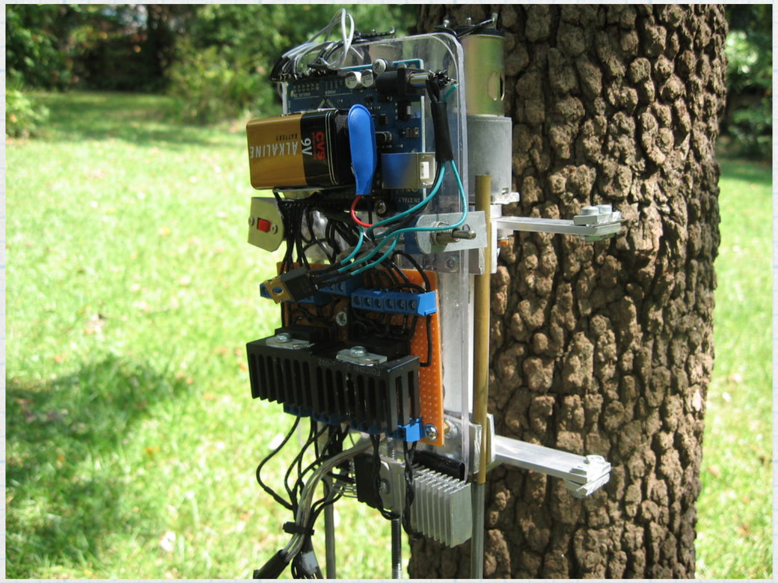
Juniper: A Functional Reactive Programming Language for the Arduino

