# Binary Numbers

723

# Binary Numbers

$$723 = 7\text{x}100 + 2\text{x}10 + 3\text{x}1$$

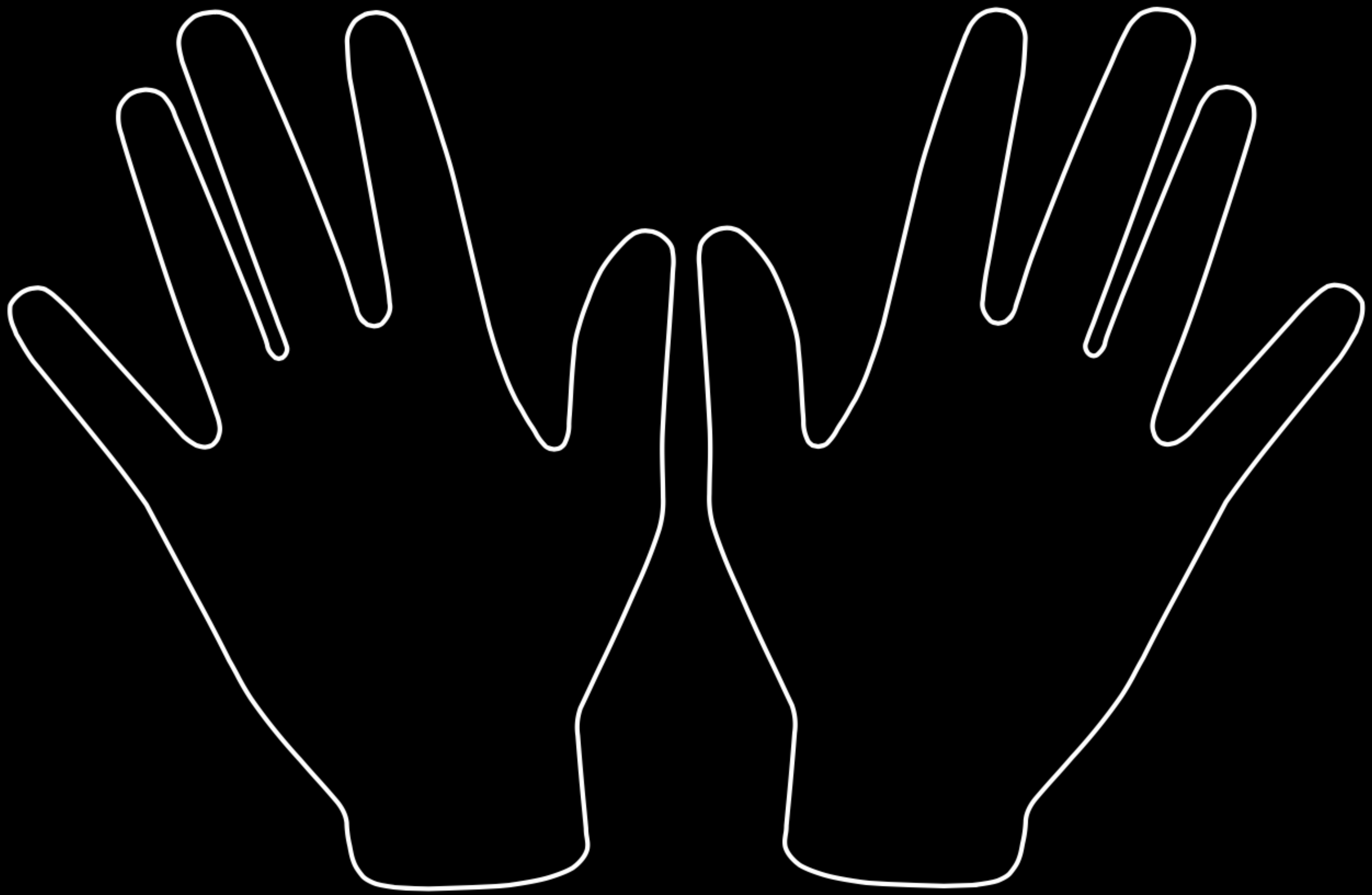
# Binary Numbers

$$723 = 7 \times 100 + 2 \times 10 + 3 \times 1$$
$$= 7 \times 10^2 + 2 \times 10^1 + 3 \times 10^0$$

# Binary Numbers

$$5349 = 5 \times 10^3 + 3 \times 10^2 + 4 \times 10^1 + 9 \times 10^0$$

# Binary Numbers



Why base 10?

# Binary Numbers

257 (base 8)

$2 \times 8^2 + 5 \times 8^1 + 7 \times 8^0$

$2 \times 64 + 5 \times 8 + 7 \times 1$

175 (base 10)

# Binary Numbers

0110 (base 2)

$0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$

$0 \times 8 + 1 \times 4 + 1 \times 2 + 0 \times 1$

6 (base 10)

# Binary Numbers

$2^1 2^0$

| | |
|---|---|
| 00 | 0 |
| 01 | 1 |
| 10 | 2 |
| 11 | 3 |

max value = $2^2 - 1$

2-bit binary number

# Binary Numbers

```
000   0
001   1
010   2
011   3
100   4
101   5
110   6
111   7
```

max value = $2^3$ -1

3-bit binary number

# Binary Numbers

| | |
|---|---|
| 0000 | 0 |
| 0001 | 1 |
| 0010 | 2 |
| 0011 | 3 |
| 0100 | 4 |
| 0101 | 5 |
| 0110 | 6 |
| 0111 | 7 |
| 1000 | 8 |
| 1001 | 9 |
| 1010 | 10 |
| 1011 | 11 |
| 1100 | 12 |
| 1101 | 13 |
| 1110 | 14 |
| 1111 | 15 |

