# CMSC 28100

# Introduction to Complexity Theory

Spring 2025
Instructor: William Hoza

# The nature of this course

- In this course, we will study
  - The mathematical and philosophical foundations of computer science
  - The ultimate limits of computation
- This course will give you powerful conceptual tools for reasoning about computation
- There will be very little programming
- Homework and exams will be primarily proof-based

# Who this course is designed for

- CS students, math students, and anyone who is curious
- Prerequisites:
  - Experience with mathematical proofs
  - CMSC 27200 or CMSC 27230 or CMSC 37000, or MATH 15900 or MATH 15910 or MATH 16300 or MATH 16310 or MATH 19900 or MATH 25500

# Who this course is designed for

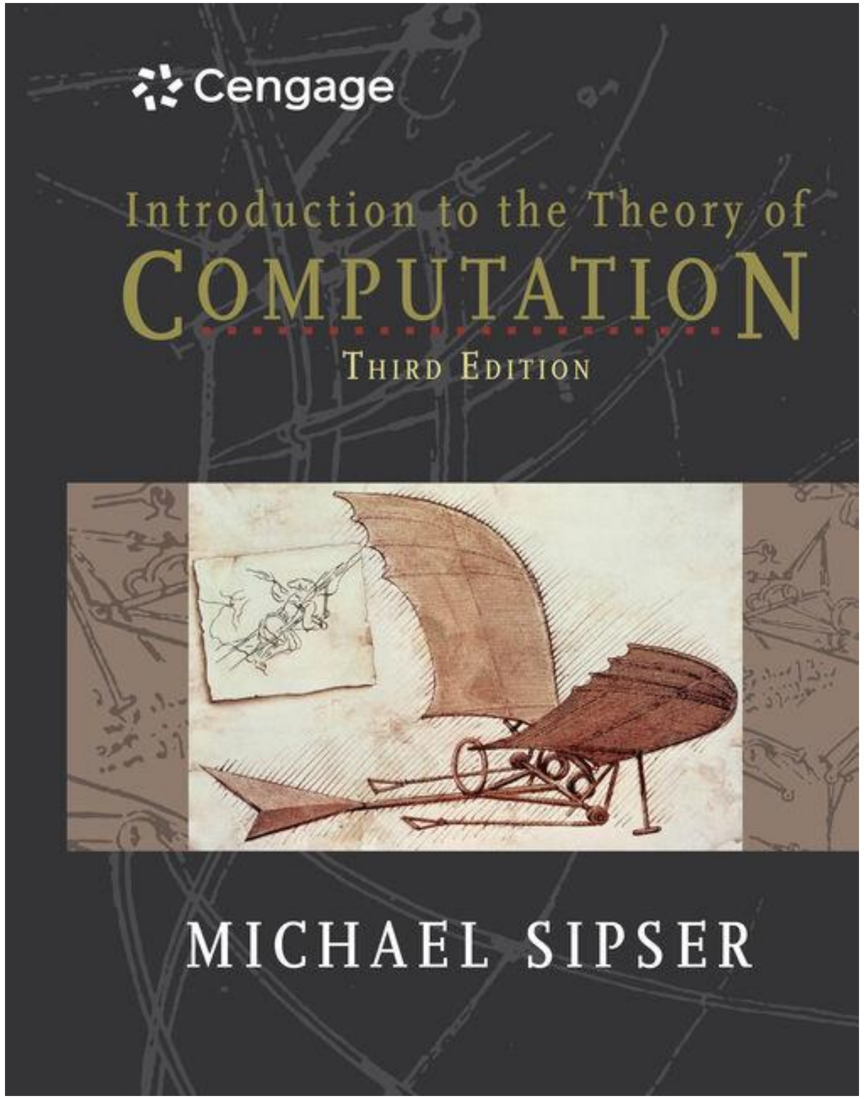
- It’s okay if you don’t consider yourself “theory-oriented”
- You belong here
- It’s my job to give you resources so you can learn and succeed
- I also consider it my job to persuade you that complexity theory is important, interesting, enlightening, fun, cool, and worthy of your attention

# Class participation

- Please ask questions!
    - “What does that notation mean?”
    - “I forget what a _____ is. Can you remind me?”
    - “How do we know _____?”
    - “I’m lost. Can you explain that again?”

# Textbook

- Classic
- Popular
- High-quality
- Not free 😦



# Assessment

- 28 homework exercises
  - Exercises 1-4 are due **Tuesday, April 1**
- Midterm exam in class on 4/23
- Final exam at the end of the quarter

# My office hours

- Mondays (starting next week), 9am to 11am, JCL 205
- Stop by! This is a great time for discussions
  - If you are confused/curious about something, I'll try to help you figure it out
  - If you are stuck on a homework exercise, I'll try to think of a good hint
  - If you have a complaint, I'll listen and try to make things better

# Teaching assistants

- Zelin Lv
  - Office hours: Fridays, 2pm to 3pm, JCL 205
- Yakov Shalunov
  - Office hours: Thursdays, 5pm to 6pm, JCL 205

# Technology

- Canvas: https://canvas.uchicago.edu/courses/62607
  - Homework exercises; practice exams; official solutions
- Course webpage: https://williamhoza.com/teaching/spring2025-intro-to-complexity
  - Course policies; slides
- Ed: https://edstem.org/us/courses/76353/
  - Discussions (≈ office hours); announcements
- Gradescope: https://www.gradescope.com/courses/988815
  - Submitting homework solutions; grades and feedback

The central question of this course:

