



Using the Terminal

Outline for Today

- ***What is the Terminal?***
 - Controlling your computer with pure text.
- ***Working from the Command Line***
 - Basic commands and how to use them.
- ***Using Coding Agents Safely***
 - A more nuanced look at what agents do.
- ***Virtualization***
 - Shielding your computer.

Previously...



Claude Code v2.1.280
Sonnet 5 · Claude Enterprise
~/projects/rain-music

/clear

Create a website that simulates a rectangular puddle of water with raindrops falling onto it.

Generate music in real time whenever a raindrop hits the water and whenever two ripples touch one another.

Bash(ls -la /home/student/projects/rain-music/ 2>&1)

```
└─ total 16
   drwxrwxr-x 2 student student 4096 Sep 22 18:59 .
   drwxrwxr-x 4 student student 4096 Sep 21 20:55 ..
   -rw-r--r-- 1 student student 5784 Sep 21 23:20 CLAUDE.md
```

...

What does all
this mean?

Bash(npx playwright test 2>&1 | tail -n 100)

```
└─ Running 5 tests using 1 worker
```

```
  ✓ 1 tests/rain-music.spec.js:8:3 › rain music puddle › loads with overlay gate and no console errors (618ms)
```

```
  ✓ 2 tests/rain-music.spec.js:30:3 › rain music puddle › clicking overlay starts audio and hides overlay (292ms)
```

```
  ✓ 3 tests/rain-music.spec.js:46:3 › rain music puddle › mute button toggles icon and slider changes intensity (278ms)
```

```
  ✓ 4 tests/rain-music.spec.js:62:3 › rain music puddle › heightfield changes over time as rain falls (canvas pixels vary) (1.7s)
```

```
  ✓ 5 tests/rain-music.spec.js:89:3 › rain music puddle › ripple collision fires the collision voice (no crash over time, drops accumulate and clear) (4.1s)
```

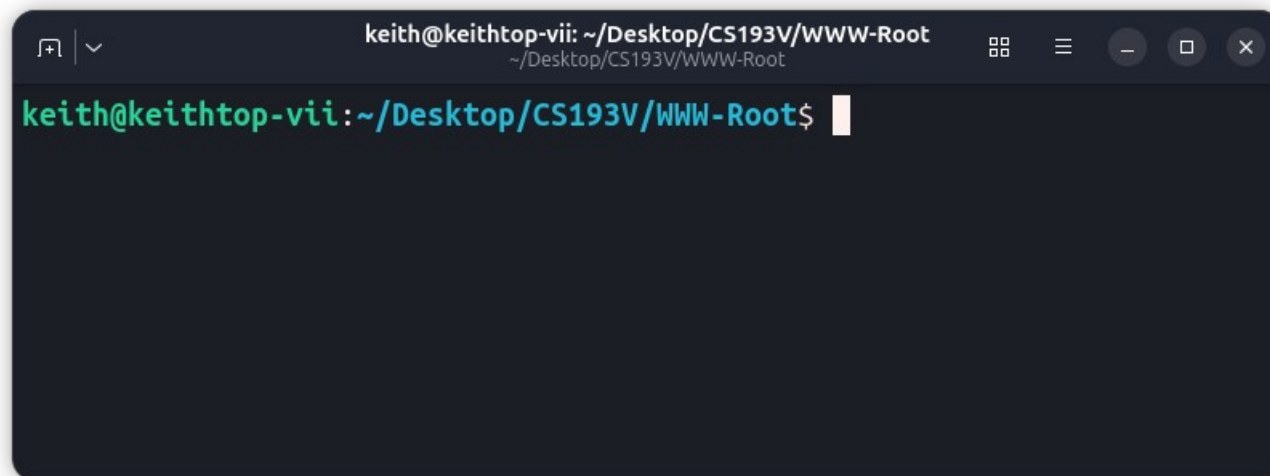
```
5 passed (7.5s)
```

```
└─ (timeout 2m)
```

Working from the Terminal

The Terminal

- The ***terminal*** (or ***command-line***) is a text-based interface for issuing commands to your computer.
 - It uses text, but it's not AI. The technology was developed in the 1970s.
- You type commands into the terminal, which then execute on your computer.



The Terminal

- The ***terminal*** (or ***command-line***) is a text-based interface for issuing commands to your computer.
 - It uses text, but it's not AI. The technology was developed in the 1970s.
- You type commands into the terminal, which then execute on your computer.
- Terminal usage is a “lingua franca” for running commands on a computer.
- ***Our goal:*** Give you a basic familiarity with the terminal so you know enough to interact with it when vibecoding.

A Note on Versions

- Most OSes ship with a terminal, but those terminals behave differently.
- The syntax we'll cover is for the *bash* command interpreter on the Linux operating system.
- You'll vibecode on your own machines using Linux (details later). What you see here today is what you'll use this quarter.
- The Windows terminal ("PowerShell") is *very* different than what we'll see today.
- The one in macOS is *slightly* different.

The Basics: ***Files and Directories***

Files and Directories

- Files on your computer are organized into **directories**. Think “folders.”
- As you navigate around your computer, the directory you’re in is called the **working directory**.
- Useful commands for getting started:
ls **cd** **clear** **exit**
- Special symbols:
 - . is the current directory.
 - .. is the parent directory.
 - ~ is your home directory.
 - / is a separator in file paths.

tldr

- The default command to get help is **man** (manual). This documentation is comprehensive but not beginner-friendly.
- This quarter, we'll use a custom utility called **tldr** where volunteers have provided lots of examples to use. (Read more [*here*](#).)
- Run it by typing
tldr *command-name*
- You *can* ask an LLM for help with commands, but be *really careful* if you do. More details later.

Switches

- Many commands can be suffixed with a ***switch*** to change their behaviors.
- Examples:
 - ls**: List everything in the directory
 - ls -l**: ... with extra information per file
 - ls -a**: ... including normally-hidden files.
- Switches are usually preceded by dashes. Two one-letter switches can be combined:
 - ls -la**: List everything, with more info, including normally-hidden files.
- Some switches have human-readable names.
 - ls --all** is the same as **ls -a**.



Claude Code v2.1.280
Sonnet 5 · Claude Enterprise
~/projects/rain-music

/clear

Create a website that simulates a rectangular puddle of water with raindrops falling onto it.

Generate music in real time whenever a raindrop hits the water and whenever two ripples touch one another.

```
Bash(ls -la /home/student/projects/rain-music/ 2>&1)
└─ total 16
   drwxrwxr-x 2 student student 4096 Sep 22 18:59 .
   drwxrwxr-x 4 student student 4096 Sep 21 20:55 ..
   -rw-r--r-- 1 student student 5784 Sep 21 23:20 CLAUDE.md
```

...

```
Bash(npx playwright test 2>&1 | tail -n 100)
└─ Running 5 tests using 1 worker

   ✓ 1 tests/rain-music.spec.js:8:3 › rain music puddle › loads with overlay gate and no console errors (618ms)
   ✓ 2 tests/rain-music.spec.js:30:3 › rain music puddle › clicking overlay starts audio and hides overlay (292ms)
   ✓ 3 tests/rain-music.spec.js:46:3 › rain music puddle › mute button toggles icon and slider changes intensity (278ms)
   ✓ 4 tests/rain-music.spec.js:62:3 › rain music puddle › heightfield changes over time as rain falls (canvas pixels vary) (1.7s)
   ✓ 5 tests/rain-music.spec.js:89:3 › rain music puddle › ripple collision fires the collision voice (no crash over time, drops accumulate and clear) (4.1s)

   5 passed (7.5s)
└─ (timeout 2m)
```

Looking at Files

- **cat *filename***: Print the contents of a file.
- **less *filename***: Print the contents of a file, letting you scroll with the arrow keys.
- **head -n *N filename***: Print first *N* lines of a file.
- **tail -n *N filename***: Print last *N* lines of a file.
- Funny things happen if you use these commands on things that aren't plain text files (PDFs, audio files, JPGs, etc.)
- There are specialized commands for other file formats. Consult your local neighborhood Google for advice. 😊

Running Programs

- There are several different ways to run a program from the terminal.
 - `python3 program.py`**: Run a Python script.
 - `./program`**: Run a program in the current directory.
 - `command`**: Run a command installed on the system.
- CTRL+C interrupts (quits) a running program.
 - Oops, that's not "copy!"
- CTRL+Z stops (pauses) a running program.
 - Oops, that's not "undo!"
 - **`fg`** resumes a stopped program.

