Finite Automata

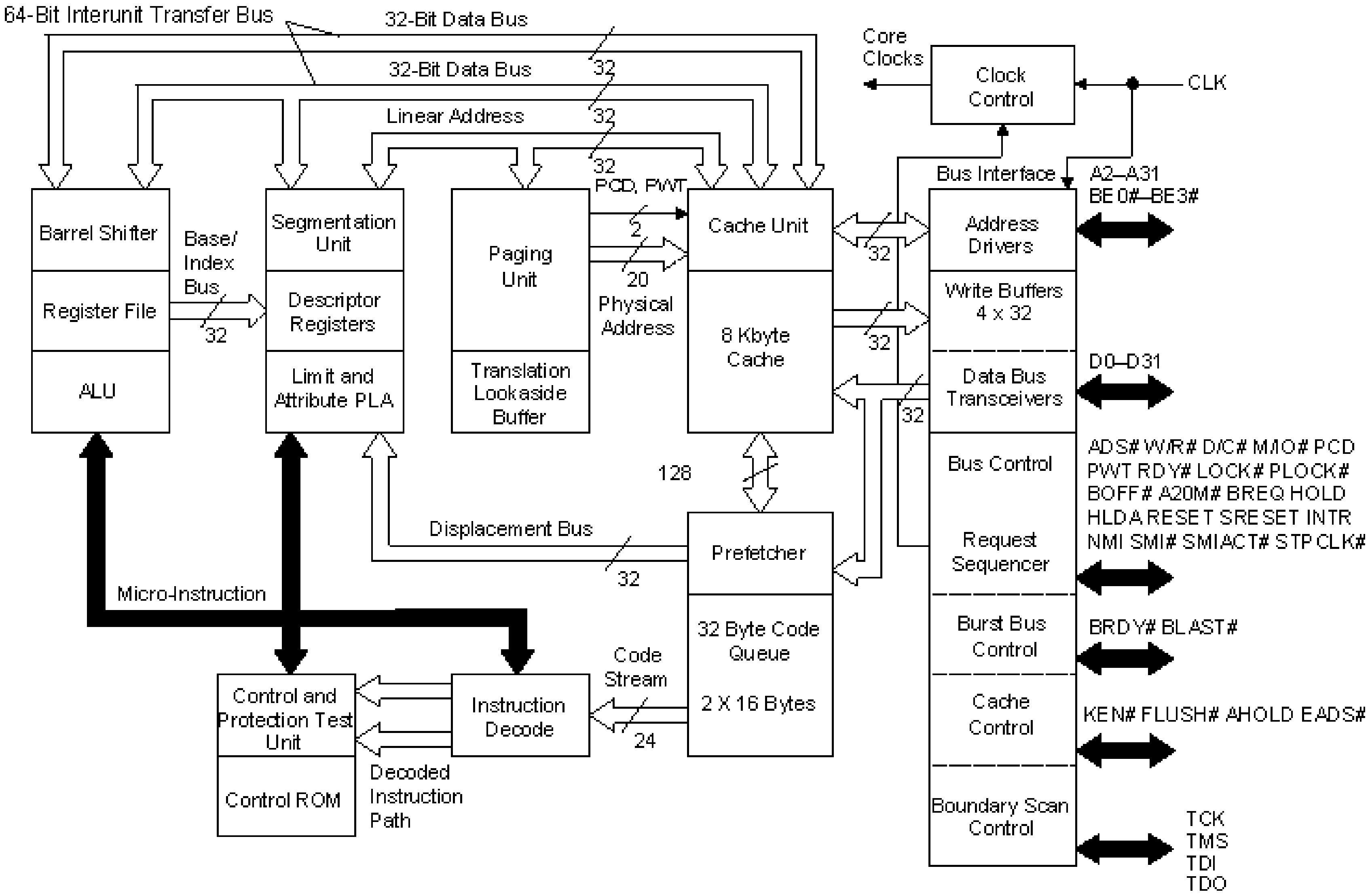
Part One

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 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**}, some valid strings over Σ include

a

aabaaabbabaaabaaaabbb

abbababba

- The ***empty string*** contains no characters and is denoted **ε**.

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 $L \subseteq \Sigma^*$.

The Model

- ***Fundamental Question:*** Given an alphabet Σ and a language L over Σ, in what cases can we build an automaton that determines which strings are in L?
- The answer depends on both the choice of L and the choice of automaton.
- The entire rest of the quarter will be dedicated to answering these questions.

To Summarize

- An ***automaton*** is an idealized mathematical computing machine.
- A ***language*** is a set of strings.
- The automata we will study will accept as input a string and (attempt to) determine whether that string is contained in a particular language.

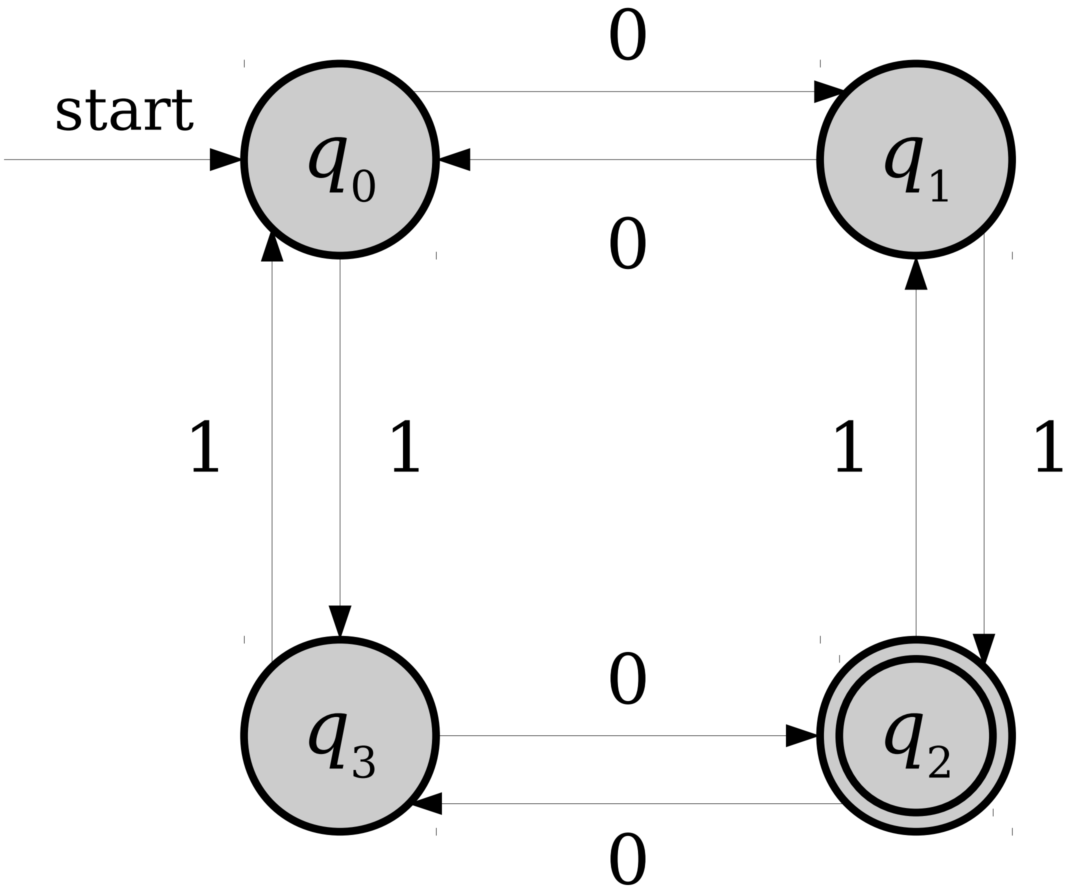
What problems can we solve with a computer?

Finite Automata

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

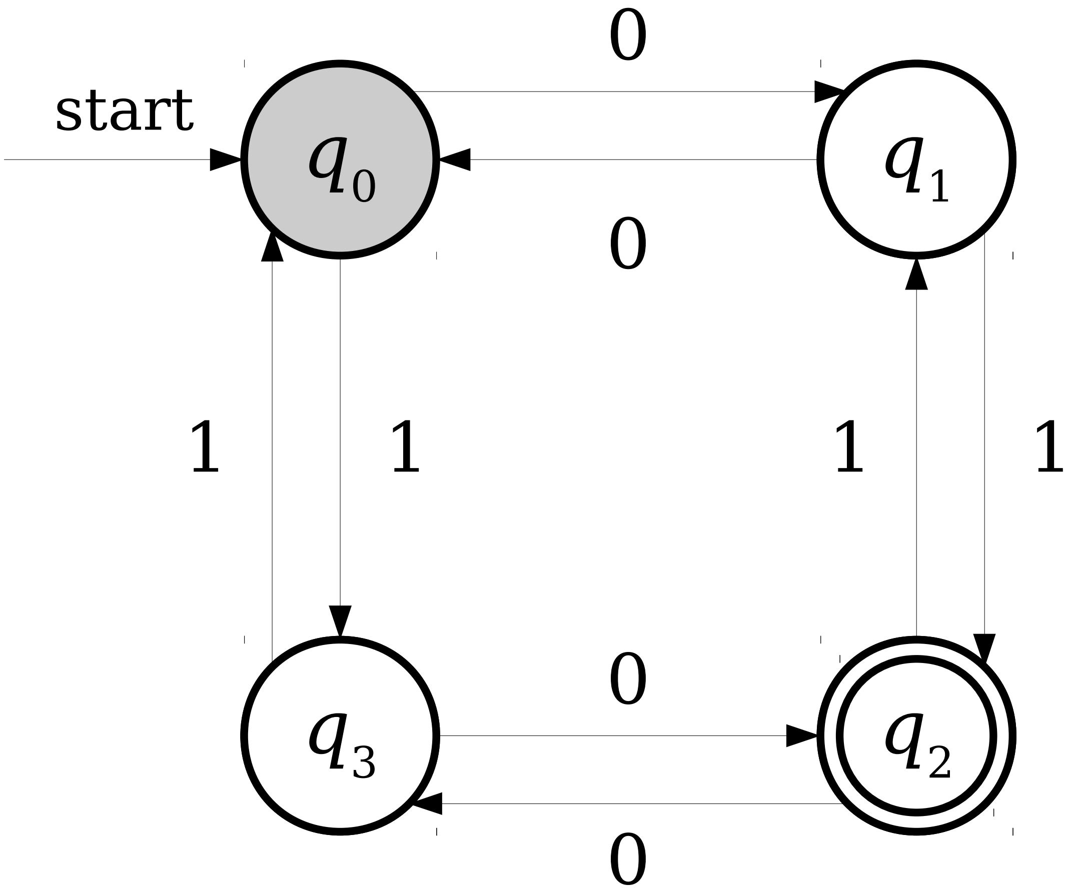
Each finite automaton consists of a set of ***states*** connected by ***transitions***.