Caleb Helbling
Tufts University

Samuel Z. Guyer
Tufts University

Workshop on Functional Art, Music, Modelling and Design (FARM)
September 2016

Project Ideas

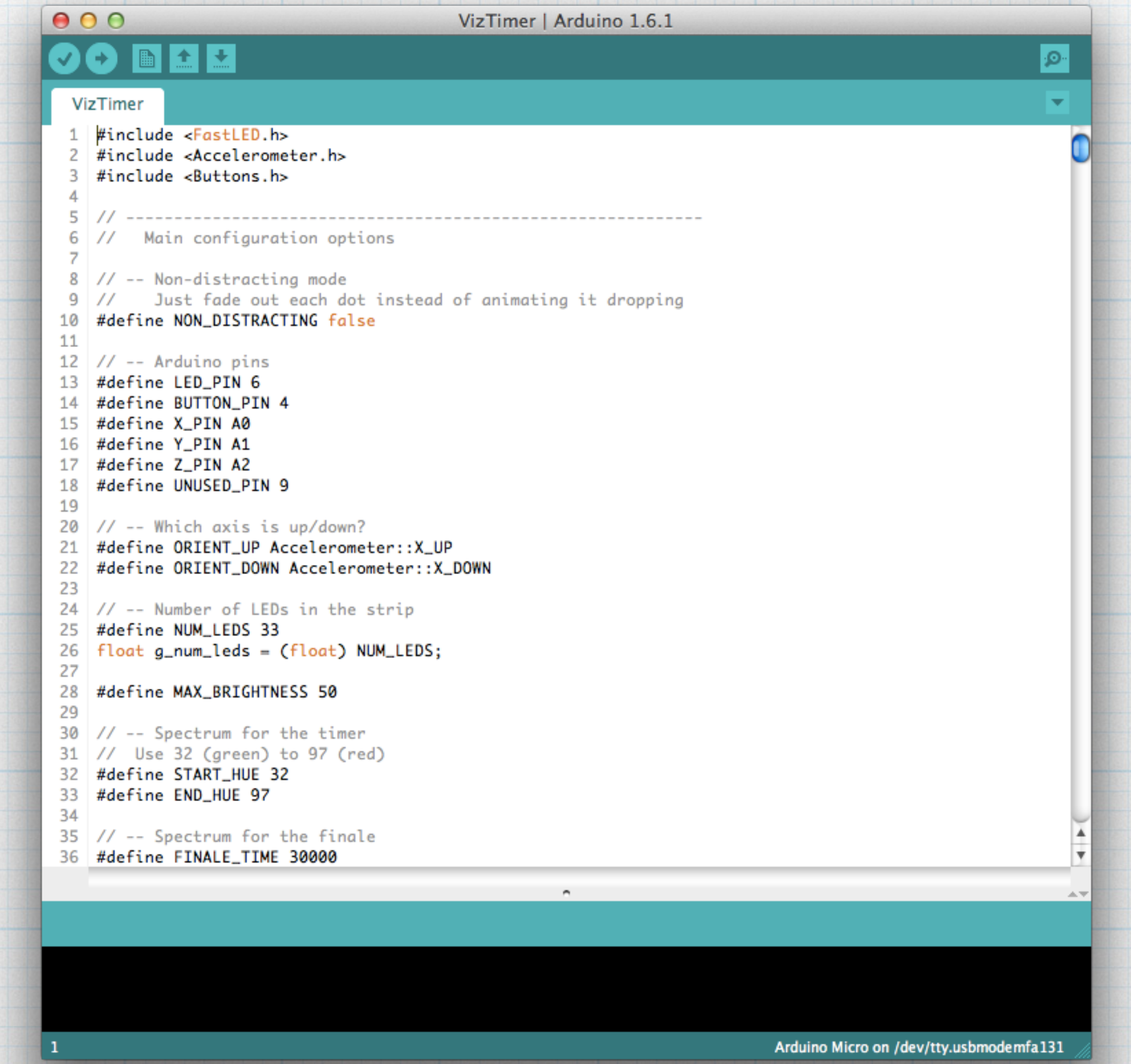


From the Arduino Web Site

"***Simple, clear*** programming environment - The Arduino programming environment is ***easy-to-use for beginners***, yet flexible enough for advanced users to take advantage of as well. For teachers, it's conveniently based on the Processing programming environment, so ***students learning to program*** in that environment will be familiar with the look and feel of Arduino"

