Lab 3c
Fun with your LED cube

ENGR 40M
Chuan-Zheng Lee
Stanford University
19 May 2017

Announcements

- Homework 6 is **not** released today.
  It will be released on Monday (May 22).
  It will be due at 11am Tuesday week (May 30).

- Homework 6 prepares you for lab 4.

- There is still a prelab 4.

- Homework 7 will be released Monday, May 29, and will be due Monday, June 5.

Overview of lab 3c

- Your task is to get your cube to respond to some sort of **input**.

- For most of you, this will involve adding hardware.

- You choose what you want to do, subject to minimum requirements.

Our suggestions for input

- Serial data
- Potentiometer
- Pushbutton switches (or other switches)
- Audio
- Capacitive touch sensing

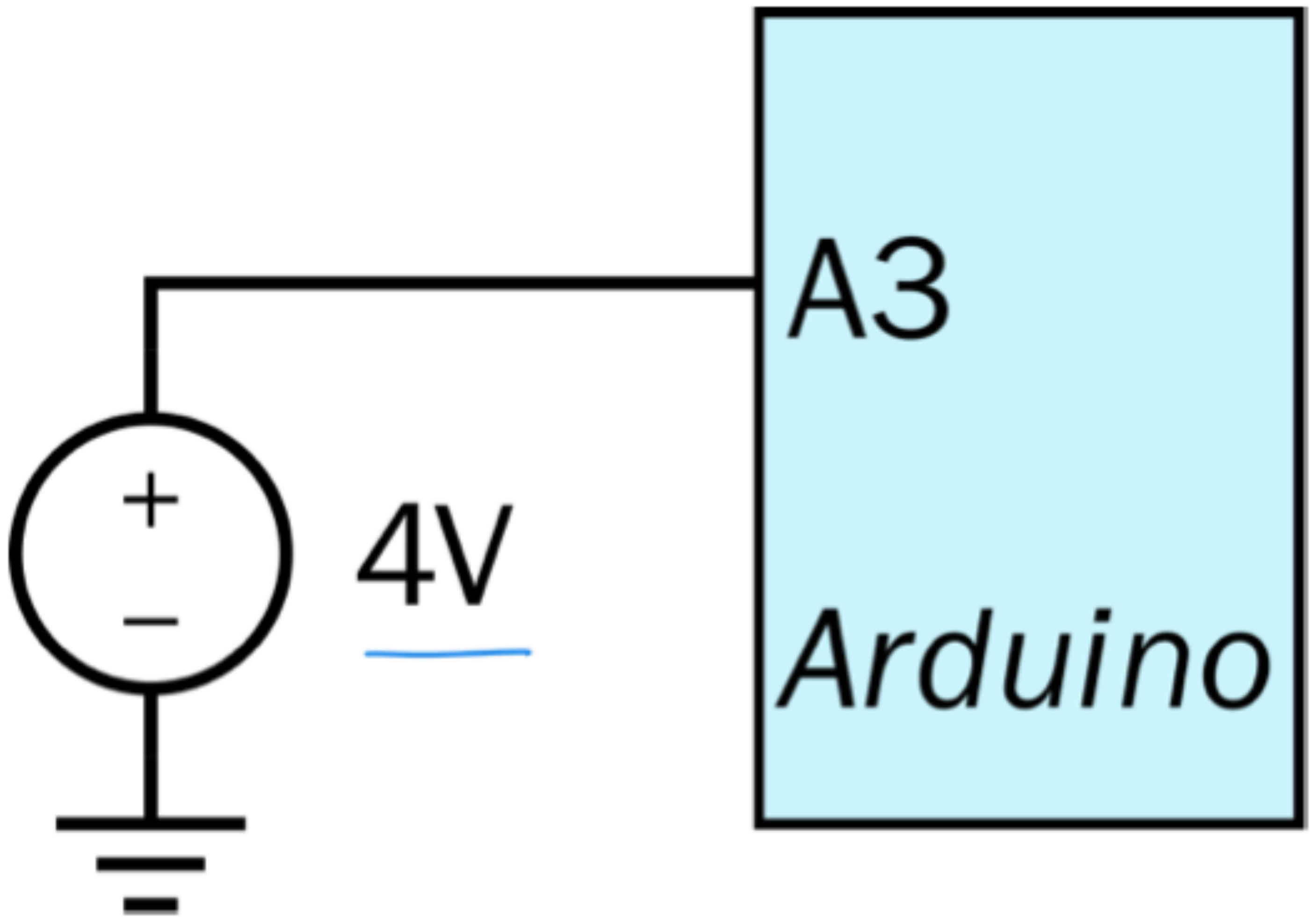
You’re free (and encouraged) to propose something else, but be sure to tell your TA well in advance!

Serial data

- You’ve already used the Serial Monitor
- You can also write computer software to interact with the serial port directly
- If you *only* respond to serial data, we’ll expect to see some impressive software

Analog-to-digital converter

- `analogRead()` reads the voltage on the pin, scaled to be a number between 0 and 1023



```
void loop() {
  int reading = analogRead(A3);
  Serial.println(reading);
}
```

Serial Monitor:

