CMPE-013/L

Introduction to "C" Programming

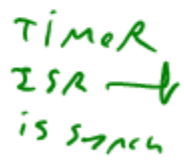
Max Lichtenstein

Piazza Poll: Which processes are synchronous?

```
static int event = 0;

int main(void)
{
    while(1){
        event = runProcessA();
        if(event){
            runProcessB(event);
            event = 0;
        }
    }
}

void __ISR(_TIMER_1_VECTOR, IPL4AUTO) Timer1Handler(void){
    event = runProcessC();
}

void __ISR(_CN_1_VECTOR, IPL4AUTO) ChangeNotification1Handler(void){
    event = runProcessD();
}
```

Synchronous = runs periodically

Asynch = runs when some external event causes it



Roadmap

- Announcements,
- Lots of Lab 6 Notes ✓
- Asynchronous vs Synchronous processes review ✓
- State Machines
  - States vs events ~
  - State machines as models ~
  - Stoplight example ~
  - Classifiers
- Break
  - A few more examples
  - Event checking SMs
  - Event receiving SMs
- Lab07

Announcements

- Lab 7 is out
  - Due next Monday
  - No autochecker!
    - (can still check submissions, but not code)
- Office hours rescheduled:
  - NOT Friday
  - Instead, Thursday

long

out of town

START

After

Class

2-3

Max Lichtenstein

Notes on lab 6: ButtonsCheckEvents()

- Don't call BUTTON_STATES() from inside ButtonsCheckEvents()!

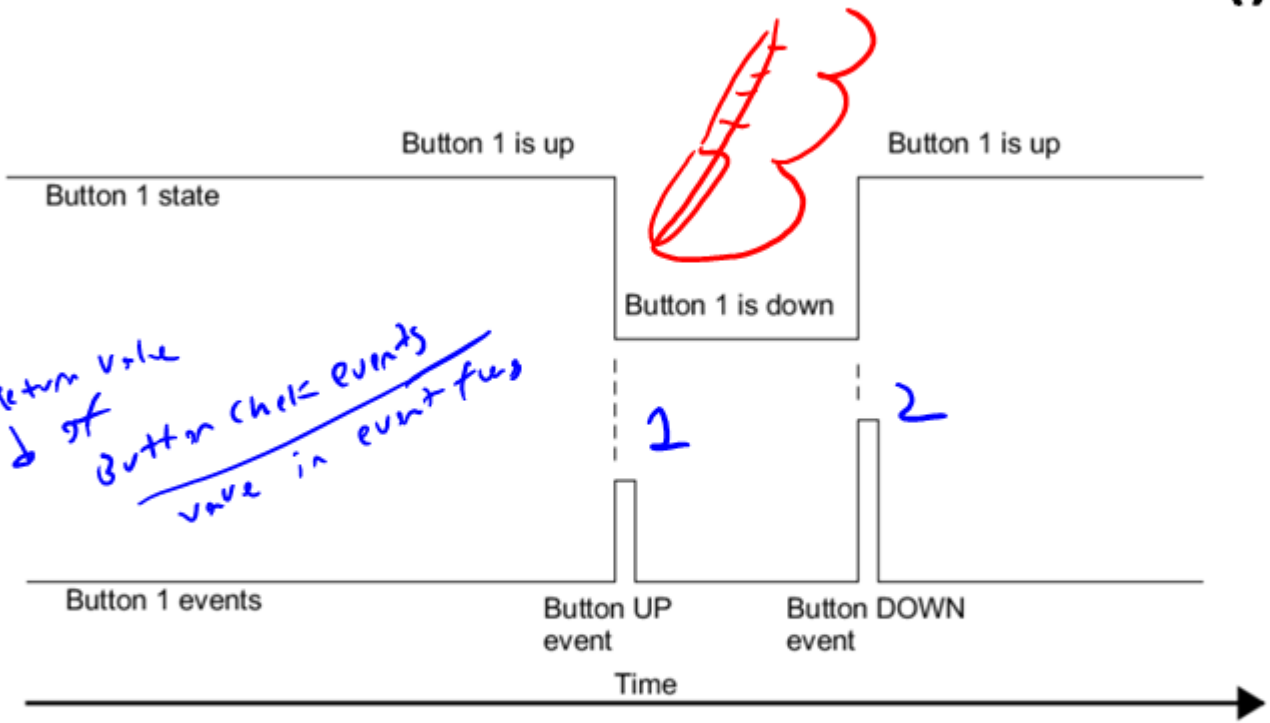
```
 * This is the interrupt for the Timer1 peripheral. It
 * checks for button events and stores them in a
 * module-level variable.
 */
void __ISR(_TIMER_1_VECTOR, IPL4AUTO) Timer1Handler(void)
{
    // Clear the interrupt flag.
    IFS0CLR = 1 << 4;

    // Check for events.
    buttonEvents = ButtonsCheckEvents(BUTTON_STATE());
}
```

Do this

main()
  ButtonCheckEvents(x30)
  (x20)

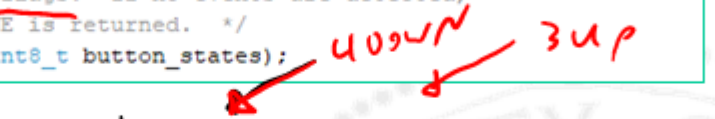
init timers
// interactive unit

Notes on lab 6: ButtonsCheckEvents()

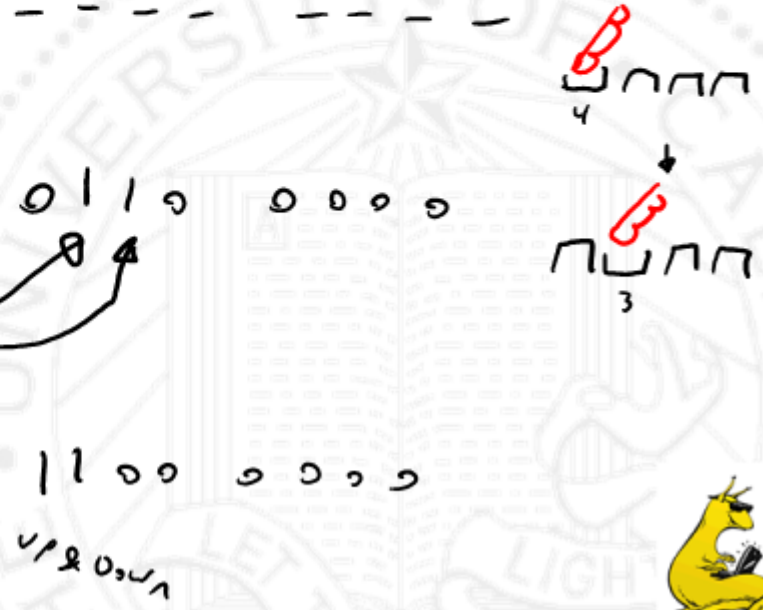


Notes on lab 6: ButtonsCheckEvents()

```
 * @return  Each bit of the return value corresponds to one ButtonEvent flag,
 *          as described in Buttons.h.  Note that more than one event can occur
 *          simultaneously, so the output should be a bitwise OR of all
 *          applicable event flags.  If no events are detected,
 *          BUTTONS_EVENT_NONE is returned.  */
uint8_t ButtonsCheckEvents(uint8_t button_states);
```

