Turing Machines

Part One

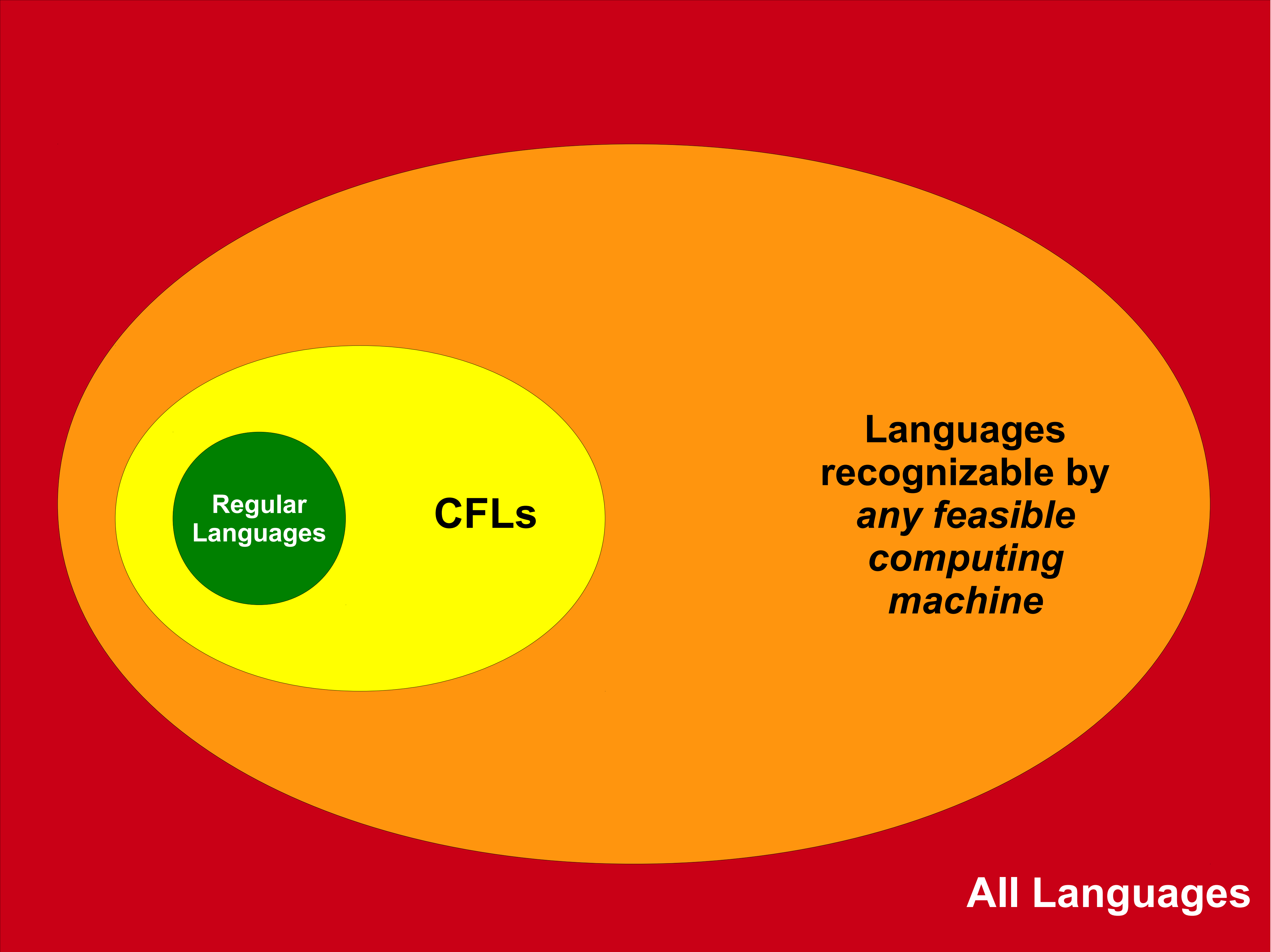
Hello Condensed Slide Readers!

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.

