Snowcast ~~Warmup~~ GEARUP

A mental model

Snowcast: an "Internet radio station"

- Server: has several "stations" that serve audio data to clients
- Clients: connect to server, ask for a station, receive audio data
  - (Actually two programs, more on this later)

"Radio station"

- Server is always "playing" music, even if no one is listening
- Everyone gets the same data at the same time

A mental model

Snowcast: an "Internet radio station"

- Server: has several "stations" that serve audio data to clients
- Clients: connect to server, ask for a station, receive audio data
  - (Actually two programs, more on this later)

"Radio station"

- Server is always "playing" music, even if no one is listening
- Everyone gets the same data at the same time

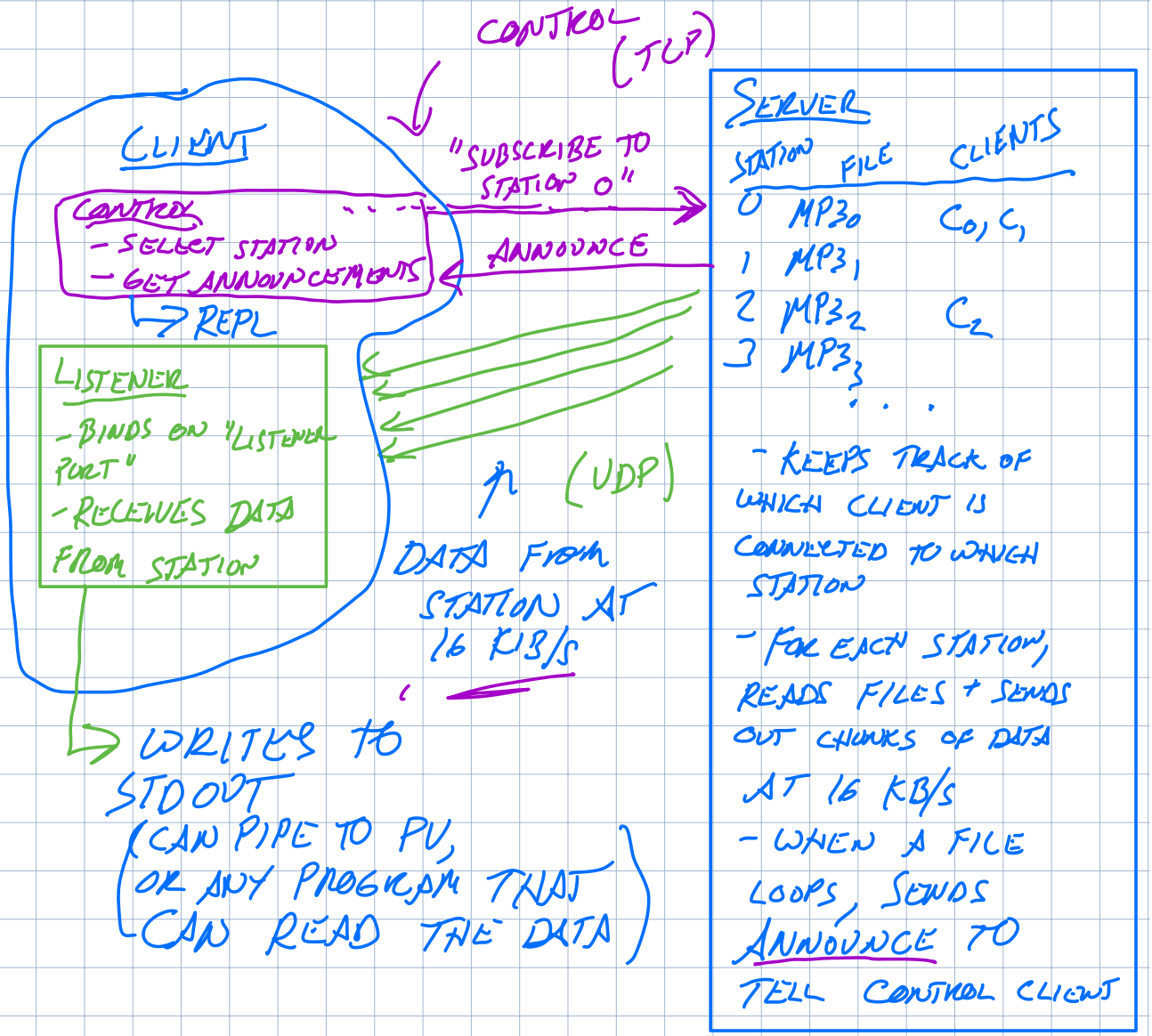
Not like Spotify. More like Pandora or iHeartRadio.

