Turing Machines

Part One

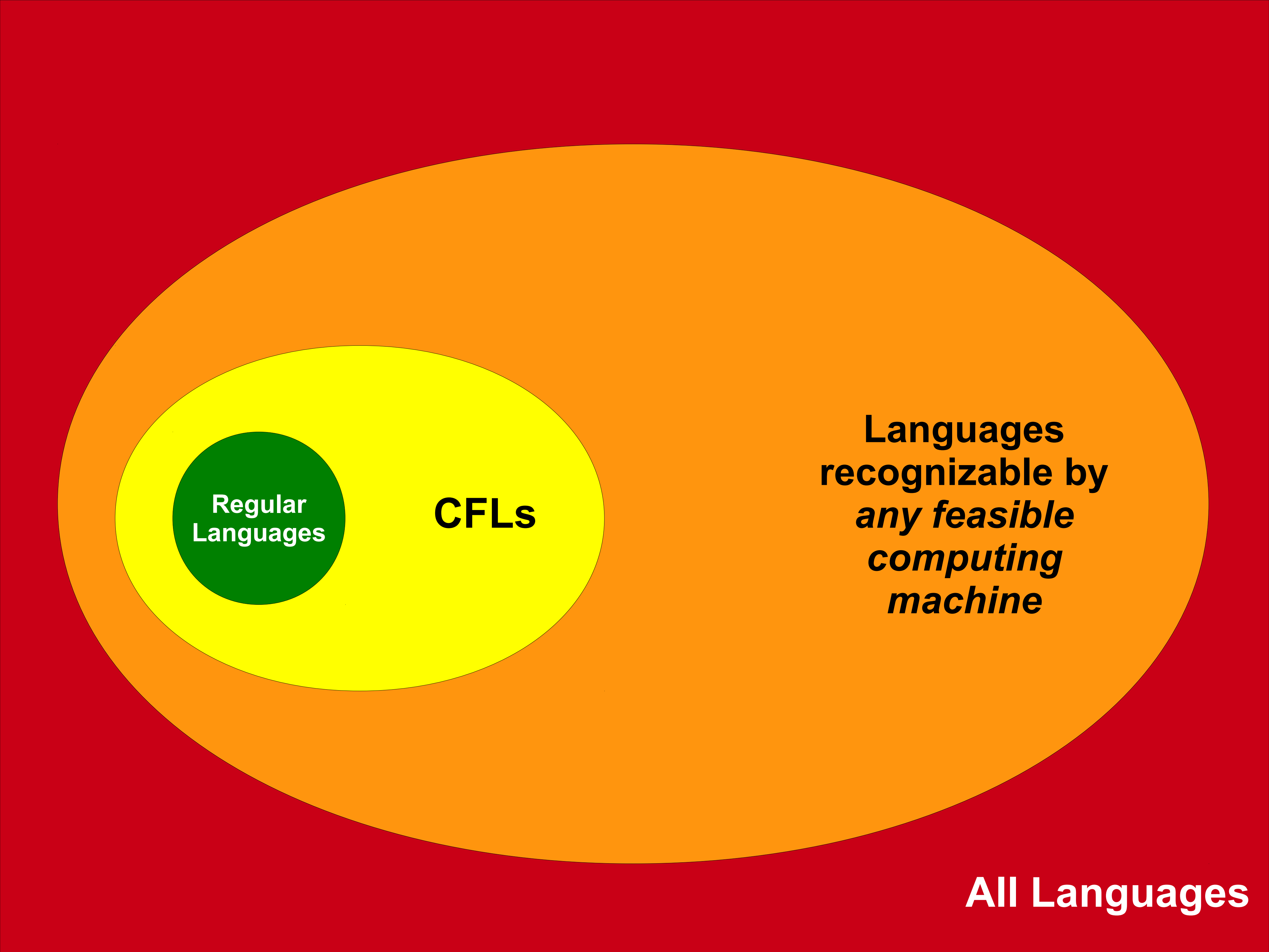
Hello Condensed Slide Readers!

Today's lecture consists almost exclusively of animations of Turing machines and TM constructions. We've presented a condensed version here, but we strongly recommend reading the full version of the slides today.

Hope this helps!

-Keith

What problems can we solve with a computer?


Regular Languages
CFLs
Languages recognizable by *any feasible computing machine*
All Languages

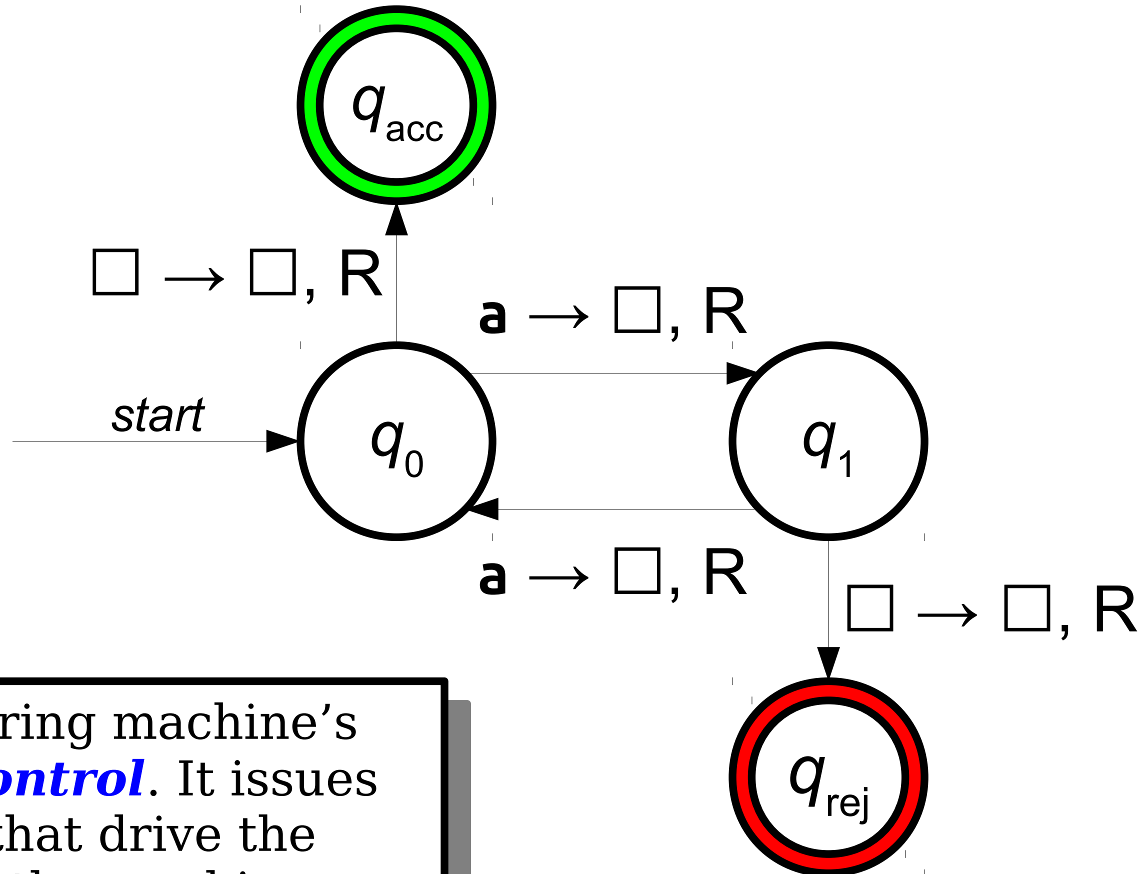
That same drawing, to scale.

The Problem

- Finite automata accept precisely the regular languages.
- We may need unbounded memory to recognize context-free languages.
  - e.g. $\{ \mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N} \}$ requires unbounded counting.
- How do we build an automaton with finitely many states but unbounded memory?

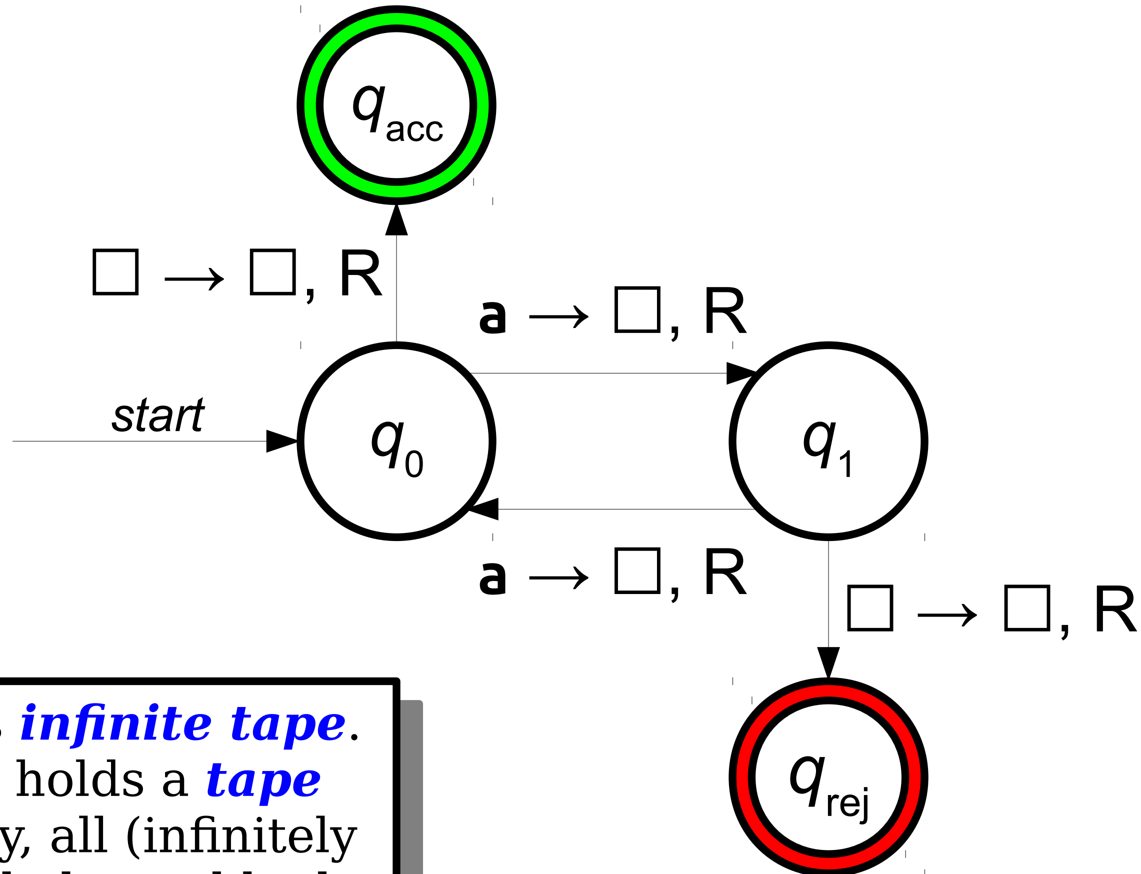
A Brief History Lesson

A Simple Turing Machine

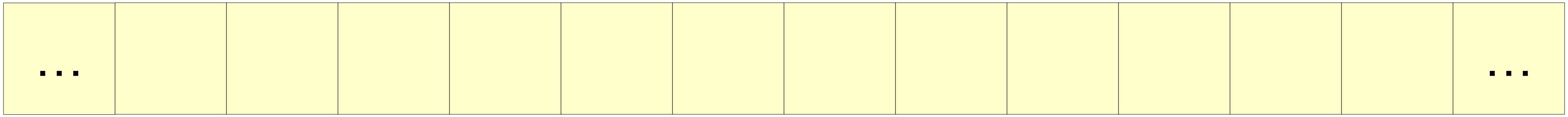


This is the Turing machine's ***finite state control***. It issues commands that drive the operation of the machine.

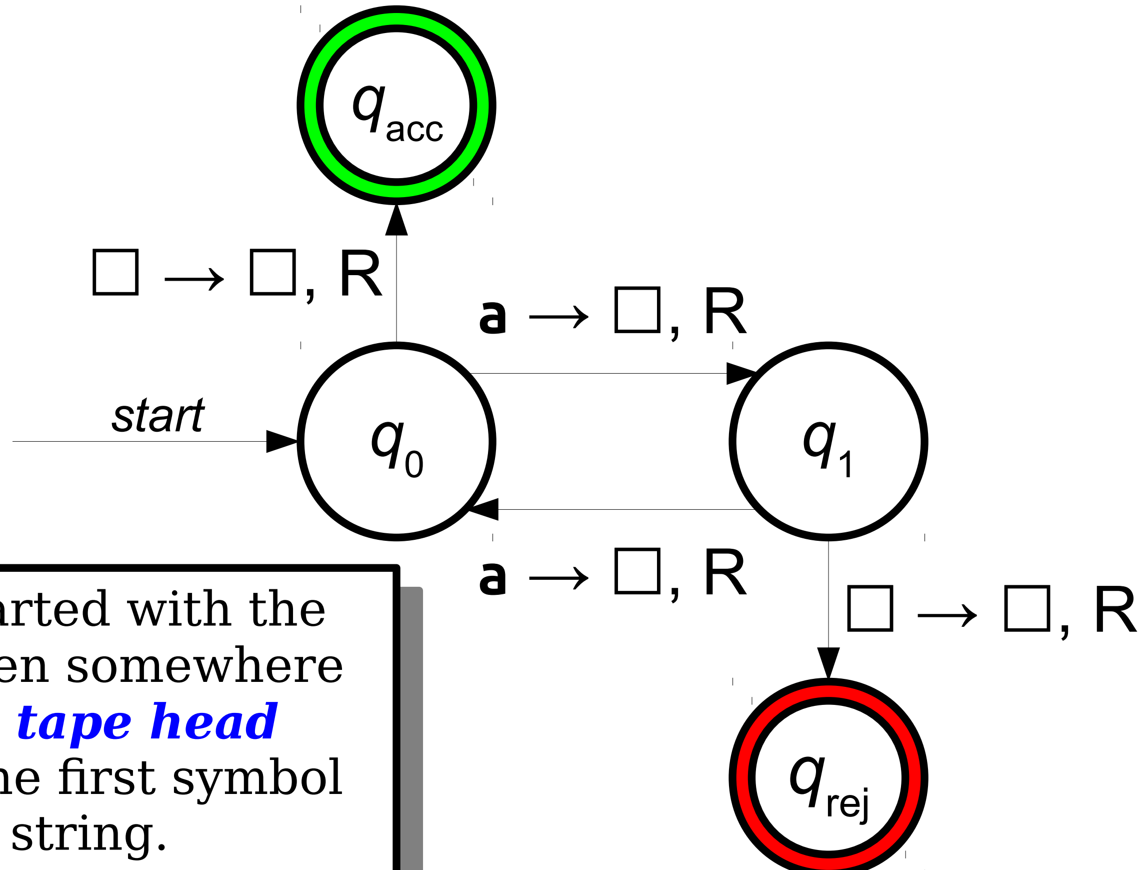
A Simple Turing Machine



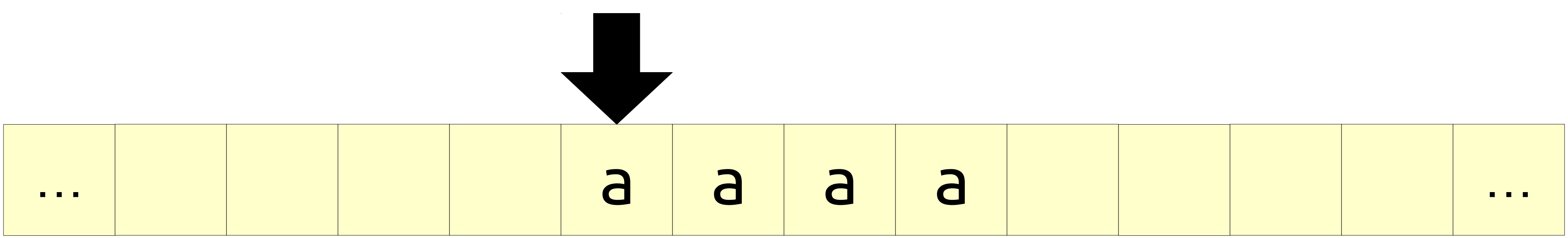
This is the TM's ***infinite tape***. Each tape cell holds a ***tape symbol***. Initially, all (infinitely many) tape symbols are blank.



