Successfully System Implementation Strategies

Mar 19th, 2021

Overview

- Understand the Concepts and Code Structure
- Iterative Design Process
  - Start Simple, then Build Up
- Modular Programming
- Tips on Debugging

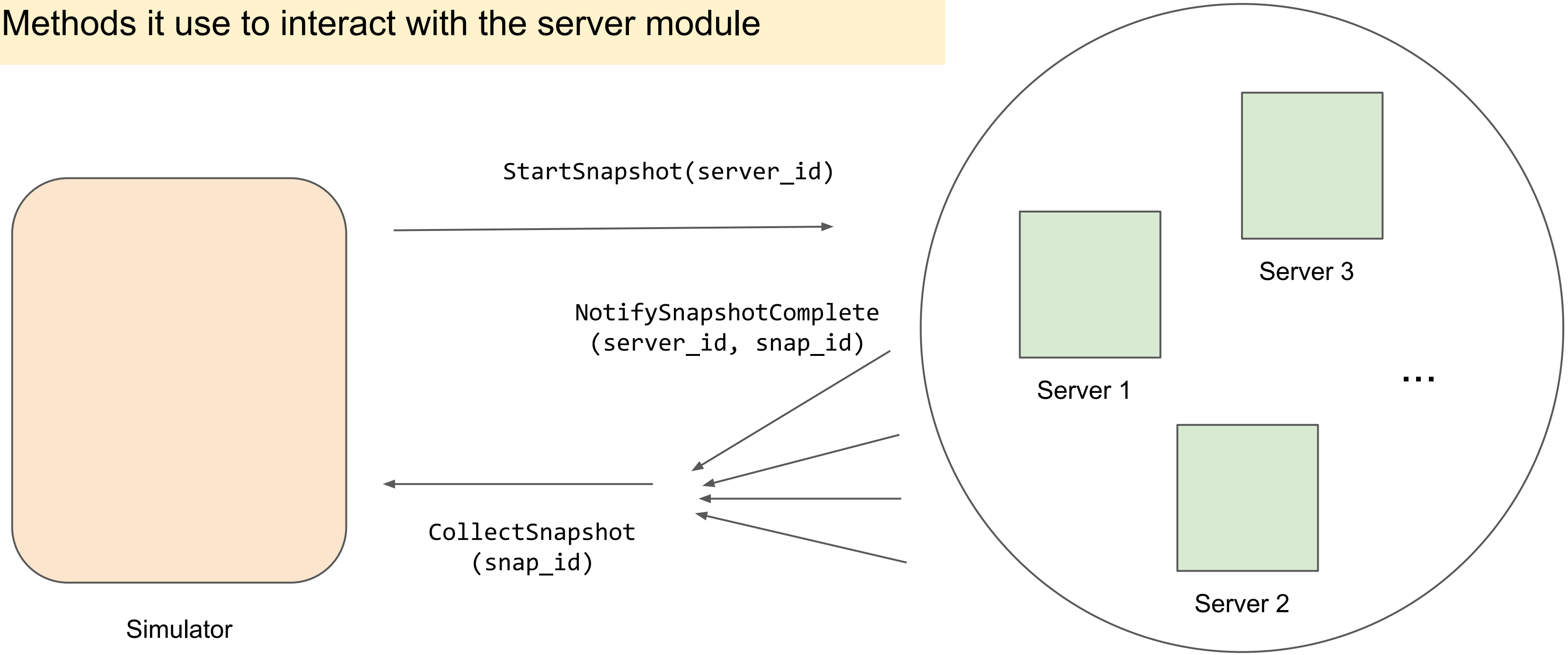
Understand the Concept and Code Structure

- What is the conceptual system you want to build?
    - Understand the concept and verify your knowledge with some examples
    - Rewrite the algorithm to some pseudocode, which can serve as the guide during actual programming
- How is the system physically built?
    - Read the skeleton code
    - Map the algorithms/concepts to the given code structure
    - Draw flow charts to understand of code flow
- How to use the system?
    - Read the testing script to see how an external user will talk to our system and invoke its APIs to accomplish desired tasks

How is the System Physically Built?

