

# Internet of Things

Module 2: Devices

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

# You Need to Build...

- Flyover drone to analyze crops
- Generator defect detector
- Intelligent smartwatch
- Shopper gaze monitor

# How do you build IoT things?

- IoT is a synthesis of hardware and software
  - Software implementation plus a collection of attached circuits
- Not enough to understand Arduino programming
  - Good designers know the system they are building on
  - Underlying algorithms, protocols, architectures
  - External interfaces, APIs, components

# Roadmap

1. IoT Hardware
  - Components, sensors/actuators, reference circuits, integrated circuits
  - Memory, microcontrollers, storage
2. IoT Platforms
  - Arduino, Raspberry PI, Amazon FreeRTOS, VxWorks, RIOT
  - Designs, architectures, and tradeoffs
3. Deep Dive: Arduino Programming
  - Development workflow, language, API, coding examples
4. Case Studies
  - Deployment scenarios: cars, aircraft, buildings, retail

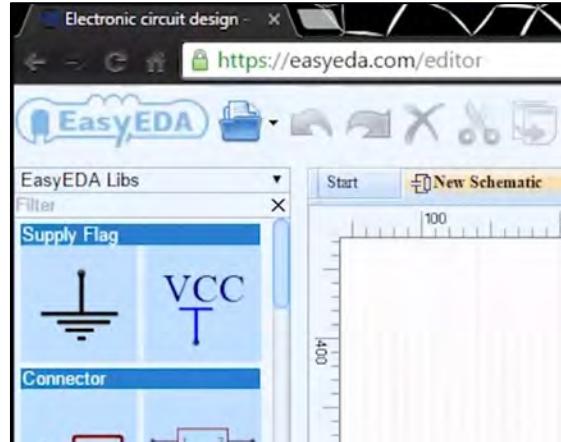
# IoT Hardware

# We Are Going through a Hardware Revolution

- Low-cost, componentized hardware enables rapid prototyping without advanced electrical engineering knowledge



***Componentized Circuits***



***Cloud-Based Fabrication***



***Open-Source Schematics***

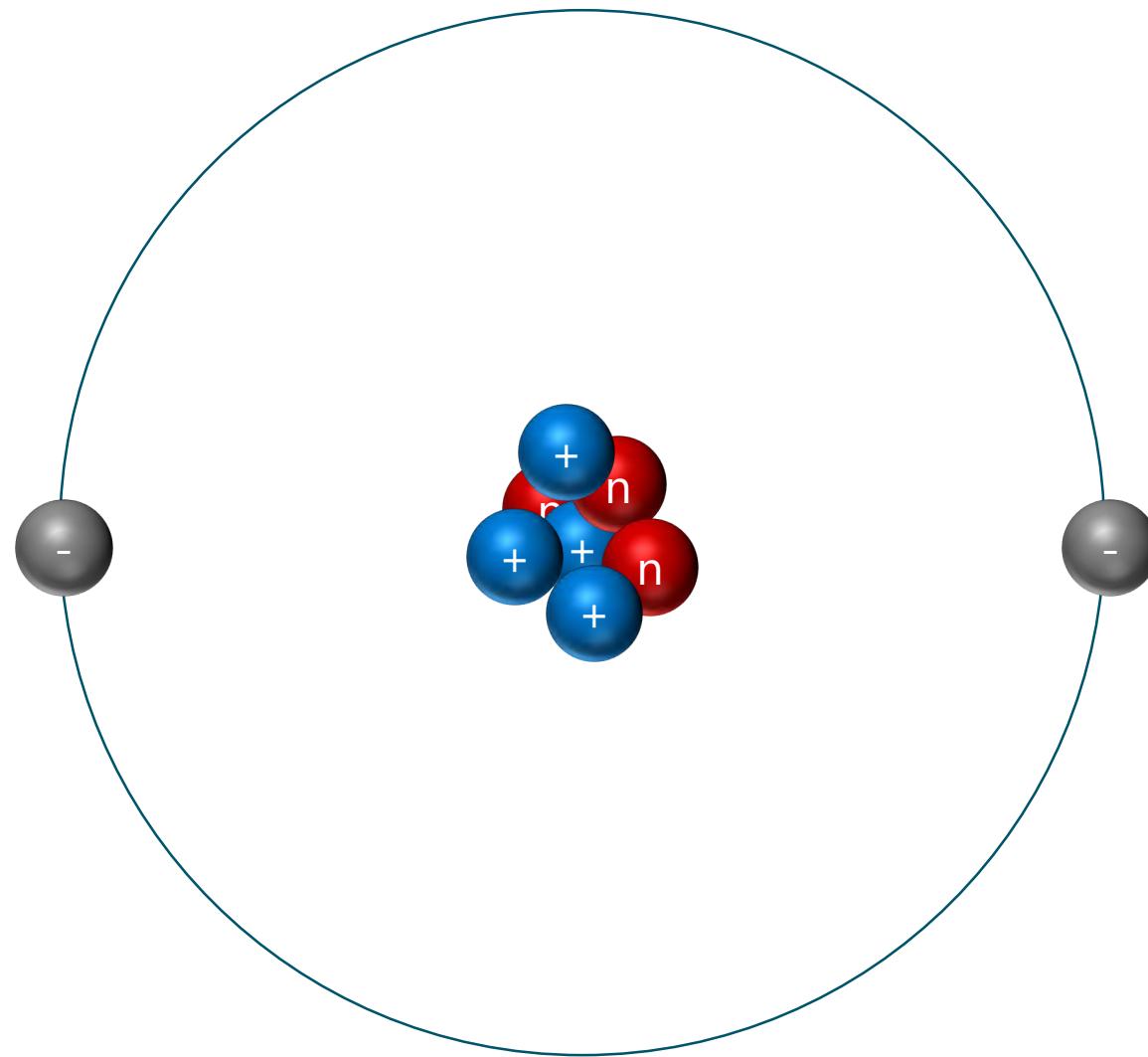
- This lecture: What You Need to Know to Build IoT Hardware Systems
  - Electricity Fundamentals, IoT Circuits, Microcontroller Platforms

SAP REV.C02  
LI145N0130



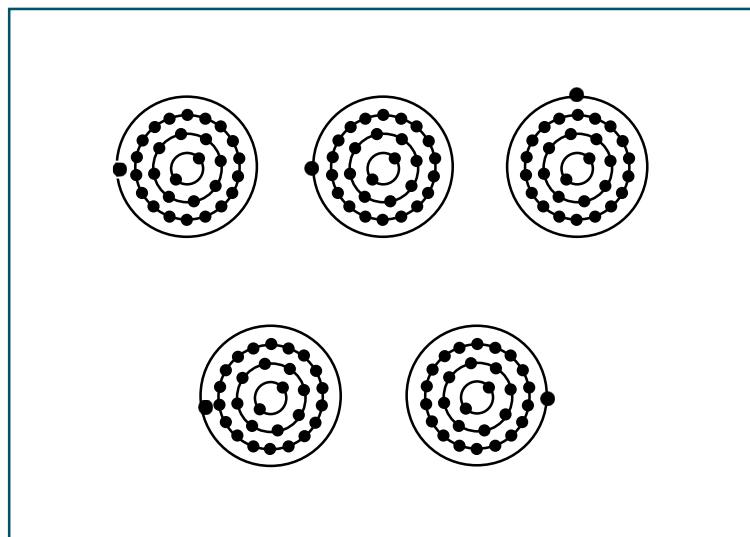
# Background: Atoms

- Made of **protons**, **neutrons**, and electrons
- In materials, nucleus held in fixed position
- Some electrons are also fixed
- But some electrons may be free to “move”

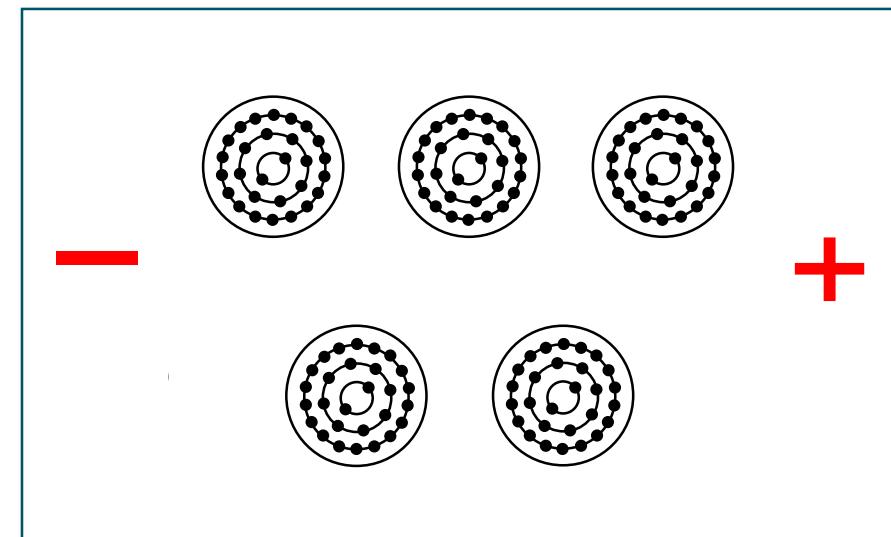


# Background: Electrical Current

- Usually free electrons hop around randomly
- However, outside forces can encourage them to flow in a particular direction
  - Magnetic field, charge differential → this is called **current**
  - We can vary properties of current to transmit information (via waves, like dominos, as electron **drift velocities** are very slow)

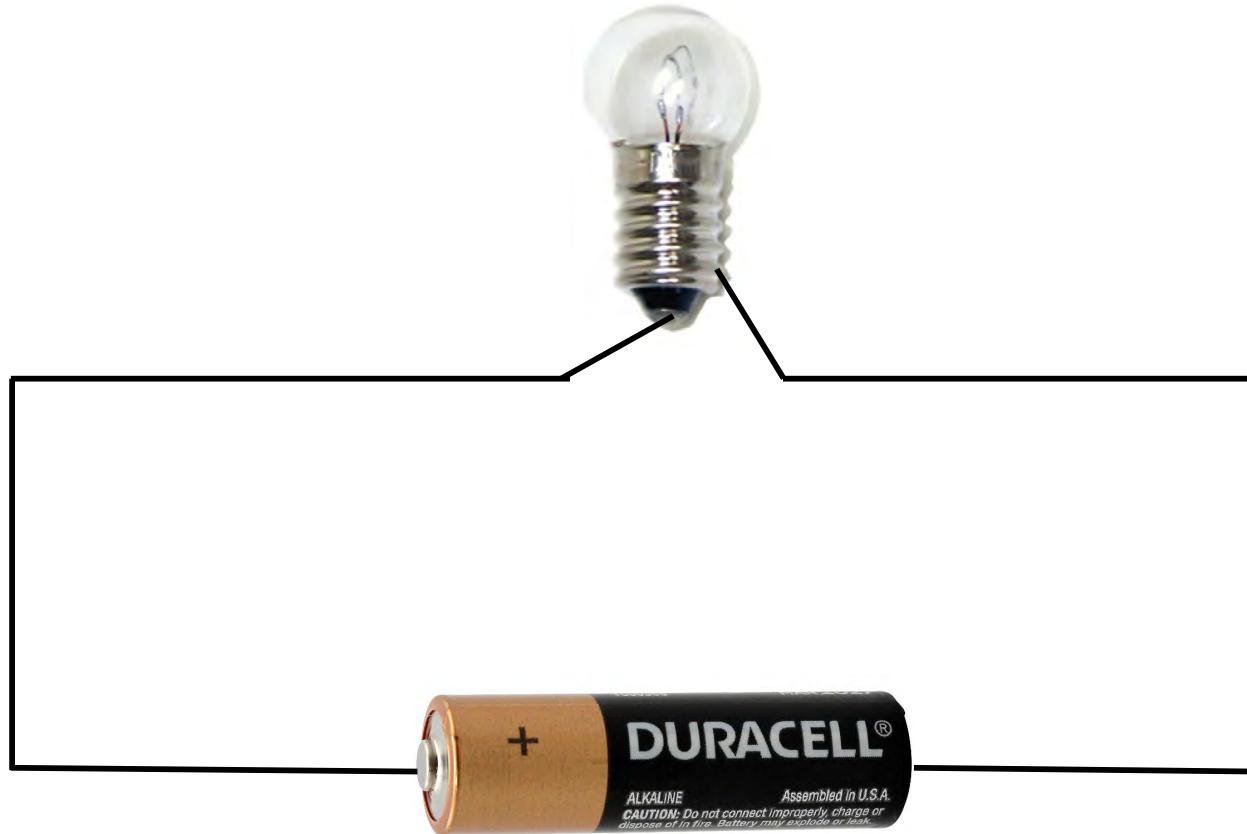


No charge differential



Charge differential

# Making an Electrical “Circuit”



# Electrical Current Can Be Manipulated

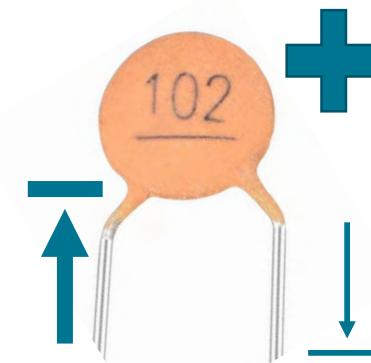
- It is possible to take current and change it

Example: Resistor



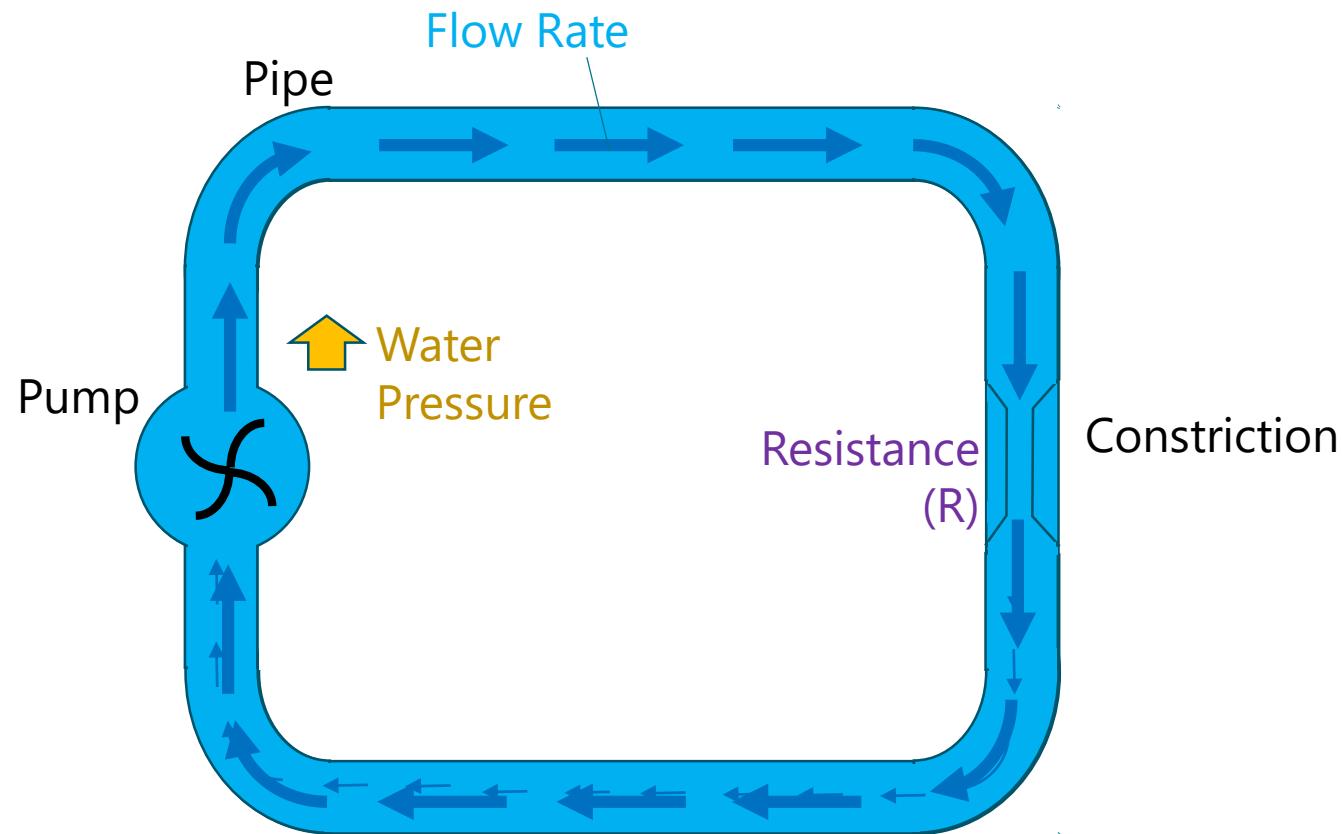
Reduces flow rate of electricity

Example: Capacitor

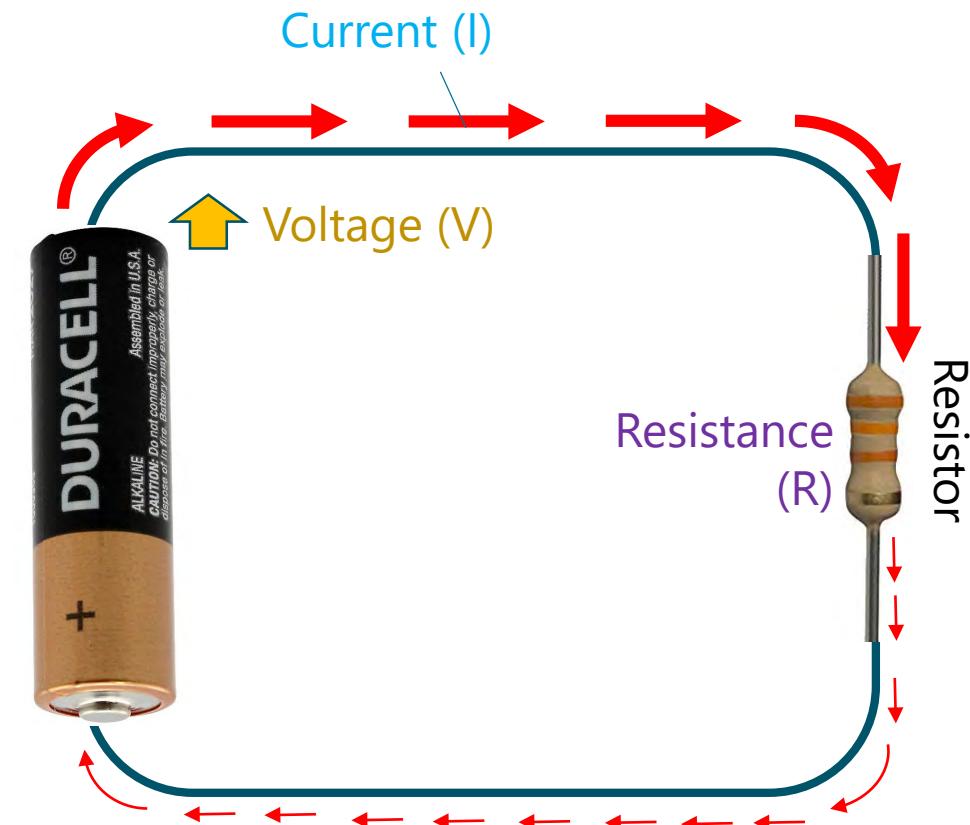


Stores electrical “charge”

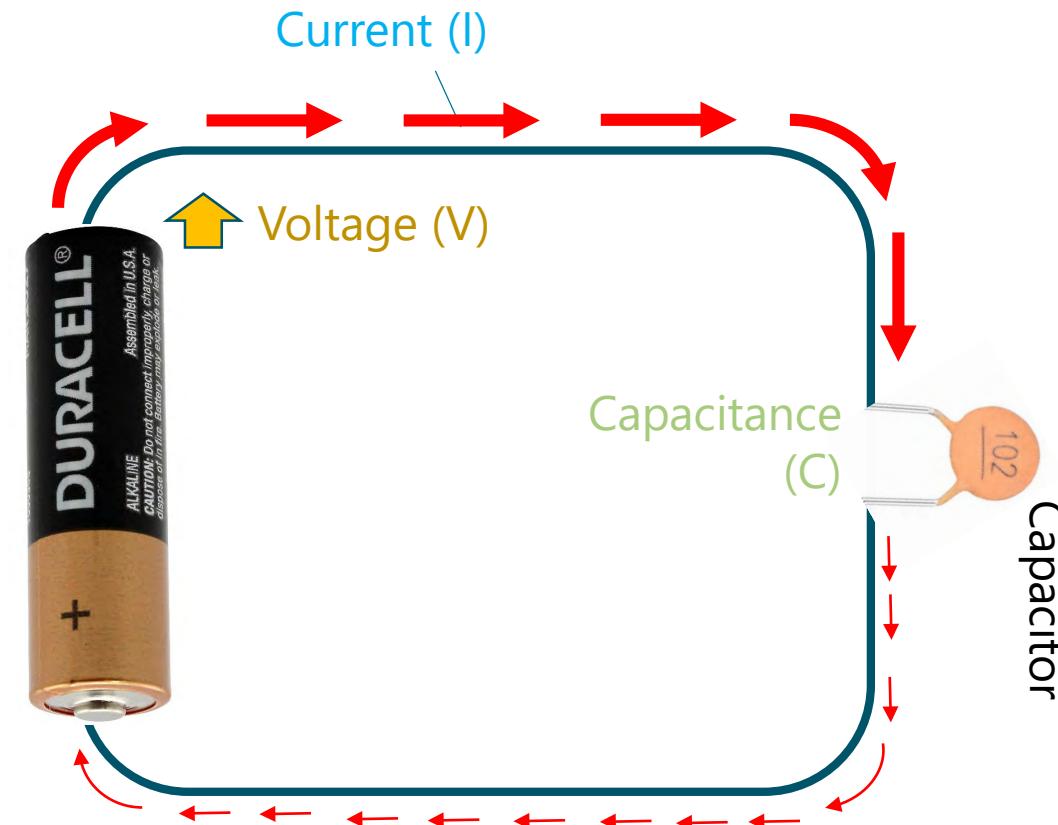
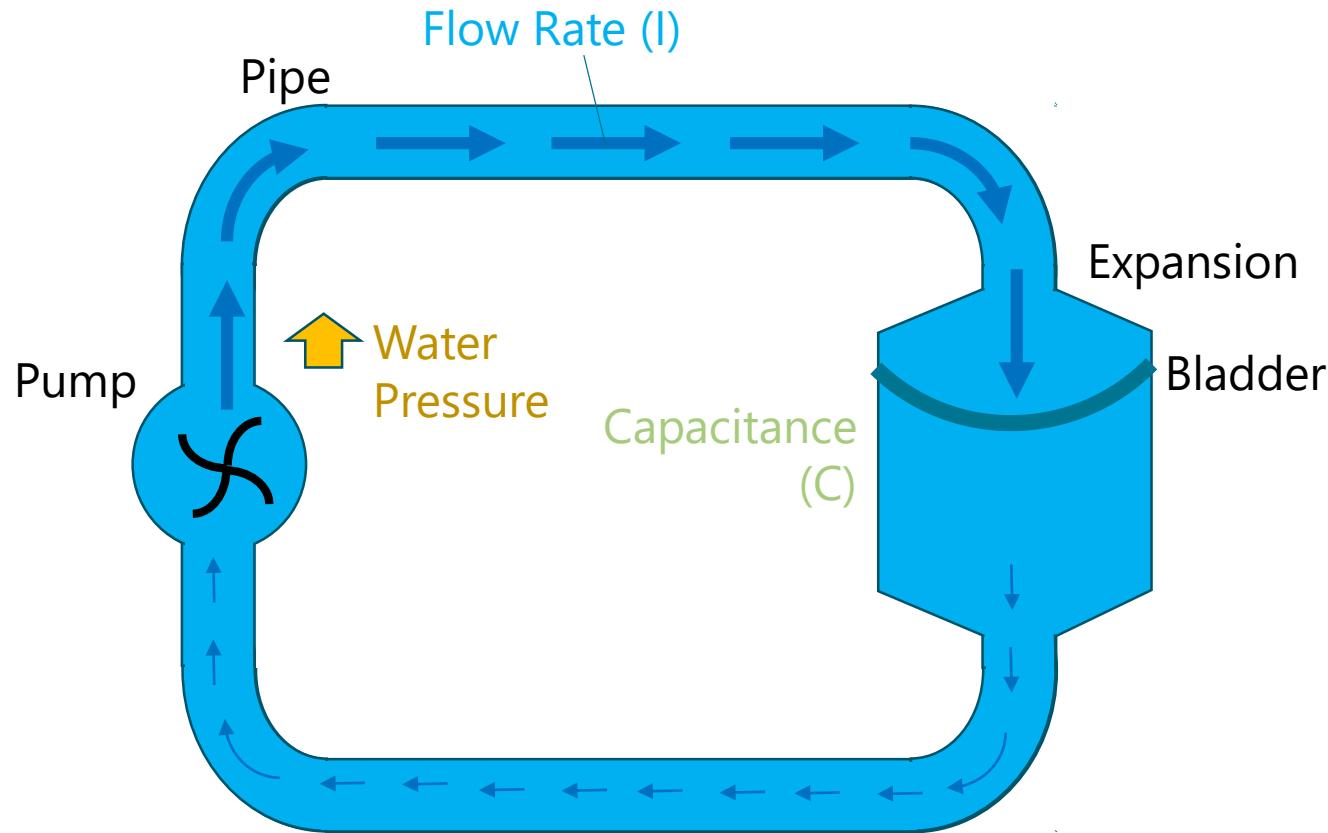
# Flowing Electricity Is Like Flowing Water



Constriction



# Water Analogy: Capacitance



# Electrical Components



Resistor



Capacitor



Potentiometer



Transistor



Relay



Diode



Photocell



Light-Emitting  
Diode (LED)



Crystal  
Oscillator



DC Fan



Speaker



Motor



Piezo Buzzer



7-Segment  
Display



Toggle  
Switch

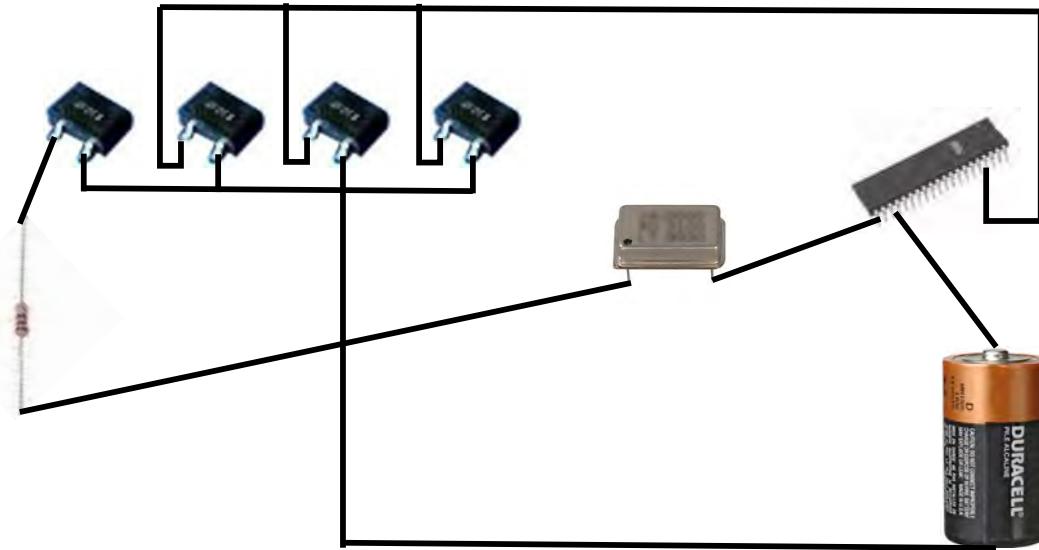


AA Battery



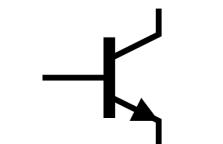
Electrical  
Ground

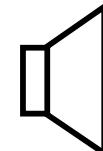
# How to draw a circuit?



- Need to remember our designs, show to others
- Need to clearly describe what each component is, properties of component, which pins are connected, etc.

# Electronic Symbols

Name	Picture	Symbol
Resistor		
Capacitor		
Diode		
Transistor		
Oscillator		

Name	Picture	Symbol
Electric Motor		
Speaker		
Voltage Source		
Switch		
Electrical Ground		

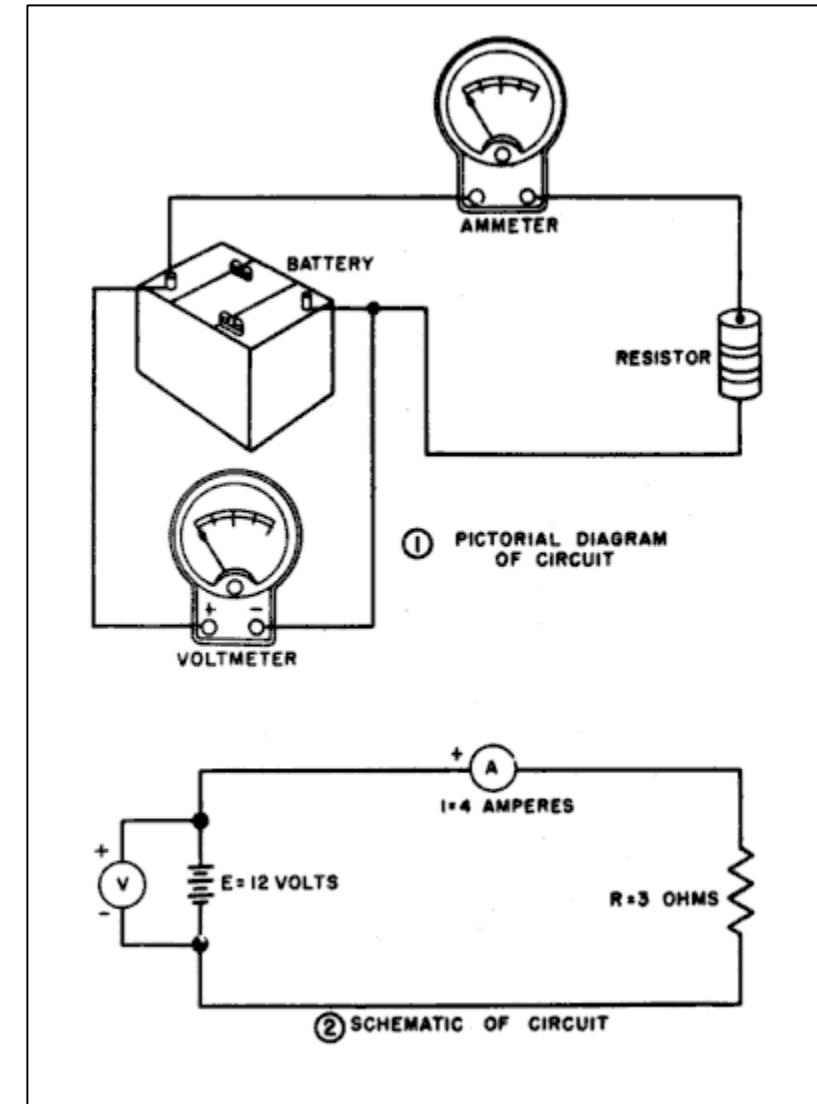


Diagram of a basic Circuit

# How to decide which components to use?

1. Think about what you want to build
2. Design a circuit that achieves that objective
3. List components that are required by the circuit and their critical parameters

# How to decide which components to use?

4. Choose components with desirable properties
  - Voltage range, tolerance, resistance value, temperature coefficient, wattage, load efficiency, etc.
  - Components that reduce circuit complexity (e.g., built-in ESD protection)
  - Environmental parameters (humidity/pressure/vibration/temperature resilience)

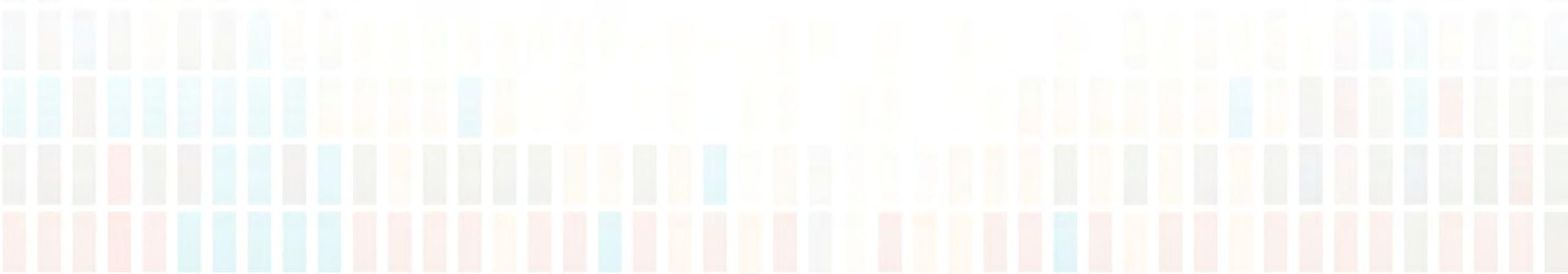
# How to decide which components to use?

## 5. Choose components that fit in the circuit well

- Manufacturer with good documentation/support/tools, long-term availability, lead-time, cost, fall-back options
- Mechanical parameters (dimensions, weight, etc.)

