Nonregular Languages

Recap from Last Time

Theorem: The following are all equivalent:

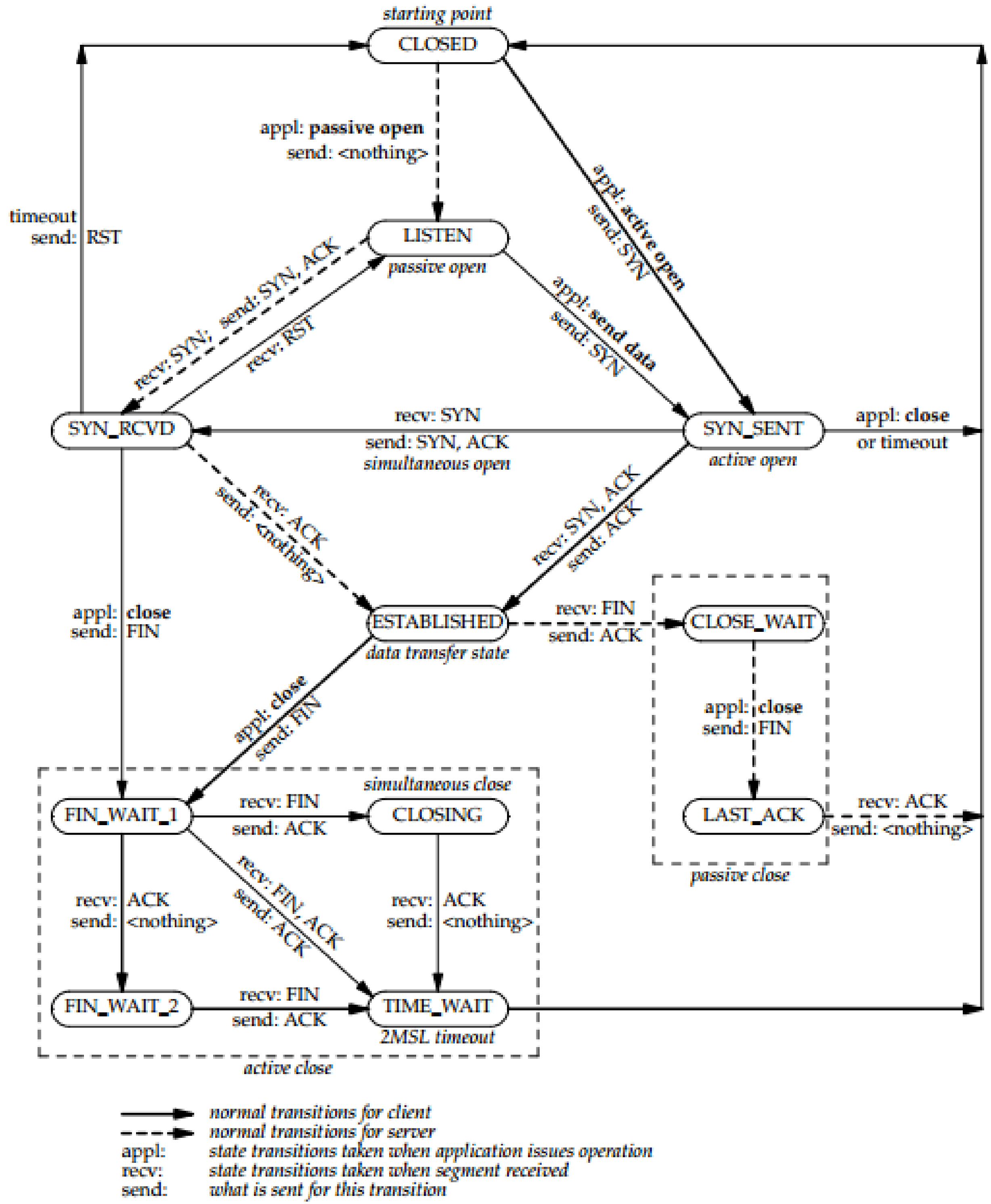
- L is a regular language.
- There is a DFA D such that $\mathcal{L}(D) = L$.
- There is an NFA N such that $\mathcal{L}(N) = L$.
- There is a regular expression R such that $\mathcal{L}(R) = L$.

New Stuff!

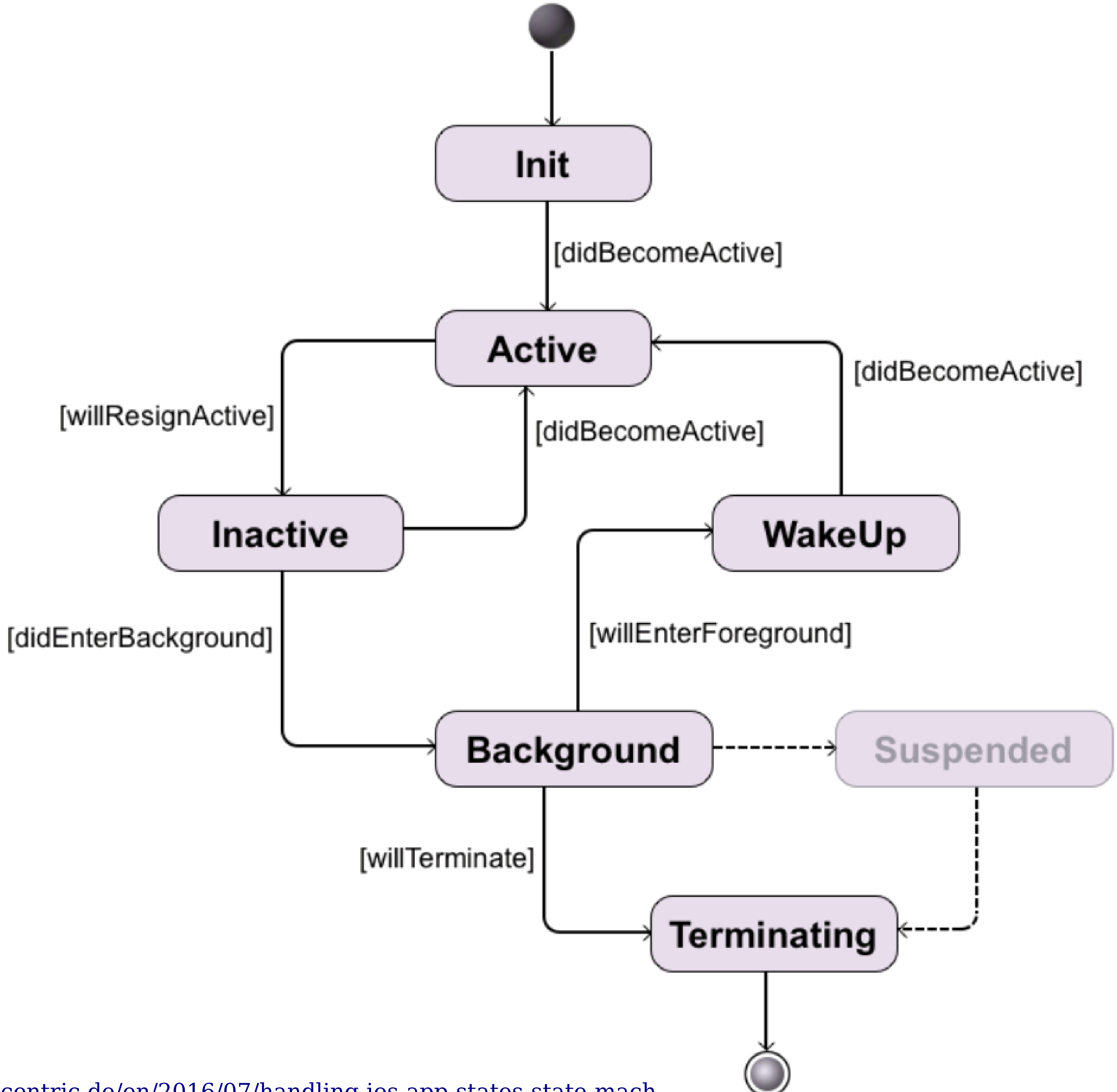
Why does this matter?

Buttons as Finite-State Machines:

http://cs103.stanford.edu/tools/button-fsm/

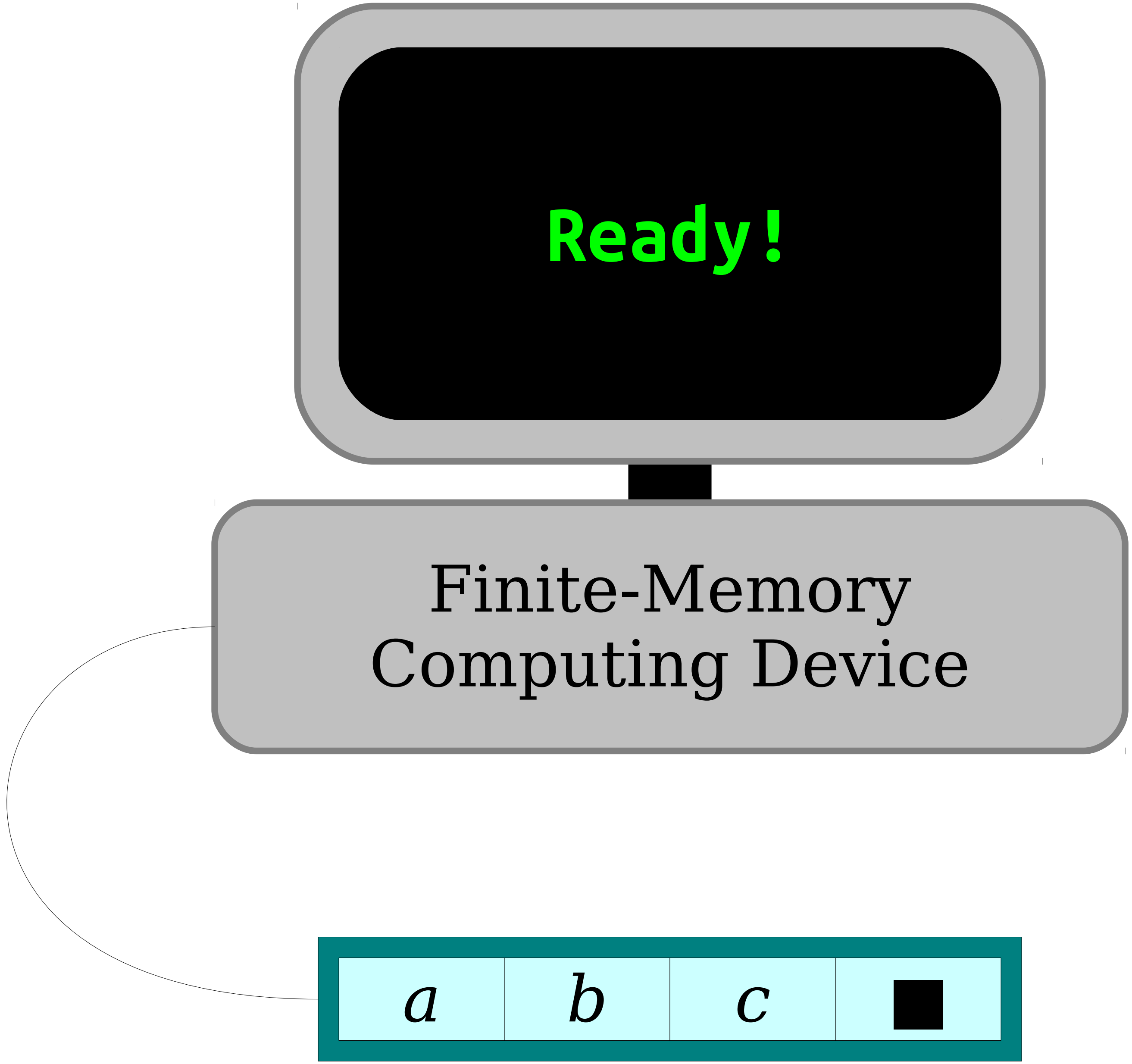


http://www.tti.unipa.it/~gneglia/ip_networks06/slides/TCPIP_State_Transition_Diagram.pdf

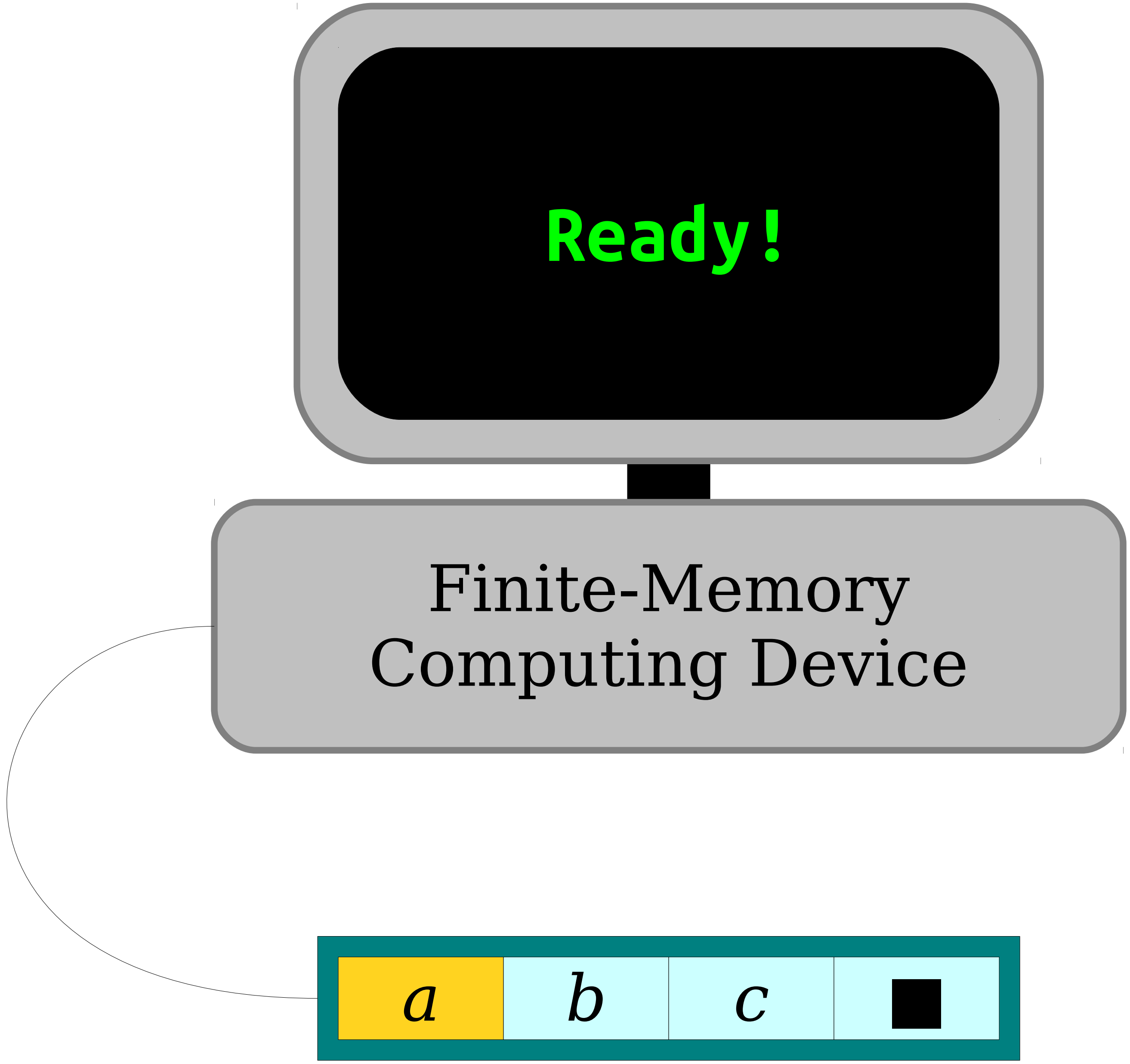


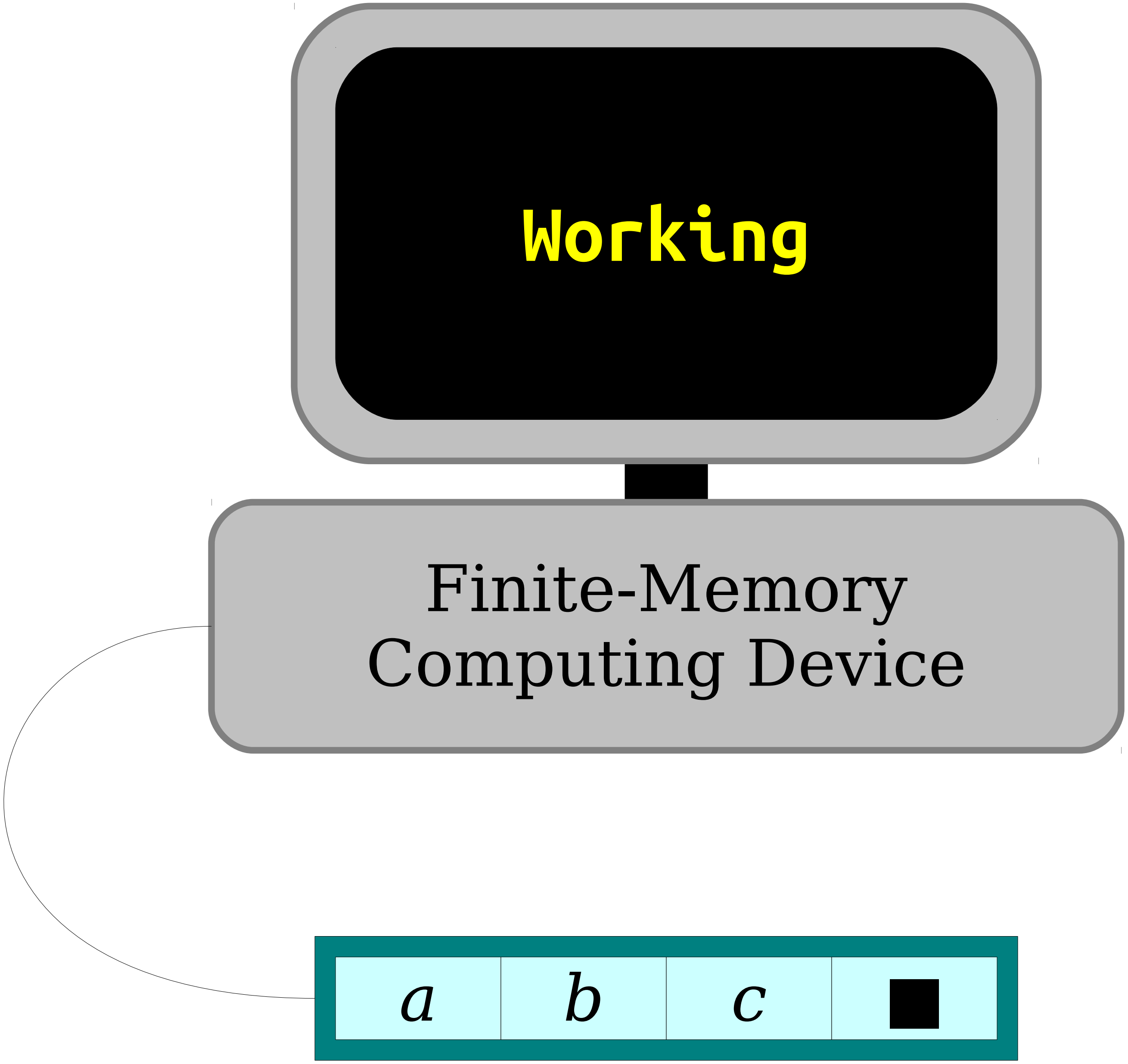
https://blog.codecentric.de/en/2016/07/handling-ios-app-states-state-mach

