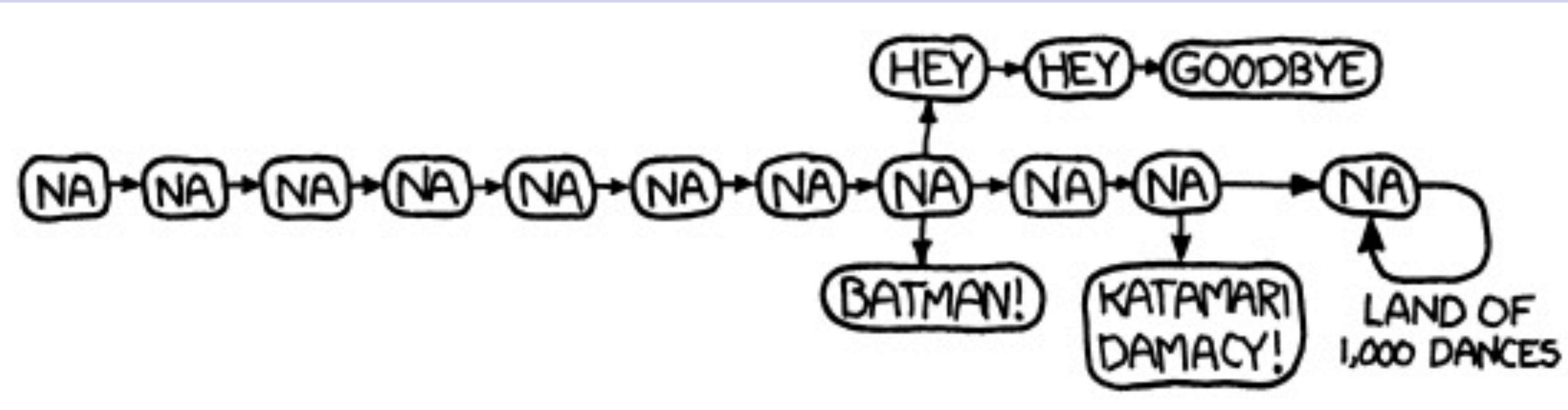
heard (of) these four songs?



finite-state machines ~ ***capture patterns***

"state machine" "finite automaton"

https://www.youtube.com/watch?v=IoyvvEWHodk @ 2:05
https://www.youtube.com/watch?v=N6Cfy58-zPU @ 0:50

https://www.youtube.com/watch?v=iMH49ieL4es @ 5:07
https://www.youtube.com/watch?v=3mz_EXHKGHs @ 1:16

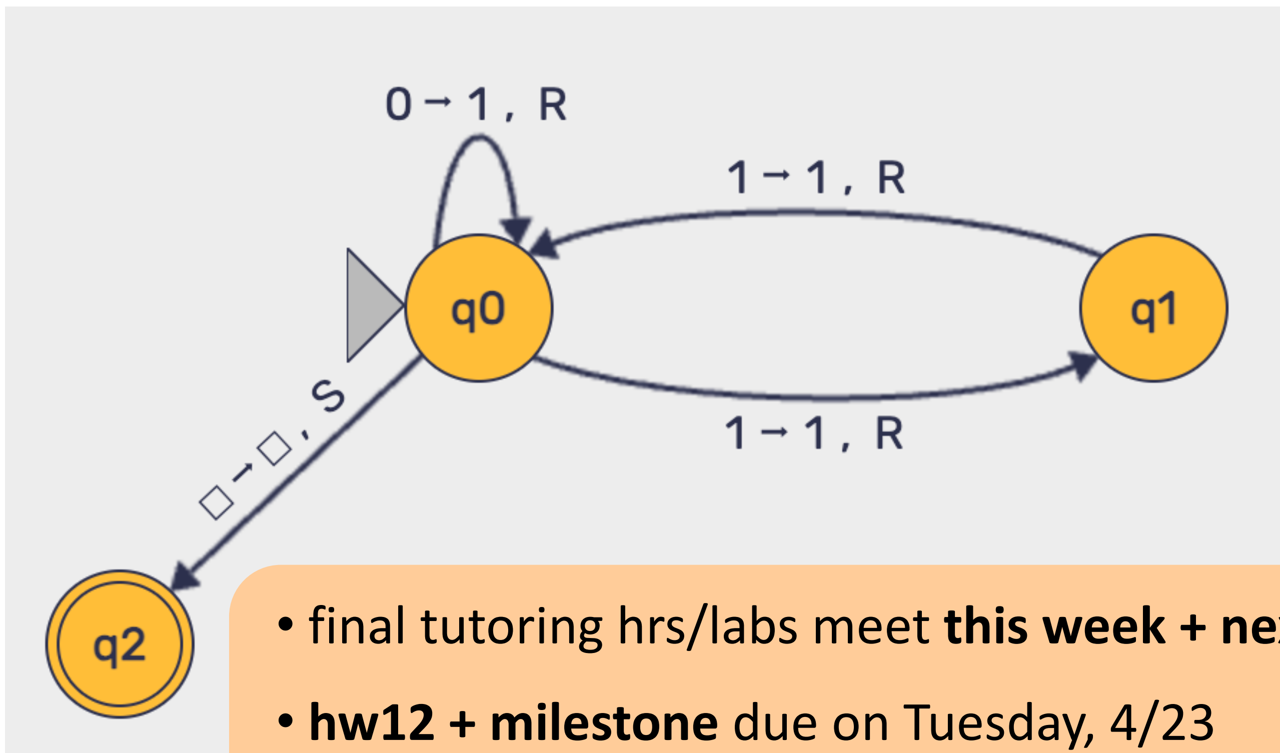
not all patterns "work"

CS 5 today: *more machines!*

Final ideas...

Turing Machines and the **MANY** *things computers can't compute... !*

This machine doesn't look all-powerful to me!

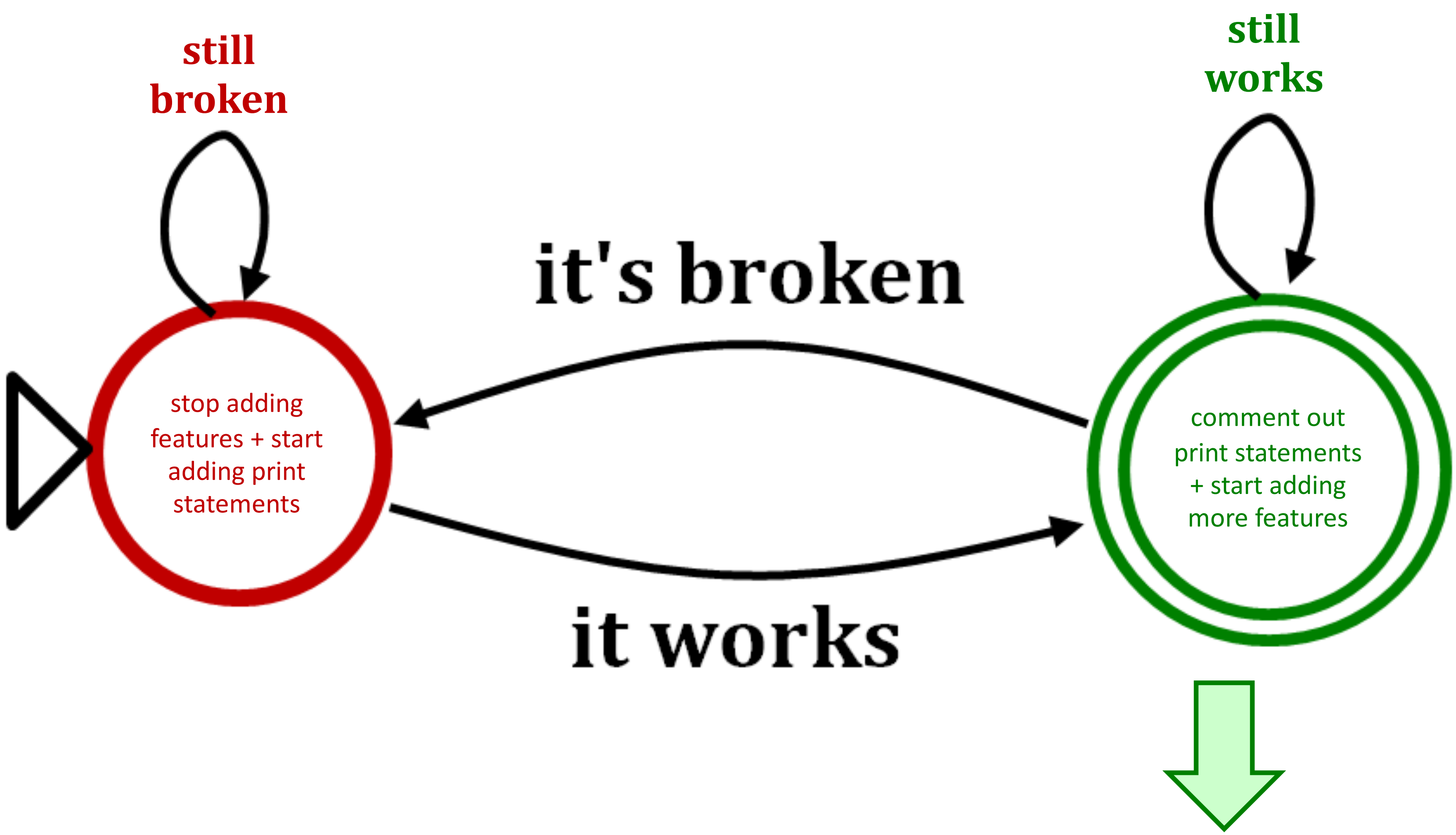


Final projects

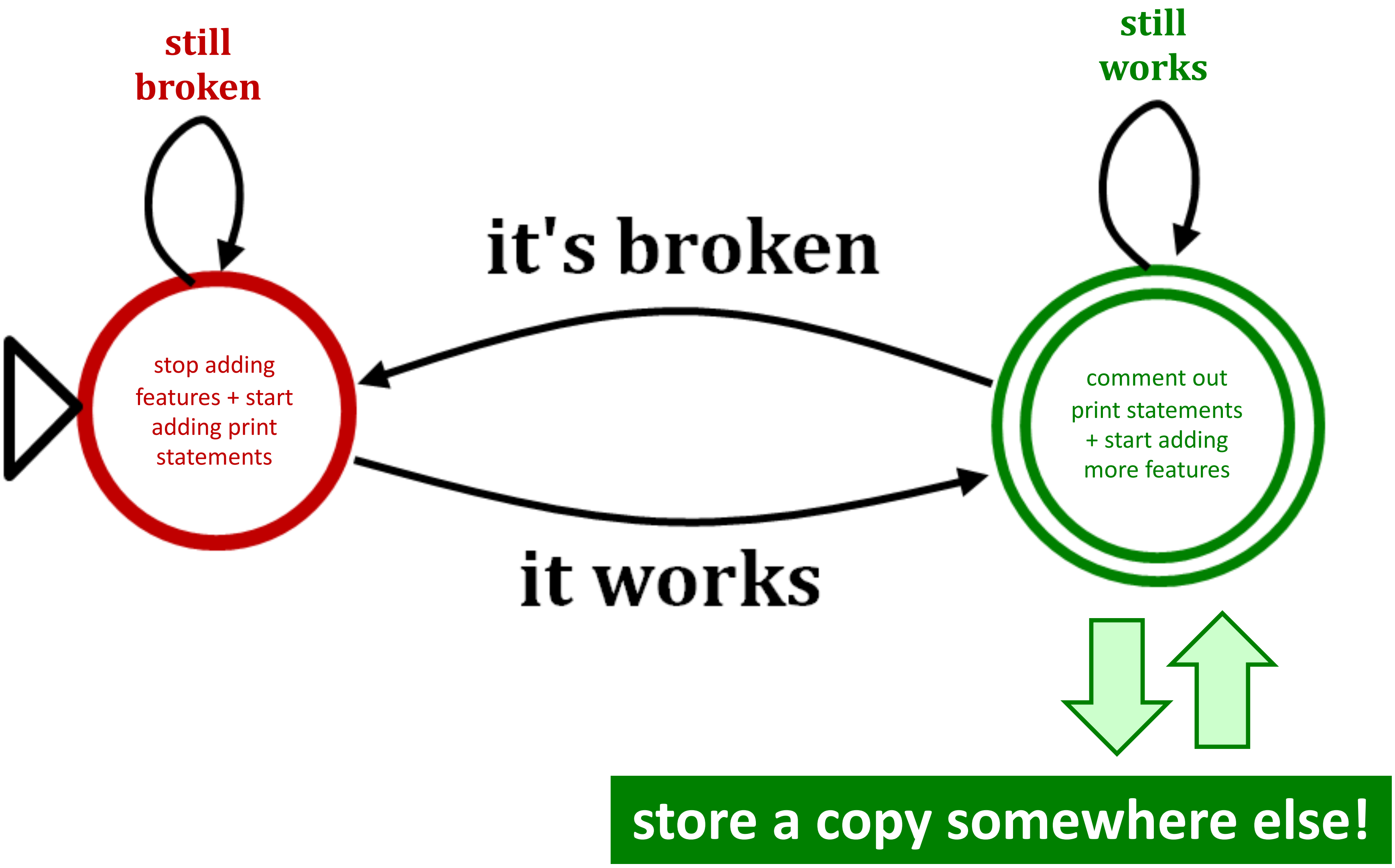
- final tutoring hrs/labs meet **this week + next**
- **hw12 + milestone** due on Tuesday, 4/23
- **5 finite-state machines** due as part of hw12

Turing machines extra!