## 6. Prototype before you start production

# Use Cases

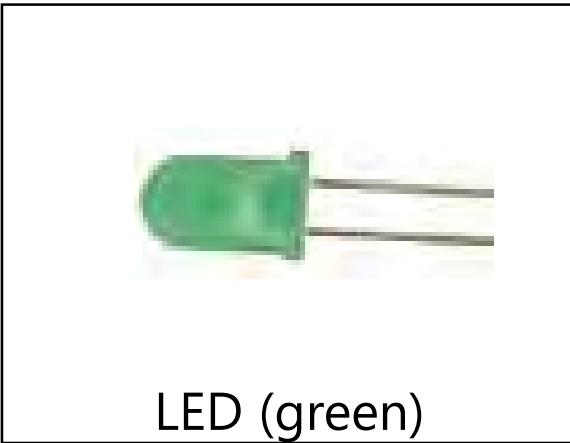


Suppose you want to build...  
*something that lights up*

- Indicator/status signals
- User interfaces and displays
- Illumination sources
- Light effects



# Components that make light



LED (green)



TFT Display



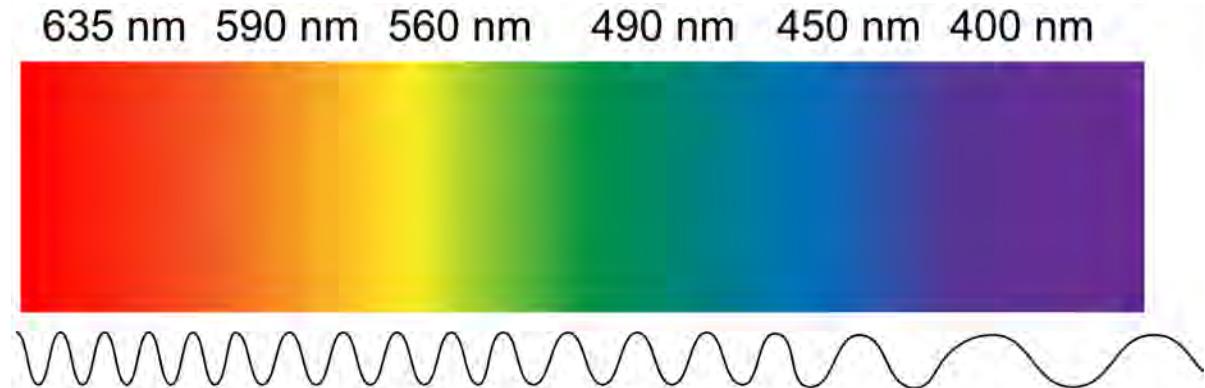
Optical Audio  
Transmitter

- **What they do:** Create light
- **Key metrics:**
  - **Brightness:** how much light is generated (*mcd/nits*)
  - **Viewing angle:** over what angle most of the light is visible (*degrees*)
  - **Color:** color of emitted light (*nm/kelvin*)
  - **Voltage:** voltage needed to "turn on" light (*volts*)
  - **Current:** current needed to "turn on" light (*mA*)

# How color is measured

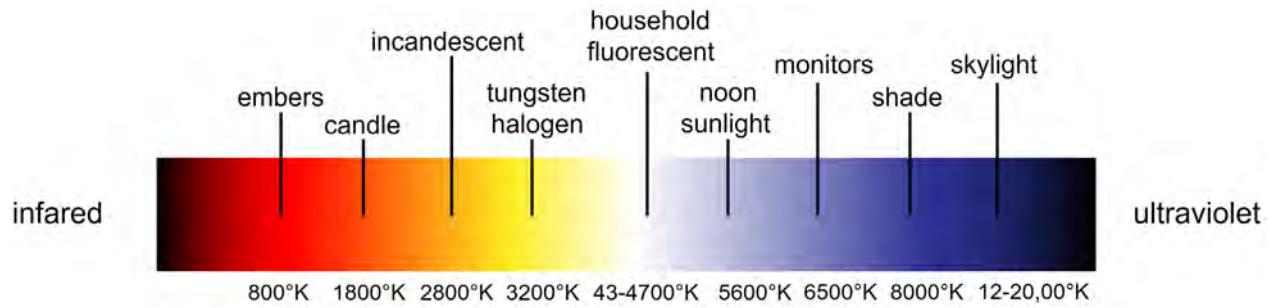
- **Option 1: Wavelength**

- Color of light as function of electromagnetic wavelength/frequency



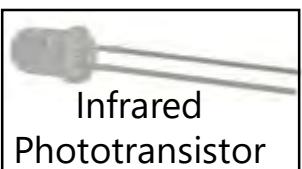
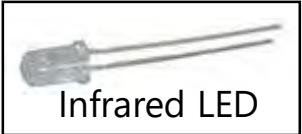
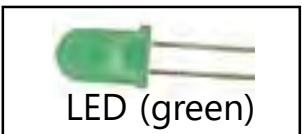
- **Option 2: Black body radiation**

- All objects radiate light based on their temperature
- Color of light as a function of "blackbody" temperature

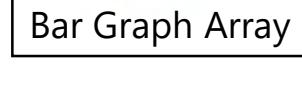


# More components that make light

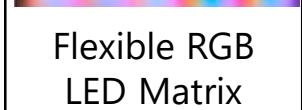
Colored LEDs



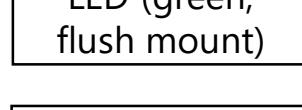
Arranged LEDs



Light Matrices



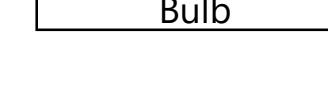
Shaped LEDs



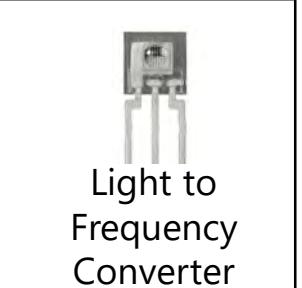
Directed Light



Other Light Tech

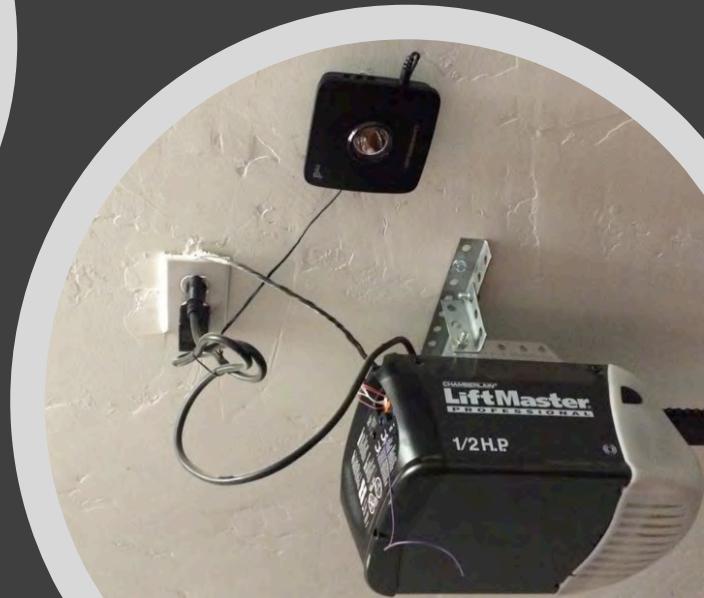


Light Signaling



Suppose you want to build...  
*something that needs electricity*

- Operate for long periods of time without management
- Move around without being plugged in
- Collect energy from environment
- Supply electricity to other devices
- Change voltage from one level to another
- Protect against voltage spikes, electronic noise, and static electricity



# Electrical power sources

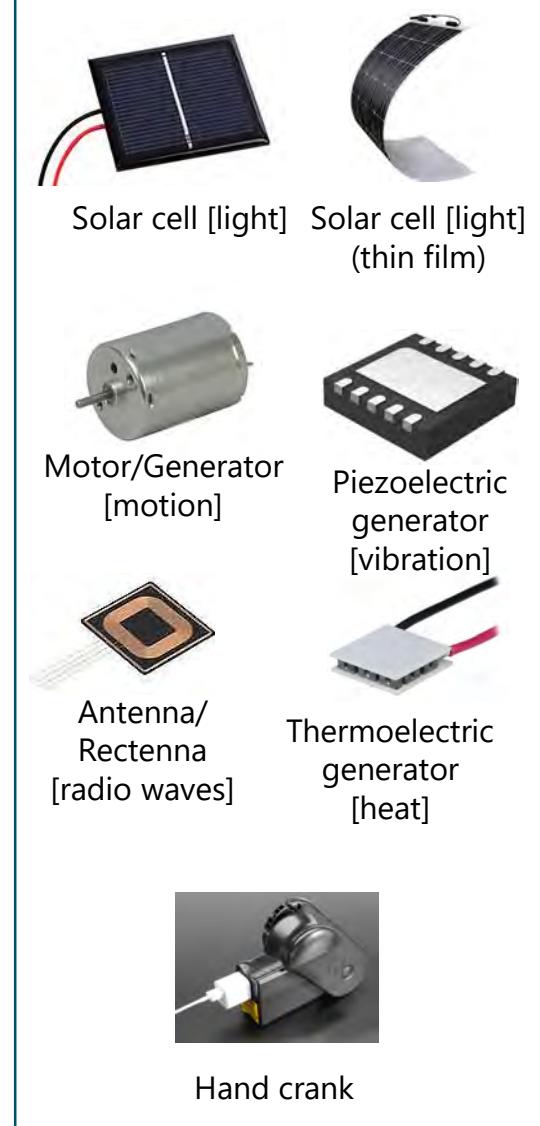
## Batteries



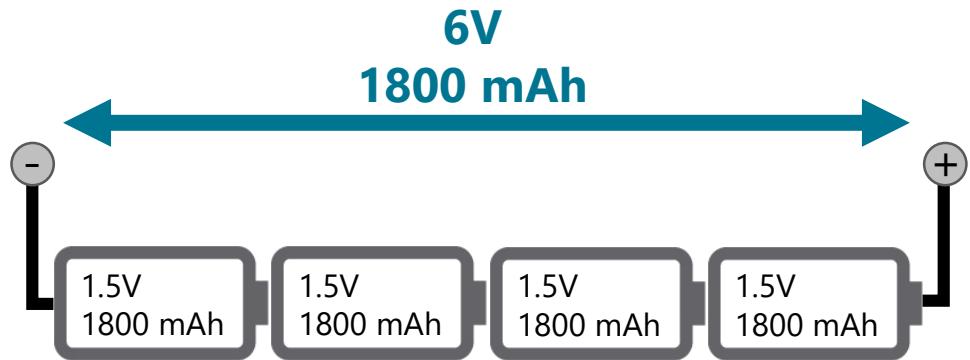
## Plugging in to the Grid



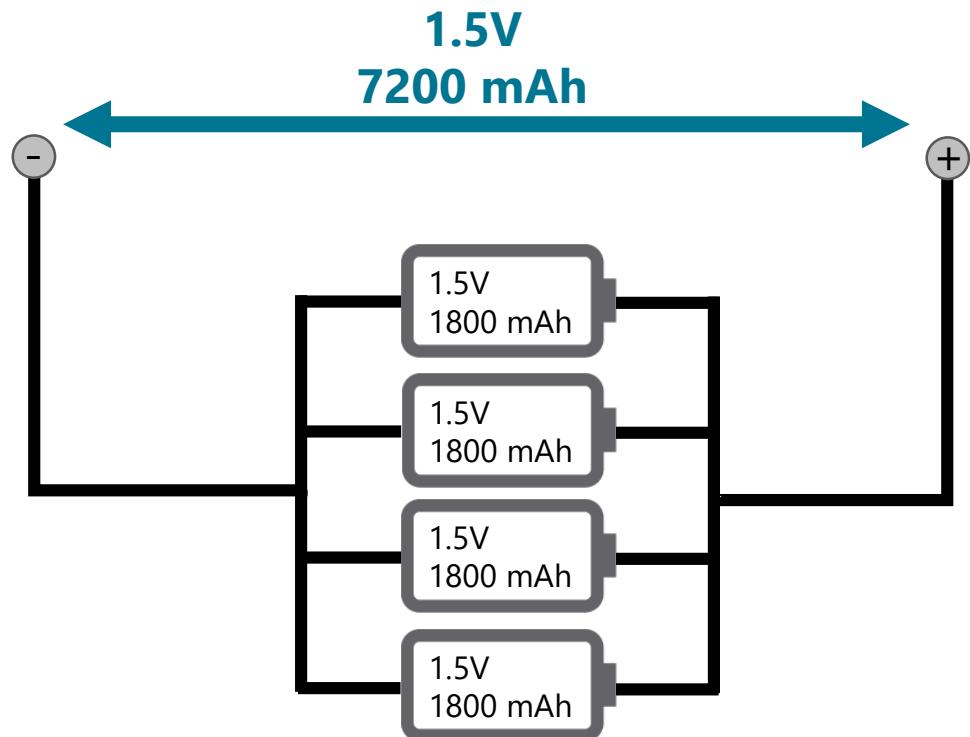
## Energy Harvesting



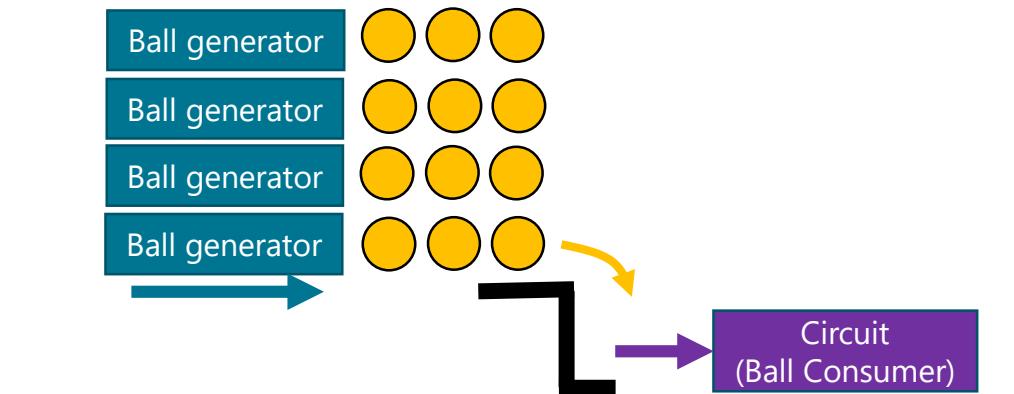
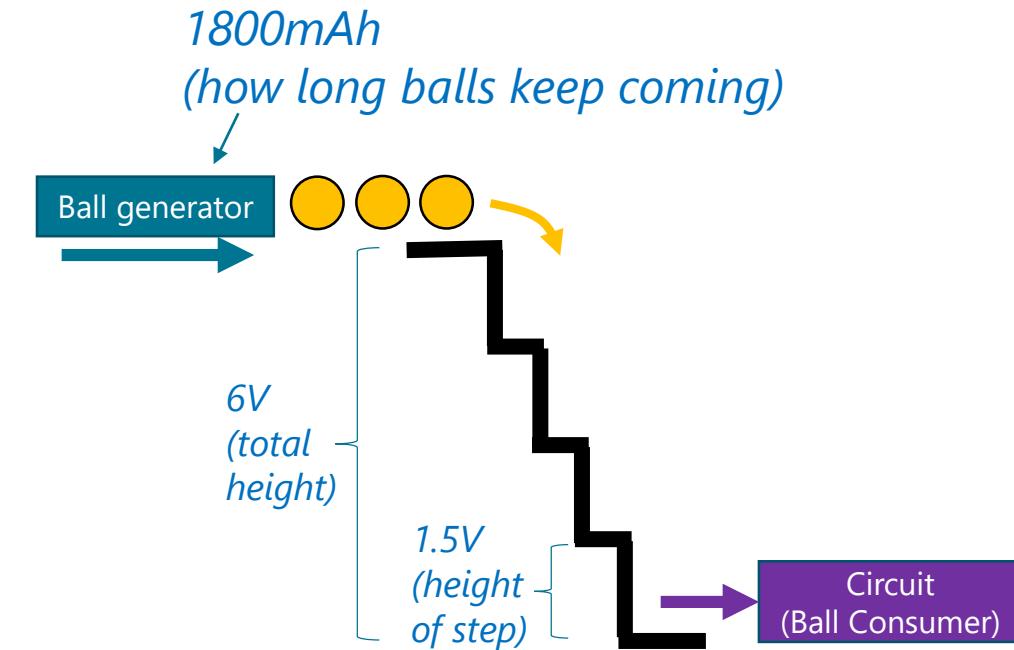
# Power: Combining Current Sources



*Connected  
in Series*



*Connected  
in Parallel*



# Components that change voltage levels



Transformer  
(steps up/down AC  
voltage)



Voltage regulator  
(keeps voltage within  
range)



Buck converter  
(steps down DC  
voltage)



Boost converter  
(steps up DC voltage)

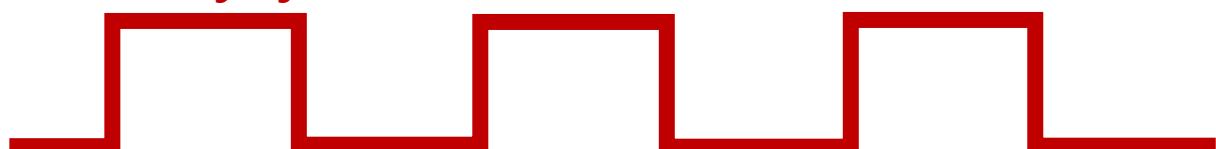
# Power: Pulse-Width Modulation (PWM)

- Suppose you want to increase/ decrease power to a component
  - E.g., vary brightness of LED, speed of motor
- **PWM:** controls power by rapidly turning on/off
  - Cheaper, more efficient electronics than varying voltage

**25% duty cycle:**



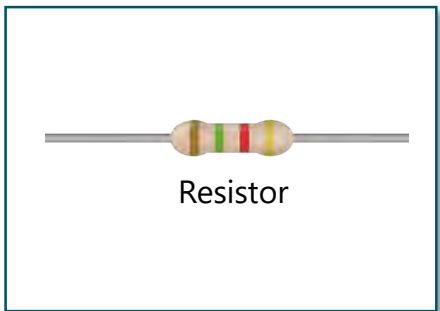
**50% duty cycle:**



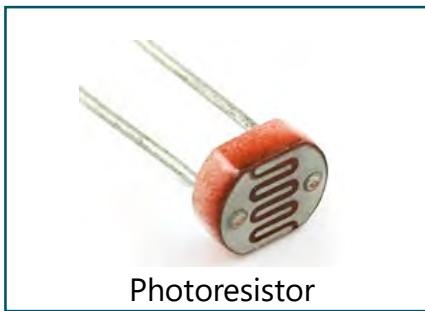
**75% duty cycle:**



# Resistors and Potentiometers



Resistor

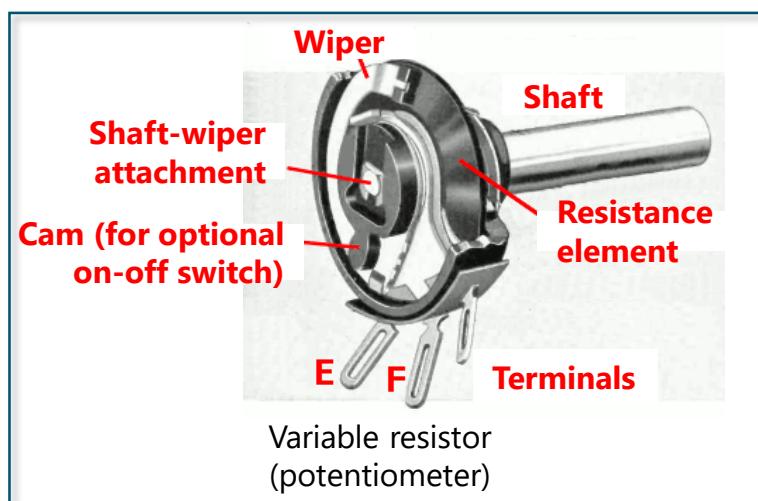


Photoresistor

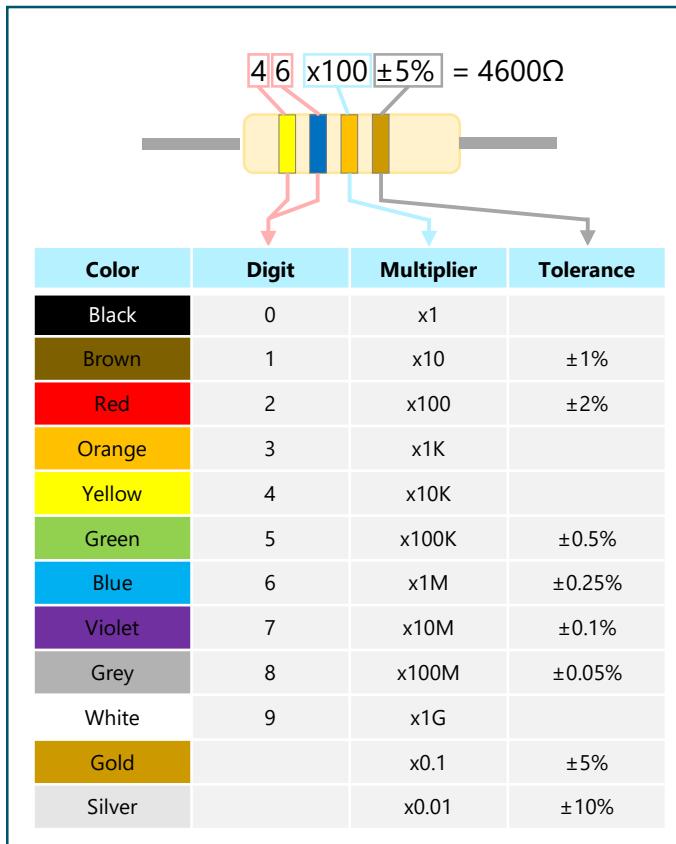
**What they do:** resist the flow of current

## Useful for:

- Protecting components from getting too much current
- Increasing and decreasing voltage
- Measuring things (e.g., light intensity)



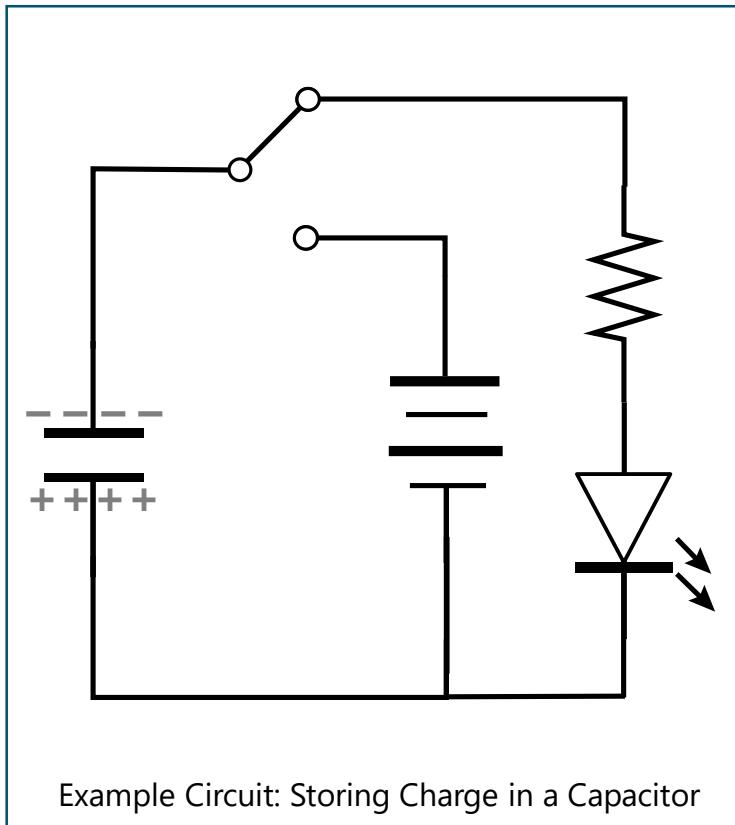
# Resistors and Potentiometers



## Key metrics:

- **Resistance:** how much they resist current (Ohms)
- **Tolerance:** how accurate their resistance rating is (%)
- **Power rating:** how much current they can handle (Watts)
- **Maximum voltage:** how much voltage difference across their input/output they can handle

# Capacitors

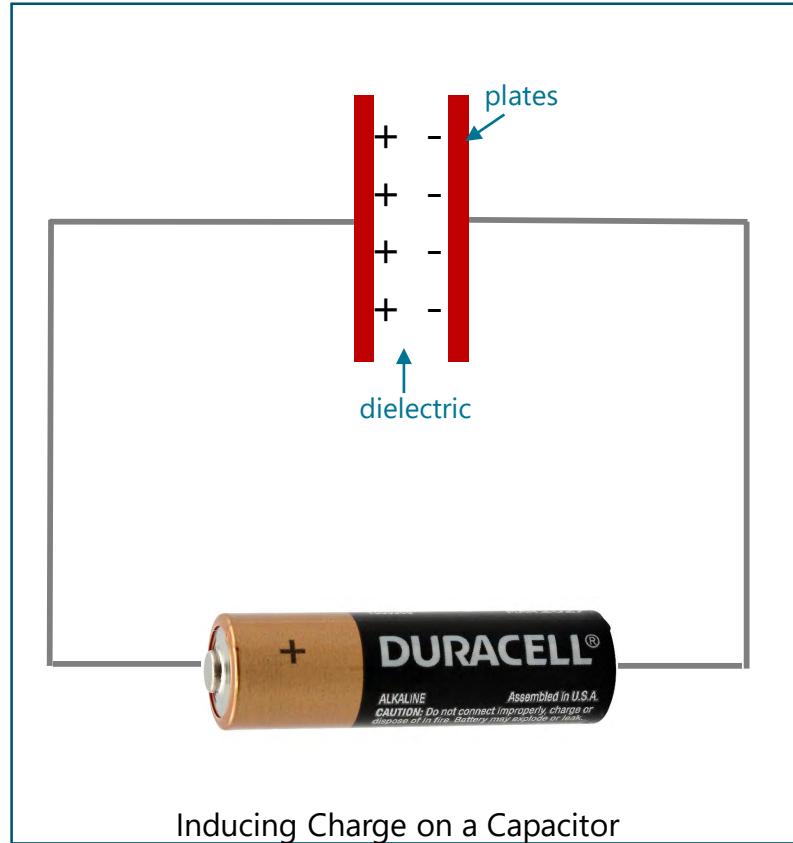


**What they do:** store charge

**Useful for:**

- Temporarily holding some charge (like a tiny battery)
- Smoothing out voltage spikes
- Measurement (humidity, pressure, touch sensors)

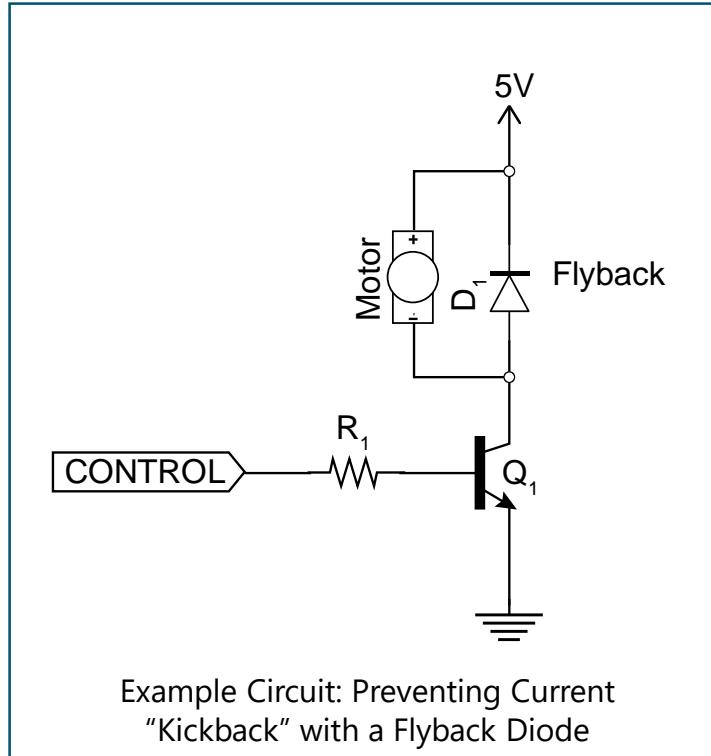
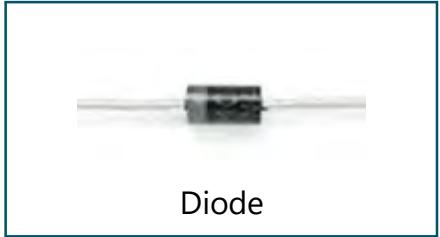
# Capacitors



## Key metrics:

- **Capacitance:** how much charge it can store (farads)
- **Tolerance:** how accurate their capacitance rating is (%)
- **Maximum voltage:** how much voltage they can handle (before they short out or burn up)
- **Leakage current:** amount of current that leaks through dielectric
  - No “current rating” – you can’t have a sustained current through a capacitor aside from leakage
- **Equivalent series resistance (ESR):** capacitors aren’t perfect and have tiny amount of resistance (usually less than  $0.01\Omega$ )

# Diode

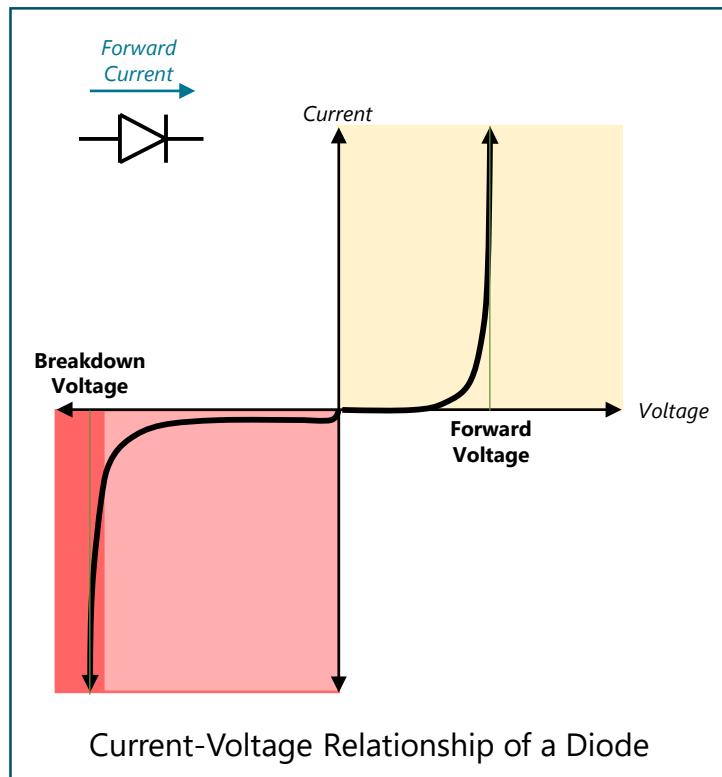


**What it does:** allows current to go in forward direction, but not reverse direction

**Useful for:**

- Reverse current protection
- “Rectifying” signals (pulling out positive parts)
  - Converting AC to DC
- Measurement (temperature, radiation)

# Diode



## Key metrics:

- **Maximum forward current:** how much current can go through in the forward direction (amps)
- **Maximum reverse voltage (breakdown voltage):** how much voltage can be withheld in reverse direction
- **Maximum forward voltage:** voltage difference between input/output when current going through forward direction (ideally should be zero, no resistance to current)

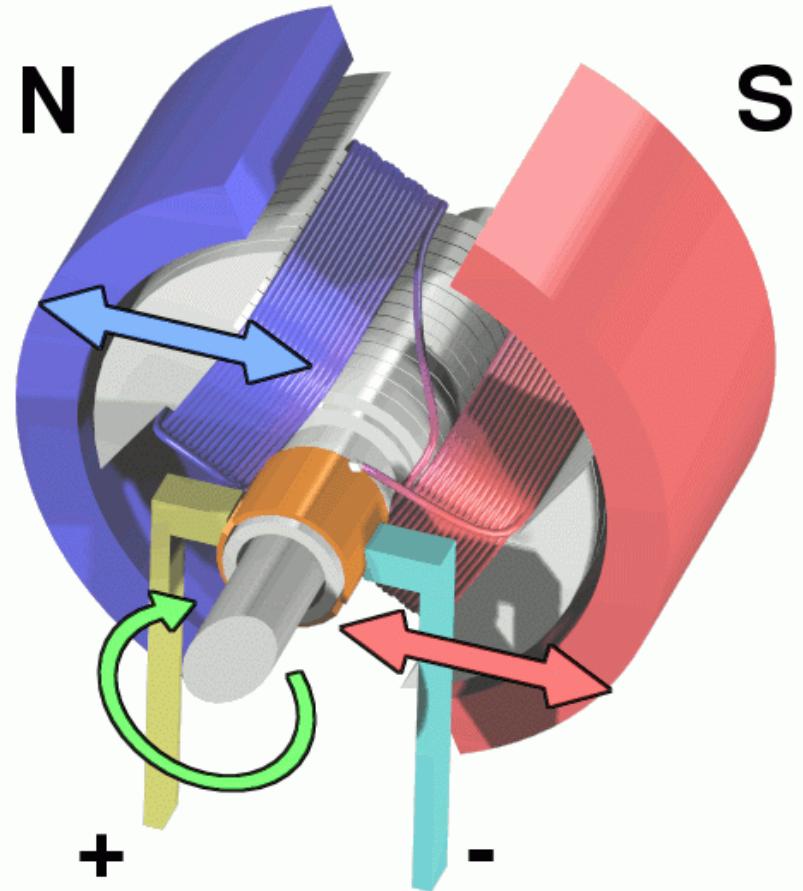
Suppose you want to build...  
*something that moves*

- Position control surfaces, antennas, conveyors, etc.
- Operating grippers, levers, dispensers, belts, cutters, etc.
- With varying speed, precision, torque/power, etc.