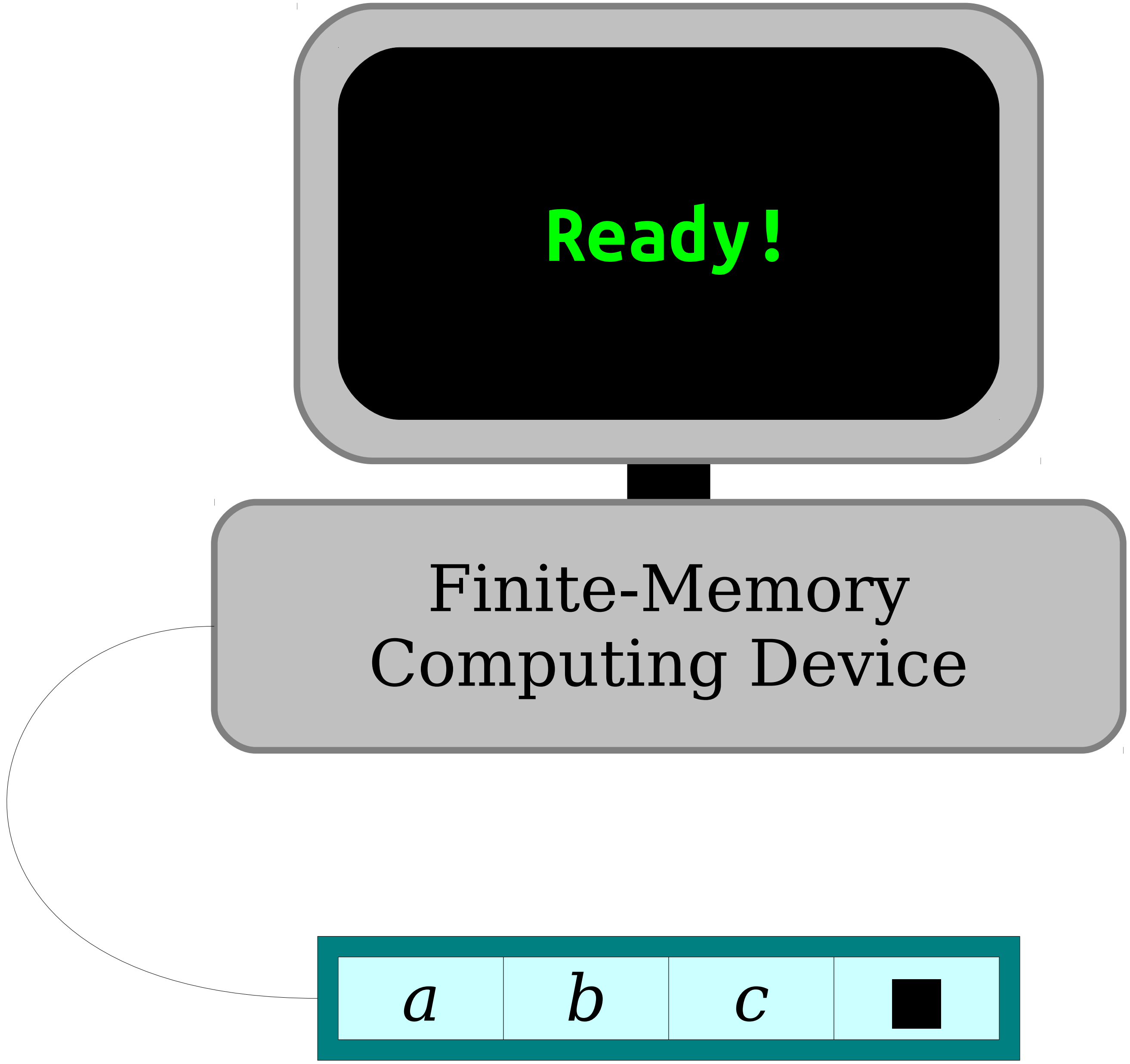
What exactly is a finite-state machine?

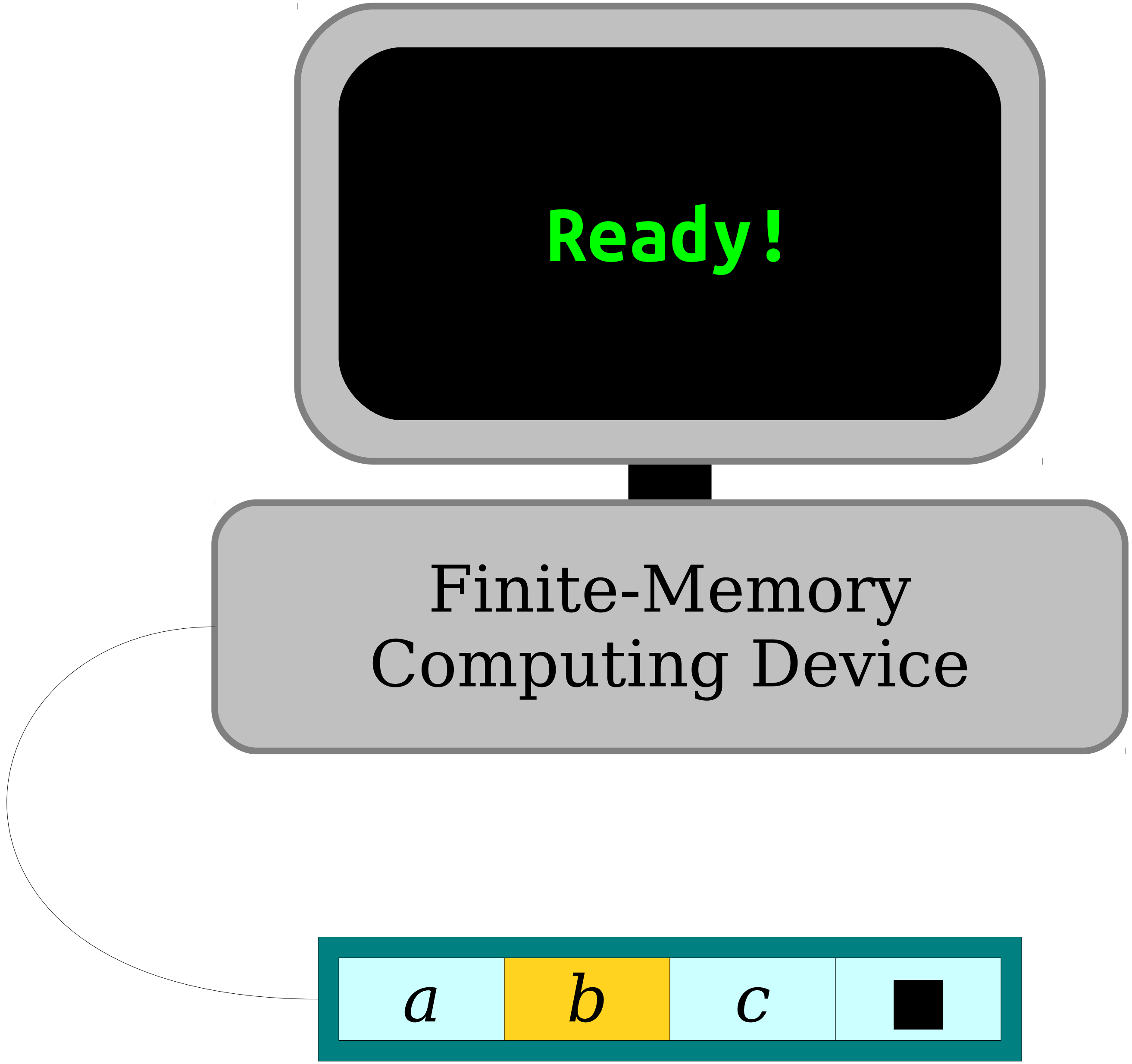


Ready!
Finite-Memory
Computing Device
a
b
c
■

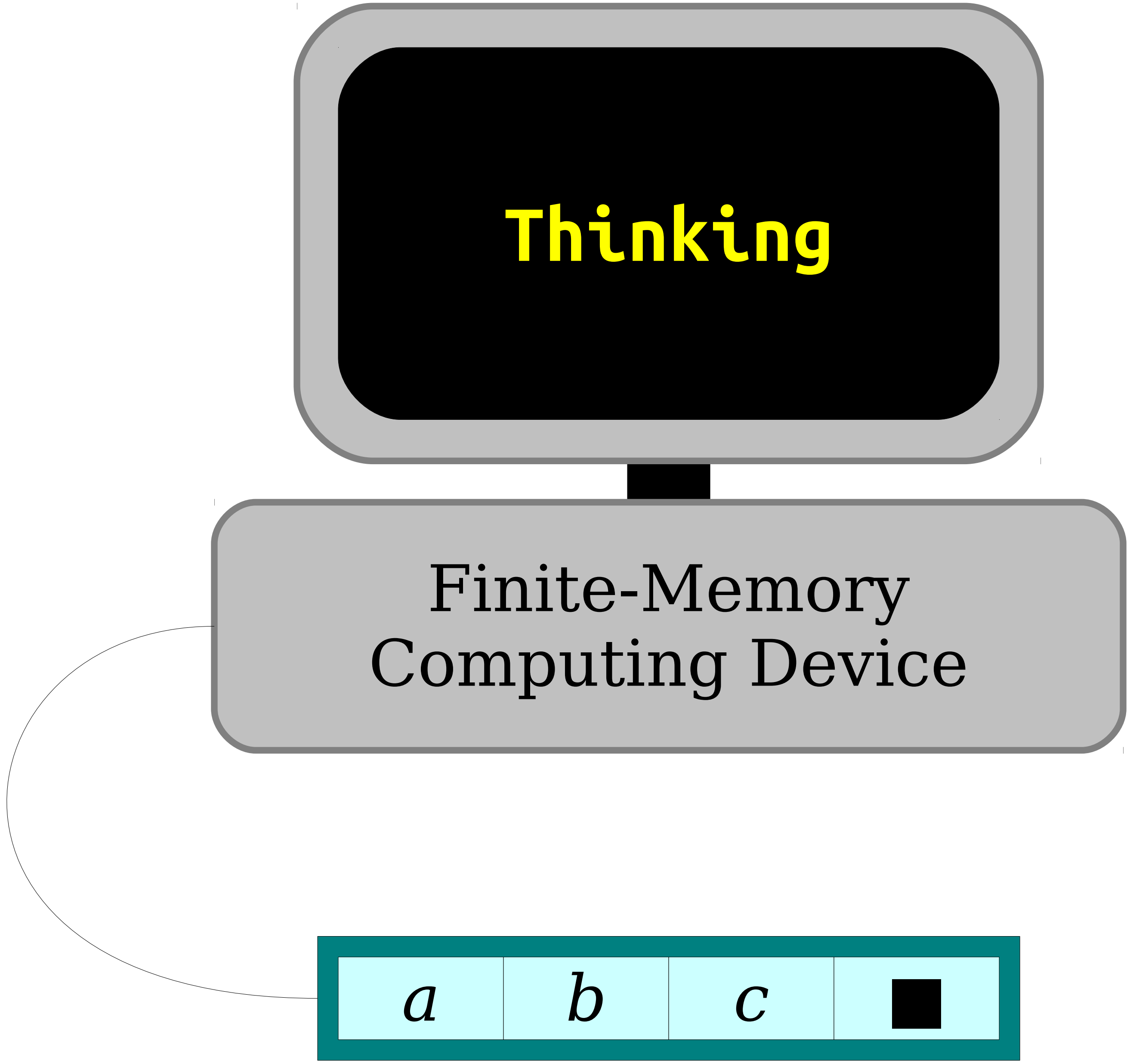

Ready!
Finite-Memory
Computing Device
a
b
c

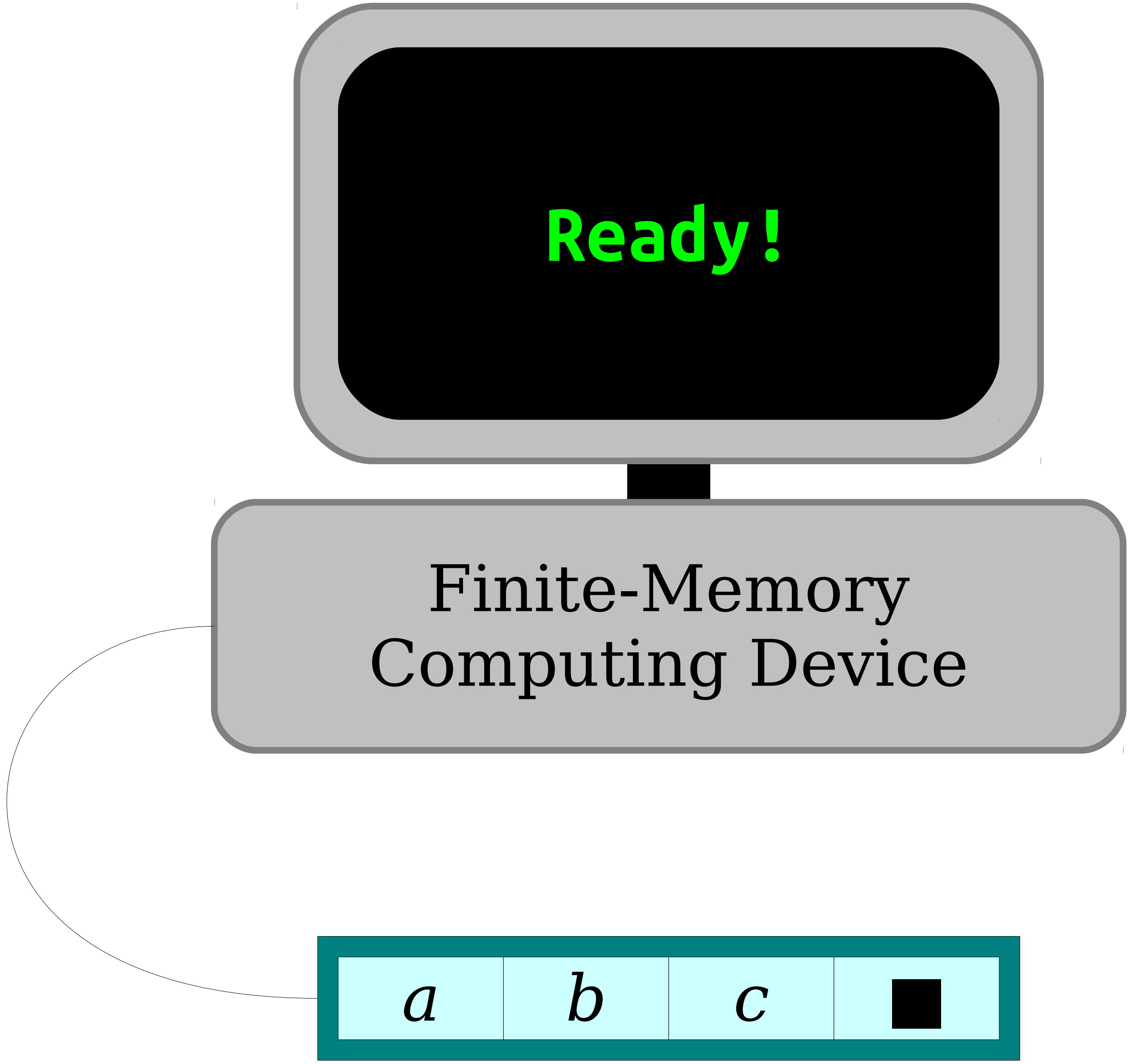

Working
Finite-Memory
Computing Device
a
b
c


Ready!
Finite-Memory
Computing Device
a
b
c

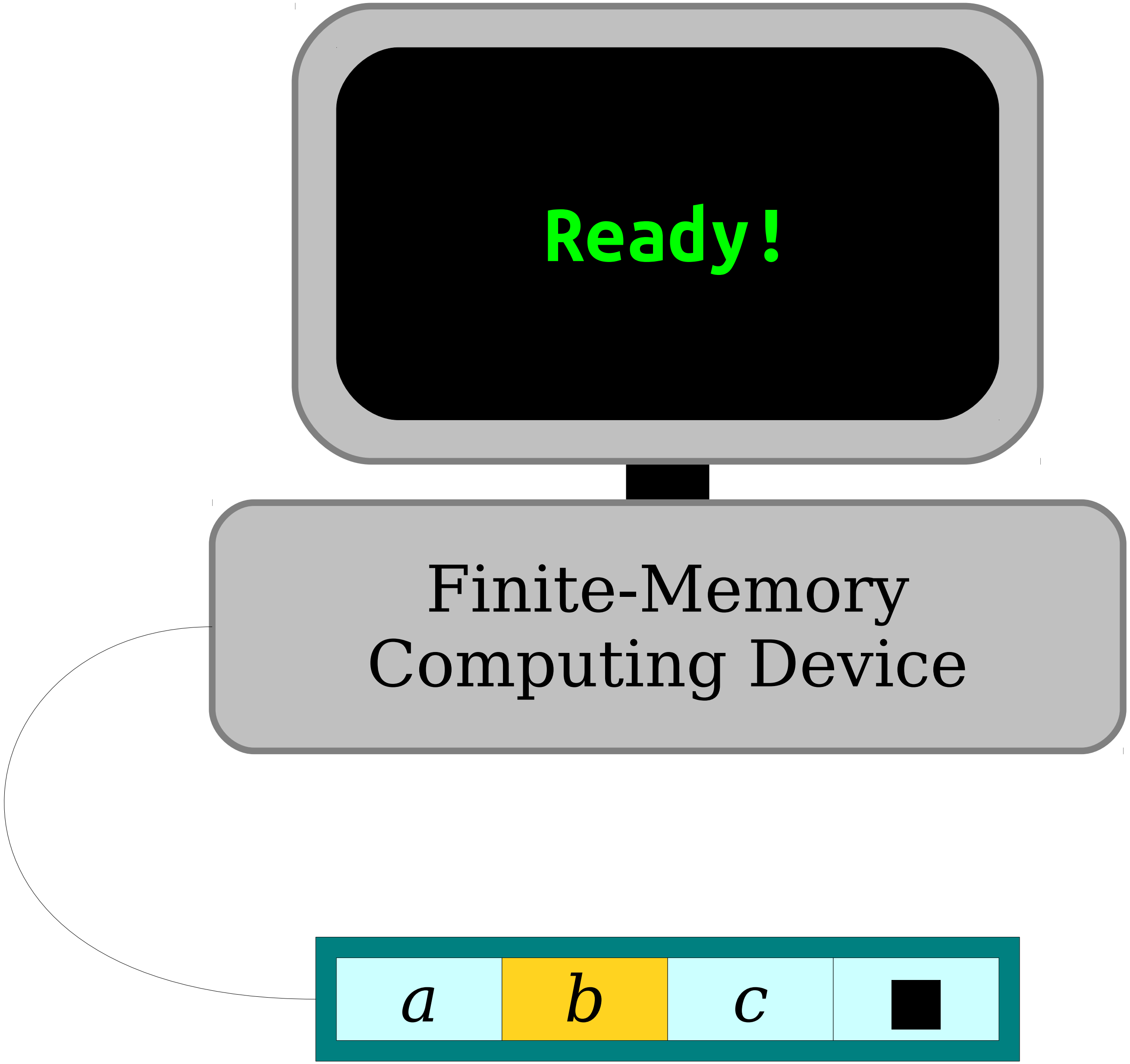


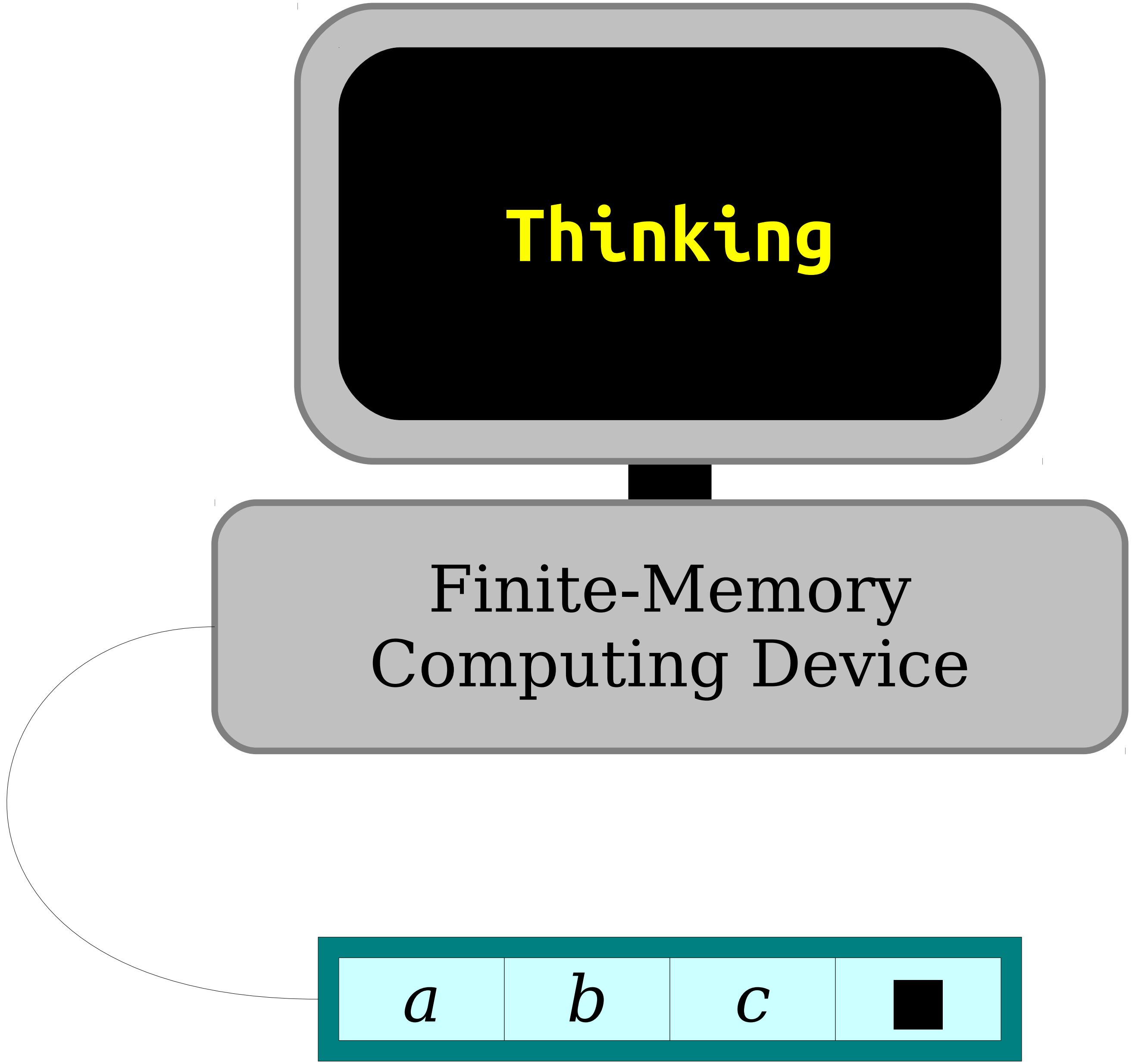
Ready!
Finite-Memory
Computing Device
a
b
c