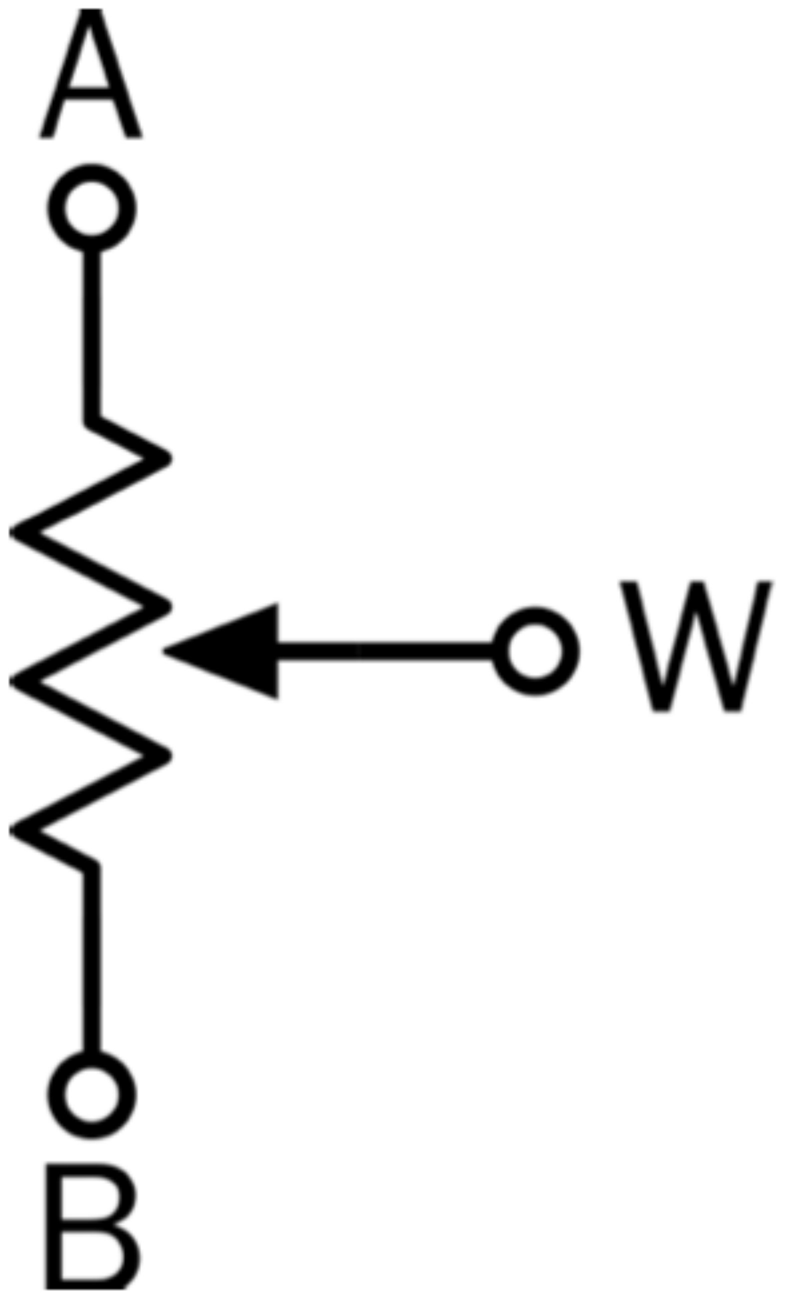
```
819
819
```

$\frac{4}{5} \times 1024 \approx 819$

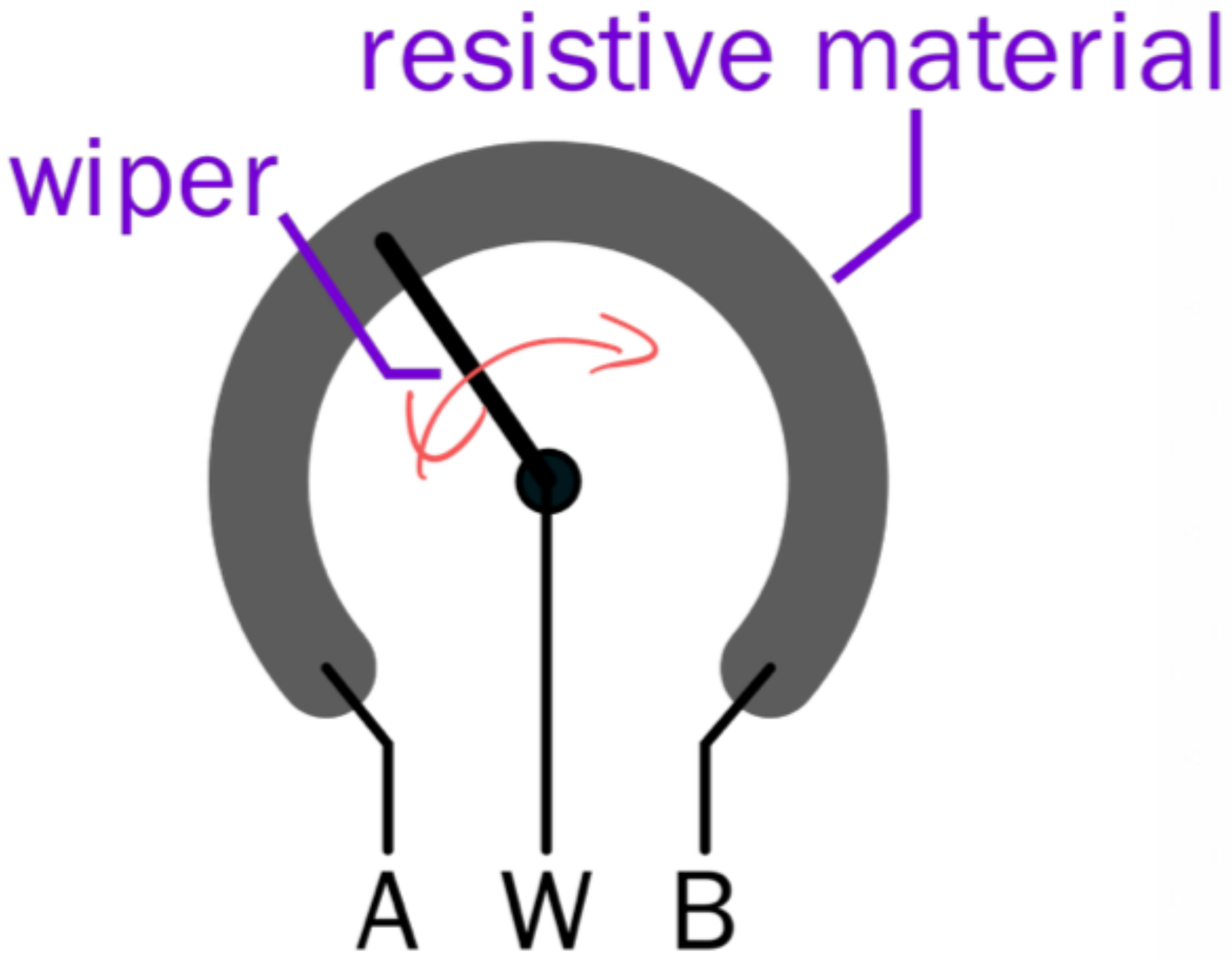
- This takes longer than `digitalRead()`, but this probably won't bother you

