# Maekawa on a Budget

## Making your threads work for YOU!

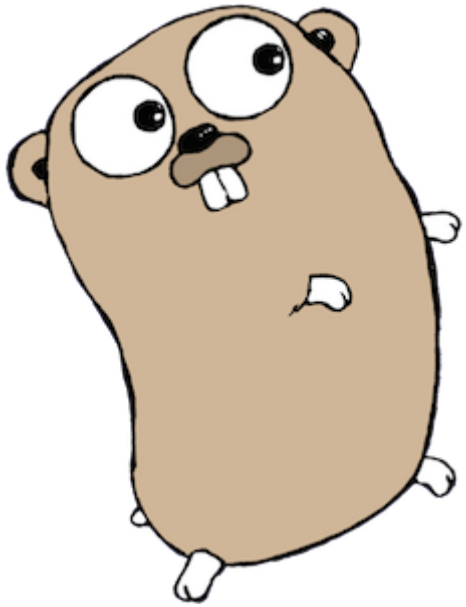
## 8 May 2018

Sampson Akwafuo, Jacob Hochstetler, Robert Podschwadt
CSCE6640

## Organization

1. Introduction
2. Message Model
3. Threading Model
4. Orchestration
5. Metrics Production

# Introduction



Yes, that's a dancing gopher. Available for download here. (https://golang.org/dl/)

# Message Model

```
type InterNodeMessage struct {
    From string
    MsgType
    Clock
}

// MsgTypes
msgRegister MsgType = iota
msgDeregister

msgMembershipUpdate

// Maekawa inter-node communication messages
msgREQUEST
msgGRANT
msgFAILED
msgINQUIRE
msgRELEASE
msgYIELD
```

# Threading Model

```
func (n *Node) StartNode(port int) {
    listener, err := net.Listen("tcp", n.Address)
    if err != nil {
        log.Fatalf("could not establish listener: %v", err)
    }
    log.Printf("[%v] Started listening\n", n.Address)
    defer listener.Close()

    n.Register()

    for {
        conn, err := listener.Accept()
        if err != nil {
            log.Fatalln(err)
            continue
        }
        go n.nodeHandler(conn)
    }
}
```

## Orchestration - Registering/Deregistering

Orchestrating node controls registration/deregistration of nodes entering the cluster.

Once a Maekawa quorum is found, each node's quorum is generated and sent to each node through a multi-part connection.
**Message 1**: *InterNodeMessage* => **Type**: msgMembershipUpdate
**Message 2**: *QuorumMessage* => **Members**: []string

All nodes are sent a new quorum on registration/deregistration.
Keeping track of Grants past a new quorum is left as an exercise for the reader.

# Orchestration - Health Checking

We keep track of *healthy* nodes through sophisticated, modern, almost blockchain based heuristics.

```
tick := time.Tick(every)
for {
    select {
    case <-tick:
        for _, node := range data.Nodes {
            go HealthCheck(node, every/3)
        }
    }
}

func HealthCheck(n *Node, timeout time.Duration) {
    conn, err := net.DialTimeout("tcp", n.Address, timeout) // Can we get a TCP handshake here?
    if err != nil { // nope, guess not
        n.Alive = false
        return
    }
    defer conn.Close()
    n.Alive = true // Handshake success!
}
```

## Metrics Production

Using the built-in *expvar* (expose variables)(https://golang.org/pkg/expvar/) package, we expose certains internal variables on a port +1 to the TCP listener.

**Node**:

- Logical (Lamport) clock
- Current quorum members

**Orchestrator**:

- Logical (Lamport) clock
- Healthy node count

# Metrics Production - Node Example


```
{
  "cmdline": [
      "C:\\Users\\jacob\\Dropbox\\School\\CSCE6640\\project\\project.exe",
      "node",
      "-server",
      "10.125.186.63:8000",
      "-port",
      "10000"
  ],
  "metrics": {
      "clock": 19,
      "quorum": [
          "10.125.186.63:11000"
      ]
  }
}
```

# Thank you

Sampson Akwafuo, Jacob Hochstetler, Robert Podschwadt
CSCE6640