Understand the simulator's implementation (see *simulator.go*)

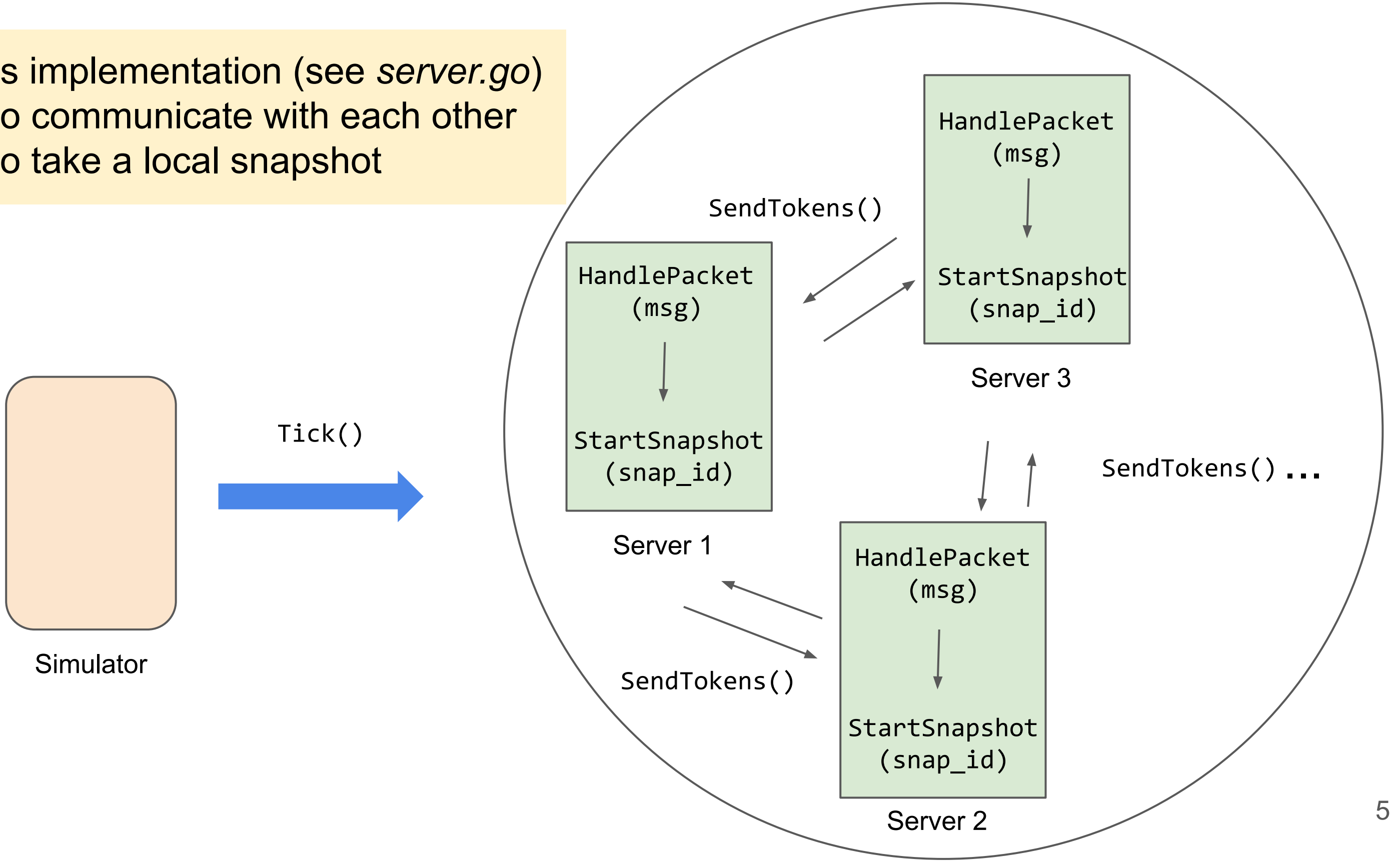
- The role of the simulator
- Methods it use to interact with the server module



How is the System Physically Built?

