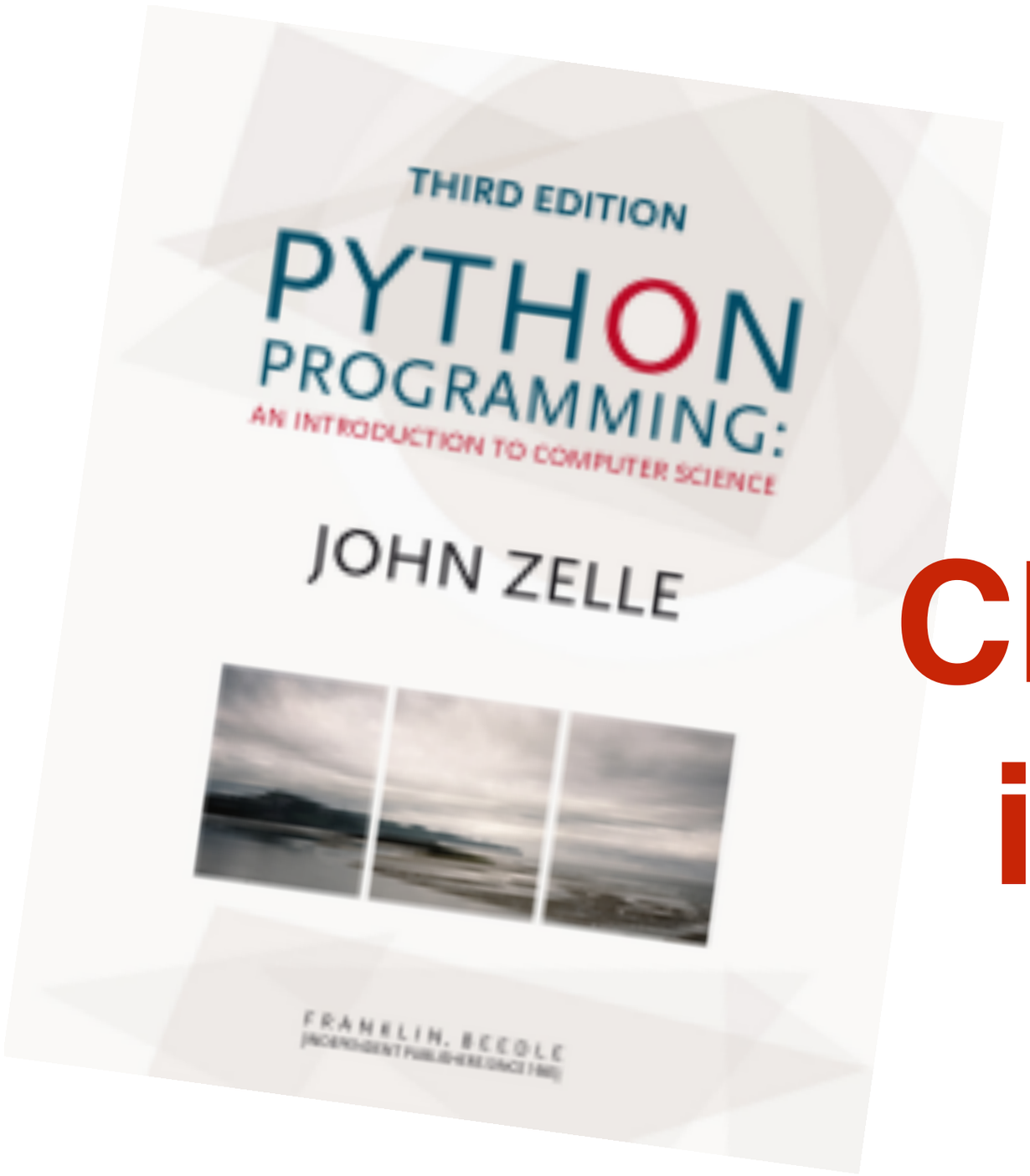
Smith College
Computer Science

Lecture Notes
CSC111

Week 3 — Sept 21, 2015

Dominique Thiébaut
dthiebaut@smith.edu



Chapter 3 in Zelle

Programming Tips

- **Never** try to solve the whole problem at once
- Figure out how to **solve smaller problems** and merge pieces of code together
- Replace **inputs** by **assignments** until the last steps
- Make the program **print intermediate** values as **debug**ging help. Remove these print statements at the end.

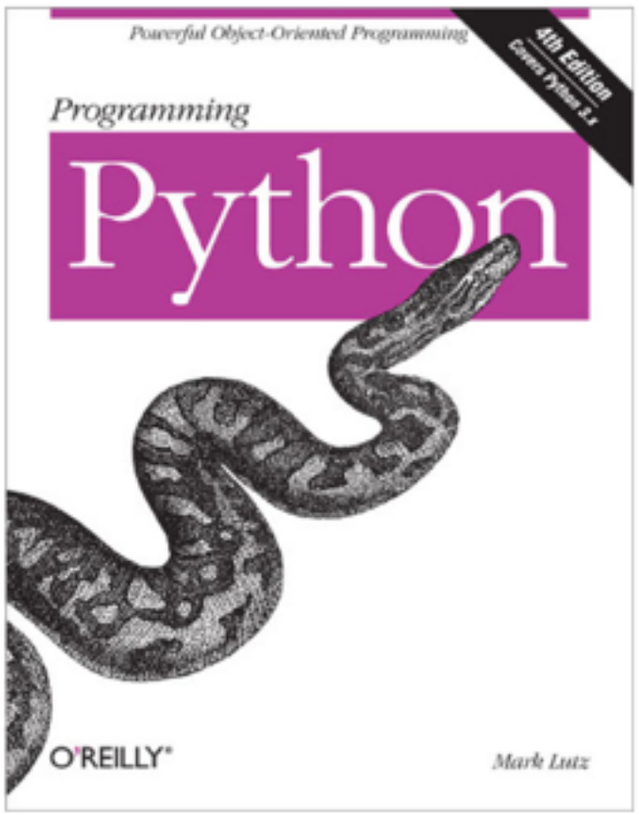
Some Apps Written in Python

(2014)

- YouTube
- Dropbox
- Google
- Quora
- Instagram
- BitTorrent
- Spotify
- Reddit
- Yahoo Maps
- Hipmunk

WHAT ARE THE 10 MOST FAMOUS SOFTWARE PROGRAMS WRITTEN IN PYTHON?

by HSG on Mar 19, 2014 in Articles from Software Fans



Python is an incredibly powerful and useful computer programming language that many of the biggest websites in the world rely on for their foundation. Python provides reliable results that are functional and involve a variety of dynamic scripted and non-scripted contexts. And because it is free and open source, it has remained a popular choice for a variety of different developers who are looking to build new sites on one of the most reliable languages available. Here is a look at 10 of the most famous software programs that are written in Python and what they do.

YouTube

If you love watching hours of homemade and professional quality video clips on YouTube, you can thank

http://www.hartmannsoftware.com/Blog/Articles_from_Software_Fans/Most-Famous-Software-Programs-Written-in-Python

Information about Midterm Exam:

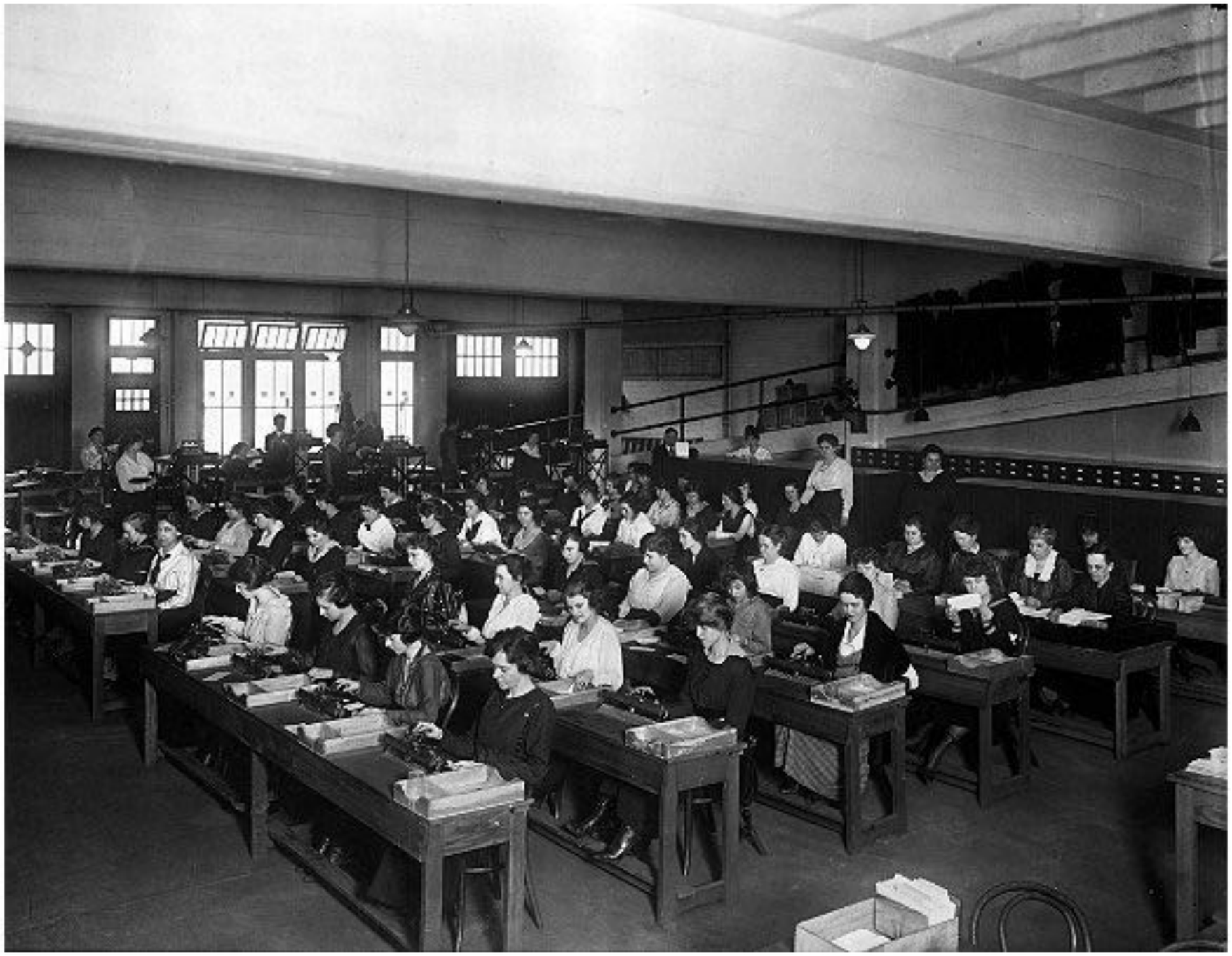
Timed, on Moodle

Closed books

Closed notes

Closed Web

Closed Idle



When computers were human, a talk
by David Grier: start at 28m12s, for 4 minutes.
http://youtu.be/YwqltwvPnkw?t=28m12s

Arithmetic operators and math functions

Printing numbers to look "nice"

Using a main() function

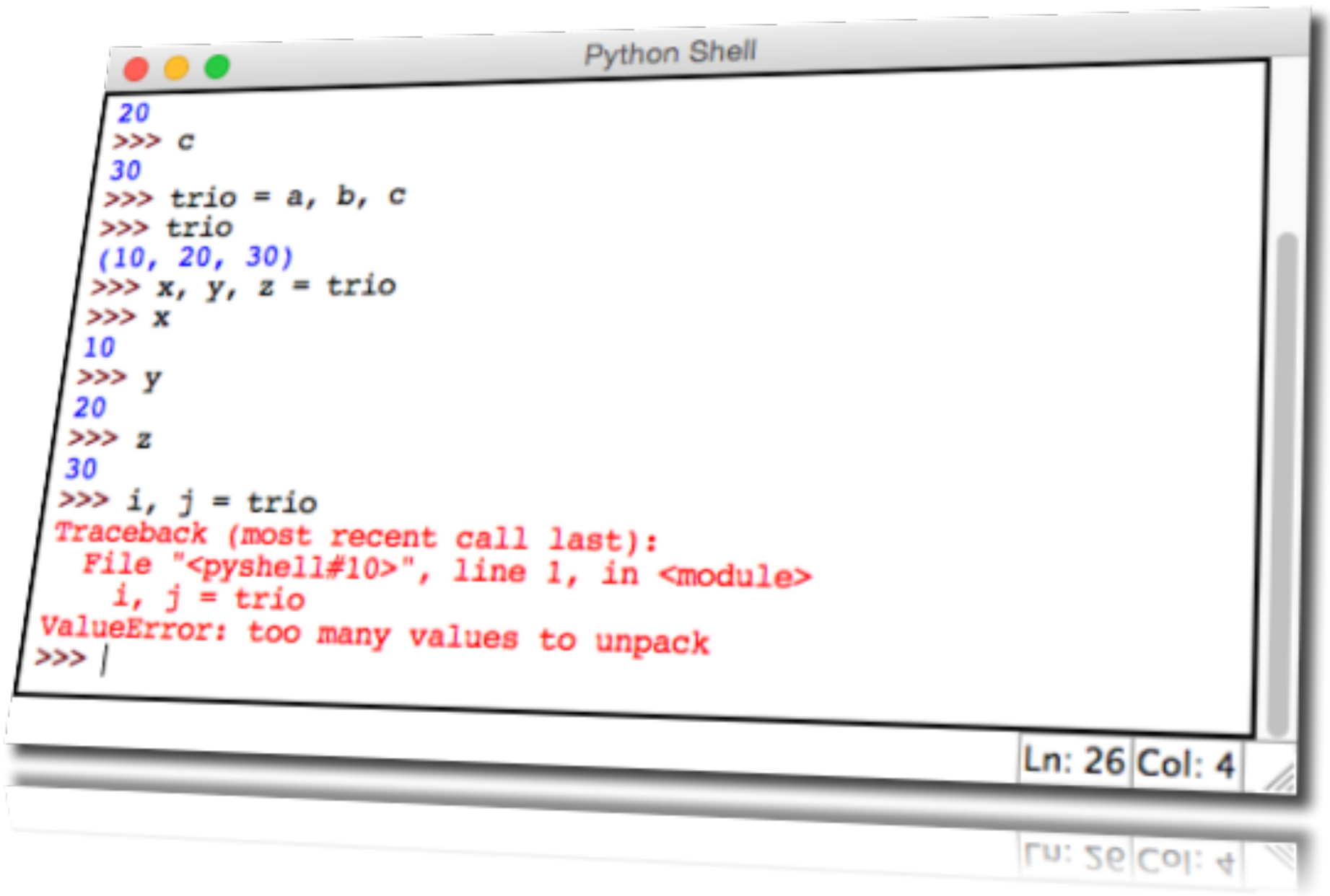
Accumulating results

What are bits?

Arithmetic Operators

Operator	Name	Result
+	sum	float if 1 side is float, else int
-	subtraction	float if 1 side is float, else int
*	multiplication	float if 1 side is float, else int
/	real division	float
//	integer division	int
%	modulo	int
**	exponentiation	float if 1 side is float, else int

Demo Time!


Python Shell
20
>>> c
30
>>> trio = a, b, c
>>> trio
(10, 20, 30)
>>> x, y, z = trio
>>> x
10
>>> y
20
>>> z
30
>>> i, j = trio
Traceback (most recent call last):
File "<pyshell#10>", line 1, in <module>
i, j = trio
ValueError: too many values to unpack
>>>
Ln: 26 Col: 4

Exercise: Writing a Teller Machine Program

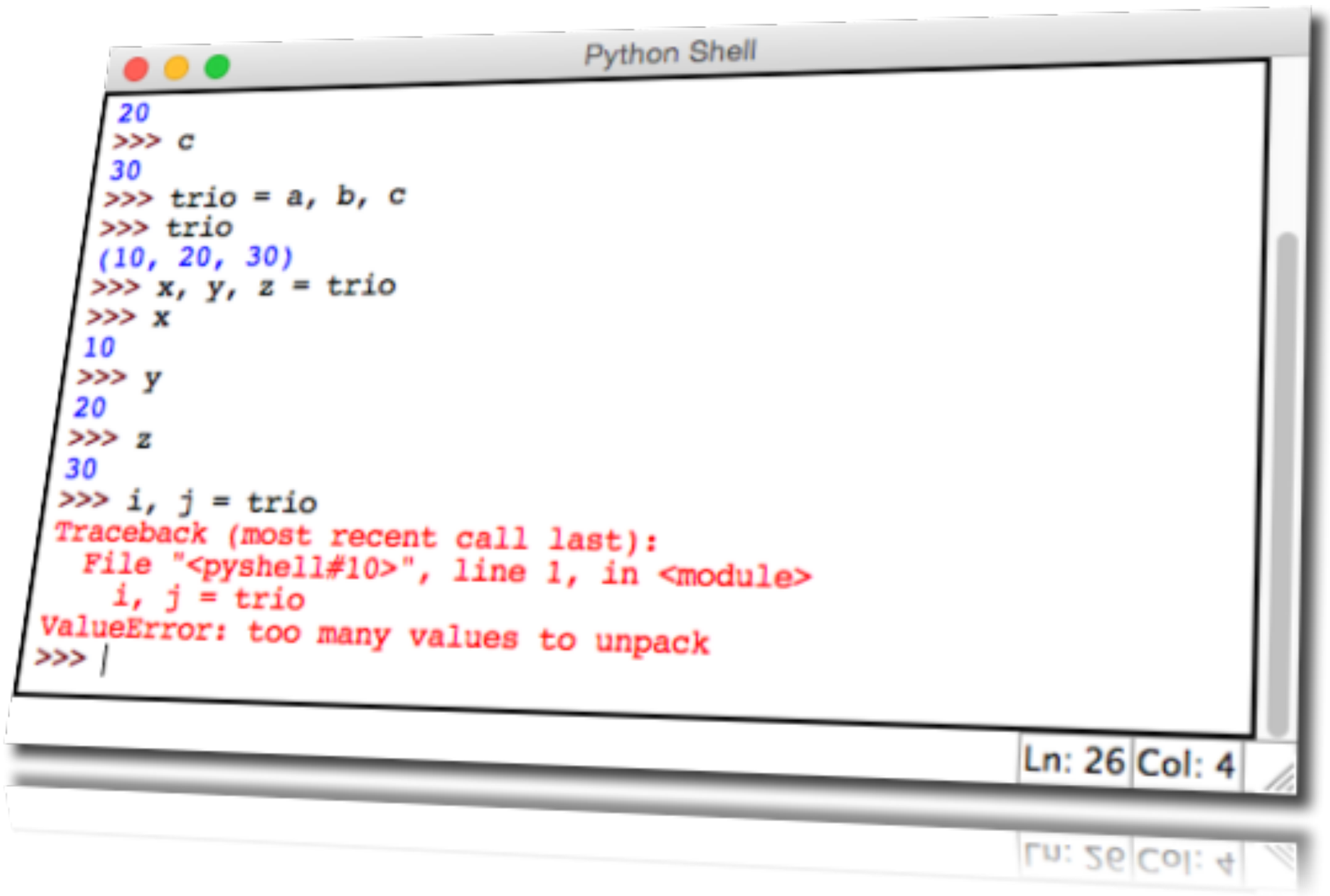
How much money do you want to withdraw?

How much money do you want to withdraw? **139**

How much money do you want to withdraw? **139**

Please find the following bills below:
6 $20-bill(s)
1 $10-bill(s)
1 $5-bill(s)
4 $1-bill(s)

Programming Time!



Arithmetic operators and math functions

Printing numbers to look "nice"

Using a main() function

Accumulating results

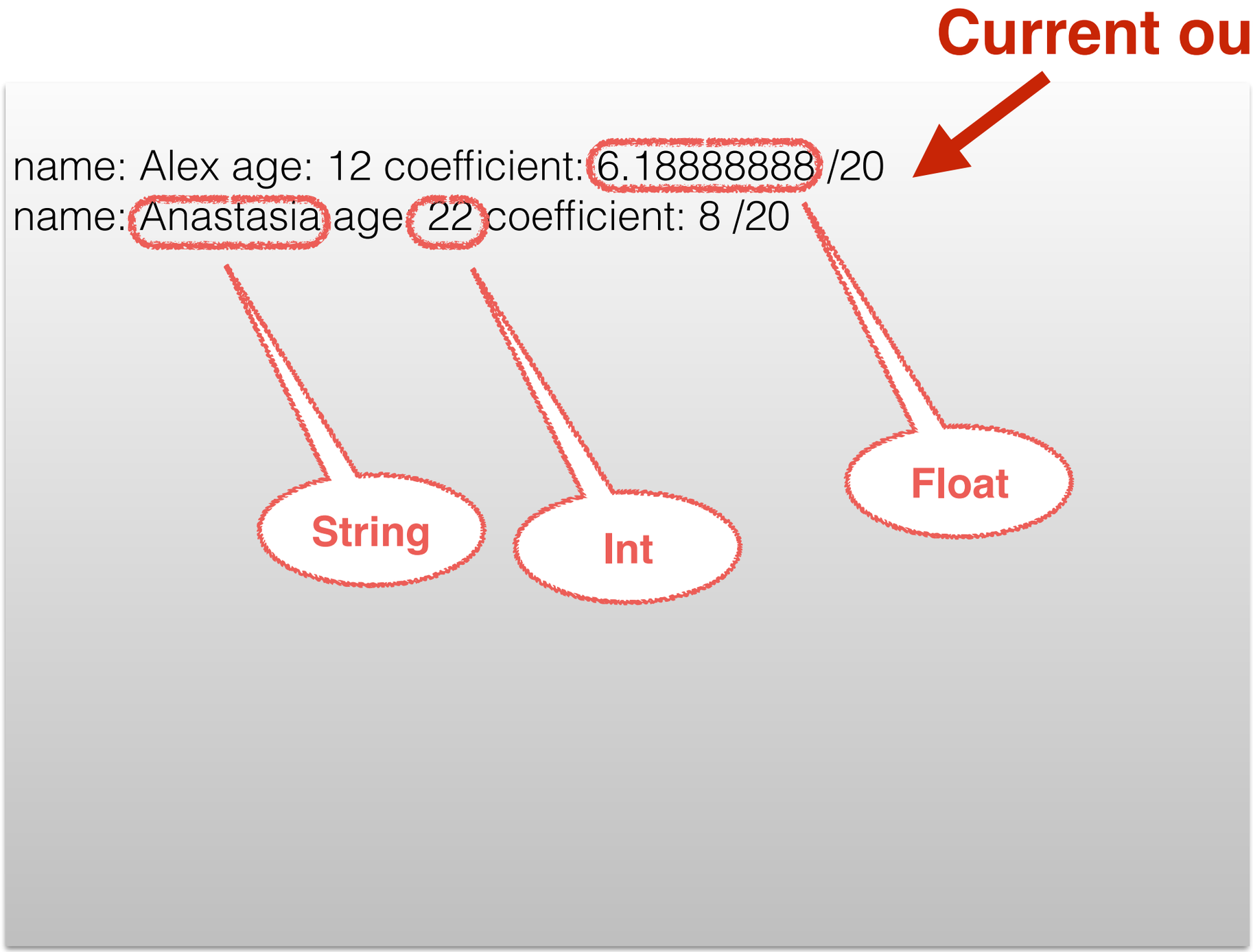
What are bits?