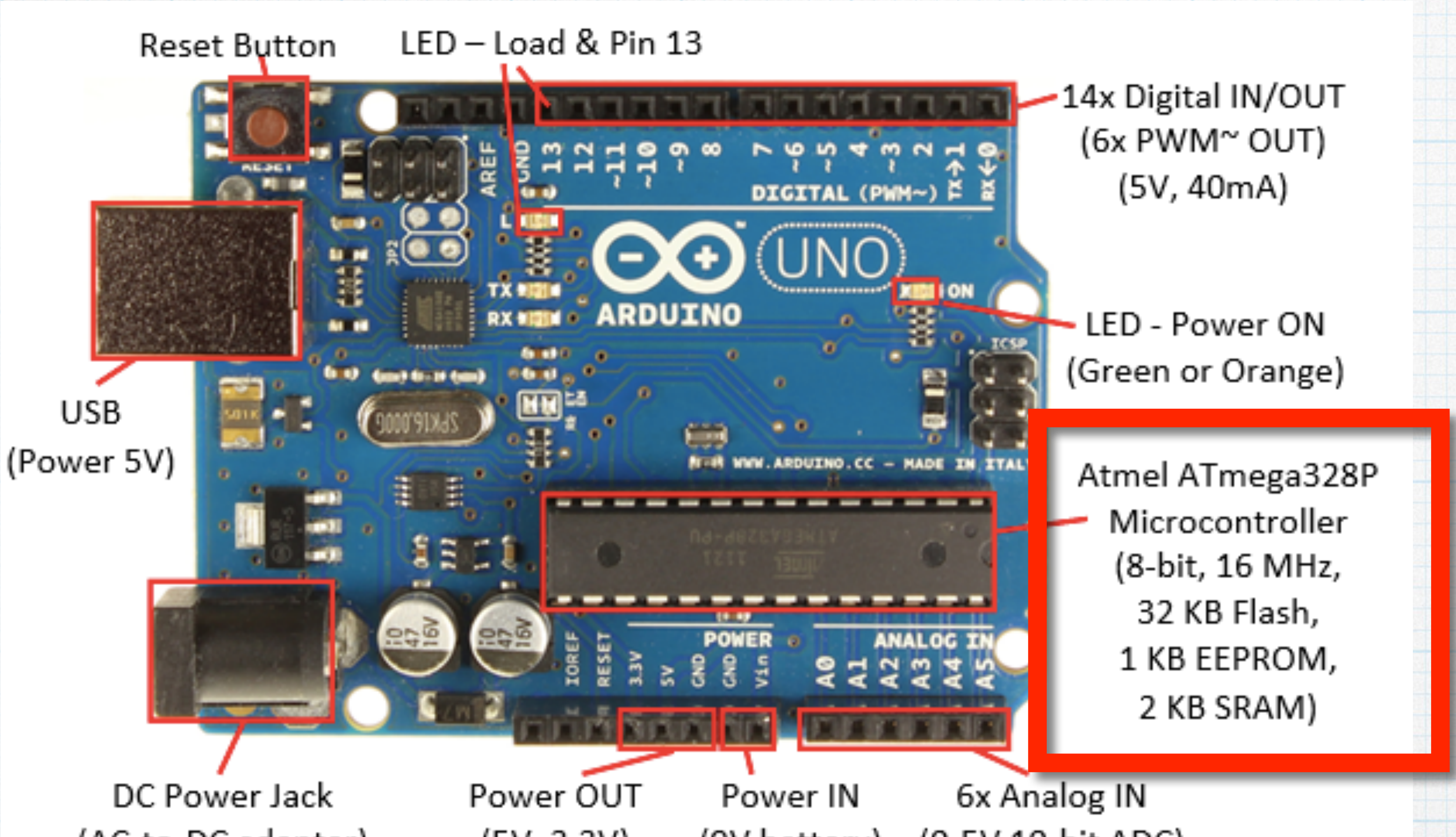
Nope

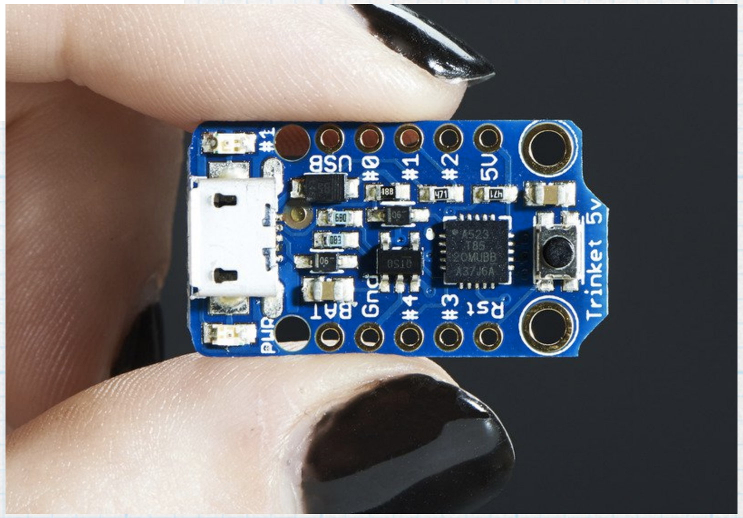
Surprise! It’s C++



```
#include <FastLED.h>
#include <Accelerometer.h>
#include <Buttons.h>

// ------------------------------------------------------------
//   Main configuration options

// -- Non-distracting mode
//    Just fade out each dot instead of animating it dropping
#define NON_DISTRACTING false

// -- Arduino pins
#define LED_PIN 6
#define BUTTON_PIN 4
#define X_PIN A0
#define Y_PIN A1
#define Z_PIN A2
#define UNUSED_PIN 9

// -- Which axis is up/down?
#define ORIENT_UP Accelerometer::X_UP
#define ORIENT_DOWN Accelerometer::X_DOWN

// -- Number of LEDs in the strip
#define NUM_LEDS 33
float g_num_leds = (float) NUM_LEDS;

#define MAX_BRIGHTNESS 50

// -- Spectrum for the timer
//  Use 32 (green) to 97 (red)
#define START_HUE 32
#define END_HUE 97

// -- Spectrum for the finale
#define FINALE_TIME 30000
```