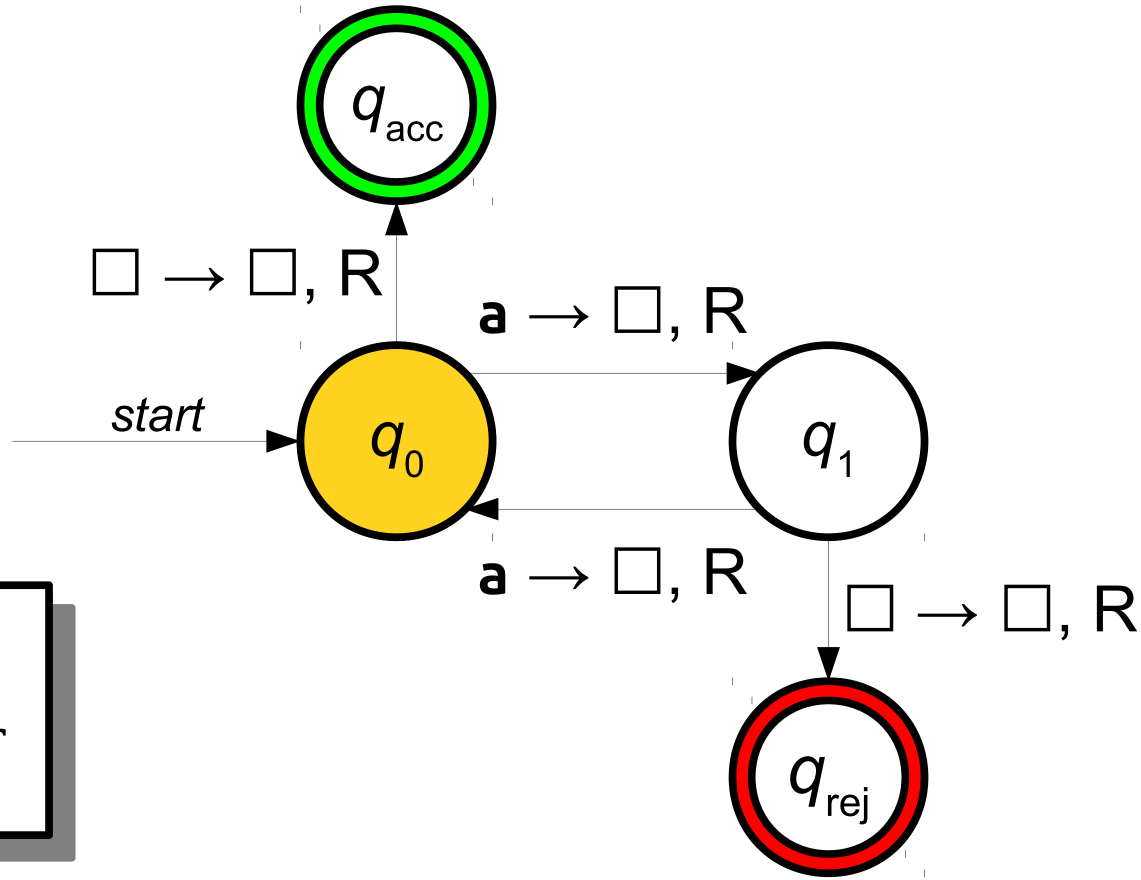
A Simple Turing Machine



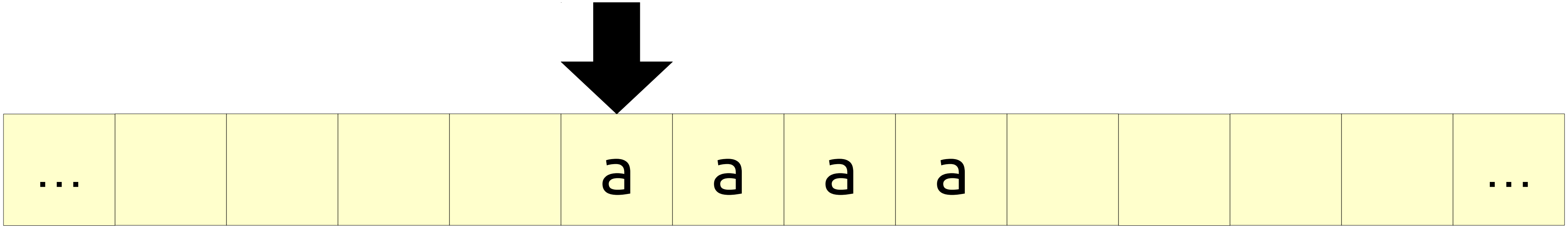
The machine is started with the ***input string*** written somewhere on the tape. The ***tape head*** initially points to the first symbol of the input string.



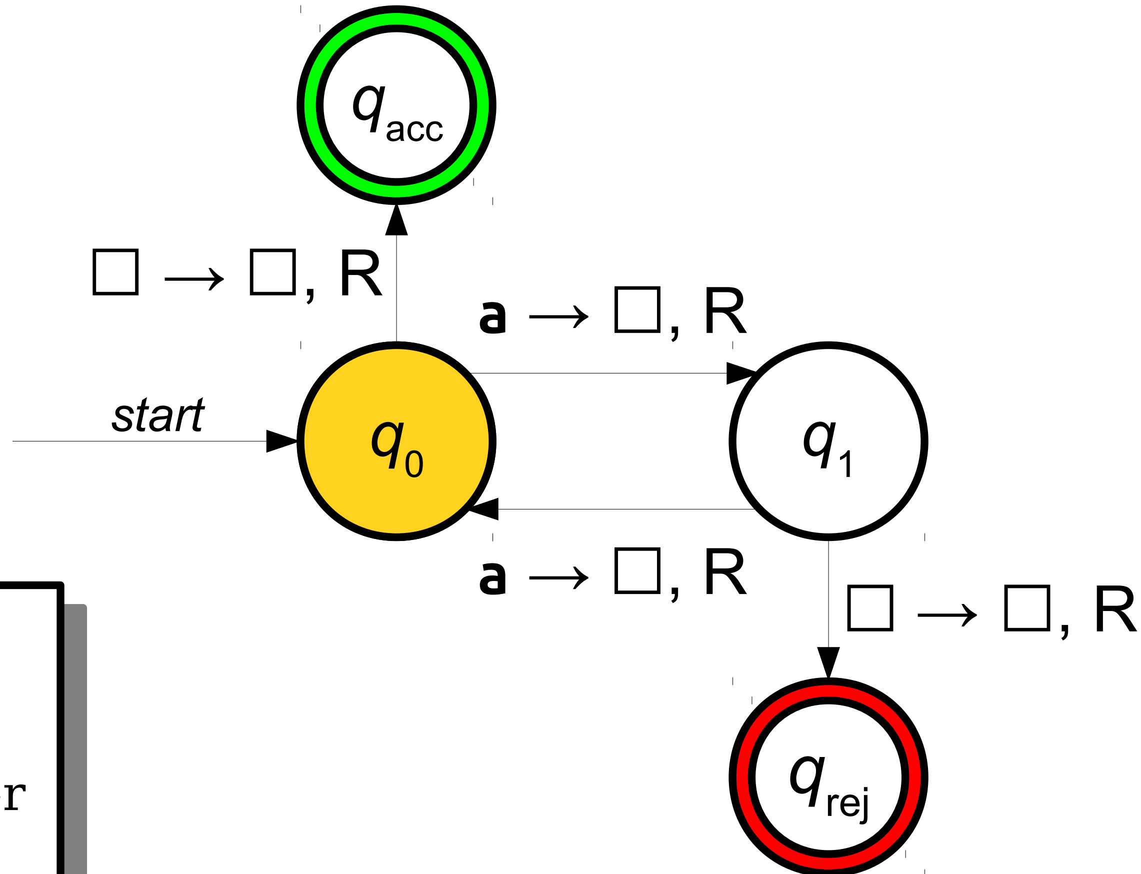
A Simple Turing Machine



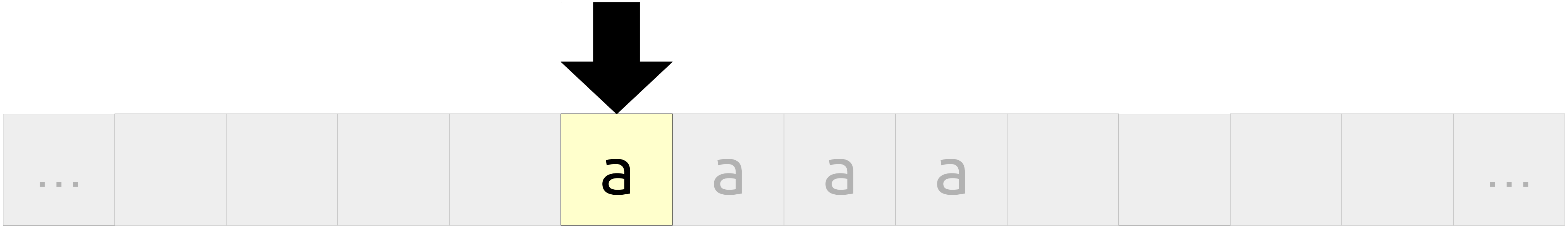
Like DFAs and NFAs, TMs begin execution in their ***start state***.



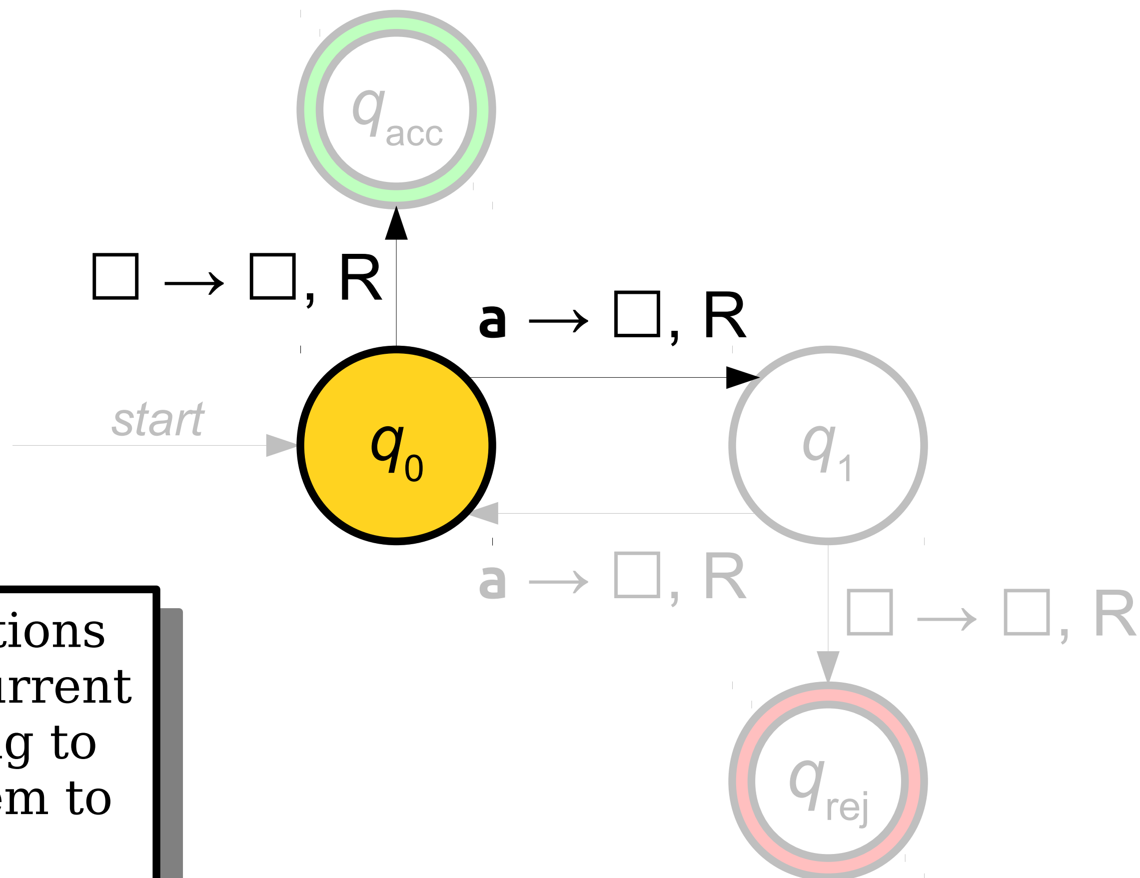
A Simple Turing Machine



At each step, the TM only looks at the symbol immediately under the ***tape head***.