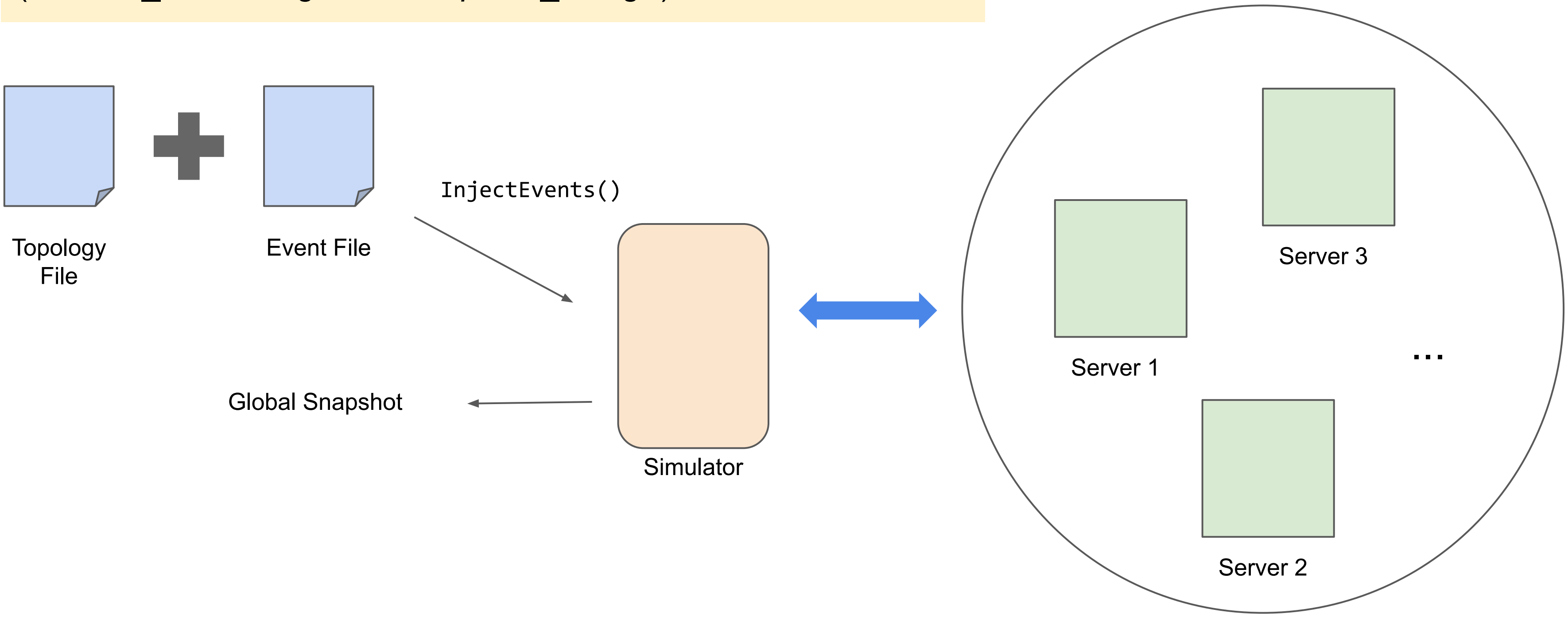
Understand the server's implementation (see *server.go*)

- Methods it uses to communicate with each other
- Methods it uses to take a local snapshot



How to Use the System?

Understand how the external environment talks to our system (see *test_common.go* and *snapshot_test.go*)



Understand Concept and Code Structure

Summary

- Fully comprehend the algorithm
- Spend time to map your understanding of the concept to the starter code
    - For both the system interface and individual modules, understand **what** data is transferred between and **how**
- Charts and pseudocode can help A LOT!

Iterative Design Process

Common design methodology in product design, including software design

You will understand a little more about your design when you start implementing it.