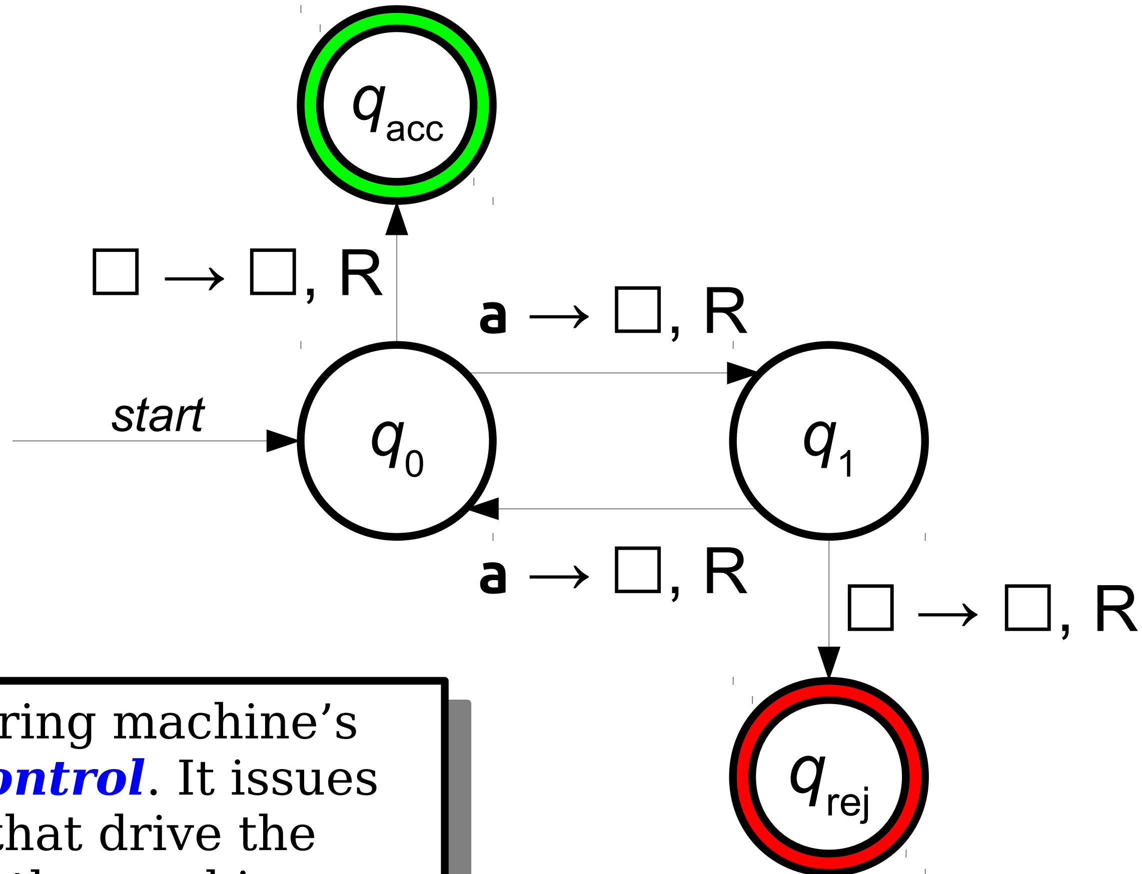
All Languages

The Problem

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

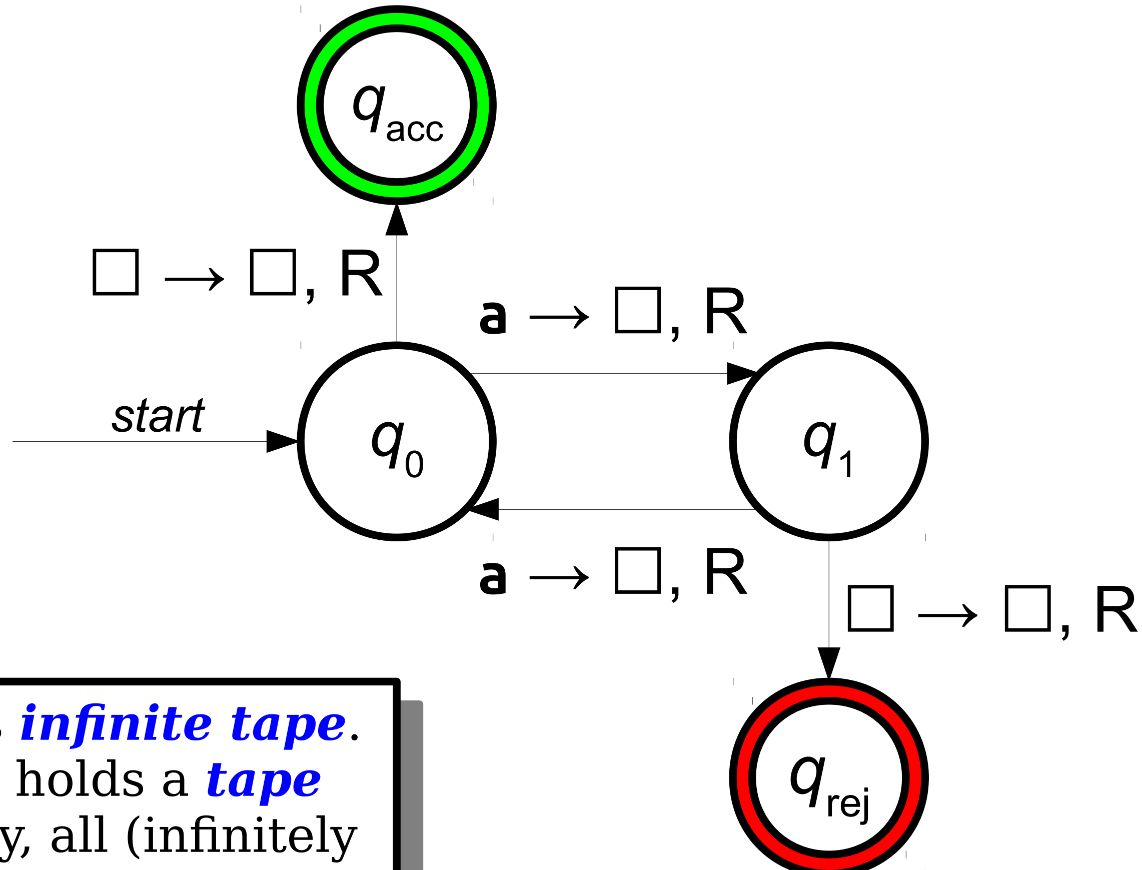
A Brief History Lesson

A Simple Turing Machine

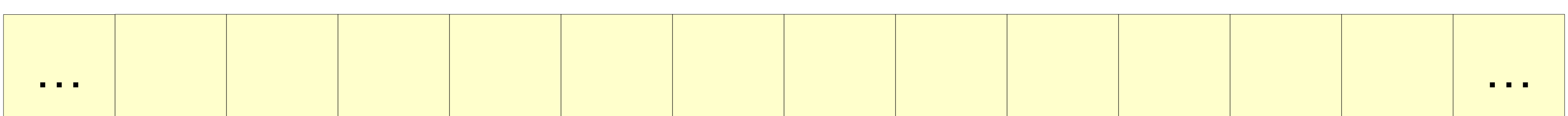


This is the Turing machine's ***finite state control***. It issues commands that drive the operation of the machine.

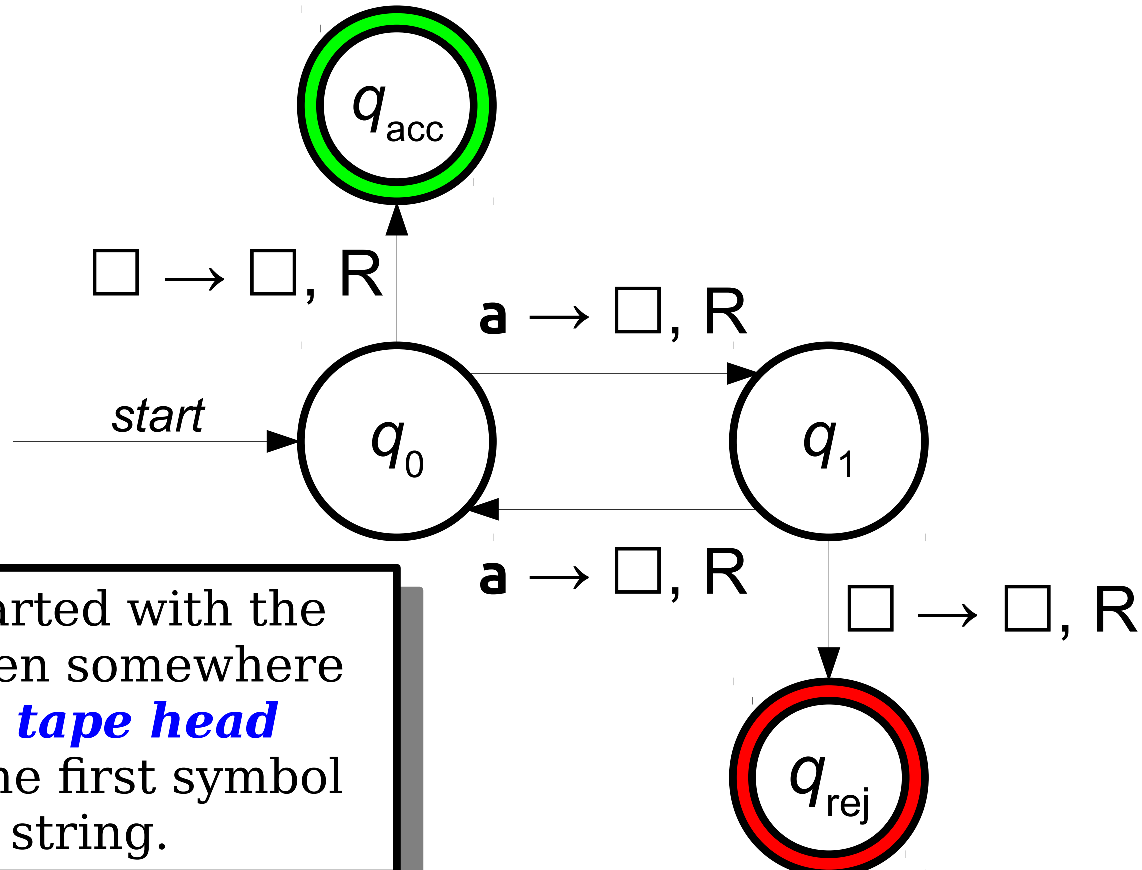
A Simple Turing Machine



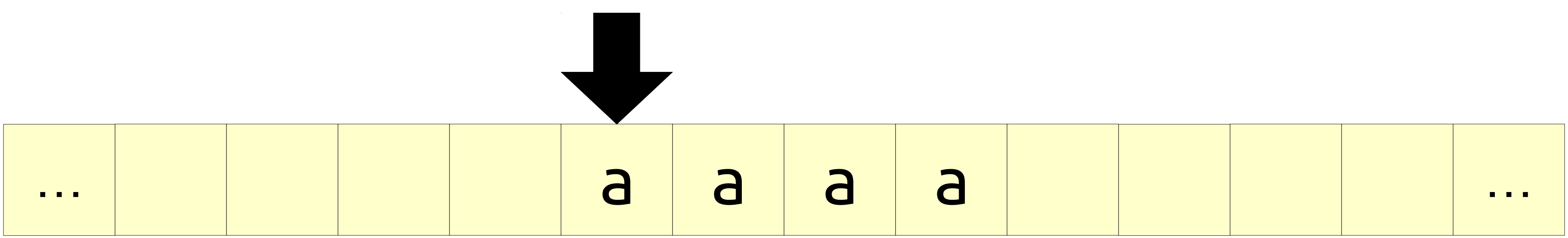
This is the TM's ***infinite tape***. Each tape cell holds a ***tape symbol***. Initially, all (infinitely many) tape symbols are blank.



