# Finite Automata

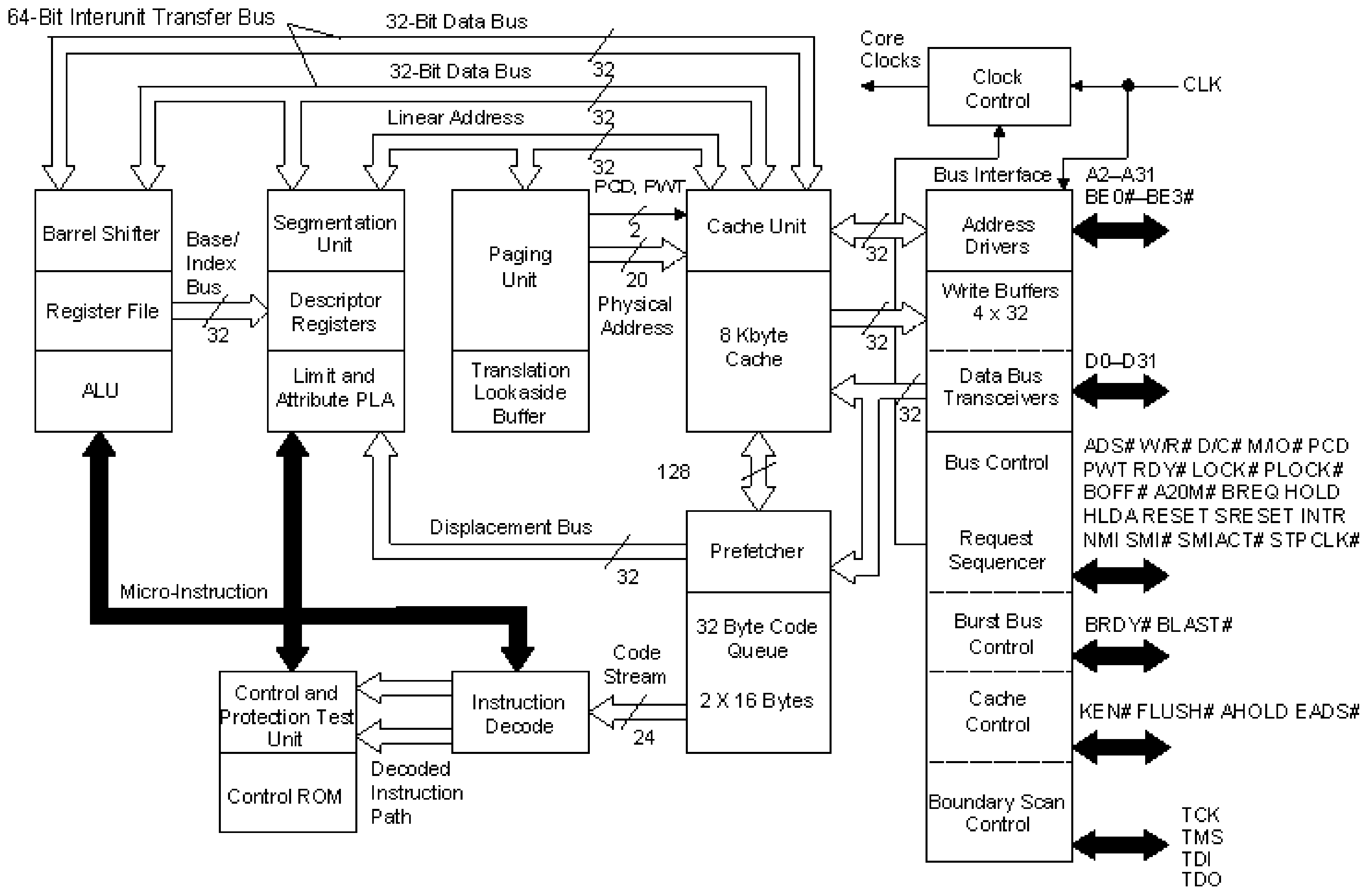
Part One

## Computability Theory

What problems can we solve with a computer?

What kind of computer?

# Computers are Messy



http://www.intel.com/design/intarch/prodbref/272713.htm

We need a simpler way of discussing computing machines.

An ***automaton*** (plural: ***automata***) is a mathematical model of a computing device.

# Why Build Models?

- ***Mathematical simplicity.***
  - It is significantly easier to manipulate our abstract models of computers than it is to manipulate actual computers.
- ***Intellectual robustness.***
  - If we pick our models correctly, we can make broad, sweeping claims about huge classes of real computers by arguing that they're just special cases of our more general models.

# Why Build Models?

- The models of computation we will explore in this class correspond to different conceptions of what a computer could do.
- ***Finite automata*** (next two weeks) are an abstraction of computers with finite resource constraints.
    - Provide upper bounds for the computing machines that we can actually build.
- ***Turing machines*** (later) are an abstraction of computers with unbounded resources.
    - Provide upper bounds for what we could ever hope to accomplish.

What problems can we solve with a computer?

What is a "problem?"

# Problems with Problems

- Before we can talk about what problems we can solve, we need a formal definition of a "problem."
- We want a definition that
    - corresponds to the problems we want to solve,
    - captures a large class of problems, and
    - is mathematically simple to reason about.
- No one definition has all three properties.

## Formal Language Theory

## Strings

- An ***alphabet*** is a finite, nonempty set of symbols called ***characters***.
  - Typically, we use the symbol **Σ** to refer to an alphabet.
- A ***string over an alphabet Σ*** is a finite sequence of characters drawn from Σ.
- Example: If Σ = {**a**, **b**}, here are some valid strings over Σ:

  **a** **aabaaabbabaaabaaaabbb** **abbababba**

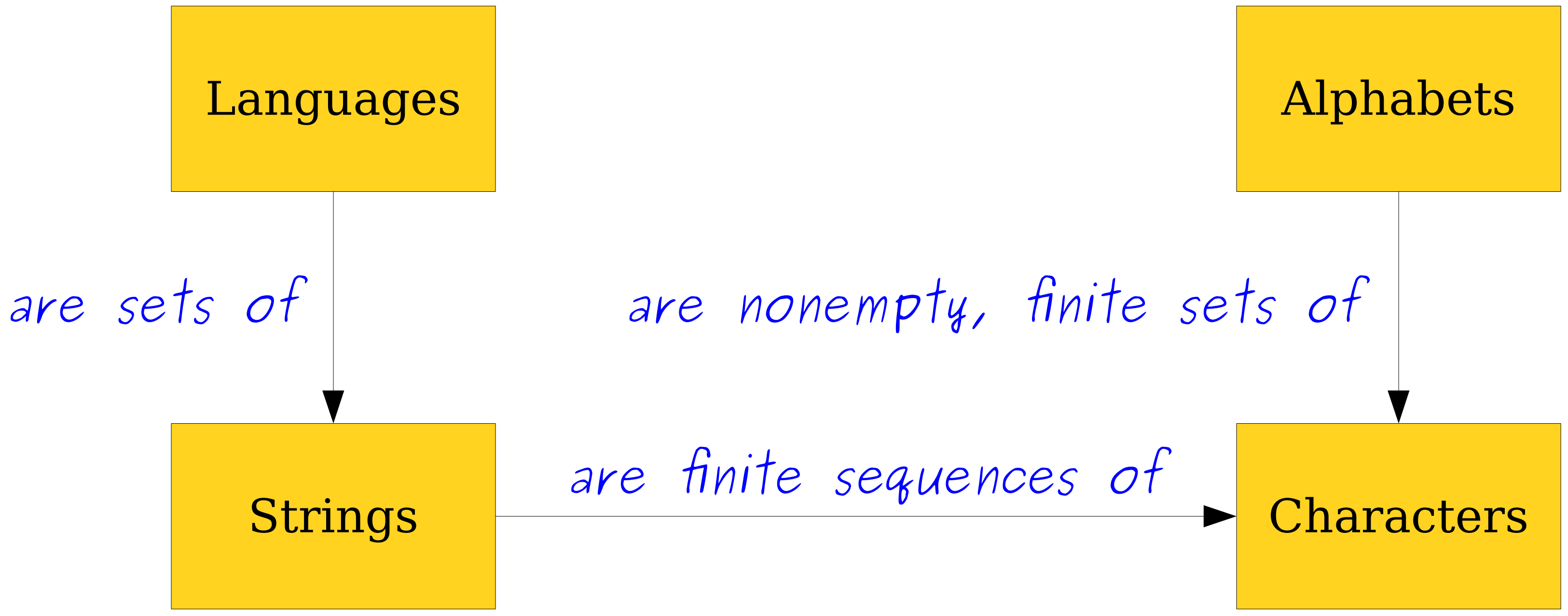
- The ***empty string*** has no characters and is denoted **ε**.
- Calling attention to an earlier point: since all strings are finite sequences of characters from Σ, you cannot have a string of infinite length.

# Languages

- A ***formal language*** is a set of strings.
- We say that $L$ is a ***language over Σ*** if it is a set of strings over Σ.
- Example: The language of palindromes over Σ = {**a**, **b**, **c**} is the set
  - {ε, **a**, **b**, **c**, **aa**, **bb**, **cc**, **aaa**, **aba**, **aca**, **bab**, … }
- The set of all strings composed from letters in Σ is denoted **Σ***.
- Formally, we say that $L$ is a language over Σ if $L \subseteq \Sigma^*$.

# To Recap

- ***Languages*** are sets of strings.
- ***Strings*** are sequences of characters.
- ***Characters*** are individual symbols.
- ***Alphabets*** are sets of characters.



## The Model

- ***Fundamental Question:*** For what languages $L$ can you design an automaton that takes as input a string, then determines whether the string is in $L$?
- The answer depends on the choice of $L$, the choice of automaton, and the definition of “determines.”
- In answering this question, we’ll go through a whirlwind tour of models of computation and see how this seemingly abstract question has very real and powerful consequences.

## To Summarize

- An ***automaton*** is an idealized mathematical computing machine.
- A ***language*** is a set of strings, a ***string*** is a (finite) sequence of characters, and a ***character*** is an element of an ***alphabet***.
- ***Goal:*** Figure out in which cases we can build automata for particular languages.

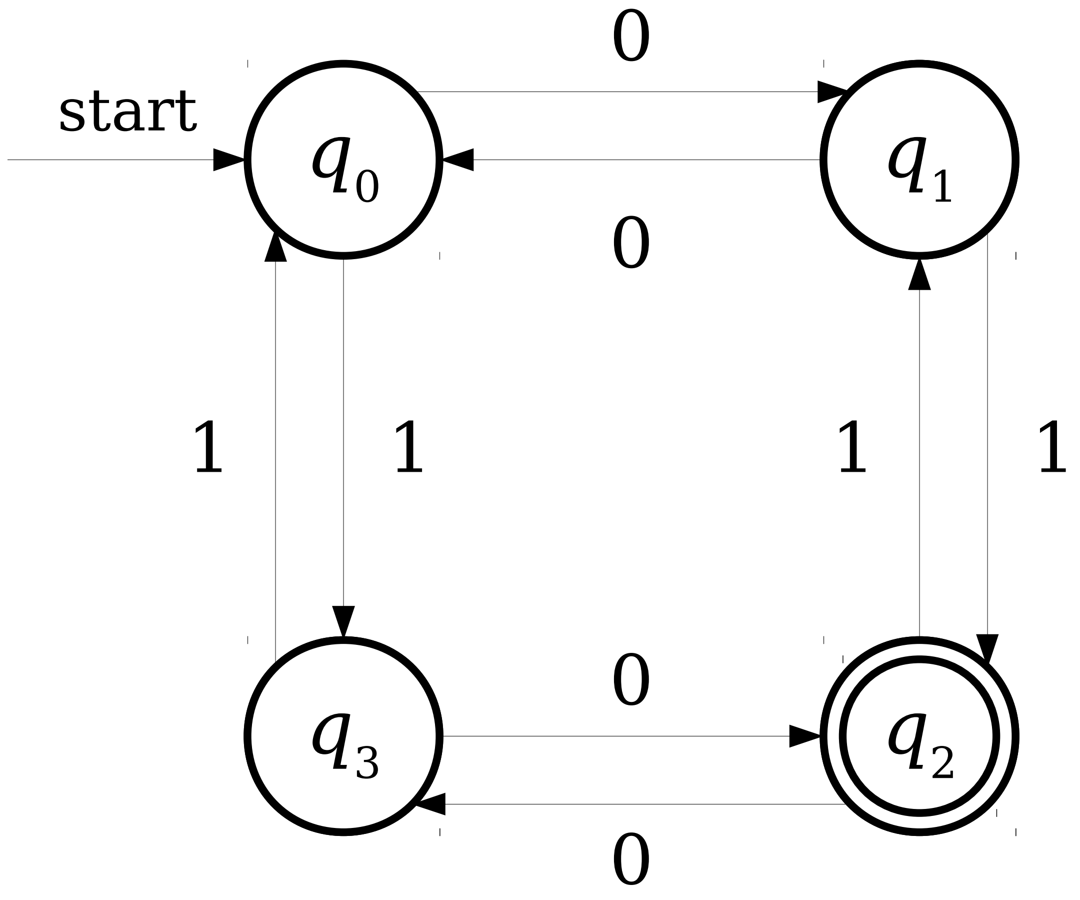
What problems can we solve with a computer?

