Unit 5

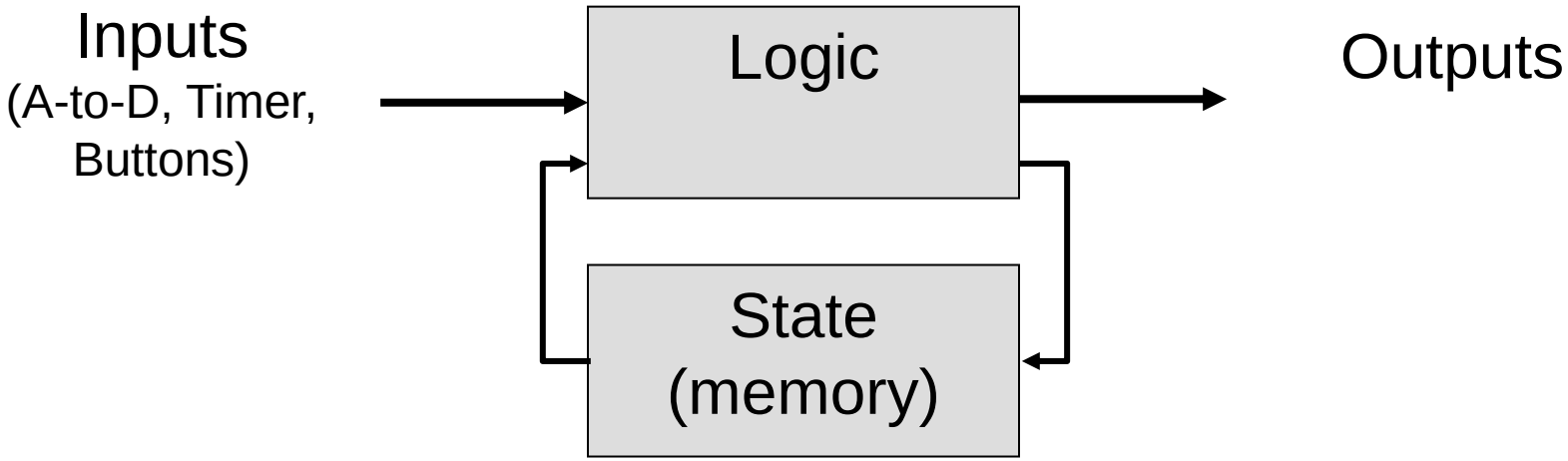
State Machines

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

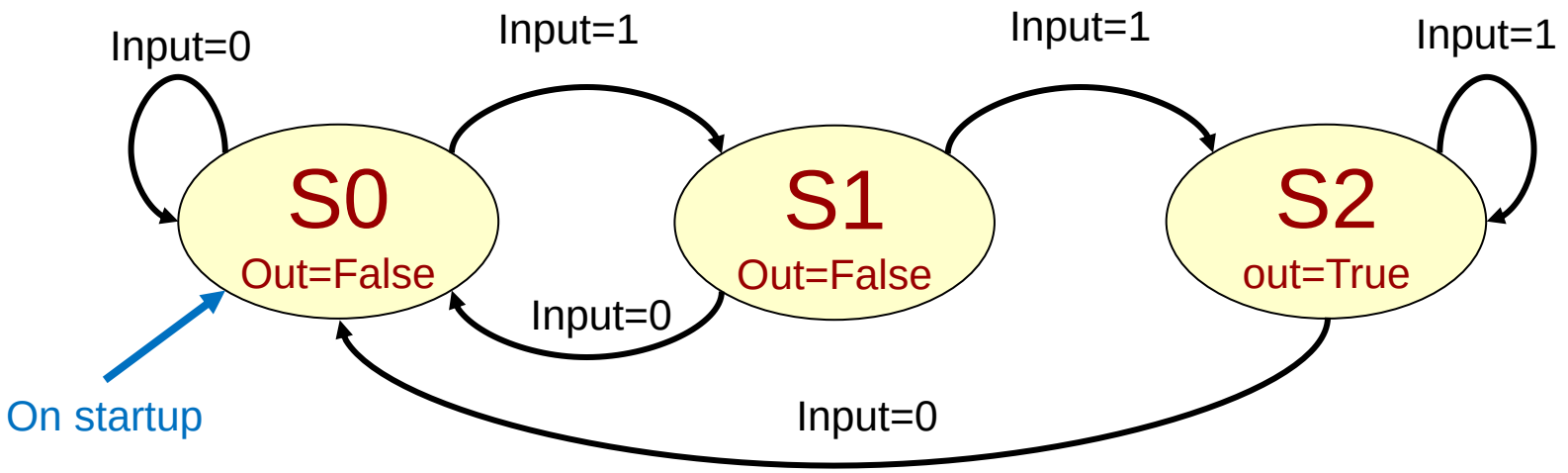
- A system that utilizes state is often referred to as a state machine
  - A.k.a. Finite State Machine [FSM]
- Most state machines can be embodied in the following form
  - Logic examines what's happening NOW (inputs) & in the PAST (state) to…
    - Produce outputs (actions you do now)
    - Update the 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



