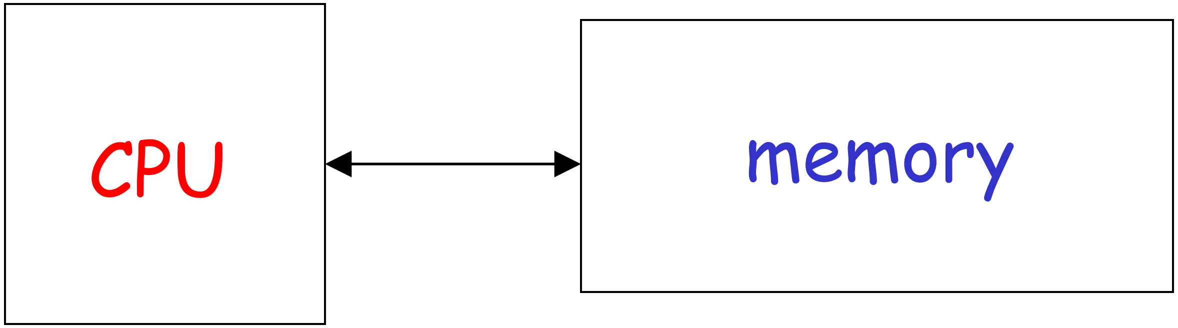
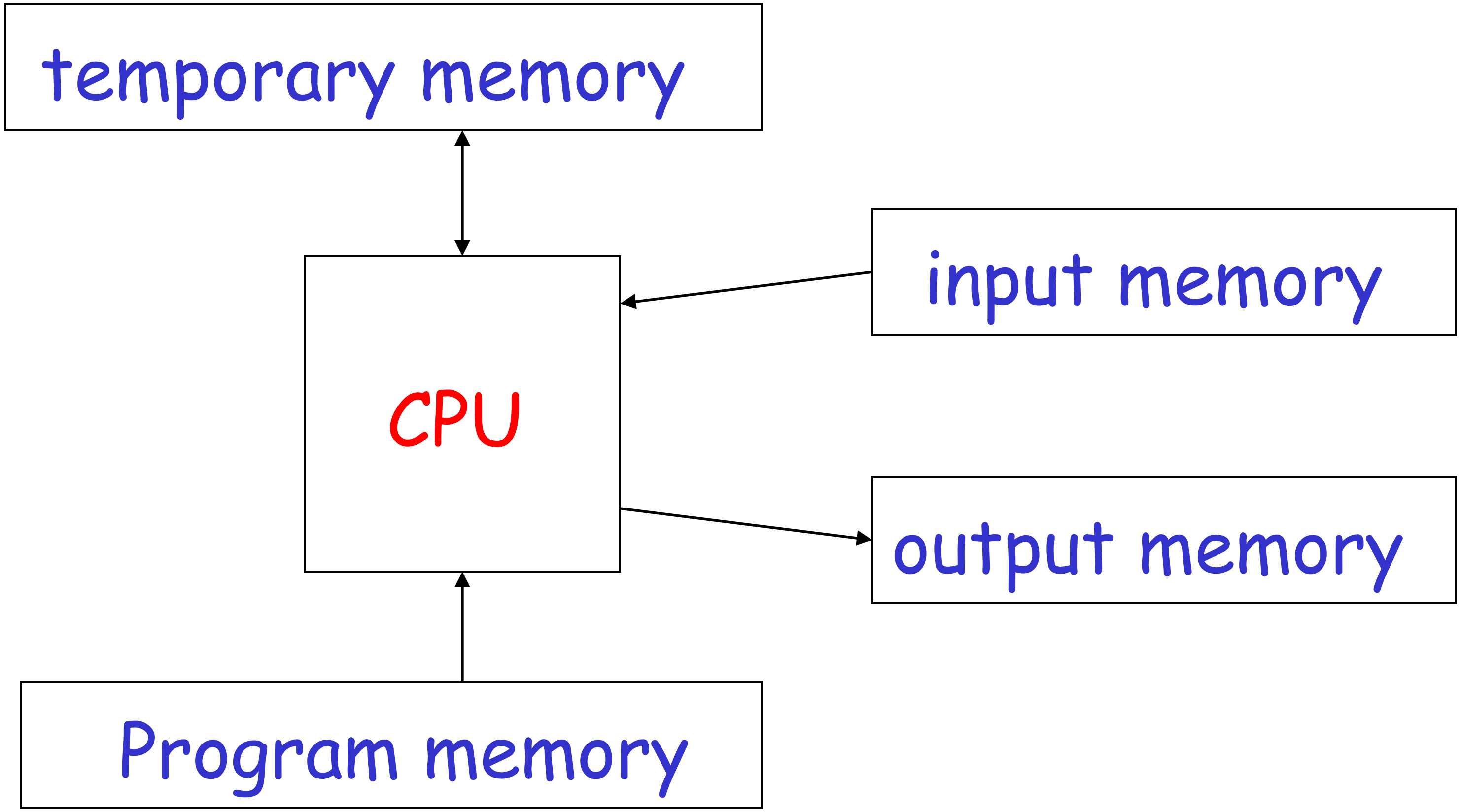
Models of Computation

Computation



CPU

memory



temporary memory

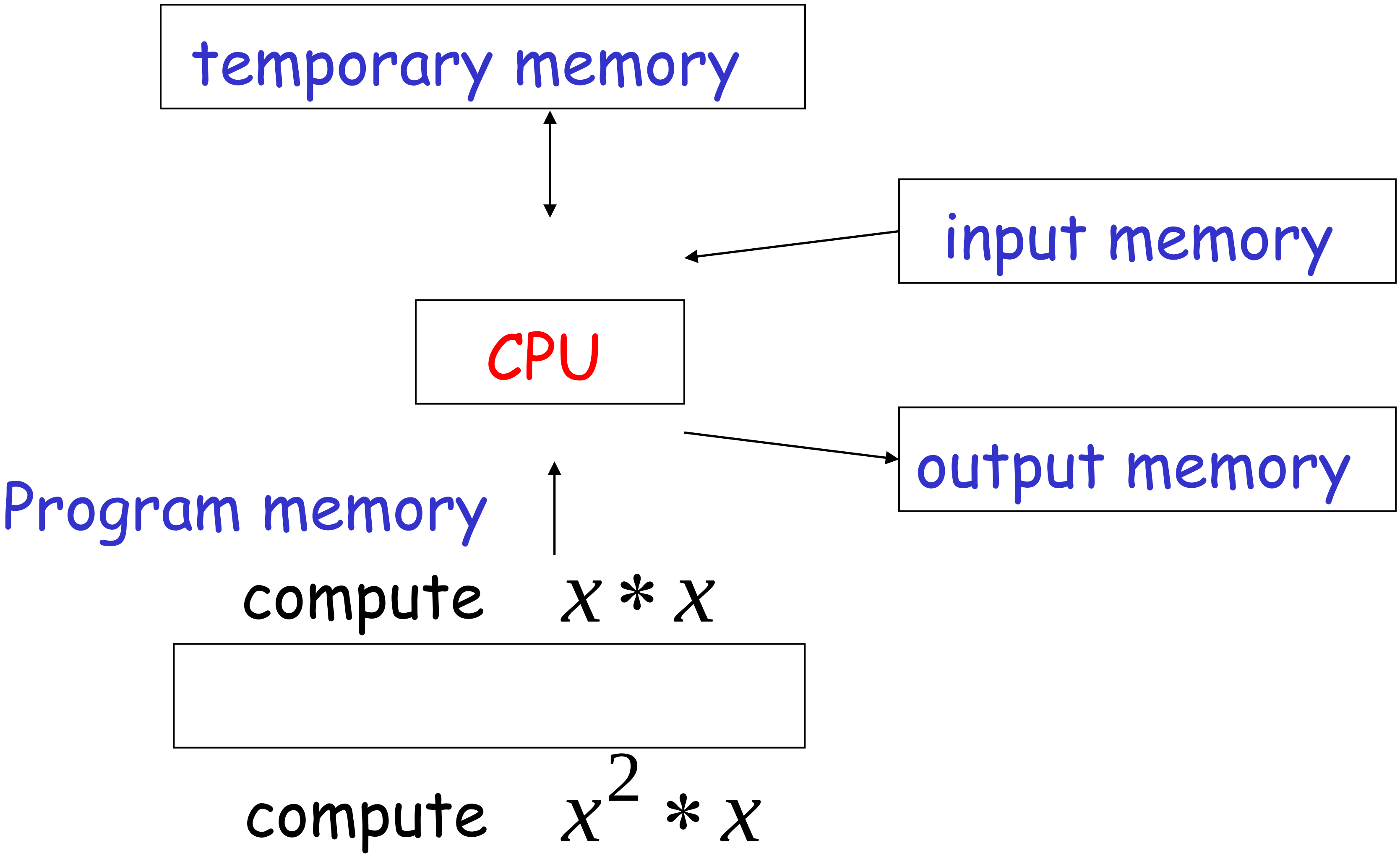
input memory

CPU

output memory

Program memory

Example: $f(x) = x^3$



temporary memory

input memory

CPU

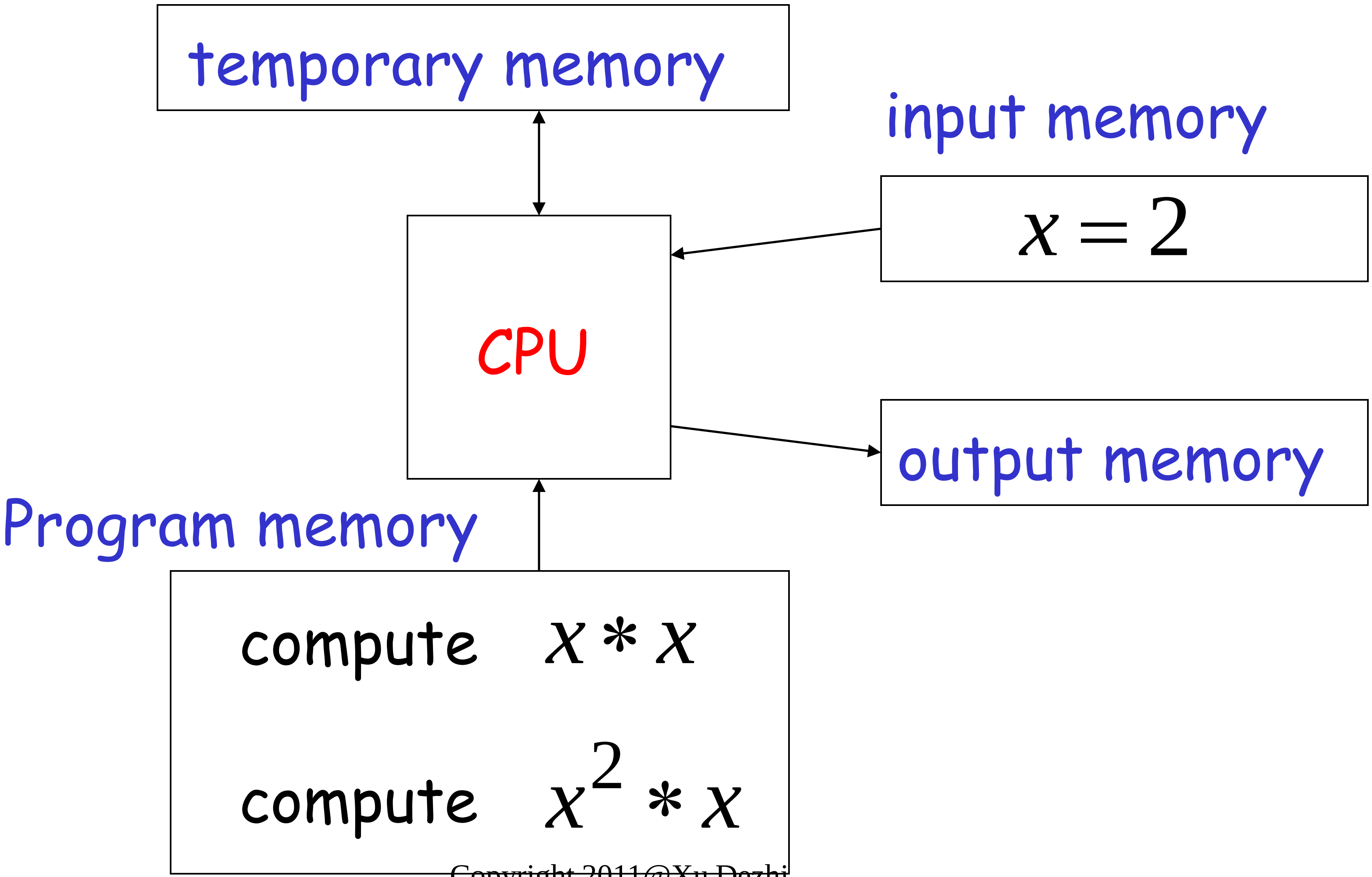
output memory

Program memory

compute $x * x$

compute $x^2 * x$

$$f(x) = x^3$$



temporary memory

input memory

$x = 2$

CPU

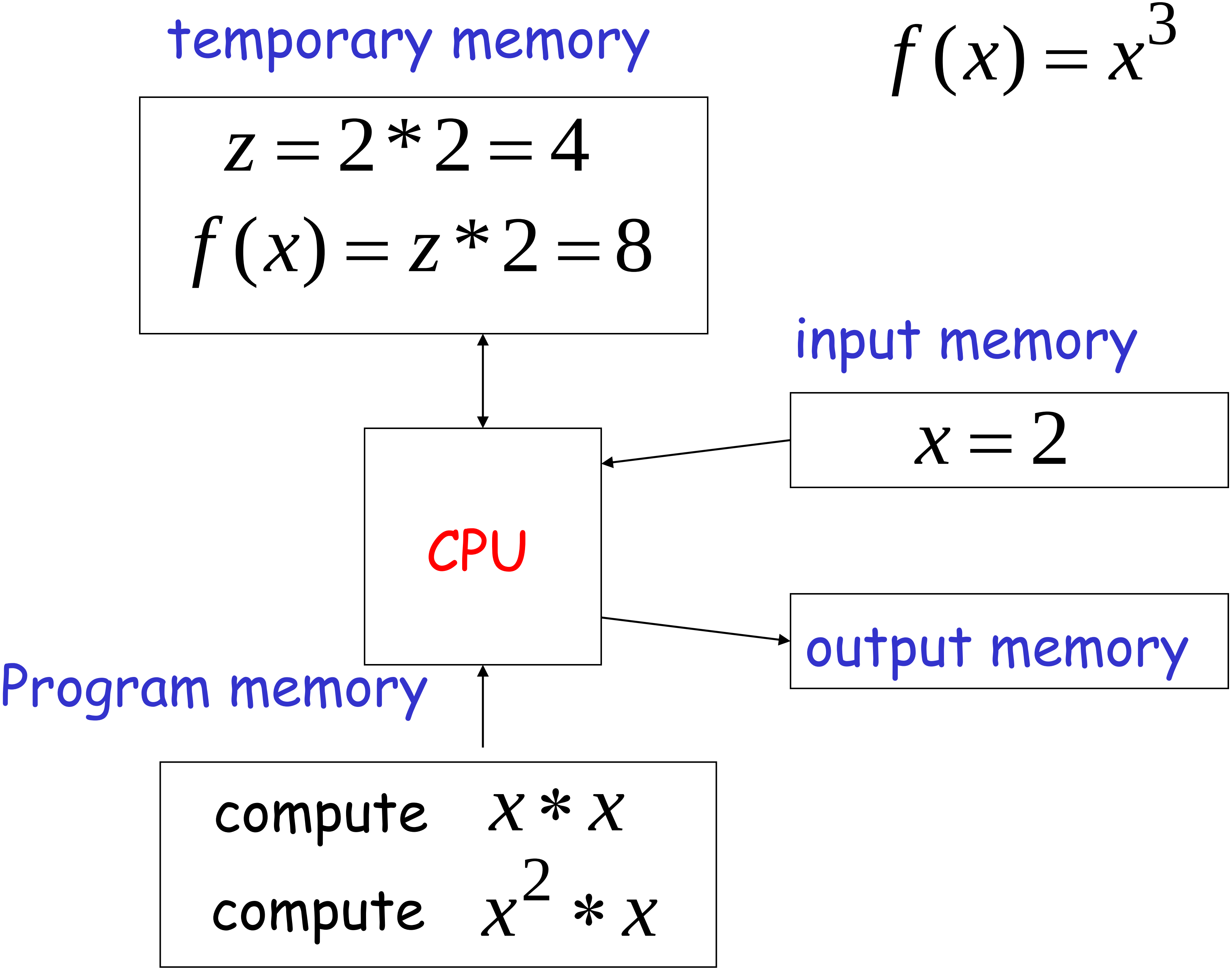
output memory

Program memory

compute $x * x$

compute $x^2 * x$

$$f(x) = x^3$$



temporary memory

$$z = 2*2 = 4$$

$$f(x) = z*2 = 8$$

input memory

$$x = 2$$

CPU

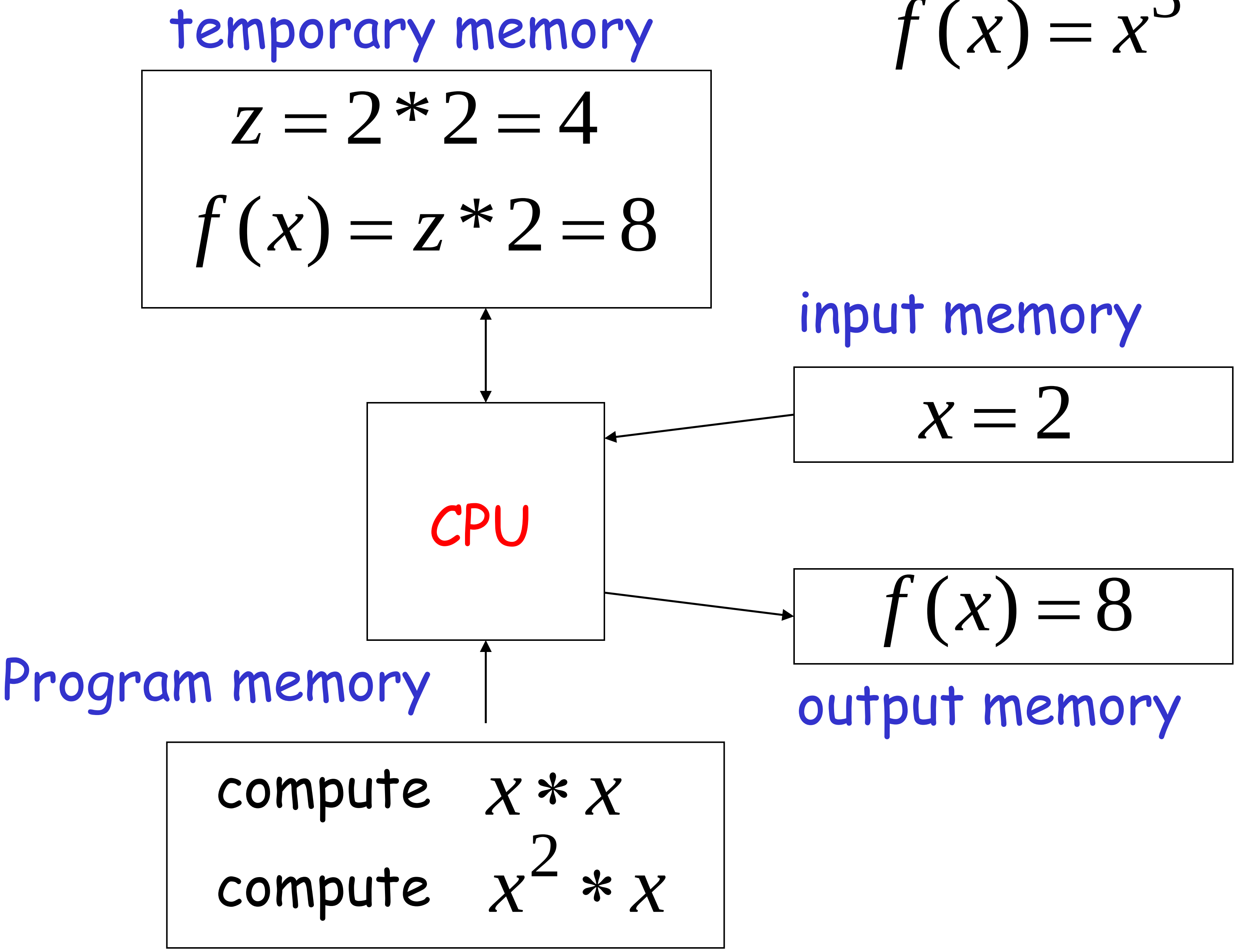
output memory

Program memory

compute $x*x$

compute x^2*x

$$f(x) = x^3$$



temporary memory

$$z = 2 * 2 = 4$$

$$f(x) = z * 2 = 8$$

input memory

$$x = 2$$

CPU

$$f(x) = 8$$

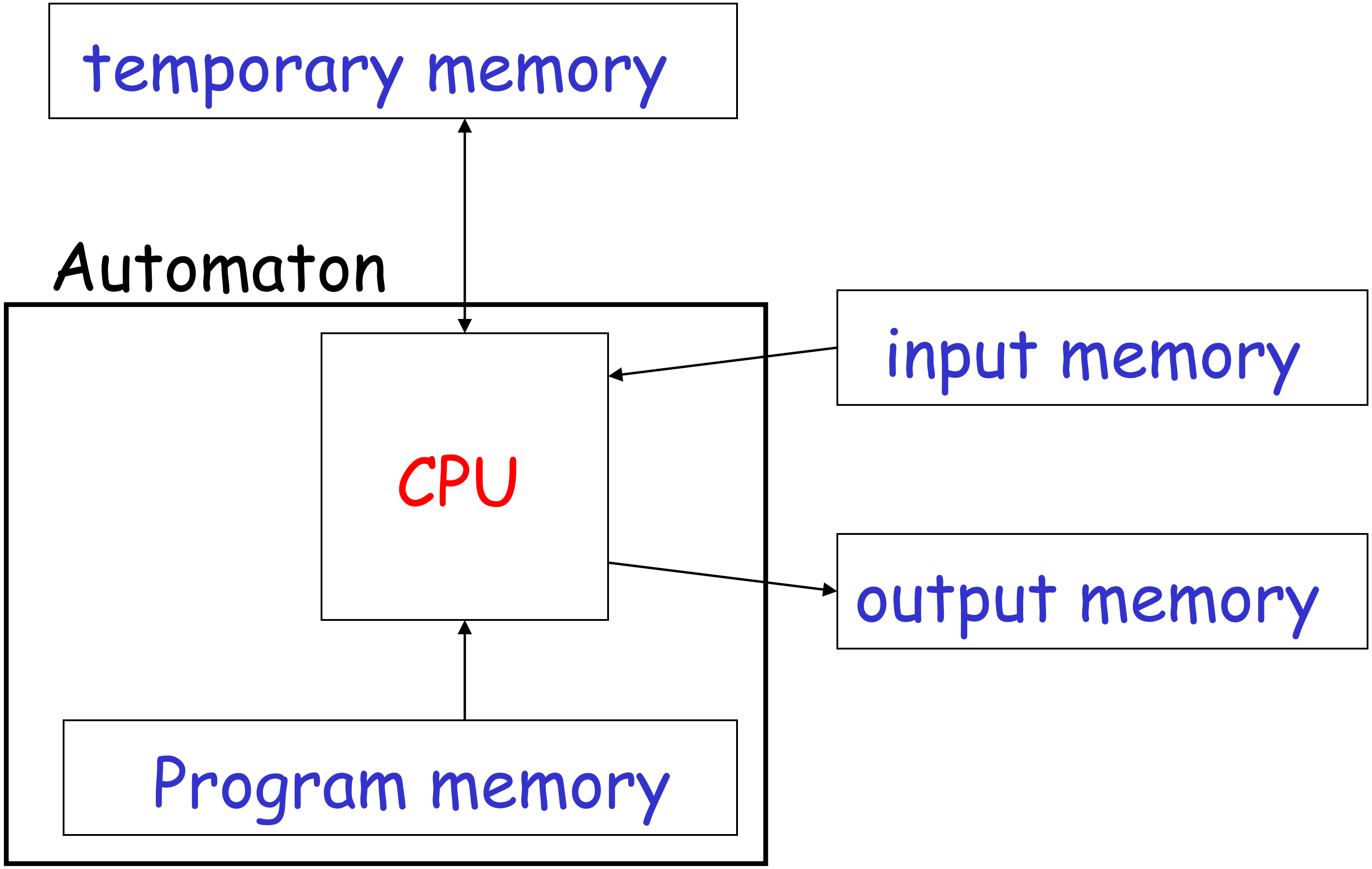
output memory

Program memory

compute $x * x$

compute $x^2 * x$

Automaton



temporary memory

Automaton

CPU

input memory

output memory

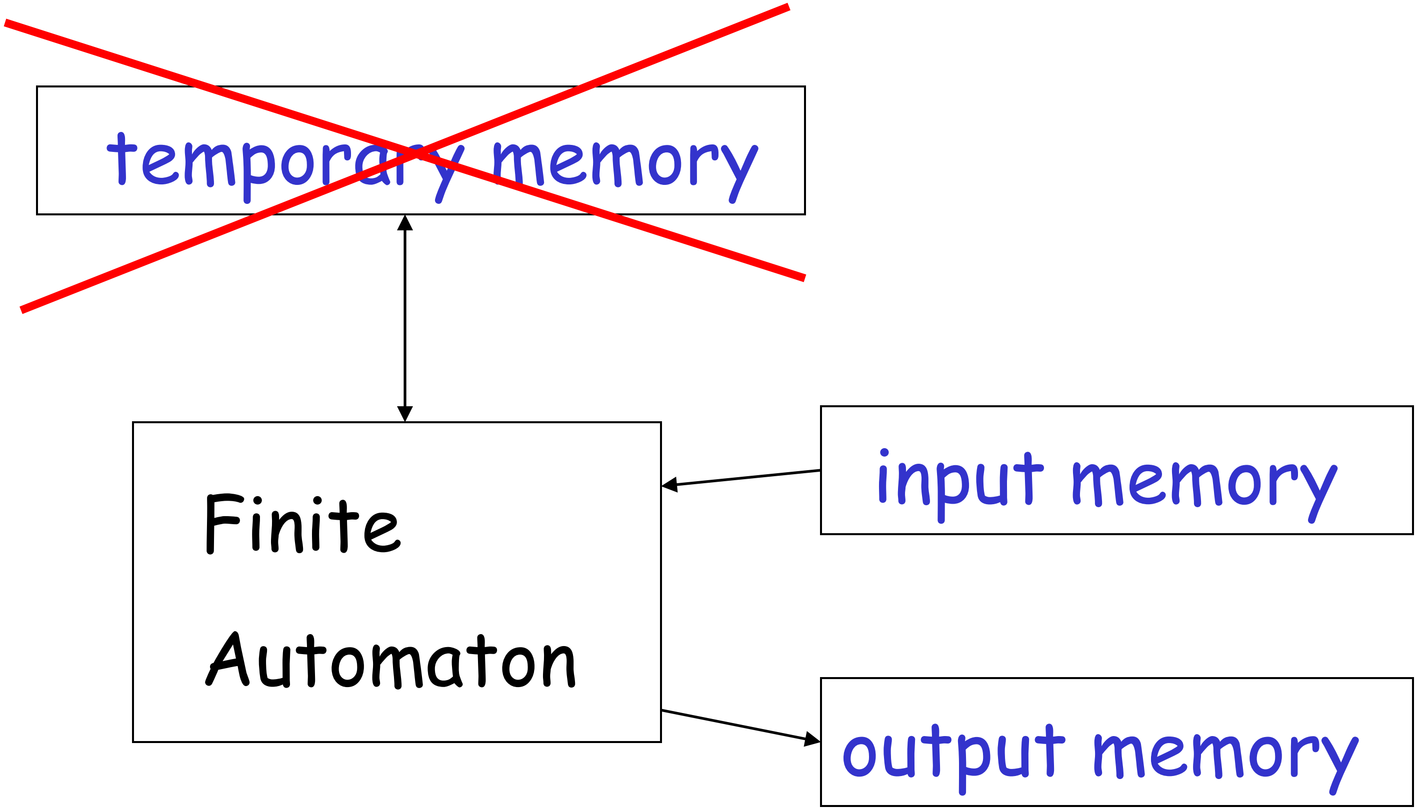
Program memory

Different Kinds of Automata

Automata are distinguished by the temporary memory

- **Finite Automata:** no temporary memory
- **Pushdown Automata:** stack
- **Turing Machines:** random access memory

Finite Automaton



~~temporary memory~~

Finite Automaton

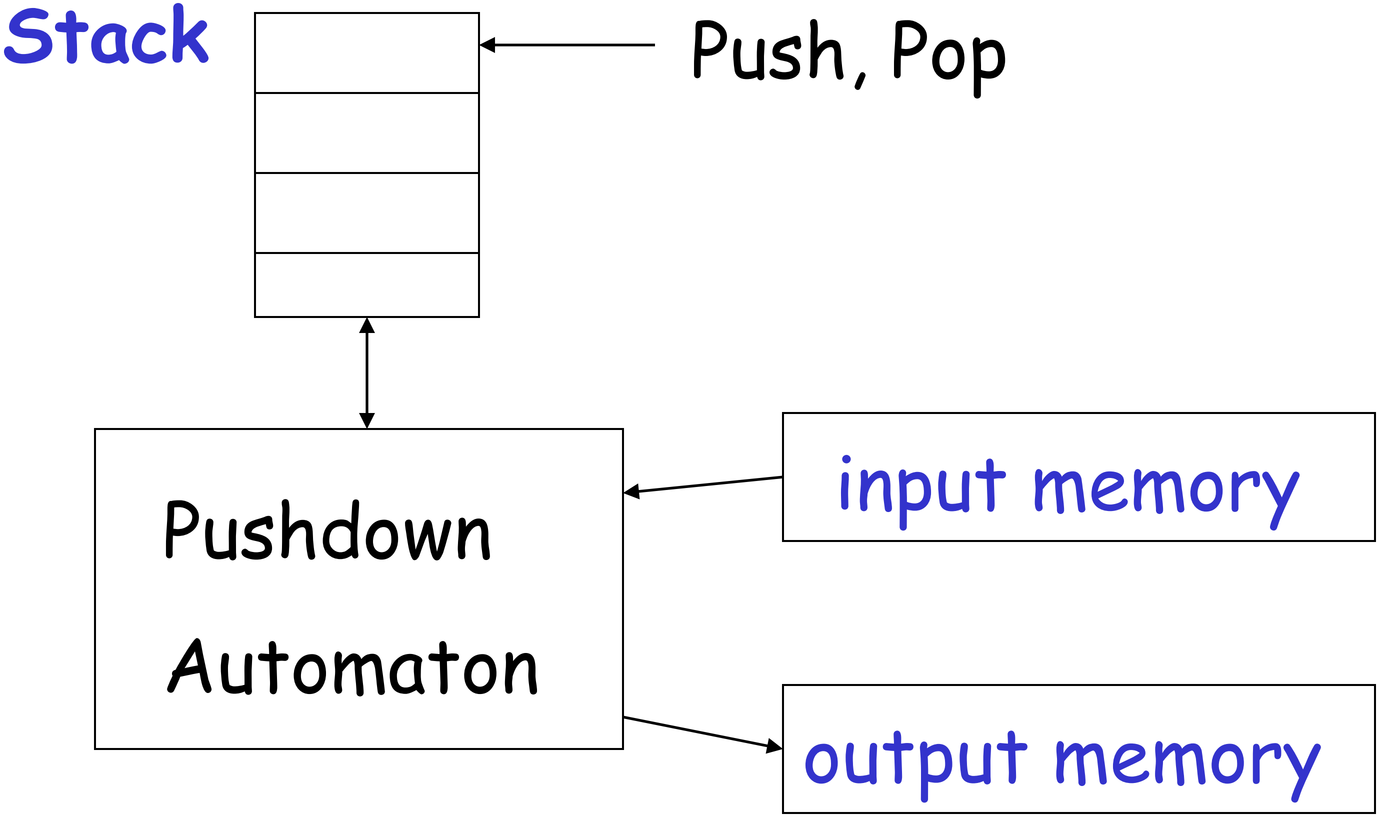
input memory

output memory

Example: Vending Machines

(small computing power)

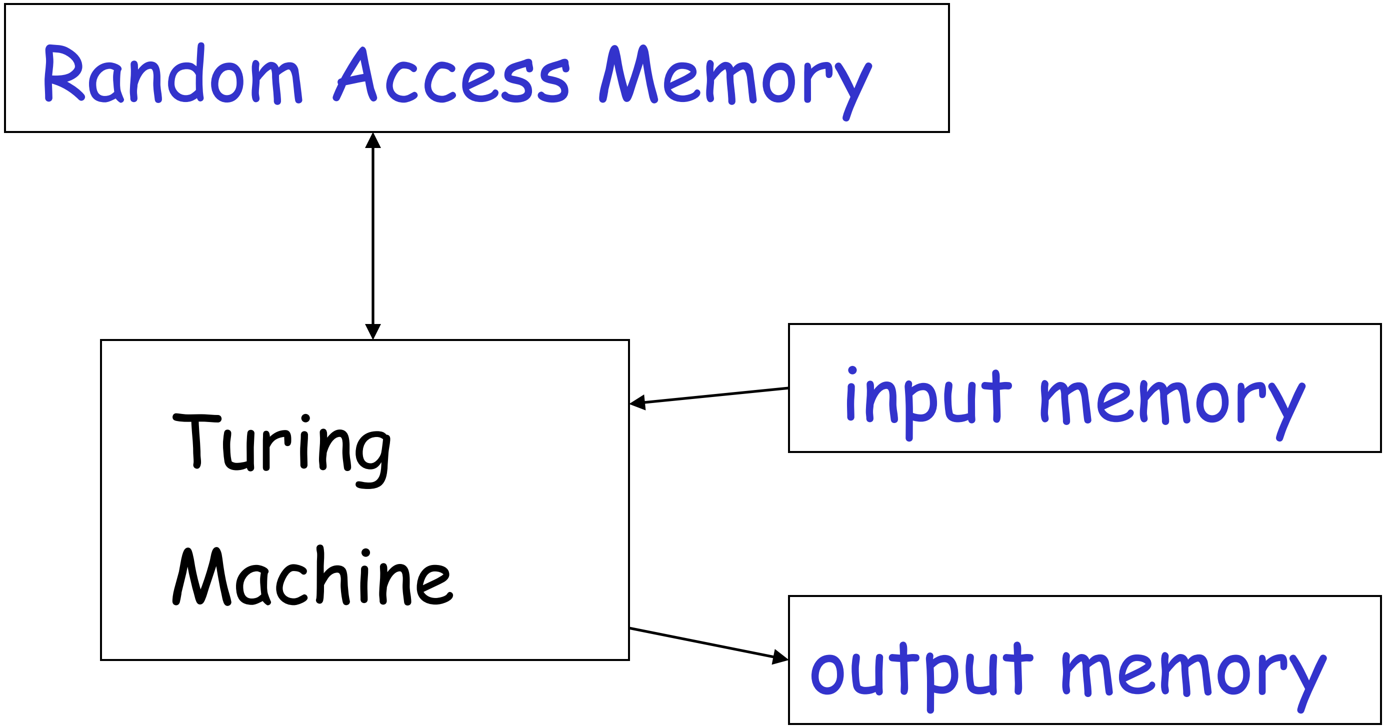
Pushdown Automaton



Example: Compilers for Programming Languages

(medium computing power)

Turing Machine



Random Access Memory

Turing Machine

input memory

output memory

Examples: Any Algorithm

(highest computing power)

Power of Automata

Finite Automata < Pushdown Automata < Turing Machine

Less power → More power

Solve more computational problems

The End

Mathematical Preliminaries

Mathematical Preliminaries

- Sets
- Functions
- Relations
- Graphs
- Proof Techniques

SETS

A set is a collection of elements

$$A = \{1, 2, 3\}$$

$$B = \{train, bus, bicycle, airplane\}$$

We write

$$1 \in A$$

$$ship \notin B$$

Set Representations

$C = \{ a, b, c, d, e, f, g, h, i, j, k \}$

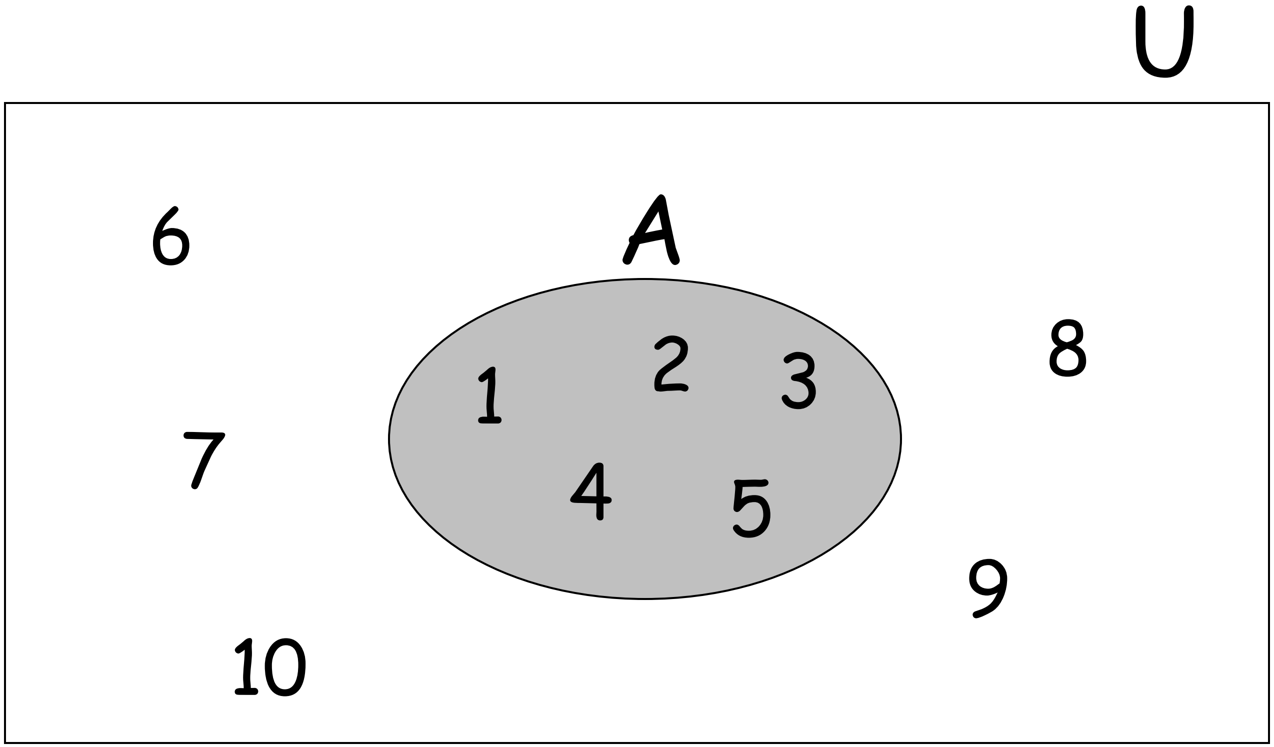
$C = \{ a, b, \ldots, k \}$ → *finite set*

$S = \{ 2, 4, 6, \ldots \}$ → *infinite set*

$S = \{ j : j > 0, \text{ and } j = 2k \text{ for some } k>0 \}$

$S = \{ j : j \text{ is nonnegative and even} \}$

$A = \{ 1, 2, 3, 4, 5 \}$



Universal Set: all possible elements

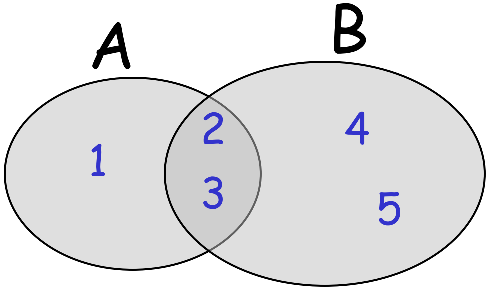
$U = \{ 1 , \ldots , 10 \}$

Set Operations

$A = \{ 1, 2, 3 \}$ $B = \{ 2, 3, 4, 5\}$

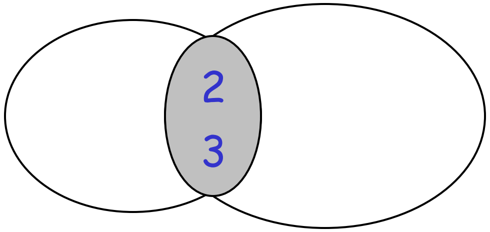
- Union

$A \cup B = \{ 1, 2, 3, 4, 5 \}$



- Intersection

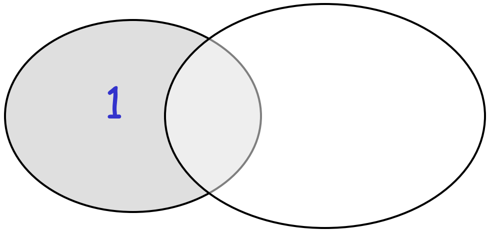
$A \cap B = \{ 2, 3 \}$



- Difference

$A - B = \{ 1 \}$

$B - A = \{ 4, 5 \}$

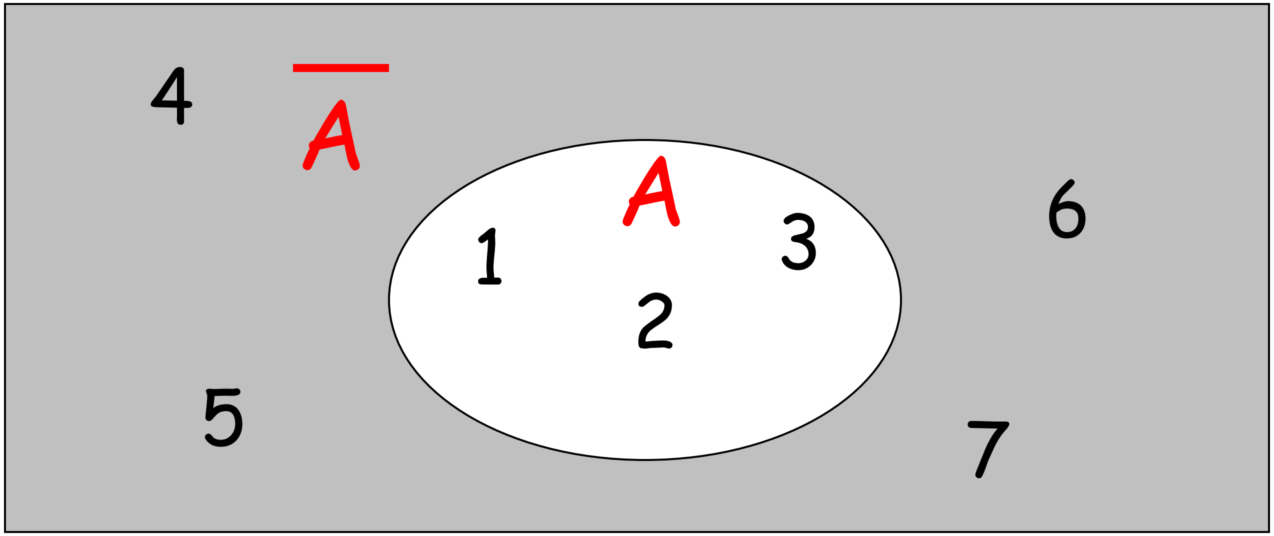


Venn diagrams

- Complement

Universal set = {1, ..., 7}

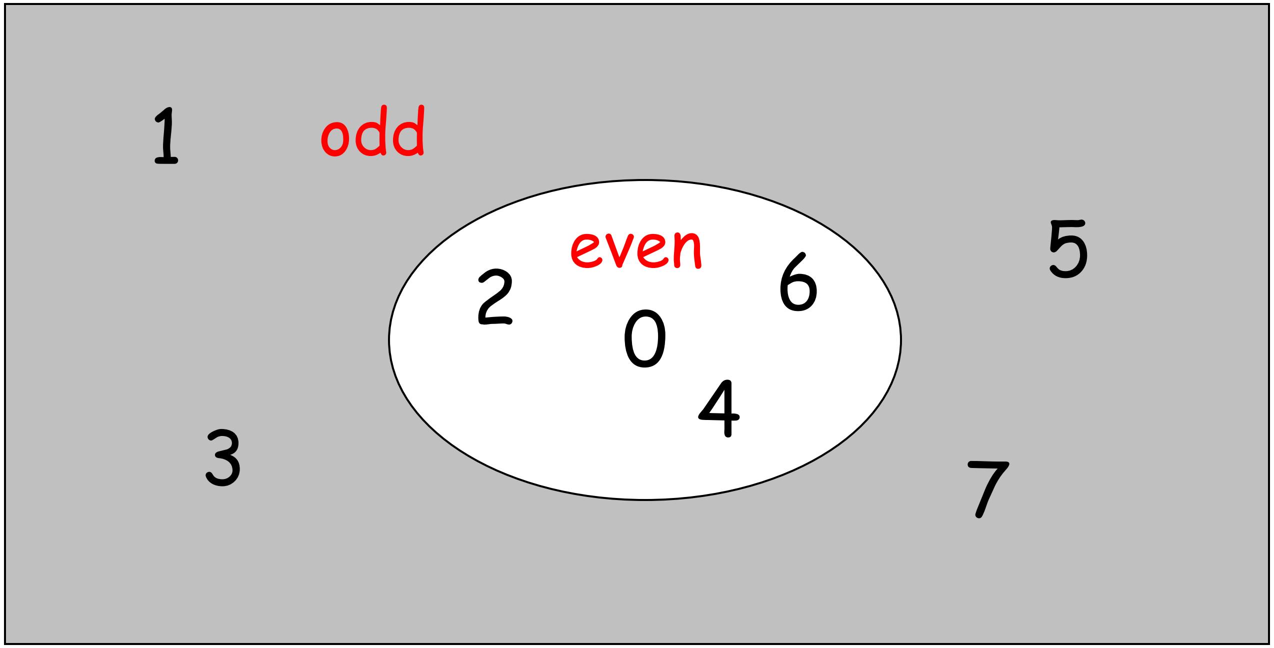
$A = \{ 1, 2, 3 \} \rightarrow \overline{A} = \{ 4, 5, 6, 7\}$



$\overline{\overline{A}} = A$

$$\overline{\{\text{ even integers }\}} = \{\text{ odd integers }\}$$

Integers



DeMorgan's Laws

$$\overline{A \cup B} = \overline{A} \cap \overline{B}$$

$$\overline{A \cap B} = \overline{A} \cup \overline{B}$$

Empty, Null Set: $\emptyset$

$\emptyset = \{ \}$

$S \cup \emptyset = S$

$S \cap \emptyset = \emptyset$

$S - \emptyset = S$

$\emptyset - S = \emptyset$

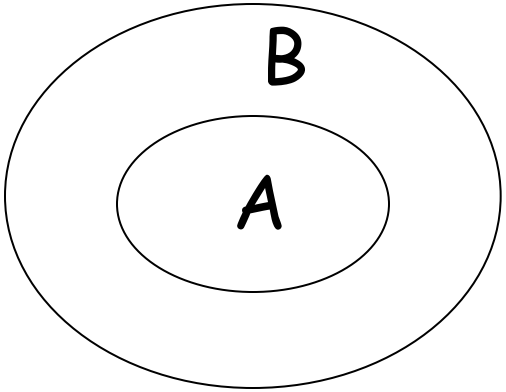
$\overline{\emptyset}$ = Universal Set

Subset

$A = \{ 1, 2, 3\}$　　　　$B = \{ 1, 2, 3, 4, 5 \}$

$$A \subseteq B$$

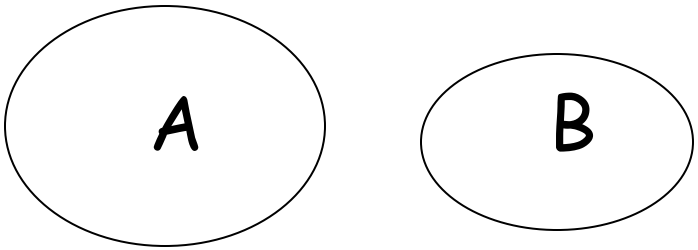
Proper Subset: $A \subset B$



Disjoint Sets

$A = \{ 1, 2, 3 \}$ $B = \{ 5, 6 \}$

$A \cap B = \emptyset$



Set Cardinality

- For finite sets

$A = \{ 2, 5, 7 \}$

$|A| = 3$

(set size)

Powersets

A powerset is a set of sets

$S = \{ a, b, c \}$

Powerset of S = the set of all the subsets of S

$2^S = \{ \emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\} \}$

Observation: $| 2^S | = 2^{|S|}$ $(8 = 2^3)$

Cartesian Product

A = { 2, 4 }   B = { 2, 3, 5 }

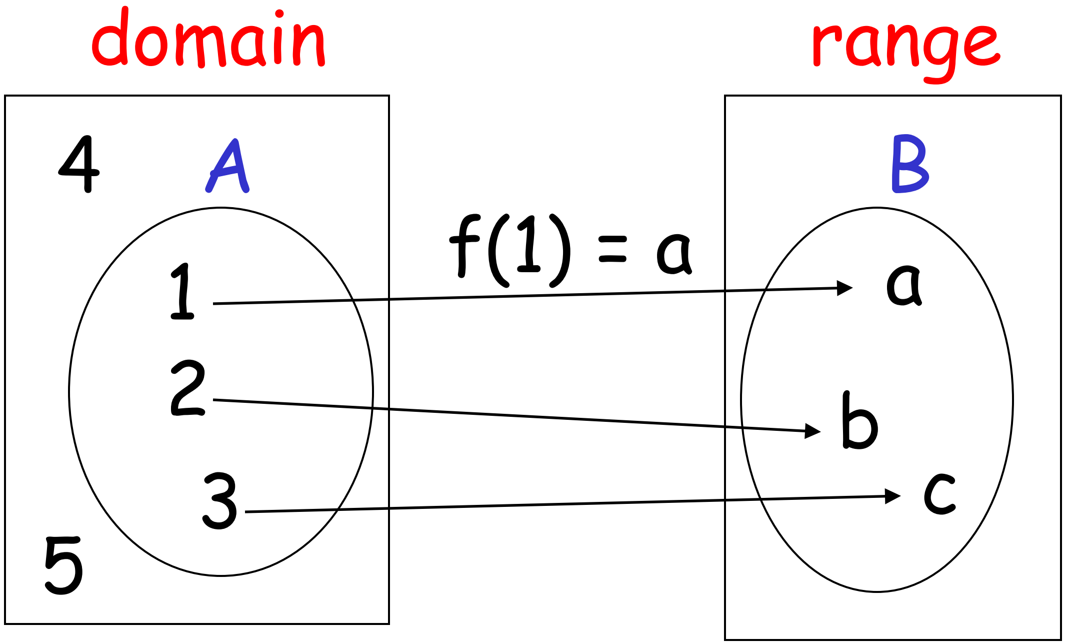
A X B = { (2, 2), (2, 3), (2, 5), (4, 2), (4, 3), (4, 5) }

|A X B| = |A| |B|

Generalizes to more than two sets

A X B X … X Z

Functions



f : A -> B

If A = domain

then f is a total function

otherwise f is a partial function

Relations

$R = \{(x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots\}$

$x_i \; R \; y_i$

e. g. if R = '>': $2 > 1$, $3 > 2$, $3 > 1$

Equivalence Relations

- Reflexive: $x \, R \, x$
- Symmetric: $x \, R \, y \Rightarrow y \, R \, x$
- Transitive: $x \, R \, y$ and $y \, R \, z \Rightarrow x \, R \, z$

Example: R = '='

- $x = x$
- $x = y \Rightarrow y = x$
- $x = y$ and $y = z \Rightarrow x = z$

Equivalence Classes

For equivalence relation R

equivalence class of x = {y : x R y}

Example:

R = { (1, 1), (2, 2), (1, 2), (2, 1),

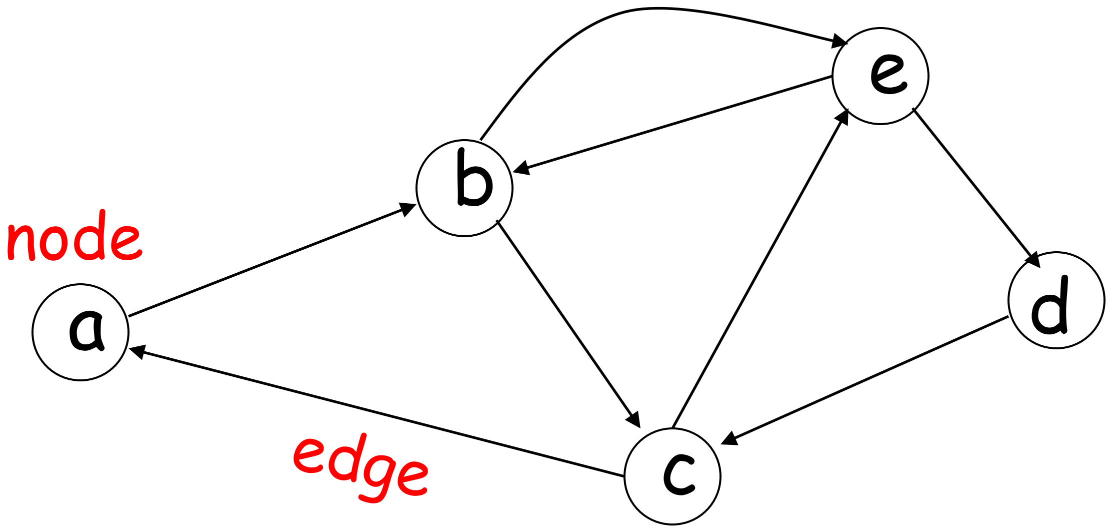
(3, 3), (4, 4), (3, 4), (4, 3) }

Equivalence class of 1 = {1, 2}

Equivalence class of 3 = {3, 4}

Graphs

A directed graph



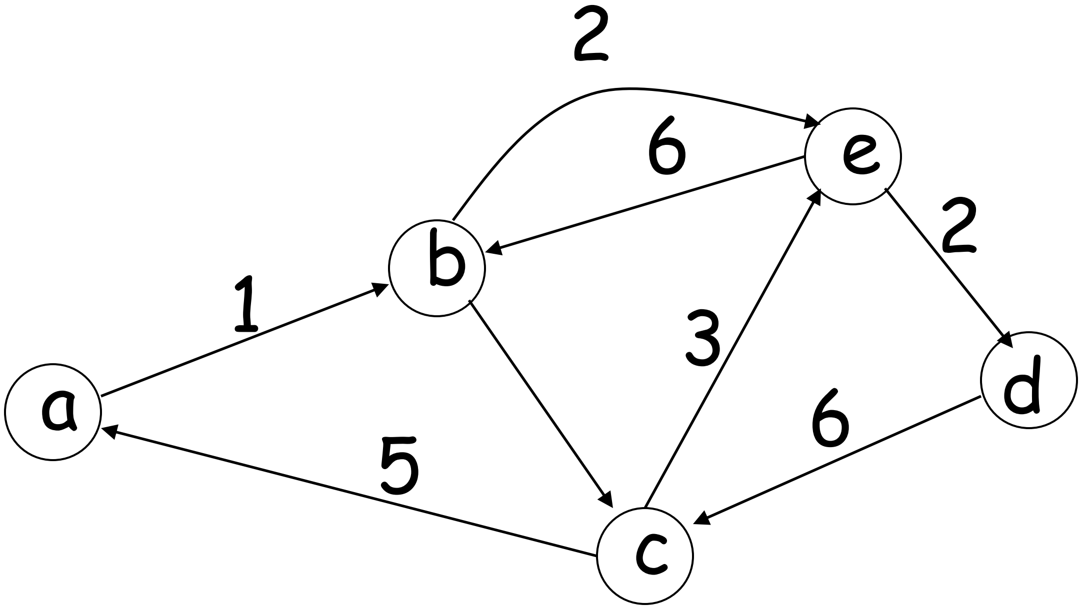
- Nodes (Vertices)

V = { a, b, c, d, e }

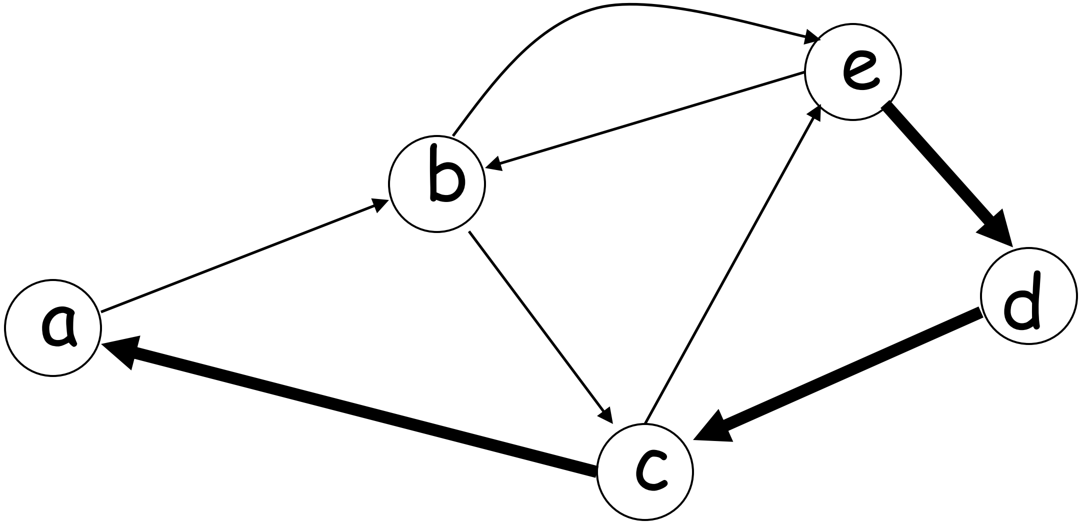
- Edges

E = { (a,b), (b,c), (b,e),(c,a), (c,e), (d,c), (e,b), (e,d) }

Labeled Graph



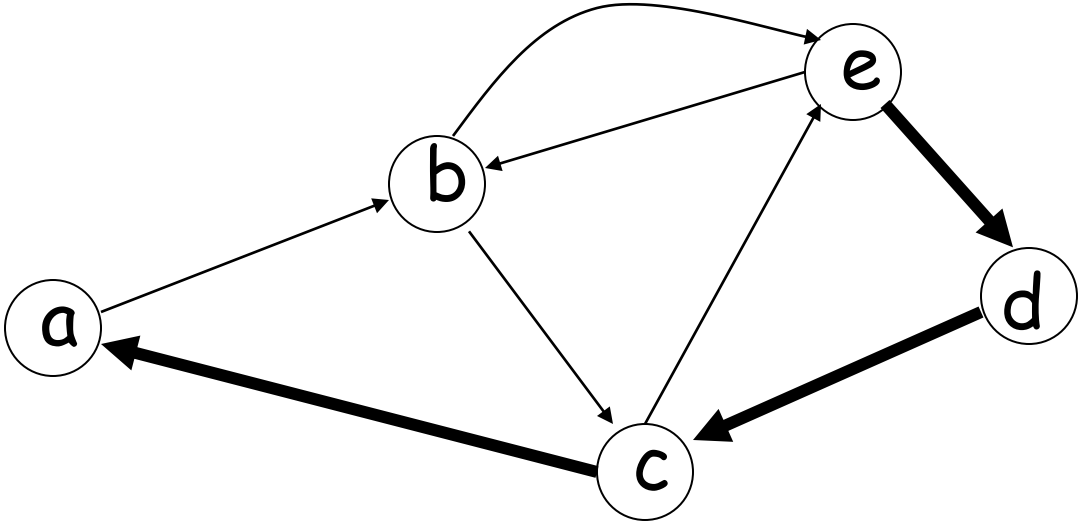
Walk



Walk is a sequence of adjacent edges

(e, d), (d, c), (c, a)

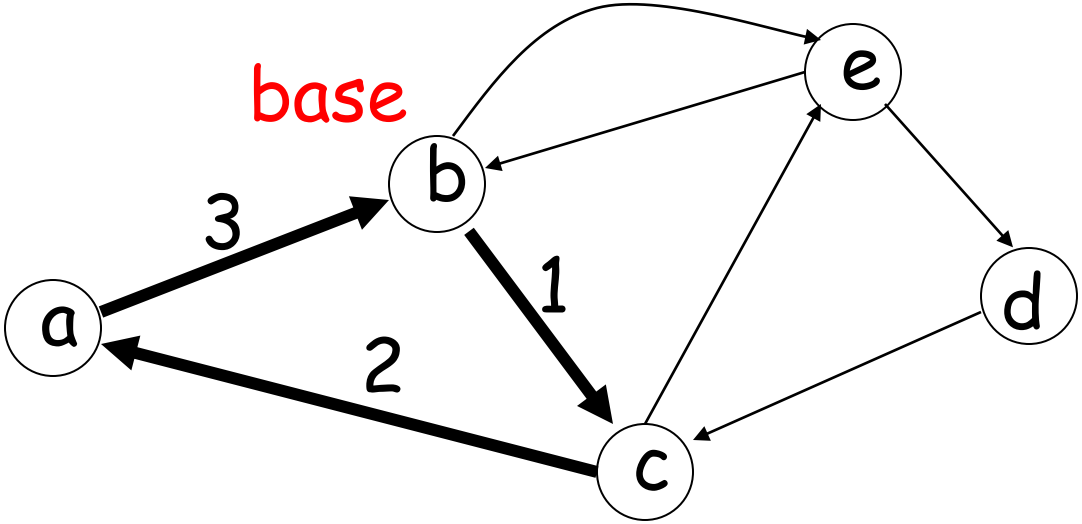
Path



Path is a walk where no edge is repeated

Simple path: no node is repeated

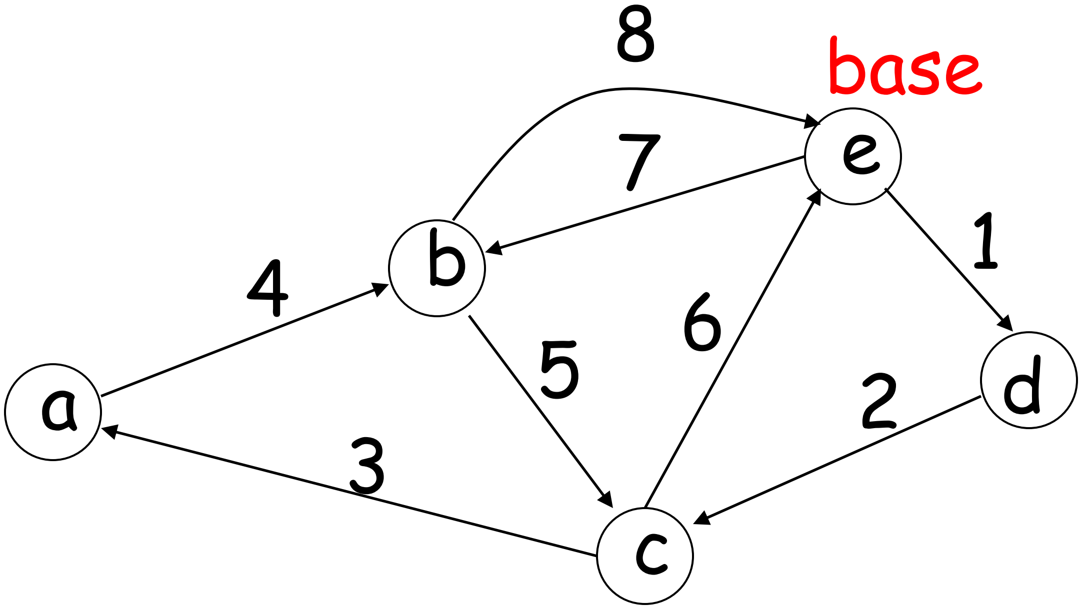
Cycle



Cycle: a walk from a node (base) to itself

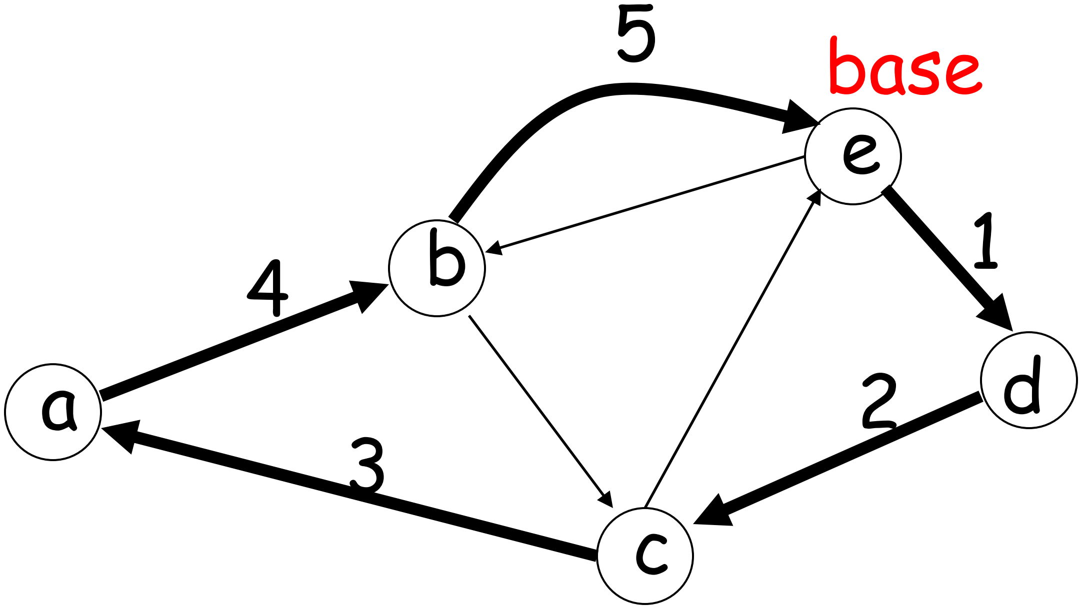
Simple cycle: only the base node is repeated

Euler Tour



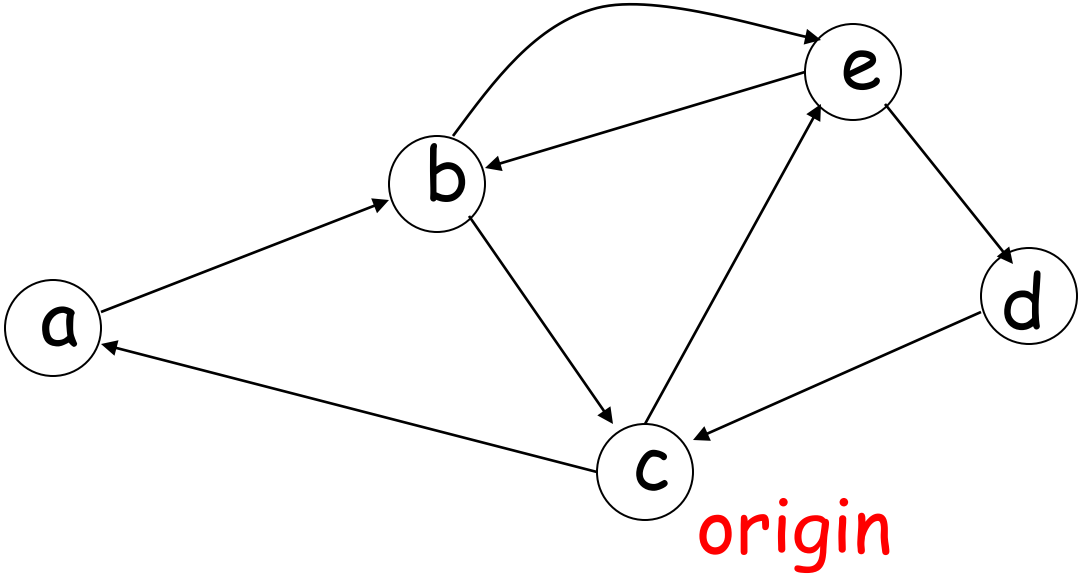
A cycle that contains each edge once

Hamiltonian Cycle

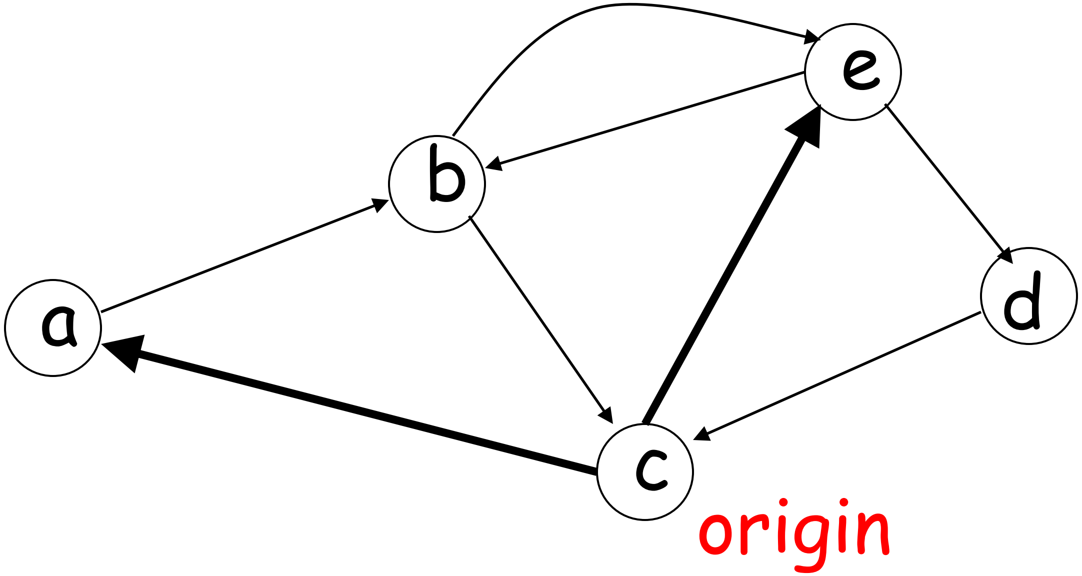


A simple cycle that contains all nodes

Finding All Simple Paths



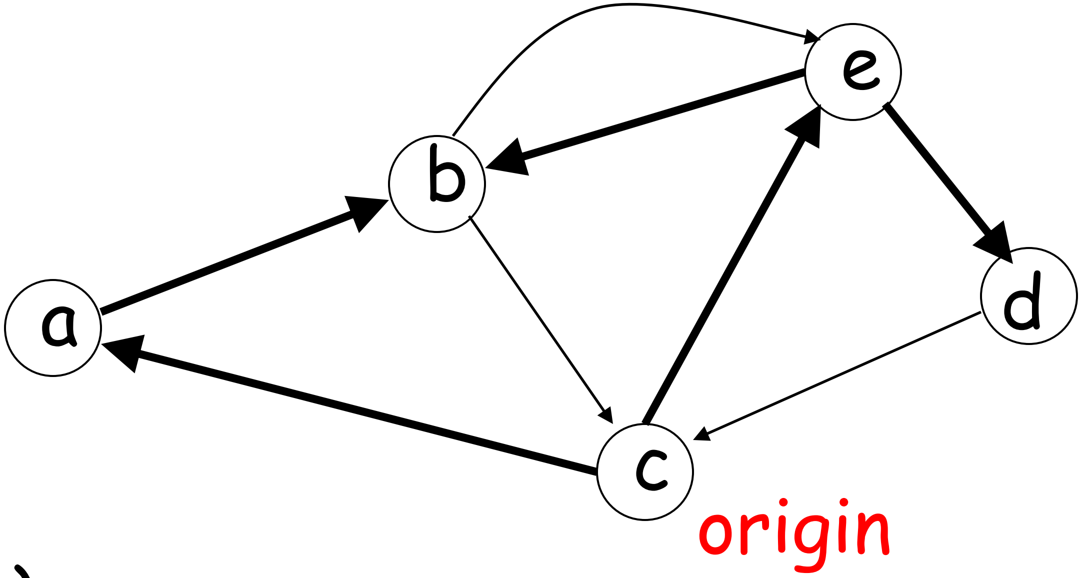
Step 1



(c, a)

(c, e)

Step 2



(c, a)

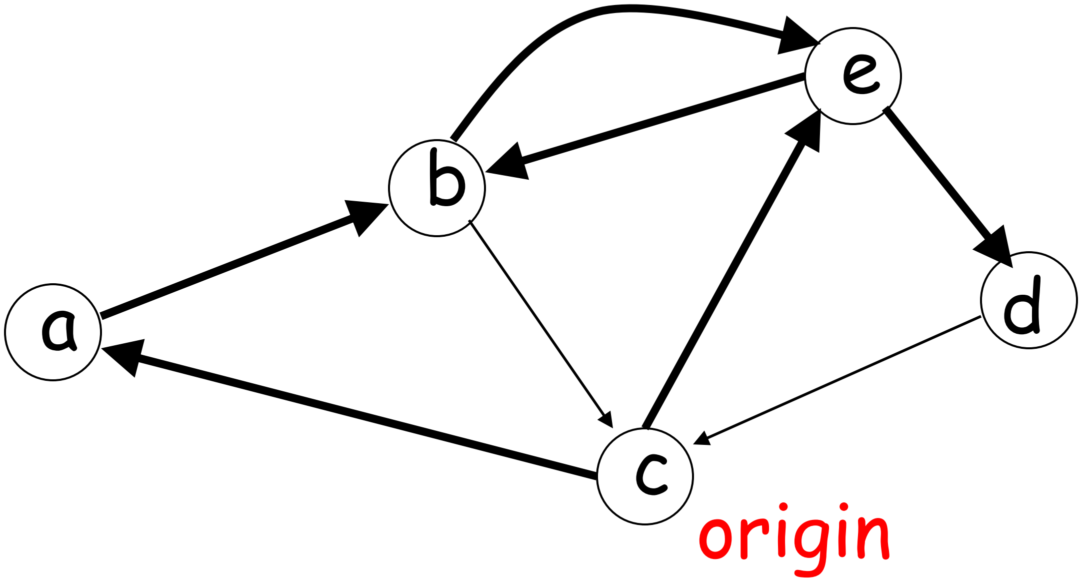
(c, a), (a, b)

(c, e)

(c, e), (e, b)

(c, e), (e, d)

Step 3



(c, a)

(c, a), (a, b)

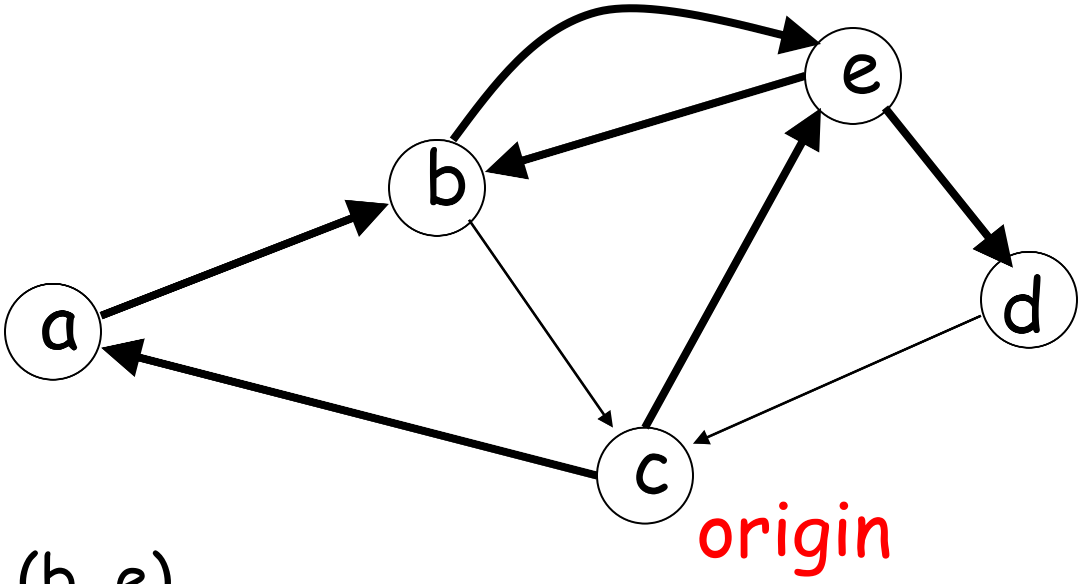
(c, a), (a, b), (b, e)

(c, e)

(c, e), (e, b)

(c, e), (e, d)

Step 4



(c, a)

(c, a), (a, b)

(c, a), (a, b), (b, e)

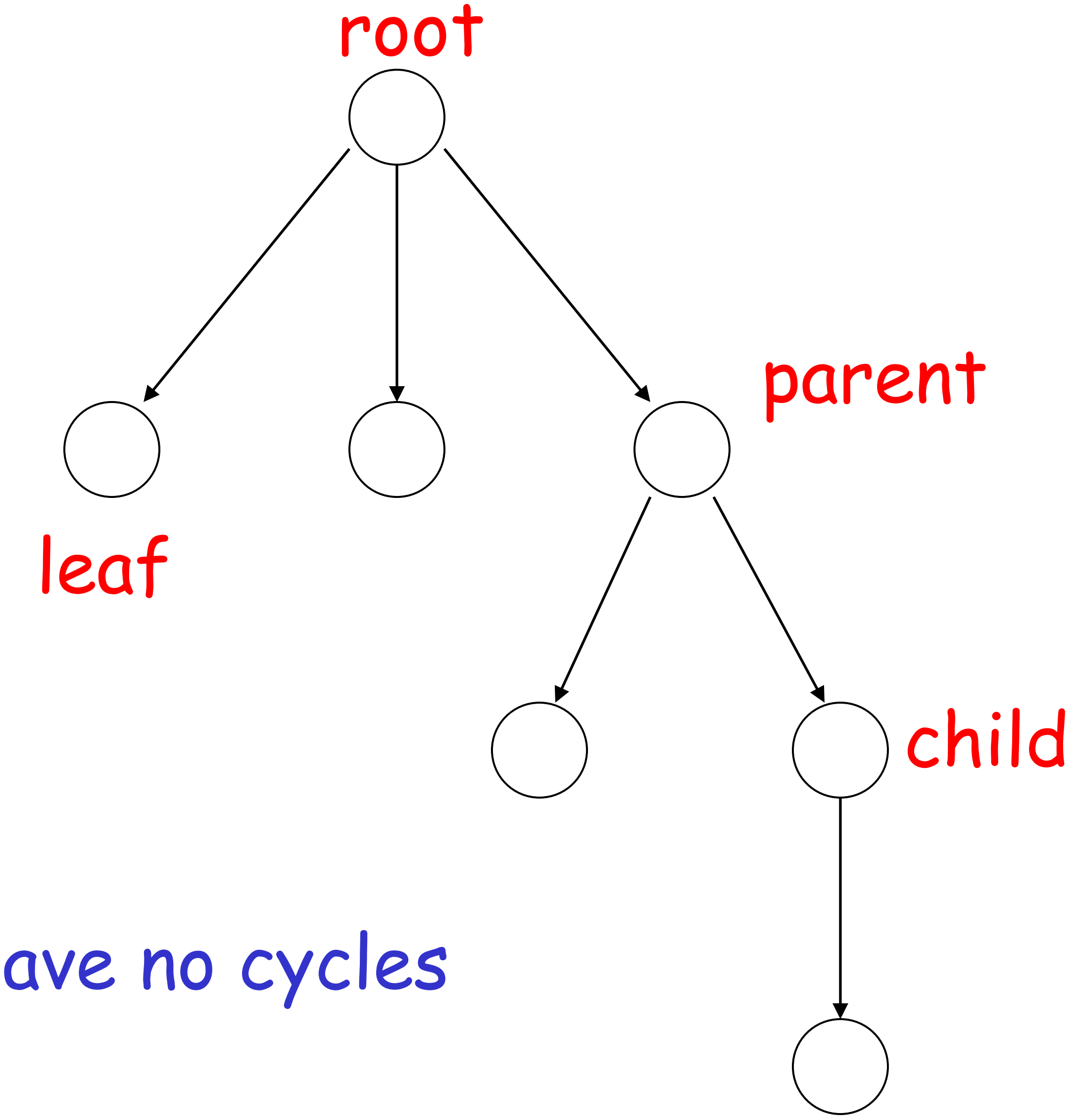
(c, a), (a, b), (b, e), (e,d)

(c, e)

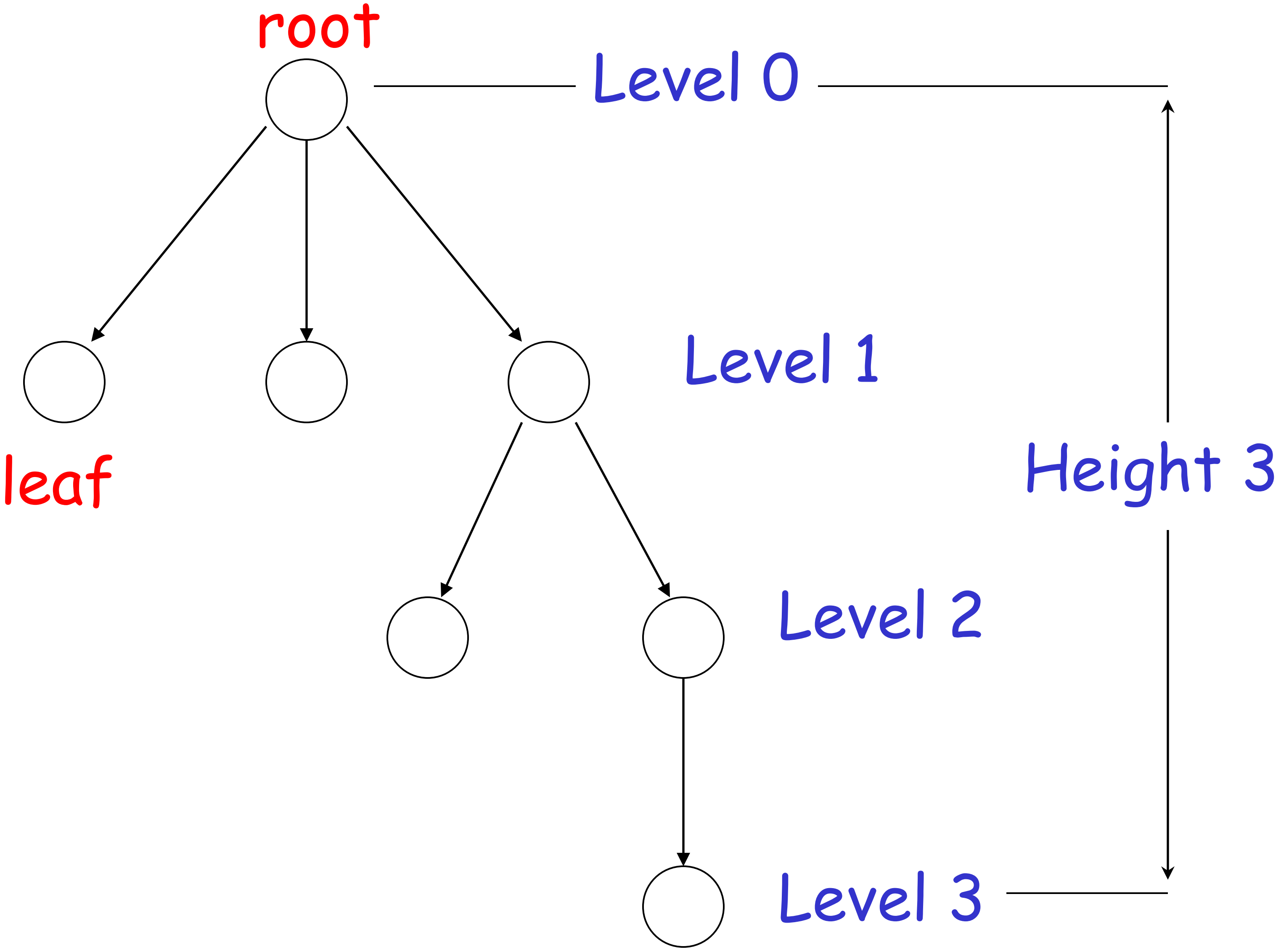
(c, e), (e, b)

(c, e), (e, d)

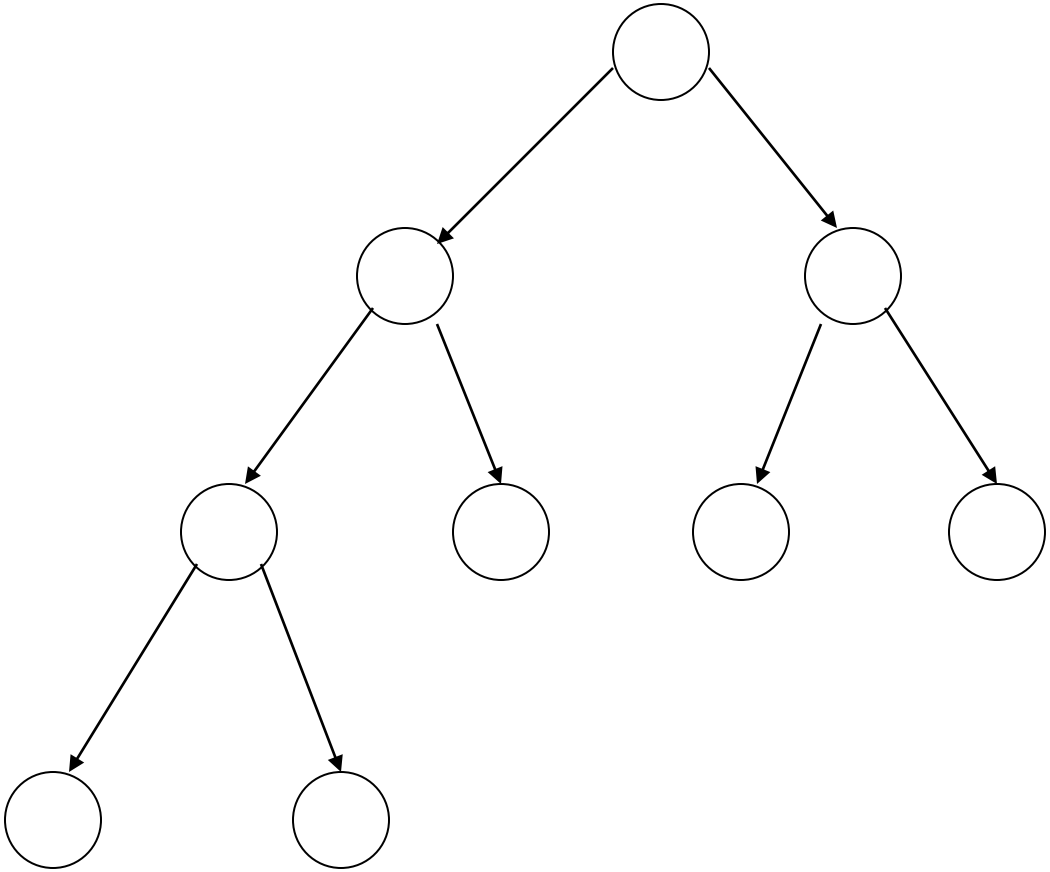
Trees



Trees have no cycles


root
Level 0
Level 1
leaf
Height 3
Level 2
Level 3

Binary Trees



Proof Techniques

- Proof by induction

- Proof by contradiction

- Proof by construction

Induction

We have statements $P_1, P_2, P_3, \ldots$

If we know

- for some b that $P_1, P_2, \ldots, P_b$ are true
- for any k >= b that

$$P_1, P_2, \ldots, P_k \text{ imply } P_{k+1}$$

Then

Every P_i is true

Proof by Induction

- Inductive basis

  Find P_1, P_2, …, P_b which are true

- Inductive hypothesis

  Let's assume P_1, P_2, …, P_k are true,

  for any k >= b

- Inductive step

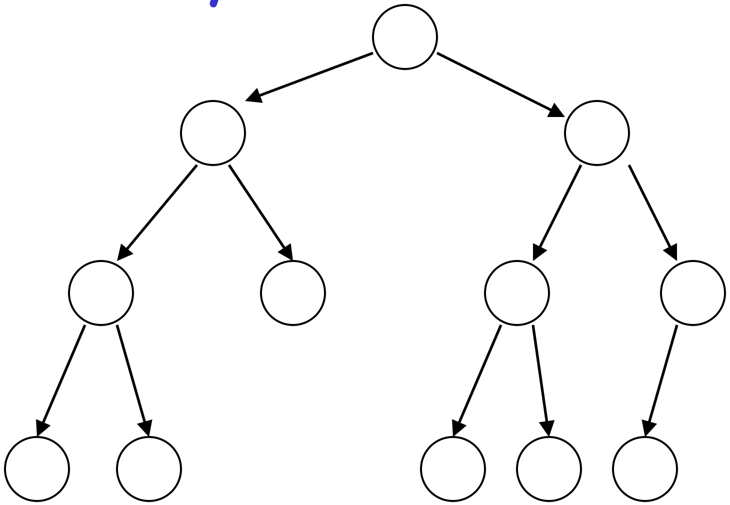
  Show that P_{k+1} is true

Example

Theorem: A binary tree of height n has at most 2^n leaves.

Proof by induction:

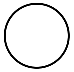
let $L(i)$ be the maximum number of leaves of any subtree at height i



We want to show: $L(i) <= 2^i$

- Inductive basis

  $L(0) = 1$ (the root node)



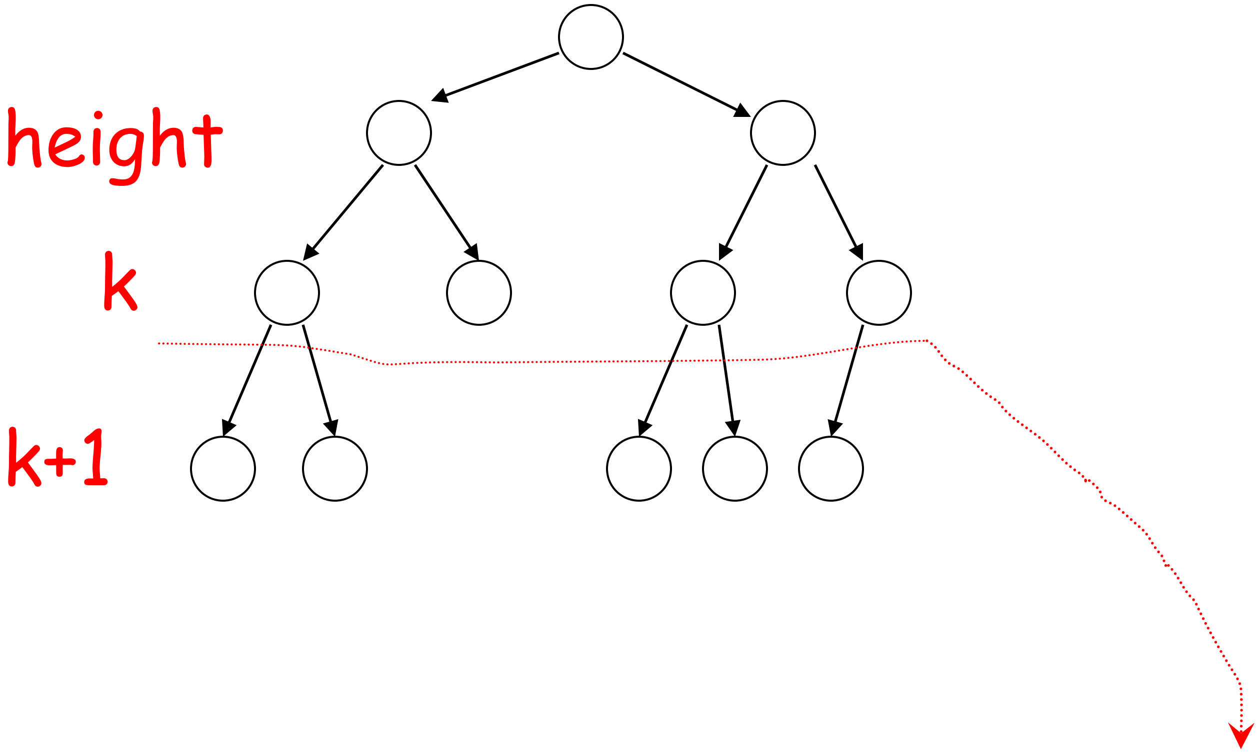
- Inductive hypothesis

  Let's assume $L(i) <= 2^i$ for all $i = 0, 1, \ldots, k$

- Induction step

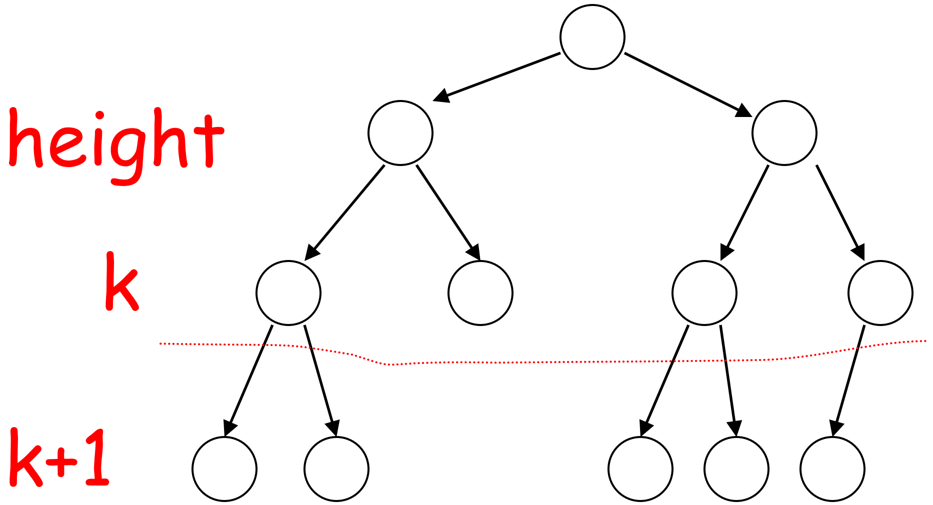
  we need to show that $L(k + 1) <= 2^{k+1}$

Induction Step



From Inductive hypothesis: $L(k) <= 2^k$

Induction Step



$L(k) <= 2^k$

$$L(k+1) <= 2 * L(k) <= 2 * 2^k = 2^{k+1}$$

(we add at most two nodes for every leaf of level k)

Remark

Recursion is another thing

Example of recursive function:

$$f(n) = f(n-1) + f(n-2)$$

$$f(0) = 1, \quad f(1) = 1$$

Proof by Contradiction

We want to prove that a statement P is true

- we assume that P is false
- then we arrive at an incorrect conclusion
- therefore, statement P must be true

Example

Theorem: $\sqrt{2}$ is not rational

Proof:

Assume by contradiction that it is rational

$\sqrt{2}$ = n/m

n and m have no common factors

We will show that this is impossible

$\sqrt{2} = n/m \Rightarrow 2\,m^2 = n^2$

Therefore, n^2 is even $\Rightarrow$ n is even

$n = 2\,k$

$2\,m^2 = 4k^2 \Rightarrow m^2 = 2k^2 \Rightarrow$ m is even

$m = 2\,p$

Thus, m and n have common factor 2

Contradiction!

Proof by Construction

We want to prove that a statement about something with a property is true

- constructing a concrete example with a property to show that something having that property exists.
- constructive proof is in contrast to a non-constructive proof which does not provide a means of constructing an example.

Example 1

16 can be exactly divided.

Proof

- A concrete example is 16/2. Therefore, the statement is true.

End

Example 2

There exist two irrational numbers which make a^b rational.

Proof

Let a=b= $\sqrt{2}$

case 1: $\sqrt{2}^{\sqrt{2}}$ is rational. done, otherwise

case 2: let a= $\sqrt{2}^{\sqrt{2}}$, then $(\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} = \sqrt{2}^2 = 2$, done.

End

Question

Is example 2 the constructive proof?

Why if yes? Why if no?

Example 3

Show that there is no "largest integer".

Proof

Let n be any integer.

Let $m = n+1$

m is an integer

$m > n$

Therefore m is an integer that is larger than n

Therefore, for any integer there exists an integer $m = n + 1$ that is larger than it.

End

Question

Is example 3 the constructive proof?

Why if yes? Why if no?

Example 4

Show that there is no "largest" prime number.

Proof

Let n be any prime number

Let m = n! + 1, then m>n

Case 1:

m = n! + 1 is a prime number, then we have constructed a prime number that is larger than the previous prime number.

Case 2:

m = n! + 1 is not a prime number, then it has at least one prime factor

Example 4 (Cont.)

Explanation:

If you divide m by any of the prime numbers that are smaller than or equal to n, you will always get a remainder of 1,

because each prime number less than or equal to n divides evenly into n!.

Therefore any prime factors of m must be greater than n.

End

Question

Is example 4 the constructive proof?

Why if yes? Why if no?

Languages

A language is a set of strings

String: A sequence of letters

Examples: "cat", "dog", "house", …

Defined over an alphabet:

$$\Sigma = \{a, b, c, \ldots, z\}$$

Alphabets and Strings

We will use small alphabets: $\Sigma = \{a, b\}$

Strings

a

ab $u = ab$

$abba$ $v = bbbaaa$

$baba$ $w = abba$

$aaabbbaabab$

String Operations

$$w = a_1 a_2 \cdots a_n$$

$$v = b_1 b_2 \cdots b_m$$

$abba$

$bbbaaa$

Concatenation

$$wv = a_1 a_2 \cdots a_n b_1 b_2 \cdots b_m$$

$abbabbbaaa$

$$w = a_1 a_2 \cdots a_n \qquad ababaaabbb$$

Reverse

$$w^R = a_n \cdots a_2 a_1 \qquad bbbaaababa$$

String Length

$$w = a_1 a_2 \cdots a_n$$

Length: $|w| = n$

Examples:

$$|abba| = 4$$

$$|aa| = 2$$

$$|a| = 1$$

Length of Concatenation

$$|uv| = |u| + |v|$$

Example: $u = aab, \quad |u| = 3$

$$v = abaab, \quad |v| = 5$$

$$|uv| = |aababaab| = 8$$

$$|uv| = |u| + |v| = 3 + 5 = 8$$

Empty String

A string with no letters: λ

Observations: $|\lambda| = 0$

$$\lambda w = w\lambda = w$$

$$\lambda abba = abba\lambda = abba$$

Substring

Substring of string:

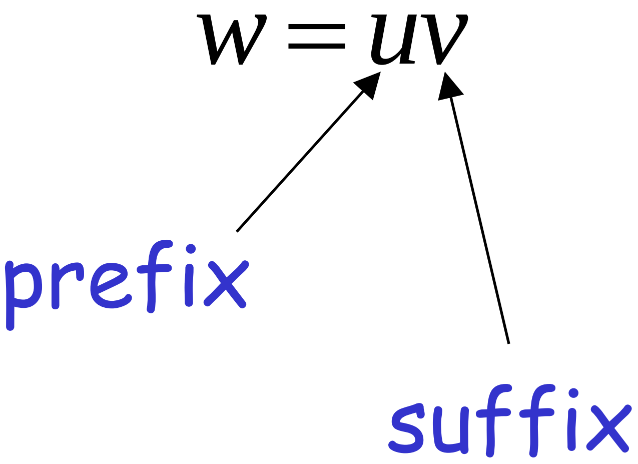
a subsequence of consecutive characters

String	Substring
$\underline{ab}bab$	ab
$\underline{abba}b$	$abba$
$ab\underline{b}ab$	b
$a\underline{bbab}$	$bbab$

Prefix and Suffix

$abbab$

Prefixes	Suffixes
λ	$abbab$
a	$bbab$
ab	bab
abb	ab
$abba$	b
$abbab$	λ



$w = uv$

Another Operation

$$w^n = \underbrace{ww\cdots w}_{n}$$

Example: $(abba)^2 = abbaabba$

Definition: $w^0 = \lambda$

$$(abba)^0 = \lambda$$

The * Operation

$\Sigma *$: the set of all possible strings from alphabet Σ

$$\Sigma = \{a, b\}$$

$$\Sigma * = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}$$

The + Operation

Σ^+ : the set of all possible strings from alphabet Σ except λ

$$\Sigma = \{a, b\}$$

$$\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}$$

$$\Sigma^+ = \Sigma^* - \lambda$$

$$\Sigma^+ = \{a, b, aa, ab, ba, bb, aaa, aab, \ldots\}$$

Languages

A language is any subset of Σ^*

Example: $\Sigma = \{a, b\}$

$$\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, \ldots\}$$

Languages: $\{\lambda\}$

$$\{a, aa, aab\}$$

$$\{\lambda, abba, baba, aa, ab, aaaaaa\}$$

Note that:

Sets	$\emptyset = \{\ \} \neq \{\lambda\}$
Set size	$\lvert\{\ \}\rvert = \lvert\emptyset\rvert = 0$
Set size	$\lvert\{\lambda\}\rvert = 1$
String length	$\lvert\lambda\rvert = 0$

Another Example

An infinite language $L = \{a^n b^n : n \geq 0\}$

$$\left.\begin{array}{l} \lambda \\ ab \\ aabb \\ aaaaabbbbb \end{array}\right\} \in L \qquad\qquad abb \notin L$$

Operations on Languages

The usual set operations

$$\{a,ab,aaaa\}\cup\{bb,ab\}=\{a,ab,bb,aaaa\}$$

$$\{a,ab,aaaa\}\cap\{bb,ab\}=\{ab\}$$

$$\{a,ab,aaaa\}-\{bb,ab\}=\{a,aaaa\}$$

Complement: $\overline{L}=\Sigma^*-L$

$$\overline{\{a,ba\}}=\{\lambda,b,aa,ab,bb,aaa,\ldots\}$$

Reverse

Definition: $L^R = \{w^R : w \in L\}$

Examples: $\{ab, aab, baba\}^R = \{ba, baa, abab\}$

$$L = \{a^n b^n : n \geq 0\}$$

$$L^R = \{b^n a^n : n \geq 0\}$$

Concatenation

Definition: $L_1L_2 = \{xy : x \in L_1, y \in L_2\}$

Example: $\{a, ab, ba\}\{b, aa\}$

$$= \{ab, aaa, abb, abaa, bab, baaa\}$$

Another Operation

Definition: $L^n = \underbrace{LL\cdots L}_{n}$

$$\{a,b\}^3 = \{a,b\}\{a,b\}\{a,b\} =$$

$$\{aaa, aab, aba, abb, baa, bab, bba, bbb\}$$

Special case: $L^0 = \{\lambda\}$

$$\{a, bba, aaa\}^0 = \{\lambda\}$$

More Examples

$$L = \{a^n b^n : n \geq 0\}$$

$$L^2 = \{a^n b^n a^m b^m : n, m \geq 0\}$$

$$aabbaaabbb \in L^2$$

Star-Closure (Kleene *)

Definition: $L^* = L^0 \bigcup L^1 \bigcup L^2 \cdots$

Example:

$$\{a, bb\}^* = \begin{Bmatrix} \lambda, \\ a, bb, \\ aa, abb, bba, bbbb, \\ aaa, aabb, abba, abbbb, \ldots \end{Bmatrix}$$

Positive Closure

Definition:

$$L^+ = L^1 \cup L^2 \cup \cdots$$

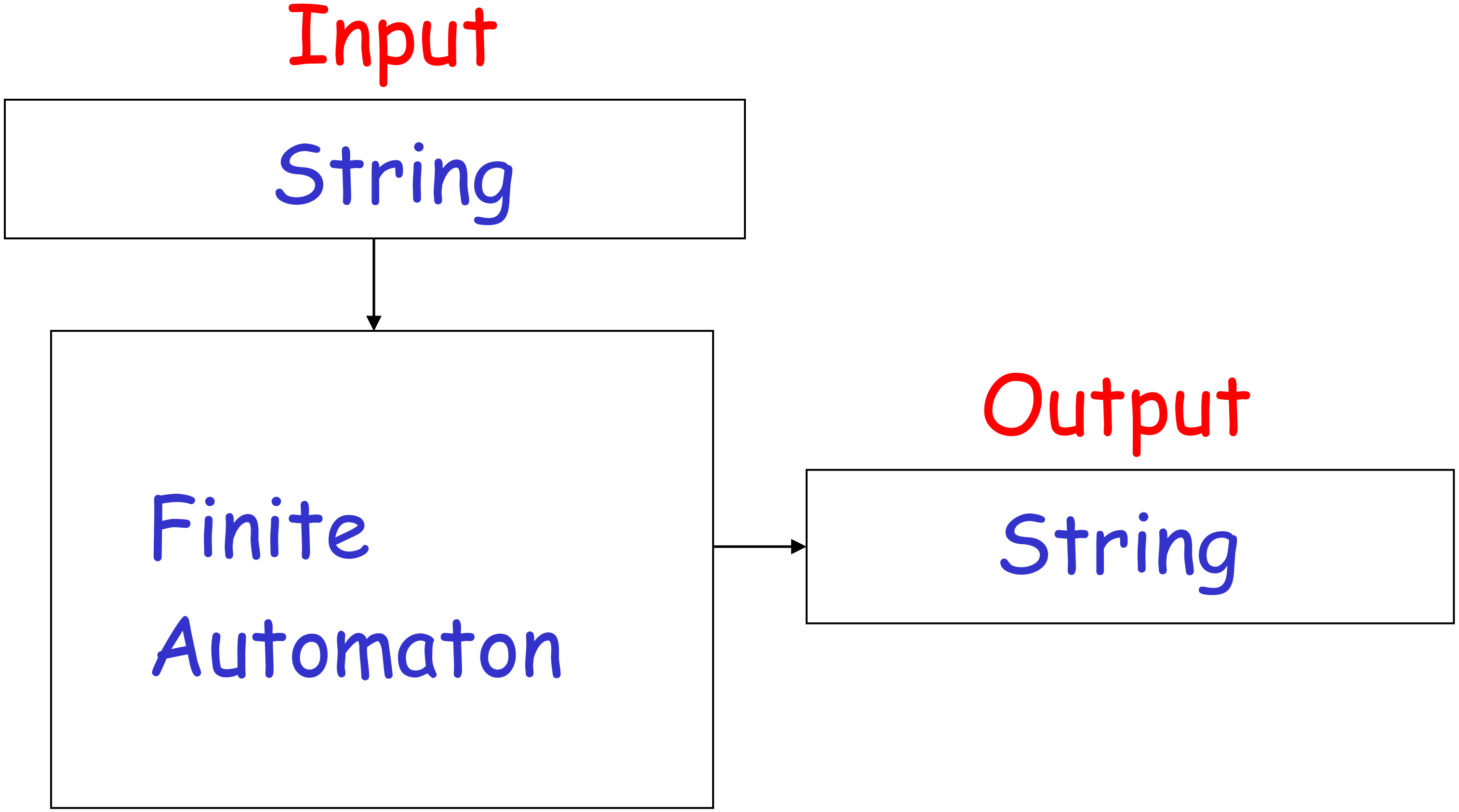
$$= L^* - \{\lambda\}$$

$$\{a, bb\}^+ = \left\{ \begin{array}{l} a, bb, \\ aa, abb, bba, bbbb, \\ aaa, aabb, abba, abbbb, \ldots \end{array} \right\}$$

The End

Finite Automata

Finite Automaton



Input

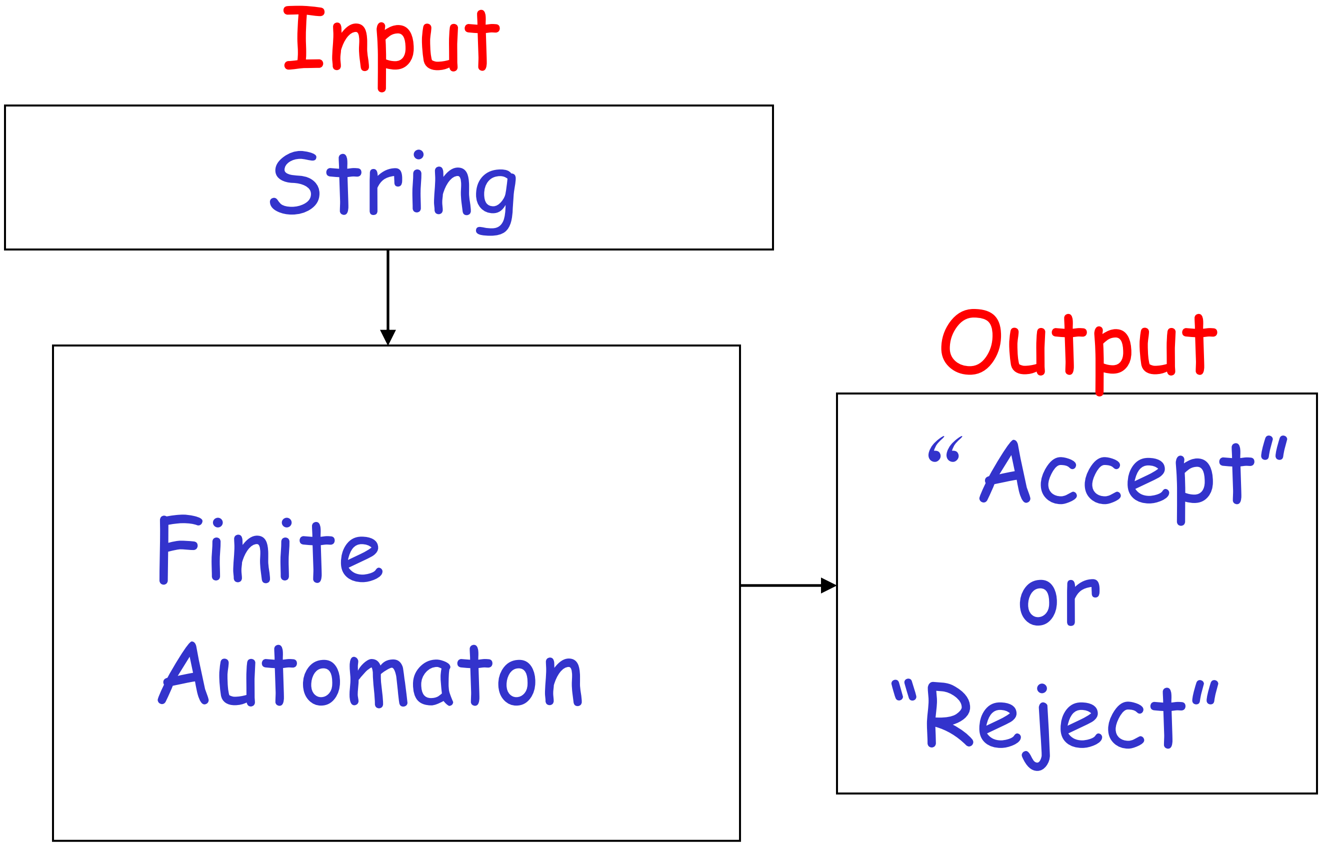
String

Finite Automaton

Output

String

Finite Accepter



Input

String

Finite Automaton

Output

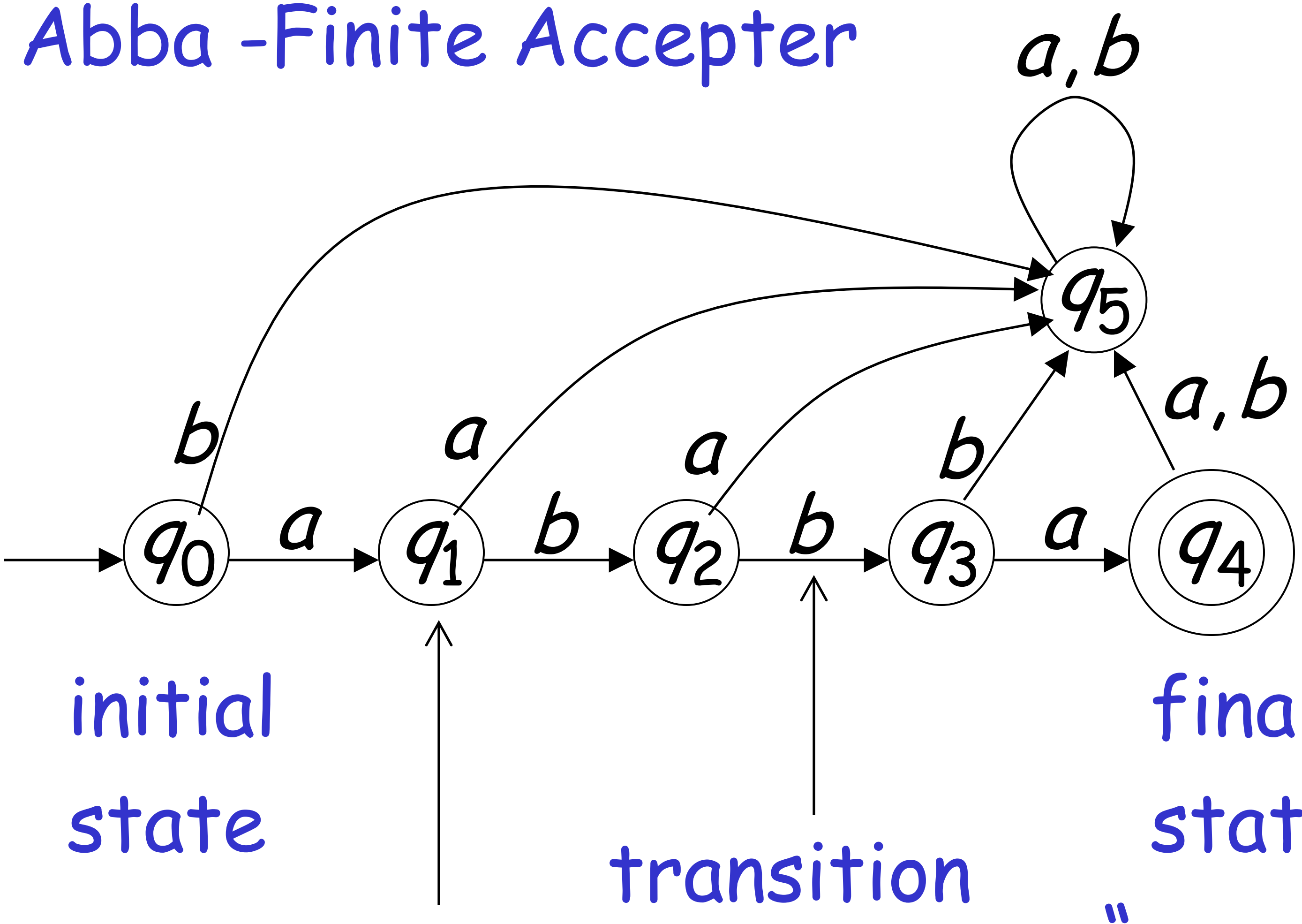
"Accept"

or

"Reject"

Transition Graph

Abba -Finite Accepter



initial
state

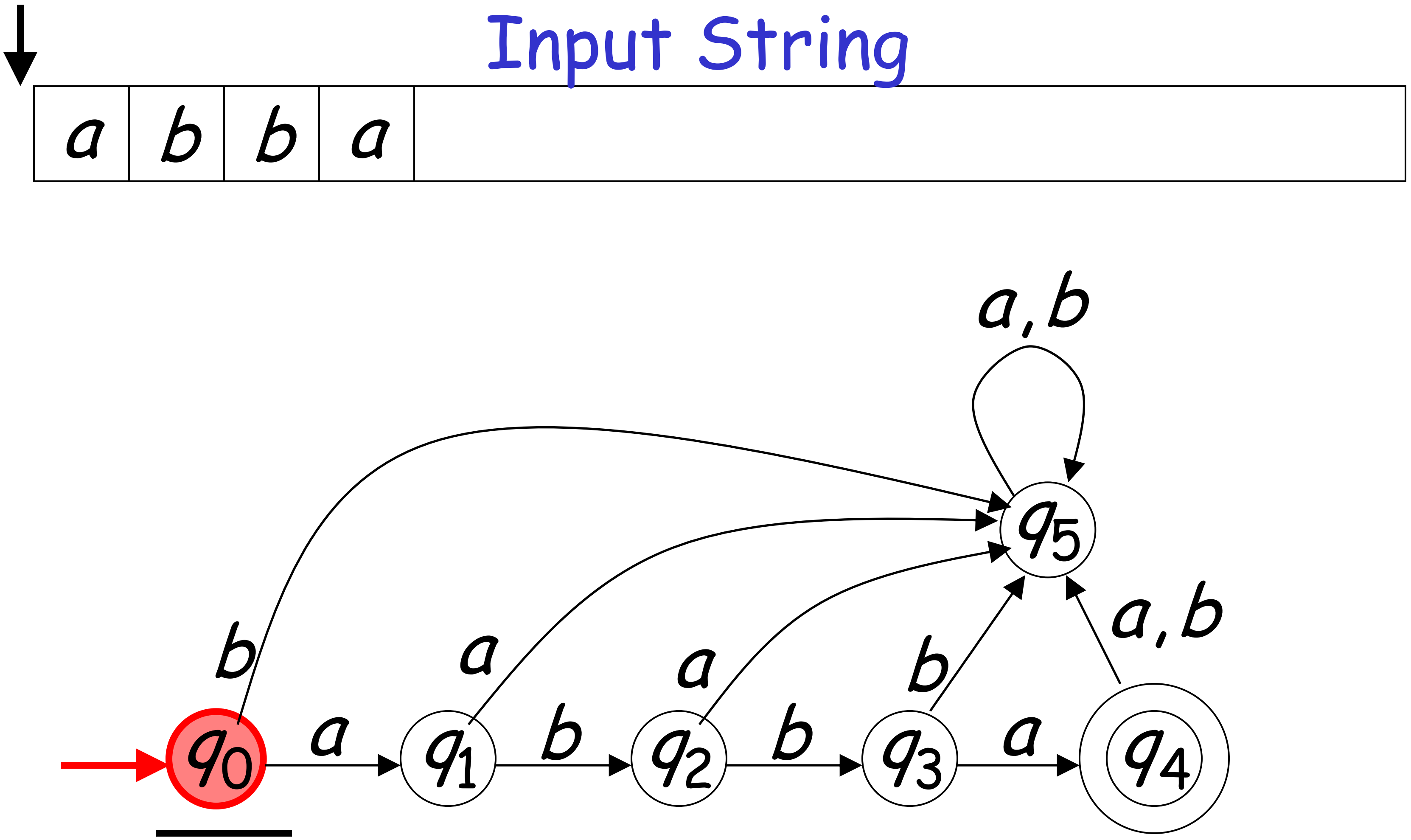
state

transition

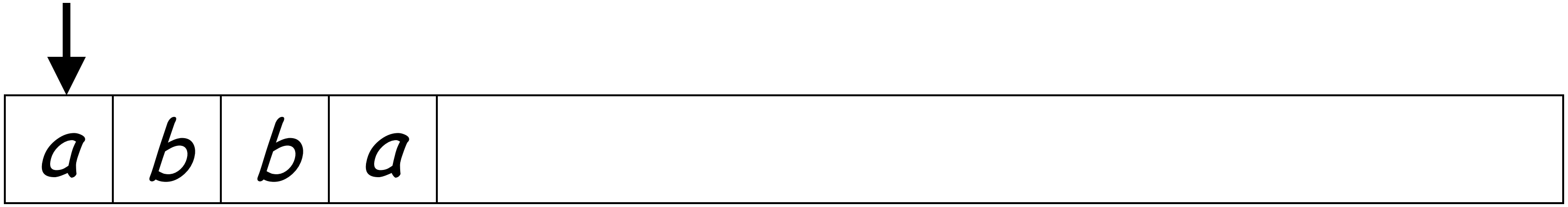
final
state
"accept"