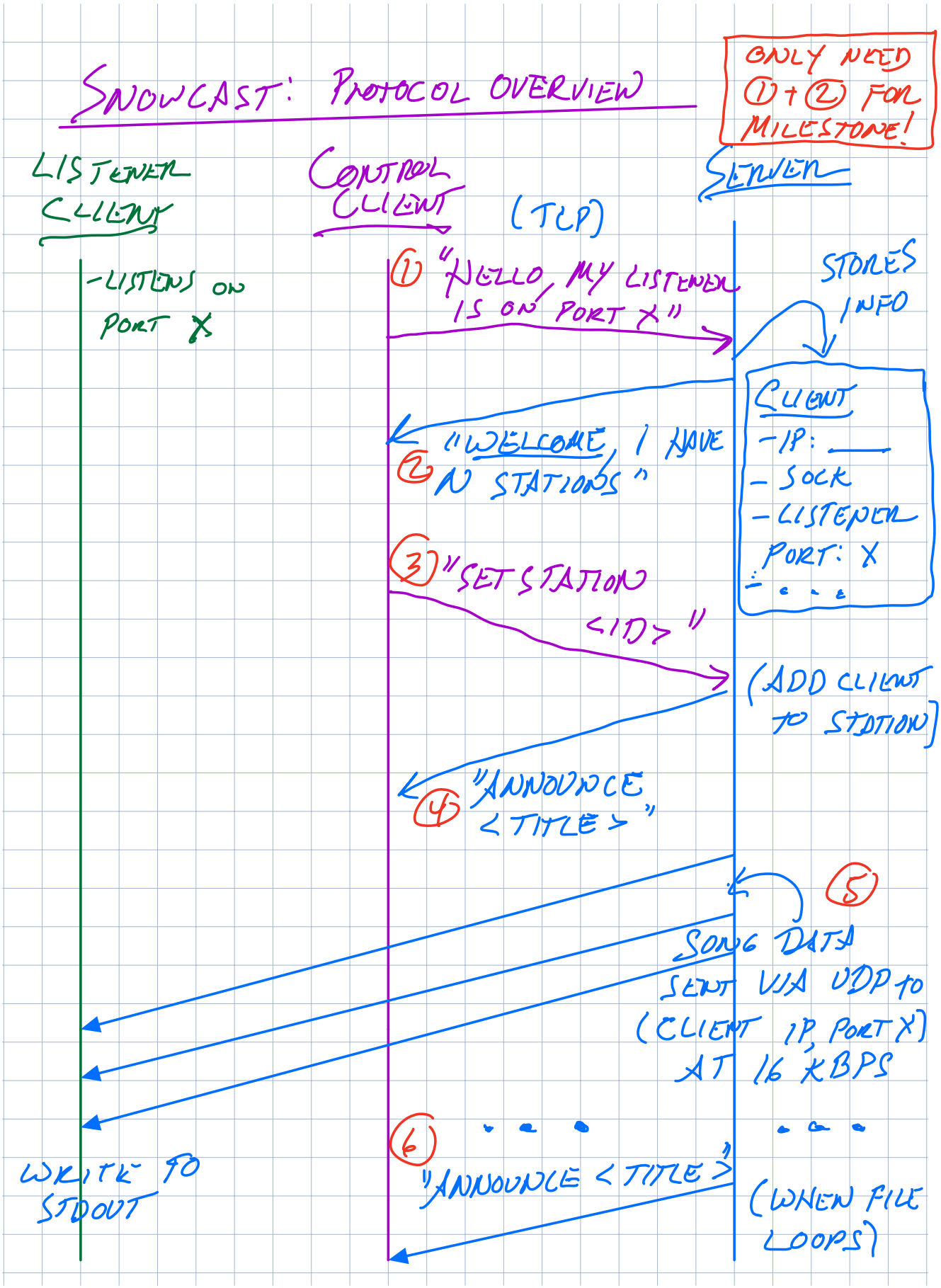
TWITCH

Goals

- Intro to socket programming
- Chance to become more comfortable with socket programming, in any language
- Learn how to implement a protocol, design a robust server

