# Smart Sprinkler Controller with Nerves

Todd Resudek

# Todd Resudek


Senior Software Engineer at Weedmaps

Hex core team

supersimple

@sprsmpl

# What is this project about?

# Properties of a sprinkler controller

1. Turns sprinklers on and off
2. Follows a preset schedule
3. Independently controls multiple zones
4. Customizable?

# Off the Shelf Options

Rachio 3

8 Zones

$229

# Why Not Buy One?

1. Cost difference

2. Fun (you enjoy building)

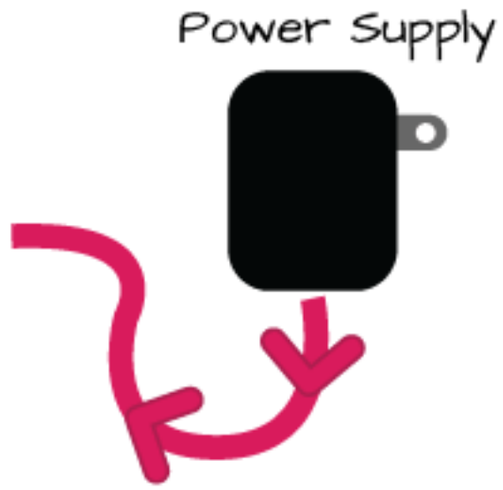
3. Learn something new

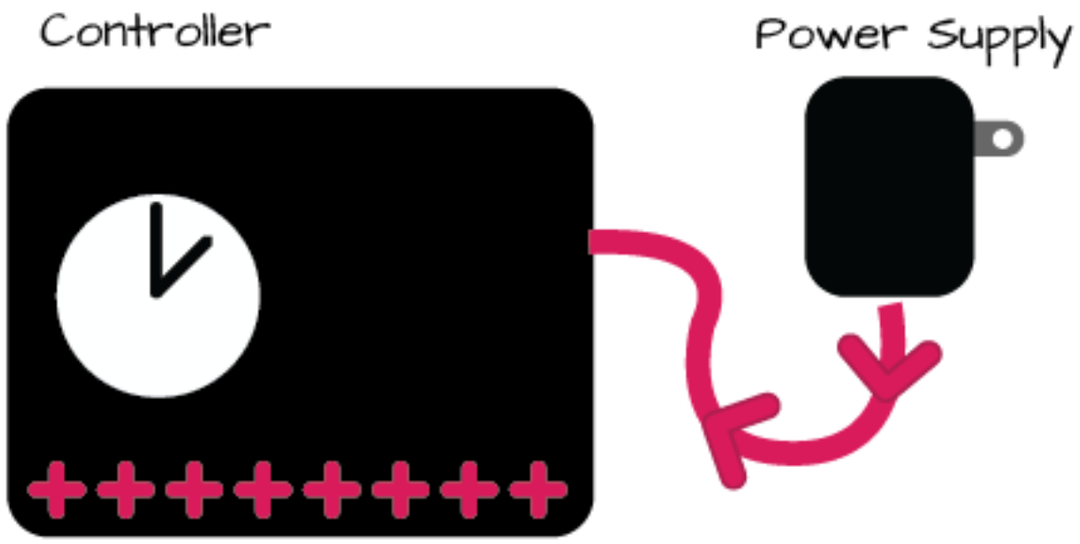
# My history with hardware

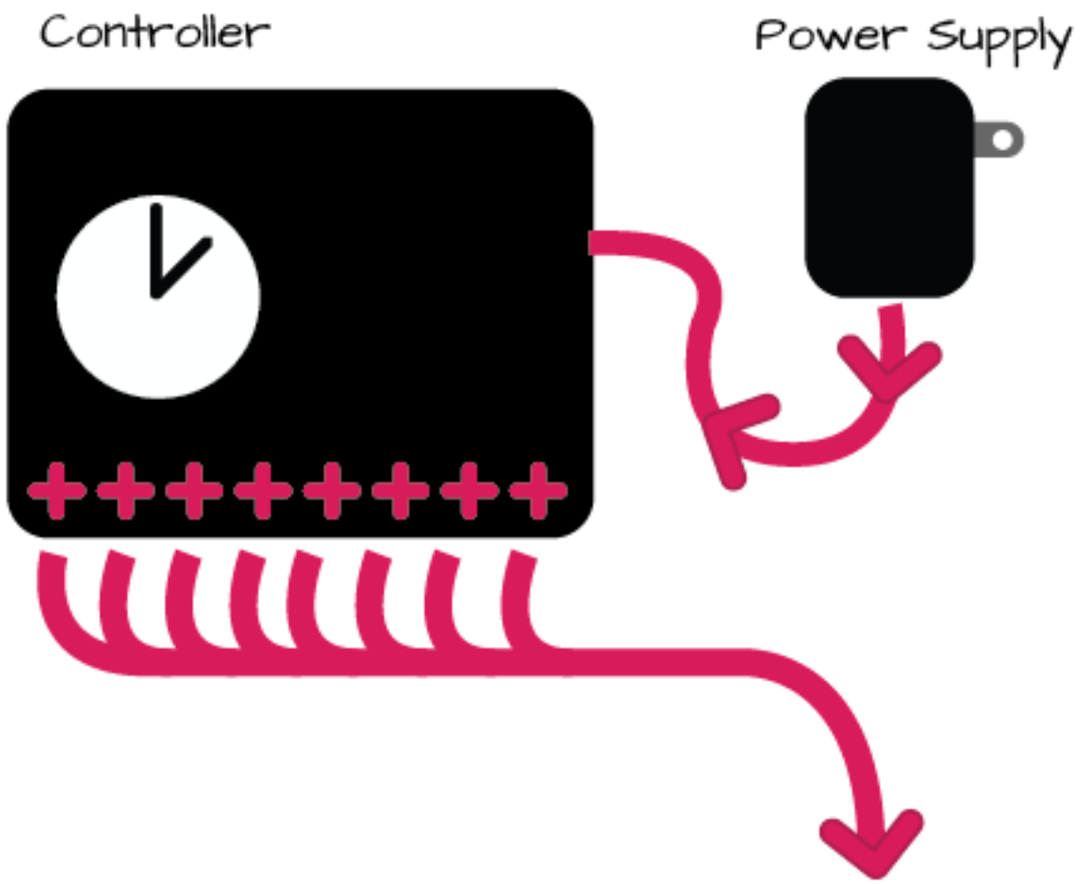
# Stall Monitor

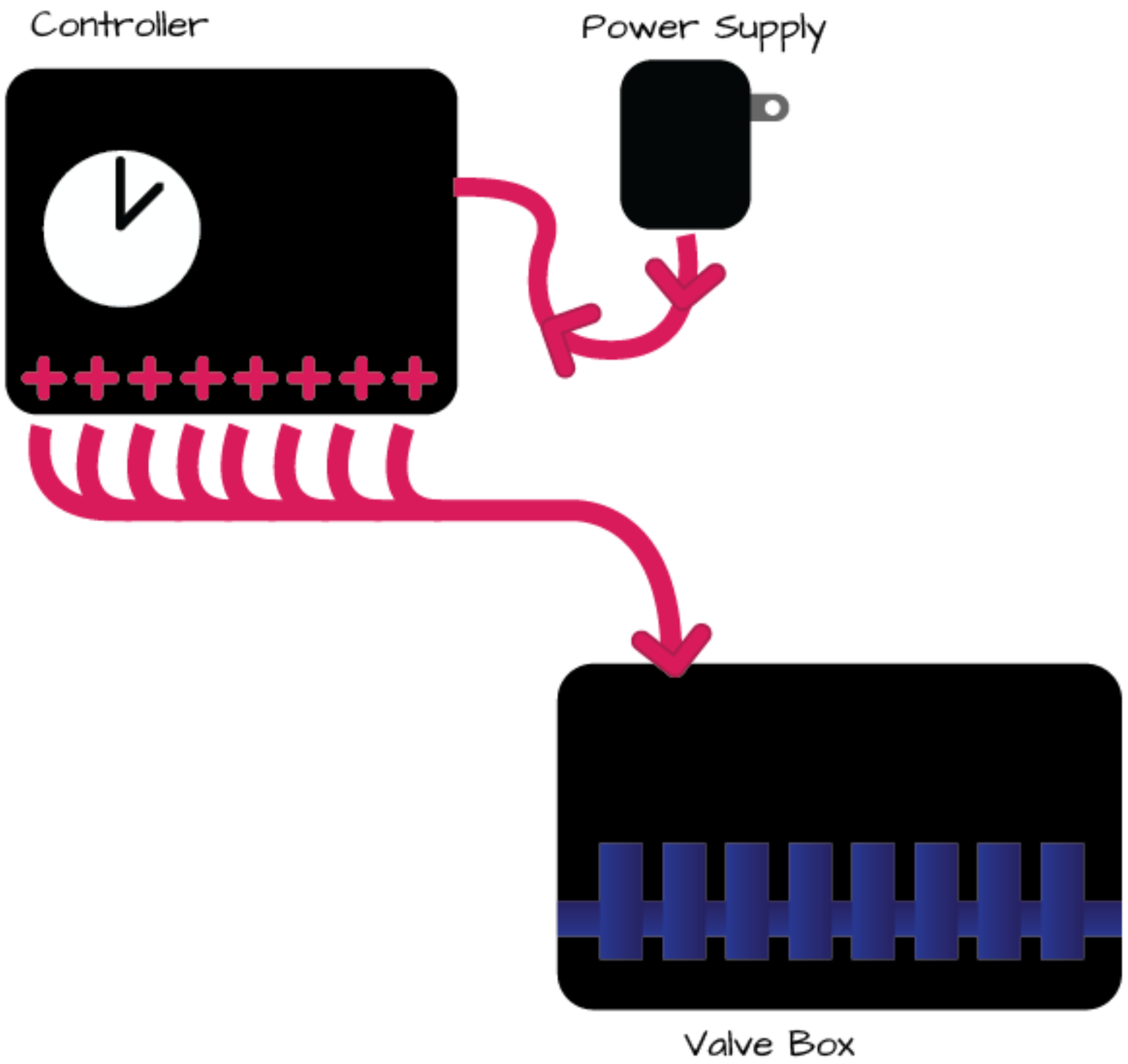