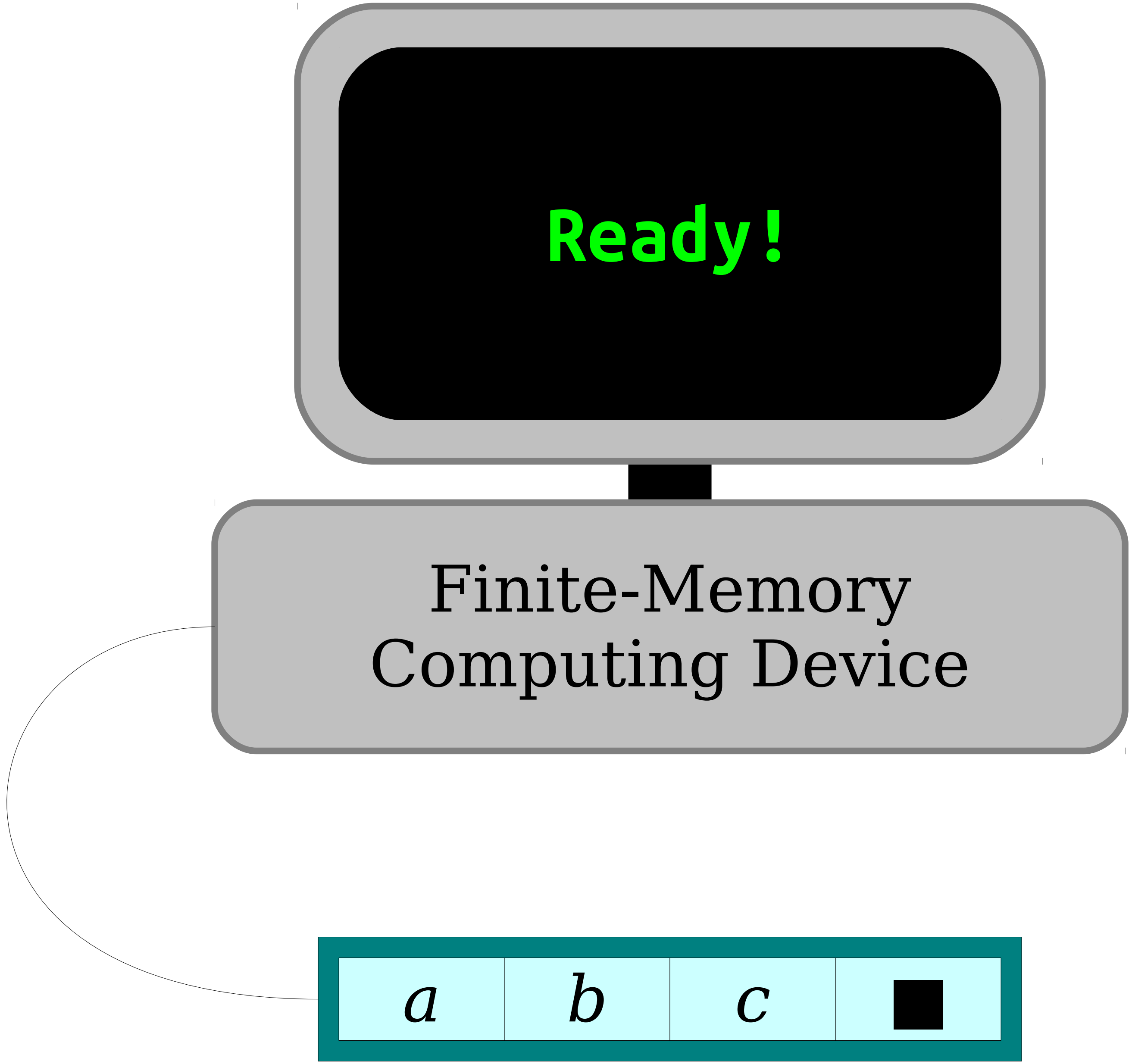

Thinking
Finite-Memory
Computing Device
a
b
c

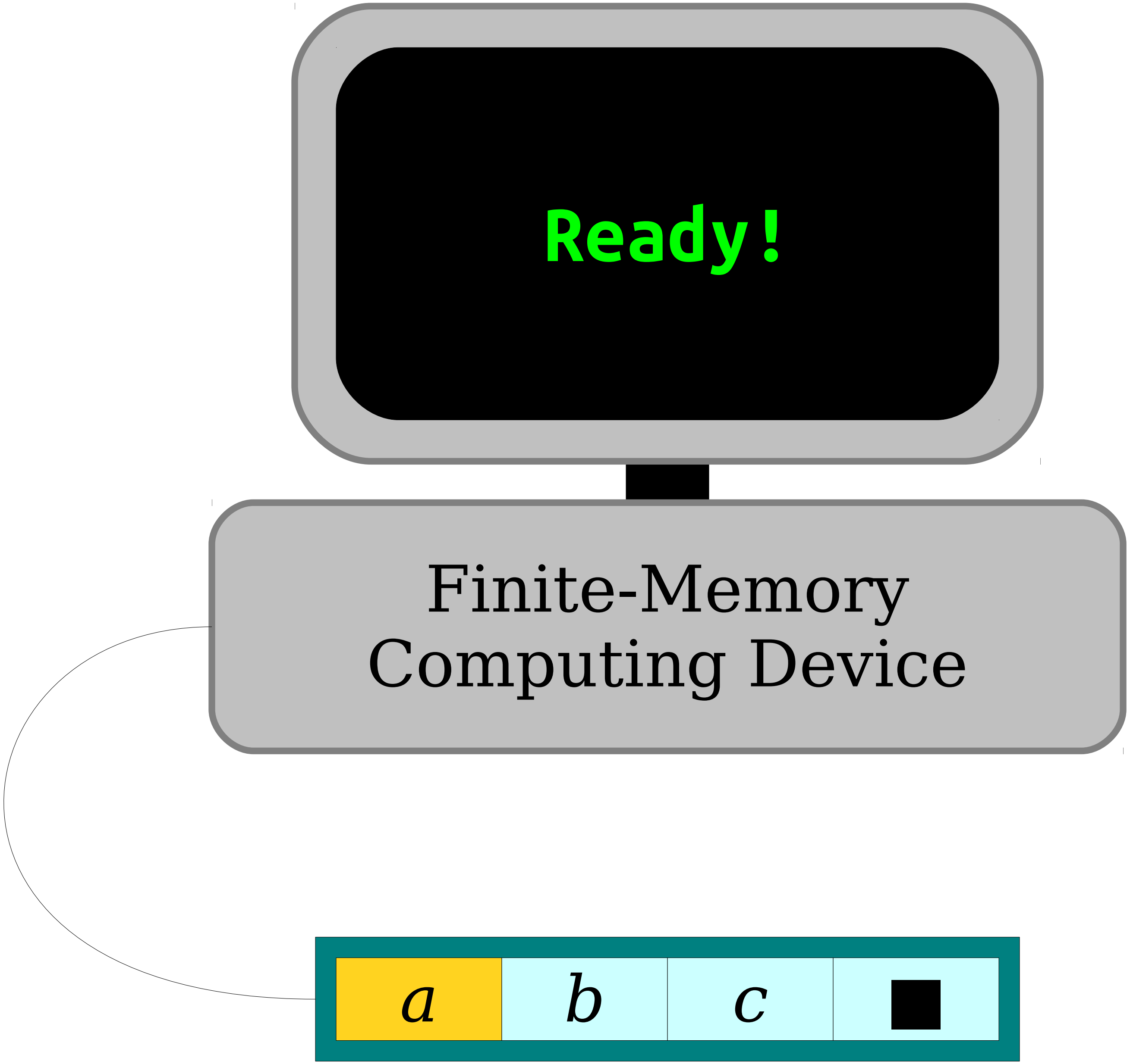


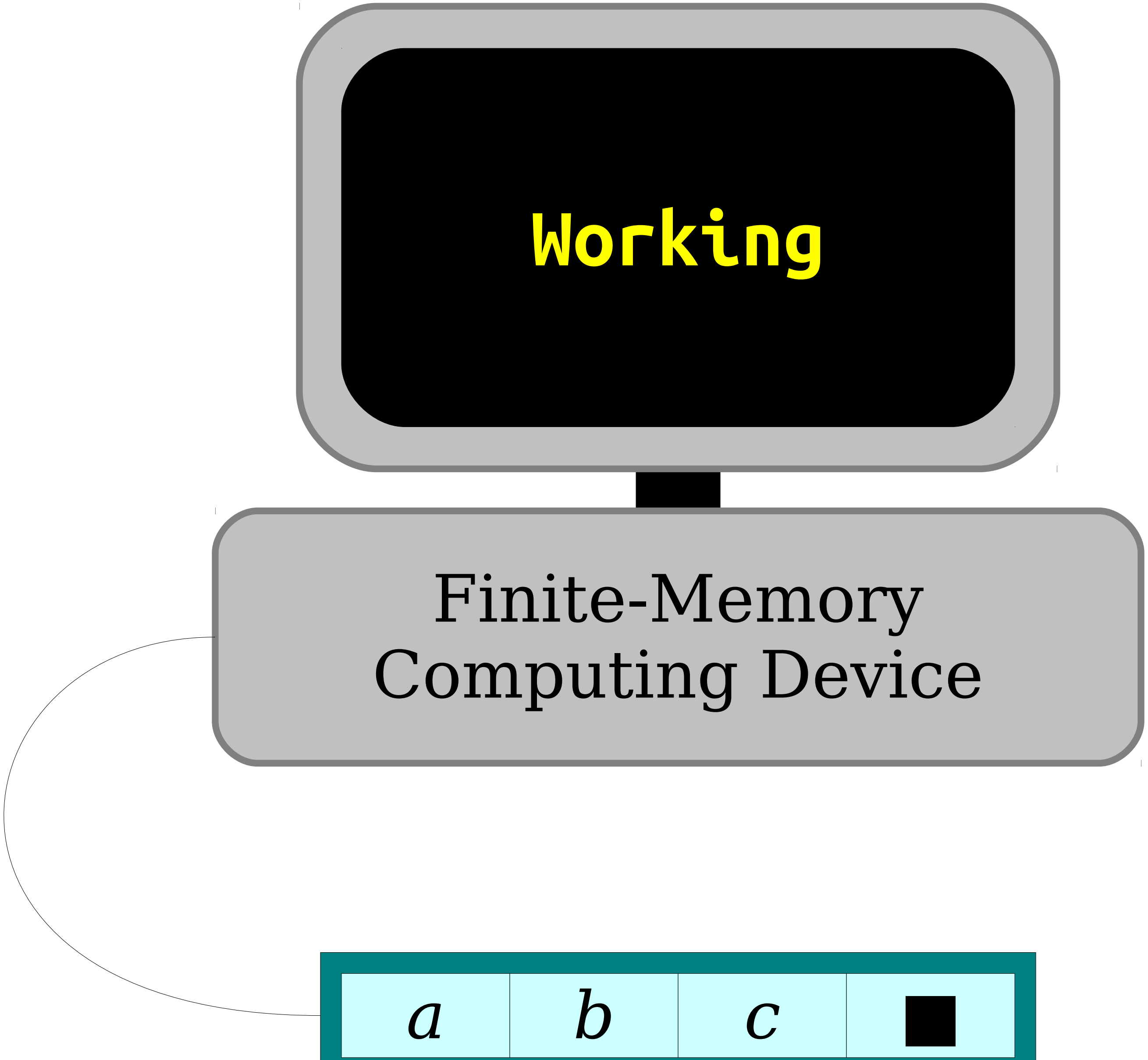
Ready!
Finite-Memory
Computing Device
a
b
c

