

# Distributed Snapshots

Spring 2025

# A Note on Channels and Goroutines

- Using channels is easy, debugging them is hard...

Bullet-proof way: Keep track of how many things go in and go out

Always ask yourself: is this channel buffered?

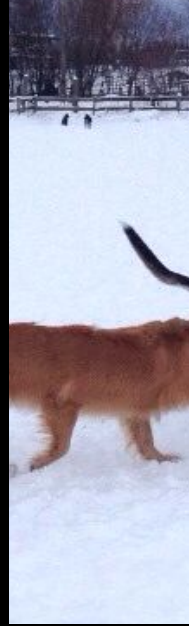
- In general, don't use locks or atomic operations with channels (awkward)
- Try not to nest goroutines (would be hard to reason about them)



N1



N2



N3



N4

Dog moves





# Distributed snapshots are hard

Must ensure **state is not duplicated** across the cluster (we only want one dog)

Must ensure **state is not lost** across the cluster (we still want the dog there)

**Thus we must carefully consider how and when exactly to record state.**



*Event order:*

- 1.Snap N1
- 2.N2 sends body
- 3.Snap N2
- 4.N1 receives body

**Should record  
message!**



*Event order:*

- 1.N2 sends body
- 2.Snap N2
- 3.N1 receives body
- 4.Snap N1

**N1 already received the body in  
step 3**

**Should NOT record message**

# Intuition: guarantee zero loss + zero duplication

If you *haven't* snapshotted your local state yet:

- *Do NOT* record future messages you receive

If you *have* snapshotted your local state:

- *Do* record future messages you receive

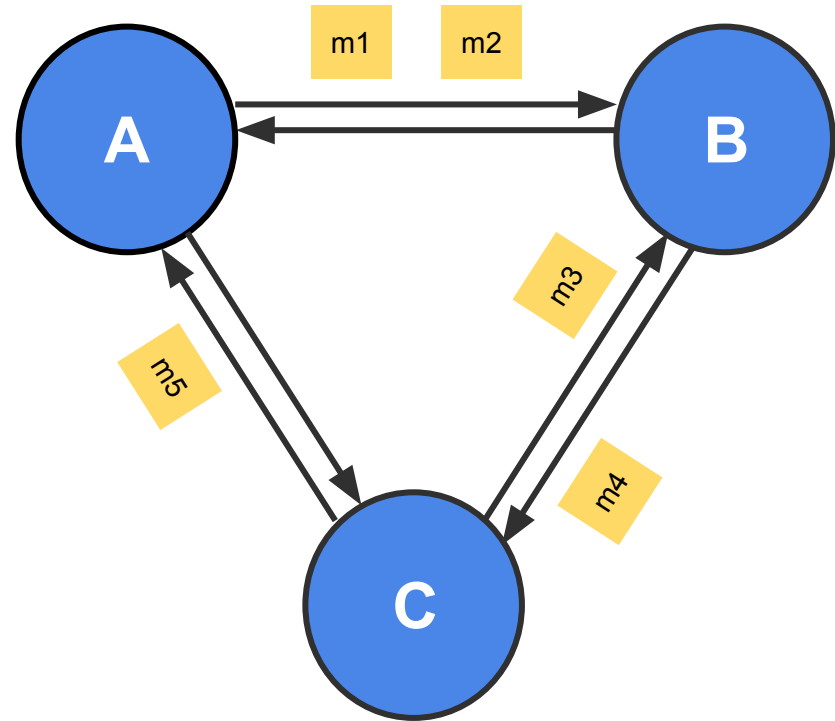
*Which one guarantees zero loss?*

*Which one guarantees zero duplication?*



# What is a Global Snapshot?

- A global snapshot captures the **global state** of a distributed system:
  - Local state of each process within the distributed system
  - Local state of each communication channel
- These local states are instantaneous
  - e.g messages in transit from one node to another only exist in transit for a few milliseconds or less



# Global Snapshots are Useful

- Checkpointing
  - Recover more quickly after failures
- Garbage Collection
  - Remove objects that are not referenced any more by other objects/processes at any other servers
- Deadlock Detection
  - Examine the global application state and identify any deadlocks, useful in transactional DB systems
- And many others ...

