Turing Machines

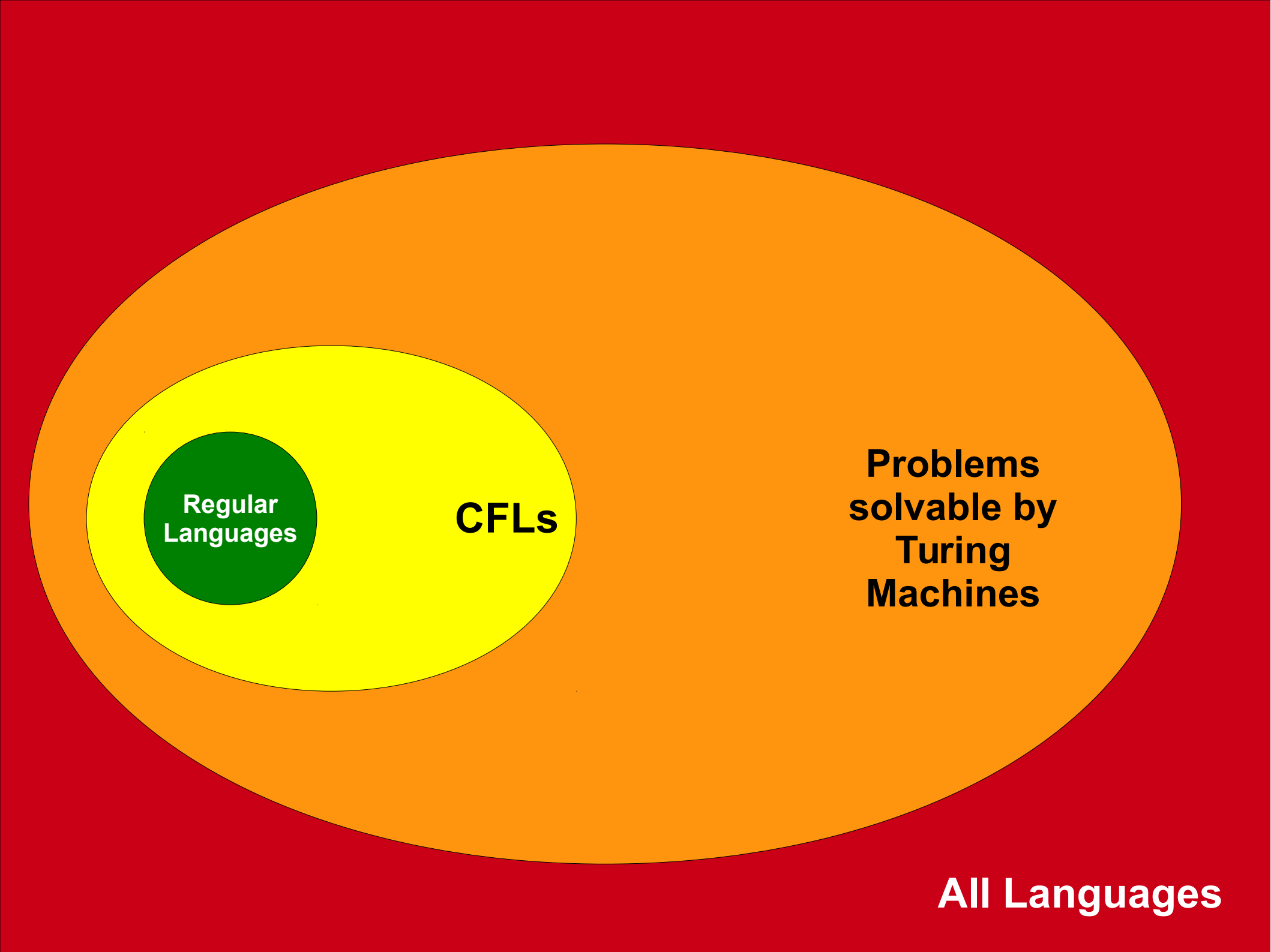
Part Three

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

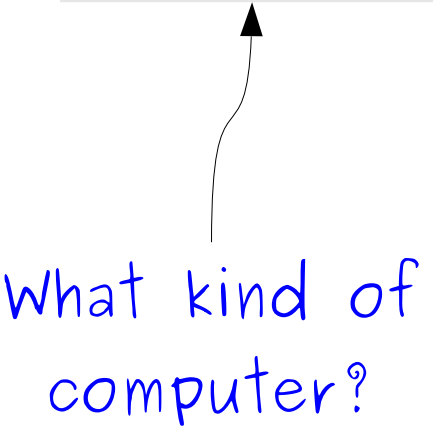
The ***Church-Turing Thesis*** claims that

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

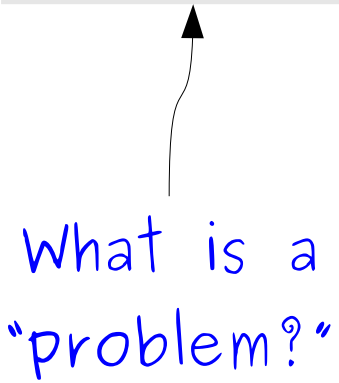
> “This is not a theorem – it is a falsifiable scientific hypothesis. And it has been thoroughly tested!”
>
> - Ryan Williams


Regular Languages
CFLs
Problems solvable by Turing Machines
All Languages

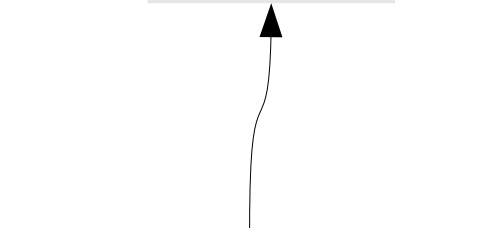
What problems can we solve with a computer?



What problems can we solve with a computer?