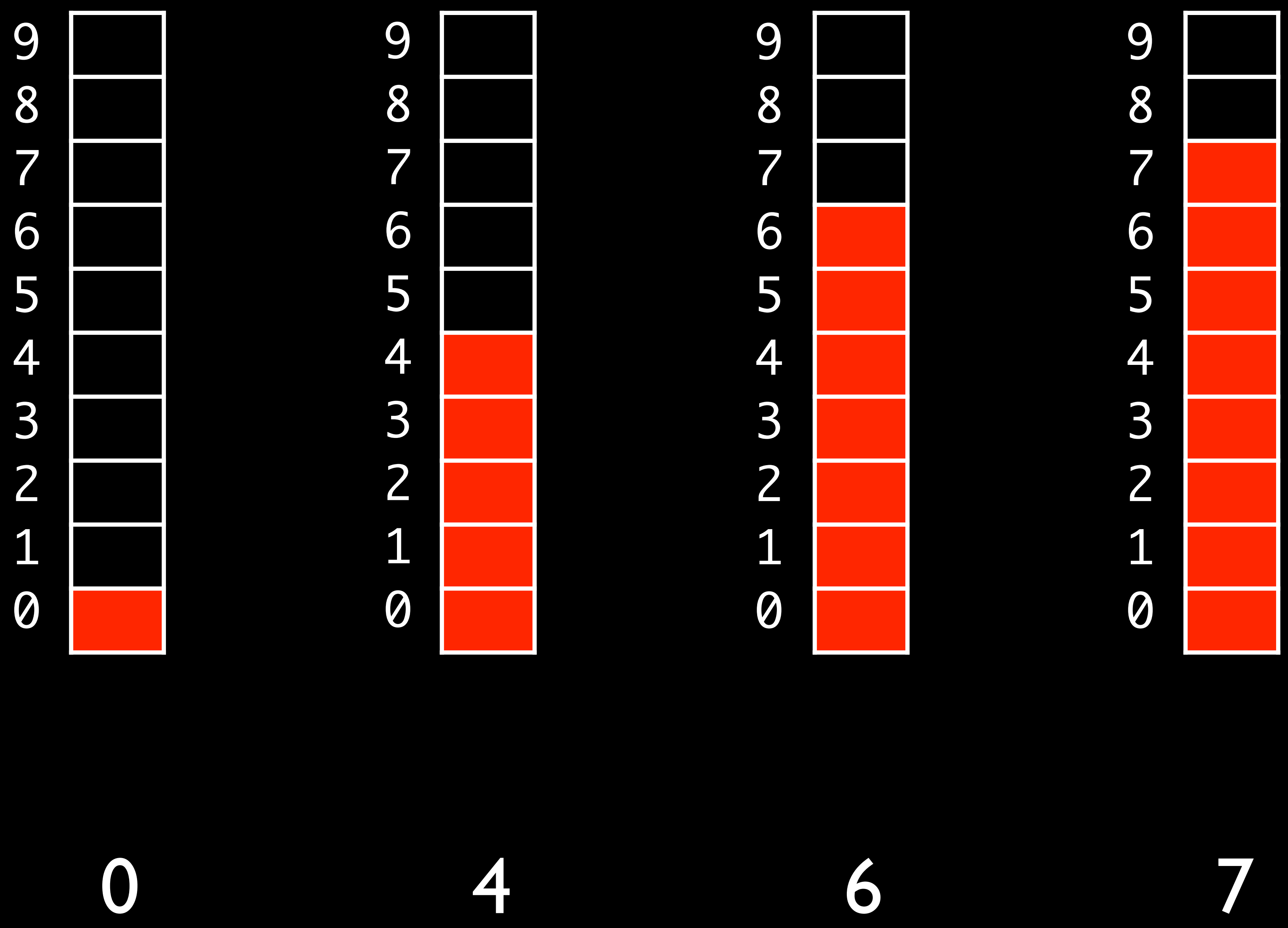
max value = $2^4 - 1$

4-bit binary number

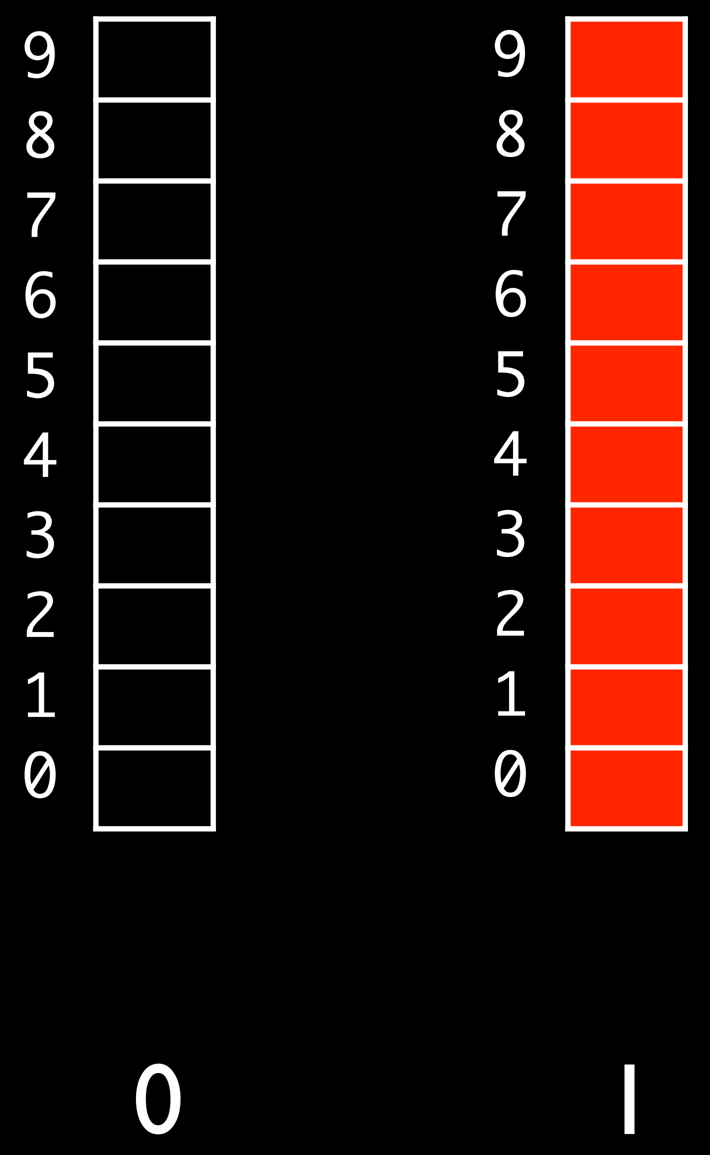
# Binary Numbers (why?)

reliability!

# Binary Numbers (why?)



# Binary Numbers (why?)



How do we encode negative numbers?

# Binary Numbers

use left-most bit to represent sign

0 = “+”

1 = “-”

# Binary Numbers

sign $2^1 2^0$

| | |
|---|---|
| 000 | 0 |
| 001 | 1 |
| 010 | 2 |
| 011 | 3 |
| 100 | |
| 101 | |
| 110 | |
| 111 | |

3-bit *signed* binary number

# Binary Numbers

sign $2^1 2^0$

| | | |
|---|---|---|
| 000 | 0 | |
| 001 | 1 | |
| 010 | 2 | |
| 011 | 3 | |
| 100 | -0 | ??? |
| 101 | -1 | |
| 110 | -2 | |
| 111 | -3 | |

3-bit *signed* binary number

# Binary Numbers (two's complement)

1. start with an unsigned 4-bit binary number where left-most bit is 0
    - `0110 = 6`

# Binary Numbers (two's complement)

1. start with an unsigned 4-bit binary number where left-most bit is 0
   - `0110 = 6`
2. complement your binary number (flip bits)
   - `1001`

# Binary Numbers (two's complement)

1. start with an unsigned 4-bit binary number where left-most bit is 0
   - `0110 = 6`
2. complement your binary number (flip bits)
   - `1001`
3. add one to your binary number
   - `1010 = -6`

# Binary Numbers (two's complement)

| positive | | complement | +1 | negative |
|---|---|---|---|---|
| 0 | 000 | | | 0 |
| 1 | 001 | | | -1 |
| 2 | 010 | | | -2 |
| 3 | 011 | | | -3 |

3-bit *signed* binary number

# Binary Numbers (two's complement)

| positive | | complement | +1 | negative |
|---|---|---|---|---|
| 0 | 000 | | | 0 |
| 1 | 001 | → 110 | → 111 | -1 |
| 2 | 010 | | | -2 |
| 3 | 011 | | | -3 |

3-bit *signed* binary number

# Binary Numbers (two's complement)

| positive | | complement | +1 | negative |
|---|---|---|---|---|
| 0 | 000 | | | 0 |
| 1 | 001 | → 110 | → 111 | -1 |
| 2 | 010 | → 101 | → 110 | -2 |
| 3 | 011 | | | -3 |

3-bit *signed* binary number

# Binary Numbers (two's complement)

| positive | | complement | +1 | negative |
|---|---|---|---|---|
| 0 | 000 | | | 0 |
| 1 | 001 | → 110 | → 111 | -1 |
| 2 | 010 | → 101 | → 110 | -2 |
| 3 | 011 | → 100 | → 101 | -3 |

3-bit *signed* binary number

# Binary Numbers (two's complement)

| positive | | complement | +1 | negative |
|---|---|---|---|---|
| 0 | 000 | → 111 | → | 0 |
| 1 | 001 | → 110 | → 111 | -1 |
| 2 | 010 | → 101 | → 110 | -2 |
| 3 | 011 | → 100 | → 101 | -3 |

3-bit *signed* binary number

# Binary Numbers (two’s complement)

| positive | | complement | +1 | negative | |
|---|---|---|---|---|---|
| 0 | 000 → | 111 → | 000 | 0 | !!! |
| 1 | 001 → | 110 → | 111 | -1 | |
| 2 | 010 → | 101 → | 110 | -2 | |
| 3 | 011 → | 100 → | 101 | -3 | |

3-bit *signed* binary number

# Binary Numbers (two's complement)

| positive | | complement | +1 | negative |
|---|---|---|---|---|
| 0 | 000 | → 111 | → 000 | 0 |
| 1 | 001 | → 110 | → 111 | -1 |
| 2 | 010 | → 101 | → 110 | -2 |
| 3 | 011 | → 100 | → 101 | -3 |

we lost a number?

# Binary Numbers (two's complement)

| positive | | complement | +1 | negative |
|---|---|---|---|---|
| 0 | 000 | 111 | 000 | 0 |
| 1 | 001 | 110 | 111 | -1 |
| 2 | 010 | 101 | 110 | -2 |
| 3 | 011 | 100 | 101 | -3 |
| | | | 100 | |

we lost a number?