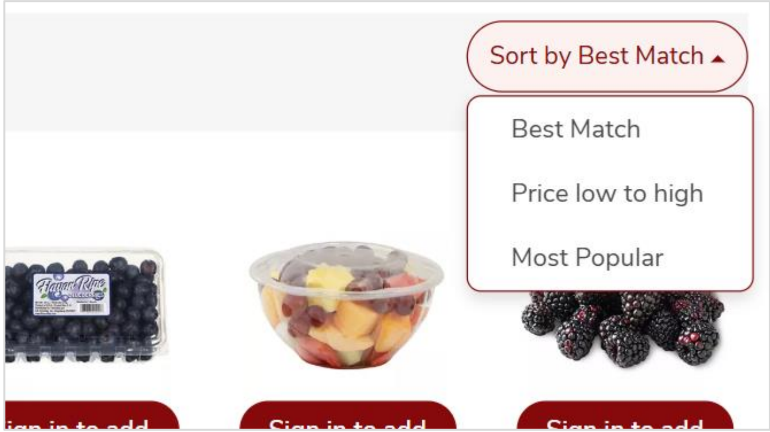
Which problems

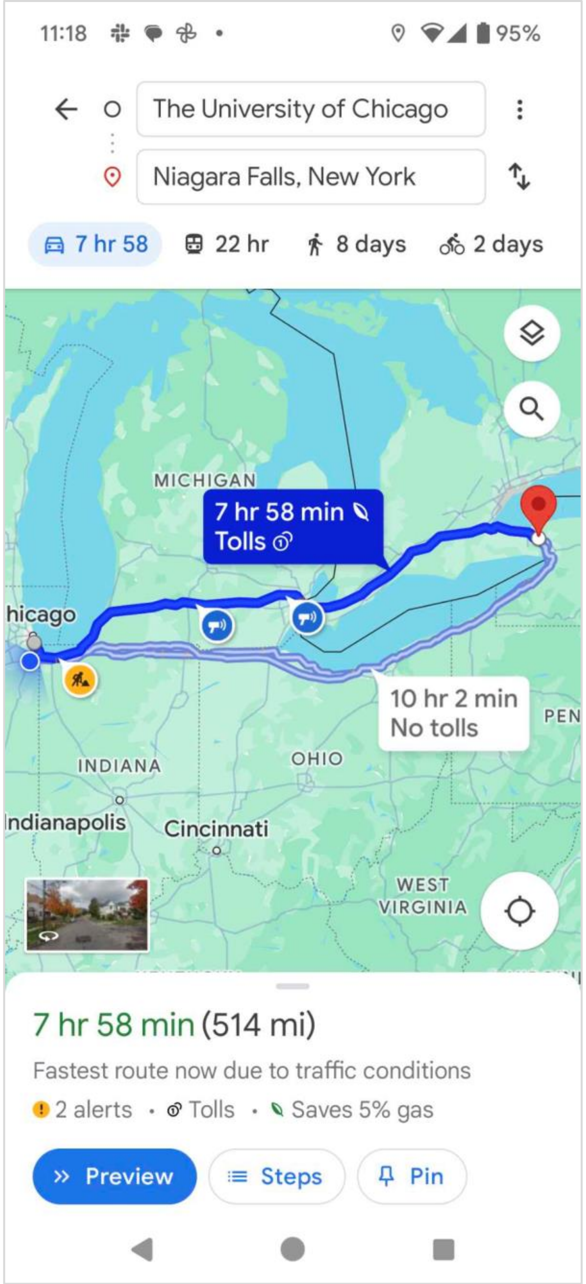
can be solved

through computation?

# Examples

- Many problems can be solved through computation:
  - Multiplication
  - Sorting
  - Shortest path




- Are there any problems that cannot be solved through computation?

# Impossibility proofs

- We will take a mathematical approach to this question
- We will formulate precise mathematical models
  - “Computation”
  - “Problem”
  - “Solve”
- Then we will write rigorous mathematical proofs of impossibility

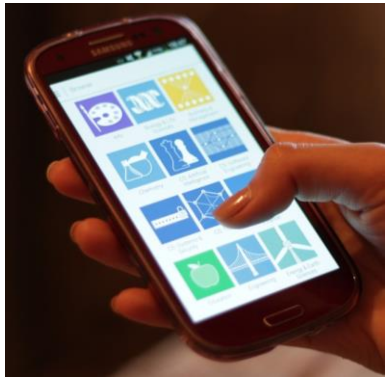
Which problems

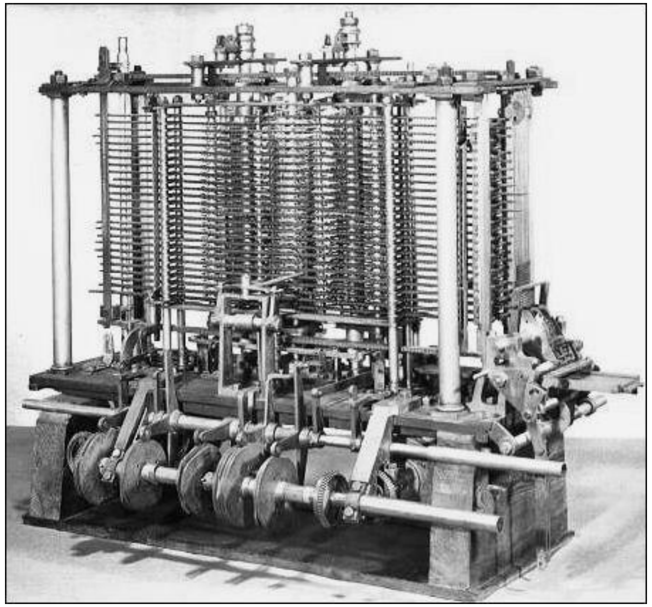
can be solved

through computation?

# Computation

- Computers: Modern technology?
- Computation is ancient
- Can be performed by:
  - A human being with paper and a pencil
  - A smartphone
  - A steam-powered machine

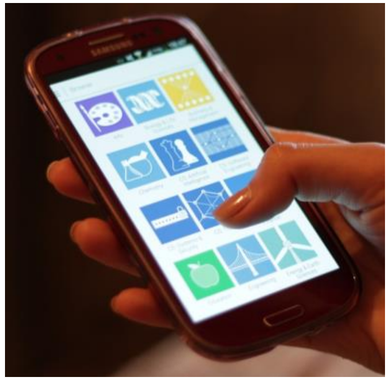




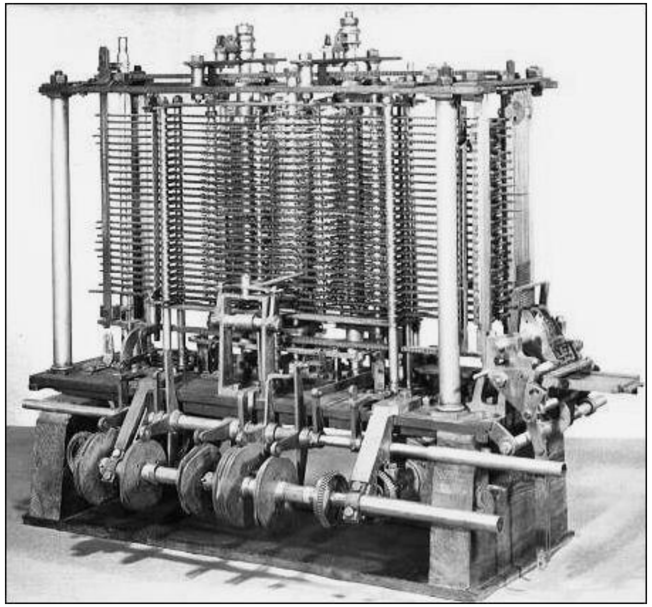
- We want a mathematical model that describes all of these and transcends any one technology