Formatted Output

```
>>> name = "Alex"
>>> age = 12
>>> coef = 6.1888888
>>> print( "name:", name, "age:", age, "coefficient:", coef, "/20" )
name: Alex age: 12 coefficient: 6.1888888 /20
>>>
```

Current output

```
name: Alex age: 12 coefficient: 6.18888888 /20
name: Anastasia age: 22 coefficient: 8 /20
```


Current output
name: Alex age: 12 coefficient: 6.18888888 /20
name: Anastasia age: 22 coefficient: 8 /20
String
Int
Float

Current output

```
name: Alex age: 12 coefficient: 6.18888888 /20
name: Anastasia age: 22 coefficient: 8 /20
```

```
name:          Alex age:  12 coefficient:   6.19/20
name:    Anastasia age:  22 coefficient:   8.00/20
```

Wanted output

(See Section 5.8.2 in Zelle)

```
name = "Alex"
"name: {0:10}X" . format( name )
```

Notation

- The dot . is an important operator
- It works with objects
- In Python, strings are objects

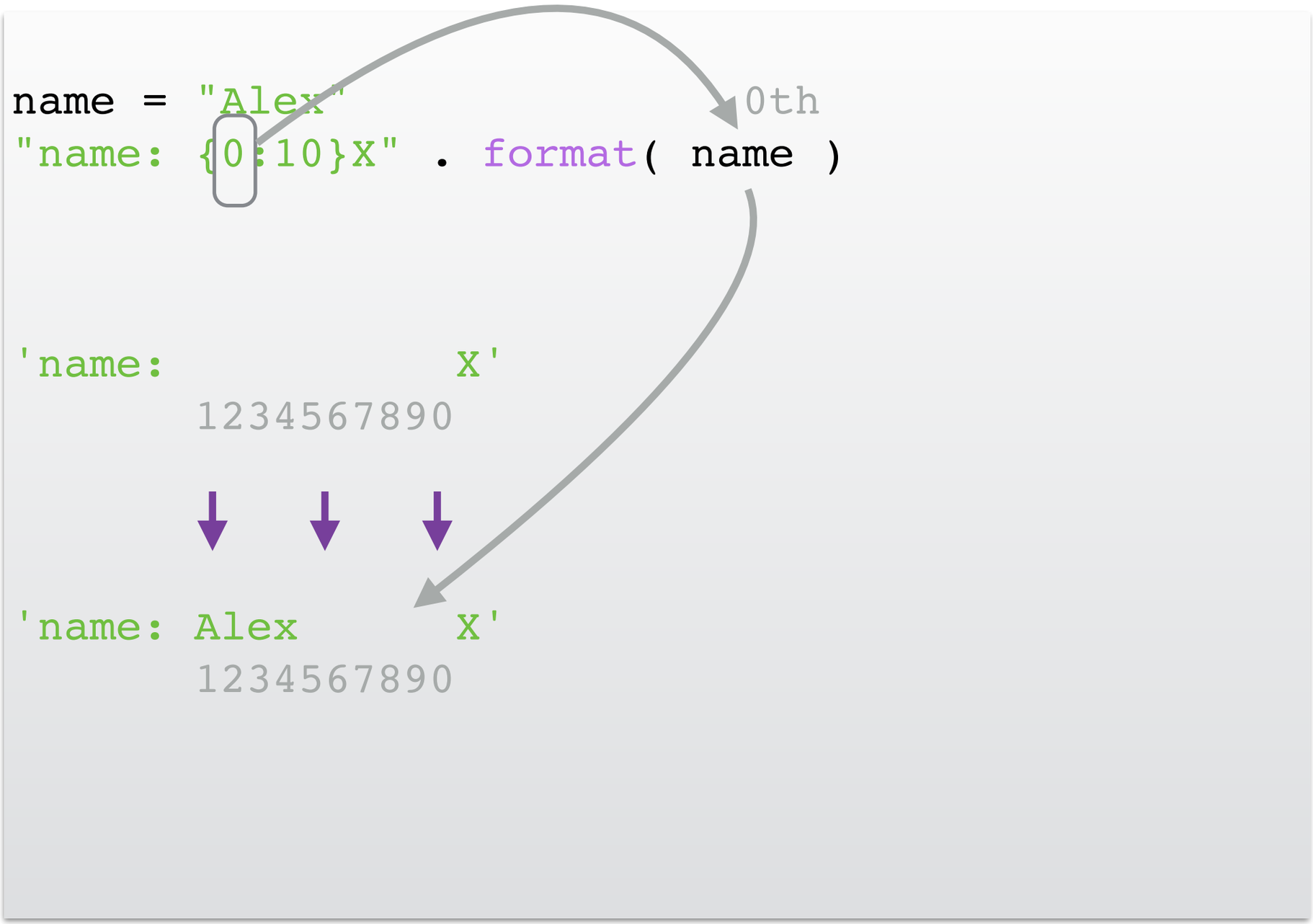
  "**hello {0:10} there!**" . **format**(name)

  —> *"String, please format yourself, and substitute the contents of name inside your brackets"*

```
name = "Alex"
"name: {0:10}X" . format( name )
```

```
name = "Alex"
“name: {0:10}X" . format( name )
```

```
'name:           X'
      1234567890
```


name = "Alex"
"name: {0:10}X" . format(name)
0th
'name: X'
1234567890
'name: Alex X'
1234567890

```
name = "Alex"
"name: {0:10}X" . format( name )



'name:           X'
      1234567890

'name: Alex      X'
      123456789
       ↓   ↓   ↓
'name: Alex      X'
```

We stopped here last time…

```
name1 = "Alex"
name2 = "Anastasia"
"names: {0:10}X{1:10}Y" . format( name1, name2 )
```

```
name1 = "Alex"
name2 = "Anastasia"
"names: {0:10}X{1:10}Y" . format( name1, name2 )
```

```
"names:           X           Y"
       1234567890 1234567890
```

```
name1 = "Alex"
name2 = "Anastasia"                          0       1
"names: {0:10}X{1:10}Y" . format( name1, name2 )


"names:           X           Y"
       1234567890 1234567890

"names: Alex      XAnastasia Y"
       1234567890 1234567890
```

```
name1 = "Alex"
name2 = "Anastasia"                          0       1
"names: {0:10}X{1:10}Y" . format( name1, name2 )


"names:           X           Y"
       1234567890 1234567890


"names: Alex      XAnastasia Y"
       1234567890 1234567890
       ↓  ↓  ↓
"names: Alex      XAnastasia Y"
```

```
name1 = "Alex"
name2 = "Anastasia"                          0       1
"names: {0:10}X{1:10}Y" . format( name1, name2 )


"names:                X                    Y"
        1234567890 1234567890


"names: Alex       XAnastasia Y"
        1234567890 1234567890
```

```
name1 = "Alex"
name2 = "Anastasia"                          0       1
"names: {0:>10}X{1:>10}Y". format( name1, name2 )


"names:              X              Y"
        1234567890 1234567890


"names:         AlexX AnastasiaY"
        1234567890 1234567890


"names:         AlexX AnastasiaY"
```

Bar Graph Exercise (revisited)

```
First name?  Dominique
Last name?   Thiebaut
Id?          990123456
Final grade? 90

+-----------------------------------------------+-------------+
|Dominique Thiebaut                             990123456    |
+-----------------------------------------------+-------------+
      00...10...20...30...40...50...60...70...80...90...100
grade: ##############################################
class: ##########################################
```

It works the same for integers!

```
name1 = "Alex"
age   = 22                                    0      1
"name: {0:10} age: {1:3}!". format( name1, age )


"name:            age:    !"
      1234567890      123

"name: Alex       age:  22!"
      1234567890      123

"name: Alex       age:  22!"
```

Ints are automatically right-aligned

Use '<' to left-align

```
name1 = "Alex"
age   = 22                                    0      1
"name: {0:10} age: {1:<3}!". format( name1, age )


"name:                age:     !"
      1234567890       123

"name: Alex           age: 22 !"
      1234567890       123
       ↓    ↓    ↓
"name: Alex           age: 22 !"
```

Floats are a bit different… We need to specify

- a total number of digits, and
- a number of digits after the decimal point.

```
Pi = 3.141592653589793

"Pi ={0:10.2f}#" . format( Pi )
```

```
"Pi =          #"
     1234567890
```

```
Pi = 3.141592653589793

"Pi ={0:10.2f}#" . format( Pi )


"Pi =           #"
     1234567890
               12
      ↓   ↓   ↓

"Pi =       3.14#"
     1234567890
               12
```

We can left- and right-align floats with '<' and '>' as well…

```
name1 = "Alex"
age1  = 12
coef1 = 6.188888
name2 = "Anastasia"
age2  = 22
coef2 = 8

print(  ??? .format( name1, age1, coef1 ) )
print(  ??? .format( name2, age2, coef2 ) )
```

```
name:       Alex age:  12 coefficient:   6.19/20
name:  Anastasia age:  22 coefficient:   8.00/20
      1234567890      123               123456
                                           12
```

Applications

- Printing temperatures
- Be flexible!
- Temporary variables are good!
- Print a conversion table of temperatures using the {…} formatting command.

Exercises

Figure out the Right Print Format

Exercise 1

```
for i in range( 10 ):
    print( i, 2**i )
```

```
0 1
1 2
2 4
3 8
4 16
5 32
6 64
7 128
8 256
9 512
```

```
for i in range( 10 ):
    print(                    )
```

```
0       1
1       2
2       4
3       8
4      16
5      32
6      64
7     128
8     256
9     512
```

1234567890123

Exercise 2

```
count = 1
for name in ["Doc", "Grumpy", "Happy",
             "Sleepy", "Dopey", "Bashful",
             "Sneezy" ]:
    print( count, name, len( name ) )
    count = count + 1
```

```
1 Doc 3
2 Grumpy 6
3 Happy 5
4 Sleepy 6
5 Dopey 5
6 Bashful 7
7 Sneezy 6
```

```
count = 1
for name in ["Doc", "Grumpy", "Happy",
             "Sleepy", "Dopey", "Bashful",
             "Sneezy" ]:
    print(                                  )
    count = count + 1
```

```
1: len(Doc    )= 3
2: len(Grumpy )= 6
3: len(Happy  )= 5
4: len(Sleepy )= 6
5: len(Dopey  )= 5
6: len(Bashful)= 7
7: len(Sneezy )= 6
```

Arithmetic operators and math functions

Printing numbers to look "nice"

Using a `main()` function

Accumulating results

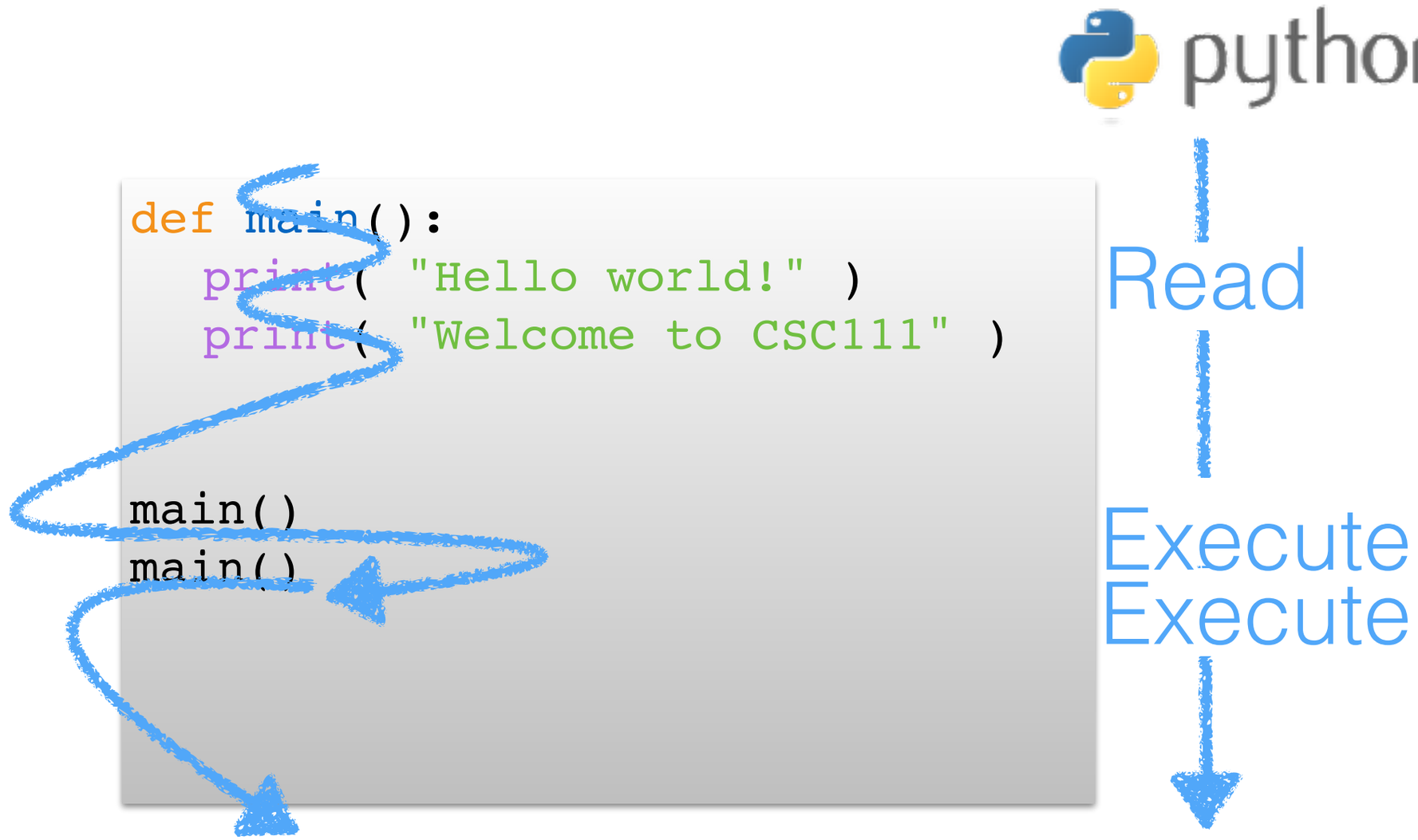
What are bits?

```
print( "Hello world!" )
print( "Welcome to CSC111!" )
```

```
def main():
    print( "Hello world!" )
    print( "Welcome to CSC111" )


main()
```

Hello world!
Welcome to CSC111


def main():
print("Hello world!")
print("Welcome to CSC111")
main()
main()
Read
Execute
Execute
Hello world!
Welcome to CSC111
Hello world!
Welcome to CSC111

Function Syntax

```
def <name> ( <parameters> ):
    <body>
```

Demo Tim

Python Shell

```
20
>>> c
30
>>> trio = a, b, c
>>> trio
(10, 20, 30)
>>> x, y, z = trio
>>> x
10
>>> y
20
>>> z
30
>>> i, j = trio
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    i, j = trio
ValueError: too many values to unpack
>>>
```

Ln: 26 Col: 4

Arithmetic operators and math functions

Printing numbers to look "nice"

Using a `main()` function

Accumulating results

What are bits?

Compute the Sum of Several Numbers.

```
ages = [12, 14, 10]

sumAge = ?
```

Compute the Sum of Several Numbers.

```
ages = [12, 14, 10]

sumAge = ?

# review printing all elements of a list
for age in [12, 14, 10]:
   print( age )
```

```
12
14
10
```

Compute the Sum of Several Numbers.

```
ages = [12, 14, 10]

sumAge = ?

# review printing all elements of a list
for age in [12, 14, 10]:
   print( age )
```

	0 <— sumAge
12	0+12 = 12 <— sumAge
14	12+14 = 26 <— sumAge
10	26+10 = 36 <— sumAge

<— Algorithm for sum

Compute the Sum of Several Numbers.

```
ages = [12, 14, 10]

sumAge = 0

# review printing all elements of a list
for age in [12, 14, 10]:
   #print( age )
   sumAge = sumAge + age                    <— new code
```

0	<— sumAge
0+12 = 12	<— sumAge
12+14 = 26	<— sumAge
26+10 = 36	<— sumAge

Compute the Sum of Several Numbers.

```
ages = [12, 14, 10]

sumAge = 0

# review printing all elements of a list
for age in [12, 14, 10]:
   #print( age )
   sumAge = sumAge + age

print( "sum = ", sumAge )
```

sum = 36

We stopped here last time…

代码写完了吗

Exercises

Compute is the sum of all the integers between 1 and 10

Compute is the sum of all the integers between 1 and 1000

Compute is the sum of all the multiples of 5 between 0 and 100, included

Exercises

Compute is the *average* of all the integers in [100, 95,100, 90, 60, 85]

Compute the total length of all the words in ["Doc", "Grumpy", "Happy", "Sleepy", "Dopey", "Bashful", "Sneezy"]

More Exercises

Compute is the factorial of 5
(factorial of 5 = 5 x 4 x 3 x 2 x 1)

Using the same approach, write a loop that *creates* a string of special characters, defined by a list of values:
Example:
[3, 2, 1, 5] would result in "###++#+++++"

[1,2,1,3,4] would result in "#++#+###++++"

Logistics

- Solution Programs appear after deadline at bottom of Homework page and Lab page
- Where are the TAs?
- Run & Evaluate
- No lab next week (Rally Day)

Arithmetic operators and math functions

Printing numbers to look "nice"

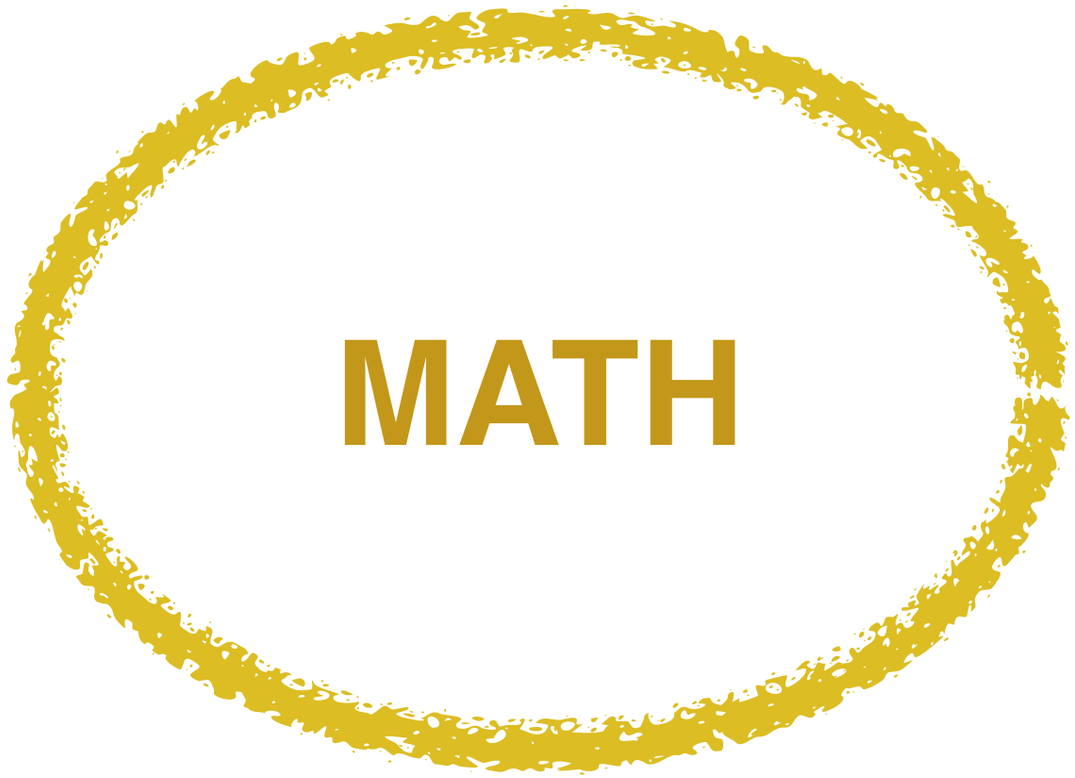
Using a `main()` function

Accumulating results

What are bits?

