CS103
Winter 2026

Lecture 14:

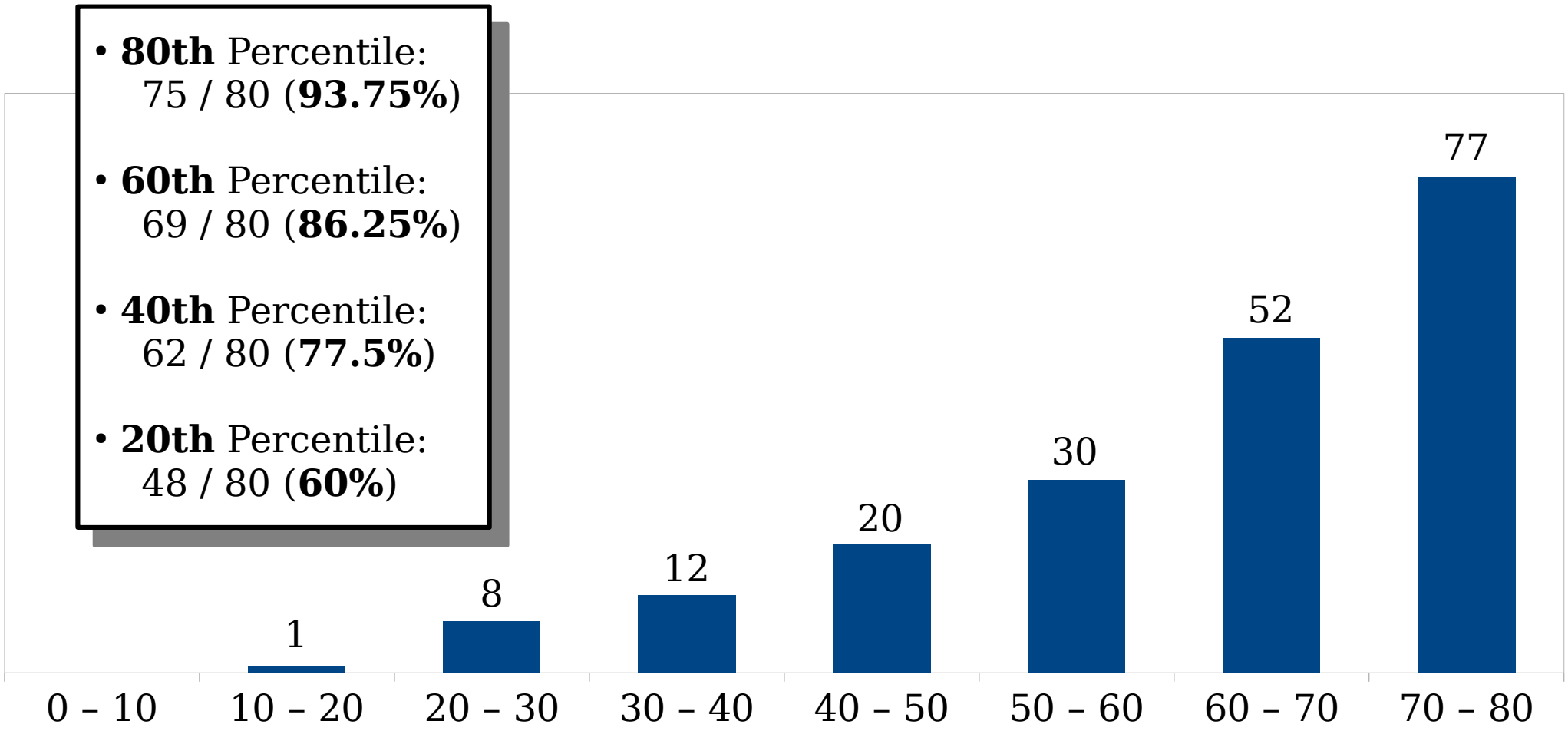
Finite Automata

Part 1 of 3

Finite Automata

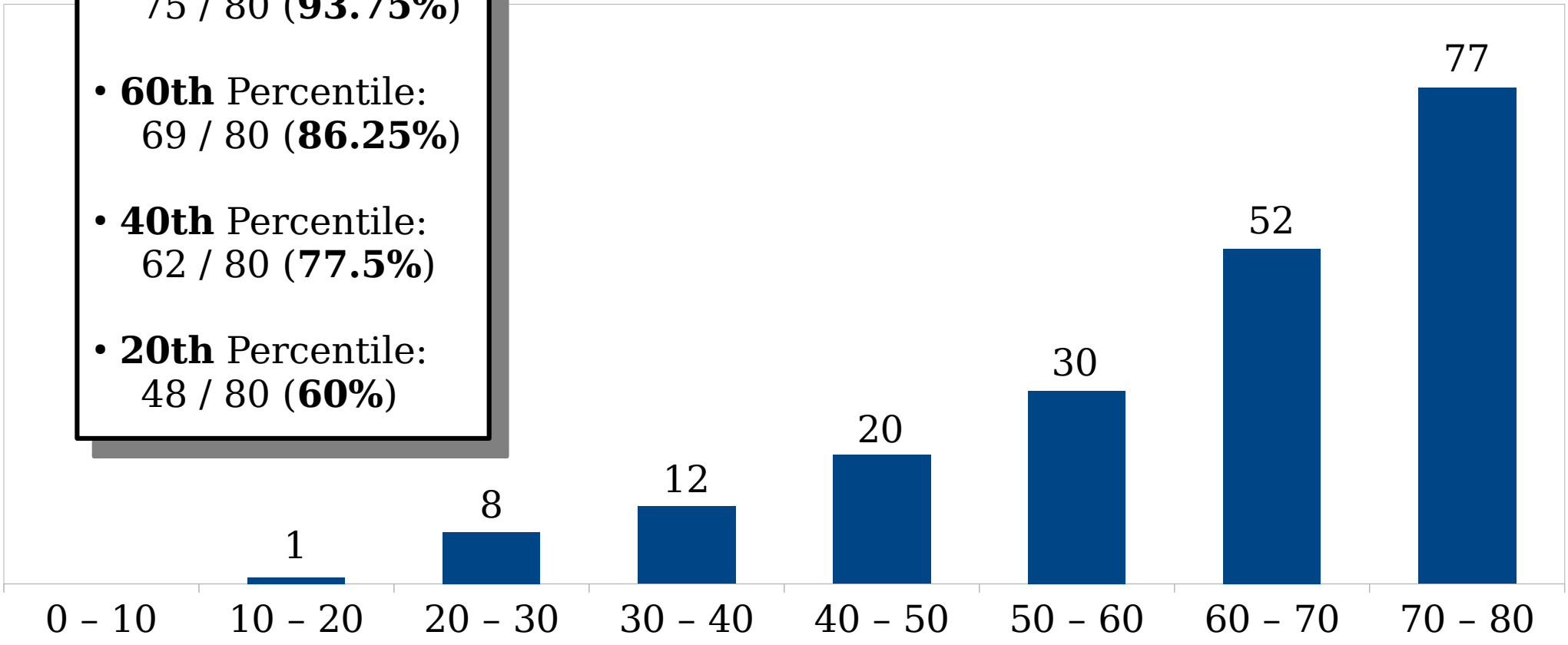
Part 1

1. **Midterm Results**
2. Computability Theory: The Central Question
3. Toward a Model of Computation
4. Automata
5. Languages, Strings, and Alphabets
6. A Small Problem
7. DFAs
8. Designing DFAs
9. What’s Next?



- If your score isn't in the range you're used to, ask yourself what happened, and be thoughtful and honest with yourself.
- Approach conversations with course staff genuinely. We approach them from a place of non-judgment.
- If anything about the grading feels arbitrary, that might be a symptom that you're overlooking a key concept. Come talk with us so we can help clarify!

- **80th** Percentile: 75 / 80 (**93.75%**)
- **60th** Percentile: 69 / 80 (**86.25%**)
- **40th** Percentile: 62 / 80 (**77.5%**)
- **20th** Percentile: 48 / 80 (**60%**)



- We want everyone to be wildly successful!
  - Review feedback
  - Assess (small scattered point losses? one large loss?)
  - 1-on-1s (see Anisha's Ed post)
- Assuming comfort going forward:
  - Contrapositive
  - Negations (implication, quant.)
  - Assume/Prove table
  - Proofwriting Checklist

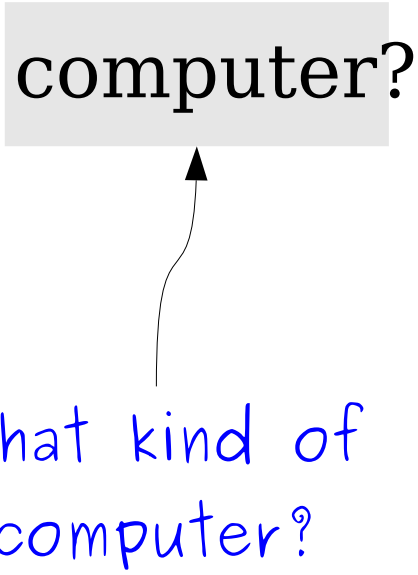
Finite Automata

Part 1

1. Midterm Results
2. **Computability Theory: The Central Question**
3. Toward a Model of Computation
4. Automata
5. Languages, Strings, and Alphabets
6. A Small Problem
7. DFAs
8. Designing DFAs
9. What’s Next?