# Electric Motors

- Converts electricity into motion
- Main idea:
  - Get two big magnets, one north and one south
  - Run electricity through a curved wire
    - Alternatively pushes/pulls against magnets
    - Rotates a mechanical shaft



# Common Types of Electric Motors

# More Types of Electric Motors



**Generator**  
(run brushed  
motor via  
external force  
to generate  
current)



**Servo**  
(like stepper, but with  
position feedback  
mechanism to control  
position – more  
expensive, higher  
torque/speed)



**Torque**  
Can operate  
indefinitely when  
stalled, applying  
constant torque  
(force feedback  
games, positioning)



**Linear**  
“Unrolled”  
motor, produces  
straight-line  
torque (maglev  
trains, pen  
plotters)

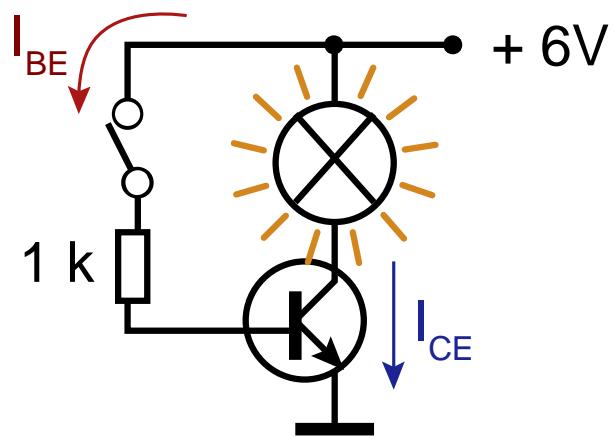


**Vibration**  
Unbalanced  
weight  
attached to  
pole (phones,  
pagers, toys)

# Transistors/Relays



Transistor



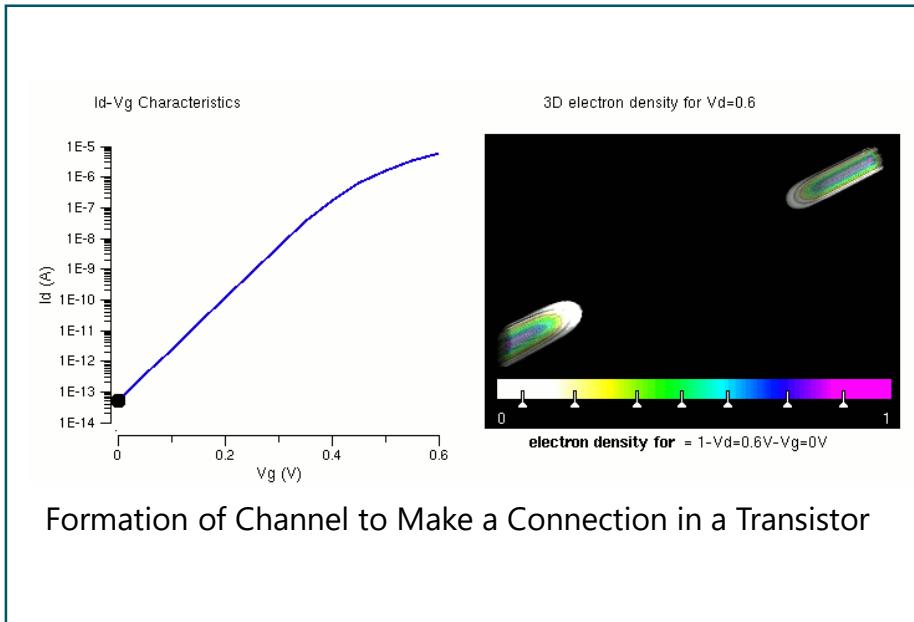
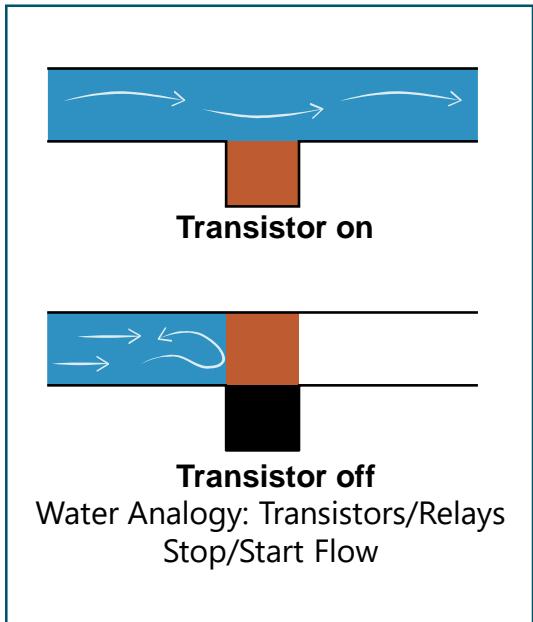
Example Circuit: Controlling a Light with an NPN Transistor

**What it does:** acts like a switch; when voltage applied to one wire, forms connection between other two wires

**Useful for:**

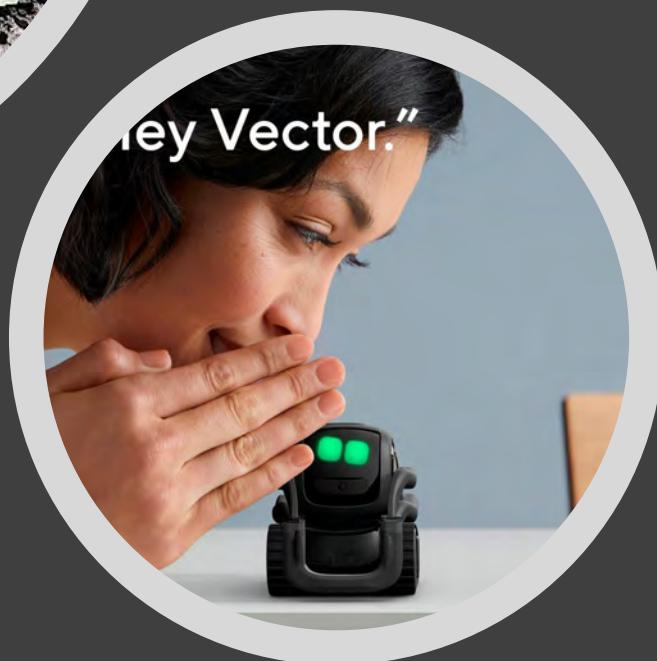
- Acting like a switch
- Amplifying a signal

# Transistors/Relays



## Key metrics (transistors):

- **Type:** does applied current turn on connection (NPN) or turn off connection (PNP)
- **Maximum reverse voltage (breakdown voltage):** how much voltage can be withheld in reverse direction
- **Maximum forward voltage:** voltage difference between input/output when current going through forward direction (ideally should be zero, no resistance to current)



Suppose you want to build...  
*something that observes*

- Listens, sees, measures
- Sound, light, temperature, humidity, magnetic fields, rotation, acceleration, etc.

# Sensors

# Sensors

# Sensors

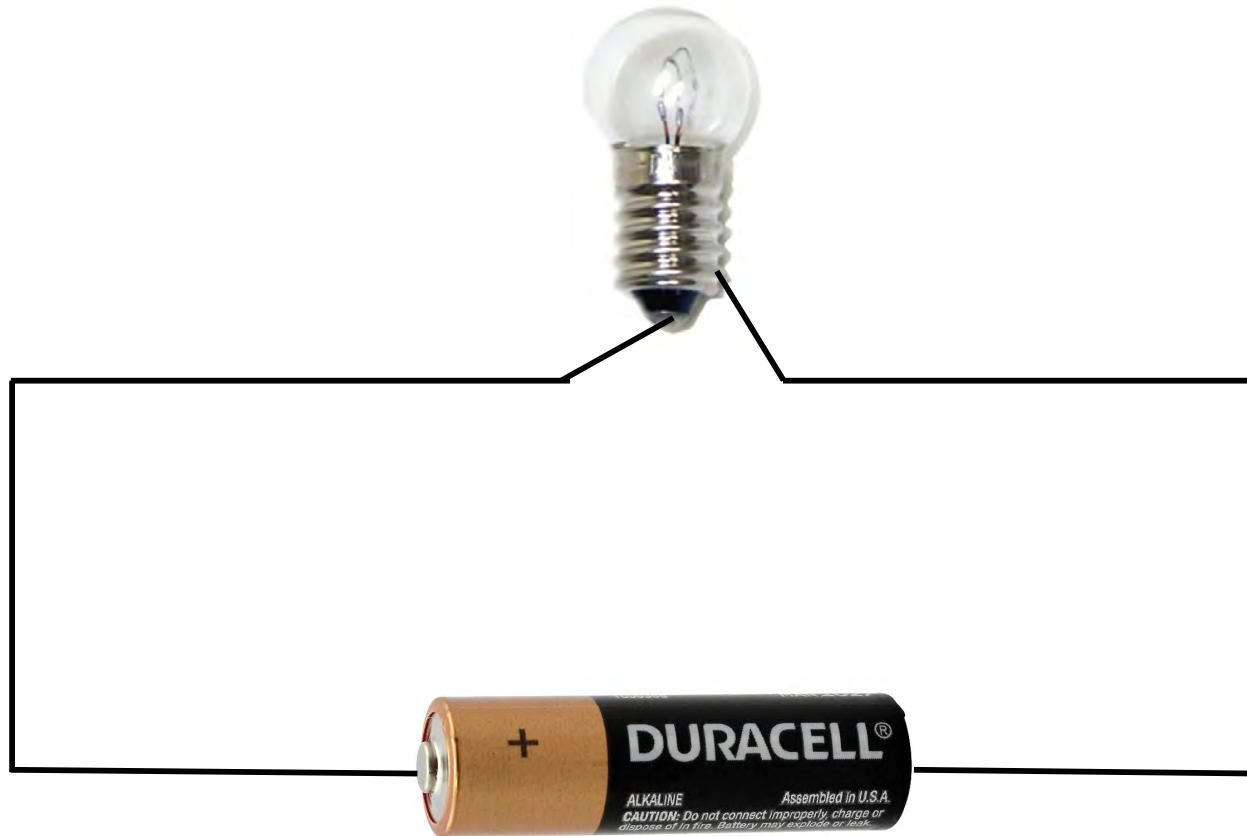
# Sensors

# Emerging Sensors

Recently emerging sensor types make IoT more powerful

---

# Making a Circuit



- Can put components together to build circuits that do things

# How to Build Circuits

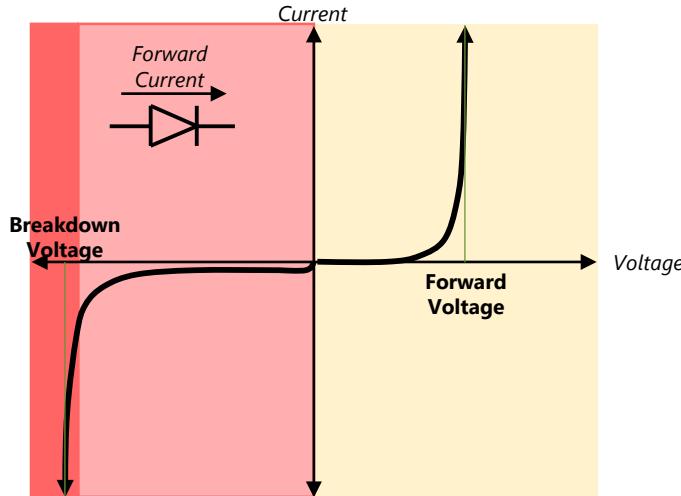
- Like algorithms, common circuits are known
  - Can learn the common ones
- You don't need to memorize a lot of circuits to do IoT
  - More complicated circuits tend to be componentized anyway
  - Can buy pre-created chips and PCBs
  - Can also look up advanced ones when needed
- Next, we will go through some that are helpful to know for IoT projects
  - Build intuition, use these when you need them

# Six Useful Circuits to Know About

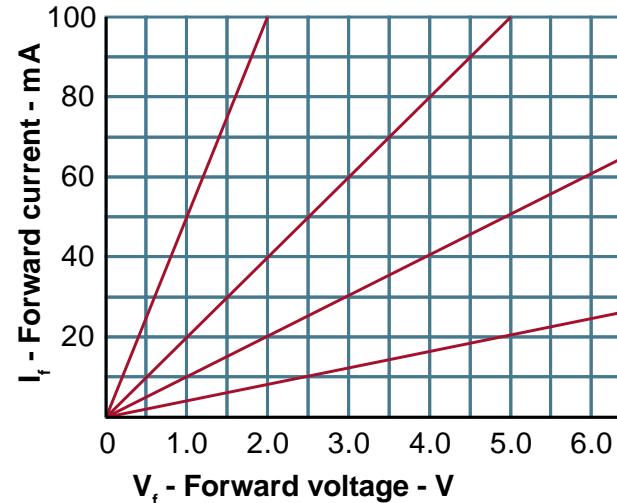
Widely used, demonstrate useful concepts:

1. Current-Limiting Resistor
2. Voltage Shifter
3. Filter Capacitor
4. Pull-Down Resistor
5. Signal Amplifier
6. Transistor Switch

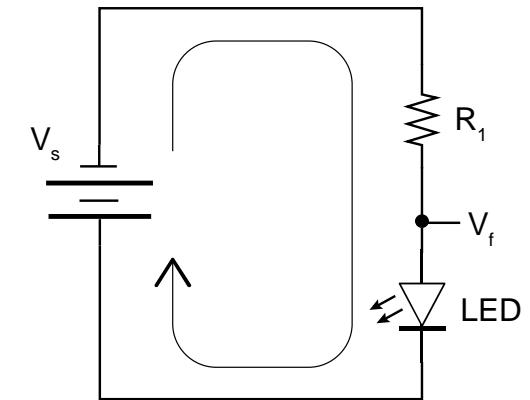
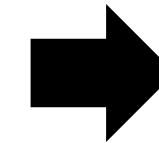
# 1. Protecting against Current Overload (Resistors)



**LEDs have  
non-linear resistance**



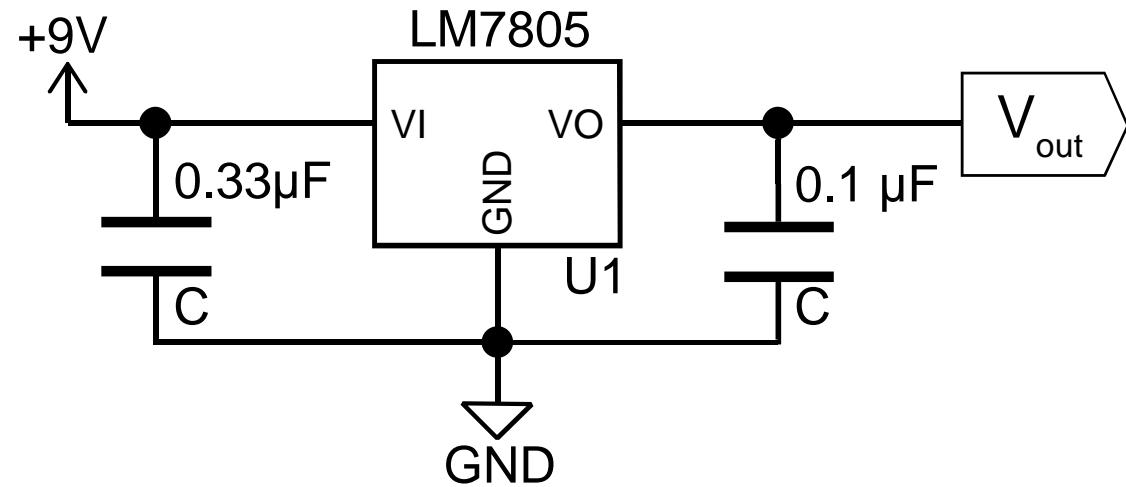
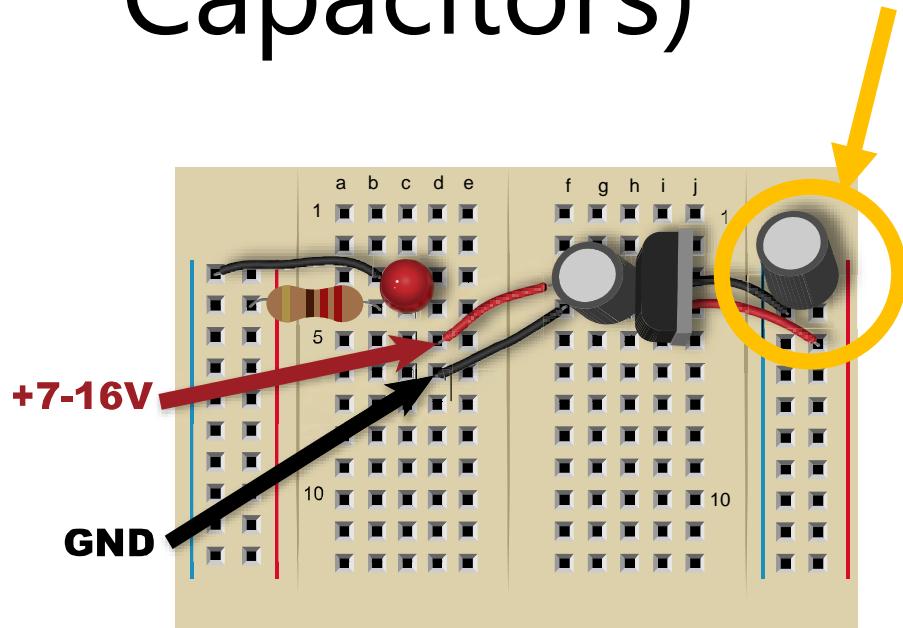
**Resistors have  
linear resistance**



**Current-limiting  
resistor**

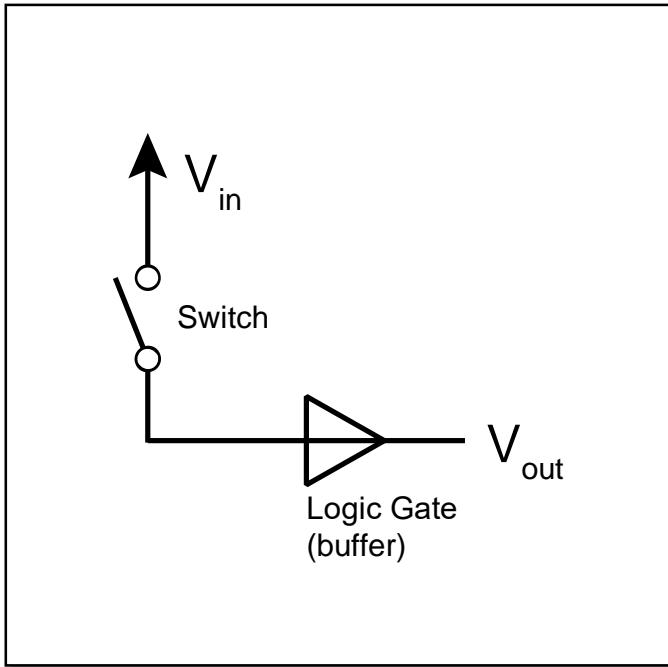
- Components can accidentally draw too much current
  - Resistors: current increases linearly with voltage
  - LEDs: if you overshoot voltage a little bit, suddenly LOTS of current
- Current-limiting resistors protect your components
  - Prevent too much current from going through your circuit

## 2. Protecting against Spikes (Filter Capacitors)

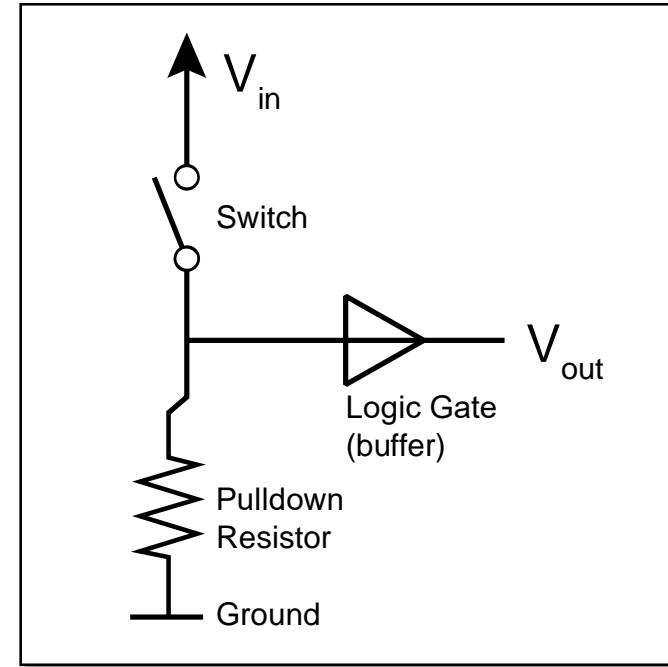


- Place capacitor between terminal and ground to smooth noise at the terminal
- Choose capacitor value based on noise you expect from the system
  - Rule of thumb:  $0.1\mu F - 10\mu F$  is pretty safe if you're just doing low voltage logic

### 3. Eliminating Noise from Disconnection (Pull-Down Resistors)

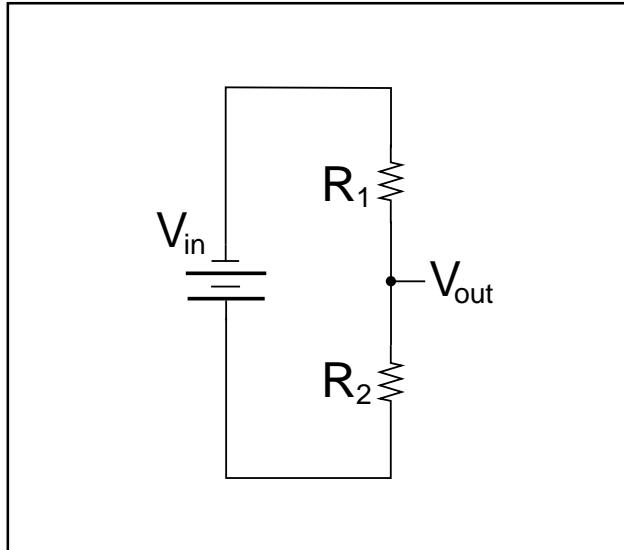


If an input pin is disconnected, its voltage is nondeterministic

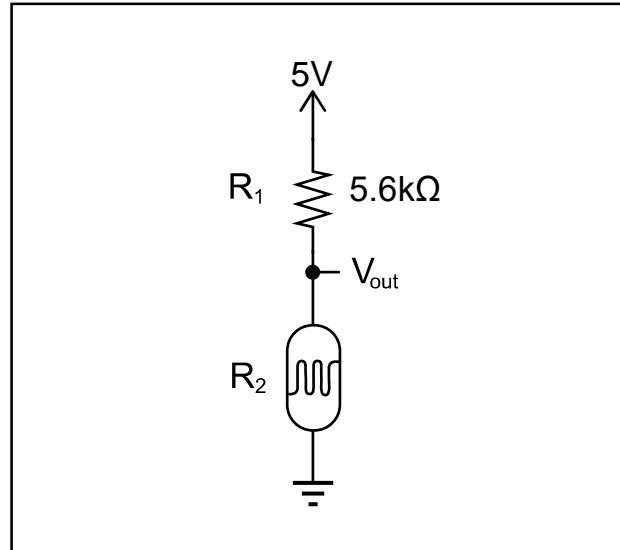


Solution – attach to ground, but add a resistor so you don't short-circuit

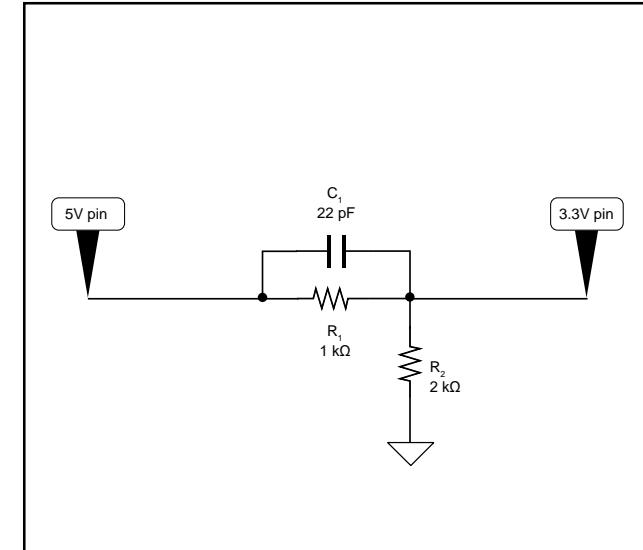
# 4. Getting the Voltage You Want (Voltage Divider/Regulator/Level Shifter)



***Voltage Divider***



***Reading Resistive Sensor***

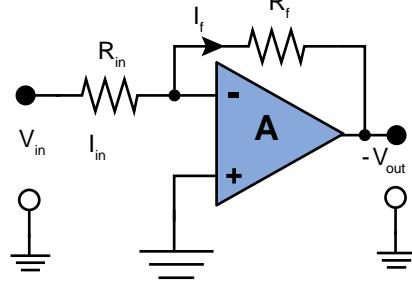


***Level Shifter***

- May need different voltage from your supply
- Resistors can be used to change voltage
- Transistors can be used to “switch in”

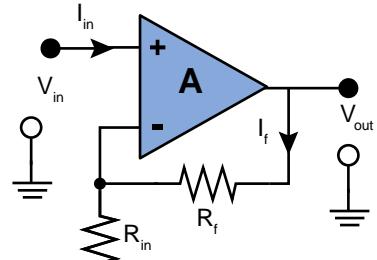
# 5. Amplifying What You Want to Hear (Operational Amplifiers)

Inverting Op-amp



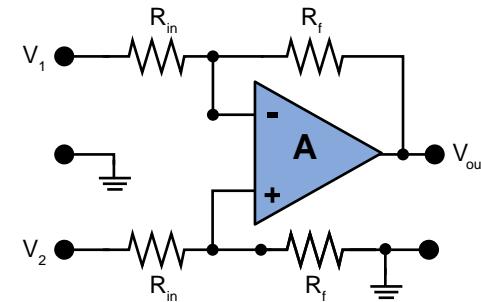
$$A = \frac{V_{out}}{V_{in}} = -\frac{R_f}{R_{in}}$$

Non-inverting Op-amp



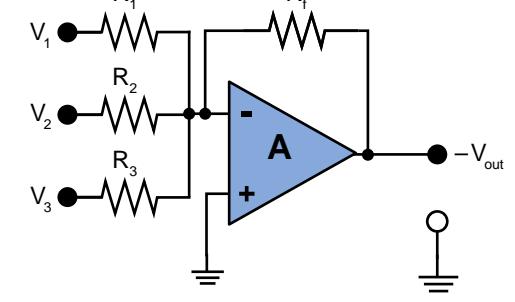
$$A = \frac{V_{out}}{V_{in}} = 1 + \frac{R_f}{R_{in}}$$

Differential Op-amp



$$V_{out} = \frac{R_f}{R_{in}} (V_2 - V_1)$$

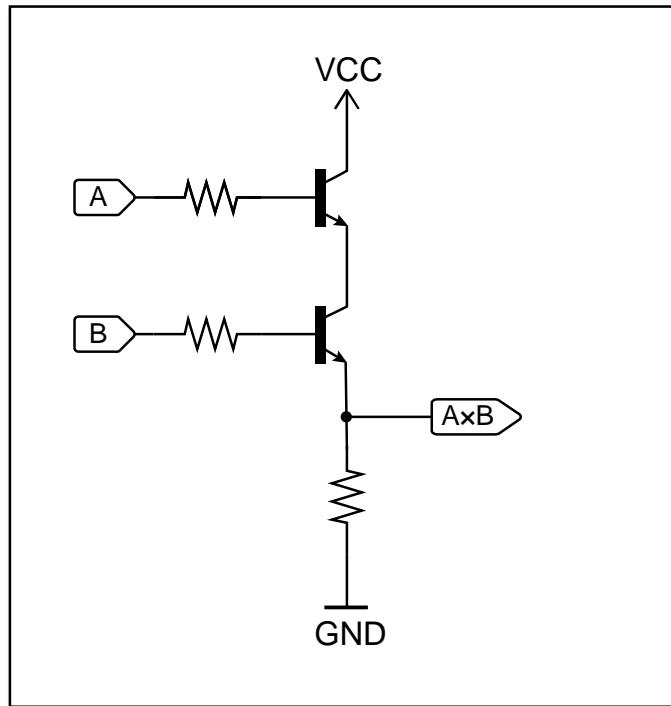
Summing Op-amp



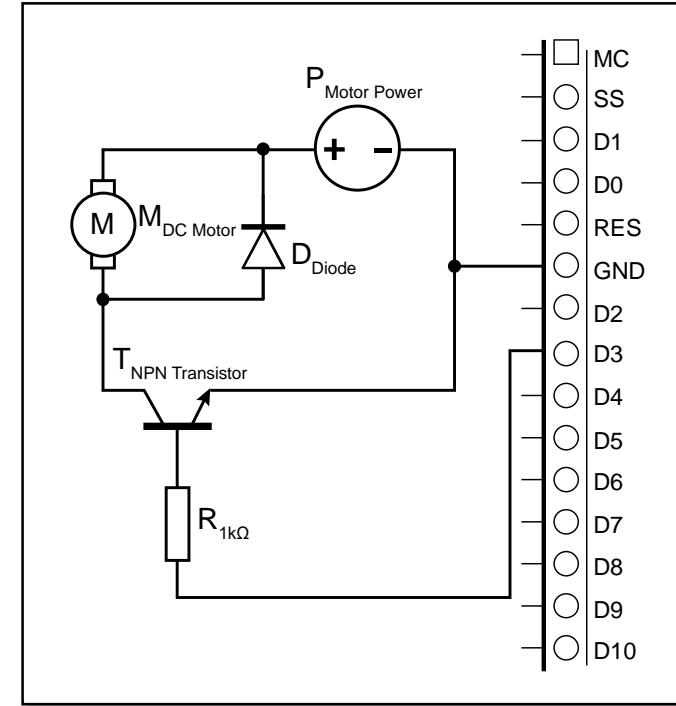
$$V_{out} = - \left( \frac{R_f}{R_{in}} V_1 + \frac{R_f}{R_2} V_2 + \frac{R_f}{R_3} V_3 \right)$$

- Two inputs, one output; amplifies difference between inputs
- Good Op-amps reject noise (common-mode voltages)
- Widely used to amplify inputs: audio amplifiers, sensors, radio NICs

# 6. Transistor Switches (Transistors, Relays)



**Logic Gates**



**Controlling Voltage**

- Allow an input to be controlled by another input
- Useful when we need to do conditional logic
- Useful when using logic to drive much larger voltage/current source

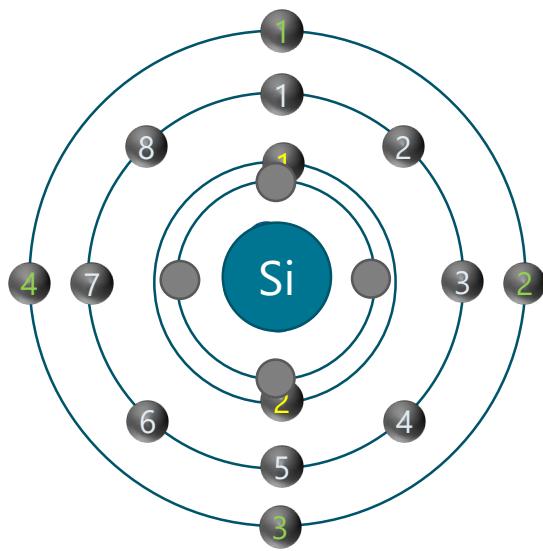
# Integrated Circuits

- “Discrete” circuits are great, but limits to how small/cheap you can build them
- Discovery in 1960s: you can actually “draw” circuits
  - Cost is low because they are printed en masse
  - Small size and tight proximity → they can switch quickly and use little power
- However, high initial cost to design and fabricate required “photomasks”
  - Practical only for high production volumes

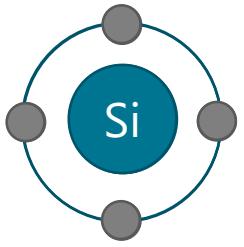
# Underlying Technology Behind ICs: Semiconductors

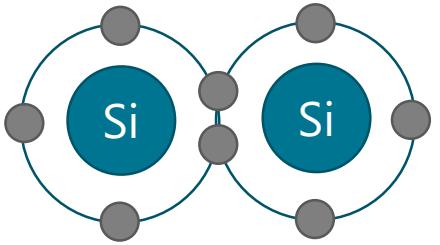
- Want a material you can “draw” on
- Semiconductor: poor insulator
  - Can “dope” it to become conductor
- Can selectively dope it in certain areas
  - Etch tiny wires, components
- Side benefits:
  - Excited electrons can emit light instead of heat (LEDs)
  - Resistivity can change with temperature, light, exposure to gas (sensors)
  - High thermal conductivity (good heat dissipation)

	Group →	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	
Period ↓																				
1		1 H																2 He		
2		3 Li	4 Be											5 B	6 C	7 N	8 O	9 F	10 Ne	
3		11 Na	12 Mg											13 Al	14 Si	15 P	16 S	17 Cl	18 Ar	
4		19 K	20 Ca	21 Sc	22 Ti	23 V	24 Cr	25 Mn	26 Fe	27 Co	28 Ni	29 Cu	30 Zn	31 Ga	32 Ge	33 As	34 Se	35 Br	36 Kr	
5		37 Rb	38 Sr	39 Y	40 Zr	41 Nb	42 Mo	43 Tc	44 Ru	45 Rh	46 Pd	47 Ag	48 Cd	49 In	50 Sn	51 Sb	52 Te	53 I	54 Xe	
6		55 Cs	56 Ba	57 La	*	72 Hf	73 Ta	74 W	75 Re	76 Os	77 Ir	78 Pt	79 Au	80 Hg	81 Tl	82 Pb	83 Bi	84 Po	85 At	86 Rn
7		87 Fr	88 Ra	89 Ac	*	104 Rf	105 Db	106 Sg	107 Bh	108 Hs	109 Mt	110 Ds	111 Rg	112 Cn	113 Nh	114 Fl	115 Mc	116 Lv	117 Ts	118 Og
		*	58 Ce	59 Pr	60 Nd	61 Pm	62 Sm	63 Eu	64 Gd	65 Tb	66 Dy	67 Ho	68 Er	69 Tm	70 Yb	71 Lu				
		*	90 Th	91 Pa	92 U	93 Np	94 Pu	95 Am	96 Cm	97 Bk	98 Cf	99 Es	100 Fm	101 Md	102 No	103 Lr				

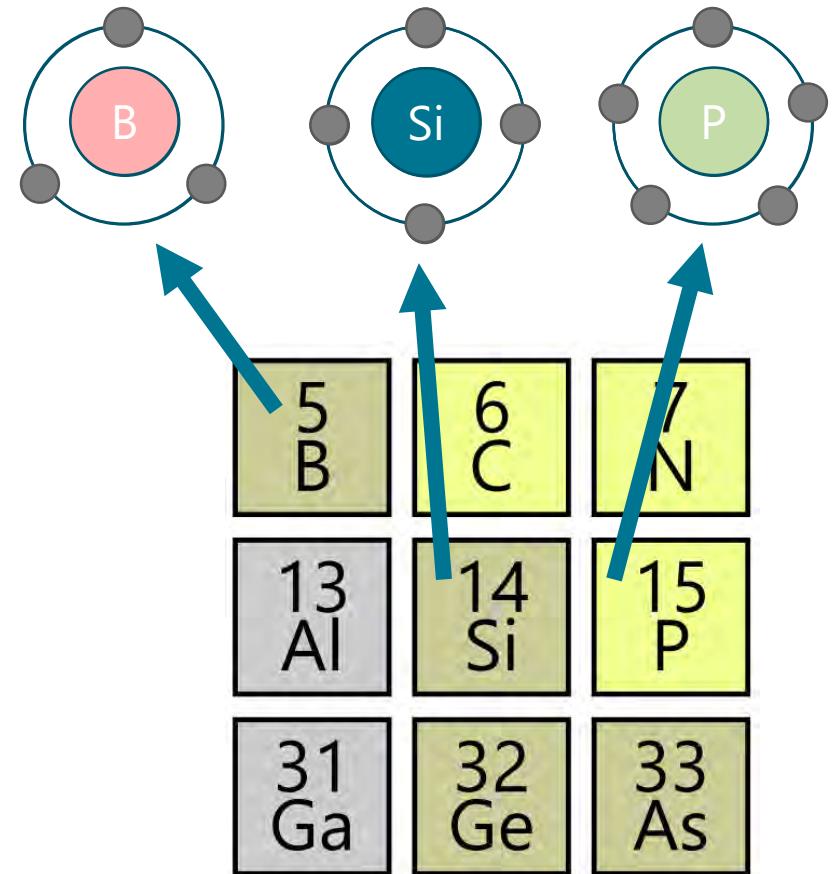
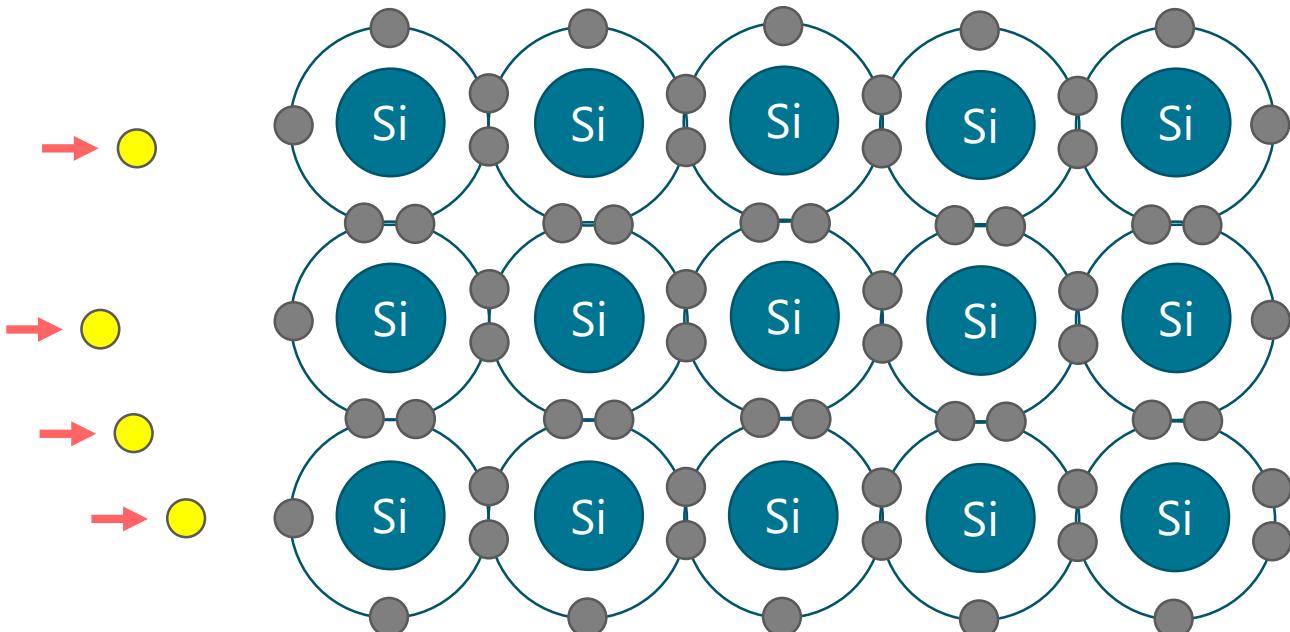