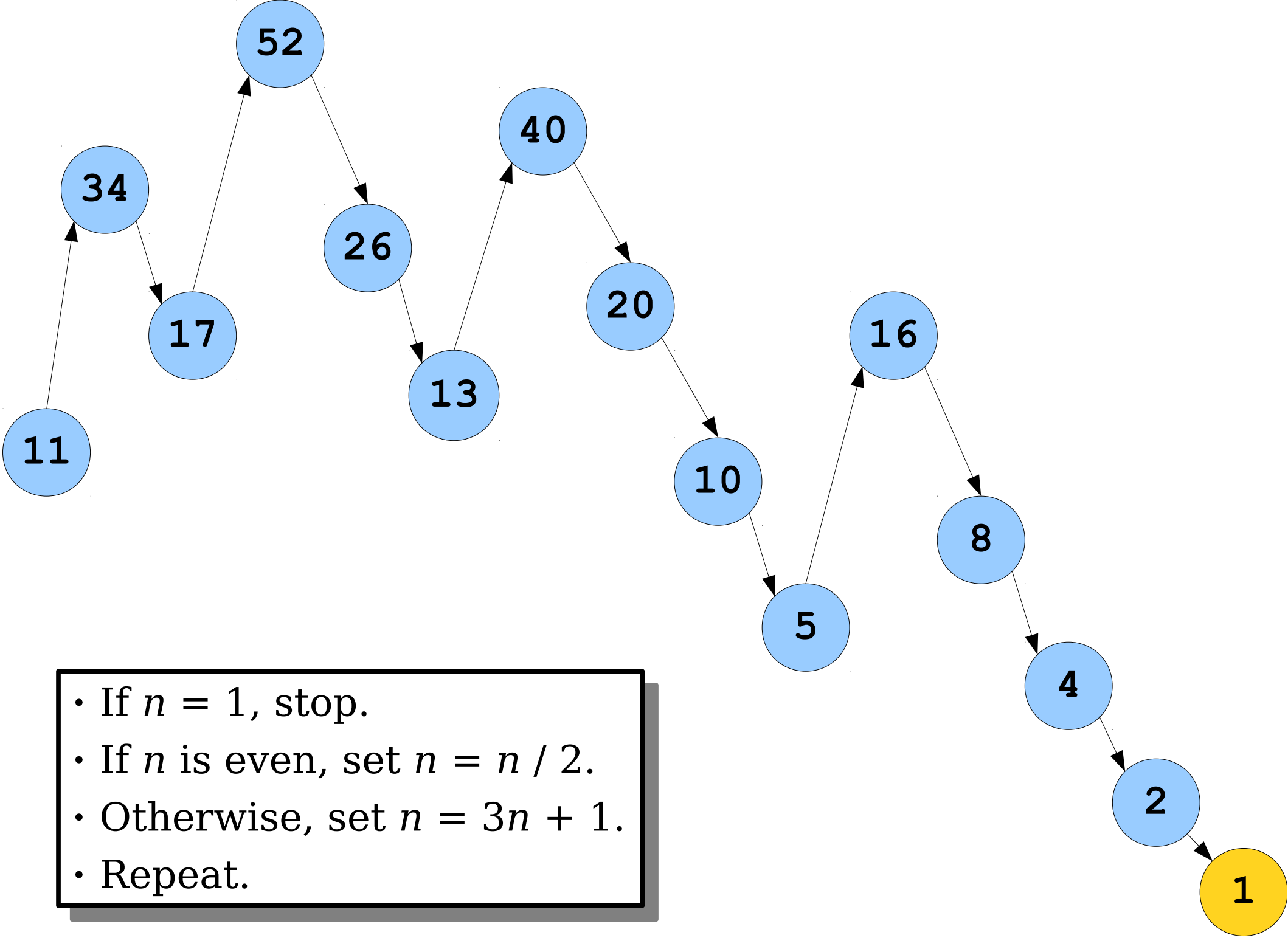


What problems can we solve with a computer?



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 $\Sigma = \{\texttt{1}\}$ and consider the language

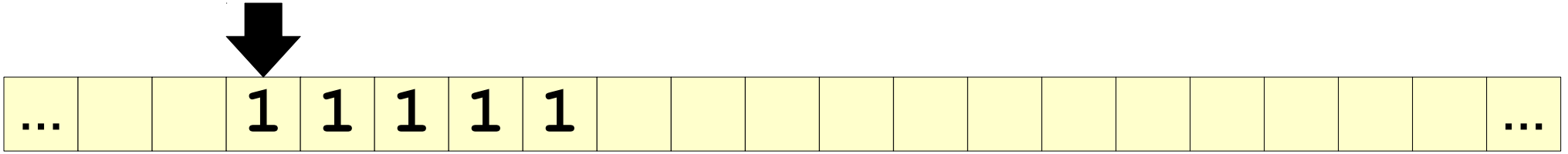
  $L = \{ \texttt{1}^n \mid n > 0$ and the hailstone sequence terminates for $n \}$.

- 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.

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

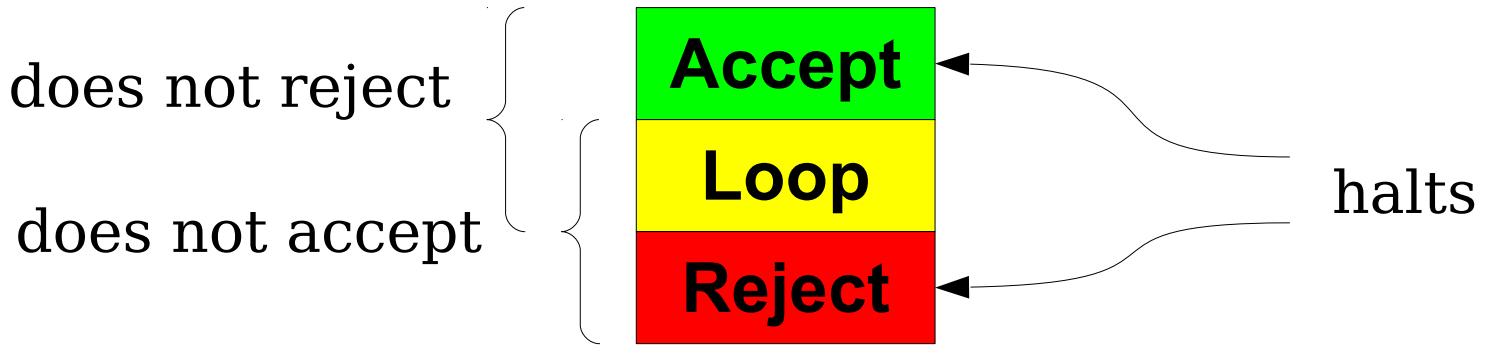
- Two years ago, some Apple employees filed a patent for a cryptographic hashing scheme based on the Collatz conjecture; see ***this link*** for details.