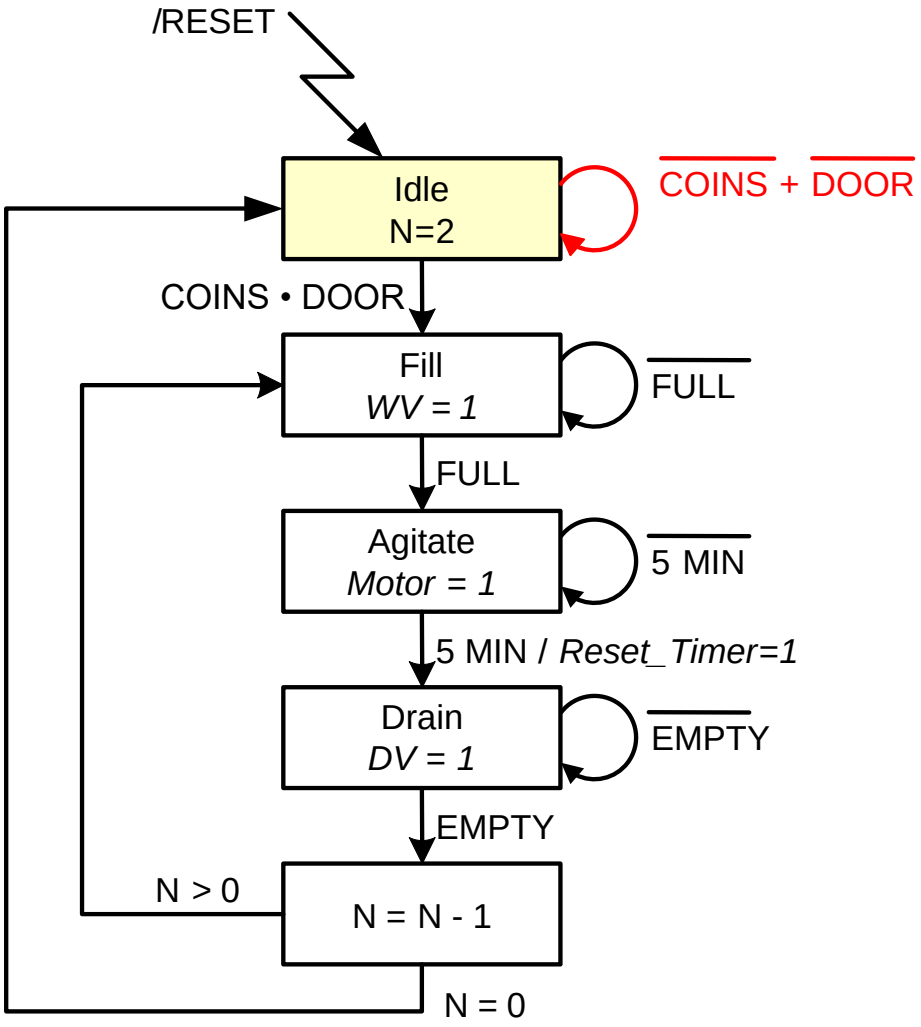
State Diagrams

- 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



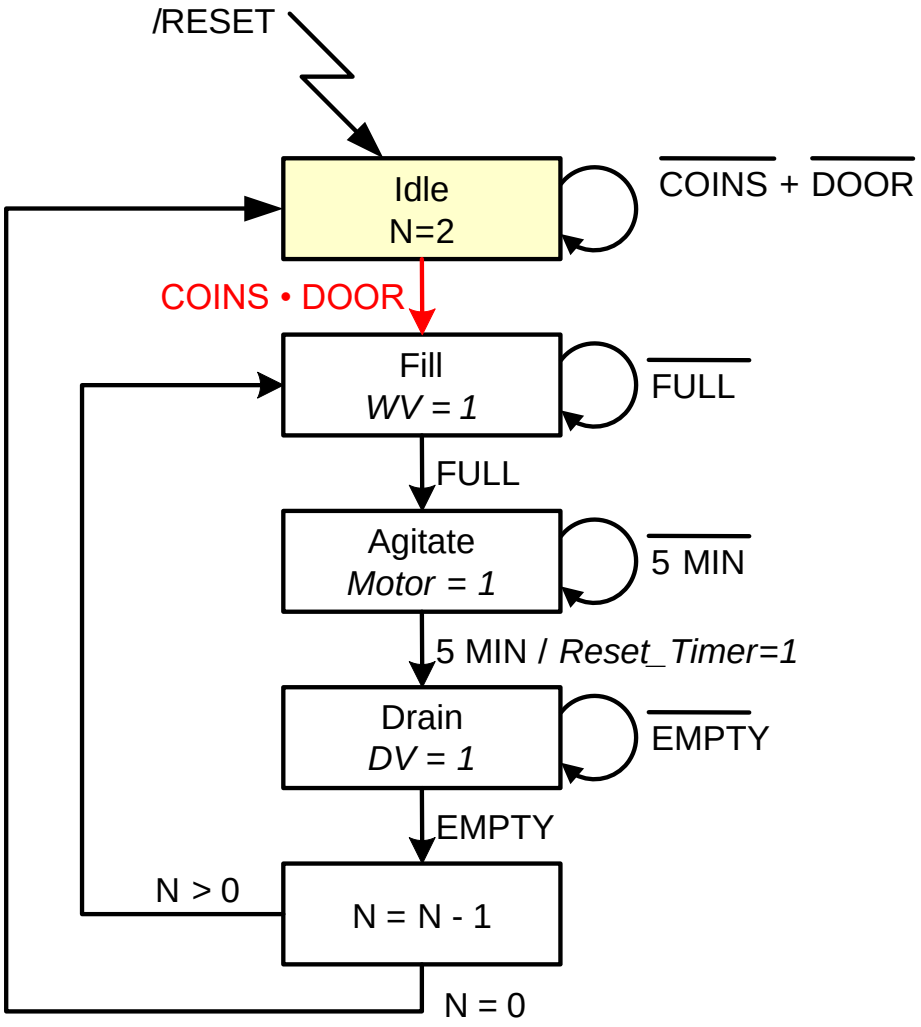
Washing Machine State Diagram



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

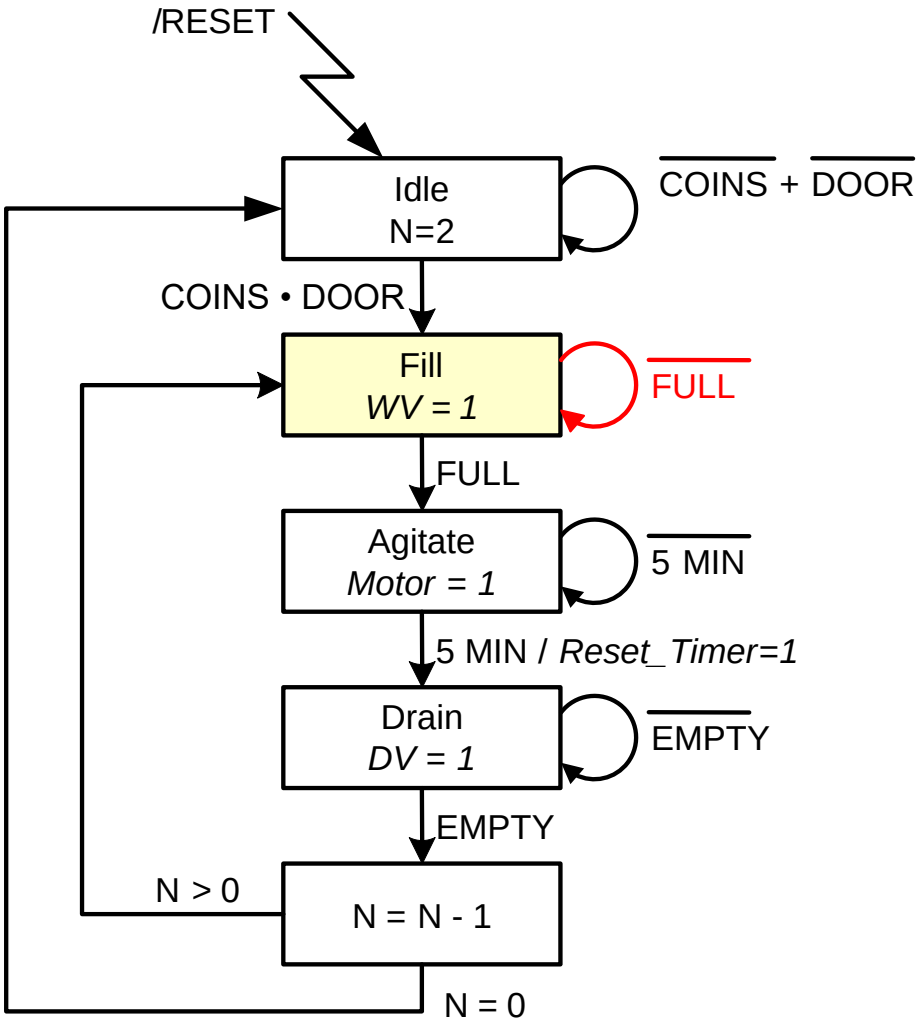
We move through the states based on the conditions. Outputs get asserted when the machine is in that state and the transition is true.

Washing Machine State Diagram



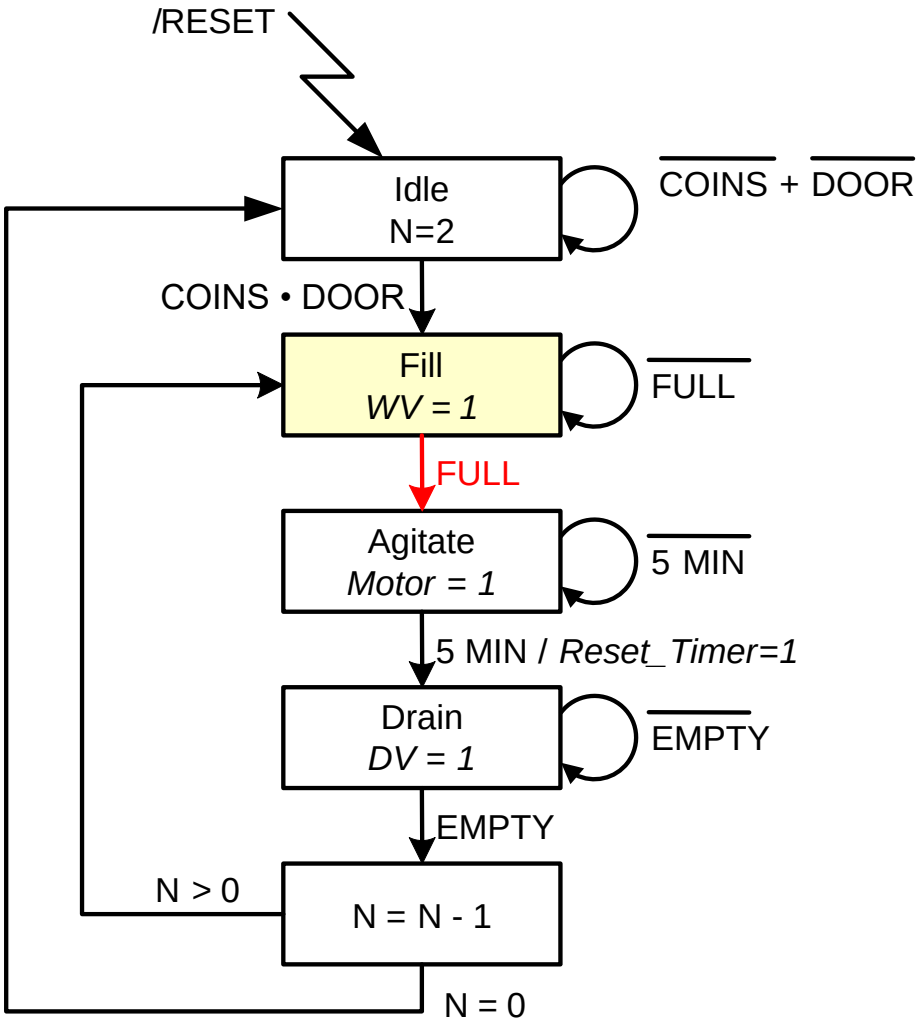
Move to the Fill state when there is enough money (coins) and the door is closed