Pipes and Redirection

- ***command > filename***: Run the given command, writing the output to the given file.
- ***command | processor***: Run the given command and format the output.
 - ***less***: Use arrow keys to scroll.
 - ***sort***: Sort alphabetically.
 - ***sort -n***: Sort numerically.
 - ***head -n N***: Just show first *N* lines.
 - ***tail -n N***: Just show last *N* lines.
- There's a ***card game*** you can play to learn more about text processing with pipes.
- For a technical explanation of what pipes do, ***check this blog post*** by former Stanford student / CA / instructor Roz Cyrus.



Claude Code v2.1.280
Sonnet 5 · Claude Enterprise
~/projects/rain-music

> /clear

Create a website that simulates a rectangular puddle of water with raindrops falling onto it.

Generate music in real time whenever a raindrop hits the water and whenever two ripples touch one another.

```
Bash(ls -la /home/student/projects/rain-music/ 2>&1)
└─ total 16
   drwxrwxr-x 2 student student 4096 Sep 22 18:59 .
   drwxrwxr-x 4 student student 4096 Sep 21 20:55 ..
   -rw-r--r-- 1 student student 5784 Sep 21 23:20 CLAUDE.md
```

...

```
Bash(npx playwright test 2>&1 | tail -n 100)
└─ Running 5 tests using 1 worker

   ✓ 1 tests/rain-music.spec.js:8:3 › rain music puddle › loads with overlay gate and no console errors (618ms)
   ✓ 2 tests/rain-music.spec.js:30:3 › rain music puddle › clicking overlay starts audio and hides overlay (292ms)
   ✓ 3 tests/rain-music.spec.js:46:3 › rain music puddle › mute button toggles icon and slider changes intensity (278ms)
   ✓ 4 tests/rain-music.spec.js:62:3 › rain music puddle › heightfield changes over time as rain falls (canvas pixels vary) (1.7s)
   ✓ 5 tests/rain-music.spec.js:89:3 › rain music puddle › ripple collision fires the collision voice (no crash over time, drops accumulate and clear) (4.1s)

   5 passed (7.5s)
└─ (timeout 2m)
```

Managing Files

- **cp *from to***: Copy a file.
- **cp -r *folder to***: Copy a folder.
- **mv *from to***: Move a file or folder.
- **rm *file***: Permanently delete a file.
- **rm -r *folder***: Permanently delete a folder.

 ***Exercise Caution!*** 

Exercising Caution

- The terminal gives low-level access to your machine. It has fewer guardrails than what you might be used to.
- We've set up your vibecoding environment to add layers of protection to your system. Details later.
- Good rules of thumb:
 - **Never** run a command in the terminal if you don't know what it does.
 - Don't be afraid to ask for help if you need it! That's how you learn.
- Think of it like learning to cook.
 - Knives are sharp. You can cut yourself.
 - You probably won't if you're careful.
 - You probably will if you trust a stranger on the internet with this "one cool knife trick." 😊

Time Out for Announcements!

Assignment 1

- Assignment 0 was due at 1PM today.
 - Forgot about that? Didn't do it? No worries. Just get it in ASAP.
- Assignment 1 is due 1PM Tuesday.
 - Set up your development environment!
 - Build something personalized!
- You should expect to get two emails from GitHub.
 - One invites you to the cs193v-students organization.
 - One adds you to a group within that organization.
- This is a one-time setup; you won't need to do again later in the quarter.
- Didn't get these emails even though you're enrolled? Let us know and we'll get it straightened out.
- The TAs will hold some office hours over the weekend and at the start of next week if you need any help.

Back to CS193V!

Security Concerns

Security Concerns

- You can destructively modify your system from the terminal:
 - Permanently delete files (oops!)
 - Run malicious programs (oops!)
 - Send files from your machine elsewhere (oops!)
- Coding agents run commands on your machine through the terminal.
- LLMs famously hallucinate and make mistakes.



- ***Is this safe?***

Constraining Coding Agents

- There are many ways we can limit what a coding agent can and cannot do.
- We'll explore two parallel routes for doing this; my personal recommendation is a combination of the two.
- ***Route 1:*** Limit the agents from within.
- ***Route 2:*** Limit the agents from without.

Constraining Coding Agents

There are many ways we can limit what a coding agent can and cannot do.

We'll explore two parallel routes for doing this; my personal recommendation is a combination of the two.

- ***Route 1:*** Limit the agents from within.

Route 2: Limit the agents from without.

Coding Agent Modes

- ***Manual Mode:*** The coding agent won't run any commands without first asking you for your approval.
- ***Pros:*** You have full control over what the agent does on your system and can block anything you don't want it to do.
- ***Cons:*** Need to know what all the commands do, LLMs run complex commands, people make mistakes, and “permission fatigue” is real.
 - Someone created *a game* to see what people approved or denied, then did *a writeup* showing how many folks got it wrong. (Spoiler: a lot of people messed up!)

Coding Agent Modes

- ***Auto Mode:*** An independent LLM auditor (“classifier”) reviews commands before they’re run and refuses anything it considers dangerous.
- ***Pros:*** Eliminates human review; LLMs classifiers are generally pretty good at not doing the wrong thing.
- ***Cons:*** LLMs can make mistakes, and if the classifier is wrong Bad Things can happen.
 - *Quis custodiet ipsos custodes?*

"Auto mode lets Claude handle permission prompts automatically [...]"

Claude can make mistakes that allow harmful commands to run, it's recommended to only use in isolated environments."

"Auto mode lets Claude handle permission prompts automatically [...]"

