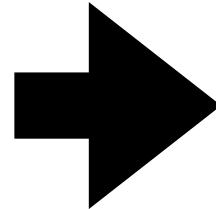
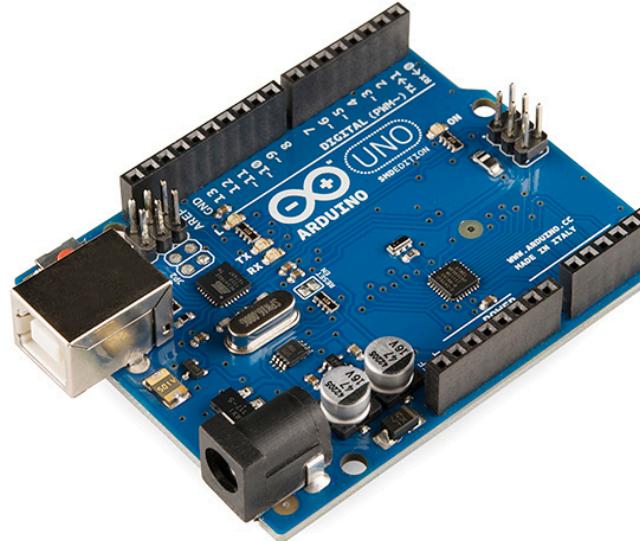


Intro to Arduino

DfPI

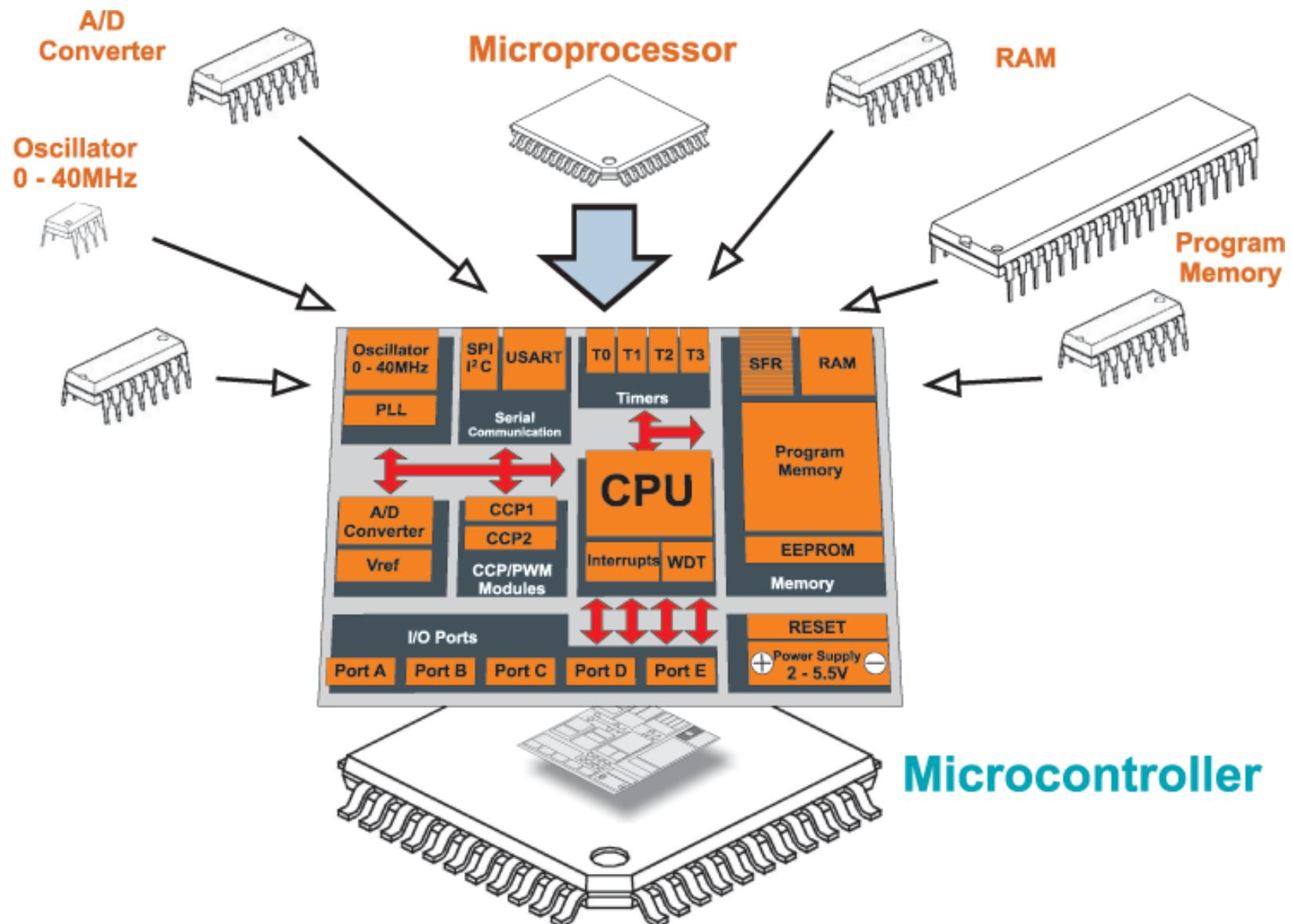


Arduino

- Essentially a mini computer.
- Can control motors, read data from sensors etc...
- Many problems already solved by enormous support community.
- Loads of tutorials online ->

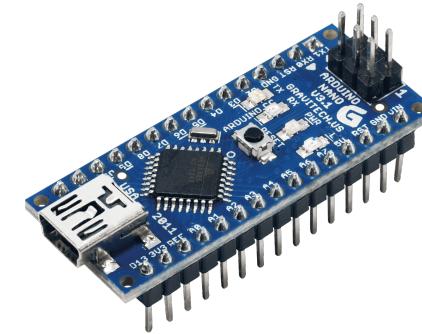


What is a micro controller?





Uno



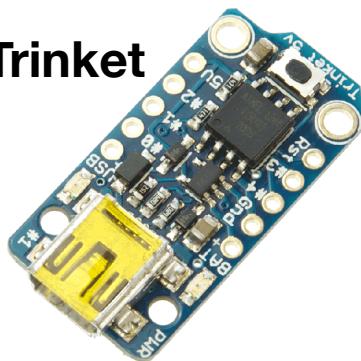
Nano



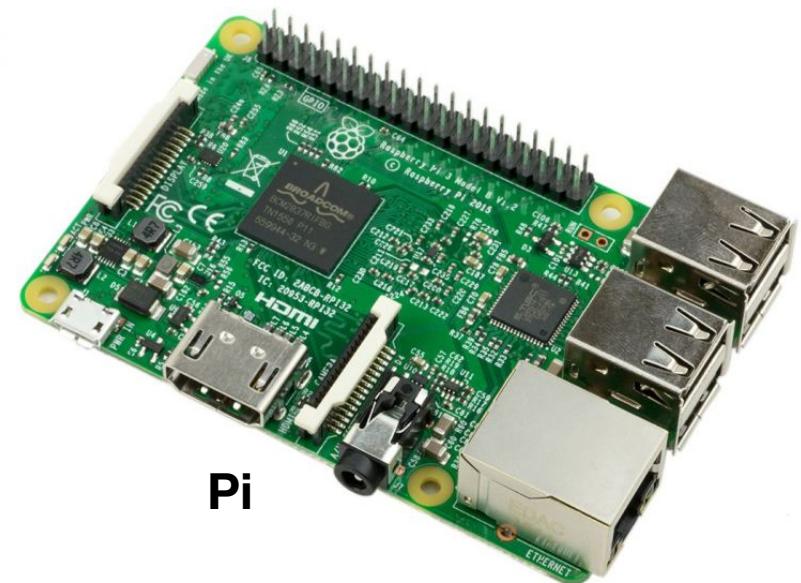
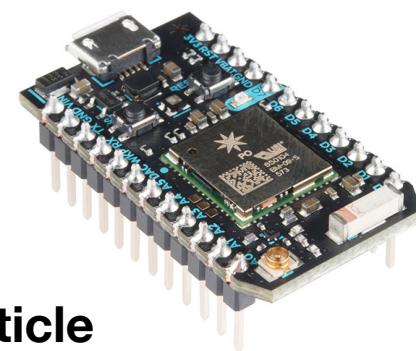
Pi zero