Potentiometer

- Essentially a variable voltage divider



standard symbol

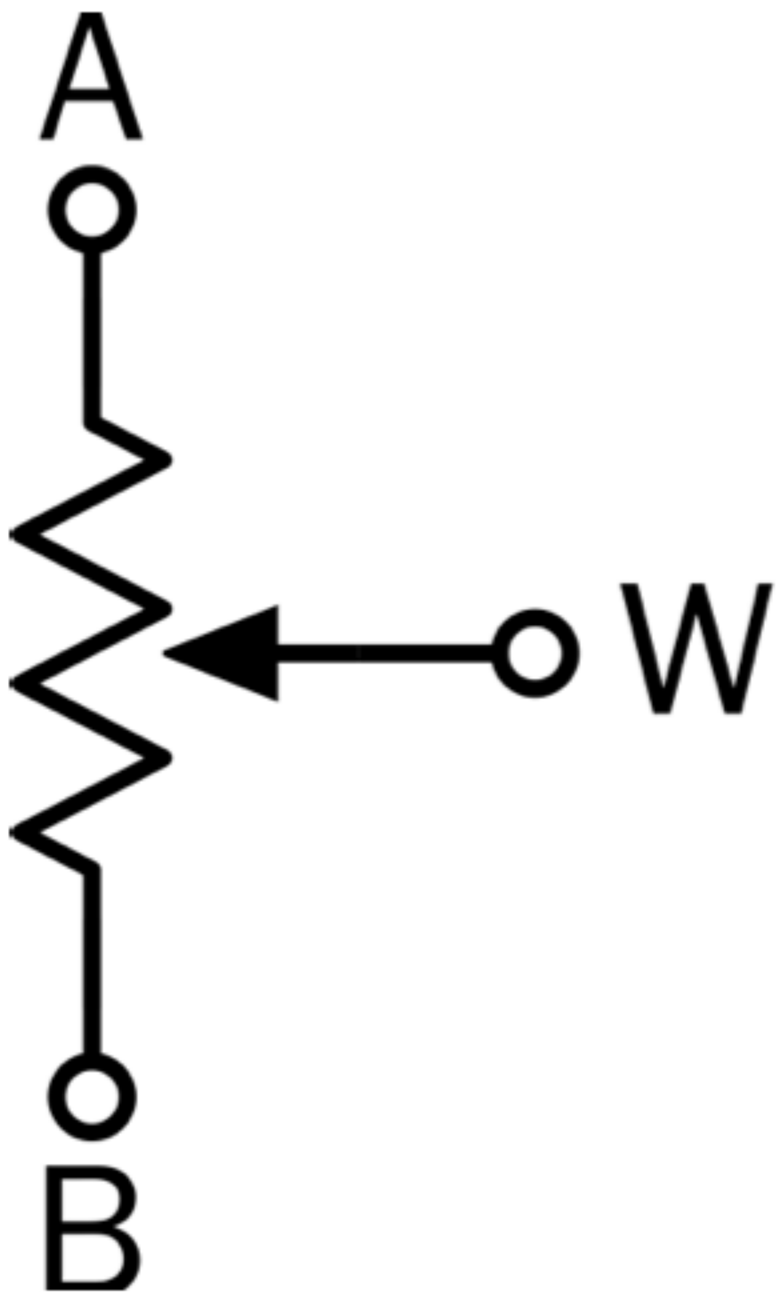


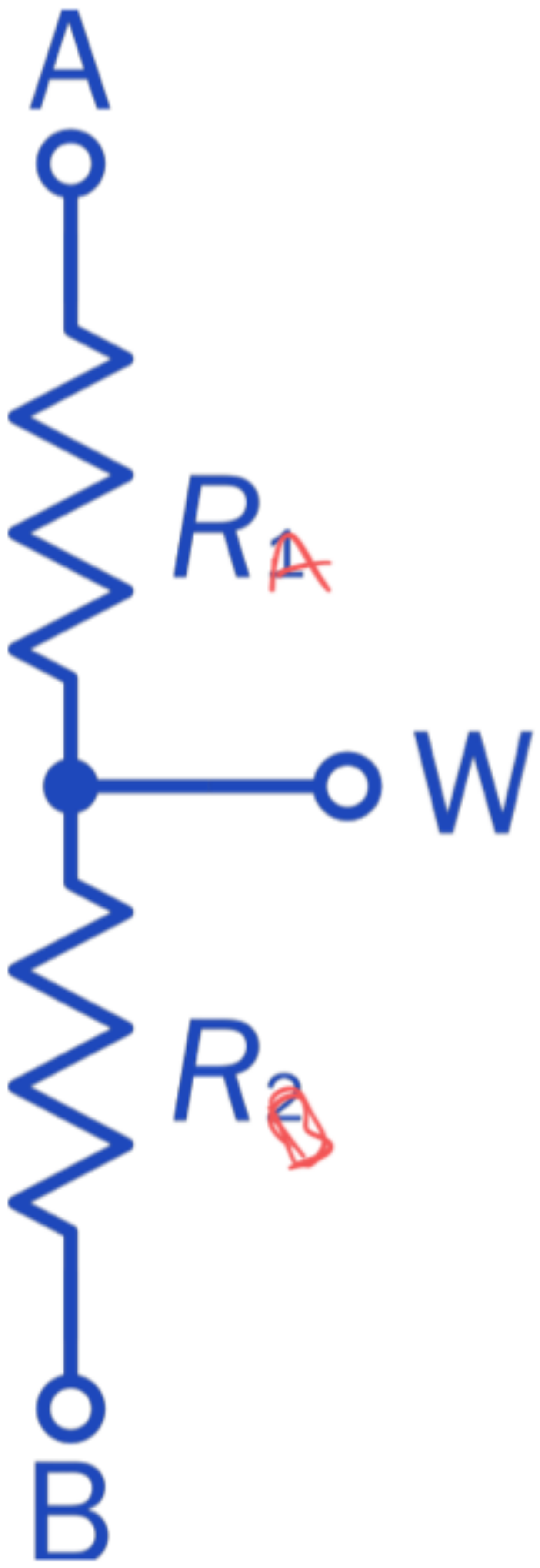
internal mechanics

photo

Potentiometer

- Essentially a variable voltage divider

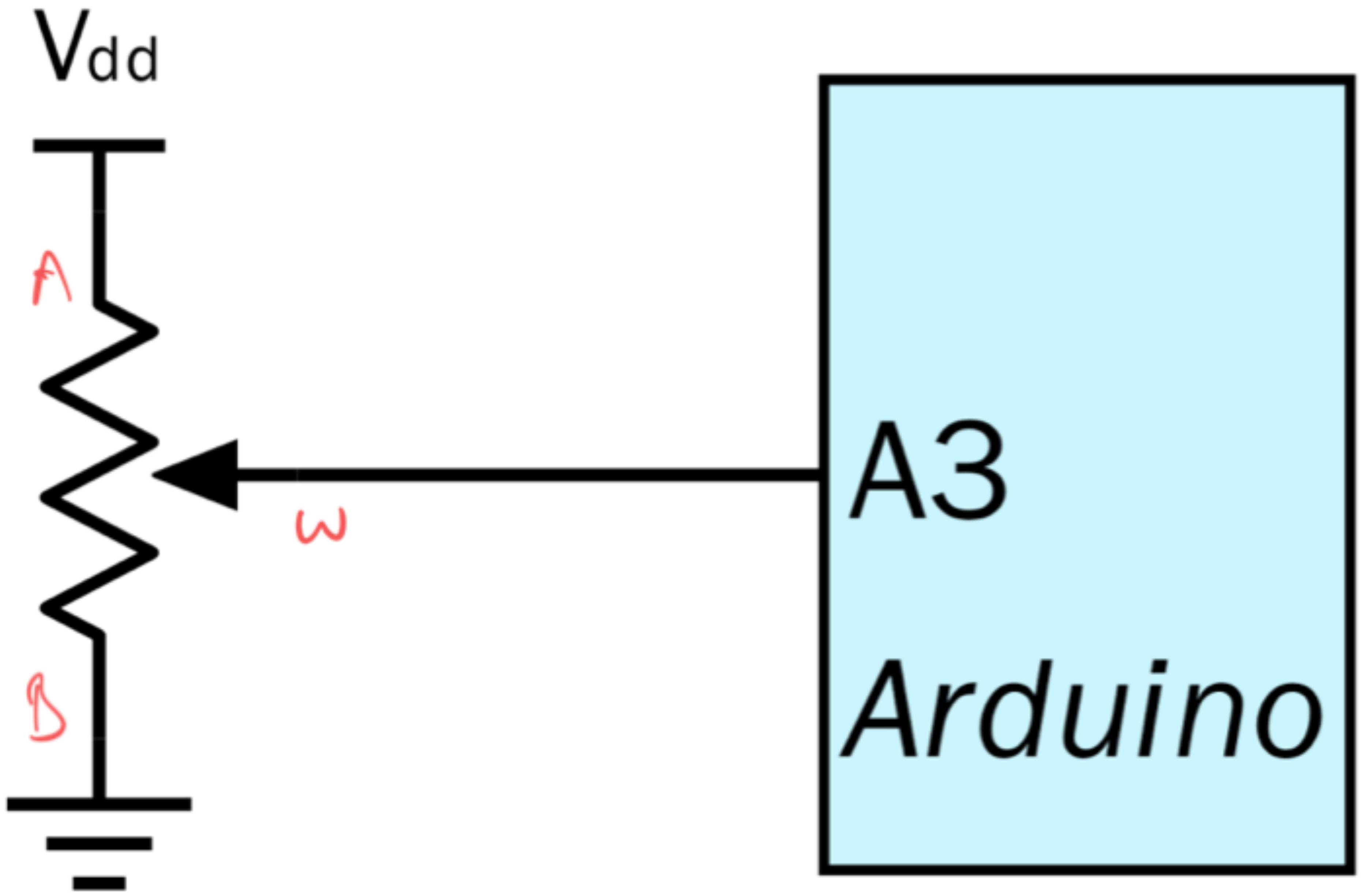