## Finite Automata

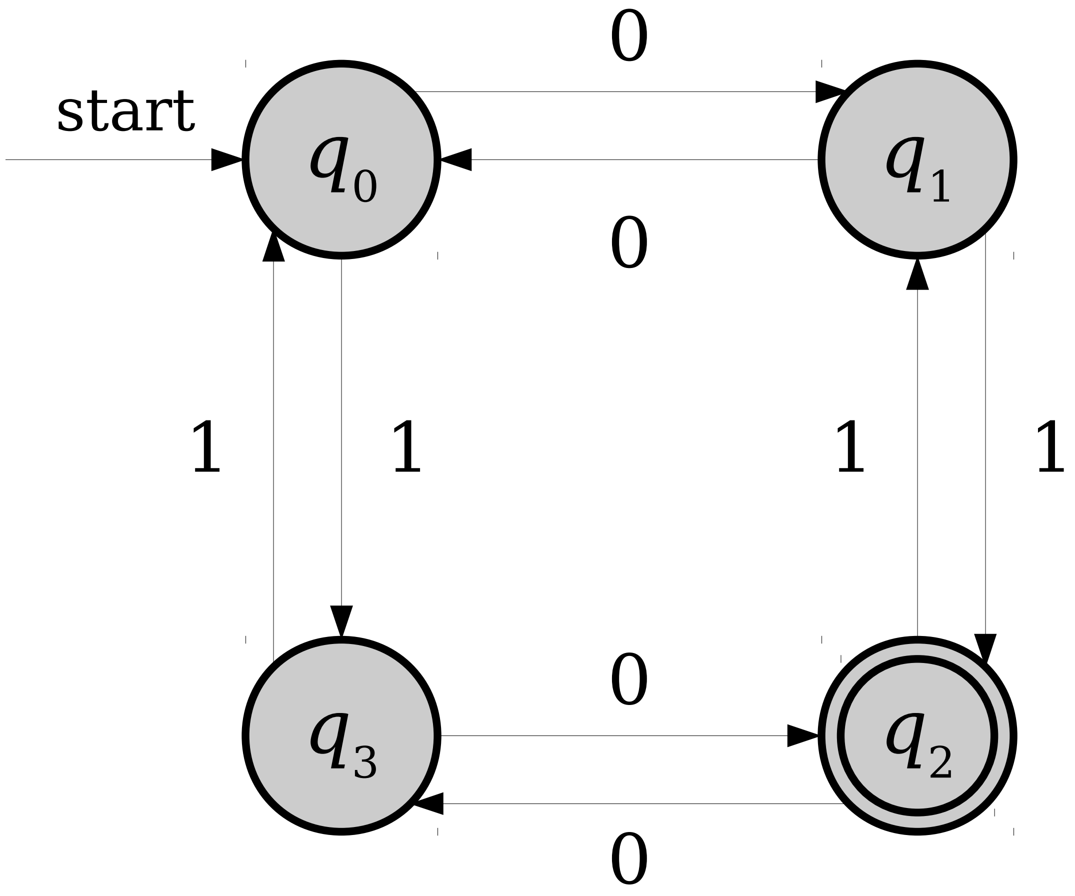
A ***finite automaton*** is a simple type of mathematical machine for determining whether a string is contained within some language.

Each finite automaton consists of a set of ***states*** connected by ***transitions***.

# A Simple Finite Automaton

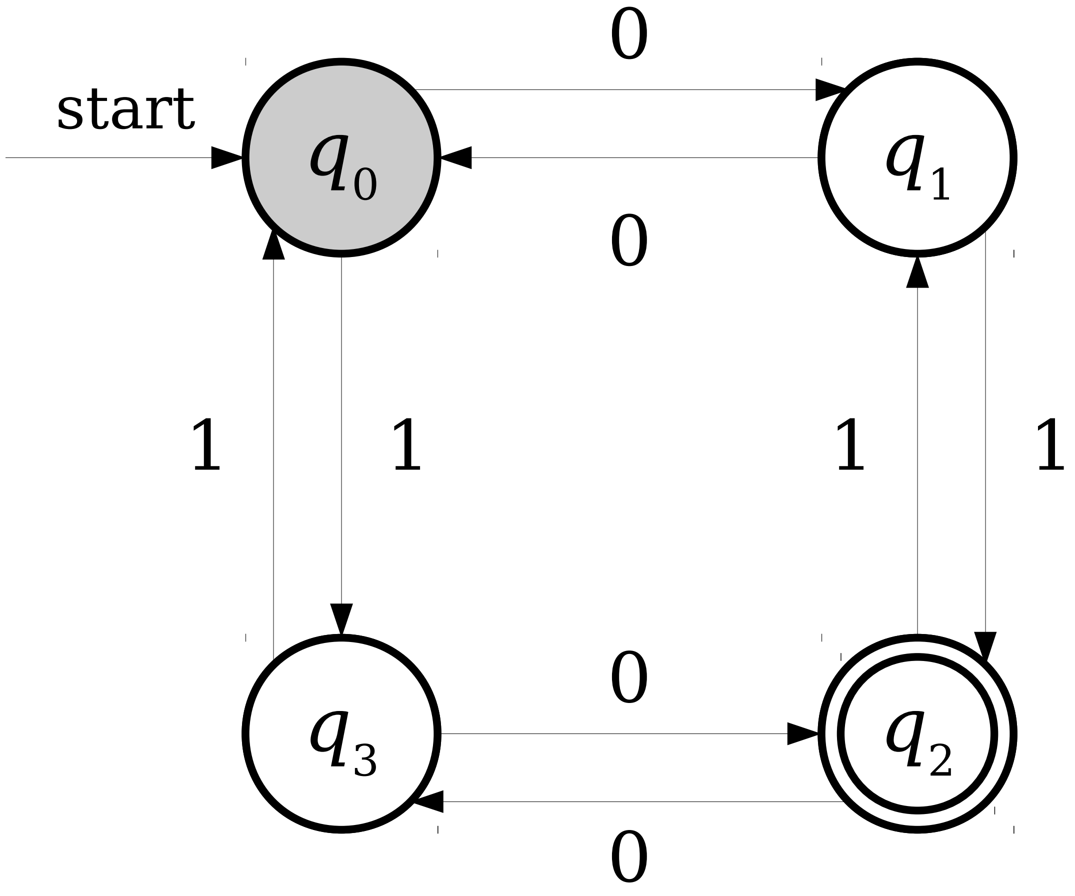


## A Simple Finite Automaton


$$\begin{tikzcd}
\text{start} \arrow[r] & q_0 \arrow[rr, bend left, "0"] \arrow[d, bend left, "1"] & & q_1 \arrow[ll, bend left, "0"] \arrow[d, bend left, "1"] \\
& q_3 \arrow[rr, bend left, "0"] \arrow[u, bend left, "1"] & & \boxed{q_2} \arrow[ll, bend left, "0"] \arrow[u, bend left, "1"]
\end{tikzcd}$$

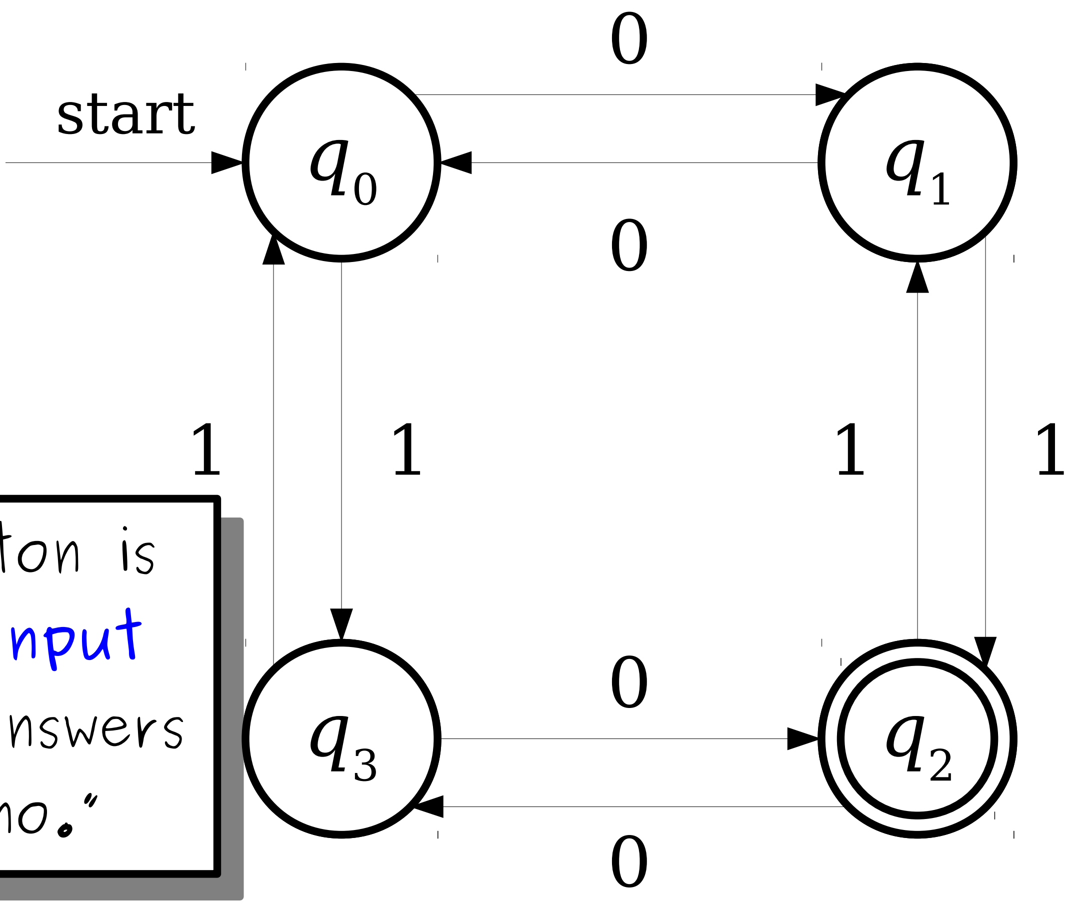
Each circle represents a state of the automaton.

