# Theory of Computation

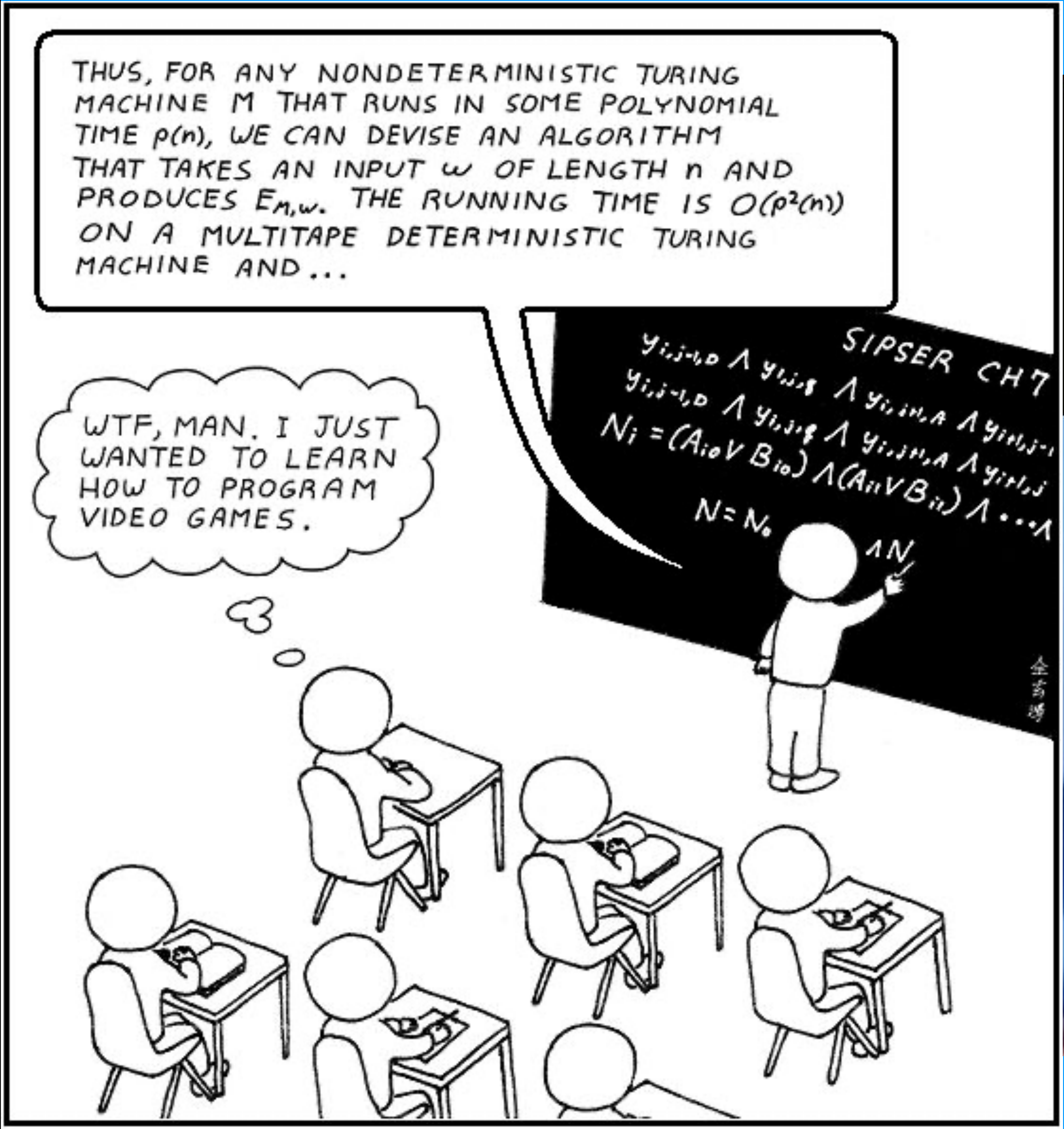
BCS1110

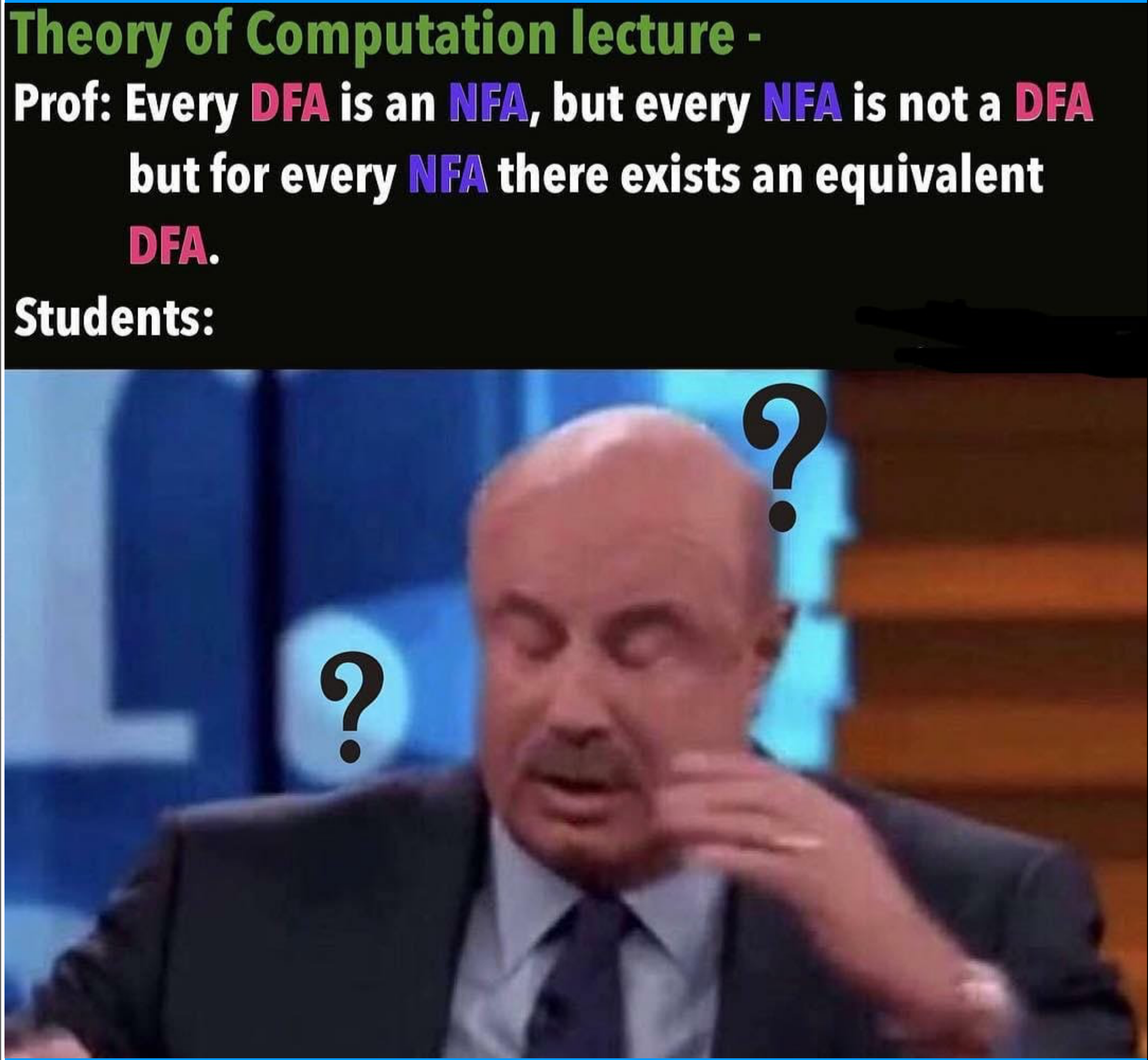
Dr. Ashish Sai

🗓️ TOC - Lecture 1

💻 bcs1110.ashish.nl

📍 EPD150


THUS, FOR ANY NONDETERMINISTIC TURING MACHINE M THAT RUNS IN SOME POLYNOMIAL TIME p(n), WE CAN DEVISE AN ALGORITHM THAT TAKES AN INPUT ω OF LENGTH n AND PRODUCES $E_{M,w}$. THE RUNNING TIME IS $O(p^2(n))$ ON A MULTITAPE DETERMINISTIC TURING MACHINE AND...
SIPSER CH7
WTF, MAN. I JUST WANTED TO LEARN HOW TO PROGRAM VIDEO GAMES.


Theory of Computation lecture -
Prof: Every DFA is an NFA, but every NFA is not a DFA but for every NFA there exists an equivalent DFA.
Students:
?
?

# Quick Recap

| Week | Lecture 1 | Lecture 2 |
|---|---|---|
| **Week 1** | Introduction (Computational Thinking) | Hardware (Transistors, Gates (*AND*, *OR*, *NOT*), Combinational Circuits, ALU, CPU, Computing Hardware) |
| **Week 2** | Algorithms (Flowcharts, Pseudocode) – Dr. Tom Bitterman | Command Line and Git – Dr. Tom Bitterman |
| **This Week** | **Theory of Computation** | **Theory of Computation** |
| **Week 4** | Computer Networks | Computer Networks |
| **Week 5** | Information Security | Information Security |

# Plan for Today

- Formal Language Theory
- Finite Automata
- FSA Examples
- Deterministic Finiate Automaton

# Why Do We Need to Know This? 🤔

- Computer science is more than just:
  1. Writing code 💻
  2. Compiling code 🔄
  3. Fixing bugs in code 🐞
  4. Compiling again 🔄
  5. And finally going for a walk 🚶 because you've ended up with even more bugs than you began with 🤦

> "Computer science at its core is all about problem solving!" 💡

# What problems can we solve with a computer?

# Theory of Computation (TOC)

- TOC answers a fundamental question:

> "What problems can we solve with a computer?"

- Importance of TOC:
  - Knowing what a computer can and cannot do helps us solve problems more efficiently.
  - Some problems cannot be solved by a computer, regardless of the algorithm.

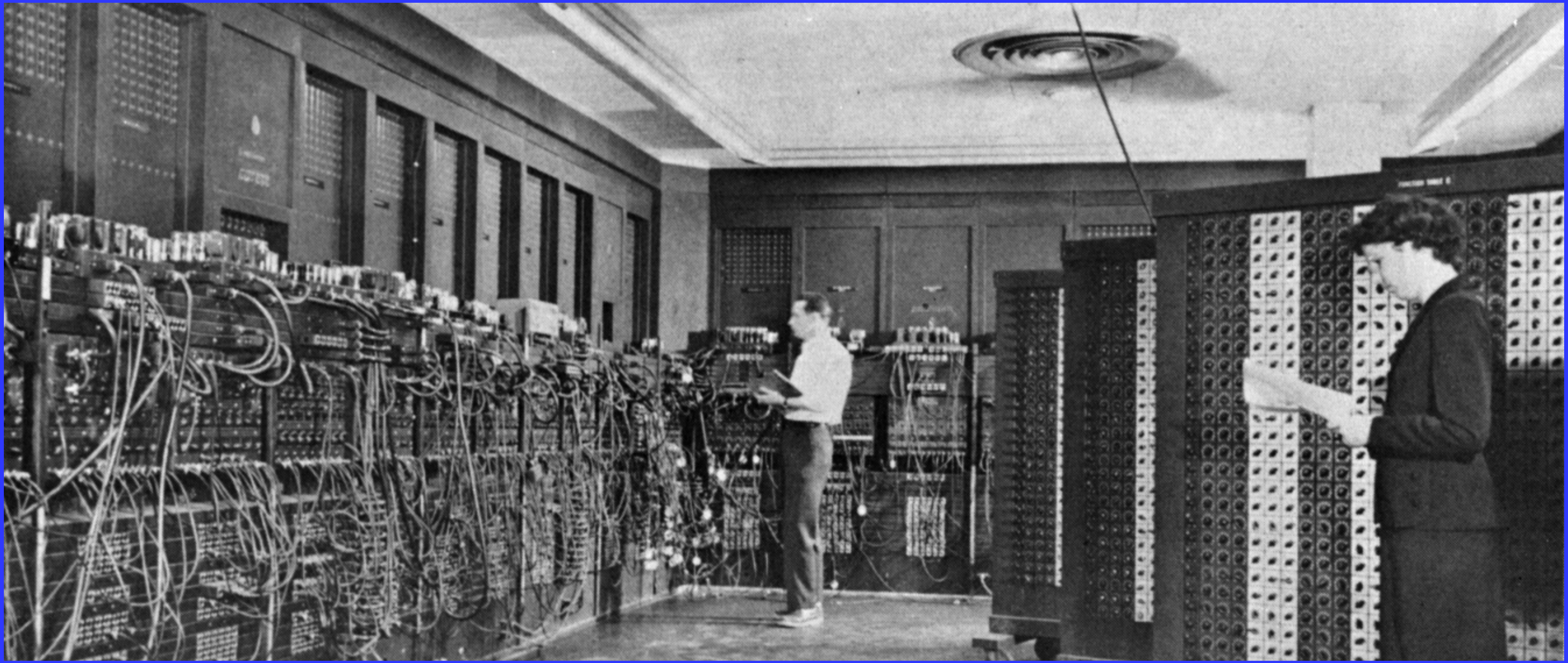
# Halting Problem

A decision problem: will the given program terminate or run forever?

```java
import java.util.Scanner;
public class HaltingProblem {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        while (!scanner.nextLine().isEmpty()) {
            // Loops if input isn't empty
        }
    }
}
```

> Can you write an automated program that could answer this question without running the code?

# Computers are Messy



## Computers are Messy

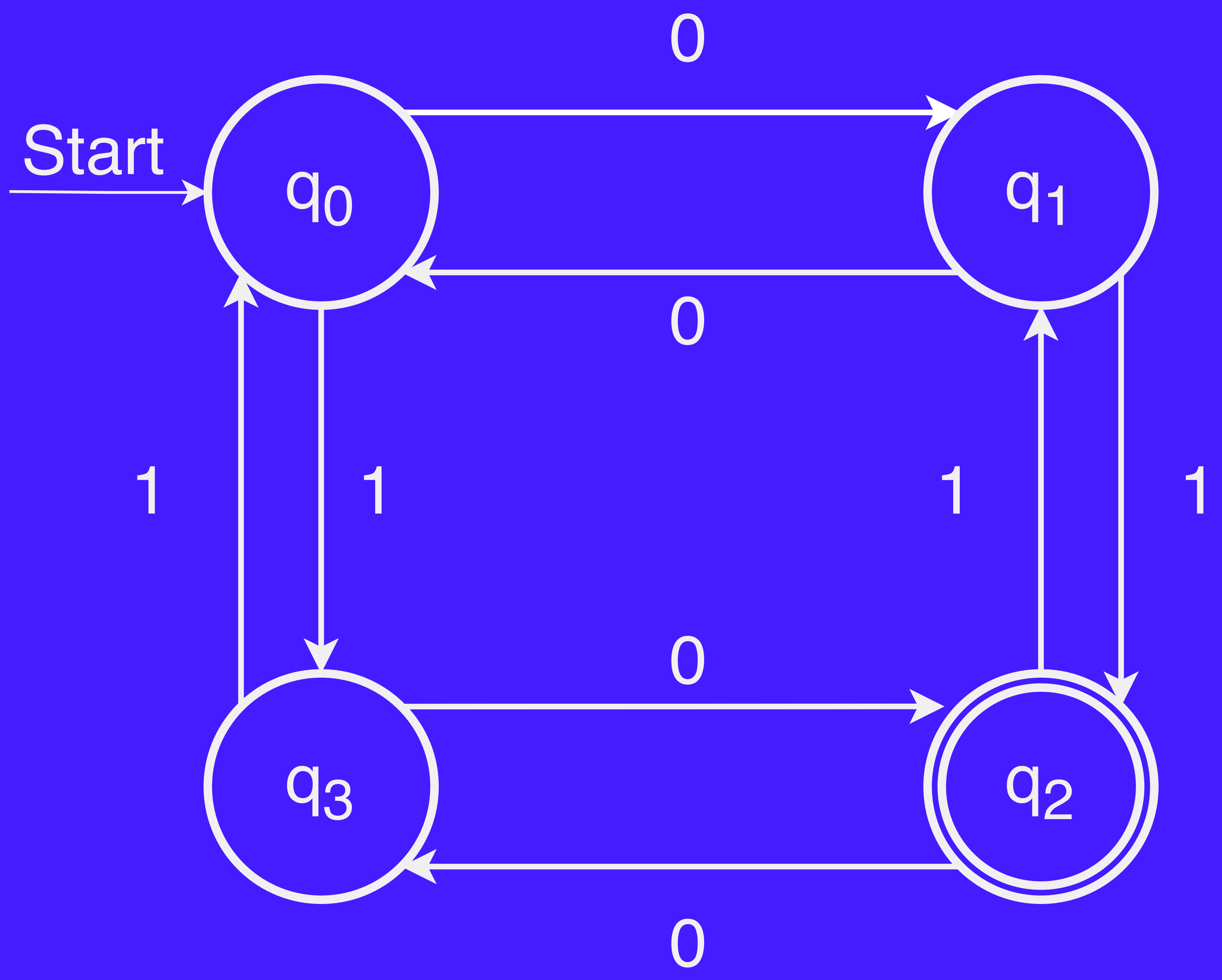
That messiness makes it hard to **rigorously** say what we **intuitively** know to be true: that, on some fundamental level, different brands of computers or programming languages are more or less equivalent in what they are capable of doing.