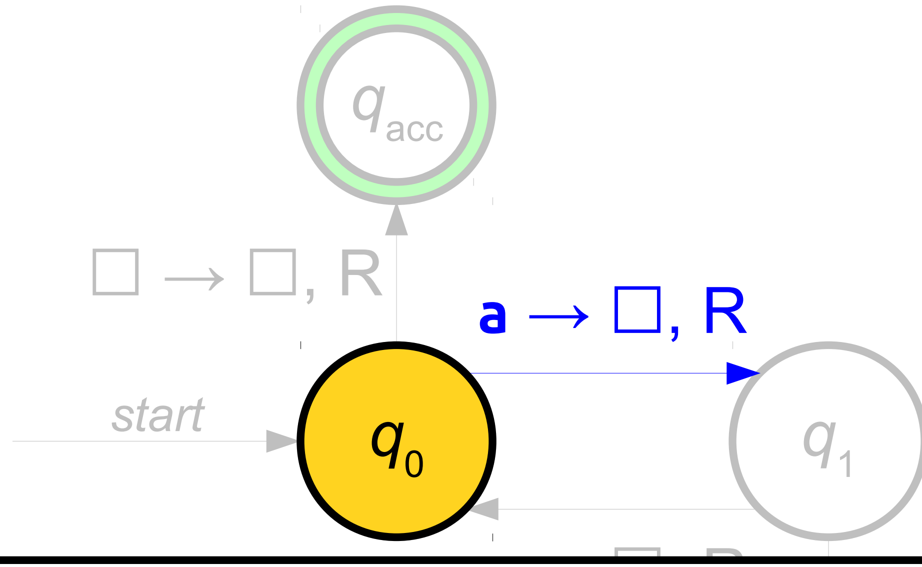


A Simple Turing Machine



These two transitions originate at the current state. We're going to choose one of them to follow.

A Simple Turing Machine

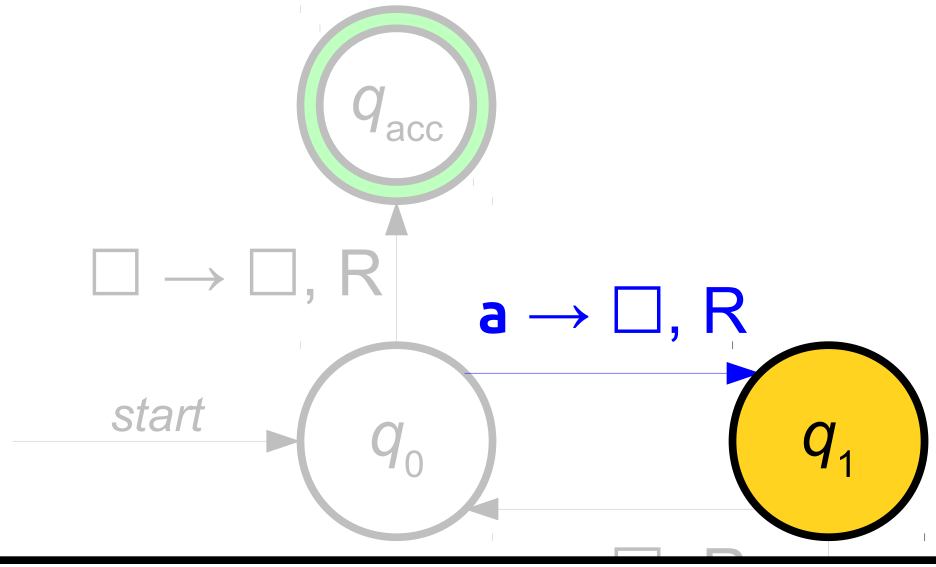


Each transition has the form

read* → *write*, *dir

and means "if symbol ***read*** is under the tape head, replace it with ***write*** and move the tape head in direction ***dir*** (L or R). The □ symbol denotes a blank cell.

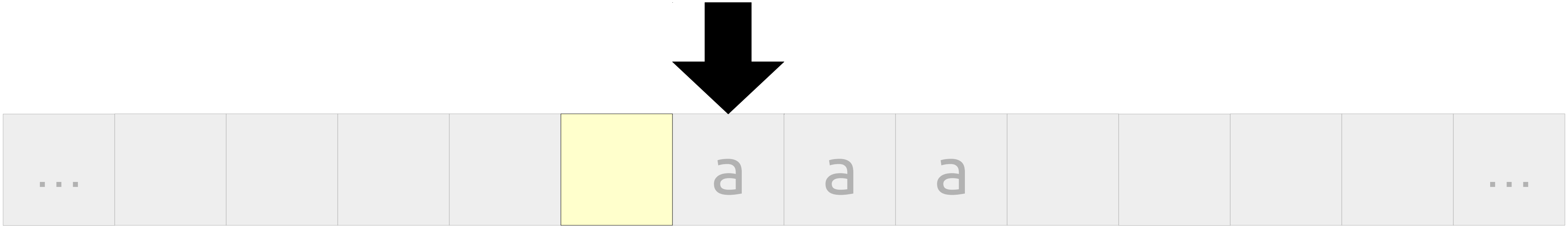
A Simple Turing Machine



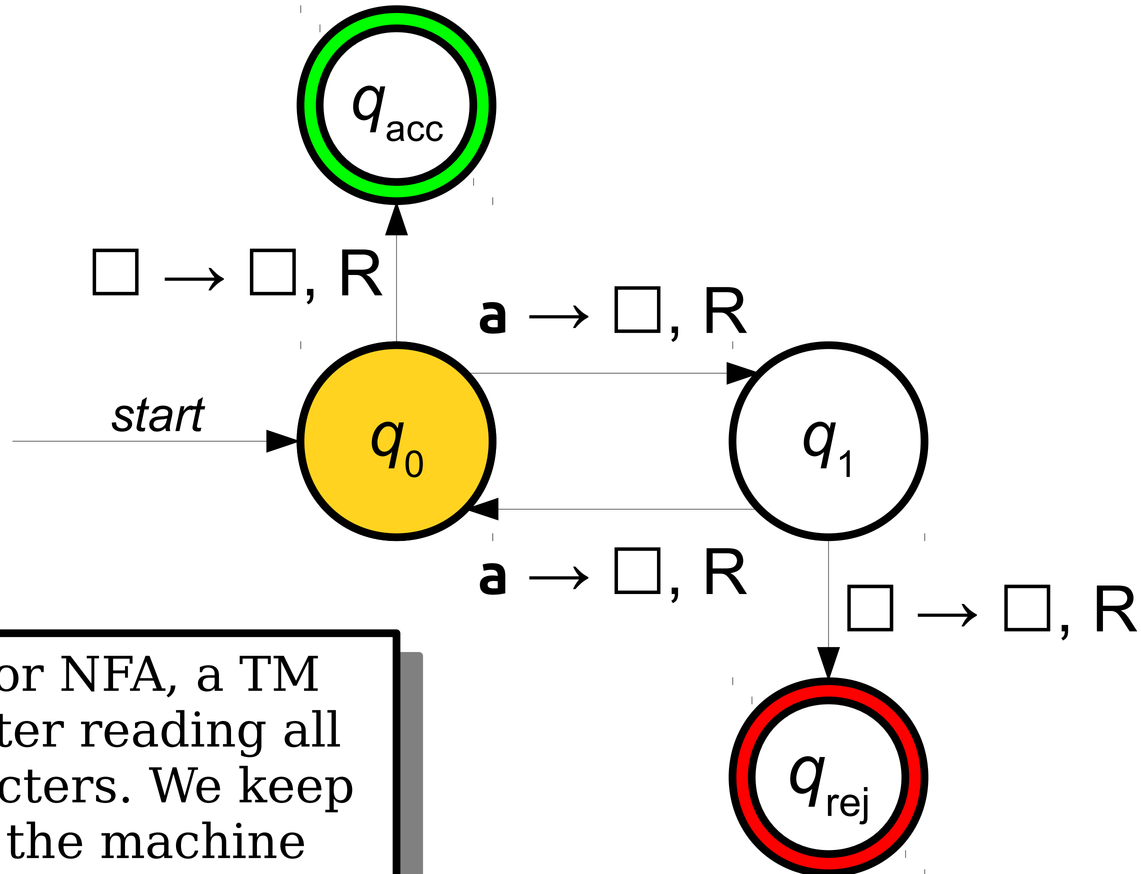
Each transition has the form

read* → *write*, *dir

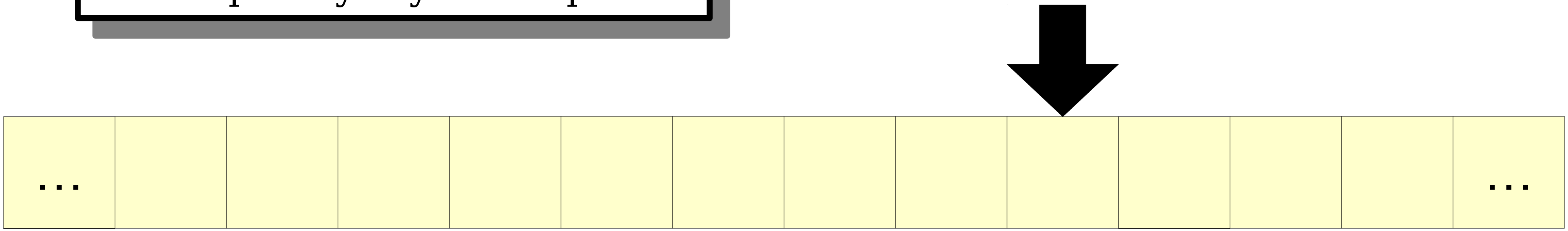
and means "if symbol ***read*** is under the tape head, replace it with ***write*** and move the tape head in direction ***dir*** (L or R). The □ symbol denotes a blank cell.



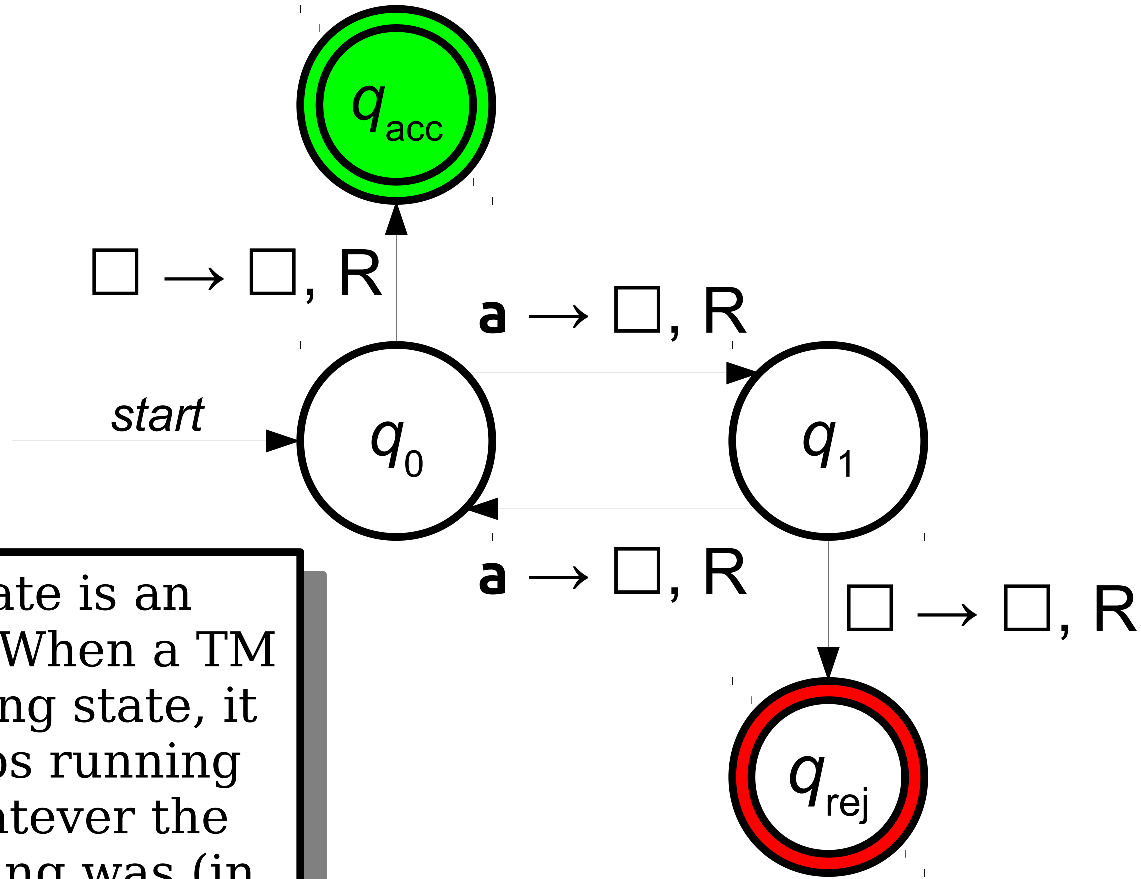
A Simple Turing Machine



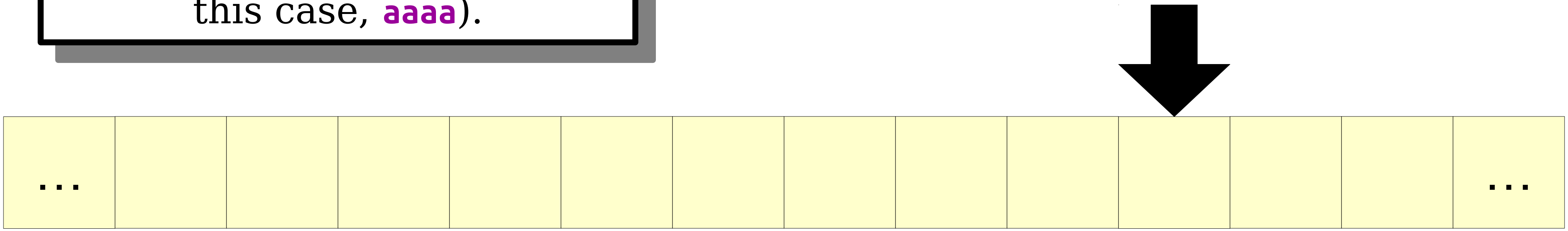
Unlike a DFA or NFA, a TM doesn't stop after reading all the input characters. We keep running until the machine explicitly says to stop.