Computability Theory

What problems can we solve with a computer?



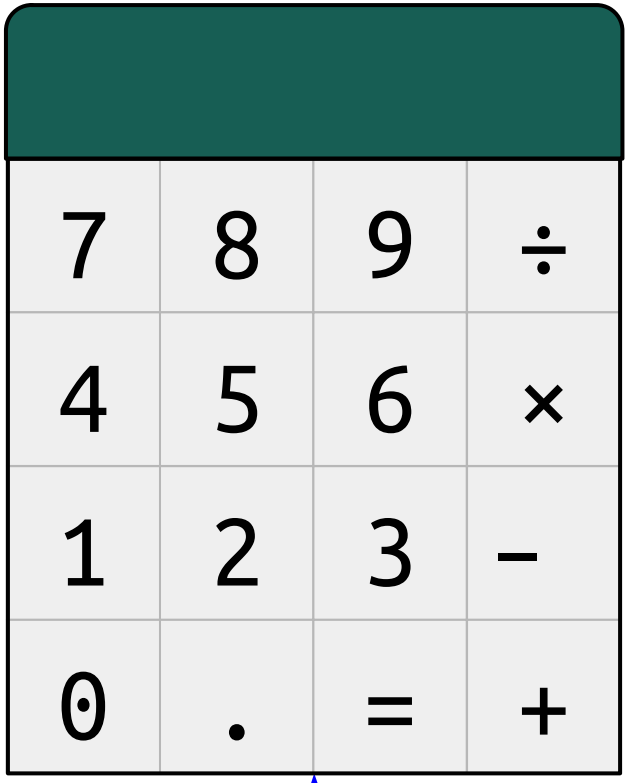
Two Challenges

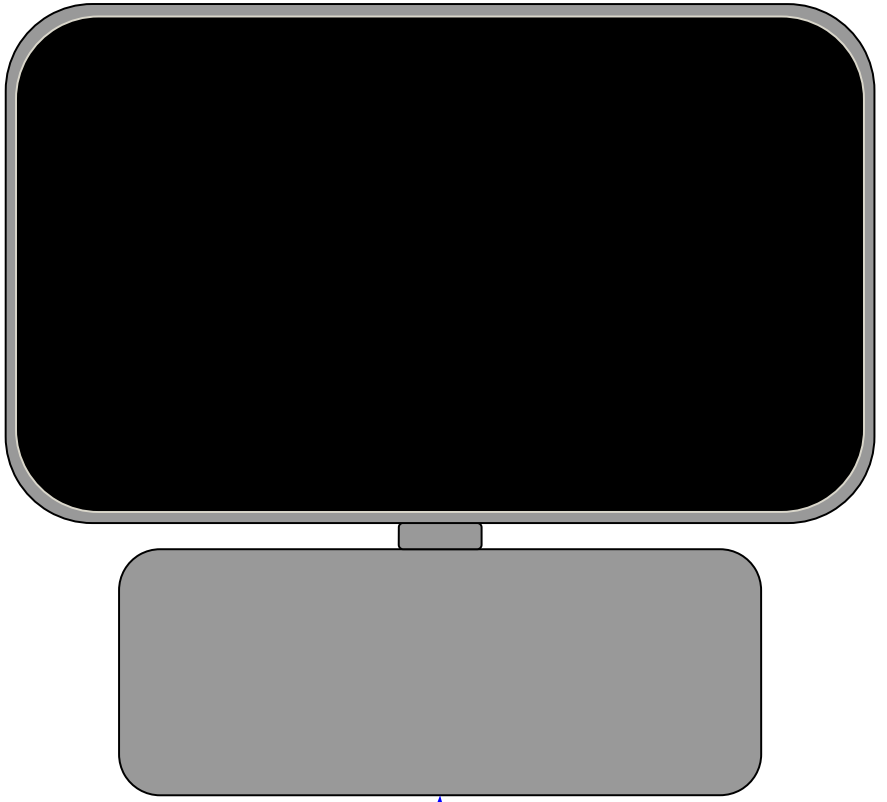
- Writing proofs on formal definitions is hard, and computers are *way* more complicated than sets, graphs, or functions.
- Computers are dramatically better now than they've ever been, and that trend continues.
- ***Key Question:*** How can we prove what computers can and can't do…
  - … without multi-hundred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

Finite Automata

Part 1

1. Midterm Results
2. Computability Theory: The Central Question
3. **Toward a Model of Computation**
4. Automata
5. Languages, Strings, and Alphabets
6. A Small Problem
7. DFAs
8. Designing DFAs
9. What’s Next?





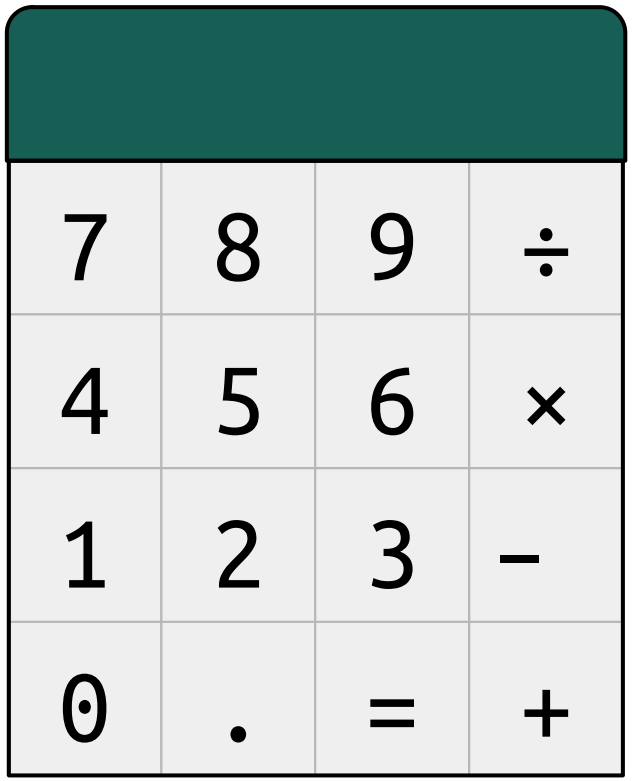
Why does this computer "feel" less powerful...

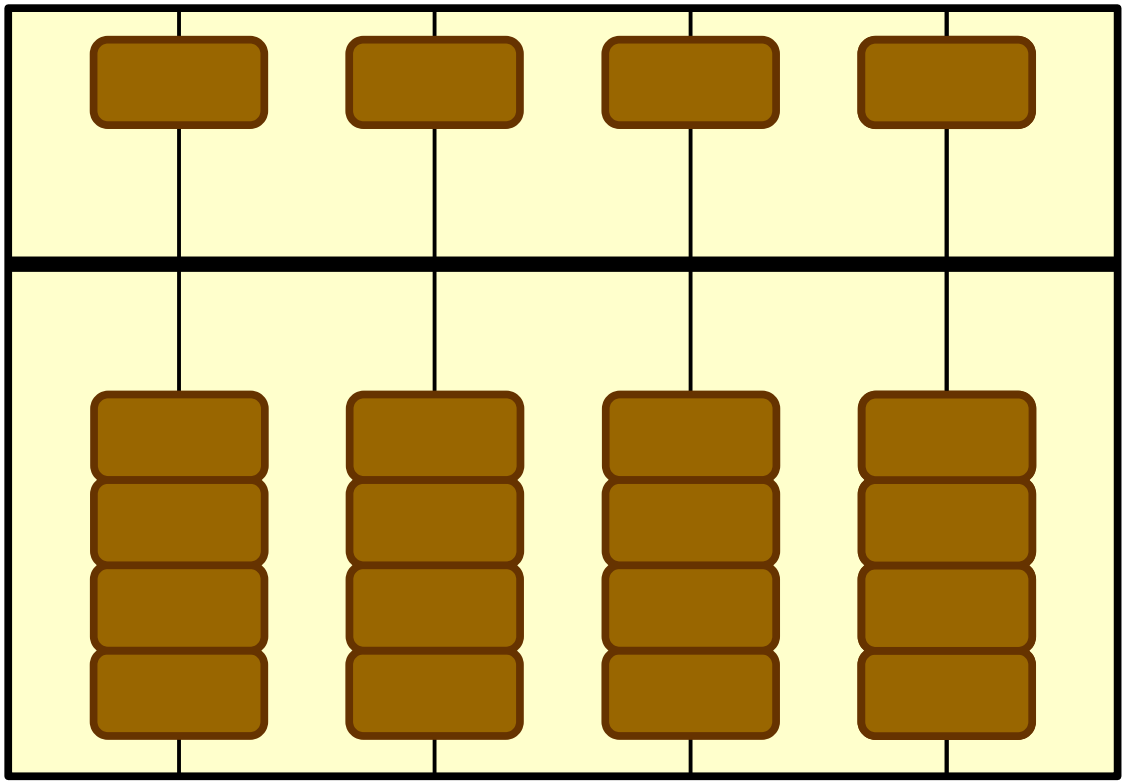
...than this one?