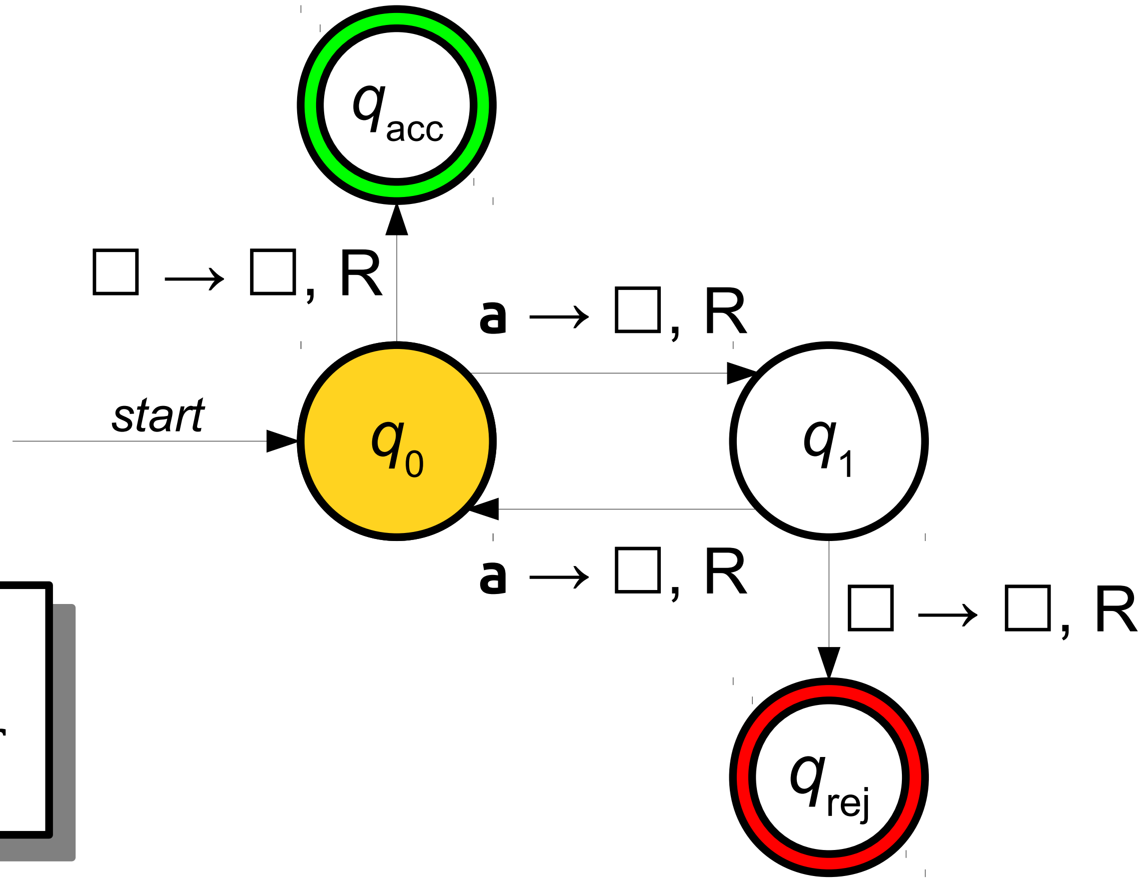
A Simple Turing Machine



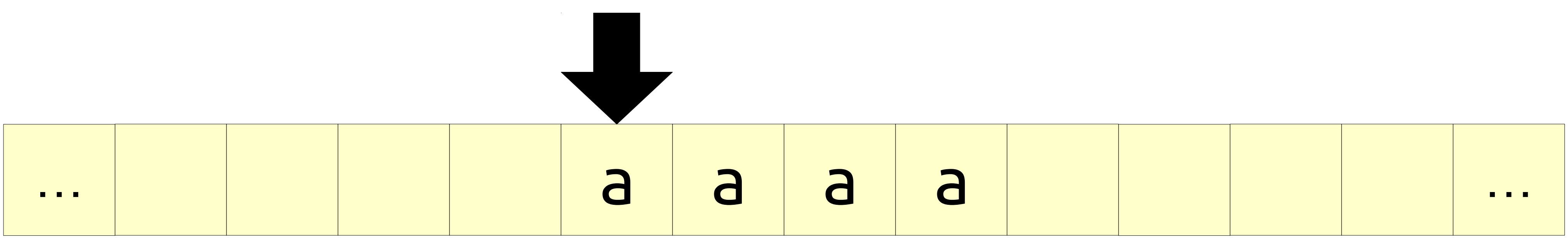
The machine is started with the ***input string*** written somewhere on the tape. The ***tape head*** initially points to the first symbol of the input string.



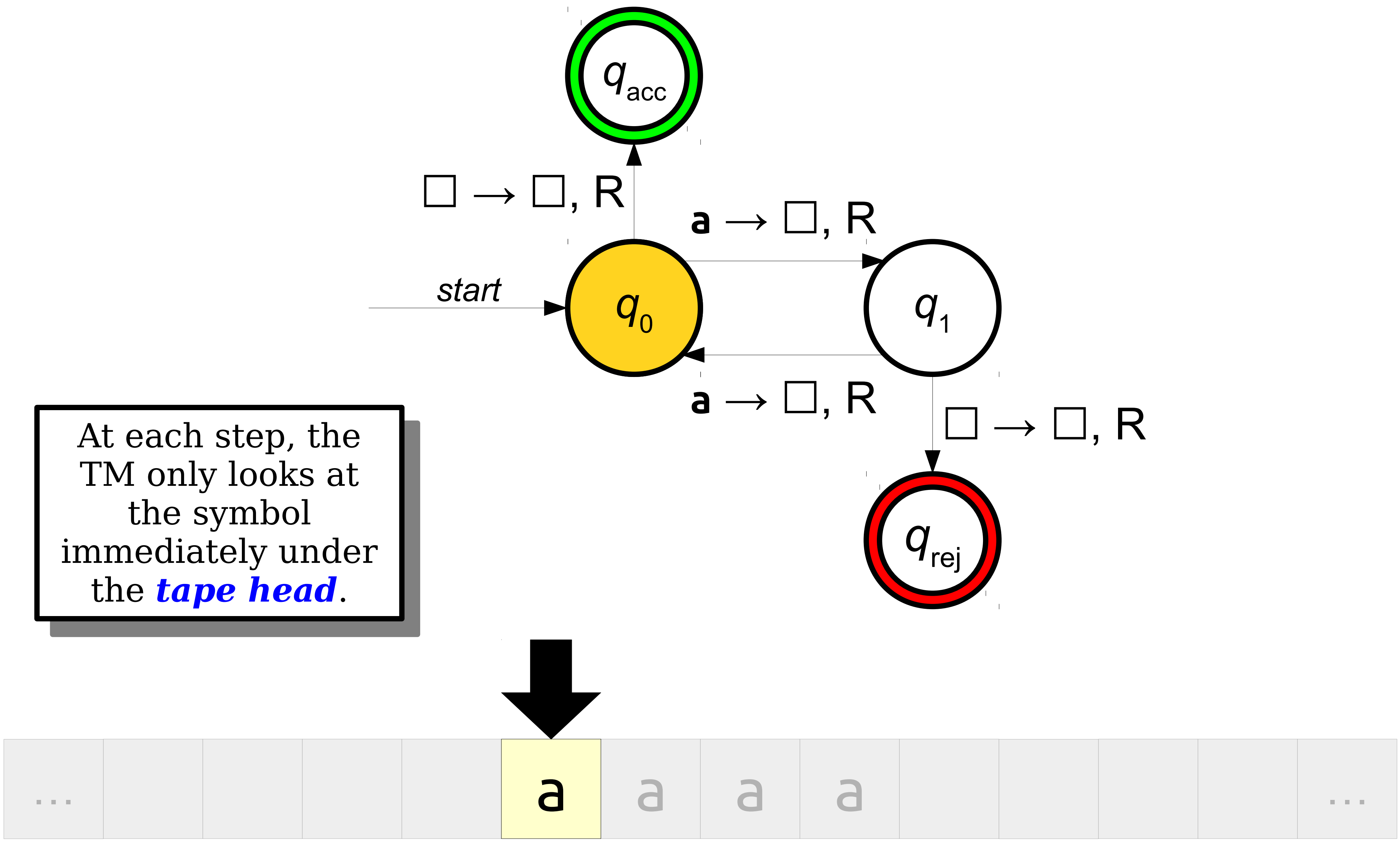
A Simple Turing Machine



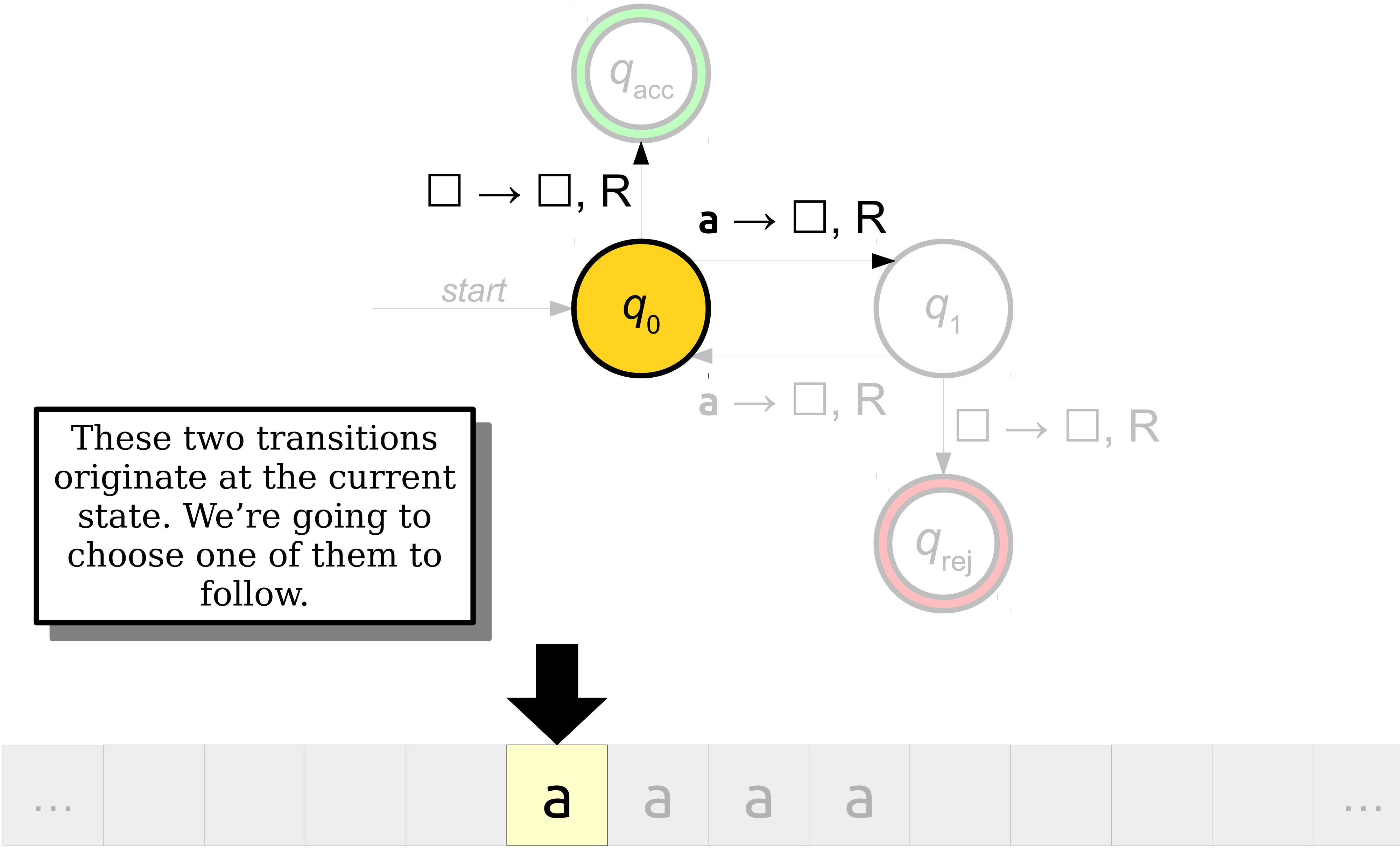
Like DFAs and NFAs, TMs begin execution in their ***start state***.



