Traffic Light Controller

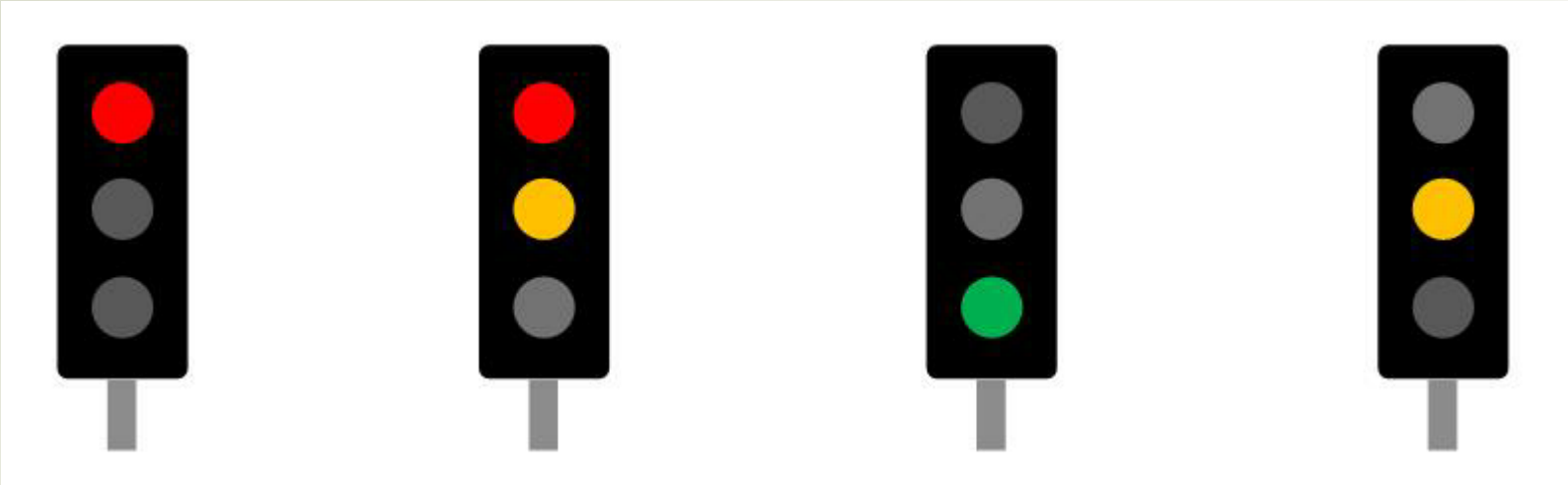
Basic sequence controller and FSM version

Contents

- UK traffic light sequence
- Building a controller with Node-Red
- Practical work
- Finite State Machines - FSM
- Building a controller with Node-Red (FSM)
- Conclusion