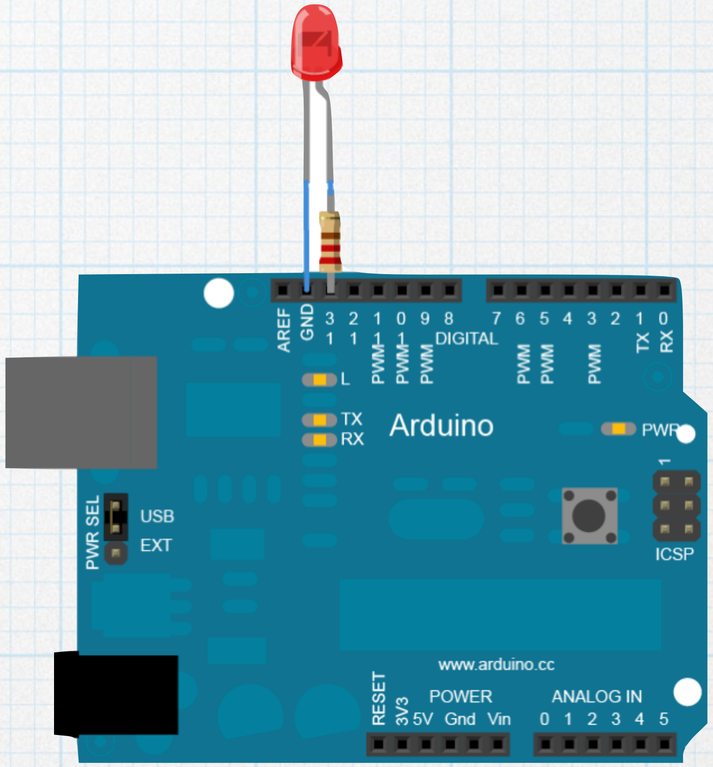
(but it kinda needs to be)


Reset Button
LED – Load & Pin 13
14x Digital IN/OUT
(6x PWM~ OUT)
(5V, 40mA)
USB
(Power 5V)
LED - Power ON
(Green or Orange)
Atmel ATmega328P
Microcontroller
(8-bit, 16 MHz,
32 KB Flash,
1 KB EEPROM,
2 KB SRAM)
DC Power Jack
(AC-to-DC adapter)
(7-12V)
Power OUT
(5V, 3.3V)
Power IN
(9V battery)
6x Analog IN
(0-5V 10-bit ADC)
UNO
ARDUINO


USB
#0
#1
#2
5V
Trinket 5V
Bat
Gnd
#4
#3
Rst

Hello, blinky world!

