Turing Machines

Part Three

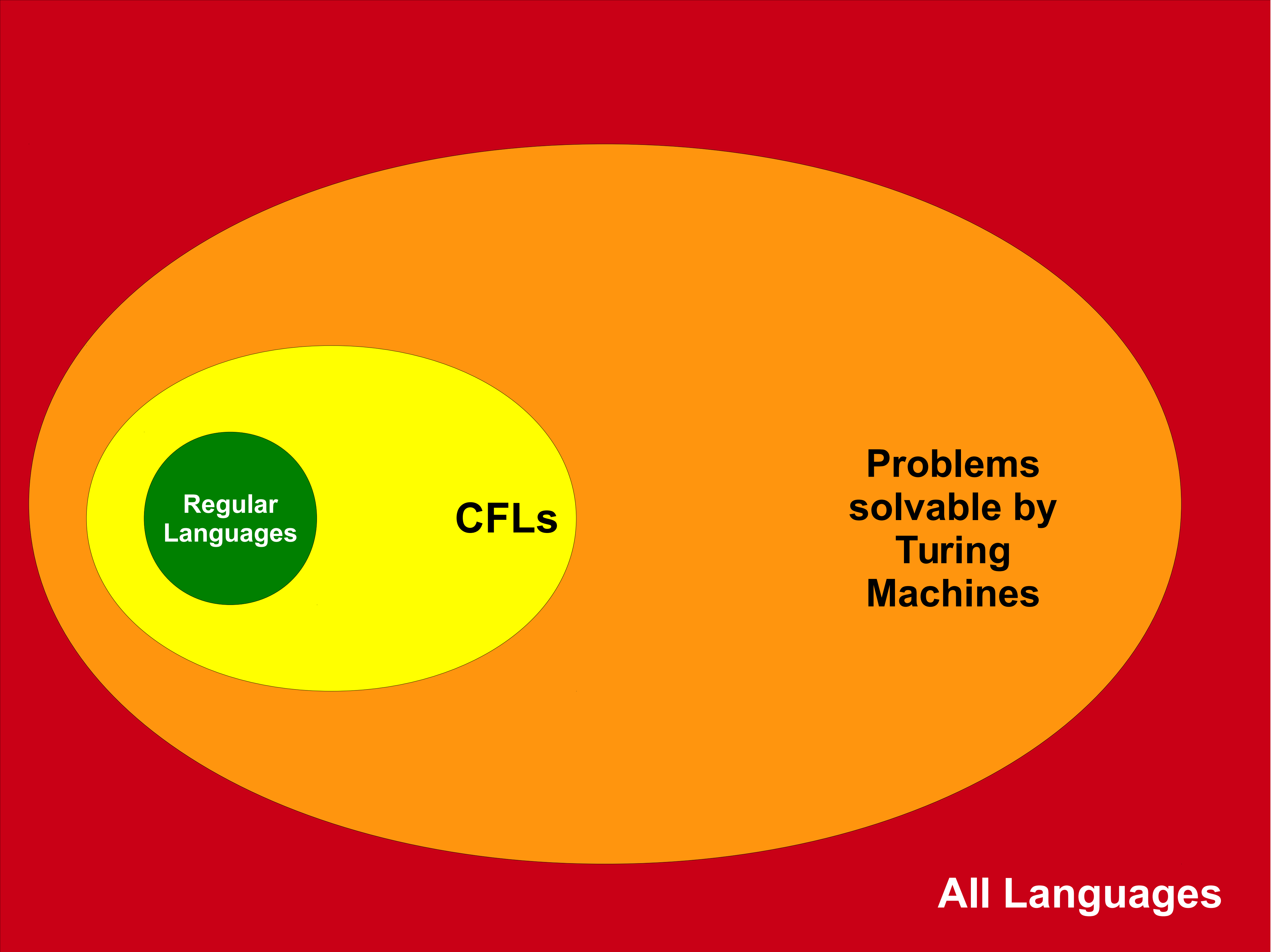
Last Time: ***How powerful are Turing machines?***

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 $\boxed{\text{computer}}$?

What kind of computer?

What problems can we solve with a computer?

What is a "problem?"

Languages and Problems

- We've been using formal languages as a way of modeling computational problems.
- However, the problems we encounter in The Real World don't look at all like language problems.
- Is this all theoretical nonsense? Or is there a reason for this?

“In theory, there's no difference between theory and practice. In practice, there is.”

Decision Problems

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

  Given x, y, and z, is $x+y=z$?

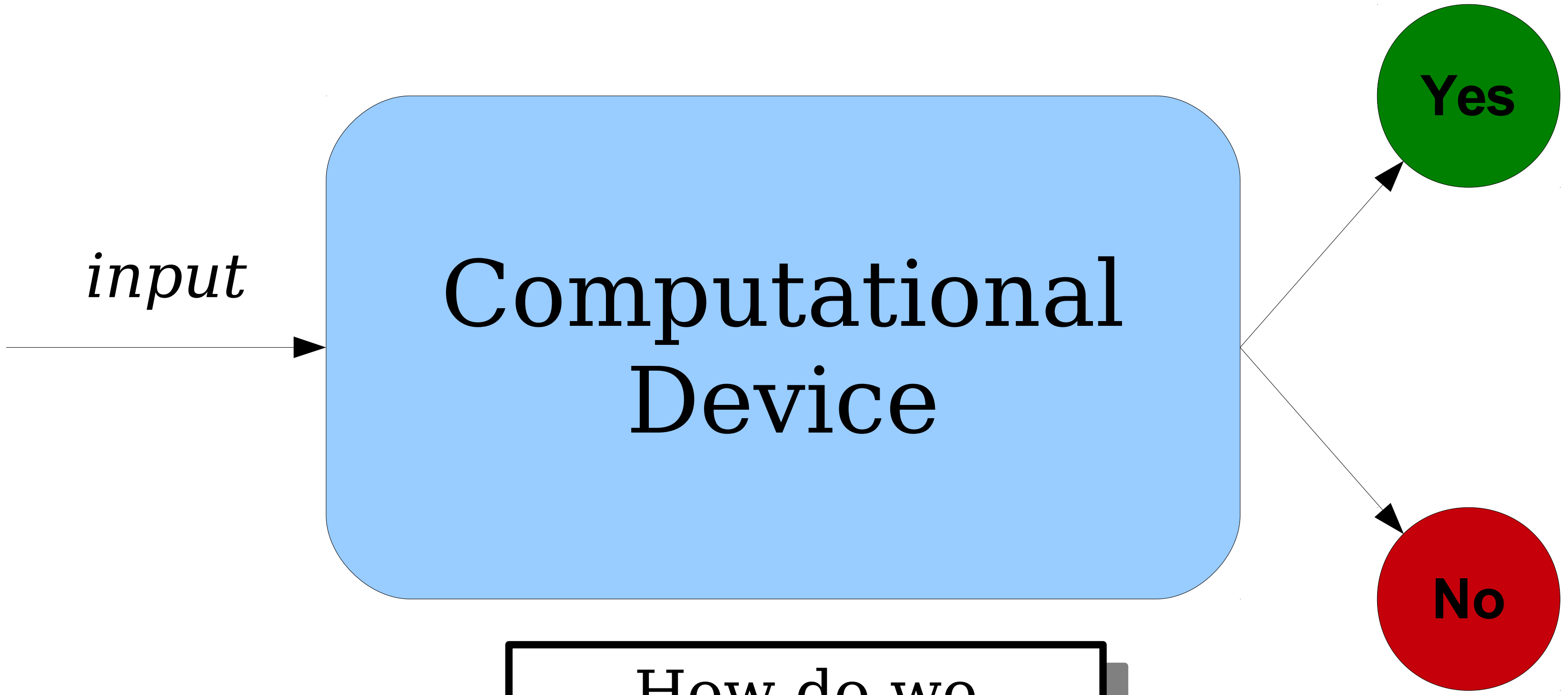
- Example: detecting relationships.