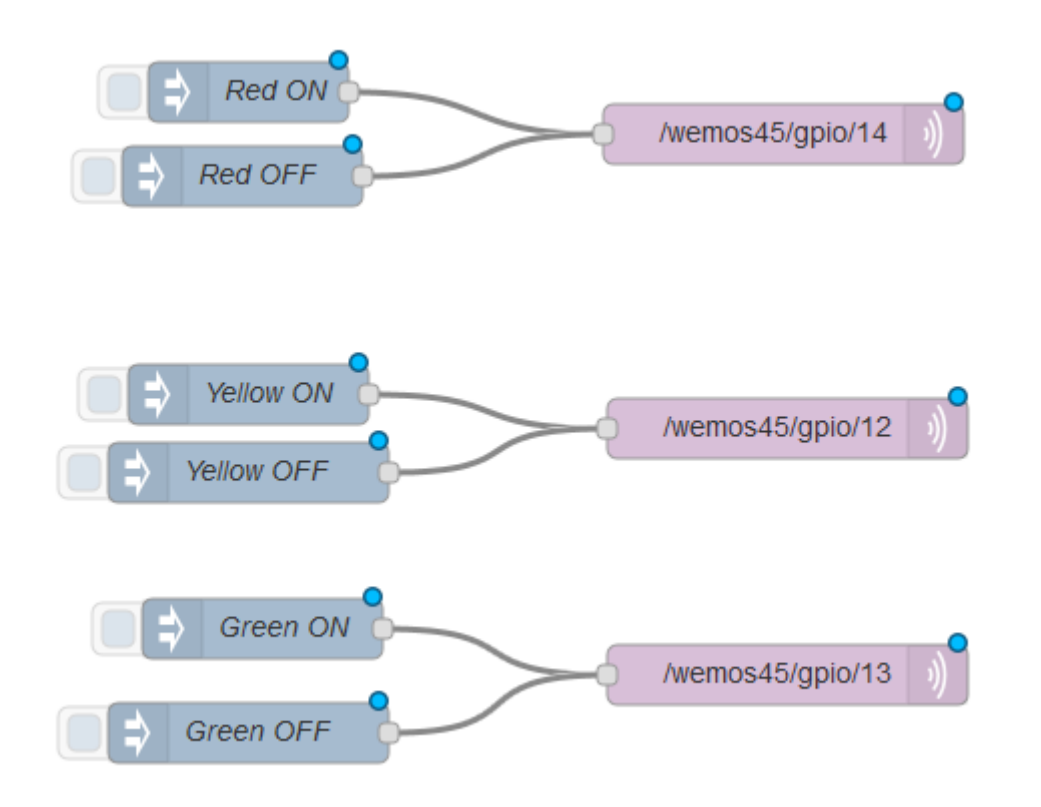
UK traffic light sequence

- RED means STOP
- RED and AMBER means get ready to GO

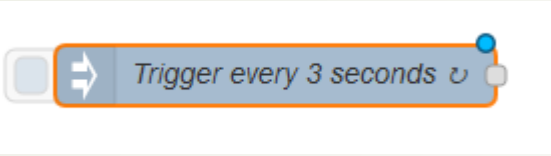


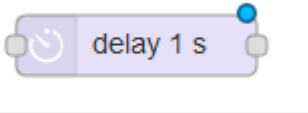
- GREEN means GO
- AMBER on its own means get ready to STOP