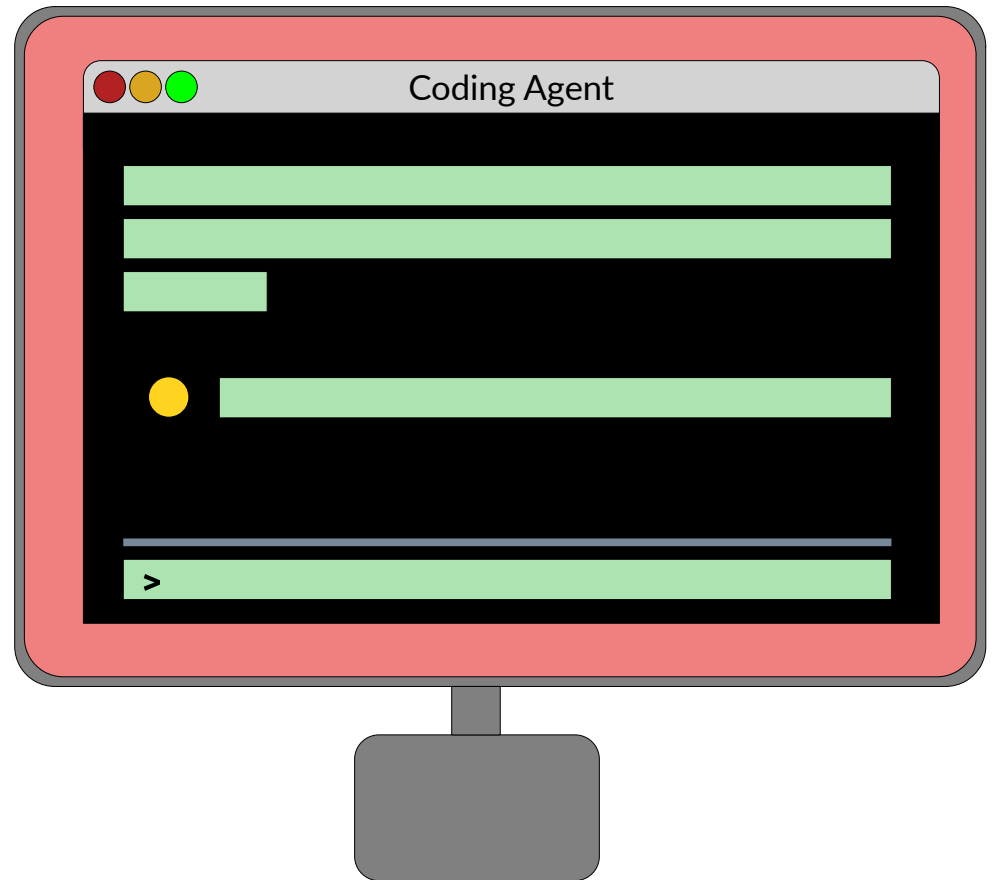
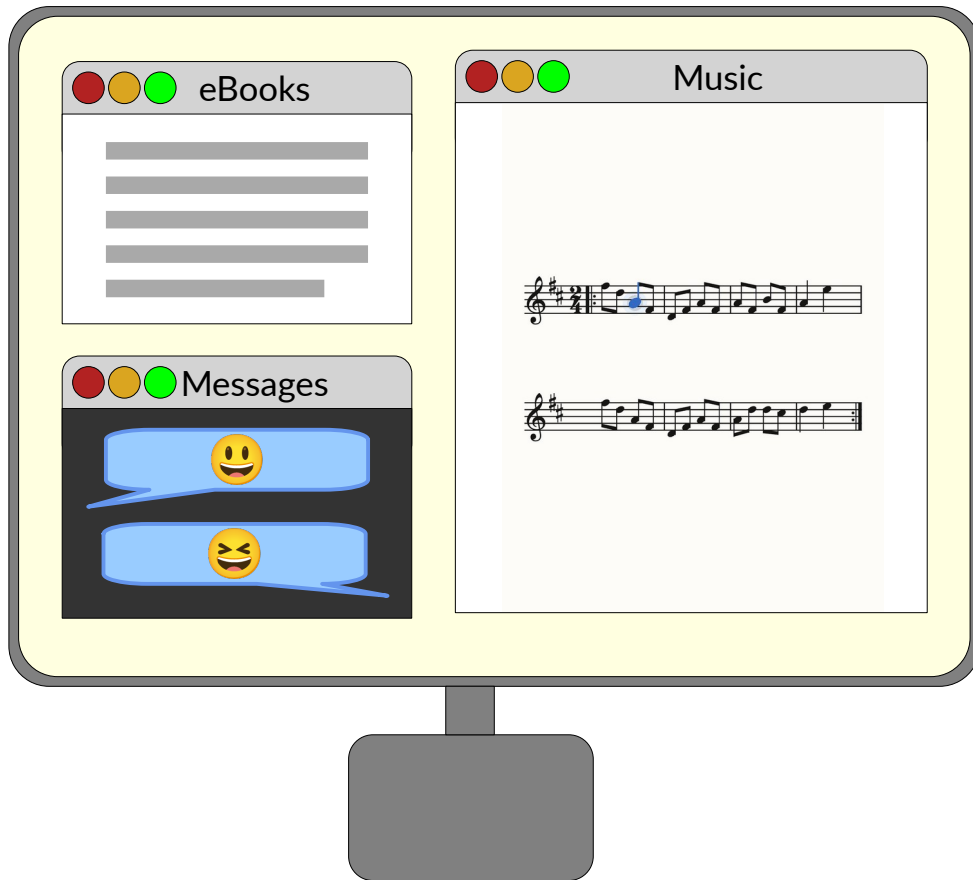
Claude can make mistakes that allow harmful commands to run, it's recommended to only use in isolated environments."

Isolating Agents

- Running a coding agent on your machine carries some risks.
- We ***isolate*** a program by making sure it can't interact with / tweak / break / ignite files and computers we haven't explicitly authorized it to touch.
- How might we do this?

Isolating Agents

- **Option 1:** Run your coding agents on a fully separate computer from your normal one.

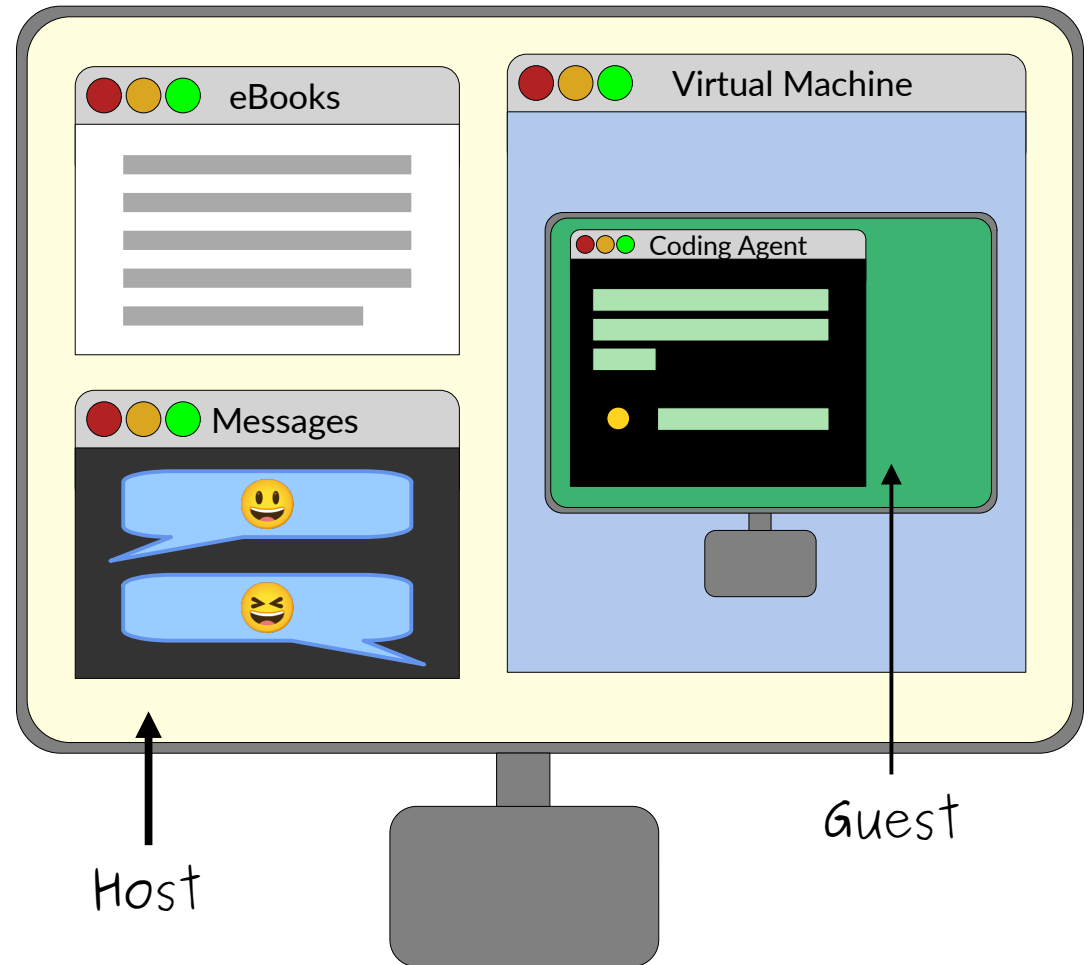


Isolating Agents

- **Option 1:** Run your coding agents on a fully separate computer from your normal one.
- **Pros:** The coding agent isn't on your computer, so it can't touch your files or mess up your settings.
- **Cons:**
 - *Expensive:* You need a physically separate computer, or you need to rent a computer somewhere.
 - *Inconvenient:* Sometimes you really *do* want a coding agent to do something with files on your computer.
- My advice: Don't do this. There are better alternatives.