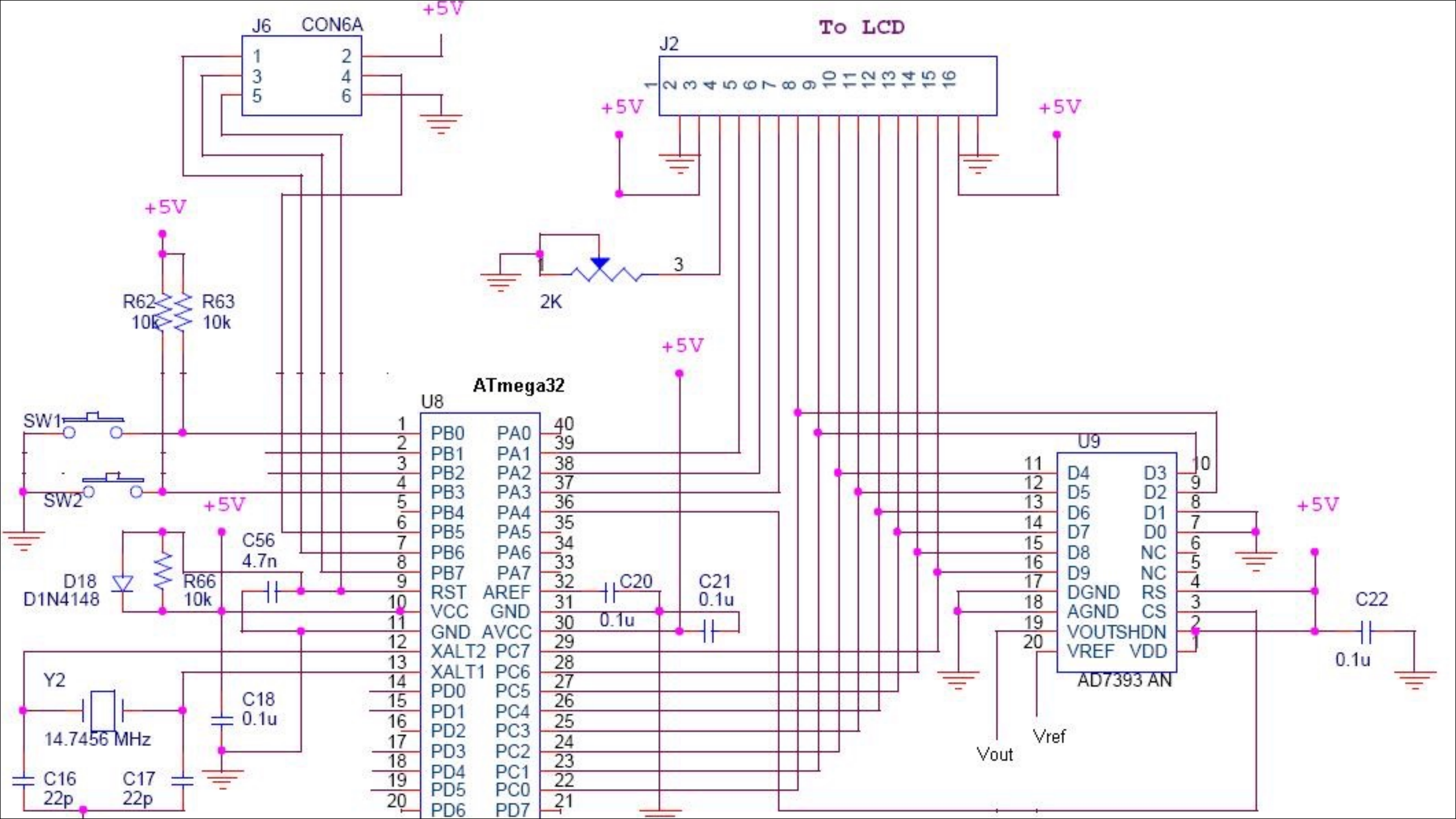
**  vs ⊞ & C vs C++ vs Java vs Python**

# We need a simpler way of discussing computing machines

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

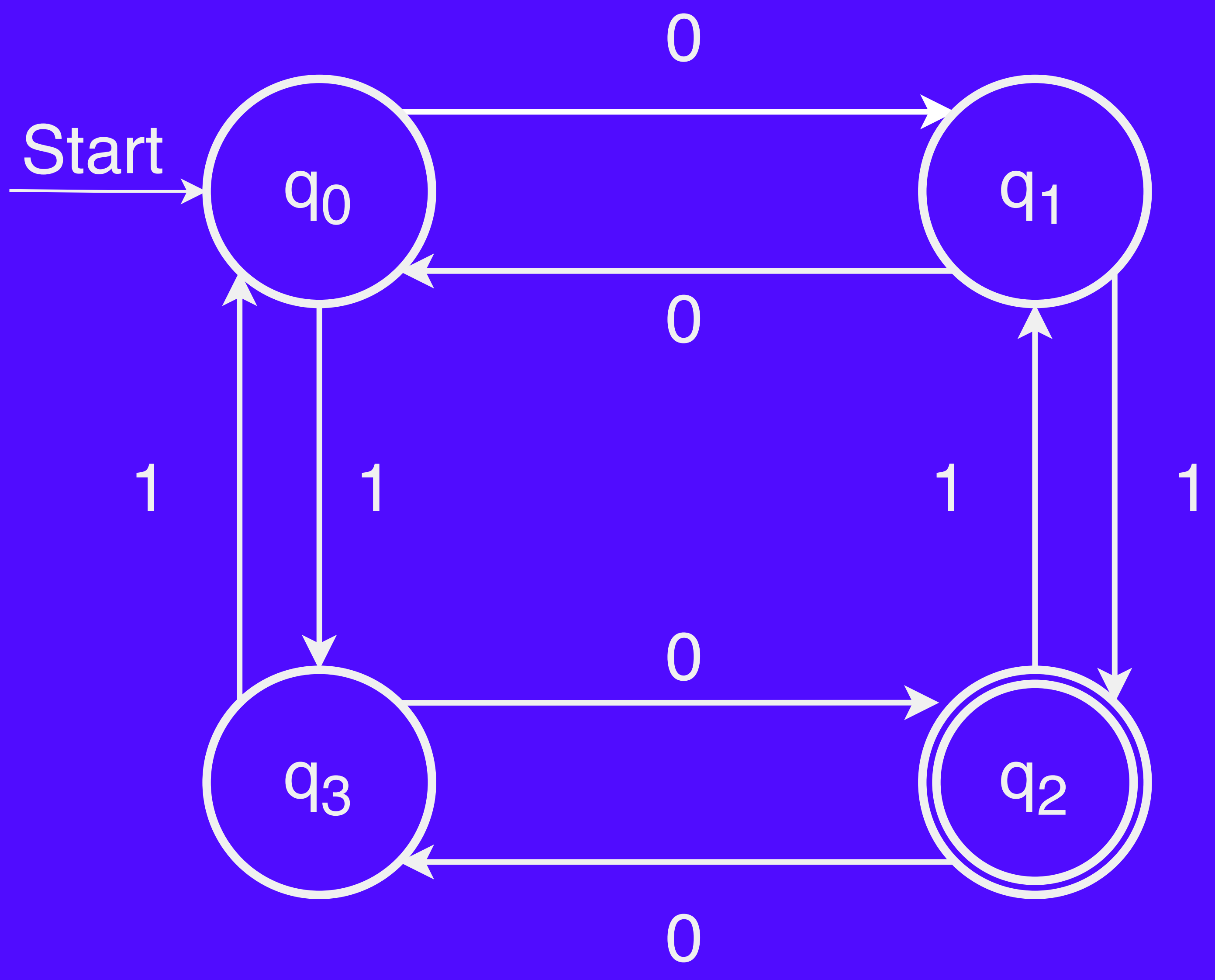
# Automata are Clean





To LCD
J2
J6 CON6A
+5V
R62 10k
R63 10k
2K
SW1
SW2
ATmega32
U8
PB0 PB1 PB2 PB3 PB4 PB5 PB6 PB7 RST VCC GND XALT2 XALT1 PD0 PD1 PD2 PD3 PD4 PD5 PD6
PA0 PA1 PA2 PA3 PA4 PA5 PA6 PA7 AREF GND AVCC PC7 PC6 PC5 PC4 PC3 PC2 PC1 PC0 PD7
C56 4.7n
D18 D1N4148
R66 10k
C20 0.1u
C21 0.1u
Y2
14.7456 MHz
C18 0.1u
C16 22p
C17 22p
U9
D4 D5 D6 D7 D8 D9 DGND AGND VOUT VREF
D3 D2 D1 D0 NC NC RS CS SHDN VDD
AD7393 AN
C22 0.1u
Vout
Vref

# Automata are Clean



# 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 (*today's lecture*) are an abstraction of computers with finite resource constraints
  - Provide upper bounds for the computing machines that we can actually build
    - *Deterministic Finite State Automata (DFA)*
    - *Non-deterministic Finite State Automata (NFA)*

# What problems can we solve with a computer?

# 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

Part 1/4

# Strings (informal)

– Sequence of any symbols (“characters”)

Example:

“hello” “1234” “🧑‍🎓🧑‍🎓🧑‍🎓🧑‍🎓” “ ”

# Strings (more formally)

- An **alphabet** is a finite set of symbols called ***characters***
  - Typically, we use the symbol **Σ** (sigma) 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, aabaaabbabaaabaaaabb**
- The empty string has no characters and is denoted by **ε**

# Languages (informal)

- Sets of strings
  - Examples
  - {hello, 1234, 🧑‍🎓🧑‍🎓🧑‍🎓🧑‍🎓, ε}
  - {10110, 0110, 10}

# Languages (more formally)

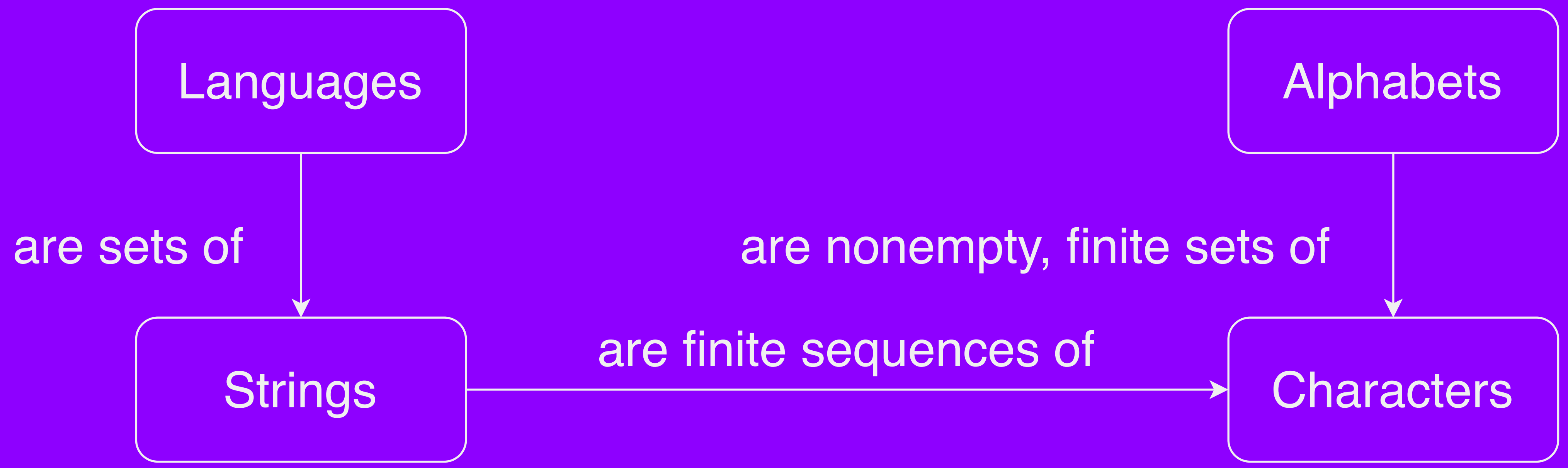
- A formal language is a set of strings.
- We say that L is a language over Σ if it is a set of strings over Σ
- Example: The language of palindromes over Σ = {a,b,c} is the set
  - {ε, a, b, c, aa, bb, cc, aaa, aba, aca, bab, ... }
- The set of all strings composed from letters in Σ is denoted Σ∗
-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

# Quick Quiz

- Which statements are true?
  - **Alphabets** are sequences of characters
  - **Languages** are sets of strings
  - **Strings** are sets of characters
  - **Characters** are individual symbols
  - **Languages** are sequences of characters

# 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*? (Essentially pattern recognition)
- 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 models of computation and see how this seemingly abstract question has very real and powerful consequences.

# To Summarise

- An **automaton** is an idealized mathematical computing machine (I use the terms machine and automata interchangeably)
- A **language** is a set of strings, a **string** is a (finite) sequence of characters, and a **character** is an element of an **alphabet**

# What problems can we solve with a computer?

# Finite Automata

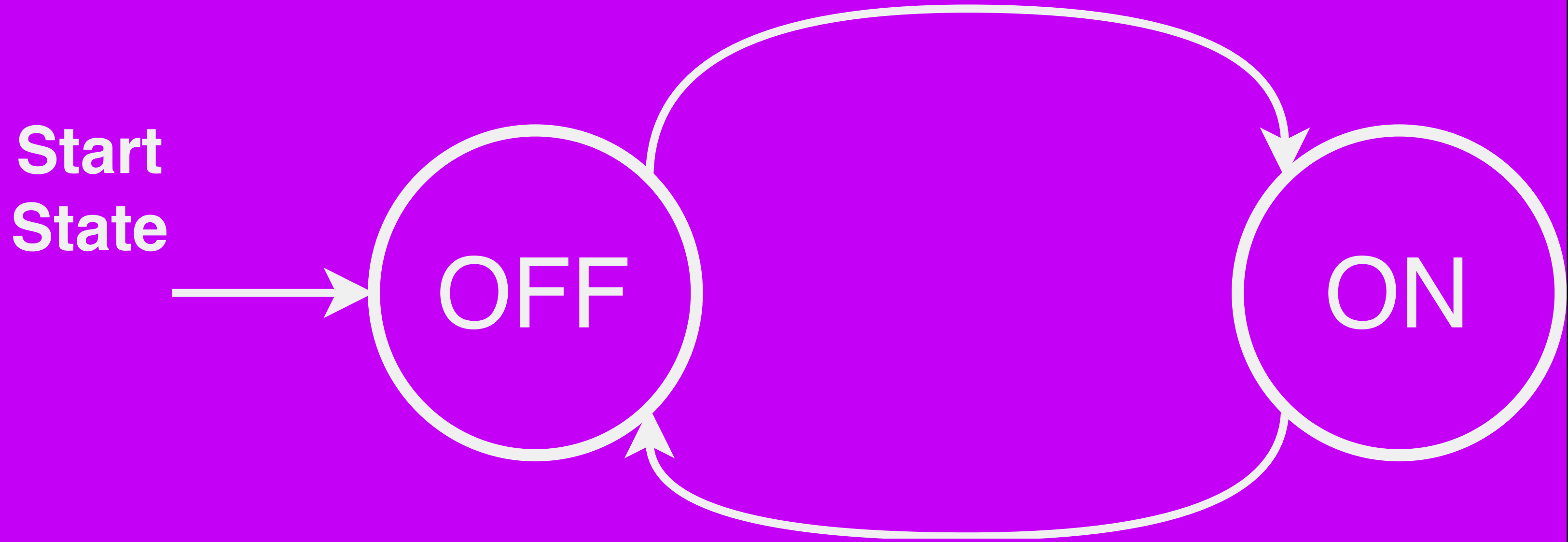
Part 2/4

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


Smart Oven
SAMSUNG


Smart Oven
SAMSUNG

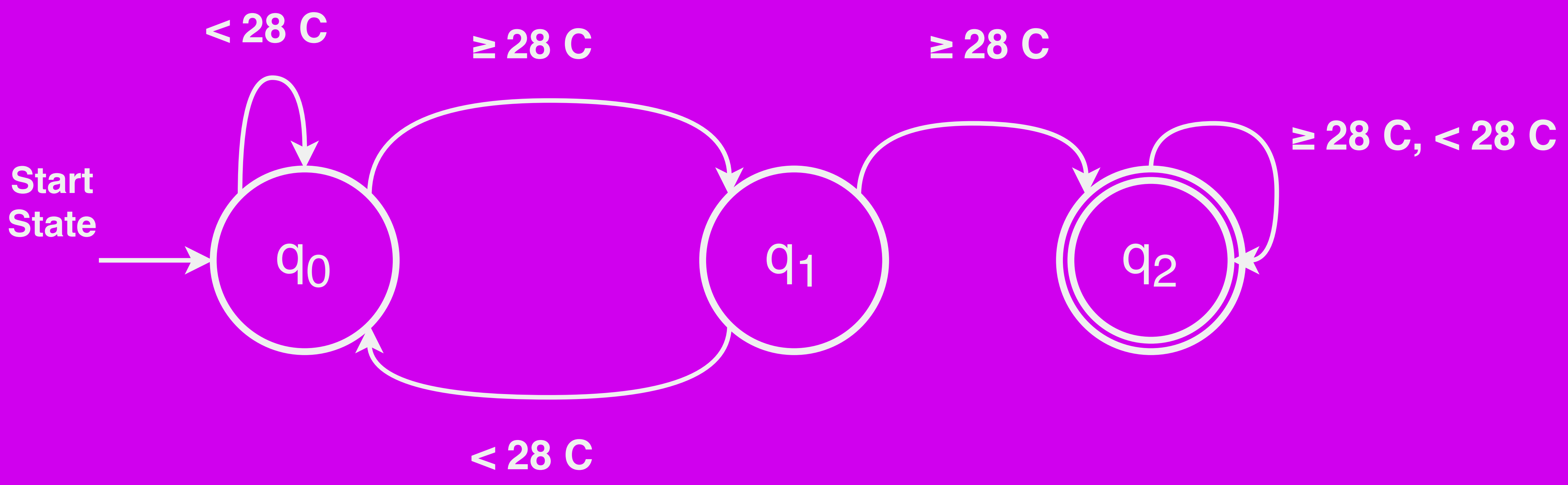

Start
State
OFF
ON

# Automata to determine if a heatwave occurred

- Input: String of weather data
- 🇬🇧 Heatwave: temperature ≥ 28 C for 2 consecutive days