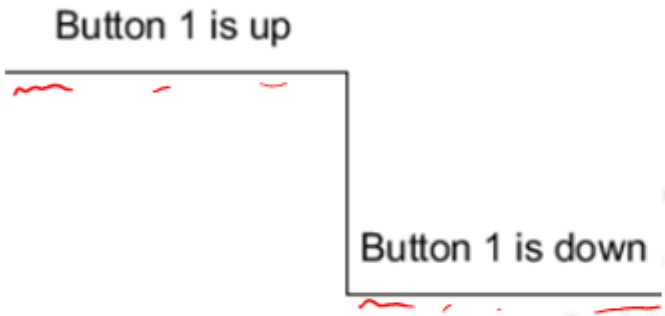


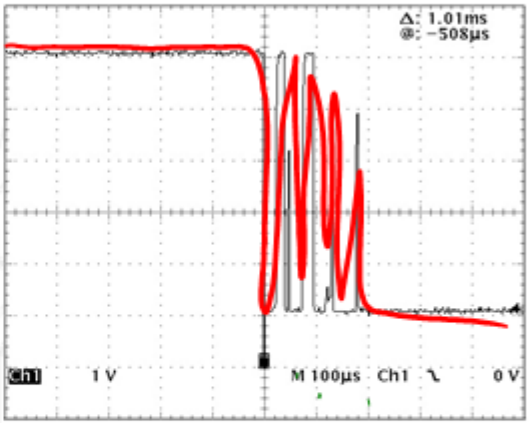

```
typedef enum {
    BUTTON_EVENT_NONE = 0x00,
    BUTTON_EVENT_1UP = 0x01,
    BUTTON_EVENT_1DOWN = 0x02,
    BUTTON_EVENT_2UP = 0x04,
    BUTTON_EVENT_2DOWN = 0x08,
    BUTTON_EVENT_3UP = 0x10,
    BUTTON_EVENT_3DOWN = 0x20,
    BUTTON_EVENT_4UP = 0x40,
    BUTTON_EVENT_4DOWN = 0x80
} ButtonEventFlags;
```



Notes on lab 6: Debouncing

IDEAL

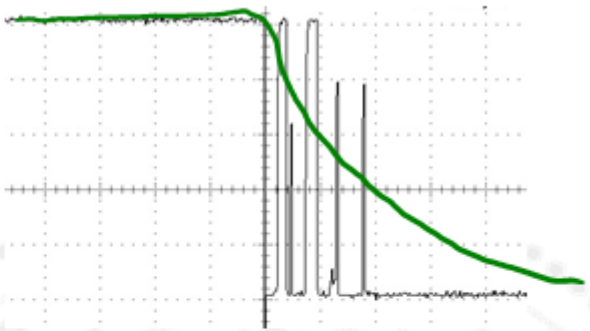




– *Debouncing is only worth 0.5 points….*

Debouncing strategies

- Get a fancy switch
  - Non-mechanical, mercury
- Hardware filtering
  - R/C lowpass filter
- Software debouncing



uint8 prev states [4];

Circular

1) Update record
   - add newest measure
2) Check if all measurements are equal

– *Debouncing is only worth 0.5 points….*

Blocking

```
counter = 0;
while (counter < 4)
    if SWITCH_STATE() ≠ prev {
        counter ++;
    }
```

external

Non-blocking

```
for (i=0; i<4; i++){
    if record[i] ≠ prev {
        set flag TRUE;
    }
}
```

internal

A few more Lab 06 Notes

- Buttons.c is a *library*
  - *No main()!* ✓
- Buttons.c does not use LEDs.h ✓
  - ButtonsTest.c doesn't need it either
- Not a lot of submissions yet… ✓

Event-Driven Programming

Event-Driven Programming

- So far, you've used code that:
  - runs once, at startup
  - Runs forever in a while loop

- But some code should run only at specific times:
  - Periodic sensor monitoring
  - React to warnings
  - Take advantage of opportunities
  - Respond to requests

Asynchronous vs Synchronous

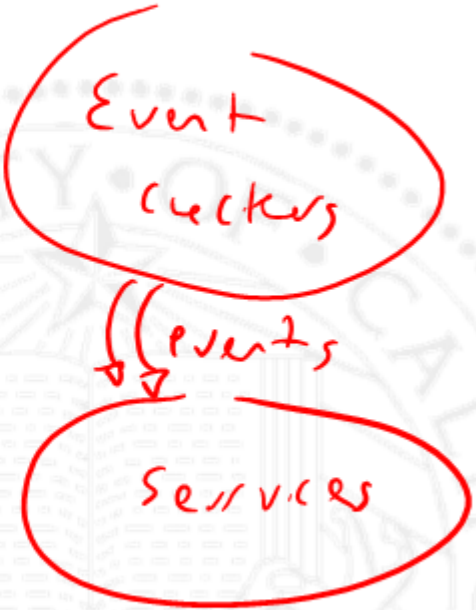
- Synchronous:
  - Code runs at periodic intervals
  - Or over a sequence of inputs (eg chars in a string)
- Asynchronous:
  - Run when an event occurs
    - Receives an event from another process
    - Event is “consumed”

Piazza Poll Results: Which processes are synchronous?

```
static int event = 0;

int main(void)
{
    while(1){
        event = runProcessA();
        if(event){
            runProcessB(event);
            event = 0;
        }
    }
}

void __ISR(_TIMER_1_VECTOR, IPL4AUTO) Timer1Handler(void){
    event = runProcessC();
}

void __ISR(_CN_1_VECTOR, IPL4AUTO) ChangeNotification1Handler(void){
    event = runProcessD();
}
```

Reactive Architecture

- Events:
  - Generated by event checkers
    - Pieces of code (or hardware)
  - Or, generated by services
  - Are fed to services
- Services:
  - Pieces of code that react to events
    - Services "consume" the events



Max Lichtenstein

Ingredients of an Event Checker

1. • Must remember previous measurement
   - – Cannot detect change from 1 measurement
   - – Needs persistent memory (static)
   - – We call this memory the checker's "state"