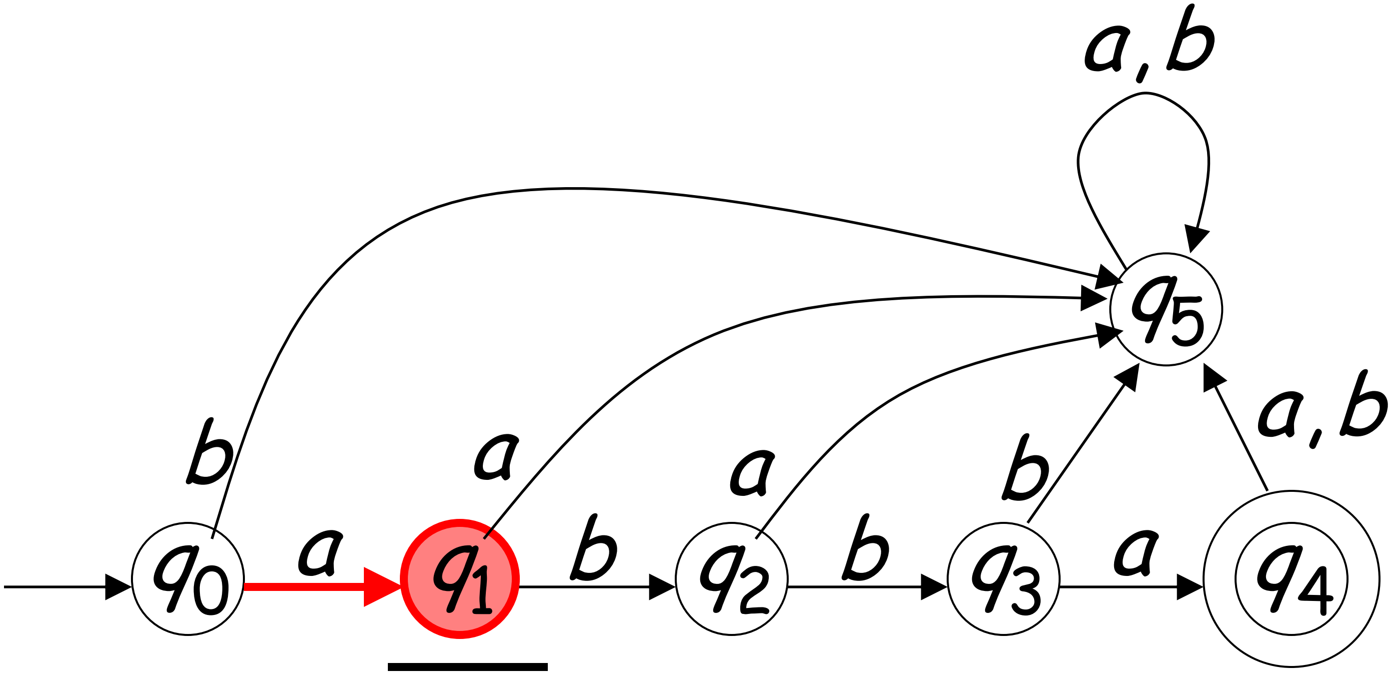
Initial Configuration

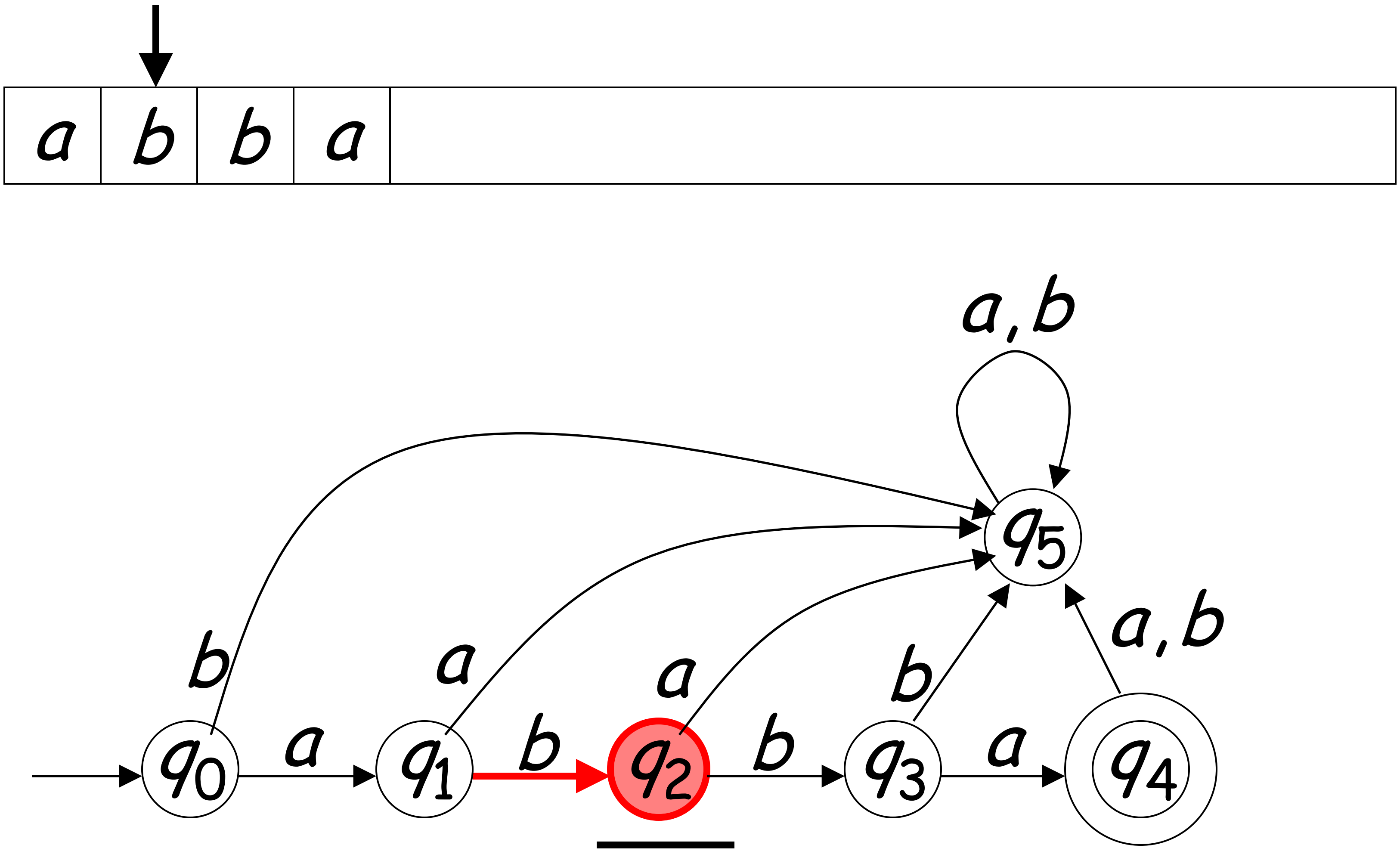
Input String

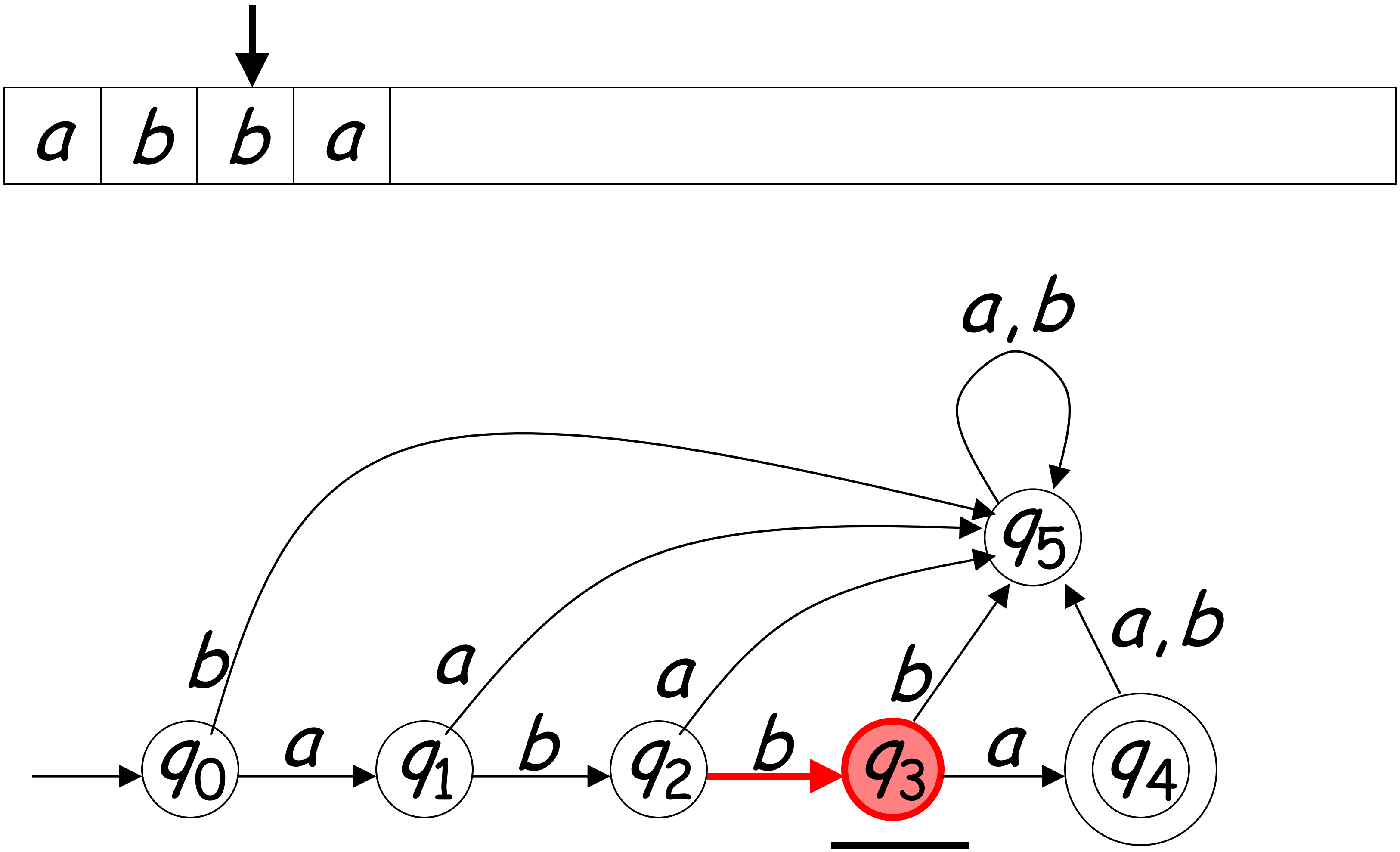


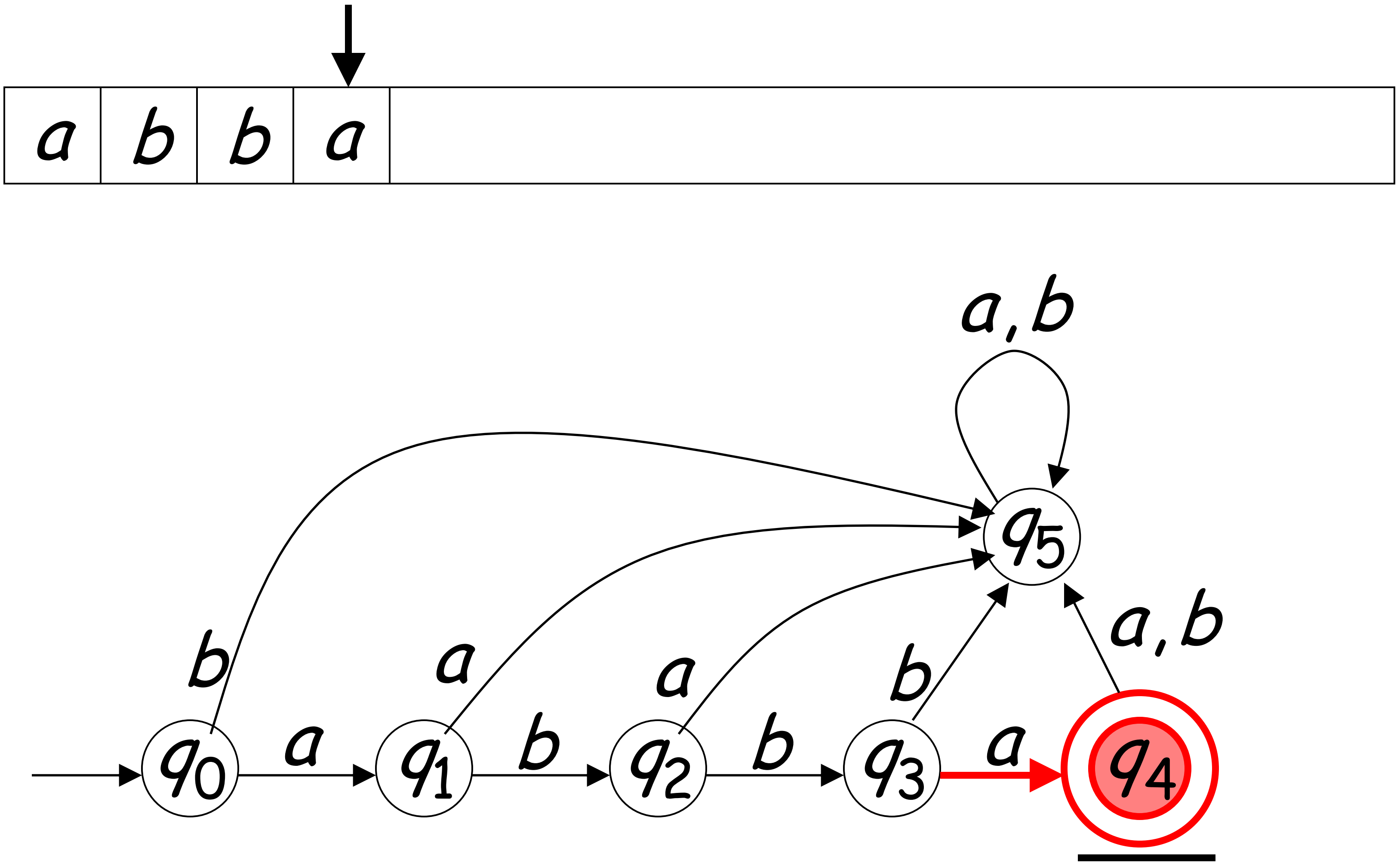
Reading the Input



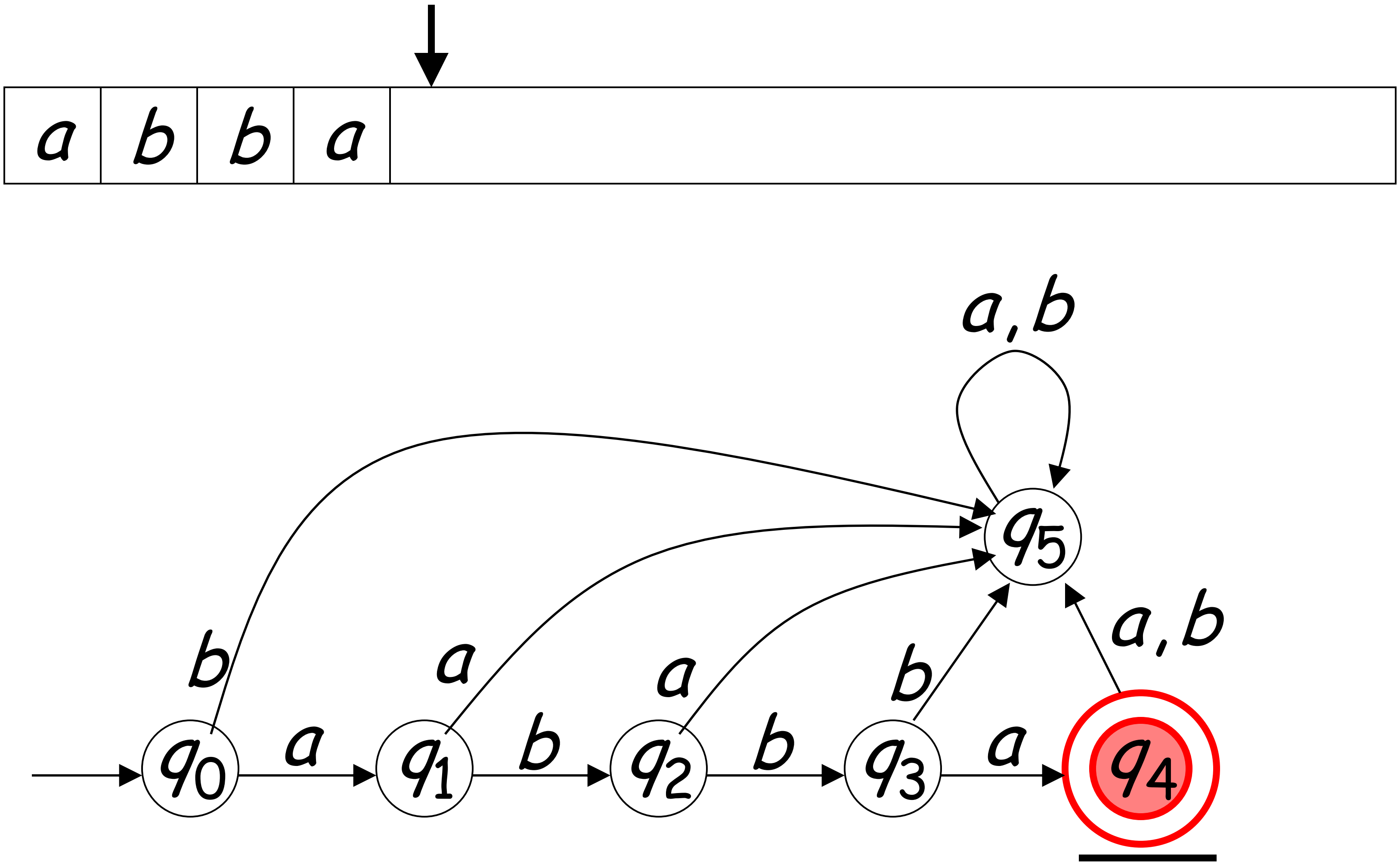



a
b
b
a
a,b
q5
b
a
a
b
a,b
q0
a
q1
b
q2
b
q3
a
q4


a
b
b
a
a,b
q5
b
a
a
b
a,b
q0
a
q1
b
q2
b
q3
a
q4

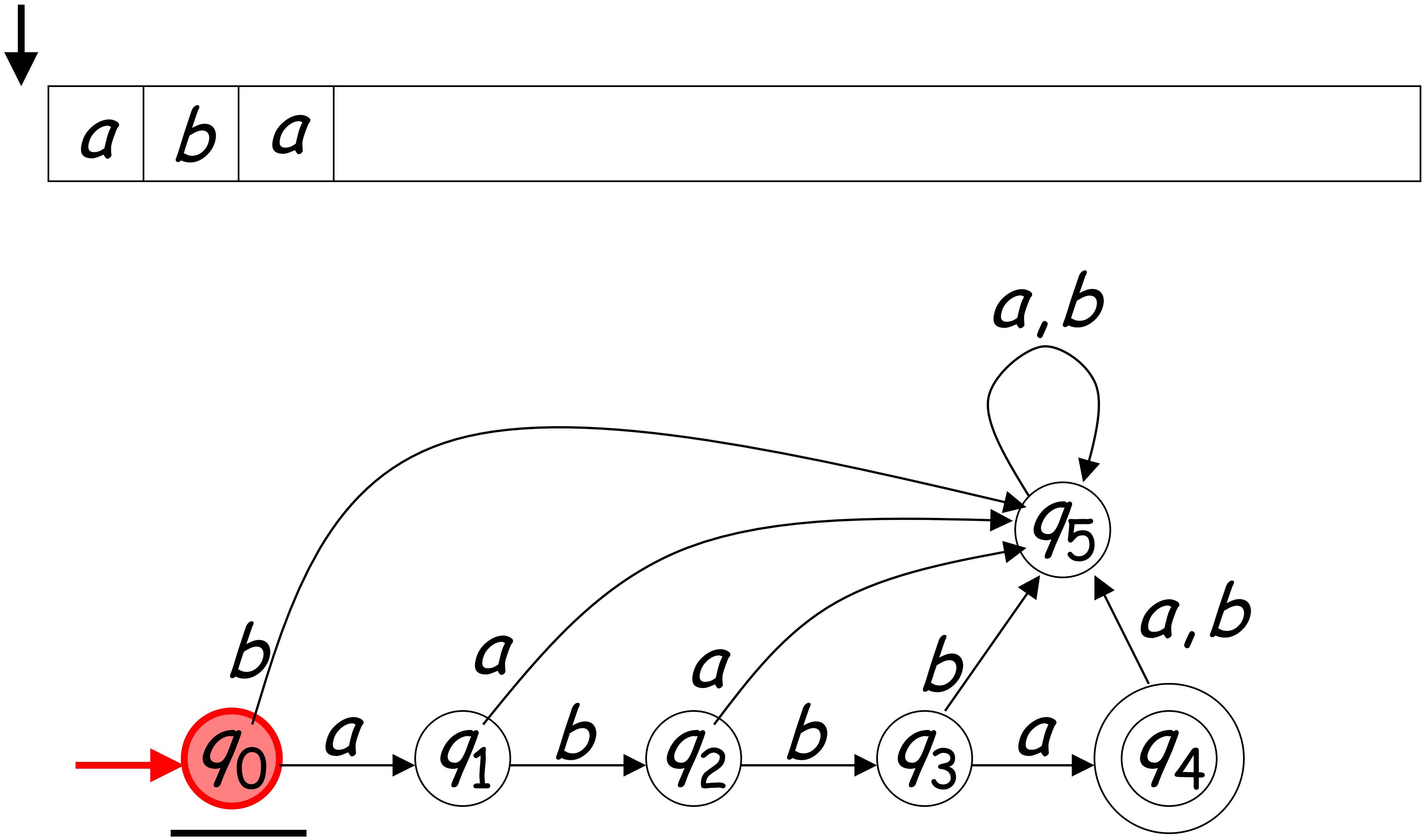

a
b
b
a
a,b
b
a
a
b
a,b
q0
a
q1
b
q2
b
q3
a
q4
q5

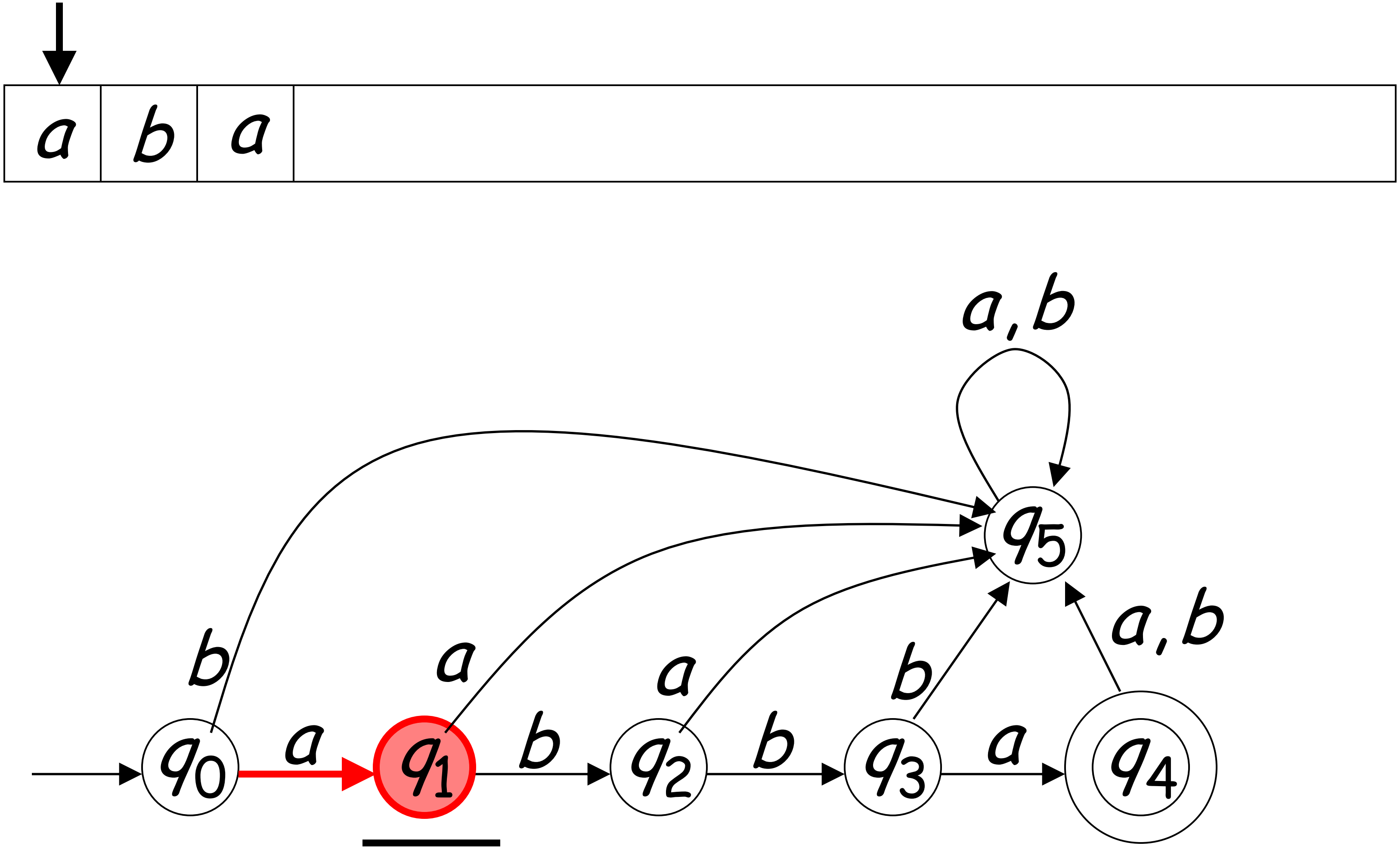
Input finished

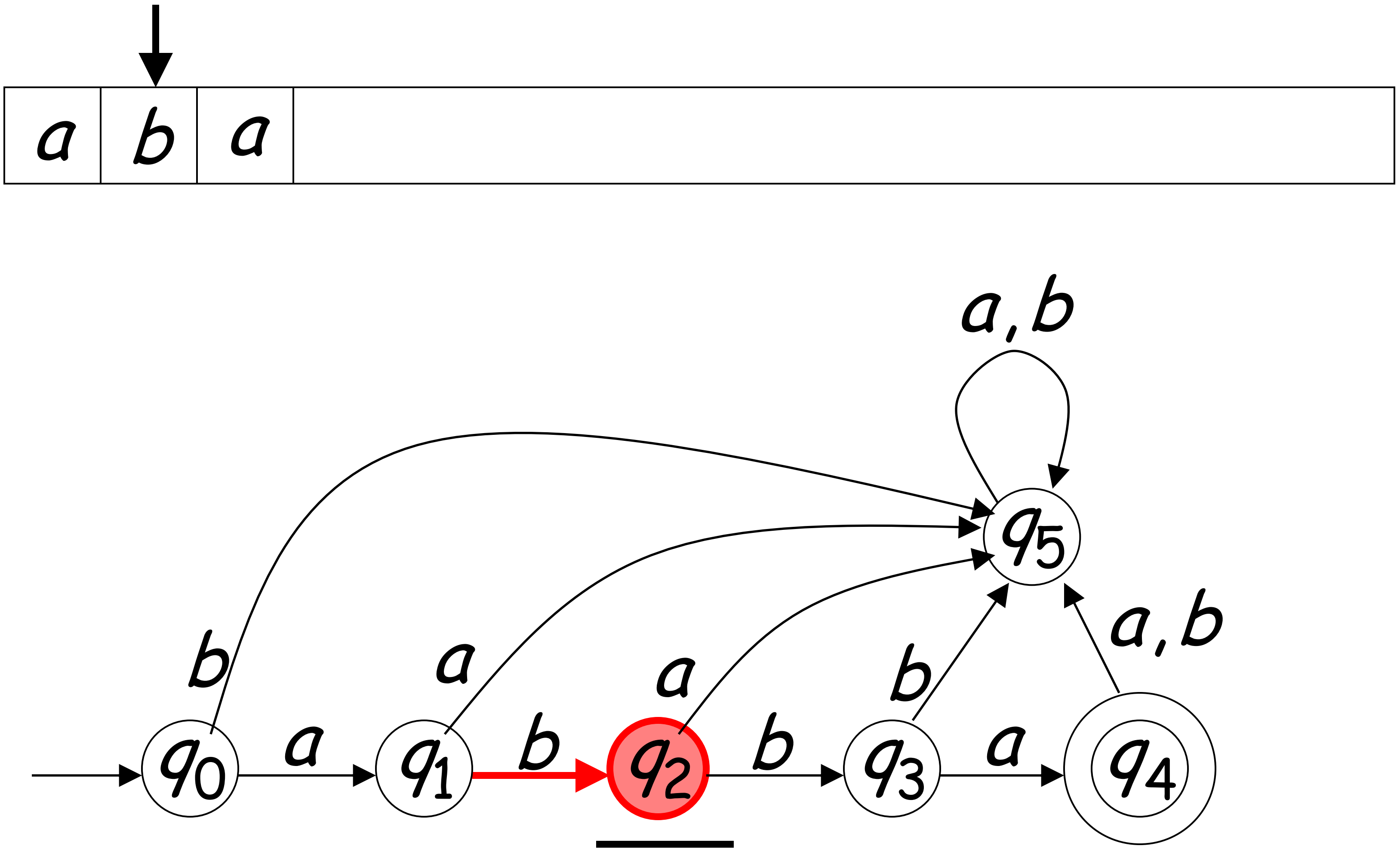


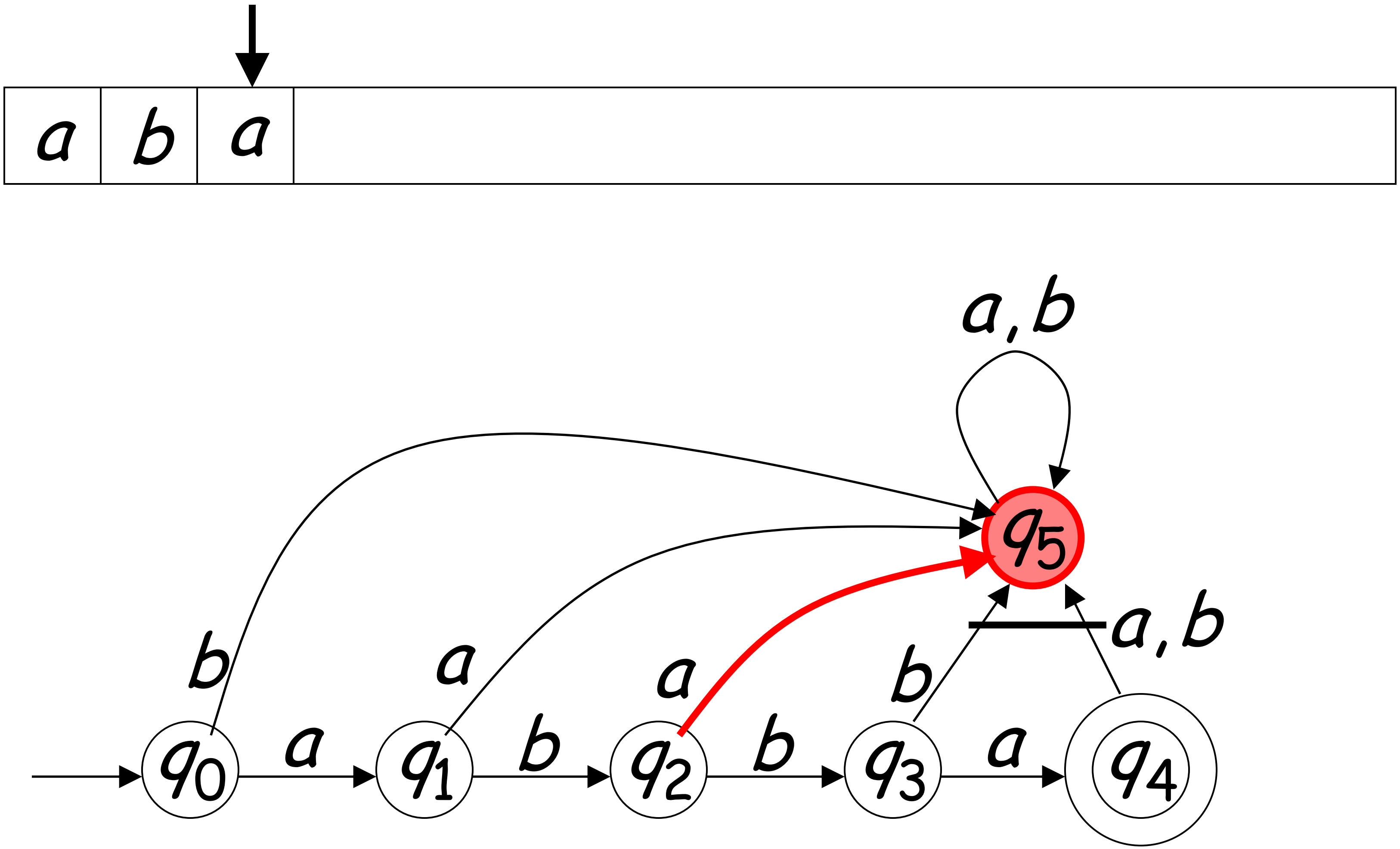
Output: "accept"

Rejection

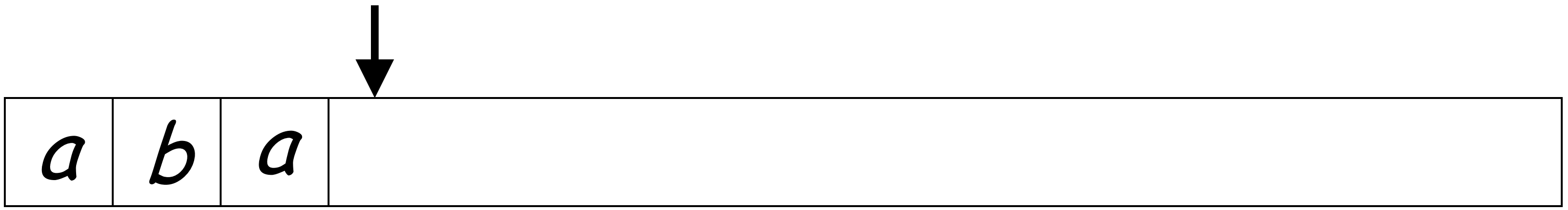


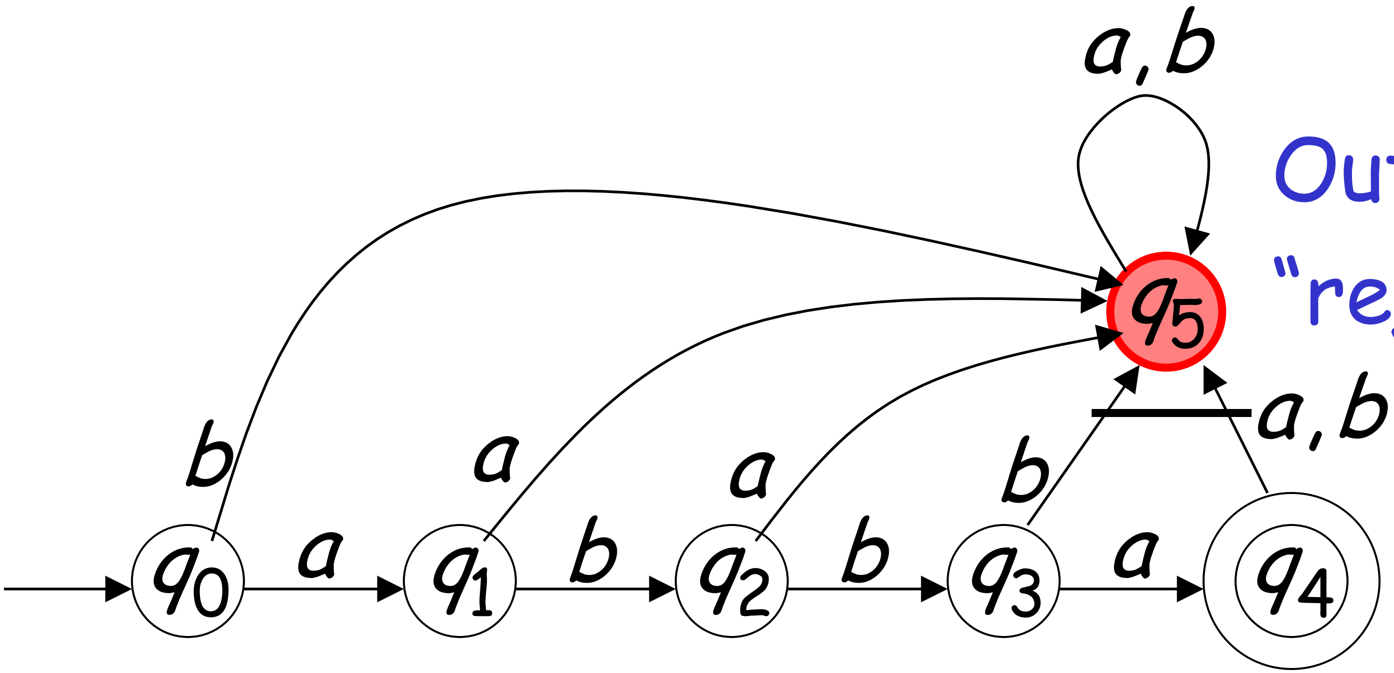

a
b
a
a,b
b
a
a
b
a
b
a,b
q0
a
q1
b
q2
b
q3
a
q4
q5


a
b
a
a,b
q5
b
a
a
b
a,b
q0
a
q1
b
q2
b
q3
a
q4


a
b
a
a,b
b
a
a
b
a,b
q0
a
q1
b
q2
b
q3
a
q4
q5

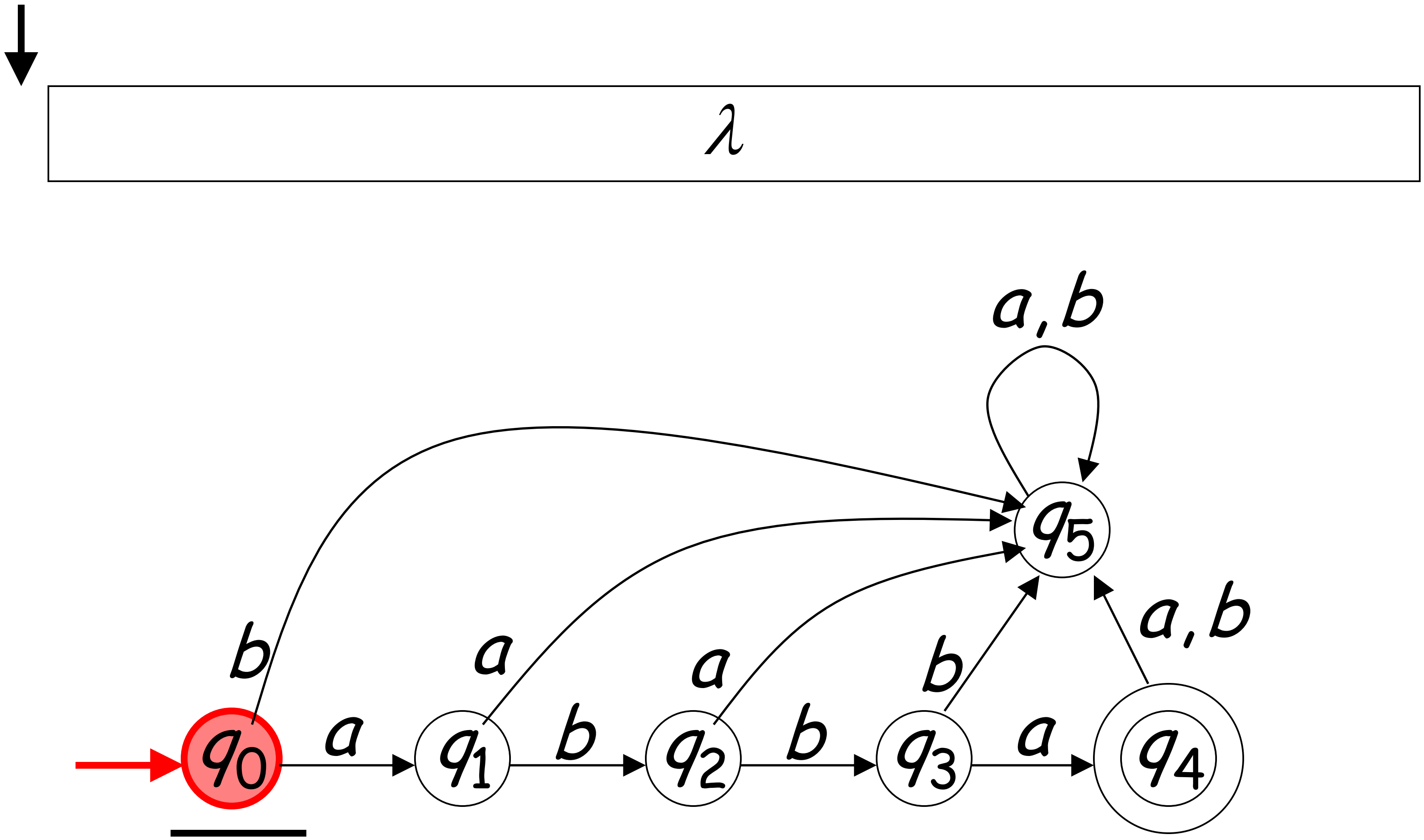
Input finished

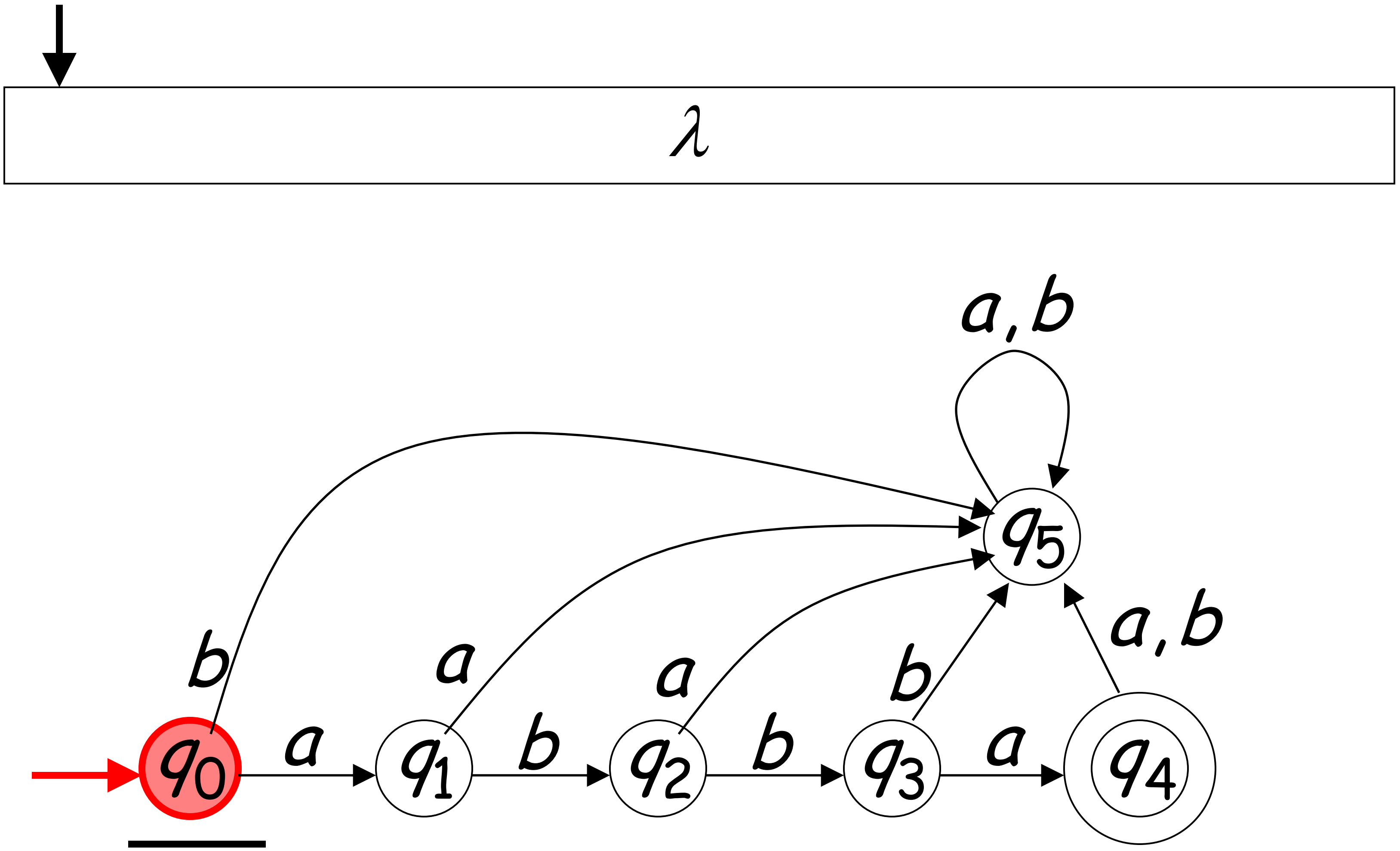




Output:
"reject"

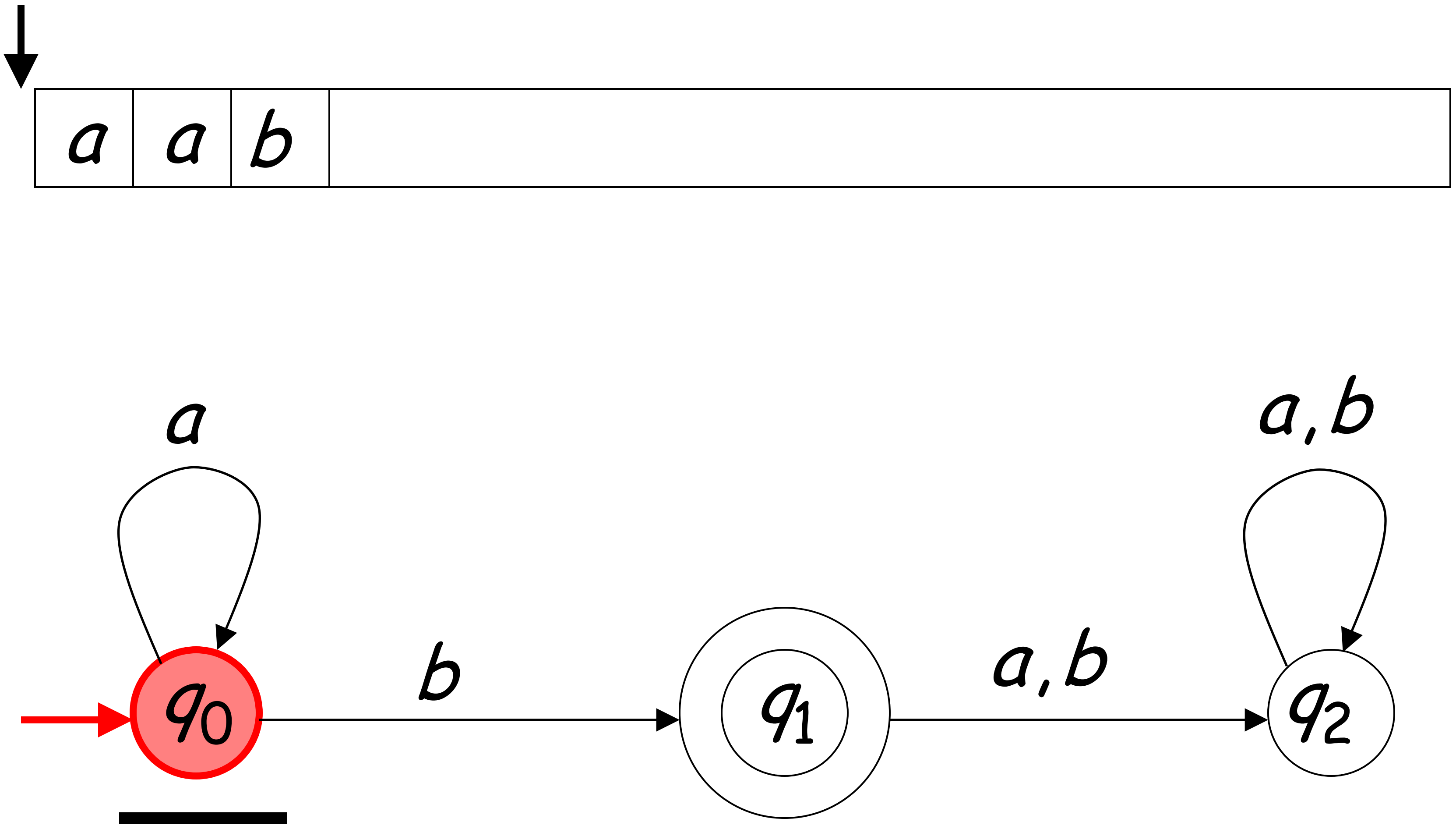
Another Rejection

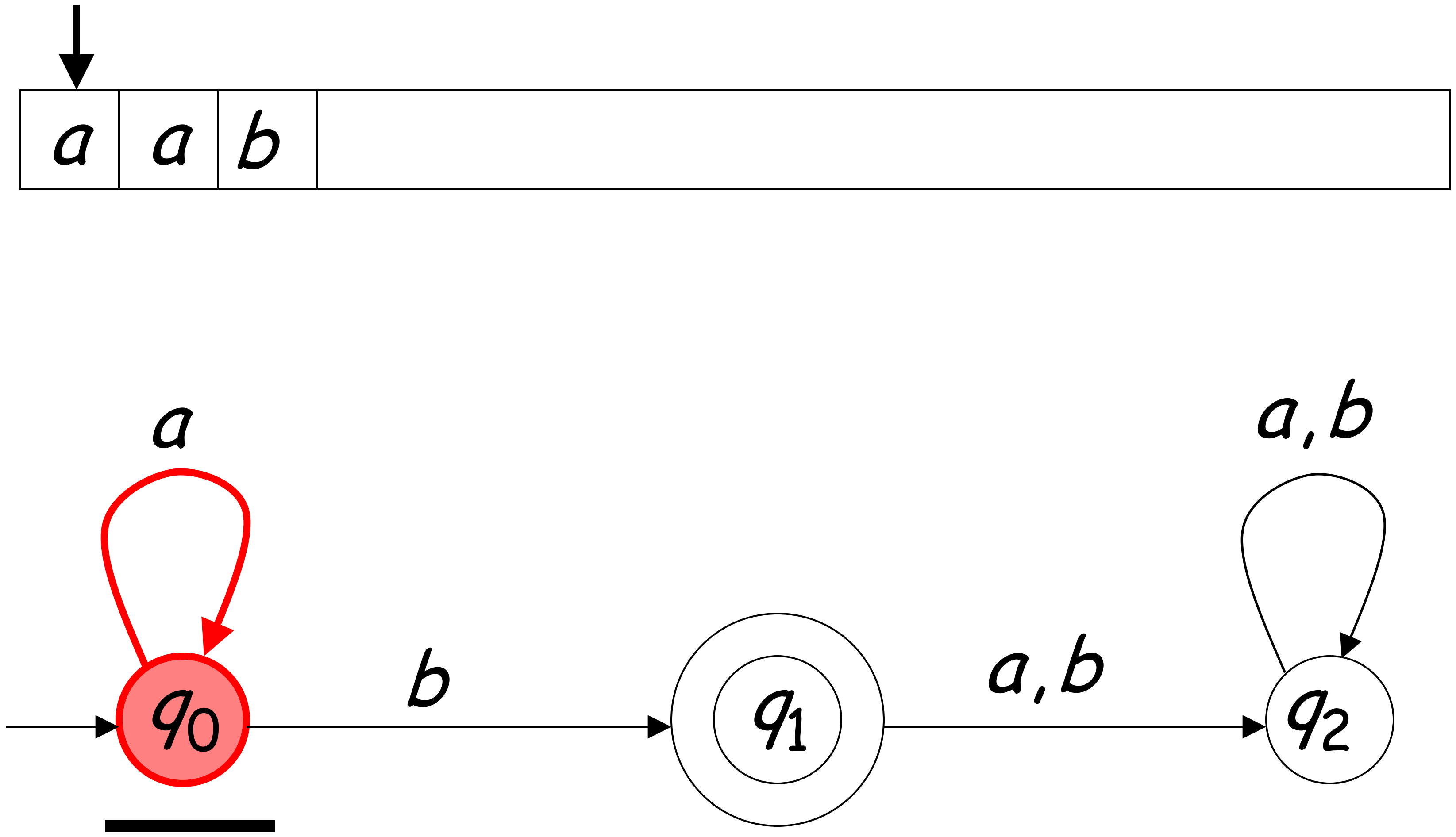


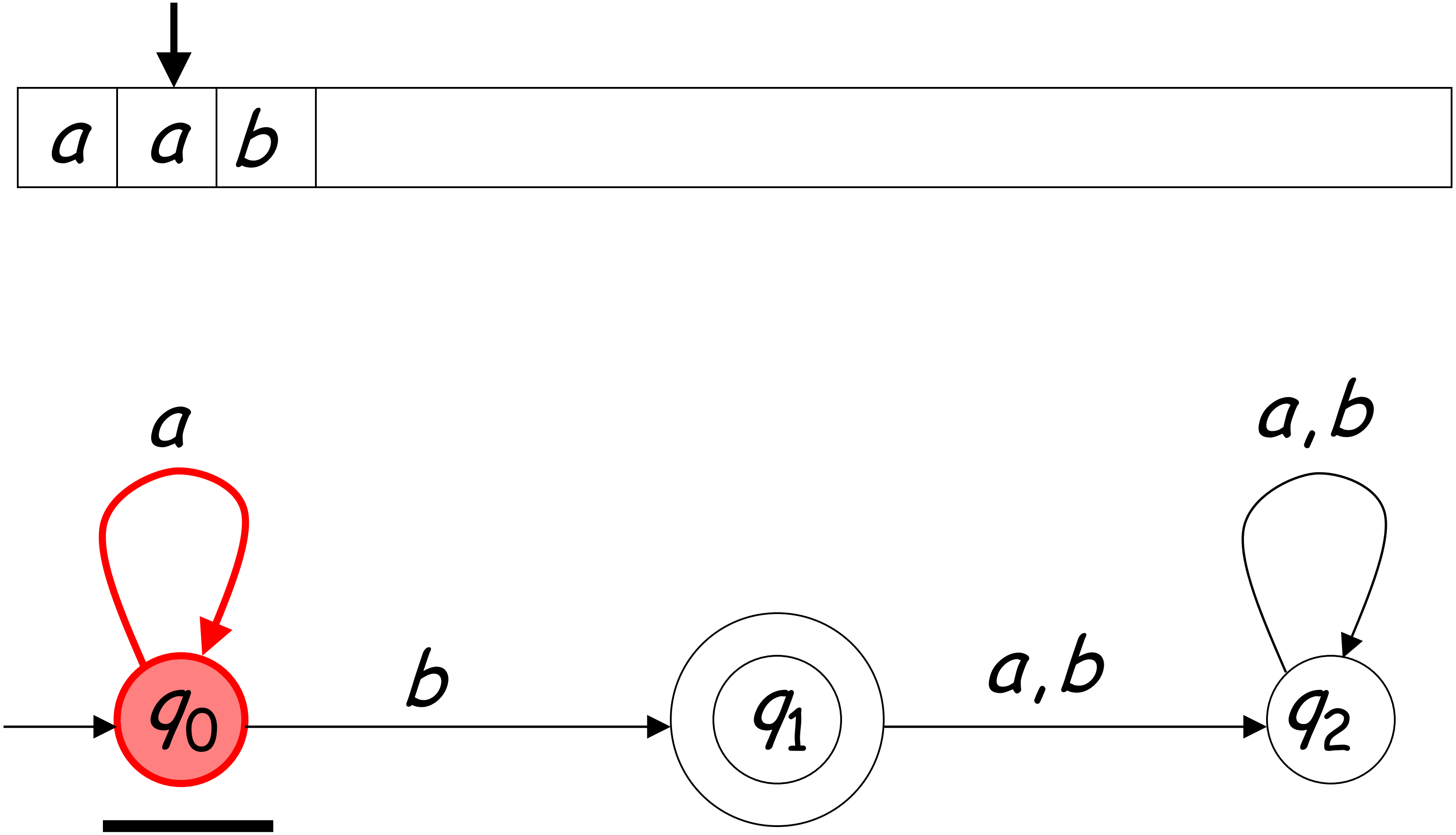


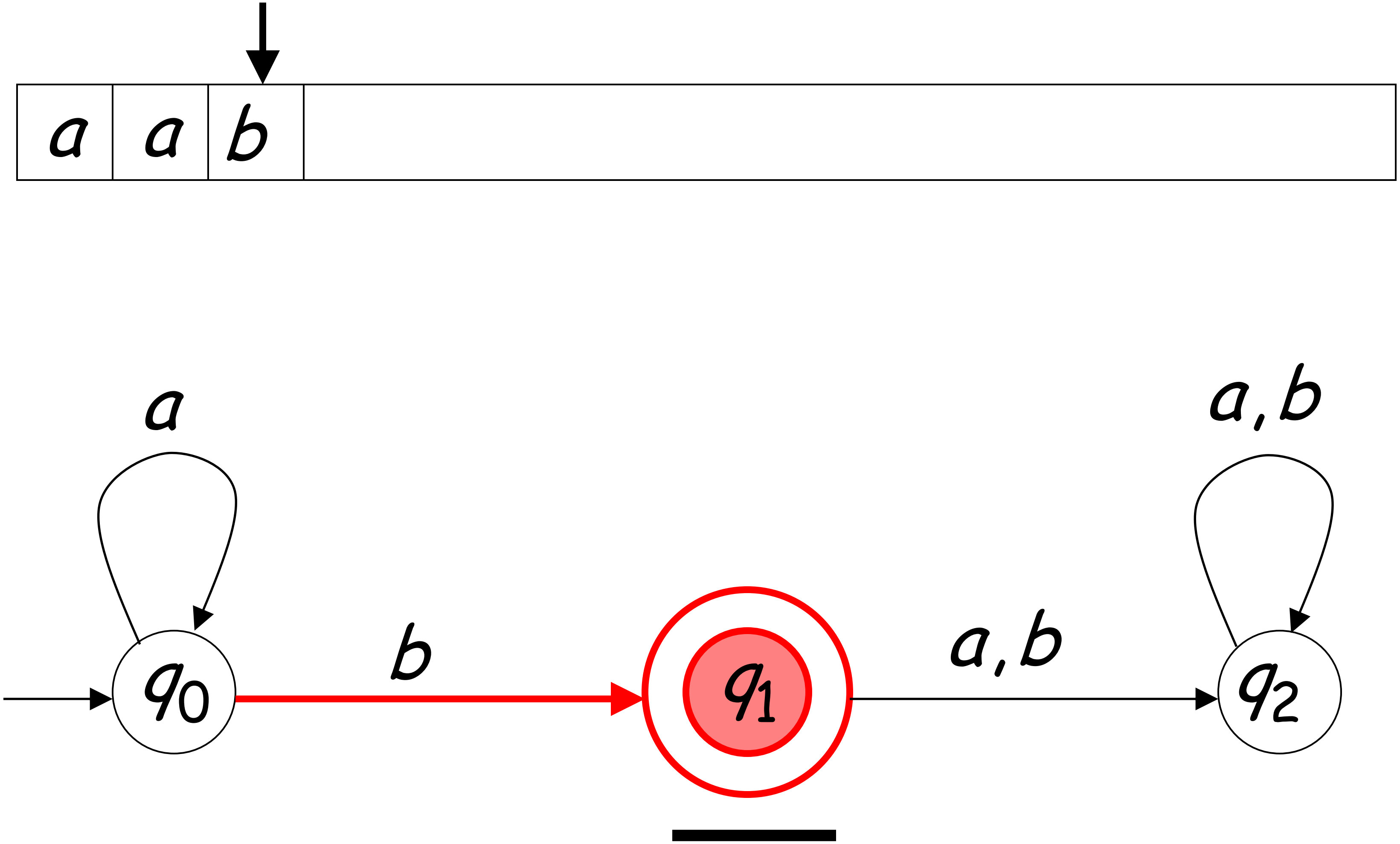
Output:
"reject"

Another Example

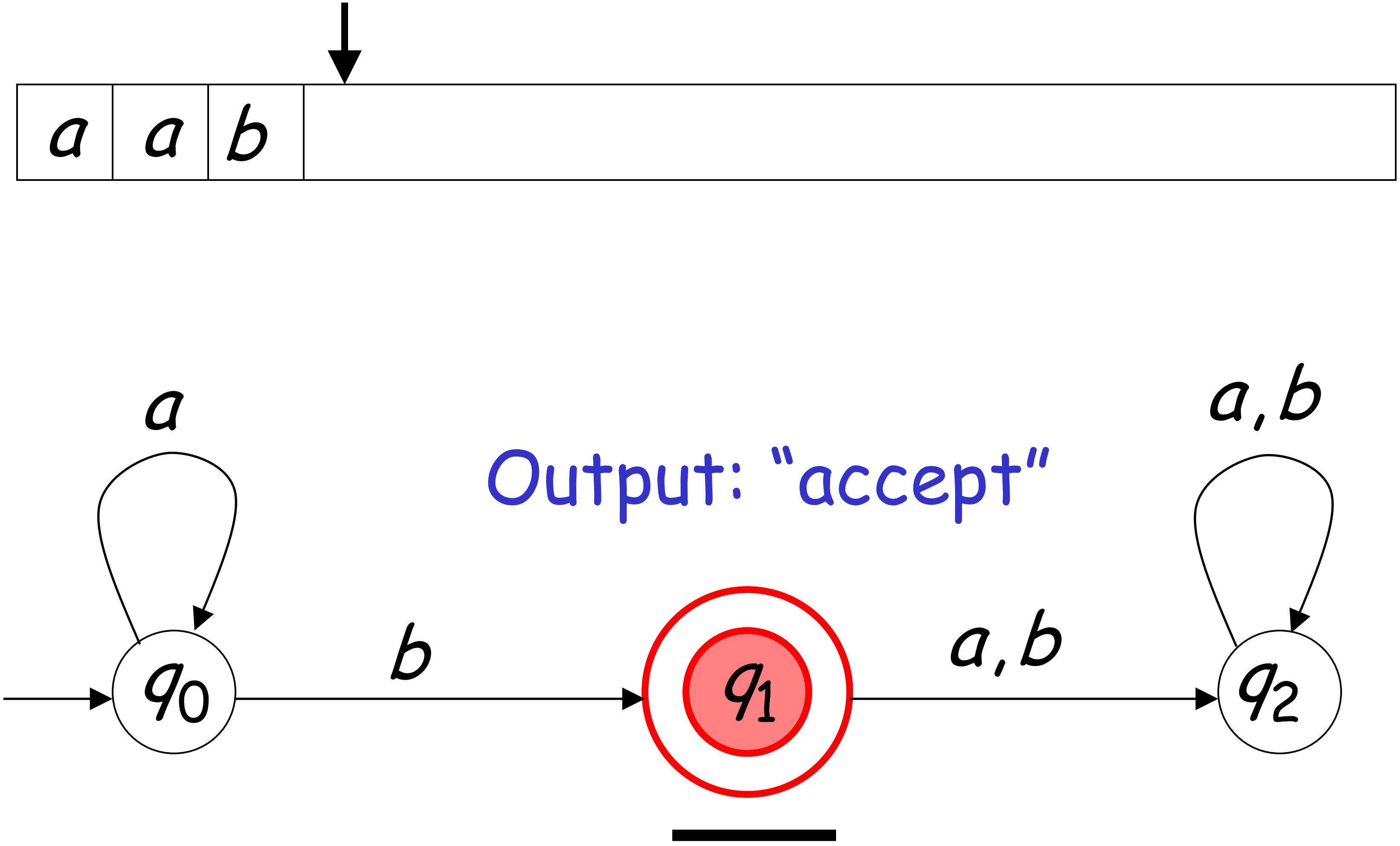



a
a
b
a
b
a,b
a,b
q_0
q_1
q_2


a
a
b
a
a, b
b
a, b
q_0
q_1
q_2

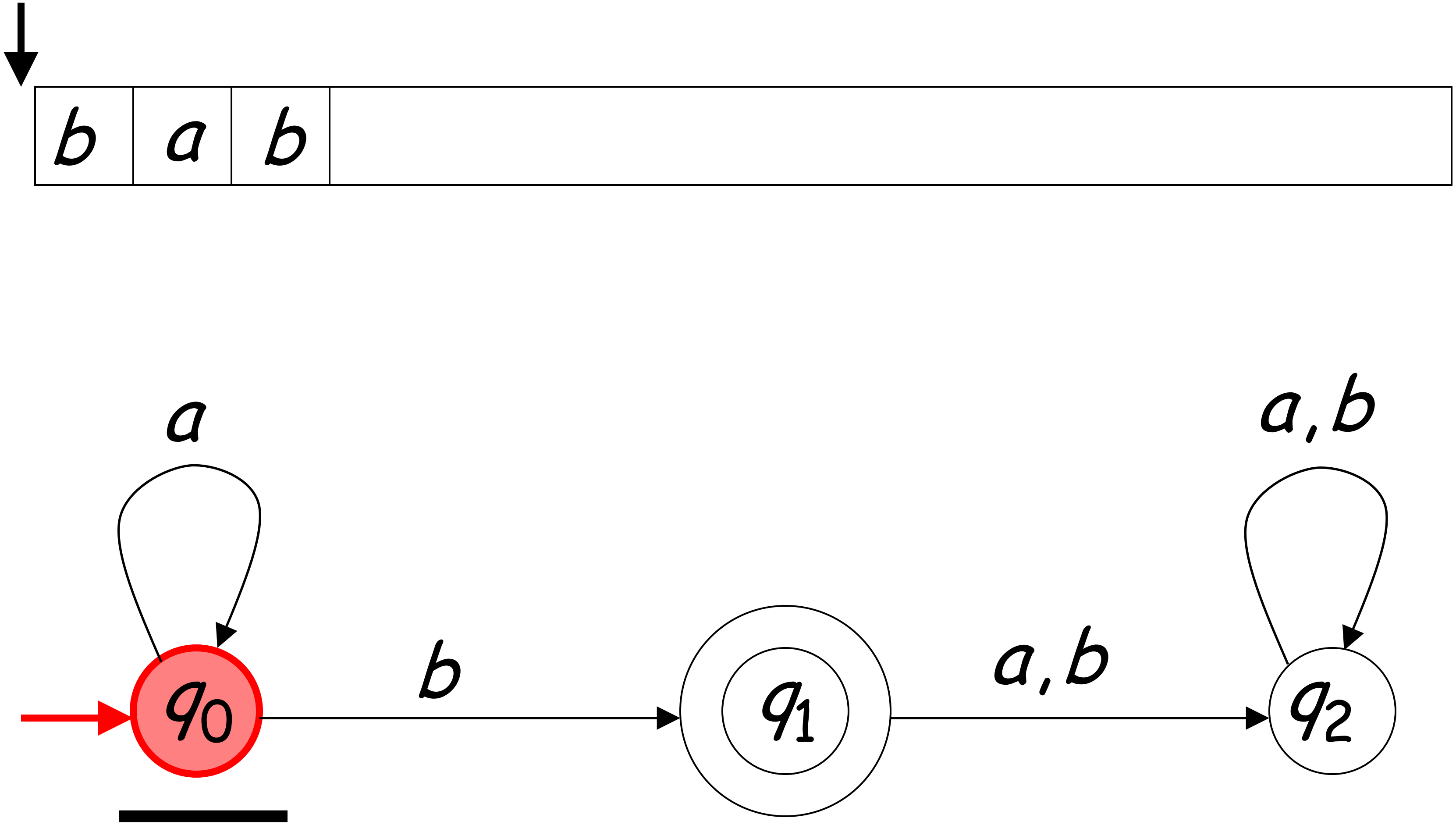

a
a
b
a
a,b
b
a,b
q_0
q_1
q_2

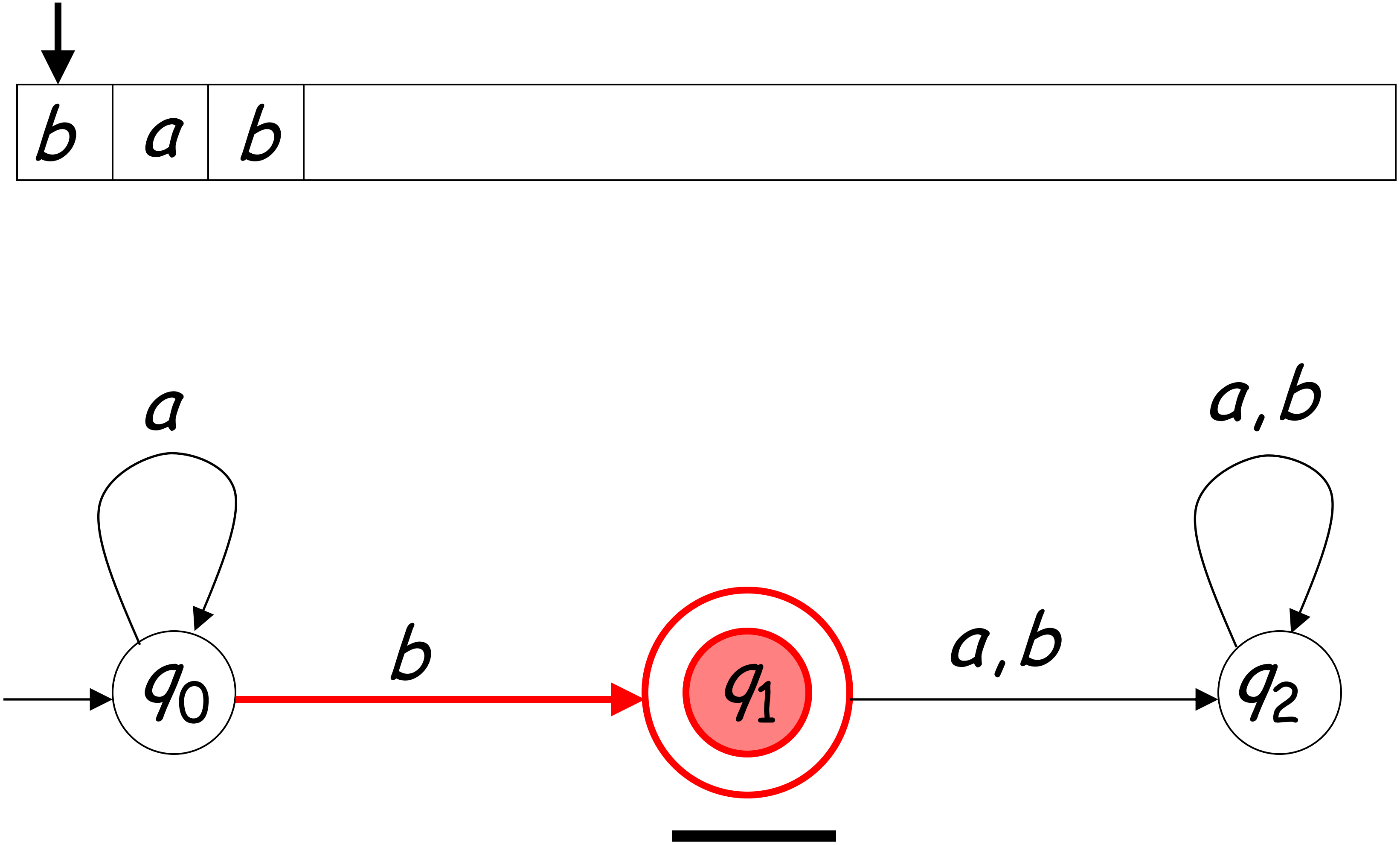
Input finished

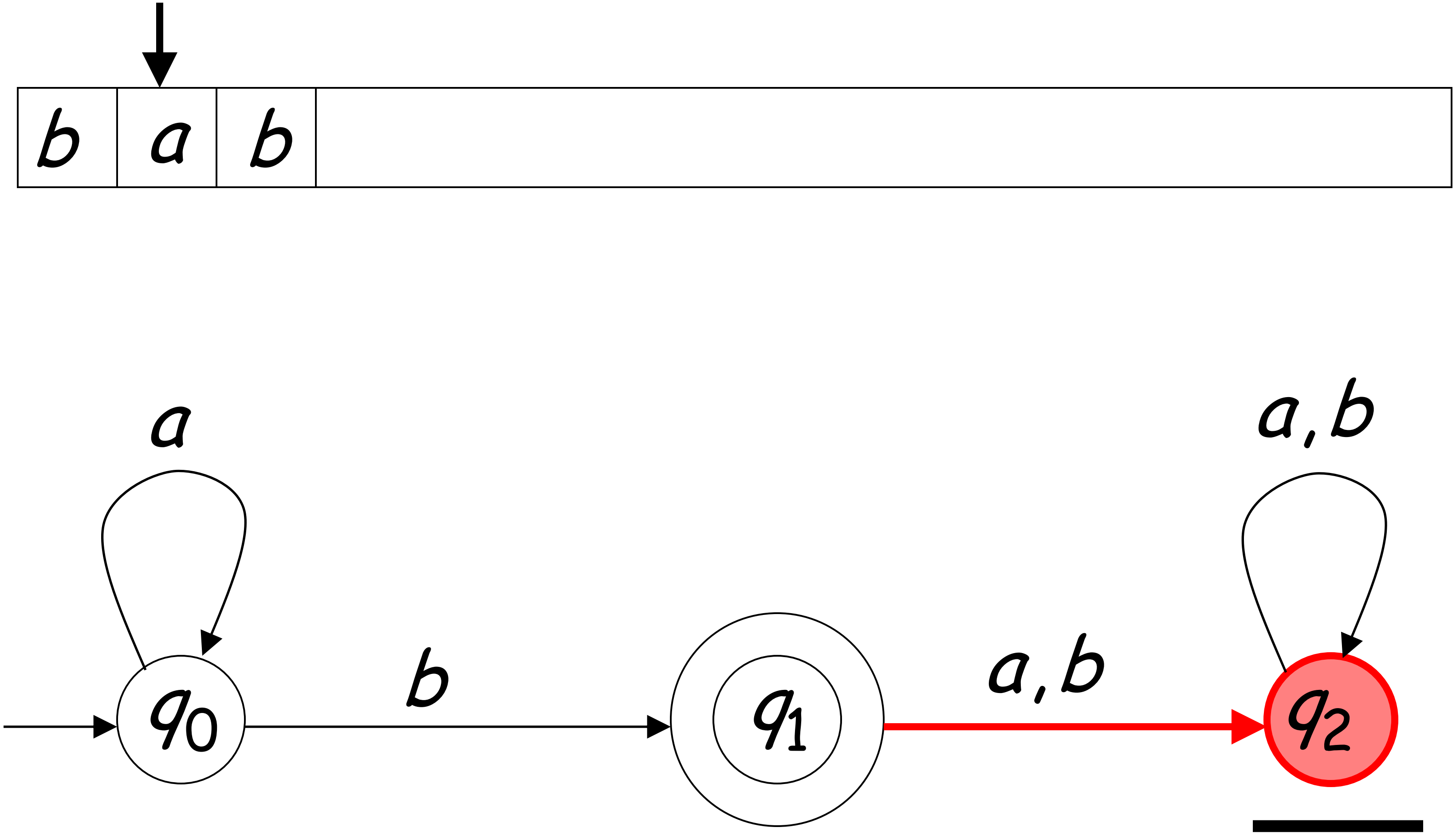


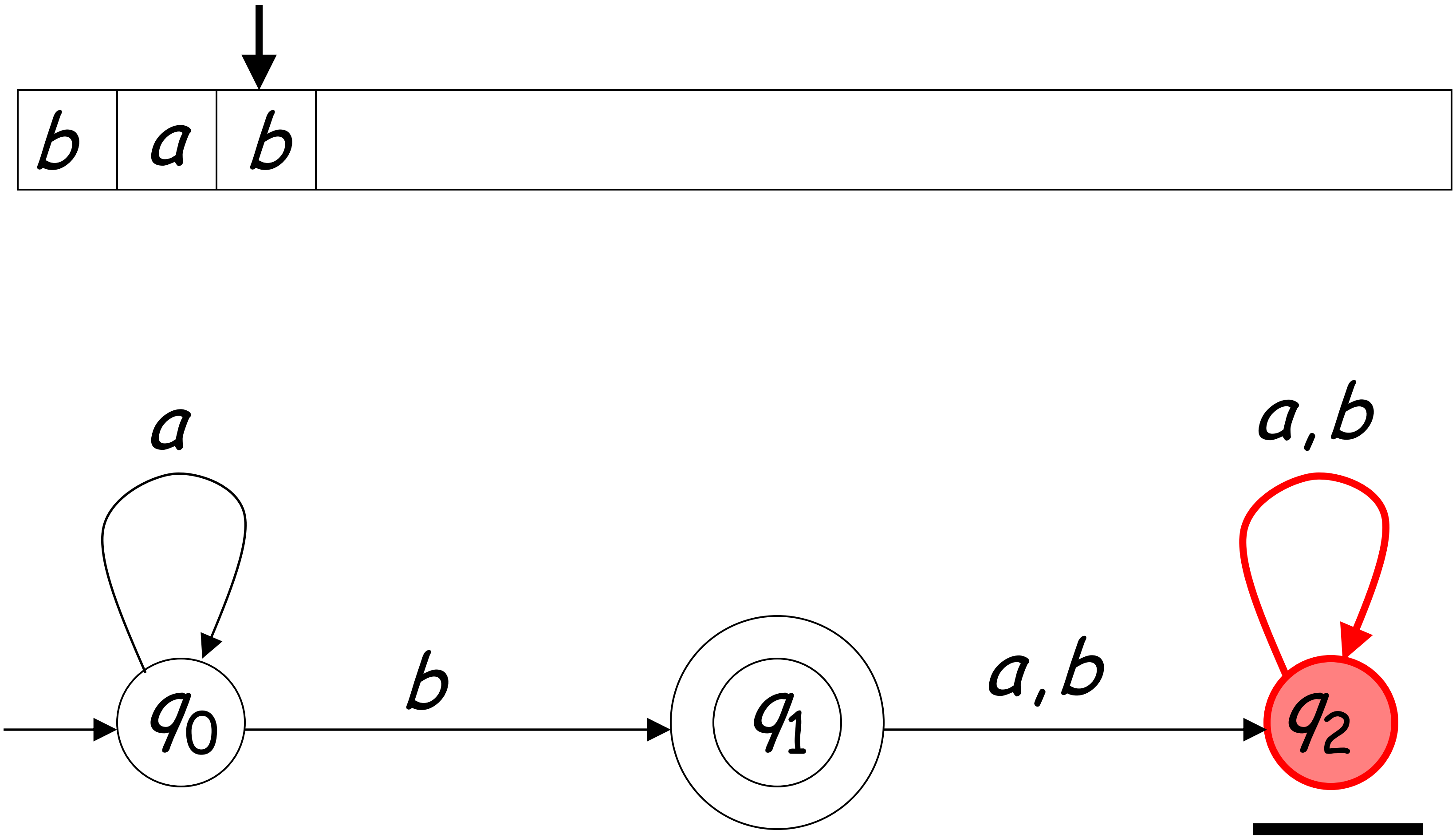
Output: "accept"

Rejection

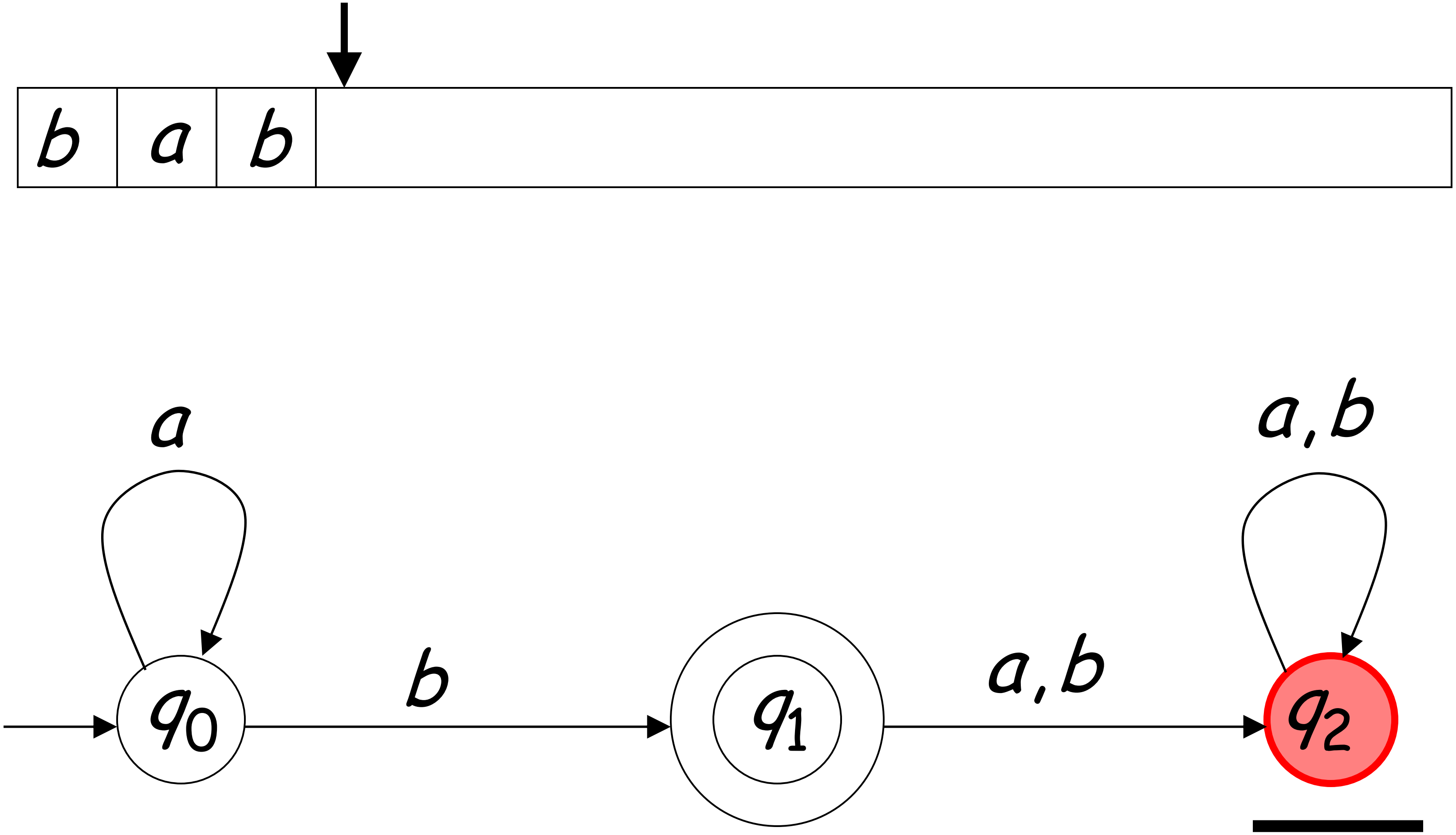



b
a
b
a
a,b
b
a,b
q_0
q_1
q_2


b
a
b
a
a,b
b
a,b
q_0
q_1
q_2


b
a
b
a
b
a,b
a,b
q_0
q_1
q_2

Input finished



Output: “reject”

Formalities

Deterministic Finite Accepter (DFA)

$$M = (Q, \Sigma, \delta, q_0, F)$$

Q : set of states

Σ : input alphabet

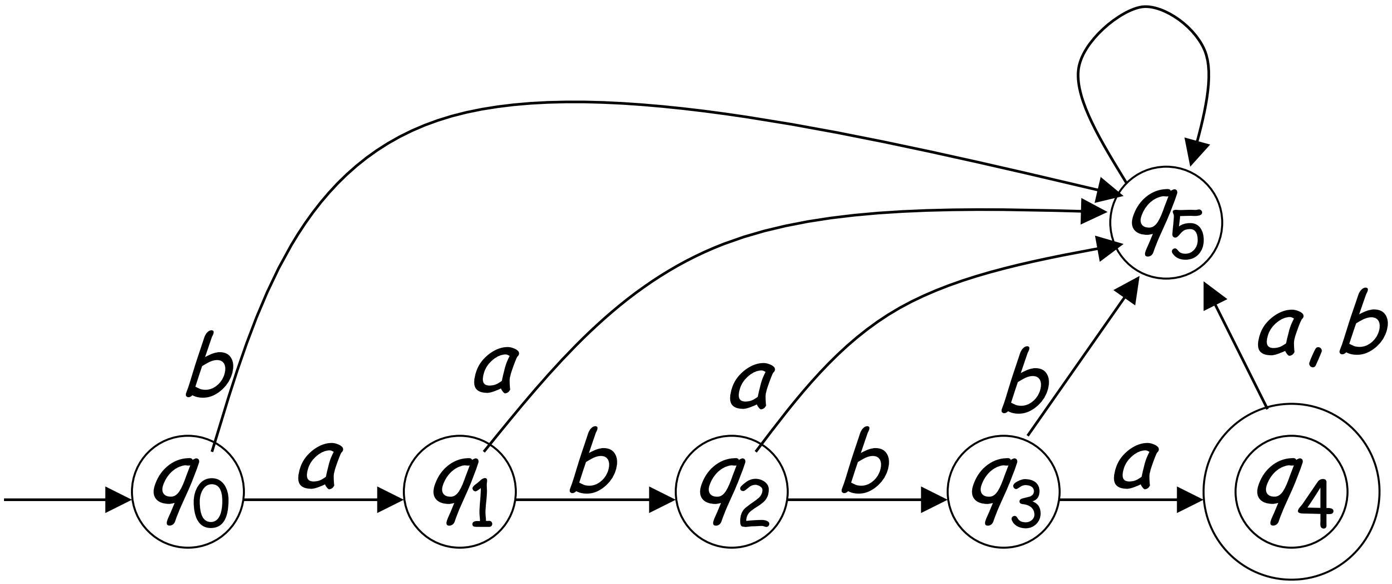
δ : transition function

q_0 : initial state

F : set of final states

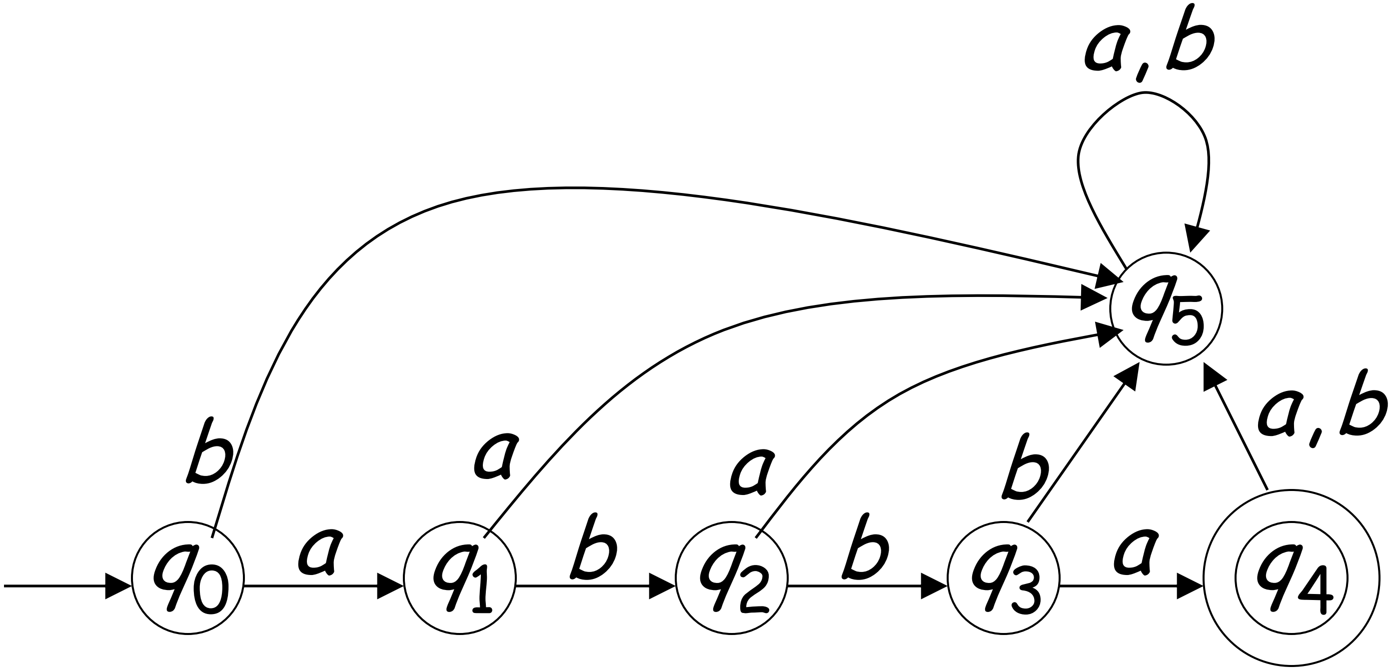
Input Alphabet Σ

$$\Sigma = \{a, b\}$$

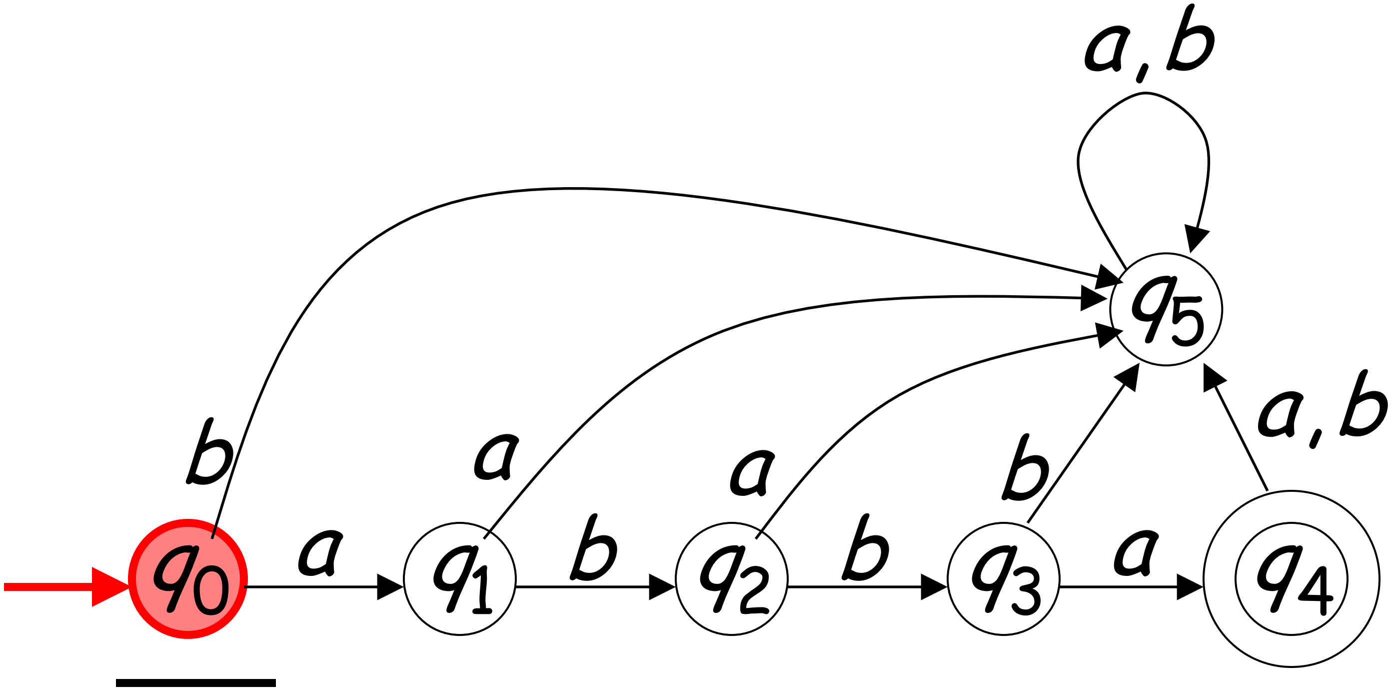


Set of States Q

$$Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$$

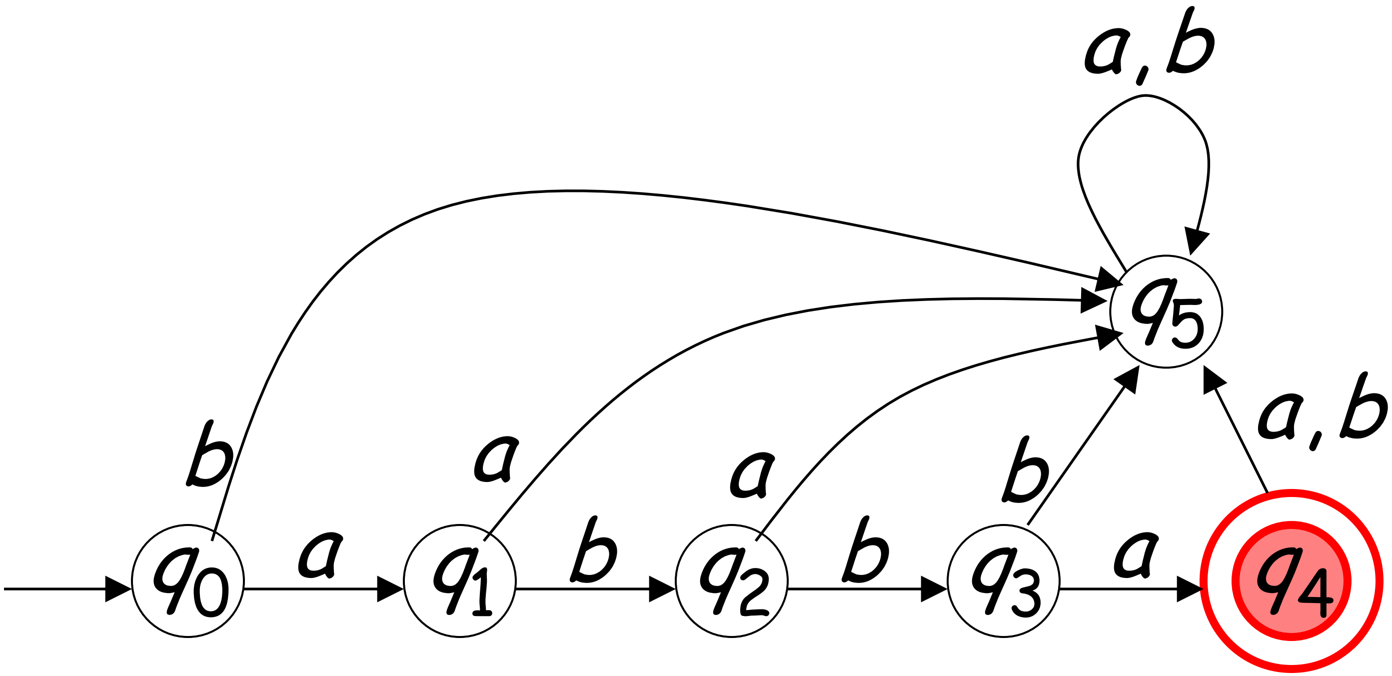


Initial State q_0



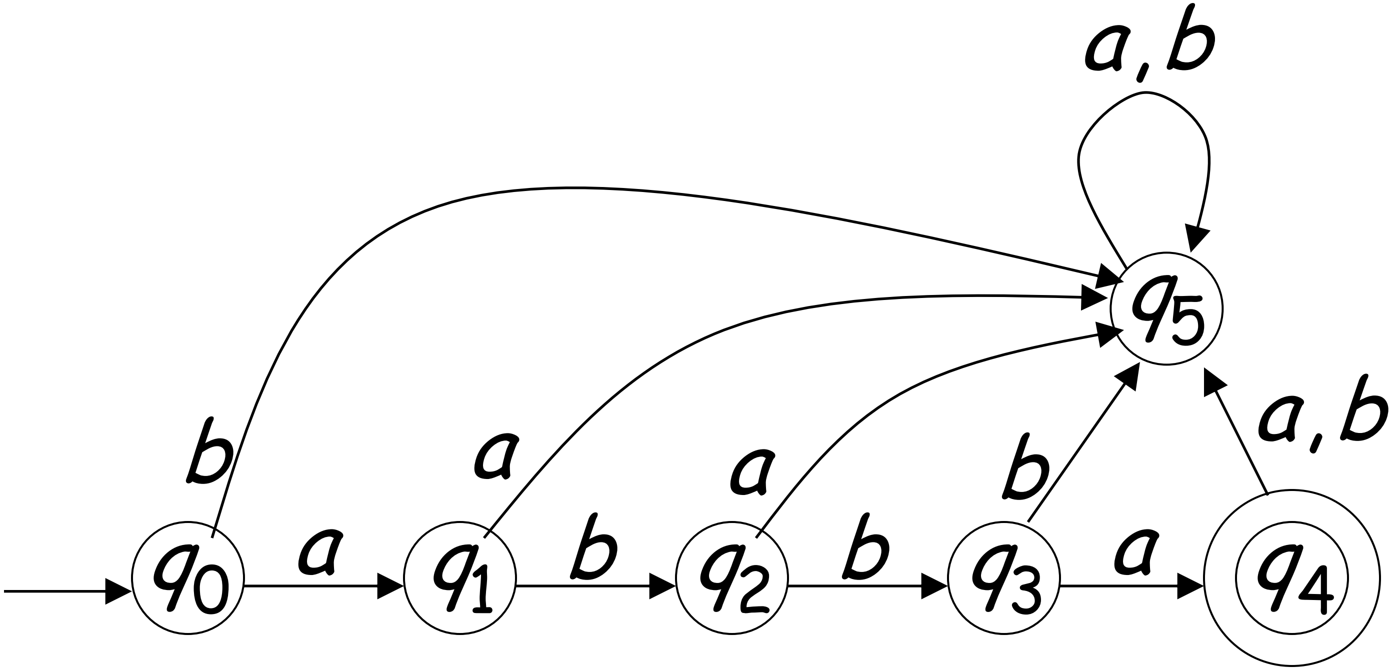
Set of Final States F

$$F = \{q_4\}$$

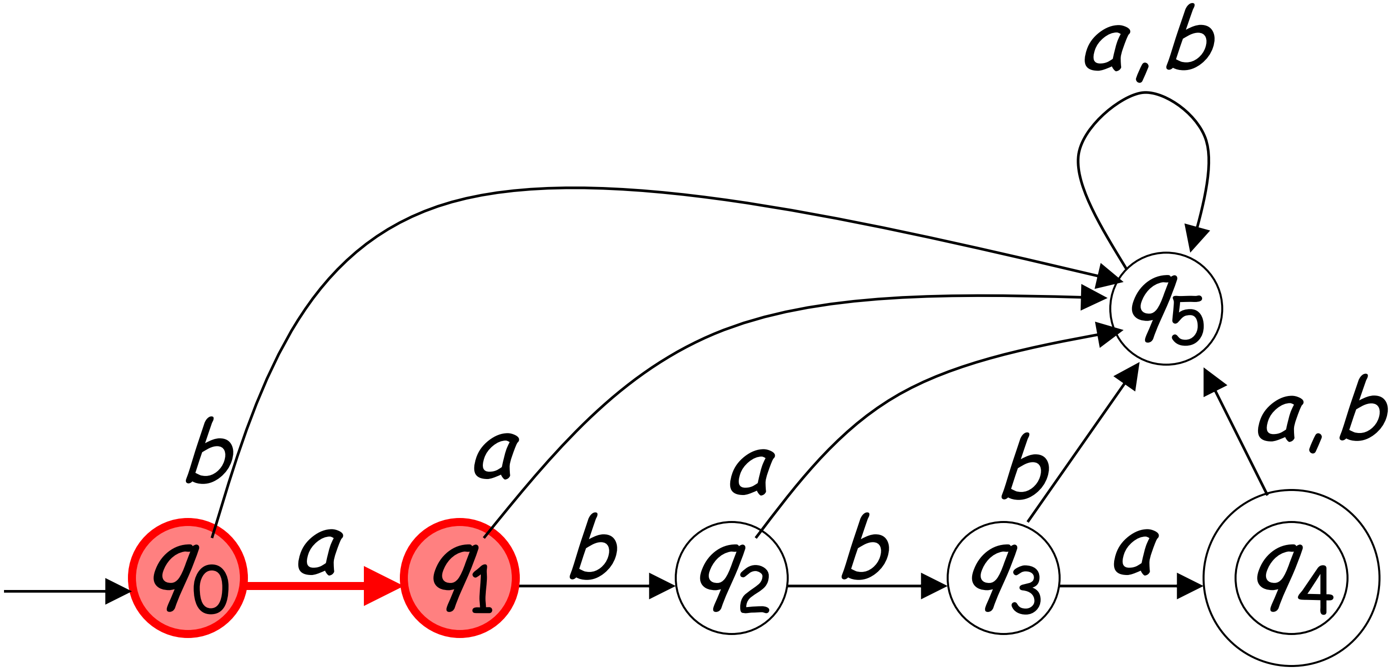


Transition Function δ

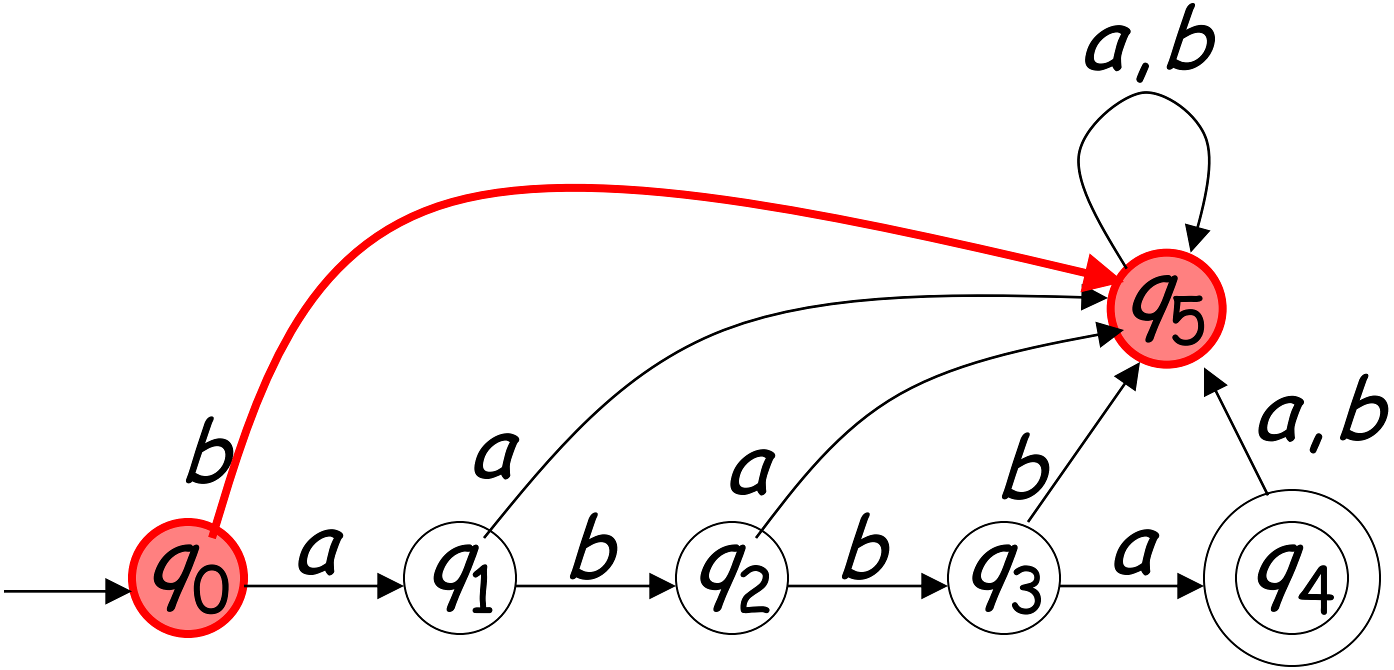
$$\delta : Q \times \Sigma \rightarrow Q$$



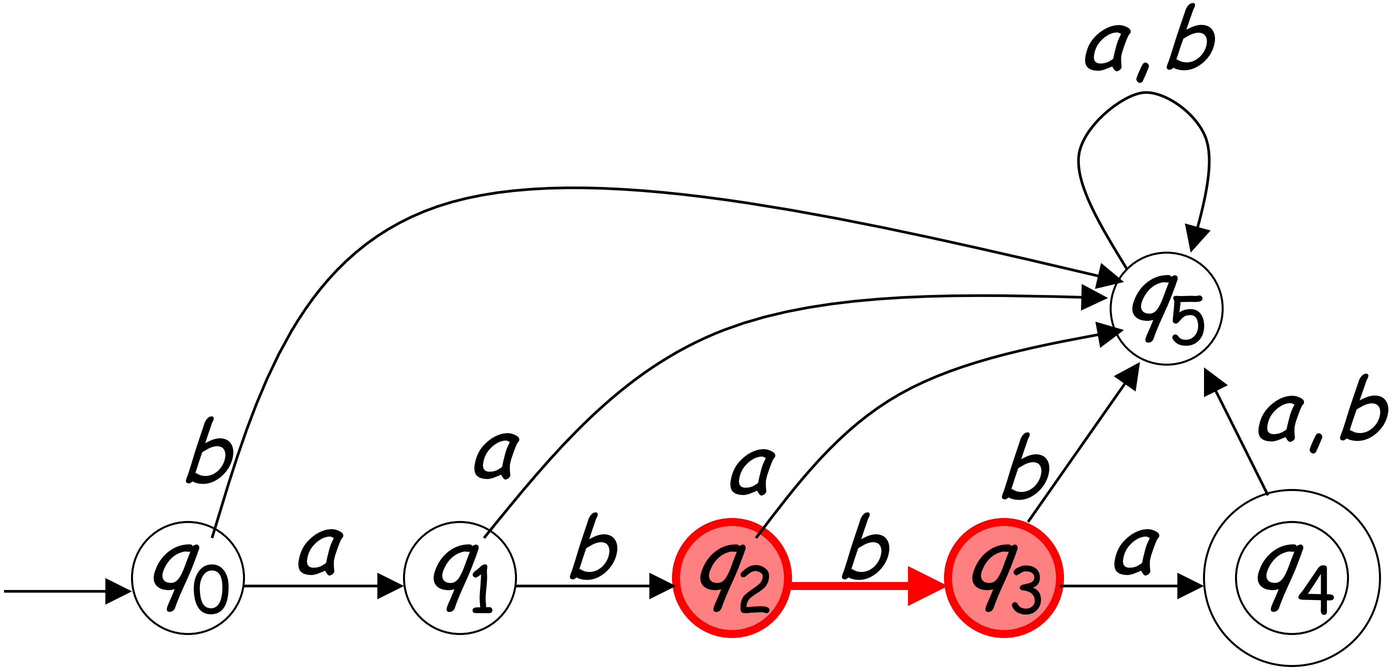
$$\delta(q_0, a) = q_1$$



$$\delta(q_0, b) = q_5$$

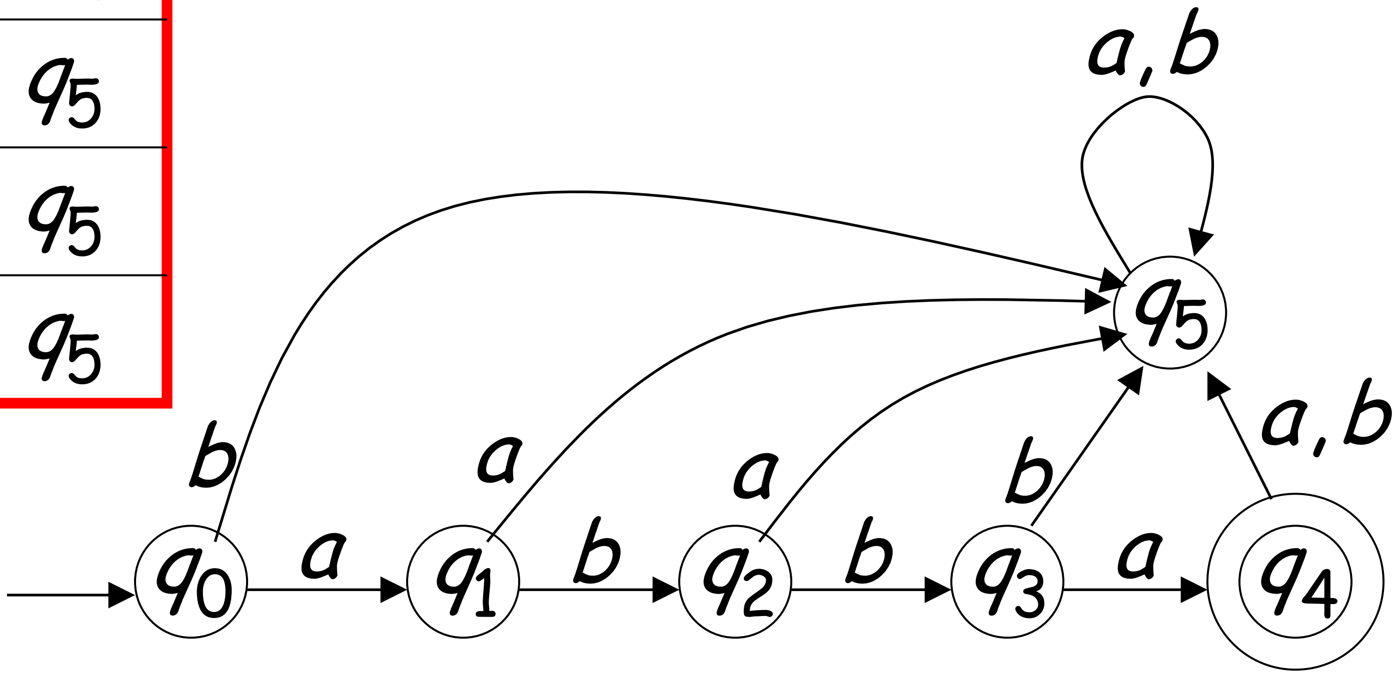


$$\delta(q_2, b) = q_3$$



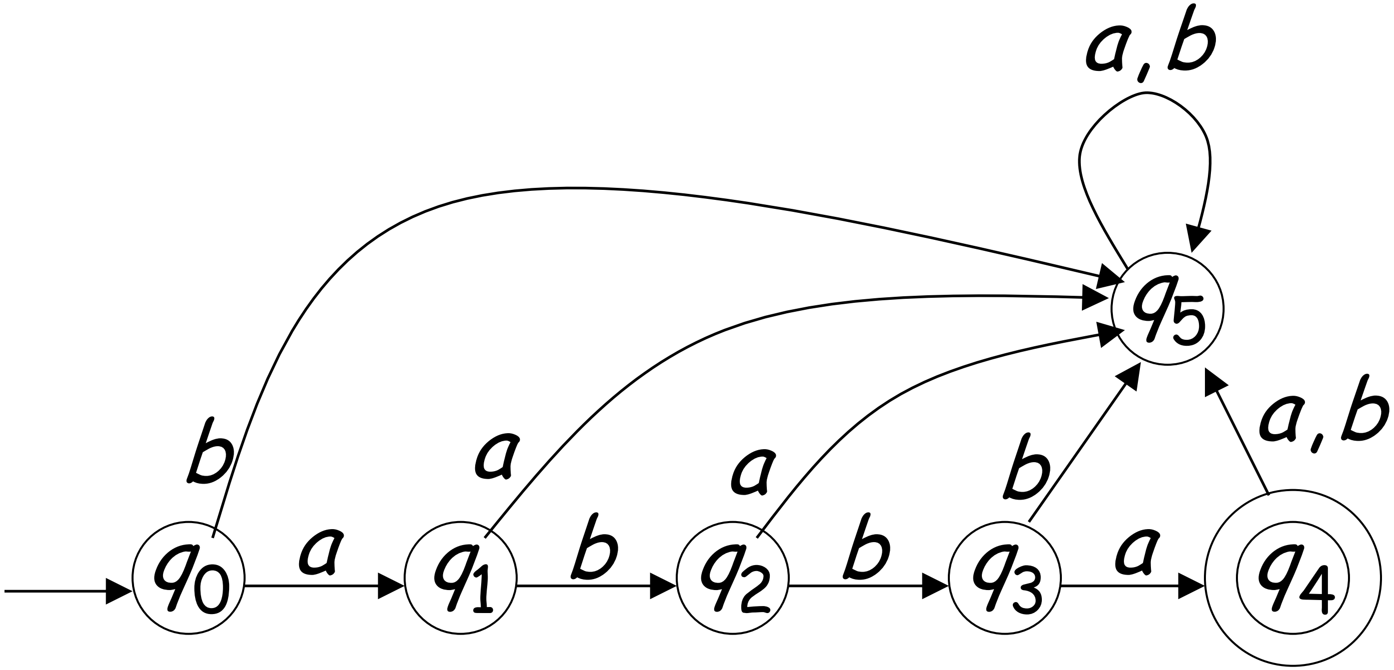
Transition Function δ

δ	a	b
q_0	q_1	q_5
q_1	q_5	q_2
q_2	q_5	q_3
q_3	q_4	q_5
q_4	q_5	q_5
q_5	q_5	q_5

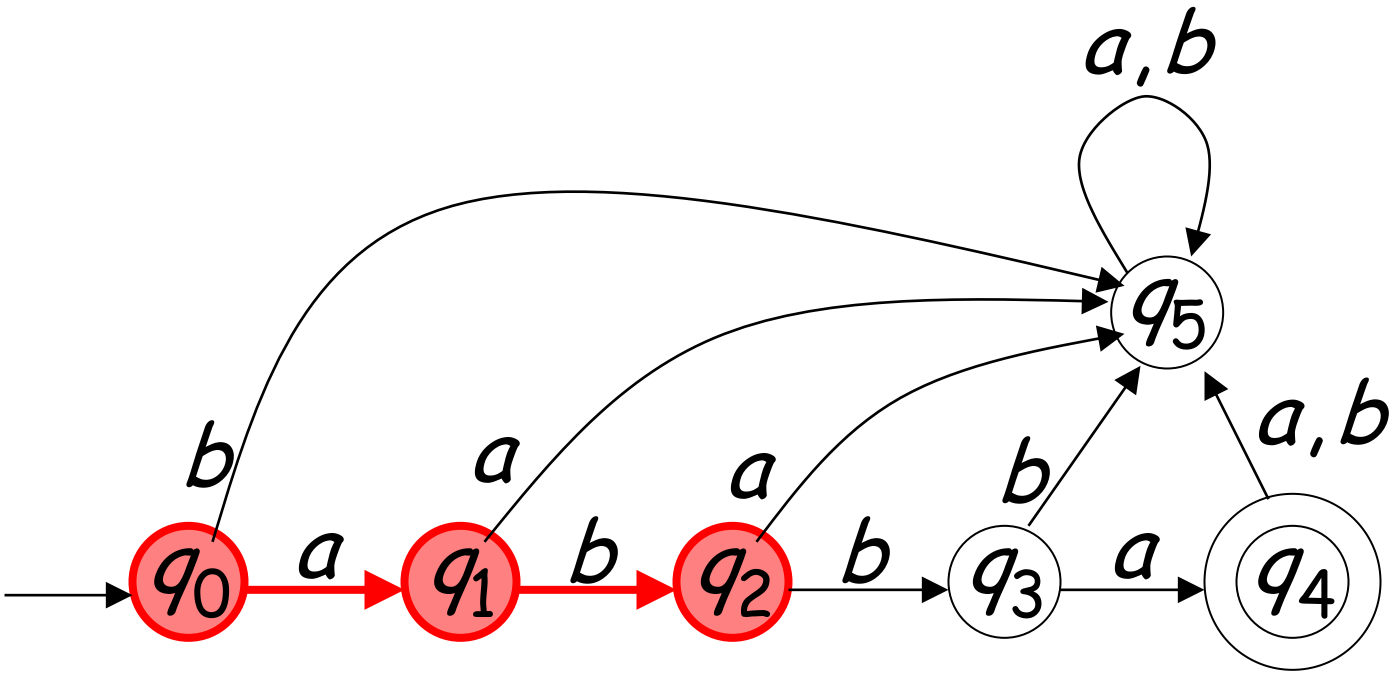


Extended Transition Function $\delta *$

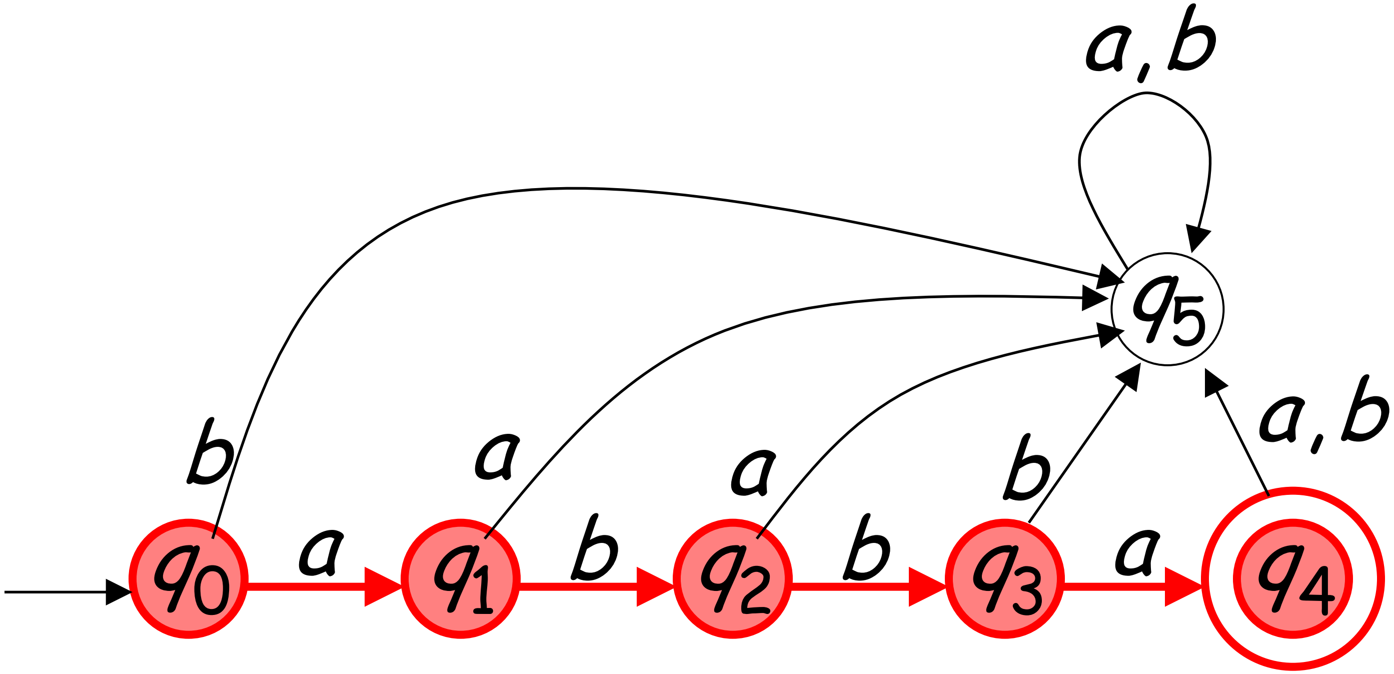
$$\delta * : Q \times \Sigma * \rightarrow Q$$