R_1 and R_2 vary with the wiper, but
$R_1 + R_2 =$ constant

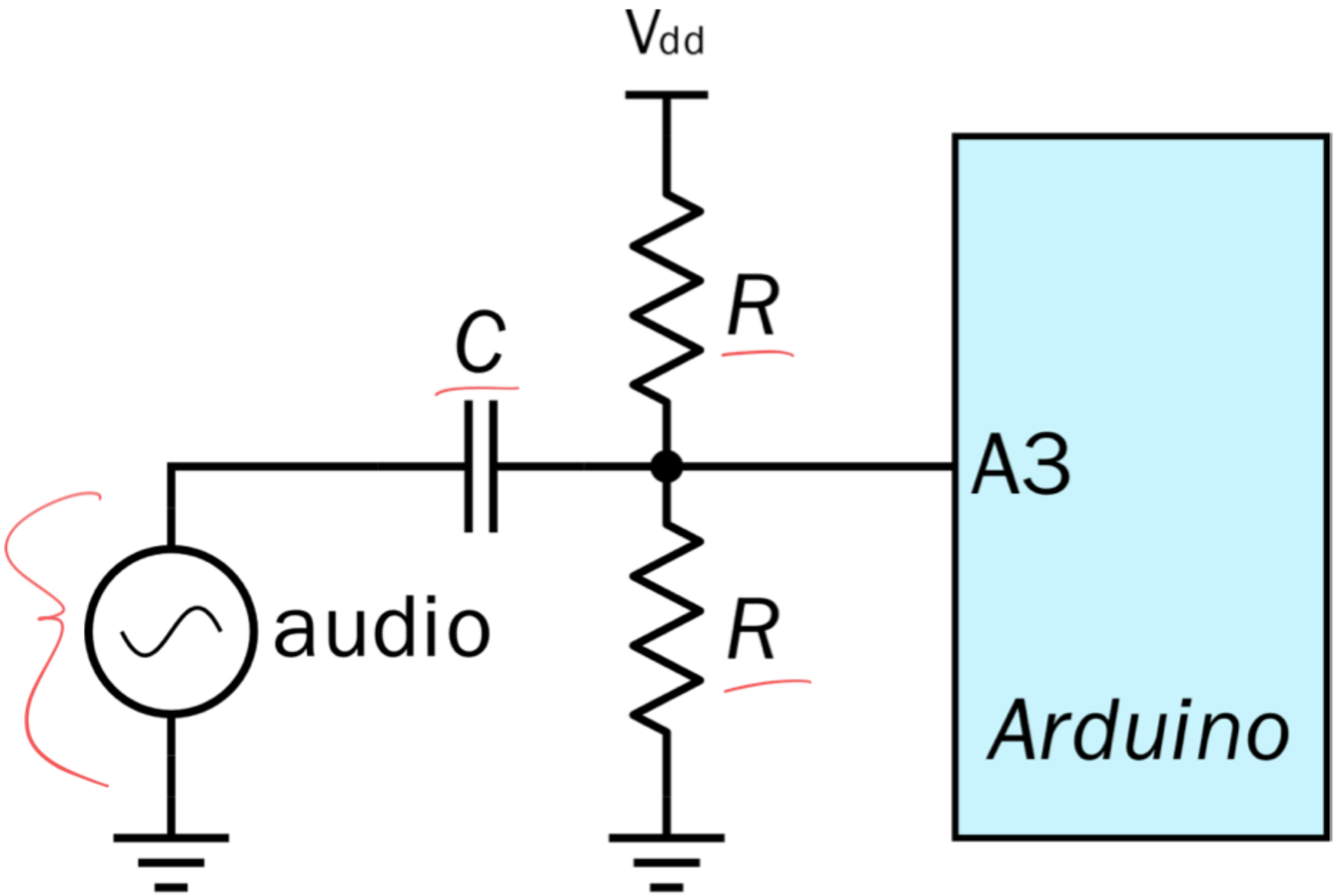
Potentiometer

- Connect the wiper to the analog input
- Read using `analogRead()`



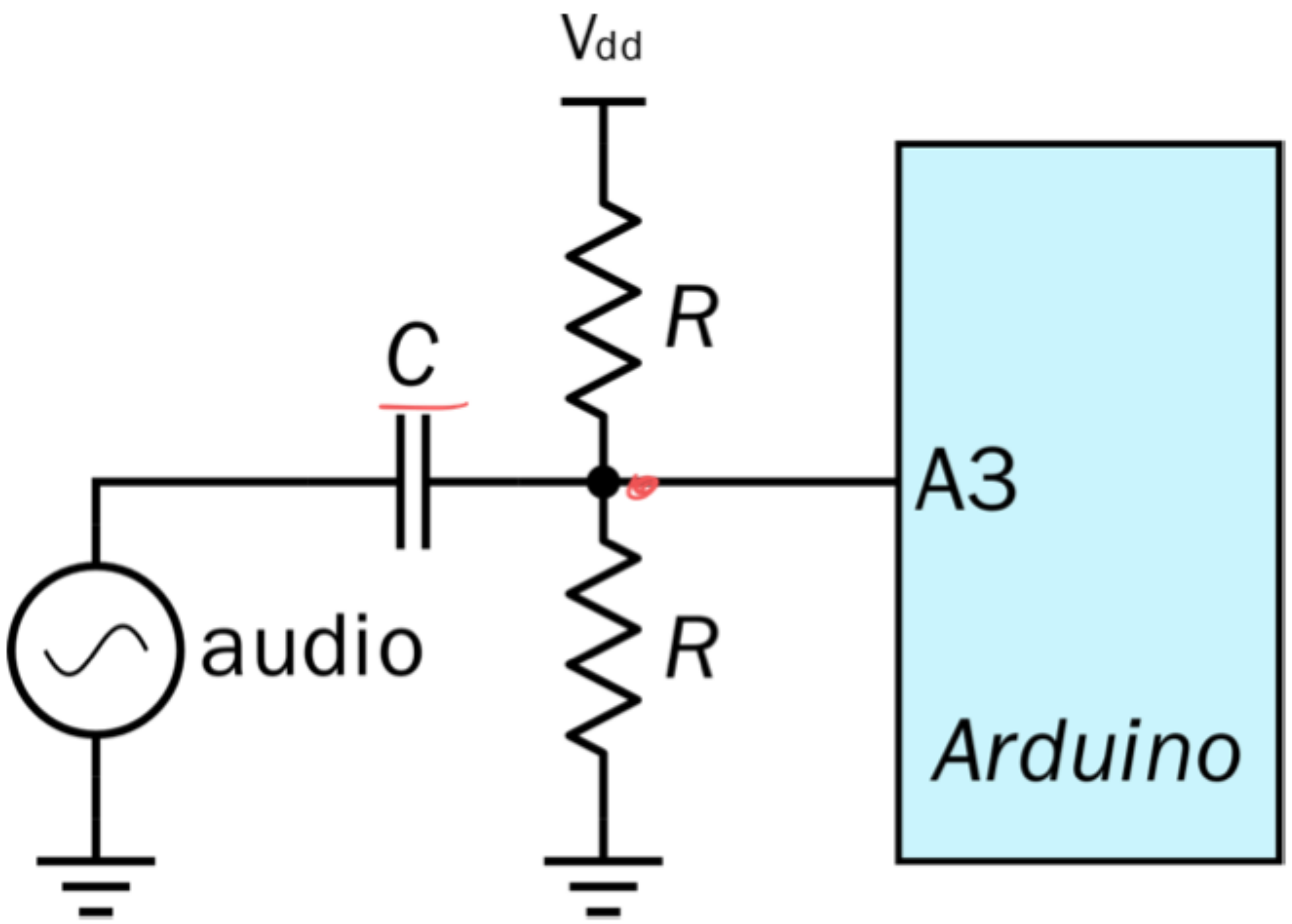
Audio

- You can connect an audio output to your Arduino… almost.