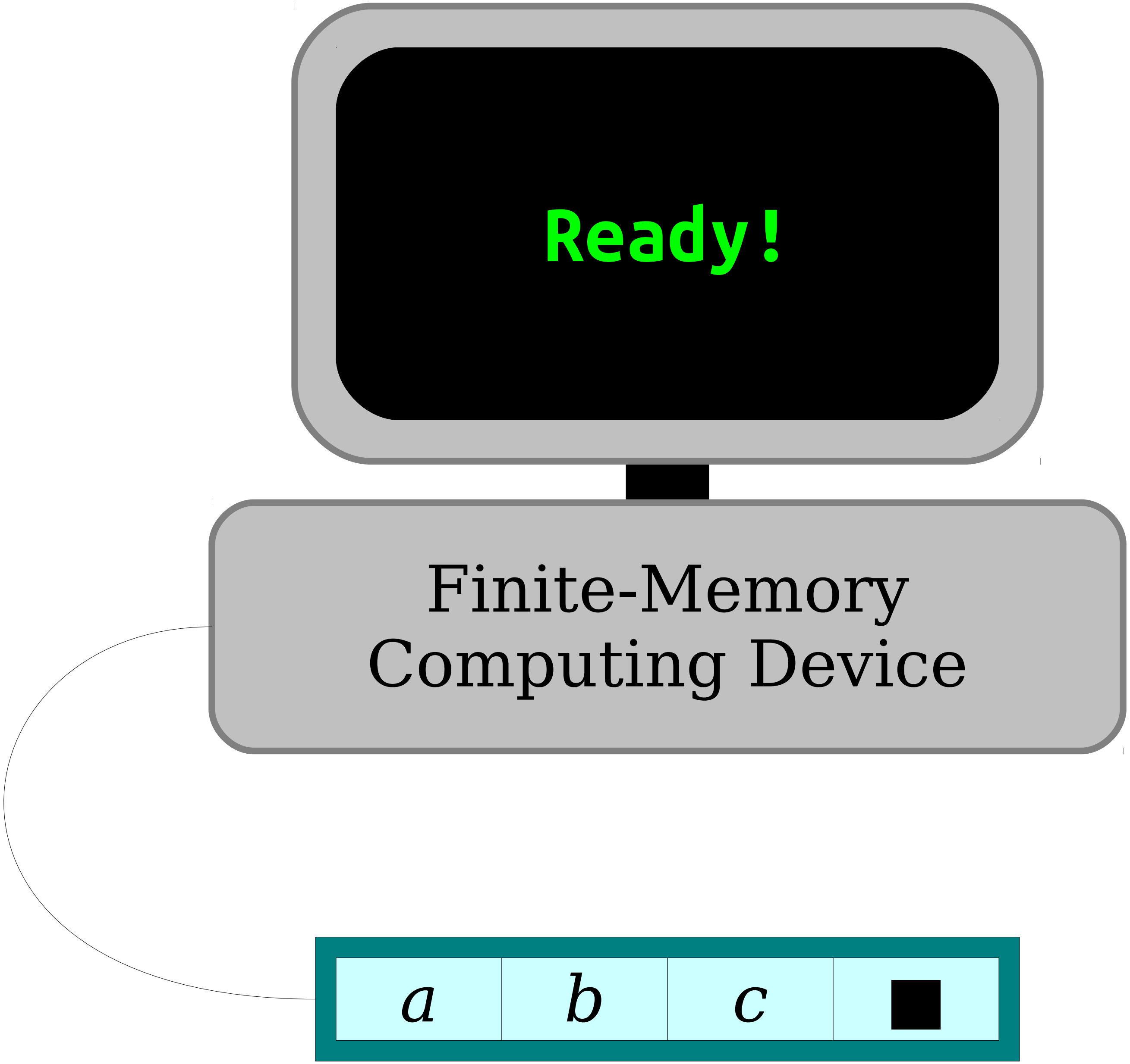

Ready!
Finite-Memory
Computing Device
a
b
c

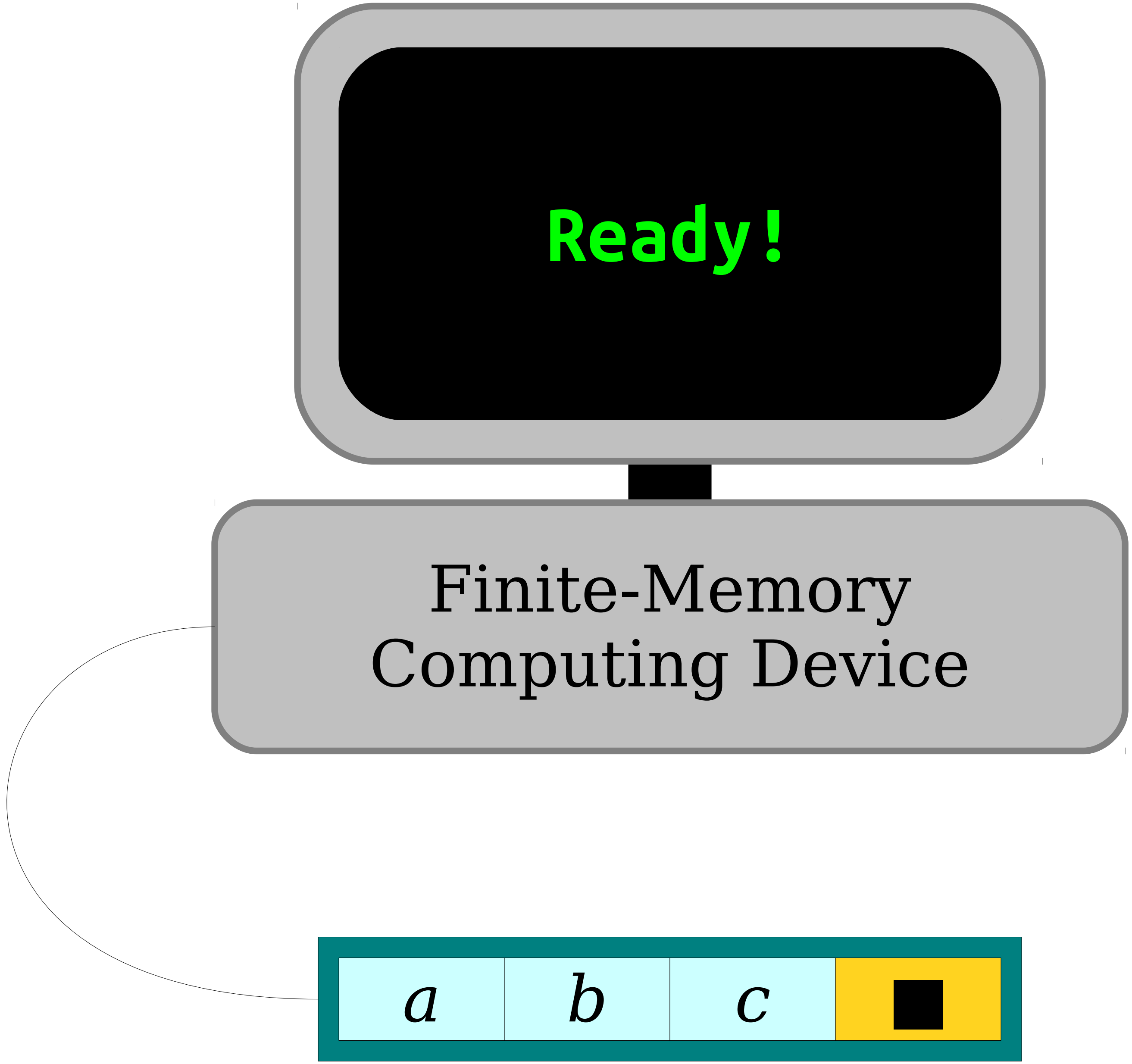

Thinking
Finite-Memory
Computing Device
a
b
c

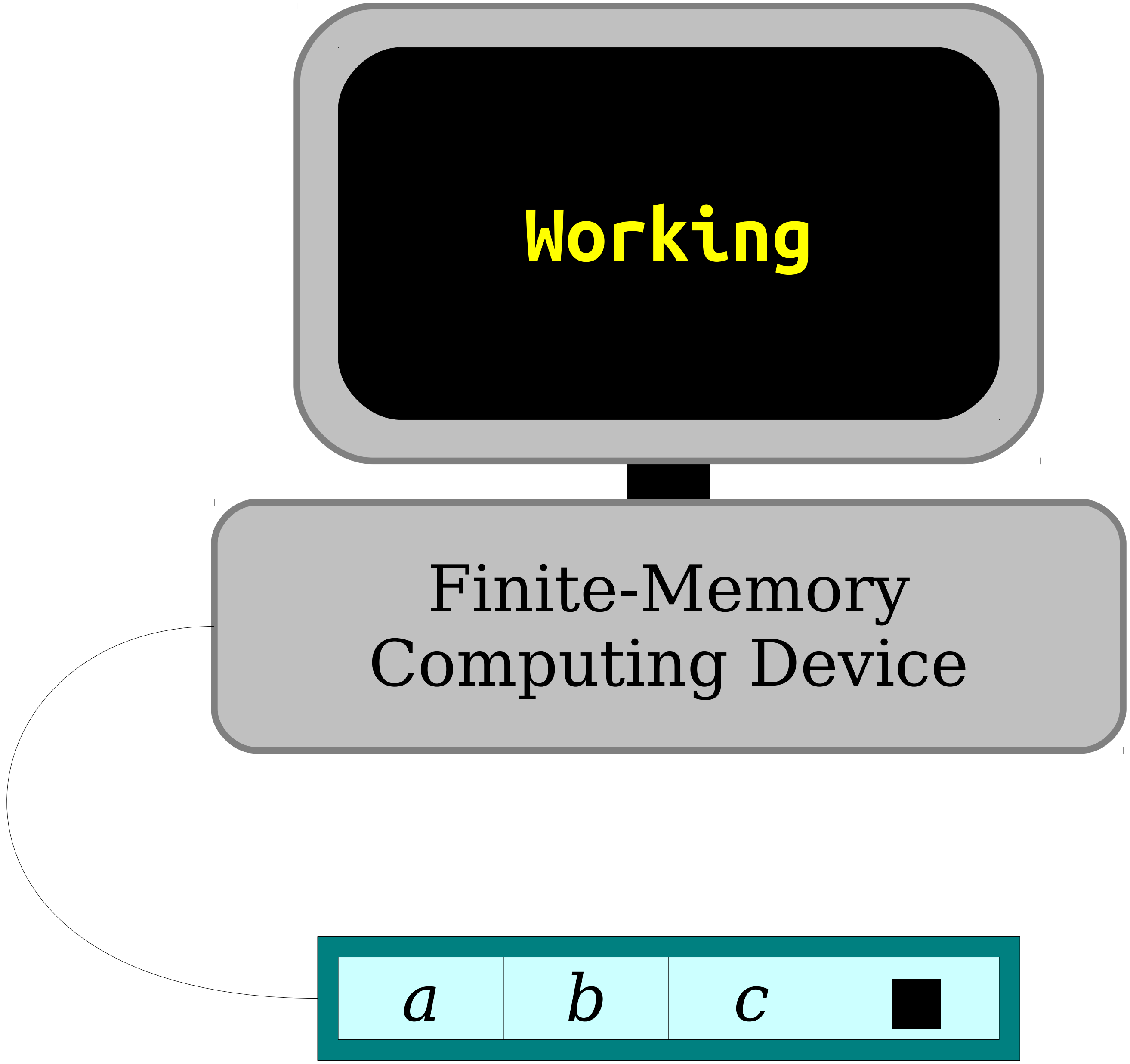

Ready!
Finite-Memory
Computing Device
a
b
c
■


Ready!
Finite-Memory
Computing Device
a
b
c

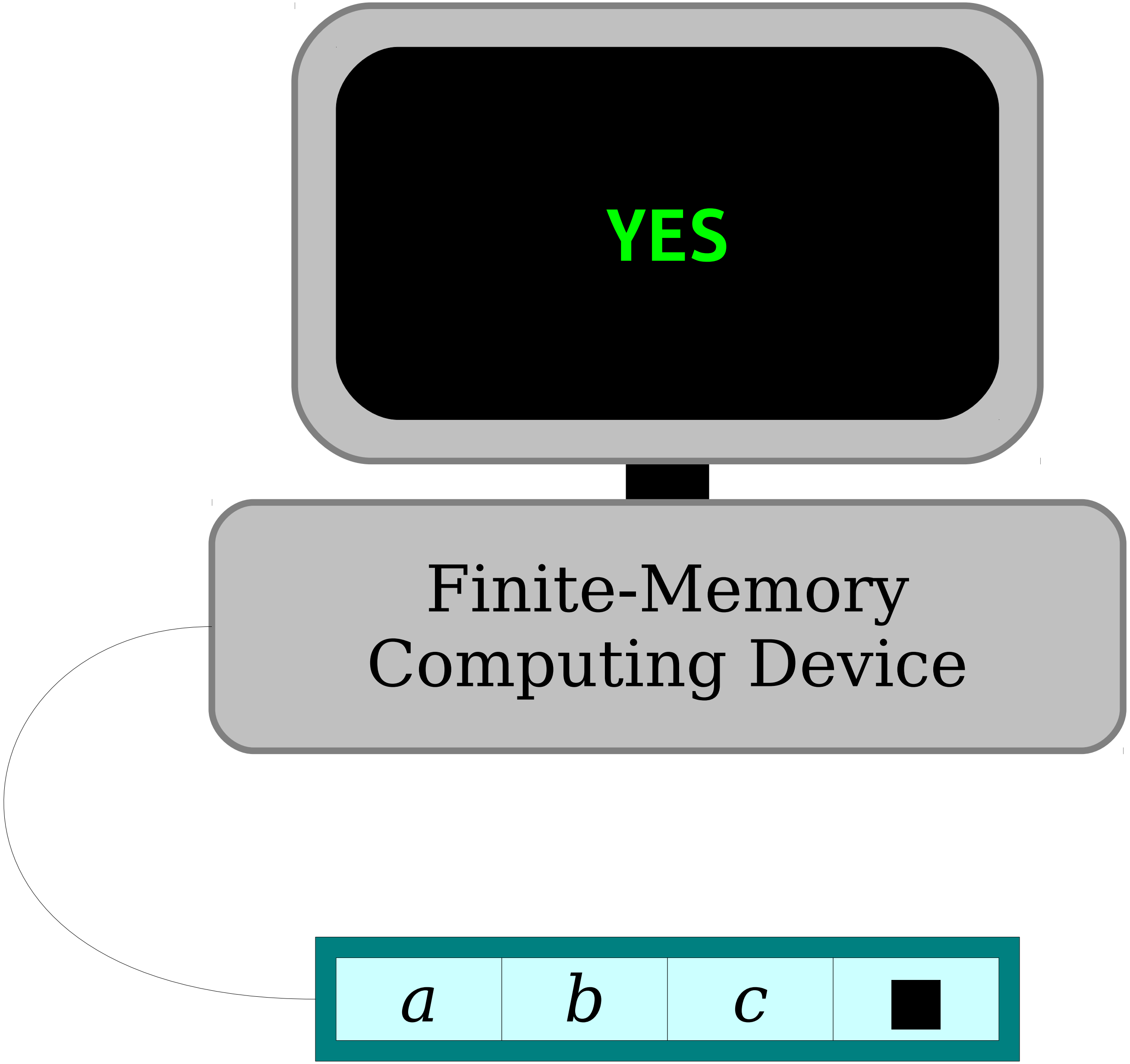

Working
Finite-Memory
Computing Device
a
b
c


Ready!
Finite-Memory
Computing Device
a
b
c


Ready!
Finite-Memory
Computing Device
a
b
c



Working
Finite-Memory Computing Device
a
b
c


YES
Finite-Memory
Computing Device
a
b
c

The Model

- The computing device has internal workings that can be in one of finitely many possible configurations.
  - Each ***state*** in a DFA corresponds to some possible configuration of the internal workings.
- After each button press, the computing device does some amount of processing, then gets to a configuration where it's ready to receive more input.
  - Each ***transition*** abstracts away how the computation is done and just indicates what the ultimate configuration looks like.
- After the user presses the “done” button, the computer outputs either YES or NO.
  - The ***accepting*** and ***rejecting*** states of the machine model what happens when that button is pressed.

Computers as Finite Automata

- My computer has 12GB of RAM and about 150GB of hard disk space.
- That's a total of 162GB of memory, which is 1,391,569,403,904 bits.
- There are “only” $2^{1,391,569,403,904}$ possible configurations of the memory in my computer.
- You could in principle build a DFA representing my computer, where there's one symbol per type of input the computer can receive.

A Powerful Intuition

- ***Regular languages correspond to problems that can be solved with finite memory.***
  - At each point in time, we only need to store one of finitely many pieces of information.
- Nonregular languages, in a sense, correspond to problems that cannot be solved with finite memory.
- Since every computer ever built has finite memory, in a sense, nonregular languages correspond to problems that cannot be solved by physical computers!

Finding Nonregular Languages

Finding Nonregular Languages

- To prove that a language is regular, we can just find a DFA, NFA, or regex for it.
- To prove that a language is not regular, we need to prove that there are no possible DFAs, NFAs, or regexes for it.
  - ***Claim:*** We can actually just prove that there's no DFA for it. Why is this?
- ***This sort of argument will be challenging.*** Our arguments will be somewhat technical in nature, since we need to rigorously establish that no amount of creativity could produce a DFA for a given language.
- Let's see an example of how to do this.

A Simple Language

- Let $\Sigma = \{\mathbf{a}, \mathbf{b}\}$ and consider the following language:

$$E = \{ \mathbf{a}^n \mathbf{b}^n \mid n \in \mathbb{N} \}$$

- E is the language of all strings of n **a**'s followed by n **b**'s:

$$\{ \varepsilon, \mathbf{ab}, \mathbf{aabb}, \mathbf{aaabbb}, \mathbf{aaaabbbb}, \ldots \}$$

A Simple Language

$$E = \{ \mathbf{a}^n \mathbf{b}^n \mid n \in \mathbb{N} \}$$

How many of the following are regular expressions for the language E defined above?

a*b*

(ab)*

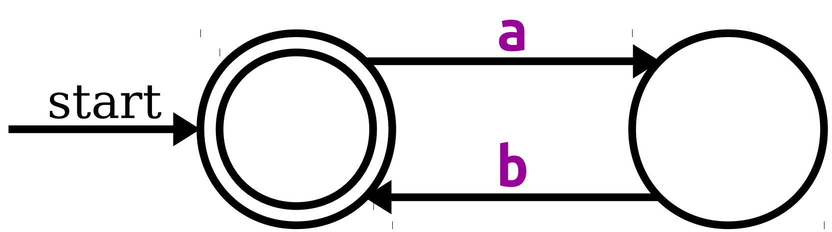
$\boldsymbol{\varepsilon} \cup \mathbf{ab} \cup \mathbf{a}^2\mathbf{b}^2 \cup \mathbf{a}^3\mathbf{b}^3$

Another Attempt

- Let’s try to design an NFA for

$$E = \{ \mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N} \}.$$

- Does this machine work?

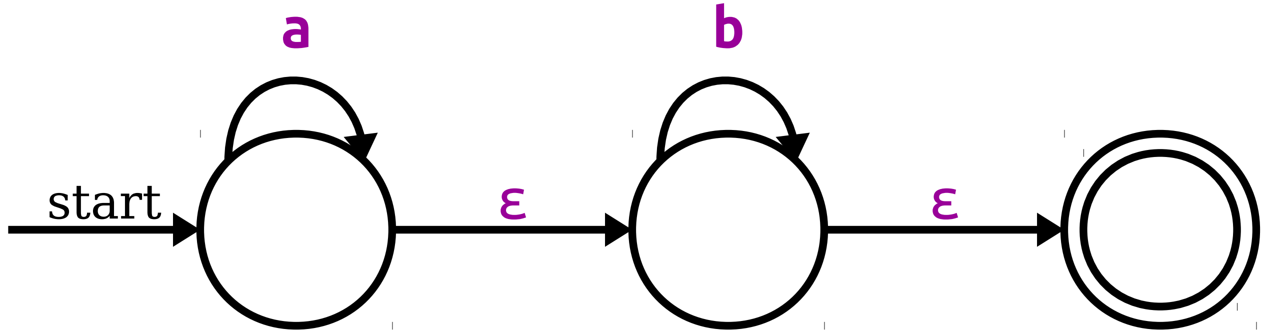


Another Attempt

- Let's try to design an NFA for

$$E = \{ \mathtt{a}^n \mathtt{b}^n \mid n \in \mathbb{N} \}.$$

- How about this one?

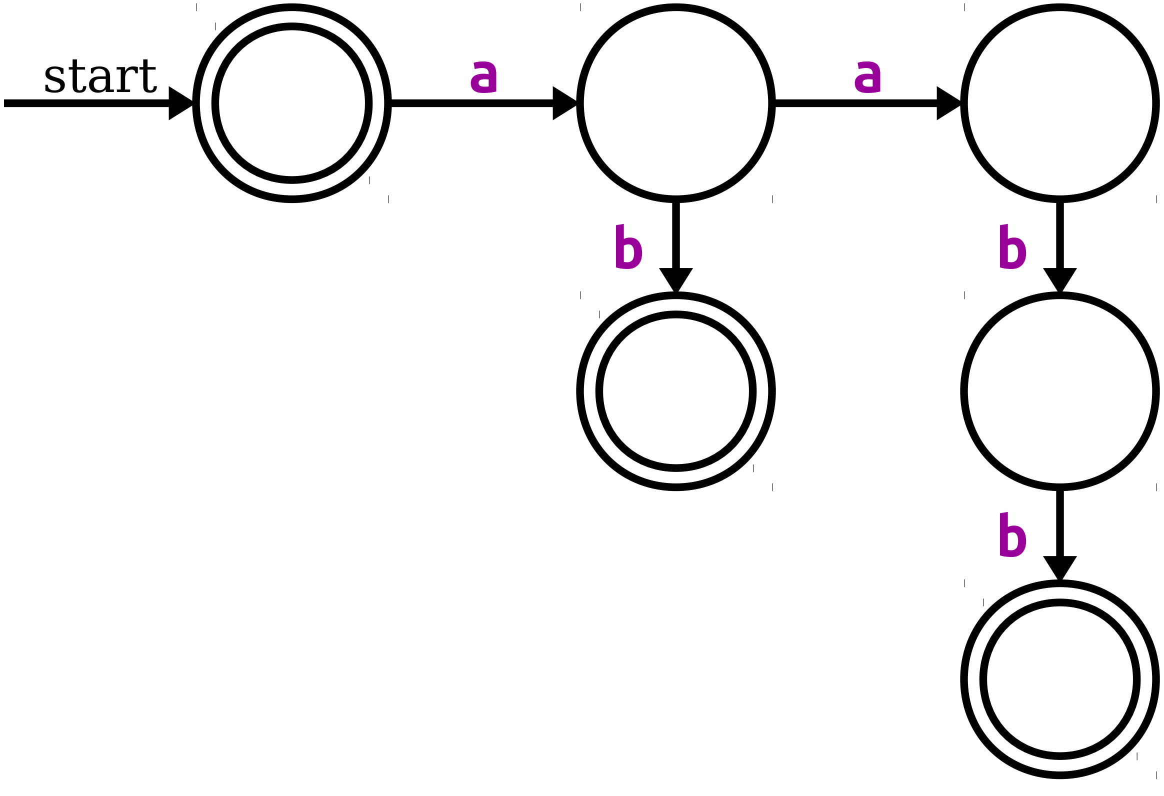


Another Attempt

- Let's try to design an NFA for

$$E = \{ \mathtt{a}^n \mathtt{b}^n \mid n \in \mathbb{N} \}.$$

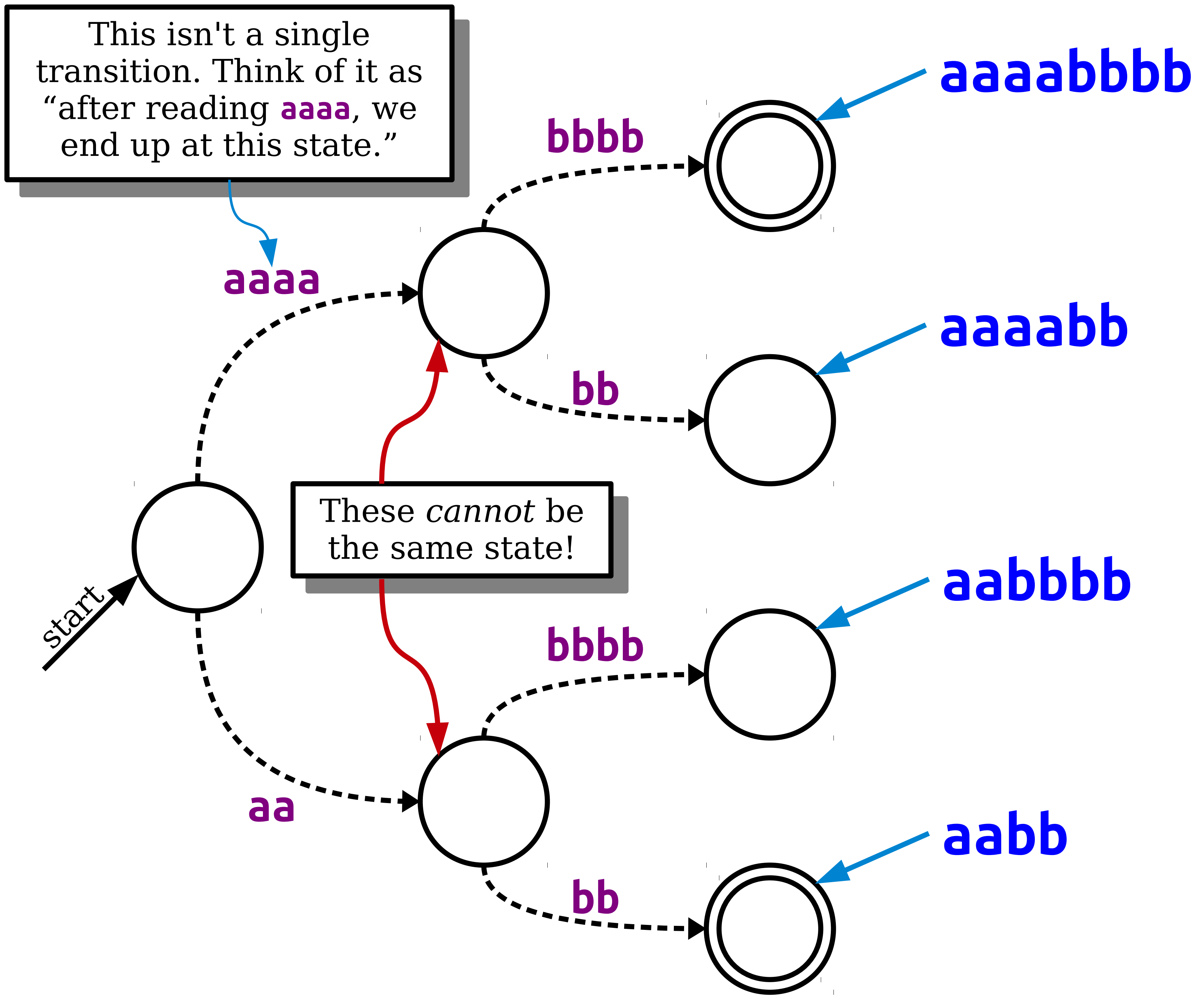
- What about this?