An Important Observation

- Unlike the other automata we've seen so far, Turing machines choose for themselves whether to accept or reject.
- 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:

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

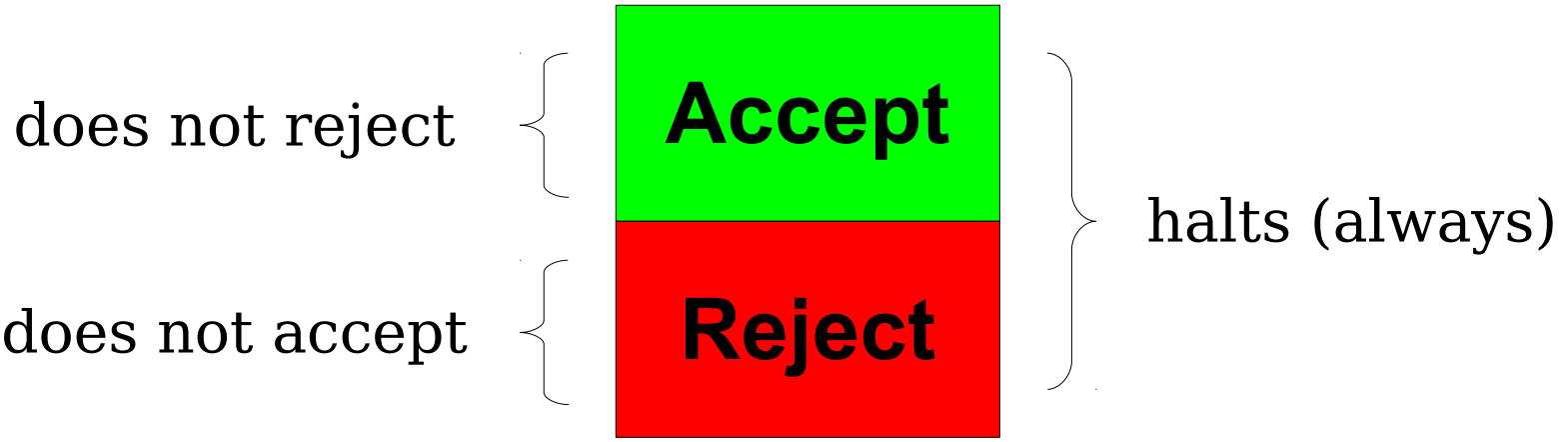
- For any $w \in \mathscr{L}(M)$, M accepts w.
- For any $w \notin \mathscr{L}(M)$, M does not accept w.
  - It might loop forever, or it might explicitly reject.
- A language is called ***recognizable*** if it is the language of some TM. A TM for a language is sometimes called a ***recognizer*** for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

