# Finite Automata

## Part One

# Recap

Let $P$ be some predicate. The ***principle of mathematical induction*** states that if

If it starts true…

$$P(0) \text{ is true}$$

…and it stays true…

and

$$\forall k \in \mathbb{N}.\ (P(k) \to P(k+1))$$

then

$$\forall n \in \mathbb{N}.\ P(n)$$

…then it's always true.

Let $P$ be some predicate. The ***principle of complete induction*** states that if

**$P(0)$ is true**

If it starts true…

and

…and it stays true…

**for all $k \in \mathbb{N}$, if $P(0)$, …, and $P(k)$ are true, then $P(k+1)$ is true**

then

**$\forall n \in \mathbb{N}.\ P(n)$**

…then it's always true.

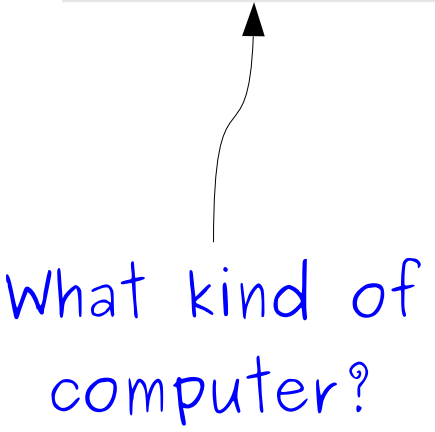
# Outline for Today

- ***Computability Theory***
  - What problems can we solve with a computer?
- ***Formal Language Theory***
  - What is a language?
- ***Finite Automata***
  - A very simple model of a computing device.

# Computability Theory

What problems can we solve with a computer?

What problems can we solve with a computer?



# Two Challenges

- Computers are dramatically better now than they've ever been, and that trend continues.
- Writing proofs on formal definitions is hard, and computers are *way* more complicated than sets, graphs, or functions.
- ***Key Question:*** How can we prove what computers can and can't do...
  - ... so that our results are still true in 20 years?
  - ... without multi-hundred page proofs?

# Enter Automata

- An ***automaton*** (plural: ***automata***) is a mathematical model of a computing device.
- It's an ***abstraction*** of a real computer, the way that graphs are abstractions of social networks, transportation grids, etc.
- The automata we'll explore are
  - powerful enough to capture huge classes of computing devices, yet
  - simple enough that we can reason about them
- They're also fascinating and useful on their own

# Toward a Model of Computation...

7 8 9 ÷

4 5 6 ×

1 2 3 −

0 . = +

Why does this computer "feel" less powerful...

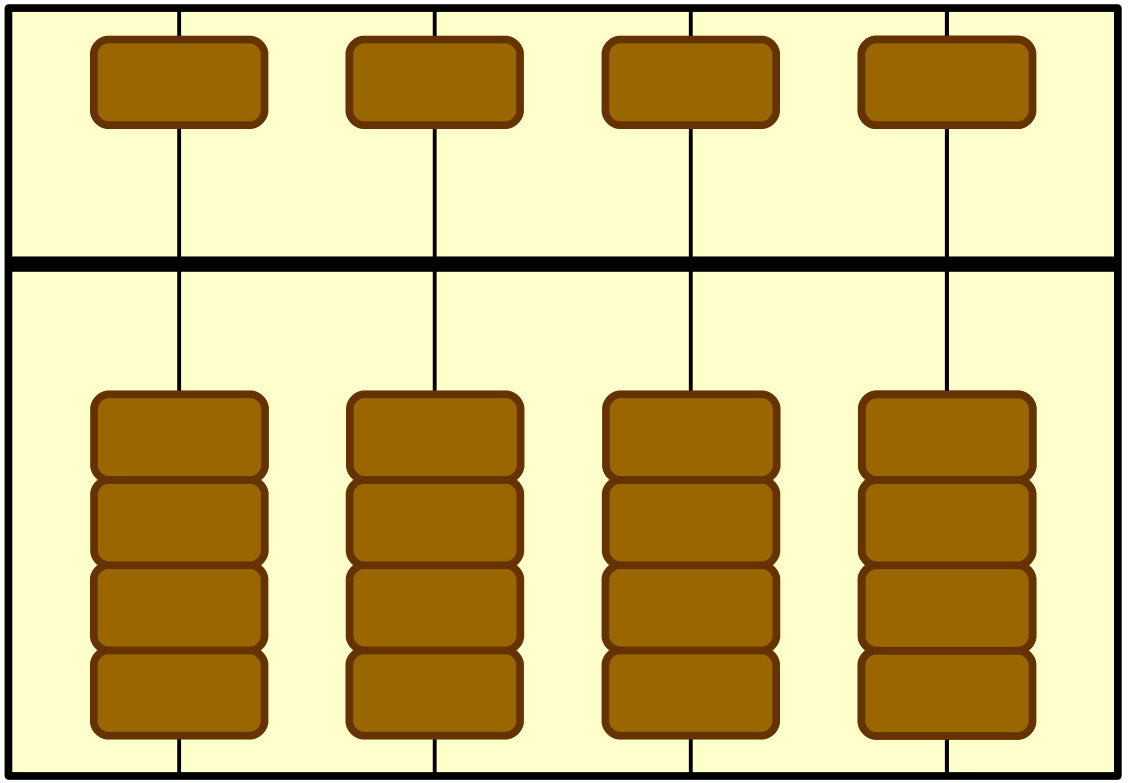
...than this one?

# Calculators vs. Desktops

- A calculator has a ***small amount of memory***. A desktop computer has a ***large amount of memory***.
- A calculator performs a ***fixed set of functions***. A desktop is ***reprogrammable*** and can run many different programs.
- These two distinctions account for much of the difference between “calculator-like” computers and “desktop-like” computers.
- We'll first explore “small-memory” computers, then discuss “large-memory” computers

# Computing with Finite, Fixed Memory

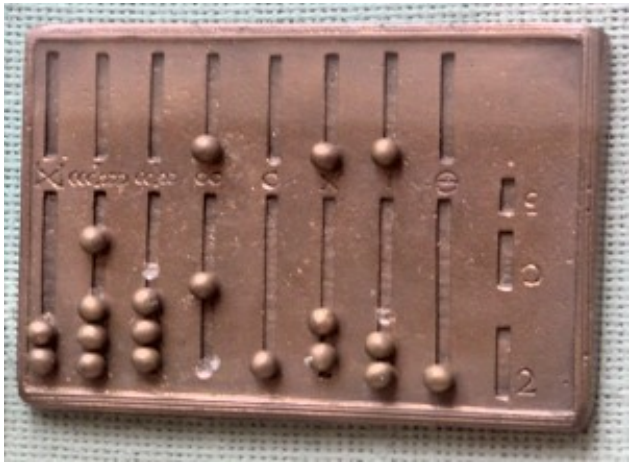
| | | | |
|---|---|---|---|
| 7 | 8 | 9 | ÷ |
| 4 | 5 | 6 | × |
| 1 | 2 | 3 | − |
| 0 | . | = | + |



Data stored electronically.
Algorithm is in silicon.
Memory limited by display.

Data stored in wood.
Algorithm is in the brain.
Memory limited by beads.

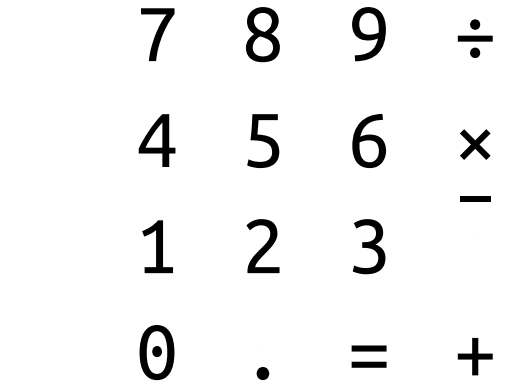
# Other Examples

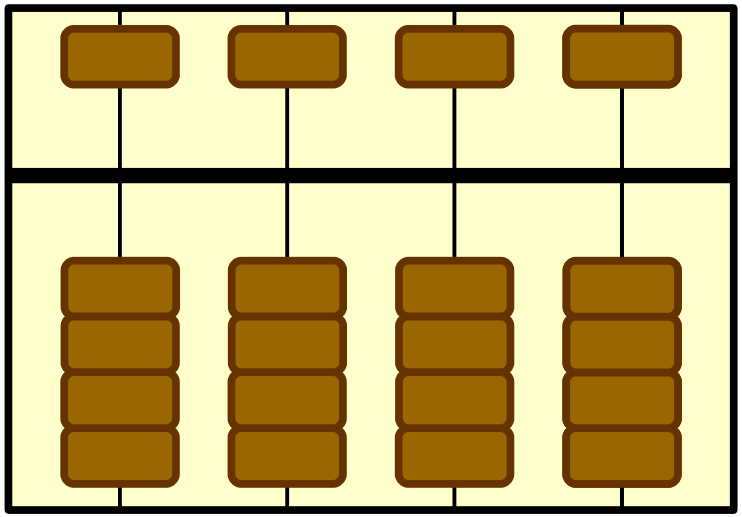


How do we model “memory” and “an algorithm” when they can take on so many forms?

# What's in Common?

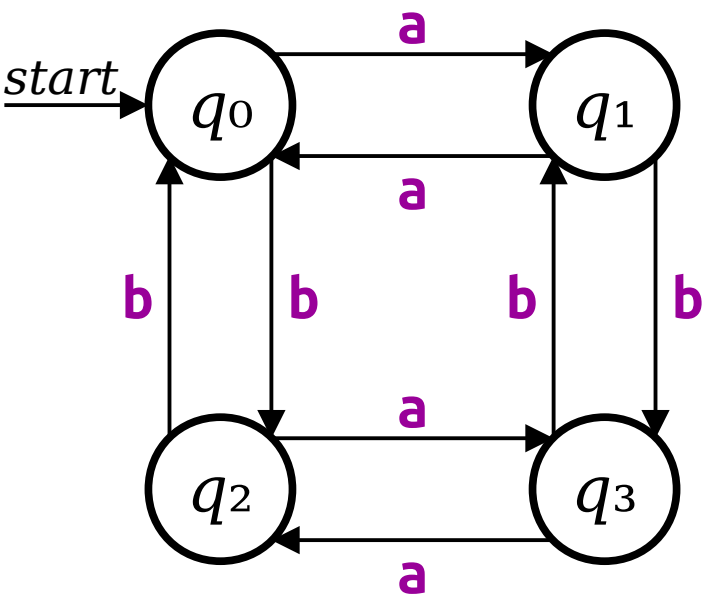
- These machines ***receive input*** from an external source.
- That input is provided ***sequentially***, one discrete unit at a time.
- Each input causes the device to ***change configuration***. This change, big or small, is where the computation happens.
- Once all input is provided, we can ***read off an answer*** based on the configuration of the device.



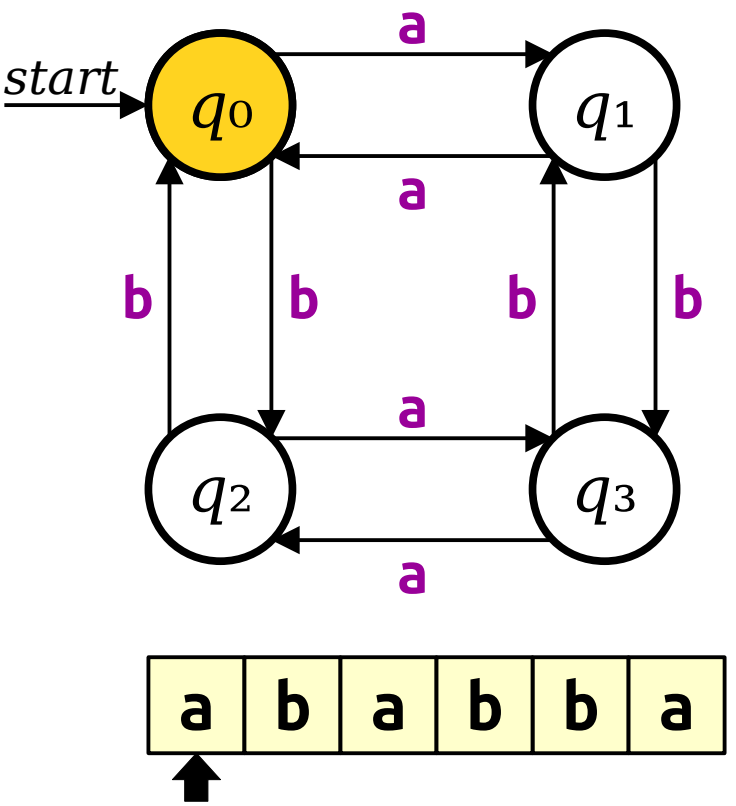
# Modeling Finite Computation

- We will model a finite-memory computer as a collection of ***states*** linked by ***transitions***.
- Each state corresponds to one possible configuration of the device's memory.
- Each transition indicates how memory changes in response to inputs.
- Some state is designated as the ***start state***. The computation begins in that state.



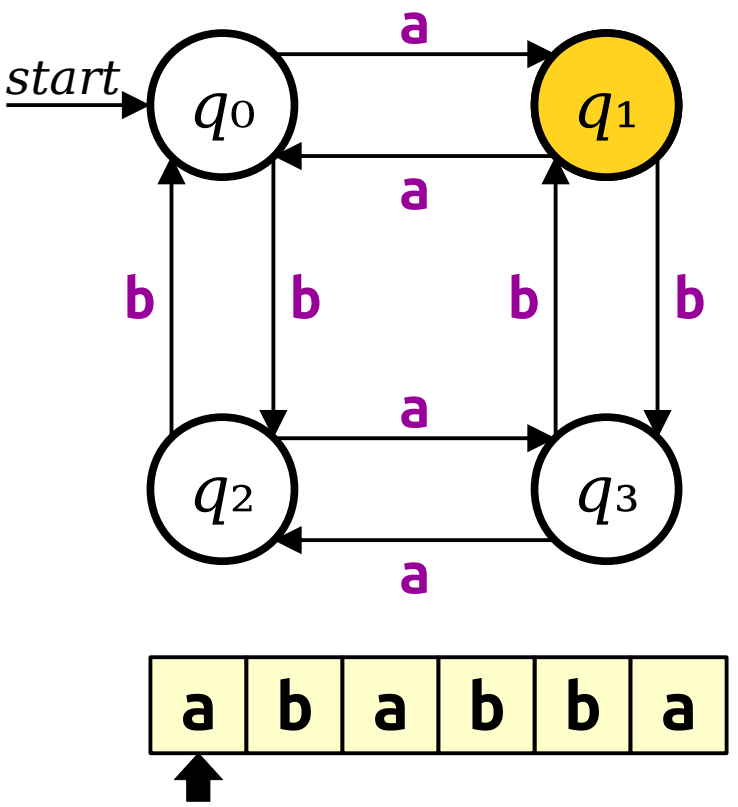
# Modeling Finite Computation

- This device processes ***strings*** of ***characters***.
  - Each character is an external input
  - The string is a sequence of inputs to the device.
- To run this device, we begin in our start state and scan the input from left to right.
- Each time the machine sees a character, it ***changes state*** by following the transition labeled with that character.



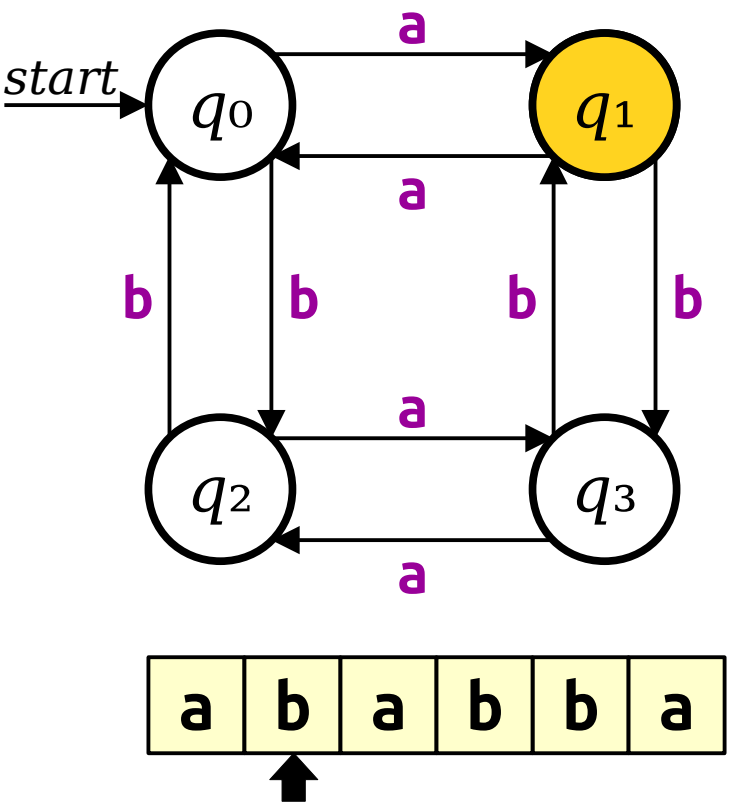
# Modeling Finite Computation

- This device processes ***strings*** of ***characters***.
  - Each character is an external input
  - The string is a sequence of inputs to the device.
- To run this device, we begin in our start state and scan the input from left to right.
- Each time the machine sees a character, it ***changes state*** by following the transition labeled with that character.



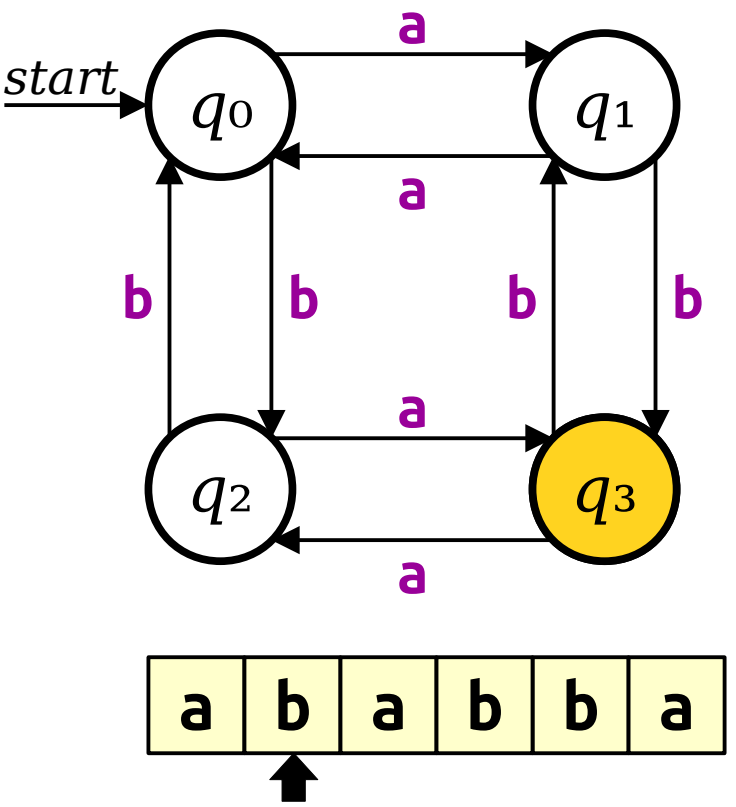
# Modeling Finite Computation

- This device processes ***strings*** of ***characters***.
  - Each character is an external input
  - The string is a sequence of inputs to the device.
- To run this device, we begin in our start state and scan the input from left to right.
- Each time the machine sees a character, it ***changes state*** by following the transition labeled with that character.