Washing Machine State Diagram



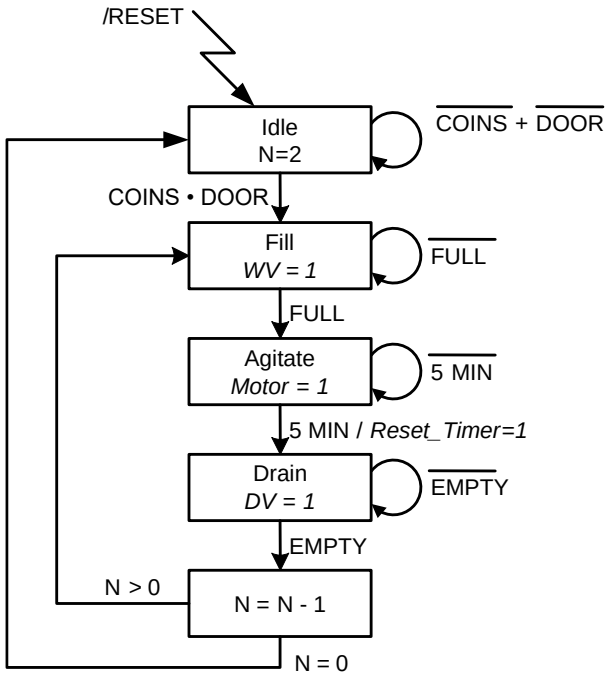
Stay in the Fill state until it is full…also set the Water Valve Open output to be true

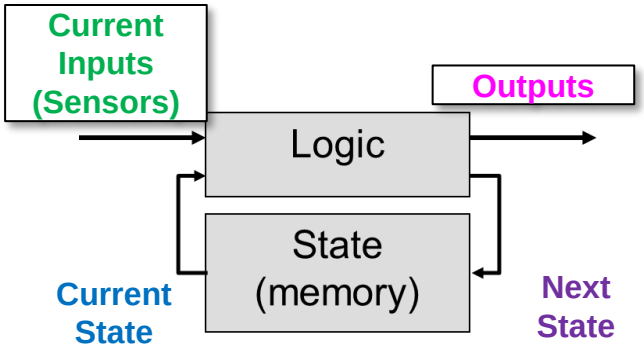
Washing Machine State Diagram

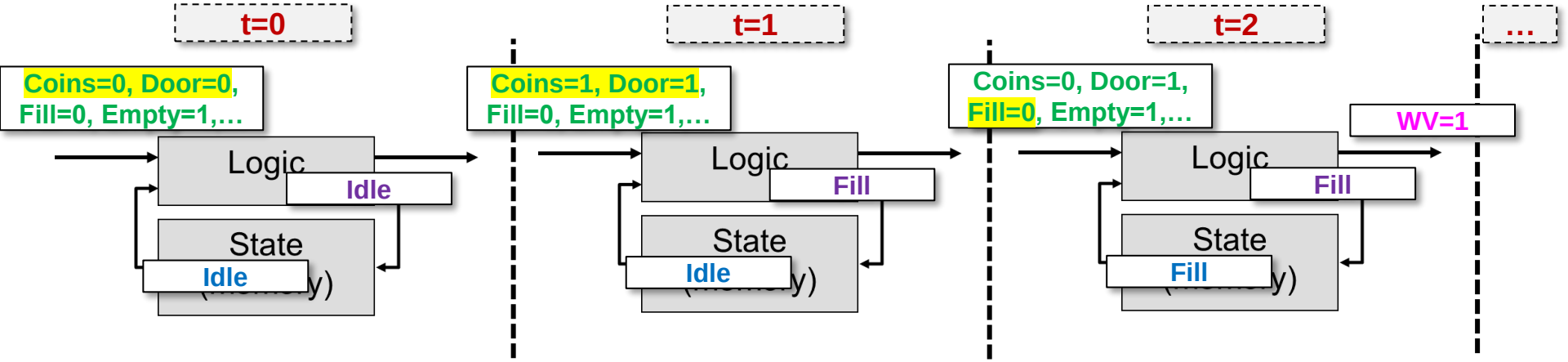


Move to the Agitate state after it is full

State Machine Operation

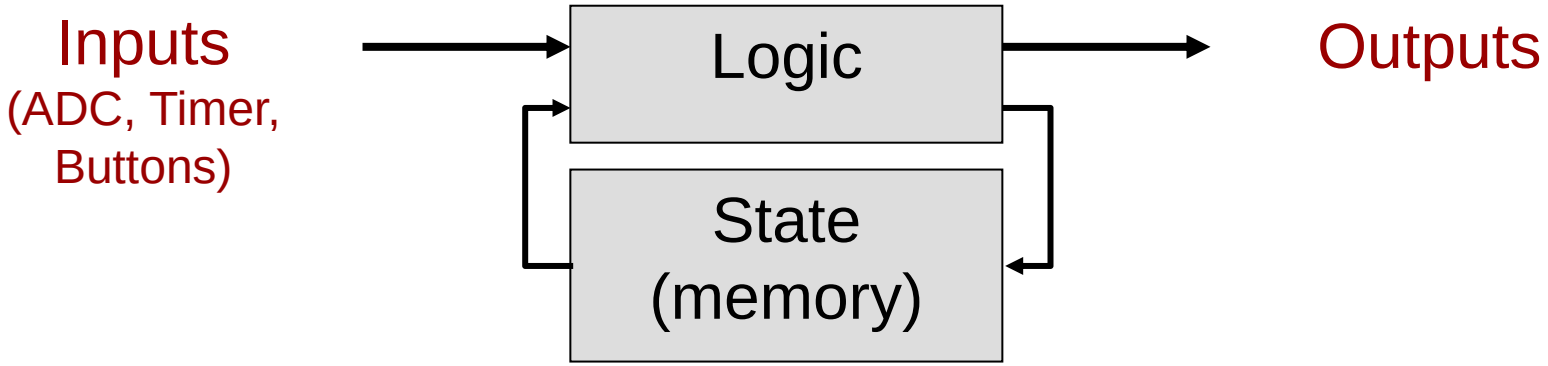






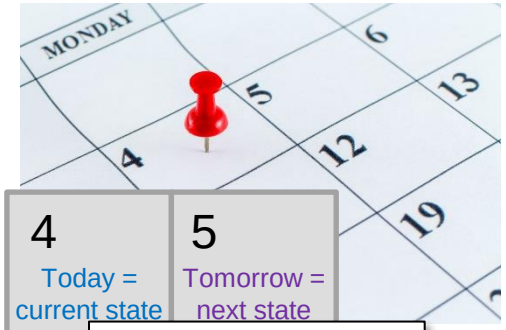
Software vs. Hardware

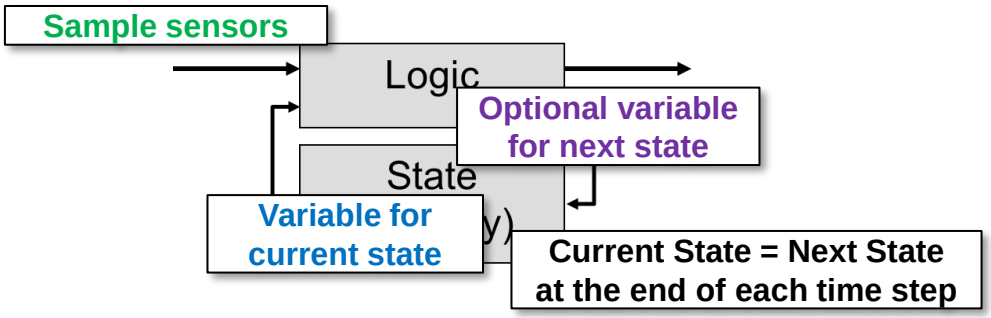
- Software
  - State = just a variable(s)
  - Logic = if statements to update the next state
    - if(state == 0 && input == 1) { state = 1; }
  - Transitions triggered by input or timers
  - **We'll start by implementing state machines in SW**
- Hardware
  - State = Register (D-Flip-Flops)
  - Logic = AND/OR Gates to produce the next state & outputs
  - Transitions triggered by clock signal
  - **More on this later in the semester**