# Binary Numbers (two’s complement)

complement $\longleftarrow$ -1 $\longleftarrow$ 100

# Binary Numbers (two's complement)

complement          -1

$\longleftarrow$ 011 $\longleftarrow$ 100

| | |
|---|---|
| 010 | -2 |
| 011 | -1 |
| 100 | |
| 101 | +1 |
| 110 | +2 |

# Binary Numbers (two's complement)

$$100 \xleftarrow{\text{complement}} 011 \xleftarrow{-1} 100$$

# Binary Numbers (two's complement)

| | | complement | | -1 | |
|---|---|---|---|---|---|
| 4 | 100 | ⟵ | 011 | ⟵ | 100 |

# Binary Numbers (two's complement)

| positive | | | complement | +1 | | negative |
|---|---|---|---|---|---|---|
| 0 | 000 | → | 111 | → | 000 | 0 |
| 1 | 001 | → | 110 | → | 111 | -1 |
| 2 | 010 | → | 101 | → | 110 | -2 |
| 3 | 011 | → | 100 | → | 101 | -3 |
| | | | | | 100 | -4 |

n-bit *unsigned* binary numbers: $0 \ldots 2^n - 1$

# Binary Numbers (two's complement)

| positive | | | complement | +1 | negative |
|---|---|---|---|---|---|
| 0 | 000 | → | 111 | → 000 | 0 |
| 1 | 001 | → | 110 | → 111 | -1 |
| 2 | 010 | → | 101 | → 110 | -2 |
| 3 | 011 | → | 100 | → 101 | -3 |
| | | | | 100 | -4 |

n-bit *signed* binary numbers: $-2^{n-1} \ldots 2^{n-1}-1$

# Binary Numbers (two's complement)

```
  0010      2
  0010      2
+ ----    + -
  0100      4
```

summing unsigned binary numbers is easy

# Binary Numbers (two's complement)

```
  0010      2
  1010     -2
+ ----    + -
  1100      0      ?
```

summing signed binary numbers

# Binary Numbers (two's complement)

```
  0011      3
  1011     -3
+ ----    + -
  1110      0      ?
```

summing signed binary numbers

# Binary Numbers (two's complement)

```
0010 -> 1101 -> 1110
```

```
         0010        2
                    -2
       + ----     + -
                     0
```

summing signed (2's complement) binary numbers

# Binary Numbers (two's complement)

```
0010 -> 1101 -> 1110

     0010       2
     1110      -2
   + ----     + -
     0000       0
```

summing signed (2‘s complement) binary numbers

# Binary Numbers (two's complement)

```
0011 -> 1100 -> 1101

          0011        3
                     -3
        + ----     + -
                      0
```

summing signed (2's complement) binary numbers

# Binary Numbers (two's complement)

```
0011 -> 1100 -> 1101

          0011     3
          1101    -3
        + ----   + -
          0000     0
```

summing signed (2's complement) binary numbers

# Binary Numbers (decoding two's complement)

`0111 = ?`

4-bit signed (two's complement) binary number

# Binary Numbers (decoding two's complement)

`0111 = 7`

4-bit signed (two's complement) binary number

# Binary Numbers (decoding two's complement)

`1011 = ?`

4-bit signed (two's complement) binary number

# Binary Numbers (decoding two's complement)

subtract 1

1011 1010

4-bit signed (two's complement) binary number

# Binary Numbers (decoding two's complement)

| | subtract 1 | complement |
|---|---|---|
| 1011 | 1010 | 0101 |

4-bit signed (two's complement) binary number

# Binary Numbers (decoding two's complement)

| | subtract 1 | complement | |
|---|---|---|---|
| 1011 | 1010 | 0101 | 5 |

4-bit signed (two's complement) binary number

# Binary Numbers (decoding two's complement)

`1011 = -5`

4-bit signed (two's complement) binary number

How do we encode fractional numbers?

# Binary Numbers

$$\pm \text{ mantissa} \times \text{base}^{\pm \text{ exponent}}$$

# Boolean Logic (variables)

1 = True

0 = False

# Boolean Logic (truth tables)

| **a** | **b** | **a** *and* **b** |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

**a** *and* **b**

# Boolean Logic (truth tables)

| **a** | **b** | **a** *or* **b** |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

**a** *or* **b**

# Boolean Logic (truth tables)

| **a** | *not* **a** |
|---|---|
| 1 | 0 |
| 0 | 1 |

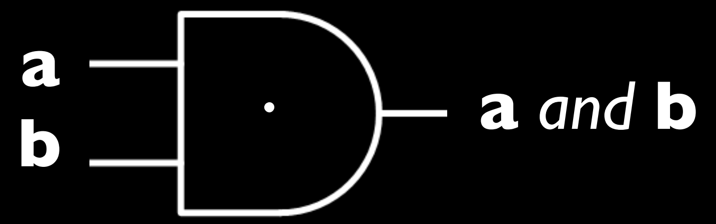
*not* **a**

# Boolean Logic (truth tables)

| *input* (boolean variable) | *output* (boolean variable) |
|---|---|
| **a**, **b** | **a** *and* **b** |
| | **a** *or* **b** |
| | *not* **a** |

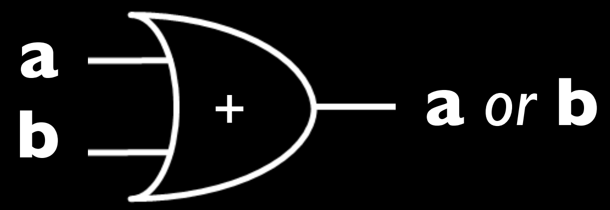
# Gates

| **a** | **b** | **a** *and* **b** |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |



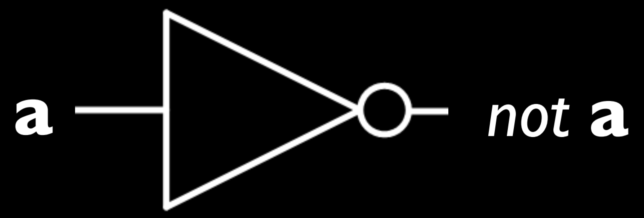
# Gates

| **a** | **b** | **a** *or* **b** |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