# A Simple Finite Automaton


$$\begin{tikzcd}
\arrow[r, "\text{start}"] & q_0 \arrow[r, bend left, "0"] \arrow[d, bend left, "1"] & q_1 \arrow[l, bend left, "0"] \arrow[d, bend left, "1"] \\
 & q_3 \arrow[u, bend left, "1"] \arrow[r, bend left, "0"] & q_2 \arrow[l, bend left, "0"] \arrow[u, bend left, "1"]
\end{tikzcd}$$

One special state is designated as the **start state**.

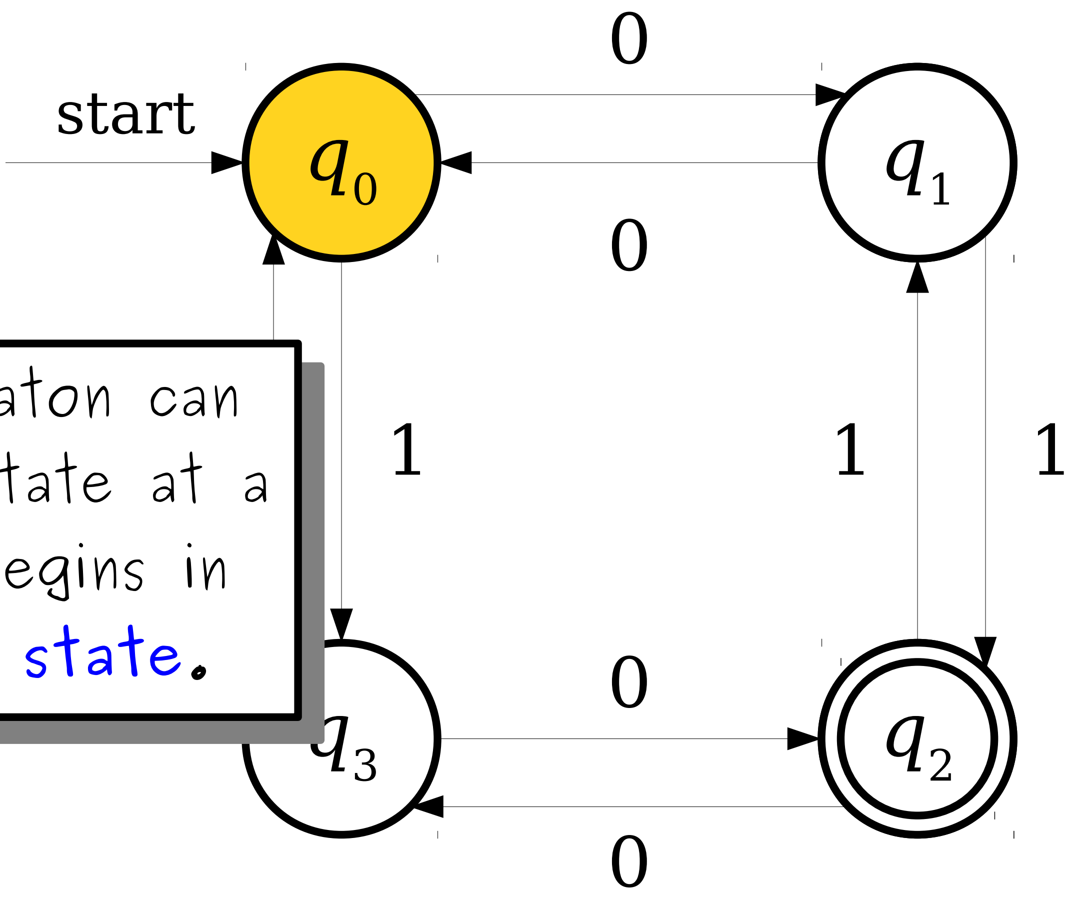
# A Simple Finite Automaton



The automaton is run on an input string and answers "yes" or "no."

**0 1 0 1 1 0**

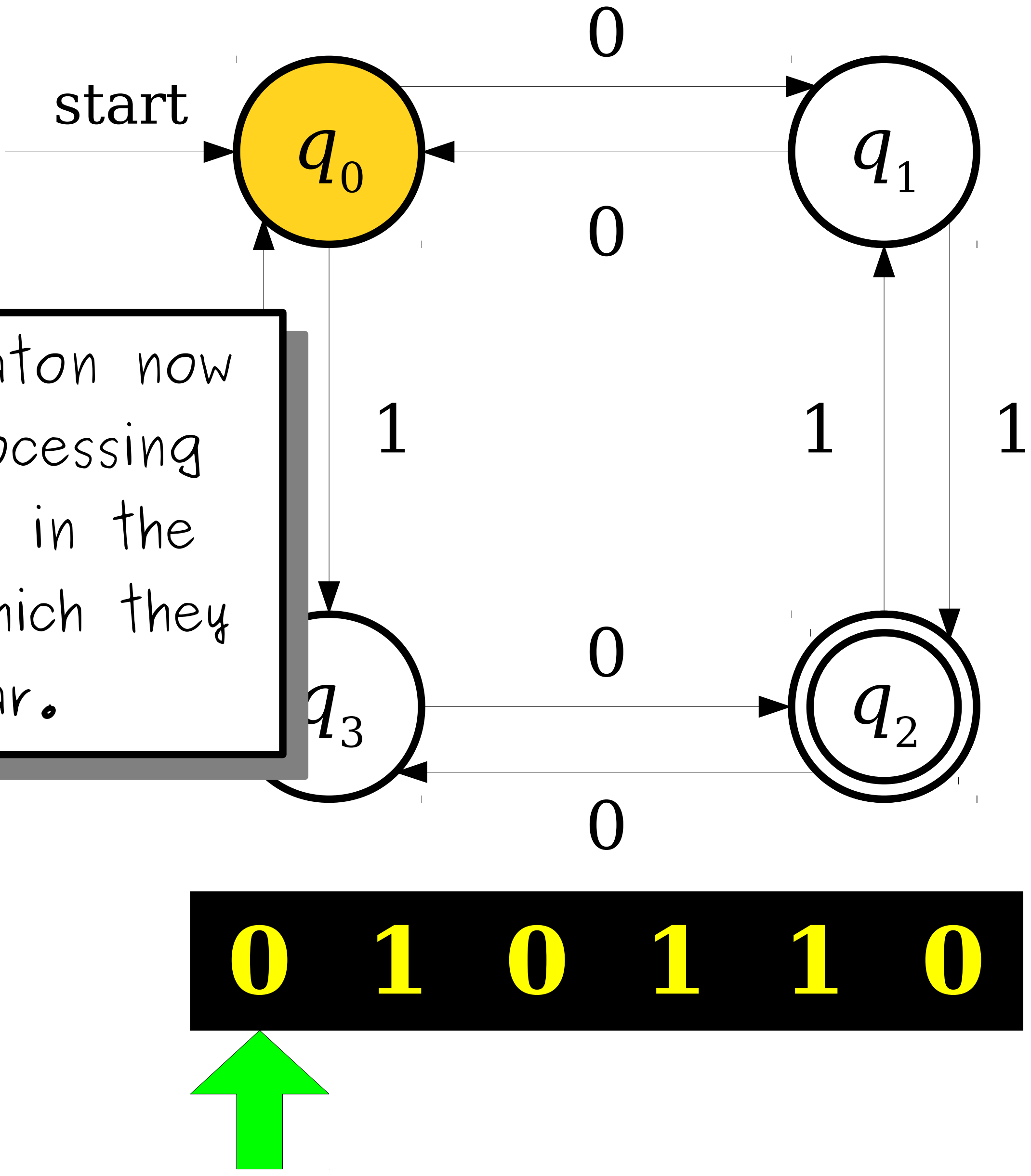
# A Simple Finite Automaton



The automaton can be in one state at a time. It begins in the start state.