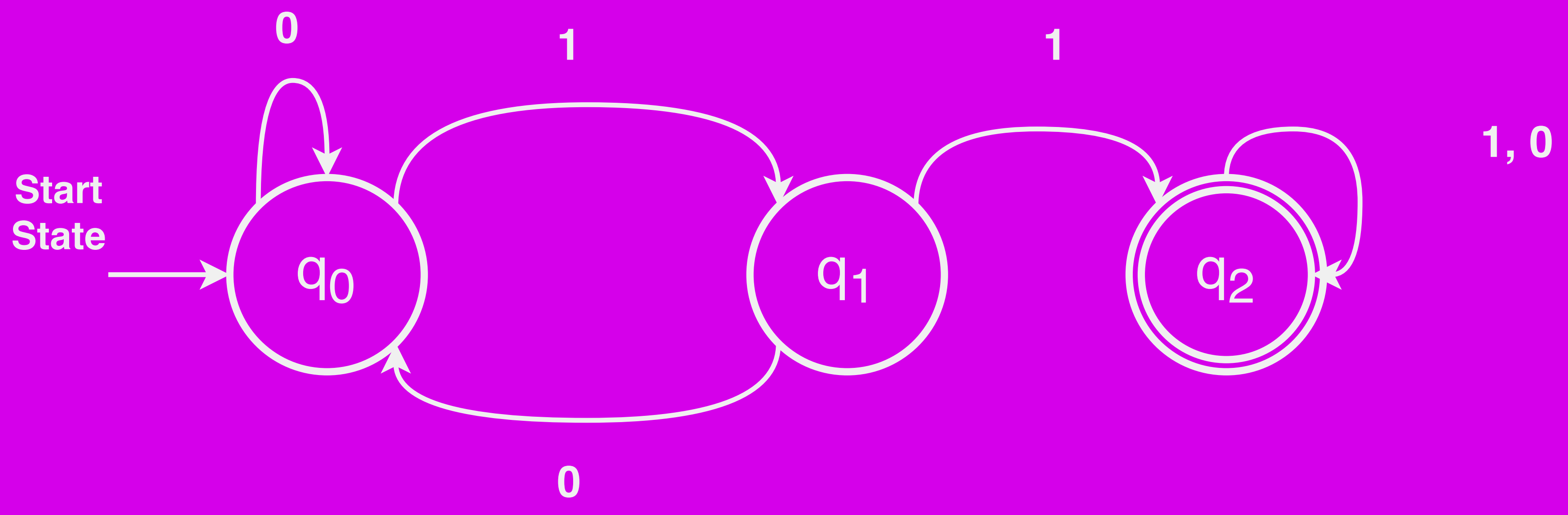
# Automata to determine if a heatwave occurred

- Input: String of weather data
- 🇬🇧 Heatwave: temperature ≥ 28 C for 2 consecutive days

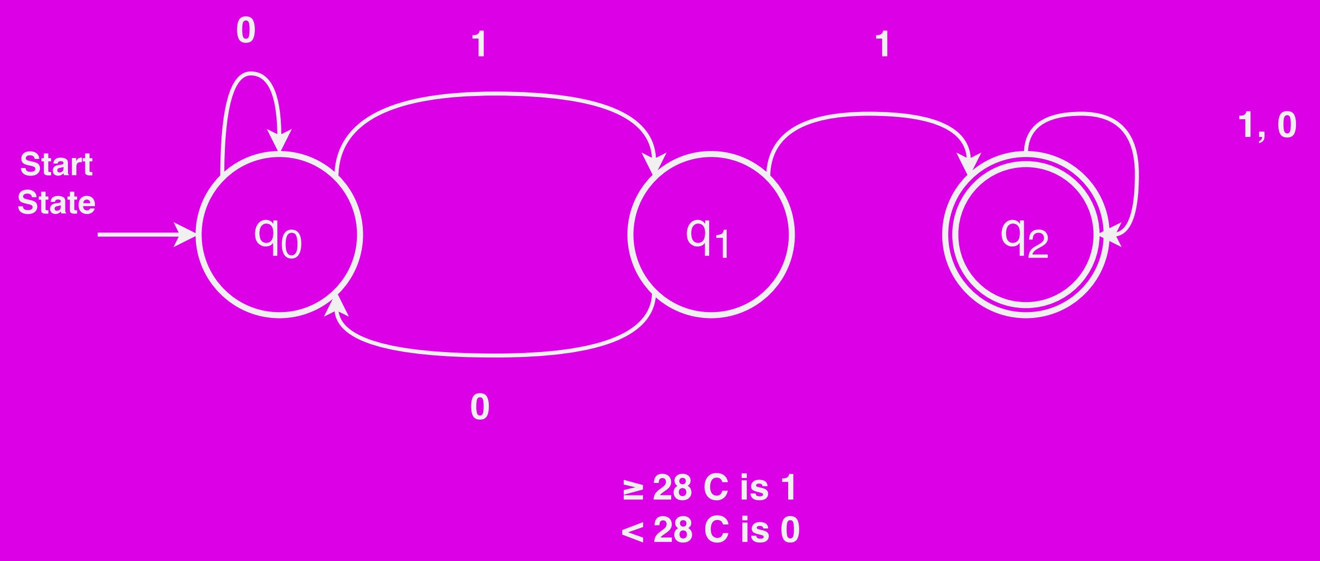


# Automata to determine if a heatwave occurred

- Input: String of weather data
- 🇬🇧 Heatwave: temperature ≥ 28 C for 2 consecutive days

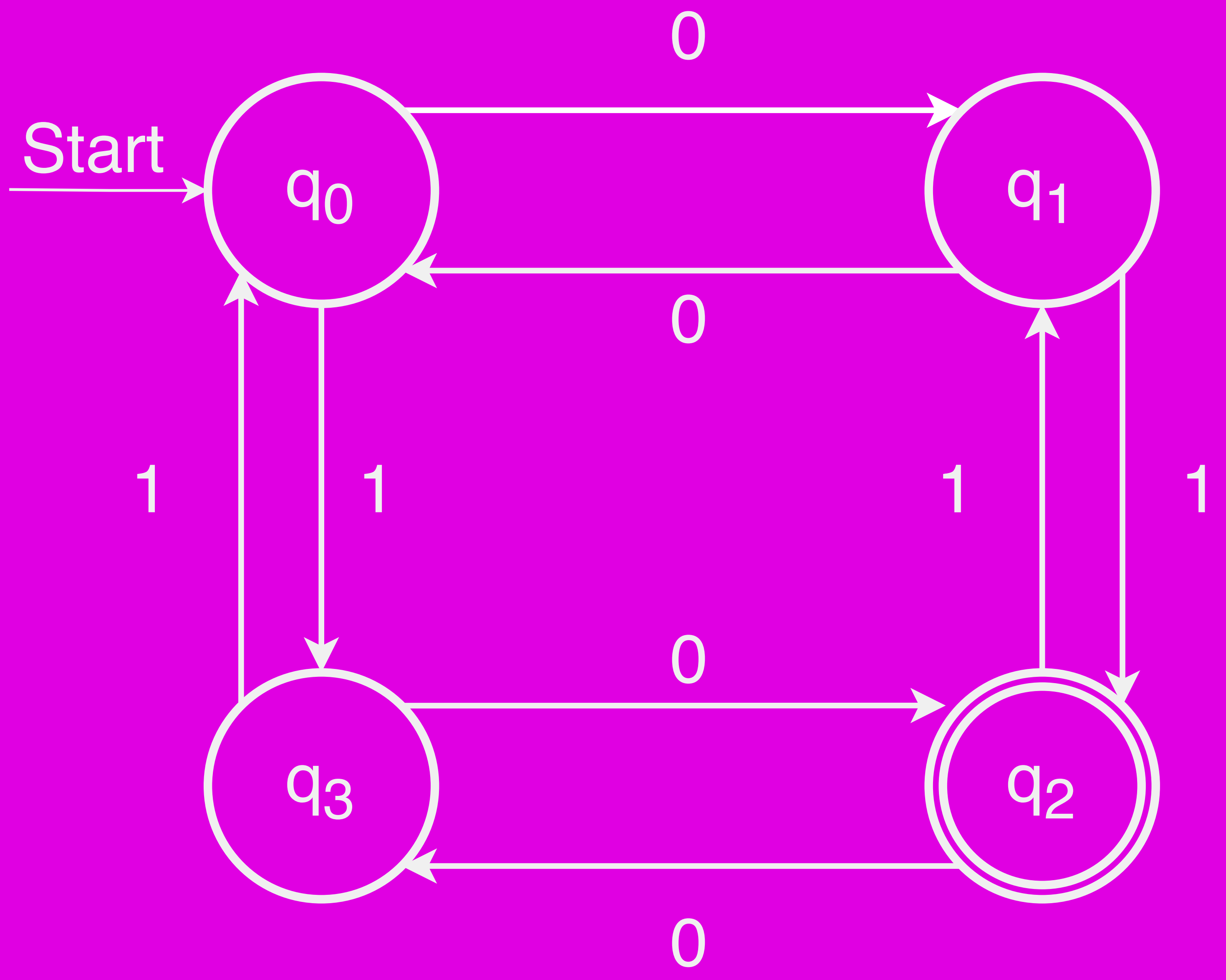



≥ 28 C is 1
< 28 C is 0

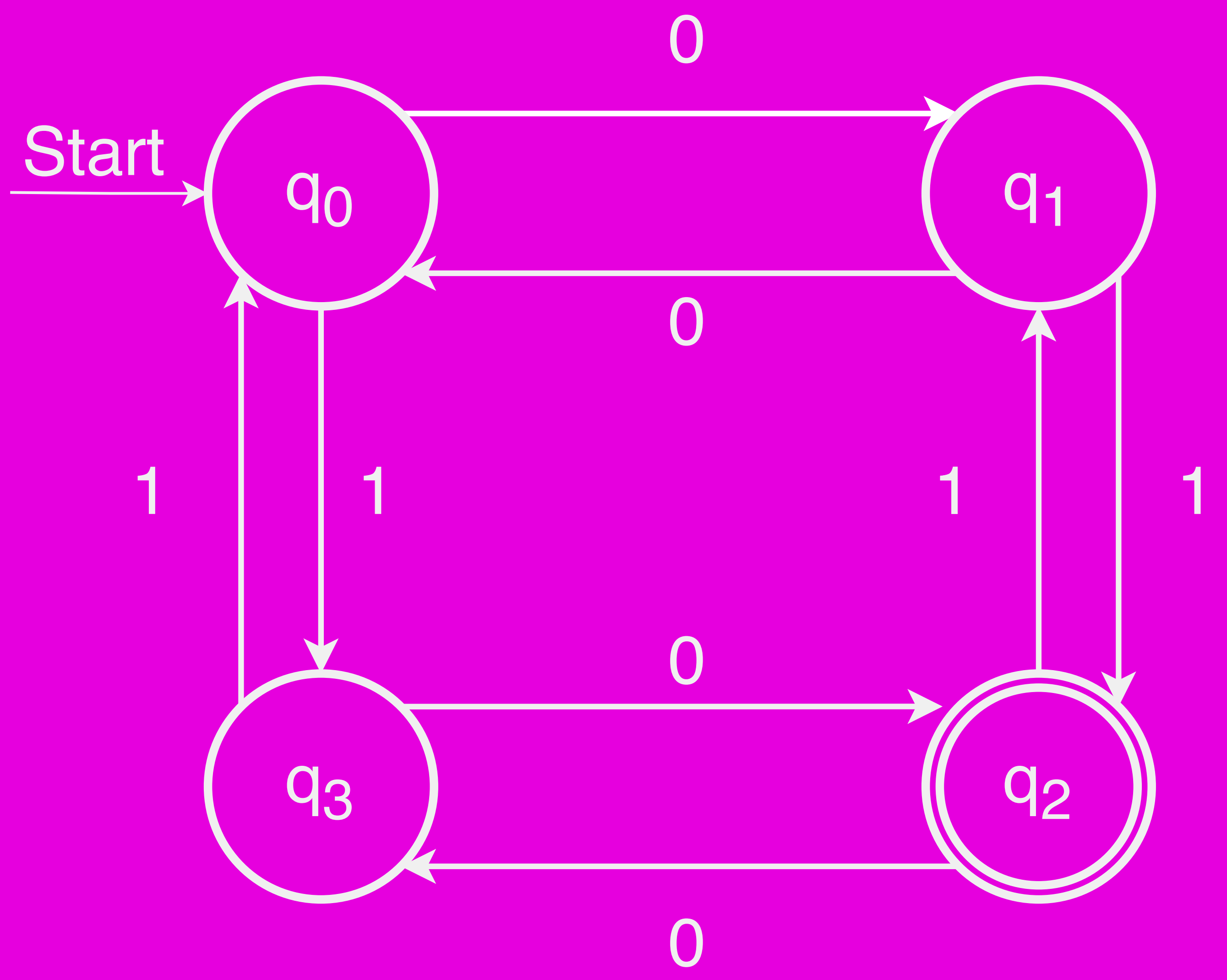



- $L_m$ = { all strings containing 11}
- The automaton above recognises $L_m$
  - Accepts everything within $L_m$ and rejects everything else

