Unit 1: The Model of the Computer

Dave Abel

February 5th, 2016

Today’s Takeaway

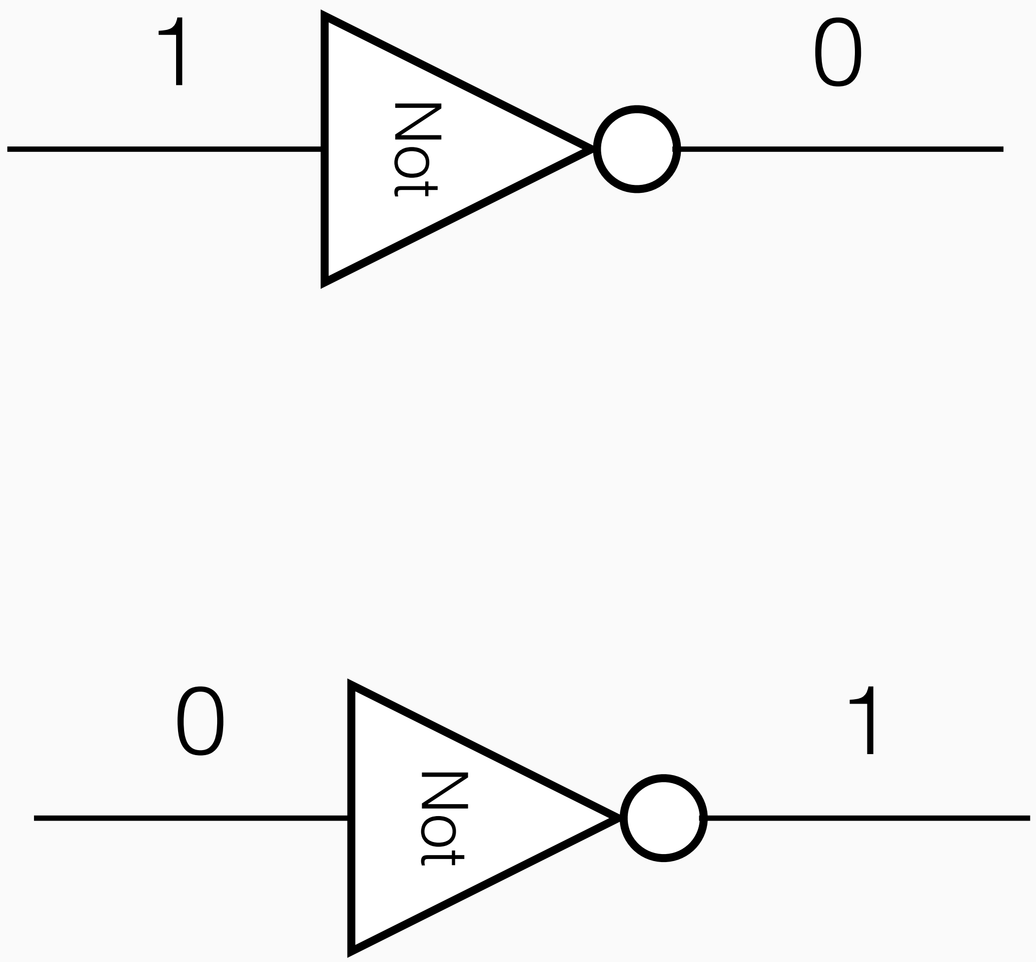
- Earlier this week: computers are doing logic with gates!
- Today: Memory, Logic, and input/output = Computer!

Outline for Today

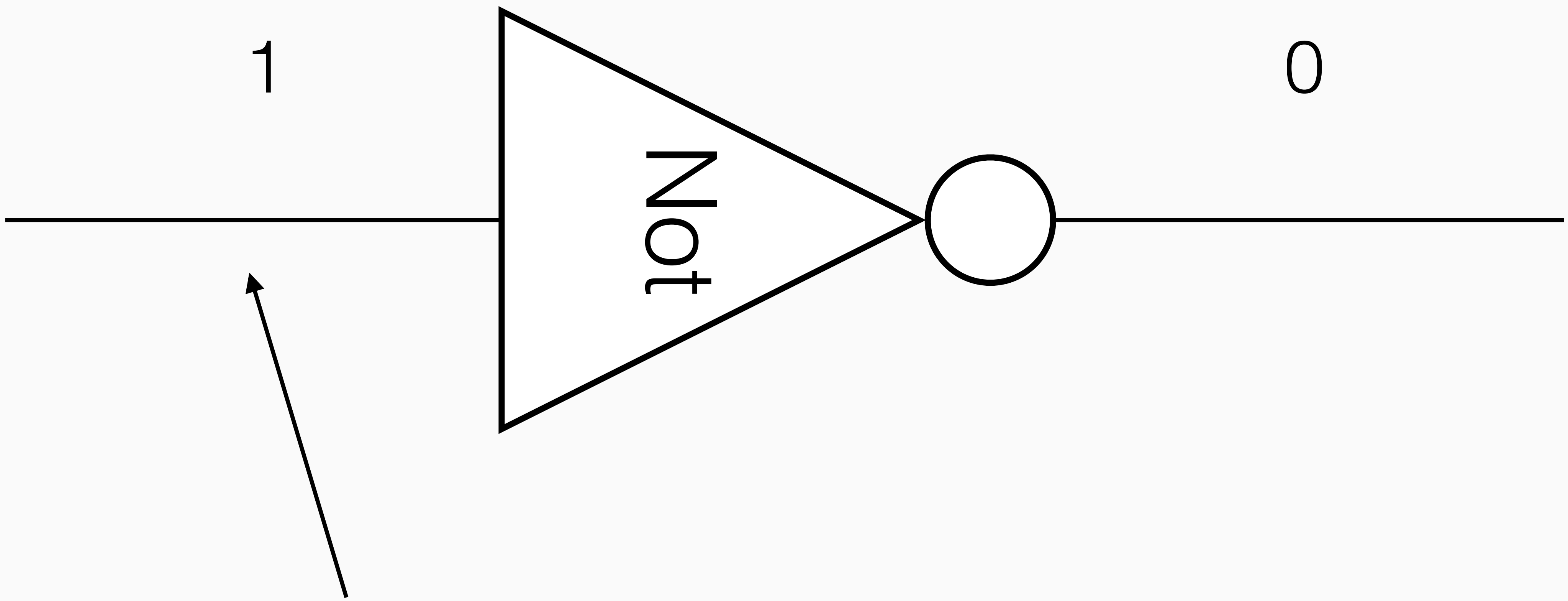
- Review of Gates
  - *AND, OR, NOT, Composite Gates*
  - Domino Gates, *XOR*, Adding with Gates
- State Machines!
- Model of a Computer

Gates: *NOT*

P	*NOT(P)*
1	**0**
0	**1**

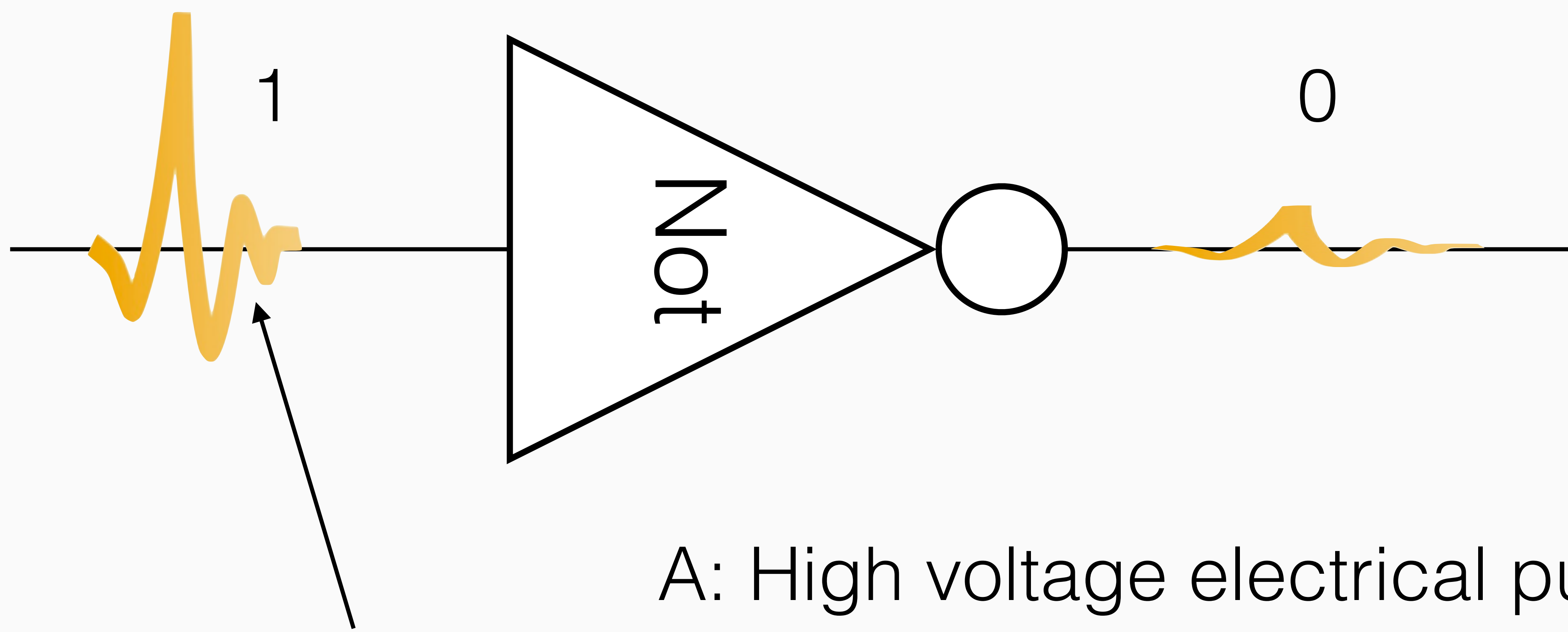


Gates: *NOT*



Q: What is this, physically?

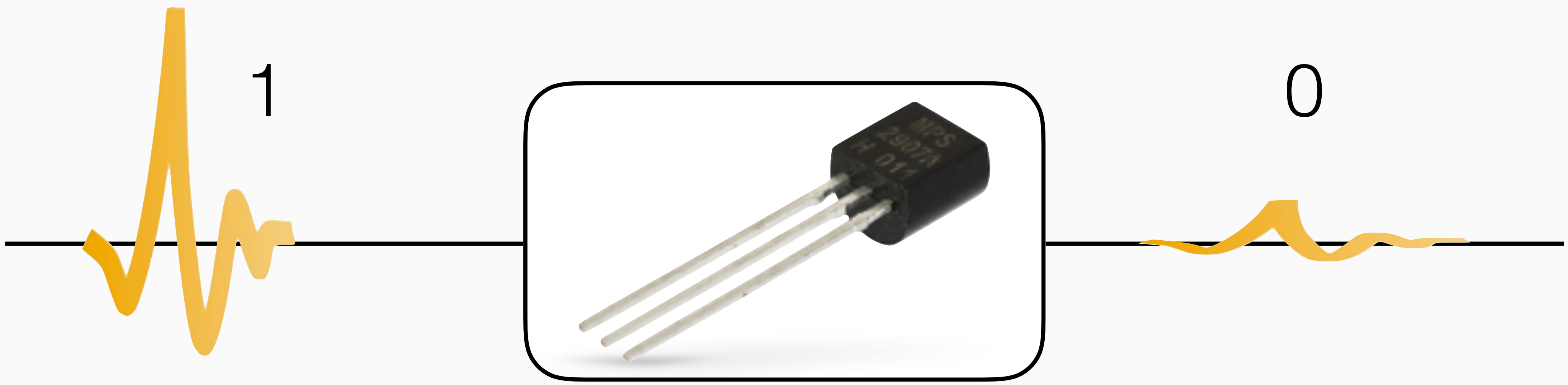
Gates: *NOT*



A: High voltage electrical pulses!

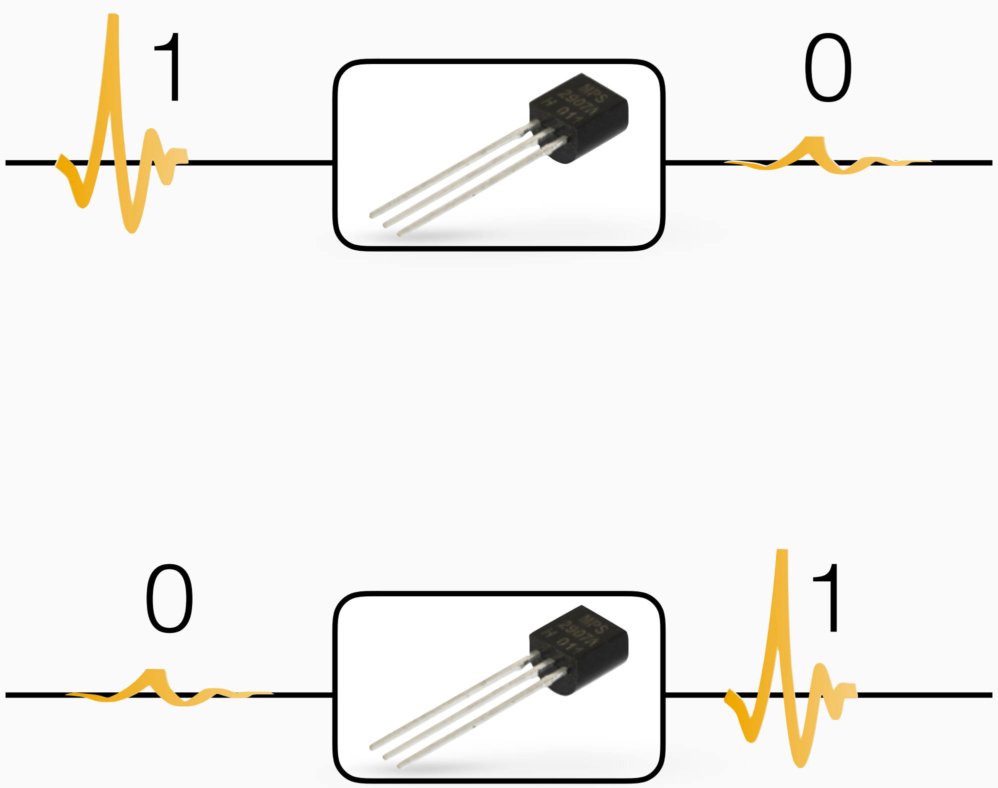
Q: What is this, physically?

Gates: *NOT*



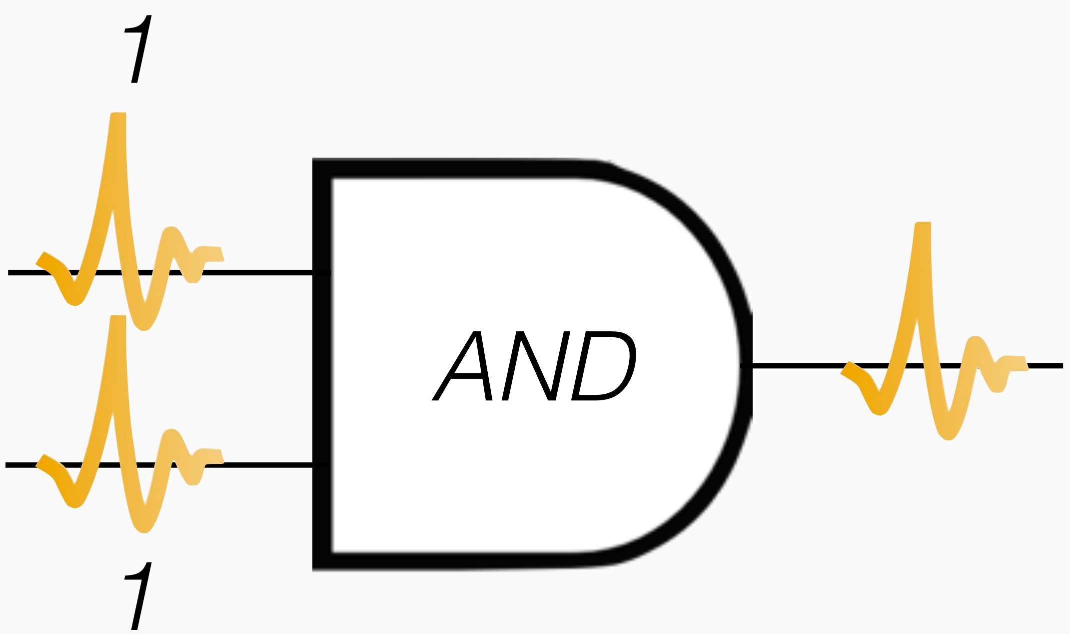
Gates: *NOT*

P	*NOT(P)*
1	**0**
0	**1**



Gates: *AND*

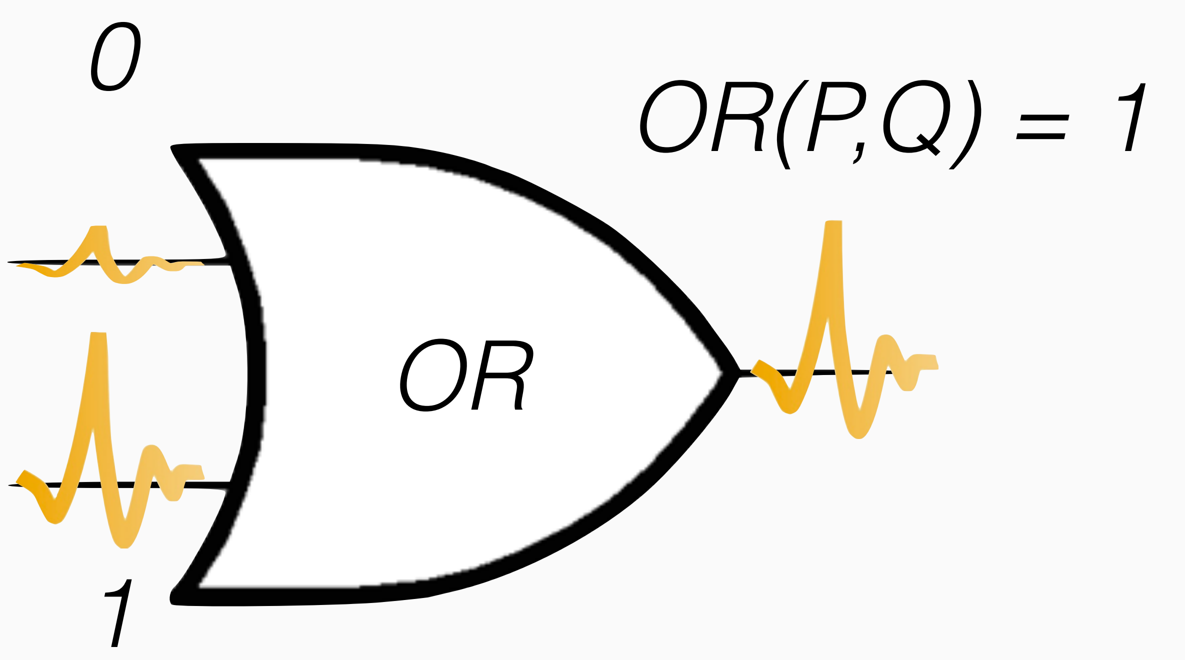
P	*Q*	***AND(***	*P*	*Q)*
1	1	**1**	1	1
1	0	**0**	1	0
0	1	**0**	0	1
0	0	**0**	0	0



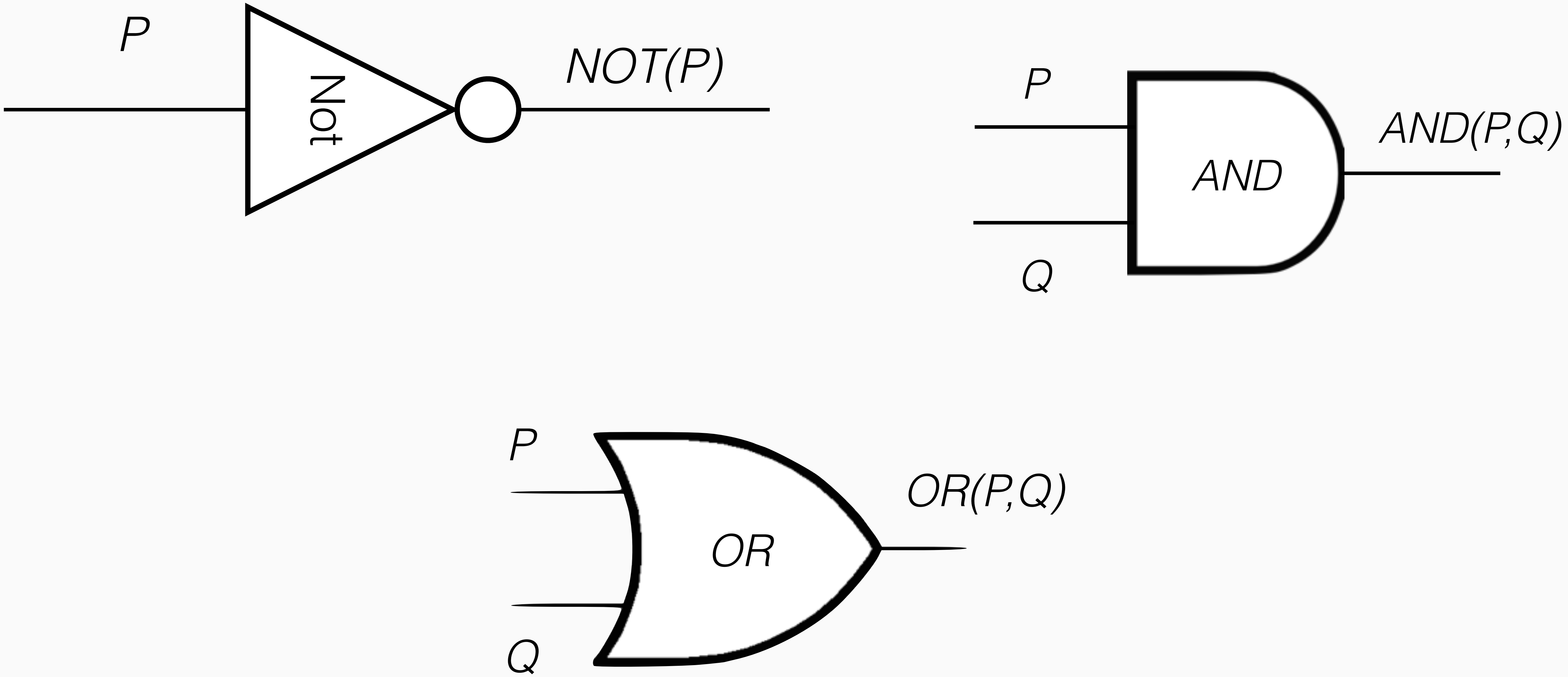
AND(P,Q) = 1

Gates: *OR*

P	*Q*	*OR(*	*P*	*Q)*
1	1	**1**	1	1
1	0	**1**	1	0
0	1	**1**	0	1
0	0	**0**	0	0



Gates

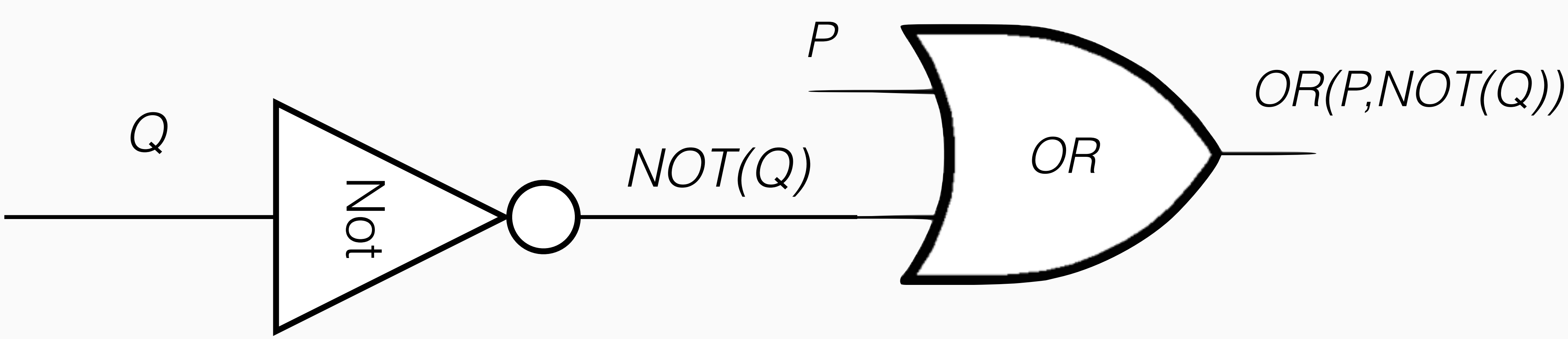


Gates: Composition

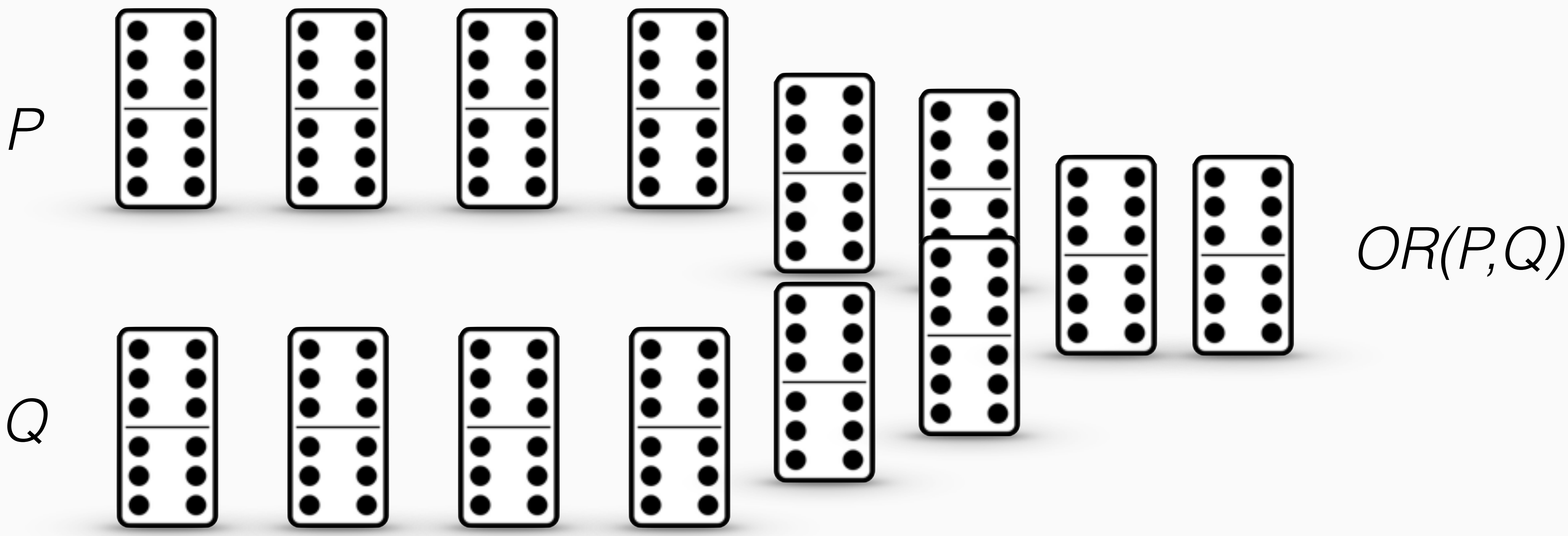
OR(P,NOT(Q))

