Turing Machines

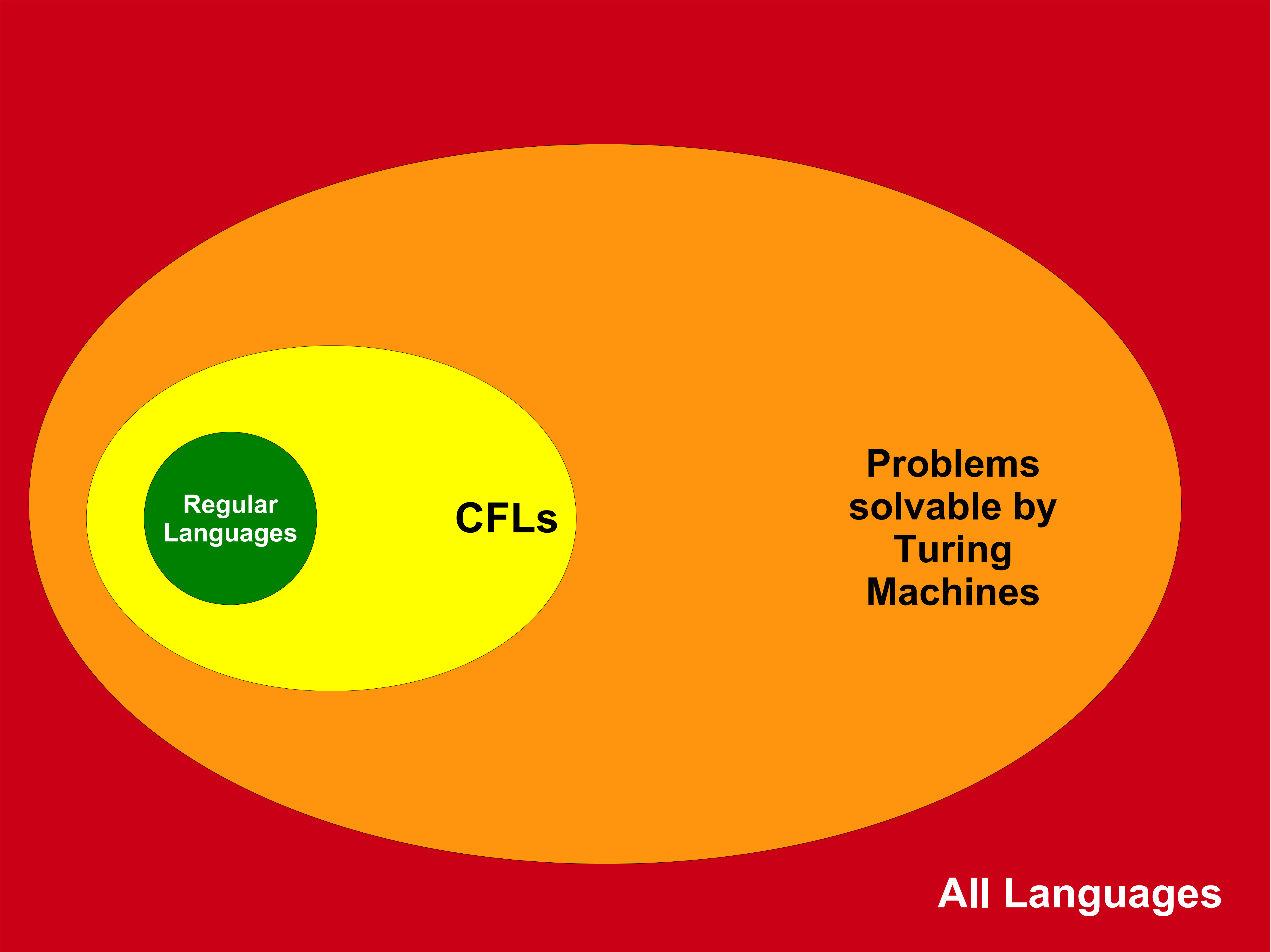
Part Three

Last Time: ***How powerful are Turing machines?***

The ***Church-Turing Thesis*** claims that

every effective method of computation is either equivalent to or weaker than a Turing machine.

> “This is not a theorem – it is a falsifiable scientific hypothesis. And it has been thoroughly tested!”
>
> \- Ryan Williams



Regular Languages

CFLs

Problems solvable by Turing Machines

All Languages

New Stuff!

Strings, Languages, Encodings, and Problems

What problems can we solve with a computer?

What kind of
computer?

What problems can we solve with a computer?

What is a
“problem?“

Decision Problems

- A ***decision problem*** is a type of problem where the goal is to provide a yes or no answer.
- Example: Bin Packing

You're given a list of patients who need to be seen and how much time each one needs to be seen for. You're given a list of doctors and how much free time they have. Is there a way to schedule the patients so that they can all be seen?

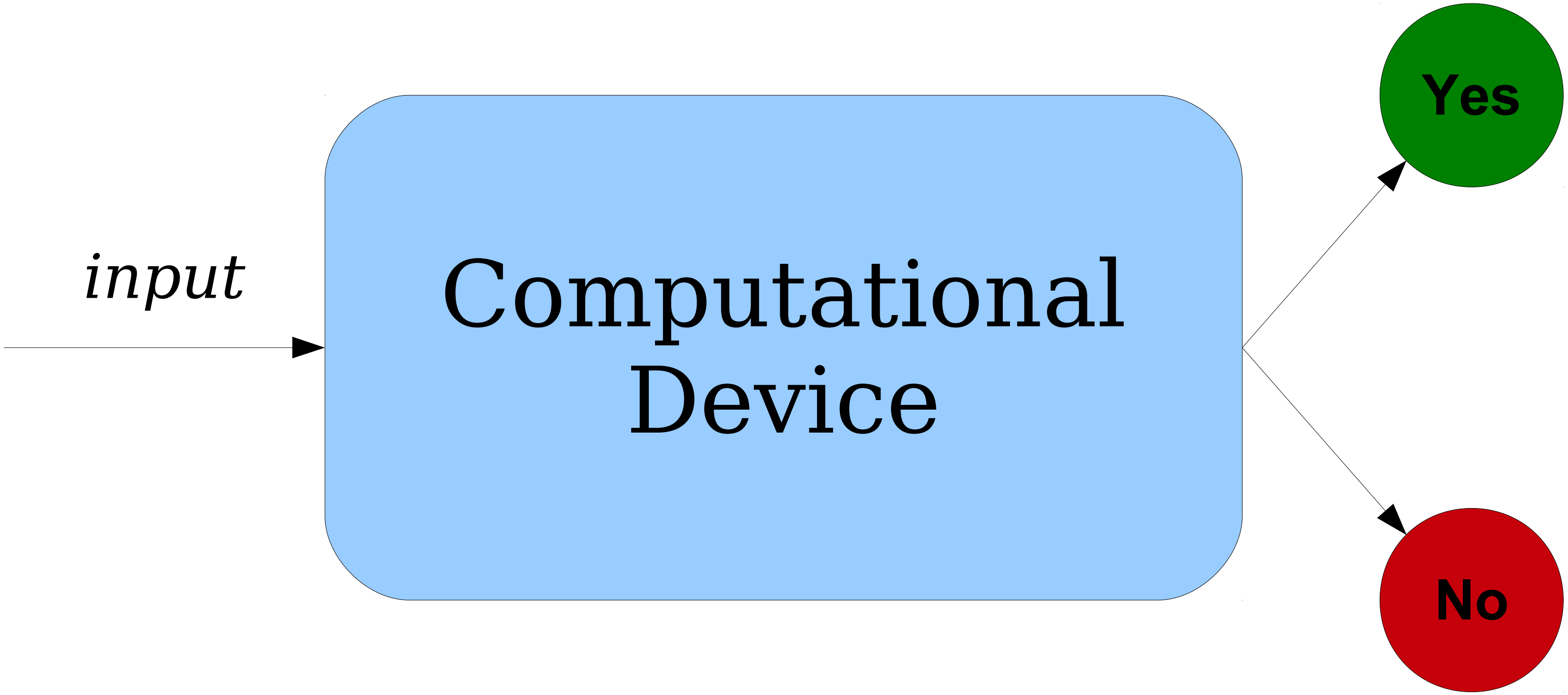
- Example: Dominating Set Problem

You're given a transportation grid and a number k. Is there a way to place emergency supplies in at most k cities so that every city either has emergency supplies or is adjacent to a city that has emergency supplies?

- Example: Route Planning

You're given the transportation grid of a city, a start location, a destination location, and information about the traffic over the course of the day. Given a time limit T, is there a way to drive from the start to the end locations in at most T hours?