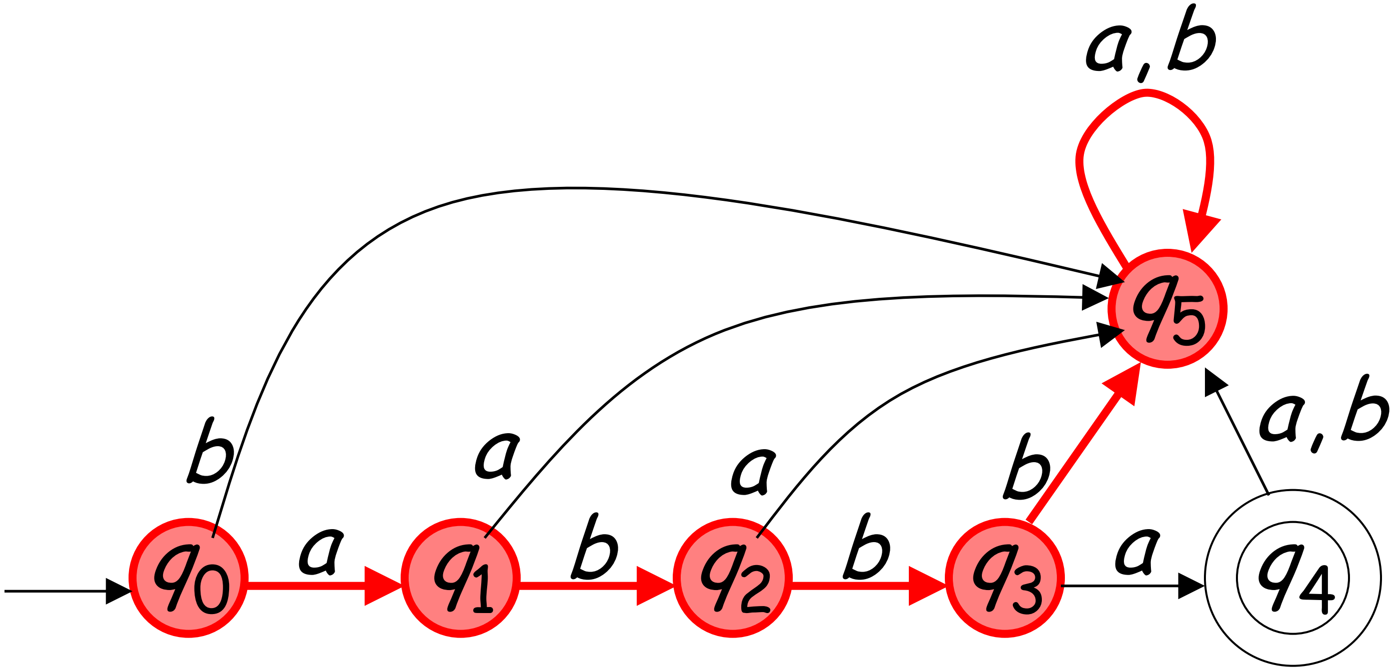
$$\delta^*(q_0, ab) = q_2$$



$$\delta^*(q_0, abba) = q_4$$

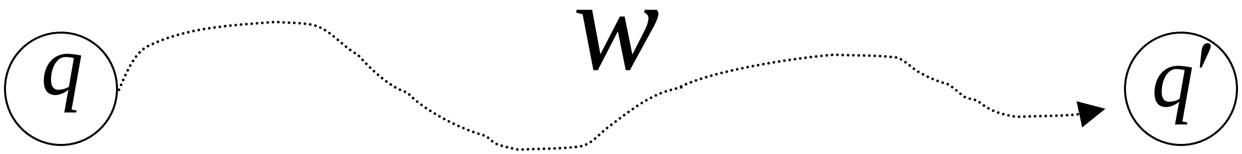


$$\delta^*(q_0, abbbaa) = q_5$$

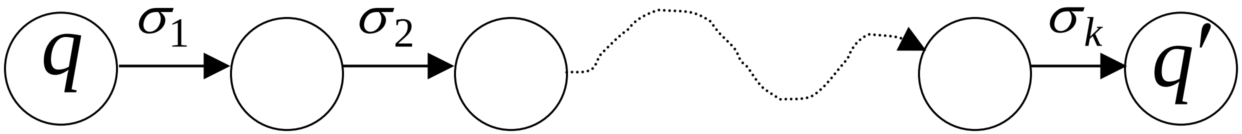


Observation: There is a walk from q to q' with label w

$$\delta^*(q,w)=q'$$

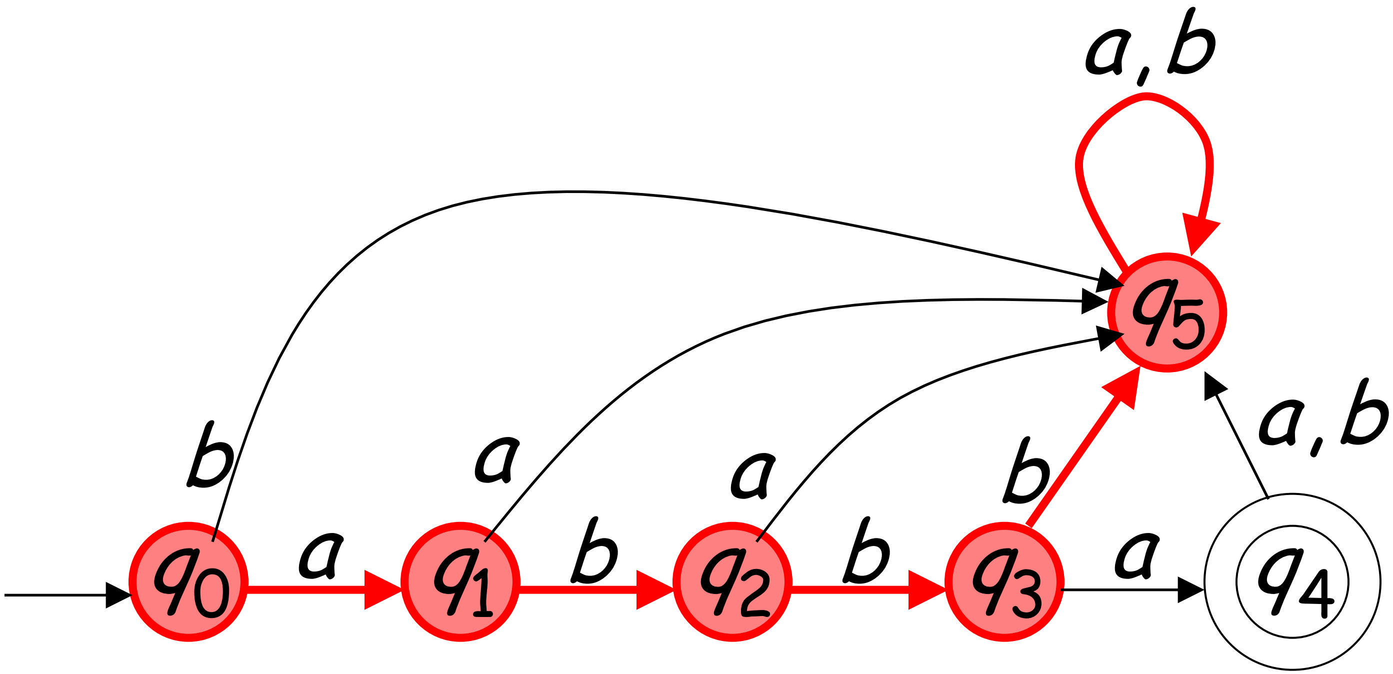


$$w=\sigma_1\sigma_2\cdots\sigma_k$$



Example: There is a walk from q_0 to q_5 with label $abbbaa$

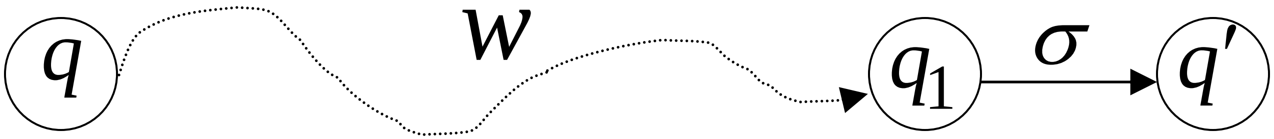
$$\delta^*(q_0, abbbaa) = q_5$$



Recursive Definition

$$\delta^*(q,\lambda)=q$$

$$\delta^*(q,w\sigma)=\delta(\delta^*(q,w),\sigma)$$



$$\left.\begin{array}{l} \delta^*(q,w\sigma)=q' \\ \\ \delta(q_1,\sigma)=q' \end{array}\right\} \Rightarrow \delta^*(q,w\sigma)=\delta(q_1,\sigma)$$

$$\left.\begin{array}{l} \delta^*(q,w\sigma)=\delta(q_1,\sigma) \\ \\ \delta^*(q,w)=q_1 \end{array}\right\} \Rightarrow \delta^*(q,w\sigma)=\delta(\delta^*(q,w),\sigma)$$

$$\delta^*(q_0, ab) =$$

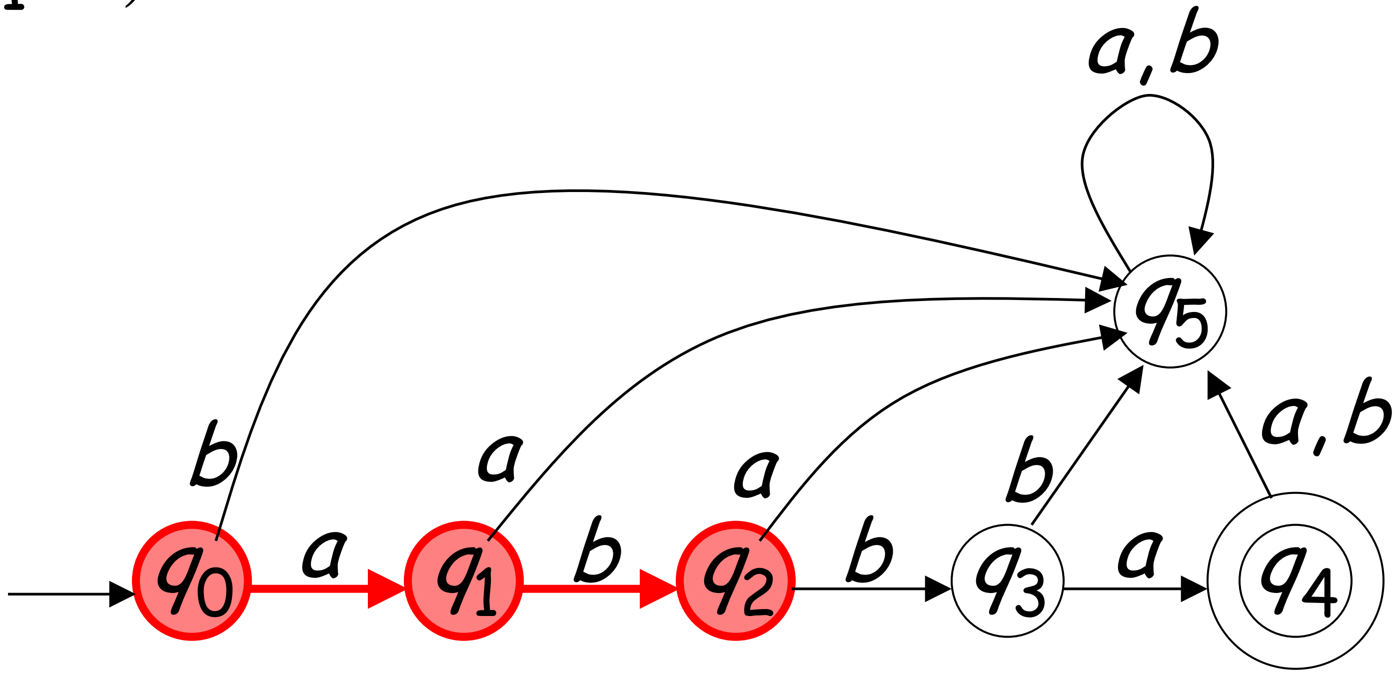
$$\delta(\delta^*(q_0, a), b) =$$

$$\delta(\delta(\delta^*(q_0, \lambda), a), b) =$$

$$\delta(\delta(q_0, a), b) =$$

$$\delta(q_1, b) =$$

$$q_2$$



Languages Accepted by DFAs

Take DFA M

Definition:

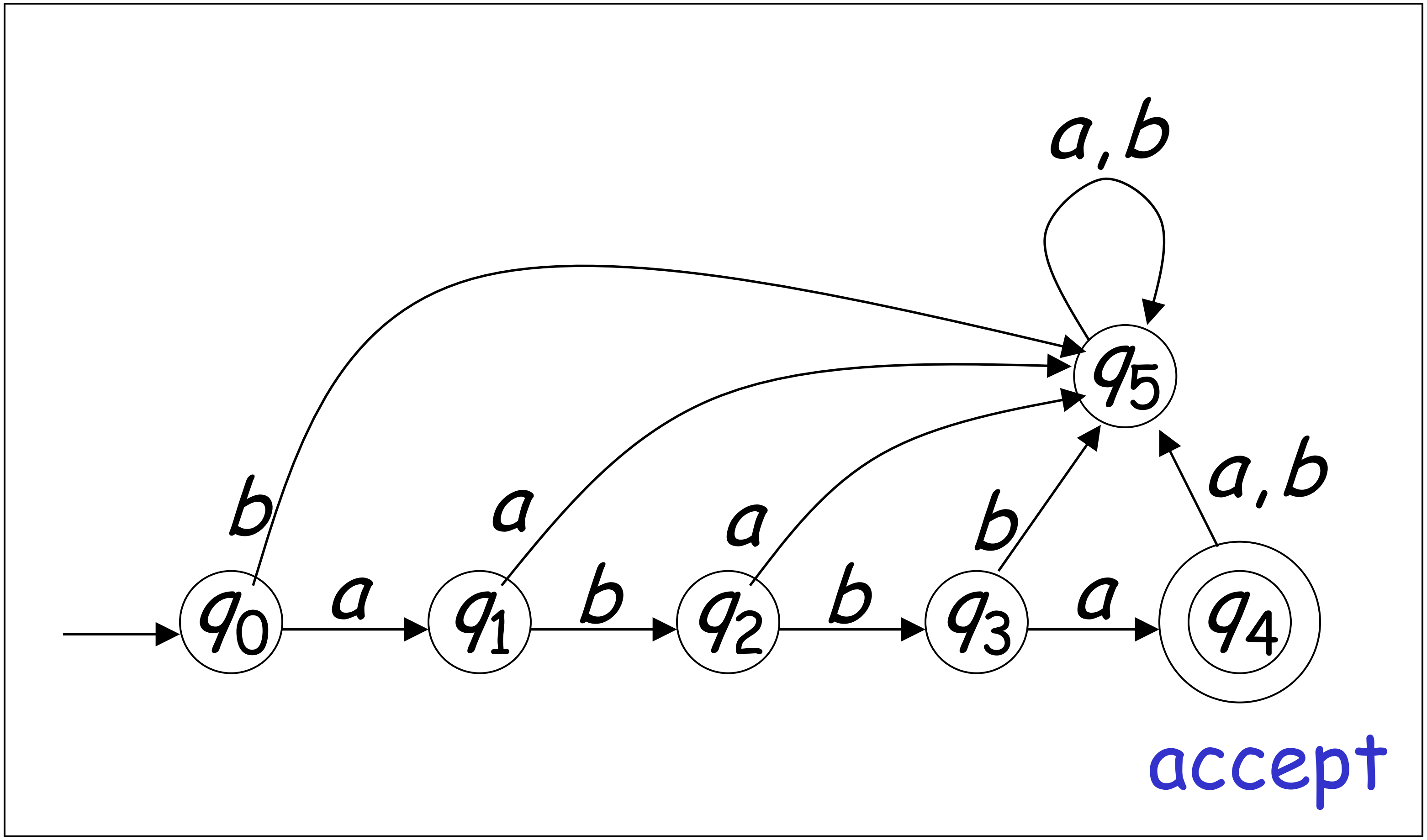
The language $L(M)$ contains all input strings accepted by M

$L(M)$ = { strings that drive M to a final state}

Example

$L(M) = \{abba\}$

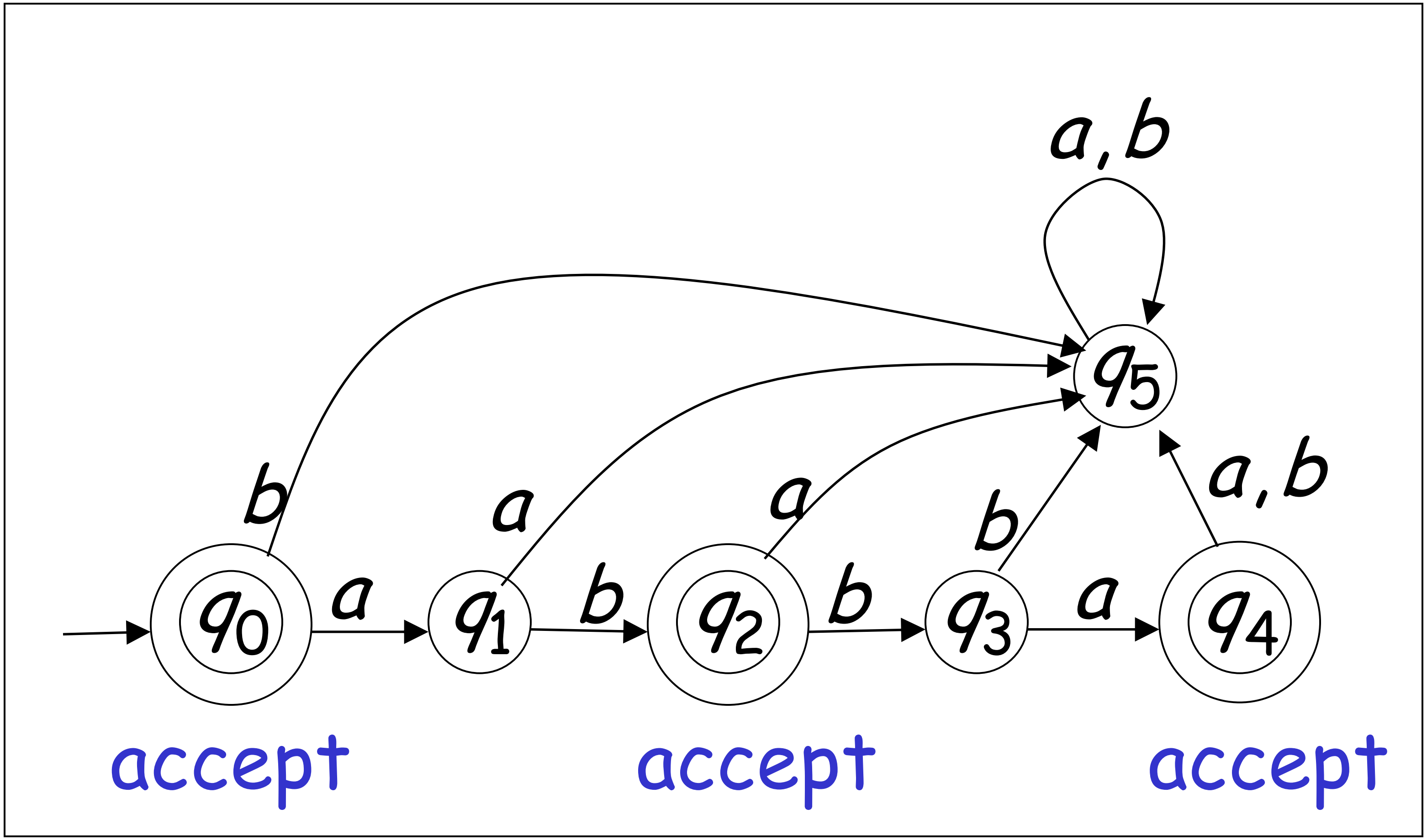
M



Another Example

$$L(M) = \{\lambda, ab, abba\}$$

M

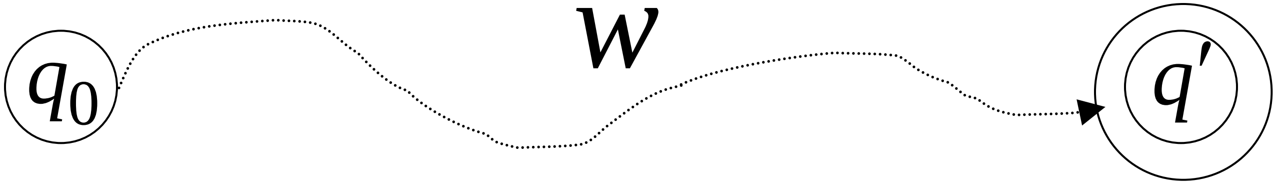


Formally

For a DFA $M = (Q, \Sigma, \delta, q_0, F)$

Language accepted by M :

$$L(M) = \{ w \in \Sigma^* : \delta^*(q_0, w) \in F \}$$

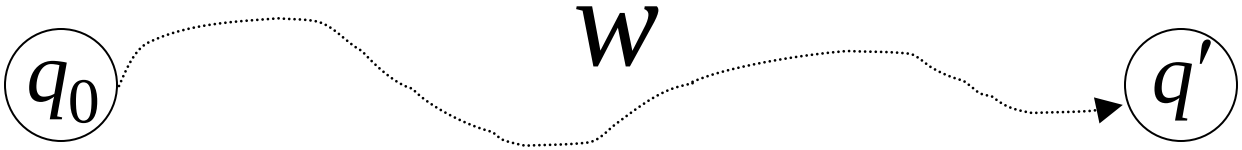


$q' \in F$

Observation

Language rejected by M :

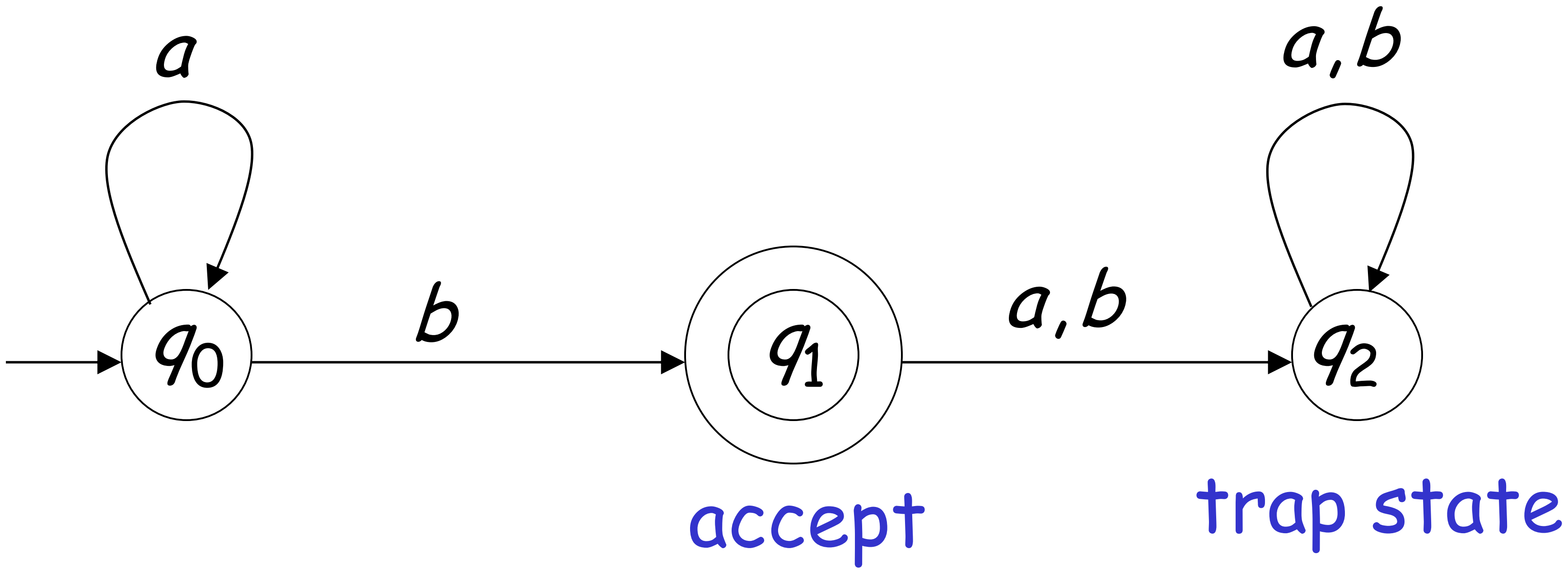
$$\overline{L(M)} = \{ w \in \Sigma^* : \delta^*(q_0, w) \notin F \}$$



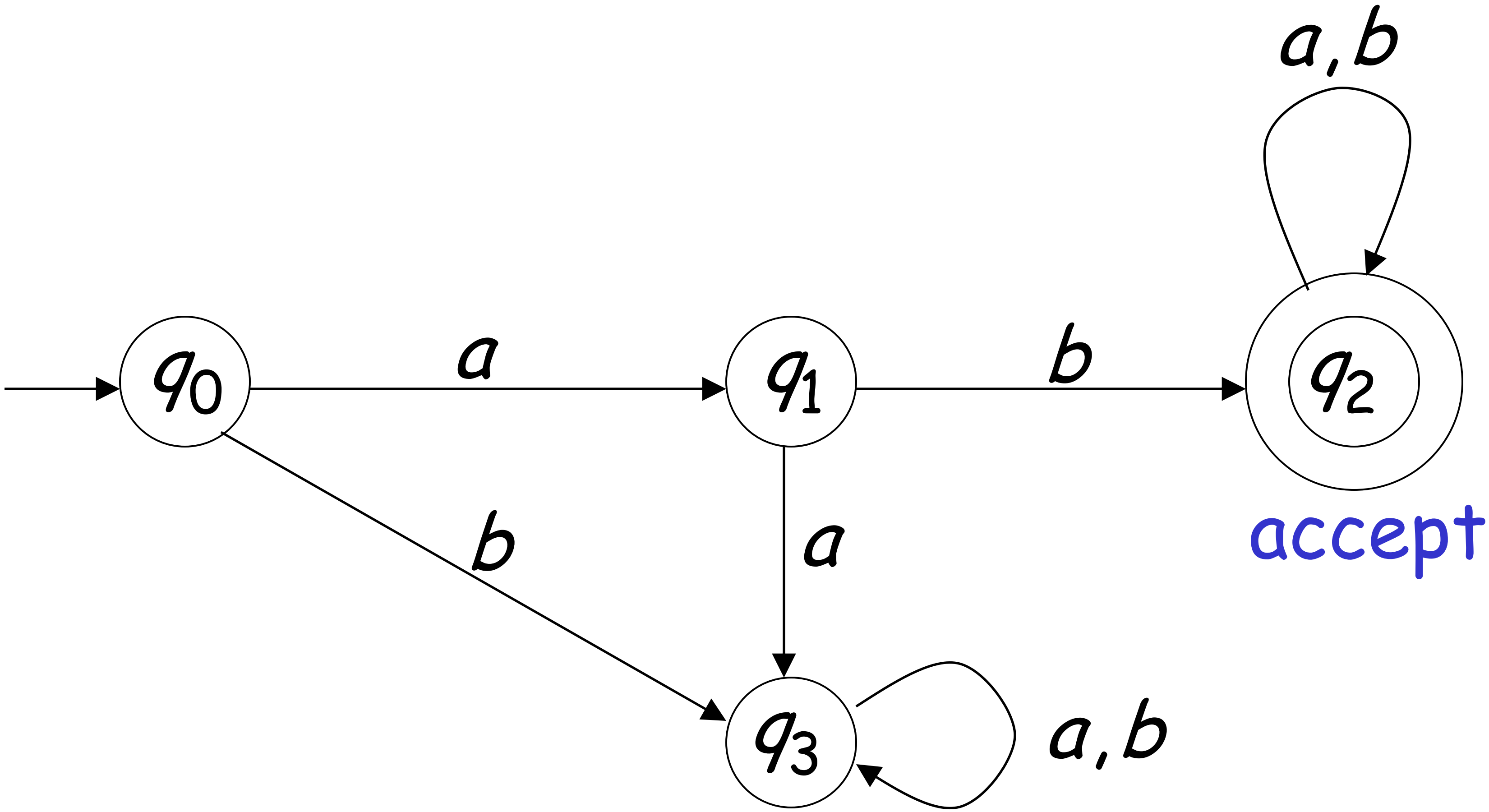
$q' \notin F$

More Examples

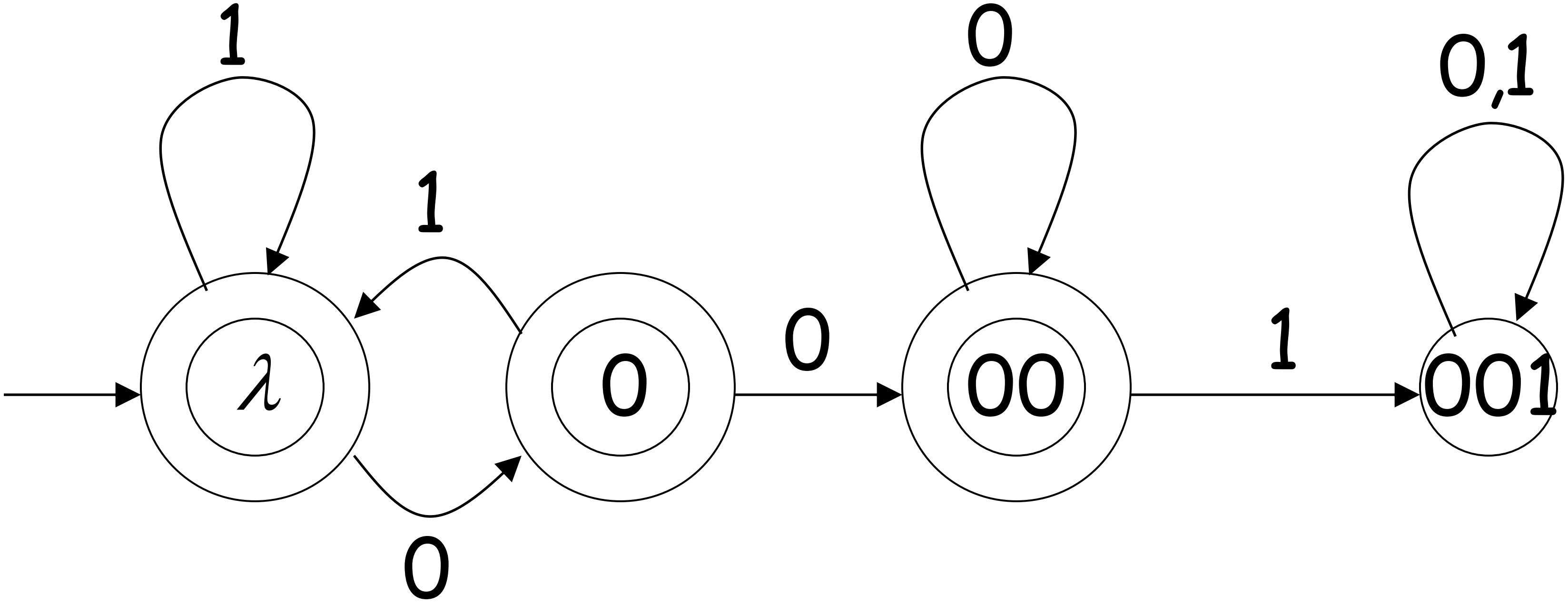
$$L(M)=\{a^n b : n \geq 0\}$$



$L(M)$ = { all strings with prefix ab }



$L(M) =$ { all strings without substring 001 }



Regular Languages

A language L is regular if there is a DFA M such that $L = L(M)$

All regular languages form a language family

Examples of regular languages:

$\{abba\}$ $\{\lambda, ab, abba\}$ $\{a^n b : n \geq 0\}$

{ all strings with prefix ab }

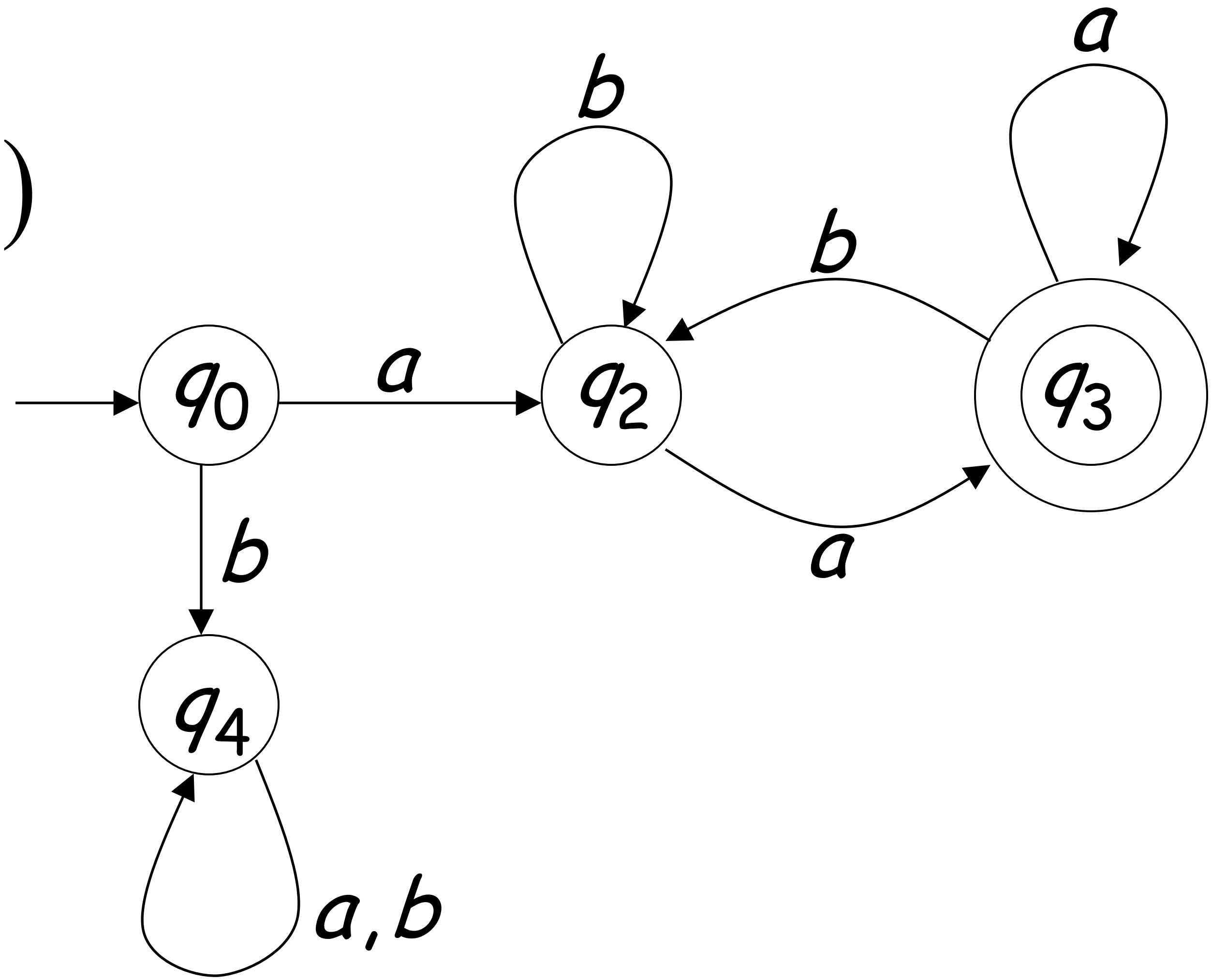
{ all strings without substring 001 }

There exist automata that accept these Languages (see previous slides).

Another Example

The language $L = \{awa : w \in \{a,b\}^*\}$ is regular:

$L = L(M)$



There exist languages which are <u>not</u> Regular:

Example: $L = \{a^n b^n : n \geq 0\}$

There is no DFA that accepts such a language

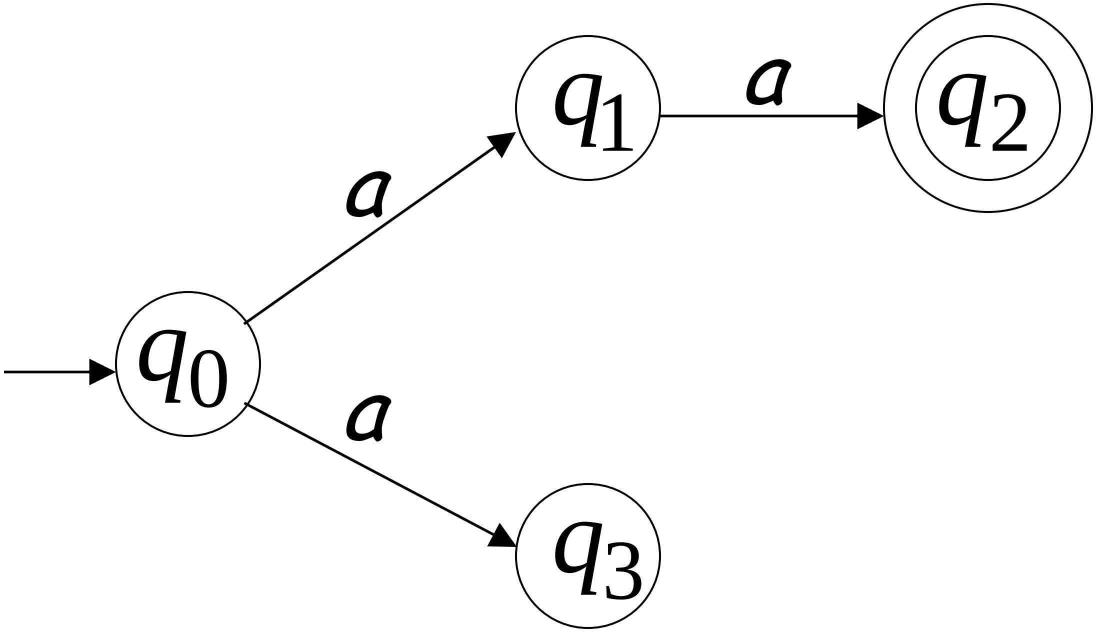
(we will prove this later in the class)

The End

Non Deterministic Automata

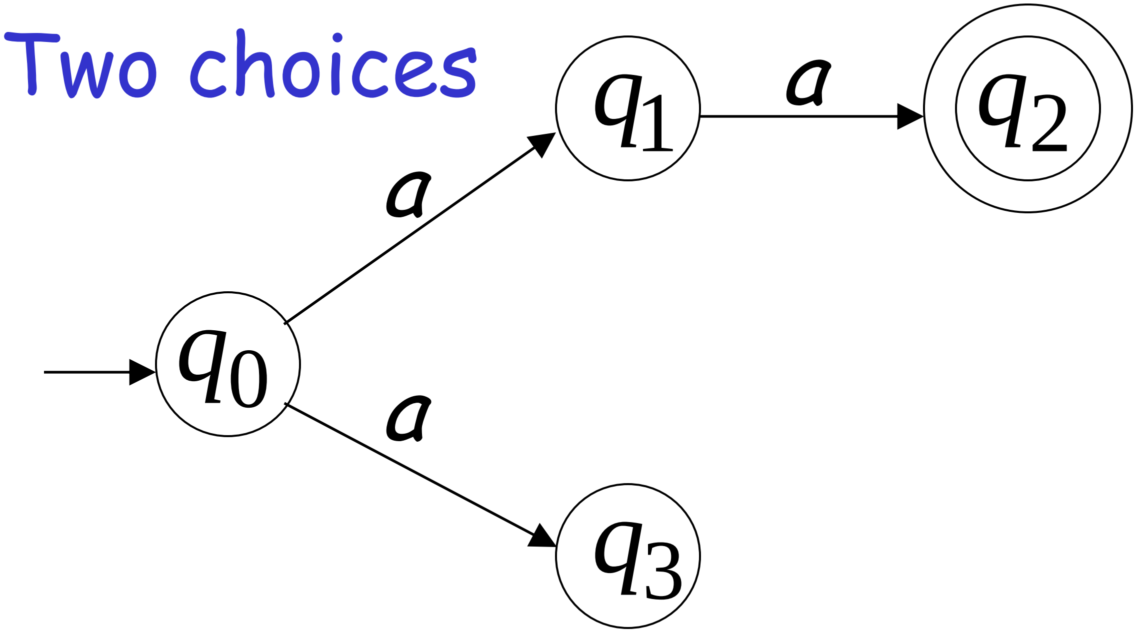
Nondeterministic Finite Accepter (NFA)

Alphabet = $\{a\}$



Nondeterministic Finite Accepter (NFA)

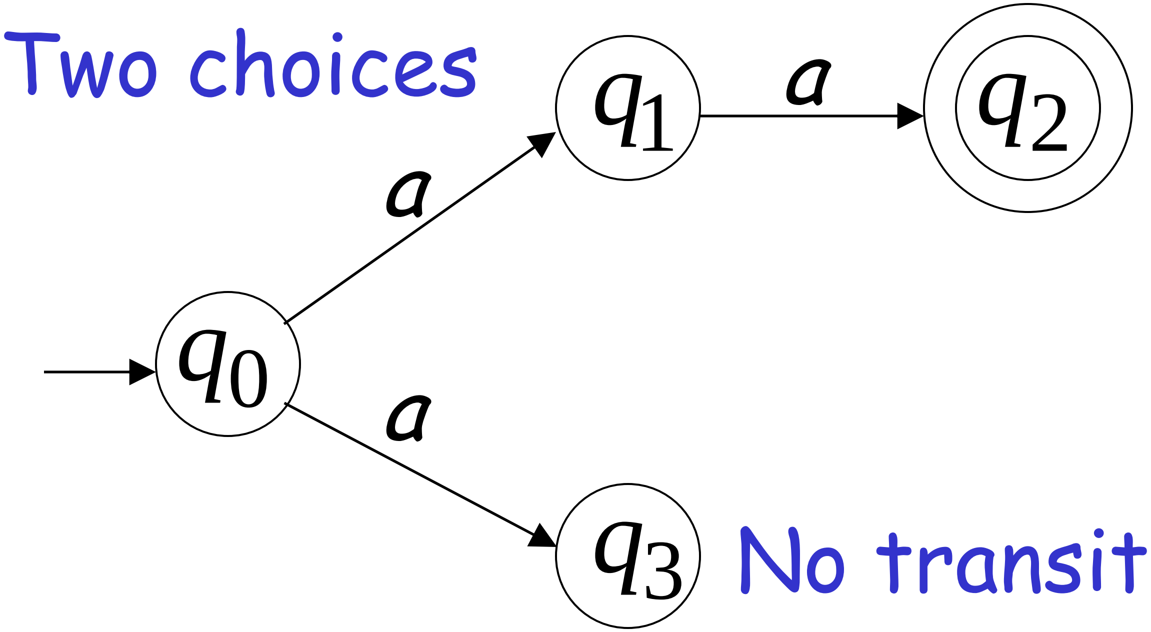
Alphabet = $\{a\}$


Two choices

Nondeterministic Finite Accepter (NFA)

Alphabet = $\{a\}$

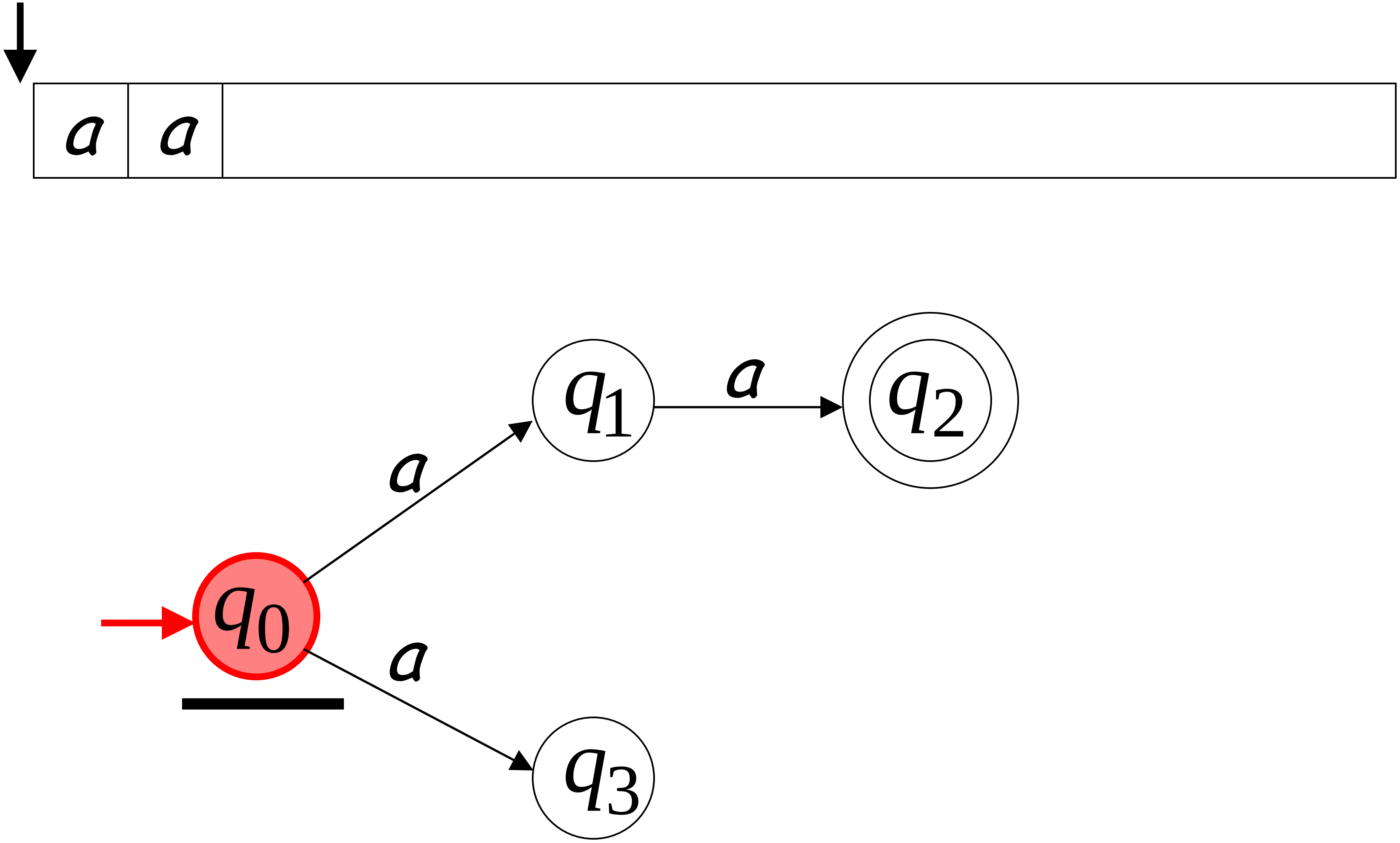
Two choices



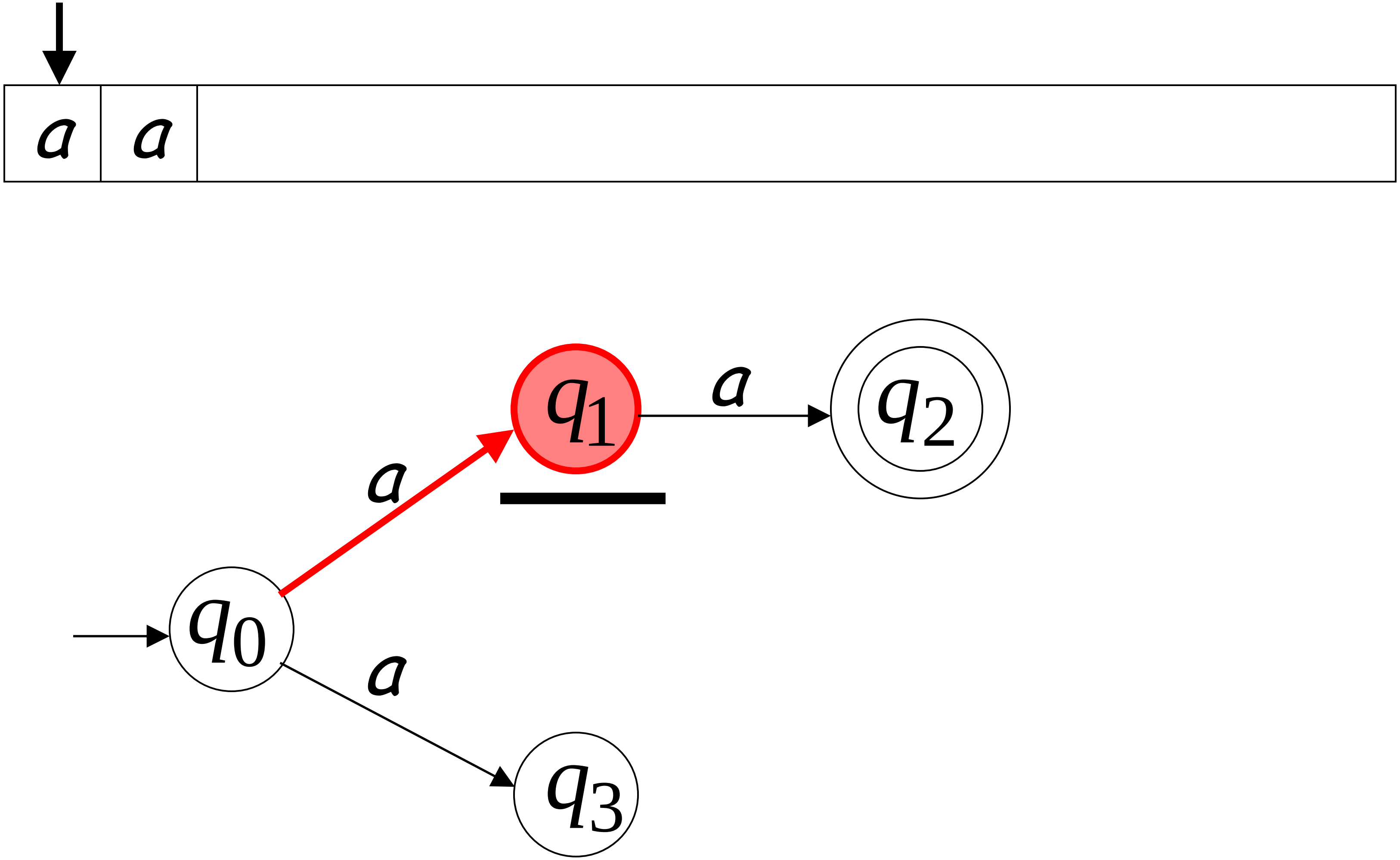
No transition

No transition

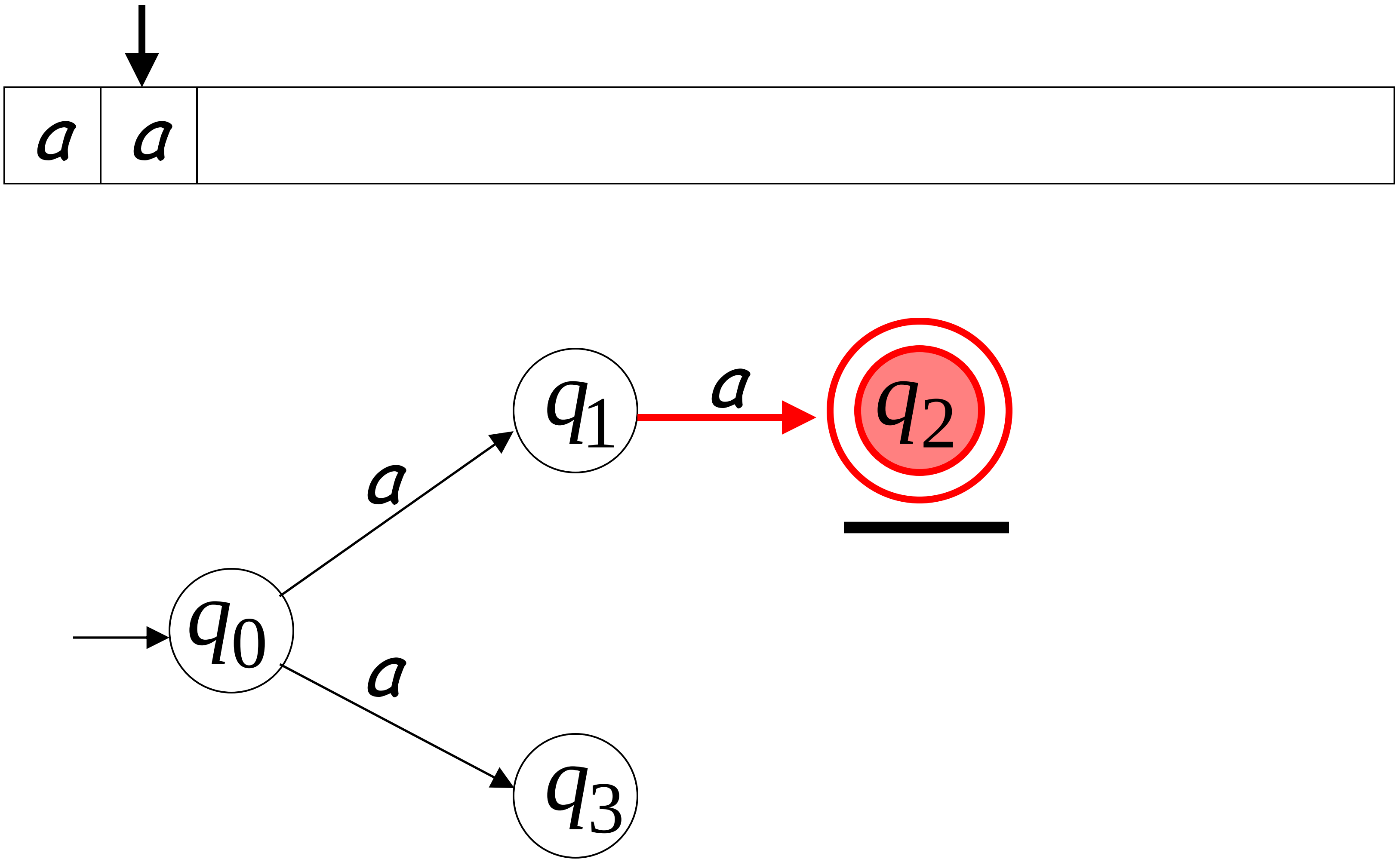
First Choice



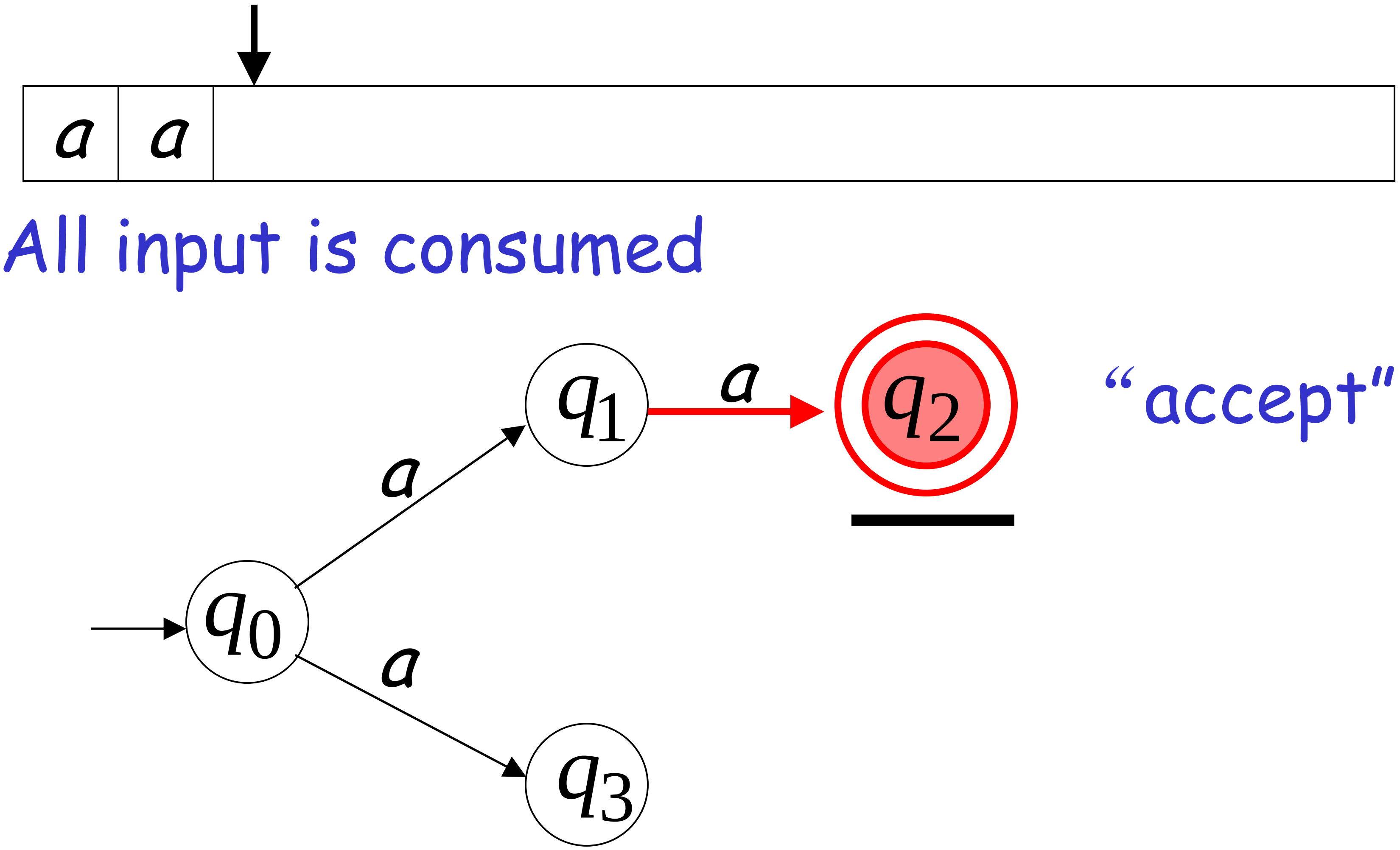
First Choice



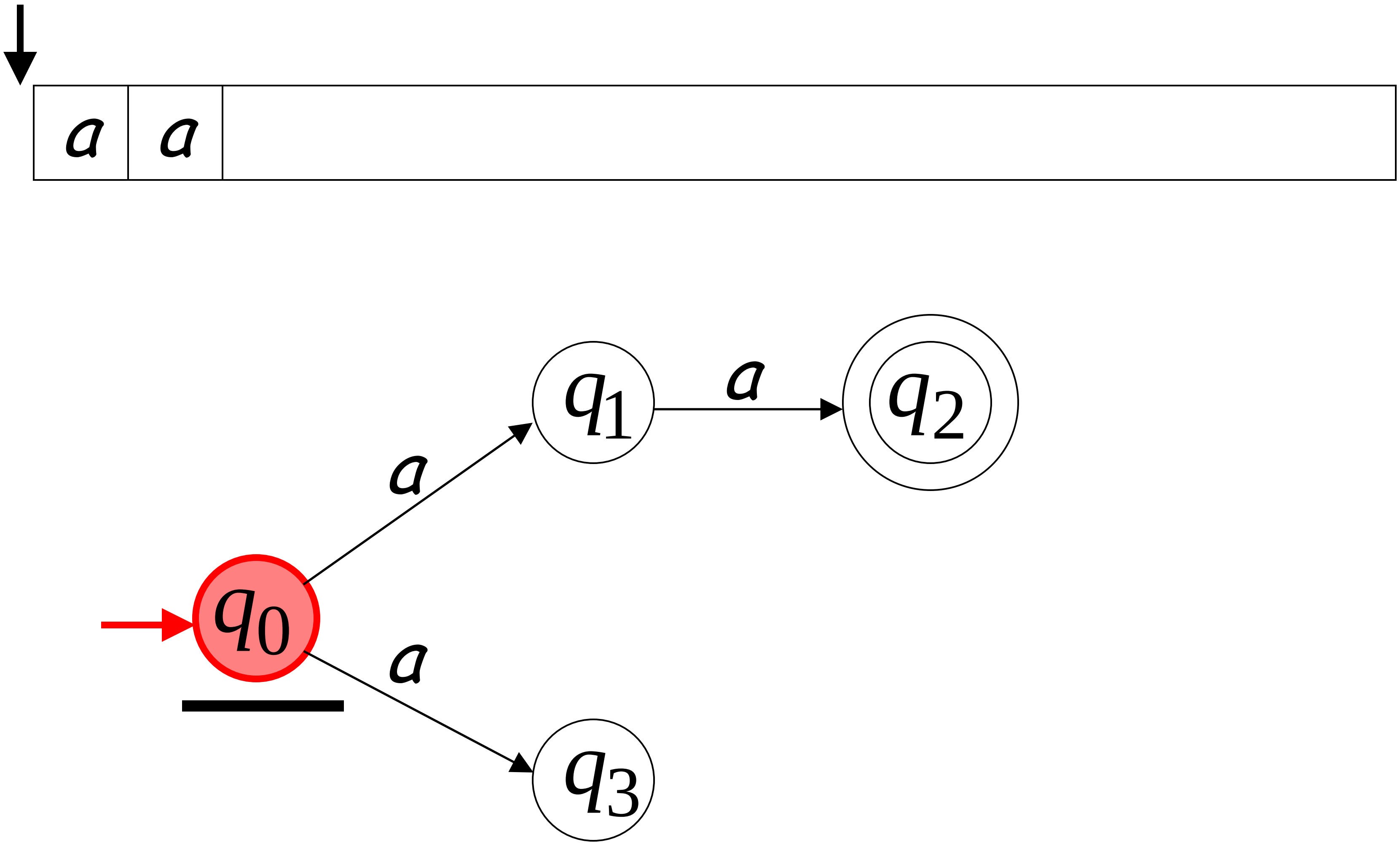
First Choice



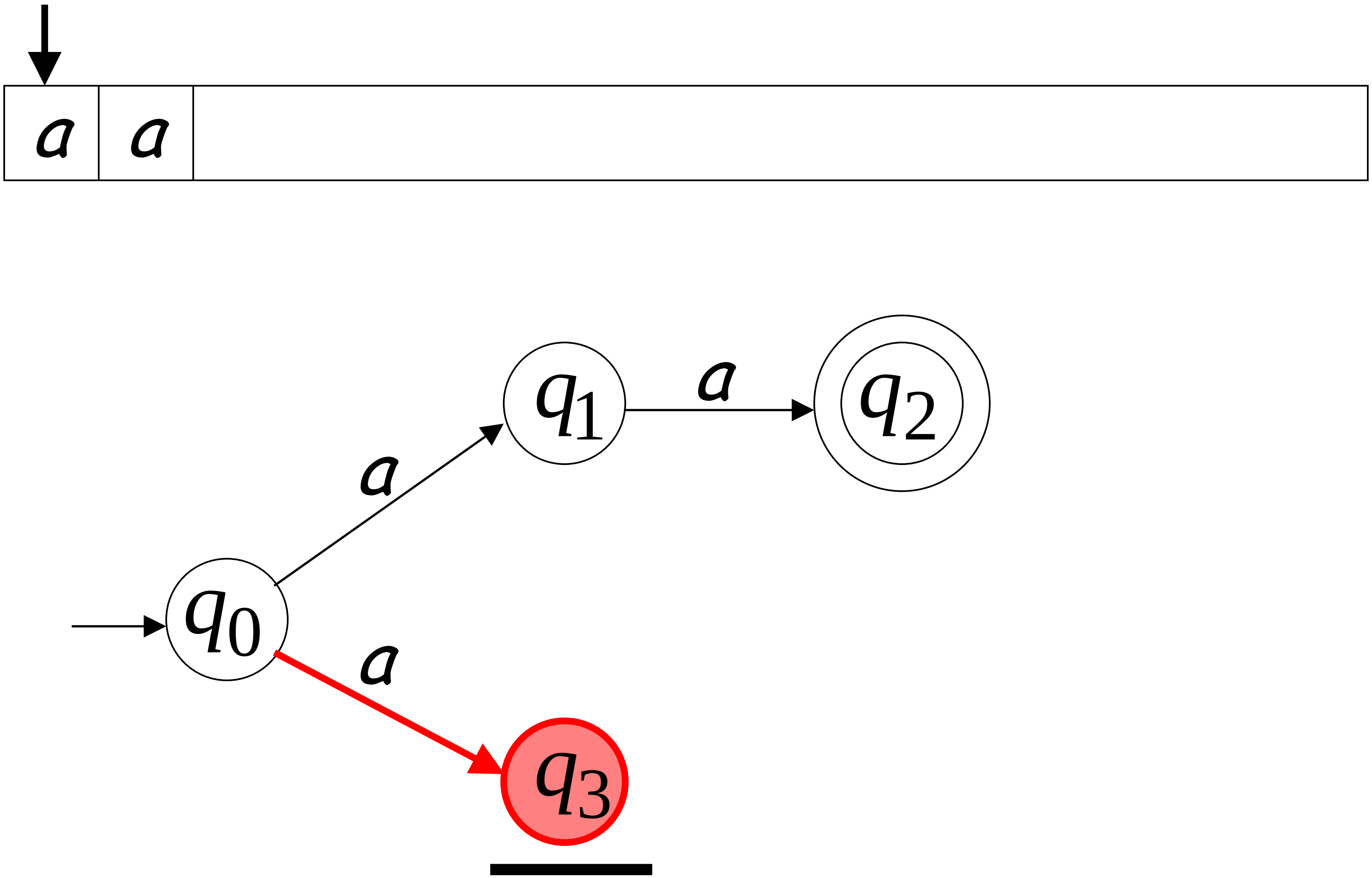
First Choice



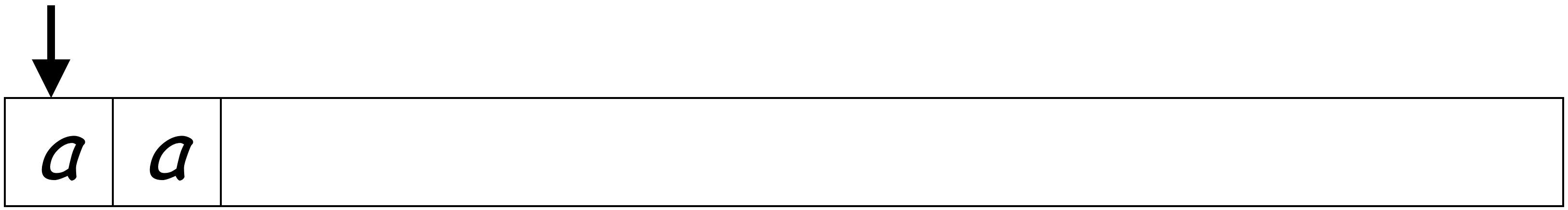
Second Choice

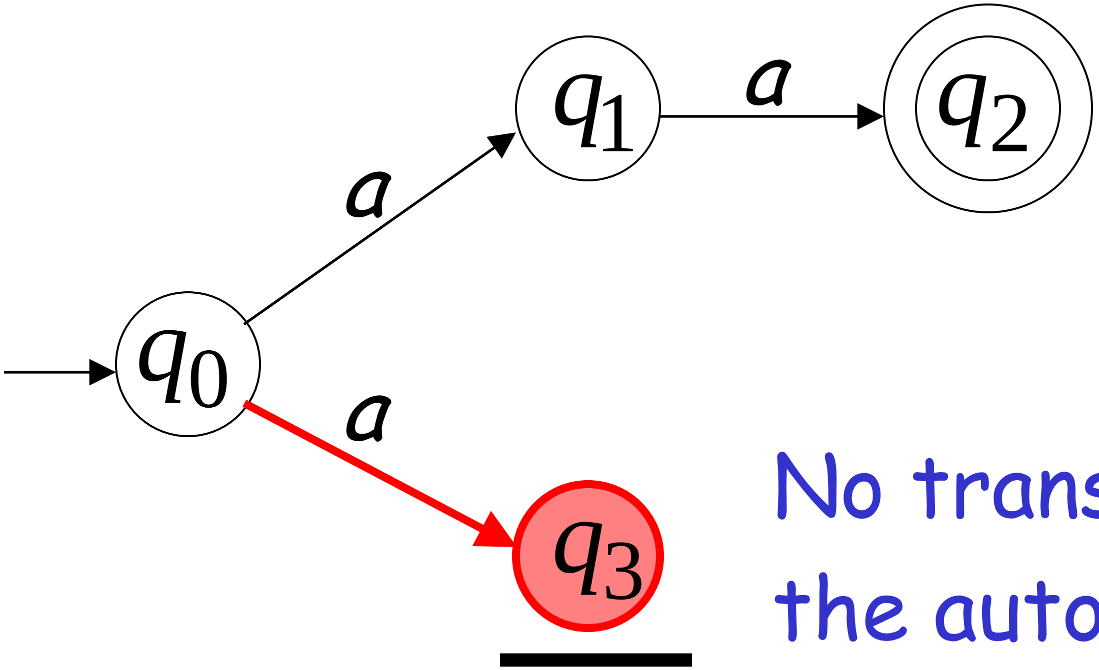


Second Choice



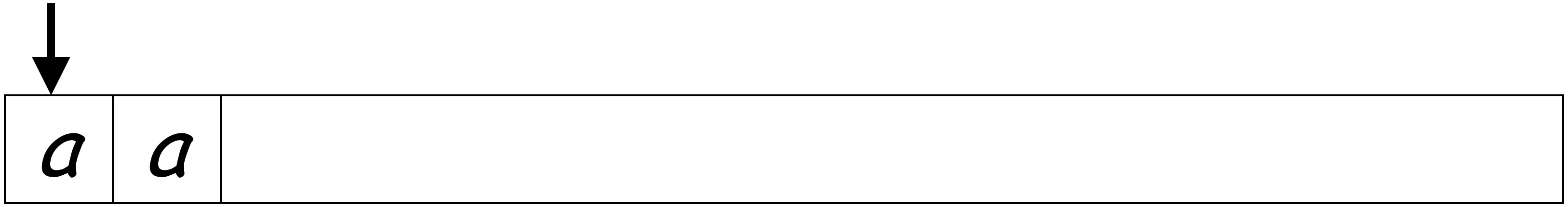
Second Choice



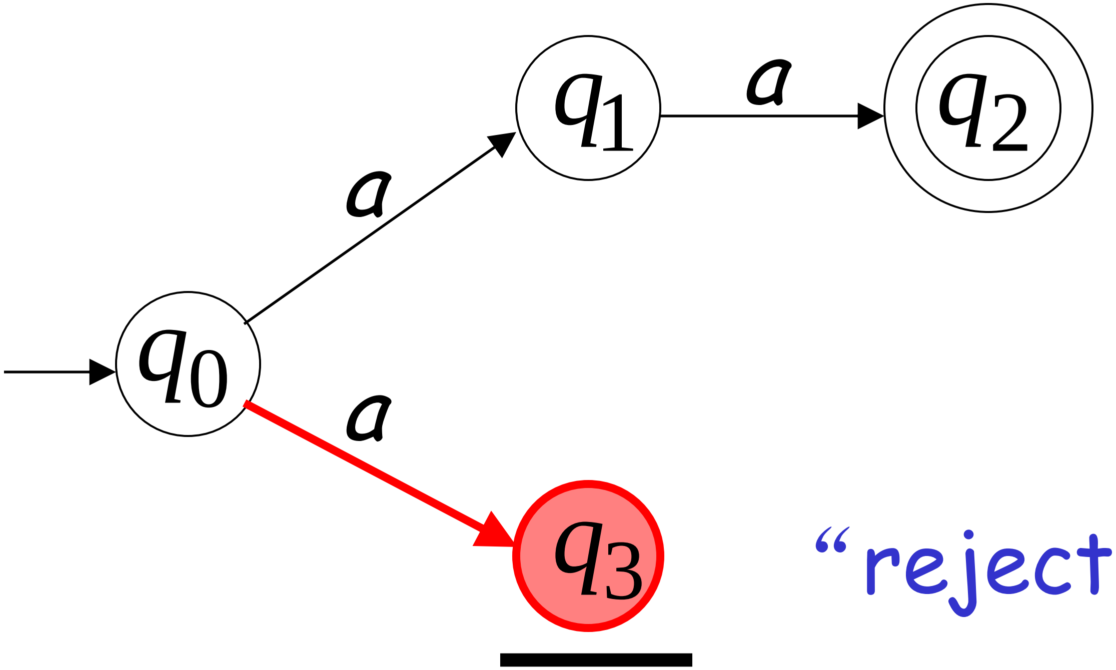


No transition:
the automaton hangs

Second Choice



Input cannot be consumed



“reject”

An NFA accepts a string:
when there is a computation of the NFA
that accepts the string

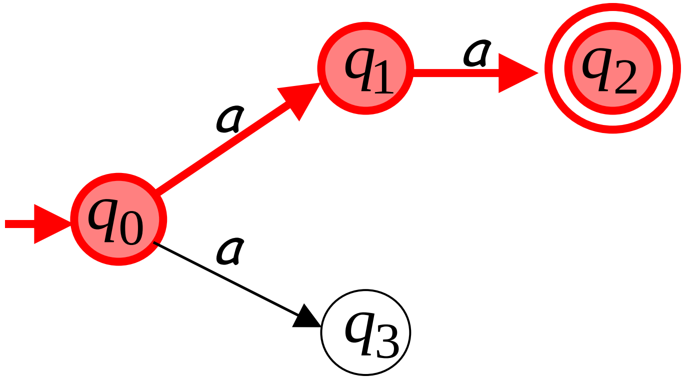
AND

all the input is consumed and the automaton
is in a final state

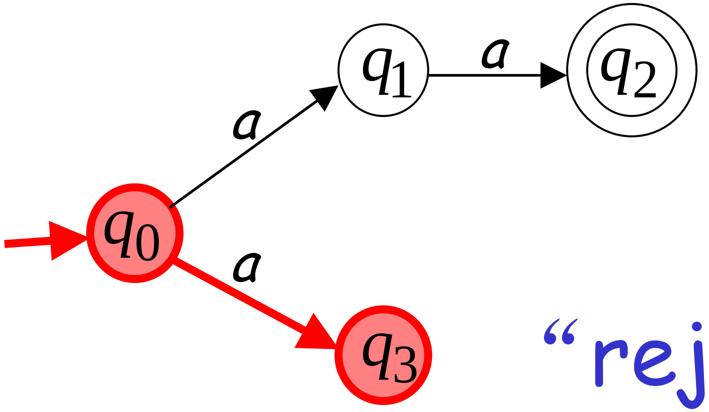
Example

aa is accepted by the NFA:

“accept”

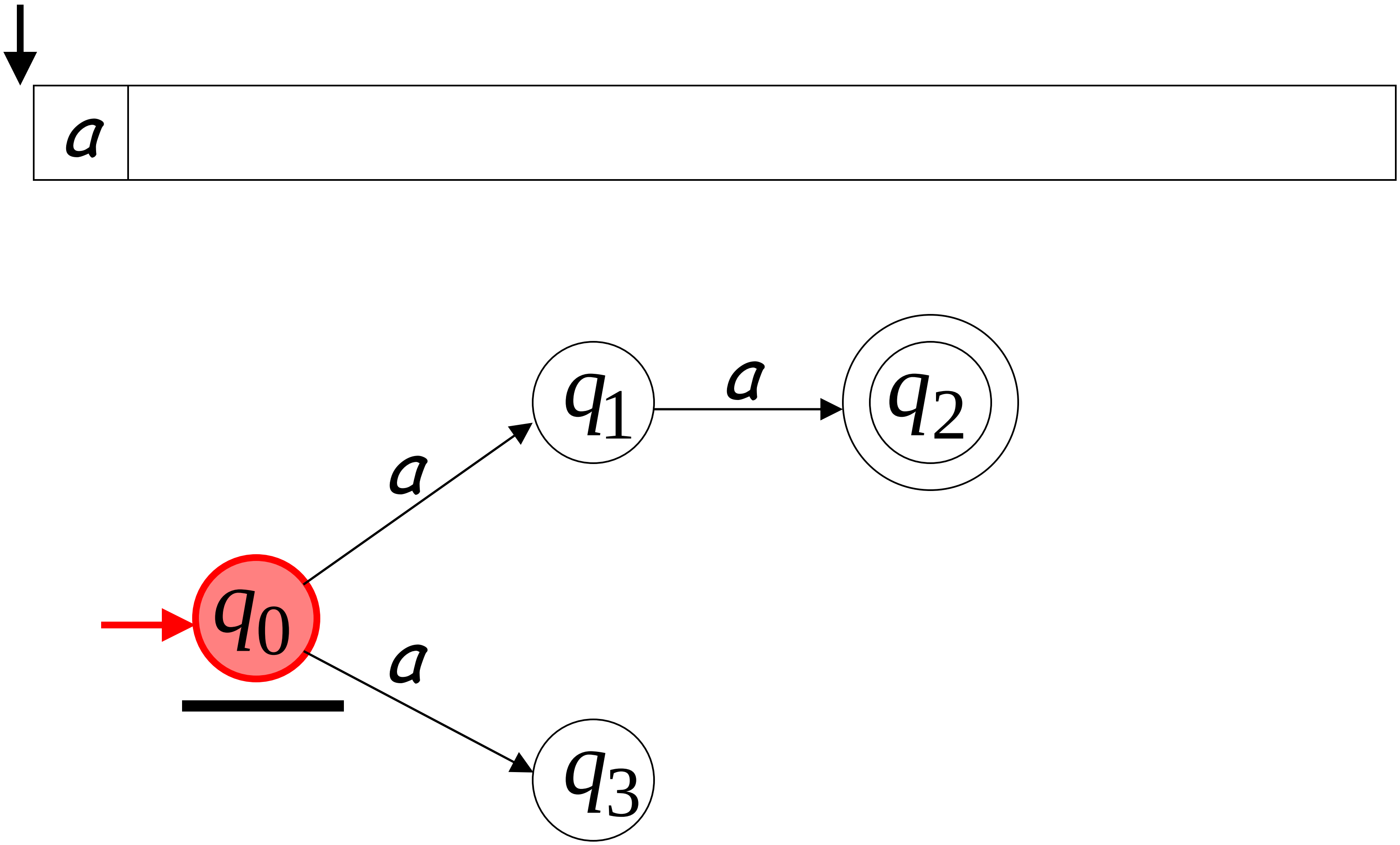


because this computation accepts aa

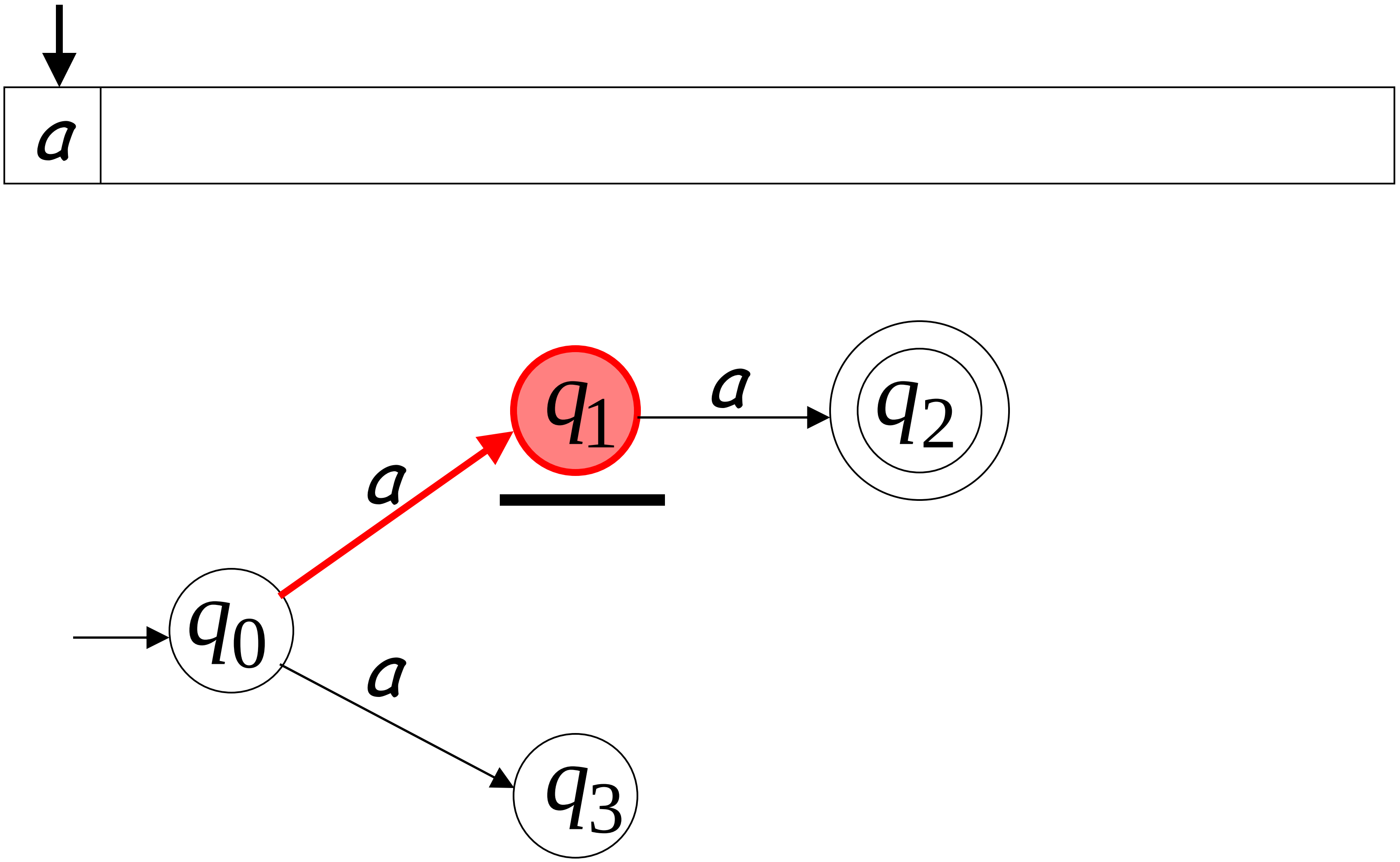


“reject

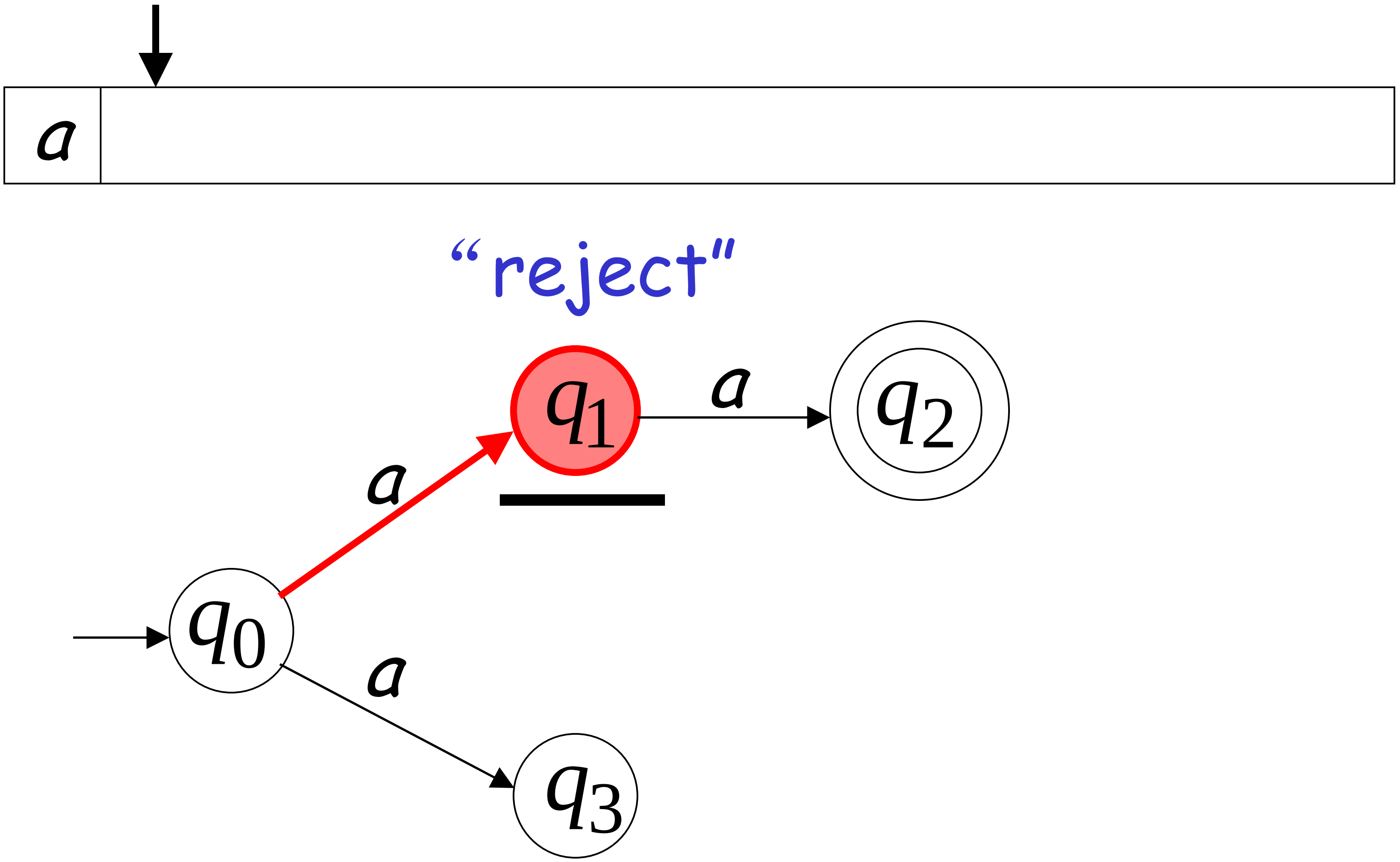
Rejection example



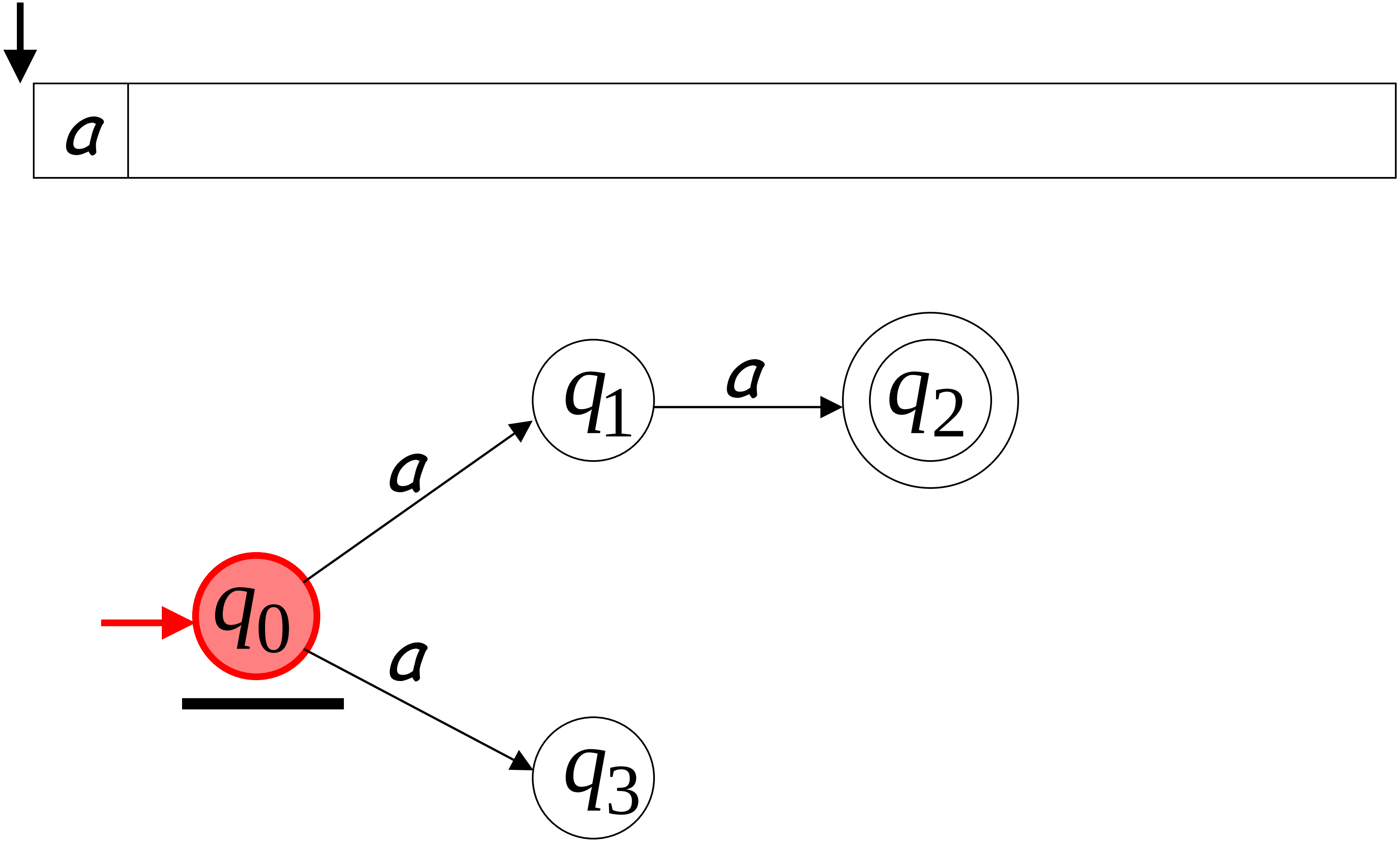
First Choice



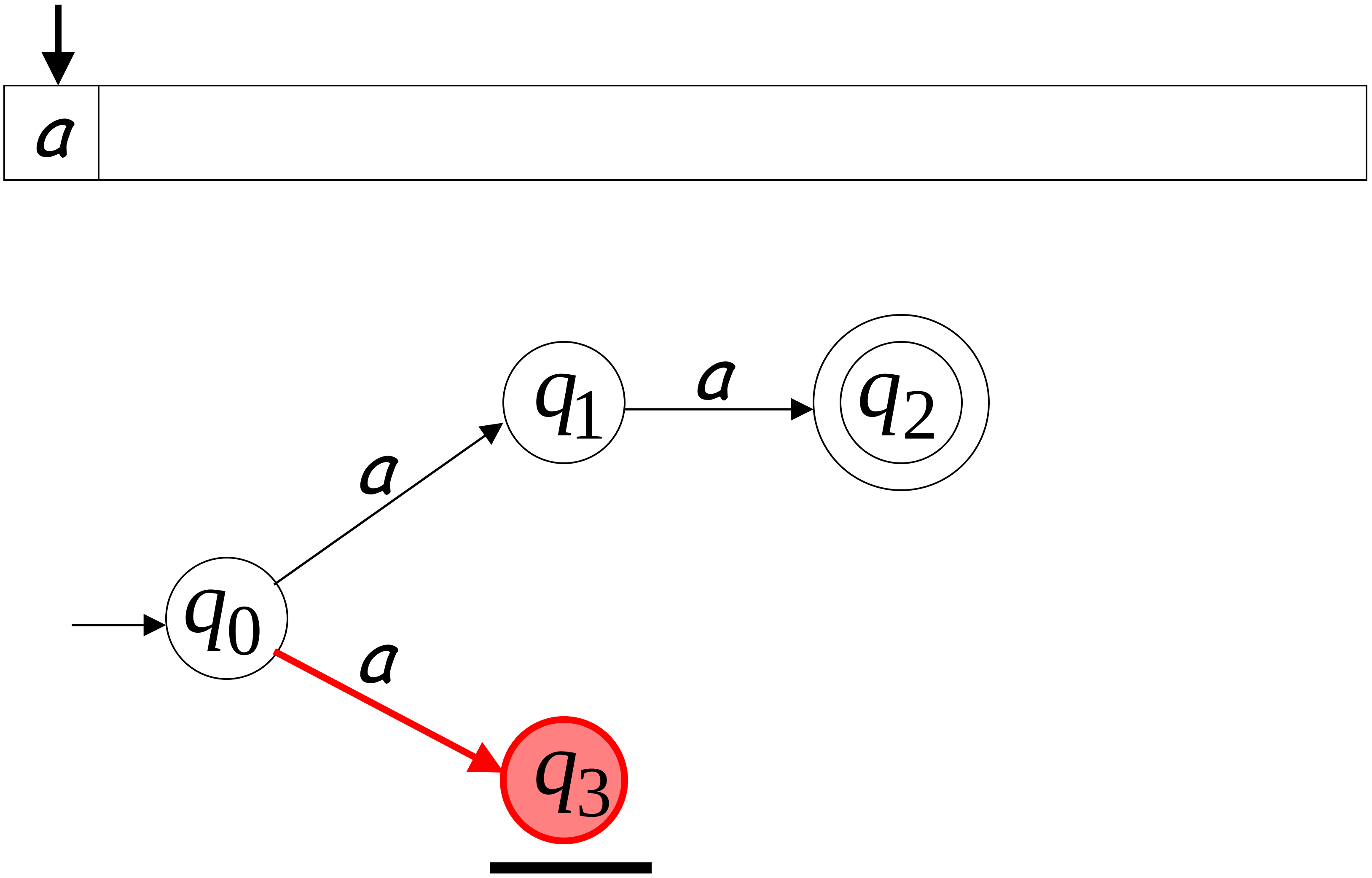
First Choice



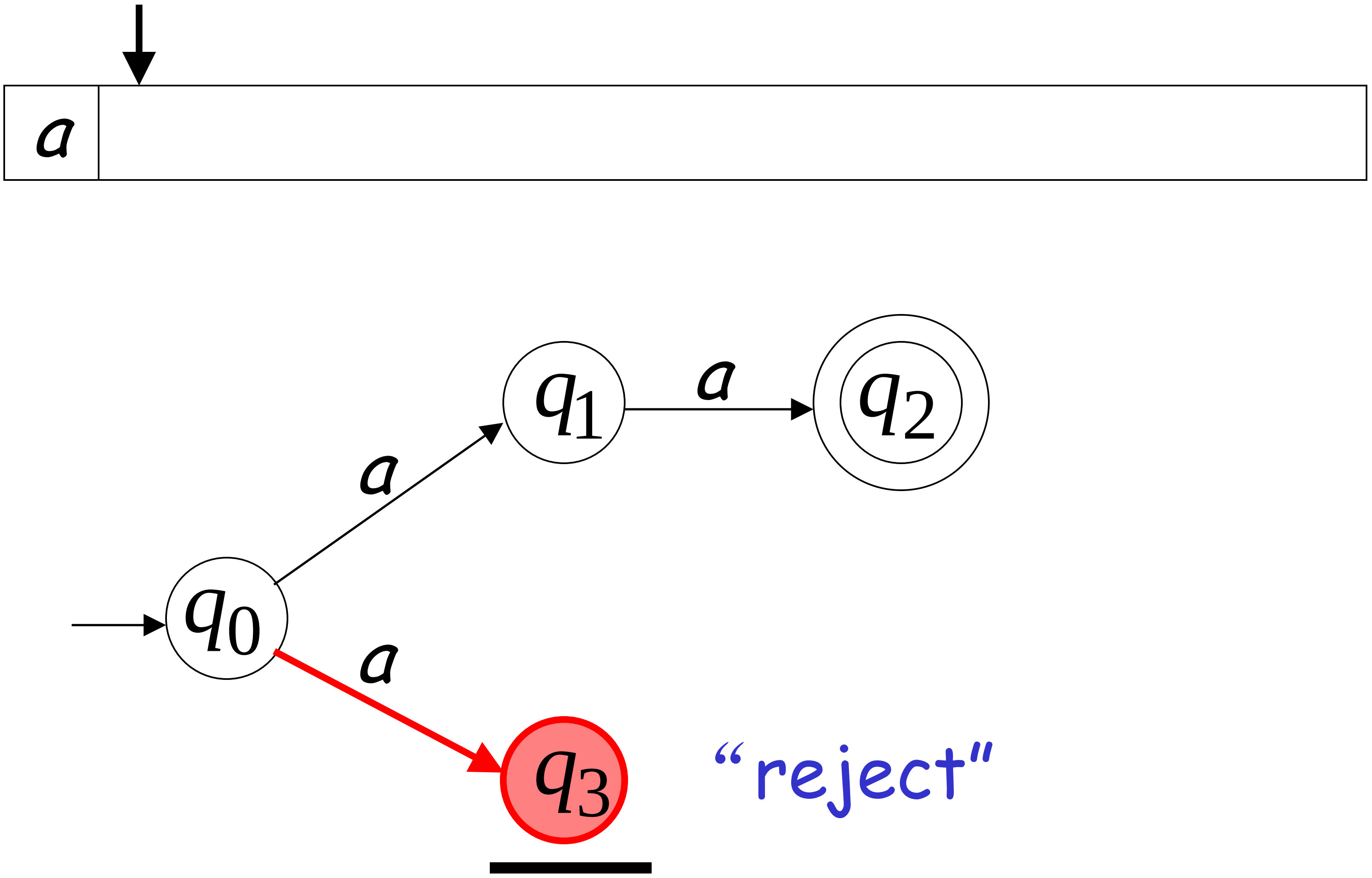
Second Choice



Second Choice



Second Choice



"reject"

An NFA rejects a string:

when there is no computation of the NFA that accepts the string:

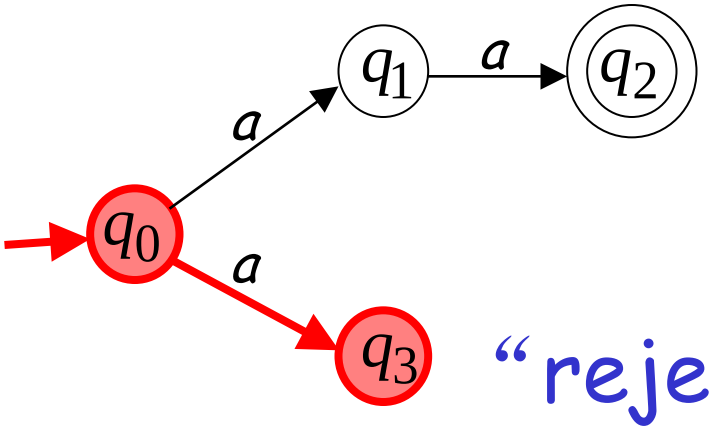
- All the input is consumed and the automaton is in a non final state

OR

- The input cannot be consumed

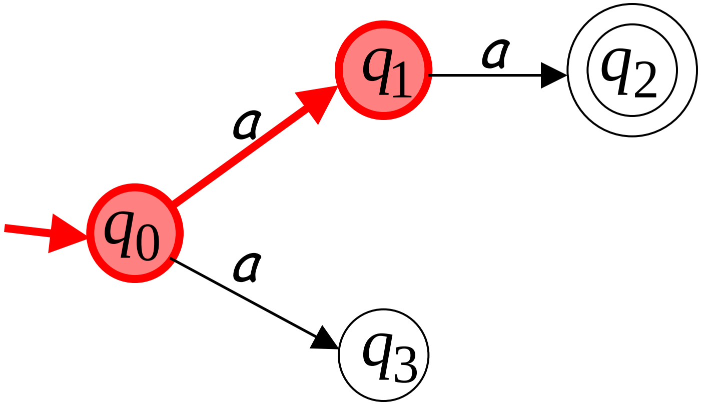
Example

a is rejected by the NFA:



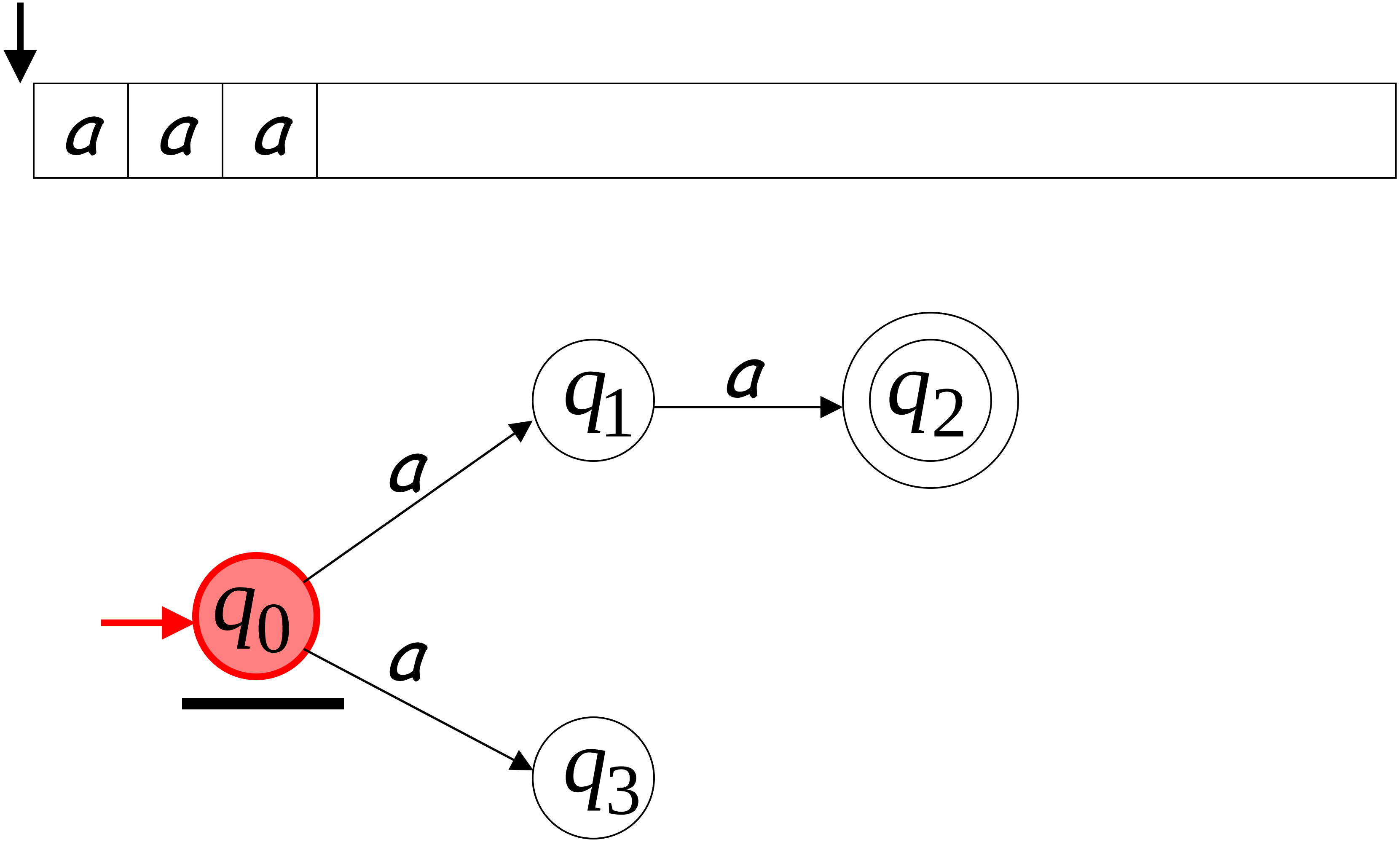
"reject"

"reject"

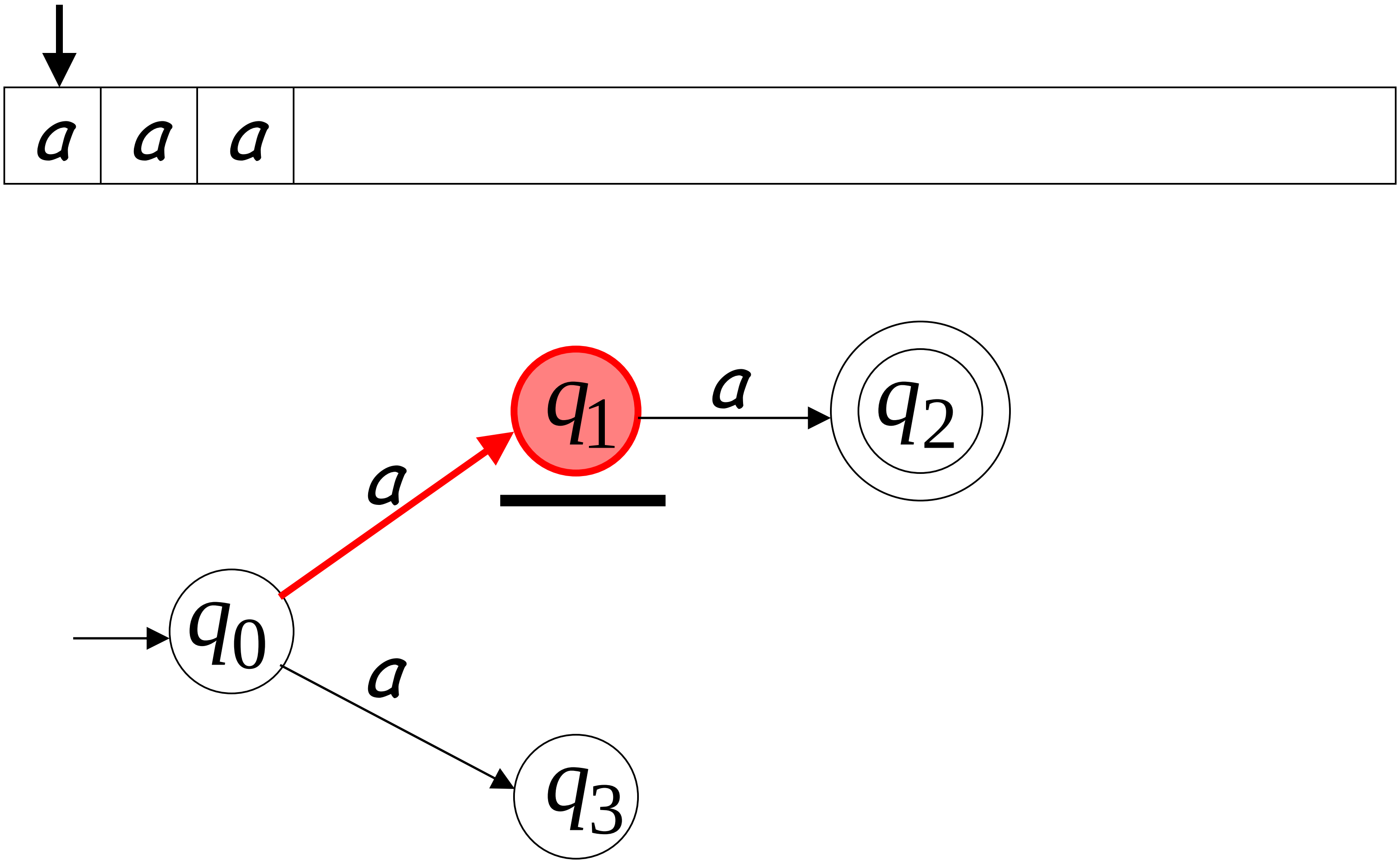


All possible computations lead to rejection

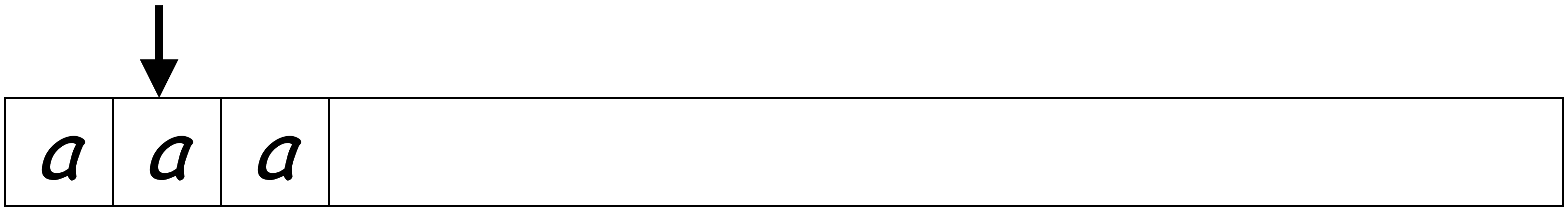
Rejection example

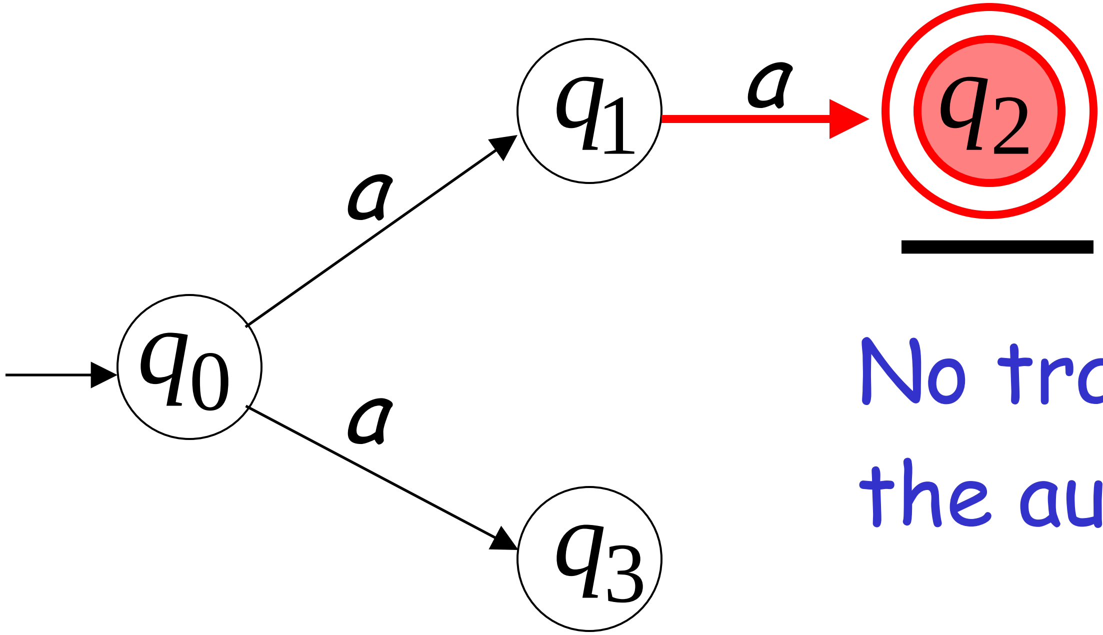


First Choice



First Choice

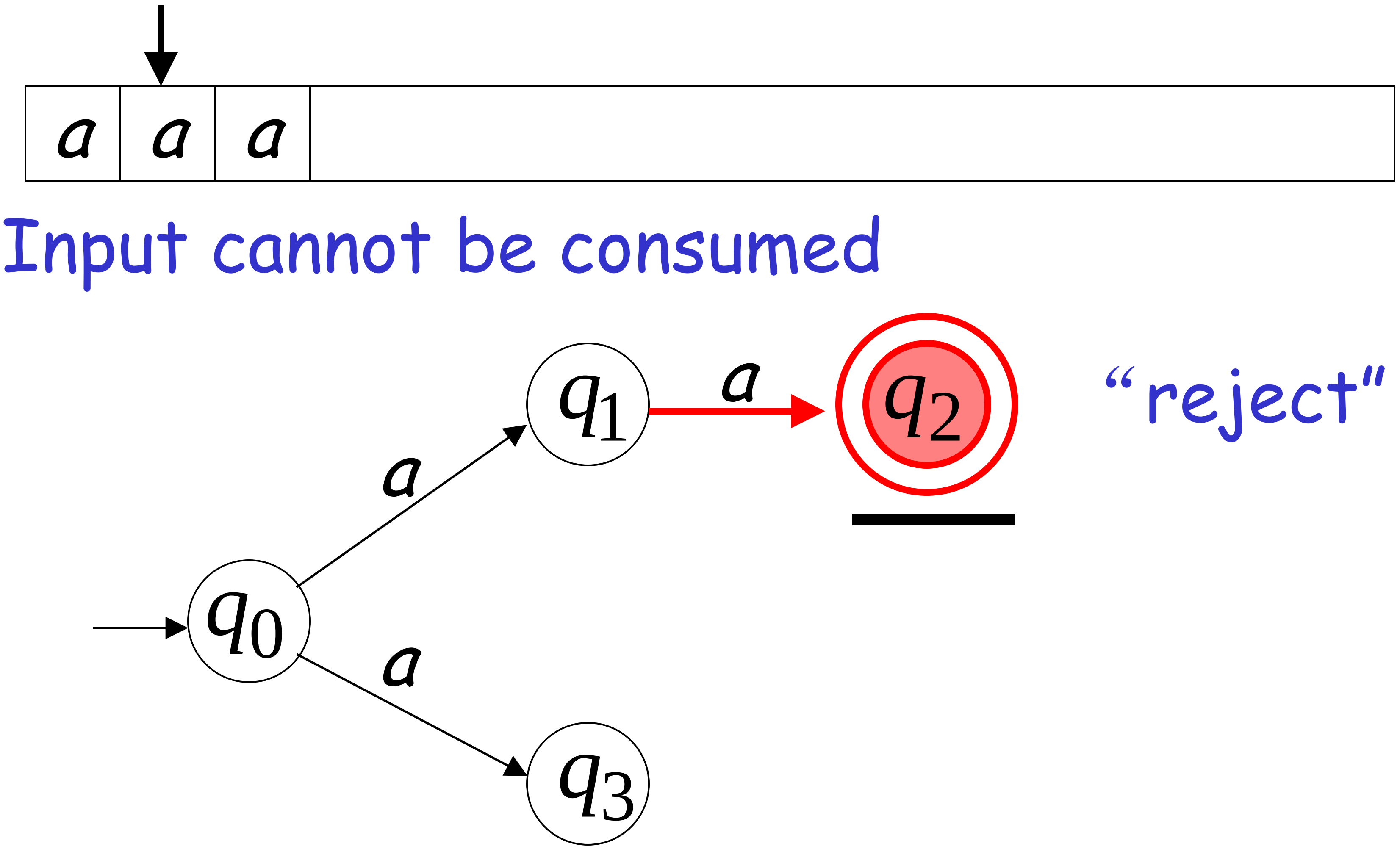




No transition:

the automaton hangs

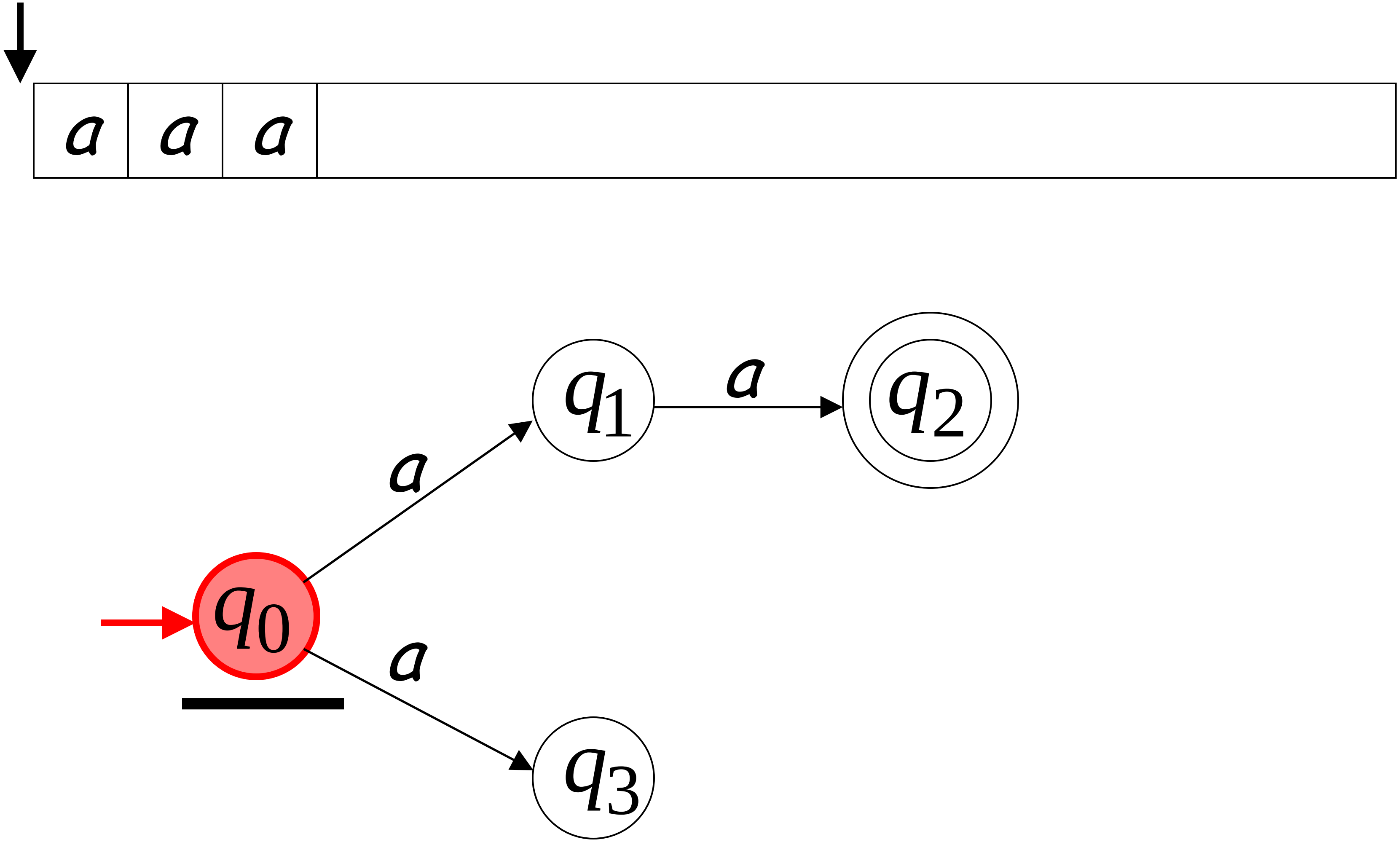
First Choice



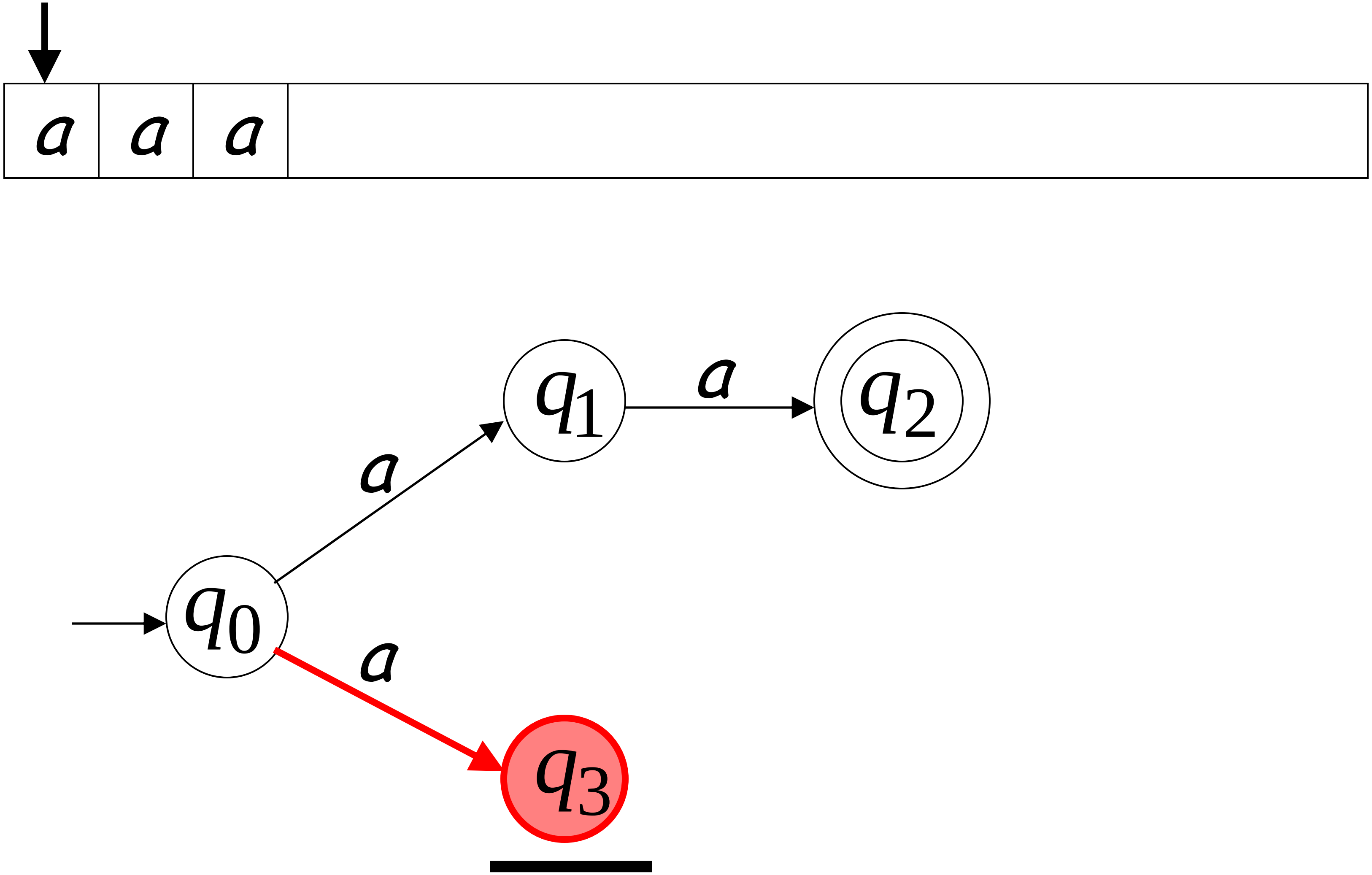
Input cannot be consumed

“reject”