State = Any Memory

State = a node in a SM

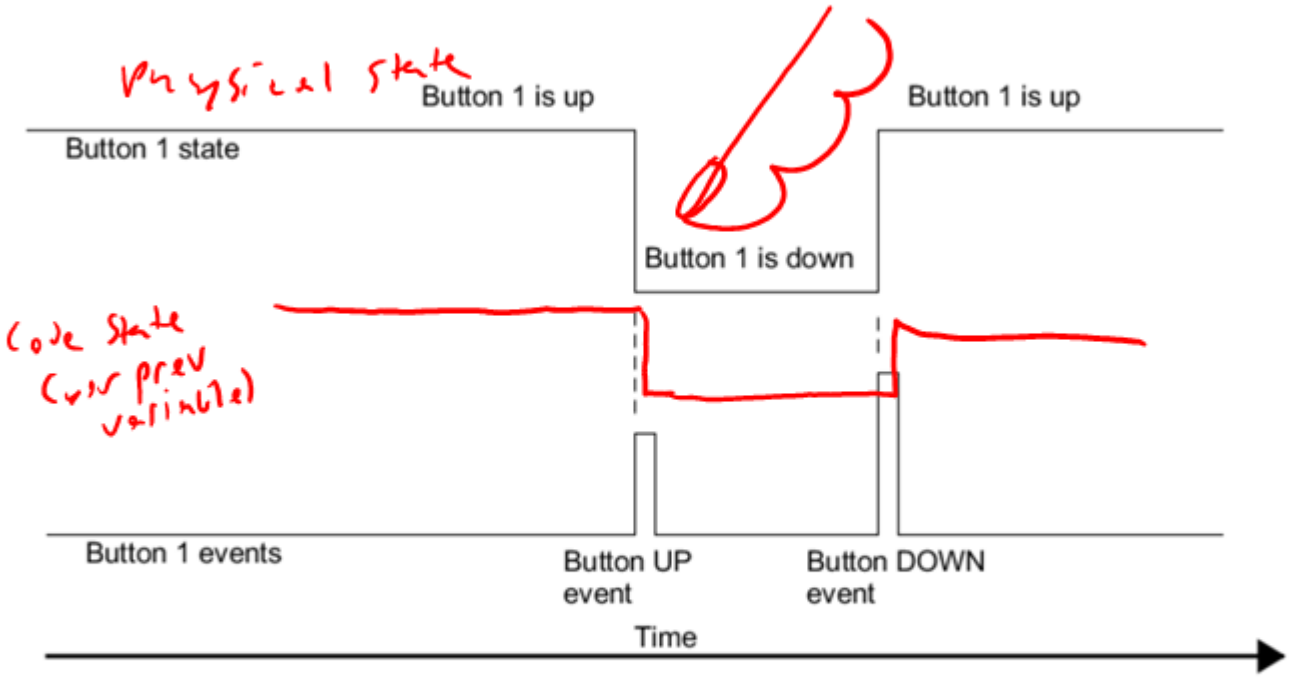
2. • Must compare previous state to current measurement
   - – That's what a change is, right?

3. • If current measurement != previous measurement:
   - – Make previous = current
   - – Generate an event

set flag to 1

add Event to queue

Event checker:



Max Lichtenstein

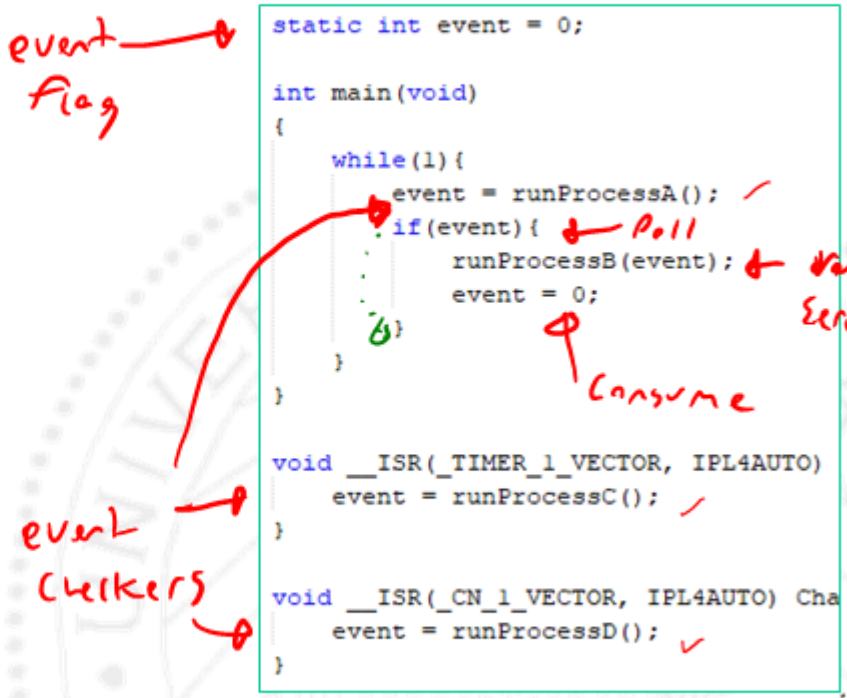
Ingredients of a Service

- When an event occurs:
  - Call service, pass in event
  - Service does some action
  - "consume" (ie, destroy) the event

Simple Reactive Architecture in C

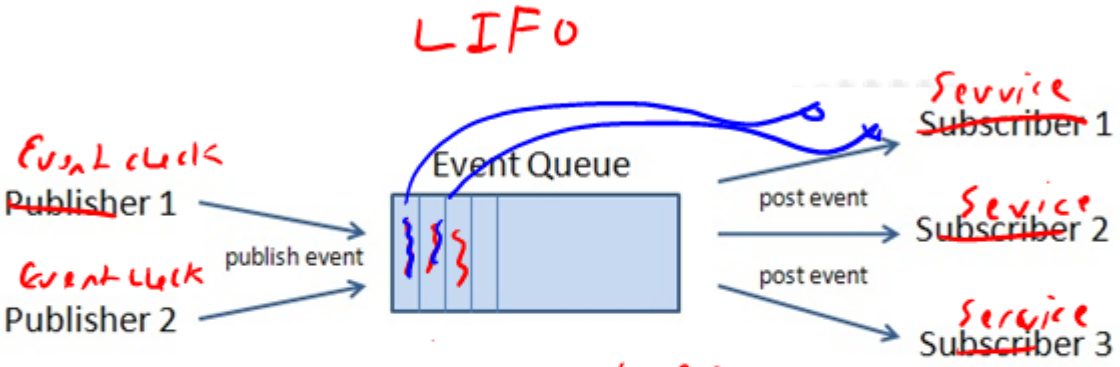
- Events are signaled with high-scope variables (module-level)
  - Flags, for specialized event checkers
  - Event Checker(s) can set flag(s) — Generate
  - Main loop polls flag(s) — react
    - Reacts if appropriate, then sets flag to 0 — Consume

Simple Reactive Architecture in C



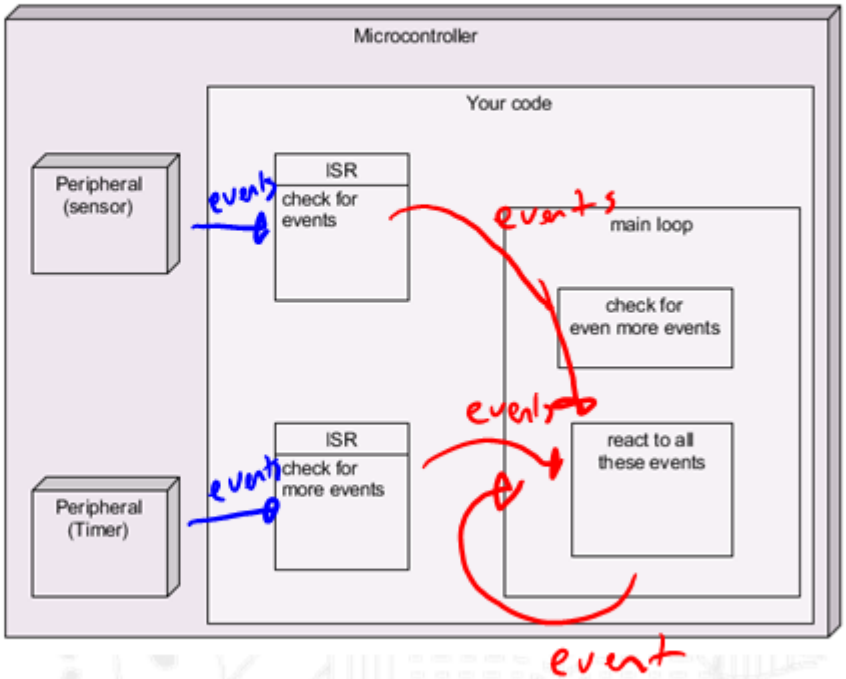
Max Lichtenstein

Event Queues



A matter of perspective

External World



Why Reactive Coding?