# Computation

- Note: Humans can do all the same computations that smartphones/laptops do
  - (less quickly and less reliably)

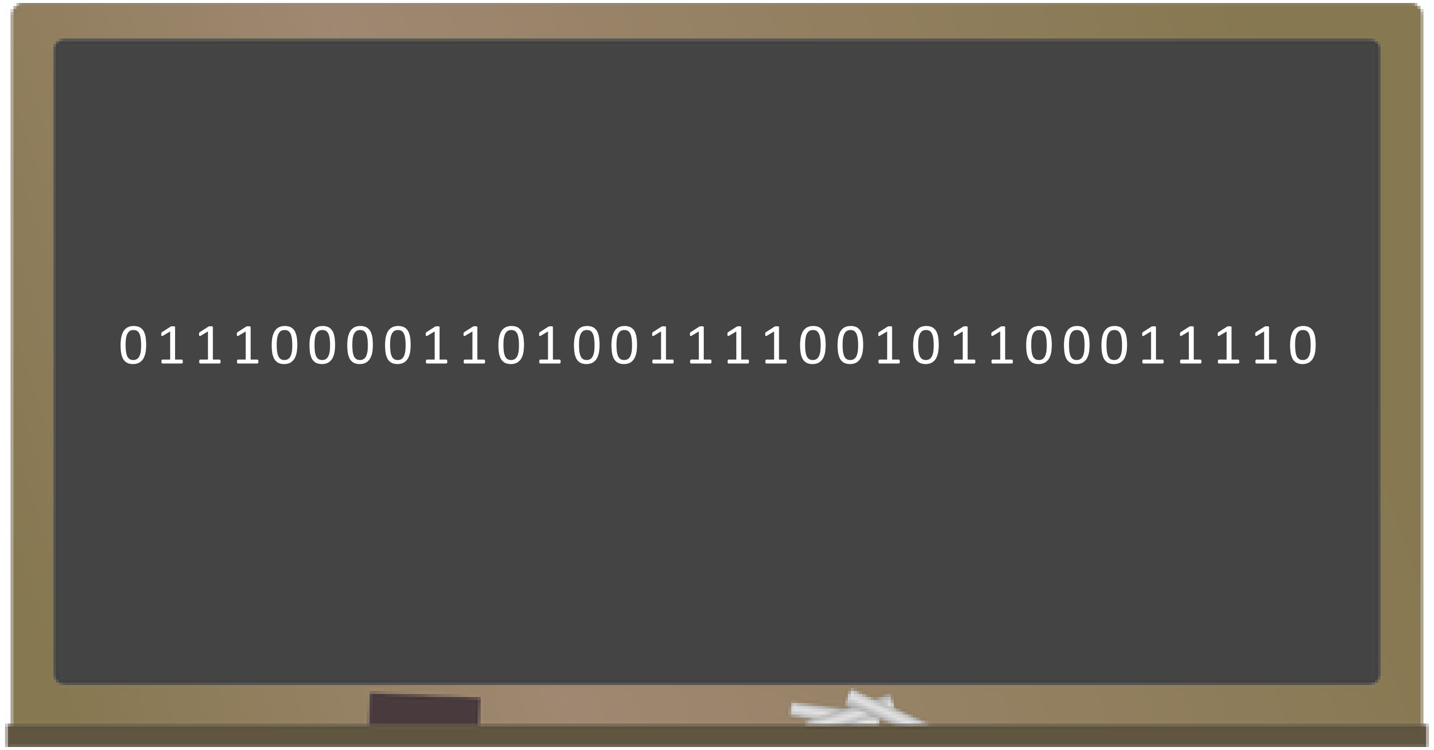


- Consequence: We can study computation without understanding electronics



- Computation is a familiar, everyday, human act

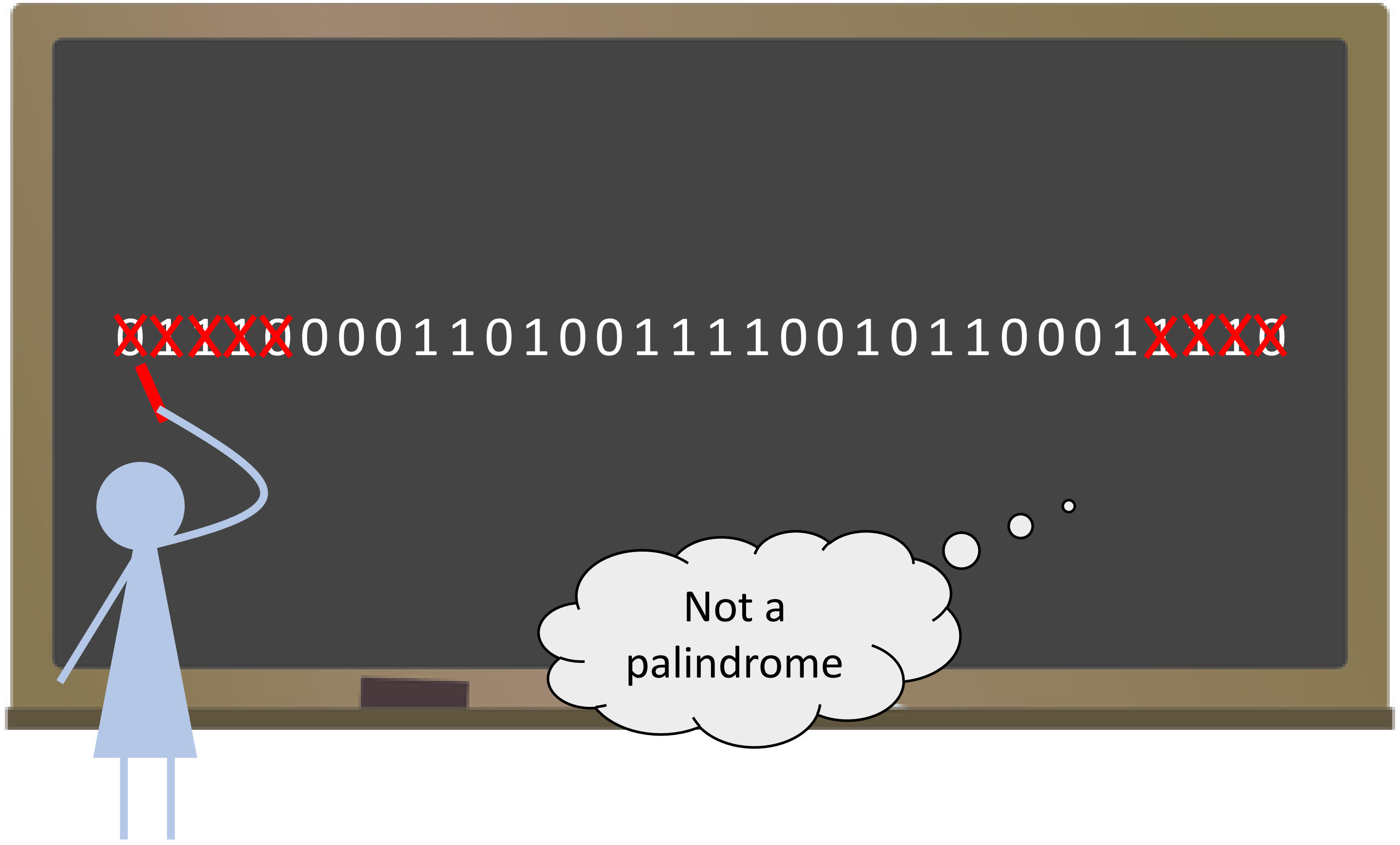
# Ex: Palindromes




- Suppose a long string of bits is written on a blackboard
- Our job: Figure out whether the string is a “palindrome,” i.e., whether it is the same forwards and backwards
- What should we do?

# Ex: Palindromes

- Idea: Compare and cross off the first and last symbols
- Repeat until we find a mismatch or everything is crossed off