A Simple Finite Automaton



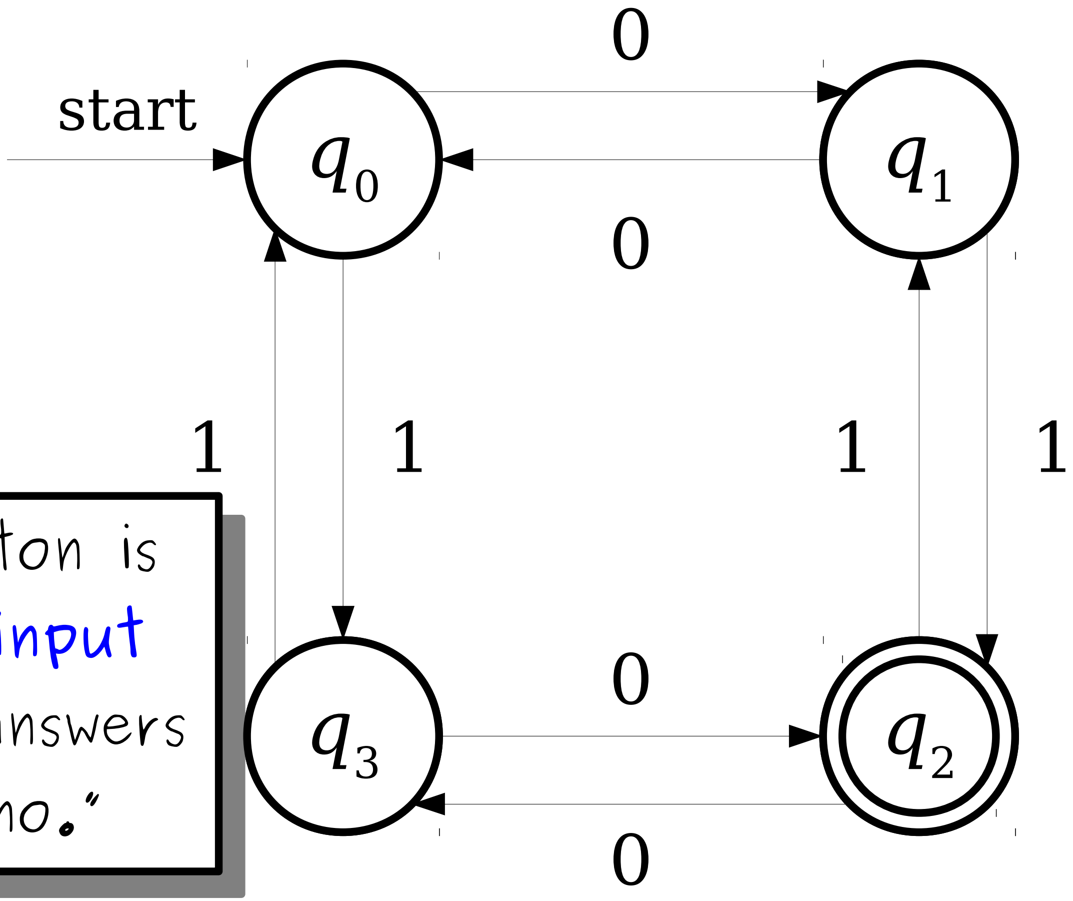
Each circle represents a state of the automaton.

