# Unit 5

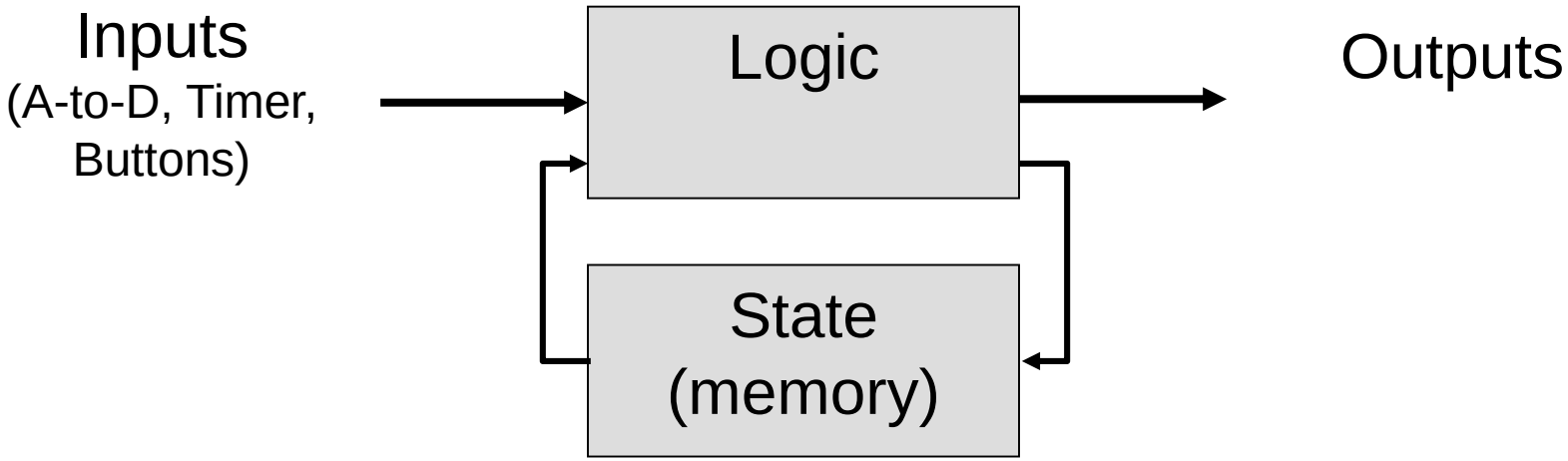
## State Machines

# What is state?

- You see a DPS officer approaching you. Are you happy?
  - It's late at night and your car broke down.
  - It's late at night and you've been partying a little too hard.
- Your interpretation is based on more than just what your senses are telling you RIGHT NOW, but by what has happened in the past
  - The sum of all your previous experiences is what is known as **state**
  - Your 'state' determines your interpretation of your senses and thoughts
- In a circuit, 'state' refers to all the bits being remembered (registers or memory)
- In software, 'state' refers to all the variable values that are being used

# State Machine Block Diagram

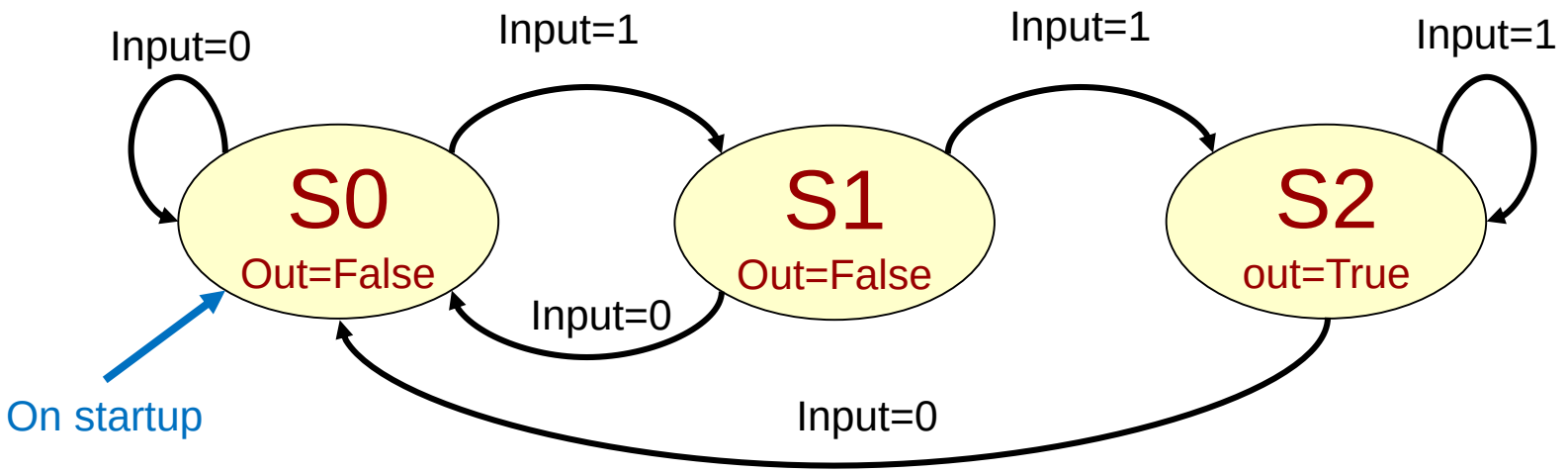
- A system that utilizes state is often referred to as a state machine
  - A.k.a. Finite State Machine [FSM]
- Most state machines can be embodied in the following form
  - Logic examines what's happening NOW (inputs) & in the PAST (state) to…
    - Produce outputs (actions you do now)
    - Update the state (which will be used in the future to change the decision)
- Inputs will go away or change, so state needs to summarize/capture anything that might need to be remembered and used in the future



