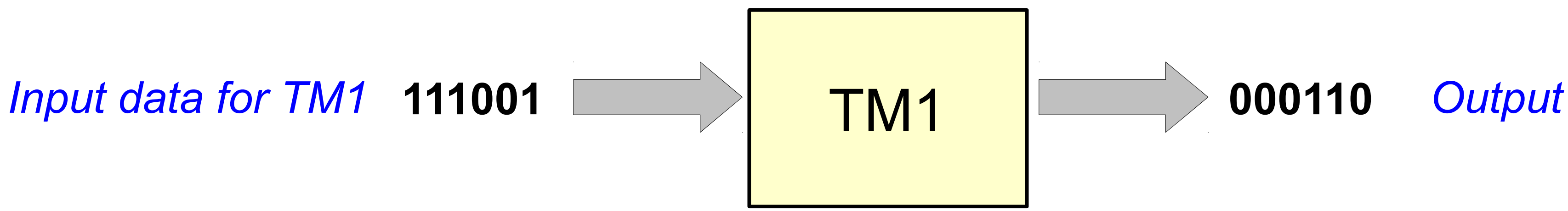
Reading for this week and next

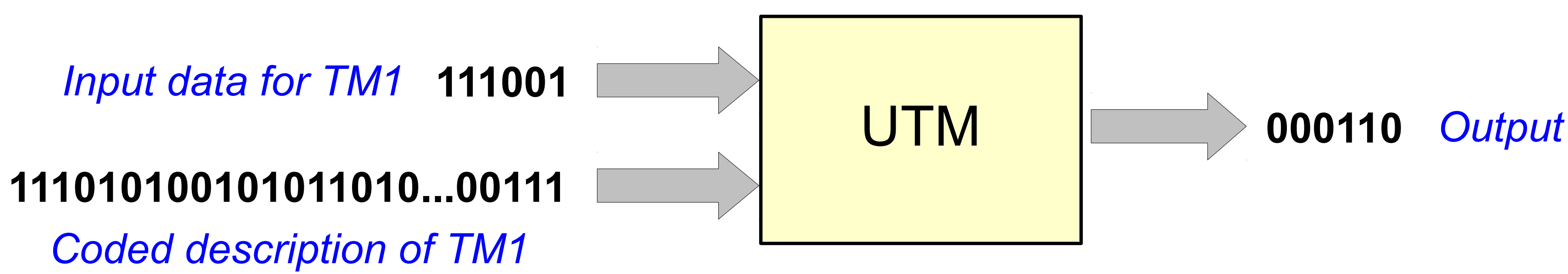
- *Complexity: a Guided Tour*
  - Chapters 5-8: background material on evolution and genetics
  - Chapter 9: genetic algorithms (“Robby the Robot”)
- *The Computational Beauty of Nature*
  - Sections 20.1 through 20.3: genetic algorithms

Universal Turing Machines

Universal Turing Machines

- A special TM, called a **Universal Turing Machine**, can simulate any other Turing machine





How to Encode a Turing Machine?

- States: s1, s2, s3, halt → 0, 00, 000, 0000, etc.
- Symbols: **x**, **y**, **z** → 0, 00, 000, etc.
- Moves: Right, Left, None → 0, 00, 000
- Rules:

  s1 **y** **y** R s3 → 0 **1** 00 **1** 00 **1** 0 **1** 000

  s2 **x** **z** L halt → 00 **1** 0 **1** 000 **1** 00 **1** 0000

111 0 **1** 00 **1** 00 **1** 0 **1** 000 **11** 00 **1** 0 **1** 000 **1** 00 **1** 0000 **111**

How to Encode a Turing Machine?

- States: s1, s2, s3, halt → 0, 00, 000, 0000, etc.
- Symbols: **x**, **y**, **z** → 0, 00, 000, etc.
- Moves: Right, Left, None → 0, 00, 000
- Rules:

  s1 **y** **y** R s3 → 0 **1** 00 **1** 00 **1** 0 **1** 000

  s2 **x** **z** L halt → 00 **1** 0 **1** 000 **1** 00 **1** 0000

111 0 **1** 00 **1** 00 **1** 0 **1** 000 **11** 00 **1** 0 **1** 000 **1** 00 **1** 0000 **111**

1110100100101000110010100010010000111

How to Encode a Turing Machine?

- States: s1, s2, s3, halt → 0, 00, 000, 0000, etc.
- Symbols: **x**, **y**, **z** → 0, 00, 000, etc.
- Moves: Right, Left, None → 0, 00, 000
- Rules:

s1 **y** **y** R s3 → 0 **1** 00 **1** 00 **1** 0 **1** 000

s2 **x** **z** L halt → 00 **1** 0 **1** 000 **1** 00 **1** 0000

111 0 **1** 00 **1** 00 **1** 0 **1** 000 **11** 00 **1** 0 **1** 000 **1** 00 **1** 0000 **111**

= 125,176,464,519 in decimal

Example: The "Binary Inverter" TM

- States: s1, halt → 0, 00
- Symbols: **0**, **1**, _ → 0, 00, 000
- Moves: Right, Left, None → 0, 00, 000
- Rules:

s1	**0**	**1**	R	s1	→	0 **1** 0 **1** 00 **1** 0 **1** 0
s1	**1**	**0**	R	s1	→	0 **1** 00 **1** 0 **1** 0 **1** 0
s1	_	_	*	halt	→	0 **1** 000 **1** 000 **1** 000 **1** 00

111 0101001010 11 0100101010 11 01000100010000100 111

1110101001010110100101010110100010001000100111

Example: The “Binary Inverter” TM

- States: s1, halt → 0, 00
- Symbols: **0**, **1**, _ → 0, 00, 000
- Moves: Right, Left, None → 0, 00, 000
- Rules:

s1	**0**	**1**	R	s1	→	0 **1** 0 **1** 00 **1** 0 **1** 0
s1	**1**	**0**	R	s1	→	0 **1** 00 **1** 0 **1** 0 **1** 0
s1	_	_	*	halt	→	0 **1** 000 **1** 000 **1** 000 **1** 00

111 0101001010 11 0100101010 11 01000100010000100 111

= 64,414,398,685,735 in decimal

Your Turn

- States: s1, halt → 0, 00
- Symbols: **0**, **1**, **_** → 0, 00, 000
- Moves: Right, Left, None → 0, 00, 000
- Rules:

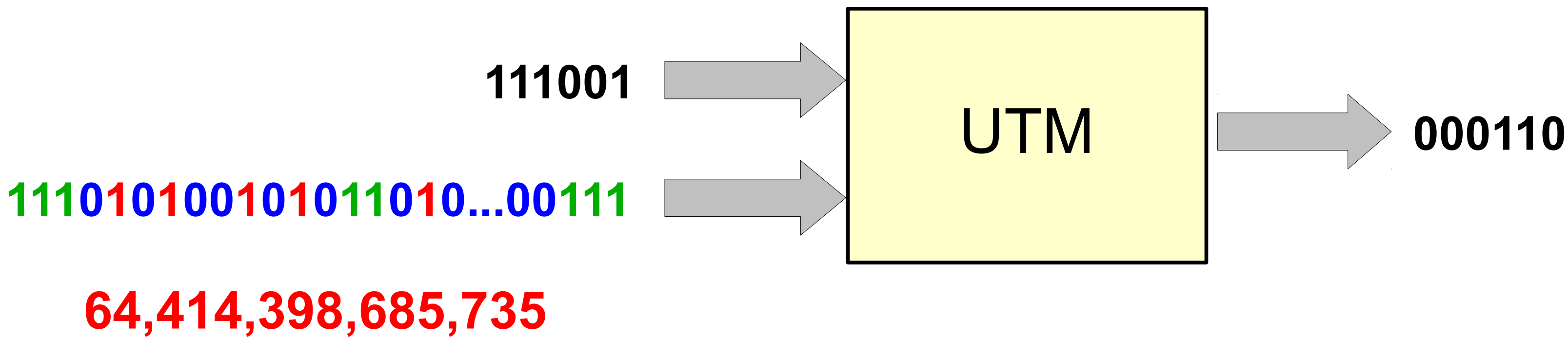
s1	**0**	**0**	R	s1	→	0 **1** 0 **1** 0 **1** 0 **1** 0
s1	**1**	**1**	L	s1	→	0 **1** 00 **1** 00 **1** 00 **1** 0
s1	**_**	**_**	*	halt	→	0 **1** 000 **1** 000 **1** 000 **1** 00

111 010101010 11 0100100100010 11 01000100010000100 111

1110101010101101001001001011010001000100010 0111

= 129,014,683,017,767 in decimal

The Universal Machine



- UTM's own internal rules are **fixed**
- Coded description acts as a **program** that UTM executes on the input string 111001
- Or we could say that the number **64,414,398,685,735** **acts** on the input 111001 to produce the output 000110

The Universal Machine

Before Turing, things were done to numbers. After Turing, numbers began doing things.