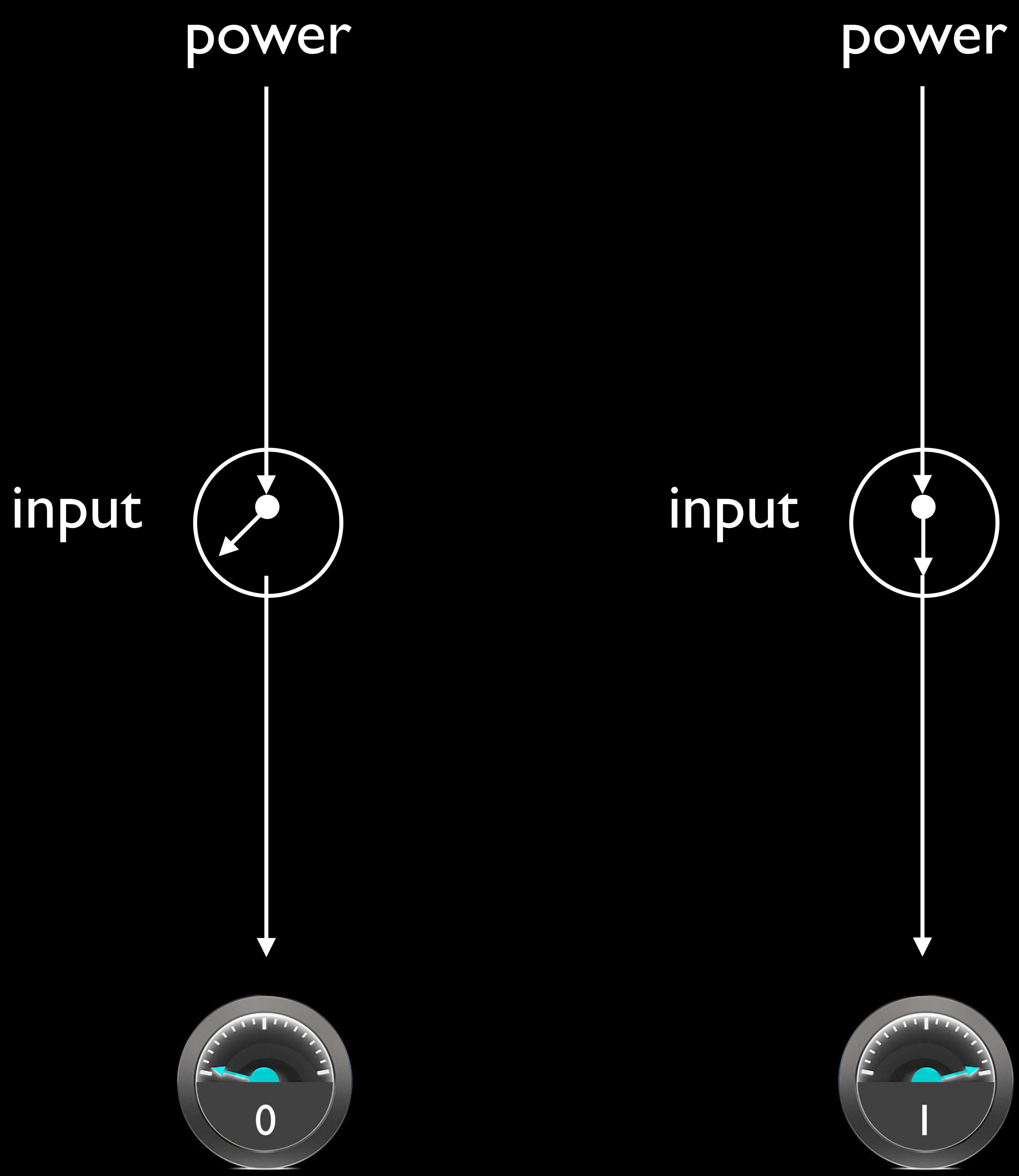


# Gates

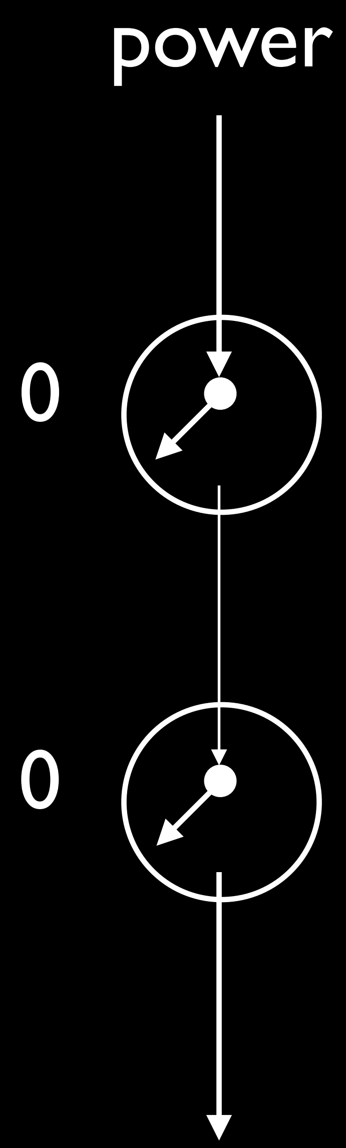
| **a** | *not* **a** |
|---|---|
| 1 | 0 |
| 0 | 1 |



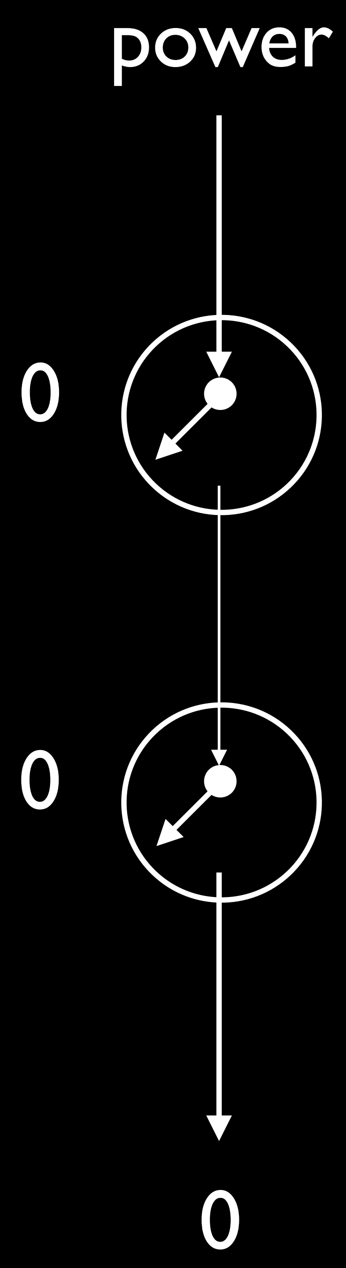
# Building Gates (transistors)



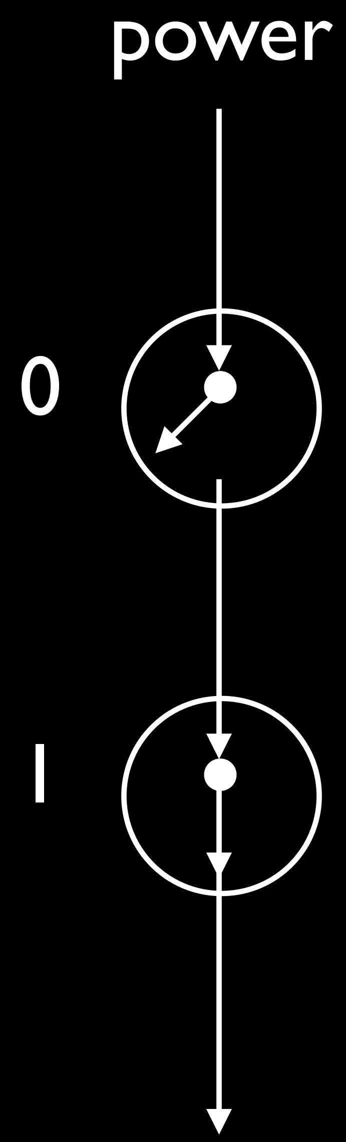
# Building Gates (transistors)



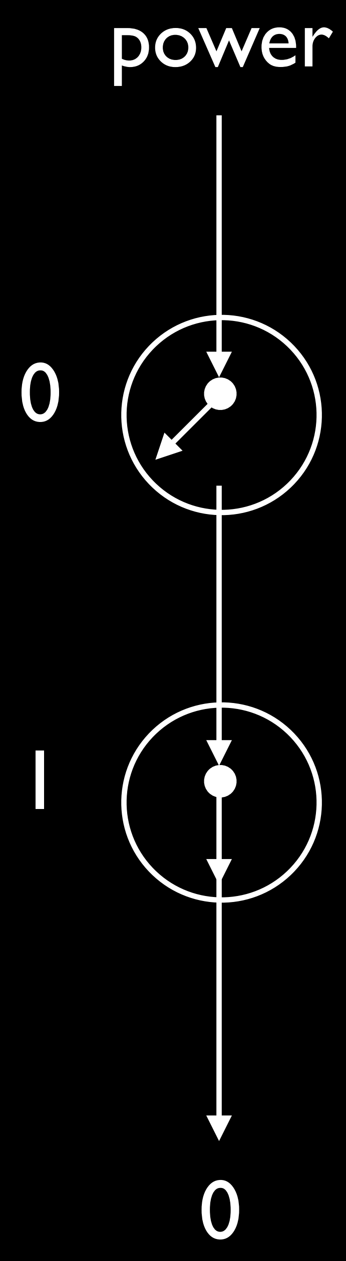
# Building Gates (transistors)



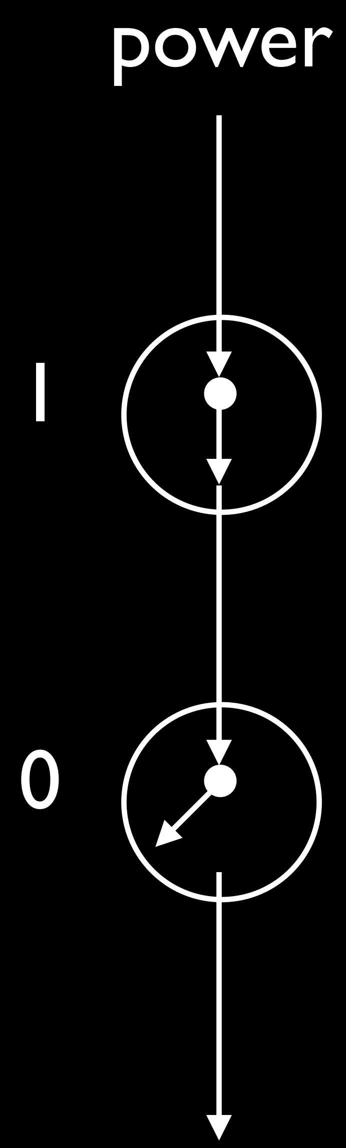
# Building Gates (transistors)