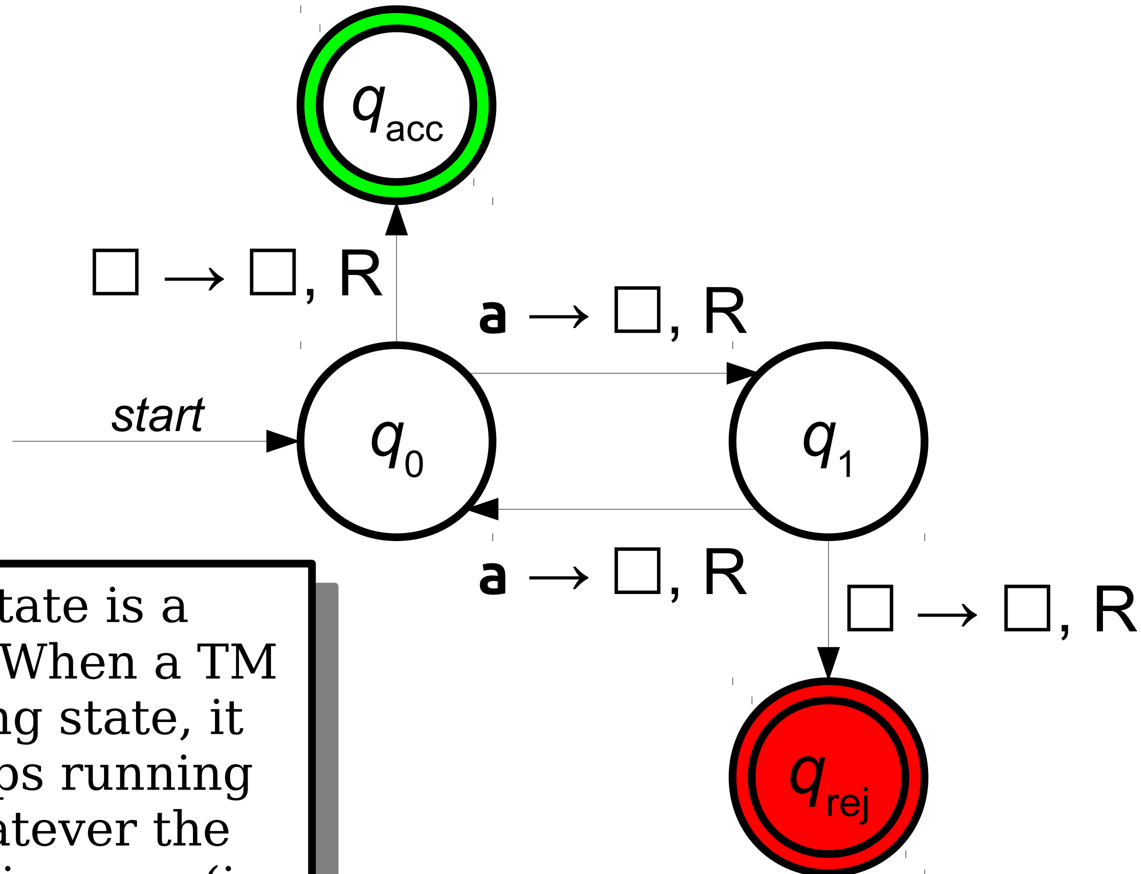
A Simple Turing Machine



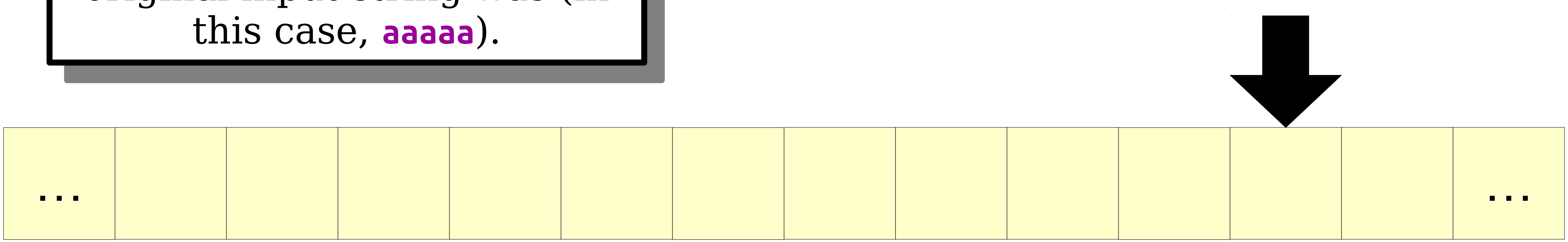
This special state is an ***accepting state***. When a TM enters an accepting state, it *immediately* stops running and accepts whatever the original input string was (in this case, **aaaa**).



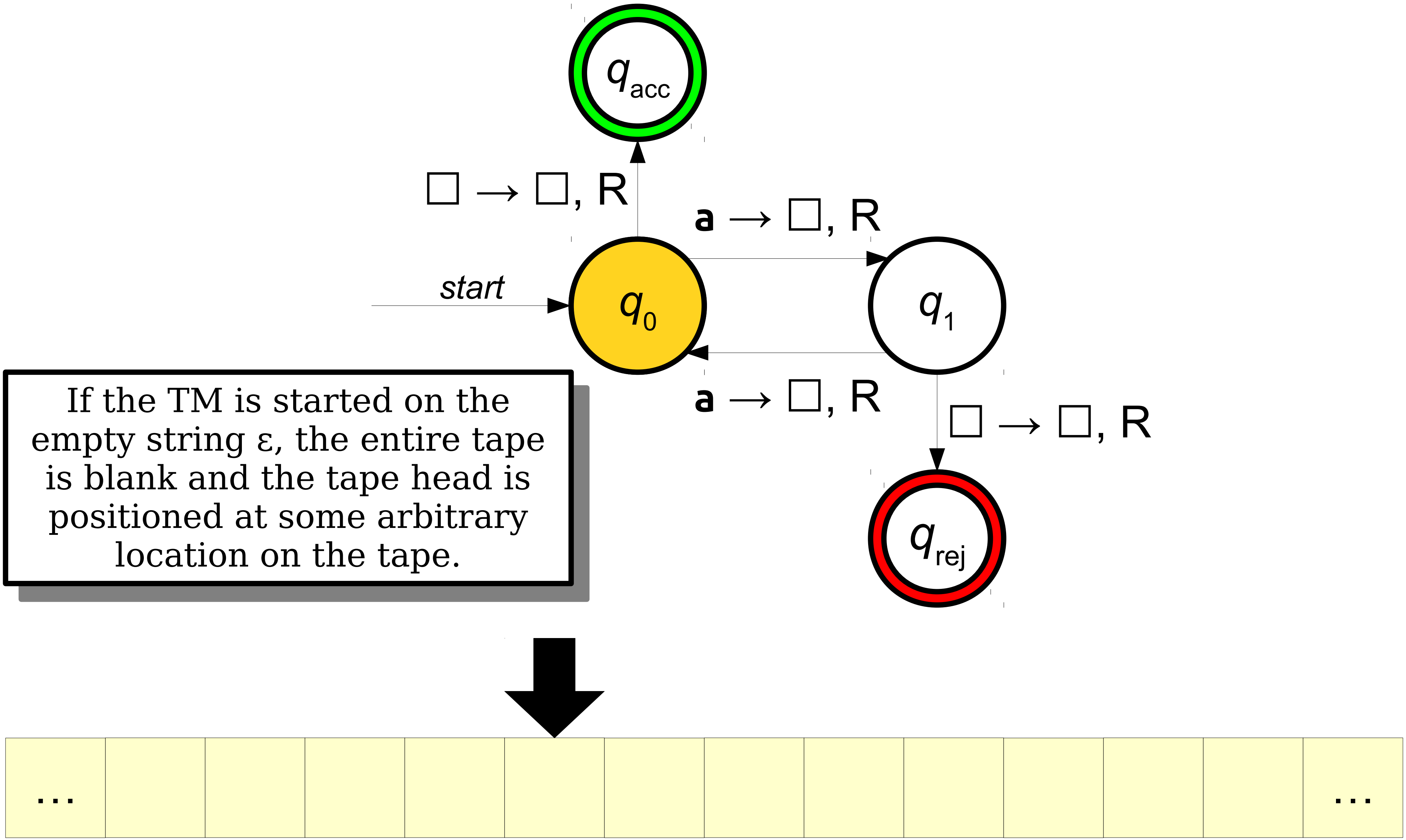
A Simple Turing Machine



This special state is a ***rejecting state***. When a TM enters a rejecting state, it *immediately* stops running and rejects whatever the original input string was (in this case, **aaaaa**).



A Simple Turing Machine



If the TM is started on the empty string ε, the entire tape is blank and the tape head is positioned at some arbitrary location on the tape.

The Turing Machine

- A Turing machine consists of three parts:
  - A ***finite-state control*** that issues commands,
  - an ***infinite tape*** for input and scratch space, and
  - a ***tape head*** that can read and write a single tape cell.
- At each step, the Turing machine
  - writes a symbol to the tape cell under the tape head,
  - changes state, and
  - moves the tape head to the left or to the right.

Input and Tape Alphabets

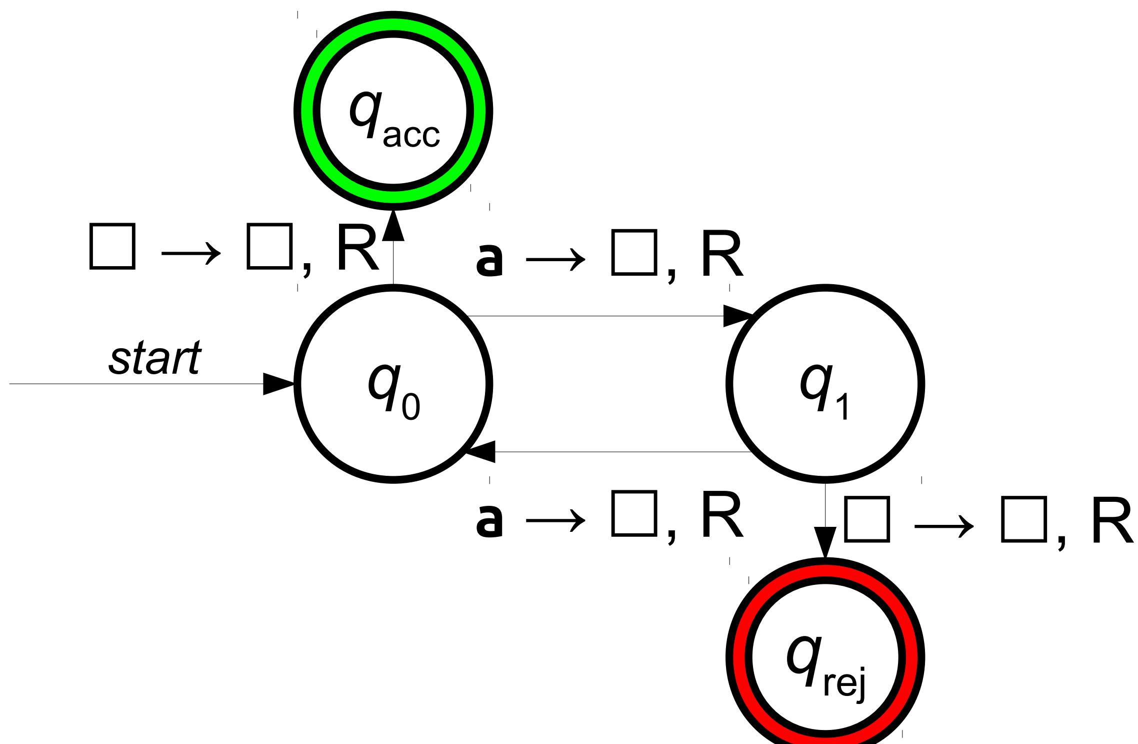
- A Turing machine has two alphabets:
  - An ***input alphabet*** Σ. All input strings are written in the input alphabet.
  - A ***tape alphabet*** Γ, where $\Sigma \subsetneq \Gamma$. The tape alphabet contains all symbols that can be written onto the tape.
- The tape alphabet Γ can contain any number of symbols, but always contains at least one ***blank symbol***, denoted $\Box$. You are guaranteed $\Box \notin \Sigma$.
- At startup, the Turing machine begins with an infinite tape of $\Box$ symbols with the input written at some location. The tape head is positioned at the start of the input.

Accepting and Rejecting States

- Unlike DFAs, Turing machines do not stop processing the input when they finish reading it.
- Turing machines decide when (and if!) they will accept or reject their input.
- Turing machines can enter infinite loops and never accept or reject; more on that later…

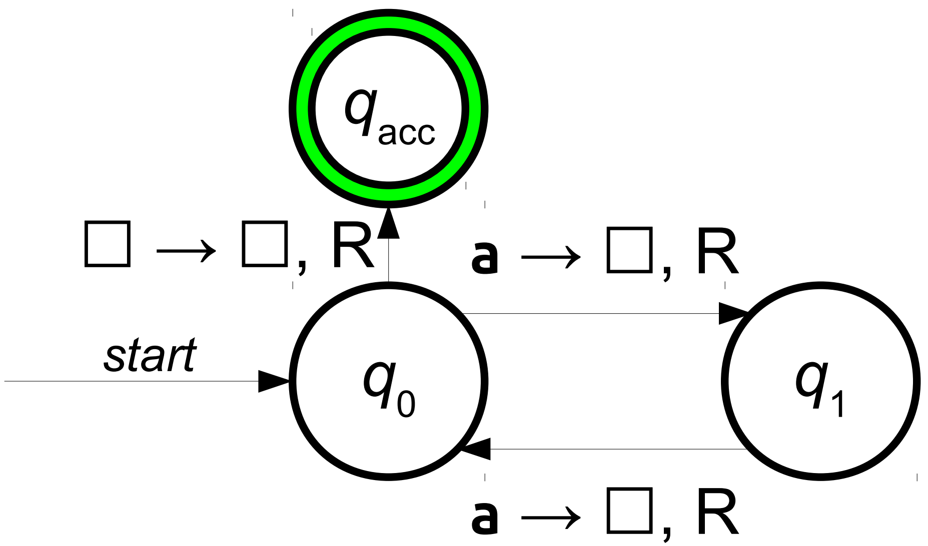
Determinism

- Turing machines are ***deterministic***: for every combination of a (non-accepting, non-rejecting) state q and a tape symbol $a \in \Gamma$, there must be exactly one transition defined for that combination of q and a.
- Any transitions that are missing implicitly go straight to a rejecting state. We'll use this later to simplify our designs.