We seem to be running into some trouble.
Why is that?

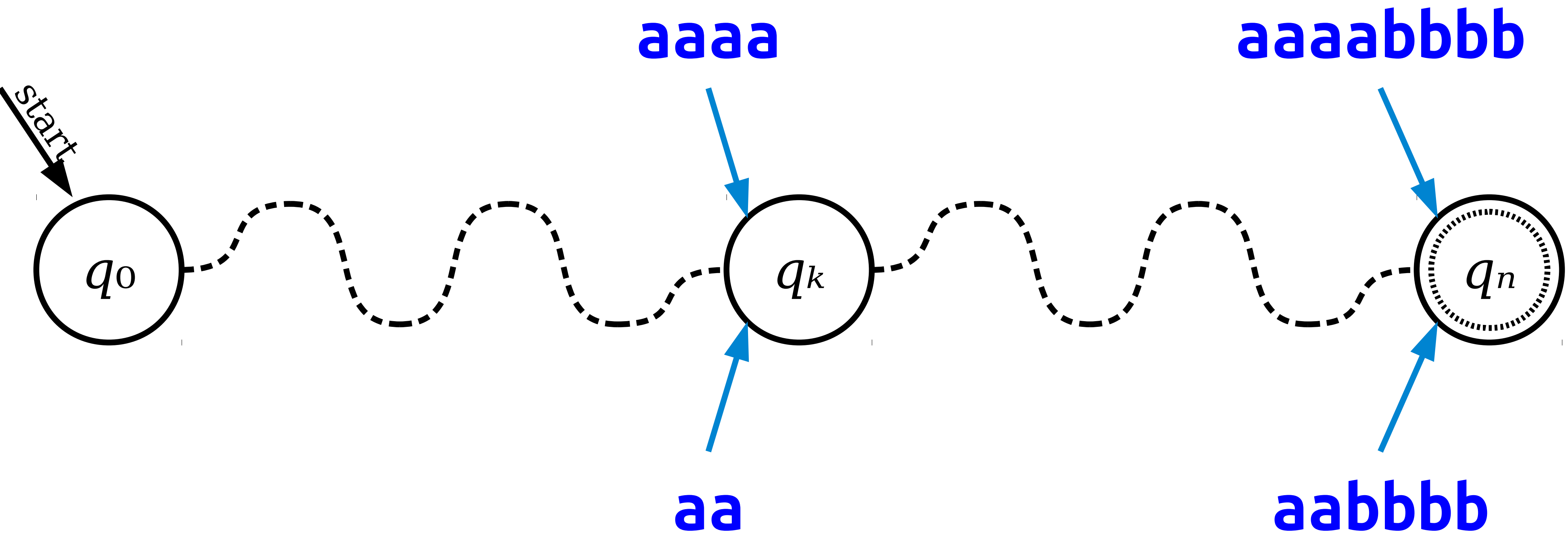
Let's imagine what a DFA for the language $\{ \mathbf{a}^n\mathbf{b}^n \mid n \in \mathbb{N}\}$ would have to look like.

Can we say anything about it?

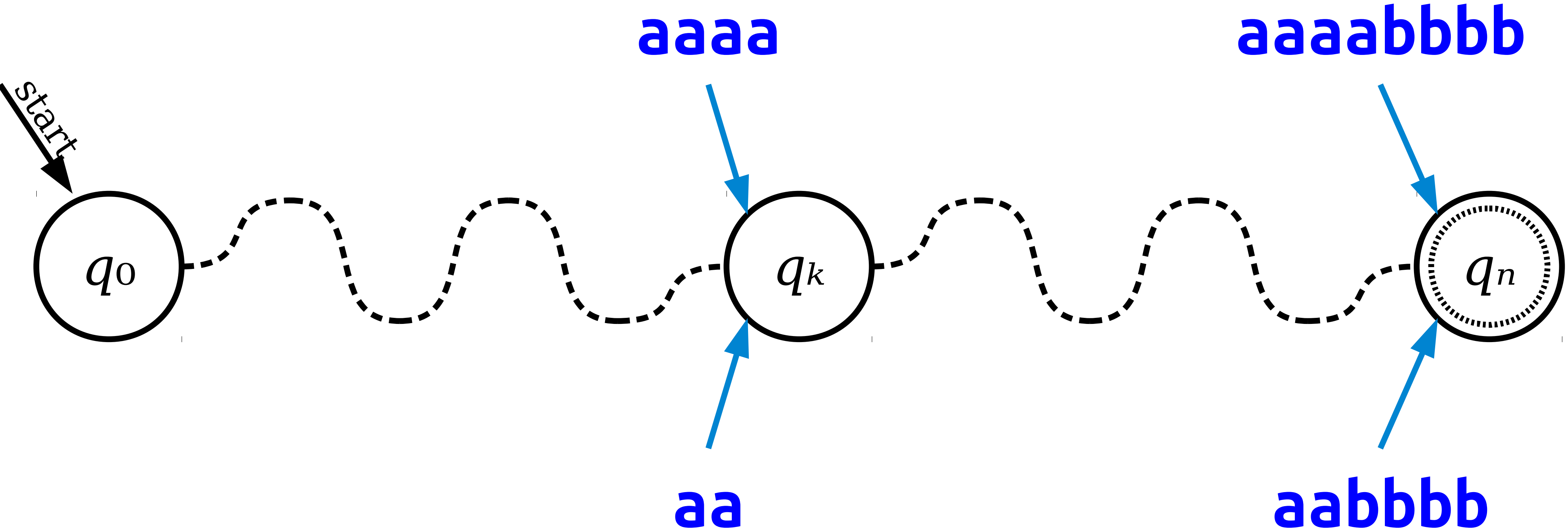


This isn't a single transition. Think of it as "after reading aaaa, we end up at this state."
aaaa
bbbb
bb
aaaabbbb
aaaabb
These *cannot* be the same state!
start
aa
bbbb
bb
aabbbb
aabb

A Different Perspective



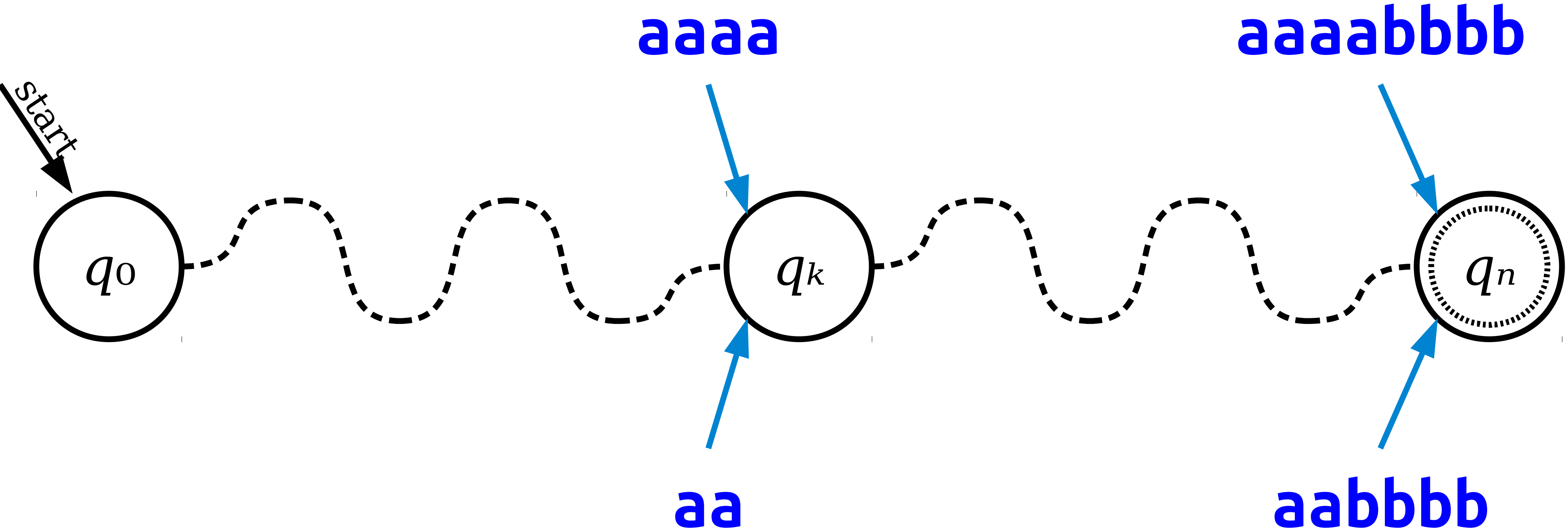
A Different Perspective



What happens if q_n is…

…an accepting state?
…a rejecting state?

A Different Perspective

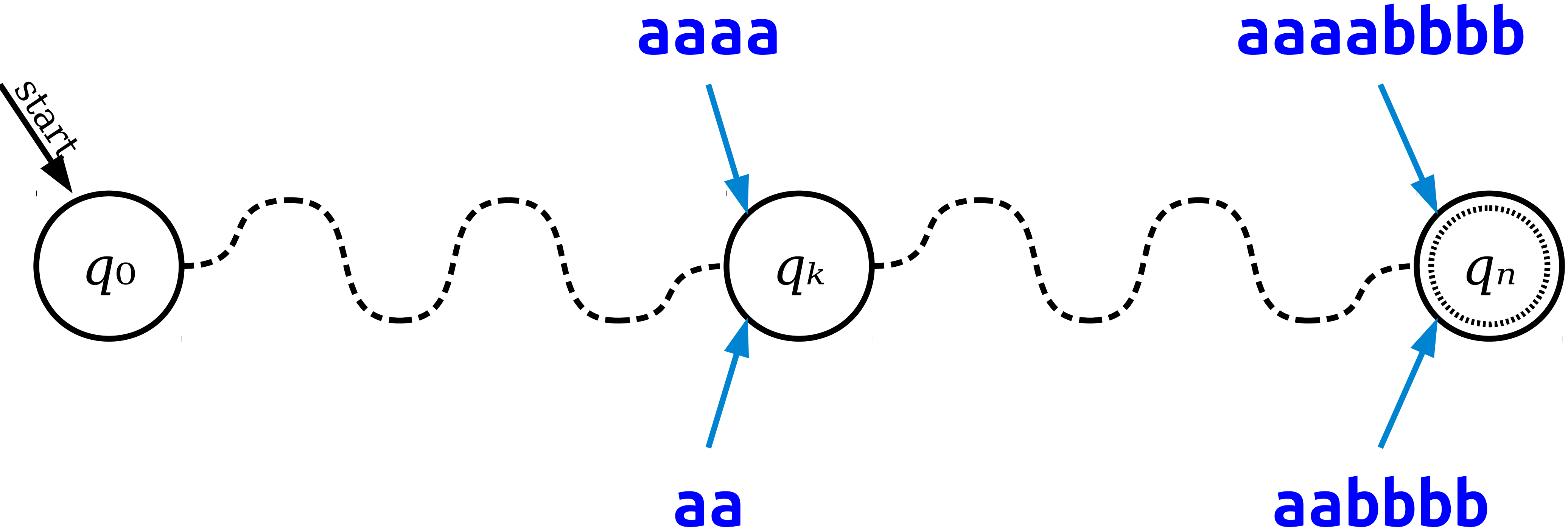


What happens if q_n is…

…an accepting state? We accept **aabbbb** $\notin E$!

…a rejecting state?

A Different Perspective



What happens if q_n is…

…an accepting state? We accept **aabbbb** $\notin E$!
…a rejecting state? We reject **aaaabbbb** $\in E$!

What's Going On?

- As you just saw, the strings $\mathtt{a}^4$ and $\mathtt{a}^2$ can't end up in the same state in *any* DFA for $E = \{\mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N}\}$.
- Two proof routes:
  - *Direct:* The states you reach for $\mathtt{a}^4$ and $\mathtt{a}^2$ have to behave differently when reading $\mathtt{b}^4$ – in one case it should lead to an accept state, in the other it should lead to a reject state. Therefore, they must be different states.
  - *Contradiction:* Suppose you do end up in the same state. Then $\mathtt{a}^4\mathtt{b}^4$ and $\mathtt{a}^2\mathtt{b}^4$ end up in the same state, so we either reject $\mathtt{a}^4\mathtt{b}^4$ (oops) or accept $\mathtt{a}^2\mathtt{b}^4$ (oops).
- ***Powerful intuition:*** Any DFA for E must keep $\mathtt{a}^4$ and $\mathtt{a}^2$ separated. It needs to remember something fundamentally different after reading those strings.

This idea – that two strings shouldn't end up in the same DFA state – is fundamental to discovering nonregular languages.

Let's go formalize this!

Distinguishability

- Let L be an arbitrary language over Σ.
- Two strings $x \in \Sigma^*$ and $y \in \Sigma^*$ are called ***distinguishable relative to L*** if there is a string $w \in \Sigma^*$ such that exactly one of xw and yw is in L.
- We denote this by writing $\boldsymbol{x \not\equiv_L y}$.
- In our previous example, we saw that $\mathtt{a}^2 \not\equiv_E \mathtt{a}^4$.
  - Try appending $\mathtt{b}^4$ to both of them.
- Formally, we say that $x \not\equiv_L y$ if the following is true:

$$\exists w \in \Sigma^*. \; (xw \in L \leftrightarrow yw \notin L)$$

If L is a language over Σ and $x, y \in \Sigma^*$, we say that

$$x \not\equiv_L y \quad \text{if} \quad \exists w \in \Sigma^*. \ (xw \in L \leftrightarrow yw \notin L)$$

Let $L = \{ w \in \{\mathtt{a}, \mathtt{b}\}^* \mid |w| \text{ is a multiple of three} \}$.

How many of the following statements are true?

$\mathtt{a} \not\equiv_L \mathtt{aa}$

$\mathtt{aaa} \not\equiv_L \mathtt{aa}$

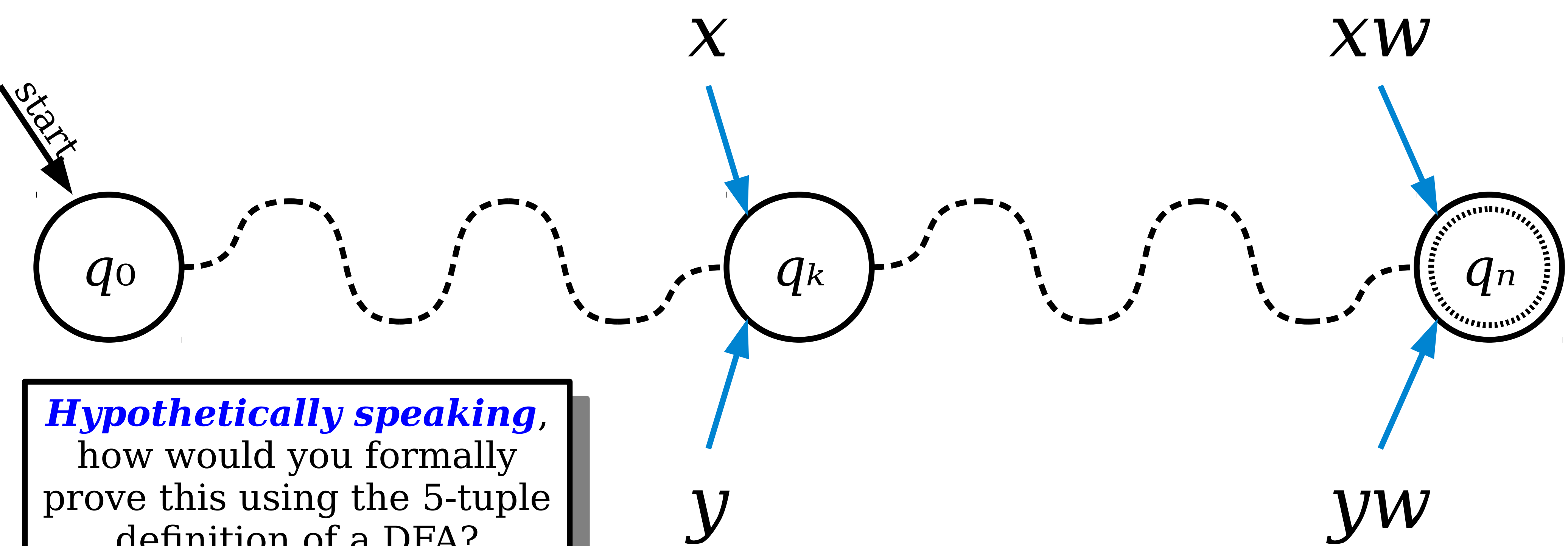
$\mathtt{a} \not\equiv_L \mathtt{aaaa}$

$\mathtt{aa} \not\equiv_L \mathtt{bb}$

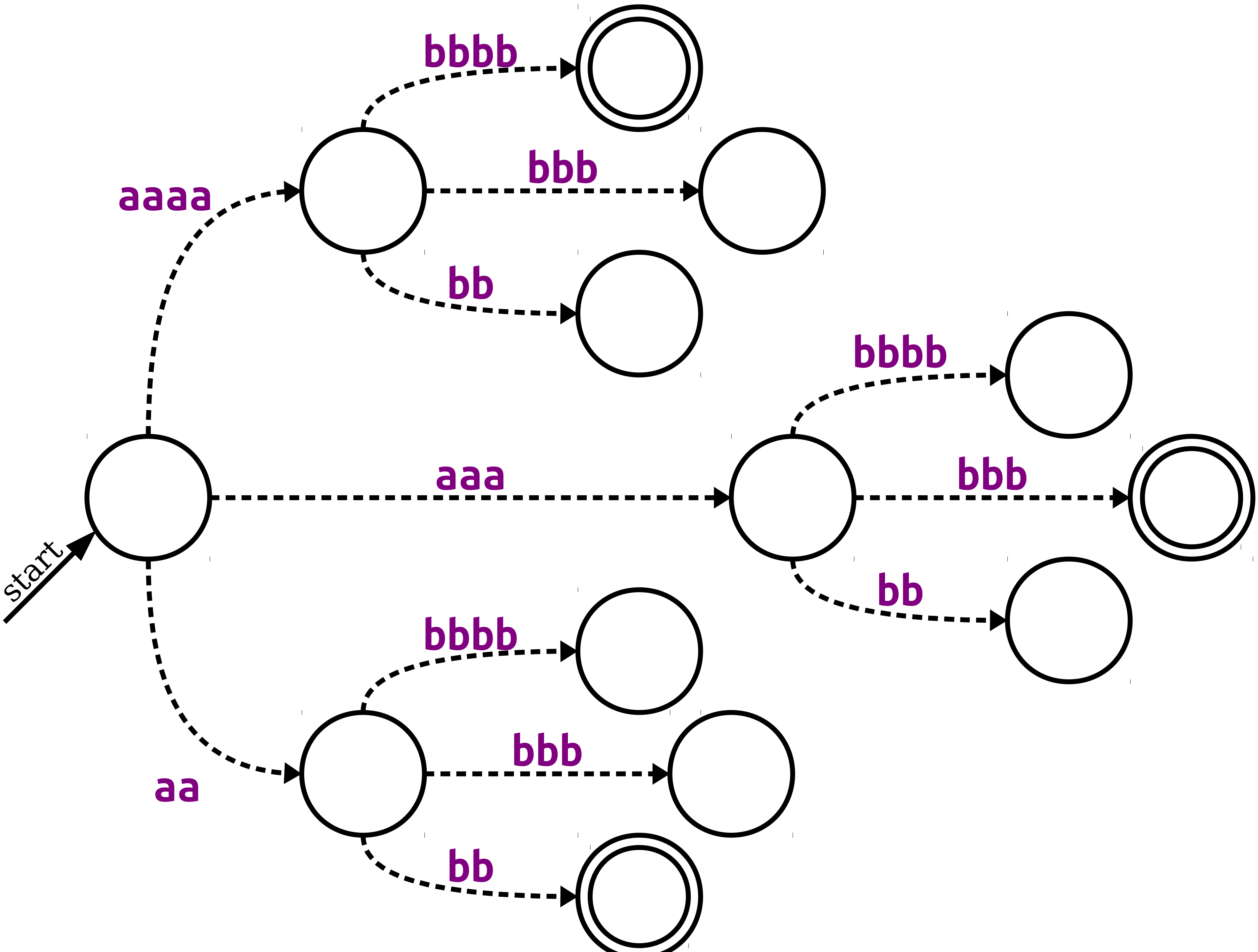
$\varepsilon \not\equiv_L \mathtt{baba}$