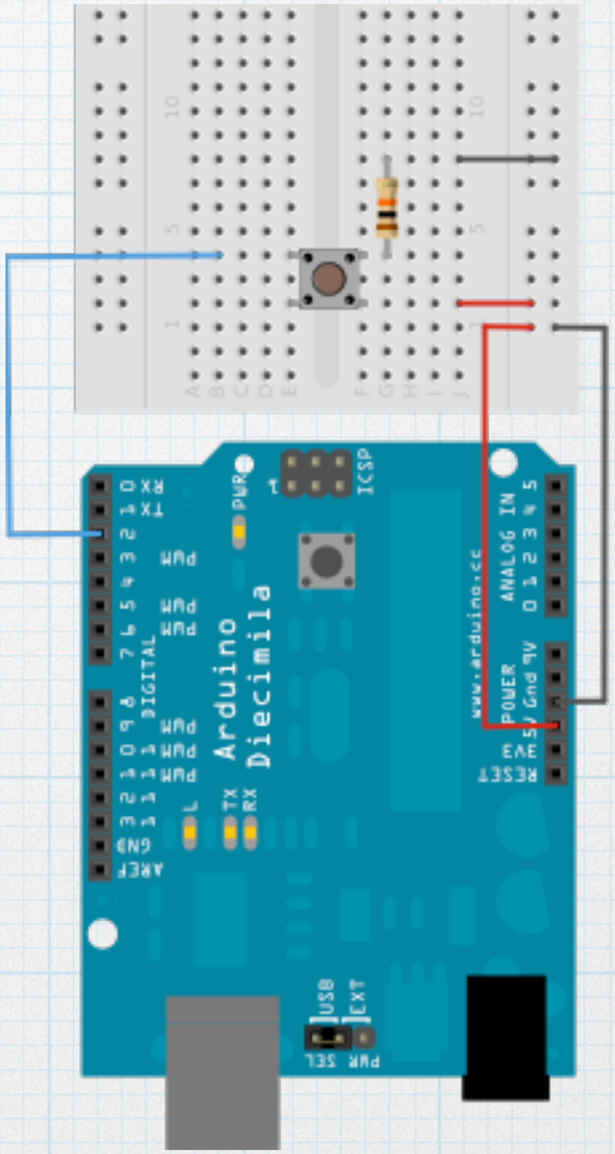
```
// -- Attach an LED to pin 13
int led = 13;


// -- The setup routine runs once
void setup() {
  // -- Initialize the pin for output
  pinMode(led, OUTPUT);
}


// -- Loop is called over and over forever:
void loop() {
  digitalWrite(led, HIGH);
  delay(1000);
  digitalWrite(led, LOW);
  delay(1000);
}
```



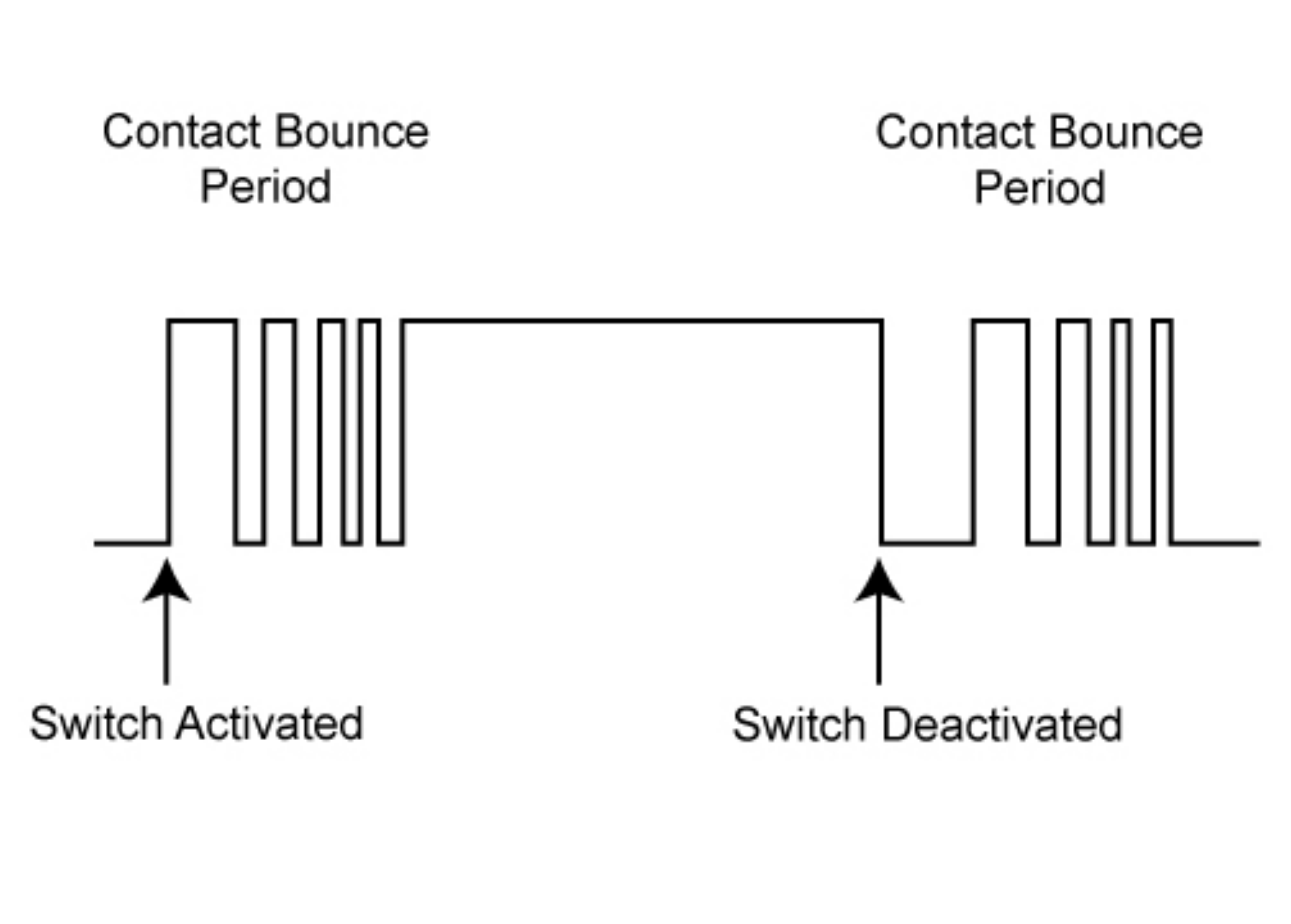
```
void blink(int pin, int interval)
{
  digitalWrite(pin, HIGH);
  delay(interval);
  digitalWrite(pin, LOW);
  delay(interval);
}
```