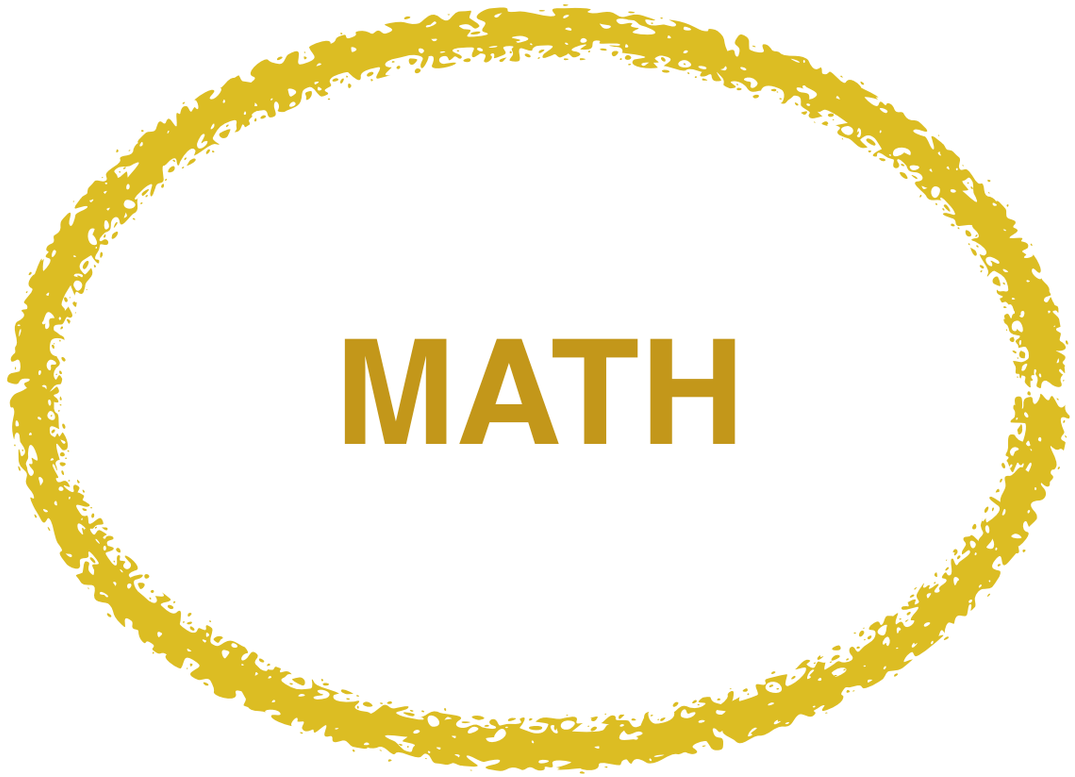
Number Systems

0
1
2
3
4
5
6
7
8
9
10



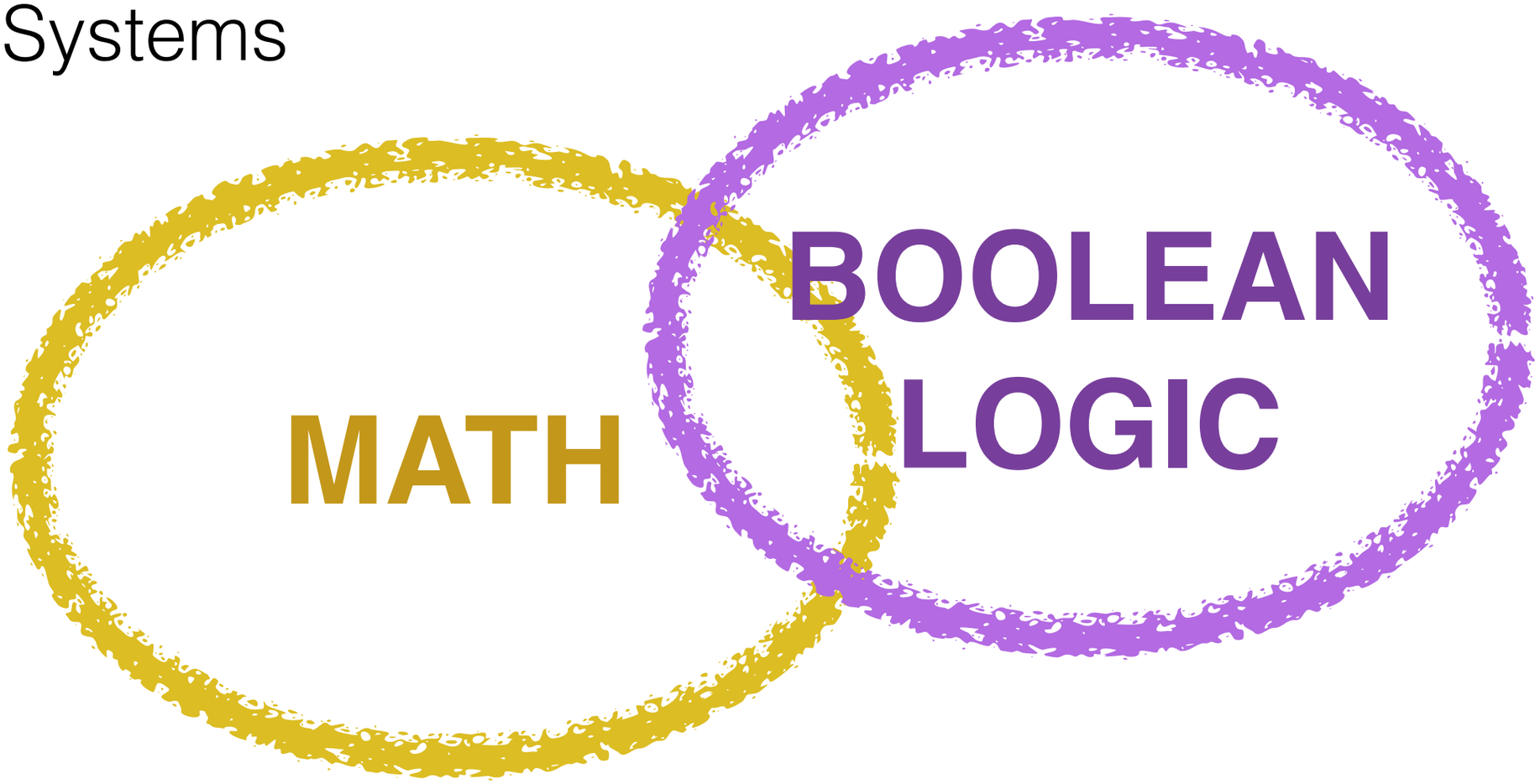
Number Systems

0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010



Number Systems

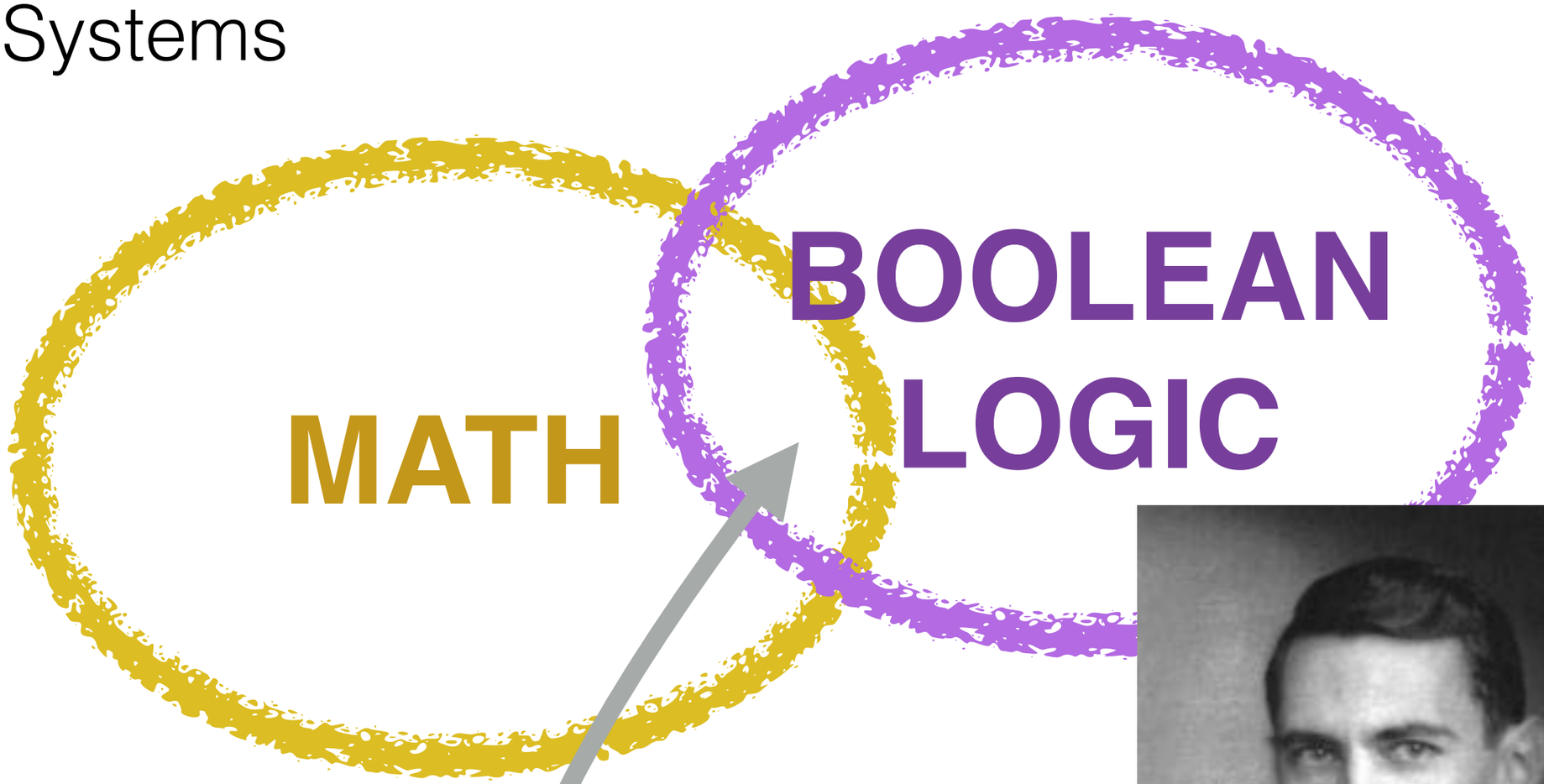
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010



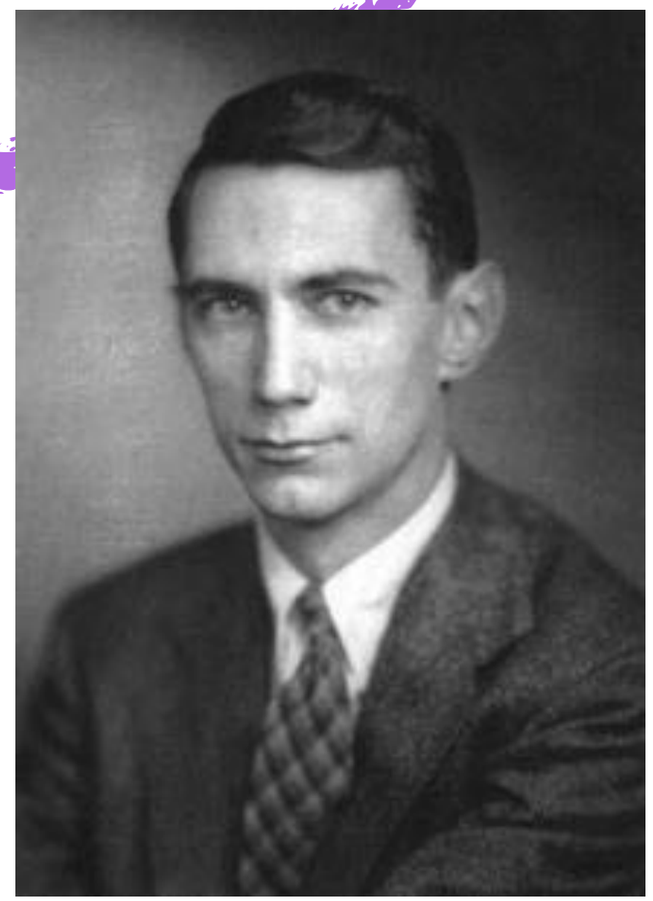
True
False
AND
OR
NOT

Number Systems

0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010



True
False
AND
OR
NOT



Claude Shannon, *1937*

```
    1  2  3
+   2  9  6
-------------
```

```
    1  2  3
+   2  9  6
-------------
          9
```

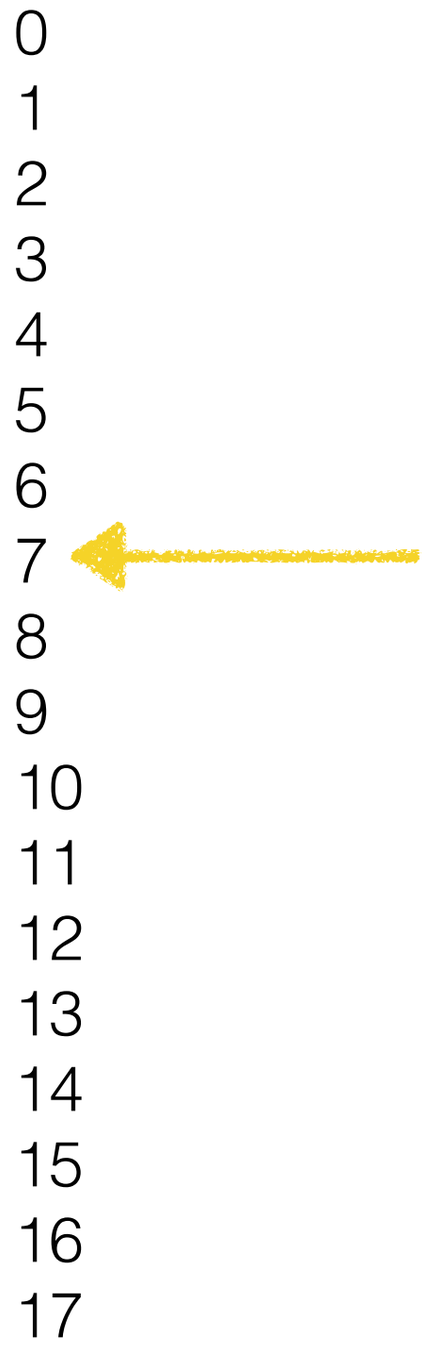
```
    1
    1  2  3
+   2  9  6
-------------
       1  9
```

```
    1
    1  2  3
+   2  9  6
-------------
    4  1  9
```

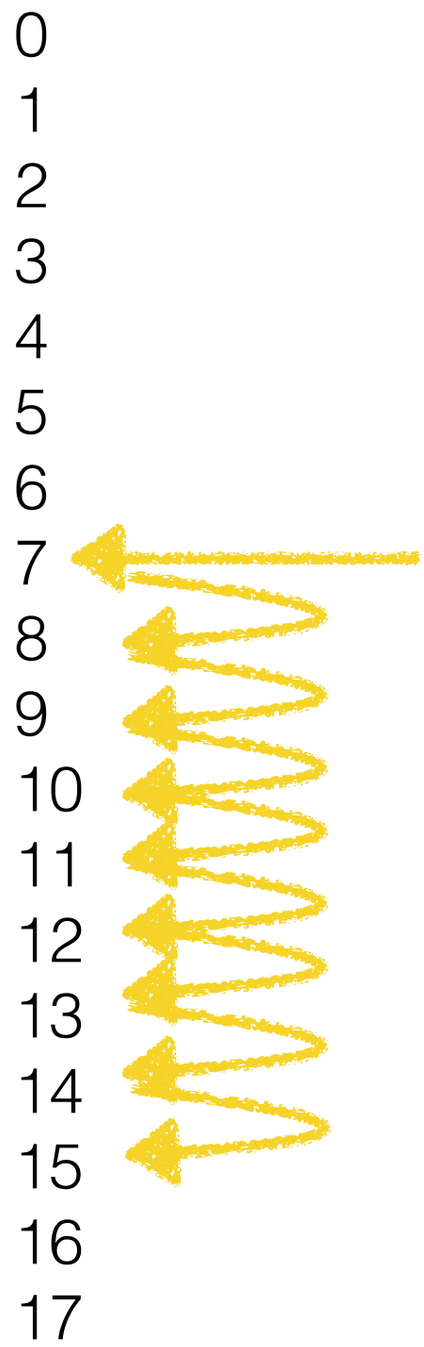
```
   1
     7
+    8
----------
```

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17

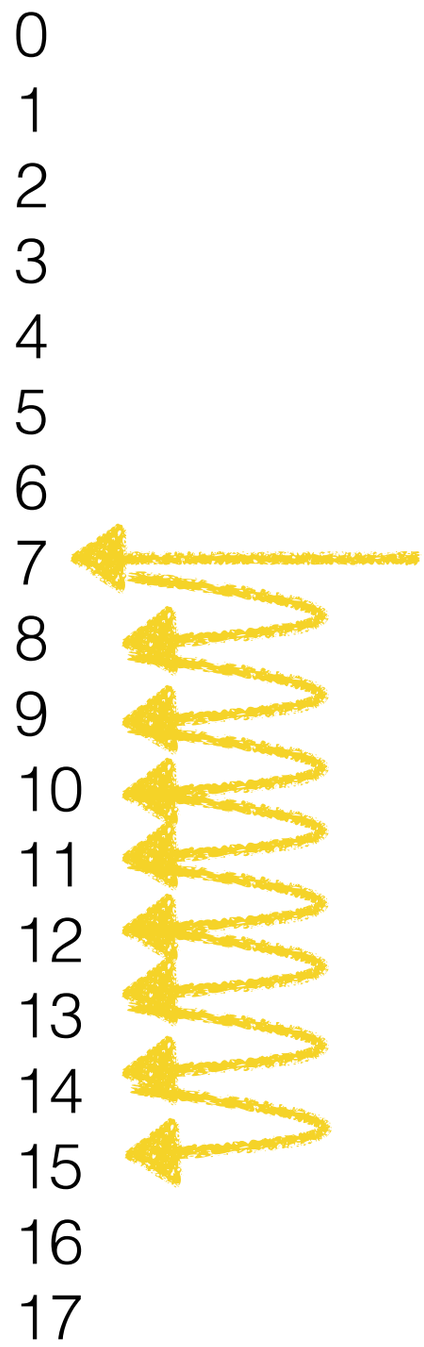
$$\begin{array}{r} \ 7 \\ +\ \ 8 \\ \hline \end{array}$$



```
    7
+   8
-------
```



```
   1
     7
+    8
------
     5
```



```
   1
   1  2  3
+  2  9  6
-------------
   4  1  9
```

```
      0  1  1
+     1  1  0
---------------
    1  0  0
```

```
0
1
10
11
100
```

```
   1
   1  2  3
+  2  9  6
-------------
   4  1  9
```

```
     0  1  1
+    1  1  0
-------------
        1
```

```
   0
   1
  10
  11
 100
```

```
   1
   1  2  3
+  2  9  6
-------------
   4  1  9
```

```
     1
     0  1  1
+    1  1  0
-------------
     1  0  1
```

```
   1
   0  1  1
+  1  1  0
-------------
   1  0  0
```

```
   0
   1
  10
  11
 100
```

```
   1
   1 2 3
+  2 9 6
---------
   4 1 9
```

```
 1  1
    0 1 1
+   1 1 0
---------
    0 0 1
```

```
   0
   1
  10
  11
 100
```

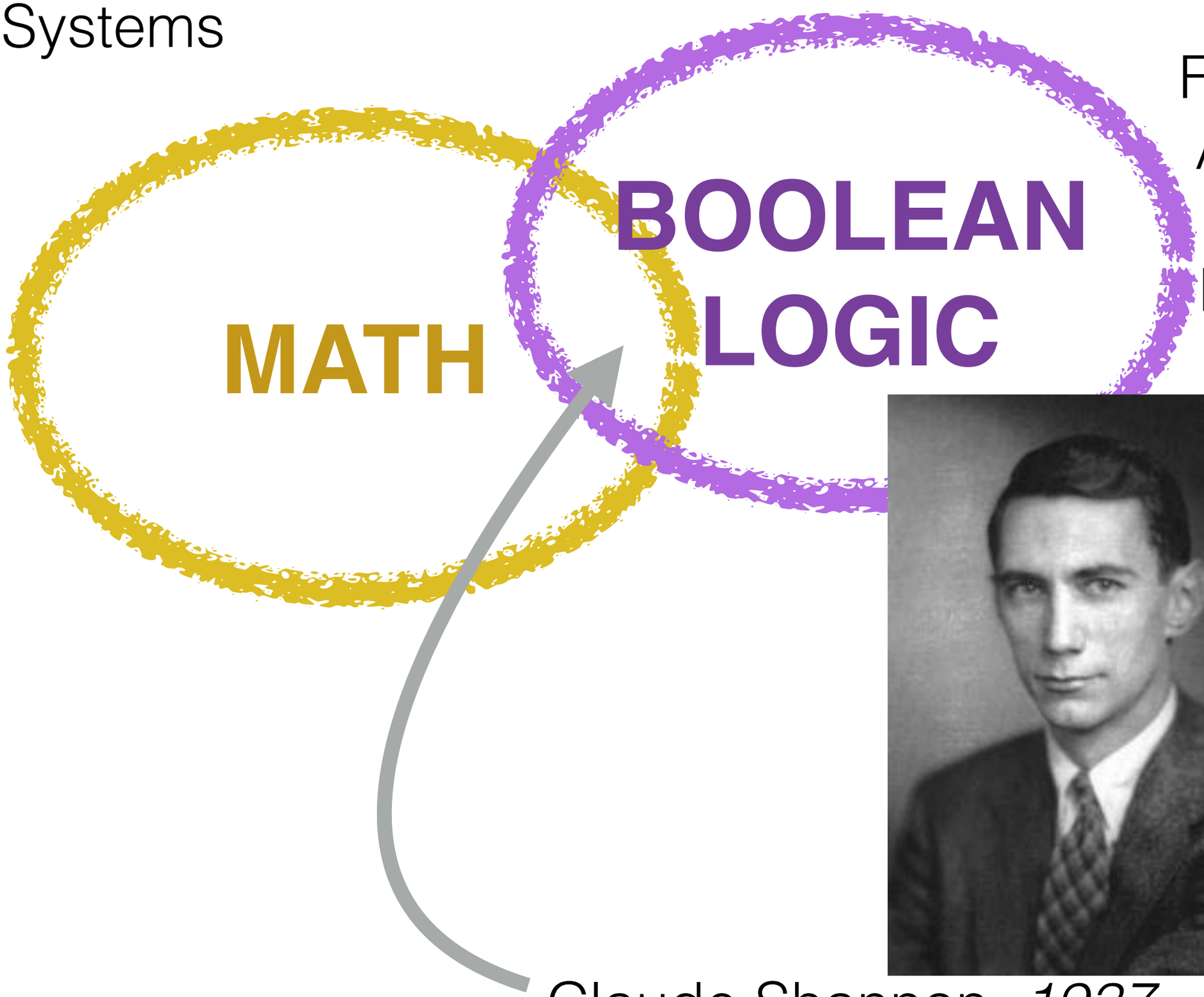
```
   1
   1 2 3
+  2 9 6
---------
   4 1 9
```

```
 1  1
    0 1 1
+   1 1 0
-----------
  1 0 0 1
```

```
   0
   1
  10
  11
 100
```