P	*Q*	*OR(*	*P*	*NOT(*	Q)
T	T	**T**	T	F	T
T	F	**T**	T	T	F
F	T	**F**	F	F	T
F	F	**T**	F	T	F

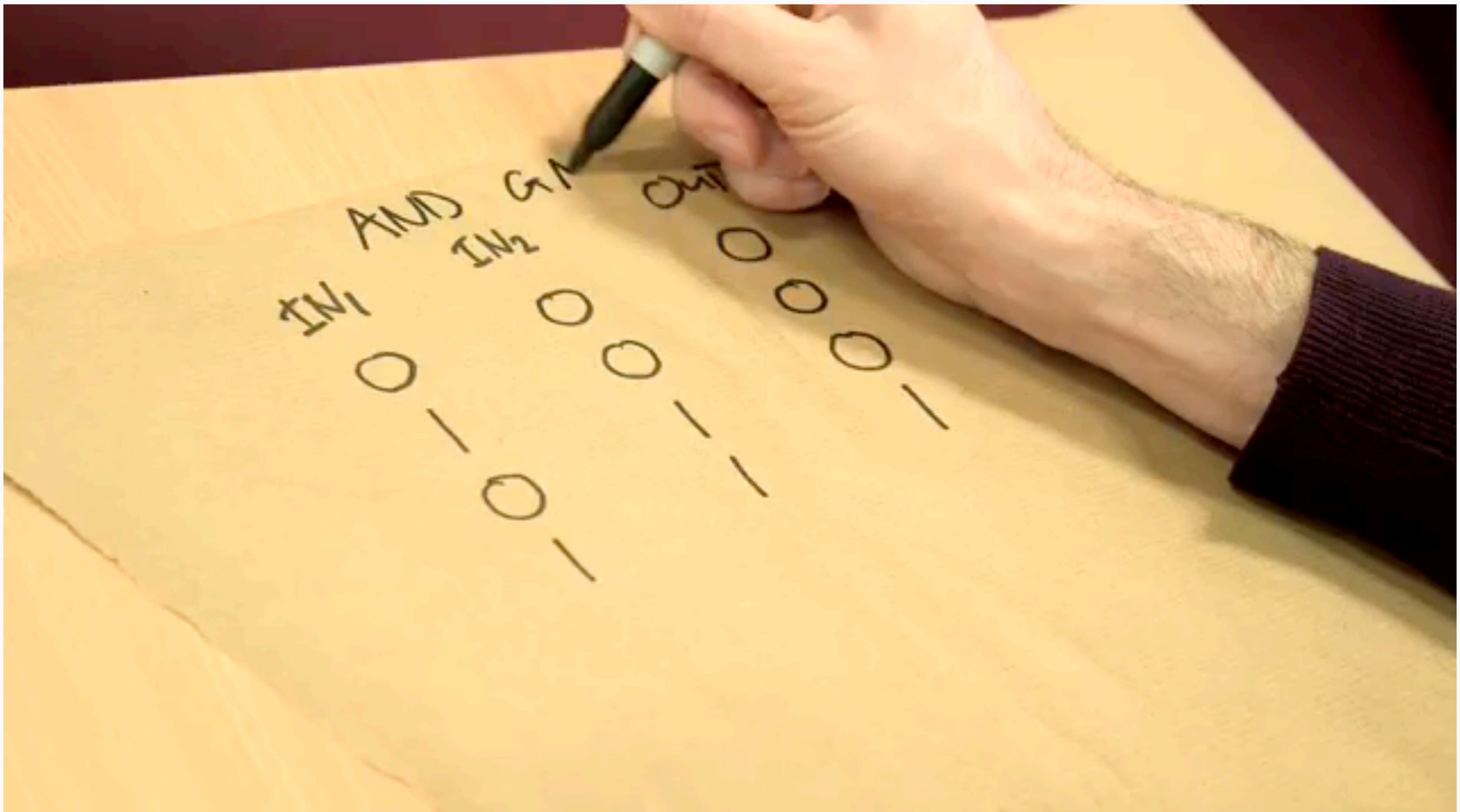
A:



Domino Gates



Domino Gates



Could It Work?

- Michael's domino *OR* gate: 24 dominoes
- The first pentium processor had 3.3 Mill transistors, or roughly 800k gates.
- So we need around 20 Mill dominoes
- World record for domino topple: 4.5 Mill
- Pentium: computes 60 Mill times a second
- Dominoes? Takes awhile to set up…

Two Different Notions of Speed

1. Electricity vs. Dominos falling over

2. Do we have the shortest computation for that problem?

Problem: Speed!

$$P = NOT(NOT(P))$$

Problem: Speed!

	P	*NOT(NOT(P))*
# of Gates:	0	2

Problem: Speed!

	P	$NOT(NOT(NOT(NOT(P))))$
# of Gates:	0	4

Problem: Speed!

	P	*NOT(NOT(NOT(NOT(NOT(P)))))*
# of Gates:	0	6

Problem: Speed!

	P	*NOT(NOT(NOT(NOT(NOT(P)))))…….*
# of Gates:	0	6024

Problem: Speed!

	P	*NOT(NOT(NOT(NOT(NOT(P)))))…….*
# of Gates:	0	6024

Think about that many domino gates…

Problem: Speed!

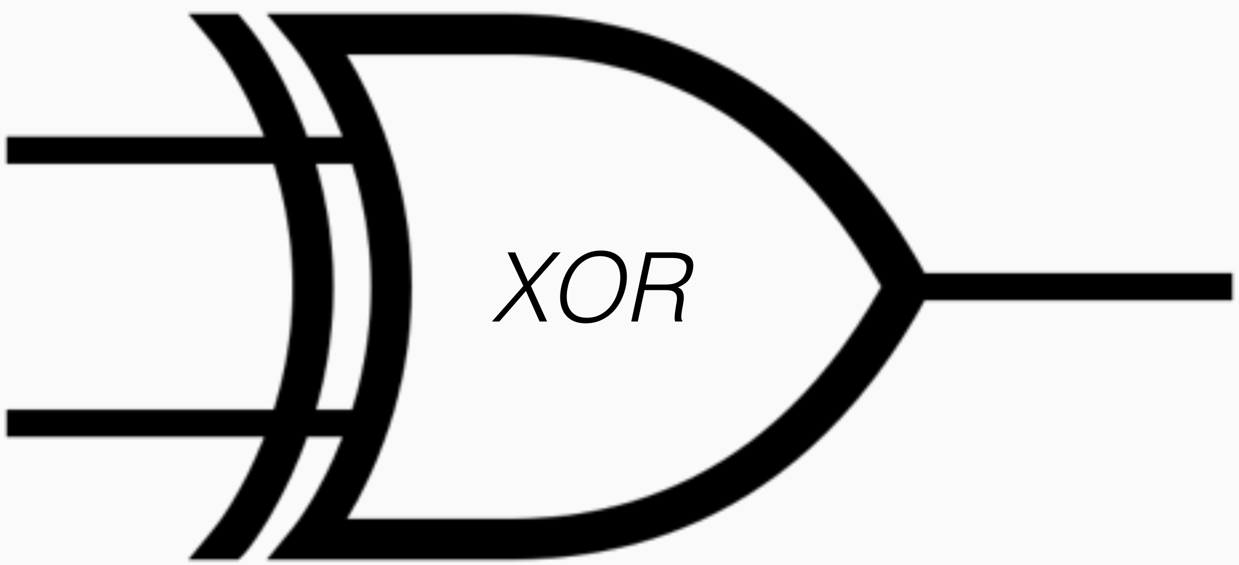
	P	*NOT(NOT(NOT(NOT(NOT(P)))))…….*
# of Gates:	0	6024

Think about that many domino gates…

Identical logical formulas, dramatically different speed of computation!

One Last Gate… *XOR*

P	*Q*	*XOR*(	*P*	*Q*)
1	1	**0**	1	1
1	0	**1**	1	0
0	1	**1**	0	1
0	0	**0**	0	0

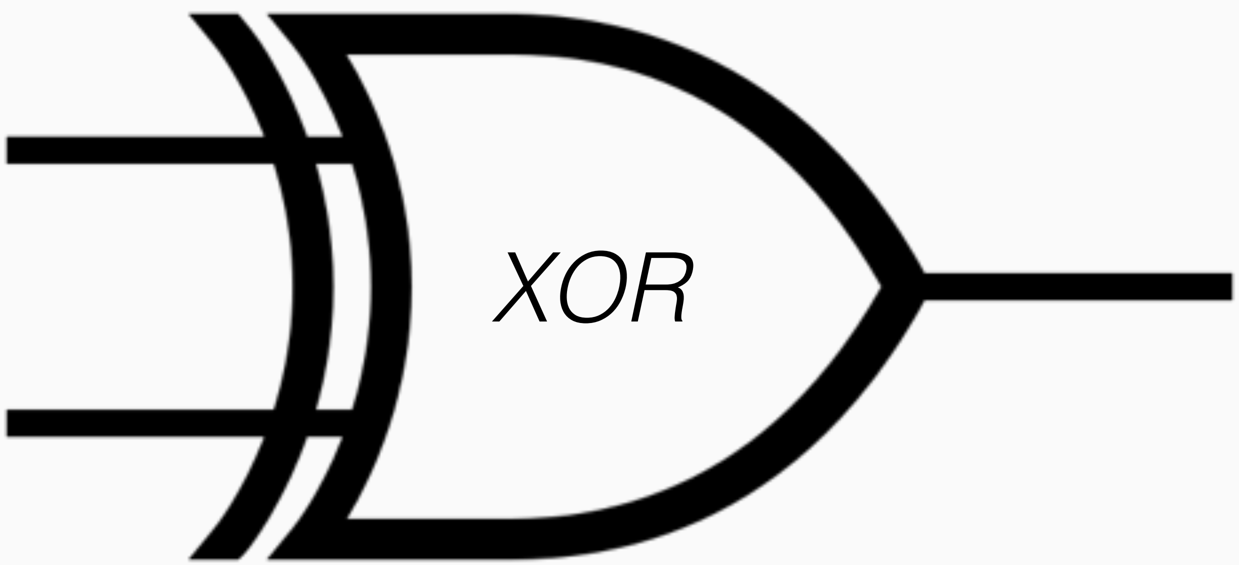


One Last Gate… *XOR*

P	*Q*	*XOR(*	*P*	*Q)*
1	1	**0**	1	1
1	0	**1**	1	0
0	1	**1**	0	1
0	0	**0**	0	0

One Last Gate… *XOR*

P	*Q*	*XOR*(	*P*	*Q*)
1	1	**0**	1	1
1	0	**1**	1	0
0	1	**1**	0	1
0	0	**0**	0	0



Arithmetic and Logic

We know how to do addition, subtraction, and multiplication in binary…

But we've claimed that computers used *AND, OR, NOT* gates at their core to do everything…

Arithmetic and Logic

We know how to do addition, subtraction, and multiplication in binary…

But we've claimed that computers used *AND, OR, NOT* gates at their core to do everything…

Q: Can we represent addition with logical gates?

Arithmetic and Gates

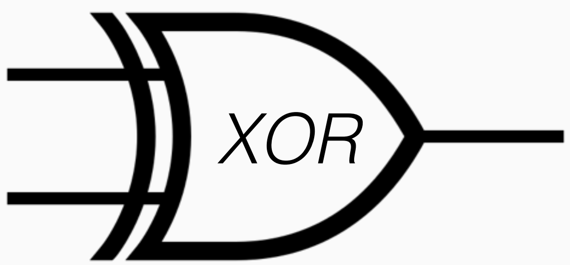
Q: Can we represent addition with logical gates?

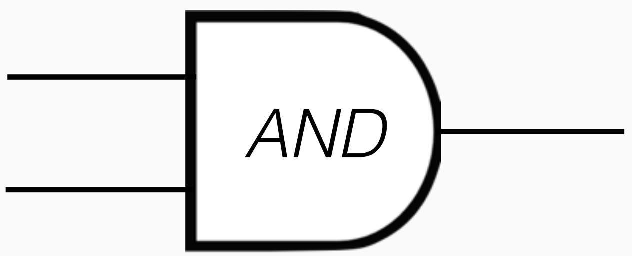
$$\begin{array}{r} 0 \\ +\ 0 \\ \hline 0\ 0 \end{array} \qquad \begin{array}{r} 1 \\ +\ 0 \\ \hline 0\ 1 \end{array} \qquad \begin{array}{r} 0 \\ +\ 1 \\ \hline 0\ 1 \end{array} \qquad \begin{array}{r} {\scriptstyle 1} \\ 1 \\ +\ 1 \\ \hline 1\ 0 \end{array}$$

Arithmetic and Gates

Q: Can we represent addition with logical gates?

Hint: you'll need these:

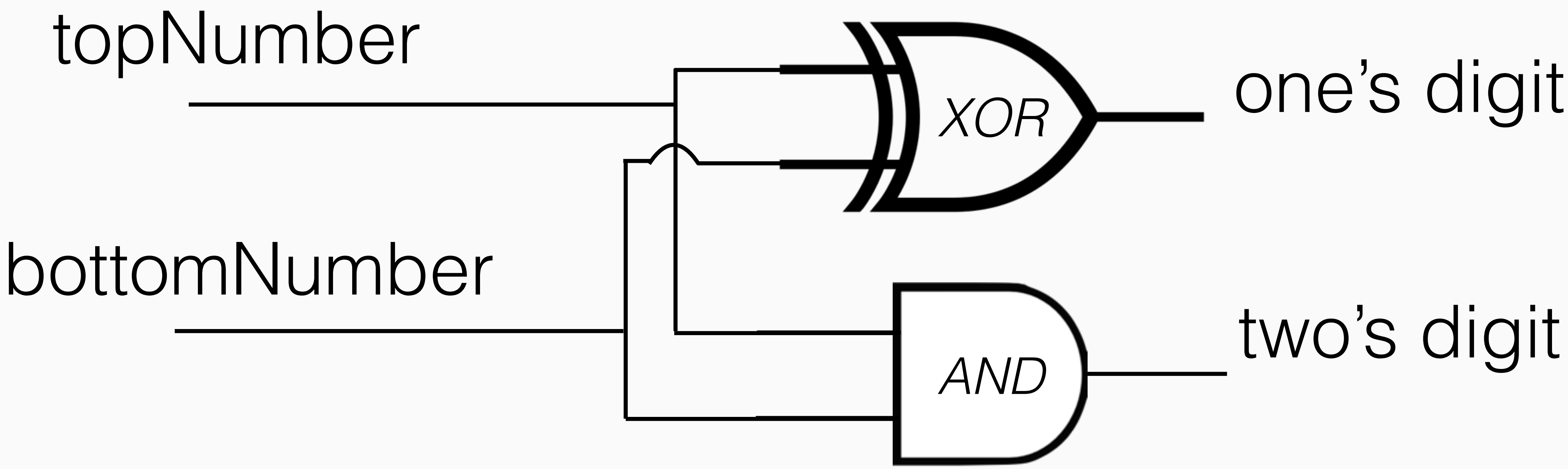




$$\begin{array}{r} 0 \\ +\ 0 \\ \hline 0\ 0 \end{array} \qquad \begin{array}{r} 1 \\ +\ 0 \\ \hline 0\ 1 \end{array} \qquad \begin{array}{r} 0 \\ +\ 1 \\ \hline 0\ 1 \end{array} \qquad \begin{array}{r} \overset{1}{}\ \ \\ 1 \\ +\ 1 \\ \hline 1\ 0 \end{array}$$

Arithmetic and Gates

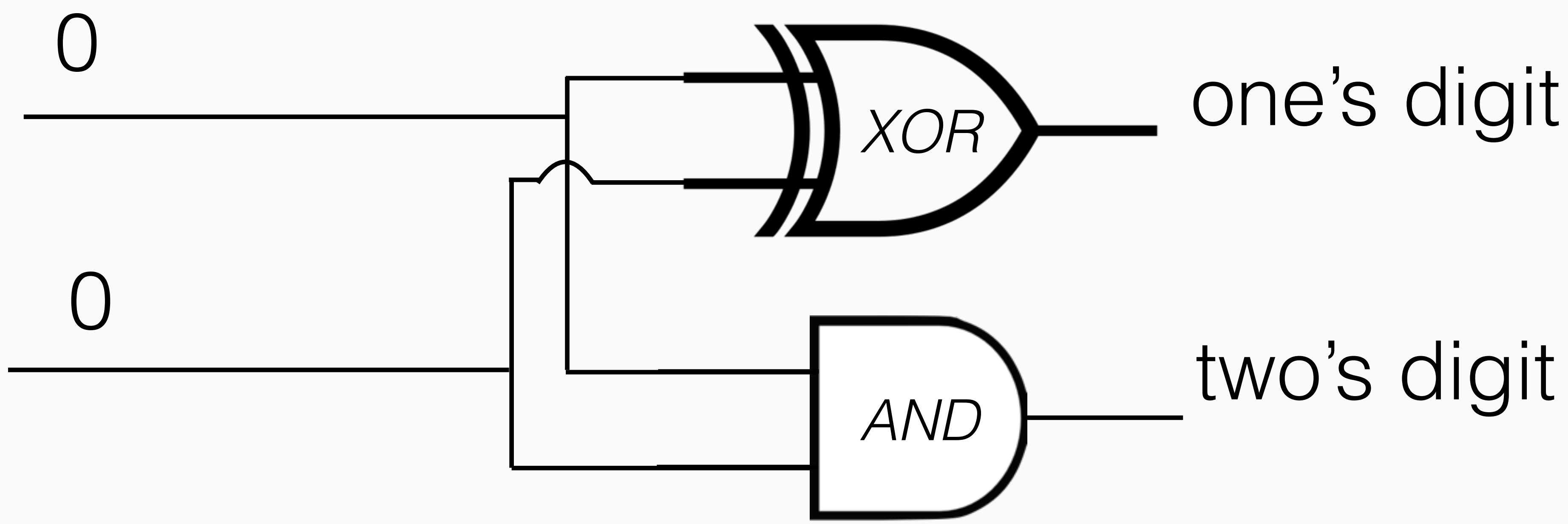
A:



$$\begin{array}{r} 0 \\ +\ 0 \\ \hline 0\ 0 \end{array} \quad \begin{array}{r} 1 \\ +\ 0 \\ \hline 0\ 1 \end{array} \quad \begin{array}{r} 0 \\ +\ 1 \\ \hline 0\ 1 \end{array} \quad \begin{array}{r} {\scriptstyle 1}\ \ \\ 1 \\ +\ 1 \\ \hline 1\ 0 \end{array}$$

Arithmetic and Gates

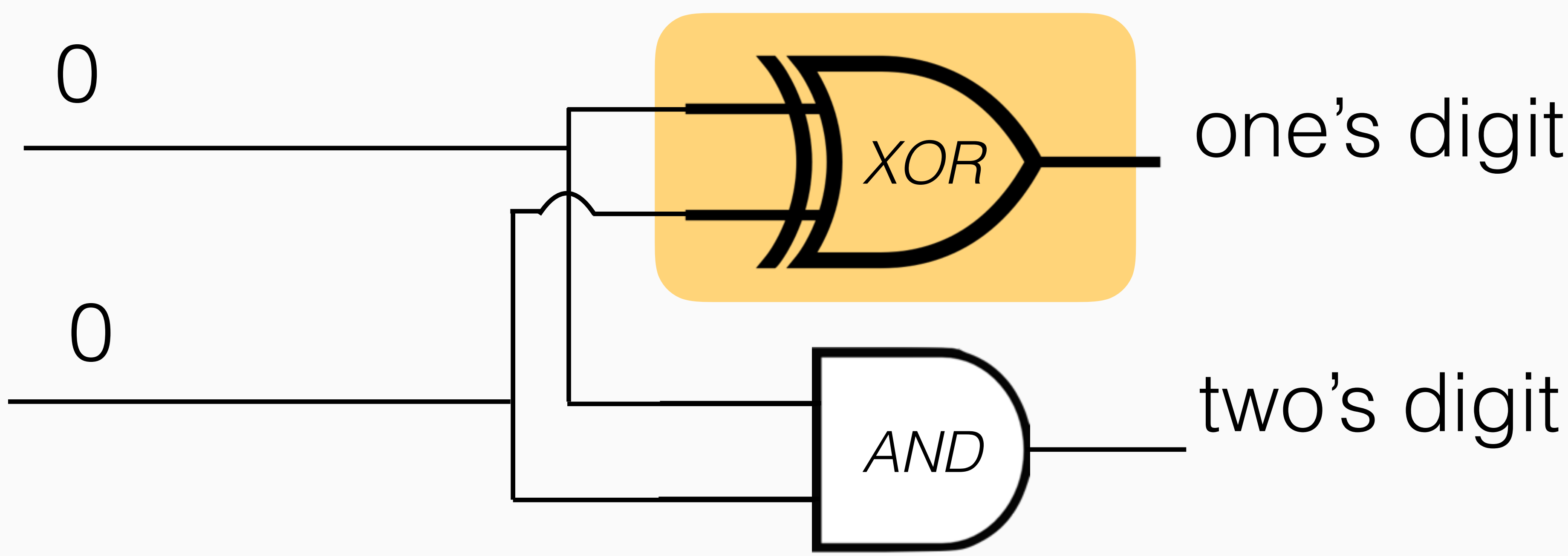
A:



$$\begin{array}{r} 0 \\ +\ 0 \\ \hline 0\ 0 \end{array} \qquad \begin{array}{r} 1 \\ +\ 0 \\ \hline 0\ 1 \end{array} \qquad \begin{array}{r} 0 \\ +\ 1 \\ \hline 0\ 1 \end{array} \qquad \begin{array}{r} {\scriptstyle 1}\ \ \\ 1 \\ +\ 1 \\ \hline 1\ 0 \end{array}$$

Arithmetic and Gates

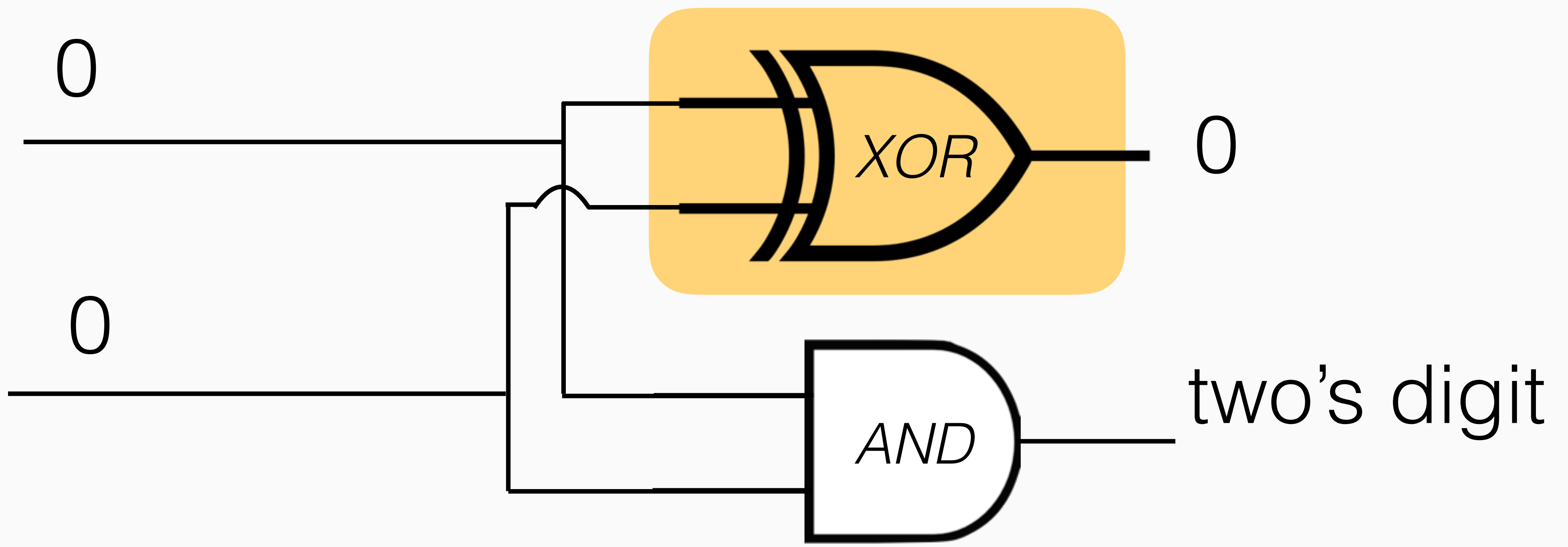
A:



$$\begin{array}{r} 0 \\ +\ 0 \\ \hline 0\ 0 \end{array} \qquad \begin{array}{r} 1 \\ +\ 0 \\ \hline 0\ 1 \end{array} \qquad \begin{array}{r} 0 \\ +\ 1 \\ \hline 0\ 1 \end{array} \qquad \begin{array}{r} {\scriptstyle 1}\quad \\ 1 \\ +\ 1 \\ \hline 1\ 0 \end{array}$$