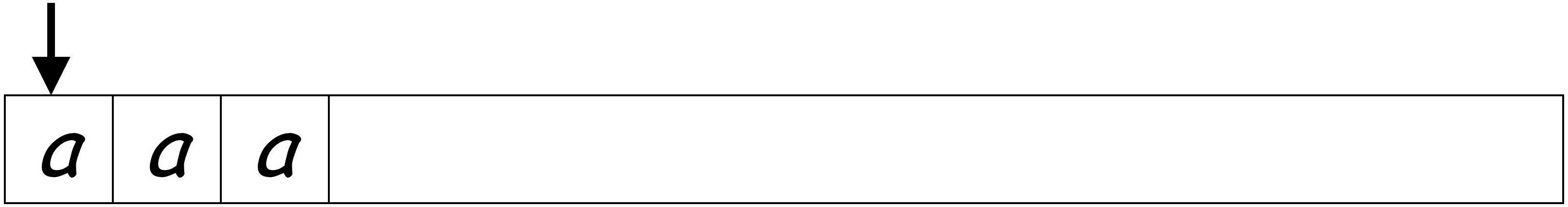
Second Choice

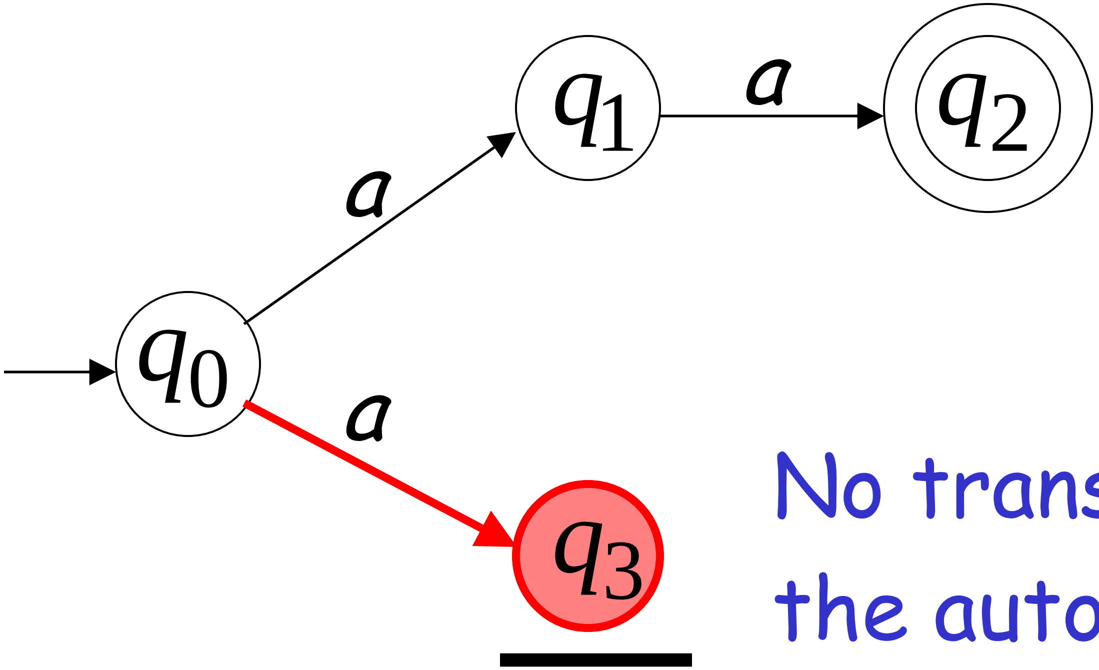


Second Choice



Second Choice

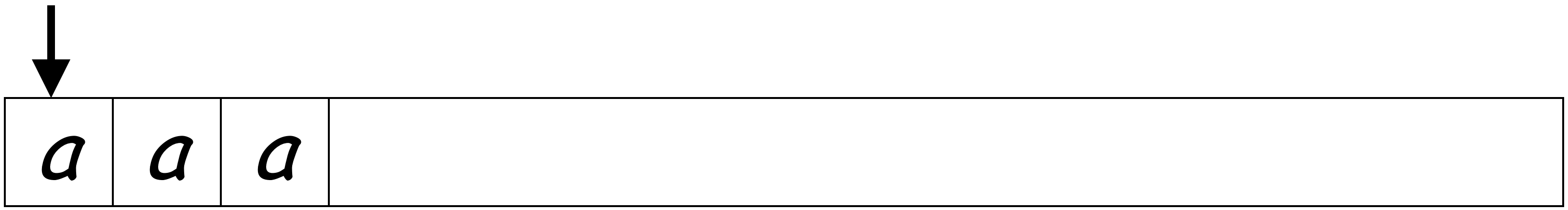




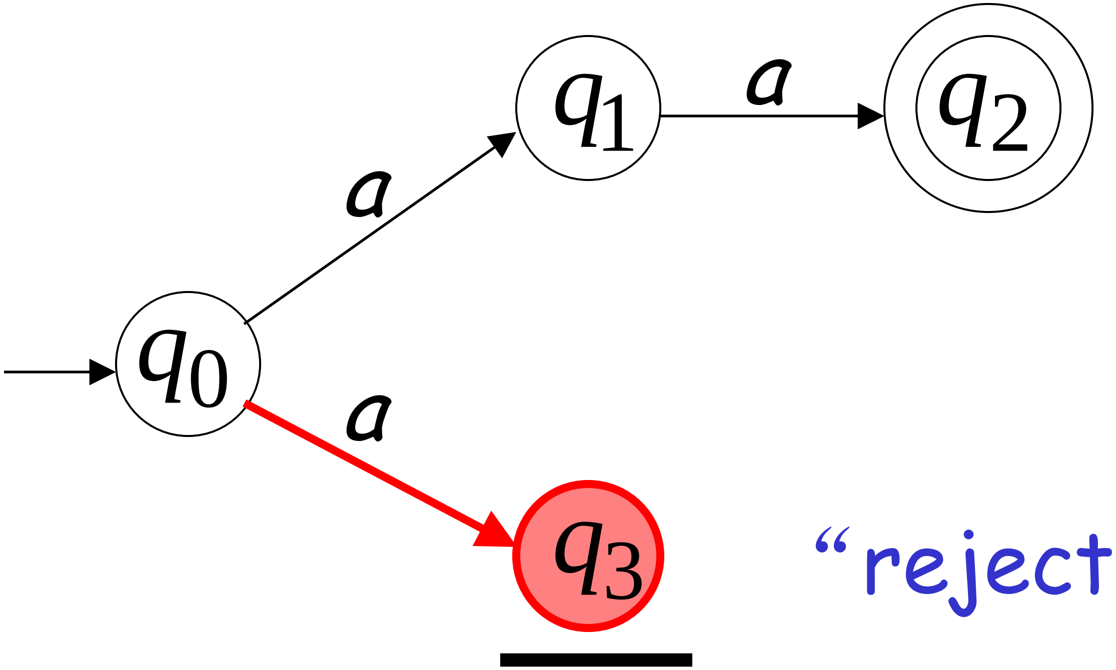
No transition:

the automaton hangs

Second Choice

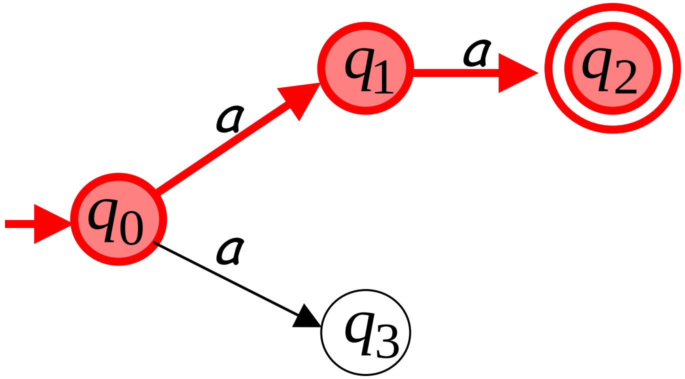


Input cannot be consumed

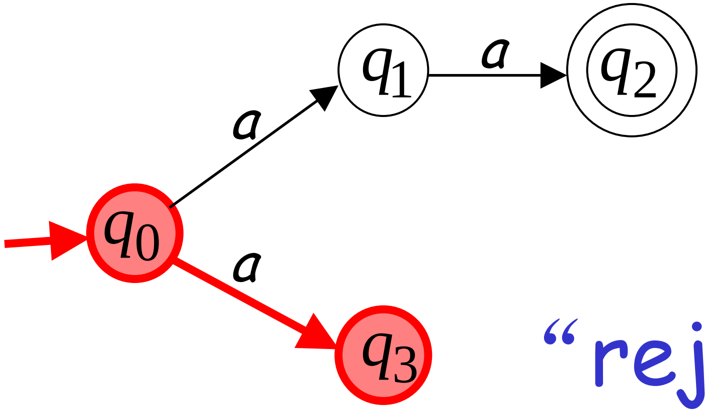


"reject"

aaa is rejected by the NFA:



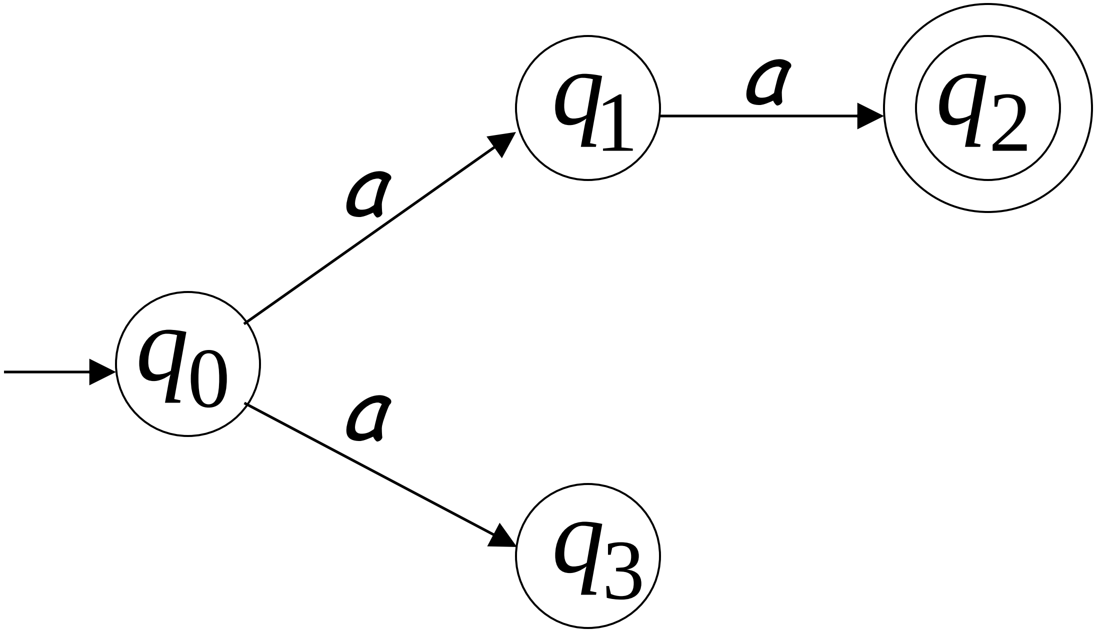
“reject”



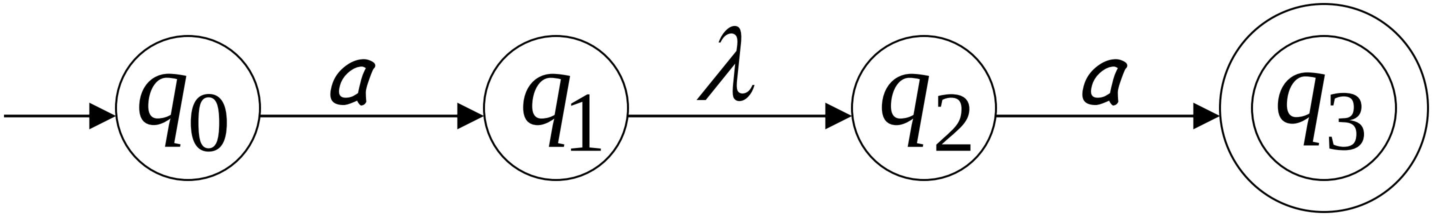
“reject

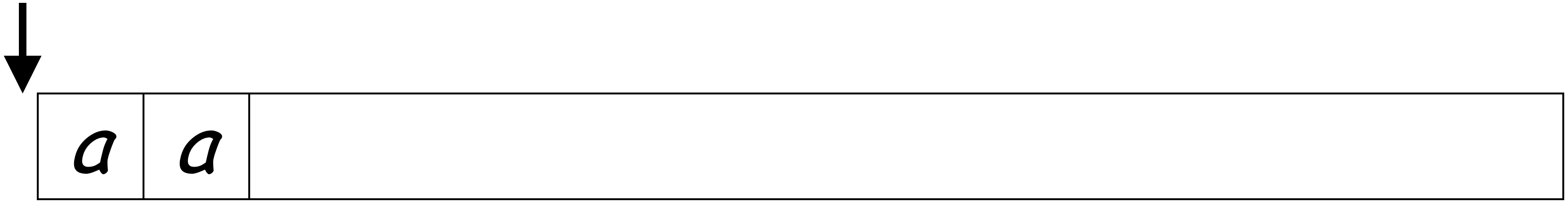
All possible computations lead to rejection

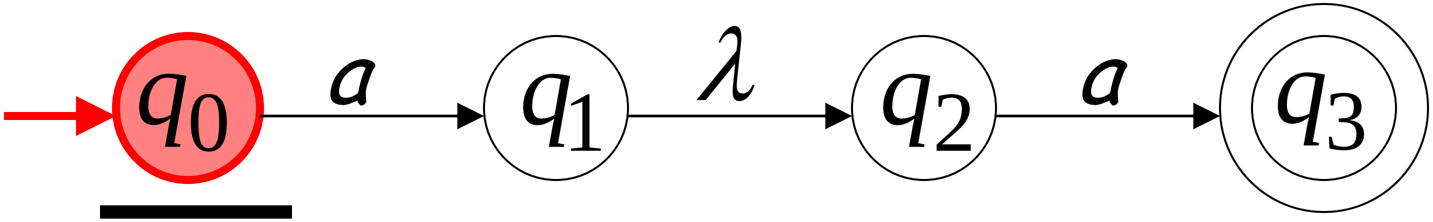
Language accepted: $L=\{aa\}$

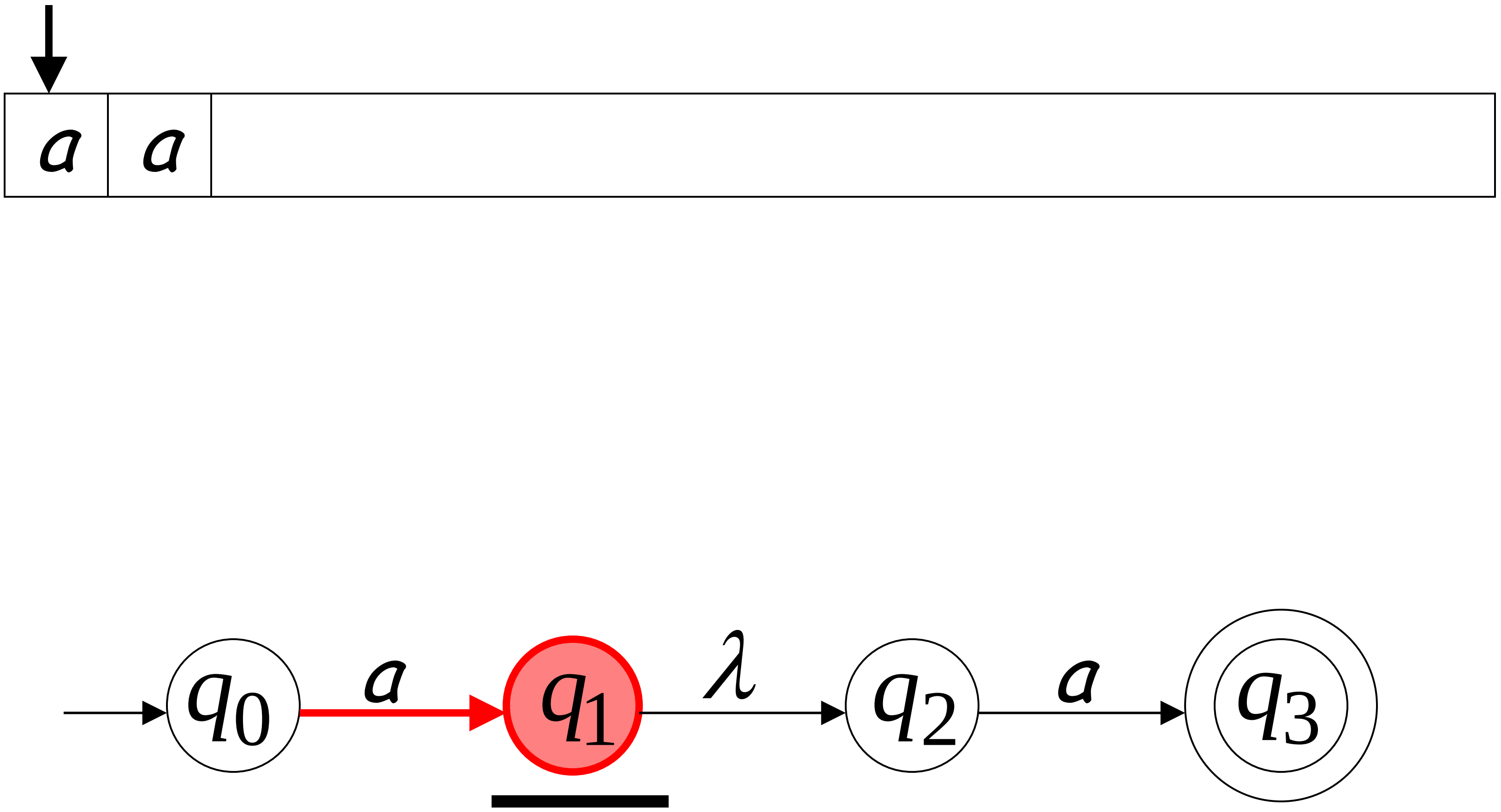


Lambda Transitions

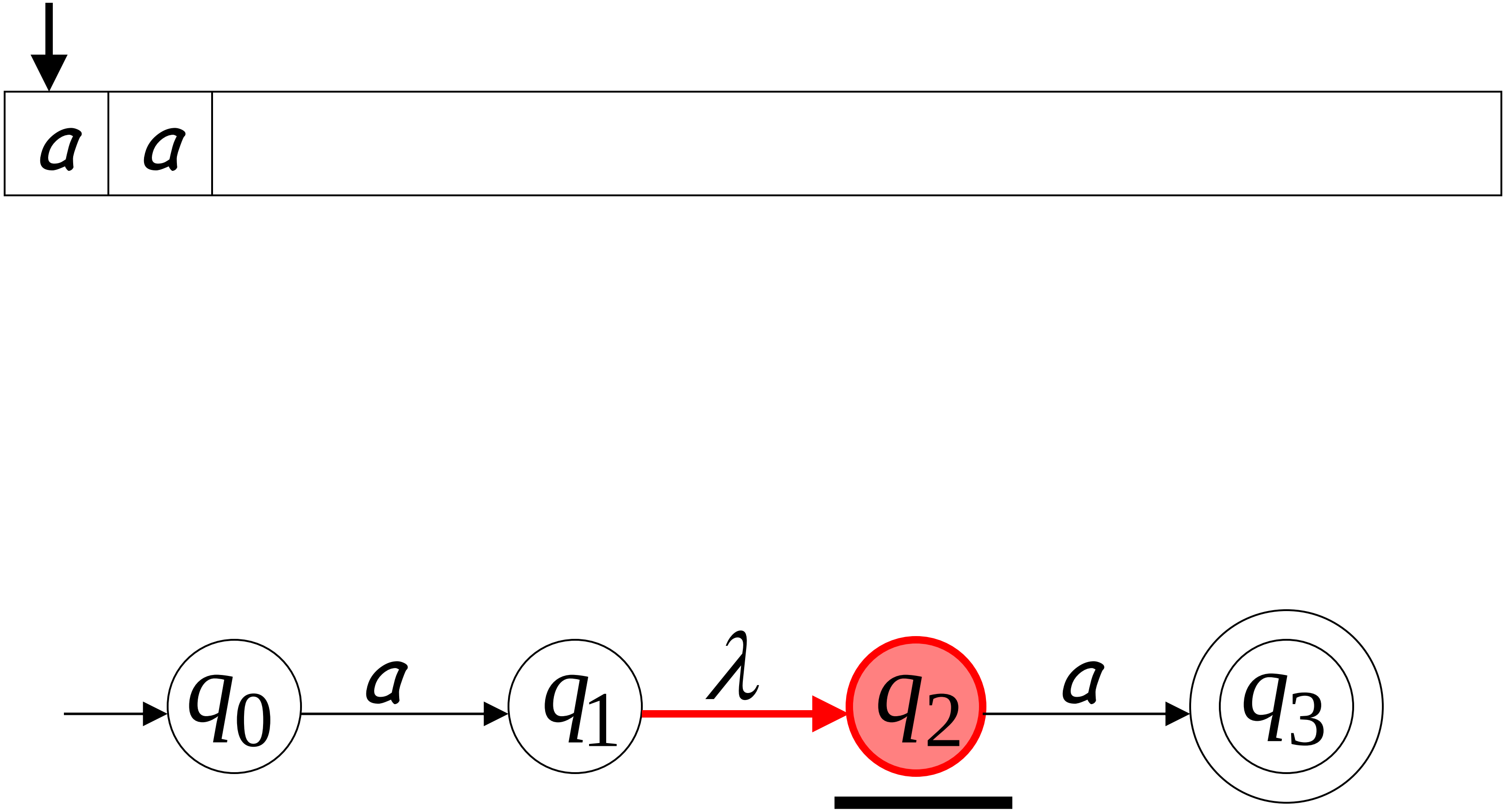


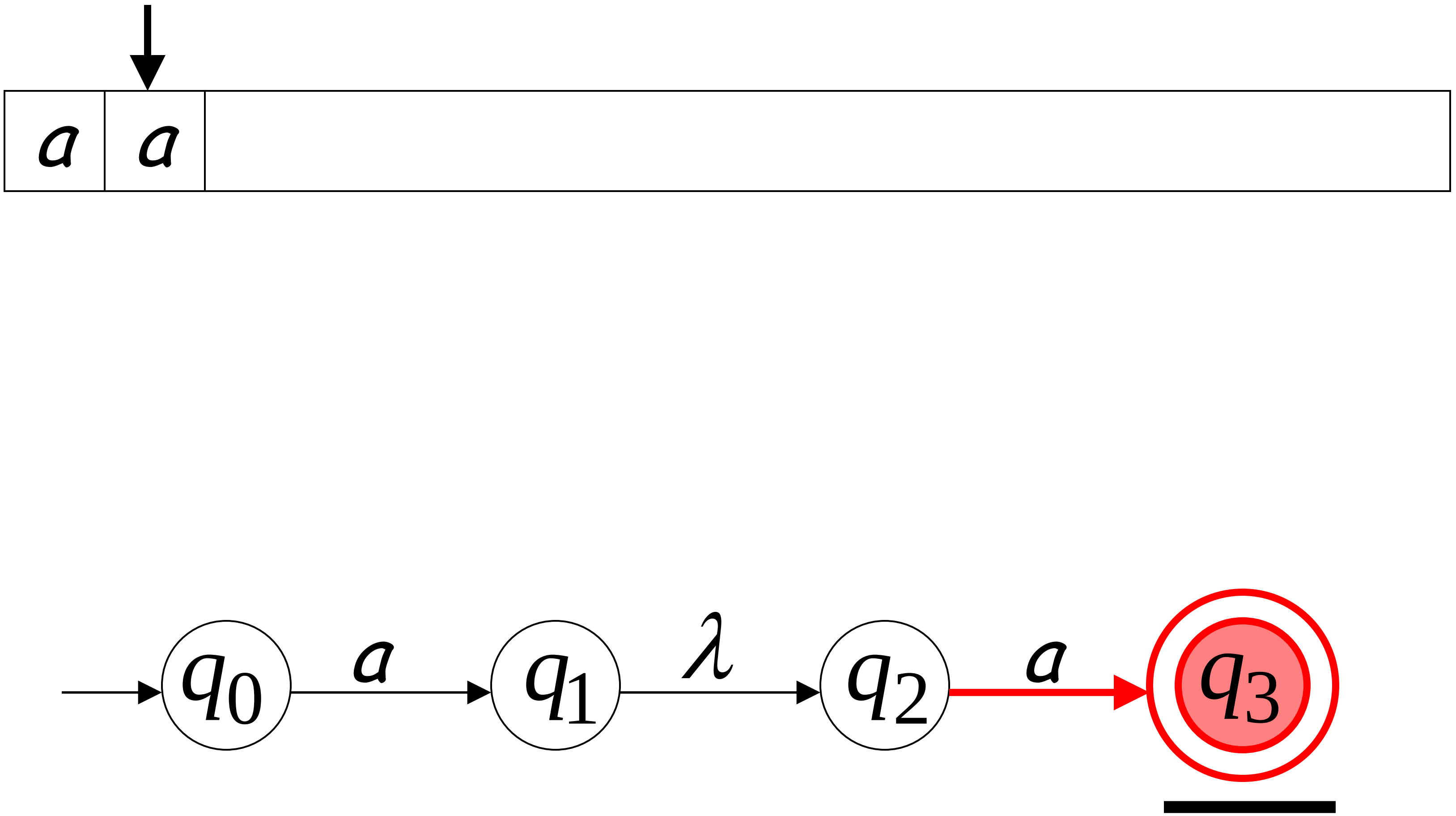

a
a


q_0
a
q_1
λ
q_2
a
q_3

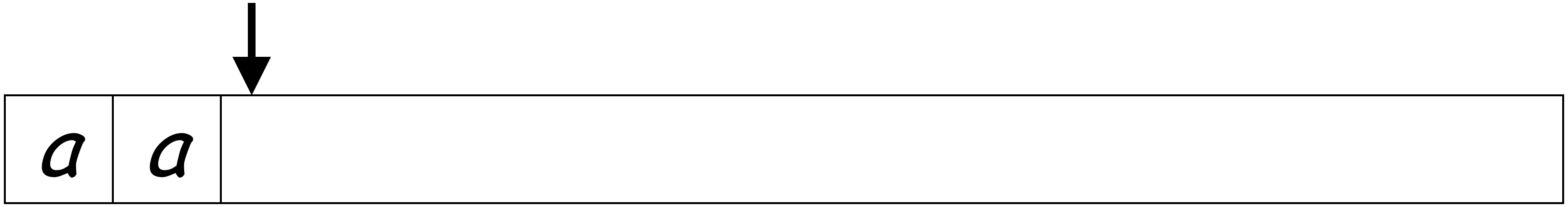

a
a
q_0
a
q_1
λ
q_2
a
q_3

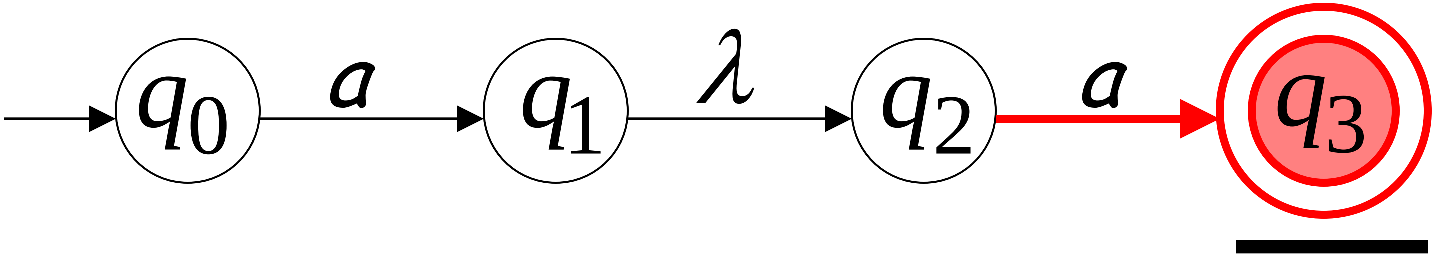
(read head does not move)




a
a
q0
a
q1
λ
q2
a
q3

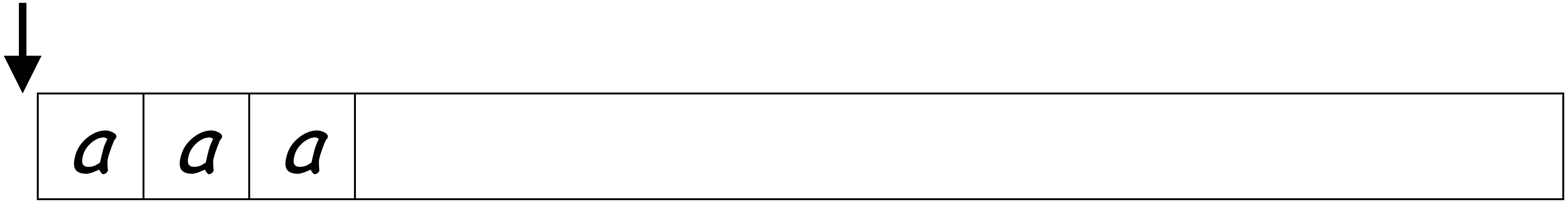
all input is consumed

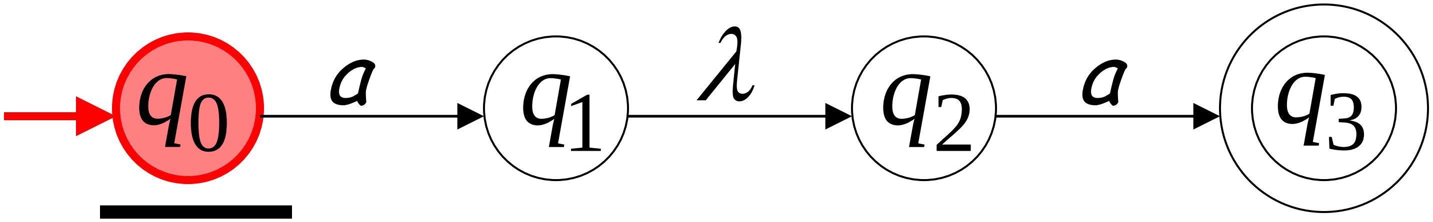


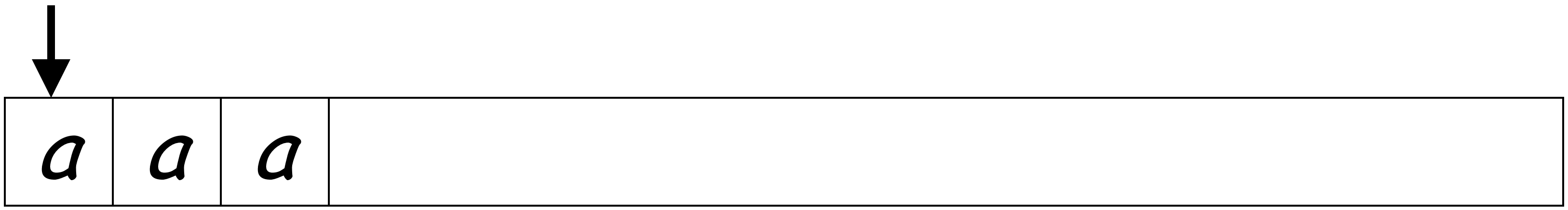

“accept”

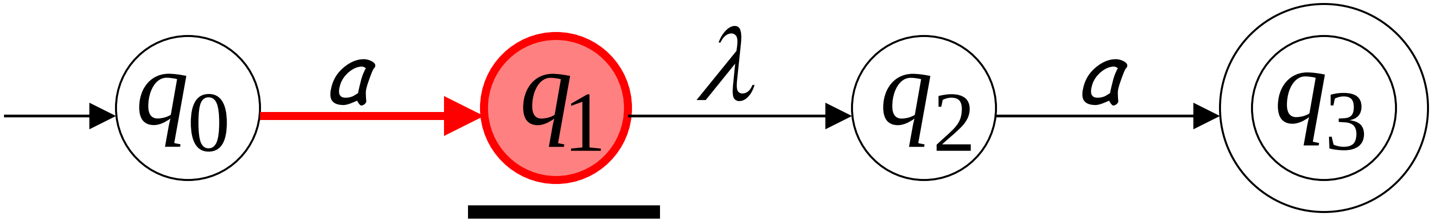
String aa is accepted

Rejection Example

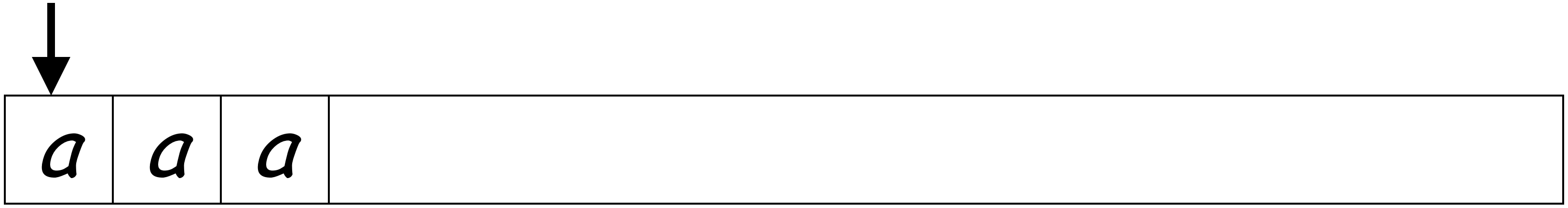


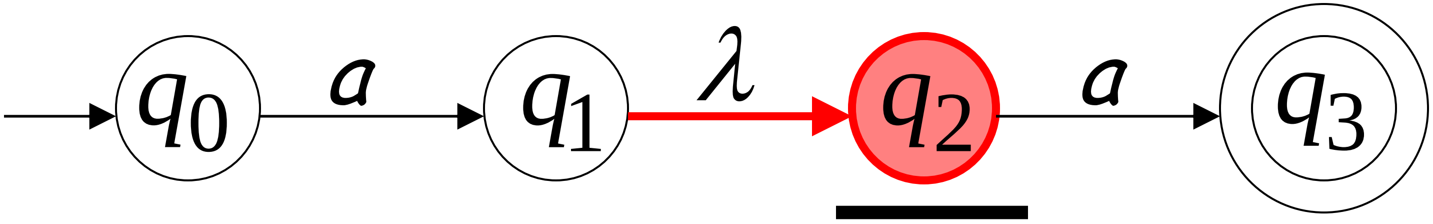


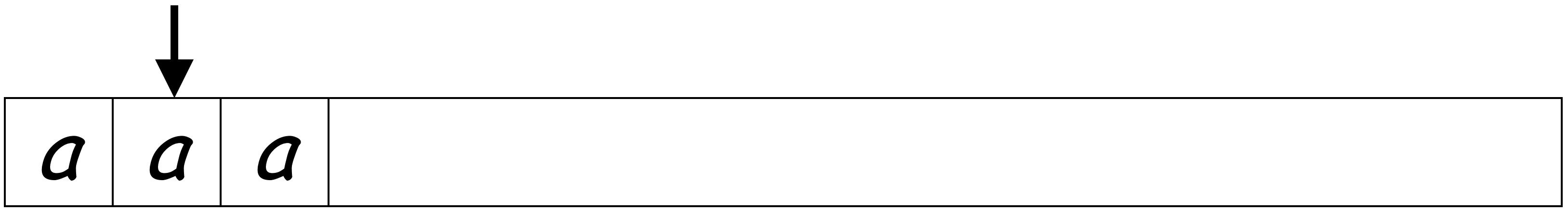

a
a
a

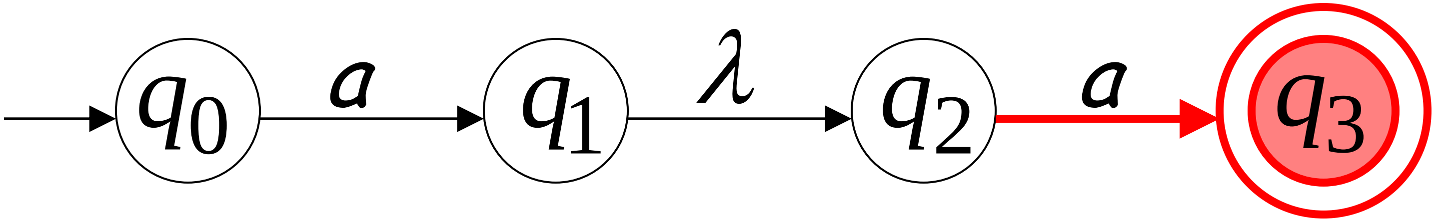

q_0
a
q_1
λ
q_2
a
q_3

(read head doesn't move)





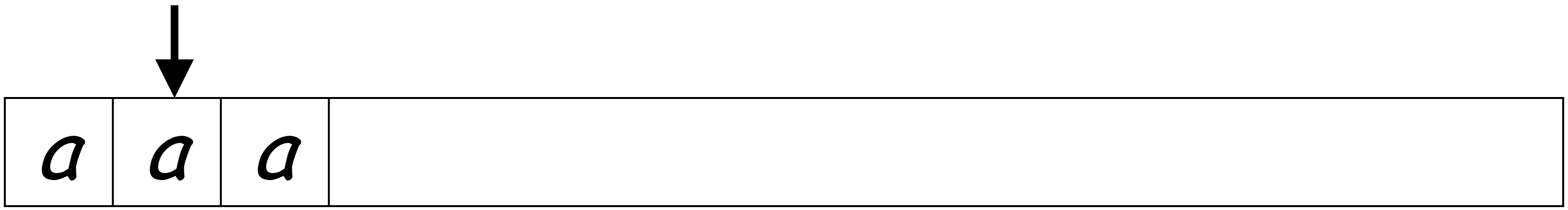




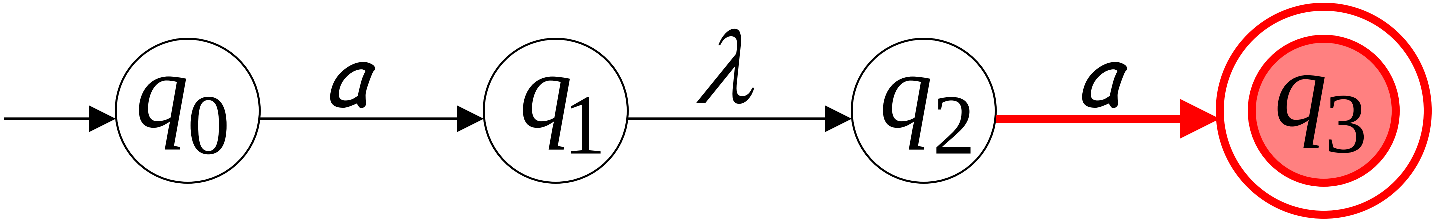
No transition:

the automaton hangs

Input cannot be consumed

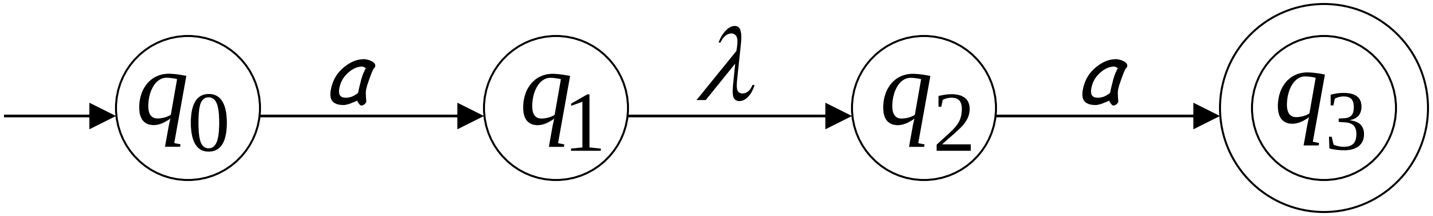


“reject”

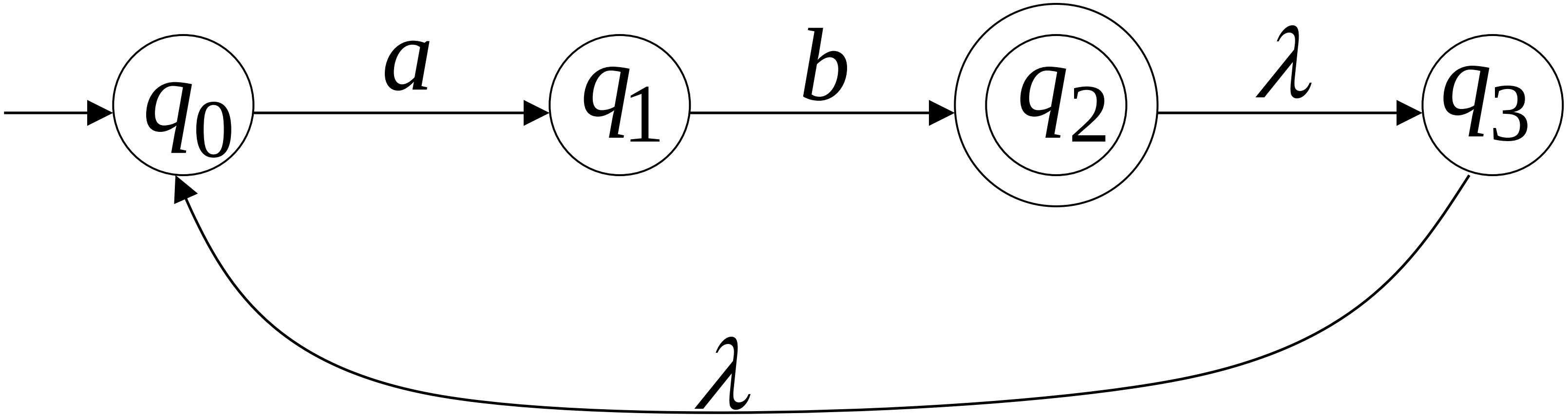


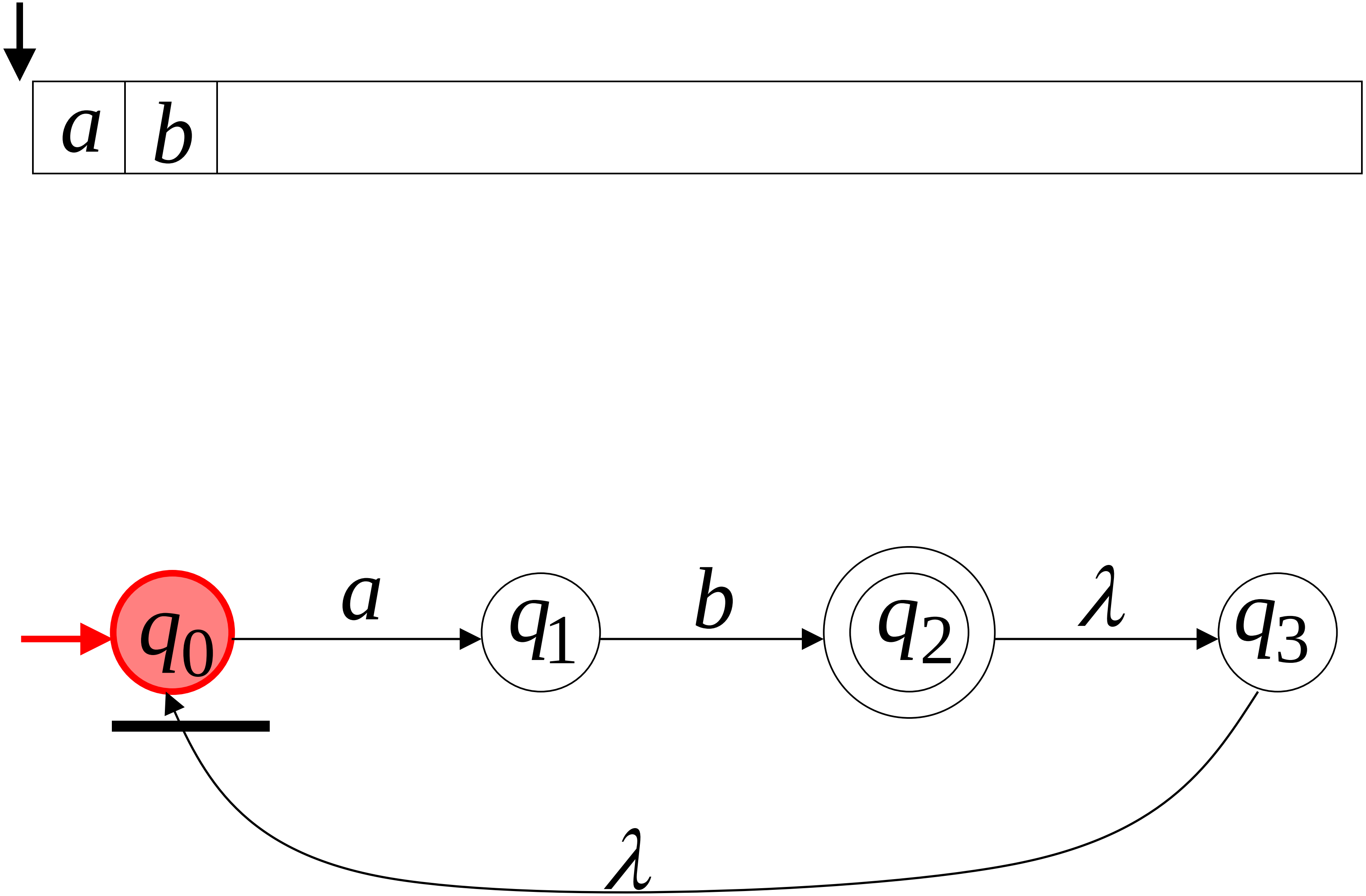
String **aaa** is rejected

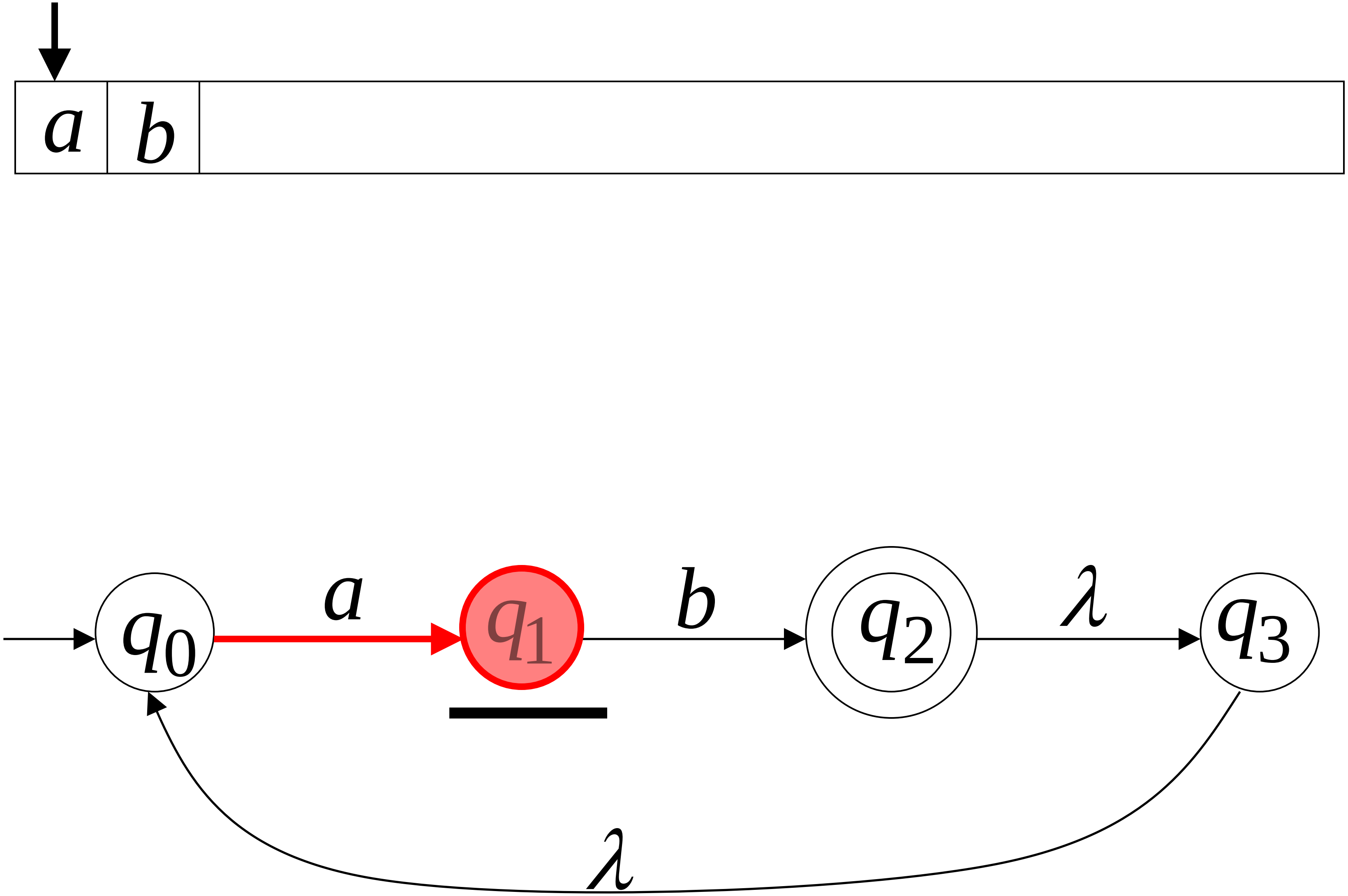
Language accepted: $L = \{aa\}$

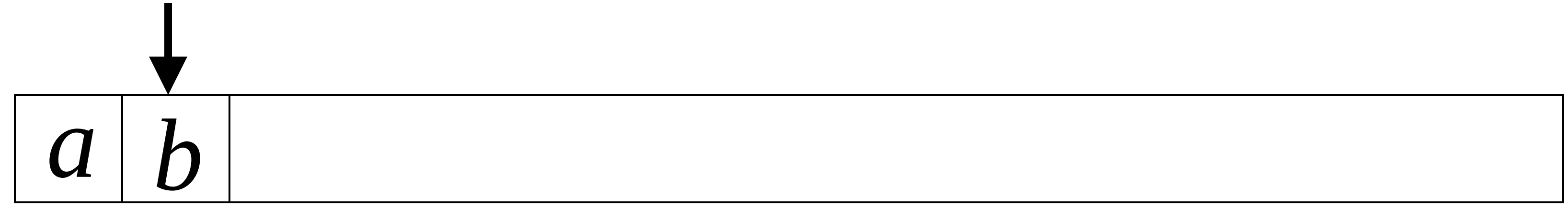


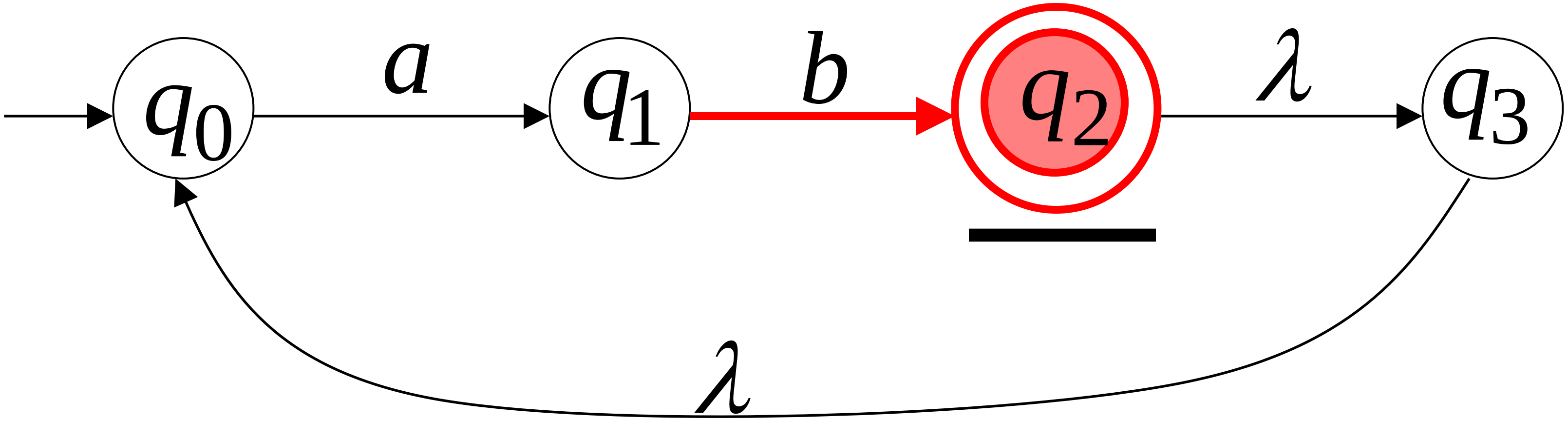
Another NFA Example

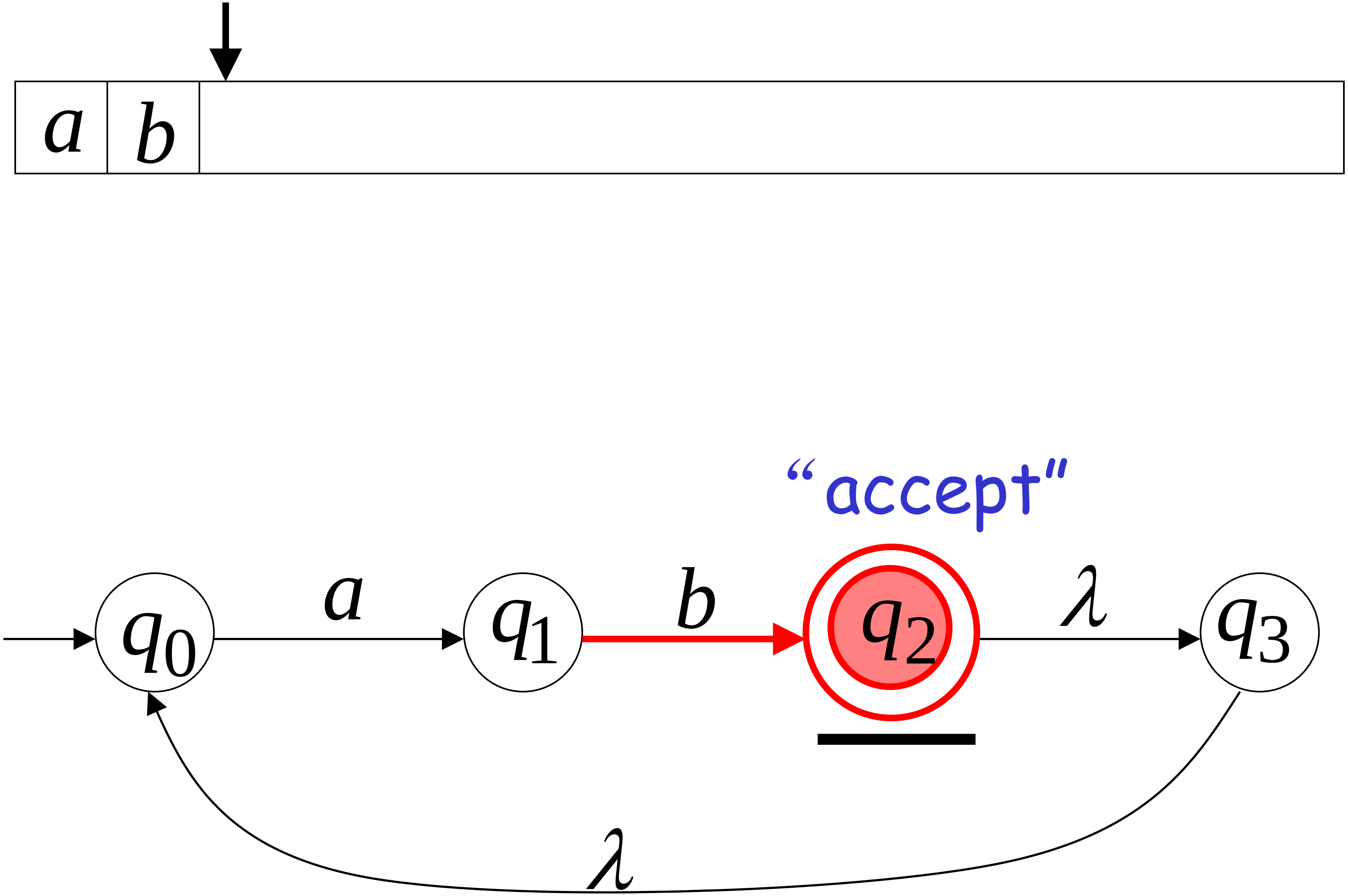



a
b
q_0
a
q_1
b
q_2
λ
q_3
λ

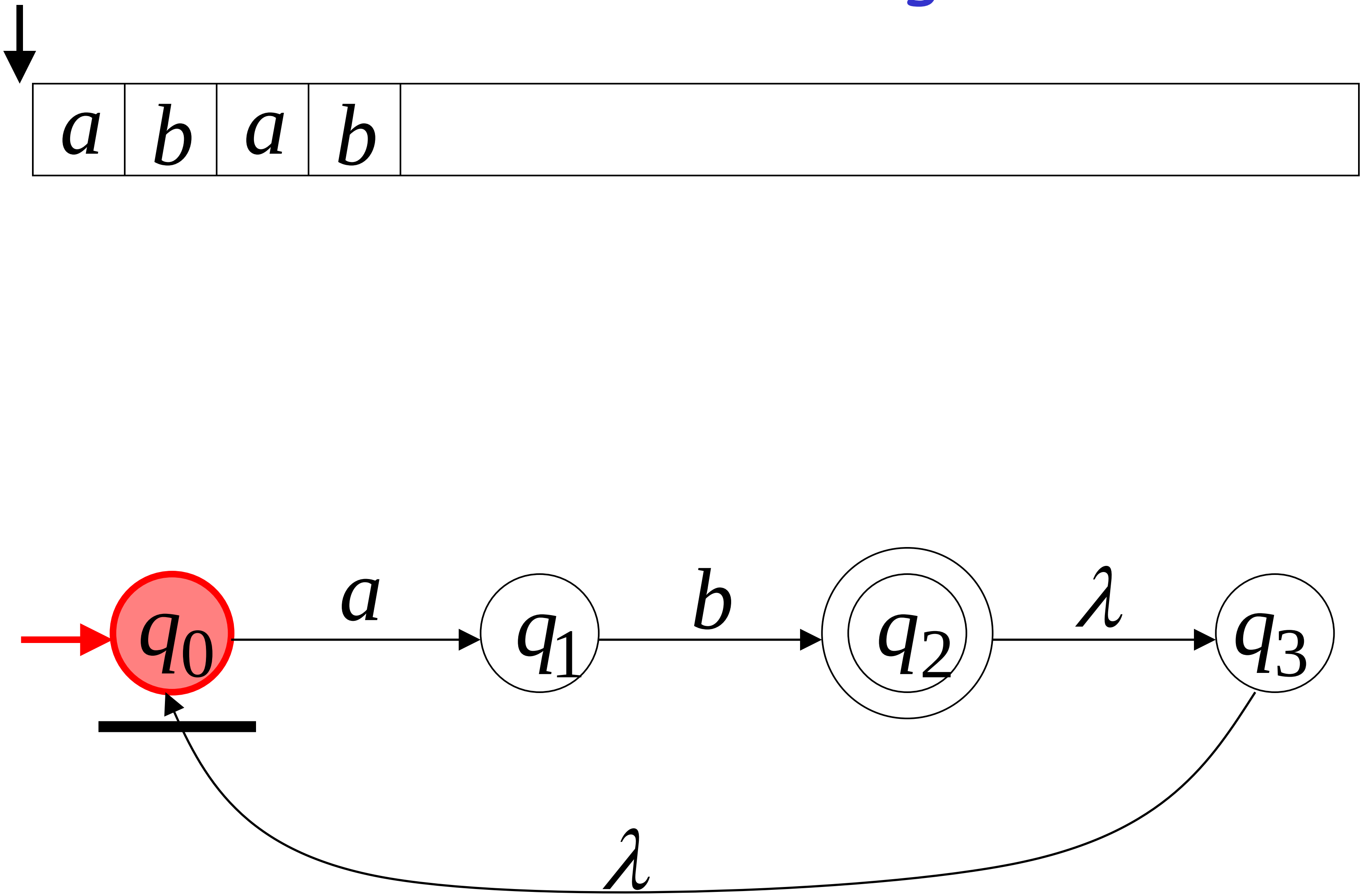

a
b
q_0
a
q_1
b
q_2
λ
q_3
λ

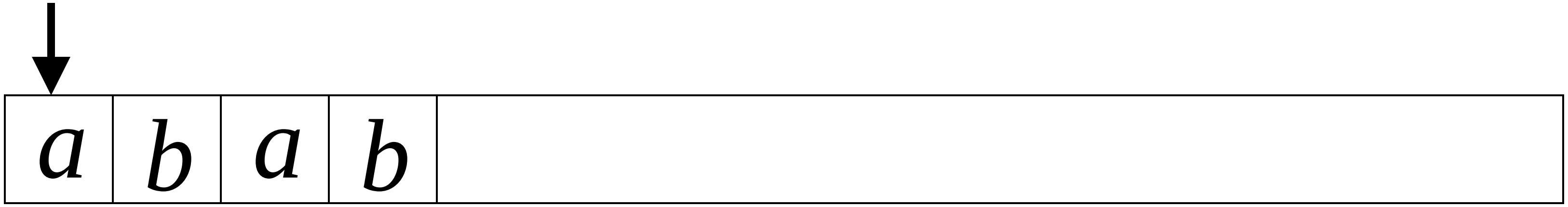

a
b

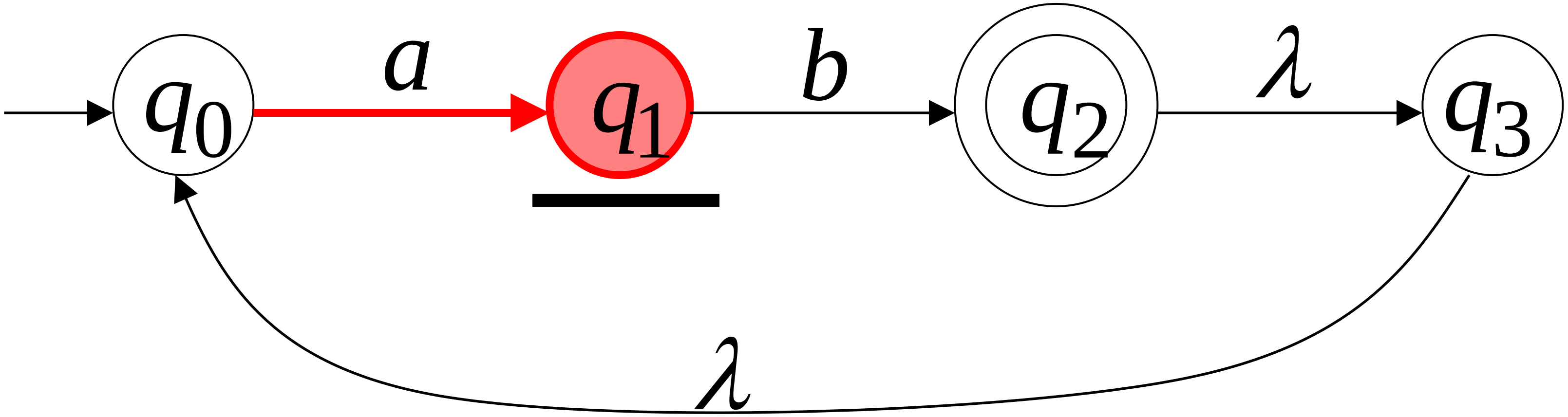

q_0
a
q_1
b
q_2
λ
q_3
λ

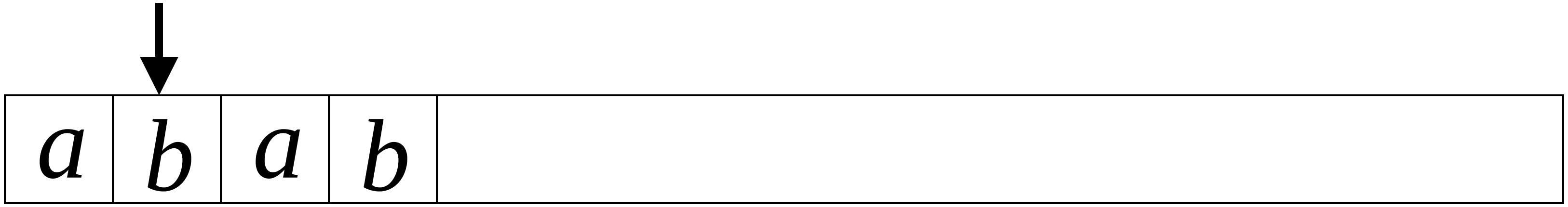

a
b
“accept”
q_0
a
q_1
b
q_2
λ
q_3
λ

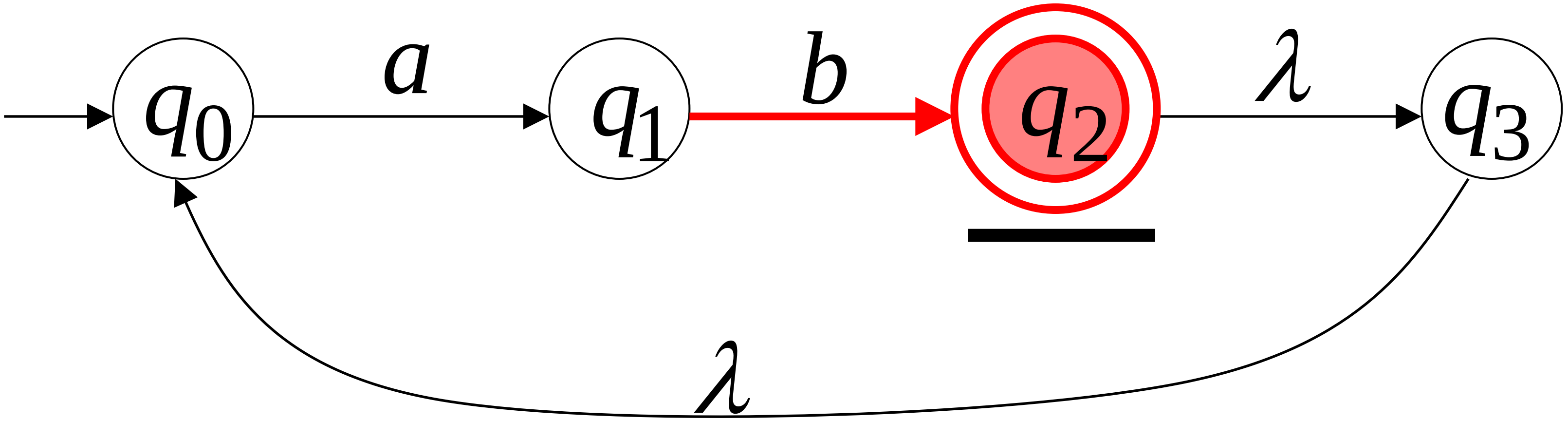
Another String

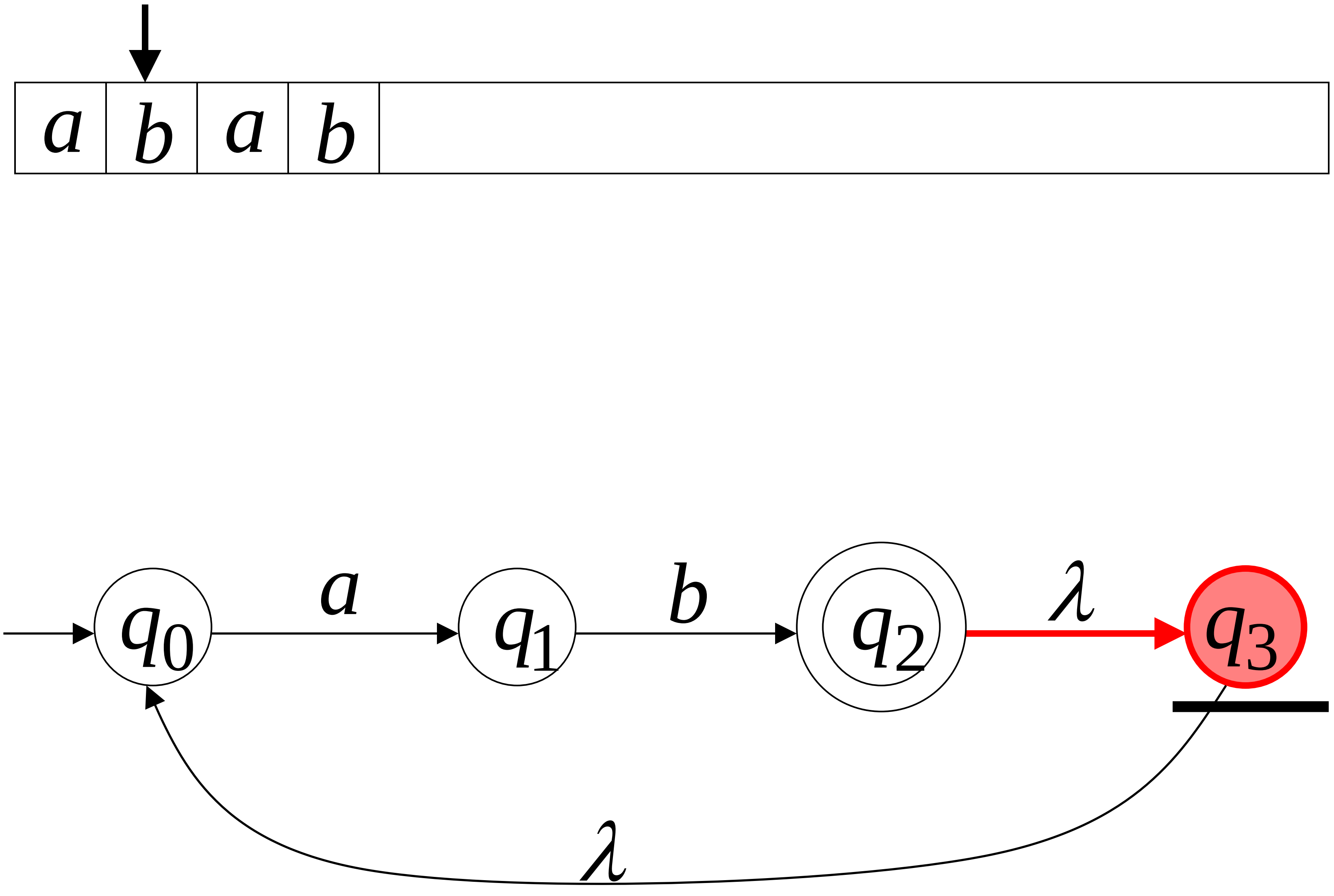


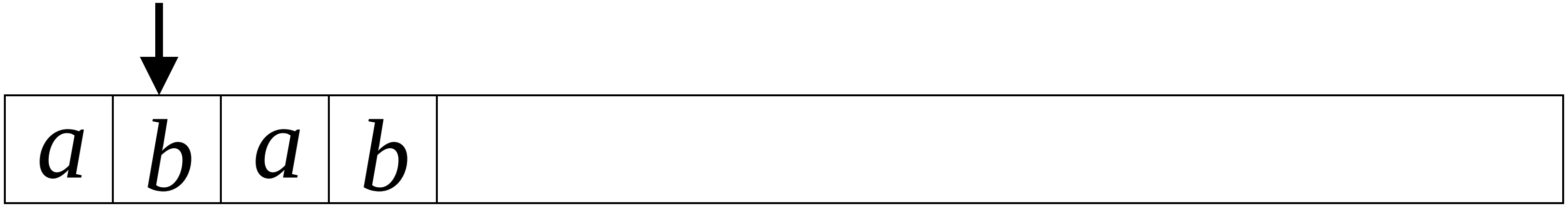

a
b
a
b

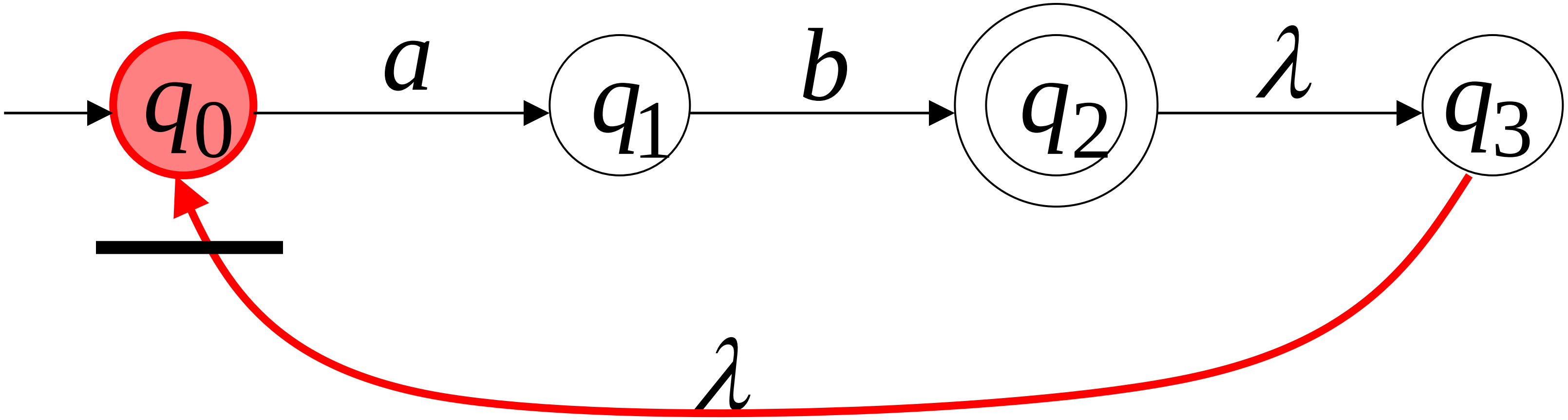

q_0
a
q_1
b
q_2
λ
q_3
λ

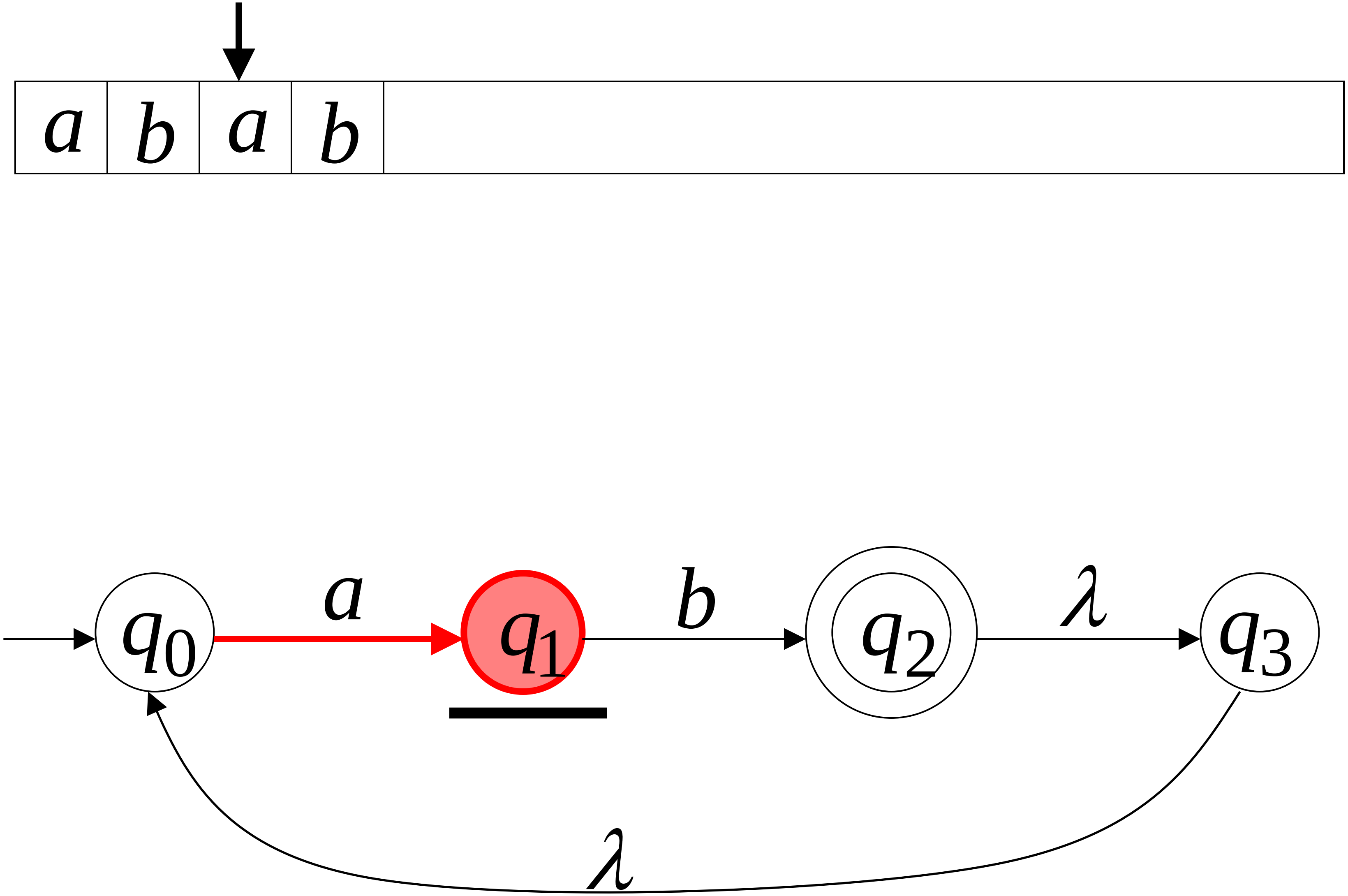

a
b
a
b

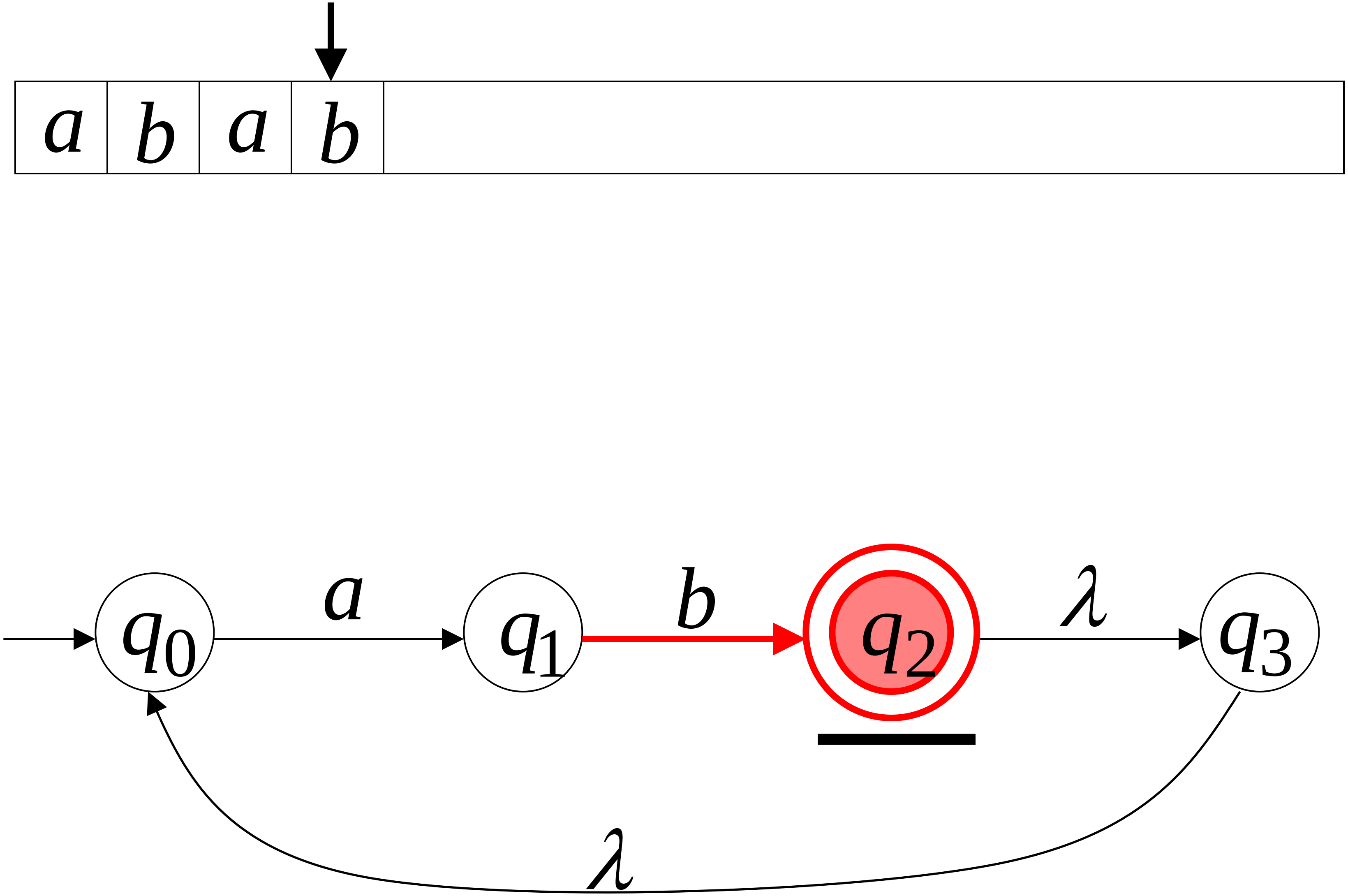

q_0
a
q_1
b
q_2
λ
q_3
λ

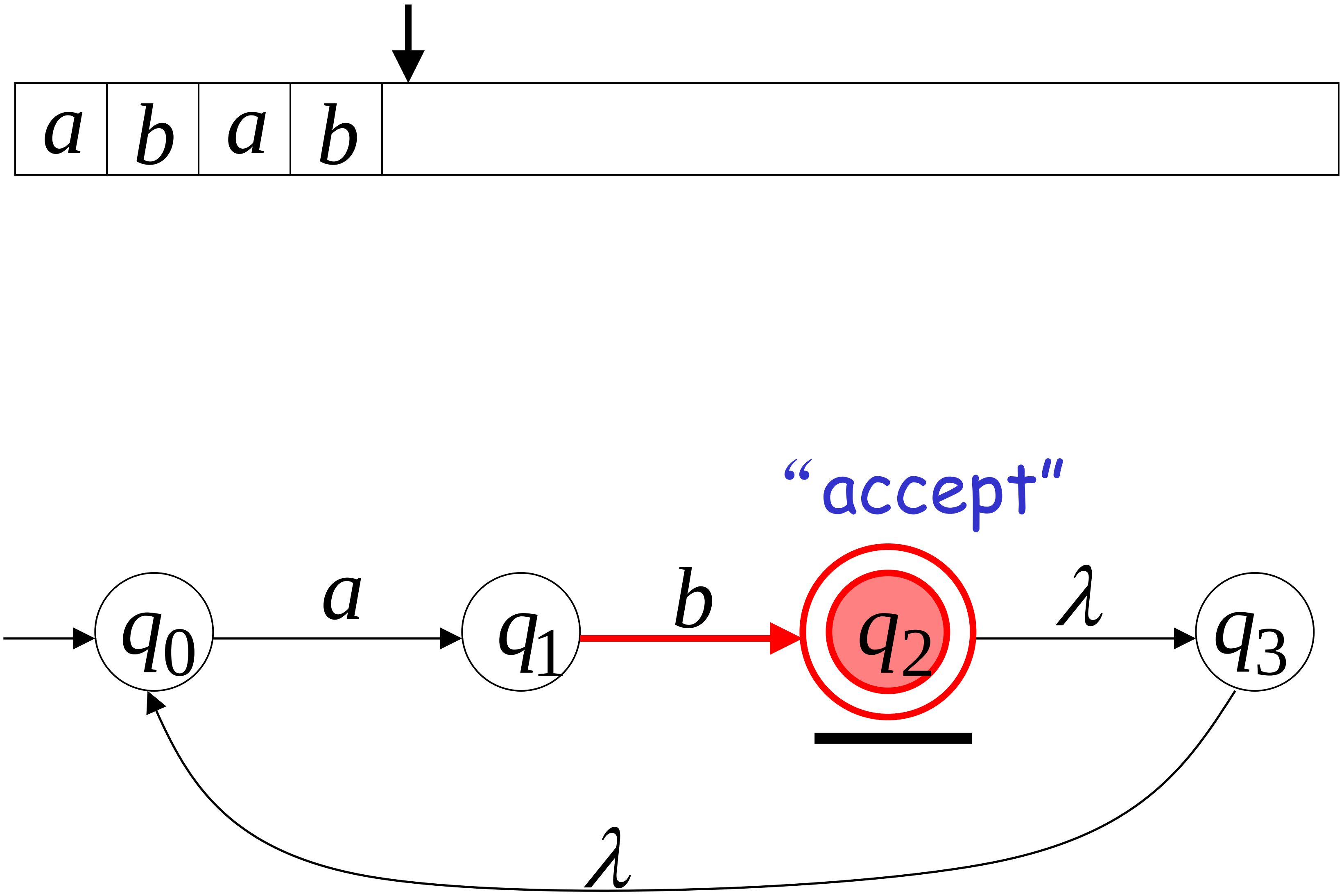

a
b
a
b
q_0
a
q_1
b
q_2
λ
q_3
λ


a
b
a
b


q_0
a
q_1
b
q_2
λ
q_3
λ


a
b
a
b
q_0
a
q_1
b
q_2
λ
q_3
λ


a
b
a
b
a
b
λ
λ
q_0
q_1
q_2
q_3

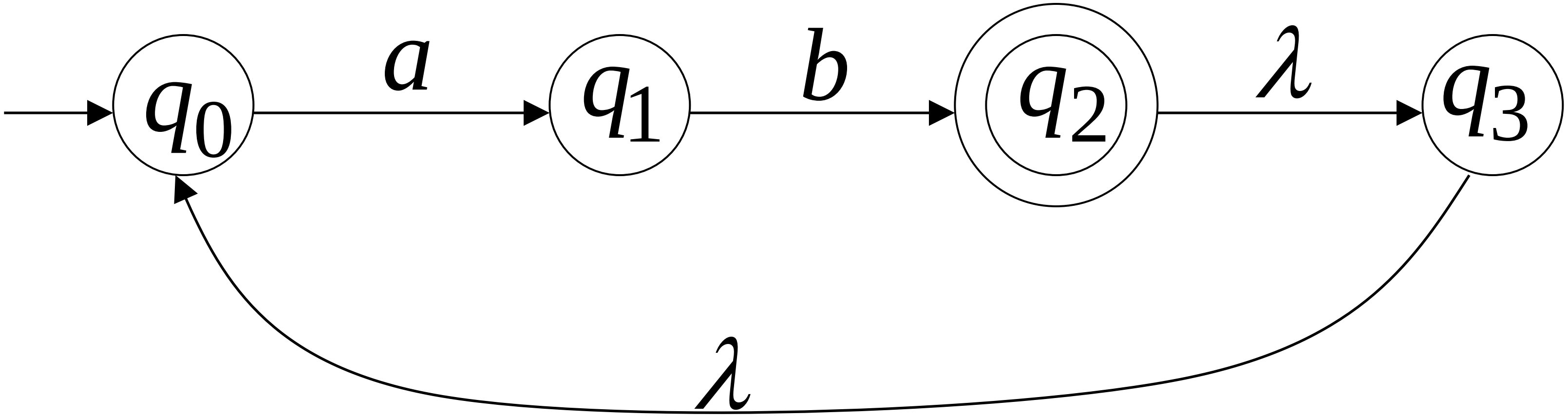


“accept”

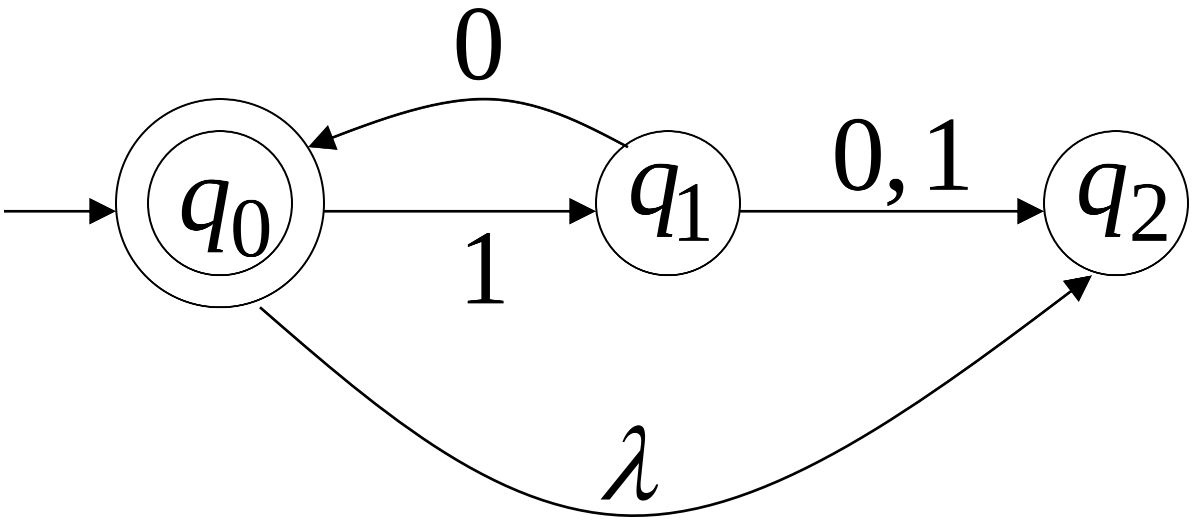
Language accepted

$$L = \{ab,\ abab,\ ababab,\ \ldots\}$$

$$= \{ab\}^{+}$$



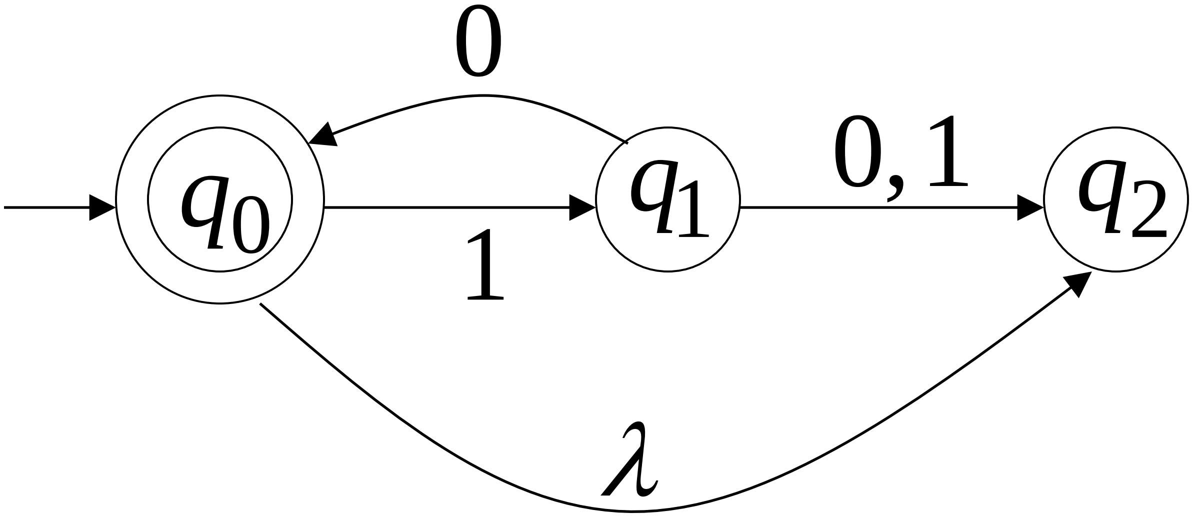
Another NFA Example



Language accepted

$$L(M) = \{\lambda,\ 10,\ 1010,\ 101010,\ \ldots\}$$

$$= \{10\}^*$$

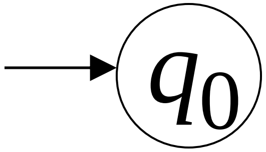


(redundant state)

Remarks:

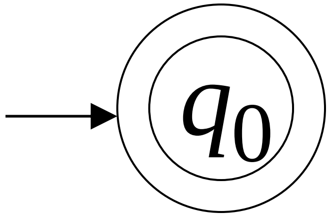
- The λ symbol never appears on the input tape
- Simple automata:

M_1



$$L(M_1) = \{\}$$

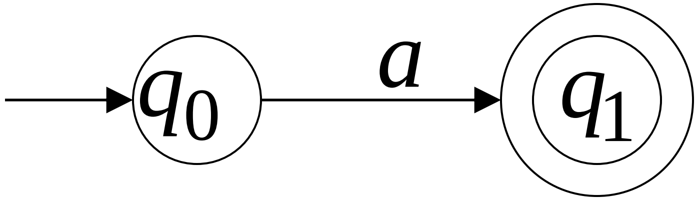
M_2



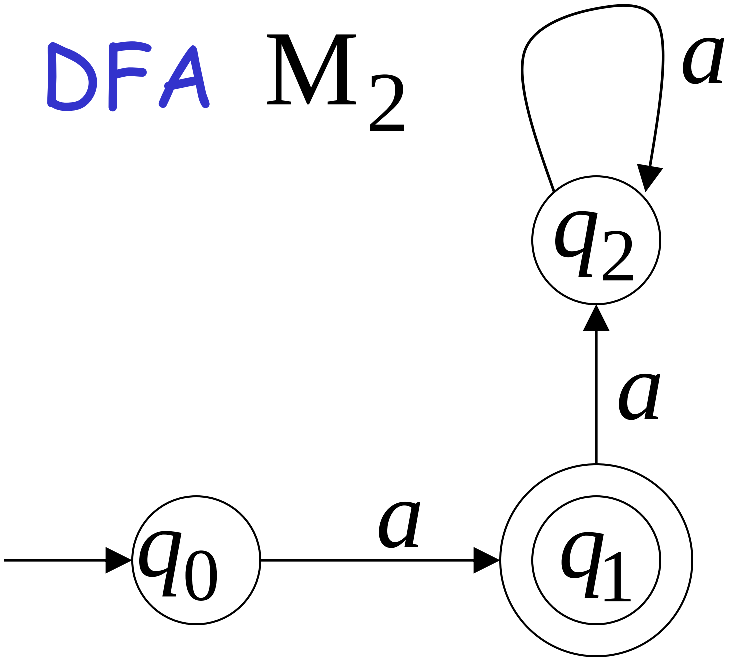
$$L(M_2) = \{\lambda\}$$

- NFAs are interesting because we can express languages easier than DFAs

NFA M_1



DFA M_2



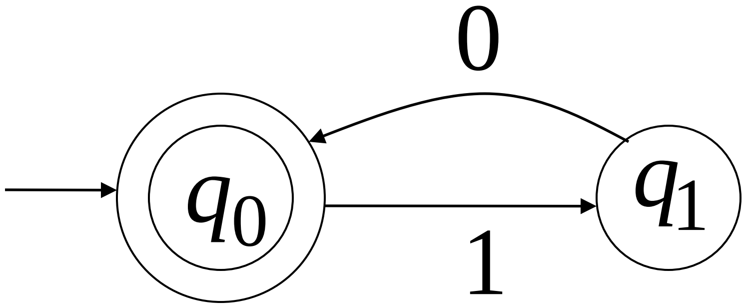
$$L(M_1) = \{a\}$$

$$L(M_2) = \{a\}$$

Example

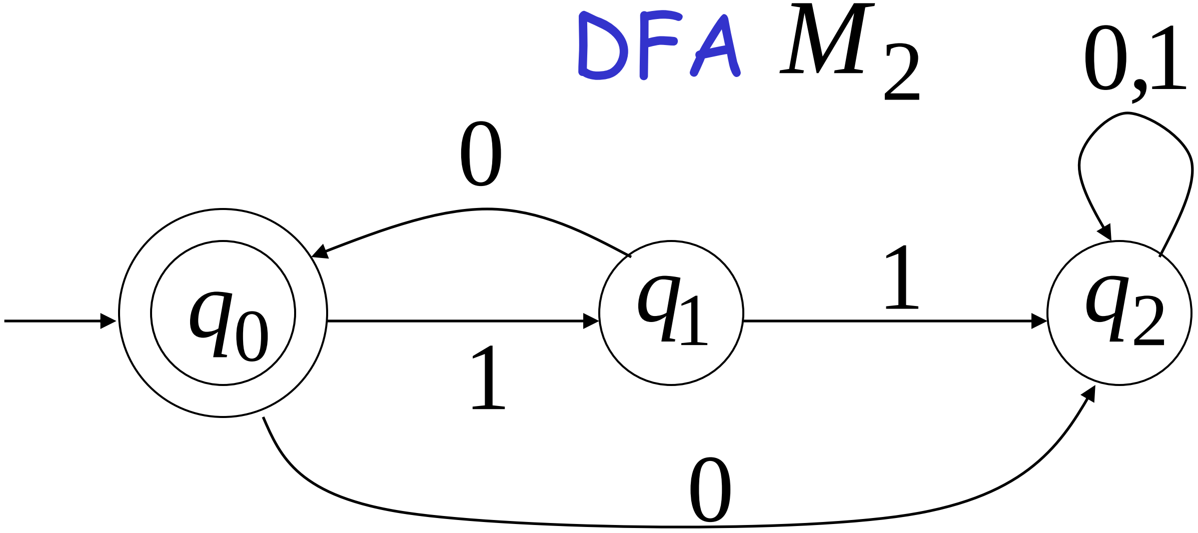
$$L(M_1) = \{10\}^*$$

NFA M_1



$$L(M_2) = \{10\}^*$$

DFA M_2



Formal Definition of NFAs

$$M = (Q,\ \Sigma,\ \delta,\ q_0,\ F)$$

$Q:$ Set of states, i.e. $\{q_0, q_1, q_2\}$

$\Sigma:$ Input aplhabet, i.e. $\{a, b\}$

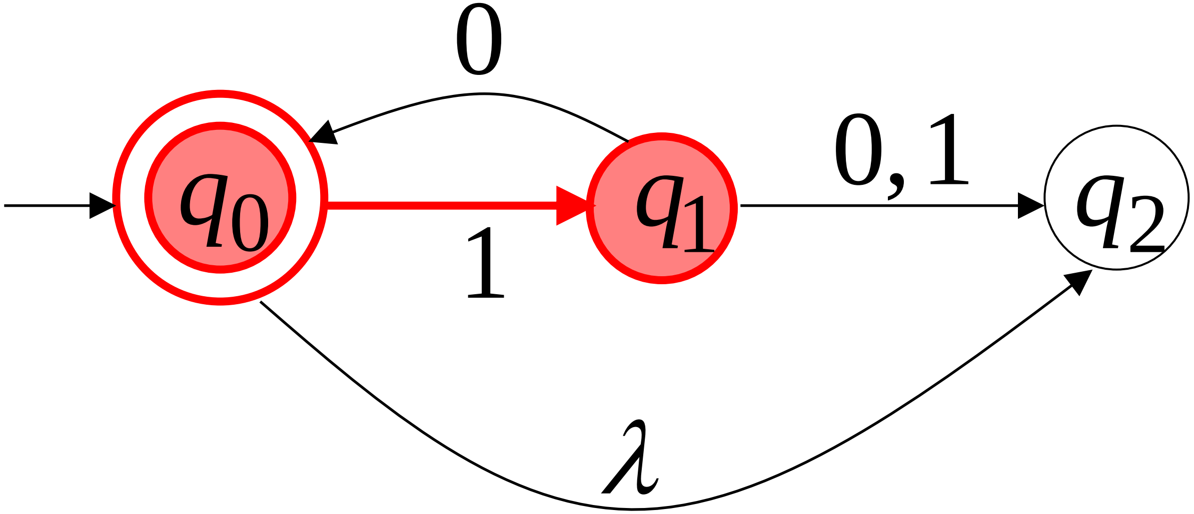
$\delta:$ Transition function

$q_0:$ Initial state

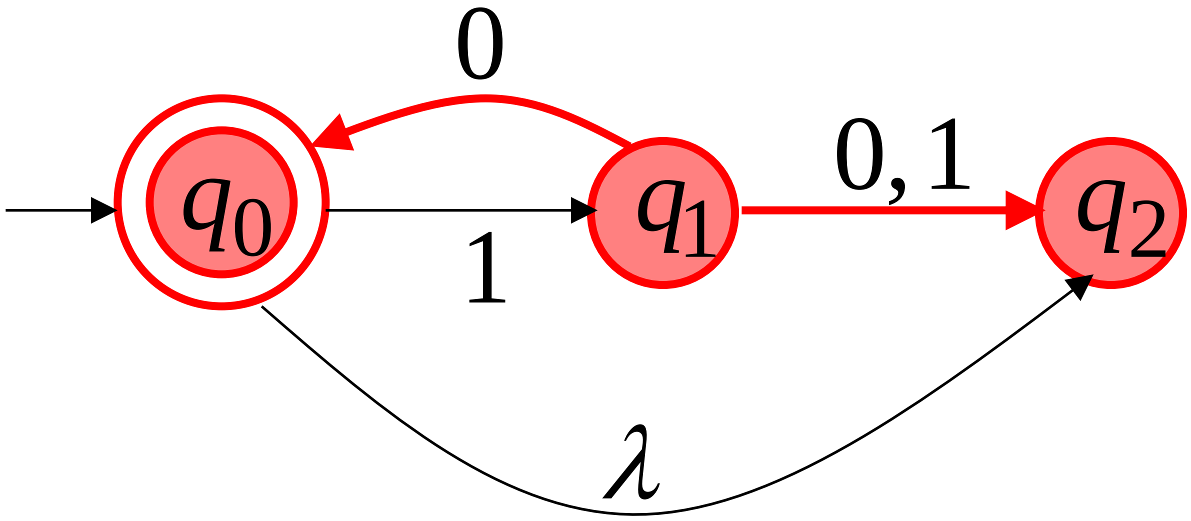
$F:$ Final states

Transition Function δ

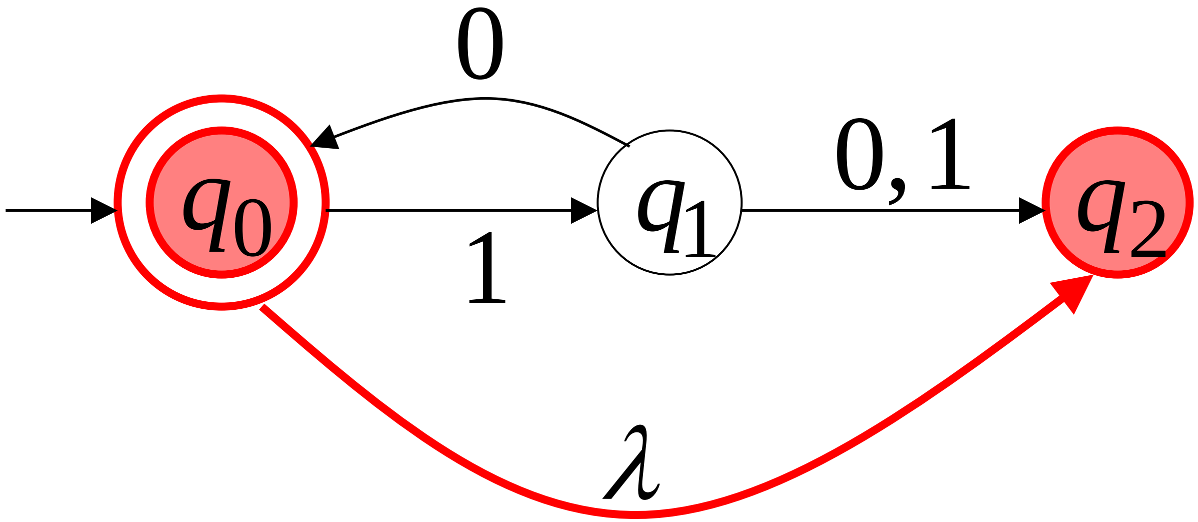
$$\delta(q_0, 1) = \{q_1\}$$



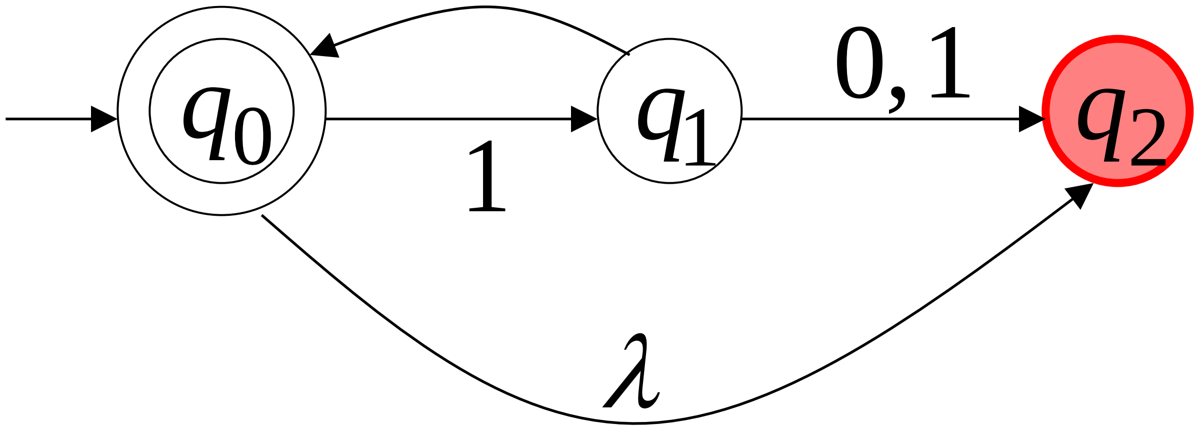
$$\delta(q_1, 0) = \{q_0, q_2\}$$



$$\delta(q_0, \lambda) = \{q_0, q_2\}$$

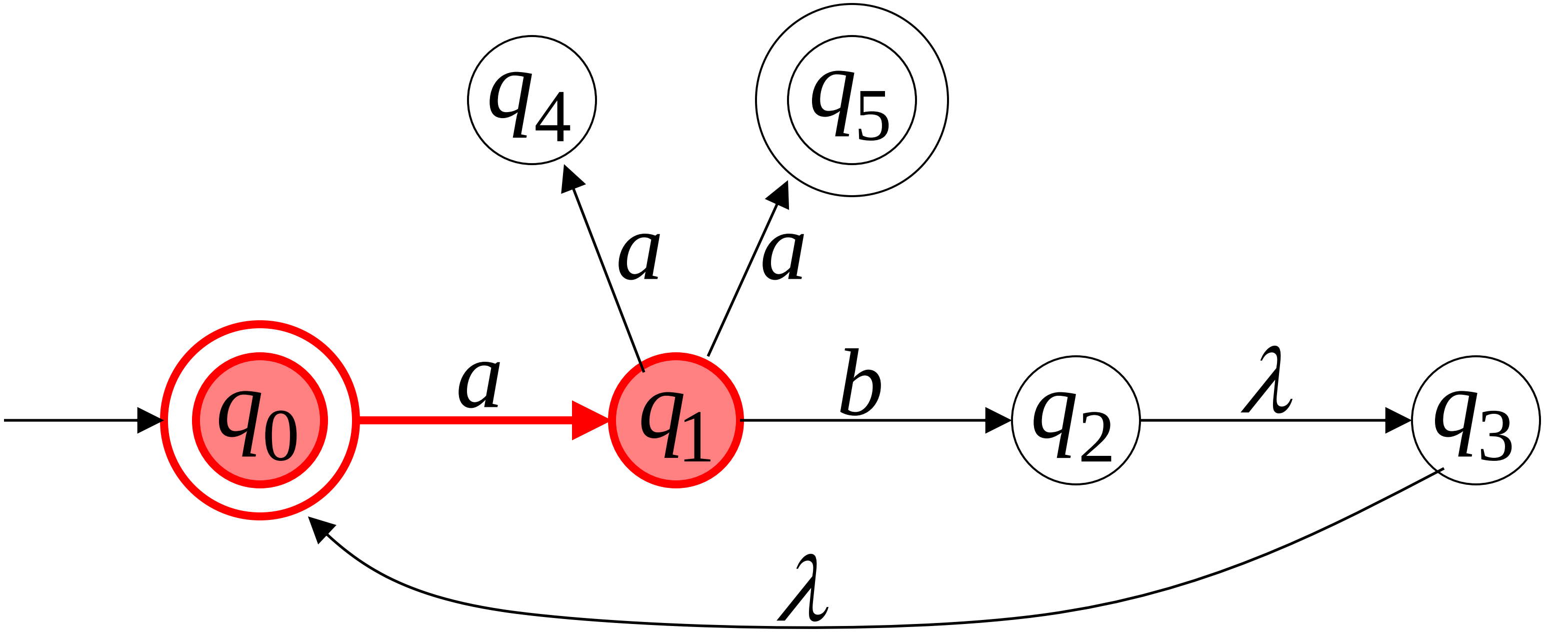


$$\delta(q_2, 1) = \varnothing$$

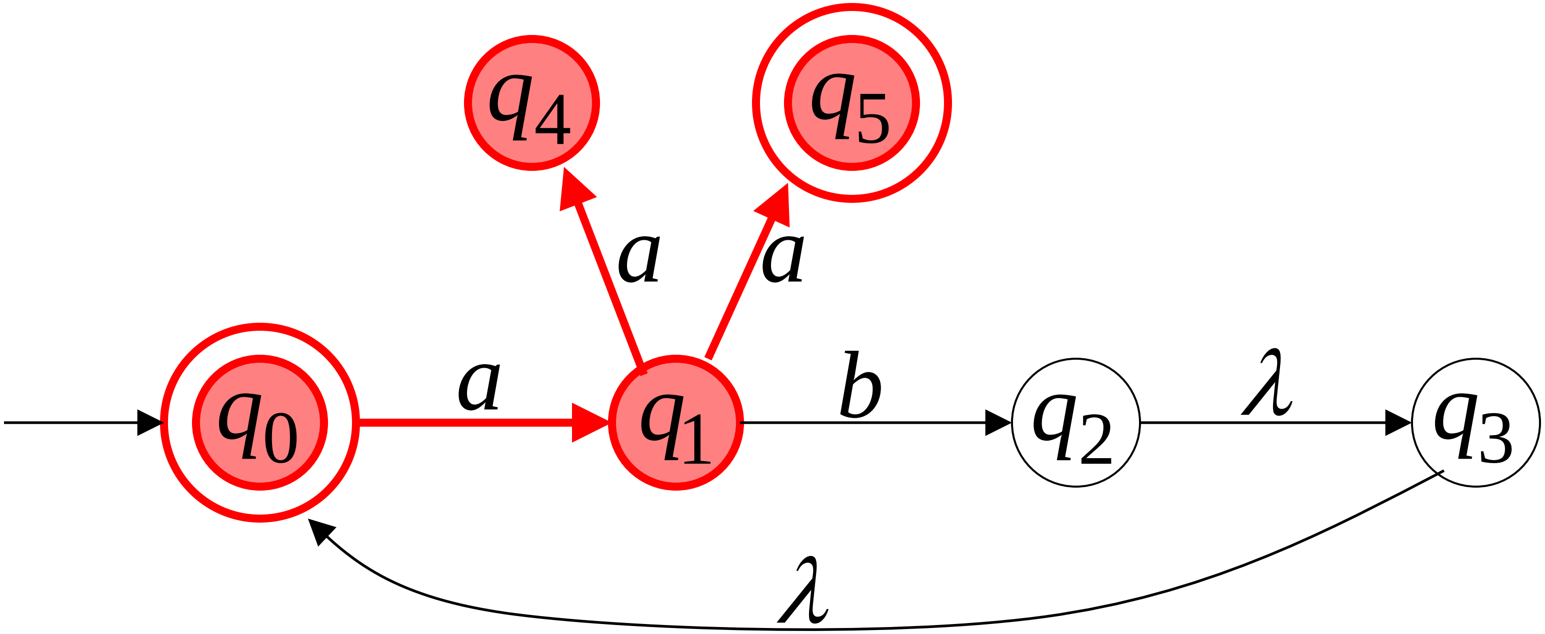


Extended Transition Function δ^*

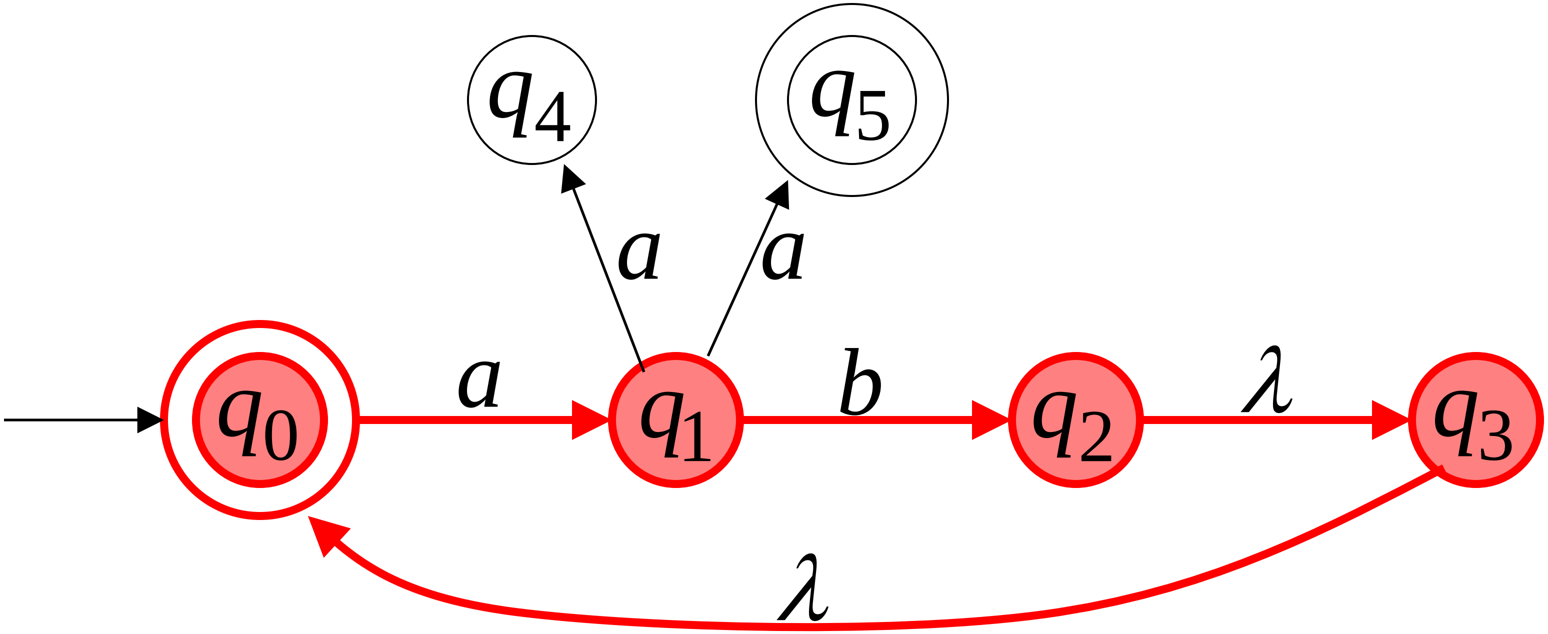
$$\delta^*(q_0, a) = \{q_1\}$$



$$\delta^*(q_0, aa) = \{q_4, q_5\}$$

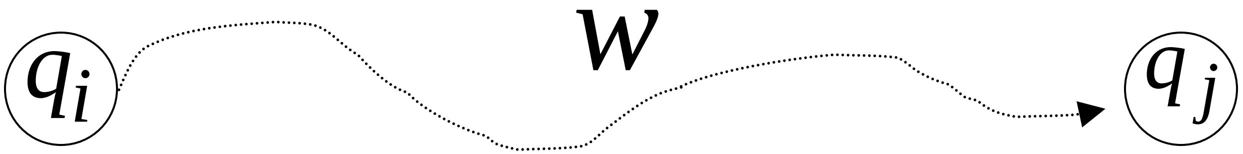


$$\delta^*(q_0, ab) = \{q_2, q_3, q_0\}$$

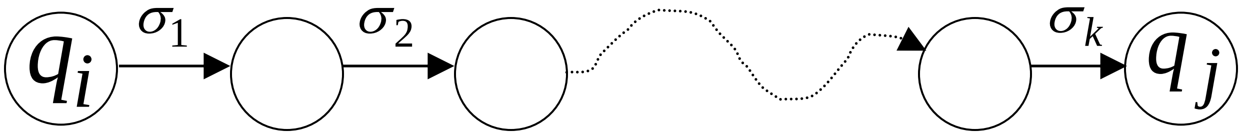


Formally

$q_j \in \delta^*(q_i, w)$: there is a walk from q_i to q_j with label w

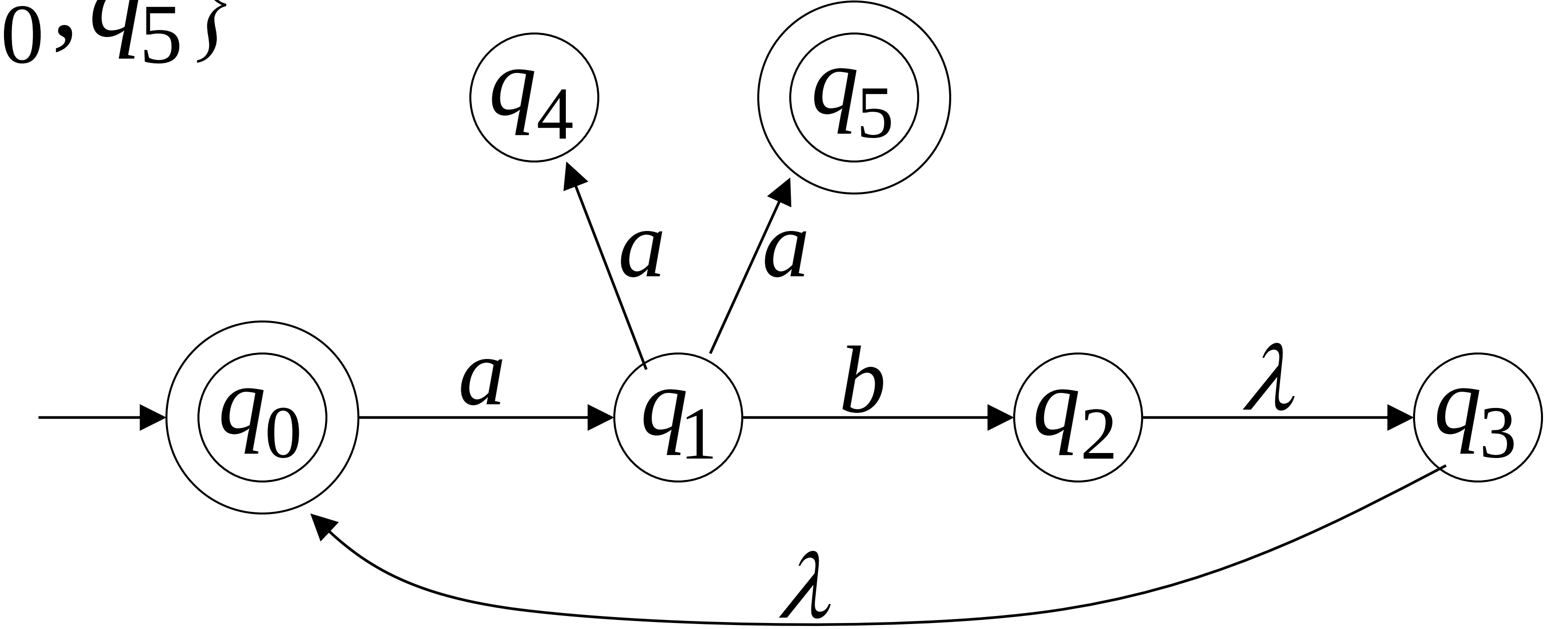


$$w = \sigma_1 \sigma_2 \cdots \sigma_k$$



The Language of an NFA M

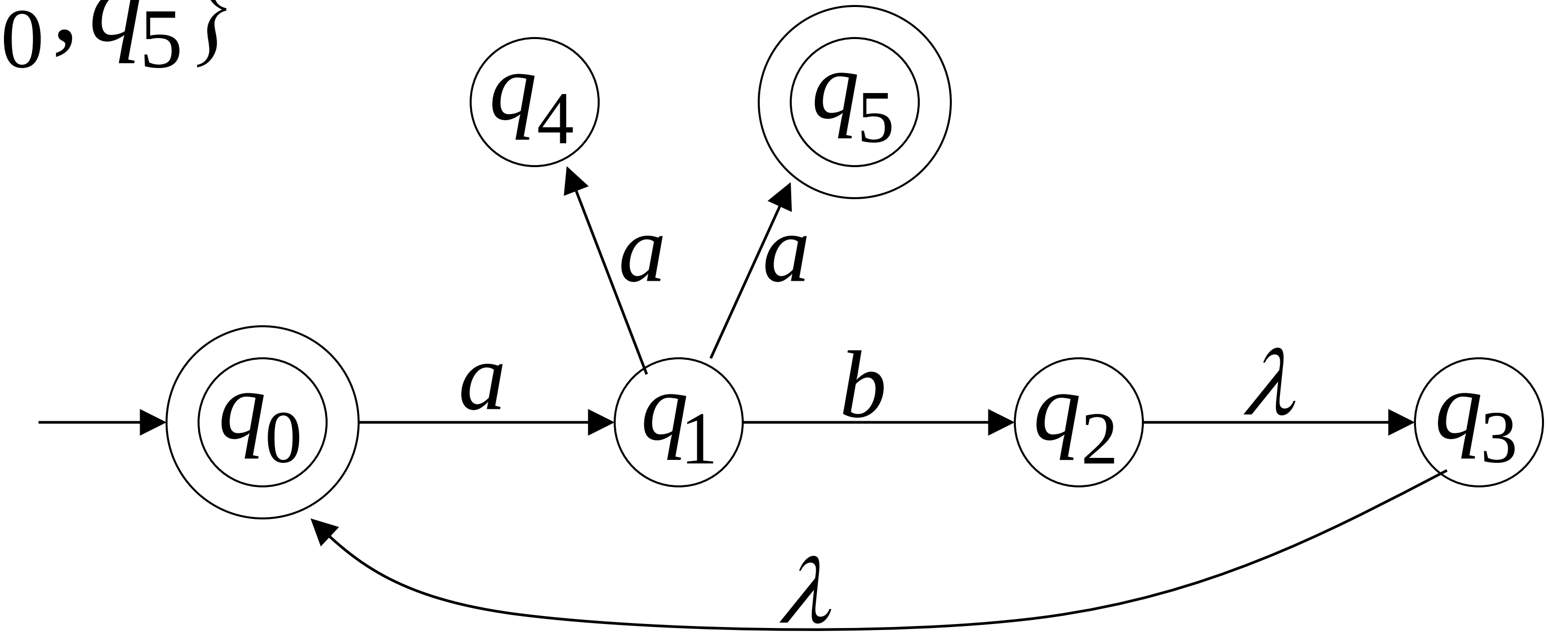
$F = \{q_0, q_5\}$



$$\delta^*(q_0, aa) = \{q_4, \underline{q_5}\} \qquad aa \in L(M)$$

$$\searrow \in F$$

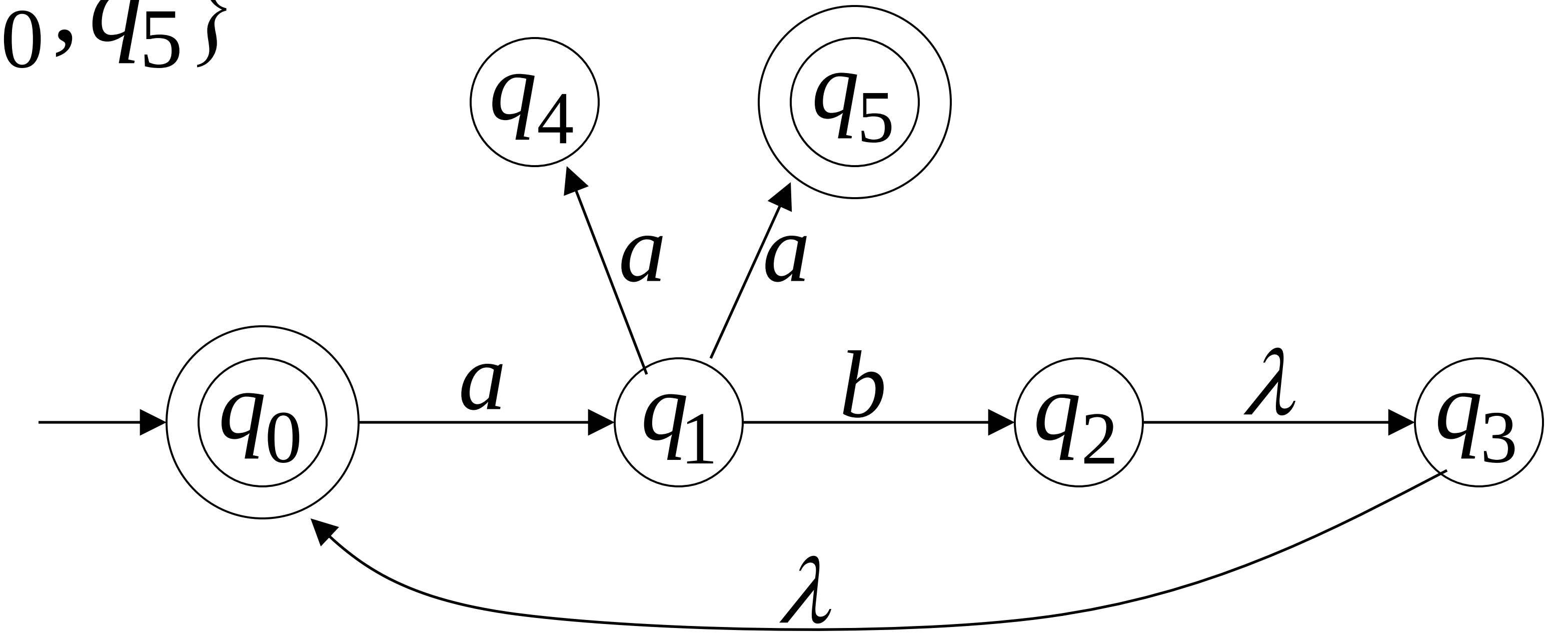
$$F = \{q_0, q_5\}$$



$$\delta^*(q_0, ab) = \{q_2, q_3, \underline{q_0}\} \qquad ab \in L(M)$$

$$\searrow \in F$$

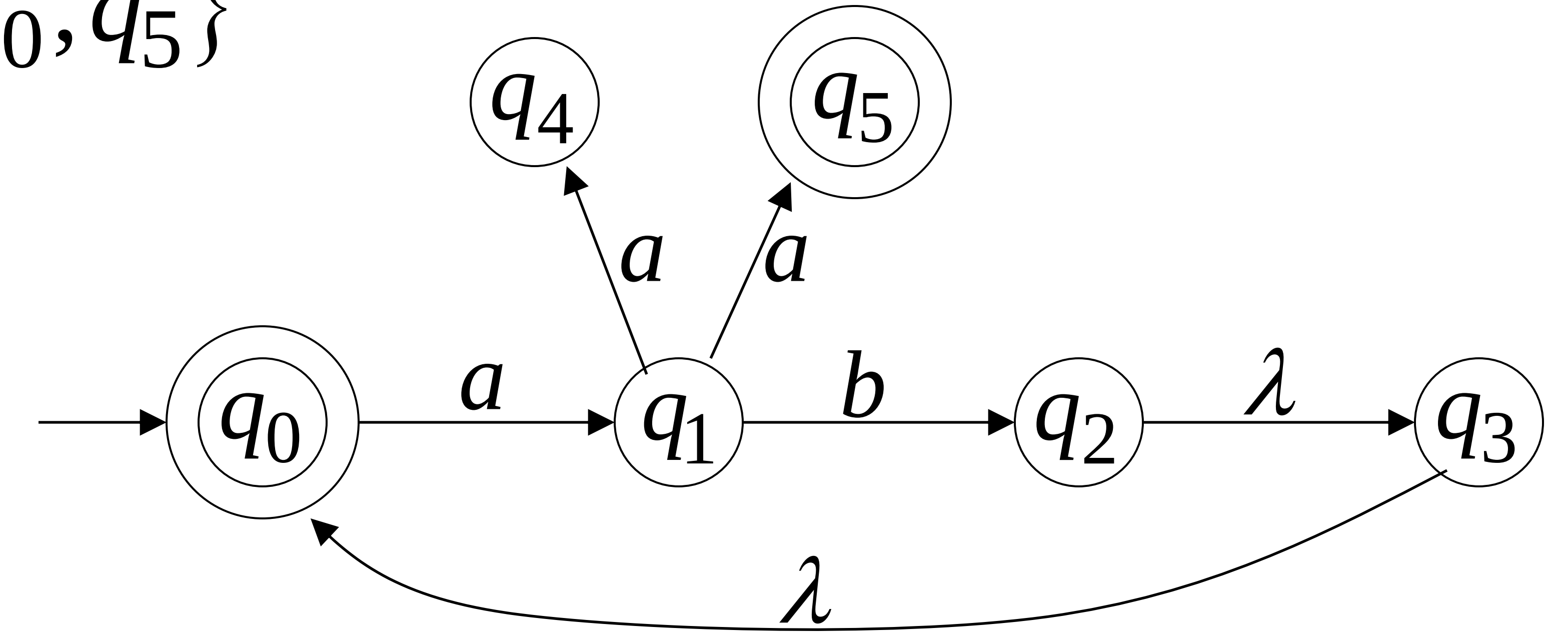
$$F = \{q_0, q_5\}$$



$$\delta^*(q_0, abaa) = \{q_4, \underline{q_5}\} \qquad aaba \in L(M)$$

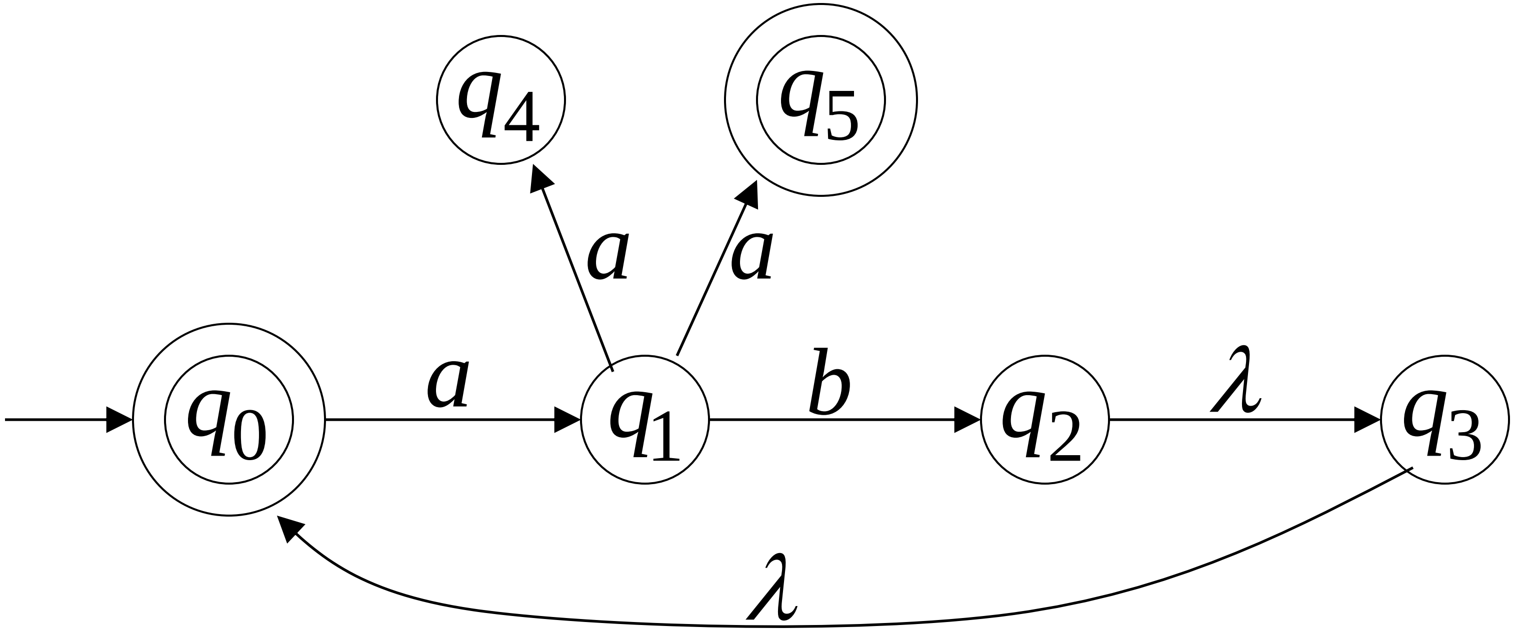
$$\searrow \in F$$

$$F = \{q_0, q_5\}$$



$$\delta^*(q_0, aba) = \{q_1\} \qquad aba \notin L(M)$$

$$\searrow \notin F$$



$$L(M) = \{\lambda\} \cup \{ab\}^* \{aa\}$$

Formally

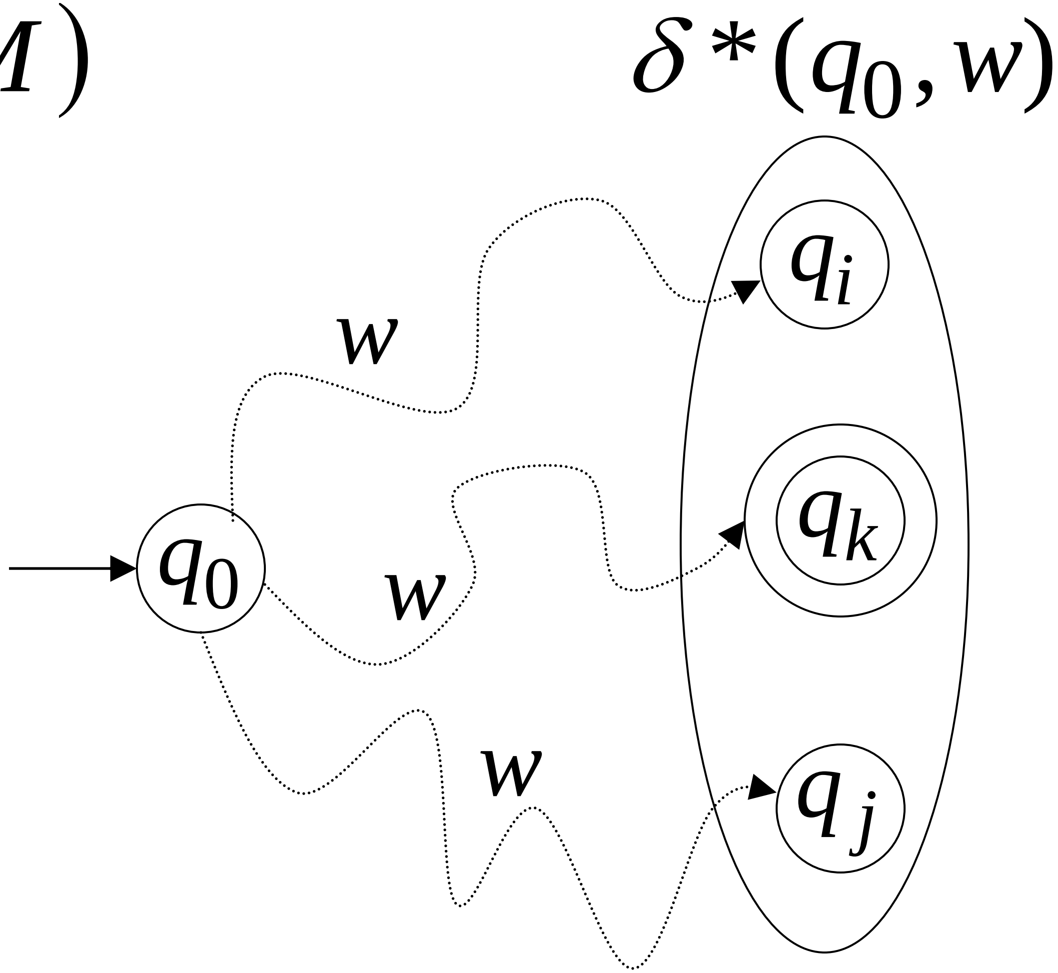
The language accepted by NFA M is:

$$L(M) = \{w_1, w_2, w_3, \ldots\}$$

where $\delta^*(q_0, w_m) = \{q_i, q_j, \ldots, q_k, \ldots\}$

and there is some $q_k \in F$ (final state)

$w \in L(M)$



$q_k \in F$

NFAs accept the Regular Languages

Equivalence of Machines

Definition for Automata:

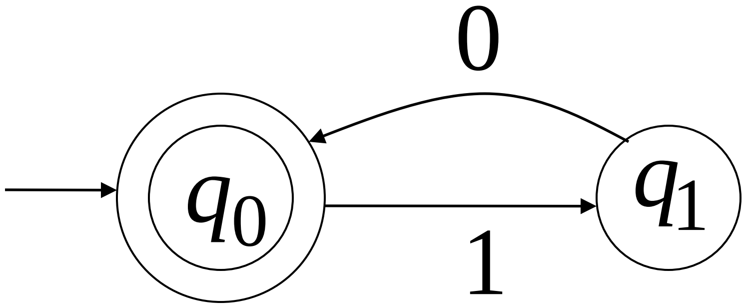
Machine M_1 is equivalent to machine M_2

if $L(M_1) = L(M_2)$

Example of equivalent machines

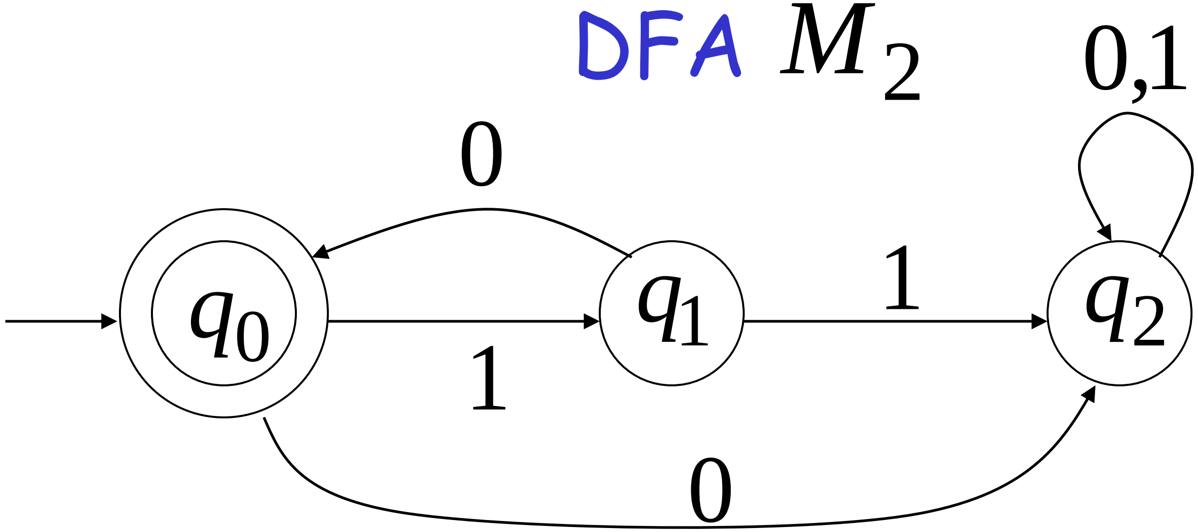
NFA M_1

$$L(M_1) = \{10\}*$$



DFA M_2

$$L(M_2) = \{10\}*$$



We will prove:

$$\left\{ \begin{array}{l} \text{Languages} \\ \text{accepted} \\ \text{by NFAs} \end{array} \right\} = \left\{ \begin{array}{l} \text{Regular} \\ \text{Languages} \end{array} \right\}$$

Languages
accepted
by DFAs

NFAs and DFAs have the same computation power

Step 1

$$\left\{\begin{array}{l}\text{Languages}\\ \text{accepted}\\ \text{by NFAs}\end{array}\right\} \supseteq \left\{\begin{array}{l}\text{Regular}\\ \text{Languages}\end{array}\right\}$$

Proof: Every DFA is trivially an NFA

⬇

Any language L accepted by a DFA is also accepted by an NFA

Step 2

$$\left\{\begin{array}{l}\text{Languages}\\ \text{accepted}\\ \text{by NFAs}\end{array}\right\} \subseteq \left\{\begin{array}{l}\text{Regular}\\ \text{Languages}\end{array}\right\}$$

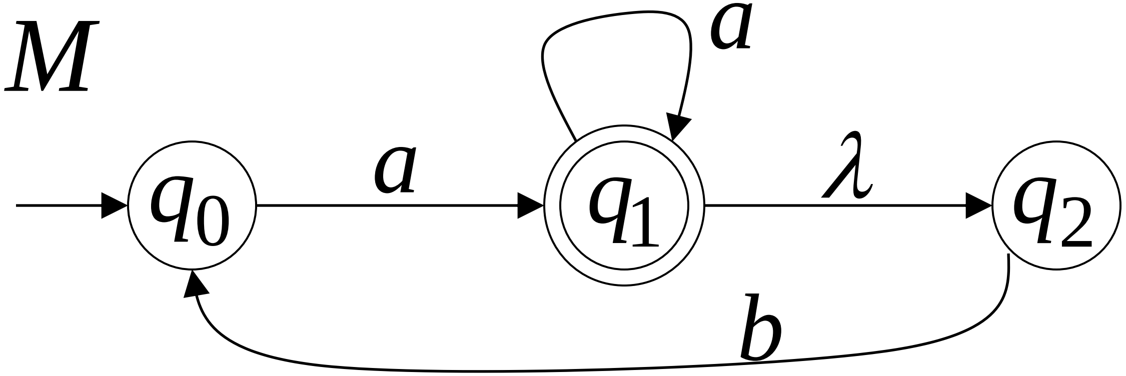
Proof: Any NFA can be converted to an equivalent DFA

⇓

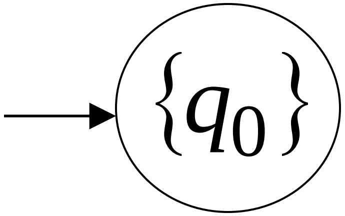
Any language L accepted by an NFA is also accepted by a DFA

Convert NFA to DFA

NFA M

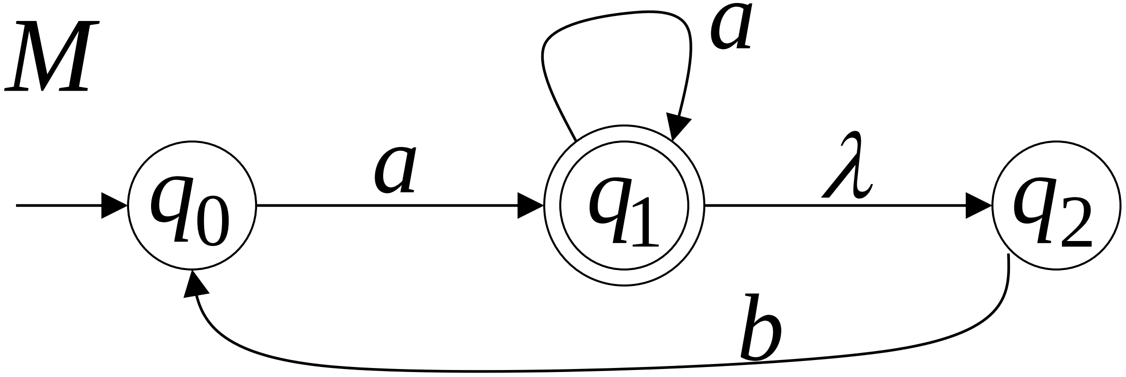


DFA M'

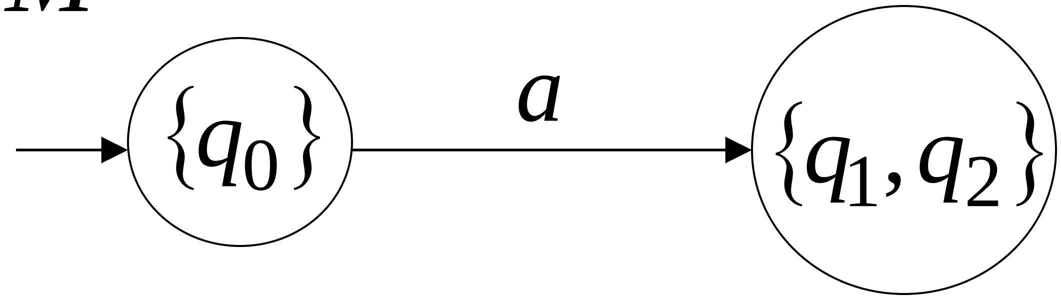


Convert NFA to DFA

NFA M

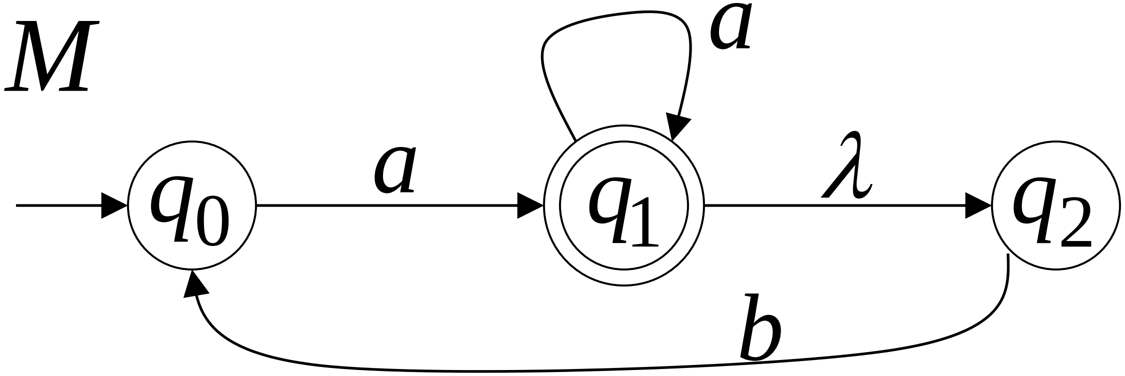


DFA M'

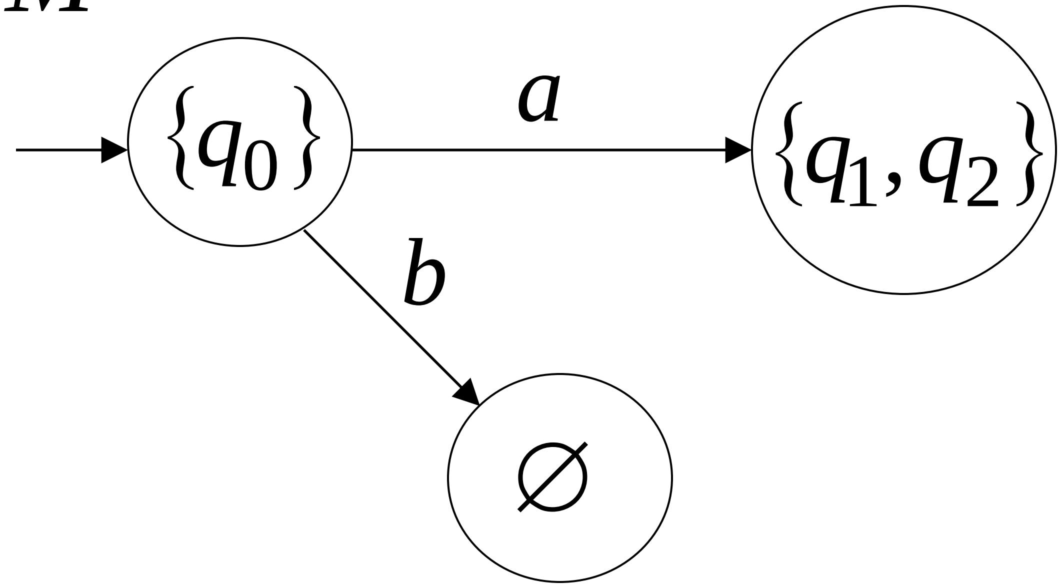


Convert NFA to DFA

NFA M

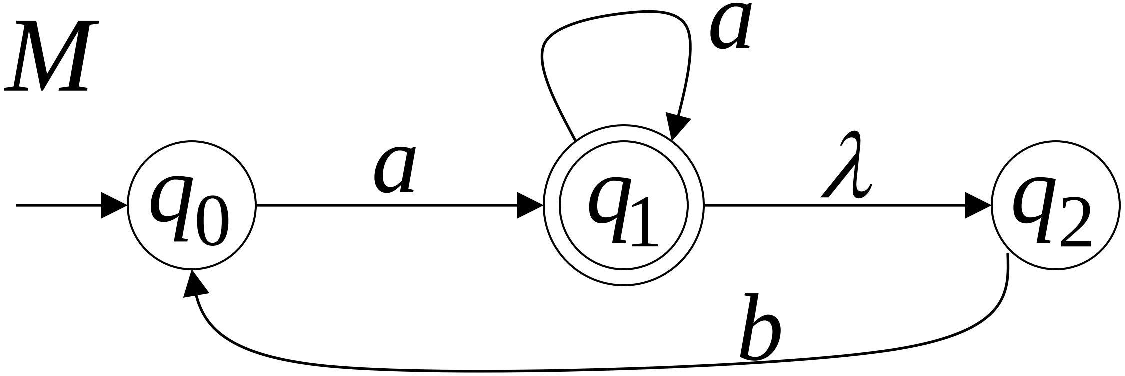


DFA M'

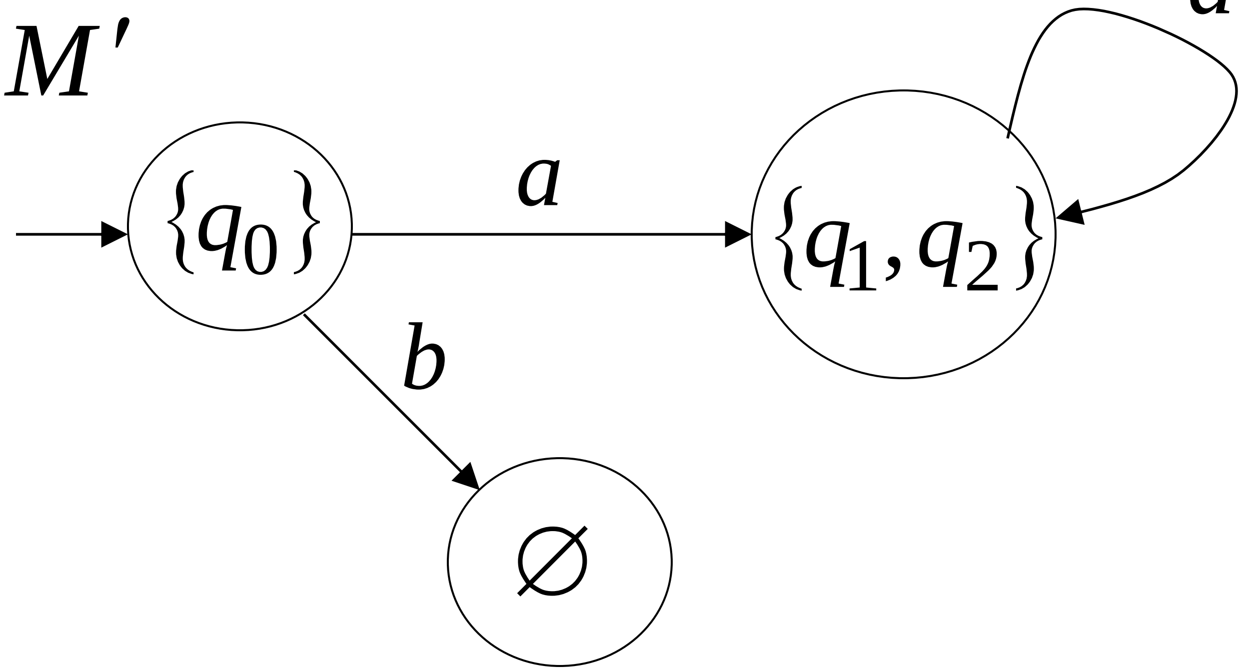


Convert NFA to DFA

NFA M

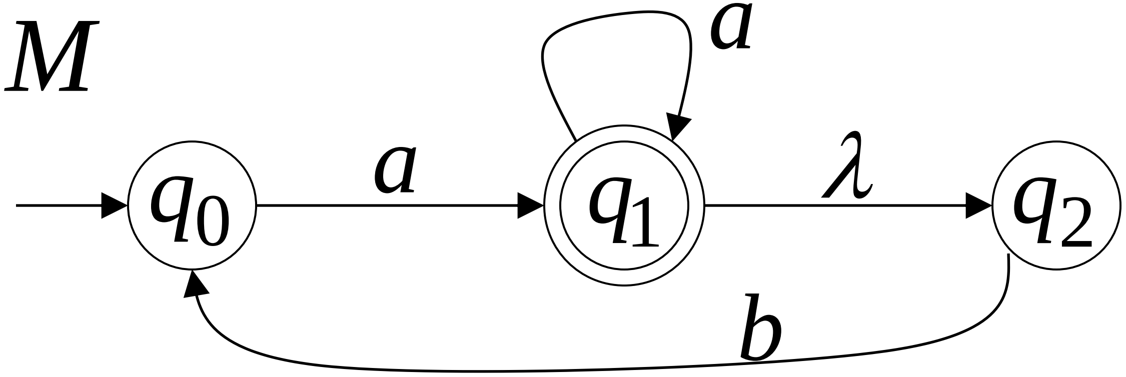


DFA M'

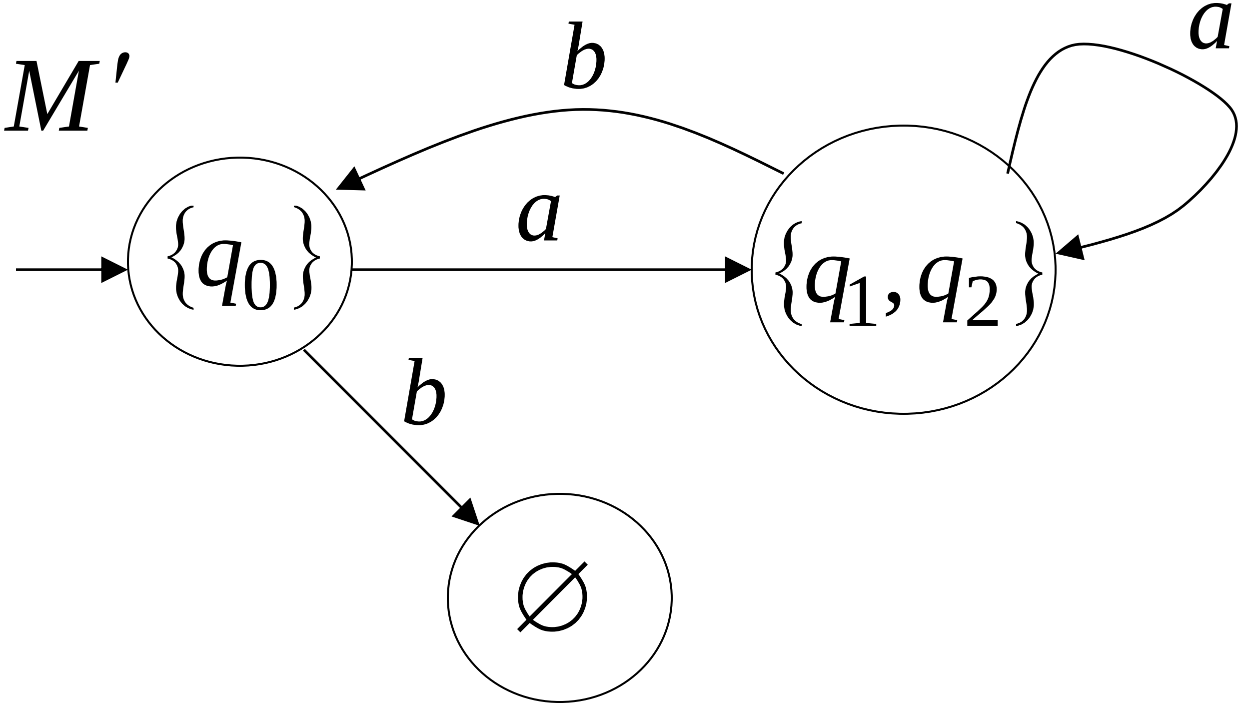


Convert NFA to DFA

NFA M

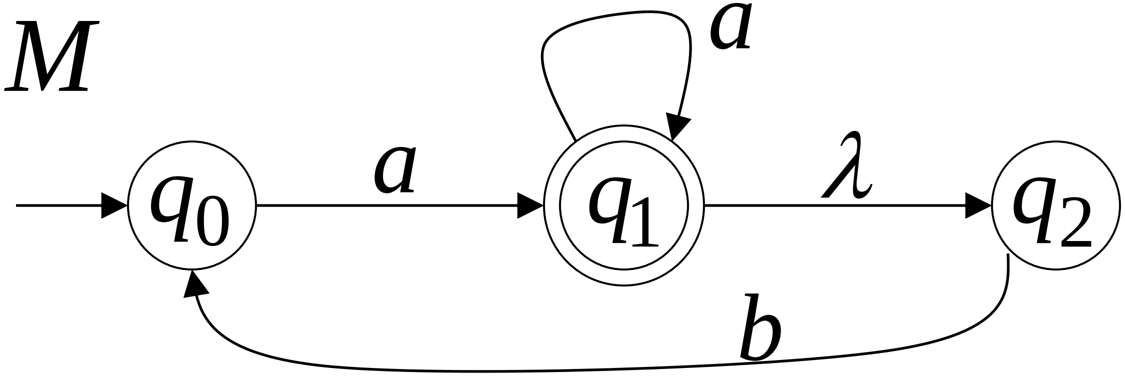


DFA M'

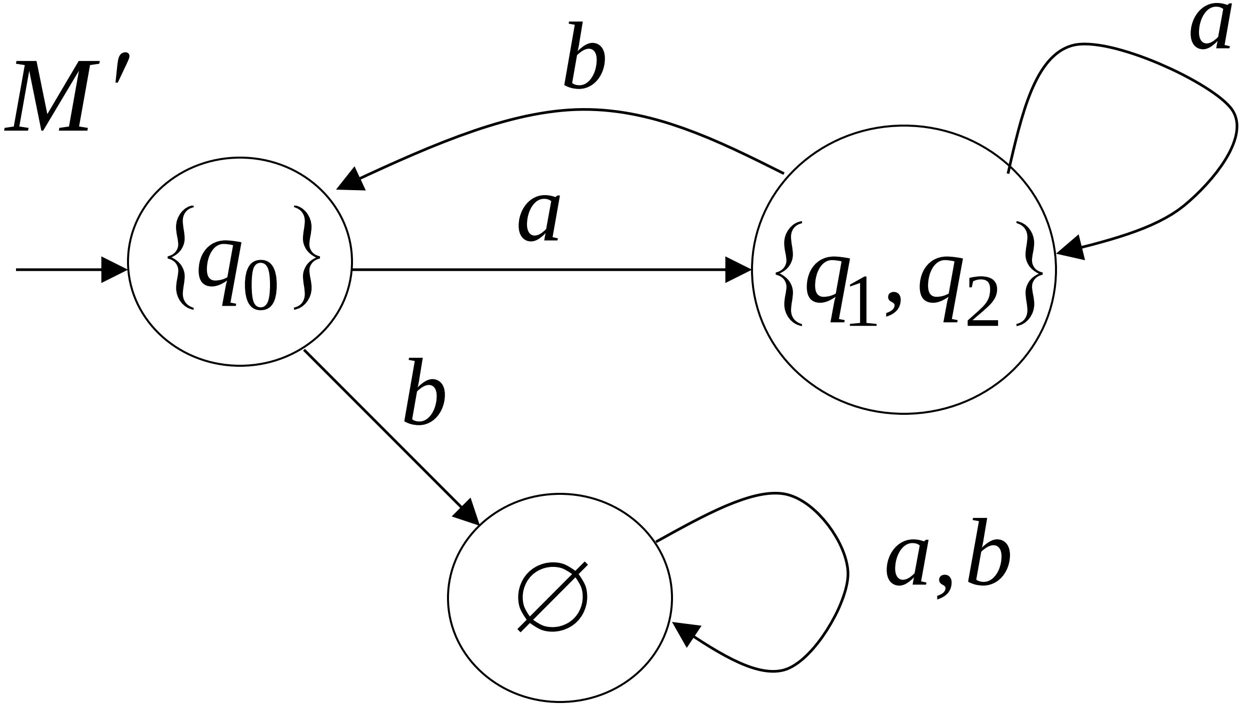


Convert NFA to DFA

NFA M

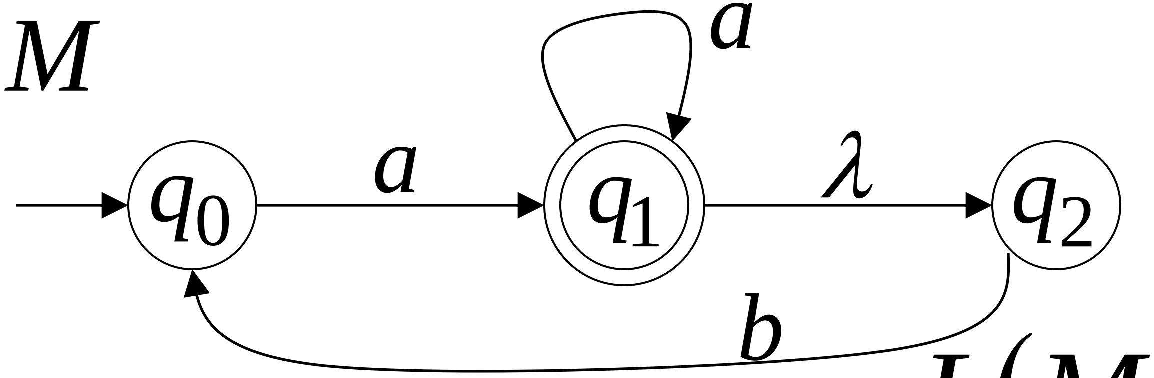


DFA M'



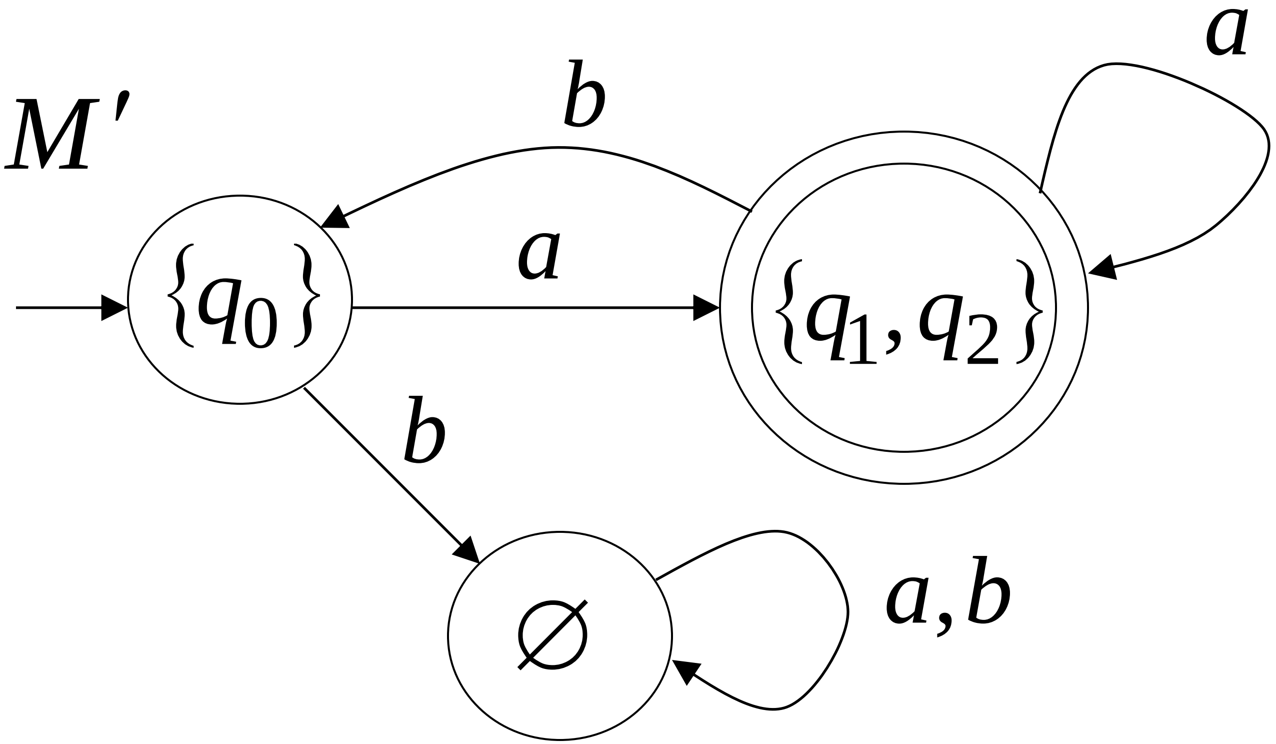
Convert NFA to DFA

NFA M



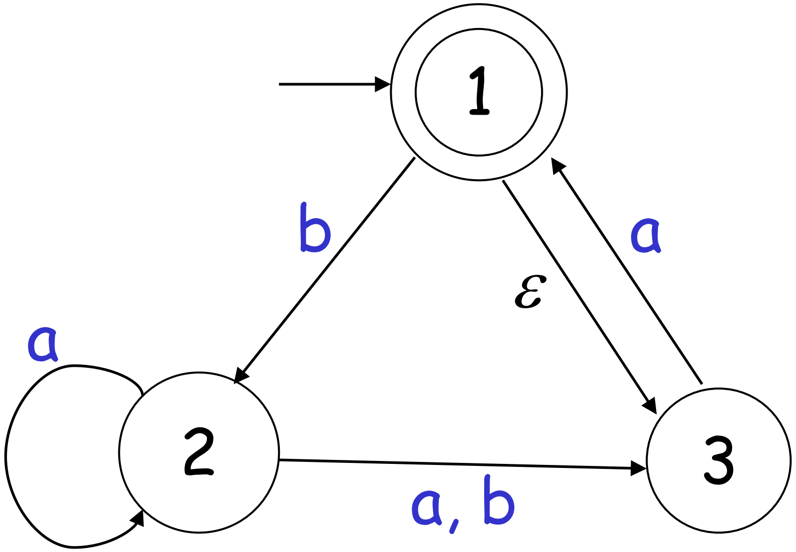
$$L(M)=L(M')$$

DFA M'



More example: Converting NFA to DFA

- NFA $N_4 = (Q, \{a,b\}, \delta, 1, \{1\})$, the set of states Q is {1,2,3} as shown in the following figure.



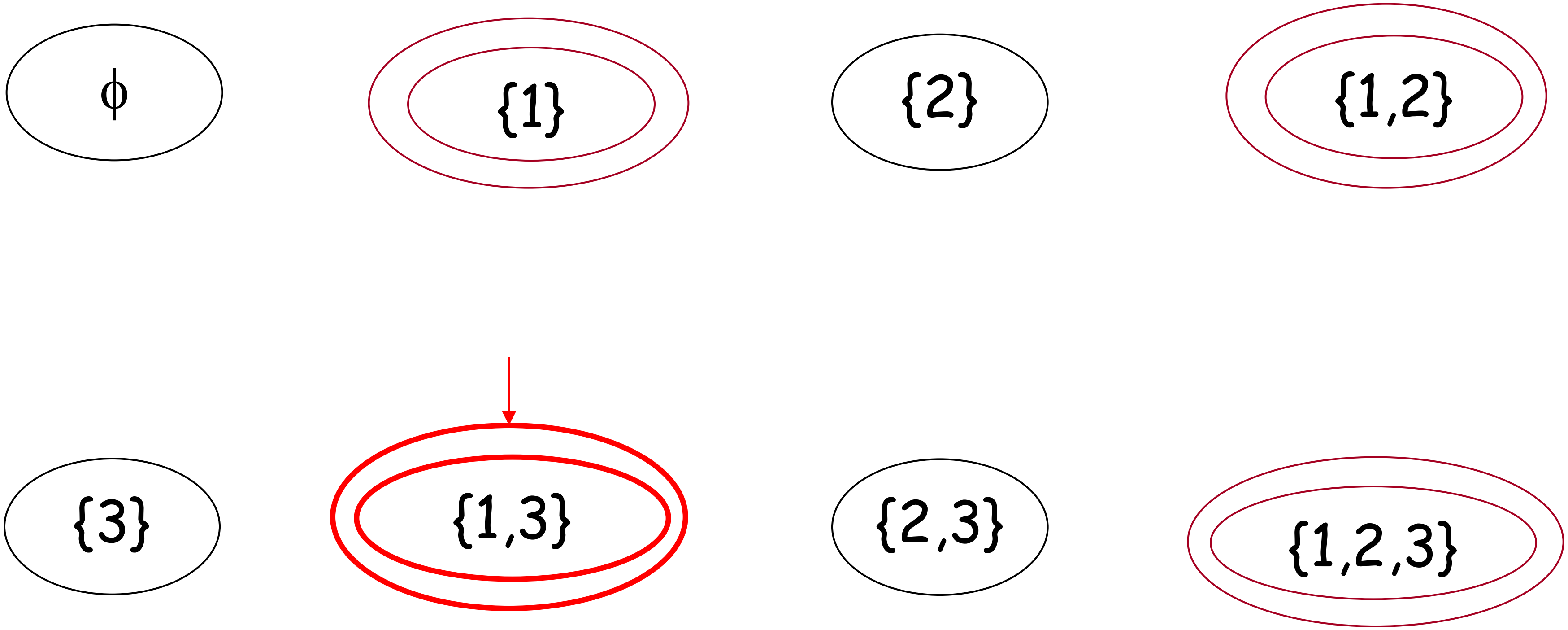
The NFA N_4

Step1: Determine DFA's states

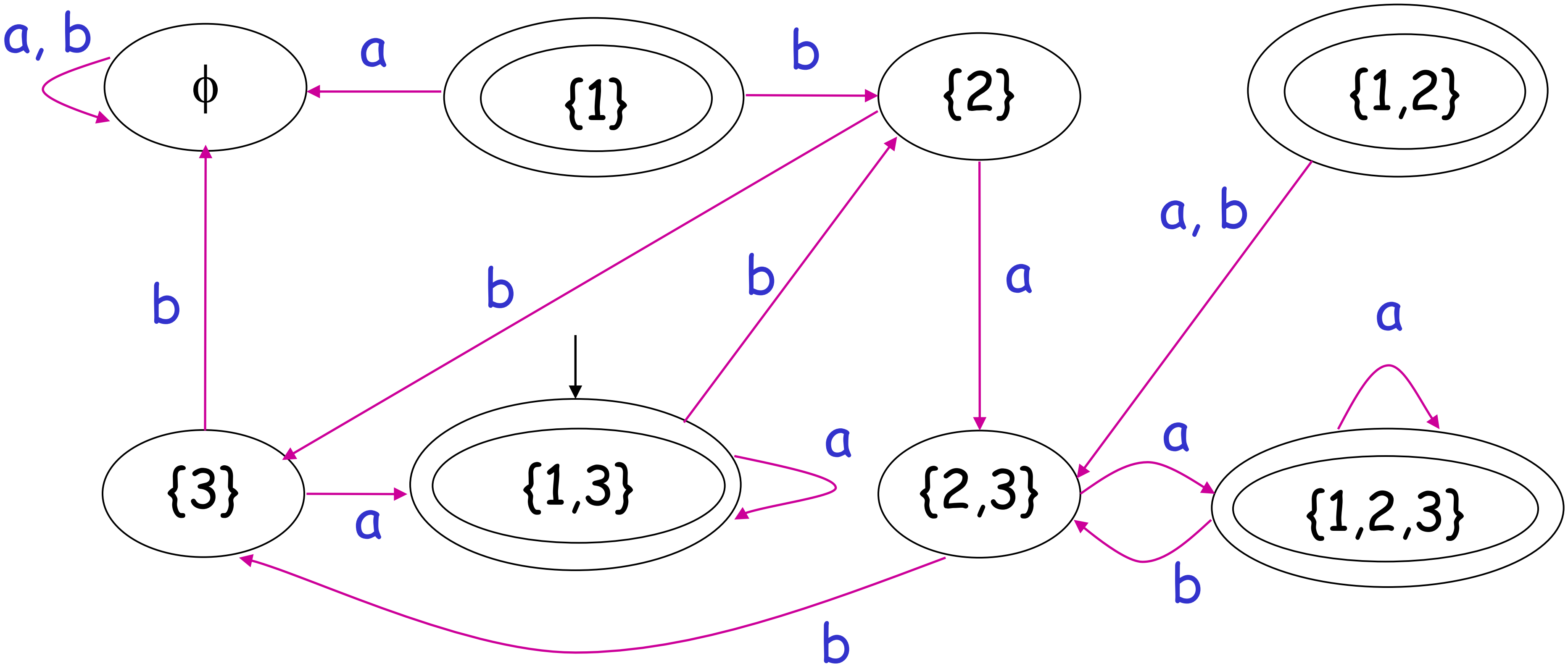
- N_4 has three states {1,2,3}, so we construct DFA D with eight.

- We label each of D's states with the corresponding subset. Thus D's state set is $\{\emptyset,\{1\},\{2\},\{3\},\{1,2\},\{1,3\},\{2,3\},\{1,2,3\}\}$

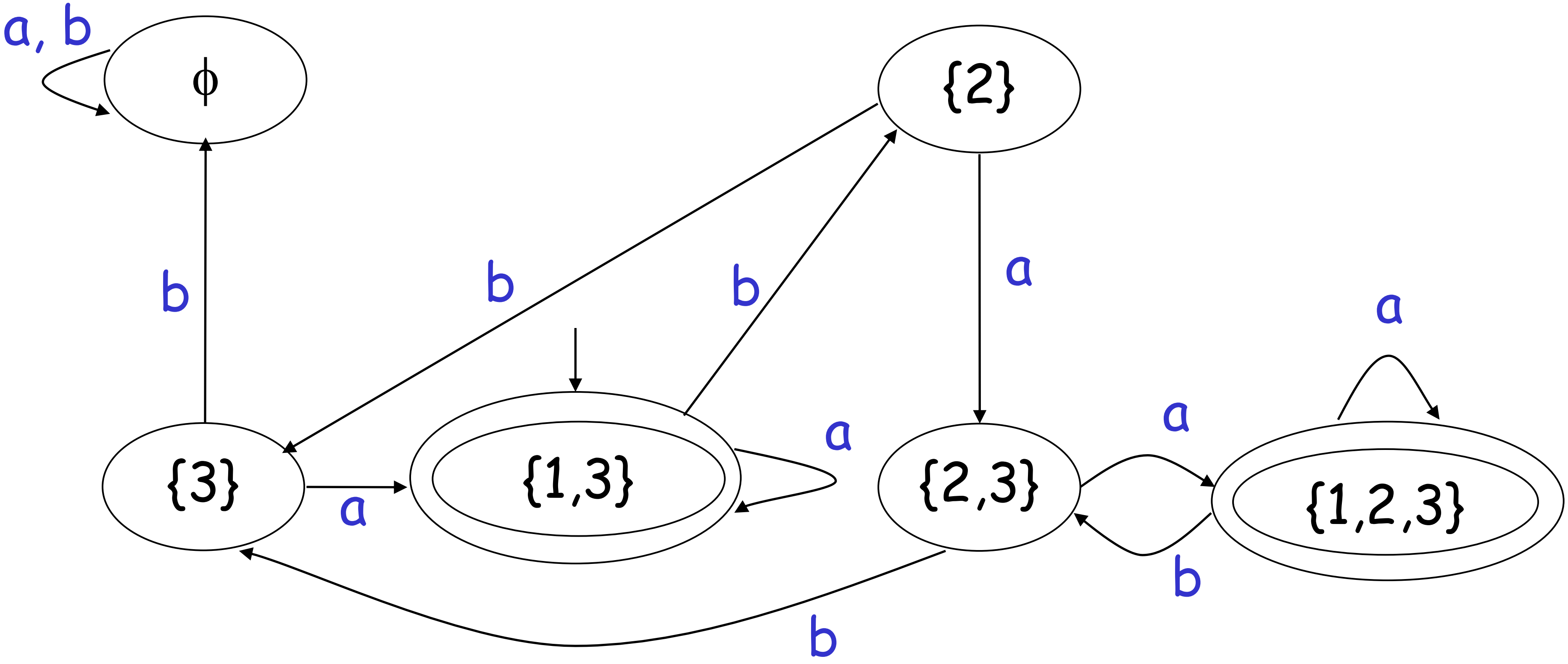
Step2:Determine the start and accept states



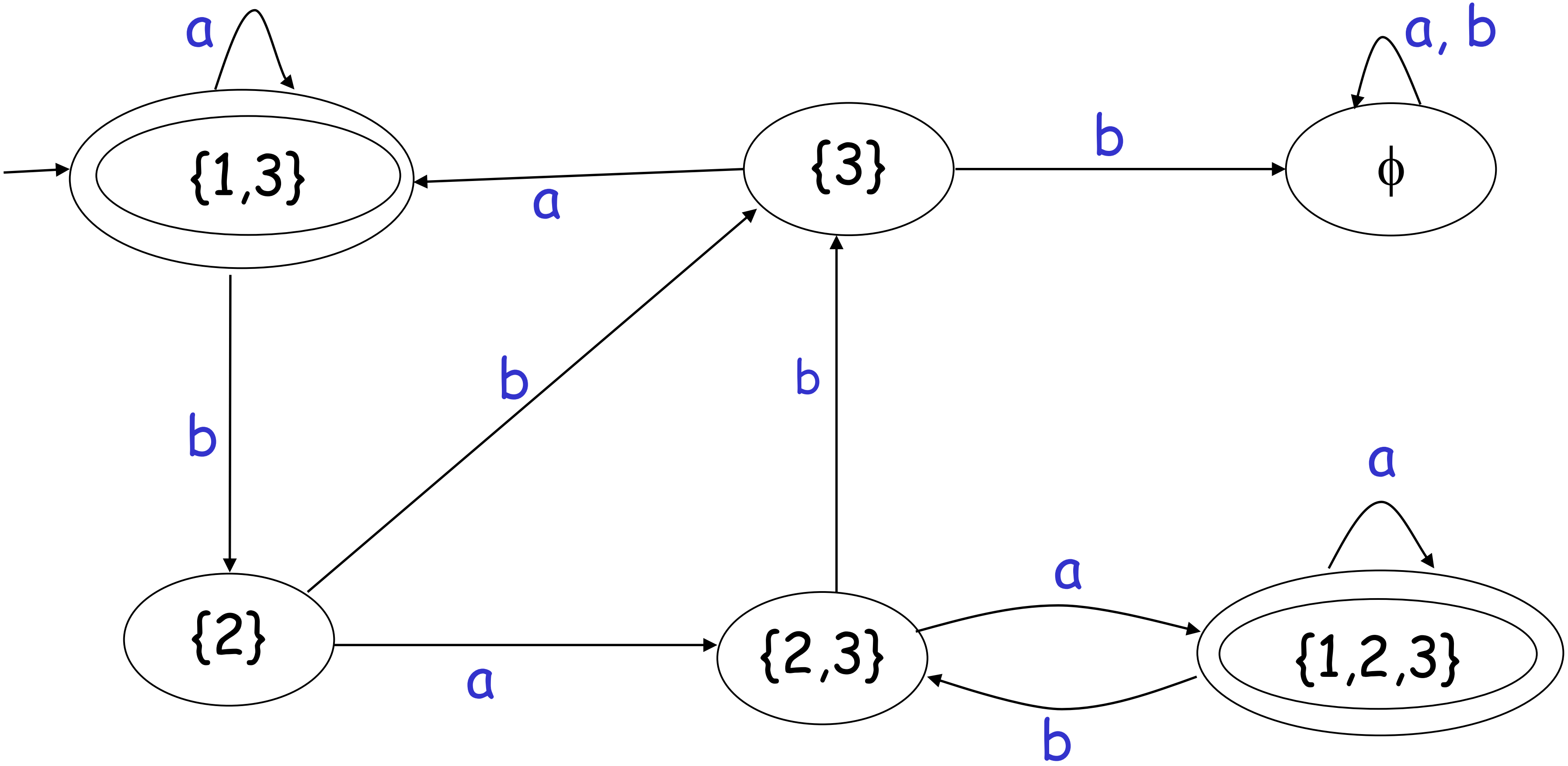
Step3: Determine transition function



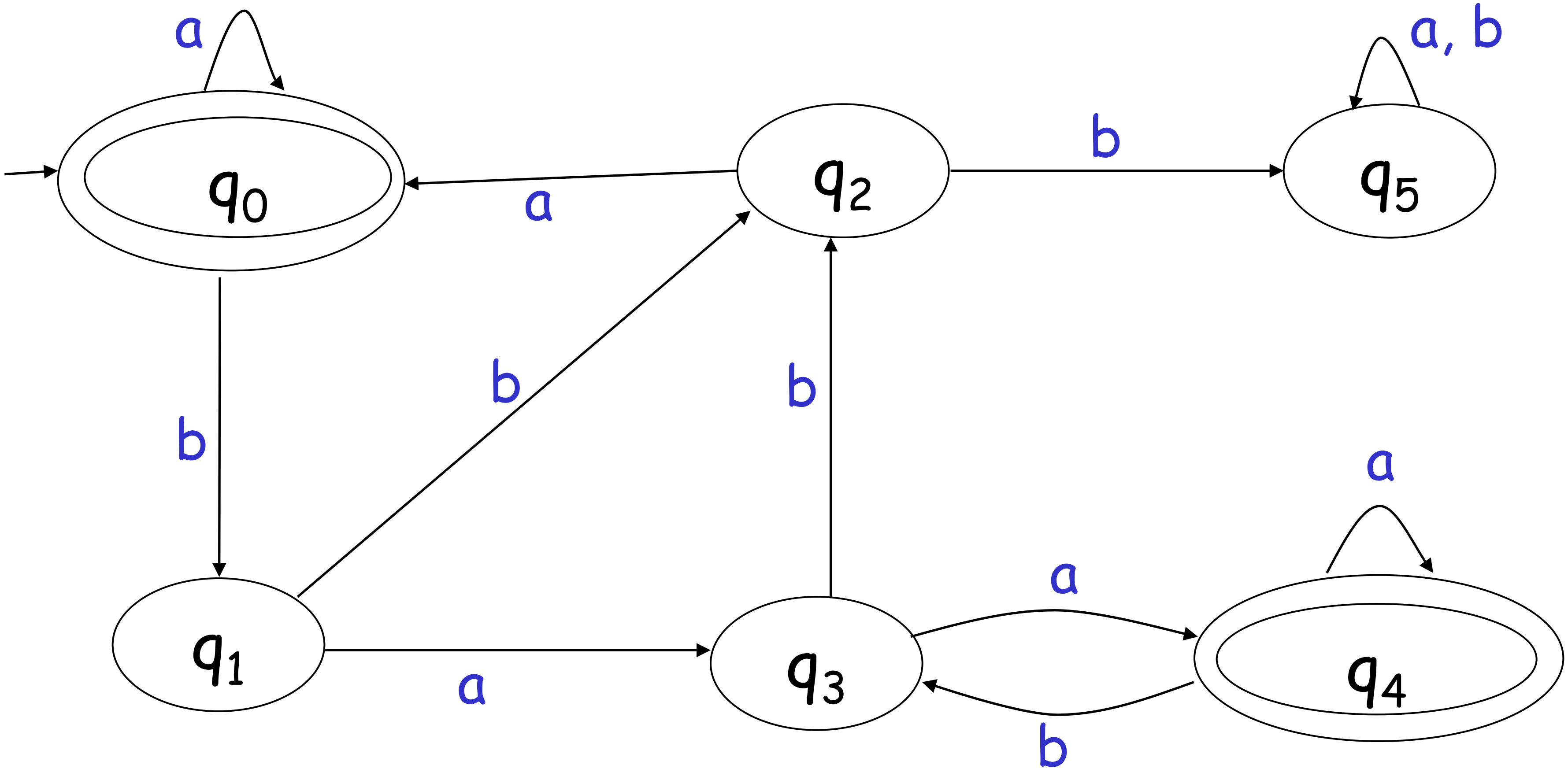
After removing unnecessary states



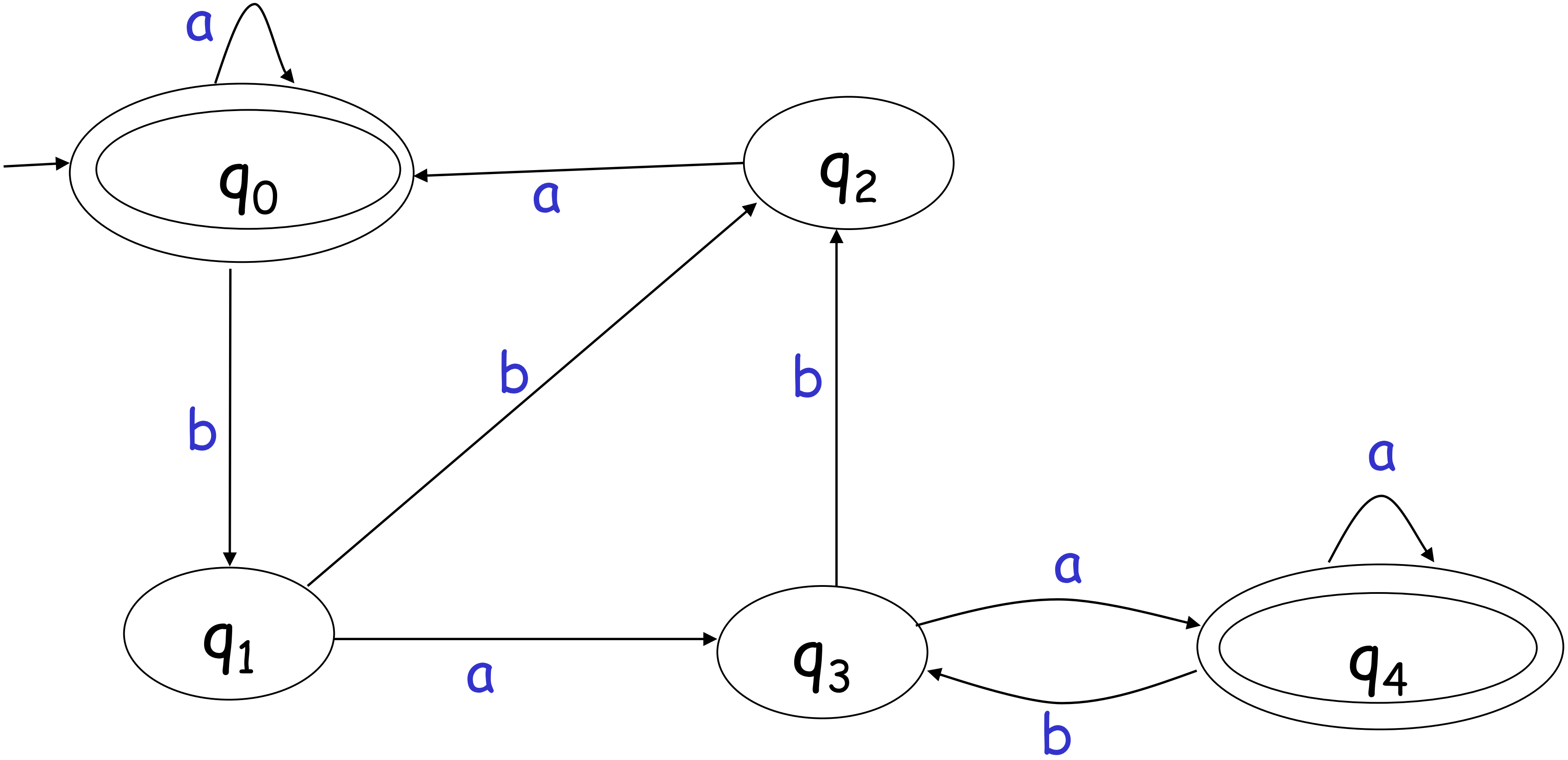
Rearranging states



Renaming states



More simplified



NFA to DFA: Remarks

We are given an NFA M

We want to convert it
to an equivalent DFA M'

With $L(M) = L(M')$

If the NFA has states

$$q_0, q_1, q_2, \ldots$$

the DFA has states in the powerset

$$\emptyset, \{q_0\}, \{q_1\}, \{q_1, q_2\}, \{q_3, q_4, q_7\}, \ldots.$$

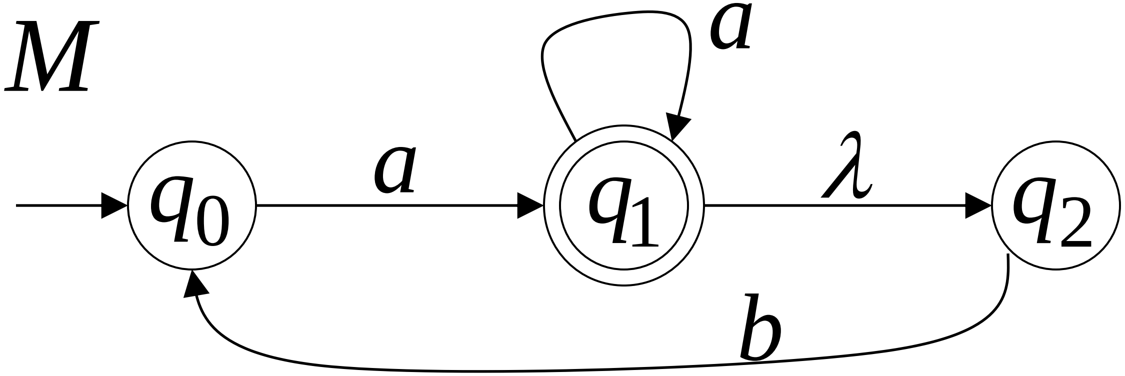
Procedure NFA to DFA

1. Initial state of NFA: q_0

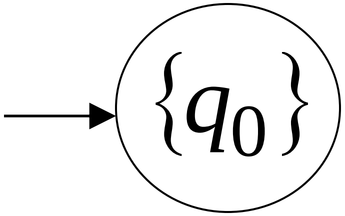
Initial state of DFA: $\{q_0\}$

Example

NFA M



DFA M'



Procedure NFA to DFA

2. For every DFA's state $\{q_i, q_j, \ldots, q_m\}$

Compute in the NFA

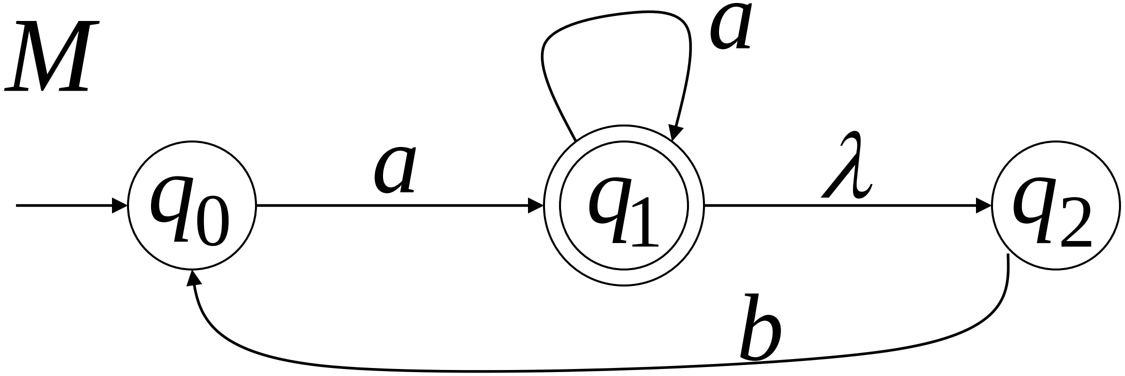
$$\left.\begin{array}{l} \delta^*(q_i, a), \\ \delta^*(q_j, a), \\ \ldots \end{array}\right\} = \{q'_i, q'_j, \ldots, q'_m\}$$

Add transition to DFA

$$\delta(\{q_i, q_j, \ldots, q_m\},\ a) = \{q'_i, q'_j, \ldots, q'_m\}$$

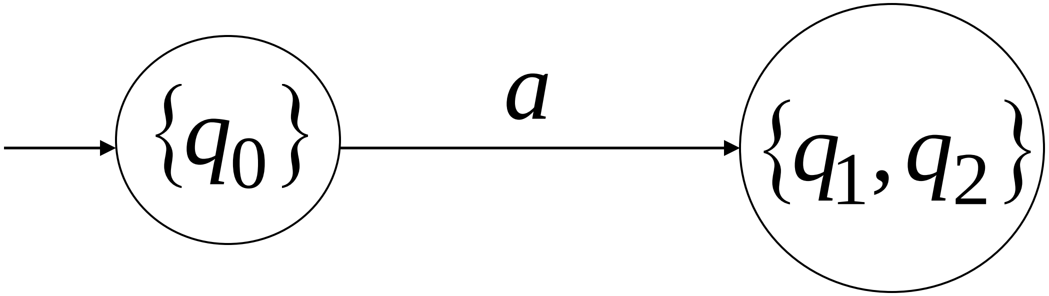
Exampe

NFA M



$$\delta^*(q_0, a) = \{q_1, q_2\}$$

DFA M'



$$\delta(\{q_0\}, a) = \{q_1, q_2\}$$

$$\delta^*(2,b)=\{3\},\ \delta^*(3,b)=\varphi$$

$$\Downarrow$$

$$\delta(\{2,\ 3\},\ b)=\{3\}\cup\varphi=\{3\}$$

Procedure NFA to DFA

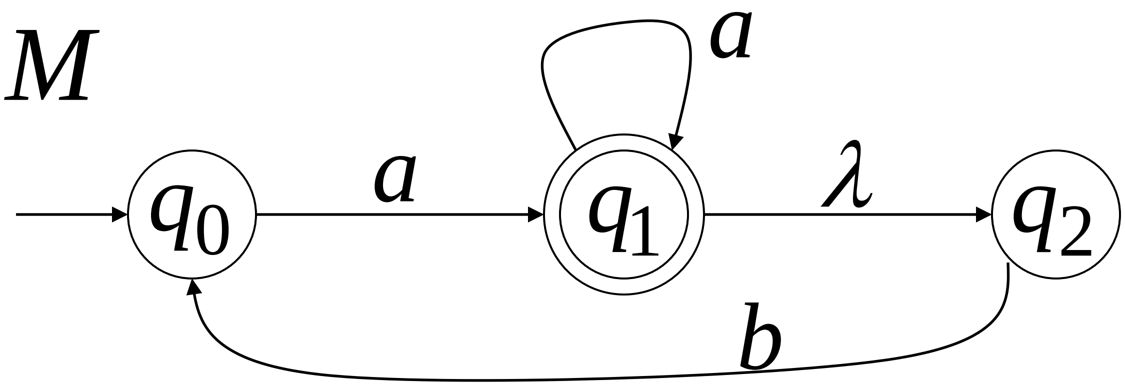
Repeat Step 2 for all letters in alphabet,

until

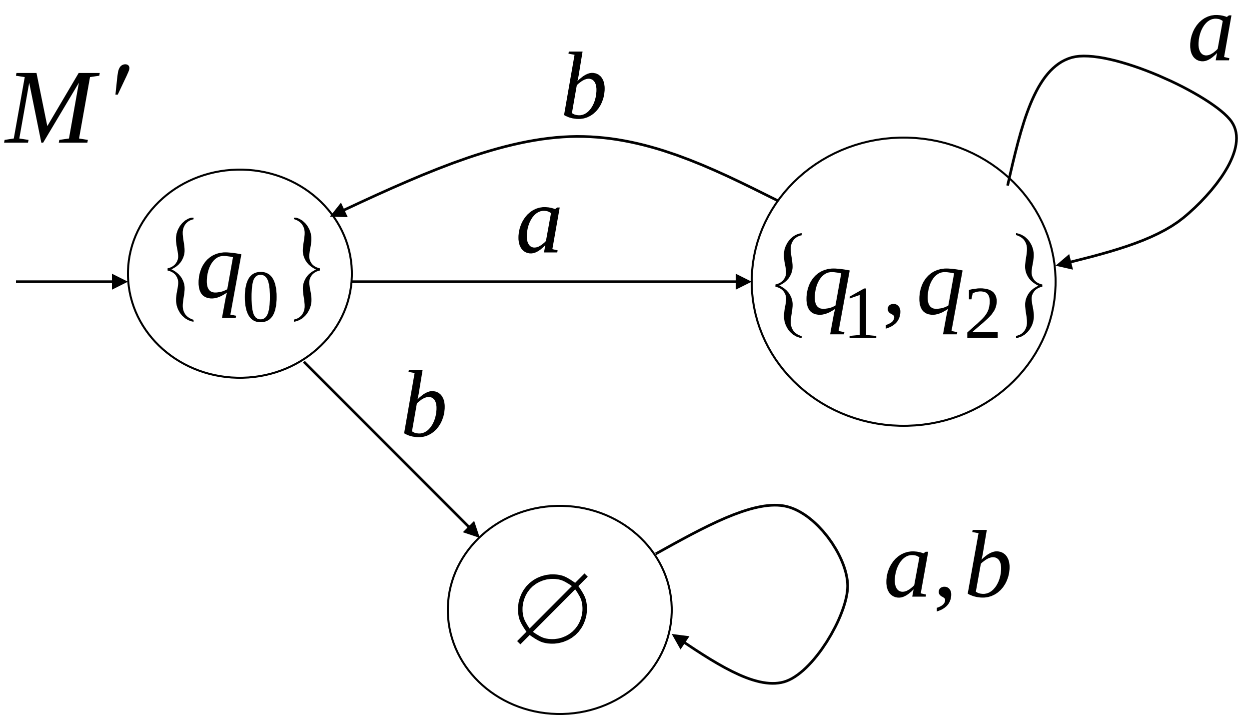
no more transitions can be added.

Example

NFA M



DFA M'



Procedure NFA to DFA

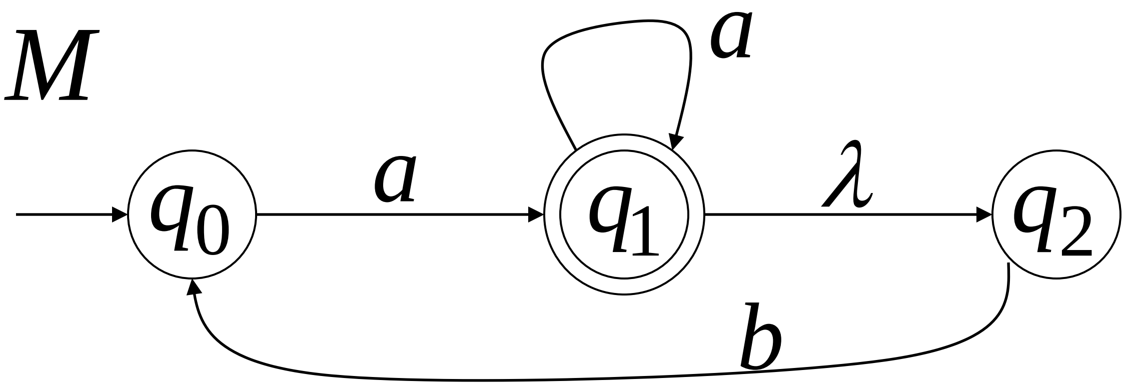
3. For any DFA state $\{q_i, q_j, \ldots, q_m\}$

If some q_j is a final state in the NFA

Then, $\{q_i, q_j, \ldots, q_m\}$
is a final state in the DFA

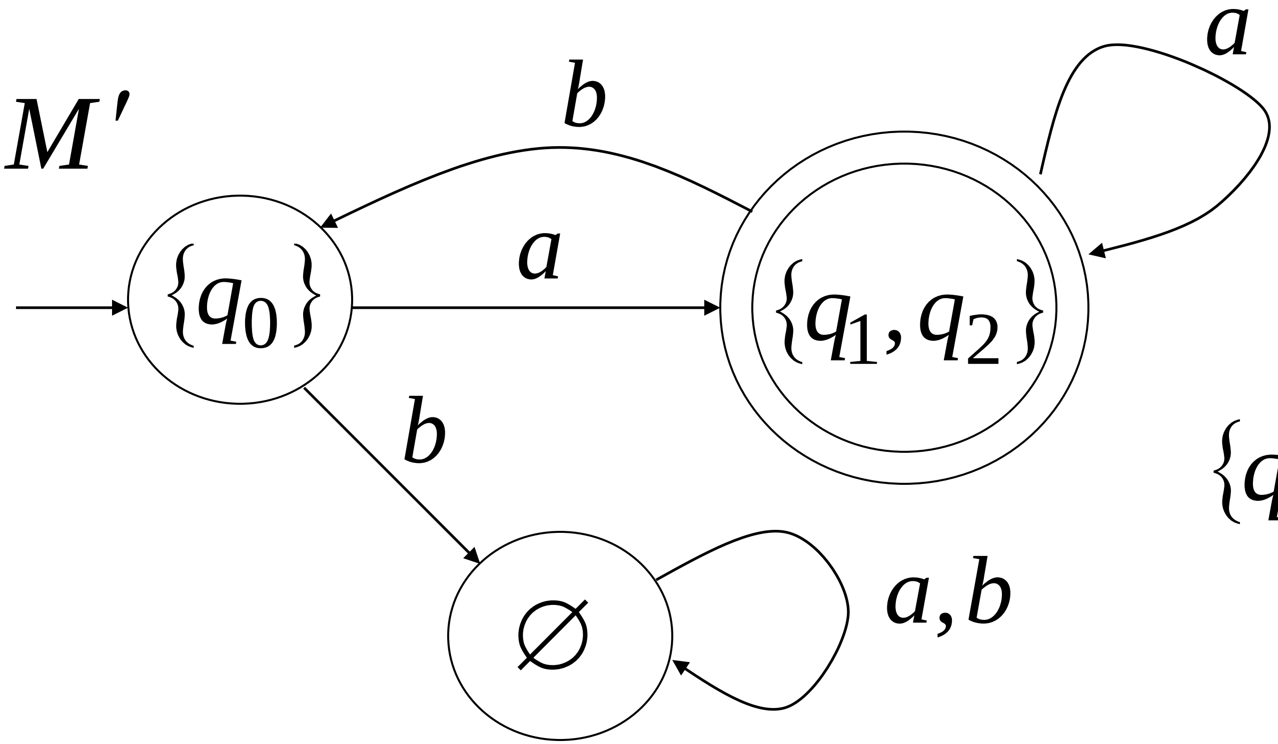
Example

NFA M



$q_1 \in F$

DFA M'



$\{q_1, q_2\} \in F'$

Theorem

Take NFA M

Apply procedure to obtain DFA M'

Then M and M' are equivalent :

$$L(M) = L(M')$$

Proof

$$L(M) = L(M')$$

$$\Updownarrow$$

$$L(M) \subseteq L(M') \quad \text{AND} \quad L(M) \supseteq L(M')$$

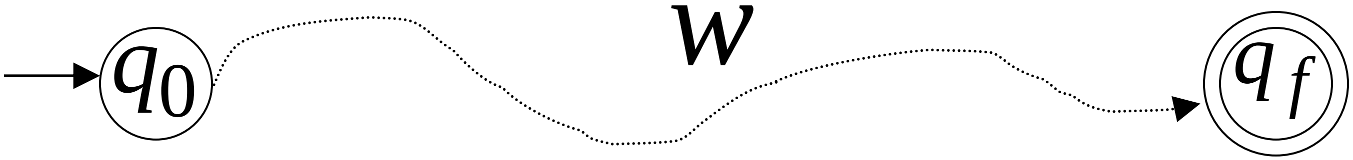
First we show: $L(M) \subseteq L(M')$

Take arbitrary: $w \in L(M)$

We will prove: $w \in L(M')$

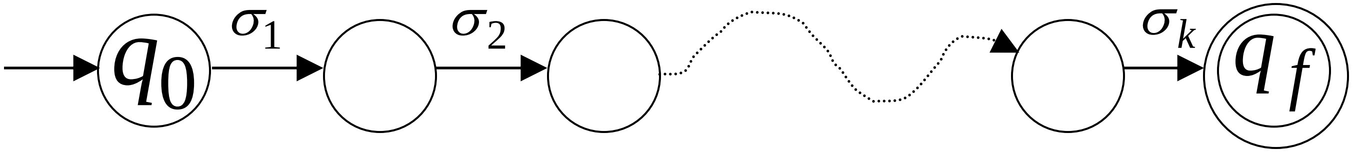
$$w \in L(M)$$

$M:$



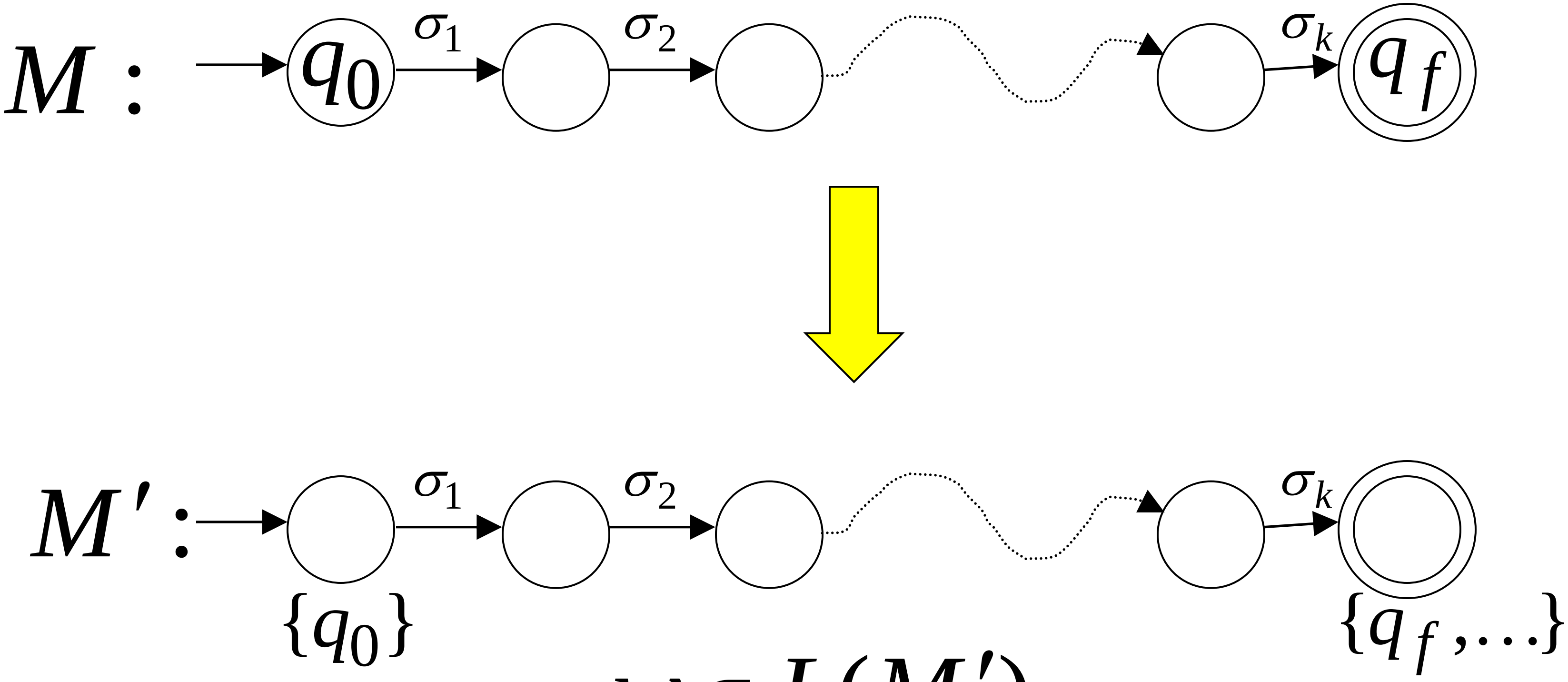
$$w = \sigma_1 \sigma_2 \cdots \sigma_k$$

$M:$



We will show that if $w \in L(M)$

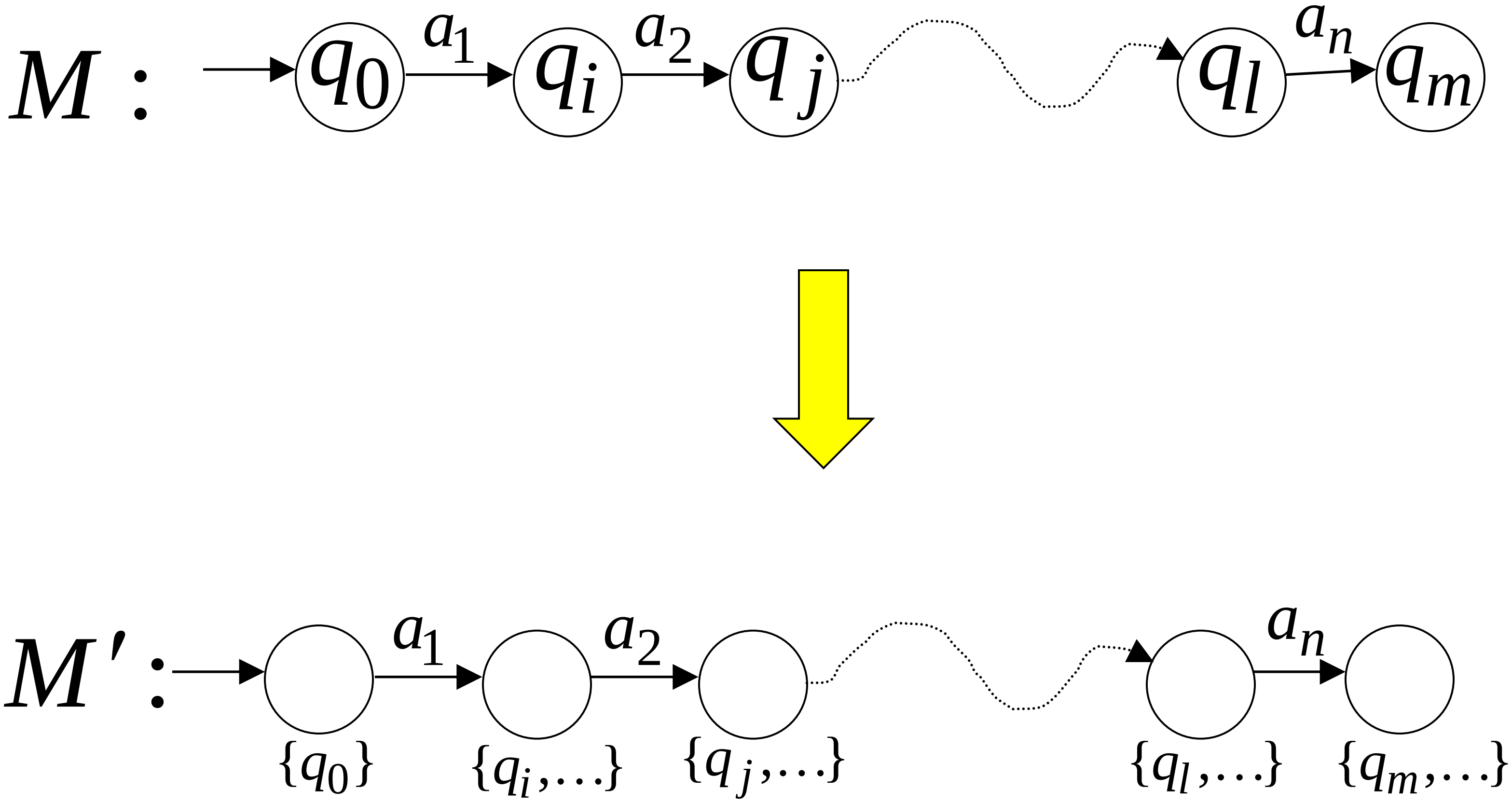
$$w = \sigma_1 \sigma_2 \cdots \sigma_k$$



$$w \in L(M')$$

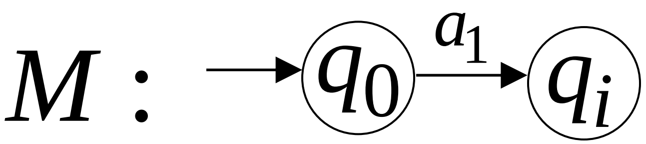
More generally, we will show that if in M:

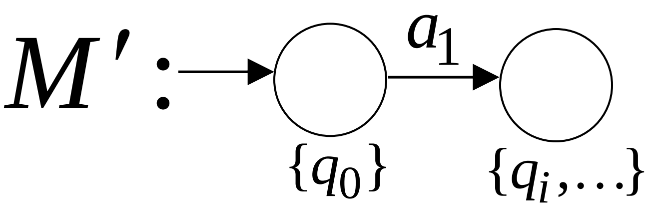
(arbitrary string) $v = a_1 a_2 \cdots a_n$



Proof by induction on $|v|$

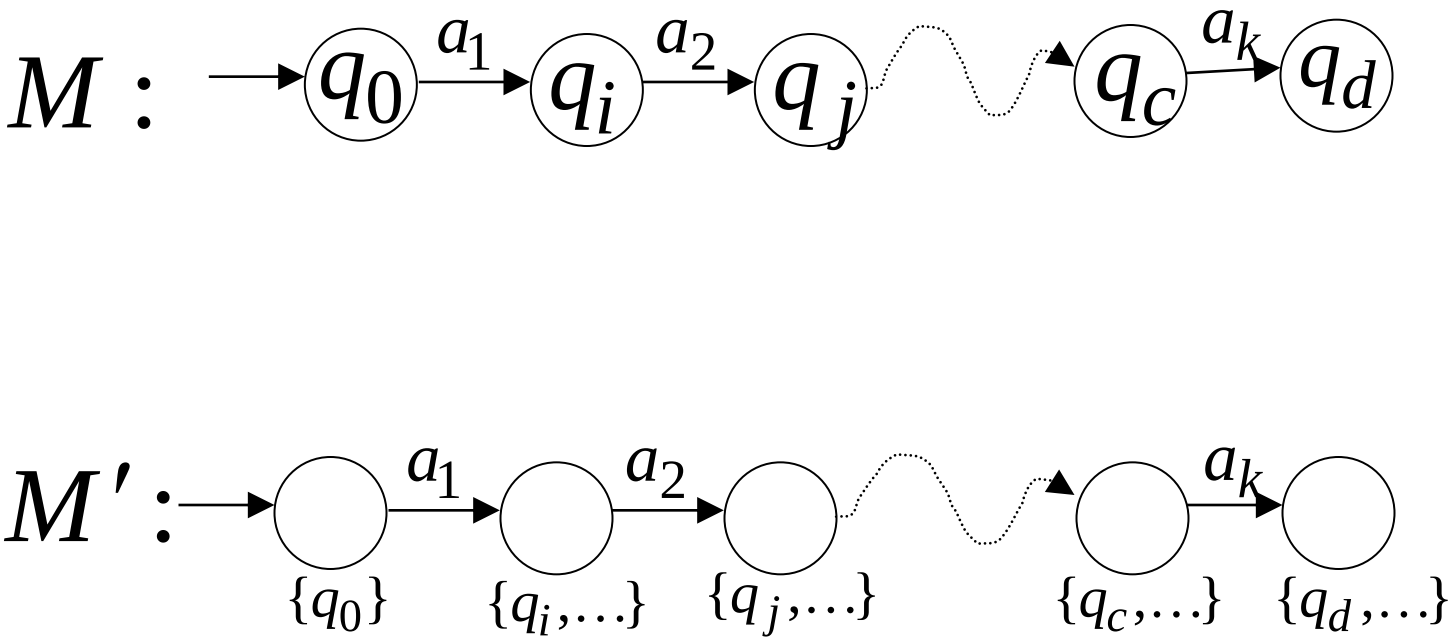
Induction Basis: $v = a_1$





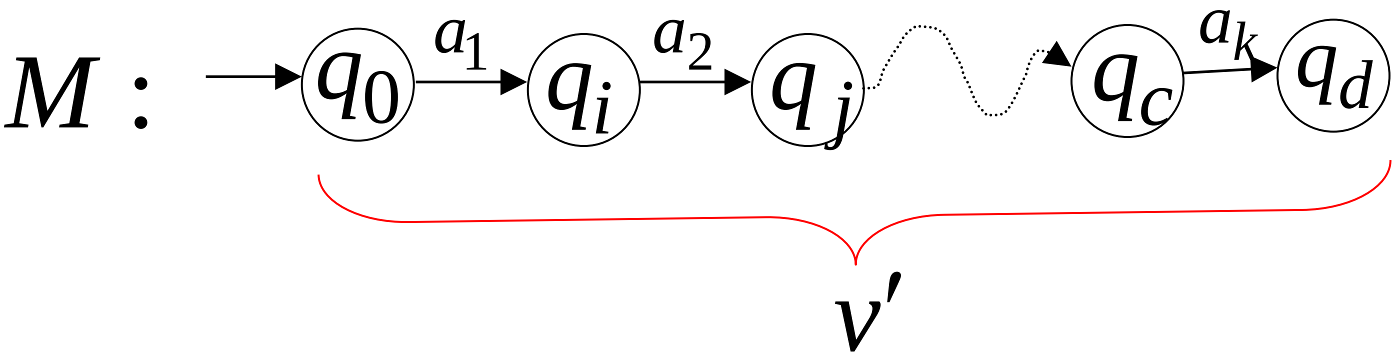
Induction hypothesis: $1 \le |v| \le k$

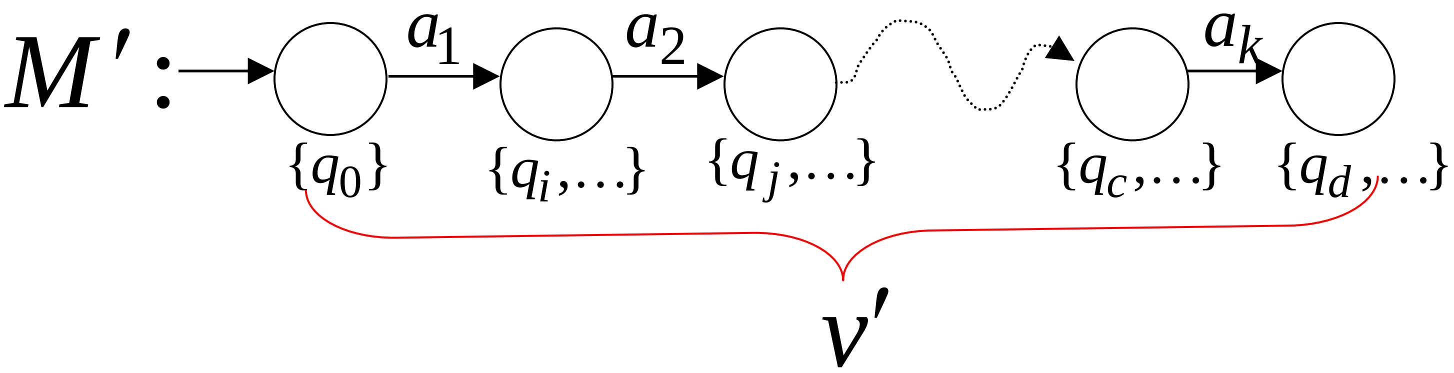
$$v = a_1 a_2 \cdots a_k$$



Induction Step: $|v| = k+1$

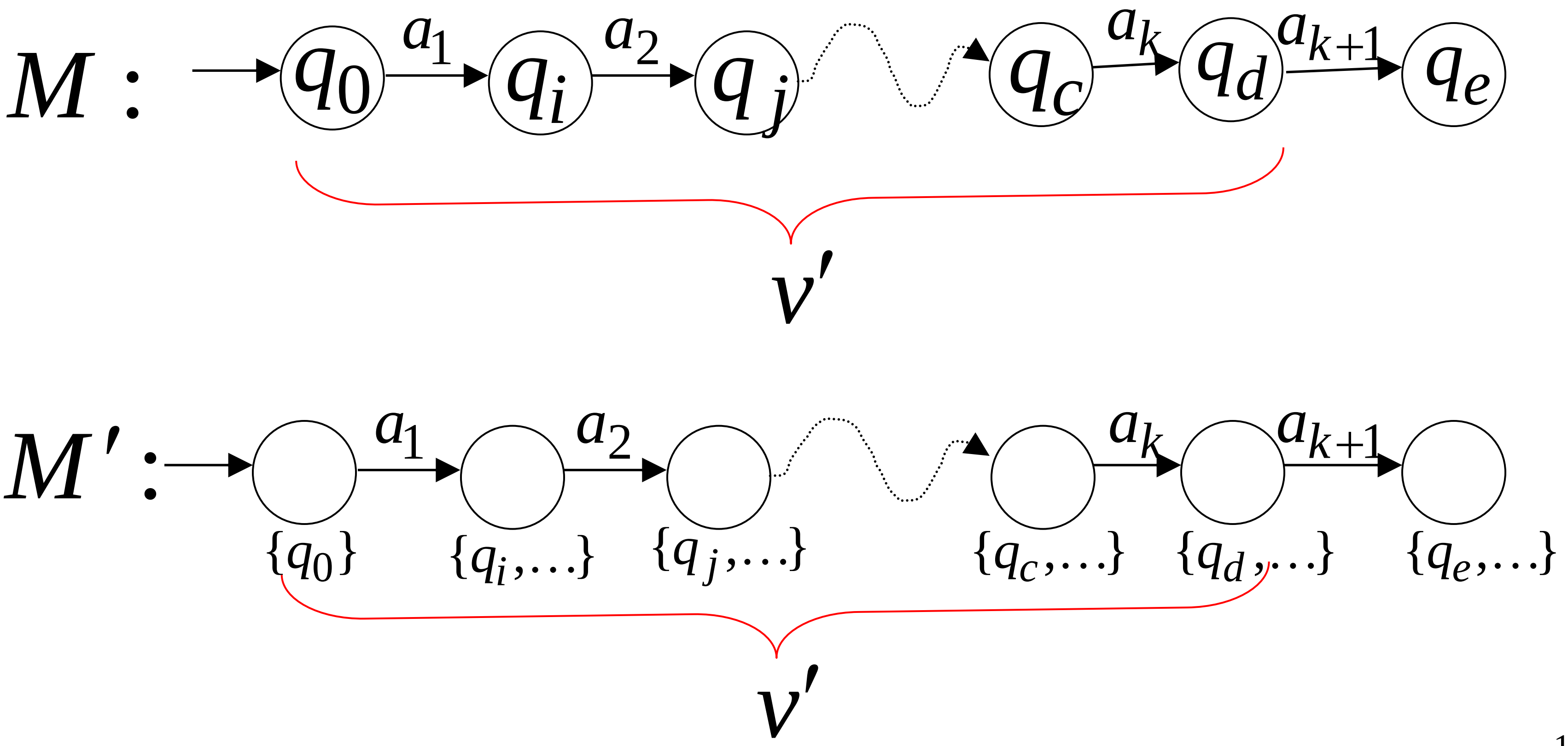
$$v = \underbrace{a_1 a_2 \cdots a_k}_{v'} a_{k+1} = v' a_{k+1}$$





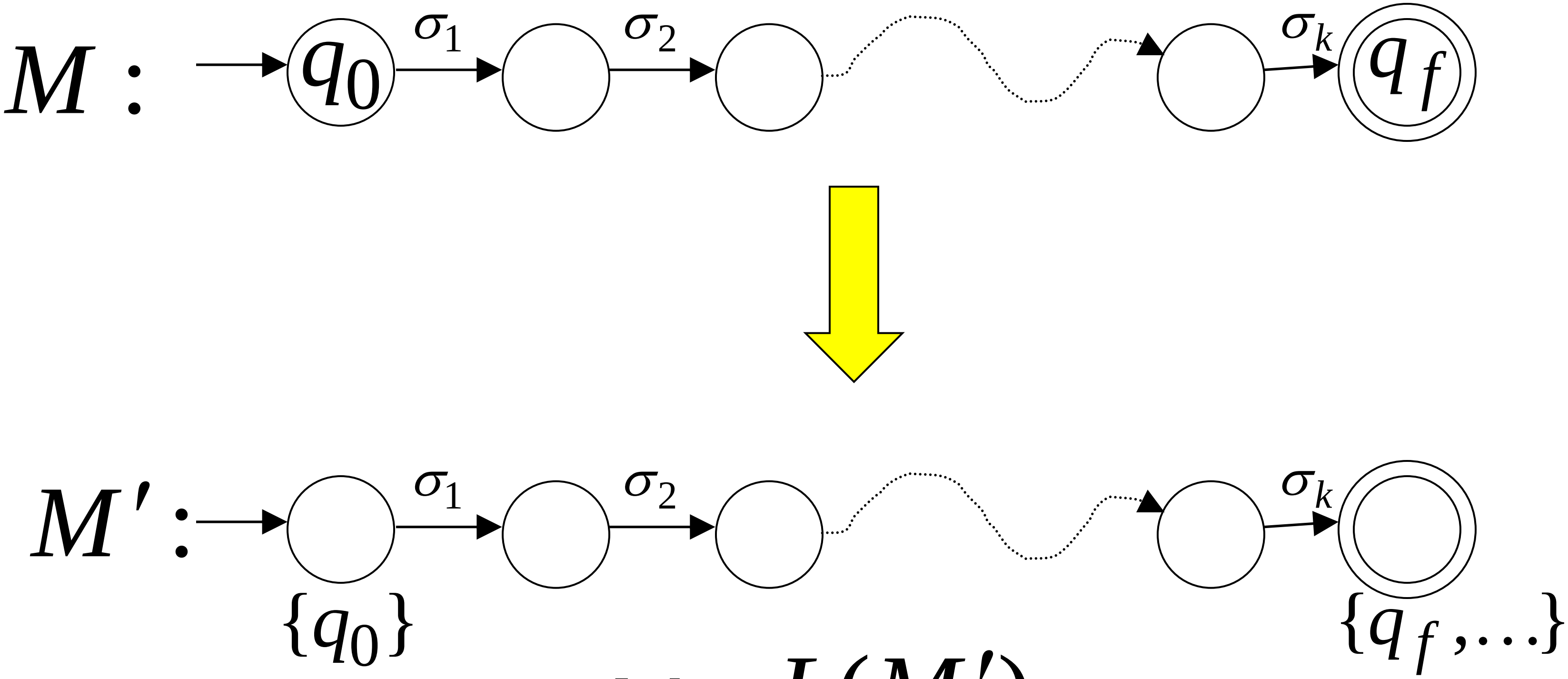
Induction Step: $|v| = k+1$

$$v = \underbrace{a_1 a_2 \cdots a_k}_{v'} a_{k+1} = v' a_{k+1}$$



Therefore if $w \in L(M)$

$$w = \sigma_1 \sigma_2 \cdots \sigma_k$$



$$w \in L(M')$$

We have shown: $L(M) \subseteq L(M')$

We also need to show: $L(M) \supseteq L(M')$

(proof is similar)

The End

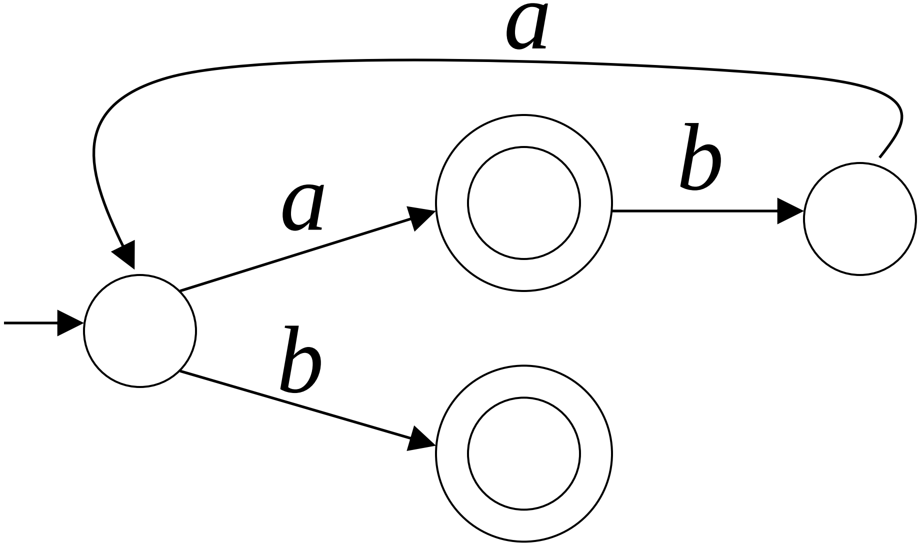
Single Final State for NFAs

Any NFA can be converted

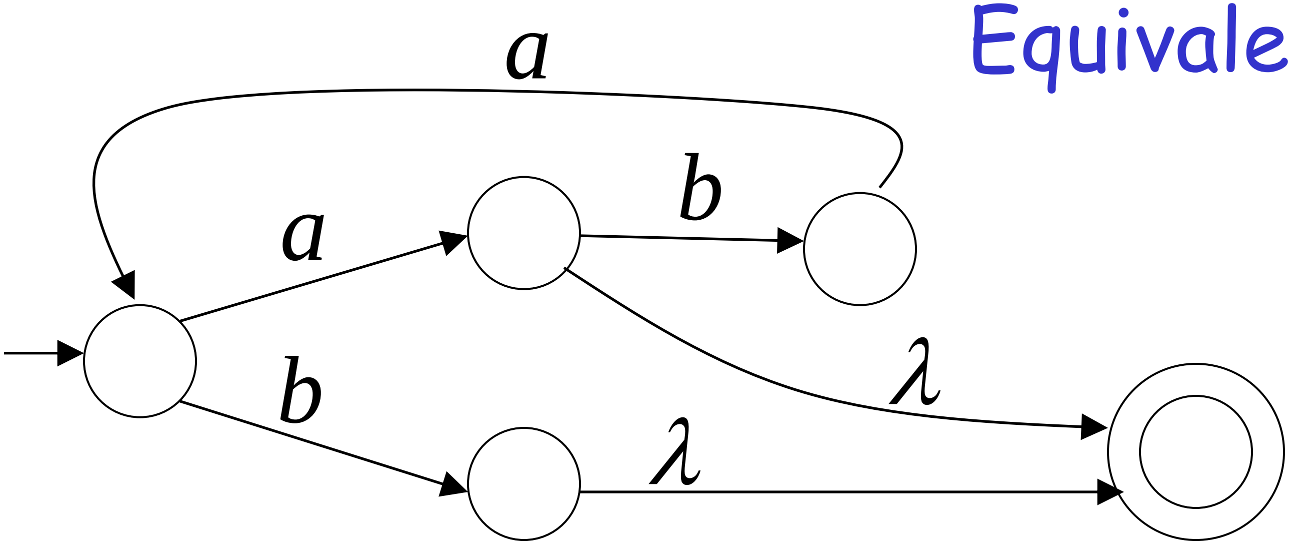
to an equivalent NFA

with a single final state

Example



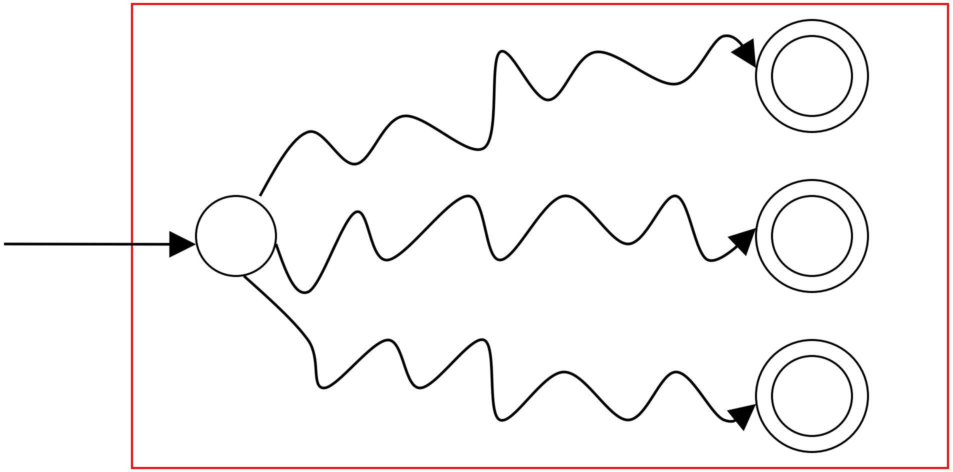
NFA



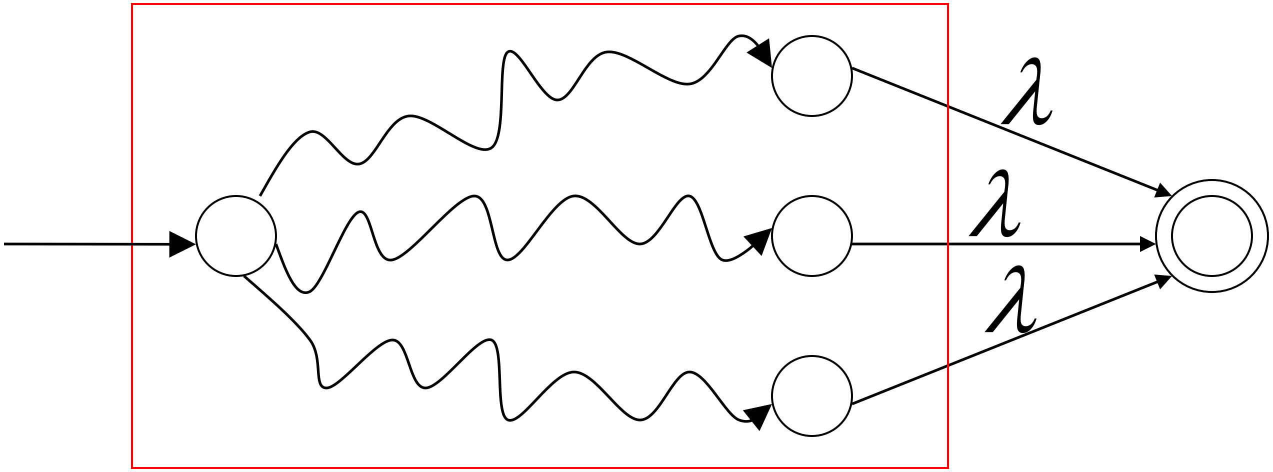
Equivalent NFA

In General

NFA



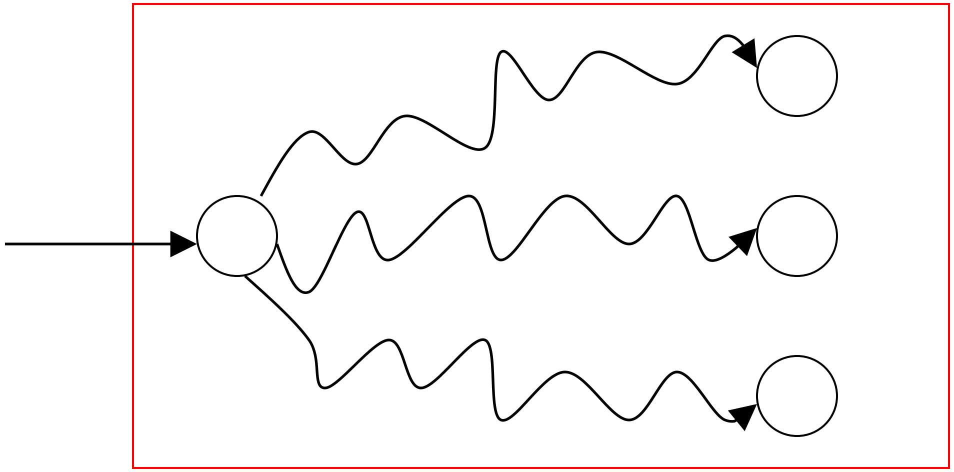
Equivalent NFA

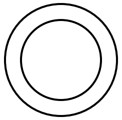


Single final state

Extreme Case

NFA without final state





Add a final state

Without transitions

Properties of Regular Languages

For regular languages L_1 and L_2
we will prove that:

Union: $L_1 \cup L_2$

Concatenation: $L_1 L_2$

Star: $L_1{}^*$

Reversal: $L_1{}^R$

Complement: $\overline{L_1}$

Intersection: $L_1 \cap L_2$

Are regular Languages

We say: Regular languages are **closed under**

Union: $L_1 \cup L_2$

Concatenation: $L_1 L_2$

Star: $L_1 *$

Reversal: L_1^R

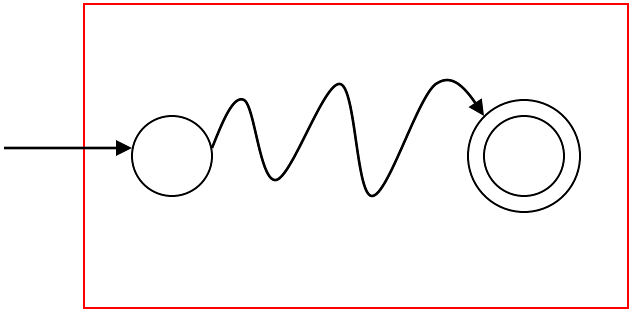
Complement: $\overline{L_1}$

Intersection: $L_1 \cap L_2$

Regular language L_1

$$L(M_1) = L_1$$

NFA M_1

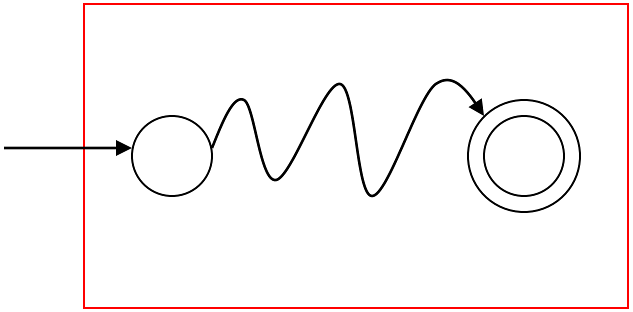


Single final state

Regular language L_2

$$L(M_2) = L_2$$

NFA M_2



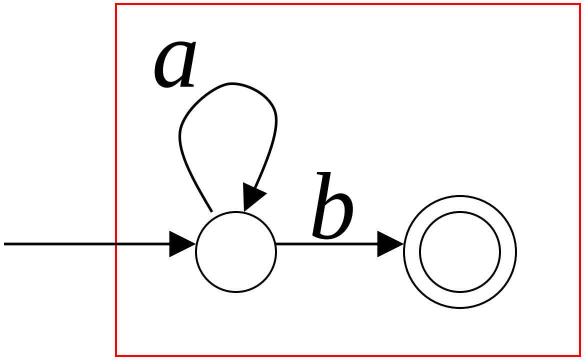
Single final state

Example

$n \geq 0$

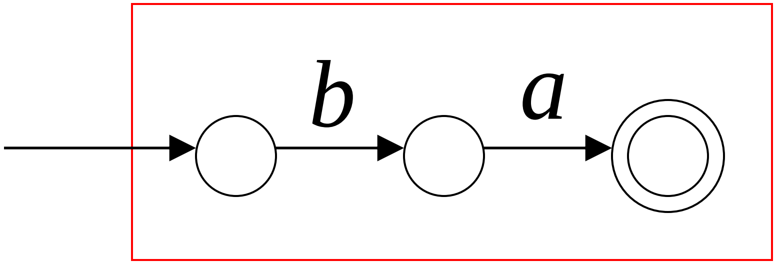
$L_1 = \{a^n b\}$

M_1



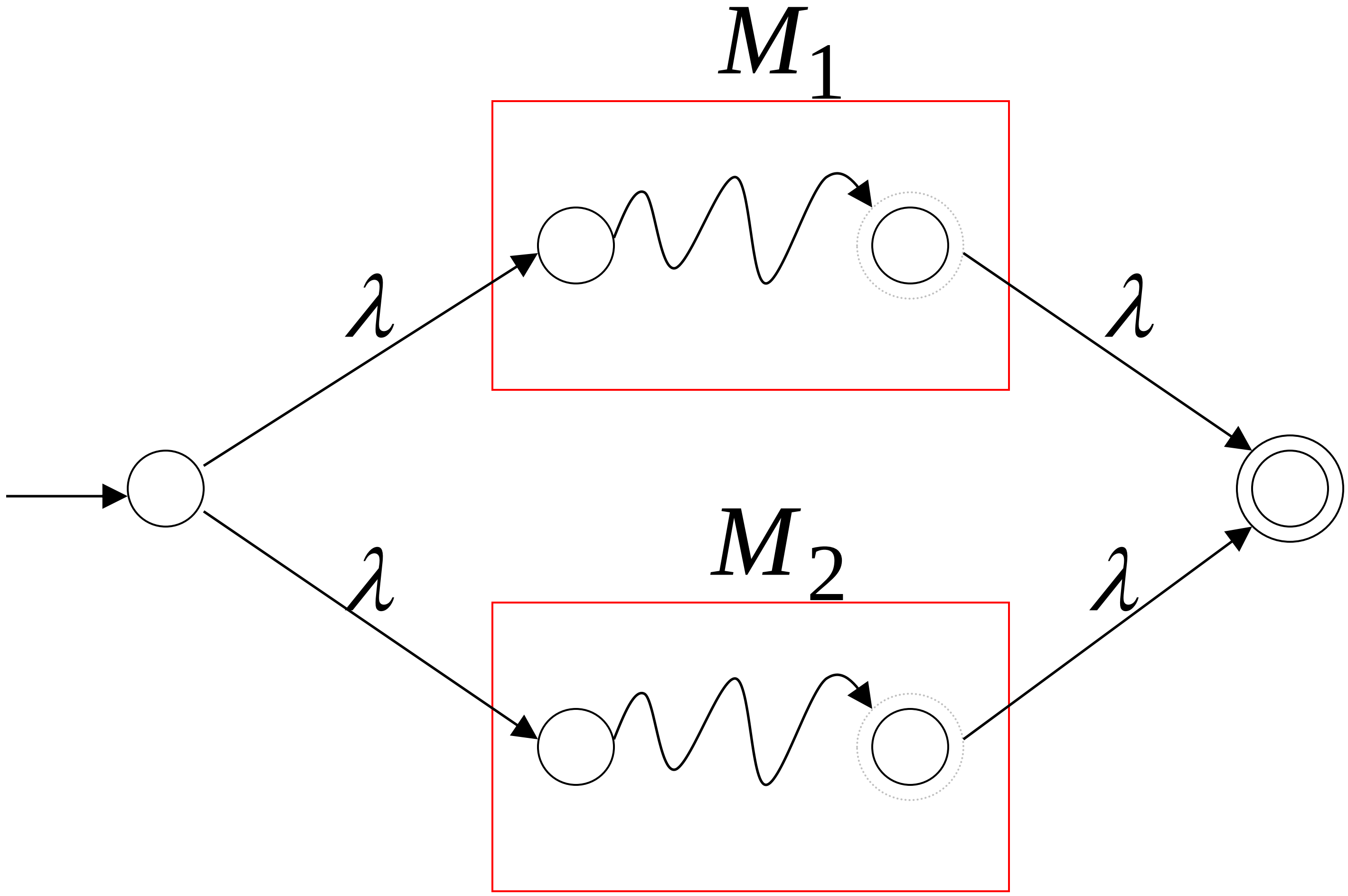
$L_2 = \{ba\}$

M_2



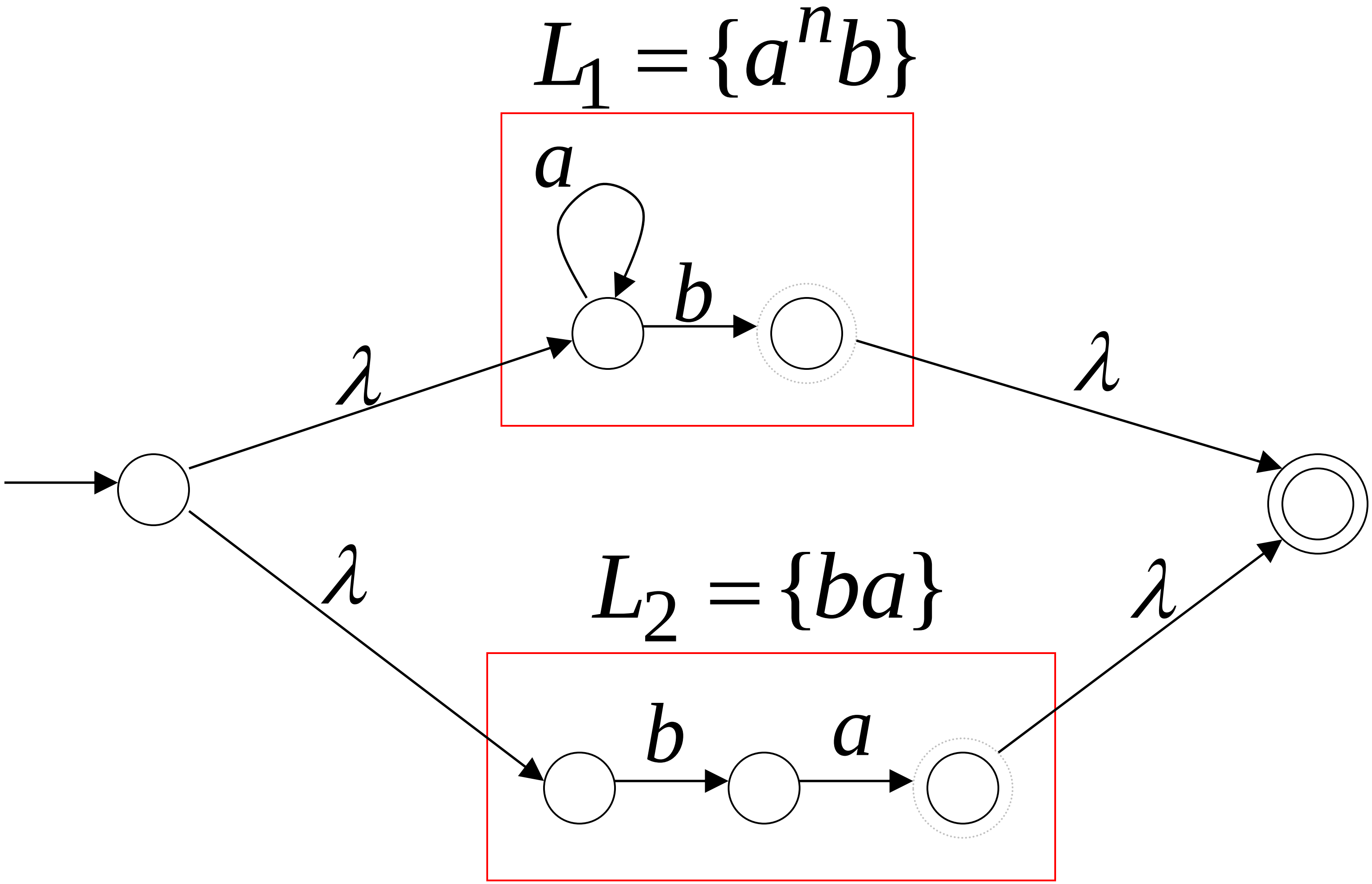
Union

NFA for $L_1 \cup L_2$



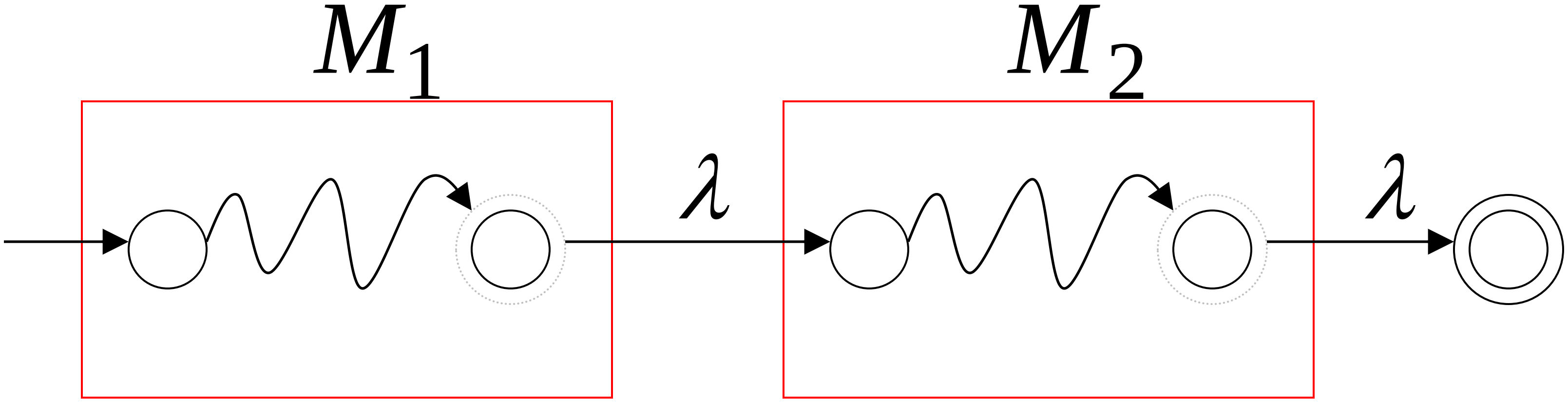
Example

NFA for $L_1 \cup L_2 = \{a^n b\} \cup \{ba\}$



Concatenation

NFA for L_1L_2

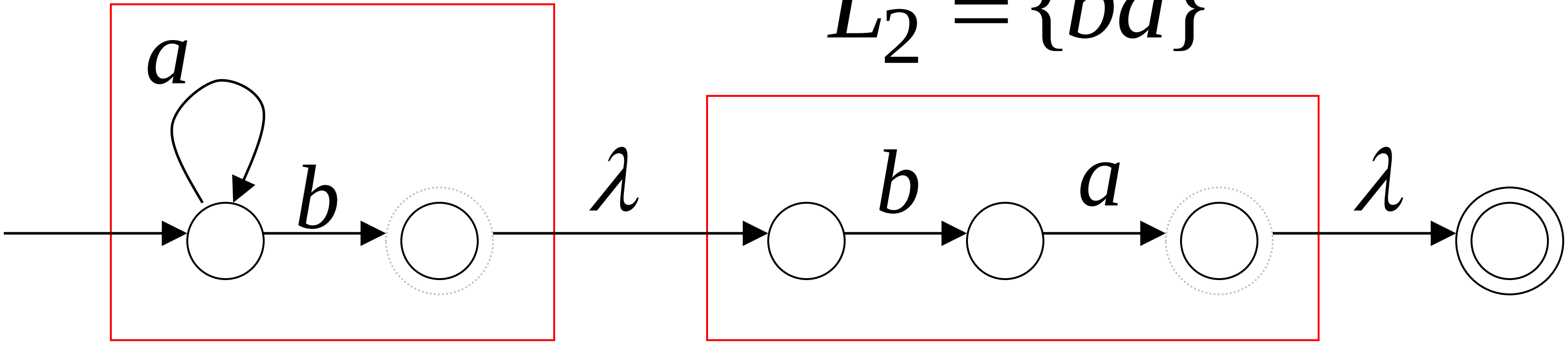


Example

NFA for $L_1 L_2 = \{a^n b\}\{ba\} = \{a^n bba\}$

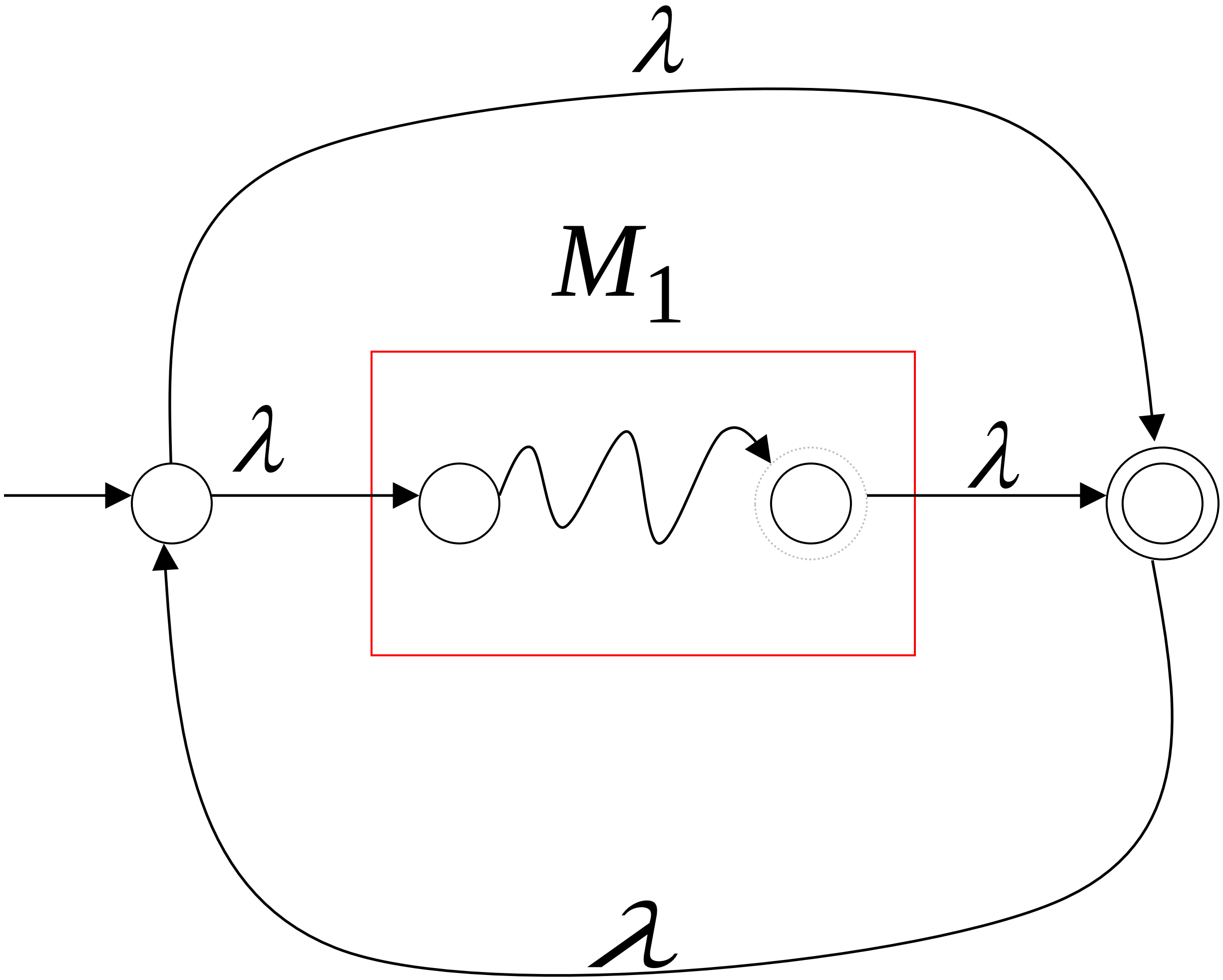
$L_1 = \{a^n b\}$

$L_2 = \{ba\}$



Star Operation

NFA for $L_1{}^*$



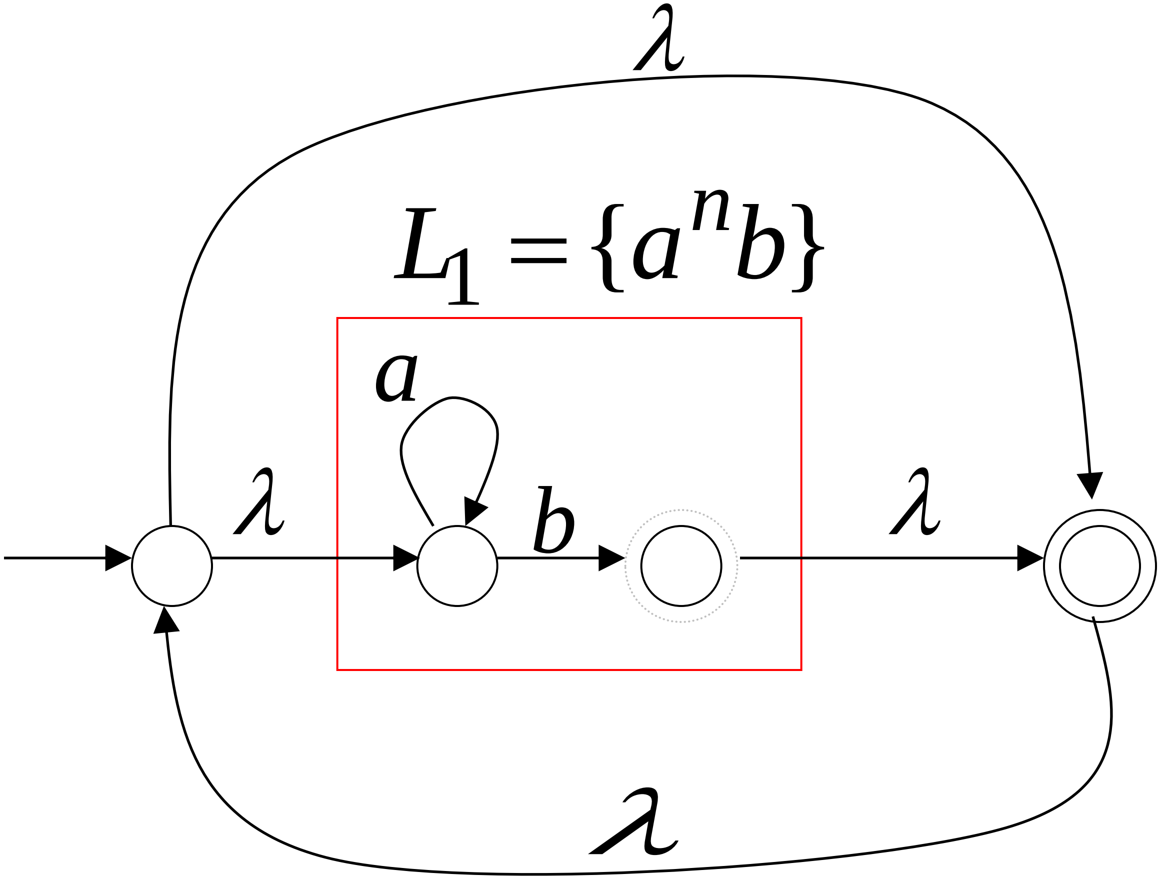
$\lambda \in L_1{}^*$

Example

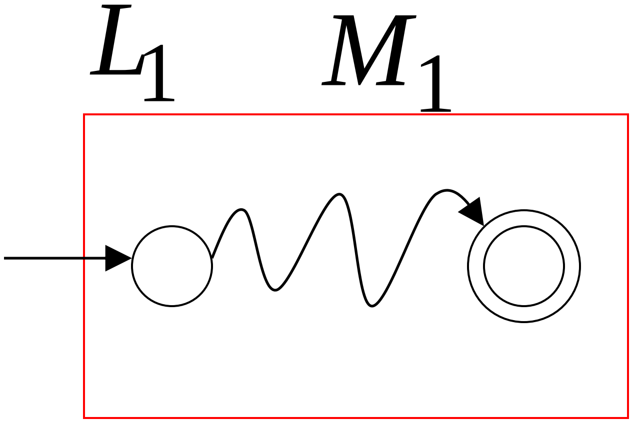
NFA for $L_1{}^* = \{a^n b\}^*$

$w = w_1 w_2 \cdots w_k$

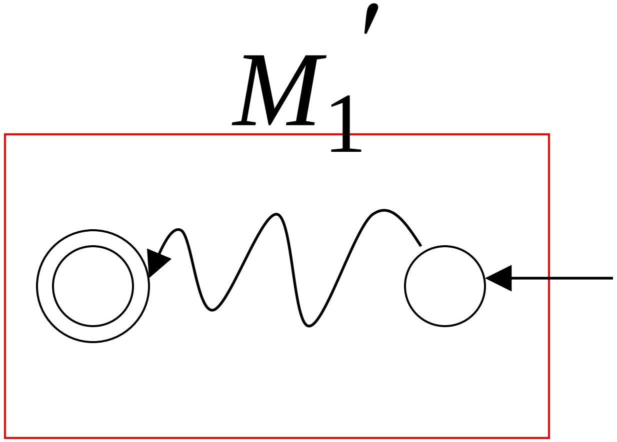
$w_i \in L_1$



Reverse



NFA for L_1^R



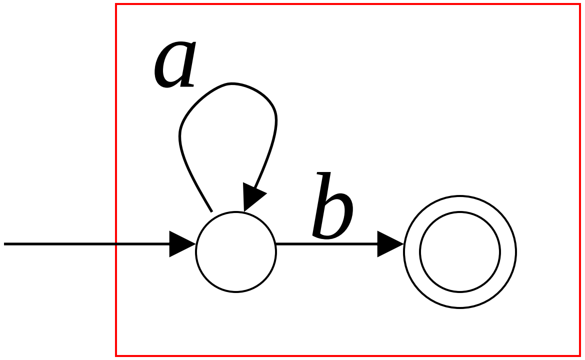
1. Reverse all transitions

2. Make initial state final state
   and vice versa

Example

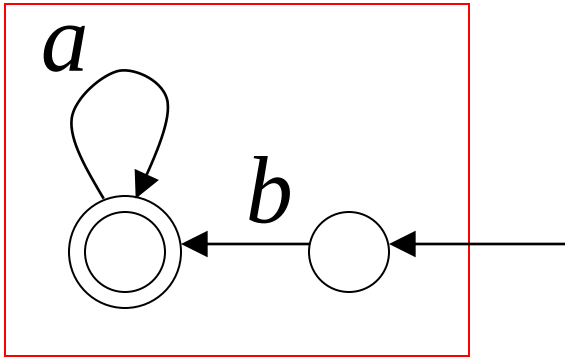
M_1

$L_1 = \{a^n b\}$

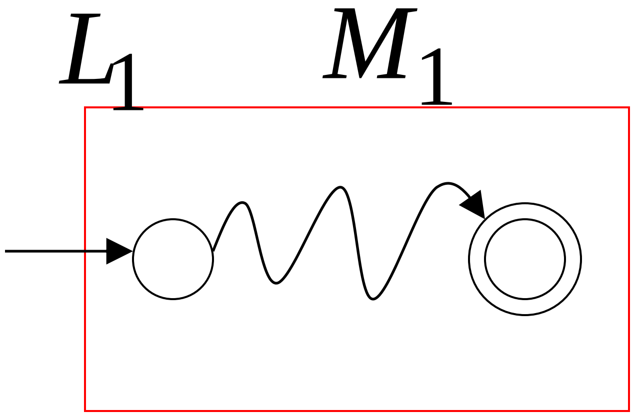


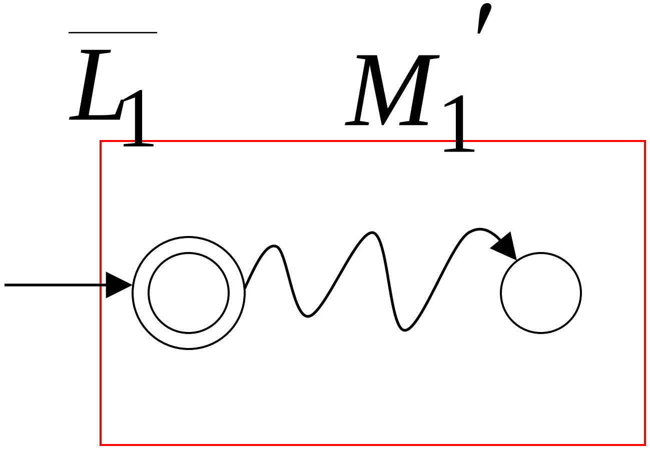
M_1'

$L_1^R = \{ba^n\}$



Complement





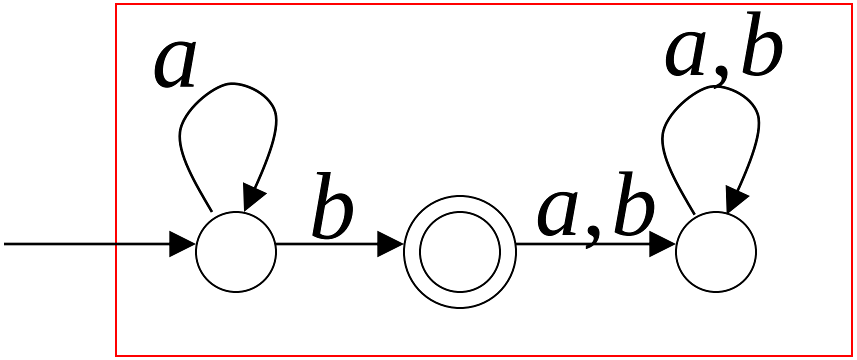
1. Take the **DFA** that accepts L_1

2. Make final states non-final,
   and vice-versa

Example

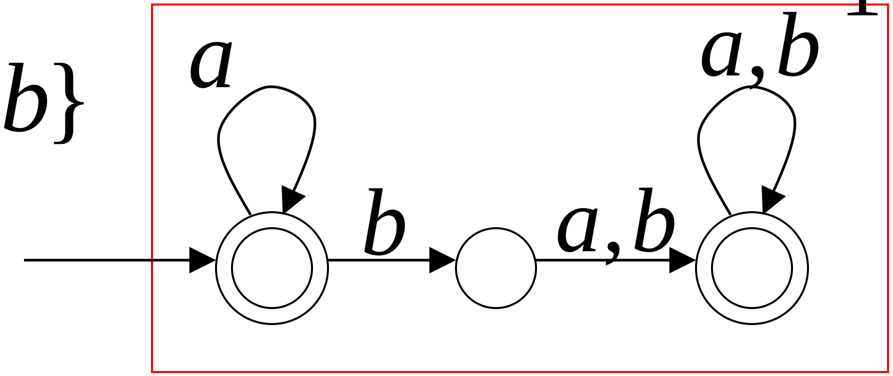
M_1

$$L_1 = \{a^n b\}$$



M_1'

$$\overline{L_1} = \{a,b\}^* - \{a^n b\}$$



Intersection

DeMorgan's Law: $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$

$L_1\,,\ L_2$ regular

➡ $\overline{L_1}\,,\ \overline{L_2}$ regular

➡ $\overline{L_1} \cup \overline{L_2}$ regular

➡ $\overline{\overline{L_1} \cup \overline{L_2}}$ regular

➡ $L_1 \cap L_2$ regular

Example

$$\left.\begin{array}{ll} L_1=\{a^nb\} & \text{regular} \\[2em] L_2=\{ab,ba\} & \text{regular} \end{array}\right\} \Rightarrow \begin{array}{c} L_1\cap L_2=\{ab\} \\[1em] \text{regular} \end{array}$$

Regular Expressions

Regular Expressions

Regular expressions
describe regular languages

Example: $(a+b \cdot c)*$

describes the language

$$\{a,bc\}* = \{\lambda, a, bc, aa, abc, bca, ...\}$$

Recursive Definition

Primitive regular expressions: $\emptyset, \quad \lambda, \quad \alpha$

Given regular expressions r_1 and r_2

$$\left.\begin{array}{l} r_1 + r_2 \\ r_1 \cdot r_2 \\ r_1 * \\ (r_1) \end{array}\right\} \text{Are regular expressions}$$

Examples

A regular expression: $(a+b\cdot c)^*\cdot(c+\emptyset)$

Not a regular expression: $(a+b+)$

Languages of Regular Expressions

$L(r)$: language of regular expression r

Example

$$L((a+b\cdot c)^*)=\{\lambda,a,bc,aa,abc,bca,...\}$$

Definition

For primitive regular expressions:

$$L(\varnothing)=\varnothing$$

$$L(\lambda)=\{\lambda\}$$

$$L(a)=\{a\}$$

Definition (continued)

For regular expressions r_1 and r_2

$$L(r_1 + r_2) = L(r_1) \cup L(r_2)$$

$$L(r_1 \cdot r_2) = L(r_1)\, L(r_2)$$

$$L(r_1 *) = (L(r_1))*$$

$$L((r_1)) = L(r_1)$$

Example

Regular expression: $(a+b)\cdot a*$

$$
\begin{aligned}
L((a+b)\cdot a*) &= L((a+b))\,L(a*) \\
&= L(a+b)\,L(a*) \\
&= (L(a)\cup L(b))\,(L(a))* \\
&= (\{a\}\cup\{b\})\,(\{a\})* \\
&= \{a,b\}\,\{\lambda,a,aa,aaa,\ldots\} \\
&= \{a,aa,aaa,\ldots,b,ba,baa,\ldots\}
\end{aligned}
$$

Example

Regular expression $r = (a + b)^*(a + bb)$

$$L(r) = \{a, bb, aa, abb, ba, bbb, ...\}$$

Example

Regular expression $r = (aa)^*(bb)^*b$

$$L(r) = \{a^{2n}b^{2m}b: \quad n, m \geq 0\}$$

Example

Regular expression $r = (0+1)^*\,00\,(0+1)^*$

$L(r)$ = { all strings with at least two consecutive 0 }

Example

Regular expression $r = (1 + 01)^*(0 + \lambda)$

$L(r)$ = { all strings without two consecutive 0 }

Equivalent Regular Expressions

Definition:

Regular expressions r_1 and r_2

are **equivalent** if $L(r_1) = L(r_2)$

Example

L = { all strings without
two consecutive 0 }

$$r_1 = (1 + 01)^* (0 + \lambda)$$

$$r_2 = (1^* 011^*)^* (0 + \lambda) + 1^* (0 + \lambda)$$

$L(r_1) = L(r_2) = L$ ⇒ r_1 and r_2 are equivalent regular expr.

Regular Expressions
and
Regular Languages

Theorem

$$\left\{\begin{array}{l}\text{Languages}\\ \text{Generated by}\\ \text{Regular Expressions}\end{array}\right\} = \left\{\begin{array}{l}\text{Regular}\\ \text{Languages}\end{array}\right\}$$

Theorem - Part 1

$$\left\{\begin{array}{l}\text{Languages}\\ \text{Generated by}\\ \text{Regular Expressions}\end{array}\right\} \subseteq \left\{\begin{array}{l}\text{Regular}\\ \text{Languages}\end{array}\right\}$$

1. For any regular expression r
   the language $L(r)$ is regular

Theorem - Part 2

$$\left\{ \begin{array}{l} \text{Languages} \\ \text{Generated by} \\ \text{Regular Expressions} \end{array} \right\} \supseteq \left\{ \begin{array}{l} \text{Regular} \\ \text{Languages} \end{array} \right\}$$

2. For any regular language L there is a regular expression r with $L(r) = L$

Proof - Part 1

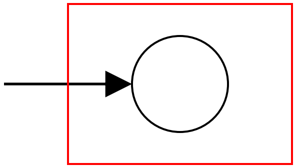
1. For any regular expression r the language $L(r)$ is regular

Proof by induction on the size of r

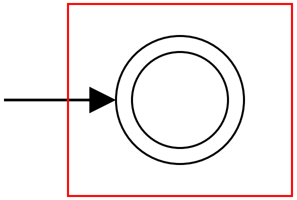
Induction Basis

Primitive Regular Expressions: $\emptyset, \quad \lambda, \quad \alpha$

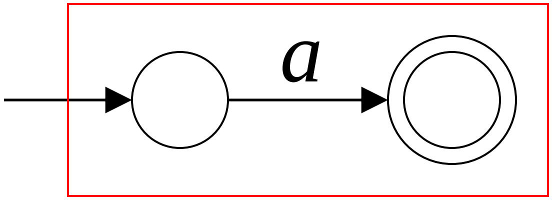
NFAs



$$L(M_1) = \emptyset = L(\emptyset)$$



$$L(M_2) = \{\lambda\} = L(\lambda)$$



$$L(M_3) = \{a\} = L(a)$$

regular languages

Inductive Hypothesis

Assume

for regular expressions r_1 and r_2

that

$L(r_1)$ and $L(r_2)$ are regular languages

Inductive Step

We will prove:

$$L(r_1 + r_2)$$

$$L(r_1 \cdot r_2)$$

$$L(r_1 *)$$

$$L((r_1))$$

Are regular Languages

By definition of regular expressions:

$$L(r_1 + r_2) = L(r_1) \cup L(r_2)$$

$$L(r_1 \cdot r_2) = L(r_1)\, L(r_2)$$

$$L(r_1 *) = (L(r_1))*$$

$$L((r_1)) = L(r_1)$$

By inductive hypothesis we know:

$L(r_1)$ and $L(r_2)$ are regular languages

We also know:

Regular languages are closed under:

Union	$L(r_1) \cup L(r_2)$
Concatenation	$L(r_1)\, L(r_2)$
Star	$(L(r_1))^*$

Therefore:

$$\left.\begin{aligned} L(r_1+r_2) &= L(r_1)\cup L(r_2) \\ L(r_1\cdot r_2) &= L(r_1)\,L(r_2) \\ L(r_1\,*) &= (L(r_1))* \end{aligned}\right\}\ \text{Are regular languages}$$

And trivially:

$L((r_1))$ is a regular language

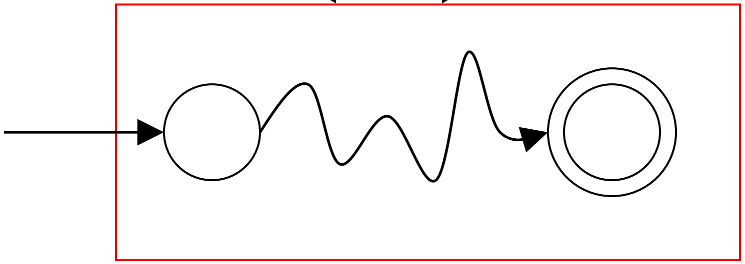
Proof - Part 2

2. For any regular language L there is a regular expression r with $L(r) = L$

Proof by construction of regular expression

Since L is regular take the NFA M that accepts it

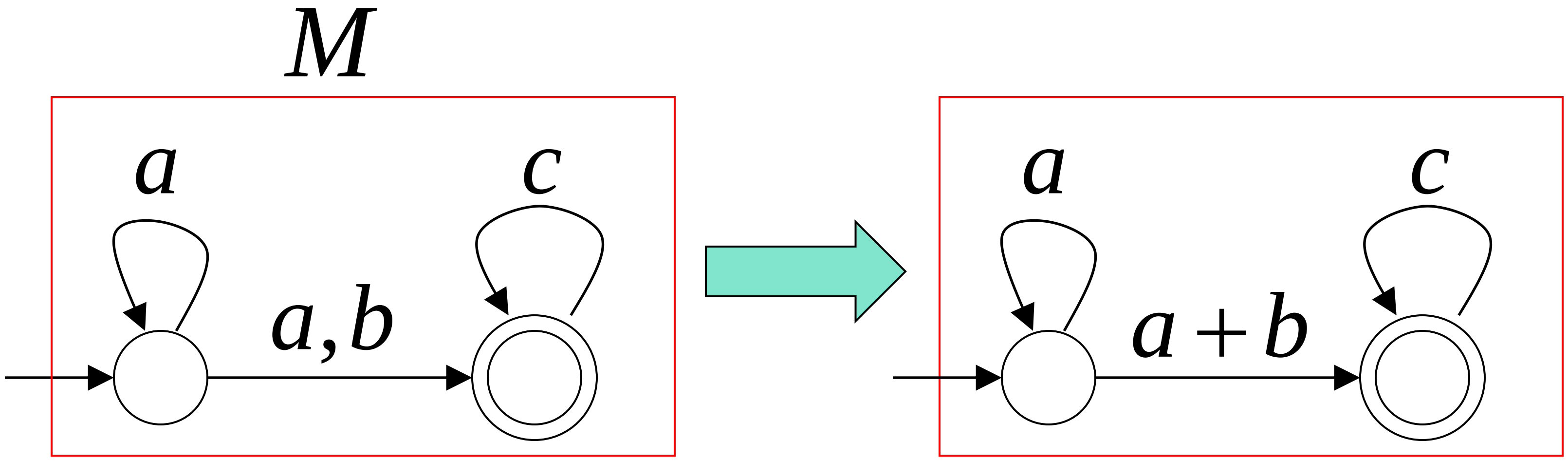
$$L(M) = L$$



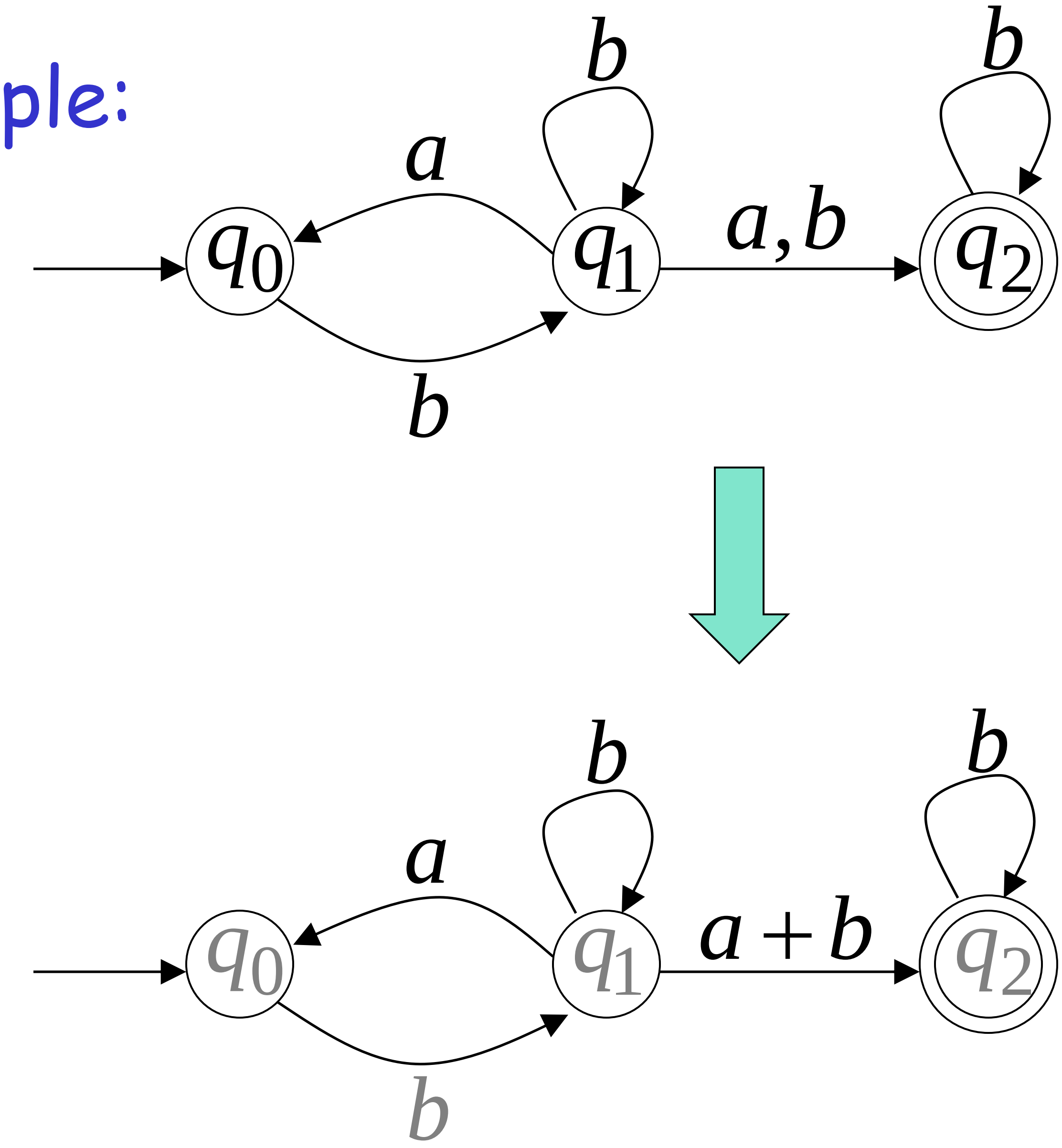
Single final state

From M construct the equivalent **Generalized Transition Graph** in which transition labels are regular expressions

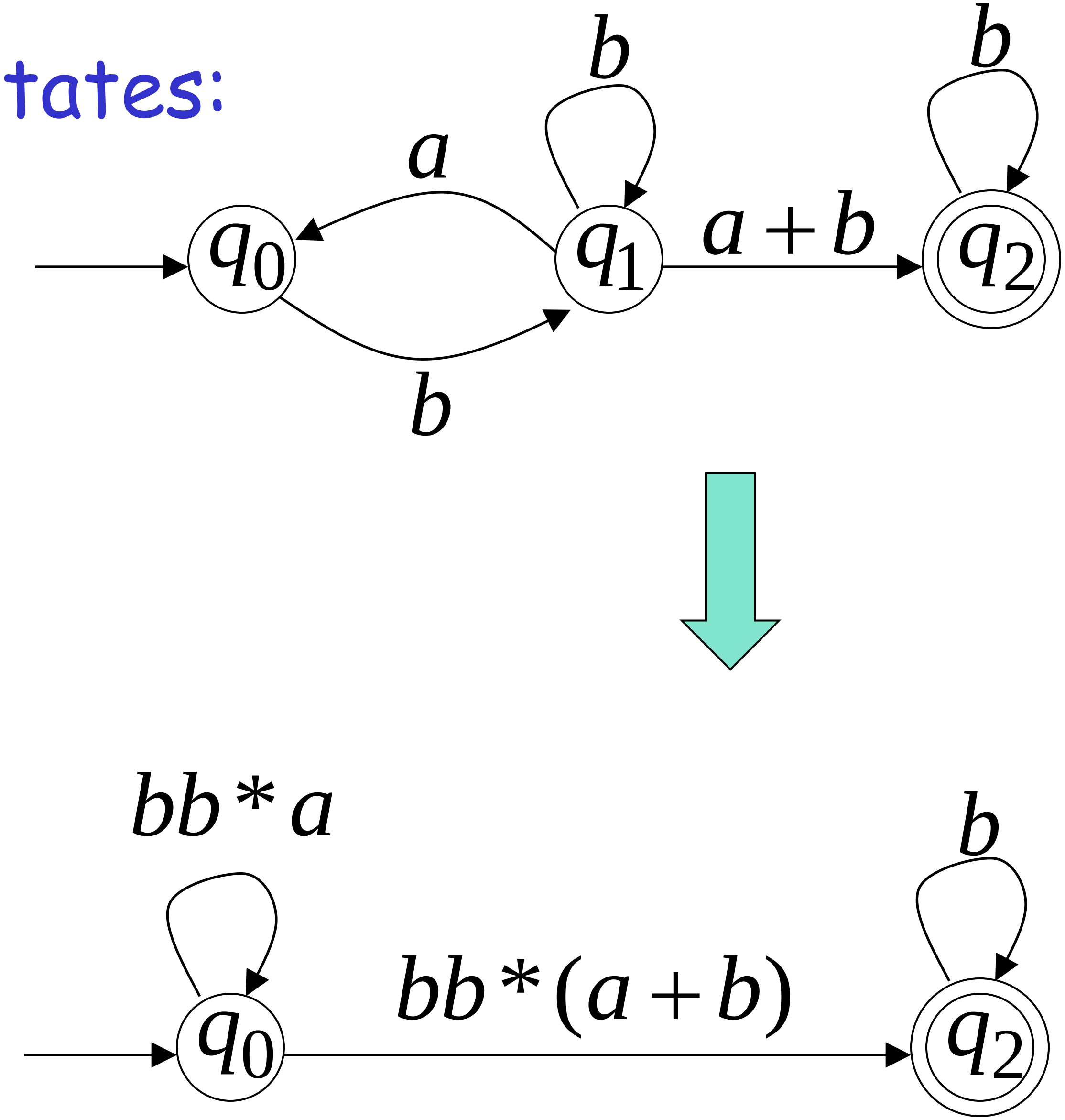
Example:



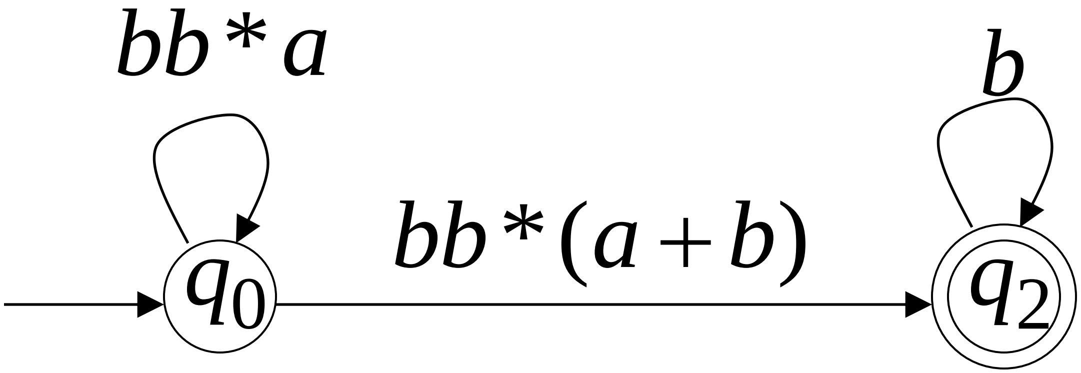
Another Example:



Reducing the states:



Resulting Regular Expression:

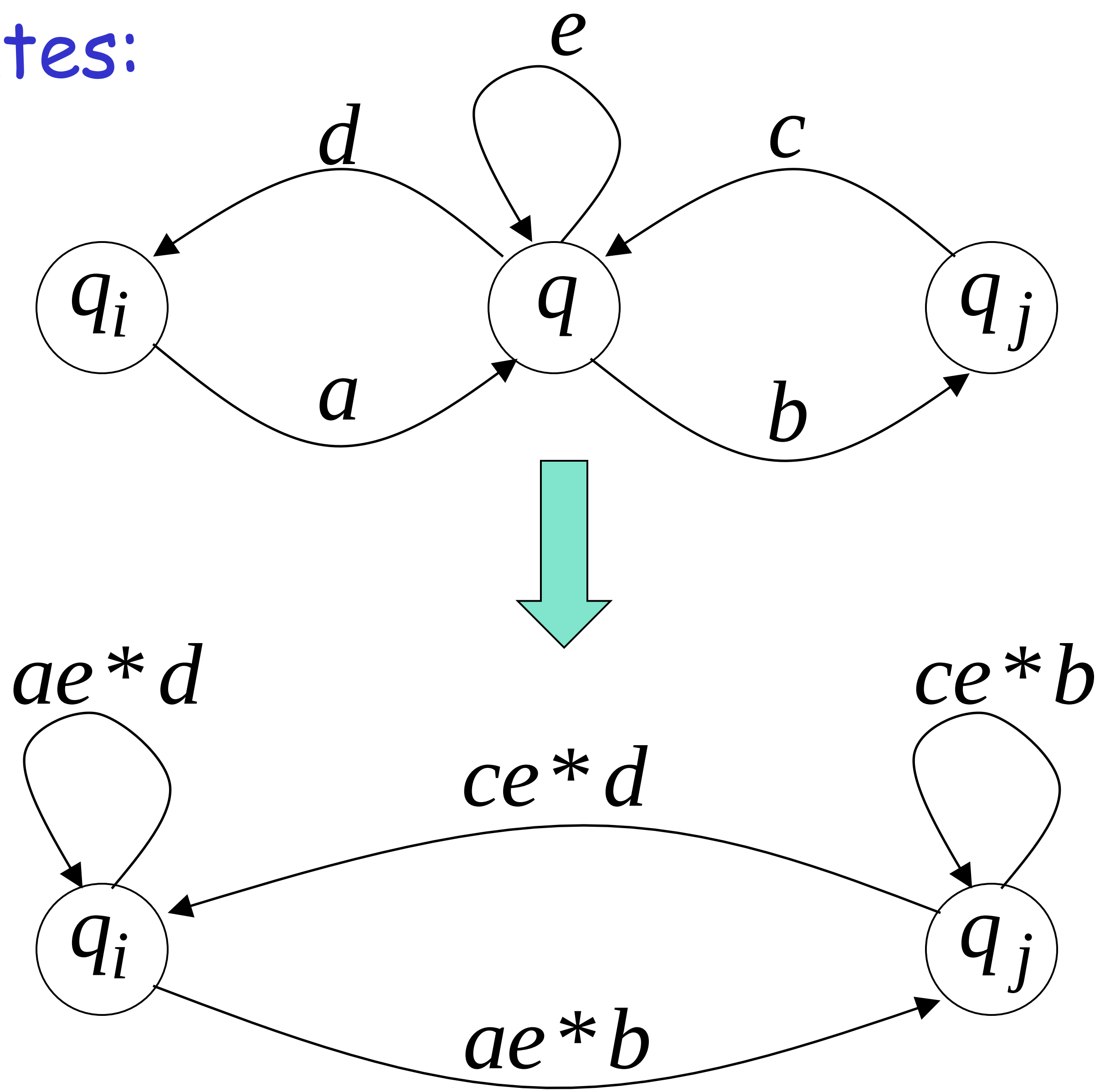


$$r = (bb^*a)^*bb^*(a+b)b^*$$

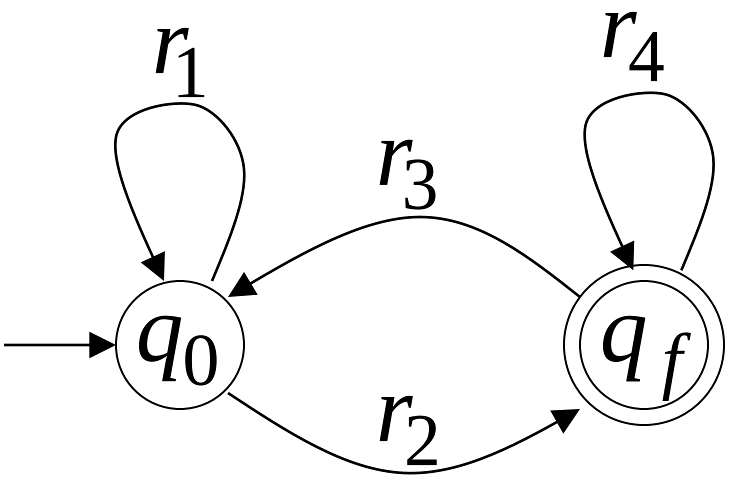
$$L(r) = L(M) = L$$

In General

Removing states:



The final transition graph:

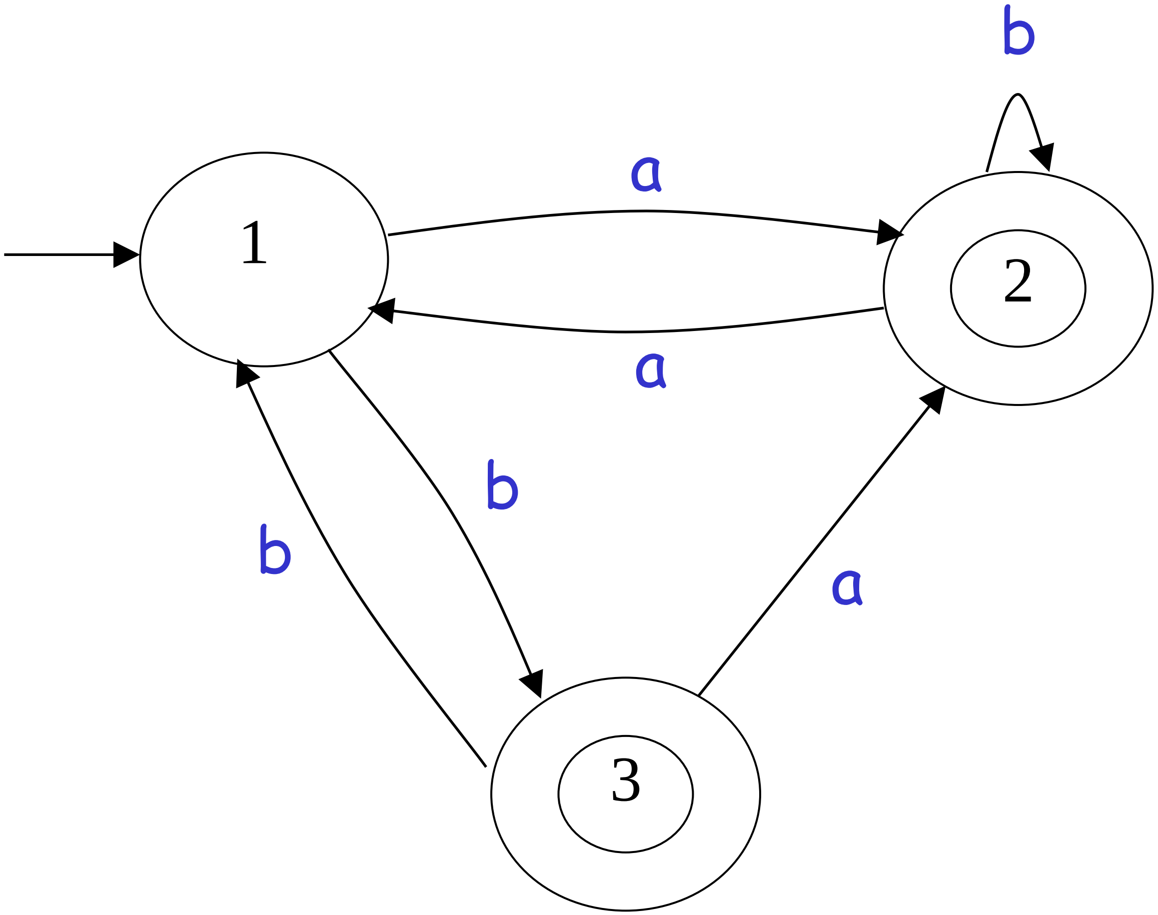


The resulting regular expression:

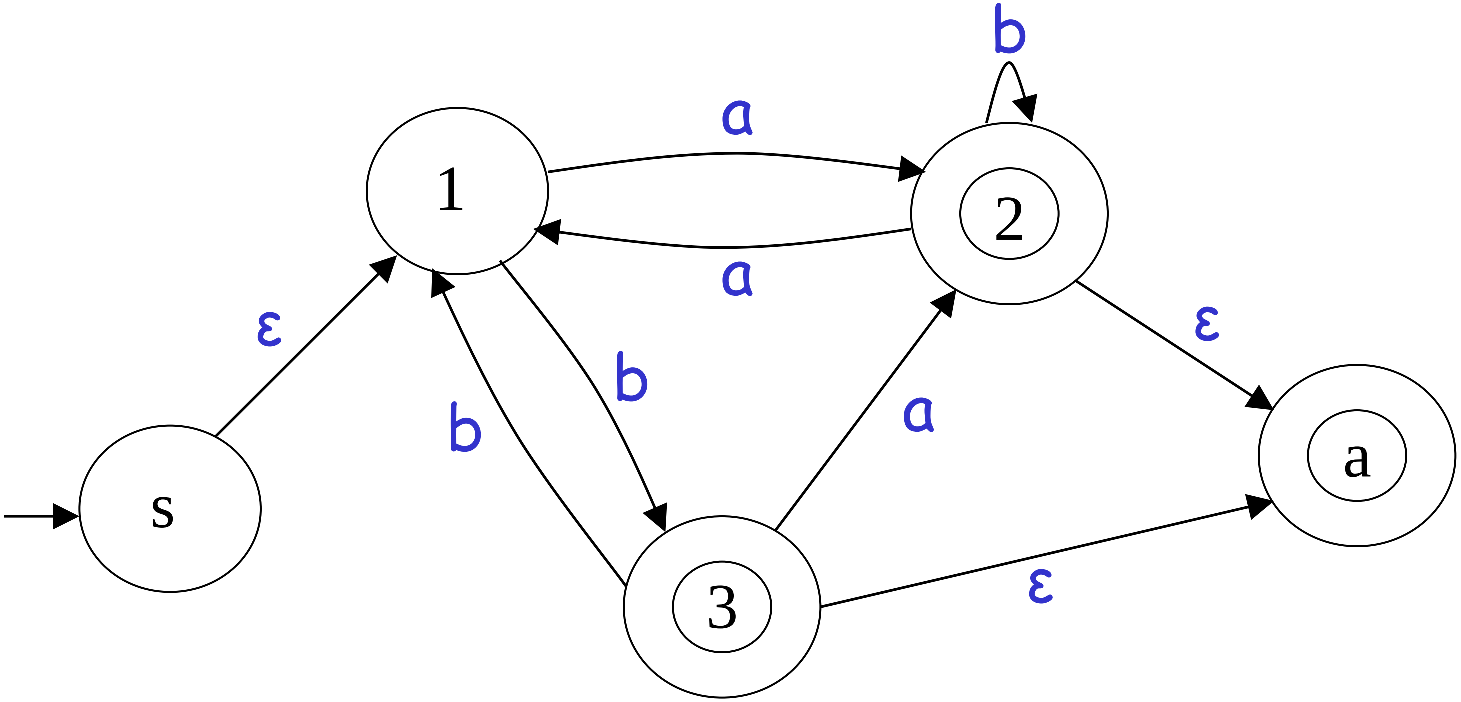
$$r = r_1 * r_2 (r_4 + r_3 r_1 * r_2) *$$

$$L(r) = L(M) = L$$

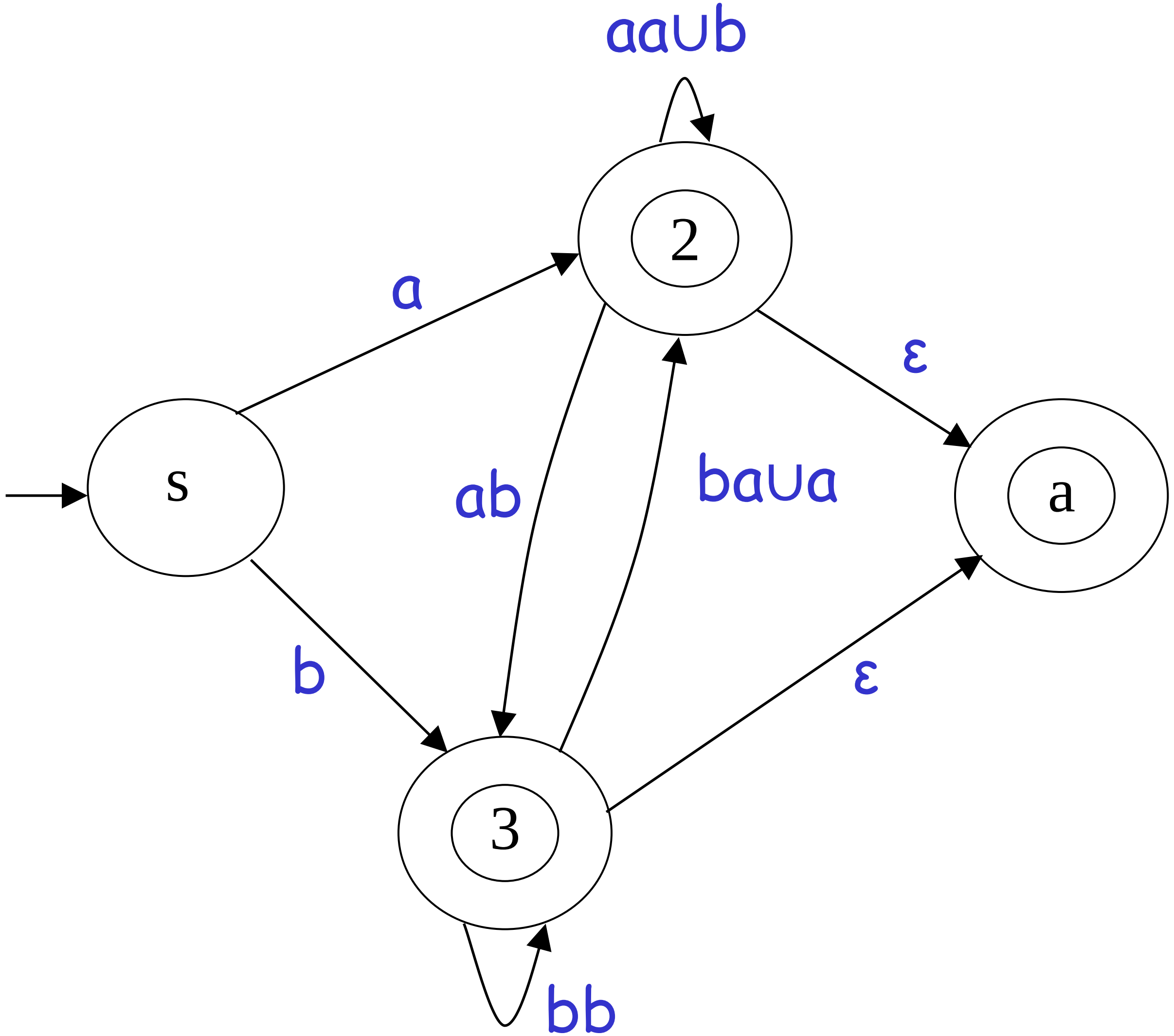
Example



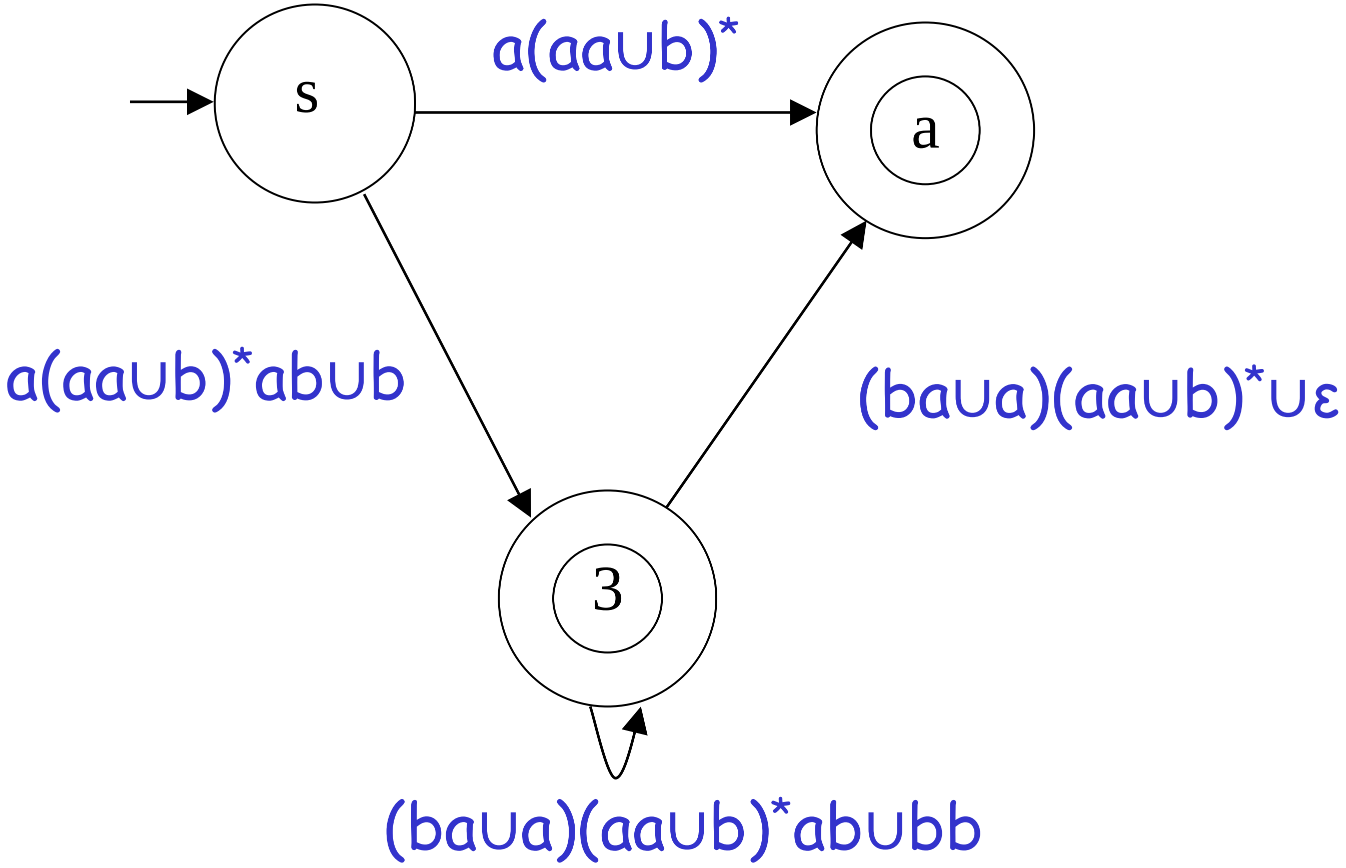
Example (Cont.)



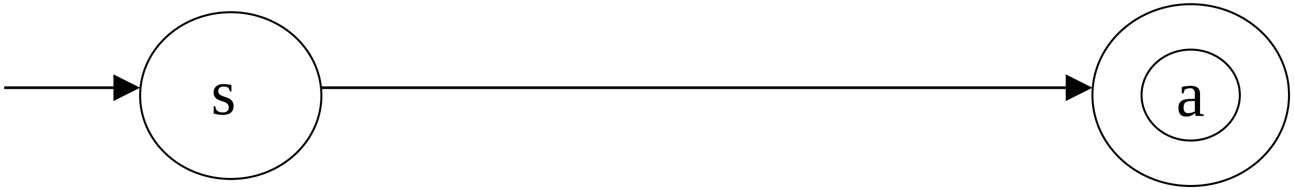
Example (Cont.)



Example (Cont.)



Example (Cont.)



(a(aa∪b)*ab ∪b)((ba ∪a)(aa ∪b)*ab ∪bb) *((ba ∪ a)(aa ∪b)* ∪ ε) ∪ a(aa ∪b)*

The End

Grammars

Grammars

Grammars express languages

Example: the English language

$$\langle sentence \rangle \rightarrow \langle noun\_phrase \rangle \ \langle predicate \rangle$$

$$\langle noun\_phrase \rangle \rightarrow \langle article \rangle \ \langle noun \rangle$$

$$\langle predicate \rangle \rightarrow \langle verb \rangle$$

$$\langle article \rangle \to a$$

$$\langle article \rangle \to the$$

$$\langle noun \rangle \to cat$$

$$\langle noun \rangle \to dog$$

$$\langle verb \rangle \to runs$$

$$\langle verb \rangle \to walks$$

A derivation of "the dog walks":

$$
\begin{aligned}
\langle sentence\rangle &\Rightarrow \langle noun\_phrase\rangle\ \langle predicate\rangle \\
&\Rightarrow \langle noun\_phrase\rangle\ \langle verb\rangle \\
&\Rightarrow \langle article\rangle\ \langle noun\rangle\ \langle verb\rangle \\
&\Rightarrow the\ \ \langle noun\rangle\ \langle verb\rangle \\
&\Rightarrow the\ \ dog\ \ \langle verb\rangle \\
&\Rightarrow the\ \ dog\ \ \ walks
\end{aligned}
$$

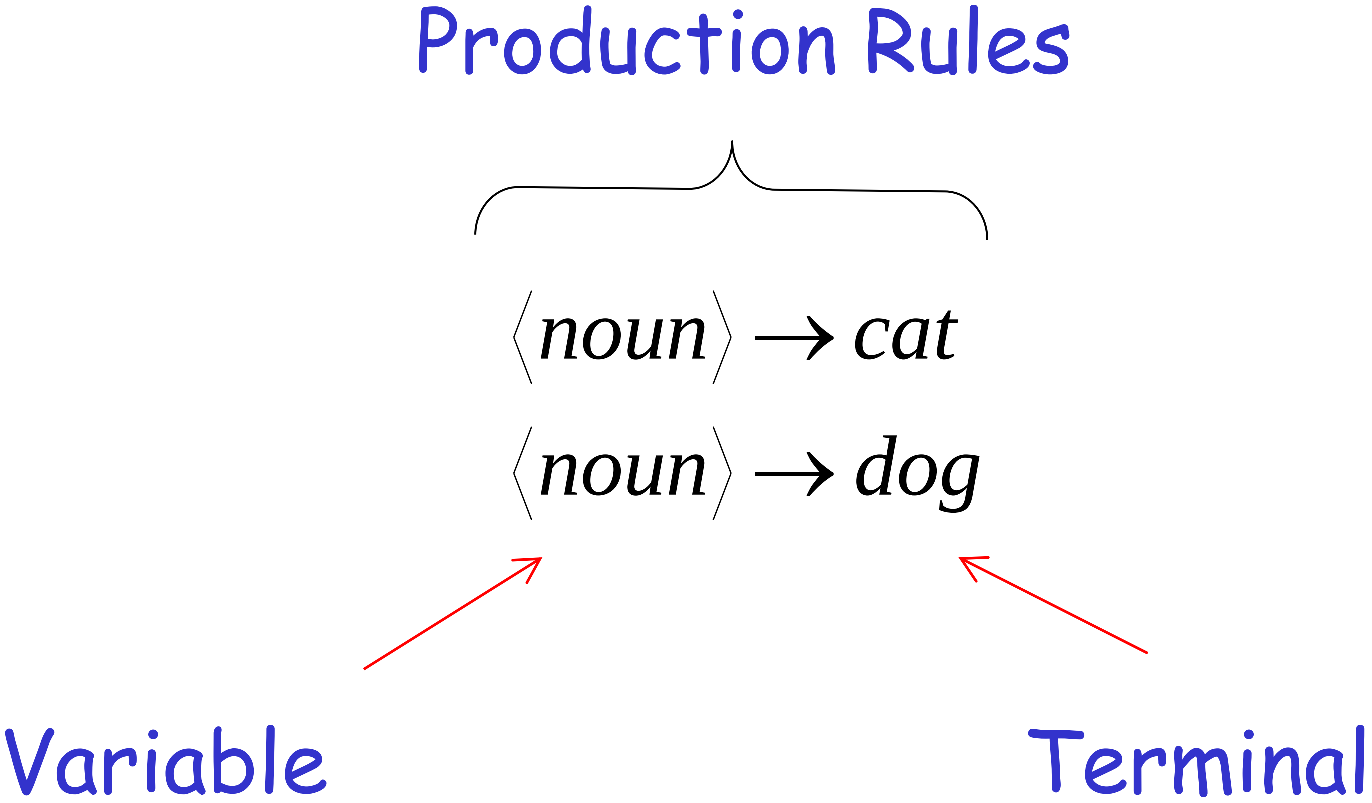
A derivation of "a cat runs":

$$
\begin{aligned}
\langle sentence \rangle &\Rightarrow \langle noun\_phrase \rangle \ \langle predicate \rangle \\
&\Rightarrow \langle noun\_phrase \rangle \ \langle verb \rangle \\
&\Rightarrow \langle article \rangle \ \langle noun \rangle \ \langle verb \rangle \\
&\Rightarrow a \ \ \langle noun \rangle \ \langle verb \rangle \\
&\Rightarrow a \ \ cat \ \ \langle verb \rangle \\
&\Rightarrow a \ \ cat \ \ runs
\end{aligned}
$$

Language of the grammar:

L = { "a cat runs",
"a cat walks",
"the cat runs",
"the cat walks",
"a dog runs",
"a dog walks",
"the dog runs",
"the dog walks" }

Notation



Production Rules

$$\langle noun \rangle \rightarrow cat$$

$$\langle noun \rangle \rightarrow dog$$

Variable

Terminal

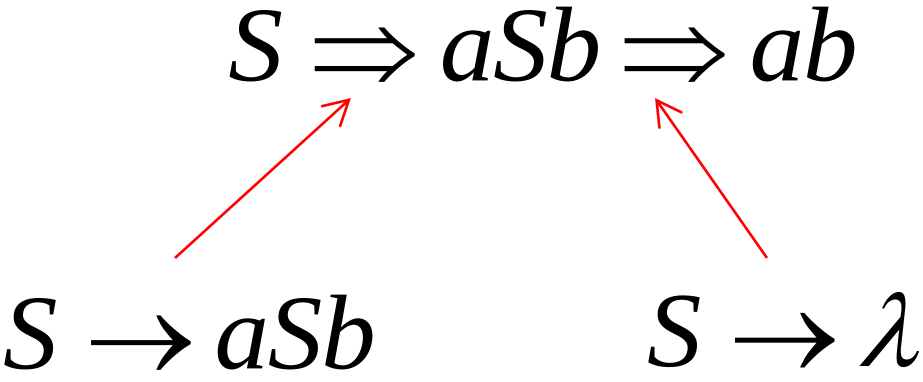
Another Example

Grammar:

$$S \rightarrow aSb$$

$$S \rightarrow \lambda$$

Derivation of sentence ab:



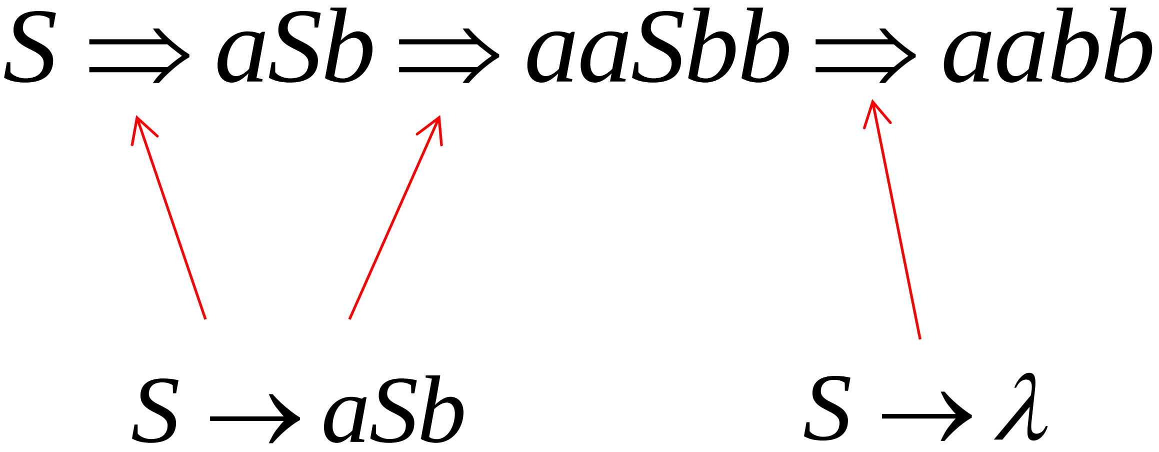
$$S \Rightarrow aSb \Rightarrow ab$$

$$S \rightarrow aSb \qquad\qquad S \rightarrow \lambda$$

Grammar: $S \to aSb$

$S \to \lambda$

Derivation of sentence $aabb$:



$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$$

$S \to aSb$

$S \to \lambda$

Other derivations:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb$$

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb$$

$$\Rightarrow aaaaSbbbb \Rightarrow aaaabbbb$$

Language of the grammar

$$S \rightarrow aSb$$

$$S \rightarrow \lambda$$

$$L = \{a^n b^n : n \geq 0\}$$

More Notation

Grammar $G=(V,T,S,P)$

$V:$ Set of variables

$T:$ Set of terminal symbols

$S:$ Start variable

$P:$ Set of Production rules

Example

Grammar G :

$$S \to aSb$$

$$S \to \lambda$$

$$G = (V, T, S, P)$$

$$V = \{S\} \qquad T = \{a, b\}$$

$$P = \{S \to aSb,\ S \to \lambda\}$$

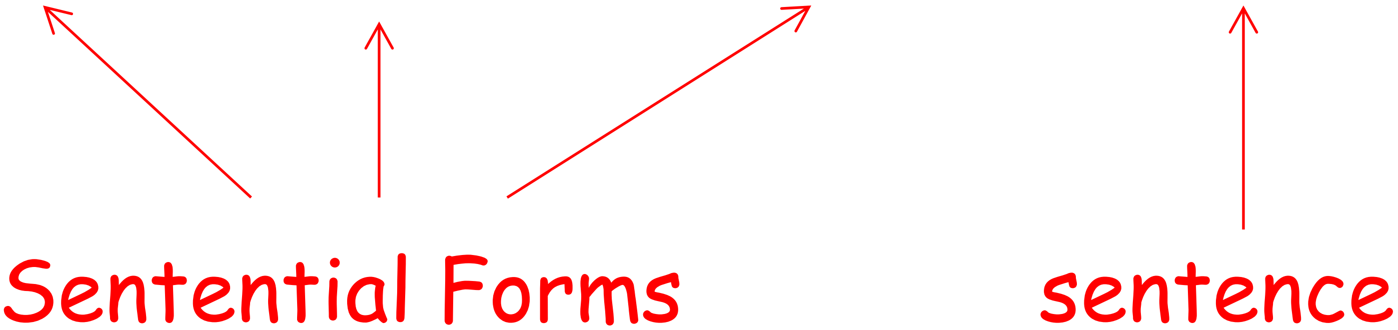
More Notation

Sentential Form:

A sentence that contains variables and terminals

Example:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb$$



Sentential Forms

sentence

We write: $S \overset{*}{\Rightarrow} aaabbb$

Instead of:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb$$

In general we write: $w_1 \overset{*}{\Rightarrow} w_n$

If: $w_1 \Rightarrow w_2 \Rightarrow w_3 \Rightarrow \cdots \Rightarrow w_n$

By default: $w \overset{*}{\Rightarrow} w$

Example

Grammar

$$S \rightarrow aSb$$

$$S \rightarrow \lambda$$

Derivations

$$S \overset{*}{\Rightarrow} \lambda$$

$$S \overset{*}{\Rightarrow} ab$$

$$S \overset{*}{\Rightarrow} aabb$$

$$S \overset{*}{\Rightarrow} aaabbb$$

Example

Grammar

$$S \to aSb$$

$$S \to \lambda$$

Derivations

$$S \overset{*}{\Rightarrow} aaSbb$$

$$aaSbb \overset{*}{\Rightarrow} aaaaaSbbbbb$$

Another Grammar Example

Grammar $G:$ $S \rightarrow Ab$

$A \rightarrow aAb$

$A \rightarrow \lambda$

Derivations:

$$S \Rightarrow Ab \Rightarrow b$$

$$S \Rightarrow Ab \Rightarrow aAbb \Rightarrow abb$$

$$S \Rightarrow Ab \Rightarrow aAbb \Rightarrow aaAbbb \Rightarrow aabbb$$

More Derivations

$$S \Rightarrow Ab \Rightarrow aAbb \Rightarrow aaAbbb \Rightarrow aaaAbbbb$$

$$\Rightarrow aaaaAbbbbb \Rightarrow aaaabbbbb$$

$$S \overset{*}{\Rightarrow} aaaabbbbb$$

$$S \overset{*}{\Rightarrow} aaaaaabbbbbbb$$

$$S \overset{*}{\Rightarrow} a^n b^n b$$

Language of a Grammar

For a grammar G
with start variable S :

$$L(G) = \{w : \quad S \overset{*}{\Rightarrow} w\}$$

String of terminals

Example

For grammar G :
$$S \rightarrow Ab$$
$$A \rightarrow aAb$$
$$A \rightarrow \lambda$$

$$L(G) = \{a^n b^n b : \quad n \geq 0\}$$

Since: $S \overset{*}{\Rightarrow} a^n b^n b$

A Convenient Notation

$$A \to aAb$$
$$A \to \lambda$$

➡

$$A \to aAb \mid \lambda$$

$$\langle article \rangle \to a$$
$$\langle article \rangle \to the$$

➡

$$\langle article \rangle \to a \mid the$$

Linear Grammars

Linear Grammars

Grammars with
at most one variable at the right side
of a production

Examples: $S \rightarrow aSb$ $\quad$ $S \rightarrow Ab$

$S \rightarrow \lambda$ $\quad$ $A \rightarrow aAb$

$A \rightarrow \lambda$

A Non-Linear Grammar

Grammar G :

$$S \to SS$$

$$S \to \lambda$$

$$S \to aSb$$

$$S \to bSa$$

$$L(G) = \{w : \quad n_a(w) = n_b(w)\}$$

Number of a in string w

Another Linear Grammar

Grammar G :

$$S \to A$$

$$A \to aB \mid \lambda$$

$$B \to Ab$$

$$L(G) = \{a^n b^n : n \geq 0\}$$

Right-Linear Grammars

All productions have form: $A \rightarrow xB$

or

$A \rightarrow x$

string of terminals

Example: $S \rightarrow abS$

$S \rightarrow a$

Left-Linear Grammars

All productions have form:

$$A \rightarrow Bx$$

or

$$A \rightarrow x$$

string of terminals

Example:

$$S \rightarrow Aab$$

$$A \rightarrow Aab \mid B$$

$$B \rightarrow a$$

Regular Grammars

Regular Grammars

A regular grammar is any
right-linear or left-linear grammar

Examples:

G_1

$$S \to abS$$

$$S \to a$$

G_2

$$S \to Aab$$

$$A \to Aab \mid B$$

$$B \to a$$

Observation

Regular grammars generate regular languages

Examples:

G_2

G_1

$S \rightarrow Aab$

$S \rightarrow abS$

$A \rightarrow Aab \mid B$

$S \rightarrow a$

$B \rightarrow a$

$$L(G_1) = (ab)^* a$$

$$L(G_2) = aab(ab)^*$$

Regular Grammars
Generate
Regular Languages

Theorem

$$\left\{\begin{array}{l}\text{Languages}\\ \text{Generated by}\\ \text{Regular Grammars}\end{array}\right\} = \left\{\begin{array}{l}\text{Regular}\\ \text{Languages}\end{array}\right\}$$

Theorem - Part 1

$$\left\{ \begin{array}{l} \text{Languages} \\ \text{Generated by} \\ \text{Regular Grammars} \end{array} \right\} \subseteq \left\{ \begin{array}{l} \text{Regular} \\ \text{Languages} \end{array} \right\}$$

Any regular grammar generates
a regular language

Theorem - Part 2

$$\left\{\begin{array}{l}\text{Languages}\\\text{Generated by}\\\text{Regular Grammars}\end{array}\right\} \supseteq \left\{\begin{array}{l}\text{Regular}\\\text{Languages}\end{array}\right\}$$

Any regular language is generated by a regular grammar

Proof - Part 1

$$\left\{\begin{array}{l}\text{Languages}\\\text{Generated by}\\\text{Regular Grammars}\end{array}\right\} \subseteq \left\{\begin{array}{l}\text{Regular}\\\text{Languages}\end{array}\right\}$$

The language $L(G)$ generated by any regular grammar G is regular

The case of Right-Linear Grammars

Let G be a right-linear grammar

We will prove: $L(G)$ is regular

Proof idea: We will construct NFA M with $L(M) = L(G)$

Grammar G is right-linear

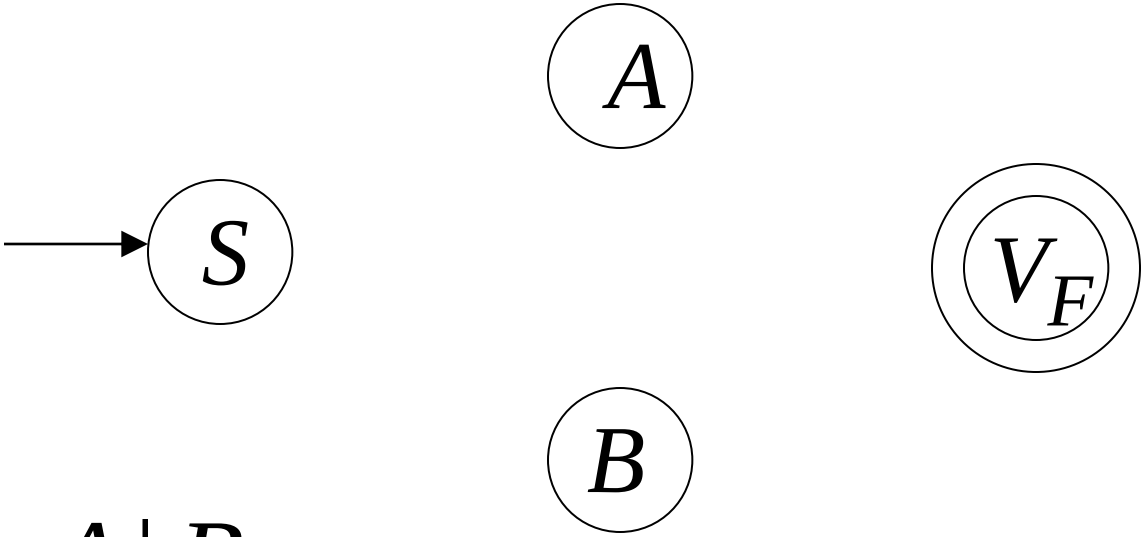
Example:

$$S \to aA \mid B$$

$$A \to aa\,B$$

$$B \to b\,B \mid a$$

Construct NFA M such that every state is a grammar variable:



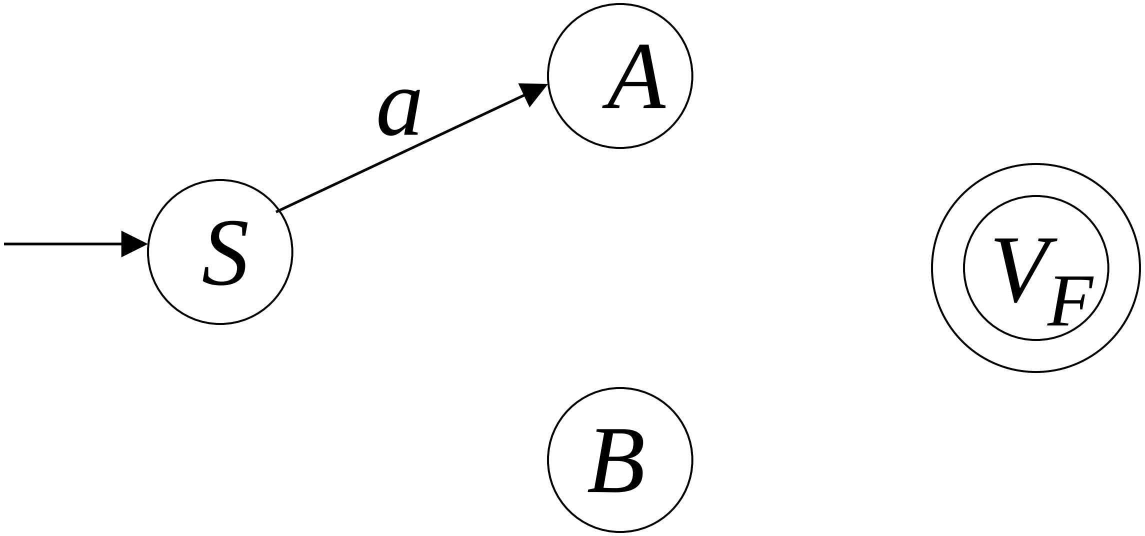
special final state

$$S \to aA \mid B$$

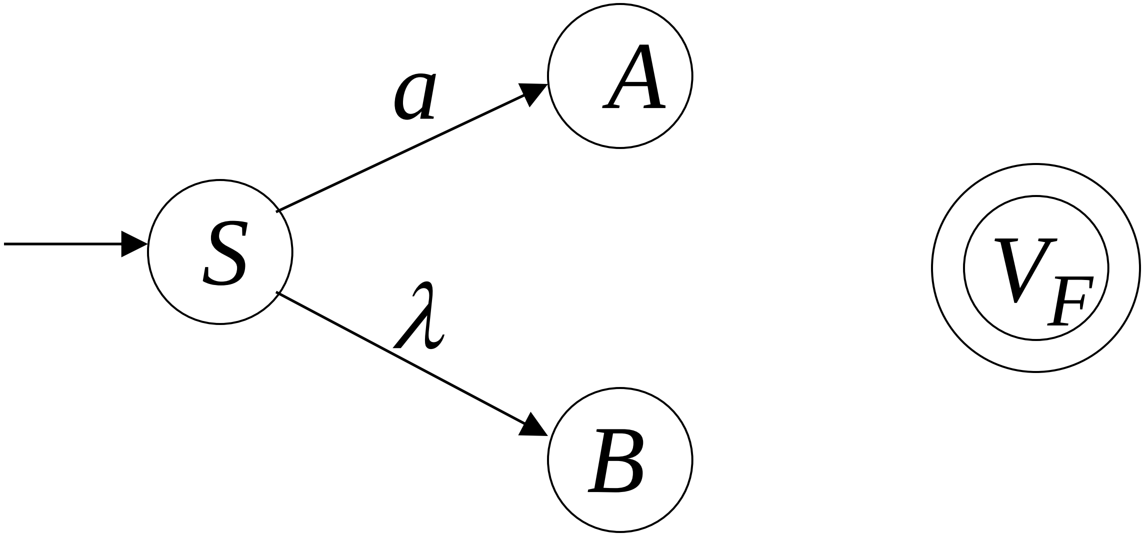
$$A \to aa\ B$$

$$B \to b\ B \mid a$$

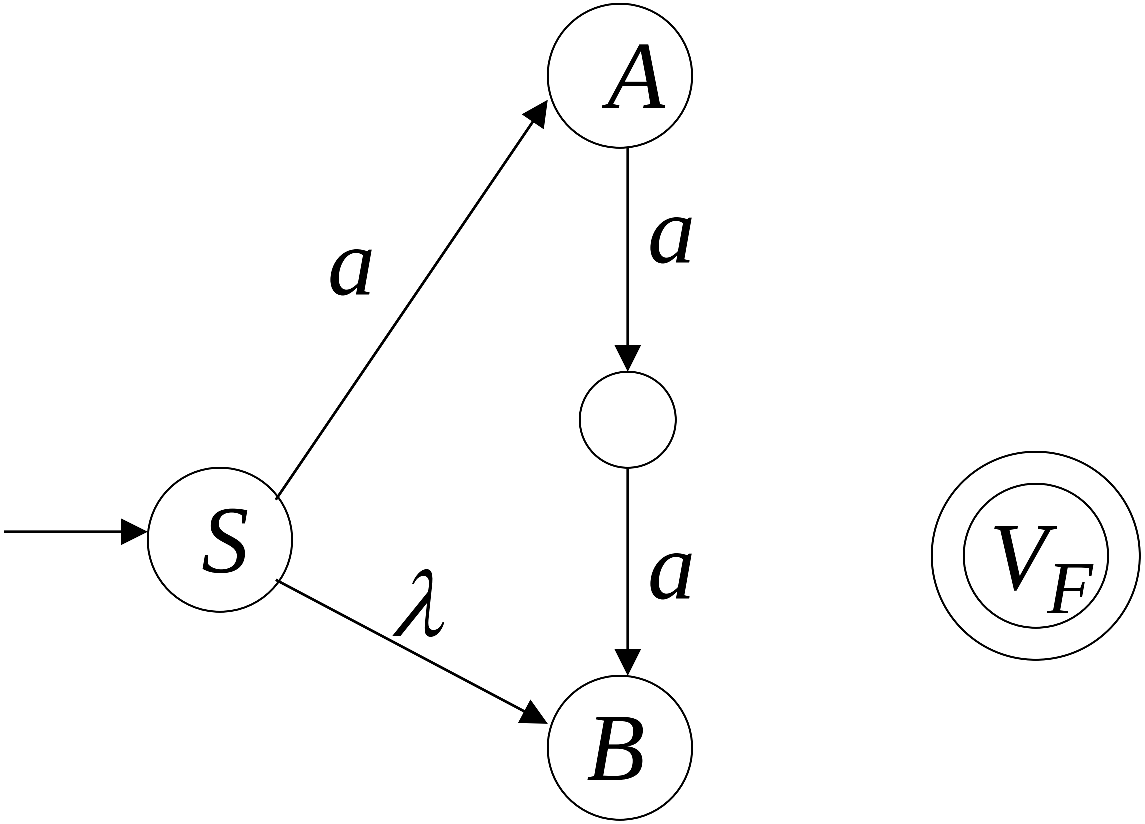
Add edges for each production:



$$S \rightarrow aA$$

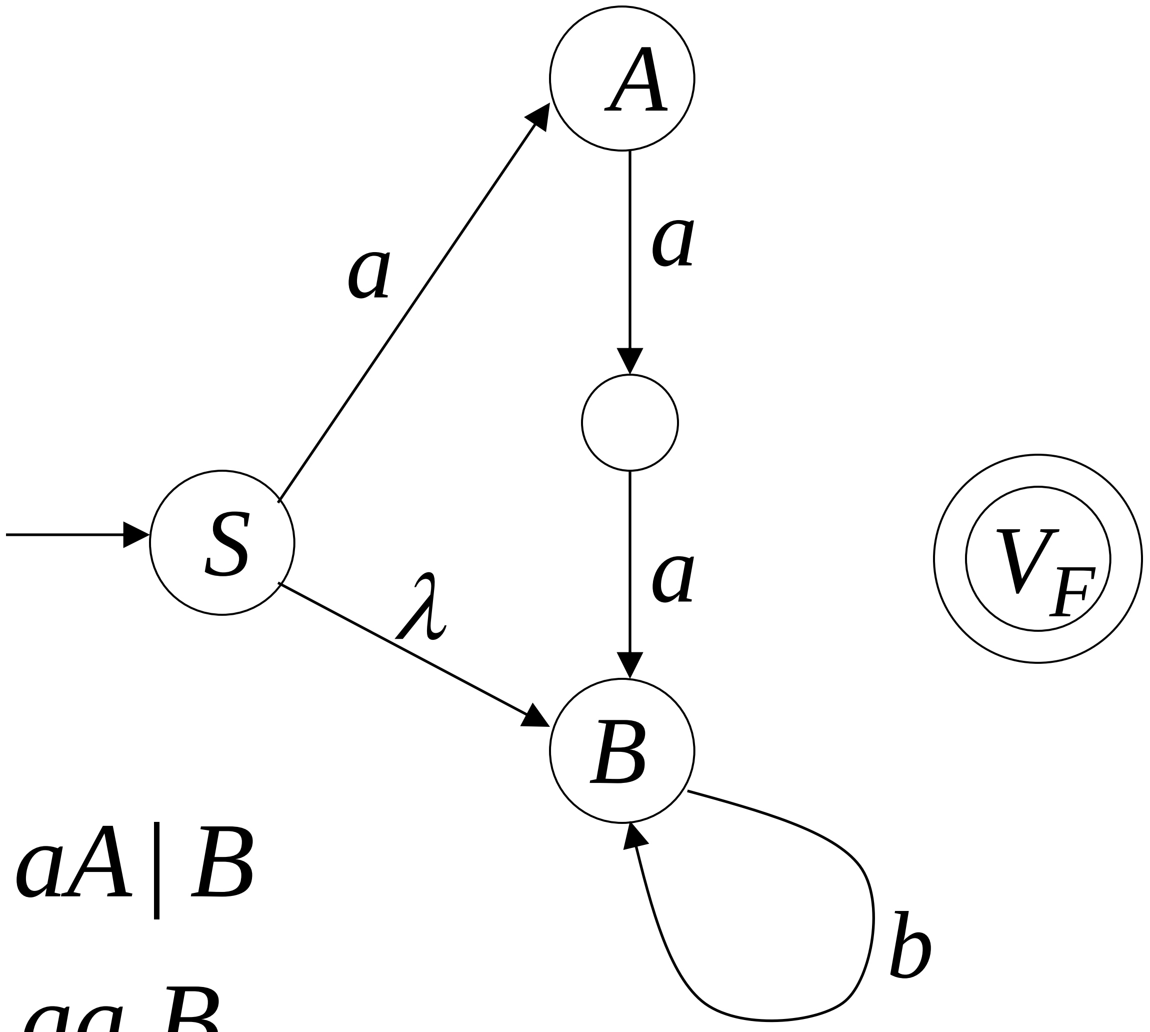


$$S \rightarrow aA \mid B$$



$$S \to aA \mid B$$

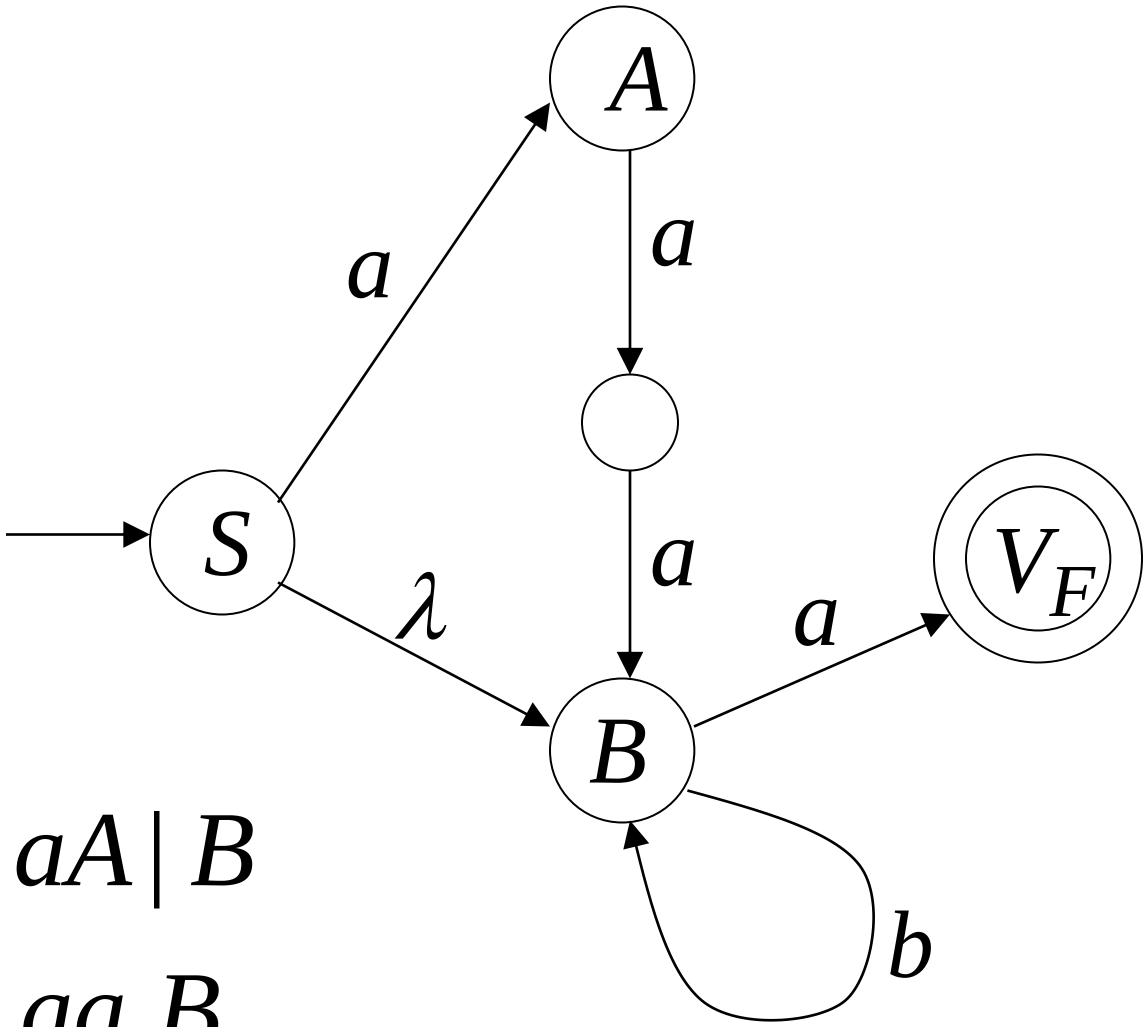
$$A \to aa\ B$$



$S \rightarrow aA \mid B$

$A \rightarrow aa\ B$

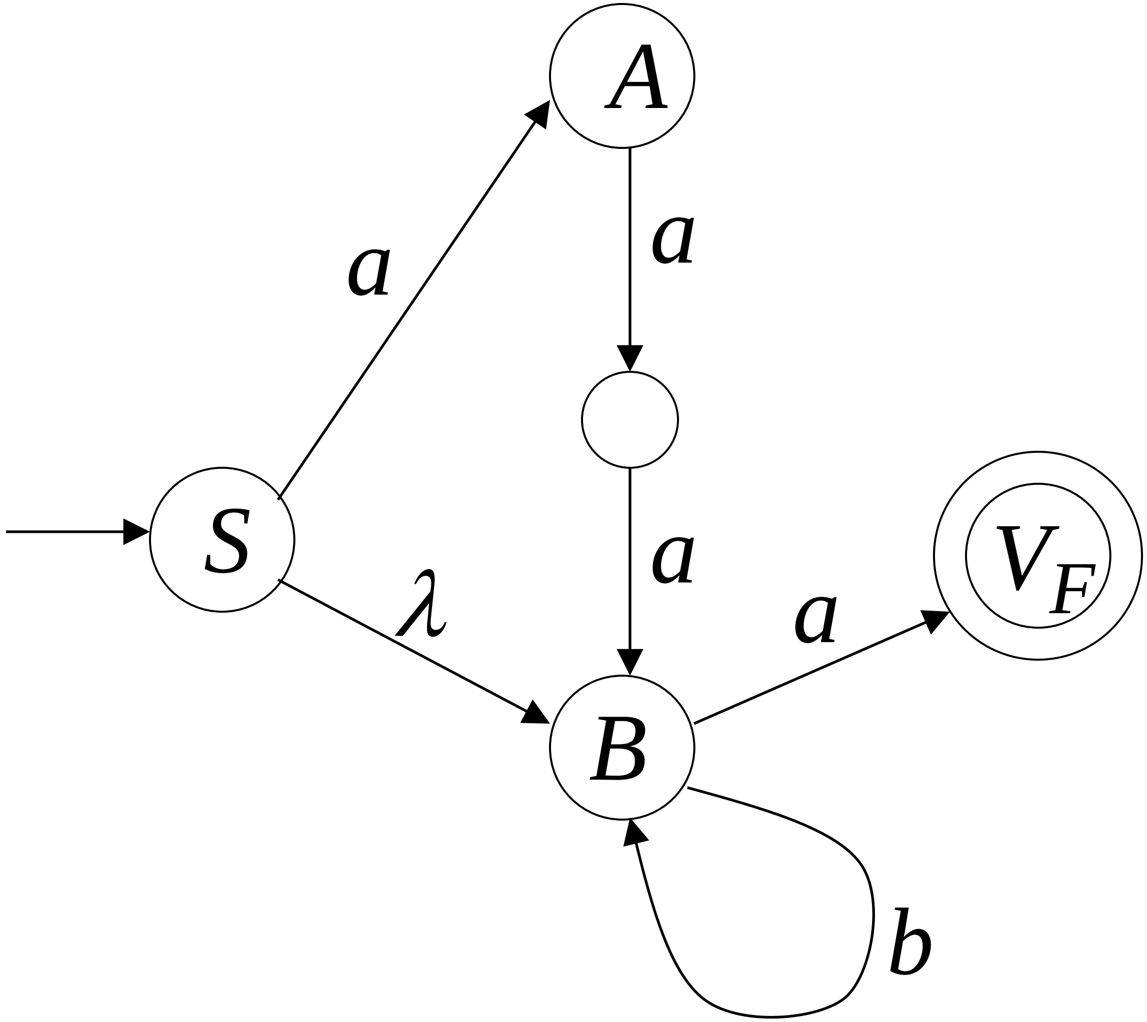
$B \rightarrow bB$



$$S \to aA \mid B$$

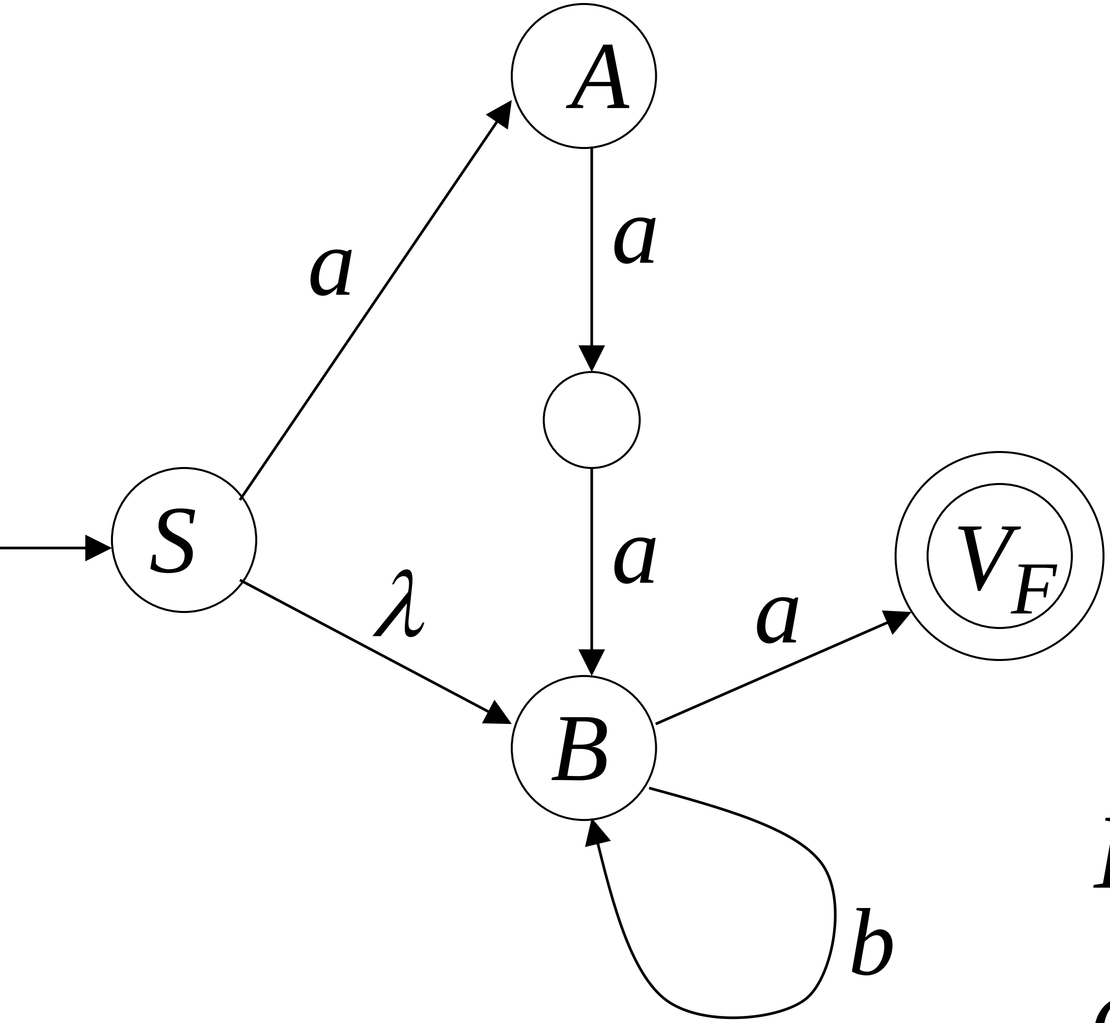
$$A \to aa\ B$$

$$B \to bB \mid a$$



$$S \Rightarrow aA \Rightarrow aaaB \Rightarrow aaabB \Rightarrow aaaba$$

NFA M



Grammar

G

$$S \rightarrow aA \mid B$$

$$A \rightarrow aa\,B$$

$$B \rightarrow bB \mid a$$

$$L(M) = L(G) =$$

$$aaab^*a + b^*a$$

In General

A right-linear grammar G

has variables: $V_0, V_1, V_2, \ldots$

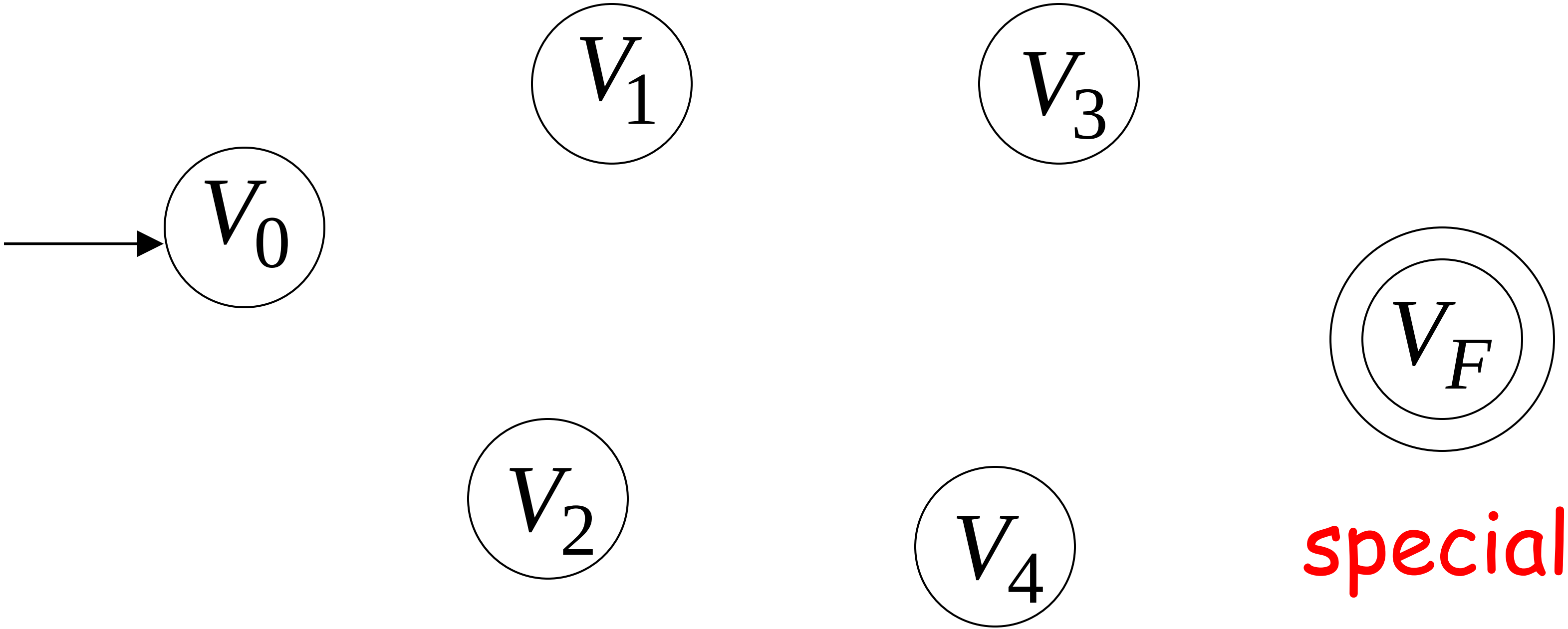
and productions: $V_i \rightarrow a_1 a_2 \cdots a_m V_j$

or

$$V_i \rightarrow a_1 a_2 \cdots a_m$$

We construct the NFA M such that:

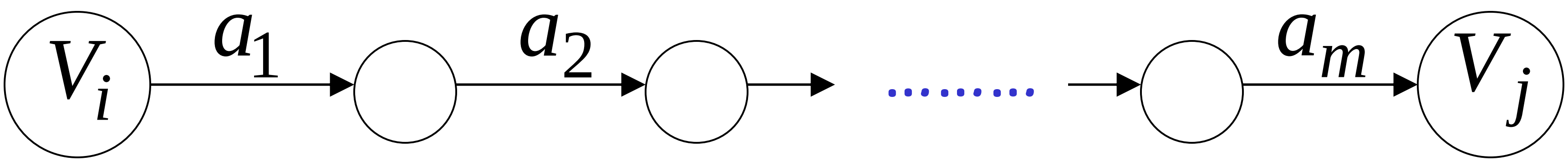
each variable V_i corresponds to a node:



special
final state

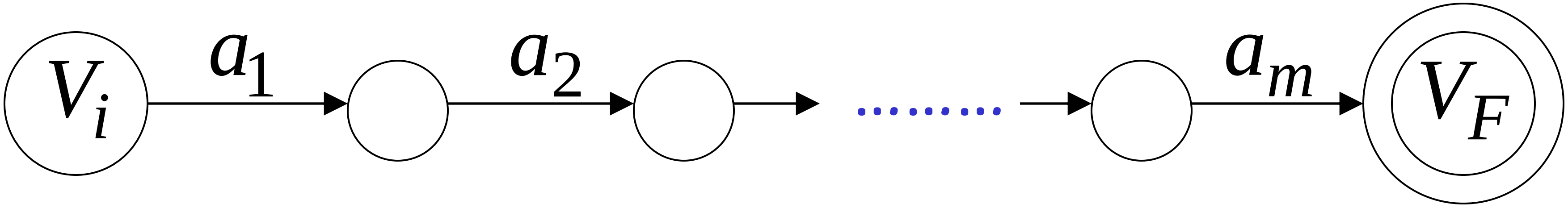
For each production: $V_i \to a_1 a_2 \cdots a_m V_j$

we add transitions and intermediate nodes

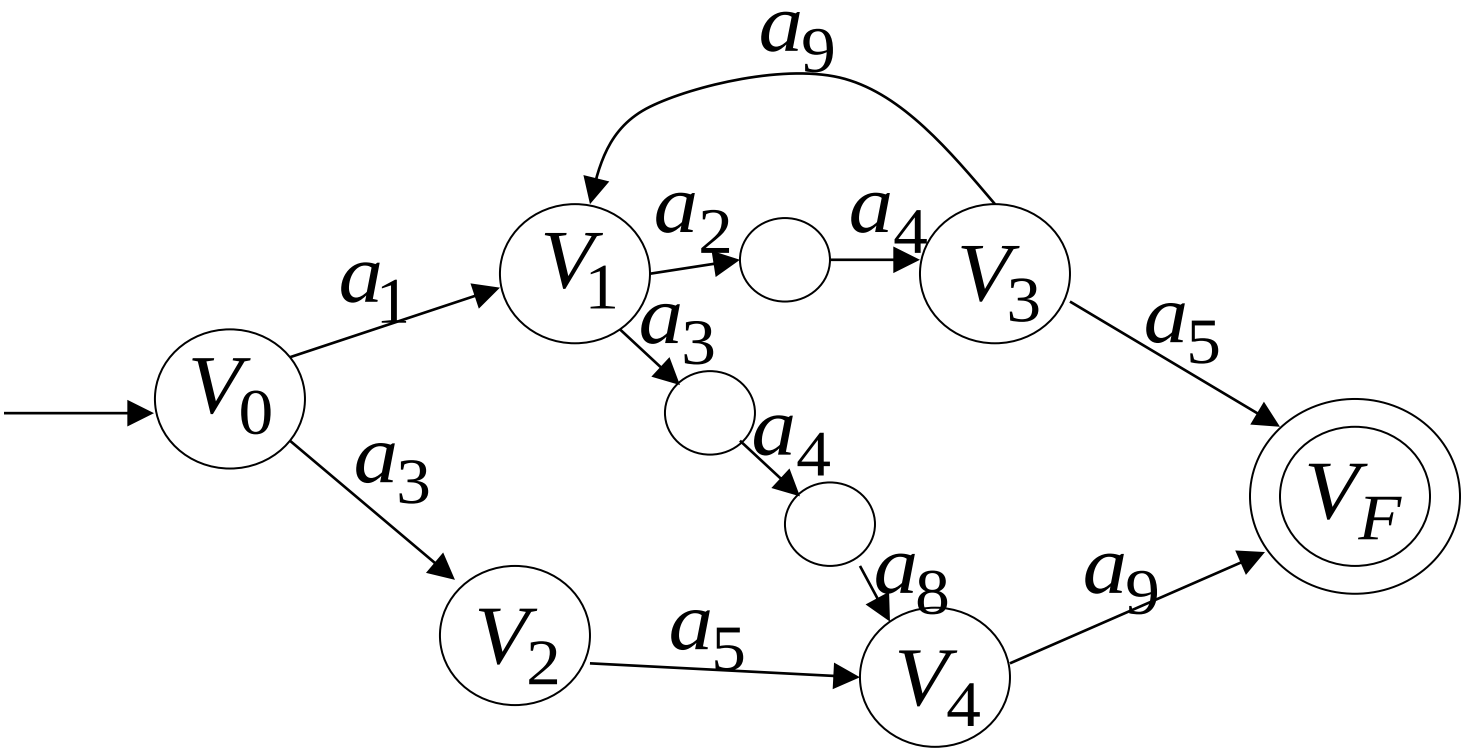


For each production: $V_i \rightarrow a_1 a_2 \cdots a_m$

we add transitions and intermediate nodes



Resulting NFA M looks like this:



It holds that: $L(G) = L(M)$

The case of Left-Linear Grammars

Let G be a left-linear grammar

We will prove: $L(G)$ is regular

Proof idea:

We will construct a right-linear grammar G' with $L(G) = L(G')^R$

Since G is left-linear grammar the productions look like:

$$A \to Ba_1a_2\cdots a_k$$

$$A \to a_1a_2\cdots a_k$$

Construct right-linear grammar G'

Left linear G

$$A \to Ba_1a_2\cdots a_k$$

$$A \to Bv$$

Right linear G'

$$A \to a_k\cdots a_2a_1B$$

$$A \to v^RB$$

Construct right-linear grammar G'

Left linear G

$$A \to a_1 a_2 \cdots a_k$$

$$A \to v$$

Right linear G'

$$A \to a_k \cdots a_2 a_1$$

$$A \to v^R$$

It is easy to see that: $L(G) = L(G')^R$

Since G' is right-linear, we have:

$$L(G') \Rightarrow L(G')^R \Rightarrow L(G)$$

Regular Language	Regular Language	Regular Language

Proof - Part 2

$$\left\{ \begin{array}{l} \text{Languages} \\ \text{Generated by} \\ \text{Regular Grammars} \end{array} \right\} \supseteq \left\{ \begin{array}{l} \text{Regular} \\ \text{Languages} \end{array} \right\}$$

Any regular language L is generated by some regular grammar G

Any regular language L is generated by some regular grammar G

Proof idea:

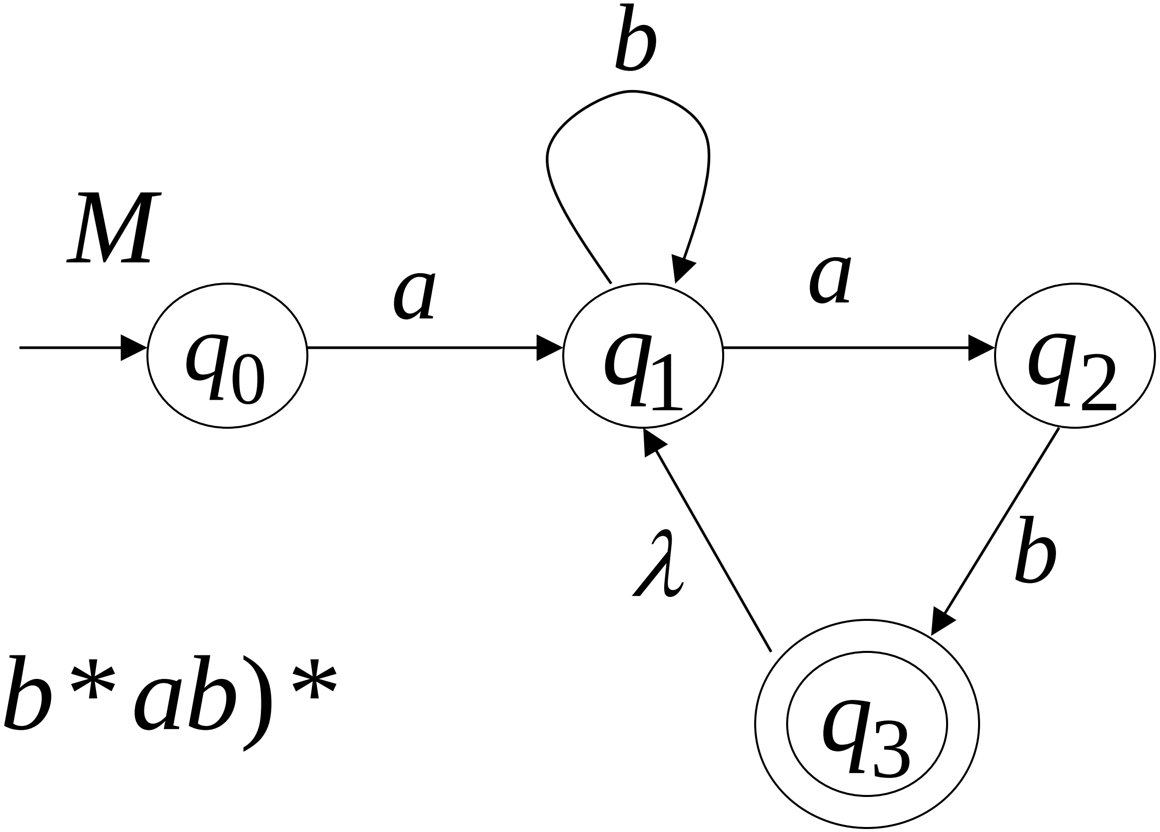
Let M be the NFA with $L = L(M)$.