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-esque” computers.
- In CS103, we’ll first explore “small-memory” computers in detail, then discuss “large-memory” computers in depth.

Our Goal: A Unifying Abstraction

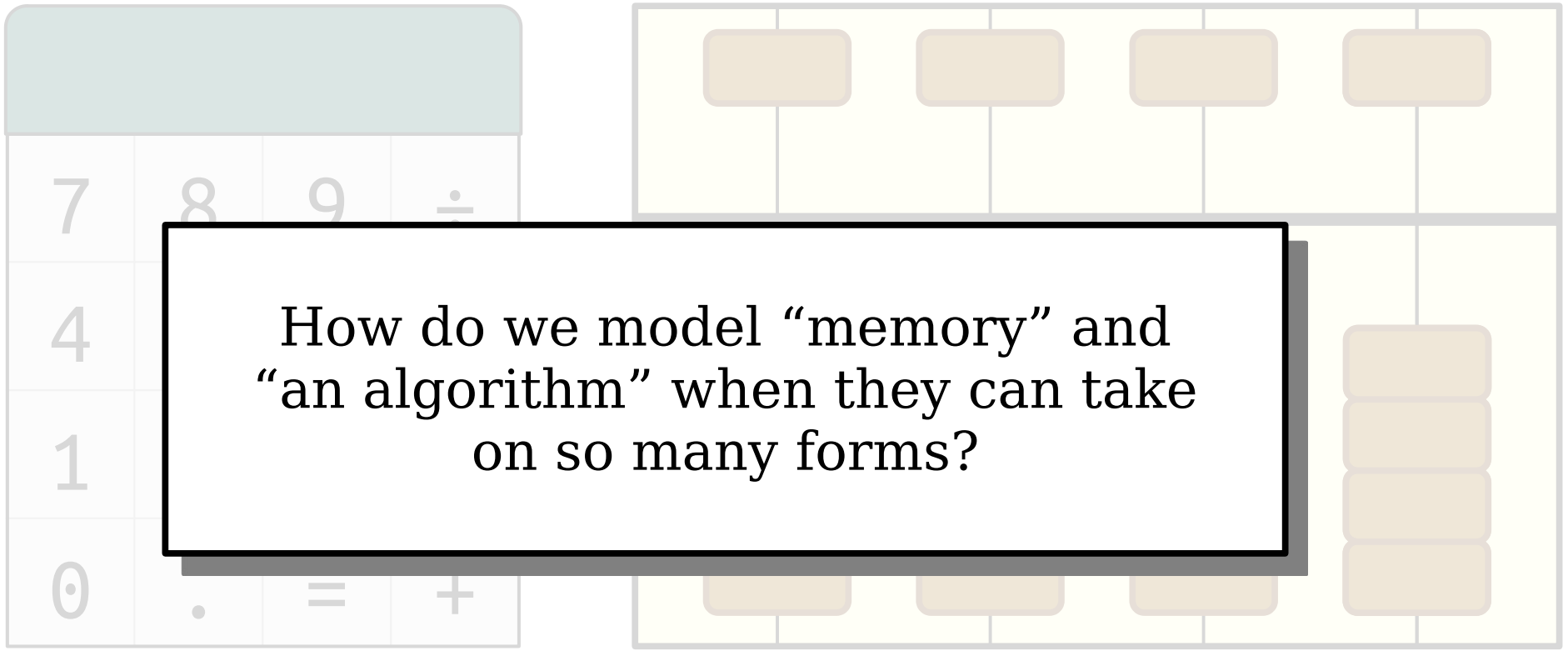




Data stored electronically.
Algorithm is in silicon.
Memory limited by display.

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

Our Goal: A Unifying Abstraction



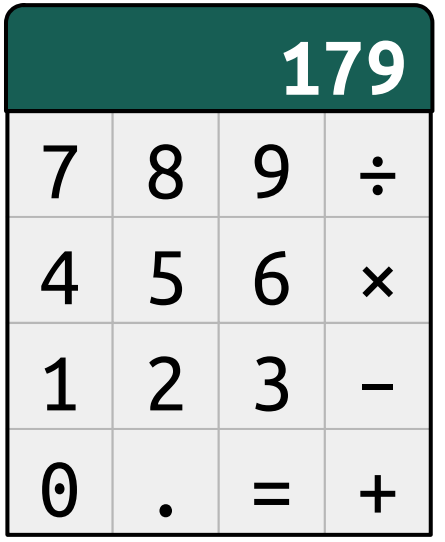
How do we model “memory” and “an algorithm” when they can take on so many forms?

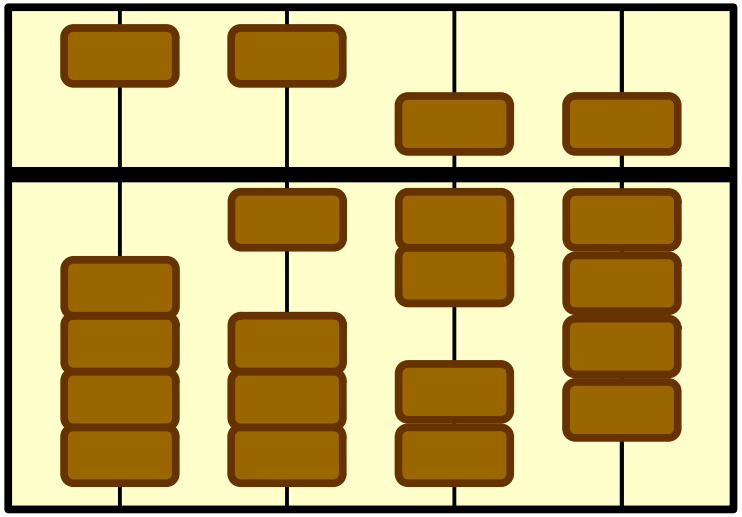
Data stored electronically.
Algorithm is in silicon.
Memory limited by display.

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

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.





Finite Automata

Part 1

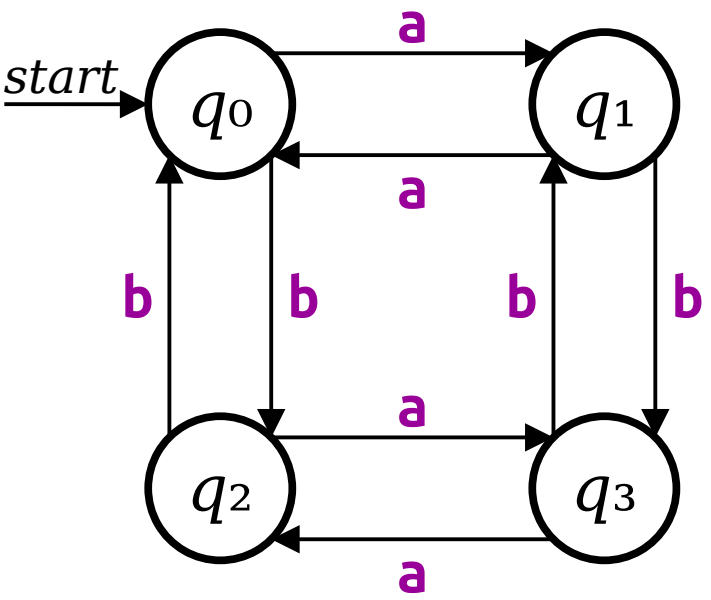
1. Midterm Results
2. Computability Theory: The Central Question
3. Toward a Model of Computation
4. **Automata**
5. Languages, Strings, and Alphabets
6. A Small Problem
7. DFAs
8. Designing DFAs
9. What's Next?

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 in a small space.
- They're also fascinating and useful in their own rights. More on that later.