A Simple Turing Machine

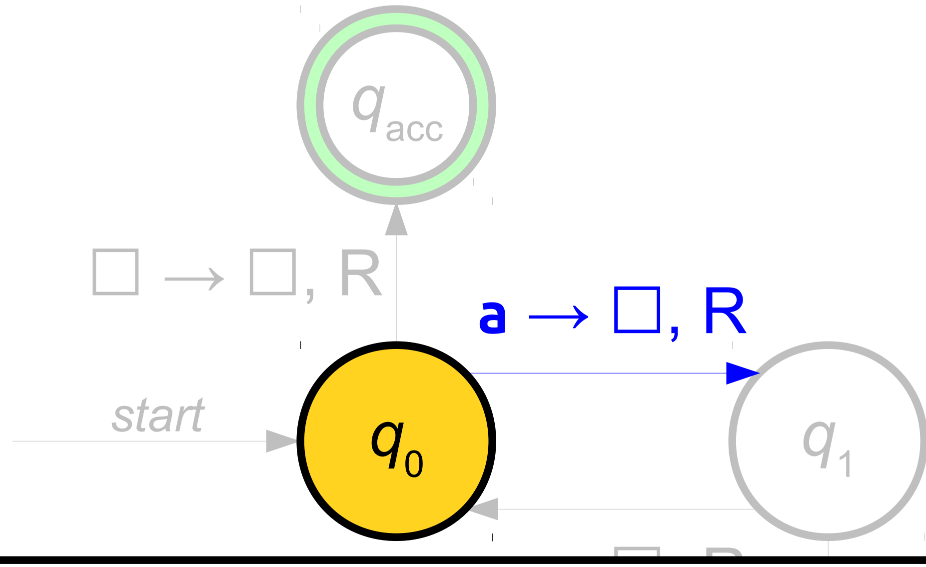


A Simple Turing Machine



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

A Simple Turing Machine

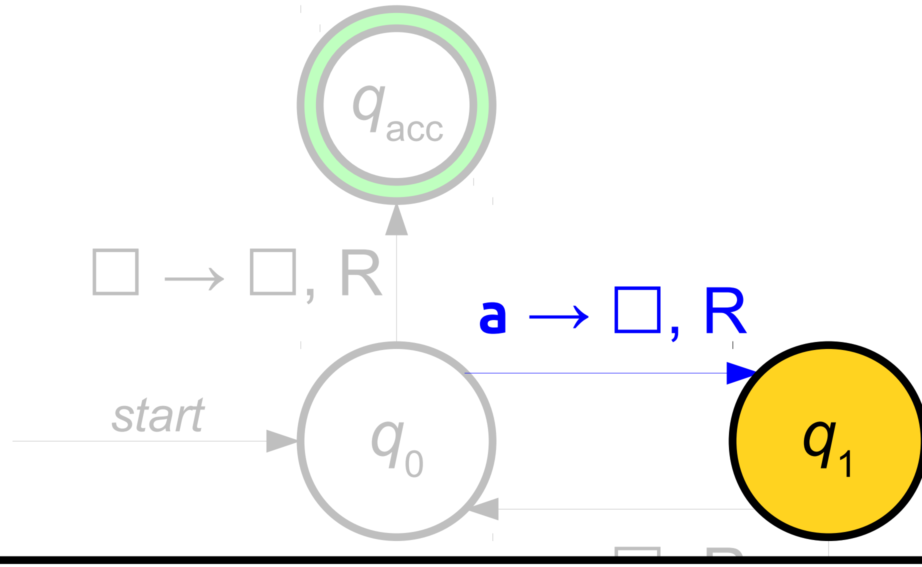


Each transition has the form

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.

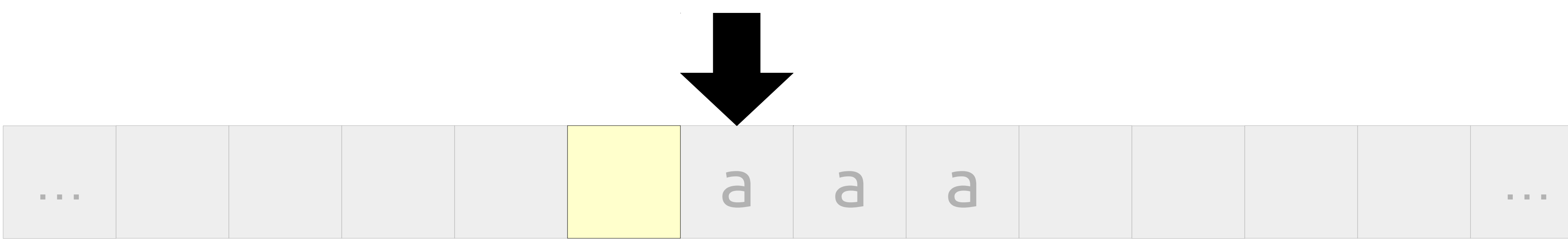
A Simple Turing Machine



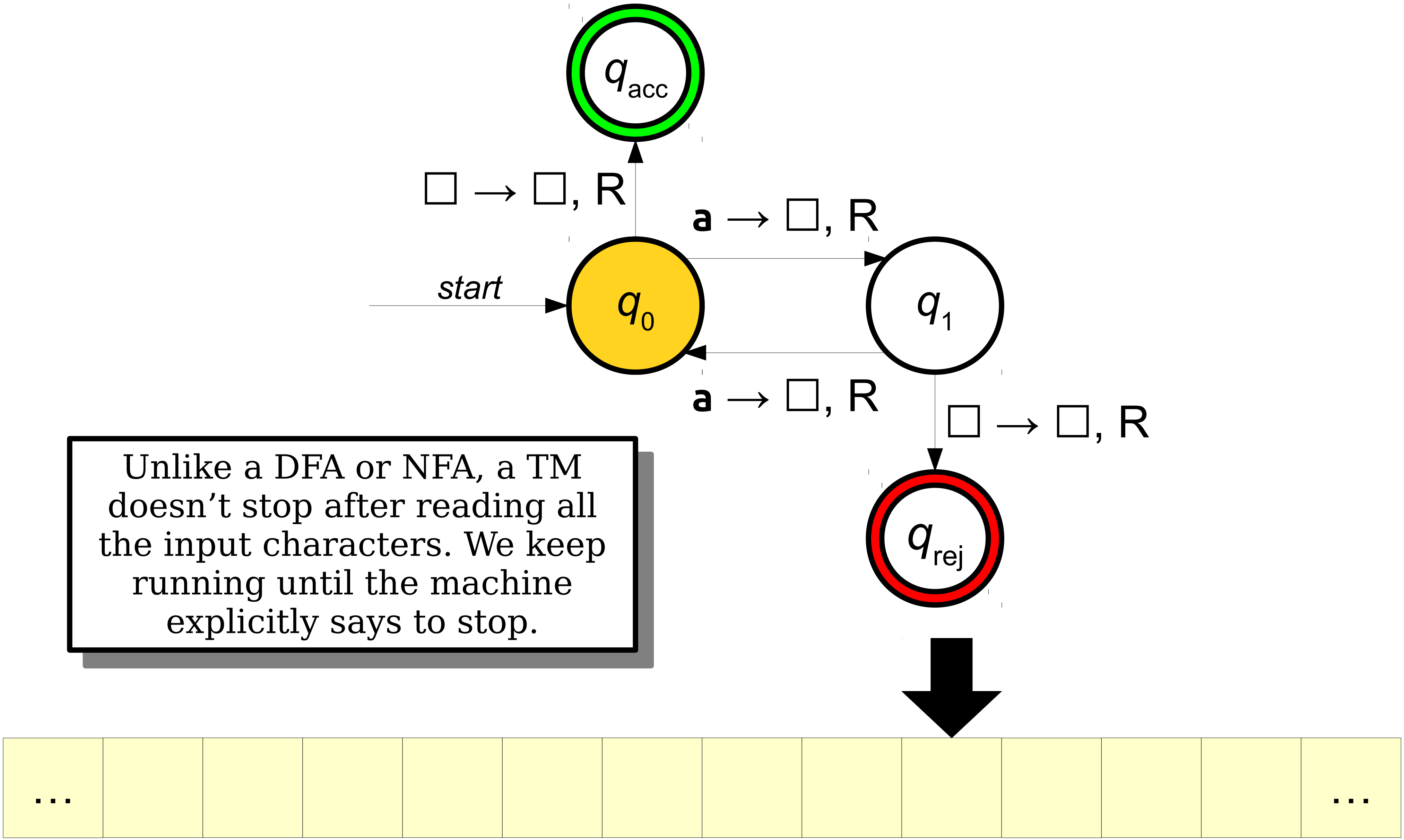
Each transition has the form

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.