- We want to be fast

- Need to be able to look & act at the same

BREAK?

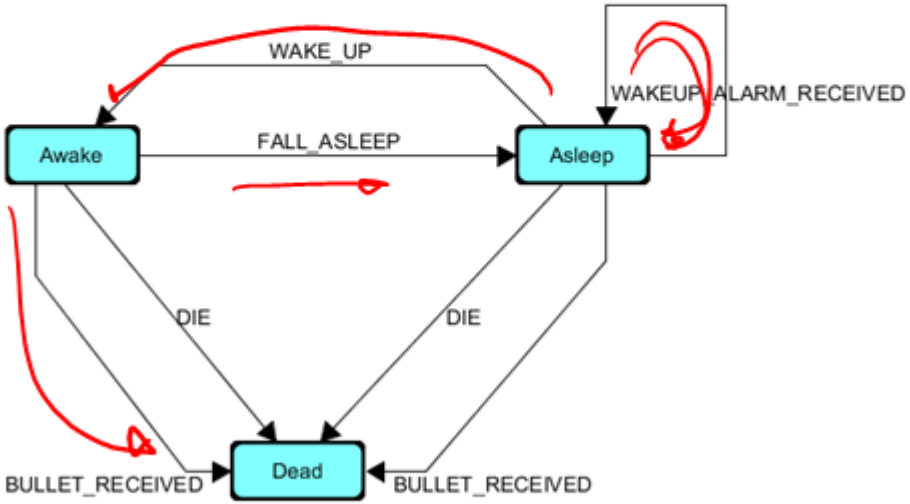
States and Events

States vs Events

- State:
  - Temporary but not instantaneous property of a system
    - For example: Sleeping, Awake, Dead
- Events:
  - Instantaneous
    - Can be internally caused changes in state:
      - For example: WAKE_UP, FALL_ASLEEP, DIE
    - Or momentary occurrences that don't necessarily cause changes in state:
      - For example, WAKEUP_ALARM_RECEIVED
    - Or external causes of changes:
      - For example: BULLET_RECEIVED

State Machine:

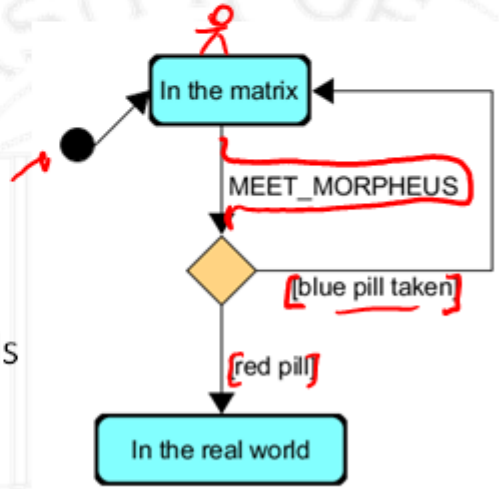
- A set of states, linked by a set of transitions (generally events):



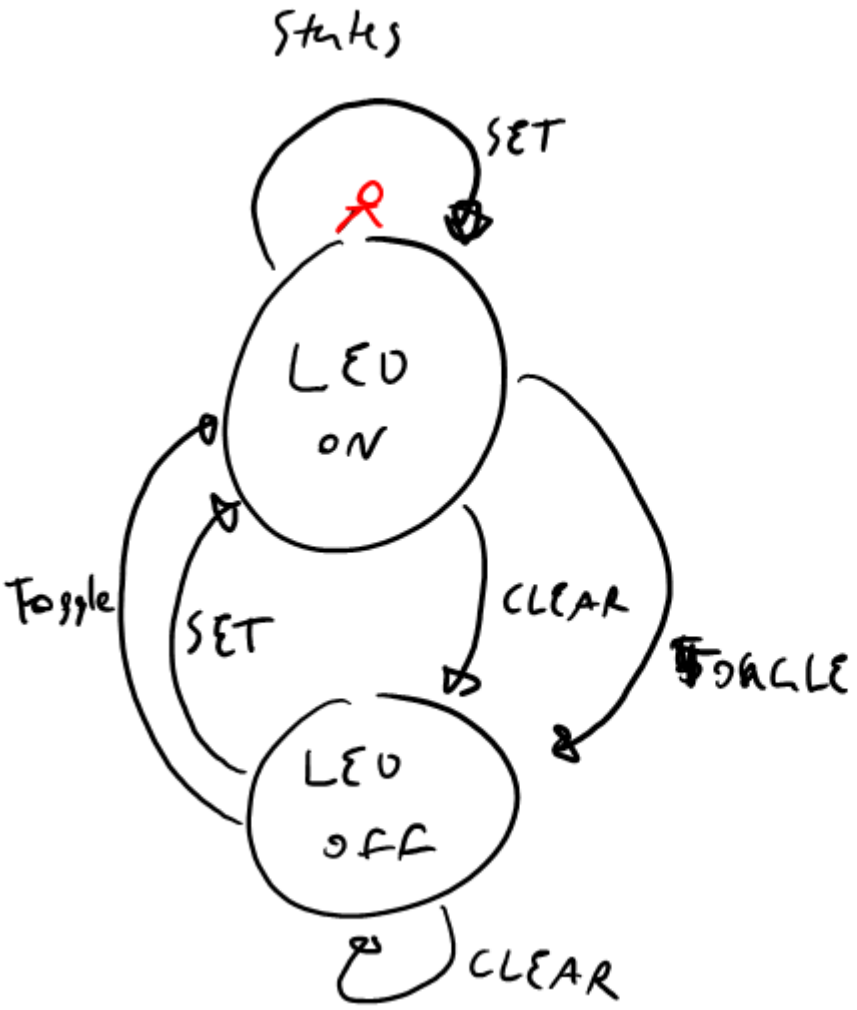
Max Lichtenstein

State Machine Diagrams

- A way of drawing state machines
- Standardized in Unified Modeling Language notation (UML)
  - States are bubbles
    - Starting state denoted by dot
  - Transitions are arrows
  - EVENTS are written on arrows
    - Can have / associated actions
  - Conditional transitions are diamonds
    - Take path of true [guard condition]



meet morpheus [Red_pill]