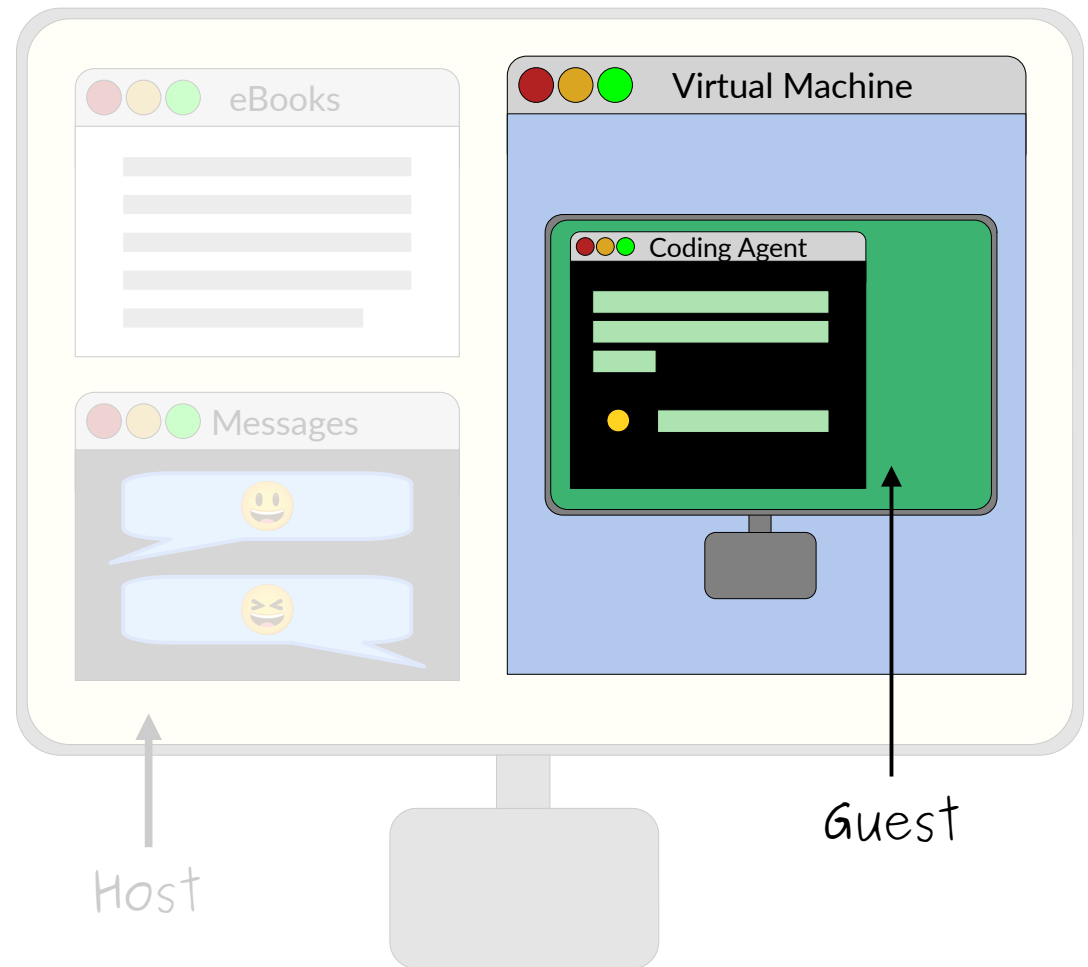
Virtual Machines

- **Option 2:** Run your coding agent in a virtual machine (VM).
- A **virtual machine** is a program *simulates* an entirely separate computer, including a separate operating system and hard drive.
- The real computer is the **host**; the simulated computer is the **guest**.



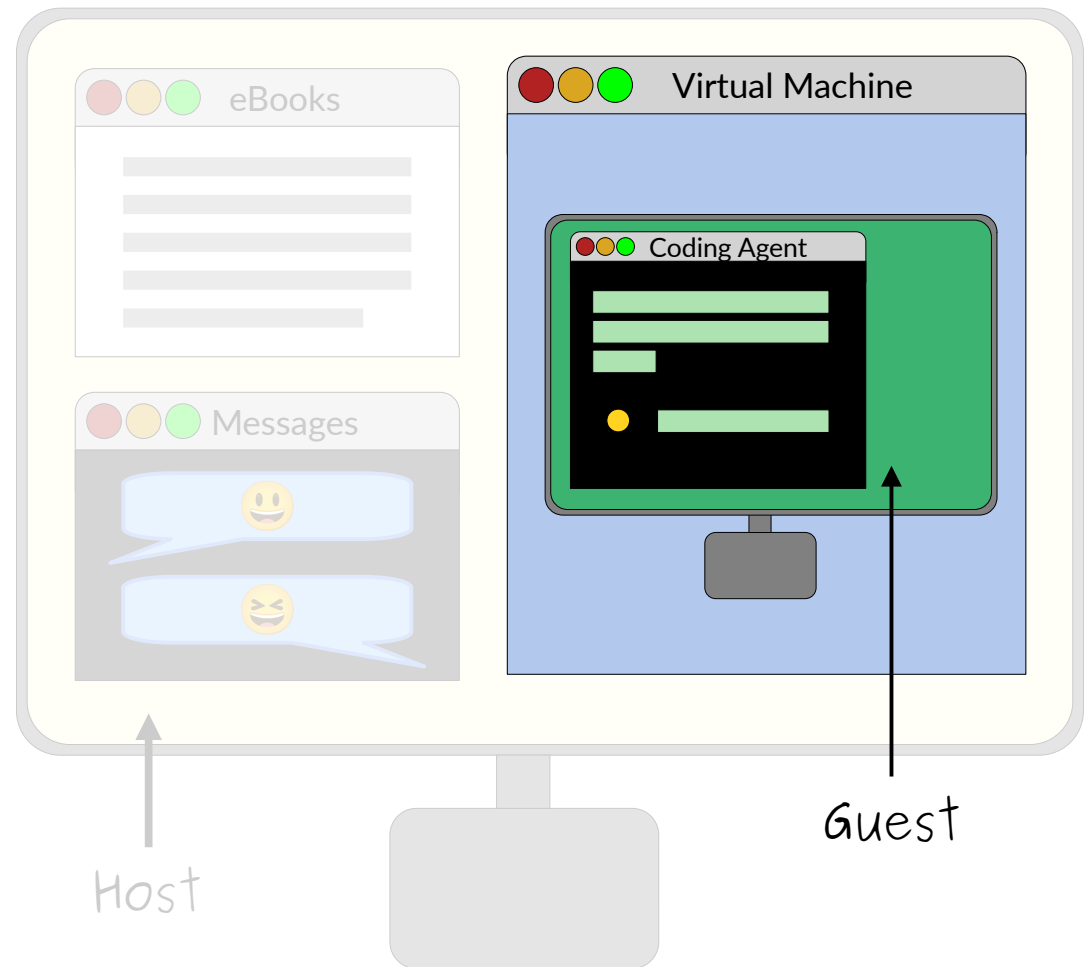
Virtual Machines

- From the *outside*, a the guest looks like any other program on your computer.
- From the *inside*, code running in the VM thinks it has its own dedicated computer to work with.