# A Simple Finite Automaton

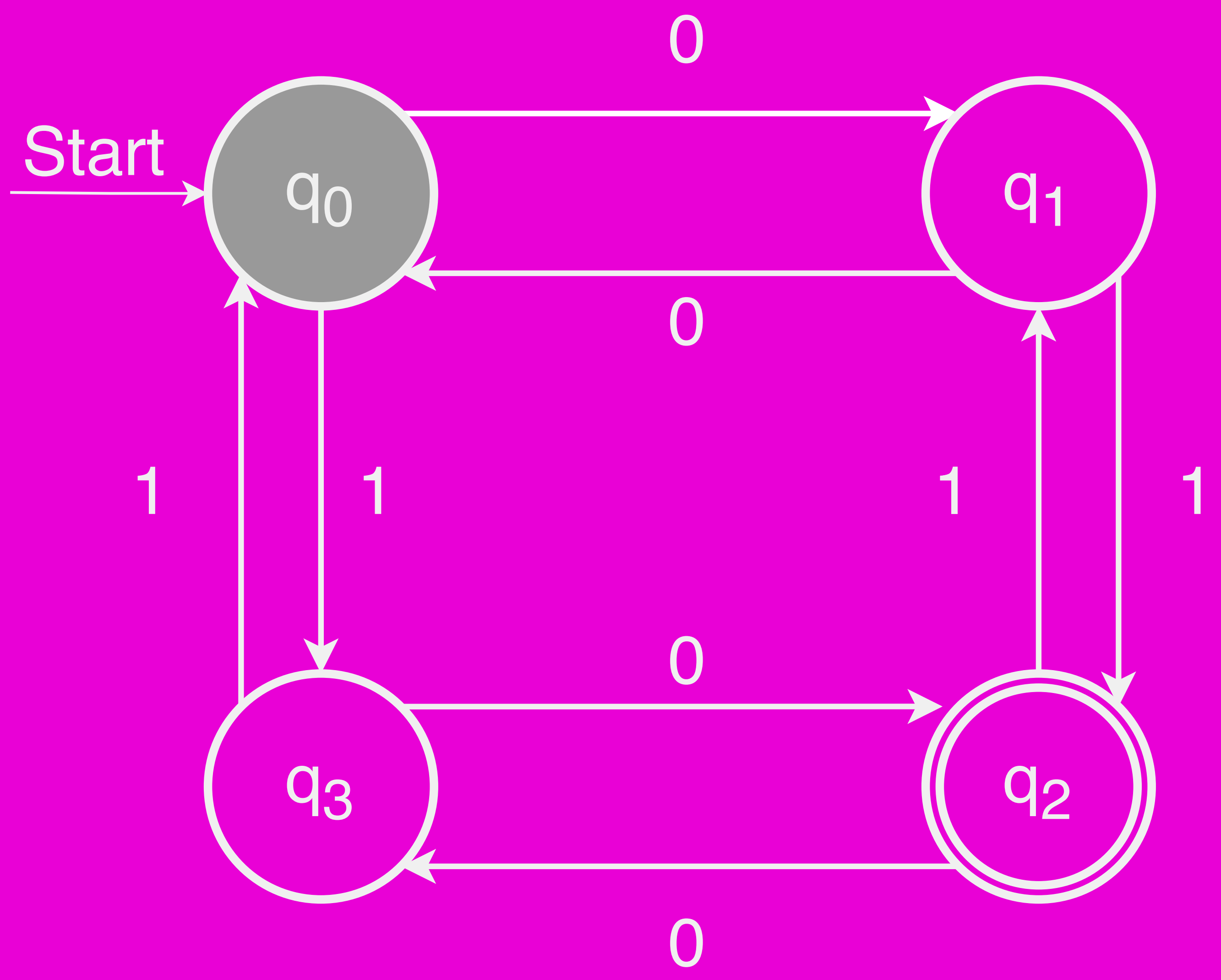


# A Simple Finite Automaton



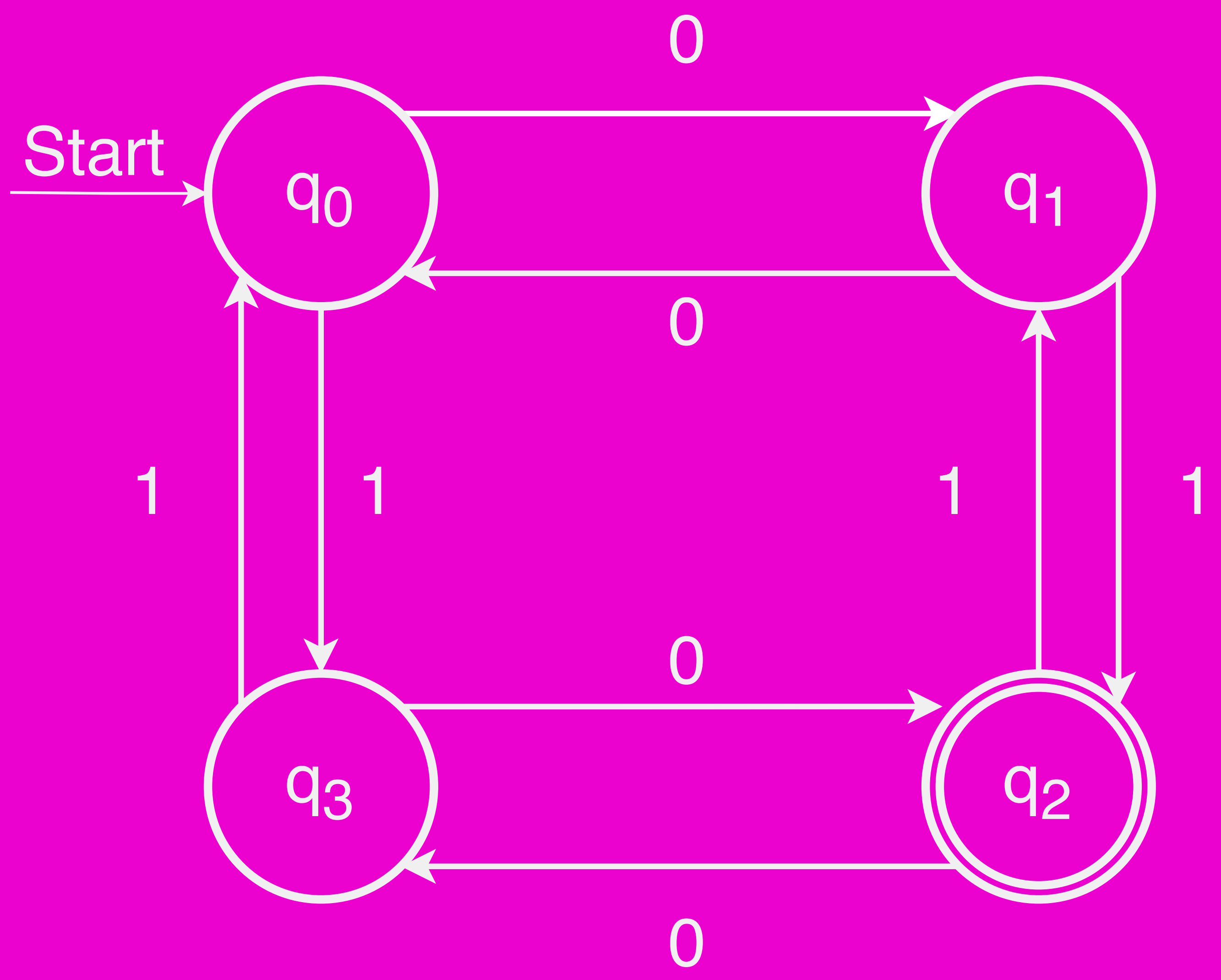

*Each circle represents a* ***state*** *of the automaton*

# A Simple Finite Automaton




*One special state is designated as the start state*

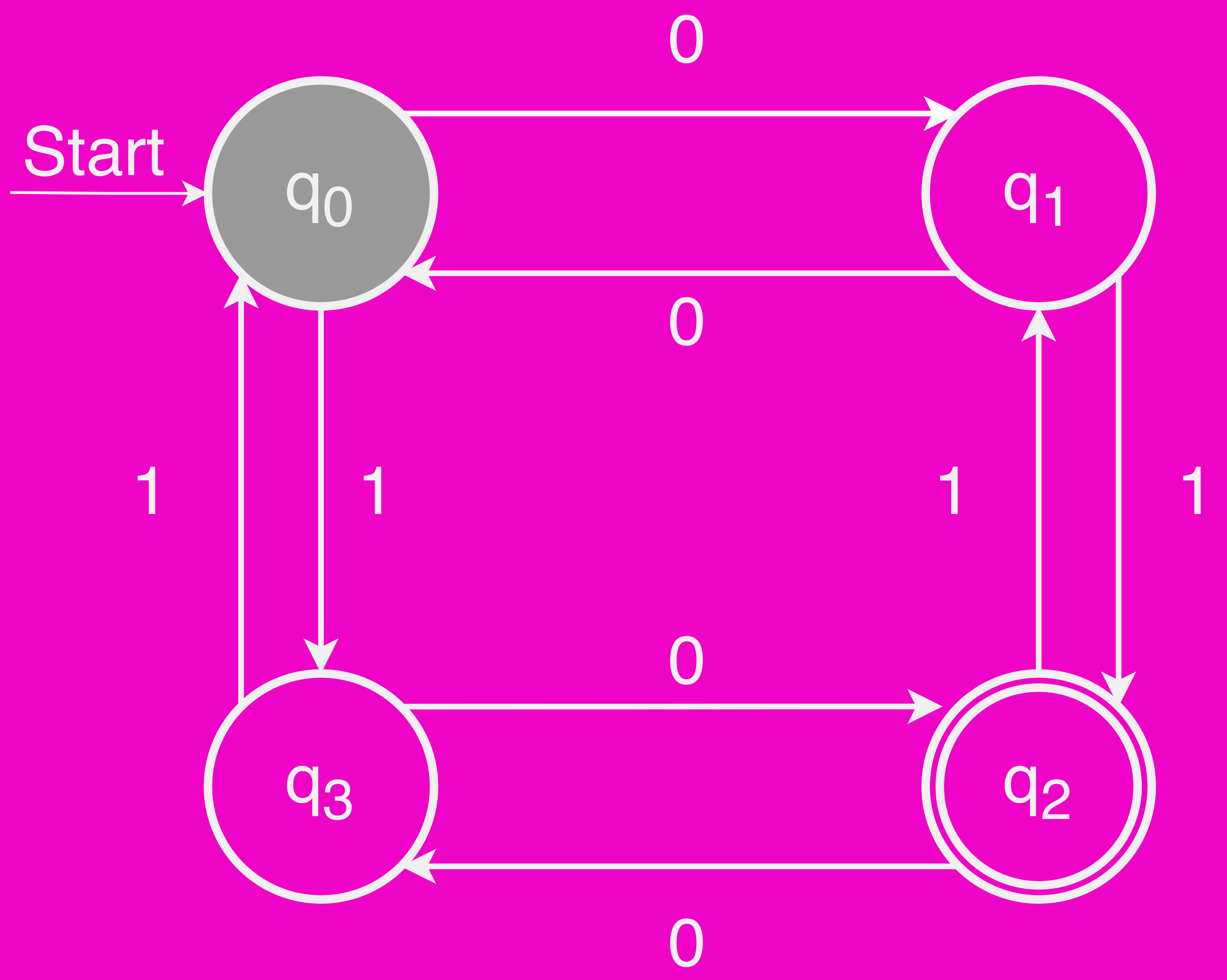
# A Simple Finite Automaton




*The automaton is run on an input string and answers “yes” or “no”*

**0 1 0 1 1 0**

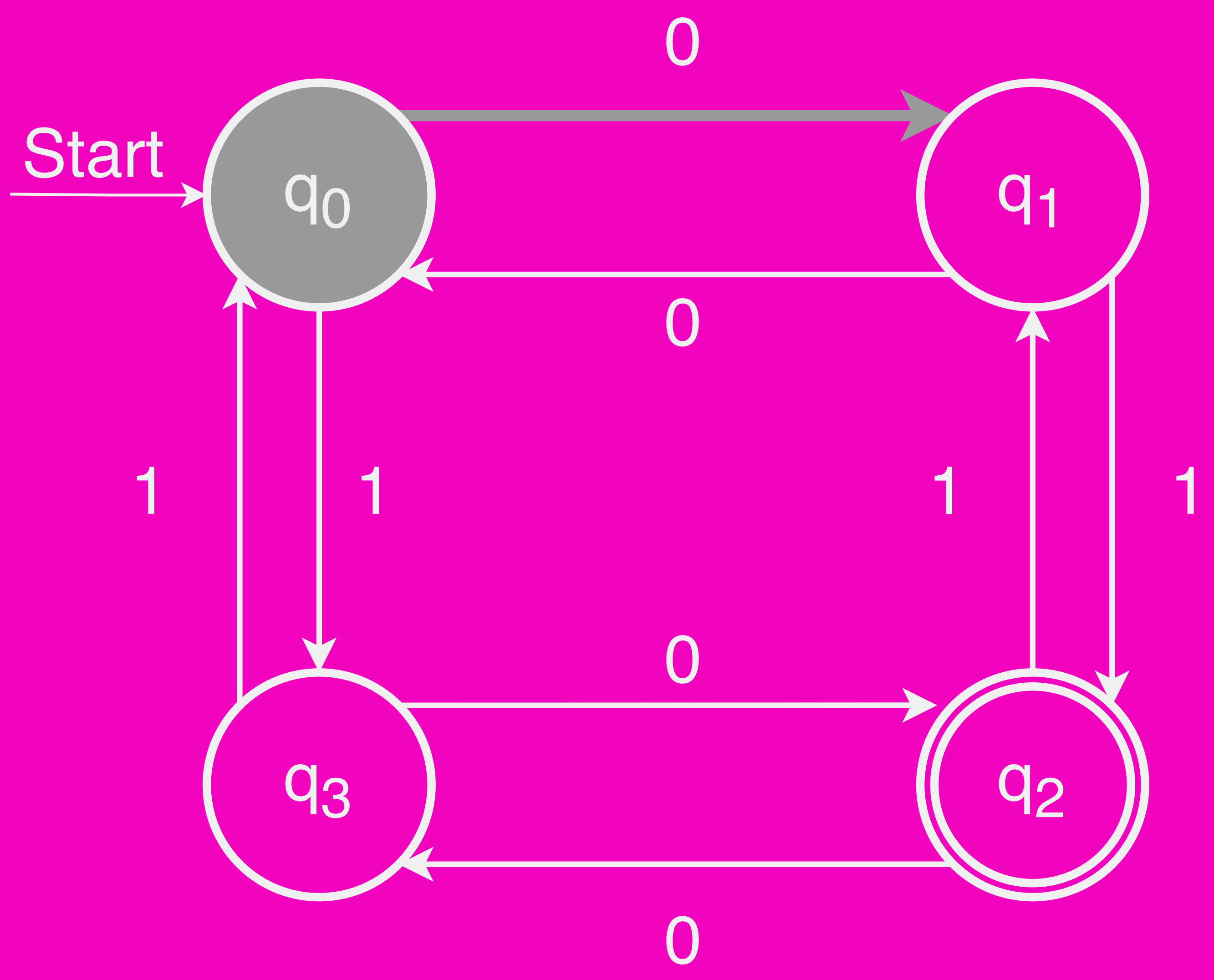
# A Simple Finite Automaton




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

*0 1 0 1 1 0*

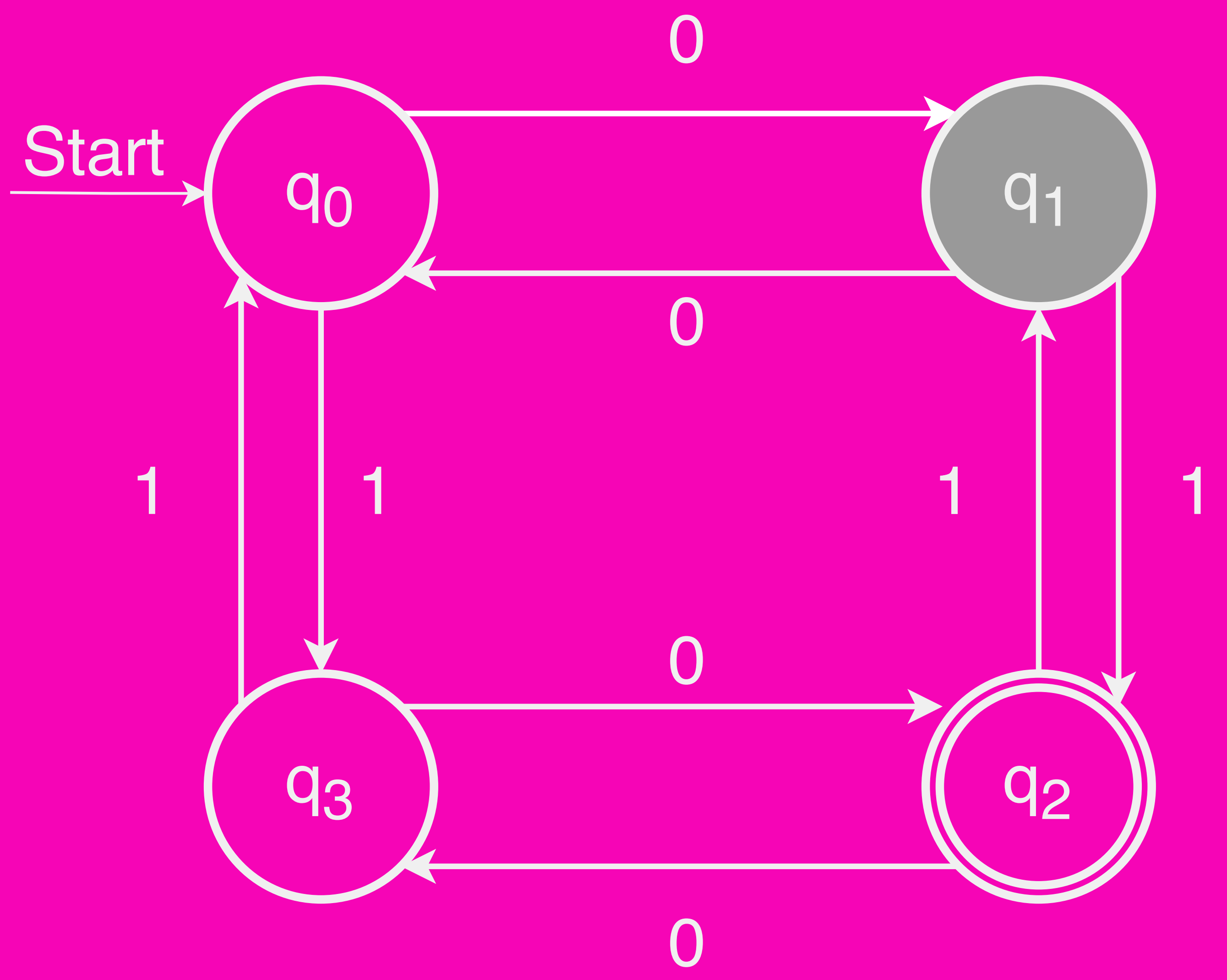
# A Simple Finite Automaton




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