—George Dyson, *Turing's Cathedral*

I am thinking about something much more important than bombs. I am thinking about computers.

—John Von Neumann, 1946

The fact that there is a universal machine to imitate all other machines...was understood by von Neumann and a few others. And when he understood it, then he knew what we could do.

—Julian Bigelow, chief engineer of the IAS Electronic Computer Project

The Universal Machine

- The existence of the UTM is what makes computers **fundamentally different** from other machines
- Computers are the only machines that can **simulate any other machine** to an arbitrary degree of accuracy
- **Universality** is why computers have taken over the world!

The Universal Machine

Even the word "cellphone" is a misnomer. They could just as easily be called cameras, video players, Rolodexes, calendars, tape recorders, libraries, diaries, albums, televisions, maps or newspapers.

—Chief Justice John Roberts Jr.,
June 25, 2014 Supreme Court ruling that police need warrants to search cellphones of people under arrest

The Universal Machine

- Are Turing Machines really as powerful as real computers?
  - Unlimited memory (infinite tape)
  - Speed / efficiency is irrelevant
  - Any type of data can be encoded in binary (numbers, text, pictures, sounds, movies, etc.)

The Universal Machine

- All proposed models of computation have turned out to be exactly equivalent to one another:
  - Turing machines
  - Lambda calculus
  - Recursive functions
  - Post production systems
  - Random access machines
  - All programming languages (Python, Javascript, C, ...)
  - etc. etc.

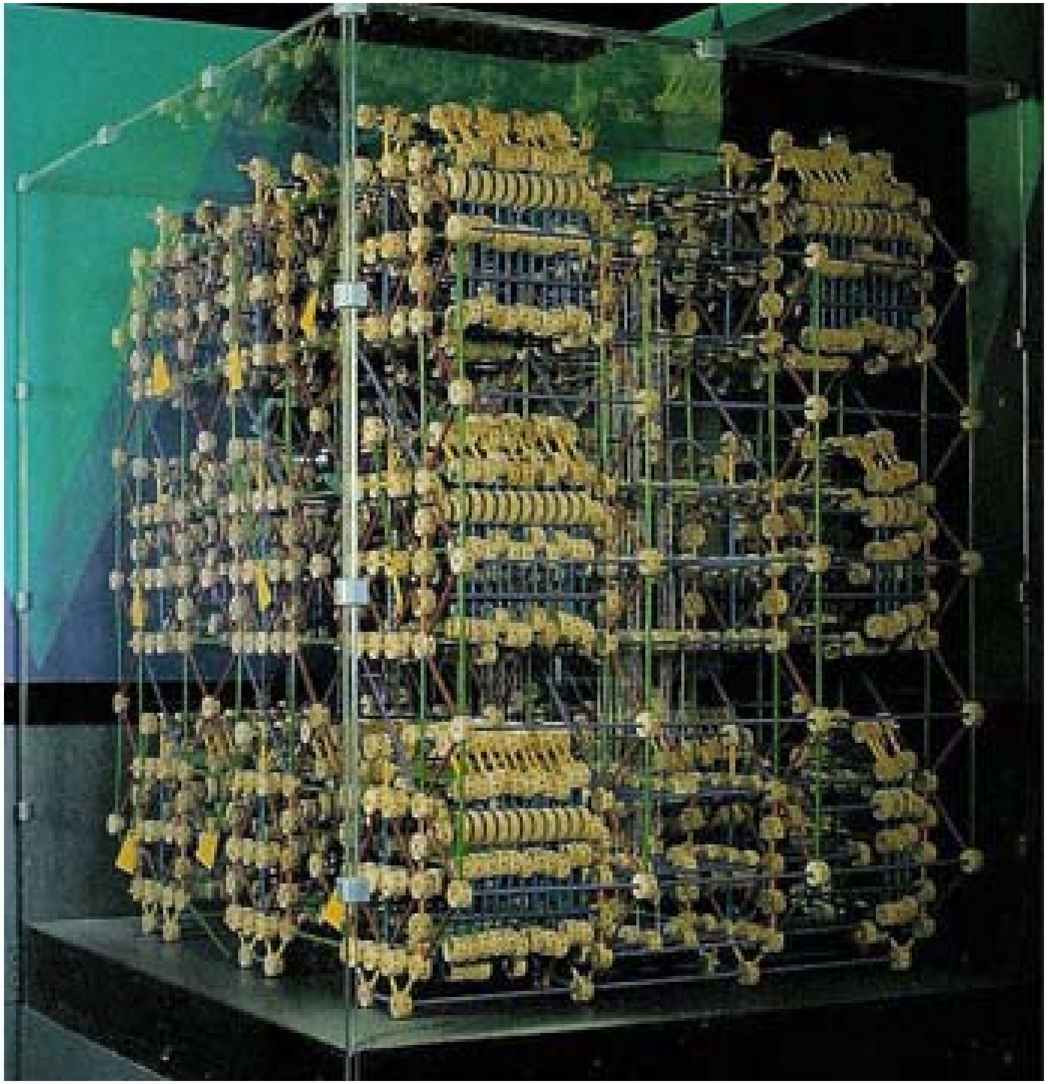
The Universal Machine

- **Church-Turing Thesis:**

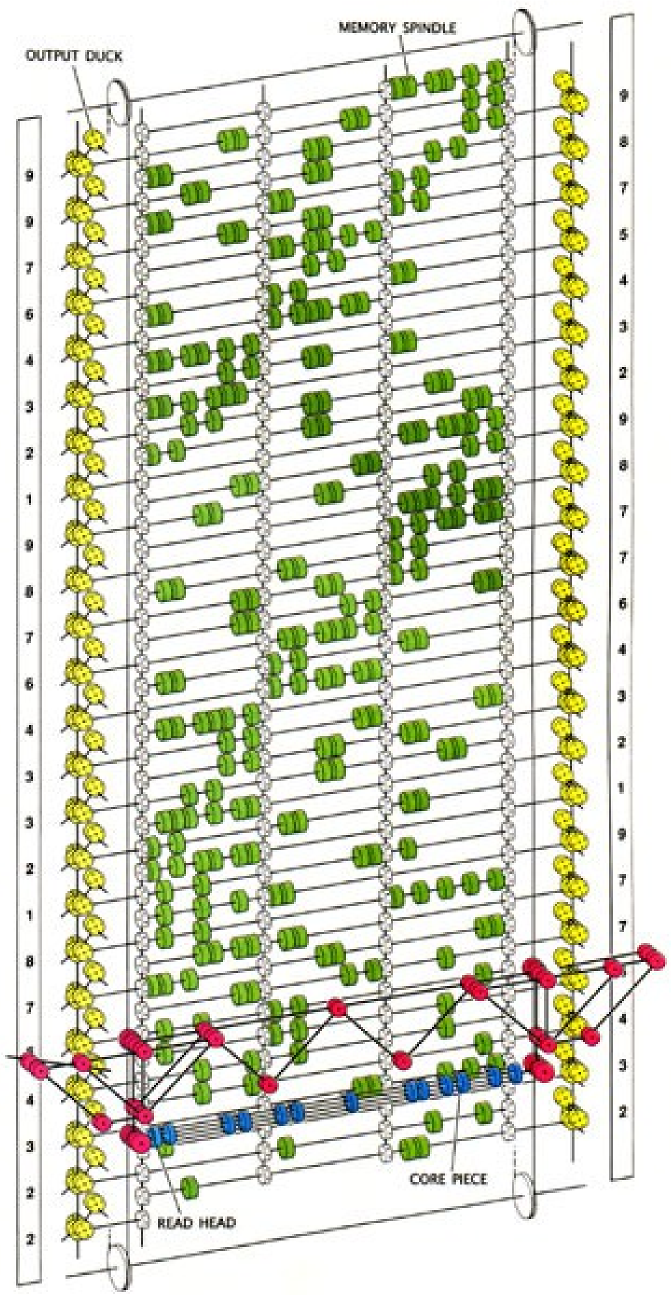
> Anything that is computable can be computed by a suitably programmed Turing machine