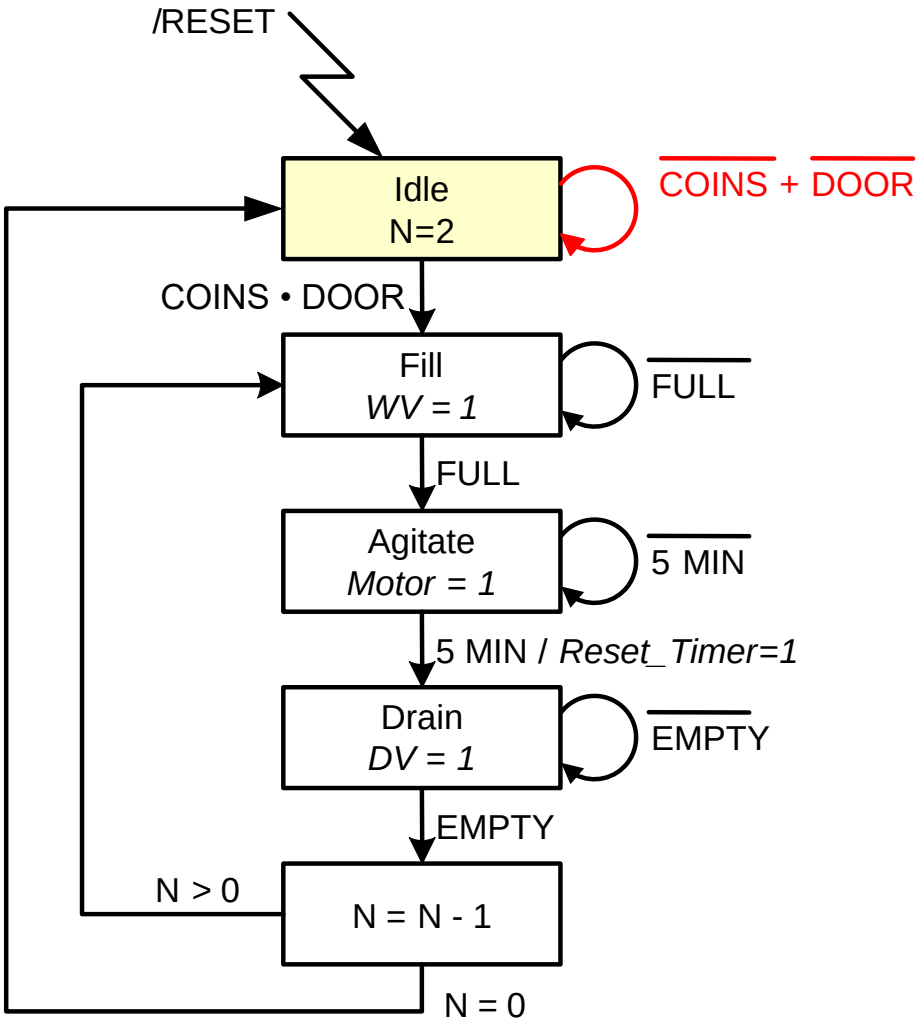
# State Diagrams

- Abstractly a state machine can be visualized and represented as a flow chart (or state diagram)
  - Circles or boxes represent state
  - Arrows show what input causes a transition
  - Outputs can be generated whenever you reach a particular state or based on the combination of state + input