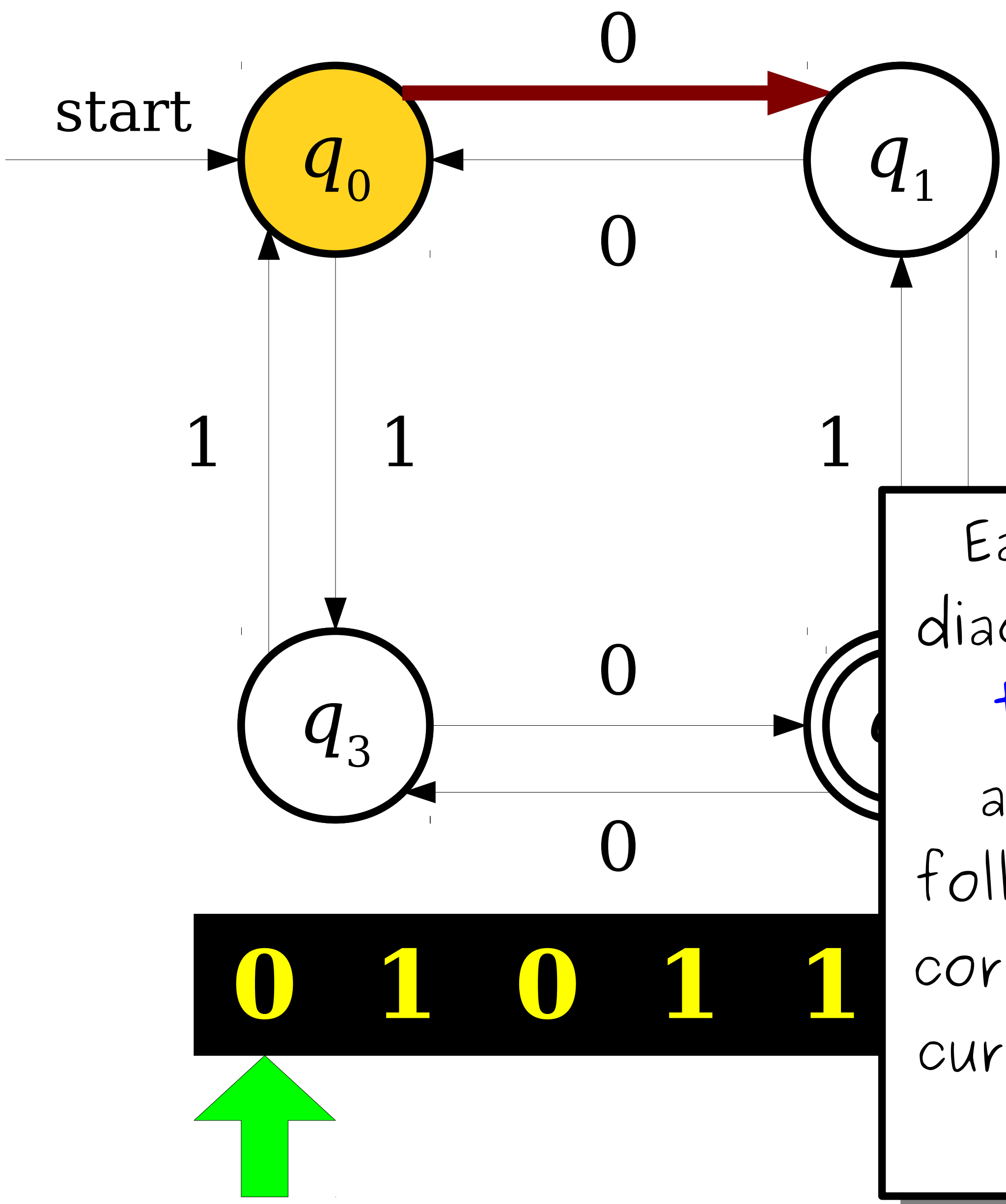
**0 1 0 1 1 0**

# A Simple Finite Automaton



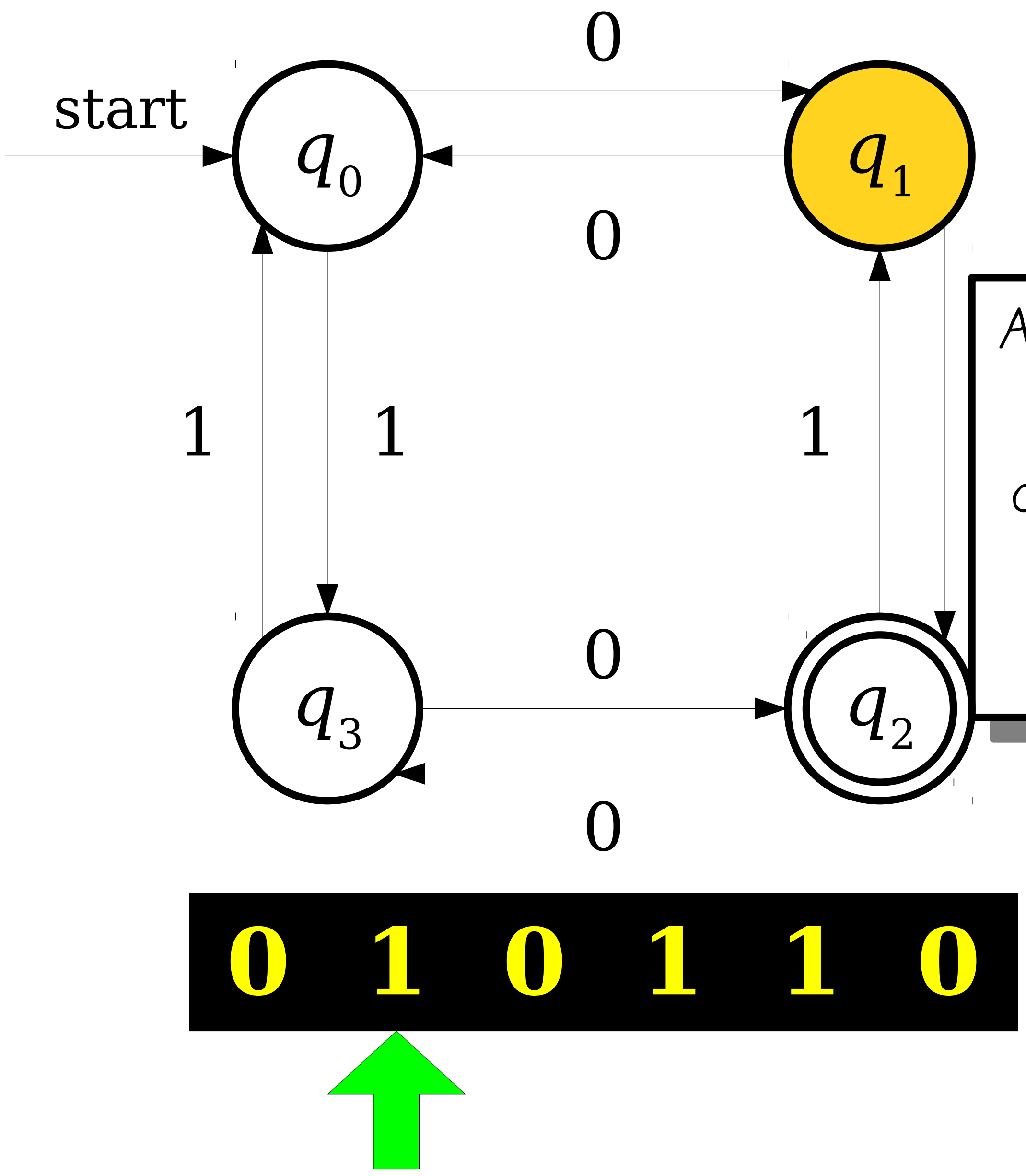
The automaton now begins processing characters in the order in which they appear.

# A Simple Finite Automaton



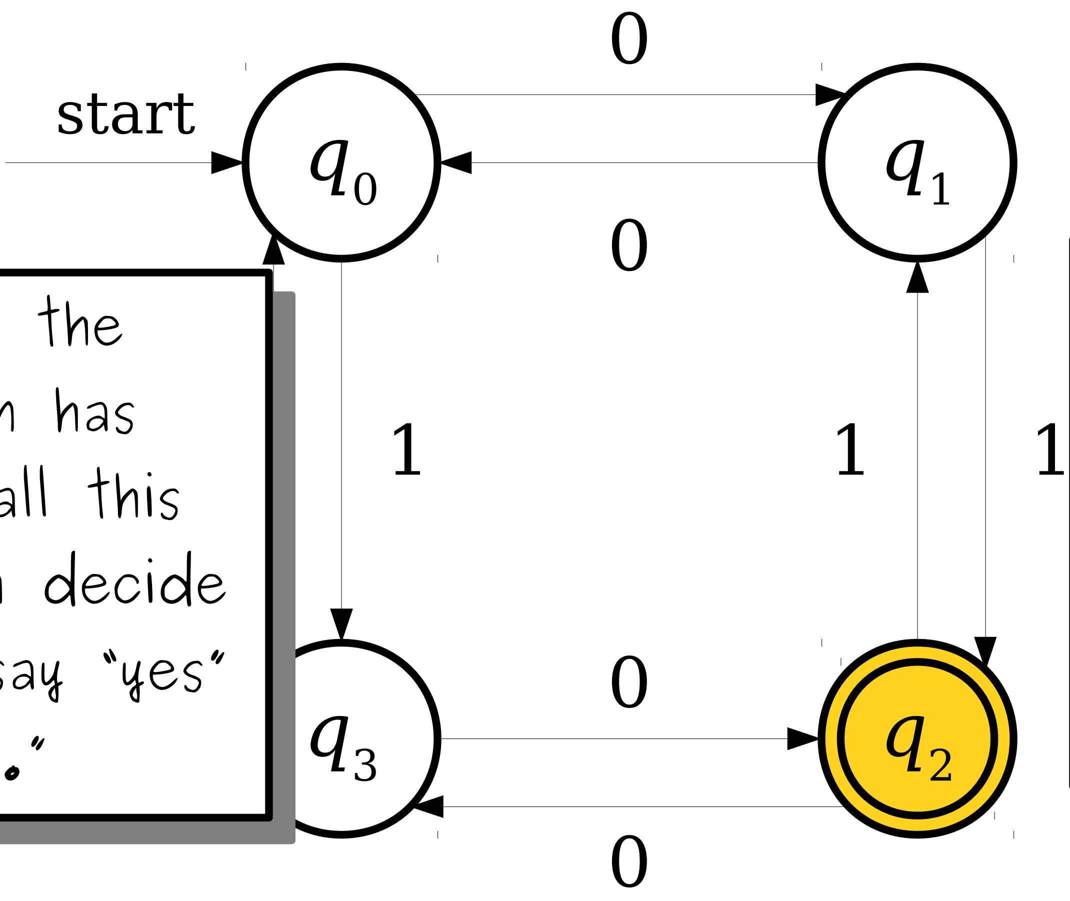
Each arrow in this diagram represents a **transition**. The automaton always follows the transition corresponding to the current symbol being read.

# A Simple Finite Automaton



After transitioning, the automaton considers the next symbol in the input.

# A Simple Finite Automaton

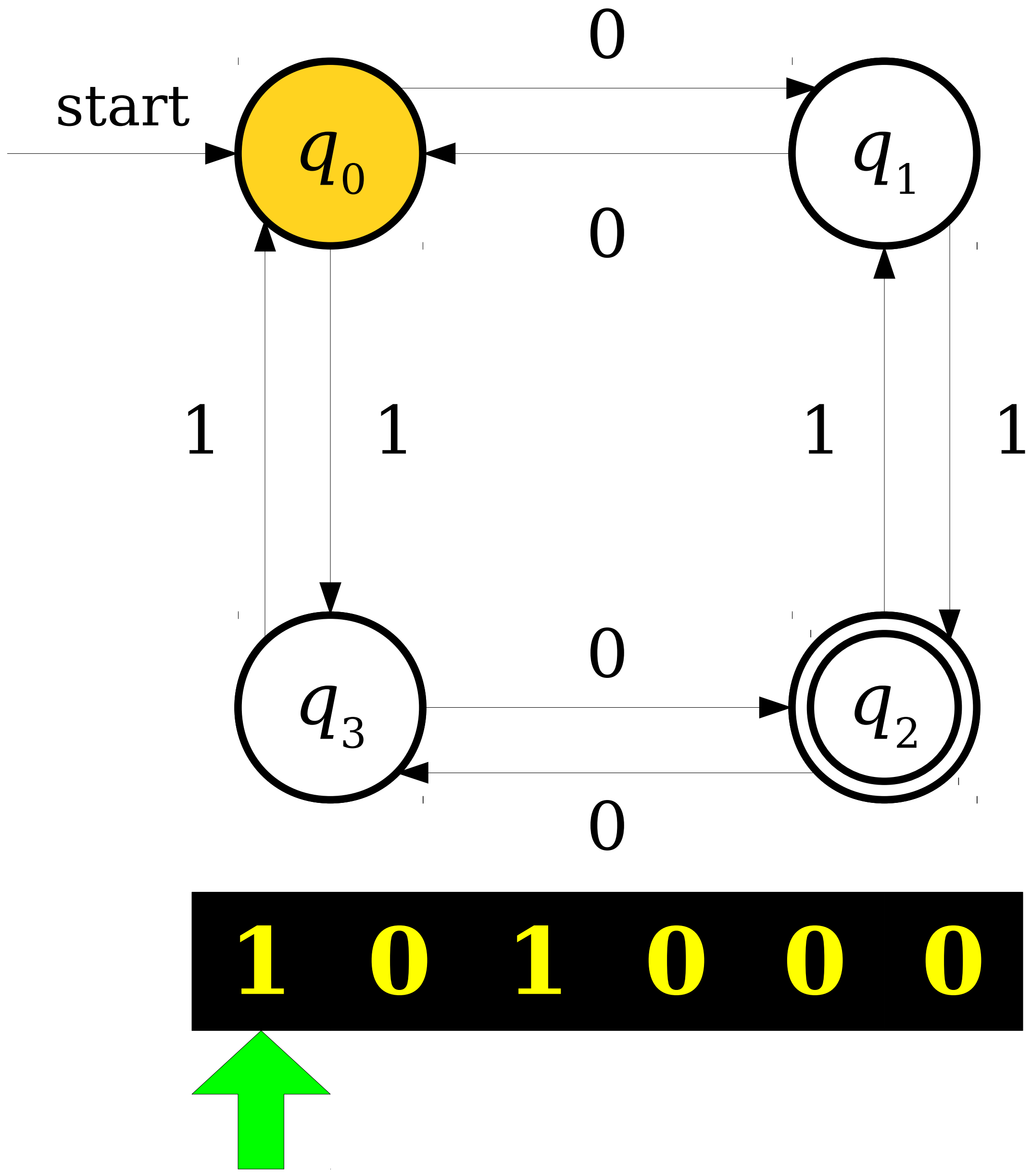


Now that the automaton has looked at all this input, it can decide whether to say "yes" or "no."