  Given a family tree T and people x and y, is x a grandparent of y?

- Example: avoiding traffic.

  Given a transportation grid G annotated with traffic information, a start location s, a destination d, and a time limit t, is there a way to get from s to d within time t?

Solving Decision Problems



How do we represent our inputs?

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.

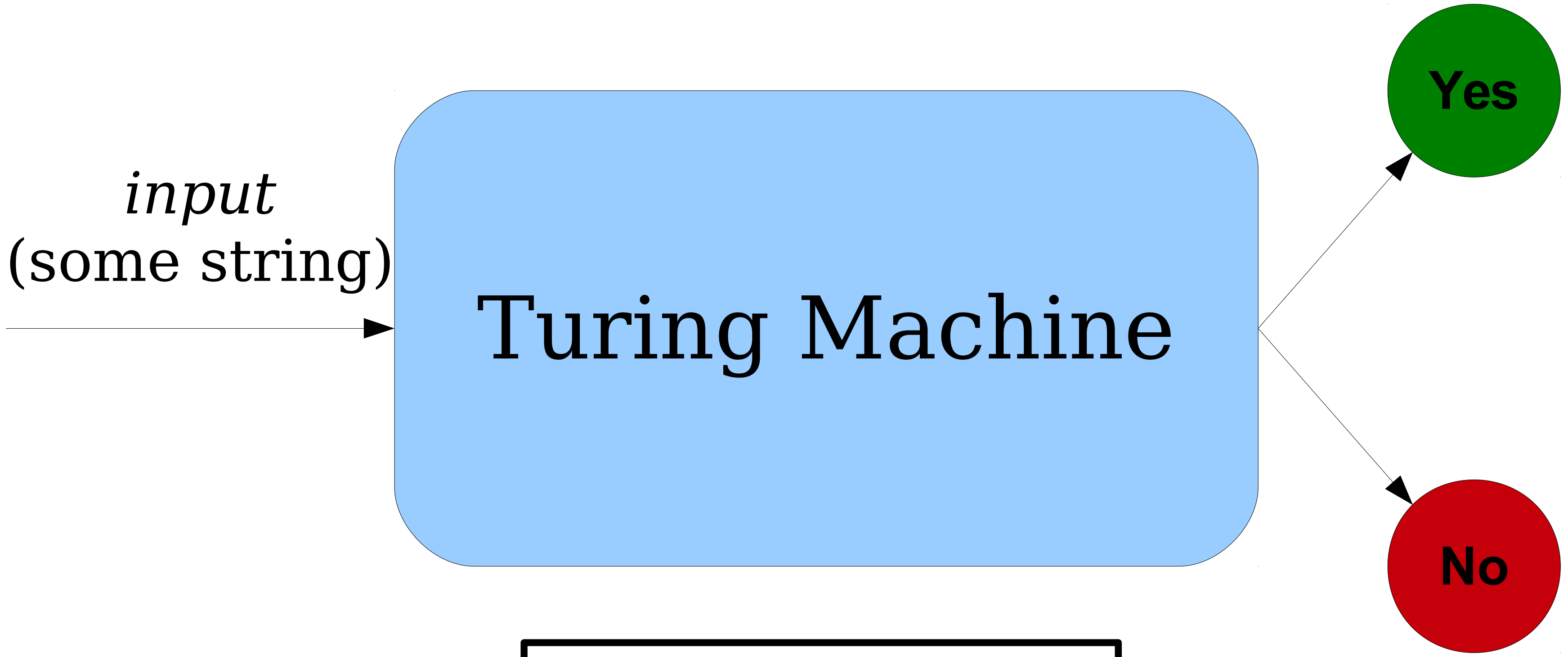
Strings and Objects

- Let *Obj* be some discrete, finite object (a string, a video, an image, a text file, etc.)
- Let Σ be some alphabet.
- We'll represent an ***encoding of Obj*** using the characters in Σ by writing **⟨*Obj*⟩**. Think of ⟨*Obj*⟩ like a file on disk – it encodes complex data as a series of characters.
- 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



How do we specify the behavior we want?

Specifying a Decision Problem

- Consider this decision problem:

Given x, y, $z \in \mathbb{N}$, determine whether $x+y=z$.

- With our computational model, we'll feed some string into a TM, and it then might come back with an answer (yes or no).
- Some strings are accepted, some are rejected, and some cause the machine to loop infinitely.

Specifying a Decision Problem

- Consider this decision problem:

Given $x, y, z \in \mathbb{N}$, determine whether $x+y=z$.

- If we give the input as $\langle x, y, z\rangle$, the set of strings the TM should say YES to is

$$\{ \langle x, y, z\rangle \mid x, y, z \in \mathbb{N} \text{ and } x + y = z \}$$

- Notice that this is a language – it's a set of strings!

Specifying a Decision Problem

- Consider this decision problem:

  Given a graph *G*, determine whether *G* is a bipartite graph.

- With our computational model, we'll feed some string into a TM, and it then might come back with an answer (yes or no).
- Some strings are accepted, some are rejected, and some cause the machine to loop infinitely.

Specifying a Decision Problem

- Consider this decision problem:

Given a graph *G*, determine whether *G* is a bipartite graph.