Arithmetic and Gates

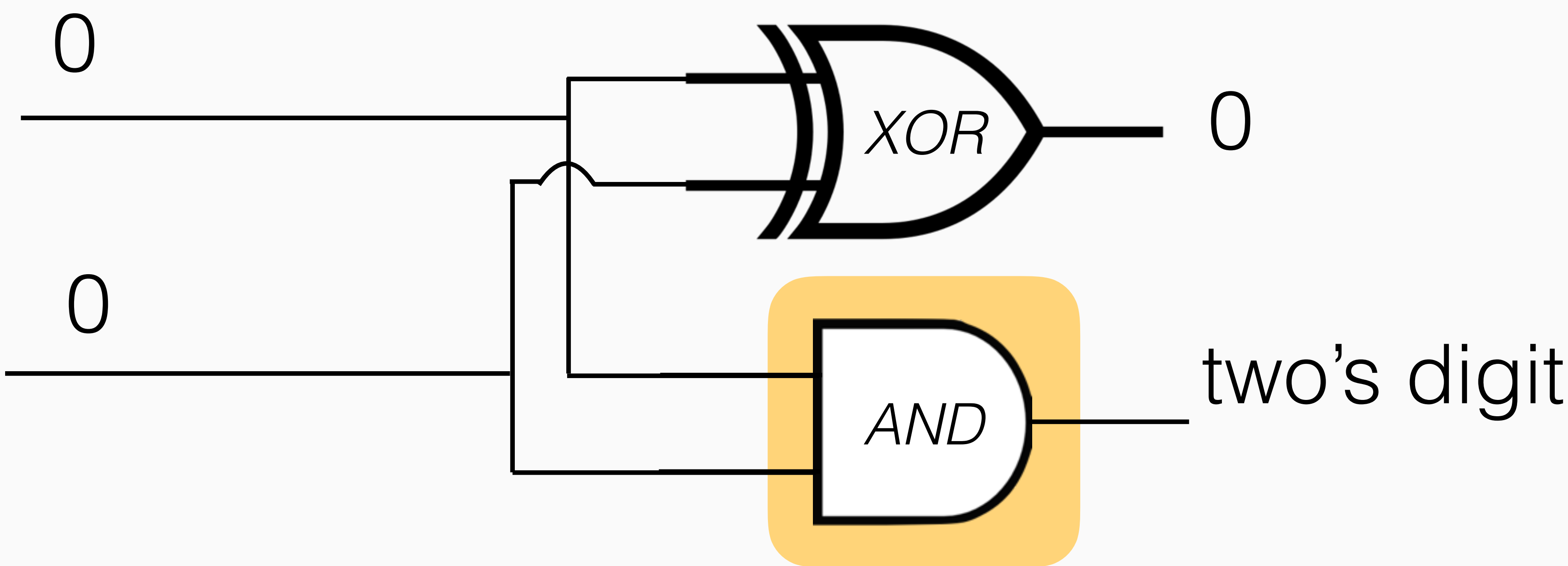
A:



$$\begin{array}{r} 0 \\ +\ 0 \\ \hline 0\ 0 \end{array} \qquad \begin{array}{r} 1 \\ +\ 0 \\ \hline 0\ 1 \end{array} \qquad \begin{array}{r} 0 \\ +\ 1 \\ \hline 0\ 1 \end{array} \qquad \begin{array}{r} {}^{1}\quad 1 \\ +\ 1 \\ \hline 1\ 0 \end{array}$$

Arithmetic and Gates

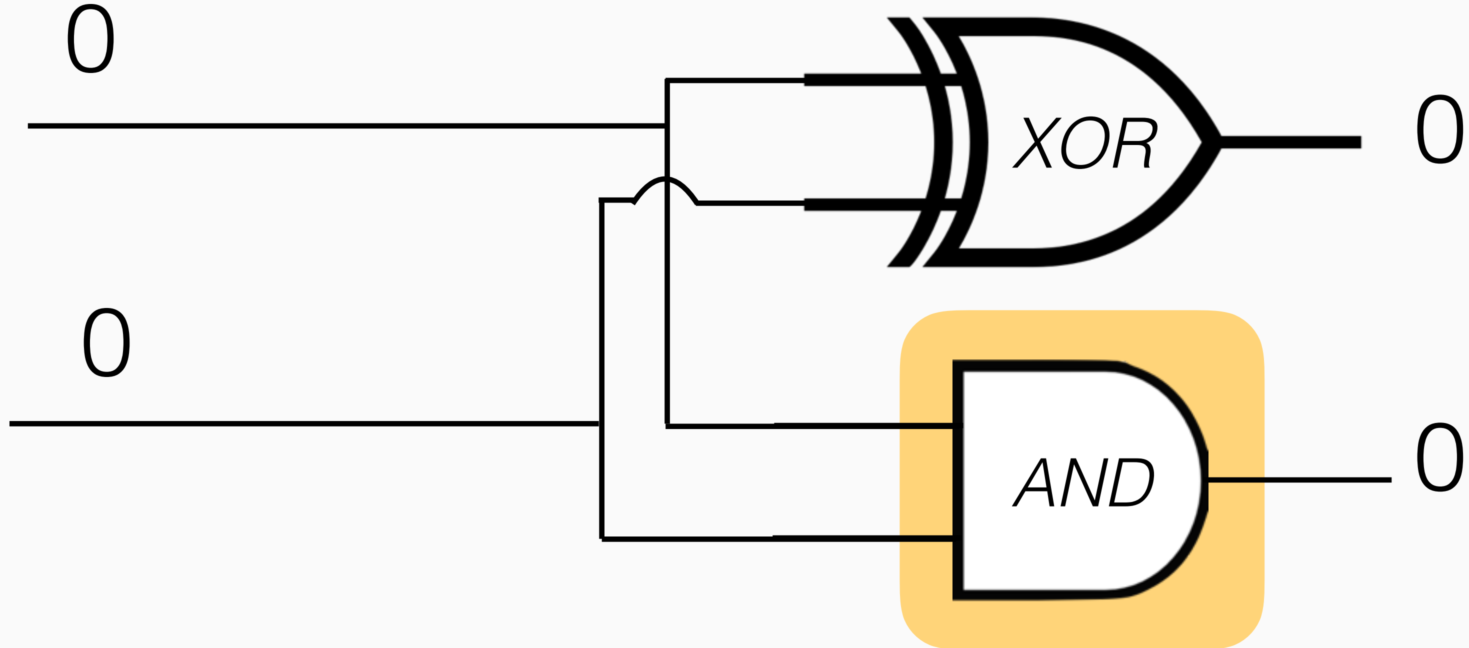
A:



$$\begin{array}{r} 0 \\ +\ 0 \\ \hline 0\ 0 \end{array} \qquad \begin{array}{r} 1 \\ +\ 0 \\ \hline 0\ 1 \end{array} \qquad \begin{array}{r} 0 \\ +\ 1 \\ \hline 0\ 1 \end{array} \qquad \begin{array}{r} {}^{1}\ \ \\ 1 \\ +\ 1 \\ \hline 1\ 0 \end{array}$$

Arithmetic and Gates

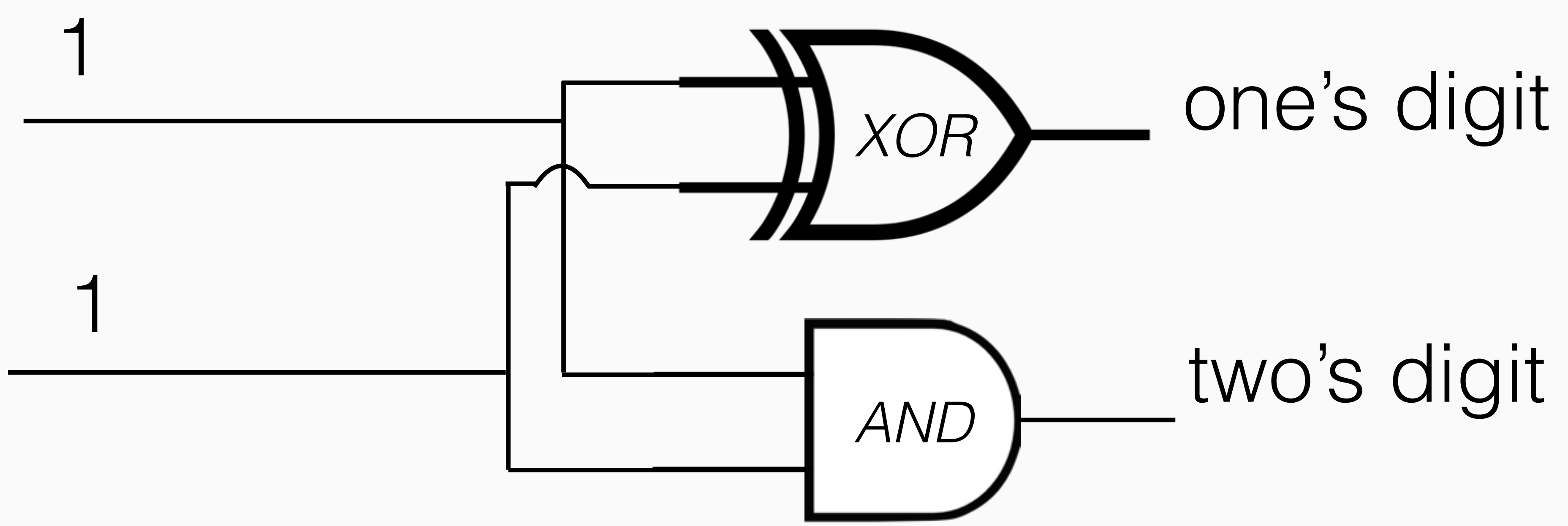
A:



$$\begin{array}{r} 0 \\ +\ 0 \\ \hline 0\ 0 \end{array} \qquad \begin{array}{r} 1 \\ +\ 0 \\ \hline 0\ 1 \end{array} \qquad \begin{array}{r} 0 \\ +\ 1 \\ \hline 0\ 1 \end{array} \qquad \begin{array}{r} {}^{1}\ \ \ \\ 1 \\ +\ 1 \\ \hline 1\ 0 \end{array}$$

Arithmetic and Gates

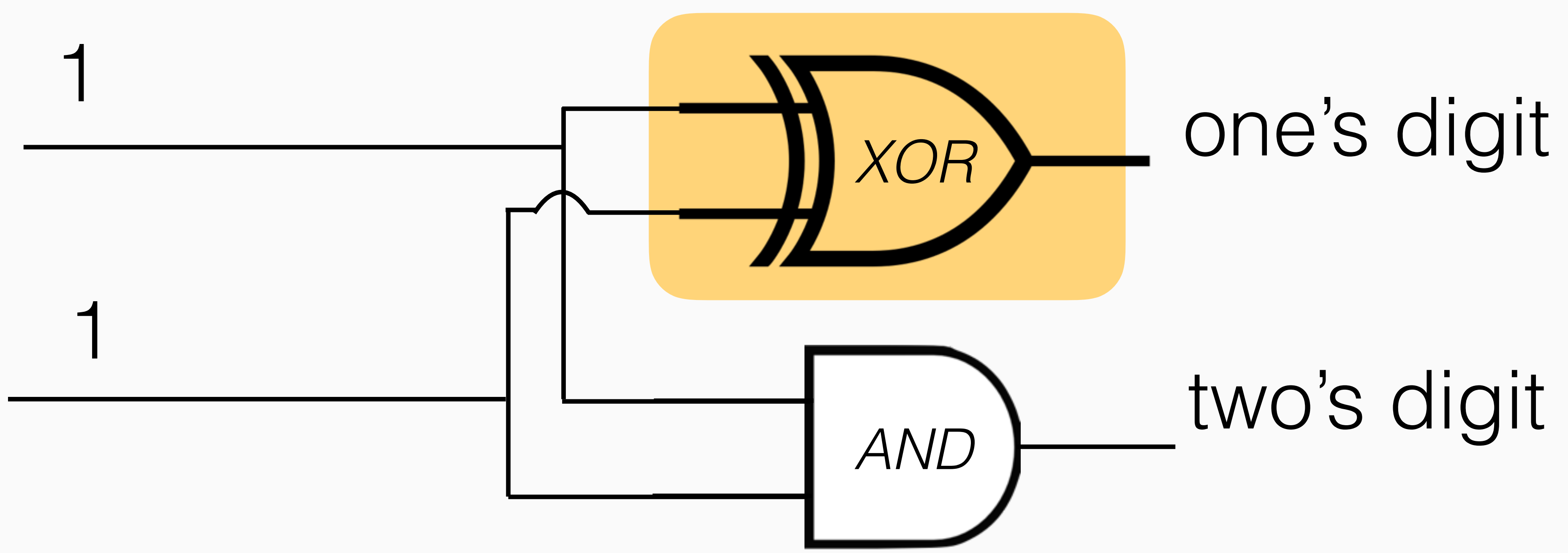
A:



$$\begin{array}{r} 0 \\ +\ 0 \\ \hline 0\ 0 \end{array} \qquad \begin{array}{r} 1 \\ +\ 0 \\ \hline 0\ 1 \end{array} \qquad \begin{array}{r} 0 \\ +\ 1 \\ \hline 0\ 1 \end{array} \qquad \begin{array}{r} 1\ \ 1 \\ +\ \ 1 \\ \hline 1\ \ 0 \end{array}$$

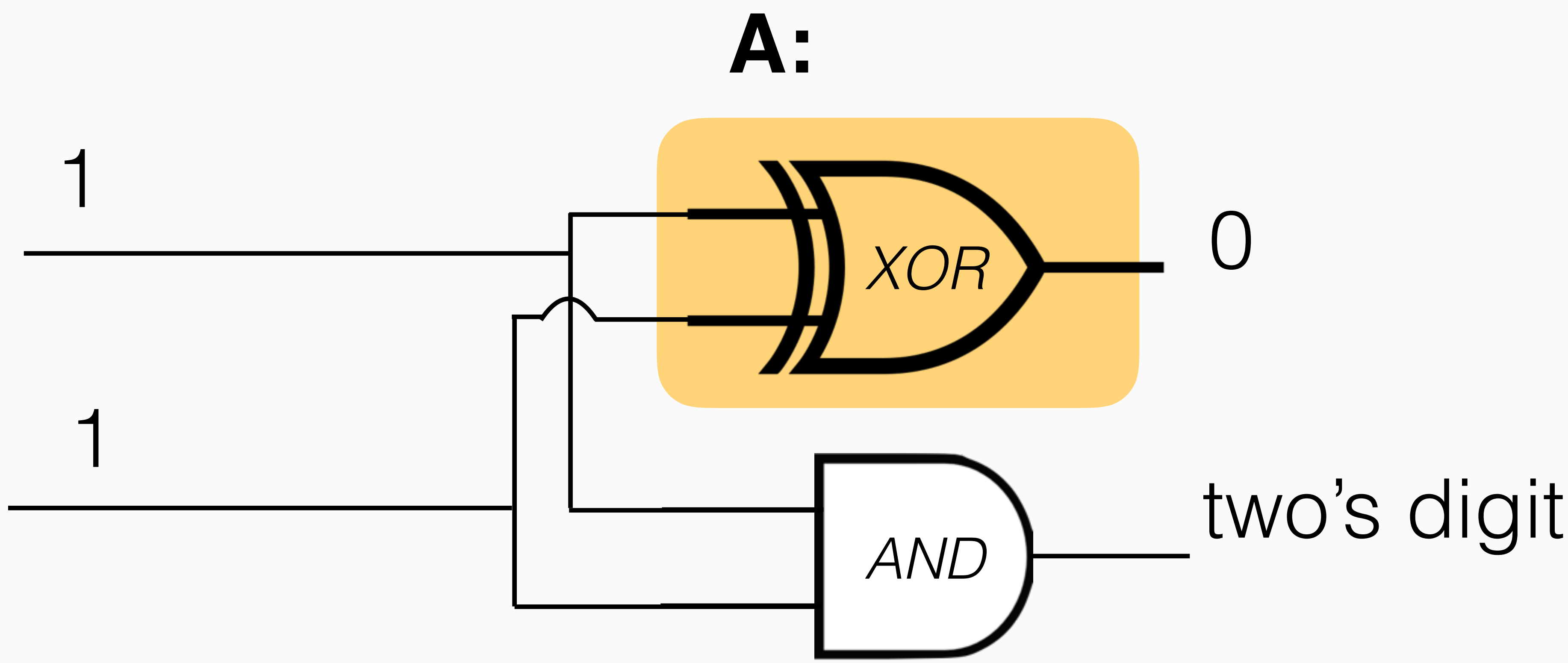
Arithmetic and Gates

A:



$$\begin{array}{r} 0 \\ +\,0 \\ \hline 0\;0 \end{array} \qquad \begin{array}{r} 1 \\ +\,0 \\ \hline 0\;1 \end{array} \qquad \begin{array}{r} 0 \\ +\,1 \\ \hline 0\;1 \end{array} \qquad \begin{array}{r} {}^{1}\;\;1 \\ +\;\;1 \\ \hline 1\;\;0 \end{array}$$

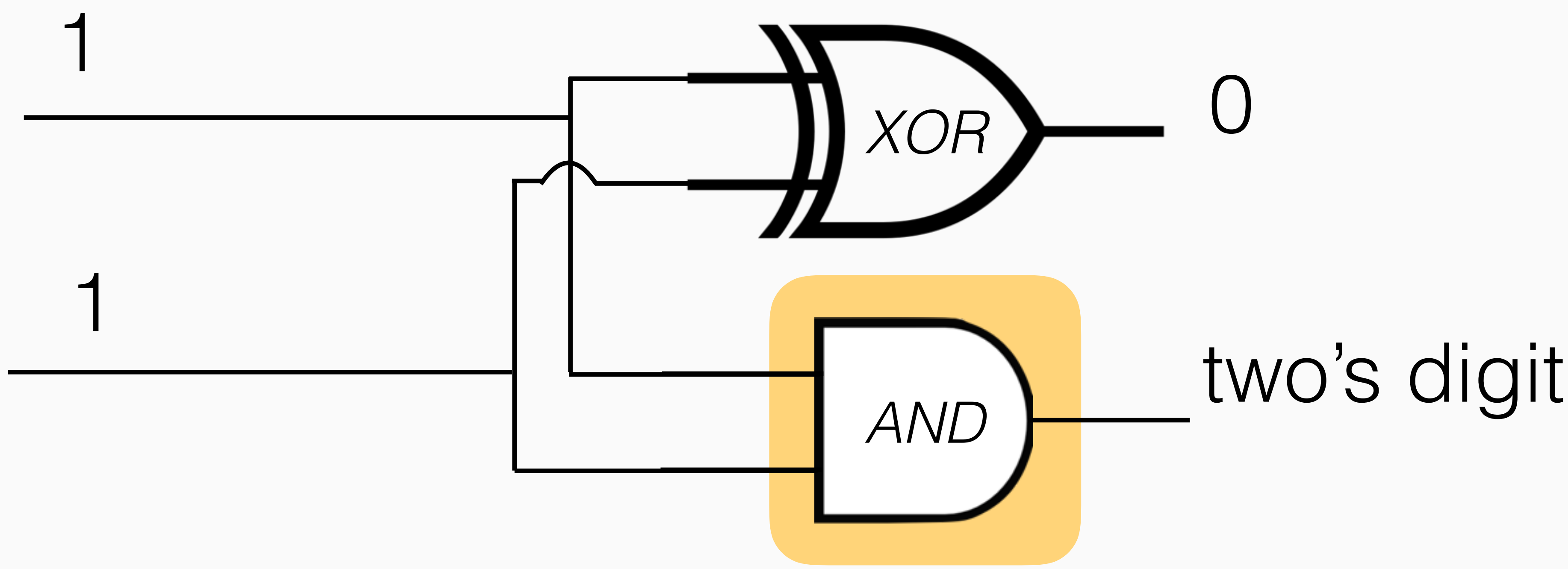
Arithmetic and Gates



$$\begin{array}{r} 0 \\ +\ 0 \\ \hline 0\ 0 \end{array} \qquad \begin{array}{r} 1 \\ +\ 0 \\ \hline 0\ 1 \end{array} \qquad \begin{array}{r} 0 \\ +\ 1 \\ \hline 0\ 1 \end{array} \qquad \begin{array}{r} 1\ \ 1 \\ +\ \ 1 \\ \hline 1\ \ 0 \end{array}$$

Arithmetic and Gates

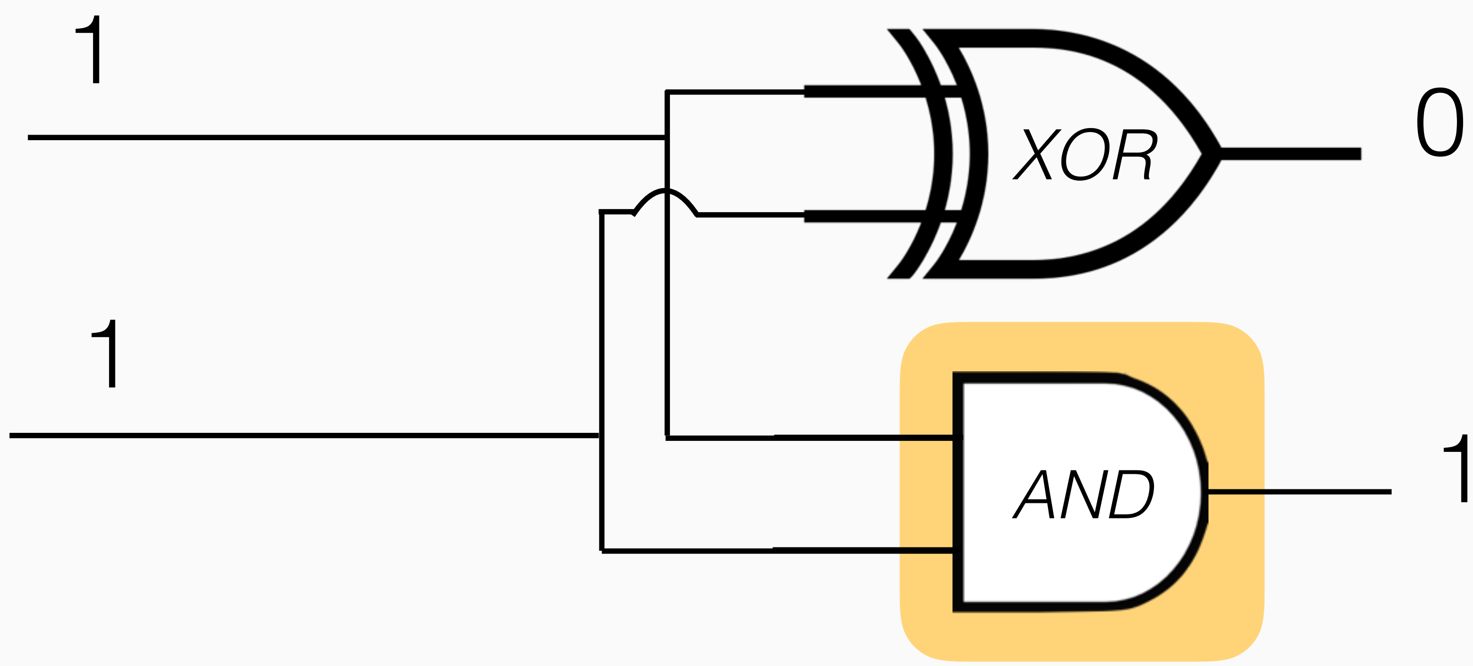
A:



$$\begin{array}{r}0\\+\,0\\\hline 0\;0\end{array}\qquad\begin{array}{r}1\\+\,0\\\hline 0\;1\end{array}\qquad\begin{array}{r}0\\+\,1\\\hline 0\;1\end{array}\qquad\begin{array}{r}{\color{gray}1}\;\;1\\+\;\;1\\\hline 1\;\;0\end{array}$$

Arithmetic and Gates

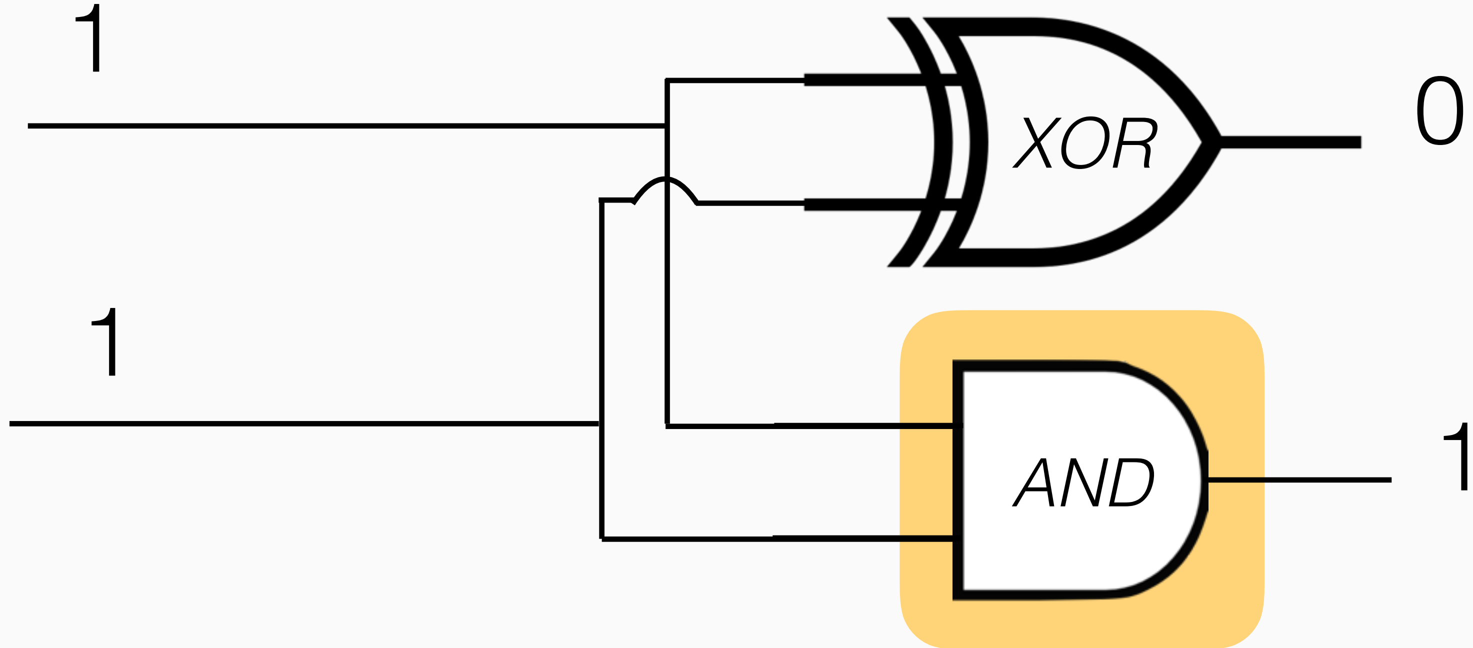
A:



$$\begin{array}{r} 0 \\ +\ 0 \\ \hline 0\ 0 \end{array} \qquad \begin{array}{r} 1 \\ +\ 0 \\ \hline 0\ 1 \end{array} \qquad \begin{array}{r} 0 \\ +\ 1 \\ \hline 0\ 1 \end{array} \qquad \begin{array}{r} 1\ \ 1 \\ +\ \ 1 \\ \hline 1\ \ 0 \end{array}$$