5	6	7
13	14	15
31	32	33

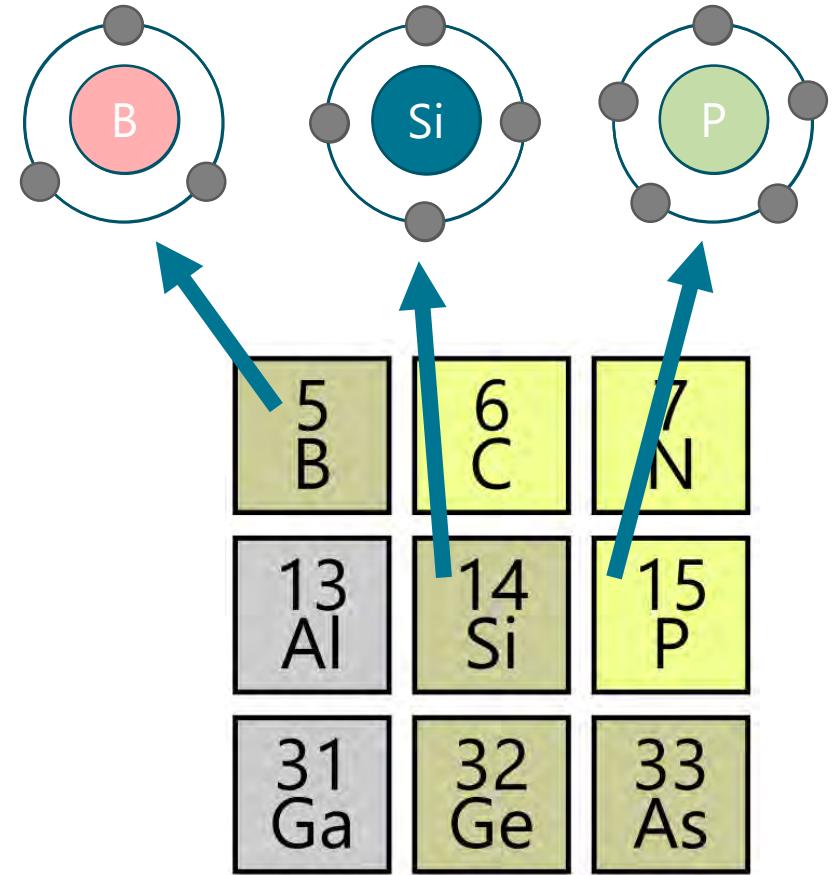
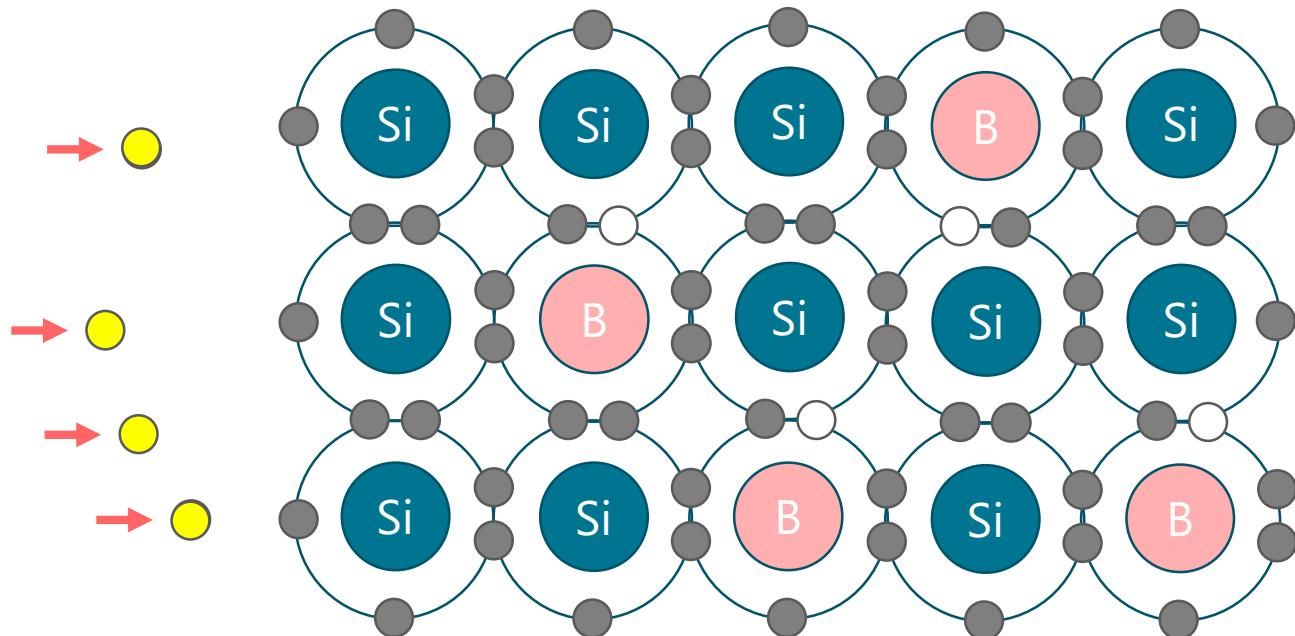




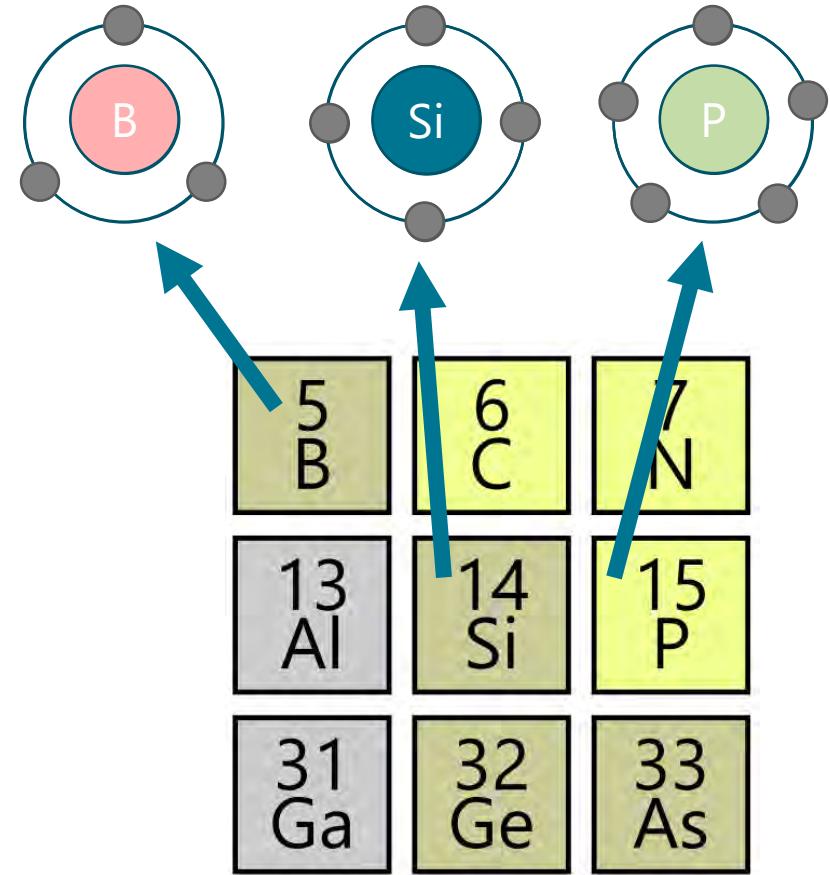
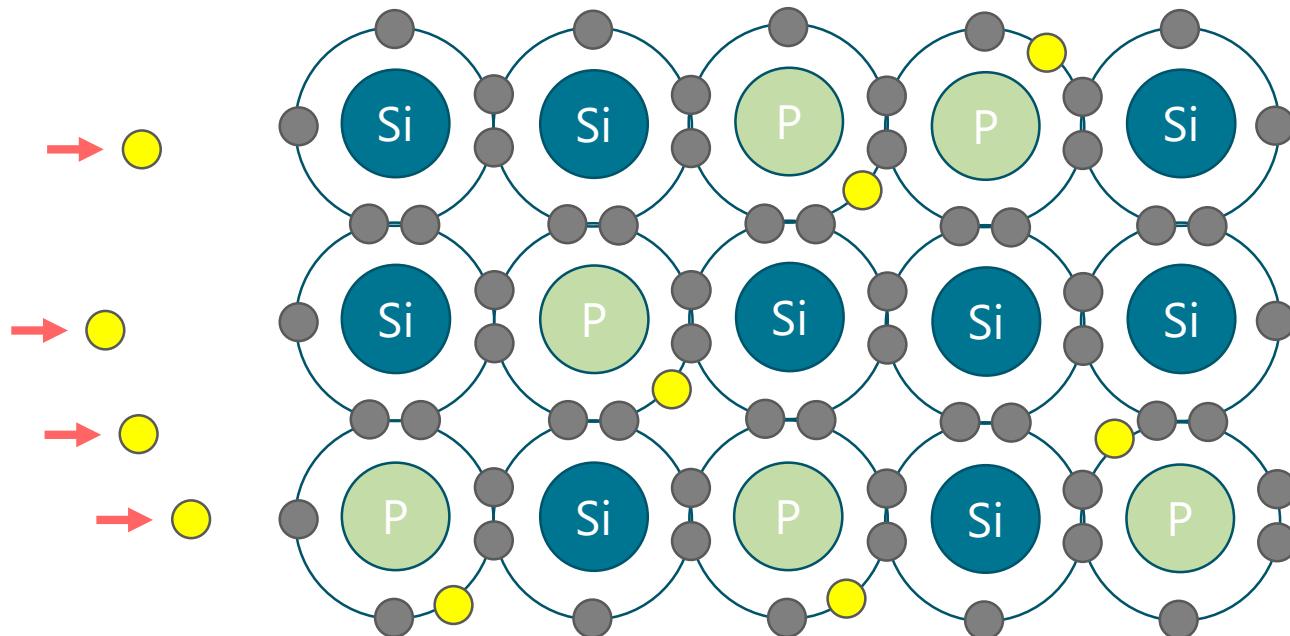
# Undoped Semiconductors Are Insulators



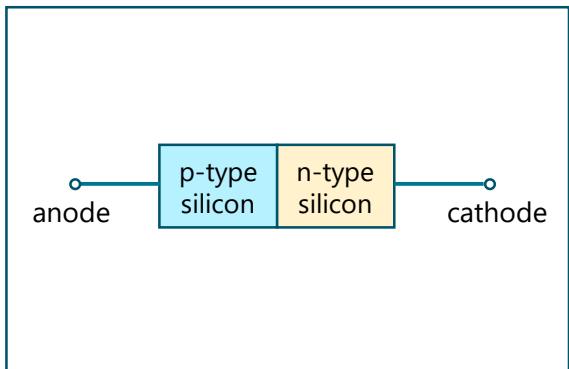
# Doping with Boron



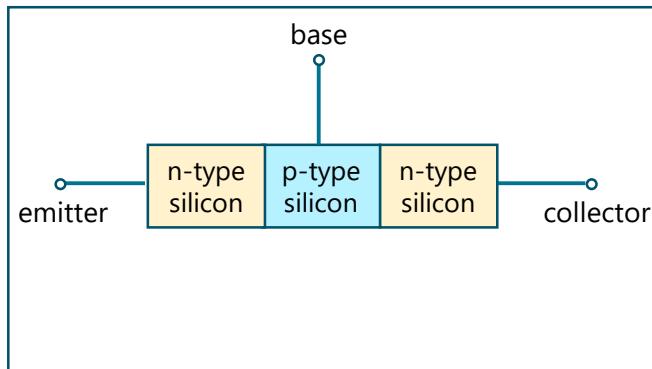
# Doping with Phosphorus



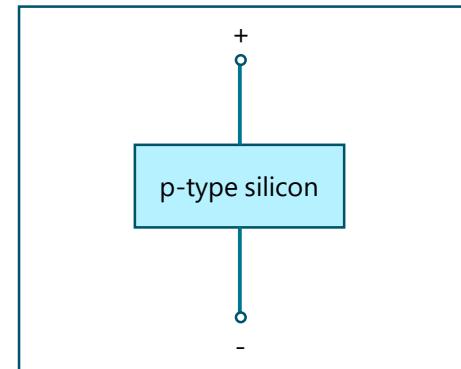
# How to “Print” Components



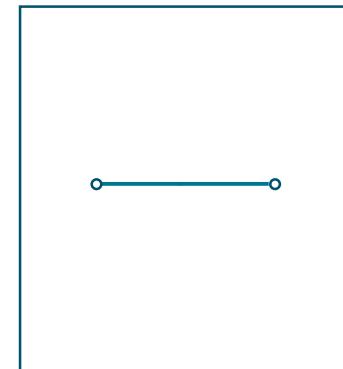
**Diode**



**Transistor**



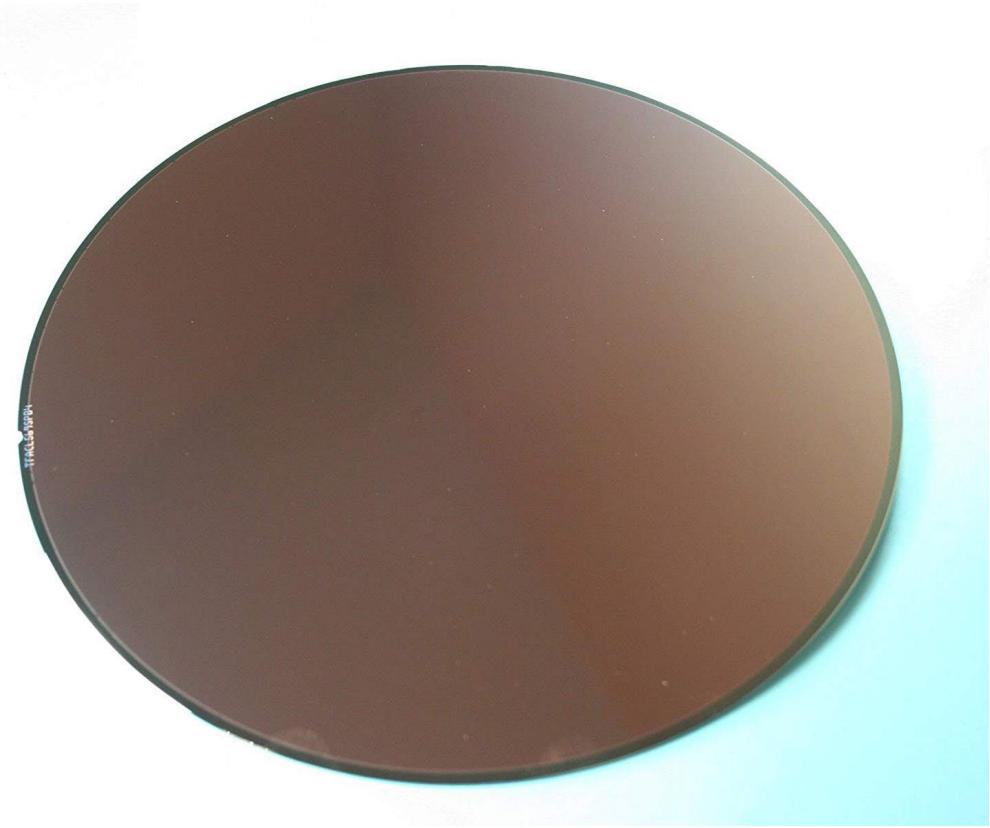
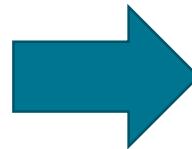
**Capacitor**

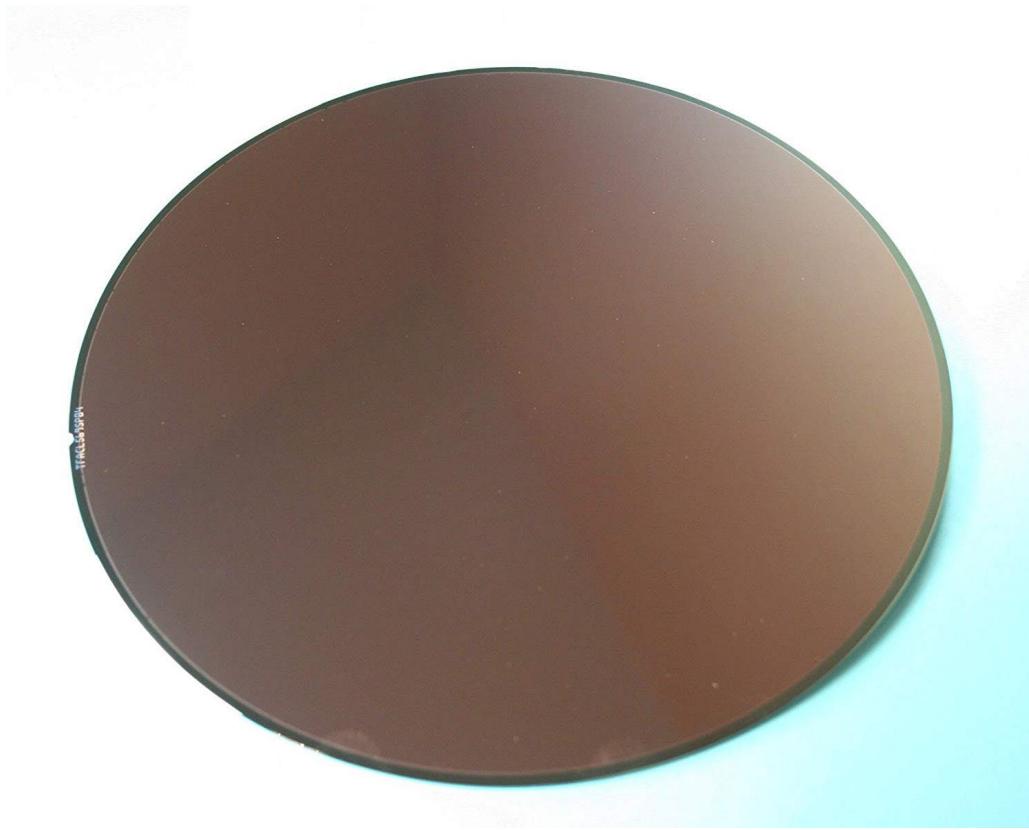


**Resistor**

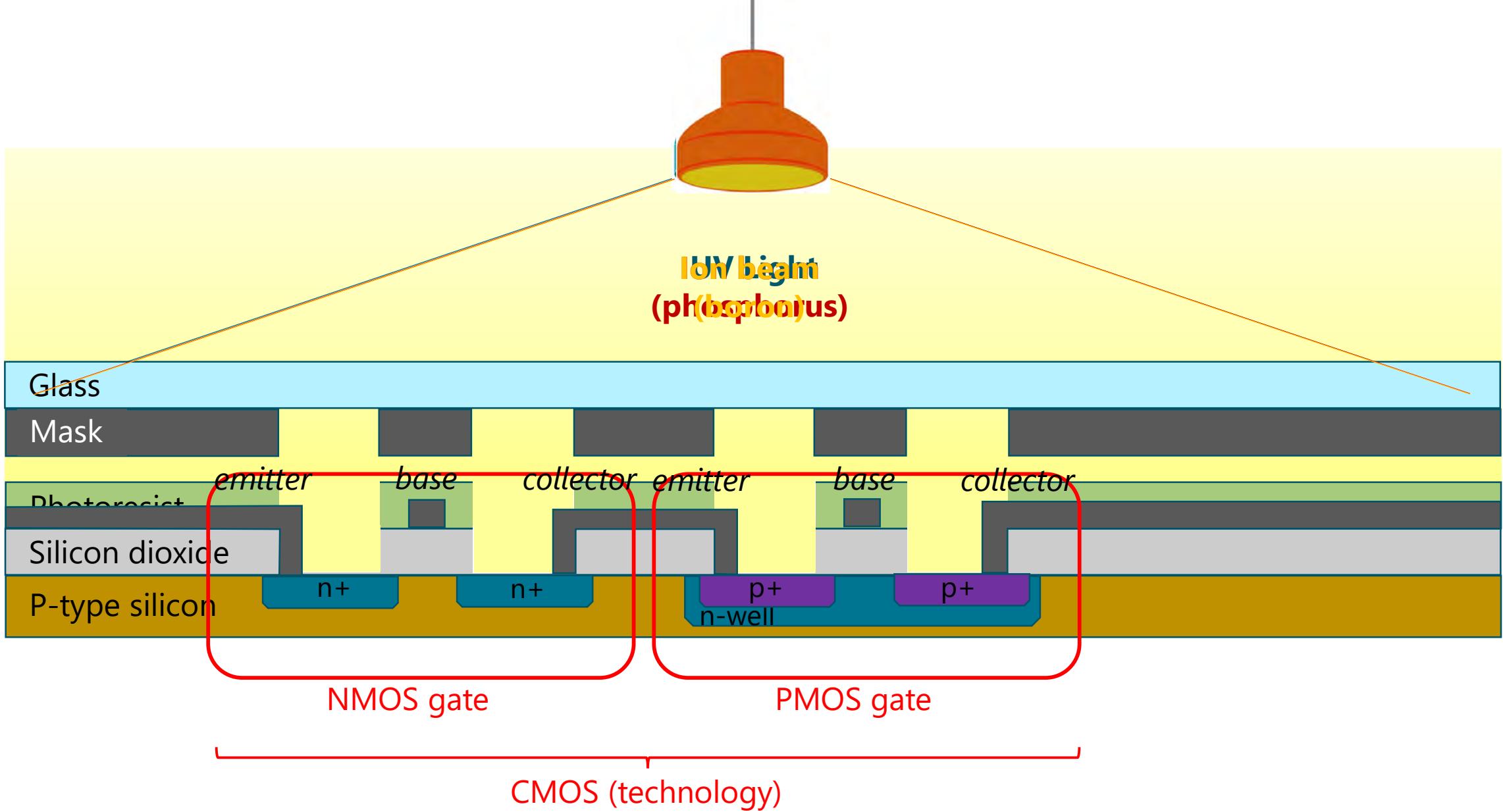
- Can “draw” (etch) tiny components using “doping”
  1. Lithography: draw your design onto photomask, apply to surface of silicon wafer
  2. Etching: “carve” your design into the silicon
  3. Deposit metal for connections

# How ICs Are Made

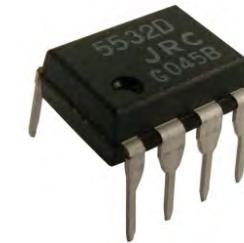
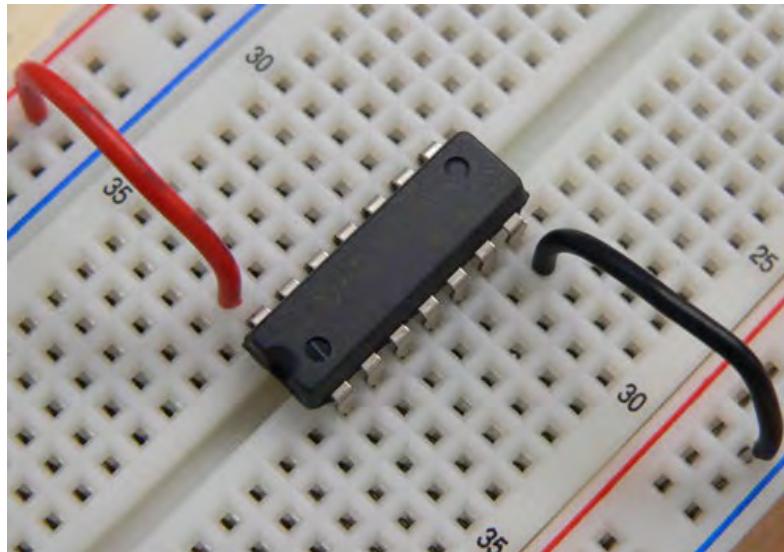








# Integrated Circuit Packaging



**Dual-Inline Package (DIP)**



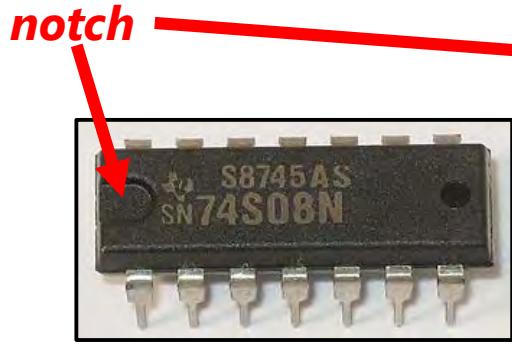
**Small-Outline IC (SOIC)**



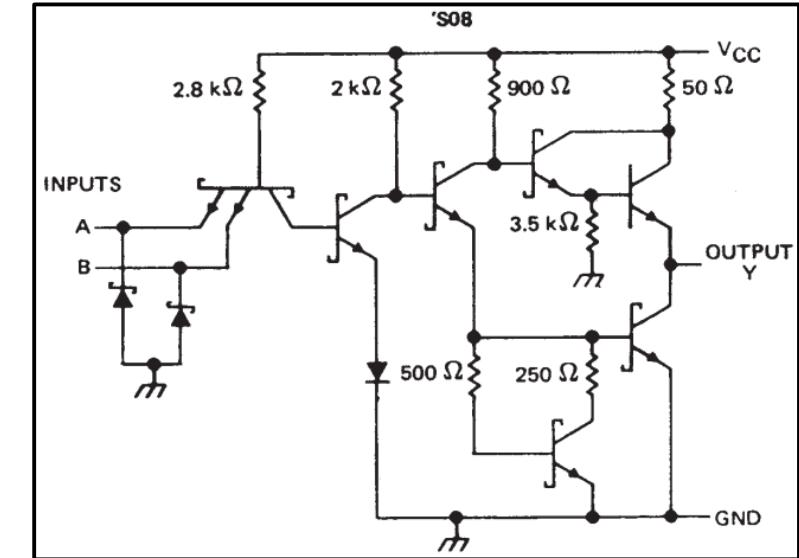
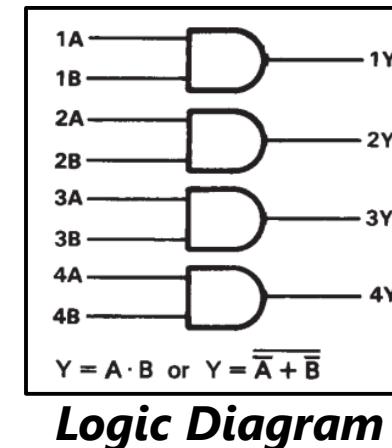
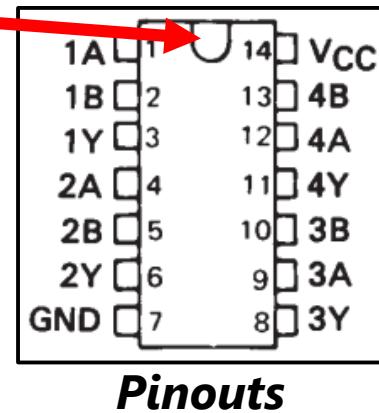
**Quad-Flat Package (QFP)**

**Surface-Mount Device (SMD)**

# Example IC: Texas Instruments SN74S08N (Quad AND Logic Gates)



**SN74S08N (DIP)**



**Schematic (each AND gate)**

- Datasheet: <https://www.ti.com/lit/ds/symlink/sn74ls08.pdf>
  - Google "SN74S08N datasheet"
- Price: \$1.61 for one, \$0.65 each for 5000 (Digikey.com, January 2019)

# Popular IC Types

IC family	Type	Pinout	Logic diagram	What it does	Alternatives/ Variants
-----------	------	--------	---------------	--------------	---------------------------

# Popular IC Types

IC family	Type	Pinout	Logic diagram	What it does	Alternatives/ Variants
-----------	------	--------	---------------	--------------	---------------------------

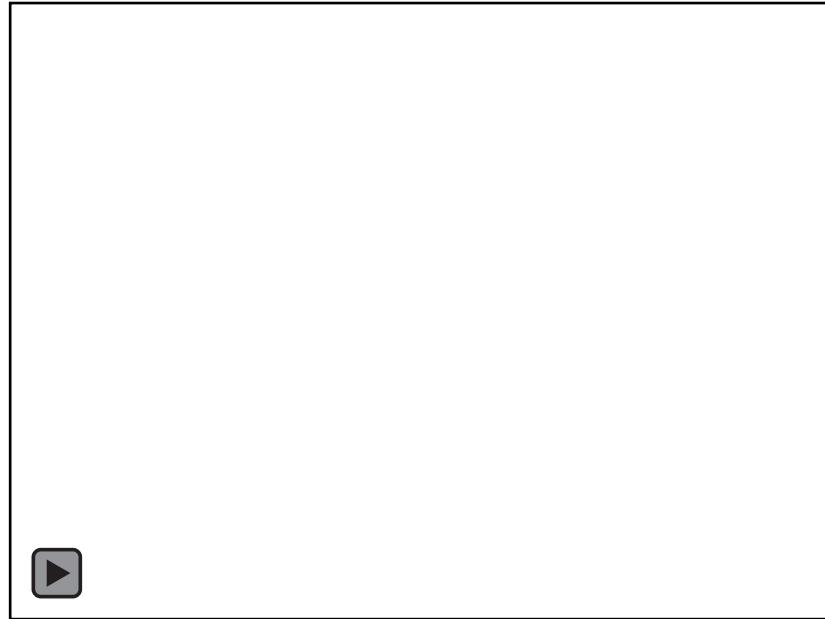
# Popular IC Types

IC family	Type	Pinout	Logic diagram	What it does	Alternatives/ Variants
-----------	------	--------	---------------	--------------	---------------------------

# So, can we just make everything an IC?

- ICs are great
  - “Componentize” circuits – don’t need to understand insides so much
  - Reduced cost
- But can’t do everything as an IC
  - High current/voltage – exceeds logic levels
  - Low volume or too new – no ICs exist for it
- Can we componentize non-IC circuits?