Construct from M a regular grammar G such that $L(M) = L(G)$

Since L is regular

there is an NFA M such that $L = L(M)$

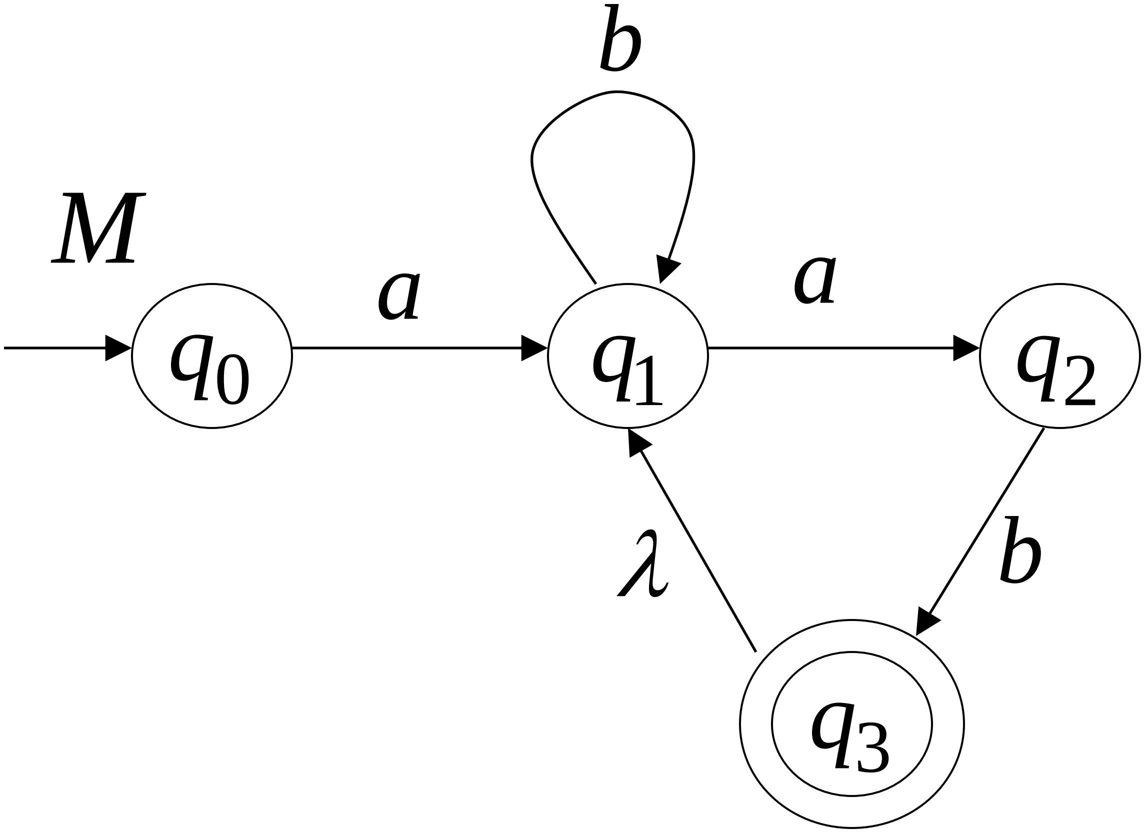
Example:



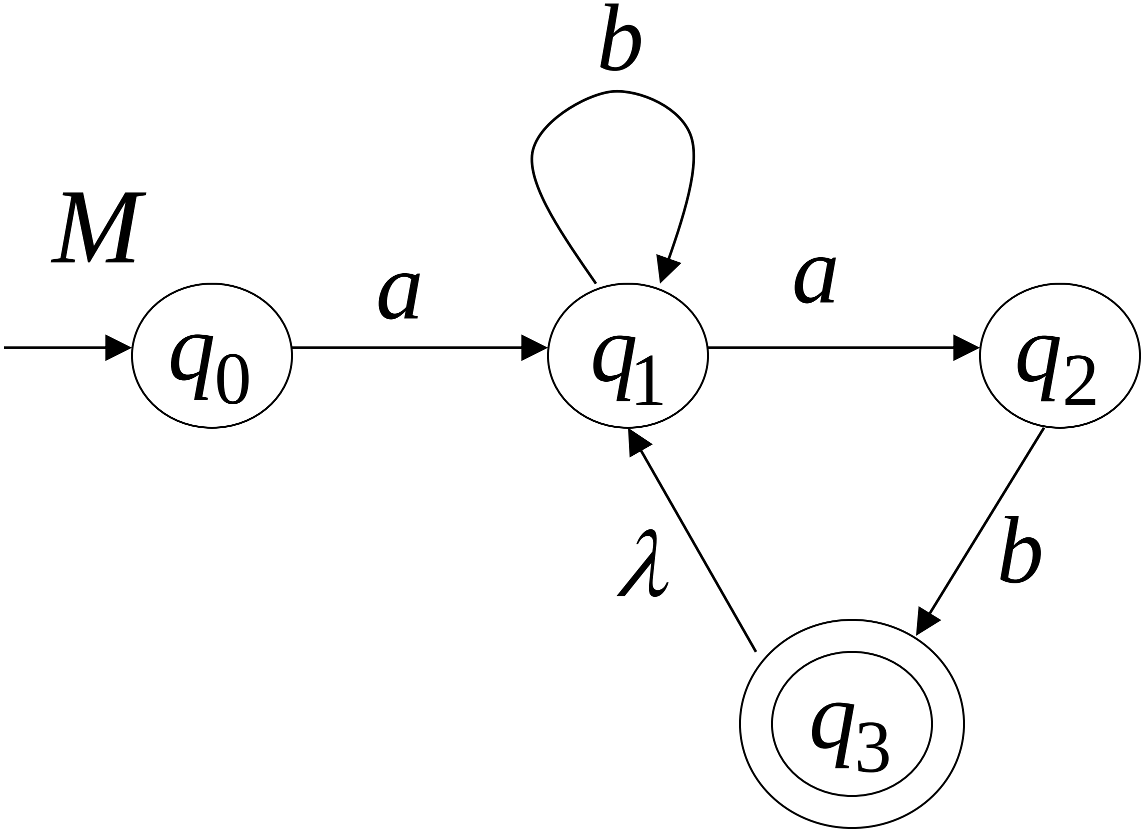
$$L = ab * ab(b * ab) *$$

$$L = L(M)$$

Convert M to a right-linear grammar



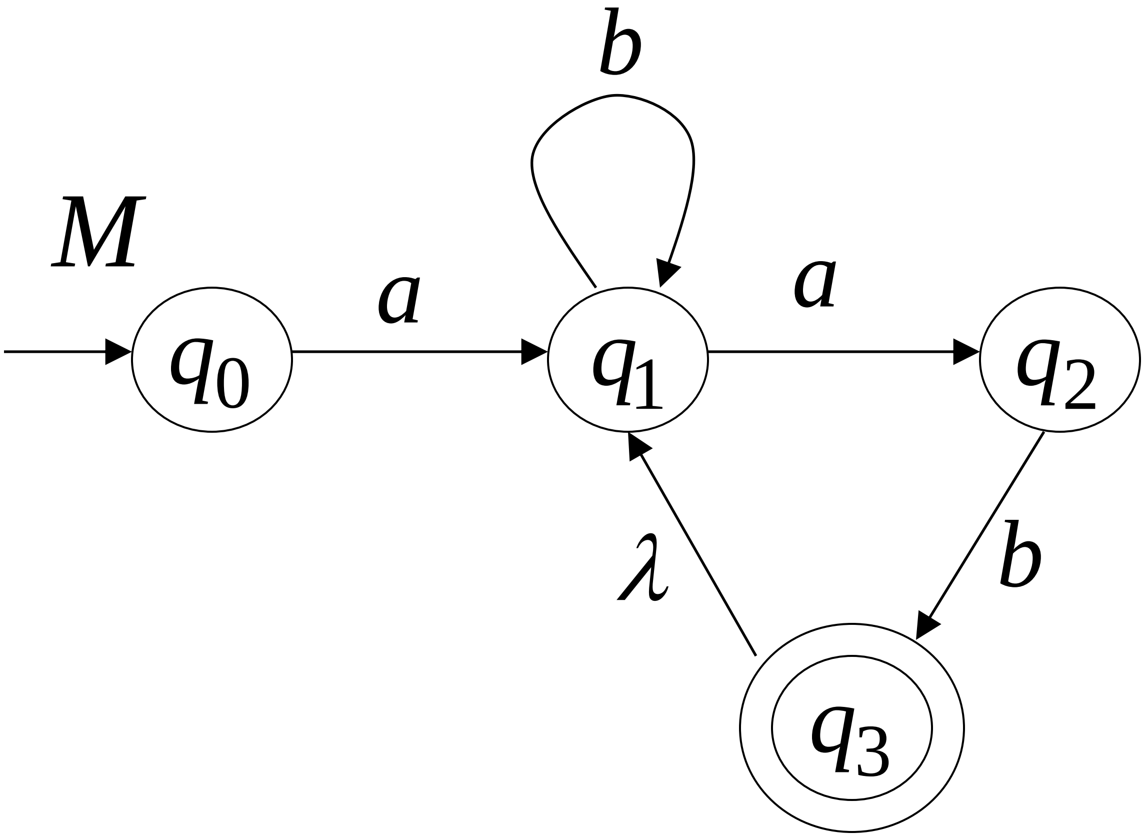
$$q_0 \to aq_1$$



$$q_0 \to aq_1$$

$$q_1 \to bq_1$$

$$q_1 \to aq_2$$



$$q_0 \to aq_1$$

$$q_1 \to bq_1$$

$$q_1 \to aq_2$$

$$q_2 \to bq_3$$

$$L(G) = L(M) = L$$

G

$q_0 \rightarrow aq_1$

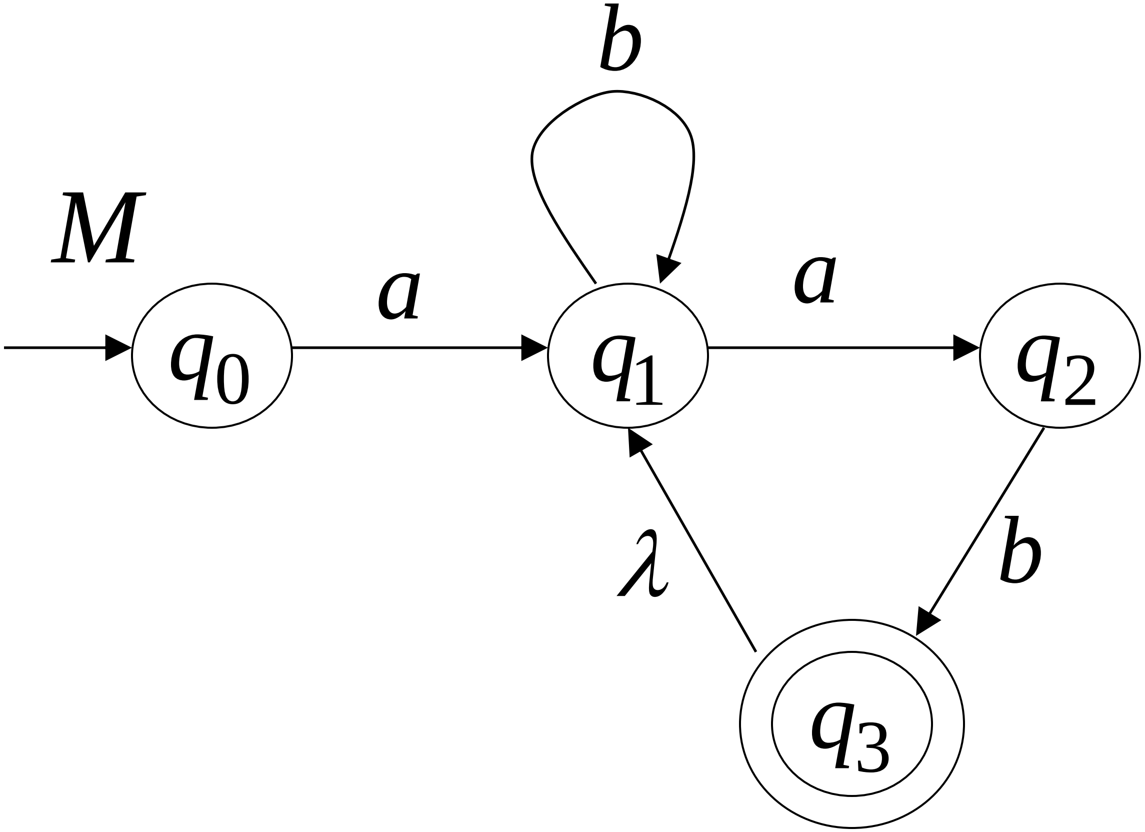
$q_1 \rightarrow bq_1$

$q_1 \rightarrow aq_2$

$q_2 \rightarrow bq_3$

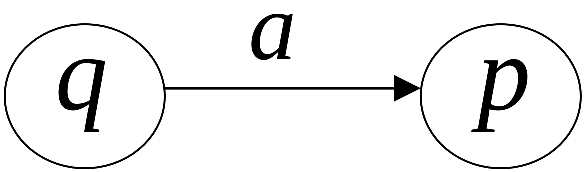
$q_3 \rightarrow q_1$

$q_3 \rightarrow \lambda$

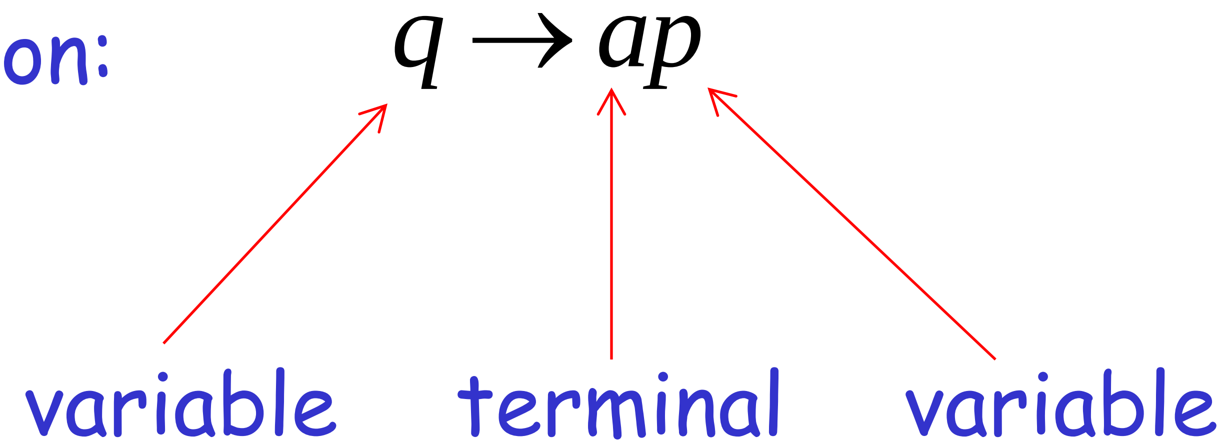


In General

For any transition:



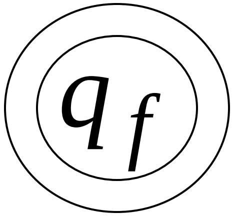
Add production:



$$q \rightarrow ap$$

variable terminal variable

For any final state:



Add production: $q_f \rightarrow \lambda$

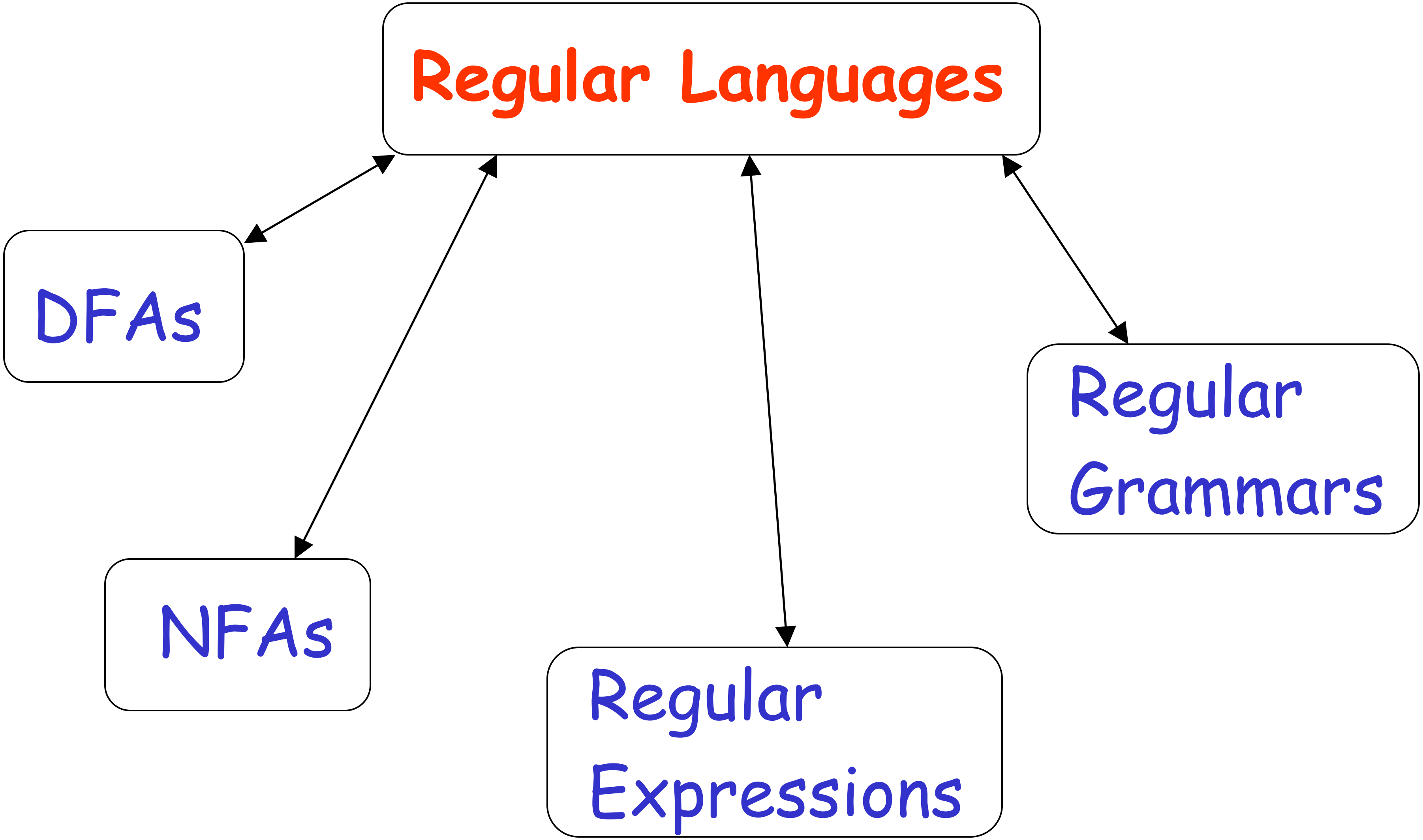
Since G is right-linear grammar

G is also a regular grammar

with $L(G) = L(M) = L$

The End

Standard Representations of Regular Languages



Regular Languages

DFAs

NFAs

Regular Expressions

Regular Grammars

When we say: We are given a Regular Language L

We mean: Language L is in a standard representation

What are the differences among NFA/DFA, regular expression and regular grammar?

NFA/DFA accepts languages

Regular expresses operate languages

Grammar generates language

Elementary Questions

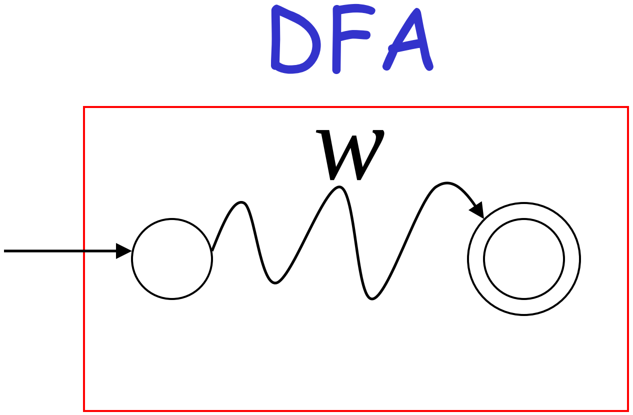
about

Regular Languages

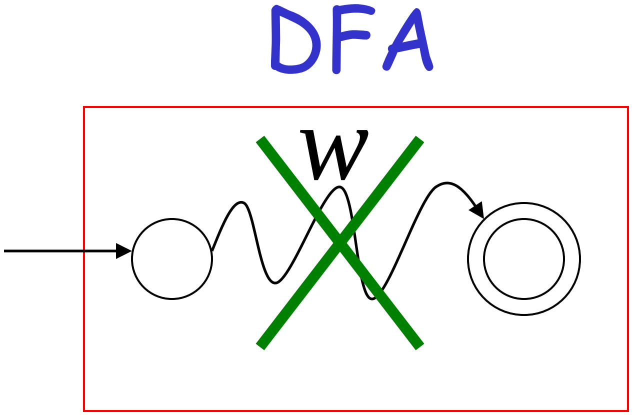
Membership Question

Question: Given regular language L
and string w
how can we check if $w \in L$?

Answer: Take the DFA that accepts L
and check if w is accepted



$$w \in L$$



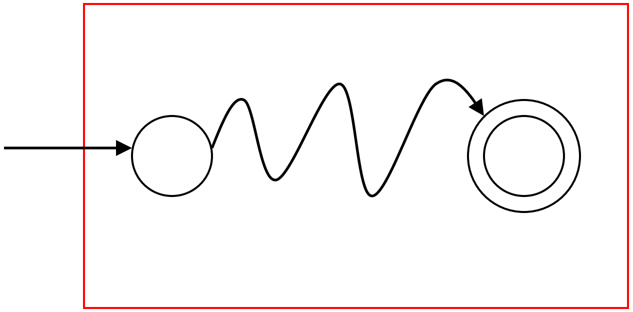
$$w \notin L$$

Question: Given regular language L
how can we check
if L is empty: $(L = \emptyset)$?

Answer: Take the DFA that accepts L

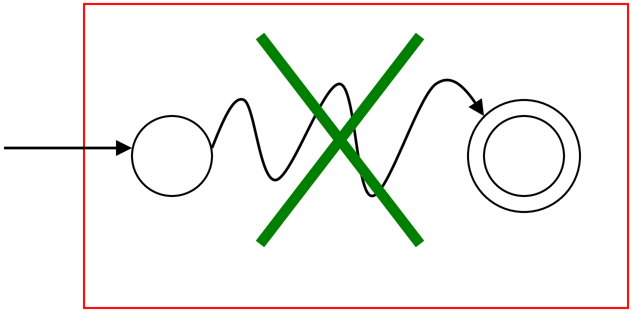
Check if there is any path from the initial state to a final state

DFA



$$L \neq \emptyset$$

DFA



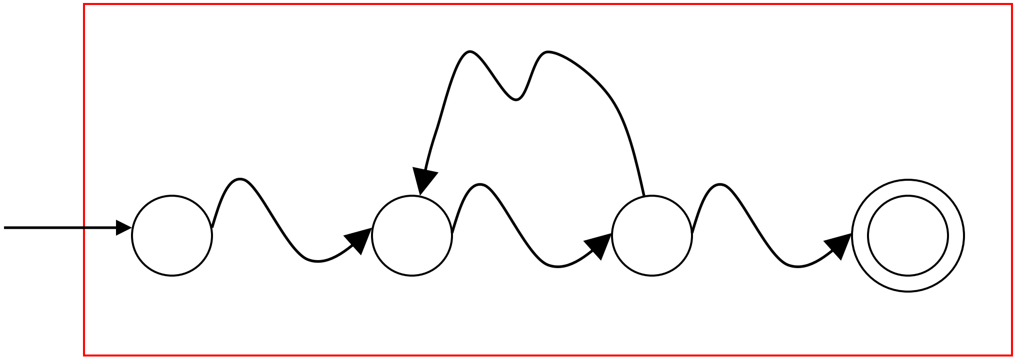
$$L = \emptyset$$

Question: Given regular language L
how can we check
if L is finite?

Answer: Take the DFA that accepts L

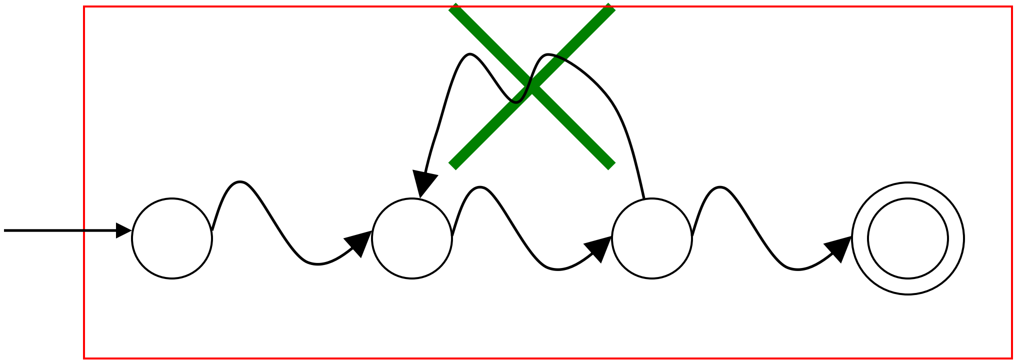
Check if there is a walk with cycle
from the initial state to a final state

DFA



L is infinite

DFA



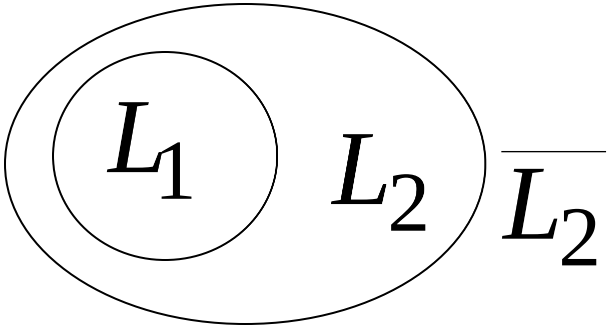
L is finite

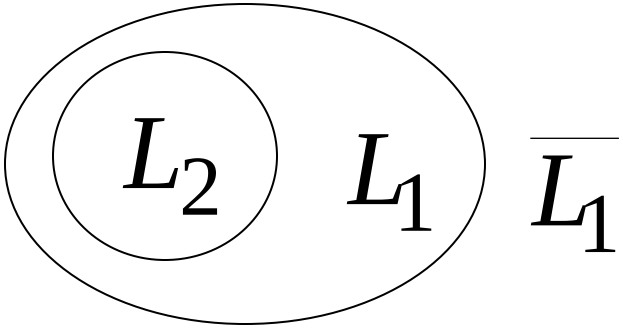
Question: Given regular languages L_1 and L_2 how can we check if $L_1 = L_2$?

Answer: Find if $(L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2) = \emptyset$

$$(L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2) = \varnothing$$

$L_1 \cap \overline{L_2} = \varnothing$ and $\overline{L_1} \cap L_2 = \varnothing$





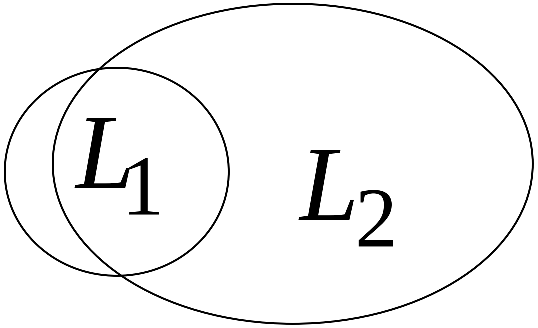
$L_1 \subseteq L_2$ $L_2 \subseteq L_1$

$$L_1 = L_2$$

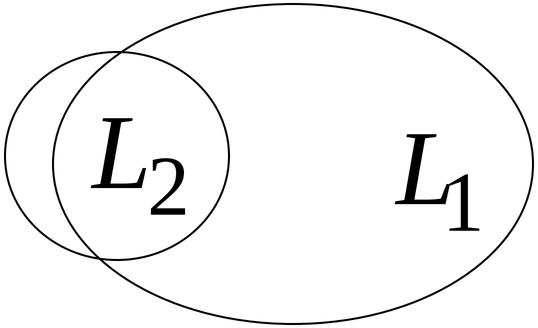
$$(L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2) \neq \emptyset$$

$$\Downarrow$$

$L_1 \cap \overline{L_2} \neq \emptyset$ **or** $\overline{L_1} \cap L_2 \neq \emptyset$



$$L_1 \not\subset L_2$$



$$L_2 \not\subset L_1$$

$$\Downarrow$$

$$L_1 \neq L_2$$

Non-regular languages

Non-regular languages

$$\{a^n b^n : \ n \geq 0\}$$

$$\{vv^R : \ v \in \{a,b\}^*\}$$

Regular languages

$$a^*b$$

$$b^*c + a$$

$$b + c(a+b)^*$$

$$etc...$$

Finite languages are regular

How can we prove that a language L is not regular?

Prove that there is no DFA that accepts L

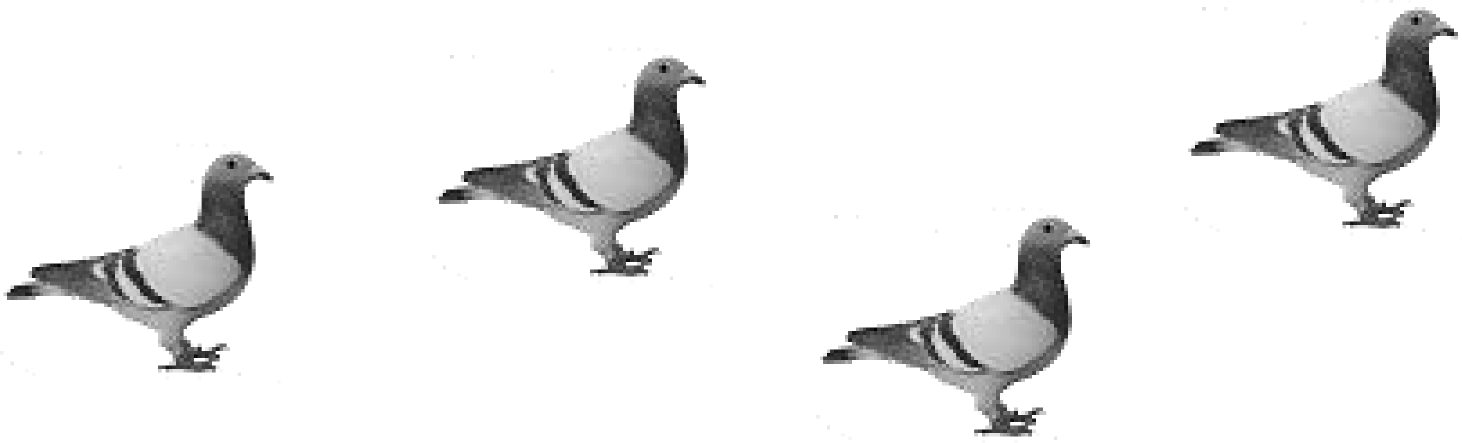
Ha Ha Ha……^-^

Problem: this is not easy to prove

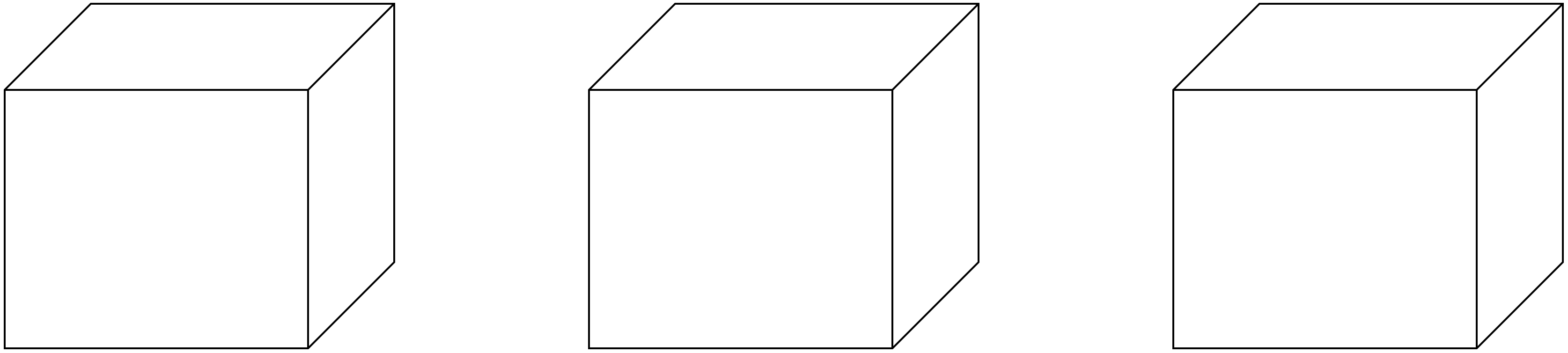
Solution: the Pumping Lemma !!!

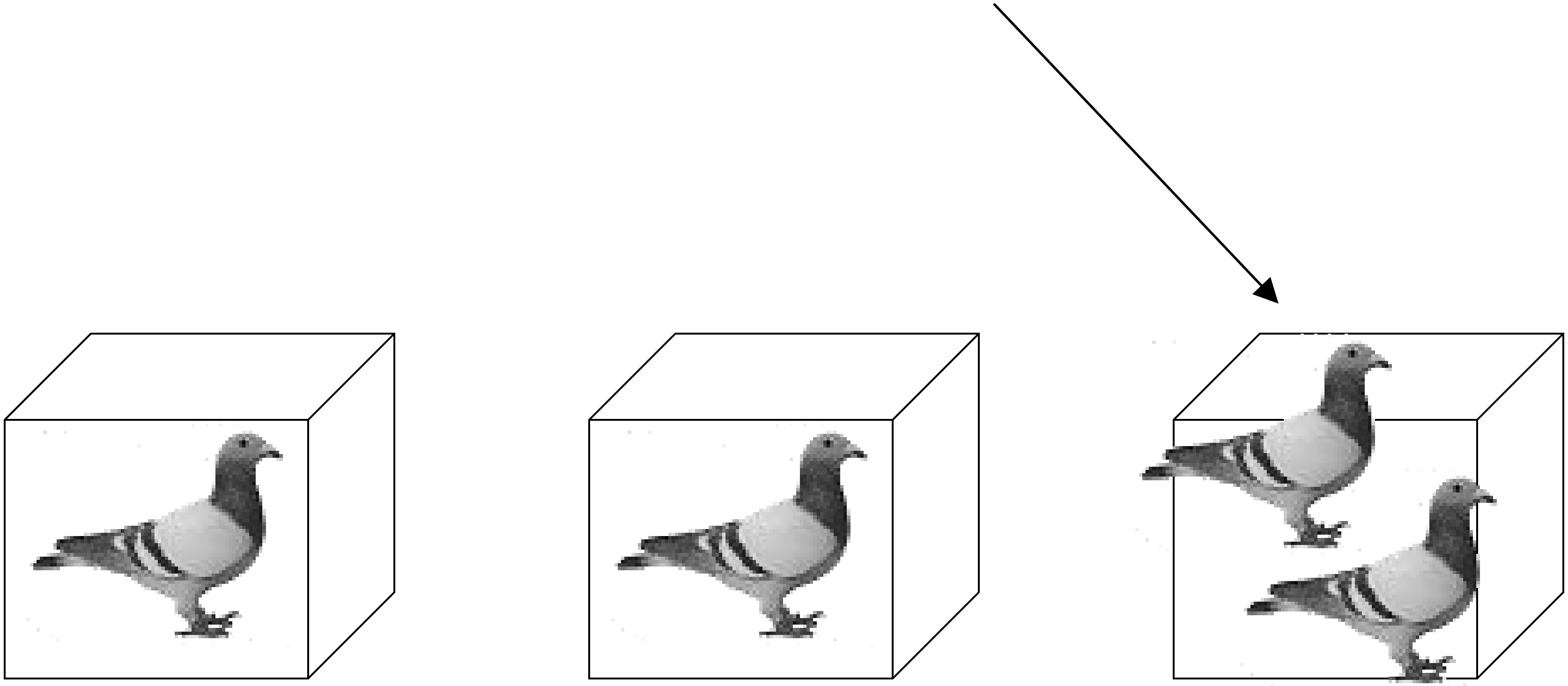
The Pigeonhole Principle

4 pigeons



3 pigeonholes

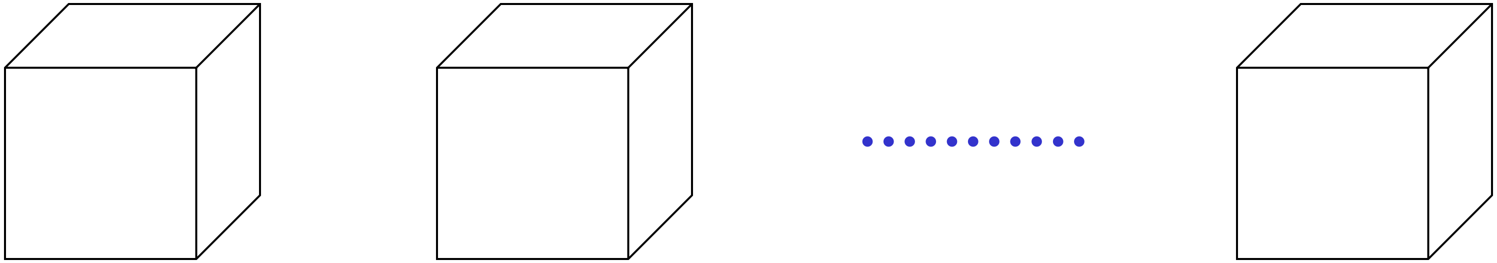




A pigeonhole must contain at least two pigeons

n pigeons

m pigeonholes $n > m$



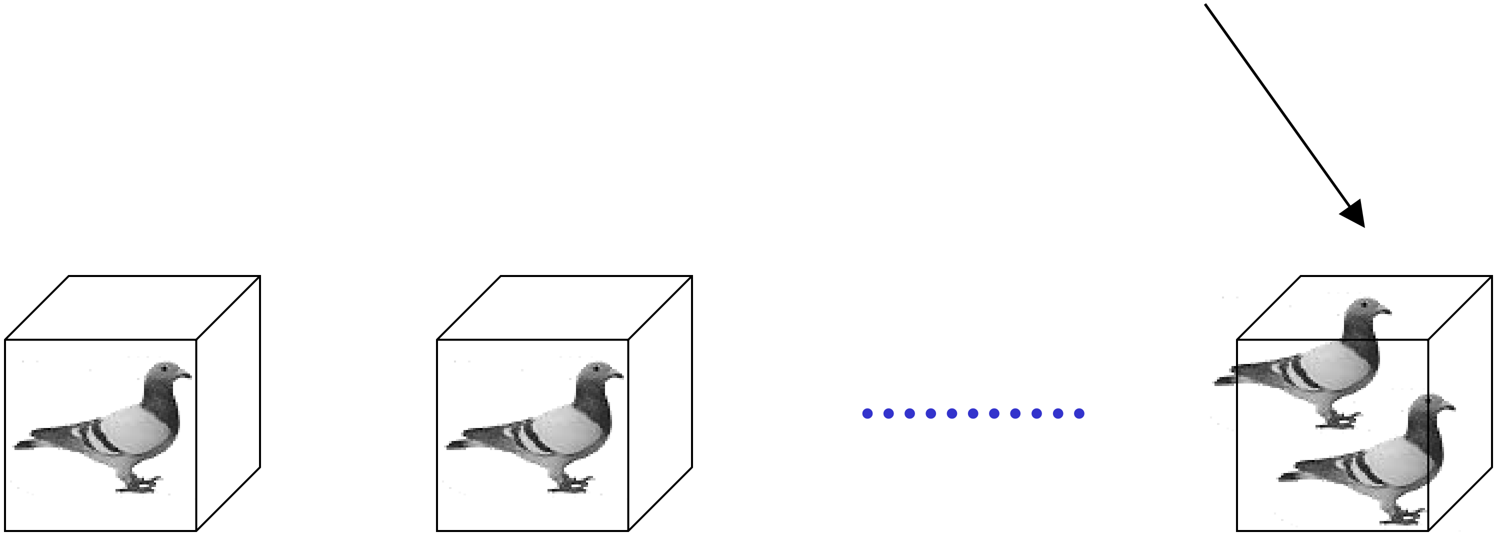
The Pigeonhole Principle

n pigeons

m pigeonholes

$$n > m$$

There is a pigeonhole with at least 2 pigeons



Ex 1: Show that if any five numbers from 1 to 8 are chosen, then two of them will add up to 9.

Solution: {1,8},{2,7},{3,6},{4,5}

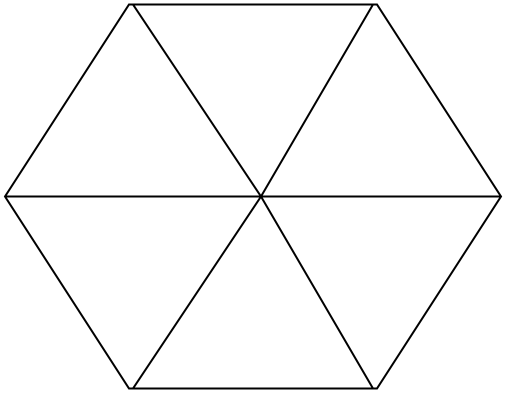
Ex 2: show that if any 11 numbers are chosen from the set {1,2,…,20}, then one of them will be a multiple of another.

Solution: {1,2,…,20}={$2^k m$|k could be any positive integer including 0, m=some odd number}, odd number={k=0,m=1,3,…,19}, 2={k=1,m=1},…,12={k=2,m=3},…,18={k=1,m=9}

…

$2^{k1}m$, $2^{k2}m$

Ex 3: Consider the region shown in the figure. It is bounded by a regular hexagon whose sides are of length 1 unit. Show that if any seven points are chosen in this region, then two of them must be no farther apart than 1 unit.



Ex 4: shirts numbered consecutively from 1 to 20 are worn by the 20 members of a bowling league. When any 3 of these members are chosen to be a team, the sum of their shirt numbers is used as a code number for the team. Show that if any 8 of the 20 members are selected, then from theses 8 we may form at least two different teams having the same code number. (one member can be in several different teams simultaneously)

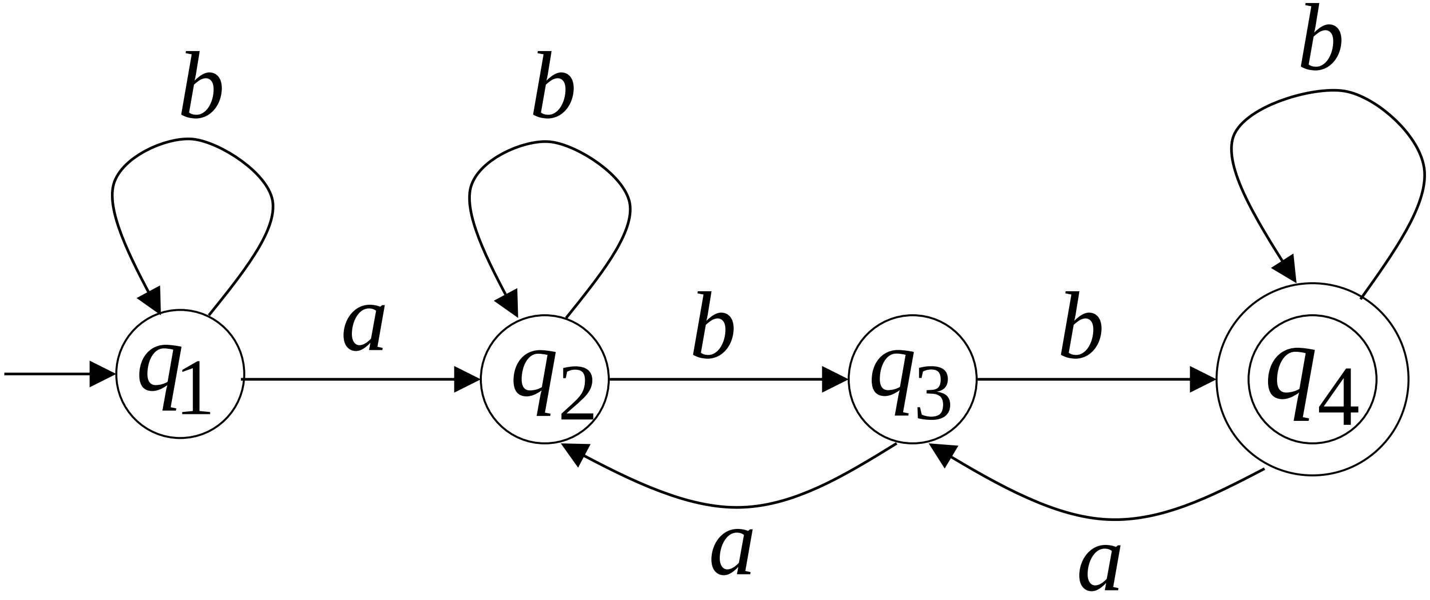
Solution: C(8,3)=56, 1+2+3=6,…,18+19+20=57, 57-6+1=52

The Pigeonhole Principle

and

DFAs

DFA with 4 states

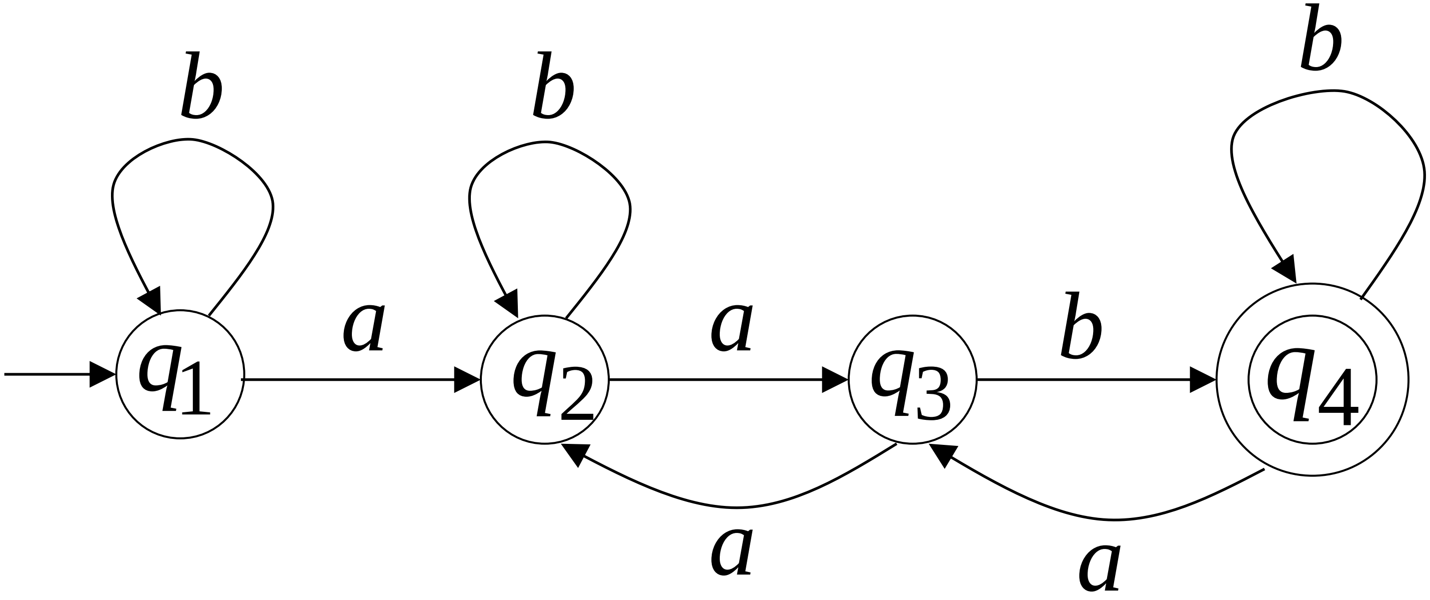


In walks of strings: a

aa

aab

no state is repeated



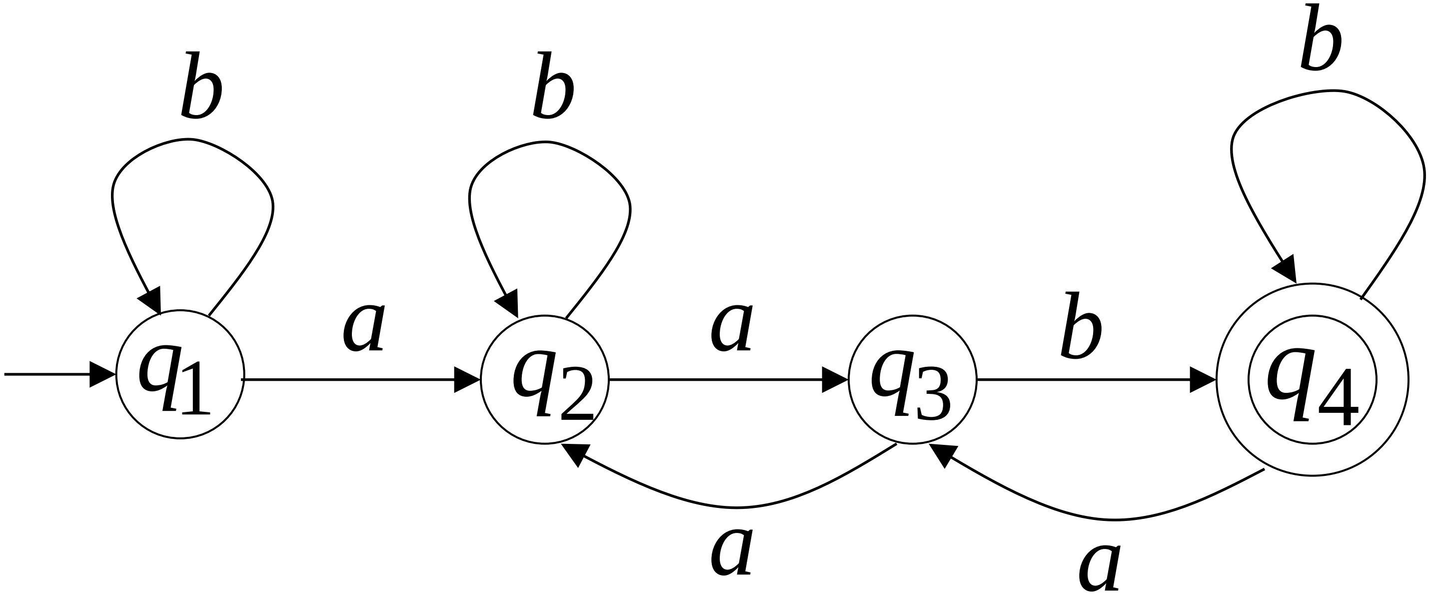
In walks of strings: $aabb$

$bbaa$

$abbabb$

$abbbabbabb...$

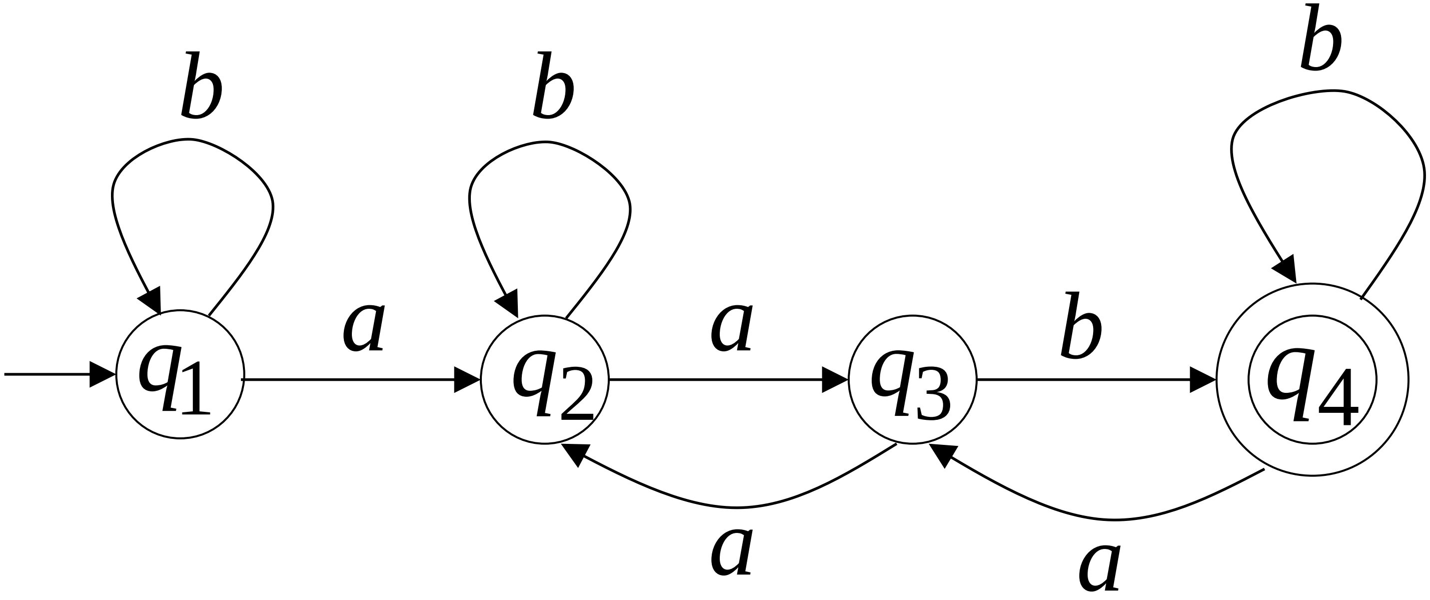
a state is repeated



If string w has length $|w| \geq 4$:

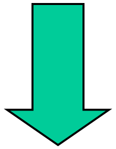
Then the transitions of string w are more than the states of the DFA

Thus, a state must be repeated

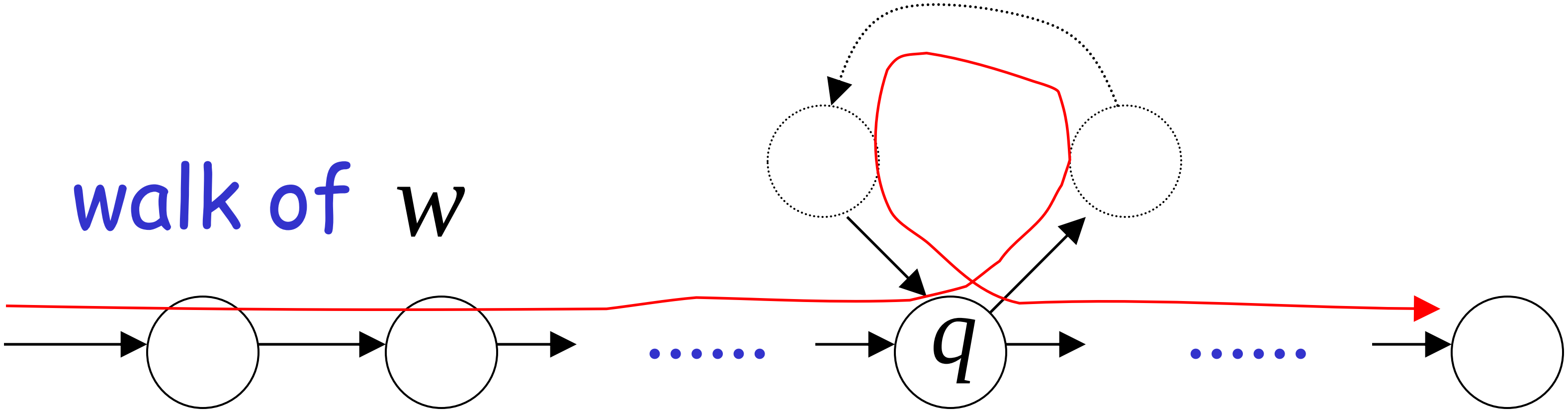


In general, for any DFA:

String w has length $\geq$ number of states



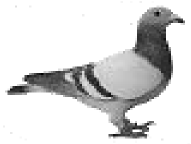
A state q must be repeated in the walk of w

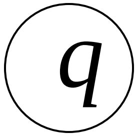


Repeated state

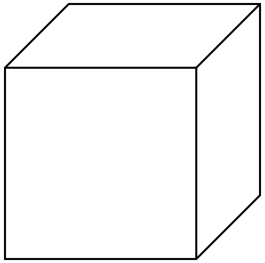
In other words for a string w:

$\xrightarrow{a}$ transitions are pigeons

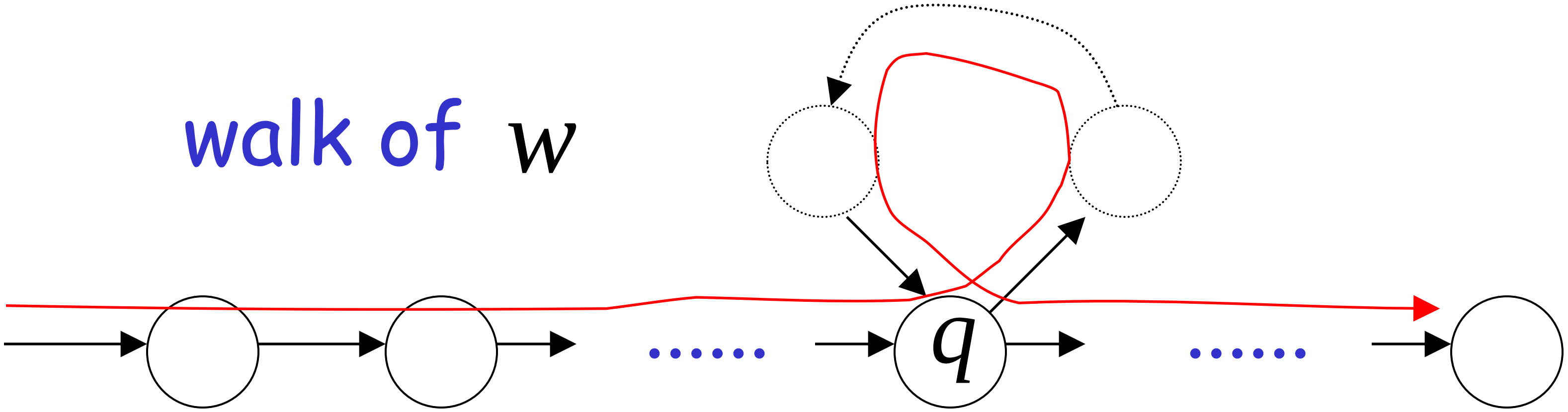




states are pigeonholes



walk of w



Repeated state

The Pumping Lemma

The Pumping Lemma:

- Given a infinite regular language L
- there exists an integer m
- for any string $w \in L$ with length $|w| \geq m$
- we can write $w = x\,y\,z$
- with $|x\,y| \leq m$ and $|y| \geq 1$
- such that: $x\,y^i\,z \in L \qquad i = 0, 1, 2, \ldots$

Applications

of

the Pumping Lemma

Theorem: The language $L = \{a^n b^n : n \geq 0\}$

is not regular

Proof: Use the Pumping Lemma

$$L = \{a^n b^n : n \geq 0\}$$

Assume for contradiction
that L is a regular language

Since L is infinite
we can apply the Pumping Lemma

$$L = \{a^n b^n : n \geq 0\}$$

Let m be the integer in the Pumping Lemma

Pick a string w such that: $w \in L$

length $|w| \geq m$

We pick $w = a^m b^m$

Write: $a^m b^m = x\,y\,z$

From the Pumping Lemma

it must be that length $|x\,y| \le m, \quad |y| \ge 1$

$$xyz = a^m b^m \;=\; \overbrace{a\ldots a a\ldots a a\ldots a}^{m}\overbrace{b\ldots b}^{m}$$

$$\underbrace{a\ldots a}_{x}\underbrace{a\ldots a}_{y}\underbrace{a\ldots a b\ldots b}_{z}$$

Thus: $y = a^k, \quad k \ge 1$

$x\ y\ z = a^m b^m$ $\qquad$ $y = a^k, \quad k \geq 1$

From the Pumping Lemma: $\quad x\ y^i\ z \ \in\ L$

$$i = 0, 1, 2, \ldots$$

Thus: $\quad x\ y^2\ z \ \in\ L$

$$x\,y\,z = a^m b^m \qquad\qquad y = a^k, \quad k \geq 1$$

From the Pumping Lemma: $x\,y^2\,z \in L$

$$xy^2z = \rlap{$\overbrace{}^{m+k}$}\underbrace{a...a}_{x}\underbrace{a...a}_{y}\underbrace{a...a}_{y}\underbrace{a...a\overbrace{b...b}^{m}}_{z} \in L$$

Thus: $a^{m+k}b^m \in L$

$$a^{m+k}b^m \in L \qquad\qquad k \geq 1$$

BUT: $L = \{a^n b^n : n \geq 0\}$

$$\Downarrow$$

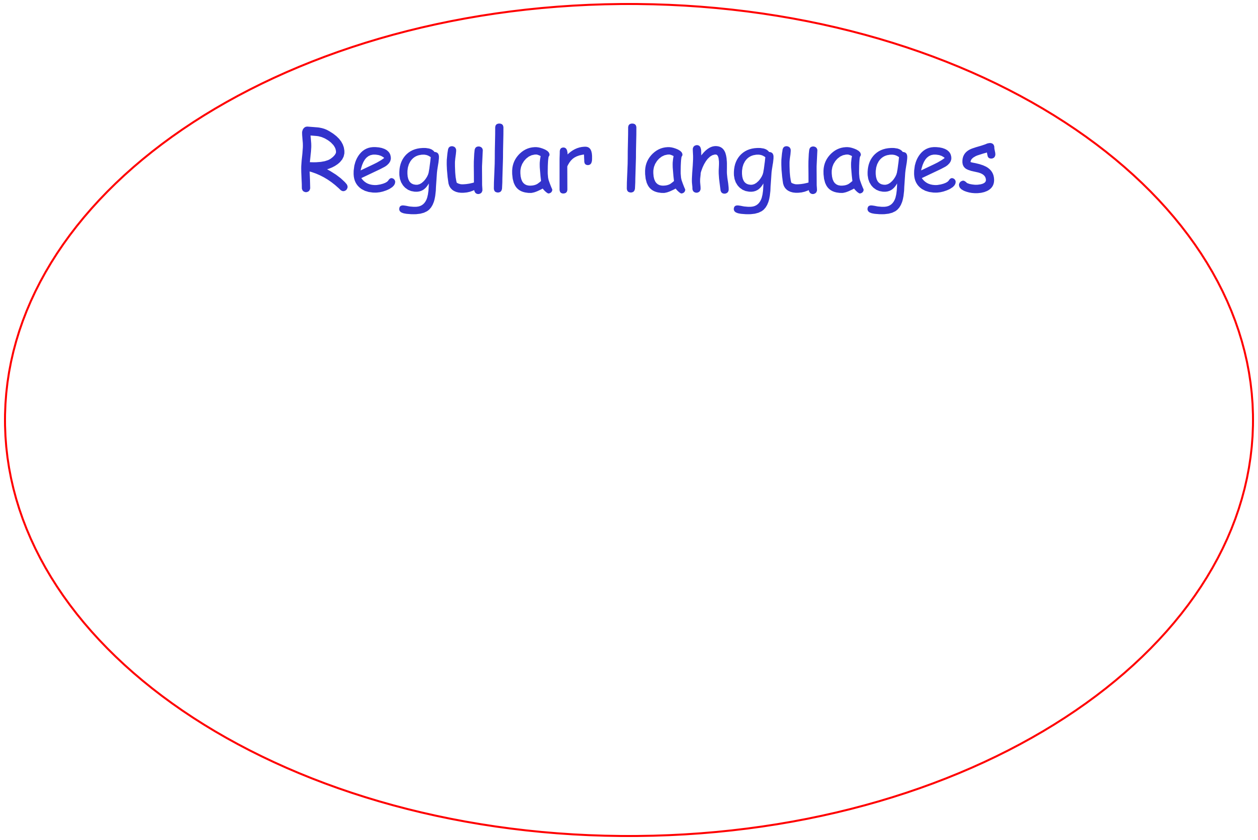
$$a^{m+k}b^m \notin L$$

CONTRADICTION!!!

Therefore: Our assumption that L is a regular language is not true

Conclusion: L is not a regular language

Non-regular languages $\{a^n b^n : \quad n \geq 0\}$

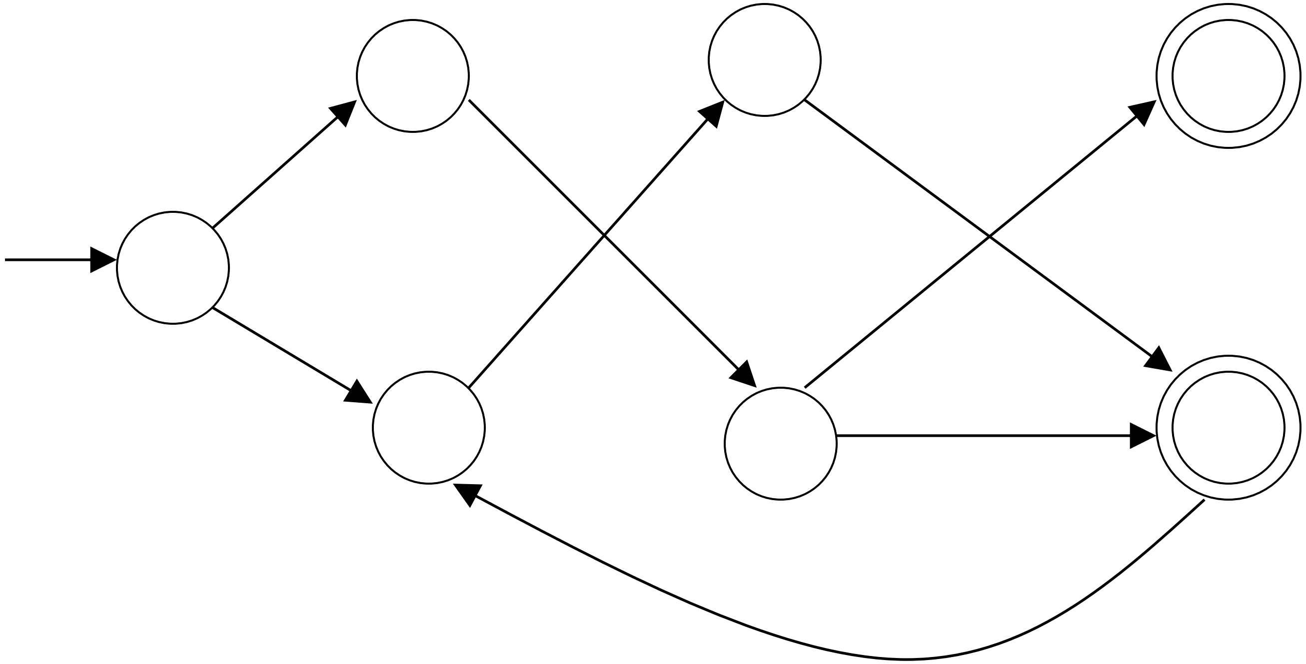


Regular languages

Understanding pumping lemma more

Take an infinite regular language L

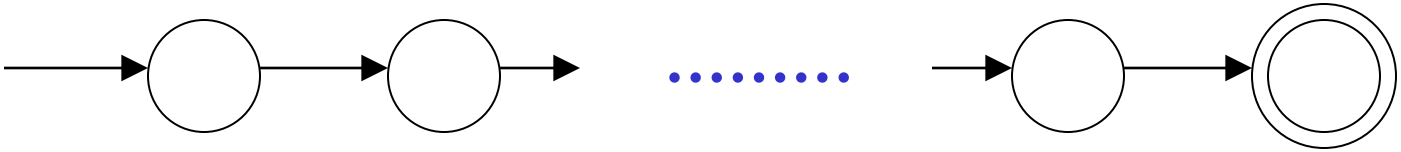
There exists a DFA that accepts L



m states

Take string w with $w \in L$

There is a walk with label w :

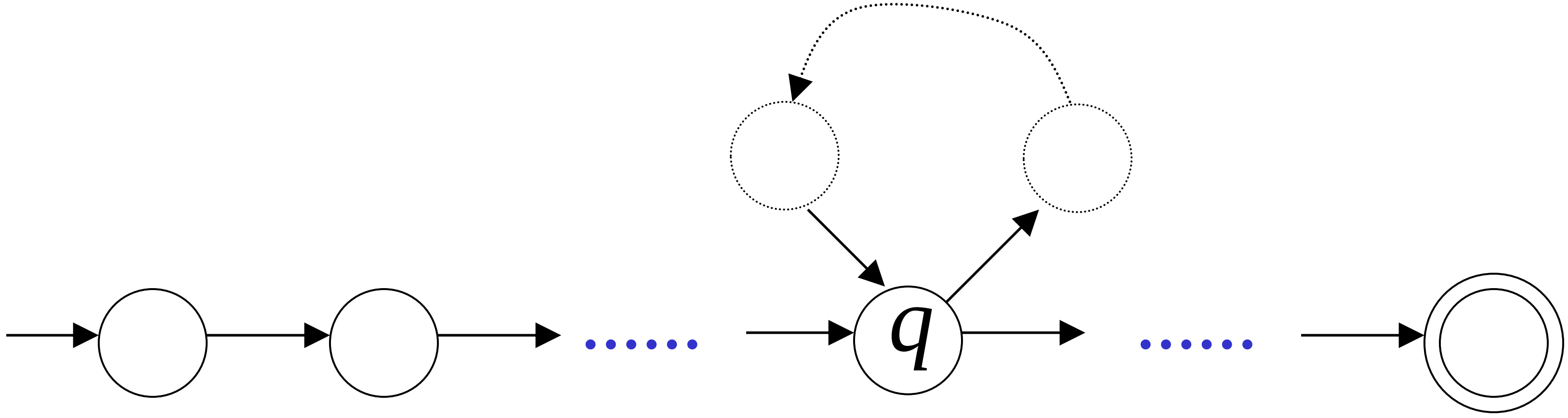


walk w

If string w has length $|w| \geq m$ (number of states of DFA)

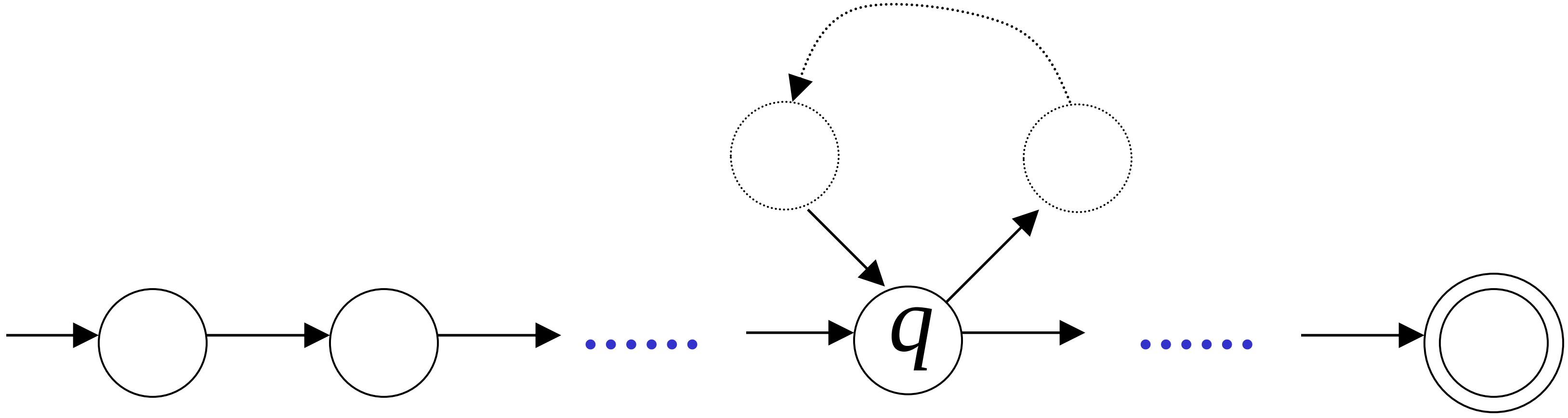
then, from the pigeonhole principle:

a state is repeated in the walk w



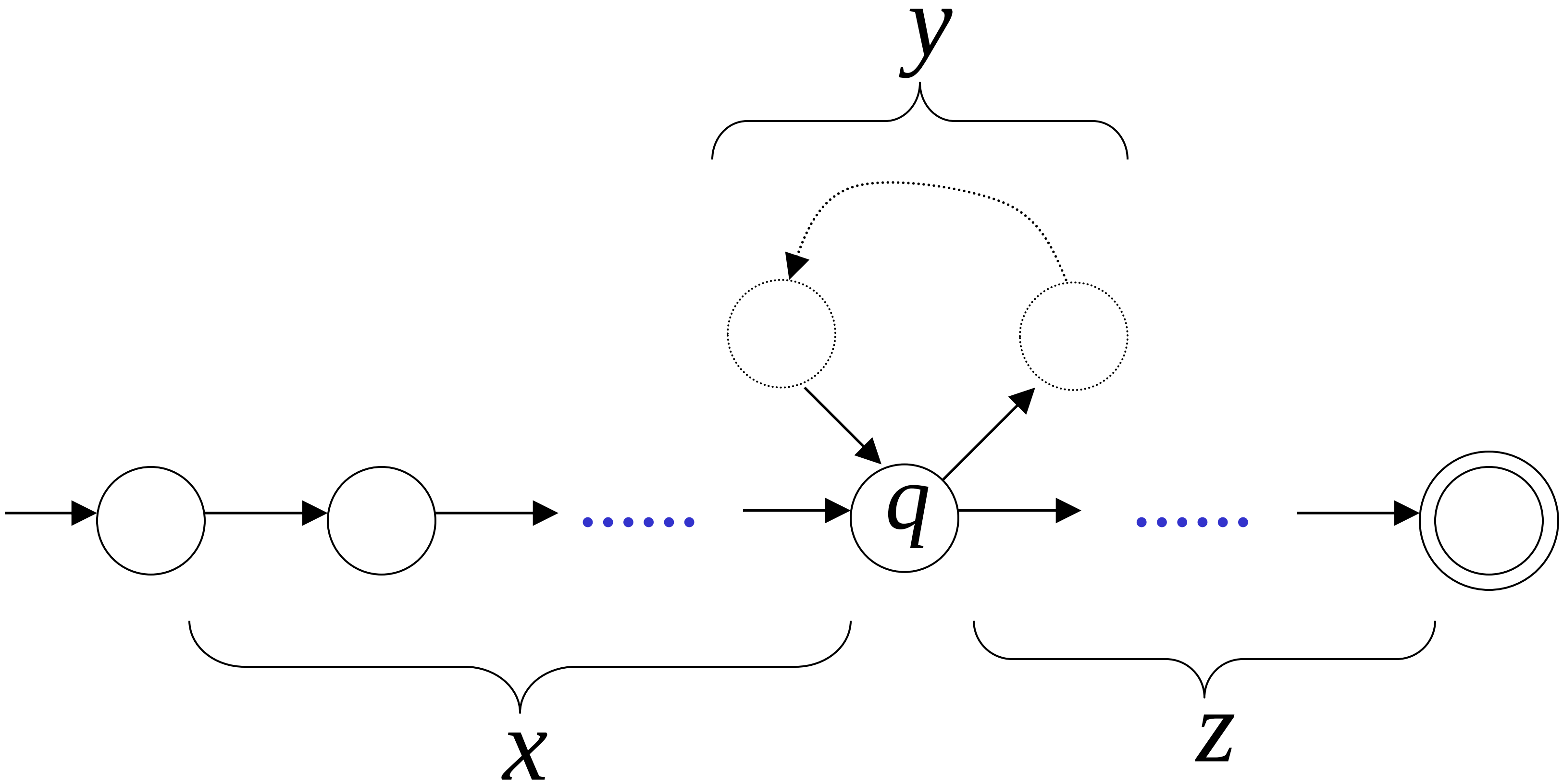
walk w

Let q be the first state repeated in the walk of w



walk w

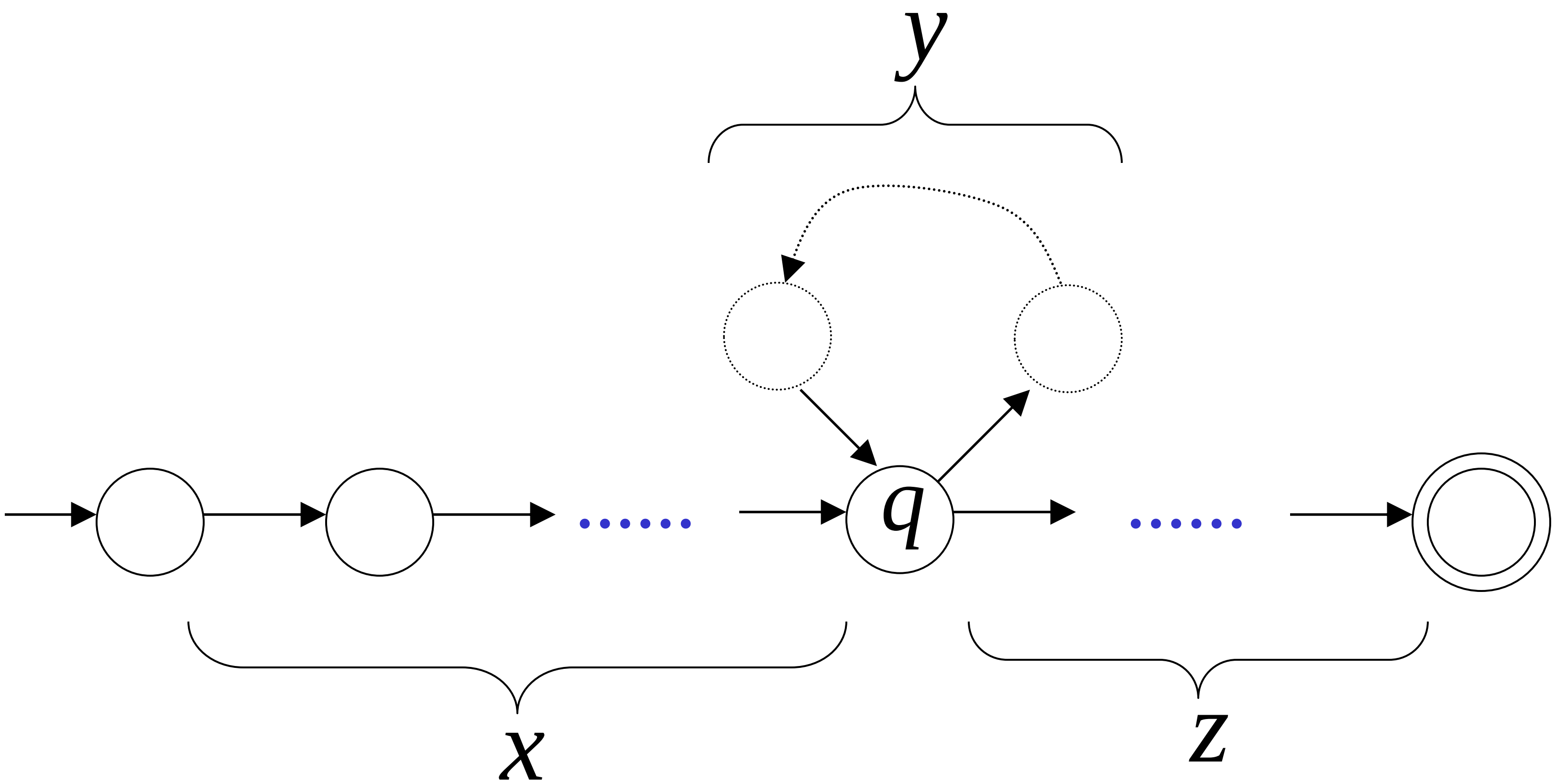
Write $w = x\ y\ z$



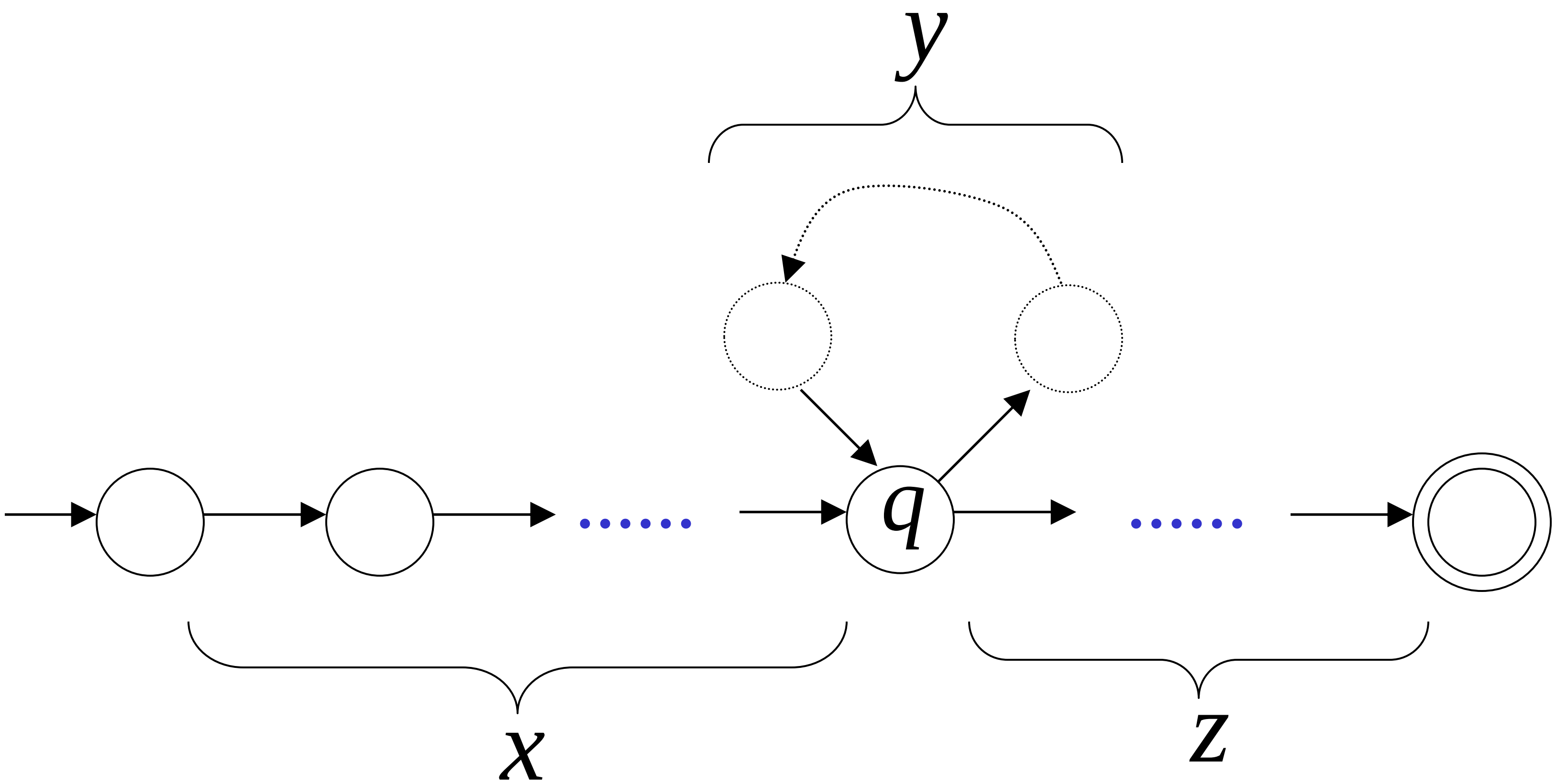
Observations:

length $|x\,y| \leq m$ number of states of DFA

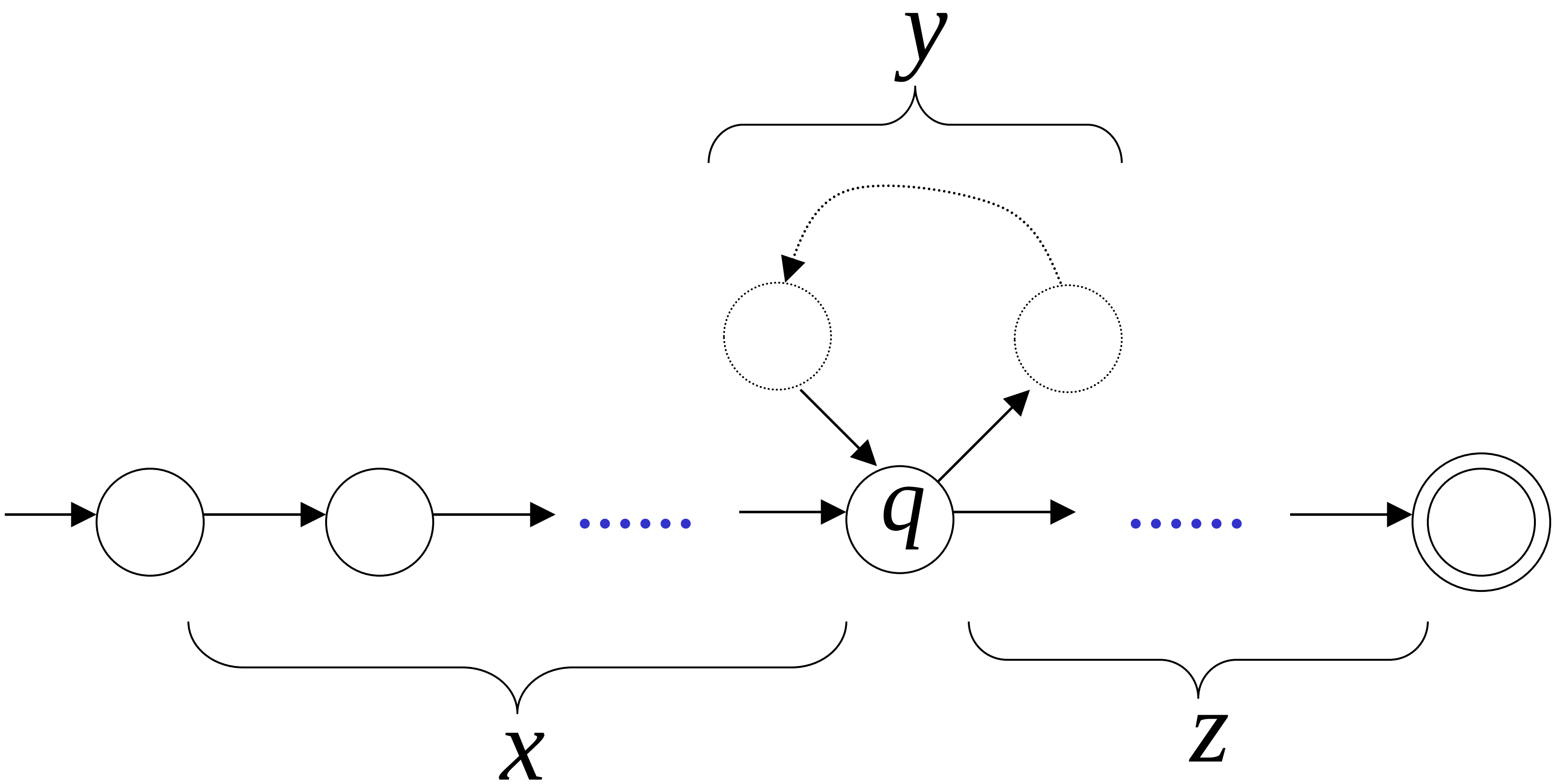
length $|y| \geq 1$



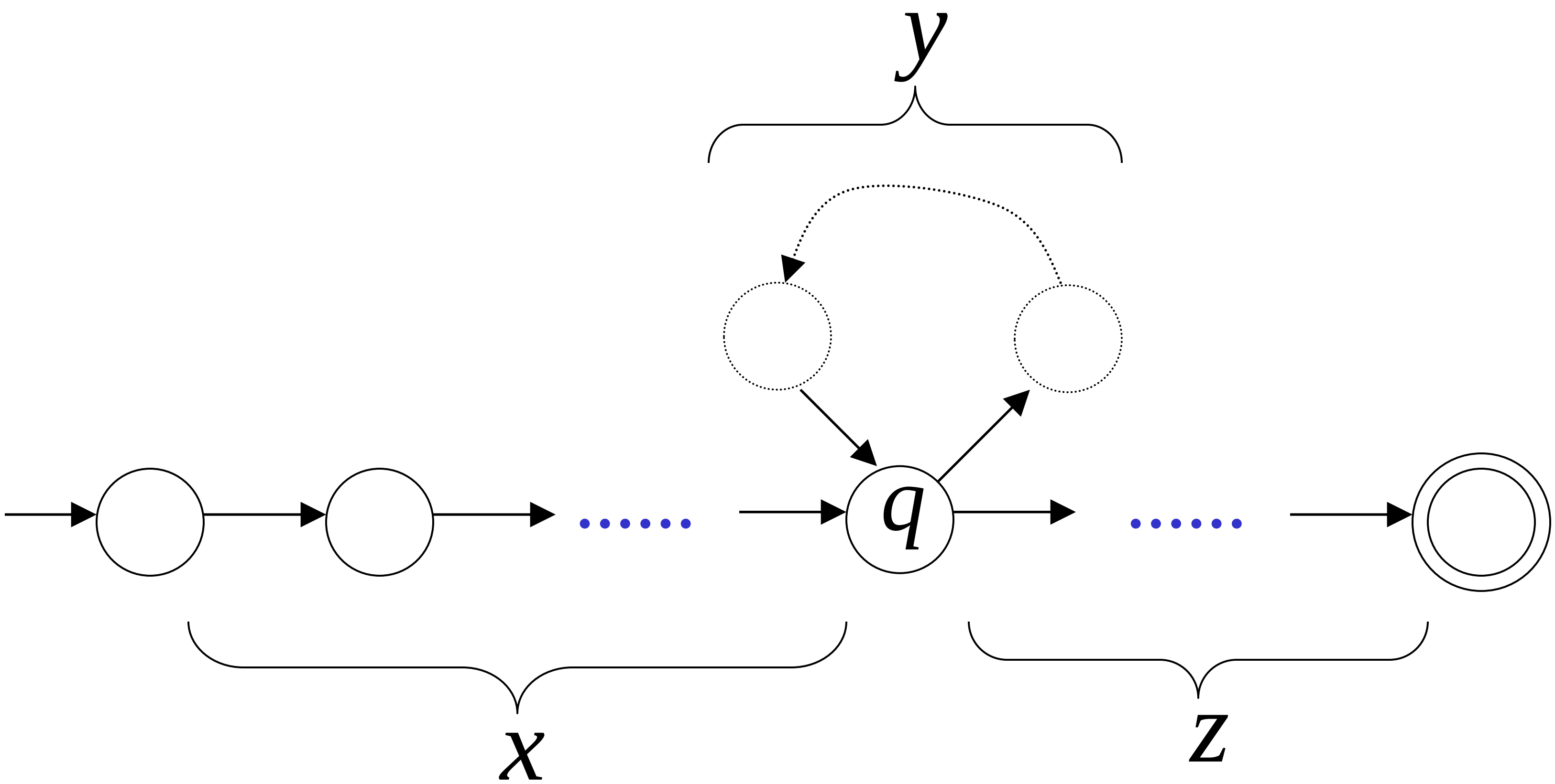
Observation: The string $x\ z$ is accepted



Observation: The string $x\ y\ y\ z$ is accepted

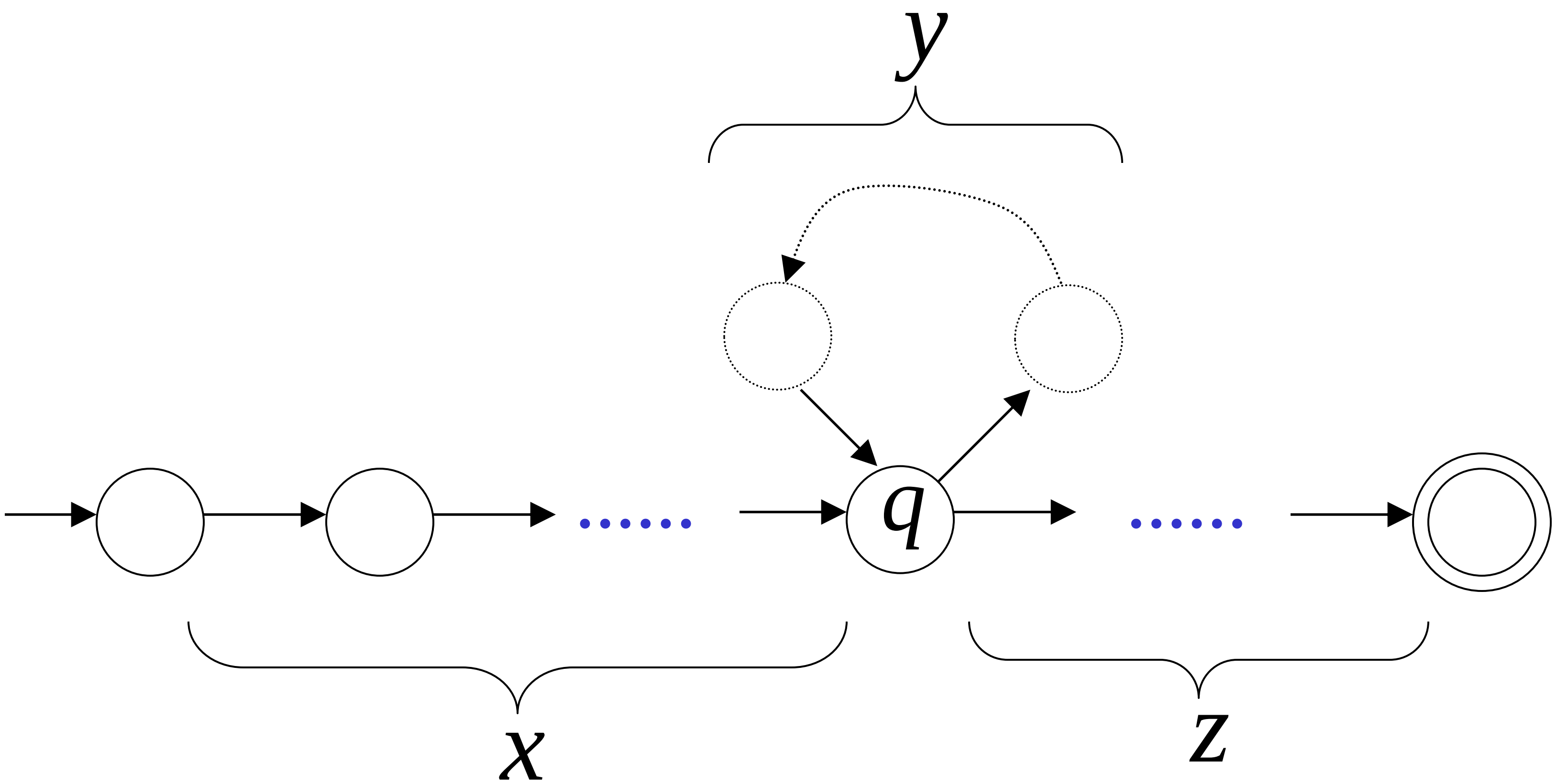


Observation: The string $x\,y\,y\,y\,z$ is accepted



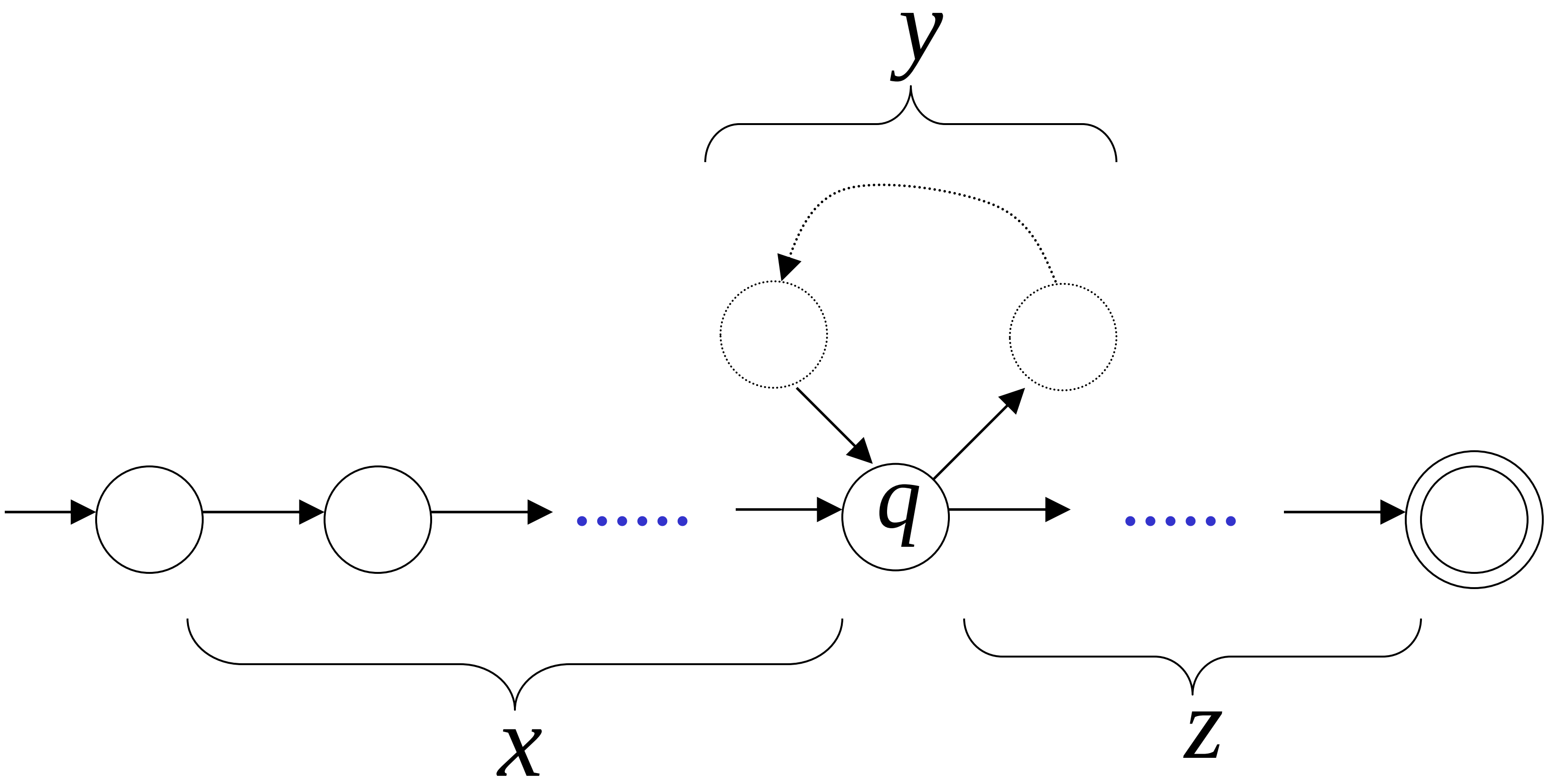
In General: The string $x\,y^i\,z$

is accepted $i = 0, 1, 2, ...$



In General: $x\,y^i\,z \in L \qquad i = 0, 1, 2, \ldots$

Language accepted by the DFA



In other words, we described:

The Pumping Lemma !!!

More Applications

of

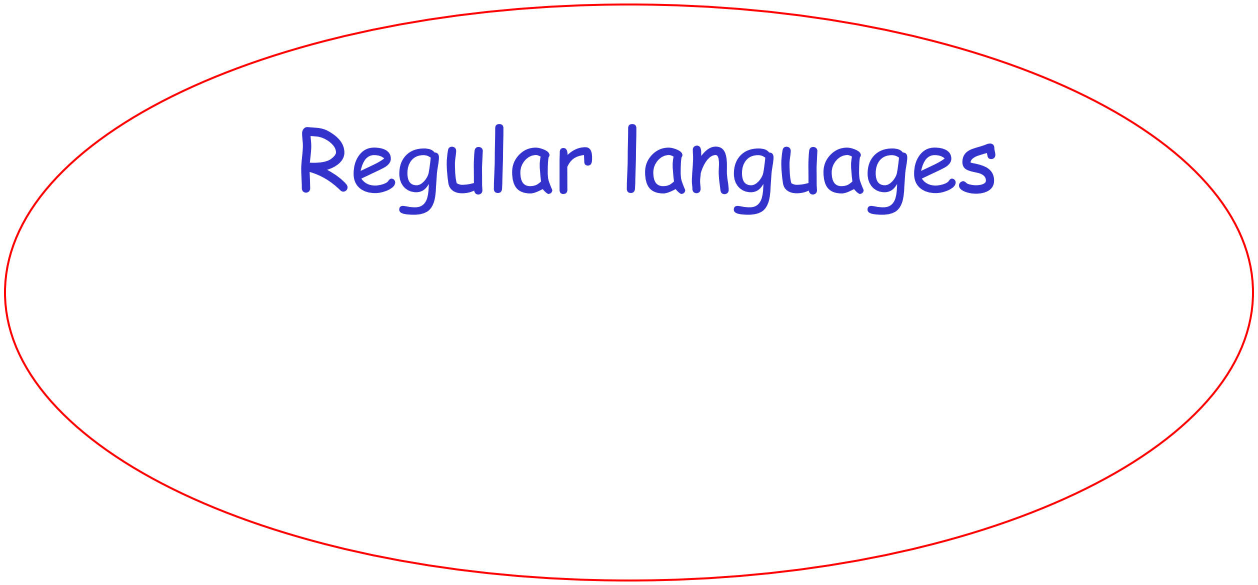
the Pumping Lemma

The Pumping Lemma:

- Given a infinite regular language L
- there exists an integer m
- for any string $w \in L$ with length $|w| \geq m$
- we can write $w = x\,y\,z$
- with $|x\,y| \leq m$ and $|y| \geq 1$
- such that: $x\,y^i\,z \in L \qquad i = 0, 1, 2, \ldots$

Non-regular languages

$$L = \{vv^R : v \in \Sigma^*\}$$



Regular languages

Theorem: The language

$$L = \{vv^R : v \in \Sigma^*\} \qquad \Sigma = \{a, b\}$$

is not regular

Proof: Use the Pumping Lemma

$$L = \{vv^R : v \in \Sigma^*\}$$

Assume for contradiction that L is a regular language

Since L is infinite we can apply the Pumping Lemma

$$L = \{vv^R : v \in \Sigma^*\}$$

Let m be the integer in the Pumping Lemma

Pick a string w such that: $w \in L$ and

length $|w| \geq m$

We pick $w = a^m b^m b^m a^m$

Write $a^m b^m b^m a^m = x\ y\ z$

From the Pumping Lemma
it must be that length $|x\ y| \leq m, \quad |y| \geq 1$

$$xyz = \rlap{$\overbrace{}^{m}$}\underbrace{a\ldots a}_{x}\underbrace{a\ldots a}_{y}\underbrace{\ldots a\overbrace{b\ldots b}^{m}\overbrace{b\ldots b}^{m}\overbrace{a\ldots a}^{m}}_{z}$$

Thus: $y = a^k, \quad k \geq 1$

$$x\,y\,z = a^m b^m b^m a^m \qquad y = a^k, \quad k \geq 1$$

From the Pumping Lemma: $x\,y^i\,z \in L$

$$i = 0, 1, 2, \ldots$$

Thus: $x\,y^2\,z \in L$

$x\,y\,z = a^m b^m b^m a^m$ $\qquad$ $y = a^k, \quad k \geq 1$

From the Pumping Lemma: $\quad x\,y^2\,z \in L$

$$xy^2z = \overbrace{\underbrace{a...a}_{x}\underbrace{a...a}_{y}\underbrace{a...a}_{y}}^{m+k}\underbrace{...a\overbrace{b...b}^{m}\overbrace{b...b}^{m}\overbrace{a...a}^{m}}_{z} \in L$$

Thus: $\quad a^{m+k} b^m b^m a^m \in L$

$$a^{m+k}b^m b^m a^m \quad \in L \qquad\qquad k \geq 1$$

BUT: $L = \{vv^R : v \in \Sigma^*\}$

$$\Downarrow$$

$$a^{m+k}b^m b^m a^m \quad \notin L$$

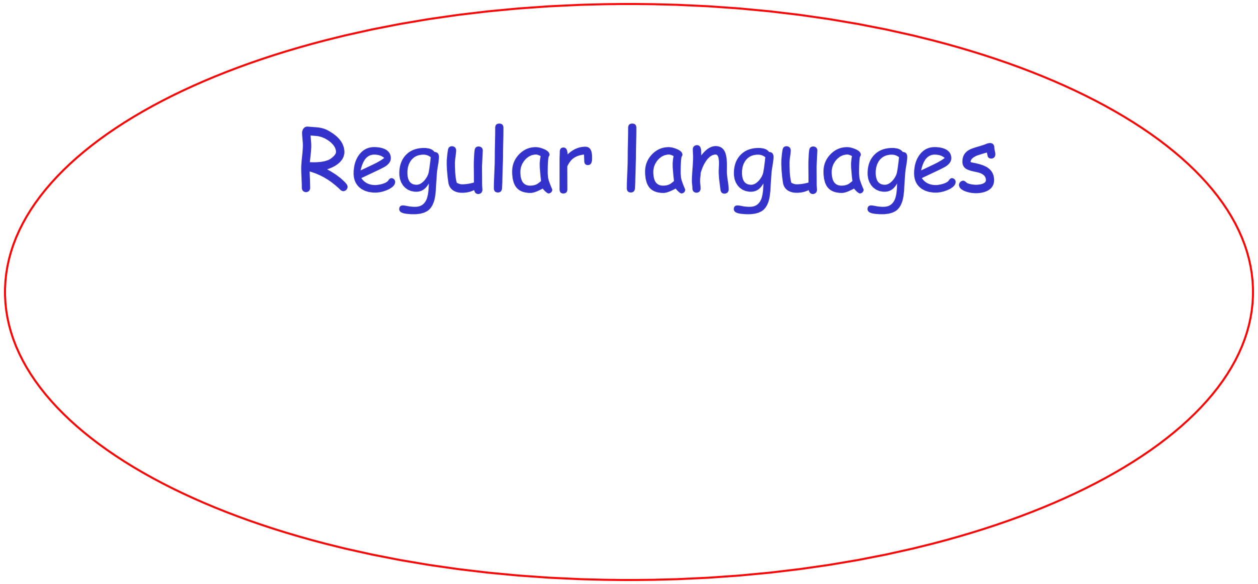
CONTRADICTION!!!

Therefore: Our assumption that L is a regular language is not true

Conclusion: L is not a regular language

Non-regular languages

$$L = \{a^n b^l c^{n+l} :\ n,l \geq 0\}$$



Regular languages

Theorem: The language

$$L = \{a^n b^l c^{n+l} : \ n, l \geq 0\}$$

is not regular

Proof: Use the Pumping Lemma

$$L = \{a^n b^l c^{n+l} : \; n, l \geq 0\}$$

Assume for contradiction
that L is a regular language

Since L is infinite
we can apply the Pumping Lemma

$$L = \{a^n b^l c^{n+l} : \; n, l \geq 0\}$$

Let m be the integer in the Pumping Lemma

Pick a string w such that: $w \in L$ and

length $|w| \geq m$

We pick $w = a^m b^m c^{2m}$

Write $a^m b^m c^{2m} = x\, y\, z$

From the Pumping Lemma
it must be that length $|\, x\, y \,| \leq m, \quad |\, y \,| \geq 1$

$$xyz = \overbrace{\underbrace{a\ldots a}_{x}\underbrace{a\ldots a}_{y}\mathrlap{\underbrace{}_{z}}a\ldots a}^{m}\overbrace{b\ldots b}^{m}\overbrace{c\ldots cc\ldots c}^{2m}$$

Thus: $y = a^k, \quad k \geq 1$

$$x\ y\ z = a^m b^m c^{2m} \qquad\qquad y = a^k, \quad k \geq 1$$

From the Pumping Lemma: $$x\ y^i\ z \ \in\ L$$

$$i = 0, 1, 2, \ldots$$

Thus: $$x\ y^0\ z\ = xz \ \in L$$

$x\,y\,z = a^m b^m c^{2m}$ $\qquad$ $y = a^k, \quad k \geq 1$

From the Pumping Lemma: $\quad xz \in L$

$$xz = \mathrlap{\overbrace{}^{m-k}}\underbrace{a...a}_{x}\underbrace{a...a\overbrace{b...b}^{m}\overbrace{c...cc...c}^{2m}}_{z} \in L$$

Thus: $\quad a^{m-k} b^m c^{2m} \in L$

$$a^{m-k}b^{m}c^{2m} \quad \in L \qquad\qquad k \geq 1$$

BUT: $$L = \{a^{n}b^{l}c^{n+l} : \ n, l \geq 0\}$$

⇓

$$a^{m-k}b^{m}c^{2m} \quad \notin L$$

CONTRADICTION!!!

Therefore: Our assumption that L is a regular language is not true

Conclusion: L is not a regular language

Non-regular languages

$$L = \{a^{n!} : \; n \geq 0\}$$

Regular languages

Theorem: The language $L = \{a^{n!} : \ n \geq 0\}$

is not regular

$$n! \ = \ 1 \cdot 2 \cdots (n-1) \cdot n$$

Proof: Use the Pumping Lemma

$$L = \{a^{n!} : \ n \geq 0\}$$

Assume for contradiction that L is a regular language

Since L is infinite we can apply the Pumping Lemma

$$L = \{a^{n!} : \ n \geq 0\}$$

Let m be the integer in the Pumping Lemma

Pick a string w such that: $w \in L$

length $|w| \geq m$

We pick $w = a^{m!}$

Write $a^{m!} = x\ y\ z$

From the Pumping Lemma

it must be that length $|x\ y| \leq m, \quad |y| \geq 1$

$$xyz = a^{m!} = \overbrace{\underbrace{a\ldots a}_{x}\underbrace{a\ldots a}_{y}\rlap{$\displaystyle\underbrace{}_{z}$}a\ldots a}^{m}\overbrace{a\ldots aa\ldots a}^{m!-m}$$

Thus: $y = a^k, \quad 1 \leq k \leq m$

$x\ y\ z = a^{m!}$ $\qquad$ $y = a^k, \quad 1 \le k \le m$

From the Pumping Lemma: $x\ y^i\ z \in L$

$$i = 0, 1, 2, \ldots$$

Thus: $x\ y^2\ z \in L$

$x\ y\ z = a^{m!}$ $\qquad$ $y = a^{k},\ \ 1 \le k \le m$

From the Pumping Lemma: $x\ y^{2}\ z \ \in\ L$

$$xy^{2}z = \overbrace{\underbrace{a...a}_{x}\underbrace{a...a}_{y}\underbrace{a...a}_{y}a...a}^{m+k}\overbrace{a...a}^{m!-m} \in\ L$$

$$\underbrace{a...aa...aa...a}_{z}$$

Thus: $a^{m!+k} \quad \in L$

$$a^{m!+k} \in L \qquad 1 \leq k \leq m$$

Since: $L = \{a^{n!} : n \geq 0\}$

⇓

There must exist p such that:

$$m! + k = p!$$

However: $m!+k \le m!+m$ for $m > 1$

$$\le m!+m!$$

$$< m!m + m!$$

$$= m!(m+1)$$

$$= (m+1)!$$

⇓

$$m!+k < (m+1)!$$

⇓

$m!+k \ne p!$ for any p

$$a^{m!+k} \in L \qquad 1 \le k \le m$$

BUT: $L = \{a^{n!} : n \ge 0\}$

⇓

$$a^{m!+k} \notin L$$

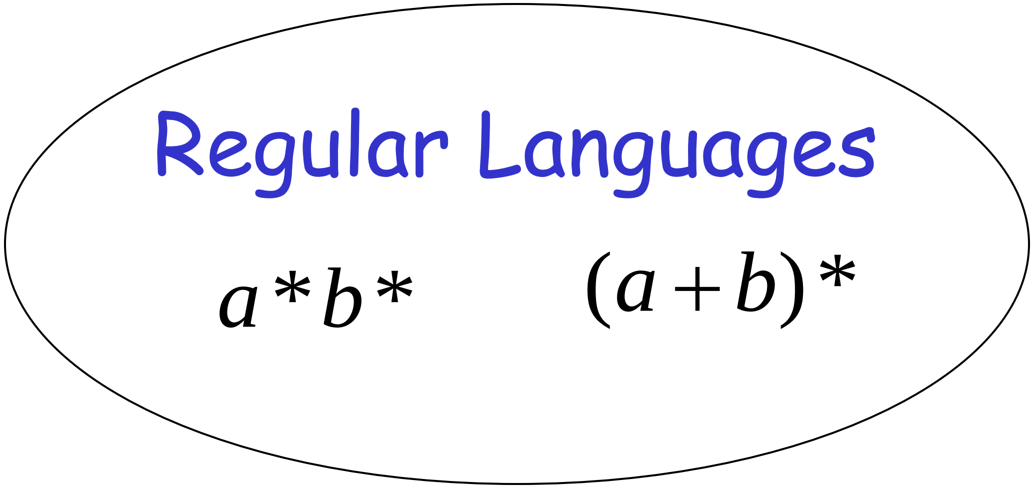
CONTRADICTION!!!