A Simple Finite Automaton



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

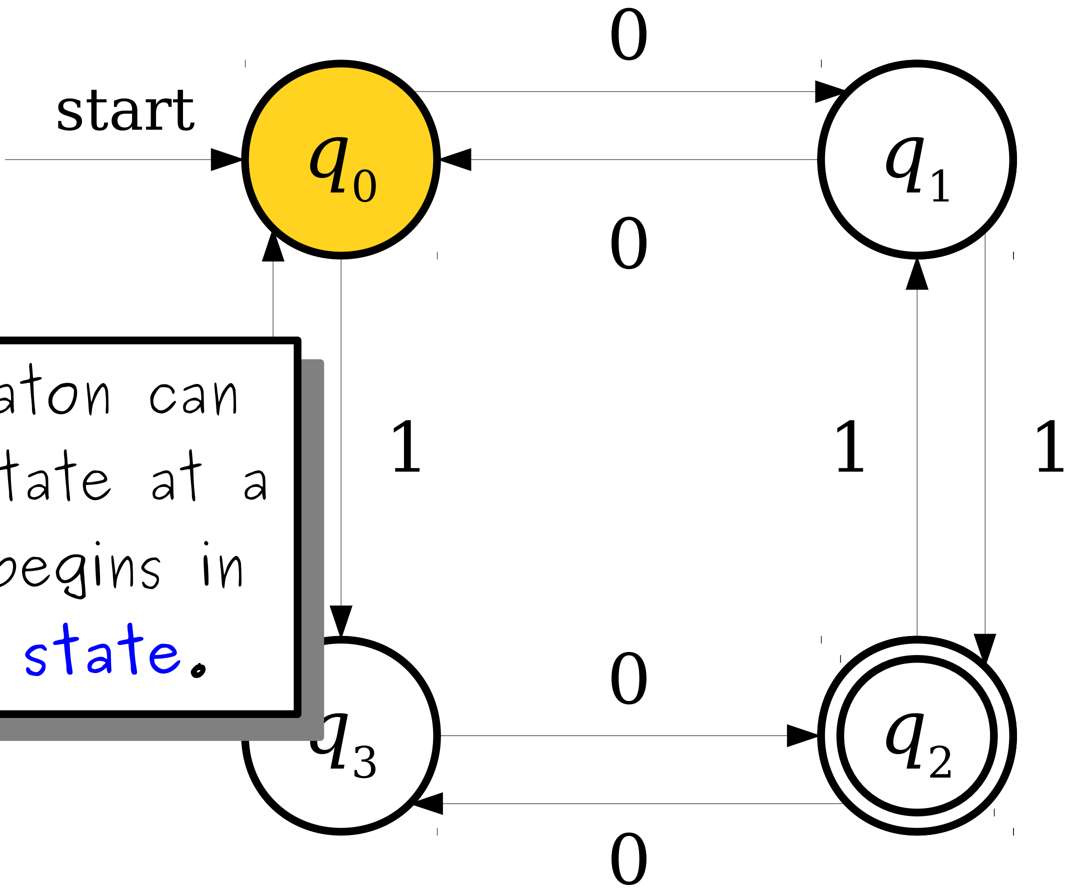
A Simple Finite Automaton



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

0 1 0 1 1 0

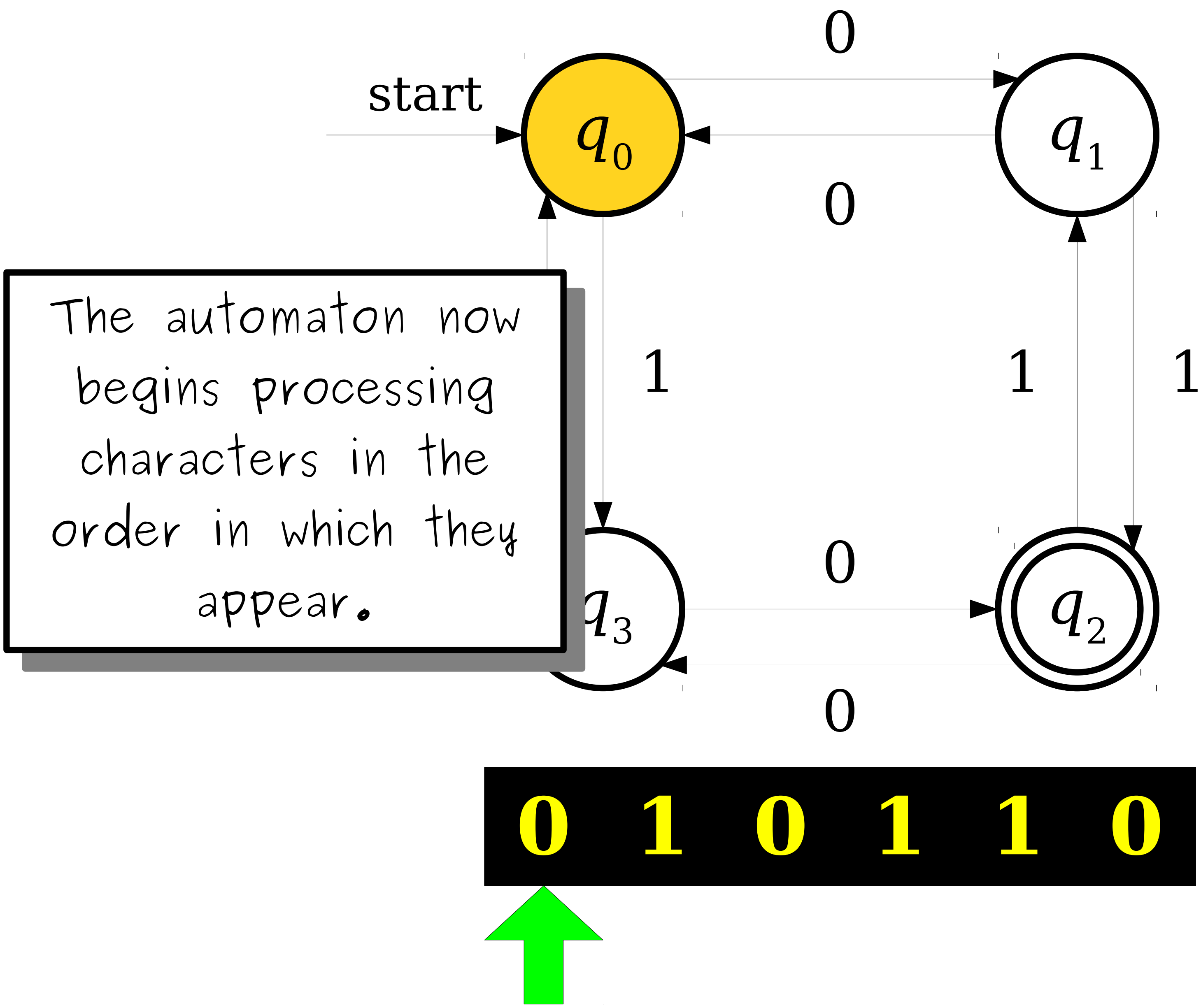
A Simple Finite Automaton



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

0 1 0 1 1 0

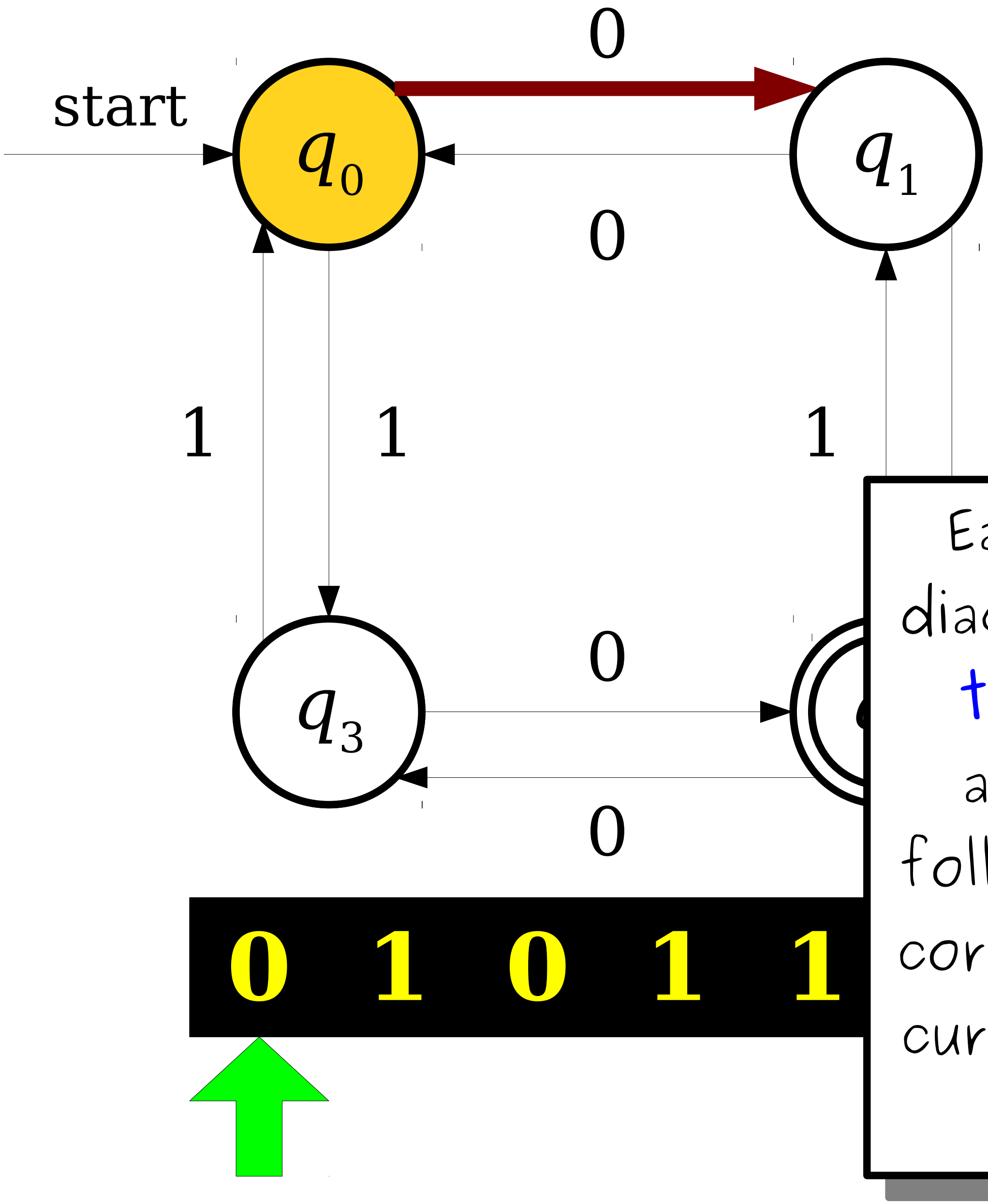
A Simple Finite Automaton



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