- Start with the base case (aka simplest case)
    - Example: one global snapshot at a time for Assignment 2, distributed MapReduce without any failure for Assignment 1.3
- Test regularly: should pass test case for 2 nodes, then 3 nodes and …
- Add one more complexity at a time

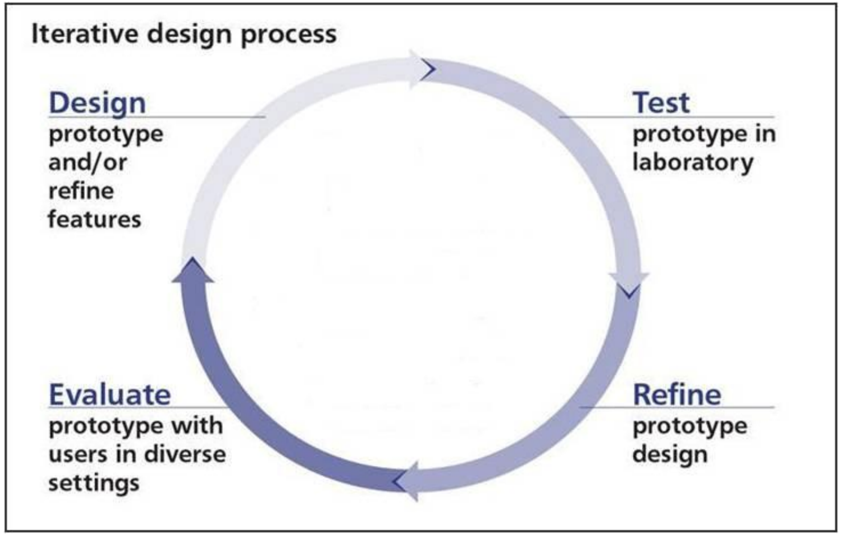
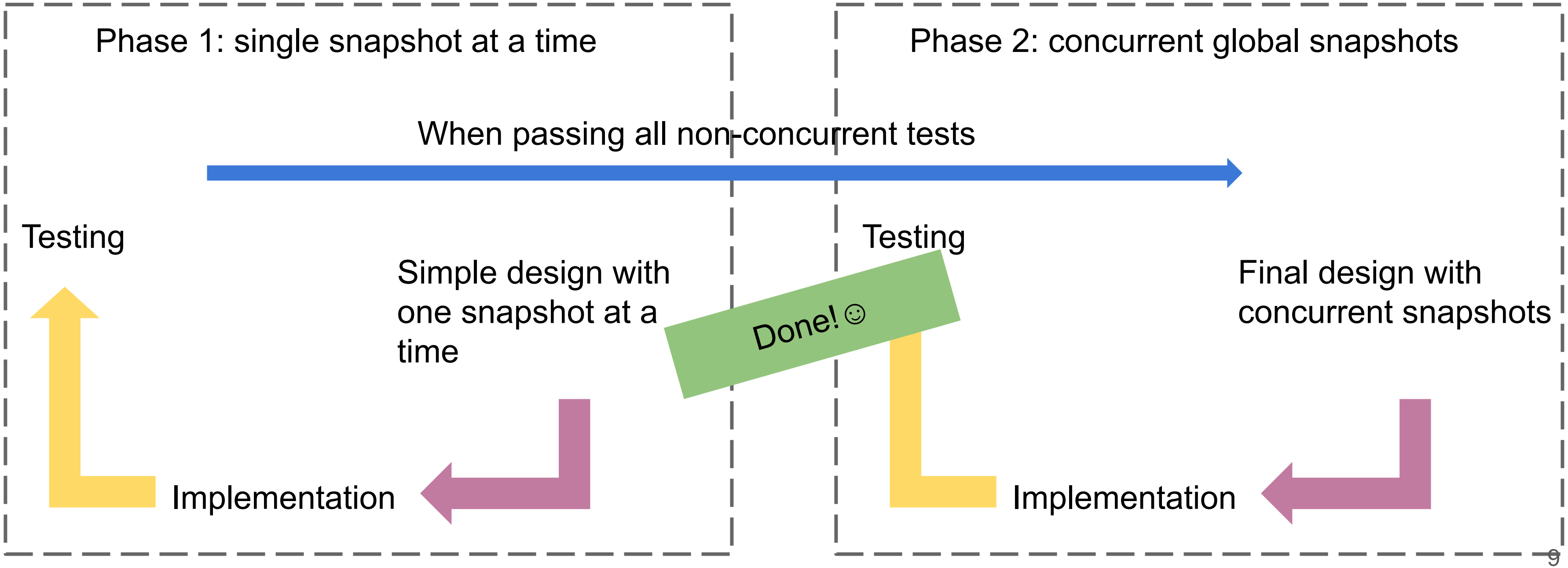