Solving Decision Problems



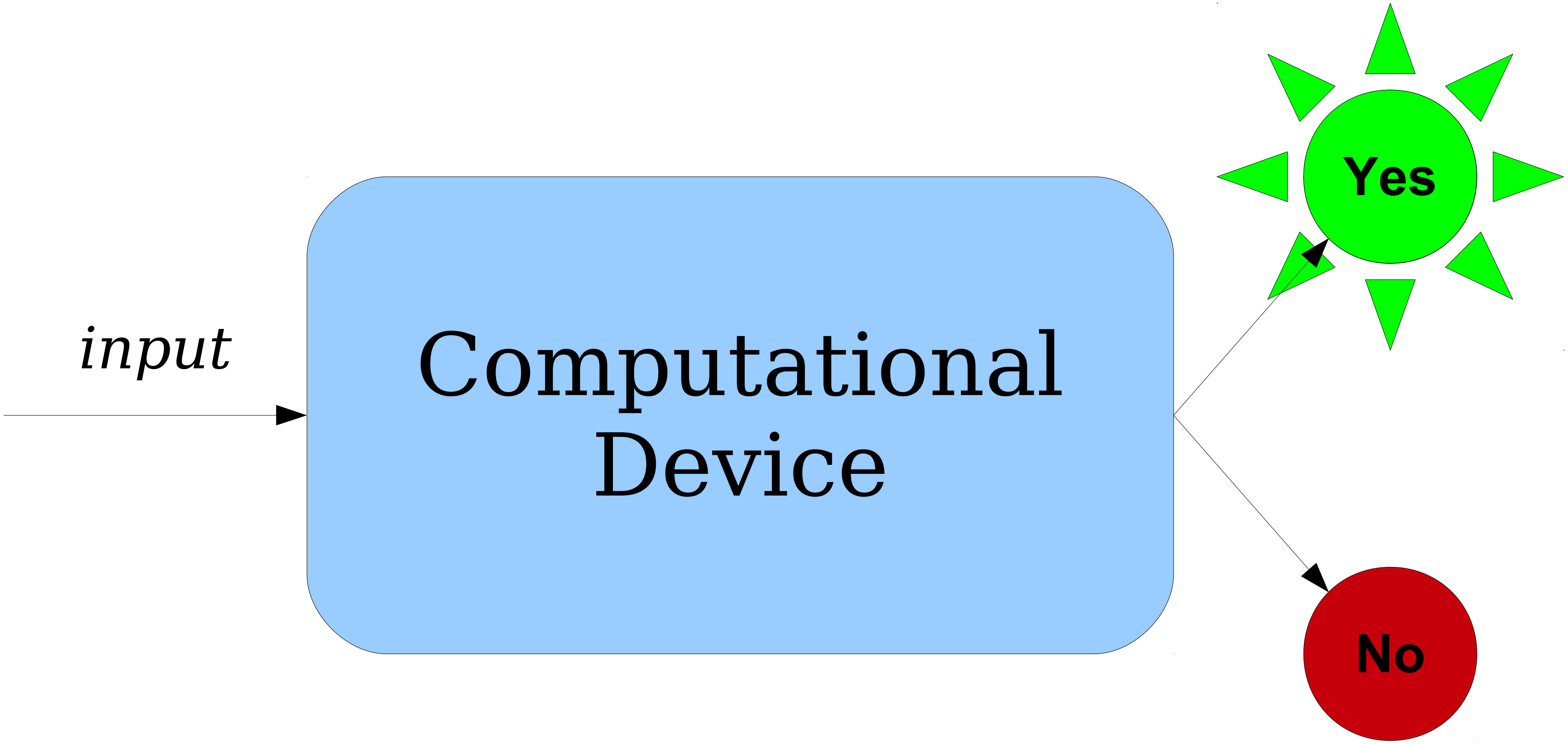
input

Computational Device

Yes

No

Solving Decision Problems



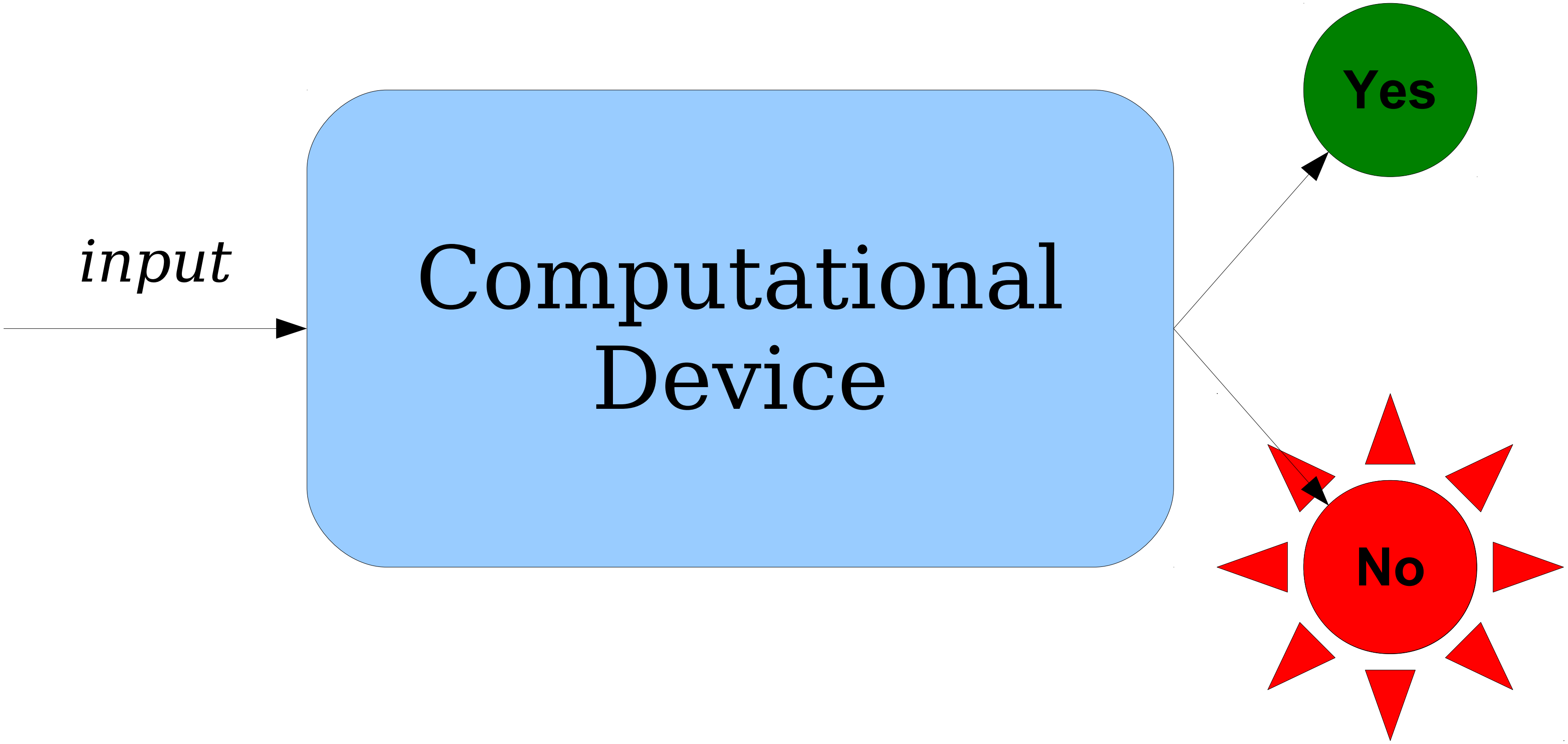
input

Computational Device

Yes

No

Solving Decision Problems



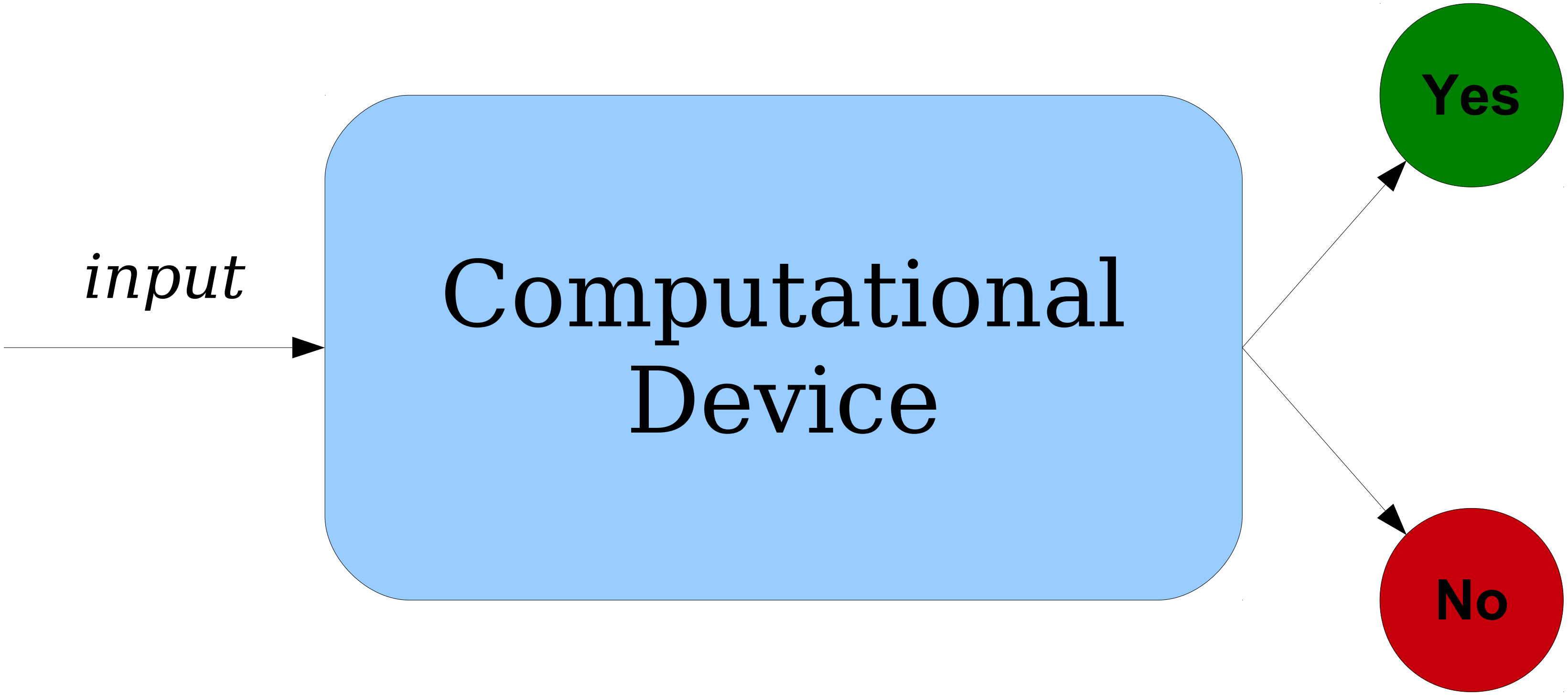
input

Computational Device

Yes

No

Solving Decision Problems



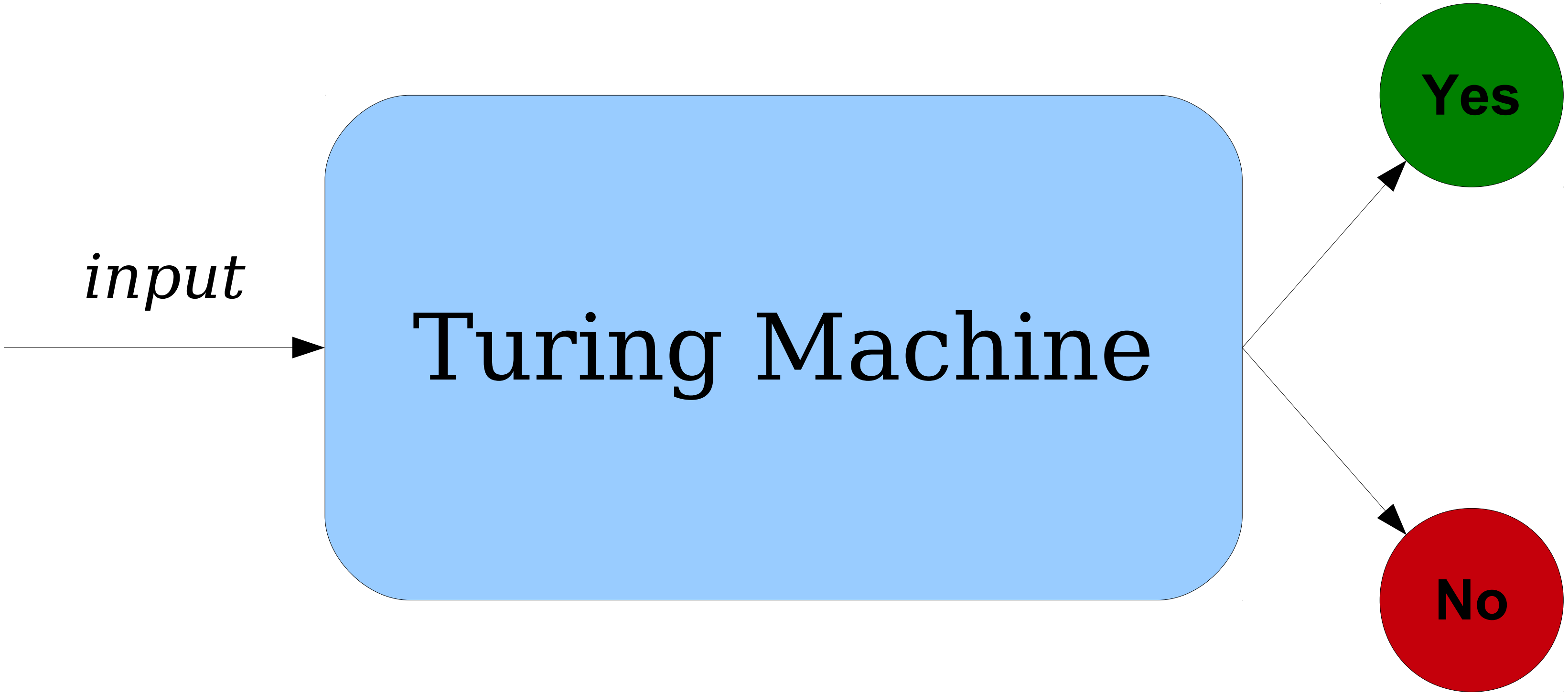
input

Computational Device

Yes

No

Solving Decision Problems



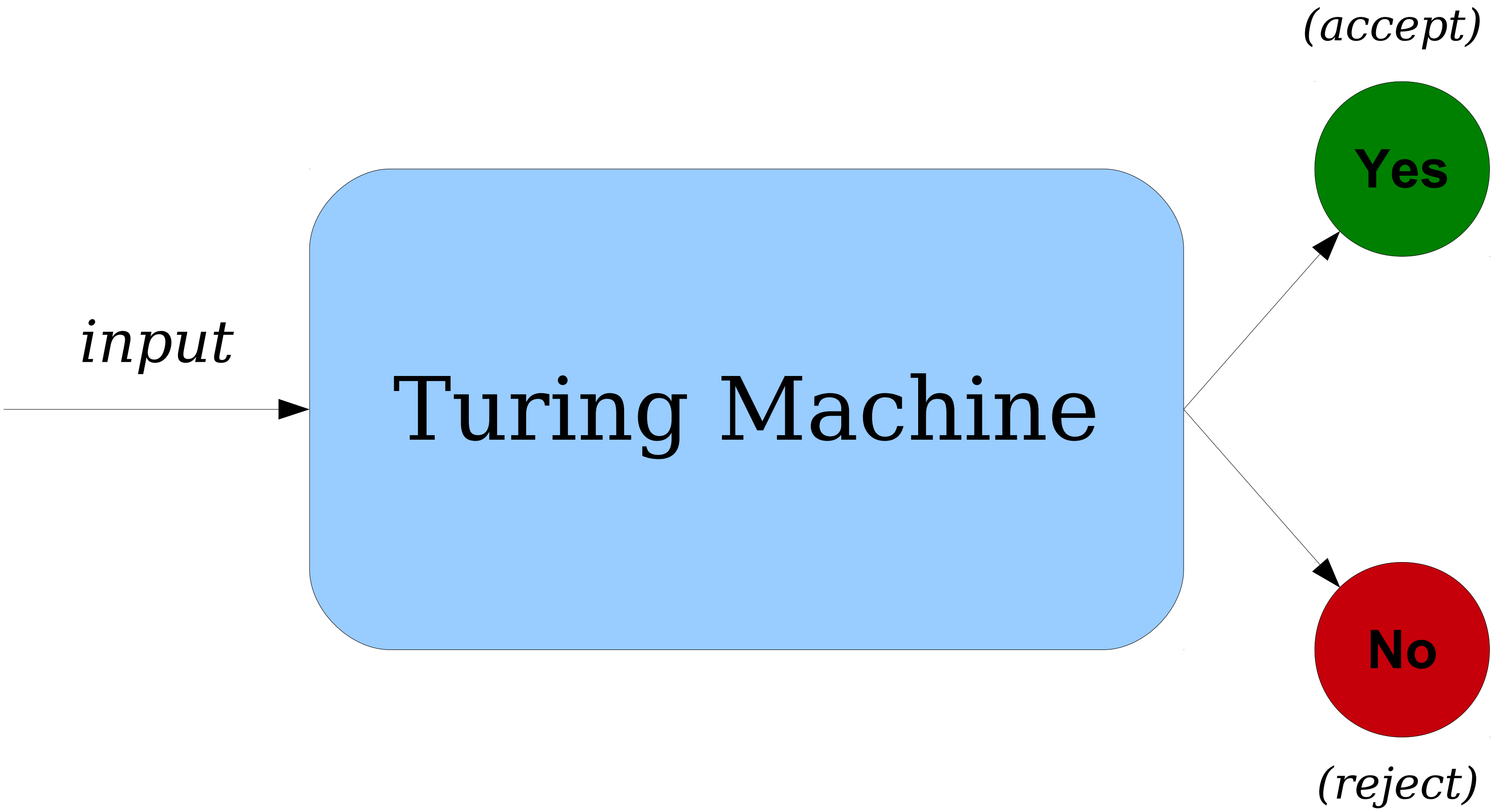
input

Turing Machine

Yes

No

Solving Decision Problems



(accept)

input

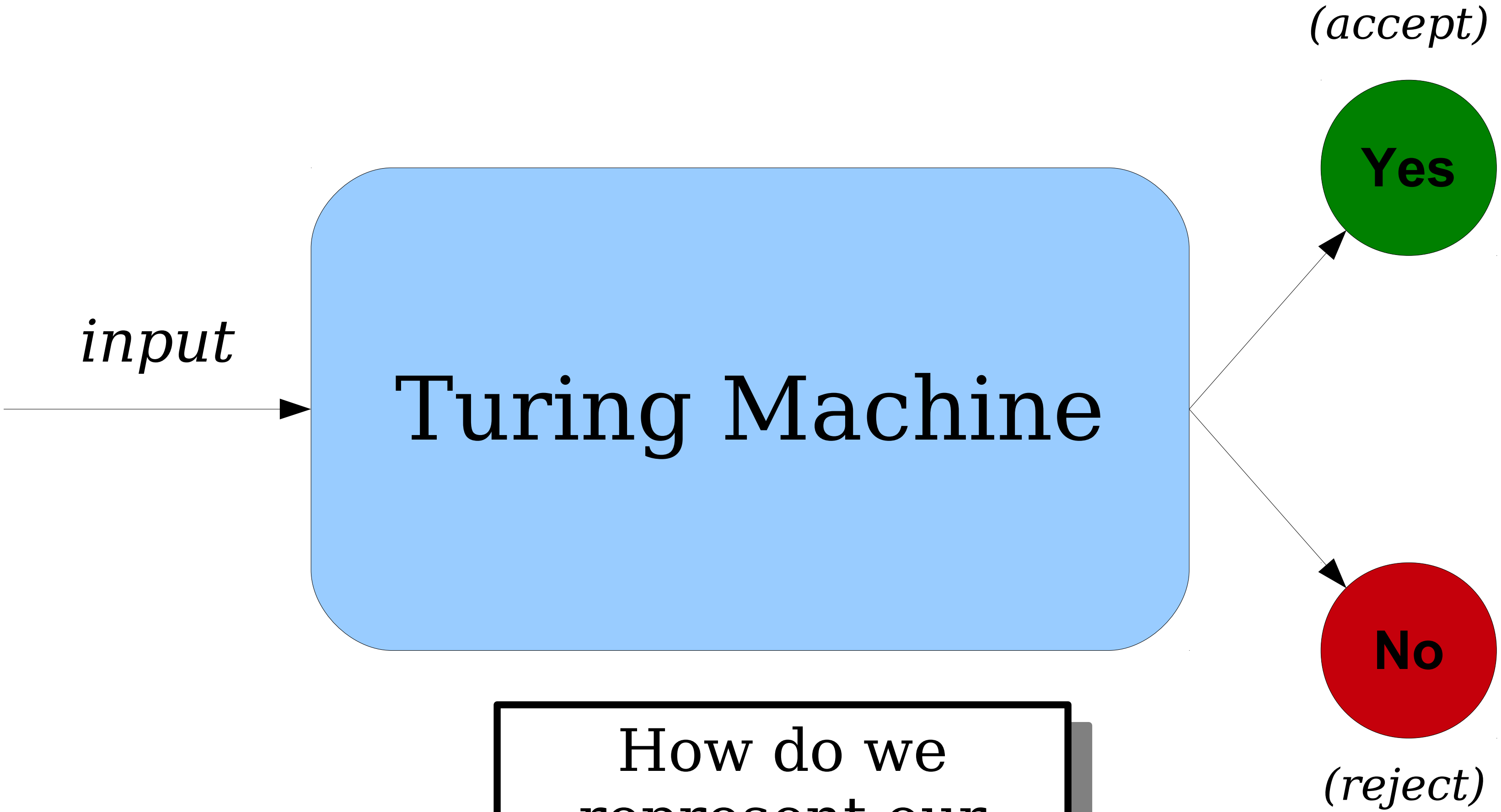
Turing Machine

Yes

No

(reject)

Solving Decision Problems



How do we represent our inputs?

An Observation

- We've seen several TMs that answer questions about numbers.
- Our first TM was for { $\mathtt{0}^n\mathtt{1}^n \mid n \in \mathbb{N}$ }, where the number n was represented by writing out some number of copies of a symbol.
- Later, we designed TMs that worked on decimal representations of integers.
- The computers we use every day work with numbers represented in binary.
- On Problem Set Seven, you’re using regular expressions to compute on Roman numerals.

An Observation

- There is a distinction between the mathematical object “the number 24” and the different ways of representing it.
- Each of the following denotes one way to write the number 24:
  - 24 (decimal)
  - XXIV (Roman numerals)
  - 18 (hexadecimal)
  - 11000 (binary)
  - 卌卌卌卌 |||| (tally marks)
  - 二十四 (Chinese numerals)
  - כ"ד (Hebrew numerals)
  - ٢٤ (Arabic numerals)
- Computers are powerful enough to convert any of these formats into any of these other formats. In a sense, what matters more is *what number we're working* with rather than *how that number is represented*.

An Observation

- Imagine that you’re implementing this method:

```
private boolean isEvenNumber(int n) {
    /* … some code … */
}
```

- As a programmer, you don't need to know how the integer `n` is represented internally in the computer in order to write a working implementation.
- Knowing how `n` is represented might be useful for implementing this method *efficiently*, but it's certainly not *necessary* for an implementation to use properties of that representation.

Strings and Objects

- Think about how my computer encodes the image on the right.
- Internally, it's just a series of zeros and ones sitting on my hard drive.
- All data on my computer can be thought of as (suitably-encoded) strings of 0s and 1s.

Strings and Objects

- A different sequence of 0s and 1s gives rise to the image on the right.
- Every image can be encoded as a sequence of 0s and 1s, though not all sequences of 0s and 1s correspond to images.

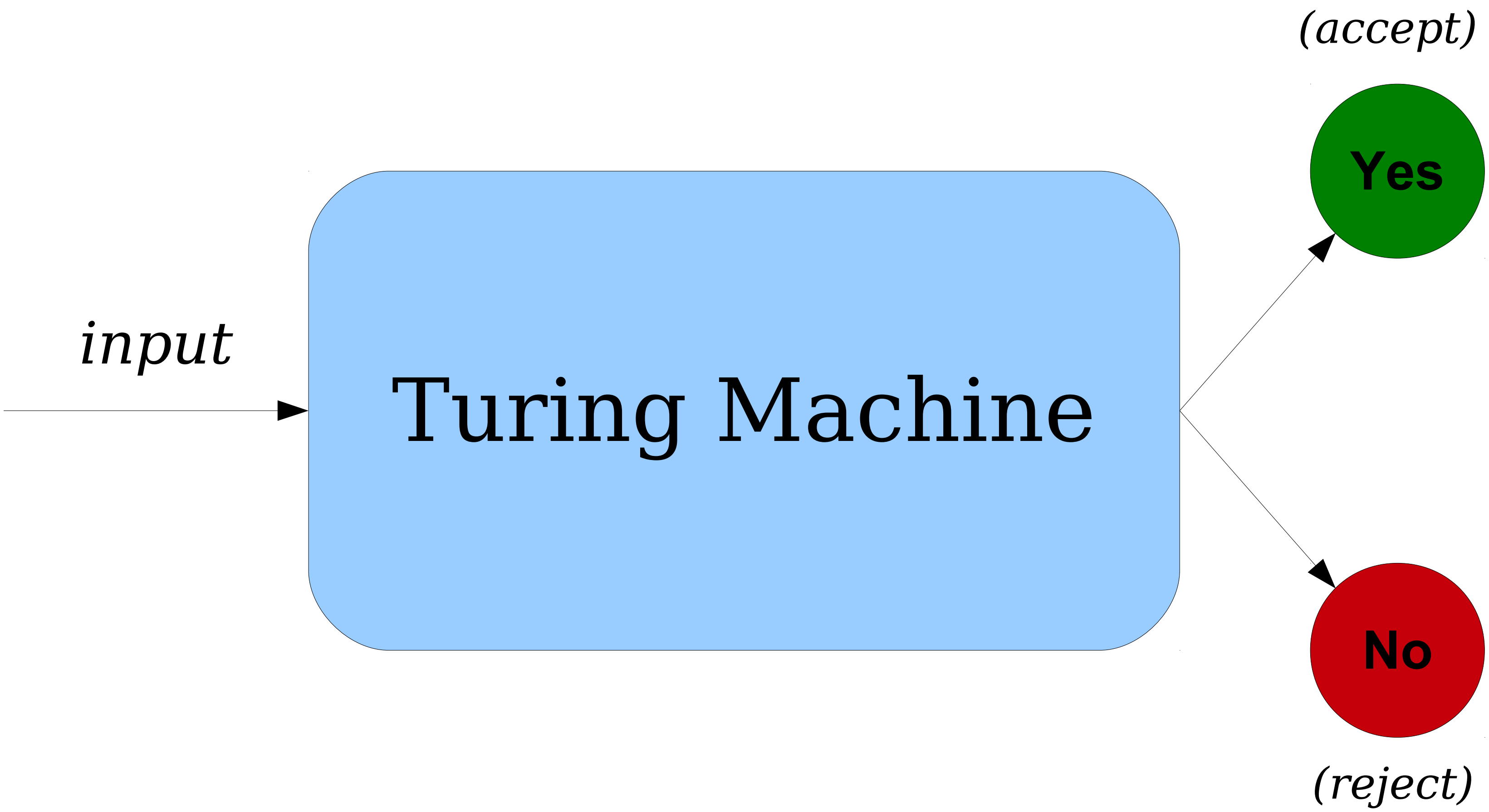
Strings and Objects

- If *Obj* is some mathematical object that is *discrete* and *finite*, then we'll use the notation ***⟨Obj⟩*** to refer to some what of encoding that object as a string.
- Think of ⟨*Obj*⟩ like a file on disk – it encodes complex data as a series of characters.
- ***Key idea:*** If you want to have a TM compute something about *Obj*, you can provide the string ⟨*Obj*⟩ as input to that Turing machine.
- A few remarks about encodings:
    - We don't care *how* we encode the object, just that we can.
    - The particular choice of alphabet isn't important. Given any alphabet, we can always find a way of encoding things.
    - We'll assume we can perform "reasonable" operations on encoded objects.

Strings and Objects

- Given a group of objects Obj_1, Obj_2, …, Obj_n, we can create a single string encoding all these objects.
  - Think of it like a .zip file, but without the compression.
- We'll denote the encoding of all of these objects as a single string by **$\langle Obj_1, \ldots, Obj_n \rangle$**.
- This lets us feed multiple inputs into our computational device at the same time.

Solving Decision Problems



(accept)

Yes

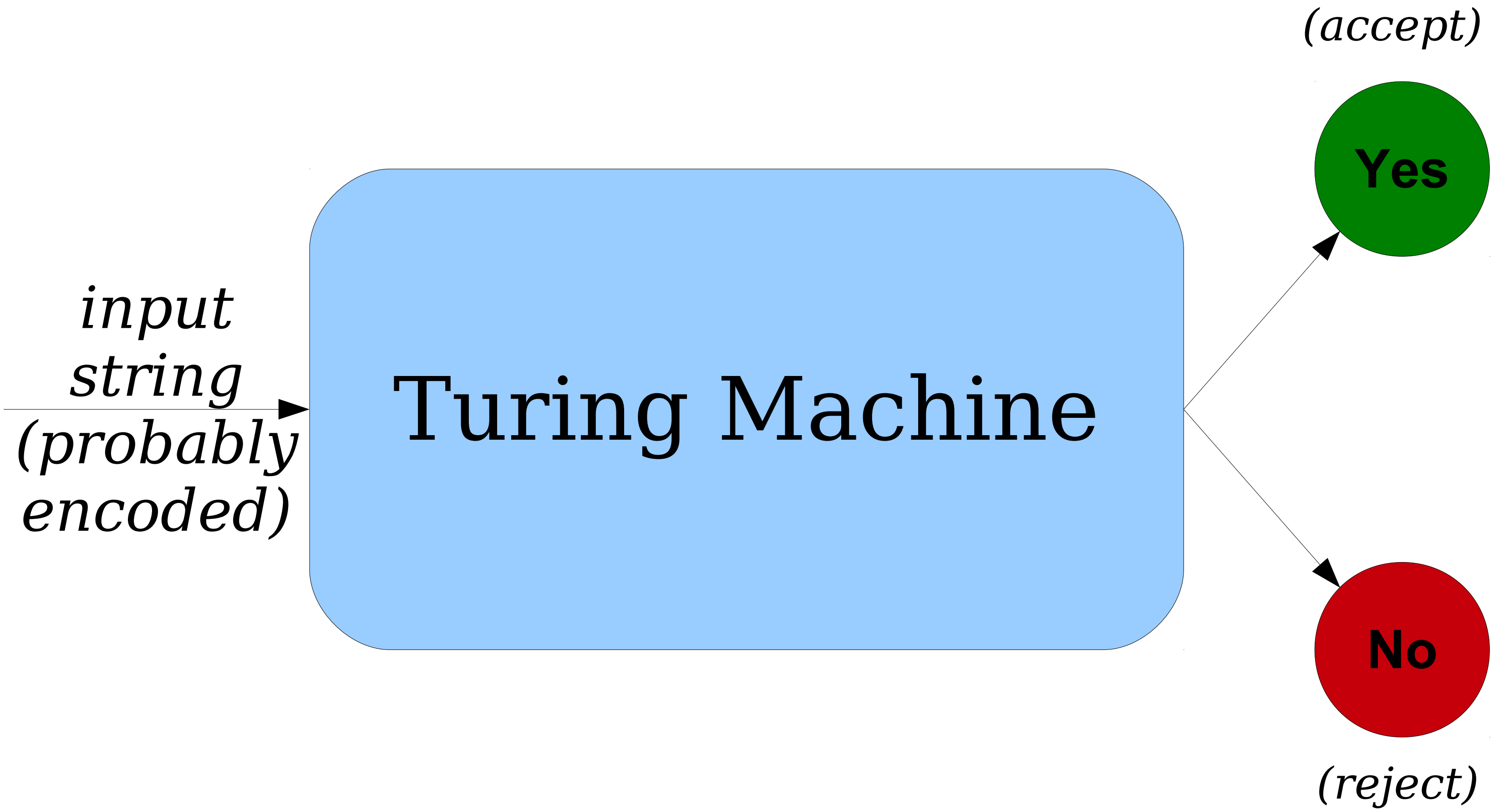
input

Turing Machine

No

(reject)

Solving Decision Problems



(accept)

Yes

input string (probably encoded)

Turing Machine

No

(reject)

What All This Means

- Our goal is to speak of *computers solving problems*.
- We will model this by looking at *TMs recognizing languages*.
- For *decision problems* that we're interested in solving, this precisely captures what we're interested in capturing.

Other Models

- Rather than talking about decision problems, we could talk about ***function problems***, where we take in an input and produce some output object rather than just a yes/no answer.
- Rather than running a single input through the TM and looking at the result, we could imagine that the TM is constantly running, processing inputs as they arrive.
- These are interesting questions to explore! Take CS154 or CS254 for more details!

What problems can we solve with a computer?

What kind of
computer?

What problems can we solve with a computer?