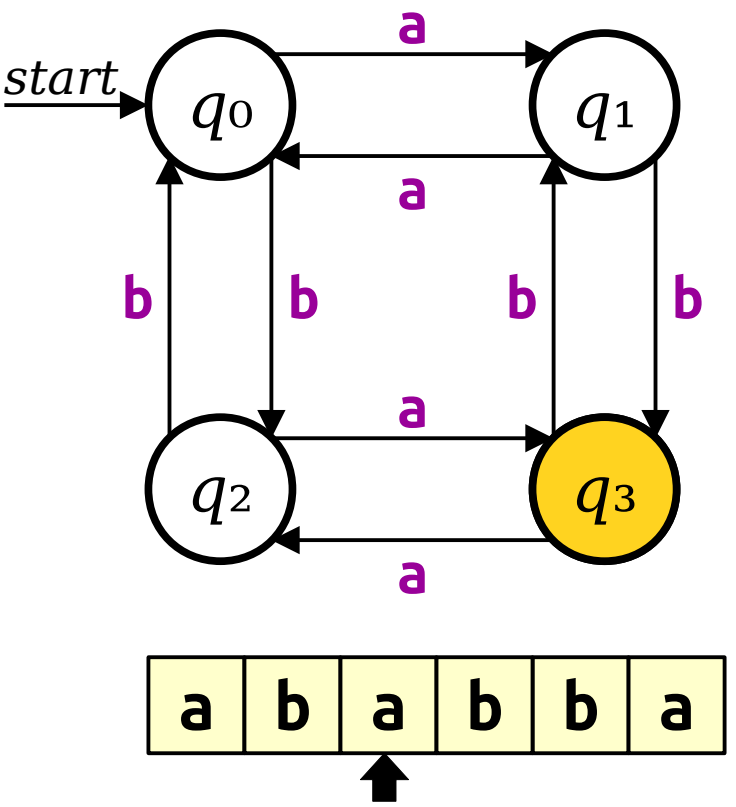
# Modeling Finite Computation

- This device processes ***strings*** of ***characters***.
  - Each character is an external input
  - The string is a sequence of inputs to the device.
- To run this device, we begin in our start state and scan the input from left to right.
- Each time the machine sees a character, it ***changes state*** by following the transition labeled with that character.



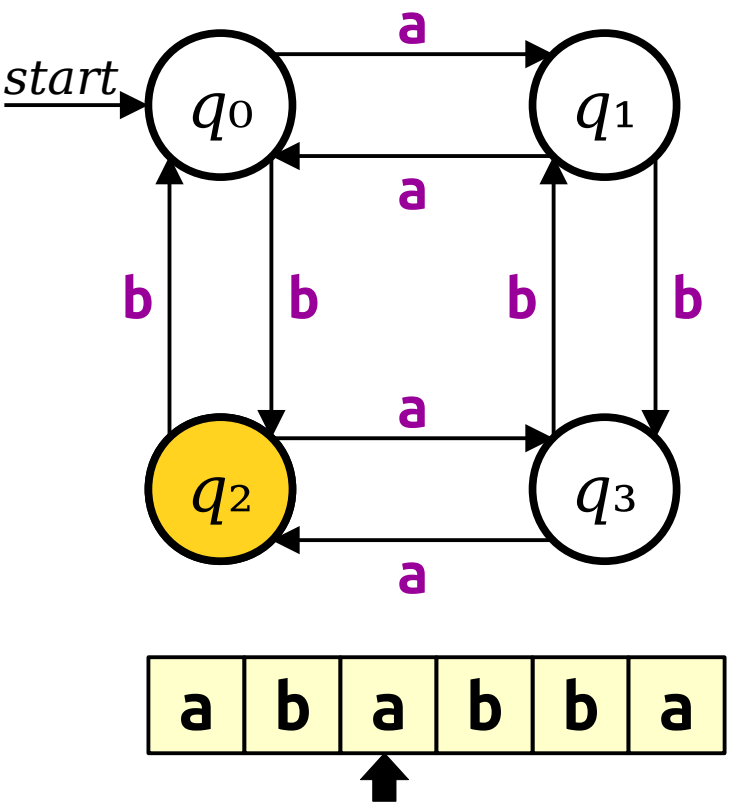
# Modeling Finite Computation

- This device processes ***strings*** of ***characters***.
  - Each character is an external input
  - The string is a sequence of inputs to the device.
- To run this device, we begin in our start state and scan the input from left to right.
- Each time the machine sees a character, it ***changes state*** by following the transition labeled with that character.



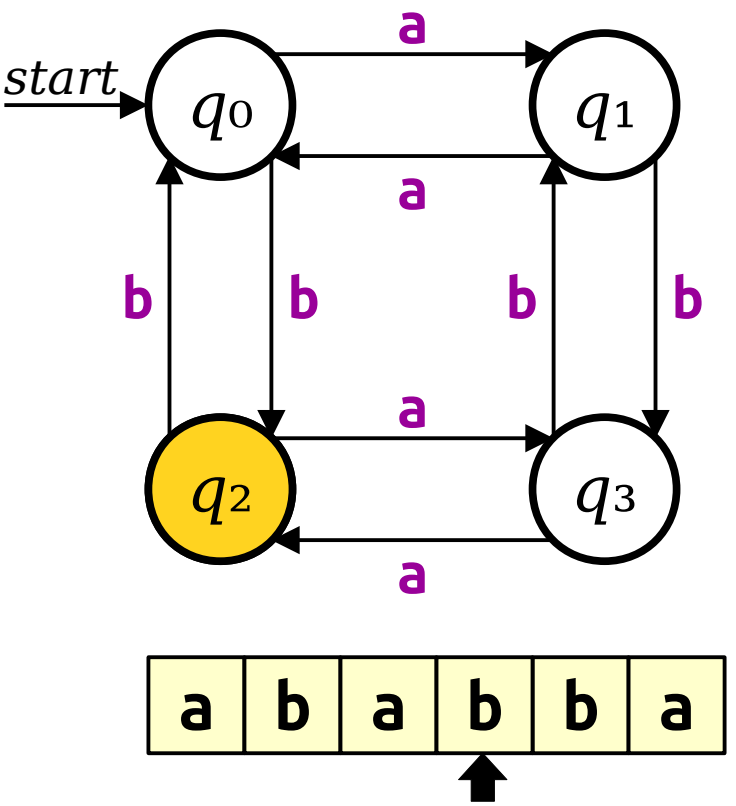
# Modeling Finite Computation

- This device processes ***strings*** of ***characters***.
  - Each character is an external input
  - The string is a sequence of inputs to the device.
- To run this device, we begin in our start state and scan the input from left to right.
- Each time the machine sees a character, it ***changes state*** by following the transition labeled with that character.



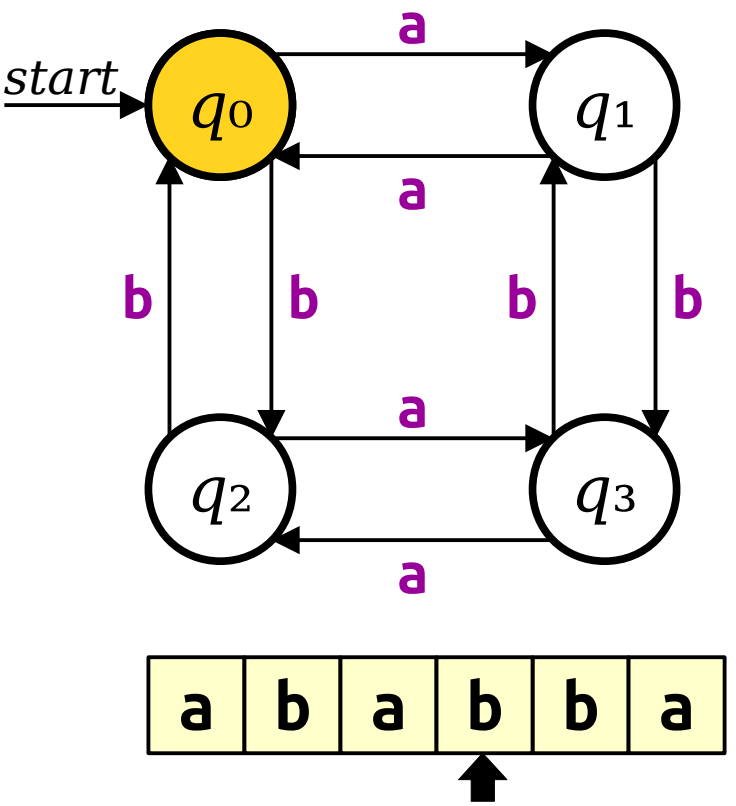
# Modeling Finite Computation

- This device processes ***strings*** of ***characters***.
  - Each character is an external input
  - The string is a sequence of inputs to the device.
- To run this device, we begin in our start state and scan the input from left to right.
- Each time the machine sees a character, it ***changes state*** by following the transition labeled with that character.



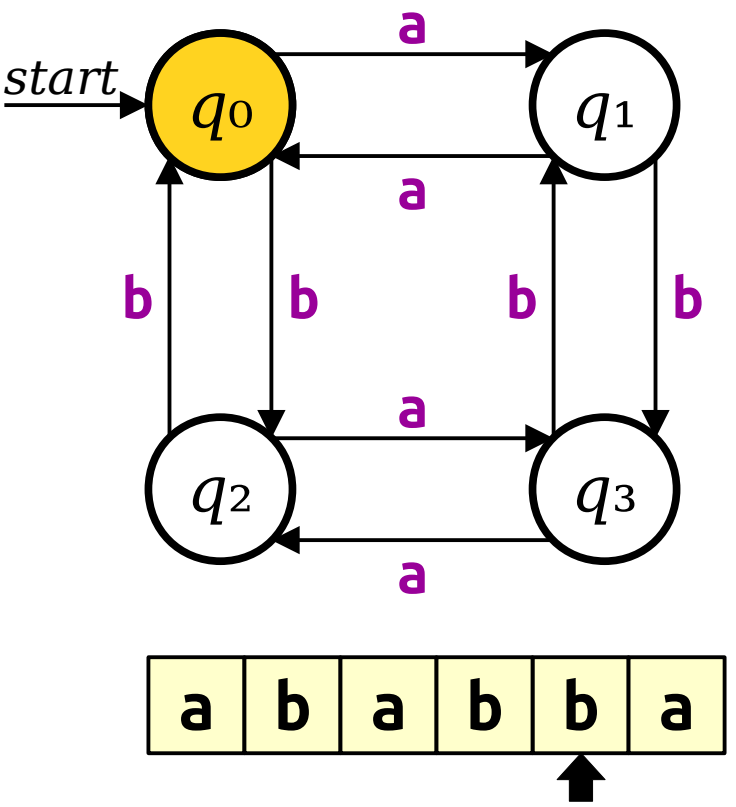
# Modeling Finite Computation

- This device processes ***strings*** of ***characters***.
  - Each character is an external input
  - The string is a sequence of inputs to the device.
- To run this device, we begin in our start state and scan the input from left to right.
- Each time the machine sees a character, it ***changes state*** by following the transition labeled with that character.



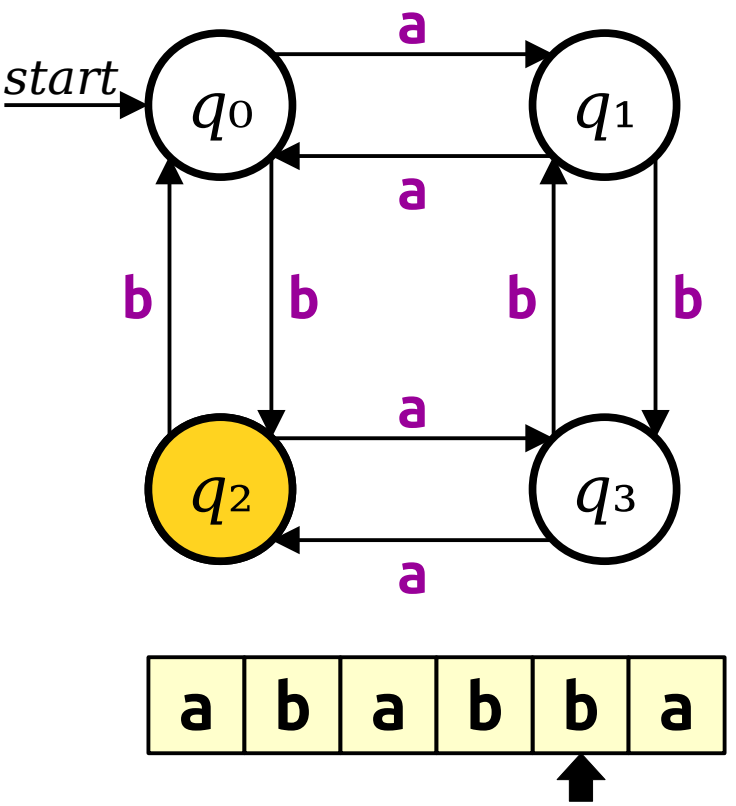
# Modeling Finite Computation

- This device processes ***strings*** of ***characters***.
  - Each character is an external input
  - The string is a sequence of inputs to the device.
- To run this device, we begin in our start state and scan the input from left to right.
- Each time the machine sees a character, it ***changes state*** by following the transition labeled with that character.



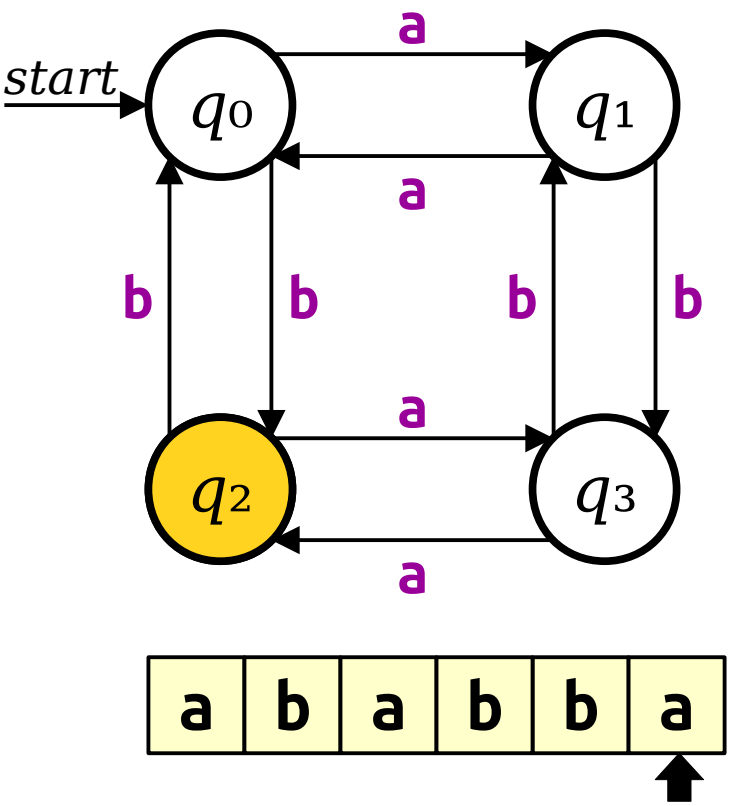
# Modeling Finite Computation

- This device processes ***strings*** of ***characters***.
  - Each character is an external input
  - The string is a sequence of inputs to the device.
- To run this device, we begin in our start state and scan the input from left to right.
- Each time the machine sees a character, it ***changes state*** by following the transition labeled with that character.



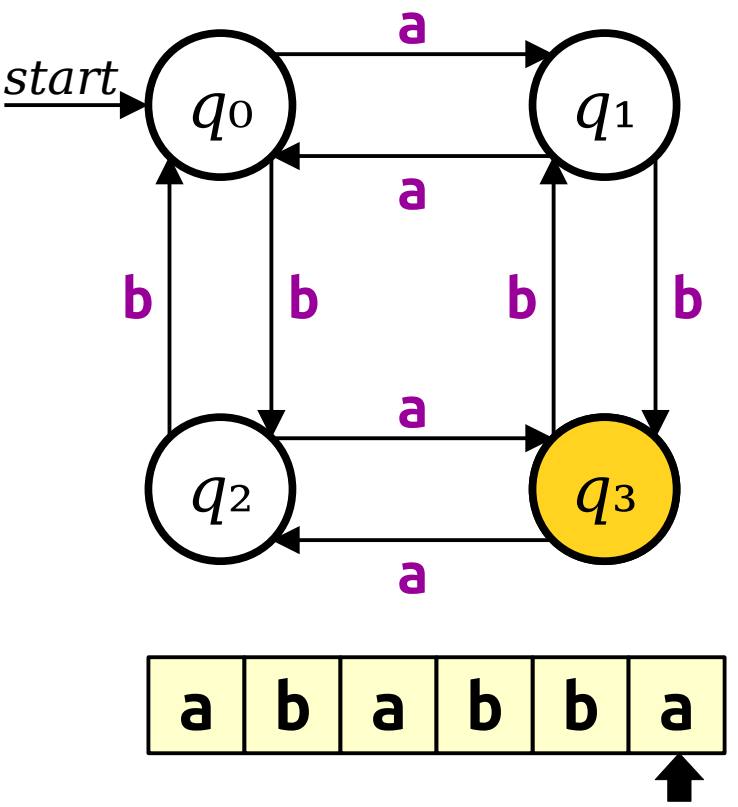
# Modeling Finite Computation

- This device processes ***strings*** of ***characters***.
  - Each character is an external input
  - The string is a sequence of inputs to the device.
- To run this device, we begin in our start state and scan the input from left to right.
- Each time the machine sees a character, it ***changes state*** by following the transition labeled with that character.



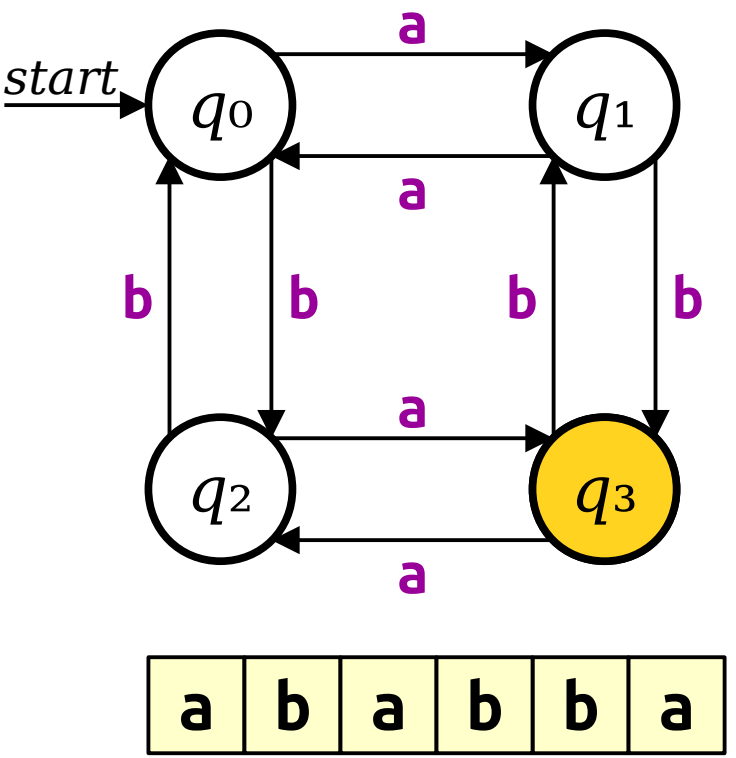
# Modeling Finite Computation

- This device processes ***strings*** of ***characters***.
  - Each character is an external input
  - The string is a sequence of inputs to the device.
- To run this device, we begin in our start state and scan the input from left to right.
- Each time the machine sees a character, it ***changes state*** by following the transition labeled with that character.