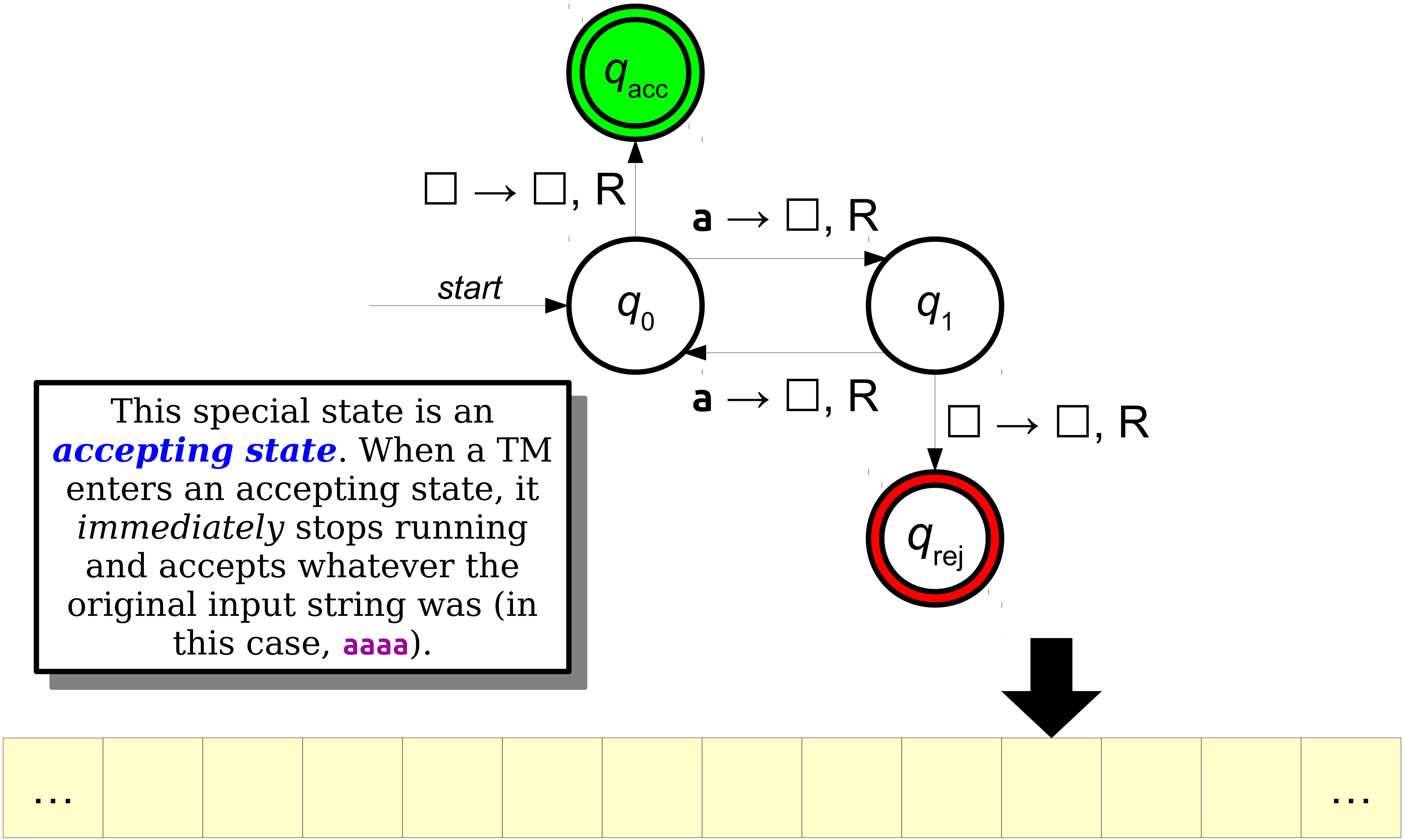


A Simple Turing Machine



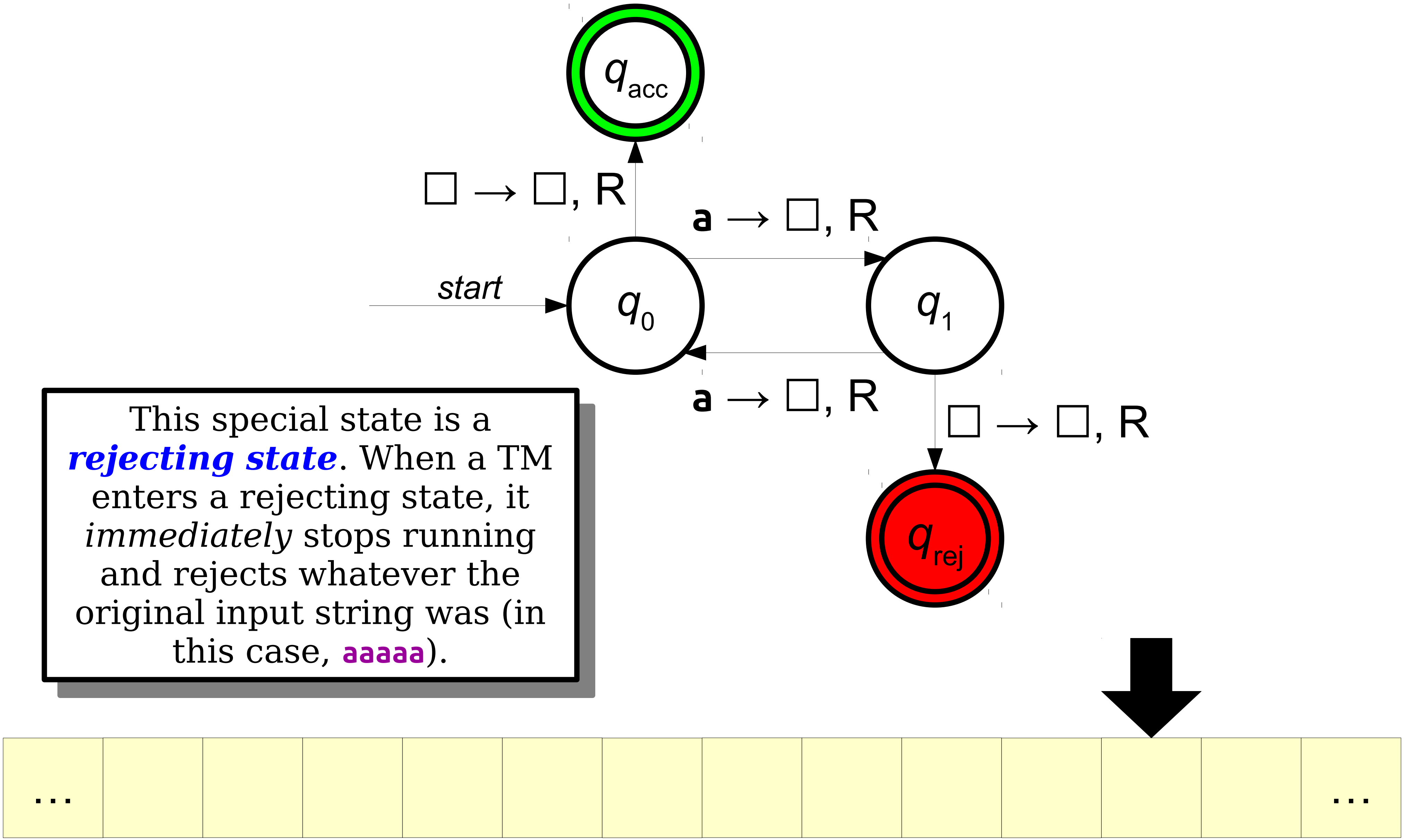
Unlike a DFA or NFA, a TM doesn’t stop after reading all the input characters. We keep running until the machine explicitly says to stop.