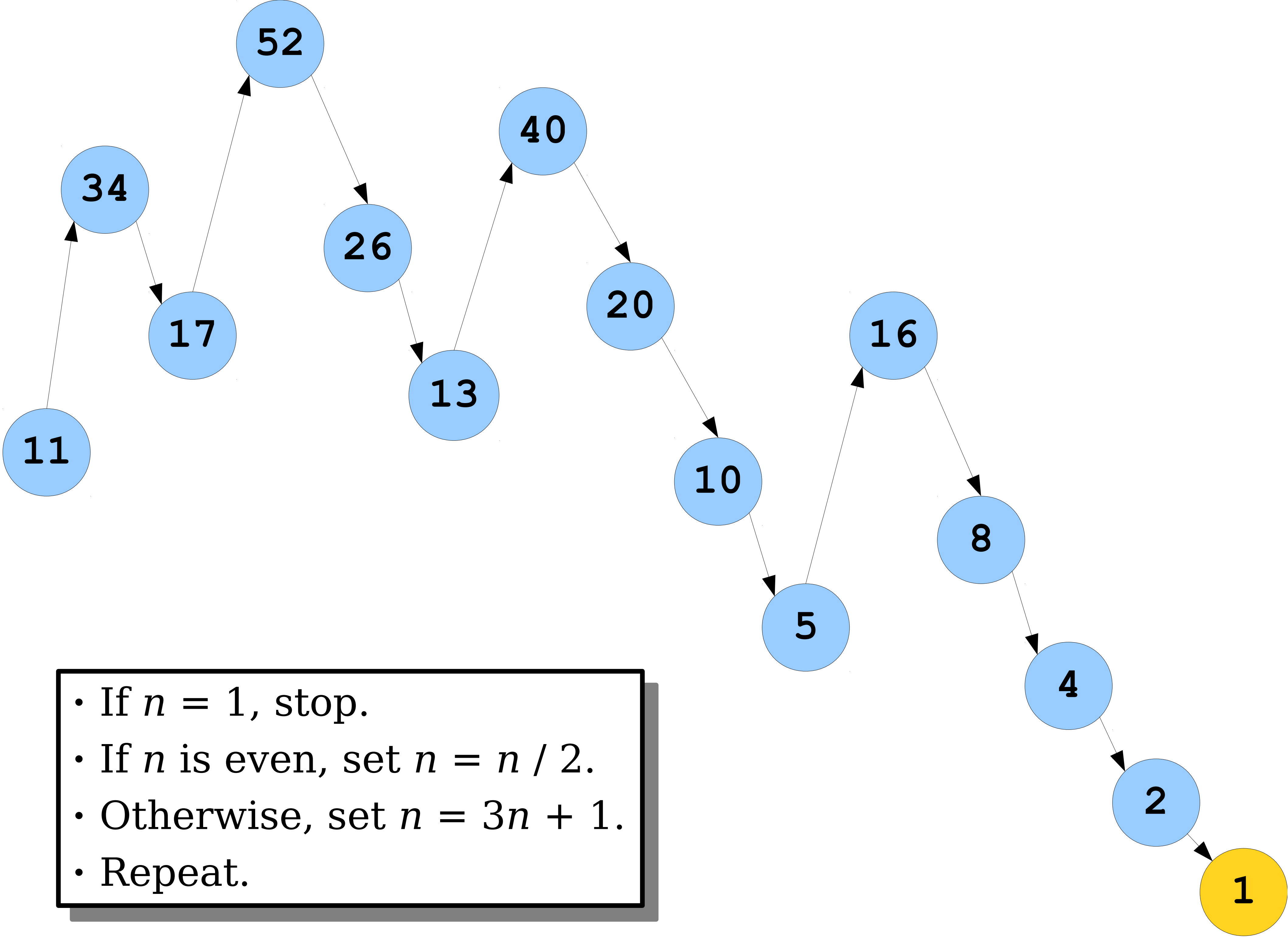
What is a "problem?"

What problems can we solve with a computer?

What does it
mean to “solve“
a problem?

The Hailstone Sequence

- Consider the following procedure, starting with some $n \in \mathbb{N}$, where $n > 0$:
  - If $n = 1$, you are done.
  - If n is even, set $n = n / 2$.
  - Otherwise, set $n = 3n + 1$.
  - Repeat.
- ***Question:*** Given a number n, does this process terminate?



- If $n = 1$, stop.
- If n is even, set $n = n / 2$.
- Otherwise, set $n = 3n + 1$.
- Repeat.

The Hailstone Sequence

- Let $\Sigma = \{\mathtt{1}\}$ and consider the language

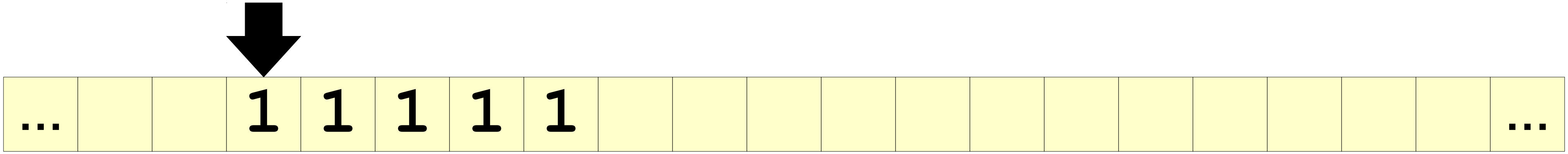
  $L = \{ \mathtt{1}^n \mid n > 0$ and the hailstone sequence terminates for n $\}$.

- Could we build a TM for L?

The Hailstone Turing Machine

- We can build a TM that works as follows:
  - If the input is ε, reject.
  - While the string is not `1`:
    - If the input has even length, halve the length of the string.
    - If the input has odd length, triple the length of the string and append a `1`.
  - Accept.

The Hailstone Turing Machine



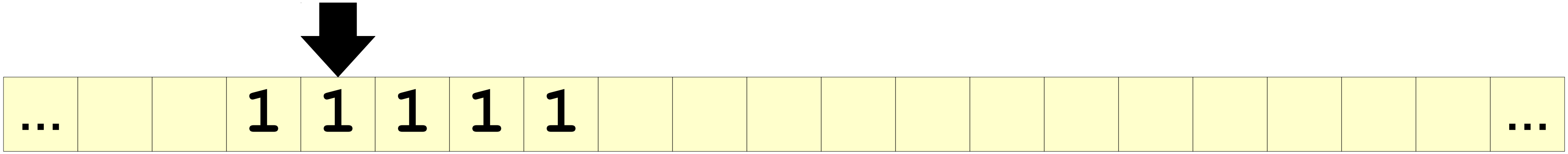
If the input is ε, reject.

While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

Accept.

The Hailstone Turing Machine



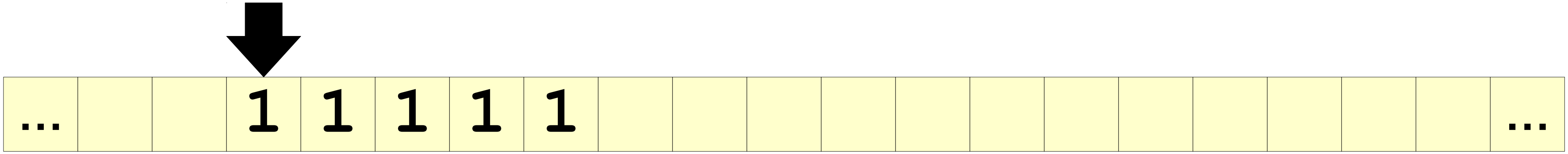
If the input is ε, reject.

While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

Accept.

The Hailstone Turing Machine



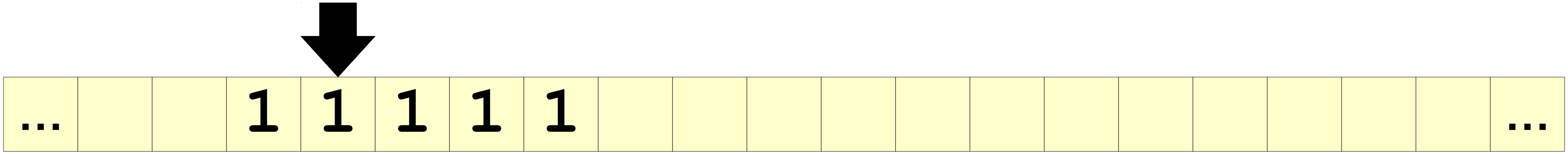
If the input is ε, reject.

While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

Accept.

The Hailstone Turing Machine



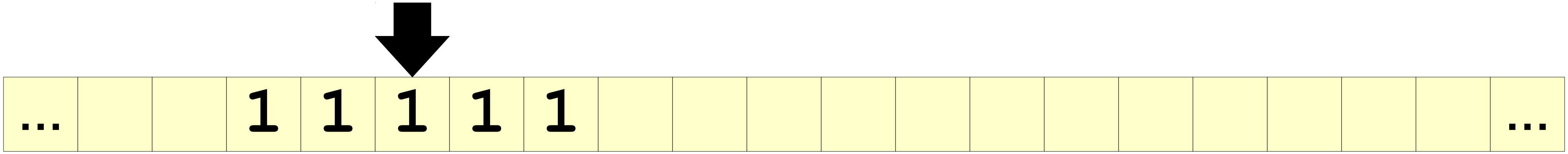
If the input is ε, reject.

While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

Accept.

The Hailstone Turing Machine



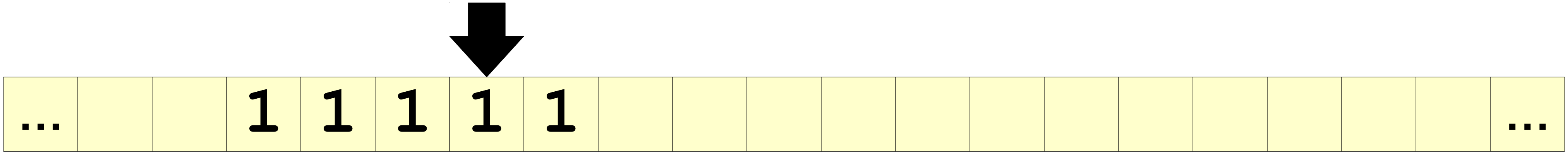
If the input is ε, reject.

While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

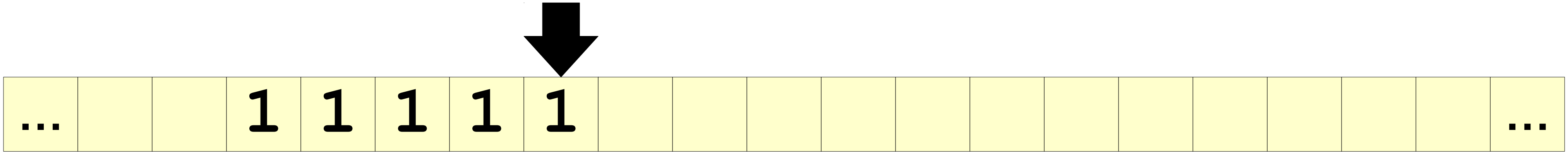
Accept.

The Hailstone Turing Machine



```
If the input is ε, reject.
While the input is not 1:
    • If the input has even length, halve the length
      of the string.
    • If the input has odd length, triple the length of the
      string and append a 1.
Accept.
```

The Hailstone Turing Machine



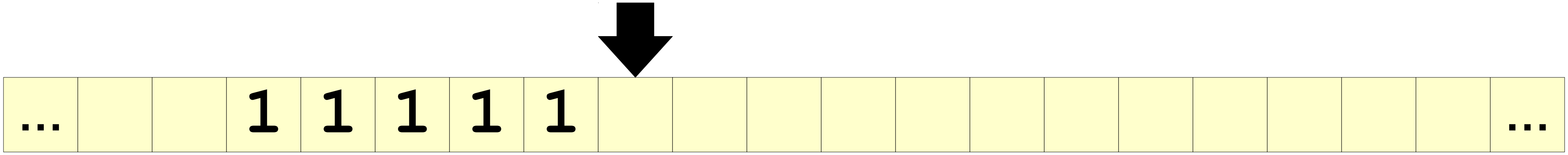
If the input is ε, reject.

While the input is not **1**:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a **1**.

Accept.

The Hailstone Turing Machine



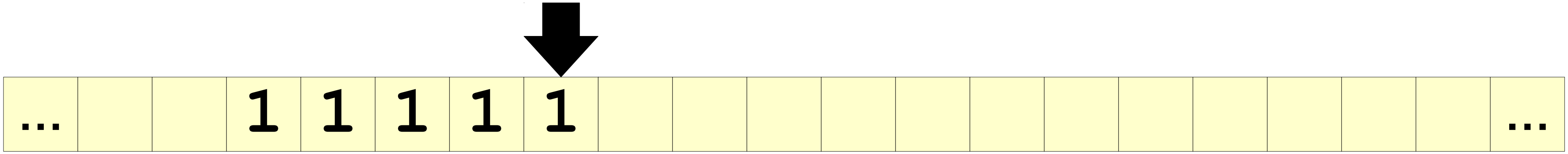
If the input is ε, reject.

While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

Accept.

The Hailstone Turing Machine



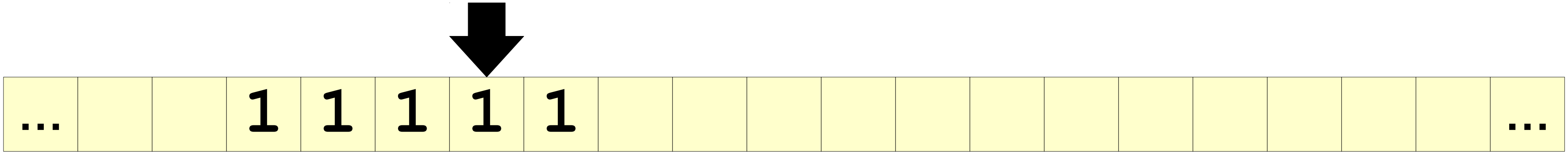
If the input is ε, reject.

While the input is not **1**:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a **1**.

Accept.

The Hailstone Turing Machine



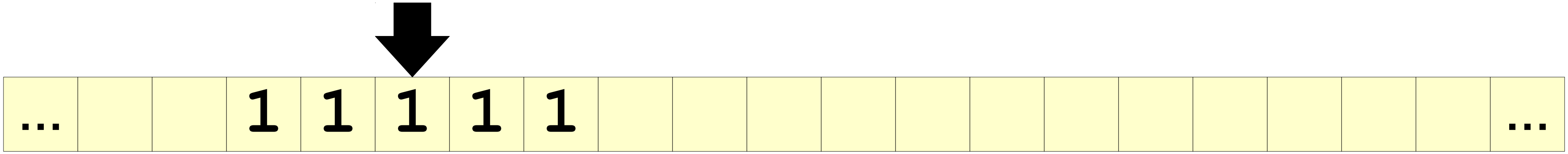
If the input is ε, reject.

While the input is not **1**:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a **1**.

Accept.

The Hailstone Turing Machine



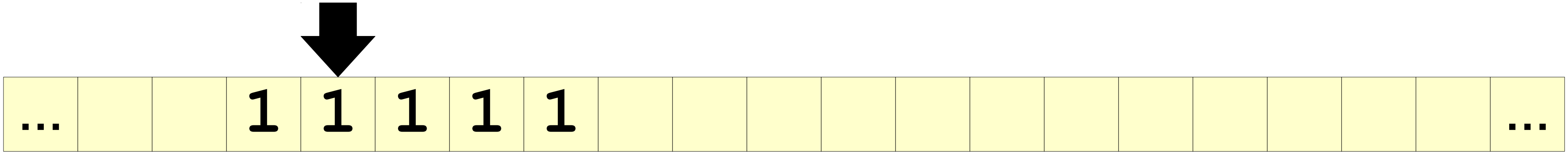
If the input is ε, reject.

While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

Accept.

The Hailstone Turing Machine



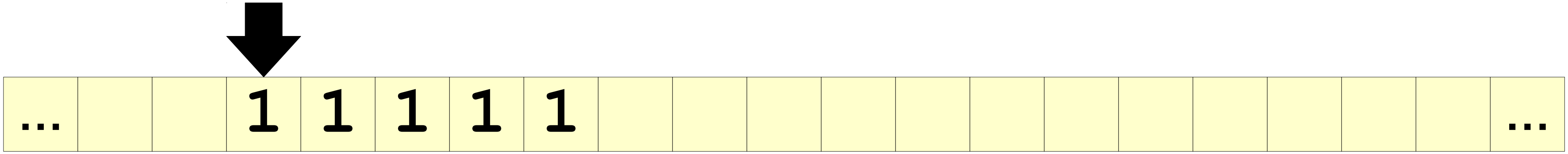
If the input is ε, reject.

While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

Accept.

The Hailstone Turing Machine



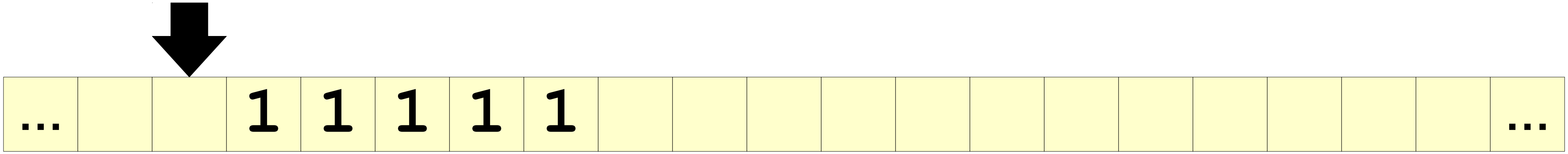
If the input is ε, reject.

While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

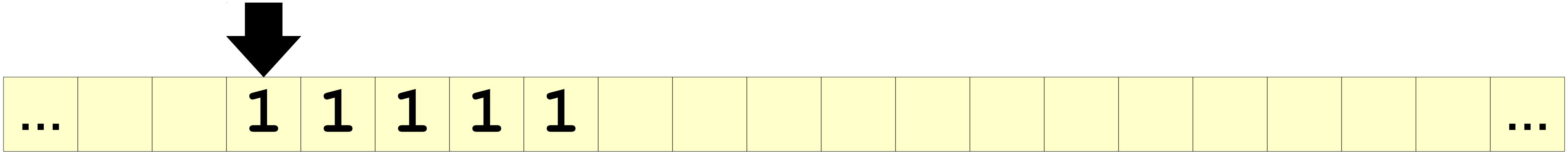
Accept.

The Hailstone Turing Machine



```
If the input is ε, reject.
While the input is not 1:
    • If the input has even length, halve the length of the string.
    • If the input has odd length, triple the length of the string and append a 1.
Accept.
```

The Hailstone Turing Machine



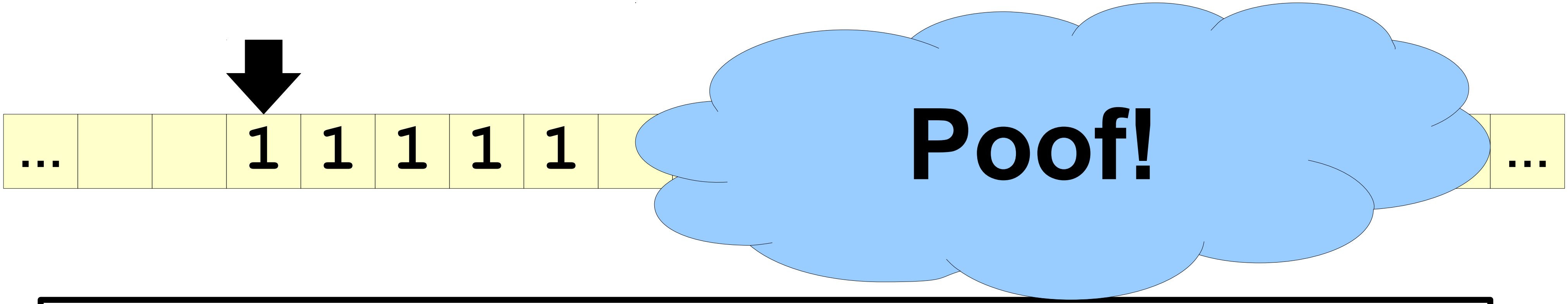
If the input is ε, reject.

While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

Accept.

The Hailstone Turing Machine



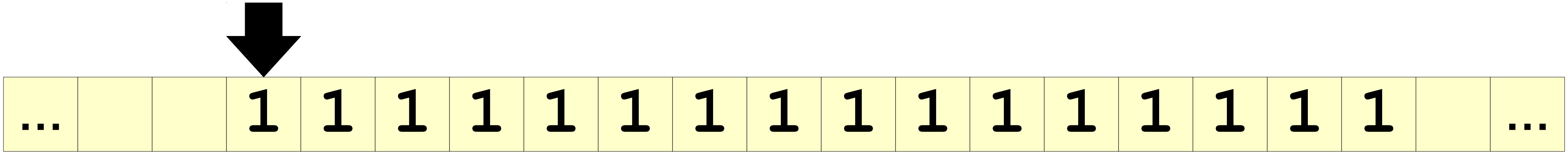
If the input is ε, reject.

While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

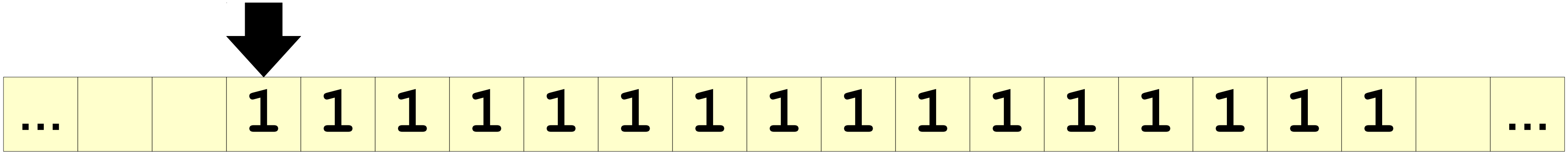
Accept.