# System Model

- N processes in the system
  - Each process keeps track of some state
- There are two unidirectional communication channels between each pair of processes P and Q
  - FIFO-ordered (i.e first-in-first-out)
  - Message arrives intact and is unduplicated
  - Each channel also has some state
- No failures

# Messages and States

- What are the messages?
  - Application messages that differ across systems (e.g “sending \$10 from A to B”, “read value at memory address X and write back with a new value”)
  - Special messages (e.g marker message) that should not interfere with application messages
- What are the states?
  - Process state: application-defined state, or the classic notion of state which includes heap, registers, program counters and etc
  - Channel state: the set of messages inside
- Tips for Assignment 2
  - See `*.top`, `*.events`, `*.snap` files under `./test_data` to understand what states and messages mean in this assignment
  - Read `test_common.go` to understand the syntax of the above files, and their relationships with the simulator

# Distributed Snapshot

[“Distributed Snapshots: Determining Global States of Distributed Systems”](#) 1985,  
by K. Mani Chandy and Leslie Lamport

**Key Idea:** Servers send **marker messages** to each other

Marker messages

1. Mark the beginning of the snapshot process on the server
2. Act as a barrier (stopper) for recording messages

# Chandy-Lamport Algorithm

Any process can **initiate** the snapshot

- Record local state
- Create marker messages and send them to all outbound channels
- Start recording messages from all incoming channels

# Chandy-Lamport Algorithm Continued

## When **receiving** a marker message from channel C

If this is the **first marker message** that this process has even seen:

- Record the local state
- Record the state of C as “empty sequence”
- Send out the marker message on all outbound channels
- Start recording messages from all of its other incoming channels

If it has **already seen** a marker message (e.g. from some other channel)

- Record the state of C as the sequence of messages received since the process’s local state has been recorded
- Stop recording messages on C (i.e done with recording the channel’s state)

See **Section 3** of the [original paper](#) for more details

# Chandy-Lamport Algorithm Continued

## When is the algorithm **terminated**?

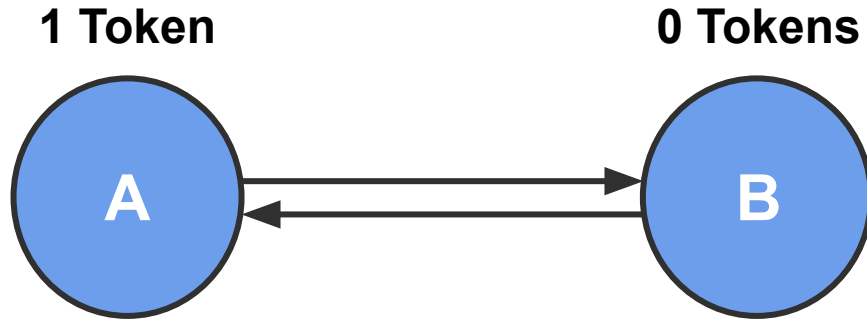
- All processes have received marker messages (i.e have recorded their local states)
- All processes have received marker messages from **all of their incoming channels** (i.e have recorded the local states of all channels)
- Both need to satisfy

## What **happens after** the termination?

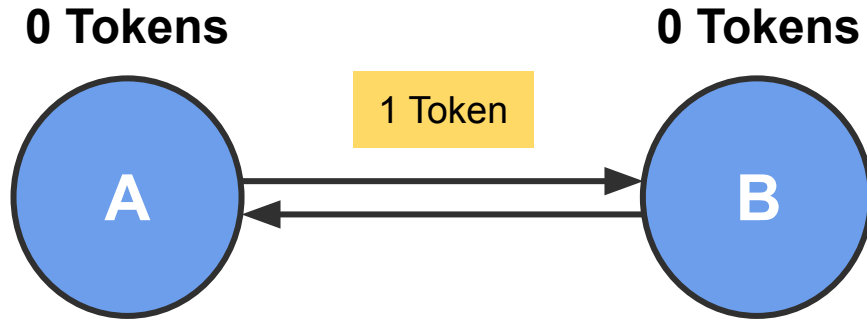
- Optional and out of the scope of Chandy-Lamport algorithm
- Usually, there will be a central server that collects local snapshots from all servers to build a global snapshot (e.g the `simulator` in Assignment 2) and maybe run some computations (e.g deadlock detection) on it