- Choice of **programming substrate** doesn't matter
- What matters is the organization and flow of **information**
- You can build a computer out of **Tinkertoys** if you like!

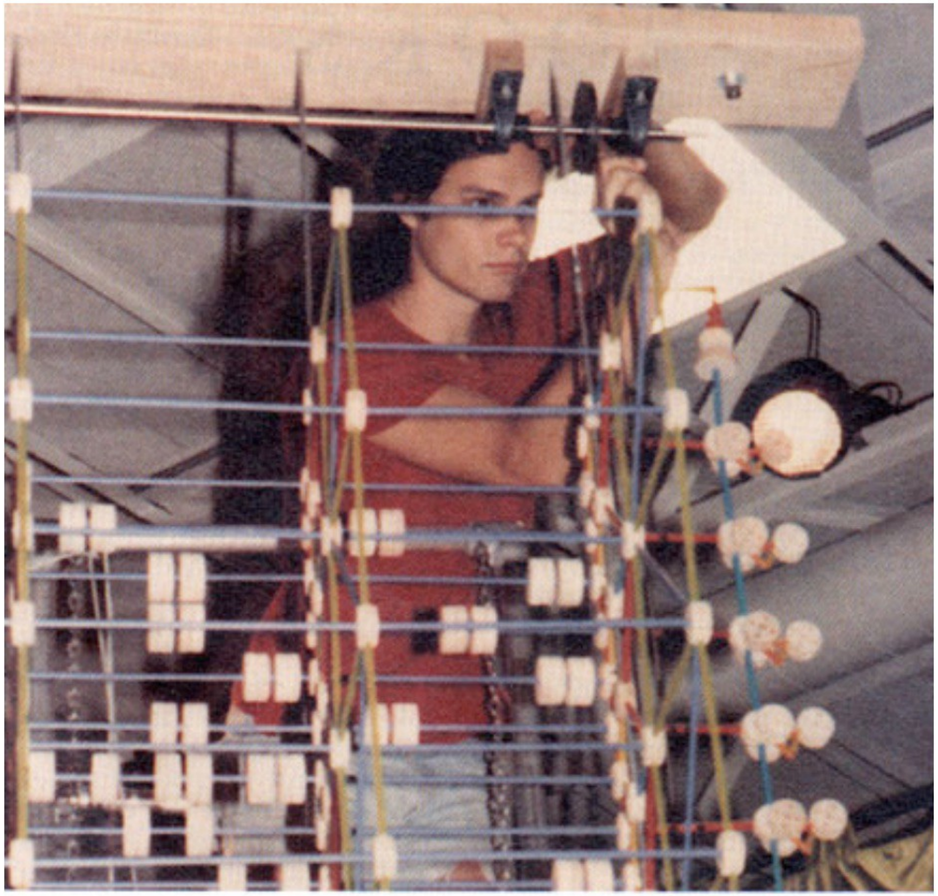
Tinkertoy Computer for Playing Tic-Tac-Toe



Tinkertoy Computer for Playing Tic-Tac-Toe



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



Edward Hardebeck helps to assemble the Tinkertoy computer

The Limits of Computation

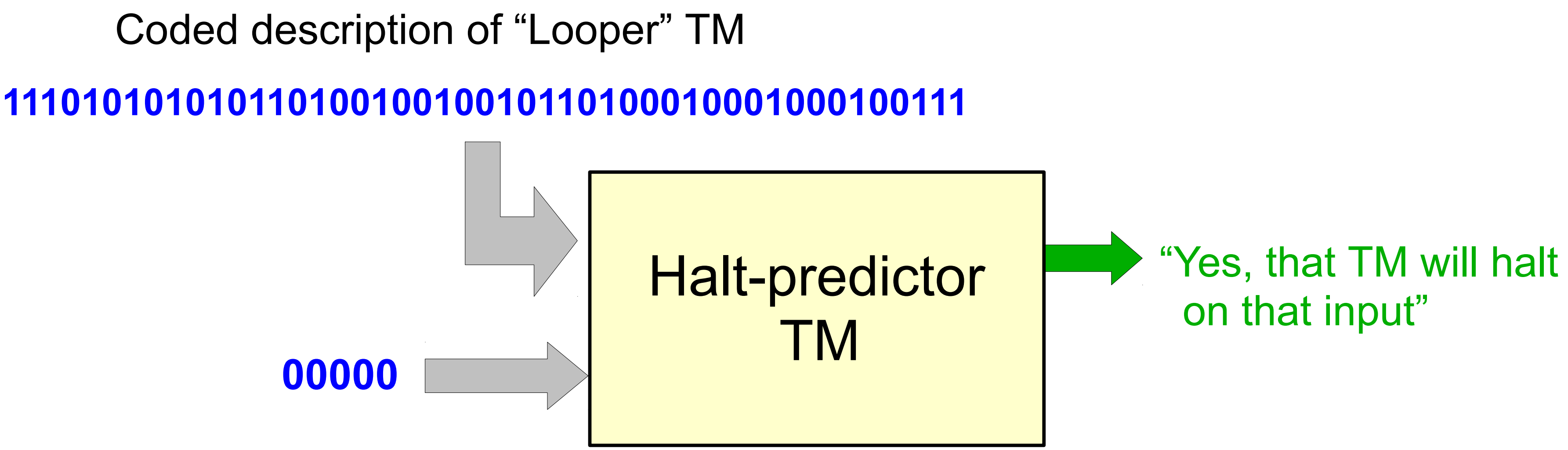
- Is there anything a TM **cannot** compute, in principle?
- YES! No TM can **infallibly** predict whether another TM will get stuck in an infinite loop when run on some input
- Example:

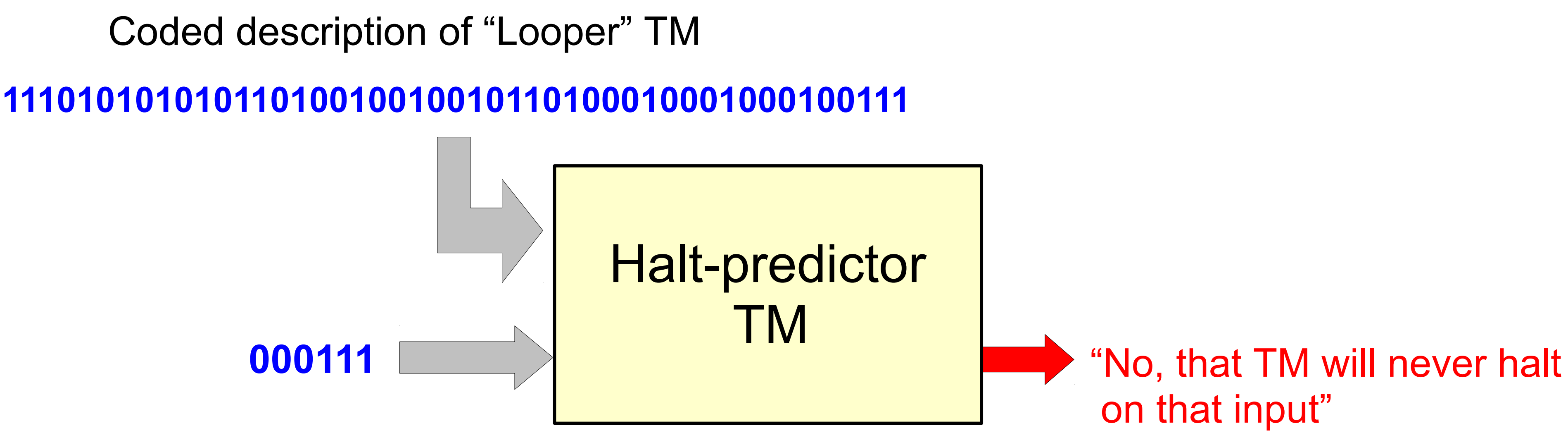
s1	0	0	R	s1
s1	1	1	L	s1
s1	_	_	*	halt

"Looper TM"

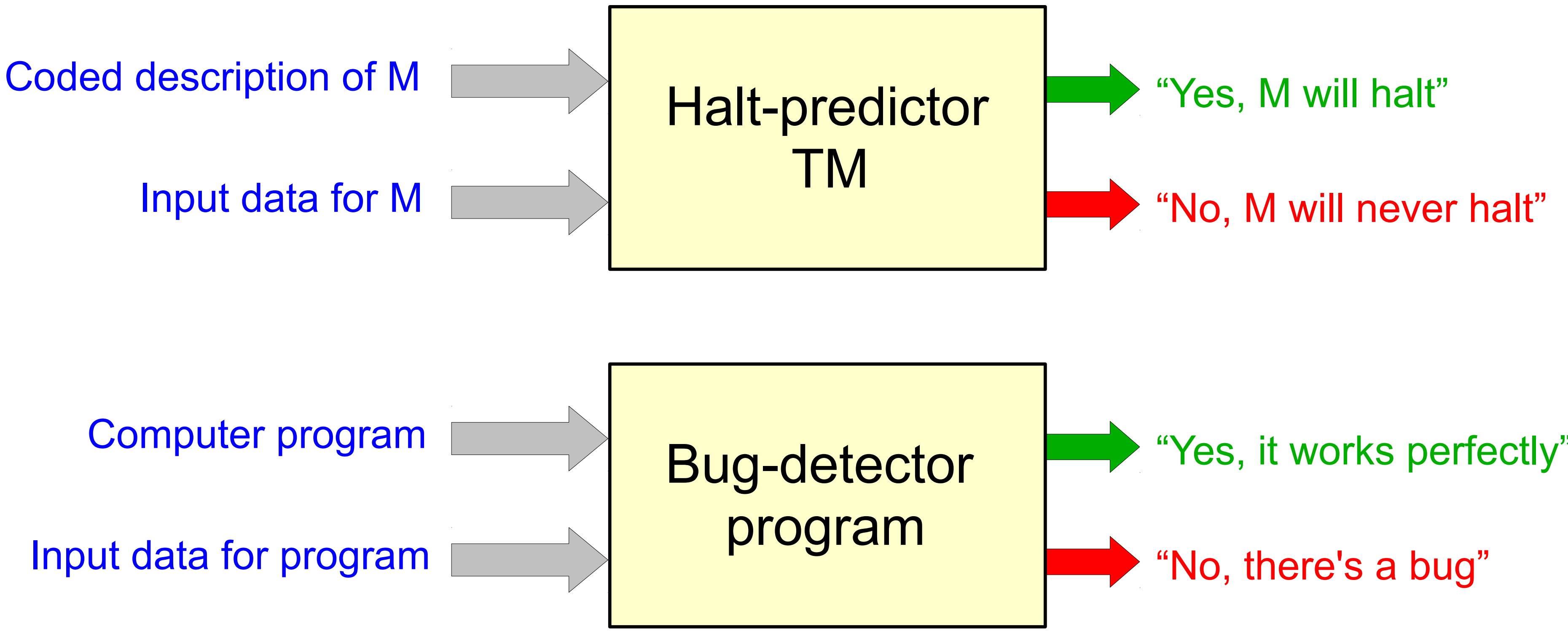
- Input: **00000** Result: halts after 5 steps
- Input: **000111** Result: never halts (infinite loop)

The Halting Problem





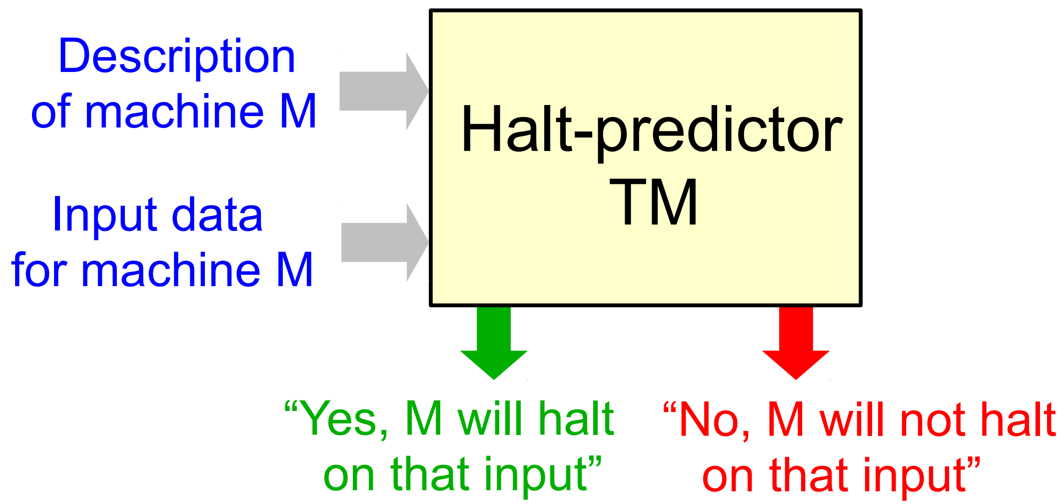
The Halting Problem



- The task of deciding in advance if an arbitrary computation will ever terminate cannot be described computationally
- This was proven by Turing in his 1936 paper