**State Machine to check for two consecutive 1's on a digital input**



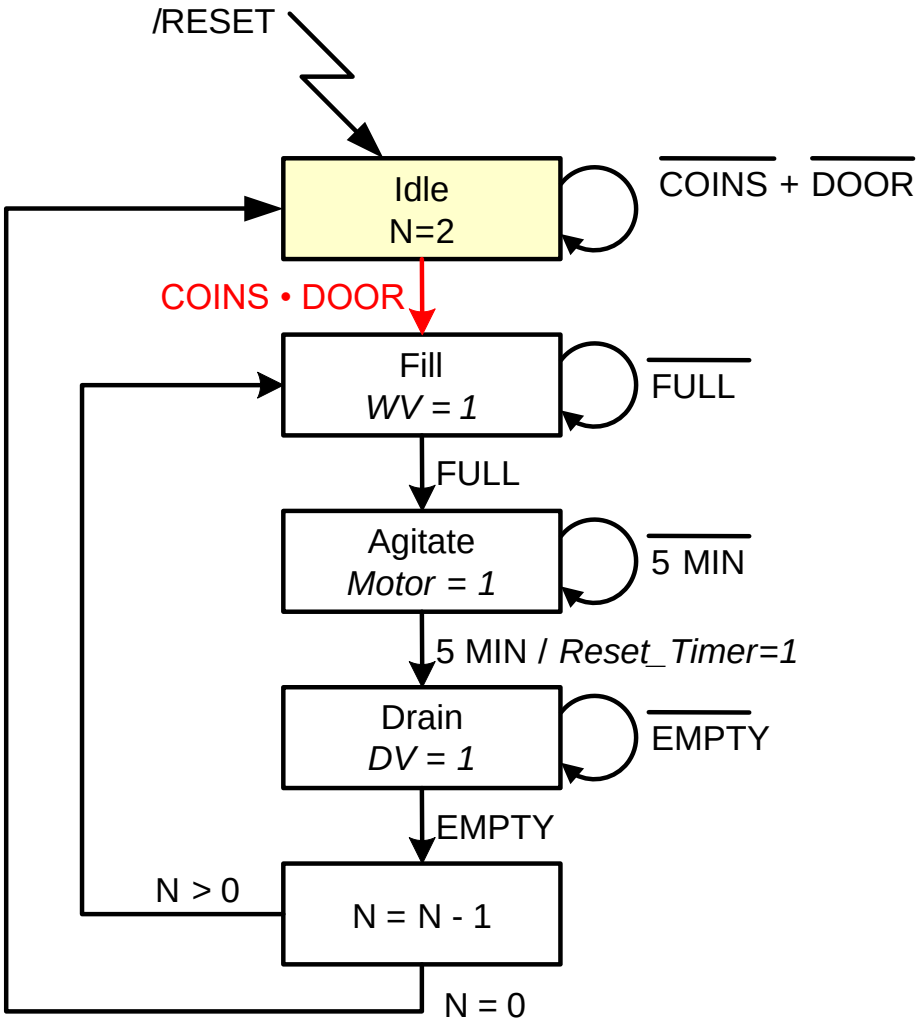
# Washing Machine State Diagram




**Stay in the initial state until there is enough money (coins) and the door is closed**

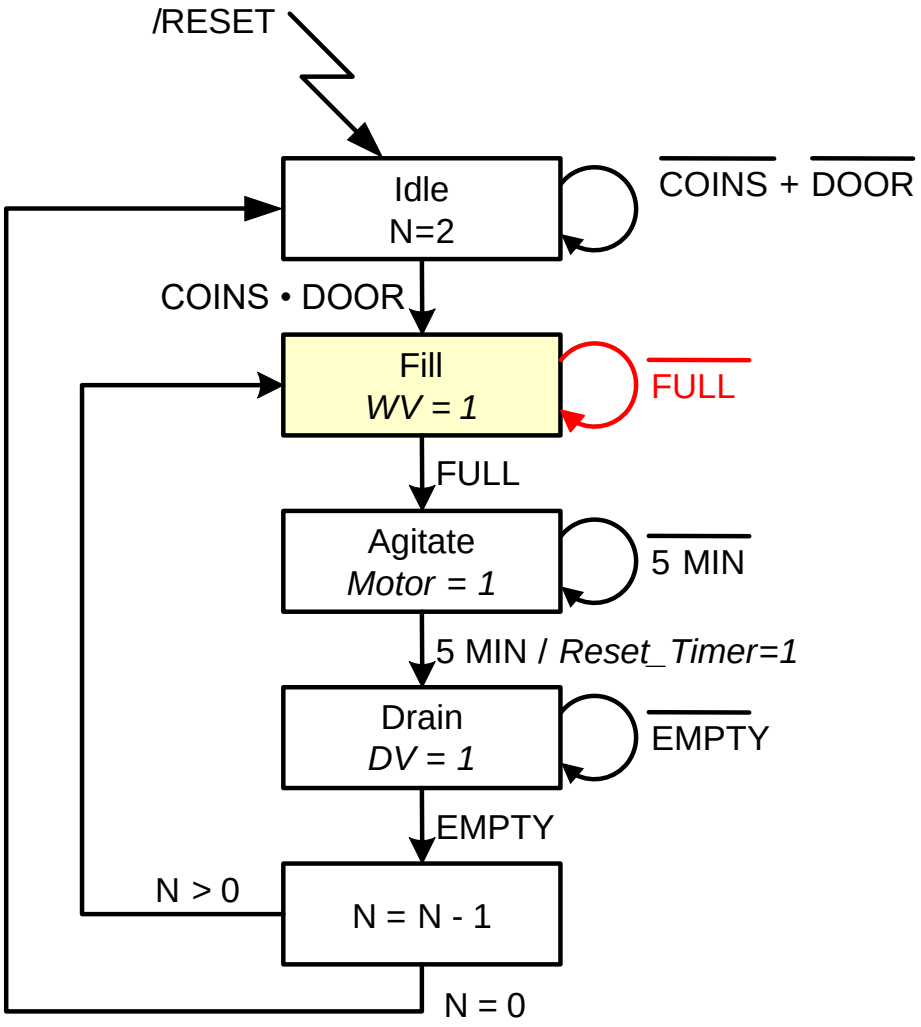
**We move through the states based on the conditions. Outputs get asserted when the machine is in that state and the transition is true.**