Image Source from the Internet

Iterative Design Process: Distributed Snapshot

Key Idea: Start Simple, then Build Up



Phase 1: single snapshot at a time

Phase 2: concurrent global snapshots

When passing all non-concurrent tests

Testing

Simple design with one snapshot at a time

Implementation

Done! ☺

Testing

Final design with concurrent snapshots

Implementation

Modular Programming

Iterative design means code change every time when refining the design ☹

Modular programming

- Decompose the system into several independent modules/pieces
- Use a set of simple yet flexible APIs for intra-module communication

Advantages of modular programming

- Makes it easier to reason about and debug each component of your system
- Requires minimal change in the code

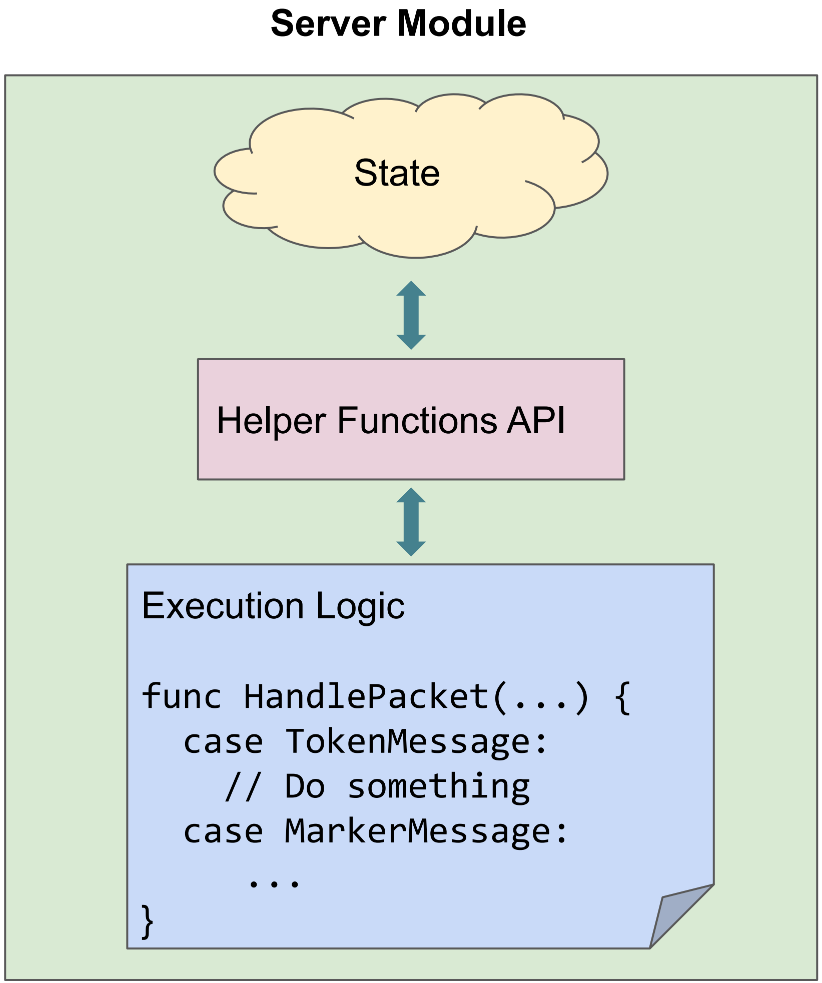
Modular Programming

Phase 1: single snapshot at a time

Divide our server module into 3 pieces:

- Server State
- Execution logic
- A layer of helper functions

Goal: write a flexible layer of helper functions

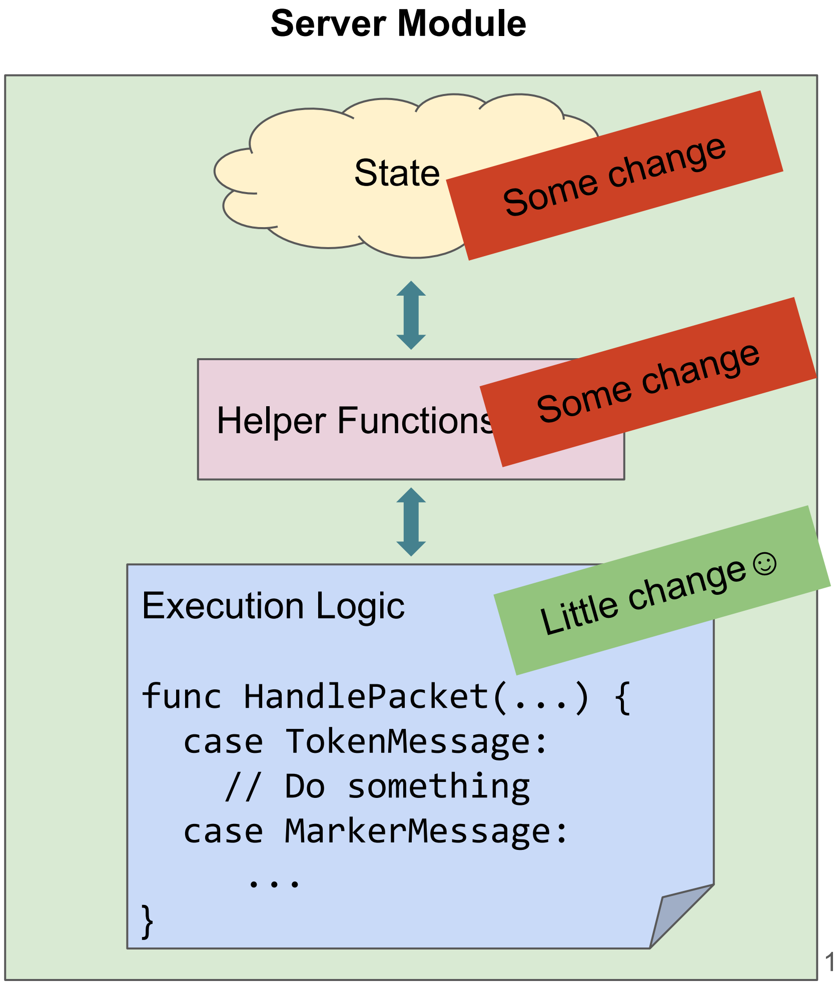


```
func HandlePacket(...) {
  case TokenMessage:
    // Do something
  case MarkerMessage:
     ...
}
```

Modular Programming

Phase 2: concurrent snapshots

- Update the state variables and helper functions' implementation
- Keep the API and execution logic unmodified (almost)



```
func HandlePacket(...) {
  case TokenMessage:
    // Do something
  case MarkerMessage:
     ...
}
```

Tips on Debugging

- Start Early!
- Commit your code to Git often and early, and every time when you pass a new test (enable comparative debugging later if necessary
- Have proper naming for variables and add comments in your code
  - Easier for both you and others to read and debug your code
- Take advantage of Go Playground if you are not familiar with any Go specifics
- Prints are your friend!

Prints Are Your Friend ☺

- Always verify the behavior of your program! Sometimes, it may not align with your expectation because of some hidden bugs.
- Track execution using printing statements to understand the code flow
  - Especially helpful in the early development of your design when the code complexity is not too high
- Help catch errors in the early stage
- Example
  - In Assignment 2, we can print out the server state before and after `HandlePacket()` and `StartSnapshot()` that you implement after each tick of the simulator