Final project state machine

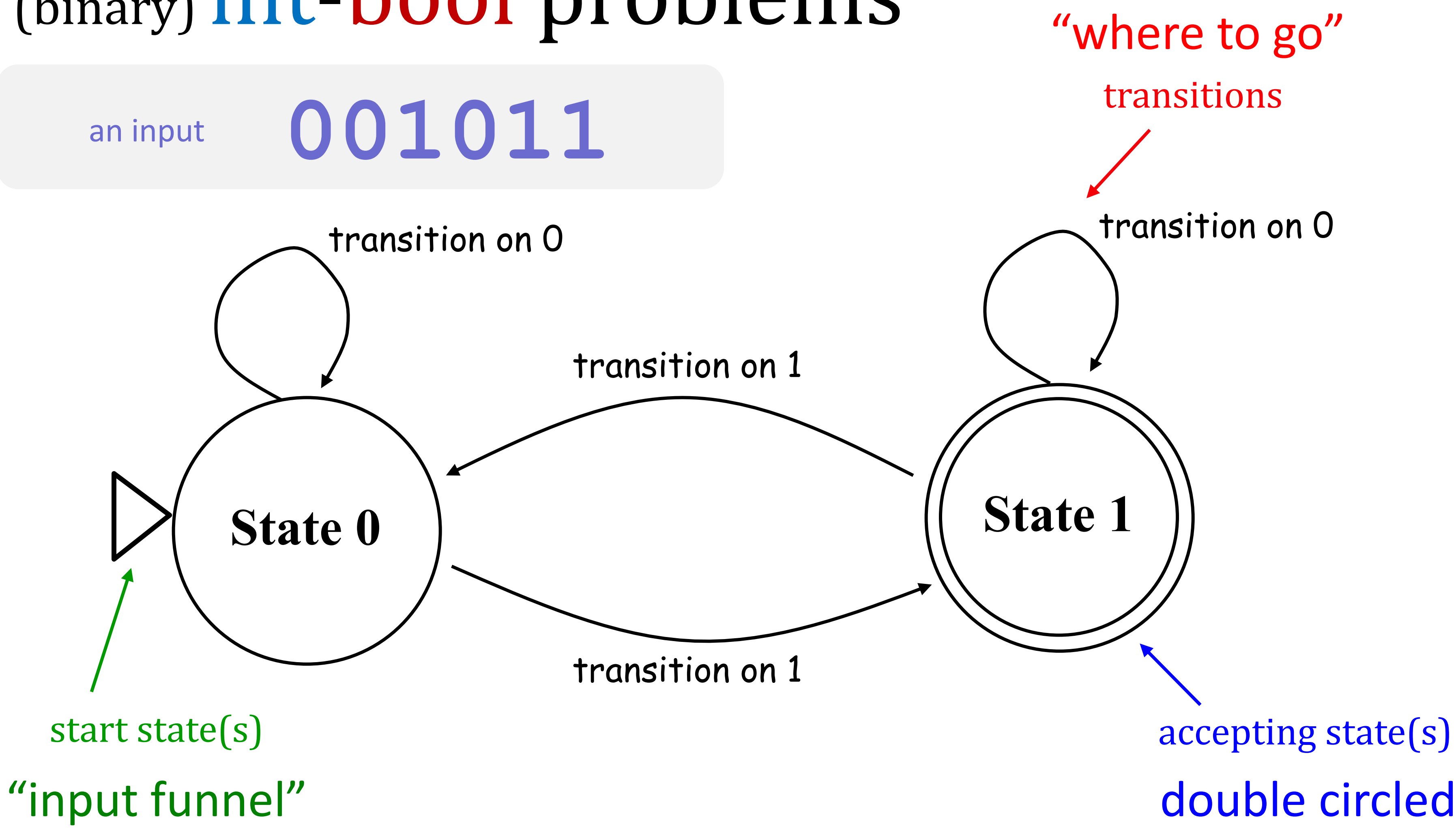


Final project state machine



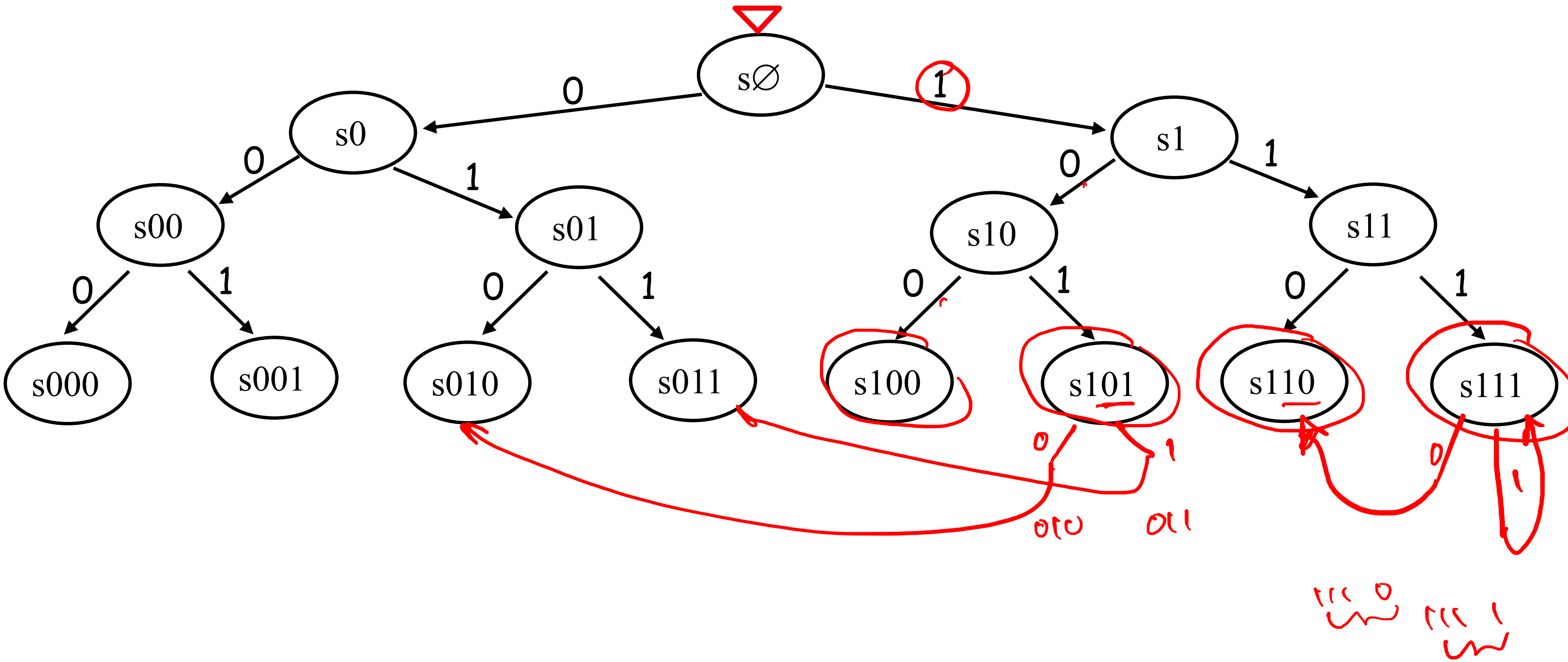
Our current-domain:
(binary) int-bool problems

an input **001011**



State-machine *limits*?

Surprising things FSMs ***can*** do...



Accept inputs whose ***third-to-last*** digit is a 1... !

Do we ***need*** 15 states?

State-machine *limits*?

Surprising things FSMs ***can*** do...

Pretend we started with 000!

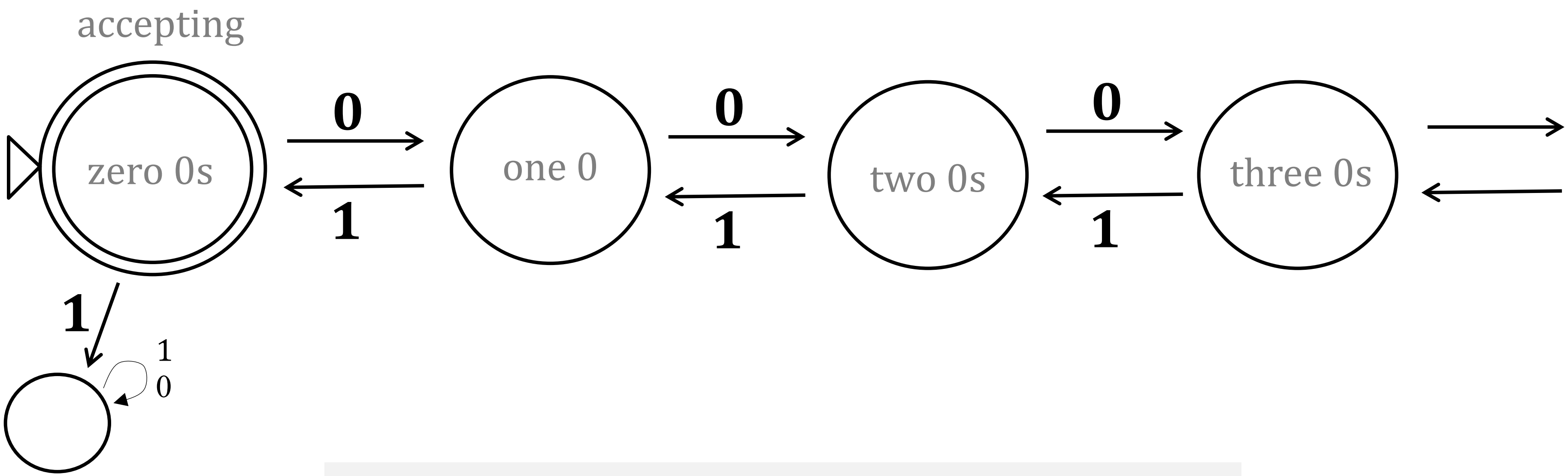
only 8 states needed...

w/ the bottom-row transitions!

Accepting inputs whose ***third-to-last*** digit is a 1... !

State-machine ***limits***?

You don't need three eyes to see some problems here!



A possible language: binary strings with ***any*** **# of 0s** followed by the ***same*** **#of 1s**

rejected

011

001

11100

00110

accepted

000111

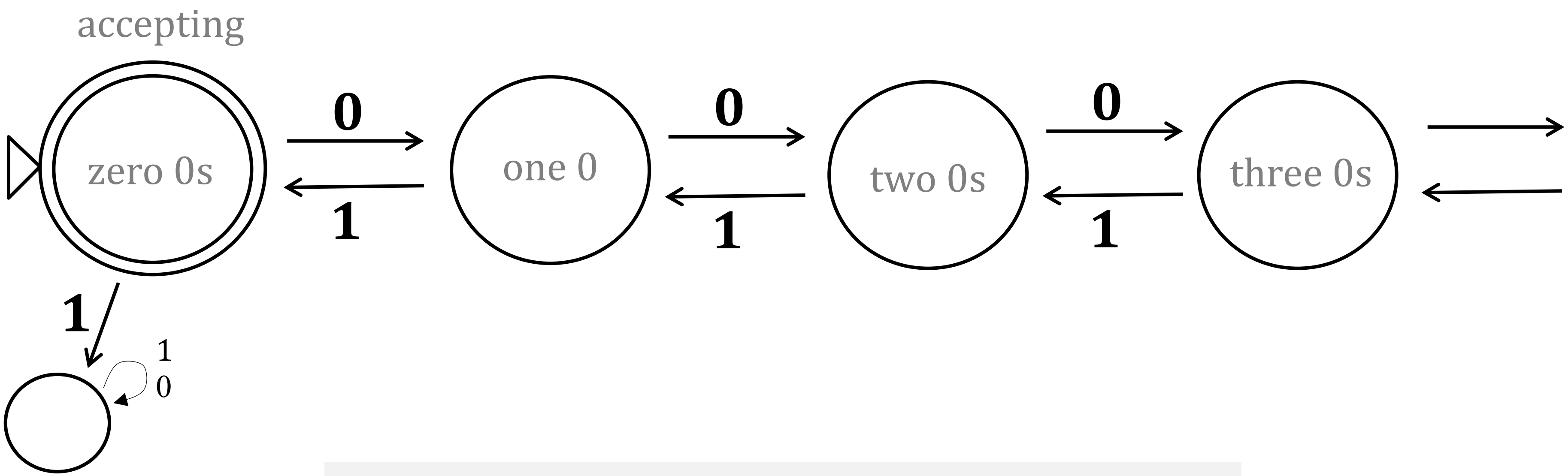
0011

01

λ

State-machine ***limits***?

You don't need three eyes to see some problems here!



A possible language: binary strings with ***any*** **# of 0s** followed by the ***same*** **#of 1s**

rejected

011

001

11100

00110

FSMs "can't count"

at least, not arbitrarily high

accepted

000111

0011

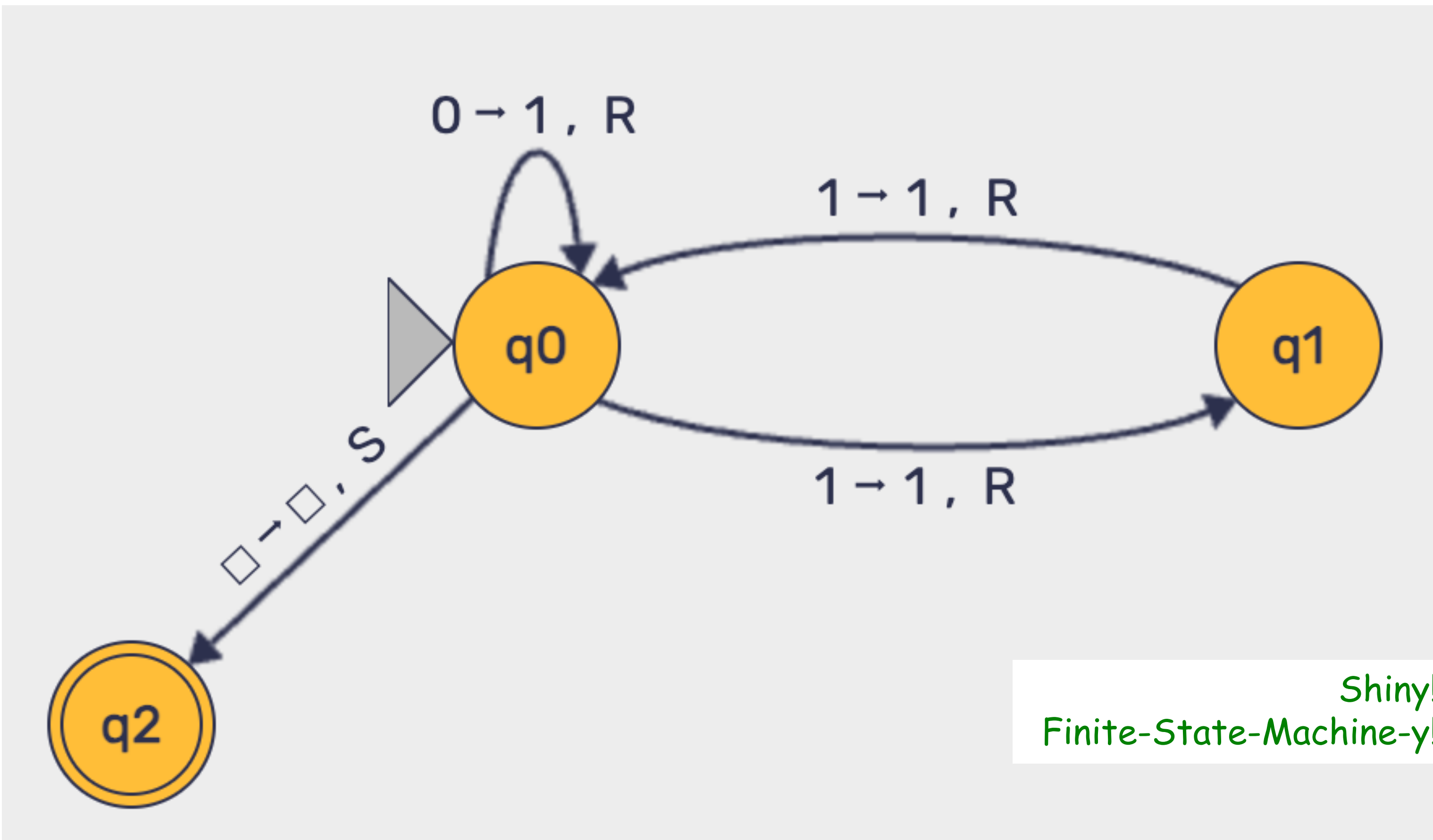
01

λ

FSMs can't ***count***...

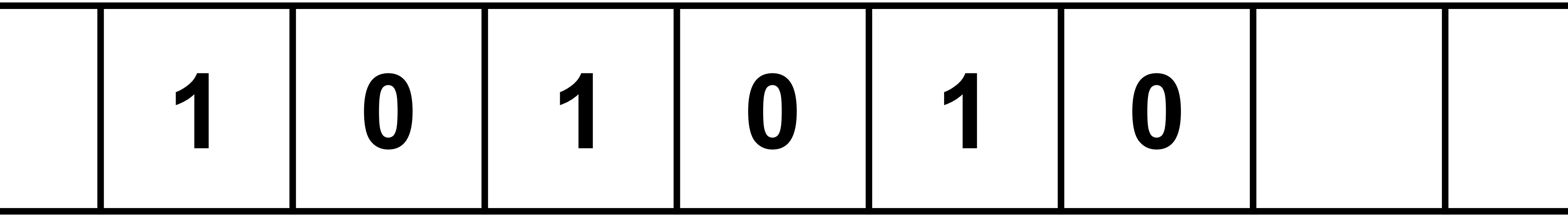
So, let's build a better machine!

(that can model <u>any</u> software, hardware, and computation!)



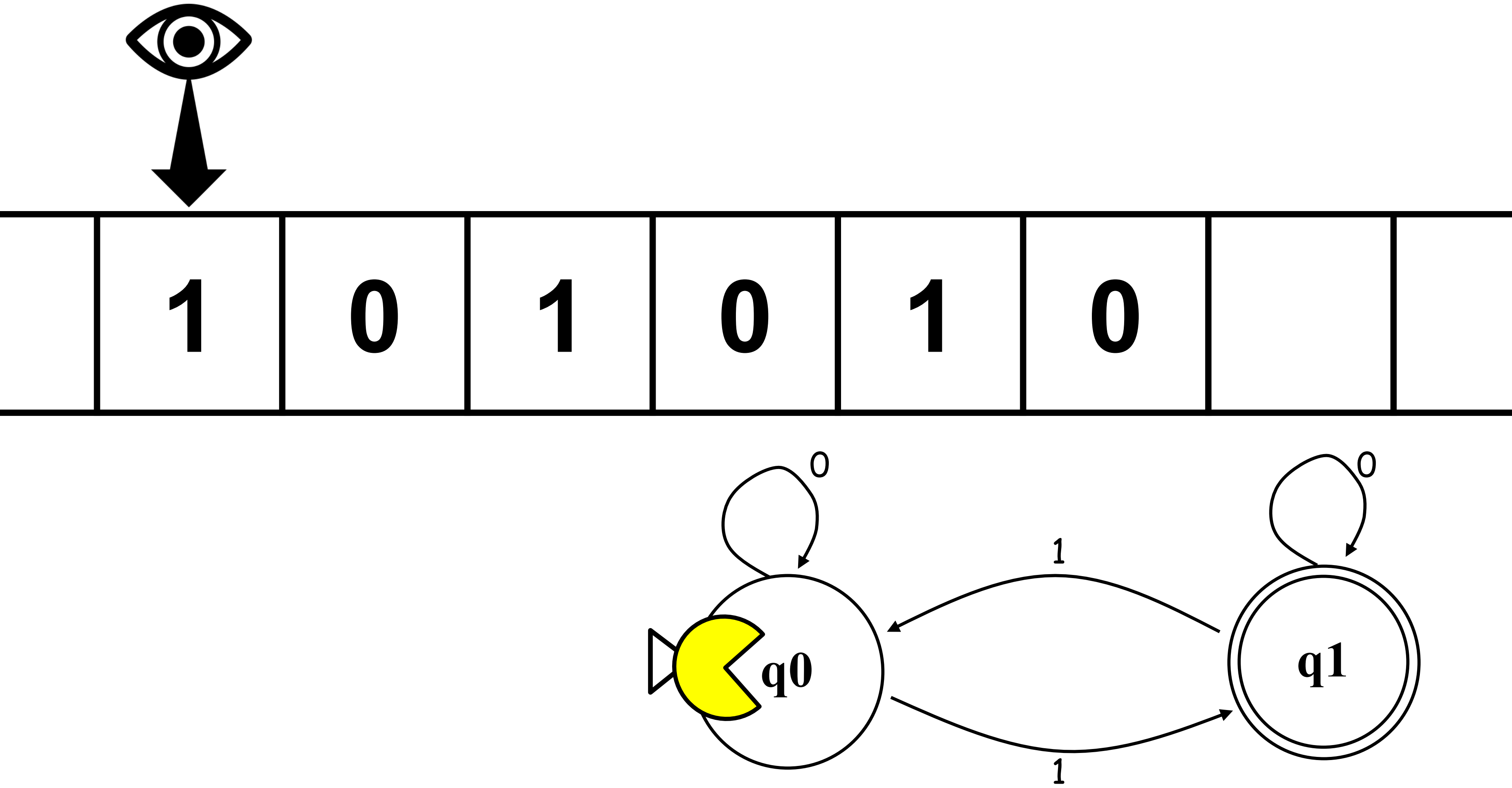
Shiny!
Finite-State-Machine-y!