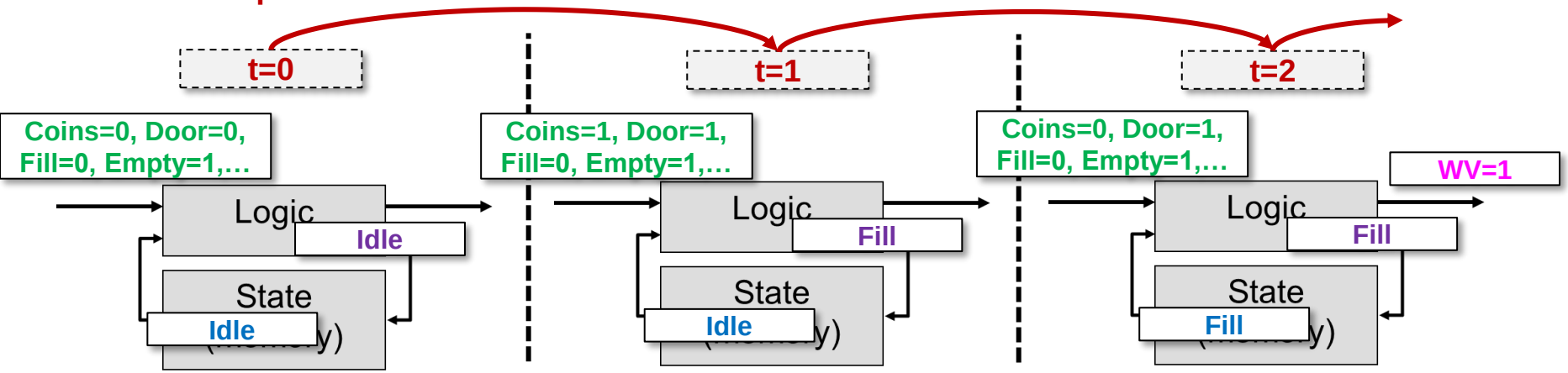
Coding State Machines 1

- Use a variable to store current state and, optionally, a second variable to hold next state
- Use while loop where each iteration is one step in the operation of the state machine



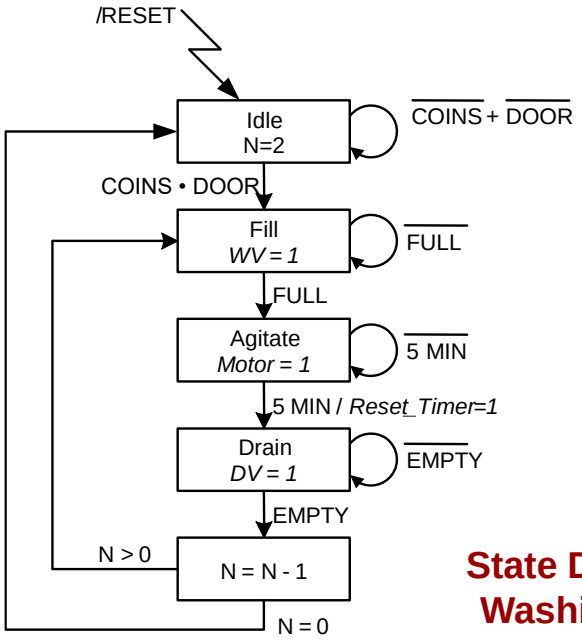


While loop iterations



Coding State Machines 2

- Store 'state' in some variable and assign numbers to represent state (0=Idle, 1=Fill, etc.)
- Use a **while** loop and delays (or timer) to repetitively sample inputs, update state, and set outputs



State Diagram for a Washing Machine

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

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

Enumerations

- In C/C++, enumerations associate an integer code (number) with a symbolic name
- Syntax
- `enum [optional_collection_name] {SymName1, SymName2, … SymNameN}`
  - SymName1 = 0
  - SymName2 = 1
  - …
  - SymNameN = N-1
- Use symbolic item names in your code and compiler will replace the symbolic names with corresponding integer values…**makes the code much more readable!**

```
const int IDLE=0;
const int FILL=1;
const int AGITATE=2;
...

int state = IDLE;
...
if(state == FILL && fullSensor == true) {
  state = AGITATE;
}
```

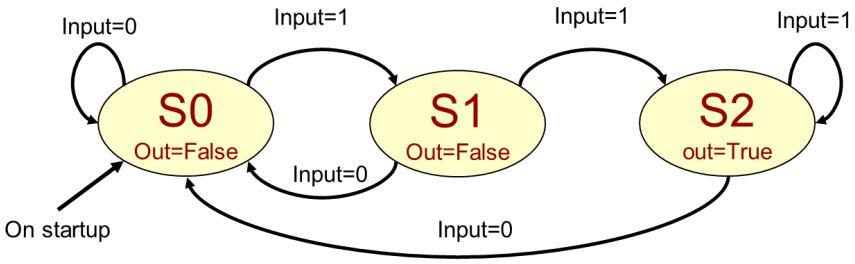
Hard coding symbolic state names with given codes

```
// First enum item is associated with code 0
enum States {IDLE, FILL, AGITATE, DRAIN, DEC};


int state = IDLE;  // same as state = 0;
...
if(state == FILL && fullSensor == true) {
   state = AGITATE;   // same as state = 2;
}
```

Using enumeration to simplify

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 input transitions from that state
  - Produce appropriate output from that state




```
enum { S0, S1, S2 };
// input = PD0,  output = PD7
int main()
{ // be sure to init. state
  unsigned char state=S0;
  unsigned char input;
  while(1)
  {
    _delay_ms(10); // use approp. time
    input = PIND & (1 << PD0);
    if(state == S0){
      PORTD &= ~(1 << PD7);
      if( input ){ state = S1; }
    }
    else if(state == S1){
      PORTD &= ~(1 << PD7);
      if( input ){ state = S2; }
      else { state = S0; }
    }
    else { // state == S2
      PORTD |= (1 << PD7);
      if( !input ) { state = S0; }
    }
  } return 0;
}
```

Select current state

Select input val.

State Machines as a Problem Solving Technique

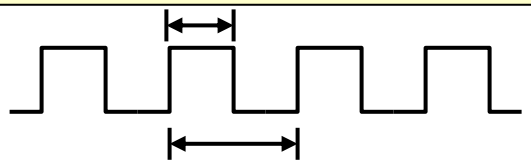
- Modeling a problem as a state machine is a powerful problem-solving tool
- When you need to write a program, design HW, or solve a more abstract problem at least consider if it can be modeled with a state machine
  - Ask questions like:
    - What do I need to remember to interpret my inputs or produce 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.]

Tunnel Vision

- Consider a program that checks two buttons
  - When button 1 is pressed, blink an LED 10 times at 2 HZ
  - When button 2 is pressed, blink an LED 15 times at 5 HZ
- **Desired behavior**: If during the blinking of one LED the other button is pressed, immediately stop the current blink cycle and start the blink cycle of the other button.
- Problem: Will using the for loops to the right allow the desired behavior?

```
// Ad-hoc implementation
int main()
{
  while(1)
  {
     int i;
     if(checkInput(1) == 0){
       for(i=0; i < 10; i++)
       {
          blink(250); // on for 250, off for 250
          // What if other button is now pressed
       }
     }
     if(checkInput(2) == 0){
       for(i=0; i < 15; i++)
       {
          blink(100);  // on for 100, off for 100
          // What if other button is now pressed
       }
     }
     // delays are in the blink() functions
     // so no delay needed here
  }
  return 0;
}
```