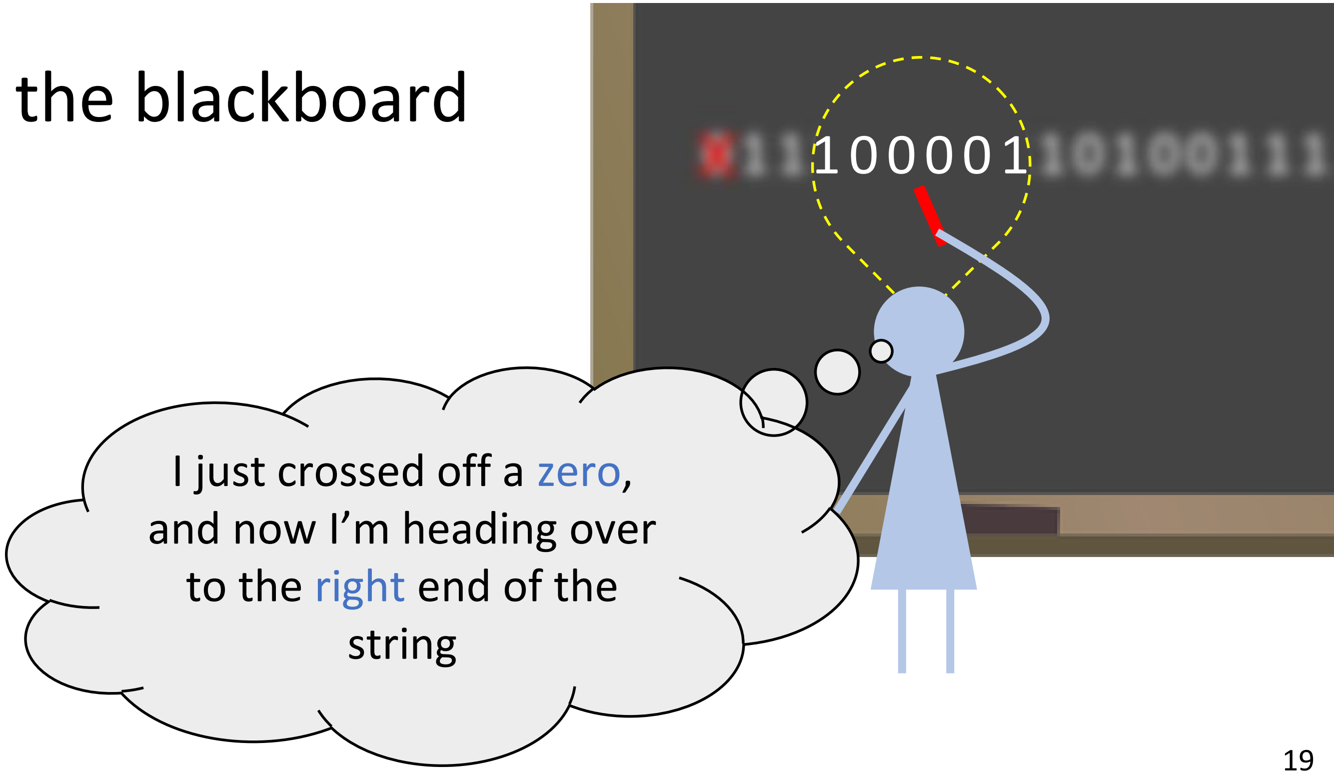
# Local decisions

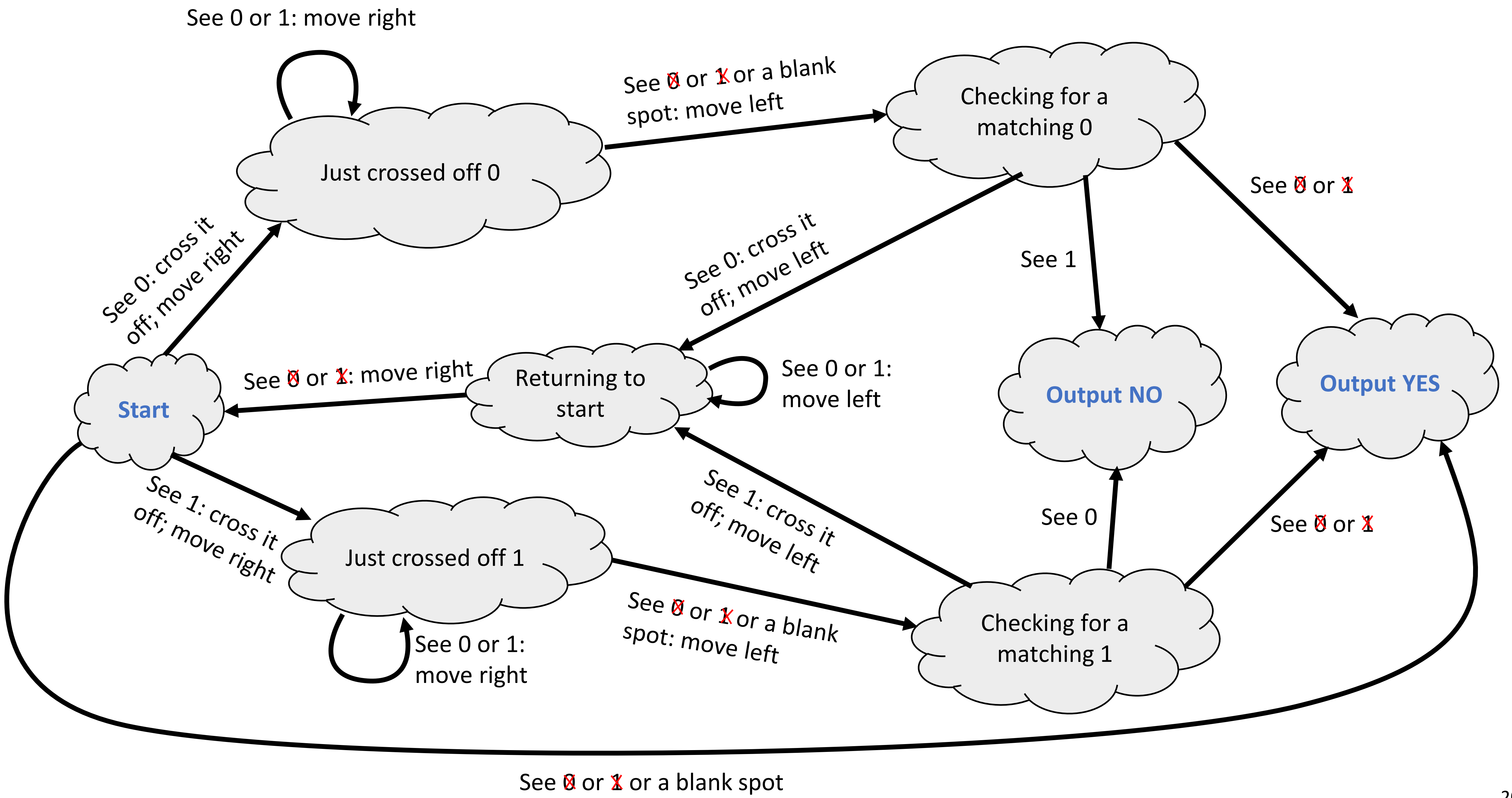
- In each step, how do we know what to do next?

1. We keep track of some information (“state”) in our mind
2. We look at the local contents of the blackboard

   (one symbol is sufficient)

- We can describe the algorithm using a “state diagram” (next slide)



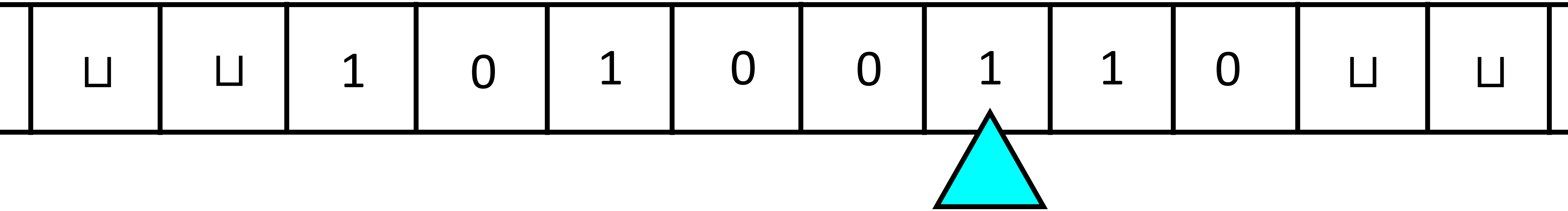


See 0 or 1: move right
Just crossed off 0
See ~~0~~ or ~~1~~ or a blank spot: move left
Checking for a matching 0
See ~~0~~ or ~~1~~
See 1
See 0: cross it off; move right
See 0: cross it off; move left