What if we could save our input somewhere?

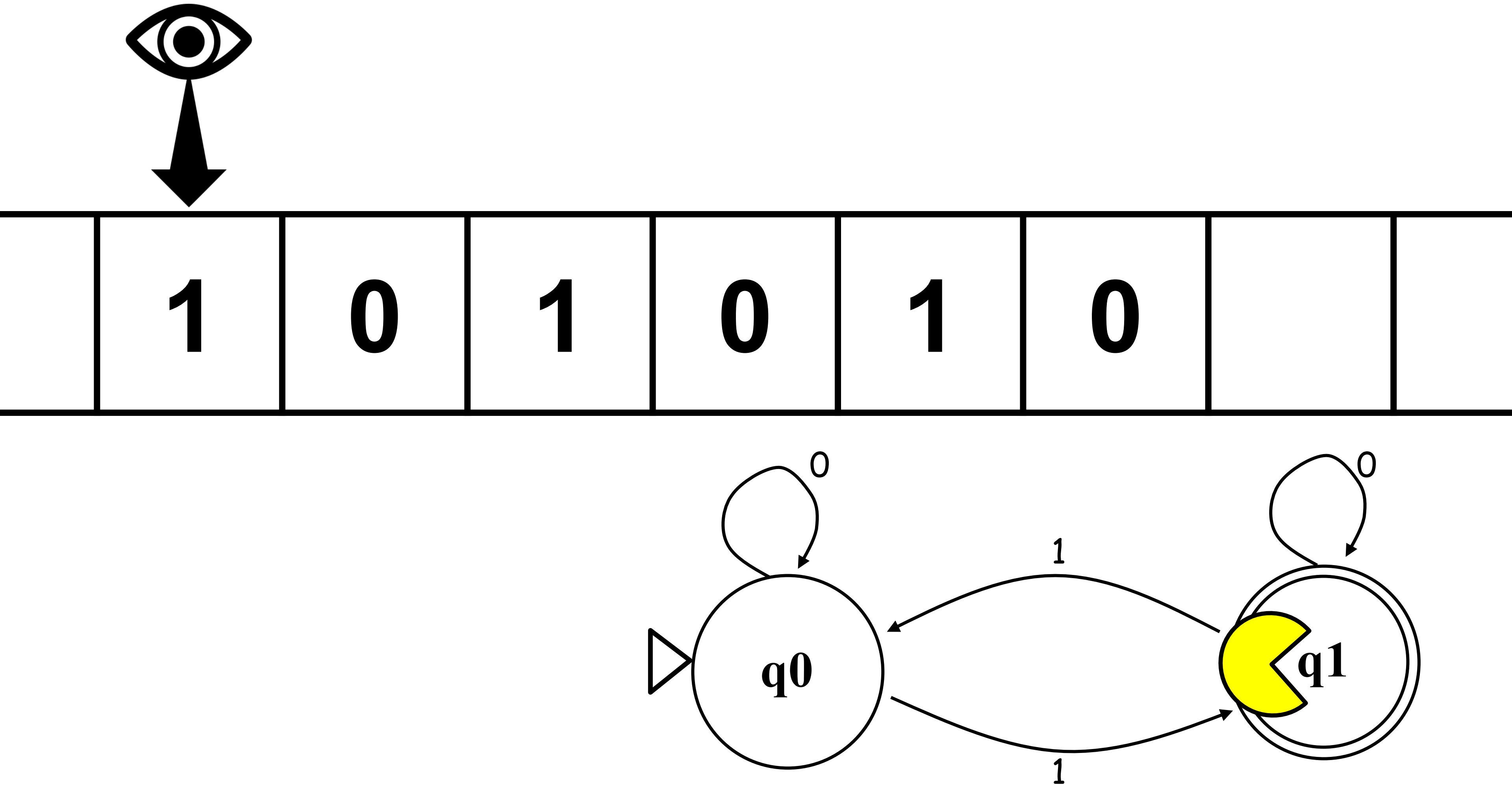


The Tape

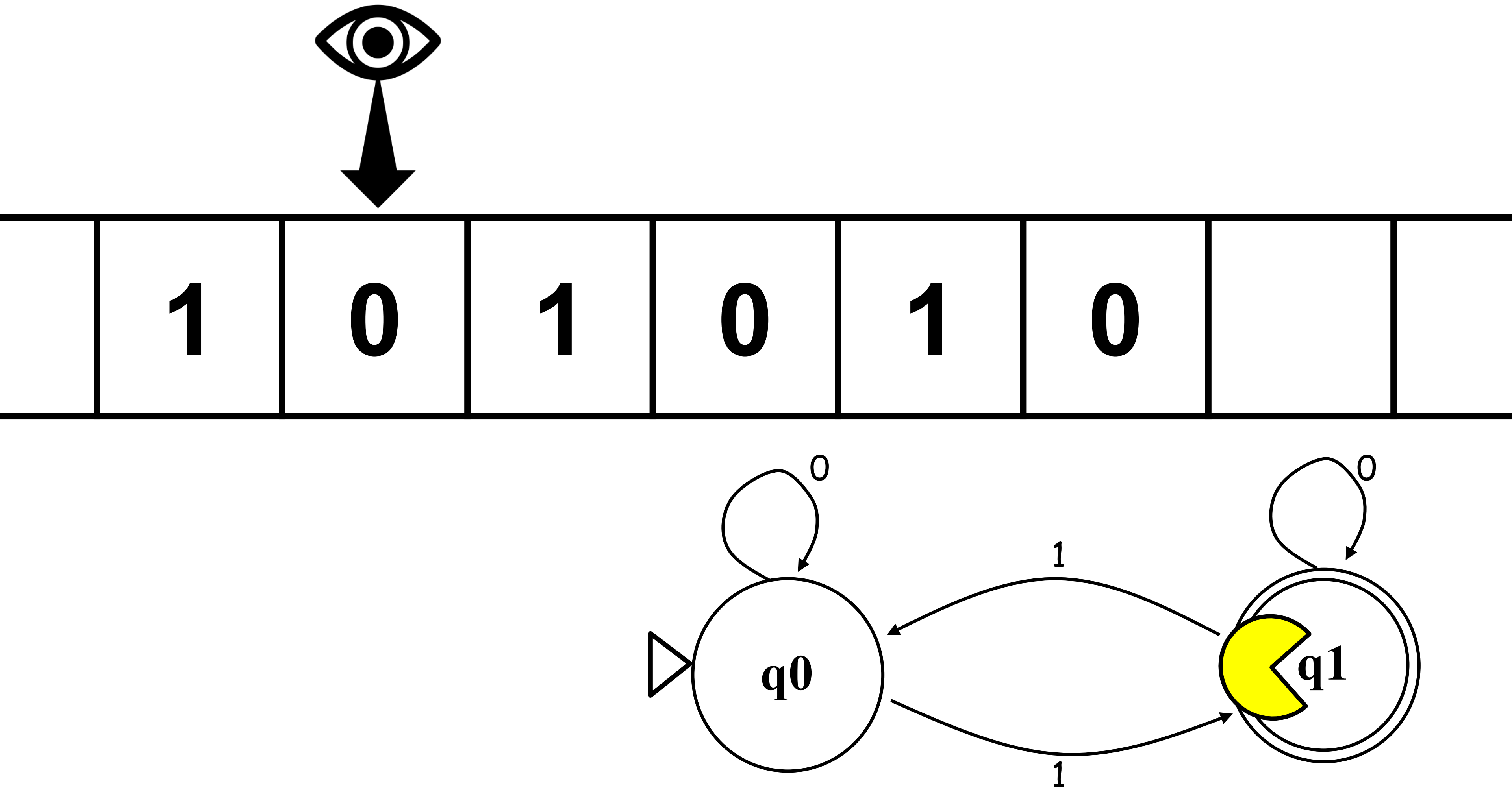
(with infinite length)



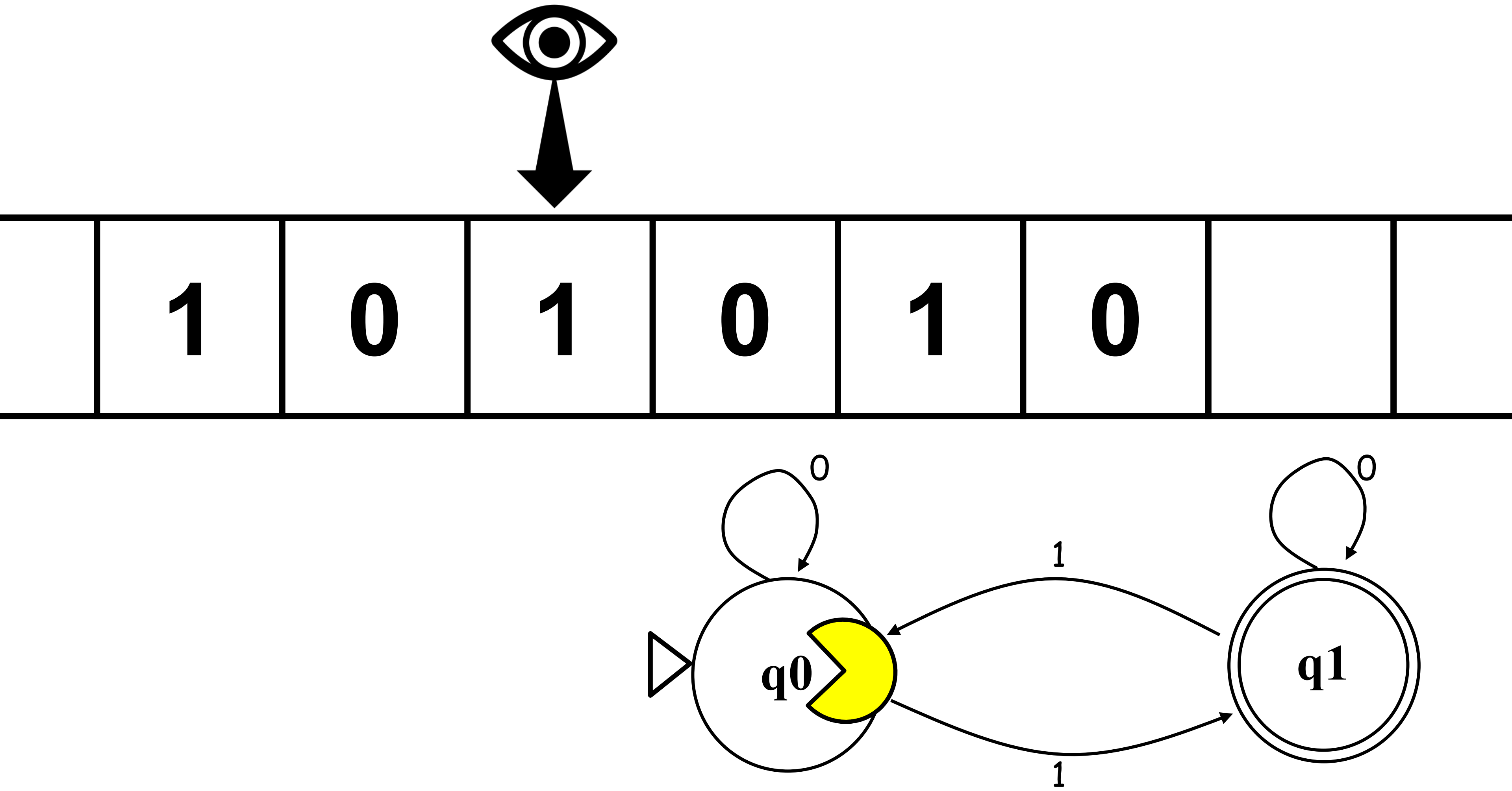
Finite state machines have to move ***one step right*** at each step



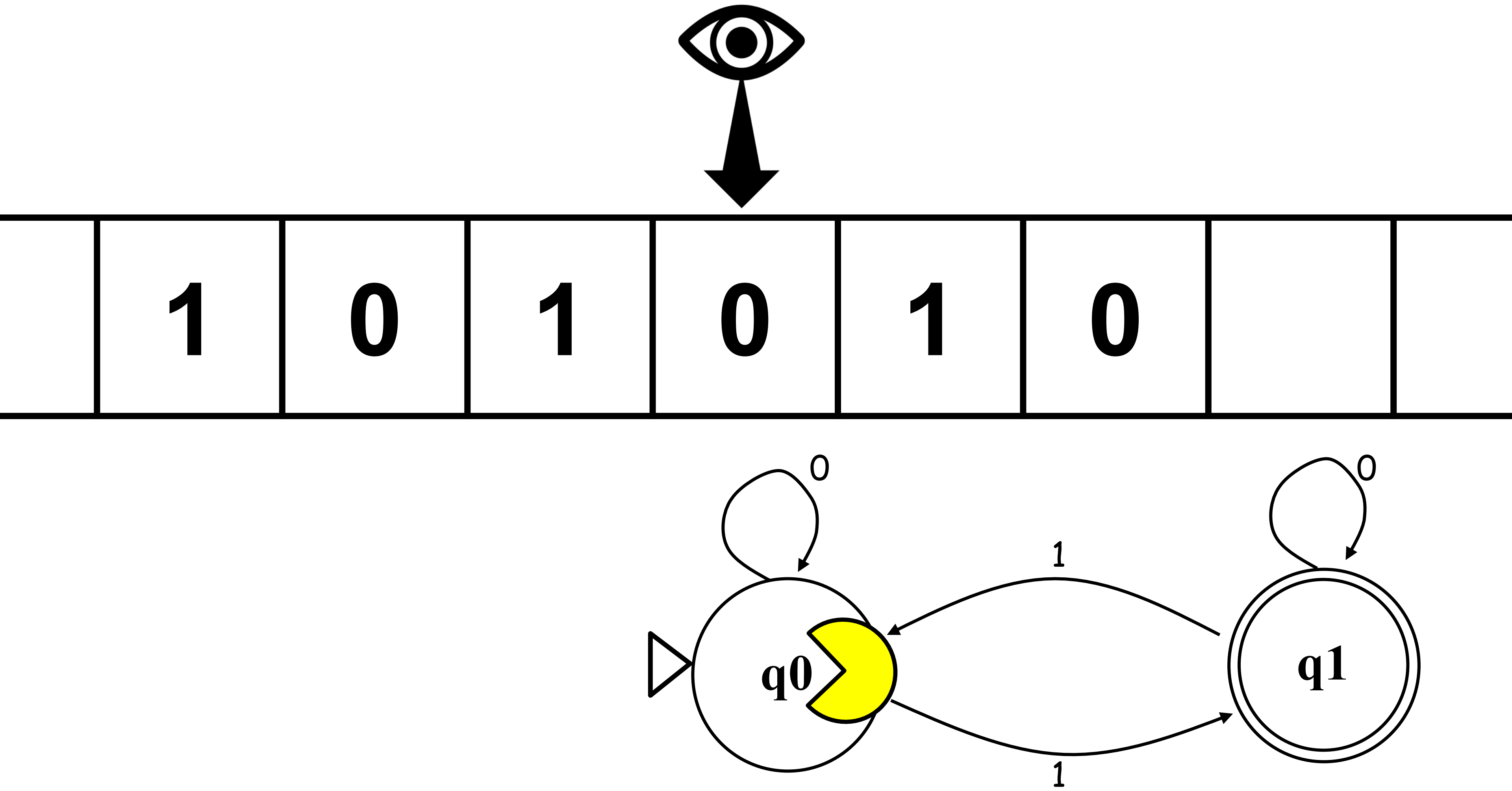
Finite state machines have to move
one step right at each step