# Breakout Boards



- Standalone board that componentizes a circuit
  - “Breaks out” key pins to simplify prototyping
  - Deals with complicated EE stuff, lets you focus on using the circuit
  - Incorporates collection of components onto a single PC
- Example: Adafruit 9-DOF Absolute Orientation IMU Fusion Breakout – BNO055 = \$34.95 (January 2019) (Source: <https://www.adafruit.com/product/2472>)

# Breakout Boards

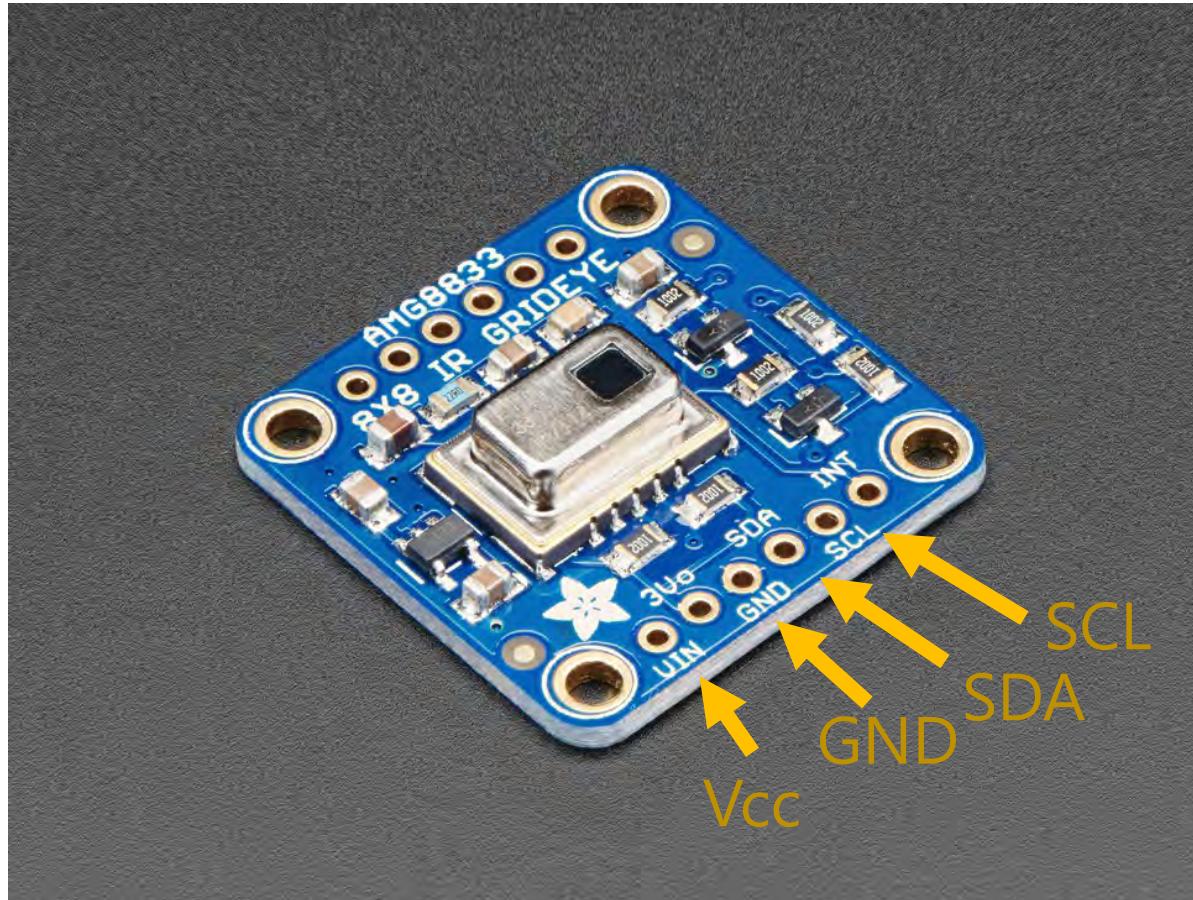
- Benefits: reduces costs, saves space, improves efficiency/reliability, simplifies reuse, clear pinouts, good documentation
  - Lets you do a lot with less domain knowledge
  - Speeds prototyping
  - Saves you soldering work
- Challenges: may not be interchangeable/compatible (e.g., may target a specific platform like Arduino), may need to solder or install header pins, need to be careful about supply voltage and other inputs

# Example: Adafruit AMG8833 IR Thermal Camera Breakout

- 8x8 array of thermal IR sensors
  - Can detect human 23 feet away
  - 10 Hz frame rate
- **Uses:** human detector, thermal camera, intruder alert
- **Incorporates:** Panasonic AMG8833 thermal IR sensor, 3.3V regulator, misc. resistors/capacitors



# Example: Adafruit AMG8833 IR Thermal Camera Breakout



- **Pinout:**

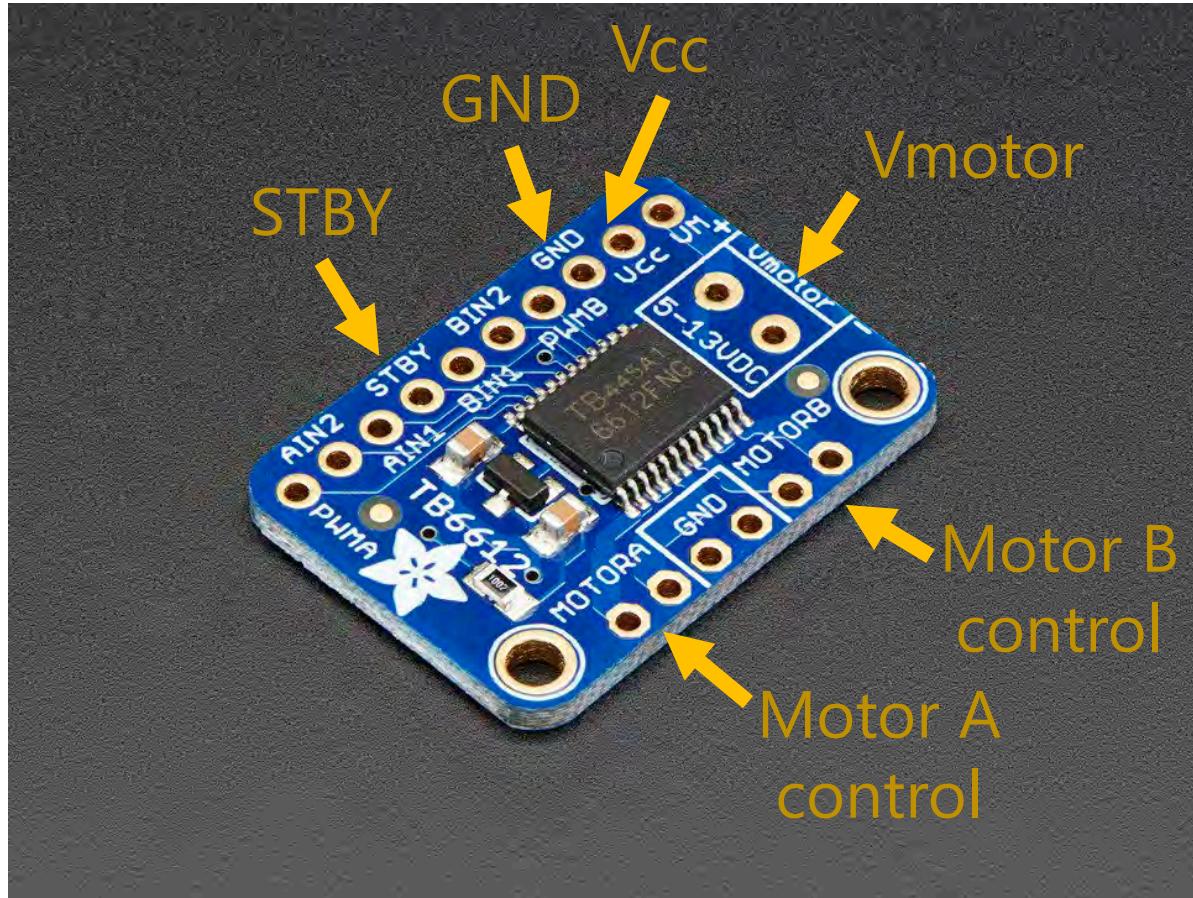
- Input voltage (Vin)
- Output voltage in case you can use it (3Vo)
- Ground (GND)
- Serial clock – output (SCL)
- Serial data – output (SDA)
- Interrupt output – when something changes in vision path (INT)
- (Pins at top for stability, unused)

# Example: Adafruit TB6612 1.2A DC/Stepper Motor Driver Breakout Board

- Driver for stepper motor
  - Can drive 1.2A sustained, 3A peak (20ms)
  - Control direction and speed, stop/brake
  - Motor voltage isolated from logic voltage
- **Uses:** robot, pan/tilt housing, automotive gauge/dial, toy car
- **Incorporates:** a Toshiba TB6612 motor driver IC, two H-bridges, polarity protection FET, pullup resistor for inputs, kick-back diodes



# Example: Adafruit TB6612 1.2A DC/Stepper Motor Driver Breakout Board



- **Pinout:**

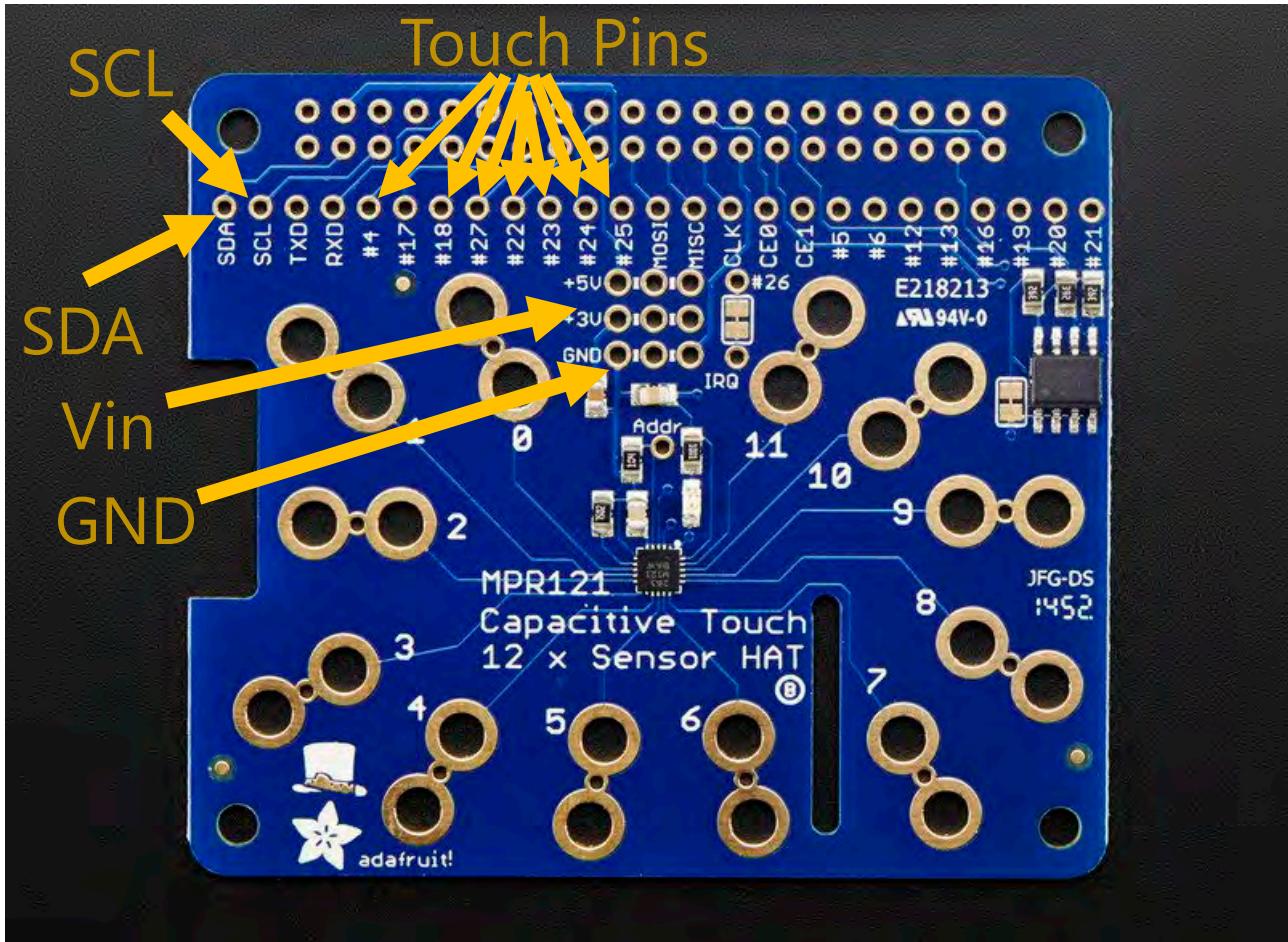
- Input voltage for logic levels (Vcc)
- Voltage to drive motor (Vmotor)
- Ground (GND)
- Motor control outputs:
  - INA1, INA2, PWMA: outputs for motor A's H-bridge
  - INB1, INB2, PWMB: outputs for motor B's H-bridge
  - STBY: quickly disable both motors

# Example: Adafruit Capacitive Touch HAT for Raspberry Pi – Mini Kit – MPR121

- Driver for detecting touch
  - Attach pins to something conductive (metal, something containing water)
  - “HAT” simplifies connecting to Raspberry Pi platform
  - Comes with python library
- **Uses:** touch-reactive tablets, control panels, and other objects
- **Incorporates:** a MPR121 12-key touch sense microcontroller



# Example: Adafruit Capacitive Touch HAT for Raspberry Pi – Mini Kit – MPR121



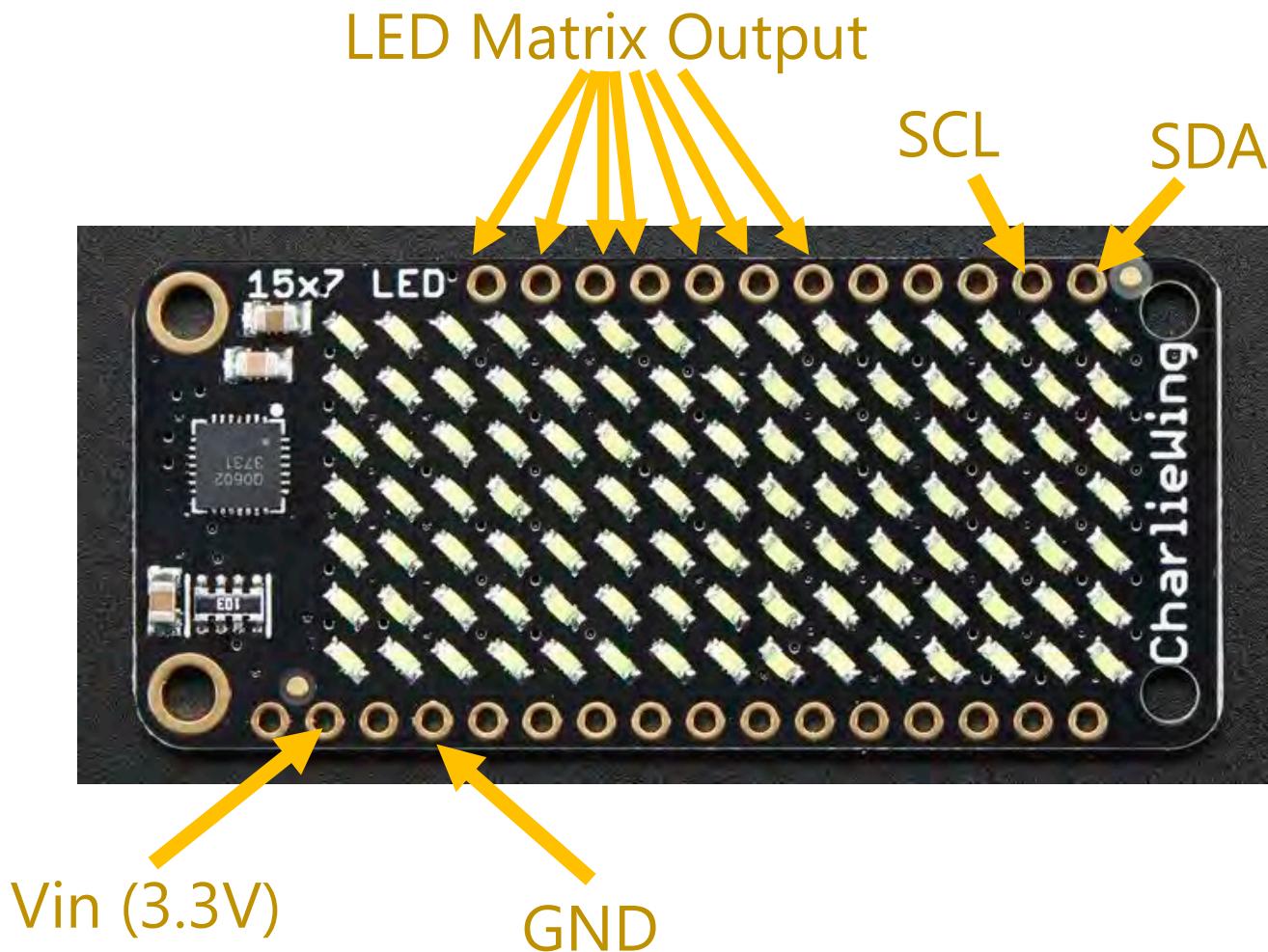
- **Pinout:** (look familiar?)
  - Input voltage (Vin)
  - Ground (GND)
  - Serial clock – output (SCL)
  - Serial data – output (SDA)
  - Touch-capacitive pins (#)

# Example: Adafruit 15x7 CharliePlex LED Matrix FeatherWing – Cool White

- Drives matrix of LEDs
  - Tell chip which LEDs you want lit and their brightness, and it will PWM them for you
  - RAM for eight separate frames, can animate through them with single command
  - Comes with python graphics library
- **Uses:** displays, status indicators, human-machine interfaces
- **Incorporates:** a IS31FL3731 I<sup>2</sup>C LED driver chip



# Example: Adafruit 15x7 CharliePlex LED Matrix FeatherWing – Cool White



- **Pinout:** (look familiar?)
  - Input voltage (Vin)
  - Output voltage in case you can use it (3Vo)
  - Ground (GND)
  - Serial clock – input (SCL)
  - Serial data – input (SDA)
  - LED matrix output pins

# Encoding

# Sending Bits on Wires

- Sometimes we need to take data and send it on wires
  - Between chips on a PCB, between different PCBs within a device, between devices in a home, between devices in different buildings
- Need some way to “encode” bits
- Idea: change some properties of the wire that can be read by the other side

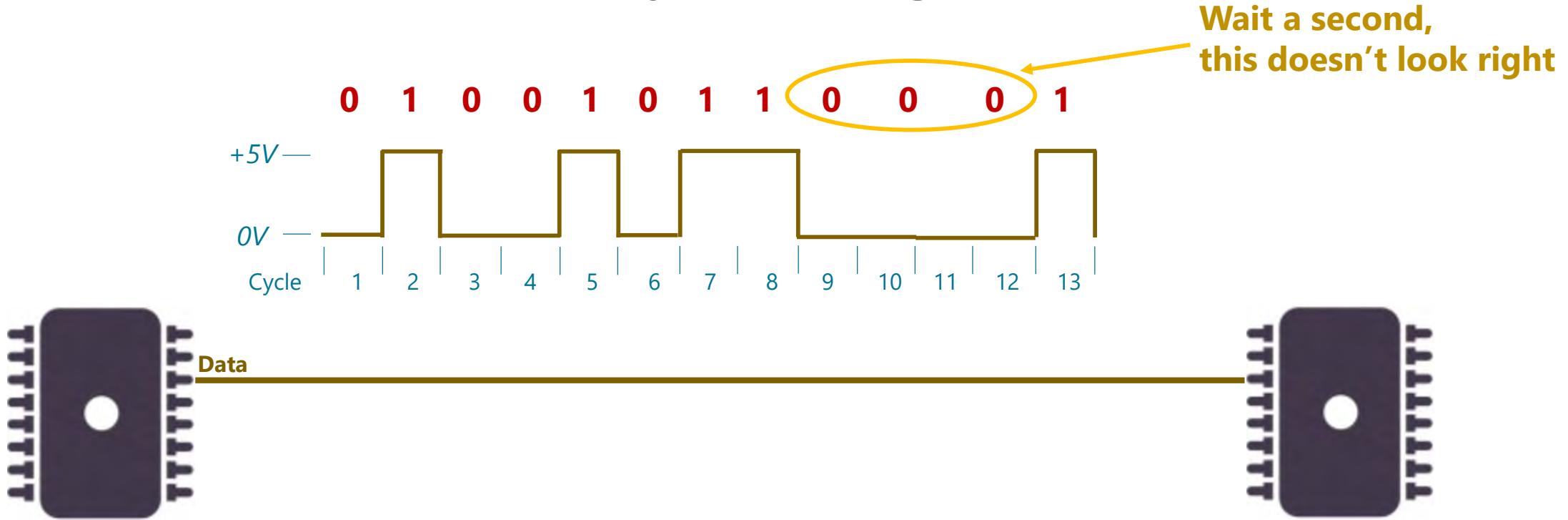
# What Properties to Change?

- Really only two options: **current** and **voltage**
- **Voltage** turns out to be much more efficient
  - Current changes over wire length, requires more electricity

# What Voltage Levels Should We Use?

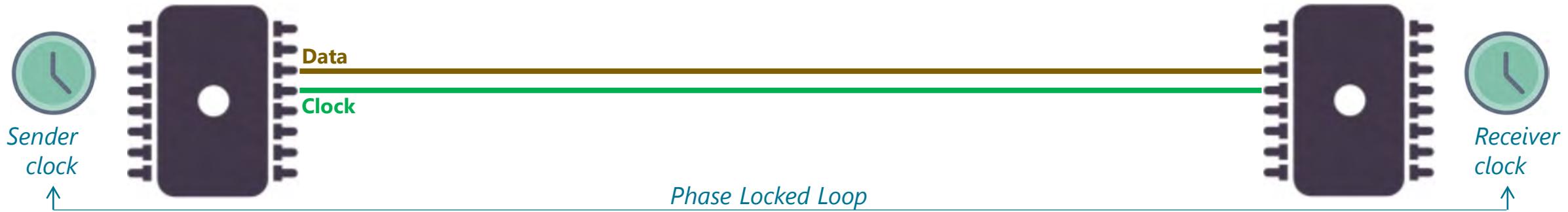
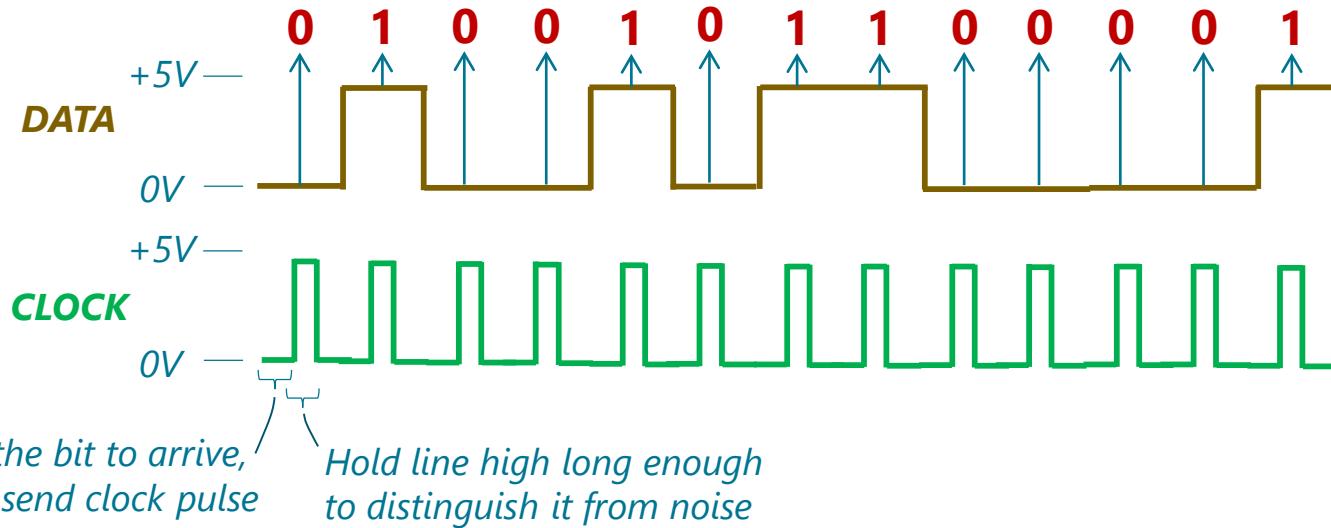
- Lower levels reduce power consumption, improve safety
- Higher levels reduce sensitivity to noise, loss over distances
- Specific values (15V, 12V, 5V, 3.3V) are common for historical reasons
  - Large existing market reduces component cost

# Two Chips Can Vary Voltage to Send Data



- One approach: raise voltage for a 1, lower it for a 0
  - Other side decodes: periodically sample voltage on wire
- Problem: clock synchronization

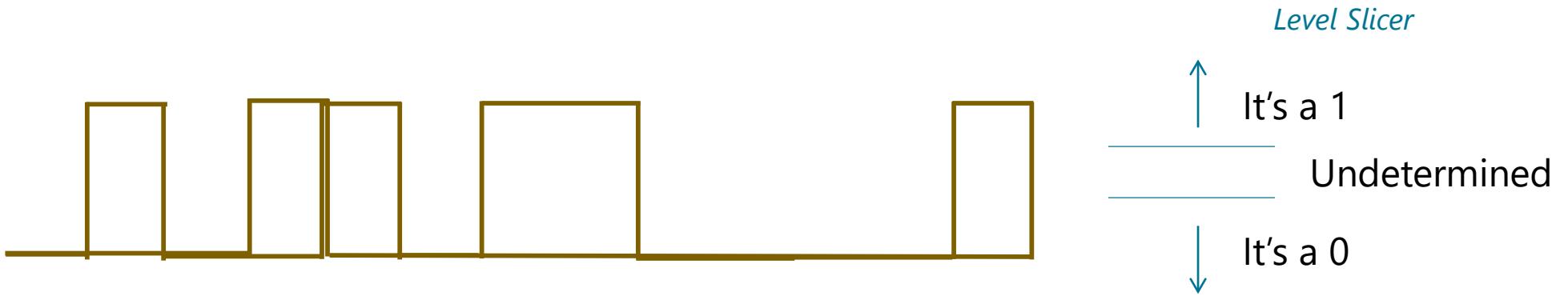
# Two Chips Can Vary Voltage to Send Data



- **Another approach:** have a separate clock line
  - Use pulses to signal when receiver should sample line

- **Problems:** need additional wire (increases costs, increases faults because faults/noise in *\*either\** wire cause loss)
  - Also need faster components (if clock transition rate exceeds that of data)

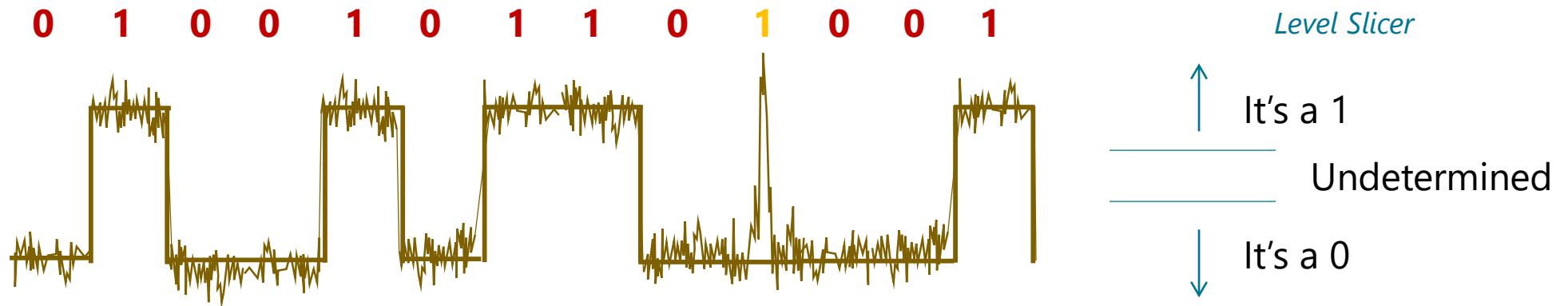
# A Few Other Issues: DC Balance



Unequal numbers of 0s and 1s can introduce bias in level detection circuitry

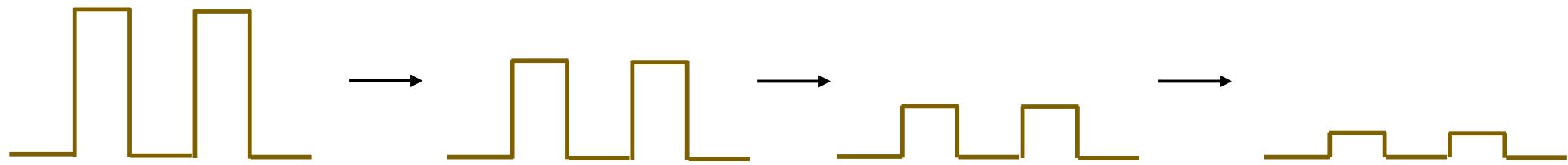
- In unbalanced channel, DC component shifts cause slicer to respond at level different to data average
- Also makes it hard to galvanically isolate (separating two circuits due to noise/safety, e.g., with capacitance, EM waves, etc.)
- Also requires more transmission power (need to send DC to receiver)

# A Few Other Issues: Electronic Noise



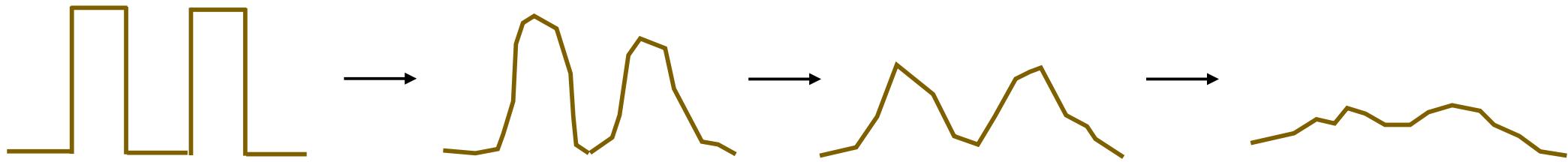
- May introduce false readings
- **Sources:**
  - **Thermal noise:** free electrons vibrate more at higher temperatures
  - **Shot/flicker/burst/transit-time noise:** noise from electrons traversing components
  - **Coupled noise:** noise leaked from adjacent wires/atmosphere/cosmic
  - **Leakage noise:** motors/speakers/bulbs/etc. induce voltage spikes), etc.

# A Few Other Issues: Loss/Attenuation



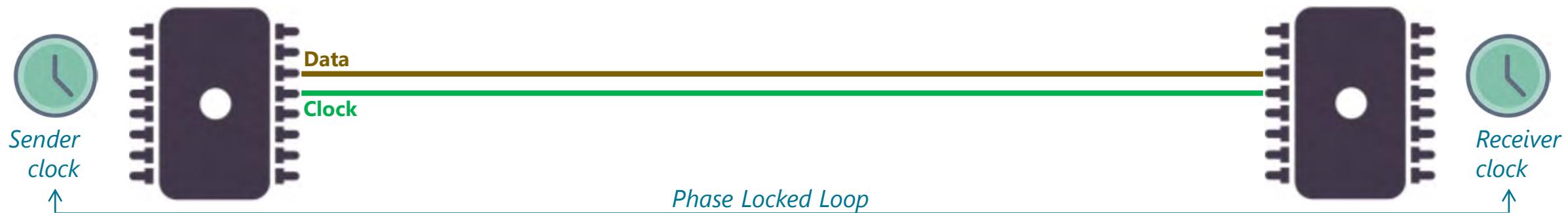
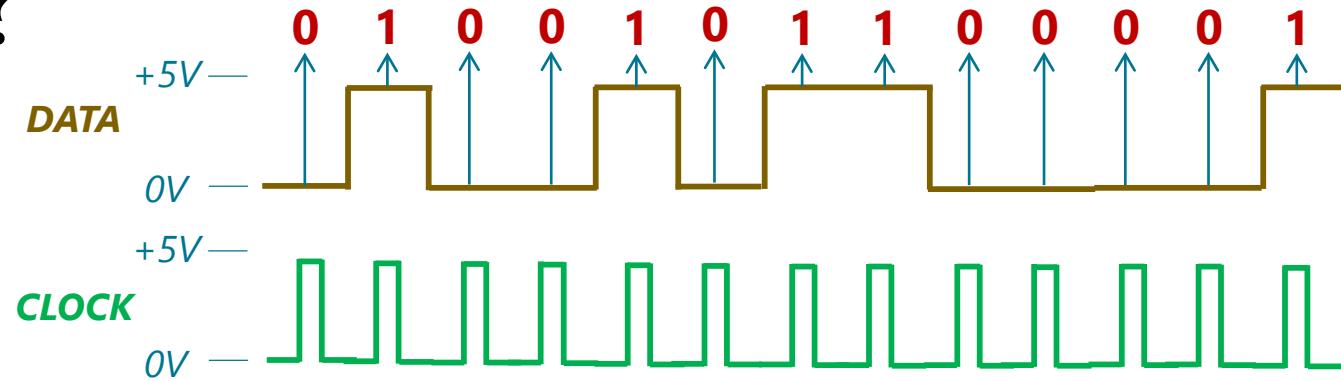
- Wires/components have resistance (impedance)
  - Reduces current and voltage. Amount proportional to length of wire
  - Larger-diameter cables/wires have less resistance per foot
  - Components also have impedance, more you traverse the worse it gets
- Impedances sum when connected in series, inversely sum with inverses when connected in parallel

# A Few Other Issues: Dispersion



- Not all electrons in the pulse go the same speed
- Bits melt into each other, making them hard to distinguish
- Some components have more dispersion than others (e.g., low-pass filters)

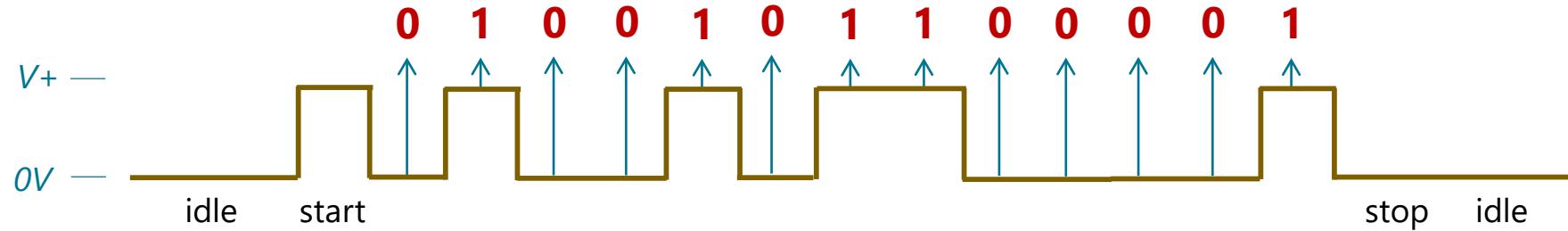
# Ok, so can we at least get rid of the extra wire?



- Idea: receiver can sync clock off transitions

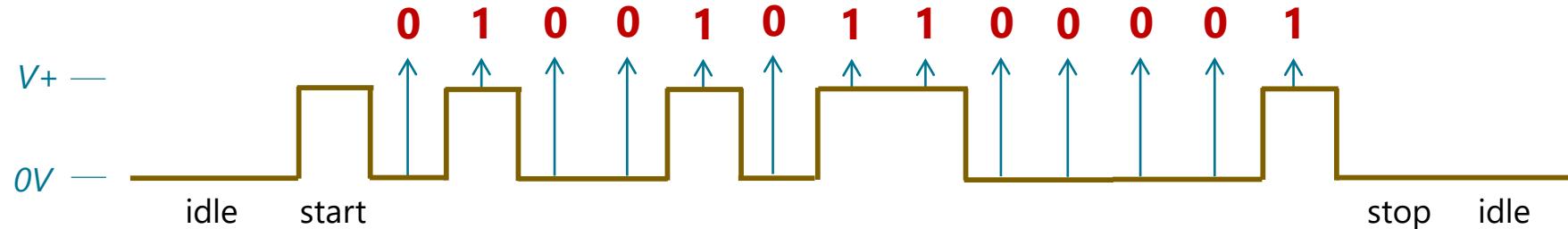
- Problem: What if not enough transitions?
  - Long runs of 1s or 0s
  - Clocks could get out of sync

# Idea: Re-sync on Each Short Message (Asynchronous Communication)



- Start and stop signals appended before/after each unit of transmission
  - Start signal used to set receivers' clock
  - Need to agree on parameters in advance (full vs. half-duplex, # bits per character, endianness, baud rate, whether to use parity, odd vs. even parity, # of stop bits)

# Idea: Re-sync on Each Short Message (Asynchronous Communication)

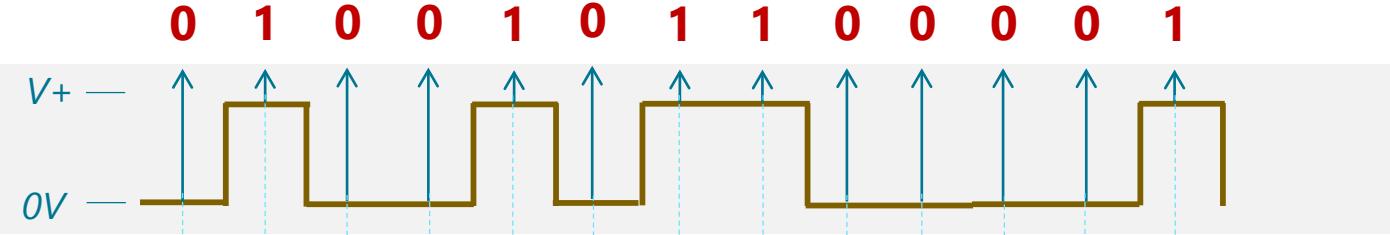


- Advantages: doesn't require continuous synchronization → cheaper hardware, faster session setup
  - Good for applications where messages are generated at irregular intervals: user input (e.g., keyboards), sensors, etc.
- Downsides: large overhead (many bits just used for control)
- Used in: RS-232/UART/PPP/etc.