Outline of Turing's Argument

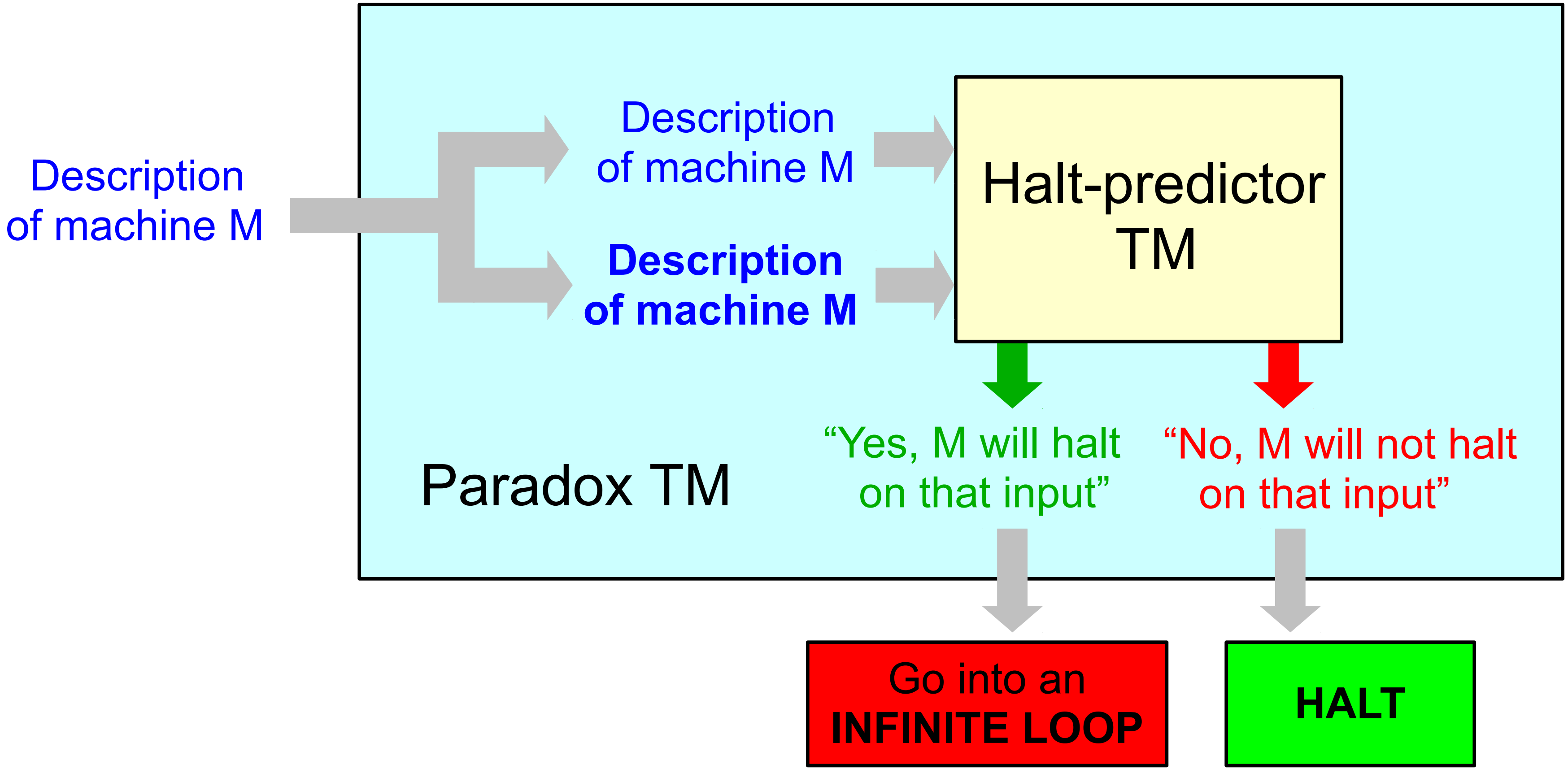
(1) **Assume for now** that the Halt-predictor TM actually exists