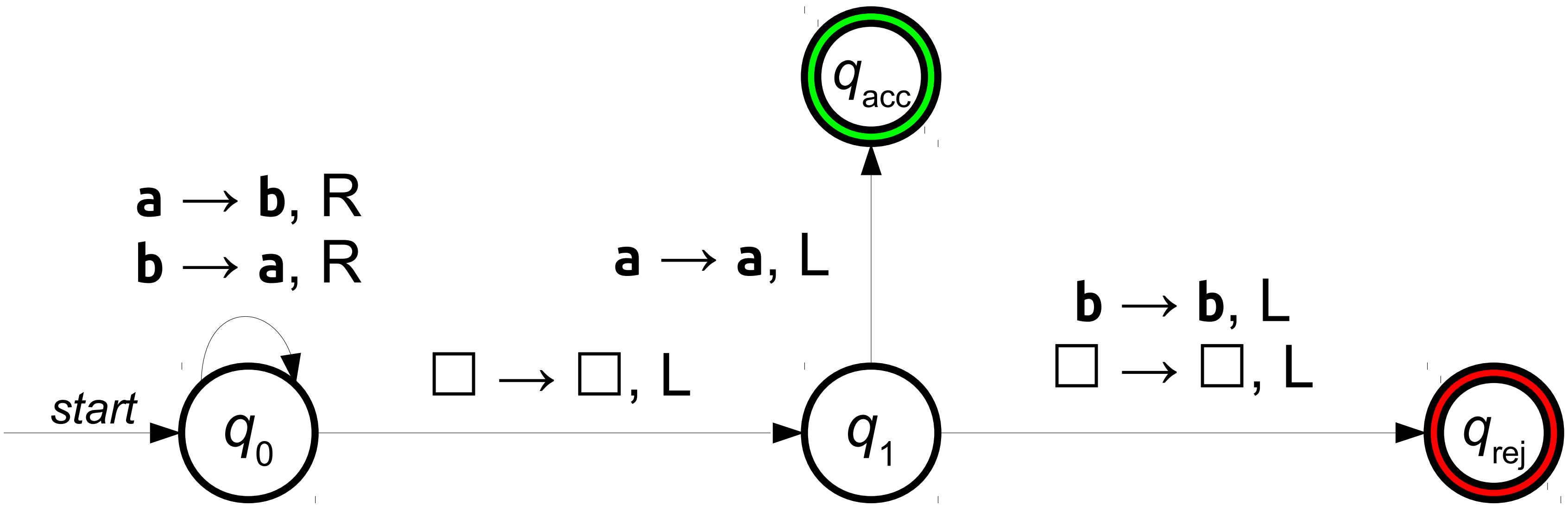


Determinism

- Turing machines are ***deterministic***: for every combination of a (non-accepting, non-rejecting) state q and a tape symbol $a \in \Gamma$, there must be exactly one transition defined for that combination of q and a.
- Any transitions that are missing implicitly go straight to a rejecting state. We'll use this later to simplify our designs.



This machine is exactly the same as the previous one.



Run the TM shown above on the input string **bba**.
What will the tape look like when the TM finishes running?

A.

…		b	b	a		…

B.

…		a	a	b		…

C.

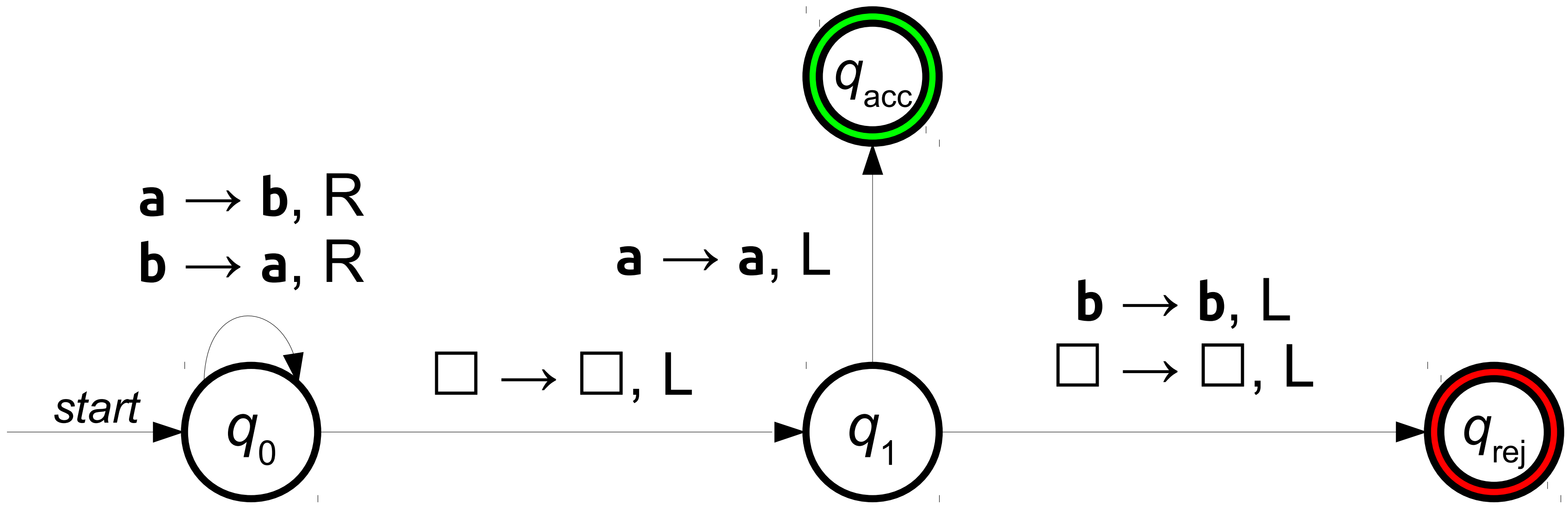
…		b	b	a		…

D.

…		a	a	b		…

E. None of these, or two or more of these.

Answer at **PollEv.com/cs103** or
text **CS103** to **22333** once to join, then ***A***, ***B***, ***C***, ***D***, or ***E***.



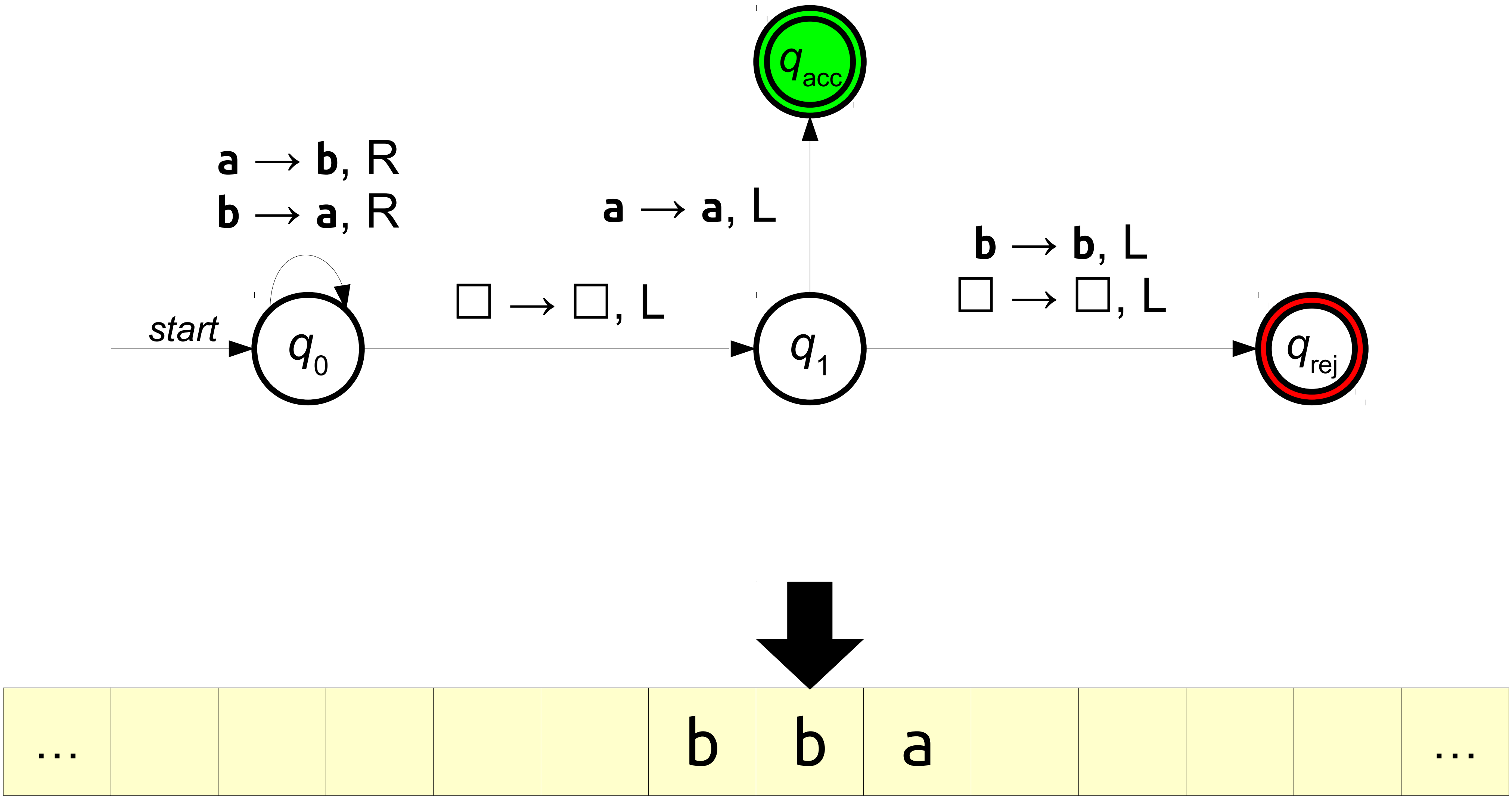
If M is a Turing machine with input alphabet Σ, then the ***language of M***, denoted $\mathscr{L}(M)$, is the set

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

Let M be the above TM, and assume its input alphabet is {**a**, **b**}. What is $\mathscr{L}(M)$?

A. { w ∈ {**a**, **b**}* | w ends in **a** }
B. { w ∈ {**a**, **b**}* | w ends in **b** }
C. Ø
D. None of these, or two or more of these.

Answer at **PollEv.com/cs103** or text **CS103** to **22333** once to join, then ***A***, ***B***, ***C***, or ***D***.



Although the tape ends with **bba** written on it, the original input string was **aab**. This shows that the TM accepts **aab**, not **bba**.

So $\mathscr{L}(M)$ = { w ∈ {**a**, **b**}* | w ends in **b** }

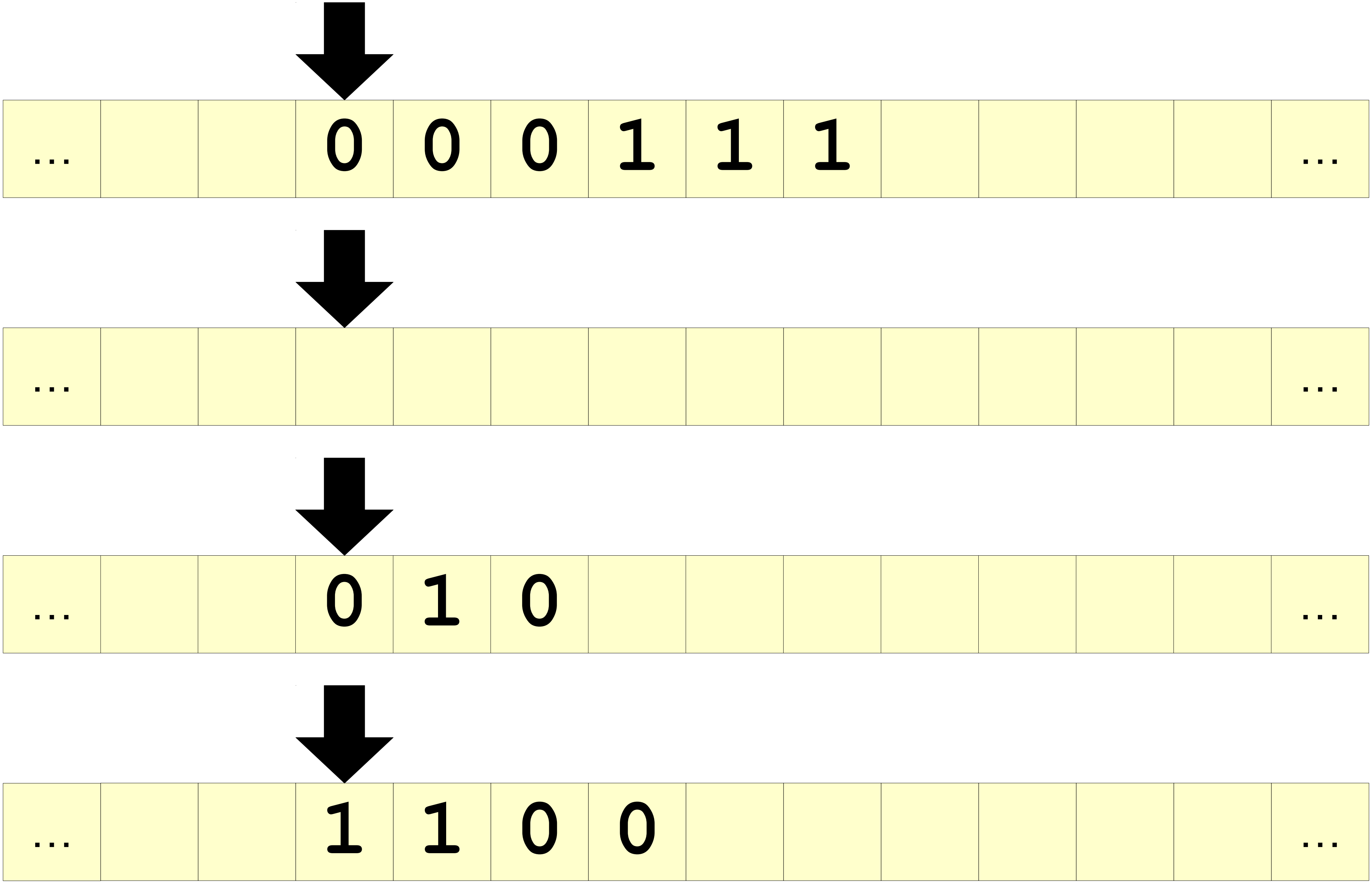
Designing Turing Machines

- Despite their simplicity, Turing machines are very powerful computing devices.
- Today's lecture explores how to design Turing machines for various languages.