A Simple Turing Machine



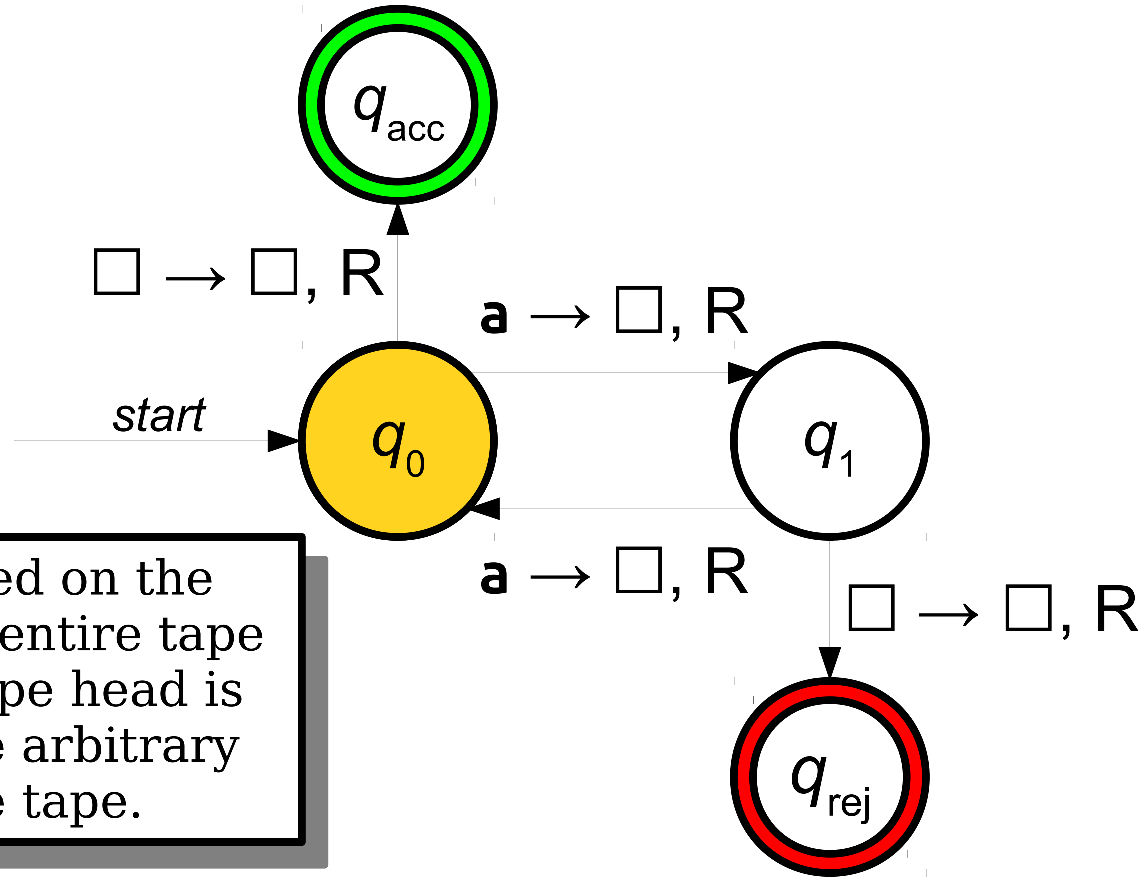
This special state is an ***accepting state***. When a TM enters an accepting state, it *immediately* stops running and accepts whatever the original input string was (in this case, **aaaa**).

A Simple Turing Machine

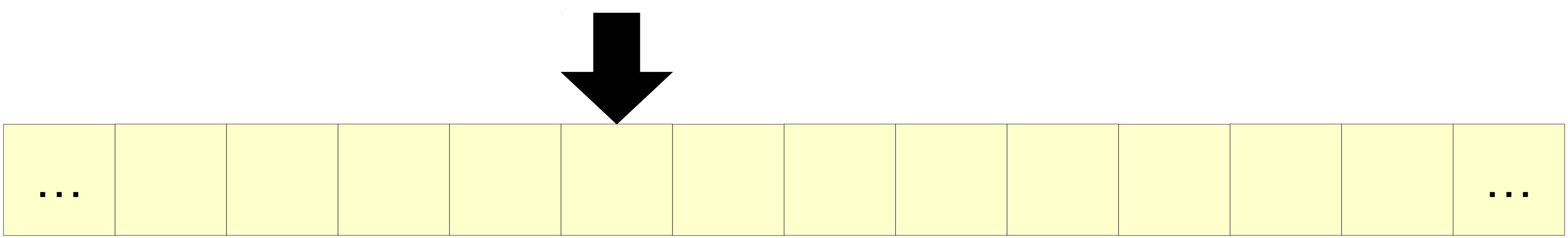


This special state is a ***rejecting state***. When a TM enters a rejecting state, it *immediately* stops running and rejects whatever the original input string was (in this case, **aaaaa**).

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

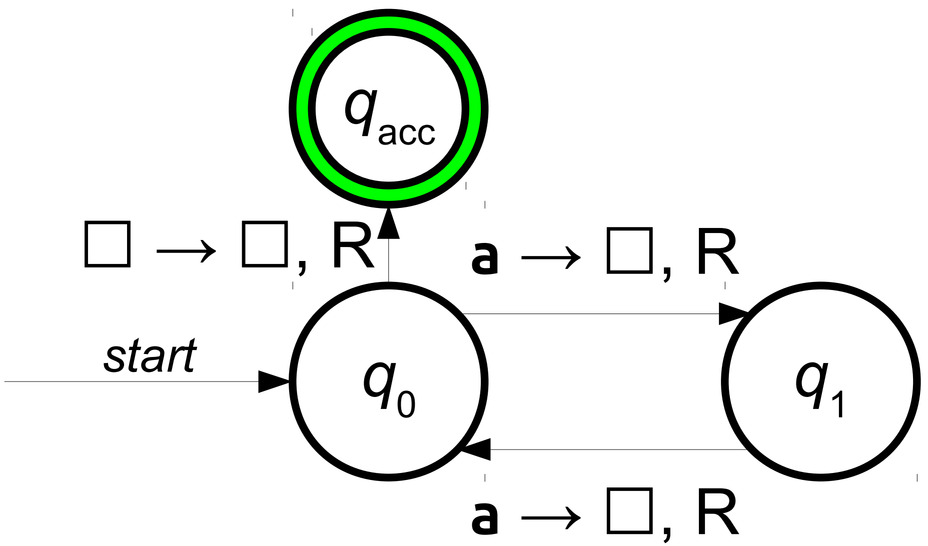
- A Turing machine has two alphabets:
  - An ***input alphabet*** Σ. All input strings are written in the input alphabet.
  - A ***tape alphabet*** Γ, where $\Sigma \subsetneq \Gamma$. The tape alphabet contains all symbols that can be written onto the tape.
- The tape alphabet Γ can contain any number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



This machine is exactly the same as the previous one.