Start
See ~~0~~ or ~~1~~: move right
Returning to start
See 0 or 1: move left
Output NO
Output YES
See 1: cross it off; move right
See 1: cross it off; move left
See 0
See ~~0~~ or ~~1~~
Just crossed off 1
See 0 or 1: move right
See ~~0~~ or ~~1~~ or a blank spot: move left
Checking for a matching 1
See ~~0~~ or ~~1~~ or a blank spot

# The Turing machine model

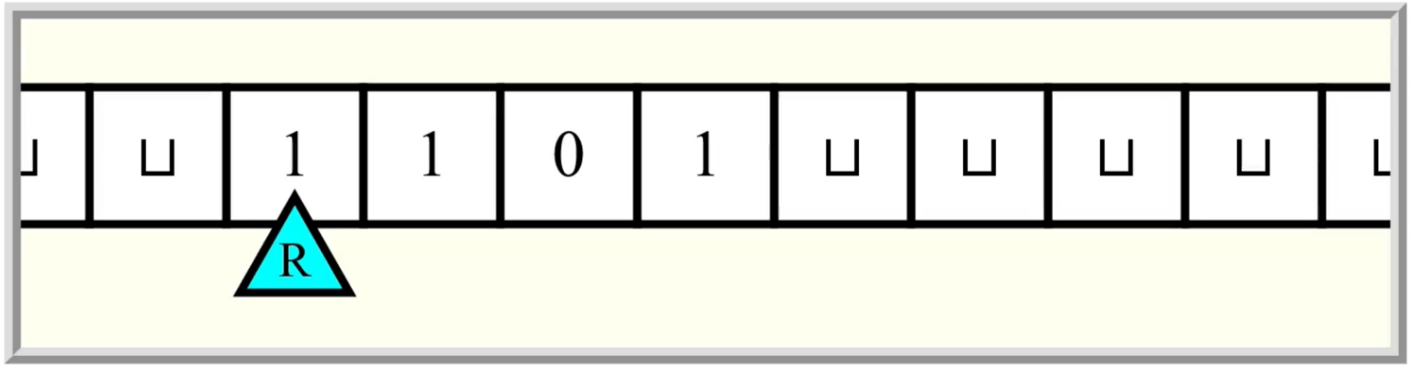
- Turing machines: A mathematical model of human computation
- In a nutshell, a Turing machine is any algorithm that can be described by a state diagram like the one we just saw