- If we give the input as $\langle G \rangle$, the set of strings the TM should say YES to is

$$\{ \langle G \rangle \mid G \text{ is a bipartite graph } \}$$

- Notice that this is a language – it's a set of strings!

Problems and Languages

- ***Key intuition:*** Every language corresponds to some decision problem.
- Example:
  - $\{ \langle x, y \rangle \mid x, y \in \mathbb{N}$ and $x \equiv_3 y \}$ is a language.
  - It corresponds to the following decision problem:

Given $x, y \in \mathbb{N}$, do x and y leave the same remainder when divided by 3?

Problems and Languages

- ***Key intuition:*** Every language corresponds to some decision problem.
- Example:
  - $\{ \langle D \rangle \mid D$ is a DFA that accepts ε $\}$ is a language.
  - It corresponds to the following decision problem:

Given a DFA *D*, does *D* accept ε?

Problems and Languages

- ***Key intuition:*** Every language corresponds to some decision problem.
- Example:
  - $\{ \langle G \rangle \mid G$ is a planar graph $\}$ is a language.
  - It corresponds to the following decision problem:

Given a graph *G*, is *G* planar?

What All This Means

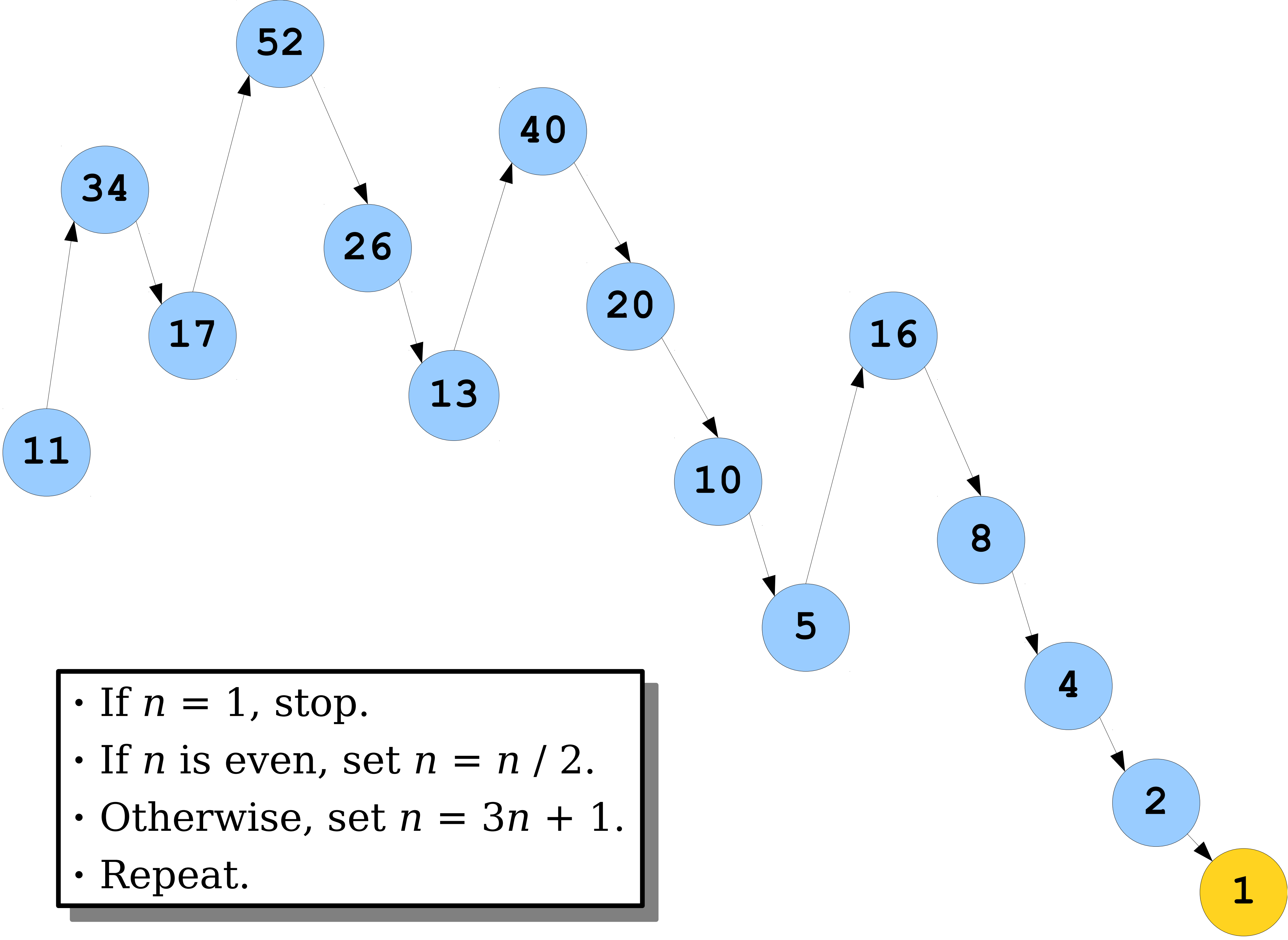
- Our goal is to speak of *computers solving problems*.
- We will model this by looking at *TMs recognizing languages*.
- For *decision problems* that we're interested in solving, this precisely captures what we're interested in capturing.

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.

Does this Turing machine always accept?

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

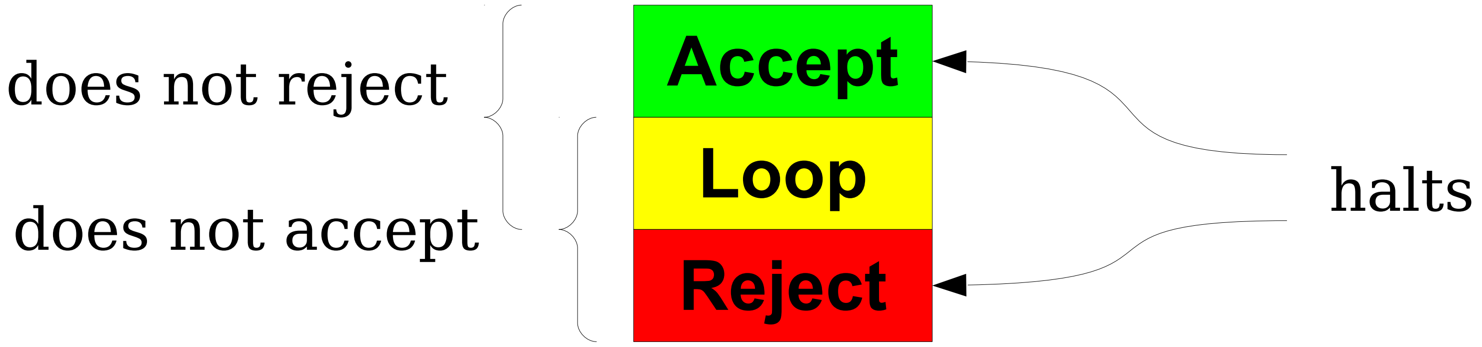
- Two years ago, some Apple employees filed a patent for a cryptographic hashing scheme based on the Collatz conjecture; see ***this link*** for details.