# Token passing example 1



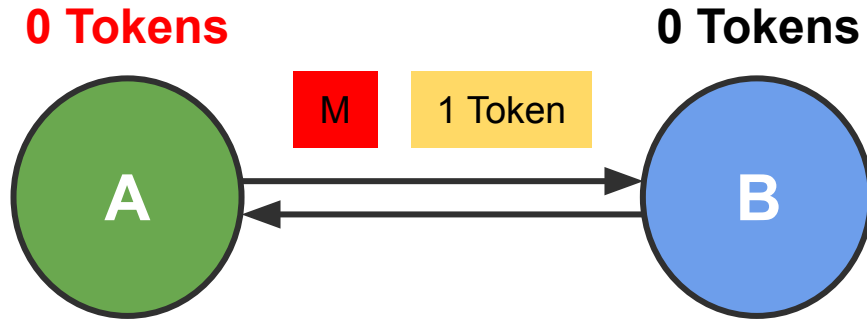
# Token passing example 1



*Event order:*

1. A sends 1 token

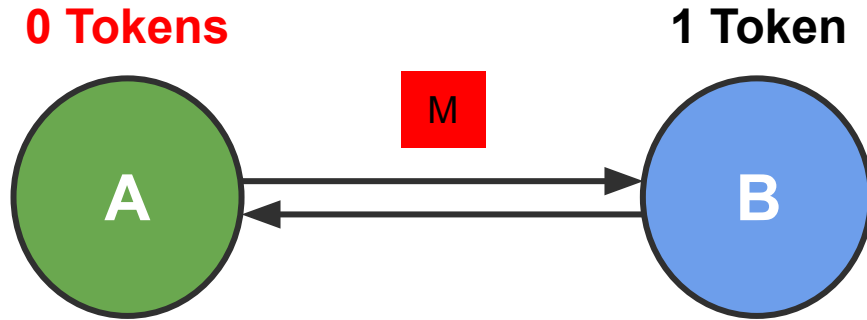
# Token passing example 1



*Event order:*

1. A sends 1 token
2. A starts snapshot, sends marker

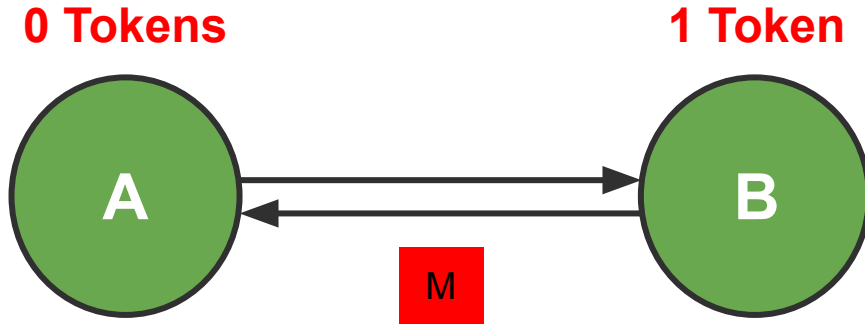
# Token passing example 1



*Event order:*

1. A sends 1 token
2. A starts snapshot, sends marker
3. B receives 1 token

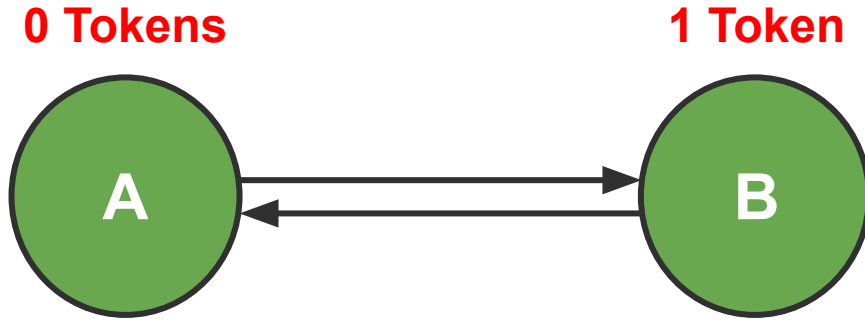
# Token passing example 1



*Event order:*

1. A sends 1 token
2. A starts snapshot, sends marker
3. B receives 1 token
4. B receives marker, starts snapshot

# Token passing example 1

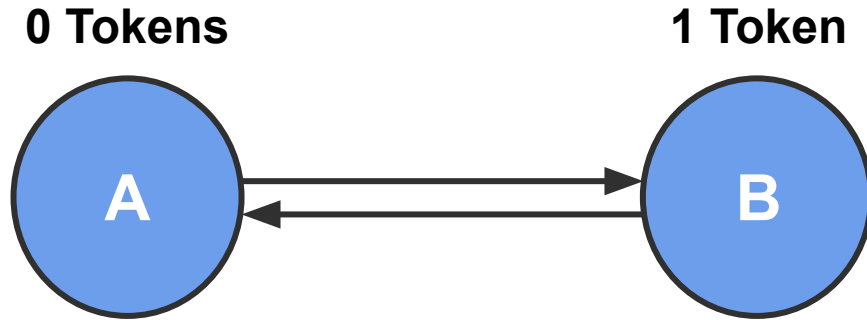


*We did not record the token message because B received it before B started the snapshot process*

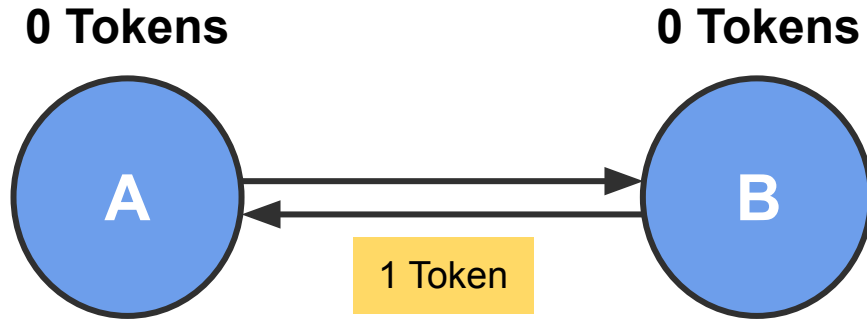
*Event order:*

1. A sends 1 token
2. A starts snapshot, sends marker
3. B receives 1 token
4. B receives marker, starts snapshot
5. A receives marker, ends snapshot