The Hailstone Turing Machine



```
If the input is ε, reject.
While the input is not 1:
    • If the input has even length, halve the length of the string.
    • If the input has odd length, triple the length of the string and append a 1.
Accept.
```

The Hailstone Turing Machine

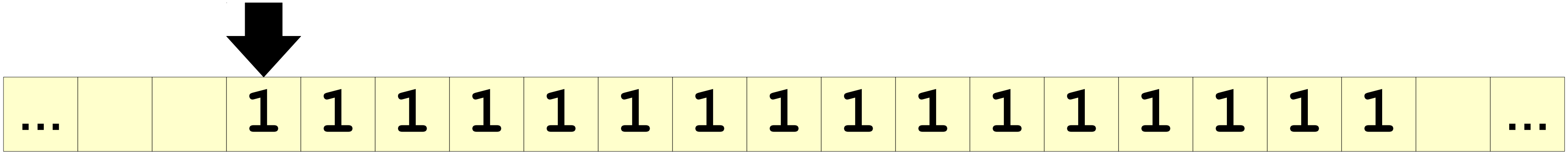


```
If the input is ε, reject.
While the input is not 1:
    · If the input has even len
      of the string.
    · If the input has odd leng
      string and append a 1.
Accept.
```

Problem Set Question:

Build a TM that, starting with n **1**s on its tape, ends with $3n + 1$ **1**s on its tape.

The Hailstone Turing Machine



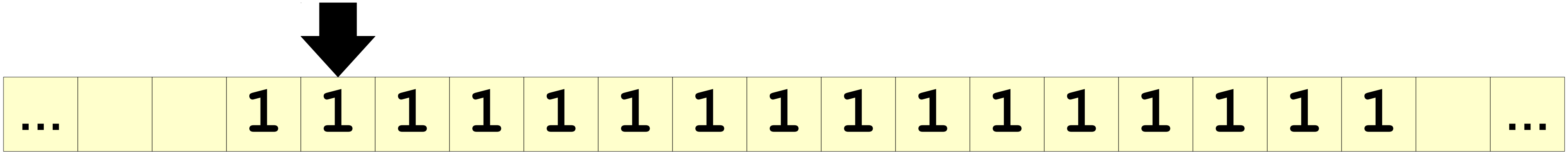
If the input is ε, reject.

While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

Accept.

The Hailstone Turing Machine



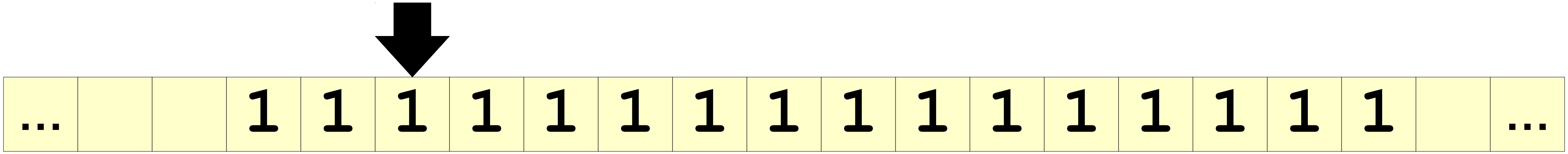
If the input is ε, reject.

While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

Accept.

The Hailstone Turing Machine



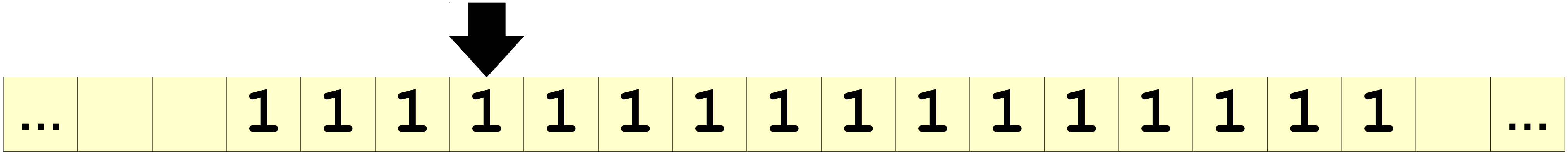
If the input is ε, reject.

While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

Accept.

The Hailstone Turing Machine



…			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	…

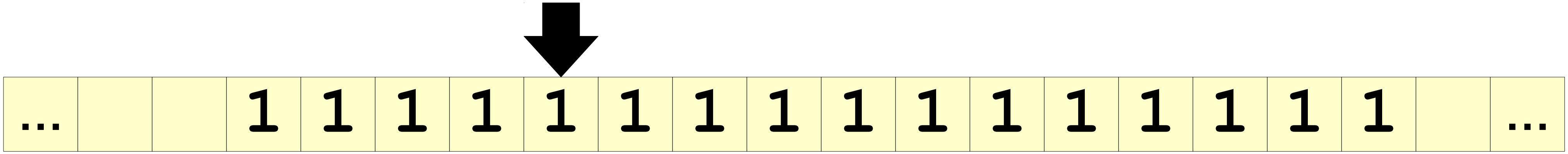
If the input is ε, reject.

While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

Accept.

The Hailstone Turing Machine



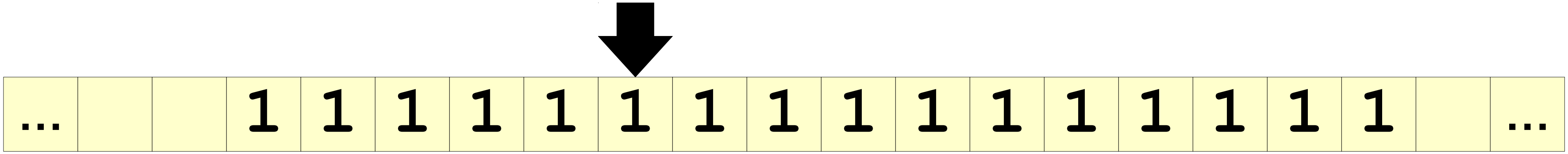
If the input is ε, reject.

While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

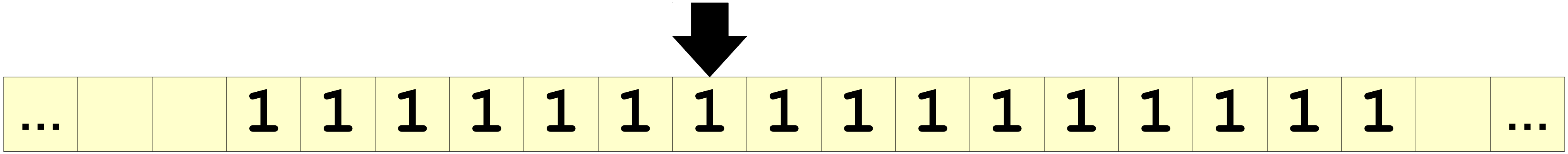
Accept.

The Hailstone Turing Machine



```
If the input is ε, reject.
While the input is not 1:
    · If the input has even length, halve the length
      of the string.
    · If the input has odd length, triple the length of the
      string and append a 1.
Accept.
```

The Hailstone Turing Machine



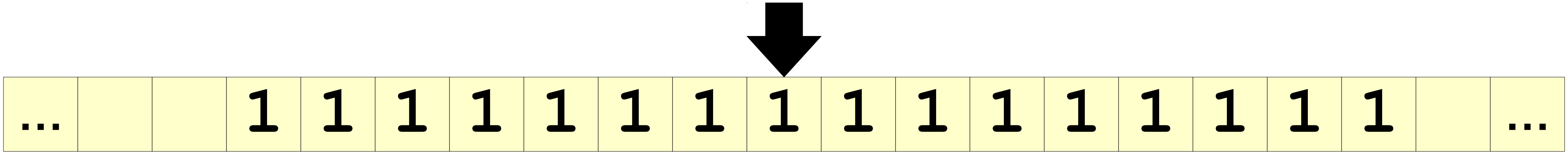
If the input is ε, reject.

While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

Accept.

The Hailstone Turing Machine



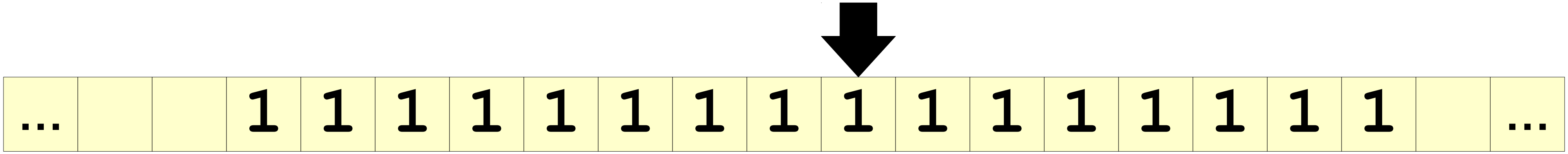
If the input is ε, reject.

While the input is not **1**:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a **1**.

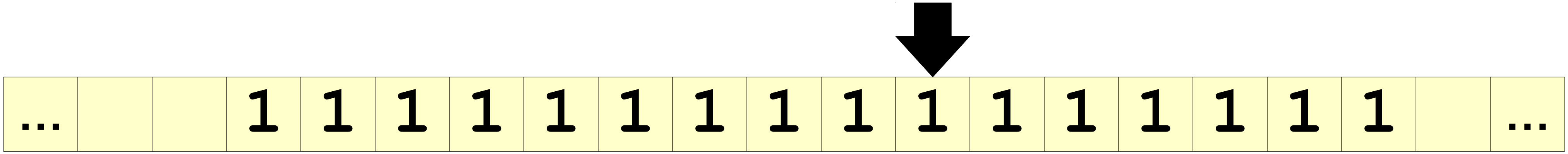
Accept.

The Hailstone Turing Machine



```
If the input is ε, reject.
While the input is not 1:
    • If the input has even length, halve the length
      of the string.
    • If the input has odd length, triple the length of the
      string and append a 1.
Accept.
```

The Hailstone Turing Machine



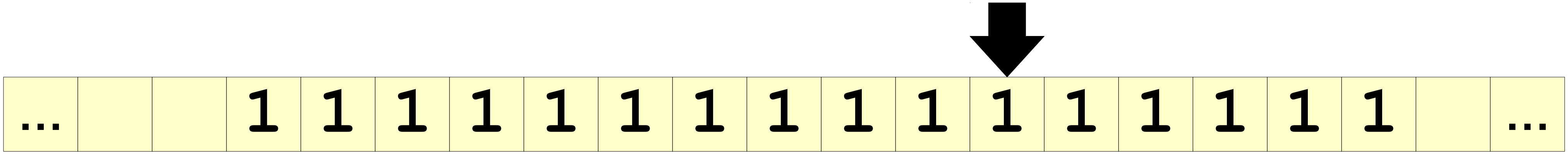
If the input is ε, reject.

While the input is not **1**:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a **1**.

Accept.

The Hailstone Turing Machine



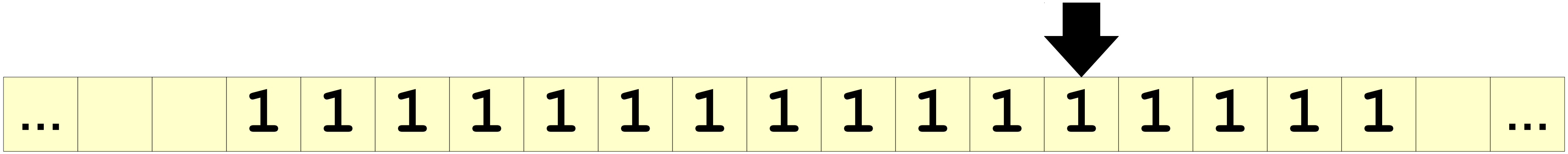
If the input is ε, reject.

While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

Accept.

The Hailstone Turing Machine



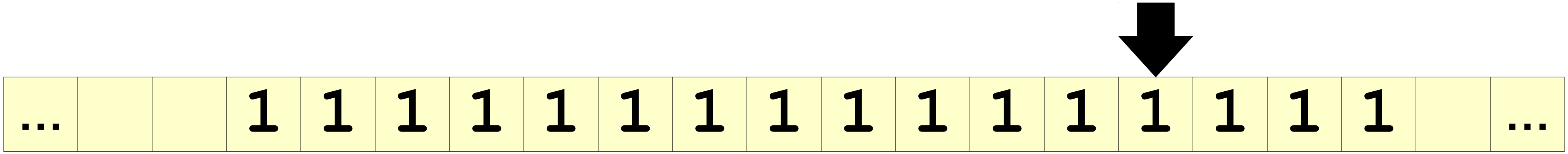
If the input is ε, reject.

While the input is not `1`:

  - If the input has even length, halve the length of the string.
  - If the input has odd length, triple the length of the string and append a `1`.

Accept.

The Hailstone Turing Machine



…			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		…

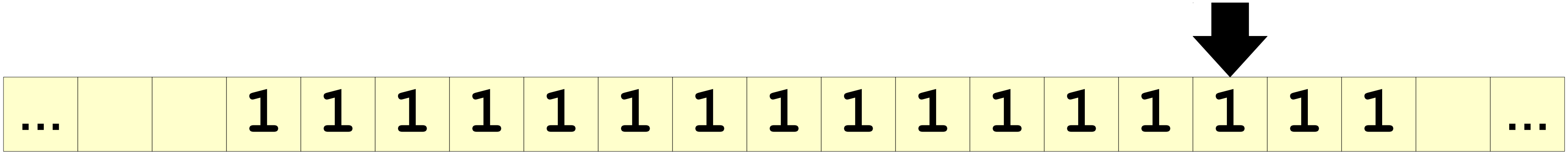
If the input is ε, reject.

While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

Accept.

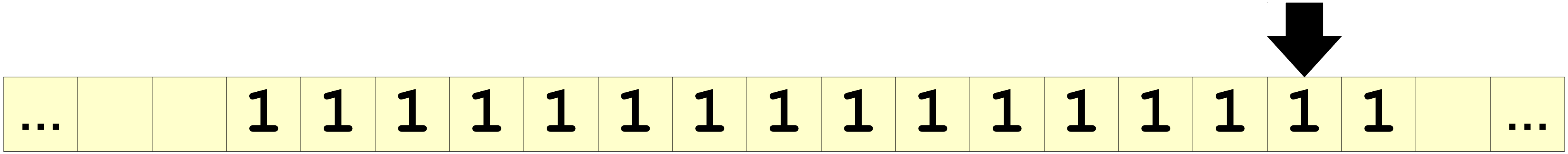
The Hailstone Turing Machine



…			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		…

```
If the input is ε, reject.
While the input is not 1:
    • If the input has even length, halve the length of the string.
    • If the input has odd length, triple the length of the string and append a 1.
Accept.
```

The Hailstone Turing Machine



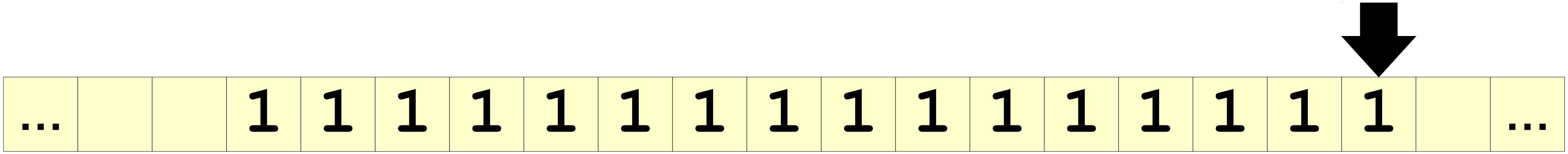
If the input is ε, reject.

While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

Accept.

The Hailstone Turing Machine



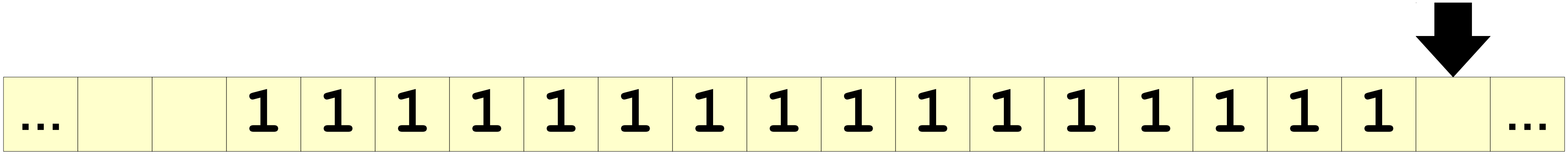
If the input is ε, reject.

While the input is not 1:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a 1.

Accept.

The Hailstone Turing Machine



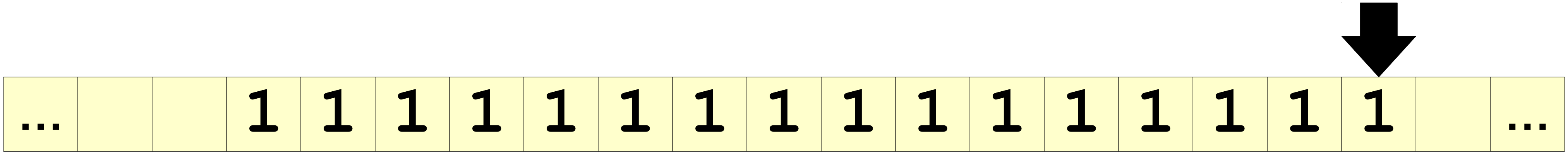
If the input is ε, reject.

While the input is not **1**:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a **1**.

Accept.

The Hailstone Turing Machine



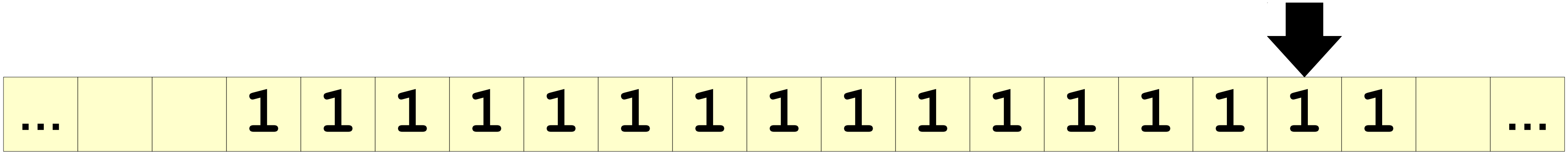
If the input is ε, reject.

While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

Accept.

The Hailstone Turing Machine



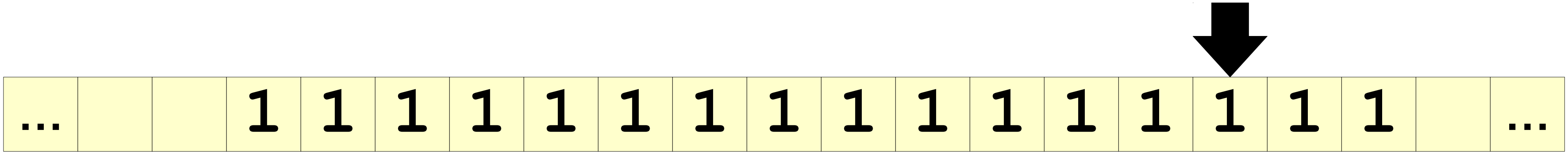
If the input is ε, reject.

While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

Accept.

The Hailstone Turing Machine



…			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		…

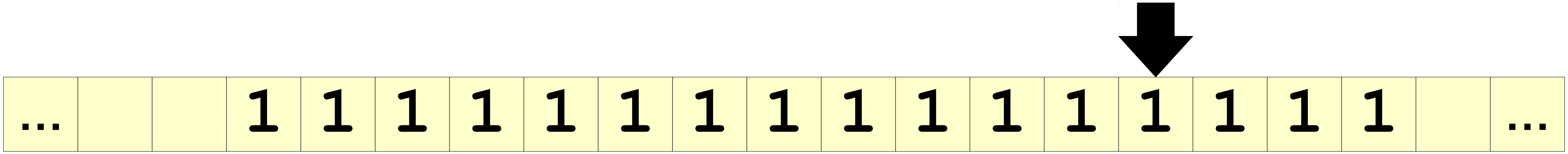
If the input is ε, reject.

While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

Accept.

The Hailstone Turing Machine



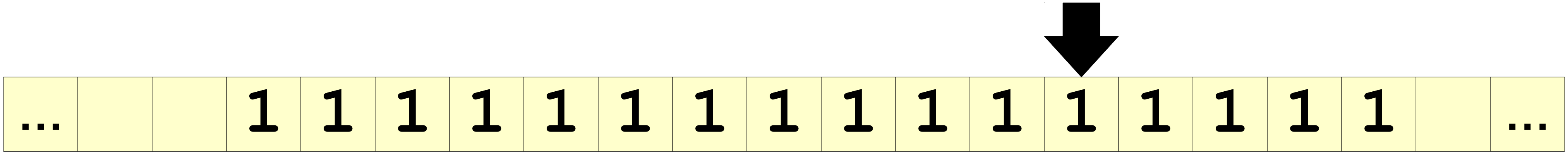
If the input is ε, reject.

While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

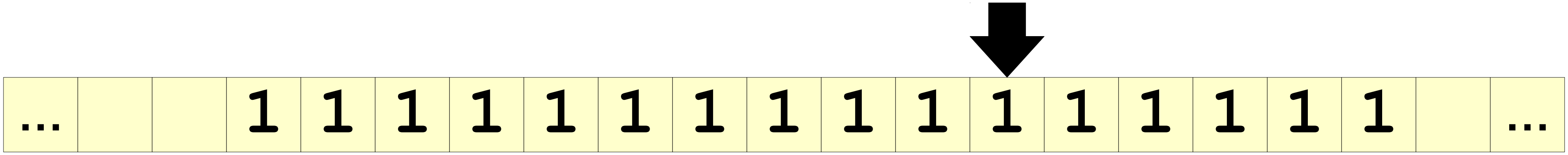
Accept.