# Parkr

# How do sprinklers work?

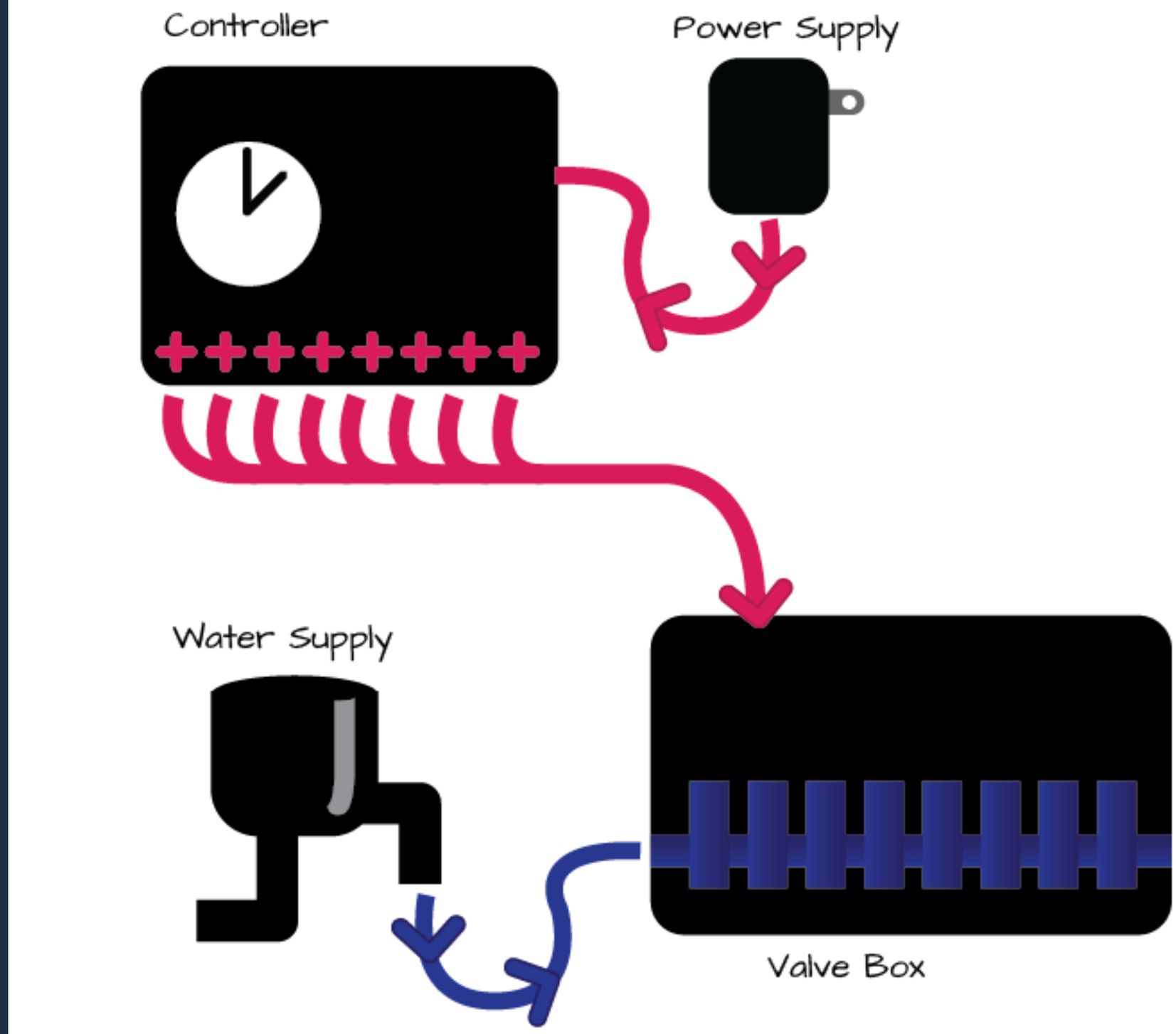

Power Supply

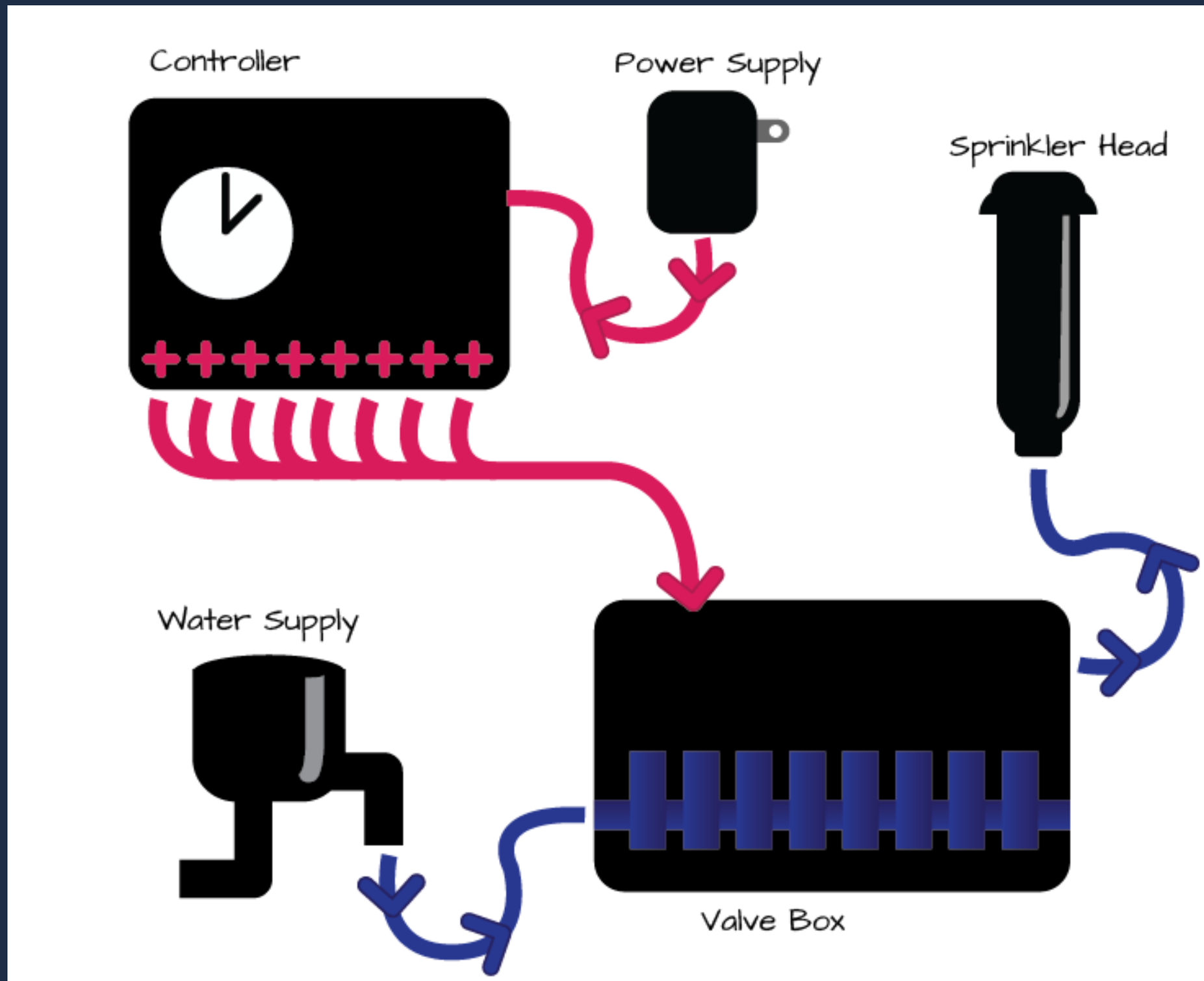

Controller
Power Supply


Controller
Power Supply

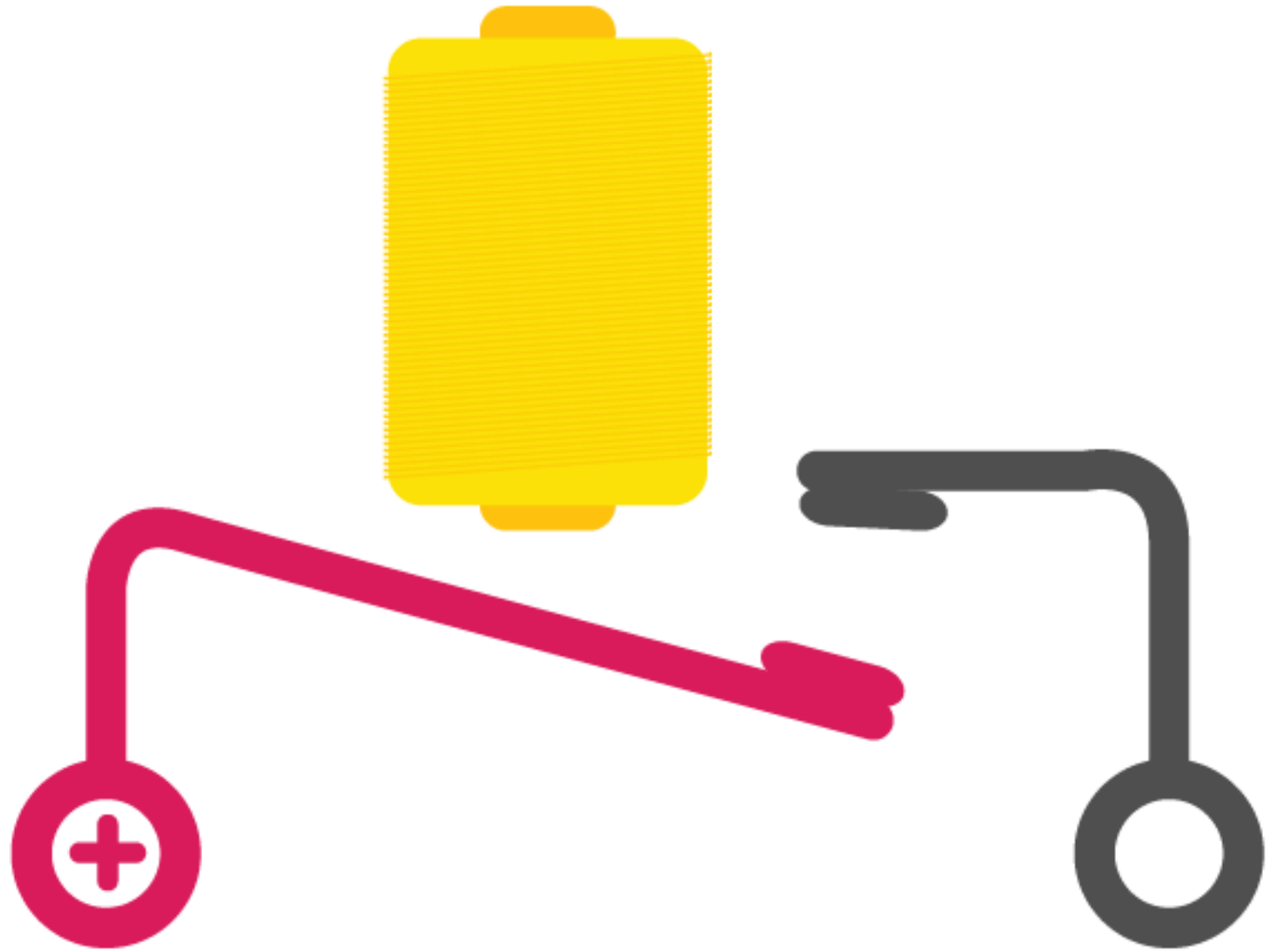

Controller
Power Supply
Valve Box

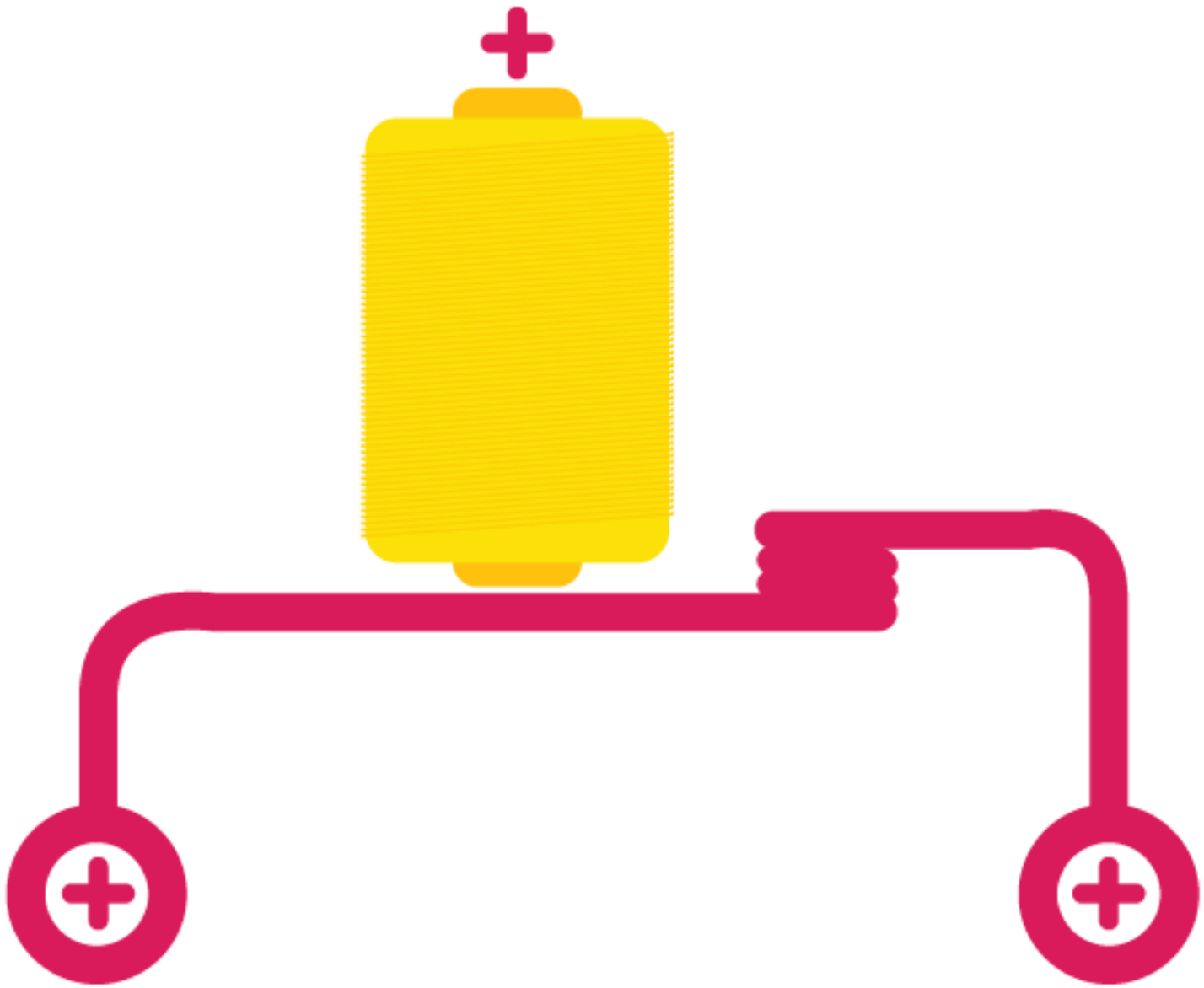