Micro controller Dev. Boards

Trinket

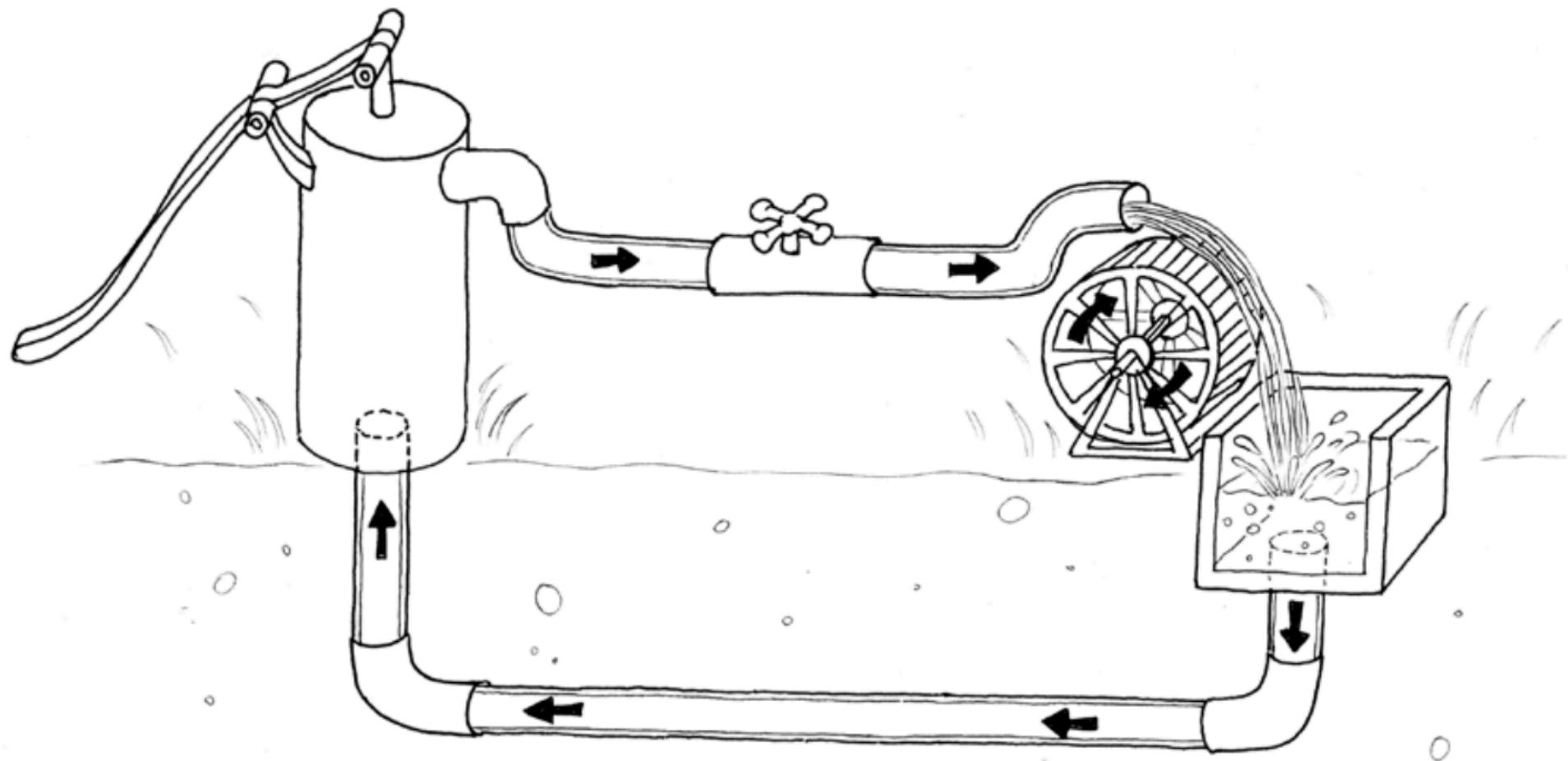


Particle



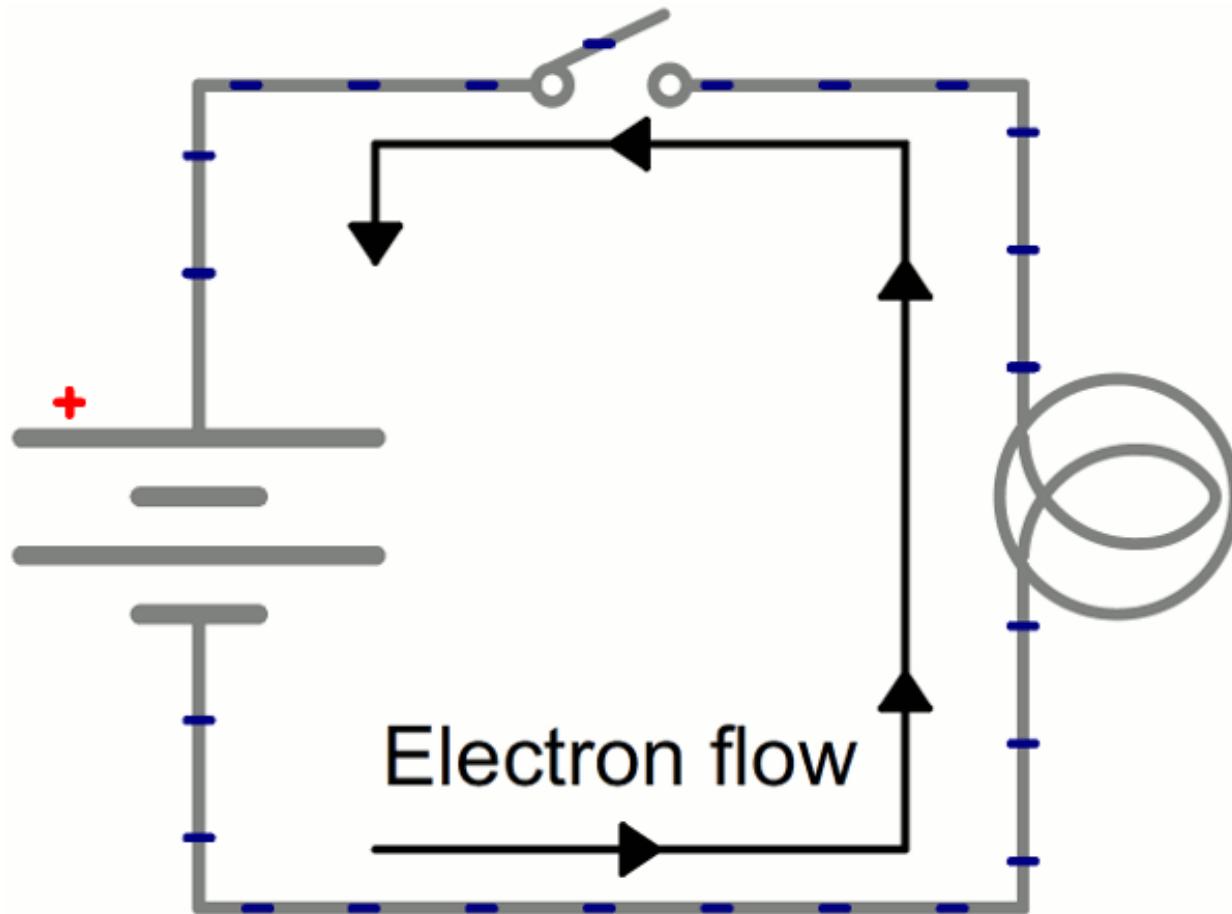
Pi

Electronics revision



There needs to be a complete circuit for current to flow.

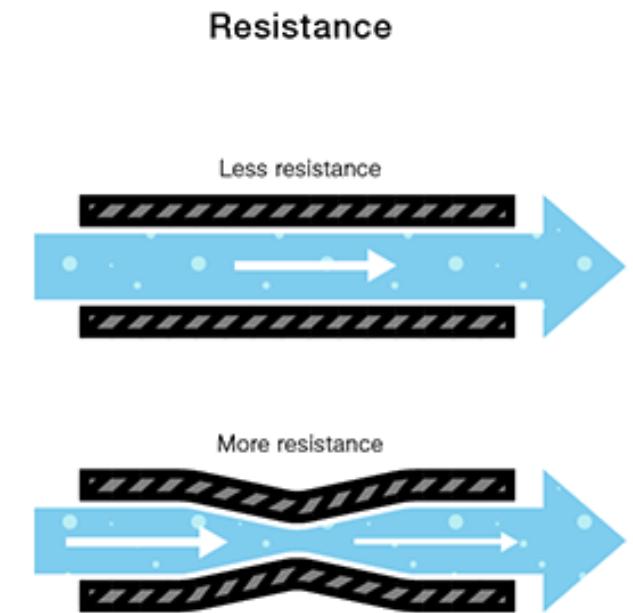
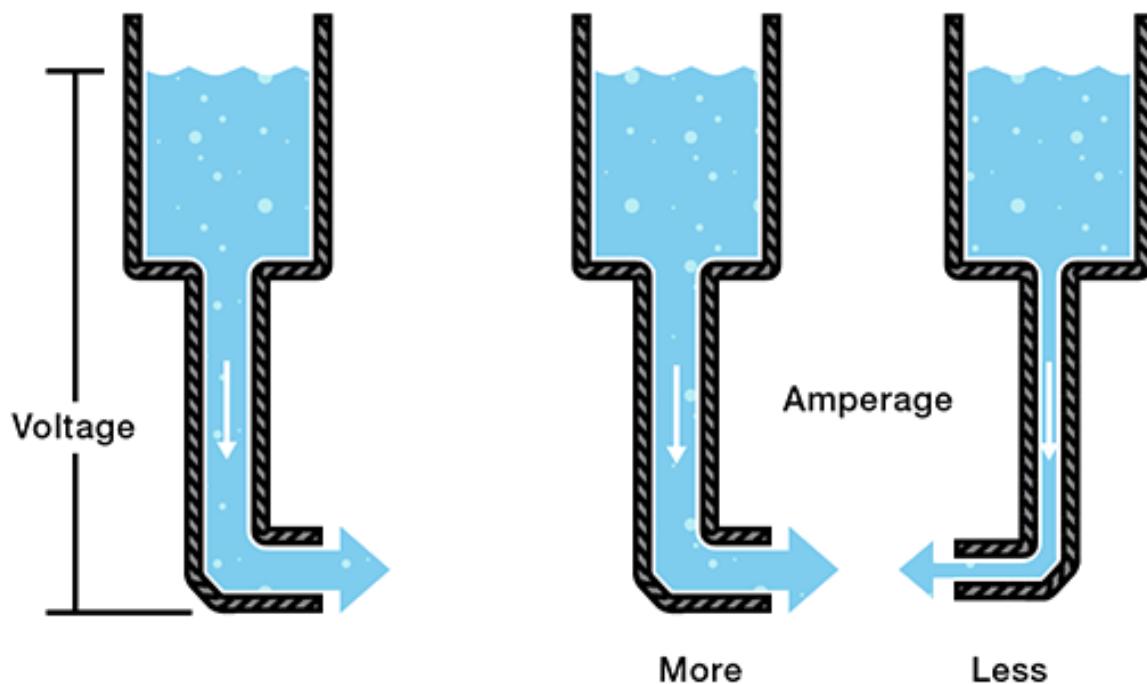
Current flow visualisation



There needs to be a **completed circuit** for current to flow.

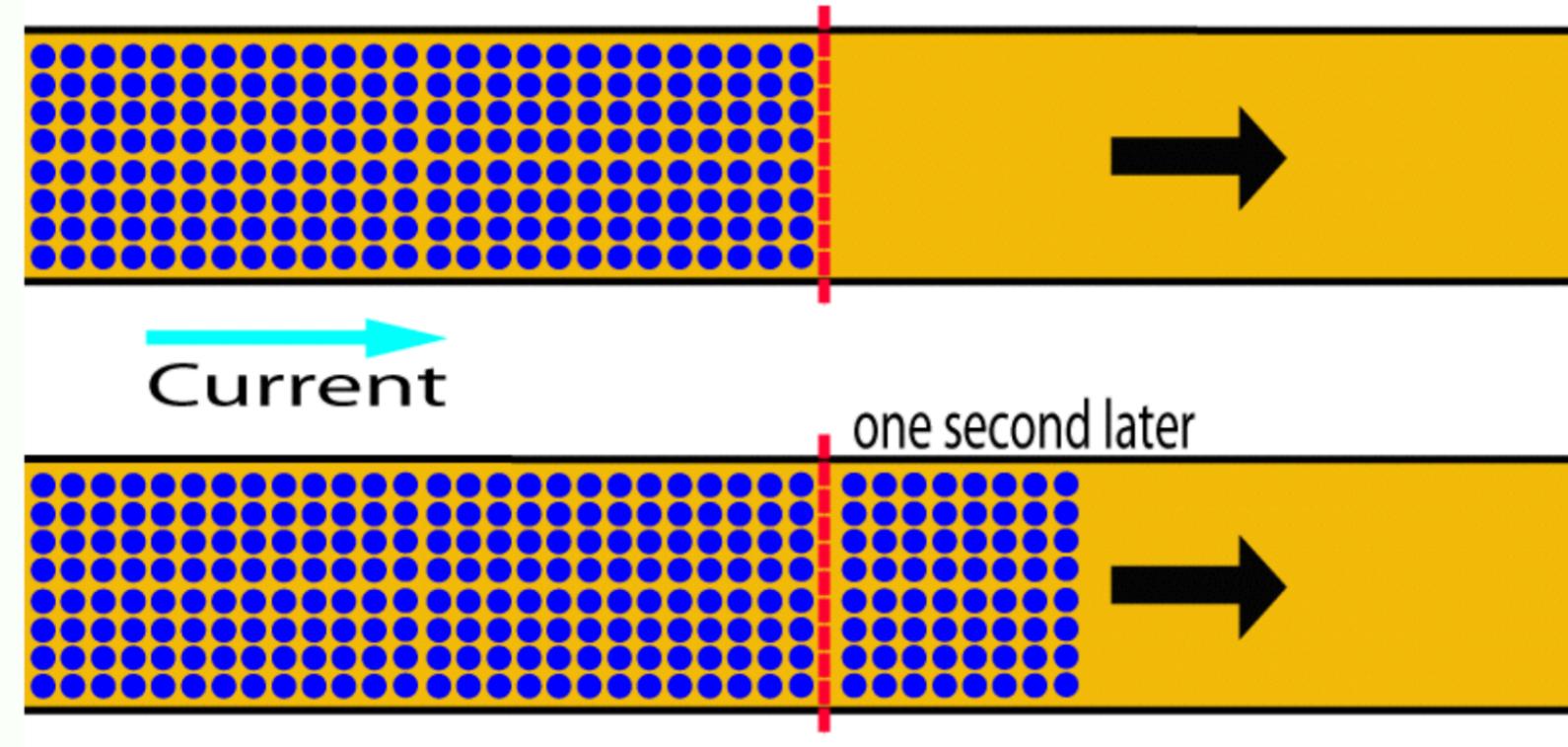
*Does anyone know why the electrons are moving from negative to positive?

Current and Voltage



- **Current - I** - Measured in Amperes (A), charge per unit time.
- **Voltage - V** - Measured in Volts (V), Electro motive force, EMF.
- **Resistance - R** - Resistance measured in Ohms, (Omega)

Visualisation of electrical current:



- **Amperes** are a measure of the rate of charge flow through an electrical circuit.

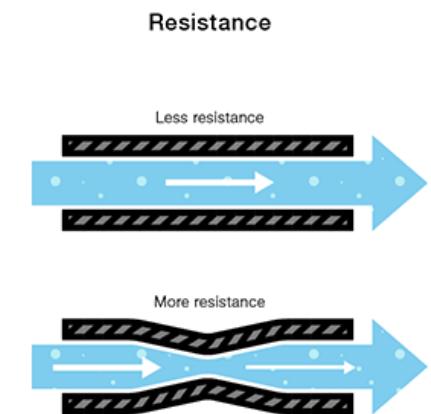
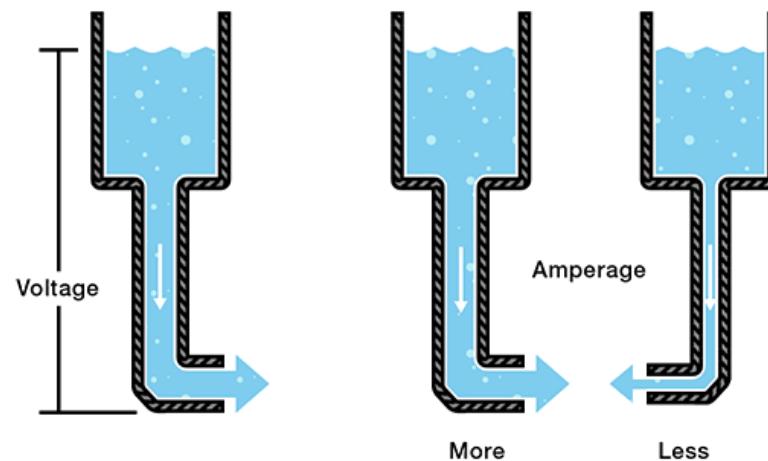
Super important Law!

Ohm's Law

$$V = I * R$$

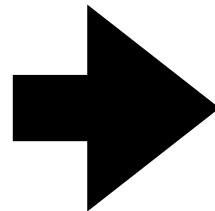
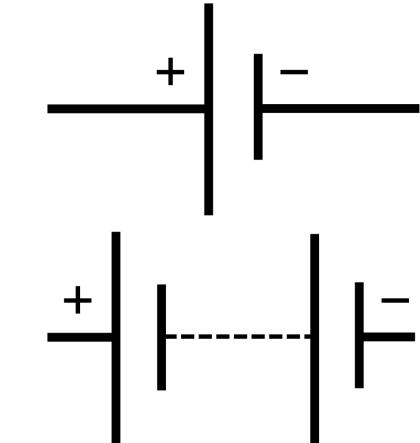
voltage in volts current in amps resistance in ohms

- **V** = Voltage
- **I** = Current
- **R** = Resistance

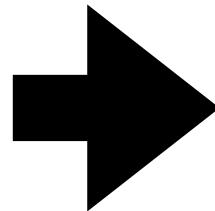


Usually you will only be varying the **Voltage** or **Resistance** in a circuit, and this will govern how the **Current** behaves.