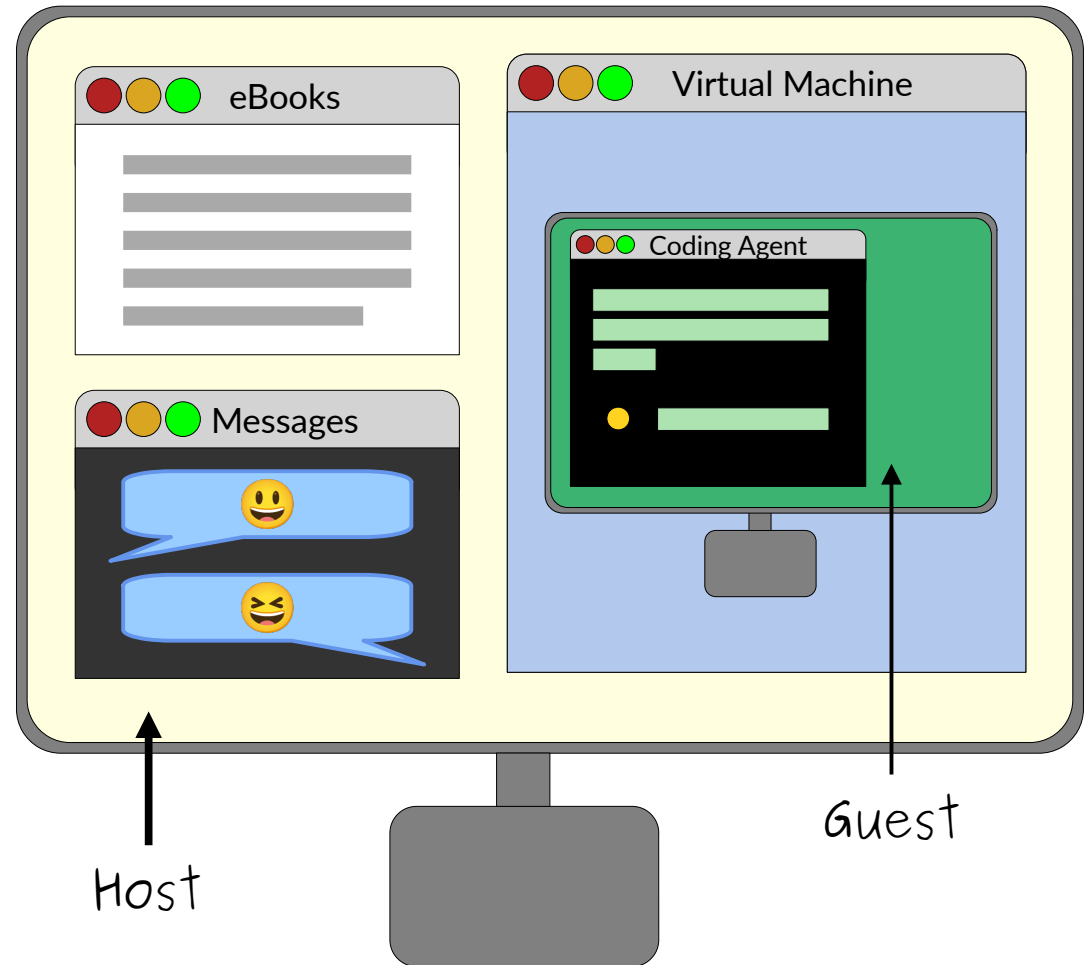
Virtual Machines

- Suppose the guest runs some commands.
- From *its* perspective, it's running commands on a real machine.
- From *our* perspective, it's just making changes to a pretend (virtual) computer.
- The host machine is fully isolated from what happens in the guest – just like if we had a separate computer!



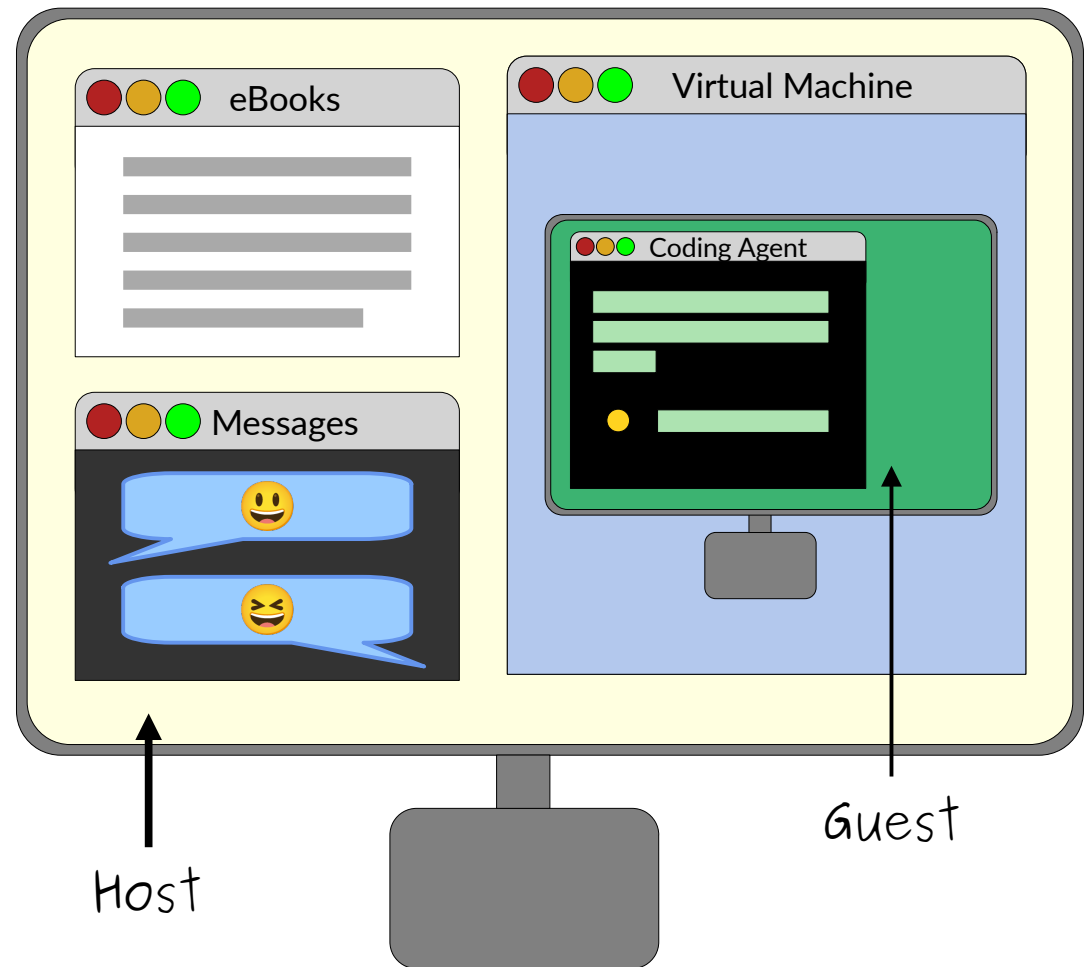
Virtual Machines

- VMs have some drawbacks.
- ***Speed***: They don't run as quickly as a real computer.
- ***Memory (RAM)***: They consume a *huge* amount of system memory.
- ***Disk***: They use a *ton* of disk space.
- Why?



Virtual Machines

- The resources from the pretend (virtual) machine have to come from somewhere.
- It needs a virtual hard drive. It stores that virtual drive on your hard disk.
- It needs RAM, which it borrows from your computer.
- It needs to run its own OS, which requires your CPU.



How do we get the benefits
of a VM without paying the cost?

Containerization

- *Containerization* is
 - *virtualization* of the
 - *operating system* rather than hardware.



- Don't worry! Let's break that down.