# The Turing machine model



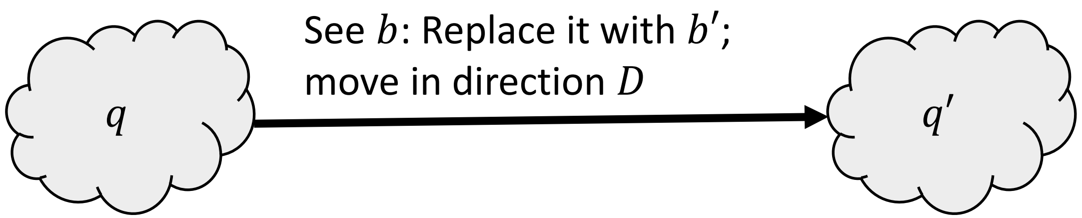

- We imagine an infinite, one-dimensional “tape”
- The tape is divided into “cells.” Each cell has one symbol written in it
- There is a “head” pointing at one cell of the tape
- The machine can be in one of finitely many internal “states”

# Turing machines




- In each step, the machine decides
  - What to write
  - Which direction to move the head (left or right)
  - The new state
- The decision is based only on the current state and the observed symbol

# Transition function




- Mathematically, we have a transition function

$$\delta: Q \times \Sigma \to Q \times \Sigma \times \{\mathrm{L}, \mathrm{R}\}$$

- Here $Q$ is the set of states and $\Sigma$ is the set of symbols

- $\delta(q, b) = (q', b', D)$ means:
  - If we are in the state $q$ and we read the symbol $b$…
  - Then our new state will be $q'$, we will write $b'$ (replacing $b$), and the head will move in direction $D$. $\quad$ ($\mathrm{L} =$ left, $\mathrm{R} =$ right)

# The input to a Turing machine

- One Turing machine represents one algorithm
- For us, the input to a Turing machine will always be a finite string of bits

# Symbols and alphabets

- An "alphabet" $\Sigma$ is any nonempty, finite set of "symbols"
  - $\Sigma = \{0, 1\}$
  - $\Sigma = \{0, 1, \cancel{0}, \cancel{1}\}$
  - $\Sigma = \{A, B, C, \ldots, Z\}$
  - $\Sigma = \{$ 😍, ⚾, 🐍, ⏰, 🍕 $\}$

# Strings

- Let $\Sigma$ be an alphabet
- A string over $\Sigma$ is a finite se

**If $|\Sigma| = m$, then what is $\left|\Sigma^0\right|$?**

A: $|\Sigma^0| = 0$

B: $|\Sigma^0| = m$

C: $|\Sigma^0| = 1$

D: $|\Sigma^0|$ is not well-defined

Respond at PollEv.com/whoza or text “whoza” to 22333

- The length of a string $x$ is the number of symbols, denoted $|x|$
- If $n$ is a nonnegative integer, then $\Sigma^n$ is the set of length-$n$ strings over $\Sigma$
- Example: If $\Sigma = \{0, 1\}$, then

$$\Sigma^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$$

# The empty string

- If $\Sigma$ is any alphabet, then $|\Sigma^0| = 1$
- There is one string of length zero, called the empty string
- We use $\epsilon$ to denote the empty string
  - Denoted `""` in popular programming languages
- $\Sigma^0 = \{\epsilon\}$

# Arbitrary-length strings

- Let $\Sigma$ be an alphabet
- We define $\Sigma^*$ to be the set of strings over $\Sigma$ of any finite length:

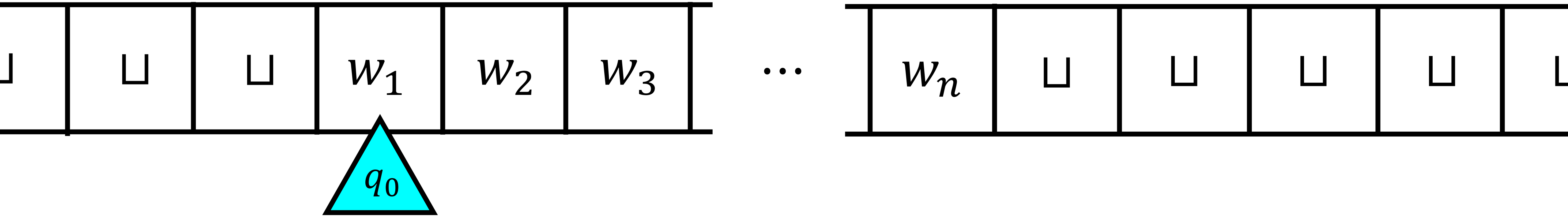
$$\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n$$

- Example: If $\Sigma = \{0, 1\}$, then

$$\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, \dots\}$$

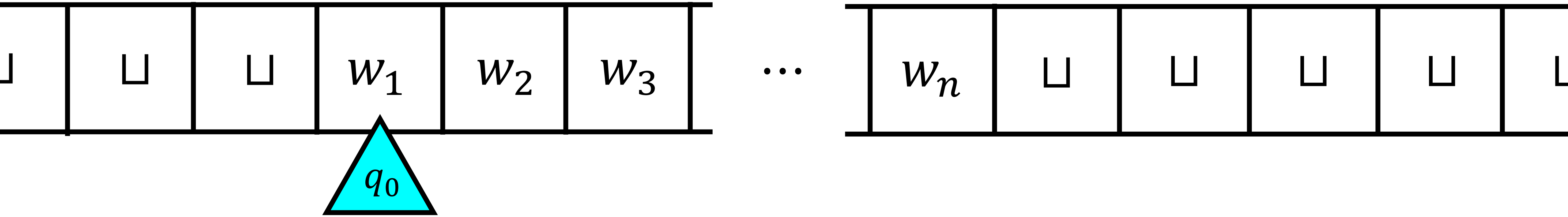
# Turing machine initialization

- The tape initially contains the input string $w \in \{0,1\}^*$ (one bit per cell)
- Each cell to the left or right of the input initially contains a special "blank symbol" ⊔