# Washing Machine State Diagram



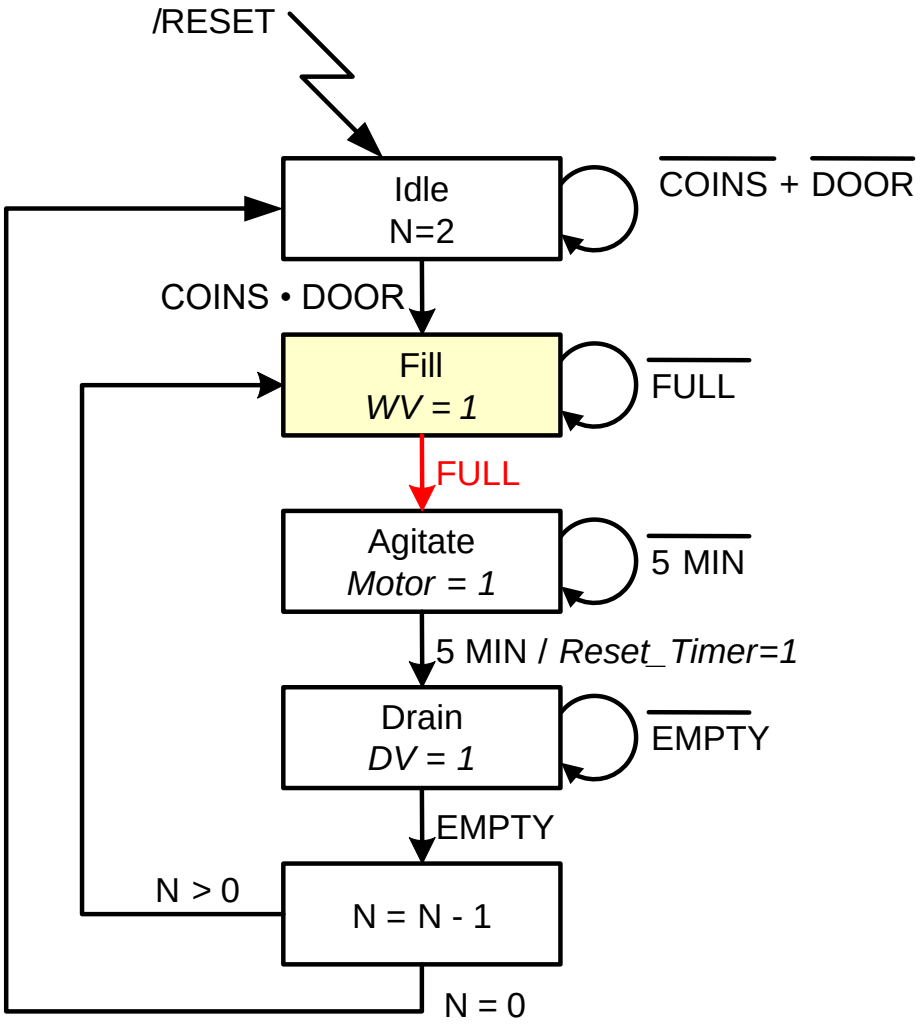

**Move to the Fill state when there is enough money (coins) and the door is closed**

# Washing Machine State Diagram




**Stay in the Fill state until it is full…also set the Water Valve Open output to be true**

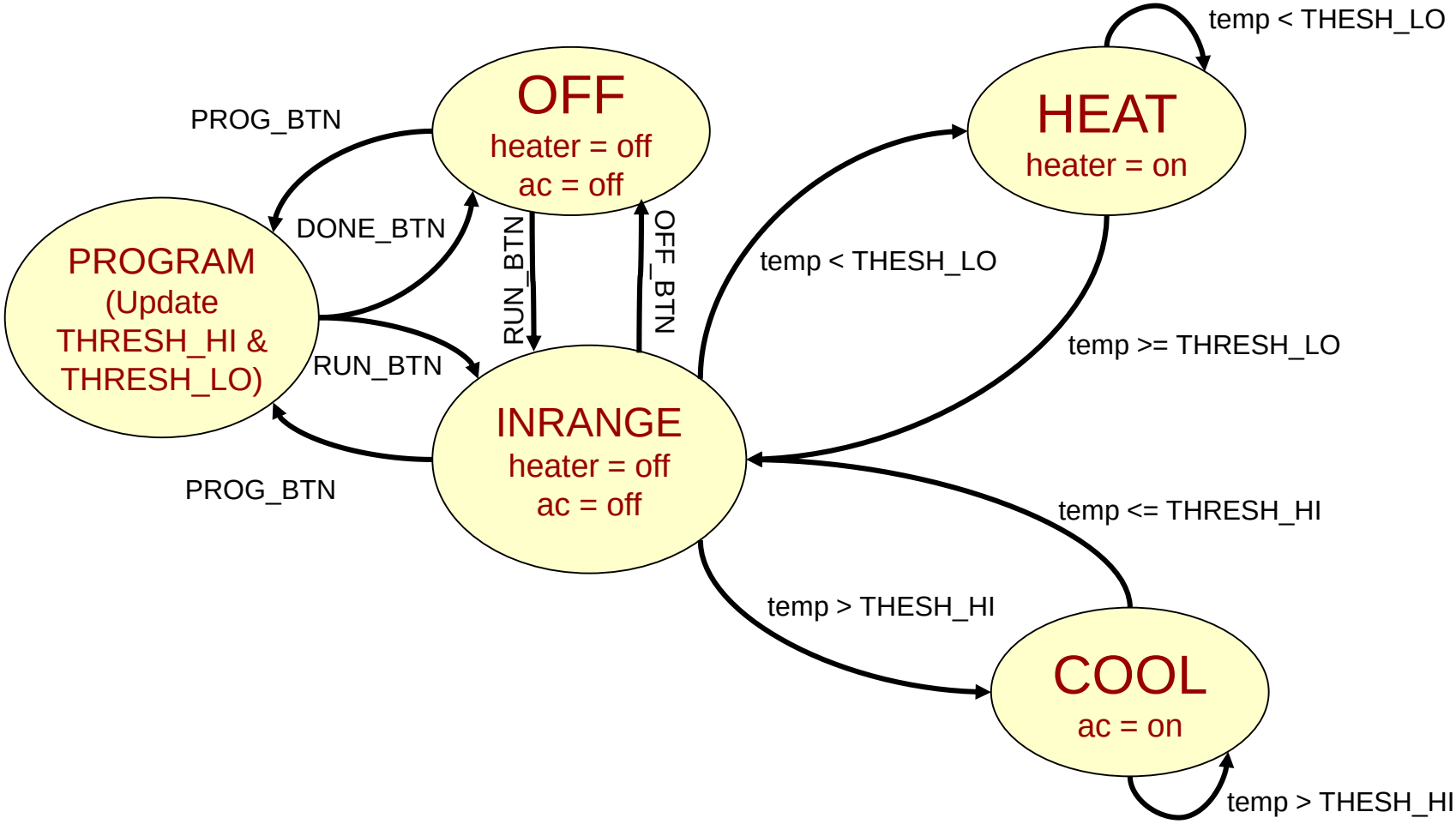
# Washing Machine State Diagram




**Move to the Agitate state after it is full**

# Thermostat

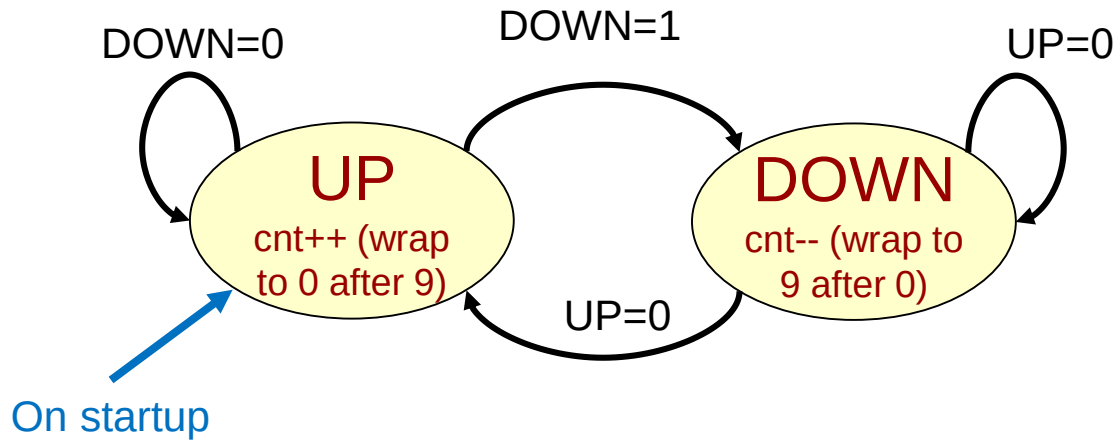
- Sample state machine to control a thermostat

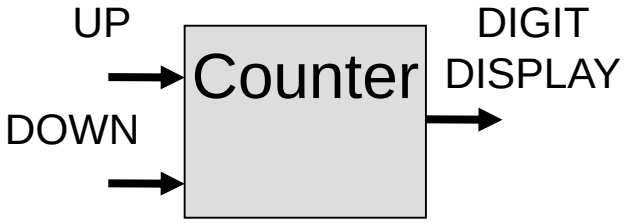


# Counter Example

- Consider a system that has two button inputs: UP and DOWN and a 1-decimal digit display. It should count up or down at a rate of 500 milliseconds and change directions only when the appropriate direction button is pressed
- Every time interval we need to poll the inputs to check for a direction change, update the state and then based on the current state, increment or decrement the count

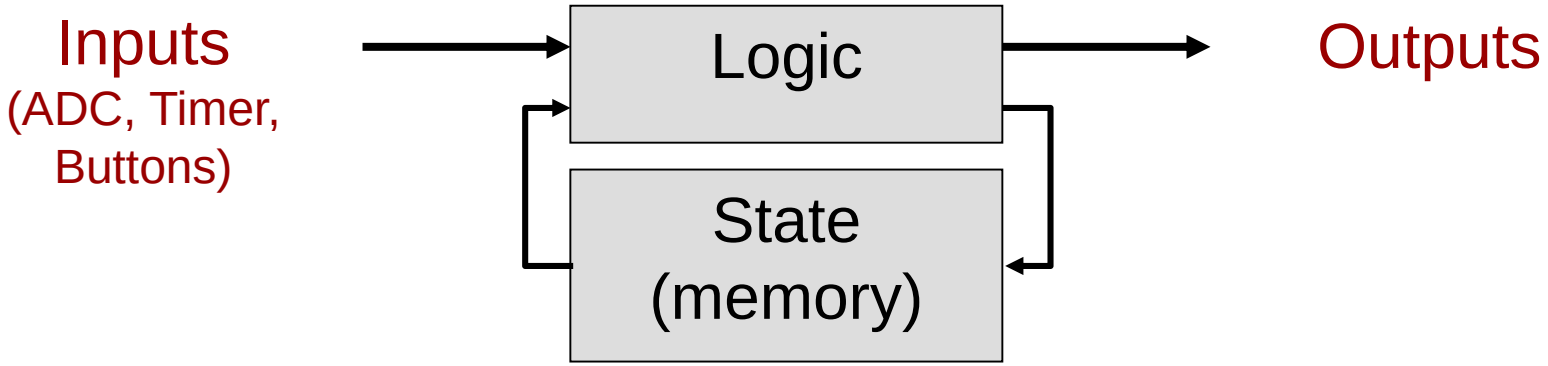
**State Machine to count up or down (and continue counting) based on 2 pushbutton inputs: UP and DOWN**