Containerization

Containerization is

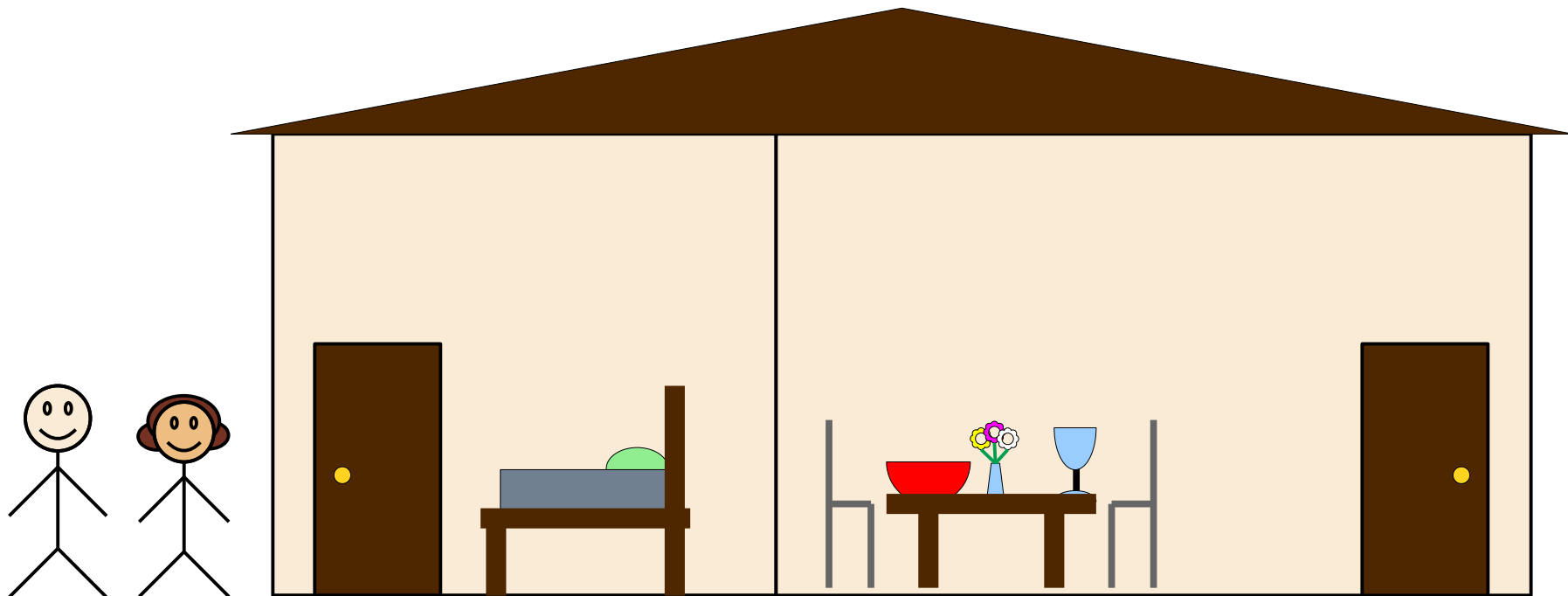
- ***virtualization*** of the *operating system* rather than hardware.



Don't worry! Let's break that down.

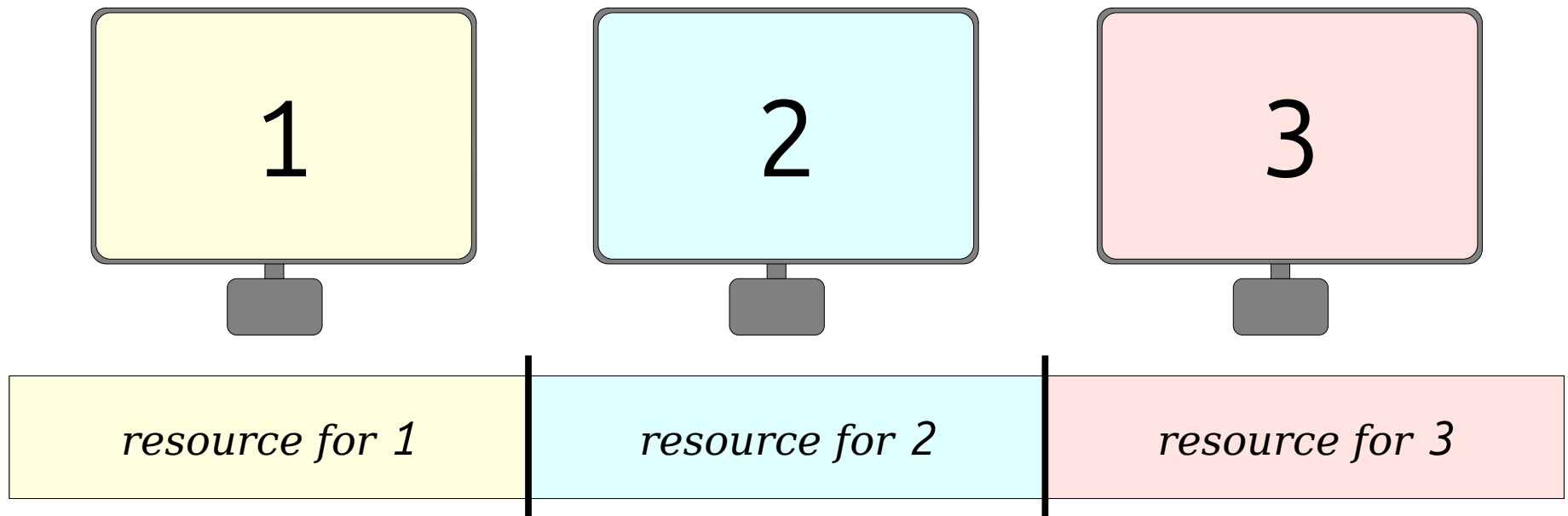
Virtualization

- **Virtualization:** Dividing up resources across multiple users so
 - from the *outside* each user only gets a slice of the resources, but
 - from the *inside* each user thinks it has all the resources.



Virtualization

- **Virtualization:** Dividing up resources across multiple users so
 - from the *outside* each user only gets a slice of the resources, but
 - from the *inside* each user thinks it has all the resources.



Containerization

- *Containerization* is
 - *virtualization* of the
 - *operating system* rather than hardware.



- Don't worry! Let's break that down.

Containerization

Containerization is

virtualization of the

- ***operating system*** rather than hardware.