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 therefore 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:

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

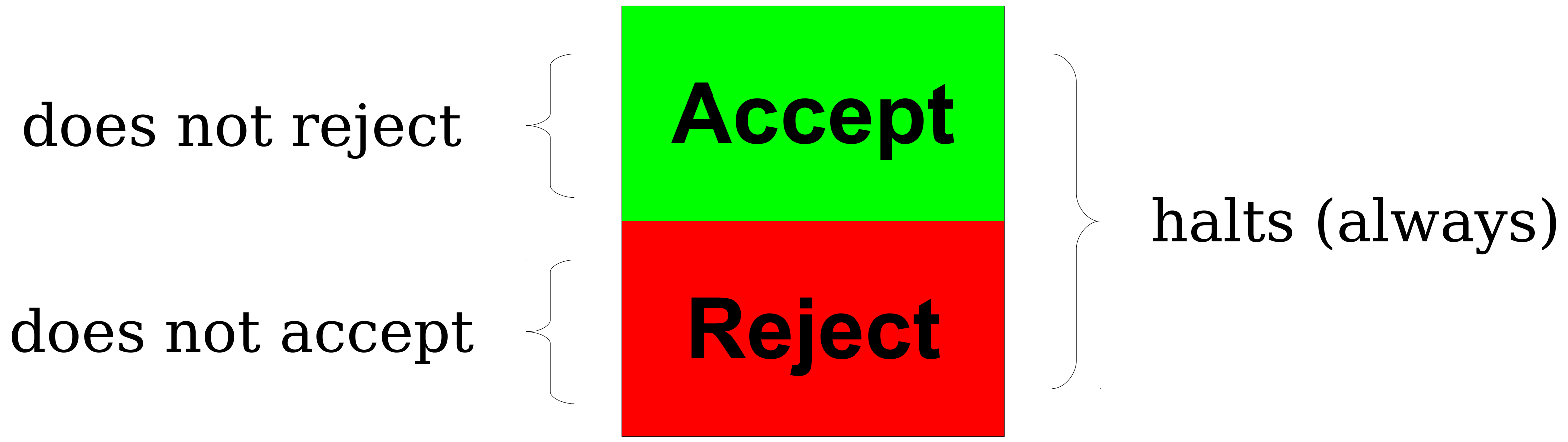
- For any $w \in \mathscr{L}(M)$, M accepts w.
- For any $w \notin \mathscr{L}(M)$, M does not accept w.
  - It might loop forever, or it might explicitly reject.
- A language is called ***recognizable*** if it is the language of some TM. A TM for a language is sometimes called a ***recognizer*** for that language.
- 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.
- $\{\ \mathtt{0}^n\mathtt{1}^n \mid n \in \mathbb{N}\ \}$ 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.***