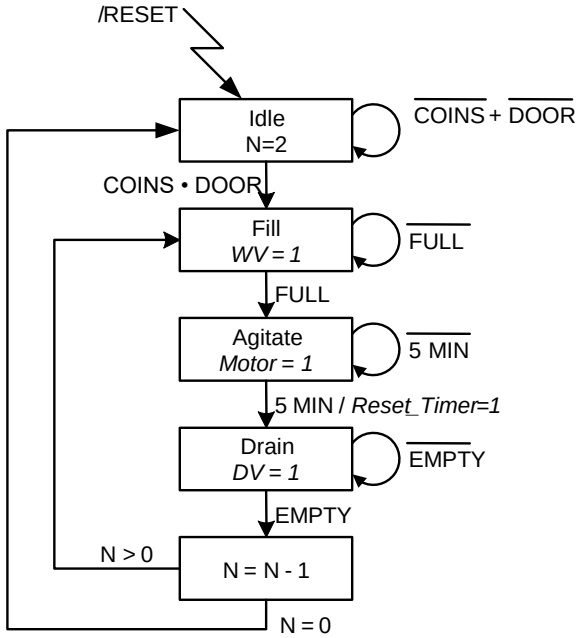
# Software vs. Hardware

- Software
  - State = just a variable(s)
  - Logic = if statements to update the next state
    - if(state == 'A' && input == 1) { state = 'B'; }
  - Transitions triggered by input or timers
  - **We'll start by implementing state machines in SW**
- Hardware
  - State = Register (D-Flip-Flops)
  - Logic = AND/OR Gates to produce the next state & outputs
  - Transitions triggered by clock signal
  - **More on this later in the semester**



# Software Implementation

- Store 'state' in some variable and assign numbers to represent state (0=Idle, 1=Fill, etc.)
- Use a timer or just poll certain inputs and then make appropriate transitions



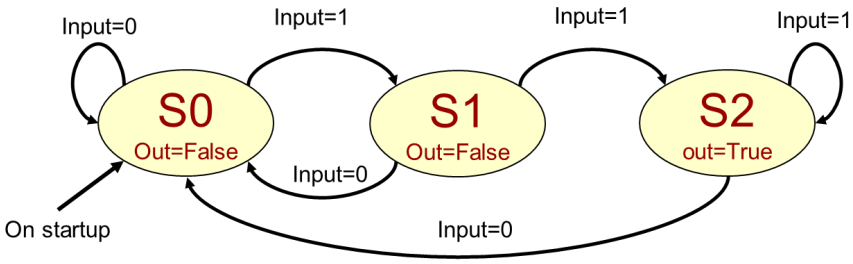

**State Diagram for a Washing Machine**

```
int main()
{
  bool coins, door;
  int currst = 0, nextst = 0, n = 2;
  while(1)
  {
    _delay_ms(10);
    coins = PIND & (1 << PD0);
    door = PIND & (1 << PD1);
    if(currst == 0){
      if( coins && door ){
        nextst = 1;
      }
    }
    else if(currst == 1){
      ...
    }
    ...
    currst = nextst; // update state
  }
  return 0;
}
```

Use nested 'if' statements: outer 'if' selects state then inner 'if' statements examine inputs

# More Implementation Tips

- Continuously loop
- Each iteration:
  - Poll inputs
  - Use if statement to decide current state
  - In each state, update state appropriately based on desired transitions from that state
  - Produce appropriate output from that state




```
// input = PD0,  output = PD7
int main()
{ // be sure to init. state
  unsigned char state=0, nstate=0;
  unsigned char input, output;
  while(1)
  {
    _delay_ms(10);
    input = PIND & (1 << PD0);
    if(state == 0){
      PORTD &= ~(1 << PD7);
      if( input ){ nstate = 1; }
    }
    else if(state == 1){
      PORTD &= ~(1 << PD7);
      if( input ){ nstate = 2; }
      else { state = 0; }
    }
    else { // state == 2
      PORTD |= (1 << PD7);
      if( !input ) { nstate = 0; }
    }
    state = nstate; // update state
  }
  return 0;
}
```

# State Machine Implementation Template

Select current state

Select input val.

Select input val.

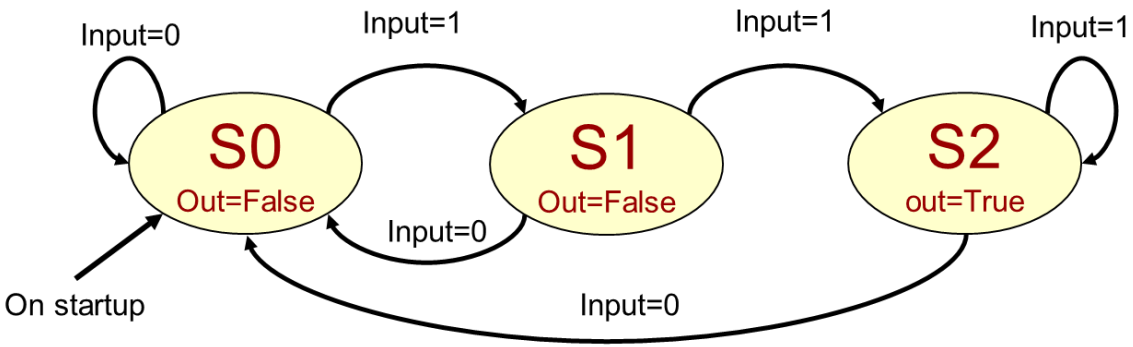
```
int main()
{ ...
  unsigned char cstate=0, nstate=0; // be sure to init. state
  unsigned char input, output;
  while(1)
  {
    _delay_ms(10); // choose appropriate delay

    // Capture inputs
    input = PIND & (1 << PD0);

    // Use if..else if statement to select current state
    // Don't use if..if..if since many can trigger
    if(cstate == 0){
      PORTD &= ~(1 << PD7); // Perform outputs based on state
      // In each state use if..else if statement
      // to determine input and go to next state
      if( input ){
         nstate = 1; /* transition */
         // Perform outputs based on state + inputs
      }
      else {
         nstate = 2; /* transition */
         // Perform outputs based on state + inputs
      }
    }
    else if(cstate == 1){
      if( input ){ nstate = 2; }
      else { nstate = 0; }
    }
    else if(cstate == 2) {
      if( !input ) { nstate = 0; }
    }
  }
  return 0; }
```