THINKING ABOUT THE ARCHITECTURE FOR SNOOCAST




SNOWCAST: PROTOCOL OVERVIEW
ONLY NEED ① + ② FOR MILESTONE!
LISTENER CLIENT
CONTROL CLIENT
(TCP)
SERVER
-LISTENS ON PORT X
① "HELLO, MY LISTENER IS ON PORT X"
STORES INFO
CLIENT
-IP: ...
-SOCK
-LISTENER PORT: X
...
② "WELCOME, I HAVE N STATIONS"
③ "SET STATION <ID>"
(ADD CLIENT TO STATION)
④ "ANNOUNCE <TITLE>"
⑤
SONG DATA SENT VIA UDP TO (CLIENT IP, PORT X) AT 16 KBPS
...
⑥ "ANNOUNCE <TITLE>"
...
(WHEN FILE LOOPS)
WRITE TO STDOUT

WHEN BUILDING A NETWORKED APPLICATION

- WHAT STATE DOES THE SERVER STORE?
  - SHARED DATA! HOW TO PROTECT IT?
- WHAT STATE DOES THE SERVER NEED PER CLIENT?
- HOW TO HANDLE MULTIPLE CLIENTS?

EG. FOR THE GUESSING GAME EXAMPLE...

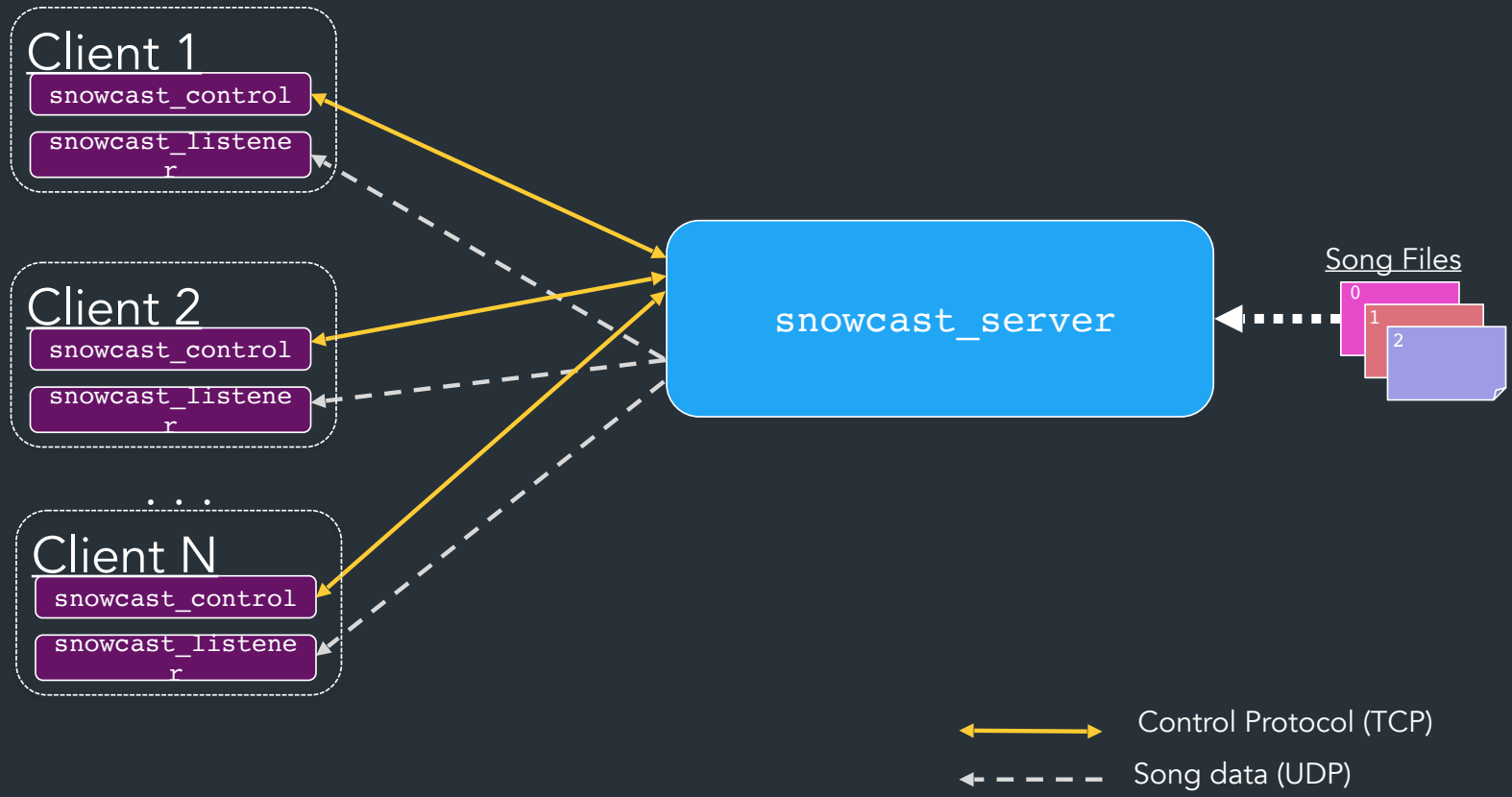
- NUMBER WE'RE TRYING TO GUESS
- TOTAL NUMBER OF GUESSES
- ...