Designing Turing Machines

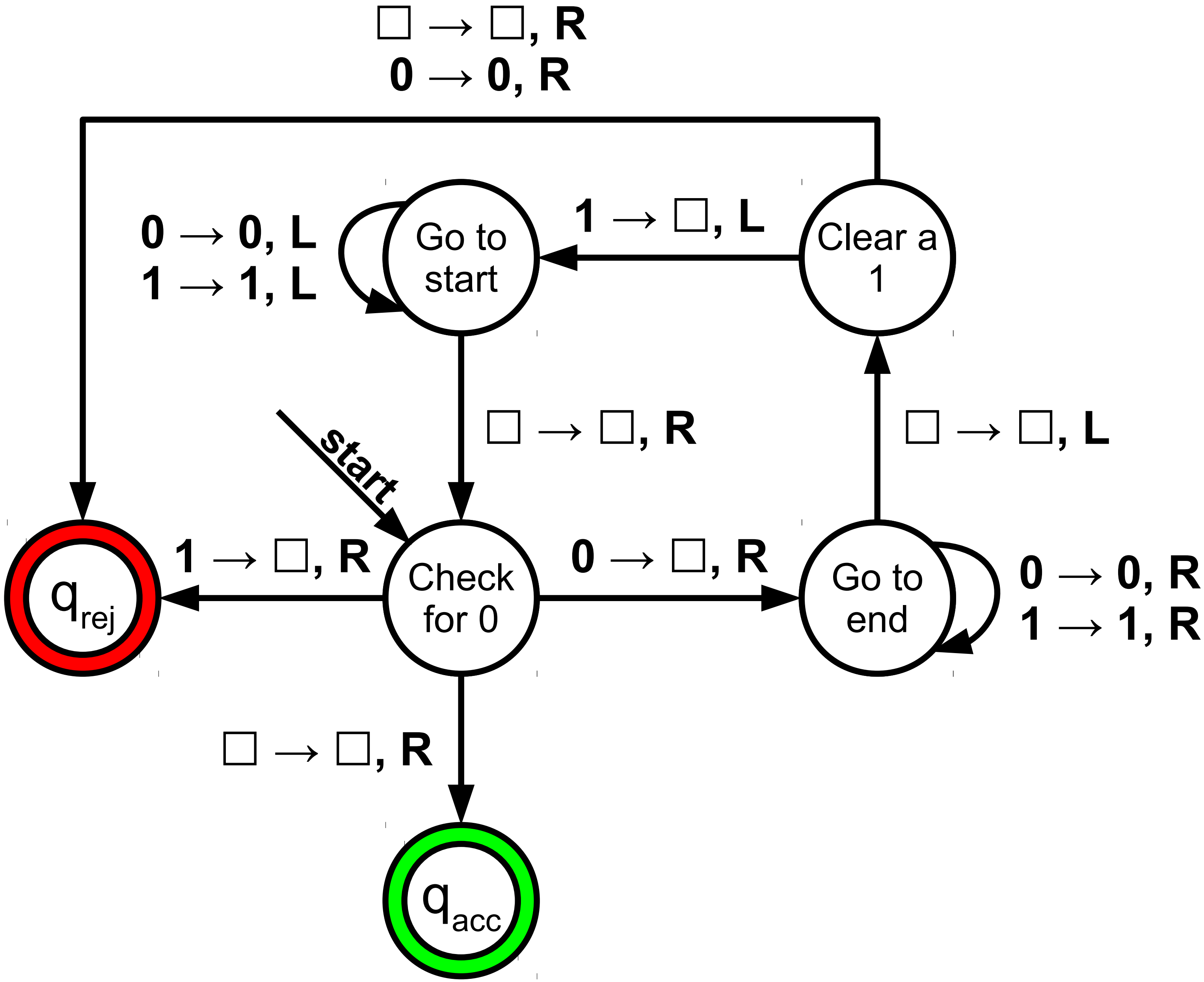
- Let $\Sigma = \{\mathtt{0}, \mathtt{1}\}$ and consider the language $L = \{\mathtt{0}^n\mathtt{1}^n \mid n \in \mathbb{N}\}$.
- We know that L is context-free.
- How might we build a Turing machine for it?

$$L = \{ \mathtt{0}^n \mathtt{1}^n \mid n \in \mathbb{N} \}$$



A Recursive Approach

- The string ε is in L.
- The string `0`w`1` is in L iff w is in L.
- Any string starting with `1` is not in L.
- Any string ending with `0` is not in L.



□ → □, R
0 → 0, R
0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
□ → □, R
start
□ → □, L
1 → □, R
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
q_rej
□ → □, R
q_acc

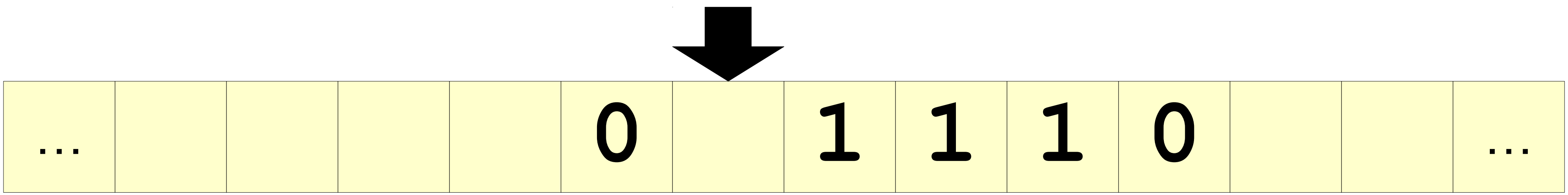
Another TM Design

- We've designed a TM for $\{\mathtt{0}^n\mathtt{1}^n \mid n \in \mathbb{N}\}$.
- Consider this language over $\Sigma = \{\mathtt{0}, \mathtt{1}\}$:

$$L = \{\ w \in \Sigma^* \mid w \text{ has the same number of } \mathtt{0}\text{s and } \mathtt{1}\text{s}\ \}$$

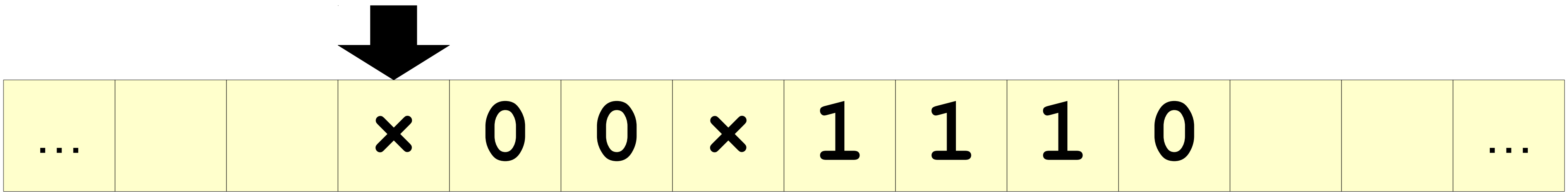
- This language is also not regular, but it is context-free.
- How might we design a TM for it?

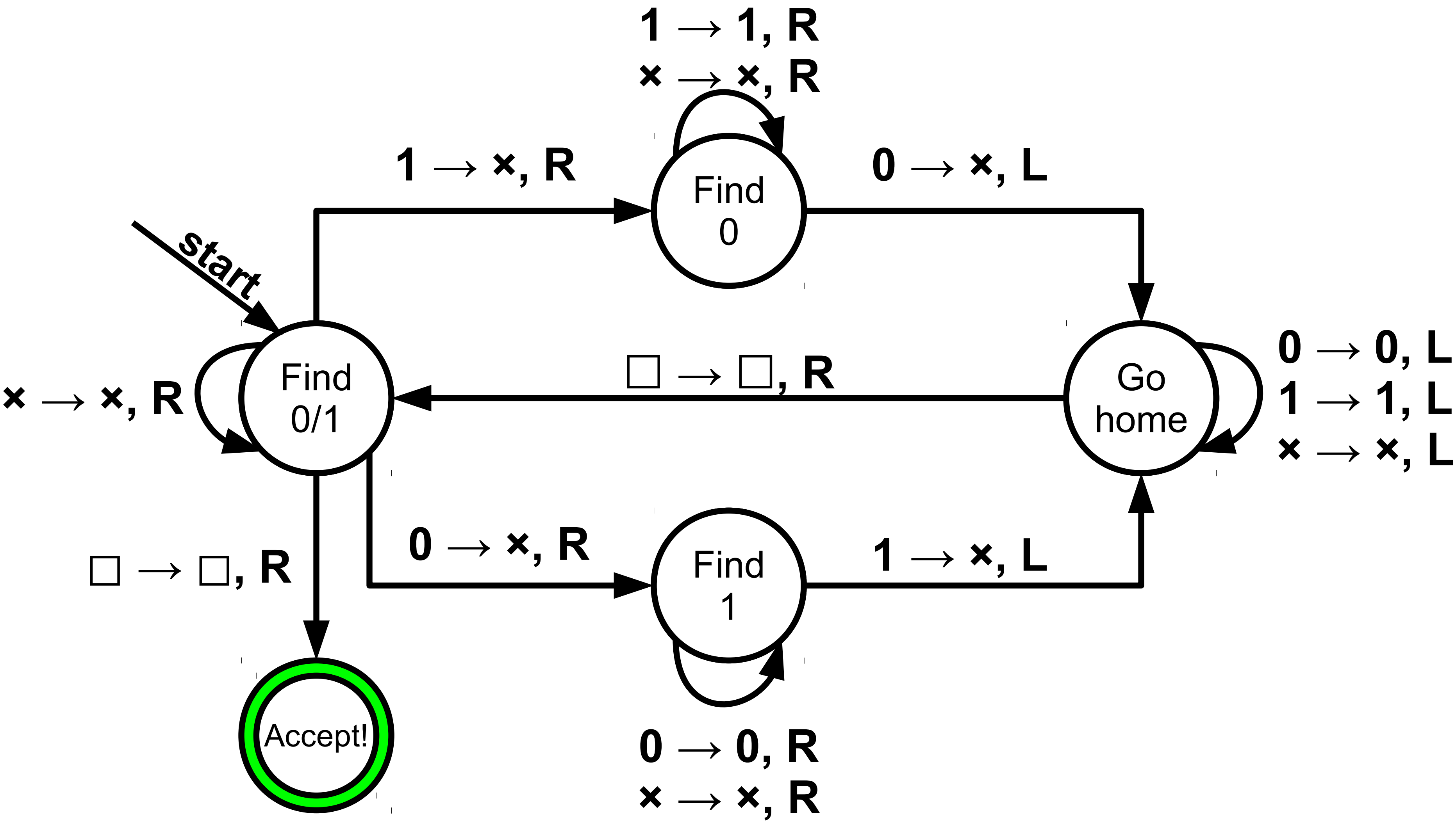
A Caveat



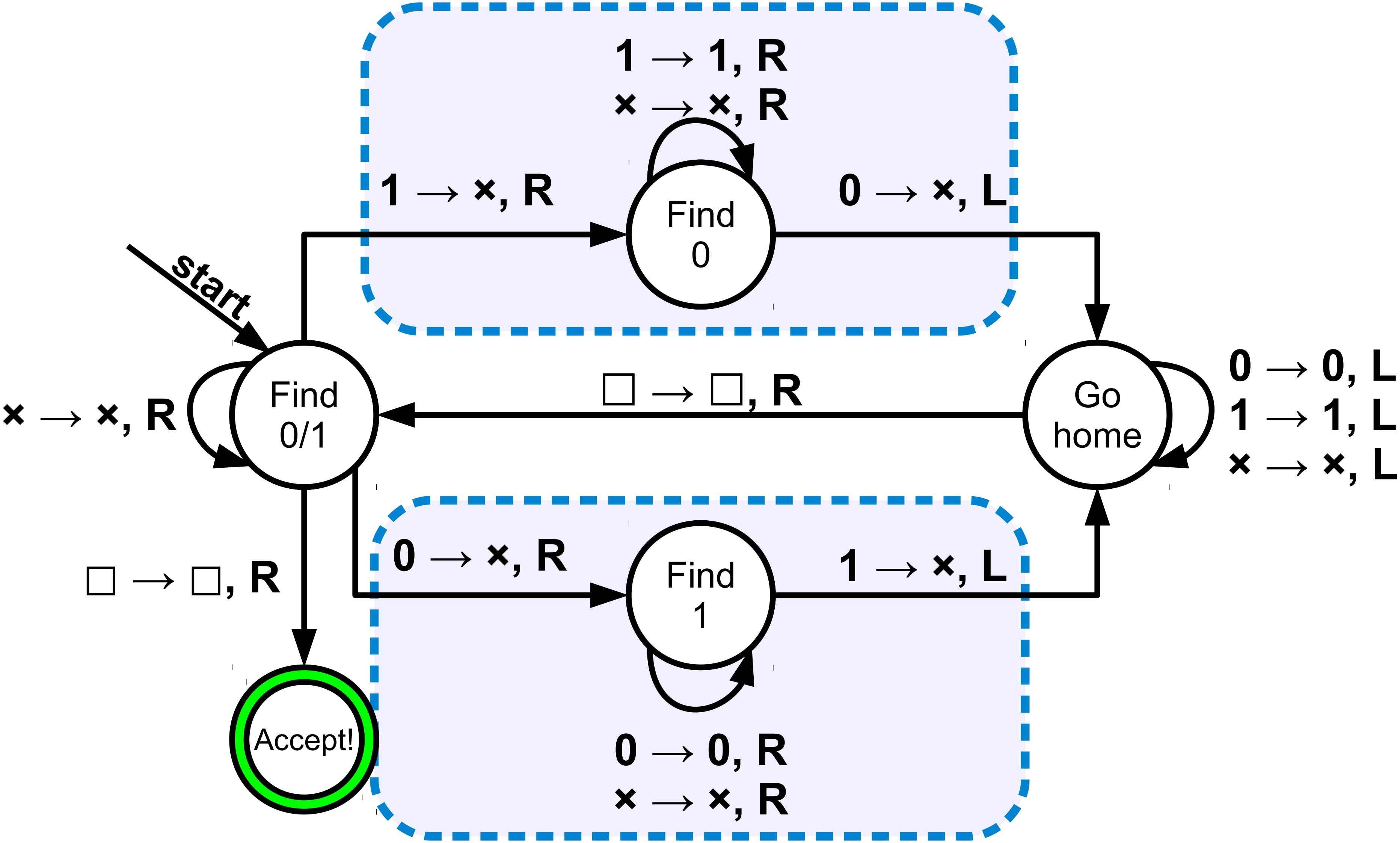
How do we know that this blank isn't one of the infinitely many blanks after our input string?