# State Machines as a Problem Solving Technique

- Modeling a problem as a state machine is a powerful problem-solving tool
- When you need to write a program, design HW, or solve a more abstract problem at least consider if it can be modeled with a state machine
  - Ask questions like:
    - What do I need to remember to interpret my inputs or produce my outputs? [e.g. Checking for two consecutive 1's]
    - Is there a distinct sequence of "steps" or "modes" that are used (each step/mode is a state) [e.g. Thermostat, washing machine, etc.]

# Formal Definition

- Mathematically, a state machine is defined by a 6-tuple (a tuple is just several pieces of information that go together):
  - A set of possible input values
  - A set of possible states
  - A set of possible output values
  - An initial state
  - A transition function: {States x Inputs} -> the Next state
  - An output function: {States x Inputs} -> Output value(s)



# Formal Definition

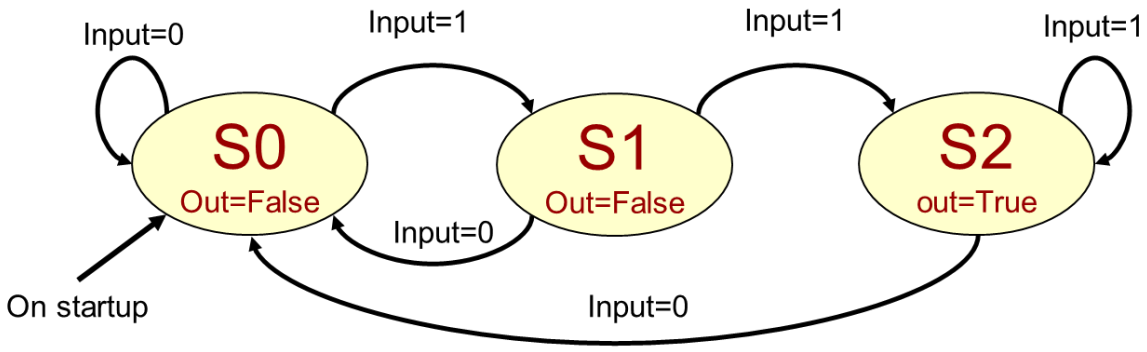
- Mathematically, a state machine consists of:
  - A set of possible input values: {0, 1}
  - A set of possible states: {S0, S1, S2}
  - A set of possible outputs: {False, True}
  - An initial state = S0
  - A transition function:
    - {States x Inputs} -> the Next state
  - An output function:
    - {States x Inputs} -> Output value(s)

| | Inputs | |
|---|---|---|
| **State** | **0** | **1** |
| **S0** | S0 | S1 |
| **S1** | S0 | S2 |
| **S2** | S0 | S2 |

**State Transition Function**

| State | Outputs |
|---|---|
| **S0** | False |
| **S1** | False |
| **S2** | True |

**Output Function**

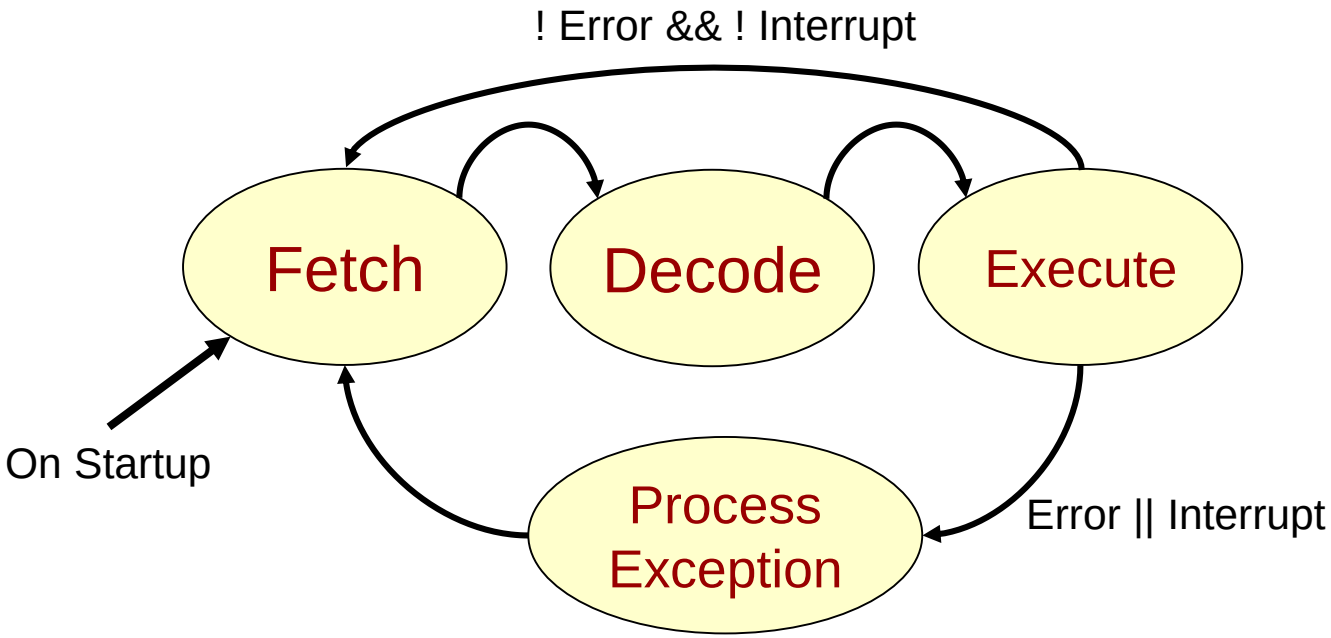


HW (Instruction Cycle) & Software (String Matching)