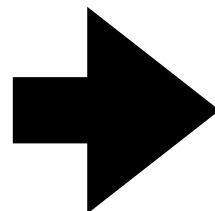
Schematics & Symbols



Resistor



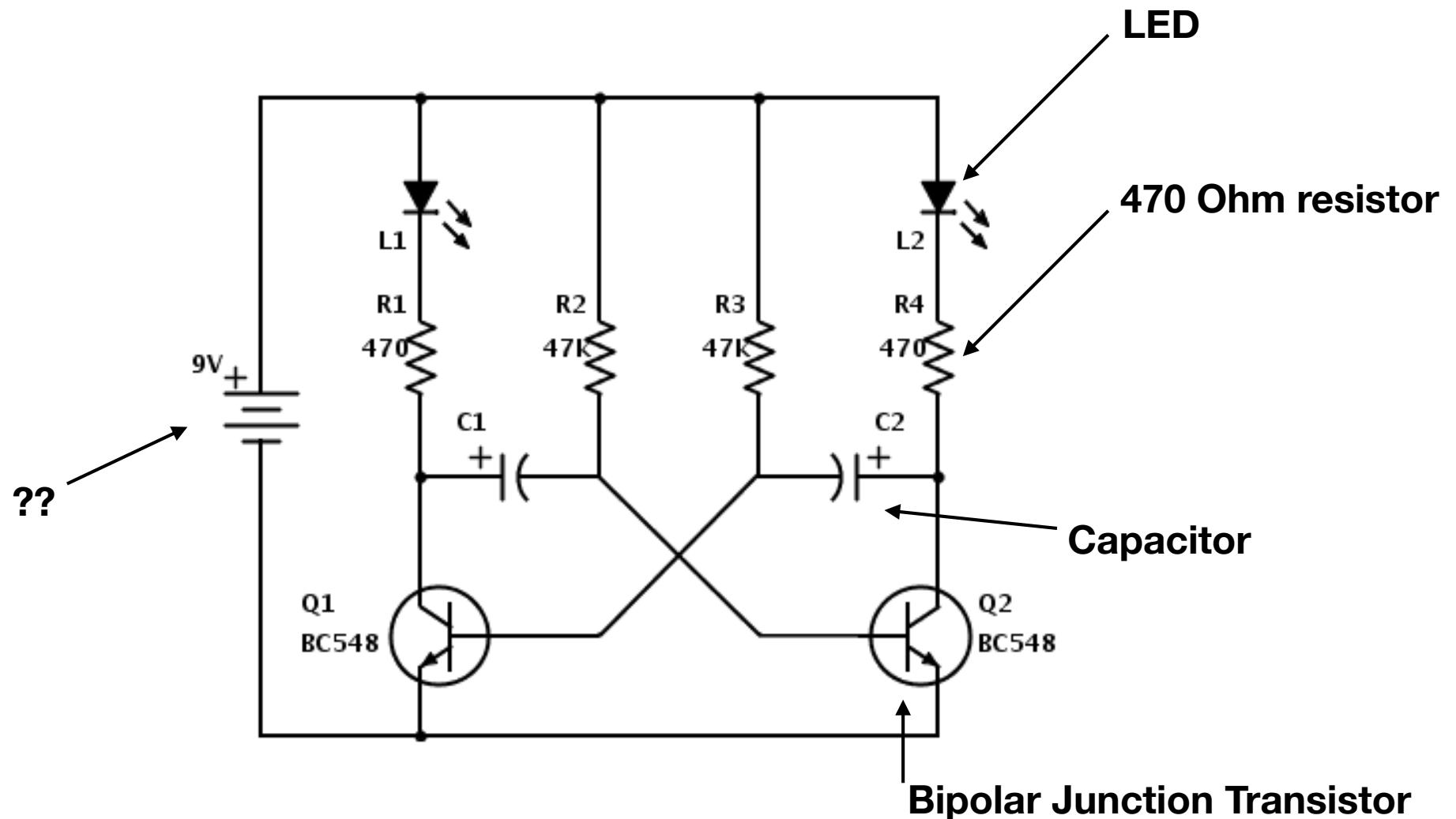
Battery



Ground

These are just a few of the essential symbols

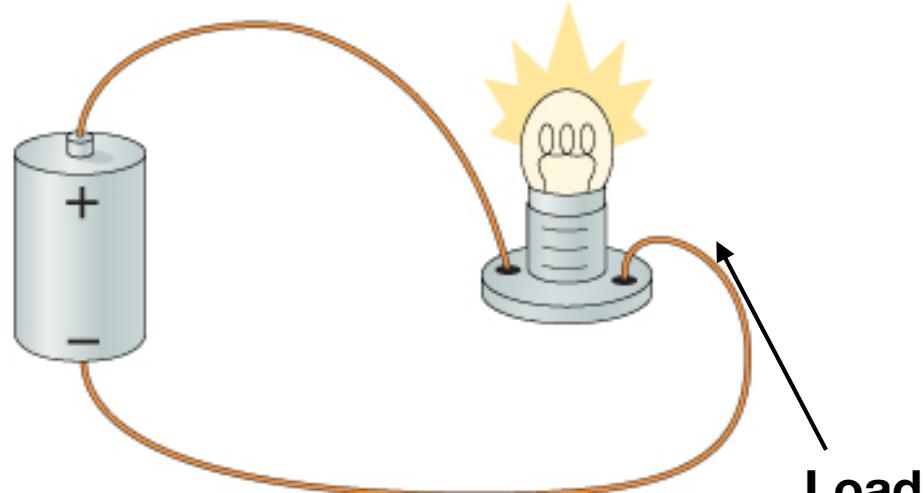
Electrical Schematics



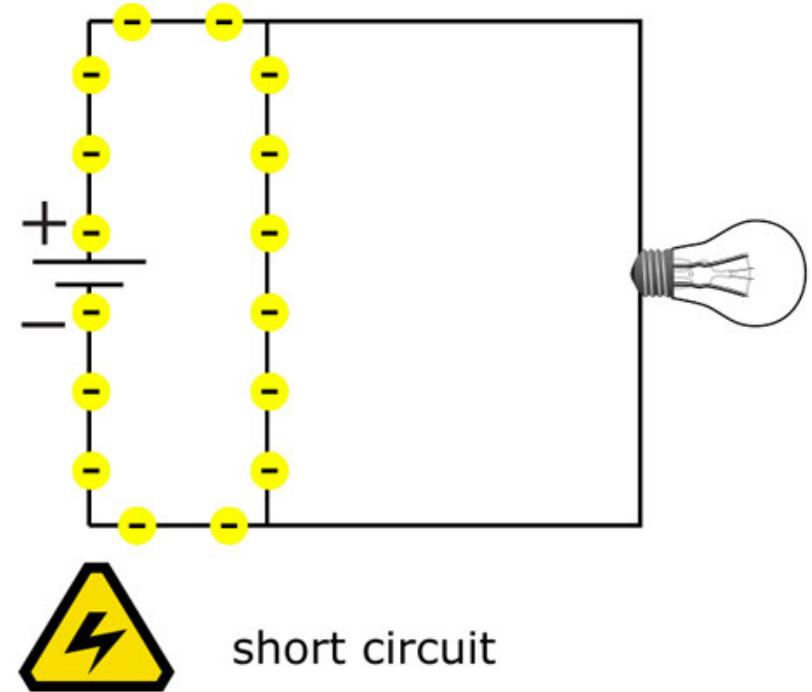
With a bit of practice you start to recognise all the components

Short Circuits

Good



Bad

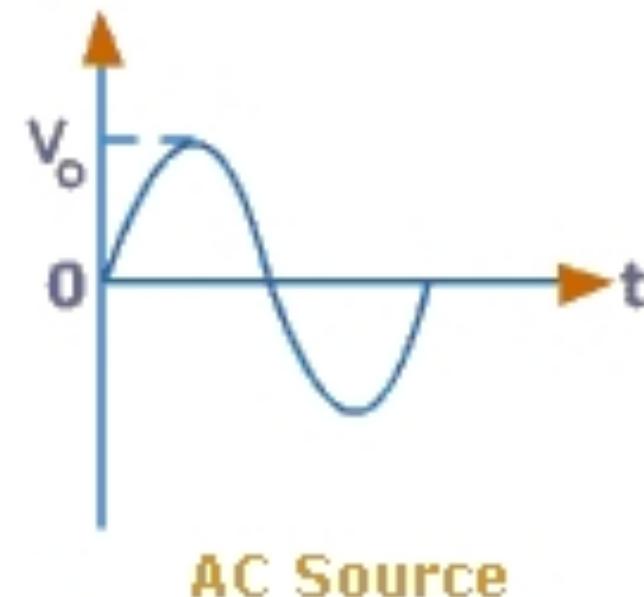
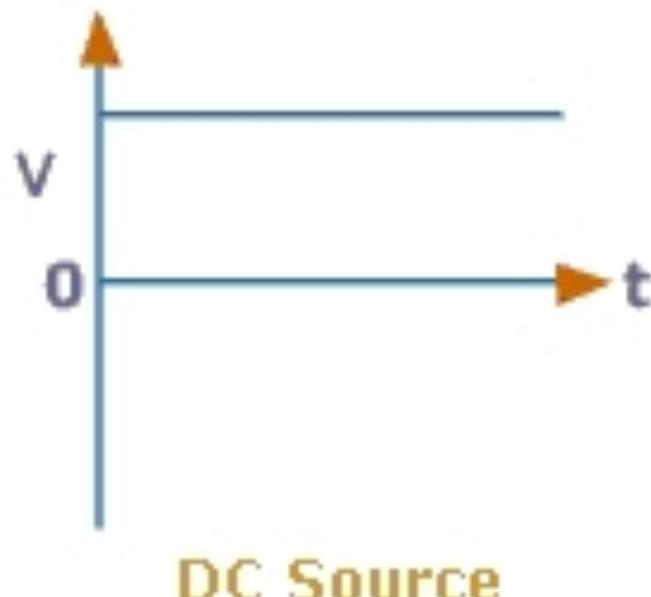


Regular circuit has wires connecting components in a loop, however if there is ever a case of a loop with **NO LOAD** then this is called a short circuit. (Load can be Motor, Light, ... etc.)

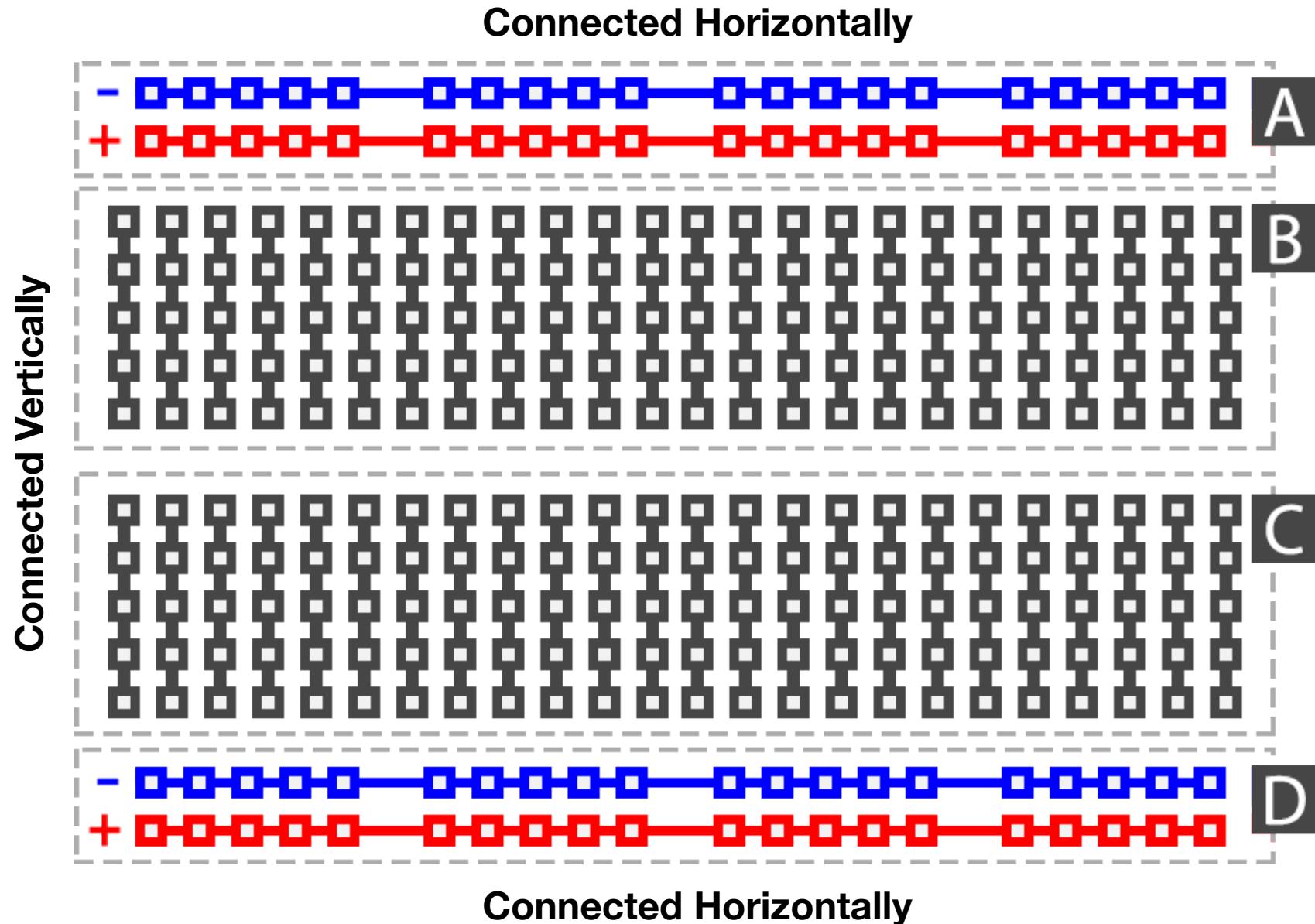
Watch out for this with practical work!