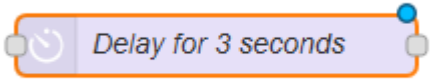
Build your controller using Node-Red



- Some extra parts you might need

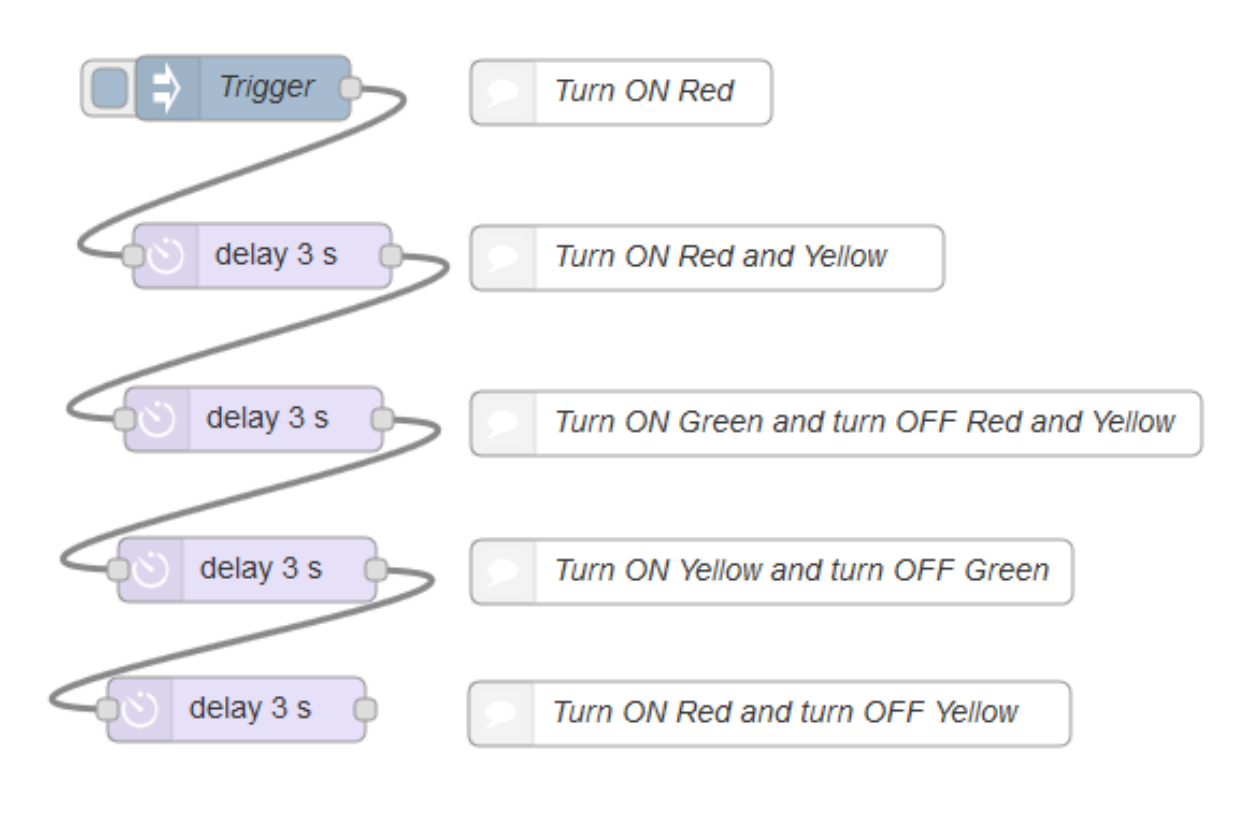






- ***And lots and lots of connecting WIRES***

One suggestion...



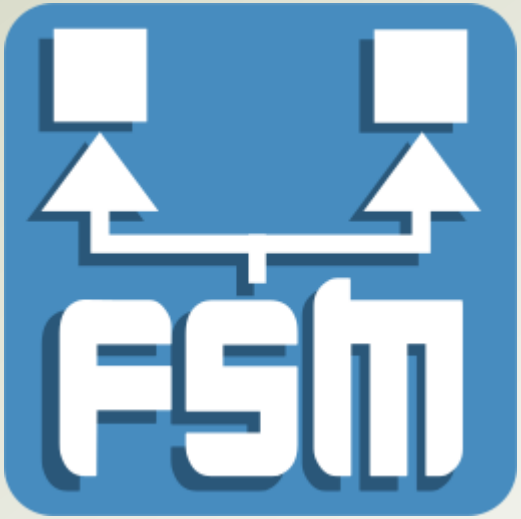
Over to you to have a go...

End of part one

- You have a few minutes to build your ‘flow’

- How did it go?
- Did you get it to work?
- Was the wiring easy to do?
- Does it look messy?

Finite State Machines - FSM



- A situation or product that has a number of unique states and paths between them
- This method can be used to '*program*' the product as it makes it easier (to understand)

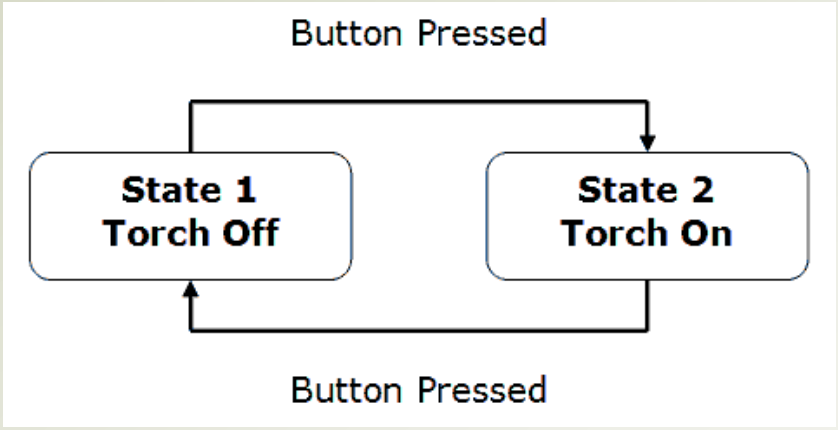
Examples of Finite State Machine

- A torch or light switch
- A washing machine
- A dishwasher
- A microwave oven
- A ‘hole in the wall’ cash machine
- And loads of other products