# MORE EXAMPLES IF TIME

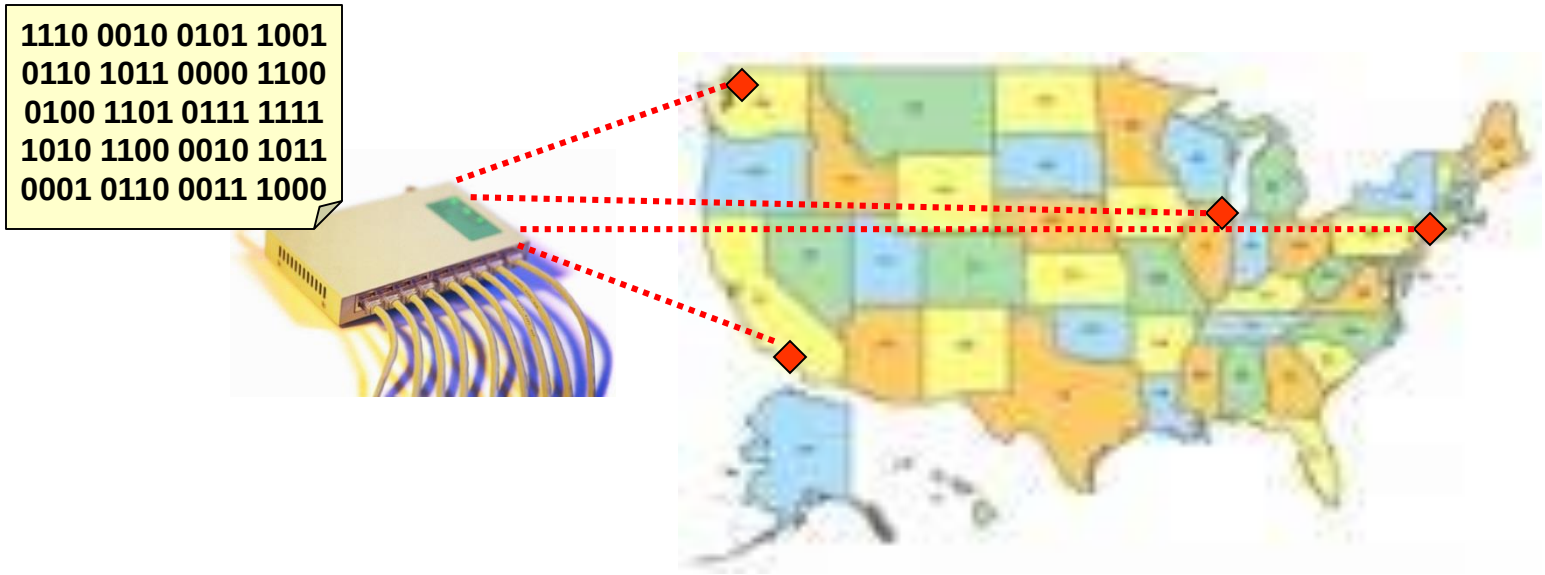
# More State Machines

- State machines are all over the place in digital systems
- Instruction Cycle of a computer processor



# Another Example

- On the Internet, packets of data are transferred between "router" devices
- Each router receives thousands of packet per second each of 100's-1000's of bytes of data
- These packets may contain viruses, spam, etc.
- Given patterns (common spam words or virus definitions), can we find these in the data and filter them out?



# Looking for Signatures

- Look for specific patterns (i.e. signatures) such as data that would indicate a specific virus, words that are typically spam, etc.
- Databases of these signatures are available
- We take a packet and search for the presence of any of these signatures in our database
- If we find a signature we can drop the packet and not deliver it

# String/Pattern Matching

- Given a large array of data (let's say text characters) how can we efficiently find the occurrence of specific strings (patterns)?

```
win
offer
cash
free
deceased
inherit
...
```

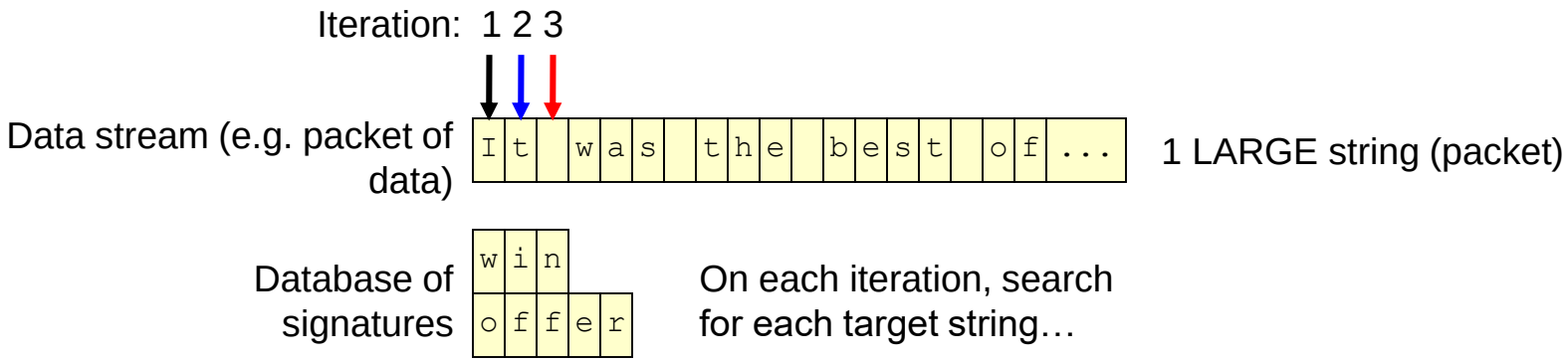
Database of signatures

```
Hello,

I am Barr. Phillip Butulezi, an attorney of law to a deceased Immigrant property Magnate, who was based in the U.K, also referred to as my client.

On the 25th of July 2000, my client, his wife, and their two Children died in the Air France concord plane crash bound for New York. They were on their way to a world cruise.
```

Data stream (e.g. packet of data)

# Brute Force

- Take each character in the data stream
  - Compare each string in the database to the string starting at the character in the data stream
  - Use strncmp()



