CS103, Autumn 2026



Lecture 22:

Turing Machines

Part 3 of 3

Turing Machines

Part 3

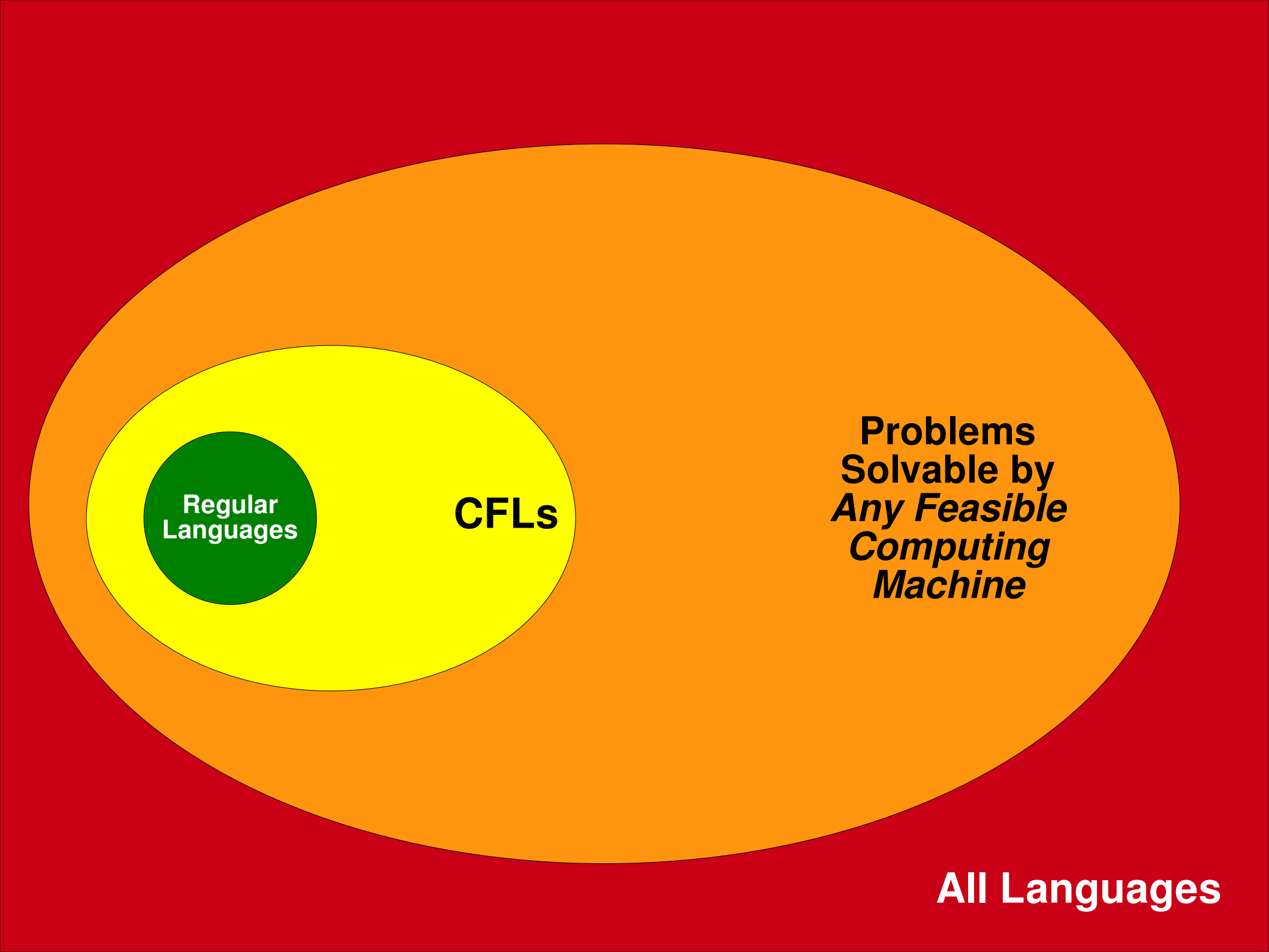
1. **Recap from Last Time**
2. Today's Agenda
3. Problems and Strings
4. Emergent Properties
5. Universal Machines
6. U_{TM} as a Recognizer for A_{TM}
7. Why Does This Matter?
8. Self-Referential Software
9. What's Next?

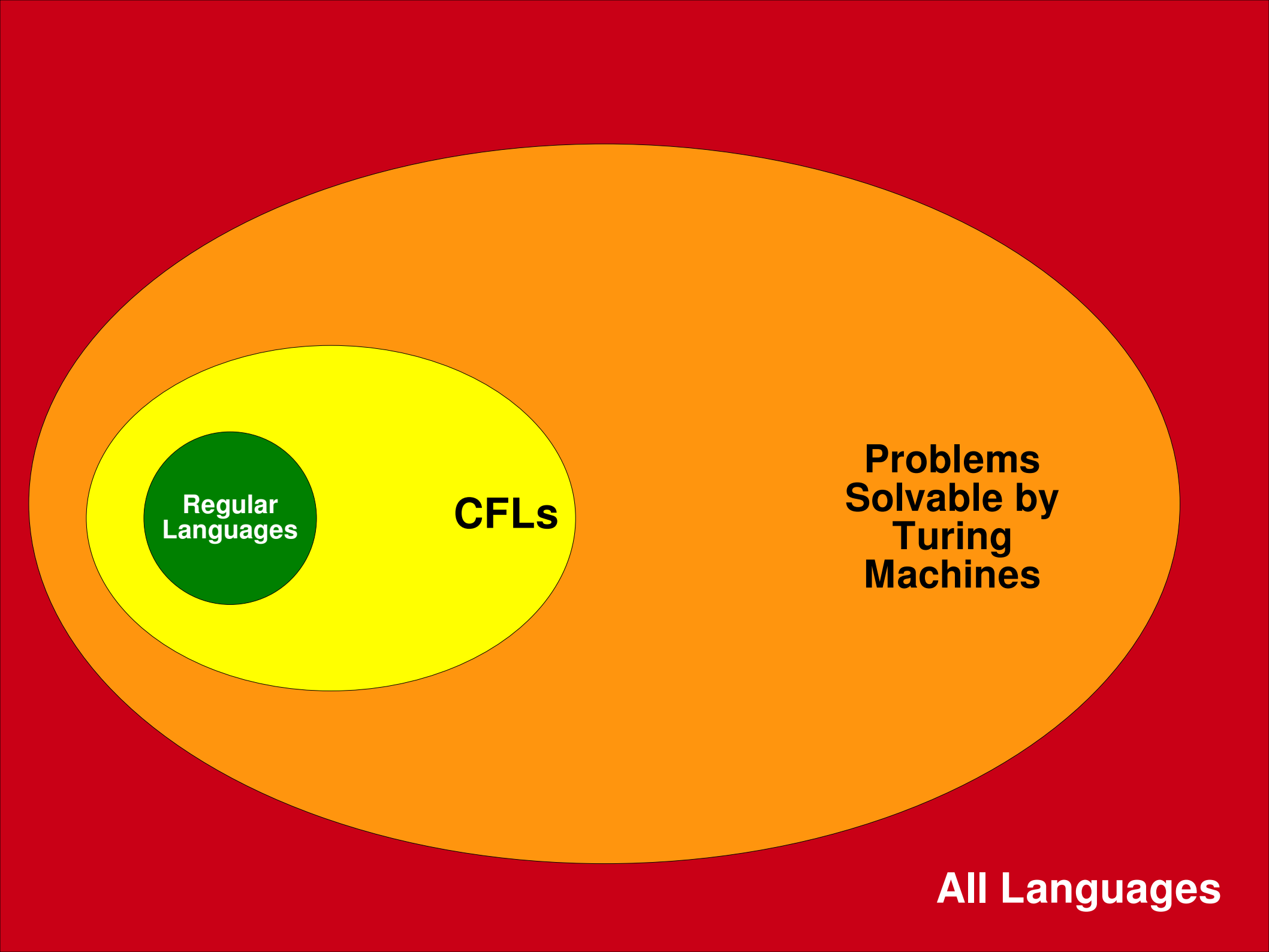
The ***Church-Turing Thesis*** claims that