Distinguishability

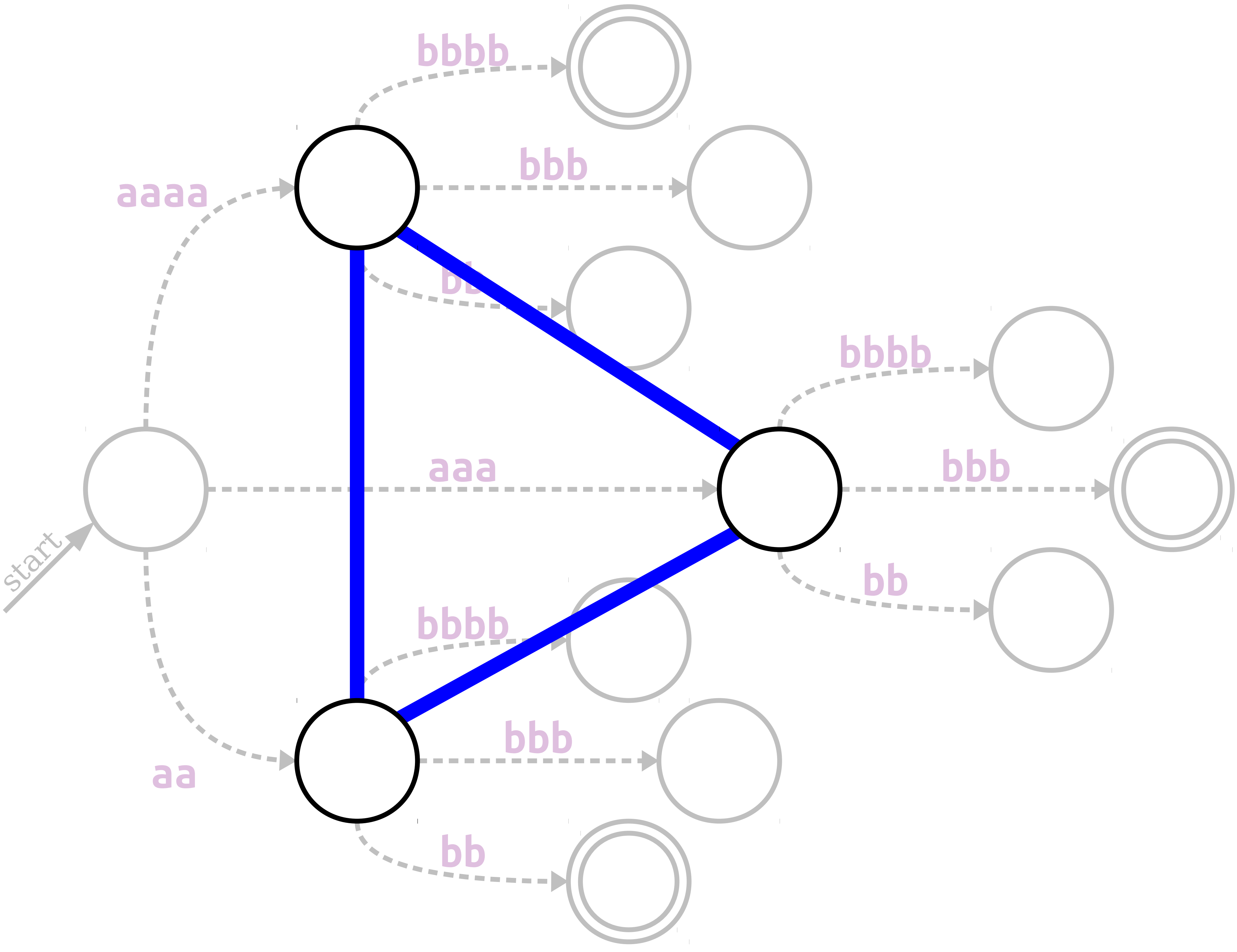
- ***Theorem:*** Let L be an arbitrary language over Σ. Let $x \in \Sigma^*$ and $y \in \Sigma^*$ be strings where $x \not\equiv_L y$. Then if D is ***any*** DFA for L, then D must end in different states when run on inputs x and y.
- ***Proof sketch:***



Hypothetically speaking, how would you formally prove this using the 5-tuple definition of a DFA?



start
aaaa
bbbb
bbb
bb
aaa
bbbb
bbb
bb
aa
bbbb
bbb
bb



bbbb
bbb
aaaa
bb
bbbb
aaa
bbb
start
bb
bbbb
bbb
aa
bb

Distinguishability

- Let's focus on this language for now:

$$E = \{ \mathbf{a}^n\mathbf{b}^n \mid n \in \mathbb{N} \}$$

Lemma: If $m, n \in \mathbb{N}$ and $m \neq n$, then $\mathbf{a}^m \not\equiv_E \mathbf{a}^n$.

Proof: Let $\mathbf{a}^m$ and $\mathbf{a}^n$ be strings where $m \neq n$. Then $\mathbf{a}^m\mathbf{b}^m \in E$ and $\mathbf{a}^n\mathbf{b}^m \notin E$. Therefore, we see that $\mathbf{a}^m \not\equiv_E \mathbf{a}^n$, as required. ■

A Bad Combination

- Suppose there is a DFA D for the language $E = \{ \mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N} \}$.
- We know the following:
  - Any two strings of the form $\mathtt{a}^m$ and $\mathtt{a}^n$, where $m \neq n$, cannot end in the same state when run through D.
  - There are infinitely many pairs of strings of the form $\mathtt{a}^m$ and $\mathtt{a}^n$.
  - However, there are only *finitely many* states they can end up in, since D is a deterministic ***finite*** automaton!
- What happens if we put these pieces together?

Theorem: The language $E = \{ \mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N} \}$ is not regular.

Theorem: The language $E = \{ \mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N} \}$ is not regular.

Proof: Suppose for the sake of contradiction that E is regular.

Theorem: The language $E = \{ \mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N} \}$ is not regular.

Proof: Suppose for the sake of contradiction that E is regular. Let D be a DFA for E, and let k be the number of states in D.

Theorem: The language $E = \{ \mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N} \}$ is not regular.

Proof: Suppose for the sake of contradiction that E is regular. Let D be a DFA for E, and let k be the number of states in D. Consider the strings $\mathtt{a}^0$, $\mathtt{a}^1$, $\mathtt{a}^2$, …, $\mathtt{a}^k$.

Theorem: The language $E = \{ \mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N} \}$ is not regular.

Proof: Suppose for the sake of contradiction that E is regular. Let D be a DFA for E, and let k be the number of states in D. Consider the strings $\mathtt{a}^0$, $\mathtt{a}^1$, $\mathtt{a}^2$, …, $\mathtt{a}^k$. This is a collection of $k+1$ strings and there are only k states in D.

Theorem: The language $E = \{ \mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N} \}$ is not regular.

Proof: Suppose for the sake of contradiction that E is regular. Let D be a DFA for E, and let k be the number of states in D. Consider the strings $\mathtt{a}^0$, $\mathtt{a}^1$, $\mathtt{a}^2$, …, $\mathtt{a}^k$. This is a collection of $k+1$ strings and there are only k states in D. Therefore, by the pigeonhole principle, there must be two distinct strings $\mathtt{a}^m$ and $\mathtt{a}^n$ that end in the same state when run through D.

Theorem: The language $E = \{ \mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N} \}$ is not regular.

Proof: Suppose for the sake of contradiction that E is regular. Let D be a DFA for E, and let k be the number of states in D. Consider the strings $\mathtt{a}^0$, $\mathtt{a}^1$, $\mathtt{a}^2$, …, $\mathtt{a}^k$. This is a collection of $k+1$ strings and there are only k states in D. Therefore, by the pigeonhole principle, there must be two distinct strings $\mathtt{a}^m$ and $\mathtt{a}^n$ that end in the same state when run through D.

Our lemma tells us that $\mathtt{a}^m \not\equiv_E \mathtt{a}^n$, so by our earlier theorem we know that $\mathtt{a}^m$ and $\mathtt{a}^n$ cannot end in the same state when run through D.

Theorem: The language $E = \{ \mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N} \}$ is not regular.

Proof: Suppose for the sake of contradiction that E is regular. Let D be a DFA for E, and let k be the number of states in D. Consider the strings $\mathtt{a}^0, \mathtt{a}^1, \mathtt{a}^2, \ldots, \mathtt{a}^k$. This is a collection of $k+1$ strings and there are only k states in D. Therefore, by the pigeonhole principle, there must be two distinct strings $\mathtt{a}^m$ and $\mathtt{a}^n$ that end in the same state when run through D.

Our lemma tells us that $\mathtt{a}^m \not\equiv_E \mathtt{a}^n$, so by our earlier theorem we know that $\mathtt{a}^m$ and $\mathtt{a}^n$ cannot end in the same state when run through D. But this is impossible, since we know that $\mathtt{a}^m$ and $\mathtt{a}^n$ do end in the same state when run through D.

We have reached a contradiction, so our assumption must have been wrong.

Theorem: The language $E = \{ \mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N} \}$ is not regular.

Proof: Suppose for the sake of contradiction that E is regular. Let D be a DFA for E, and let k be the number of states in D. Consider the strings $\mathtt{a}^0, \mathtt{a}^1, \mathtt{a}^2, \ldots, \mathtt{a}^k$. This is a collection of $k+1$ strings and there are only k states in D. Therefore, by the pigeonhole principle, there must be two distinct strings $\mathtt{a}^m$ and $\mathtt{a}^n$ that end in the same state when run through D.

Our lemma tells us that $\mathtt{a}^m \not\equiv_E \mathtt{a}^n$, so by our earlier theorem we know that $\mathtt{a}^m$ and $\mathtt{a}^n$ cannot end in the same state when run through D. But this is impossible, since we know that $\mathtt{a}^m$ and $\mathtt{a}^n$ do end in the same state when run through D.

We have reached a contradiction, so our assumption must have been wrong. Therefore, E is not regular.

Theorem: The language $E = \{\ \mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N}\ \}$ is not regular.

Proof: Suppose for the sake of contradiction that E is regular. Let D be a DFA for E, and let k be the number of states in D. Consider the strings $\mathtt{a}^0, \mathtt{a}^1, \mathtt{a}^2, \ldots, \mathtt{a}^k$. This is a collection of $k+1$ strings and there are only k states in D. Therefore, by the pigeonhole principle, there must be two distinct strings $\mathtt{a}^m$ and $\mathtt{a}^n$ that end in the same state when run through D.

Our lemma tells us that $\mathtt{a}^m \not\equiv_E \mathtt{a}^n$, so by our earlier theorem we know that $\mathtt{a}^m$ and $\mathtt{a}^n$ cannot end in the same state when run through D. But this is impossible, since we know that $\mathtt{a}^m$ and $\mathtt{a}^n$ do end in the same state when run through D.

We have reached a contradiction, so our assumption must have been wrong. Therefore, E is not regular. ■

Theorem: The language $E = \{ \mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N} \}$ is not regular.

Proof: Suppose for the sake of contradiction that E is regular. Let D be a DFA for E, and let k be the number of states in D. Consider the strings $\mathtt{a}^0, \mathtt{a}^1, \mathtt{a}^2, \ldots, \mathtt{a}^k$. This is a collection of $k+1$ strings and there are only k states in D. Therefore, by the pigeonhole principle, there must be two distinct strings $\mathtt{a}^m$ and $\mathtt{a}^n$ that end in the same state when run through D.

Our lemma tells us that $\mathtt{a}^m \not\equiv_E \mathtt{a}^n$, so by our earlier theorem we know that $\mathtt{a}^m$ and $\mathtt{a}^n$ cannot end in the same state when run through D. But this is impossible, since we know that $\mathtt{a}^m$ and $\mathtt{a}^n$ do end in the same state when run through D.

We have reached a contradiction, so our assumption must have been wrong. Therefore, E is not regular. ■

We're going to see a simpler proof of this result later on once we've built more machinery. If (hypothetically speaking) you want to prove something like this in the future, we'd recommend not using this proof as a template.

What Just Happened?

- ***We've just hit the limit of finite-memory computation.***
- To build a DFA for $E = \{ \mathbf{a}^n\mathbf{b}^n \mid n \in \mathbb{N} \}$, we need to have different memory configurations (states) for all possible strings of the form $\mathbf{a}^n$.
- There's no way to do this with finitely many possible states!

Where We're Going

- We just used the idea of *distinguishability* to show that no possible DFA can exist for some language.
- This technique turns out to be pretty powerful.
- We're going to see one more example of this technique in action, then generalize it to an extremely powerful theorem for finding nonregular languages.

Time-Out for Announcements!

Problem Sets

- Problem Set Four solutions are now available online and in hardcopy.
- We’ll aim to get PS4 graded and returned by Thursday morning.
- Problem Set Five is due this Friday at 3:00PM.
  - ***Important***: please see course website for updated assignment handout!
  - As always, ask questions if you have them! Office hours and Piazza are great places to start.

“Practice Midterm” Exam

- Solutions for the optional “practice midterm” exam are up on the course website.
- **There is no midterm in this course**, but we recommend taking some time in the next week to actually sit down and try taking this exam to check your understanding.
- Please don’t read the solutions until you’ve attempted the problems!

Final Exam

- As a reminder, the final exam for CS103 will be on Friday August 16th from 7-10 PM.
- In the past, we've had a sit-down practice exam and/or a review session – if you're interested in either of those, send me an email!
- If you have OAE accommodations, please email the me ASAP with your letter so that we have adequate space and staffing for everyone.

Let’s take a five minute break!

More Nonregular Languages

Another Language

- Consider the following language L over the alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}, \stackrel{?}{=}\}$:

$$EQ = \{\ w \stackrel{?}{=} w \mid w \in \{\mathtt{a}, \mathtt{b}\}^*\}$$

- EQ is the language all strings consisting of the same string of **a**'s and **b**'s twice, with a $\stackrel{?}{=}$ symbol in-between.
- Examples:

$\mathtt{ab}\stackrel{?}{=}\mathtt{ab} \in EQ$ $\qquad$ $\mathtt{bbb}\stackrel{?}{=}\mathtt{bbb} \in EQ$ $\qquad$ $\stackrel{?}{=} \in EQ$

$\mathtt{ab}\stackrel{?}{=}\mathtt{ba} \notin EQ$ $\qquad$ $\mathtt{bbb}\stackrel{?}{=}\mathtt{aaa} \notin EQ$ $\qquad$ $\mathtt{b}\stackrel{?}{=} \notin EQ$

Another Language

$$EQ = \{ \ w \stackrel{?}{=} w \mid w \in \{\texttt{a}, \texttt{b}\}^* \}$$

- This language corresponds to the following problem:

Given strings x, y ∈, {a, b}*, does $x = y$?

- Justification: $x = y$ happens if and only if $x \stackrel{?}{=} y \in EQ$.
- Is this language regular?

The Intuition

$$EQ = \{\ w \stackrel{?}{=} w \mid w \in \{\texttt{a}, \texttt{b}\}^*\}$$

- Intuitively, any machine for EQ has to be able to remember the contents of everything to the left of the $\stackrel{?}{=}$ so that it can match them against the contents of the string to the right of the $\stackrel{?}{=}$.
- There are infinitely many possible strings we can see, but we only have finite memory to store which string we saw.
- That's a problem… can we formalize this?

If L is a language over Σ and $x, y \in \Sigma^*$, we say that

$$\boldsymbol{x \not\equiv_L y} \quad \textbf{if} \quad \boldsymbol{\exists w \in \Sigma^*. (xw \in L \leftrightarrow yw \notin L)}$$

Let $\Sigma = \{\mathbf{a}, \mathbf{b}, \stackrel{?}{=}\}$ and
Let $EQ = \{\ w\stackrel{?}{=}w \mid w \in \{\mathbf{a}, \mathbf{b}\}^*\}$

How many of the following statements are true?