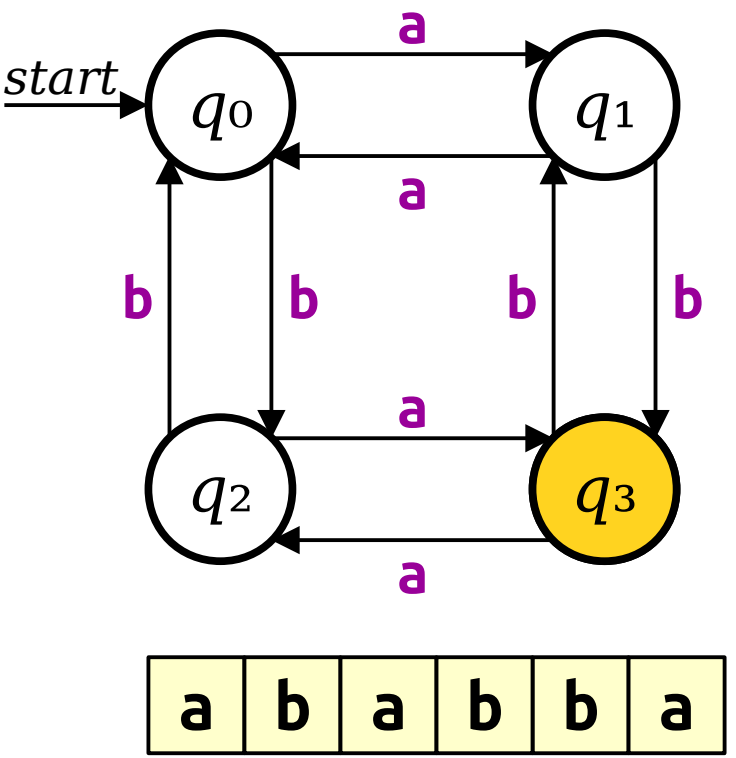
# Modeling Finite Computation

- This device processes ***strings*** of ***characters***.
  - Each character is an external input
  - The string is a sequence of inputs to the device.
- To run this device, we begin in our start state and scan the input from left to right.
- Each time the machine sees a character, it ***changes state*** by following the transition labeled with that character.



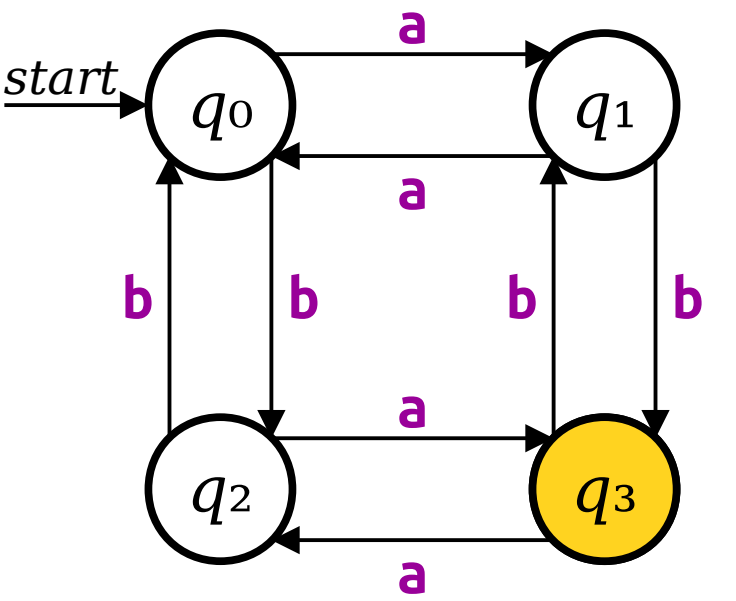
# Modeling Finite Computation

- This device processes ***strings*** of ***characters***.
  - Each character is an external input
  - The string is a sequence of inputs to the device.
- To run this device, we begin in our start state and scan the input from left to right.
- Each time the machine sees a character, it ***changes state*** by following the transition labeled with that character.



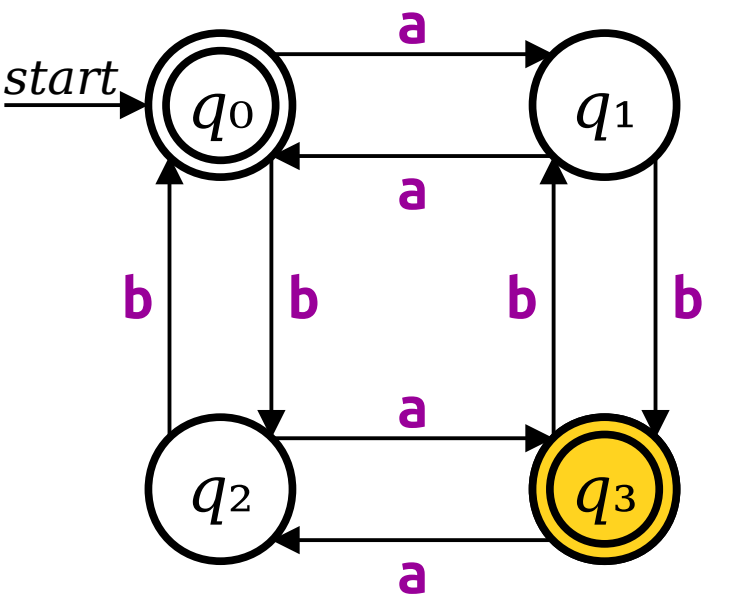
# Modeling Finite Computation

- Some of the states in our computational device will be marked as ***accepting states***. These are denoted with a double ring.
- If the device ends in an accepting state after seeing all the input, ***accepts*** the input (says `YES`)
- If the device does not end in an accepting state after seeing all the input, it ***rejects*** the input (says `NO`).



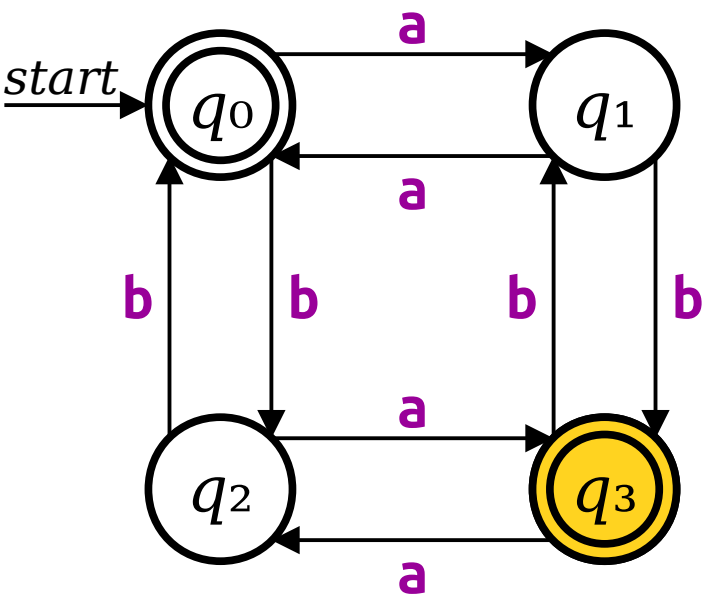
# Modeling Finite Computation

- Some of the states in our computational device will be marked as ***accepting states***. These are denoted with a double ring.
- If the device ends in an accepting state after seeing all the input, ***accepts*** the input (says `YES`)
- If the device does not end in an accepting state after seeing all the input, it ***rejects*** the input (says `NO`).



# Modeling Finite Computation

- Some of the states in our computational device will be marked as ***accepting states***. These are denoted with a double ring.
- If the device ends in an accepting state after seeing all the input, ***accepts*** the input (says `YES`)
- If the device does not end in an accepting state after seeing all the input, it ***rejects*** the input (says `NO`).



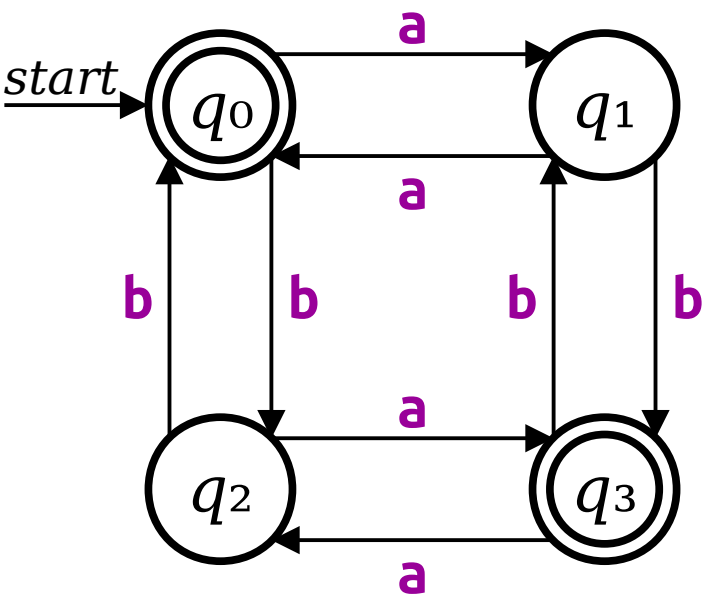

# Modeling Finite Computation

- Try it yourself! Which of these strings does this device accept?

**aab**

**aabb**

**abbababba**




Go to
PollEv.com/cs103spr25

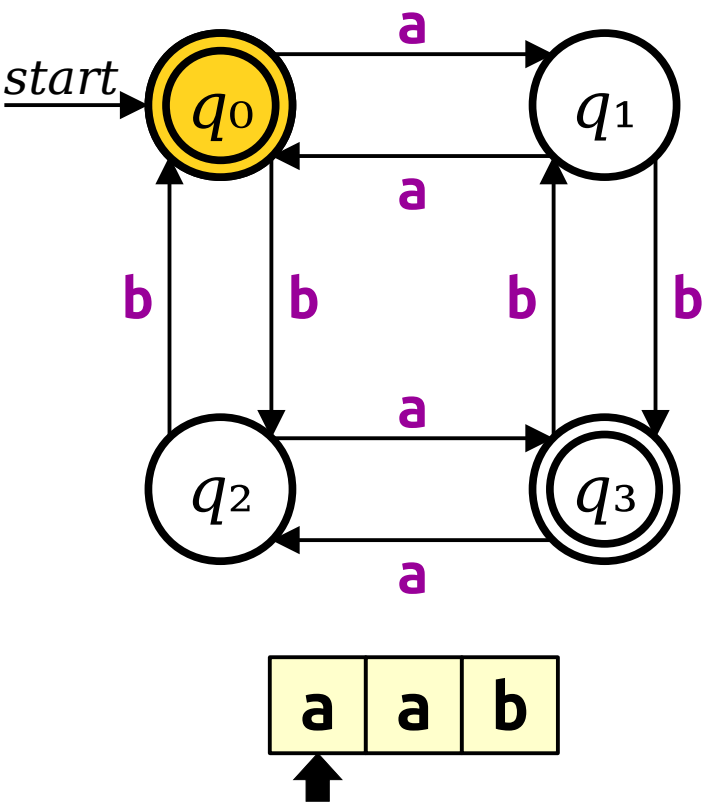
# Modeling Finite Computation

- Try it yourself! Which of these strings does this device accept?

**aab**

**aabb**

**abbababba**



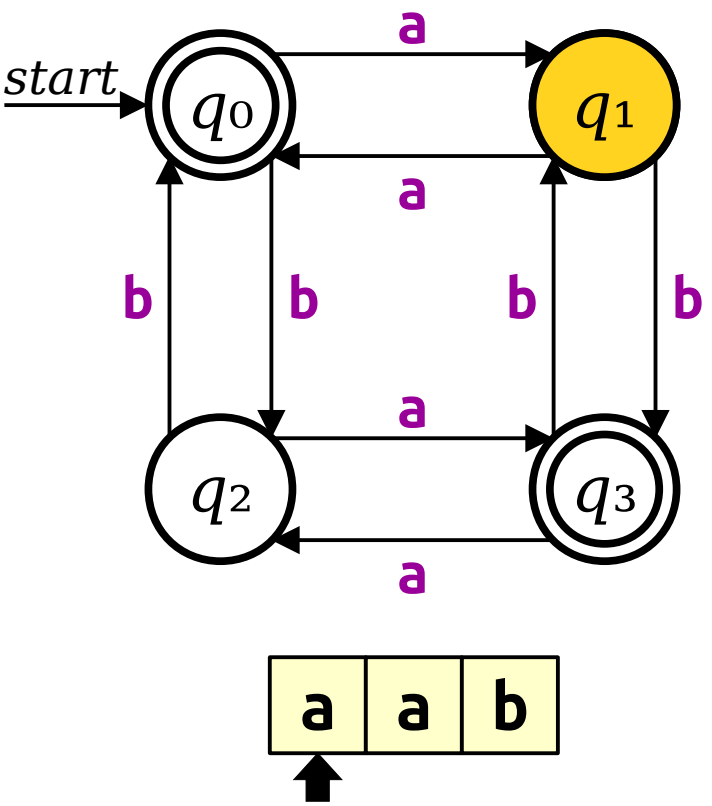
# Modeling Finite Computation

- Try it yourself! Which of these strings does this device accept?

**aab**

**aabb**

**abbababba**



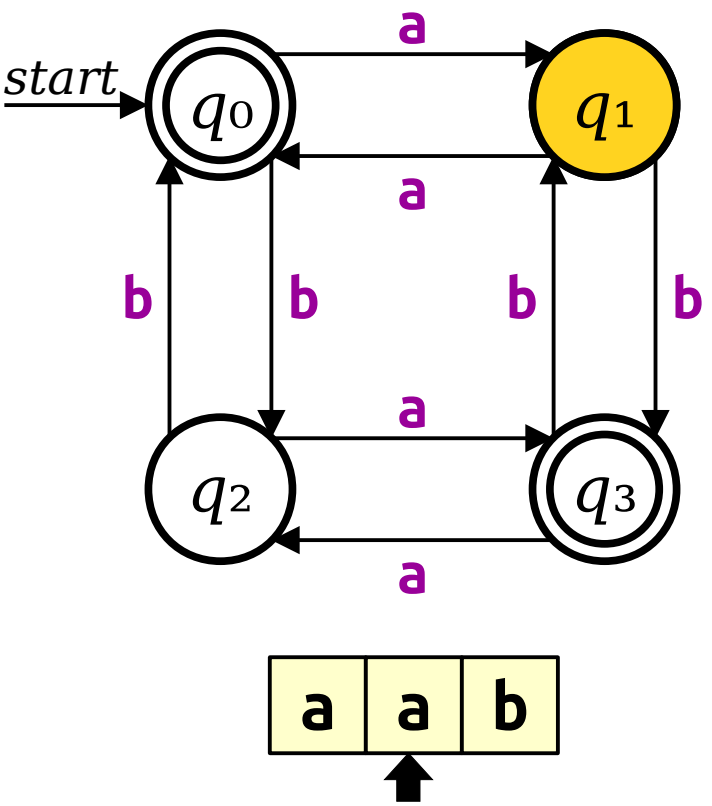
# Modeling Finite Computation

- Try it yourself! Which of these strings does this device accept?

**aab**

**aabb**

**abbababba**



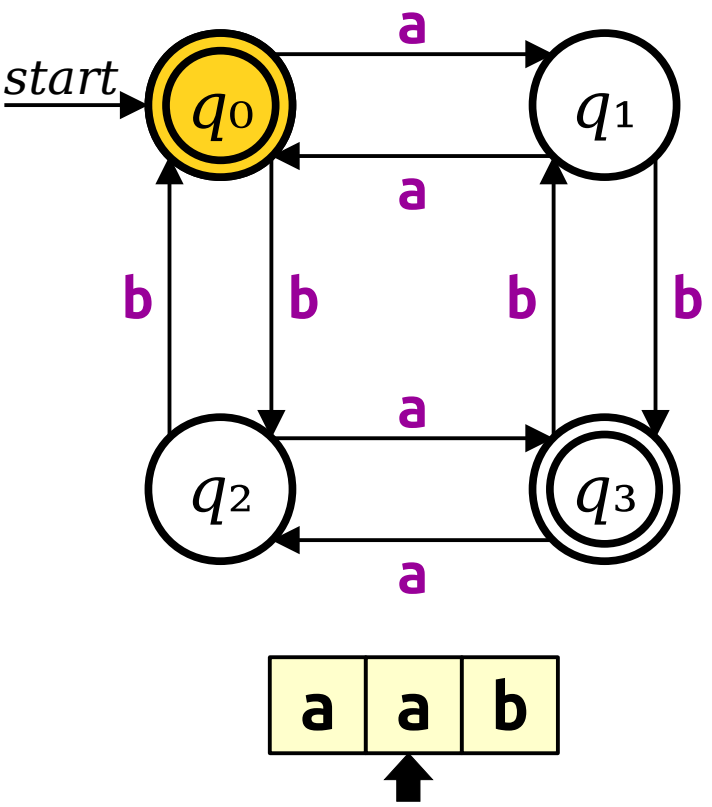
# Modeling Finite Computation

- Try it yourself! Which of these strings does this device accept?

**aab**

**aabb**

**abbababba**



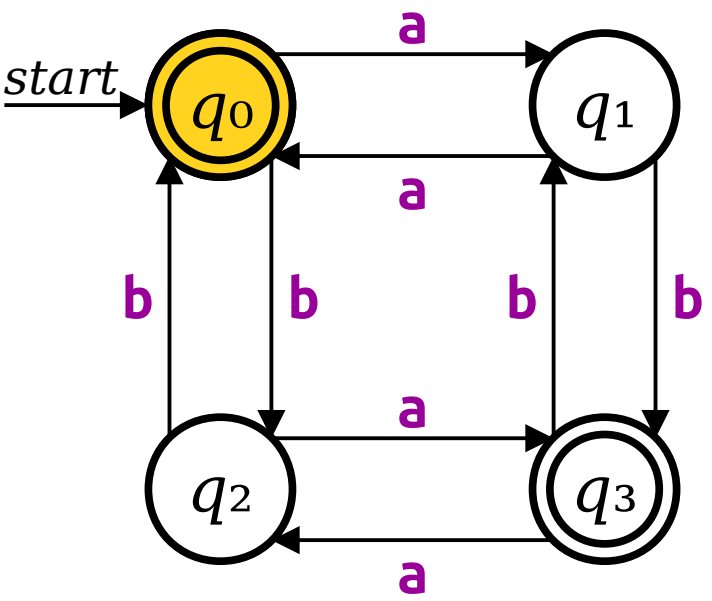
# Modeling Finite Computation

- Try it yourself! Which of these strings does this device accept?

**aab**

**aabb**

**abbababba**



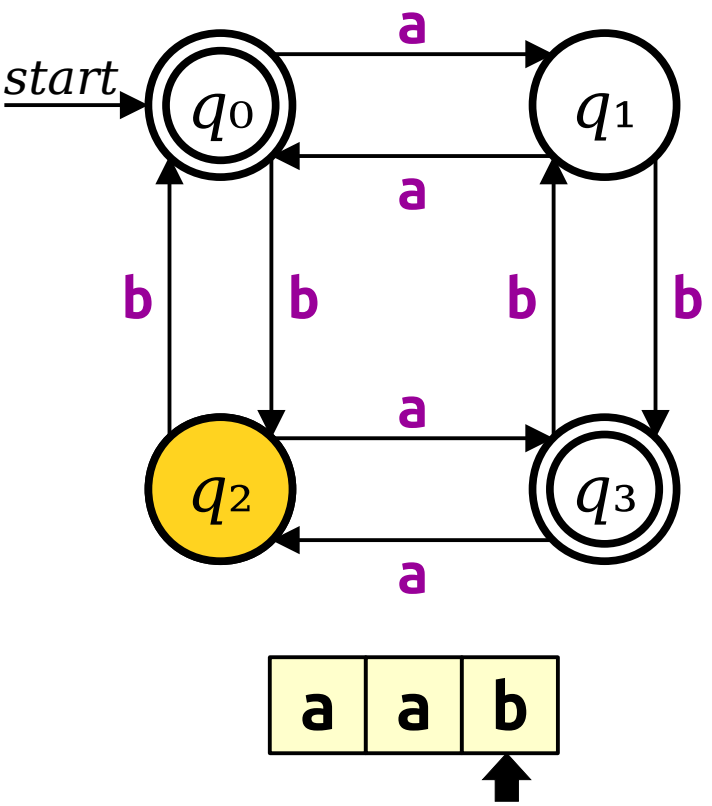
# Modeling Finite Computation

- Try it yourself! Which of these strings does this device accept?