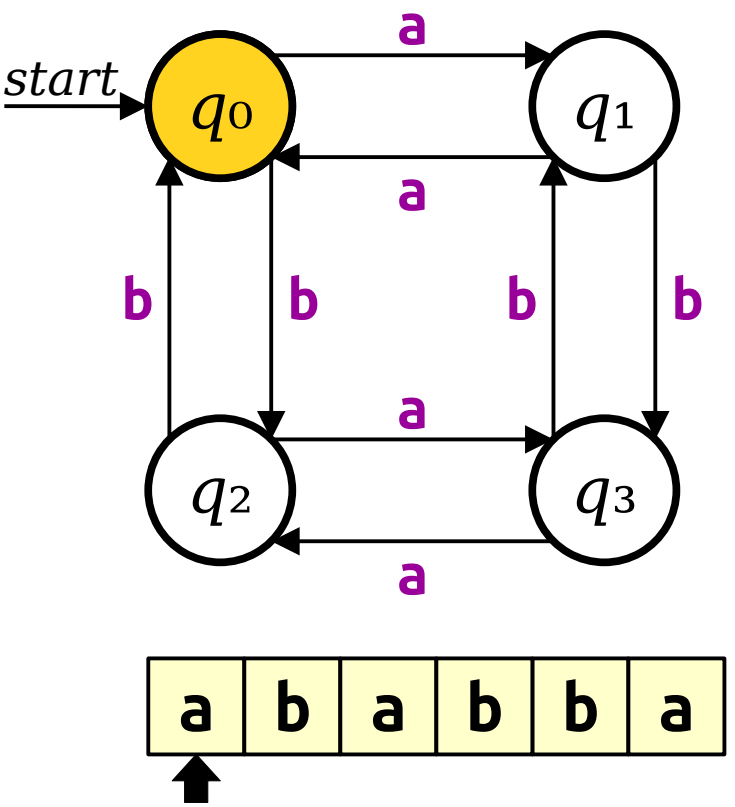
Modeling Finite Computation

- 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. This is super abstract!
- Each transition indicates how memory changes in response to inputs.
- Some state is designated as the ***start state***. The computation begins in that state.



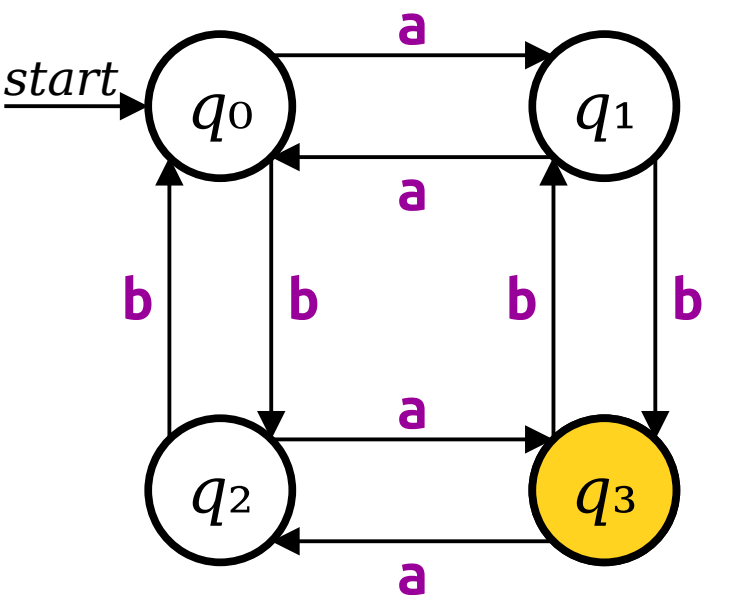
Modeling Finite Computation

- This device processes ***strings*** made of ***characters***.
  - Each character represents some external input to the device.
  - The string represents the full 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.



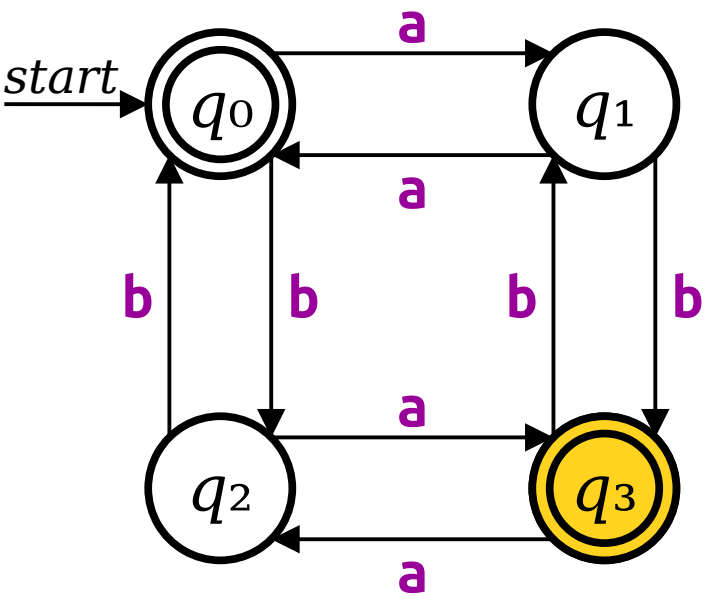
Modeling Finite Computation

- Once we've finished entering all the characters of our input, we need to obtain the result of the computation.
- In general, computers can produce all sorts of things as the result of a computation: a number, a piece of text, etc.
- As a simplifying assumption, we'll assume that we just need to get a single bit of output. That is, our machines will just say `YES` or `NO`.
- (This can be generalized – come talk to us after class if you're curious how!)



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`).



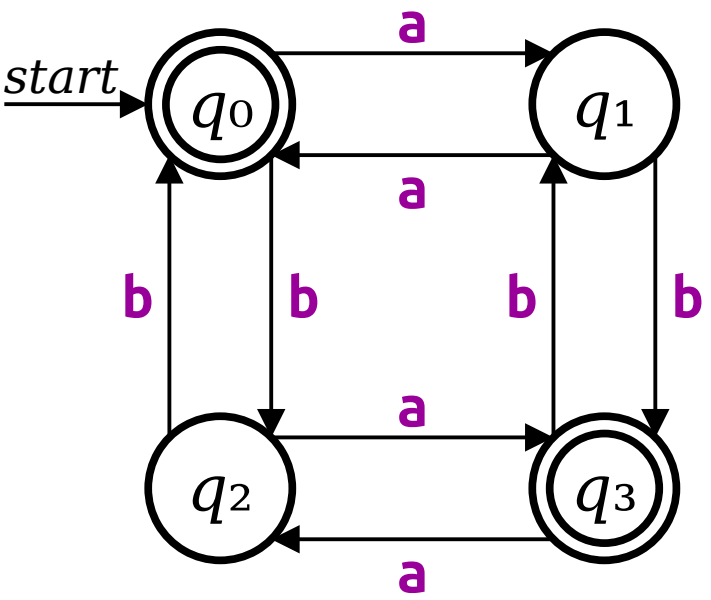
Modeling Finite Computation

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

aab

aabb

abbababba

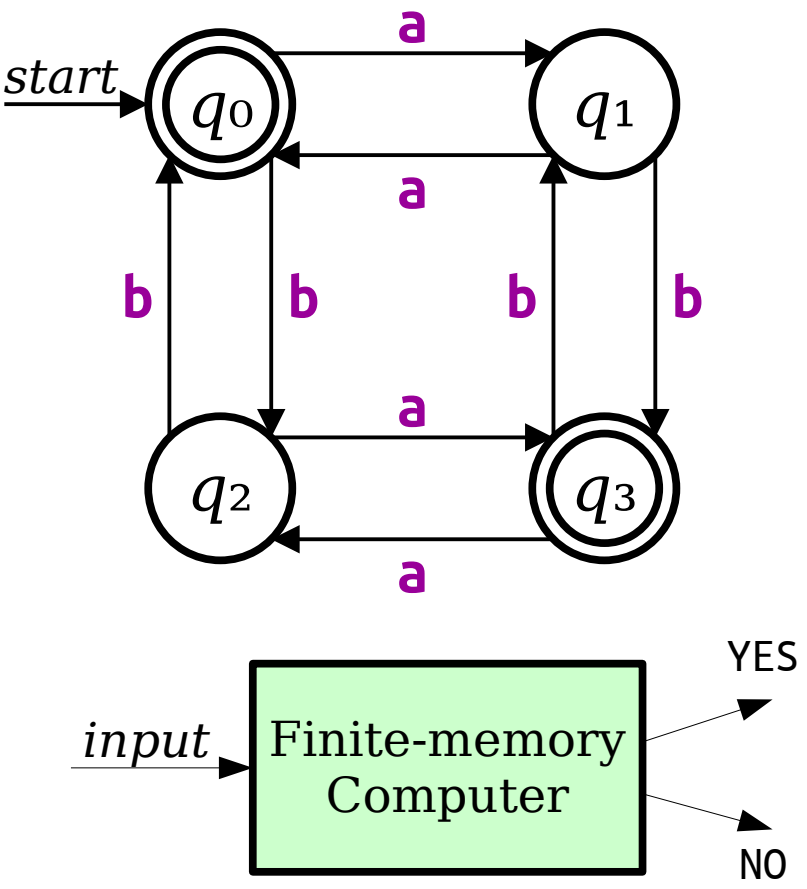


Answer at

https://cs103.stanford.edu/pollev

Finite Automata

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



Finite Automata

Part 1

1. Midterm Results
2. Computability Theory: The Central Question
3. Toward a Model of Computation
4. Automata
5. **Languages, Strings, and Alphabets**
6. A Small Problem
7. DFAs
8. Designing DFAs
9. What’s Next?