- CLIENT SOCKET
- MAYBE:
  - UNIQUE ID, GUESS HISTORY...

- ONE GOROUTINE ON SERVER FOR EACH CLIENT
- SERVER KEEPS LIST OF CLIENTS TO NOTIFY WHEN GAME RESETS

FOR YOUR DESIGN DOCUMENT, THINK ABOUT HOW YOU WOULD DO THIS FOR SNOWCAST!

Concretely: how Snowcast works



What you will implement

- You will implement all three programs
  - `snowcast_server`: the server
  - The client (two programs):
    - `snowcast_control`: Control client
    - `snowcast_listener`: Listener client
- We give you the specification for the protocol, and how the programs should behave
- You decide how to implement them

What you will implement

- You will implement all three programs
  - `snowcast_server`: the server
  - The client (two programs):
    - `snowcast_control`: Control client
    - `snowcast_listener`: Listener client
- We give you the specification for the protocol, and how the programs should behave
- You decide how to implement them

Need to be able to interoperate with our reference version (and tests)!

Roadmap

- Setup <--- you are here
- Milestone: Sending initial messages (welcome/hello)
- Building your server (where to put state, etc.)
  - Subscribing to stations
  - Listing clients
- Listener + streaming ⇒ IMPLEMENTATION GUIDE.
- Announcements while streaming (WHEN FILE LOOPS)

Roadmap

- Setup <--- you are here
- Milestone: Sending initial messages (welcome/hello)
- Building your server (where to put state, etc.)
  - Subscribing to stations
  - Listing clients
- Listener + streaming
- Announcements while streaming
- Error handling/timeouts/etc

What we will test

- Your programs must interoperate with ours (ie, speak the same protocol)
- Don’t need to stream music—we just measure for a streaming rate of ~16KiB/s

- Some server design guidelines (see spec)
  - Must support multiple clients, protect shared data
  - Reasonable error handling (+timeouts)

Languages

You can work in any of the following languages:

- Go
- C/C++
- Rust

SEE
– A TOUR OF GO
– GO BY EXAMPLE.

We recommend Go, even if it is new to you.

The time to learn go will be less than the time you'd otherwise spend debugging stuff in C.

Assignment structure

- Milestone
  - Warmup: guide to sending your first few messages and inspecting them in Wireshark
    - Should pass the "milestone tests" in your repository
  - Design doc: tell us how you plan to design the rest of the system

- Final submission: your code + a README explaining your major design decisions

Resources

- The Handout
  - Protocol specification: what messages should look like
  - Implementation specification: how programs should be have (command line arguments, etc.)

- Warmup/Implementation guide
  - Implementation-level resources: FAQs, how to run tests, how to use wireshark, etc.