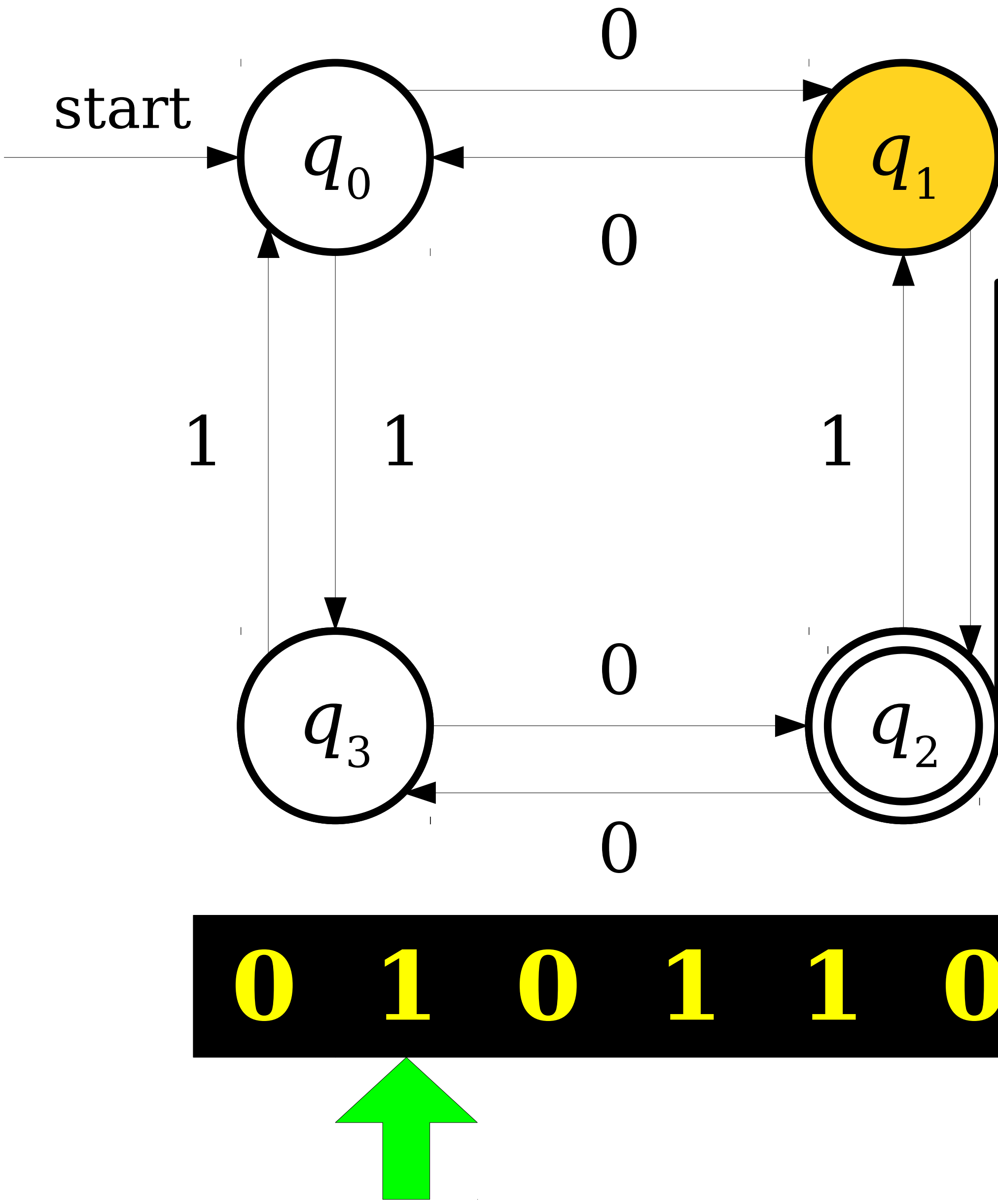
0 1 0 1 1 0

A Simple Finite Automaton



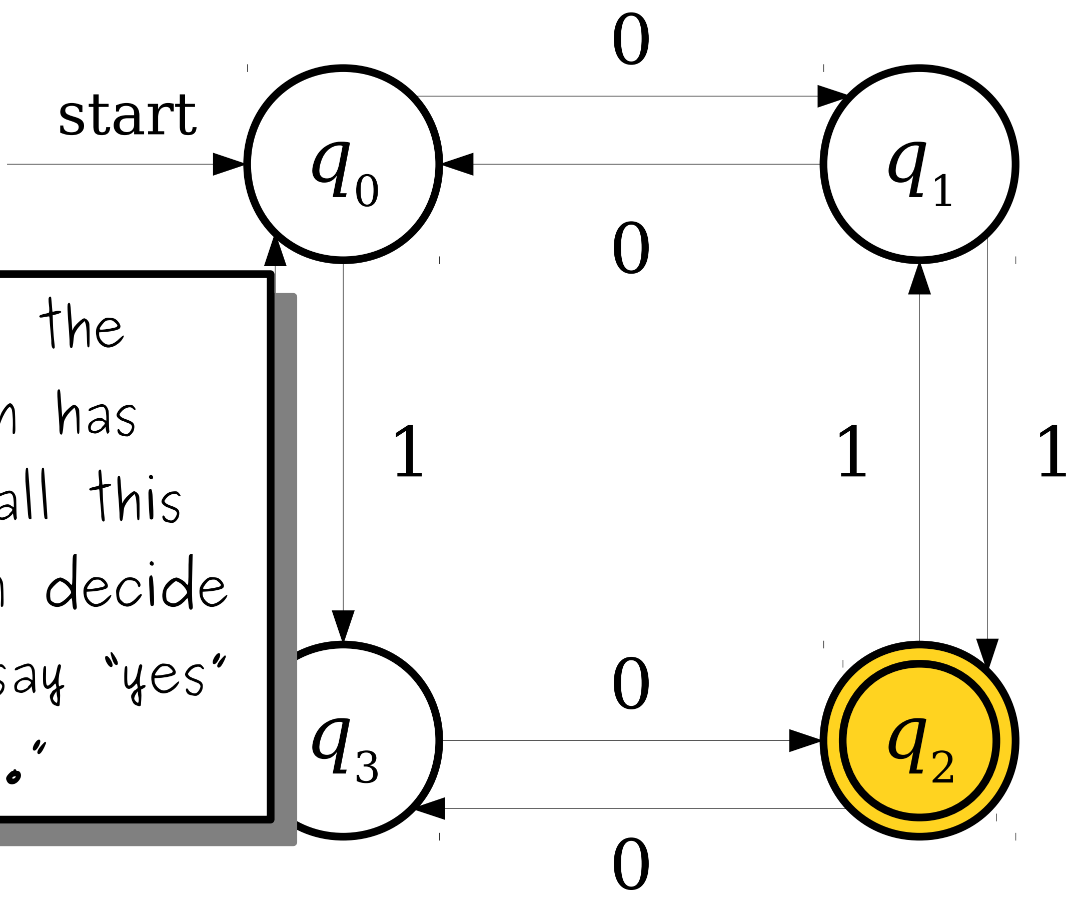
Each arrow in this diagram represents a transition. The automaton always follows the transition corresponding to the current symbol being read.

A Simple Finite Automaton



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

A Simple Finite Automaton

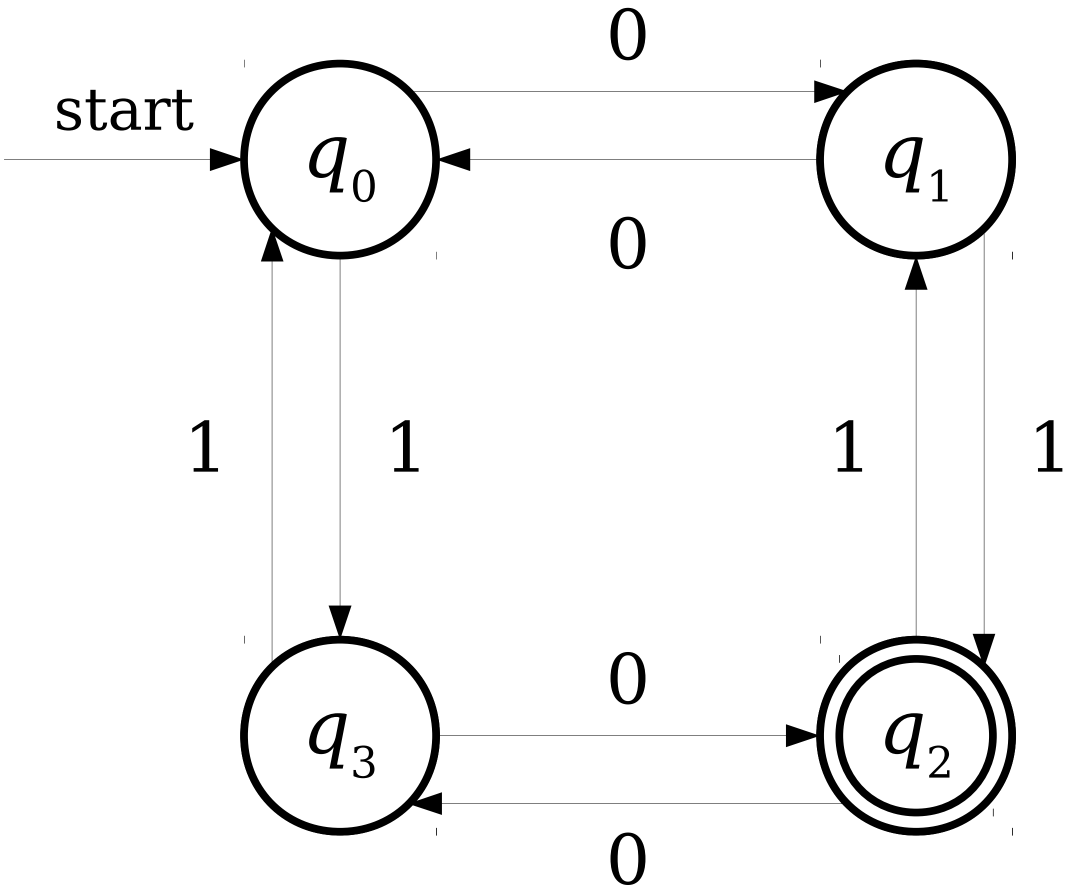


Now that the automaton has looked at all this input, it can decide whether to say "yes" or "no."