The Hailstone Turing Machine



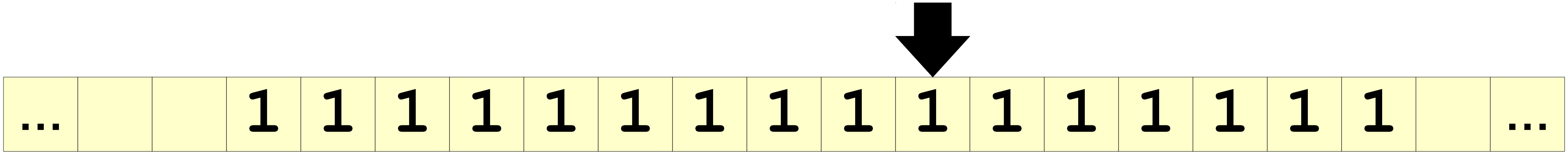
```
If the input is ε, reject.
While the input is not 1:
    • If the input has even length, halve the length
      of the string.
    • If the input has odd length, triple the length of the
      string and append a 1.
Accept.
```

The Hailstone Turing Machine



```
If the input is ε, reject.
While the input is not 1:
    • If the input has even length, halve the length
      of the string.
    • If the input has odd length, triple the length of the
      string and append a 1.
Accept.
```

The Hailstone Turing Machine



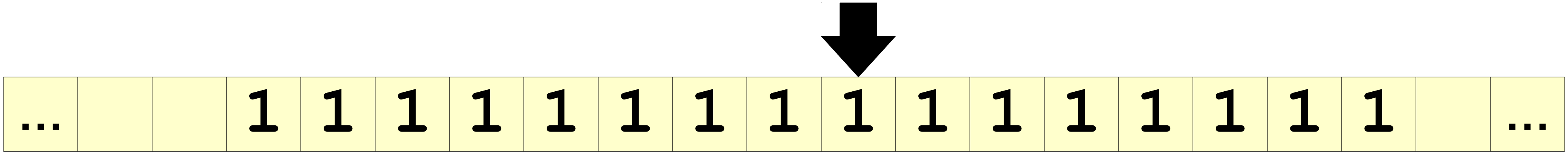
If the input is ε, reject.

While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

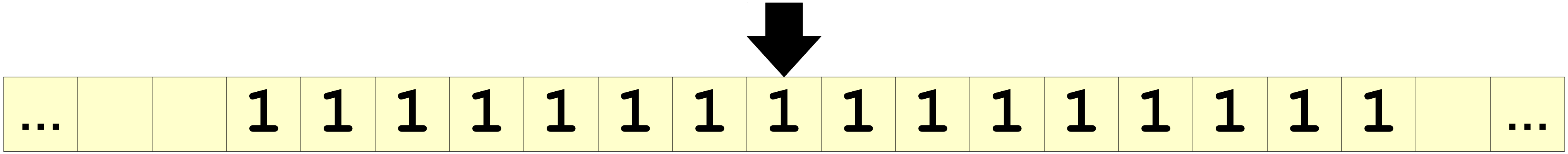
Accept.

The Hailstone Turing Machine



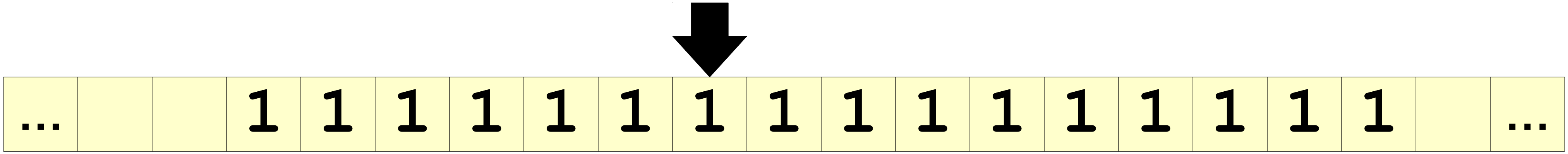
```
If the input is ε, reject.
While the input is 1:
    • If the input has even length, halve the length
      of the string.
    • If the input has odd length, triple the length of the
      string and append a 1.
Accept.
```

The Hailstone Turing Machine



```
If the input is ε, reject.
While the input is not 1:
    • If the input has even length, halve the length of the string.
    • If the input has odd length, triple the length of the string and append a 1.
Accept.
```

The Hailstone Turing Machine



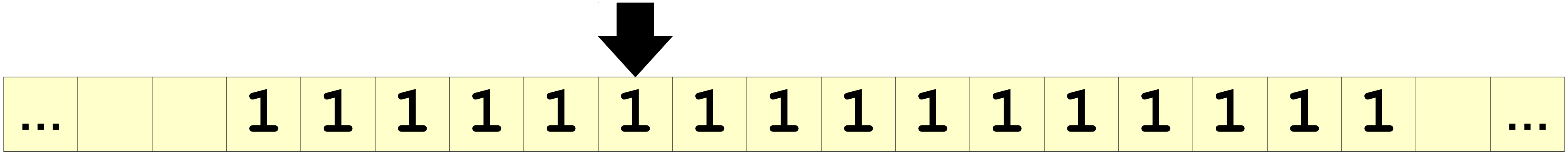
If the input is ε, reject.

While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

Accept.

The Hailstone Turing Machine



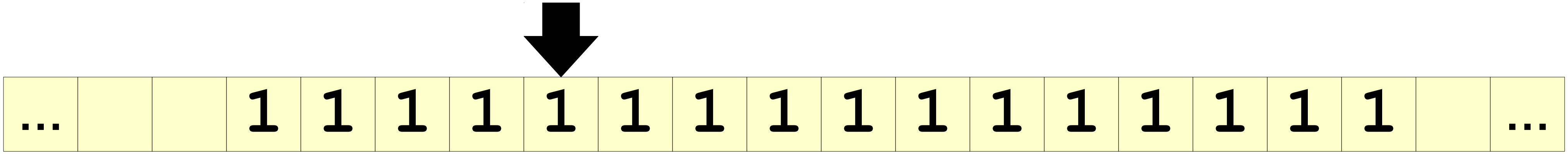
If the input is ε, reject.

While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

Accept.

The Hailstone Turing Machine



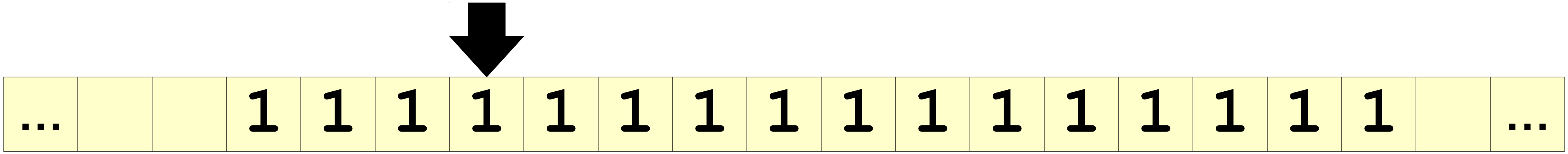
If the input is ε, reject.

While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

Accept.

The Hailstone Turing Machine



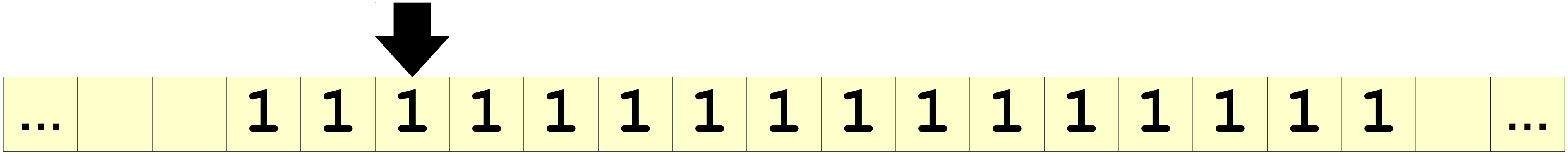
If the input is ε, reject.

While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

Accept.

The Hailstone Turing Machine



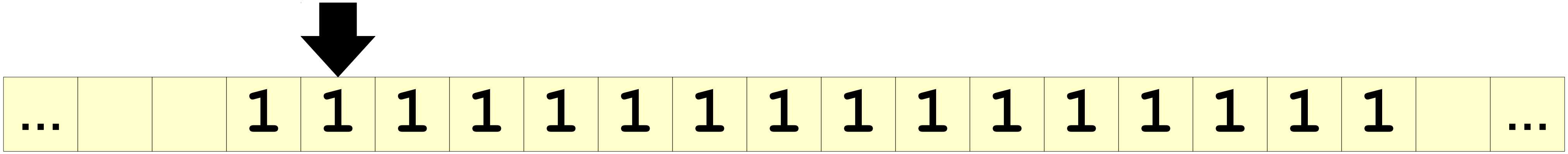
If the input is ε, reject.

While the input is not **`1`**:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a **`1`**.

Accept.

The Hailstone Turing Machine



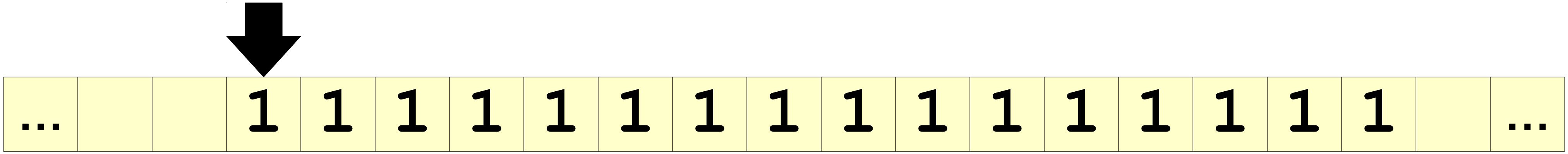
If the input is ε, reject.

While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

Accept.

The Hailstone Turing Machine



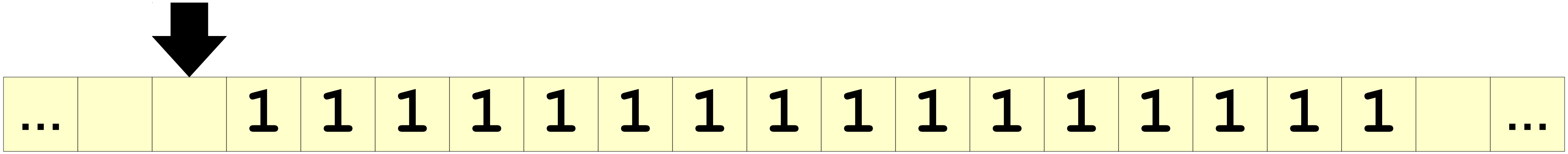
If the input is ε, reject.

While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

Accept.

The Hailstone Turing Machine



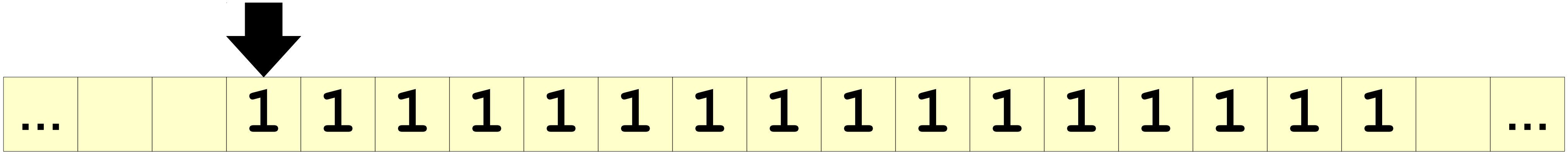
If the input is ε, reject.

While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

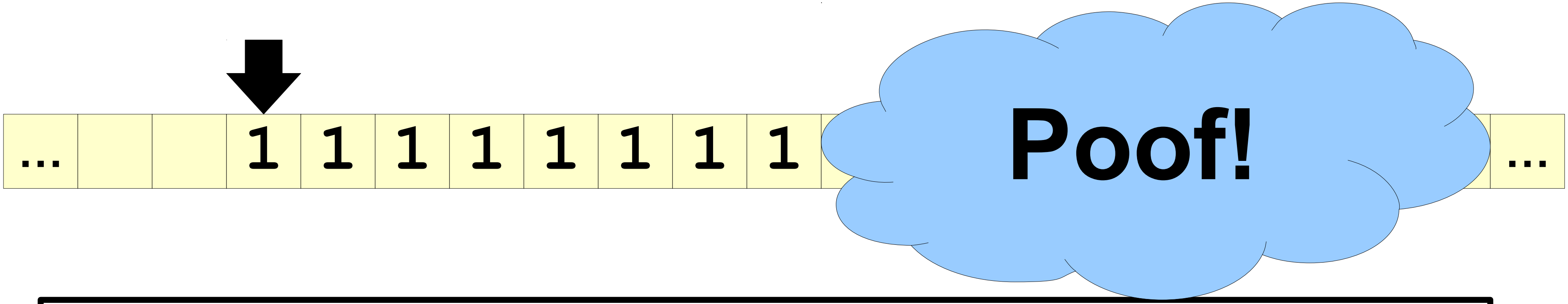
Accept.

The Hailstone Turing Machine



```
If the input is ε, reject.
While the input is not 1:
    • If the input has even length, halve the length
      of the string.
    • If the input has odd length, triple the length of the
      string and append a 1.
Accept.
```

The Hailstone Turing Machine



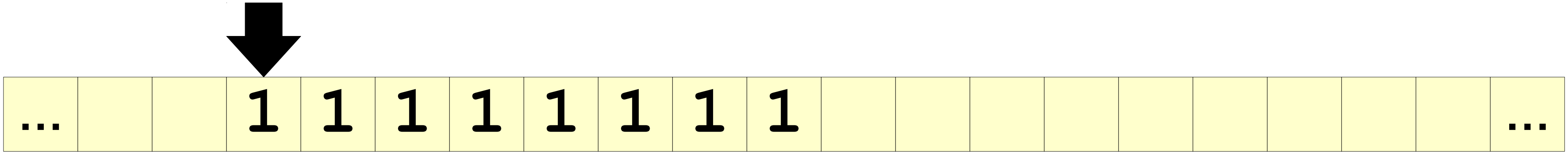
If the input is ε, reject.

While the input is not 1:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a 1.

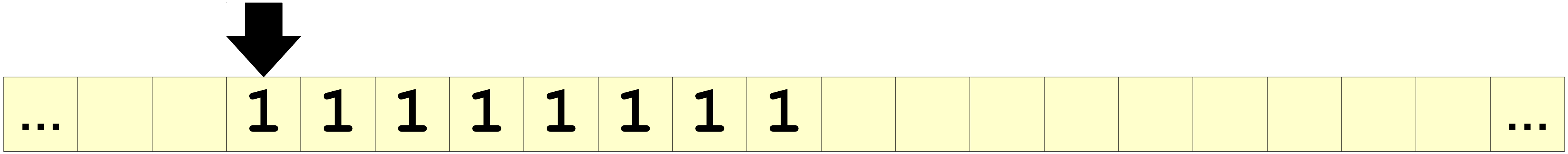
Accept.

The Hailstone Turing Machine



```
If the input is ε, reject.
While the input is not 1:
    • If the input has even length, halve the length
      of the string.
    • If the input has odd length, triple the length of the
      string and append a 1.
Accept.
```

The Hailstone Turing Machine



If the input is ε, reject.

While the input is not **`1`**:

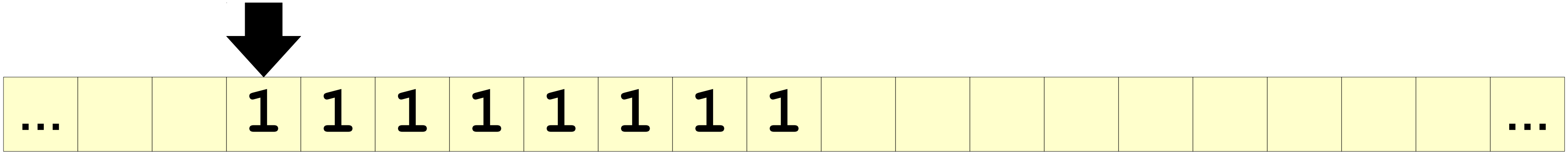
- If the input has even leng
  of the string.
- If the input has odd lengt
  string and append a **`1`**.

Accept.

Problem Set Question:

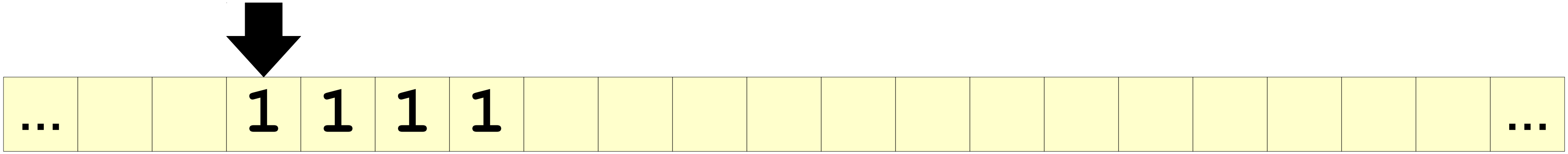
Build a TM that, starting with $2n$ **`1`**s on its tape, ends with n **`1`**s on its tape.

The Hailstone Turing Machine



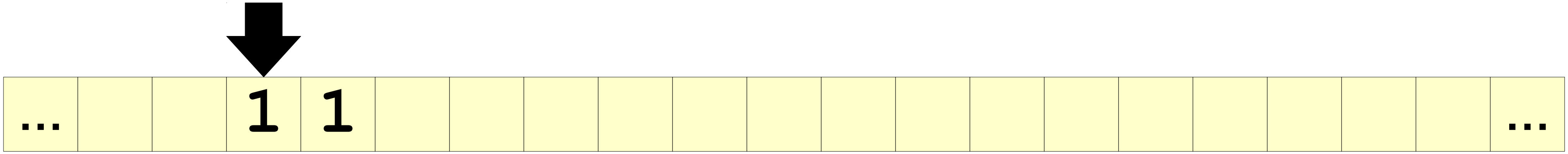
```
If the input is ε, reject.
While the input is not 1:
    • If the input has even length, halve the length of the string.
    • If the input has odd length, triple the length of the string and append a 1.
Accept.
```

The Hailstone Turing Machine



```
If the input is ε, reject.
While the input is not 1:
    • If the input has even length, halve the length of the string.
    • If the input has odd length, triple the length of the string and append a 1.
Accept.
```

The Hailstone Turing Machine



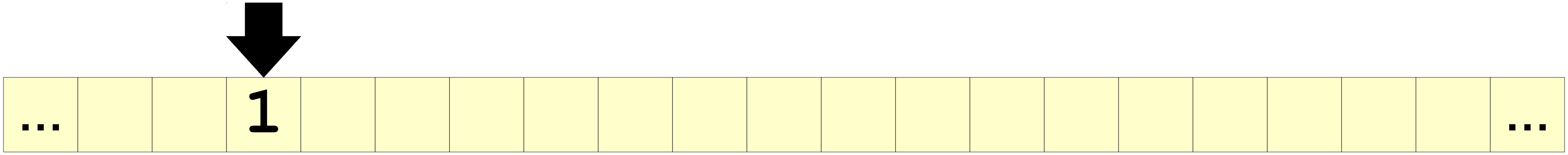
If the input is ε, reject.

While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

Accept.

The Hailstone Turing Machine



If the input is ε, reject.

While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

Accept.

Does this Turing machine accept all nonempty strings?

The Collatz Conjecture

- It is *unknown* whether this process will terminate for all natural numbers.
- In other words, ***no one knows whether the TM described in the previous slides will always stop running!***
- The conjecture (unproven claim) that this always terminates is called the ***Collatz Conjecture***.

The Collatz Conjecture

“Mathematics may not be ready for such problems.” - Paul Erdős

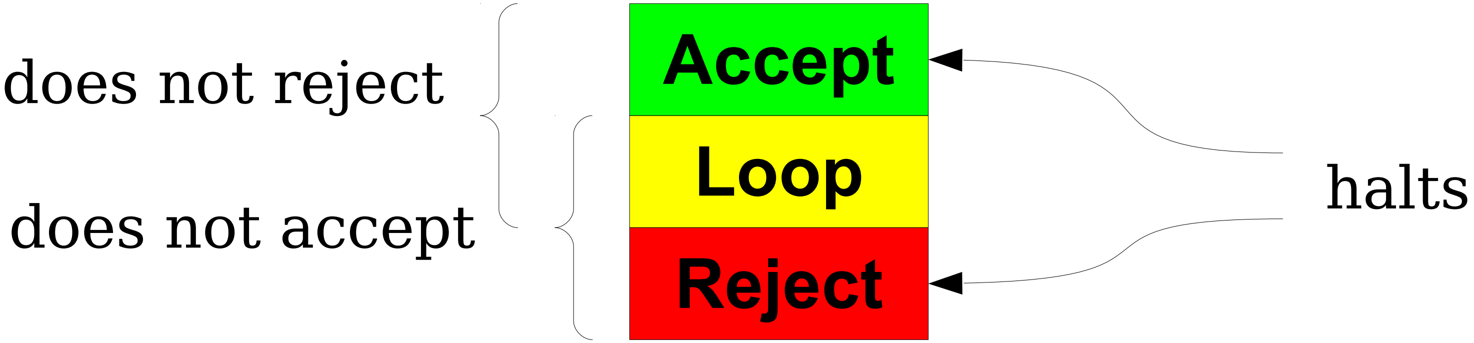
- The fact that the Collatz Conjecture is unresolved is useful later on for building intuitions. Keep this in mind!