Outline of Turing's Argument

(1) **Assume for now** that the Halt-predictor TM actually exists

(2) **Construct** a new TM called **Paradox** that uses Halt-predictor

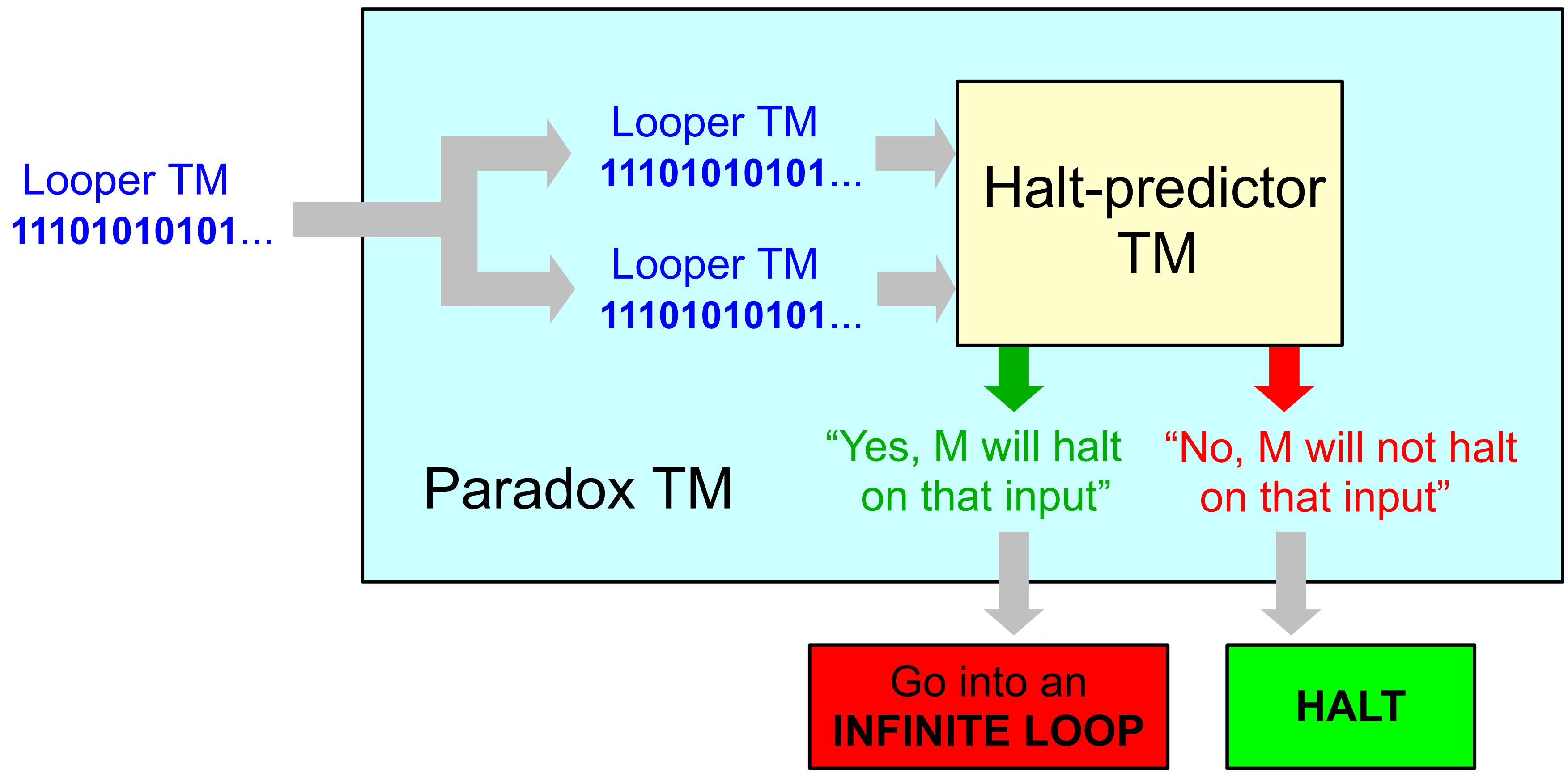


Outline of Turing's Argument

(1) **Assume for now** that the Halt-predictor TM actually exists

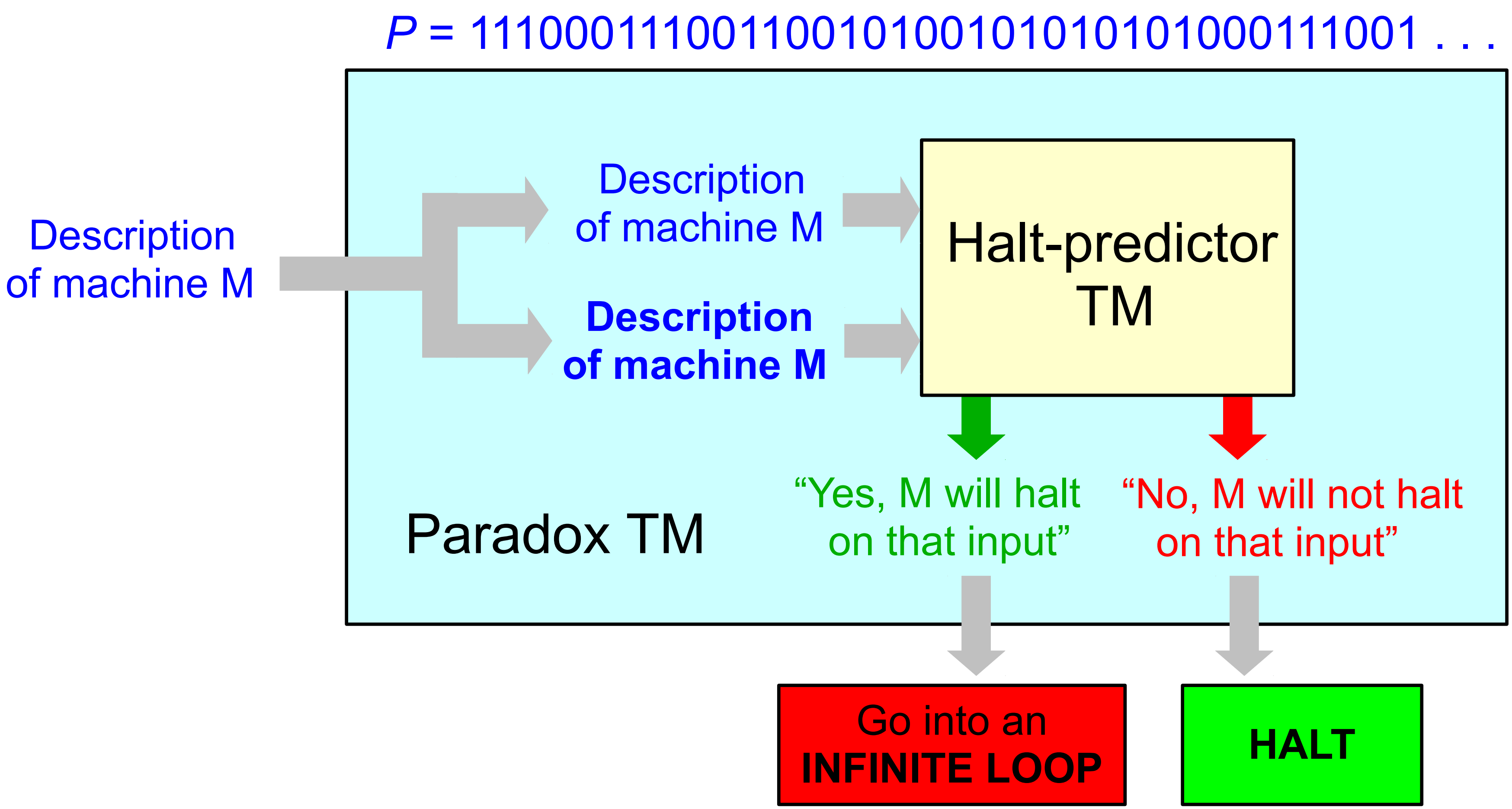
(2) **Construct** a new TM called **Paradox** that uses Halt-predictor

Example: we could feed Paradox the Looper TM description



Outline of Turing's Argument

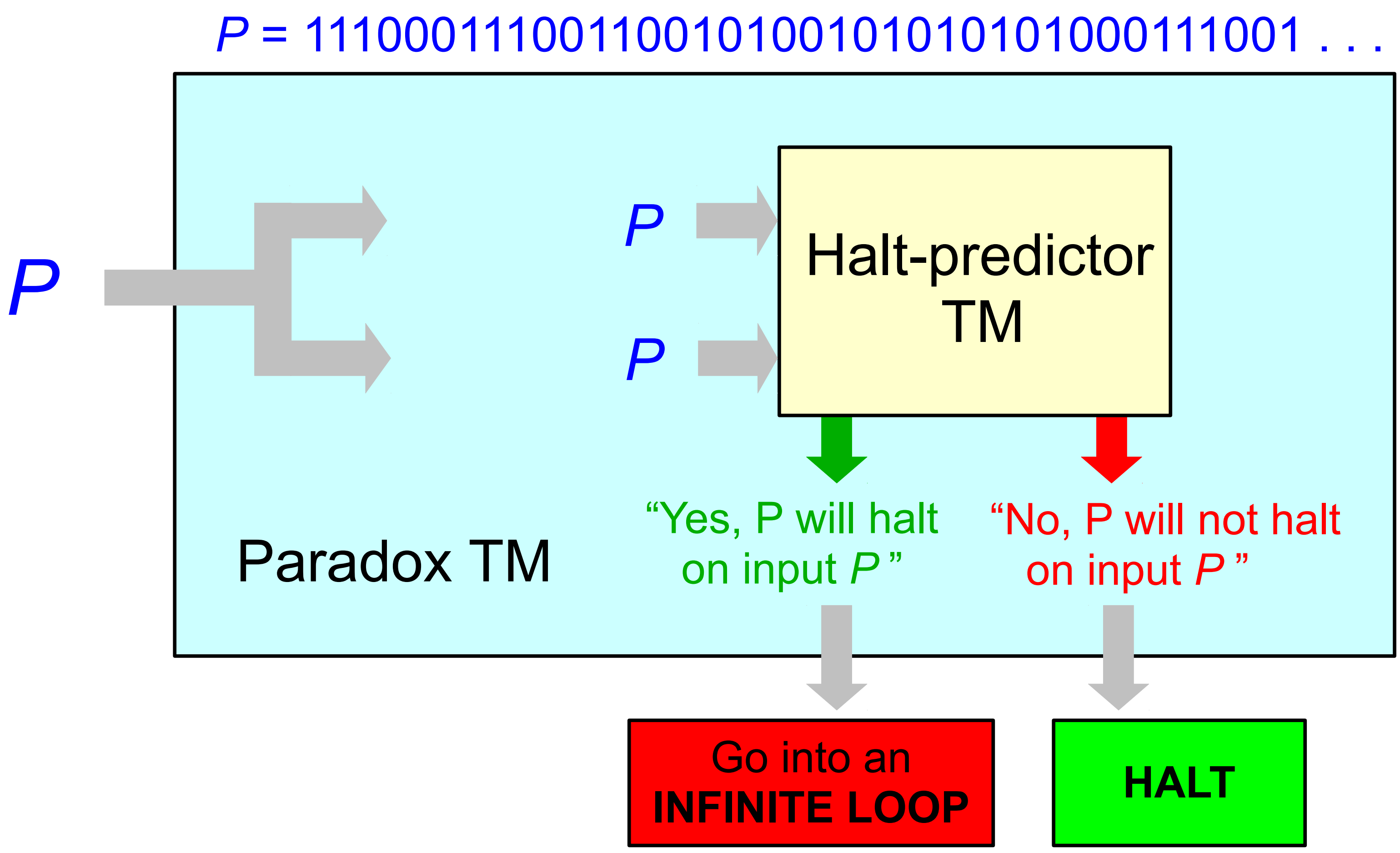
(3) Write down the **binary description *P*** of the Paradox TM