every feasible 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!"

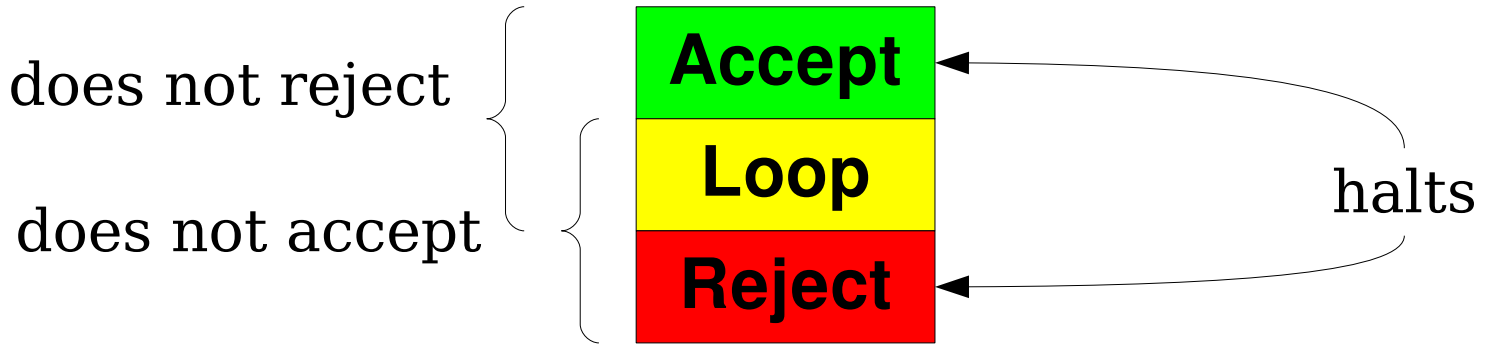
- Ryan Williams


Regular Languages
CFLs
Problems Solvable by *Any Feasible Computing Machine*
All Languages


Regular Languages
CFLs
Problems Solvable by Turing Machines
All Languages

Very Important Terminology

- Let M be a Turing machine.
- M ***accepts*** a string w if it returns true on w.
- M ***rejects*** a string w if it returns false on w.
- M ***loops infinitely*** (or just ***loops***) on a string w if when run on w it neither returns true nor returns false.
- M ***does not accept w*** if it either rejects w or loops 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.



Recognizers and Recognizability

- A TM M is called a ***recognizer*** for a language L over Σ if the following statement is true:

$$\forall w \in \Sigma^*. (w \in L \leftrightarrow M \text{ accepts } w)$$

- A language is ***recognizable*** when there is a recognizer for it.

Which of these statements are true for all choices of TM M, string w, and language L?

(1) If M recognizes L and M rejects w, then $w \notin L$.
(2) If M recognizes L and $w \notin L$, then M rejects w.
(3) If M loops on w and $w \in L$, then M does not recognize L.
(4) If M loops on w and $w \in L$, then L is not recognizable.

Answer at ***https://cs103.stanford.edu/pollev***

Deciders and Decidability

- A TM M is called a ***decider*** for a language L over Σ if the following statements are true:

$$\forall w \in \Sigma^*.\ M \text{ halts on } w.$$

$$\forall w \in \Sigma^*.\ (w \in L \leftrightarrow M \text{ accepts } w)$$

- A language is ***decidable*** when there is a decider for it.

Which of these statements are true for all choices of TM M, string w, and language L?

(1) If M decides L and M rejects w, then $w \notin L$.
(2) If M decides L and $w \notin L$, then M rejects w.
(3) If M loops on w and $w \in L$, then M does not decide L.
(4) If M loops on w and $w \in L$, then L is not decidable.

Answer at ***https://cs103.stanford.edu/pollev***

R and **RE** Languages

- The class **R** consists of all decidable languages.
- The class **RE** consists of all recognizable languages.
- By definition, we know **R** ⊆ **RE**.
- ***Key Question:*** Does **R** = **RE**?

Turing Machines

Part 3

1. Recap from Last Time
2. **Today's Agenda**
3. Problems and Strings
4. Emergent Properties
5. Universal Machines
6. U_{TM} as a Recognizer for A_{TM}
7. Why Does This Matter?
8. Self-Referential Software
9. What's Next?

Today's Agenda

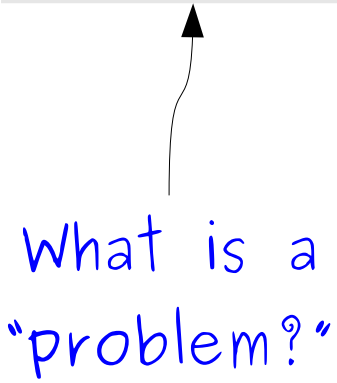
- ***Why Languages and Strings?***
  - We've been using languages to model problems. Why?
- ***Universal Machines***
  - A single computer that can compute anything computable anywhere.
- ***Self-Referential Software***
  - Programs that compute on themselves.

Turing Machines

Part 3

1. Recap from Last Time
2. Today's Agenda
3. **Problems and Strings**
4. Emergent Properties
5. Universal Machines
6. U_{TM} as a Recognizer for A_{TM}
7. Why Does This Matter?
8. Self-Referential Software
9. What's Next?