**aab**

**aabb**

**abbababba**



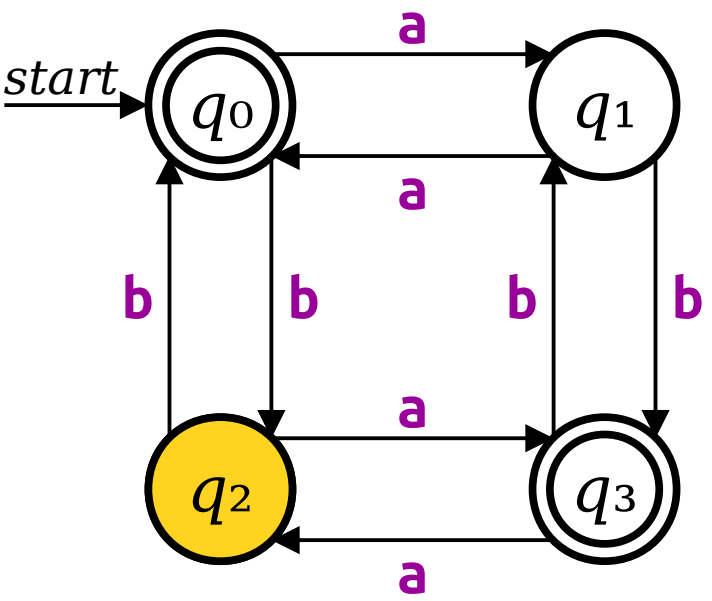
# Modeling Finite Computation

- Try it yourself! Which of these strings does this device accept?

**aab**

**aabb**

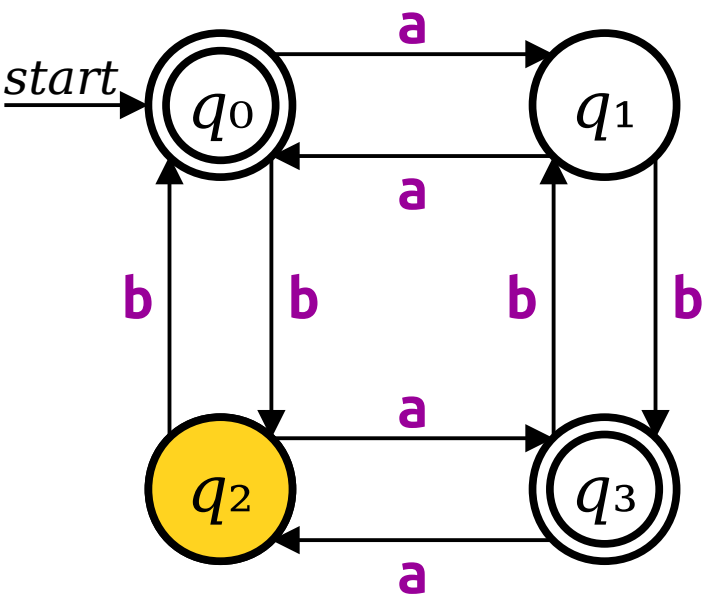
**abbababba**



# Modeling Finite Computation

- Try it yourself! Which of these strings does this device accept?

**aab**




| a | a | b |
|---|---|---|

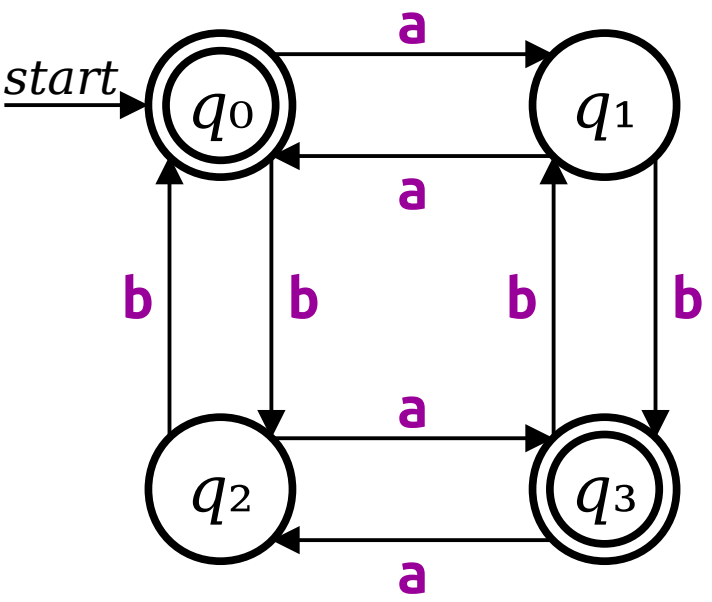
# Modeling Finite Computation

- Try it yourself! Which of these strings does this device accept?

**aab**

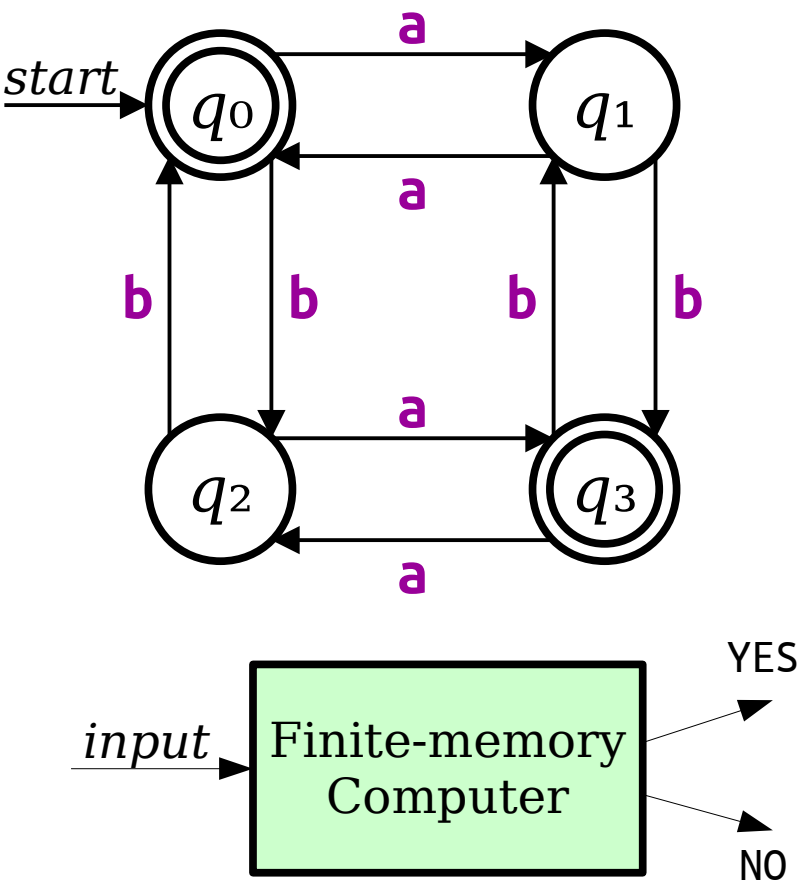
**aabb**

**abbababba**



# Finite Automata

- This device is a ***finite automaton*** (plural: ***finite automata***).
- Finite automata model computers where (1) memory is finite and (2) the computation produces a YES/NO answer.
- In other words, finite automata model predicates, and do so with a fixed, finite amount of memory.



# Formalization

# 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: Let Σ = {`a`, `b`}. Here are some strings over Σ:

  `a` `aabaaabbabaaabaaaabbb` `abbababba`

- But wait! There are no quotes here!
- The ***empty string*** has no characters and is denoted **ε**.

# Languages

- A ***language over Σ*** 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: Σ* = { *w* | *w* is a string over Σ }.
- Formally, we say that *L* is a language over Σ when *L* ⊆ Σ*.

# Mathematical Lookalikes

- We now have ∈, ε, Σ, and Σ*. Yikes!
- The symbol ∈ is the… ?

# Mathematical Lookalikes

- We now have ∈, ε, Σ, and Σ*. Yikes!
- The symbol ∈ is the ***element-of*** relation.

# Mathematical Lookalikes

- We now have ∈, ε, Σ, and Σ*. Yikes!
- The symbol ∈ is the ***element-of*** relation.
- The symbol ε is the… ?

# Mathematical Lookalikes

- We now have ∈, ε, Σ, and Σ*. Yikes!
- The symbol ∈ is the ***element-of*** relation.
- The symbol ε is the ***empty string***.

# Mathematical Lookalikes

- We now have ∈, ε, Σ, and Σ*. Yikes!
- The symbol ∈ is the ***element-of*** relation.
- The symbol ε is the ***empty string***.
- The symbol Σ denotes… ?

# Mathematical Lookalikes

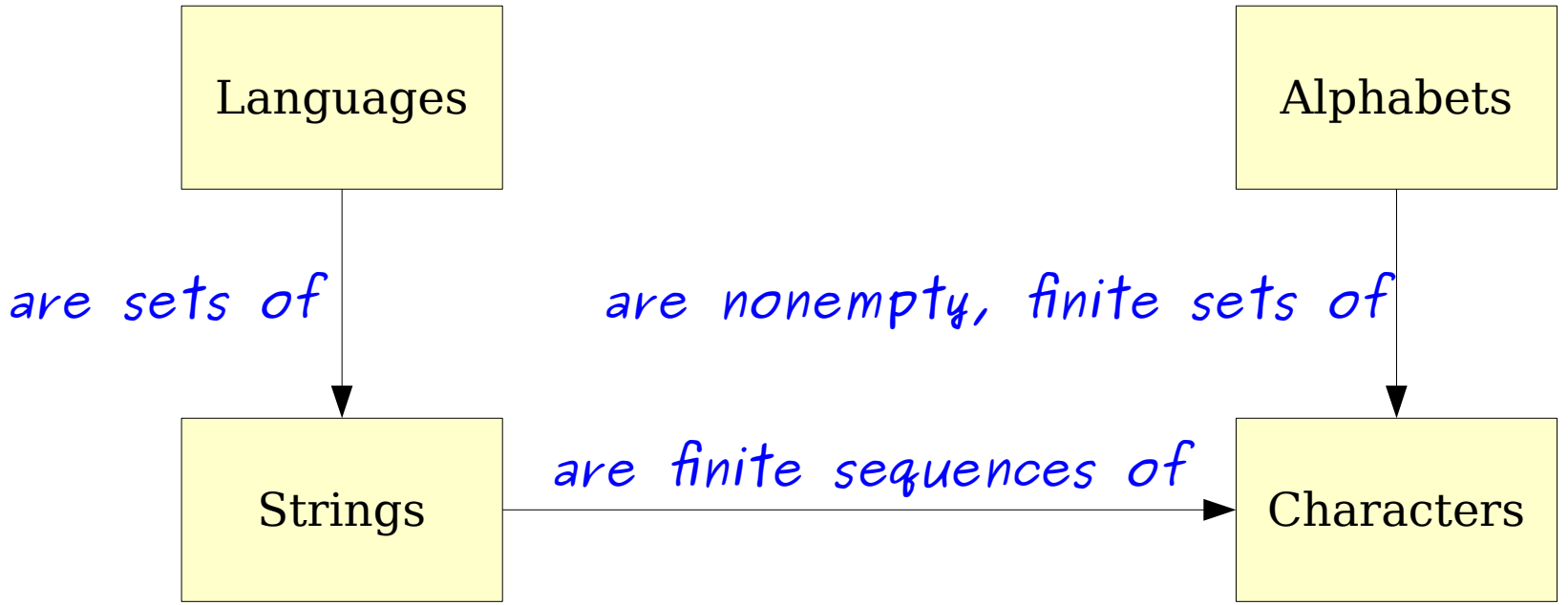
- We now have ∈, ε, Σ, and Σ*. Yikes!
- The symbol ∈ is the ***element-of*** relation.
- The symbol ε is the ***empty string***.
- The symbol Σ denotes an ***alphabet***.

# Mathematical Lookalikes

- We now have ∈, ε, Σ, and Σ*. Yikes!
- The symbol ∈ is the ***element-of*** relation.
- The symbol ε is the ***empty string***.
- The symbol Σ denotes an ***alphabet***.
- The expression Σ* means “all strings that can be made from characters in Σ.”
- That lets us write things like
    - We have ε ∈ Σ*, but ε ∉ Σ.
- Ever get confused? ***Just ask!***

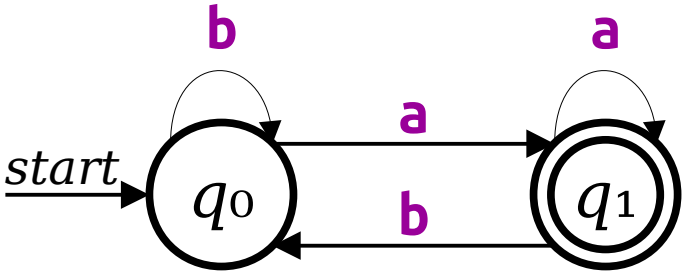
# The Cast of Characters

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



# Finite Automata and Languages

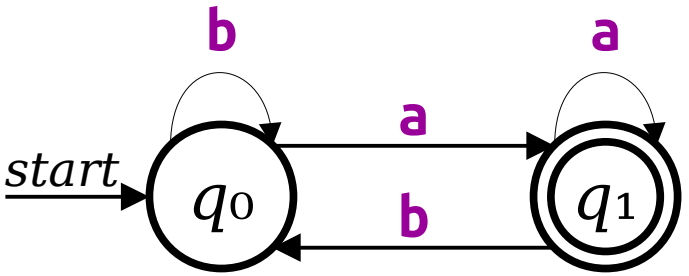
- Let *A* be an automaton that processes strings drawn from an alphabet Σ.
- The ***language of A***, denoted **Ǹ(*A*)**, is the set of strings over Σ that *A* accepts:




$$Ǹ(A) = \{ w \in \Sigma^* \mid A \text{ accepts } w \}$$

# Finite Automata and Languages

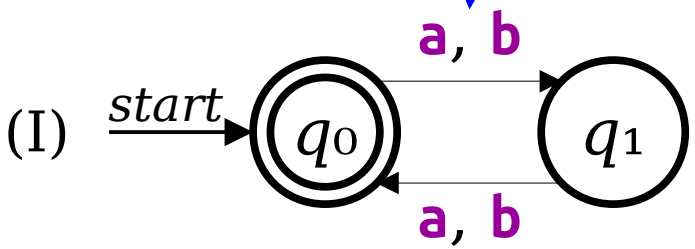
- Let $D$ be the automaton shown to the right. It processes strings over {**a**, **b**}.
- Notice that $D$ accepts all strings of **a**'s and **b**'s that end in **a** and rejects everything else.

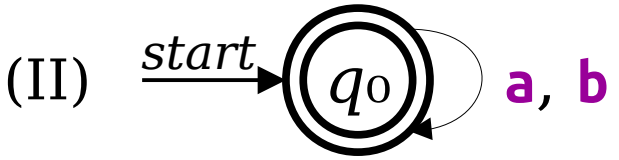


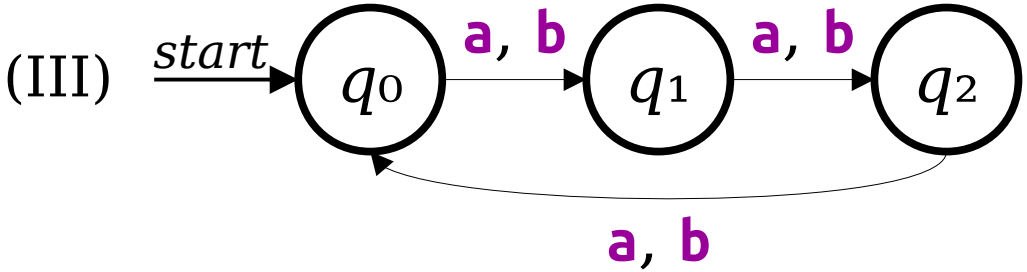

- 
- So $\grave{N}(D) = \{ w \in \{a, b\}^* \mid w \text{ ends in } a \}$.

**$\grave{N}(A) = \{ w \in \Sigma^* \mid A \text{ accepts } w \}$**

This means “take this transition if you see an **a** or a **b**.”




Go to PollEv.com/cs103spr25

$$\grave{N}(A) = \{ w \in \Sigma^* \mid A \text{ accepts } w \}$$

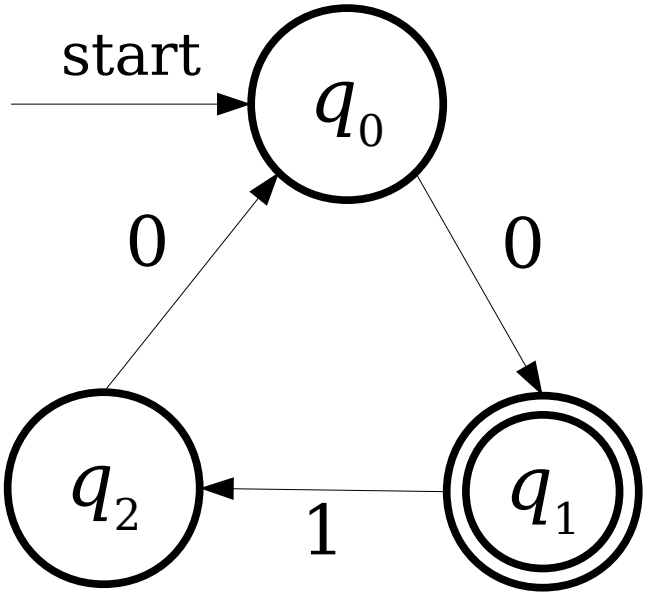
# The Story So Far

- A ***finite automaton*** is a collection of ***states*** joined by ***transitions***.
- Some state is designated as the ***start state***.
- Some number of 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.
- The ***language*** of an automaton is the… ?

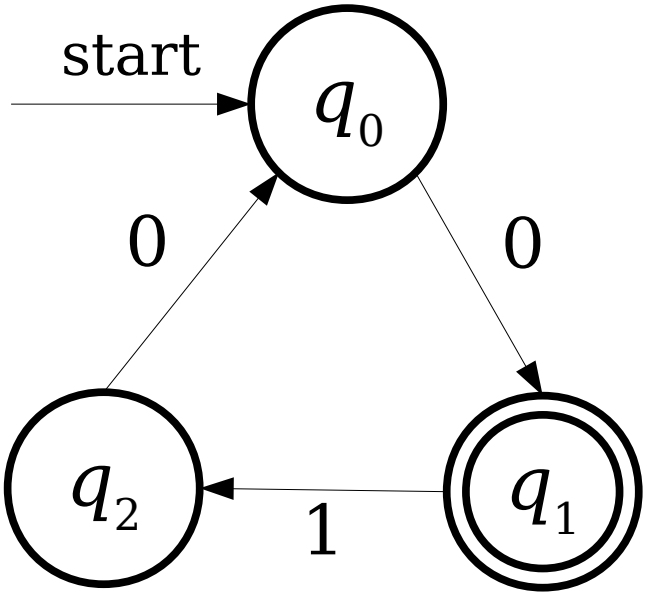
# The Story So Far

- A ***finite automaton*** is a collection of ***states*** joined by ***transitions***.
- Some state is designated as the ***start state***.
- Some number of 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.
- The ***language*** of an automaton is the set of strings it accepts.