Types of electricity

- **AC** - Alternating current - Voltage supplied varies when supplied to electrical device. For mains and **high power** electronics.
- **DC** - Direct current - Voltage supply is held constant, You will be dealing mainly with **DC** electronics with voltages ranging from 0 - 12V.



How a solderless breadboard works



Exercise 1.

Digital Input - Any Digital Pin on the **Arduino** is capable of recognising a binary change in voltage **Eg:** A change in voltage from **0V - 5V**.

***Note** Any voltage on the input pin greater than **1.5V** will be recognised as a **Binary 1 or HIGH**, and anything below will be a **Binary 0 or LOW**.

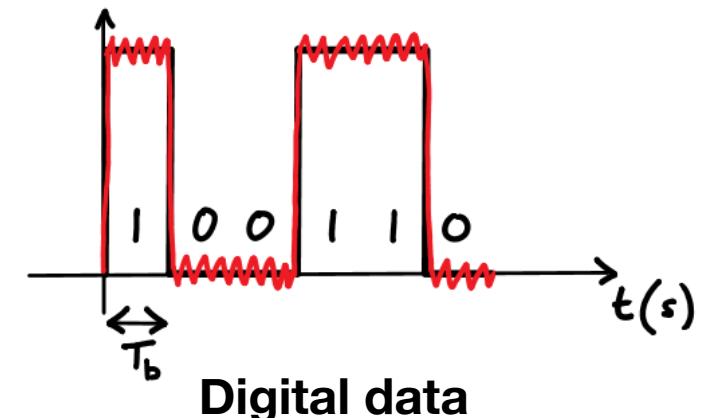
Examples of **digital** inputs.



Micro Switch

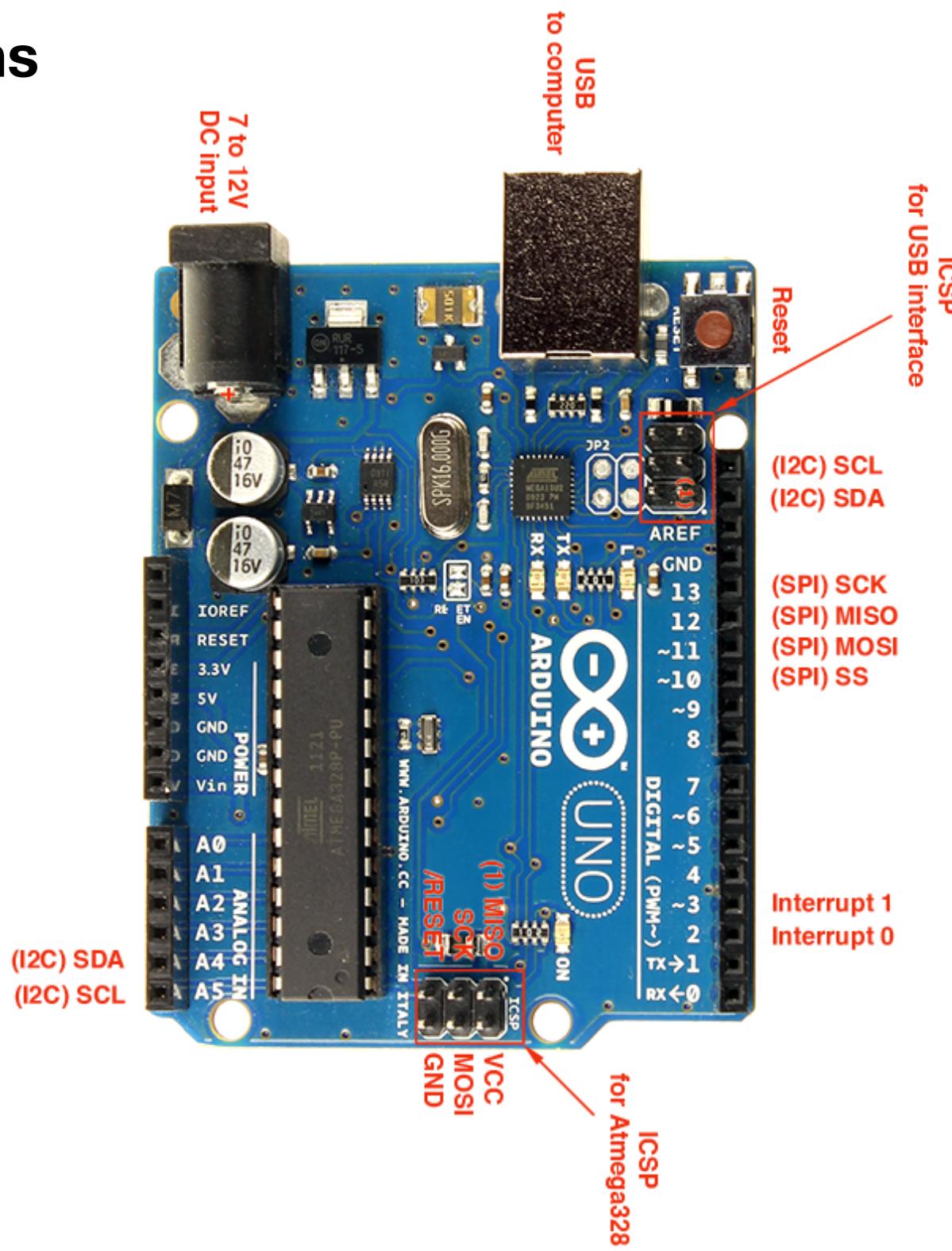


Push button



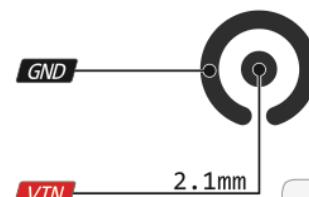
Digital data

Arduino Pins



UNO PINOUT

 7-12V Depending
on current drawn



**Absolute MAX per pin 40mA
recommended 20mA**

 Absolute MAX 200mA
for entire package

I_{OREF} provides a logic reference voltage for shields that use it. It is connected to the 5V bus.

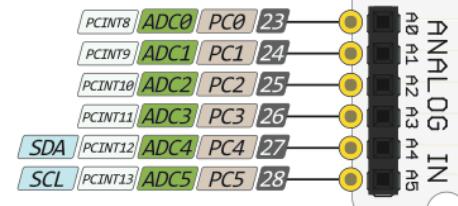
R3 O

PCINT14

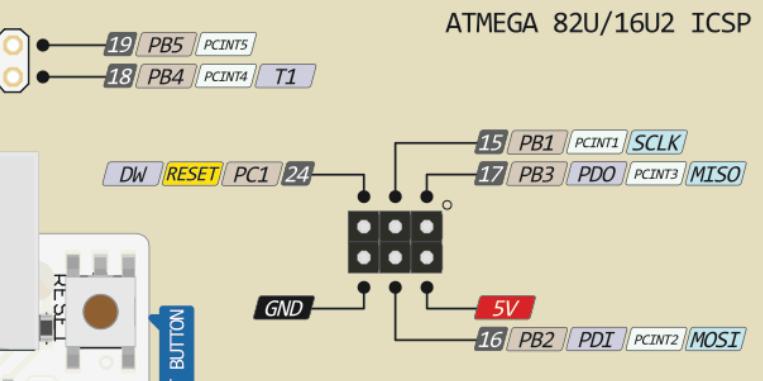
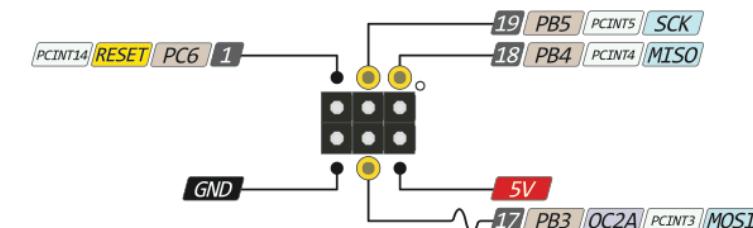
PCINT

1

The input voltage to the board when it is running from external power. Not USB bus power.



 The power sum for each pin's group should not exceed 100mA

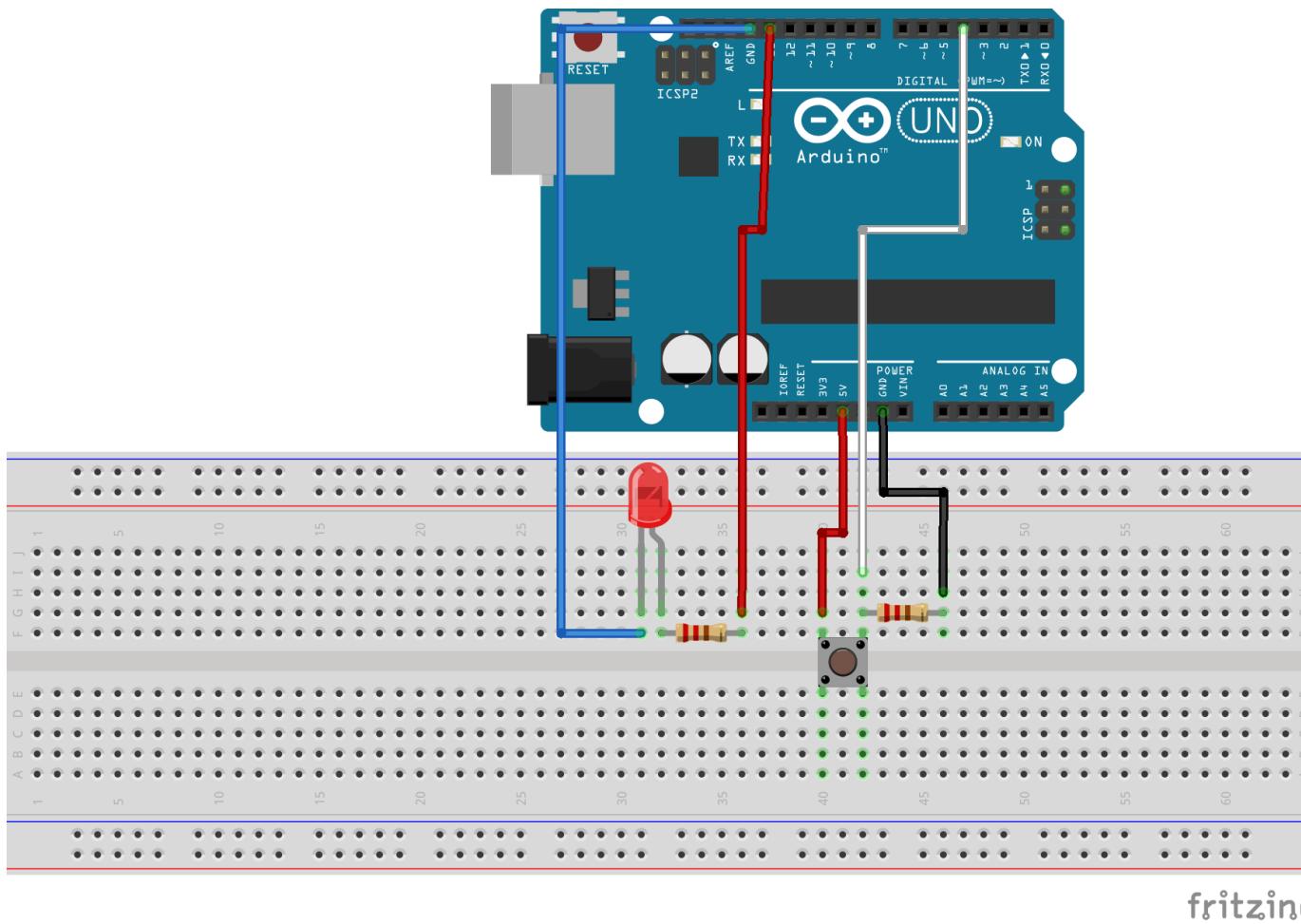


! R3 Only

- Power
- GND
- Serial Pin
- Analog Pin
- Control
- INT
- Physical Pin
- Port Pin
- Pin function
- Interrupt Pin
- PWM Pin
- Port Power

 Connected to the ATMega and used for USB program and communicating with it

The Circuit



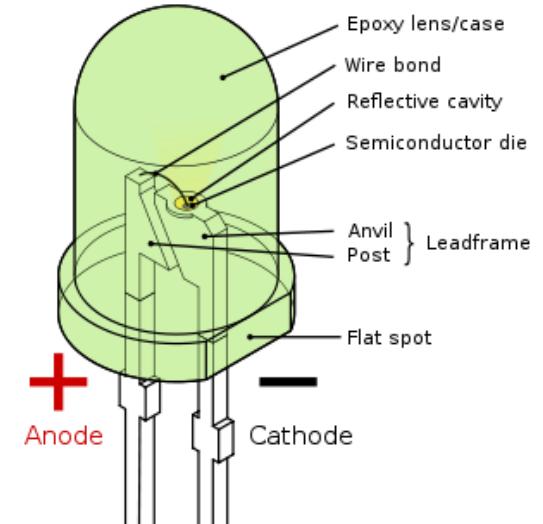
fritzing

Push button as **Digital input**, and LED as **Digital output**.