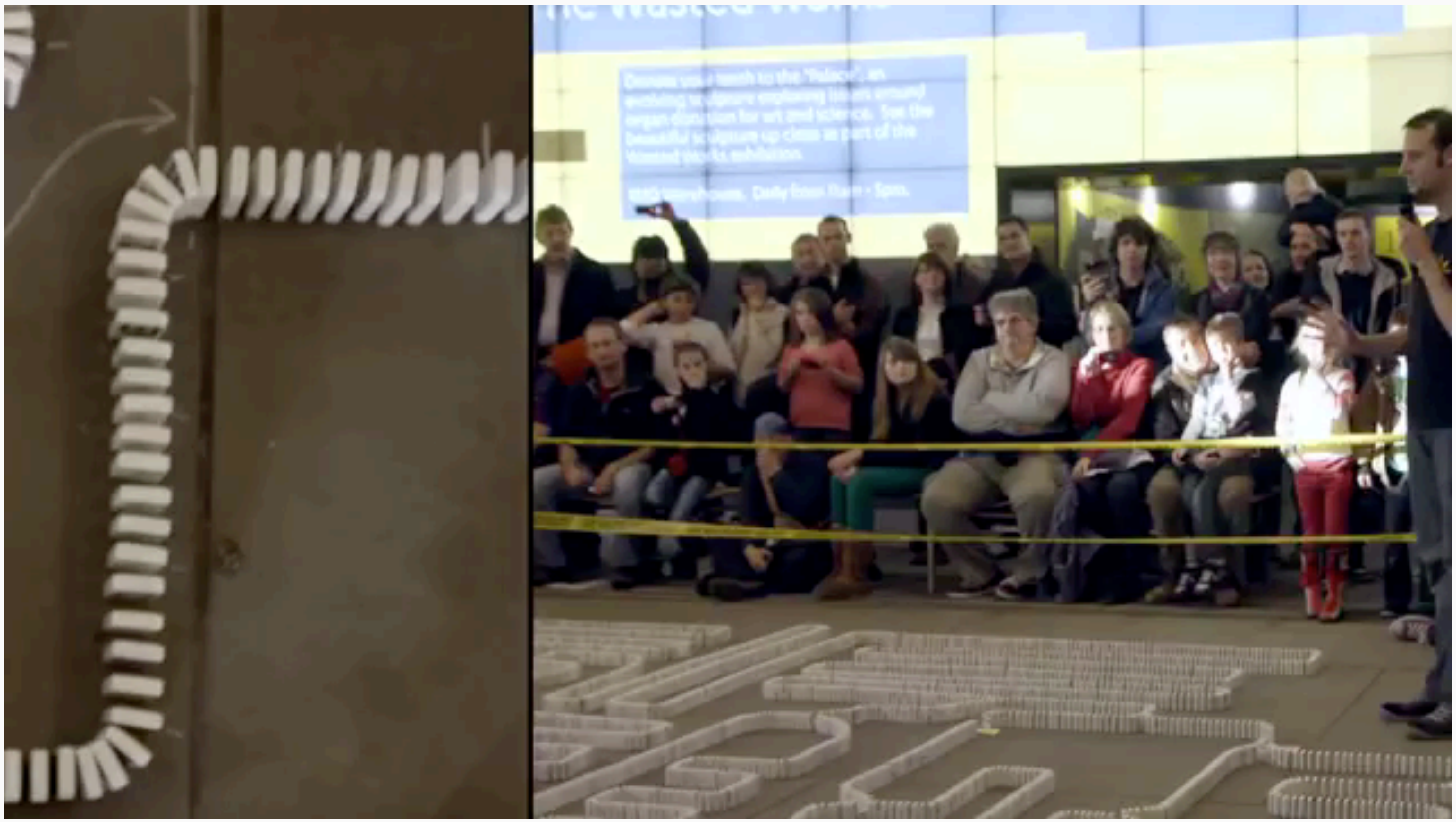
Arithmetic and Gates

A:



$$\begin{array}{r} 0 \\ +\ 0 \\ \hline 0\ 0 \end{array} \quad \begin{array}{r} 1 \\ +\ 0 \\ \hline 0\ 1 \end{array} \quad \begin{array}{r} 0 \\ +\ 1 \\ \hline 0\ 1 \end{array} \quad \begin{array}{rr} 1 & 1 \\ + & 1 \\ \hline 1 & 0 \end{array}$$

Binary Adder



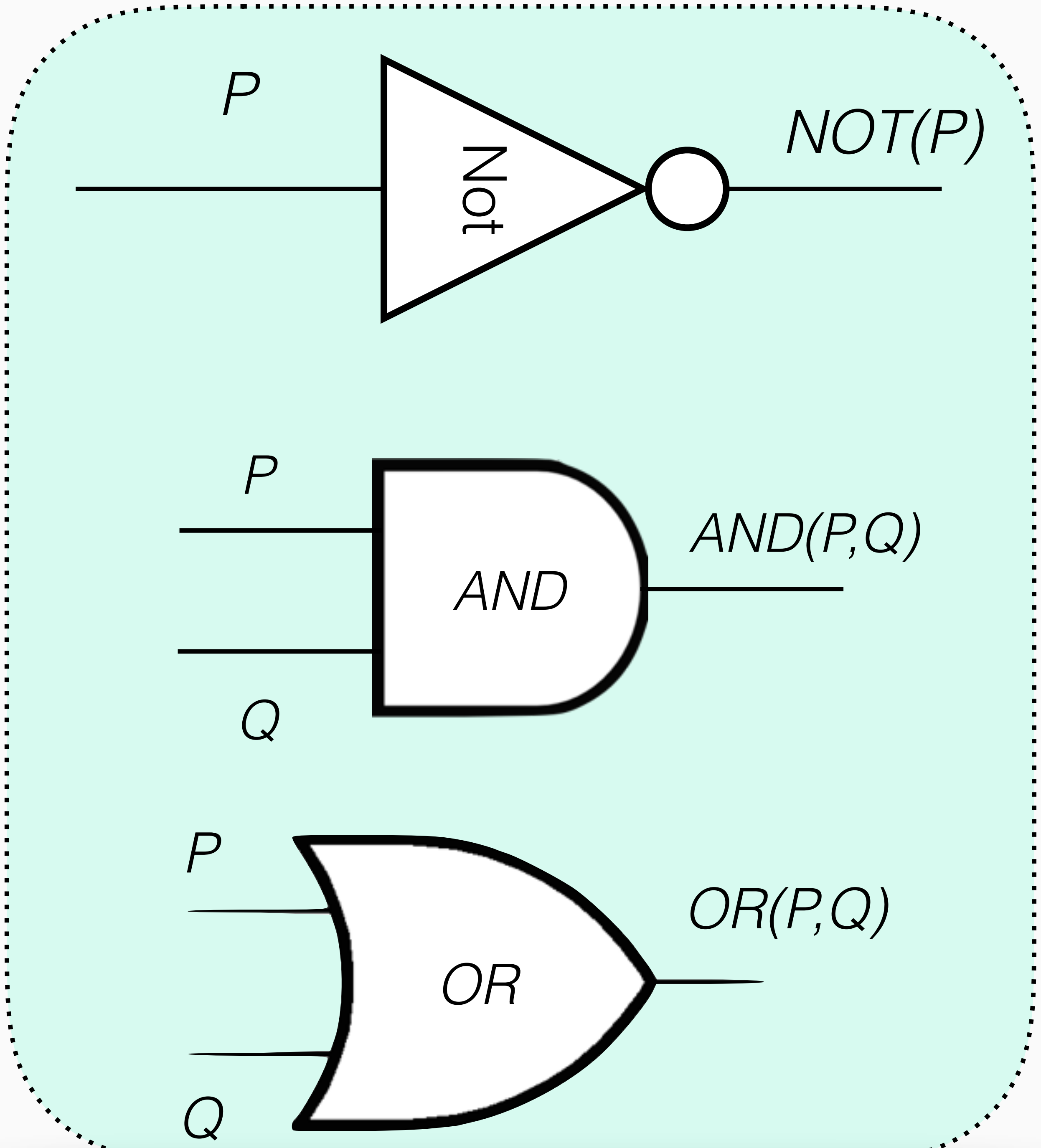
Truth Table to Formula

Idea: with a certain set of logical functions, we can represent all possible logical formulas!

Truth Table to Gate

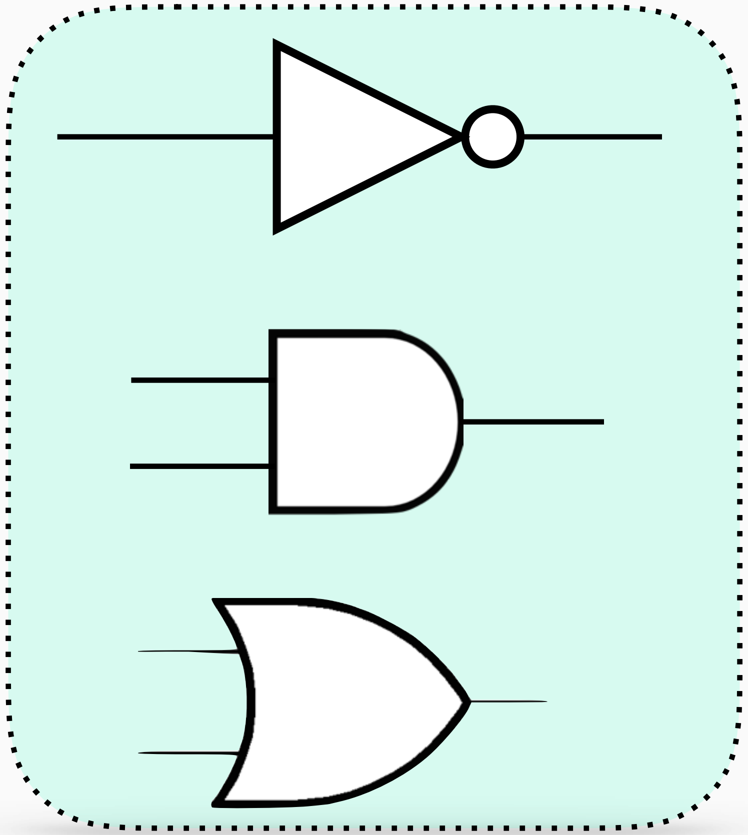
Idea: with a certain set of logical ~~functions~~ gates, we can represent all possible logical formulas!

Abstraction:



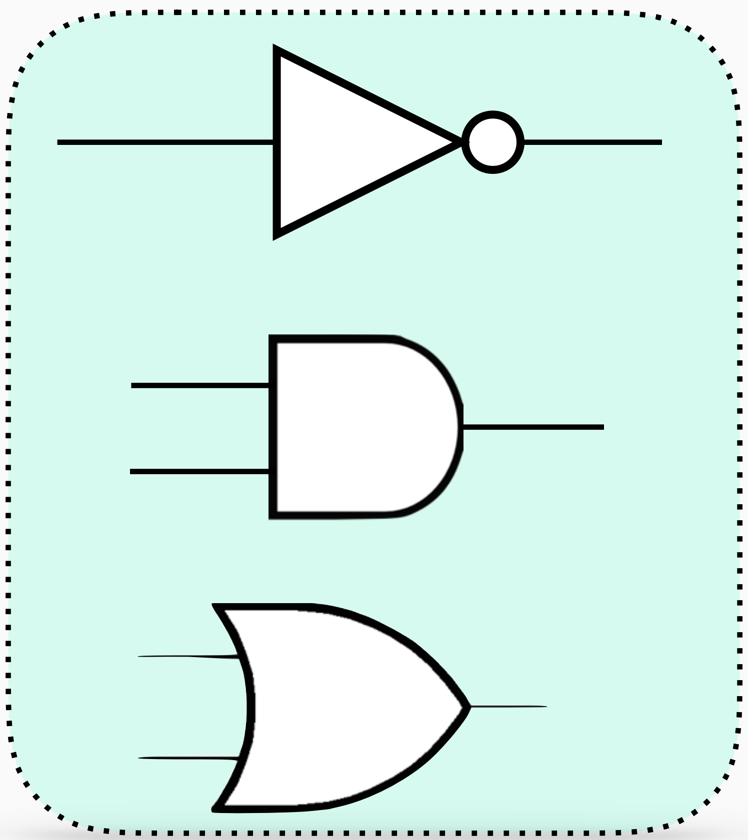
all possible logical formulas!

Abstraction:



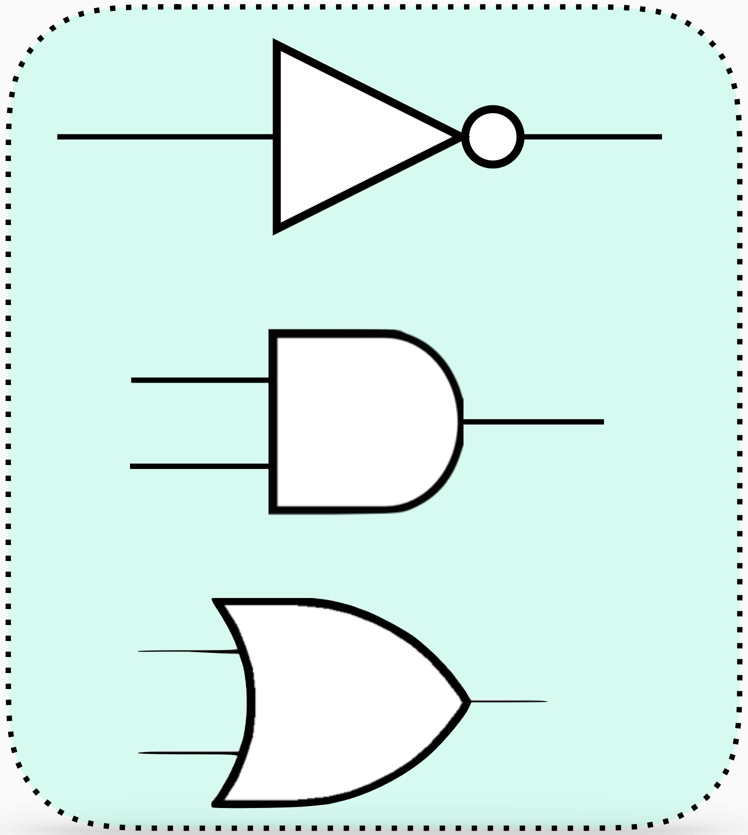
Low level programs

Abstraction:



→ Low level programs → High level programs

Abstraction:



→ Low level programs → High level programs

=

Your ideas!

State Machines

State Machines: Example

Suppose we want a blinking christmas light:

State Machines: Example

Suppose we want a blinking christmas light:

O = was the light on a second ago?

N = is the light on now?

State Machines: Example

Suppose we want a blinking christmas light:

oldLight = was the light on a second ago?

newLight = is the light on now?

State Machines: Example

Suppose we want a blinking christmas light:

oldLight = was the light on a second ago?

newLight = is the light on now?

Q: How can we write the value of newLight in terms of oldLight?

State Machines: Example

Suppose we want a blinking christmas light:

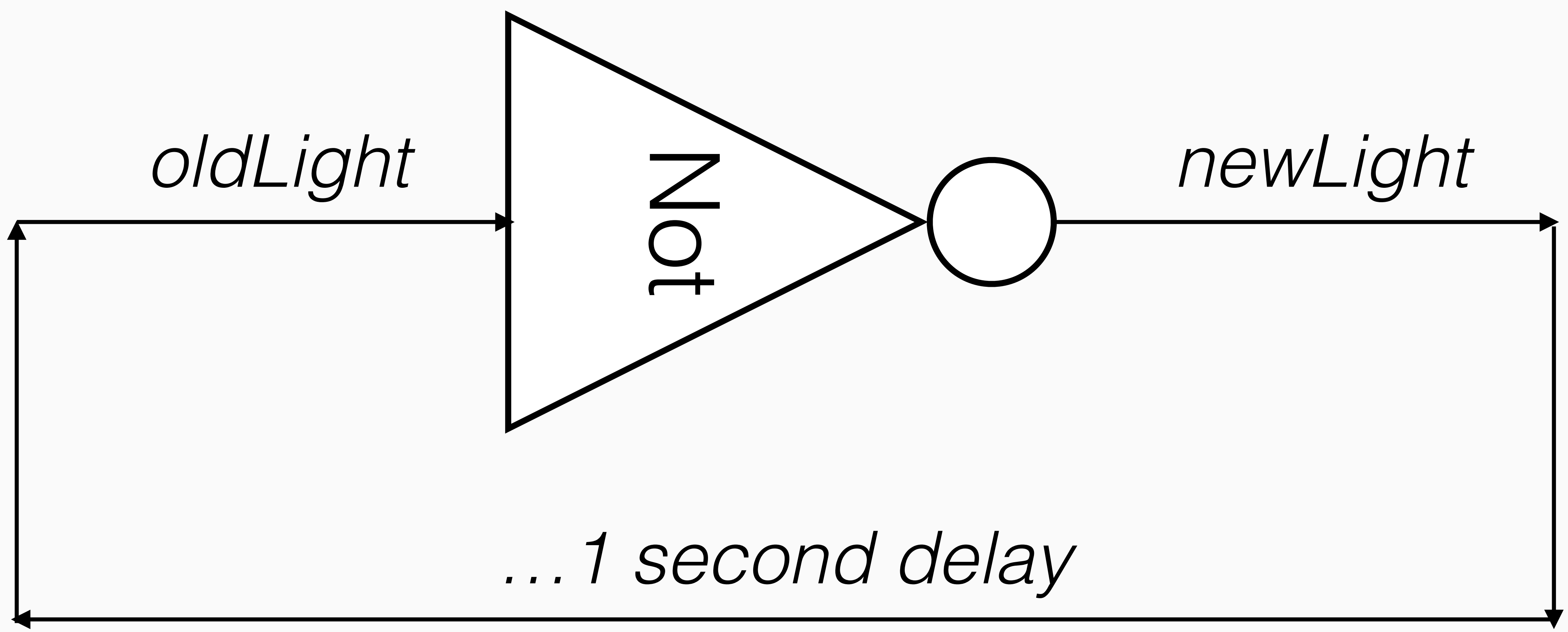
oldLight = was the light on a second ago?

newLight = is the light on now?

A: *newLight = NOT(oldLight)*

State Machines: Example

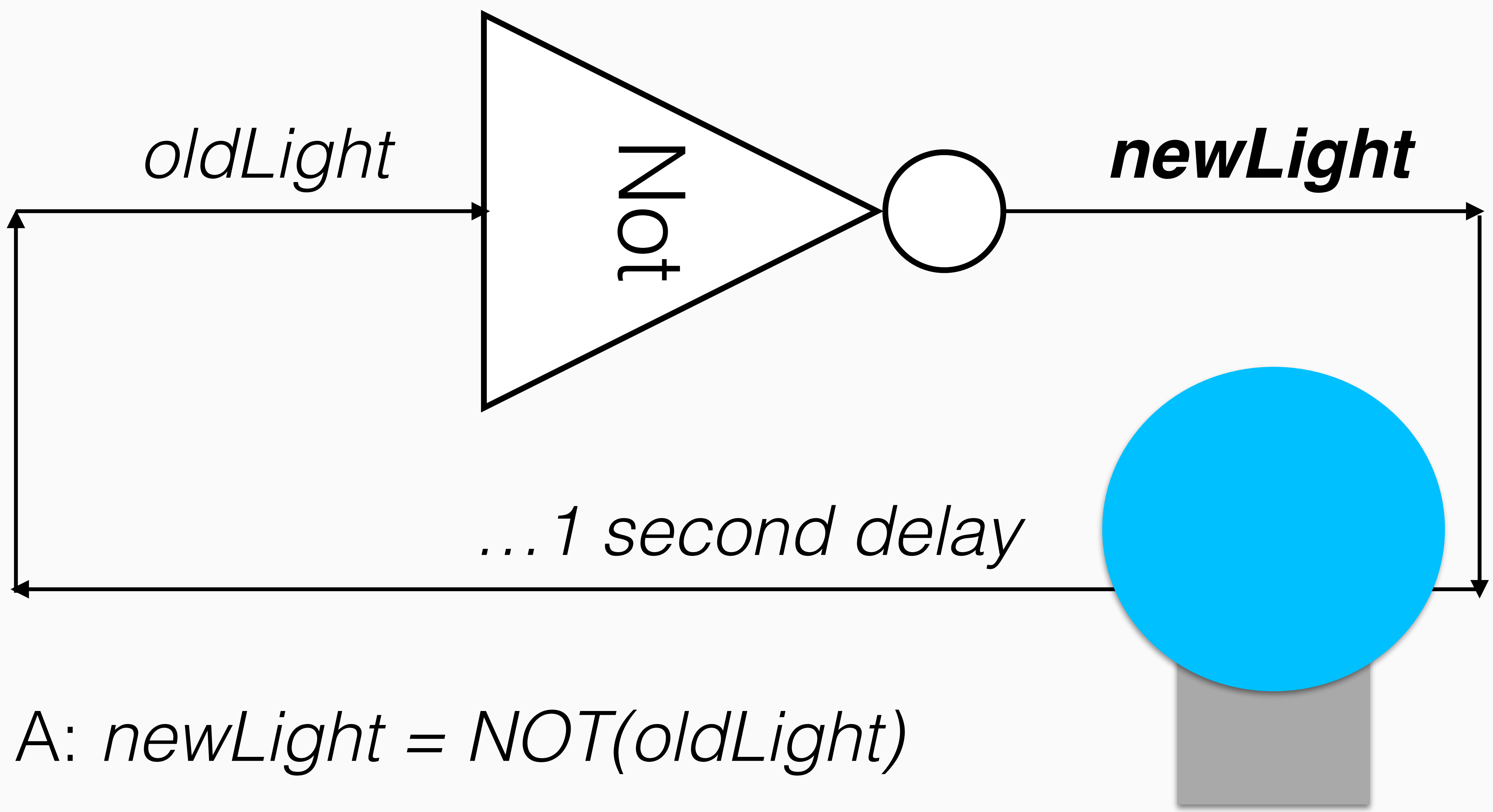
Suppose we want a blinking christmas light:



A: *newLight = NOT(oldLight)*

State Machines: Example

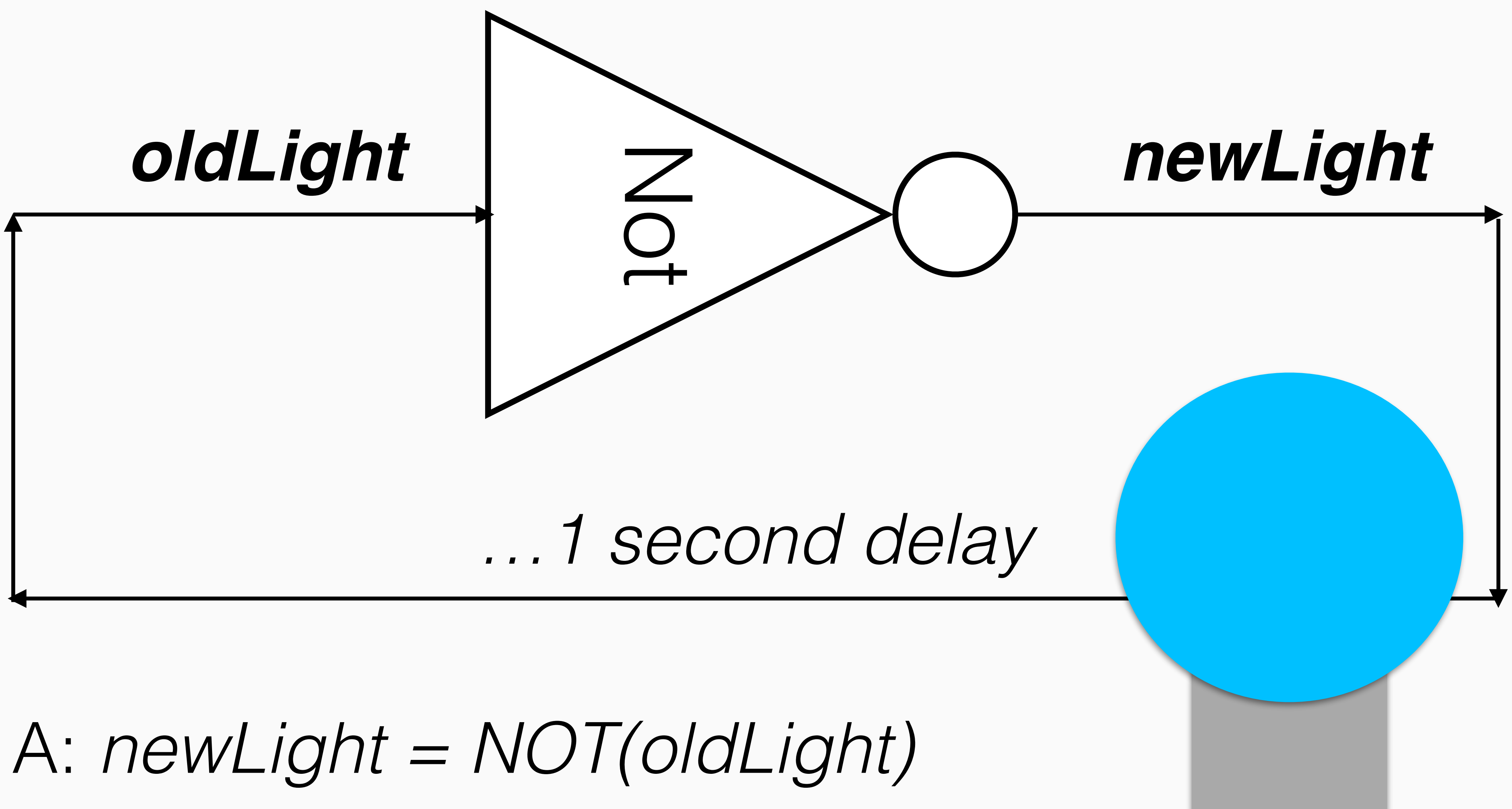
Suppose we want a blinking christmas light:



A: *newLight = NOT(oldLight)*

State Machines: Example

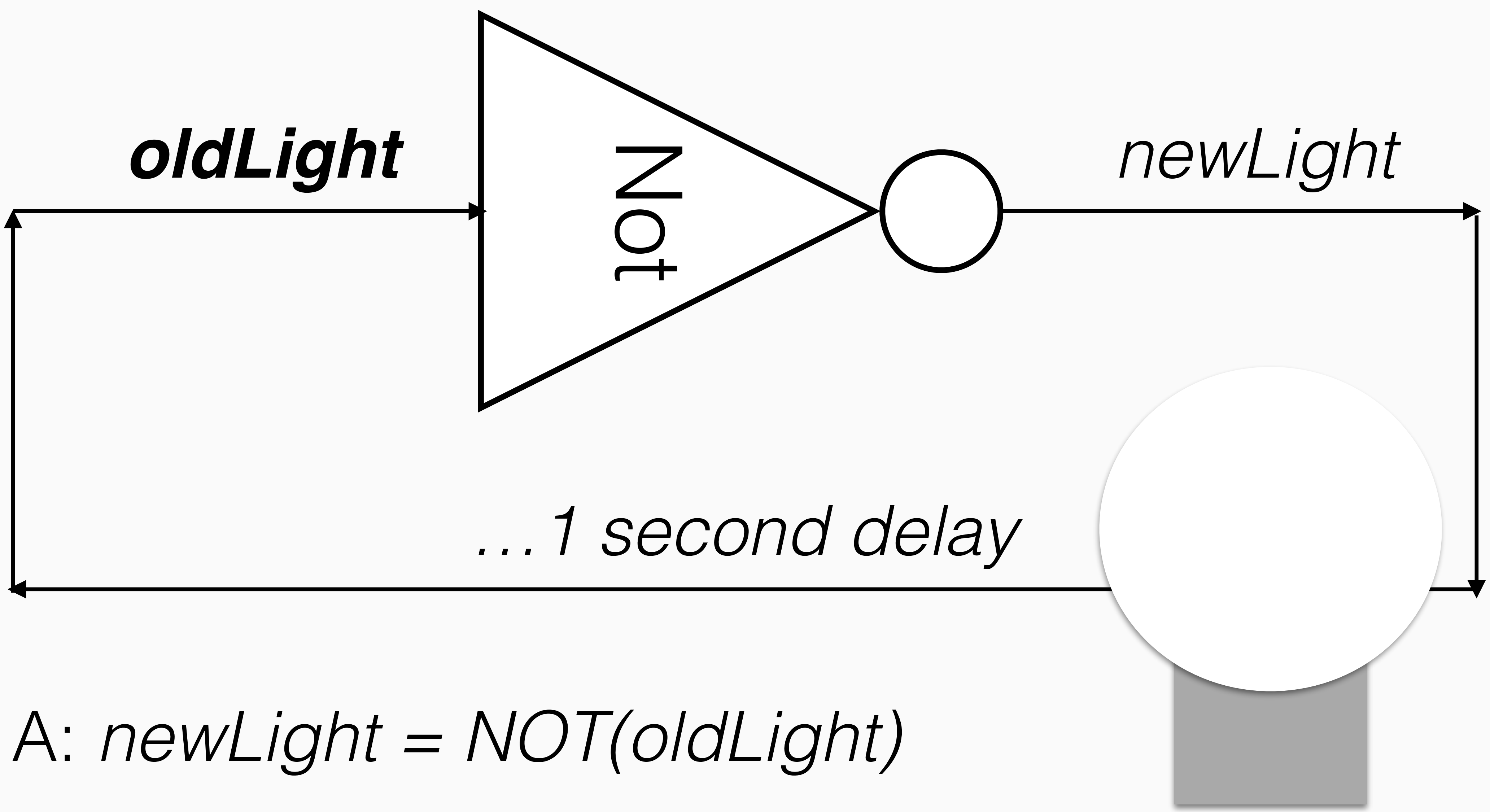
Suppose we want a blinking christmas light:



A: *newLight = NOT(oldLight)*

State Machines: Example

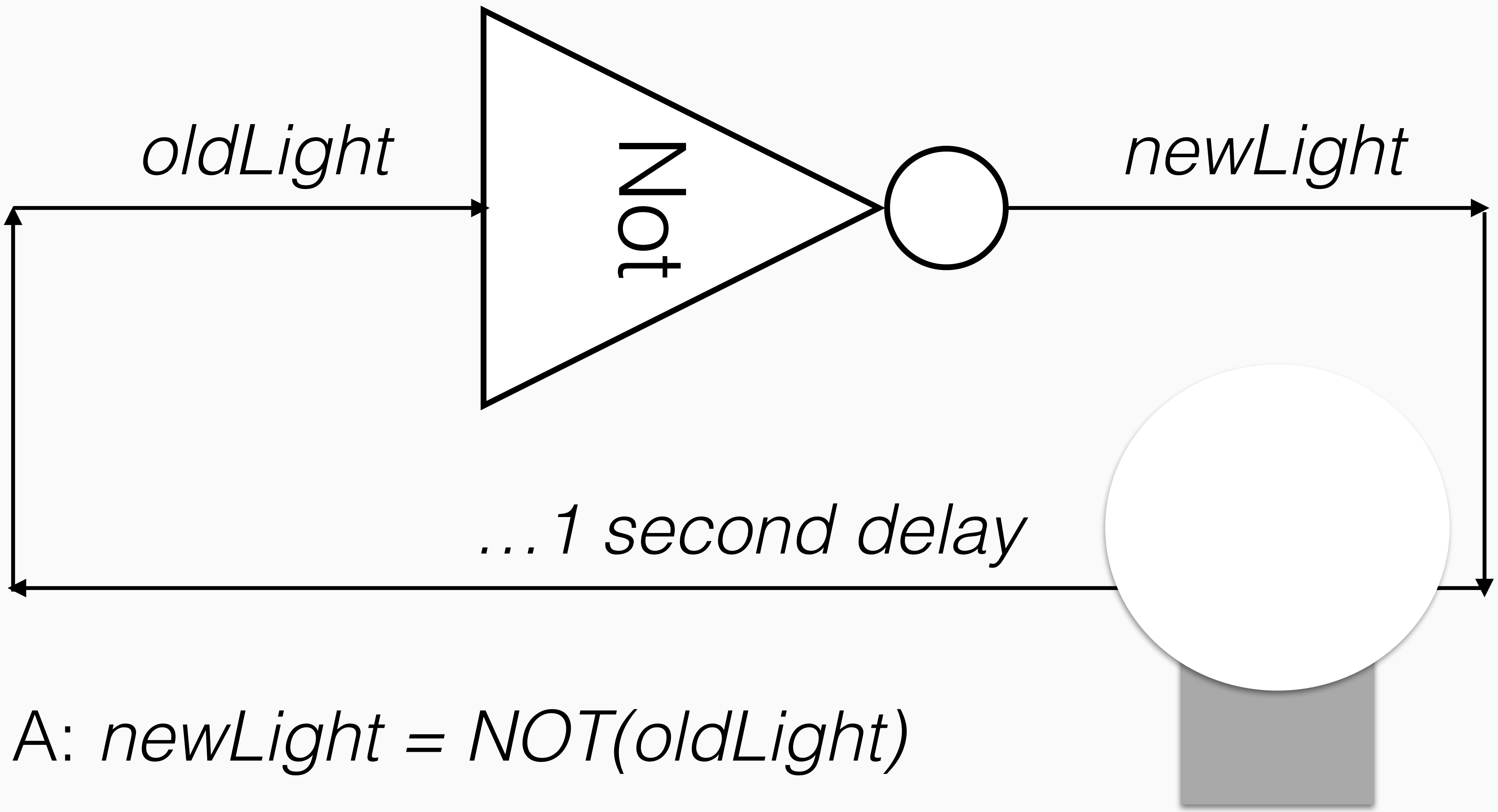
Suppose we want a blinking christmas light:



A: *newLight = NOT(oldLight)*

State Machines: Example

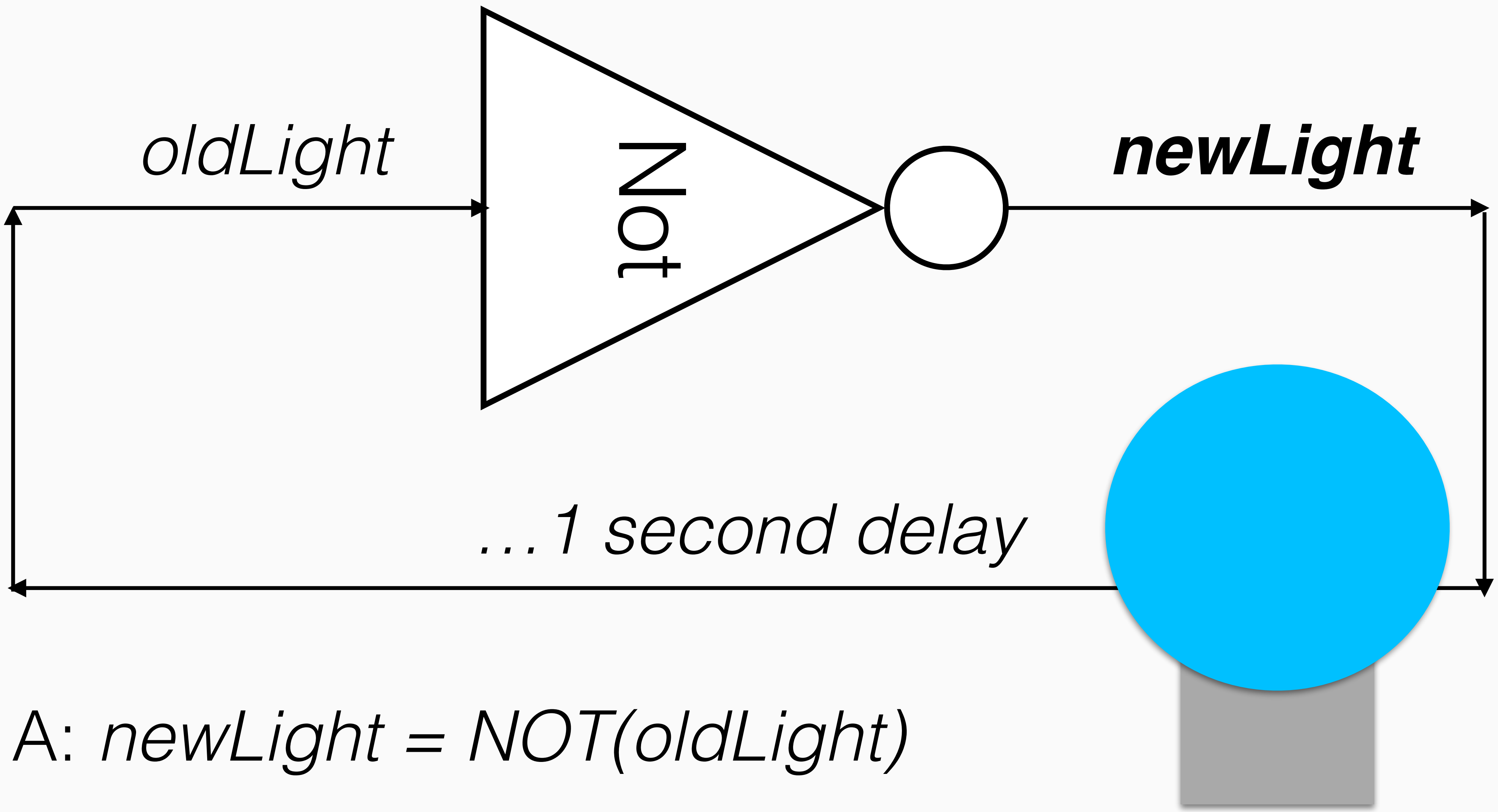
Suppose we want a blinking christmas light:



A: *newLight = NOT(oldLight)*

State Machines: Example

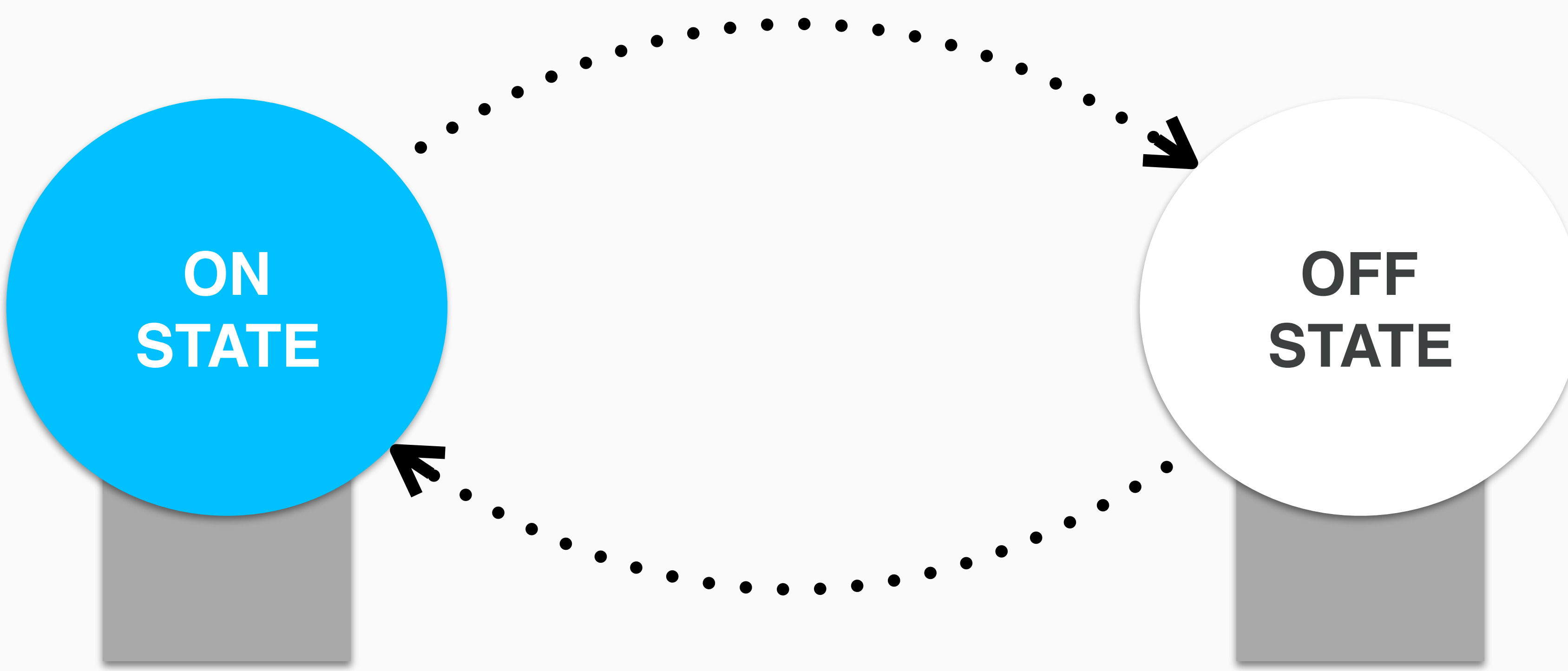
Suppose we want a blinking christmas light:



A: *newLight = NOT(oldLight)*

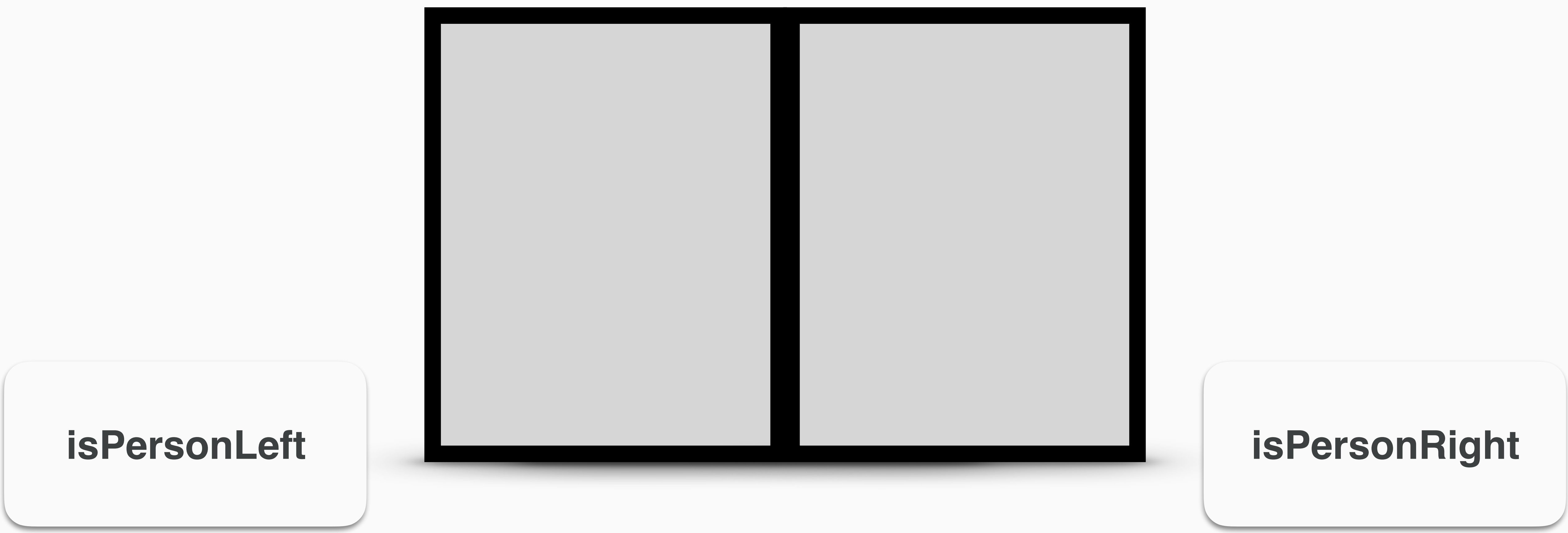
State Machines: Example

Suppose we want a blinking christmas light:

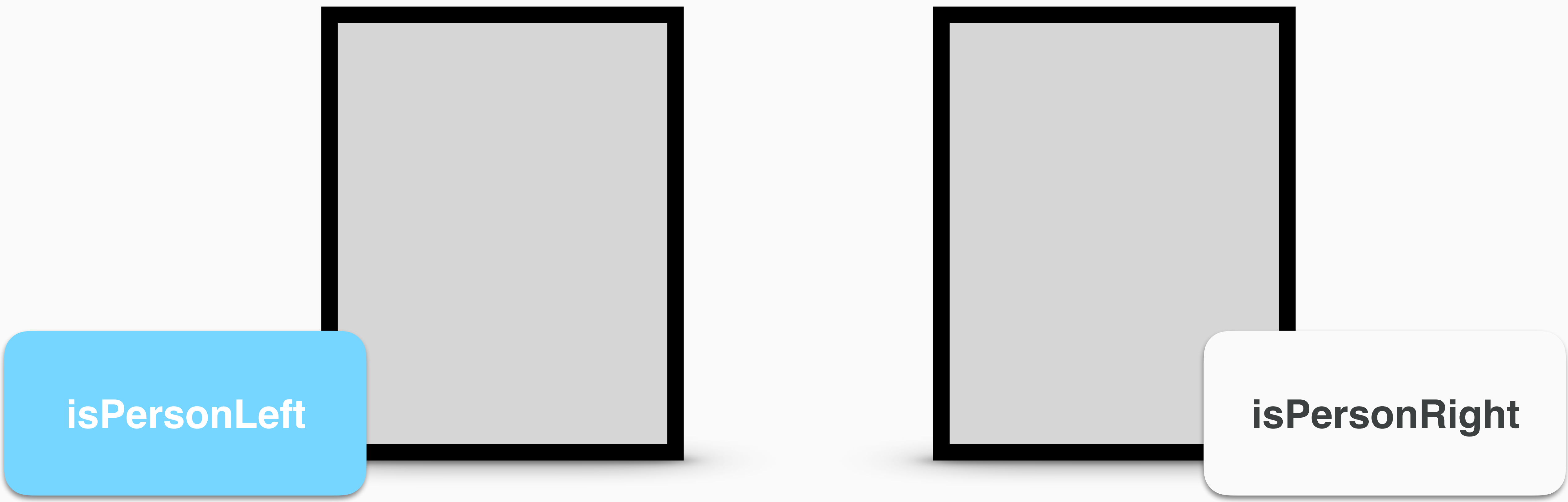


A More Complicated Example: Sliding Doors

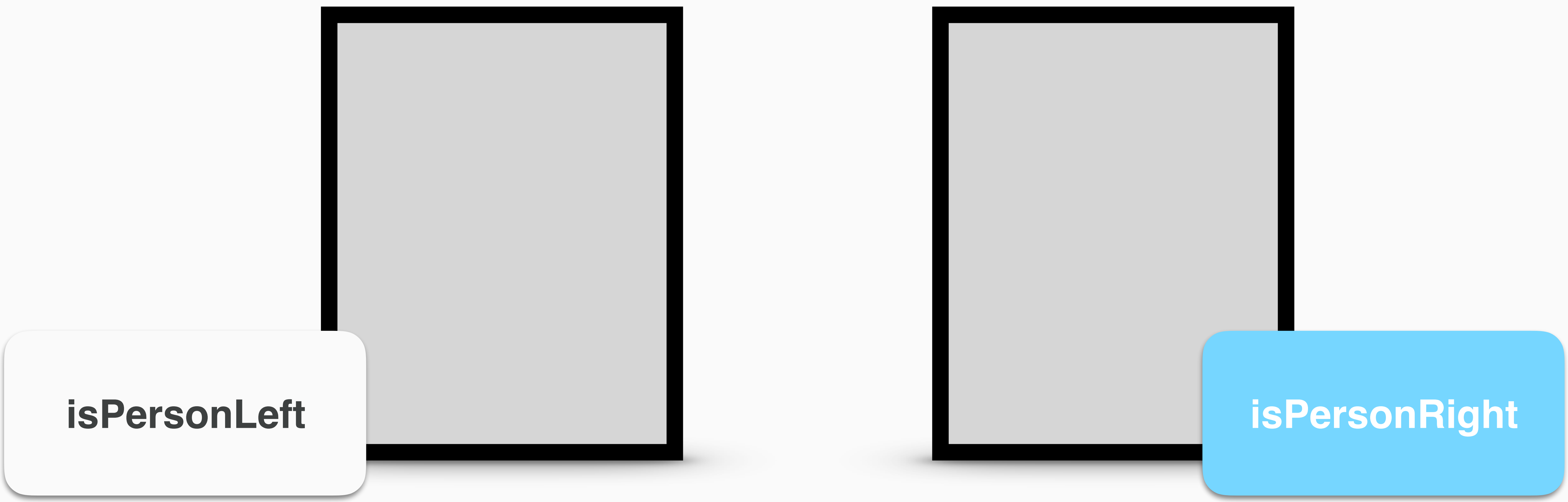
A More Complicated Example: Sliding Doors



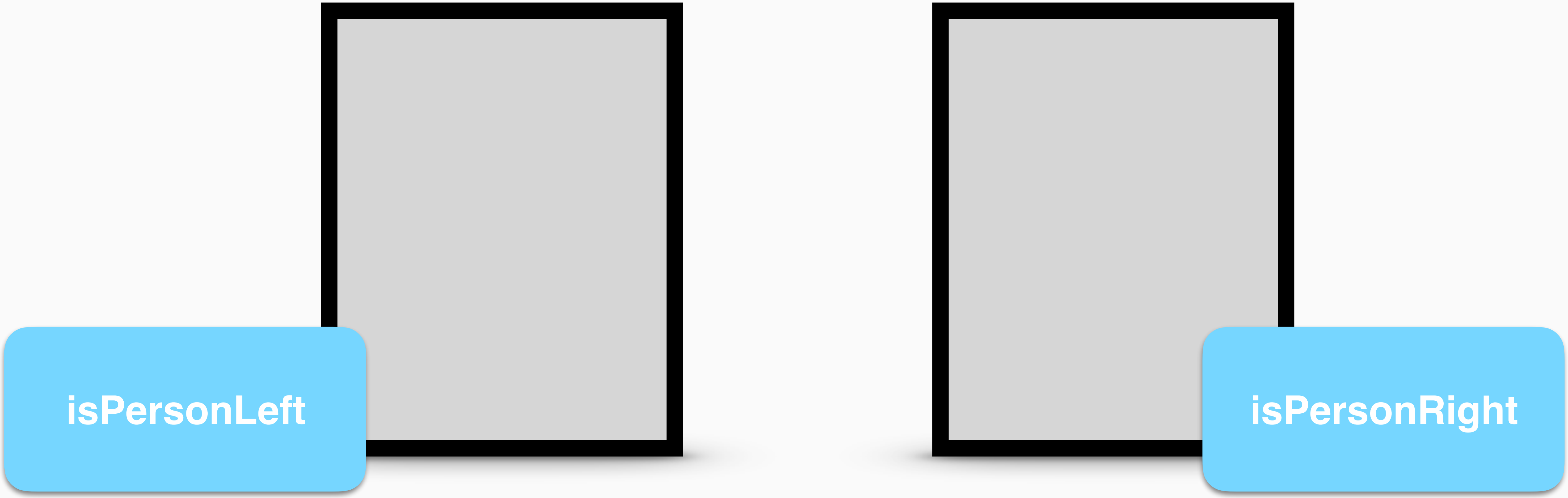
A More Complicated Example: Sliding Doors



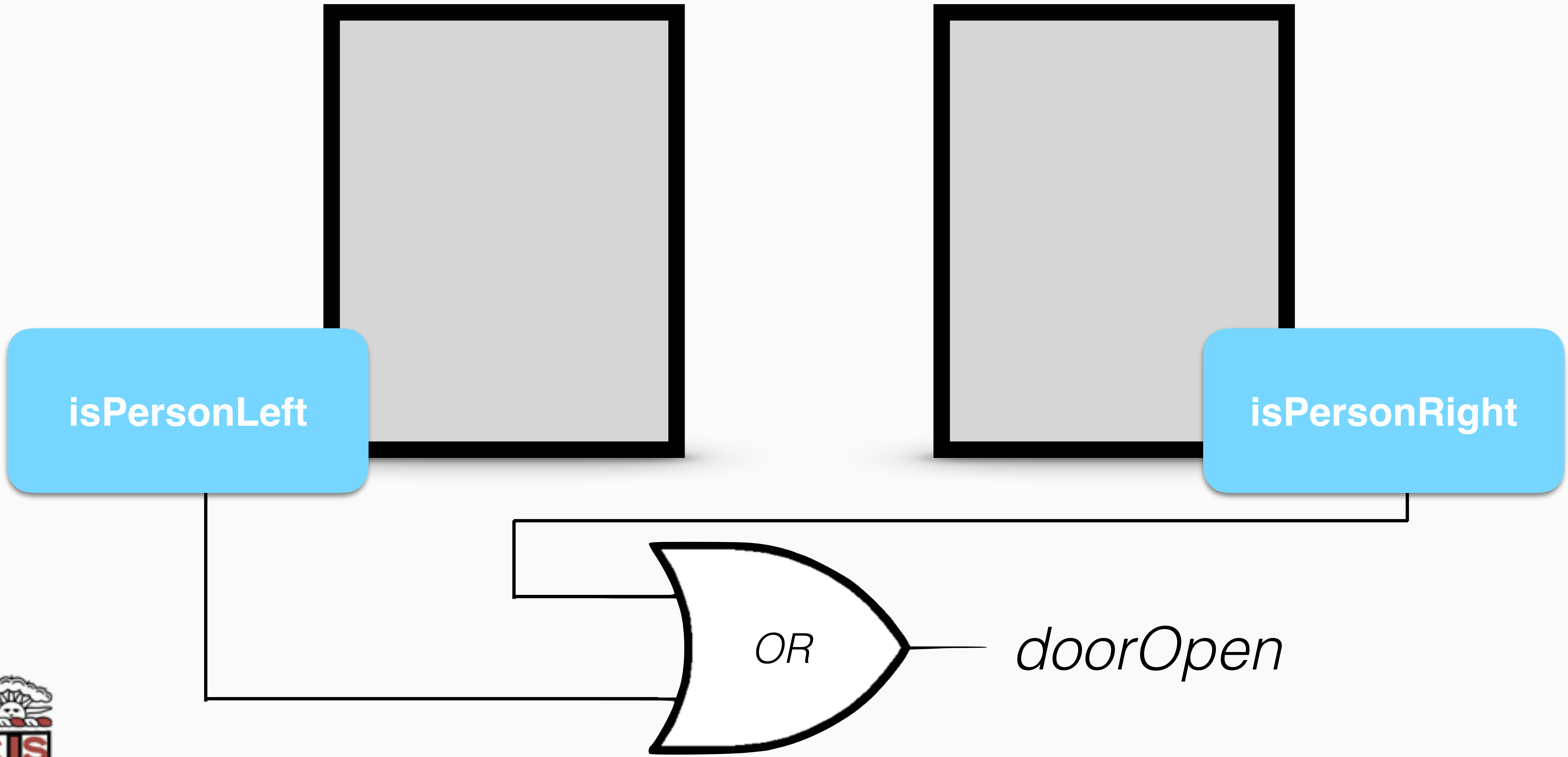
A More Complicated Example: Sliding Doors