# Token passing example 2



# Token passing example 2

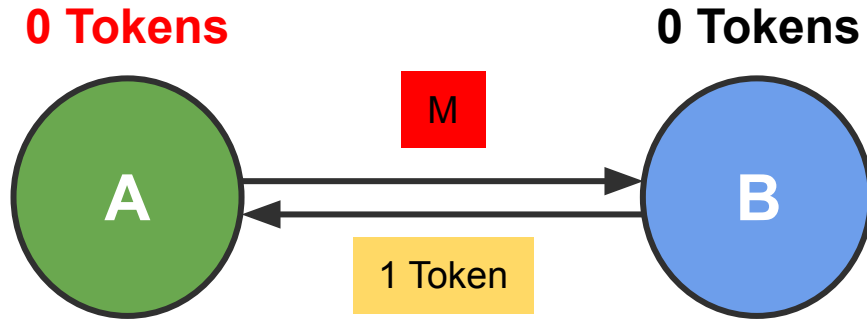


*Event order:*

1. B sends 1 token



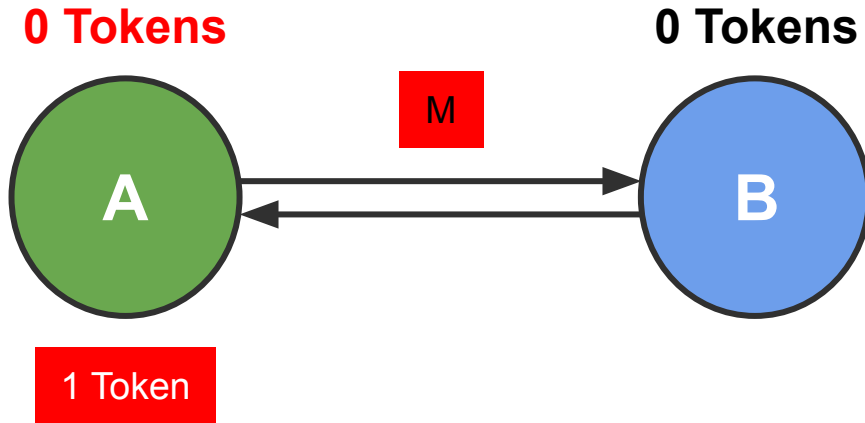
# Token passing example 2



*Event order:*

1. B sends 1 token
2. A starts snapshot, sends marker

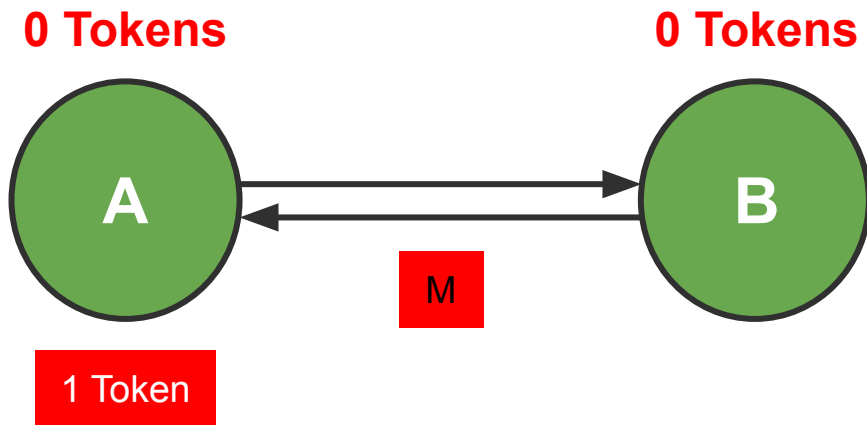
# Token passing example 2



*Event order:*

1. B sends 1 token
2. A starts snapshot, sends marker