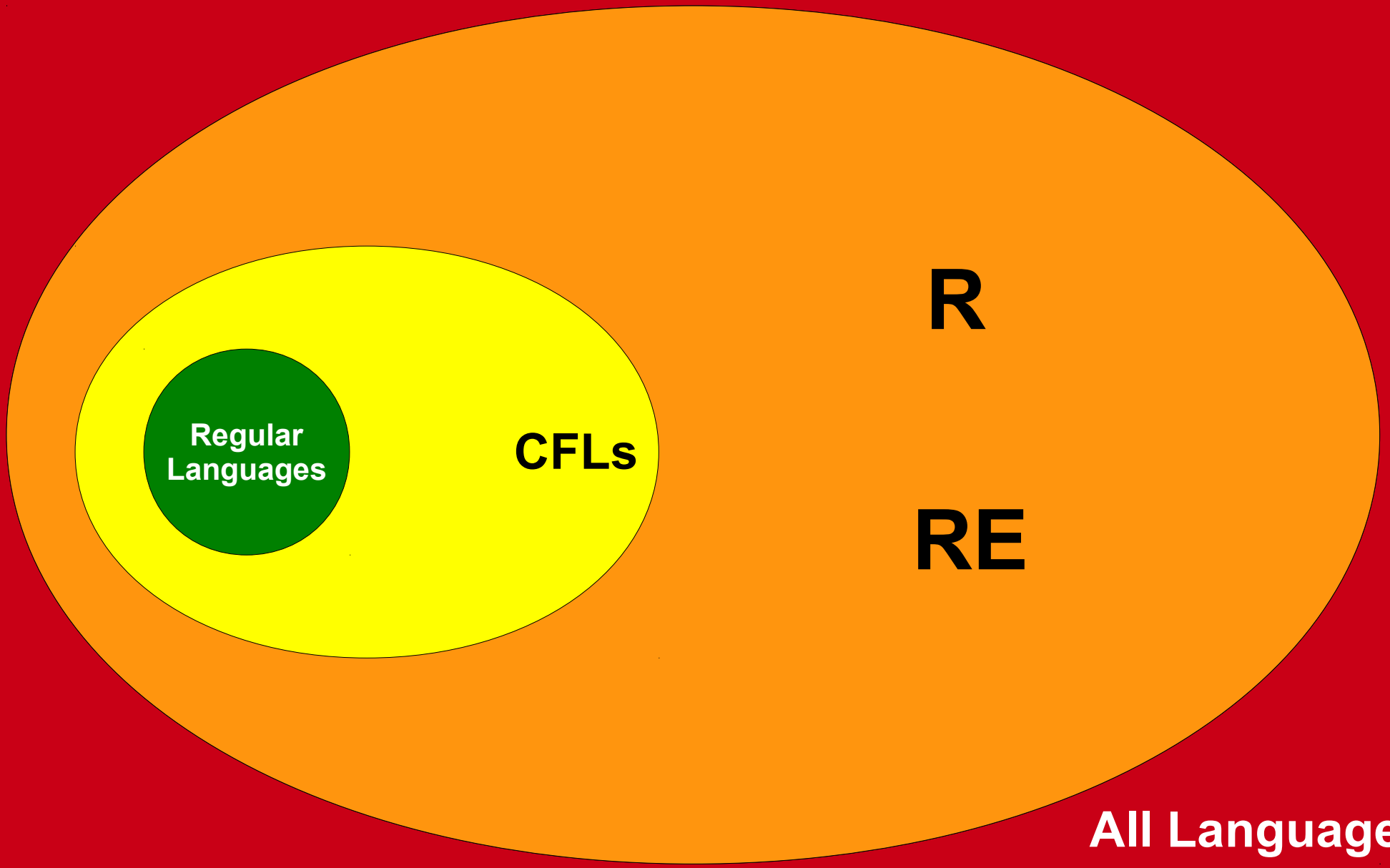
R and **RE** Languages

- Every decider is a Turing machine, but not every Turing machine is a decider.
- Thus $\mathbf{R} \subseteq \mathbf{RE}$.
- Hugely important theoretical question:

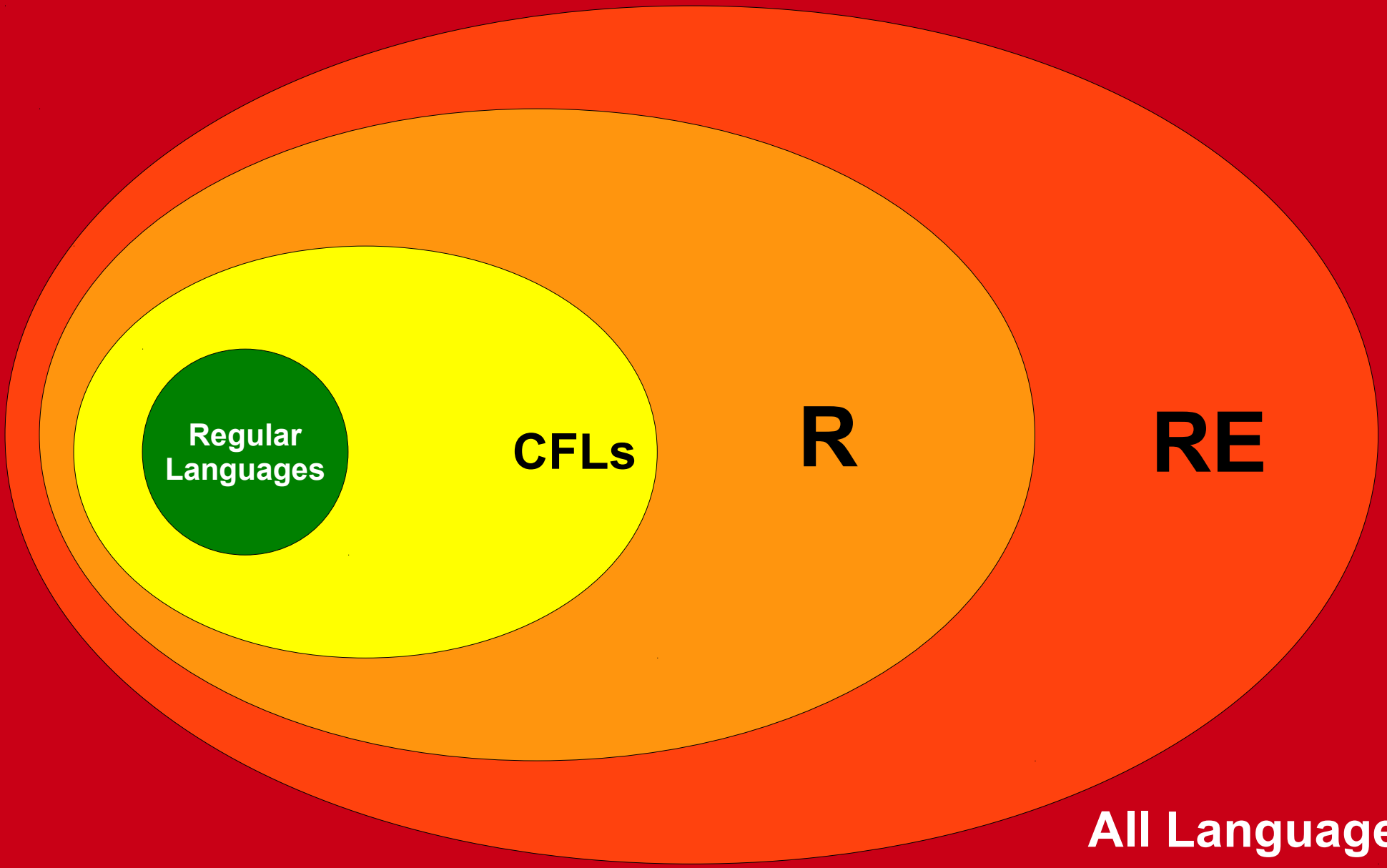
$$\mathbf{R} \stackrel{?}{=} \mathbf{RE}$$

- 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 $\mathbf{R} \stackrel{?}{=} \mathbf{RE}$ question mean?
- Is $\mathbf{R} = \mathbf{RE}$?
- What lies beyond **R** and **RE**?
- We'll see the answers to each of these in due time.

Time-Out for Announcements!

Second Midterm Exam

- The second midterm exam is next **Monday, November 16** from 7PM – 10PM.
- Room assignments divvied up by last (family) name:
  - **`Aga – Ven`**: Go to Hewlett 200.
  - **`Ver – Zhe`**: Go to 370-370
- Cumulative exam with a focus on material from Lectures 10 – 18 and PS4 – 6.
  - Main focus is on graphs, the pigeonhole principle, induction, finite automata, regular expressions, and regular languages.
  - Will *not* test the Myhill-Nerode theorem, CFGs, or Turing machines.
  - Since everything builds on itself in this course, the exam may require topics from earlier in the course.
- As before, you get one double-sided, 8.5” × 11” sheet of notes during the exam.