- And of course… A traffic light controller

A very simple two state FSM

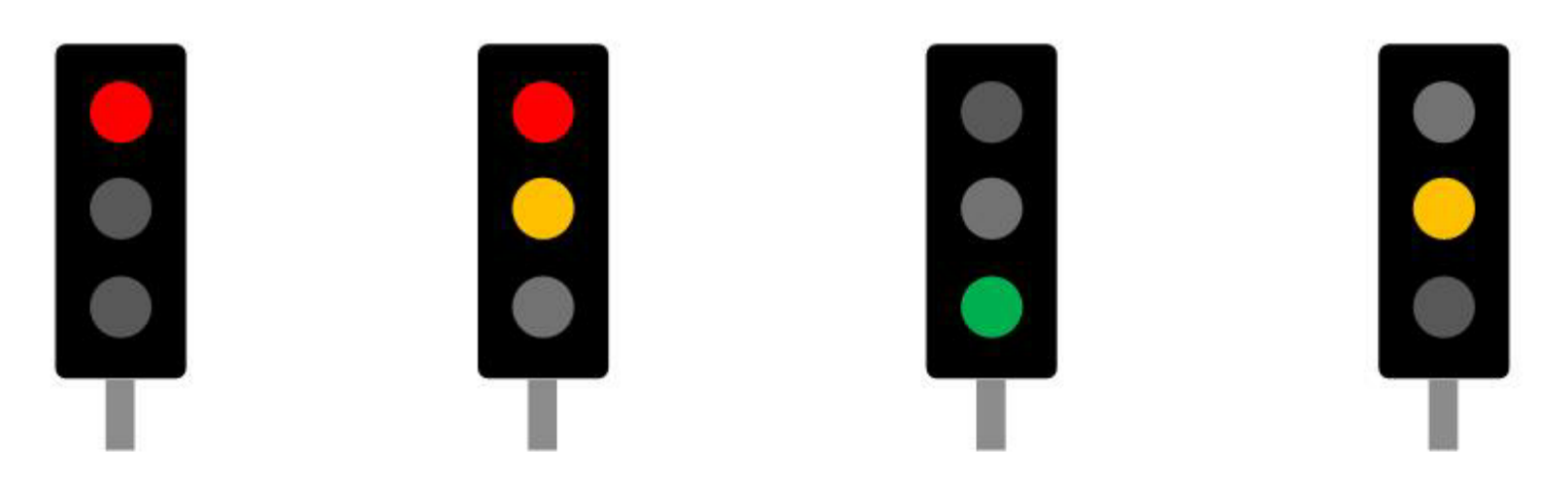
- A torch or light switch



- FSM has two states

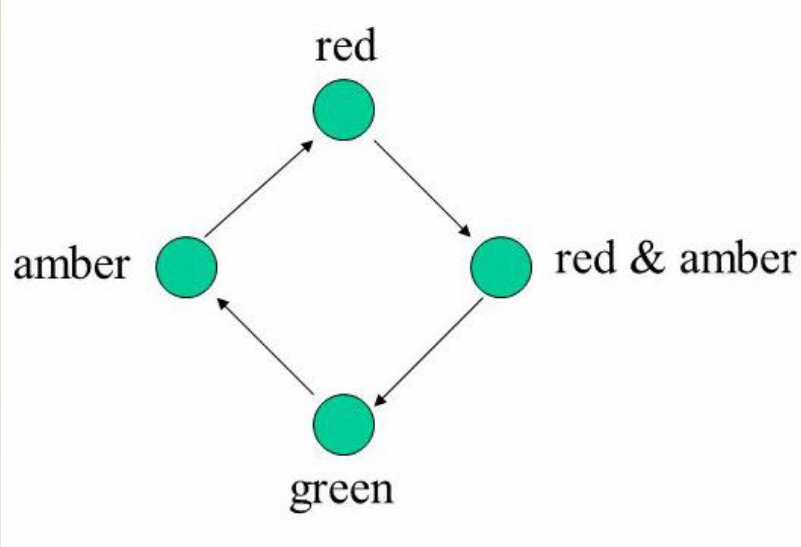
- FSM has two paths or transitions

UK traffic light sequence



- How many unique states does it have?
- How many transitions does it have?