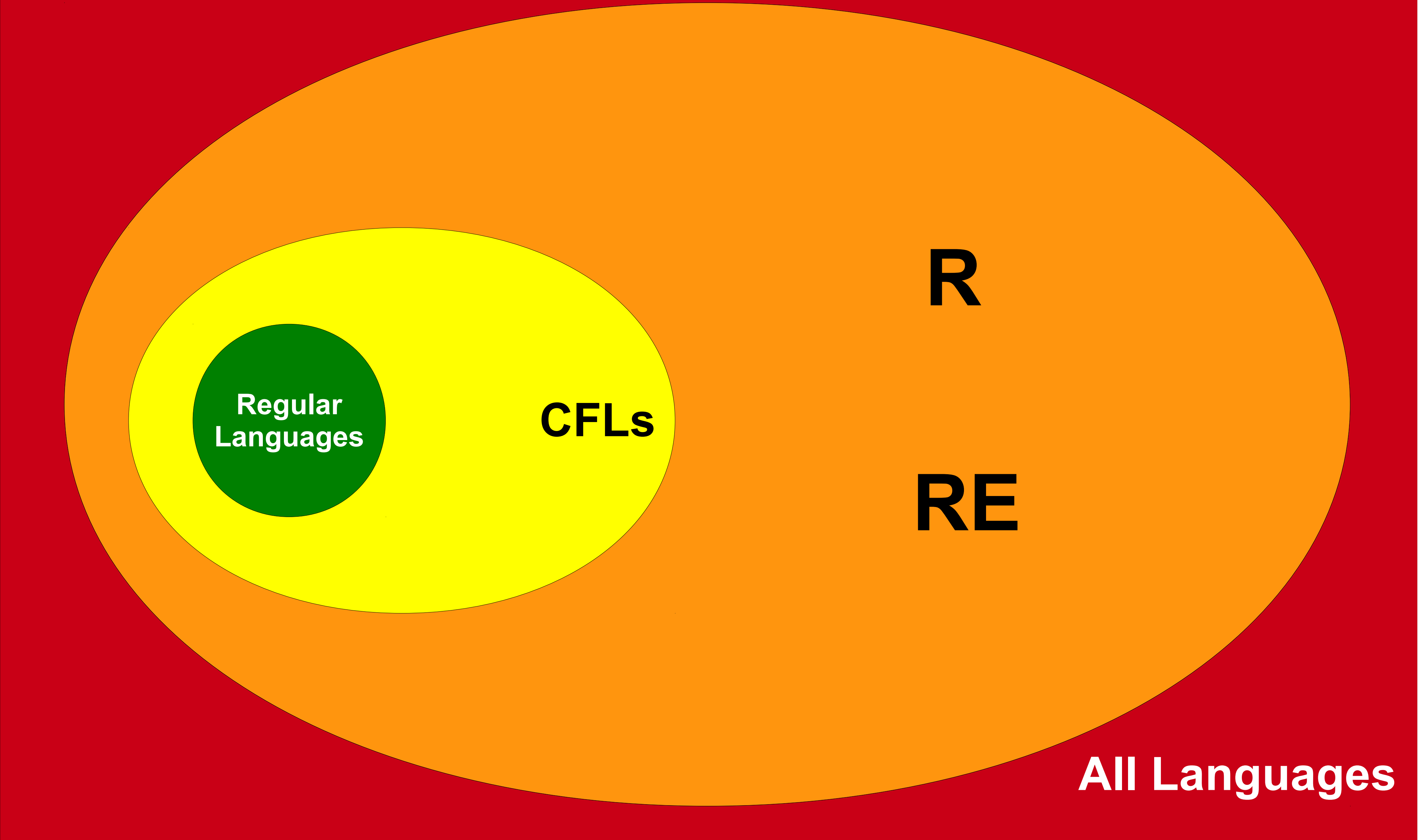
R and **RE** Languages

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

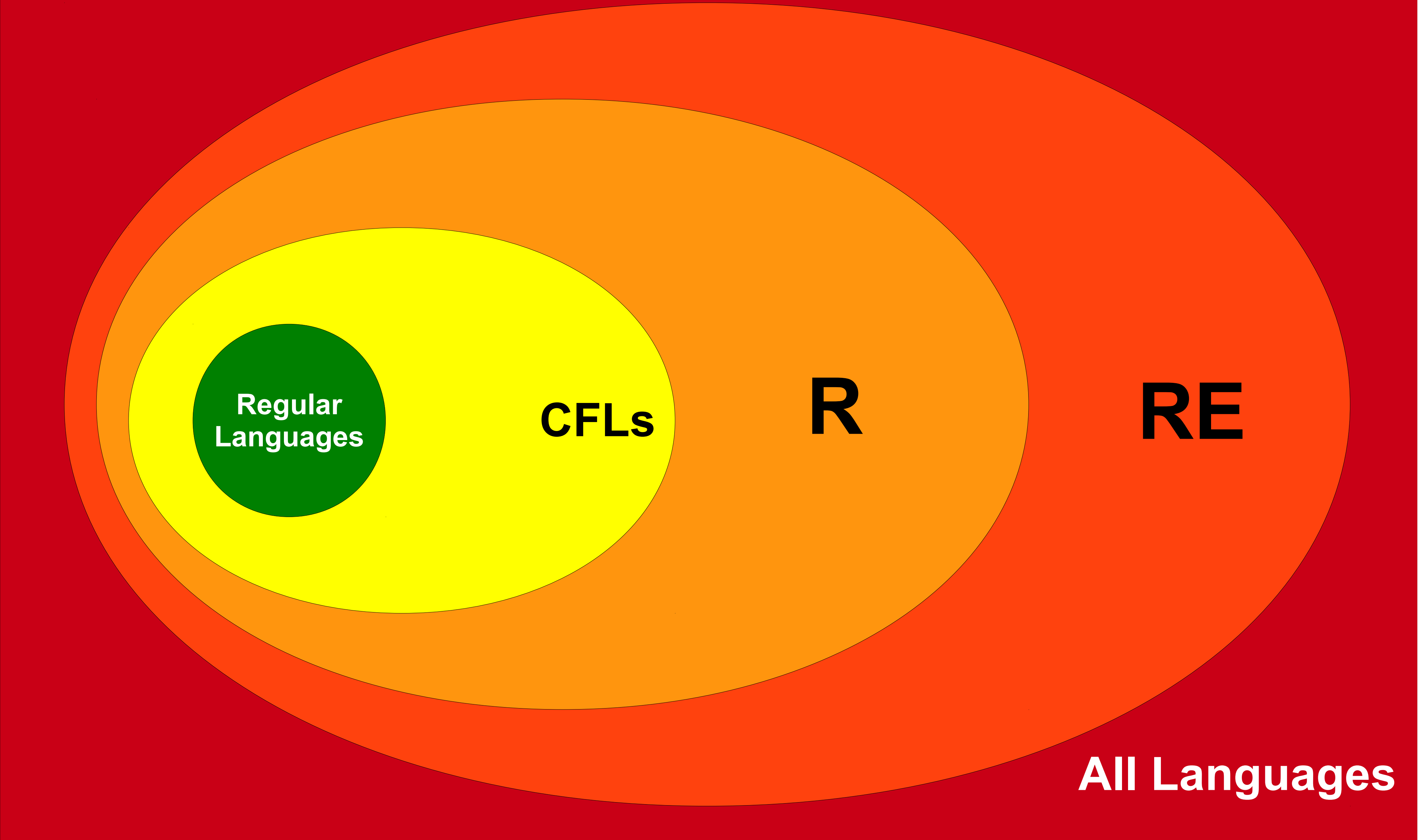
$$\mathbf{R} \overset{?}{=} \mathbf{RE}$$

- 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!

Problem Sets

- Problem Set Seven was due at the start of class today.
  - Want to use late days? Submit by Monday at 3:00PM.
  - Note that late days can't be used on PS9.
- Problem Set Eight goes out today. It's due next Friday at 3:00PM.
  - Play around with TMs and their properties!
  - Explore the limits of the **R** and **RE** languages!
  - *Will require some material from Monday; those sections are clearly marked.*

Second Midterm Exam

- The second midterm exam is next **Monday, February 29** from 7PM – 10PM.
- Topic coverage:
  - Focus is on PS4 – PS6 and lectures 09 – 16.
  - Topics from PS7 and from lecture 17 onward not tested.
  - ***Major topics:*** strict orders, graphs, the pigeonhole principle, induction, finite automata, regular expressions, regular languages, closure properties.
- Policies and procedures same as the first midterm:
  - Three hours, four questions.
  - Closed-computer, closed-book, and limited-note. You can have a double-sided 8.5” × 11” sheet of paper with you when you take the exam.

Midterm Locations

- The midterm is in Hewlett.
- Specifically, locations are divvied up by last (family) name:
  - `Abd` – `Pre`: Go to Hewlett 200.
  - `Pri` – `Vil`: Go to Hewlett 201.
  - `Vo` – `Xie`: Go to Hewlett 101.
  - `Yan` – `Zhu`: Go to Hewlett 103.

Preparing for the Exam

- As a reminder, we've posted
  - four sets of extra practice problems,
  - two practice midterms,
  - one set of challenge problems, and
  - one set of CS103A problems.
- Solutions are available in Gates. As with the previous midterm, we'll move the solution sets down to the basement over the weekend.
- ***Recommendation:*** If you haven't already done so, take at least one of the practice exams under realistic conditions and get a TA to look over it.
- ***Ask questions!*** If you aren't 100% sure you understand something, ask for help and advice. We want you to master this material. Let us know what we can do to help.

Your Questions

“I'm worried because I did really badly on Pset 6, mainly because I missed several edge cases in designing DFAs / NFAs / regexes. Do you have any advice for how to go about finding all the edge cases, particularly in a time-pressured scenario like the midterm?”

For DFAs, ask this question: what does every state “mean?” What “information” does it represent? If you can't answer this question, you may have an error in the machine.