Audio

- `analogRead()` the instantaneous value
- The input will be “centered” at 2.5 V, so you’ll need to process it in software
- *An additional handout guides you through this*



$\frac{1}{2\pi f C}$

Audio—frequency response

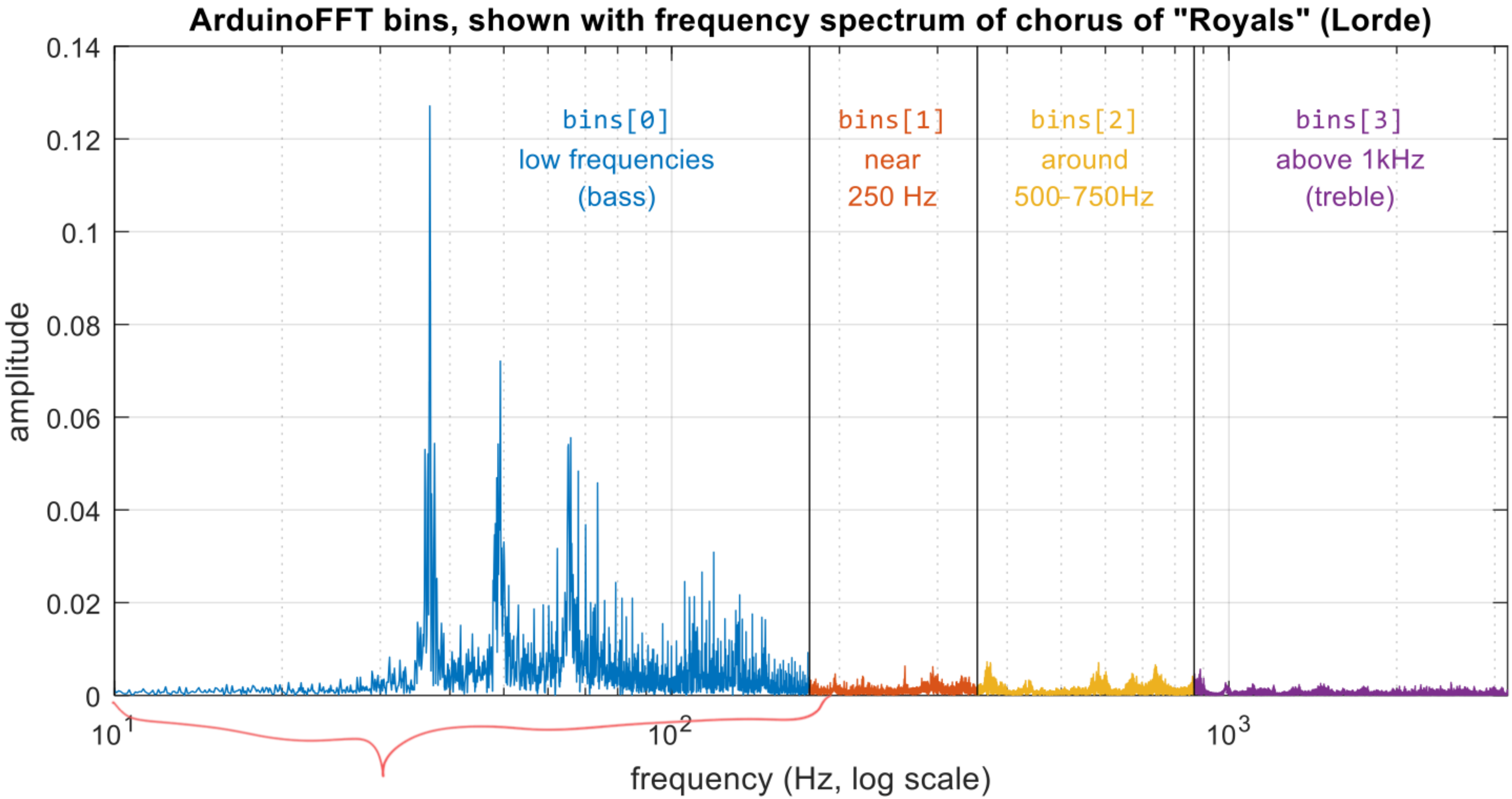
- The **ArduinoFFT3** library can process your signal to return a frequency-domain representation

- Implements the *fast Fourier transform*, an algorithm which computes a close cousin of the Fourier series