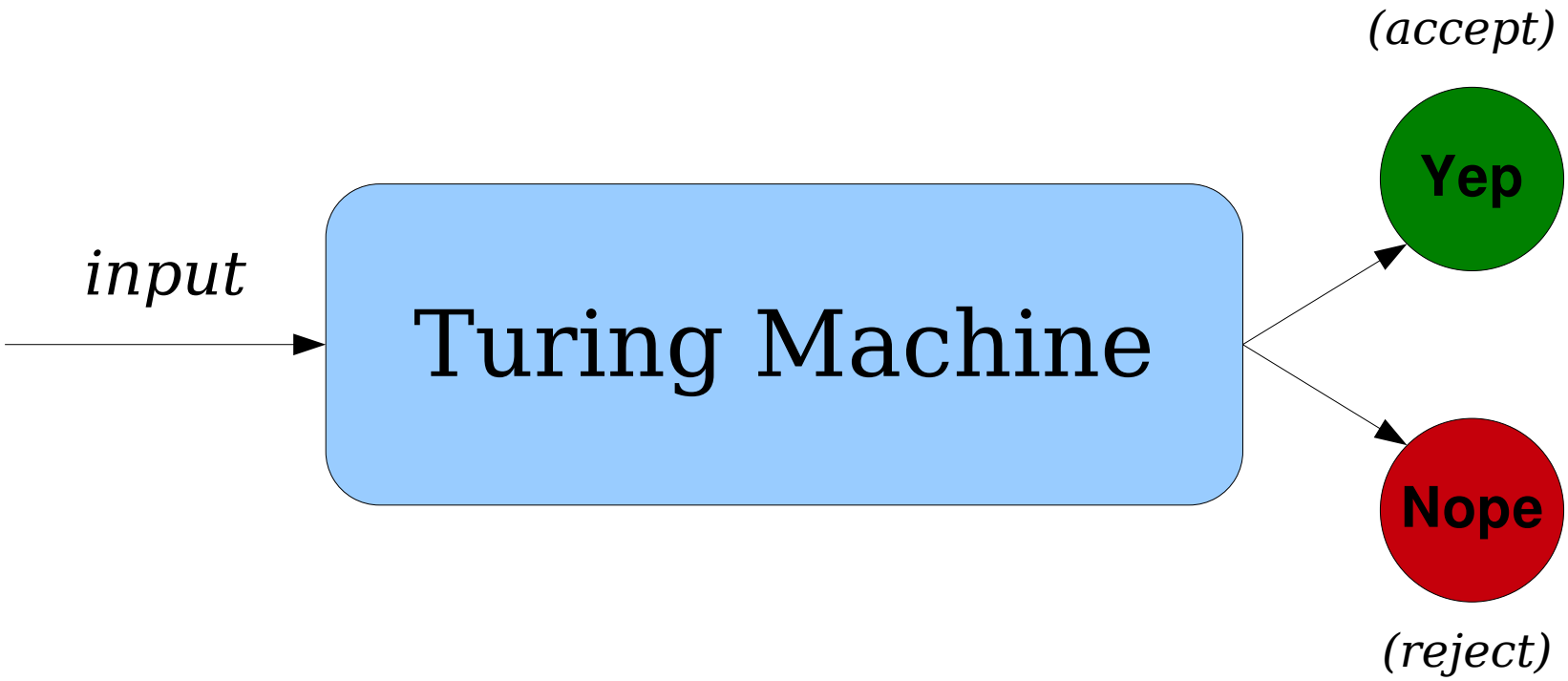
What problems can we solve with a computer?



Decision Problems

- A ***decision problem*** is a problem with a yes-or-no answer.
- For example:
  - "Given integers x, y, and z, is $x + y = z$?" is a decision problem.
  - "Given integers x and y, what is $x + y$?" is ***not*** a decision problem.
- DFAs, NFAs, and TMs solve decision problems: they get an input and produce a yes/no output.

A Model for Solving Problems




```
bool containsCat(Picture P) {

    // … do something …

}
```

How does this match our model?

Humbling Thought:

Everything on your computer is a string over {0, 1}!

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.



Strings and Objects

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



Object Encodings

- If *Obj* is some mathematical object that is *discrete* and *finite*, then we'll use the notation ***⟨Obj⟩*** to refer to some way of encoding that object as a string.
- Think of ⟨*Obj*⟩ like a file on disk – it encodes some high-level object as a series of characters.

⟨  ⟩ = 110111001011…110

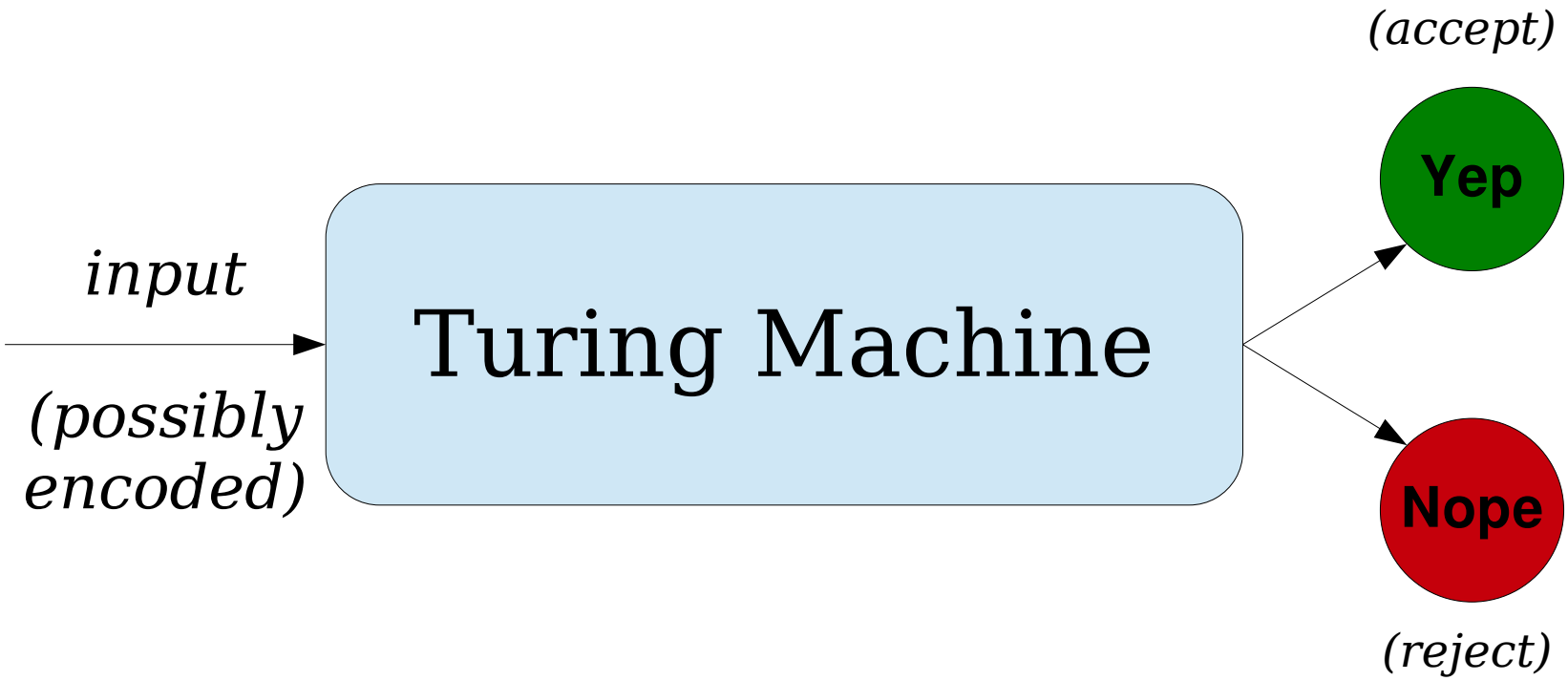
Object Encodings

- For the purposes of what we're going to be doing, we aren't going to worry about exactly *how* objects are encoded.
- For example, we can say ⟨137⟩ to mean "some encoding of 137" without worrying about how it's encoded.
  - Analogy: do you need to know how numbers are represented in Python to be a Python programmer? That's more of a CS107 question.
- We'll assume, whenever we're dealing with encodings, that some Smart, Attractive, Witty person has figured out an encoding system for us and that we're using that encoding system.