# Valve Box


Controller
Power Supply
Water Supply
Valve Box

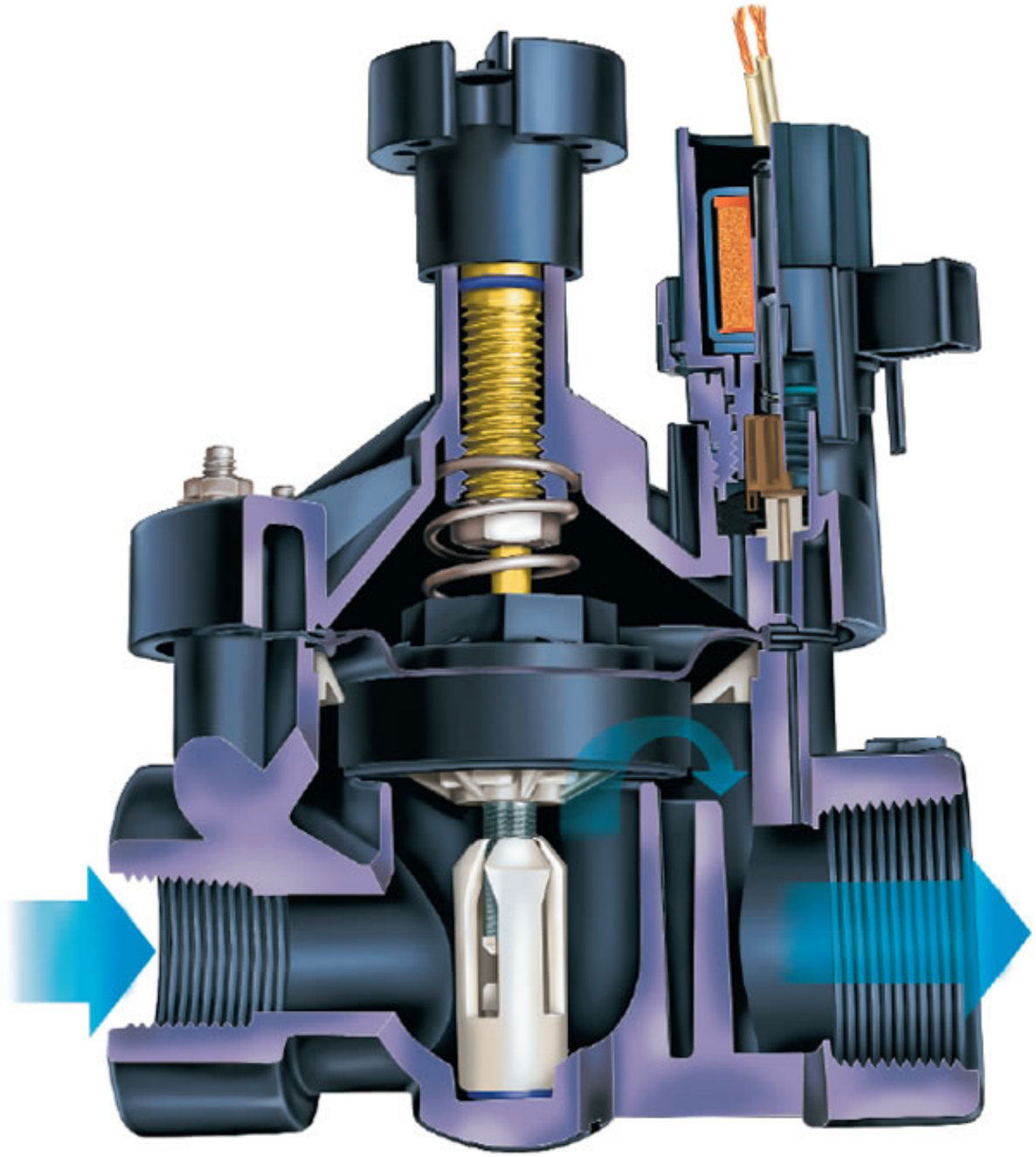

Controller
Power Supply
Sprinkler Head
Water Supply
Valve Box

# How relays work





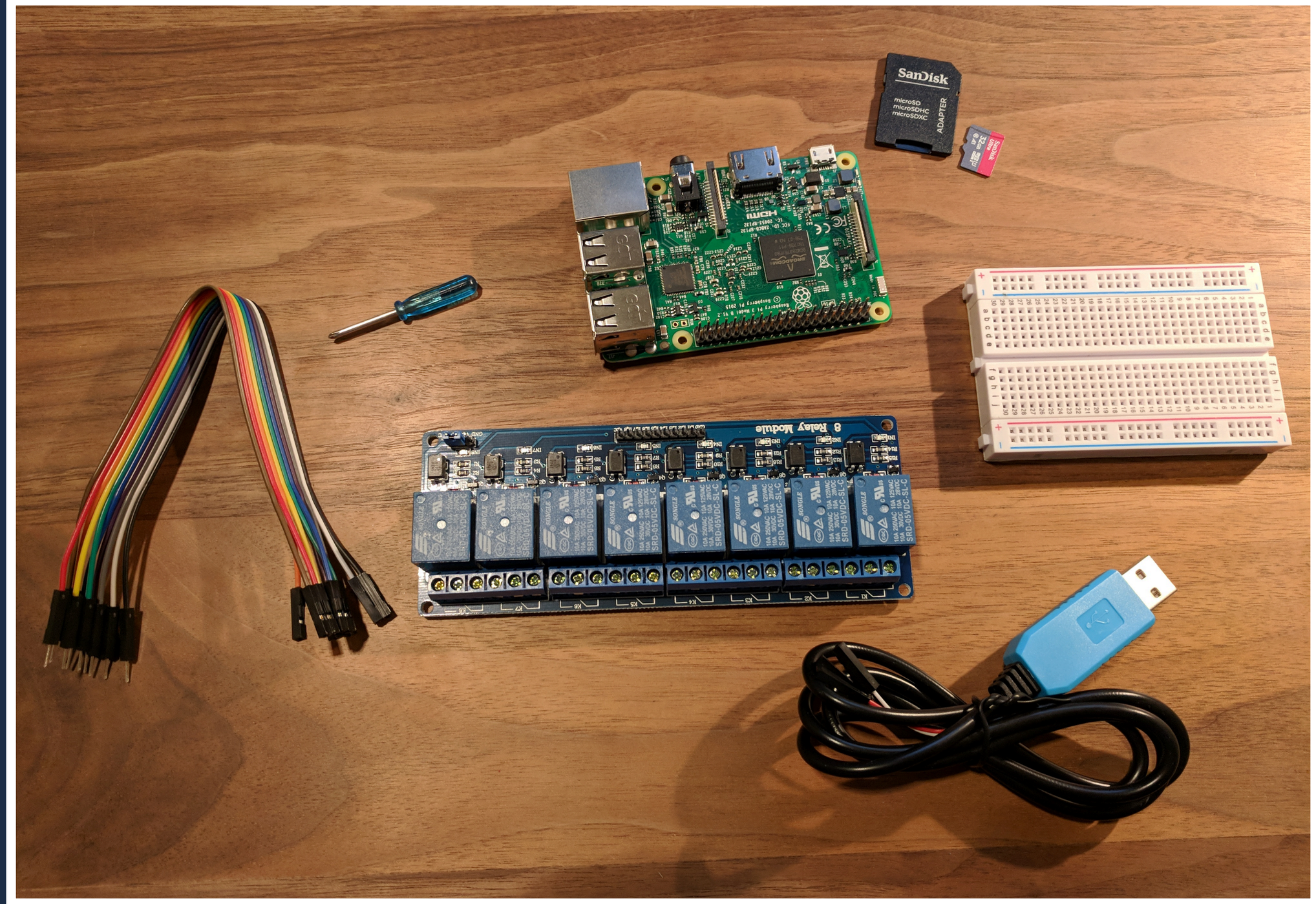
# How sprinkler valves work



*PESB Cutaway*

# Let's start building

# Parts Required



# Parts Required

1. Raspberry Pi ($10-35)