One Solution





Remember that all missing transitions implicitly reject.



start
Find
0/1
× → ×, R
1 → ×, R
Find
0
1 → 1, R
× → ×, R
0 → ×, L
Go
home
0 → 0, L
1 → 1, L
× → ×, L
□ → □, R
0 → ×, R
Find
1
1 → ×, L
0 → 0, R
× → ×, R
□ → □, R
Accept!

Constant Storage

- Sometimes, a TM needs to remember some additional information that can't be put on the tape.
- In this case, you can use similar techniques from DFAs and introduce extra states into the TM's finite-state control.
- The finite-state control can only remember one of finitely many things, but that might be all that you need!

Time-Out for Announcements!

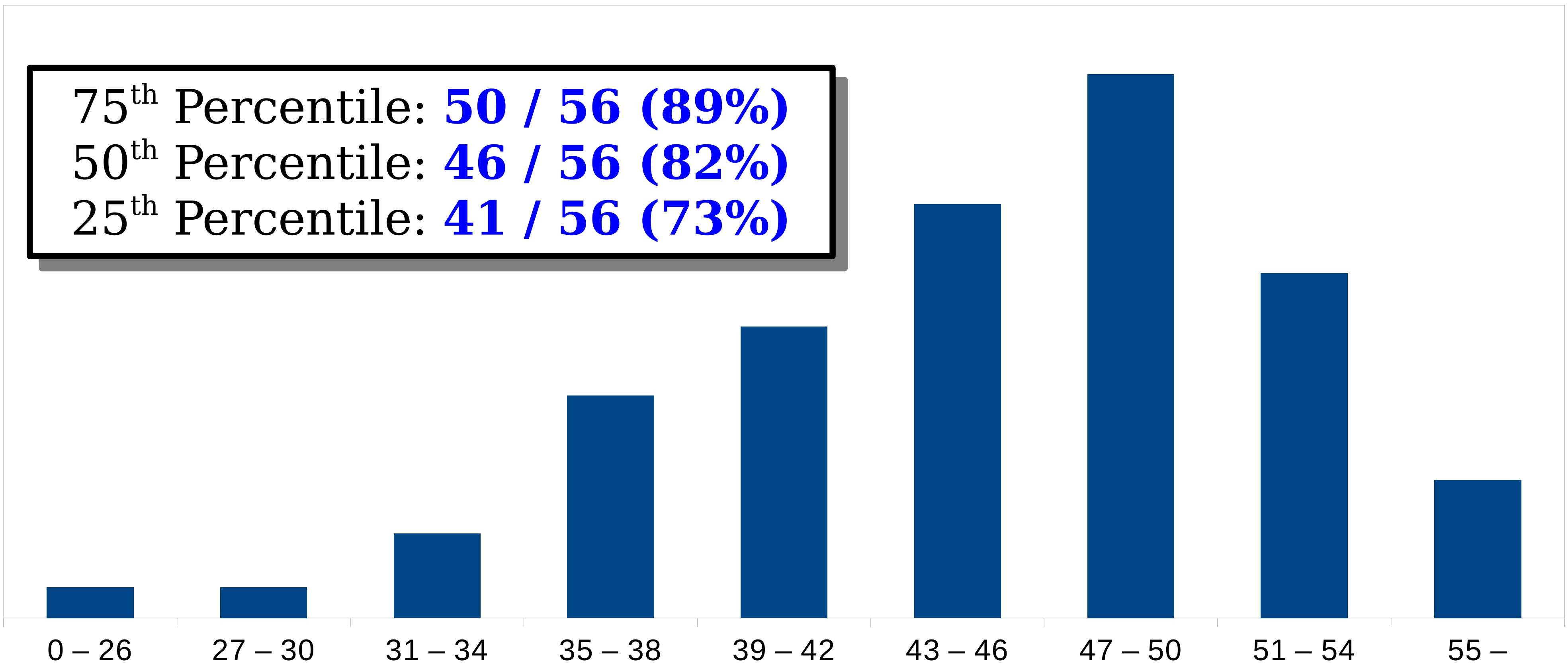
Second Midterm Exam

- You’re done with the second midterm exam! Woohoo!
- We’ll be grading the exam this weekend. Unfortunately, we will not be able to get grades back before Friday.
- Have questions? Feel free to ask in office hours or on Piazza!

Problem Set Seven

- Problem Set Seven is due this Friday at 2:30PM.
- As always, if you have questions, feel free to stop by office hours or ask on Piazza!

Problem Set Six Scores



Back to CS103!

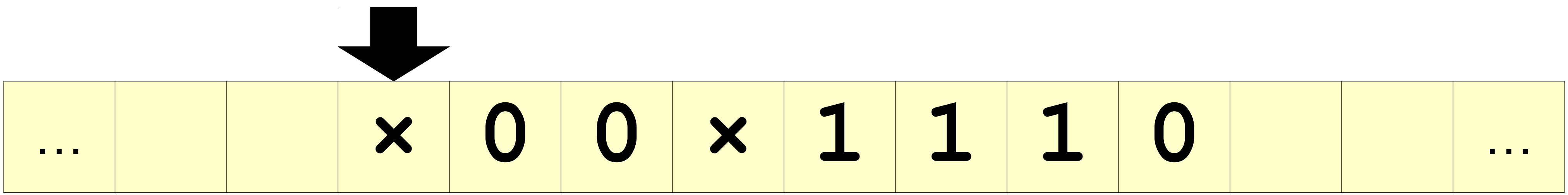
Another TM Design

- We just designed a TM for this language over $\Sigma = \{$**0**, **1**$\}$:

  $L = \{ w \in \Sigma^* \mid w$ has the same number of **0**s and **1**s $\}$

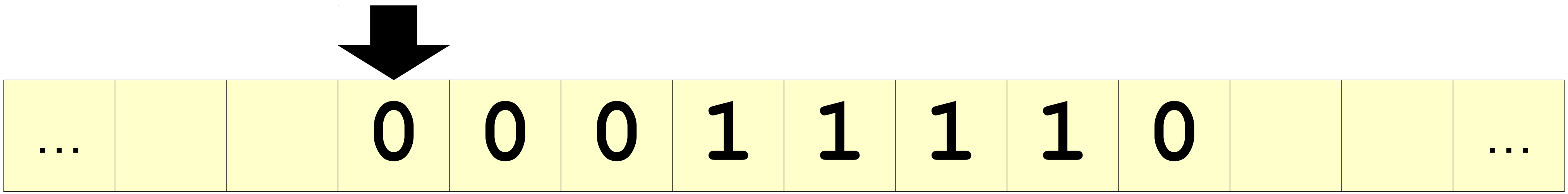
- Let's do a quick review of how it worked.

The Solution



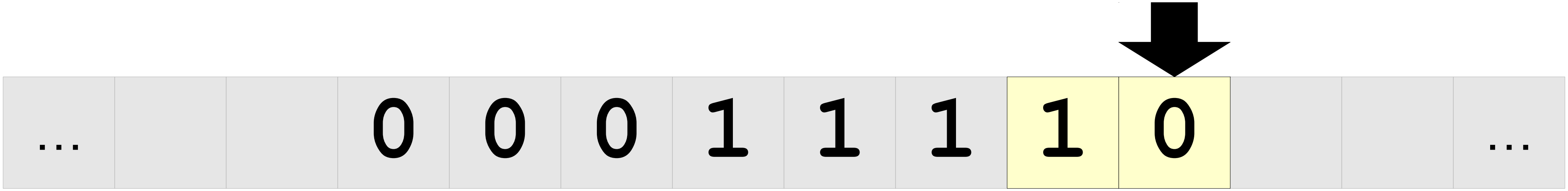
A Different Idea

A Different Strategy



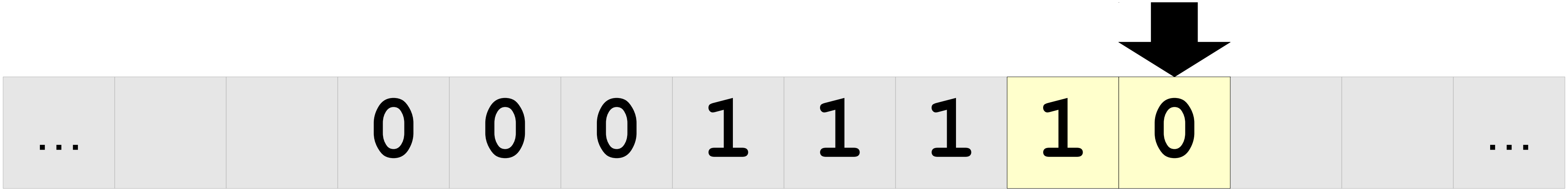
Could we sort the characters of this string?

A Different Strategy



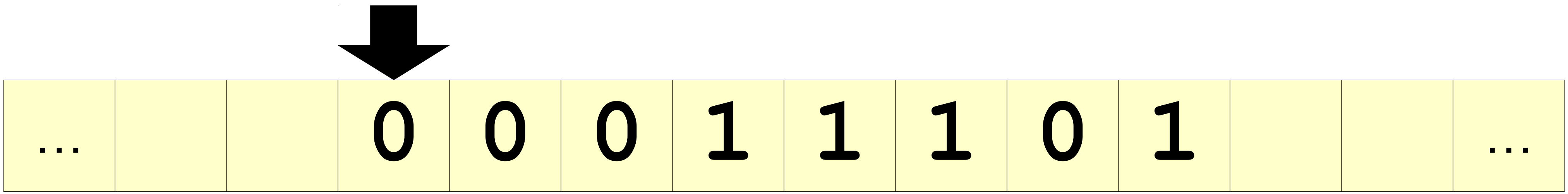
Observation 1: A string of 0s and 1s is sorted if it matches the regex 0*1*.

A Different Strategy



Observation 2: A string of 0s and 1s is <u>not</u> sorted if it contains 10 as a substring.

A Different Strategy

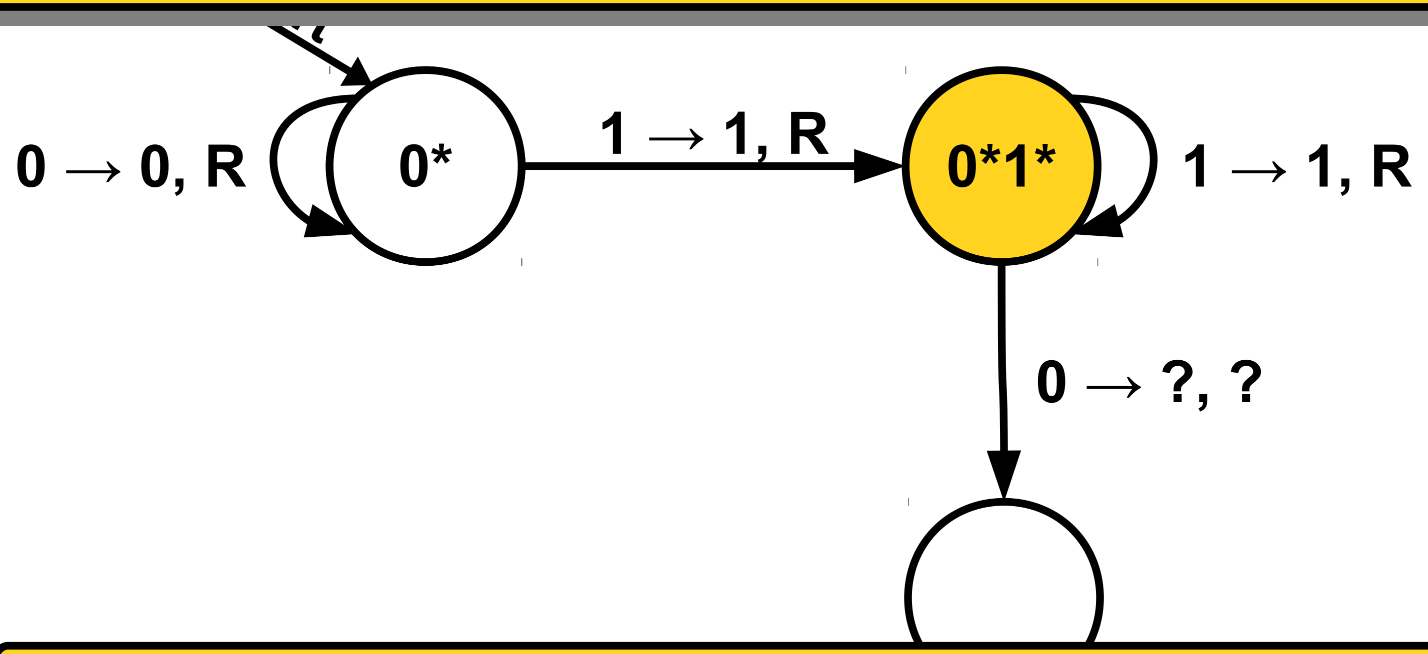


Idea: Repeatedly find a copy of 10 and replace it with 01.