# A Small Problem

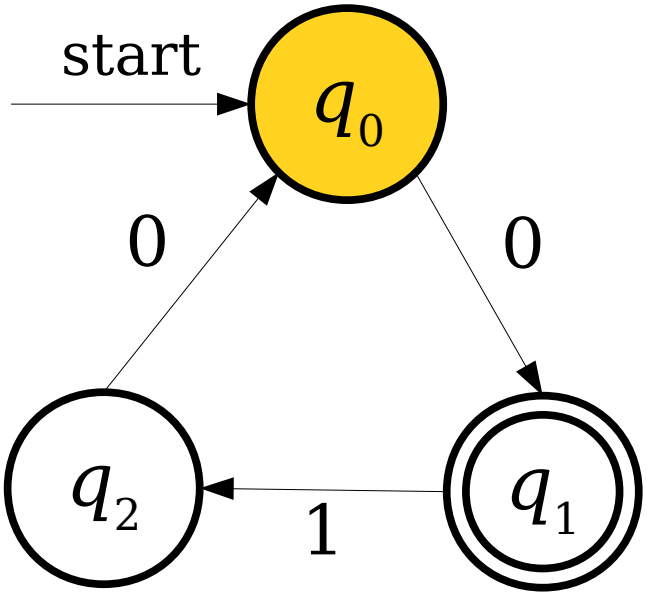


# A Small Problem



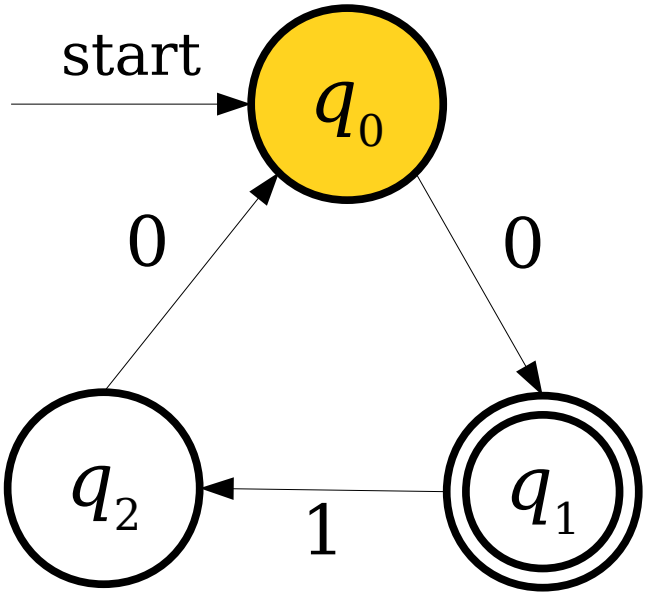

| | | | |
|---|---|---|---|
| **0** | **1** | **1** | **0** |

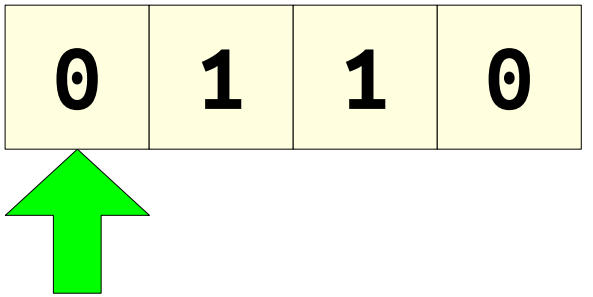
# A Small Problem



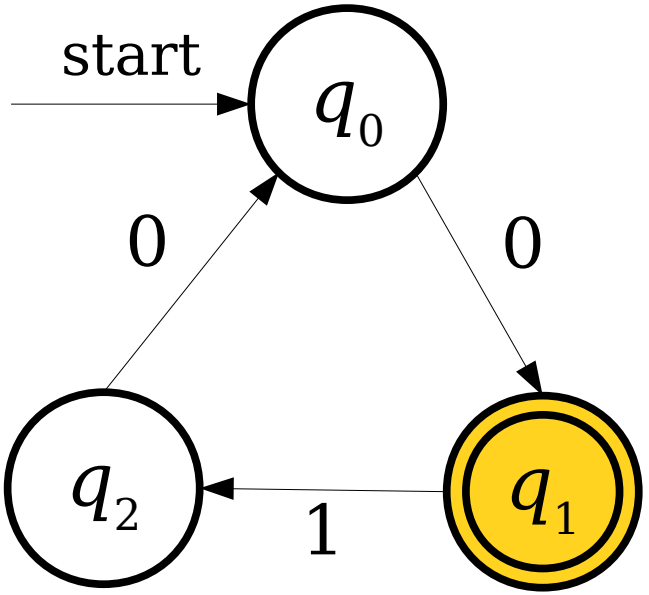

| | | | |
|---|---|---|---|
| **0** | **1** | **1** | **0** |

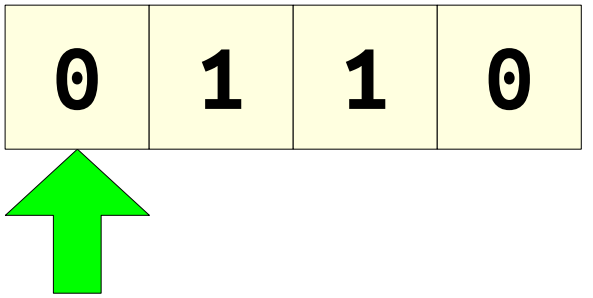
# A Small Problem



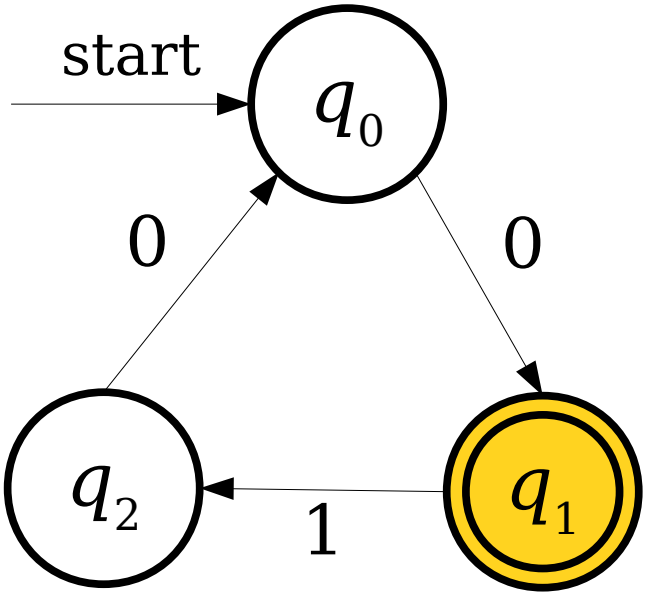



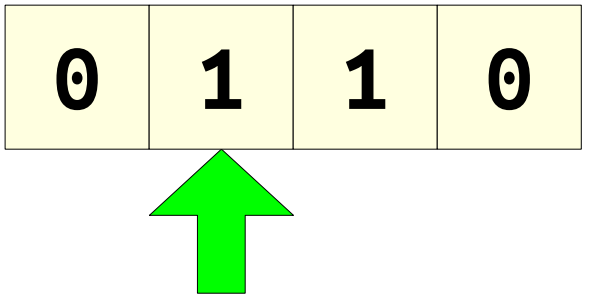
# A Small Problem



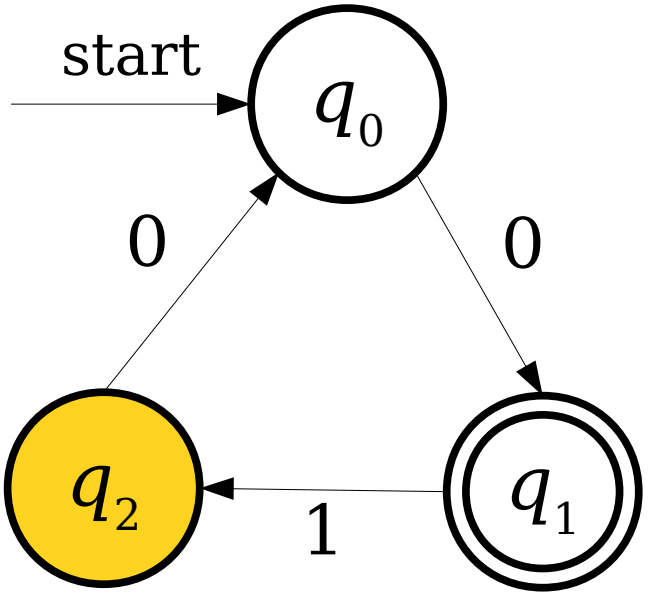



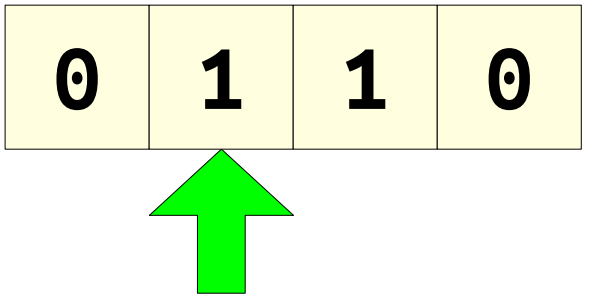
# A Small Problem



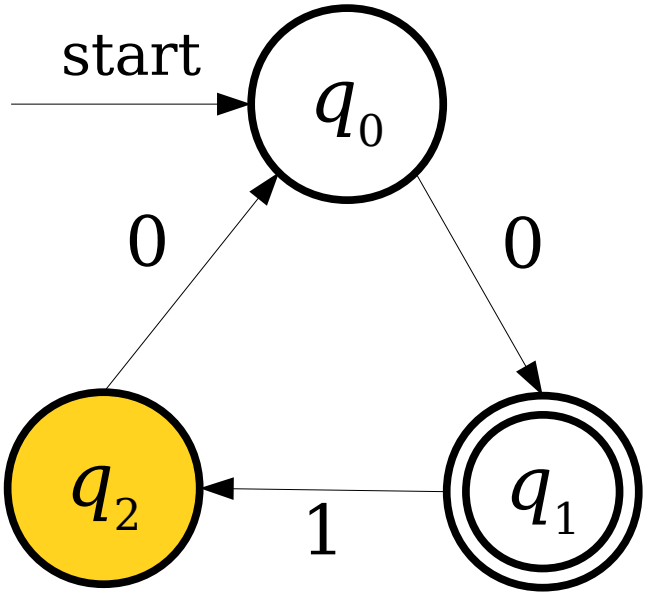



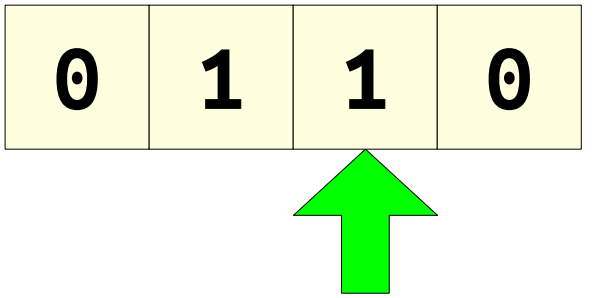
# A Small Problem



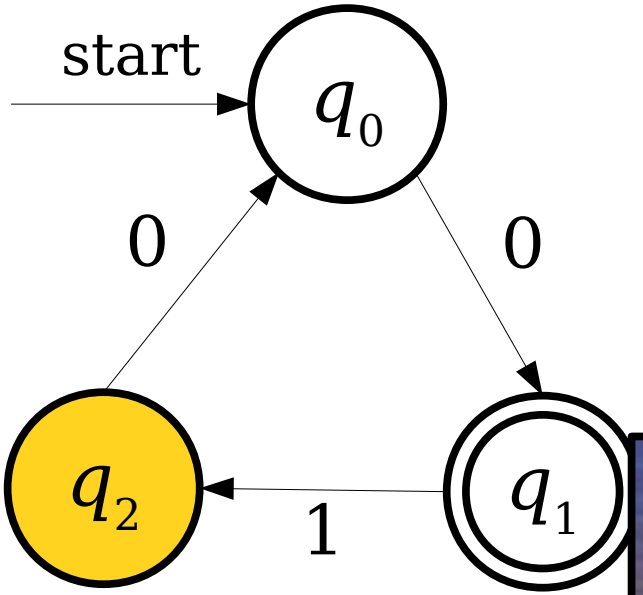



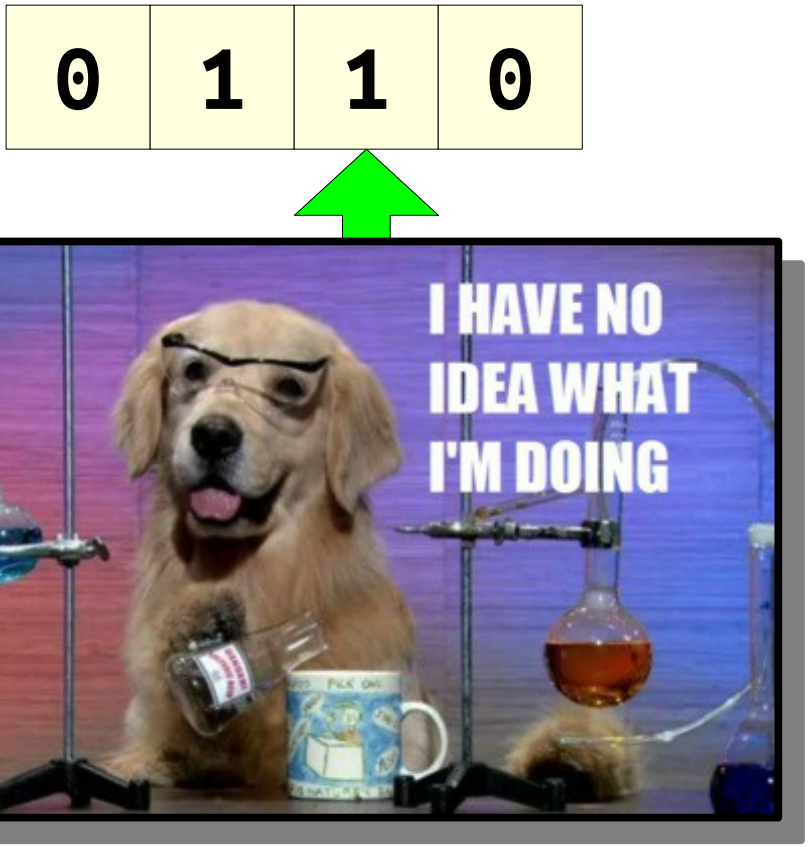
# A Small Problem



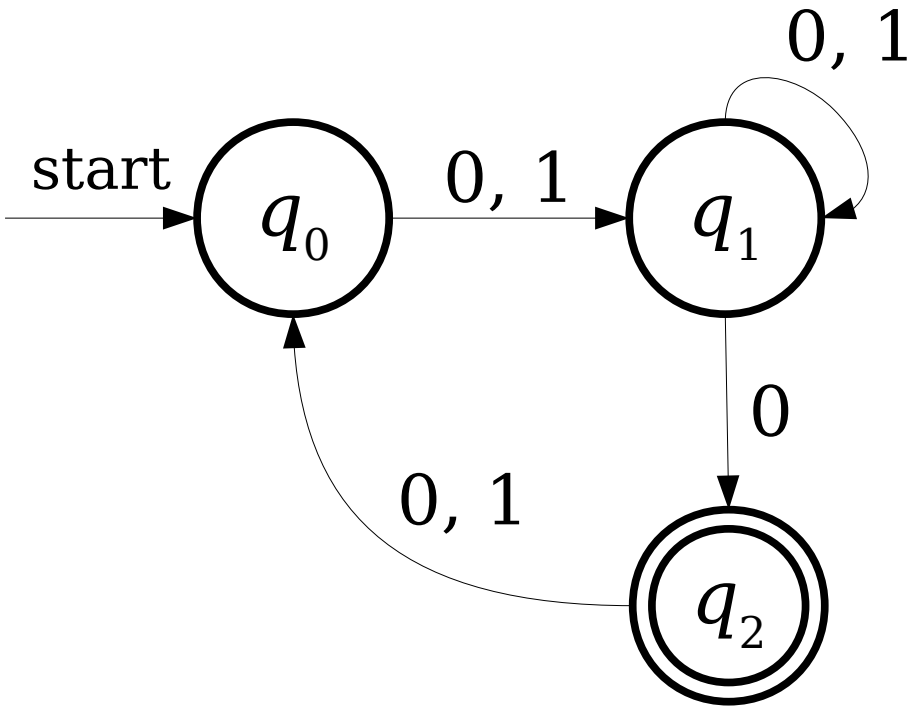



# A Small Problem

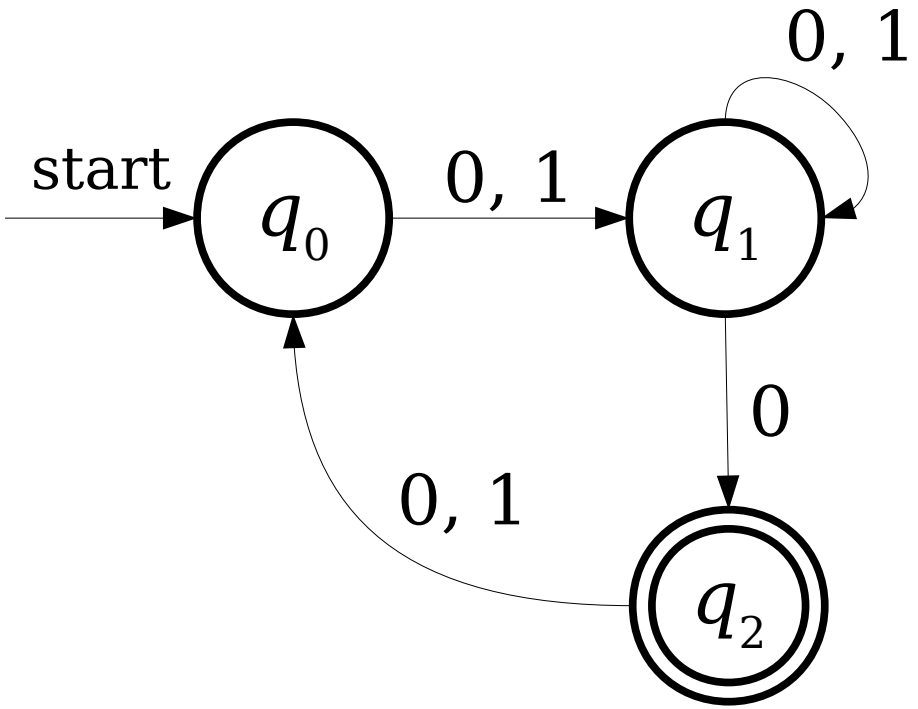


# Another Small Problem

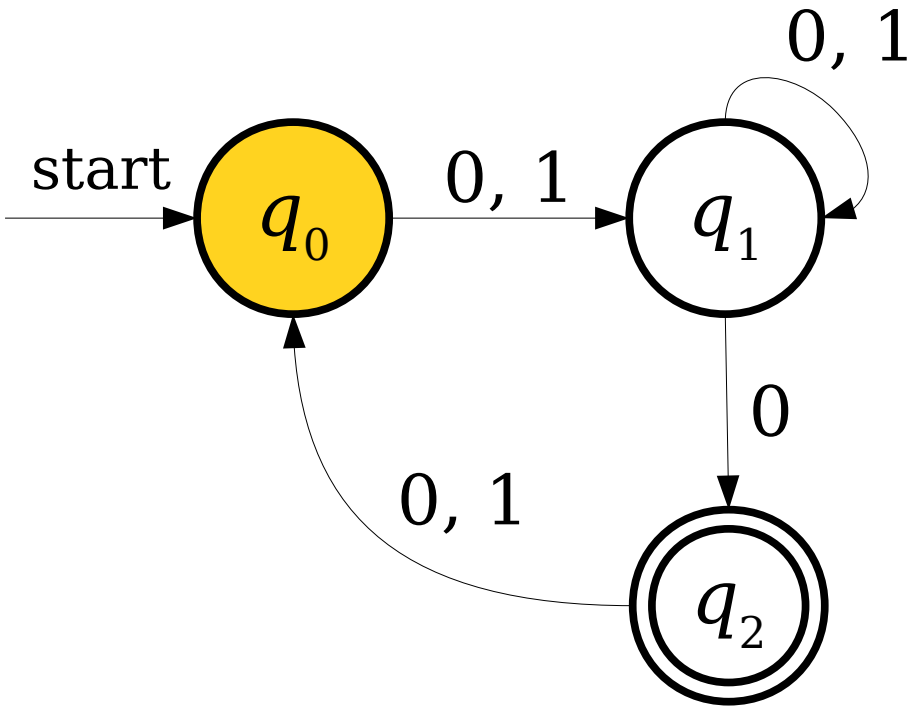


# Another Small Problem



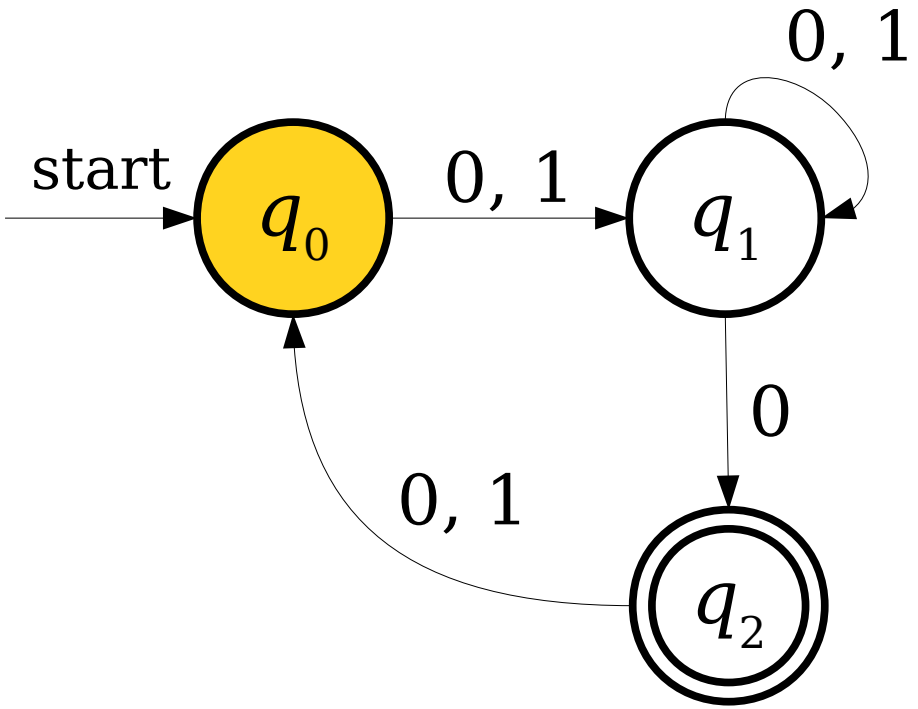