Number Systems

0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010



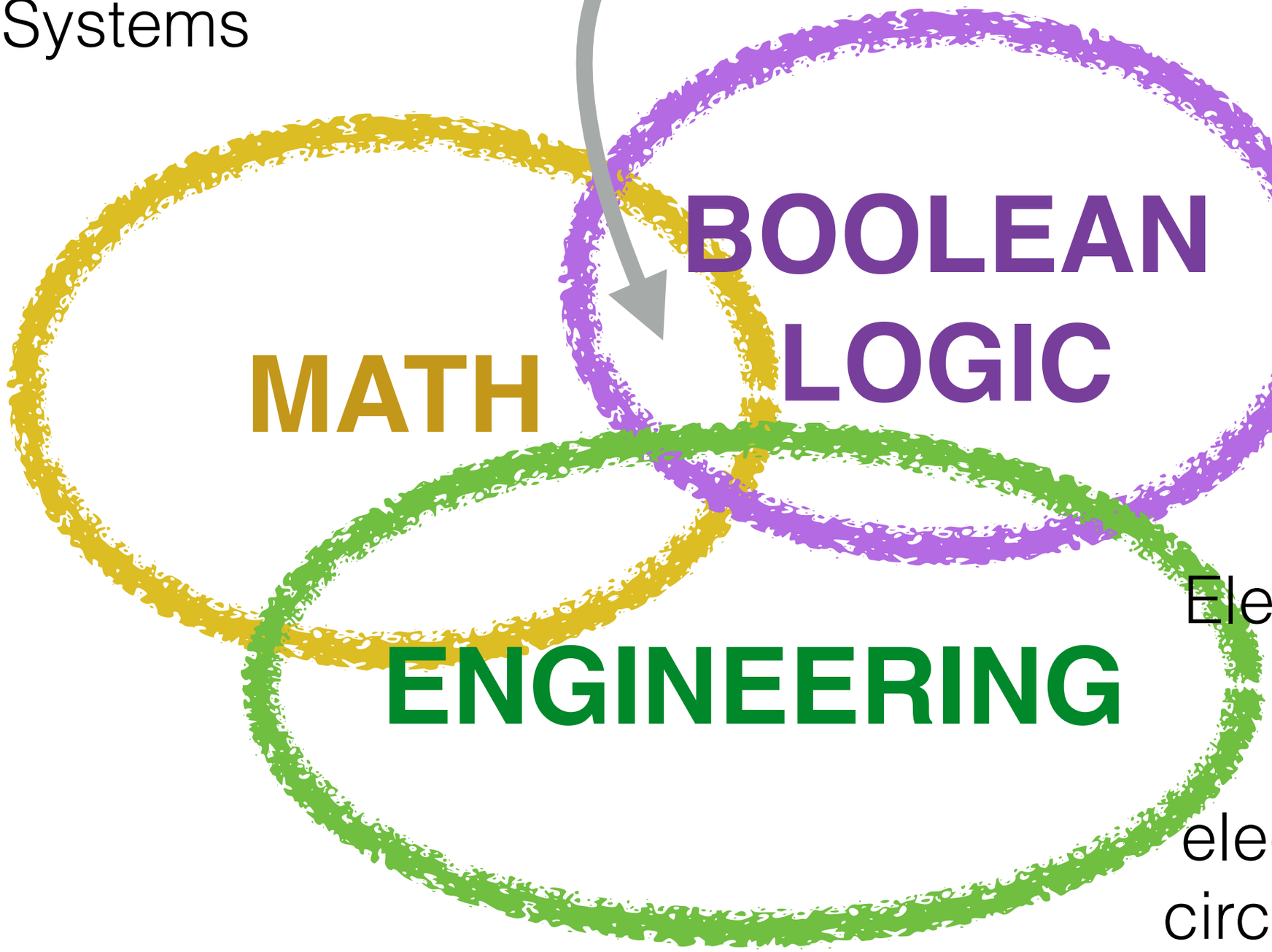
True
False
AND
OR
NOT

Claude Shannon, *1937*

Claude Shannon

Number Systems

0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010



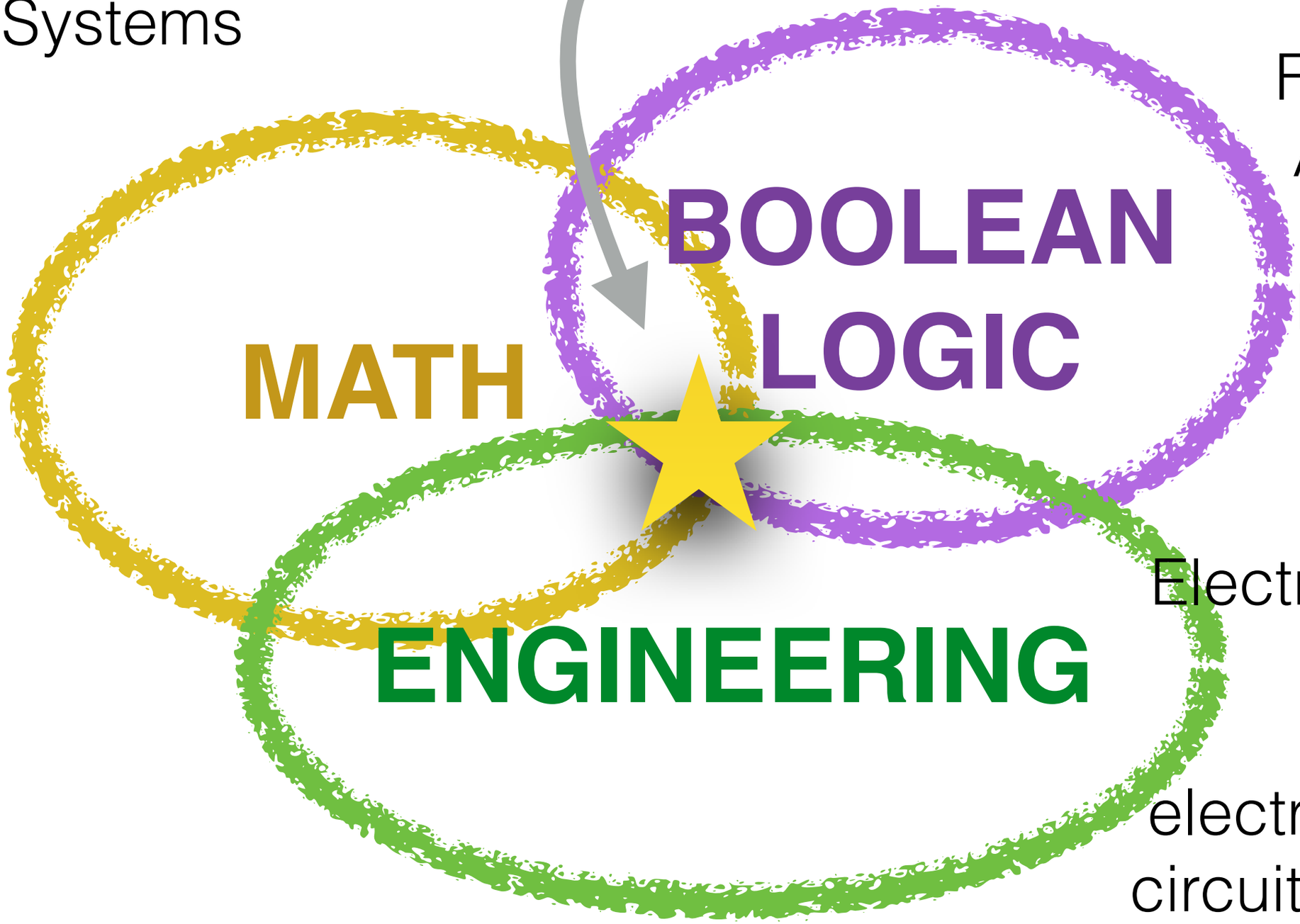
True
False
AND
OR
NOT

Electricity
ON
OFF
electronic circuits for
AND, OR, NOT

Claude Shannon

Number Systems

0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010



True
False
AND
OR
NOT

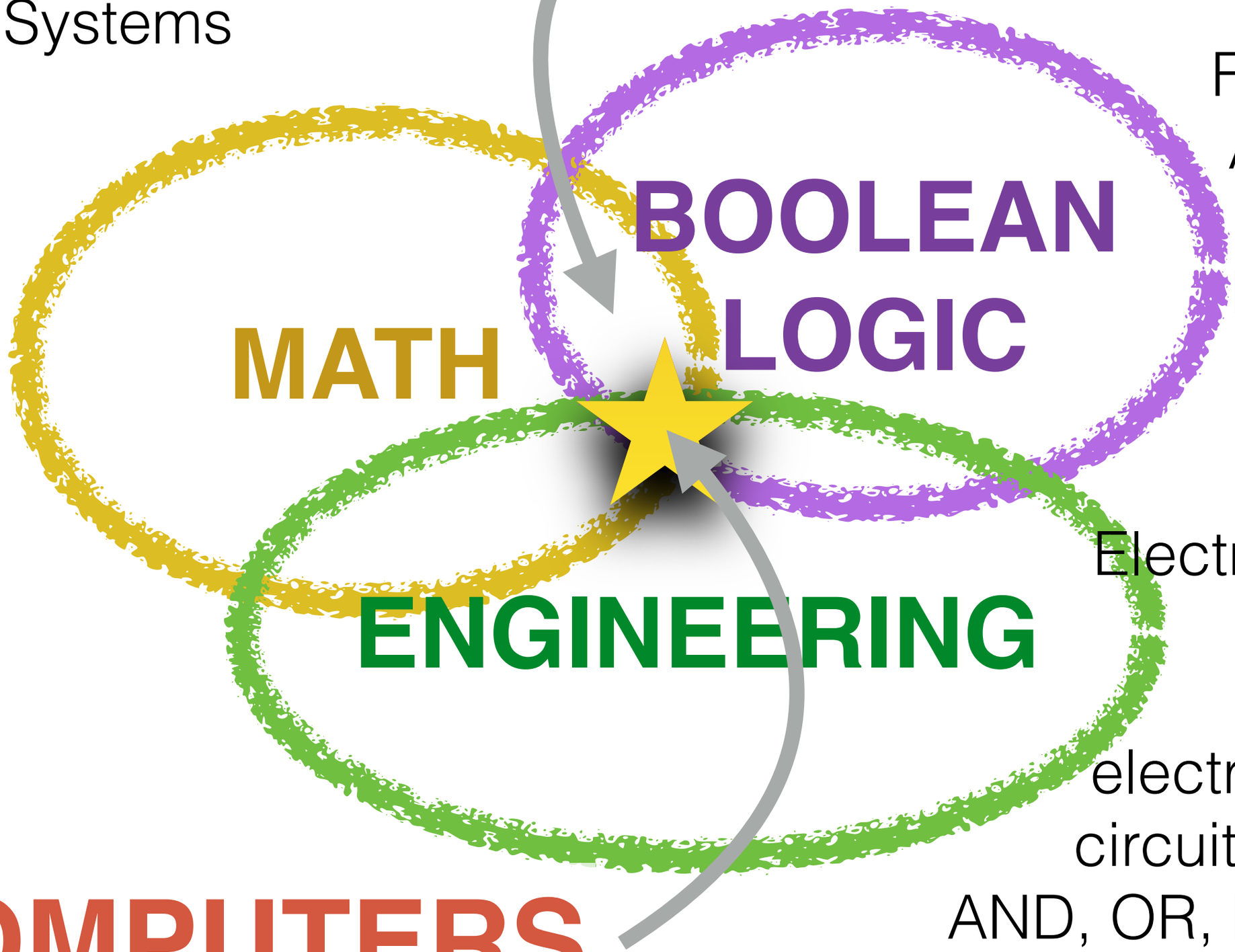
Electricity
ON
OFF
electronic circuits for
AND, OR, NOT

Claude Shannon

Number Systems

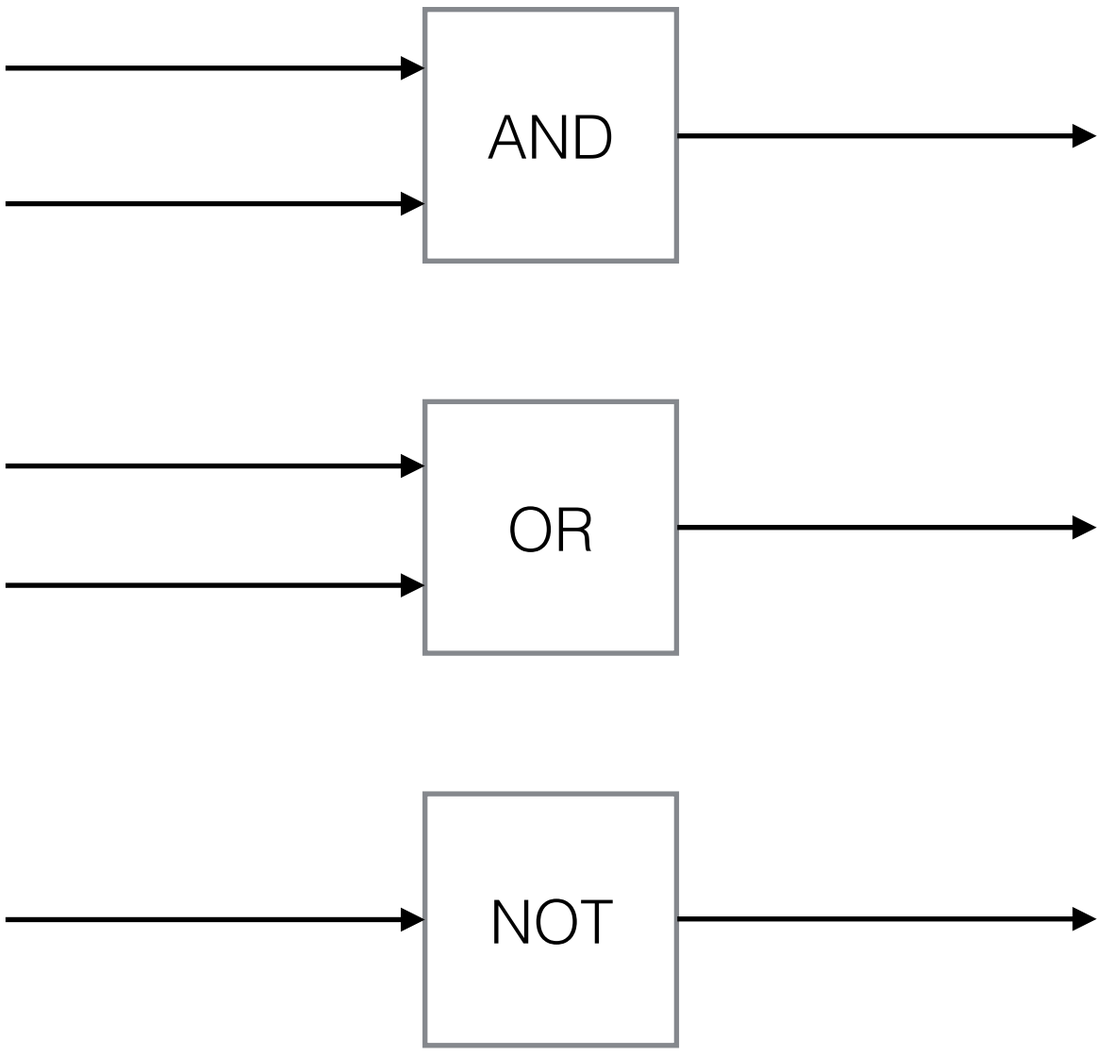
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010

True
False
AND
OR
NOT



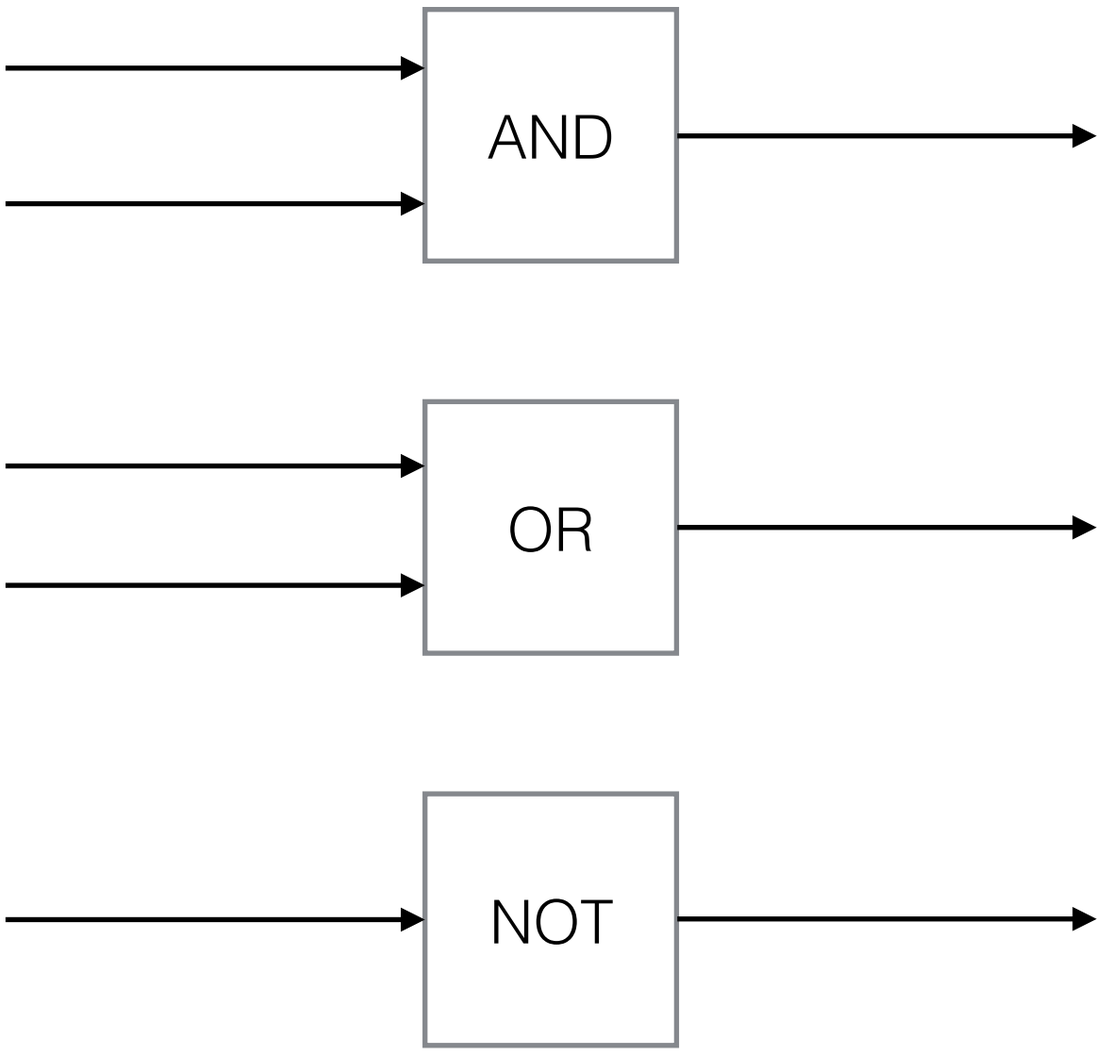
Electricity
ON
OFF
electronic
circuits for
AND, OR, NOT

