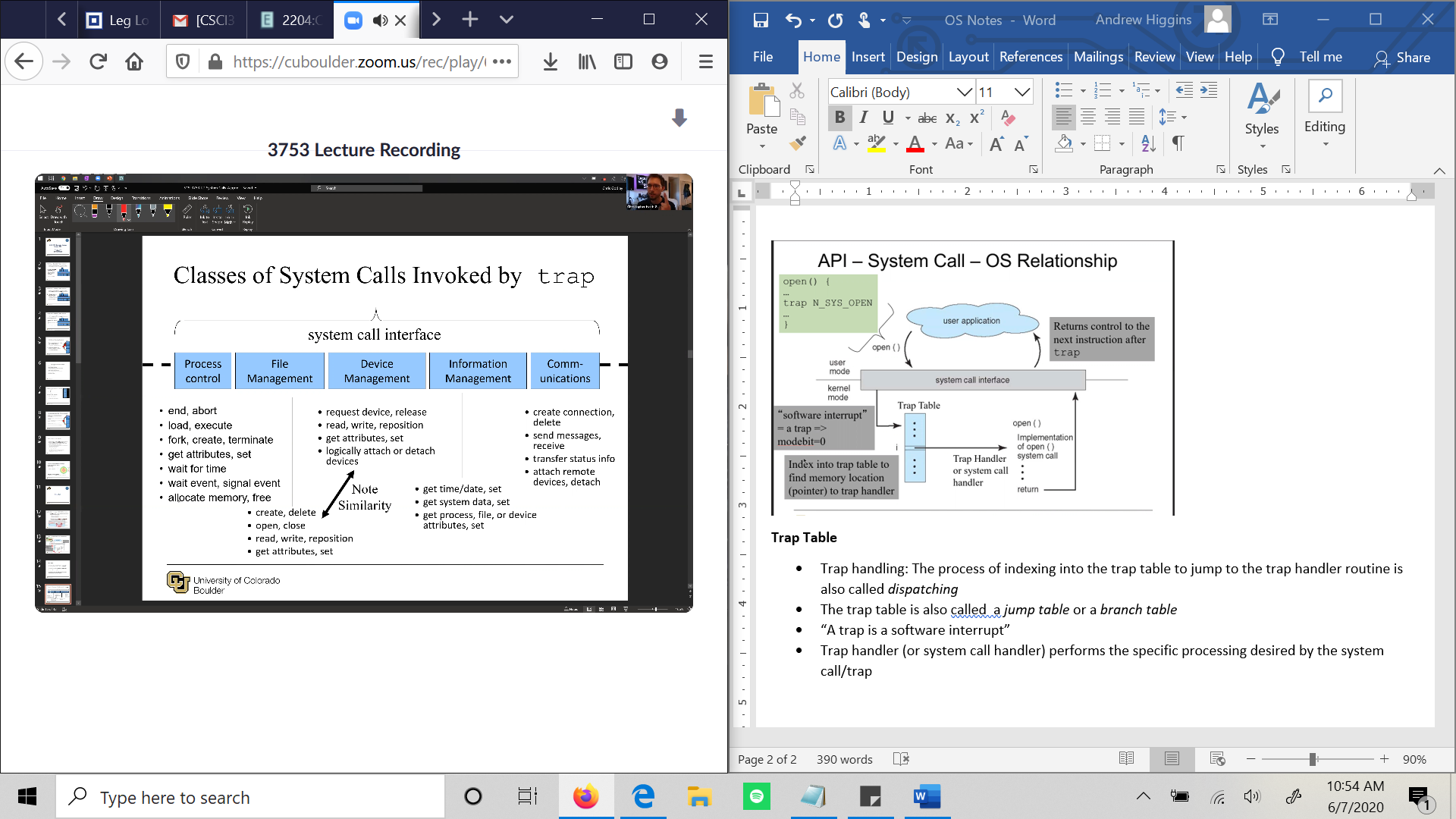
**API – System Call – OS Relationship**



**Trap Table**

* Trap handling: The process of indexing into the trap table to jump to the trap handler routine is also called *dispatching*
* The trap table is also called a *jump table* or a *branch table*
* “A trap is a software interrupt”
* Trap handler (or system call handler) performs the specific processing desired by the system call/trap

**Classes of System Calls invoked by *trap***



**Standard C Library**

* C program invoking printf() library call, which calls write() system call

**System Call Parameter Passing**

* Three genereal methods used to pass paramters to the OS
  + Register: Simplest, pass the parameters in *registers*
    - In some cases, may be more parameters than registers
  + Pointer: Parameters stored in a block or table, in memory and address of block passed as a parameter in a register
    - This approach taken by Linux and Solaris
  + Stack: Parameters placed, or pushed, onto the stack by the program and popped off the stack by the operating system

-Block and stack methods do not limit the number or length of the parameters being passed