Formalizing Things

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

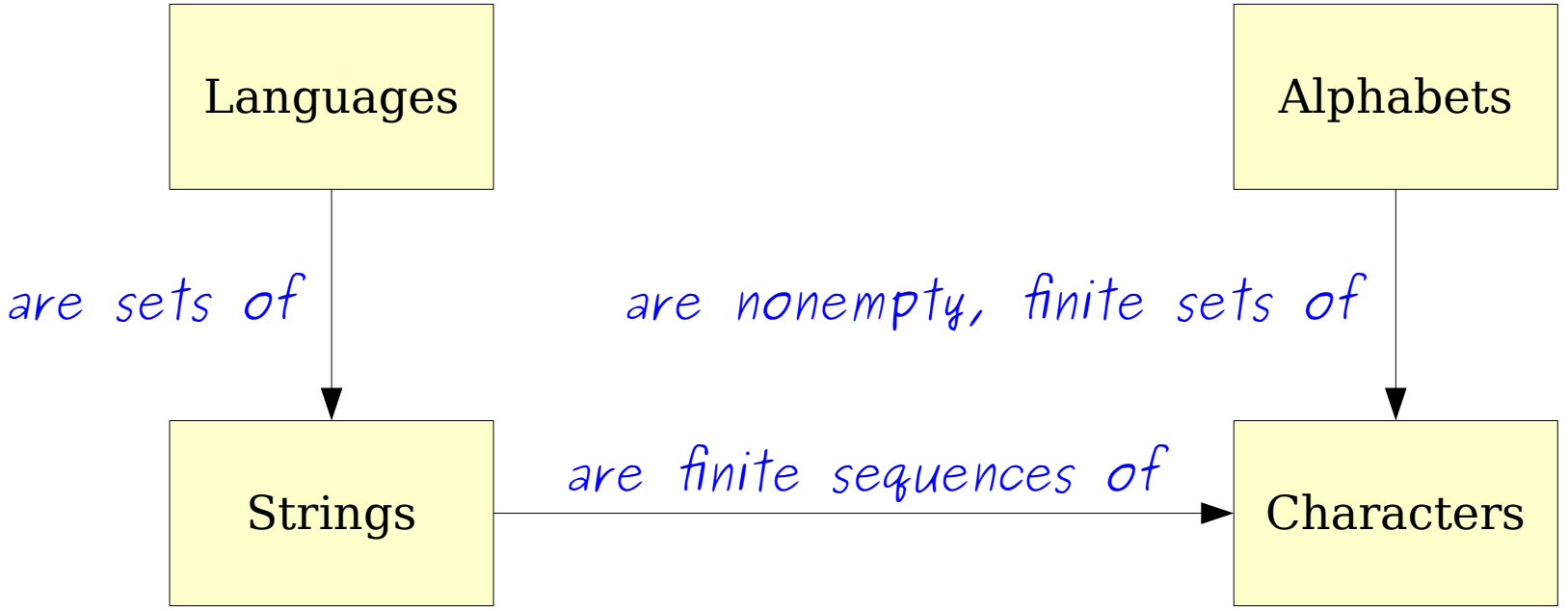
- A ***language over Σ*** is a set L consisting of strings over Σ.
- Example: The language of palindromes over Σ = {**a**, **b**, **c**} is the set
  - {ε, **a**, **b**, **c**, **aa**, **bb**, **cc**, **aaa**, **aba**, **aca**, **bab**, … }
- The set of all strings composed from letters in Σ is denoted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

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!***

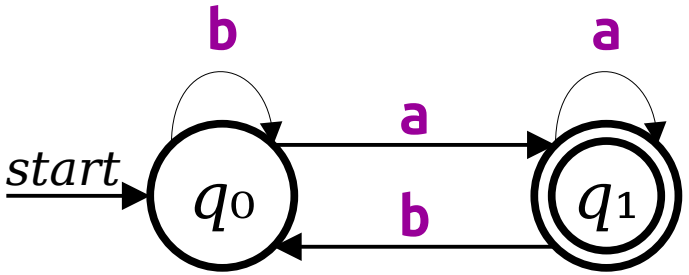
The Cast of Characters

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



Finite Automata and Languages

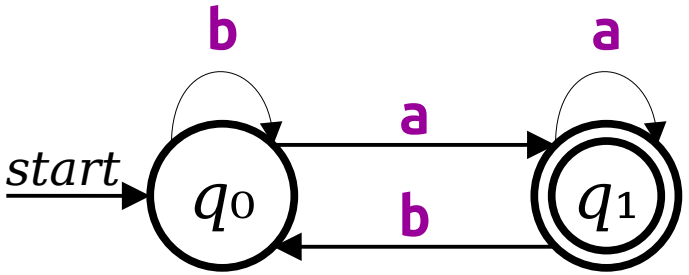
- Let A be an automaton that processes strings drawn from an alphabet Σ.
- The ***language of A***, denoted $\mathcal{L}(A)$, is the set of strings over Σ that A accepts:



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

Finite Automata and Languages

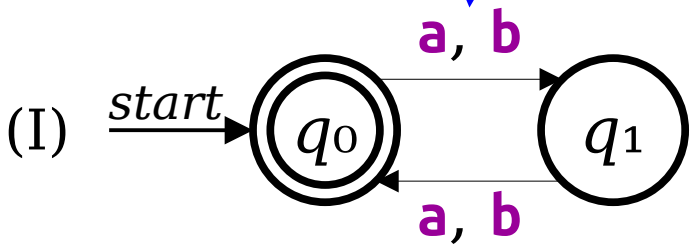
- Let D be the automaton shown to the right. It processes strings over {**a**, **b**}.
- Notice that D accepts all strings of **a**'s and **b**'s that end in **a** and rejects everything else.

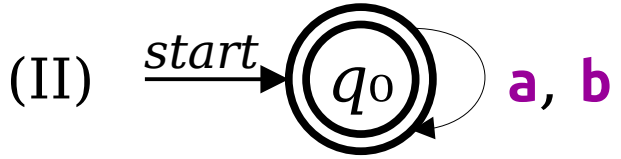


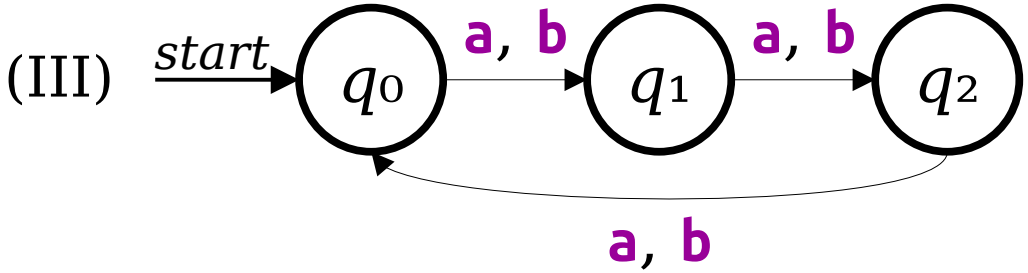
- So $\mathcal{L}(D) = \{ w \in \{\mathbf{a}, \mathbf{b}\}^* \mid w \text{ ends in } \mathbf{a} \}$.

$\mathcal{L}(A) = \{ w \in \Sigma^* \mid A$ accepts $w \}$

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







What are the languages of these automata? Answer at ***https://cs103.stanford.edu/pollev***

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

The Story So Far

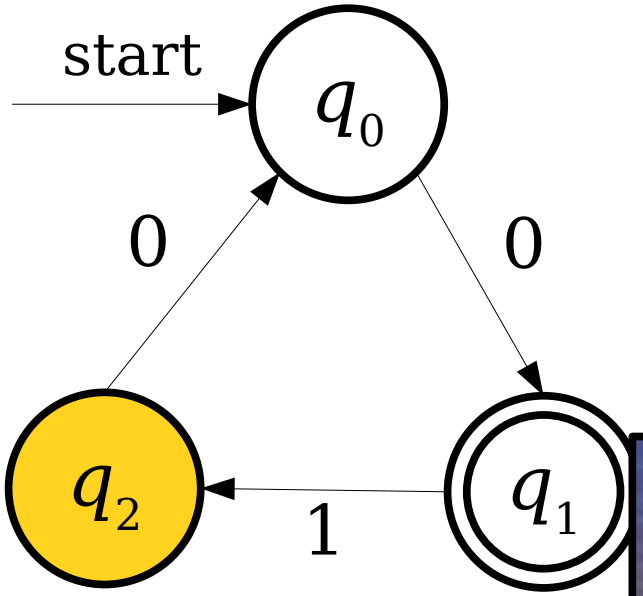
- A ***finite automaton*** is a collection of ***states*** joined by ***transitions***.
- Some state is designated as the ***start state***.
- Some number of states are designated as ***accepting states***.
- The automaton processes a string by beginning in the start state and following the indicated 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 set of strings it accepts.