Boolean Operators



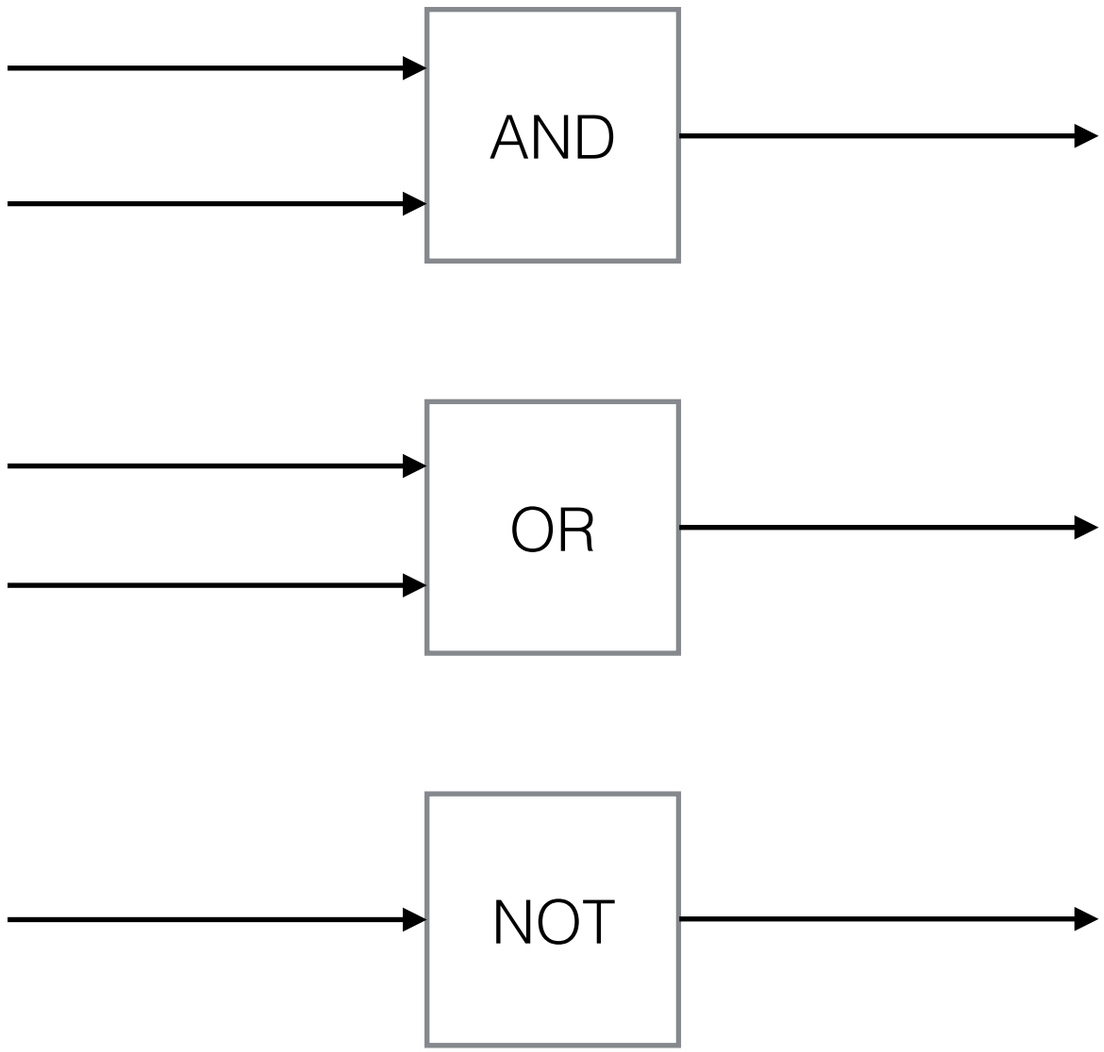
a	b	a and b

Boolean Operators



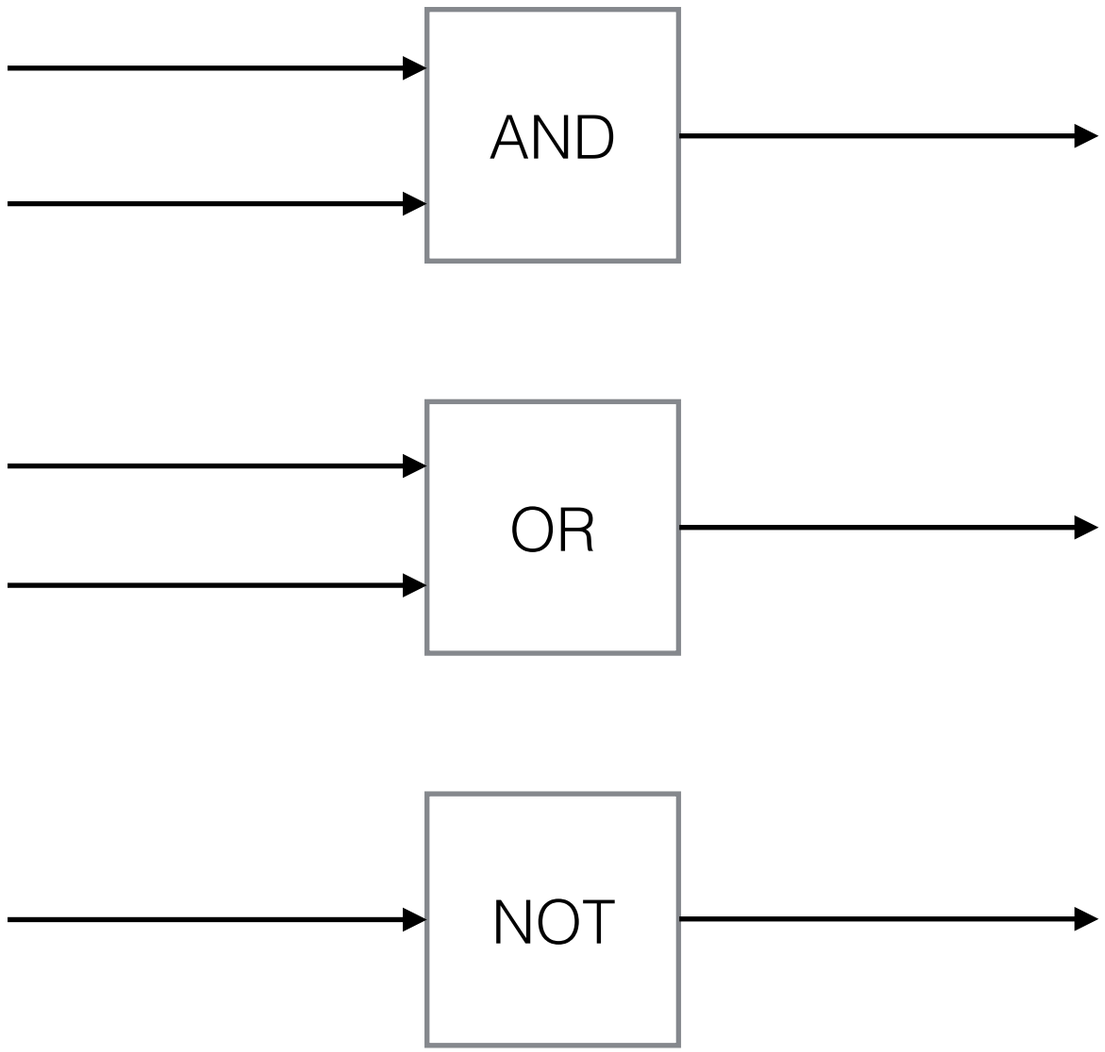
a	b	a and b
False	False	False

Boolean Operators



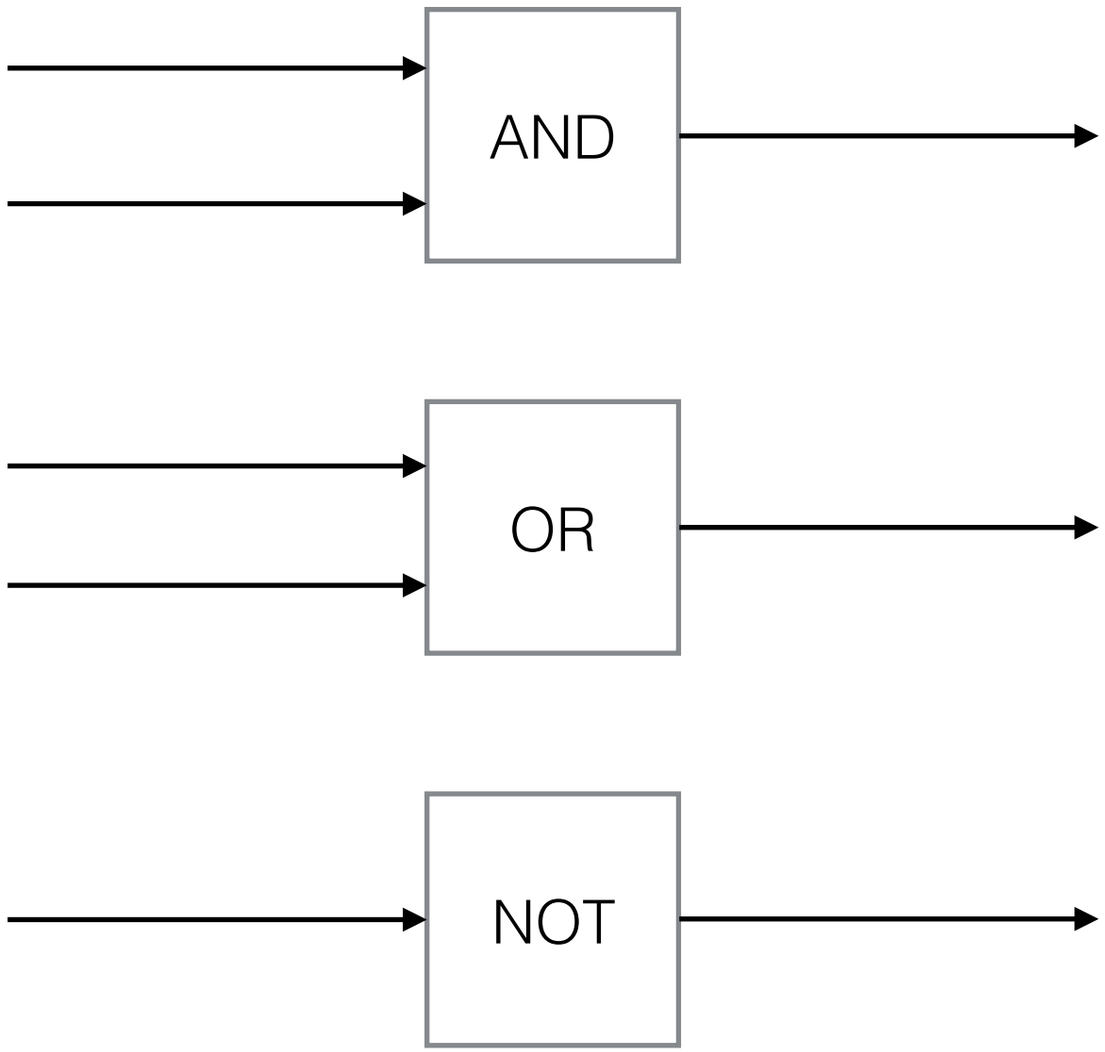
a	b	a and b
False	False	False
False	True	False

Boolean Operators

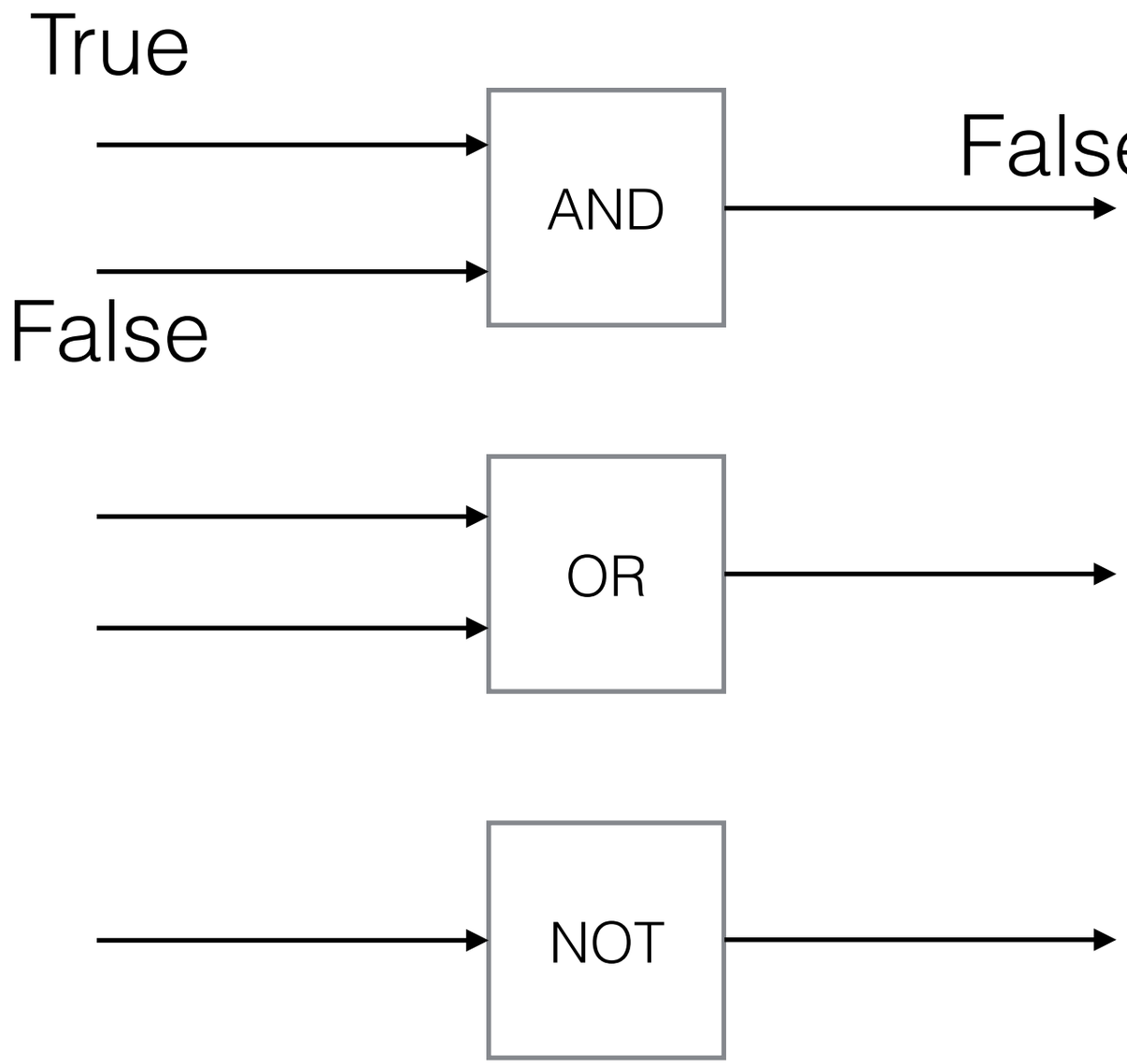


a	b	a and b
False	False	False
False	True	False
True	False	False

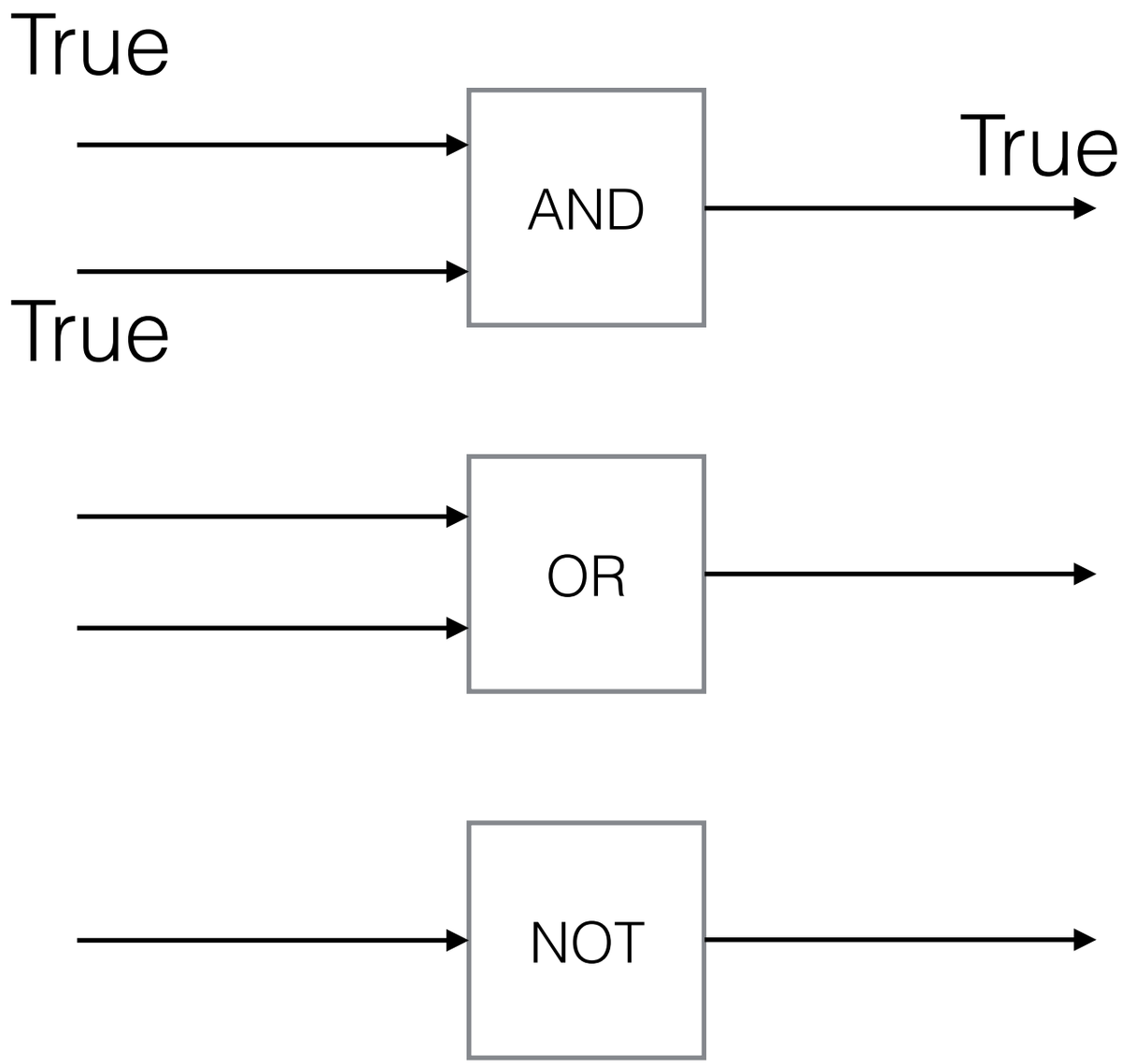
Boolean Operators



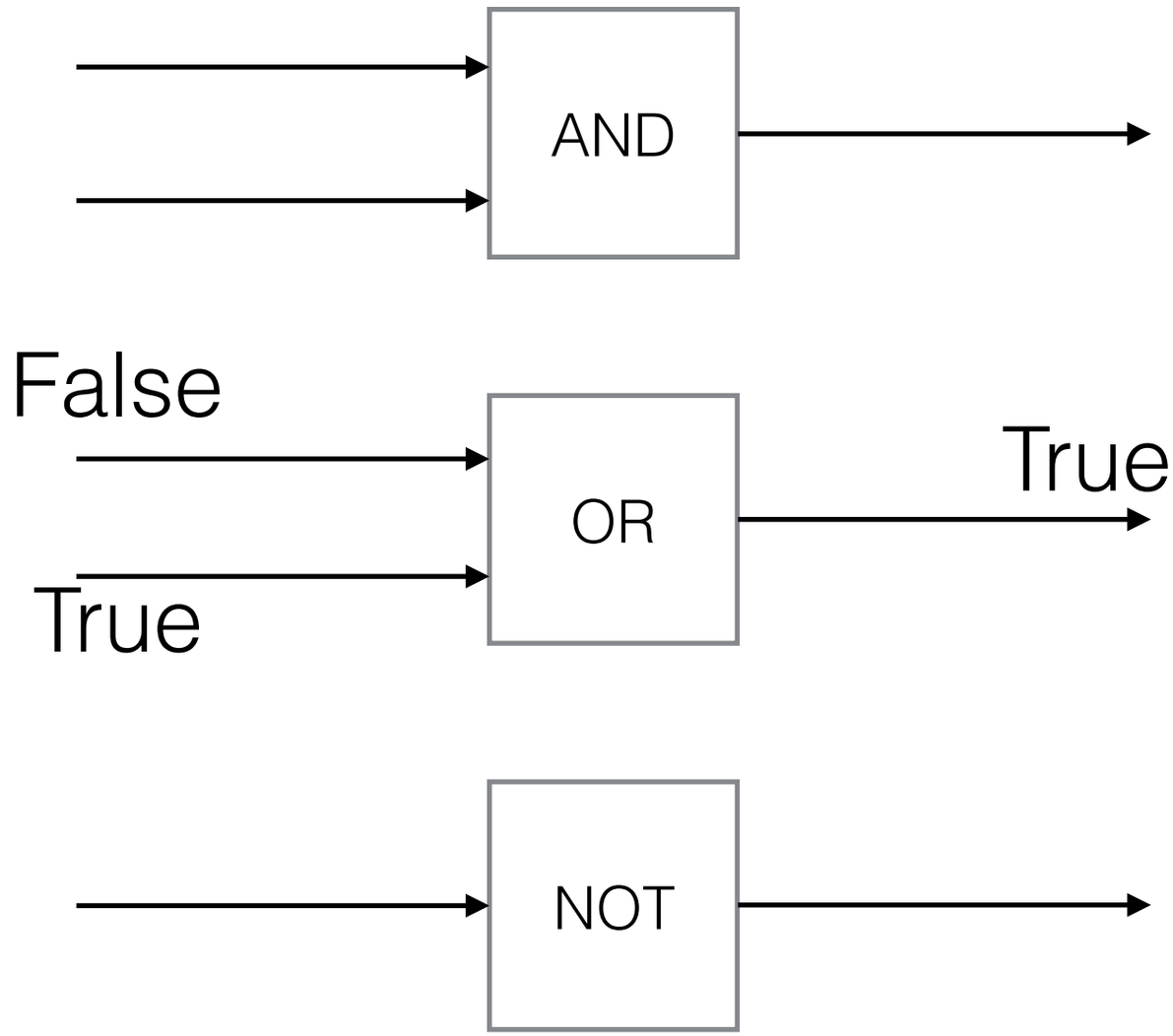
a	b	a and b
False	False	False
False	True	False
True	False	False
True	True	True



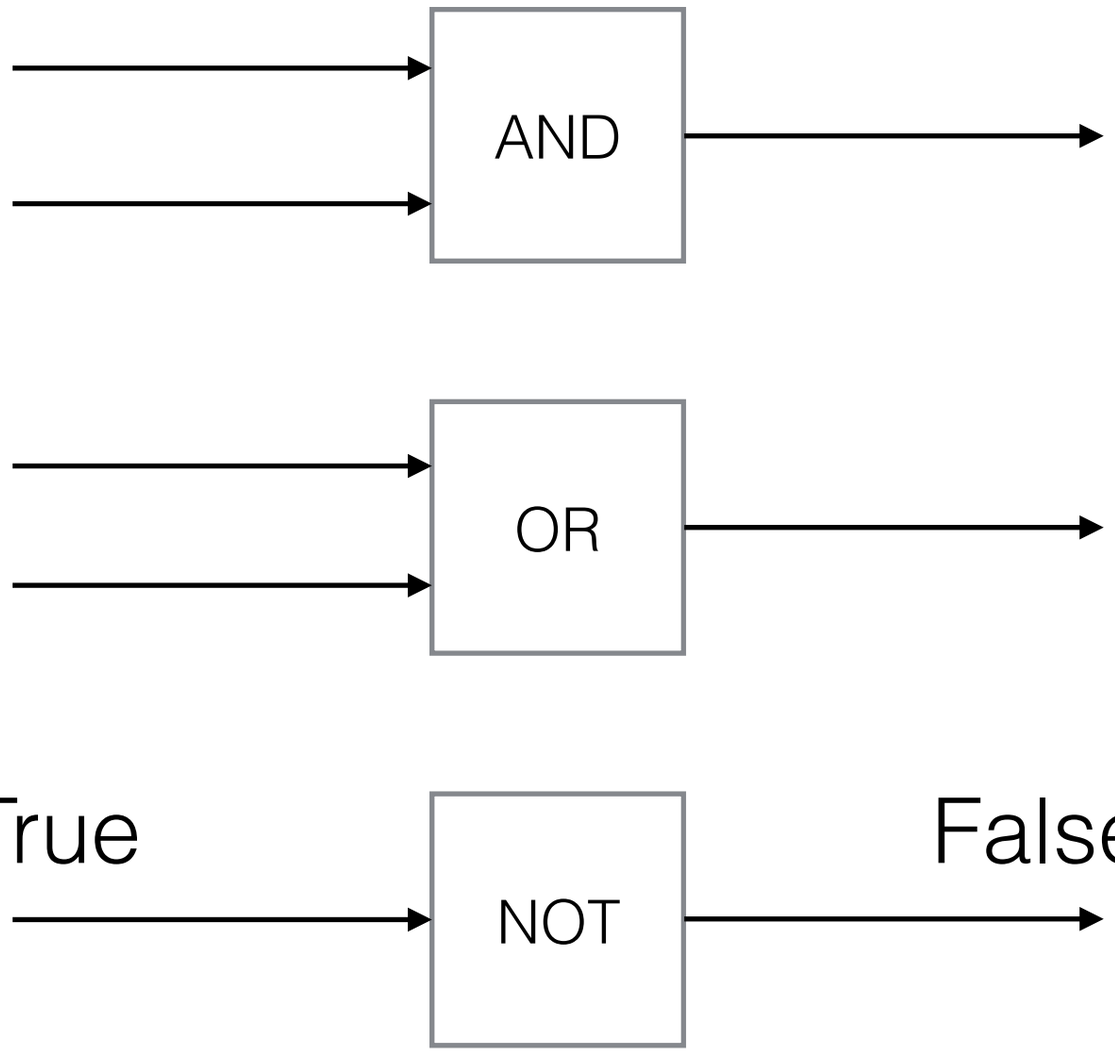
a	b	a and b
False	False	False
False	True	False
True	False	False
True	True	True