T = 0.25 second = 250 ms



1 cycle
F = 2 Hz implies
T = ½ = 0.5 seconds

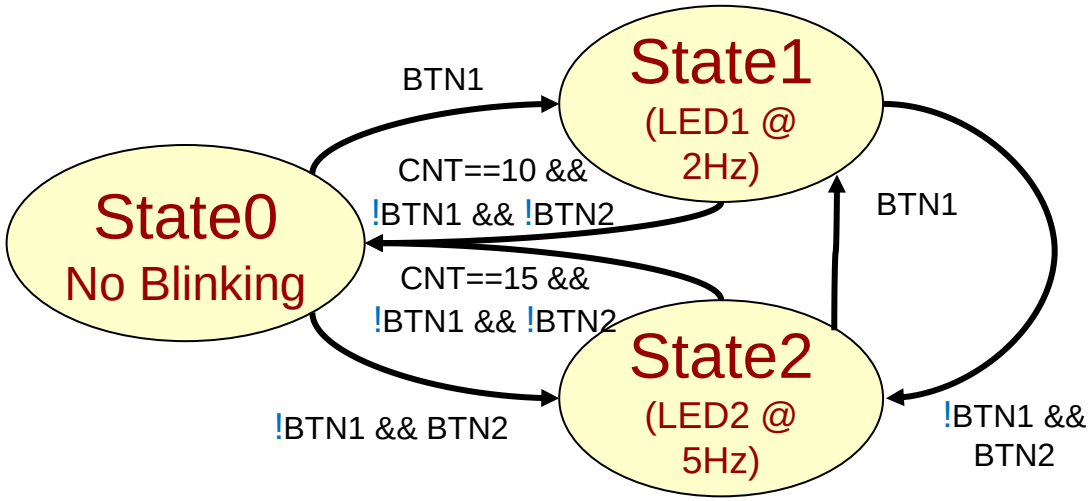
Example: Ad-hoc Implementation

- Could add checks after each blink but this becomes clunky in larger, more complex examples
- **Better approach**: Formulate the design as state machine and do NOT use additional loops (i.e. only the main while loop)

```
// Ad-hoc implementation
int main()
{
  while(1)
  {
     int i;
     if(checkInput(1) == 0){
       for(i=0; i < 10; i++)
       {
          blink(250); // on for 250, off for 250
          if(checkInput(2) == 0)
          {
             break;
          }
       }
     }
     if(checkInput(2) == 0){
       for(i=0; i < 15; i++)
       {
          blink(100);  // on for 100, off for 100
          if(checkInput(1) == 0)
          {
             break;
          }
       }
     }
     // delays are in the blink() functions
     // so no delay needed here

     // Problem:  what if button that caused
     // break from for  loop is released right now
  }
  return 0;
}
```

Example: FSM implementation

- Formulated as a state machine:
  - Separate code to update state and then perform actions based on state
- Tip: Avoid loops other than the primary `while` and instead use state, counters, and `if` statements




```
int main()
{
   int state = 0, cnt = 0;
   while(1)
   {
     // Update the state based on inputs
     if (checkInput(1) == 0) {
        state = 1; cnt = 0;
     }
     else if (checkInput(2) == 0) {
        state = 2; cnt = 0;
     }
     else if( (state == 1 && cnt == 10) ||
              (state == 2 && cnt == 15 ) ) {
        state = 0; cnt = 0;
     }

     // Use state to determine output actions
     if(state == 1) {
        blink(250); // on for 250, off for 250
        cnt++;
     }
     else if(state == 2) {
        blink(100); // on for 100, off for 100
        cnt++;
     }
   }
   return 0;
}
```

Assume if no transition it true, we intend to stay in the same state.

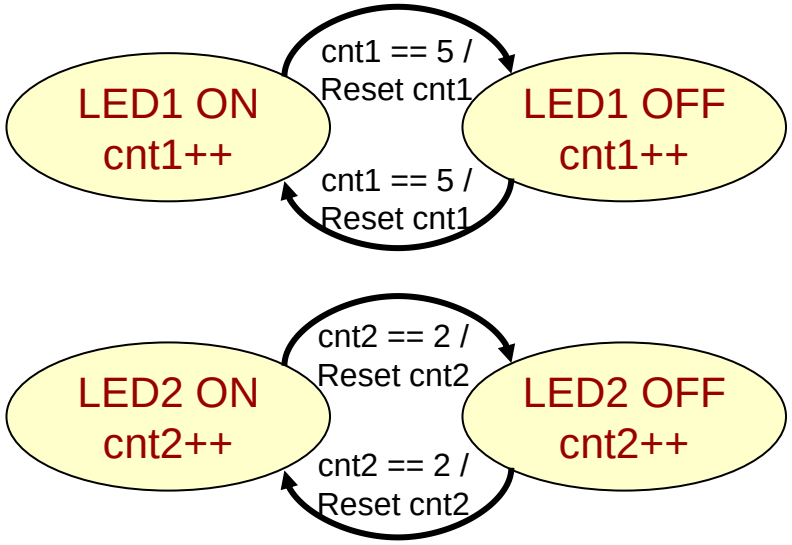
Operations at Different Rates (1)

- Consider a program to blink one LED at a rate of 2 Hz and another at 5 Hz at the same time
- **Problem**: Does the code to the right work correctly?
  - No! When one LED blinks the other will be off

```
int main()
{
    while(1)
    {
        LED1_ON();
        _delay_ms(250);
        LED1_OFF();
        _delay_ms(250);

        LED2_ON();
        _delay_ms(100);
        LED2_OFF();
        _delay_ms(100);

    }

    return 0;
}
```

Operations at Different Rates (2)

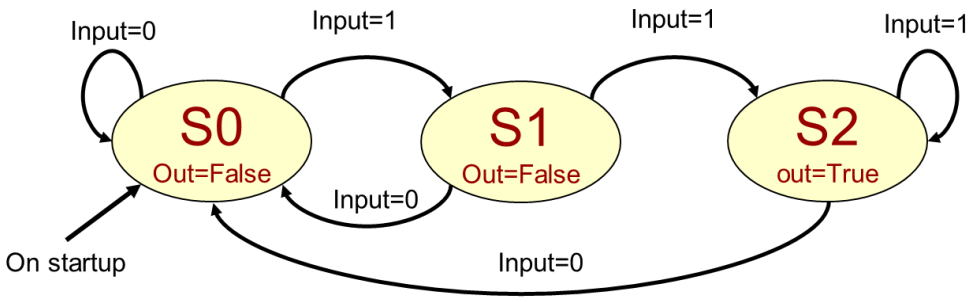
- Use separate state machines running in parallel
- Given various rates of operations, find the GCD of the various rates and loop with that delay, using counts to track time




```
int main()
{
    int cnt1 = 0, cnt2 = 0;

    // set initial state of LEDs as "on"
    int s1 = 1, s2 = 1;
    LED1_ON();
    LED2_ON();

    while(1)
    {
        cnt1++;
        if(cnt1 == 5)
        {
            if(s1) { LED1_OFF(); s1 = 0; }
            else   { LED1_ON();  s1 = 1; }
            cnt1 = 0;
        }

        cnt2++;
        if(cnt2 == 2)
        {
            if(s2) { LED2_OFF(); s2 = 0; }
            else   { LED2_ON();  s2 = 1; }
            cnt2 = 0;
        }

        // Delay the minimum granularity
        _delay_ms(50);
    }
    return 0;
}
```

Summary Definition

- To specify a state machine, we must specify 6 things:
  - 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)



All the info in the state diagram is presented in the sets and tables to the right