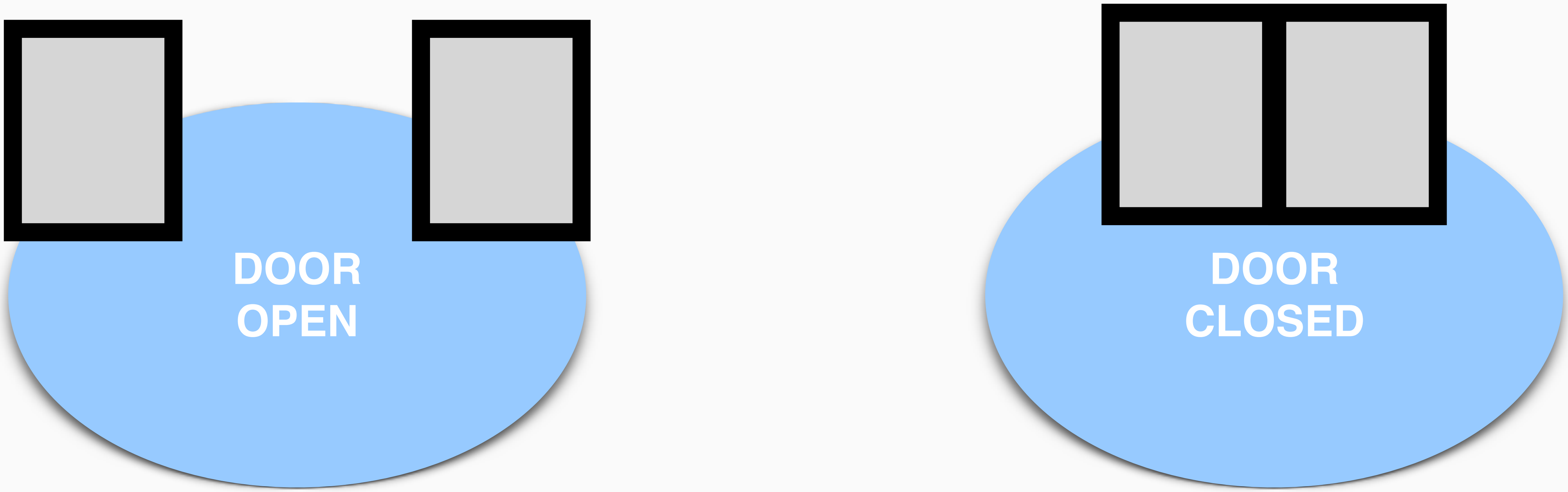
A More Complicated Example: Sliding Doors



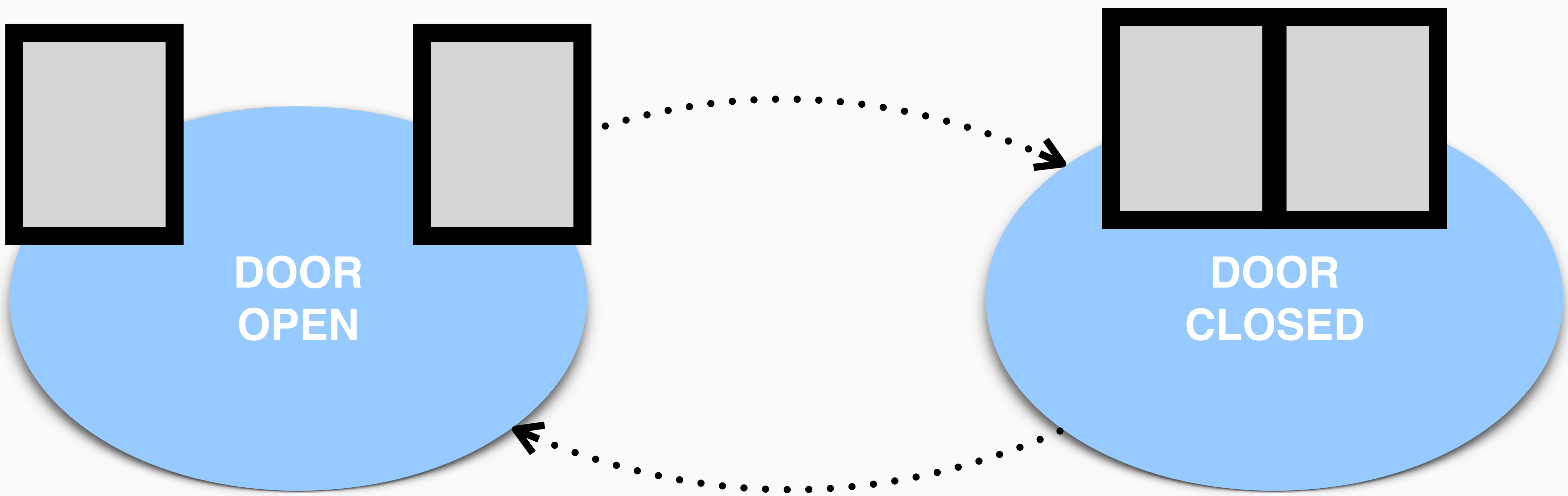
A More Complicated Example: Sliding Doors



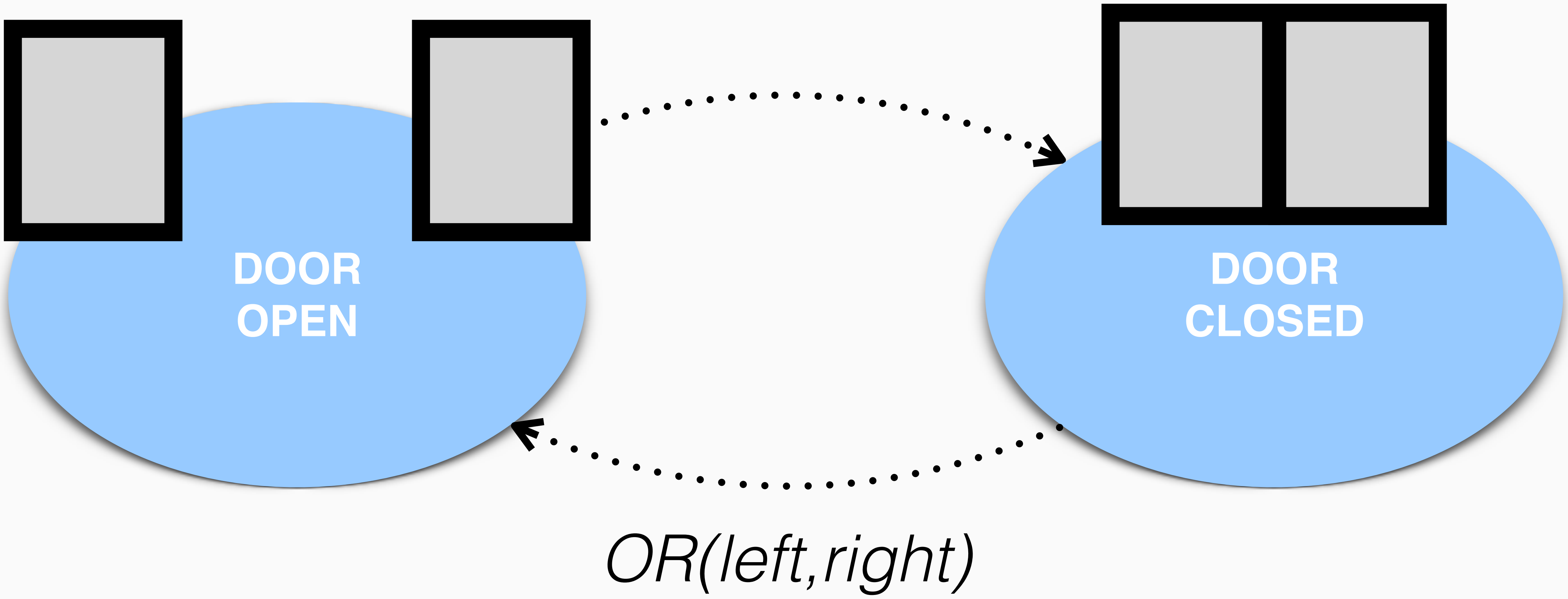
A More Complicated Example: Sliding Doors



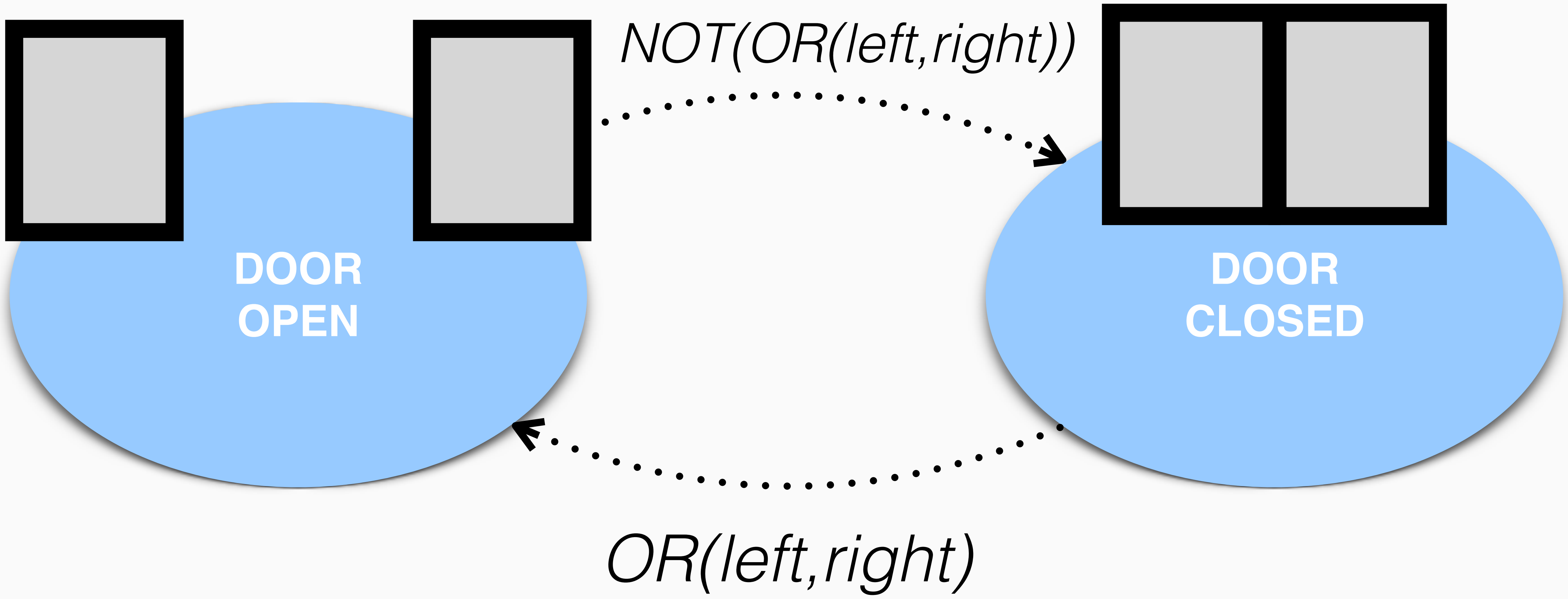
A More Complicated Example: Sliding Doors



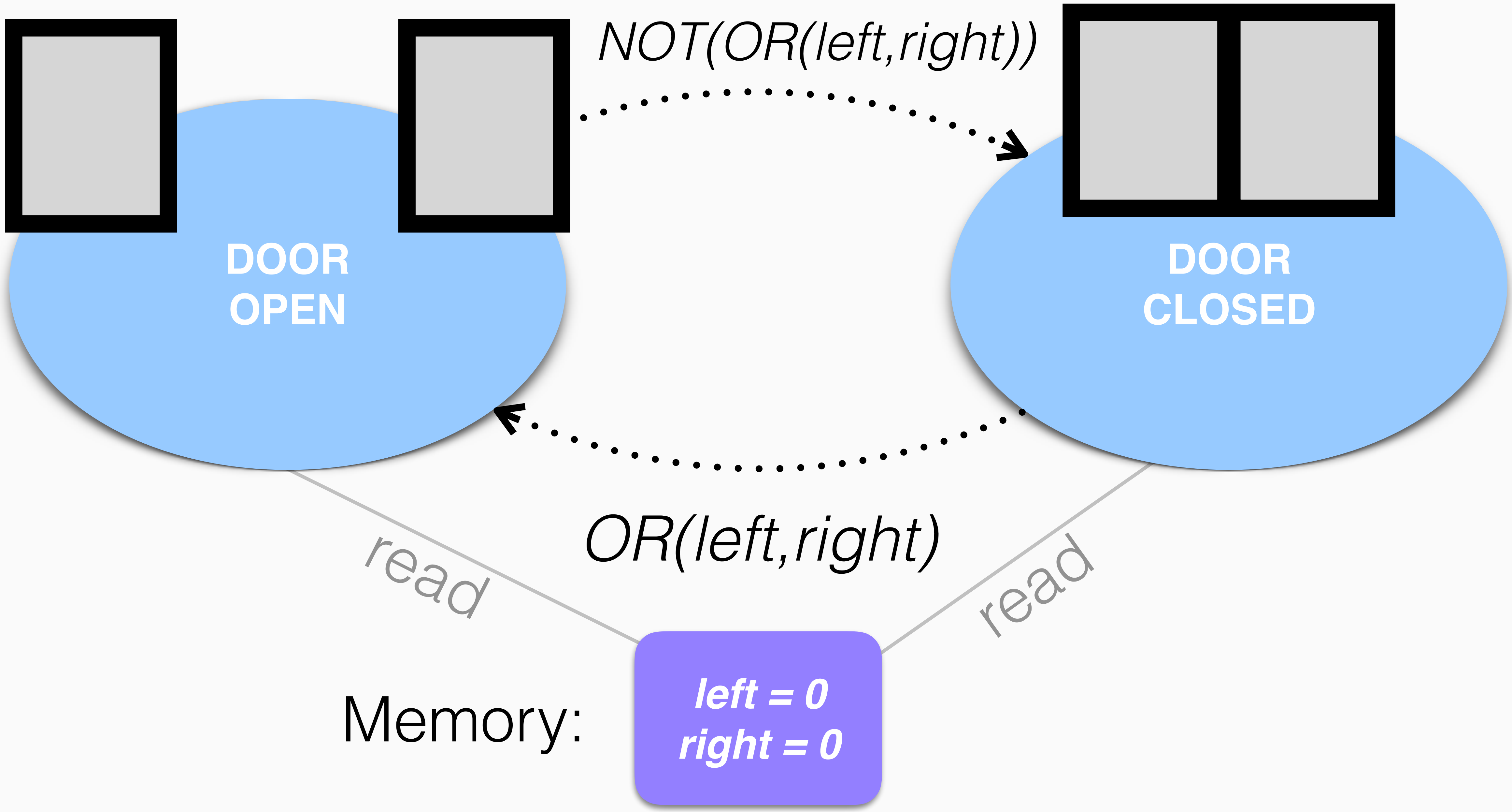
A More Complicated Example: Sliding Doors



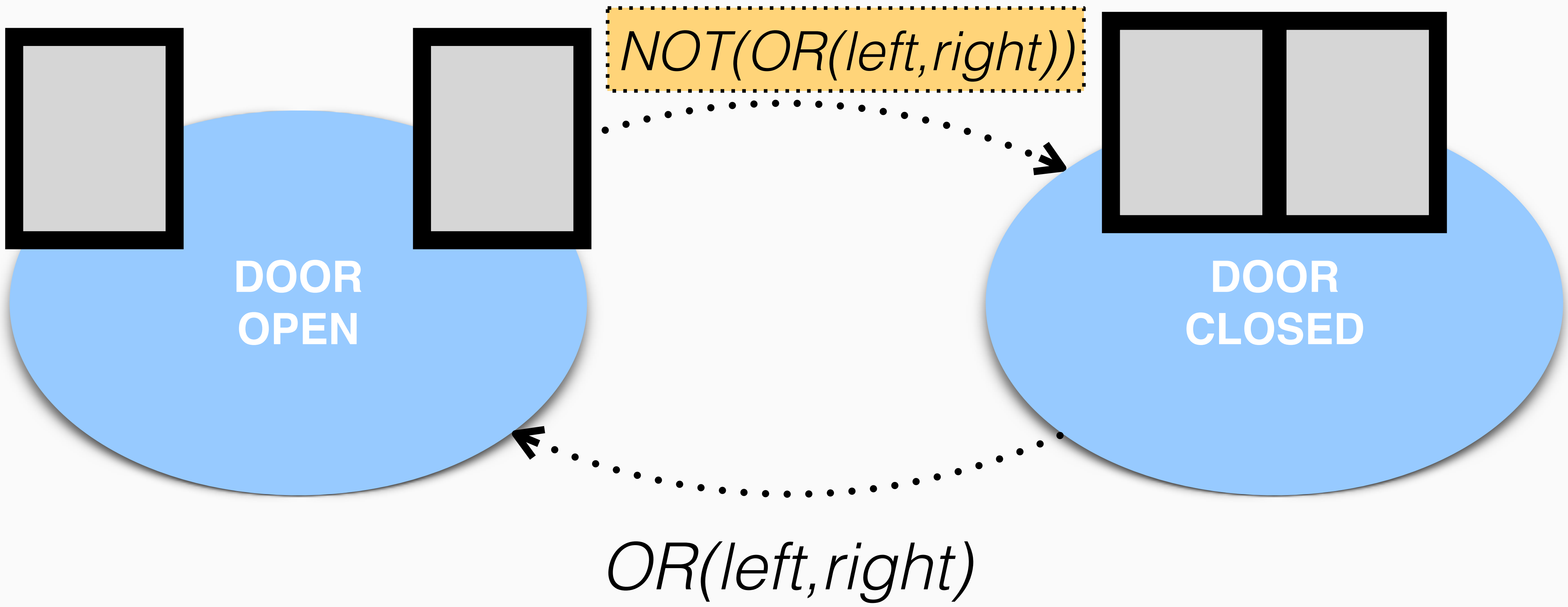
A More Complicated Example: Sliding Doors



A More Complicated Example: Sliding Doors

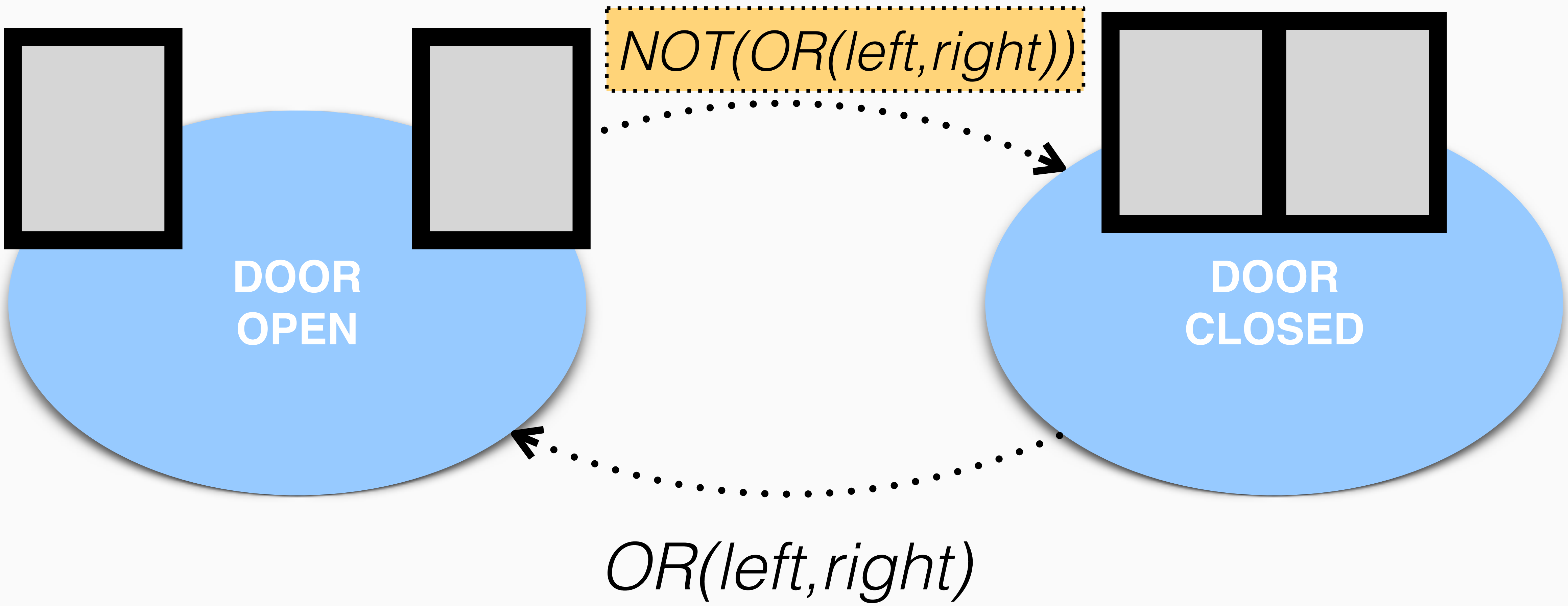


A More Complicated Example: Sliding Doors



Q: How else can we write this with logic?

A More Complicated Example: Sliding Doors



Q: How else can we write this with logic?

A: *AND(NOT(left),NOT(right))*

Side Note: DeMorgan's Law

AND(NOT(left),NOT(right))

=

NOT(OR(left,right))

Side Note: DeMorgan's Law

P	*Q*	*AND(*	*NOT(P)*	*NOT(Q))*
1	1	**0**	0	0
1	0	**0**	0	1
0	1	**0**	1	0
0	0	**1**	1	1

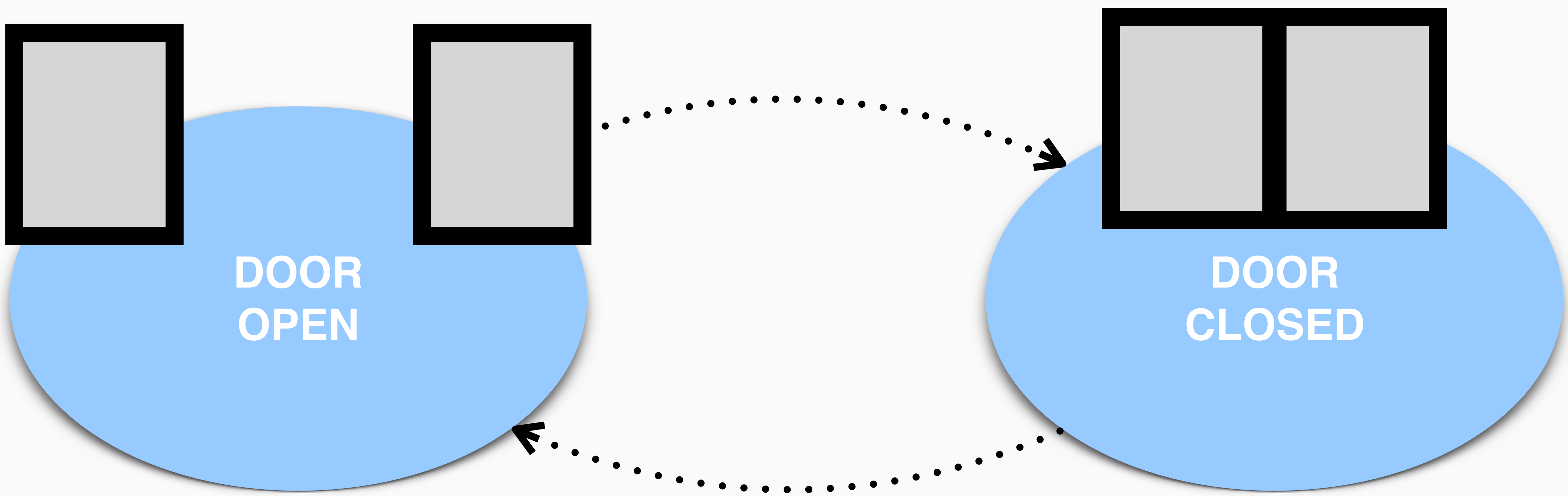
P	*Q*	*NOT(*	*OR(P,Q))*
1	1	**0**	1
1	0	**0**	1
0	1	**0**	1
0	0	**1**	0