States
Events
SET
LED ON
Toggle
SET
CLEAR
TOGGLE
LED OFF
CLEAR
SET
CLEAR
TOGGLE

State Machines as Models

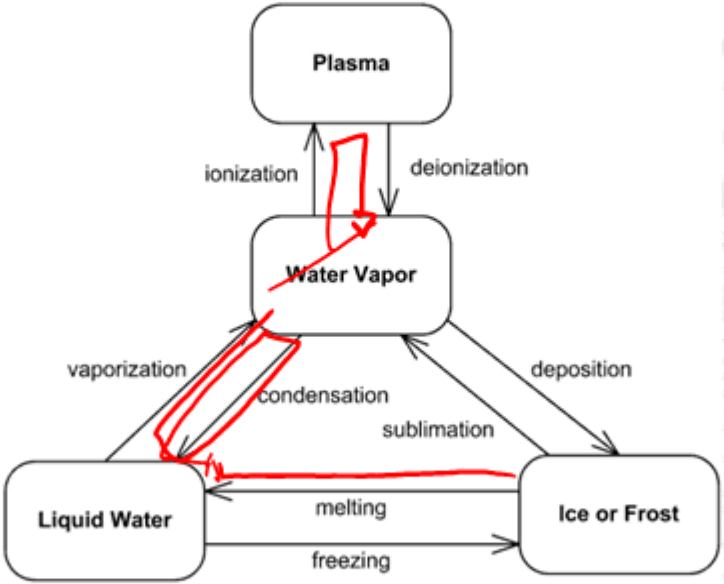
- Model of the physical world:
  - Used to abstract (simplify) a complex system
    - Example: Button is up, button is down

Button is burning

- Software Design Model
  - Used to plan, document, and build code
    - Or circuits, or machines
    - Example: Button event checker from last lecture
  - Easy to observe and debug (SO USEFUL)

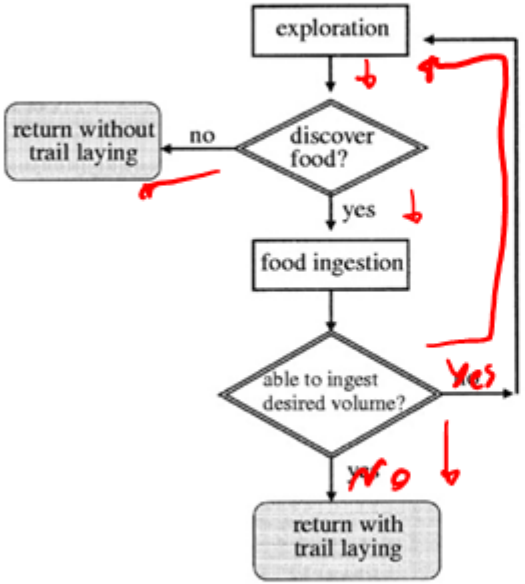
Max Lichtenstein

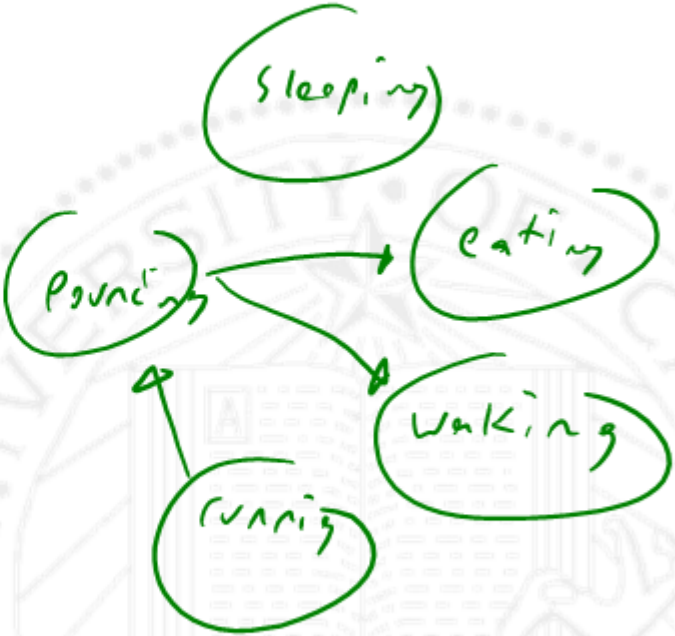
Physical Models



https://www.uml-diagrams.org/examples

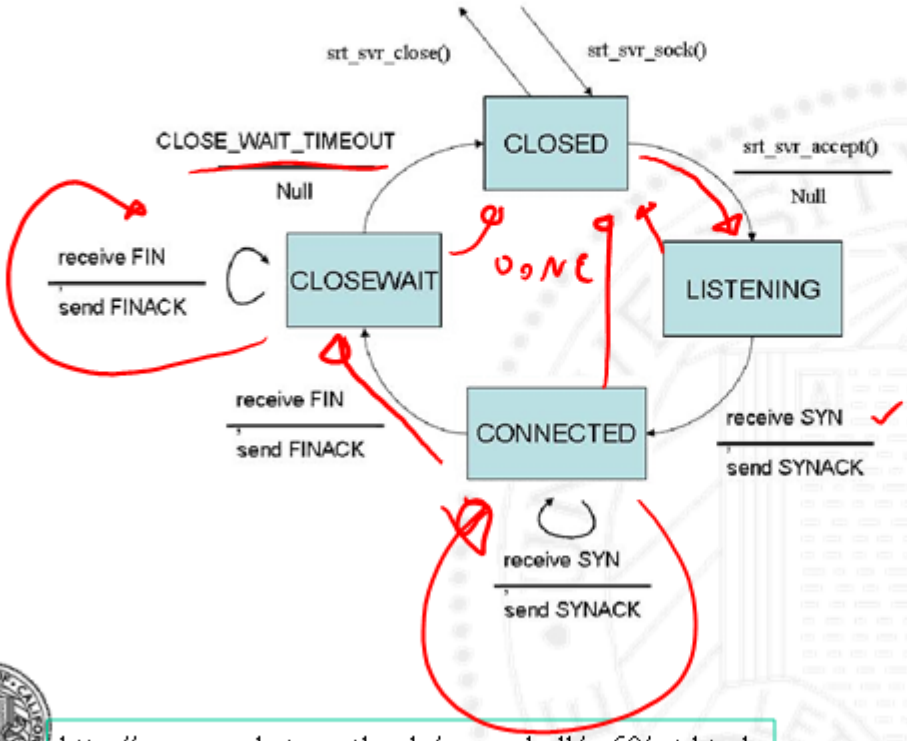
Behavioral Models





The principles of collective animal behaviour
D.J.T Sumpter Phil. Trans. R. Soc. B 29 January 2006