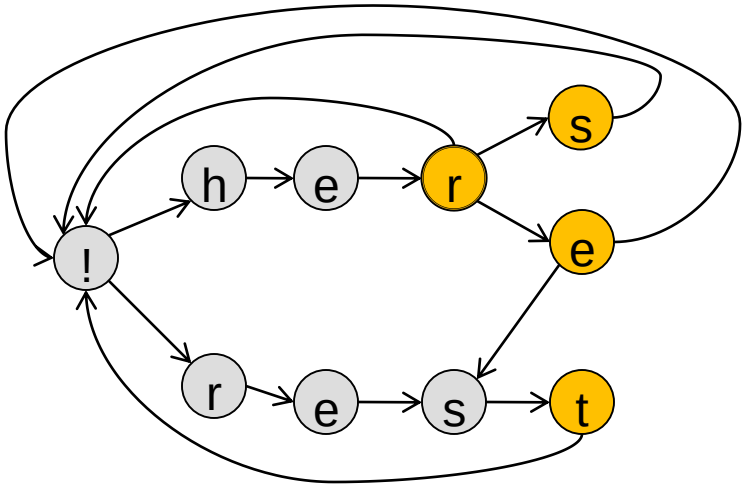

Data Stream = N chars with T Targets => Run Time proportional to N*T

# A Better way

- Can we avoid checking each of the T target strings for each character in the data stream
- Can we take a letter from the data stream and simultaneously track possible (partial) target string matches
  - Example strings: her, hers, here, rest
  - Data Stream: heresthers
    - Don't check all 4 target strings, just grab 'h' and see what options are possible and which are ruled out… (i.e. keep track of all options simultaneously)
    - h [could be her or hers or here]
    - e [could still be her or hers or here]
    - r [found her! But could also be hers or here or start of rest]
    - e [found here! Could be start of rest]
    - s [Could be rest ]
    - t [Found rest ]
    - h [Could be start of her or hers or here]

# Use a state machine

- '!' represents 'null' state
  - No part of a definition found
- Slightly different notation used
  - State label indicates the input character that would put you into that state
- What state you're in "tracks" what you've seen thus far AND what target strings you might be about to find…

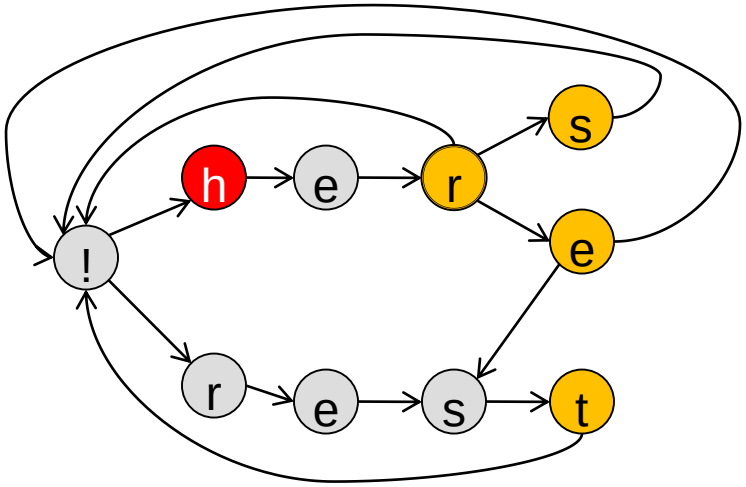



Internal state

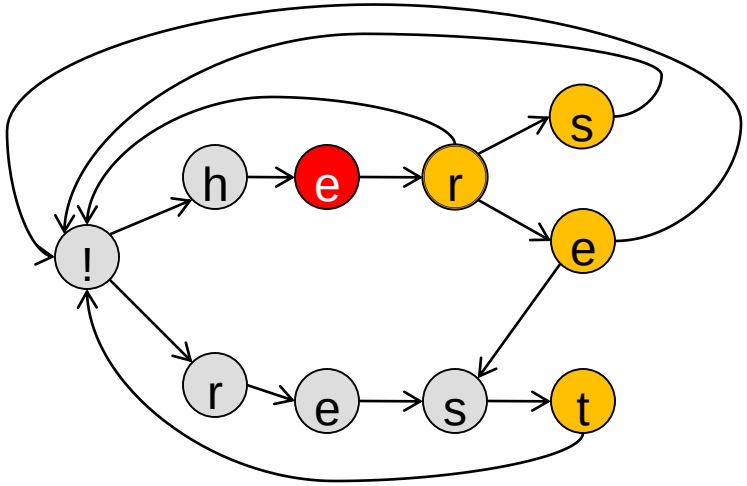
Terminal / Pattern Found State (i.e. output should be True)

# Finite State Automaton

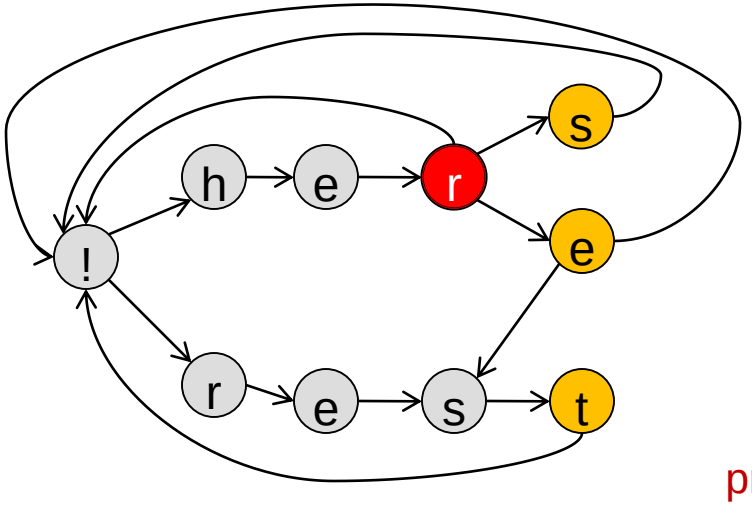
- Data Stream: heresthers



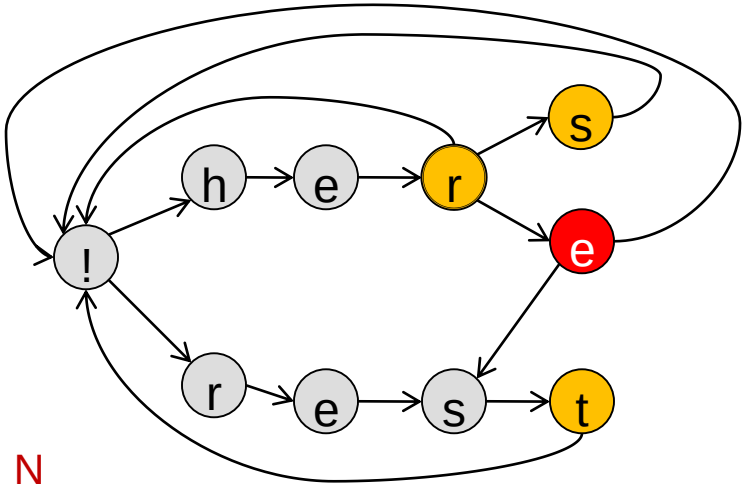

1




2




3




4

Run-Time proportional to N