# Another Idea: Force There to Be More Transitions

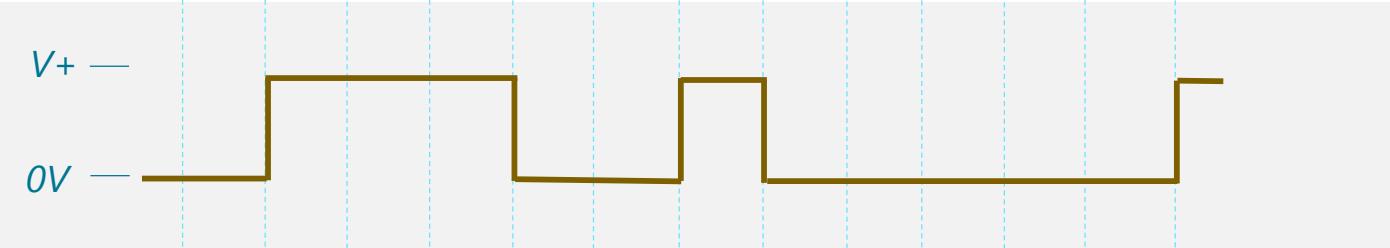
## NRZ [Non-Return to Zero]:

High voltage for 1, low voltage for 0  
(what we've been seeing)



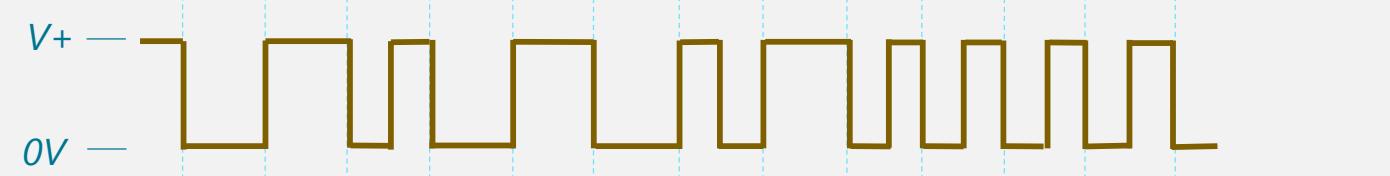
## NRZI [Non-Return to Zero Inverted]:

Transition for 1, no transition for 0. NRZI is a "differential encoding," as value depends on preceding bits (increases transition density)



## Manchester (IEEE 802.3):

Up transition for 1, down transition for 0 (further increases transition density, but requires faster clock)

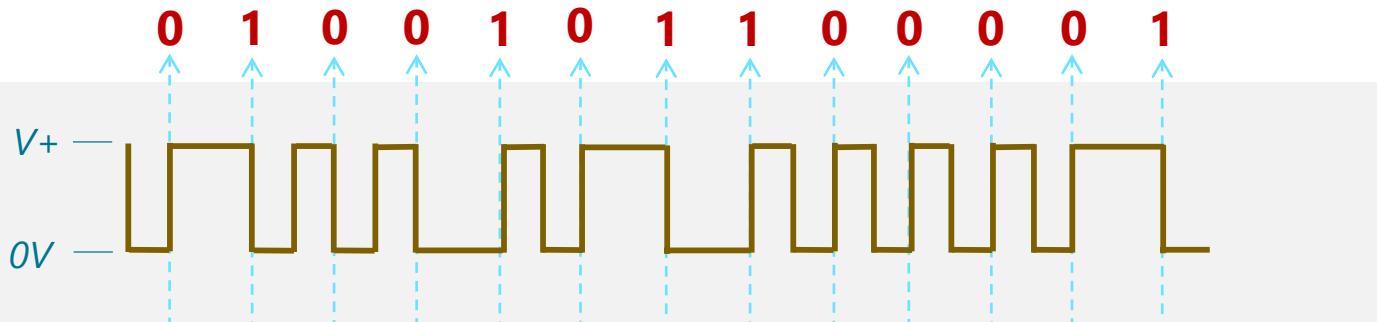


- Use transitions rather than voltage levels to encode data
  - Cause more cycles → improve clock synchronization
  - Transitions are also easier to detect than levels

# A Few Other Approaches

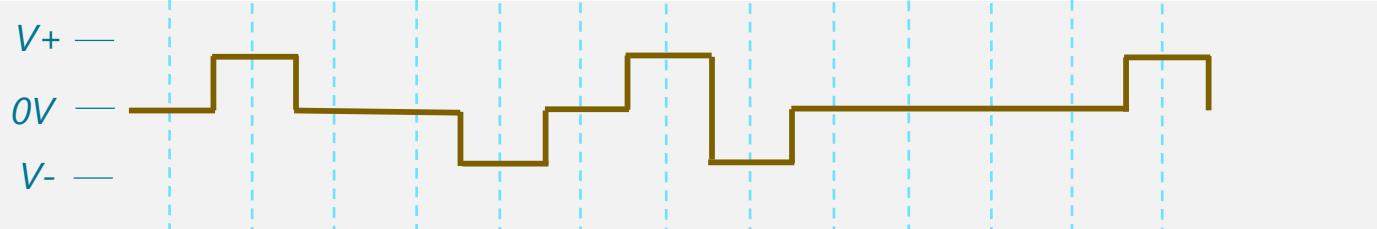
## Differential Manchester:

Transition either up or down each cycle. Repeat previous transition if 0, invert if 1. Unlike Manchester, works the same if lines swapped



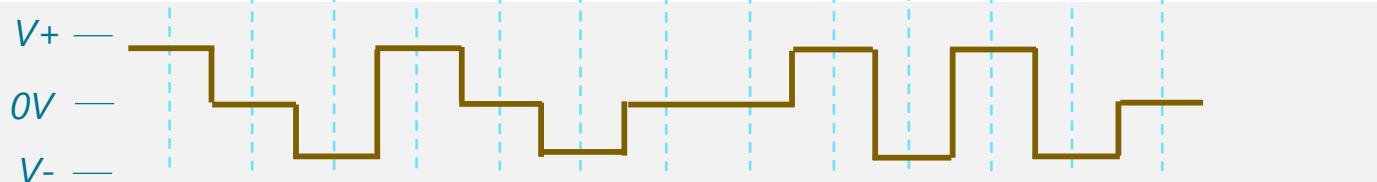
## Bipolar Alternate Mark Inversion (AMI):

Three levels; 0 is no voltage, 1s alternate between positive and negative voltage levels. (doesn't require faster clock, but "wastes" levels)



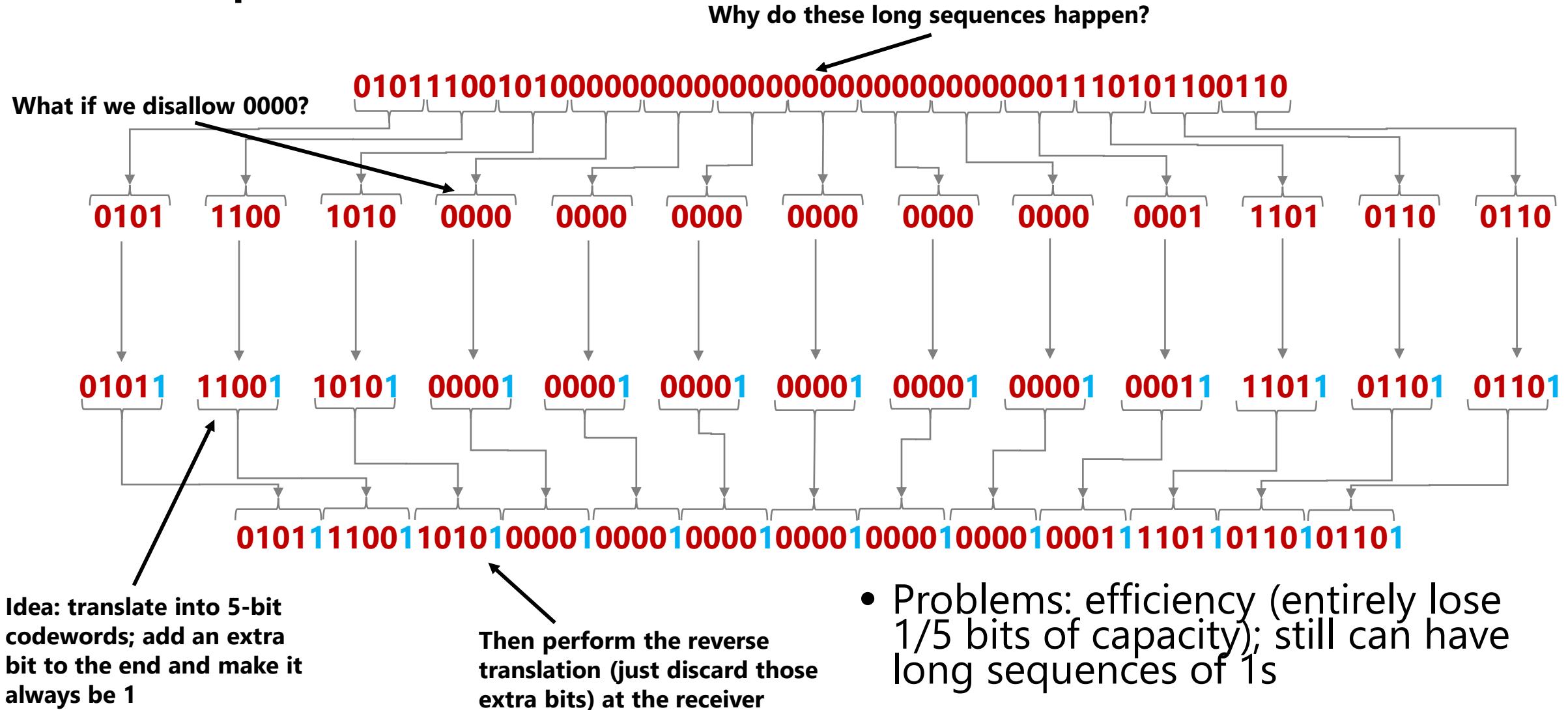
## Pseudoternary:

Opposite of Bipolar AMI; 1 is no voltage, 0s alternate between positive and negative voltage levels



- These are pretty good but can we do better?
  - Manchester/Differential Manchester requires 2x clock
  - Bipolar AMI/Pseudoternary requires more levels

# Idea: Map onto Codewords with Nice Properties



# Better Approach: 4B/5B (Choose Codewords Smarter)

- Encode every 4 consecutive bits as a 5-bit word
- Codewords
  - At most one leading 0, at most two trailing 0s
  - Never more than three consecutive 0s
  - Transmit with NRZI

# Better Approach: 4B/5B (Choose Codewords Smarter)

- 16 of 32 possible codes used for data, others used for command characters
  - Ensures at least two transitions for each code
- Variants
  - 8B/10B (used in digital audio – compact discs, DAT tapes, etc.)
  - 64B/66B (used in 10Gig Ethernet)
  - 128B/130B, 128B/132B (used in PCI Express 3.0 and USB 3.1)

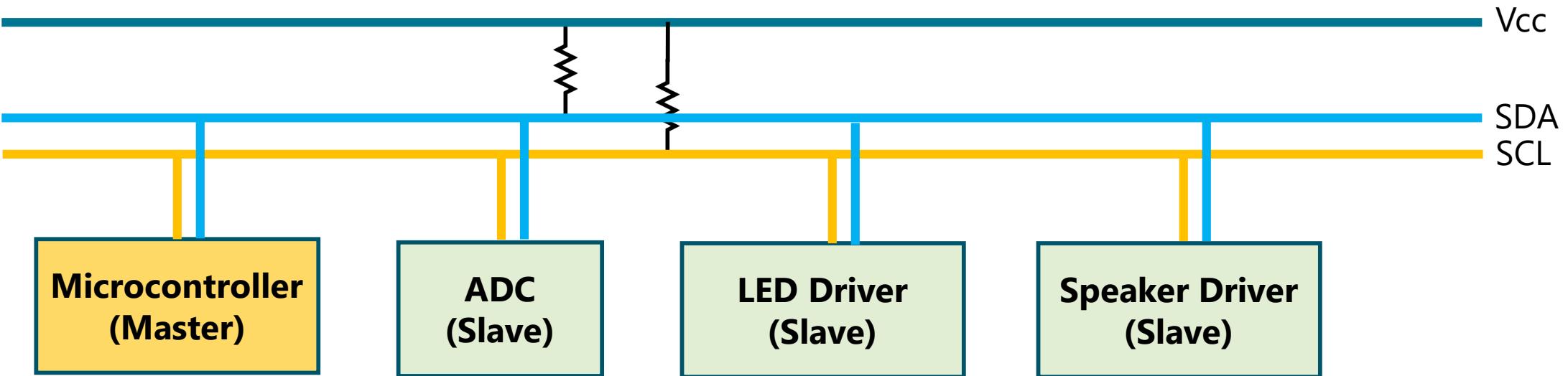
# 4B/5B Encoding Table

At most two trailing zeros	
0000 →	11110
0001 →	01001
0010 →	10100
0011 →	10101
0100 →	01010
0101 →	01011
0110 →	01110
0111 →	01111
At most one leading zero	
1000 →	10010
1001 →	10011
1010 →	10110
1011 →	10111
1100 →	11010
1101 →	11011
1110 →	11100
1111 →	11101

# Where is this stuff used?

- Different encoding schemes are used in different settings
- Advanced/Complex encodings used for high-speed links between computers
  - Where it's worth investing in more complex circuitry
  - See more Manchester, 64B/66B, etc.
- Between chips on a PCB you care more about low power and cost
  - Simpler techniques, easier to run shared clock lines everywhere
  - See more asynchronous, clocked encodings, etc.

# Connecting ICs with the I2C Protocol



- Easy to use, very expandable, serial bus widely used to connect ICs
  - Reduces pinouts, transfers data/commands
- Address-oriented scheme, where addresses are part of packets that you send
  - Handles many devices; each device has an address (can solder jumpers to change address)
  - Message-based communication (unlimited length), 0.1-5Mbps, synchronous
- Multi-master/Multi-slave architecture: master generates clock, initiates communications

# Other Kinds of Buses

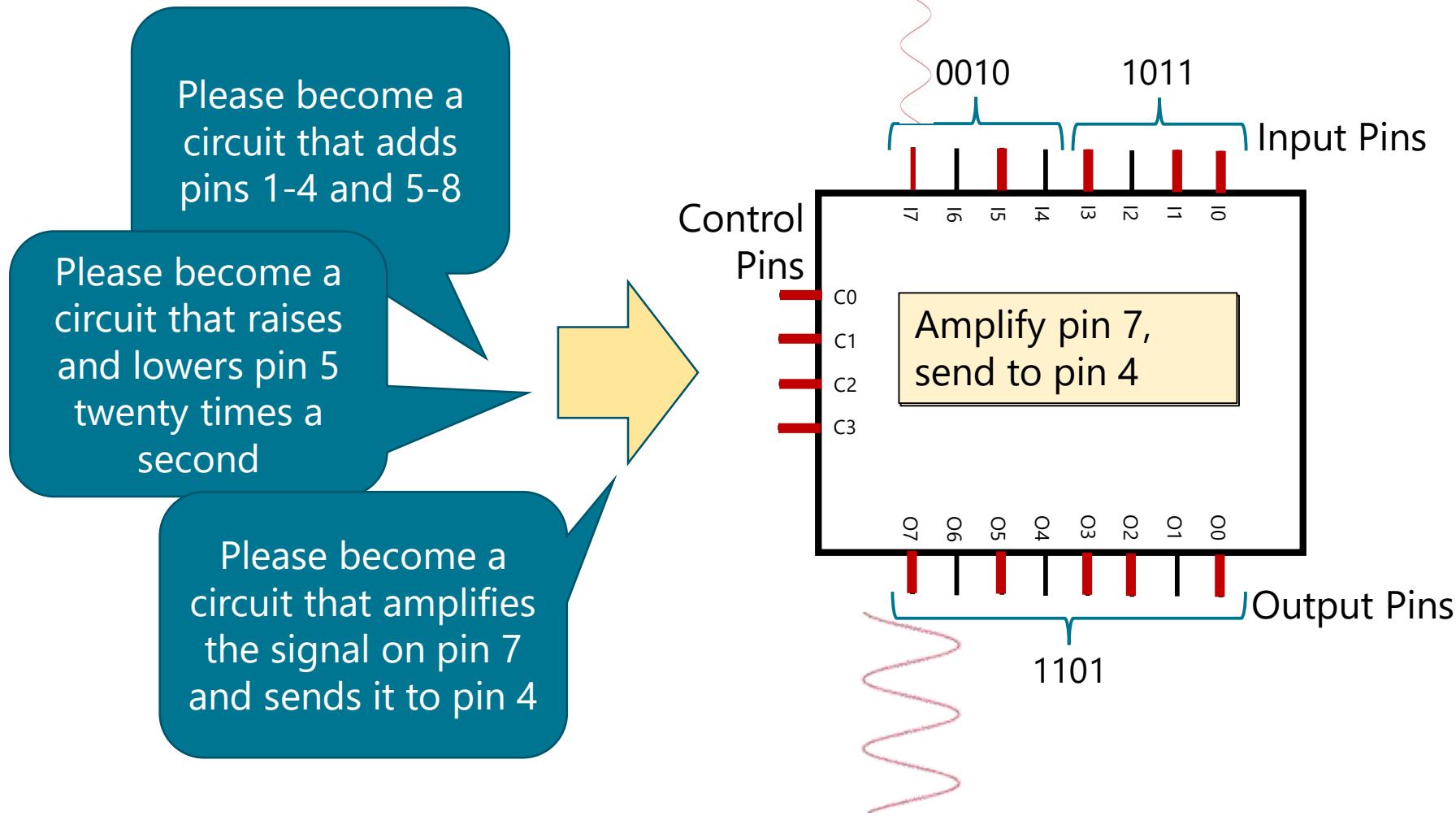
- **Serial Peripheral Interface (SPI)**
  - Max # devices very limited (chip-select for each chip on the bus)
  - Significantly faster than I2C (25Mbps), works over longer cable runs
  - Used for, eg. Connecting microcontroller to SD/microSD card
- **UART** (RS-232/422/485/etc., aka Serial Port)
  - Asynchronous (no shared clock)
  - Parameters: baudRate, parity, stopBits, dataBits (e.g., 115200, none, 1, 8)
  - Max speed 8Mbps
- **GPIO**
  - Not really a protocol – turning pins on and off, defines threshold voltages
  - Used for switches, buttons, motion detectors, LEDs, turning on/off appliances
  - Max speed 1kHz

# Example Breakout Boards

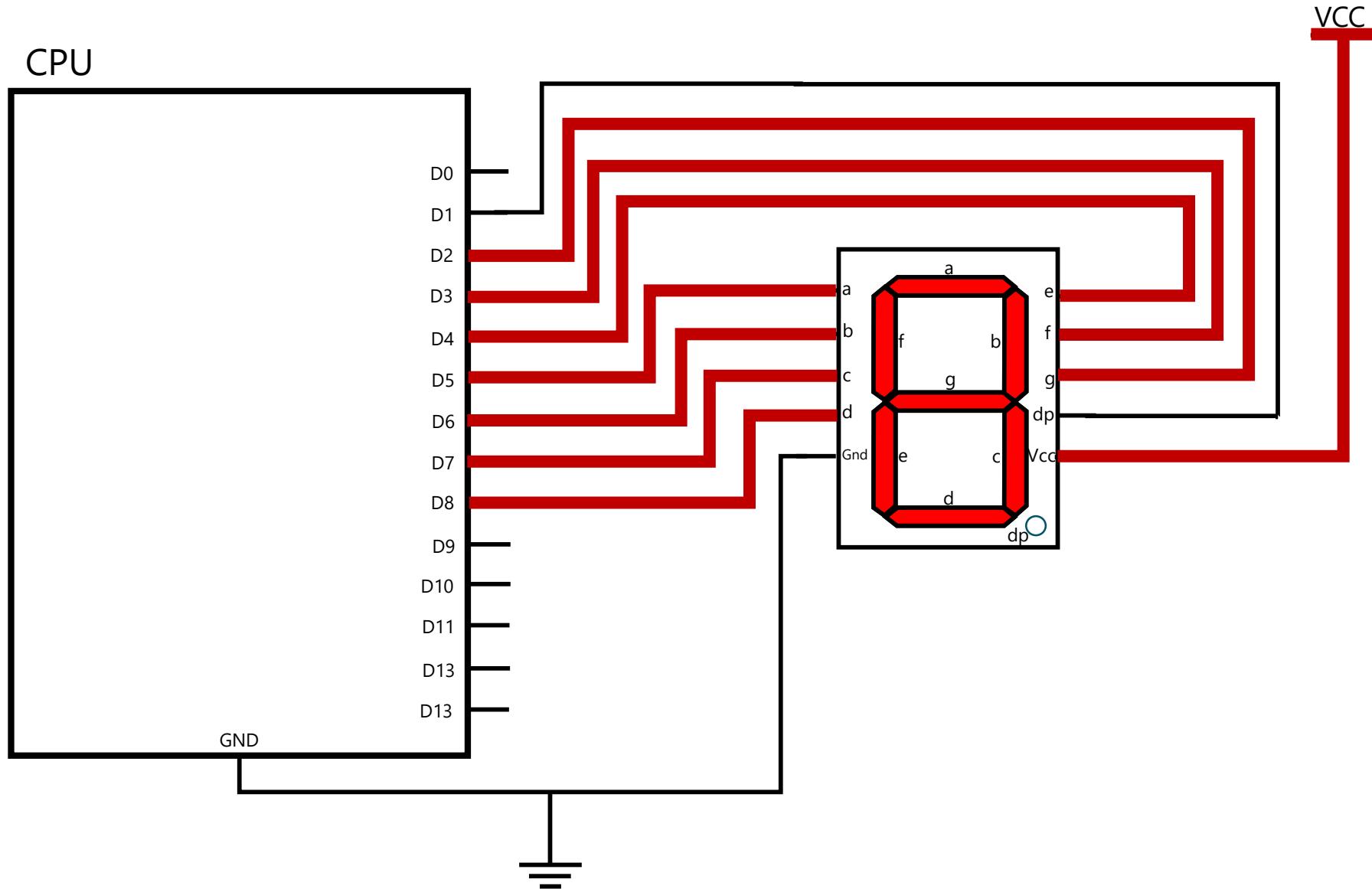
# What if we want to build something flexible?

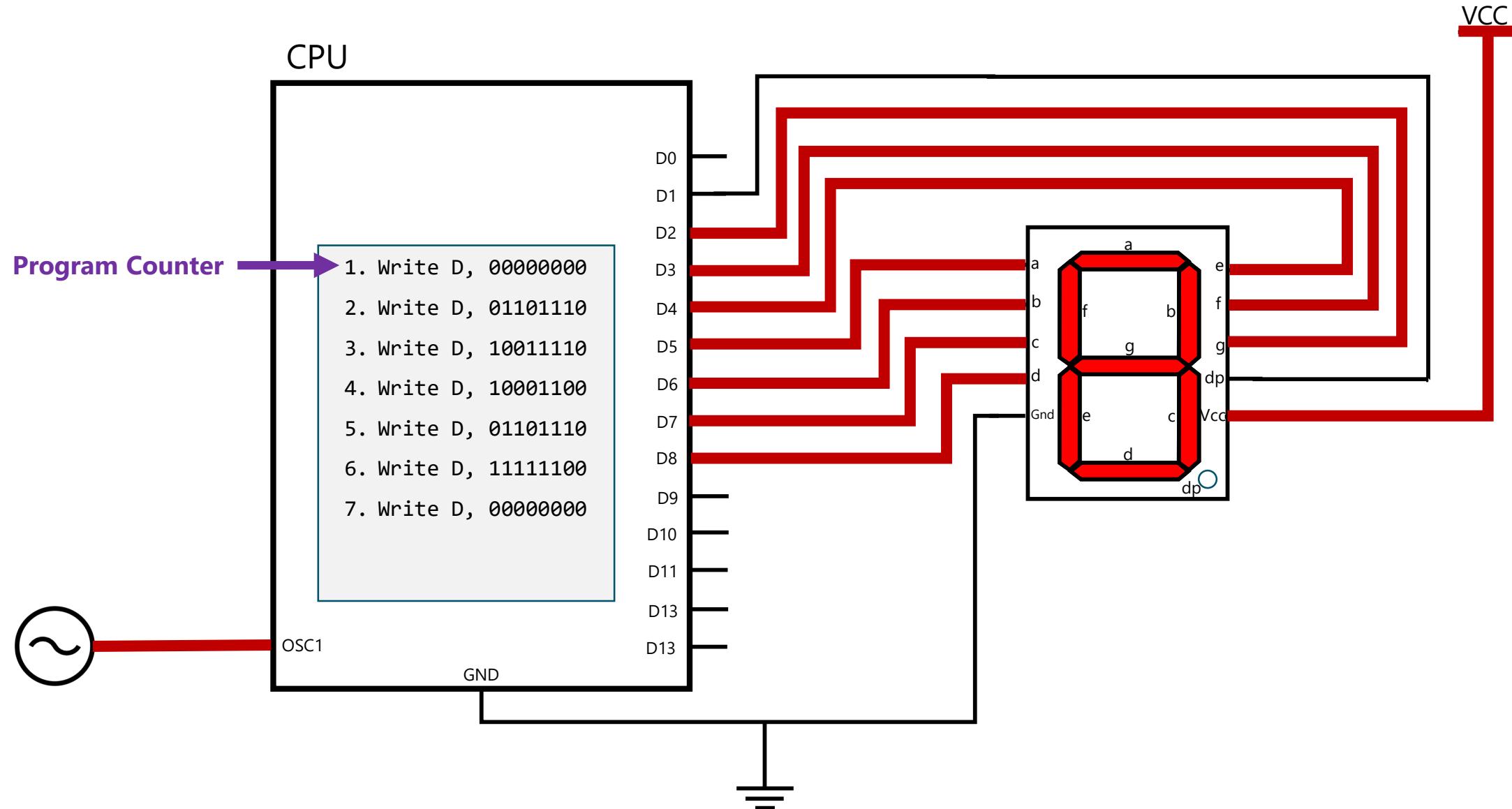
- Circuits are good for very specific, single-purpose functions
  - Increase voltage from 3.3V to 5V, drive a motor, etc.
- But what about
  - A car that recognizes human shapes?
  - A drone that flies a programmable path?
  - A thermostat that displays updatable pictures?
- Problem: need to deal with many contingencies

# Idea: A “Programmable IC”



CPU





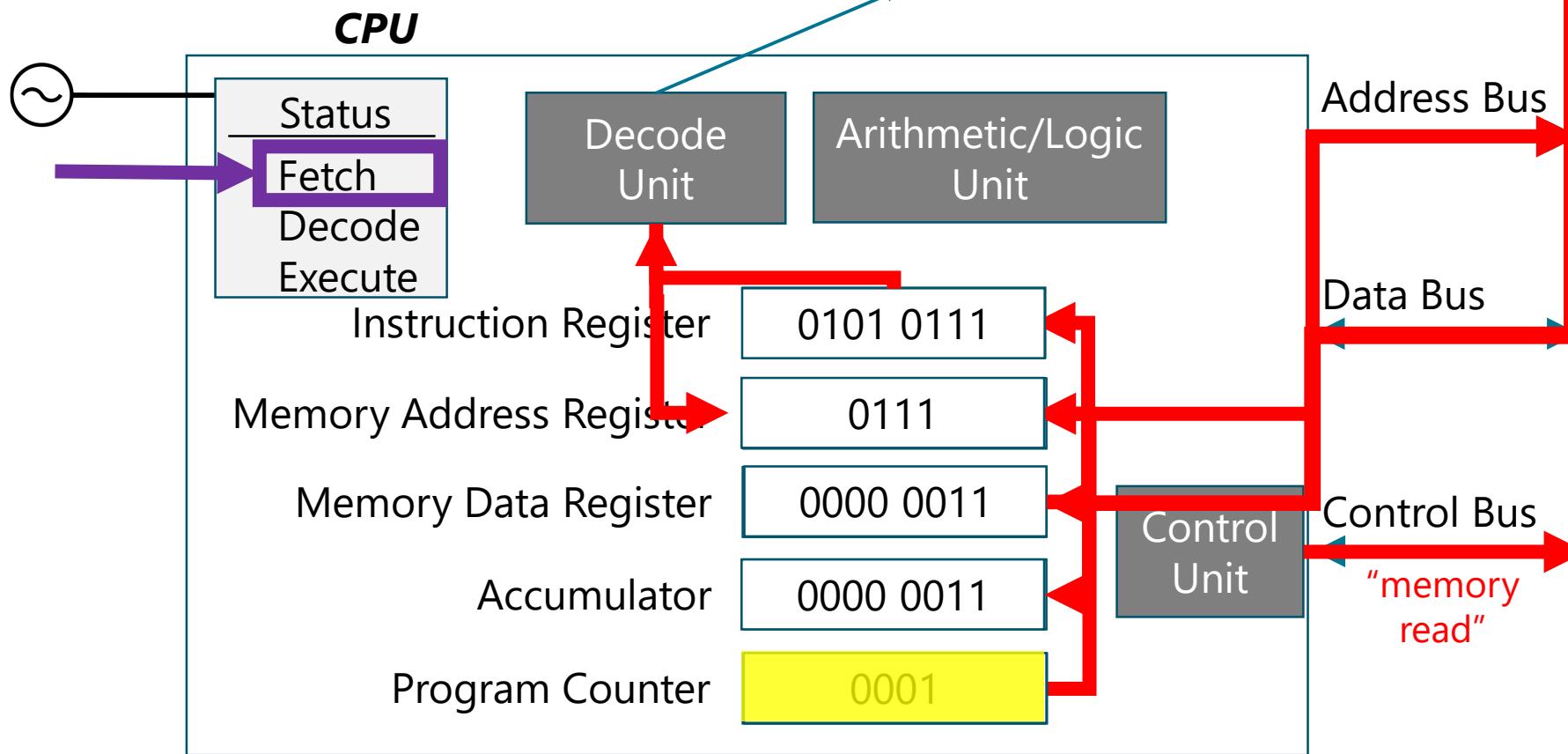
Opcode [Operand]	Instruction
0001 [address]	add
0010 [address]	subtract
0011 [address]	store
0101 [address]	load

**Memory**

Instruction memory	
Address	Value
0000	0101 0111
0001	0001 0110
0010	0011 0111

Data memory	
Address	Value
0111	0000 0011
1000	0001 0110
1001	0011 0111



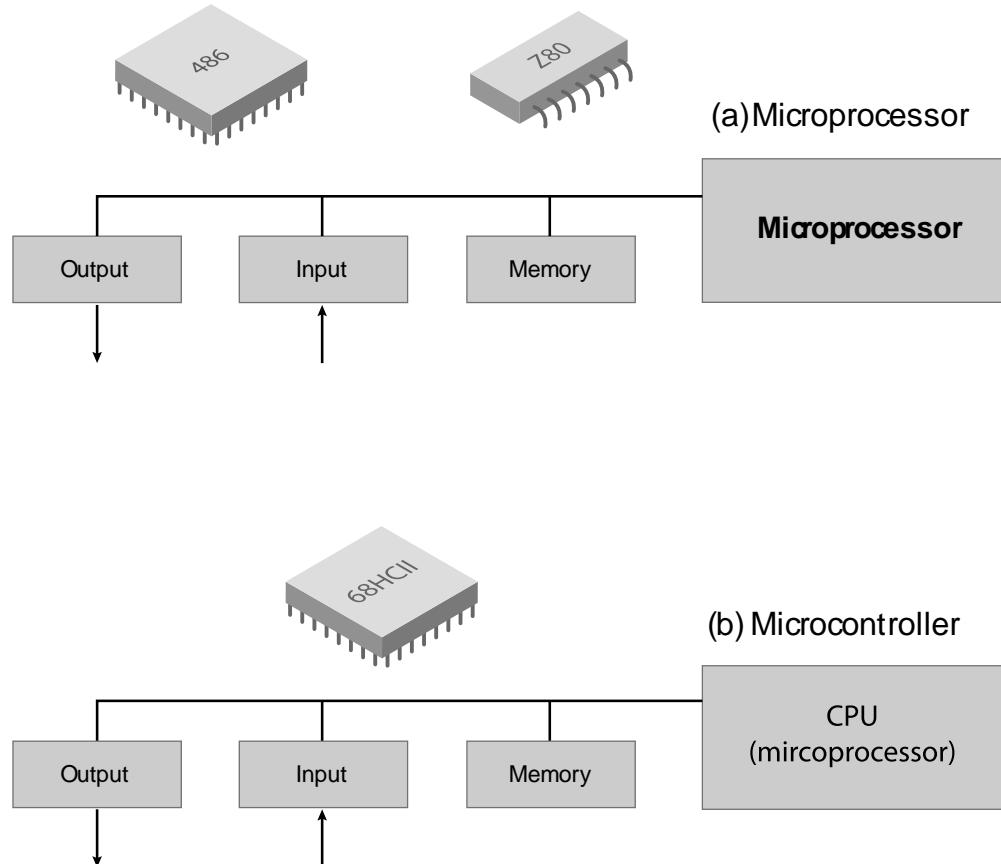
# Benefits of CPUs

- Programmable circuits allow highly customizable logic in hardware
  - Enables explosion of sophistication
- Focused on digital applications
  - Easier to consolidate into a compact unit
- CPUs are great for many applications
  - Extremely flexible, high-speed
  - Can run operating systems, machine vision, NLP, etc.

# Downsides of CPUs

- However, hard to make them very cheaply
  - Hard to make general, powerful platforms also cheap
  - What do we do for ultra-cheap applications?
- Idea: make special-purpose CPUs
  - Architectures targeted toward specific uses
  - Only the capabilities you need → reduced costs

# Microcontrollers



A microcontroller is like a CPU, but:

1. Lower-end design
  - Lower-performance (lower power usage/clockspeed)
  - Simpler (limited instruction set)
2. Consolidated design
  - Memory, oscillator, I/O controllers, watchdog timers, are within chipset
3. Special on-chip functions (peripherals)
  - Onboard DAC/ADC, PWM, NCO, programmable logic

# How to Choose a Microcontroller

- **Core design:** vendor-proprietary vs. third-party
  - 32-bit ARM core becoming ubiquitous, Intel 8051 also used
- **High- vs. low-end:**
  - Microcontrollers come in “families”: some have hundreds of models to choose from, some have much fewer. Smaller: economy of scale; larger: select what you need
  - Sunken cost in choosing a family
- **Package availability**
  - Wafer-level CSP package that's 2x2mm? Larger DIP package?