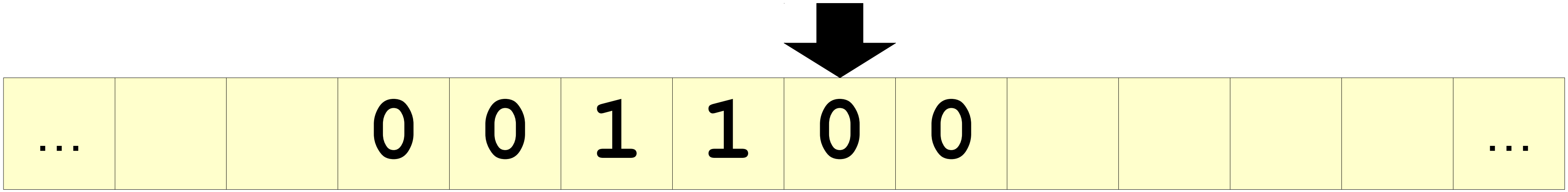
Let's Build It!

Based on we want this TM to do, what should this transition say?

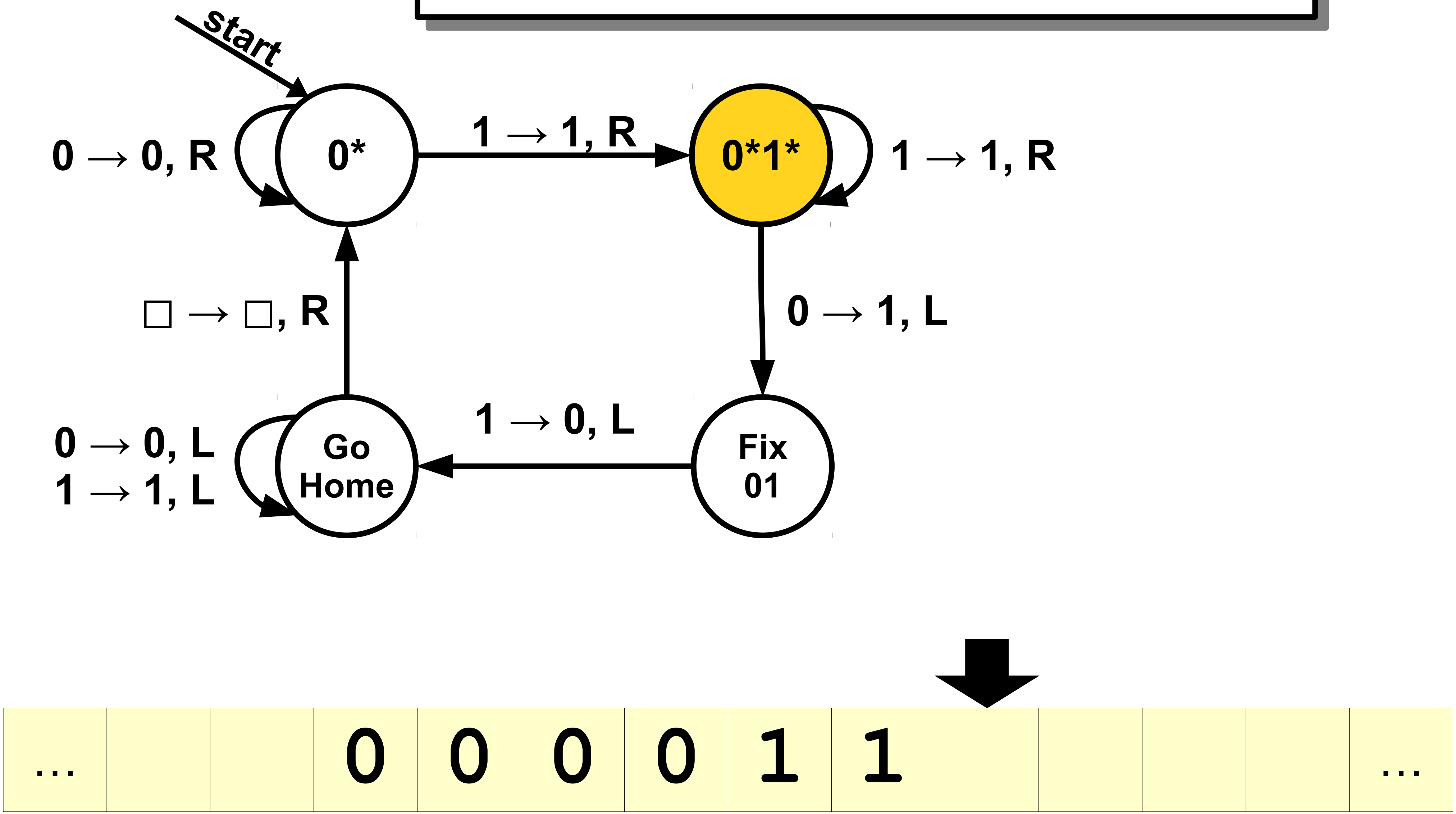
A. **0** → **0**, R
B. **0** → **1**, R
C. **0** → **0**, L
D. **0** → **1**, L
E. None of these, or two or more of these.

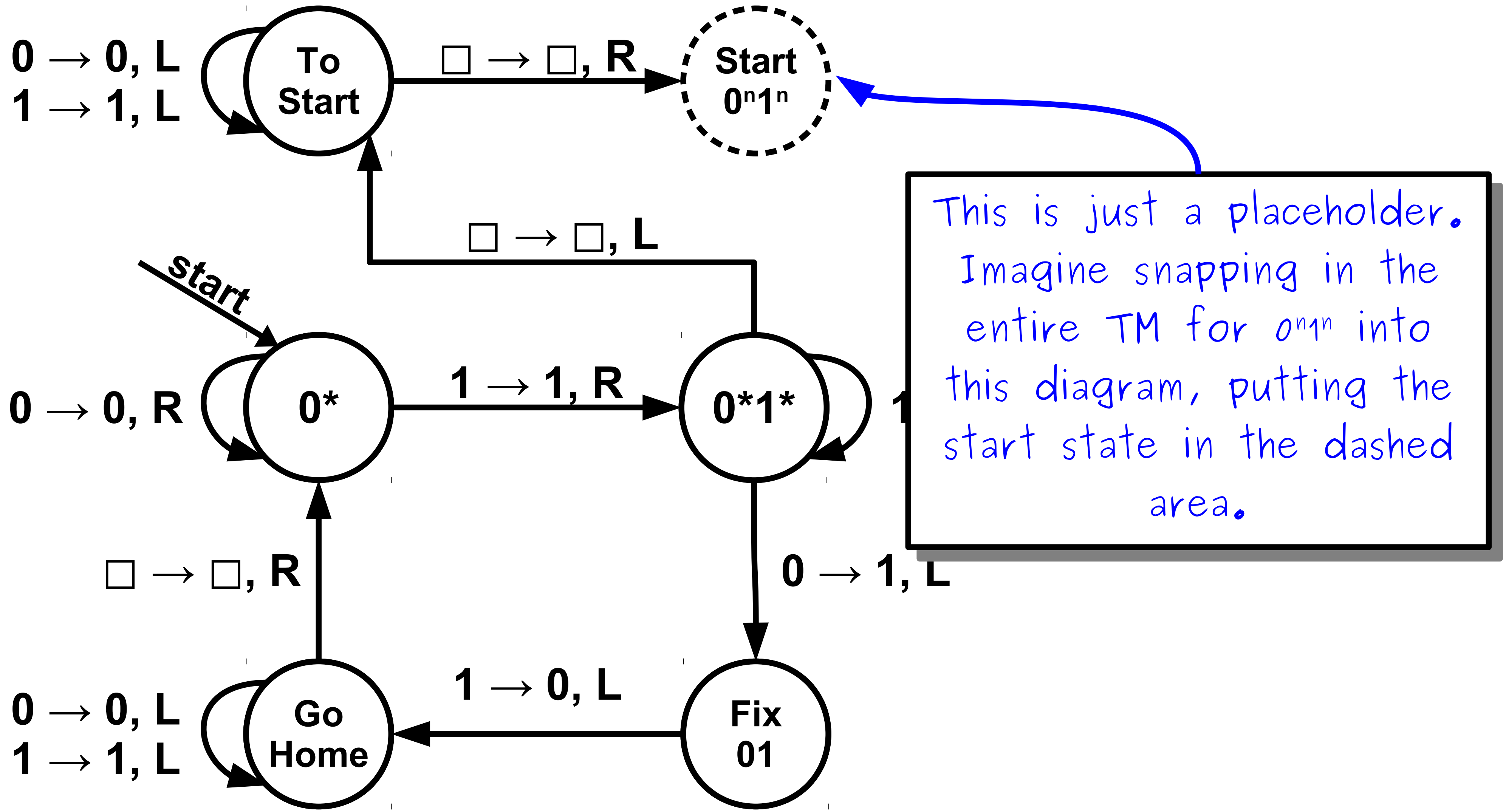


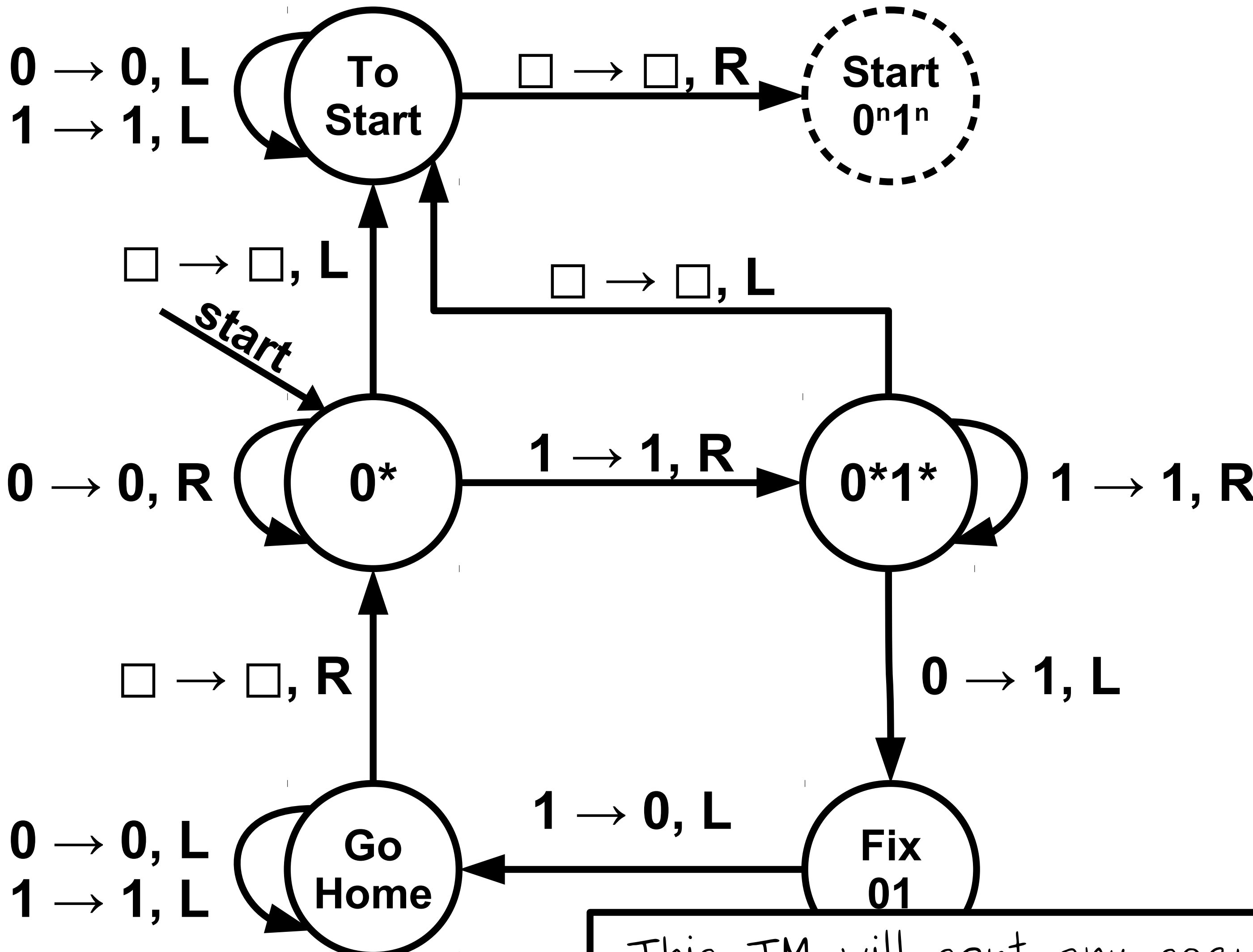
Answer at **PollEv.com/cs103** or
text **CS103** to **22333** once to join, then ***A***, ***B***, ***C***, ***D***, or ***E***.



Our ultimate goal here was to sort everything so we could hand it off to the machine to check for 0^n1^n. Let's rewind the tape head back to the start.




0 → 0, L
1 → 1, L
To Start
□ → □, R
Start 0^n1^n
□ → □, L
start
0 → 0, R
0*
1 → 1, R
0*1*
□ → □, R
0 → 1, L
0 → 0, L
1 → 1, L
Go Home
1 → 0, L
Fix 01
This is just a placeholder. Imagine snapping in the entire TM for 0^n1^n into this diagram, putting the start state in the dashed area.



This TM will sort any sequence of 0s and 1s, but it might take a while.

Fun problem: design a TM that sorts a string of 0s and 1s, but does so while taking way fewer steps than this machine.

TM Subroutines

- A ***TM subroutine*** is a Turing machine that, instead of accepting or rejecting an input, does some sort of processing job.
- TM subroutines let us compose larger TMs out of smaller TMs, just as you'd write a larger program using lots of smaller helper functions.
- Here, we saw a TM subroutine that sorts a sequence of 0s and 1s into ascending order.

TM Subroutines

- Typically, when a subroutine is done running, you have it enter a state marked “done” with a dashed line around it.
- When we're composing multiple subroutines together – which we'll do in a bit – the idea is that we'll snap in some real state for the “done” state.

What other subroutines can we make?