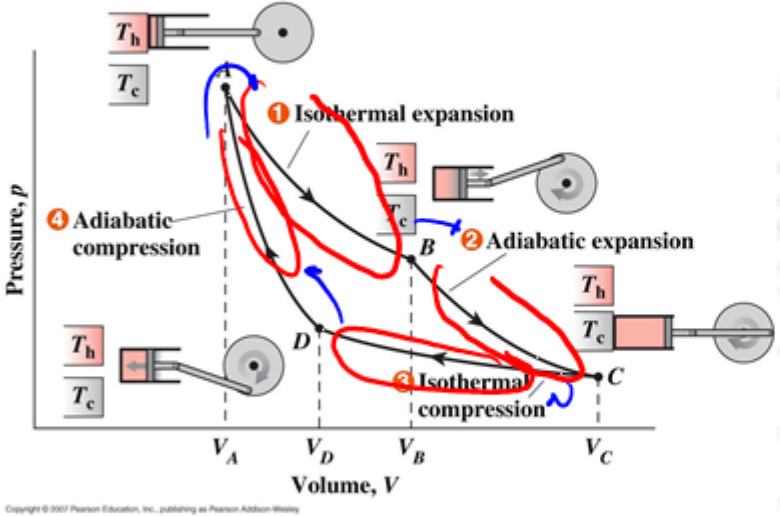
Server Design Models

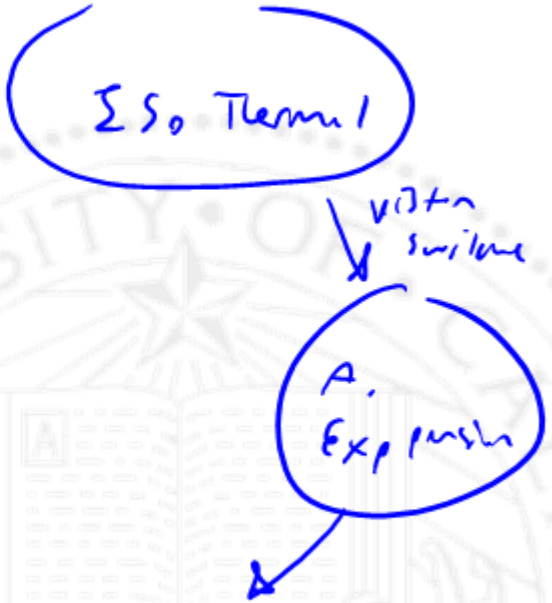


TCP protocol

http://www.cs.dartmouth.edu/~campbell/cs60/srt.html

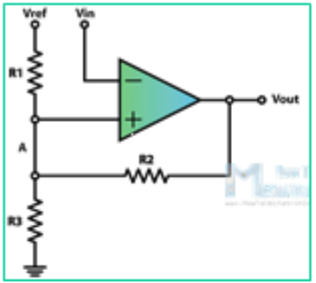
Physical Design Models

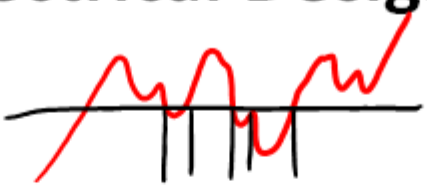


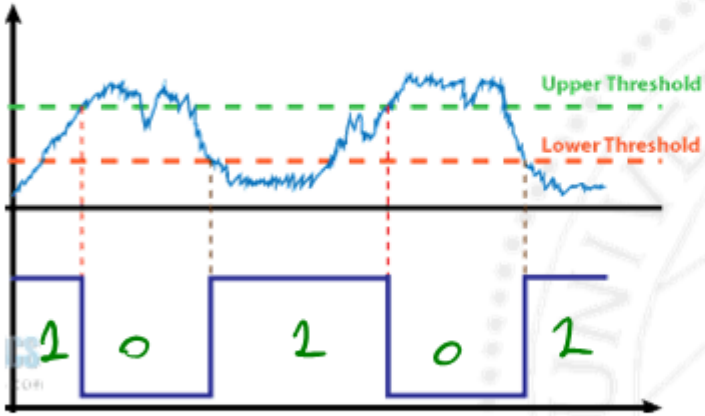


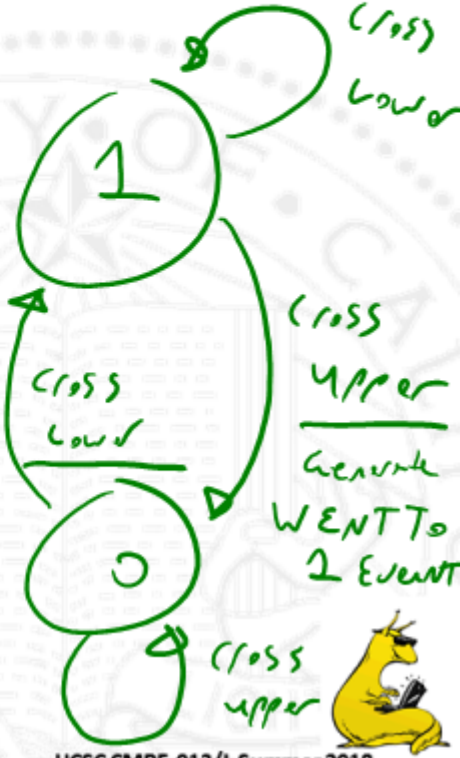
Max Lichtenstein

Electrical Design Models

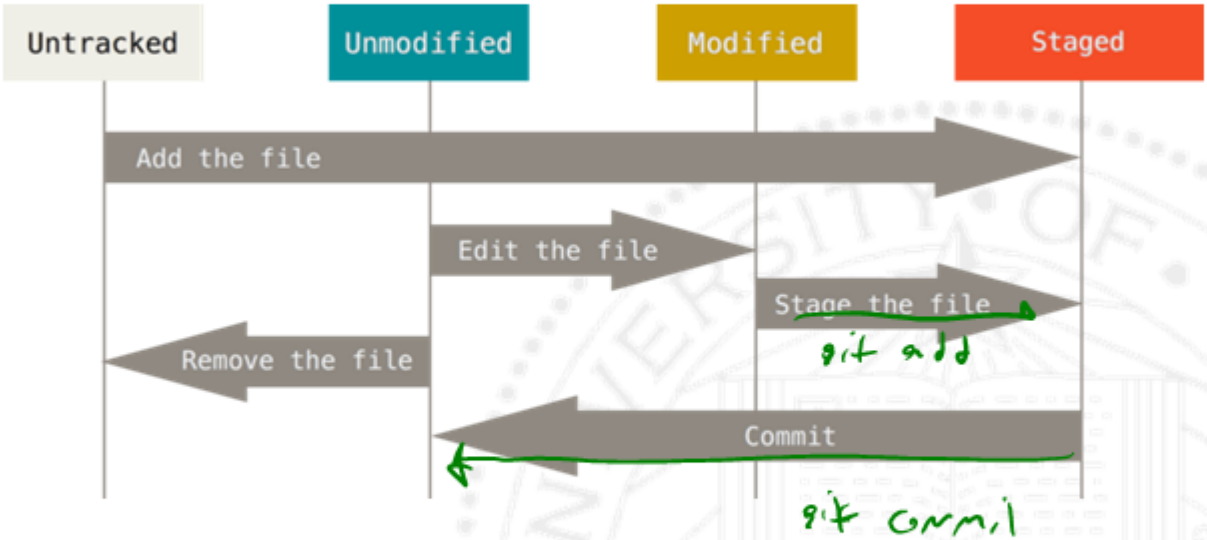






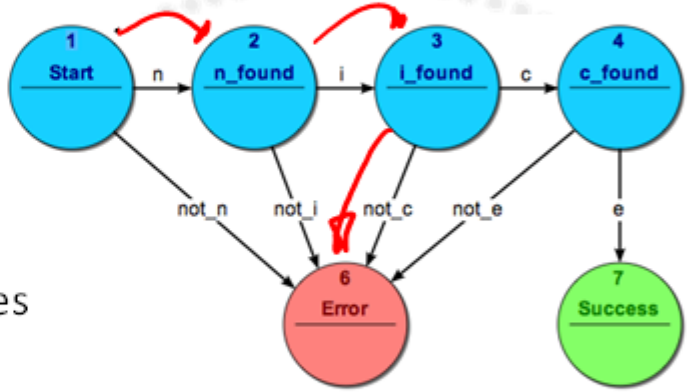


Git's file status model



State Machines as Classifiers/Parsers

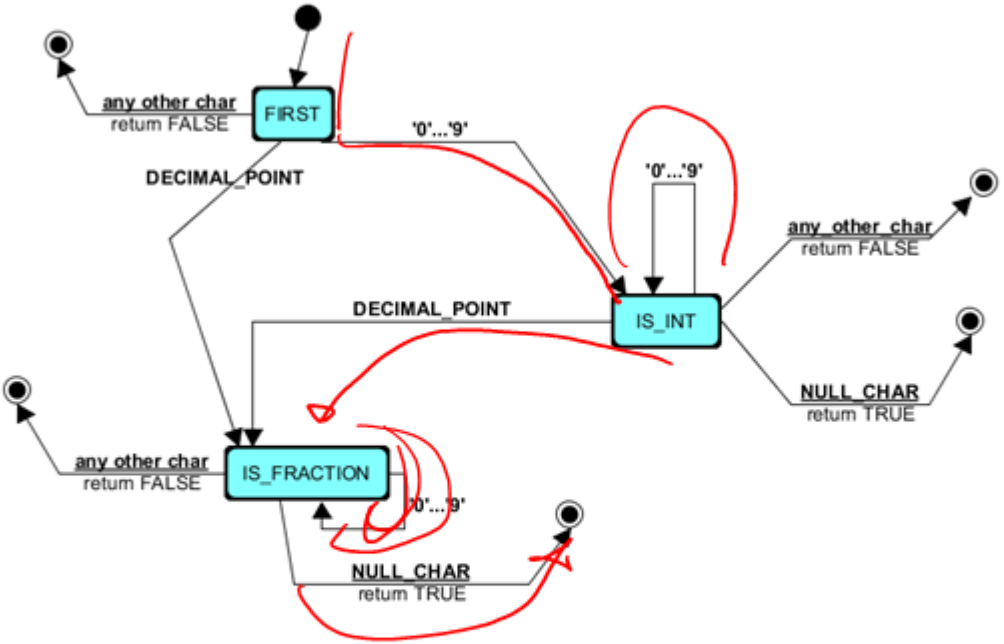
- State machines can also be used for some kinds of parsing problems
  - Regexes