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

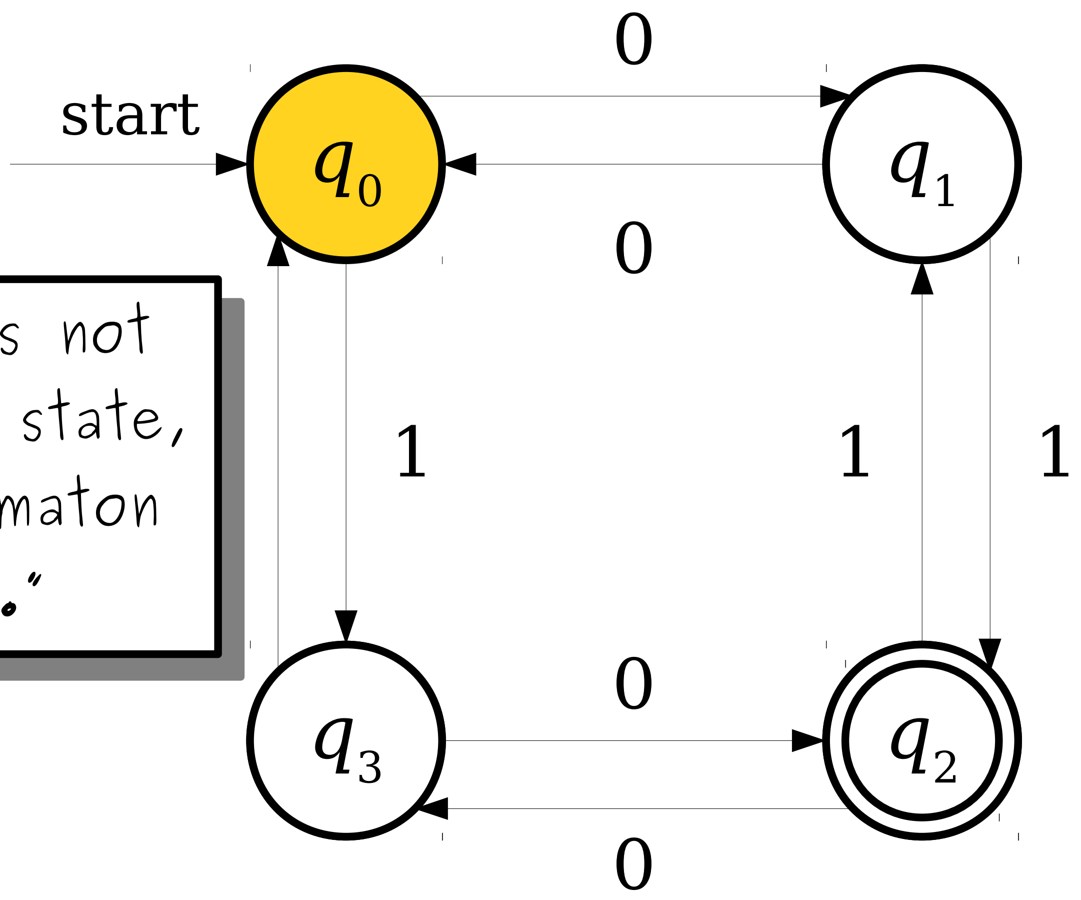
0 1 0 1 1 0

A Simple Finite Automaton



1 0 1 0 0 0

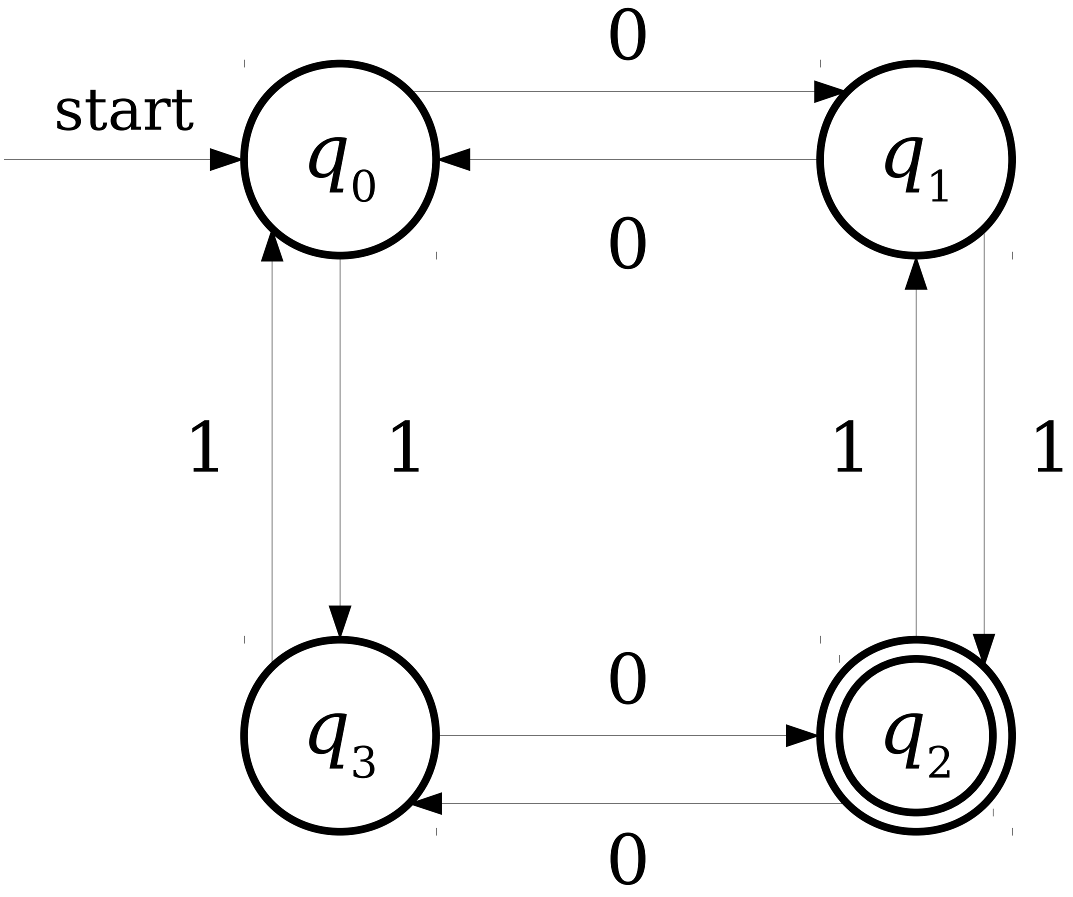
A Simple Finite Automaton



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

1 0 1 0 0 0

A Simple Finite Automaton



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!

Midterm Logistics

- Midterm exam is tonight from 7PM – 10PM.
- Room assignments divvied up by last (family) name:
    - **`Aga – Ven`**: Go to Hewlett 200.
    - **`Ver – Zhe`**: Go to 370-370
- Remember that you get an 8.5” × 11” notes sheet, double-sided, during the exam.
- ***Good luck!***

Your Questions

“Does it reflect poorly on you in the job search if you only take the minimum required classes for the CS major? I love CS, but there's so much else I want to explore, especially if I'm going to do a career in CS.”

Not at all! A CS degree is a CS degree. I doubt that companies are even going to notice this. If anything, taking more classes outside CS makes you even *more* attractive because it means that you bring a more interesting perspective to your work. I've heard anecdotes of interviewers at certain companies asking questions to make sure that you're not just a Code Machine That Has No Life, so you should be covered on that front. ☺

“Are you going to leave Stanford one day and start your own company?”

I seriously doubt it. I can't think of anything I'd like to do that would be best accomplished by making a startup. (Except for my temporary permanent tattoo idea, but that's not worth quitting my day job for.)

“What's one of the most memorable Halloween costumes you've ever seen or worn?”

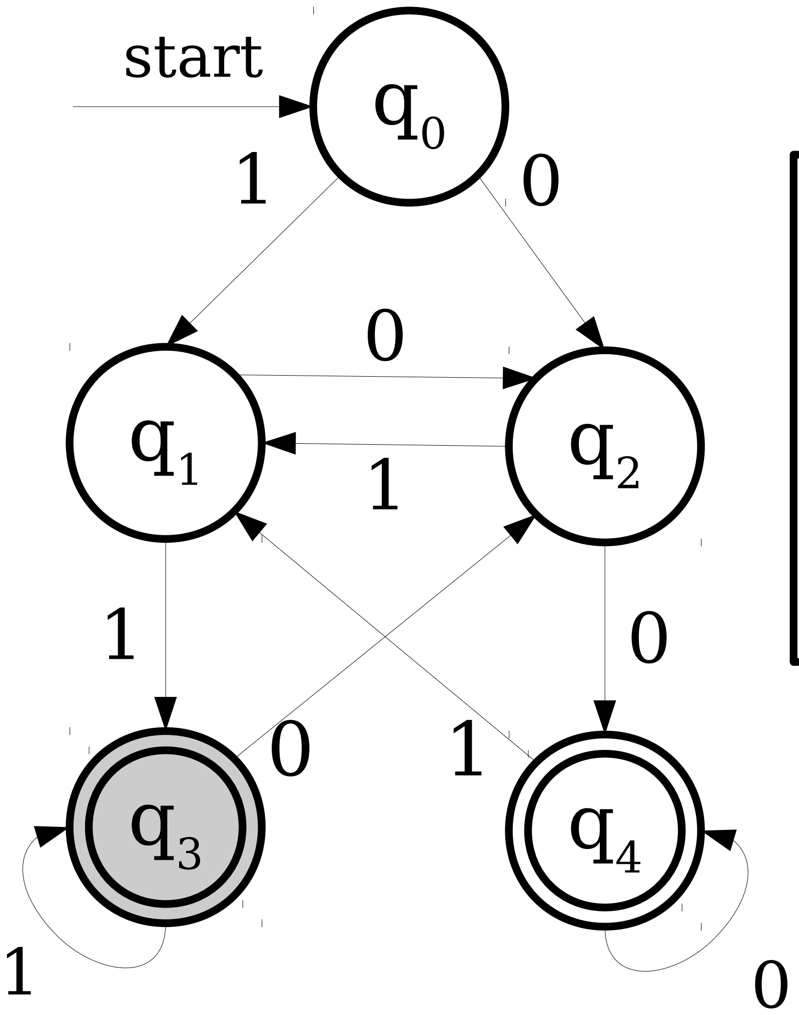
Someone sat in the front row of my class wearing a “Where's Waldo?” costume. I then remarked to the entire class that I was surprised no one was in costume. (Awkward…)

Back to CS103!