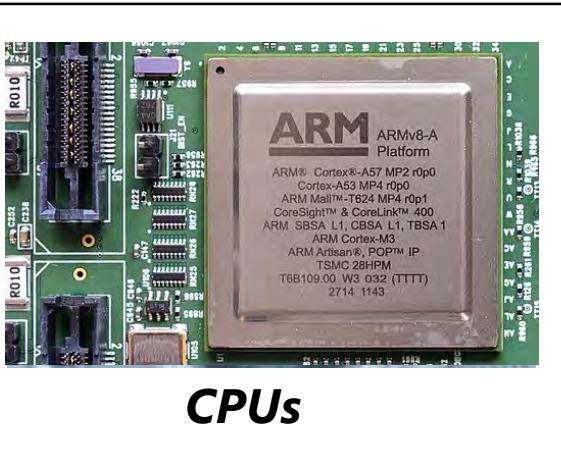
# How to Choose a Microcontroller

- **Peripherals:** multiple interrupt channels, DMA, internal clocks, power configuration control options, etc.
- **Development experience:**
  - What kinds of IDEs/SDKs/compilers/emulators/dev boards/debuggers are available? How much do they cost? Are they cross-platform?
- **Performance:** power efficiency, clock cycle efficiency, code size

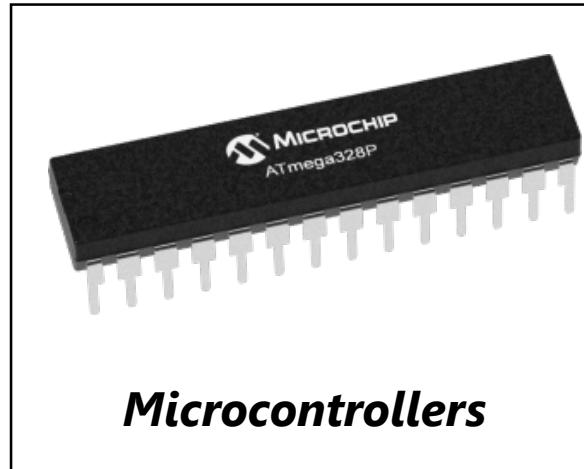
# Popular Microcontroller Cores

# Popular Microcontroller Cores

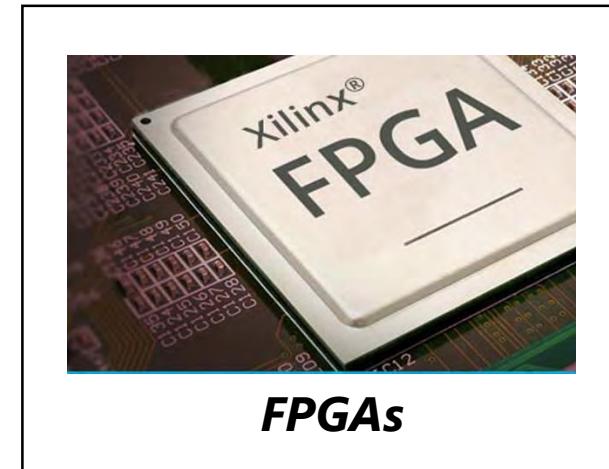
# Other Kinds of Programmable Circuits



# **CPUs**



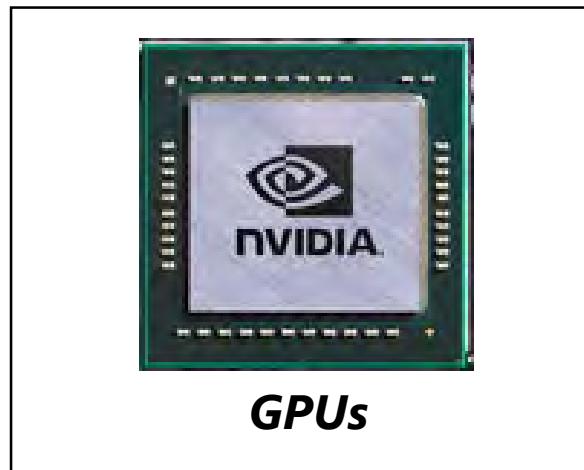
# **Microcontrollers**



## **FPGAs**



**ASICS**



# **GPUs**



# **System on Chip (SoC)**



# FPGAs (Field-Programmable Gate Arrays)

---

- FPGAs consist of a bunch of:
  - Programmable logic blocks – can be configured into logic gates (AND/XOR/NOT/etc.) or complex combinatorial functions
  - Interconnects to hook them together
- Many FPGAs can be reprogrammed after deployment (in the “field”)
  - Enables flexible “reconfigurable computing”
  - Can fix vulnerabilities/bugs post-deployment

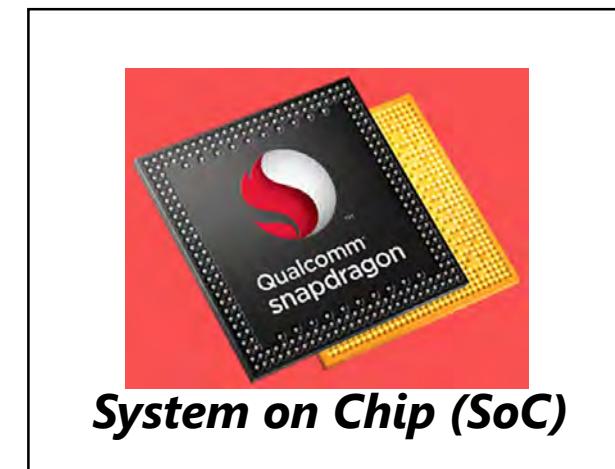
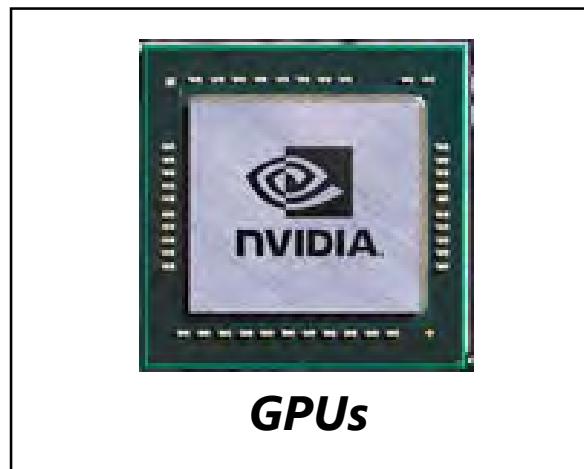
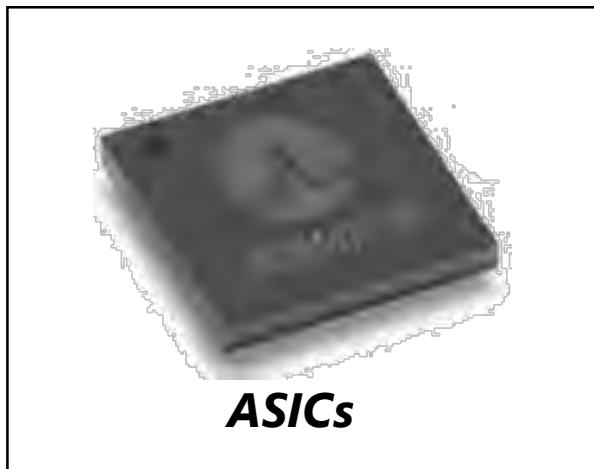
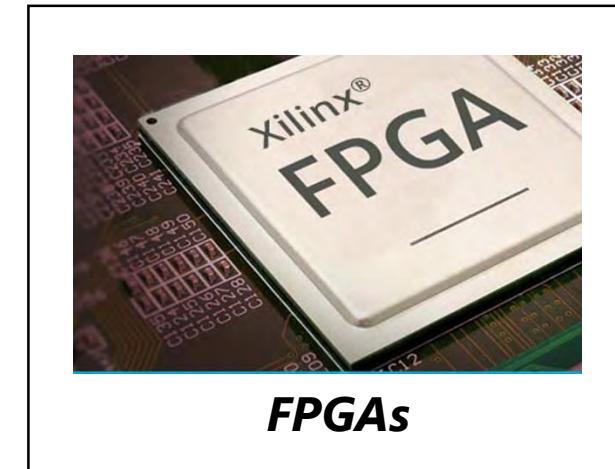
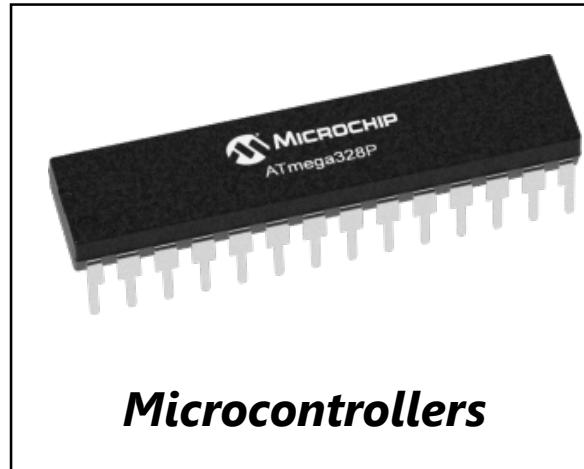
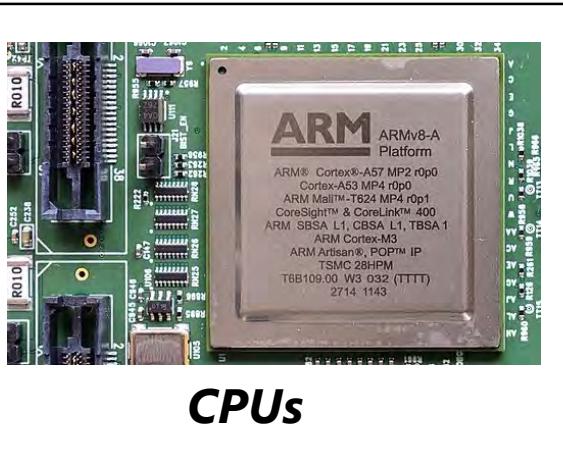


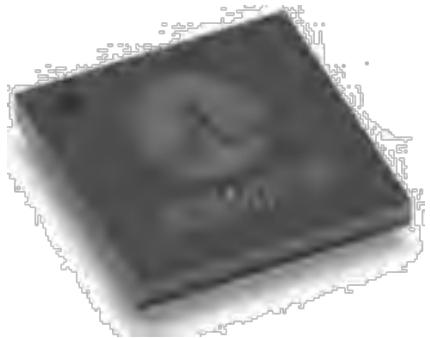
# FPGAs (Field-Programmable Gate Arrays)

---

- Can write whatever circuits you want → parallel execution
  - Can be more power efficient than single-threaded microcontrollers for some applications (ML/AI/data processing)
- However, current tech targets high end market → expensive (>\$500)
- Key players: Xilinx (49% of market), Altera (40% of market)
  - Intel (see Cyclone 10)

# Programmable Circuits





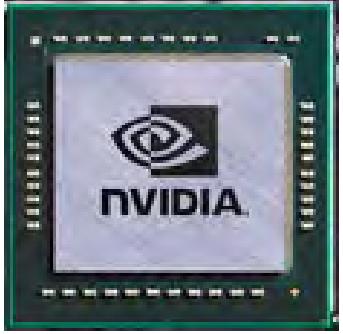
# ASICs (Application-Specific Integrated Circuit)

- Burn your design into silicon
  - 100s of millions of logic gates
  - Can include ROM, RAM, EEPROM, flash memory, and other large building blocks
- Core logic not field-programmable, hard to update designs
  - May contain some reconfigurable logic



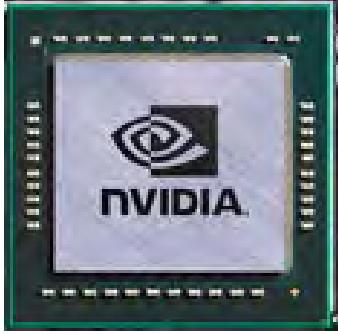
# ASICs (Application-Specific Integrated Circuit)

- ASICs (and FPGAs) used to implement CPUs/microcontrollers/etc.
  - Typical design: IP core CPU, digital signal processor, communication interfaces, memory
- Non-recurring engineering (NRE) of millions of dollars
  - Best for large production volumes, shared functionalities
- Key players: TSMC, GlobalFoundries, Samsung Semiconductor, Texas Instruments



# GPUs (Graphics Processing Unit)

- Designed to rapidly manipulate and process images and video
  - Many threads, like a highly parallel CPU (e.g., 480-1536 stream processors)
- Commonly used in mobile phones, personal computers, game consoles, self-driving cars, drones, automated surveillance, etc.



# GPUs (Graphics Processing Unit)

- Contains hardware implementations of important graphics functions
  - Texture mapping, rendering polygons, rotation/translation of vertices, shading, special effects (chroma keying, bokeh, volumetric lighting)
- Non-graphics applications
  - Applications that can be parallelized work well: processing hashes (bitcoin mining), matrix operations, audio processing, machine learning
- Key players: NVIDIA, Intel, AMD



## SoCs (System on Chip)

- SoC=system on chip
- Basically pairs a CPU/microcontroller with other functions, on one chip
  - Tighter integration → better reliability, power savings, lower cost
- Paired functions: AI acceleration, machine vision, wireless communications, digital camera hardware/firmware, etc.

# Emerging Programmable Circuits

- **Vision Processing Unit (VPU)** – accelerates **computer vision** – Microsoft HoloLens, Movidius Myriad X, NVIDIA VLIW Vision Processor
- **Neural Network Processor (NNP)** – accelerates **neural network** machine learning – Google (Tensor Processing Unit), Intel (Nervana), AWS (Inferentia), Apple (Neural Engine)
- **Machine Learning Processor (MLP)** – general programmable platform for **machine learning** – ARM (Trillium), IBM (Power9)

# Open-Source Hardware Logic

- Don't create complex hardware designs from scratch!
- Highly advanced complexity in hardware → takes a lot of time and investment to create
  - Re-use cuts product development times and creates better products
  - Open-source organizations provide free IP cores
    - E.g., [www.opencores.org](http://www.opencores.org)

# Open-Source Hardware Logic

- **Cell libraries:** libraries of logical sub-components that can be plugged together
- **IP cores:** primitives purchased from a third-party
  - May be provided as hardware-description language ("soft macro")
  - May be provided as a fully routed design ("hard macro")
  - ARM *only* sells IP cores ("fabless manufacturer")

# How to decide which to use?

Technology	When to use	Example applications
------------	-------------	----------------------

# Example Products

# Memory/Storage

- No “universal memory” (yet)
  - So best way to build something is use a mix of technologies
- Usage characteristics of memories:
  - May be easily writeable (RAM) or hard to write more than once (ROM)
  - May lose contents when not powered (volatile), or may retain contents when powered off (non-volatile)
- Performance characteristics of memories:
  - Max erase cycles, cost per byte, speed, density, heat, energy efficiency

# Memory Technologies

Type	Volatile?	Writeable?	Speed	Cost per byte	Notes
------	-----------	------------	-------	---------------	-------

# IoT Platform Design

# How can we design things out of these components?

- You now know many components you can use
- You now know how to connect them together with wires
- But how do you build actual physical things?
- Next: let's study some real platforms to get a sense of that
  - Let's start with the Arduino platform

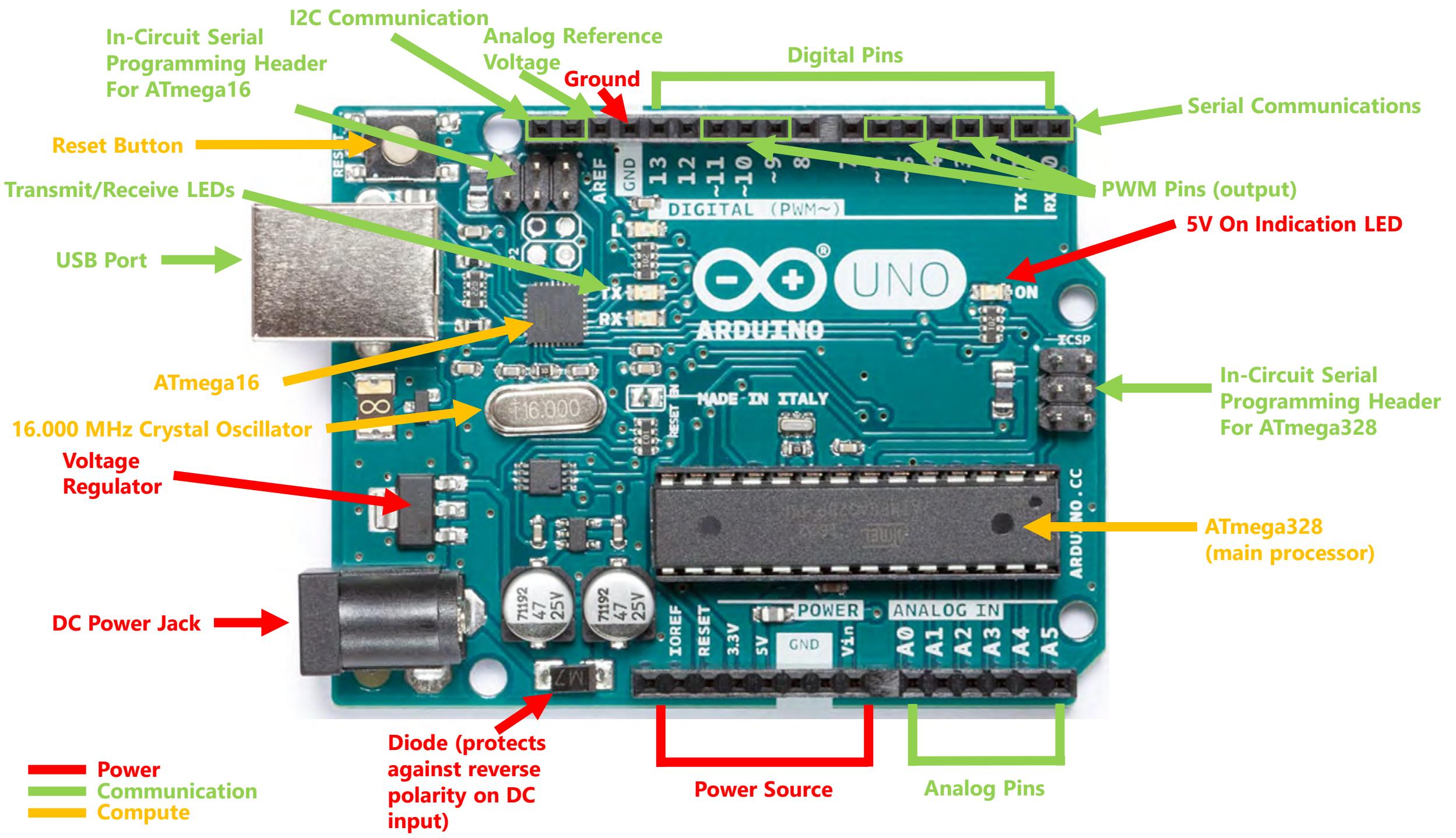


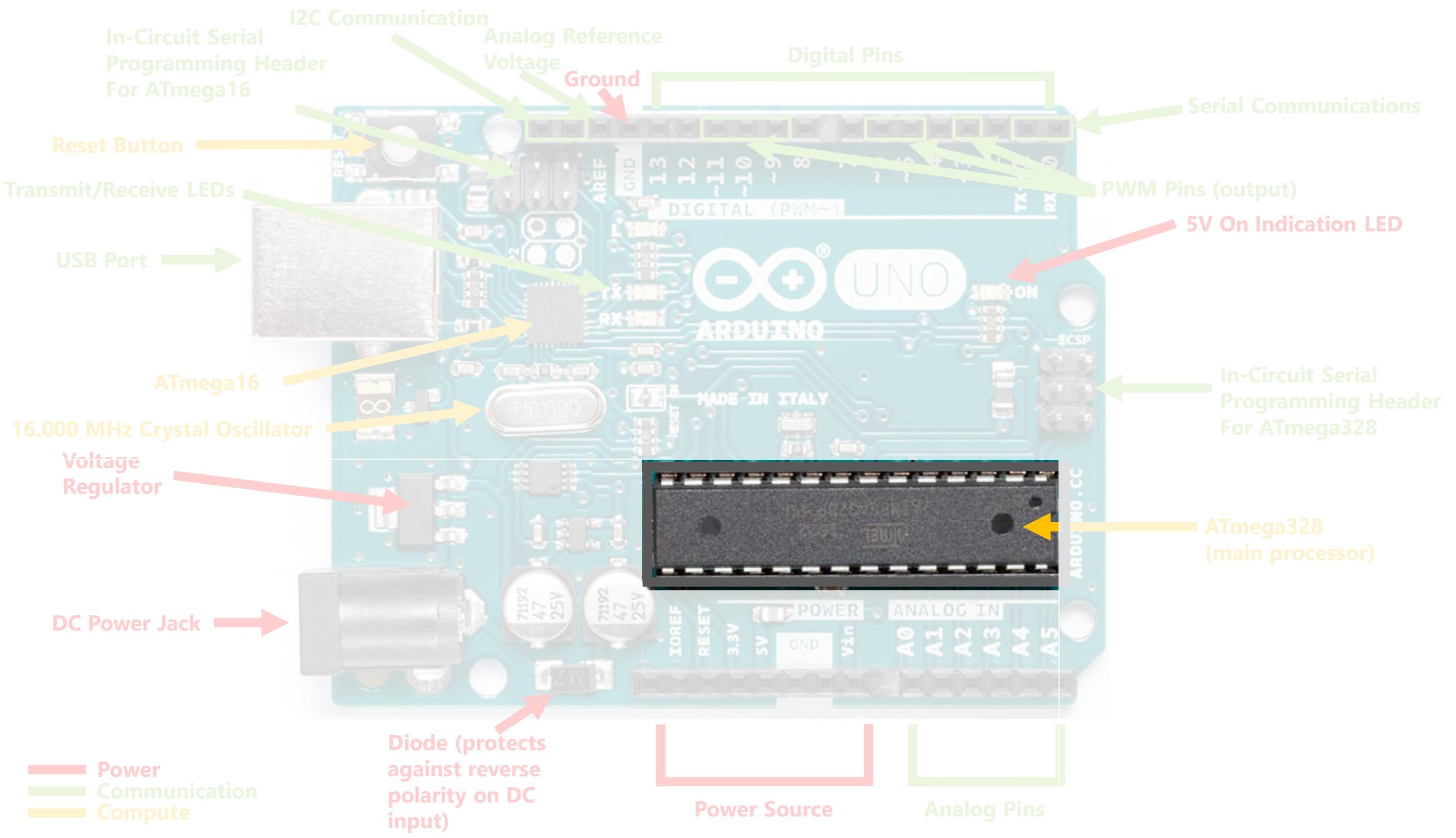
# Arduino Platform

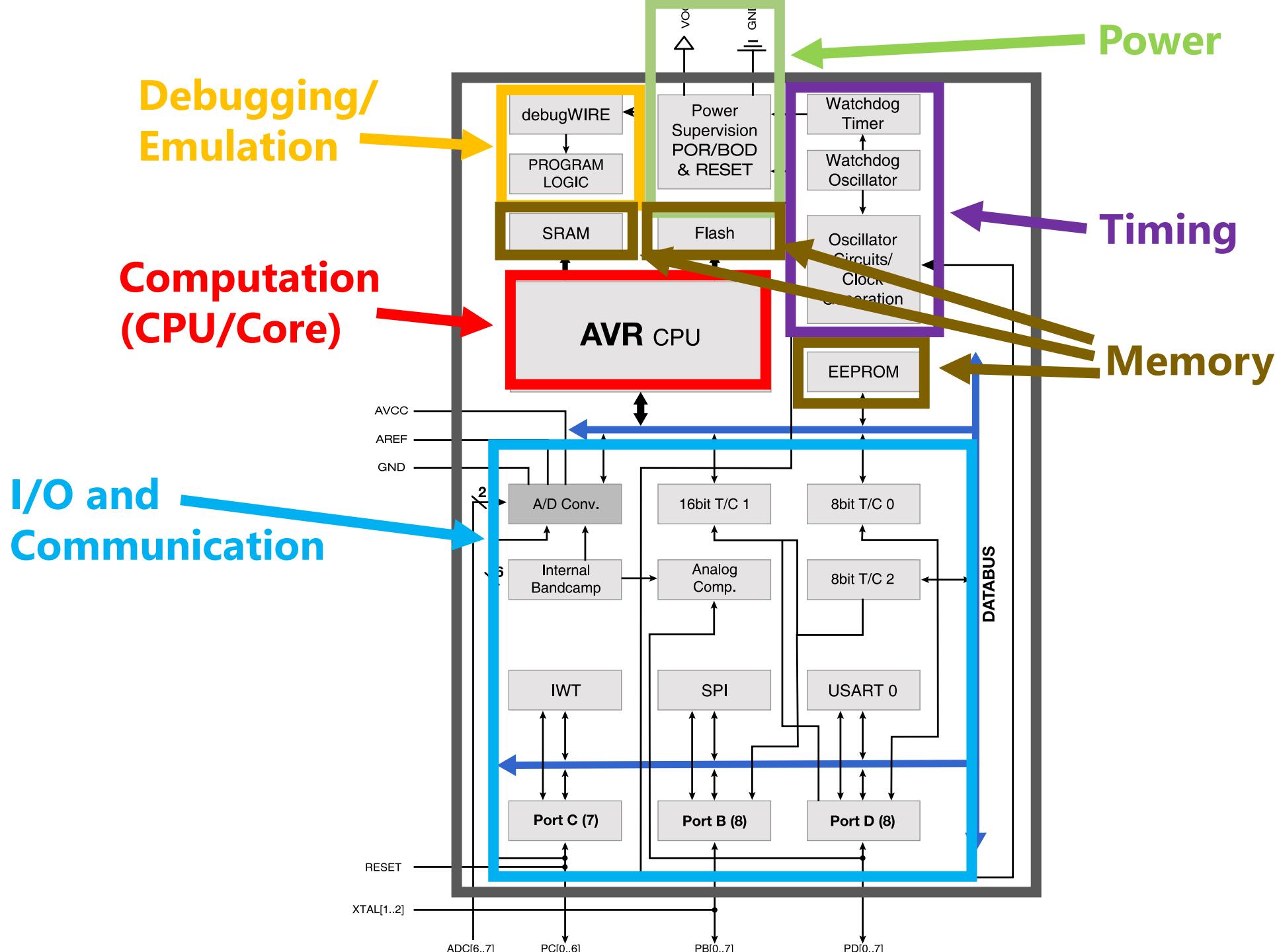
- Hobbyist prototyping platform for IoT devices
  - Simplifies some aspects of programming and circuit development
- You can do so many things with it
  - Can plug in virtually unlimited sensors/indicators/displays/motors/etc.
- Started as a research project at Interaction Design Institute in Ivrea, Italy
  - Name comes from a bar in Italy where founders used to meet
  - Project goal: develop a cheap and easy way for novices to create devices that interact with their environment using sensors/actuators

# Arduino Technical Components

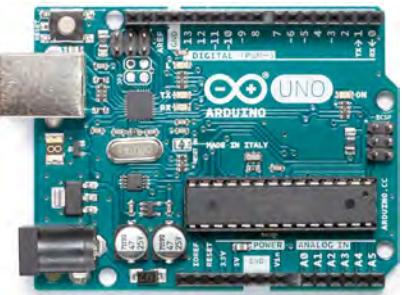
- Arduino provides software to simplify development
  - Interactive development environment
  - Collection of libraries to simplify code
- Code written in C++
  - Program is called a “sketch,” saved with extension “.ino”
- Multiple board designs, using various microprocessors/controllers
  - Both hardware (Creative Commons) and software (GPL) is open-source







# Official Arduino Boards



**Arduino Uno Rev3**  
*(most popular Arduino board)*

ATmega328P@16MHz

14 digital I/O pins (6 PWM), 6 analog inputs

USB, power jack, ICSP header

32KB Flash, 2KB SRAM, 1KB EEPROM

2.7"x2.1", 25 grams



**Arduino Uno Wi-Fi Rev2**  
*(Similar to Uno, but has Wi-Fi)*

ATmega4809@16MHz  
ECC608 crypto co-processor

14 digital I/O pins (5 PWM), 6 analog inputs

USB, power jack, ICSP header

48KB Flash, 6KB SRAM, 0.25KB EEPROM

2.7"x2.1", 25 grams



**Arduino Nano**  
*(smaller Arduino board)*

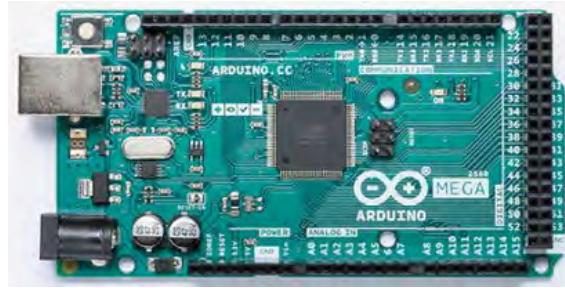
ATmega328

22 digital I/O pins (6 PWM), 8 analog inputs

USB, ICSP header

32KB Flash, 2KB SRAM, 1KB EEPROM

0.7"x1.7", 7 grams



**Arduino Mega**  
*(larger Arduino board)*

ATmega2560

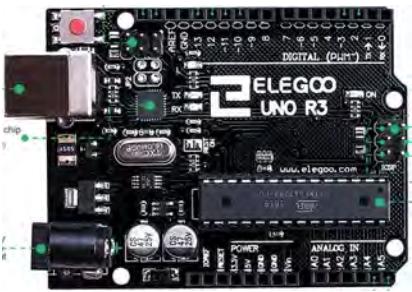
54 digital I/O pins (15 PWM), 16 analog inputs

256KB Flash, 8KB SRAM, 4KB EEPROM

USB, power jack, ICSP header

4"x2.1", 37 grams

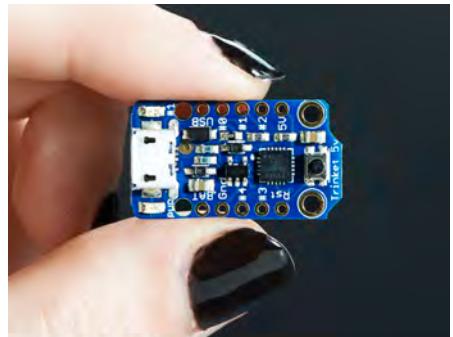
# Third-Party Arduino-Compatible Boards



**Elegoo Uno  
(lower-cost Arduino)**

Fully compatible; uses faster control processor (ATMega16U2)

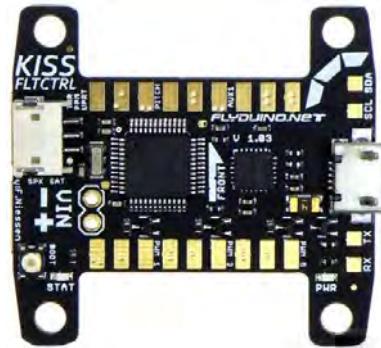
Other boards exist at <\$3



**Adafruit Trinket  
(small Arduino)**

Small (1.2"x0.6"), light (1.85 grams). Good for wearables (e.g., necklaces), dog tracking, covert computing, etc.

Attiny85@16MHz, 8KB Flash, 512bytes SRAM, 512bytes EEPROM



**FlyDuino KISS FC  
(domain-specific Arduino)**

Arduino-compatible board designed for autonomous navigation and auto-piloting of aircraft.

Flight control firmware monitors and performs telemetry of motor speed (ESCs). Stabilizes flight and automates movement and tracking.



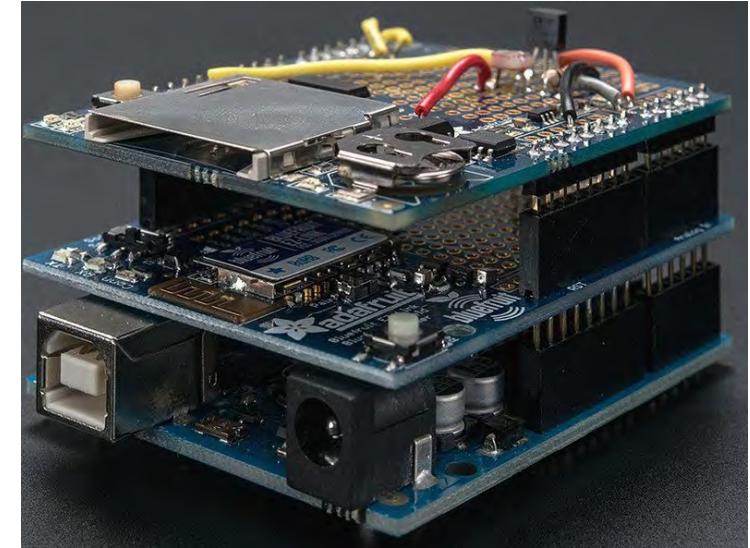
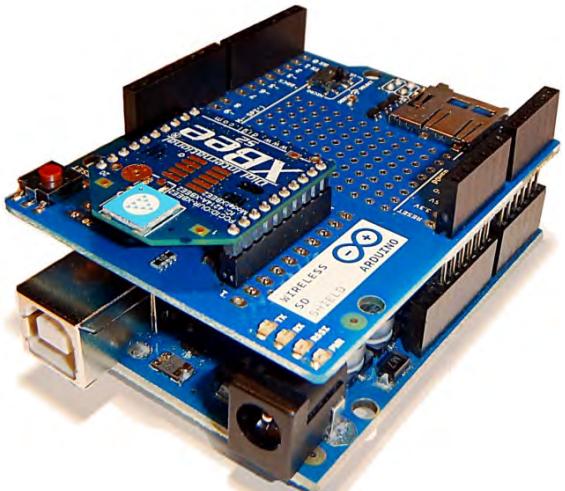
**Ardusat  
(domain-specific Arduino)**

Nanosatellite based on CubeSat standard

Public allowed to use platform via ArduSatSDK while in space

Includes a 3-axis magnetometer, a 3-axis gyroscope, a 3-axis accelerometer, an infrared temperature sensor, four temperature sensors, two luminosity sensors (infrared and vis light), two Geiger counters, one optical spectrometer, one 1.3MP camera

# Arduino Shields



**Arduino Uno + Xbee Module + SD Card Storage**

Many shields are out there:

Motor drivers, cellular (e.g., GSM GPRS), data logging, GPS, blank (for prototyping), LCD/Displays, Cameras, Speakers/buzzers, sensors (e.g., temperature/humidity), pushbuttons, capacitive touchpads, IR receivers, medical sensors, biometrics, wired ethernet, relays (high voltage), lasers, smoke detectors, joystick/game controller, MP3 players, etc.

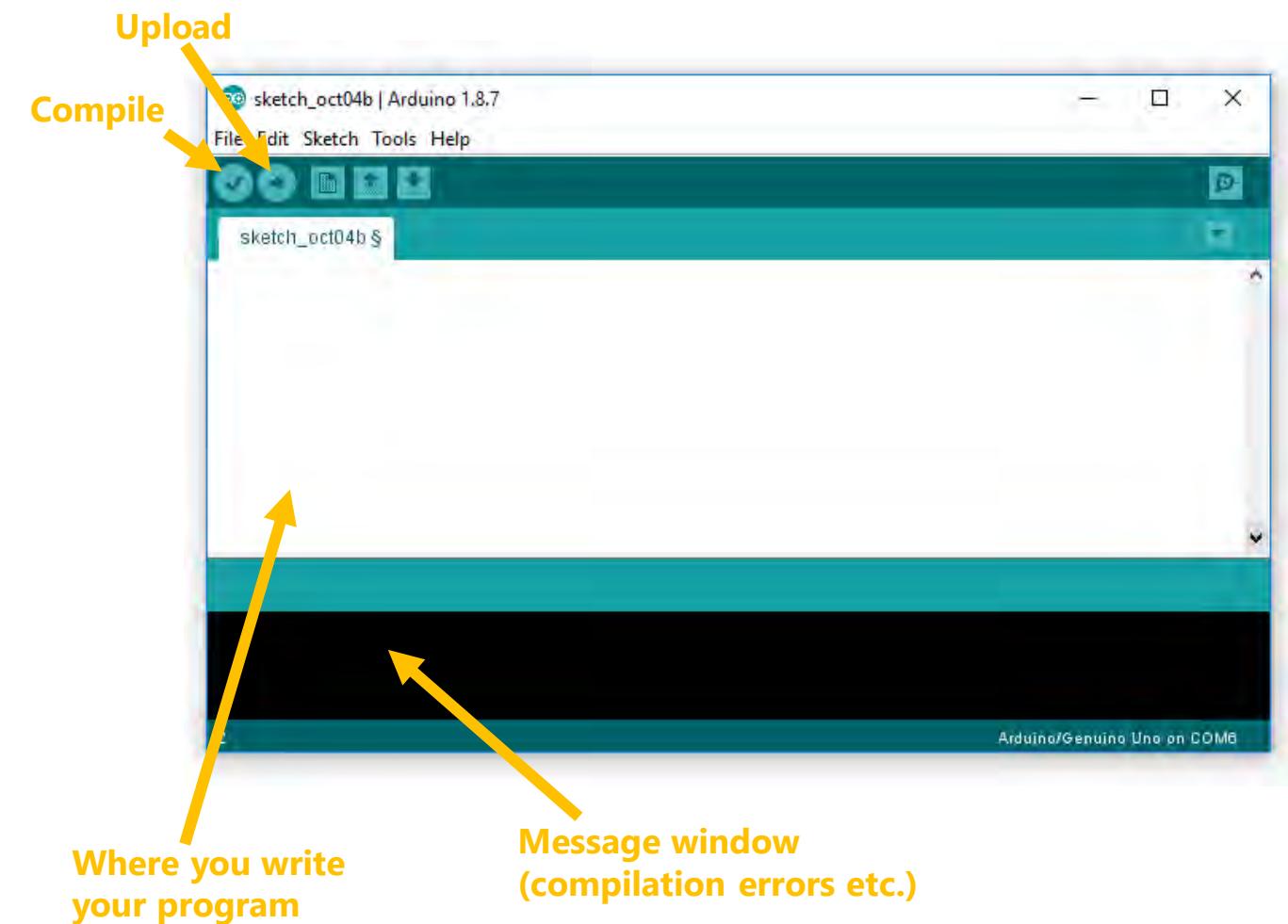
# How do we program an Arduino?