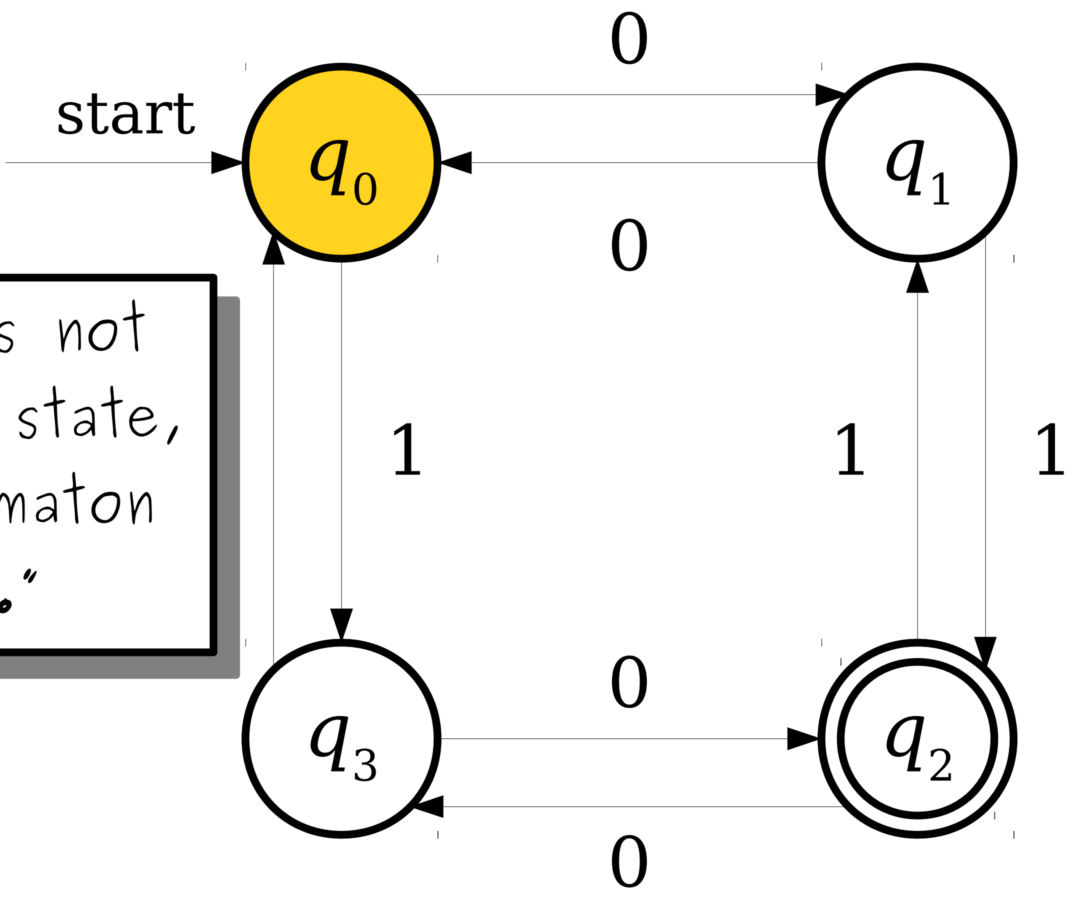
The double circle indicates that this state is an **accepting state**, so the automaton outputs "yes."

**0 1 0 1 1 0**

# A Simple Finite Automaton



# A Simple Finite Automaton

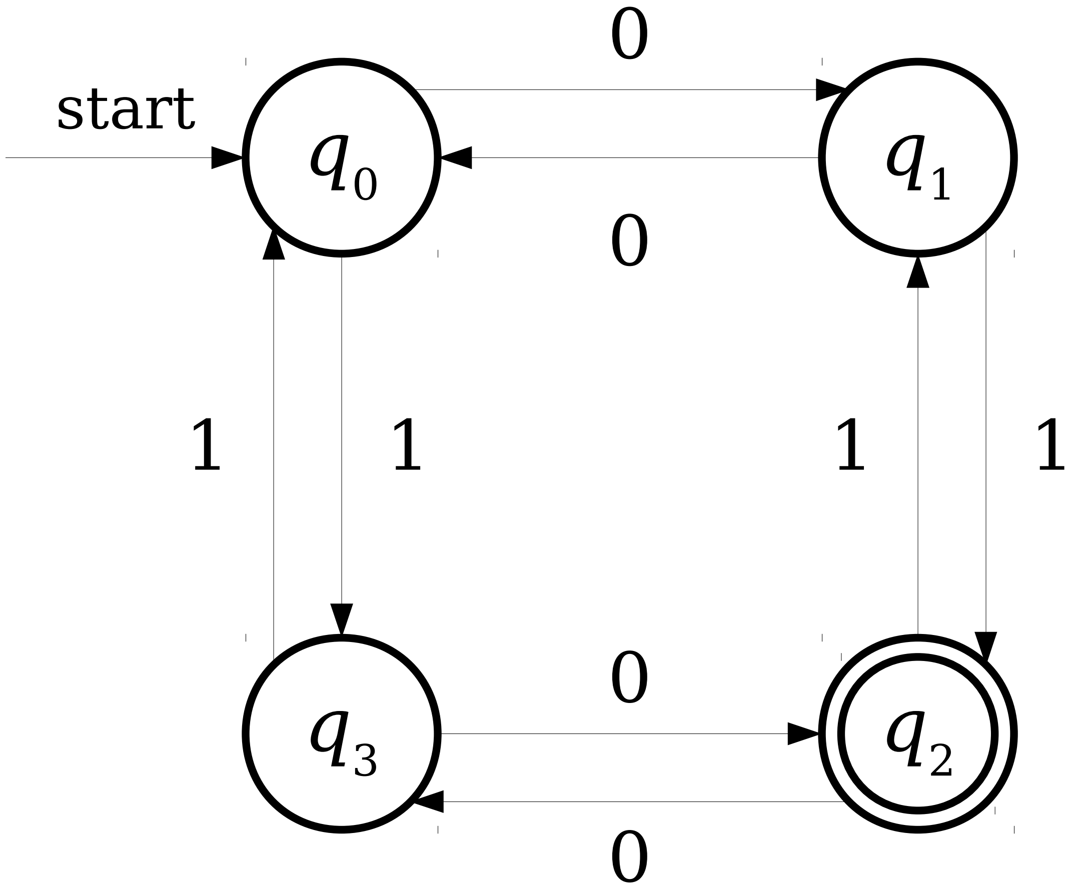


This state is not an accepting state, so the automaton says "no."

**1 0 1 0 0 0**

## A Simple Finite Automaton

$$\begin{tikzcd}
\arrow[r, "\text{start}"] & q_0 \arrow[r, bend left, "0"] \arrow[d, bend left, "1"] & q_1 \arrow[l, bend left, "0"] \arrow[d, bend left, "1"] \\
& q_3 \arrow[r, bend left, "0"] \arrow[u, bend left, "1"] & (q_2) \arrow[l, bend left, "0"] \arrow[u, bend left, "1"]
\end{tikzcd}$$



Try it yourself!
Does the automaton **accept** (say yes) or **reject** (say no)?

**1 1 0 1 1 1 0 0**

# The Story So Far

- A ***finite automaton*** is a collection of ***states*** joined by ***transitions***.
- Some state is designated as the ***start state***.
- Some states are designated as ***accepting states***.
- The automaton processes a string by beginning in the start state and following the indicated transitions.
- If the automaton ends in an accepting state, it ***accepts*** the input.
- Otherwise, the automaton ***rejects*** the input.

## Time-Out For Announcements!

Stanford Women
in Computer Science

# CASUAL CS DINNER

{w}

Monday, October 30th from 5:30-6:30 PM at
AT&T Patio Lobby Lawn

**Come have dinner with CS students and faculty.
Everyone is welcome, especially students
just starting out in CS!**

Engineers for a Sustainable World
STANFORD UNIVERSITY

Are you interested in working on an engineering problem for a low-income community abroad? If so, consider applying to the **Engineers for a Sustainable World class!**

**What**: Engineers for a Sustainable World runs a two-quarter class in which groups of Stanford students work with social enterprises on an engineering project to solve an international development problem. In the summer, a few members from each team may be funded to travel and deploy their technology. This year's projects include a solar service station that provides low-cost energy in the Philippines, a monitoring system for micro-hydroelectric generators in Indonesia, and a solar power system to provide energy to a community in Borneo. See the **project description sheet** for more information.

**When**: The class runs during winter and spring quarters for 1-3 units and 3-5 units, respectively. The course number is CEE 177X/S for undergrads and CEE 277X/S for graduate students.

**Who**: If you are an undergraduate or graduate student at Stanford, you are eligible to apply! For each project, preference will be given to meet the needs of the team - see the **project description sheet**.

Please apply **here** by Friday, 11/3, at 11:59pm!!

## Problem Sets

- Problem Set Four solutions are now available.
  - We’ll get PS4 graded and returned as soon as we can.
- Problem Set Five is due this Friday at 2:30PM.
  - Ask questions on Piazza if you have them!
  - Stop by office hours with questions!

Your Questions

## “What's are some of the biggest benefits you've gotten out of learning math? Sometimes it can be hard to see the light at the end of the tunnel”

On a practical side, there's a certain "mathematical perspective" on things that you get by studying mathematics. There's a value placed in math on avoiding special cases, about finding the most general version of some concept, about looking for commonalities across disparate topics, etc. Applying those ideas to software design is hugely valuable.

On the super philosophical side, I think that studying math gives you a new perspective on truth. Mathematical truth is not legal truth is not ethical truth is not religious truth. All forms of truth have their own rules, and seeing the mathematical perspective (give rigorous definitions to concepts, apply well-vetted rules to manipulate those definitions, etc.) is valuable for understanding truth more broadly.

“What do you think the next significant advancement in technology will be (like HUGE, not just a new iPhone)?”

I have absolutely no idea. Some of the biggest, most revolutionary changes in software and hardware came from very unexpected places. (Think the Fast Fourier Transform, the Internet, etc.) I'm just as curious about this as you are!

“What's the recipe for you mom's cookies (pretty please)?”

## ***Yellow Cake Chocolate Chip Cookies***

These are a real nostalgia treat. They're not even vaguely close to healthy and are far too easy to make. You might think that using prebought mix is cheating. My response: “Don't laugh. It works.” ☺

I first got this recipe from my mom, who got it from one of her friends. The original recipe called for shortening rather than coconut oil, but I happened to have coconut oil on hand and made some modifications to make that work instead.

These are perfect for when the weather – or your mood – is cold and rainy.

### ***Ingredients***

- 1 box yellow cake mix
- 1/4 cup coconut oil
- 1 tbsp cooking oil
- 2 eggs
- 16oz package chocolate chips
- 1 tsp vanilla

### ***Directions***

1. Preheat oven to 375F
2. Mix all the ingredients except for the chocolate chips so that they're well-blended.
3. Stir in the chocolate chips.
4. Spoon tablespoons of the batter onto a greased baking sheet.
5. Bake for 12-15 minutes.

Back to CS103!

A finite automaton does ***not*** accept as soon as it enters an accepting state.

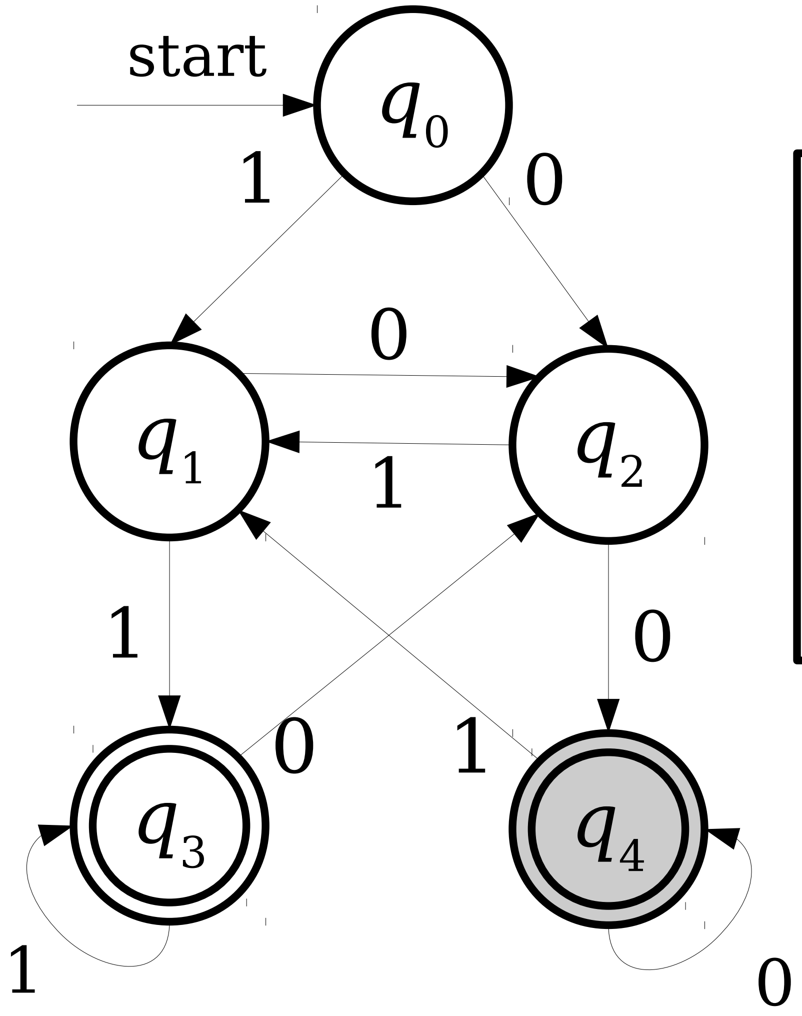
A finite automaton accepts if it ***ends*** in an accepting state.