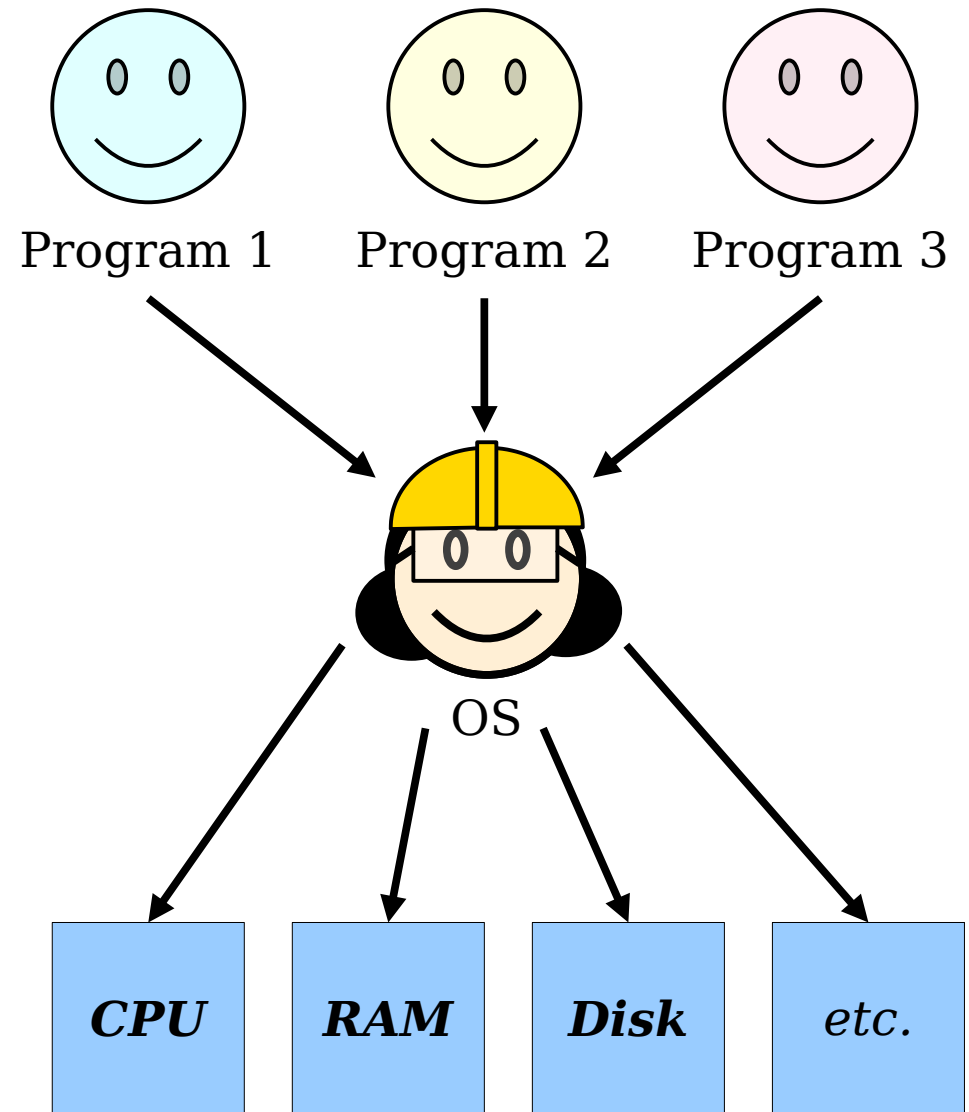
Don't worry! Let's break that down.

Operating Systems

- You might be familiar with an operating system in an informal sense.
 - Programs running on macOS *look different* and *respond differently* than programs running on Windows.
 - Some software only runs on certain operating systems (e.g. iOS apps won't run on a Windows desktop computer).
- But what exactly *is* an operating system?

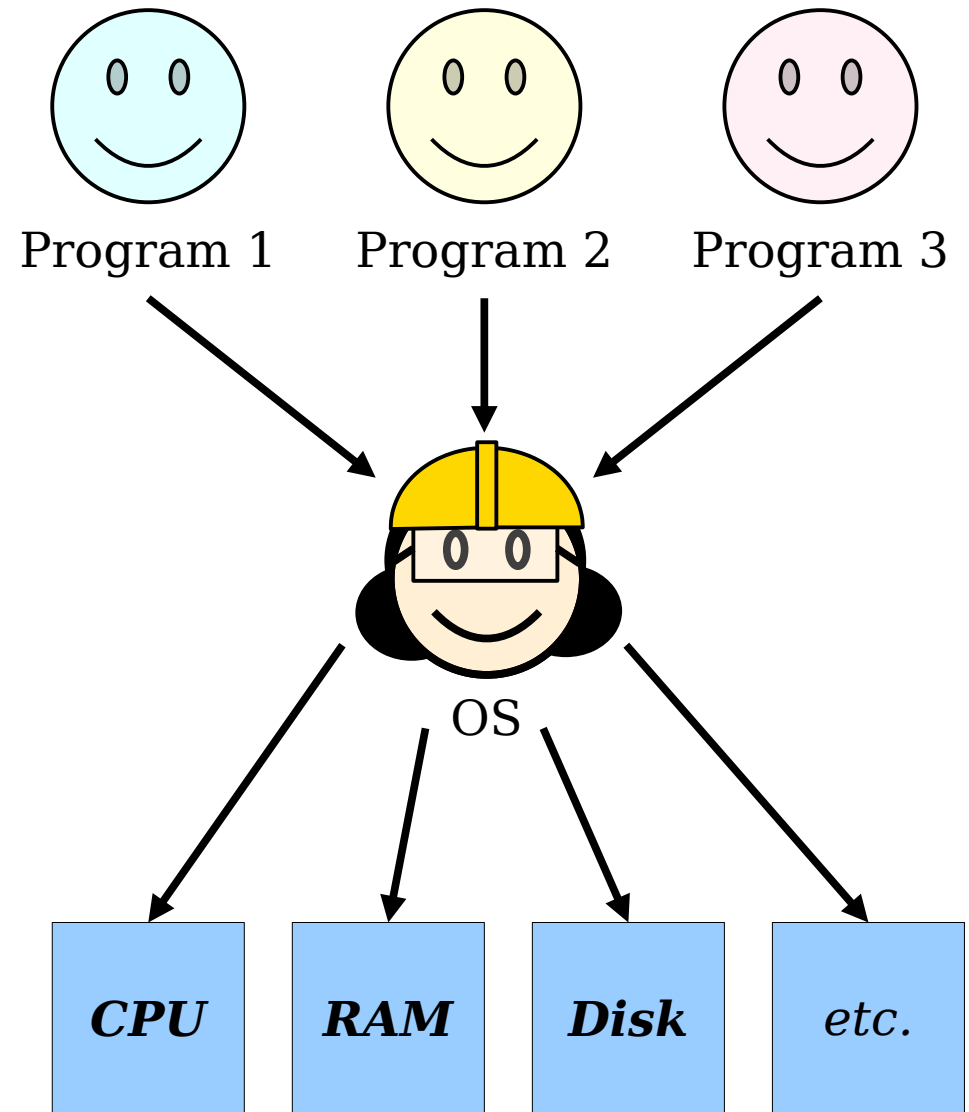
Operating Systems

- An ***operating system*** is the software on your computer that is responsible for managing hardware resources.
- Think of it as a “conductor” that determines which programs get to use system resources.