Side Note: DeMorgan’s Law

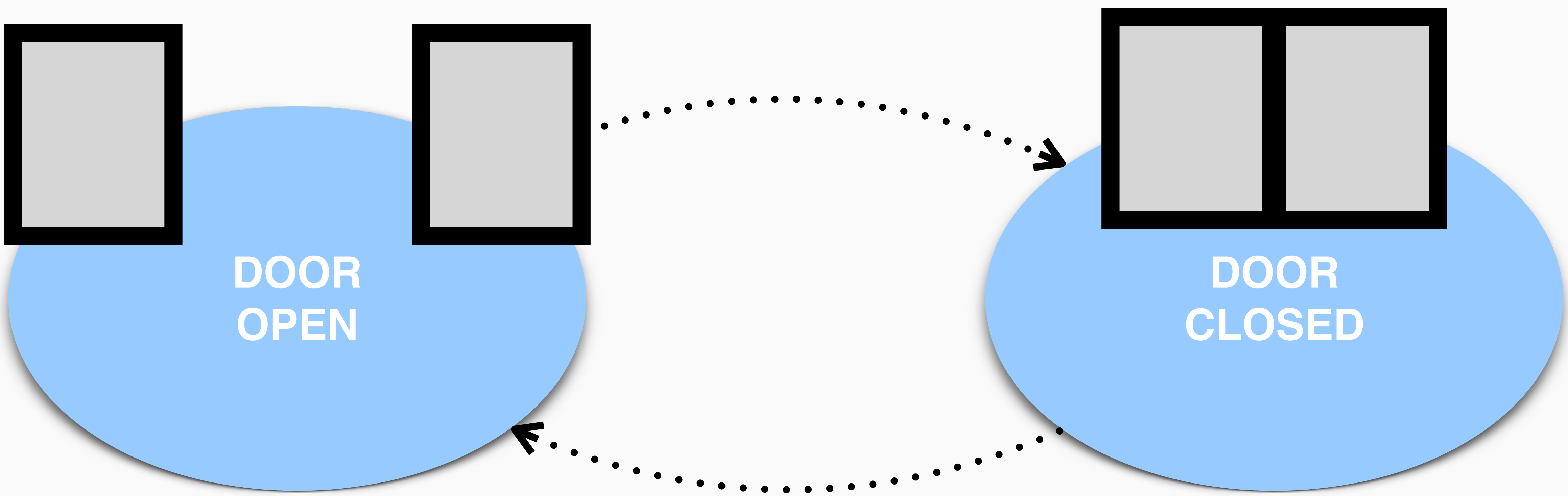
P	*Q*	*AND(*	*NOT(P)*	*NOT(Q))*
1	1	**0**	0	0
1	0	**0**	0	1
0	1	**0**	1	0
0	0	**1**	1	1

P	*Q*	*NOT(*	*OR(P,Q))*
1	1	**0**	1
1	0	**0**	1
0	1	**0**	1
0	0	**1**	0

State Machines: Sliding Doors



State Machines: Sliding Doors

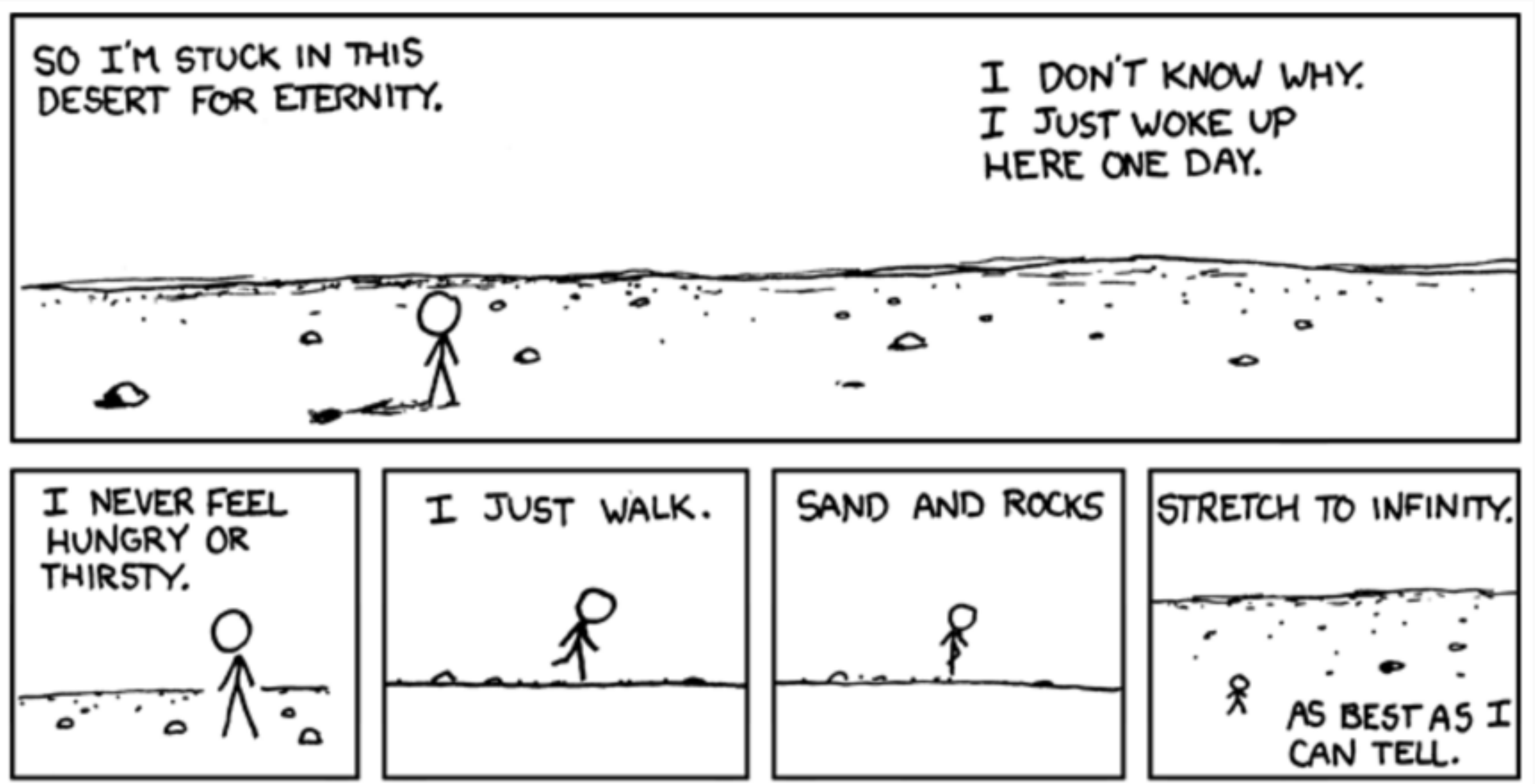


Sensors! Input! Reading from Memory!

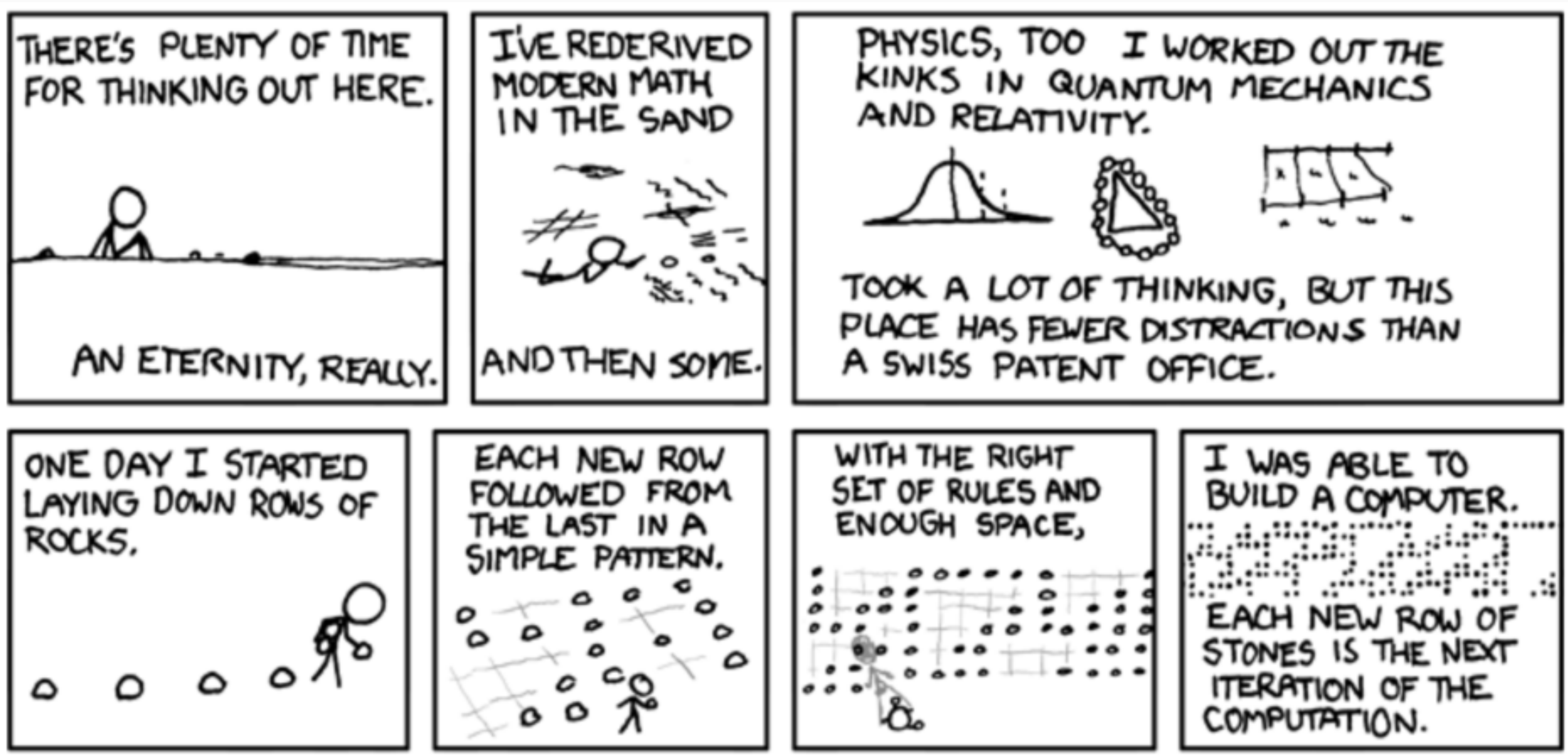
The Computer

- Internal states (**memory**)
- Complex **logic** relating bits/information
- Mechanism for setting bits (**input**)
- Mechanism for displaying bits (**output**)

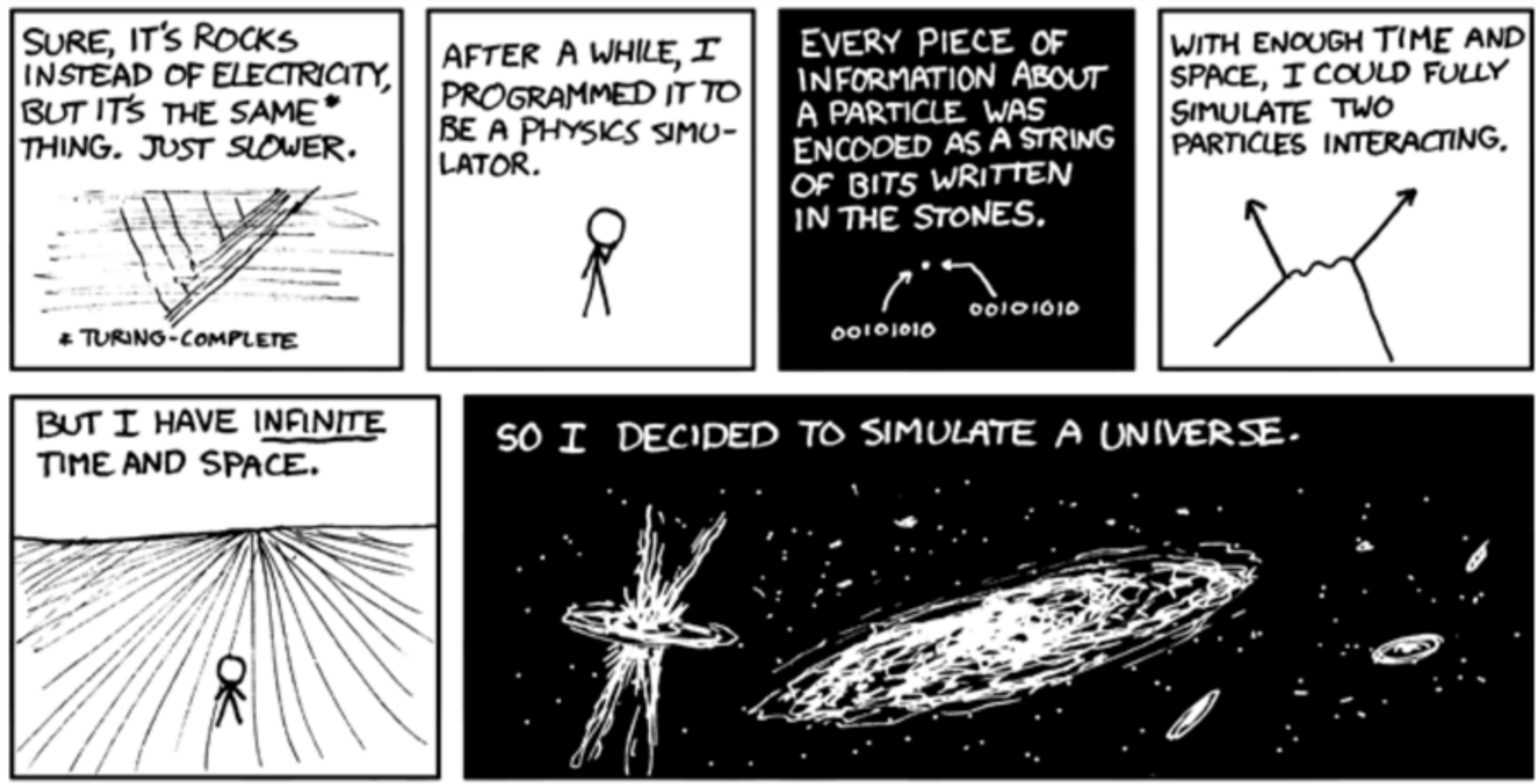
The Computer



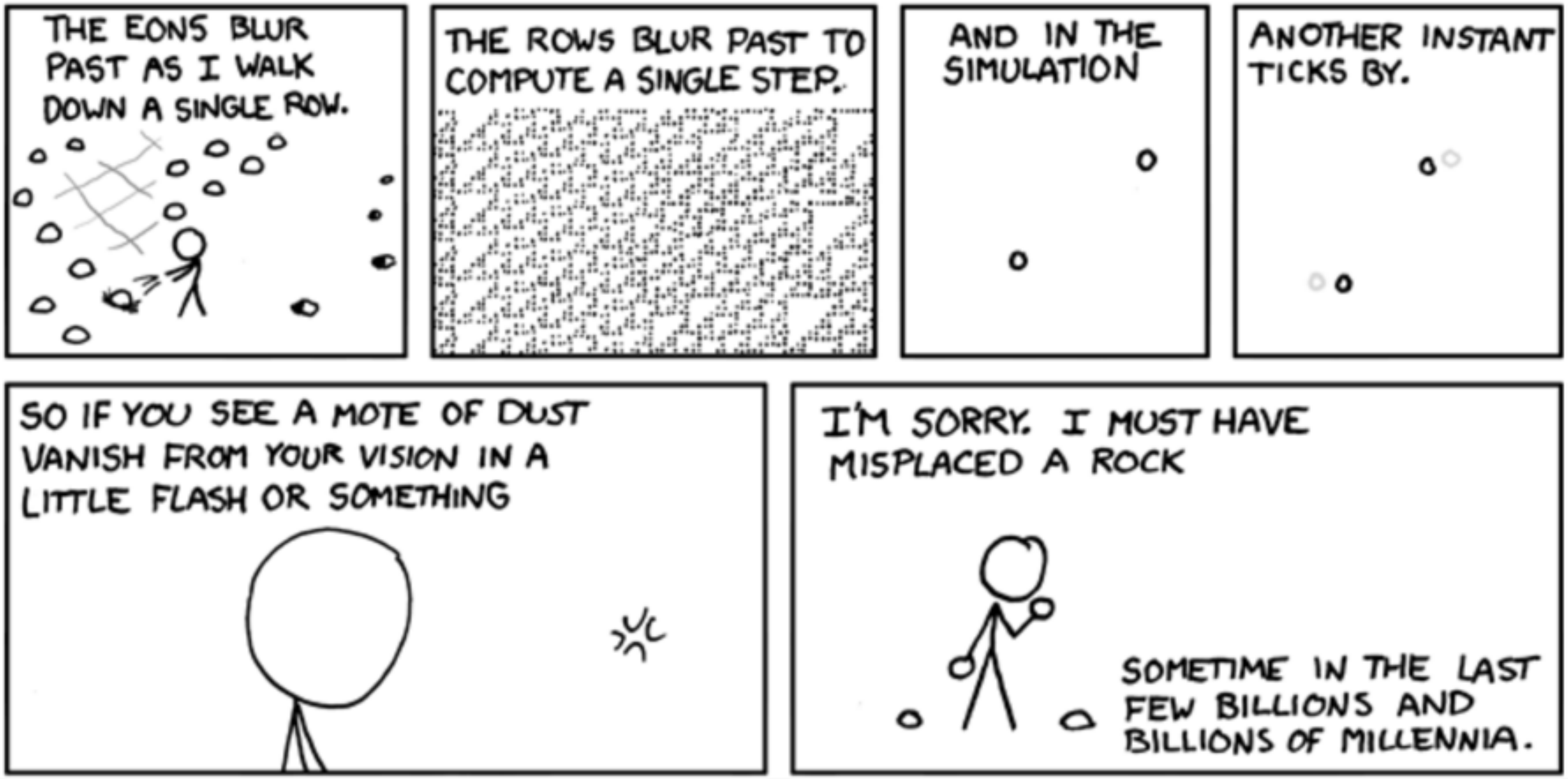
The Computer



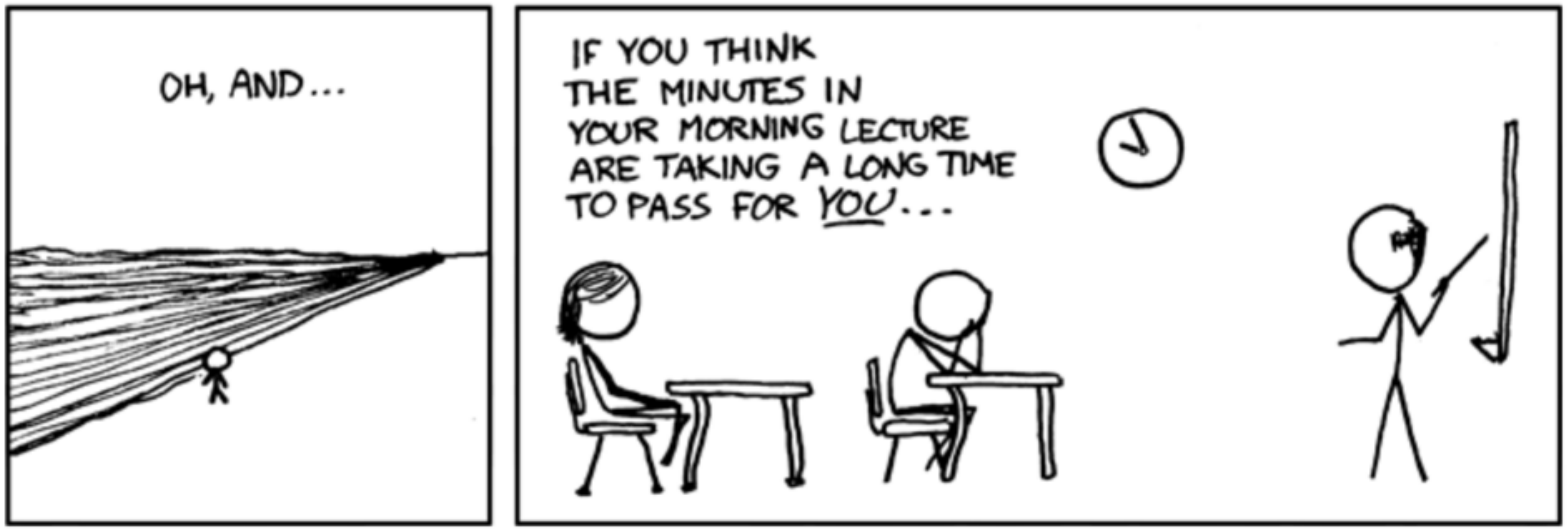
The Computer



The Computer



The Computer



https://xkcd.com/505/

The Church-Turing Thesis

Alonzo Church

Alan Turing

The Church-Turing Thesis

"There is only one notion of computation"

The Church-Turing Thesis

“There is only one notion of computation”

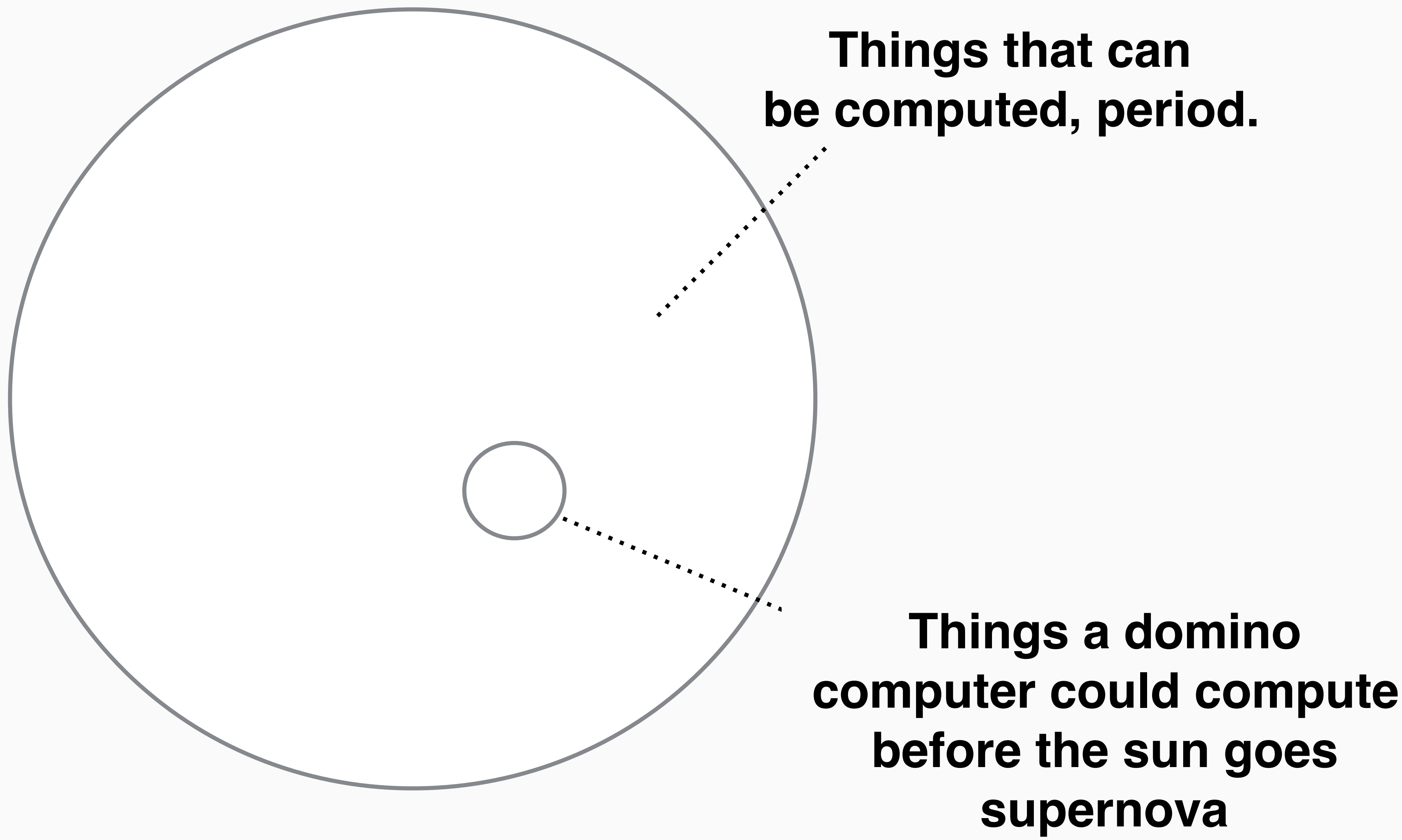
(and anything with these properties is doing it)

- Internal states
- Complex logic relating bits
- Mechanism for setting bits (input)
- Mechanism for displaying bits (output)

The Church-Turing Thesis

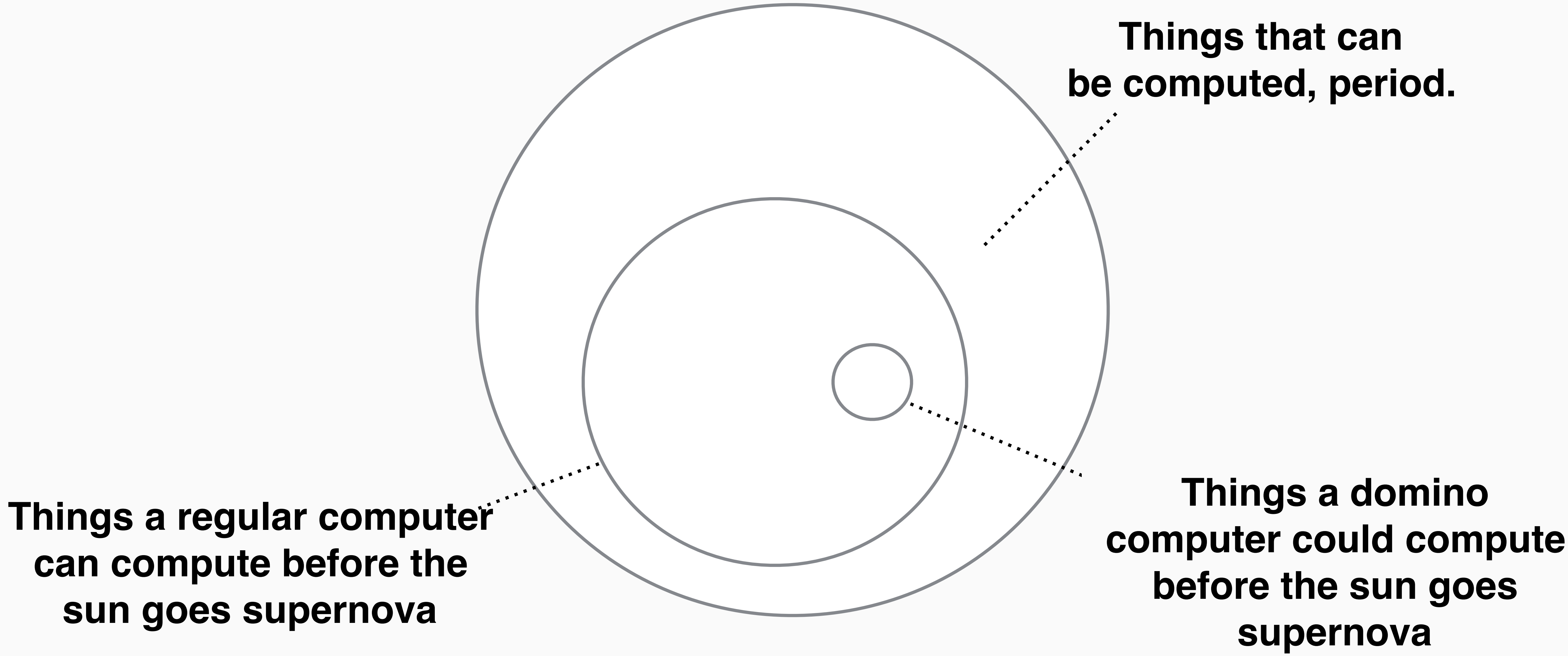
“There is only one notion of computation”