A simple four state FSM

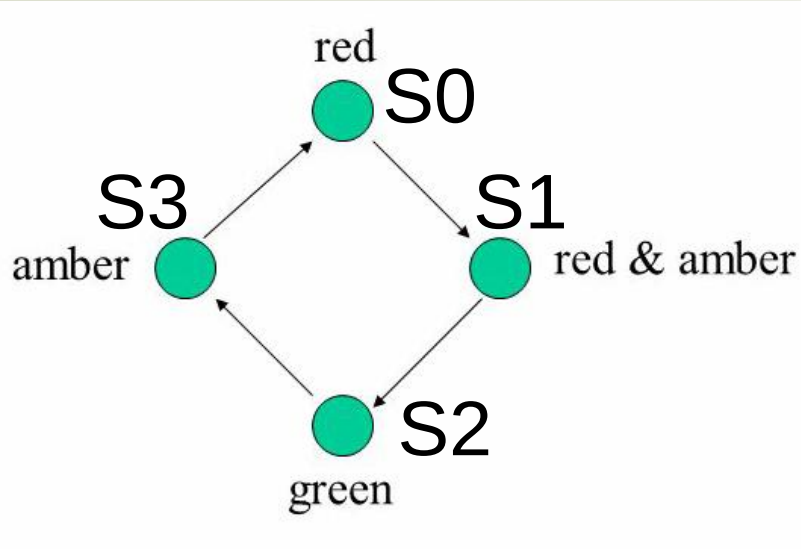


- One state is normally designated the ‘starting’ point
- Each state is numbered, starting from S0