Exam Practice

- We have now released
  - three sets of extra practice problems, with solutions;
  - a practice exam, with solutions; and
  - a second set of challenge problems, without solutions.
- Please feel free to ask questions on Piazza or to stop by our weekend office hours with questions. We'd be happy to help out!

Your Questions!

“Can you share interesting, little-known fact or story / about Alan Turing?”

I visited Bletchley Park – where Turing and others worked breaking German codes – and they had a whole exhibit about Turing! A few highlights…


Successful break at BP of a wartime Enigma cypher message.
that can be exploited.
His idea is tried out but at this time produces no useful results.
somewhat disappointing and it is of only limited operational value.
MOST SECRET.
Alan Turing

Director:
Professor of Computing
and Data Processing:
E. S. PAGE, M.A., Ph.D.

Professor of Computing Science:
B. RANDELL, B.Sc., A.R.C.S.

Telephone Newcastle (0632) 29233

N.U.M.A.C.
Computer Manager:
Miss E. D. BARRACLOUGH, M.Sc.

Ref: 7/1/2(S)

27th November, 1975

Dear Mrs. Turing,

I thought you would like to know that the Government have recently made an official release of information which contains an explicit recognition of the importance of your son's work to the development of the modern computer. They have admitted that there was a special purpose electronic computer developed for the Department of Communications at the Foreign Office in 1943. Their information release states that Charles Babbage's work in the 19th century and your son's work in the 1930s laid the theoretical foundations for the modern computer and that the Foreign Office's special purpose computer was, to their belief, the first such electronic computer to be constructed anywhere. Moreover, they credited your son's work with having had a considerable influence on the design of this machine.

Further, a book has recently been published in the United States entitled 'Bodyguard of Lies' which describes the work of the Allies during the war to deceive the Germans and to prevent them from anticipating the D-day invasion. This book credits your son as being the main person involved in the breaking of one of the most important German codes, the Enigma Code and thus implies that his work was of vital importance to the outcome of World War II. This latter information in the book 'Bodyguard of Lies' is not, of course, official information but nevertheless it will, I believe, enable many people to obtain a yet fuller understanding of your son's genius. I am very pleased that this is now happening at last.

Yours sincerely,

Brian Randell

Mrs. E.S. Turing,
Stonycrest Nursing Home,
Court Road,
Hindhead,
Surrey.

BR/EMS

CLAREMONT TOWER, CLAREMONT ROAD
NEWCASTLE UPON TYNE NE1 7RU

“Which other faculty members inspired you as an undergrad? ☺”

Julie Zelenski really helped and supported me as I was getting started in CS and I really credit her with helping me end up where I am now. Tim Roughgarden and Serge Plotkin made me fall in love with algorithms and data structures, and John Mitchell and Dave Dill totally sold me on CS theory.

I also need to give a big shoutout to Nirvana Tanoukhi, my TF in my IHUM course, for making me realize what writing and literature are really about, and to Shay Brawn, my freshman PWR instructor, for teaching me how to write and structure a nuanced argument.

“What should I do if I want to do research in a certain field, but didn't even do above the median on the exam for the class? Should I move to a different field, or is there hope?”

Classwork and research are completely different beasts. To do research, you need to be a self-starter and have to have the discipline to focus on a topic and really learn it. I definitely wouldn't let your performance in a single class discourage you from doing research - keep taking classes in the area, see if you like it, and keep reaching out to professors!

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?