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

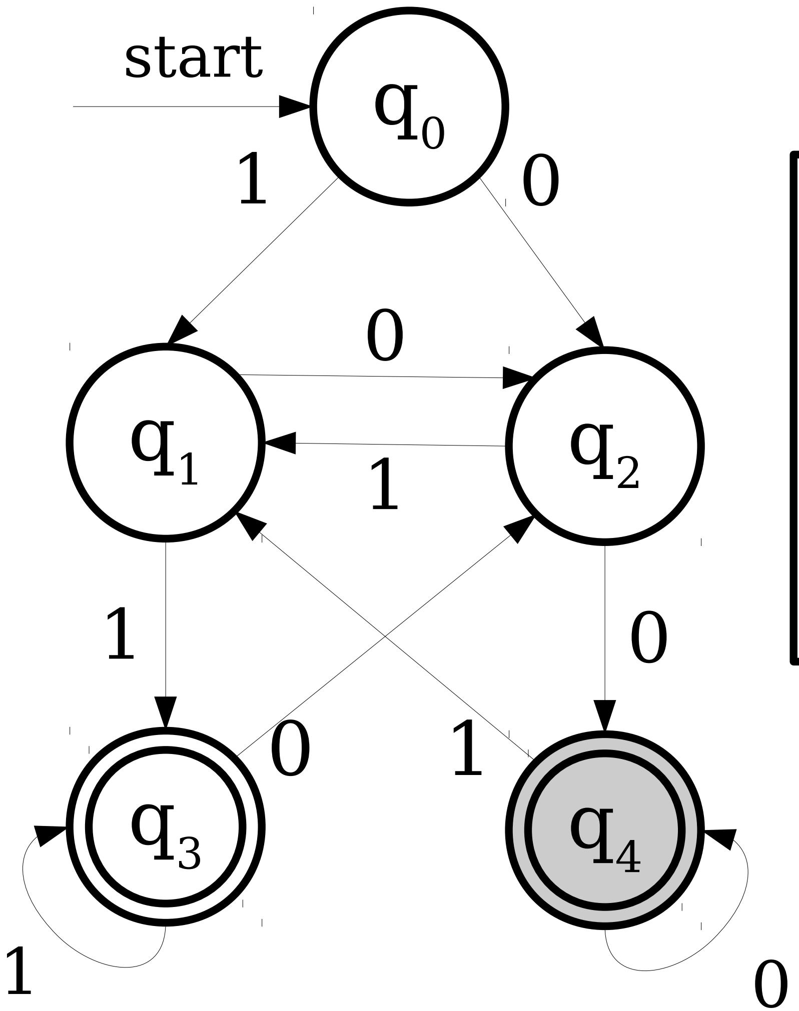
A finite automaton accepts if it ***ends*** in an accepting state.

What Does This Accept?



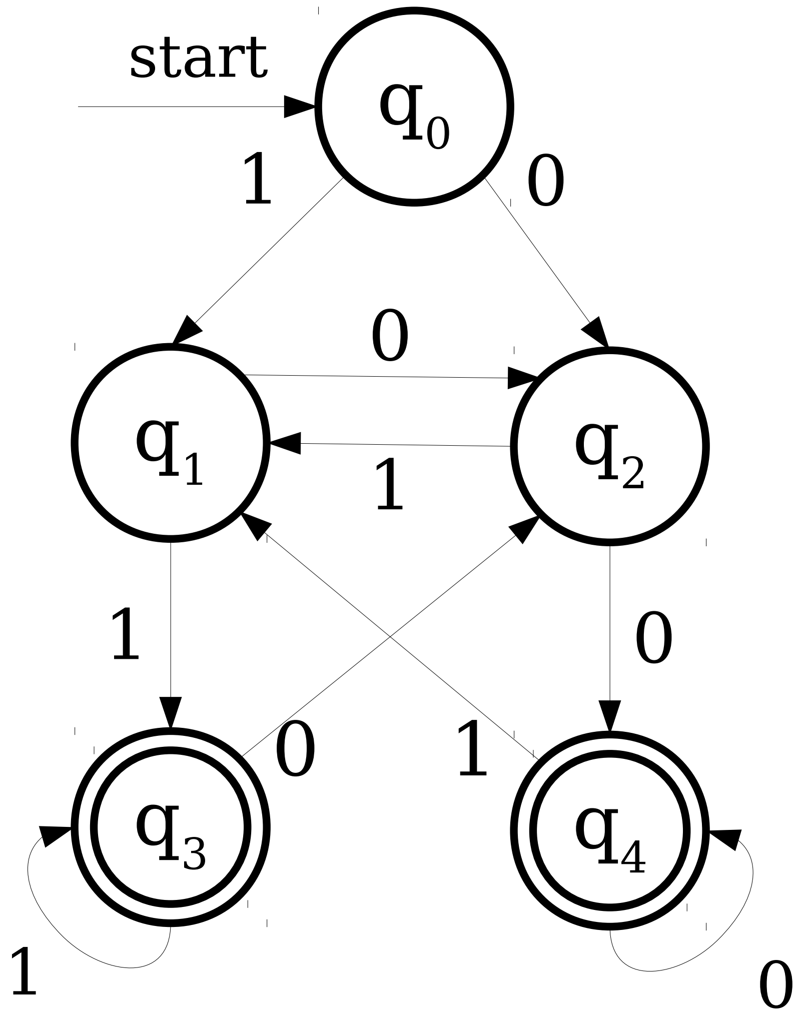
No matter where we start in the automaton, after seeing two 1's, we end up in accepting state q_3.

What Does This Accept?



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?



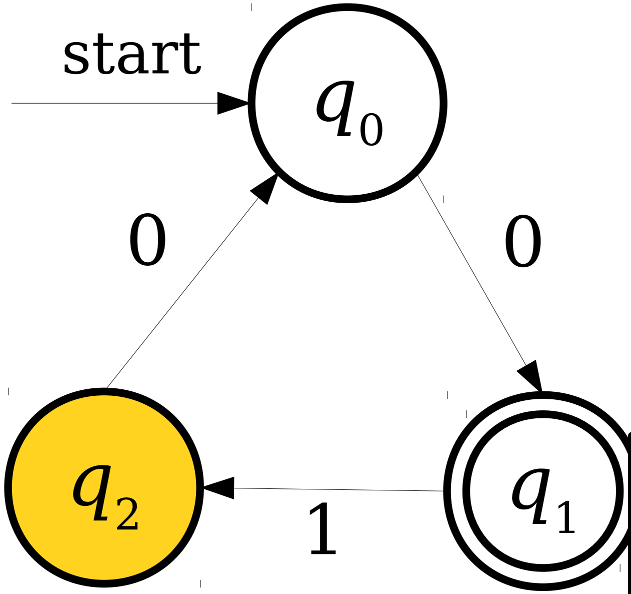
This automaton accepts a string iff the string ends in 00 or 11.

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

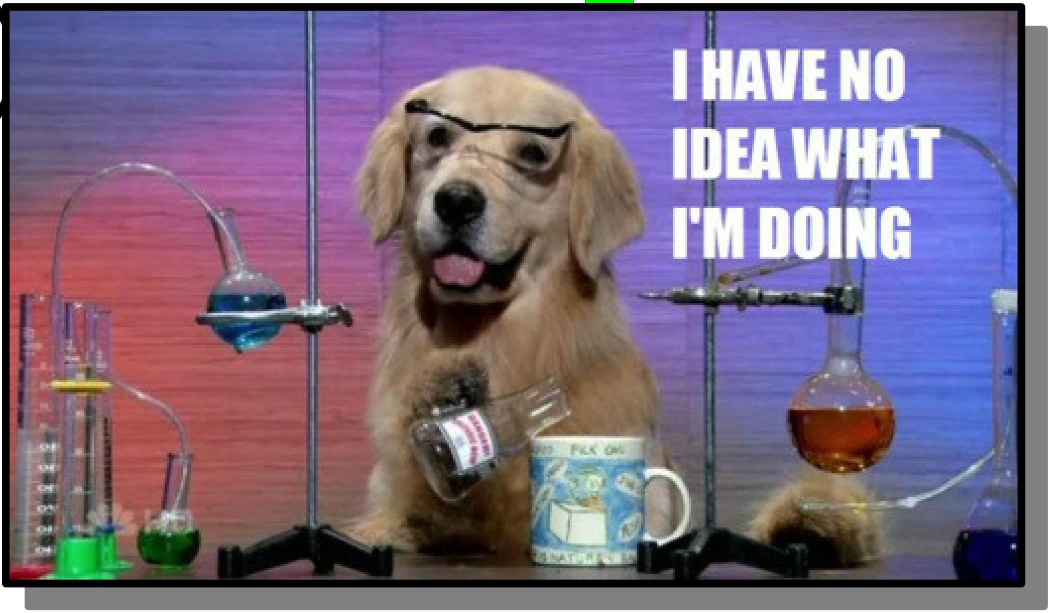
If D is an automaton, we denote the language of D as $\mathscr{L}$**(*D*)**.