3. A receives 1 token, records message

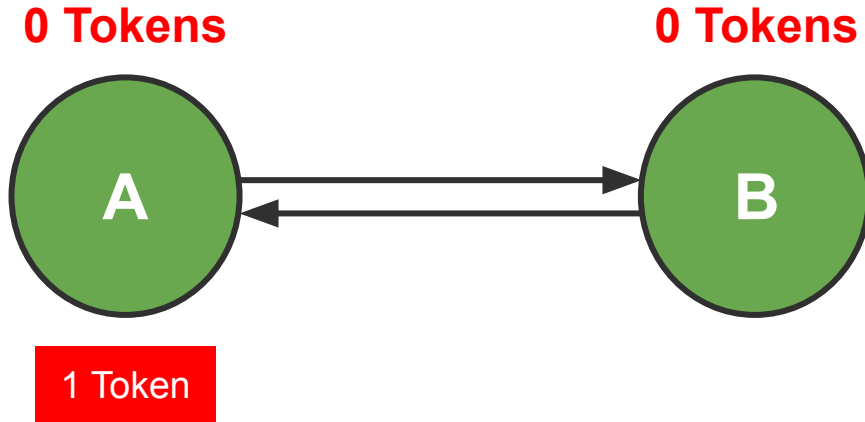
# Token passing example 2



*Event order:*

1. B sends 1 token
2. A starts snapshot, sends marker
3. A receives 1 token, records message
4. B receives marker, starts snapshot

# Token passing example 2

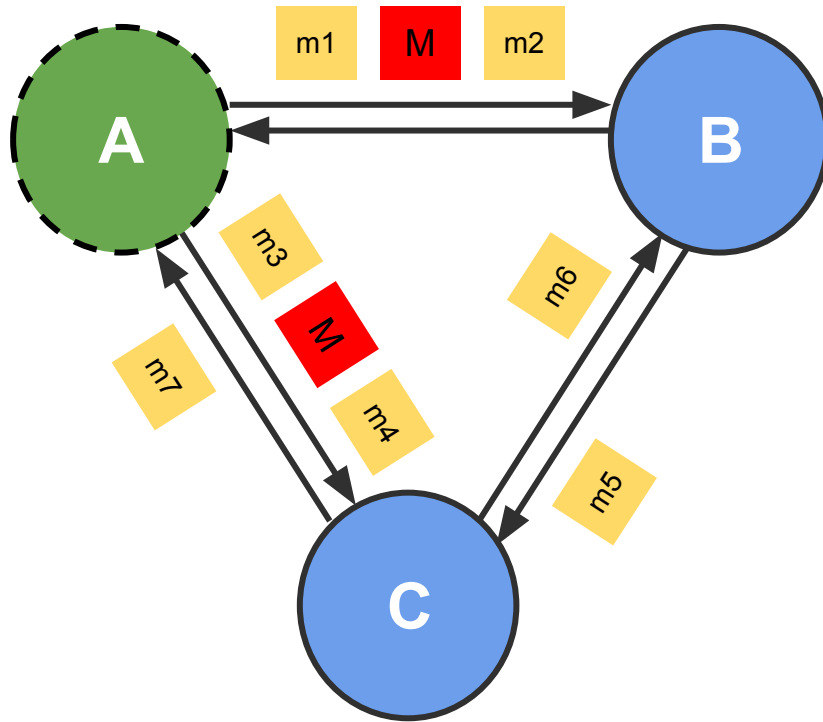


*We recorded the token message because A received it **after** it has already started the snapshot process*