2. Power Supplies ($15)
3. Relay Board ($9)
4. Memory Card ($6)
5. Wires ($3)

# Hardware design

# Mitch Hedberg

*An escalator can never break: it can only become stairs.*

# User Interface

# User Interface

# Intro to Nerves

# Introduction

*Nerves defines an entirely new way to build embedded systems using Elixir. It is specifically designed for embedded systems, not desktop or server systems.*

# 3 Parts

1. **Platform** - a customized, minimal Buildroot-derived Linux that boots directly to the BEAM VM.

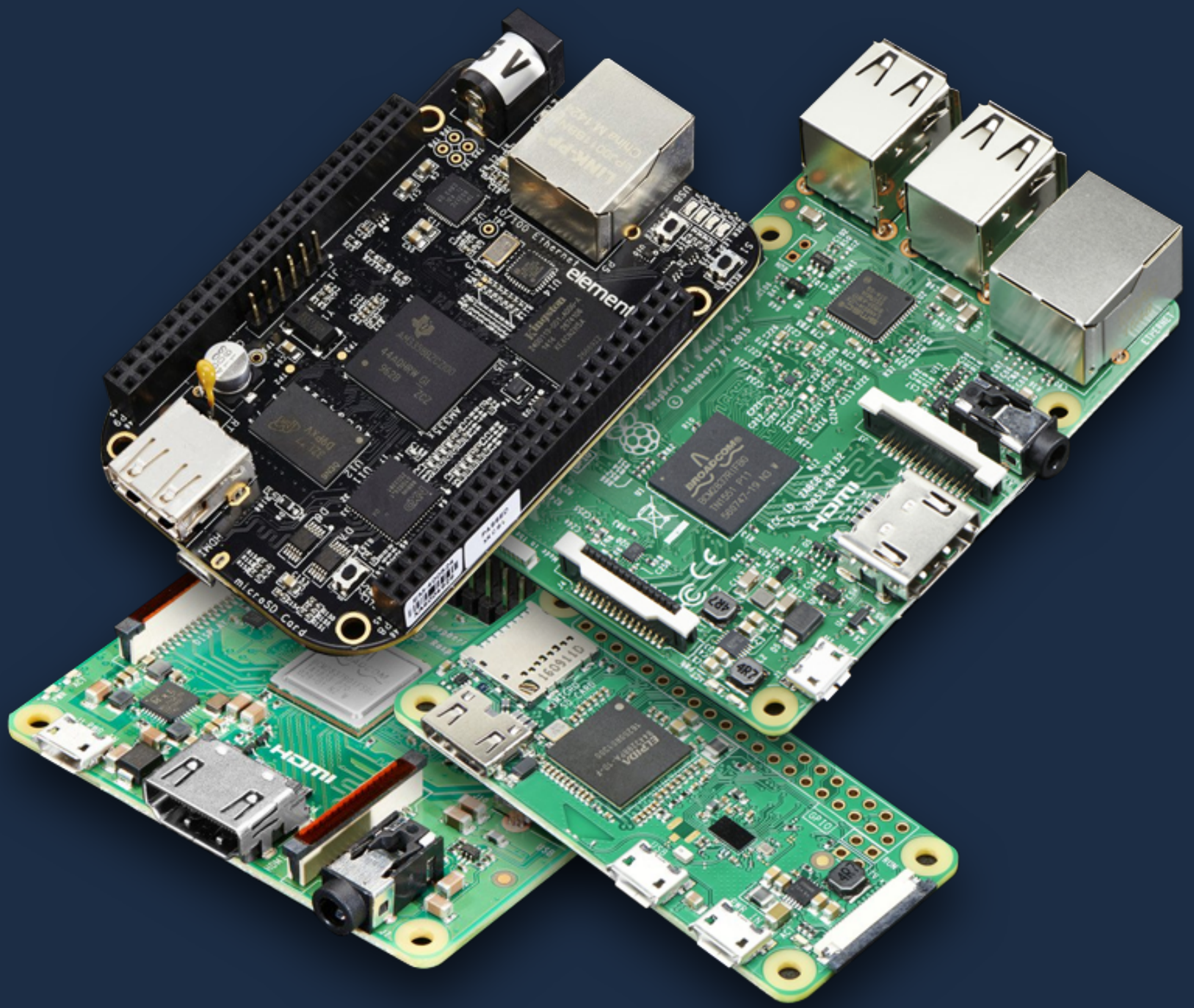
2. **Framework** - ready-to-go library of Elixir modules to get you up and running quickly