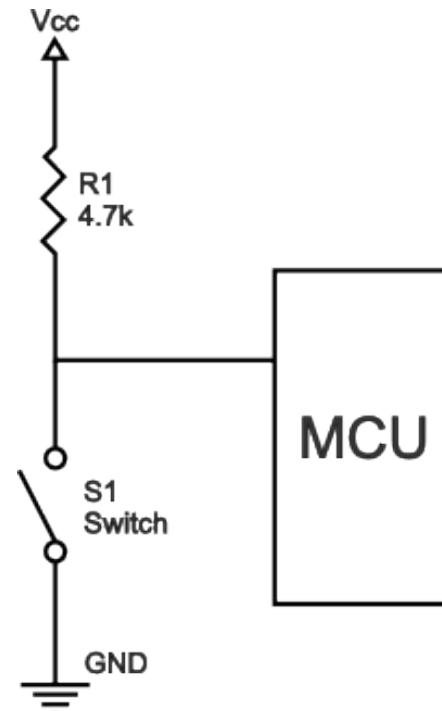
Does anyone know what the **resistors** are for?

*Anyone want to try the challenge?

Current limiting for LED - LED's will draw current until they explode if you do not have this resistor. Also you can't plug them in **Backward!**



Pull up resistor - Used to define the digital input value when the button is left as an open switch. If you were to leave the input pin "**floating**" the voltage can fluctuate and cause weird behaviours in your circuit.





Digital_Read_Example

```
//DfPI Tutorial Code  
//Sean Malikides  
//Oct 2017
```

```
/*  
Code Function Description
```

- Reads a digital input on pin 2, prints the result to the serial monitor.
- Uses push button to turn a light on and off.

```
*/
```

```
// declare variable "pushButton" as an integer, and assign it the value 2, for later pin assignment.
```

```
int pushButton = 2; 1.
```

```
// constants won't change. Used here to set a pin number.  
const int ledPin = 13; // the number of the LED pin 2.
```

```
// the setup routine runs once when you press reset.
```

```
void setup() { 3.
```

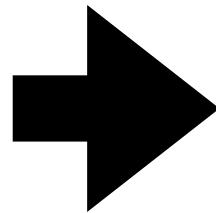
```
// initialize serial communication at 9600 bits per second.  
Serial.begin(9600);
```

```
// make the pushbutton's pin an input.  
pinMode(pushButton, INPUT);
```

```
// set the digital pin as output:  
pinMode(ledPin, OUTPUT);
```

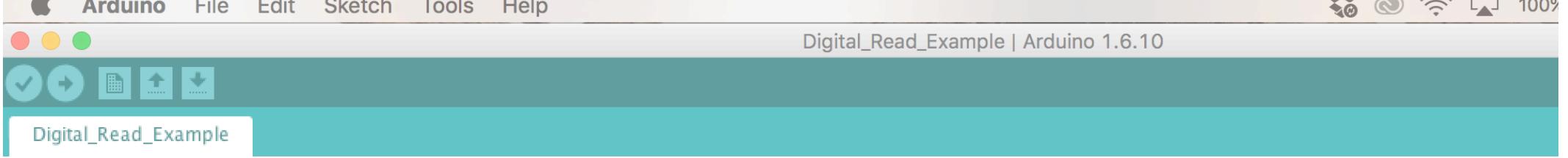
```
}
```

```
// the loop routine runs over and over again forever.
```



Computers **usually** read through your code sequentially.

Comments (For Humans)



```
// set the digital pin as output:  
pinMode(ledPin, OUTPUT);  
  
}  
  
// the loop routine runs over and over again forever.  
void loop() {  
    // read the input pin.  
    int buttonState = digitalRead(pushButton);  
  
    // print out the state of the button, So you can see what the state of the pushbutton is.  
    Serial.println(buttonState);  
  
    if (buttonState == HIGH) {  
        //Turn the LED on.  
        digitalWrite(ledPin, HIGH);  
  
    } else {  
  
        //Turn the LED off.  
        digitalWrite(ledPin, LOW);  
  
    }  
  
    //just a little delay for stability.  
    delay(10);  
}
```

This Loop section of code runs forever!

If statement like a logical switch.

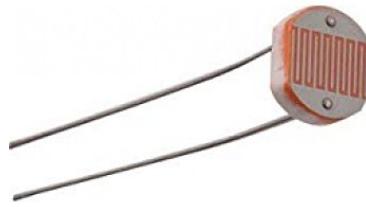
Wait for a little bit, Delay(t) where " t " is in milliseconds.

Exercise 2.

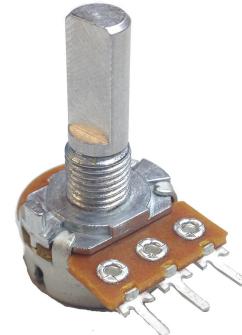
Analog Input - Any **Analog Pin** on the **Arduino** is capable of registering a Voltage variation within the range of **0V - 5V** with a resolution of 12 bits (ie. 0 - 1023 integer range).

Analog output - On the Arduino **Uno** you cannot generate a pure analog output. Instead you fake it, by using **PWM**.

Examples of **Analog** inputs.



Light Dependant Resistor



Potentiometer



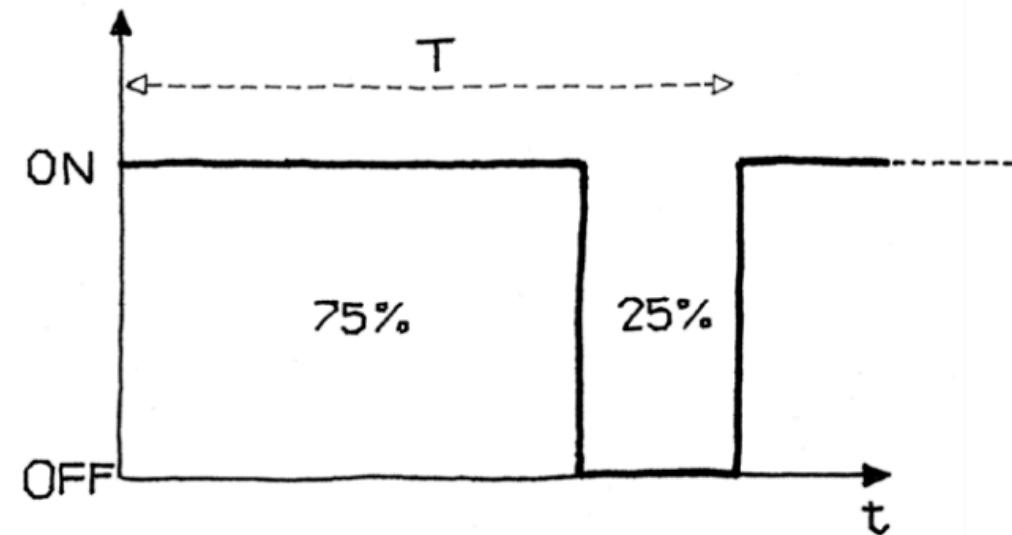
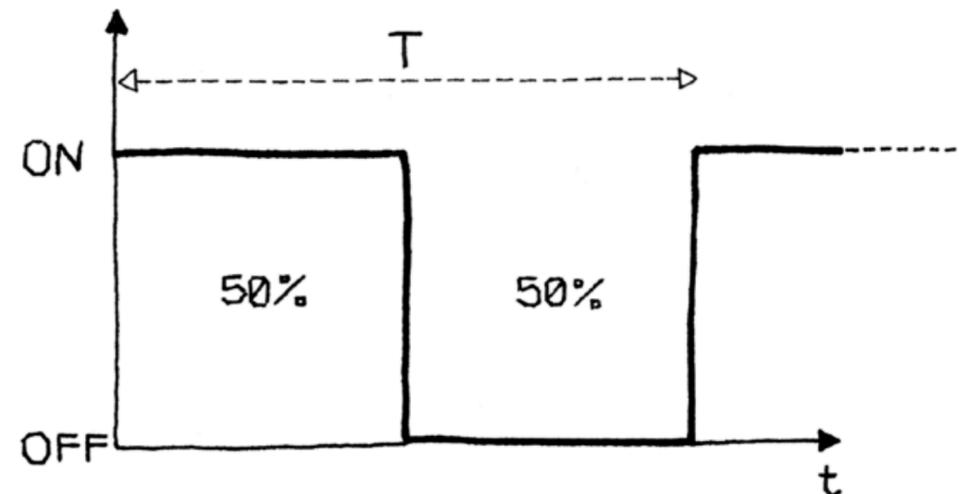
Microphone

PWM - Pulse Width Modulation

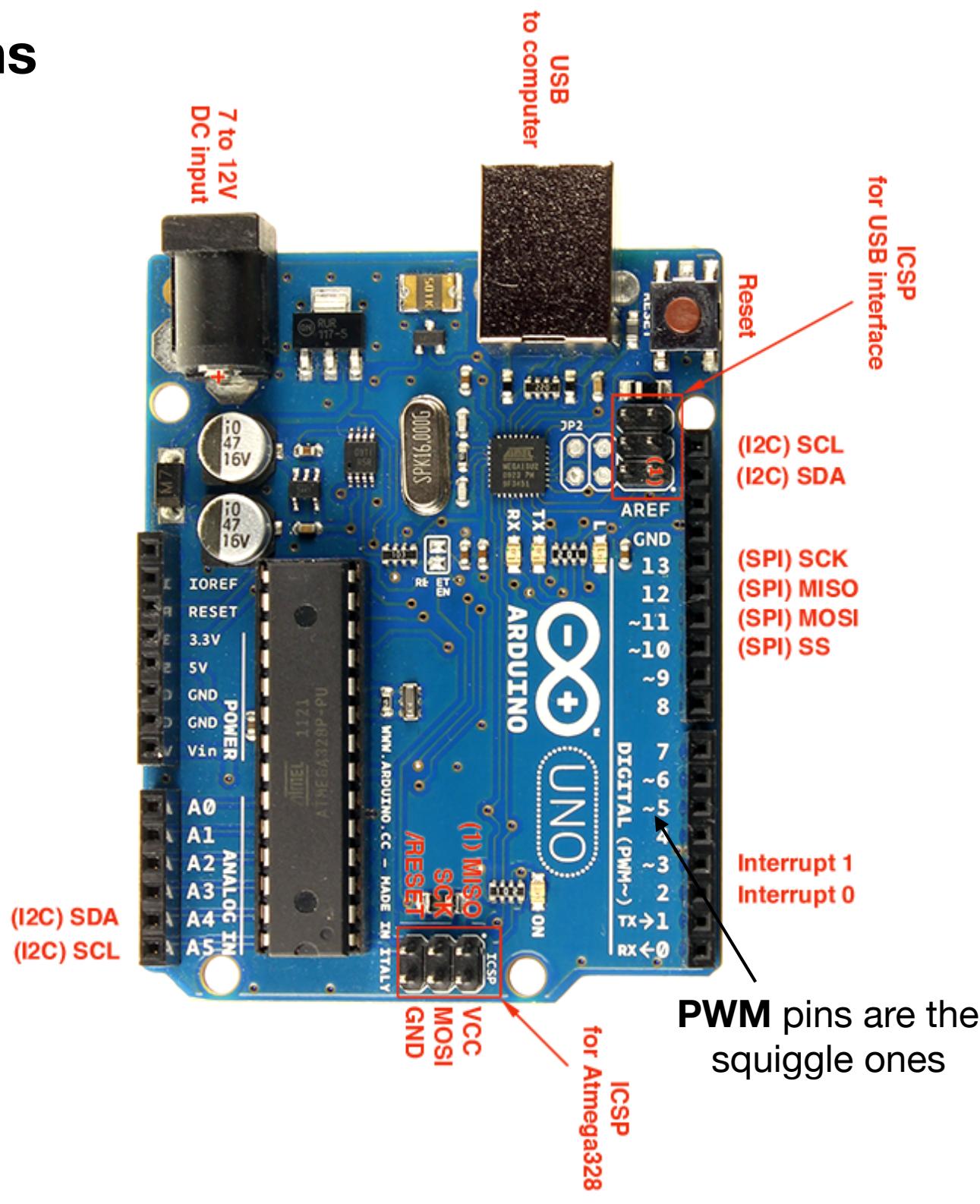
Effectively the **average voltage** over time is dependant on the OnTime:Offtime ratio of the signal.

This effect is obvious when used on something like an **LED**, where you can dim the output brightness of the **LED** by changing the PWM **“Duty Cycle”**.