| | | |
|---|---|---|
| **0** | **0** | **0** |

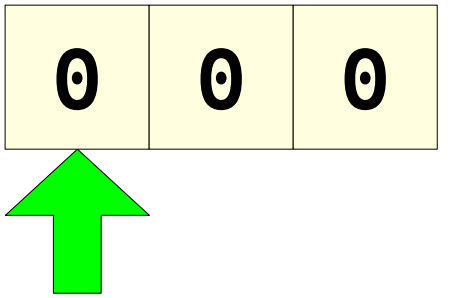
# Another Small Problem



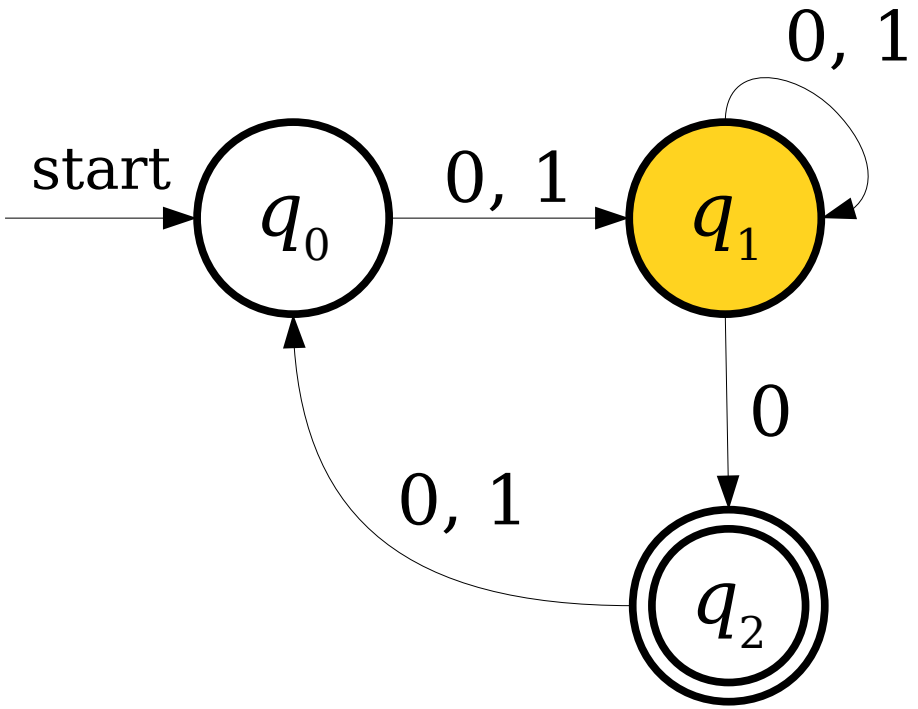

| | | |
|---|---|---|
| 0 | 0 | 0 |

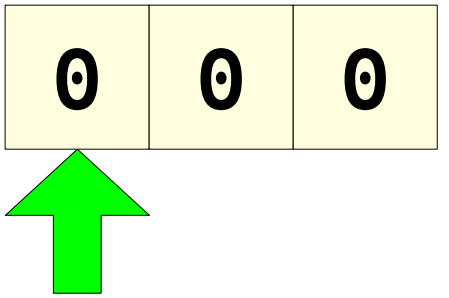
# Another Small Problem



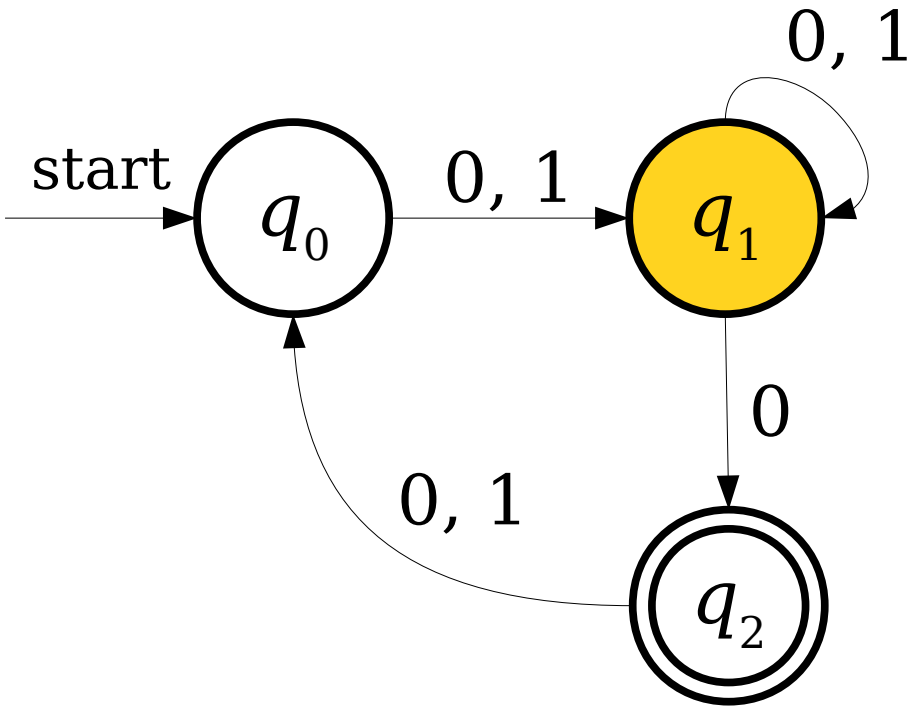



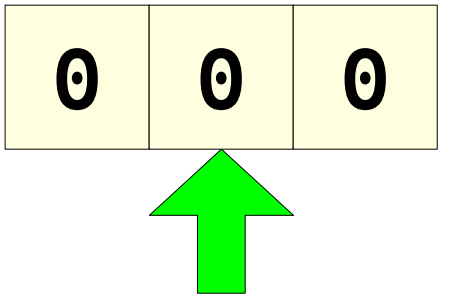
# Another Small Problem



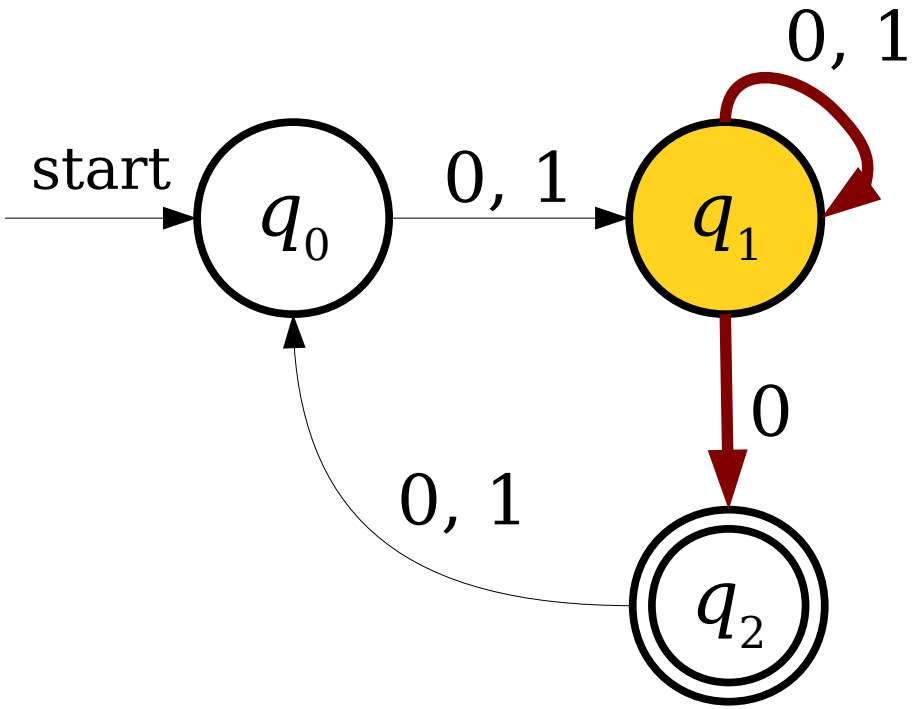



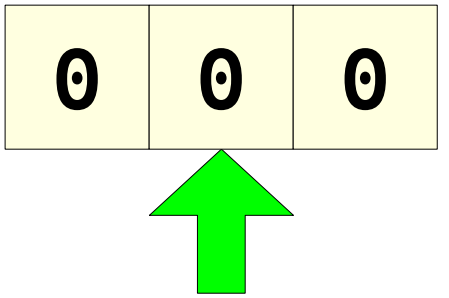
# Another Small Problem



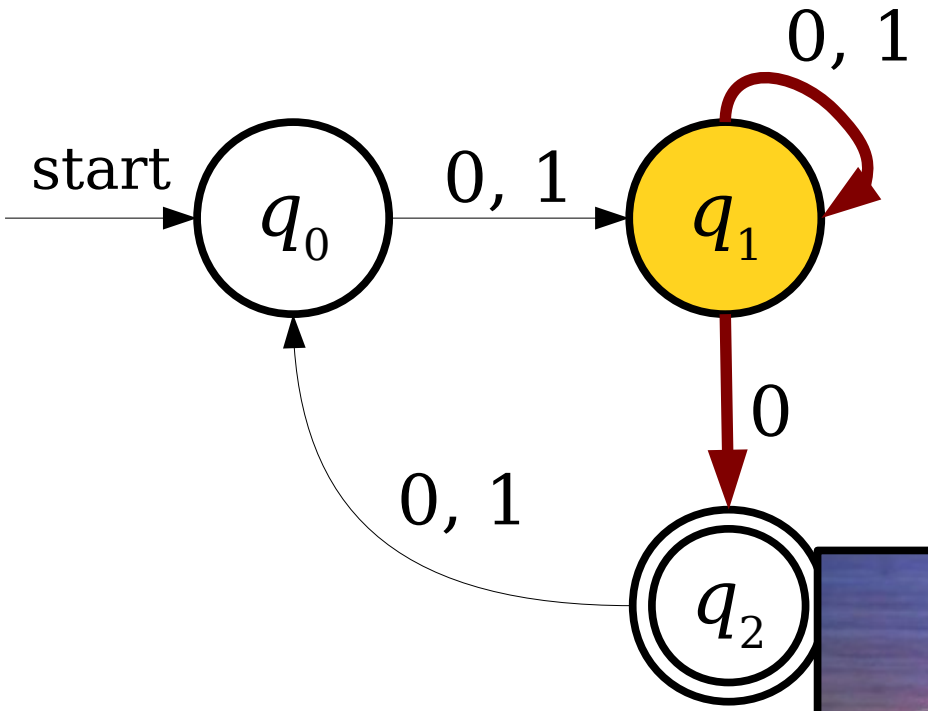



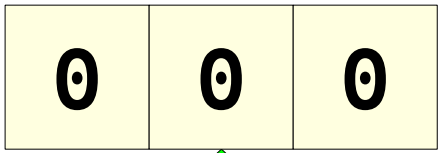
# Another Small Problem

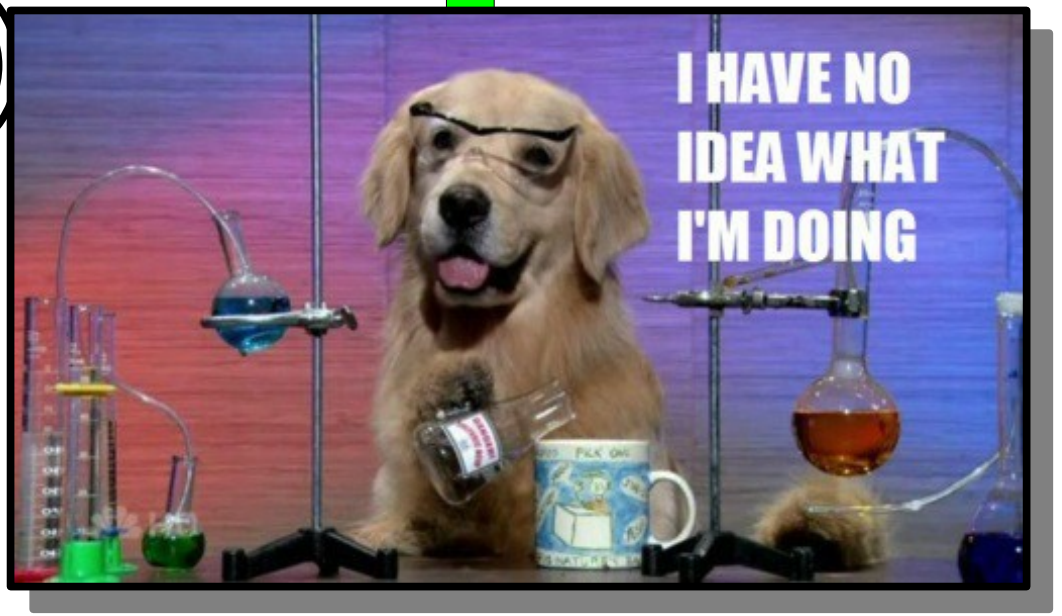


# Another Small Problem



# The Need for Formalism

- To reason about what finite automata can and cannot do, we should specify their behavior in *all* cases.
- 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?

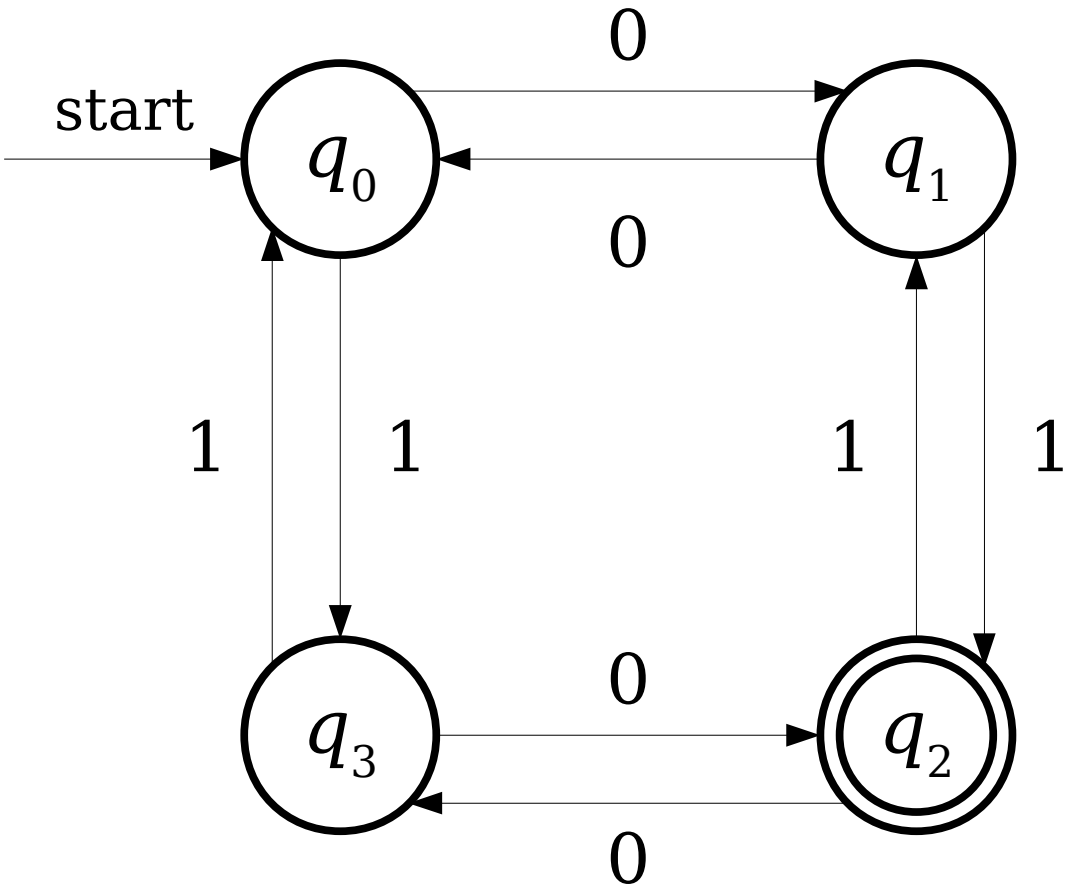
# DFAs

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

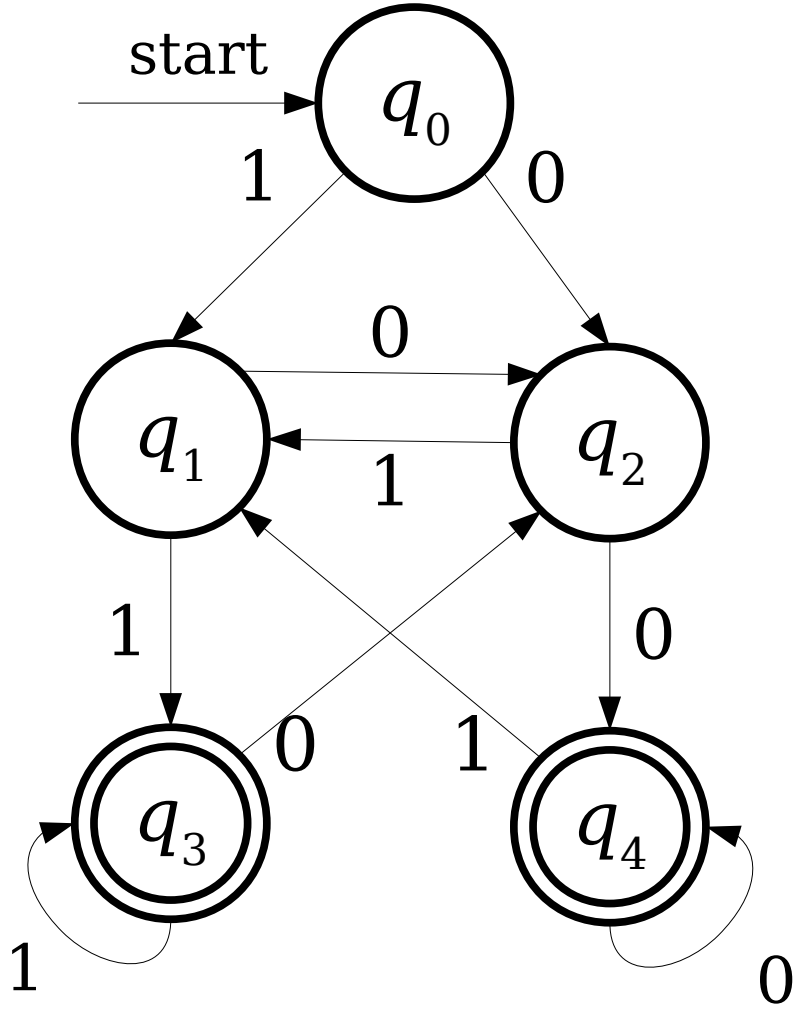
# DFAs

- 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.