Add a momentary button

```
int buttonPin = 2;
int ledPin = 13;
bool ledOn = false;


void loop(){
  // -- Look for press
  if (digitalRead(buttonPin) == HIGH) {
    // -- Wait for button release
    while (digitalRead(buttonPin) != LOW) { }
    // -- Toggle LED on or off
    if ( ! ledOn) {
      digitalWrite(ledPin, HIGH);
      ledOn = true;
    } else {
      digitalWrite(ledPin, LOW);
      ledOn = false;
    }
  }
}
```



Signal bounce



```
bool isPressed(int pin)
{
  // -- Look for press
  if (digitalRead(pin) == HIGH) {
    // -- Wait 50ms
    delay(50);
    // -- Still pressed? OK, continue
    if (digitalRead(pin) == HIGH) {
      // Wait for the release
      while (digitalRead(pin) != LOW)
      return true;
    }
  }
  return false;
}
```

Debounce

Challenge: button turns blinking led on and off

```
void loop()
{
  if (isPressed(buttonPin)) {
    if ( ! ledOn) {
      digitalWrite(ledPin, HIGH);
      ledOn = true;
    } else {
      digitalWrite(ledPin, LOW);
      ledOn = false;
    }
  }
}
```

Does this work?

```
void blink(int pin, int interval)
{
  digitalWrite(pin, HIGH);
  delay(interval);
  digitalWrite(pin, LOW);
  delay(interval);
}

void loop()
{
  if (isPressed(buttonPin)) {
    if ( ! ledOn) {
      ledOn = true;
    } else {
      ledOn = false;
    }
  }
  if (ledOn) {
    blink(13, 1000);
  }
}
```

Stuck waiting for button release

Stuck here for 2 seconds!