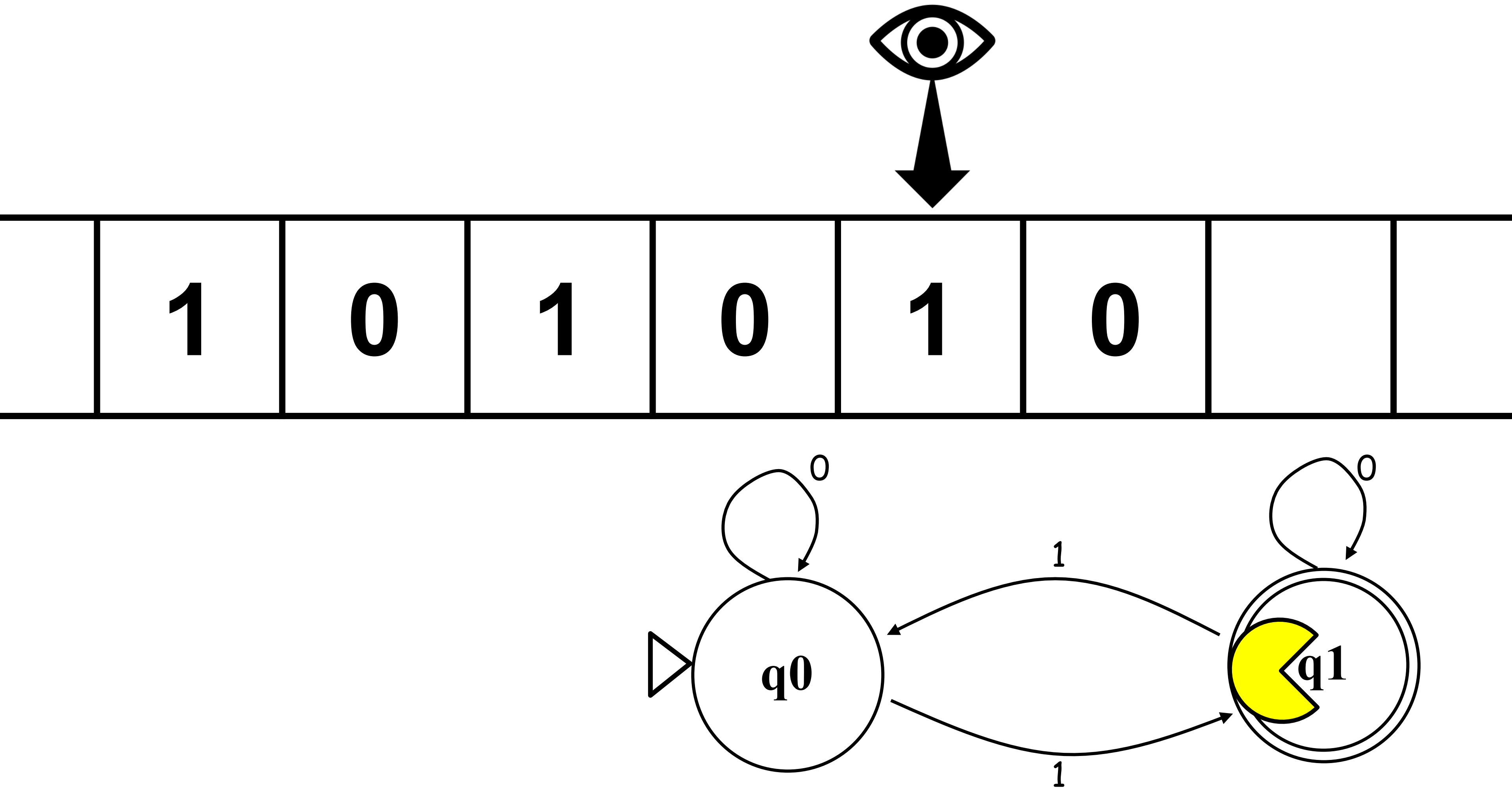
Finite state machines have to move ***one step right*** at each step



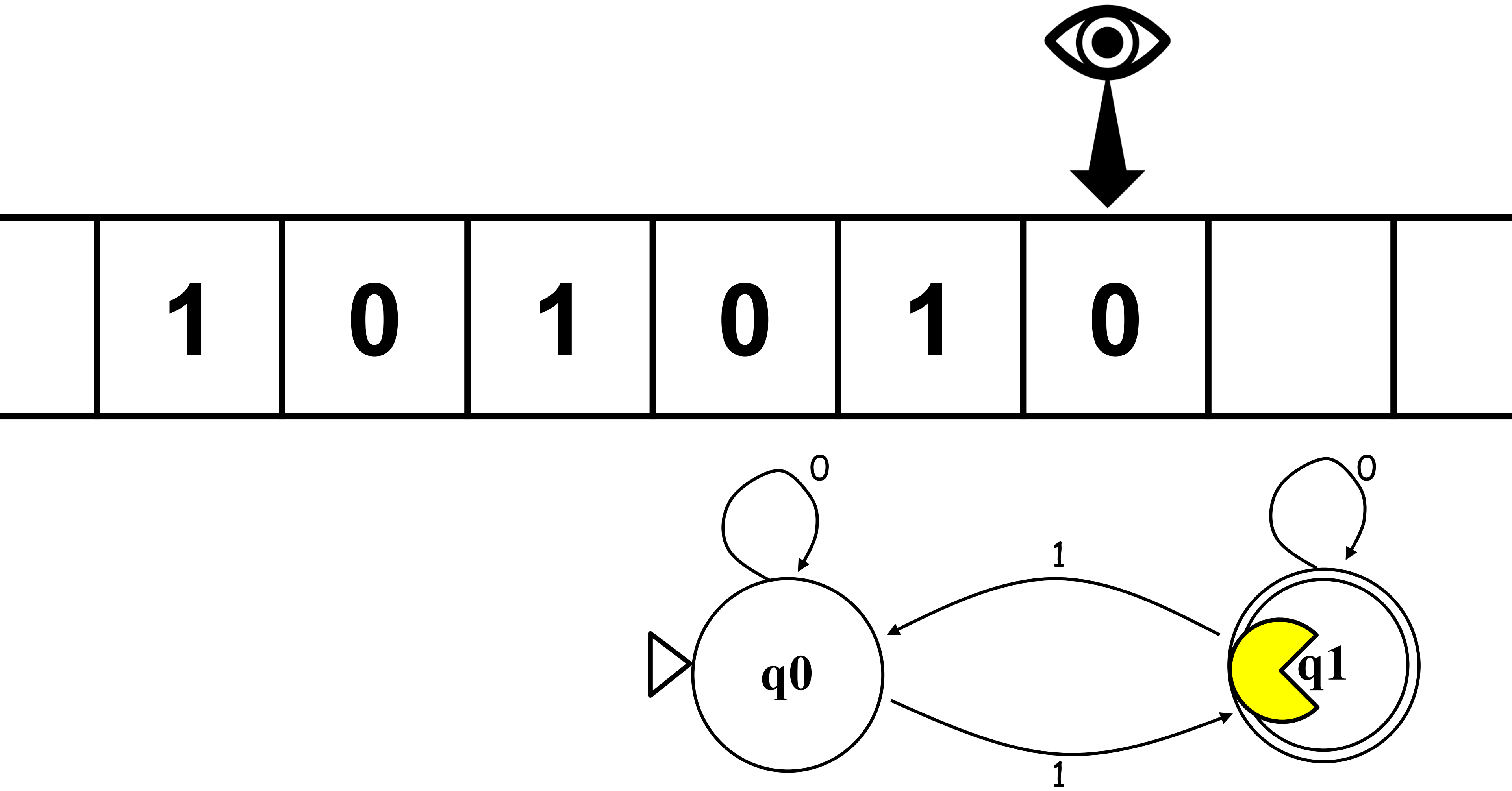
Finite state machines have to move
one step right at each step



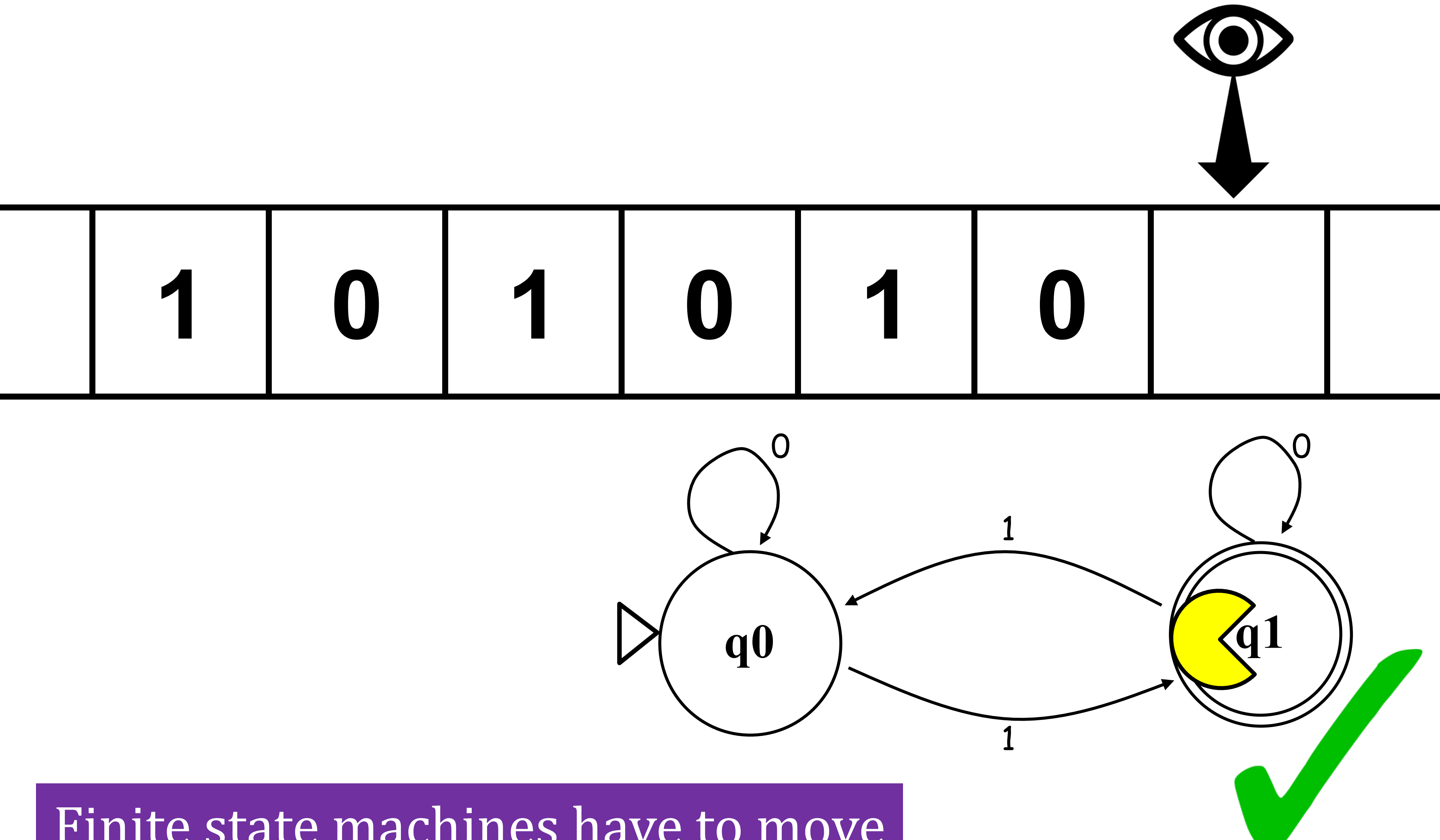
Finite state machines have to move ***one step right*** at each step



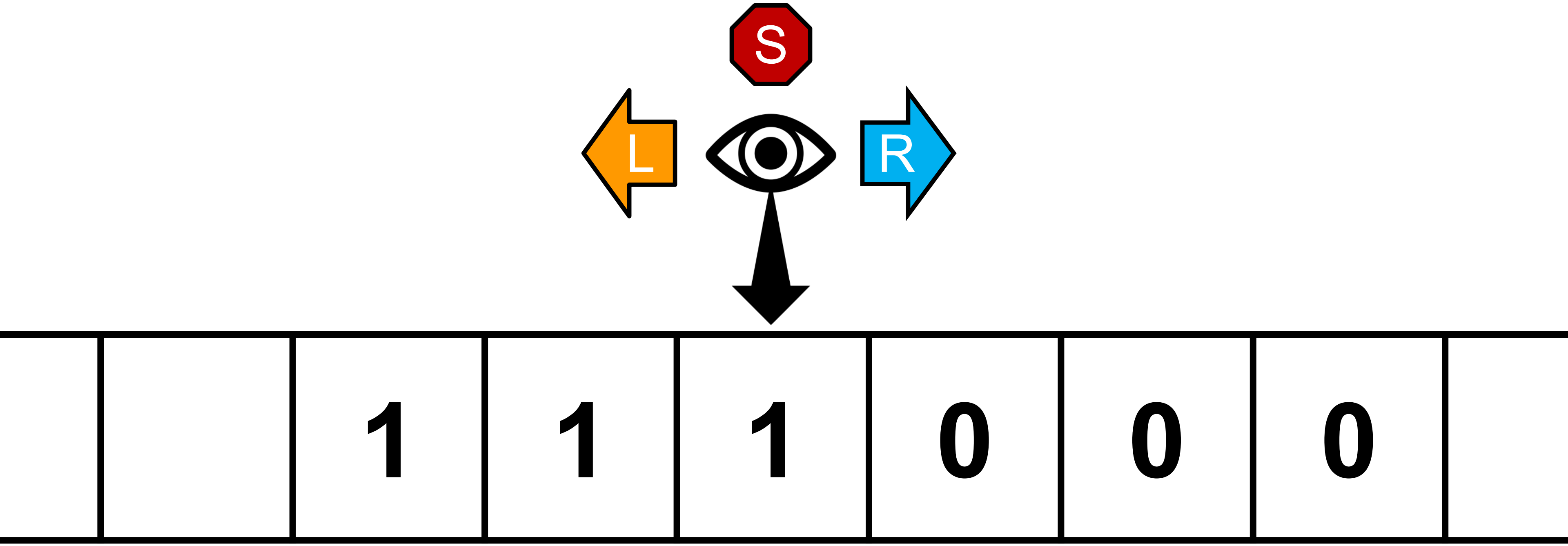
Finite state machines have to move
one step right at each step



Finite state machines have to move ***one step right*** at each step

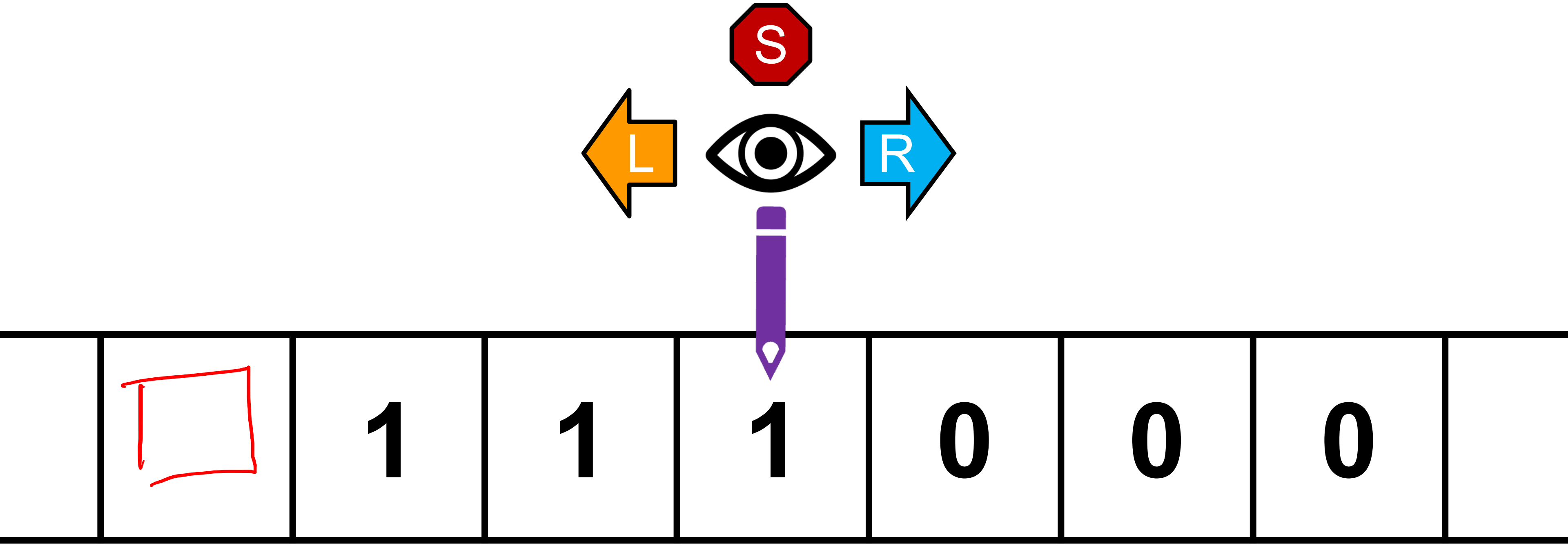


Finite state machines have to move
one step right at each step



What if we were allowed three motions:
L(eft), R(ight), and S(tay)

Could this solve our problem from before?
"*n* 0s followed by *n* 1s"



What if we were allowed three motions:
L(eft), R(ight), and S(tay)

AND we could also ***replace the character at the location we read?***

This is called a **Turing Machine** (or TM for short)

... motions:

..., R(ight), and S(tay)

AND we could also ***replace the character at the location we read?***

So far, all known computational devices can compute <u>only</u> what Turing Machines can...

some are faster than others...