- A state-counter is used to define which state the state machine is in

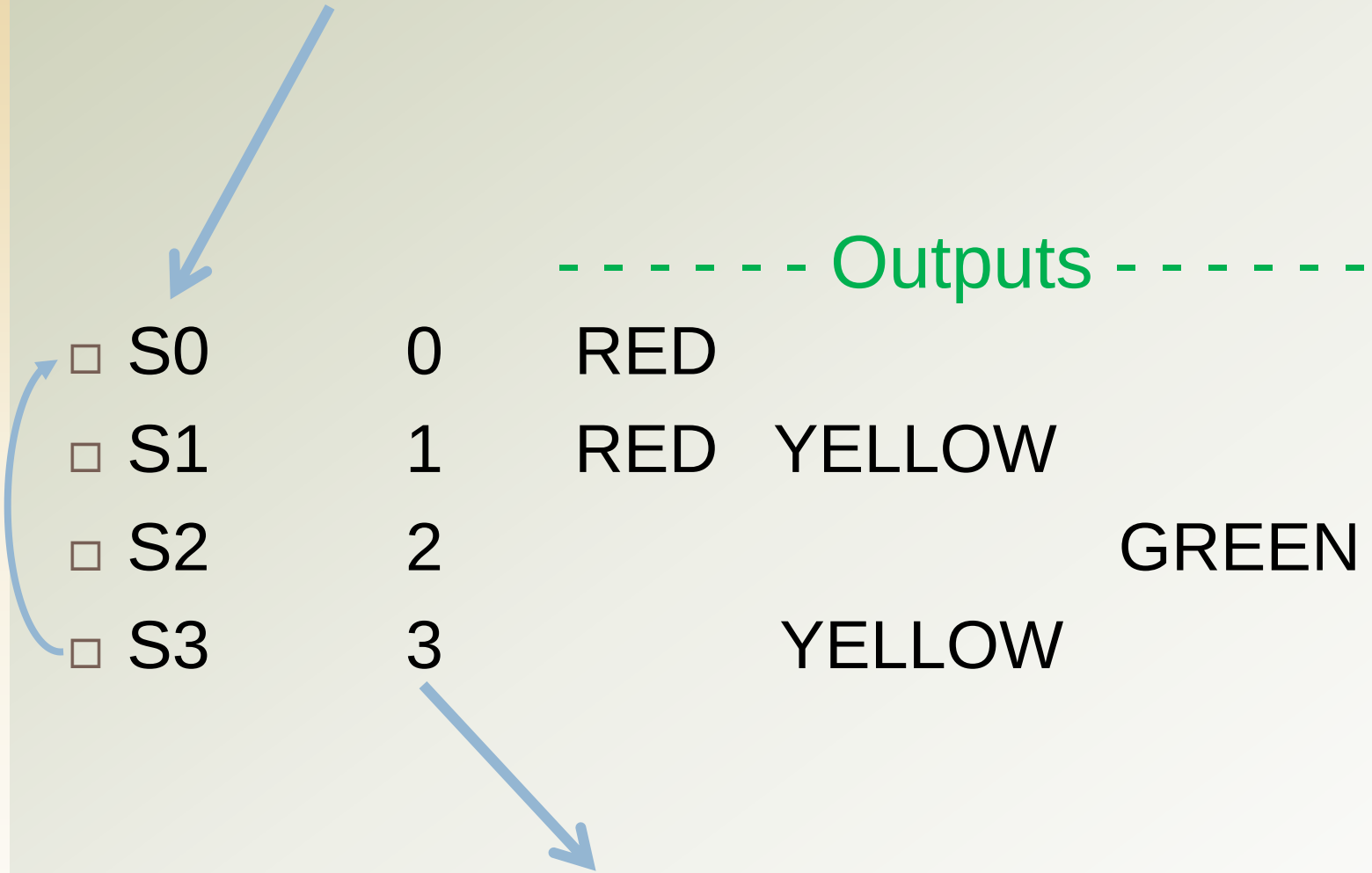
FSM for our traffic light controller

- S0 (this could also be the starting point)



  - It is assumed the transition from state to state happens every 3 seconds

State Counter for our FSM



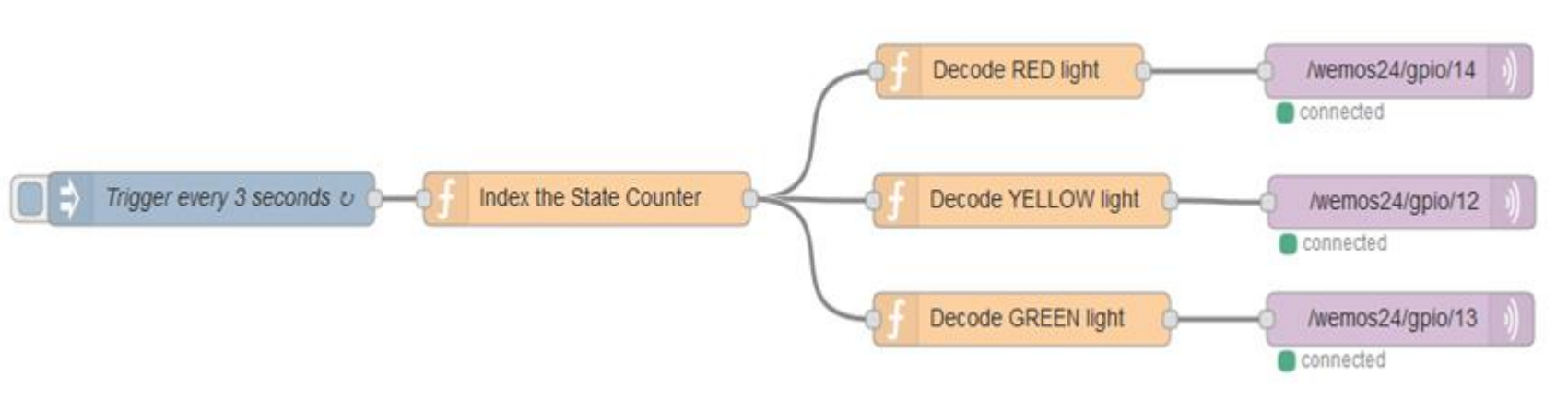
What the state machine has to do

- Increment or Reset the State Counter
  - Check if it has reached state S3
    - If it has, then reset it to state S0
      - **state_counter = 0**
    - Otherwise increment the state counter
      - **state_counter = state_counter + 1**

- Decode the states
  - Turn RED on if state is: S0 or S1
  - Turn Yellow on if state is: S1 or S3
  - Turn GREEN on if state is: S2