Outline of Turing's Argument

(3) Write down the **binary description *P*** of the Paradox TM

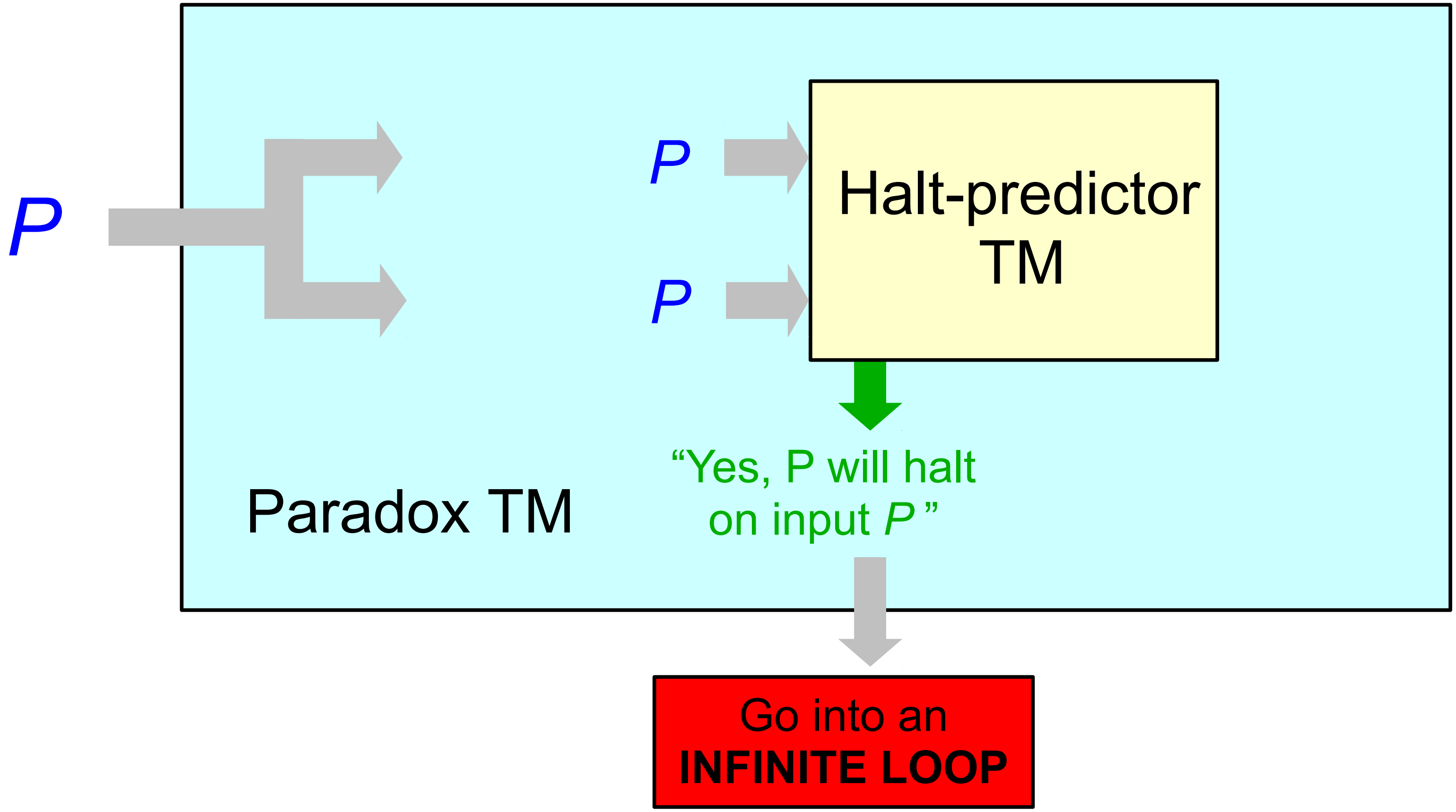
(4) Feed the description *P* to the Paradox TM itself



Outline of Turing's Argument

If Halt-predictor says “Yes”, then P **never halts**

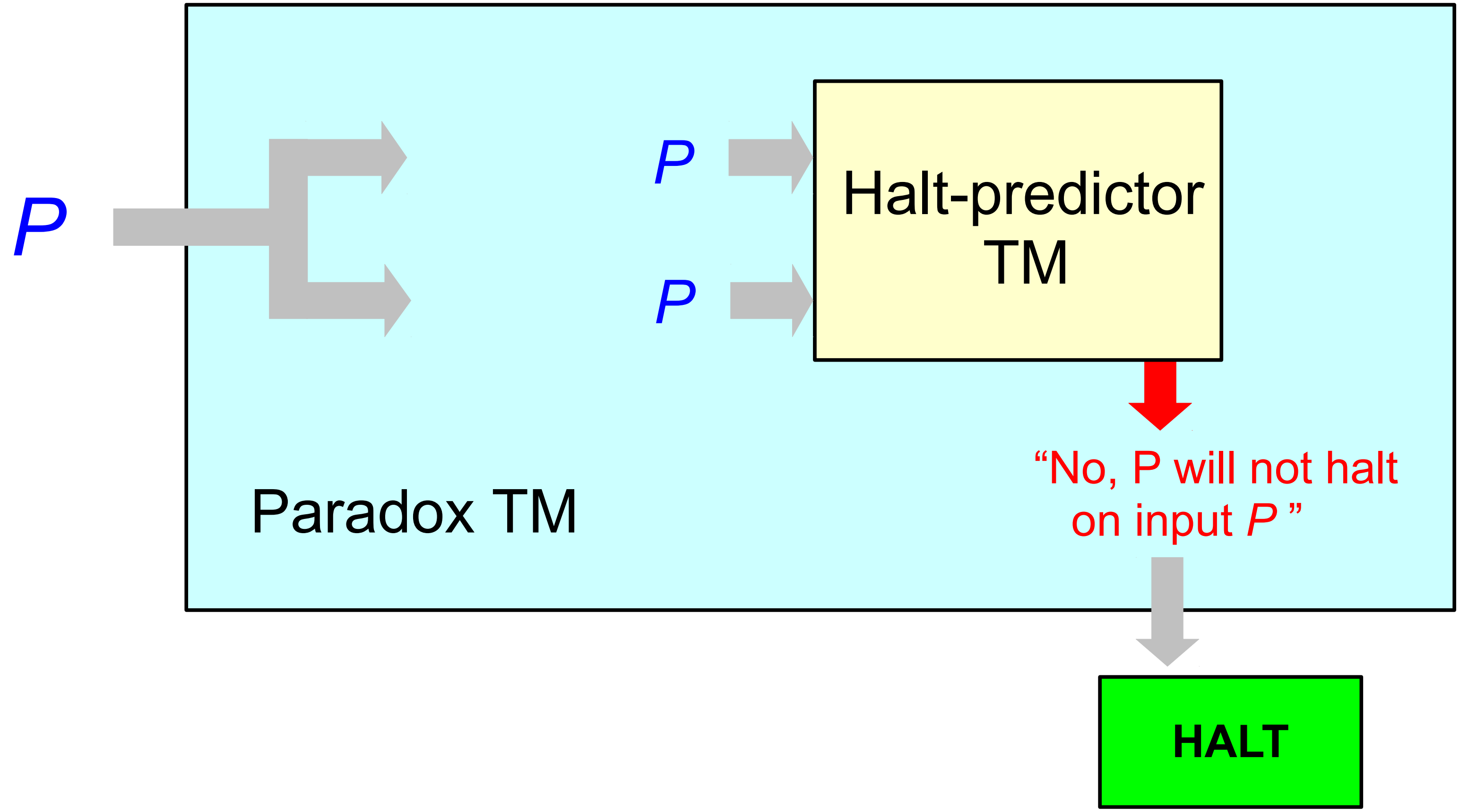
*This **contradicts** what Halt-predictor just said!*