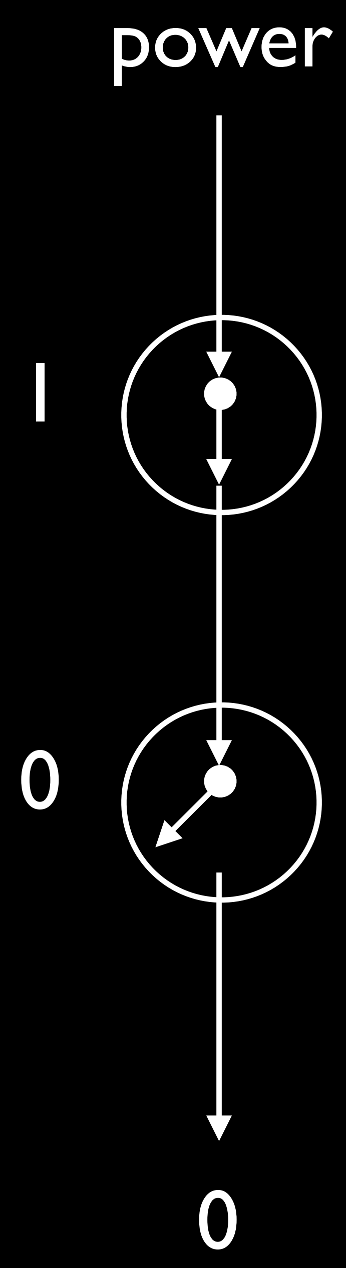
# Building Gates (transistors)



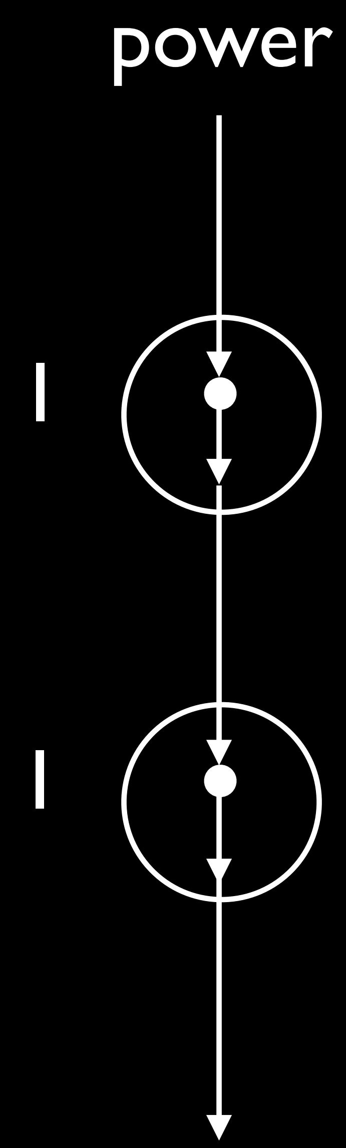
# Building Gates (transistors)



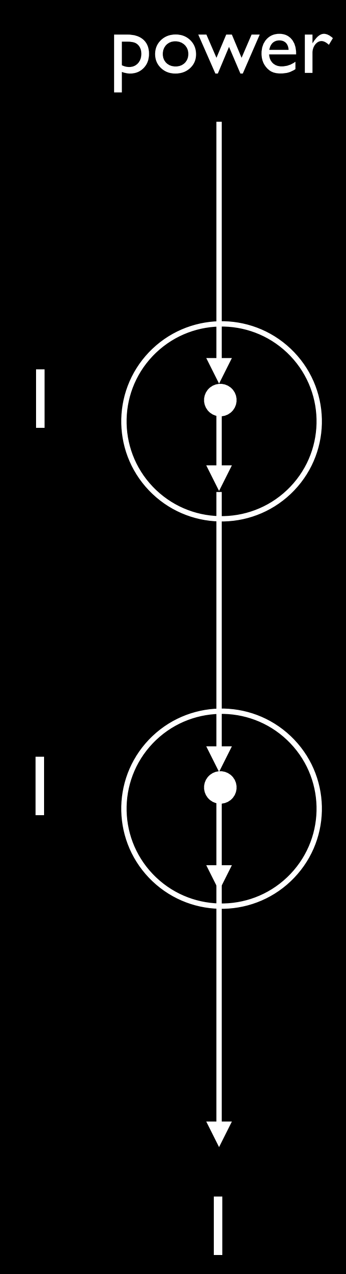
# Building Gates (transistors)



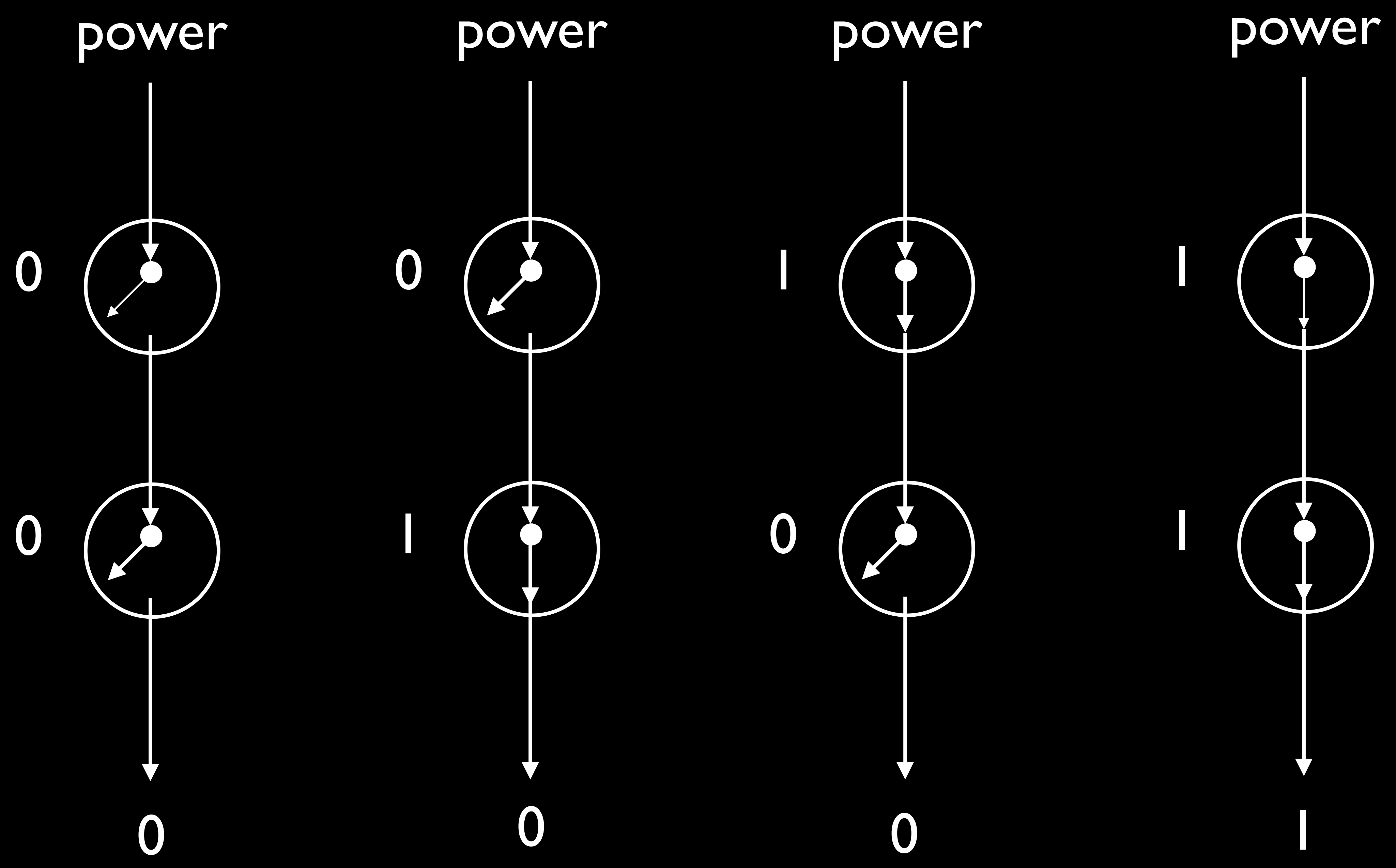
# Building Gates (transistors)