An Important Observation

- Unlike finite automata, which automatically halt after all the input is read, TMs keep running until they explicitly enter an accept or reject state.
- It is possible for a TM to run forever without accepting or rejecting.
- This leads to several important questions:
    - How do we formally define what it means to build a TM for a language?
    - What implications does this have about problem-solving?

Very Important Terminology

- Let *M* be a Turing machine.
- *M* ***accepts*** a string *w* if it enters an accept state when run on *w*.
- *M* ***rejects*** a string *w* if it enters a reject state when run on *w*.
- *M* ***loops infinitely*** (or just ***loops***) on a string *w* if when run on *w* it enters neither an accept nor a reject state.
- *M* ***does not accept w*** if it either rejects *w* or loops infinitely on *w*.
- *M* ***does not reject w*** *w* if it either accepts *w* or loops on *w*.
- *M* ***halts on w*** if it accepts *w* or rejects *w*.



The Language of a TM

- The language of a Turing machine M, denoted $\mathscr{L}(M)$, is the set of all strings that M accepts:

$$\mathscr{L}(\boldsymbol{M}) = \{ \boldsymbol{w} \in \Sigma^* \mid \boldsymbol{M} \textbf{ accepts } \boldsymbol{w} \}$$

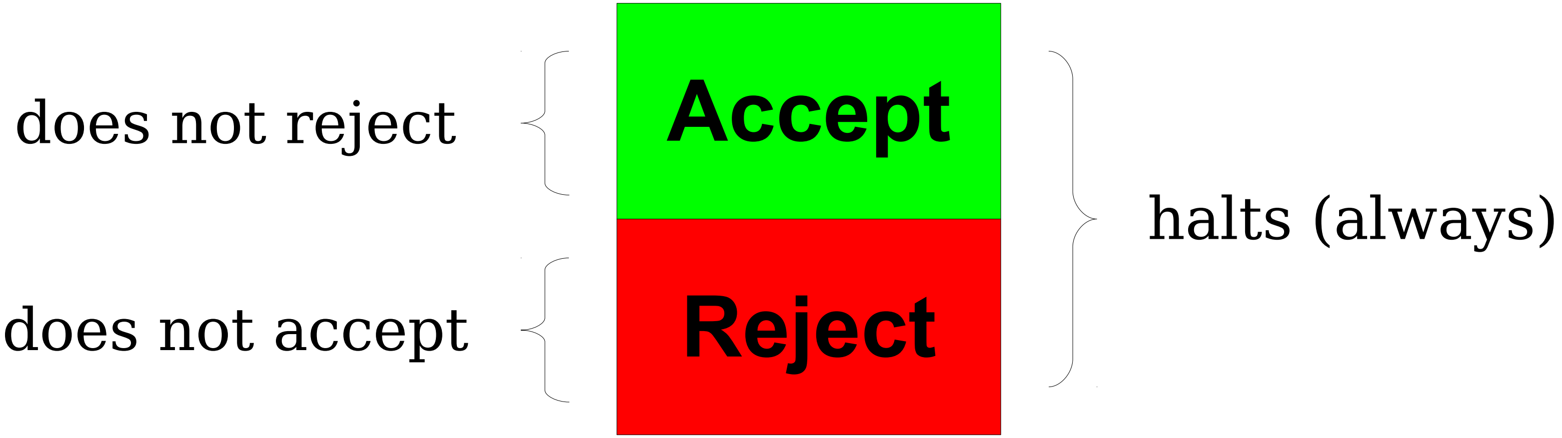
- For any $w \in \mathscr{L}(M)$, M accepts w.
- For any $w \notin \mathscr{L}(M)$, M does not accept w.
  - It might loop forever, or it might explicitly reject.
- A language is called ***recognizable*** if it is the language of some TM.
- A TM M where $\mathscr{L}(M) = L$ is called a ***recognizer*** for L.
- Notation: the class **RE** is the set of all recognizable languages.

$$L \in \mathbf{RE} \quad \leftrightarrow \quad L \text{ is recognizable}$$

What do you think? Does that correspond to what you think it means to solve a problem?

Deciders

- Some Turing machines always halt; they never go into an infinite loop.
- If M is a TM and M halts on every possible input, then we say that M is a ***decider***.
- For deciders, accepting is the same as not rejecting and rejecting is the same as not accepting.



Decidable Languages

- A language L is called ***decidable*** if there is a decider M such that $\mathscr{L}(M) = L$.
- Equivalently, a language L is decidable if there is a TM M such that
  - If $w \in L$, then M accepts w.
  - If $w \notin L$, then M rejects w.
- The class **R** is the set of all decidable languages.

$$L \in \mathbf{R} \quad \leftrightarrow \quad L \text{ is decidable}$$

Examples of **R** Languages

- All regular languages are in **R**.
  - If *L* is regular, we can run the DFA for *L* on a string *w* and then either accept or reject *w* based on what state it ends in.
- $\{ \texttt{0}^n\texttt{1}^n \mid n \in \mathbb{N} \}$ is in **R**.
  - The TM we built is a decider.
- All CFLs are in **R**.
  - Proof is tricky; check Sipser for details.
  - (This is why it's possible to build the CFG tool online!)

Why **R** Matters

- If a language is in **R**, there is an algorithm that can decide membership in that language.
    - Run the decider and see what it says.
- If there is an algorithm that can decide membership in a language, that language is in **R**.
    - By the Church-Turing thesis, any effective model of computation is equivalent in power to a Turing machine.
    - Therefore, if there is *any* algorithm for deciding membership in the language, there is a decider for it.
    - Therefore, the language is in **R**.
- ***A language is in R if and only if there is an algorithm for deciding membership in that language.***

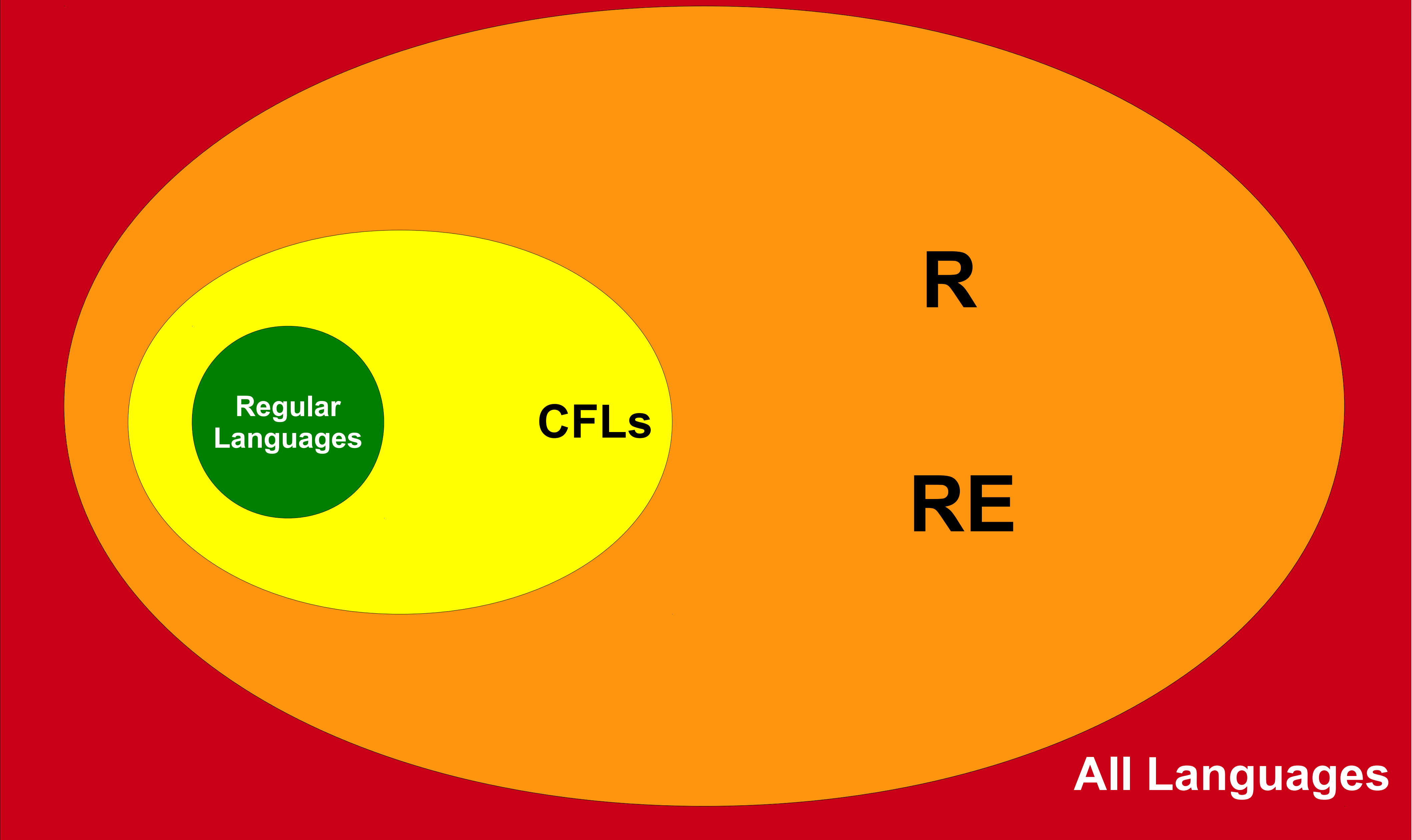
R and **RE** Languages

- Every decider is a Turing machine, but not every Turing machine is a decider.
- This means that **R** ⊆ **RE**.
- Hugely important theoretical question:

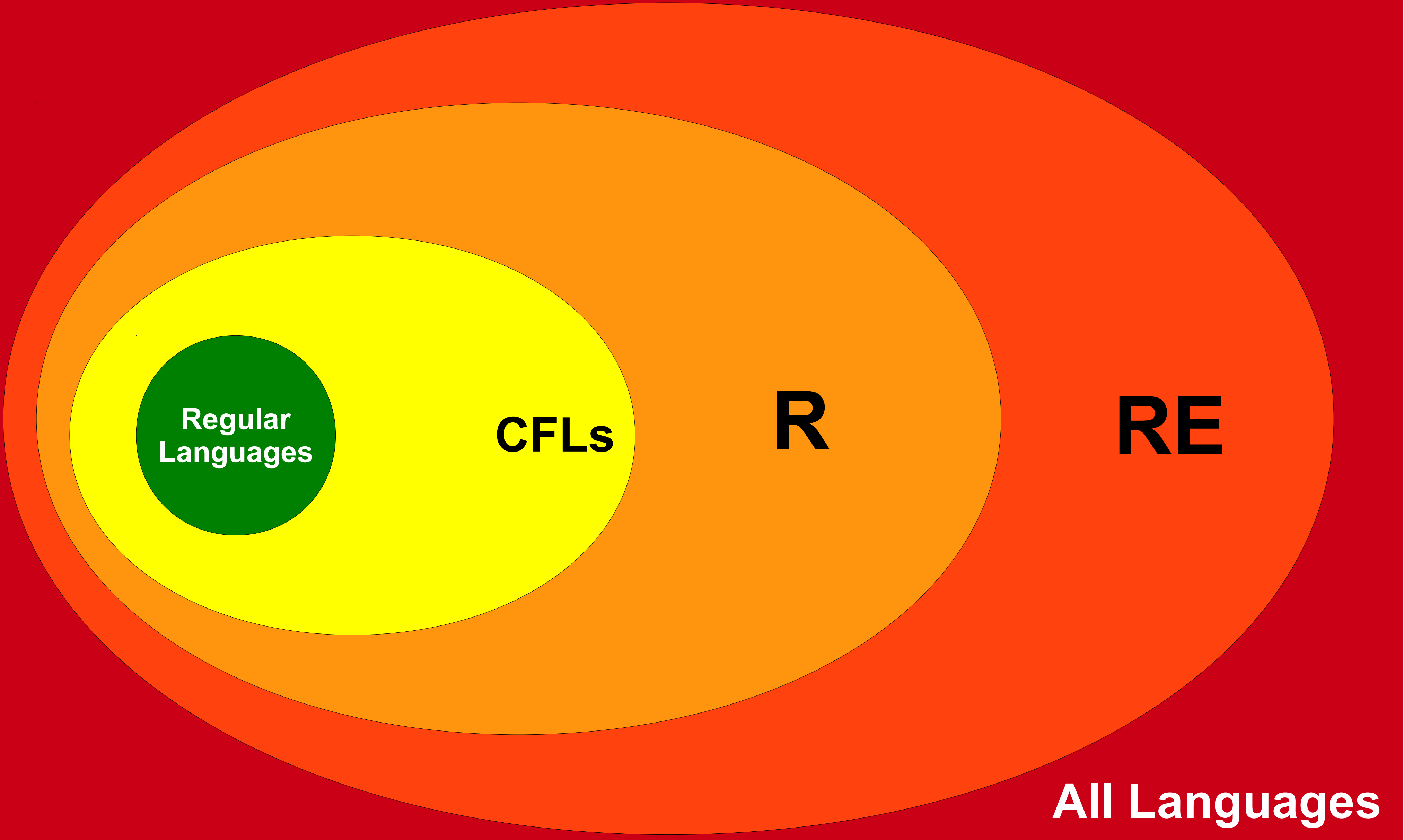
$$\mathbf{R} \overset{?}{=} \mathbf{RE}$$

- That is, if you can just confirm "yes" answers to a problem, can you necessarily *solve* that problem?

Which Picture is Correct?



Which Picture is Correct?



Unanswered Questions

- Why exactly is **RE** an interesting class of problems?
- What does the $\mathbf{R} \overset{?}{=} \mathbf{RE}$ question mean?
- Is **R** = **RE**?
- What lies beyond **R** and **RE**?
- We'll see the answers to each of these in due time.

Time-Out for Announcements!

Midterm Debrief

- You’re done with the second midterm! Woohoo!
- We’ll be grading the exam over the weekend. We’ll get back with solutions and statistics as soon as the exam is graded.
- If you have questions about the midterm in the meantime, please feel free to contact us over email, ask in office hours, or ask on Piazza.

Problem Set Seven

- PS7 is due on Friday.
  - You can use late days to extend the deadline to Monday if you'd like.
  - ***Planning ahead:*** You can't use late days on PS9. You can use late days on PS8, but it will cut into the time you'll likely want to spend working on PS9.
- As always, please feel free to reach out to us if you have any questions!

Back to CS103!

Emergent Properties

Emergent Properties

- An ***emergent property*** of a system is a property that arises out of smaller pieces that doesn't seem to exist in any of the individual pieces.
- Examples:
  - Individual neurons work by firing in response to particular combinations of inputs. Somehow, this leads to thought and consciousness.
  - Individual atoms obey the laws of quantum mechanics and just interact with other atoms. Somehow, it's possible to combine them together to make iPhones and pumpkin pie.

Emergent Properties of Computation

- All computing systems equal to Turing machines exhibit several surprising emergent properties.
- If we believe the Church-Turing thesis, these emergent properties are, in a sense, “inherent” to computation. You can't have computation without these properties.
- These emergent properties are what ultimately make computation so interesting and so powerful.
- As we'll see, though, they're also computation's Achilles heel – they're how we find concrete examples of impossible problems.

Two Emergent Properties

- There are two key emergent properties of computation that we will discuss:
  - ***Universality***: There is a single computing device capable of performing any computation.
  - ***Self-Reference***: Computing devices can ask questions about their own behavior.
- As you'll see, the combination of these properties leads to simple examples of impossible problems and elegant proofs of impossibility.

Universal Machines

An Observation

- When we've been discussing Turing machines, we've talked about designing specific TMs to solve specific problems.
- Does this match your real-world experiences? Do you have one computing device for each task you need to perform?

Computers and Programs

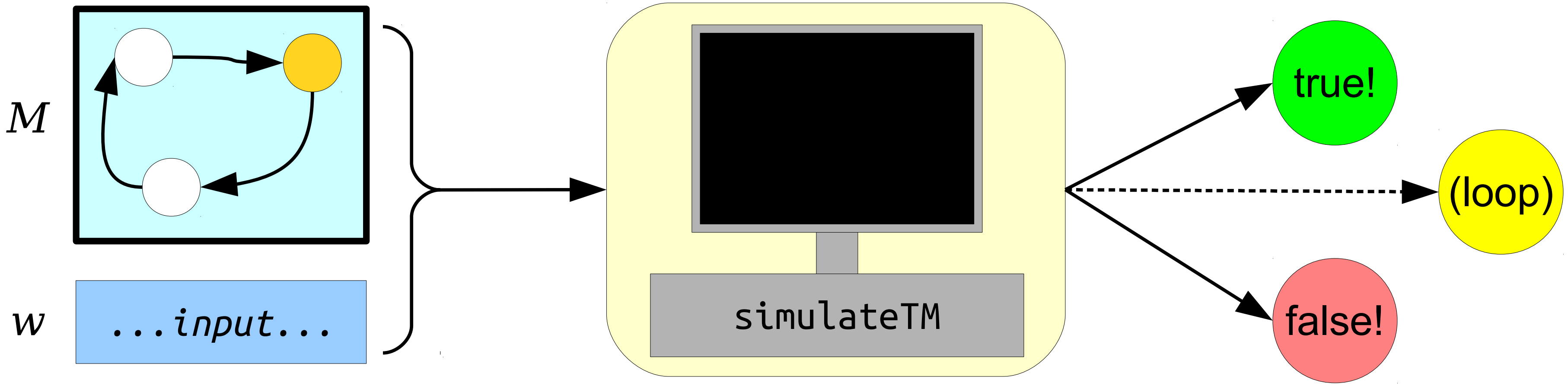
- When talking about actual computers, most people just have a single computer.
- To get the computer to perform a particular task, we load a program into it and have the computer execute that program.
- In certain cases it's faster or more efficient to make dedicated hardware to solve a problem, but the benefits of having one single computer outweigh the costs.
- ***Question:*** Can we do something like this for Turing machines?

A TM Simulator

- It is possible to program a TM simulator on an unbounded-memory computer.
  - In fact, we did this! It's on the CS103 website.
- We could imagine it as a method

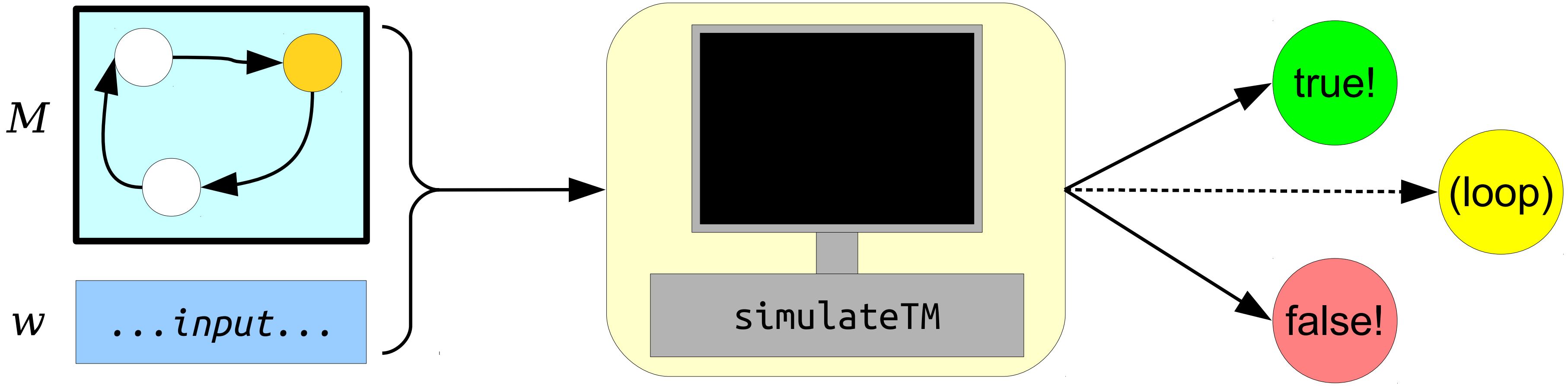
  `boolean simulateTM(TM M, string w)`

  with the following behavior:
  - If M accepts w, then `simulateTM(M, w)` returns **`true`**.
  - If M rejects w, then `simulateTM(M, w)` returns **`false`**.
  - If M loops on w, then `simulateTM(M, w)` loops infinitely.



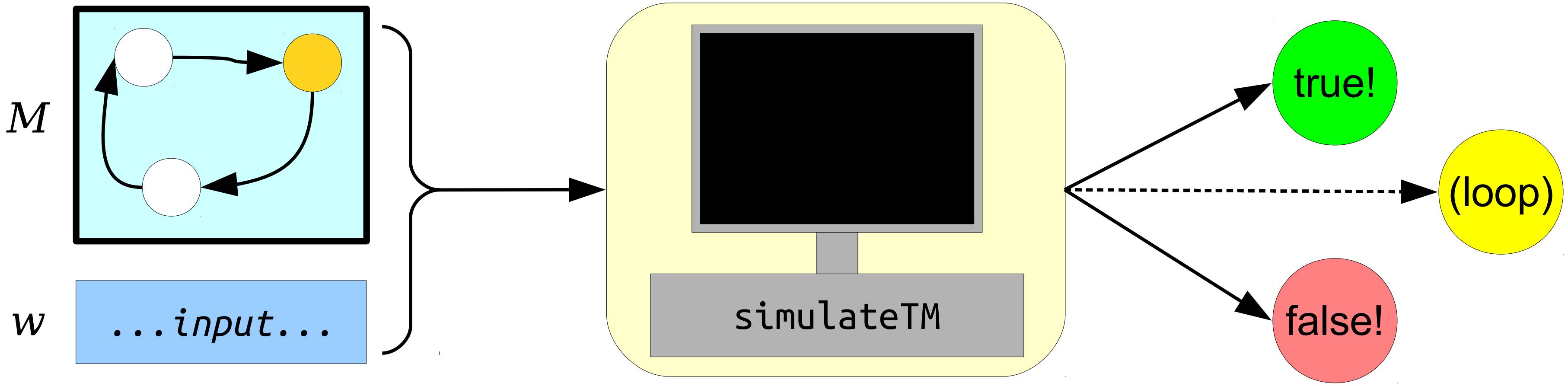
A TM Simulator

- It is known that anything that can be done with an unbounded-memory computer can be done with a TM.