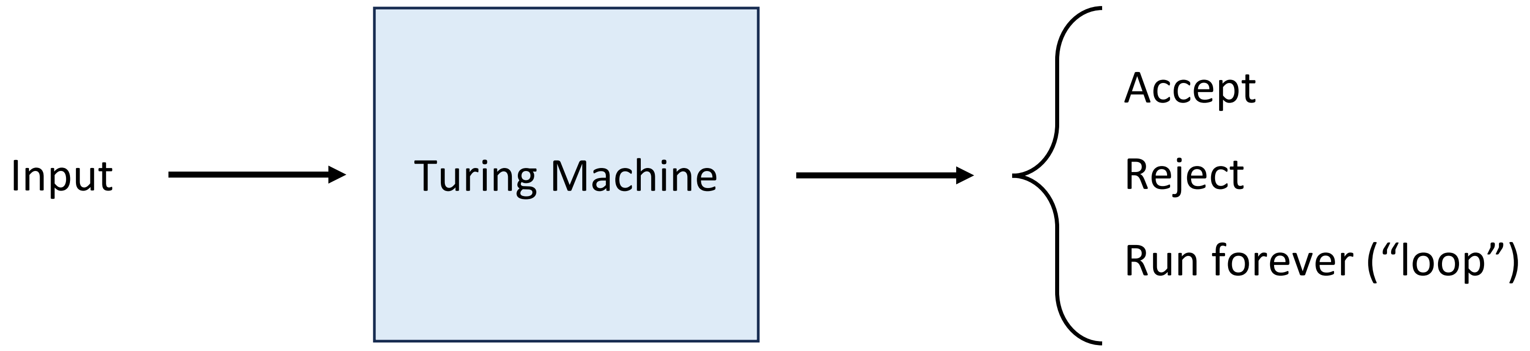


# Turing machine initialization

- The head is initially at the cell containing the first bit of the input
- The machine is initially in a special "start state" $q_0$

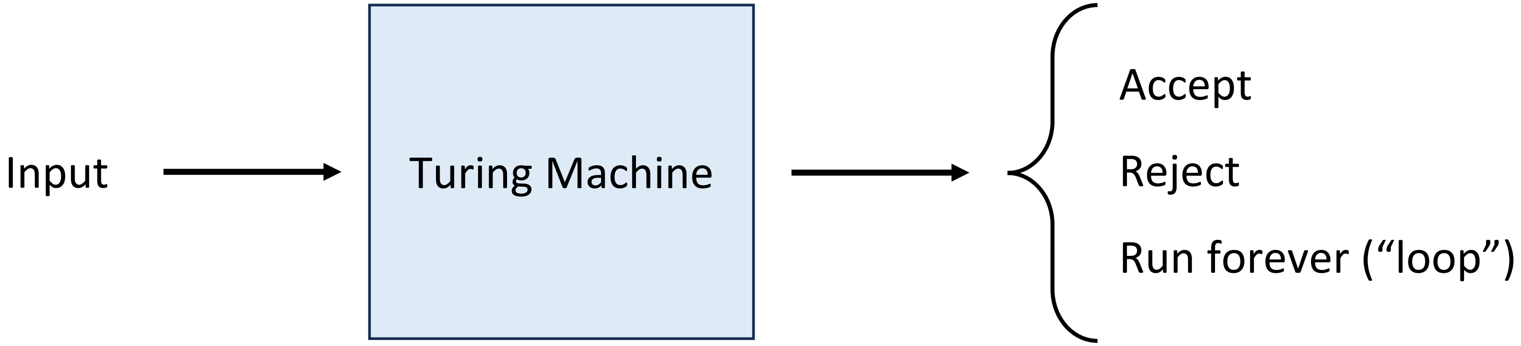


# Halting states




- There are two special “halting states,” $q_{\text{accept}}$ and $q_{\text{reject}}$
- If the machine ever reaches $q_{\text{accept}}$, this means it has accepted the input
- If the machine ever reaches $q_{\text{reject}}$, this means it has rejected the input
- Either way, the computation is finished. We say that the machine halts

# Looping




- It is also possible that the machine runs forever without ever reaching $q_{\text{accept}}$ or $q_{\text{reject}}$
- In this case, we say that the machine does not halt, does not accept the input, and does not reject the input