For NFAs, work backwards from the accepting states. Remember that NFAs will try as hard as possible to accept, so make sure that you didn't accidentally put in a path that accepts incorrectly.

For regular expressions, try extreme cases. What if you expand out stars and question marks zero times? What do you get? Also, make sure you have a concise explanation for what you wrote. If you just listed a bunch of cases without any rhyme or reason, nine times out of ten you've got it wrong.

“I've always wondered--how do you explain things so clearly while simultaneously talking at 1.5x a normal person's speed? Do you rehearse your talk track a ton before lecture, or does your brain just work at super-speed?”

I come from a family of fast talkers. Our family dinners are really entertaining. ☺

For the very first class I ever taught (back in 2007) I was so terrified of messing up in front of a group of people that I memorized literally everything I was going to say. After doing that for enough repetitions, I got a lot more comfortable just making stuff up on the fly.

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.

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?

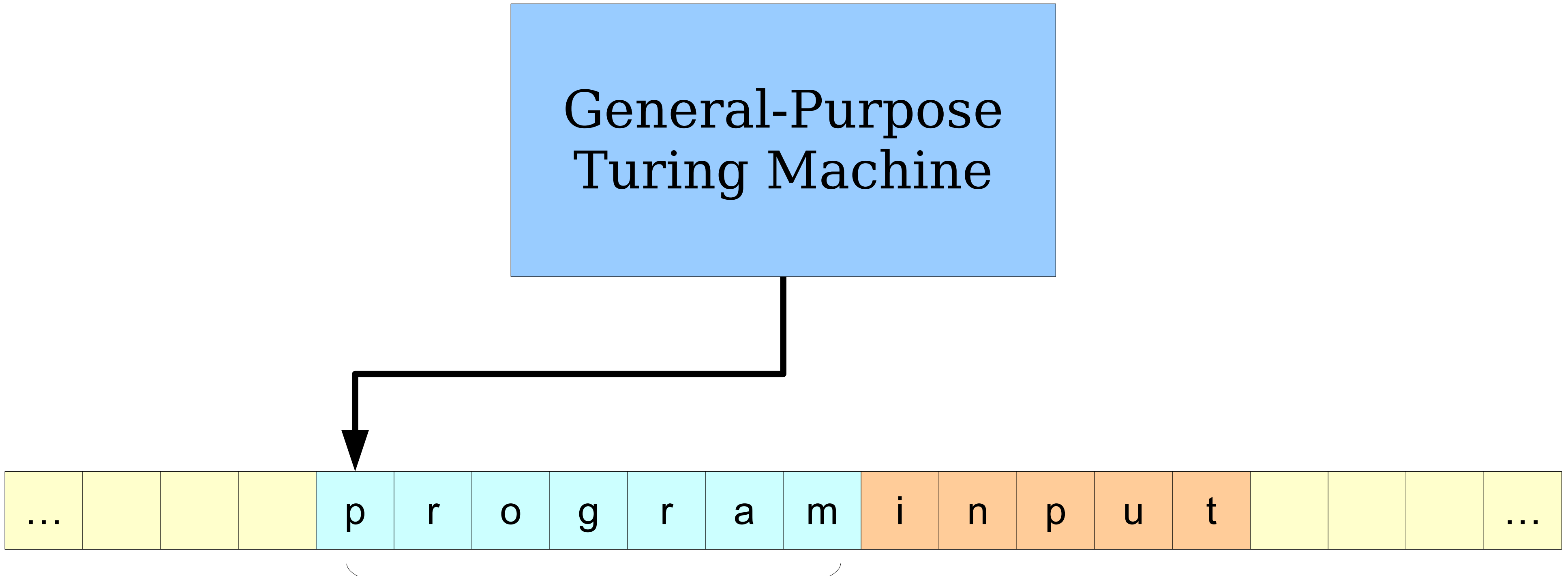
Encodings and TMs

- ***Recall:*** If *Obj* is some finite, discrete object, then let ⟨*Obj*⟩ denote a string representation of that object.
- As a specific case: if M is a TM, then $\langle M \rangle$ is a string representing a string encoding of M.
  - A helpful analogy: think of M as an executable file and $\langle M \rangle$ as its source code.
- Because TMs can be encoded as strings, *it is possible to feed a TM as an input to another TM!*

A Single Turing Machine?

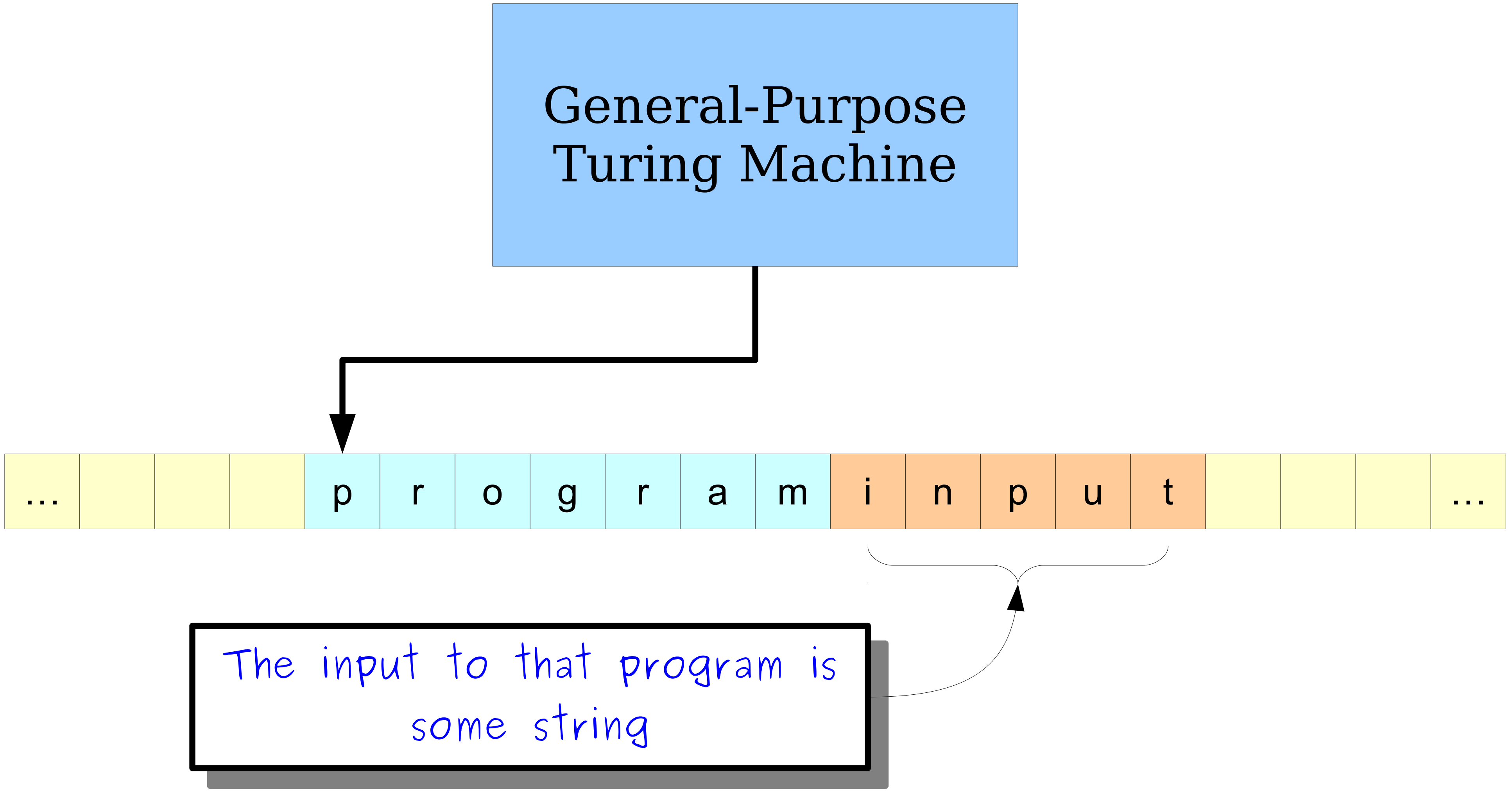
- Just as we can build a single computer that can run any program, could we build a single TM that could run any Turing machine?
- ***Idea:*** Build a machine that takes as input a description of a TM and a string to run that TM on, then simulates the behavior of that TM on that string.