Quantum computation
http://www.cs.virginia.edu/~robins/The_Limits_of_Quantum_Computers.pdf

Molecular computation
http://www.arstechnica.com/reviews/2q00/dna/dna-1.html

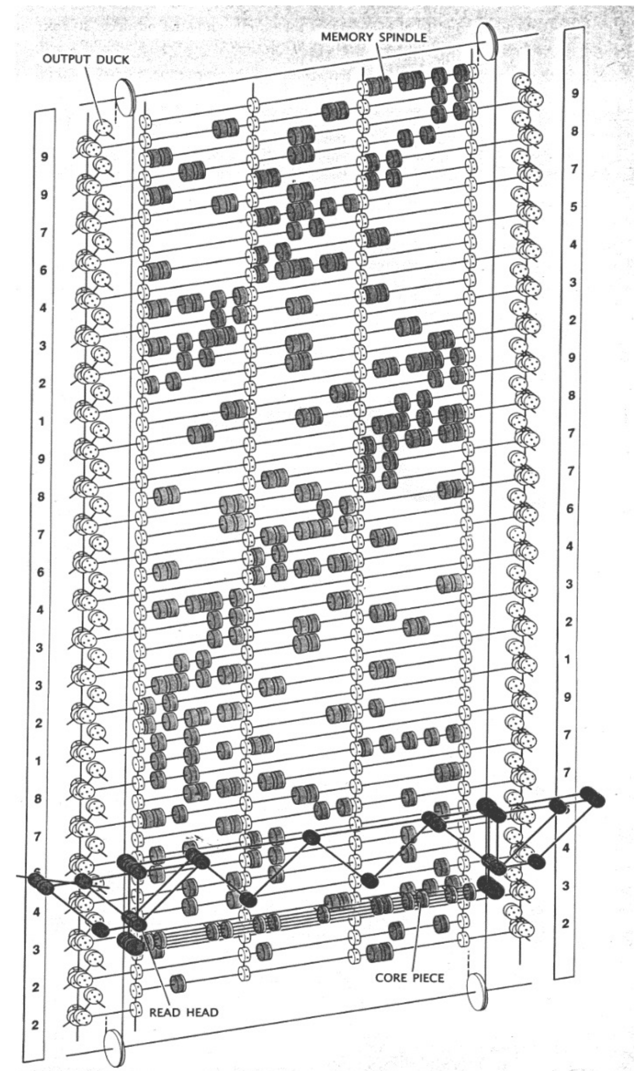
Parallel computers

Integrated circuits

Electromechanical computation

Water-based computation

Tinkertoy computation



The Tinkertoy computer: ready for a game of tic-tac-toe

Turing machine

So far, all known computational devices
compute only what Turi

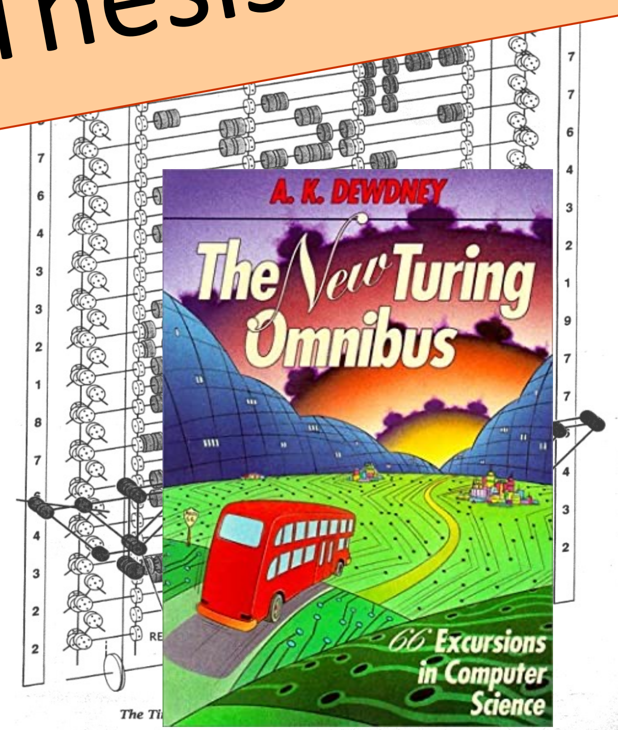
TMs can compute all that *can* be computed!

"Church-Turing Thesis"

echanical computation

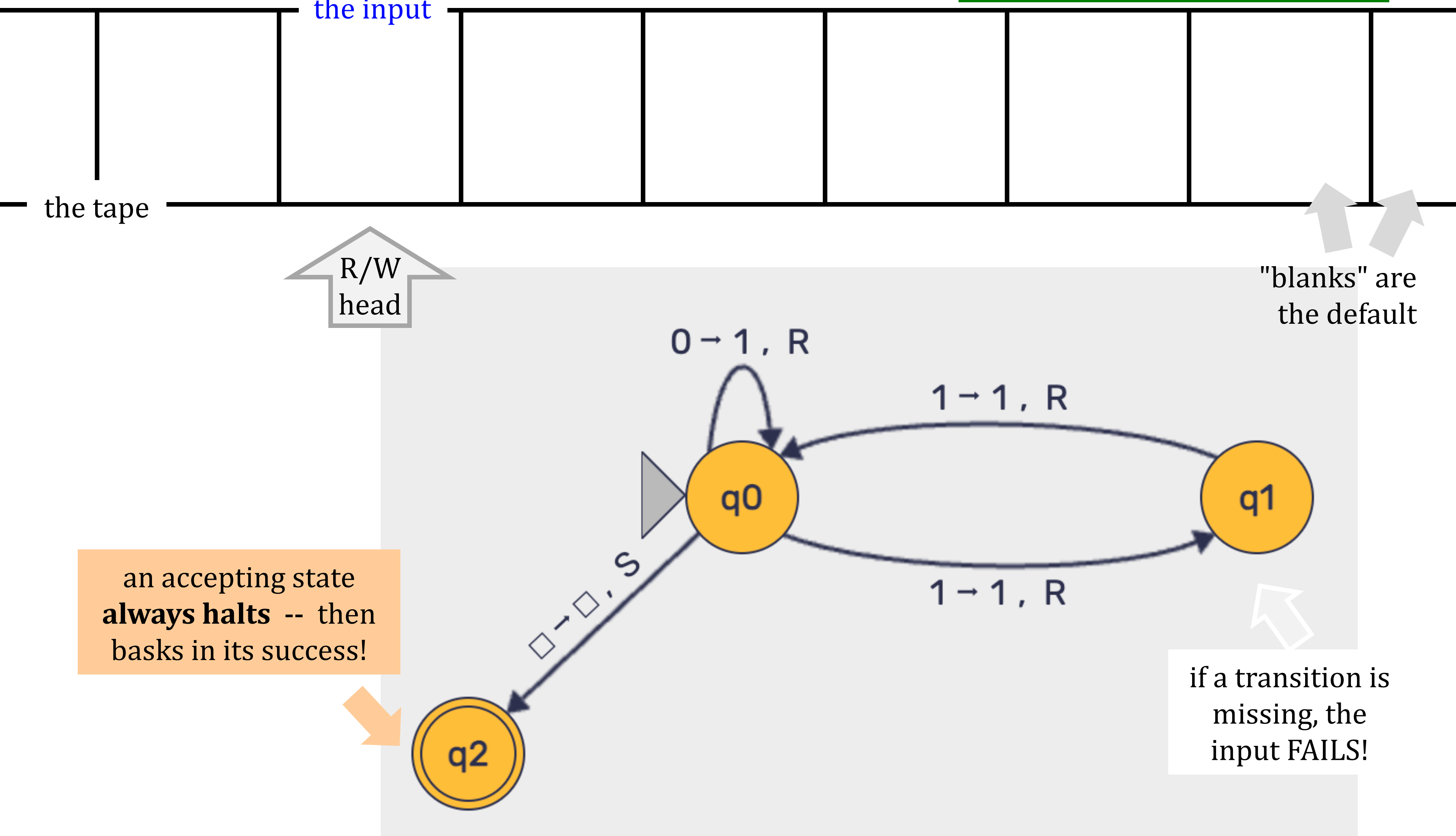
Water-based computation

Tinkertoy computation



Turing machine

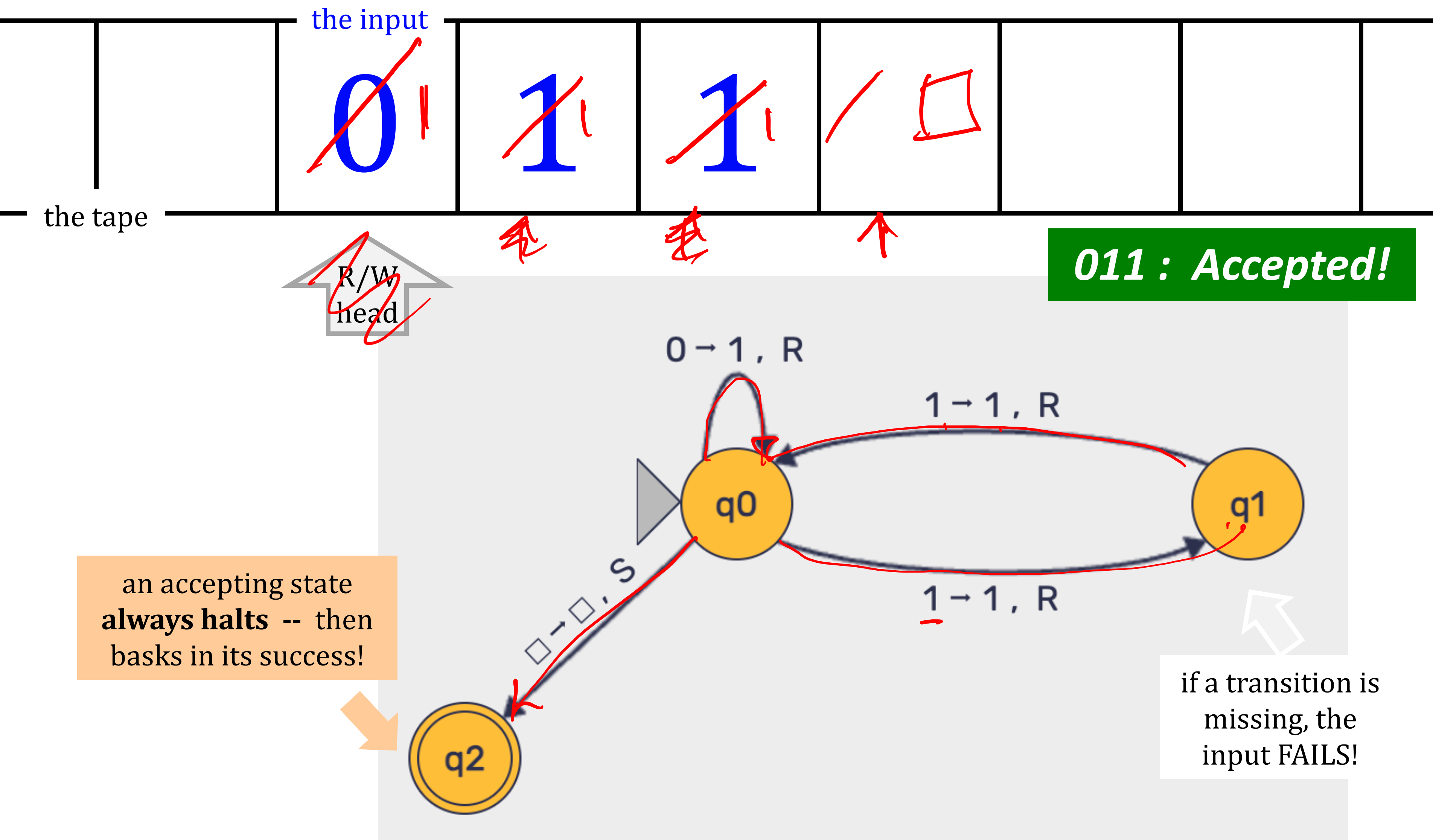
Empty input: Accepted!



a Turing Machine rule: 0 → 1 , R

READ WRITE MOTION

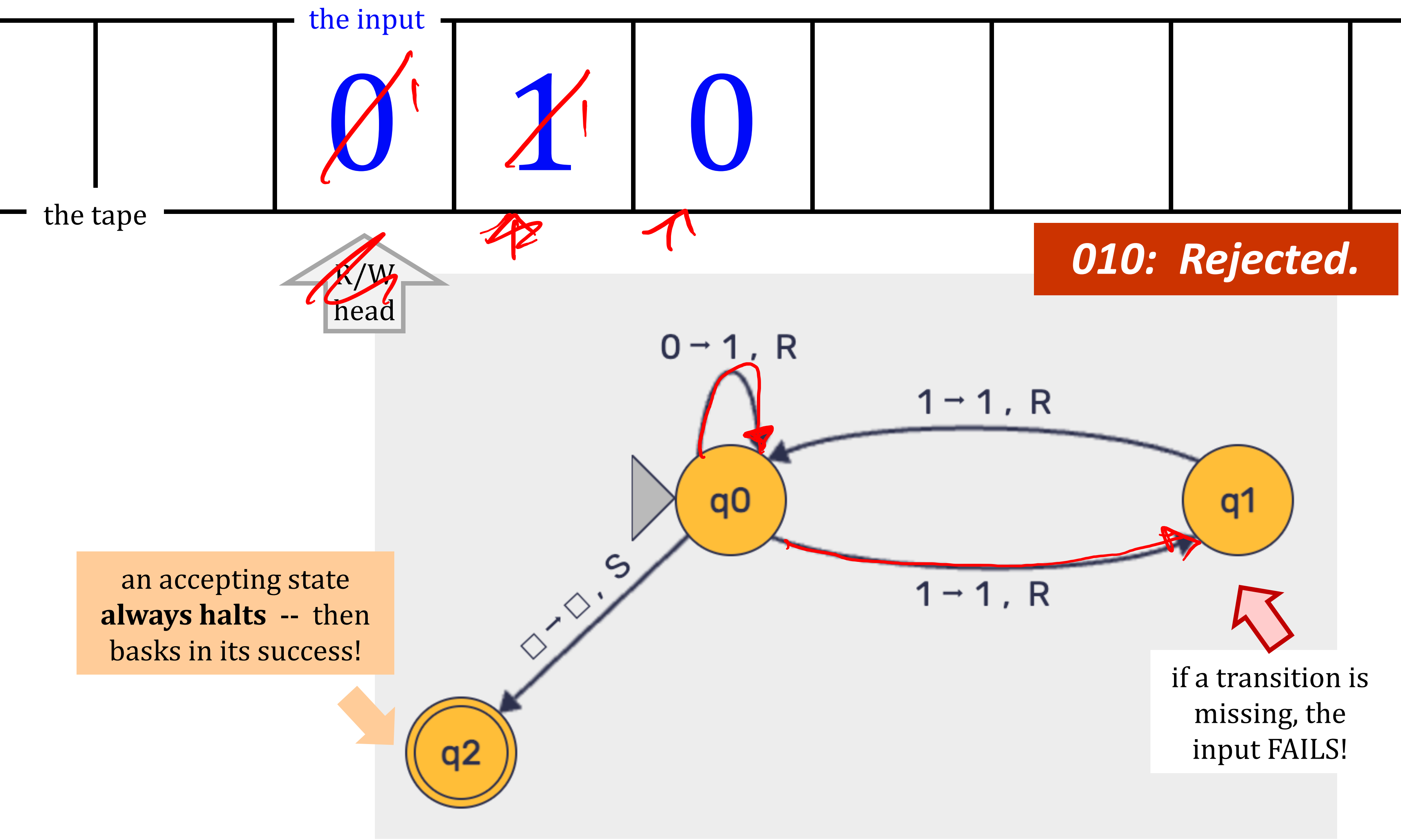
try it in JSFLAP...



a Turing Machine rule: $0 \rightarrow 1 \;,\; R$

READ WRITE MOTION

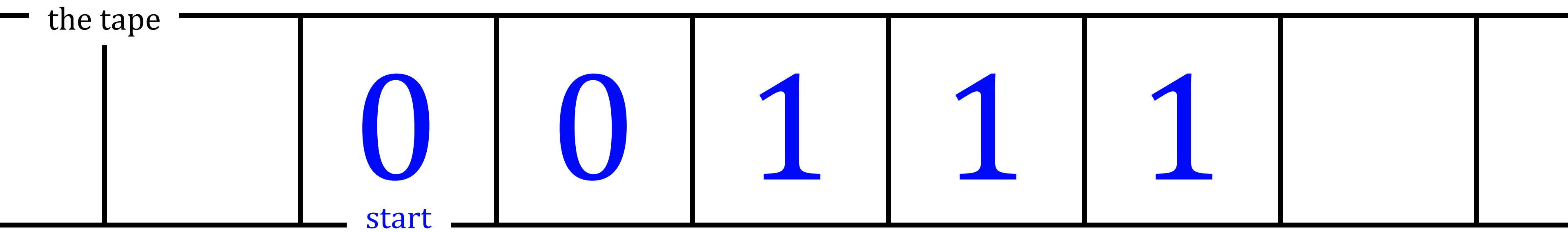
try it in JSFLAP...



a Turing Machine rule: $0 \rightarrow 1$, R

READ WRITE MOTION

try it in JSFLAP...