3. **Tooling** - powerful command-line tools to manage builds, update firmware, configure devices, and more.

# Put them together

*Taken together, the Nerves platform, framework, and tooling provide a highly specialized environment for using Elixir to build advanced embedded devices.*

# Compatible Hardware



# Why Elixir?

1. Easier to write

2. More features (OTP)

3. Libraries

# Installation

*Nerves requires a number of programs on your system to work. These include Erlang, Elixir, and a few tools for packaging firmware images. Nerves is actively used on MacOS and various Linux distributions.*

# Installation

```
brew update
brew install fwup squashfs
coreutils xz
```

# Installation

```
asdf install erlang 21.2.2
asdf install elixir 1.8.0-otp-21
asdf global erlang 21.2.2
asdf global elixir 1.8.0-otp-21
mix local.hex
mix local.rebar
```

# Installation

```
mix archive.install hex nerves_bootstrap
```

# Let’s do this.

# Step 1. Zones

```
config :drizzle,
zone_pins: %{
    zone1: 7,
    zone2: 8,
    zone3: 25,
    zone4: 24,
    zone5: 23,
    zone6: 18,
    zone7: 15,
    zone8: 14
}
```

# Step 2. Durations

```
config :drizzle,
schedule: %{
    sun: [{:zone1, 20}],
    mon: [{:zone2, 10}],
    tue: [{:zone3, 10}],
    wed: [{:zone1, 20}],
    thu: [{:zone2, 10}],
    fri: [{:zone3, 10}],
    sat: [{:zone1, 15}]
}
```

# Step 3. Run

```
def children(_target) do
    [
      {Drizzle.Init, []},
      {Drizzle.Scheduler, %{}},
      {Drizzle.TodaysEvents, []}
    ]
```

# Drizzle.Init

```
@zone_pins Application.get_env(:drizzle, :zone_pins, %{})

def init(_state) do
    state =
      Enum.map(@zone_pins, fn {name, pin} ->
        register_pin(name, pin)
        Drizzle.IO.deactivate_zone(name)
      end)

    {:ok, state}
end
```

# Drizzle.Init (cont)

```
defp register_pin(_name, pin) do
    {:ok, output_pid} =
       @gpio_module.open(pin, :output)
    output_pid
end
```

# Drizzle.Scheduler

```
@schedule
  Application.get_env(:drizzle, :schedule, %{})

@days_as_atoms
  {:zero, :sun, :mon, :tue, :wed, :thu, :fri, :sat}

@utc_offset
  Application.get_env(:drizzle, :utc_offset, 0)
```

# Drizzle.Scheduler (cont)

```
def init(state) do
    schedule_work()
    {:ok, state}
end
```

```
defp schedule_work(), do:
    Process.send_after(self(), :work, 60 * 1000)

def handle_info(:work, state) do
   execute_scheduled_events()
   schedule_work()
   {:noreply, state}
end
```

# Drizzle.Scheduler (cont)

```
defp execute_scheduled_events do
    if current_time() == 0 || TodaysEvents.current_state() == [] do
      TodaysEvents.reset()
      TodaysEvents.update(Map.get(@schedule, current_day_of_week()))
    end

    case Enum.find(TodaysEvents.current_state(), fn {time, _a, _z} ->
           time == current_time()
         end) do
      {_time, :on, zone} -> Drizzle.IO.activate_zone(zone)
      {_time, :off, zone} -> Drizzle.IO.deactivate_zone(zone)
      _ -> "Nothing to do right now."
    end
  end
```

# Drizzle.IO

```
def activate_zone(zone), do:
    @gpio_module.write(zone, 0)

def deactivate_zone(zone), do:
    @gpio_module.write(zone, 1)
```

# Drizzle.TodaysEvents

```
def update(today) do
    Agent.update(__MODULE__, fn _state ->
      calculate_start_stop_time(today)
    end)
end
```

```
defp calculate_start_stop_time(today) do
    today
    |> Enum.map(fn {key, list} -> reduce_event_groups(key, list) end)
    |> Enum.reduce([], fn m, acc -> acc ++ m[:events] end)
end
```