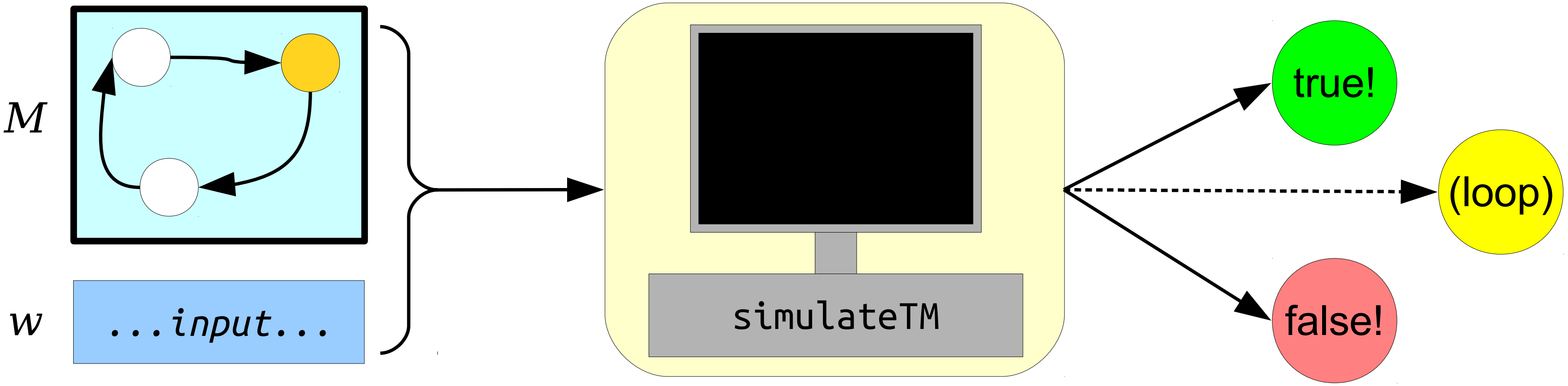
A TM Simulator

- It is known that anything that can be done with an unbounded-memory computer can be done with a TM.
- This means that there must be some TM that has the behavior of this `simulateTM` method.



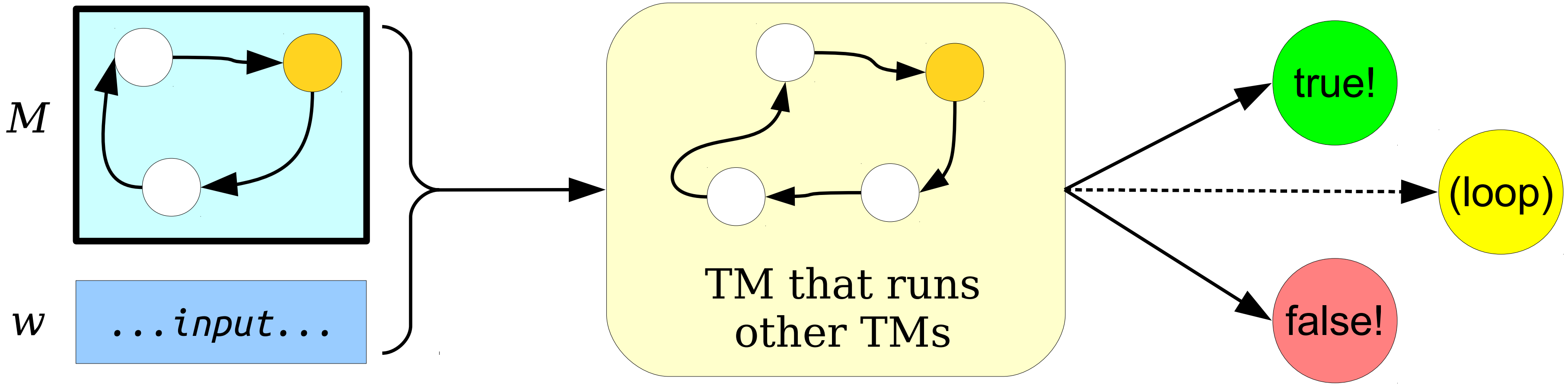
A TM Simulator

- It is known that anything that can be done with an unbounded-memory computer can be done with a TM.
- This means that there must be some TM that has the behavior of this `simulateTM` method.
- What would that look like?



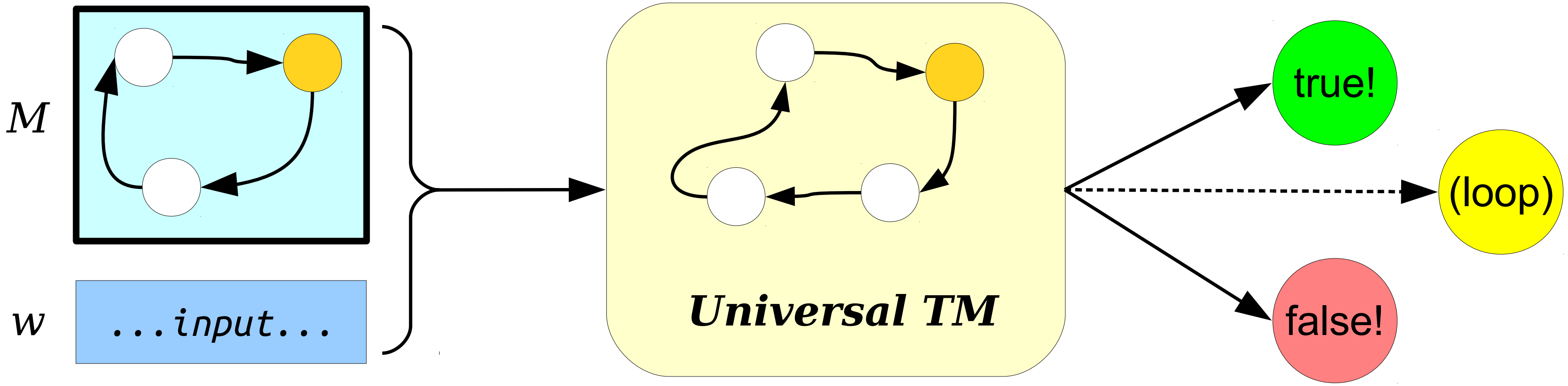
A TM Simulator

- It is known that anything that can be done with an unbounded-memory computer can be done with a TM.
- This means that there must be some TM that has the behavior of this `simulateTM` method.
- What would that look like?



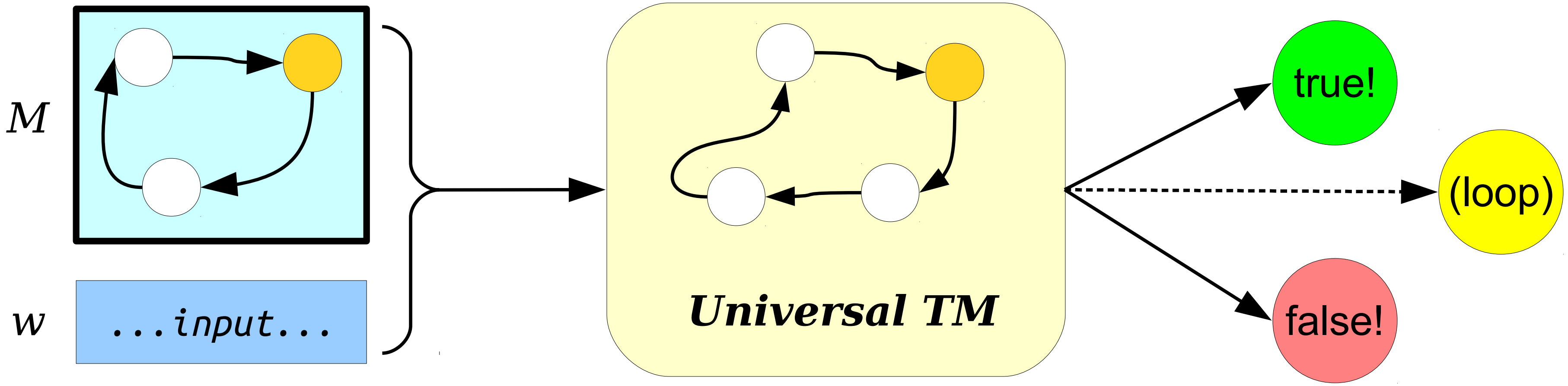
A TM Simulator

- It is known that anything that can be done with an unbounded-memory computer can be done with a TM.
- This means that there must be some TM that has the behavior of this `simulateTM` method.
- What would that look like?



The Universal Turing Machine

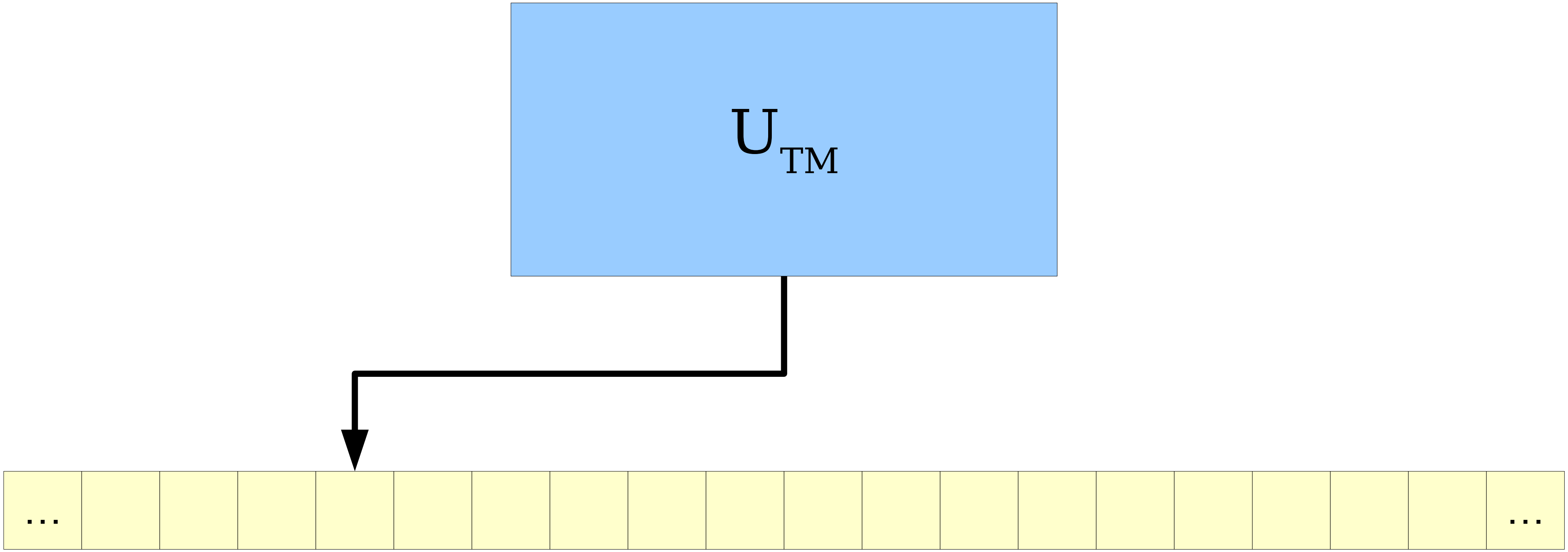
- ***Theorem (Turing, 1936)****:* There is a Turing machine $\mathbf{U_{TM}}$ called the ***universal Turing machine*** that, when run on an input of the form $\langle M, w \rangle$, where M is a Turing machine and w is a string, simulates M running on w and does whatever M does on w (accepts, rejects, or loops).
- The observable behavior of U_{TM} is the following:
  - If M accepts w, then U_{TM} accepts $\langle M, w \rangle$.
  - If M rejects w, then U_{TM} rejects $\langle M, w \rangle$.
  - If M loops on w, then U_{TM} loops on $\langle M, w \rangle$.
- **U_{TM} accepts $\langle M, w \rangle$ if and only if M accepts w.**



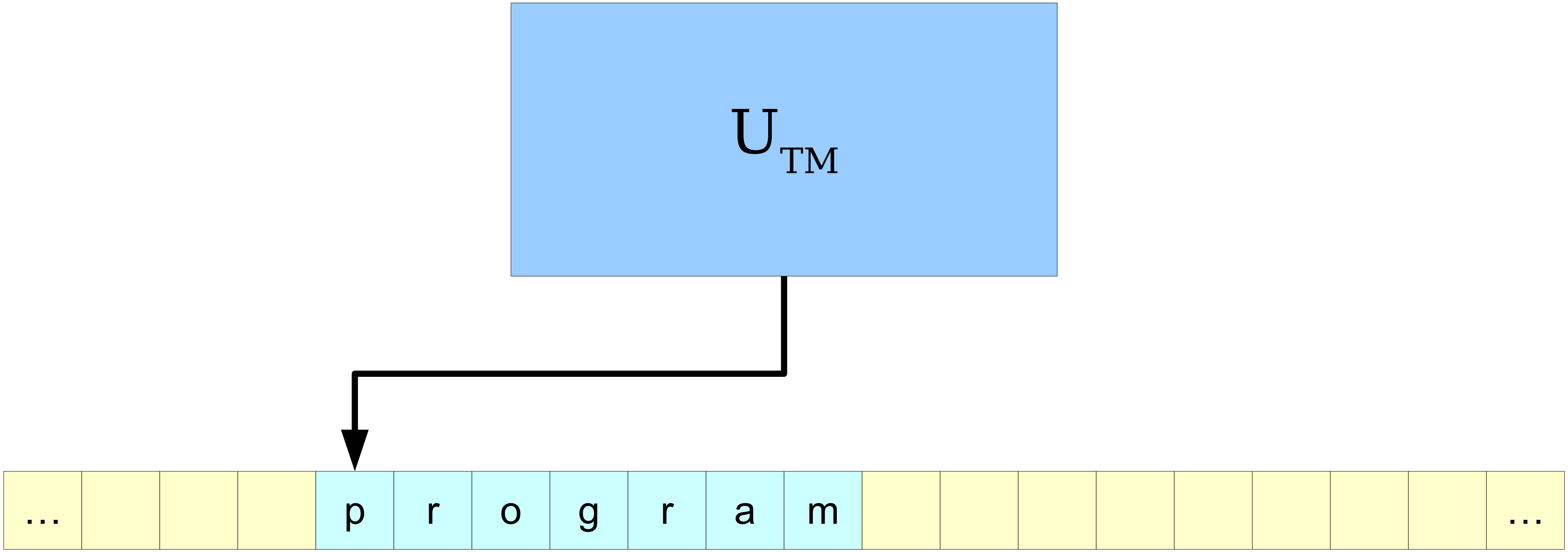
An Intuition for U_{TM}

- You can think of U_{TM} as a general-purpose, programmable computer.
- Rather than purchasing one TM for each language, just purchase U_{TM} and program in the "software" corresponding to the TM you actually want.
- U_{TM} is a powerful machine: ***it can perform any computation that could be performed by any feasible computing device!***