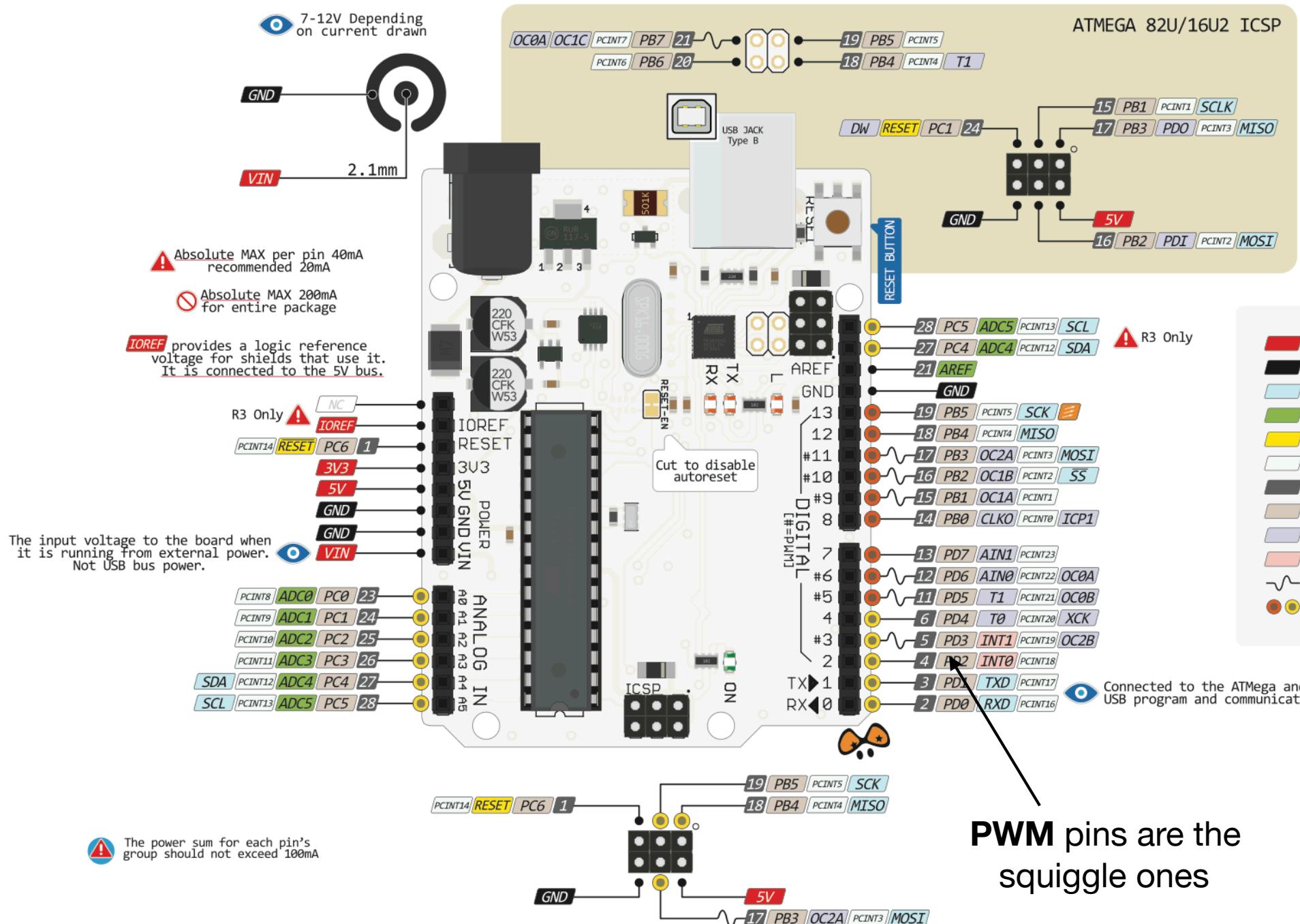
Only specific pins support PWM on the Arduino Uno.



Arduino Pins

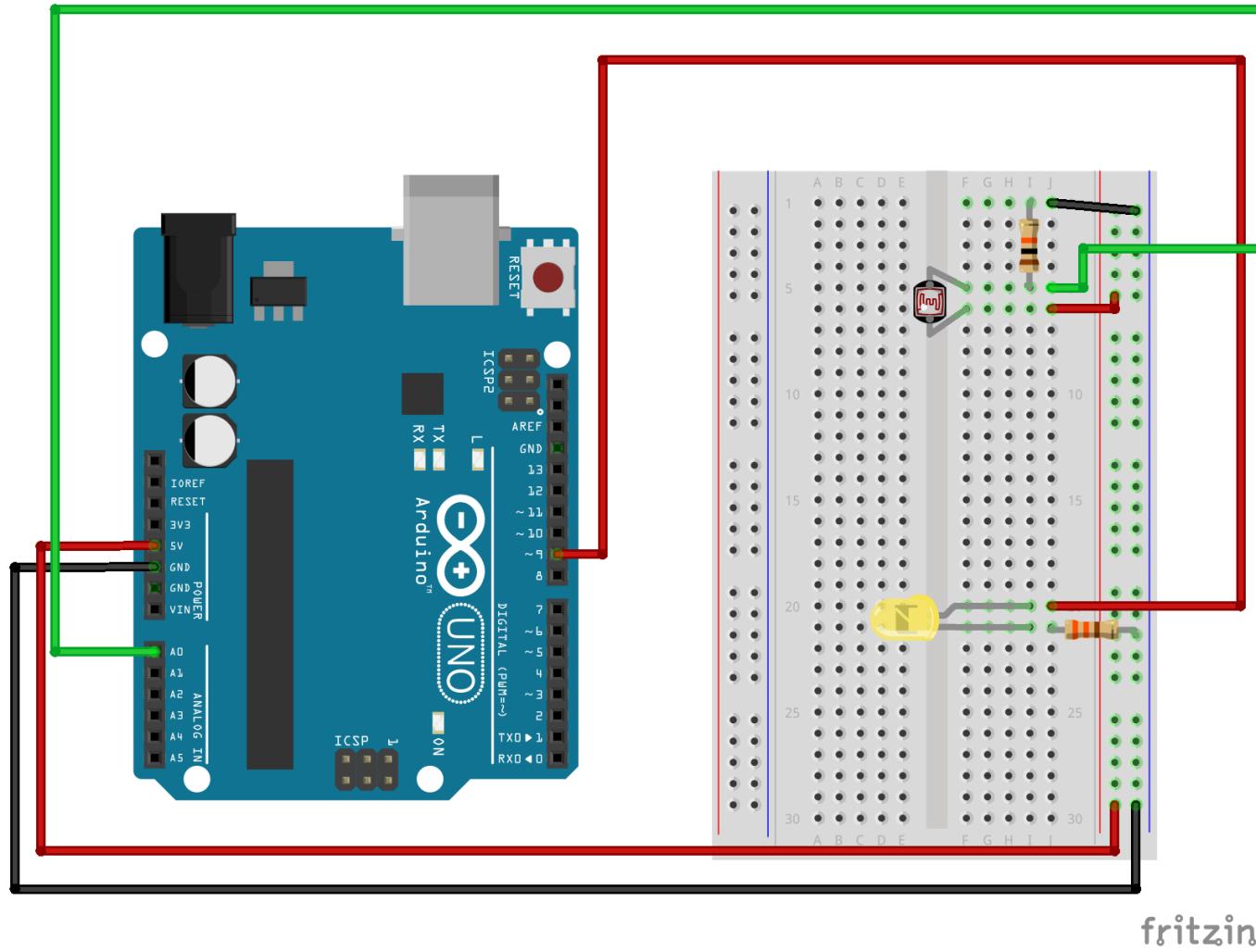


UNO PINOUT



PWM pins are the squiggle ones

The Circuit



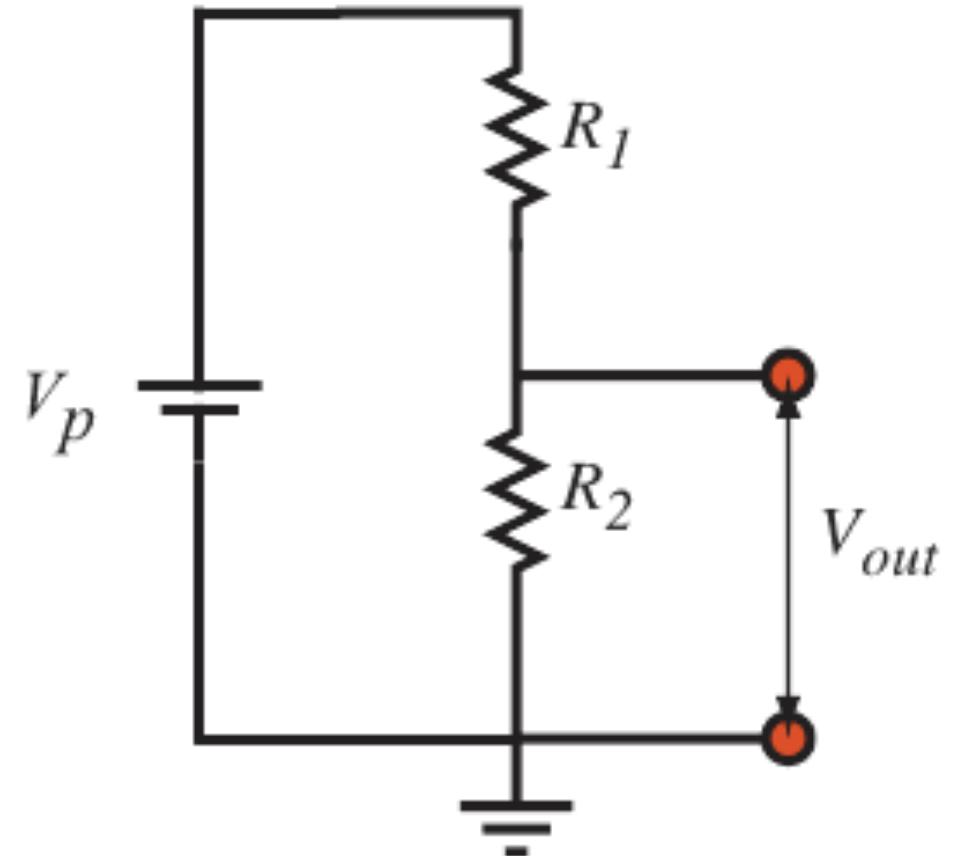
Light dependant resistor, LDR as **Analog input**, and LED as **Analog output**.

Does anyone know what the **resistors** are for?

Voltage divider - Super important concept

Often used to convert a sensor value that varies in resistance into a change in Voltage, such as in our case with the **Light Dependant Resistor**.

*The circuit we are using is essentially the same just replacing one of the resistors with an **LDR**.



$$V_{out} = V_{in} \cdot \frac{R_2}{R_1 + R_2}$$



Analog_Read_Example

```
//setting output pin for LED, ensuring PWM pin.  
const int ledPin = 6;  
  
// the setup routine runs once when you press reset:  
void setup() {  
  
    // initialize serial communication at 9600 bits per second:  
    Serial.begin(9600);  
  
    // set the digital pin as output:  
    pinMode(ledPin, OUTPUT);  
  
}  
  
// the loop routine runs over and over again forever:  
void loop() {  
  
    // read the input on analog pin 0:  
    int sensorValue = analogRead(A0);  
  
    // print out the value you read:  
    Serial.println(sensorValue);  
  
    //map sensor value to a value between 0 and 255.  
    int brightness = map(sensorValue, 0,1023,0,255);  
  
    //outputting brightness with PWM, must be a value between 0 and 255.  
    analogWrite(ledPin,brightness);  
    delay(10);      // delay in between reads for stability  
  
}
```

Read the sensor voltage

map() function is like a linear interpolation function.

PWM output

Done Saving.

Invalid library found in /Users/seanmalikides/Documents/Arduino/libraries/jeroendoggen-Arduino-signal-filtering-library-ef8b188: /Users/seanmalik
Invalid library found in /Users/seanmalikides/Documents/Arduino/libraries/jeroendoggen-Arduino-signal-filtering-library-ef8b188: /Users/seanmalik

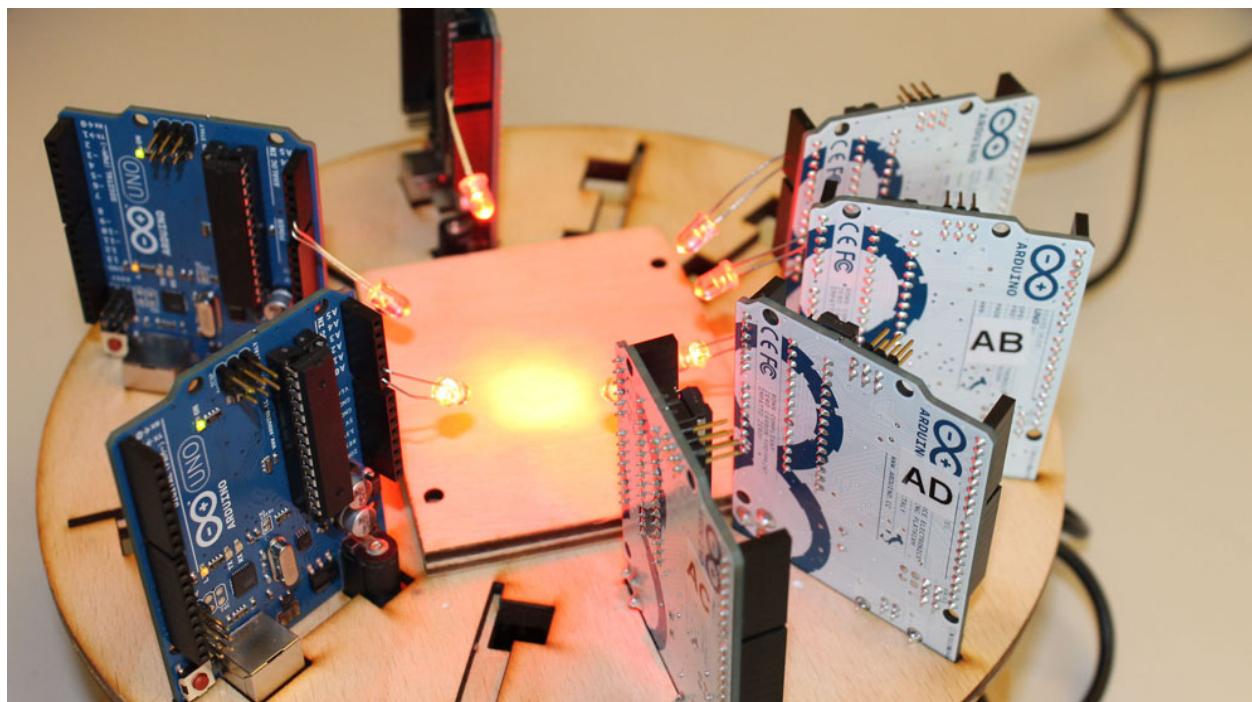
A black and white photograph capturing a vast, dense field of numerous white origami cranes. These cranes are arranged in rows, creating a rhythmic pattern of peaks and valleys. They are illuminated from within, casting a soft glow that highlights their intricate folds and the tops of their pointed wings. The background is dark, making the bright, glowing cranes stand out sharply. The perspective is from a low angle, looking up at the cranes, which emphasizes their collective height and the scale of the installation.