# Drizzle.TodaysEvents

```
defp reduce_event_groups(key, list) do
    start_time = 0
    Enum.reduce(list,
                %{last_time: start_time, events: []},
                fn {zone, duration}, acc ->
      new_start_event = {acc[:last_time], :on, zone}
      acc = update_in(acc[:last_time], &(&1 + duration))
      new_stop_event = {acc[:last_time], :off, zone}
      acc = update_in(acc[:last_time], &(&1 + 1))
      update_in(acc[:events], &(&1 ++ [new_start_event, new_stop_event]))
    end)
end
```

# Make it smart(er)

# Step 1. Watering Times

```
config :drizzle,
available_watering_times: %{
    morning: {500, 800},
    afternoon: {1430, 1600}
    evening: {2100, 2300}
}
```

# Step 2. Durations

```
config :drizzle,
schedule: %{
   sun: [
    {:zone1, :morning, 20},
    {:zone2, :afternoon, 10},
    {:zone1, :evening, 20}
  ], …
}
```

# Step 3. Winter Months

```
config :drizzle,
winter_months:
  [:jan, :feb, :nov, :dec]
```

# Step 4. Run

```
def children(_target) do
    [
      {Drizzle.WeatherData, []},
      {Drizzle.Init, []},
      {Drizzle.Scheduler, %{}},
      {Drizzle.Forecaster, %{}},
      {Drizzle.TodaysEvents, []}
    ]
```

# Step 5. WeatherData

```
def start_link() do
  init = for _n <- 1..12, do: nil
  Agent.start_link(fn -> init end, name: __MODULE__)
end
```

```
def update(next_24_hours) do
  Agent.update(__MODULE__, fn state ->
    Enum.slice(state, 1..12) ++ next_24_hours
  end)
end
```

# Step 6. Forecaster

```
config :drizzle,
location:
  %{
    latitude: 39.3898838,
    longitude: -104.8287546
  }
```

# Step 6. Forecaster

```
defp deps do
  [
    {:darkskyx, "~> 0.1.4”}
    . . .
  ]
end
```

# Step 6. Forecaster

```
def init(state) do
  schedule_work()
  {:ok, state}
end

def handle_info(:work, state) do
  Weather.get_todays_forecast()
  schedule_work()
  {:noreply, state}
end

defp schedule_work(), do:
  Process.send_after(self(), :work, 60 * 60 * 1000)
```

# Step 7. Weather

```
def weather_adjustment_factor do
    if month_as_atom(DateTime.utc_now().month) in @winter_months do
      0
    else
      {low, high, precipitation} =
        Drizzle.WeatherData.current_state()
        |> Enum.filter(&(!is_nil(&1)))
        |> weather_info()

      temperature_adjustment(low, high)
      |> Kernel.*(precipitation_adjustment(precipitation))
  end
end
```

# Step 7. Weather

```
defp weather_info(data) do
    with {cumulative_amount, cumulative_percent} <-
           Enum.reduce(data, {0, 0}, fn {_, am, pr}, {acc_a, acc_b} ->
             {acc_a + am, acc_b + pr}
           end),
         {low, high} <- Enum.min_max_by(data, fn {temp, _, _} -> temp end),
         rainfall <- cumulative_amount * cumulative_percent do
      {low_temp, _, _} = low
      {high_temp, _, _} = high
      {low_temp, high_temp, rainfall}
    end
end
```

# Step 7. Weather

```
@low_temp 40
@high_temp 90

defp temperature_adjustment(low, _high)
  when low ≤ @low_temp, do: 0

defp temperature_adjustment(_low, high)
  when high ≥ @high_temp, do: 1.33

defp temperature_adjustment(_low, _high), do: 1
```

# Step 7. Weather

```
defp precipitation_adjustment(prec)
  when prec ≥ 1.0, do: 0

defp precipitation_adjustment(prec)
  when prec ≥ 0.5, do: 0.5

defp precipitation_adjustment(prec)
  when prec ≥ 0.25, do: 0.75

defp precipitation_adjustment(_prec), do: 1
```

# Drizzle.TodaysEvents (updated)

```
defp reduce_event_groups(key, list) do
    factor = Drizzle.Weather.weather_adjustment_factor()
    {start_time, _stop_time} = Map.get(@available_watering_times, key)
    Enum.reduce(list,
                %{last_time: start_time, events: []},
                fn {zone, _grp, duration}, acc ->
      new_start_event = {acc[:last_time], :on, zone}
      acc = update_in(acc[:last_time], &(&1 + duration * factor))
      new_stop_event = {acc[:last_time], :off, zone}
      acc = update_in(acc[:last_time], &(&1 + 1))
      update_in(acc[:events], &(&1 ++ [new_start_event, new_stop_event]))
    end)
end
```

# Burn firmware

# Build and Burn

```
export MIX_TARGET=rpi0

mix nerves.release.init
mix deps.get
mix firmware
mix firmware.burn
```

# Build and Burn

# Setup

```
def deps do
  [
    {:shoehorn, "~> 0.4"},
    {:nerves_init_gadget, "~> 0.6"}
  ]
end
```

# Setup

```
config :nerves_firmware_ssh,
  authorized_keys: [
    File.read(
     Path.join(
      System.user_home!,
      “.ssh/id_rsa.pub"
     )
    )
  ]
```

# Setup

```
config :nerves_init_gadget,
  ifname: "usb0",
  address_method: :dhcpd,
  mdns_domain: “drizzle.local",
  node_name: nil,
  node_host: :mdns_domain
```


MMMMMM
MDNS
imgflip.com

# Setup

```
config :nerves,
  interface: :wlan0,
  ssid: <your wifi ssid>,
  psk: <passkey>,
  key_mgmt:"WPA-PSK"
```

# Build and Burn

```
export MIX_TARGET=rpi0

mix deps.get
mix firmware
mix firmware.push drizzle.local
```

# Build and Burn

# Next steps

# For the future

1. Welcome Pull Requests
2. Support more sensors
3. Touchscreen interface
4. Web/Mobile interface
5. ...
6. 💰

# Credits

# Thanks!

Michael Reis & Jon Carstens *@nervesmeetup*

Jeffrey Matthias *@idlehands*

Nerves Project contributors

weedmaps™


THANK
YOU
DIYLOL.COM