$\mathbf{a} \not\equiv_{EQ} \mathbf{b}$

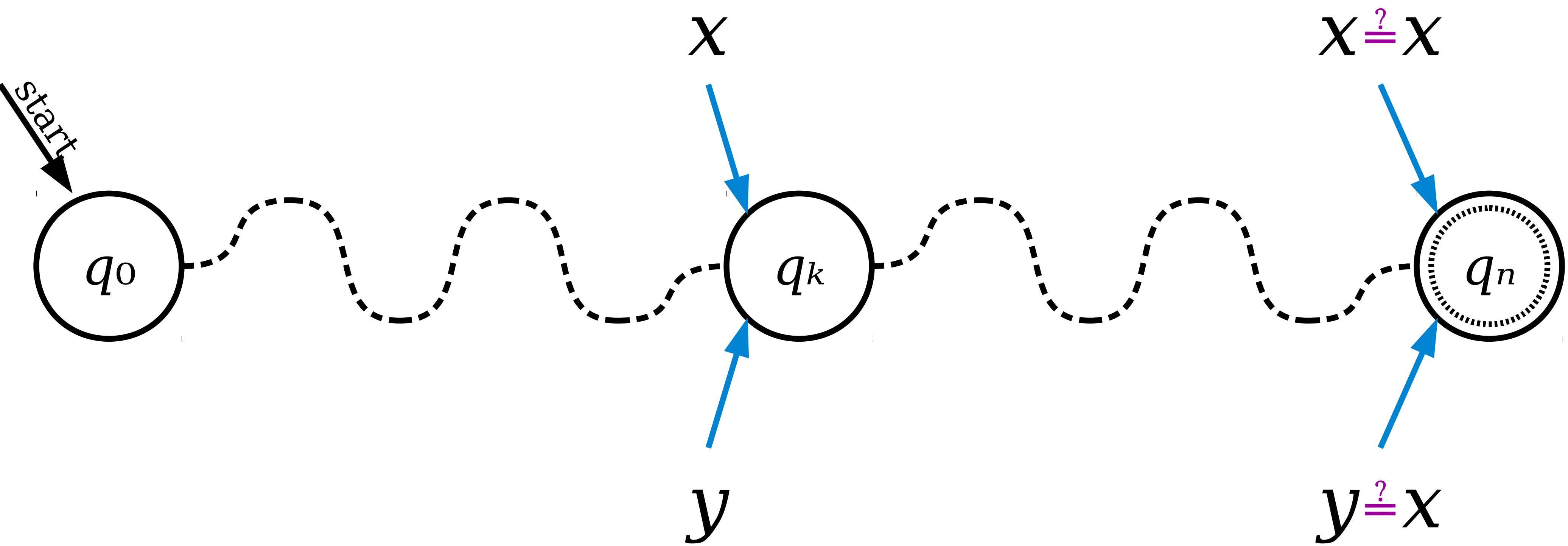
$\mathbf{abb} \not\equiv_{EQ} \mathbf{abb}$

$\varepsilon \not\equiv_{EQ} \mathbf{ab}$

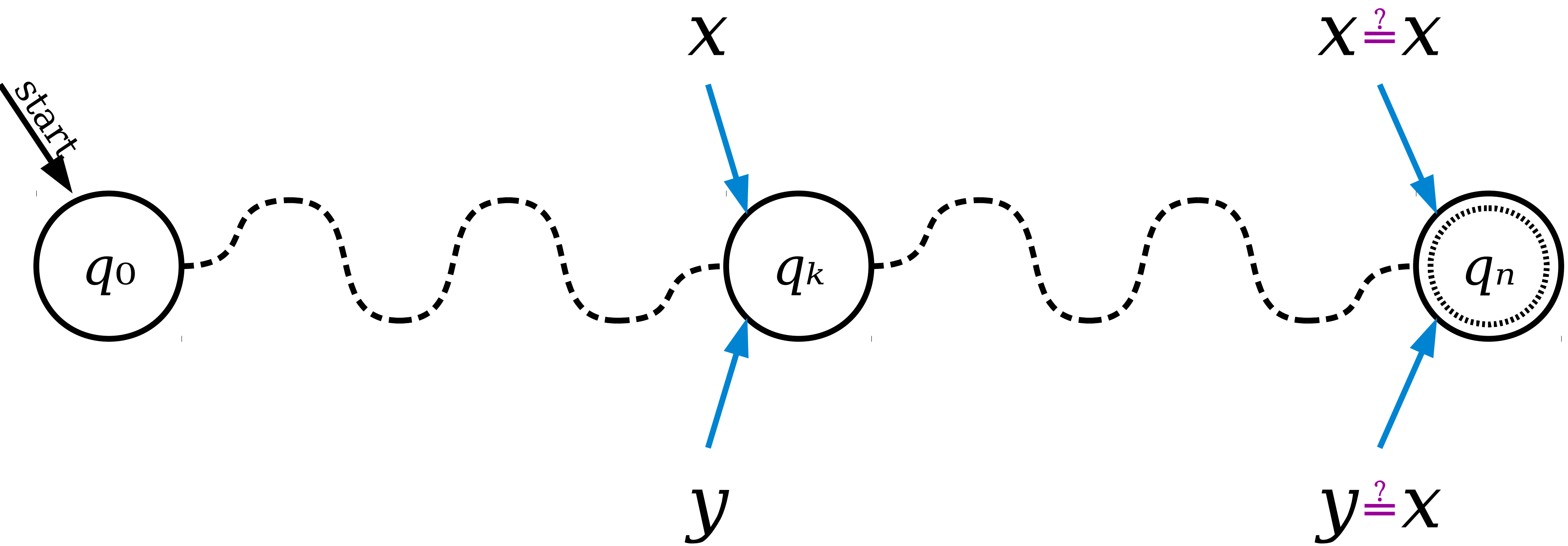
$\stackrel{?}{=}\stackrel{?}{=} \not\equiv_{EQ} \stackrel{?}{=}$

$\stackrel{?}{=}\stackrel{?}{=}\stackrel{?}{=} \not\equiv_{EQ} \stackrel{?}{=}\stackrel{?}{=}$

The Intuition



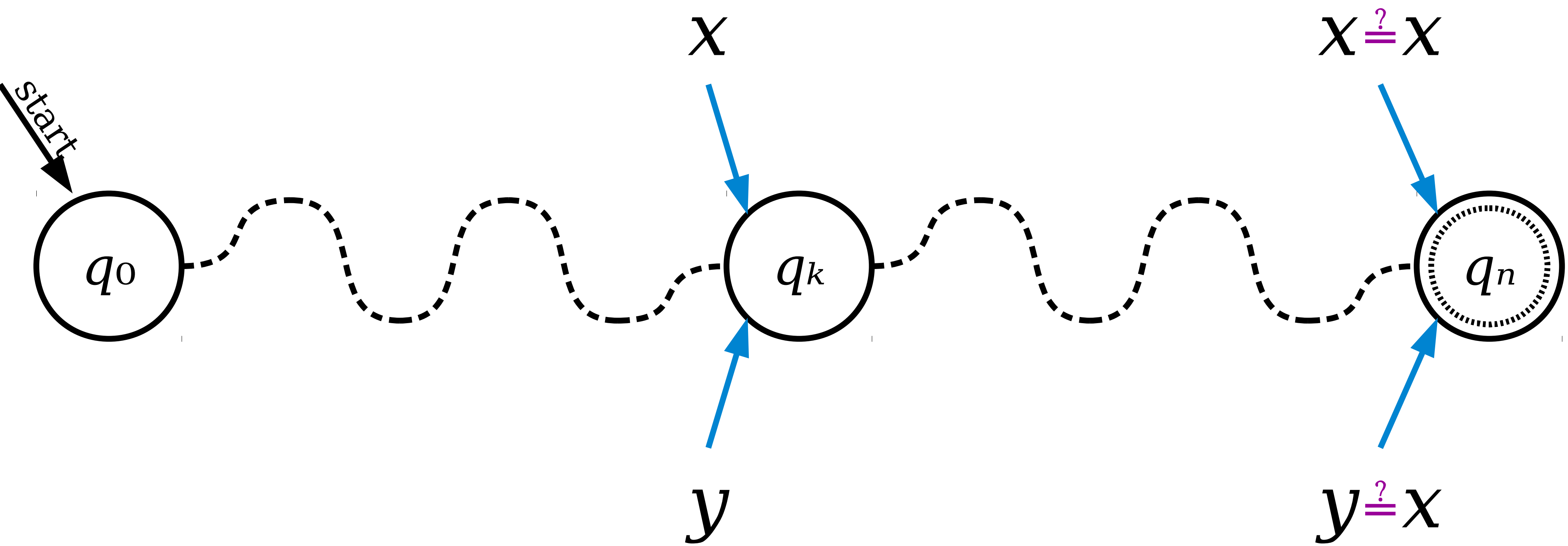
The Intuition



What happens if q_n is…

…an accepting state?
…a rejecting state?

The Intuition

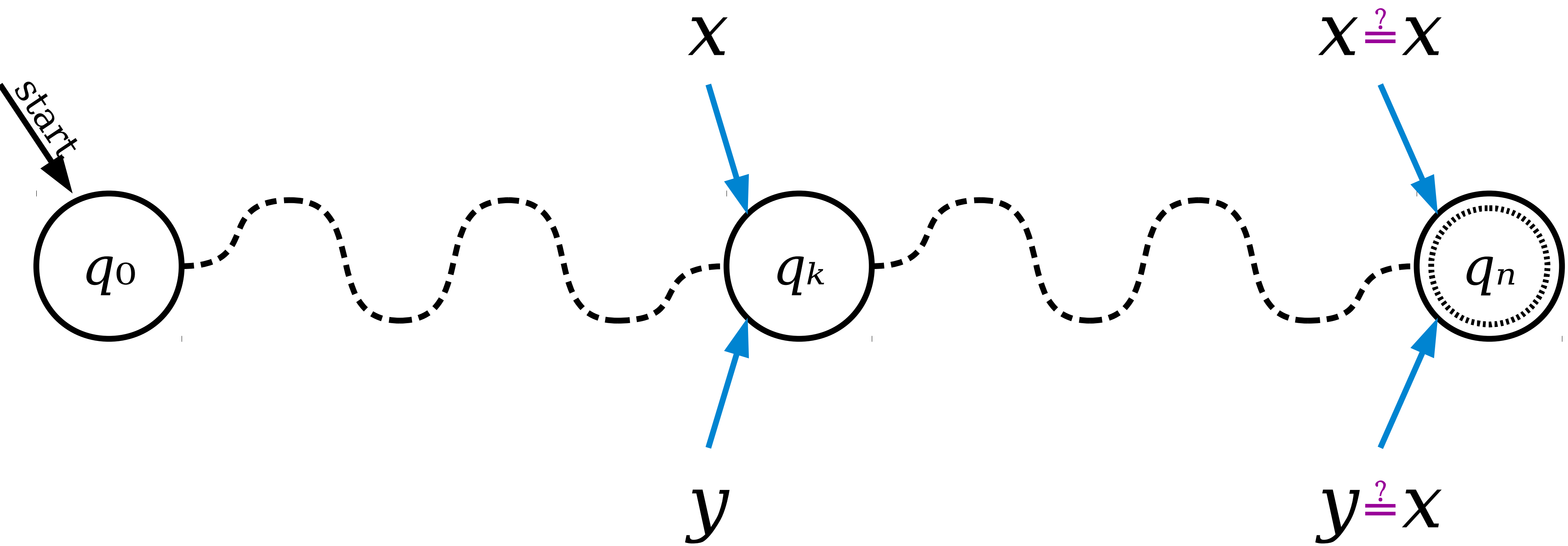


What happens if q_n is...

...an accepting state? We accept $y \stackrel{?}{=} x \notin EQ$!

...a rejecting state?

The Intuition



What happens if q_n is…

…an accepting state? We accept $y \stackrel{?}{=} x \notin EQ$!

…a rejecting state? We reject $x \stackrel{?}{=} x \in EQ$!

Distinguishability

- Let's focus on this language for now:

$$EQ = \{ \, w \stackrel{?}{=} w \mid w \in \{\texttt{a}, \texttt{b}\}^* \}$$

Lemma: If $x, y \in \{\texttt{a}, \texttt{b}\}^*$ and $x \neq y$, then $x \not\equiv_{EQ} y$.

Proof: Let x and y be two distinct, arbitrary strings from $\{\texttt{a}, \texttt{b}\}^*$. Then we see that $x \stackrel{?}{=} x \in EQ$ and $y \stackrel{?}{=} x \notin EQ$, so we conclude that $x \not\equiv_{EQ} y$, as required. ■

Theorem: The language $EQ = \{ w \stackrel{?}{=} w \mid w \in \{\mathtt{a}, \mathtt{b}\}^* \}$ is not regular.

Theorem: The language $EQ = \{ w \stackrel{?}{=} w \mid w \in \{\texttt{a}, \texttt{b}\}^* \}$ is not regular.

Proof:

Theorem: The language $EQ = \{\ w \stackrel{?}{=} w \mid w \in \{\texttt{a}, \texttt{b}\}^*\}$ is not regular.

Proof: Suppose for the sake of contradiction that EQ is regular.

Theorem: The language $EQ = \{ w \stackrel{?}{=} w \mid w \in \{\texttt{a}, \texttt{b}\}^* \}$ is not regular.

Proof: Suppose for the sake of contradiction that EQ is regular. Let D be a DFA for EQ and let k be the number of states in D.

Theorem: The language $EQ = \{\ w \stackrel{?}{=} w \mid w \in \{$**`a`**, **`b`**$\}^*\}$ is not regular.

Proof: Suppose for the sake of contradiction that EQ is regular. Let D be a DFA for EQ and let k be the number of states in D. Consider any $k+1$ distinct strings in $\{$**`a`**, **`b`**$\}^*$.

Theorem: The language $EQ = \{\ w \stackrel{?}{=} w \mid w \in \{\mathtt{a}, \mathtt{b}\}^*\}$ is not regular.

Proof: Suppose for the sake of contradiction that EQ is regular. Let D be a DFA for EQ and let k be the number of states in D. Consider any $k+1$ distinct strings in $\{\mathtt{a}, \mathtt{b}\}^*$. Because D only has k states, by the pigeonhole principle there must be at least two strings x and y that, when run through D, end in the same state.

Theorem: The language $EQ = \{ w \stackrel{?}{=} w \mid w \in \{\texttt{a}, \texttt{b}\}^* \}$ is not regular.

Proof: Suppose for the sake of contradiction that EQ is regular. Let D be a DFA for EQ and let k be the number of states in D. Consider any $k+1$ distinct strings in $\{\texttt{a}, \texttt{b}\}^*$. Because D only has k states, by the pigeonhole principle there must be at least two strings x and y that, when run through D, end in the same state.

Our lemma tells us that $x \not\equiv_{EQ} y$.

Theorem: The language $EQ = \{ w \stackrel{?}{=} w \mid w \in \{\texttt{a}, \texttt{b}\}^* \}$ is not regular.

Proof: Suppose for the sake of contradiction that EQ is regular. Let D be a DFA for EQ and let k be the number of states in D. Consider any $k+1$ distinct strings in $\{\texttt{a}, \texttt{b}\}^*$. Because D only has k states, by the pigeonhole principle there must be at least two strings x and y that, when run through D, end in the same state.

Our lemma tells us that $x \not\equiv_{EQ} y$. By our earlier theorem, this means that x and y cannot end in the same state when run through D.

Theorem: The language $EQ = \{ w \stackrel{?}{=} w \mid w \in \{\texttt{a}, \texttt{b}\}^* \}$ is not regular.

Proof: Suppose for the sake of contradiction that EQ is regular. Let D be a DFA for EQ and let k be the number of states in D. Consider any $k+1$ distinct strings in $\{\texttt{a}, \texttt{b}\}^*$. Because D only has k states, by the pigeonhole principle there must be at least two strings x and y that, when run through D, end in the same state.

Our lemma tells us that $x \not\equiv_{EQ} y$. By our earlier theorem, this means that x and y cannot end in the same state when run through D. But this is impossible, since we specifically chose x and y to end in the same state when run through D.

Theorem: The language $EQ = \{ w \stackrel{?}{=} w \mid w \in \{\mathtt{a}, \mathtt{b}\}^* \}$ is not regular.

Proof: Suppose for the sake of contradiction that EQ is regular. Let D be a DFA for EQ and let k be the number of states in D. Consider any $k+1$ distinct strings in $\{\mathtt{a}, \mathtt{b}\}^*$. Because D only has k states, by the pigeonhole principle there must be at least two strings x and y that, when run through D, end in the same state.

Our lemma tells us that $x \not\equiv_{EQ} y$. By our earlier theorem, this means that x and y cannot end in the same state when run through D. But this is impossible, since we specifically chose x and y to end in the same state when run through D.

We have reached a contradiction, so our assumption must have been wrong.

Theorem: The language $EQ = \{ w \stackrel{?}{=} w \mid w \in \{\mathtt{a}, \mathtt{b}\}^* \}$ is not regular.

Proof: Suppose for the sake of contradiction that EQ is regular. Let D be a DFA for EQ and let k be the number of states in D. Consider any $k+1$ distinct strings in $\{\mathtt{a}, \mathtt{b}\}^*$. Because D only has k states, by the pigeonhole principle there must be at least two strings x and y that, when run through D, end in the same state.

Our lemma tells us that $x \not\equiv_{EQ} y$. By our earlier theorem, this means that x and y cannot end in the same state when run through D. But this is impossible, since we specifically chose x and y to end in the same state when run through D.

We have reached a contradiction, so our assumption must have been wrong. Thus EQ is not regular.

Theorem: The language $EQ = \{\ w \overset{?}{=} w \mid w \in \{\texttt{a}, \texttt{b}\}^*\}$ is not regular.

Proof: Suppose for the sake of contradiction that EQ is regular. Let D be a DFA for EQ and let k be the number of states in D. Consider any $k+1$ distinct strings in $\{\texttt{a}, \texttt{b}\}^*$. Because D only has k states, by the pigeonhole principle there must be at least two strings x and y that, when run through D, end in the same state.

Our lemma tells us that $x \not\equiv_{EQ} y$. By our earlier theorem, this means that x and y cannot end in the same state when run through D. But this is impossible, since we specifically chose x and y to end in the same state when run through D.

We have reached a contradiction, so our assumption must have been wrong. Thus EQ is not regular. ■

Theorem: The language $EQ = \{\ w \stackrel{?}{=} w \mid w \in \{$`a`$,$ `b`$\}^*\}$ is not regular.

Proof: Suppose for the sake of contradiction that EQ is regular. Let D be a DFA for EQ and let k be the number of states in D. Consider any $k+1$ distinct strings in $\{$`a`$,$ `b`$\}^*$. Because D only has k states, by the pigeonhole principle there must be at least two strings x and y that, when run through D, end in the same state.

Our lemma tells us that $x \not\equiv_{EQ} y$. By our earlier theorem, this means that x and y cannot end in the same state when run through D. But this is impossible, since we specifically chose x and y to end in the same state when run through D.

We have reached a c

have been wrong. Th

We're going to see a simpler proof of this result later on once we've built more machinery. If (hypothetically speaking) you want to prove something like this in the future, we'd recommend not using this proof as a template.

Comparing Proofs

Theorem: The language $E = \{ \mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N} \}$ is not a regular language.

Proof: Suppose for the sake of contradiction that E is regular. Let D be a DFA for E and let k be the number of states in D.

Consider the strings $\mathtt{a}^0, \mathtt{a}^1, \mathtt{a}^2, \ldots, \mathtt{a}^k$. This is a collection of $k+1$ strings and there are only k states in D. Therefore, by the pigeonhole principle, there must be two distinct strings $\mathtt{a}^m$ and $\mathtt{a}^n$ that end in the same state when run through D.

Our lemma tells us that $\mathtt{a}^m \not\equiv_E \mathtt{a}^n$. By our earlier theorem we know that $\mathtt{a}^m$ and $\mathtt{a}^n$ cannot end in the same state when run through D. But this is impossible, since we know that $\mathtt{a}^m$ and $\mathtt{a}^n$ do end in the same state when run through D.

We have reached a contradiction, so our assumption must have been wrong. Therefore, E is not regular. ■

Theorem: The language $EQ = \{\ w \stackrel{?}{=} w \mid w \in \{\mathtt{a}, \mathtt{b}\}^*\}$ is not a regular language.

Proof: Suppose for the sake of contradiction that EQ is regular. Let D be a DFA for EQ and let k be the number of states in D.

Consider any $k+1$ distinct strings in $\{\mathtt{a}, \mathtt{b}\}^*$.These are $k+1$ strings and there are only k states in D. By the pigeonhole principle, there must be two distinct strings x and y from this group that end in the same state when run through D.

Our lemma tells us that $x \not\equiv_{EQ} y$. By our earlier theorem we know that x and y cannot end in the same state when run through D. But this is impossible, since specifically chose x and y to end in the same state when run through D.

We have reached a contradiction, so our assumption must have been wrong. Therefore, EQ is not regular. ■

Theorem: The language L = **[fill in the blank]** is not a regular language.

Proof: Suppose for the sake of contradiction that L is regular. Let D be a DFA for L and let k be the number of states in D.

Consider **[some k+1 specific strings.]** This is a collection of $k+1$ strings and there are only k states in D. Therefore, by the pigeonhole principle, there must be two distinct strings x and y that end in the same state when run through D.

[Somehow we know] that $x \not\equiv_L y$. By our earlier theorem we know that x and y cannot end in the same state when run through D. But this is impossible, since we know that x and y must end in the same state when run through D.

We have reached a contradiction, so our assumption must have been wrong. Therefore, L is not regular. ■

Theorem: The language L = **[fill in the blank]** is not a regular language.

Proof: Suppose for the sake of contradiction that L is regular. Let D be a DFA for L and let k be the number of states in D.

Consider **[some *k*+1 specific strings.]** This is a collection of $k+1$ strings and there are only k states in D. Therefore, by the pigeonhole principle, there must be two distinct strings x and y that end in the same state when run through D.