- Usually people program microcontrollers in high-level languages
- Arduino platform makes this process even easier
  - Simple IDE
  - Libraries, objects (some from Arduino, some from Atmel/AVR)

# How do we program an Arduino?

- Arduino sketches are C++ programs calling some extra supporting libraries
- Arduino programming is similar to other kinds of embedded programming
  - Approaches very generalizable to other microcontrollers, platforms, etc.

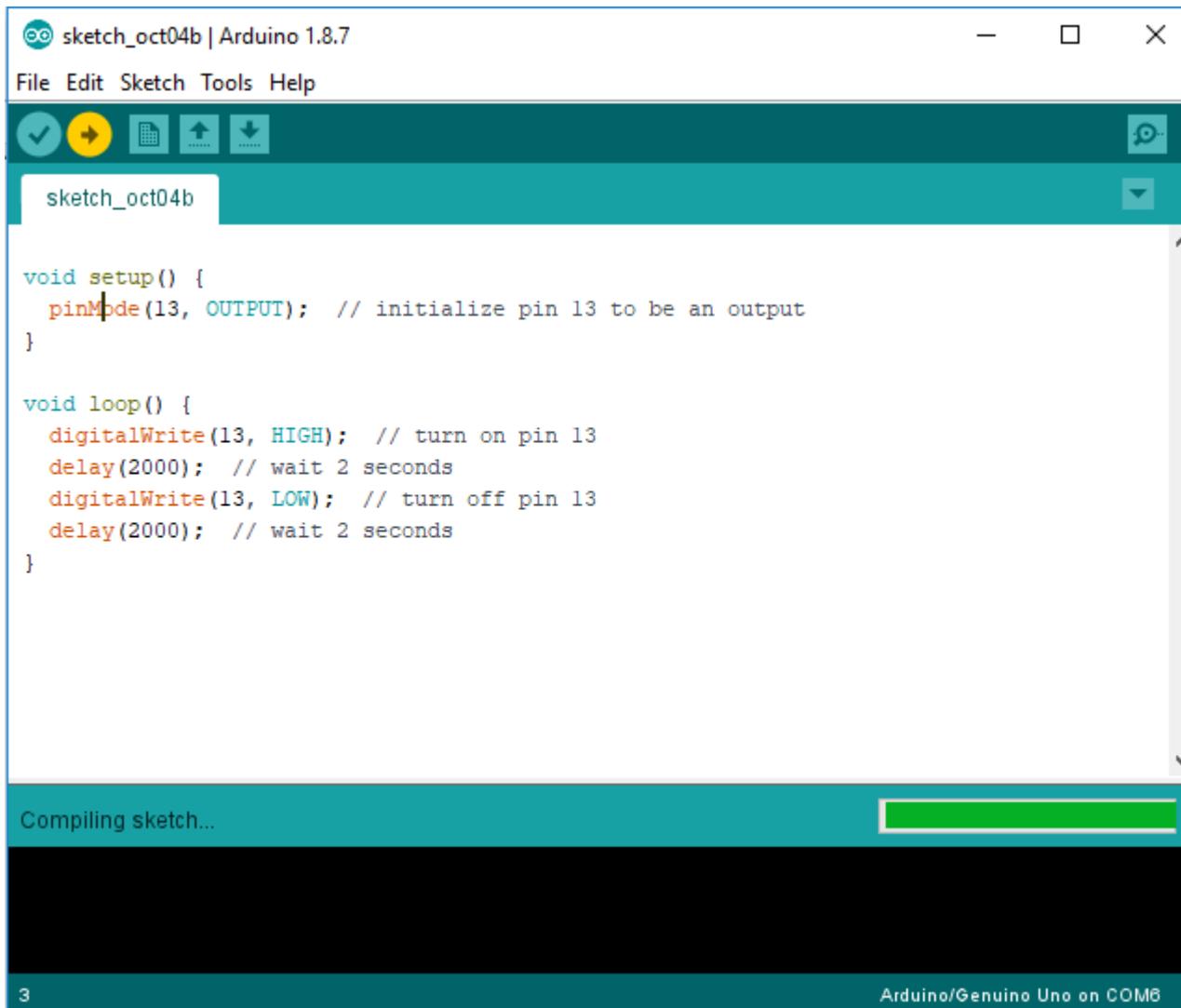
# What the IDE Looks Like



## Setting up Arduino:

1. To tell IDE how to reach the Arduino, choose Tools::Port: Select COM1 or whatever port your Arduino is plugged into
  - Windows: Device manager, expand Ports, see where Arduino is
  - Mac: type "ls /dev/\*" in terminal, use port number listed for /dev/tty.usbmodem\* or /dev/tty.usbserial\*
  - Linux: type "ls /dev/ttUSB\*" or "/dev/ttACM\*"
2. To tell IDE how to talk to the Arduino, tell it the type of board under Tools:Board
  - For Uno or Uno-compatible board select Arduino/Genuino Uno

# Writing an Arduino Program



```
sketch_oct04b | Arduino 1.8.7
File Edit Sketch Tools Help
sketch_oct04b

void setup() {
  pinMode(13, OUTPUT); // initialize pin 13 to be an output
}

void loop() {
  digitalWrite(13, HIGH); // turn on pin 13
  delay(2000); // wait 2 seconds
  digitalWrite(13, LOW); // turn off pin 13
  delay(2000); // wait 2 seconds
}

Compiling sketch...
Arduino/Genuino Uno on COM6
```

- Two main functions, executed by bootloader
  - **setup()** – executed once on startup
  - **loop()** – run repeatedly, forever
- Functions you can call (digital I/O):
  - **pinMode(<pin>, <type>)** – configures pin as detecting an input signal (INPUT) or outputting a signal (OUTPUT). Can also enable an internal pullup resistor (INPUT\_PULLUP)
  - **digitalWrite(<pin>, <value>)** – sets pin state (HIGH or LOW)
  - **<int> = digitalRead()** – reads pin state (HIGH or LOW)

# Uploading/Running an Arduino Program

Three key steps:

## 1. Verify/Compile your code

- Sketch::Verify/Compile, or hit the check mark button

## 2. Upload bootloader to board

- Code that runs before main program
- Arduino bootloader enables re-programming flash memory over serial port
- Previous step takes care of this for you

## 3. Upload your code to board

- Sketch::Upload, or hit right-arrow button

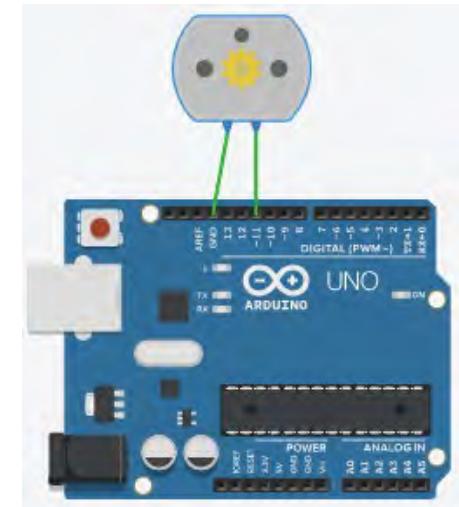
# Uploading/Running an Arduino Program

- Program will just start running automatically
- Troubleshooting ideas:
  - Make sure you have right board type and port selected
  - Check for short circuits, hot components, smoky smells
  - Use a multimeter to check output values and current loads
  - Use serial monitor (`Serial.println()`) to log progress
  - Use a simulator/emulator

# A Few More Example Sketches

# Suppose you...

- Need to debug an Arduino
  - How do you know what's going on inside?
  - Can use LEDs but that's kind of hard
  - May need to log and collect data
- Want to measure/send non-digital (analog) values?
  - E.g., read output from a sensor, control speed of motor



# Serial Programming, Analog I/O



The screenshot shows the Arduino IDE interface. The title bar reads "asketch | Arduino 1.8.7". The menu bar includes File, Edit, Sketch, Tools, and Help. Below the menu is a toolbar with icons for save, upload, and other functions. The main area displays the following Arduino sketch:

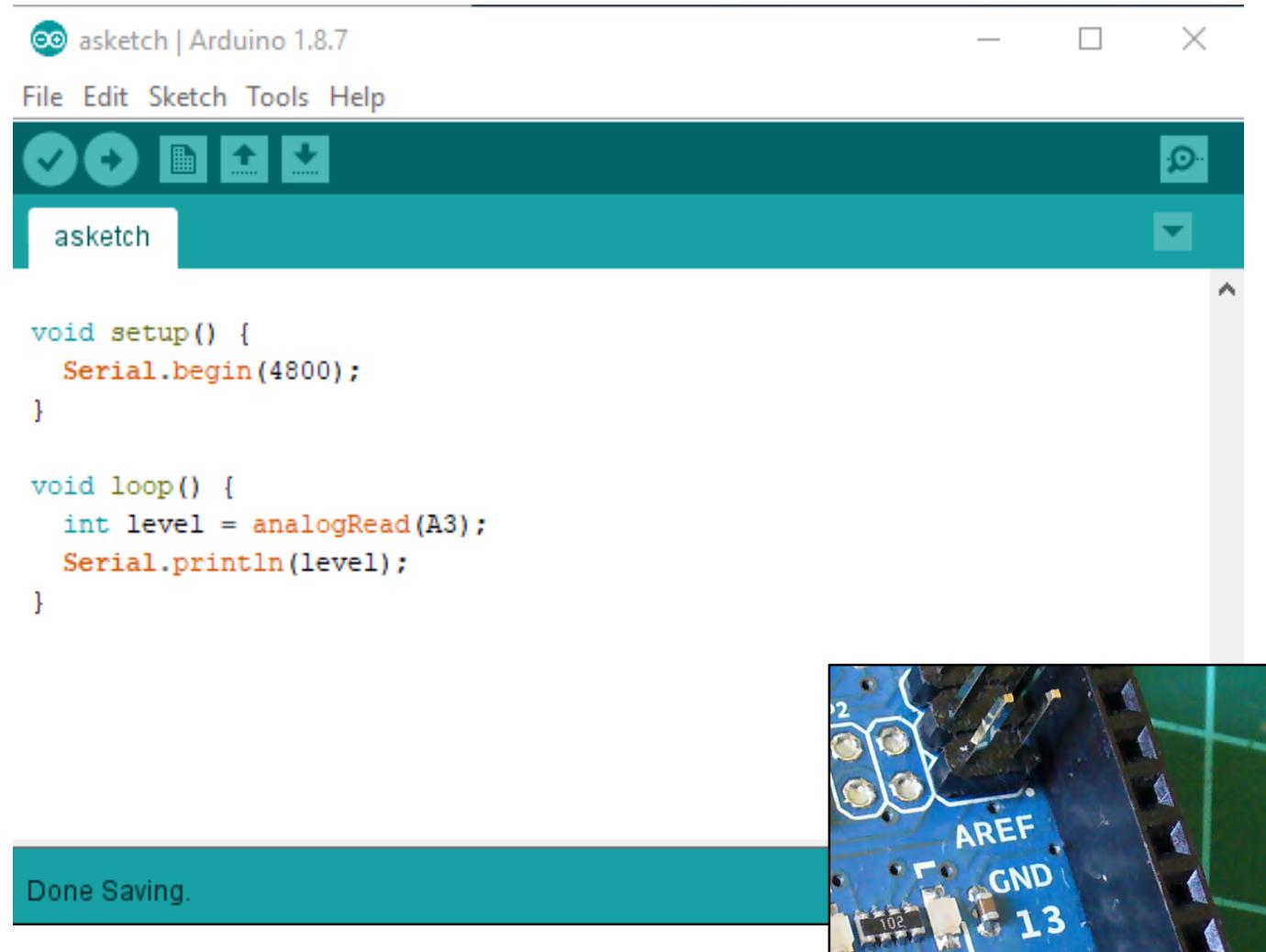
```
void setup() {
  Serial.begin(4800);
}

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

A status bar at the bottom says "Done Saving."

- Serial functions:
  - **begin(<speed>)** – sets data rate (baud)
  - **println(<value>)** – outputs data as ASCII text to serial port
  - **<int>=read()** – reads one byte from serial port, returns -1 if no data to read
  - Warning: data is written to USB \*and\* pins 0/1 – will clash with other uses of pins 0/1
  - Also see: hardware debugging (debugWire)

# Serial Programming, Analog I/O

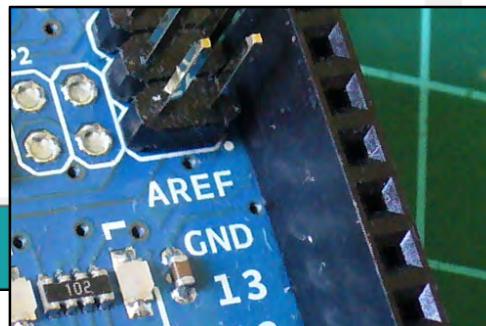


The image shows the Arduino IDE interface. The title bar says "asketch | Arduino 1.8.7". The menu bar includes File, Edit, Sketch, Tools, and Help. Below the menu is a toolbar with icons for save, upload, and refresh. The main area contains the following code:

```
void setup() {
  Serial.begin(4800);
}

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

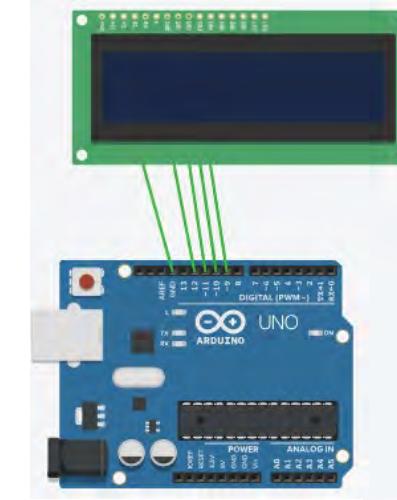
In the status bar at the bottom, it says "Done Saving."



- Analog I/O functions:
  - **<int>=analogRead(<pin>)** reads voltage from specified input pin as value between [0,1023]
  - **analogWrite(<level>)** – sends voltage level (PWM) to pin, specified as range from 0 (always off) to 255 (always on)
  - **analogReference()** – sets what 1023 corresponds to for analog input – DEFAULT (5V), INTERNAL (1.1V), EXTERNAL (voltage applied to AREF pin), etc.

# Suppose You...

- Want to send or read digital data
  - E.g., you have an external component with a microcontroller
- Measure pulses
  - Measure how long user holds down a button, how long car was blocking highway laser sensor
- Generate tones
  - Alarm sounds when burglar steps on smart floor, make music play when you walk into a room



# Digital Streaming I/O Functions

- Functions for shifting data in/out:
-

# Suppose You...



- Want to do multiple things at once
  - Run autopilot for drone, but user gives you a new command
  - Display camera feed, but pop up character pad when user touches screen



# Interrupts

```
sketch_jan26a | Arduino 1.8.7
File Edit Sketch Tools Help
sketch_jan26a §
int ledPin = 12;
int alarmPin = 13;
volatile int numButtonPresses = 0;

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(alarmPin, OUTPUT);
  attachInterrupt(digitalPinToInterrupt(5), alarm, RISING);
}

void loop() {
  digitalWrite(ledPin, HIGH);
  sleep(1000);
  digitalWrite(ledPin, LOW);
  sleep(1000);
  if (numButtonPresses>=5) {
    Serial.println("Button pressed 5 times!");
  }
}

void alarm() {
  digitalWrite(alarmPin, HIGH);
  sleep(2000);
  digitalWrite(alarmPin, LOW);
  numButtonPresses++;
}

Save Canceled.
```

- **attachInterrupt(pin, function, mode)** – associates function pointer to be called on pin level change
  - Modes: **LOW, CHANGE, RISING, FALLING**
- **digitalPinToInterrupt(pin)** – converts pin # to interrupt #
- **noInterrupts()/interrupts()** – enables/disables interrupts
- Notes:
  - Only one ISR can run at a time
    - Next ISR to run selected according to priority
  - Shared variables must be declared volatile

# What if something goes wrong?

- Program crashes or hangs in the field
  - It is 10,000 feet in the air or floating in the arctic
  - Your plant watering system locks up when you're on vacation
  - You can't get out there to help it



# Watchdog Timer

- “Deadman’s switch” hardware-supported timer
  - Counter counts up continuously in hardware
  - If not reset within timeout, board is rebooted
- **wdt\_enable(time)** – sets timeout
- **wdt\_reset()** – resets timer
- Levels supported by ATmega328:  
15ms, 30ms, 60ms, 120ms,  
250ms, 500ms, 1s, 2s, 4s, 8s



The screenshot shows the Arduino IDE interface with a sketch named "asketch". The code uses the `<avr/wdt.h>` library to manage a watchdog timer. In the setup() function, it enables the timer with a 25-second timeout, commenting that it will "wake up the dog". In the loop() function, it processes data and then calls `wdt_reset()` to "pat the dog", which is a common idiom for resetting the timer. A message at the bottom indicates "Done Saving."

```
#include <avr/wdt.h>

void setup() {
    wdt_enable(WDTO_25); // wake up the dog
}

void loop() {
    process_data();
    wdt_reset(); // pat the dog
}
```

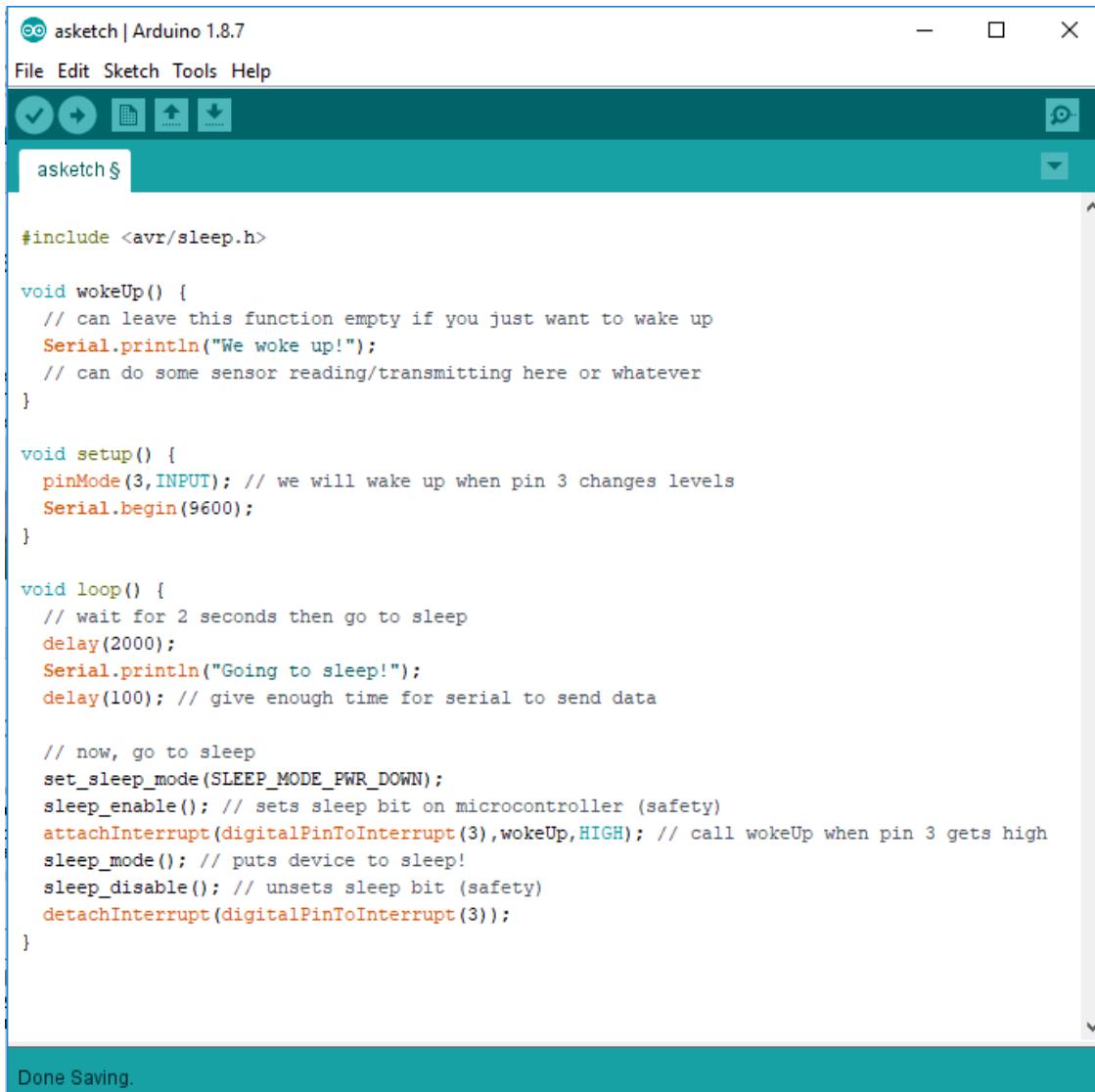
Done Saving.

# What if you are running out of power?

- May need to operate for long times on low battery
- May be reaching end of battery life
- May not be able to replace battery
- May not even have a battery – energy harvesting and sources drying up



# Sleep Mode



The screenshot shows the Arduino IDE interface with a sketch titled "asketch". The code demonstrates how to enter sleep mode using the `<avr/sleep.h>` library. It includes a `wakeUp()` function for pin change detection, a `setup()` function to set up pin 3 as an input and initialize serial communication, and a `loop()` function that waits for 2 seconds, prints a message, and then enters sleep mode via `set_sleep_mode(SLEEP_MODE_PWR_DOWN)`. It also attaches an interrupt on pin 3 to call `wakeUp` when it goes high, and uses `sleep_enable()` and `sleep_disable()` for safety.

```
#include <avr/sleep.h>

void wakeUp() {
    // can leave this function empty if you just want to wake up
    Serial.println("We woke up!");
    // can do some sensor reading/transmitting here or whatever
}

void setup() {
    pinMode(3, INPUT); // we will wake up when pin 3 changes levels
    Serial.begin(9600);
}

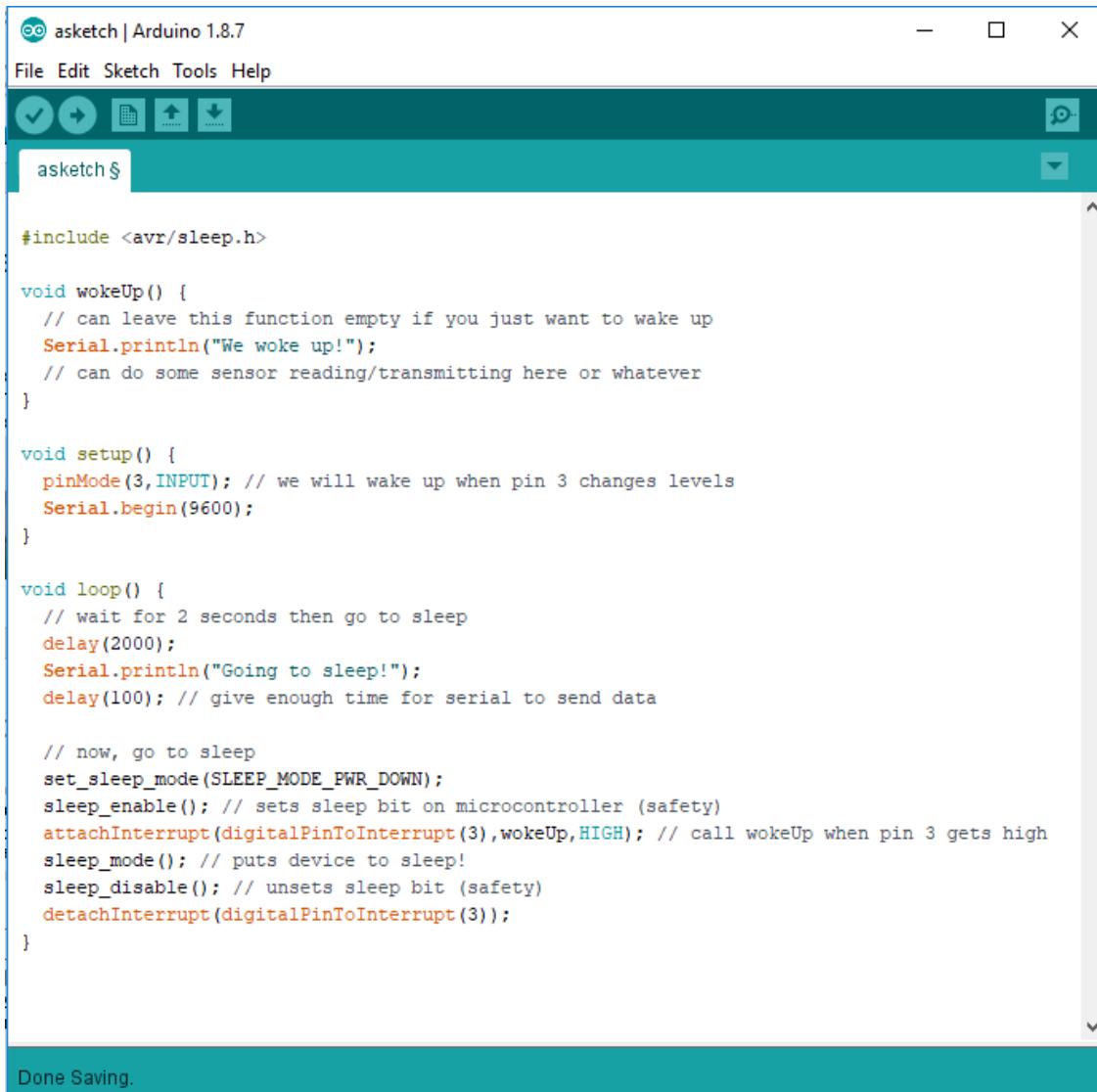
void loop() {
    // wait for 2 seconds then go to sleep
    delay(2000);
    Serial.println("Going to sleep!");
    delay(100); // give enough time for serial to send data

    // now, go to sleep
    set_sleep_mode(SLEEP_MODE_PWR_DOWN);
    sleep_enable(); // sets sleep bit on microcontroller (safety)
    attachInterrupt(digitalPinToInterrupt(3), wakeUp, HIGH); // call wakeUp when pin 3 gets high
    sleep_mode(); // puts device to sleep!
    sleep_disable(); // unsets sleep bit (safety)
    detachInterrupt(digitalPinToInterrupt(3));
}
```

Done Saving.

- Can be woken up by timer, pin level change
- **set\_sleep\_mode(mode)** – sets how deeply to sleep
  - Sleep modes:  
`SLEEP_MODE_PWR_DOWN` (most savings), `SLEEP_MODE_STANDBY`, `SLEEP_MODE_PWR_SAVE`, `SLEEP_MODE_ADC`, `SLEEP_MODE_IDLE` (least savings)
  - Disables brown-out detection, timers, analog-digital conversion
- **sleep\_enable()/sleep\_disable()** – enables/disables safety “sleep” bit on microcontroller
- **sleep\_mode()** – triggers sleep

# Sleep Mode



The screenshot shows the Arduino IDE interface with a sketch titled "asketch". The code implements a sleep mode using the `<avr/sleep.h>` library. It includes a wake-up function, a setup function that configures pin 3 as an input and initializes serial communication, and a loop function that waits for 2 seconds before entering sleep mode. The code uses `delay()` instead of `sleep()` for demonstration purposes.

```
#include <avr/sleep.h>

void wokeUp() {
    // can leave this function empty if you just want to wake up
    Serial.println("We woke up!");
    // can do some sensor reading/transmitting here or whatever
}

void setup() {
    pinMode(3, INPUT); // we will wake up when pin 3 changes levels
    Serial.begin(9600);
}

void loop() {
    // wait for 2 seconds then go to sleep
    delay(2000);
    Serial.println("Going to sleep!");
    delay(100); // give enough time for serial to send data

    // now, go to sleep
    set_sleep_mode(SLEEP_MODE_PWR_DOWN);
    sleep_enable(); // sets sleep bit on microcontroller (safety)
    attachInterrupt(digitalPinToInterrupt(3), wokeUp, HIGH); // call wokeUp when pin 3 gets high
    sleep_mode(); // puts device to sleep!
    sleep_disable(); // unsets sleep bit (safety)
    detachInterrupt(digitalPinToInterrupt(3));
}
```

Done Saving.

- Can replace `delay()` calls, use when not doing anything important
- Doesn't actually save much power on some Arduinos because of voltage regulator
  - However you can use this concept when programming other platforms
- Other ways to save power: avoid linear regulator, lower voltage (5V to 3.3V reduces ~4mA to ~1mA), reduce clock speed

# What if you need to store some information?

- May need to log data
  - Could send to Serial, but what if not attached to a computer?
  - Could send to Wi-Fi, but what if no clearance to send?
- Need to retain state across power outages/resets
  - Storing configurations and user preferences



# Arduino Storage

## EEPROM



The screenshot shows the Arduino IDE interface with a sketch titled "sketch\_jan27a". The code uses the EEPROM library to write analog values from an ADC to memory at address 0.

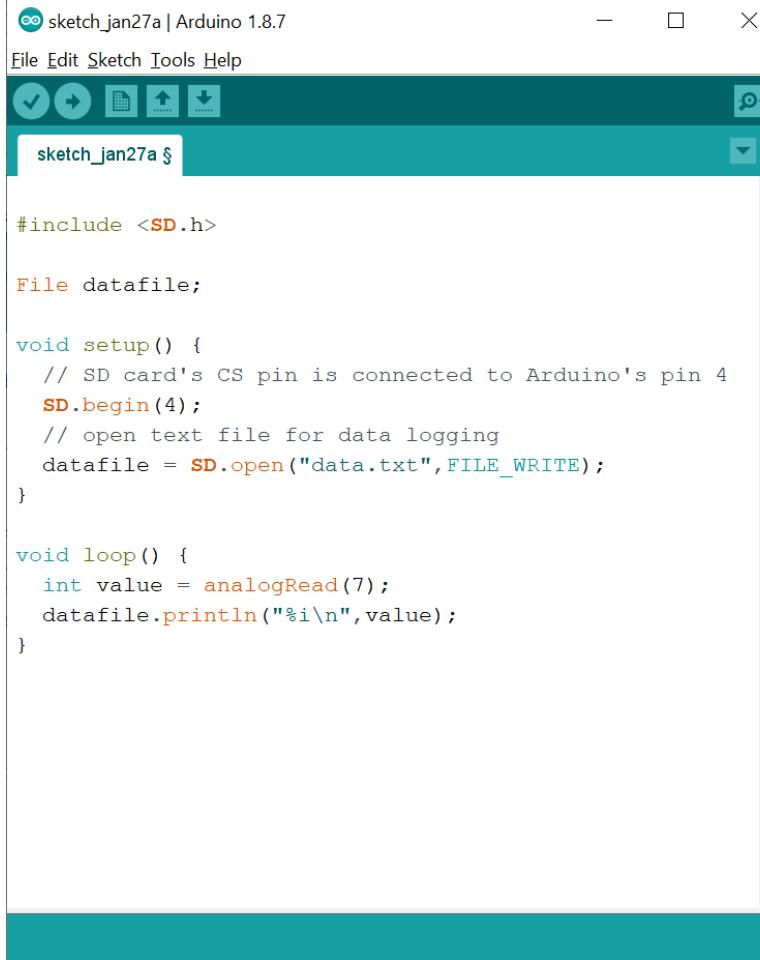
```
#include <EEPROM.h>

int address=0;

void setup() {
}

void loop() {
    int value = analogRead(7);
    EEPROM.write(address,value);
    address++;
}
```

## SD Card



The screenshot shows the Arduino IDE interface with a sketch titled "sketch\_jan27a". The code uses the SD library to write analog values to a file named "data.txt" on an SD card.

```
#include <SD.h>

File datafile;

void setup() {
    // SD card's CS pin is connected to Arduino's pin 4
    SD.begin(4);
    // open text file for data logging
    datafile = SD.open("data.txt",FILE_WRITE);
}

void loop() {
    int value = analogRead(7);
    datafile.println("%i\n",value);
}
```

- Can access memory and storage. Examples:
- Write to EEPROM (byte-addressable):
  - read()/write()
  - clear() – clear entire EEPROM
  - crc() – calculates CRC of entire EEPROM contents
- Write to SD card (file-based storage):
  - Similar to POSIX: open()/close()/println()/read()/write()/mkdir()/remove()/rmdir()

# References

- Mouser Electronic / EasyEDA
- BatteryJunction.com
- Uxcell
- E-Projects
- NTE Electronics / <https://www.ntepartsdirect.com/>
- Jameco Electronics / <https://www.jameco.com/>
- Digi-Key Electronics / <https://www.digikey.com/>
- Dilson Enterprises
- Regency Semiconductors

# References

- Nest
- TickPlant
- Amazon
- Adafruit
- FilipeFlop Componentes Eletrônicos
- Anker
- Transfer Multisort Elektronik
- SparkFun Electronics / <https://www.sparkfun.com/>
- DJI
- Yale Security
- Open Bionics

# References

- Banggood.com
- Active Robots / <https://www.active-robots.com/>
- Davis Instruments / <https://www.davisinstruments.com/>
- Honda
- Anki
- Parallax Inc. / <https://www.parallax.com/>
- HT Sensor / <http://htc-sensor.com/>
- AIRSENSE Analytics GmbH / <http://airsense.com/en>
- Integrated Device Technology, Inc. / <https://www.idt.com/>

# References

- Texas Instruments
- National Semiconductor
- SeekIC
- Microchip
- Espressif Systems
- Adafruit / <https://www.adafruit.com/>
- NikeLG Electronics
- Moverio
- Fitbit
- HTC Vive

# References

- Parrot <http://www.parrot.com>
- Honeywell
- Pleo
- ElectronicSurplus.ca
- Arduino.cc