The Church-Turing Thesis



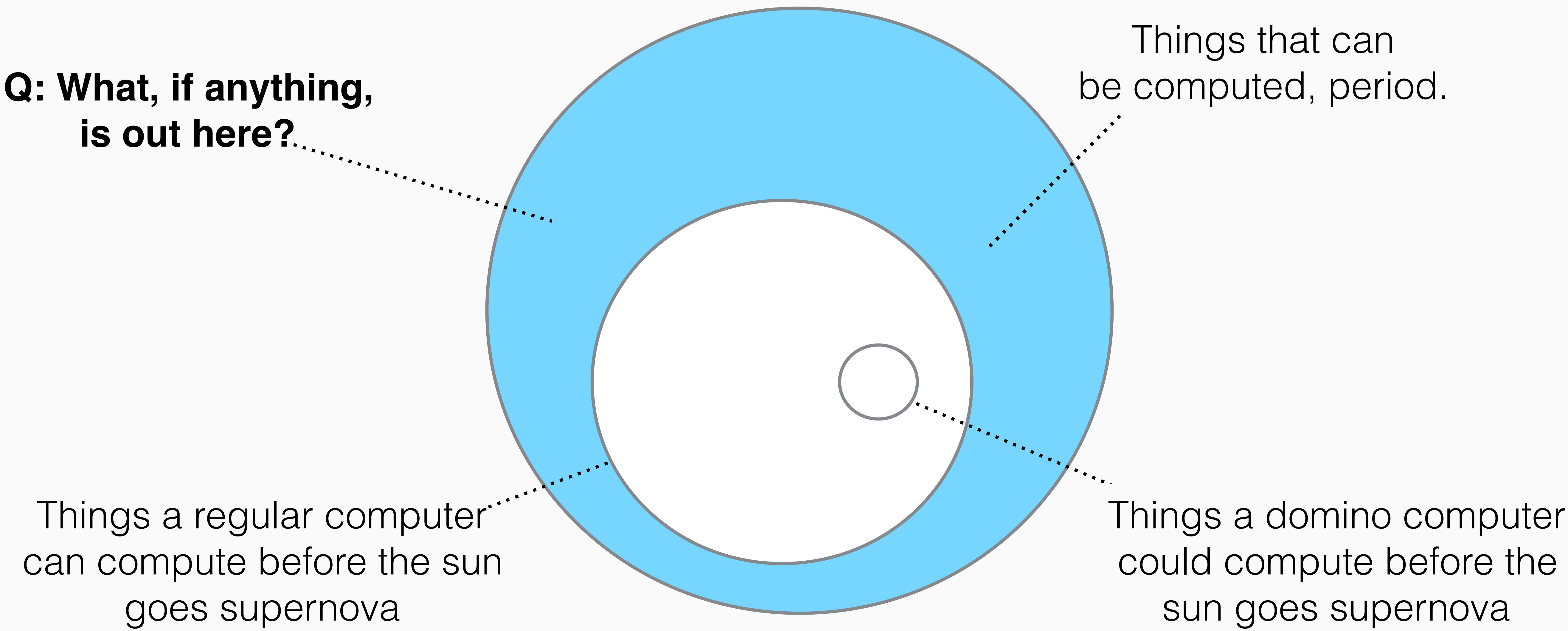
“There is only one notion of computation”

The Church-Turing Thesis



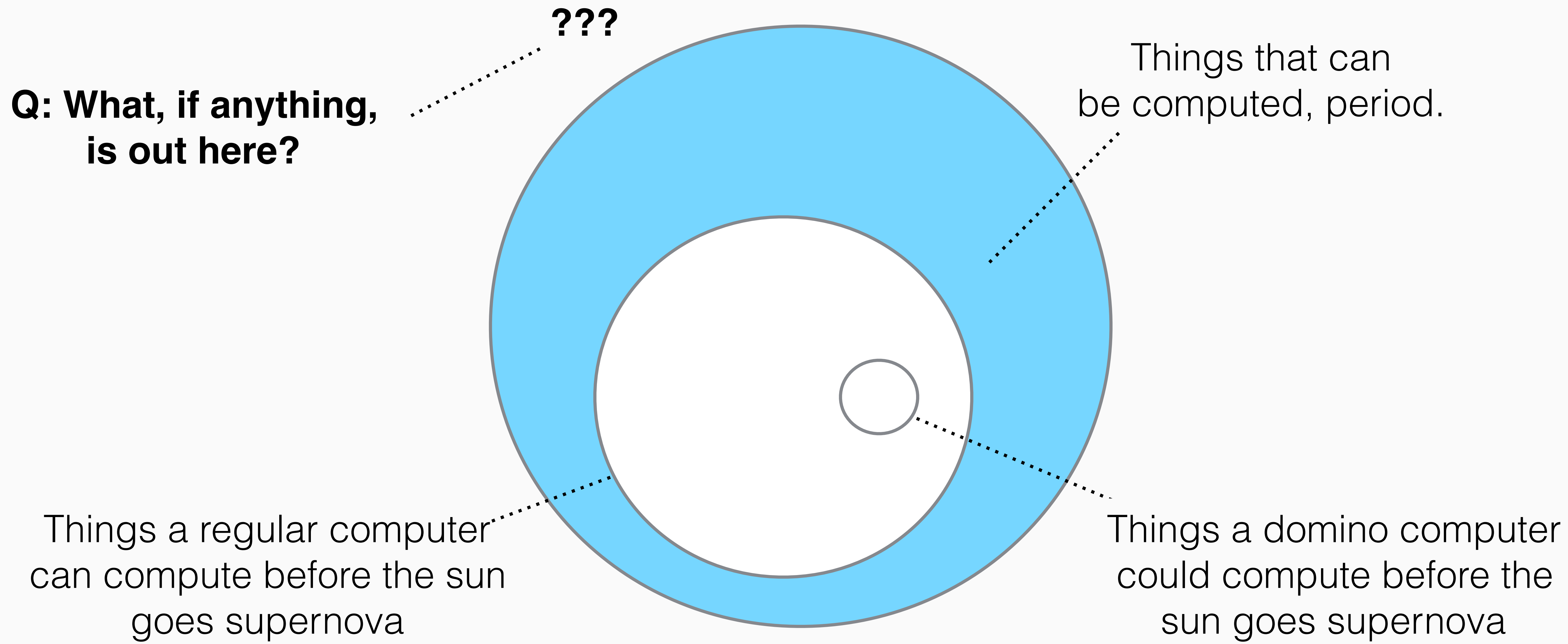
"There is only one notion of computation"

The Church-Turing Thesis



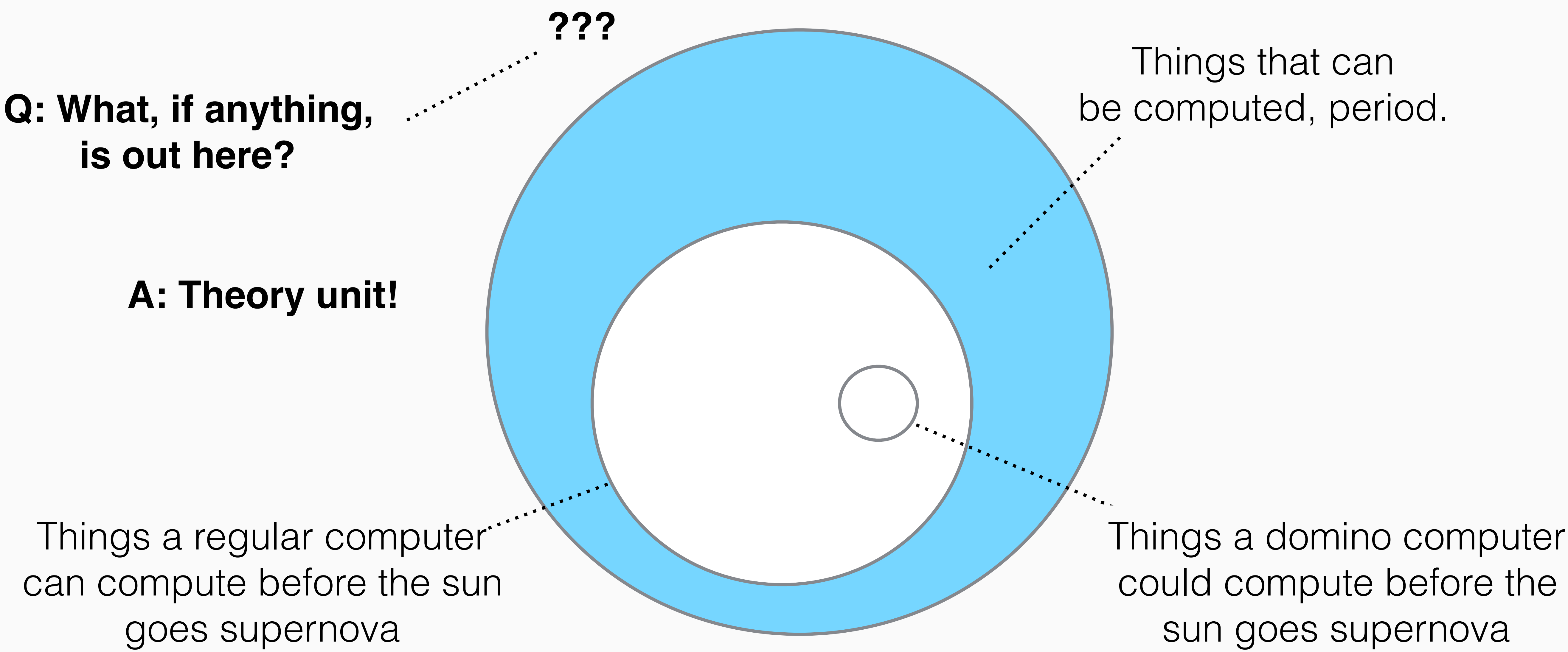
"There is only one notion of computation"

The Church-Turing Thesis



"There is only one notion of computation"

The Church-Turing Thesis



“There is only one notion of computation”

The Church-Turing Thesis

"There is only one notion of computation"

- Some other questions:
  - Is the human brain carrying out computation?
  - Is the comic right? Or are there things in the world that feel like they aren't computation?
  - Is "randomness" computable? How?
  - Is there one "fastest" model of a computer? (e.g. *X* is to the transistor as the transistor is to the domino)

The Computer

- Internal states (**memory**)
- Complex **logic** relating bits
- Mechanism for setting bits (**input**)
- Mechanism for displaying bits (**output**)

Q: What are these in our computers?

The Computer

- Internal states
- Complex logic relating bits
- Mechanism for setting bits (input)
- Mechanism for displaying bits (output)

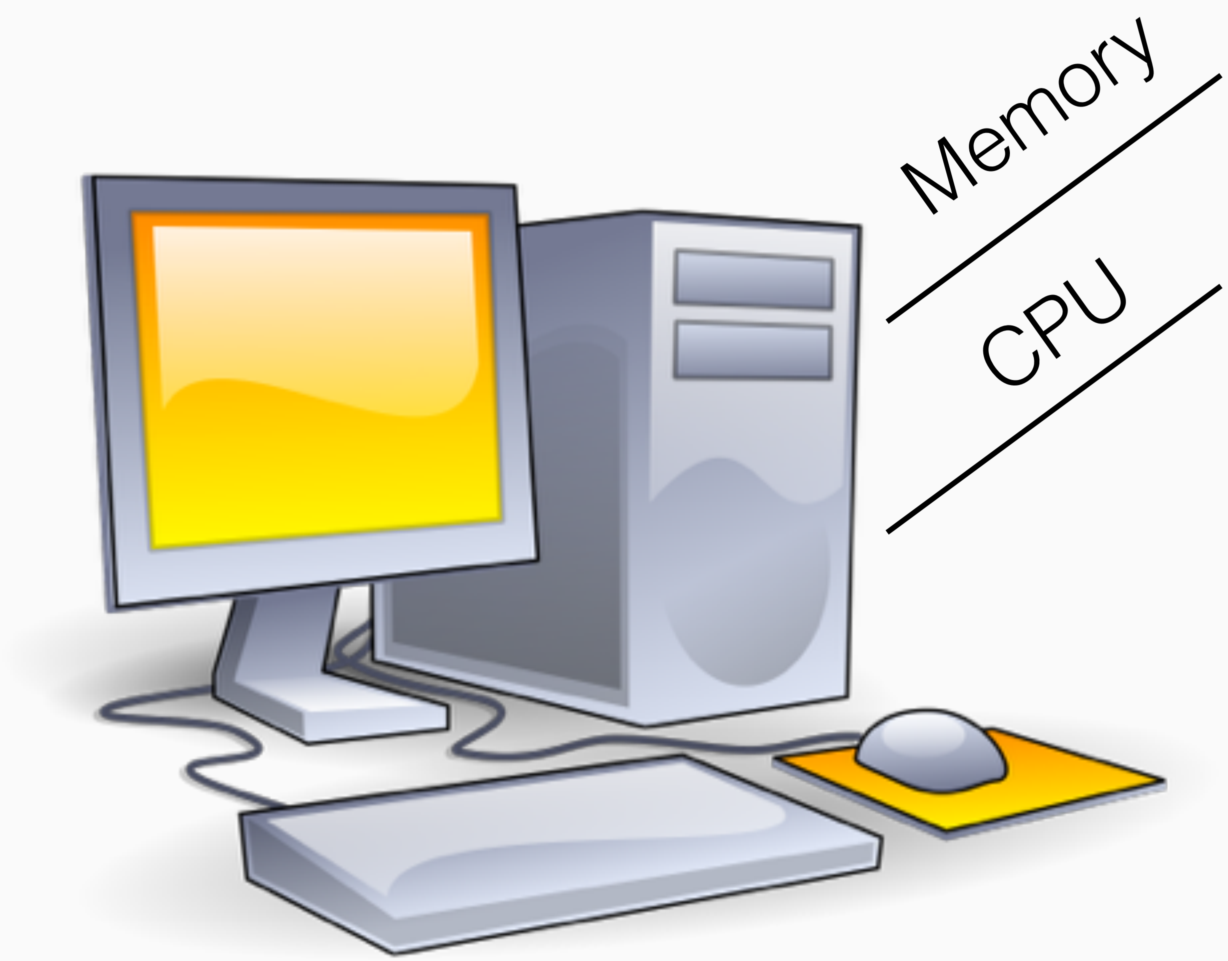
Q: What are these in our computers?

The Computer

- ***Internal states***
- Complex logic relating bits
- Mechanism for setting bits (input)
- Mechanism for displaying bits (output)

Q: What are these in our computers?

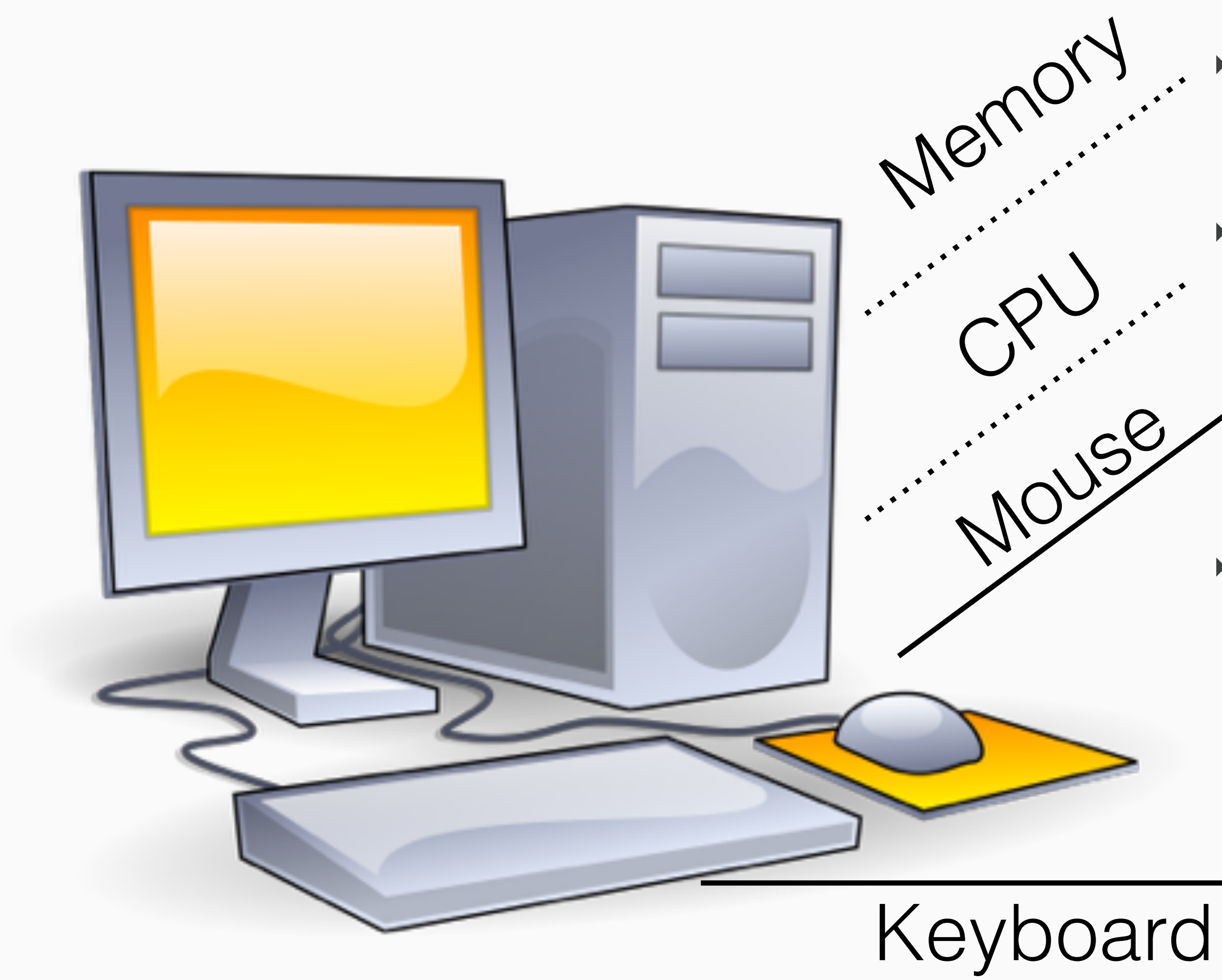
The Computer



- Internal states
- ***Complex logic relating bits***
- Mechanism for setting bits (input)
- Mechanism for displaying bits (output)

Q: What are these in our computers?

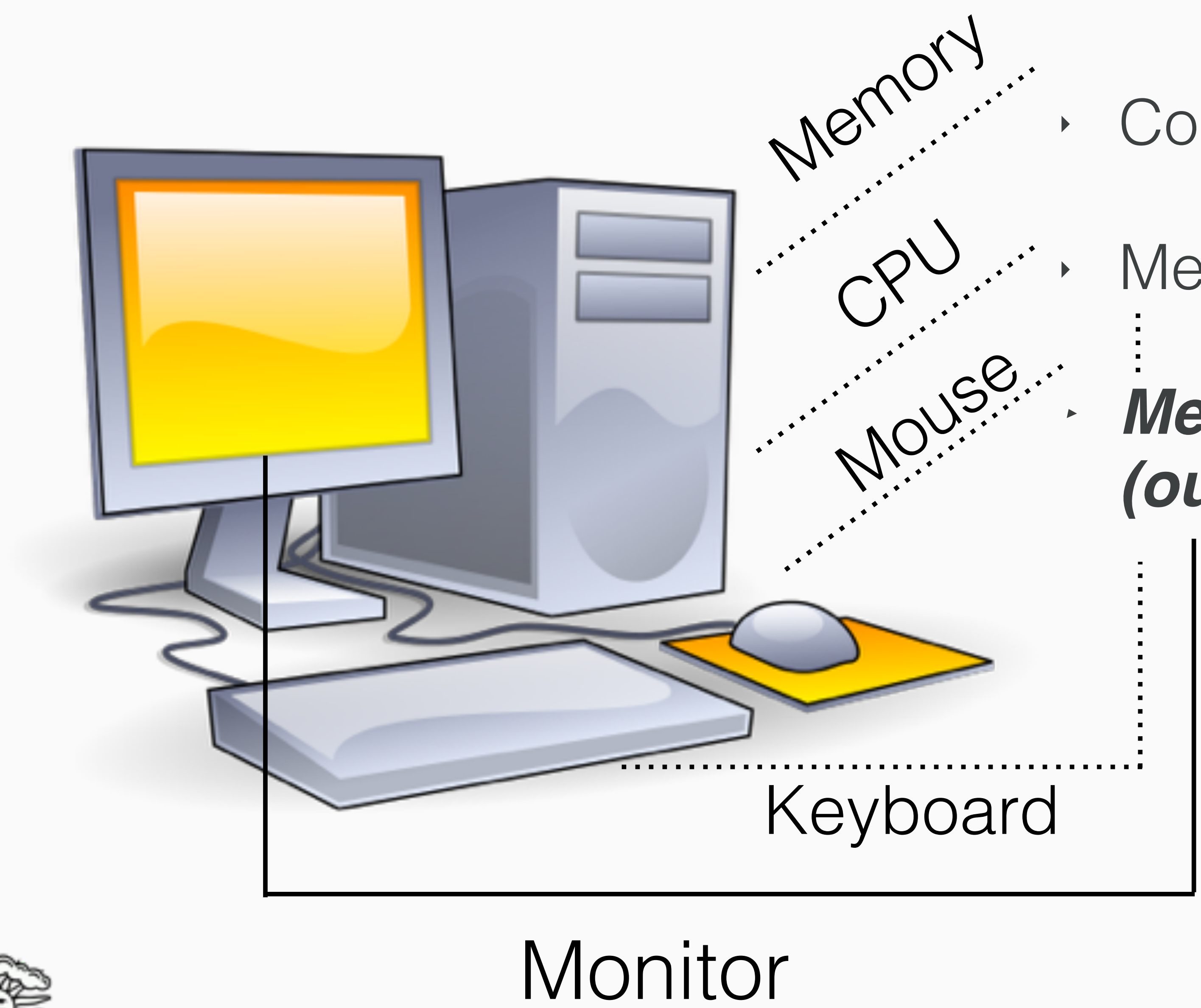
The Computer



- Internal states
- Complex logic relating bits
- ***Mechanism for setting bits (input)***
- Mechanism for displaying bits (output)

Q: What are these in our computers?

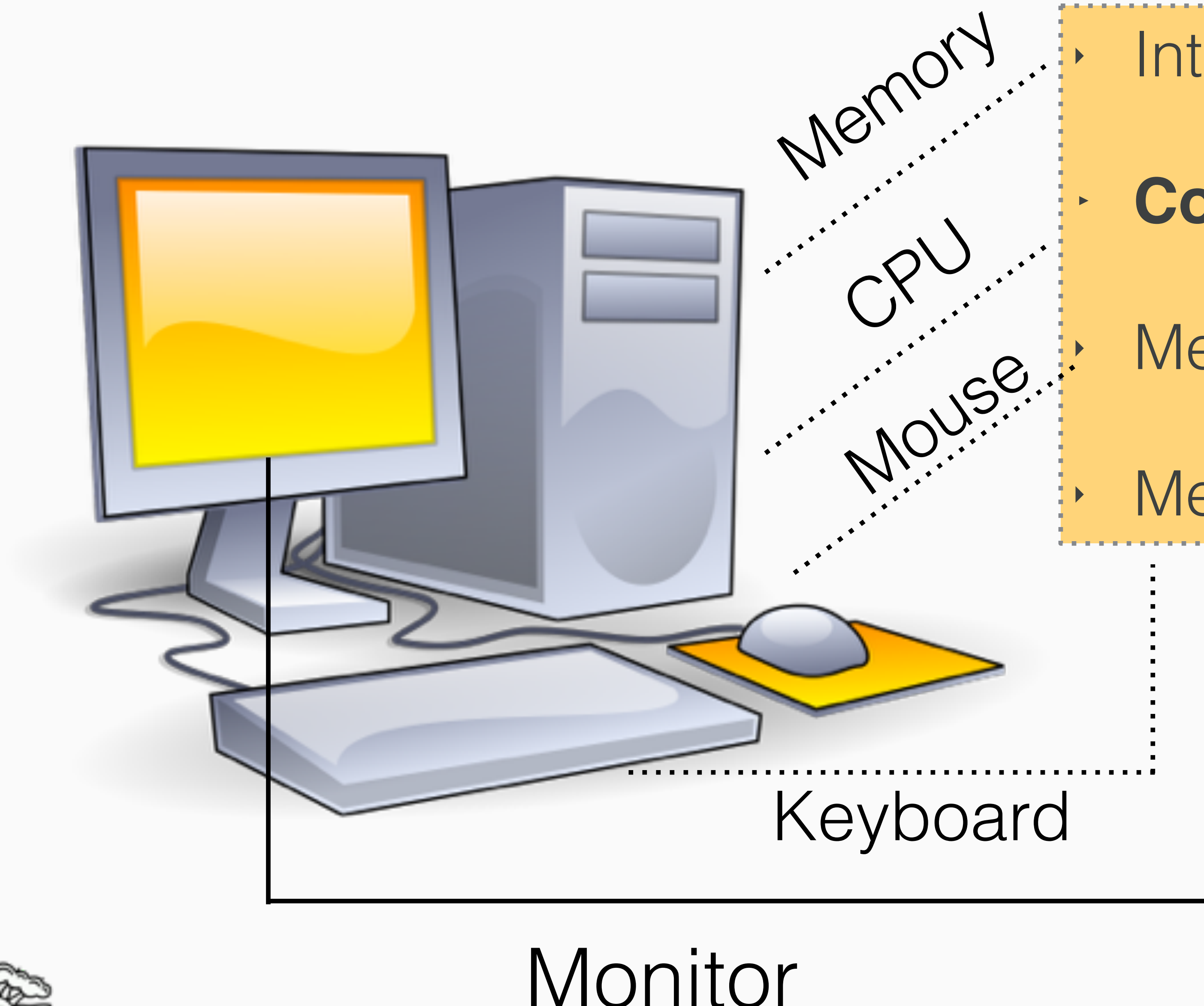
The Computer



- Internal states
- Complex logic relating bits
- Mechanism for setting bits (input)
- ***Mechanism for displaying bits (output)***

Q: What are these in our computers?

Next Time



- Internal states
- **Complex logic relating bits**
- Mechanism for setting bits (input)
- Mechanism for displaying bits (output)

Programming lets us define the “complex logic”!