Object Encodings

- Object encodings let us define languages like these:
  - $\{ \langle n \rangle \mid n \in \mathbb{N}$ and n is even $\}$
  - $\{ \langle n \rangle \mid n \in \mathbb{N}$ and the Hailstone sequence terminates for $n \}$
  - $\{ \langle G \rangle \mid G$ is a graph and G is bipartite $\}$
- We can focus more on what property of an object we're checking for rather than how the object is represented as a string.

A Model for Solving Problems




```
bool matchesRegex(string w, Regex R) {

    // … do something …

}
```

How does this match our model?

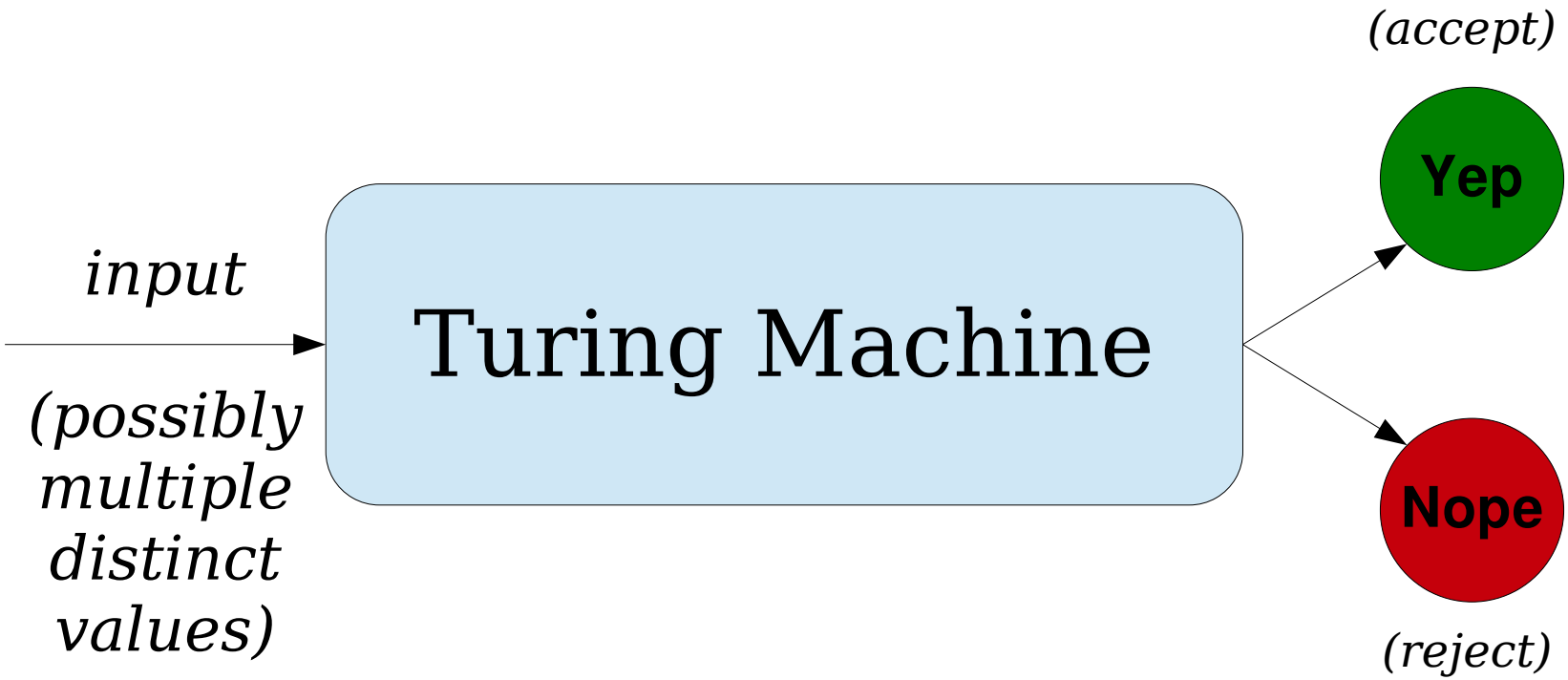
Encoding Groups of Objects

- Given a group of objects Obj_1, Obj_2, …, Obj_n, we can create a single string encoding all these objects.
  - ***Intuition 1:*** Think of it like a .zip file, but without the compression.
  - ***Intuition 2:*** Think of it like a tuple or `struct`.
- We'll denote the encoding of all of these objects as a single string by $\langle Obj_1, \ldots, Obj_n \rangle$.

Encoding Groups of Objects

- We can now talk about languages like these:
  - $\{ \langle R, w \rangle \mid R$ is a regex and R matches $w \}$
  - $\{ \langle G, s, t \rangle \mid G$ is a graph, s and t are nodes in G, and there's a path from s to $t \}$
- Our languages are starting to look a lot more like problems in the traditional sense than sets of strings.

A Model for Solving Problems




```
bool matchesRegex(string w, Regex R) {

    // … do something …

}
```

These form one large bitstring.

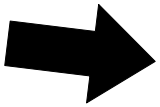
Summary

- If *Obj* is some mathematical object that is *discrete* and *finite*, then we'll use the notation $\langle Obj \rangle$ to refer to some way of encoding that object as a string.
- We can use this notation to feed objects to TMs even if we might not usually conceive of those objects as strings.
- We can also use the $\langle Obj_1, \ldots, Obj_n \rangle$ notation to feed *groups* of objects to TMs.

Turing Machines

Part 3

1. Recap from Last Time
2. Today's Agenda
3. Problems and Strings
4. Emergent Properties
5. Universal Machines
6. U_{TM} as a Recognizer for A_{TM}
7. Why Does This Matter?
8. Self-Referential Software
9. What's Next?