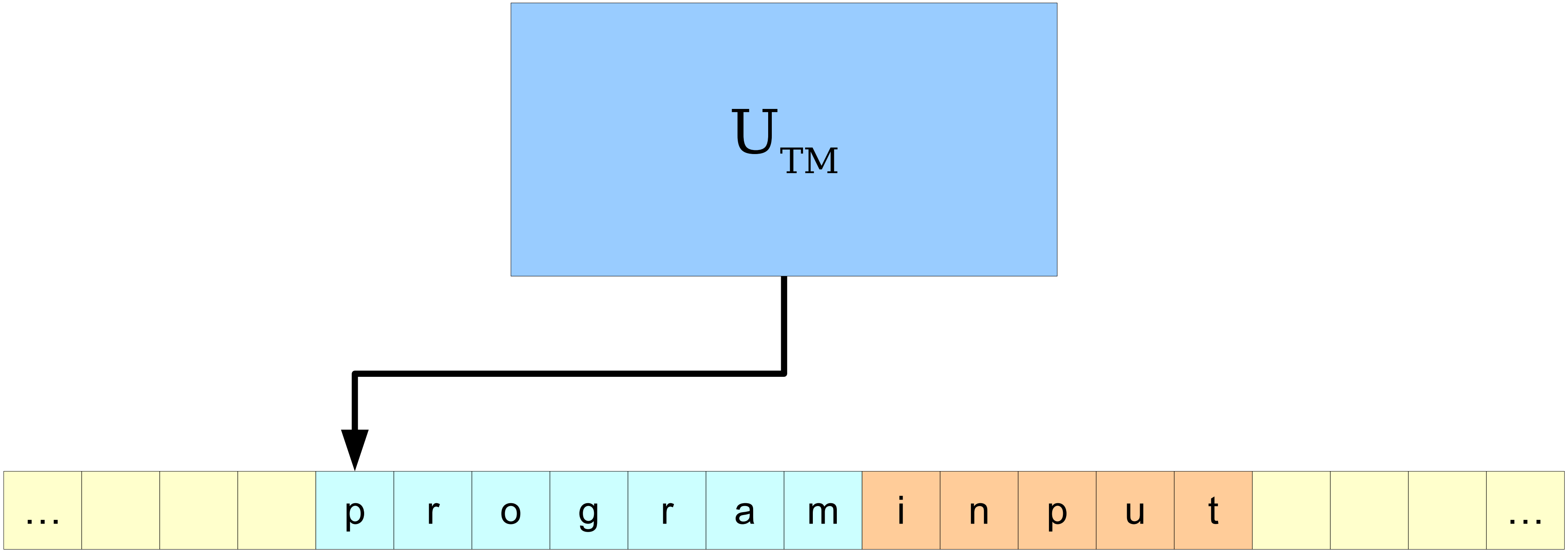
A Universal Machine



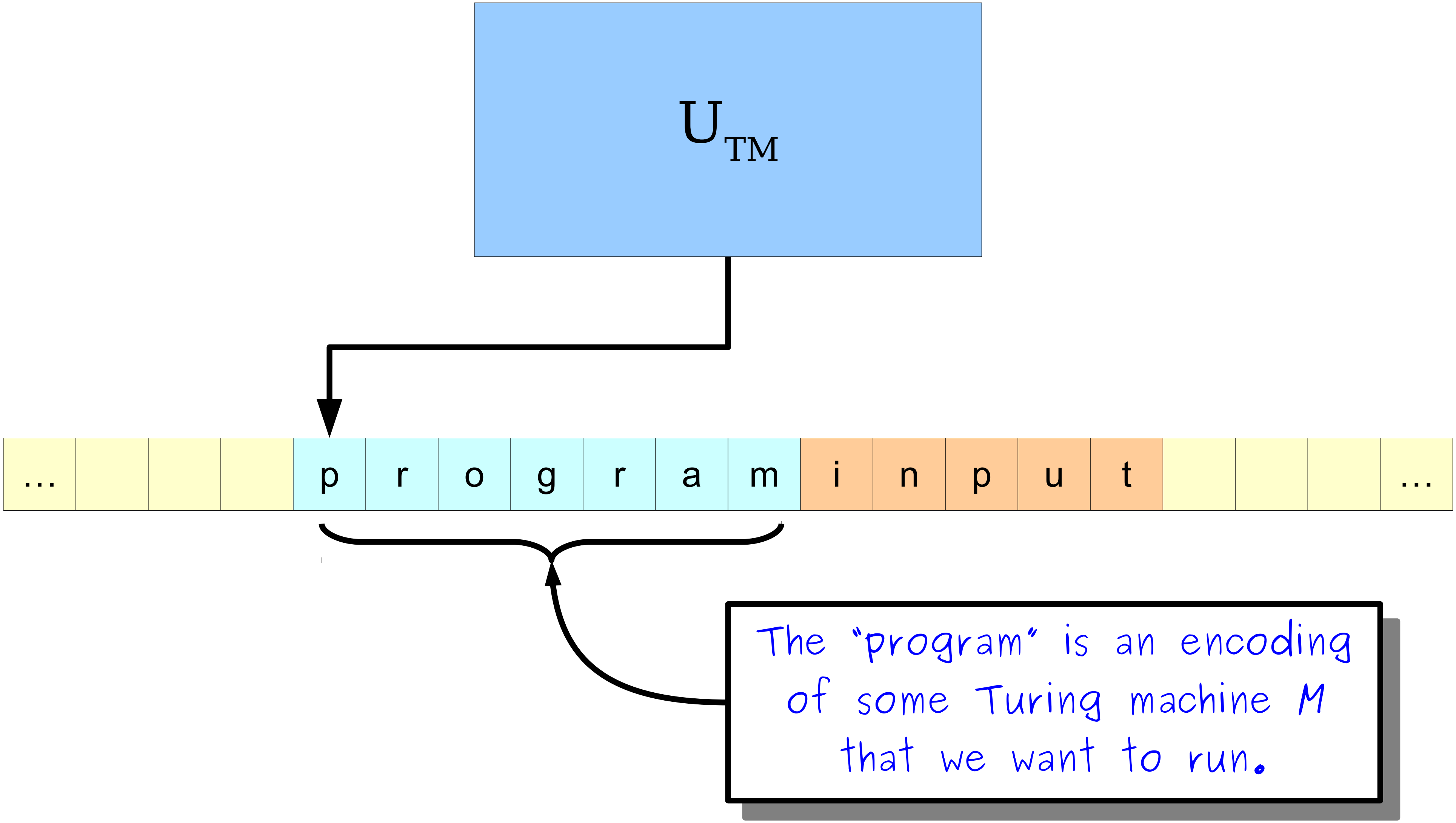
A Universal Machine



A Universal Machine



A Universal Machine

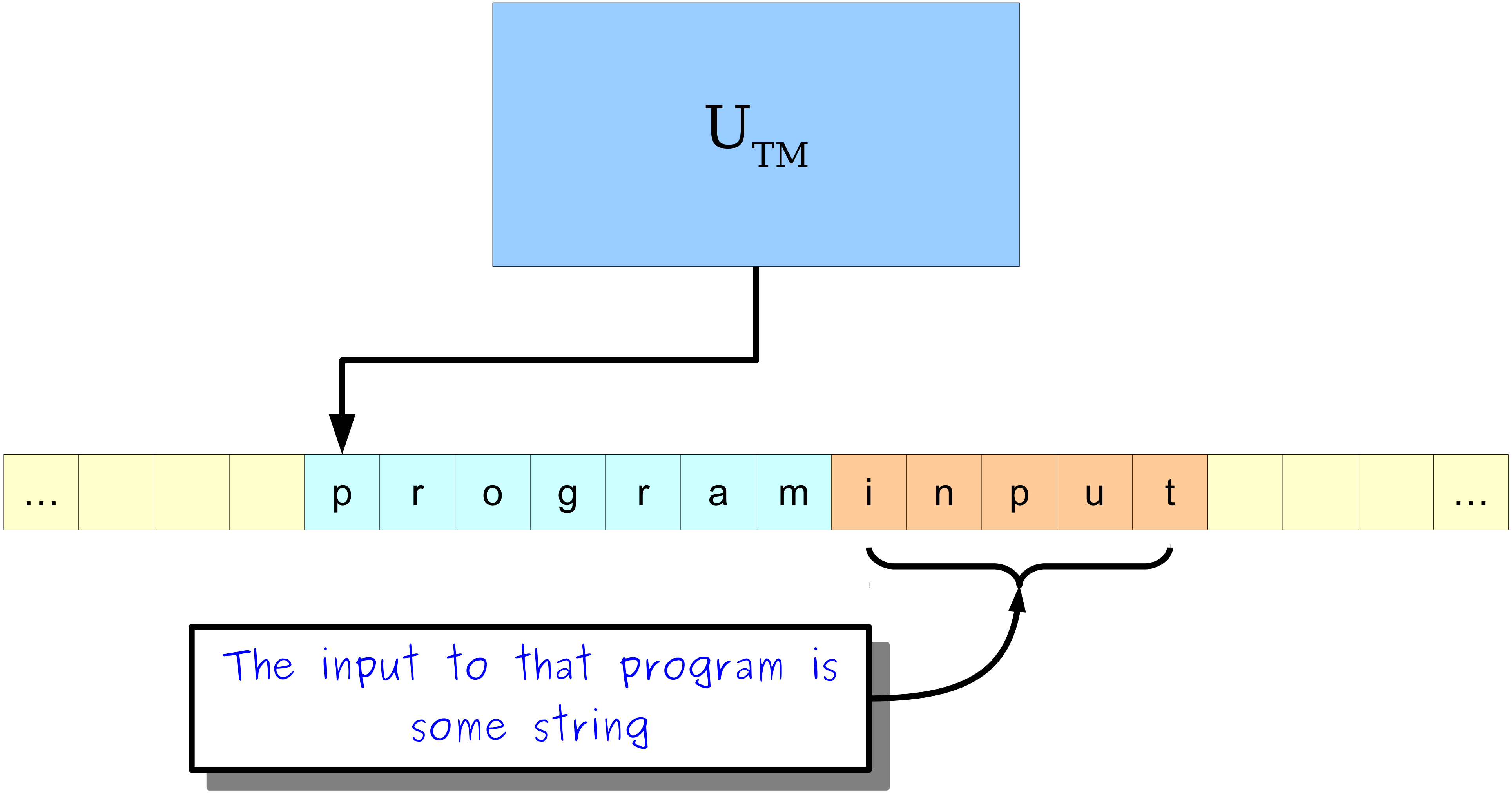


U_{TM}

… p r o g r a m i n p u t …

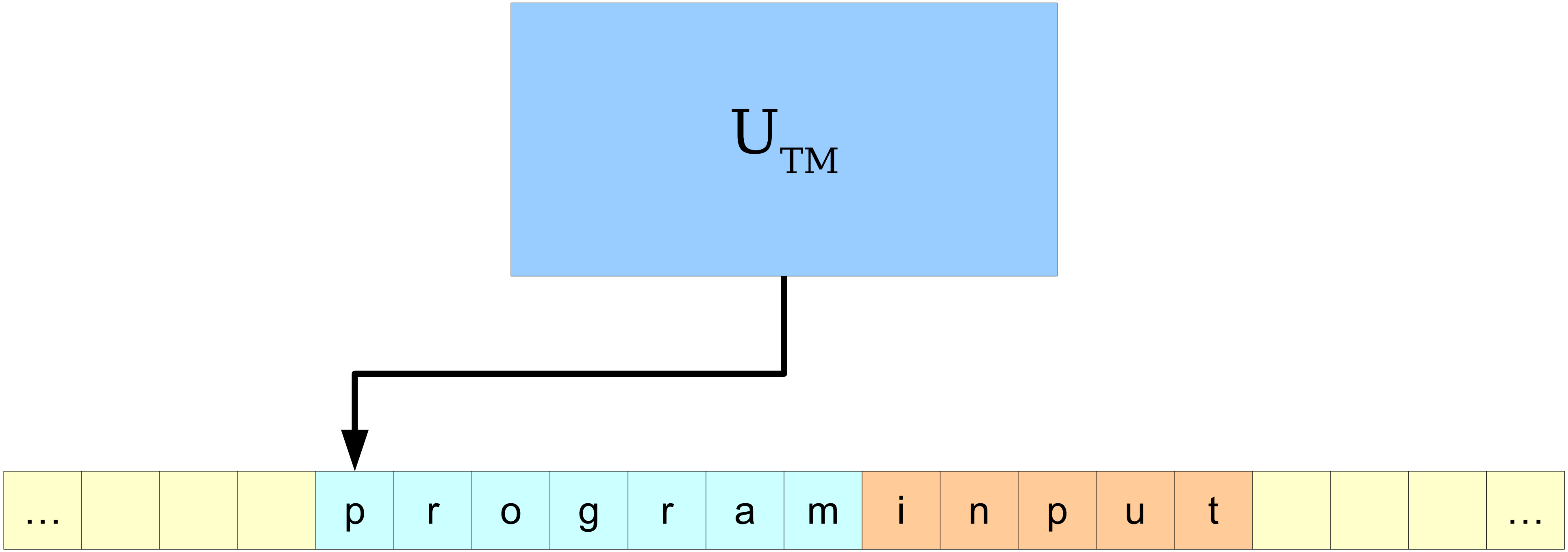
The "program" is an encoding of some Turing machine M that we want to run.

A Universal Machine



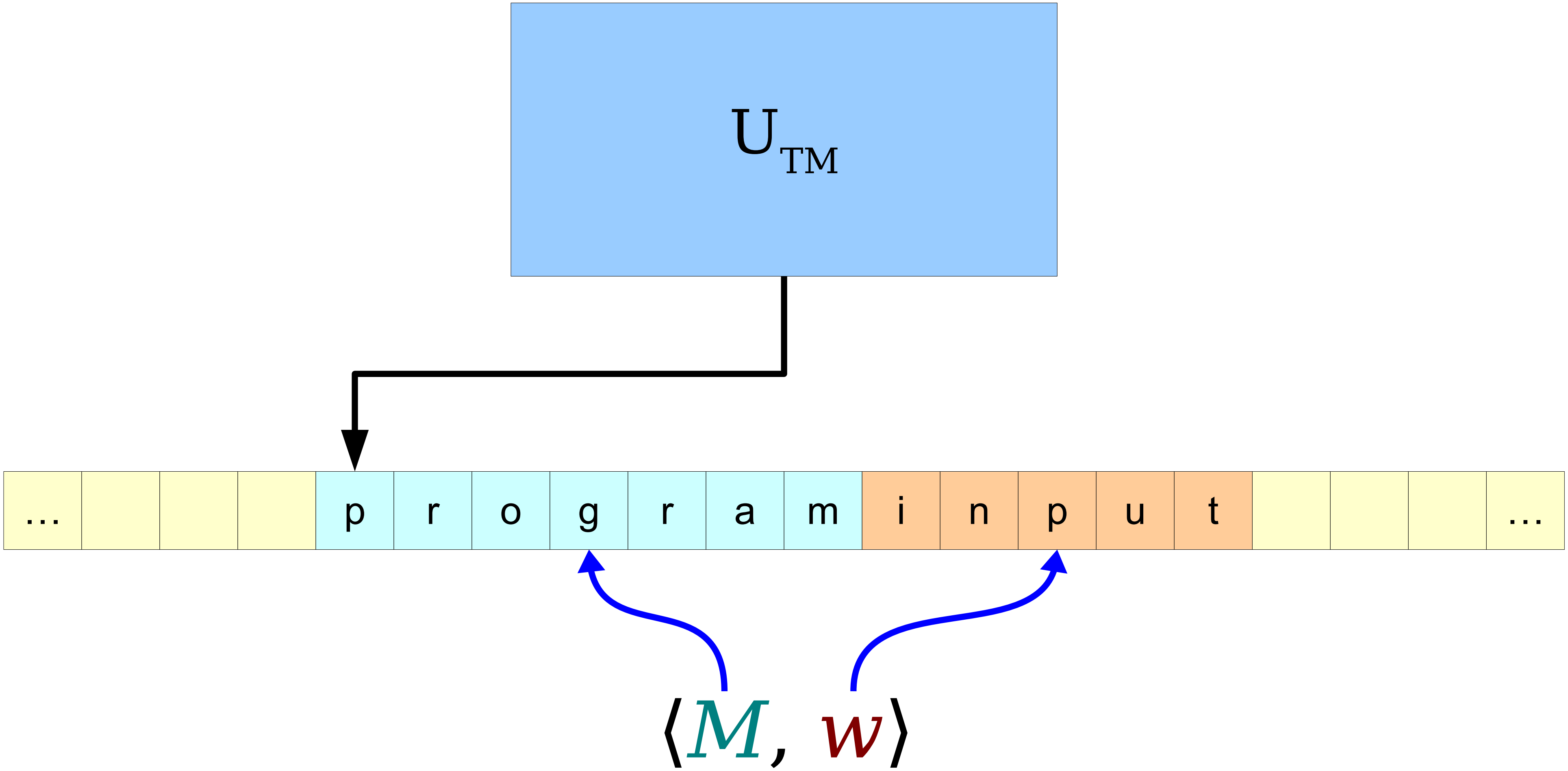
The input to that program is some string

A Universal Machine



The input has the form ⟨M, w⟩, where M is some TM and w is some string.

A Universal Machine



$\langle M, w \rangle$

Since U_{TM} is a TM, it has a language.

What is the language of the universal Turing machine?

The Language of U_{TM}

- Recall: For any TM M, the language of M, denoted $\mathscr{L}(M)$, is the set

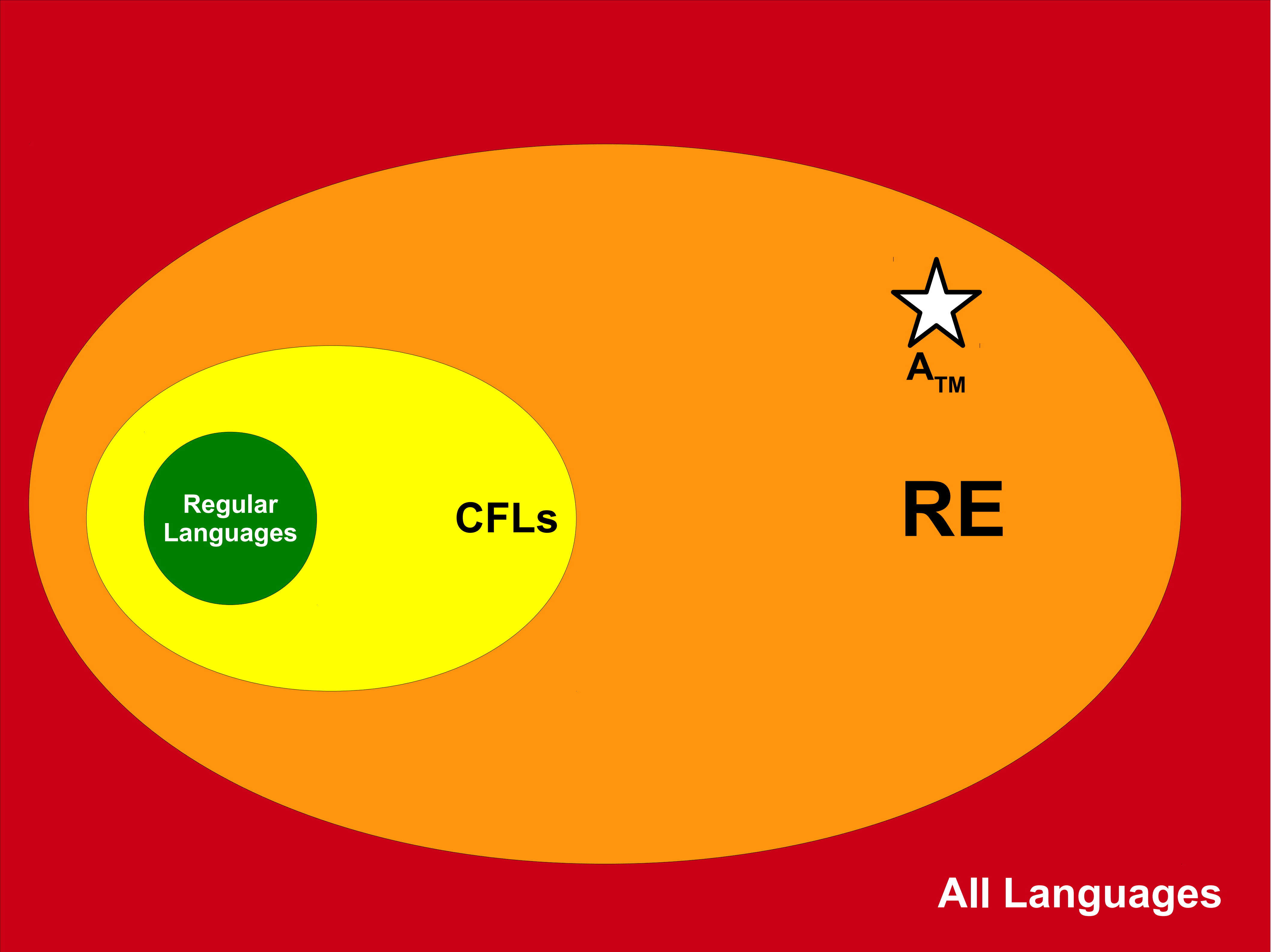
$$\mathscr{L}(M) = \{\ w \in \Sigma^* \mid M \text{ accepts } w\ \}$$

- What is the language of U_{TM}?
- U_{TM} accepts $\langle M, w\rangle$ iff M is a TM that accepts w.
- Therefore:

$$\mathscr{L}(U_{TM}) = \{\ \langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\ \}$$

$$\mathscr{L}(U_{TM}) = \{\ \langle M, w\rangle \mid M \text{ is a TM and } w \in \mathscr{L}(M)\ \}$$

- For simplicity, define $\mathbf{A_{TM}} = \mathscr{L}(\mathbf{U_{TM}})$. This is an important language and we'll see it many times.



Regular Languages

CFLs

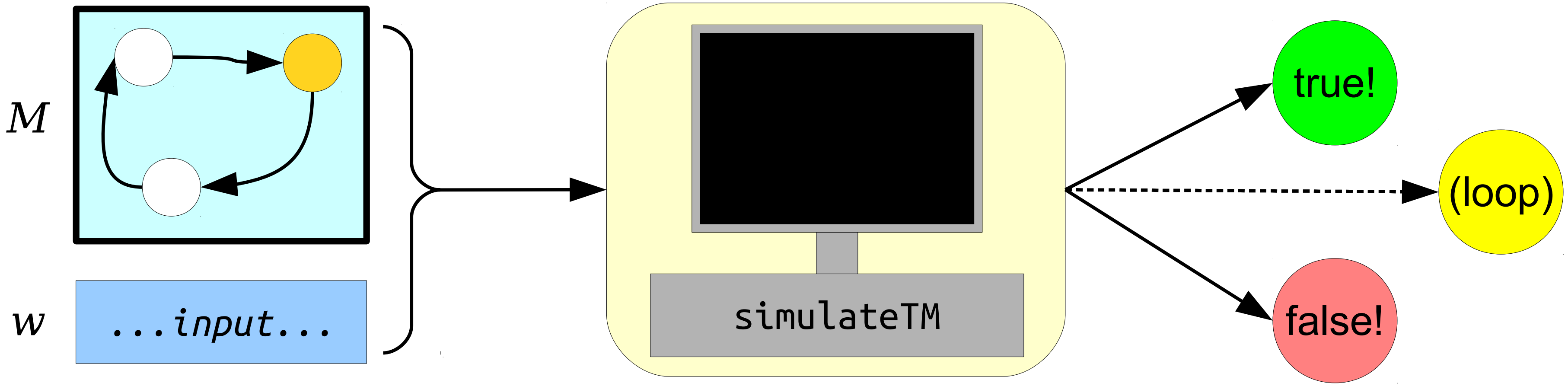
A_{TM}

RE

All Languages

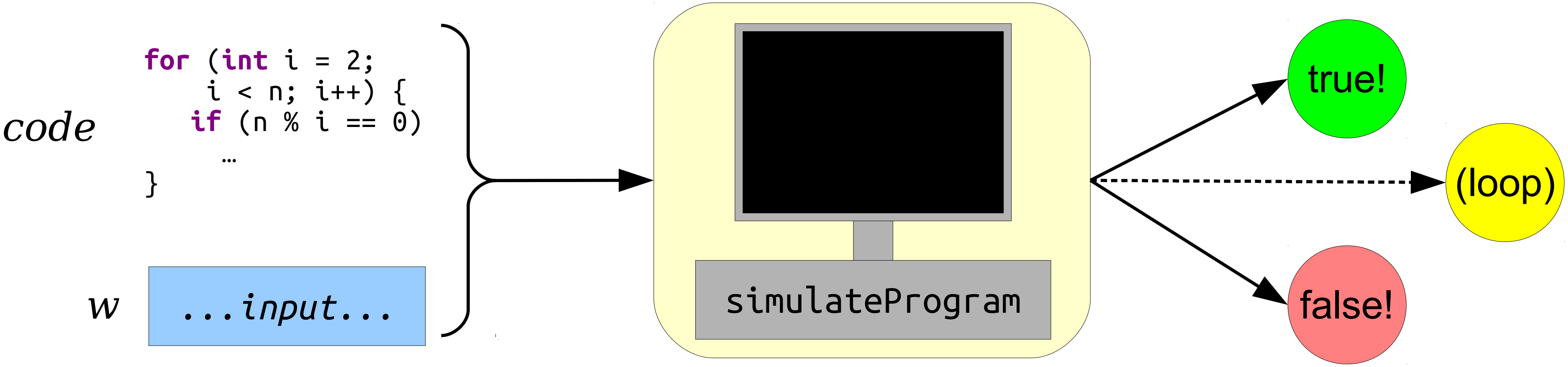
Why Does This Matter?

- The existence of a universal Turing machine has both theoretical and practical significance.
- For a practical example, let's review this diagram from before.
- Previously we replaced the *computer* with a TM. (This gave us the universal TM.)
- What happens if we replace the *TM* with a computer program?



Why Does This Matter?

- The existence of a universal Turing machine has both theoretical and practical significance.
- For a practical example, let's review this diagram from before.
- Previously we replaced the *computer* with a TM. (This gave us the universal TM.)
- What happens if we replace the *TM* with a computer program?



Universal Computers

- In the context of TMs, a TM that simulates other TMs is called a universal TM.
- In the context of computers, a program that simulates other programs goes by many names:
    - An ***interpreter***, like the Java Virtual Machine.
    - An ***emulator***, like VirtualBox.
- The existence of the universal TM means that ***any model of computation equal to a Turing machine can simulate itself!***

Why Does This Matter?

- The key idea behind the universal TM is that idea that TMs can be fed as inputs into other TMs.
  - Similarly, an interpreter is a program that takes other programs as inputs.
  - Similarly, an emulator is a program that takes entire computers as inputs.
- This hits at the core idea that ***computing devices can perform computations on other computing devices.***

Next Time

- ***Self-Reference***
    - Half party trick, half fundamental property of computing, half ancient source of philosophical questions.
- ***Undecidable Problems***
    - A truly impossible problem!