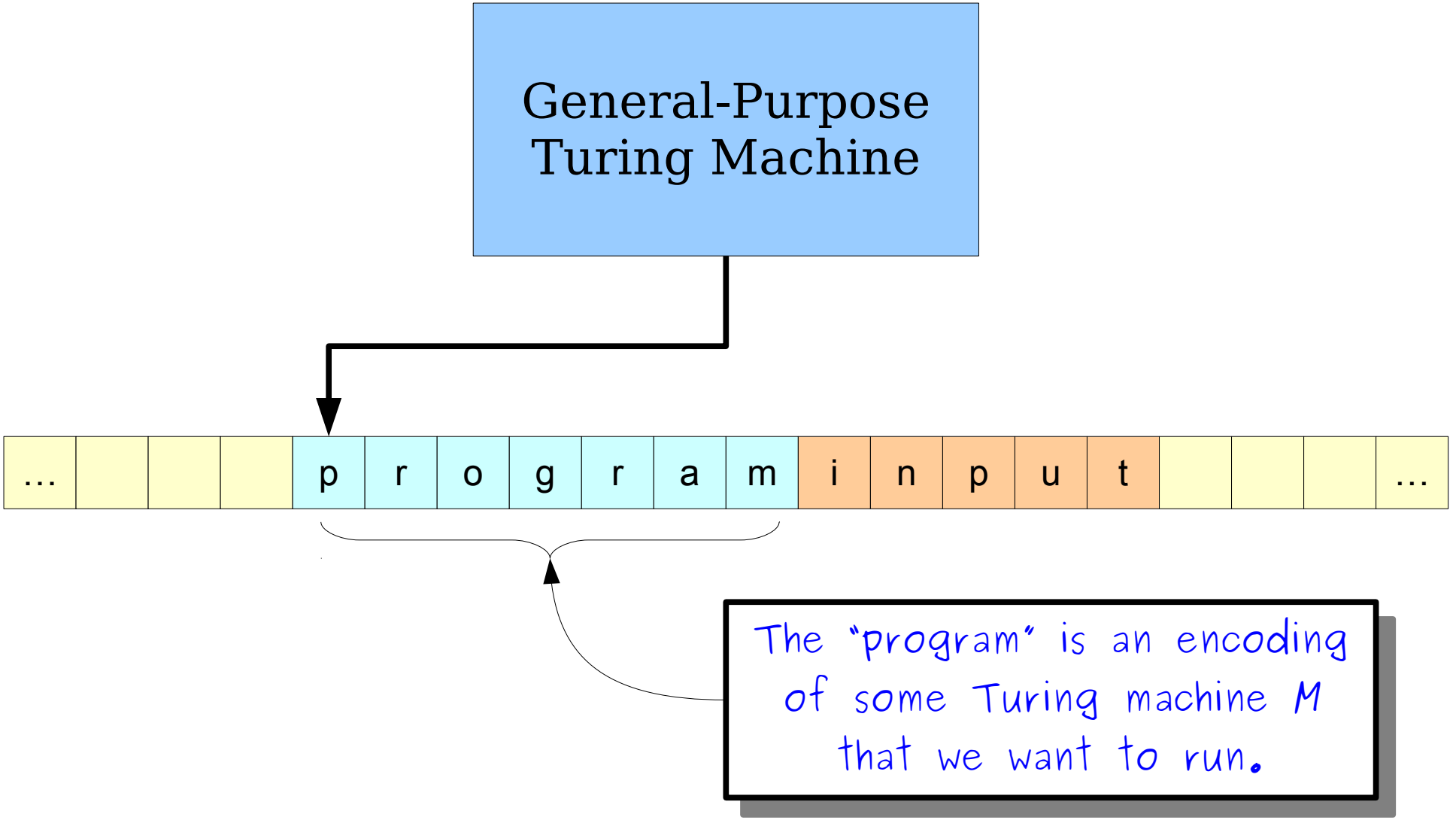
Encodings and TMs

- ***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 ⟨*M*⟩ is a string representing a string encoding of *M*.
  - A helpful analogy: think of *M* as an executable file and ⟨*M*⟩ as its source code.
- Because TMs can be encoded as strings, *it is possible to feed a TM as an input to another TM!*

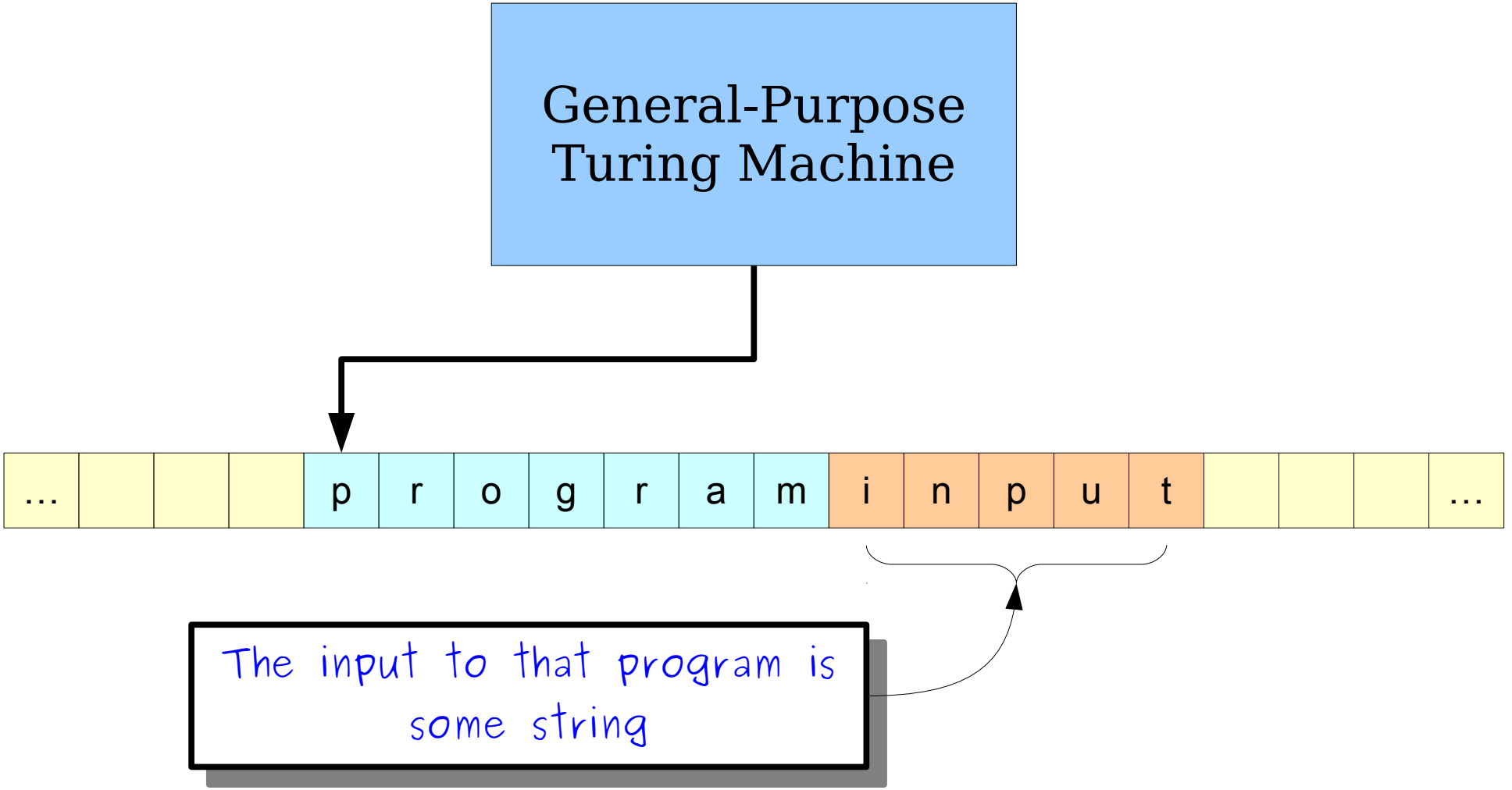
A Single Turing Machine?

- 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, then simulates the behavior of that TM.