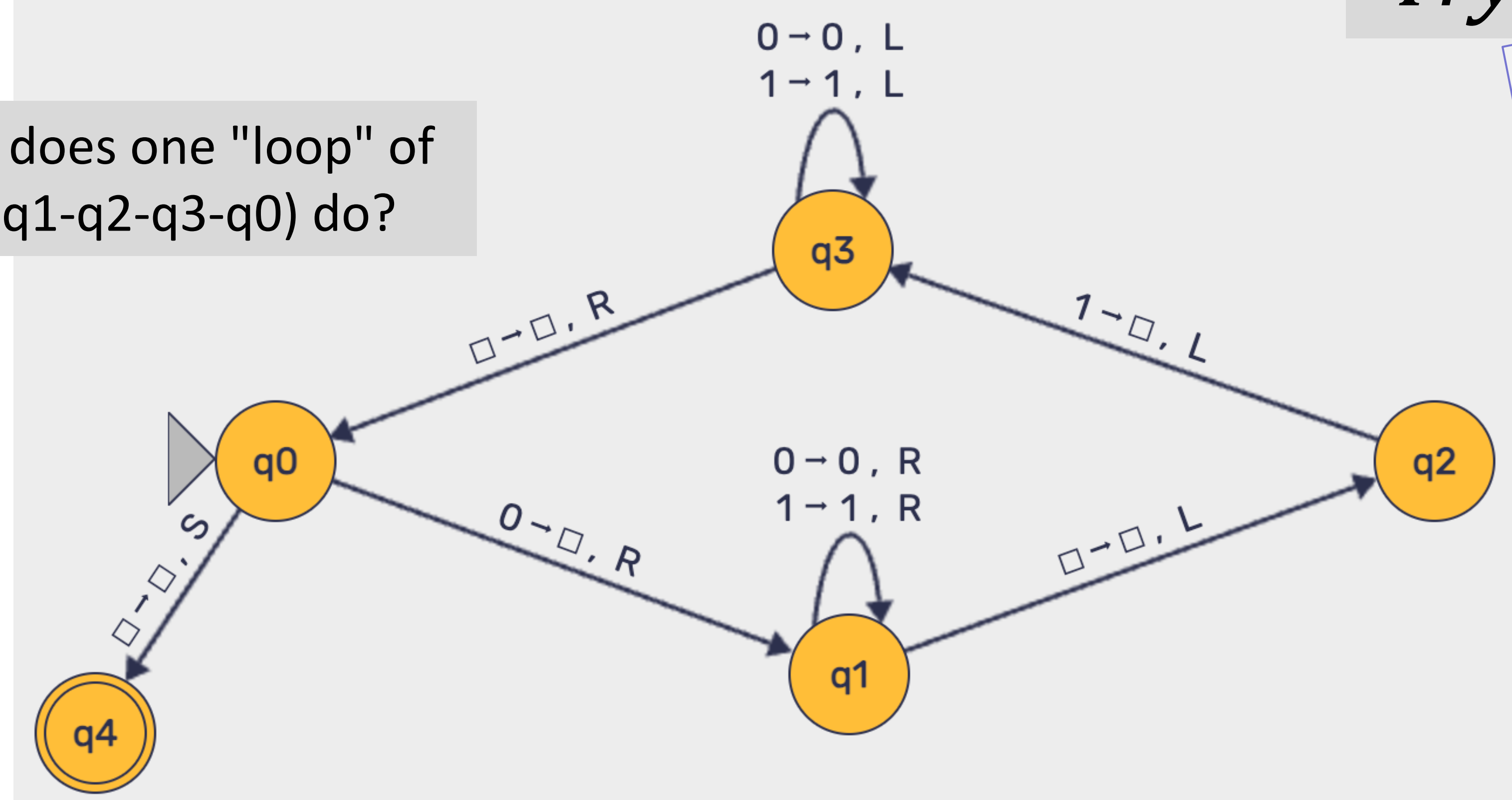


Is *this* input accepted or rejected by this TM?

Try it!

but not on this slide...

What does one "loop" of (q0-q1-q2-q3-q0) do?

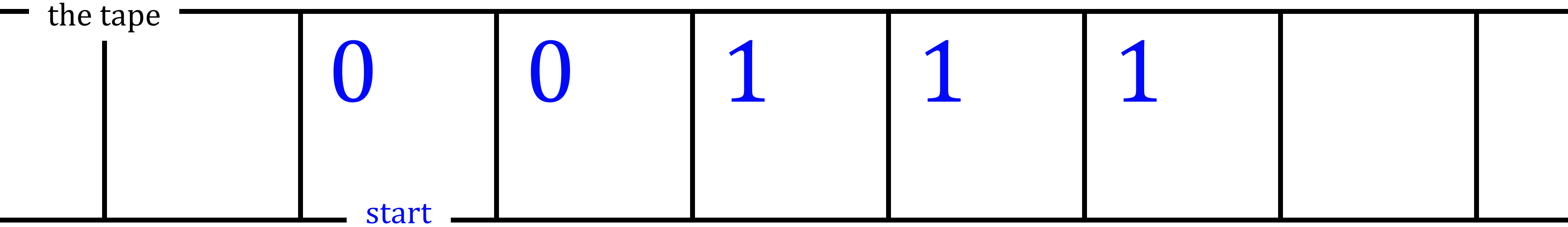


What inputs are accepted *in general*?

Extra: How could you change this TM to accept *palindromes?*

(this extra is both a thought-experiment and is ex. cr. in hw12)

Names: ____________________

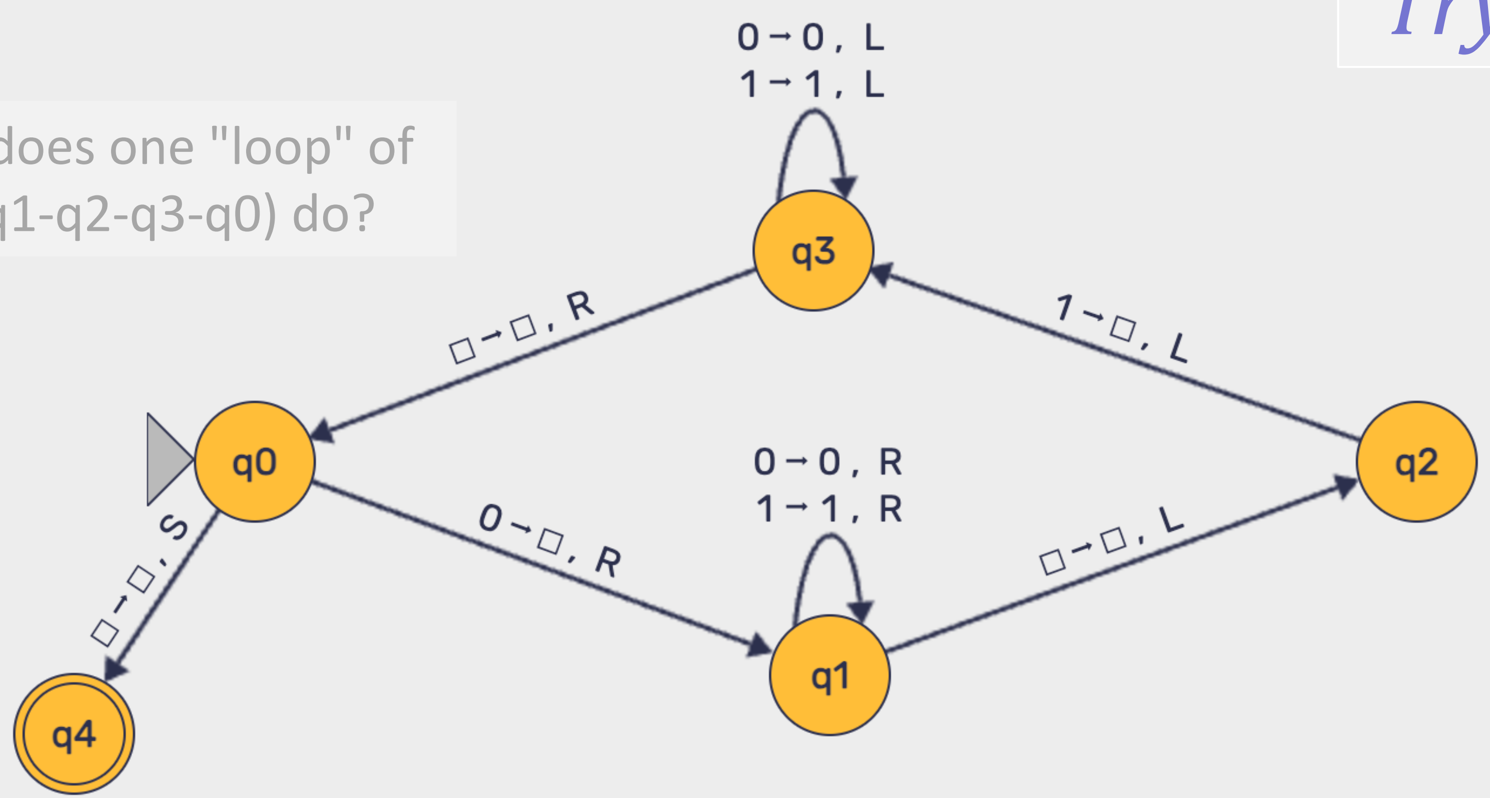


now, with room to work...

Is *this* input accepted or rejected by this TM?

Try it!

What does one "loop" of (q0-q1-q2-q3-q0) do?



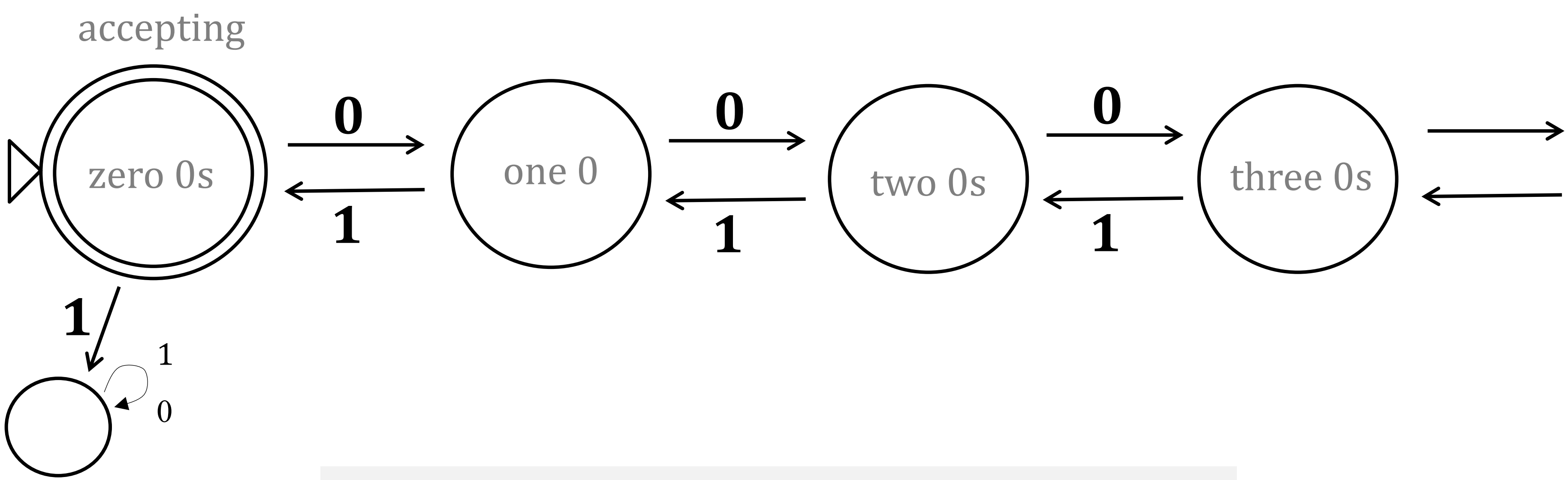
What inputs are accepted *in general*?

Extra: How could you change this TM to accept *palindromes?*

(this extra is <u>both</u> a thought-experiment <u>and</u> is ex. cr. in hw12)

State-machine ***limits***?

You don't need three eyes to see some problems here!



Let's build a FSM that accepts strings with ***any*** **# of 0s** followed by the ***same*** **#of 1s**

rejected

011

001

11100

00110

FSMs "can't count"

at least, not arbitrarily high

accepted

000111

0011

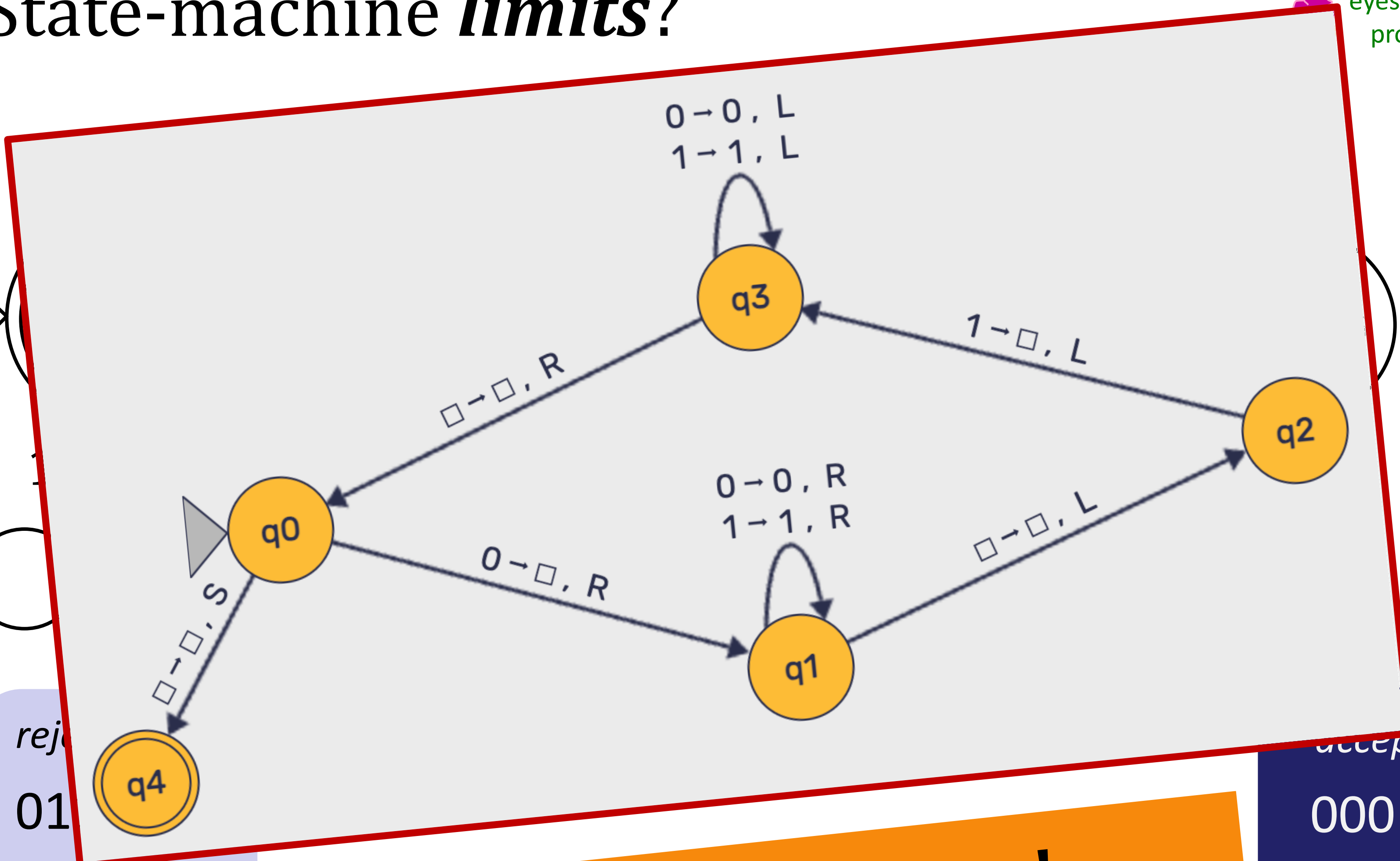
01

λ

but Python can count... What's going on?