- *An additional handout guides you through this*

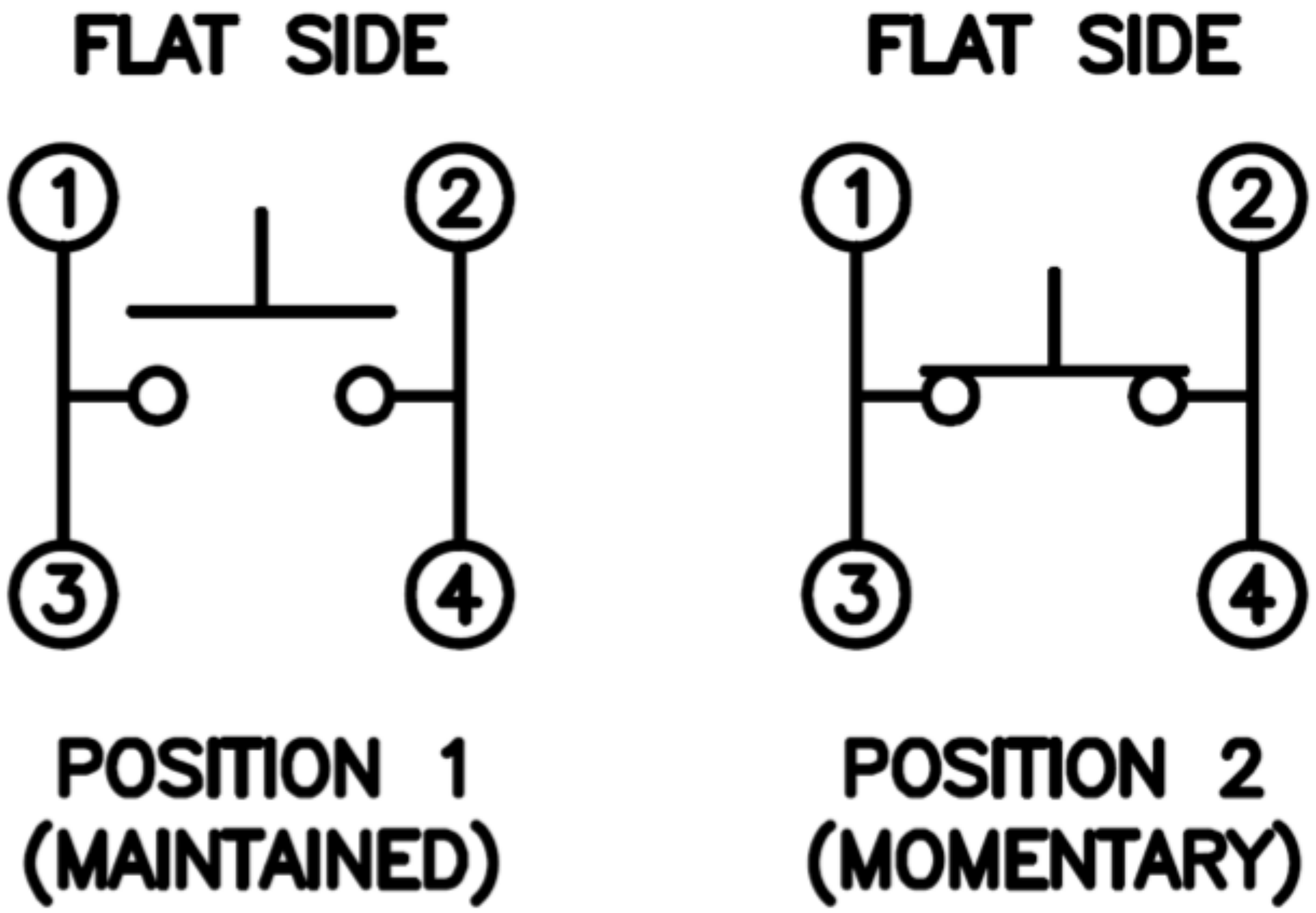
- Video: https://youtu.be/FRXDTiOHFII

Audio—frequency response



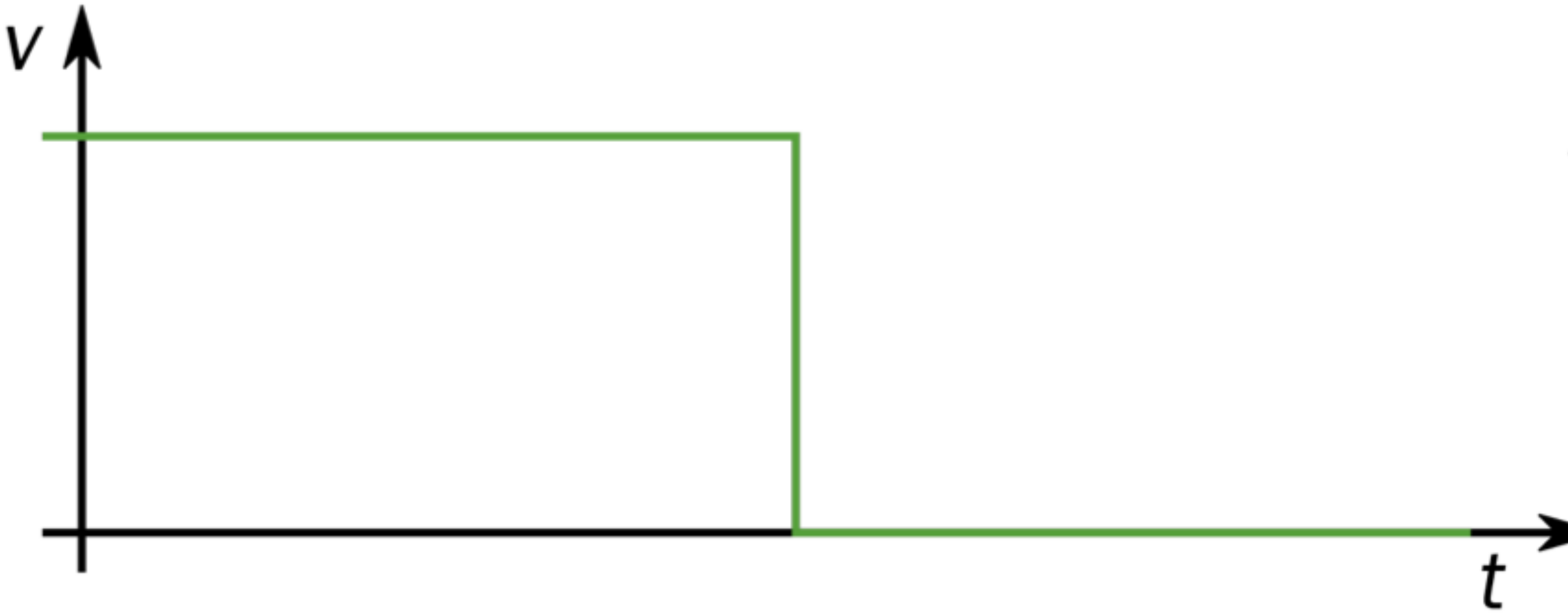
Pushbutton switch

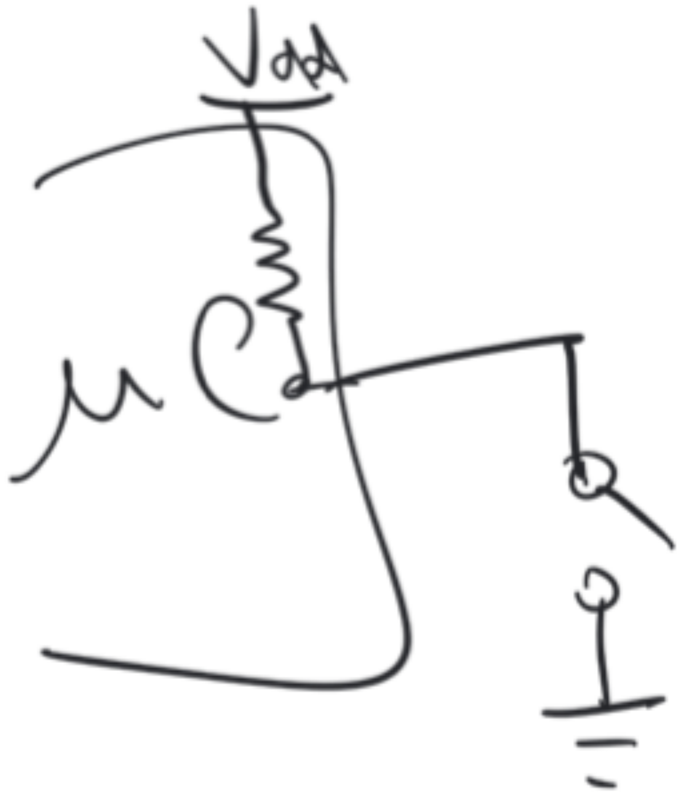
- SPST, momentary and normally open (sometimes known as *push-to-make*)



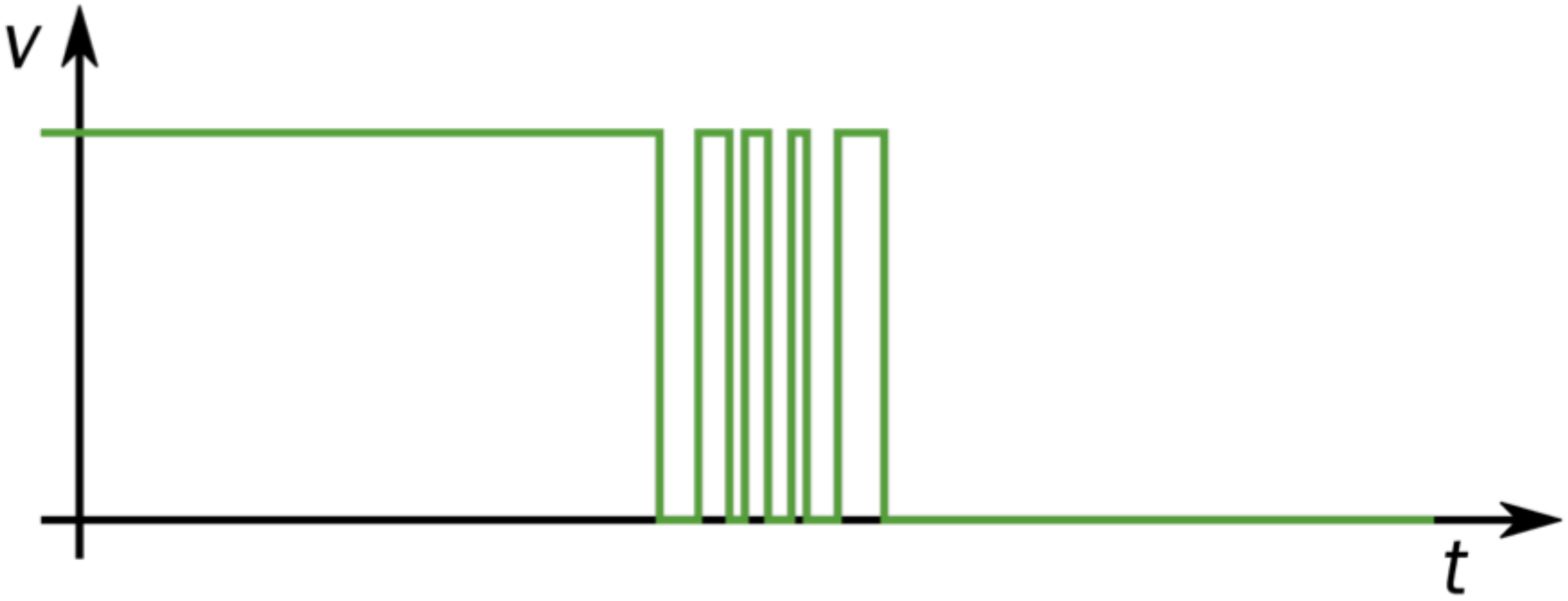
Pushbutton switch: debouncing

What we think a switch does:

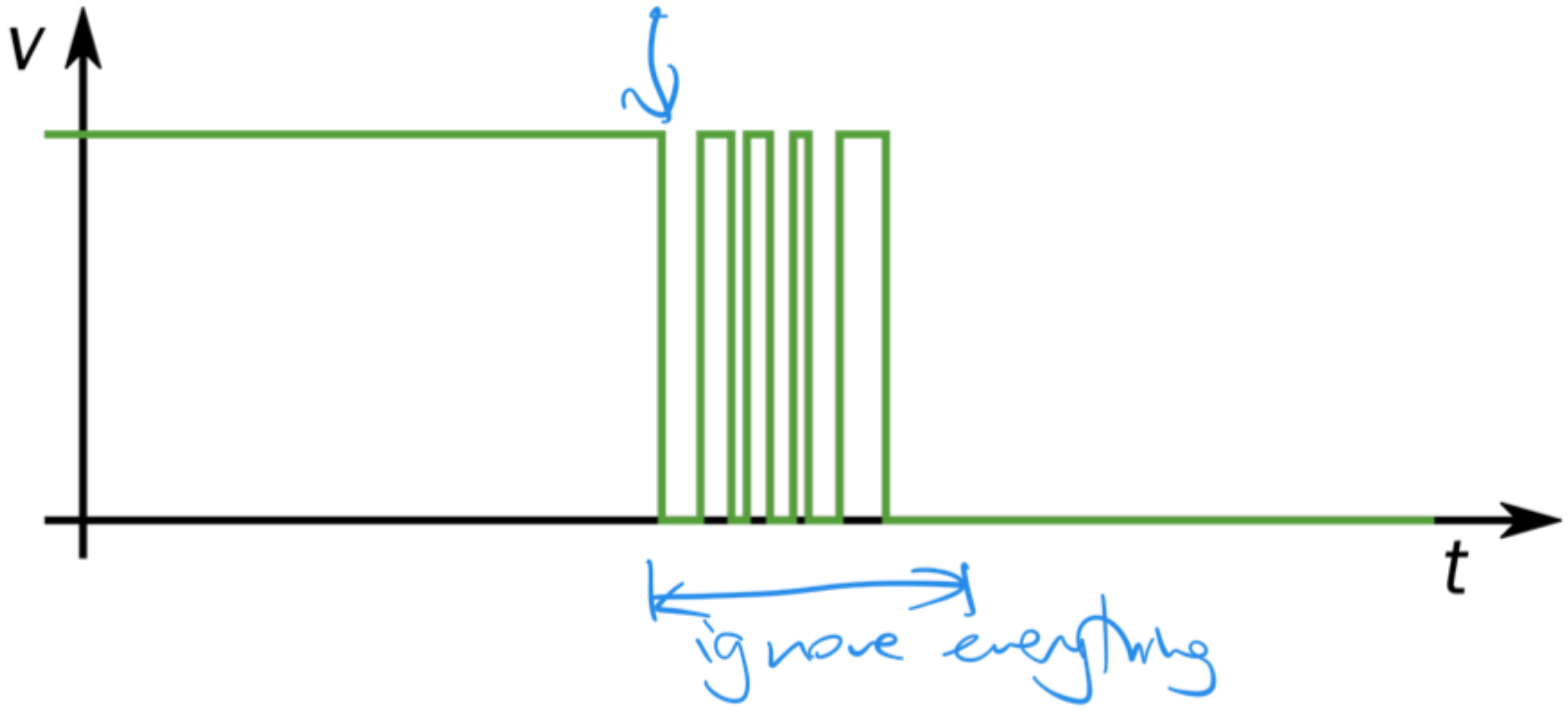




What it actually does:



Pushbutton switch: debouncing



- As the switch bounces, the Arduino can register many transitions
- Dealing with this is called *debouncing*
- One strategy: Ignore transitions too soon after the last one
- *An additional handout guides you through this*

Programming a pattern

- Since you've *abstracted* away the time-multiplexing part, displaying a pattern consists mainly of filling in the `pattern` array.

- Recall triply-nested loops:

```
for (int z = 0; z < 4; z++) {
  for (int y = 0; y < 4; y++) {
    for (int x = 0; x < 4; x++) {
      pattern[z][y][x] = (some value);
    }
  }
}
```

Raindrop pattern

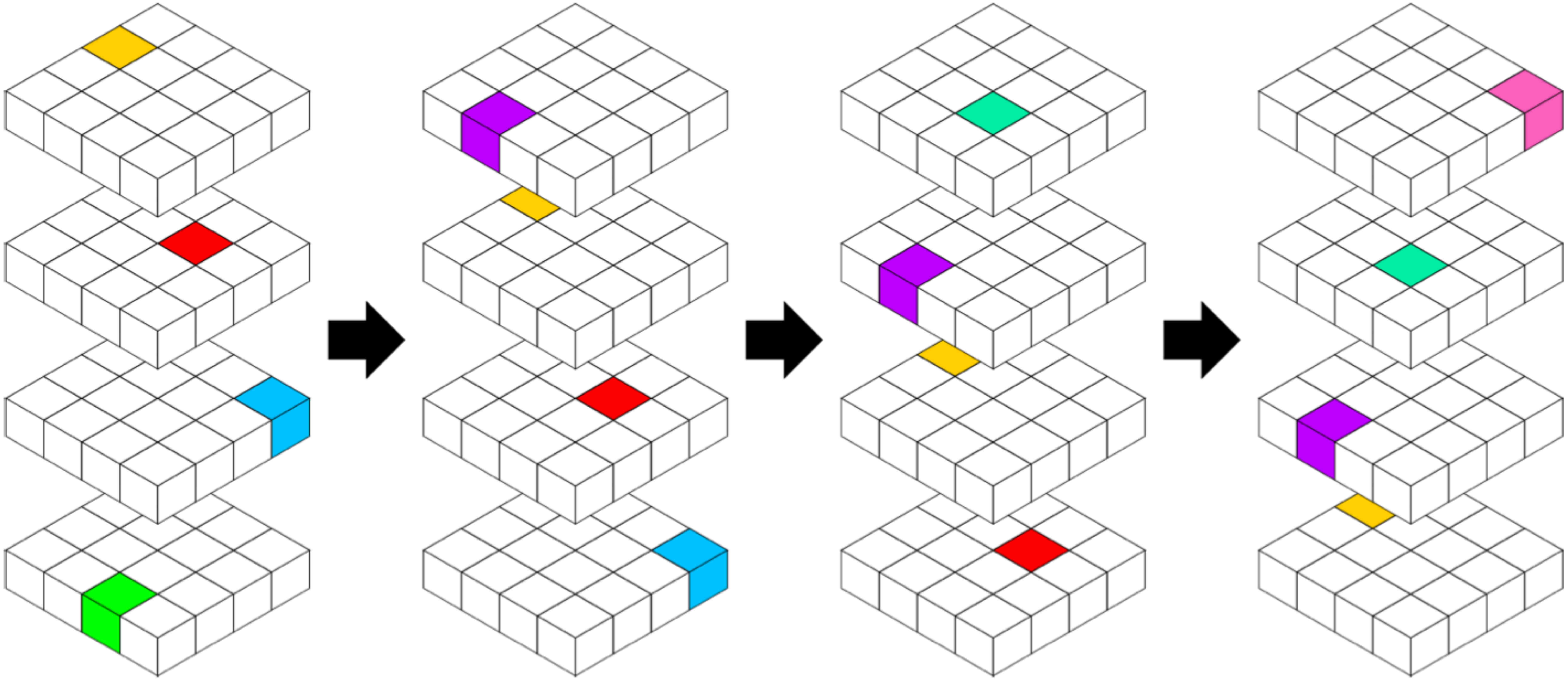
- The raindrop pattern looks like rain is falling from the top of the cube to the bottom