Finite Automata

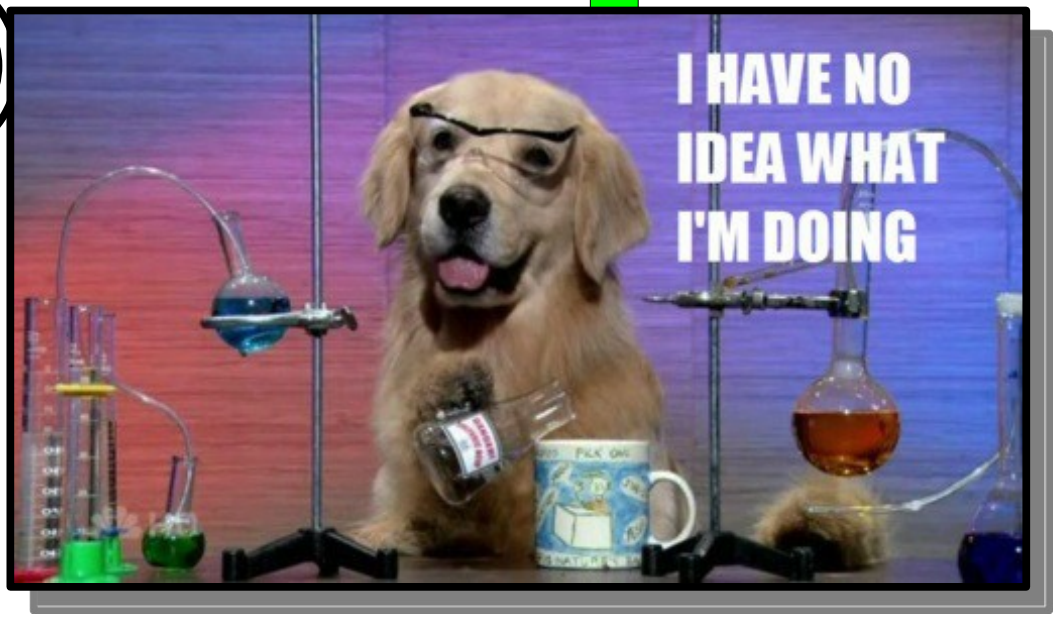
Part 1

1. Midterm Results
2. Computability Theory: The Central Question
3. Toward a Model of Computation
4. Automata
5. Languages, Strings, and Alphabets
6. **A Small Problem**
7. DFAs
8. Designing DFAs
9. What’s Next?

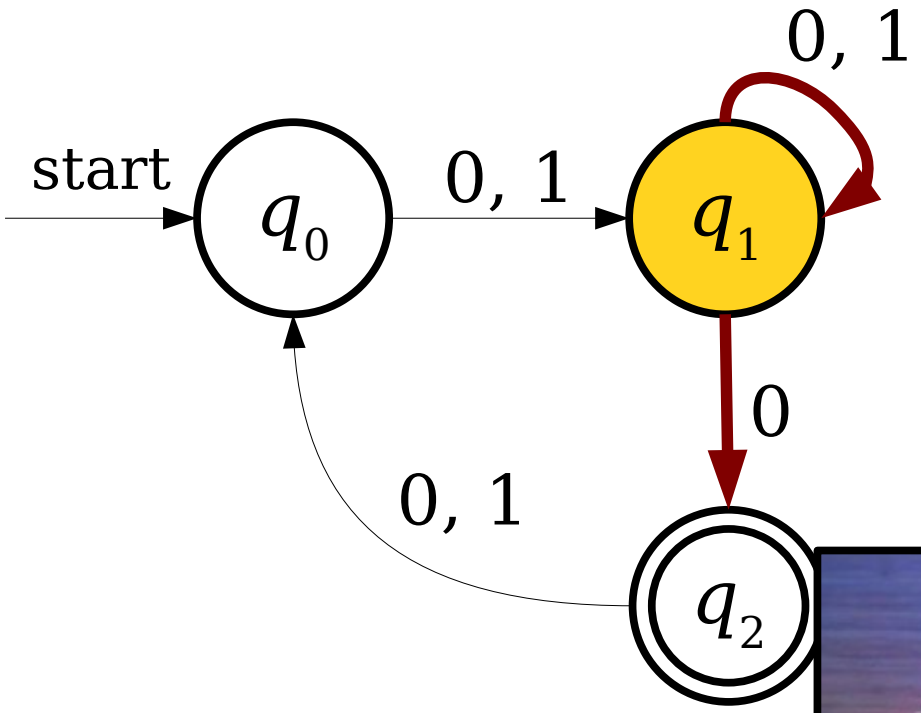
A Small Problem

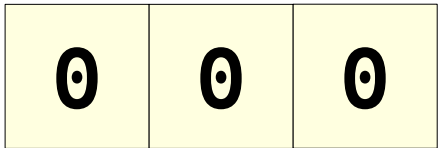


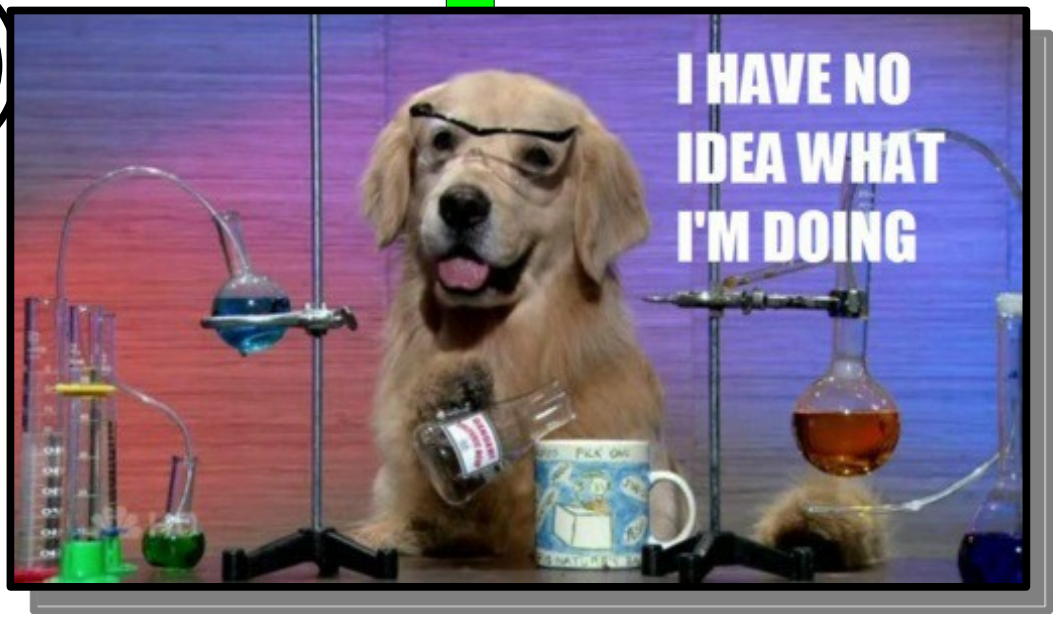
0	1	1	0



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?

Finite Automata

Part 1

1. Midterm Results
2. Computability Theory: The Central Question
3. Toward a Model of Computation
4. Automata
5. Languages, Strings, and Alphabets
6. A Small Problem
7. **DFAs**
8. Designing DFAs
9. What’s Next?