  - MIPS assembler
  - atoi(), atof() ✓
  - Command line parser ✓
  - Reading transmitted messages
    - IP datagrams
    - Morse code sequences
    - NEMA messages



is the string "nice"

strcmp(input, "nice")

Piazza Poll: `int IsNumber(char * token)`



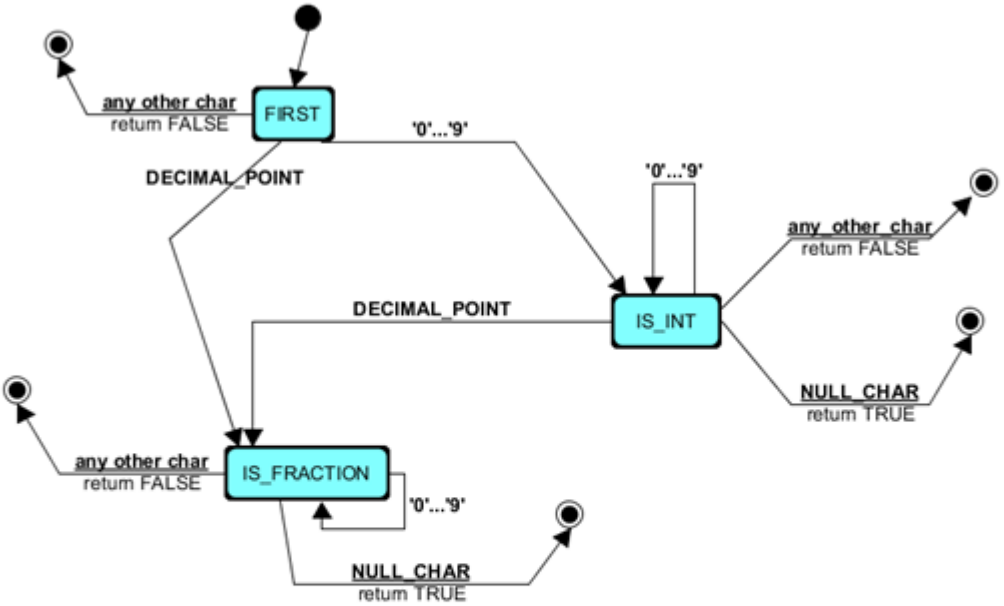
This state machine operates over the chars in a string to determine if the string represents a number.

Which token(s) will return "FALSE"?

A: `"003.14000"`
B: `"192.168.1.1"`
C: `"-.000"`
D: `"12.5%"`

BREAK

Piazza Poll: `int IsNumber(char * token)`

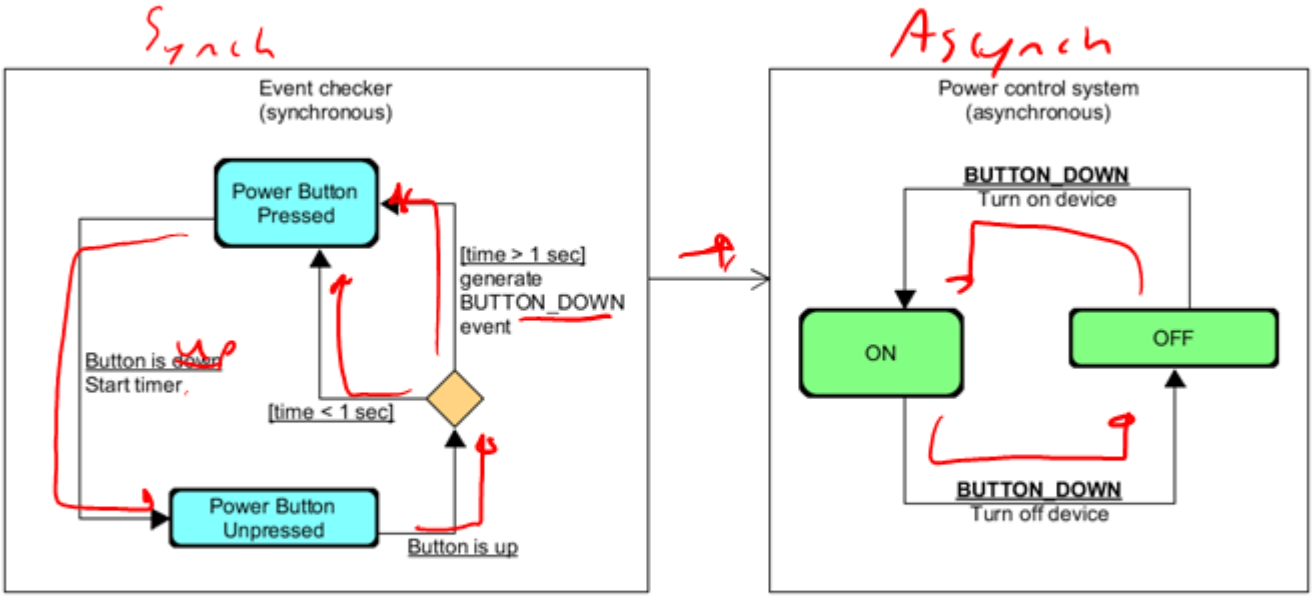


This state machine operates over the chars in a string to determine if the string represents a number.