State-machine *limits*?

You don't need three eyes to see some problems here!



rej

01

001

11100

00110

accepted

000111

0011

01

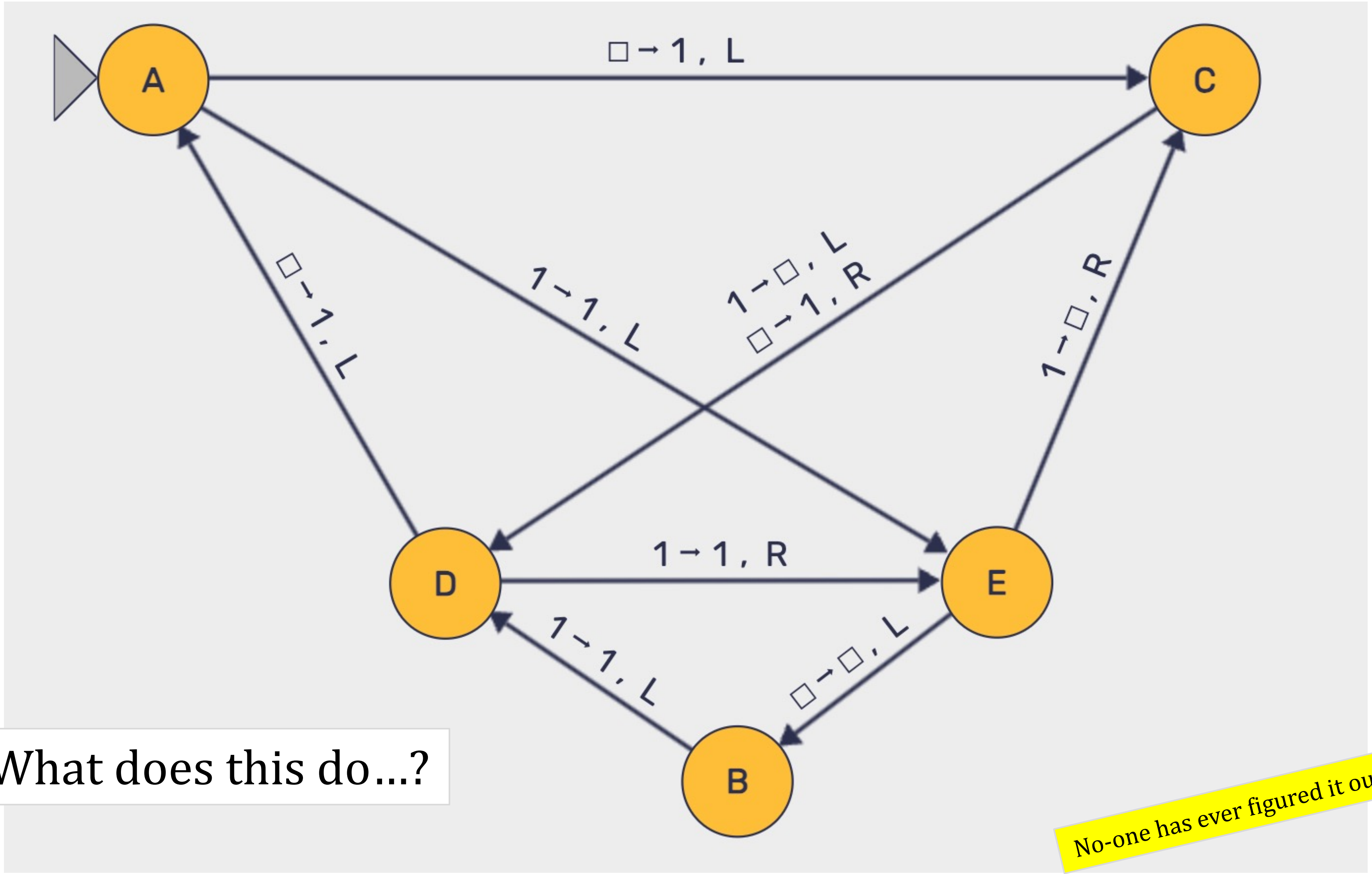
λ

but TMs can...!

including arbitrarily high...

but Python can count... Python's a TM! ...sort of...

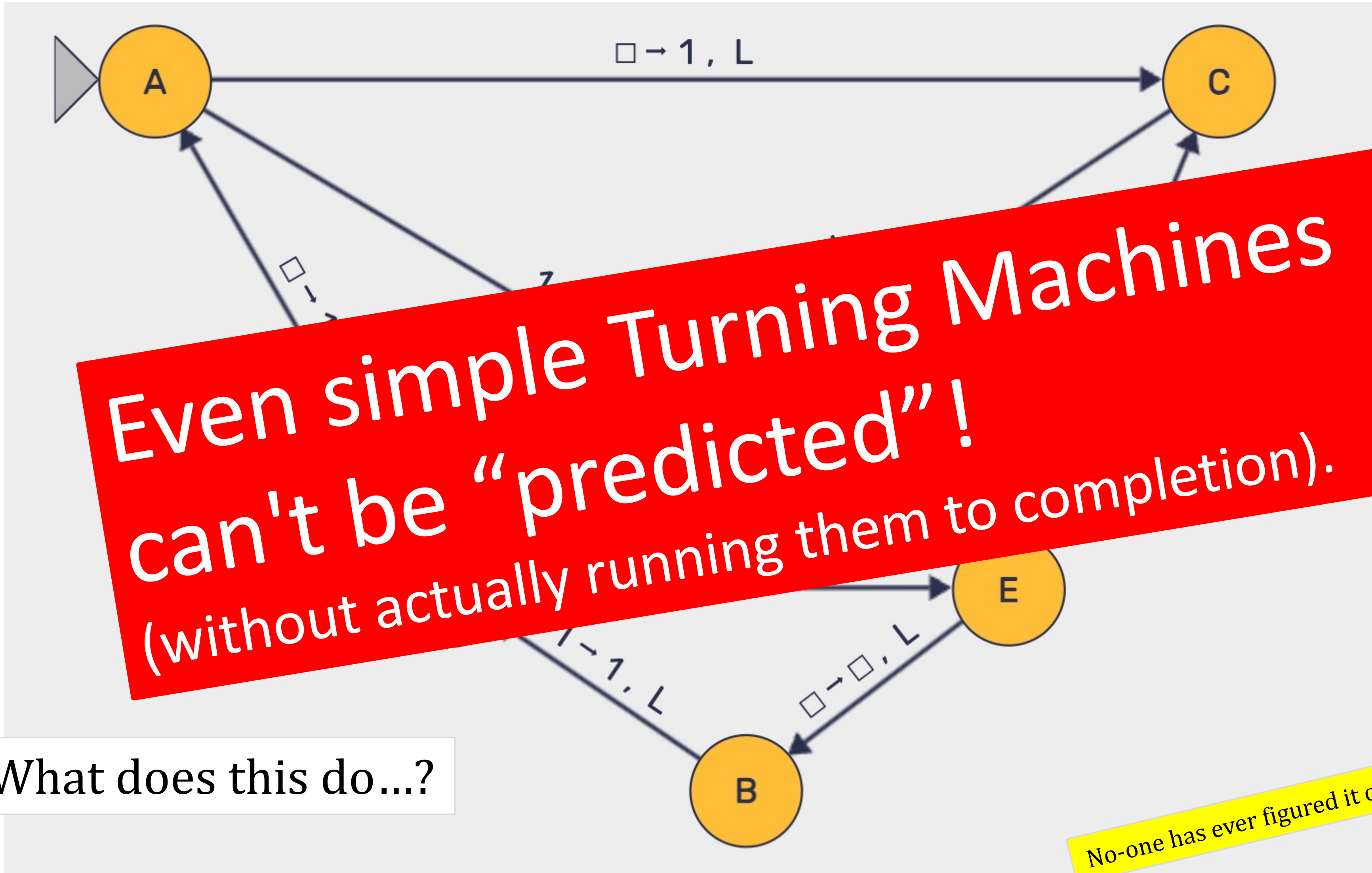
Turing Machine Limits?



What does this do…?

No-one has ever figured it out!

Turing Machine Limits?



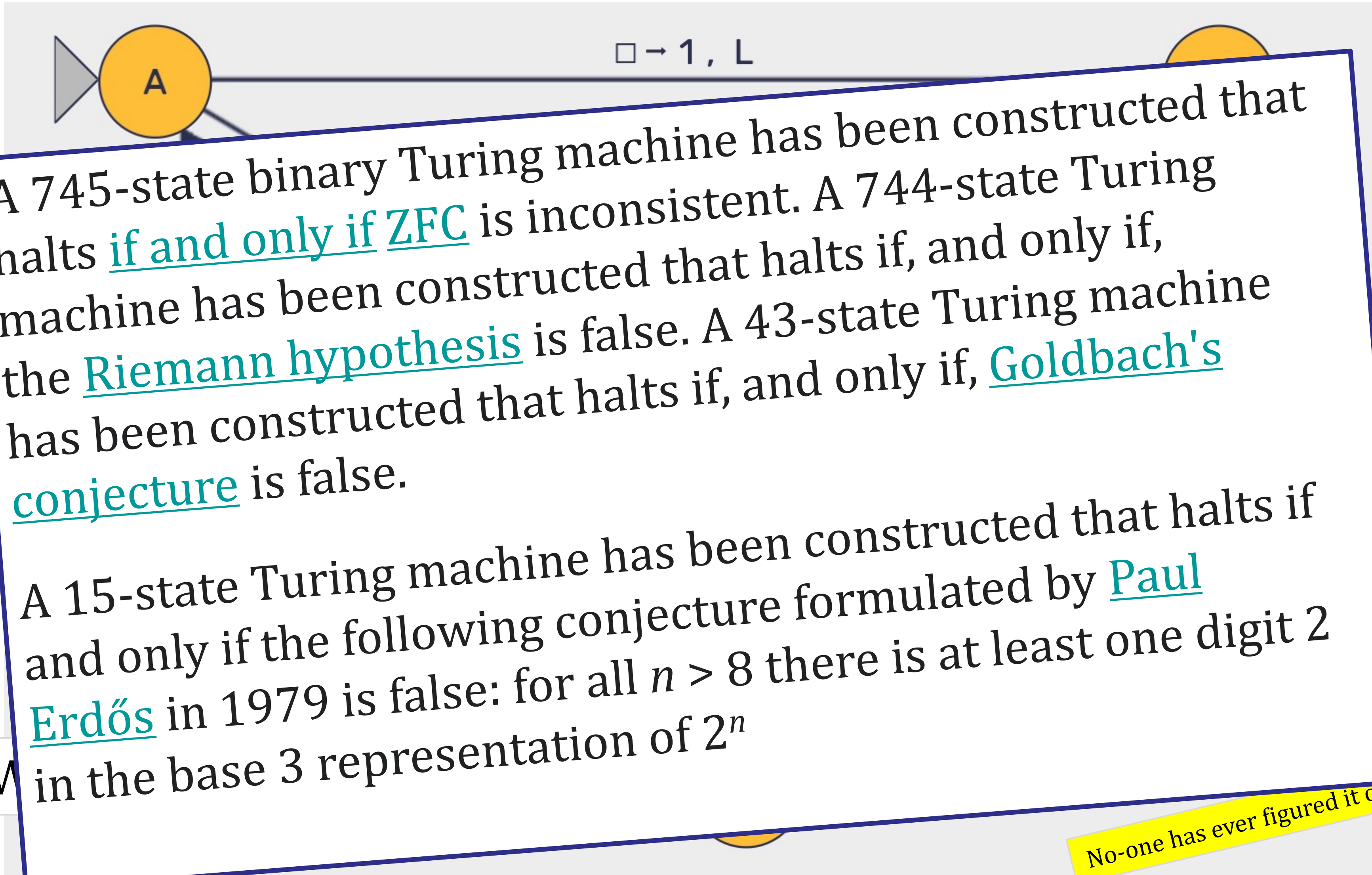
Even simple Turning Machines can't be “predicted”!

(without actually running them to completion).

What does this do…?

No-one has ever figured it out!

Turing Machine Limits?



A 745-state binary Turing machine has been constructed that halts if and only if ZFC is inconsistent. A 744-state Turing machine has been constructed that halts if, and only if, the Riemann hypothesis is false. A 43-state Turing machine has been constructed that halts if, and only if, Goldbach's conjecture is false.

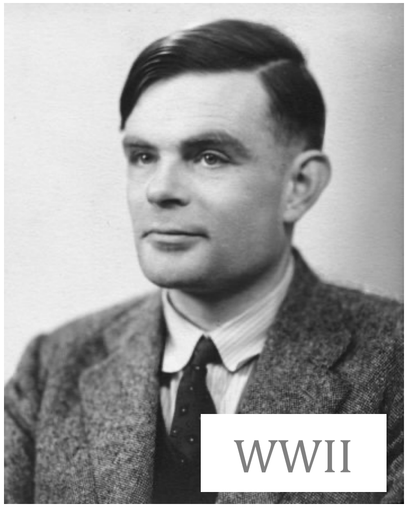
A 15-state Turing machine has been constructed that halts if and only if the following conjecture formulated by Paul Erdős in 1979 is false: for all $n > 8$ there is at least one digit 2 in the base 3 representation of 2^n

No-one has ever figured it out!

AI!