A Universal Machine

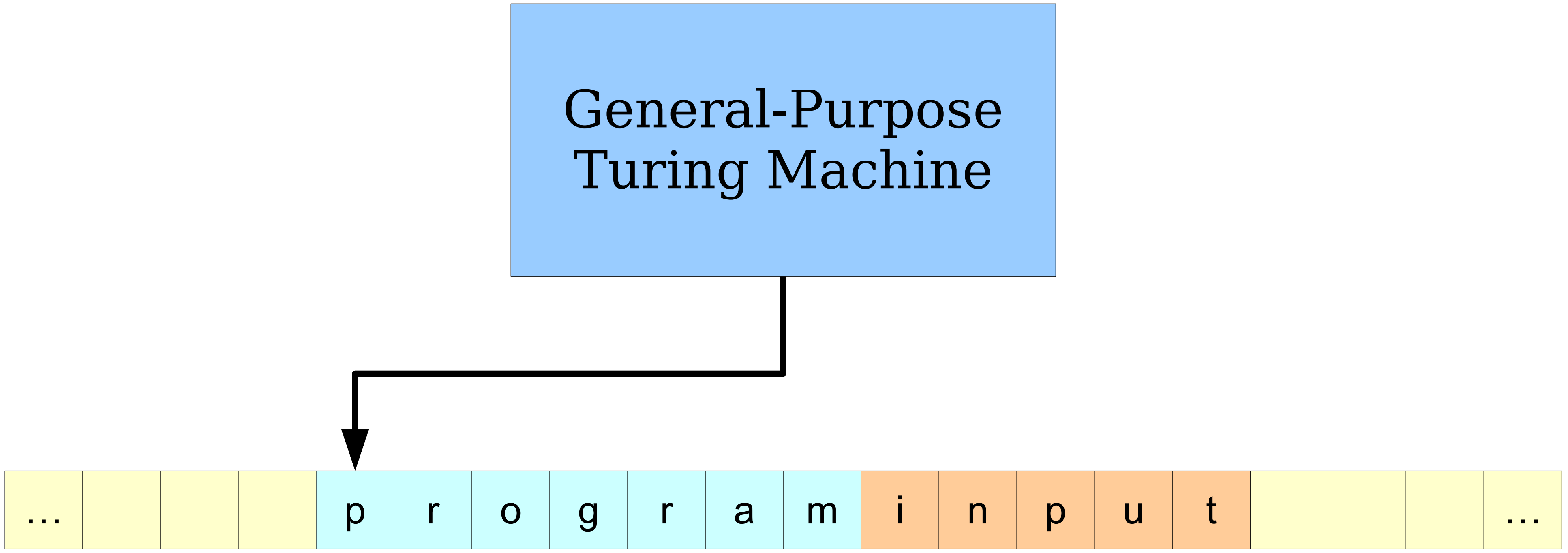


The "program" is an encoding of some Turing machine M that we want to run.