Inputs: {0, 1}

States: {S0, S1, S2}

Outputs: {False, True}

Initial State: S0

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

State Transition Function

State	Outputs
S0	False
S1	False
S2	True

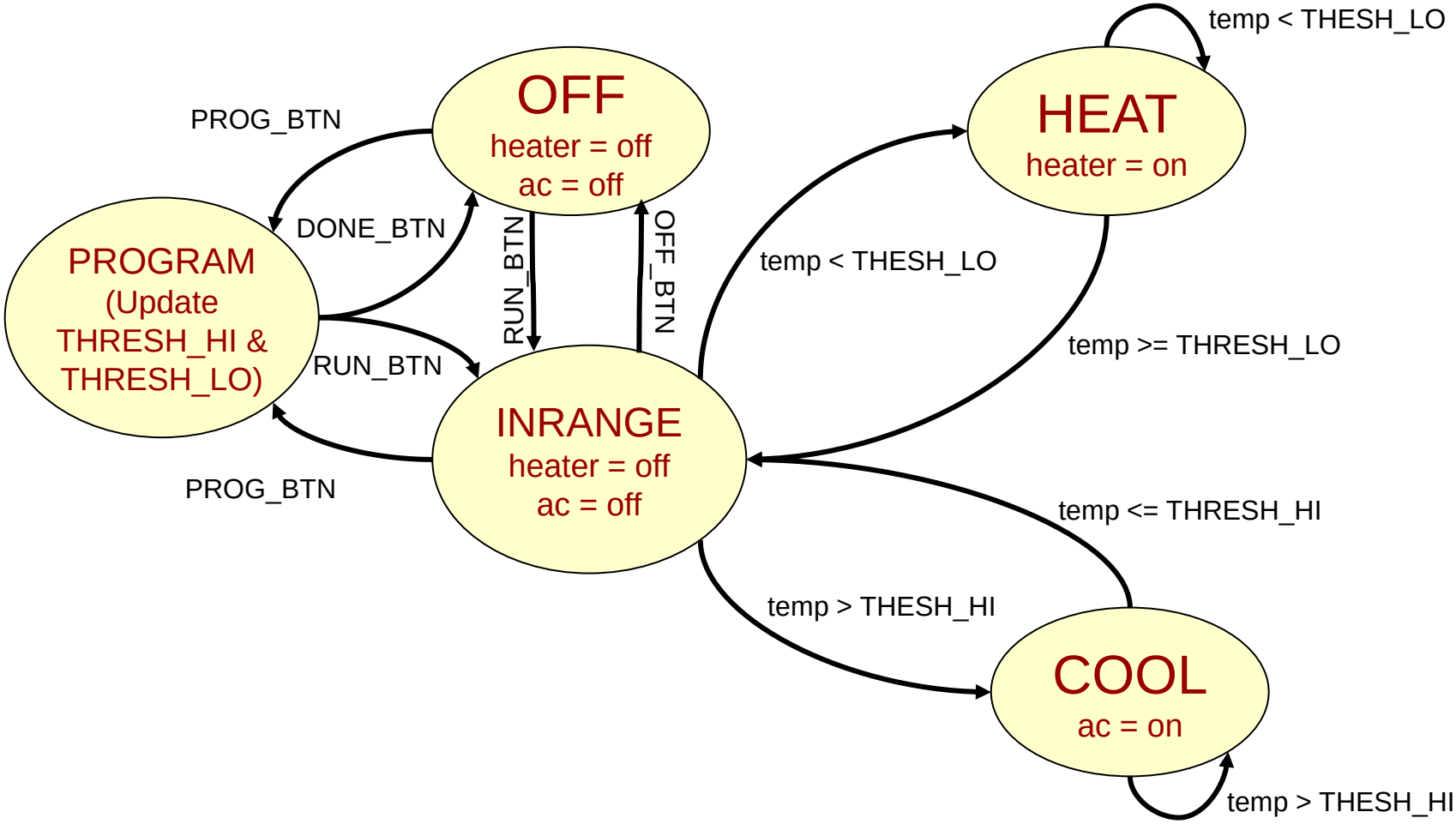
Output Function

HW (Instruction Cycle) & Software (String Matching)

MORE EXAMPLES IF TIME

Thermostat

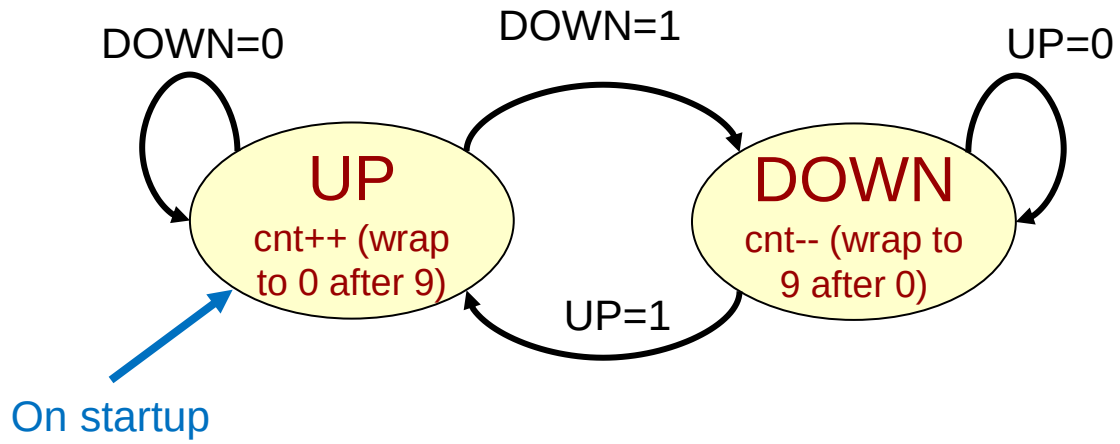
- Sample state machine to control a thermostat

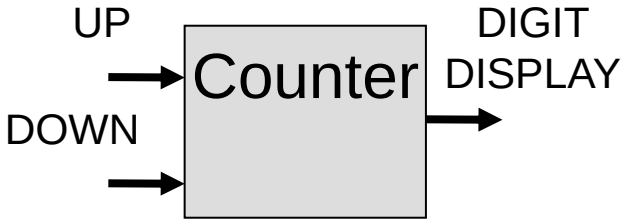


Counter Example

- 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

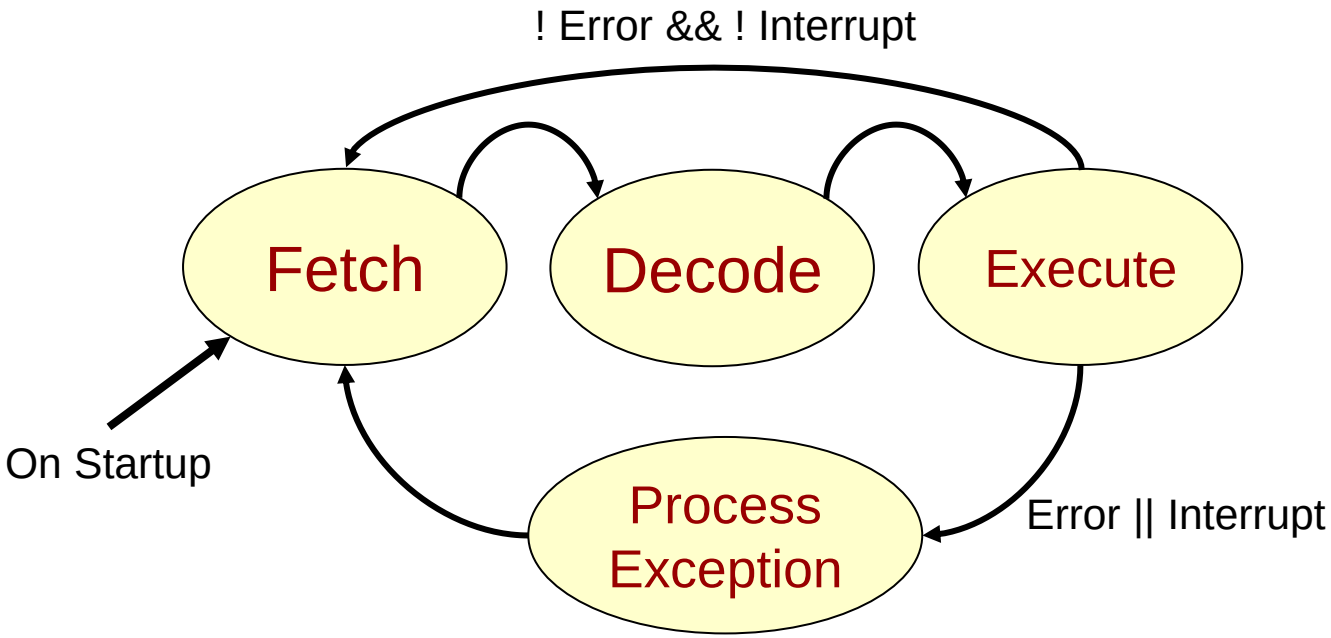
State Machine to count up or down (and continue counting) based on 2 pushbutton inputs: UP and DOWN