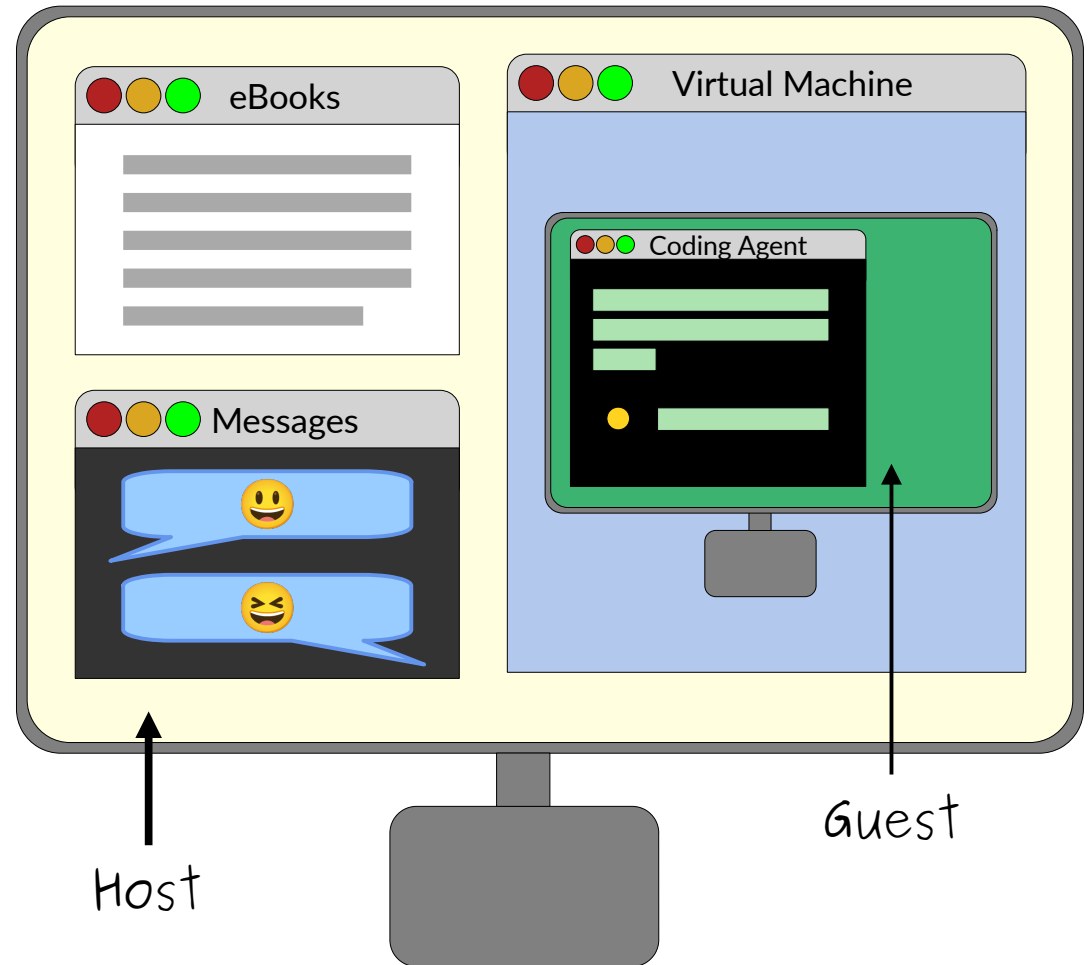
Operating Systems

- Your OS also does access control.
 - Only admins can install software.
 - Untrusted programs can't access your desktop without your permission.
 - etc.
- So it's a “bouncer” in addition to a “conductor.”

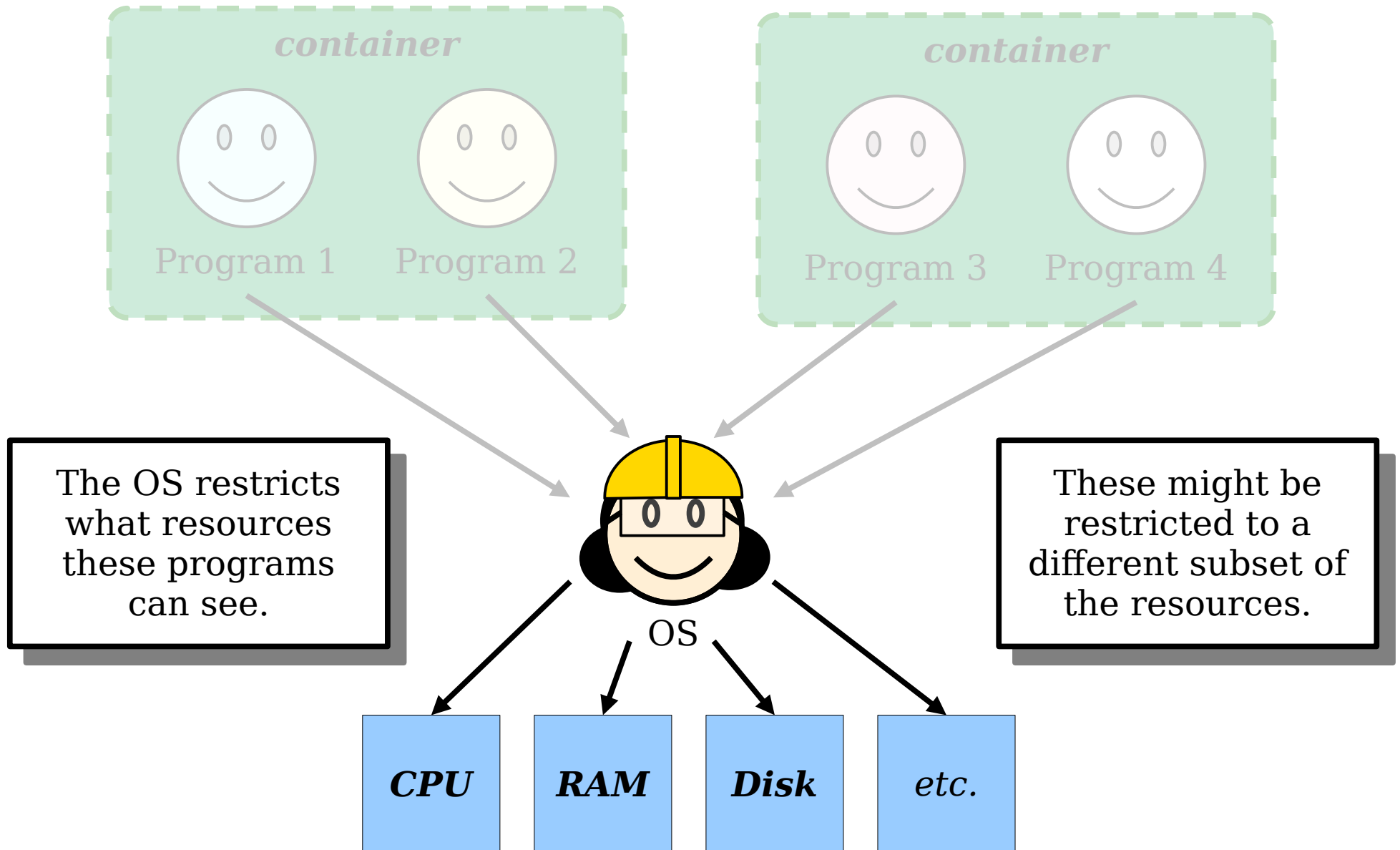


Operating Systems

- In a VM, the host and guest run *independent* operating systems.
- This has some benefits:
 - The guest OS can be anything (run Windows on macOS, for example).
 - The guest can't touch the host OS, which boosts security.
- And some drawbacks:
 - Two different OSes each spend time, memory, and disk managing computers (one real, one pretend).
- That duplication is not free, and it's one reason why VMs use lots of time and resources.



Containerization



Containers vs VMs

- Containers are *faster*, use *less RAM*, and use *less disk* than VMs.
 - There's only one OS running and it can manage resources for programs both inside and outside the container.
- VMs offer *more security* than containers.
 - Operating systems contain bugs! A bug in an OS can let something inside a container get access to the full machine. (Search “container escape.”)
 - VMs also can contain bugs, but VMs are less complex than a full OS and thus have fewer of them. (Search “VM escape.”)

My Recommendation

- Use ***Auto Mode*** as a primary line of defense against AIs making mistakes, and ***containers*** as secondary line of defense.
 - Auto Mode will restrict most, but not all, potentially bad commands.
 - The container – through the operating system – guards against almost all accidents.
- This is what the development environment we've set up for you will do.
- And on that note...

The CS193V Development Environment

The CS193V Dev Environment

- We provide a development environment you can use to run coding agents inside a container.
- It comes with Claude Code, OpenAI Codex, and a bunch of other software preinstalled.
- It runs inside a container, so those agents can't see files on your machine unless you explicitly give them access.
- ***Demo time!***

To Summarize

- The terminal gives us a way to control a computer using text commands, and it's what LLMs use.
- You only need a few basic terminal commands to move around your computer, copy and move files, run programs, and read text files.
- LLMs can run commands on your machine, which comes with some security risks.
- Virtual machines simulate entire computers and use a lot of resources.
- Containers provide isolation while getting the speed benefits of your host operating system.

Next Time

- ***Version Control***
 - Making backups and experimental copies of software, all in one tool!
- ***GitHub***
 - Backing up your code In The Cloud, with so *many* more features.