0 1 0 1 1 0

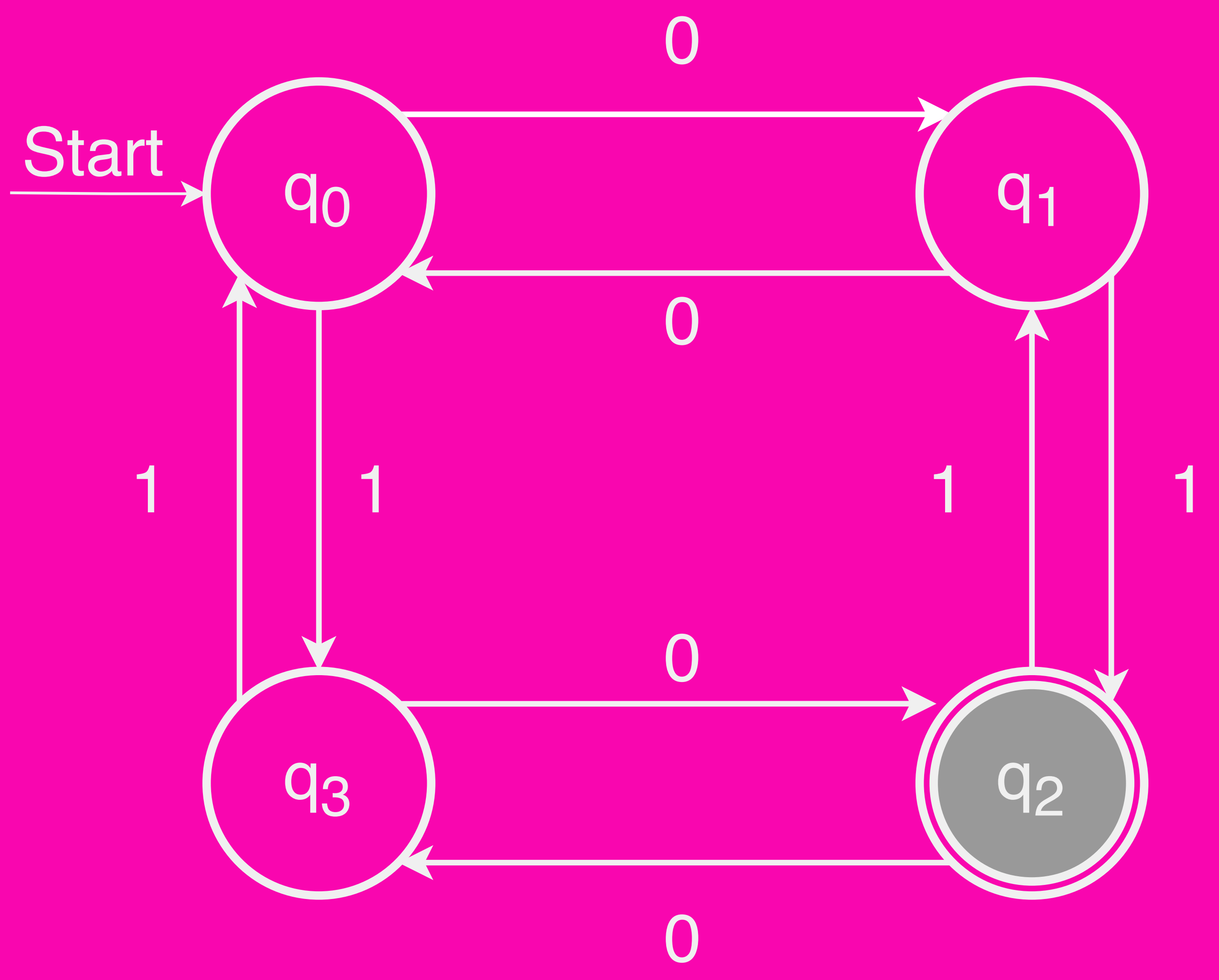
# A Simple Finite Automaton




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

*0 1 0 1 1 0*

## A Simple Finite Automaton

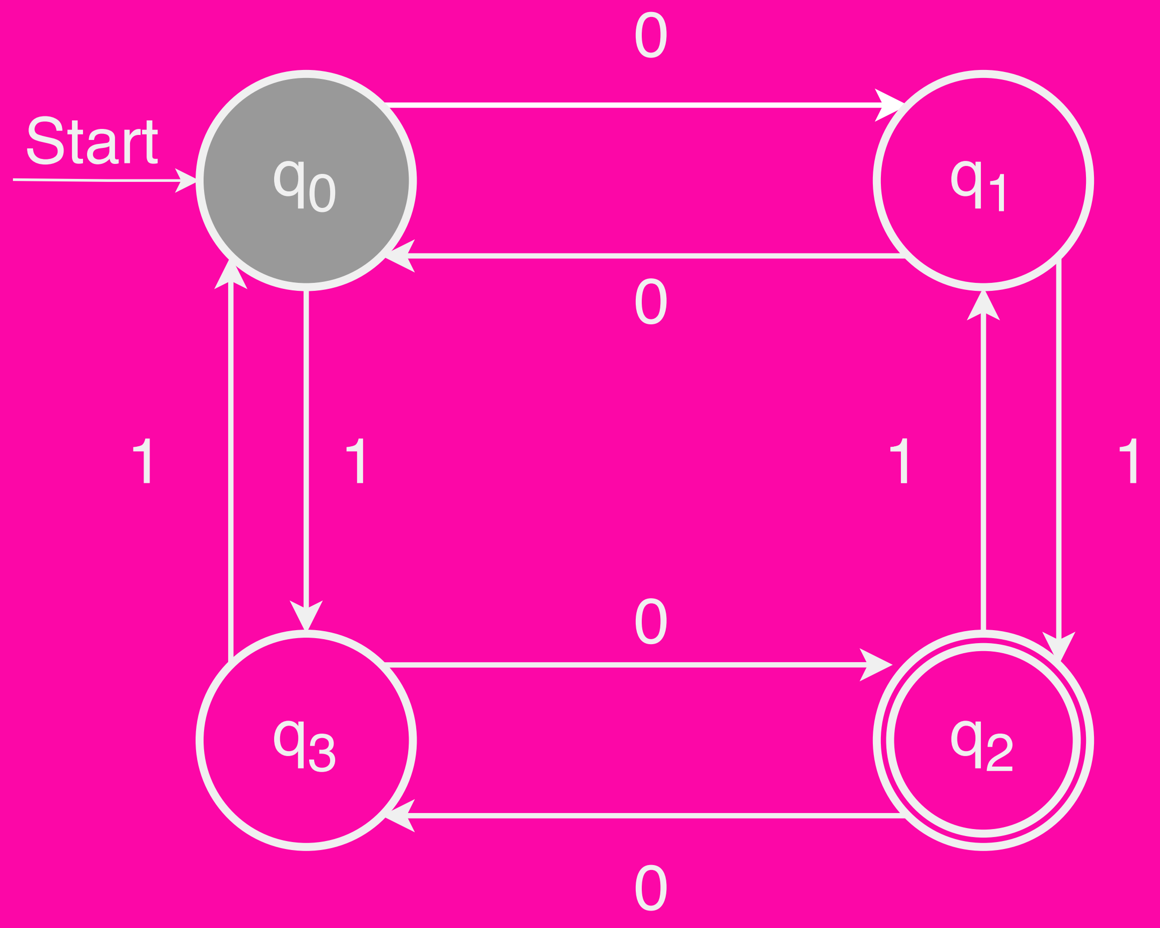




Now that the automaton has looked at all this input, it can decide whether to say "Yes" or "No"

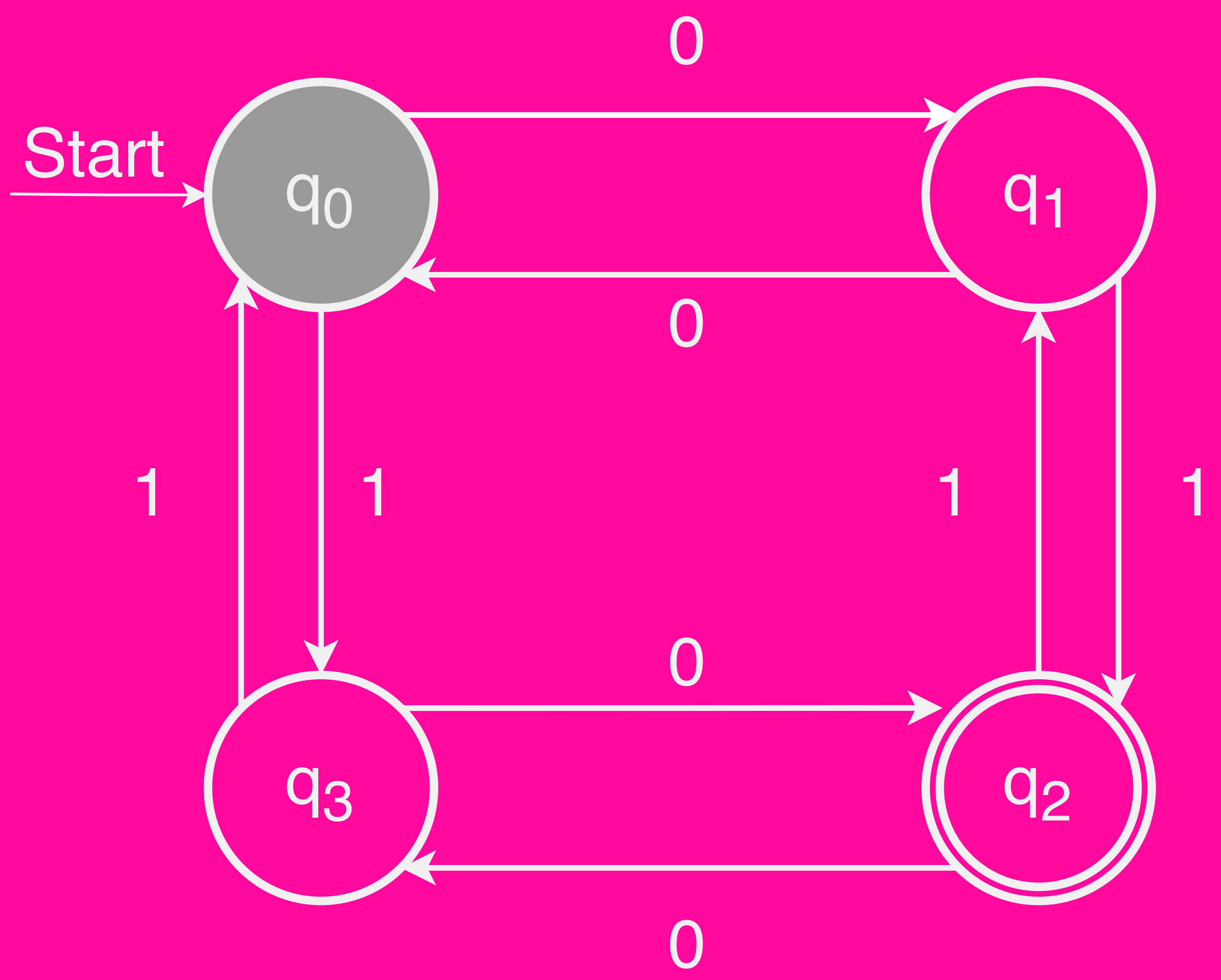
The double circle indicates that this state is an accepting state, so **the automation outputs "Yes"**

# A Simple Finite Automaton




Input: 1 0 1 0 0 0

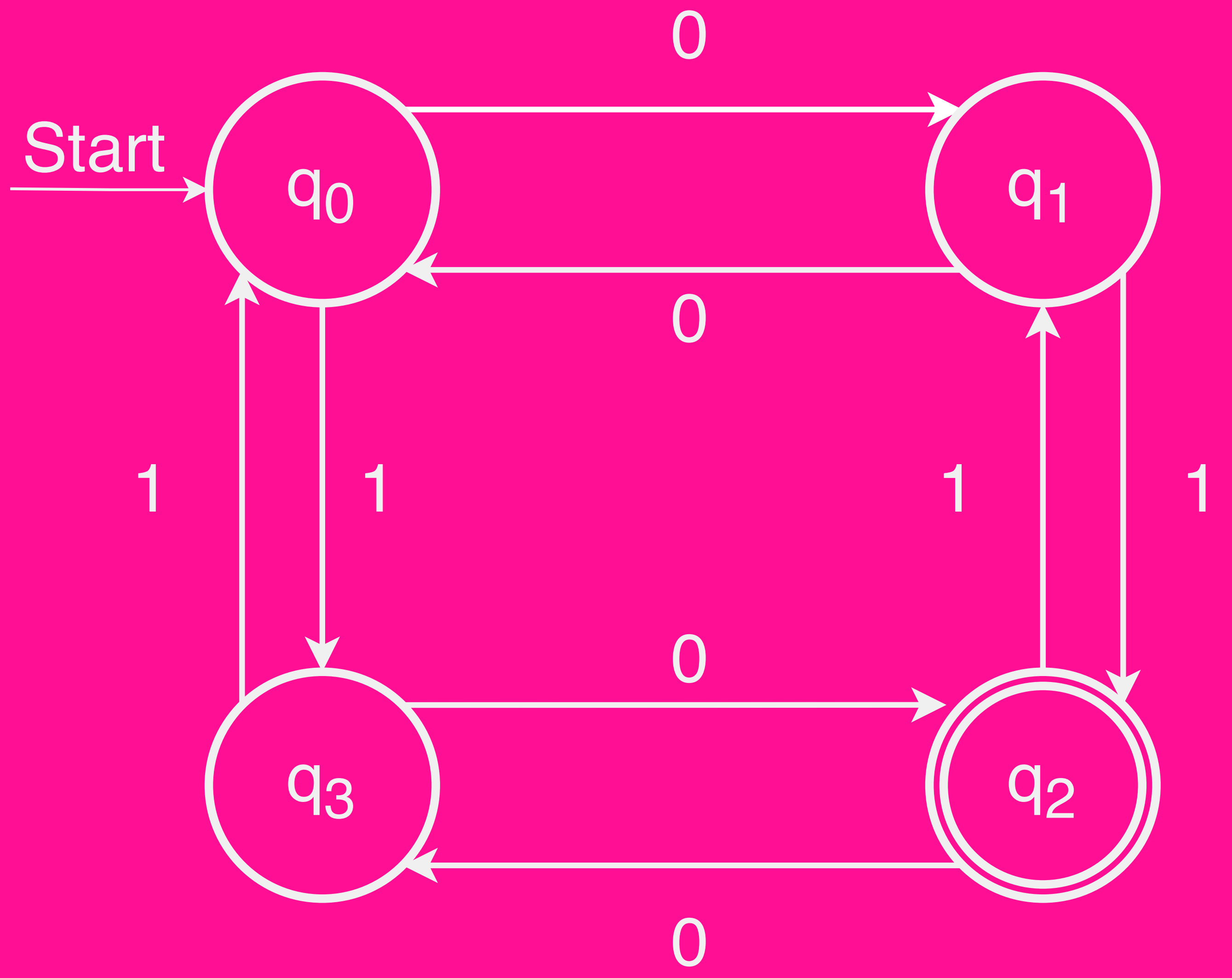
# A Simple Finite Automaton




This state is not an accepting state (it is a rejecting state), so the automaton says "No".