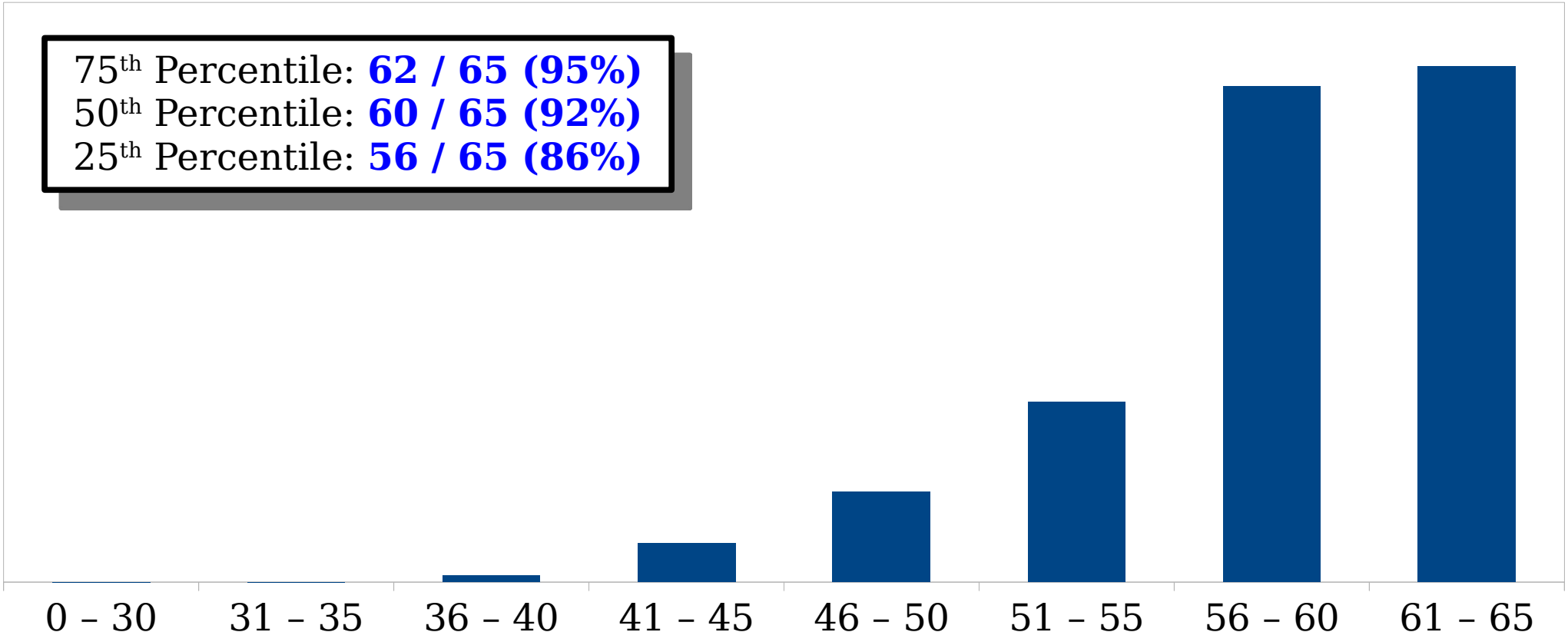


But first, some announcements!

Announcements

- PS6 solutions are available on the course website. (They were posted Monday, but I had not announced that in class until now.)
- PS7 was due today (Friday) at 1:00 PM.
- PS8 will go out later today (by 4:00 PM) and is due next week on **Sunday**, Nov. 22, at 1:00 PM as usual.
- PS8 will be the last problem set where you can use late days! Late days cannot be used for PS9.
- Have questions? Come talk to us in office hours, or post online on Ed!

Problem Set Six Graded



Turing Machines

Part 3

1. Recap from Last Time
2. Today's Agenda
3. Problems and Strings
4. **Emergent Properties**
5. Universal Machines
6. U_{TM} as a Recognizer for A_{TM}
7. Why Does This Matter?
8. Self-Referential Software
9. What's Next?

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 consciousness, love, and ennui.
  - 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

- Computational devices (TMs and other equivalent devices) have two surprising emergent properties:
  - ***Universality***: There is a single computing device capable of performing any computation.
  - ***Self-Reference***: Computing devices can ask questions about their own behavior.
- These properties are, in a sense, inherent to all computing devices. Computing can't exist without them.
- These properties are interesting in their own rights – and are the theoretical basis for much of modern computing.
- They also are an "Achilles' heel" of computational devices, and we'll use them to find concrete examples of problems computers can't solve.

Turing Machines

Part 3

1. Recap from Last Time
2. Today's Agenda
3. Problems and Strings
4. Emergent Properties
5. **Universal Machines**
6. U_{TM} as a Recognizer for A_{TM}
7. Why Does This Matter?
8. Self-Referential Software
9. What's Next?

An Observation

- Think about how you interact with your physical computer.
  - You have a single, physical computer.
  - That computer then runs multiple programs.
- Contrast that with how we've worked with TMs.
  - We have a TM for $\{ \mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N} \}$. That TM will always perform that calculation and never do anything else.
  - We have a TM for the hailstone sequence. That TM can't compose poetry, write music, etc.
- How do we reconcile this difference?

An Observation

- Think about how you interact with your physical computer.
  - You have a single, physical computer.
  - That com…s.
- Contrast tha…s.
  - We have …will always perform …ing else.
  - We have a TM for the hailstone sequence. That TM can't compose poetry, write music, etc.
- How do we reconcile this difference?

Can we make a "reprogrammable Turing machine?"

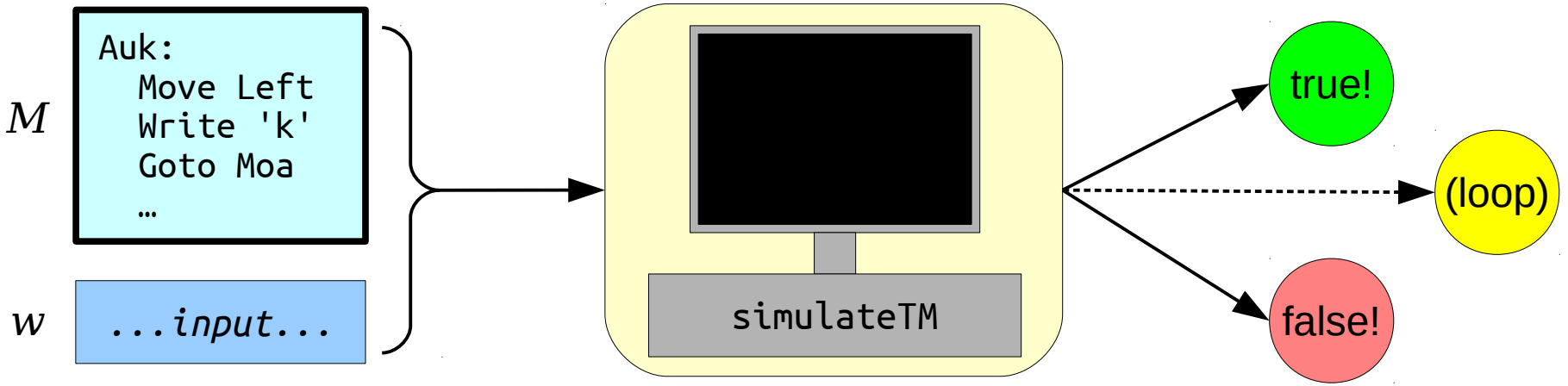
A TM Simulator

- It's possible to program a TM simulator on an unbounded-memory computer.
  - You've seen this in class, and you're using one in PS8.
- We could imagine it as a method

```
bool simulateTM(TM M, string w)
```