a	b	a and b
False	False	False
False	True	False
True	False	False
True	True	True

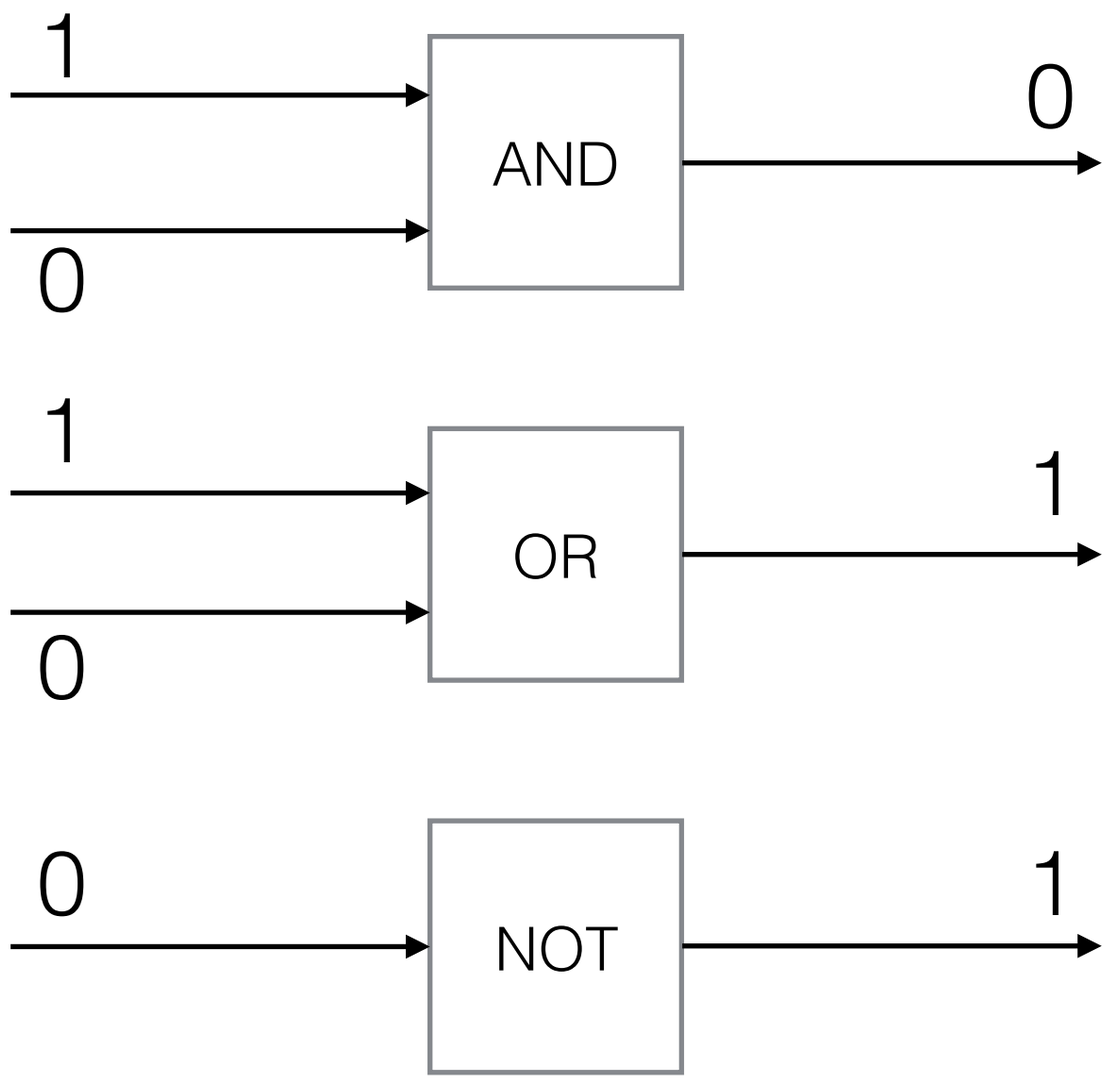


a	b	a **or** b
False	False	False
False	True	True
True	False	True
True	True	True



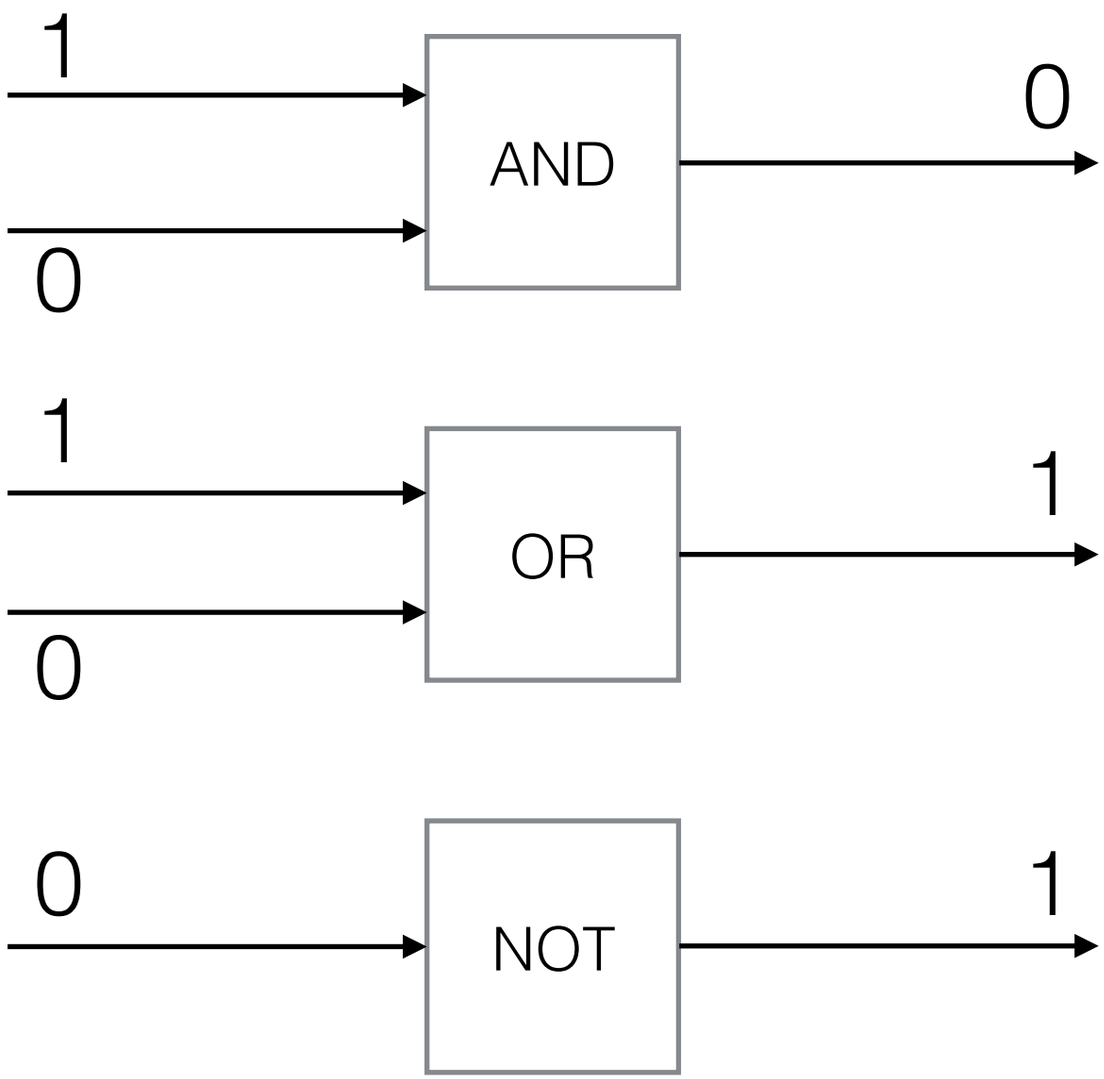
a	not a
False	True
True	False

True —-> 1
False —-> 0

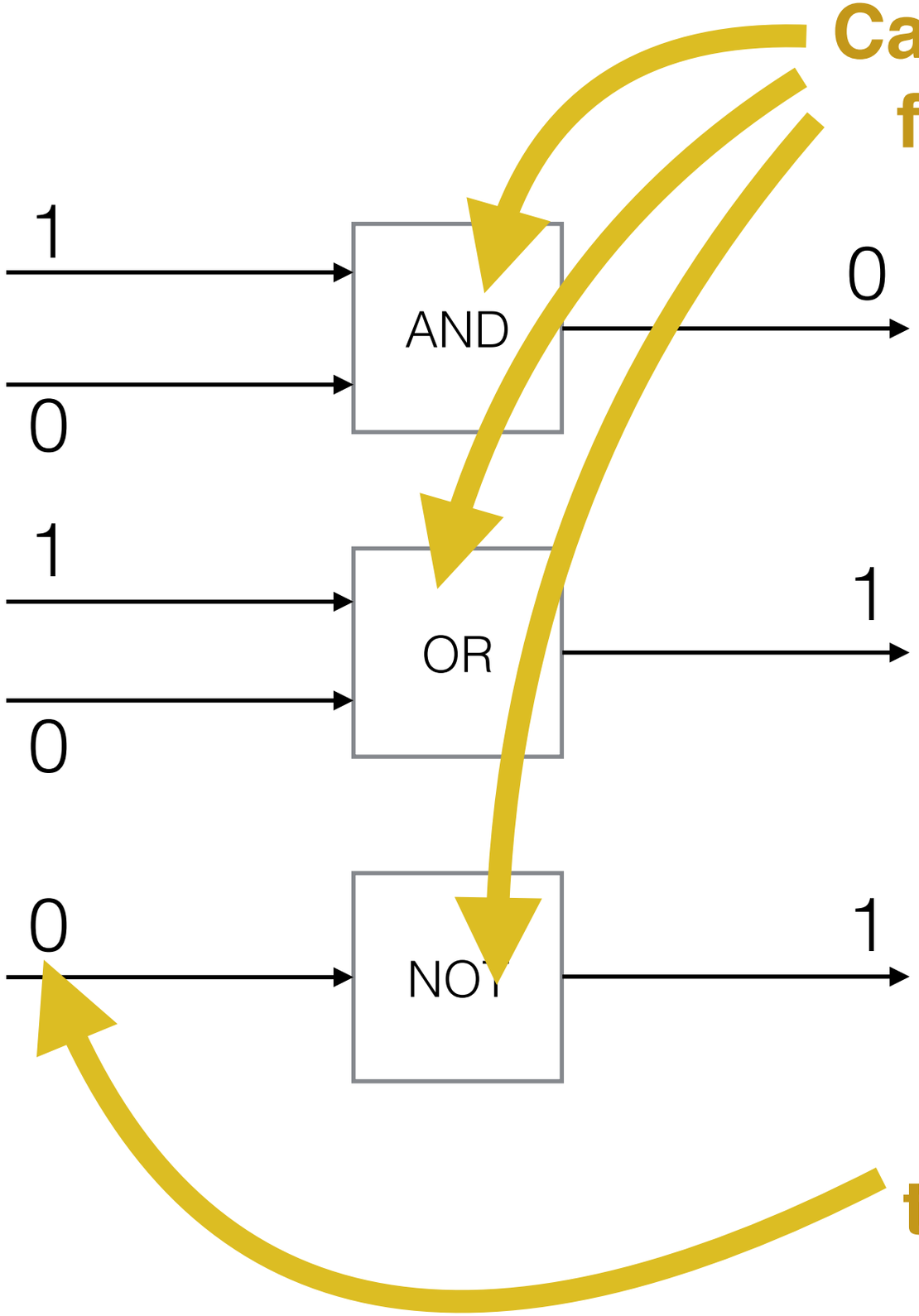


True —-> 1

False —-> 0

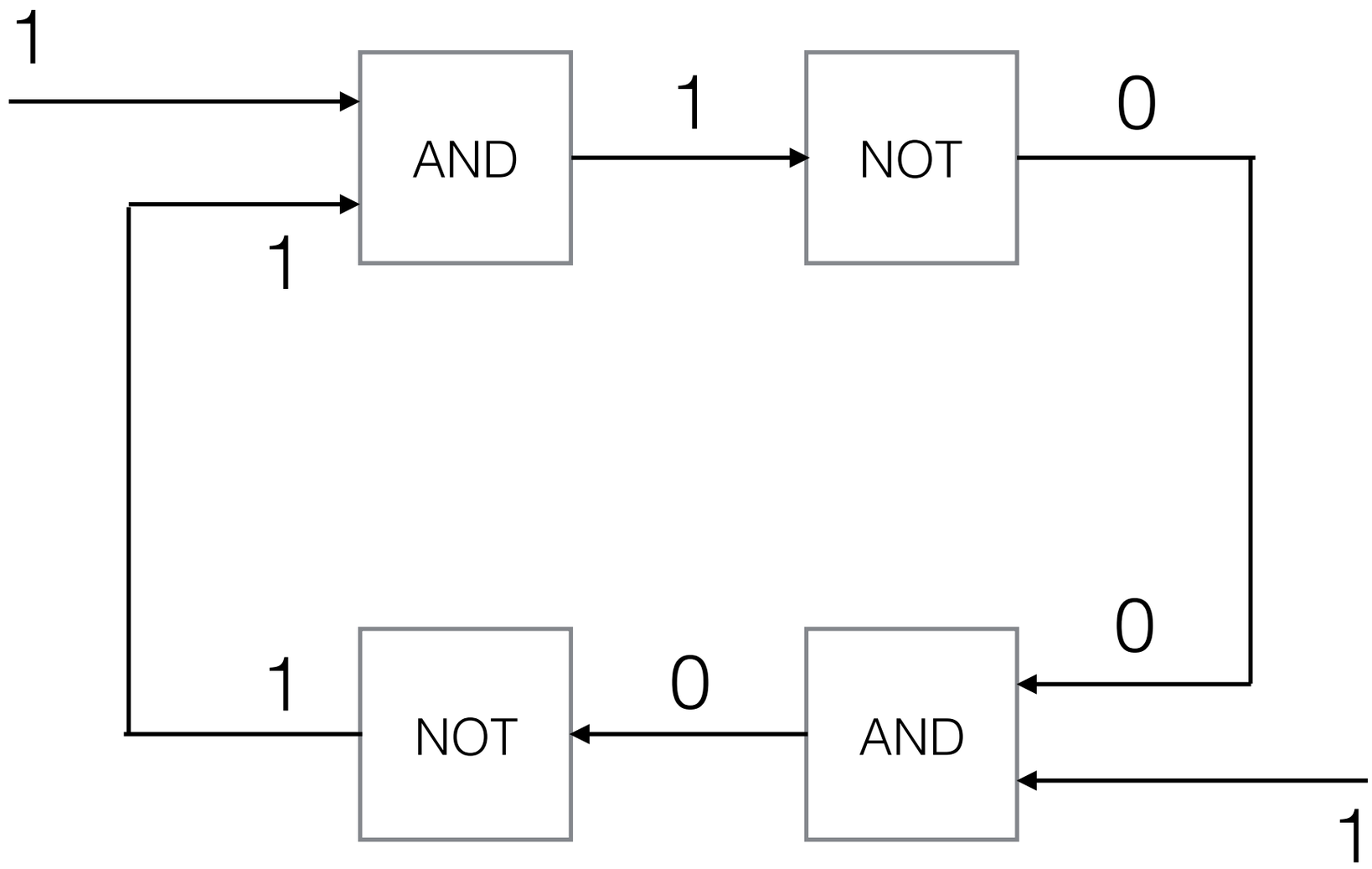


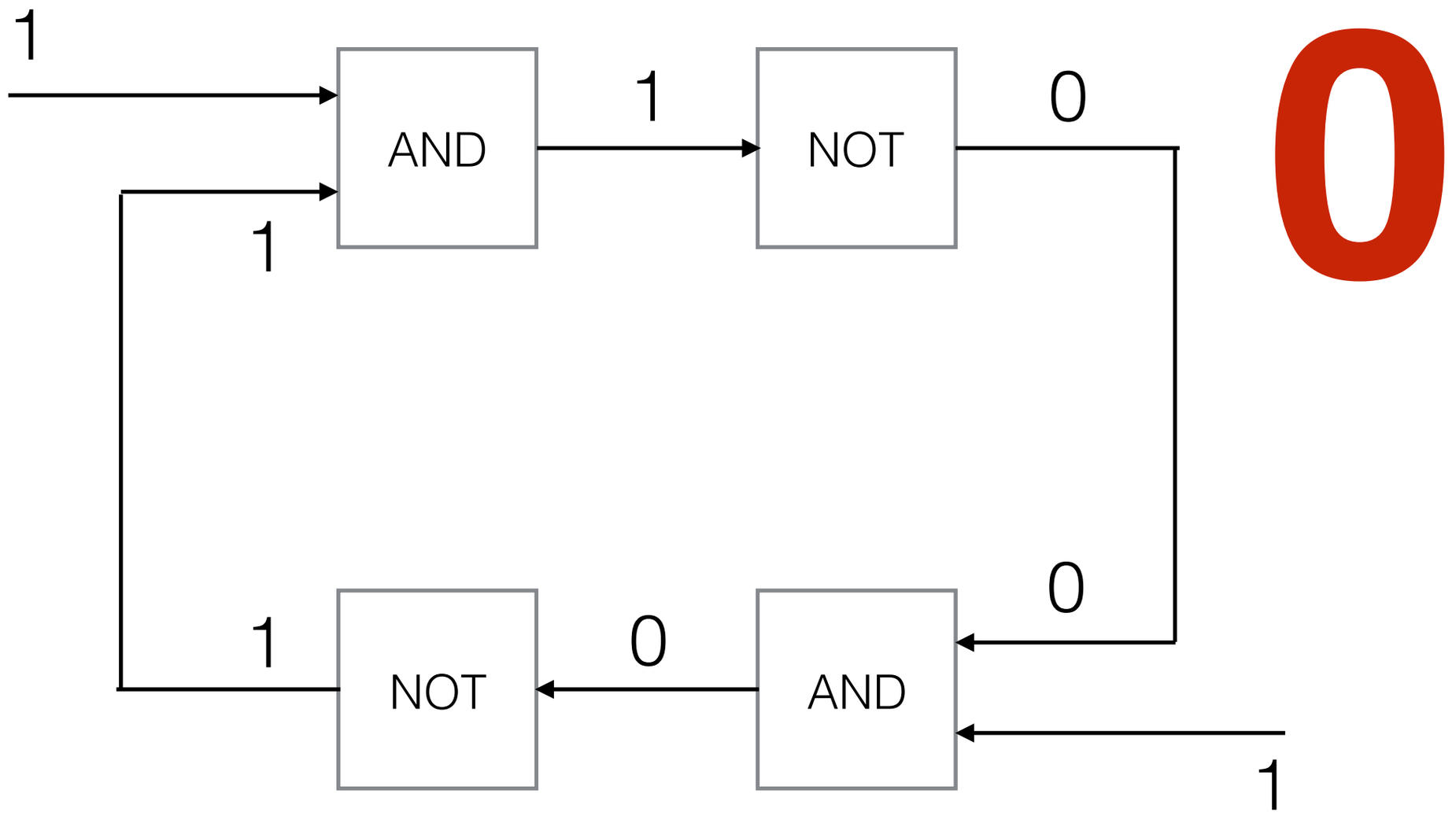
a	b	a **or** b
0	0	0
0	1	0
1	0	0
1	1	1

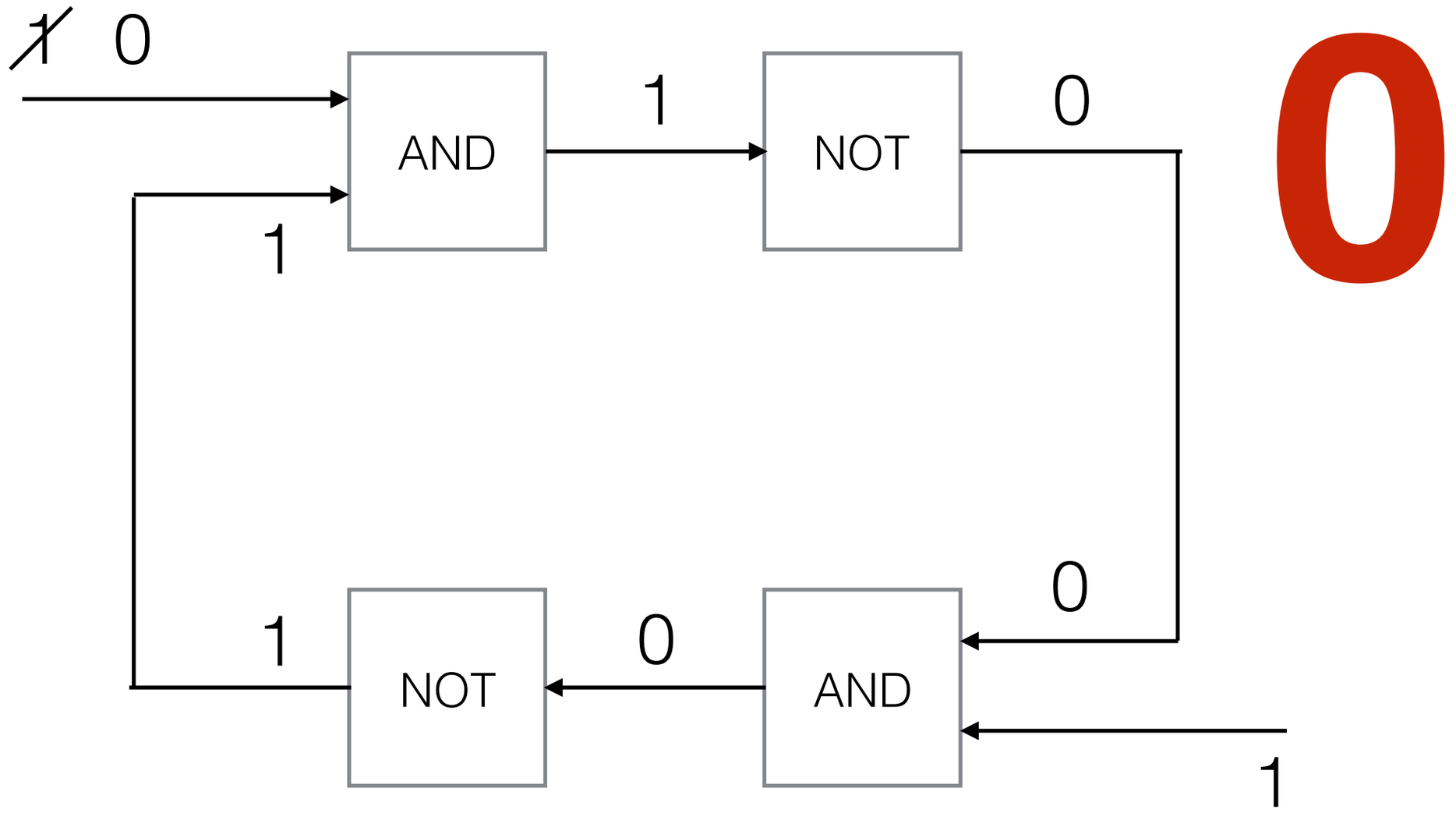


Can be built with a few transistors

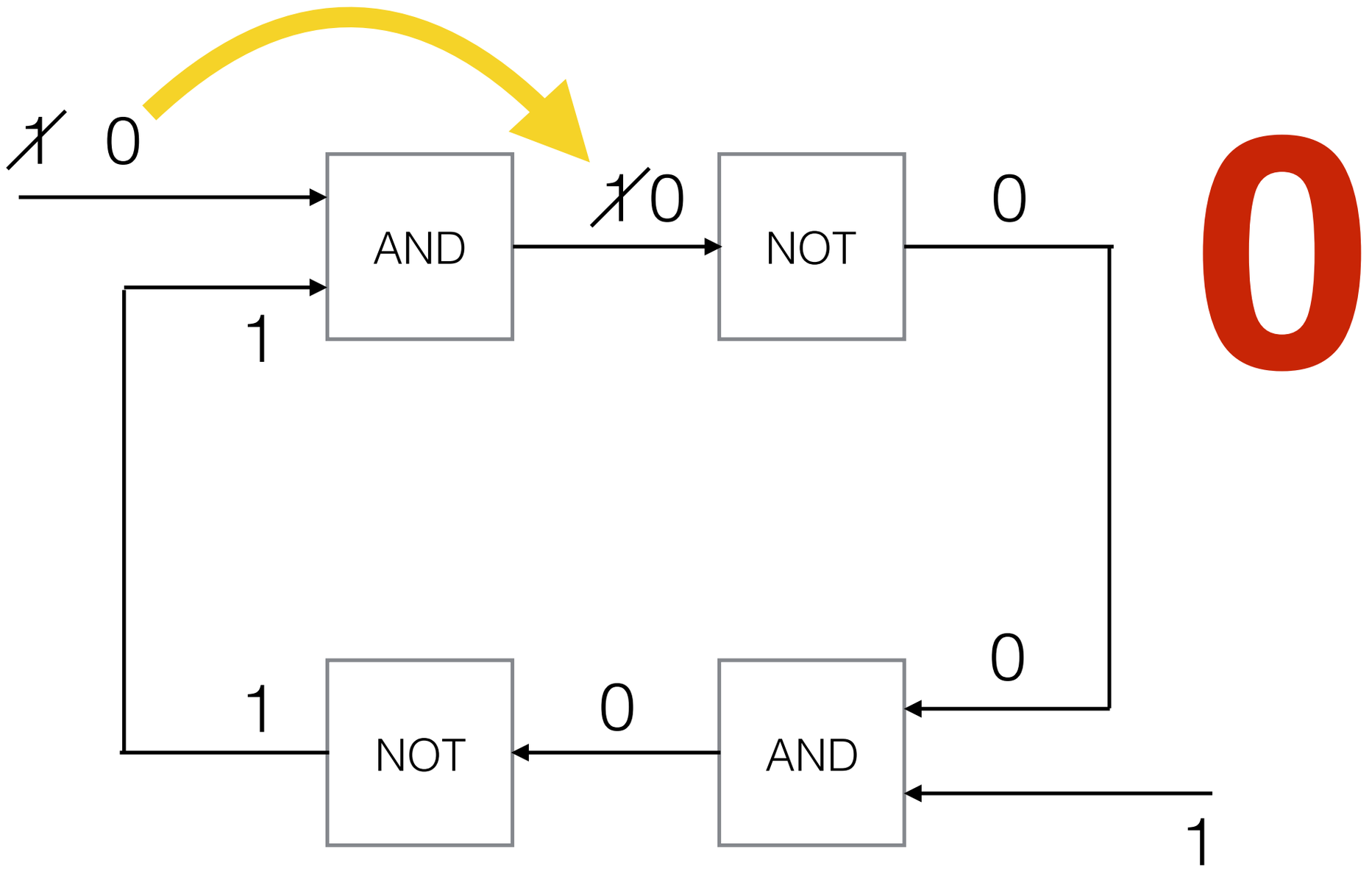
Can use electricity to represent 0 and 1