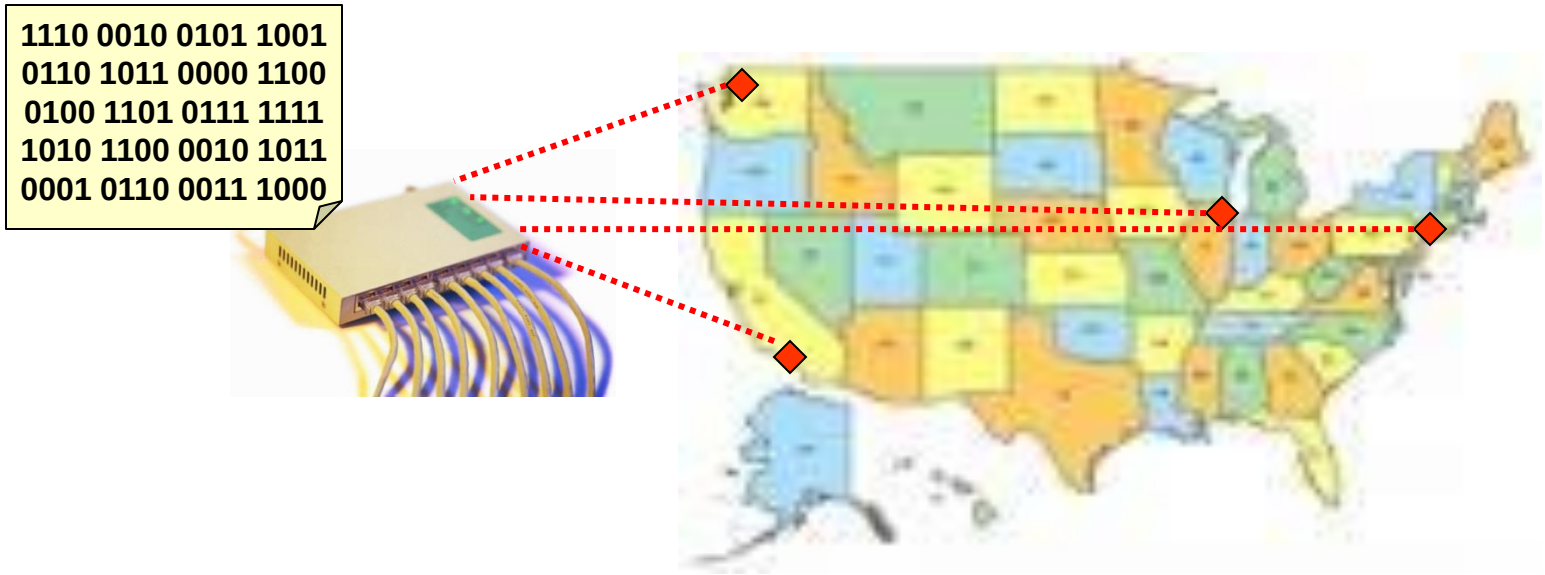
More State Machines

- 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
...
```

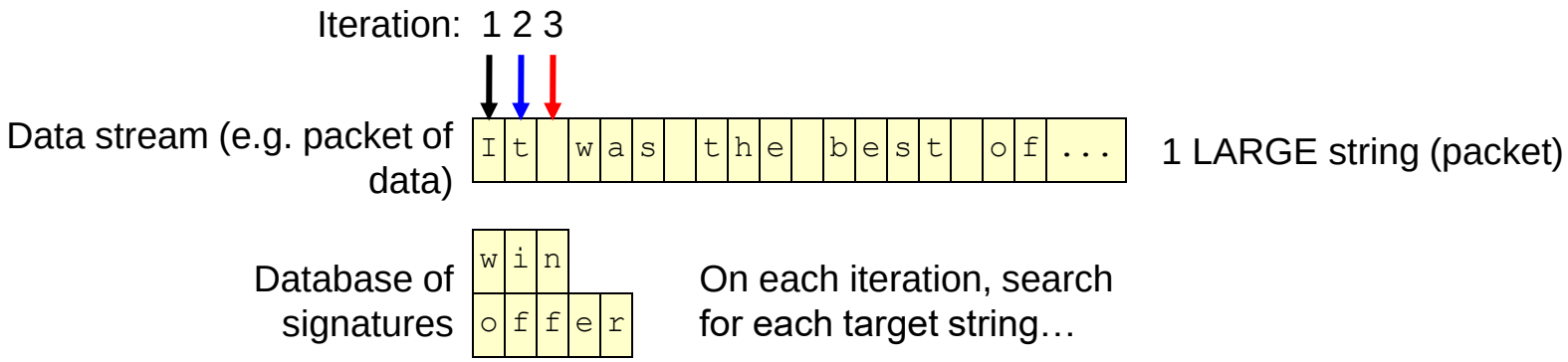
Database of signatures

```
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()



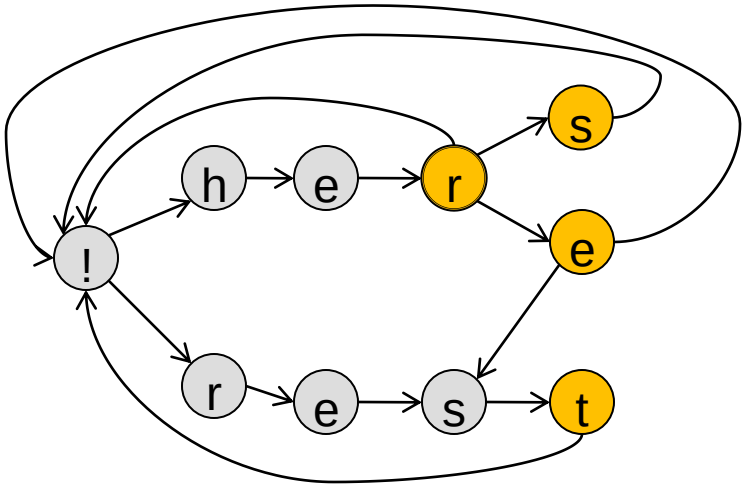
Data Stream = N chars with T Targets => Run Time proportional to N*T

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…

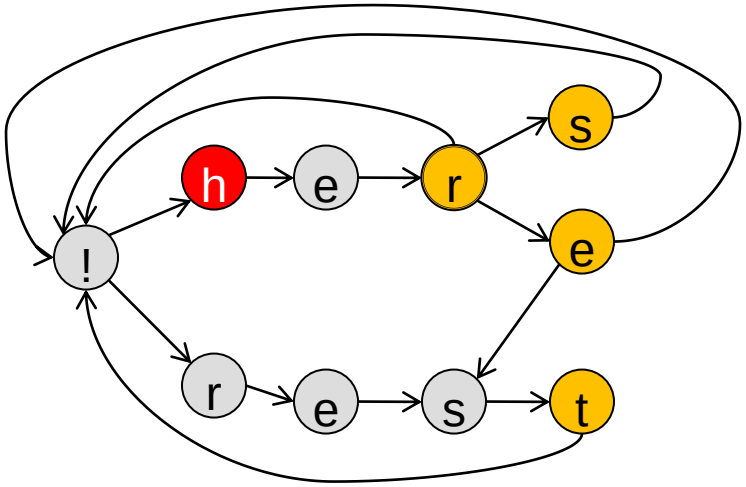


Internal state

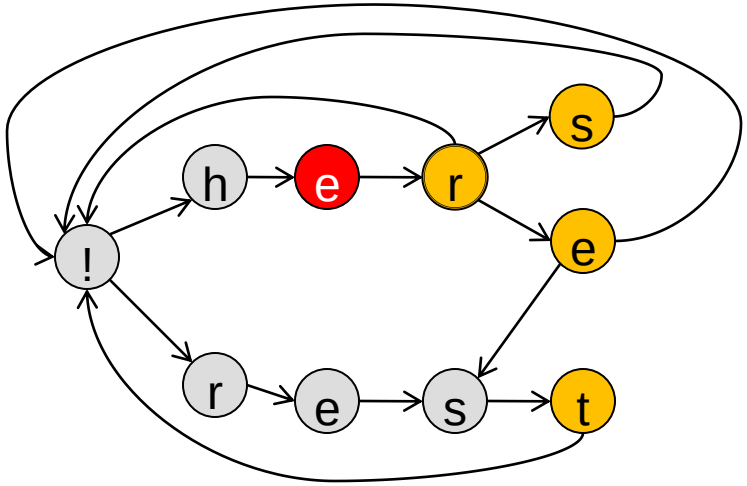
Terminal / Pattern Found State (i.e. output should be True)