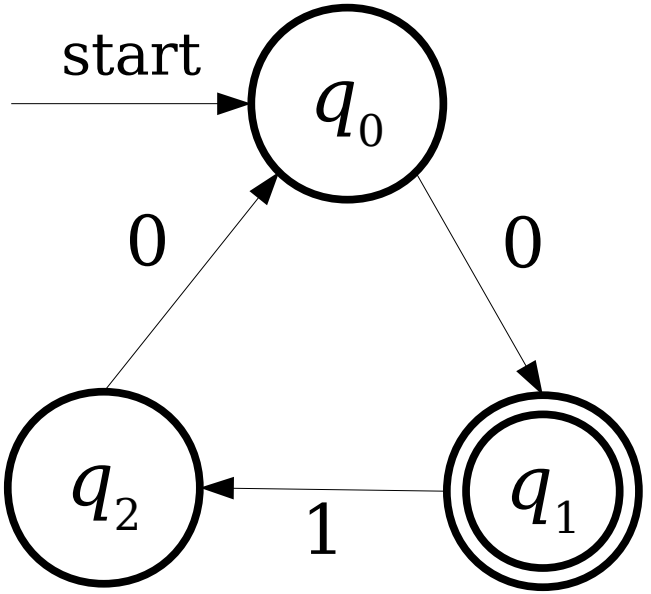
# Is this a DFA over {0, 1}?



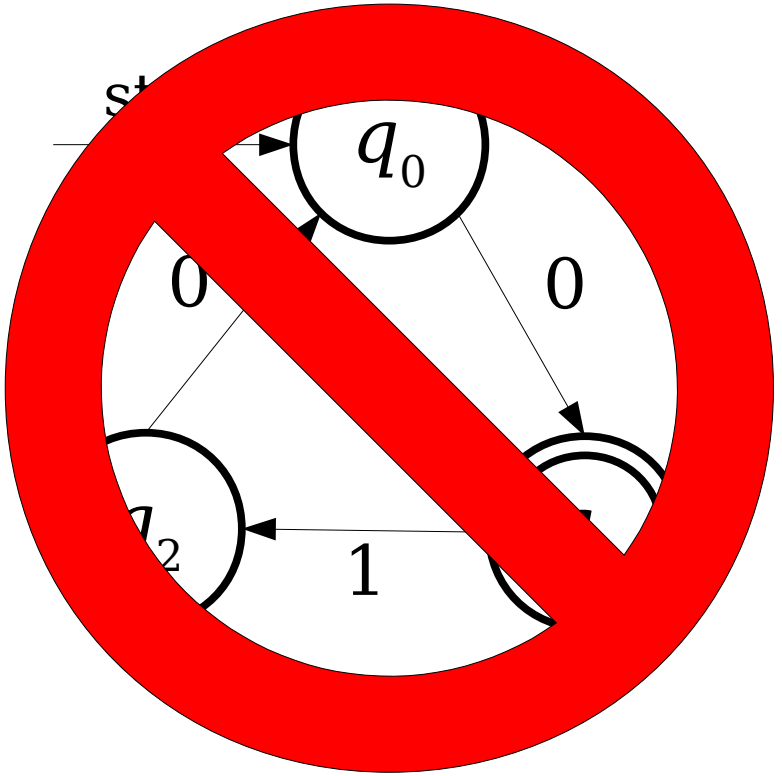
# Is this a DFA over {0, 1}?



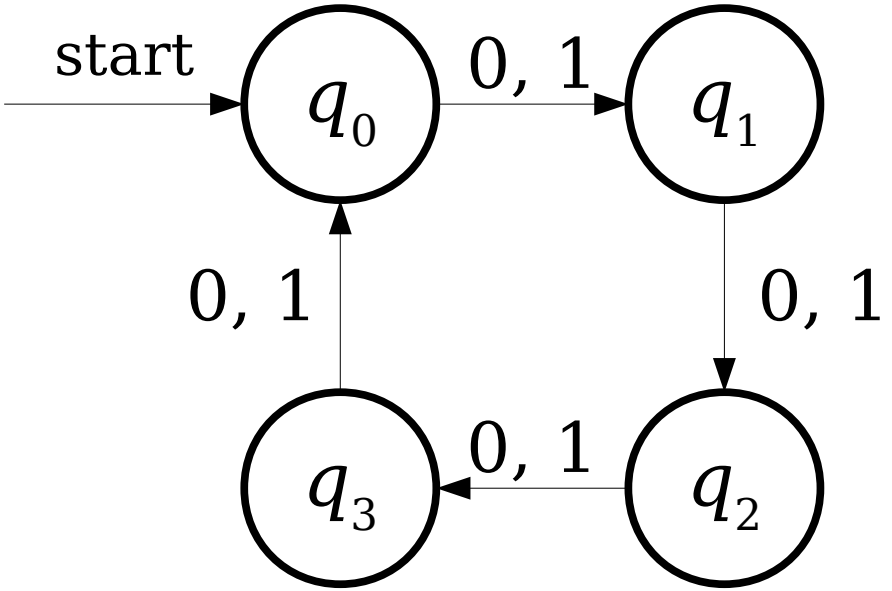
# Is this a DFA over {0, 1}?



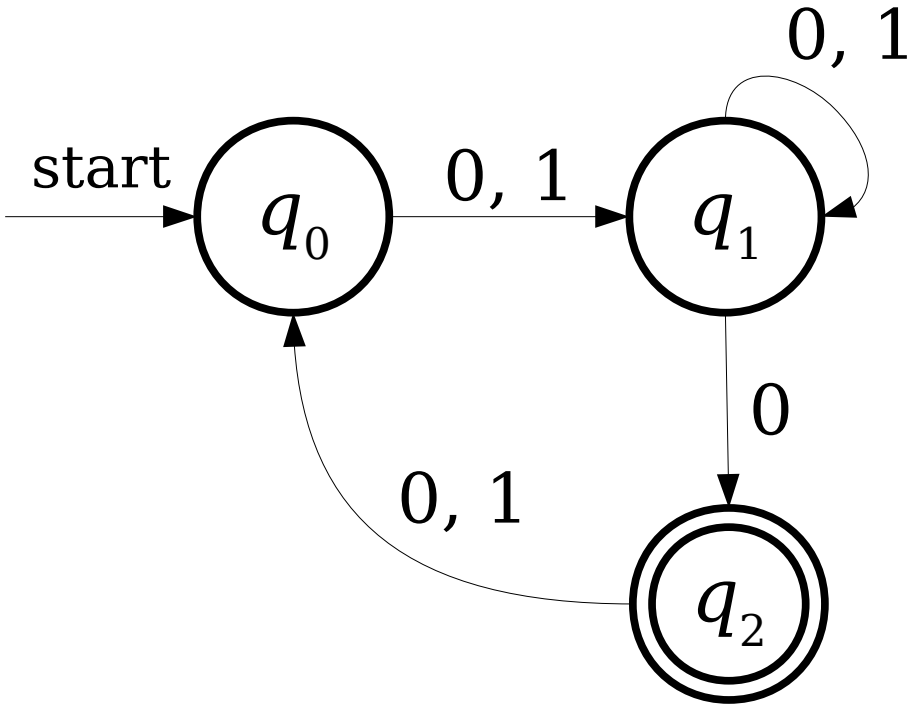
# Is this a DFA over {0, 1}?



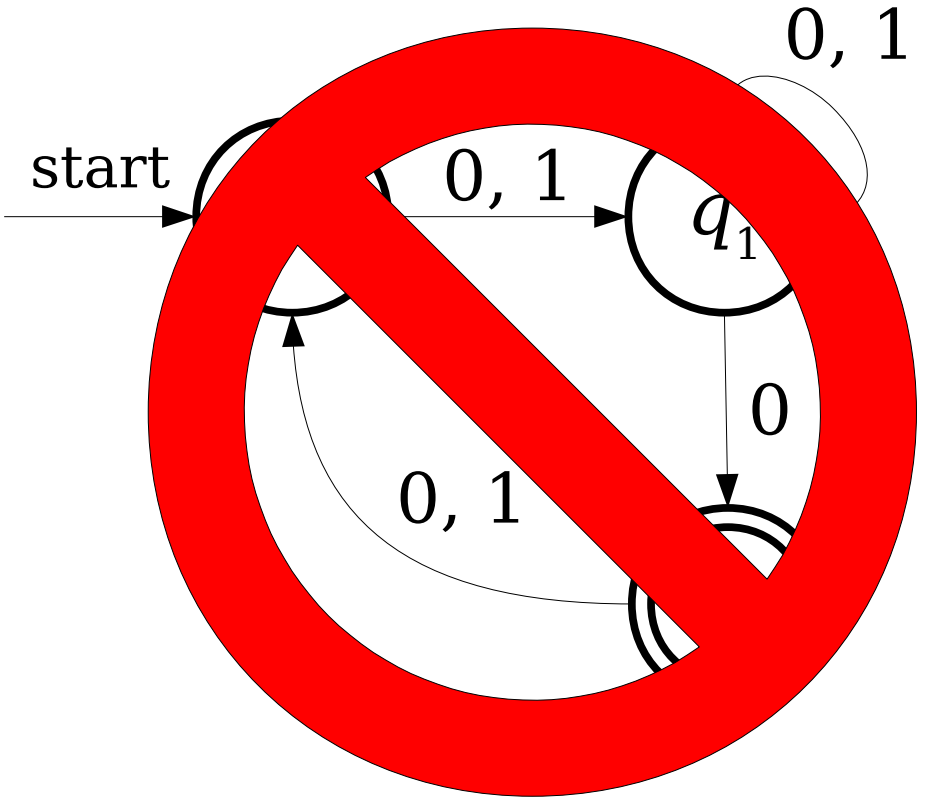
# Is this a DFA over {0, 1}?



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



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



# Designing DFAs

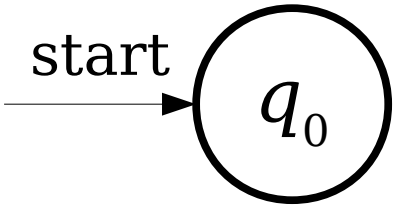
- At each point in its execution, the DFA can only remember what state it is in.
- ***DFA Design Tip:*** Build each state to correspond to some piece of information you need to remember.
  - Each state acts as a “memento” of what you're supposed to do next.
  - Only finitely many different states means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