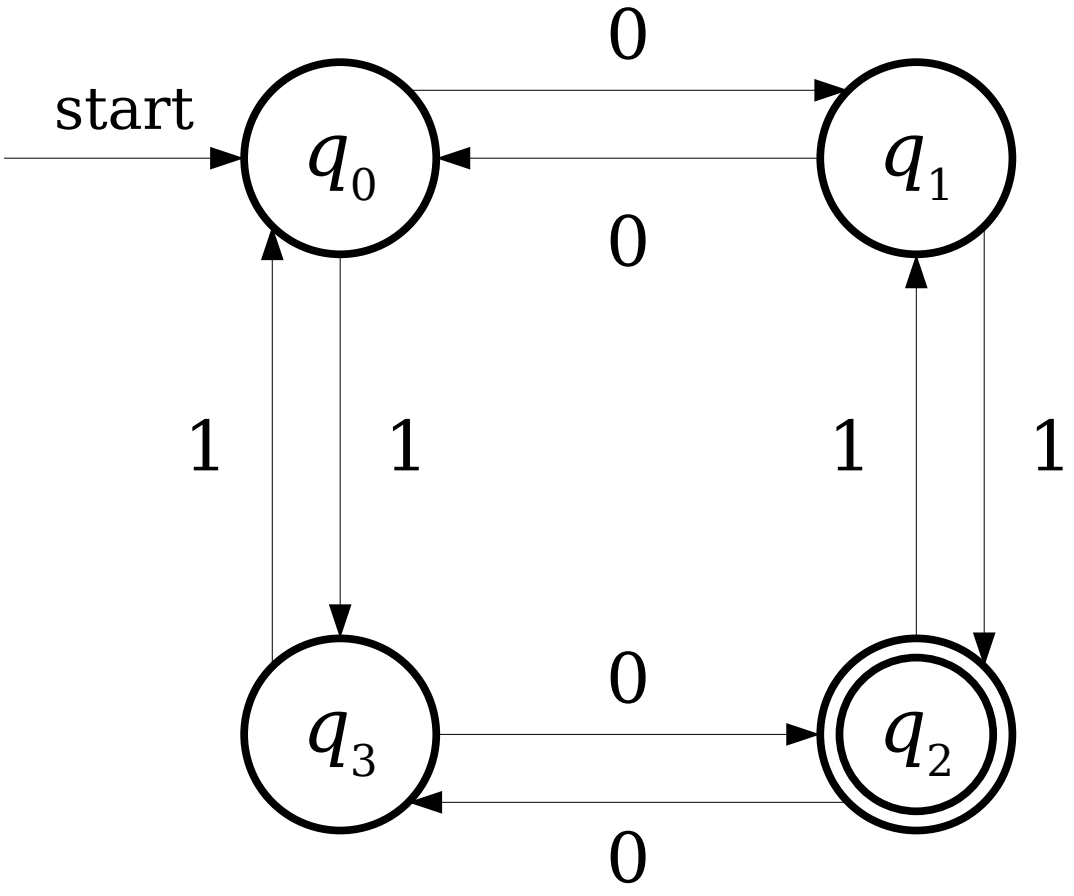
DFAs

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

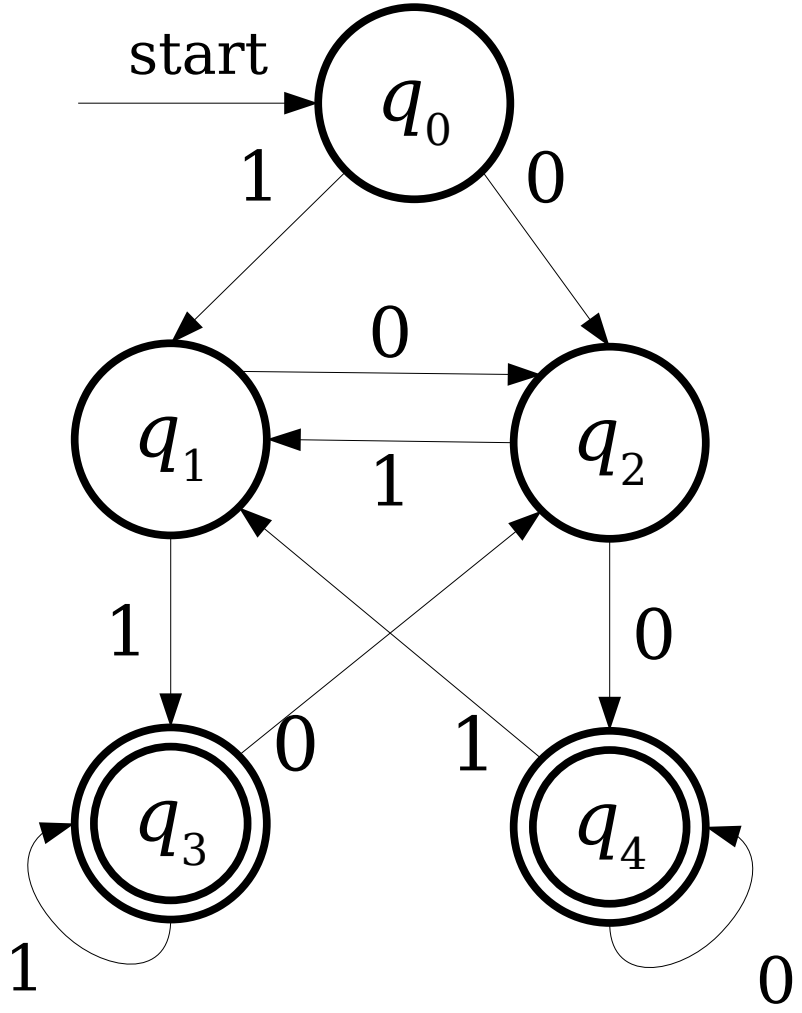
DFAs

- 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.

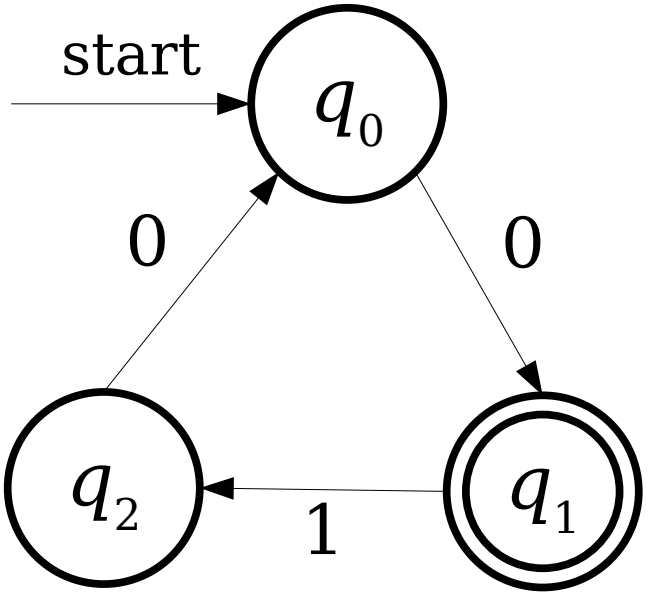
Is this a DFA over {0, 1}?



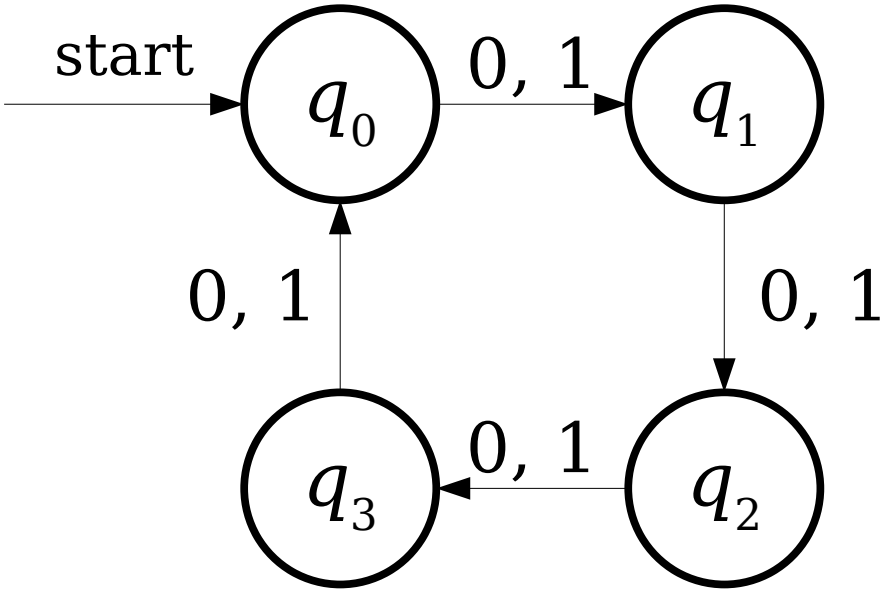
Is this a DFA over {0, 1}?



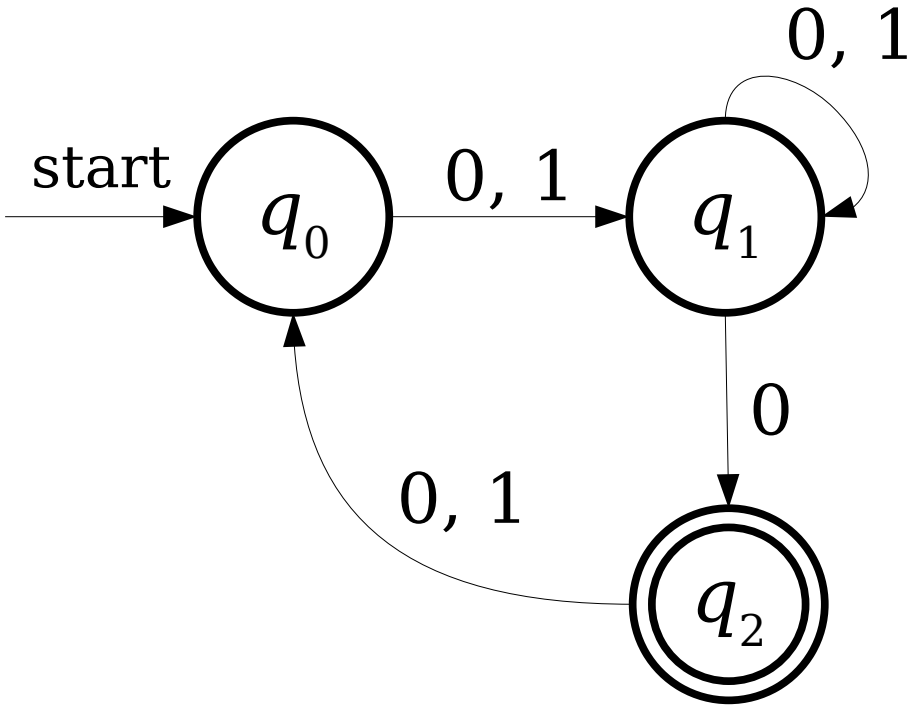
Is this a DFA over {0, 1}?