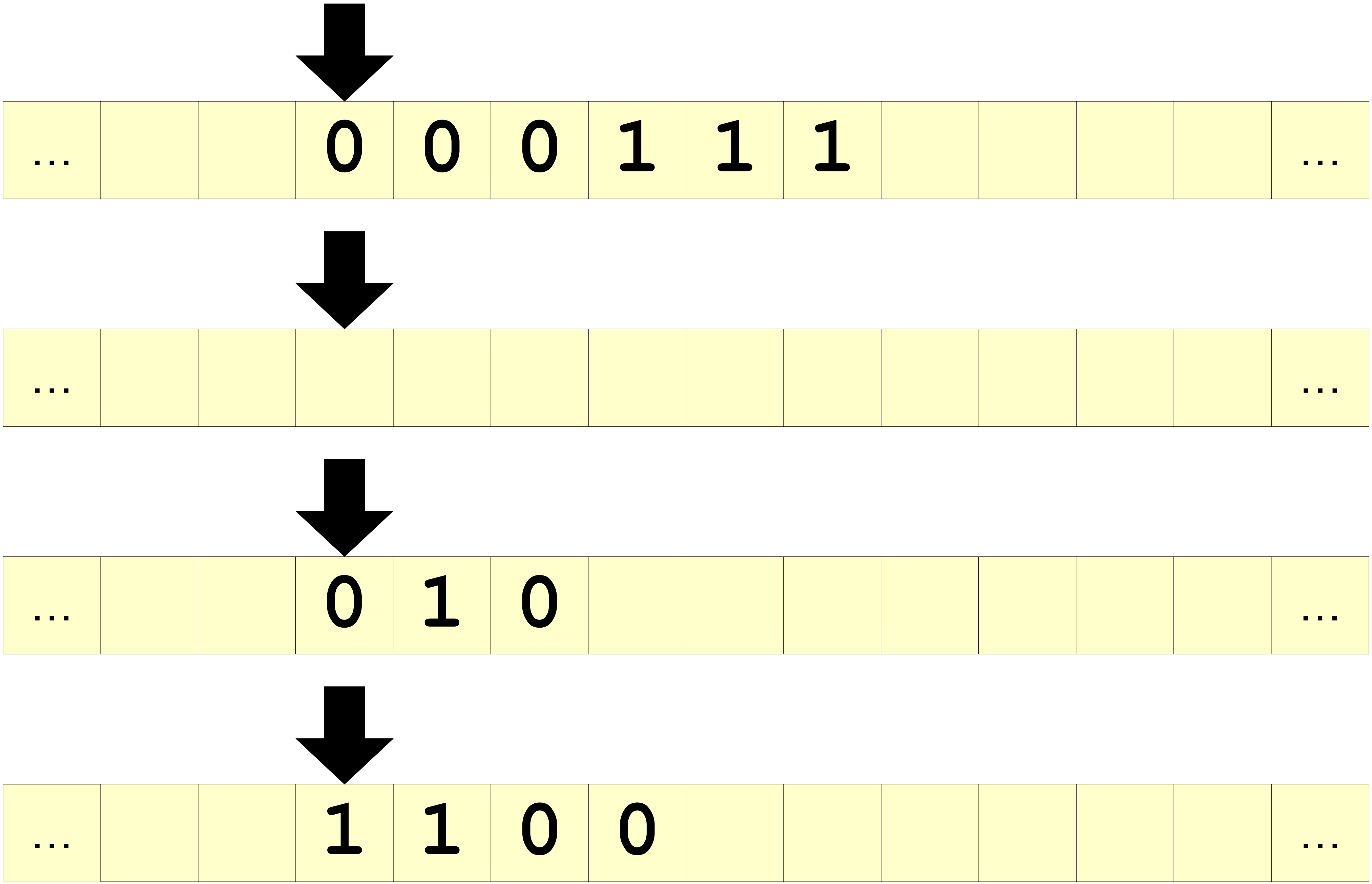
Designing Turing Machines

- 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

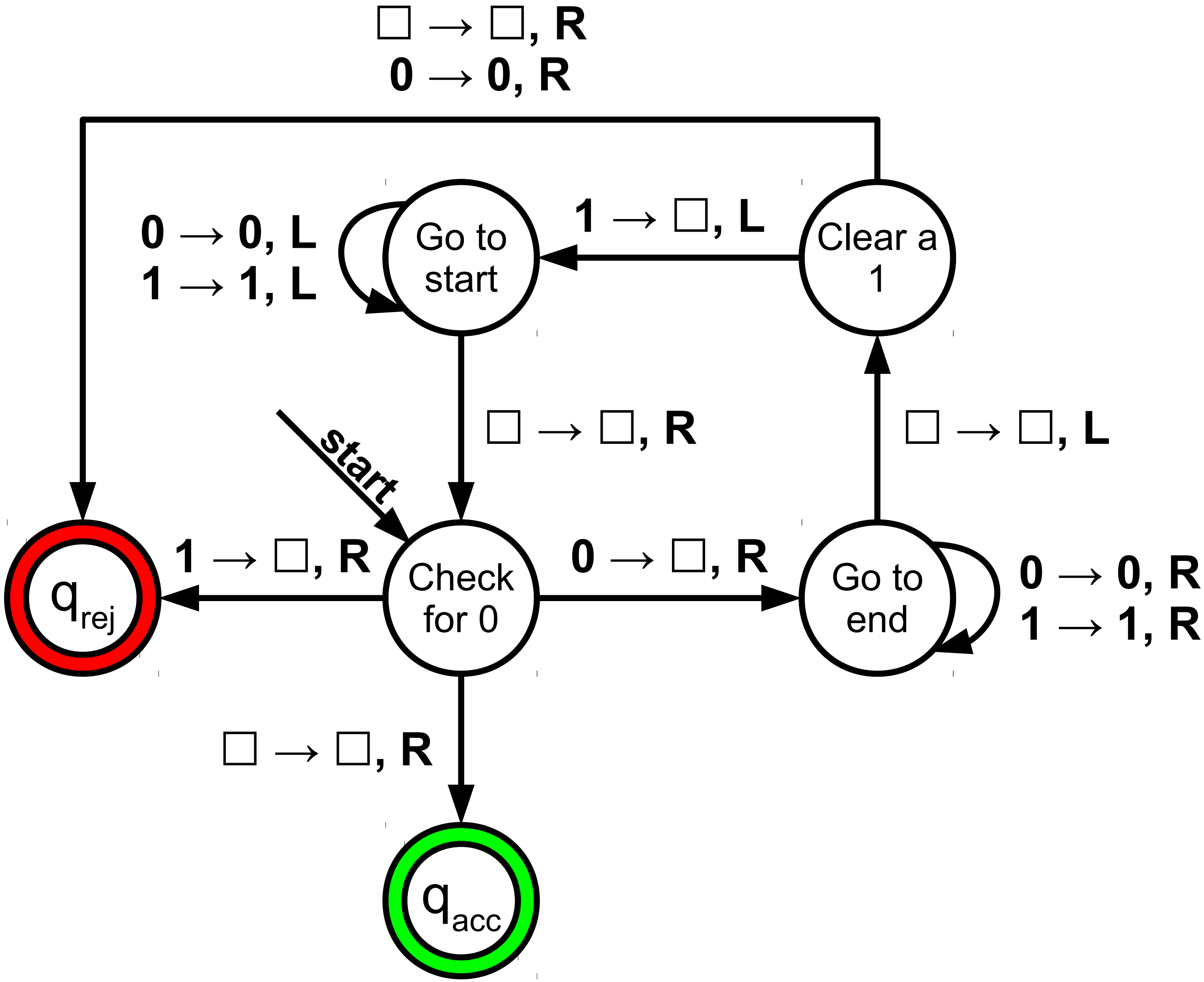
- Let $\Sigma = \{\mathtt{0}, \mathtt{1}\}$ and consider the language $L = \{\mathtt{0}^n\mathtt{1}^n \mid n \in \mathbb{N}\}$.
- We know that L is context-free.
- How might we build a Turing machine for it?

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



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.

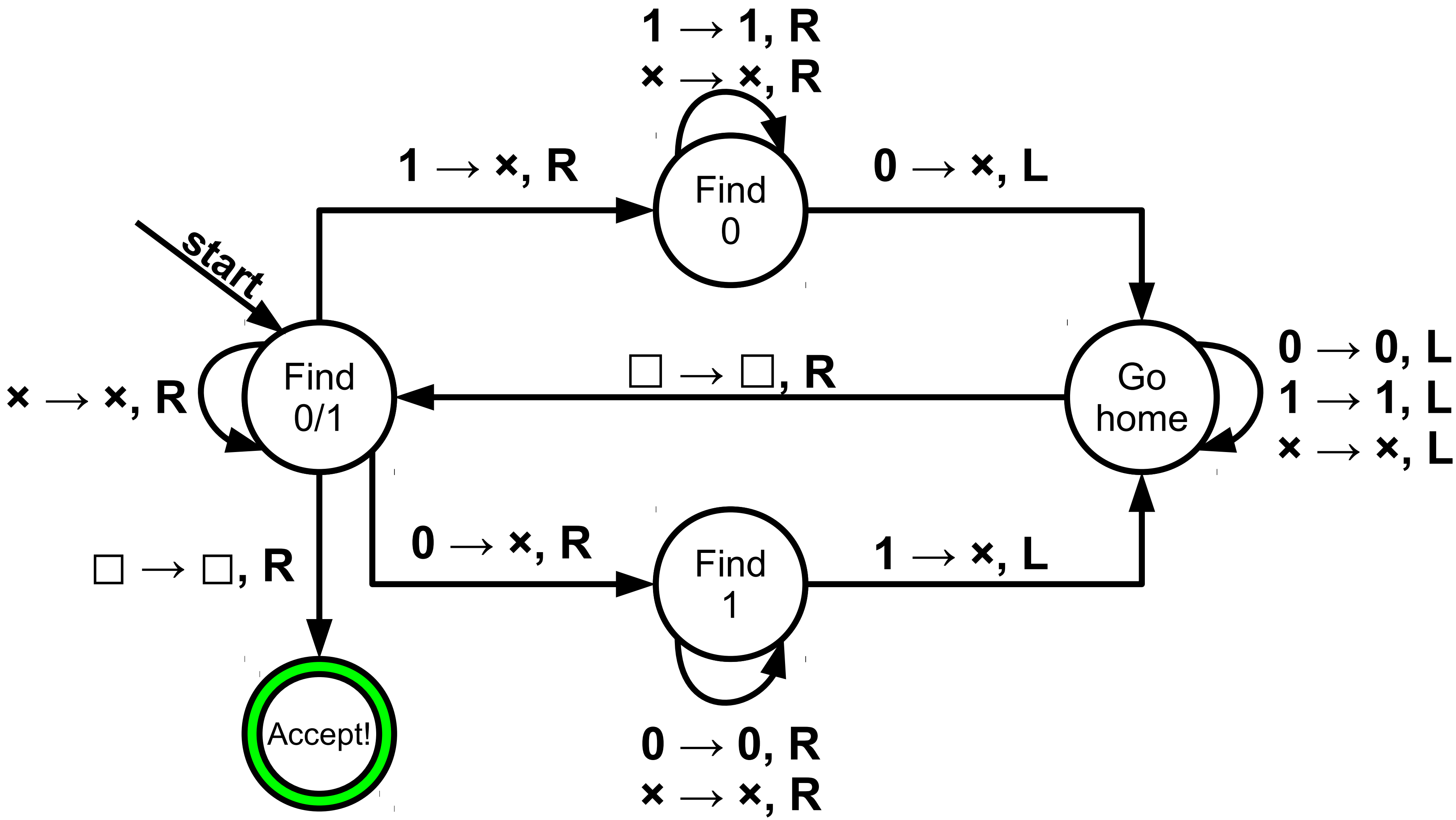

☐ → ☐, R
0 → 0, R
0 → 0, L
1 → 1, L
Go to start
1 → ☐, L
Clear a 1
☐ → ☐, R
start
☐ → ☐, L
q_rej
1 → ☐, R
Check for 0
0 → ☐, R
Go to end
0 → 0, R
1 → 1, R
☐ → ☐, R
q_acc

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?