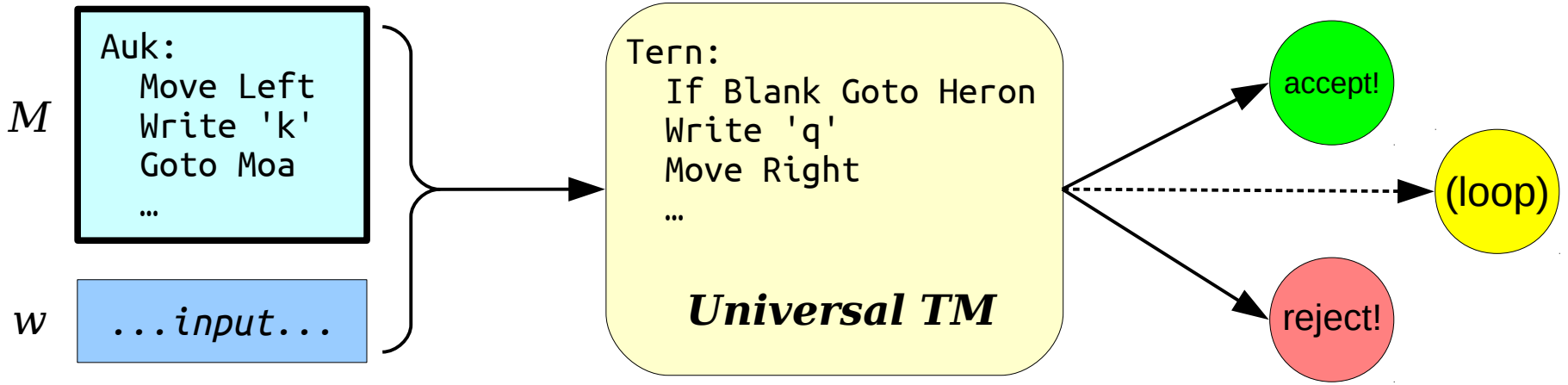
  with the following behavior:

  - 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

- Anything that can be done with an unbounded-memory computer can be done with a TM.
- So there must be some TM that has the behavior of `simulateTM`.
- What would that TM do?



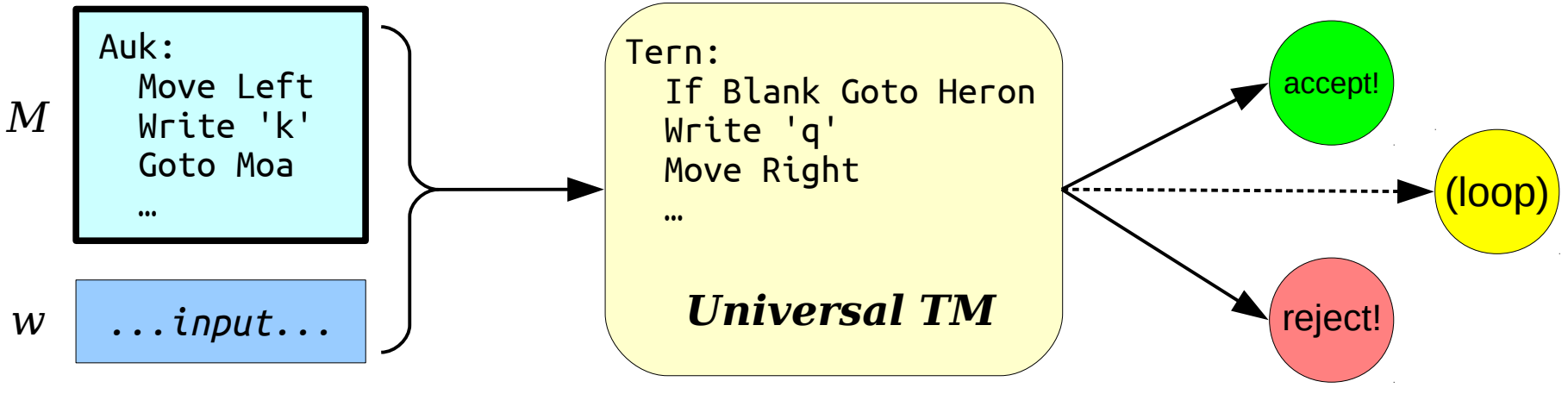
The Universal Turing Machine

- ***Theorem (Turing, 1936)***: There is a Turing machine **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} does to $\langle M, w \rangle$

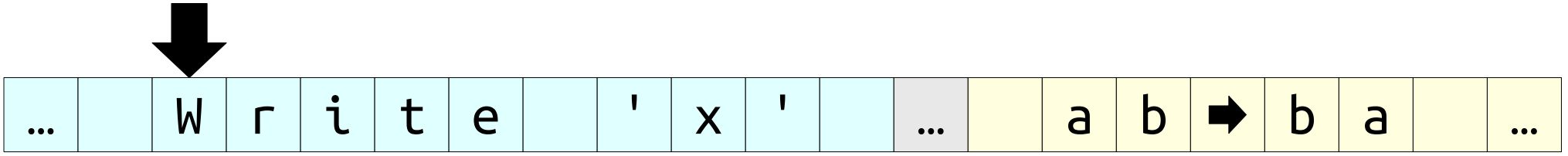
what

M does to w.



The Universal Turing Machine

- Building out U_{TM} is nontrivial, but the conceptual idea behind it isn't too bad.
- Essentially:
  - U_{TM} splits its tape into two regions: one spot holding the source code of the TM to simulate, and one holding the tape contents for that TM.
  - U_{TM} somehow marks where in the simulated TM's tape the simulated TM's tape head is, perhaps by having a special symbol indicating "tape head here."
  - U_{TM} repeatedly consults the source code of the simulated TM to determine what action to take, then simulates that action.
  - If the simulated TM accepts or rejects, then U_{TM} also accepts or rejects.



The Universal Turing Machine

- ***Intuition:*** Modern computers – laptops, phones, network routers, etc. – are universal Turing machines.
  - Each computer is a single piece of hardware. With rare exceptions, we don't make specific changes to the hardware after we purchase the computer.
  - We load programs into those computers, and those computers then execute the commands in those programs.
- Turing came up with this idea in 1936 – before any programmable computers had been built!

The Universal Turing Machine

- ***Amazing Thought:*** U_{TM} is the most powerful computational device that can be built.
  - Assuming the Church-Turing thesis, any computation that can be performed by any computing system can be performed by a TM.
  - The universal TM can "run" any TM, so it can perform any computation any TM can perform.
  - So U_{TM} can do any computation that could ever be done by any possible feasible computing system. *(Wow!)*
- And yet – it's just a simulator! All it does is simulate one step of a TM after another.