Is this a DFA over {0, 1}?



Is this a DFA over {0, 1}?



Finite Automata

Part 1

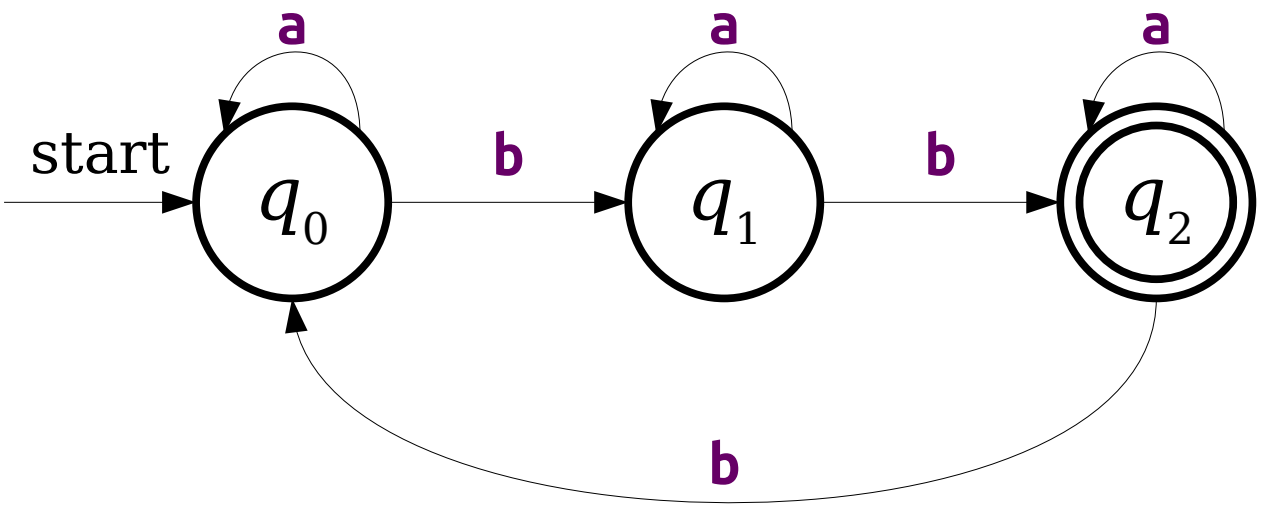
1. Midterm Results
2. Computability Theory: The Central Question
3. Toward a Model of Computation
4. Automata
5. Languages, Strings, and Alphabets
6. A Small Problem
7. DFAs
8. **Designing DFAs**
9. What's Next?

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

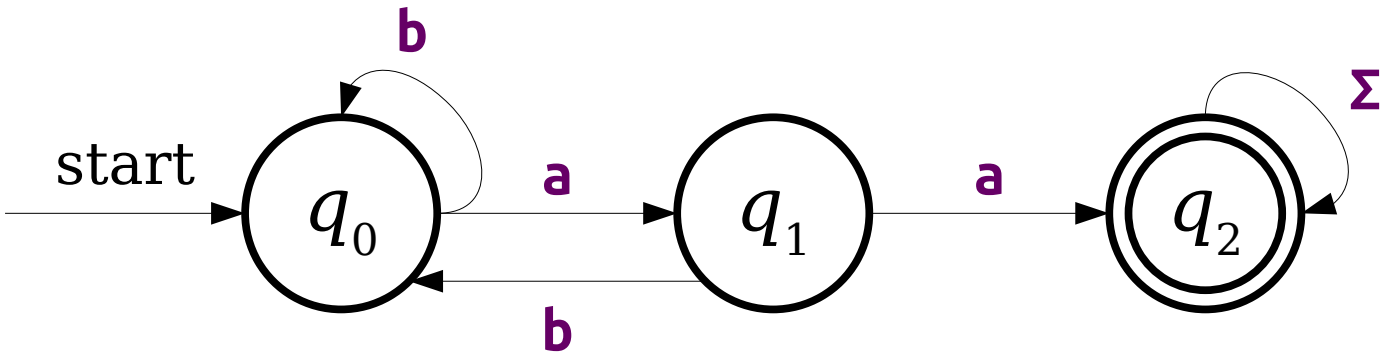
$L = \{ w \in \{\mathbf{a}, \mathbf{b}\}^* |$ the number of **b**'s in w is congruent to two modulo three $\}$



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

Recognizing Languages with DFAs

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



More Elaborate DFAs

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

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

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

Accepted:

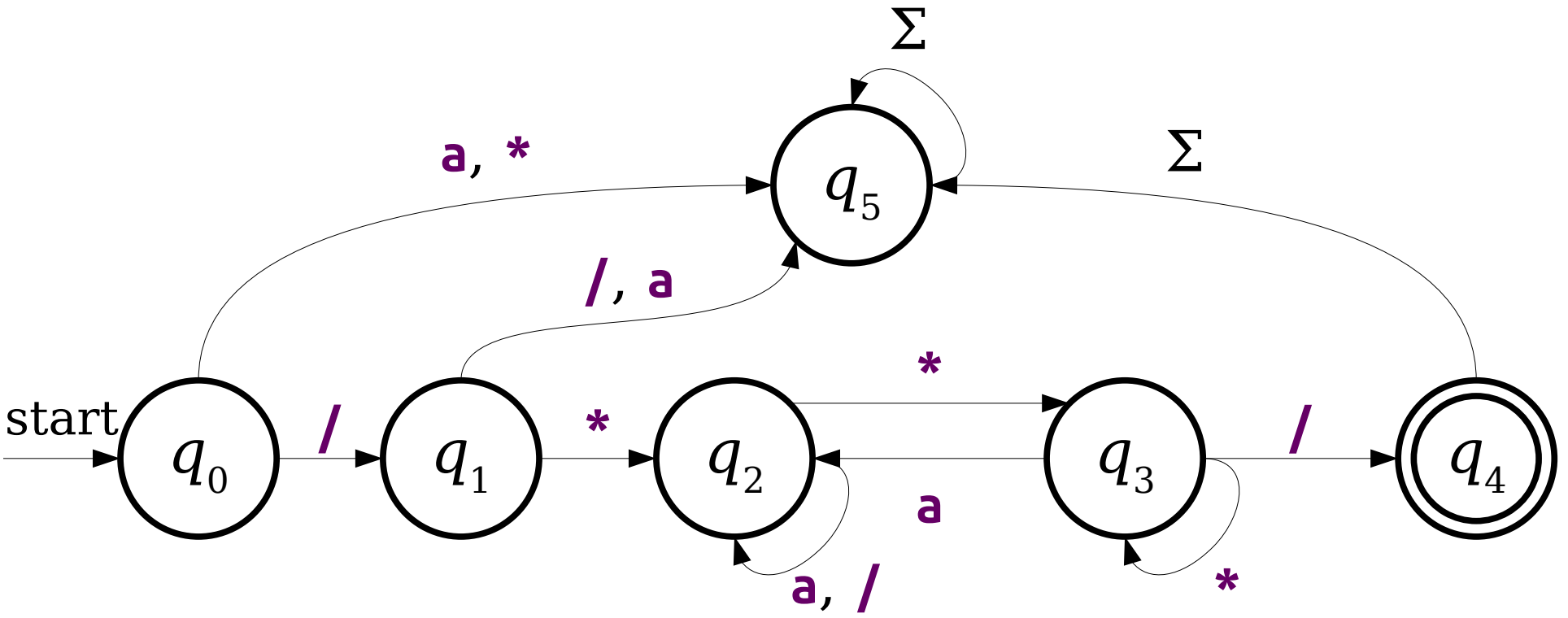
/*a*/
//**
/*/**
/*aaa*aaa*/
/*a/a*/

Rejected:

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

More Elaborate DFAs

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



Finite Automata

Part 1

1. Midterm Results
2. Computability Theory: The Central Question
3. Toward a Model of Computation
4. Automata
5. Languages, Strings, and Alphabets
6. A Small Problem
7. DFAs
8. Designing DFAs
9. **What's Next?**

Next Time

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