Input: **101000**

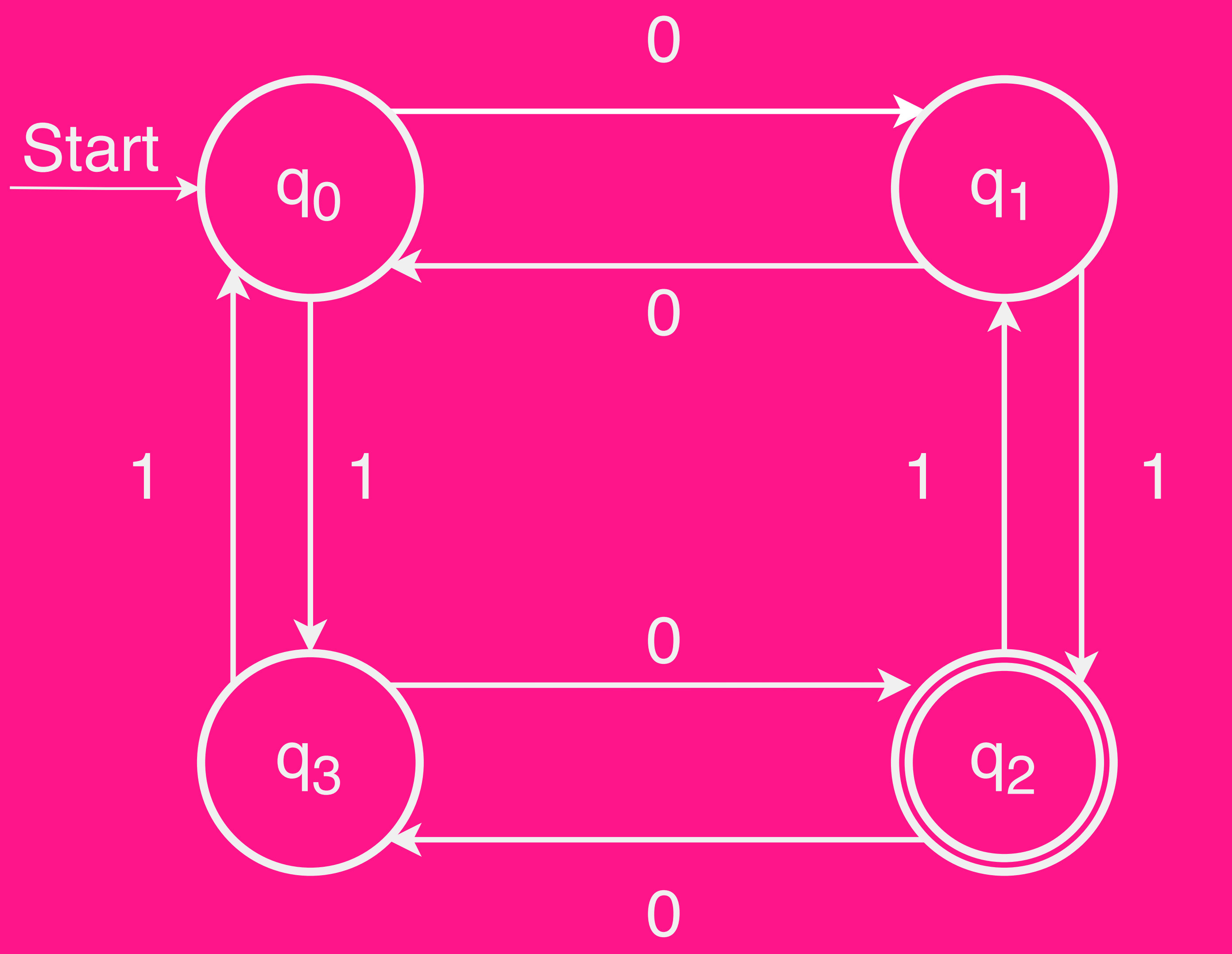
# A Simple Finite Automaton




*Try it yourself! Does this automaton **accept** or **reject**?*

Input: 11011100

# A Simple Finite Automaton





Input: `11011100`

# To Summarise

- 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`

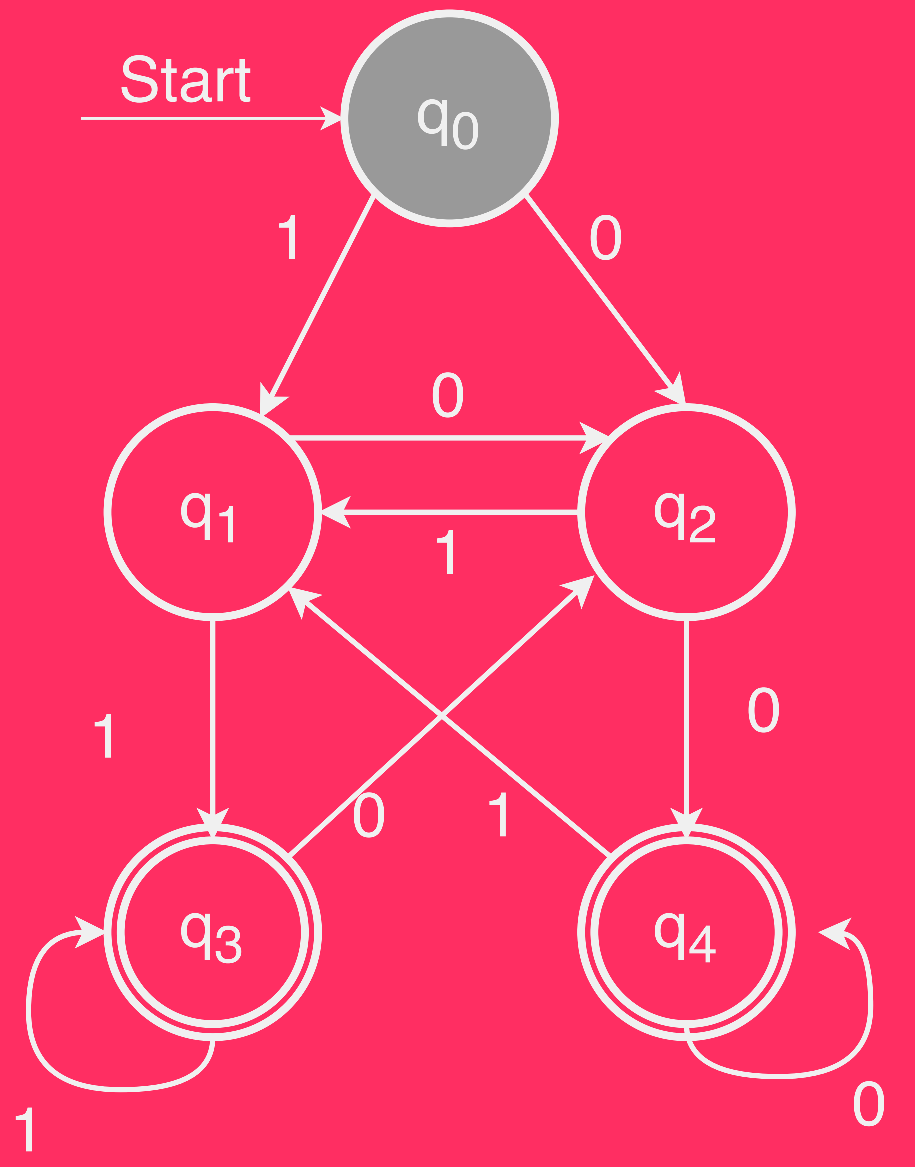
# Short break

Do not leave your seats (5 min)

# FSA Examples

Part 3/4

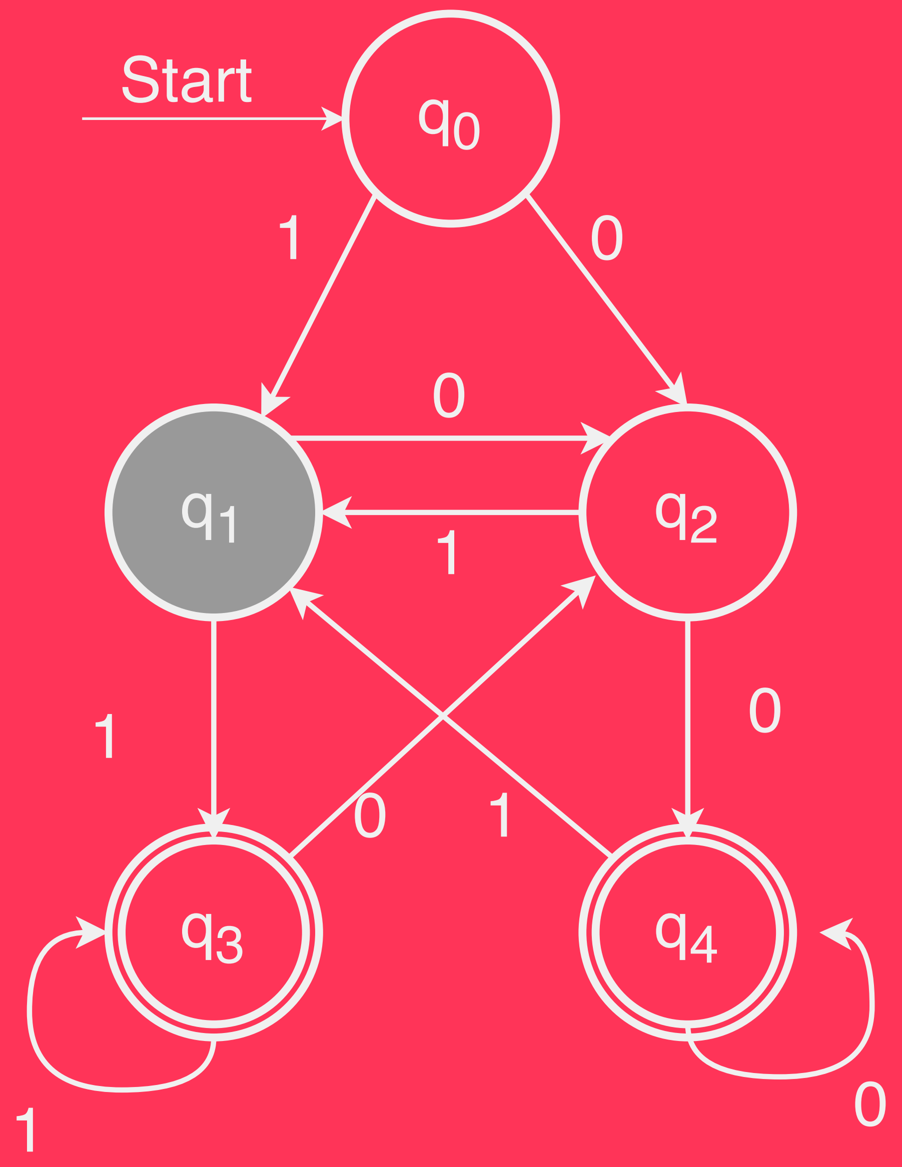
# Just Passing Through




*Input*

*1 1 0 1*

# Just Passing Through




*Input*

**1 1 0 1**

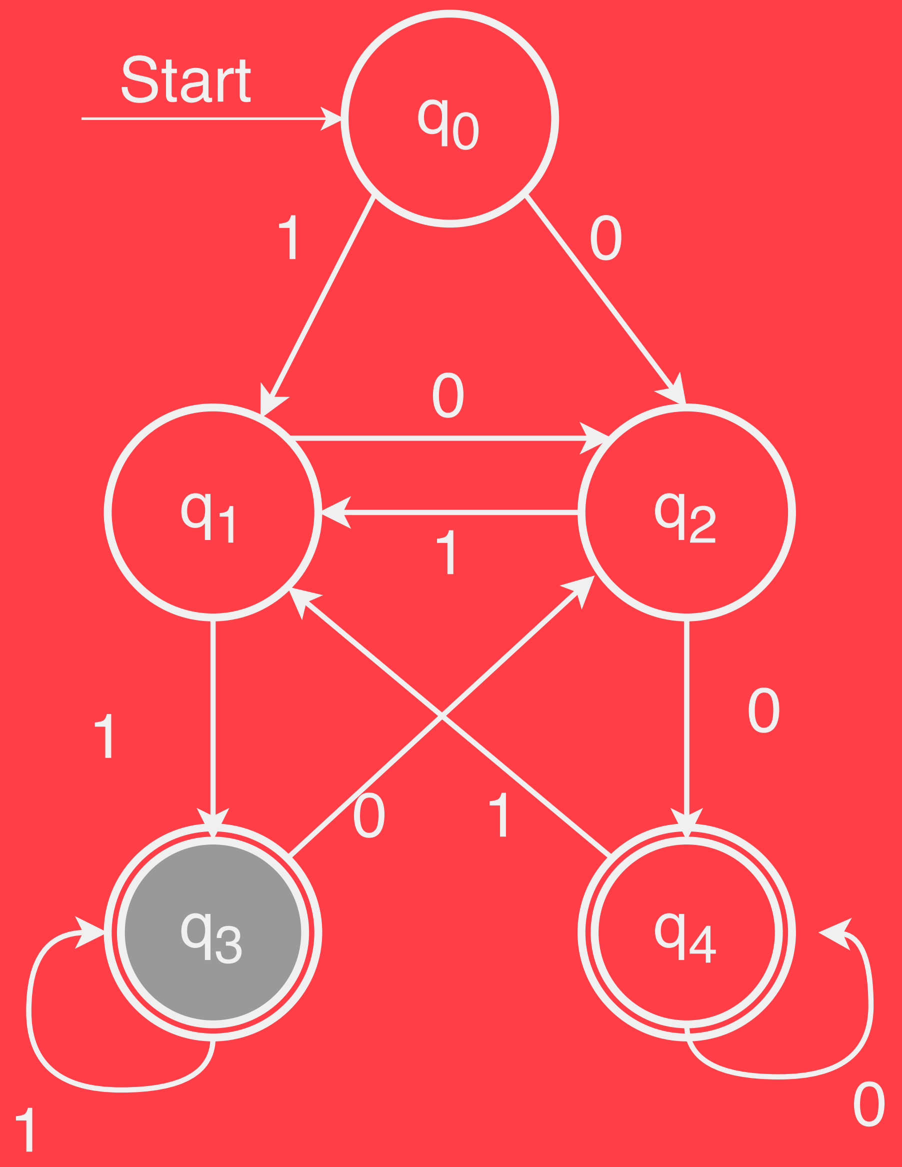
—

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

—

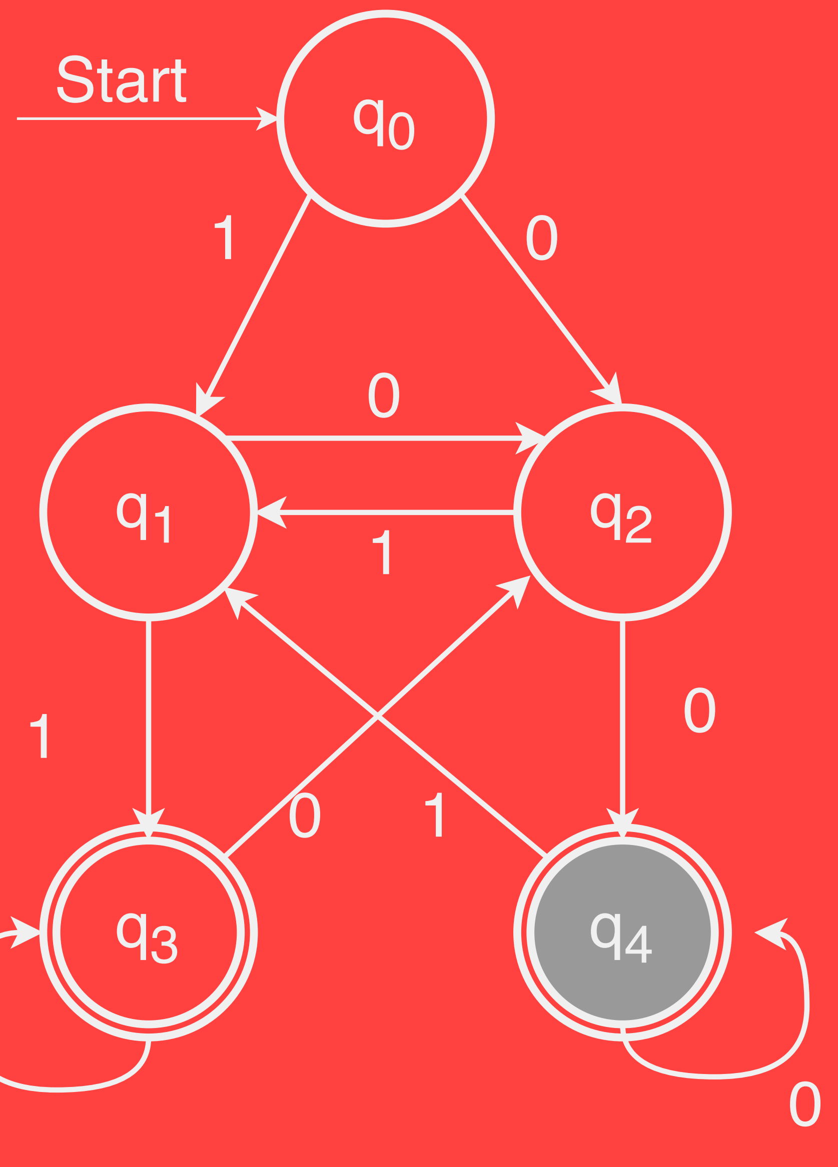
A finite automaton accepts if it ends in an accepting state

# What Does This Accept?



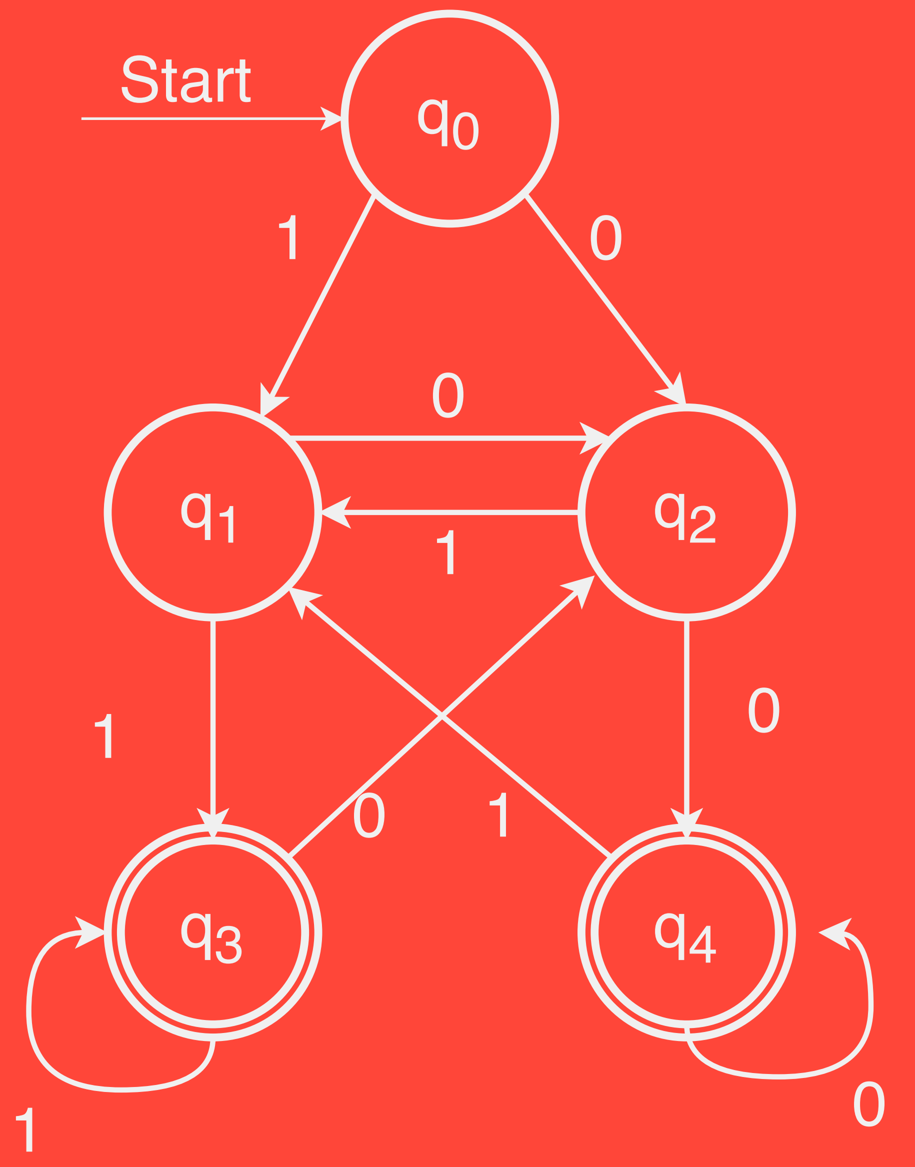

*No matter where we start in the automaton, after seeing two 1's, we end up in accepting state $q_3$*

# What Does This Accept?




*No matter where we start in the automaton, after seeing two 0's, we end up in accepting state $q_4$*

# What Does This Accept?




*This automaton accepts a string in {0, 1} * if and only if the string ends in 00 or 11*