# Recognizing Languages with DFAs

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

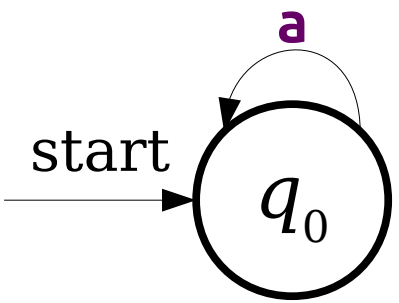
# Recognizing Languages with DFAs

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



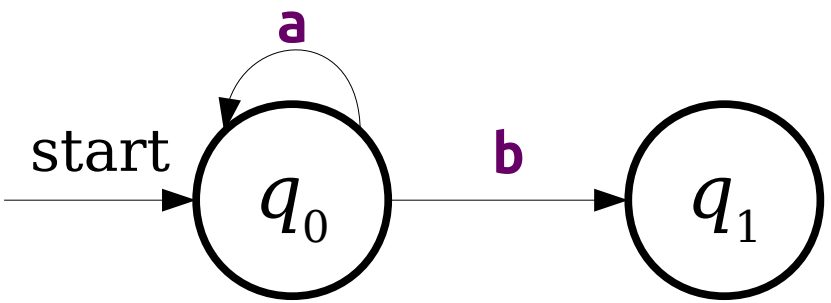
# Recognizing Languages with DFAs

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



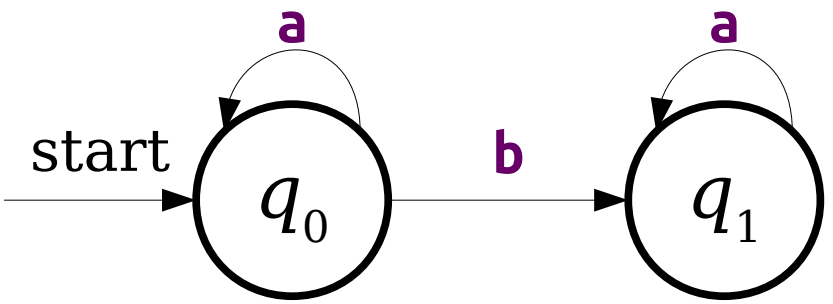
# Recognizing Languages with DFAs

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



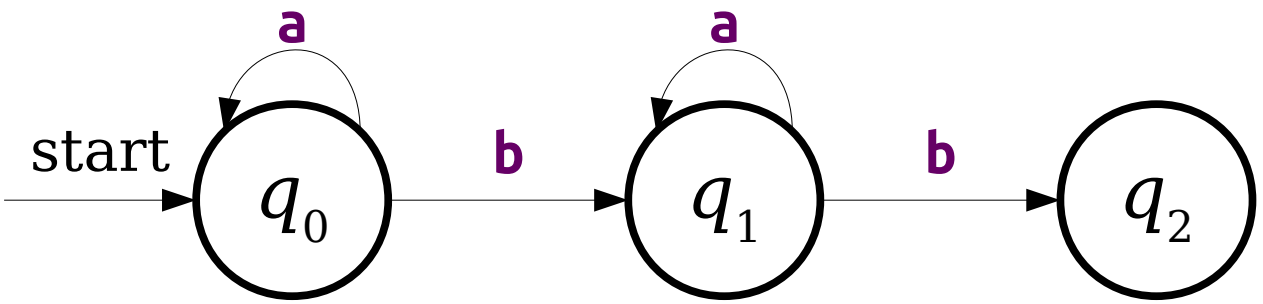
# Recognizing Languages with DFAs

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



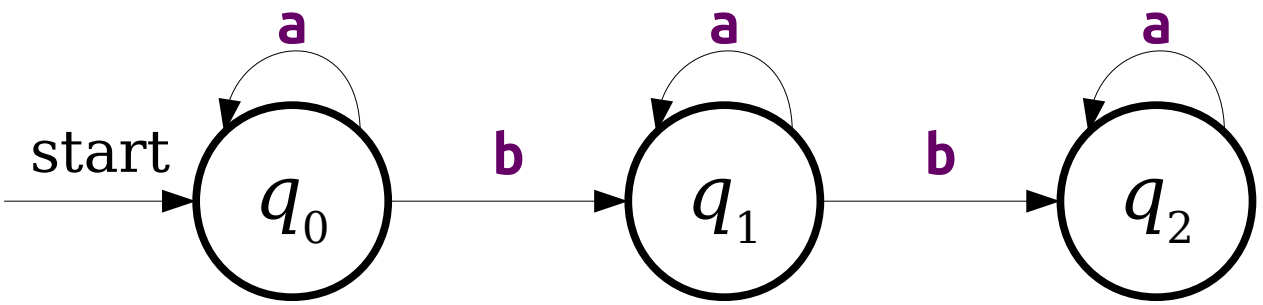
# Recognizing Languages with DFAs

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



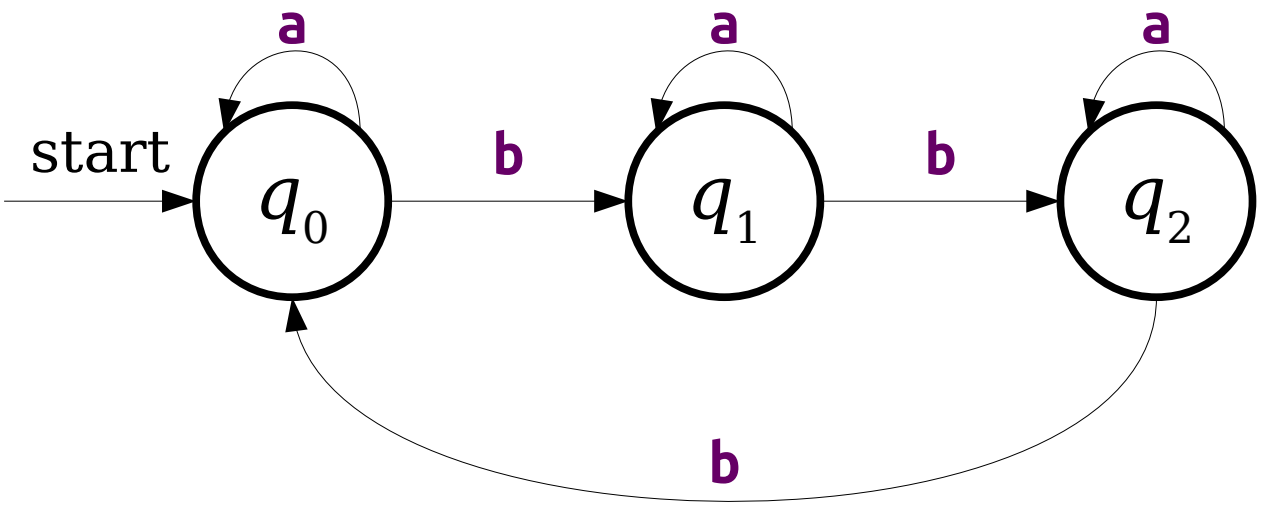
# Recognizing Languages with DFAs

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



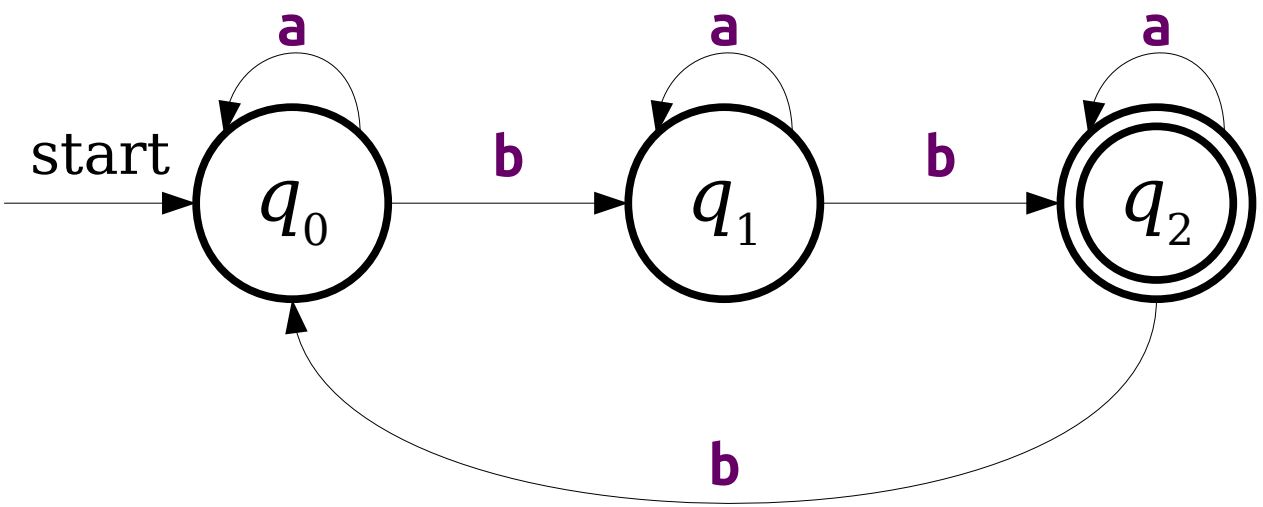
# Recognizing Languages with DFAs

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



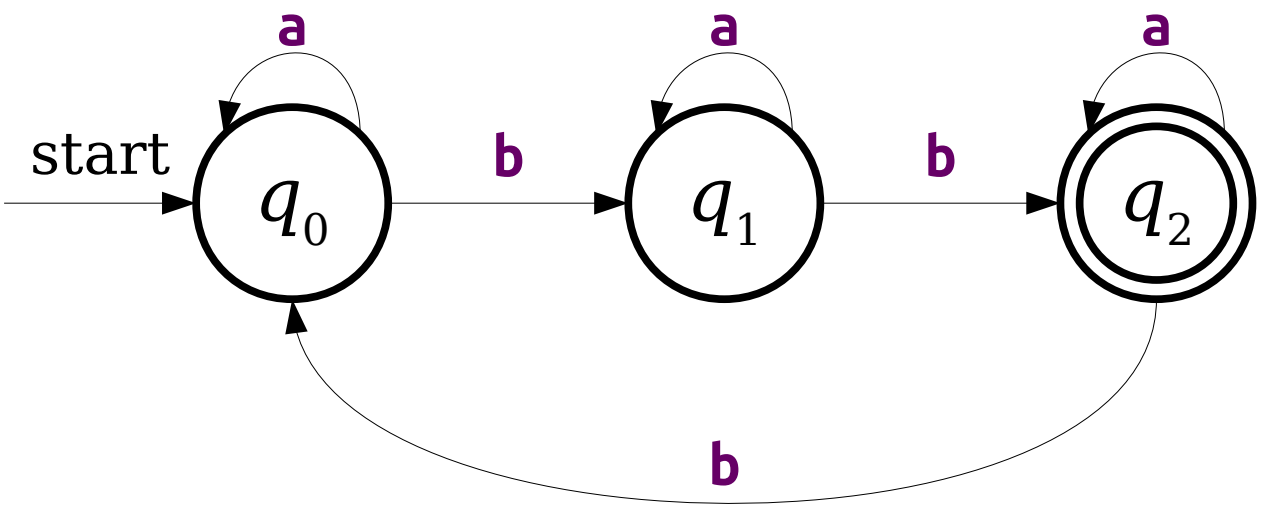
# Recognizing Languages with DFAs

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



# Recognizing Languages with DFAs

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



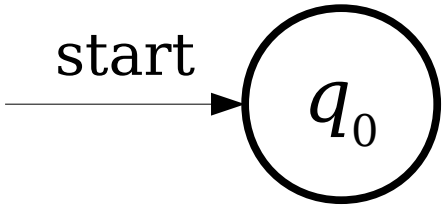

Each state remembers the remainder of the number of **b**s seen so far modulo three.

# Recognizing Languages with DFAs

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

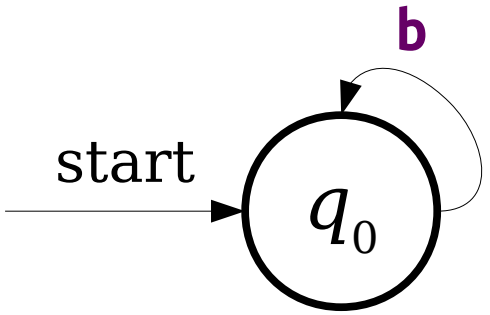
# Recognizing Languages with DFAs

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



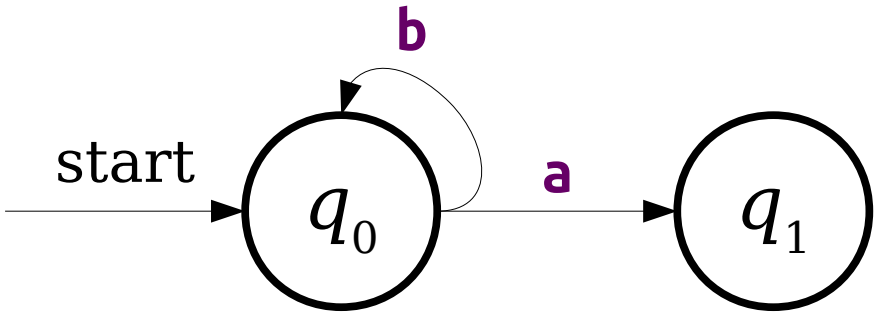
# Recognizing Languages with DFAs

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



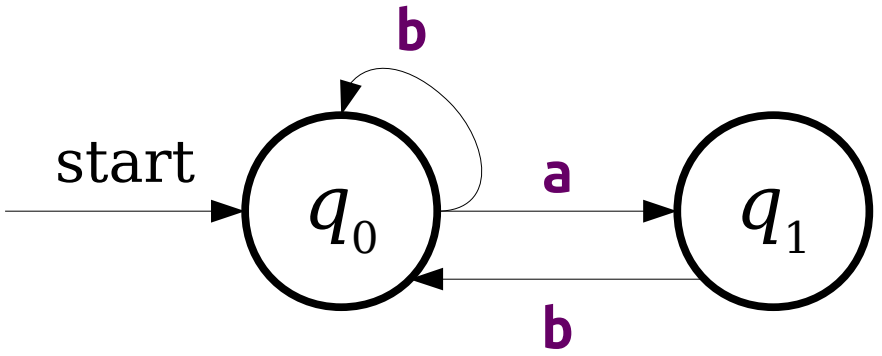
# Recognizing Languages with DFAs

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



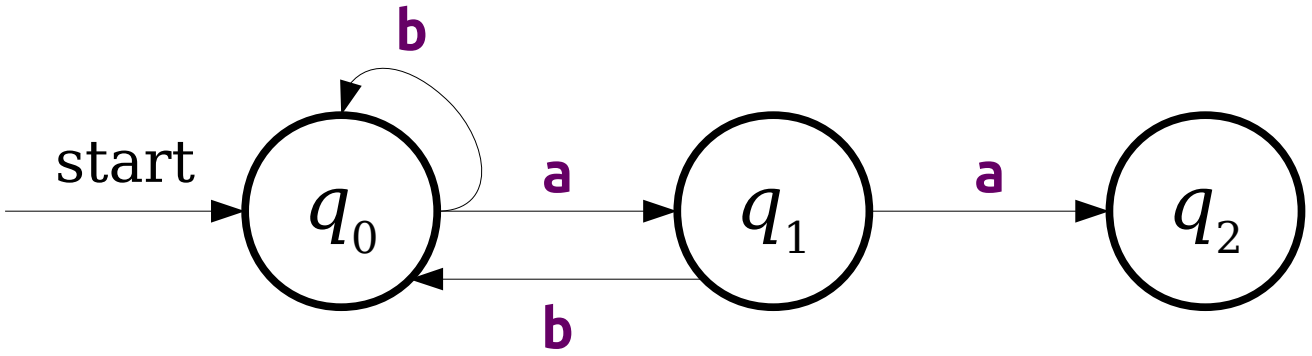
# Recognizing Languages with DFAs

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



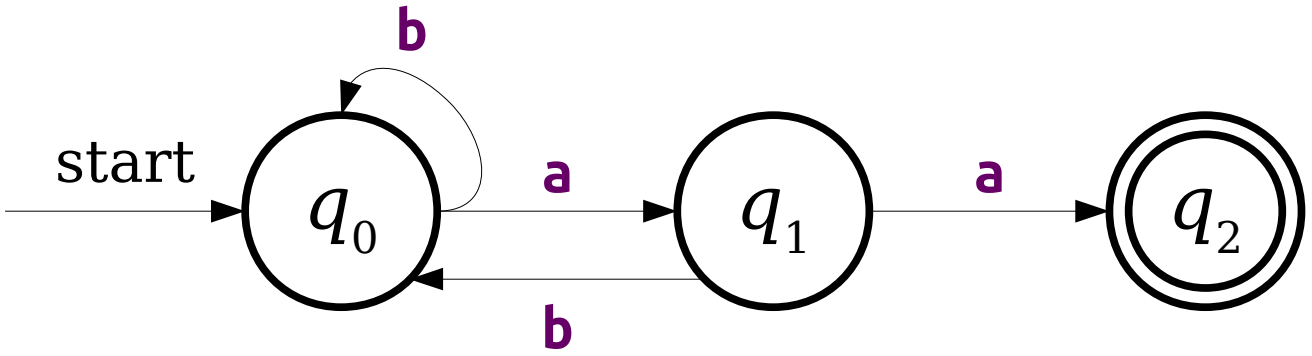
# Recognizing Languages with DFAs

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



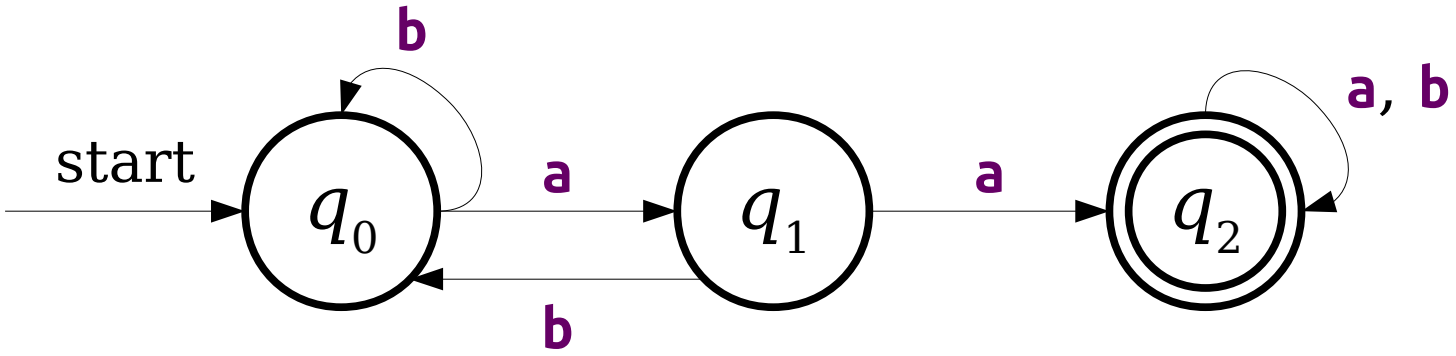
# Recognizing Languages with DFAs

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



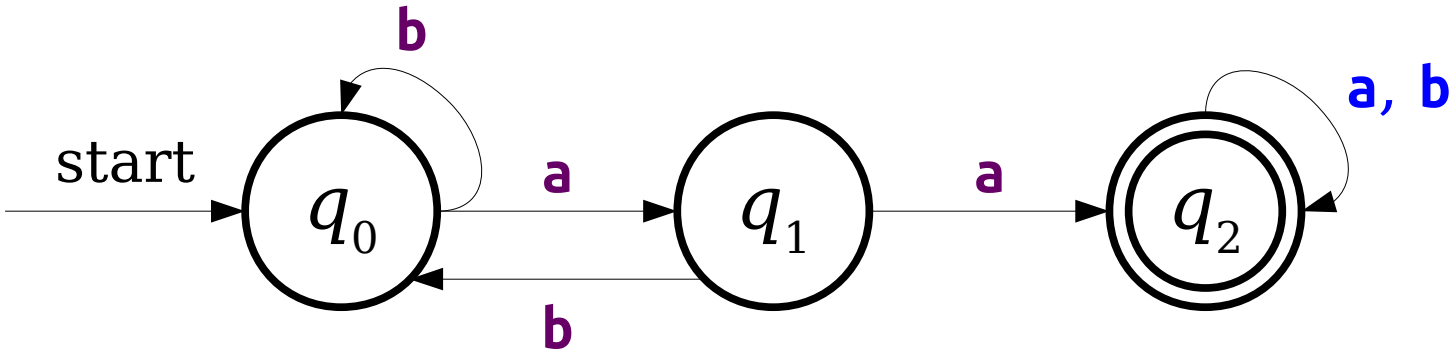
# Recognizing Languages with DFAs

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



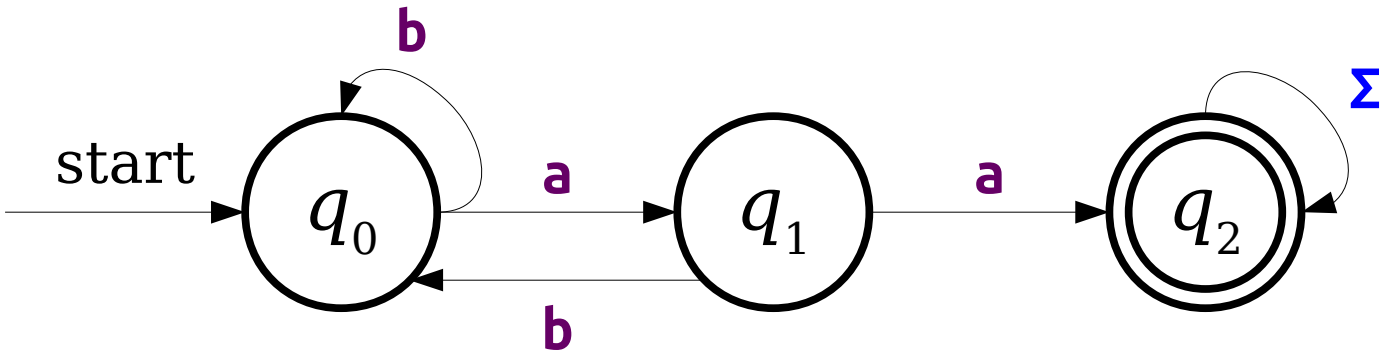
# Recognizing Languages with DFAs

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



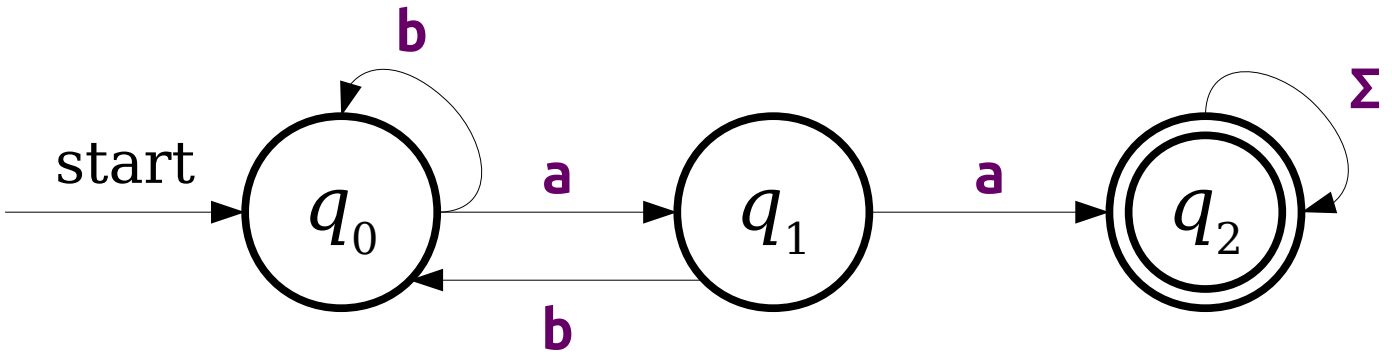
# Recognizing Languages with DFAs

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



# Recognizing Languages with DFAs

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



# More Elaborate DFAs

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

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

We'll design a DFA for comments! Some cases we need to handle:

Accepted:

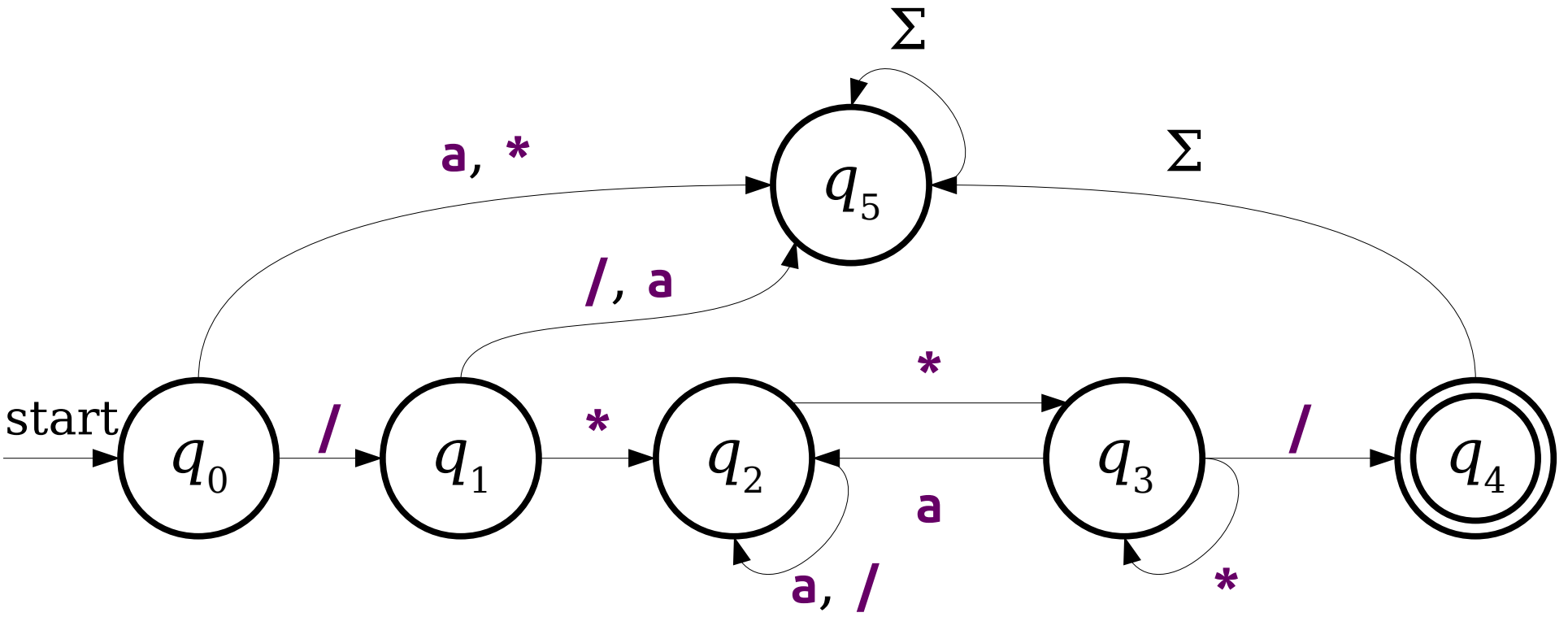
```
/*a*/
/**/
/***/
/*aaa*aaa*/
/*a/a*/
```

Rejected:

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

# More Elaborate DFAs

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



# Next Time

- ***Regular Languages***
  - An important class of languages.
- ***Nondeterministic Computation***
  - Why must computation be linear?
- ***NFAs***
  - Automata with superpowers.