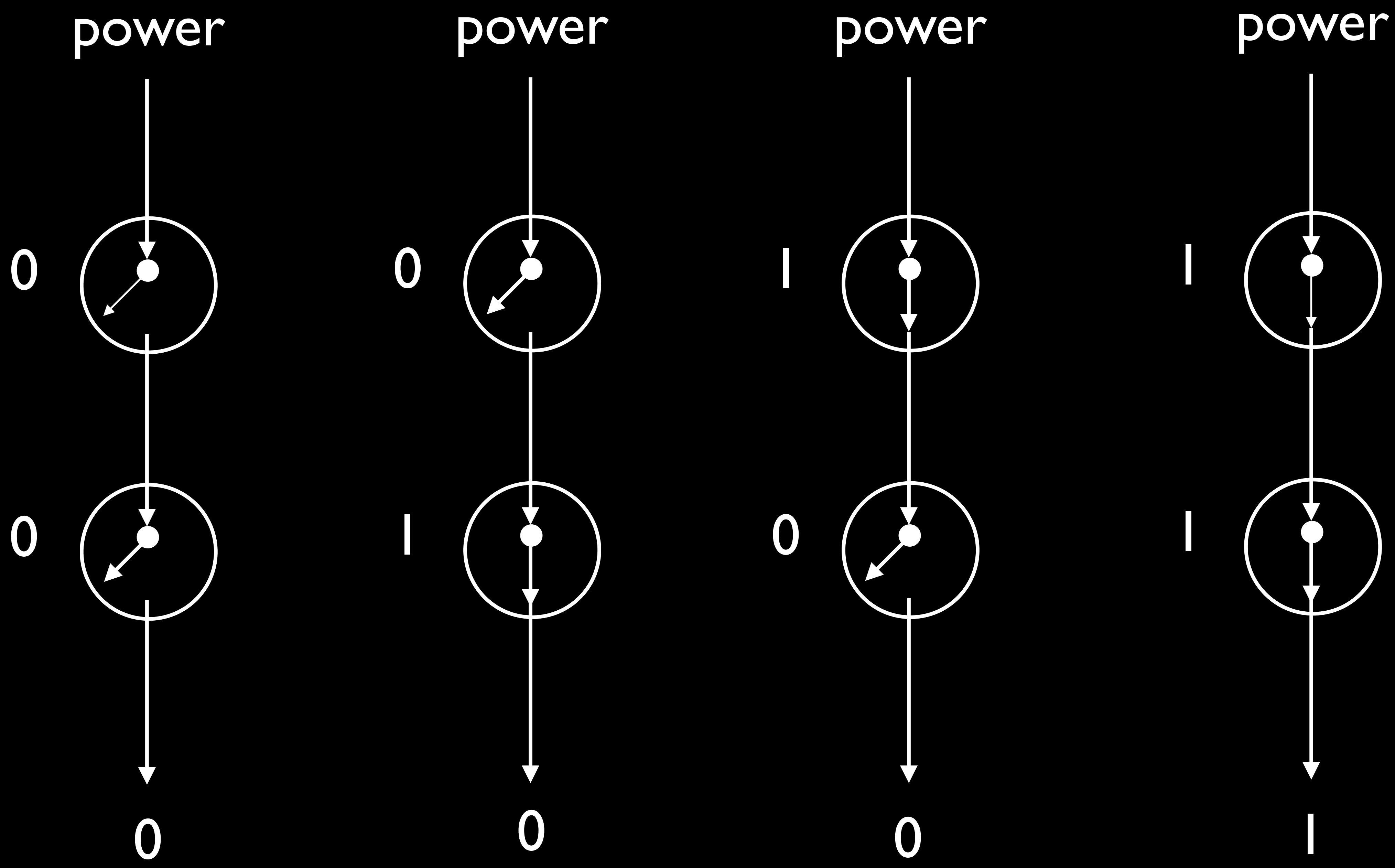
# Building Gates (transistors)