- Videos
  - https://youtu.be/-tZJ-3NSIhY?t=52
  - https://youtu.be/DahwcDeqyA0

Raindrop pattern

Every time period (say, 150 ms):

1. Move the pattern down by one plane
2. Choose an LED at random in the top plane



Decomposition

Every time period (say, 150 ms):

1. Move the pattern down by one plane
2. Choose an LED at random in the top plane

- *Decompose* this into smaller steps:
  1. `movePatternDown(pattern)`
  2. `chooseRandomLEDInTopPlane(pattern)`
- As a principle, each function should do exactly one thing

Timing and inputs

- Update the pattern once every (say) second
- With *no inputs*, this will work:

```
void loop() {
   static byte ledOn[4][4][4];
   updatePattern(ledOn);  // updates pattern
   delay(1000);
}
```

Timing and inputs

- Stop/start whenever the button is pressed
- Why won't this work?

```
void loop() {
    static byte ledOn[4][4][4];
    static bool running = false;
    if (running)
        updatePattern(ledOn);    // updates pattern
    if (digitalRead(BUTTON) == HIGH)
        running = !running;
    delay(1000);
}
```

state

~~HIGH~~ Low

Timing and inputs

- Better: Check button without delay

```
void loop() {
    static byte ledOn[4][4][4];   state
    static bool running = false;
    static long nextUpdateTime = millis();
    if (millis() > nextUpdateTime) {
        if (running)
            updatePattern(ledOn);      // updates pattern
        nextUpdateTime += 1000;
    }
    if (digitalRead(BUTTON) == ~~HIGH~~)
                               Low
        running = !running;
}
```

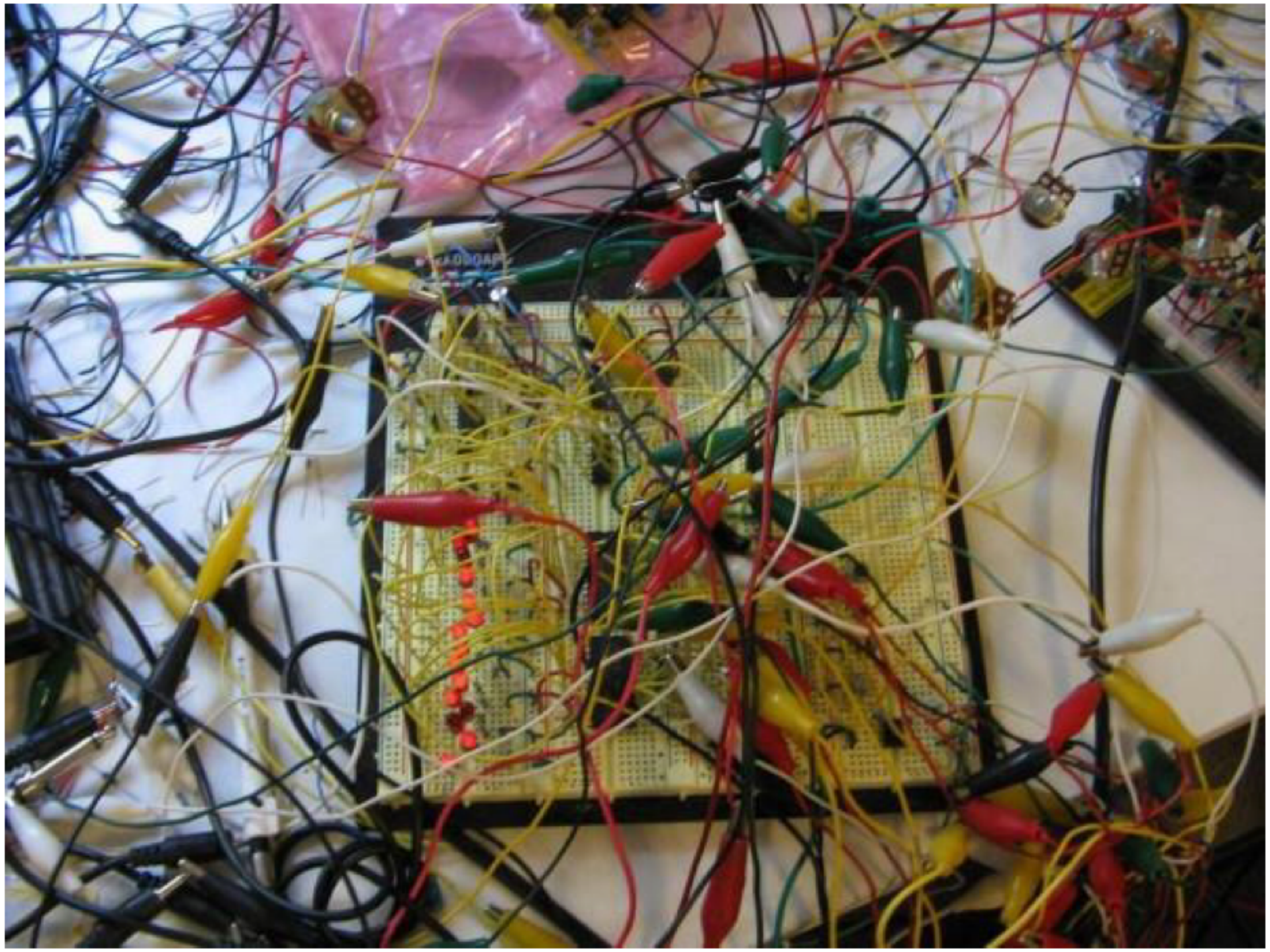
Complexity requirements

- Minimum requirement:
  cubes: 25; 6×6 planes: 35; larger planes: 30
- Must do one “additional handout”, or propose your own
- Details are in the “overview” handout

Points	Hardware	Software
5	No additional hardware (includes serial data)	Simple response
10	Minor additional hardware	Minor complexity • *Raindrop pattern*
15	Moderate additional hardware • *Pushbuttons* • *Audio non-frequency*	Moderate complexity
20	Complex additional hardware	Impressive complexity

Breadboard style

Please don't:



http://www.electro-music.com/forum/phpbb-files/thumbs/t_klee_bb_131.jpg

Coding style

IT'S LIKE A LEET-SPEAK TRANSLATION OF A MANIFESTO BY A SURVIVALIST CULT LEADER WHO'S FOR SOME REASON OBSESSED WITH MEMORY ALLOCATION.
I CAN GET SOMEONE ELSE TO REVIEW MY CODE.
NOT MORE THAN ONCE, I BET.

https://www.xkcd.com/1833/