Finite State Automaton

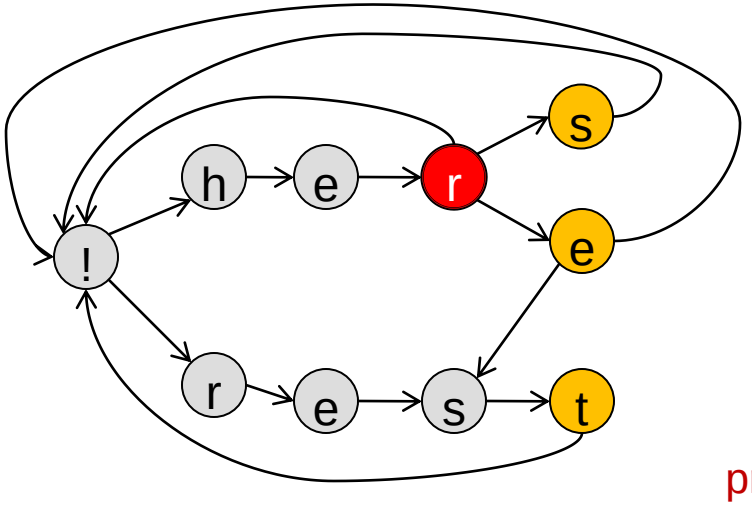
- Data Stream: heresthers



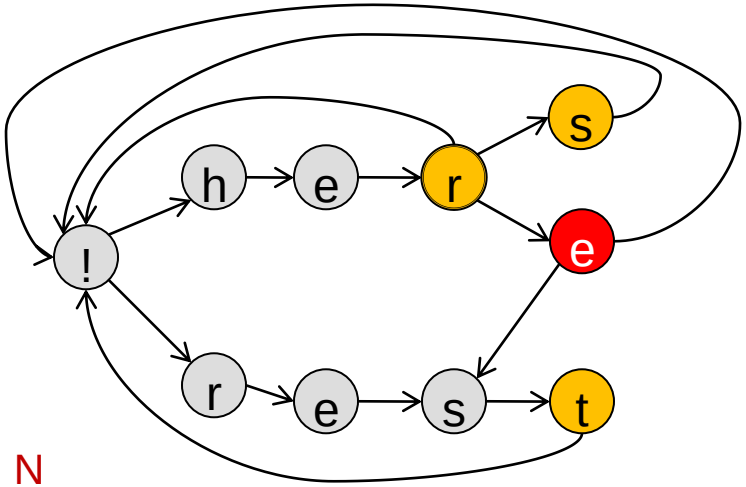
1



2



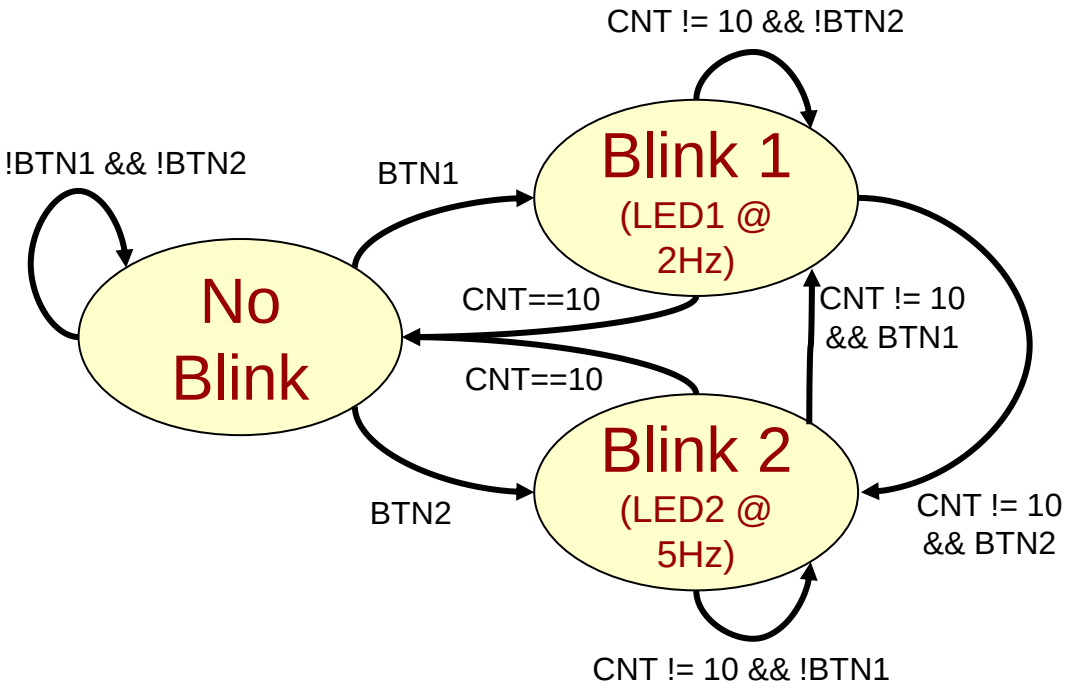
3



4

Run-Time proportional to N

- Formulated as a state machine:
  - Separate code to update state and then perform actions based on state

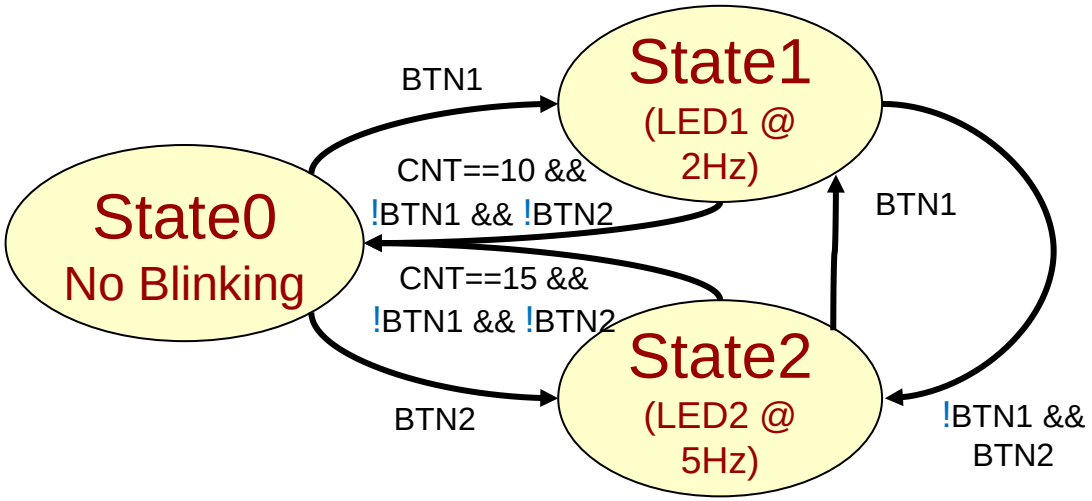


Assume if no transition it true, we

```
// State machine implementation
int main()
{
  int state = 0, cnt = 0;
  while(1)
  {
     // Update the state
     if(checkInput(1) == 0) {
        state = 1; cnt = 0;
     }
     else if(checkInput(2) == 0) {
        state = 2; cnt = 0;
     }

     // Use state to determine actions
     if(state == 1) {
        blink(250); // on for 250, off for 250
        cnt++;
        if(cnt == 10) {
           cnt = 0;
           state = 0;
        }
     }
     else if(state == 2) {
        blink(100); // on for 100, off for 100
        cnt++;
        if(cnt == 15) {
           cnt = 0;
           state = 0;
        }
     }
  }
  return 0;
}
```

Example: FSM implementation

- Formulated as a state machine:
  - Separate code to update state and then perform actions based on state
- Tip: Avoid loops other than the primary `while` and use state and `if` statements, instead




```
int main()
{
   int state = 0, cnt = 0;
   while(1)
   {
     // Use state to determine actions
     if (state == 0) {
        cnt = 0;
        if (checkInput(1) == 0)
           state = 1;
        else if (checkInput(2) == 0)
           state = 2;
     }
     else if(state == 1) {
        if (checkInput(2) == 0) {
           state = 2; cnt = 0;
        }
        else {
           blink(250); // on/off for 250
           cnt++;
           if (cnt == 10) { state = 0; }
        }
     }
     else if(state == 2) {
        if (checkInput(1) == 0) {
           state = 1; cnt = 0;
        }
        else {
            blink(100); // on/off for 100
            cnt++;
            if (cnt == 15) { state = 0; }
        }
     }
   }
   return 0;
}
```

Assume if no transition it true, we intend to stay in the same state.