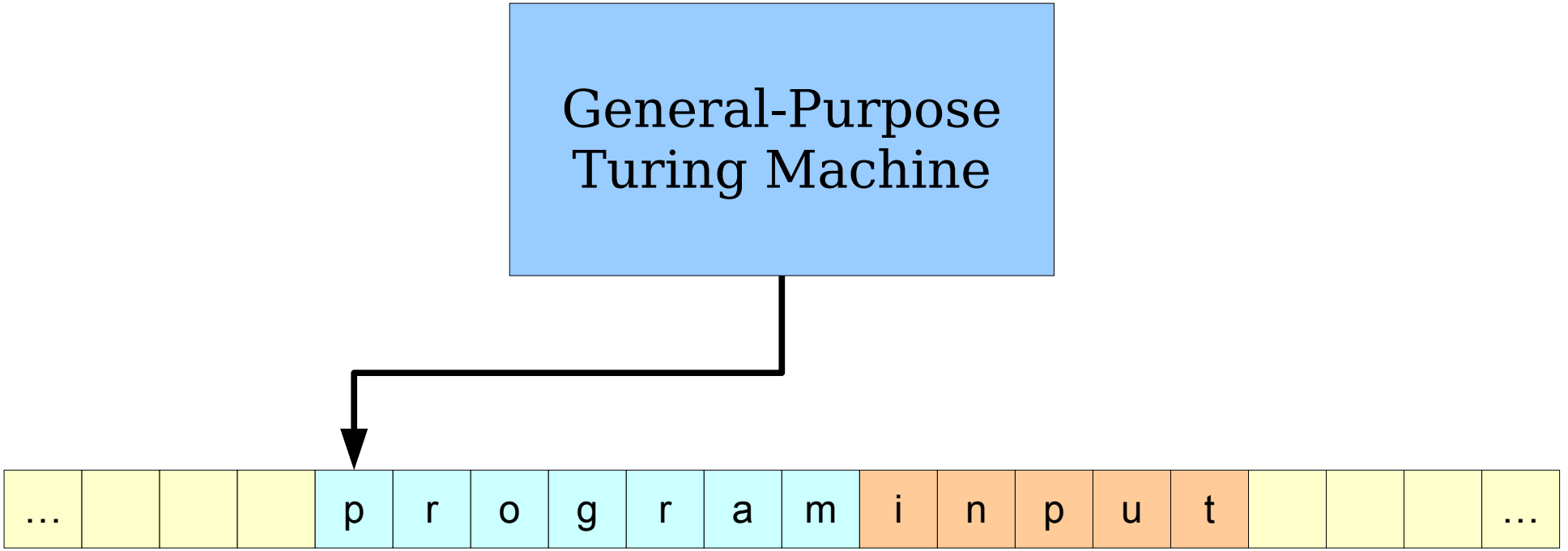
A Universal Machine



A Universal Machine

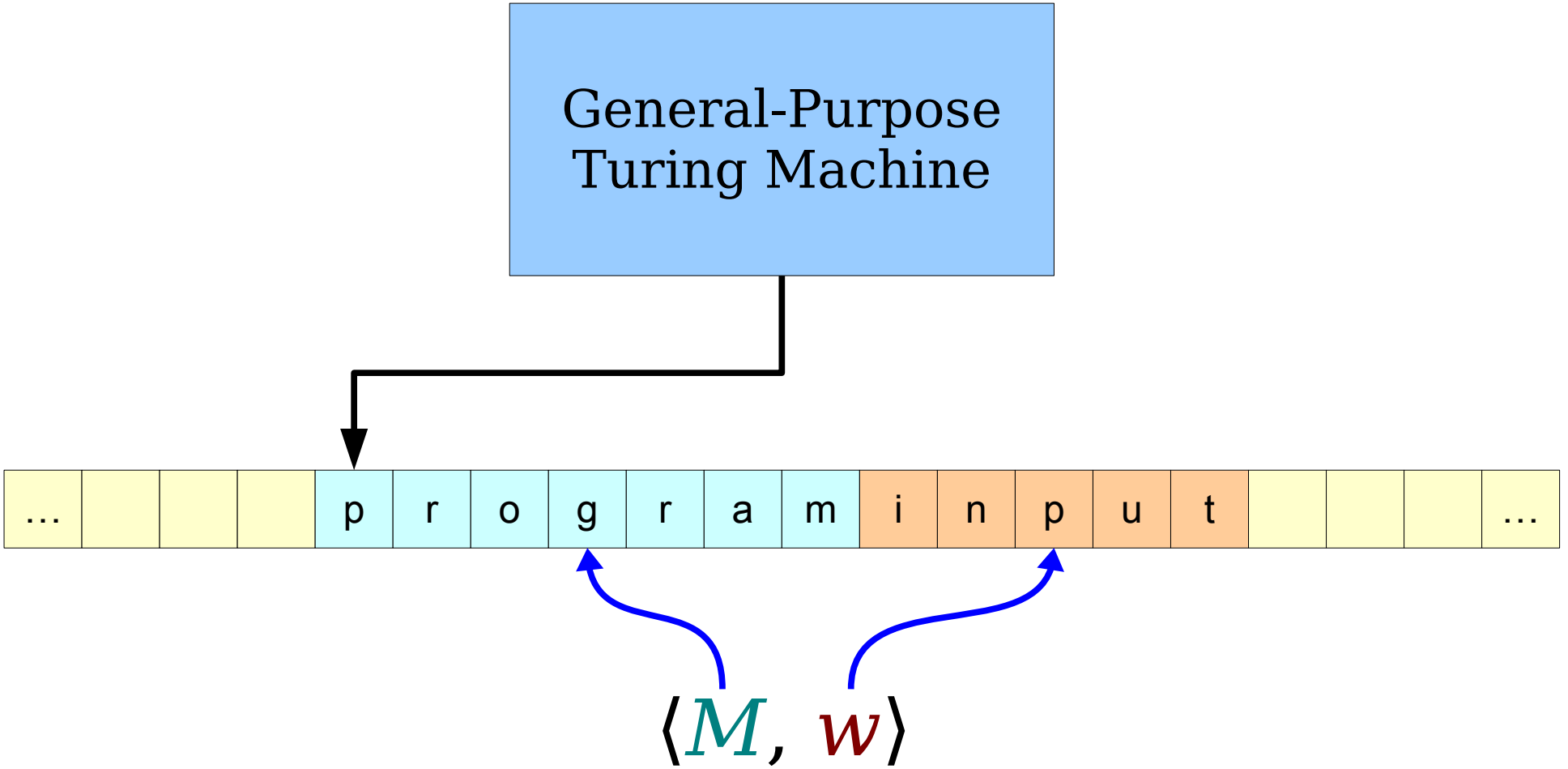


A Universal Machine



The input has the form $\langle M, w \rangle$, 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 $\mathbf{U_{TM}}$ 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).
- Conceptually:

U_{TM} = "On input $\langle M, w \rangle$, where M is a TM and $w \in \Sigma^*$:

Set up a simulation of M running on w.

```
while (true) {
```

If the simulated version of M is in an accepting state, then U_{TM} accepts $\langle M, w \rangle$.

If the simulated version of M is in a rejecting state, then U_{TM} rejects $\langle M, w \rangle$.

Otherwise, U_{TM} simulates one more step of M.

```
}
```
"

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

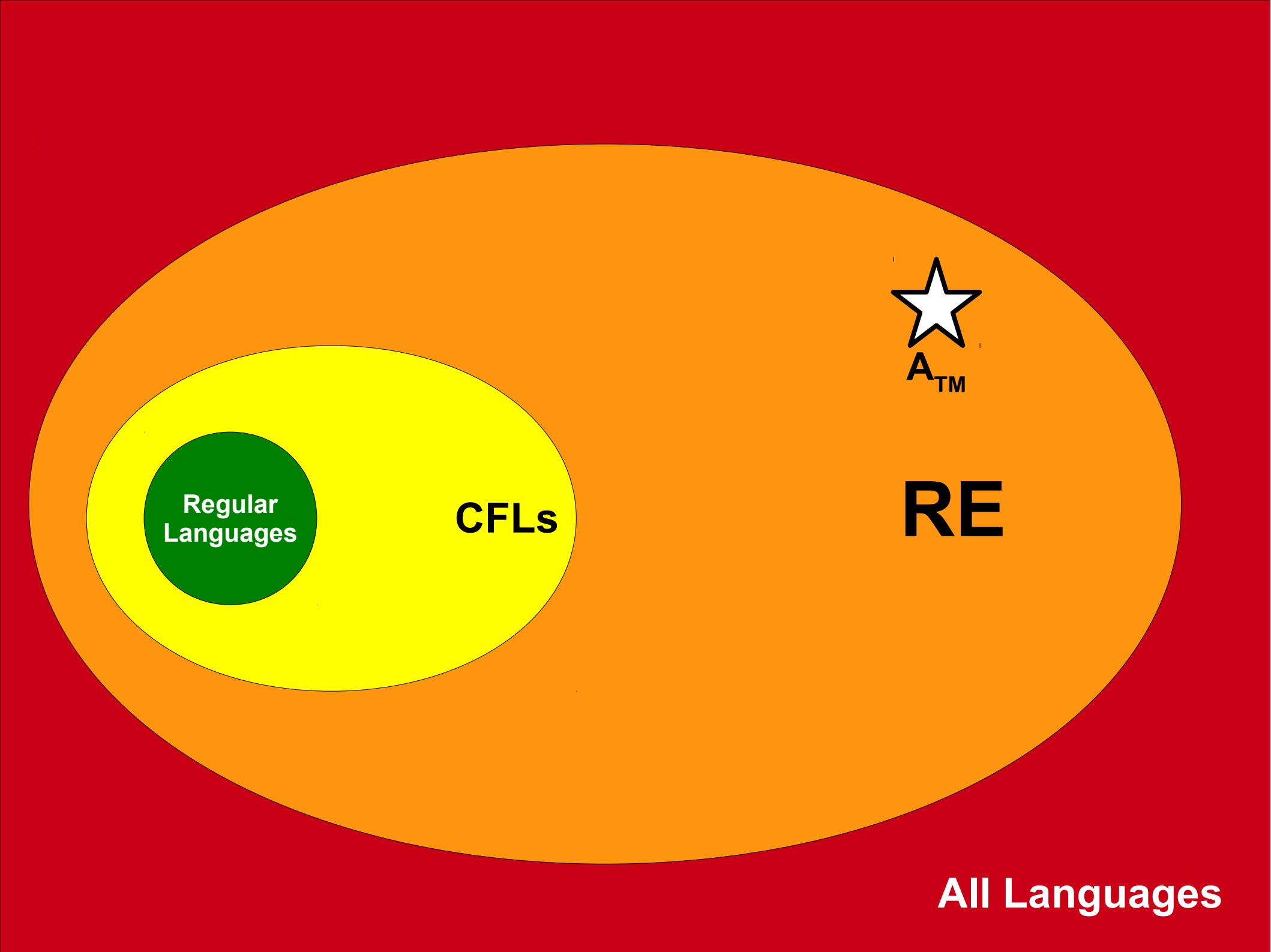
$$\mathscr{L}(M) = \{ w \in \Sigma^* \mid M \text{ accepts } w \}$$

- 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

Self-Referential Software

Quines

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.
- How would you write such a program?

Writing a Quine

Self-Referential Programs

- ***Claim:*** Going forward, assume that any program can be augmented to include a method called `mySource()` that returns a string representation of its source code.
- General idea:
  - Write the initial program with `mySource()` as a placeholder.
  - Use the Quine technique we just saw to convert the program into something self-referential.
  - Now, `mySource()` magically works as intended.

The Recursion Theorem

- There is a deep result in computability theory called ***Kleene's second recursion theorem*** that, informally, states the following:

  It is possible to construct TMs that perform arbitrary computations on their own descriptions.

- Intuitively, this generalizes our Quine constructions to work with arbitrary TMs.
- Want the formal statement of the theorem? Take CS154!