Turing Machines

Part 3

1. Recap from Last Time
2. Today's Agenda
3. Problems and Strings
4. Emergent Properties
5. Universal Machines
6. **U_{TM} as a Recognizer for A_{TM}**
7. Why Does This Matter?
8. Self-Referential Software
9. What's Next?

U_{TM} as a Recognizer

- U_{TM}, when run on a string $\langle M, w \rangle$, where M is a TM and w is a string, will
  - … accept $\langle M, w \rangle$ if M accepts w,
  - … reject $\langle M, w \rangle$ if M rejects w, and
  - … loop on $\langle M, w \rangle$ if M loops on w.
- Although we didn't design U_{TM} as a recognizer, it does recognize some language.
- Which language is that?

U_{TM} as a Recognizer

- U_{TM}, when run on a string $\langle M, w \rangle$, where M is a TM and w is a string, will

  ... accept $\langle M, w \rangle$ if M accepts w,

  ... reject $\langle M, w \rangle$ if M rejects w, and

  ... loop on $\langle M, w \rangle$ if M loops on w.

- Let's let $\mathbf{A_{TM}}$ be the language recognized by the universal TM U_{TM}. This means that

$$\forall M. \forall w \in \Sigma^*. (M \text{ accepts } w \leftrightarrow \langle M, w \rangle \in A_{TM})$$

- So we have

$$\mathbf{A_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w \}}$$

The Language A_{TM}

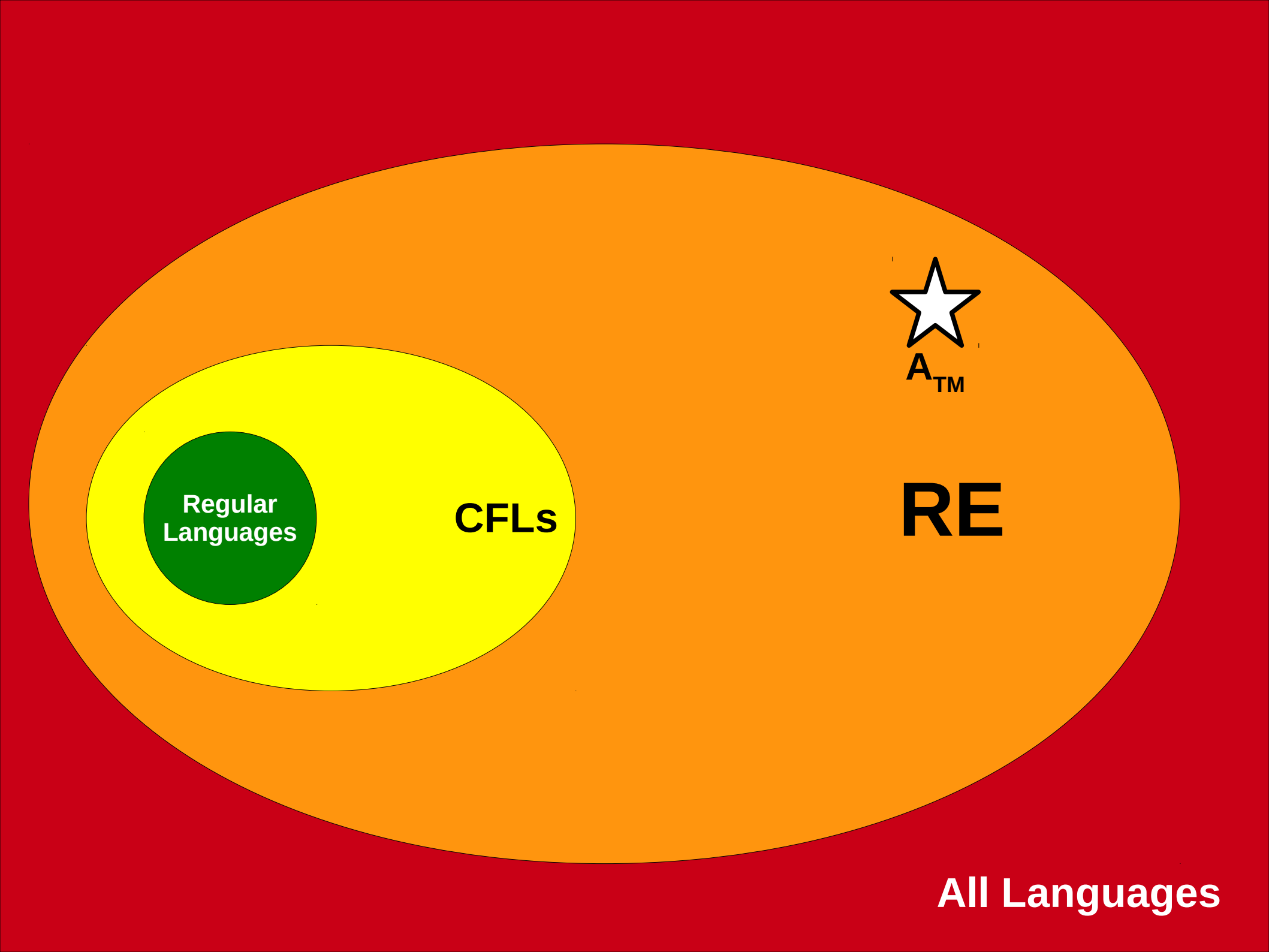
$A_{TM} = \{ \langle M, w \rangle \mid M$ is a TM and M accepts $w \}$

- Here's a complicated expression. Can you simplify it?

$$\langle U_{TM}, \langle N, x \rangle \rangle \in A_{TM}.$$

- Given the definition of A_{TM} and U_{TM}, the following statements are all equivalent to one another.
  - M accepts w.
  - U_{TM} accepts $\langle M, w \rangle$.
  - $\langle M, w \rangle \in A_{TM}$.