Remember that all missing transitions implicitly reject.

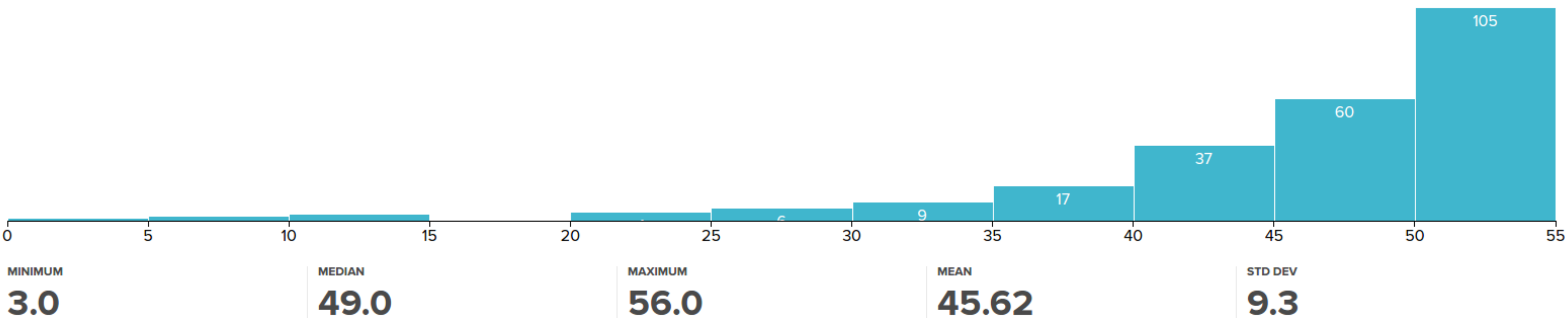
Constant Storage

- 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!

Problem Set Five

- Problem Set Five has been graded. Here's the score distribution:



- ***(Nice job, everyone!)***
- As always, feel free to reach out to us if you have any questions!

Problem Sets

- Problem Set Six was due at the start of class today.
  - You can use late days to extend the deadline up through Monday, but be careful about doing this given that the midterm is on Tuesday.
- Problem Set Seven goes out today. It’s due next Friday at the start of class.
  - Play around with regular expressions, properties of regular languages, the limits of regular languages, and the Myhill-Nerode theorem!

Midterm Exam Logistics

- The second midterm exam is next ***Tuesday, May 23rd***, from ***7:00PM – 10:00PM***. Locations are divvied up by last (family) name:
  - `Abb` – `Pag`: Go to Hewlett 200.
  - `Par` – `Tak`: Go to Sapp 114.
  - `Tan` – `Val`: Go to Hewlett 101.
  - `Var` – `Yim`: Go to Hewlett 102.
  - `You` – `Zuc`: Go to Hewlett 103.
- You're responsible for Lectures 00 – 13 and topics covered in PS1 – PS5. Later lectures and problem sets won't be tested. The focus is on PS3 – PS5 and Lectures 06 – 13.
- The exam is closed-book, closed-computer, and limited-note. You can bring a double-sided, 8.5" × 11" sheet of notes with you to the exam, decorated however you'd like.

Your Questions

“What is your favourite mathematical theorem? Walk us through a proof.”

One of my favorite theorems is a tiny little one that I like not because it's super deep, but because the setup is just so beautiful. Let me show you!

“Recommendations for thought-provoking short stories and books, or just stories you have heard that made you challenge your existence / perception of reality?”

I've mentioned “Before the Law” on one of the practice exams and highly recommend it. It's a great short read to talk about.

Other ones: “Scott and Scurvy,” “Story of Your Life,” “Post-Operative Check,” and the Pulitzer-Prize winning article “The Fighter” from last year.

Back to CS103!

Another TM Design

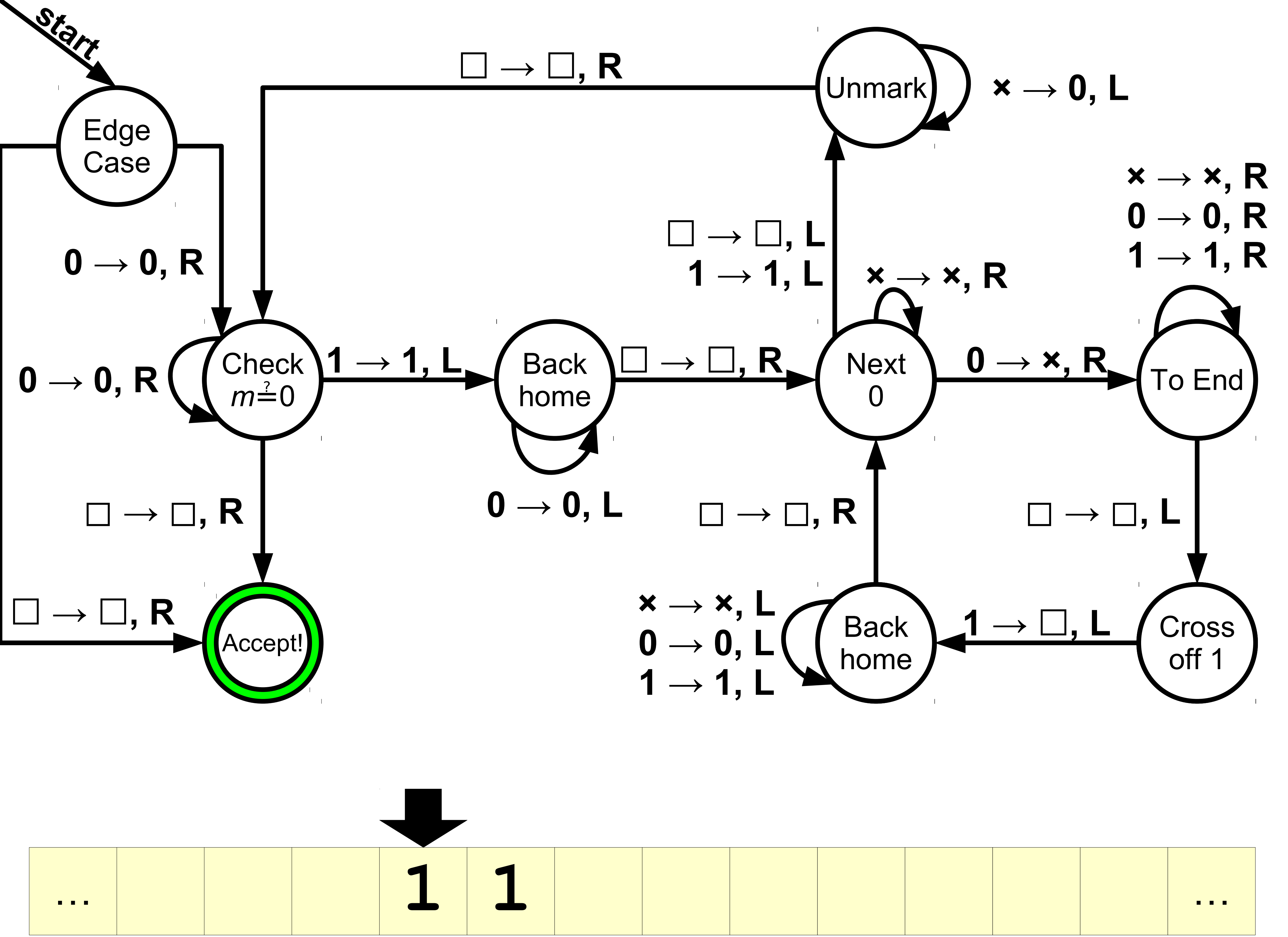
- Consider the following language over $\Sigma = \{0, 1\}$:

$$L = \{ 0^n 1^m \mid n, m \in \mathbb{N} \text{ and } m \text{ is a multiple of } n \}$$

- Is this language regular?
- How might we design a TM for this language?

An Observation

- We can recursively describe when one number m is a multiple of n:
  - If $m = 0$, then m is a multiple of n.
  - Otherwise, m is a multiple of n iff $m - n$ is a multiple of n.
- ***Idea:*** Repeatedly subtract n from m until m becomes zero (good!) or drops below zero (bad!)


start
Edge Case
0 → 0, R
□ → □, R
Check m ≟ 0
0 → 0, R
1 → 1, L
□ → □, R
Accept!
Back home
0 → 0, L
□ → □, R
Next 0
□ → □, L
1 → 1, L
× → ×, R
0 → ×, R
Unmark
× → 0, L
□ → □, R
To End
× → ×, R
0 → 0, R
1 → 1, R
□ → □, L
Cross off 1
1 → □, L
Back home
× → ×, L
0 → 0, L
1 → 1, L
□ → □, R
…
1
1
…

Concepts from Today

- Turing machines are a generalization of finite automata equipped with an infinite tape.
- It's often helpful to think recursively when designing Turing machines.
- It's often helpful to introduce new symbols into the tape alphabet.
- Watch for edge cases that might lead to infinite loops – though we'll say more about that later on.

Next Time

- ***TM Subroutines***
  - Combining multiple TMs together!
- ***The Church-Turing Thesis***
  - Just how powerful *are* Turing machines?