Outline of Turing's Argument

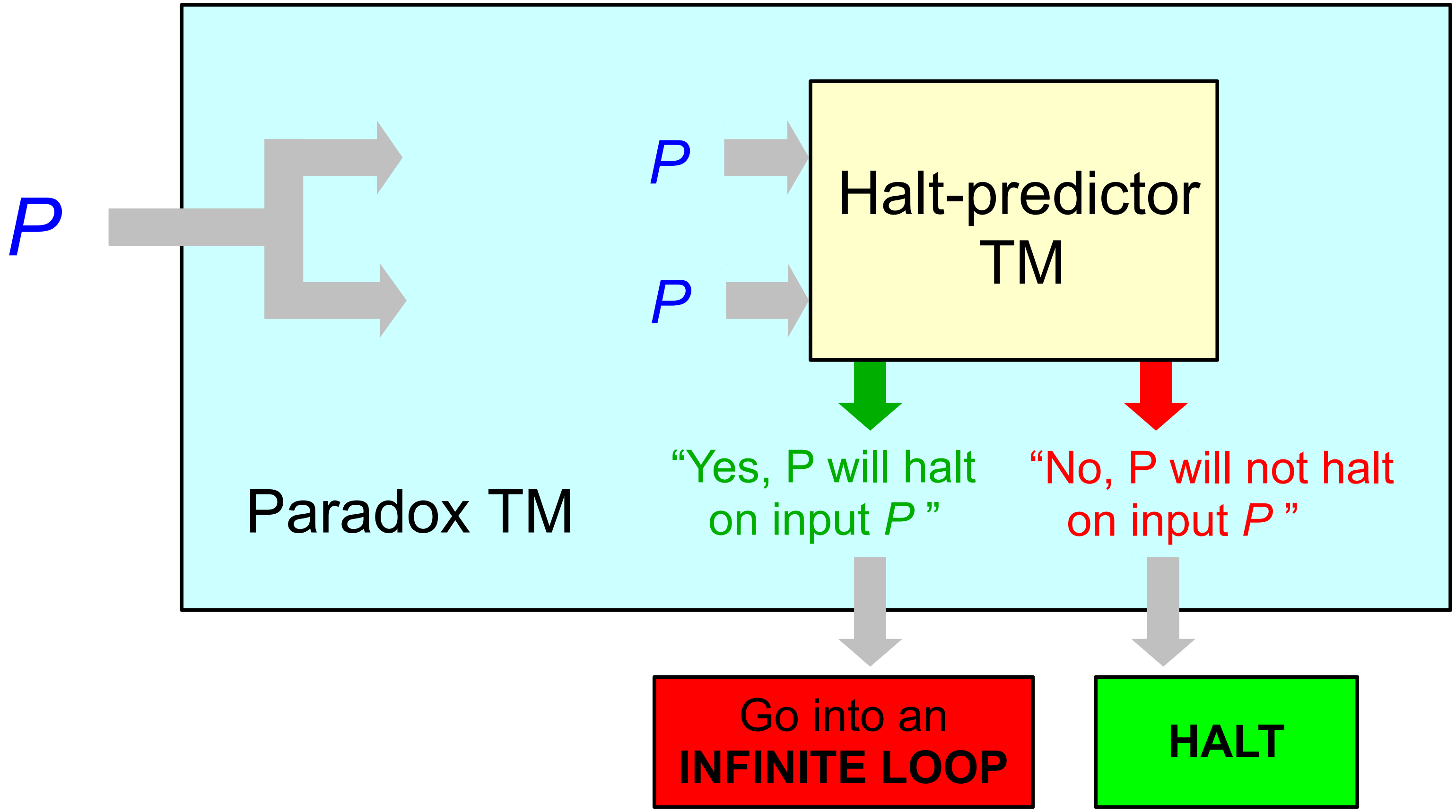
If Halt-predictor says "No", then P **halts**

*This **contradicts** what Halt-predictor just said!*



Outline of Turing's Argument

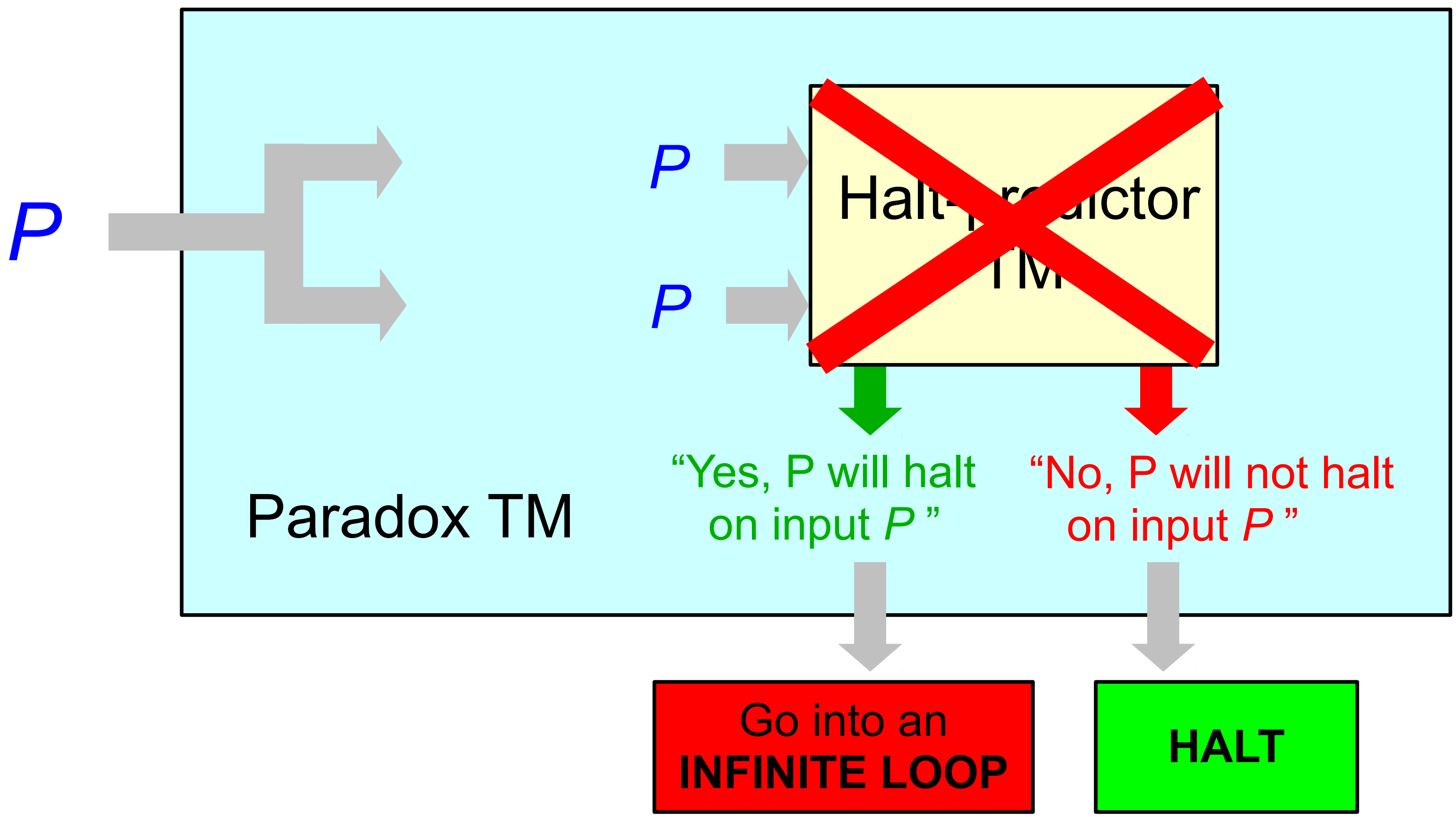
Either way, we get a **logical contradiction!**



Outline of Turing's Argument

The only possible conclusion:

The Halt-predictor TM cannot exist



Undecidable Problems

- The Halting Problem was the first **undecidable problem** to be discovered
- … but certainly not the last
- The class of undecidable problems is **infinitely large**
- The study of undecidable problems constitutes an extremely rich area of **theoretical computer science**