*Event order:*

1. B sends 1 token
2. A starts snapshot, sends marker
3. A receives 1 token, records message
4. B receives marker, starts snapshot
5. A receives marker, ends snapshot

# Token passing example 3



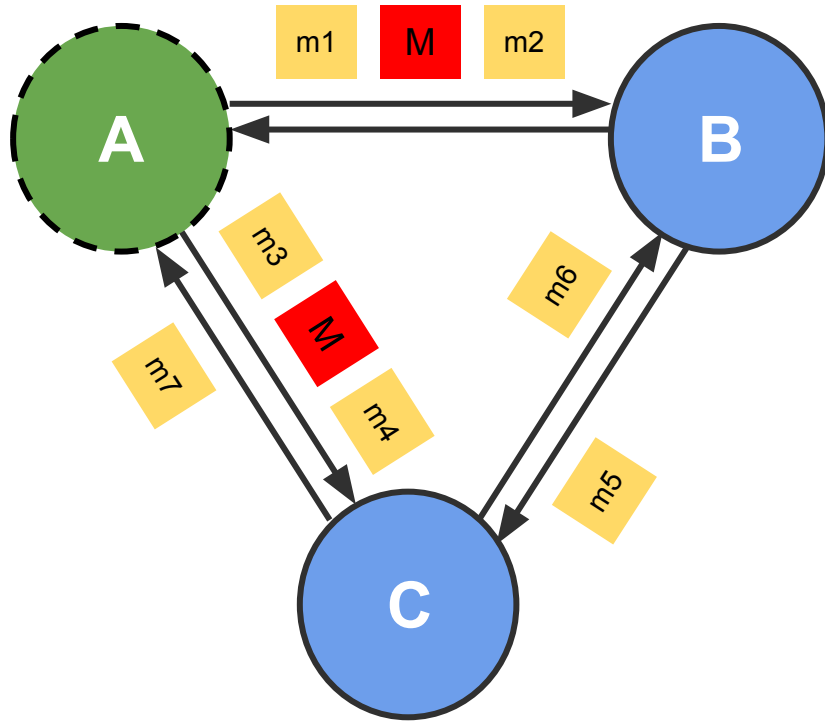
Which messages are definitely recorded\*?

Which messages are definitely *not* recorded?

Which messages *might* be recorded?

\* recorded as in-flight messages, i.e., as part of *channel state* rather than *process state*

# Token passing example 3



Which messages are definitely recorded\*?

**m7**

Which messages are definitely *not* recorded?

**m1, m3**

Which messages *might* be recorded?

**m2, m4, m5, m6**

\*recorded as in-flight messages

# Assignment 2 Overview

- You will implement the Chandy-Lamport snapshot algorithm
- Application is a token passing system
  - Number of tokens must be preserved in your snapshots
- Implementation uses *discrete time* simulator to order events
  - Simulator manages servers and injects events into the system
  - Server implements the snapshot algorithm
- Allow multiple active snapshot processes
  - E.g, The second snapshot can start before the first snapshot completes in the system

# Assignment 2 Interfaces

```
func (sim *Simulator) Tick()
```

```
func (sim *Simulator) StartSnapshot(serverId string)
```

```
func (sim *Simulator) NotifySnapshotComplete(serverId string, snapshotId int)
```

```
func (sim *Simulator) CollectSnapshot(snapshotId int) *SnapshotState
```

- What kind of state does the simulator need to keep track of?
  - Time
  - Topology
  - Channels to signal the completion of snapshots
  - ...



# Assignment 2 Interfaces

```
func (server *Server) SendToNeighbors(message interface{})
```

```
func (server *Server) SendTokens(numTokens int, dest string)
```

```
func (server *Server) HandlePacket(src string, message interface{})
```

```
func (server *Server) StartSnapshot(snapshotId int)
```

- What kind of state does the server need to keep track of?
  - Local state
  - Neighbors
  - Which channels received markers
  - Recorded messages
  - ...

# Assignment 2

Start Early 😊

Due Feb 28 (Friday) at 11:59pm!