- Test suites: you can run our autograder tests!
- Lecture examples: don't copy, but look at them side by side

Language resources

- Language resources on website
- Go: will post more on channels
- Some utilities for C (linked list, hash table)

Libraries

- You can use libraries you find online (go packages, rust crates, etc), as long as it doesn't trivialize the assignment
- You must manually parse packets on your own
- Easy examples: argument parsing, logging, …

If you're unsure (especially networking-related stuff), please ask!

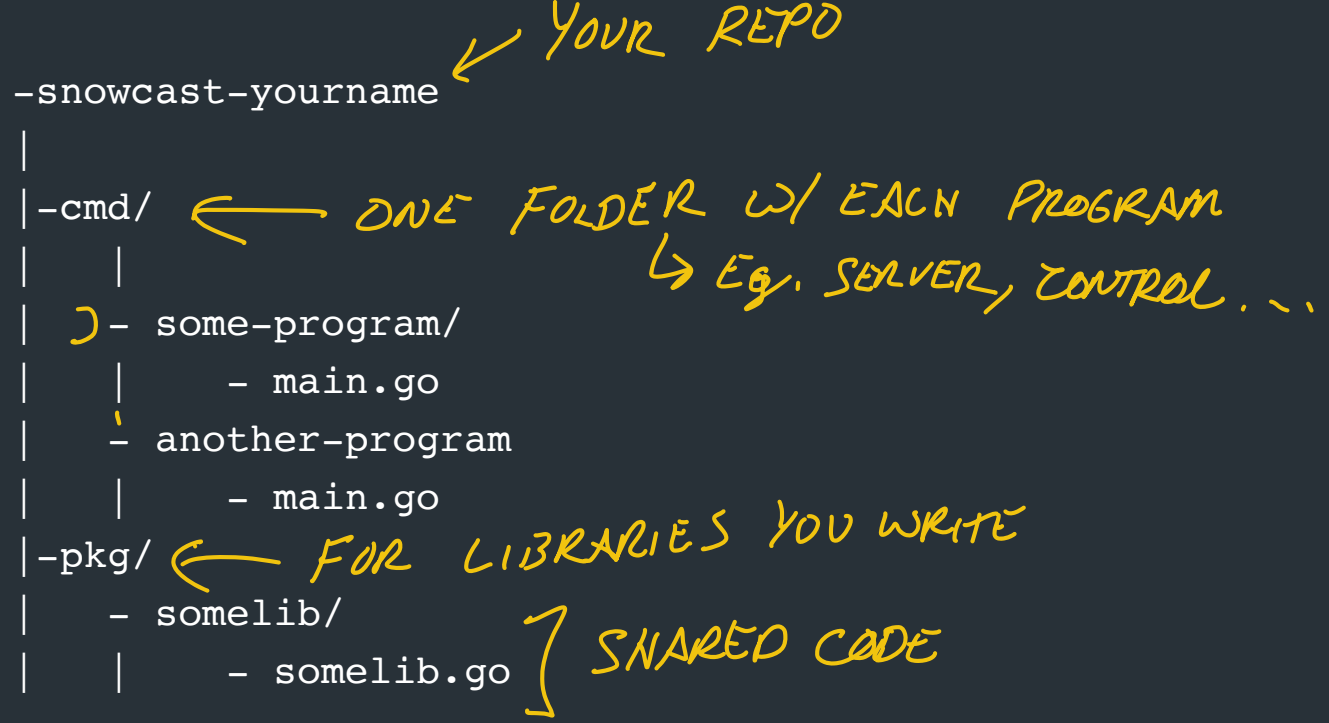
How to get started

Dev environment

- You should be working in the container environment
- Be sure to clone your repo where you can access it from the container

```
- ...
|--DEV-ENVIRONMENT
| |--docker/
| |--home/
| | |--snowcast-yourname/
| |--run-container
| |-- ...
...
```

How to start a go project



The tester

We have provided a test suite with all of our tests

- Check your work as you go, see it in Wireshark
- We'll have the same test suite available in gradescope soon

  ⇒ Can use to make sure you don't have any compatibility issues
- Want to know what a test does? See the list of tests!

If you are failing a test and don't know why, see the "what to do if you have a failing test" section of the warmup/implementation guide