1
AND
1
NOT
0
1
1
NOT
0
AND
0
1

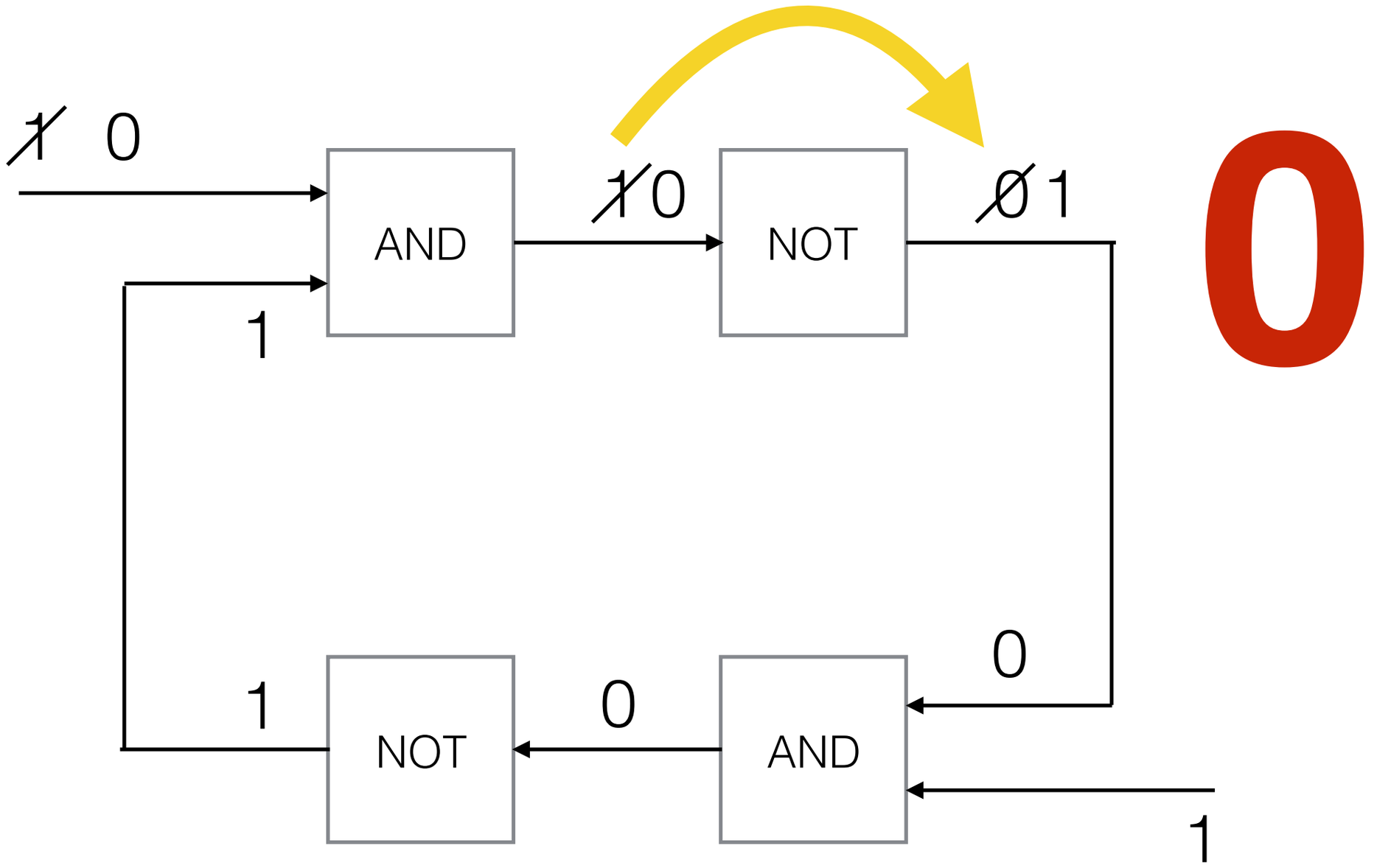

1
AND
1
NOT
0
0
1
NOT
0
AND
1
1
0



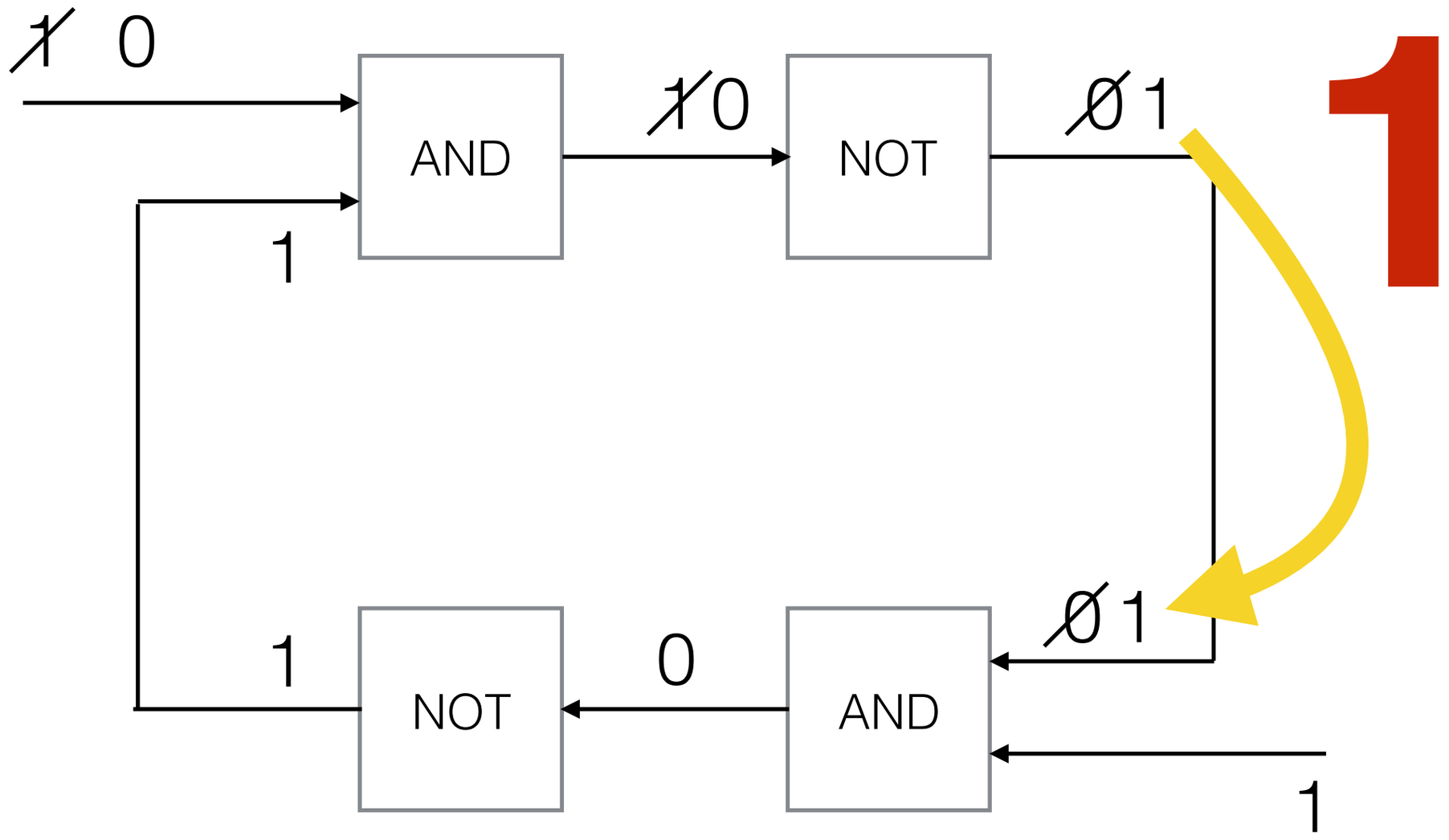
0

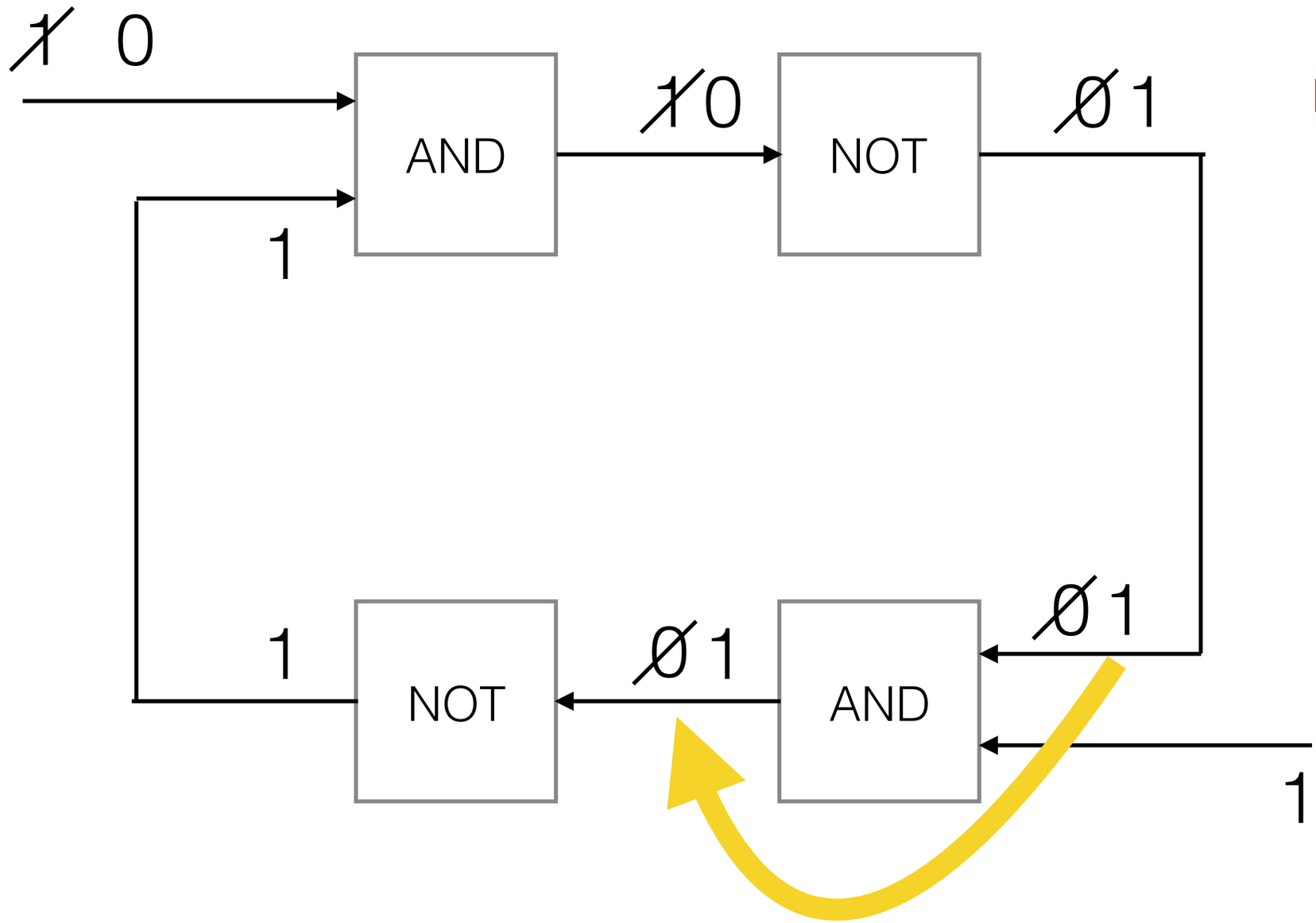


0

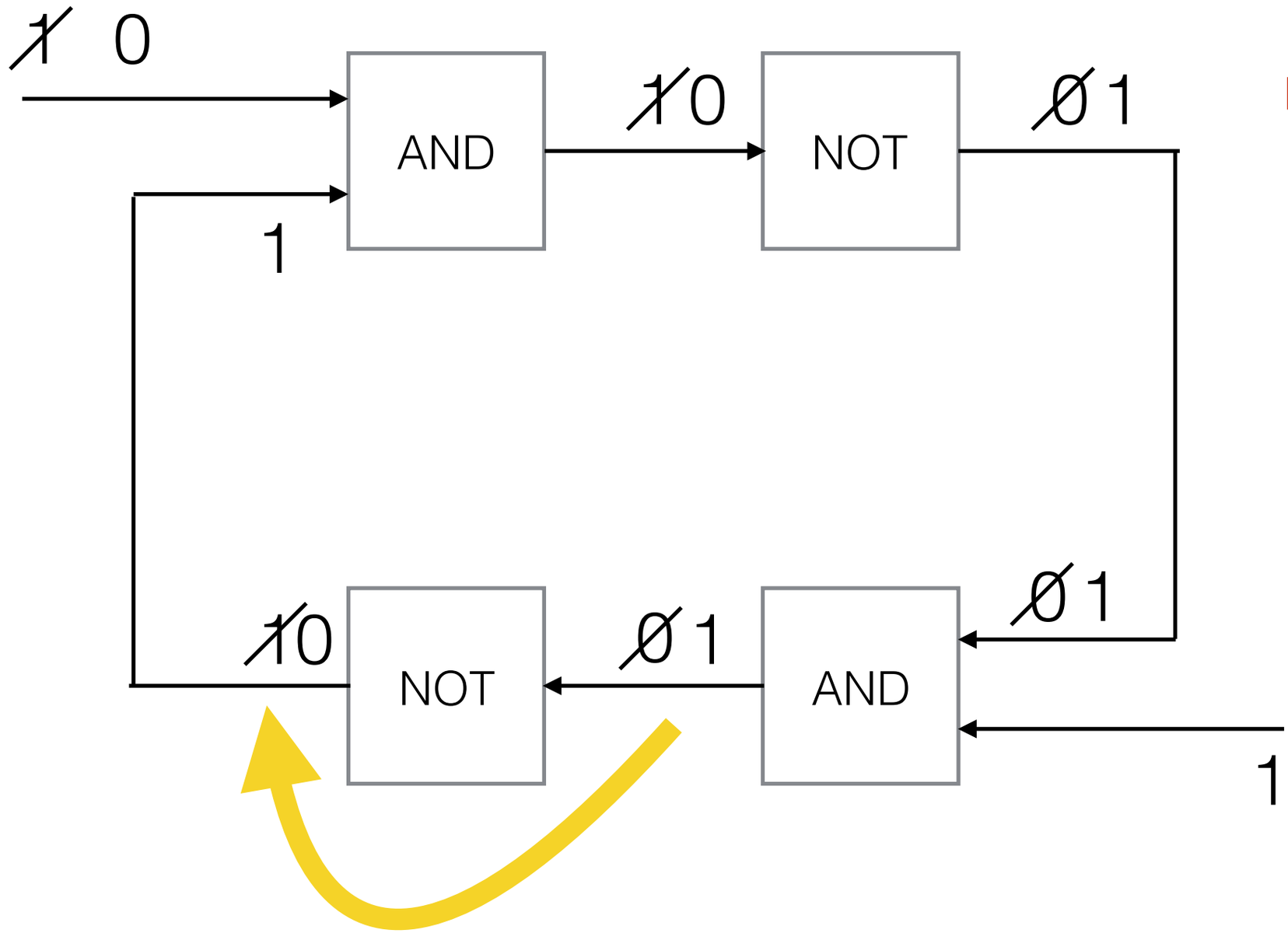


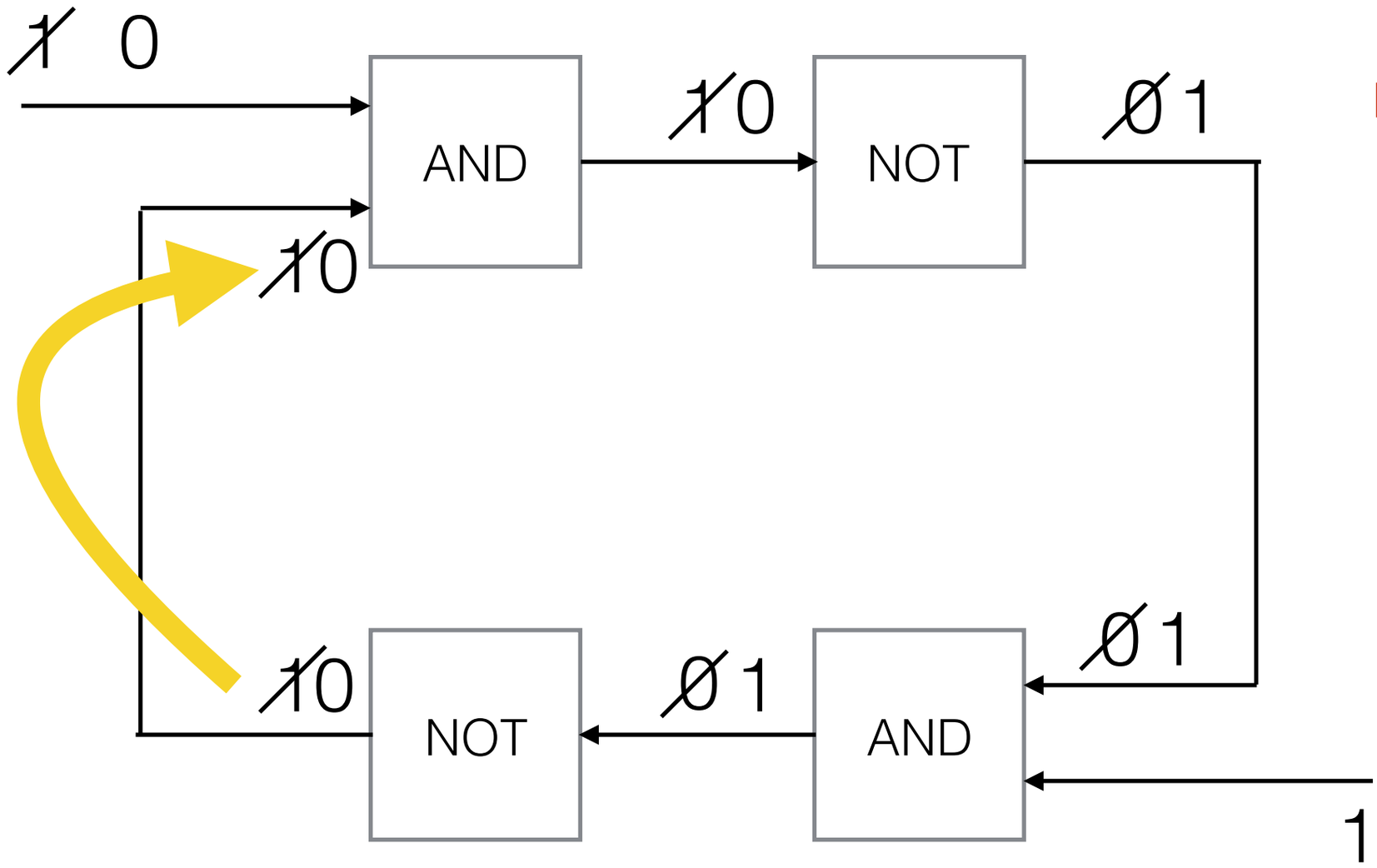
0

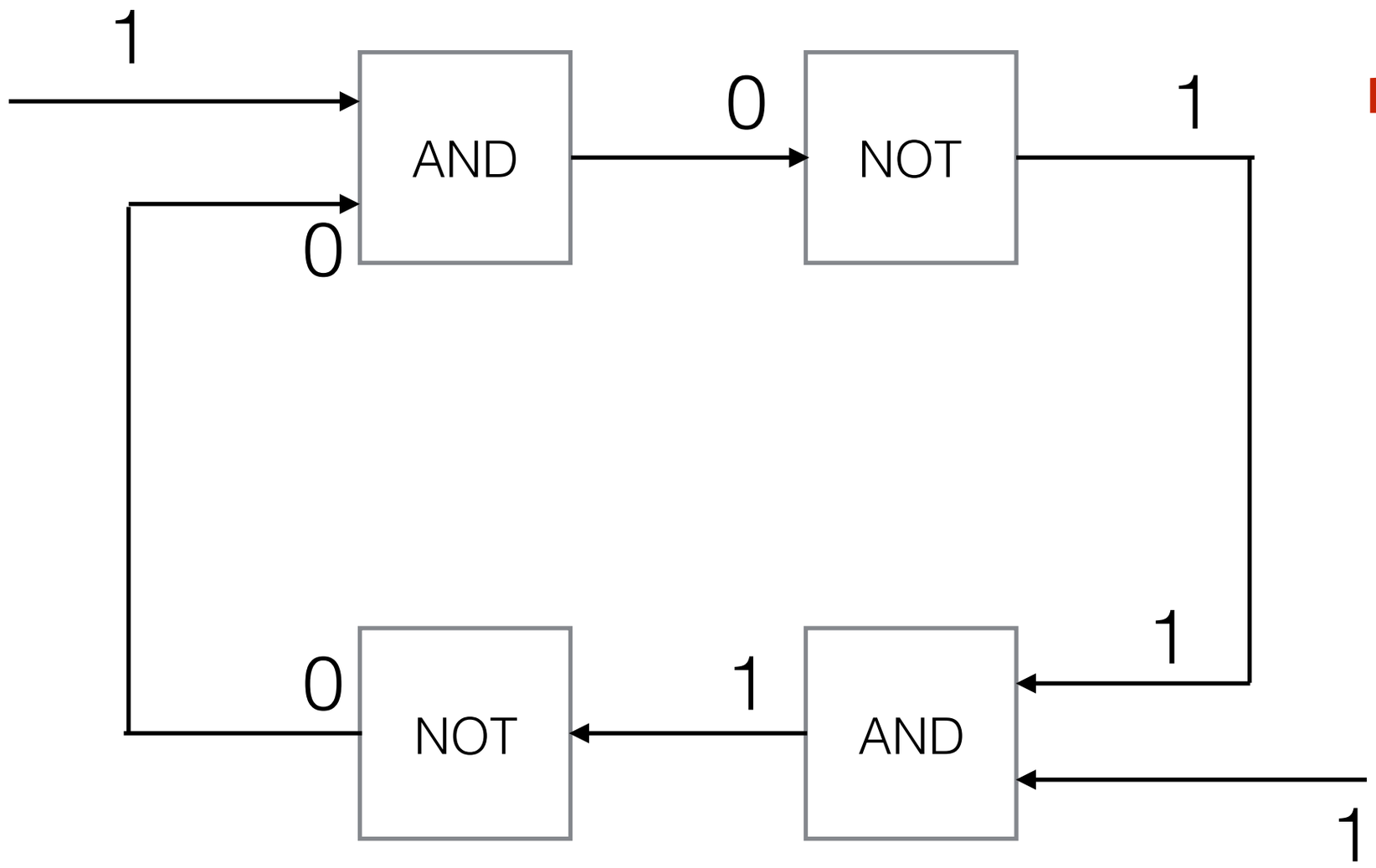

0
AND
0
NOT
1
1
1
NOT
0
AND
1
1



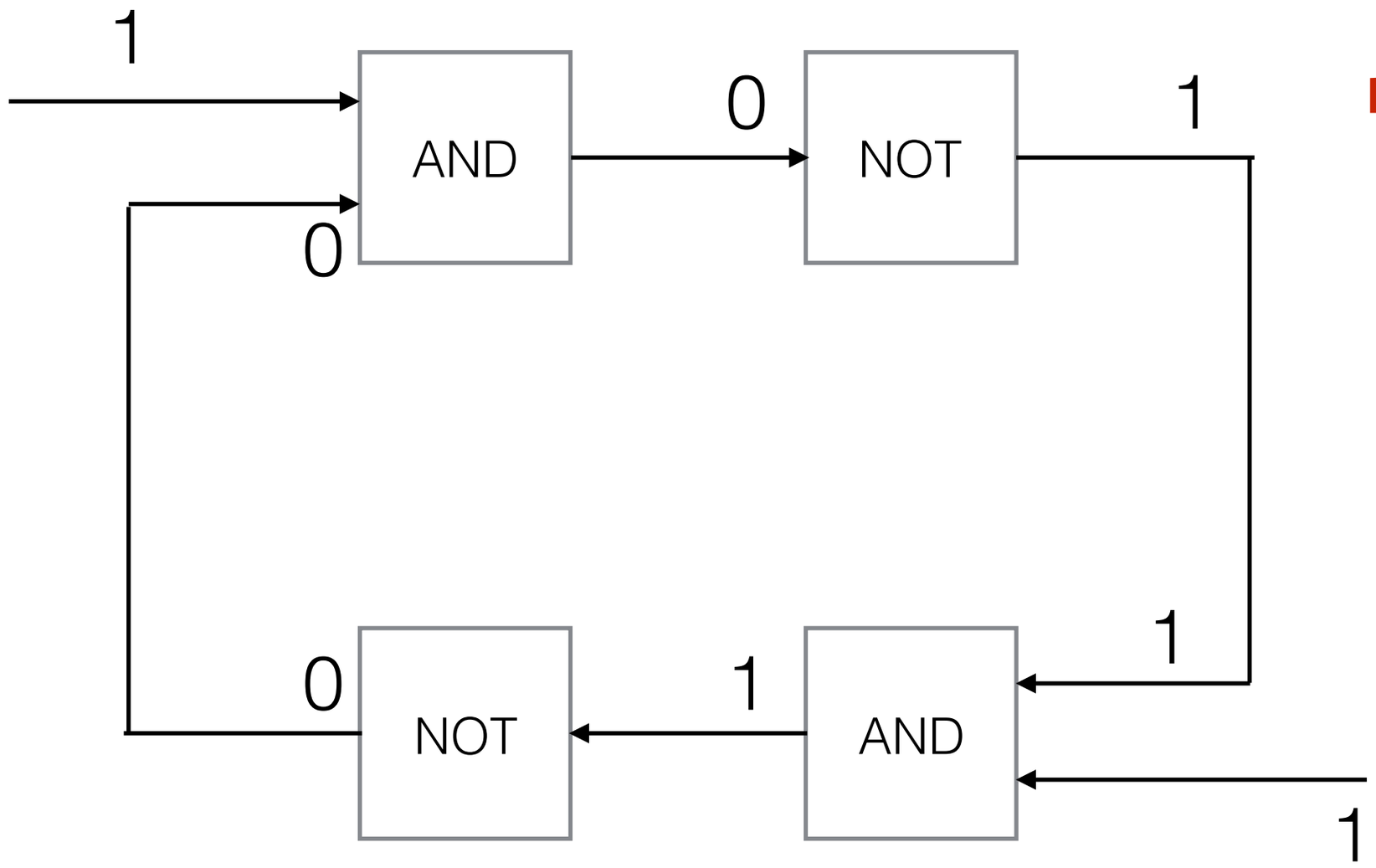
1


0
AND
0
NOT
1
1
0
NOT
1
AND
1
1
1


0
AND
0
0
1
1
NOT
0
1
NOT
1
AND
1
1



1



1

- A **bit** is a device that stores either 1 or 0

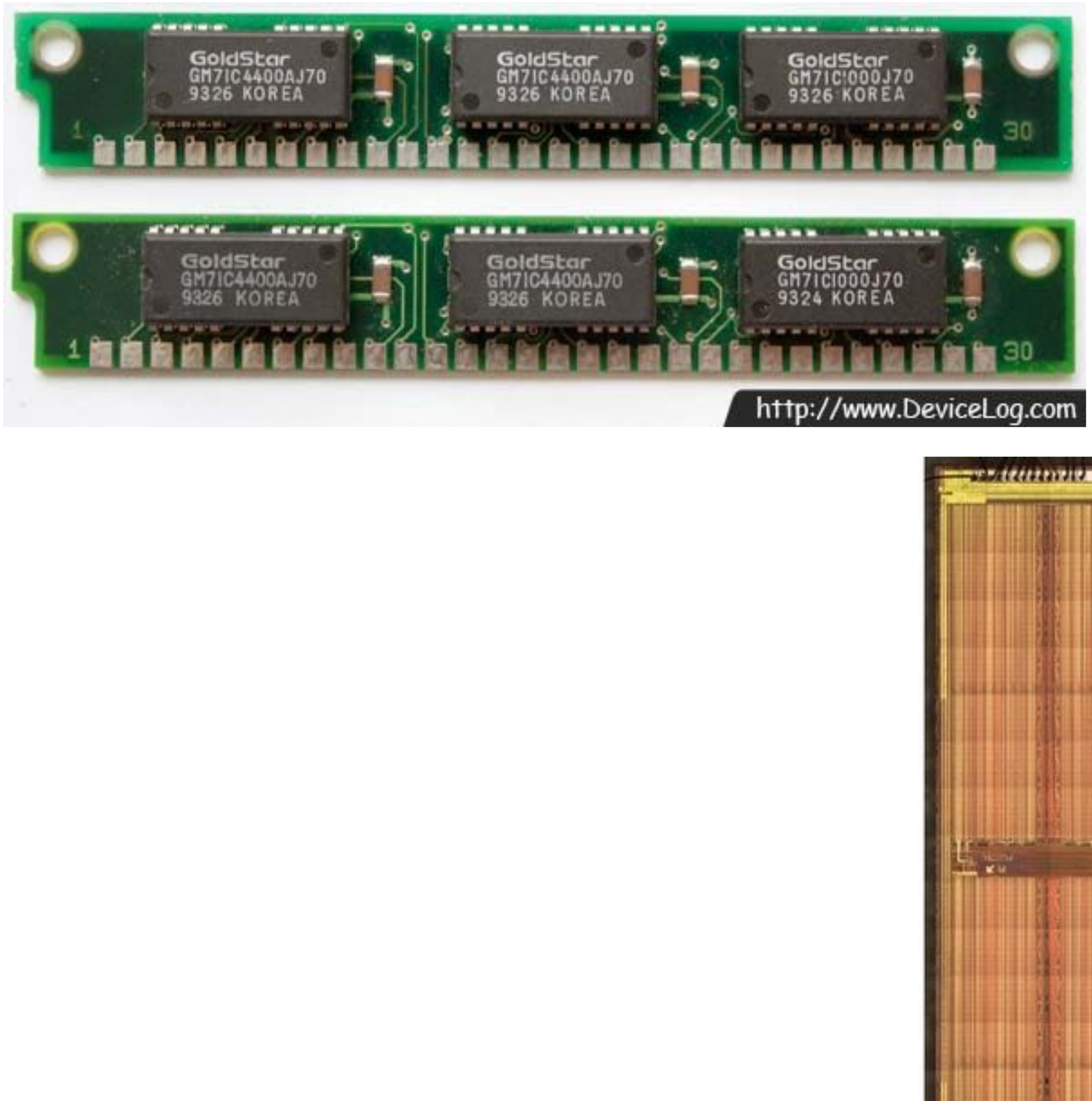
- A bit is a device that stores either 1 or 0
- By extension, a bit is either 1 or 0

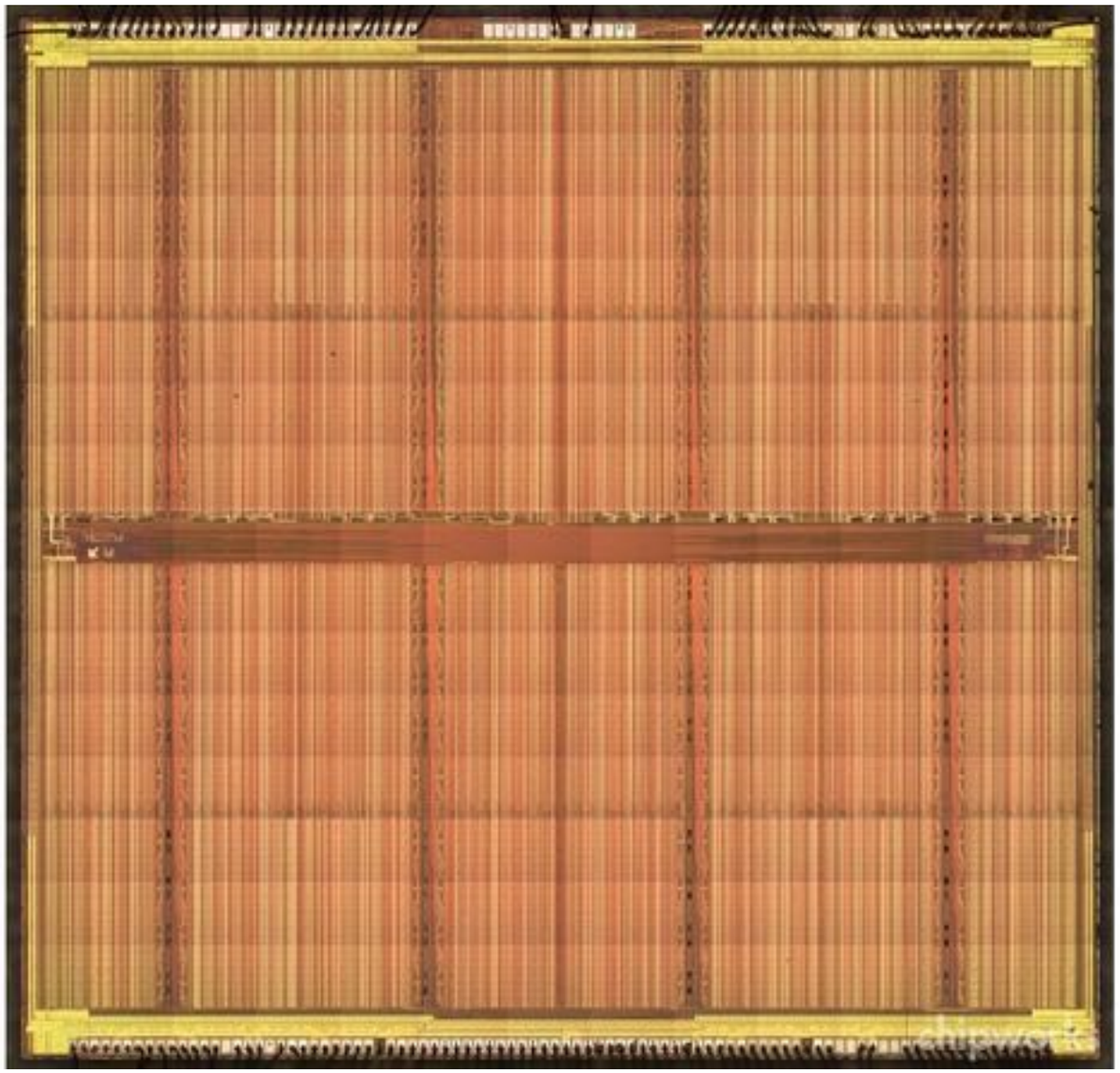
- A bit is a device that stores either 1 or 0
- By extension, a bit is either 1 or 0
- A bit is a unit of information

- A bit is a device that stores either 1 or 0
- By extension, a bit is either 1 or 0
- A bit is a unit of information
- 2 bits take on 1 of 4 states: 00, 01, 10, 11