[Somehow we know] that $x \not\equiv_L y$. By our earlier theorem we know that x and y cannot end in the same state when run through D. But this is impossible, since we know that x and y must end in the same state when run through D.

We have reached a contradiction, so our assumption must have been wrong. Therefore, L is not regular. ■

For any number of states k, we need a way to find k+1 strings so that two of them get into the same state…

Theorem: The language L = **[**
regular language.

Proof: Suppose for the sake of
Let D be a DFA for L and let k be the number of states in D.

Consider **[some *k*+1 specific strings.]** This is a collection of $k+1$ strings and there are only k states in D. Therefore, by the pigeonhole principle, there must be two distinct strings x and y that end in the same state when run through D.

[Somehow we know] that $x \not\equiv_L y$. By our earlier theorem we know that x and y cannot end in the same state when run through D. But this is impossible, since we know that x and y must end in the same state when run through D.

We have reached a contradiction, so our assumption must have been wrong. Therefore, L is not regular. ■

For any number of states k, we need a way to find $k+1$ strings so that two of them get into the same state…

Theorem: The language L = [

regular language.

Proof: Suppose for the sake of

Let D be a DFA for L and let k be the number of states in D.

Consider **[some *k*+1 specific strings.]** This is a collection of $k+1$ strings and there are only k states in D. Therefore, by the pigeonhole principle, there must be two distinct strings x and y that end in the same state when run through D.

[Somehow we know] that $x \not\equiv_L y$. By our earlier theorem we know that x and y cannot end in the same state when run through D. But this is impossible, since we know that x and y must end in the same state when run through D.

We have reached a contradiction, s

have been wrong. Therefore, L is n

… and all those strings need to be distinguishable so that we get a contradiction.

Imagine we have an infinite set of strings S with the following property:

$$\forall x \in S.\ \forall y \in S.\ (x \neq y \to x \not\equiv_L y)$$

What happens?

For any number of states k, we need a way to find k+1 strings so that two of them get into the same state…

[

of

et k be the number of states in D.

Consider **[some *k*+1 specific strings.]** This is a collection of $k+1$ strings and there are only k states in D. Therefore, by the pigeonhole principle, there must be two distinct strings x and y that end in the same state when run through D.

[Somehow we know] that $x \not\equiv_L y$. By our earlier theorem we know that x and y cannot end in the same state when run through D. But this is impossible, since we know that x and y must end in the same state when run through D.

We have reached a contradiction,

have been wrong. Therefore, L is

… and all those strings need to be distinguishable so that we get a contradiction.

The Myhill-Nerode Theorem

Theorem: Let L be a language over Σ.
If there is a set $S \subseteq \Sigma^*$ with the following properties, then L is not regular:

- S is infinite (that is, S contains infinitely many strings).
- The strings in S are ***pairwise distinguishable relative to L***. That is,

$$\forall x \in S.\ \forall y \in S.\ (x \neq y \to x \not\equiv_L y).$$

The Myhill-Nerode Theorem

Theorem: Let L be a language over Σ. If there is a set $S \subseteq \Sigma^*$ with the following properties, then L is not regular:

- S is infinite (that is, S contains infinitely many strings).
- The strings in S are ***pairwise distinguishable relative to L***. That is,

$$\forall x \in S.\ \forall y \in S.\ (x \neq y \to x \not\equiv_L y).$$

If you pick any two strings in S that aren't equal to one another…

… then they're distinguishable relative to L.

Proof: Let L be an arbitrary language over Σ. Let $S \subseteq \Sigma^*$ be an infinite set of strings with the following property: if $x, y \in S$ and $x \neq y$, then $x \not\equiv_L y$. We will show that L is not regular.

Suppose for the sake of contradiction that L is regular. This means that there must be some DFA D for L. Let k be the number of states in D. Since there are infinitely many strings in S, we can choose $k+1$ distinct strings from S and consider what happens when we run D on all of those strings. Because there are only k states in D and we've chosen $k+1$ strings from S, by the pigeonhole principle we know that at least two strings from S must end in the same state in D. Choose any two such strings and call them x and y.

Because x and y are distinct strings in S, we know that $x \not\equiv_L y$. Our earlier theorem therefore tells us that when we run D on inputs x and y, they must end up in different states. But this is impossible – we chose x and y precisely because they end in the same state when run through D.

We have reached a contradiction, so our assumption must have been wrong. Thus L is not a regular language. ■

Using the Myhill-Nerode Theorem

Theorem: The language $E = \{ \mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N} \}$ is not regular.

Theorem: The language $E = \{\ \mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N}\ \}$ is not regular.

To use the Myhill-Nerode theorem, we need to find an infinite set of strings that are pairwise distinguishable relative to E.

Theorem: The language $E = \{ \mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N} \}$ is not regular.

To use the Myhill-Nerode theorem, we need to find an infinite set of strings that are pairwise distinguishable relative to E.

We know that any two strings of the form $\mathtt{a}^n$ and $\mathtt{a}^m$, where $n \neq m$, are distinguishable.

Theorem: The language $E = \{ \mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N} \}$ is not regular.

To use the Myhill-Nerode theorem, we need to find an infinite set of strings that are pairwise distinguishable relative to E.

We know that any two strings of the form $\mathtt{a}^n$ and $\mathtt{a}^m$, where $n \neq m$, are distinguishable.

So pick the set $S = \{ \mathtt{a}^n \mid n \in \mathbb{N} \}$.

Theorem: The language $E = \{\ \mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N}\ \}$ is not regular.

To use the Myhill-Nerode theorem, we need to find an infinite set of strings that are pairwise distinguishable relative to E.

We know that any two strings of the form $\mathtt{a}^n$ and $\mathtt{a}^m$, where $n \neq m$, are distinguishable.

So pick the set $S = \{\ \mathtt{a}^n \mid n \in \mathbb{N}\ \}$.

Notice that S isn't a subset of E. That's okay: we never said that S needs to be a subset of E!

Theorem: The language $E = \{ \mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N} \}$ is not regular.

Proof: Let $S = \{ \mathtt{a}^n \mid n \in \mathbb{N} \}$. This set is infinite because it contains one string for each natural number. Now, consider any strings $\mathtt{a}^n, \mathtt{a}^m \in S$ where $\mathtt{a}^n \neq \mathtt{a}^m$. Then $\mathtt{a}^n\mathtt{b}^n \in E$ and $\mathtt{a}^m\mathtt{b}^n \notin E$. Consequently, $\mathtt{a}^n \not\equiv_E \mathtt{a}^m$. Therefore, by the Myhill-Nerode theorem, E is not regular. ■

Theorem: The language $EQ = \{\ w \stackrel{?}{=} w \mid w \in \{\mathtt{a},\ \mathtt{b}\}^*\}$ is not regular.

Theorem: The language $EQ = \{\ w \stackrel{?}{=} w \mid w \in \{\texttt{a}, \texttt{b}\}^*\}$ is not regular.

To use the Myhill-Nerode theorem, we need to find an infinite set of strings that are pairwise distinguishable relative to EQ.

Theorem: The language $EQ = \{ w \overset{?}{=} w \mid w \in \{\mathtt{a}, \mathtt{b}\}^* \}$ is not regular.

The Myhill-Nerode theorem asks for a set $S \subseteq \{\mathtt{a}, \mathtt{b}, \overset{?}{=}\}^*$ where S is infinite and

$$\forall x \in S.\ \forall y \in S.\ (x \neq y \rightarrow x \not\equiv_{EQ} y.)$$

Which of these sets meets these criteria?

A. $S = \{\mathtt{a}, \mathtt{b}, \overset{?}{=}\}^*$
B. $S = \{\mathtt{a}, \mathtt{b}\}^*$
C. $S = \{\mathtt{a}\overset{?}{=}\}^*$
D. $S = \{\mathtt{a}\}^*$
E. None of these, or two or more of these.

Theorem: The language $EQ = \{\ w \stackrel{?}{=} w \mid w \in \{\mathtt{a}, \mathtt{b}\}^*\}$ is not regular.

To use the Myhill-Nerode theorem, we need to find an infinite set of strings that are pairwise distinguishable relative to EQ.

We know that any two distinct strings over the alphabet $\{\mathtt{a}, \mathtt{b}\}$ are distinguishable.

Theorem: The language $EQ = \{ w \stackrel{?}{=} w \mid w \in \{\texttt{a}, \texttt{b}\}^* \}$ is not regular.

To use the Myhill-Nerode theorem, we need to find an infinite set of strings that are pairwise distinguishable relative to EQ.

We know that any two distinct strings over the alphabet $\{\texttt{a}, \texttt{b}\}$ are distinguishable.

So pick the set $S = \{\texttt{a}, \texttt{b}\}^*$.

Theorem: The language $EQ = \{ w \stackrel{?}{=} w \mid w \in \{\texttt{a}, \texttt{b}\}^* \}$ is not regular.

To use the Myhill-Nerode theorem, we need to find an infinite set of strings that are pairwise distinguishable relative to EQ.

We know that any two distinct strings over the alphabet $\{\texttt{a}, \texttt{b}\}$ are distinguishable.

So pick the set $S = \{\texttt{a}, \texttt{b}\}^*$.

Notice that S isn't a subset of EQ. That's okay: we never said that S needs to be a subset of EQ!

Theorem: The language $EQ = \{ w\overset{?}{=}w \mid w \in \{\mathtt{a}, \mathtt{b}\}^* \}$ is not regular.

Proof: Let $S = \{\mathtt{a}, \mathtt{b}\}^*$. This set contains infinitely many strings. Now, consider any $x, y \in S$ where $x \neq y$. Then $x\overset{?}{=}x \in EQ$ and $y\overset{?}{=}x \notin EQ$. Consequently, $x \not\equiv_{EQ} y$. Therefore, by the Myhill-Nerode theorem, EQ is not regular. ■

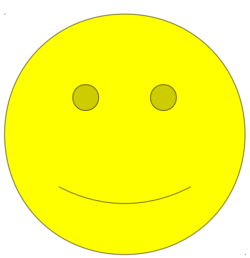
Approaching Myhill-Nerode

- The challenge in using the Myhill-Nerode theorem is finding the right set of strings.
- ***General intuition:***
  - Start by thinking about what information a computer "must" remember in order to answer correctly.
  - Choose a group of strings that all require different information.
  - Prove that those strings are distinguishable relative to the language in question.

An Analogy

Imagine a scenario where Bob is thinking of a string and Alice has to figure out whether that string is in a particular language

language L



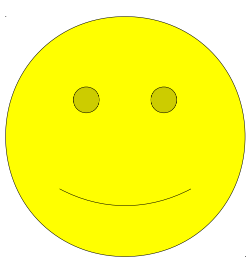
Alice

Bob

An Analogy

The catch: Bob can only send Alice one character at a time, and Alice doesn't know how long the string is until Bob tells her that he's done sending input

language L



Alice

Bob

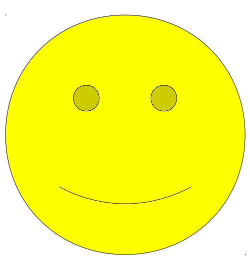
An Analogy

What does Alice need to remember about the characters she’s receiving from Bob?

An Analogy

What does Alice need to remember about the characters she's receiving from Bob?

$L = \{\ w \text{ is divisible by } 5\ \}$



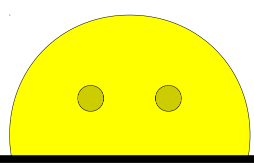
Alice

Bob

An Analogy

What does Alice need to remember about the characters she's receiving from Bob?

$L = \{ \; w \text{ is divisible by 5 } \}$

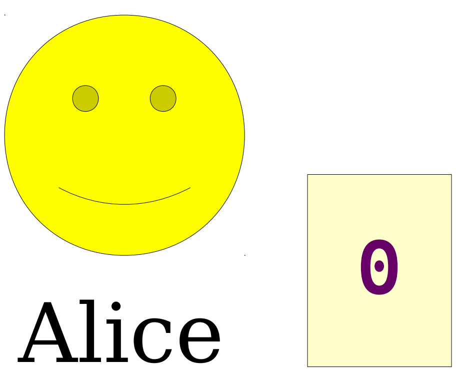


Initially it seems like Alice has to remember the whole number that Bob is sending to her, but we only care about divisibility by 5 here so we can get away with remembering a lot less.

An Analogy

Key insight: Alice only needs to remember ***the last character*** she received from Bob

L = { w is divisible by 5 }



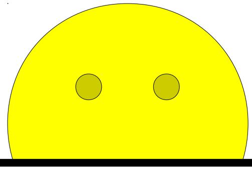
Alice

Bob

An Analogy

Key insight: Alice only needs to remember ***the last character*** she received from Bob

$L = \{\ w \text{ is divisible by } 5\ \}$

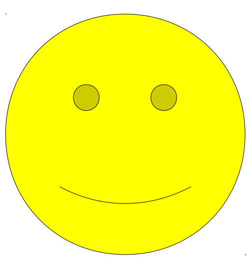


The number that Bob is thinking of could get unboundedly large, but the size of what Alice needs to remember remains constant (finite).

An Analogy

Let’s contrast this with one of the non-regular languages we saw today:

$$L = \{\ \mathbf{a}^n\mathbf{b}^n \mid n \in \mathbb{N}\ \}$$



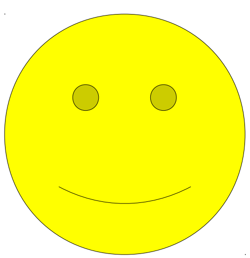
Alice

Bob

An Analogy

Alice needs to remember how many **a**'s she's seen so far, since she needs to verify that the number of **b**'s matches

$$L = \{ \mathbf{a}^n\mathbf{b}^n \mid n \in \mathbb{N} \}$$

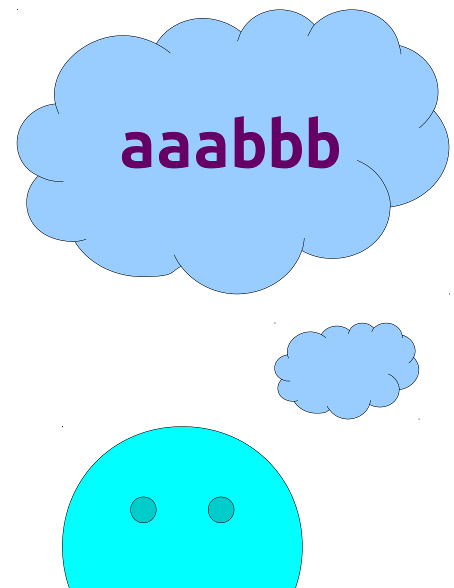


Alice

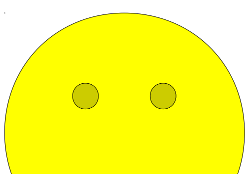
Bob

An Analogy

Alice needs to remember how many **a**'s she's seen so far, since she needs to verify that the number of **b**'s matches



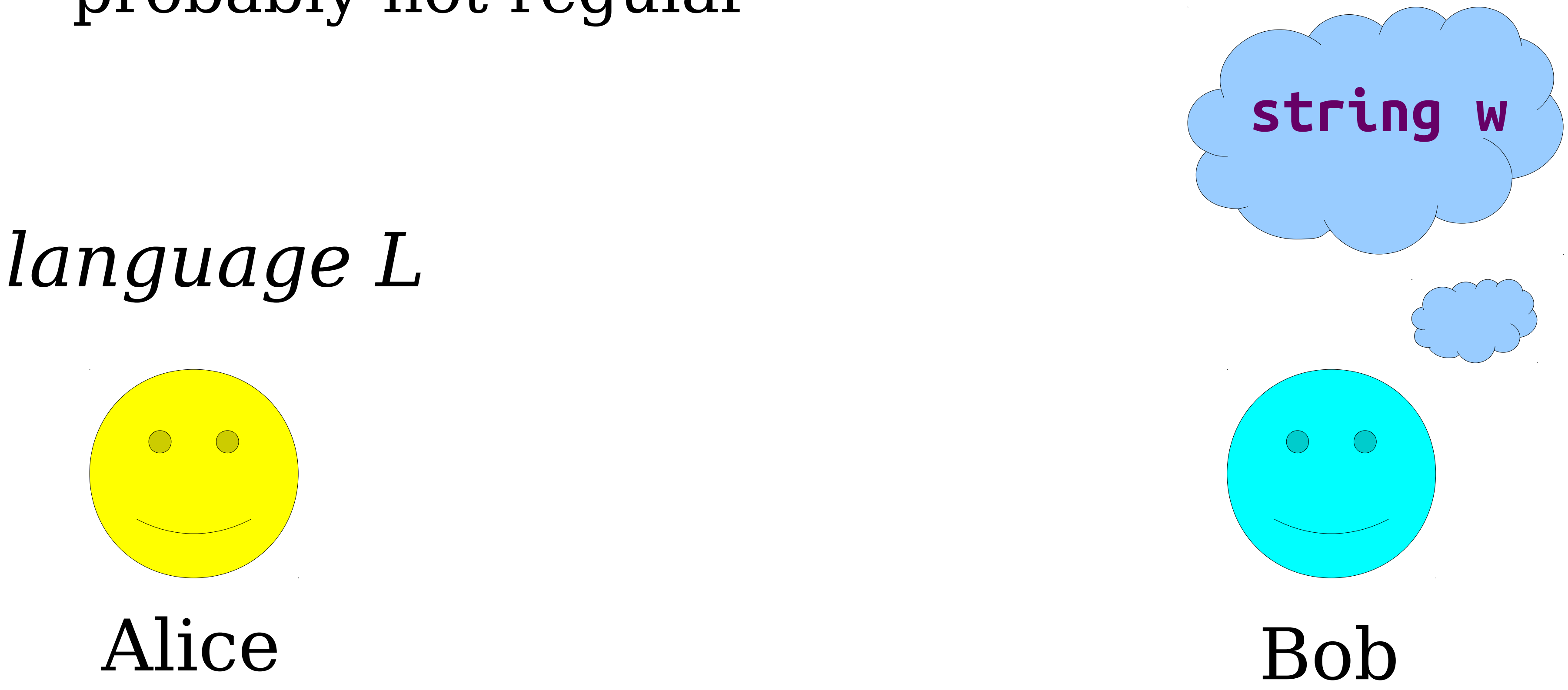
$$L = \{\ \mathbf{a}^n \mathbf{b}^n \mid n \in \mathbb{N}\ \}$$



As the size of Bob's string gets larger, the amount of memory Alice needs also increases. Since Bob's string could get unboundedly large, we need infinite memory.

An Analogy

Key insight: if Alice has to remember ***infinitely*** many things, or one of ***infinitely*** many possibilities, the language is probably not regular



Tying Everything Together

- One of the intuitions we hope you develop for DFAs is to have each state in a DFA represent some key piece of information the automaton has to remember.
- If you only need to remember one of finitely many pieces of information, that gives you a DFA.
  - ***You can formalize this!*** If we have time, we'll see this later this quarter. If not, and you're curious, take CS154!
- If you need to remember one of infinitely many pieces of information, you can use the Myhill-Nerode theorem to prove that the language has no DFA.

Where We Stand

Where We Stand

- We've ended up where we are now by trying to answer the question “what problems can you solve with a computer?”
- We defined a computer to be DFA, which means that the problems we can solve are precisely the regular languages.
- We've discovered several equivalent ways to think about regular languages (DFAs, NFAs, and regular expressions) and used that to reason about the regular languages.
- We now have a powerful intuition for where we ended up: DFAs are finite-memory computers, and regular languages correspond to problems solvable with finite memory.
- Putting all of this together, we have a much deeper sense for what finite memory computation looks like – *and what it doesn't look like!*

Where We're Going

- What does computation look like with unbounded memory?
- What problems can you solve with unbounded-memory computers?
- What does it even mean to “solve” such a problem?
- And how do we know the answers to any of these questions?

Next Time

- ***Context-Free Languages***
  - Context-Free Grammars
  - Generating Languages from Scratch