# What Does This Accept?

$$\begin{tikzcd}
\text{start} \arrow[r] & q_0 \arrow[dl, "1"'] \arrow[dr, "0"] & \\
q_1 \arrow[rr, shift left, "0"] \arrow[d, "1"'] & & q_2 \arrow[ll, shift left, "1"] \arrow[d, "0"] \\
q_3 \arrow[loop left, "1"] \arrow[uurr, "0"] & & q_4 \arrow[loop right, "0"] \arrow[uull, "1"']
\end{tikzcd}$$

No matter where we start in the automaton, after seeing two 1's, we end up in accepting state $q_3$.

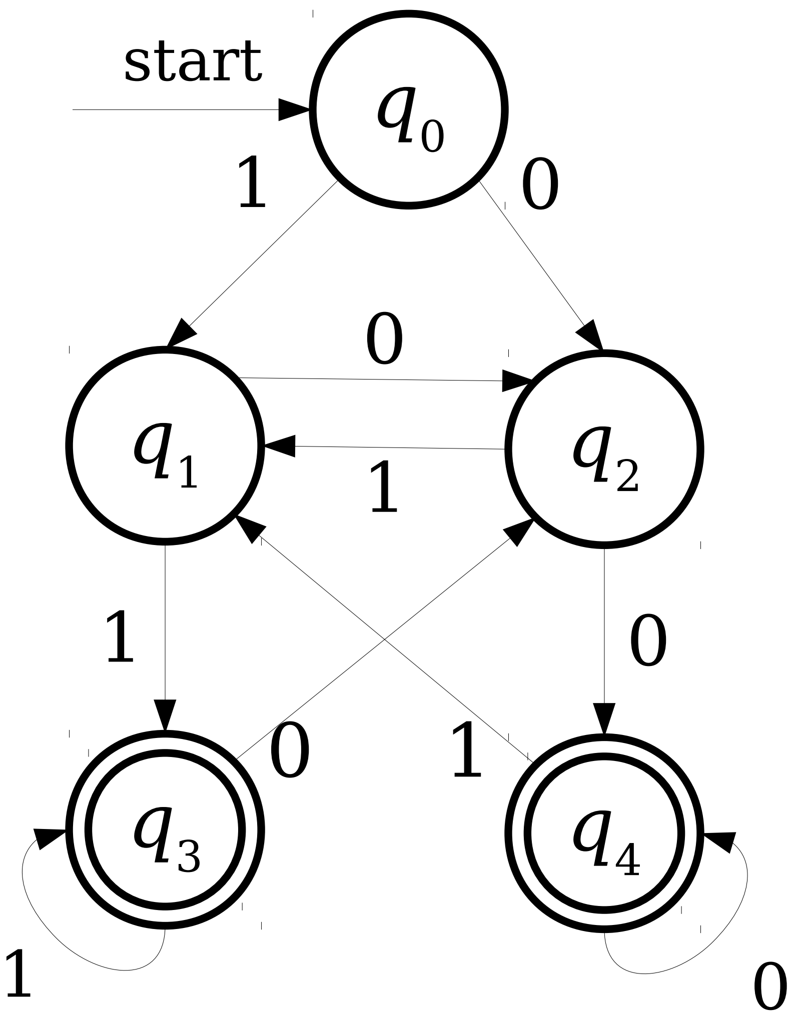
# What Does This Accept?



$$\begin{tikzcd}
\text{start} \arrow[r] & q_0 \arrow[dl, "1"'] \arrow[dr, "0"] & \\
q_1 \arrow[rr, shift left, "0"] \arrow[d, "1"'] & & q_2 \arrow[ll, shift left, "1"] \arrow[d, "0"] \\
\boxed{q_3} \arrow[loop left, "1"] \arrow[urr, "0"'] & & \boxed{q_4} \arrow[loop right, "0"] \arrow[ull, "1"]
\end{tikzcd}$$

No matter where we start in the automaton, after seeing two 0's, we end up in accepting state $q_5$.

# What Does This Accept?



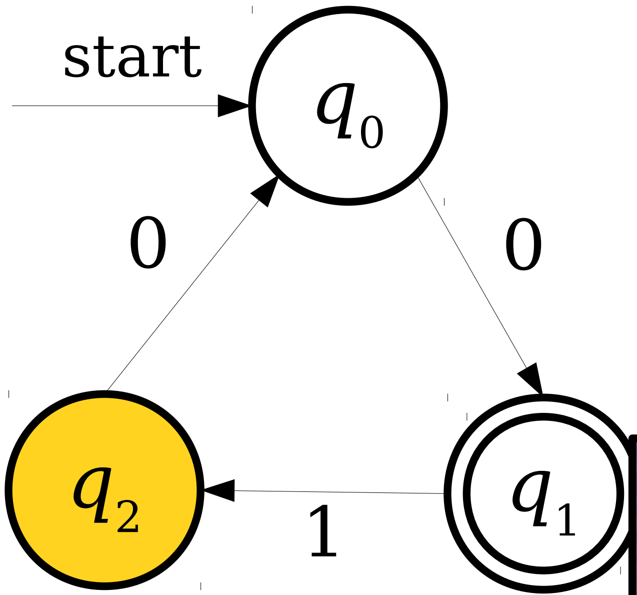
This automaton accepts a string iff the string ends in 00 or 11.

The ***language of an automaton*** is the set of strings that it accepts.

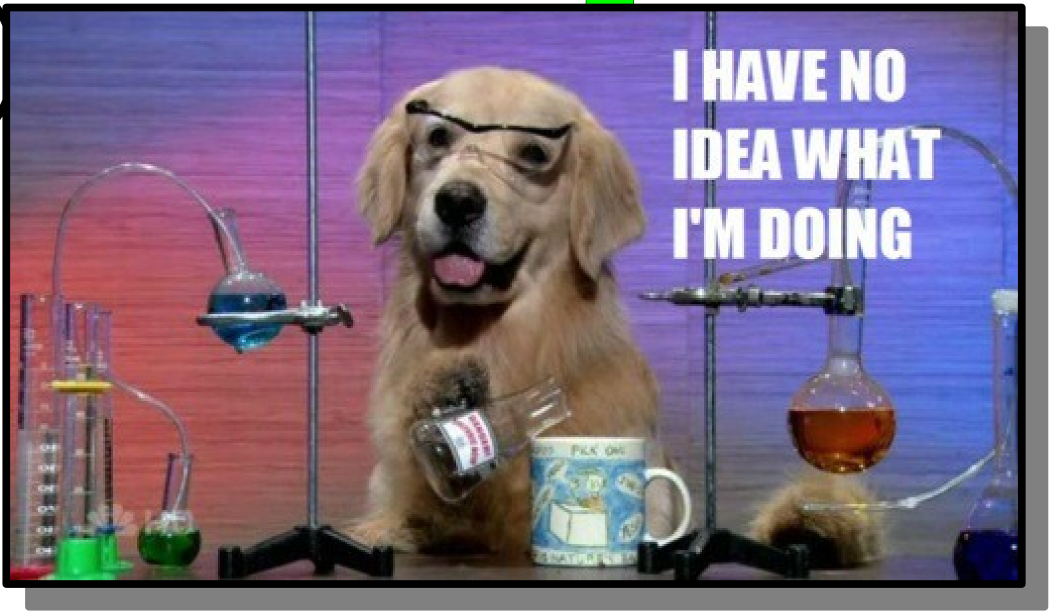
If $D$ is an automaton, we denote the language of $D$ as $\mathscr{L}(\boldsymbol{D})$.

$$\mathscr{L}(\boldsymbol{D}) = \{\ \boldsymbol{w} \in \Sigma^* \mid \boldsymbol{D}\ \textbf{accepts}\ \boldsymbol{w}\ \}$$