PanGenerator

Challenge

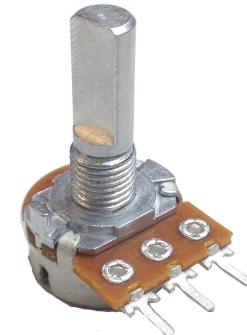
With the knowledge from the previous 2 examples attempt to get your Arduino **respond** to your partners Arduino with **Light!** Like the modules in the video.

We would like to put all the modules in a swarm together at the end and get them all to speak to each other.

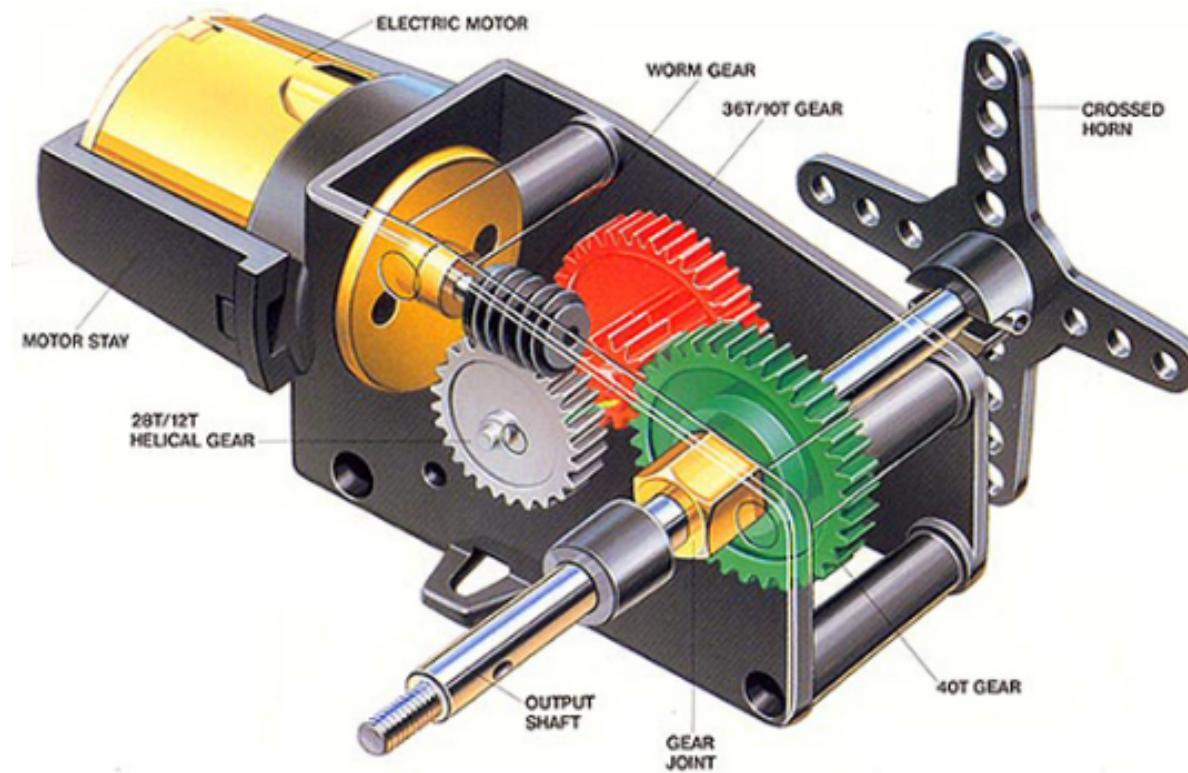


Add features

- Try adding multiple light sensors so you can sense a light pulse from multiple directions. You should be able to simply use the circuit from the previous examples.
- Try adding a potentiometer to enable you to adjust the light threshold for which your device responds to light.



Gearbox Motors - Giving your projects motion

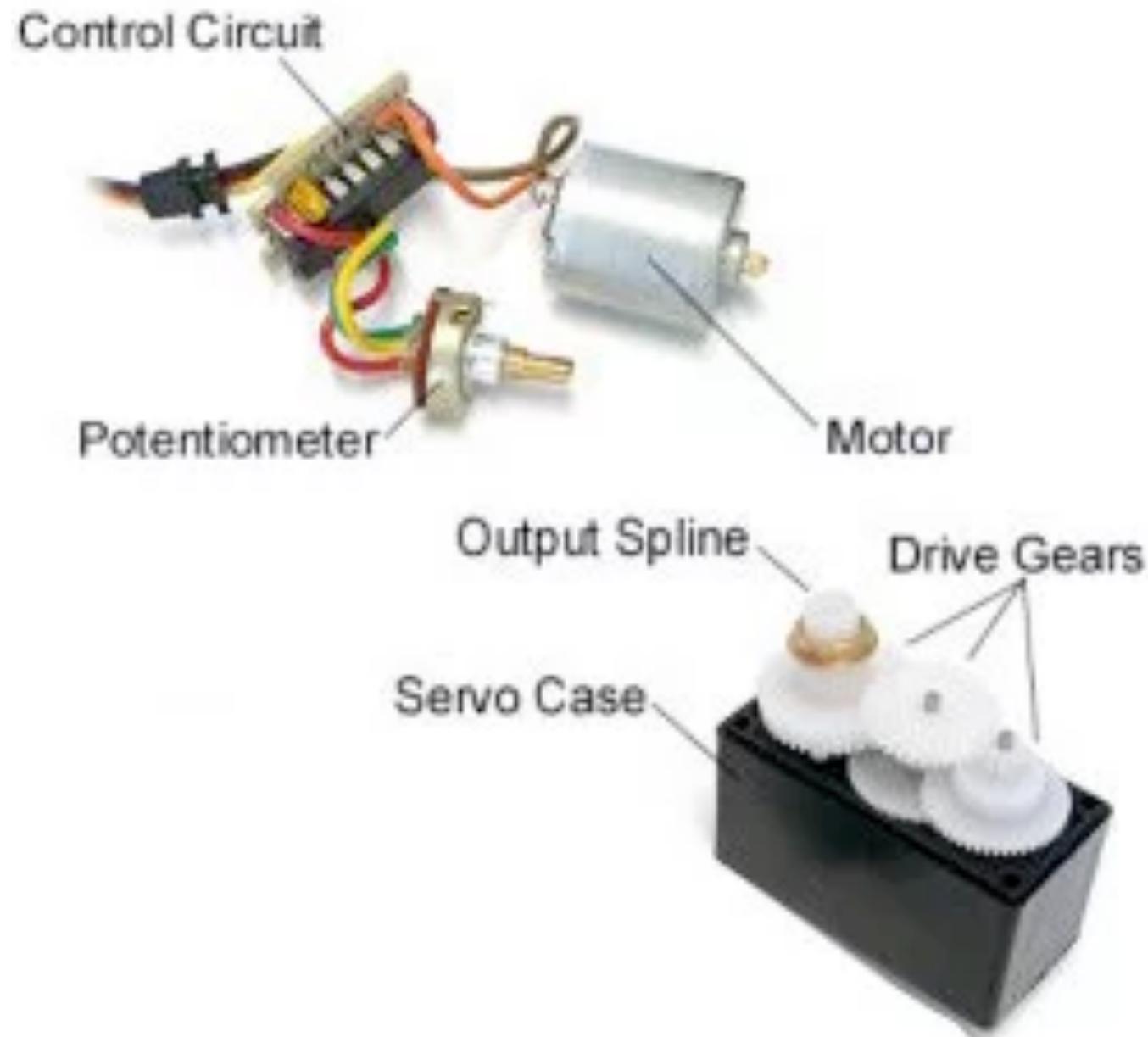


- Super easy to use.
- Can be super strong!

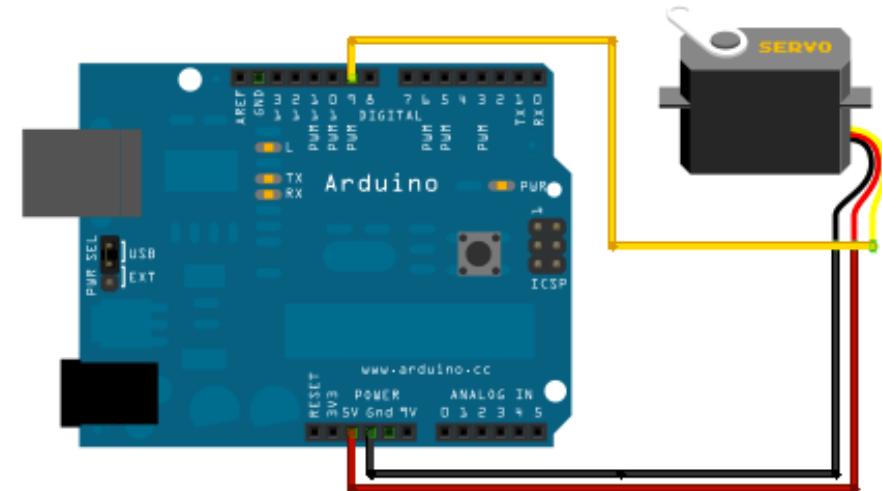
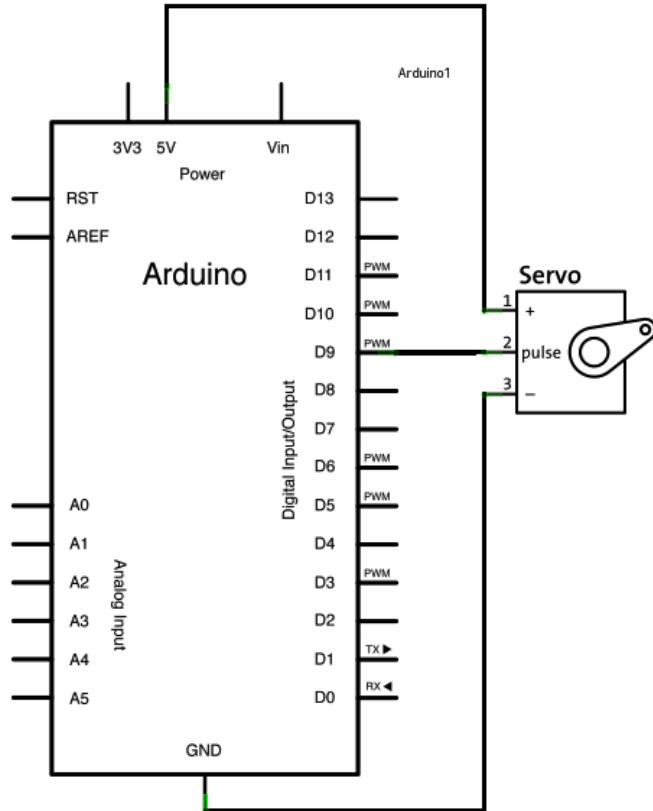
“Hobby” Servo Motors – We will work with these



- Super easy to use.
- Provide smooth accurate motion.
- Good for making small things move