# Building Gates (transistors)

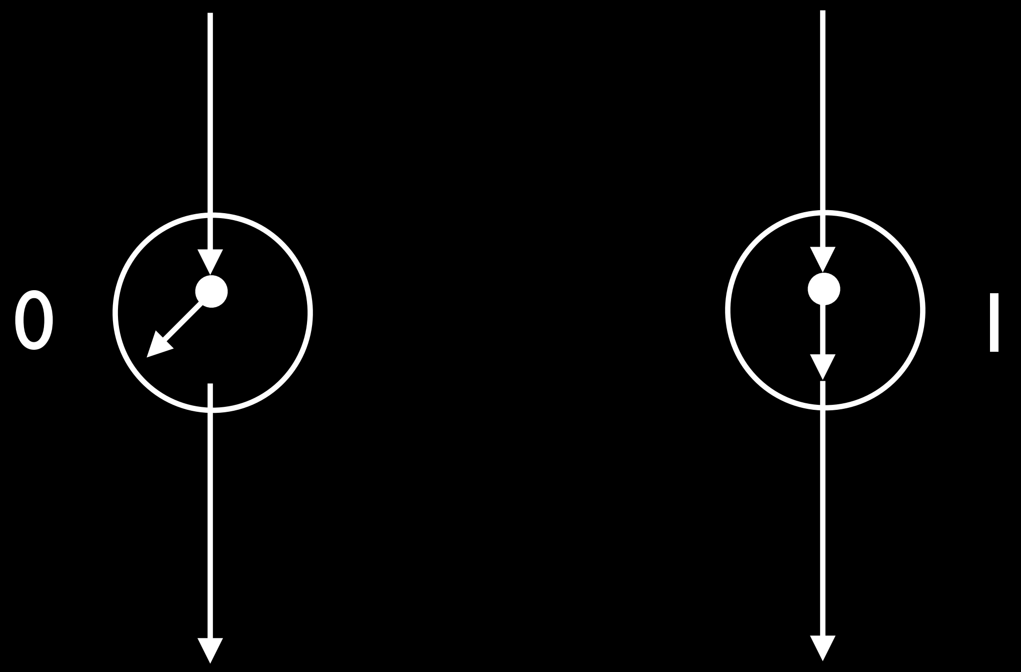


# Building Gates (transistors)



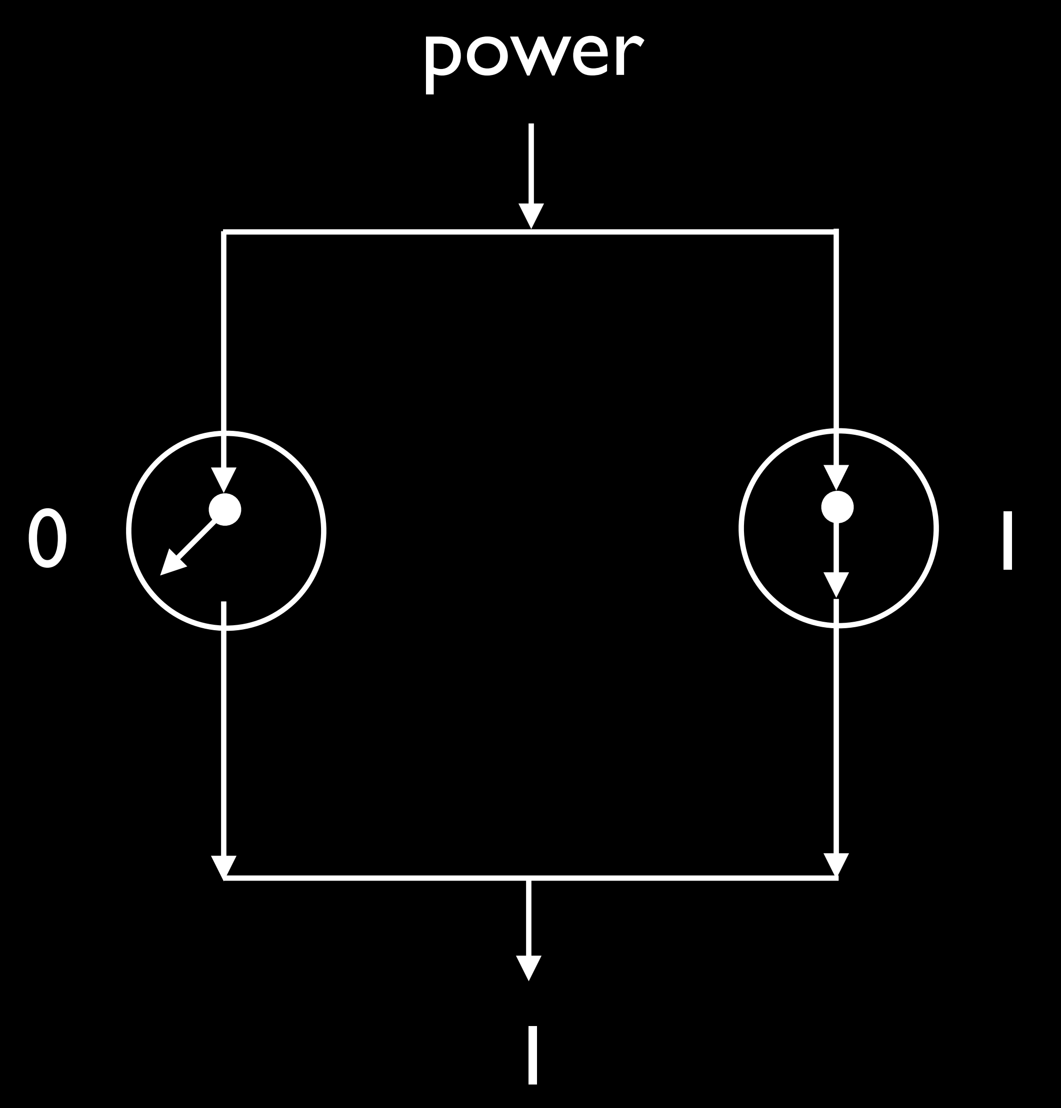

AND gate

# Building Gates (transistors)



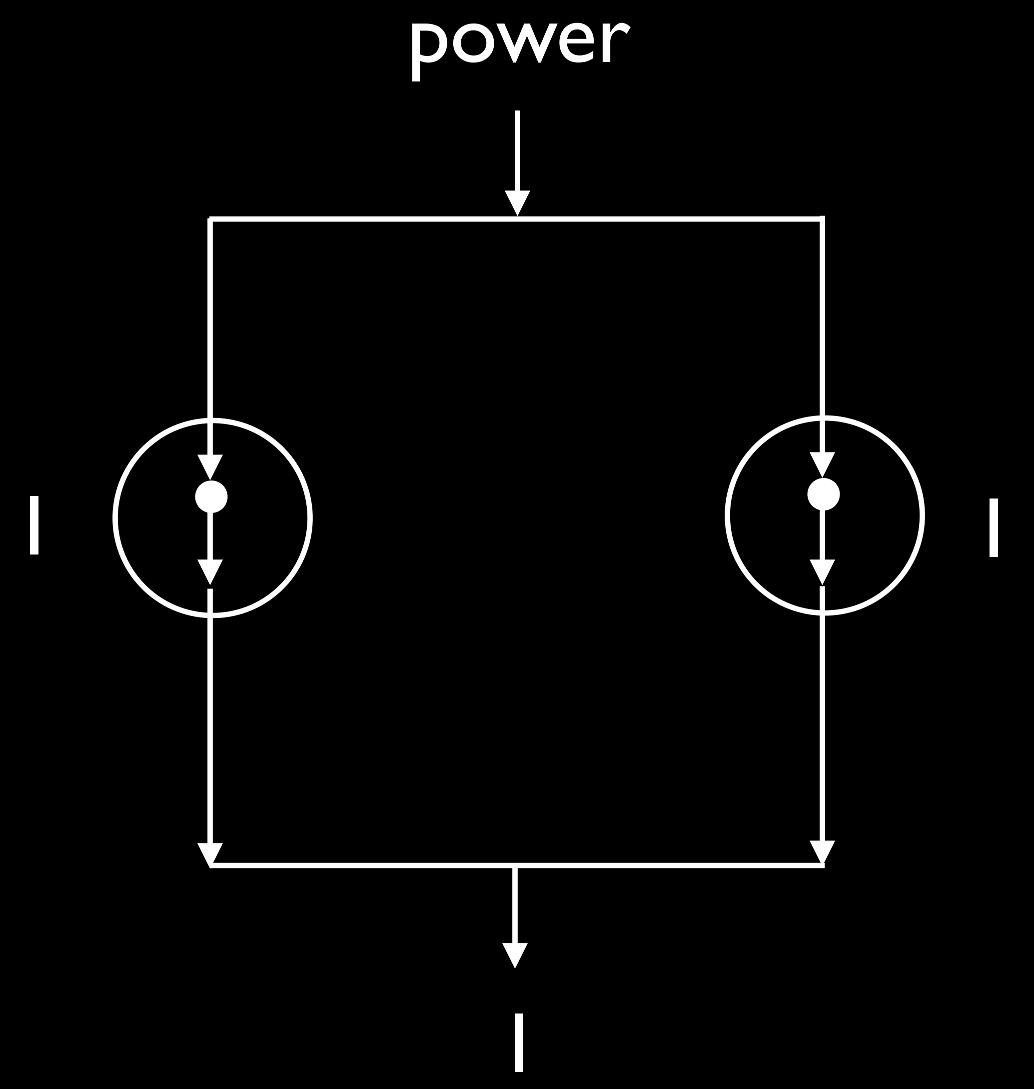

OR gate

# Building Gates (transistors)



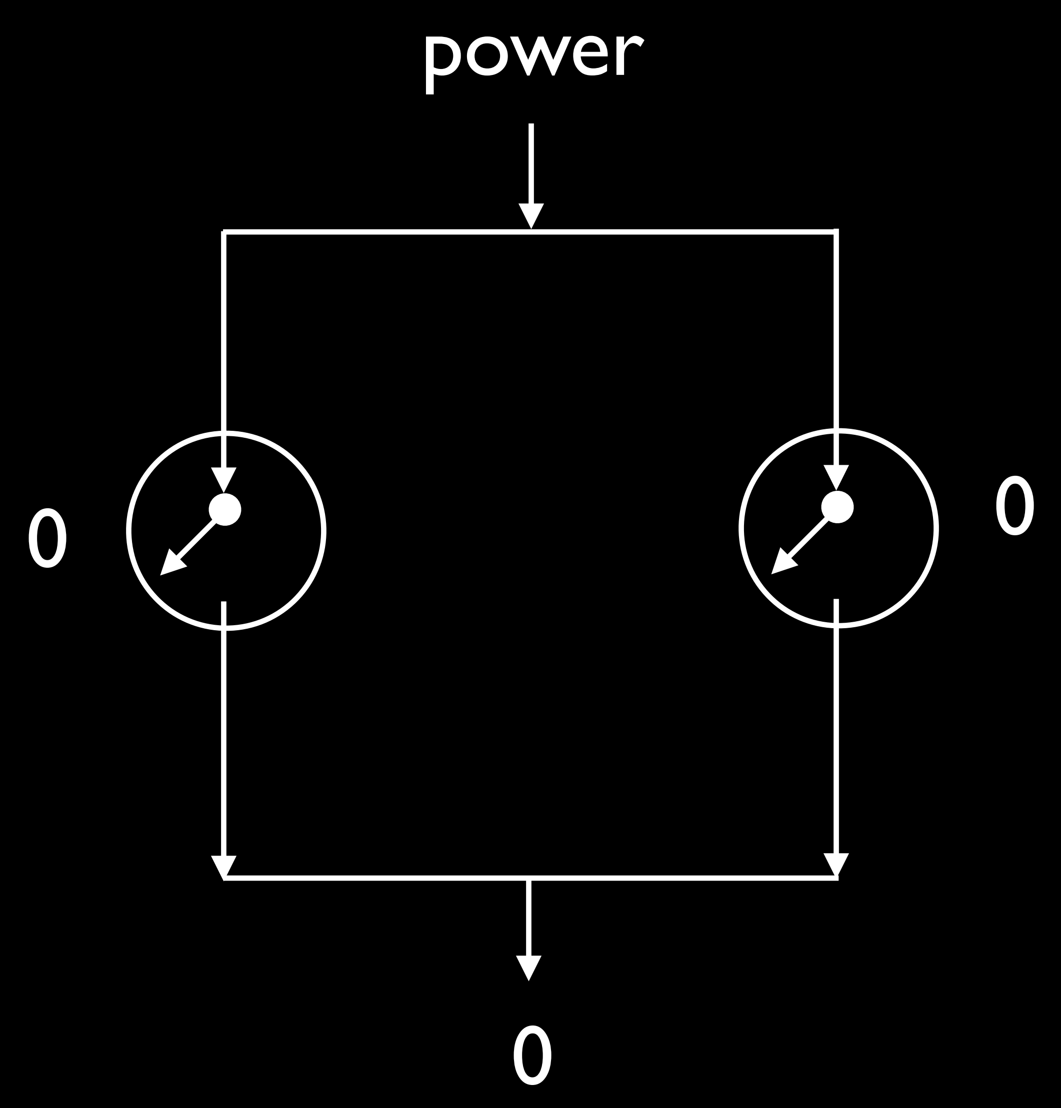