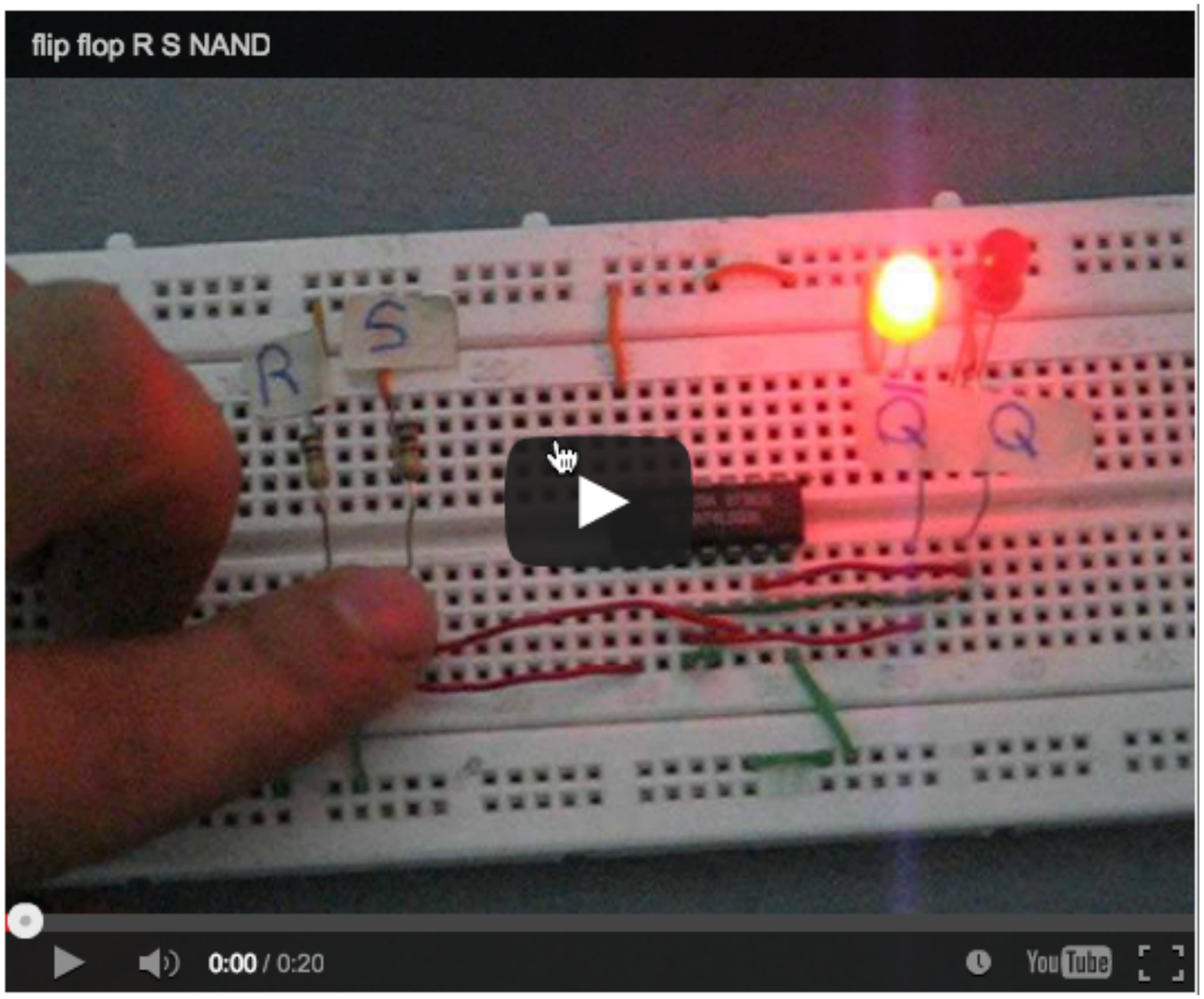
- A bit is a device that stores either 1 or 0
- By extension, a bit is either 1 or 0
- A bit is a unit of information
- 2 bits take on 1 of 4 states: 00, 01, 10, 11
- 3 bits: 000, 001, 010, 011, 100, 101, 110, 111

- A bit is a device that stores either 1 or 0
- By extension, a bit is either 1 or 0
- A bit is a unit of information
- 2 bits take on 1 of 4 states: 00, 01, 10, 11
- 3 bits: 000, 001, 010, 011, 100, 101, 110, 111
- 8 bits = 1 byte
  00000000, 00000001, … to 11111111
  256 possible combinations of 0s and 1s


GoldStar
GM71C4400AJ70
9326 KOREA
GoldStar
GM71C4400AJ70
9326 KOREA
GoldStar
GM71C1000J70
9326 KOREA
1
30
GoldStar
GM71C4400AJ70
9326 KOREA
GoldStar
GM71C4400AJ70
9326 KOREA
GoldStar
GM71C1000J70
9324 KOREA
1
30
http://www.DeviceLog.com



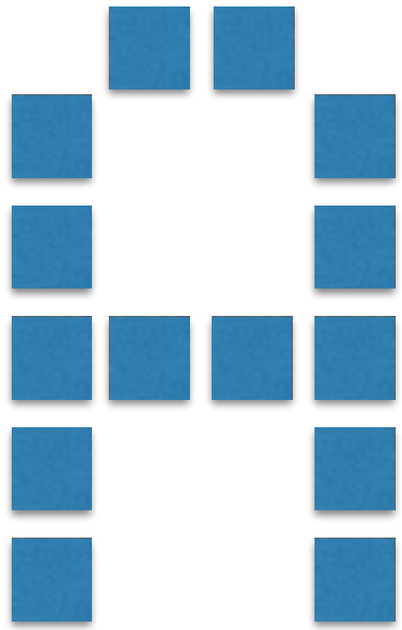
https://www.youtube.com/watch?v=xaTywFdZrOk



A

Character

01000001



0110
1001
1001
1111
1001
1001

Pixel

RED **GREEN** **BLUE**

10001000 **01101010** **00001000**