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

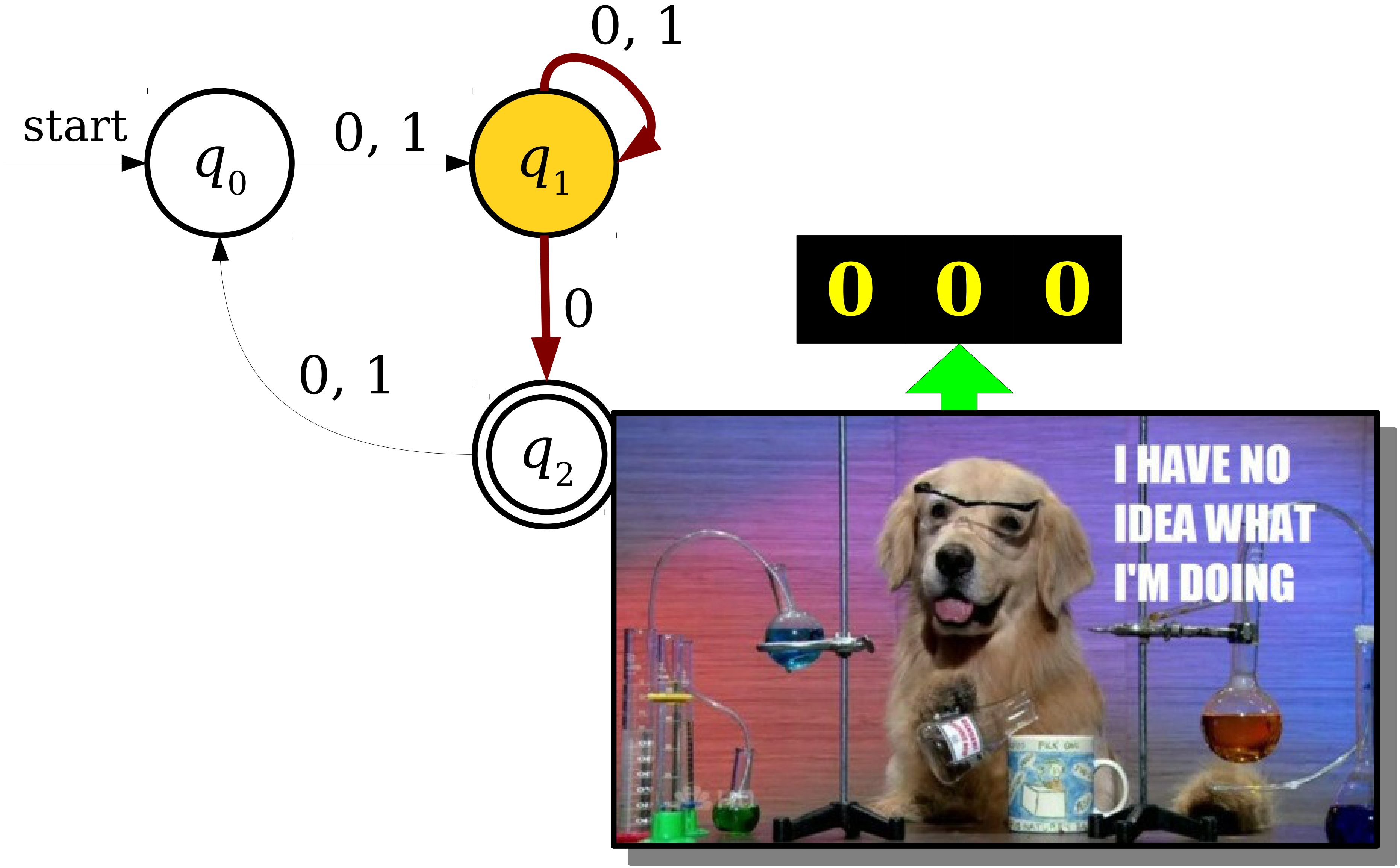
A Small Problem



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?

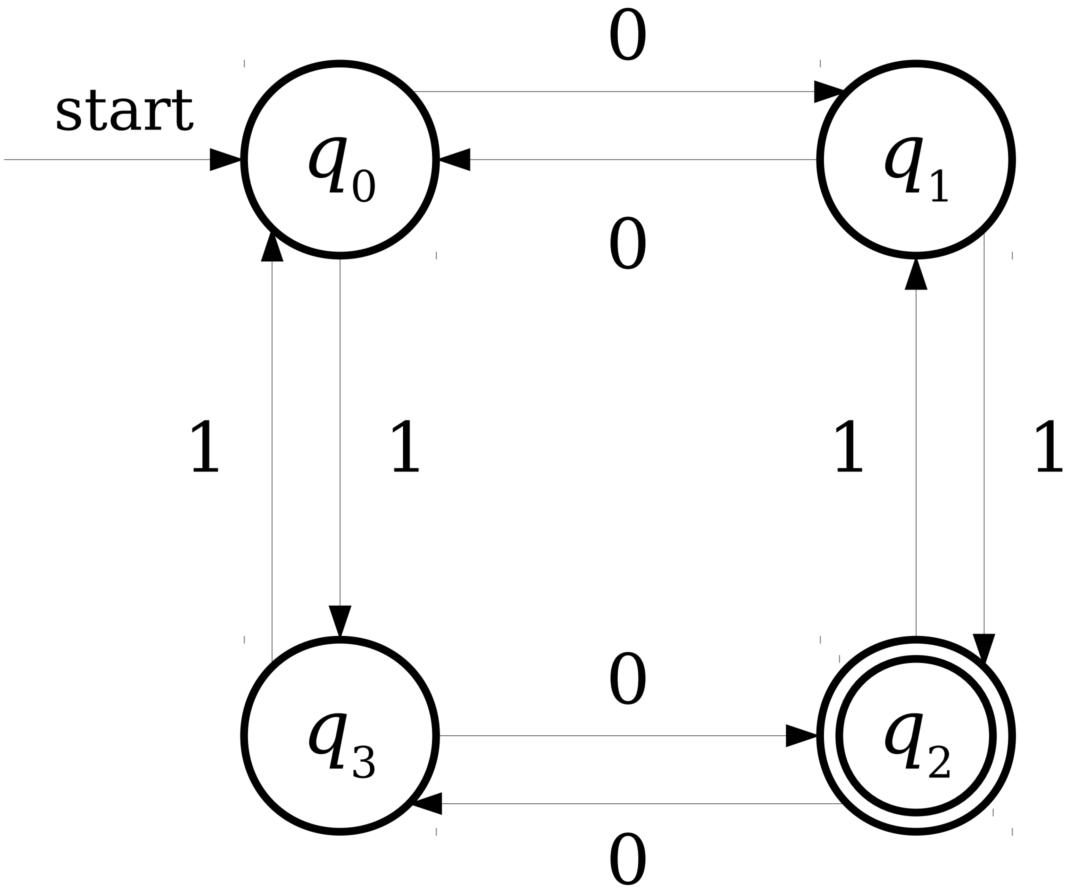
DFAs

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

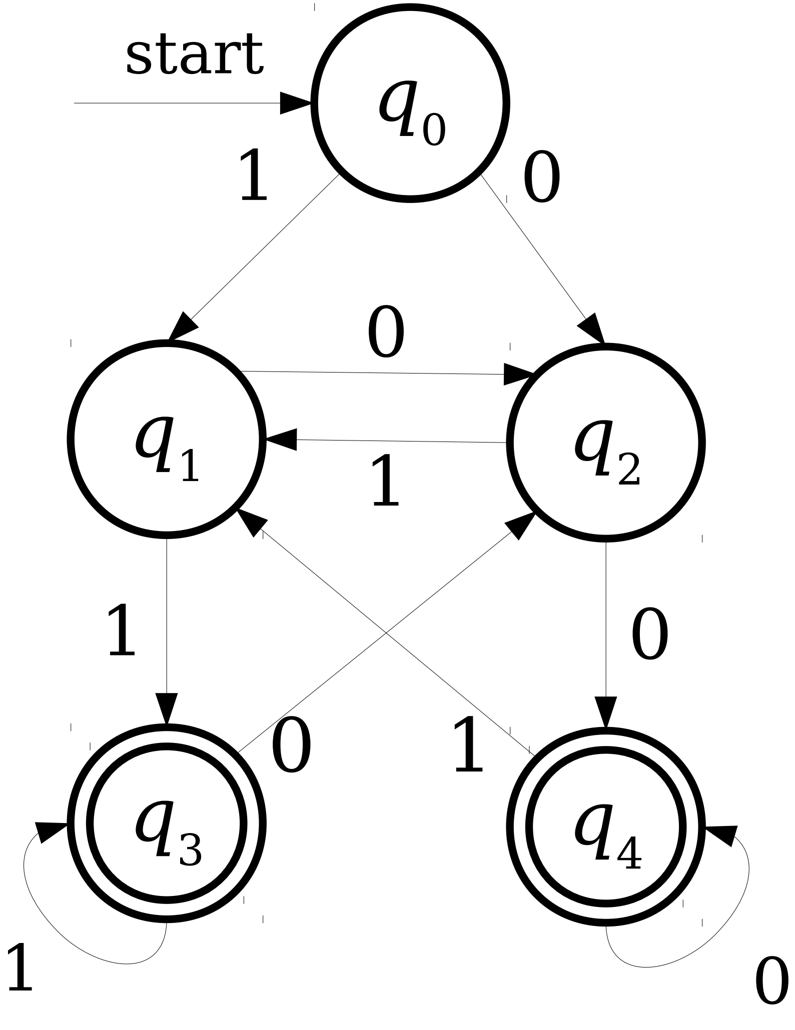
DFAs, Informally

- 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.

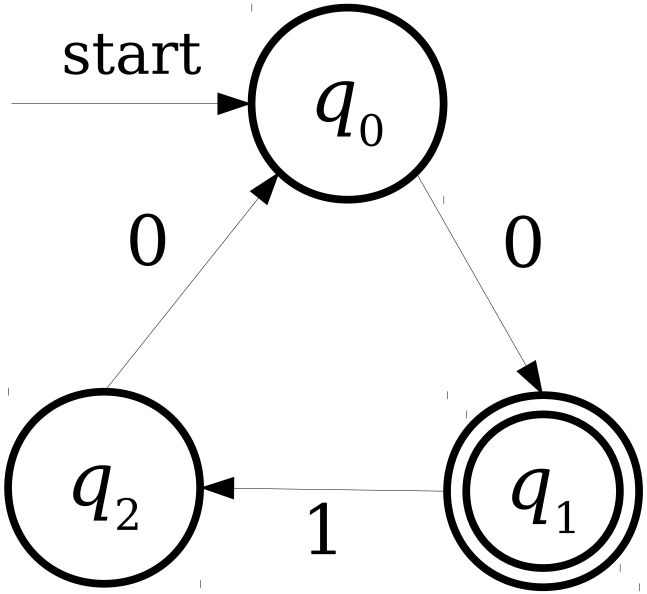
Is this a DFA over {0, 1}?