A Universal Machine

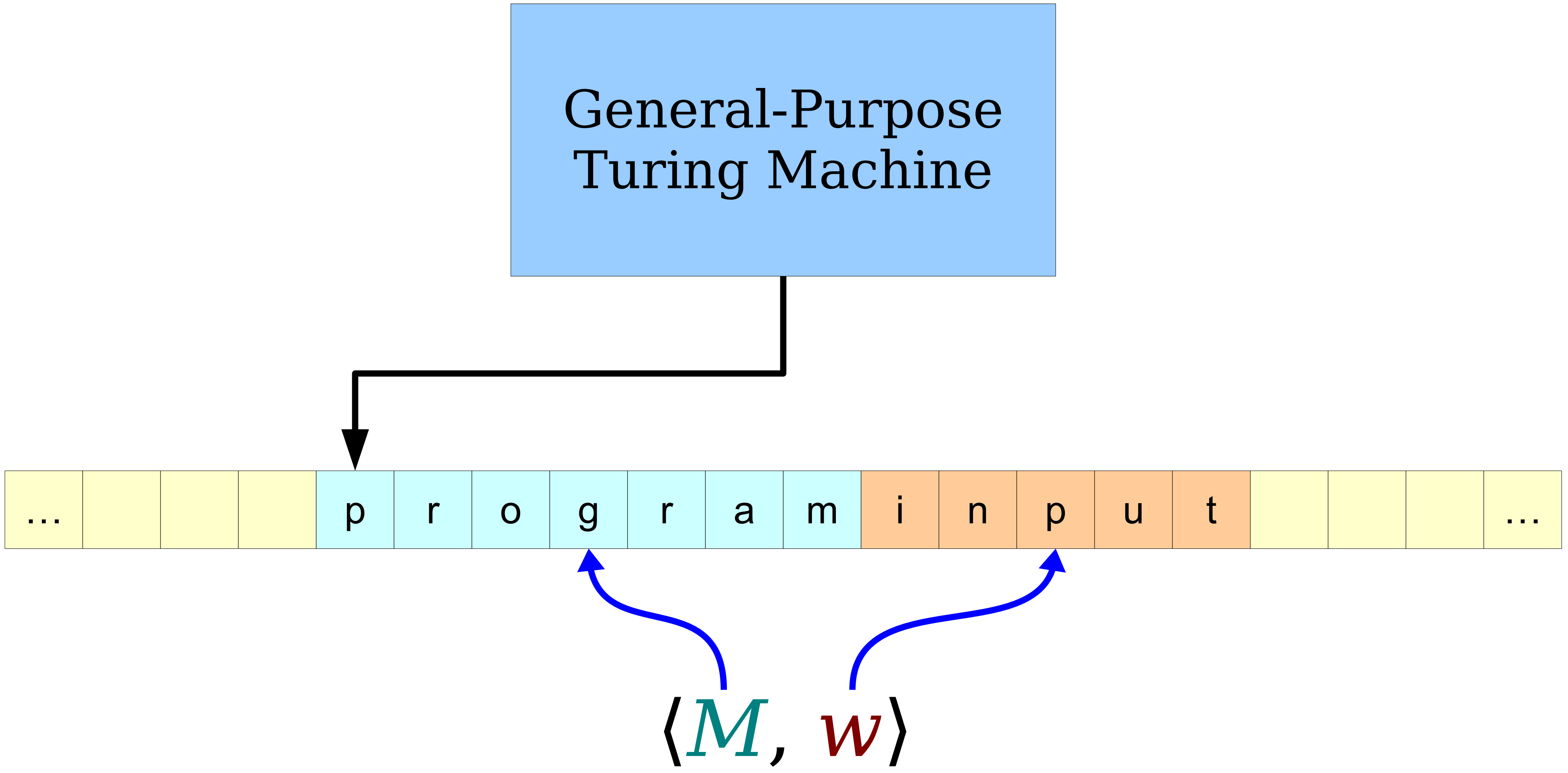

General-Purpose Turing Machine
…
p
r
o
g
r
a
m
i
n
p
u
t
…
The input to that program is some string

A Universal Machine



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

A Universal Machine



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} accepts $\langle M, w\rangle$ if and only if M accepts w.**

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).
- Conceptually:

```
bool simulateTM(TM M, string w) {
  set up a simulation of M running on w;
  while (true) {
    if (the simulated version of M is in an accepting state)
        return true;  // Accept
    if (the simulated version of M is in a rejecting state)
        return false; // Reject
    simulate one more step of M running on 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!***

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

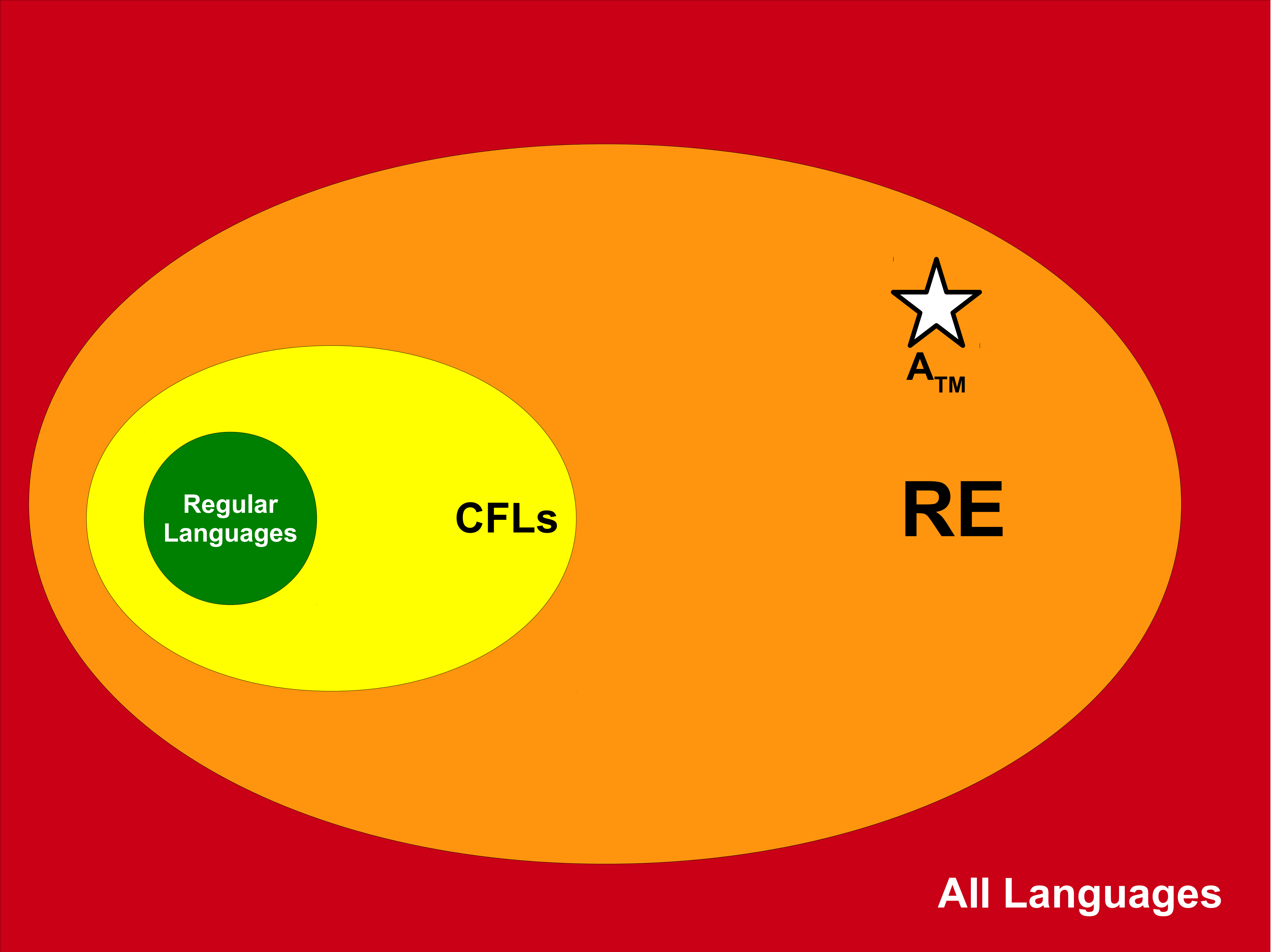
$$\mathscr{L}(M) = \{ w \in \Sigma^* \mid M \text{ accepts } w \}$$

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


Regular Languages
CFLs
A_{TM}
RE
All Languages