This doesn’t work

```
void blink(int pin,
           int interval)
{
  digitalWrite(pin, HIGH);
  delay(interval);
  digitalWrite(pin, LOW);
  delay(interval);
}

void loop()
{
  blink(13, 1000);
  blink(9, 300);
}
```

Even simpler:
blink two lights
at different intervals

```
uint32_t last_time_2 = 0;
bool led_state_2 = false;


void loop()
{
  uint32_t curtime = millis();

  if (curtime - last_time_1 > 1000) {
    last_time_1 = curtime;
    if (led_state_1)
      digitalWrite(13, LOW);
    else
      digitalWrite(13, HIGH);
    led_state_1 = ! led_state_1;
  }

  if (curtime - last_time_2 > 300) {
    last_time_2 = curtime;
    if (led_state_2)
      digitalWrite(9, LOW);
    else
      digitalWrite(9, HIGH);
```

Functions that use delay()
do not compose

Combining concurrent activities
requires explicit scheduling

“Blinking” is an ongoing process

Need composition in time

A.k.a., concurrency

Any reasonably sophisticated software application for the Arduino consists of:

ad hoc discrete event scheduler +
finite state machine(s)

Fairly advanced to implement

Our Approach

Use Functional Reactive Programming to handle events/streams of events

Use the “foldP” (fold over the past) FRP function to simulate state machines

FRP Classification

Juniper is a higher-order discrete impure monadic FRP Language

What this actually means:

Dynamic signal graphs allowed

Signals of signals are allowed

Lose equational reasoning to avoid space leak

No continuous signals

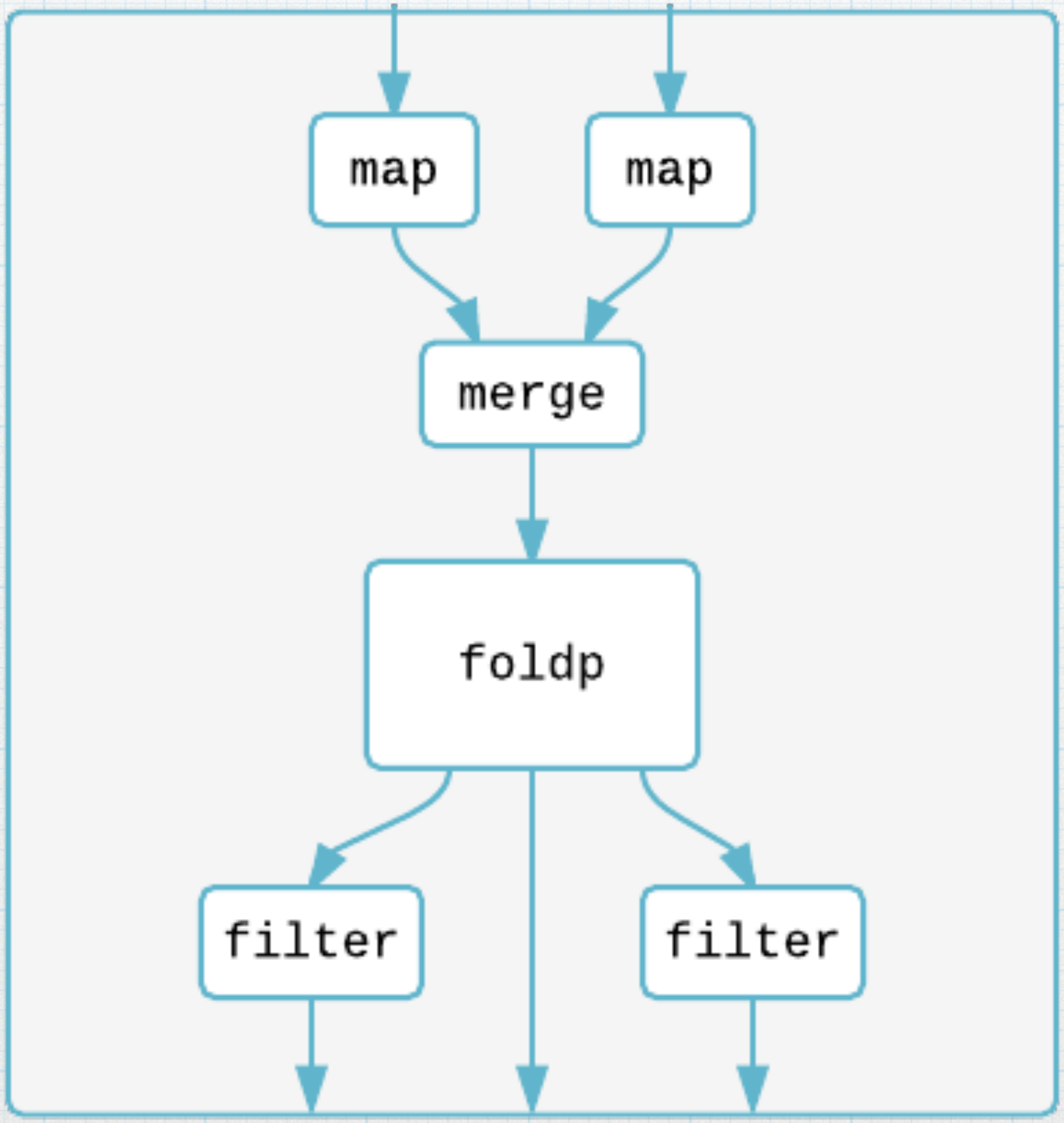
Language Features

- Algebraic data types
- Parametric polymorphic functions
- Lambdas
- Closures
- Type inference
- Limited dependent typing (size is part of an array type)
- Pattern matching
- Immutable data structures
- Imperative features
- Mutable references
- Inline C++

Signal Graphs

Events "flow" along signals or signals are time varying values

Signals connected together form a directed graph



Signal graph representation

2 KB RAM → Not enough space to store the data structure itself + necessary runtime components

One possibility: static signal graph known at compile time - use adjacency list

Our approach: Signal graph embedded within the call graph

Signals in Juniper

```
type maybe<'a> = just of 'a
               | nothing
```

```
type sig<'a> = signal of maybe<'a>
```

Blinking LED in Juniper

```
module Blink
open(Prelude, Io, Time)

let boardLed = 13
let tState = Time:state()
let ledState = ref low()

fun blink() = ...

fun setup() =
    Io:setPinMode(boardLed, Io:output())

fun main() = (
    setup();
    while true do
        blink()
    end
)
```

Blinking LED in Juniper

```
module Io
...
type pinState = high | low
...
```

```
fun blink() = (
    let timerSig = Time:every(1000, tState);
    let ledSig =
        Signal:foldP(
            fn (currentTime, lastState) ->
                Io:toggle(lastState)
            end,
            ledState, timerSig);
    Io:digOut(boardLed, ledSig)
)
```

Compilation

```
type maybe<'a> = just of 'a | nothing
```

```
template<typename a>
struct maybe {
    uint8_t tag;
    bool operator==(maybe rhs) {
        if (this->tag != rhs.tag) { return false; }
        switch (this->tag) {
            case 0:
                return this->just == rhs.just;
            case 1:
                return this->nothing == rhs.nothing;
        }
        return false;
    }

    bool operator!=(maybe rhs) { return !(rhs == *this); }
    union {
        a just;
        uint8_t nothing;
    };
};
```

Compilation

```
while true do
    ...
end
```

```
(([&]() -> Prelude::unit {
  while (true) {
    ...
  }
  return {};
})());
```

Case Study: Digital Hourglass

Rich Set of Behaviors

- Program Mode
- Timing Mode
- Pause Mode
- Finale Mode

C++: 950 lines

(and it required a lot of thought)

Juniper: 350 lines

(and it worked the first time)

Conclusion

- Juniper is a new FRP language designed to be run on small microcontrollers like the Arduino
- Has many functional programming features
- Compiles to C++
- Shows clear benefits for logic re-use; specifically with time dependent behaviors

Thank you!

http://www.juniper-lang.org/