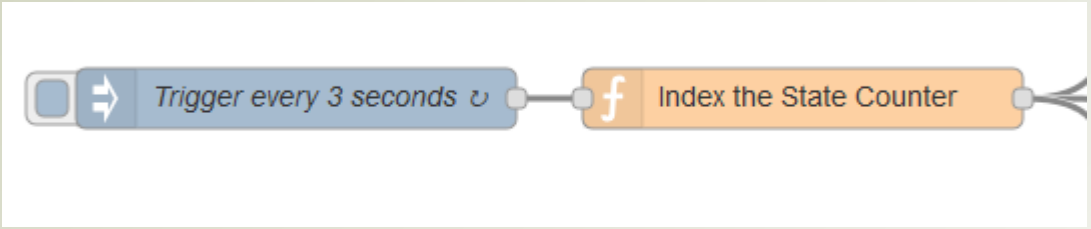
Overview of the Node-Red flow

- Trigger the flow every 3 seconds
- Index the *state_counter*



- Decode the *state_counter*
- Send command to node to drive LED

Increment the *state_counter*



Name: Index the State Counter

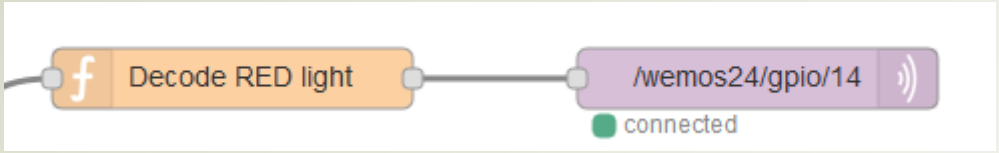
Function

```
1  var fsm_state = flow.get("state_counter") || 0;
2
3  if (fsm_state > 2)
4     {fsm_state = 0;}
5  else
6     {fsm_state = fsm_state + 1;}
7
8  flow.set("state_counter", fsm_state);
9  msg.payload = fsm_state;
10 node.status({text:"State counter = " + fsm_state});
11
12 return msg;
```

- Increment or reset the *state_counter*

Decode value of the *state_counter*

- This example is for the RED light



Name: Decode RED light

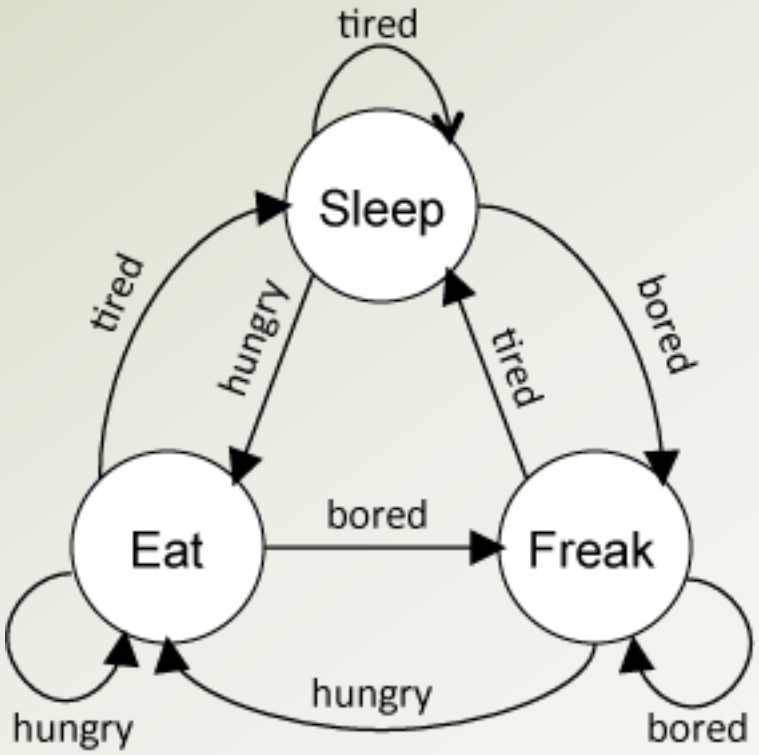
Function

```
1  var fsm_state = flow.get("state_counter");
2
3  if (fsm_state === 0 || fsm_state == 1)
4     {msg.payload = 1;
5     }
6
7  else
8     {msg.payload = 0;
9     }
10
11 return msg;
```

- You can work out the Yellow and Green lights

End of part two

- Your chance to create a simple FSM



- Over to you to have a go…

Conclusion

- FSM useful method to visualise a ‘machine’
- FSM can be ‘programmed’ in Node-Red
- You will meet many FSMs in your working life