Answer at
https://cs103.stanford.edu/pollev


Regular Languages
CFLs
A_{TM}
RE
All Languages

Turing Machines

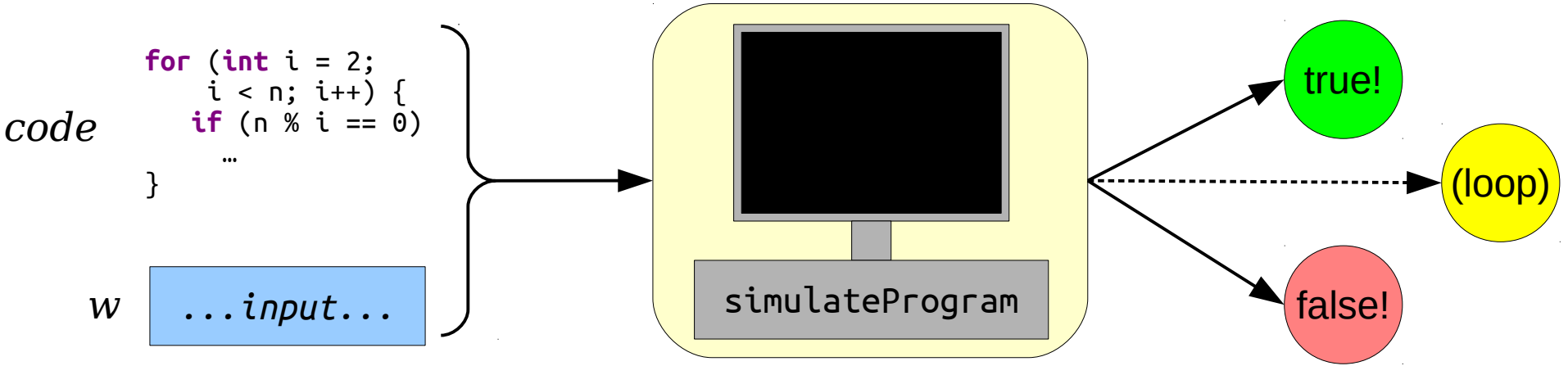
Part 3

1. Recap from Last Time
2. Today's Agenda
3. Problems and Strings
4. Emergent Properties
5. Universal Machines
6. U_{TM} as a Recognizer for A_{TM}
7. **Why Does This Matter?**
8. Self-Referential Software
9. What's Next?

Reason 1: ***It has practical consequences.***

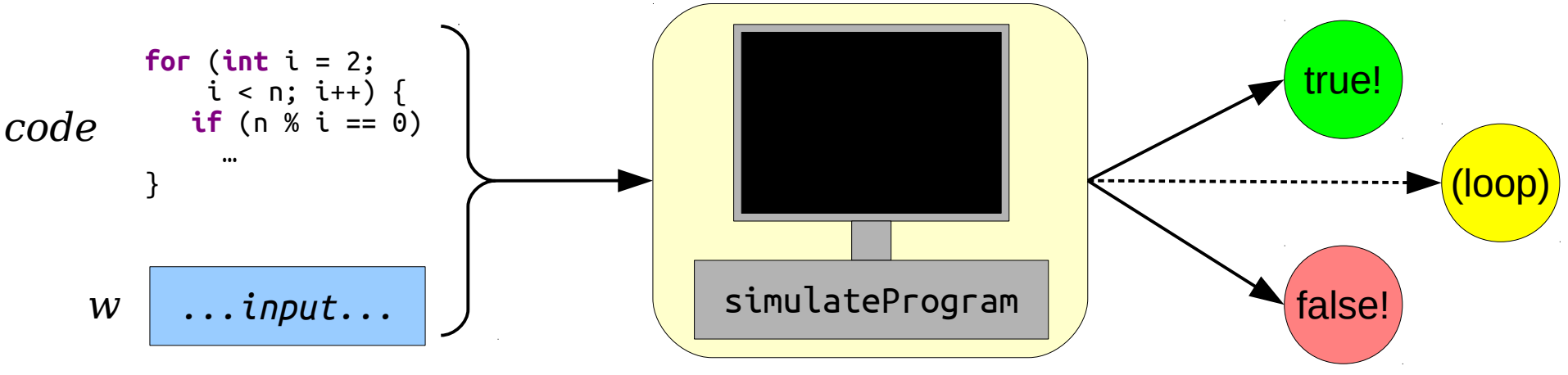
Why Does This Matter?

- 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?

- We now have a computer program that runs other computer programs!
  - An ***interpreter*** is a program that simulates other programs. Python programs are usually executed by interpreters. Your web browser interprets JavaScript code when it visits websites.
  - A ***virtual machine*** is a program that simulates an entire operating system. Virtual machines are used in computer security, cloud computing, and even by individual end users.
- It's not a coincidence that this is possible – Turing's 1936 paper says that any general-purpose computing system must be able to do this!



Reason 2: ***It's philosophically interesting.***

Can Computers Think?

- On May 15, 1951, Alan Turing delivered ***a radio lecture on the BBC*** on the topic of whether computers can think.
- He had the following to say about whether a computer can be thought of as an electric brain...

“In fact I think [computers] could be used in such a manner that they could be appropriately described as brains. I should also say that

‘If any machine can be appropriately described as a brain, then any digital computer can be so described.’

This last statement needs some explanation. It may appear rather startling, but with some reservations it appears to be an inescapable fact.

It can be shown to follow from a characteristic property of digital computers, which I will call their ***universality***. A digital computer is a universal machine in the sense that it can be made to replace any machine of a certain very wide class. It will not replace a bulldozer or a steam-engine or a telescope, but it will replace any rival design of calculating machine, that is to say any machine into which one can feed data and which will later print out results. In order to arrange for our computer to imitate a given machine it is only necessary to programme the the computer to calculate what the machine in question would do under given circumstances, and in particular what answers it would print out. The computer can then be made to print out the same answers.

If now some machine can be described as a brain we have only to programme our digital computer to imitate it and it will also be a brain.”

Turing Machines

Part 3

1. Recap from Last Time
2. Today's Agenda
3. Problems and Strings
4. Emergent Properties
5. Universal Machines
6. U_{TM} as a Recognizer for A_{TM}
7. Why Does This Matter?
8. **Self-Referential Software**
9. What's Next?

Quines

- A ***Quine*** is a program that, when run, prints its own source code.
- Quines aren't allowed to just read the file containing their source code and print it out; that's cheating (and technically incorrect if someone changes that file!)
- How would you write such a program?

Quines

- A ***Quine*** is a program that, when run, prints its own source code.
- Quines are [illegible] their sourc[illegible]eating (and technically [illegible]hat file!)
- How would [illegible]

Let’s write a Quine!

Self-Referential Programs

- The fact that we can write Quines is not a coincidence.

Theorem: It is possible to construct
TMs that perform arbitrary computations
on their own source code.

- In other words, any computing system that's equal to a Turing machine possesses some mechanism for self-reference!
- Want to see how deep the rabbit hole goes? Take CS154!

Self-Referential Programs

- ***Claim:*** Going forward, assume that any function has the ability to get access to its own source code.
- This means we can write programs like the ones shown here:

```
bool narcissist(string input) {
    string me = /* source code of narcissist */;

    return input == me;
}

bool acceptLongerStrings(string input) {
    string me = /* source code of acceptLongerStrings */;

    return input.length() > me.length();
}
```

Turing Machines

Part 3

1. Recap from Last Time
2. Today's Agenda
3. Problems and Strings
4. Emergent Properties
5. Universal Machines
6. U_{TM} as a Recognizer for A_{TM}
7. Why Does This Matter?
8. Self-Referential Software
9. **What's Next?**

What's Next?

- ***Self-Defeating Objects***
  - Objects "too powerful" to exist.
- ***Undecidable Problems***
  - Problems truly beyond the limits of algorithmic problem-solving!
- ***Consequences of Undecidability***
  - Why does any of this matter outside of Theoryland?