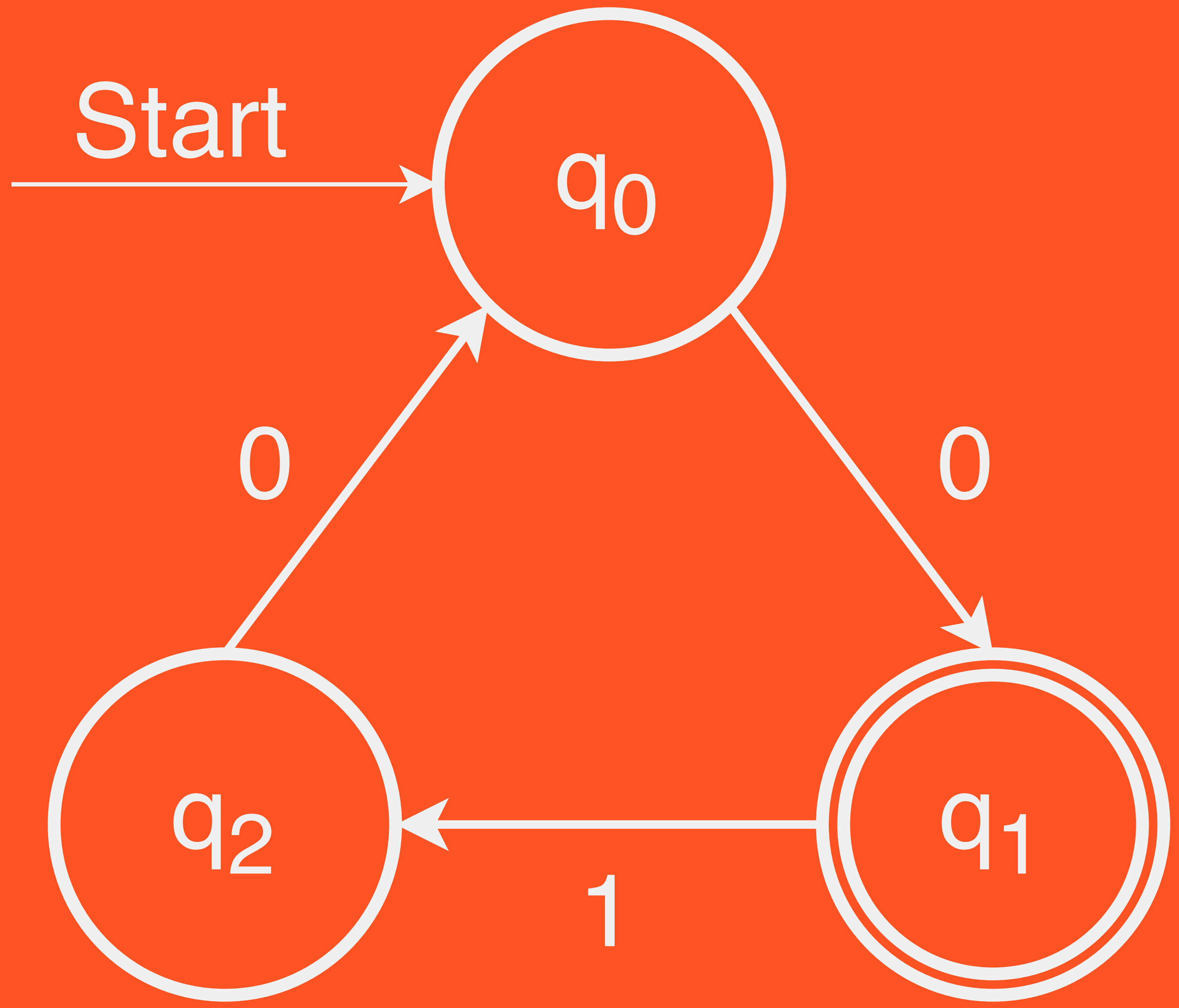
The **language of an automaton** is the set of strings that it accepts

- If **D** is an automaton that processes characters from the alphabet $\Sigma$, then **L(D)** is formally defined as:
  - $L(D) = \{w \in \Sigma^* \mid D \text{ accepts } w\}$

# Quick Quiz

- How many of the following statements are true?
  - **A language** of an automaton can have an infinitely long string (or many of them) in it
  - **A language** of an automaton can contain infinitely many strings
  - **A language** of an automaton can contain no string

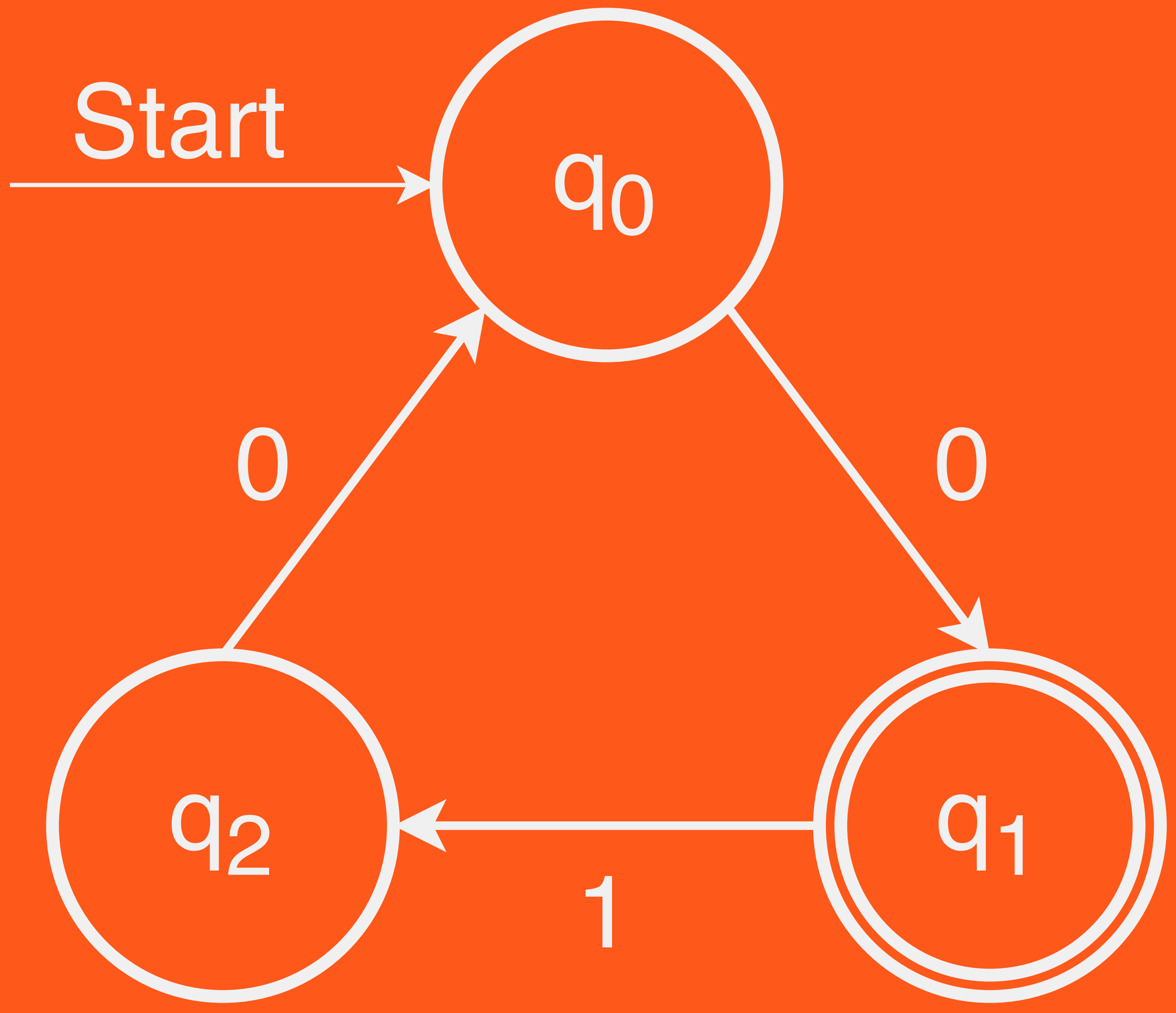
# A Small Problem




*Input:*

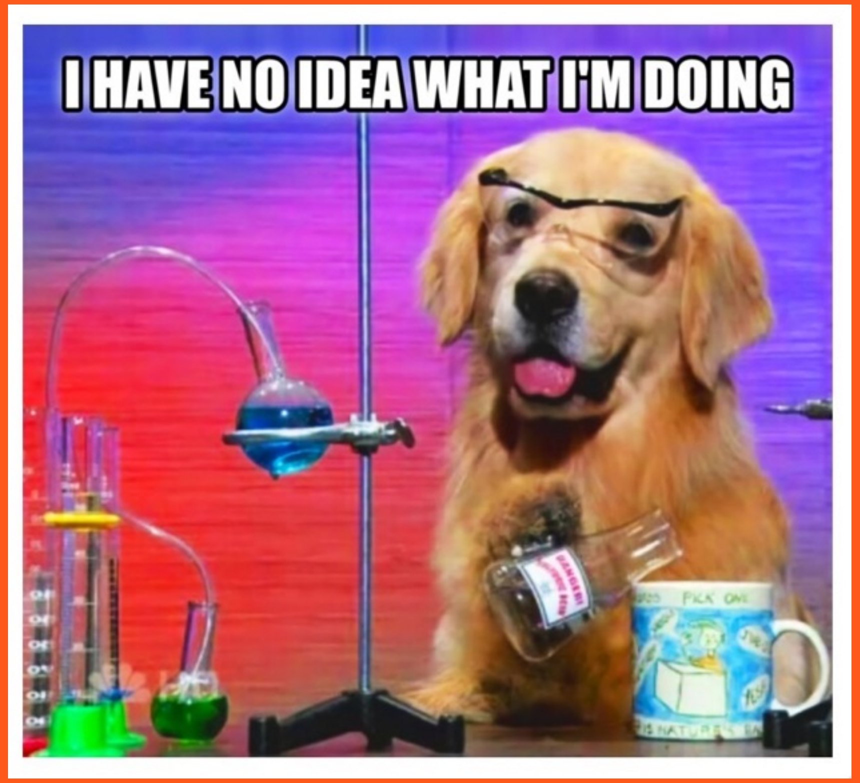
**0 1 1 0**

# A Small Problem

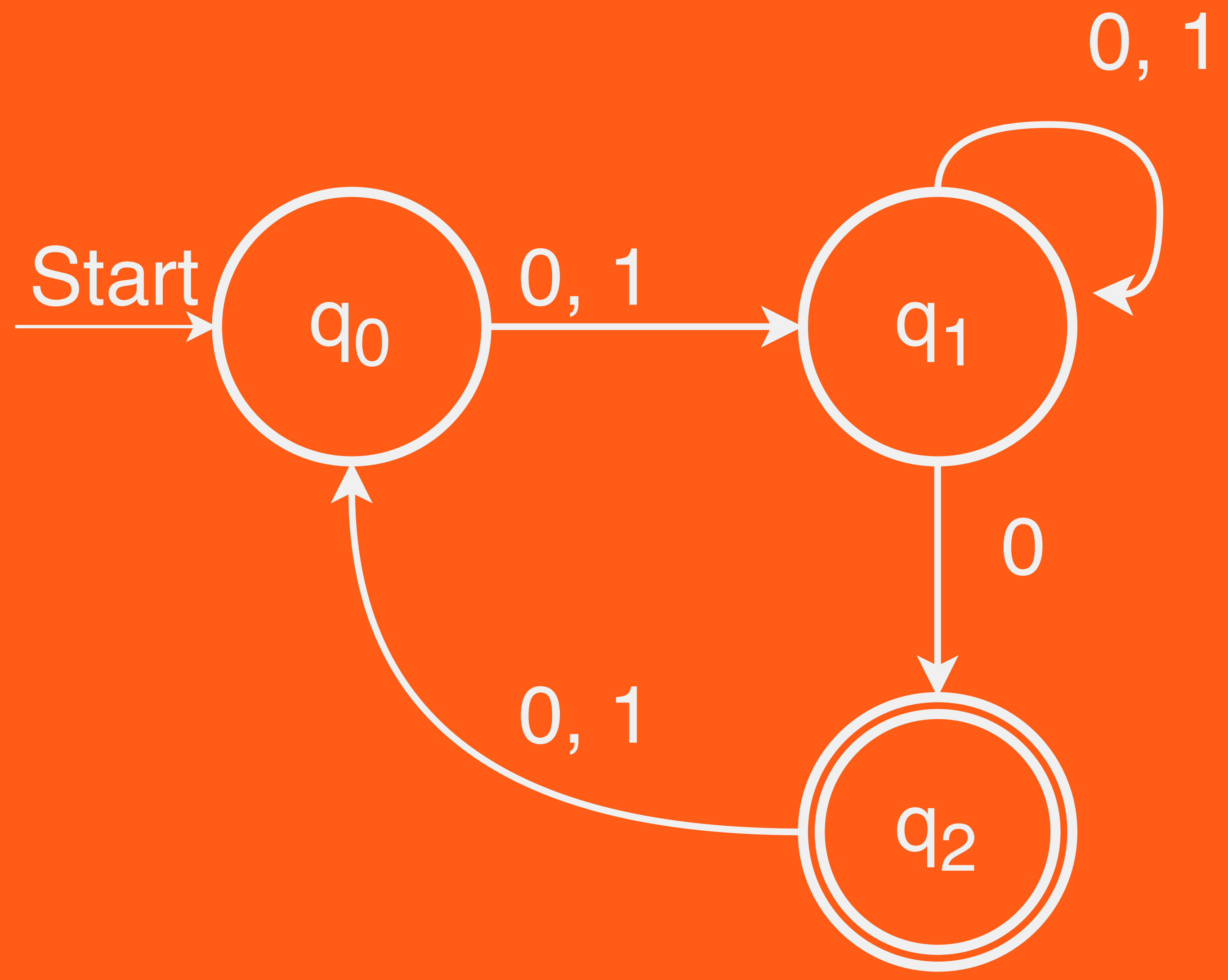



*Input:*

0 1 1 0

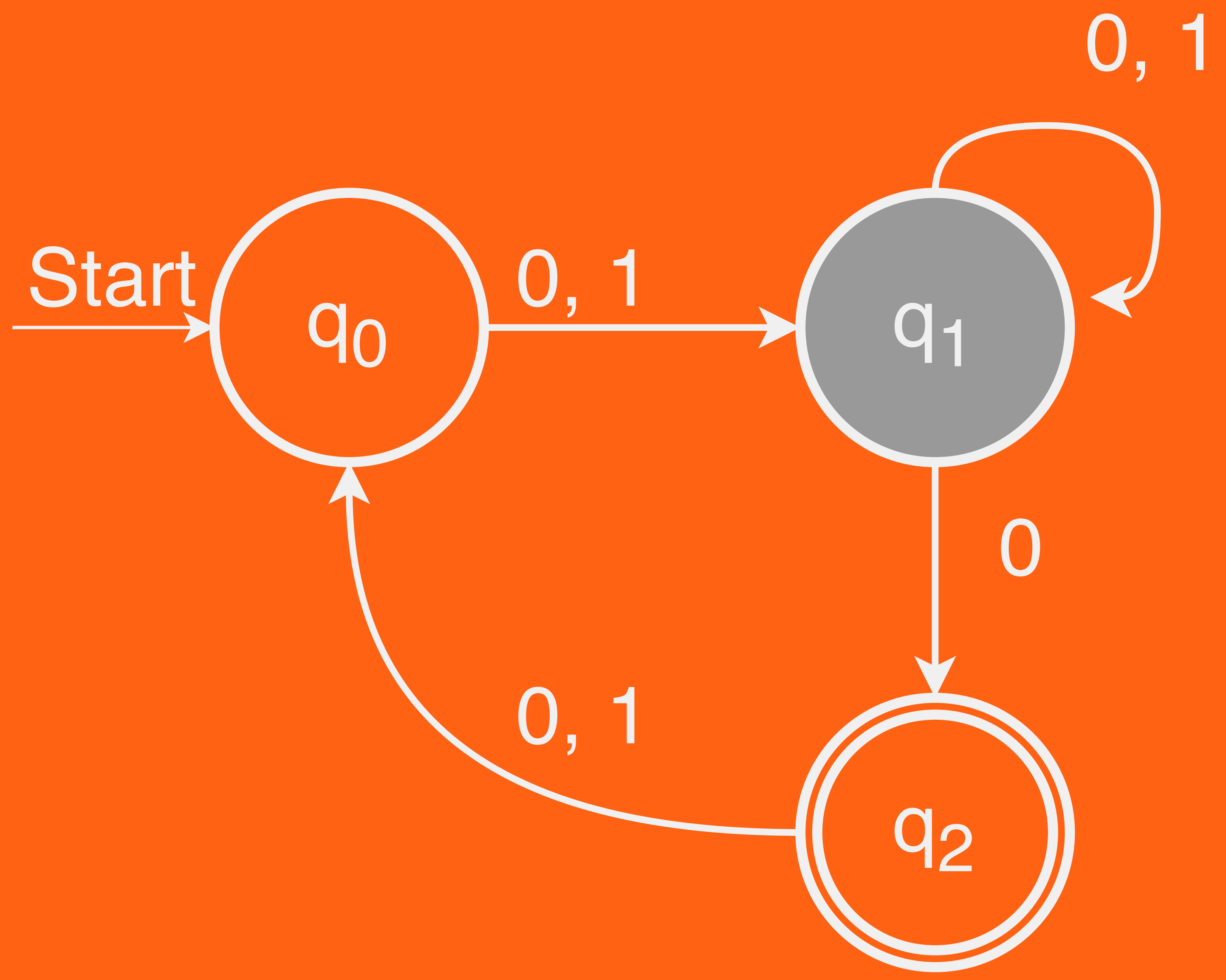


# A Small Problem



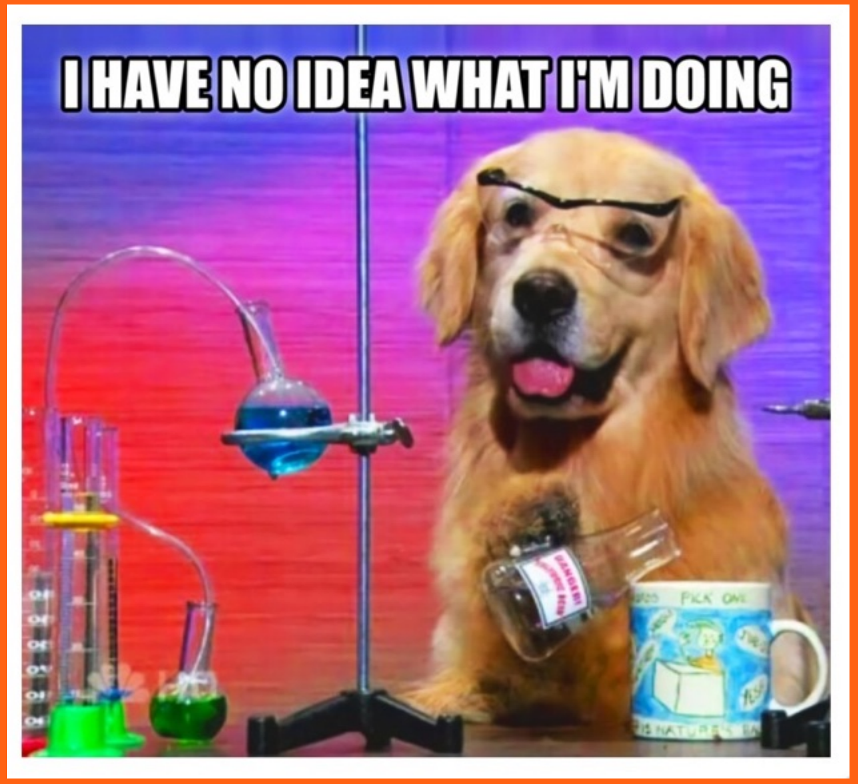

*Input:*
**0 0 0**

# A Small Problem




Input:

`0 0 0`



# 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 behaviour 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?

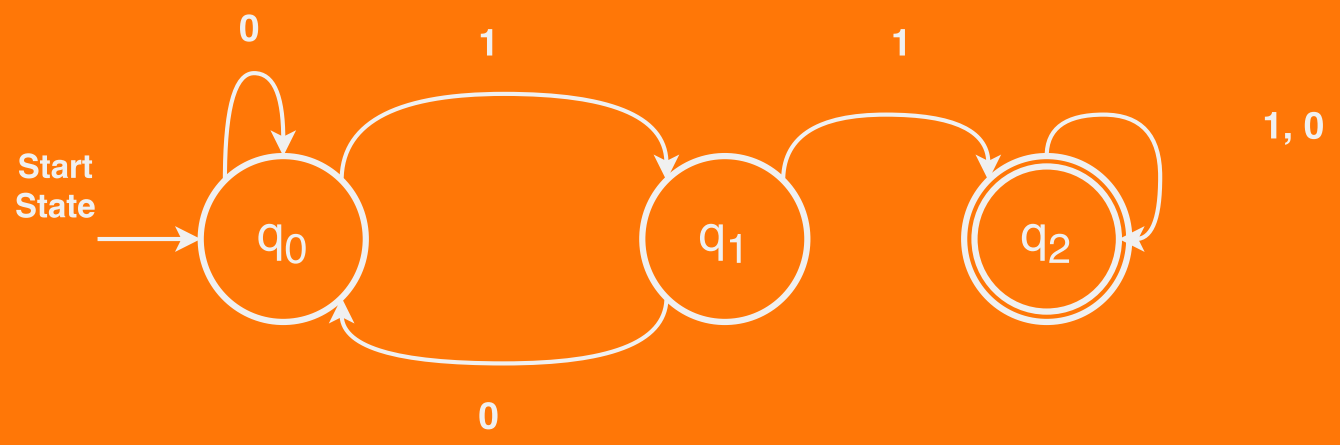
# Deterministic Finite Automaton

Part 4/4

# DFAs

- 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

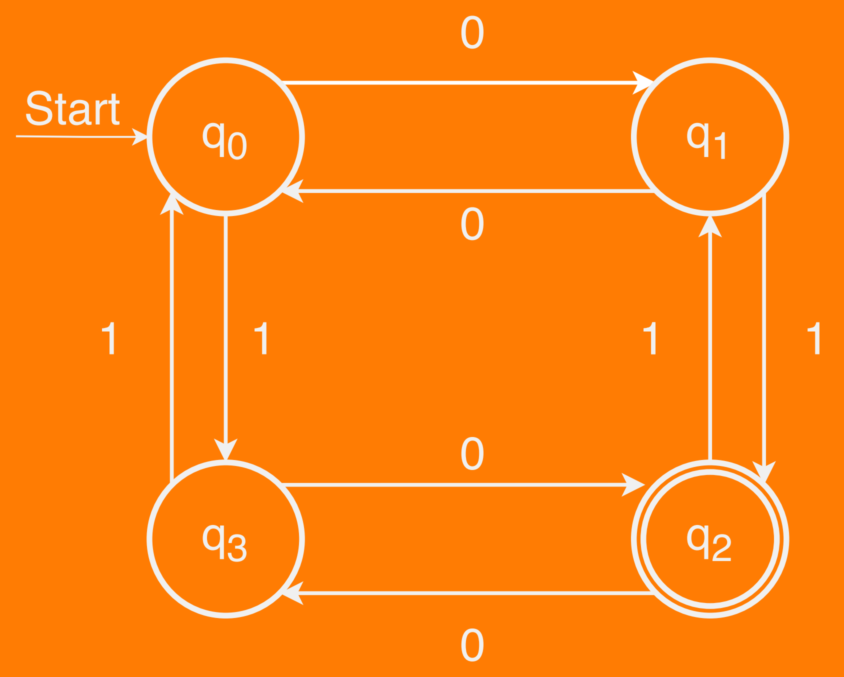
# Deterministic Finite Automaton (Formal Definition)

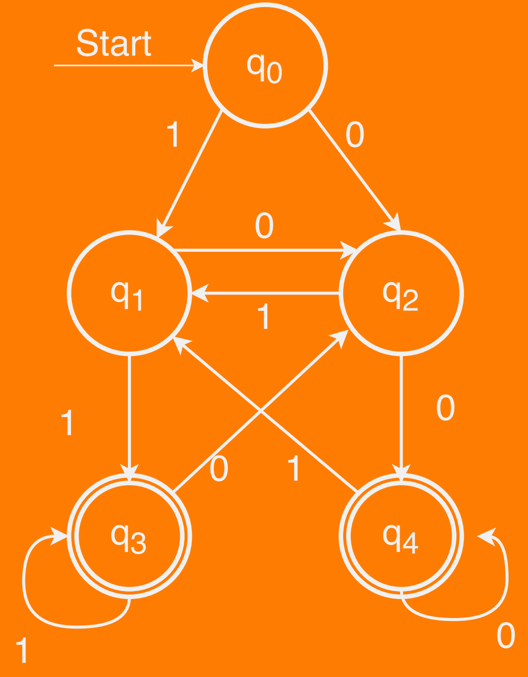


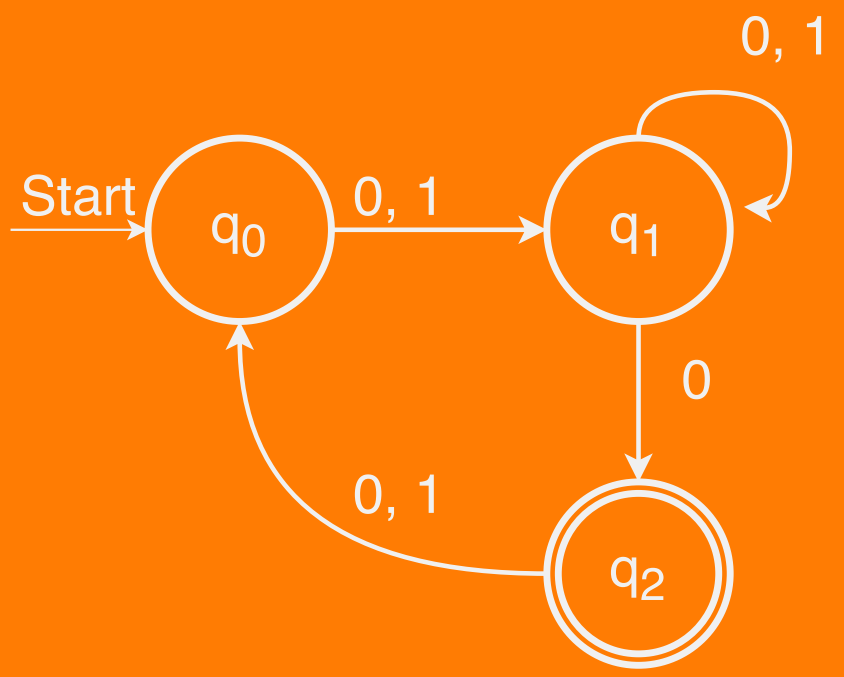

D = (Q, Σ, δ, $q_0$, F)

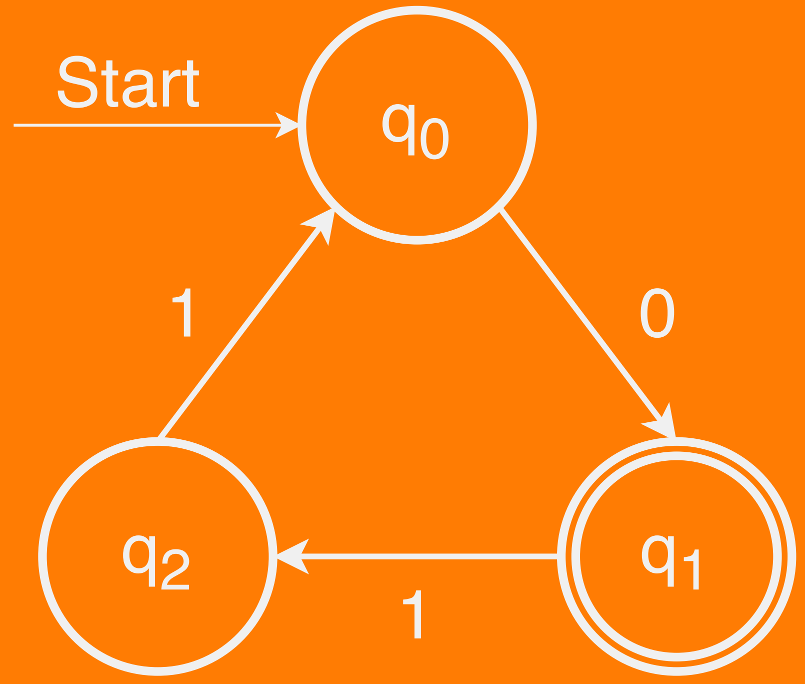
- Q is the set of states [Q = { $q_0$, $q_1$ , $q_2$ } ]
- Σ is the alphabet [Σ = {1,0} ]
- δ is the transition function [I will cover this tomorrow]
- $q_0$ is the start state
- F is an accepting state [F = { $q_2$ } ]

# How many of these are DFAs over {0, 1}?



Is this a DFA?
Drinking Family of Alpaca

# Designing DFAs

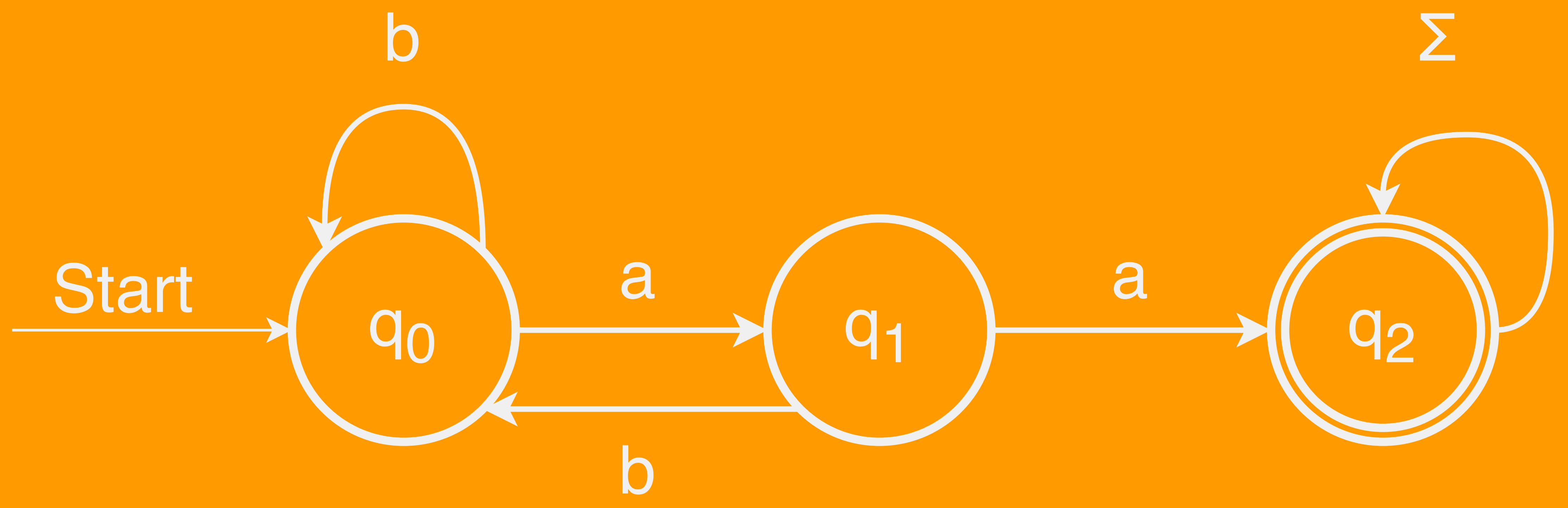
- At each point in its execution, the DFA can only remember what state it is in
- **DFA Design Tip:** Build each state to correspond to some piece of information you need to remember
  - Each state acts as a “memento” of what you're supposed to do next
  - Only finitely many different states means only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

# Recognizing Languages with DFAs

L = { w ∈ {a,b} * | w contains aa as a substring }

# Recognizing Languages with DFAs

L = { w ∈ {a,b} * | w contains aa as a substring }



# More Elaborate DFAs

L = { w ∈ {a,*,/} * | w represents a Java-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:

```
/*a*/
/**/
/***/
/*aaa*aaa*/
/*a/a*/
```

# Rejected:

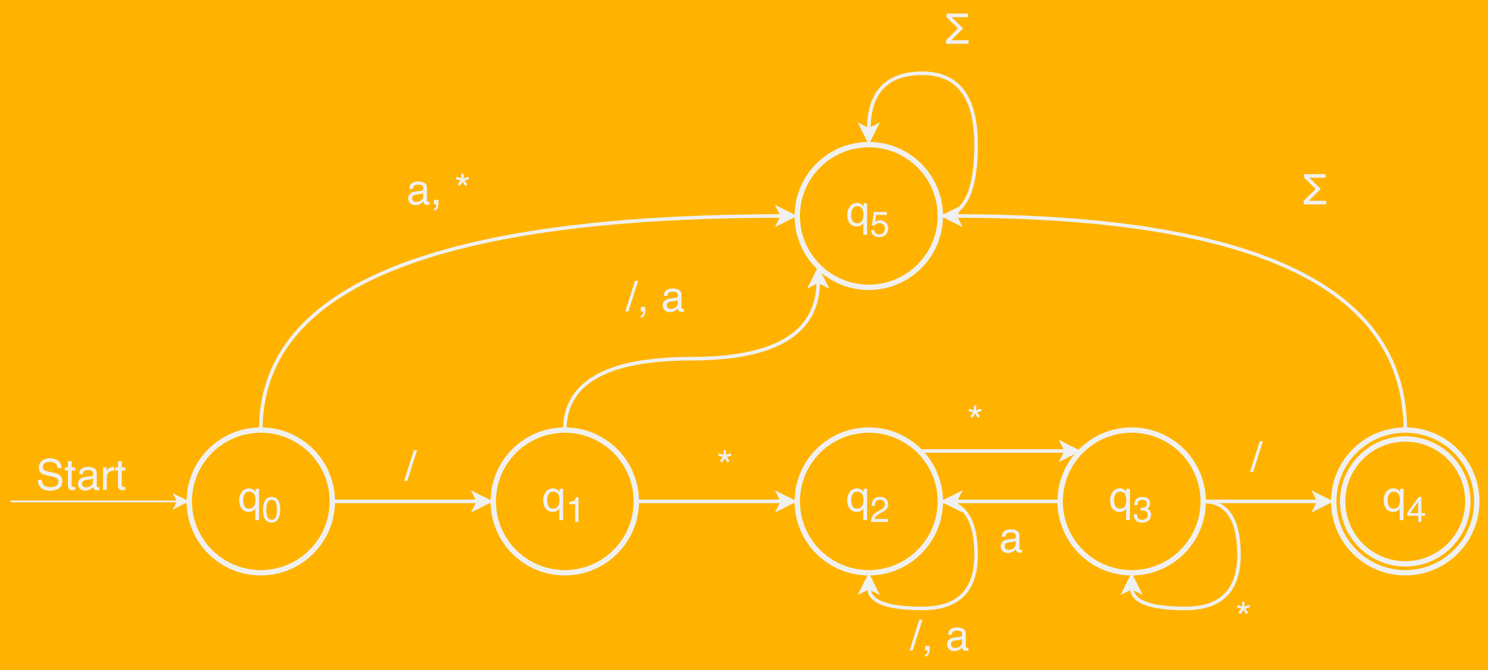
```
/**
/**/a/*aa*/
aaa/**/aa
/*/
/**a/
//aaaa
```

# More Elaborate DFAs

L = { w ∈ {a,*,/} * | w represents a Java-style comment }

# More Elaborate DFAs

L = { w ∈ {a,*,/}* | w represents a Java-style comment }



# See you tomorrow! 👋