OR gate

# Building Gates (transistors)



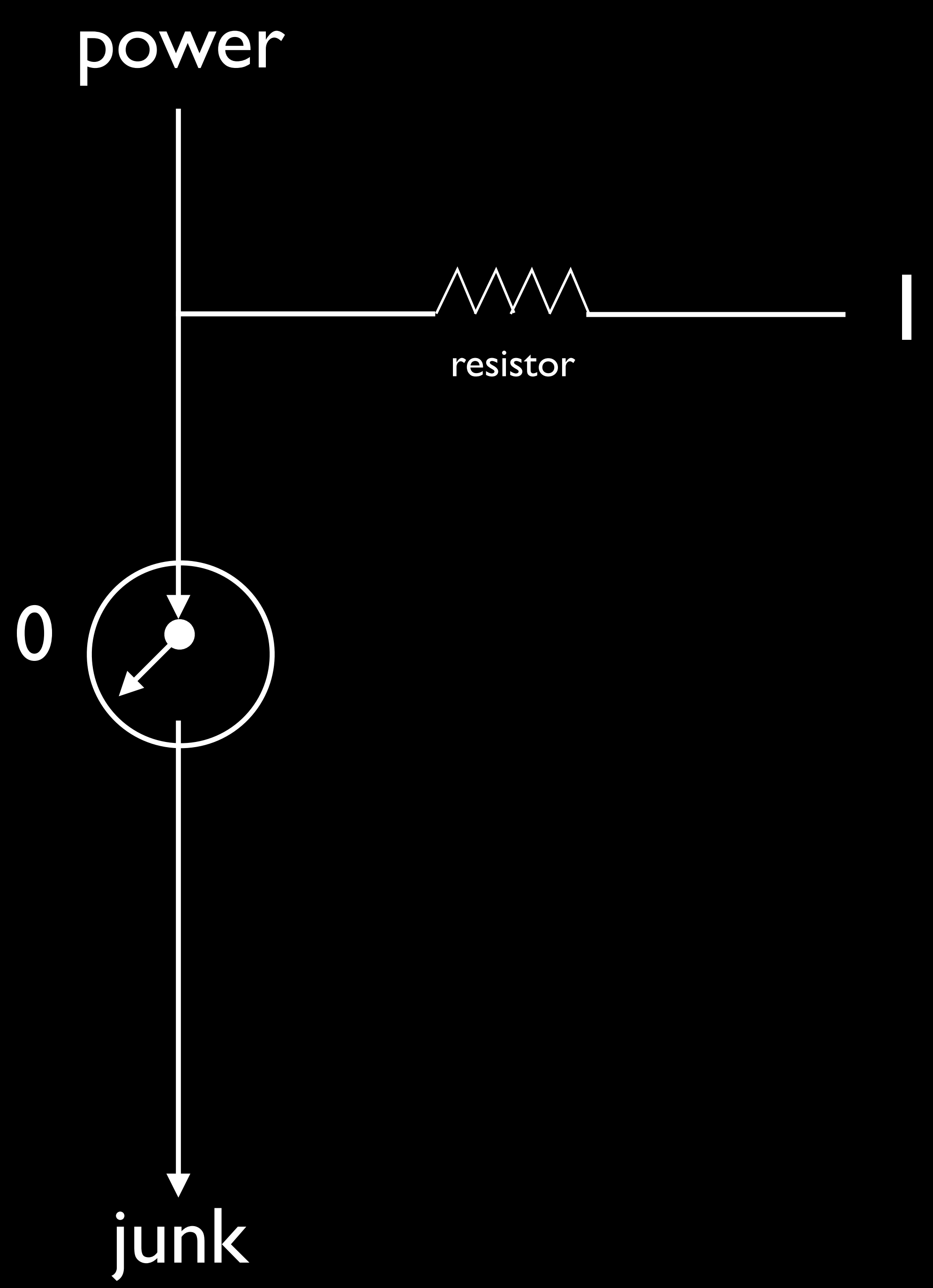

OR gate

# Building Gates (transistors)



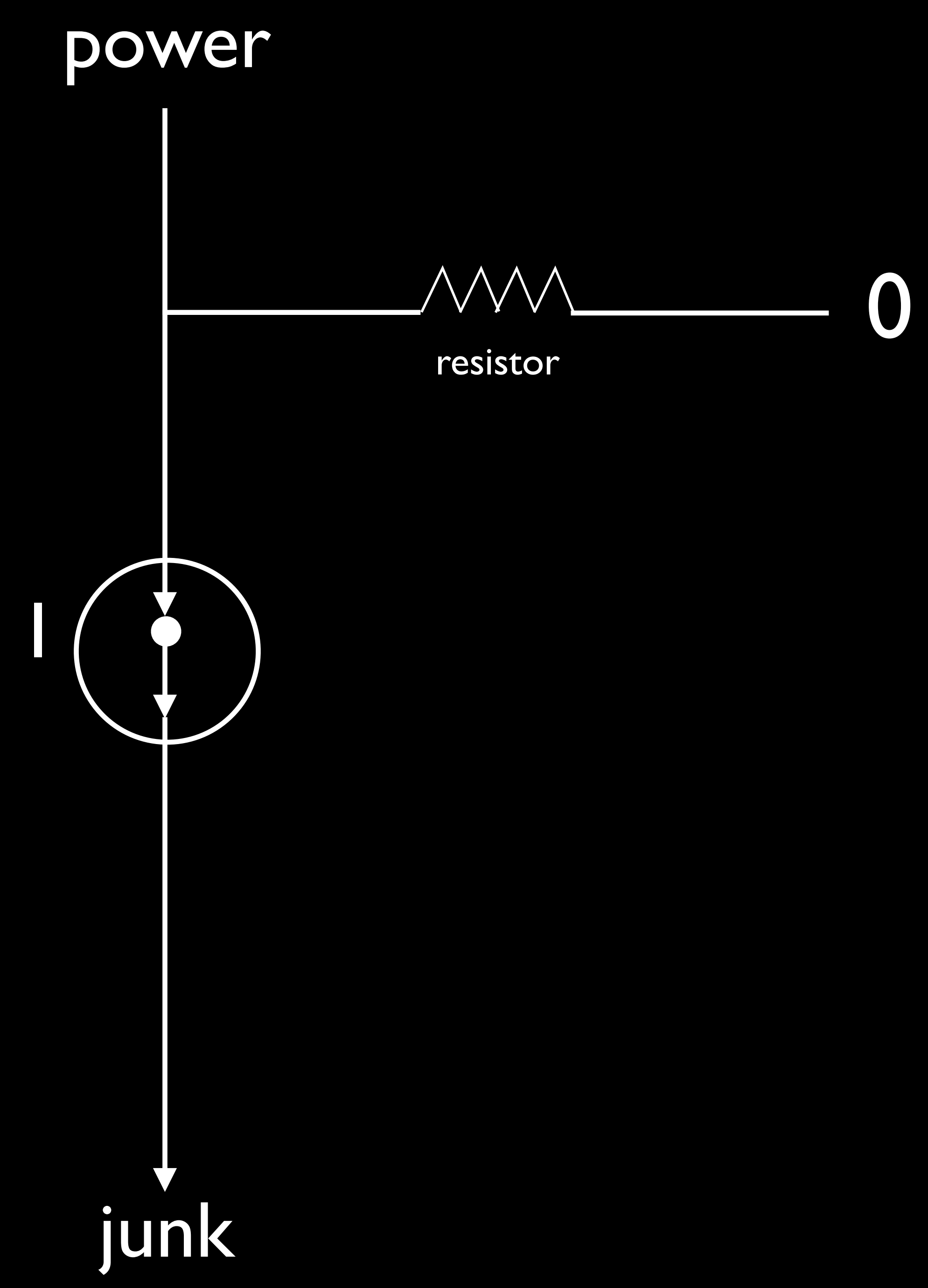

OR gate

# Building Gates (transistors)




NOT gate

# Building Gates (transistors)




NOT gate