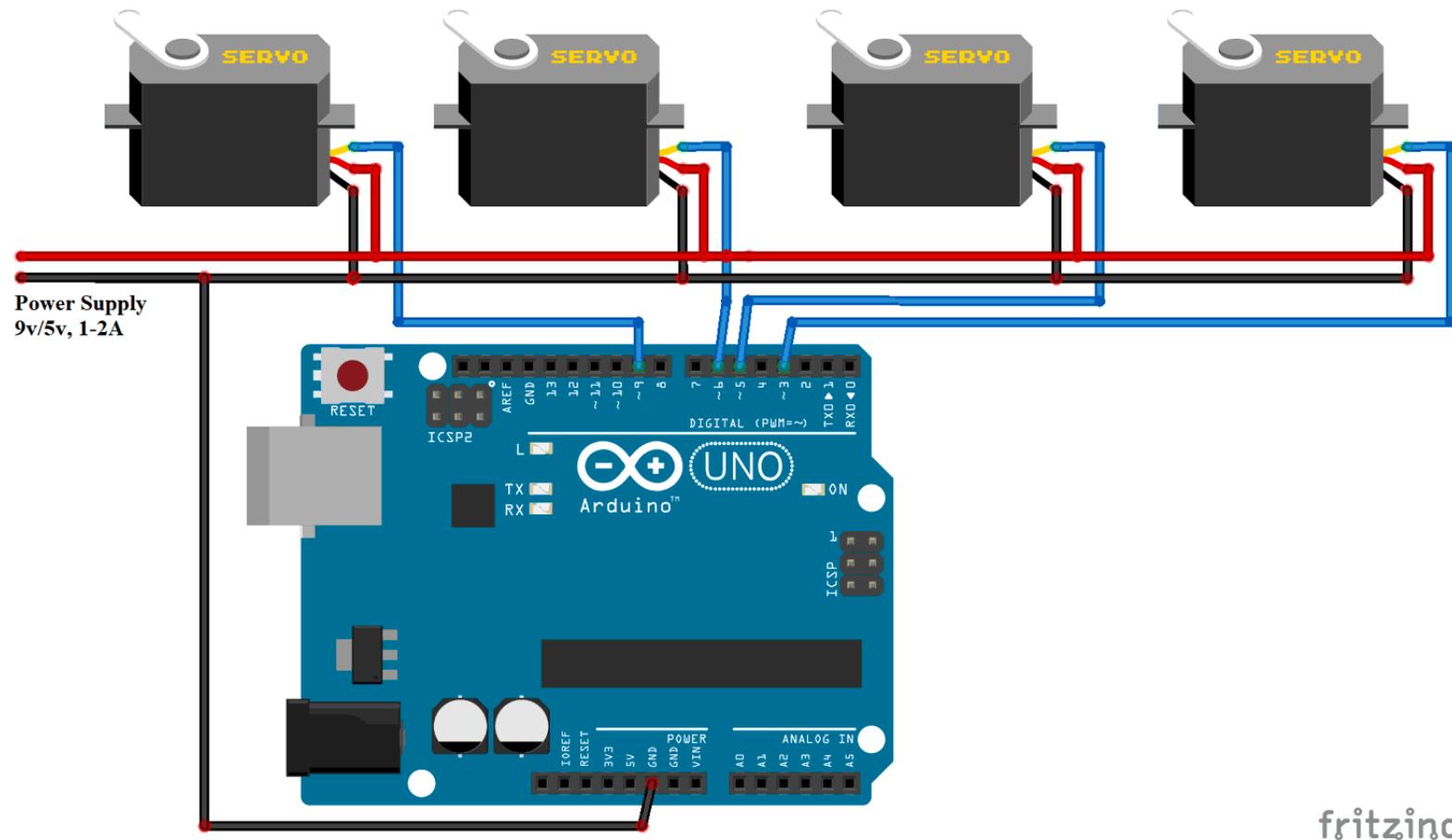


Control Circuit



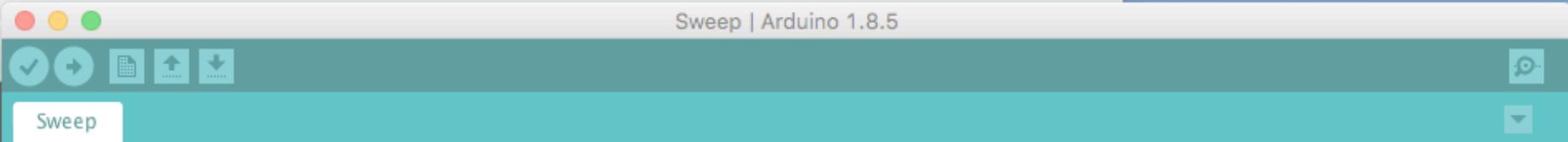
With Arduino you only need to use a **single** pin for control, you do however need to power the servo from an extra power supply. **WHY IS THIS?**

Multiple Servo Control Circuit



fritzing

With Arduino you only need to use a **single** pin for control, you do however need to power the servo from an extra power supply. **WHY IS THIS?**



```
#include <Servo.h> // use a Servo Library which simplifies servo control

Servo myservo; // create servo object to control a servo
// twelve servo objects can be created on most boards

int pos = 0; // variable to store the servo position

void setup() {
    myservo.attach(9); // attaches the servo on pin 9 to the servo object
}

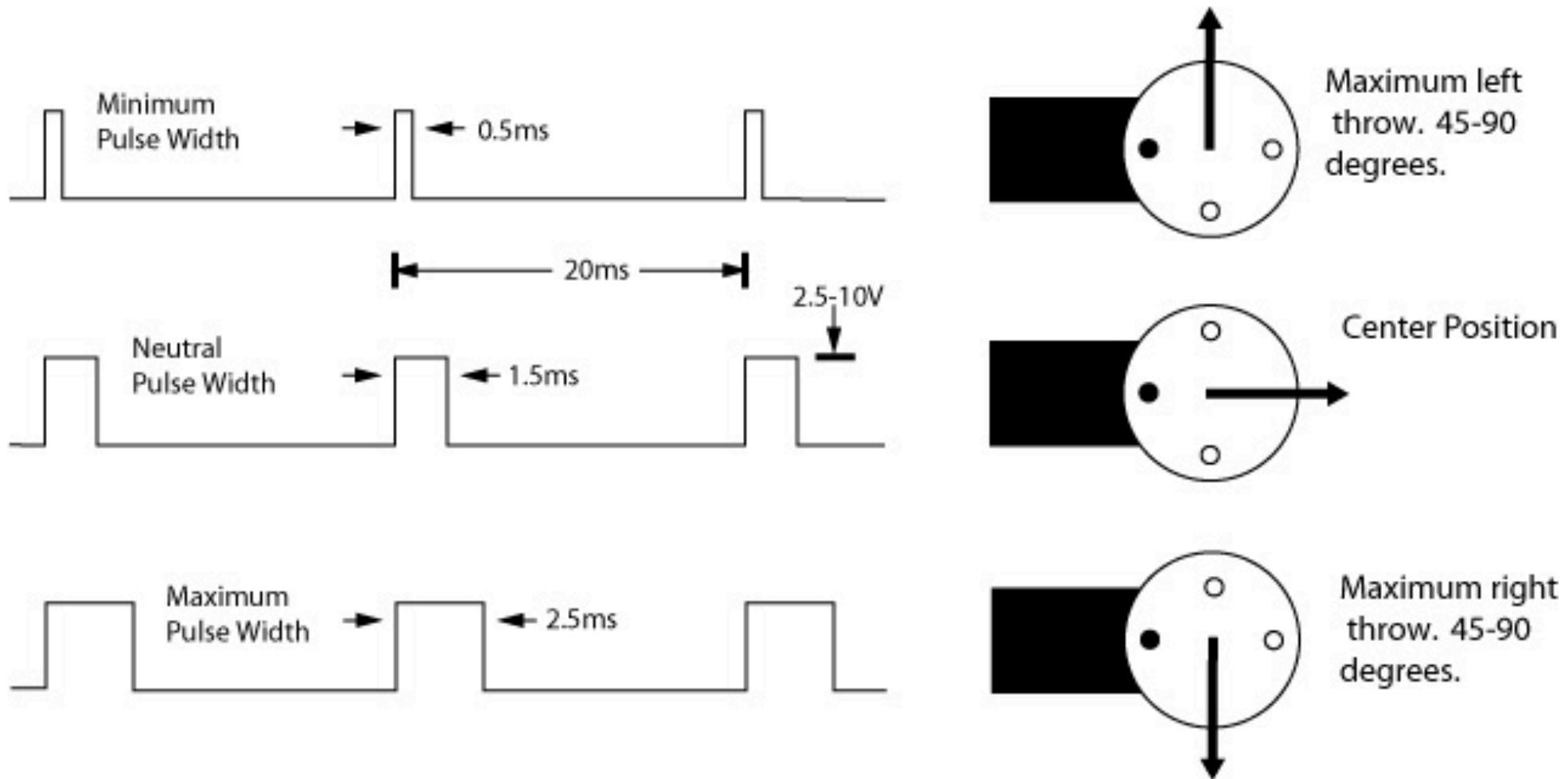
void loop() {
    for (pos = 0; pos <= 180; pos += 1) { // goes from 0 degrees to 180 degrees
        // in steps of 1 degree
        myservo.write(pos); // tell servo to go to position in variable 'pos'
        delay(15); // waits 15ms for the servo to reach the position
    }
    for (pos = 180; pos >= 0; pos -= 1) { // goes from 180 degrees to 0 degrees
        myservo.write(pos); // tell servo to go to position in variable 'pos'
        delay(15); // waits 15ms for the servo to reach the position
    }
}
```

Function to initialise servo

Function to set servo angle, 0 - 180 degrees

Done Saving.

R/C Control Signal Theory



The minimum and maximum pulse width for different manufacturers can vary considerably; however, the neutral position is generally quite near 1.5ms regardless of manufacturer. Typical variance for the minimum pulse width is from 0.5ms to 0.8ms, and the typical variance for the maximum pulse width is from 2.5ms to 3.0ms. The frequency of the signal is generally near 50Hz; however, it can range from 30Hz to 200Hz. The output voltage can vary from 2.5V to as much as 10V.



homemadeServoLibrary0

```
// Servo Control without library

int servo = 9;// set servo pin

void setup() {
  pinMode(servo, OUTPUT); // set pin as output
}

void loop() {
  servoPulse(servo, 1500); // continuously pulse 1500 = 90 degrees
  //servoPulse(servo,500); // continuously pulse 500 = 0 degrees
  //servoPulse(servo,2500); // continuously pulse 2500 = 180 degrees
  delay(20);
}

void servoPulse(int servoPin, int pulse) {
  digitalWrite(servoPin, HIGH);
  delayMicroseconds(pulse);
  digitalWrite(servoPin, LOW);
}
```

Why can you can only pulse one of these at a time? Because the Servos need time to get to position

Our own servoPulse Function

Done Saving.

/Users/ruairiglynn/Desktop/Sweep/Sweep.ino

homemadeServoLibrary0 | Arduino 1.8.8

```
// Servo Control without library

int servo = 9;// set servo pin
int pos = 500;

void setup() {
  pinMode(servo, OUTPUT); // set pin as output
}

void loop() {
  for (pos = 500; pos <= 2500; pos += 1) { // goes from 0 degrees to 180 degrees
    // in steps of 1 degree
    servoPulse(servo, pos);           // tell servo to go to position in variable 'pos'
    delay(15);                      // waits 15ms for the servo to reach the position
  }
  for (pos = 2500; pos >= 500; pos -= 1) { // goes from 180 degrees to 0 degrees - using pulse vals
    servoPulse(servo, pos);           // tell servo to go to position in variable 'pos'
    delay(15);                      // waits 15ms for the servo to reach the position
  }
}

void servoPulse(int servoPin, int pulse) {
  digitalWrite(servoPin, HIGH);
  delayMicroseconds(pulse);
  digitalWrite(servoPin, LOW);
}
```

Servo Sweep without Library

Done compiling.

Sketch uses 1100 bytes (3%) of program storage space. Maximum is 32256 bytes.
Global variables use 11 bytes (0%) of dynamic memory, leaving 2037 bytes for local variables. Maximum is

Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr		Dec	Hx	Oct	Html	Chr		Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040	 	Space		64	40	100	@	Ø		96	60	140	`	`
1	1	001	SOH	(start of heading)	33	21	041	!	!		65	41	101	A	A		97	61	141	a	a
2	2	002	STX	(start of text)	34	22	042	"	"		66	42	102	B	B		98	62	142	b	b
3	3	003	ETX	(end of text)	35	23	043	#	#		67	43	103	C	C		99	63	143	c	c
4	4	004	EOT	(end of transmission)	36	24	044	$	\$		68	44	104	D	D		100	64	144	d	d
5	5	005	ENQ	(enquiry)	37	25	045	%	%		69	45	105	E	E		101	65	145	e	e
6	6	006	ACK	(acknowledge)	38	26	046	&	&		70	46	106	F	F		102	66	146	f	f
7	7	007	BEL	(bell)	39	27	047	'	'		71	47	107	G	G		103	67	147	g	g
8	8	010	BS	(backspace)	40	28	050	((72	48	110	H	H		104	68	150	h	h
9	9	011	TAB	(horizontal tab)	41	29	051))		73	49	111	I	I		105	69	151	i	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	*	*		74	4A	112	J	J		106	6A	152	j	j
11	B	013	VT	(vertical tab)	43	2B	053	+	+		75	4B	113	K	K		107	6B	153	k	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	,	,		76	4C	114	L	L		108	6C	154	l	l
13	D	015	CR	(carriage return)	45	2D	055	-	-		77	4D	115	M	M		109	6D	155	m	m
14	E	016	SO	(shift out)	46	2E	056	.	.		78	4E	116	N	N		110	6E	156	n	n
15	F	017	SI	(shift in)	47	2F	057	/	/		79	4F	117	O	O		111	6F	157	o	o
16	10	020	DLE	(data link escape)	48	30	060	0	0		80	50	120	P	P		112	70	160	p	p
17	11	021	DCL	(device control 1)	49	31	061	1	1		81	51	121	Q	Q		113	71	161	q	q
18	12	022	DC2	(device control 2)	50	32	062	2	2		82	52	122	R	R		114	72	162	r	r
19	13	023	DC3	(device control 3)	51	33	063	3	3		83	53	123	S	S		115	73	163	s	s
20	14	024	DC4	(device control 4)	52	34	064	4	4		84	54	124	T	T		116	74	164	t	t
21	15	025	NAK	(negative acknowledge)	53	35	065	5	5		85	55	125	U	U		117	75	165	u	u
22	16	026	SYN	(synchronous idle)	54	36	066	6	6		86	56	126	V	V		118	76	166	v	v
23	17	027	ETB	(end of trans. block)	55	37	067	7	7		87	57	127	W	W		119	77	167	w	w
24	18	030	CAN	(cancel)	56	38	070	8	8		88	58	130	X	X		120	78	170	x	x
25	19	031	EM	(end of medium)	57	39	071	9	9		89	59	131	Y	Y		121	79	171	y	y
26	1A	032	SUB	(substitute)	58	3A	072	:	:		90	5A	132	Z	Z		122	7A	172	z	z
27	1B	033	ESC	(escape)	59	3B	073	;	:		91	5B	133	[[123	7B	173	{	{
28	1C	034	FS	(file separator)	60	3C	074	<	<		92	5C	134	\	\		124	7C	174	|	
29	1D	035	GS	(group separator)	61	3D	075	=	=		93	5D	135]]		125	7D	175	}	}
30	1E	036	RS	(record separator)	62	3E	076	>	>		94	5E	136	^	^		126	7E	176	~	~
31	1F	037	US	(unit separator)	63	3F	077	?	?		95	5F	137	_	_		127	7F	177		DEL