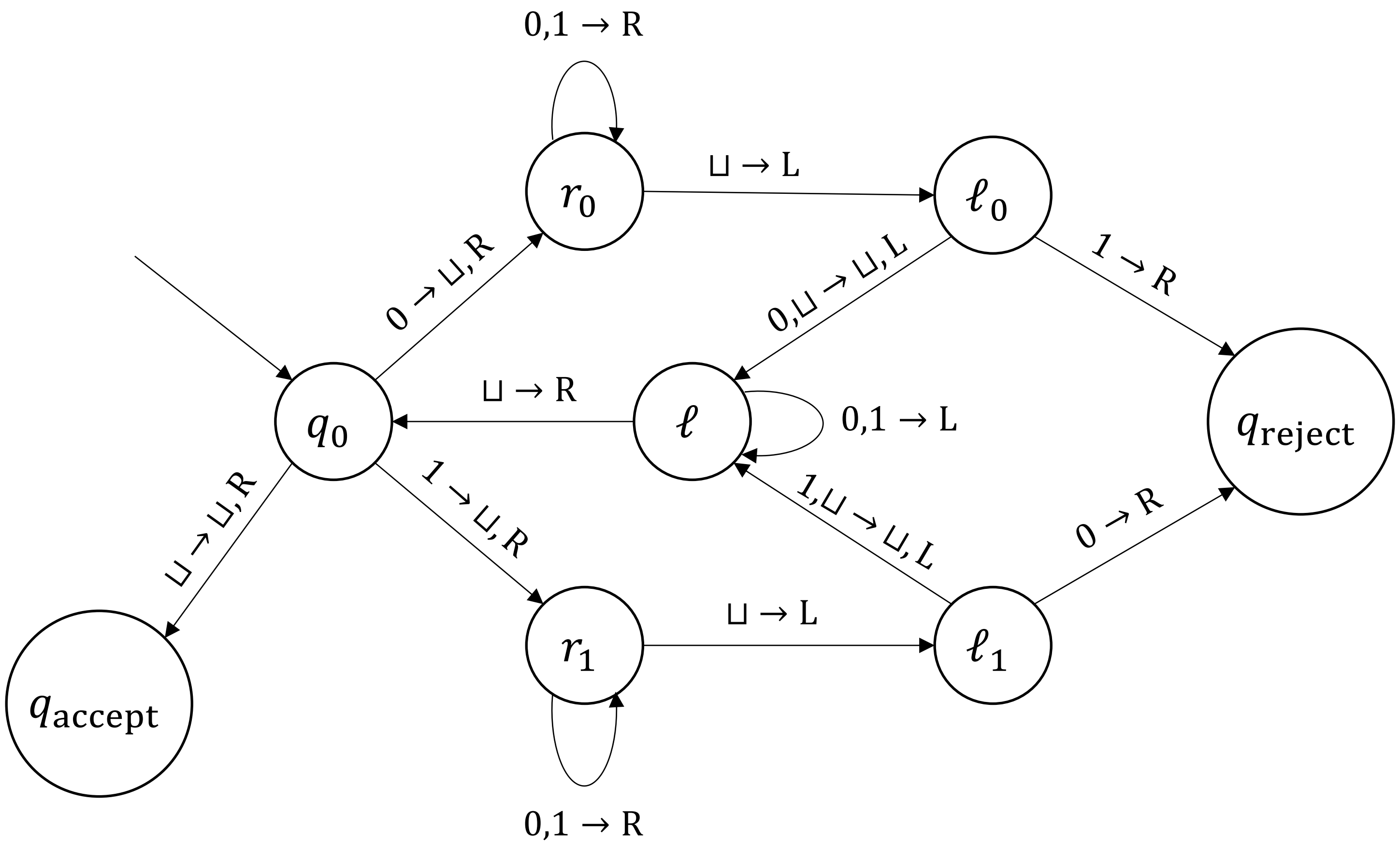
# Defining Turing machines rigorously

- **Definition**: A Turing machine is a 7-tuple $M = (Q, q_0, q_{\text{accept}}, q_{\text{reject}}, \Sigma, \sqcup, \delta)$ such that
  - $Q$ is a finite set (the set of "states")
  - $q_0, q_{\text{accept}}, q_{\text{reject}} \in Q$ and $q_{\text{accept}} \neq q_{\text{reject}}$
  - $\Sigma$ is a finite set of symbols (the "tape alphabet")
  - $\sqcup$ is a symbol (the "blank symbol")
  - $\{0, 1, \sqcup\} \subseteq \Sigma$ and $\sqcup \notin \{0, 1\}$
  - $\delta$ is a function $\delta: Q \times \Sigma \to Q \times \Sigma \times \{\text{L}, \text{R}\}$ (the "transition function")

⚠️ Warning: The definition in the textbook is slightly different. Sorry!

(The two models are equivalent.)

# State diagram



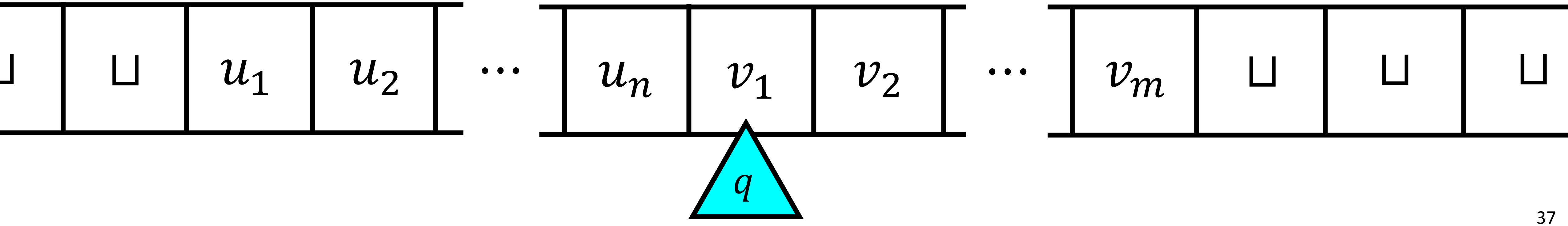

- Each node represents a state
- An arc from $q$ to $q'$ labeled "$b \to b', D$" means $\delta(q, b) = (q', b', D)$
- The label "$b \to D$" is shorthand for "$b \to b, D$"
- An arc labeled "$a, b \to \cdots$" represents two arcs ("$a \to \cdots$" and "$b \to \cdots$")

# Defining TM computation rigorously

- Transition function $\delta$ describes the local evolution of the computation
- What about the global evolution?

# Configurations of a Turing machine

- Let $M = (Q, q_0, q_{\text{accept}}, q_{\text{reject}}, \Sigma, \sqcup, \delta)$ be a Turing machine
- A configuration of $M$ is a triple $(u, q, v)$ where $u \in \Sigma^*$, $q \in Q$, $v \in \Sigma^*$, and $v \neq \epsilon$. Interpretation:
  - The tape currently contains $\cdots \sqcup\sqcup\sqcup\sqcup\ uv\ \sqcup\sqcup\sqcup\sqcup \cdots$
  - The machine is currently in state $q$ and the head is pointing at the first symbol of $v$



# Configuration shorthand

- Instead of $(u, q, v)$, we often write $uqv$
- We think of $uqv$ as a string over the alphabet $\Sigma \cup Q$
- This shorthand can only be used if $Q \cap \Sigma = \emptyset$, which we can assume without loss of generality by renaming states if necessary

# Equivalent configurations

- Note: $uqv$ and $uqv\ \sqcup$ are technically two distinct configurations…
- However, they represent the same scenario
- We can say that they are "equivalent"
- (A configuration is a finite string, even though the tape is infinitely long)
- Similarly, $\sqcup\ uqv$ is equivalent to $uqv$

# The initial configuration

- Let $w \in \{0, 1\}^*$ be an input
- The initial configuration of $M$ on $w$ is

$$\begin{cases} q_0 w & \text{if } w \neq \epsilon \\ q_0 \sqcup & \text{if } w = \epsilon \end{cases}$$

# The "next" configuration

- For any configuration $uqv$, we define $\mathrm{NEXT}(uqv)$ as follows:
  - Break $uqv$ into individual symbols: $uqv = u_1 u_2 \dots u_{n-1} u_n q v_1 v_2 v_3 \dots v_m$
  - If $\delta(q, v_1) = (q', b, \mathrm{R})$, then $\mathrm{NEXT}(uqv) = u_1 u_2 \dots u_{n-1} u_n b q' v_2 v_3 \dots v_m$
    - Edge case: If $m = 1$, then $\mathrm{NEXT}(uqv) = u_1 u_2 \dots u_{n-1} u_n b q' \sqcup$
  - If $\delta(q, v_1) = (q', b, \mathrm{L})$, then $\mathrm{NEXT}(uqv) = u_1 u_2 \dots u_{n-1} q' u_n b v_2 v_3 \dots v_m$
    - Edge case: If $n = 0$, then $\mathrm{NEXT}(uqv) = q' \sqcup b' v_2 v_3 \dots v_m$
- We write $\mathrm{NEXT}_M(uqv)$ if $M$ is not clear from context

# Halting configurations

- An accepting configuration is a configuration of the form $uq_{\mathrm{accept}}v$
- A rejecting configuration is a configuration of the form $uq_{\mathrm{reject}}v$
- A halting configuration is an accepting or rejecting configuration