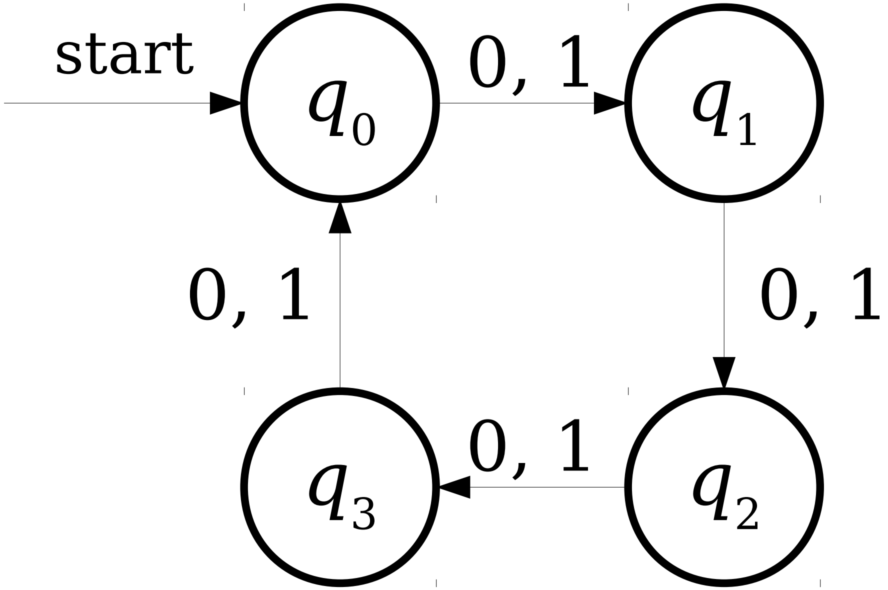
Is this a DFA over {0, 1}?



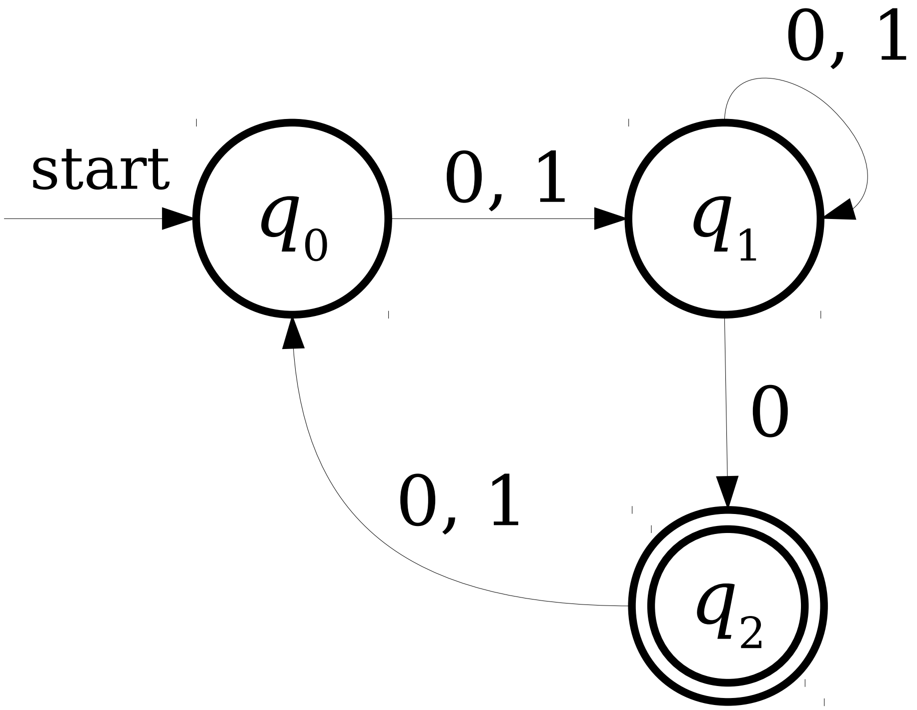
Is this a DFA over {0, 1}?



Is this a DFA over {0, 1}?



Is this a DFA over {0, 1}?



Is this a DFA?

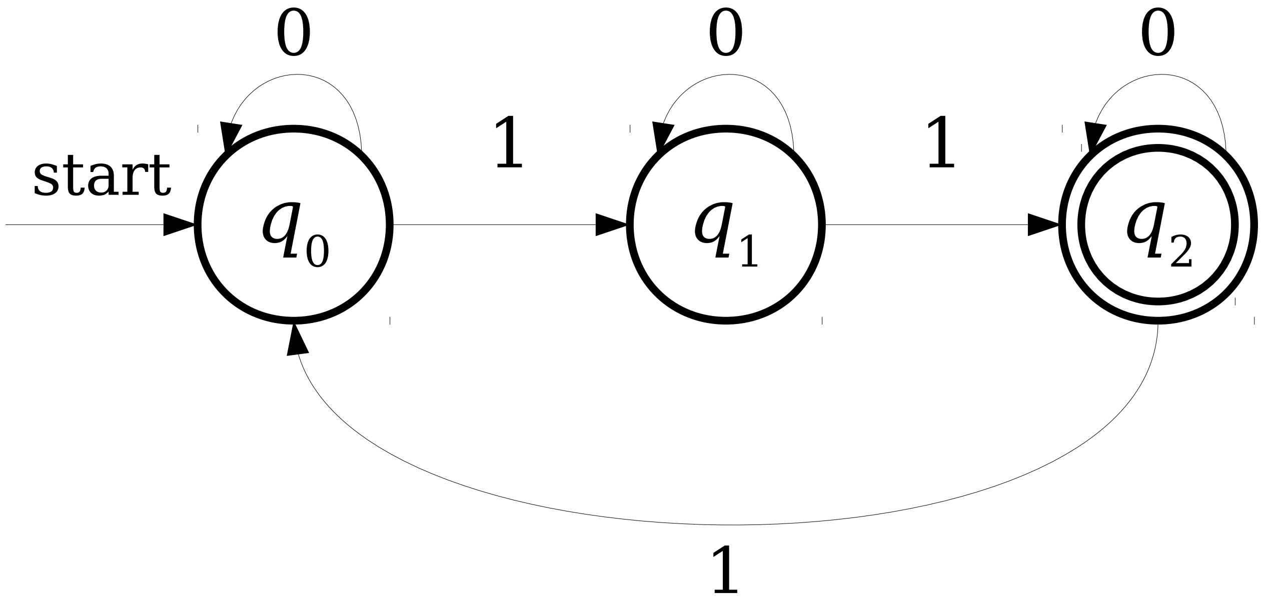
Drinking **F**amily of **A**ardvarks

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.

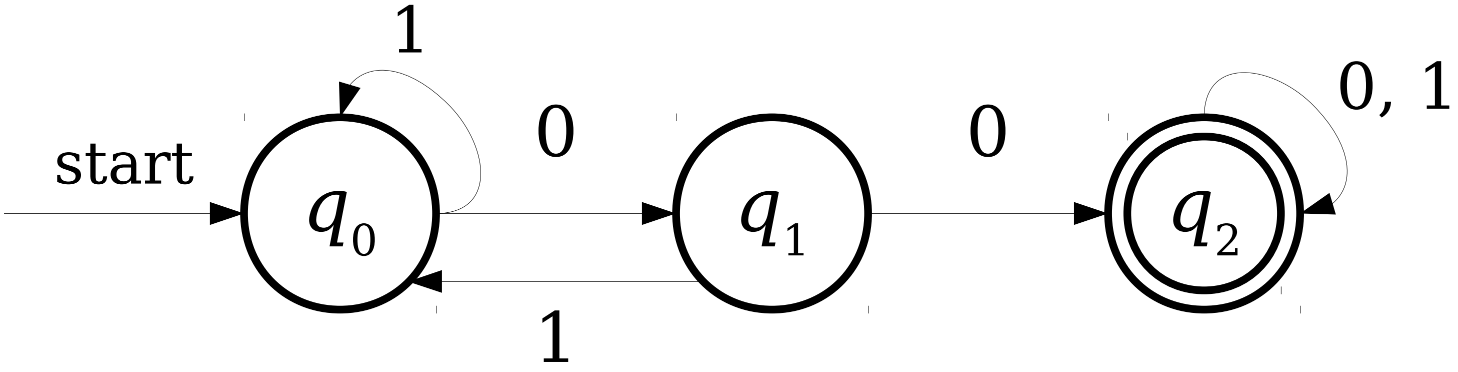
Recognizing Languages with DFAs

$L = \{ w \in \{0, 1\}^* |$ the number of 1's in w is congruent to two modulo three $\}$



Recognizing Languages with DFAs

$L = \{ w \in \{\texttt{0}, \texttt{1}\}^* \mid w \text{ contains } \texttt{00} \text{ as a substring } \}$


start
q_0
q_1
q_2
1
0
1
0
0, 1