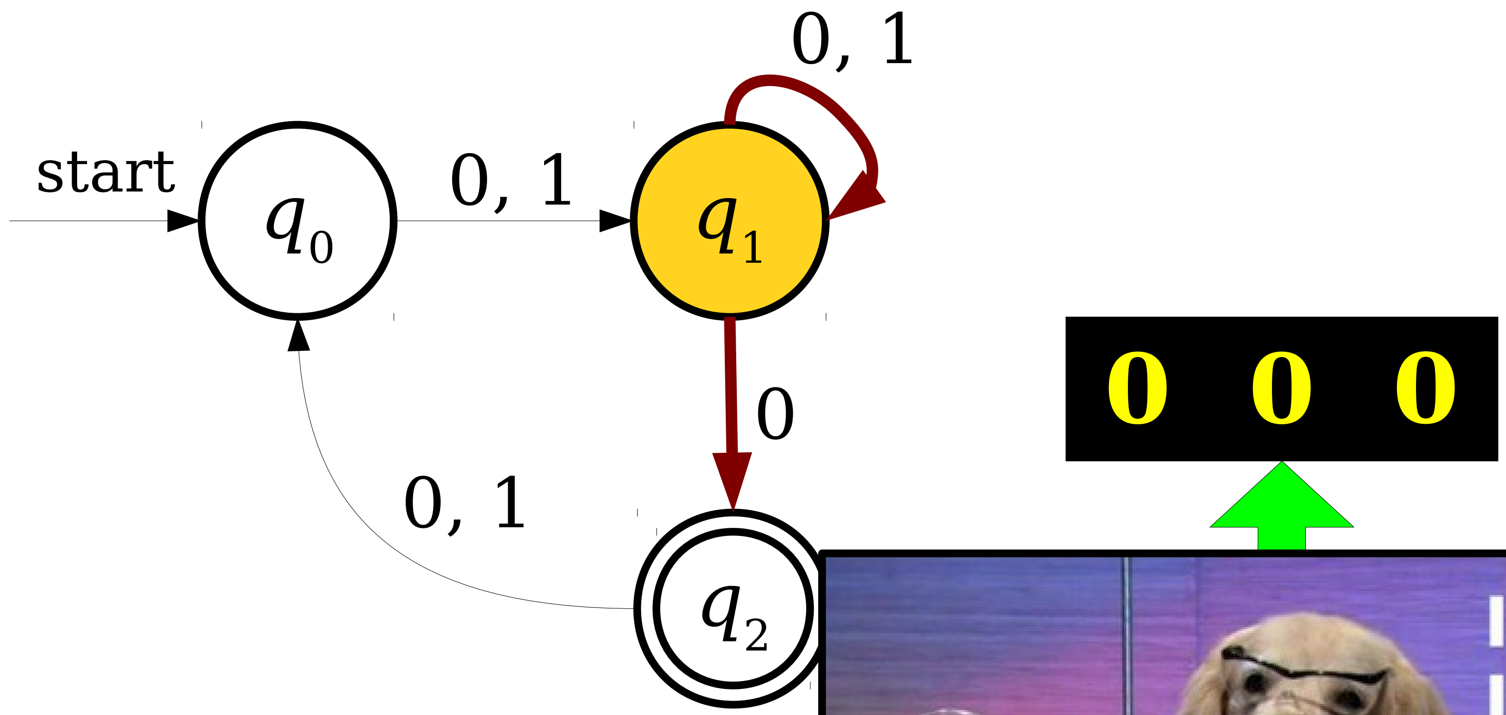
# A Small Problem

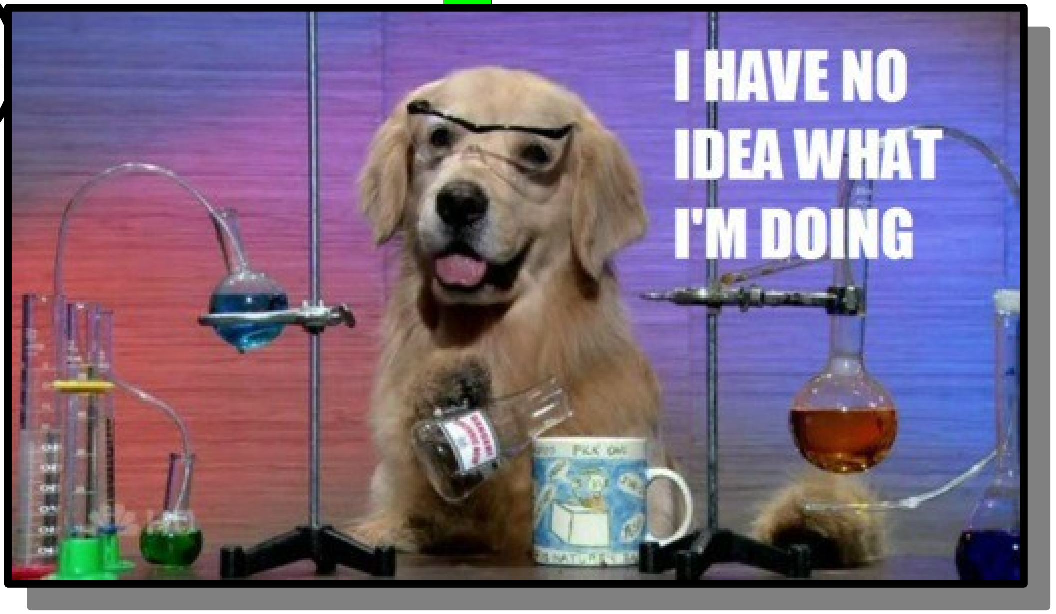


**0 1 1 0**



# Another Small Problem





## The Need for Formalism

- In order to reason about the limits of what finite automata can and cannot do, we need to formally specify their behavior in *all* cases.
- All of the following need to be defined or disallowed:
  - What happens if there is no transition out of a state on some input?
  - What happens if there are *multiple* transitions out of a state on some input?

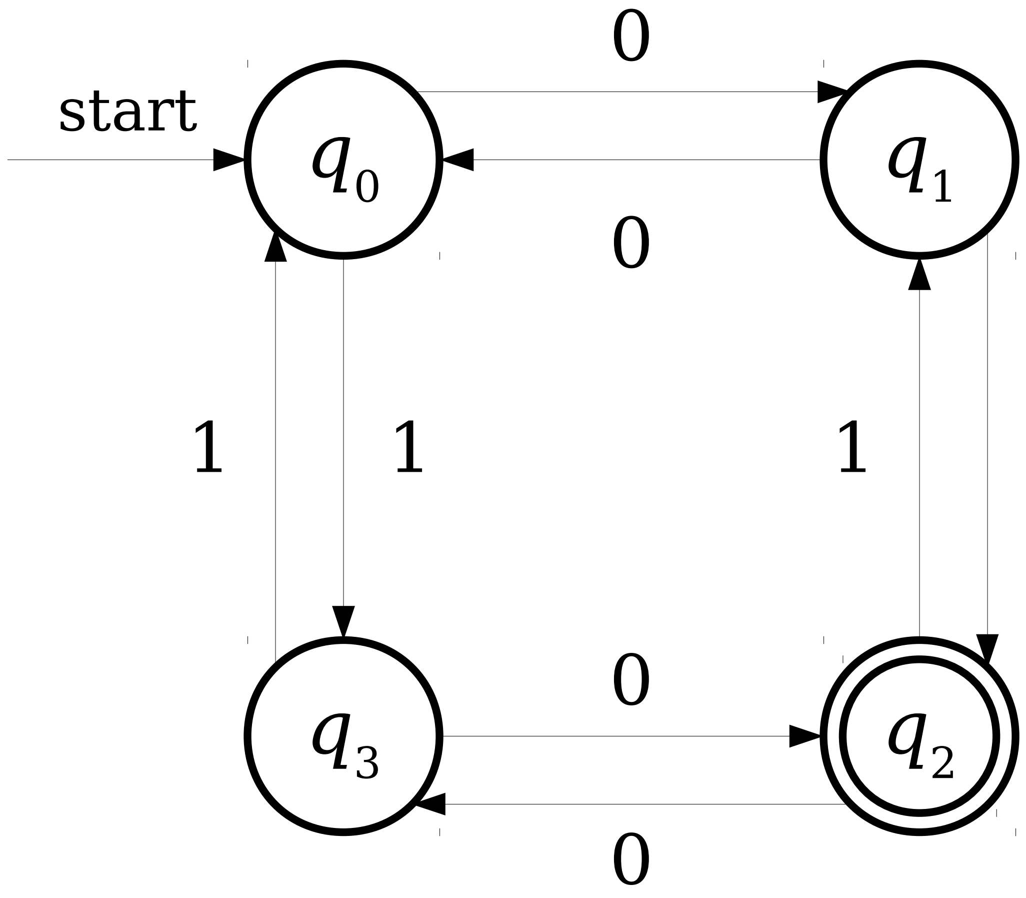
# DFAs

- A ***DFA*** is a
  - ***D***eterministic
  - ***F***inite
  - ***A***utomaton
- DFAs are the simplest type of automaton that we will see in this course.

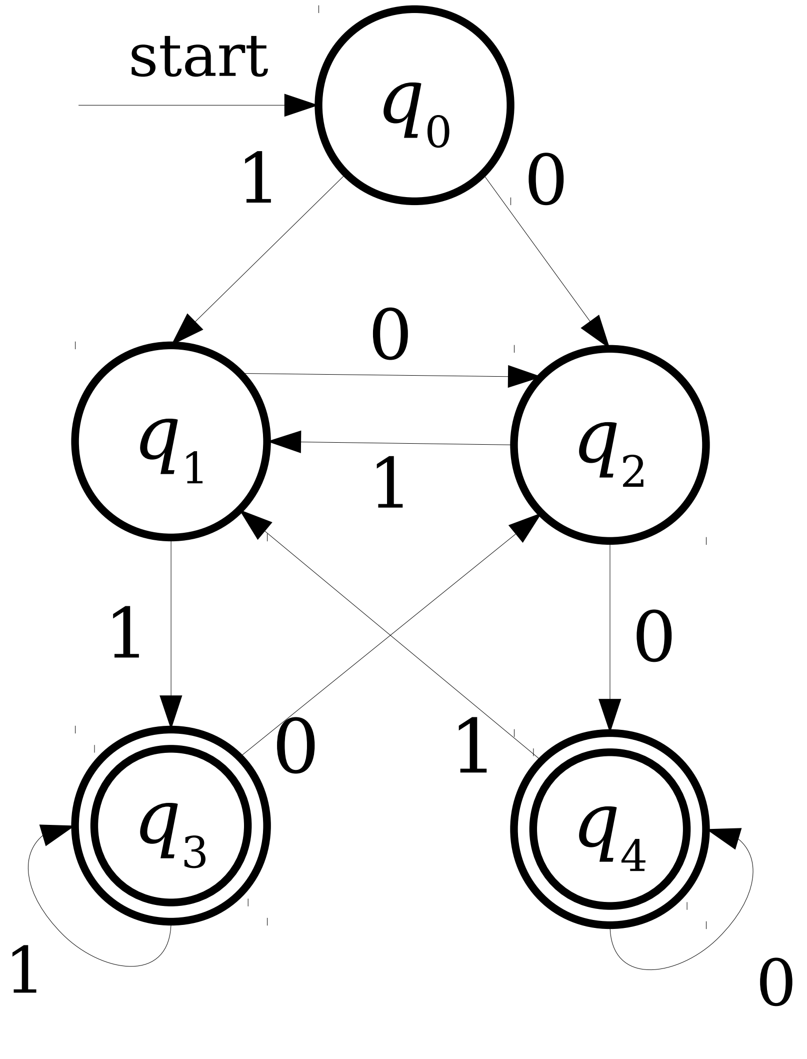
## DFAs, Informally

- A DFA is defined relative to some alphabet Σ.
- For each state in the DFA, there must be ***exactly one*** transition defined for each symbol in Σ.
    - This is the "deterministic" part of DFA.
- There is a unique start state.
- There are zero or more accepting states.

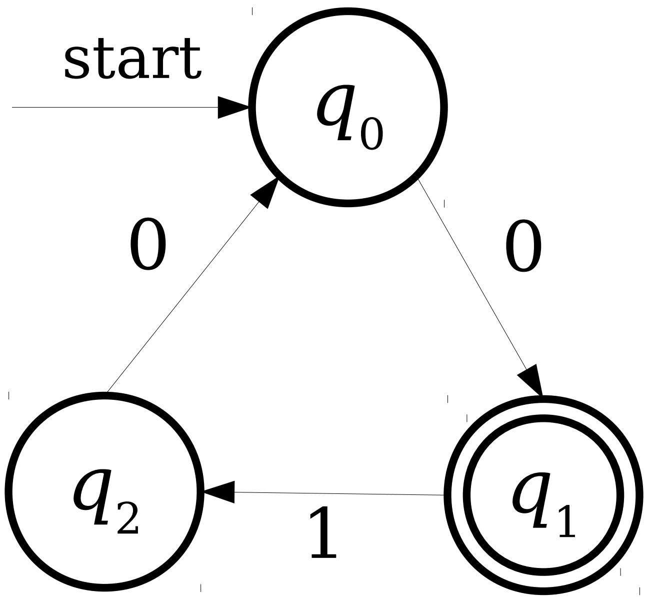
# Is this a DFA over {0, 1}?



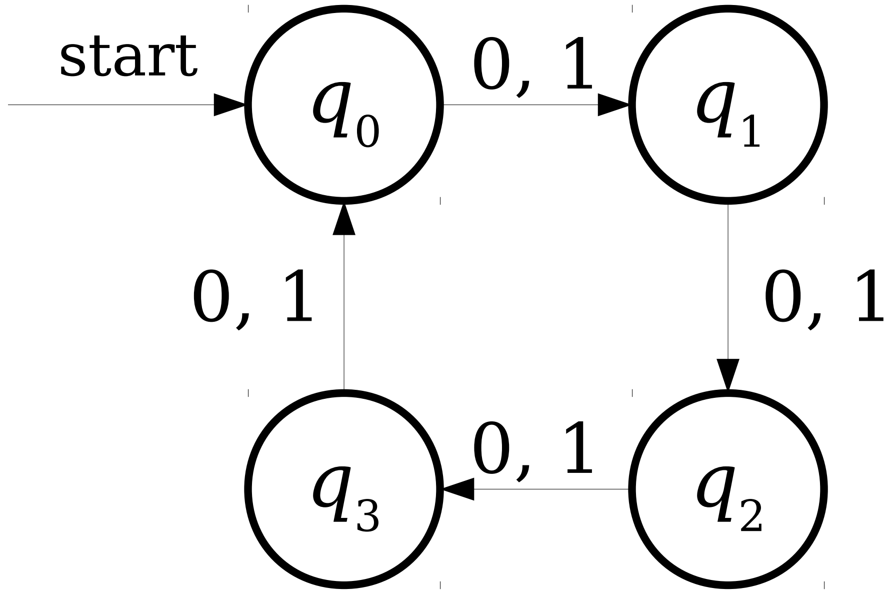
## Is this a DFA over {0, 1}?