Alan Turing

1912-1954


WWII


Enigma machine ~ The axis's encryption engine

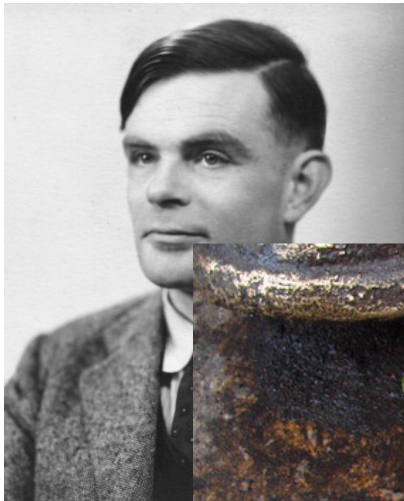

1946

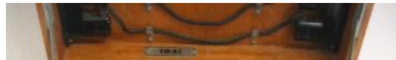

2007
Bletchley Park

Alan Turing

1912-1954

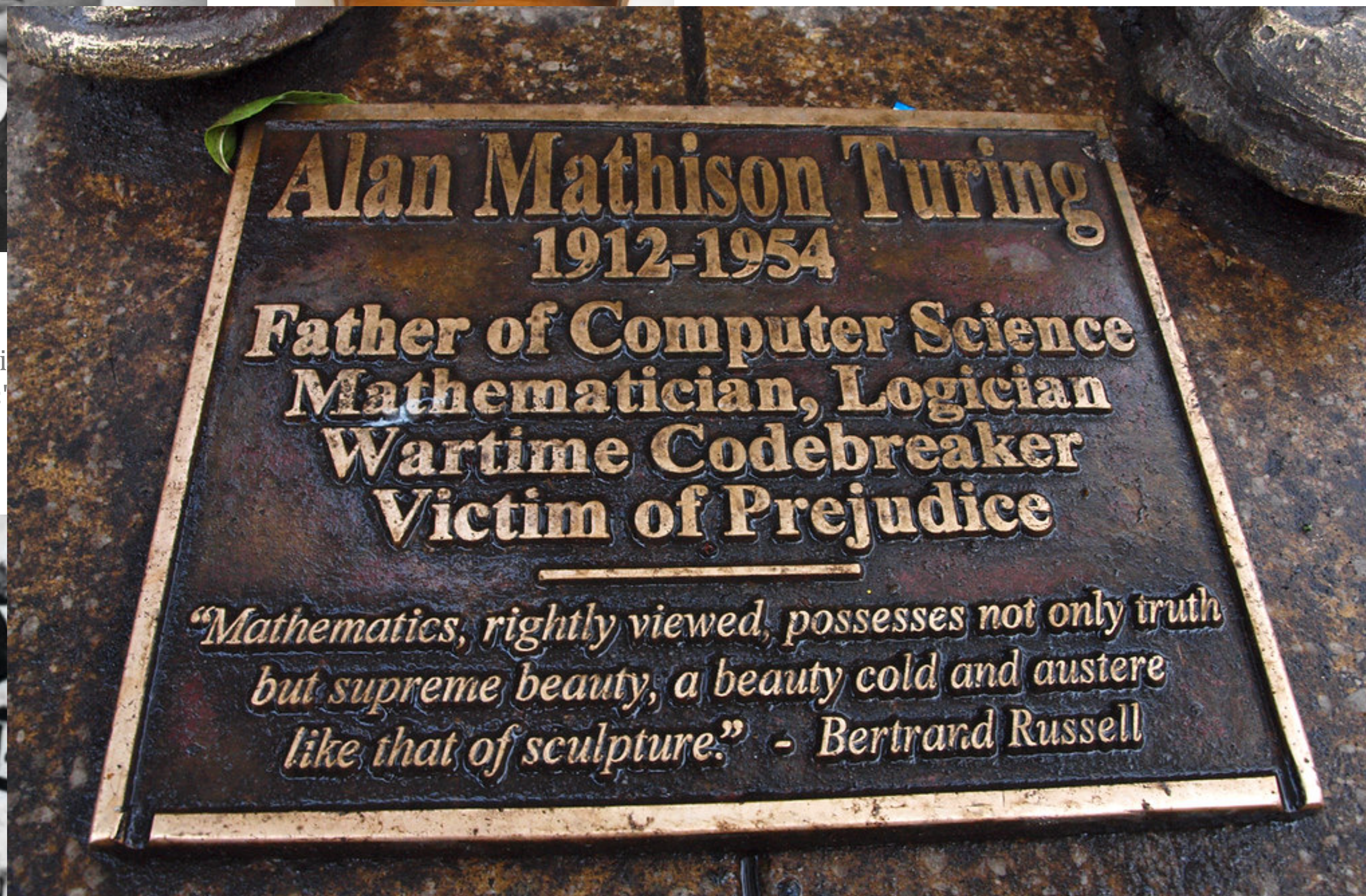
AI!





Eni

axis'



1946

2007

Bletchley Park

CS 5 spokesperson of the day!

Can TMs compete with other models?

Alan Turing says **yes**...

the tape elsewhere do not affect the behaviour of the machine. However the tape can b ... elementary opera ... ally have an innings.

Turing called them *Logical Computing Machines*

These machines will here be called 'Logical Computing Machines'. They are chiefly of interest when we wish to consider what a machine could in principle be designed to do, when we are willing to allow it both unlimited time and unlimited storage capacity.

Universal Logical Computing Machines. It is possible to describe L.C.Ms in a very standard way, and to put the description into a form which can be 'understood' (i.e. applied by) a special machine. In particular it is possible to design a 'universal machine' which is an L.C.M ... other L.C.M. is imposed on the otherw ... machine then set going it will carry ... whose description it was given. For details the reader must refer to Turing (1).

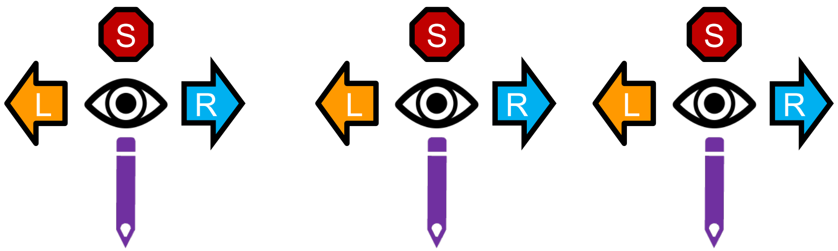
Turing's *Intelligent Machines, 1948*

The importance of the universal machine is clear. We do not need to have an infinity of different machines doing different jobs. A single one will suffice. The engineering problem of producing various machines for various jobs is replaced by the office work of 'programming' the universal machine to do these jobs.

It is found in practice that L.C.Ms can do anything that could be described as 'rule of thumb' or 'purely mechanical'. This is sufficiently well established that it is now agreed amongst logicians that 'calculable by means of an L.C.M.' is the correct accurate rendering of such phrases. There are several mathematically equivalent but superficially very different renderings.

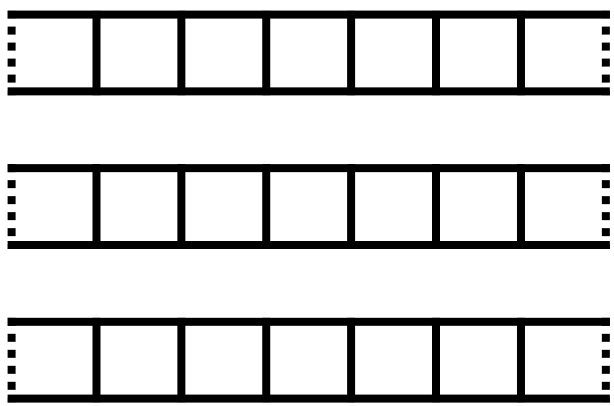
http://www.alanturing.net/turing_archive/archive/l/l32/L32-005.html

TheoComp loves to "hack" TMs



Multiple heads!

Multiple tapes!



Σ * ♡

A bigger alphabet!

But the computational power is <u>still</u> the same!

Can ~~TMs~~ computers compute *everything*?

Alan Turing says **No!**

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

decision problems!

By A. M. Turing.

[Received 28 May, 1936.—Read 12 November, 1936.]

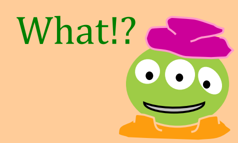
The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable *numbers*, it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are, however, the same in each case, and I have chosen the computable numbers for explicit treatment as involving the least cumbrous technique. I hope

http://www.cs.virginia.edu/~robins/Turing_Paper_1936.pdf

... there are many problems computers can't solve at all !

Perhaps this is not that surprising...

- rising sea levels
- disbelief in aliens
- losing to your own Connect4 (at 0 ply!)
- crashfree driving and fast towel folding!



... but **`int-bool`** functions!?

in ⟶ out

Unprogrammable *functions?*

There are well-defined mathematical functions that no computer program can compute

even `int-bool`

or TM

even with ***any*** *amount of memory!*

how? why?

functions

programs

$$f(x) = \begin{cases} 1 & \text{if x is odd} \\ 0 & \text{if x is even} \end{cases}$$

```
def prog1(x):
    return x%2
```

Unprogrammable *functions!*

There are

well-defined mathematical functions

that no

computer program

or TM[TM]

can compute

even with ***any*** *amount of memory!*

There are <u>many</u> more functions

than programs!

Real #'s vs. Natural #'s

There are well-defined mathematical functions that no computer program (or TM™) can compute *even with **any** amount of memory!*

There are <u>many</u> more functions than programs!

$\mathbb{R}$ ***vs.*** $\mathbb{N}$

functions **int - bool**

$$f_A(x) = 1$$

$$f_B(x) = \begin{cases} 1 & \text{if x is odd} \\ 0 & \text{if x is even} \end{cases}$$

$$f_C(x) = \begin{cases} 1 & \text{if x is 0, 1, or 2} \\ 0 & \text{otherwise} \end{cases}$$

Some example ***"int-bool"*** mathematical functions

These are also sometimes called *"integer predicates."*

- Input is an integer, x >= 0
- Output is 0/1 (*boolean or bit*)

even if we only use functions w/ input ints + output bools!

int - bool programs

```
def prog1(x):
   return x%2
```

all int inputs: `x >= 0`

```
def prog2(x):
   return x<3
```

```
def prog3(x):
   return 1
```

```
def prog4(x):
   return len(str(x+42))>1
```

Example programs

- Input is one integer, **`x >= 0`**
- Output is **0/1** (*boolean or bit*)

If programs <u>look different</u> they are different – *even if they compute the same function!*

... and even if we allow ANY programs at all ~ *even syntax errors*

Let's match!

functions

$$f_A(x) = 1$$

$$f_B(x) = \begin{cases} 1 & \text{if x is odd} \\ 0 & \text{if x is even} \end{cases}$$

$$f_C(x) = \begin{cases} 1 & \text{if x is 0, 1, or 2} \\ 0 & \text{otherwise} \end{cases}$$

1. Match each program with the function it computes.

2. There are three different functions on the left side -- how many *different programs* are in the right side?

*How – or why – is the set of **all functions** larger than the set of **all programs** ?*

programs

```
def prog1(x):
  return x%2
```

all int inputs:
`x >= 0`

```
def prog2(x):
  return x<3
```

```
def prog3(x):
  return 1
```

```
def prog4(x):
  return len(str(x+42))>1
```

```
def prog5(x):
  return x in [0,1,2]
```

```
def prog6(x):
  if x<2: return x
  else: return prog6(x-2)
```

functions

$f_A(x) = 1$

$$f_B(x) = \begin{cases} 1 & \text{if x is odd} \\ 0 & \text{if x is even} \end{cases}$$

$f_C(x) = \begin{cases} 1 & \text{if x is 0, 1, or 2} \end{cases}$

There are *many* more functions

$\mathbb{R}$

1. M... mputes.

2. The ...
how m... are in the ...

*How – or why – is the set of **all fu...***
*larger than the set of **all programs** ?*

programs

```
def prog1(x):
   return x%2
```

all int inputs:
`x >= 0`

```
def prog2(x):
   return x<3
```

```
def prog3(x):
   return 1
```

```
def pr...
      ...)>1
```

```
de...
```

than programs!

$\mathbb{N}$

```
def prog6(x):
    if x<2: return x
    else: return prog6(x-2)
```

Quiz! Name(s): ____________________

Programs are, like, integers…

```
def alien( x ):
  if x == 42:
    return True
  else:
    return not \
         alien(x+1)
```

`programs`

For each program, there is an integer.

For each integer, there is a "program."

This is true:

how so?

from **progs** to **ints** ~ and *back*...

```
def prog(i):
    """ return the program whose int is i """
    # convert to a string (just a base-128 int!)
    if i <= 0: return ''
    last_char = chr(i % 128)
    return prog(i // 128) + last_char


def intify(prog):
    """ return the int whose program is prog """
    # convert to an int (just interpret as base-128!)
    if prog == '': return 0
    last_char = prog[-1]
    return 128 * intify(prog[:-1]) + ord(last_char)
```

```
In [7]: eval(prog(6706))
Out[7]: 42

In [8]: eval(prog(1229340410842616847622082154303469913707433))
Hello World
```

Programs are, like, integers…

```
def alien( x ):
  if x == 42:
    return True
  else:
    return not \
         alien(x+1)
```

programs = $\mathbb{N}$ Positive integers

Every program is a **string**.

Every string is just a sequence of **bits**

Every sequence of bits is also an **int**!

To infinity - *and beyond*

Pixar owes Cantor for this one!

Countably infinite:

If you give me an **element** from the set, I can tell you a **finite counting number** it corresponds to!

$\mathbb{N}$ Positive integers

`programs`

Lots of things are countable…

	0	1	2	3	4	5	
Integers	0	1	-1	2	-2	3	…
Odd integers	1	-1	3	-3	5	-5	…
Strings with only 'a's and 'b's	''	'a'	'b'	'aa'	'ab'	'ba'	…
Fractions	0/1	1/1	0/2	1/2	2/2	0/3	…

Hint: it's okay to repeat!

Lots of things are countable...

	0	1	2	3	4	5	
Integers	0	1				3	...
i							...
String only 'a's				'aa'	'ab'	'ba'	...
Fractions	0/1	1/1	0/2	1/2	2/2	0/3	...

What counts as countable:
if you give me an item (e.g. 'aaaabbbaabbaabbbbba'), then I can tell you its index!

Hint: it's okay to repeat!

functions vs. programs !

the Reals
from 0 to 1 $\mathbb{R}$

$\mathbb{N}$ Positive
integers

functions

vs.

`programs`

To infinity - *and beyond*

Pixar owes Cantor for this one!

the Reals from 0 to 1 $\mathbb{R}$ > $\mathbb{N}$ Positive integers

functions > `programs`

Uncountably infinite > **Countably** infinite

There are ***lots of*** functions

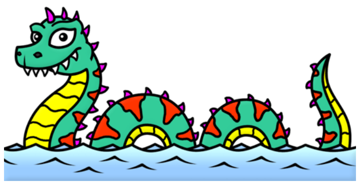
uncountably many ~ "big"

Well-defined mathematical

`(int-bool)` functions

(uncountable)

$\mathbb{R}$



What's swimming around out here?

$\mathbb{N}$

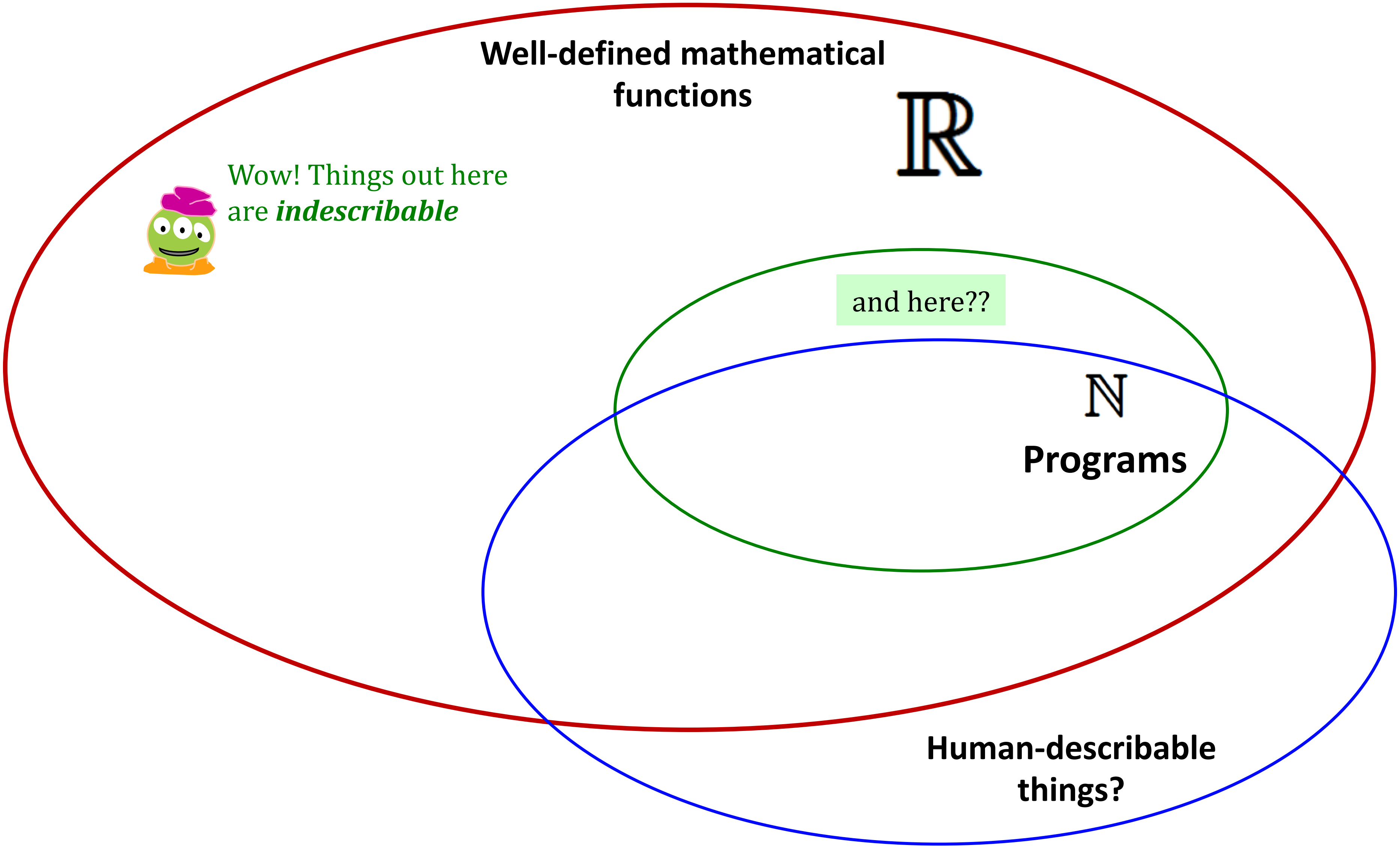
Programs

(countable)

There are ***"a few"*** programs

countably many

How about an *actual example* !?!!



How about an *actual example* !?!!