Which token(s) will return "FALSE"?

A: `"003.14000"`
B: `"192.168.1.1"`
C: `"-.000"`
D: `"12.5%"`

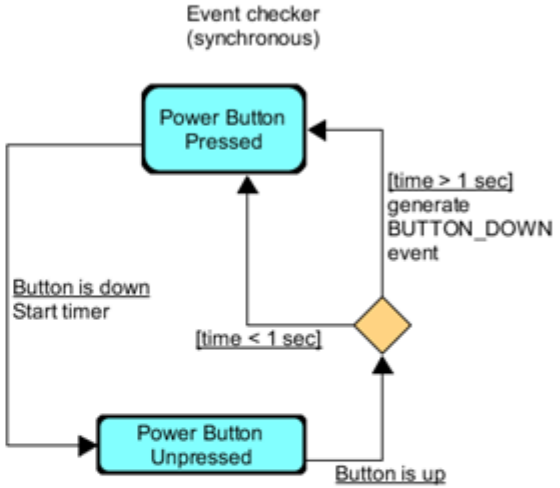
Synchronous vs Asynchronous SMs



State machines are useful to both *generate* **and** *react to* events!

Synchronous SMs:

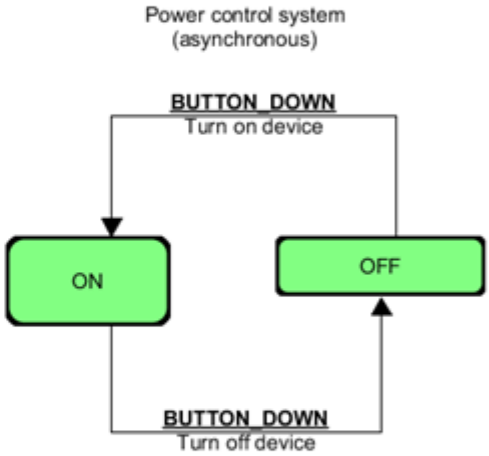
- All event checkers are synchronous SMs
  - Event checkers need memory!
- Run periodically
  - In a timer ISR ✓
  - In your main loop ✓
  - Transitions are NOT events
  - Instead, they are conditions
    - (observations of some other system's state)

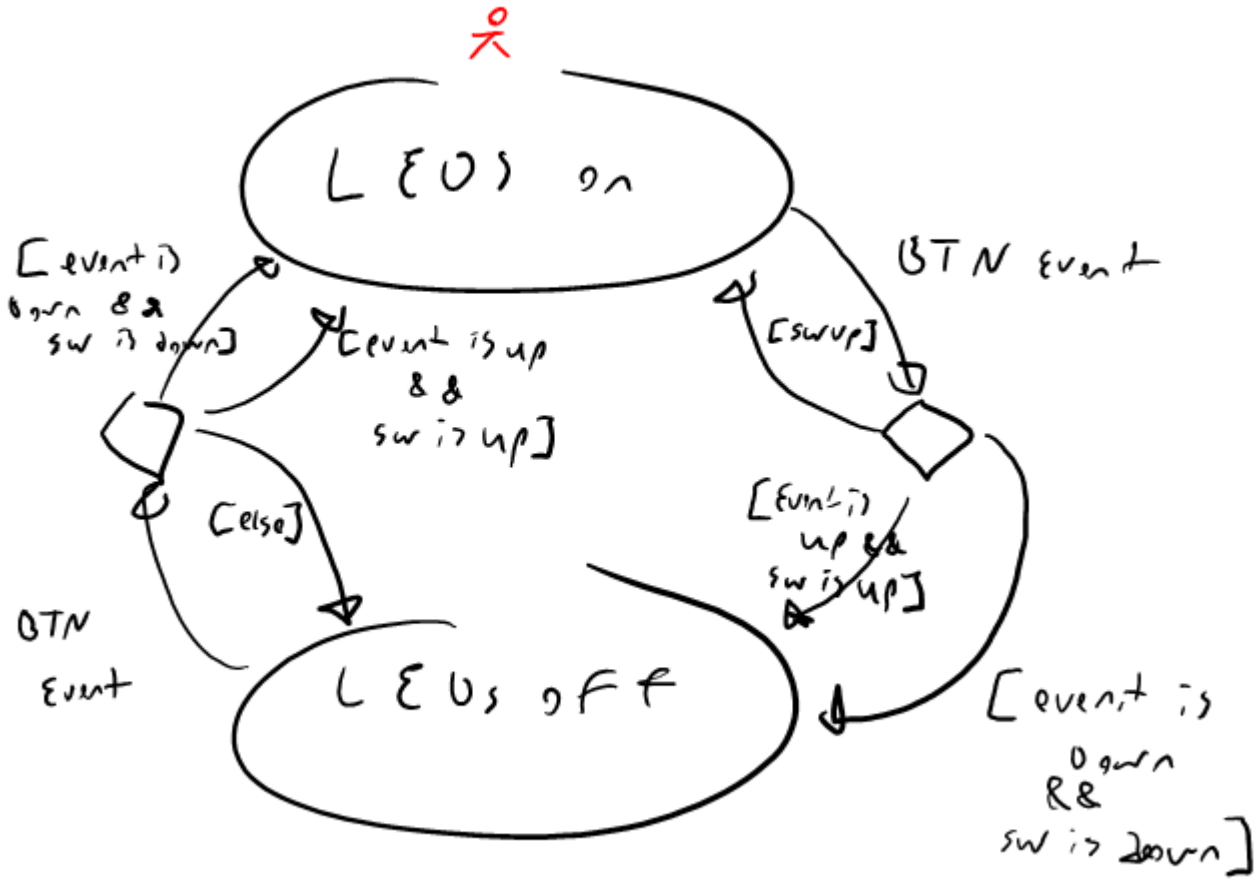


Max Lichtenstein

Asynchronous SMs:

- One (good) way to implement a services
- React to events
  - Consume events




LED on
[event is down && sw is down]
[event is up && sw is up]
BTN event
[sw up]
[else]
[event is up && sw is up]
BTN event
LED off
[event is down && sw is down]

```
event = Button_Clock_Events(0x00);
if (event == btn_event) { ...
event = Button_Clock_Event(0x01);
if (event == BUTTON1_DOWN);
```