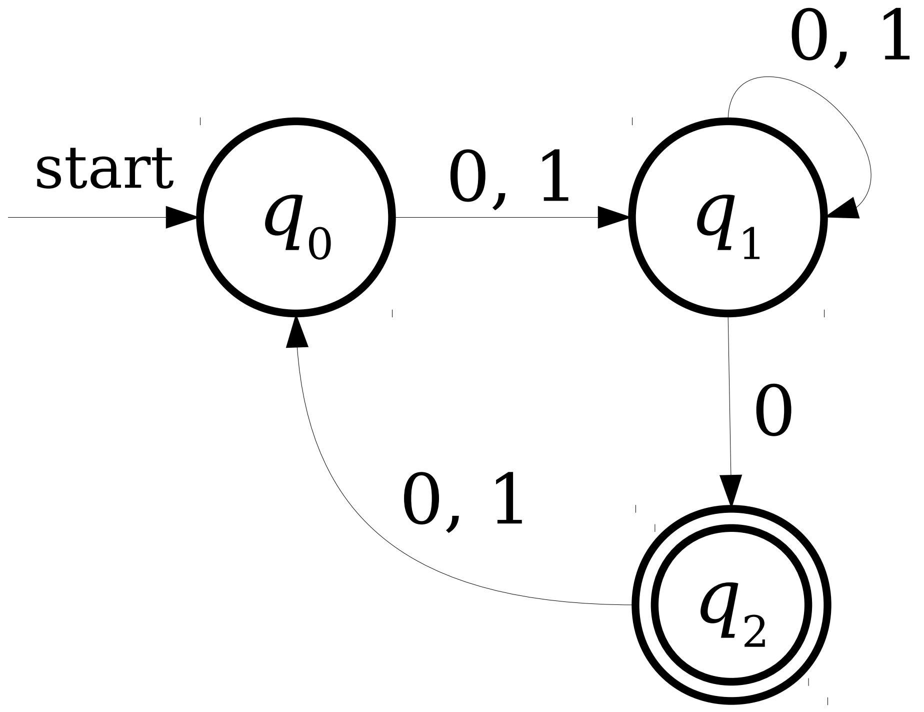
# Is this a DFA over {0, 1}?



# Is this a DFA over {0, 1}?


$$\begin{tikzcd}
\arrow[r, "\text{start}"] & q_0 \arrow[r, "{0, 1}"] & q_1 \arrow[d, "{0, 1}"] \\
& q_3 \arrow[u, "{0, 1}"] & q_2 \arrow[l, "{0, 1}"]
\end{tikzcd}$$

# Is this a DFA over {0, 1}?



## Is this a DFA?

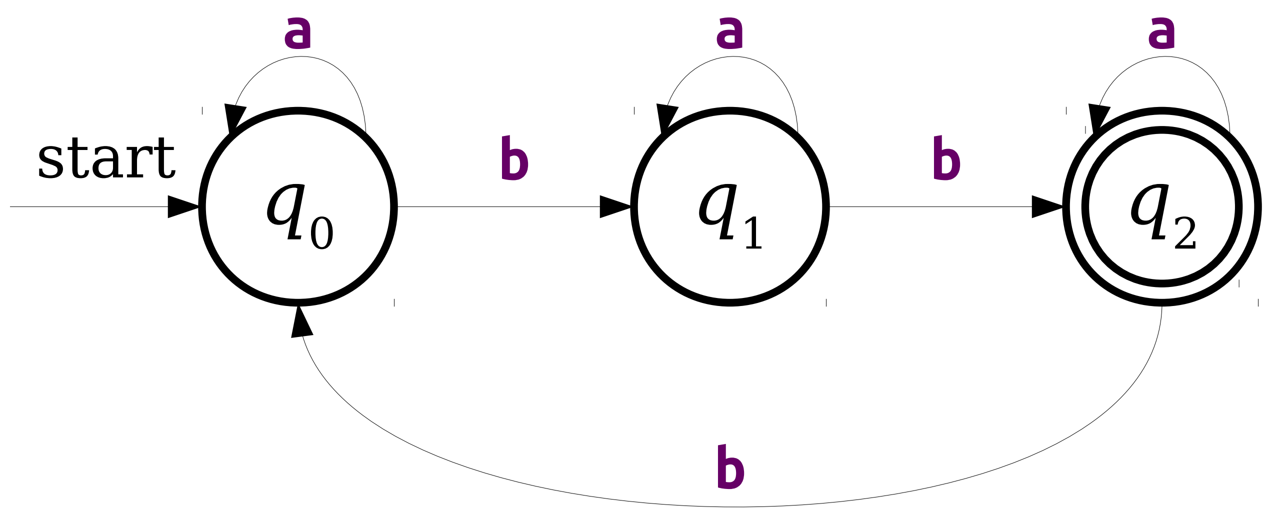
**D**rinking **F**amily of **A**ardvarks

# Designing DFAs

- At each point in its execution, the DFA can only remember what state it is in.
- ***DFA Design Tip:*** Build each state to correspond to some piece of information you need to remember.
  - Each state acts as a “memento” of what you're supposed to do next.
  - Only finitely many different states ≈ only finitely many different things the machine can remember.

# Recognizing Languages with DFAs

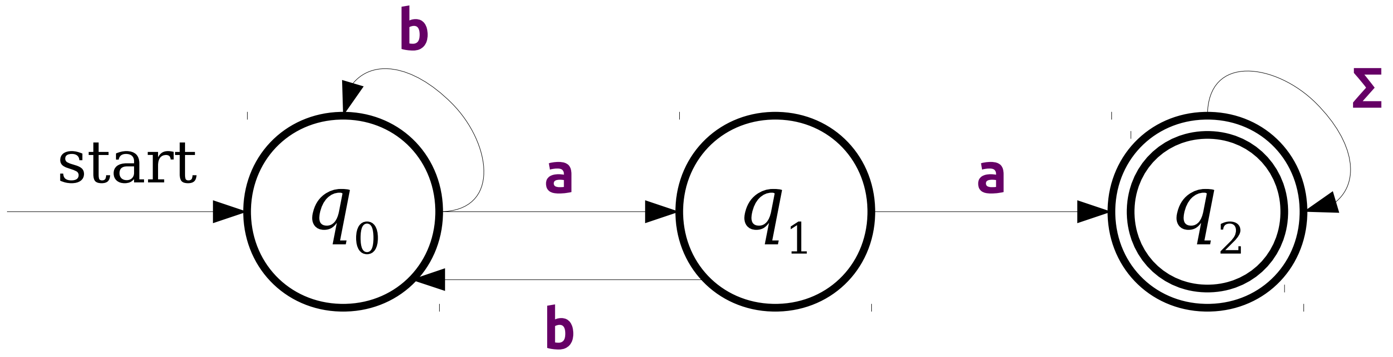
$L$ = { $w$ ∈ {**a**, **b**}*| the number of **b**'s in $w$ is congruent to two modulo three }



Each state remembers the remainder of the number of **b**s seen so far modulo three.

# Recognizing Languages with DFAs

$L = \{ w \in \{\mathtt{a}, \mathtt{b}\}^* \mid w \text{ contains } \mathtt{aa} \text{ as a substring } \}$



# More Elaborate DFAs

$L = \{ w \in \{$**a**, *****, **/**$\}^* \mid w$ represents a C-style comment $\}$

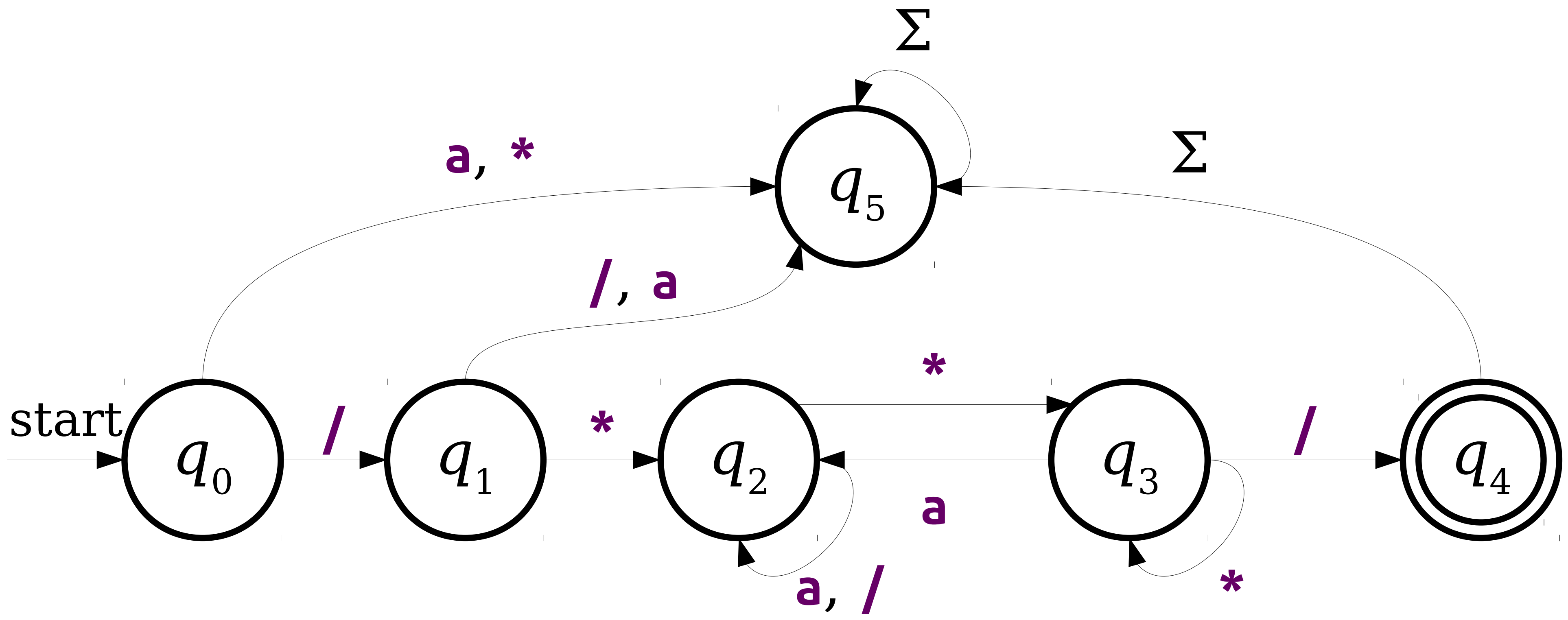
Let's have the **a** symbol be a placeholder for "some character that isn't a star or slash."

Try designing a DFA for comments! Here's some test cases to help you check your work:

| Accepted: | Rejected: |
|---|---|
| /*a*/ | /** |
| /**/ | /**/a/*aa*/ |
| /***/ | aaa/**/aa |
| /*aaa*aaa*/ | /*/ |
| /*a/a*/ | /**a/ |
|  | //aaaa |

# More Elaborate DFAs

$L = \{ w \in \{$**a**, *****, **/**$\}^* \mid w$ represents a C-style comment $\}$



Σ
a, *
Σ
/, a
start
q0
/
q1
*
q2
*
q3
/
q4
q5
a
a, /
*