Therefore: Our assumption that L is a regular language is not true

Conclusion: L is not a regular language

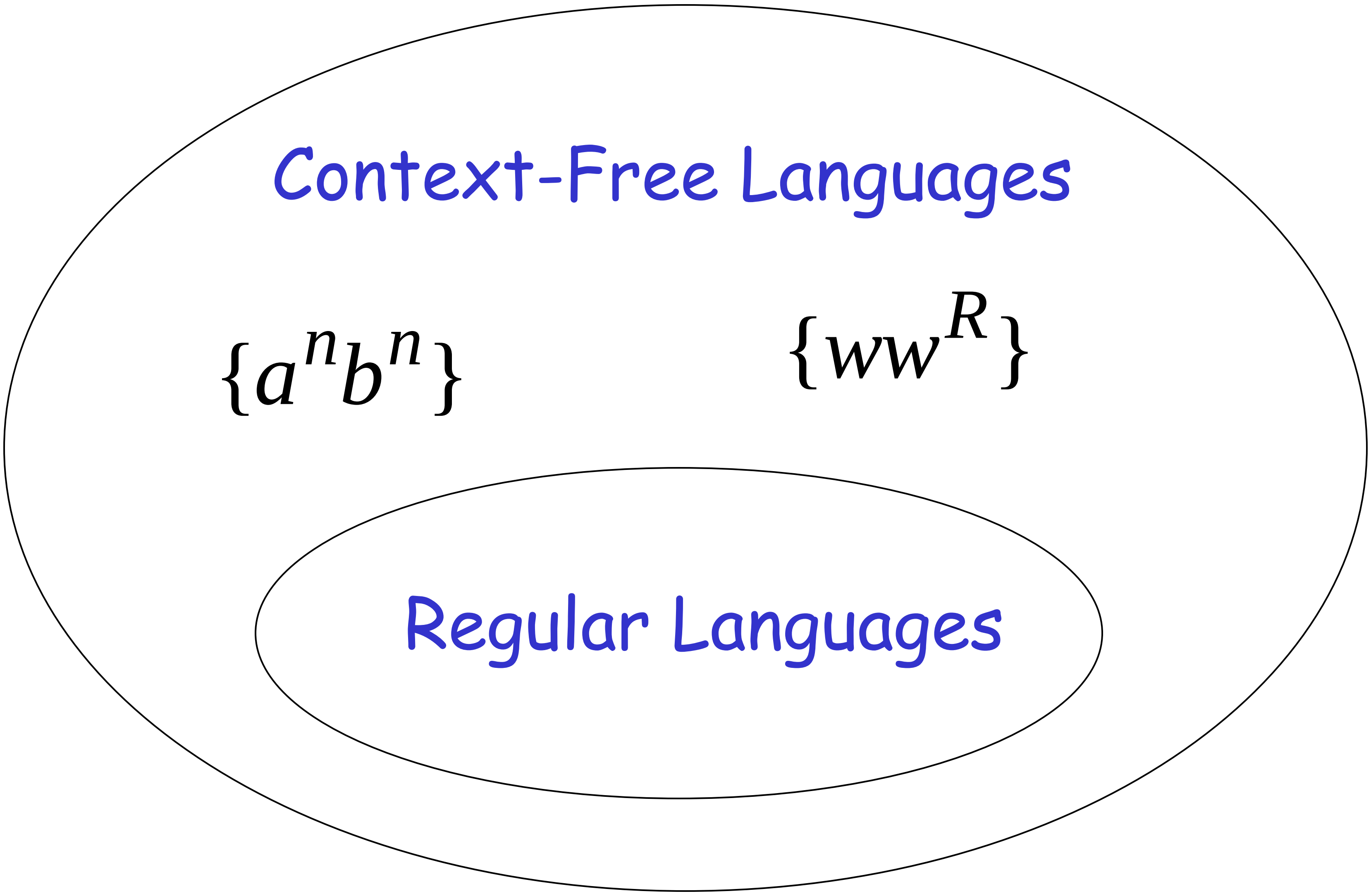
Context-Free Languages

$\{a^n b^n : n \geq 0\}$ $\{ww^R\}$



Regular Languages

a^*b^* $(a+b)^*$



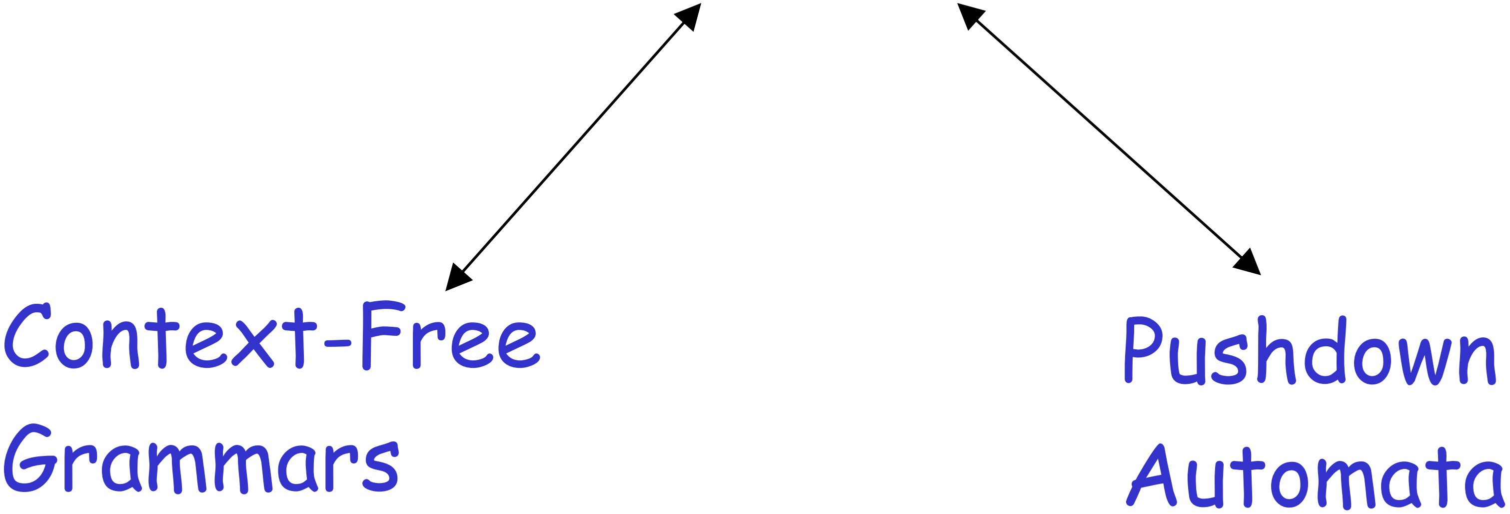
Context-Free Languages

$\{a^n b^n\}$

$\{ww^R\}$

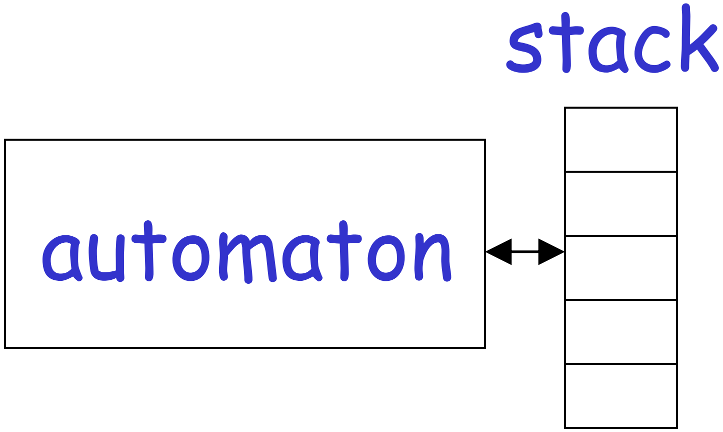
Regular Languages

Context-Free Languages



Context-Free Grammars

Pushdown Automata



stack

automaton

Context-Free Grammars

Example

A context-free grammar G:

$$S \to aSb$$

$$S \to \lambda$$

A derivation:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$$

A context-free grammar G:

$$S \to aSb$$

$$S \to \lambda$$

Another derivation:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb$$

$$S \to aSb$$

$$S \to \lambda$$

$$L(G) = \{a^n b^n : n \geq 0\}$$

Example

A context-free grammar G:

$$S \to aSa$$

$$S \to bSb$$

$$S \to \lambda$$

A derivation:

$$S \Rightarrow aSa \Rightarrow abSba \Rightarrow abba$$

A context-free grammar G:

$$S \to aSa$$

$$S \to bSb$$

$$S \to \lambda$$

Another derivation:

$$S \Rightarrow aSa \Rightarrow abSba \Rightarrow abaSaba \Rightarrow abaaba$$

$$S \to aSa$$

$$S \to bSb$$

$$S \to \lambda$$

$$L(G) = \{ww^R : \quad w \in \{a,b\}^*\}$$

Example

A context-free grammar G:

$$S \rightarrow aSb$$

$$S \rightarrow SS$$

$$S \rightarrow \lambda$$

A derivation:

$$S \Rightarrow SS \Rightarrow aSbS \Rightarrow abS \Rightarrow ab$$

A context-free grammar G:

$$S \to aSb$$

$$S \to SS$$

$$S \to \lambda$$

A derivation:

$$S \Rightarrow SS \Rightarrow aSbS \Rightarrow abS \Rightarrow abaSb \Rightarrow abab$$

$$S \to aSb$$

$$S \to SS$$

$$S \to \lambda$$

$$L(G) = \{w \quad : n_a(w) = n_b(w),$$

$$\text{and } \; n_a(v) \geq n_b(v)$$

$$\text{in any } \; \text{prefix } \; v\}$$

Definition: Context-Free Grammars

Grammar $G=(V,T,S,P)$

Variables

Terminal symbols

Start variable

Productions of the form:

$$A \to x$$

Variable

String of variables and terminals

$$G = (V, T, S, P)$$

$$L(G) = \{w : \quad S \overset{*}{\Rightarrow} w, \quad w \in T^*\}$$

Definition: Context-Free Languages

A language L is context-free

if and only if

there is a context-free grammar G
with $L = L(G)$

Derivation Order

1. $S \to AB$

2. $A \to aaA$

3. $A \to \lambda$

4. $B \to Bb$

5. $B \to \lambda$

Leftmost derivation:

$$S \overset{1}{\Rightarrow} AB \overset{2}{\Rightarrow} aaAB \overset{3}{\Rightarrow} aaB \overset{4}{\Rightarrow} aaBb \overset{5}{\Rightarrow} aab$$

Rightmost derivation:

$$S \overset{1}{\Rightarrow} AB \overset{4}{\Rightarrow} ABb \overset{5}{\Rightarrow} Ab \overset{2}{\Rightarrow} aaAb \overset{3}{\Rightarrow} aab$$

$$S \to aAB$$

$$A \to bBb$$

$$B \to A \mid \lambda$$

Leftmost derivation:

$$S \Rightarrow aAB \Rightarrow abBbB \Rightarrow abAbB \Rightarrow abbBbbB$$

$$\Rightarrow abbbbB \Rightarrow abbbb$$

Rightmost derivation:

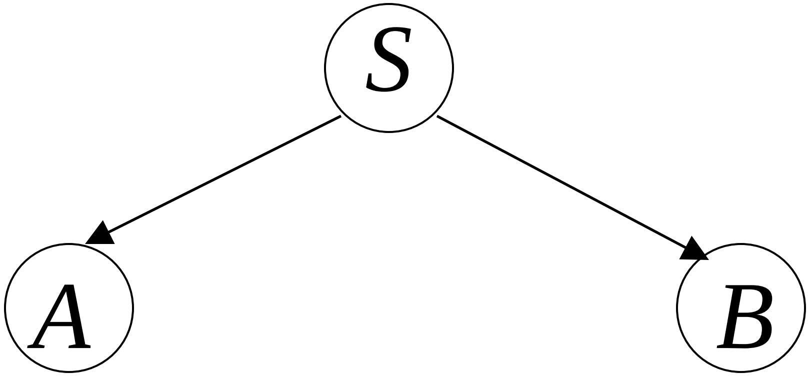
$$S \Rightarrow aAB \Rightarrow aA \Rightarrow abBb \Rightarrow abAb$$

$$\Rightarrow abbBbb \Rightarrow abbbb$$

Derivation Trees

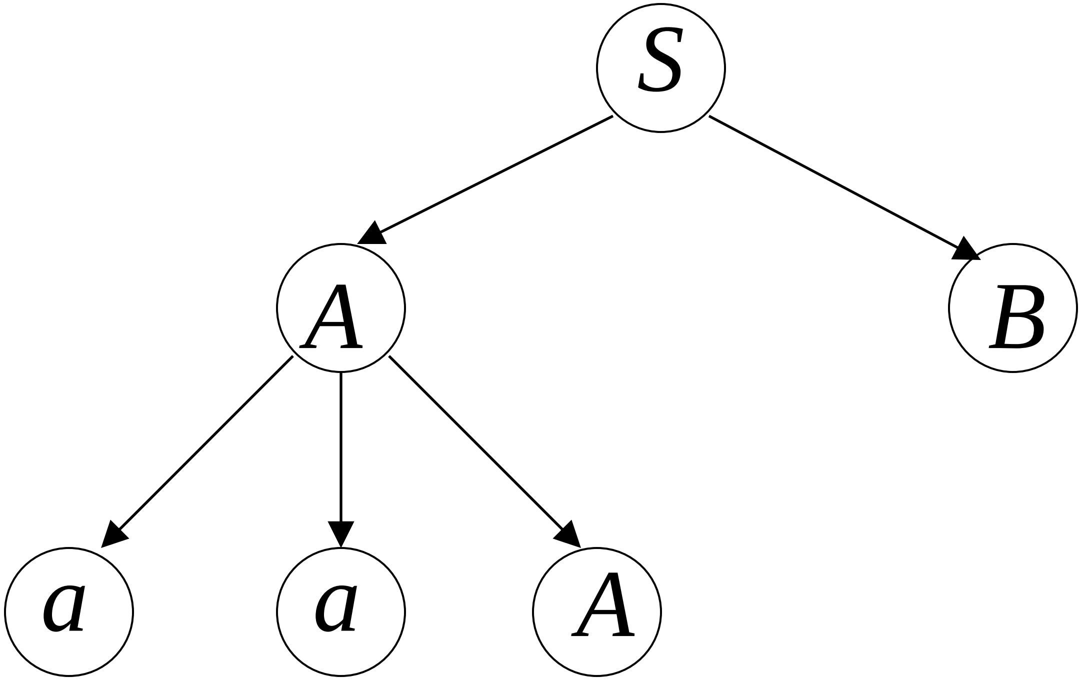
$S \to AB \qquad A \to aaA \mid \lambda \qquad B \to Bb \mid \lambda$

$$S \Rightarrow AB$$



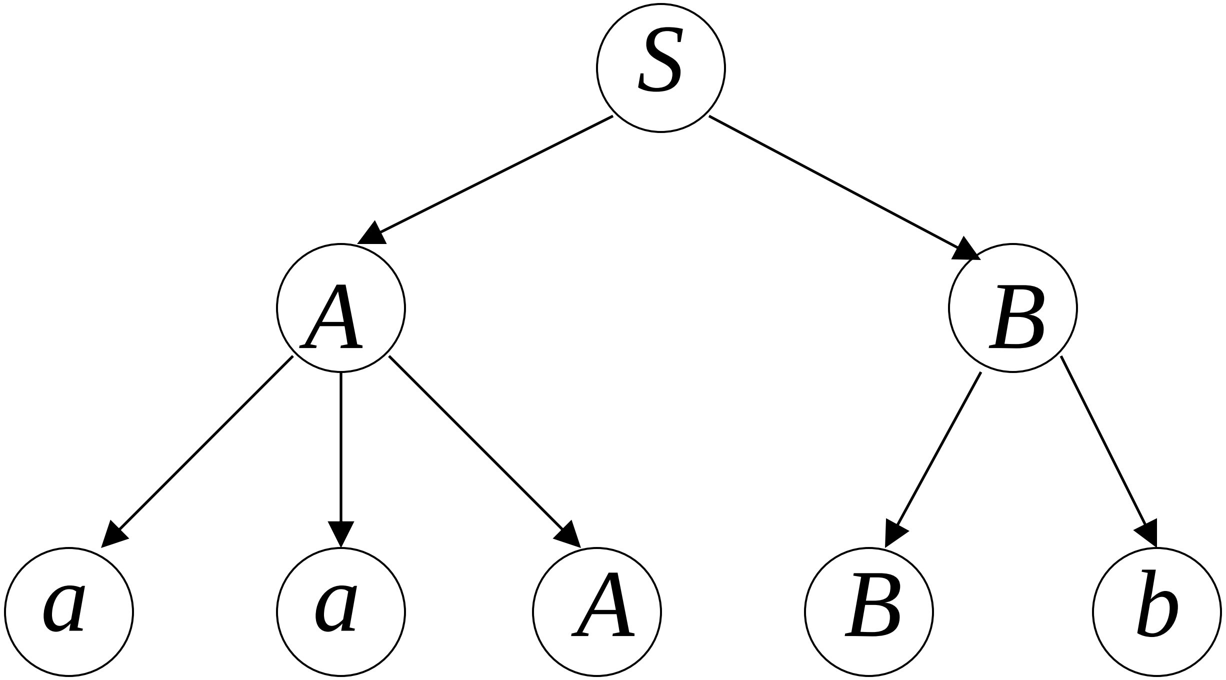
$S \to AB \qquad A \to aaA \mid \lambda \qquad B \to Bb \mid \lambda$

$$S \Rightarrow AB \Rightarrow aaAB$$



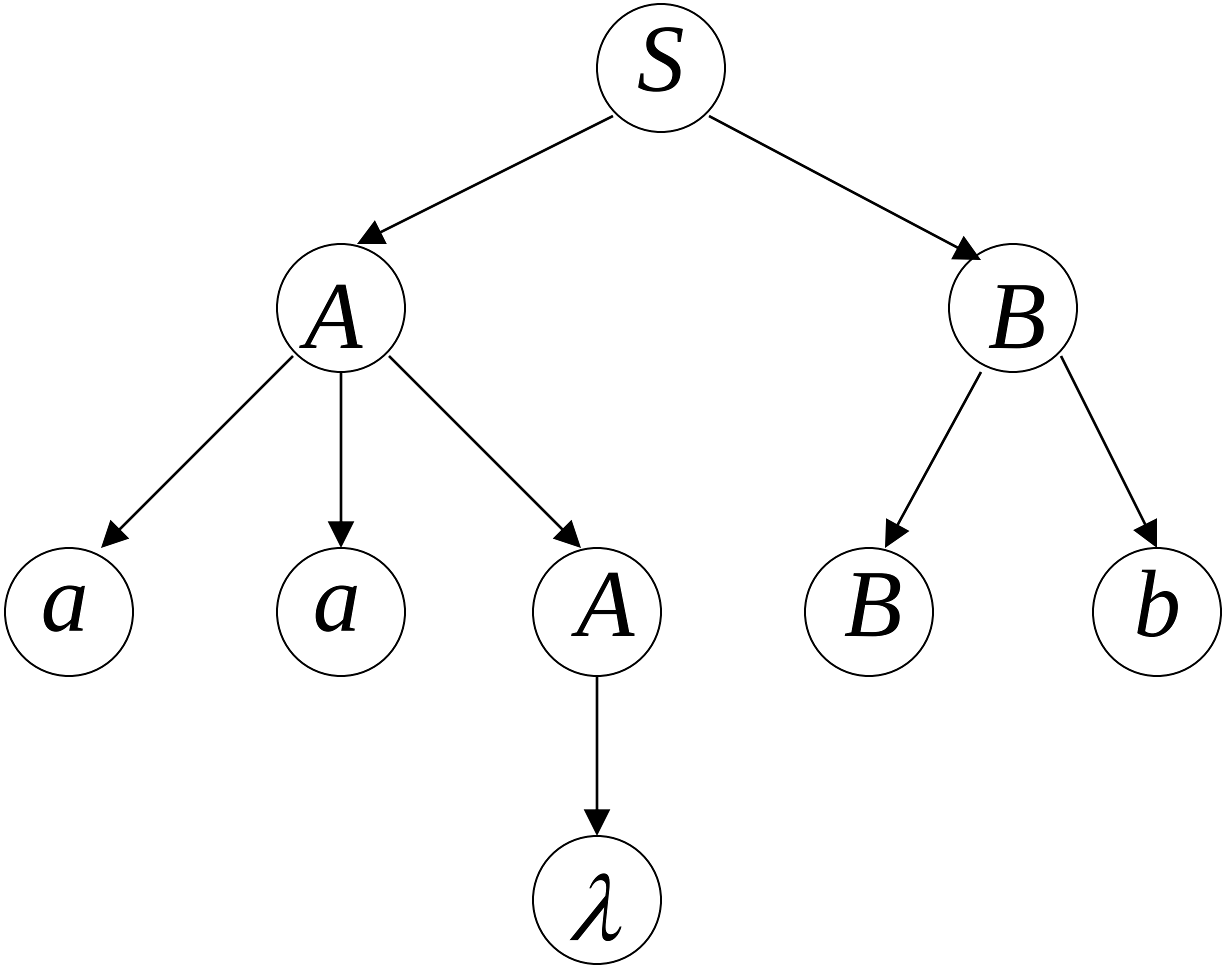
$$S \to AB \qquad A \to aaA \mid \lambda \qquad B \to Bb \mid \lambda$$

$$S \Rightarrow AB \Rightarrow aaAB \Rightarrow aaABb$$



$S \to AB \qquad A \to aaA \mid \lambda \qquad B \to Bb \mid \lambda$

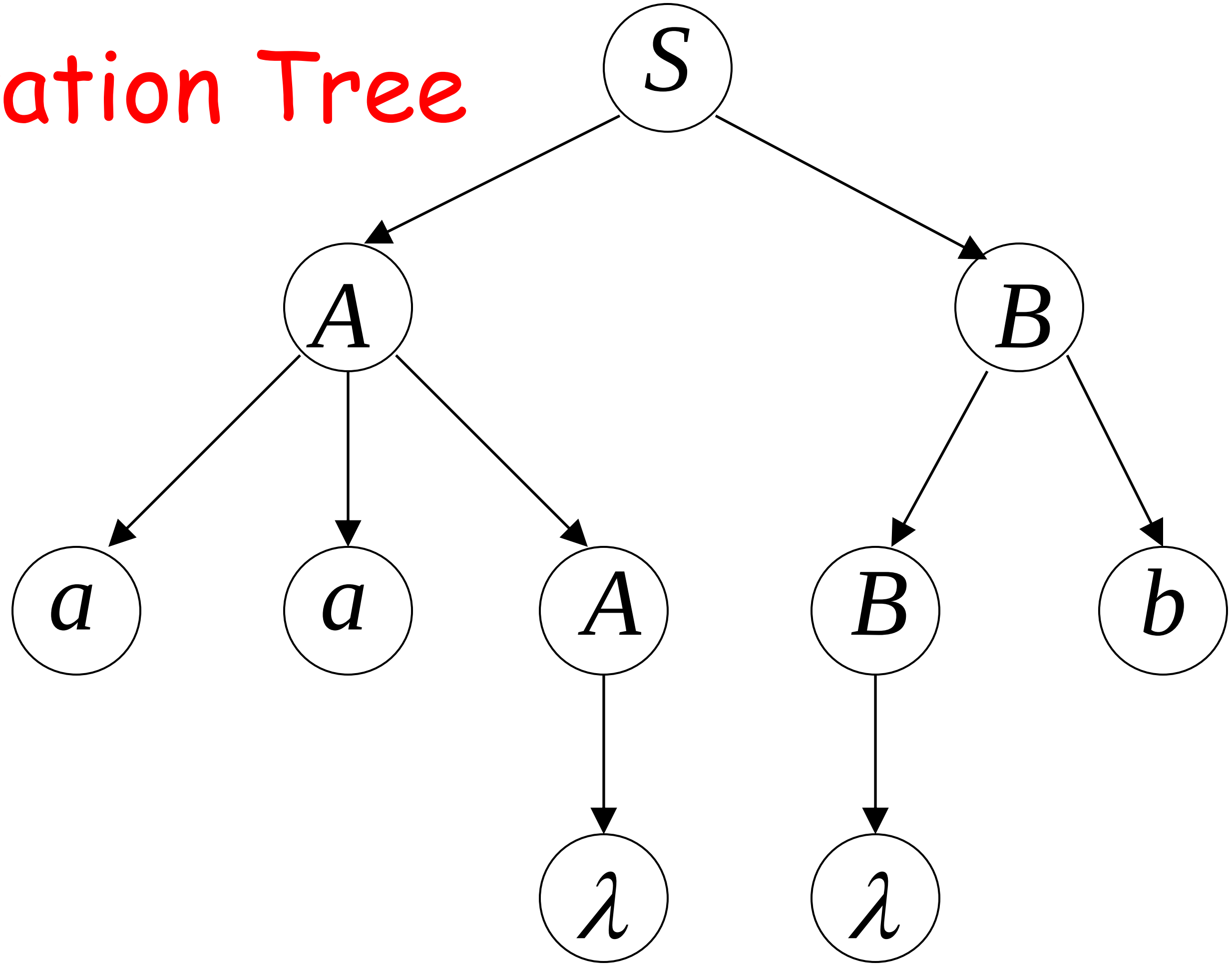
$$S \Rightarrow AB \Rightarrow aaAB \Rightarrow aaABb \Rightarrow aaBb$$



$$S \to AB \qquad A \to aaA \mid \lambda \qquad B \to Bb \mid \lambda$$

$$S \Rightarrow AB \Rightarrow aaAB \Rightarrow aaABb \Rightarrow aaBb \Rightarrow aab$$

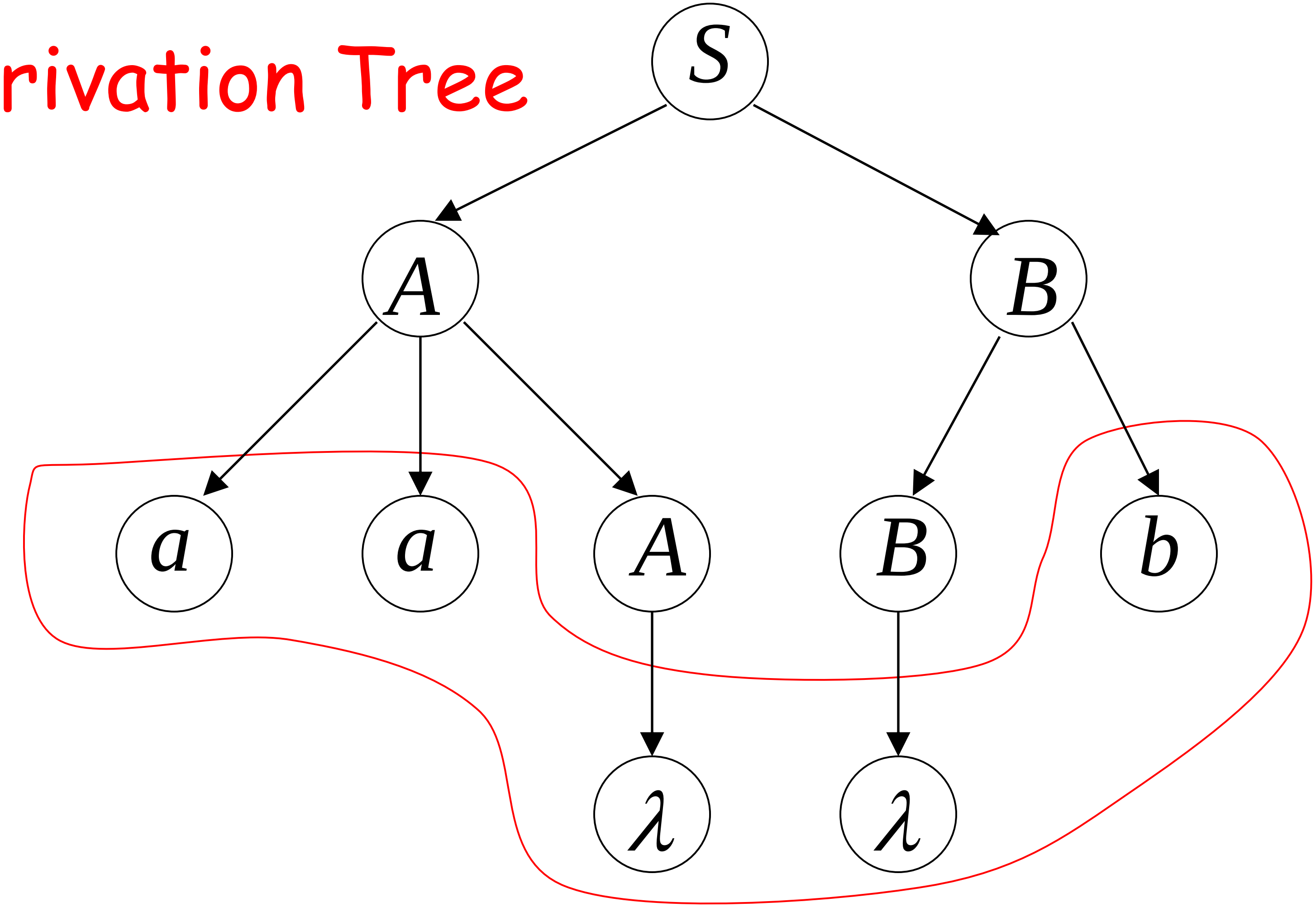
Derivation Tree



$$S \to AB \qquad A \to aaA \mid \lambda \qquad B \to Bb \mid \lambda$$

$$S \Rightarrow AB \Rightarrow aaAB \Rightarrow aaABb \Rightarrow aaBb \Rightarrow aab$$

Derivation Tree



yield

$$aa\lambda\lambda b$$

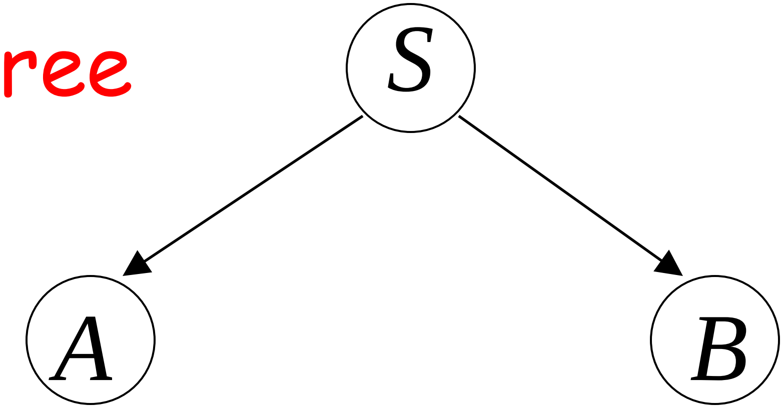
$$= aab$$

Partial Derivation Trees

$S \to AB \qquad A \to aaA \mid \lambda \qquad B \to Bb \mid \lambda$

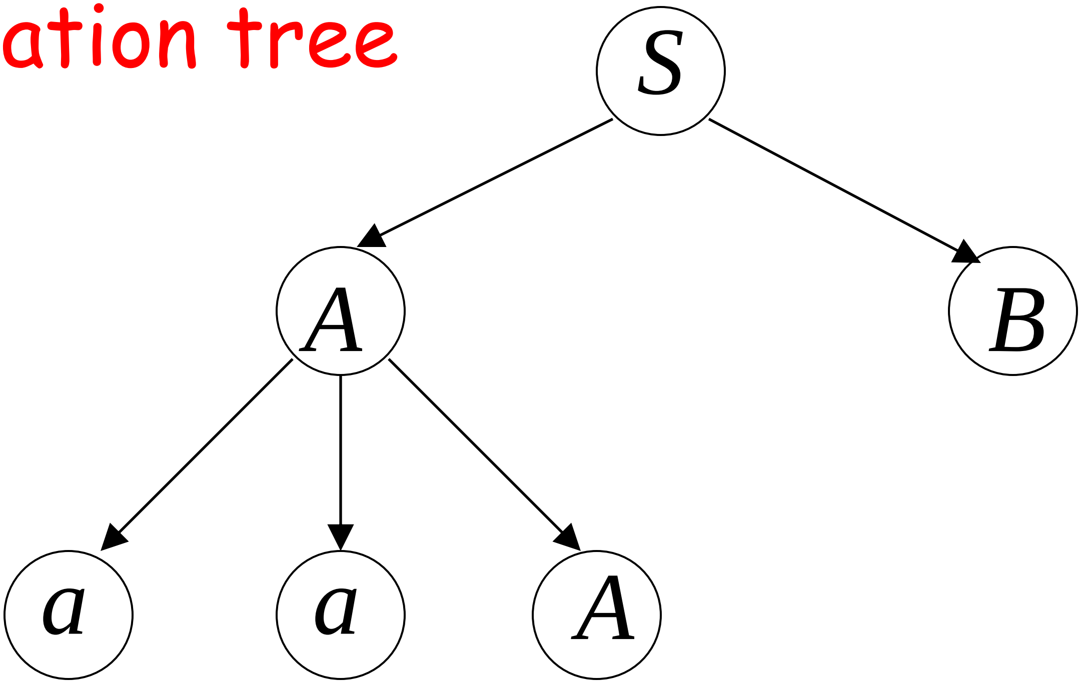
$$S \Rightarrow AB$$

Partial derivation tree



$$S \Rightarrow AB \Rightarrow aaAB$$

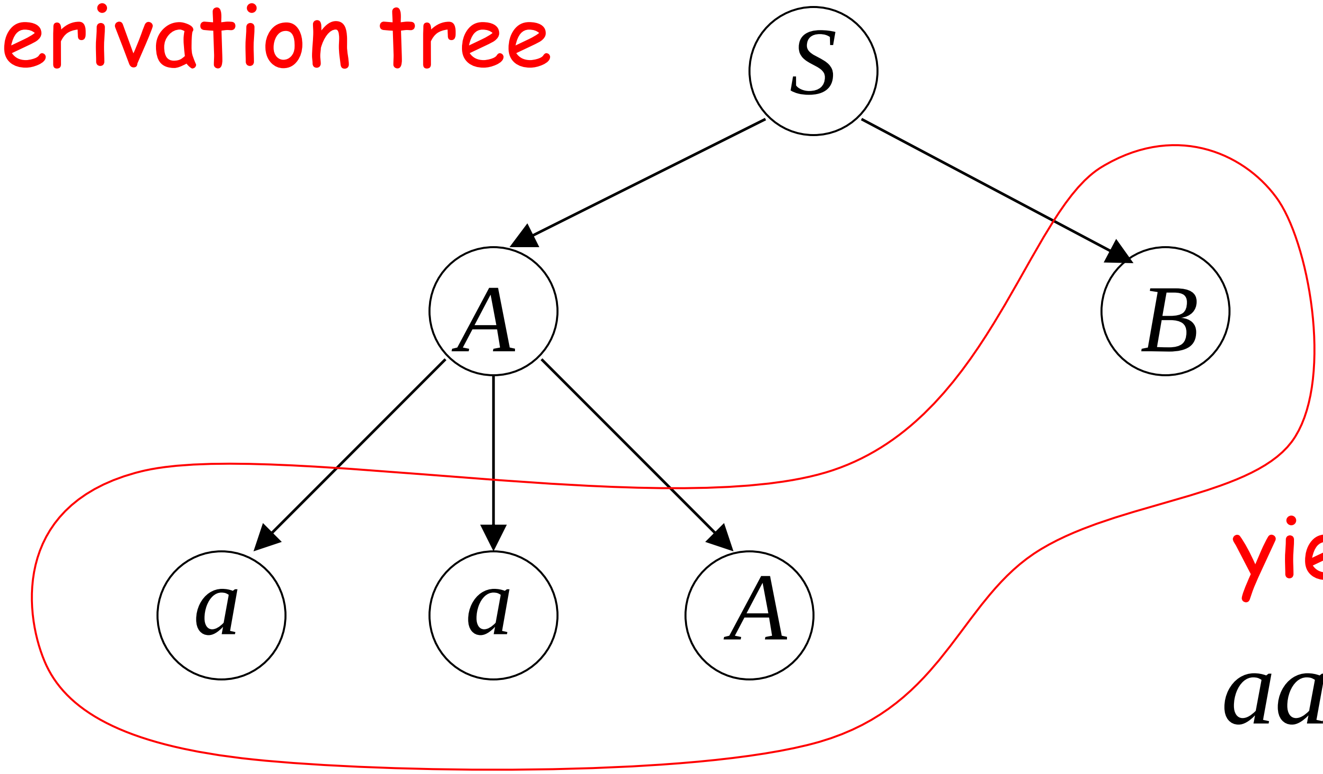
Partial derivation tree



$$S \Rightarrow AB \Rightarrow aaAB$$

sentential form

Partial derivation tree



yield

$aaAB$

Sometimes, derivation order doesn't matter

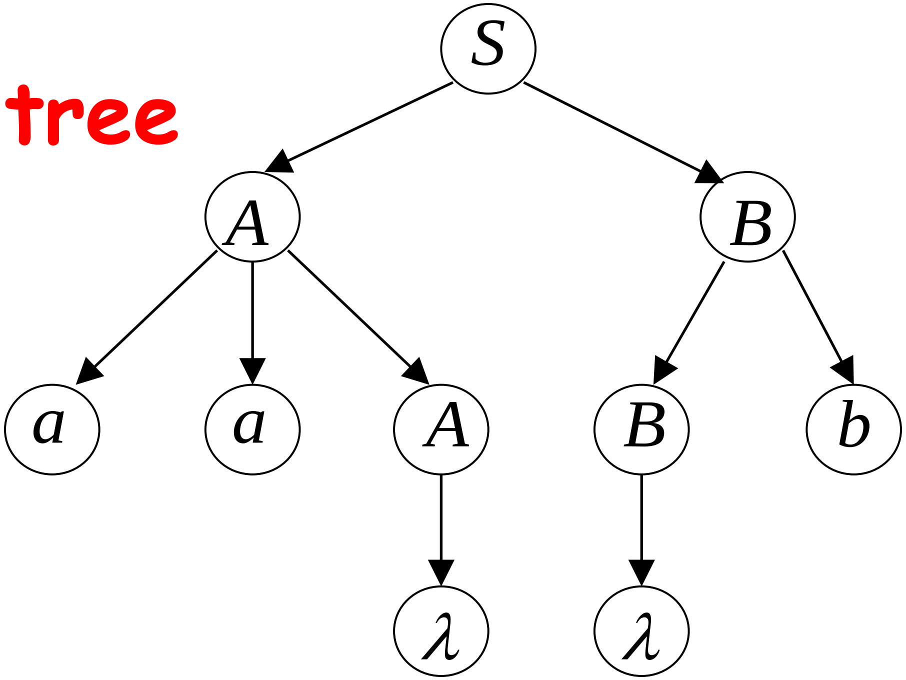
Leftmost:

$$S \Rightarrow AB \Rightarrow aaAB \Rightarrow aaB \Rightarrow aaBb \Rightarrow aab$$

Rightmost:

$$S \Rightarrow AB \Rightarrow ABb \Rightarrow Ab \Rightarrow aaAb \Rightarrow aab$$

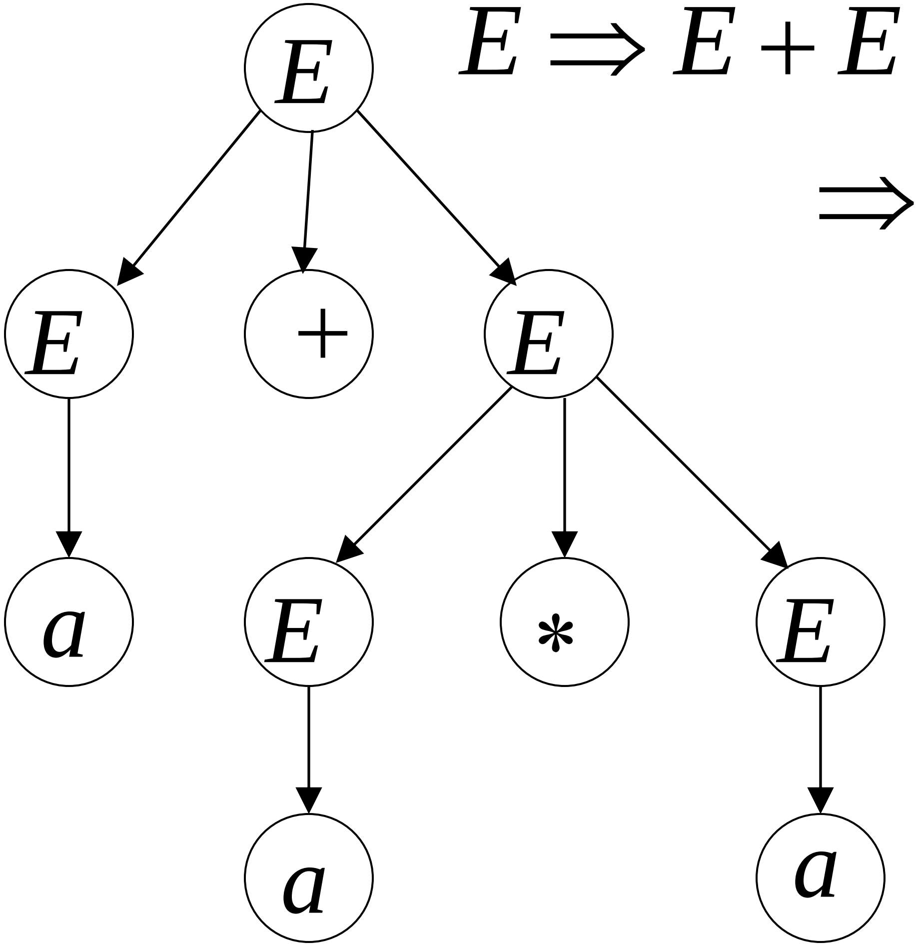
Same derivation tree



Ambiguity

$$E \rightarrow E+E \mid E*E \mid (E) \mid a$$

$$a+a*a$$



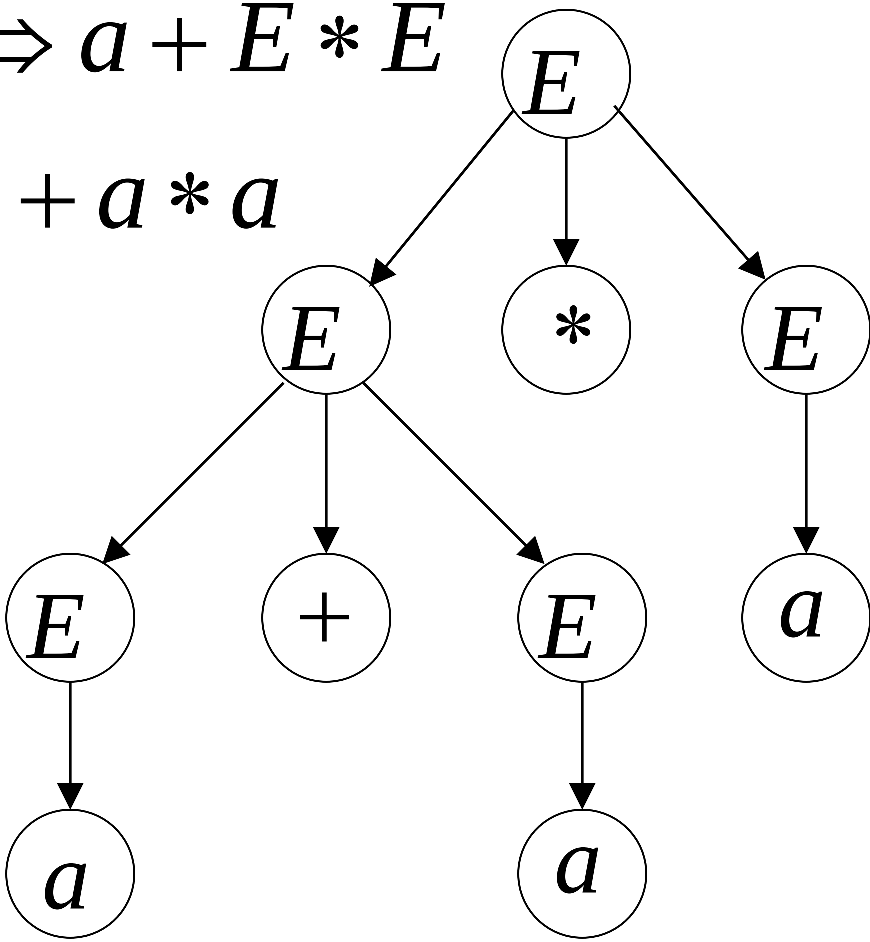
$$E \Rightarrow E+E \Rightarrow a+E \Rightarrow a+E*E$$
$$\Rightarrow a+a*E \Rightarrow a+a*a$$

leftmost derivation

$$E \rightarrow E + E \;\mid\; E * E \mid (E) \mid a$$

$$a + a * a$$

$$E \Rightarrow E * E \Rightarrow E + E * E \Rightarrow a + E * E$$
$$\Rightarrow a + a * E \Rightarrow a + a * a$$

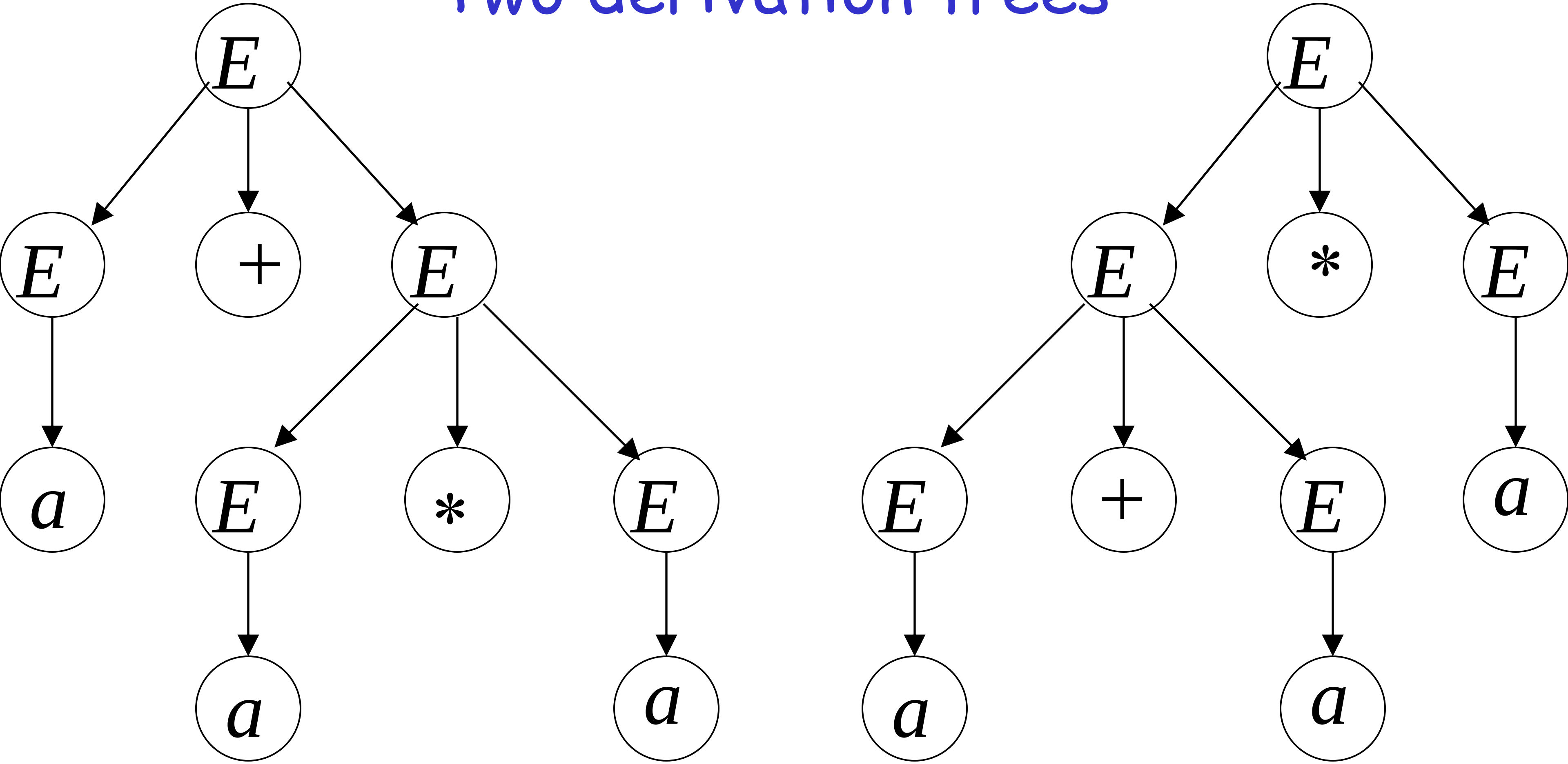


leftmost derivation

$$E \to E+E \;\mid\; E*E \;\mid\; (E) \;\mid\; a$$

$$a+a*a$$

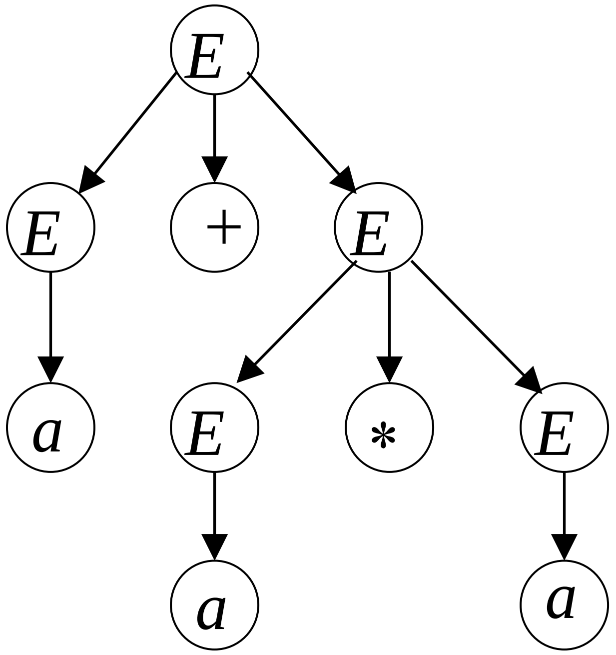
Two derivation trees

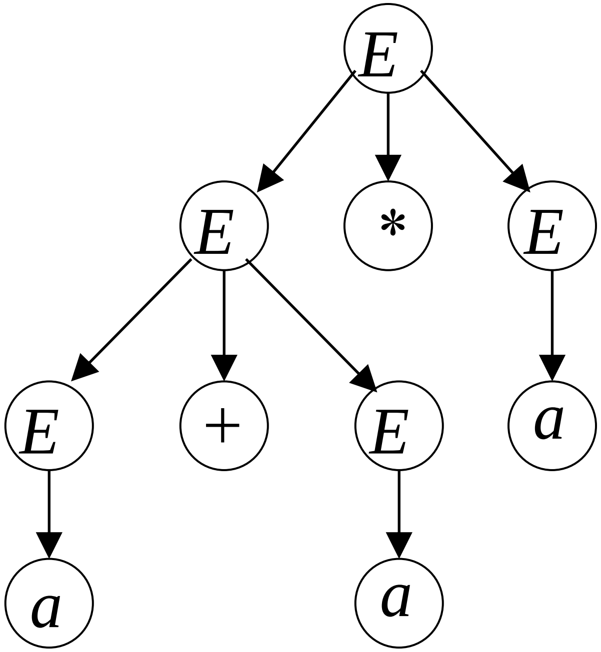


The grammar $E \to E + E \mid E * E \mid (E) \mid a$

is ambiguous:

string $a + a * a$ has two derivation trees





The grammar $E \to E + E \mid E * E \mid (E) \mid a$

is ambiguous:

string $a + a * a$ has two leftmost derivations

$$E \Rightarrow E + E \Rightarrow a + E \Rightarrow a + E * E$$
$$\Rightarrow a + a * E \Rightarrow a + a * a$$

$$E \Rightarrow E * E \Rightarrow E + E * E \Rightarrow a + E * E$$
$$\Rightarrow a + a * E \Rightarrow a + a * a$$

Definition:

A context-free grammar G is **ambiguous**

if some string $w \in L(G)$ has:

two or more derivation trees

In other words:

A context-free grammar G is **ambiguous**

if some string $w \in L(G)$ has:

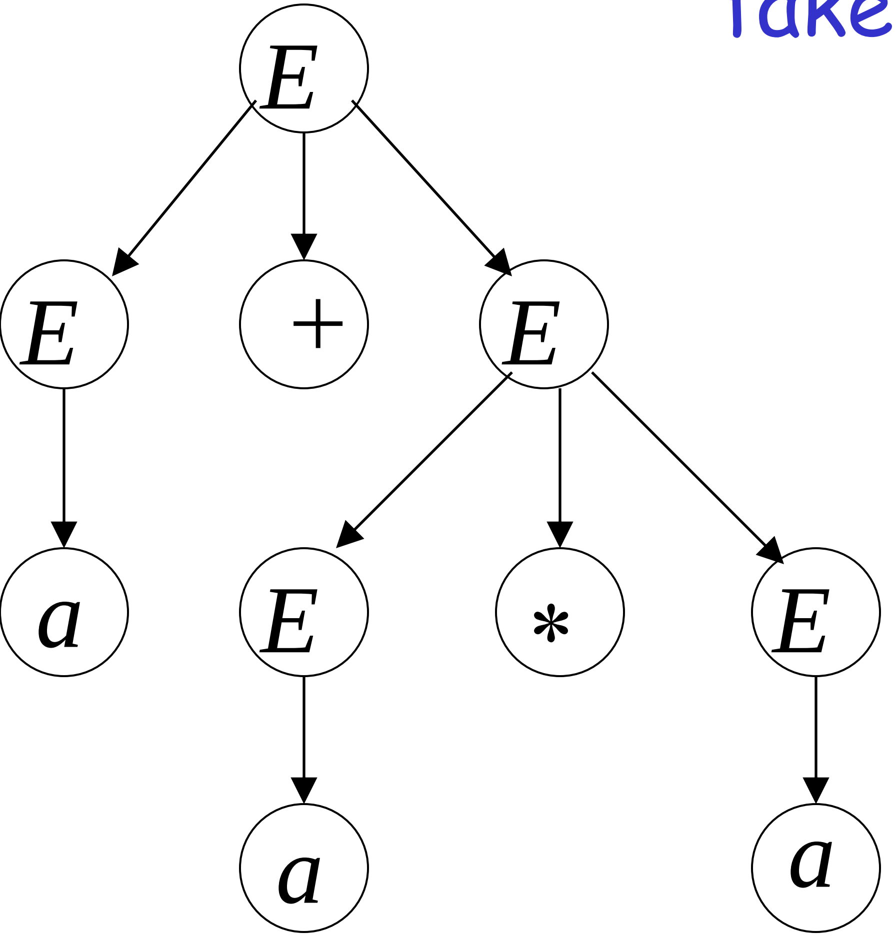
two or more leftmost derivations

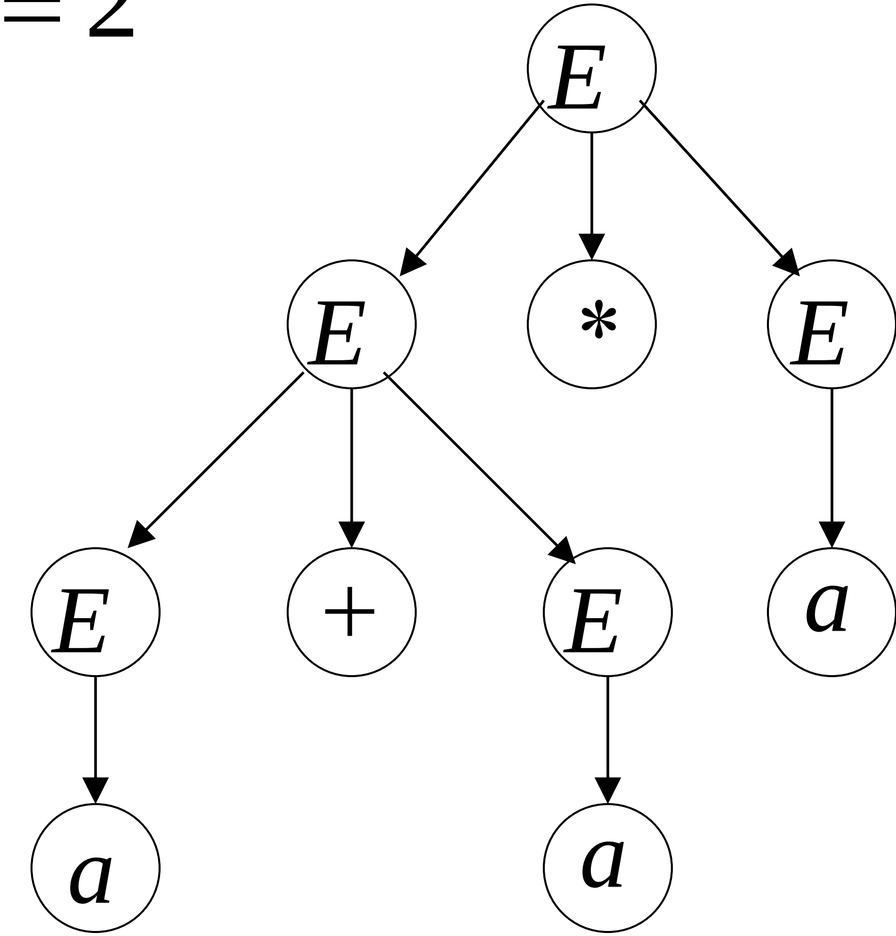
(or rightmost)

Why do we care about ambiguity?

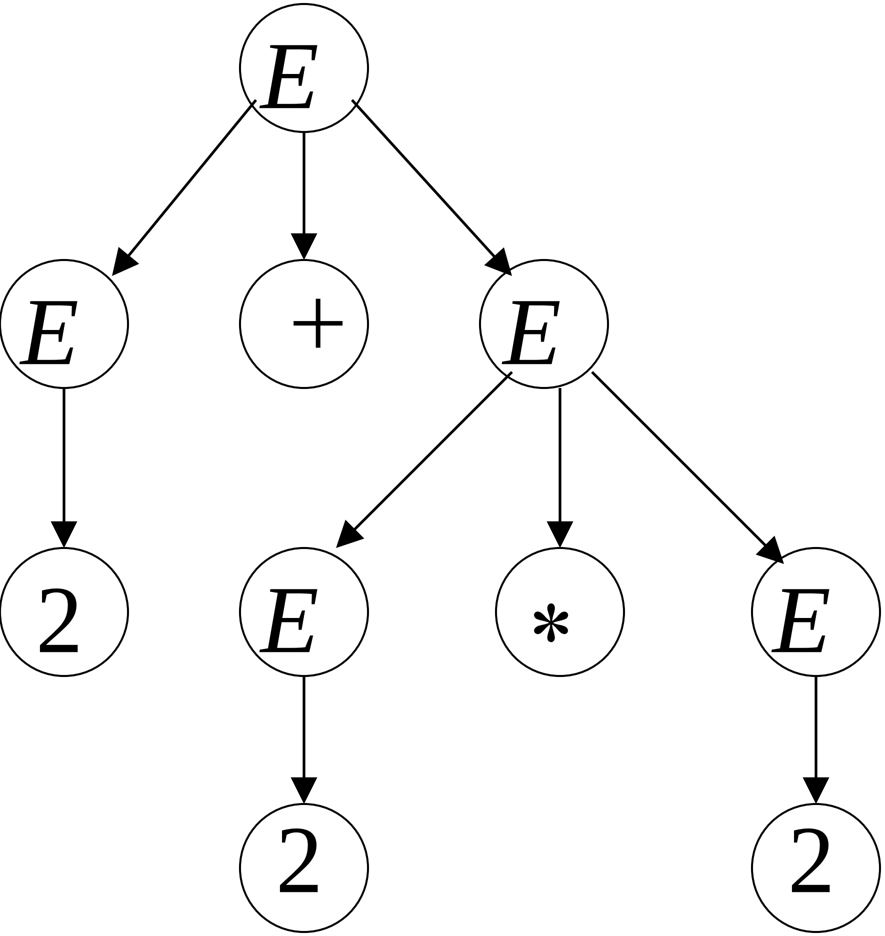
$$a + a * a$$

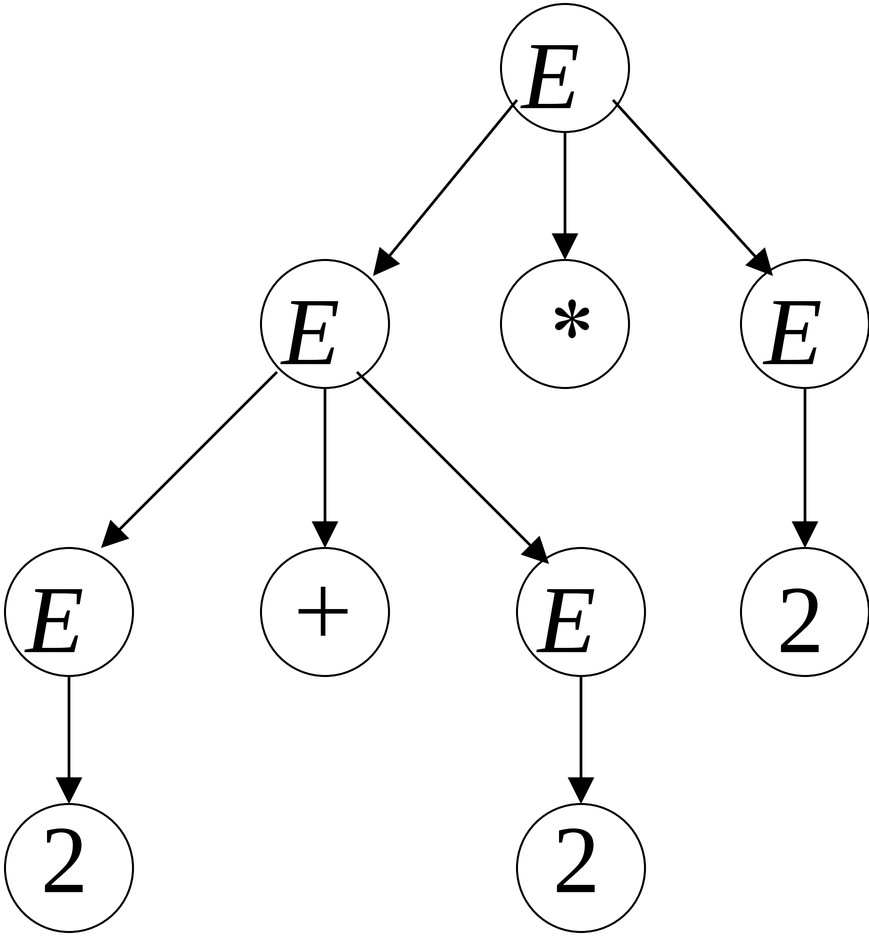
take $a = 2$



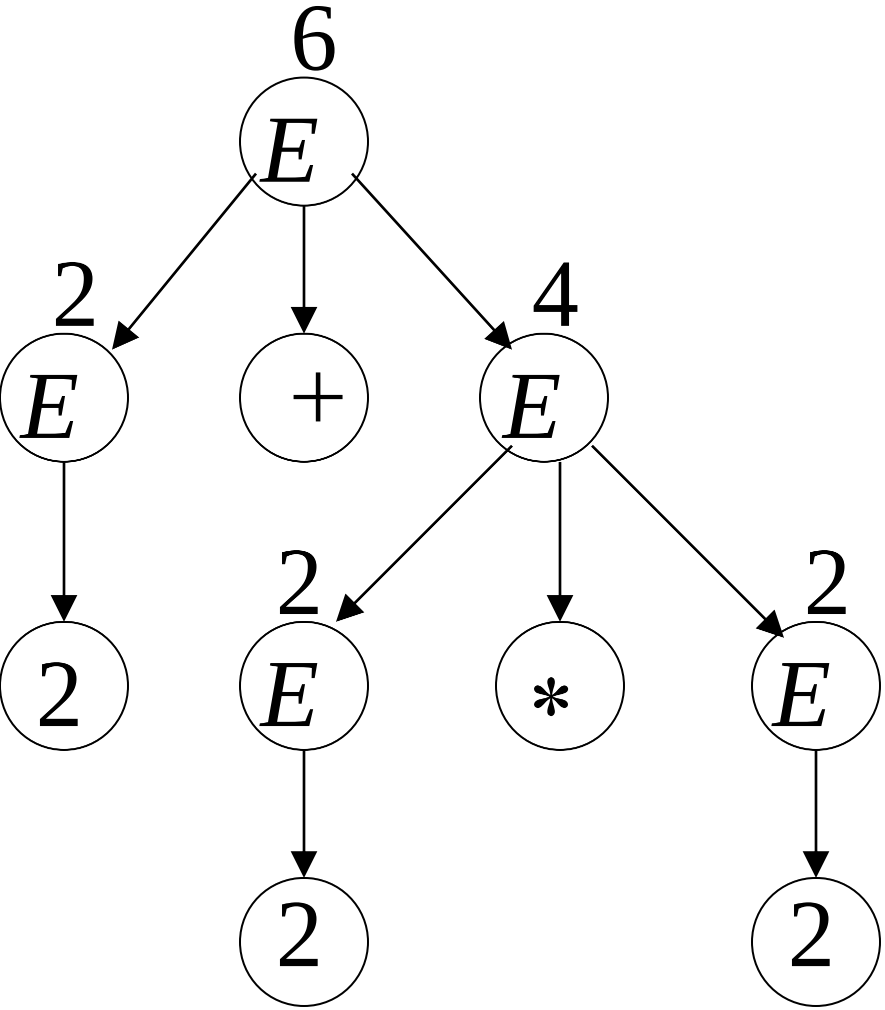


$$2 + 2 * 2$$

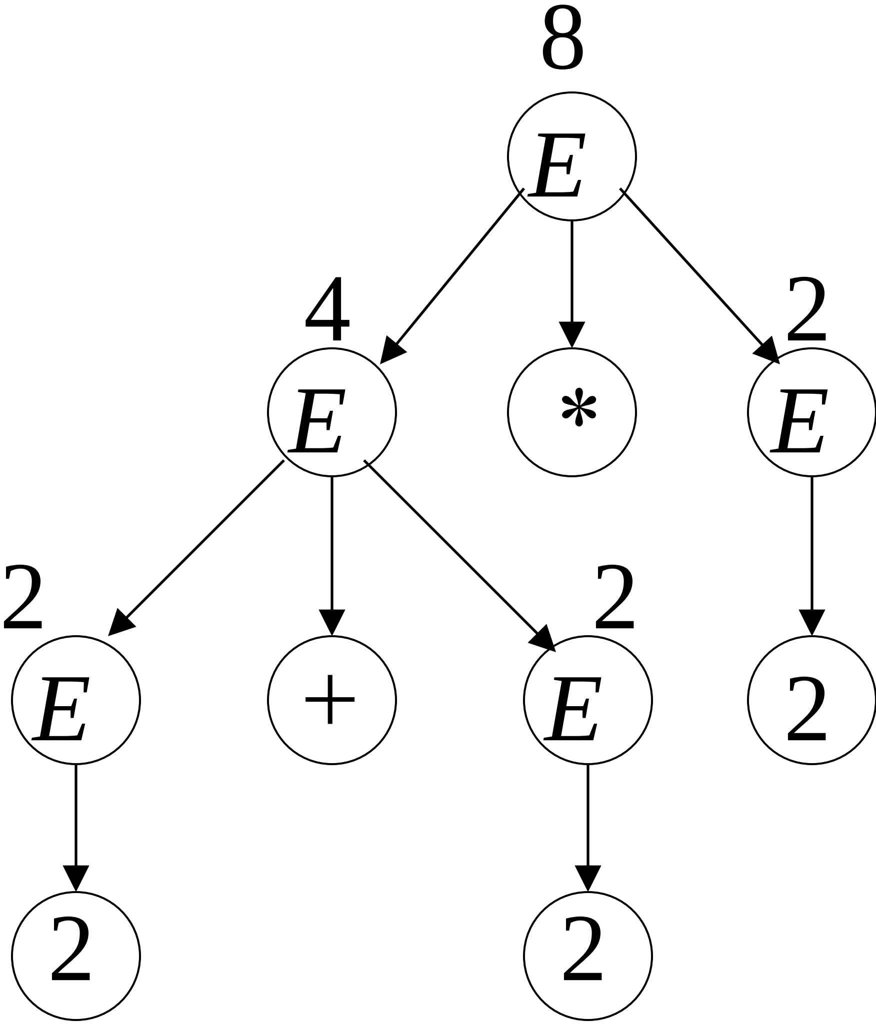




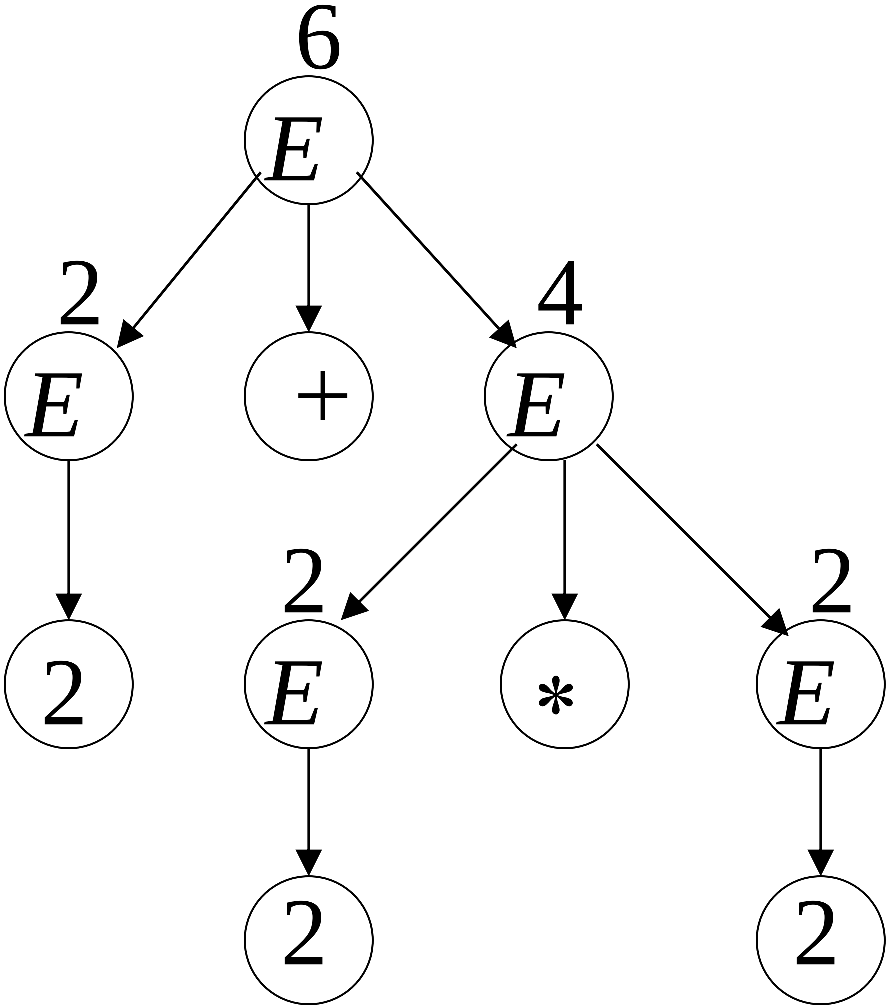
$$2 + 2 * 2 = 6$$



$$2 + 2 * 2 = 8$$



Correct result: $2+2*2=6$



- Ambiguity is **bad** for programming languages

- We want to remove ambiguity

We fix the ambiguous grammar:

$$E \to E + E \quad | \; E * E \; | \; (E) \; | \; a$$

New non-ambiguous grammar:

$$E \to E + T$$

$$E \to T$$

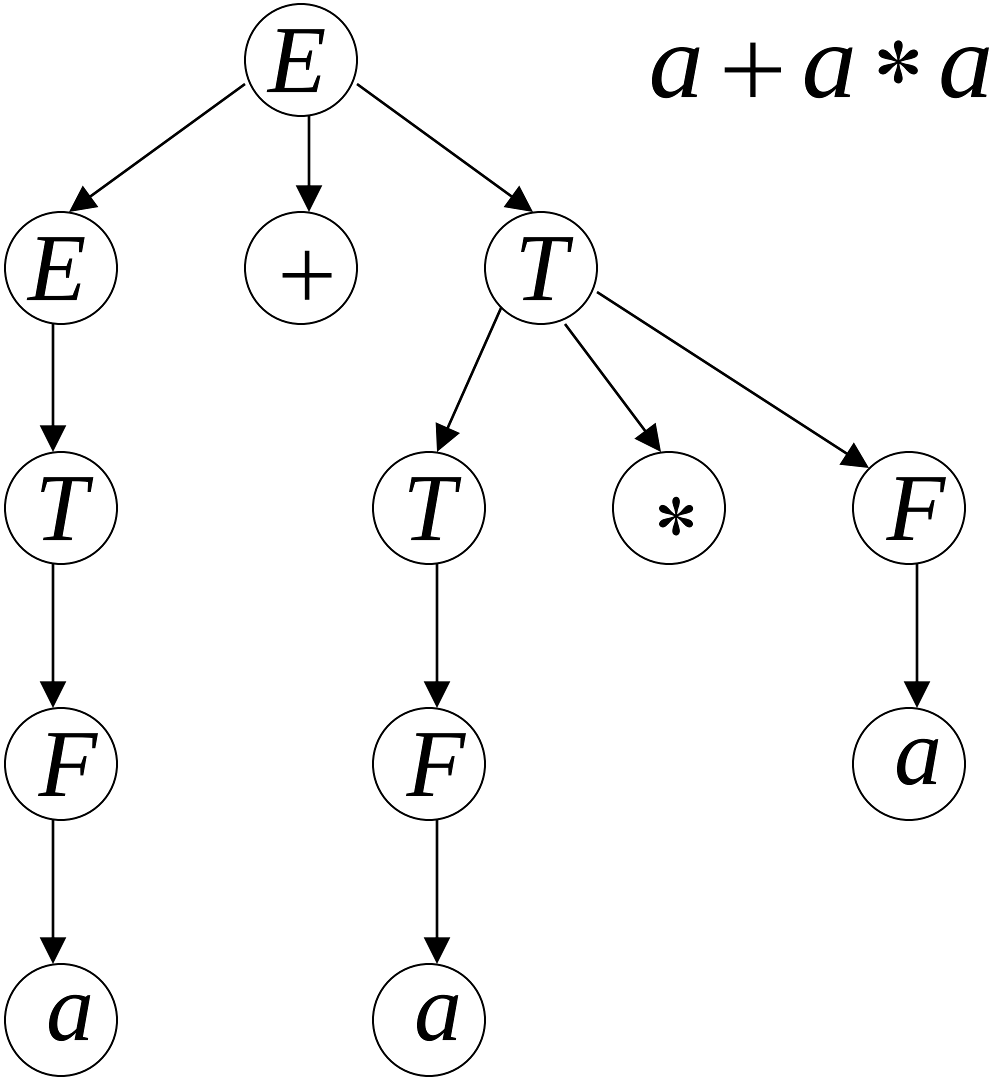
$$T \to T * F$$

$$T \to F$$

$$F \to (E)$$

$$F \to a$$

$$E \Rightarrow E + T \Rightarrow T + T \Rightarrow F + T \Rightarrow a + T \Rightarrow a + T * F$$
$$\Rightarrow a + F * F \Rightarrow a + a * F \Rightarrow a + a * a$$

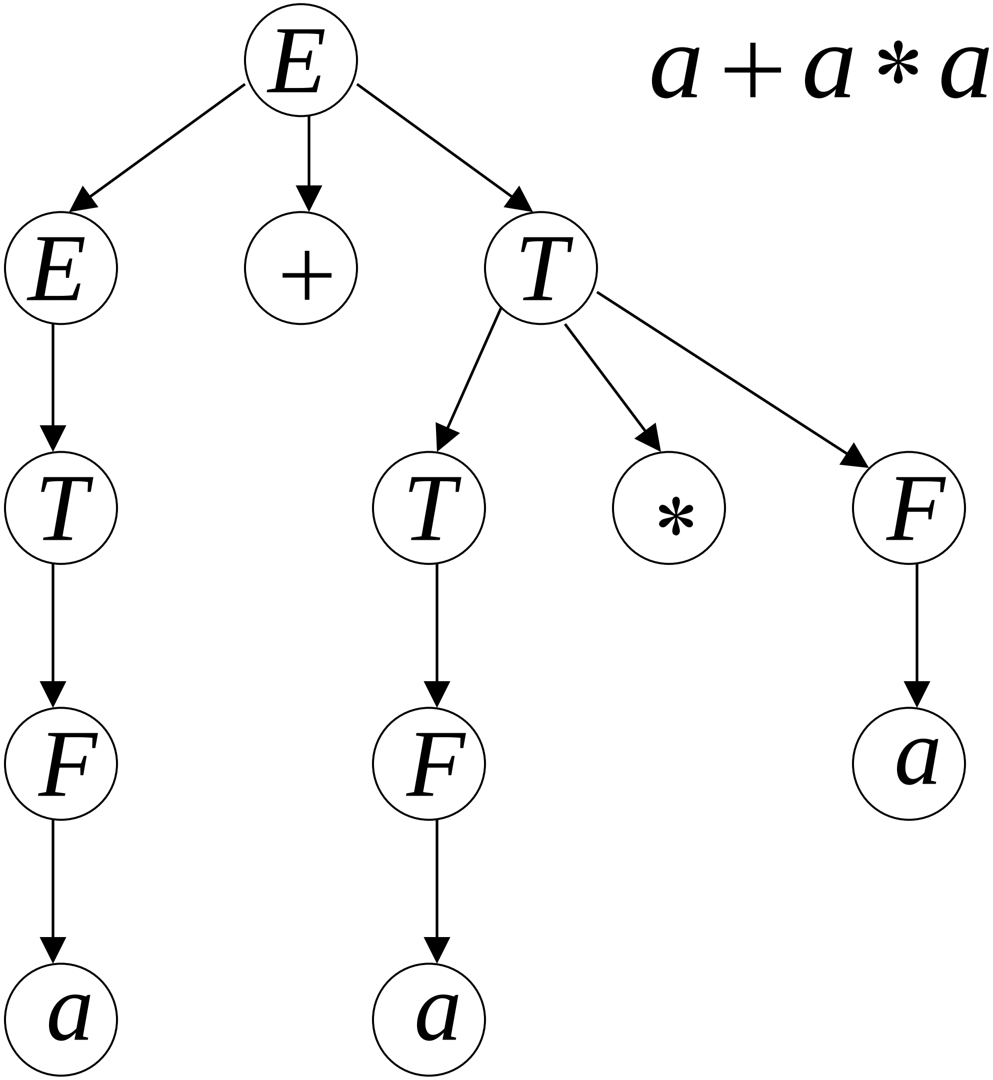


$a + a * a$

$$E \rightarrow E + T$$
$$E \rightarrow T$$
$$T \rightarrow T * F$$
$$T \rightarrow F$$
$$F \rightarrow (E)$$
$$F \rightarrow a$$

Unique derivation tree

$a + a * a$



The grammar G:

$$E \to E + T$$

$$E \to T$$

$$T \to T * F$$

$$T \to F$$

$$F \to (E)$$

$$F \to a$$

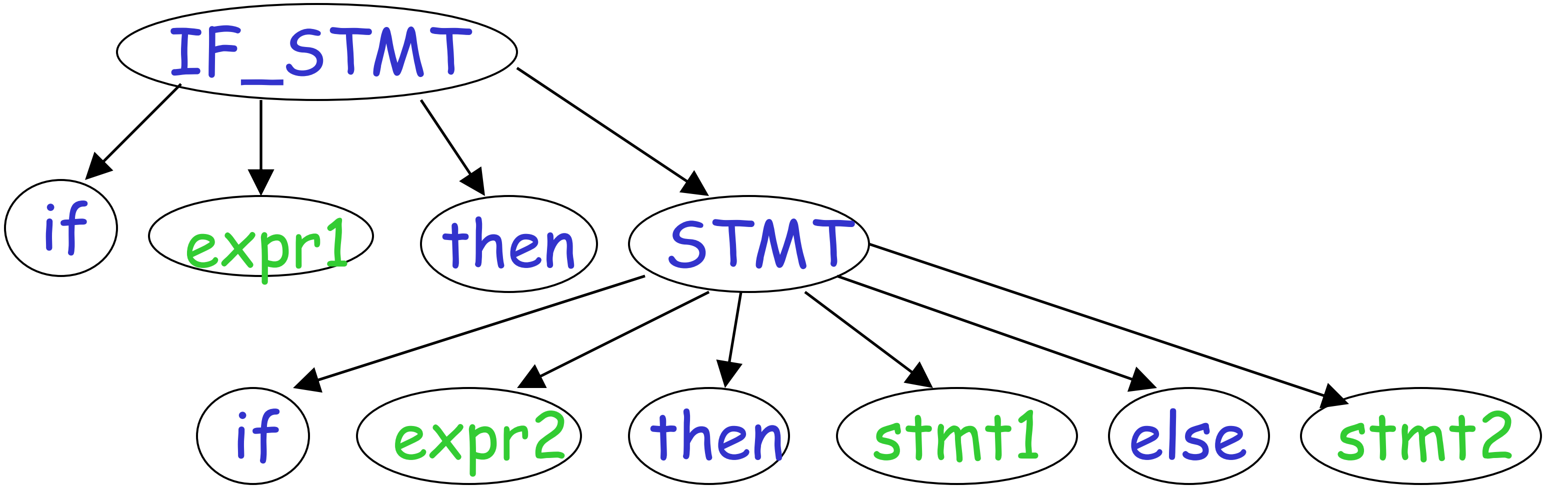
is non-ambiguous:

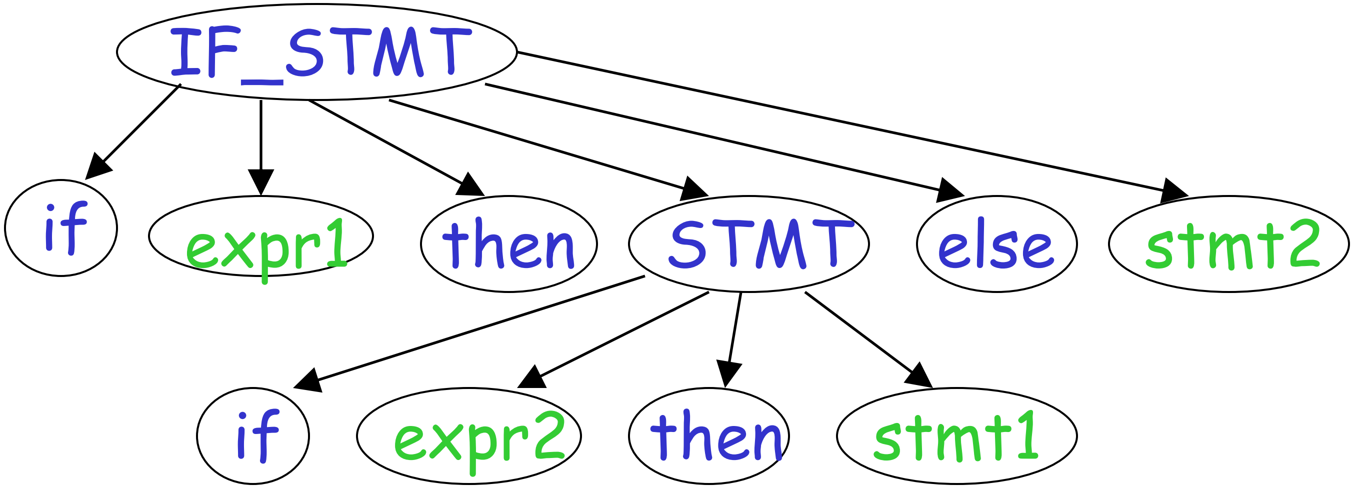
Every string $w \in L(G)$ has a unique derivation tree

Another Ambiguous Grammar

$$
\begin{aligned}
\text{IF\_STMT} \;\rightarrow\;& \text{if EXPR then STMT} \\
\mid\;& \text{if EXPR then STMT else STMT}
\end{aligned}
$$

If expr1 then if expr2 then stmt1 else stmt2





Inherent Ambiguity

Some context free languages have only ambiguous grammars

Example: $L = \{a^n b^n c^m\} \cup \{a^n b^m c^m\}$

$S \to S_1 \mid S_2$

$S_1 \to S_1 c \mid A$

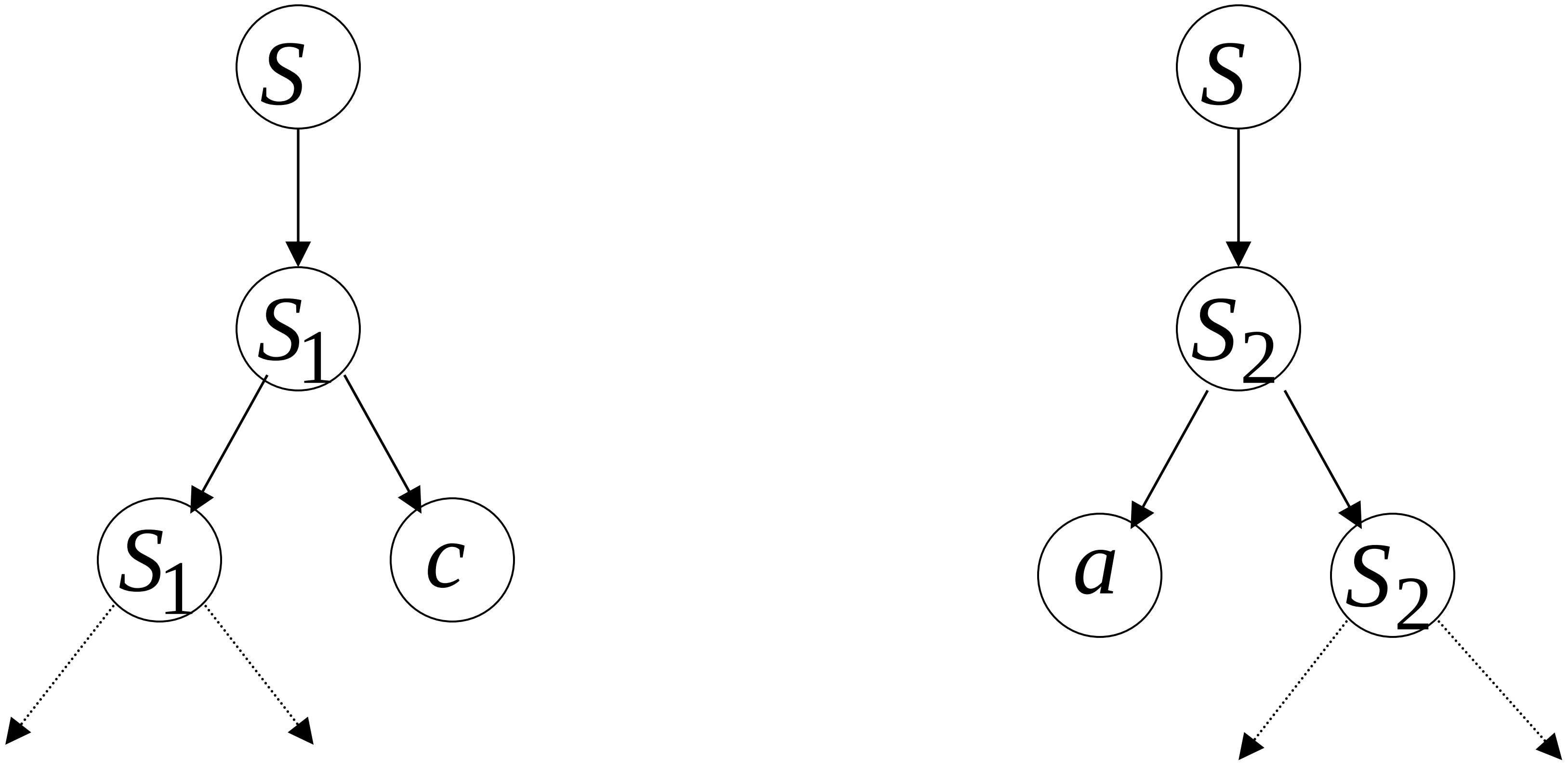
$A \to aAb \mid \lambda$

$S_2 \to aS_2 \mid B$

$B \to bBc \mid \lambda$

The string $a^n b^n c^n$

has two derivation trees



Simplifications
of
Context-Free Grammars

A Substitution Rule

$$S \to aB$$

$$A \to aaA$$

$$A \to abBc$$

$$B \to aA$$

$$B \to b$$

Substitute

$$B \to b$$

Equivalent grammar

$$S \to aB \mid ab$$

$$A \to aaA$$

$$A \to abBc \mid abbc$$

$$B \to aA$$

A Substitution Rule

$$S \to aB \mid ab$$

$$A \to aaA$$

$$A \to abBc \mid abbc$$

$$B \to aA$$

Substitute

$$B \to aA$$

$$S \to \cancel{aB} \mid ab \mid aaA$$

$$A \to aaA$$

$$A \to \cancel{abBc} \mid abbc \mid abaAc$$

Equivalent grammar

In general:

$$A \rightarrow xBz$$

$$B \rightarrow y_1$$

Substitute $B \rightarrow y_1$

$$A \rightarrow xBz \mid xy_1z$$

equivalent grammar

Nullable Variables

$\lambda - \text{production}:$ $A \rightarrow \lambda$

Nullable Variable: $A \Rightarrow \ldots \Rightarrow \lambda$

Removing Nullable Variables

Example Grammar:

$$S \to aMb$$

$$M \to aMb$$

$$M \to \lambda$$

Nullable variable

$$S \to aMb$$

$$M \to aMb$$

$$\xcancel{M \to \lambda}$$

Substitute

$$M \to \lambda$$

Final Grammar

$$S \to aMb$$

$$S \to ab$$

$$M \to aMb$$

$$M \to ab$$

Unit-Productions

Unit Production: $A \to B$

(a single variable in both sides)

Removing Unit Productions

Observation:

$$A \to A$$

Is removed immediately

Example Grammar:

$$S \to aA$$

$$A \to a$$

$$A \to B$$

$$B \to A$$

$$B \to bb$$

$$S \to aA$$

$$A \to a$$

$$\cancel{A \to B}$$

$$B \to A$$

$$B \to bb$$

Substitute

$$A \to B$$

$$S \to aA \mid aB$$

$$A \to a$$

$$B \to A \mid B$$

$$B \to bb$$

$$S \to aA \mid aB$$

$$A \to a$$

$$B \to A \mid \cancel{B}$$

$$B \to bb$$

Remove

$$B \to B$$

$$S \to aA \mid aB$$

$$A \to a$$

$$B \to A$$

$$B \to bb$$

$$S \to aA \mid aB$$

$$A \to a$$

$$\xcancel{B \to A}$$

$$B \to bb$$

Substitute

$$B \to A$$

$$S \to aA \mid aB \mid aA$$

$$A \to a$$

$$B \to bb$$

Remove repeated productions

$$S \rightarrow aA \mid aB \mid \cancel{aA}$$

$$A \rightarrow a$$

$$B \rightarrow bb$$

Final grammar

$$S \rightarrow aA \mid aB$$

$$A \rightarrow a$$

$$B \rightarrow bb$$

Useless Productions

$$S \rightarrow aSb$$

$$S \rightarrow \lambda$$

$$S \rightarrow A$$

$$\boxed{A \rightarrow aA}$$ Useless Production

Some derivations never terminate...

$$S \Rightarrow A \Rightarrow aA \Rightarrow aaA \Rightarrow \ldots \Rightarrow aa\ldots aA \Rightarrow \ldots$$

Another grammar:

$$S \rightarrow A$$

$$A \rightarrow aA$$

$$A \rightarrow \lambda$$

$$\boxed{B \rightarrow bA}$$ Useless Production

Not reachable from S

In general:

contains only terminals

if $S \Rightarrow \ldots \Rightarrow xAy \Rightarrow \ldots \Rightarrow w$

$w \in L(G)$

then variable A is useful

otherwise, variable A is useless

A production $A \to x$ is useless
if any of its variables is useless

$$S \to aSb$$

$$S \to \lambda$$

Productions

Variables $S \to A$ useless

useless $A \to aA$ useless

useless $B \to C$ useless

useless $C \to D$ useless

Removing Useless Productions

Example Grammar:

$$S \to aS \mid A \mid C$$

$$A \to a$$

$$B \to aa$$

$$C \to aCb$$

First: find all variables that can produce strings with only terminals

$$S \to aS \mid A \mid C$$

$$\boxed{A \to a}$$

$$\boxed{B \to aa}$$

$$C \to aCb$$

Round 1: $\{A, B\}$

$$S \to A$$

Round 2: $\{A, B, S\}$

Keep only the variables that produce terminal symbols: $\{A, B, S\}$

(the rest variables are useless)

$$S \to aS \mid A \mid \cancel{C}$$

$$A \to a$$

$$B \to aa$$

$$\cancel{C \to aCb}$$

➡

$$S \to aS \mid A$$

$$A \to a$$

$$B \to aa$$

Remove useless productions

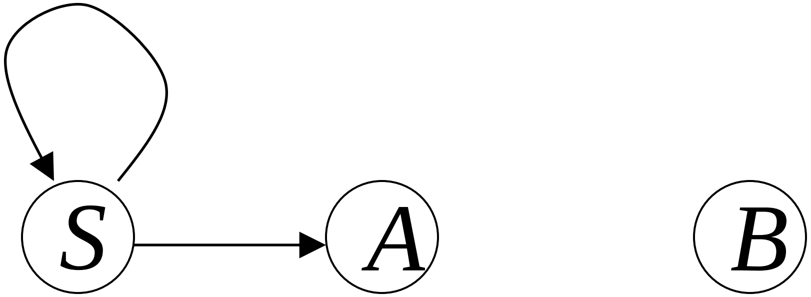
Second: Find all variables reachable from S

Use a Dependency Graph

$$S \to aS \mid A$$

$$A \to a$$

$$B \to aa$$



not reachable

Keep only the variables reachable from S

(the rest variables are useless)

$$S \to aS \mid A$$

$$A \to a$$

$$\xcancel{B \to aa}$$

Final Grammar

$$S \to aS \mid A$$

$$A \to a$$

Remove useless productions

Removing All

Step 1: Remove Nullable Variables

Step 2: Remove Unit-Productions

Step 3: Remove Useless Variables

Normal Forms
for
Context-free Grammars

Chomsky Normal Form

Each productions has form:

$A \to BC$ or $A \to a$

variable variable terminal

Examples:

$$S \to AS$$

$$S \to a$$

$$A \to SA$$

$$A \to b$$

Chomsky
Normal Form

$$S \to AS$$

$$S \to \boxed{AAS}$$

$$A \to SA$$

$$A \to \boxed{aa}$$

Not Chomsky
Normal Form

Convertion to Chomsky Normal Form

Example:

$$S \to ABa$$

$$A \to aab$$

$$B \to Ac$$

Not Chomsky
Normal Form

Introduce variables for terminals: T_a, T_b, T_c

$$S \to ABa$$

$$A \to aab$$

$$B \to Ac$$

⇒

$$S \to ABT_a$$

$$A \to T_aT_aT_b$$

$$B \to AT_c$$

$$T_a \to a$$

$$T_b \to b$$

$$T_c \to c$$

Introduce intermediate variable: V_1

$$S \to ABT_a$$
$$A \to T_aT_aT_b$$
$$B \to AT_c$$
$$T_a \to a$$
$$T_b \to b$$
$$T_c \to c$$

$$\Rightarrow$$

$$S \to AV_1$$
$$V_1 \to BT_a$$
$$A \to T_aT_aT_b$$
$$B \to AT_c$$
$$T_a \to a$$
$$T_b \to b$$
$$T_c \to c$$

Introduce intermediate variable: V_2

$$S \to AV_1$$
$$V_1 \to BT_a$$
$$A \to T_aT_aT_b$$
$$B \to AT_c$$
$$T_a \to a$$
$$T_b \to b$$
$$T_c \to c$$

➡

$$S \to AV_1$$
$$V_1 \to BT_a$$
$$A \to T_aV_2$$
$$V_2 \to T_aT_b$$
$$B \to AT_c$$
$$T_a \to a$$
$$T_b \to b$$
$$T_c \to c$$

Final grammar in Chomsky Normal Form:

$$S \to AV_1$$

$$V_1 \to BT_a$$

$$A \to T_aV_2$$

$$V_2 \to T_aT_b$$

$$B \to AT_c$$

$$T_a \to a$$

$$T_b \to b$$

$$T_c \to c$$

Initial grammar

$$S \to ABa$$

$$A \to aab$$

$$B \to Ac$$

In general:

From any context-free grammar
(which doesn't produce λ)
not in Chomsky Normal Form

we can obtain:

An equivalent grammar
in Chomsky Normal Form

The Procedure

First remove:

Nullable variables

Unit productions

Then, for every symbol a:

Add production $T_a \rightarrow a$

In productions: replace a with T_a

New variable: T_a

Replace any production $A \rightarrow C_1 C_2 \cdots C_n$

with
$$
\begin{aligned}
&A \rightarrow C_1 V_1 \\
&V_1 \rightarrow C_2 V_2 \\
&\cdots \\
&V_{n-2} \rightarrow C_{n-1} C_n
\end{aligned}
$$

New intermediate variables: $V_1, V_2, \ldots, V_{n-2}$

Theorem: For any context-free grammar (which doesn't produce λ) there is an equivalent grammar in Chomsky Normal Form

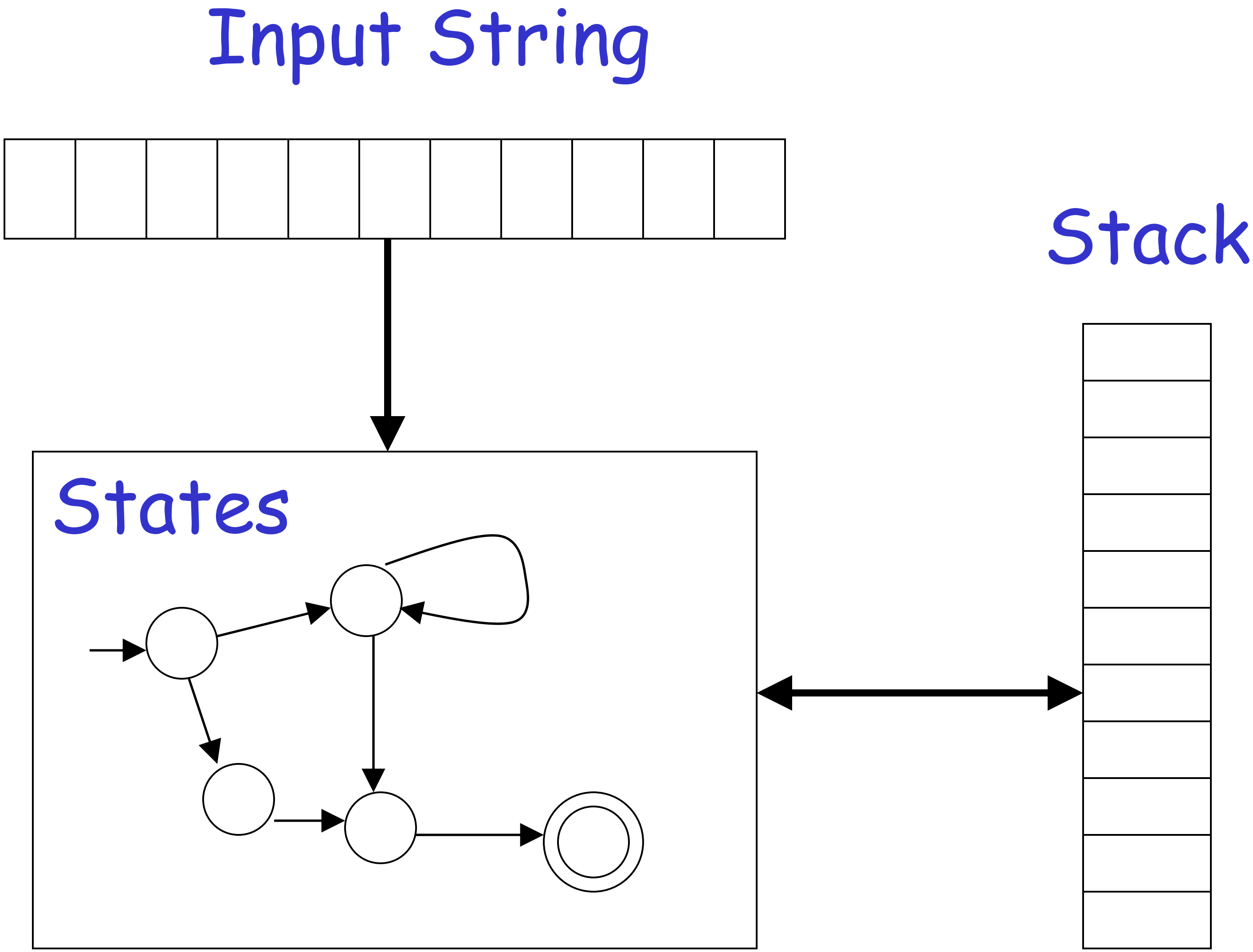
Observations

- Chomsky normal forms are good for parsing and proving theorems

- It is very easy to find the Chomsky normal form for any context-free grammar

Pushdown Automata
PDAs

Pushdown Automaton -- PDA

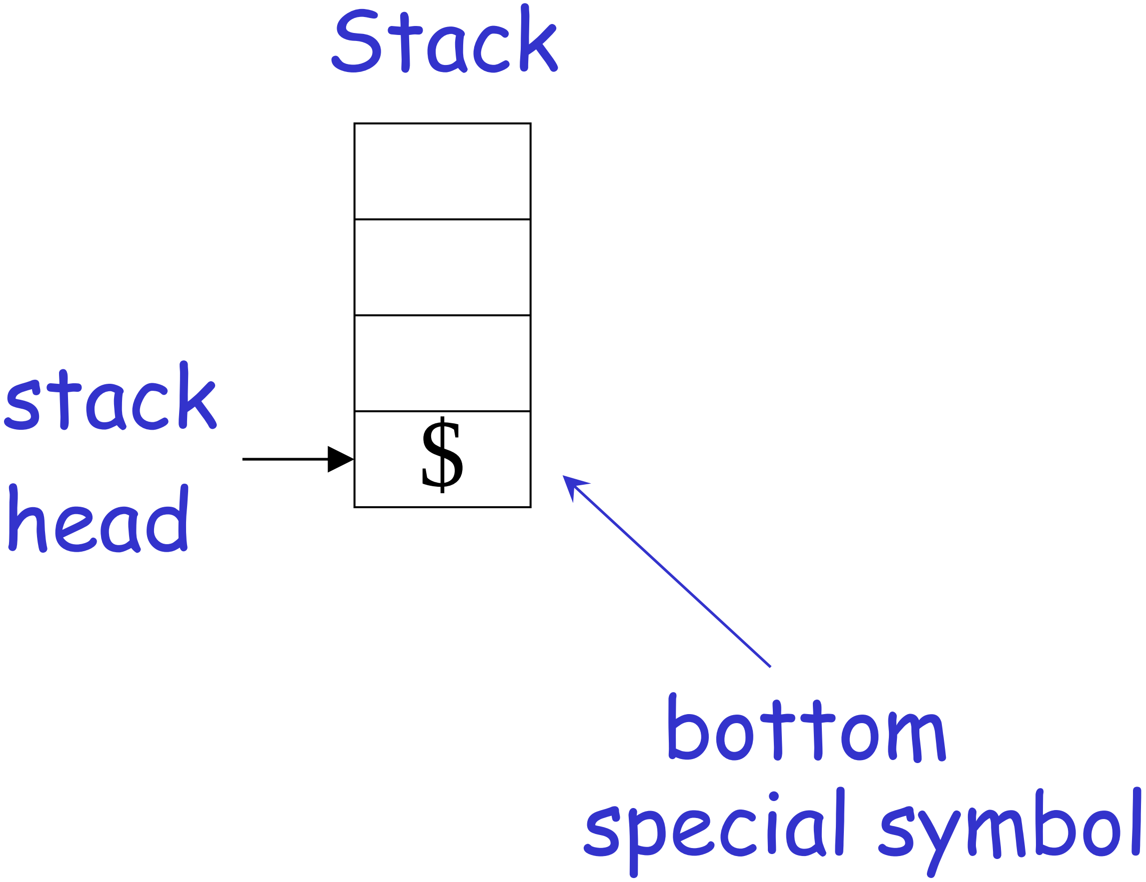


Input String

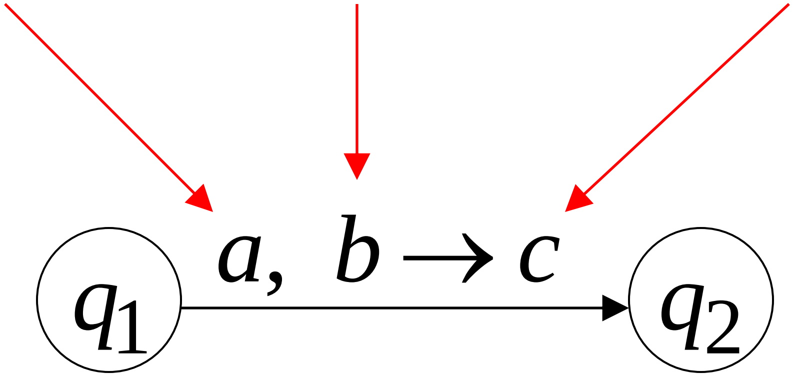
Stack

States

Initial Stack Symbol



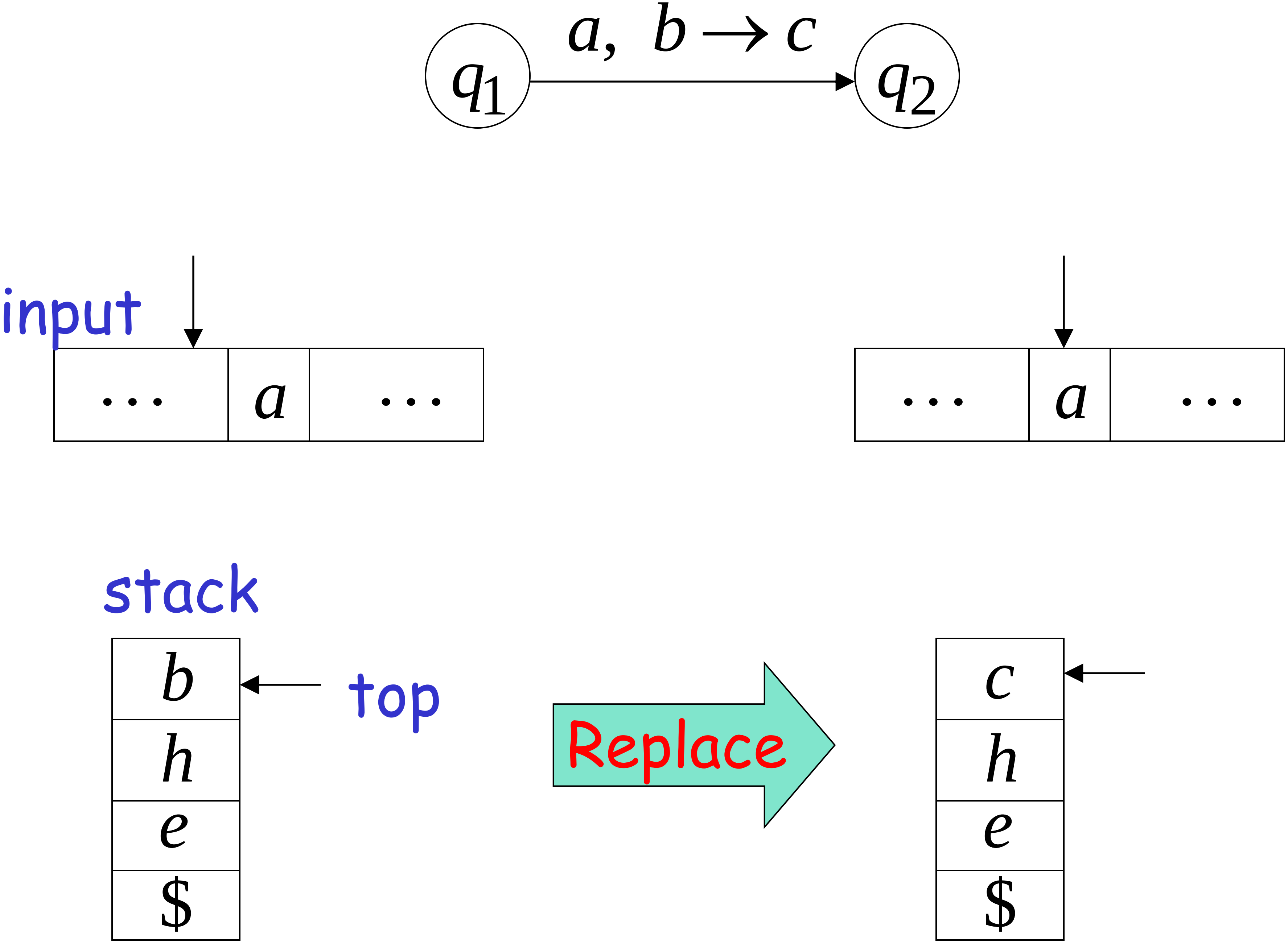
The States

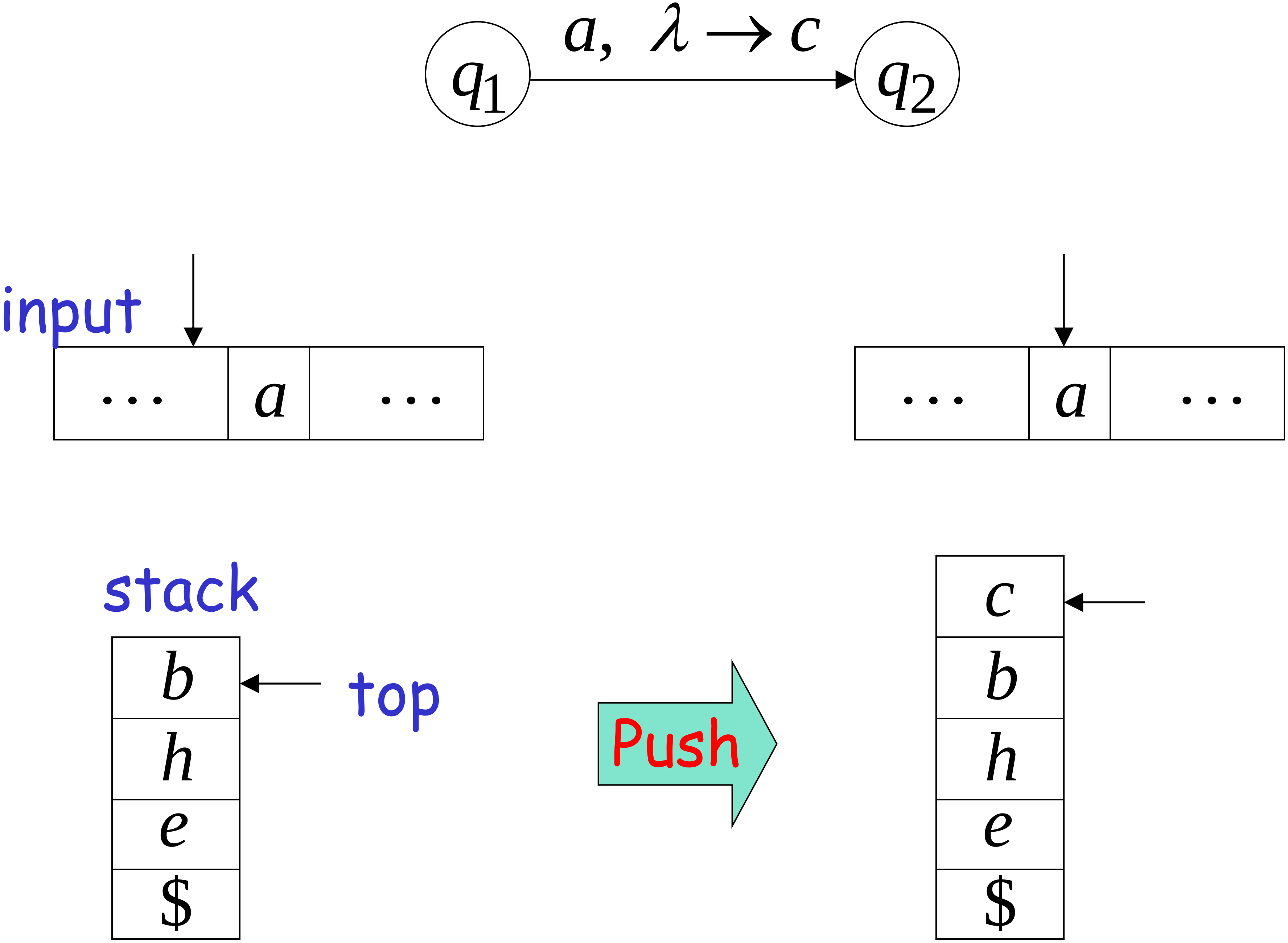


Input symbol

Pop symbol

Push symbol


q_1
$a,\ b \to c$
q_2
input
$\cdots$ a $\cdots$
$\cdots$ a $\cdots$
stack
b
h
e
$\$$
top
Replace
c
h
e
$\$$



q_1 $\xrightarrow{a,\ \lambda \rightarrow c}$ q_2

input

$\cdots$ a $\cdots$

$\cdots$ a $\cdots$

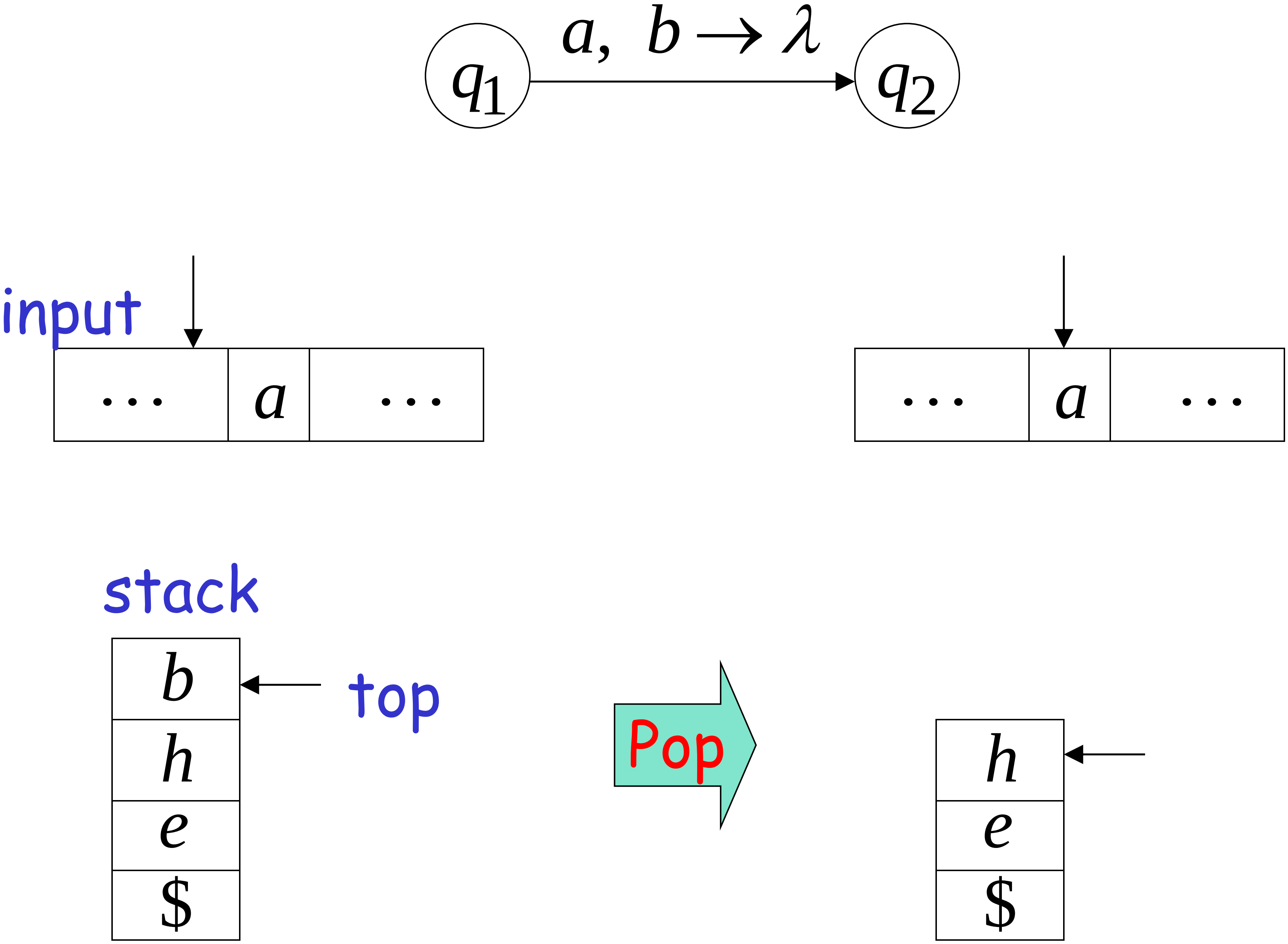
stack

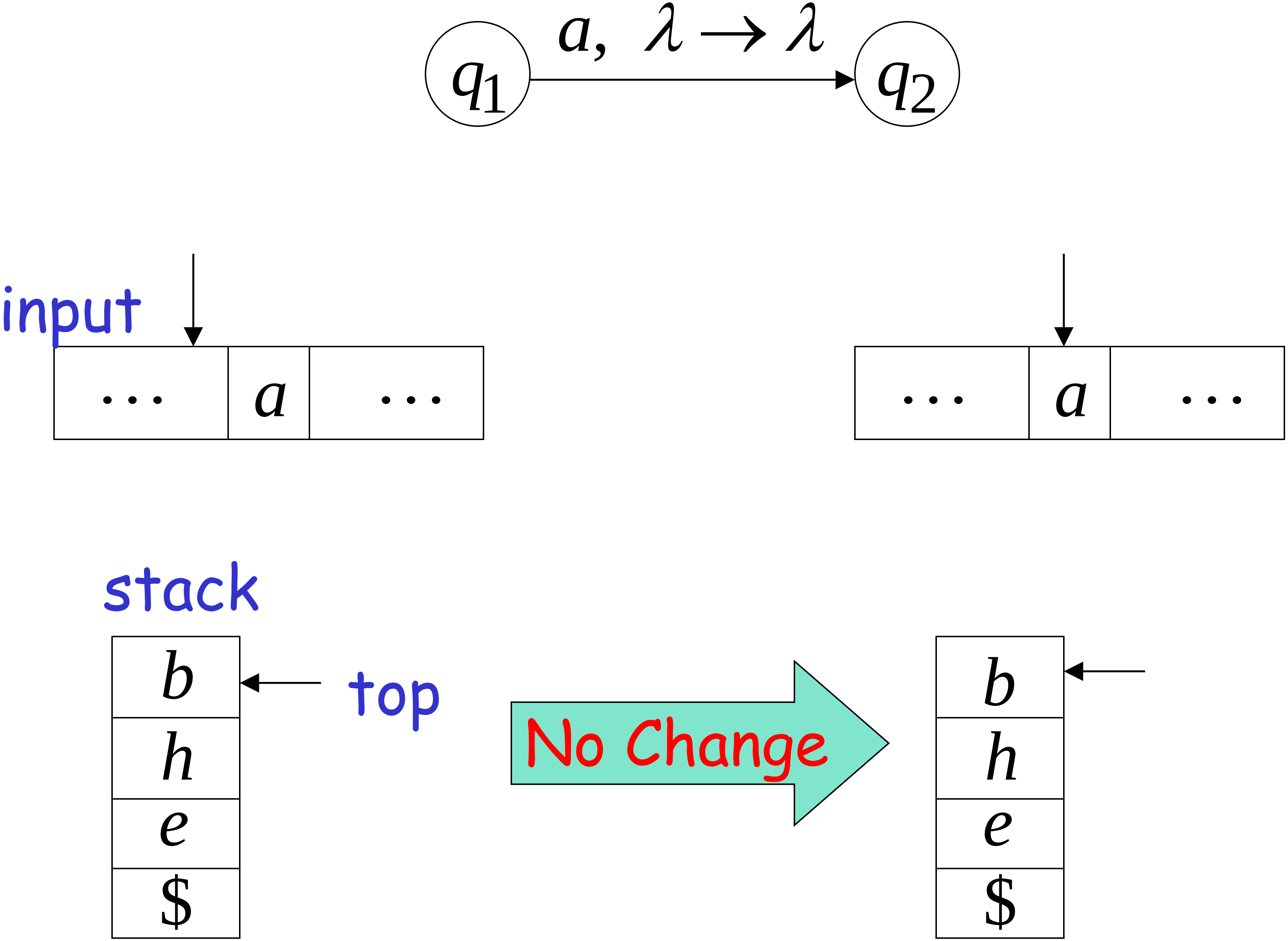
b
h
e
$\$$

top

Push

c
b
h
e
$\$$


q_1
$a,\ b \to \lambda$
q_2
input
$\cdots$ a $\cdots$
$\cdots$ a $\cdots$
stack
b
h
e
$\$$
top
Pop
h
e
$\$$



q_1

$a,\ \lambda \to \lambda$

q_2

input

$\cdots$ a $\cdots$

$\cdots$ a $\cdots$

stack

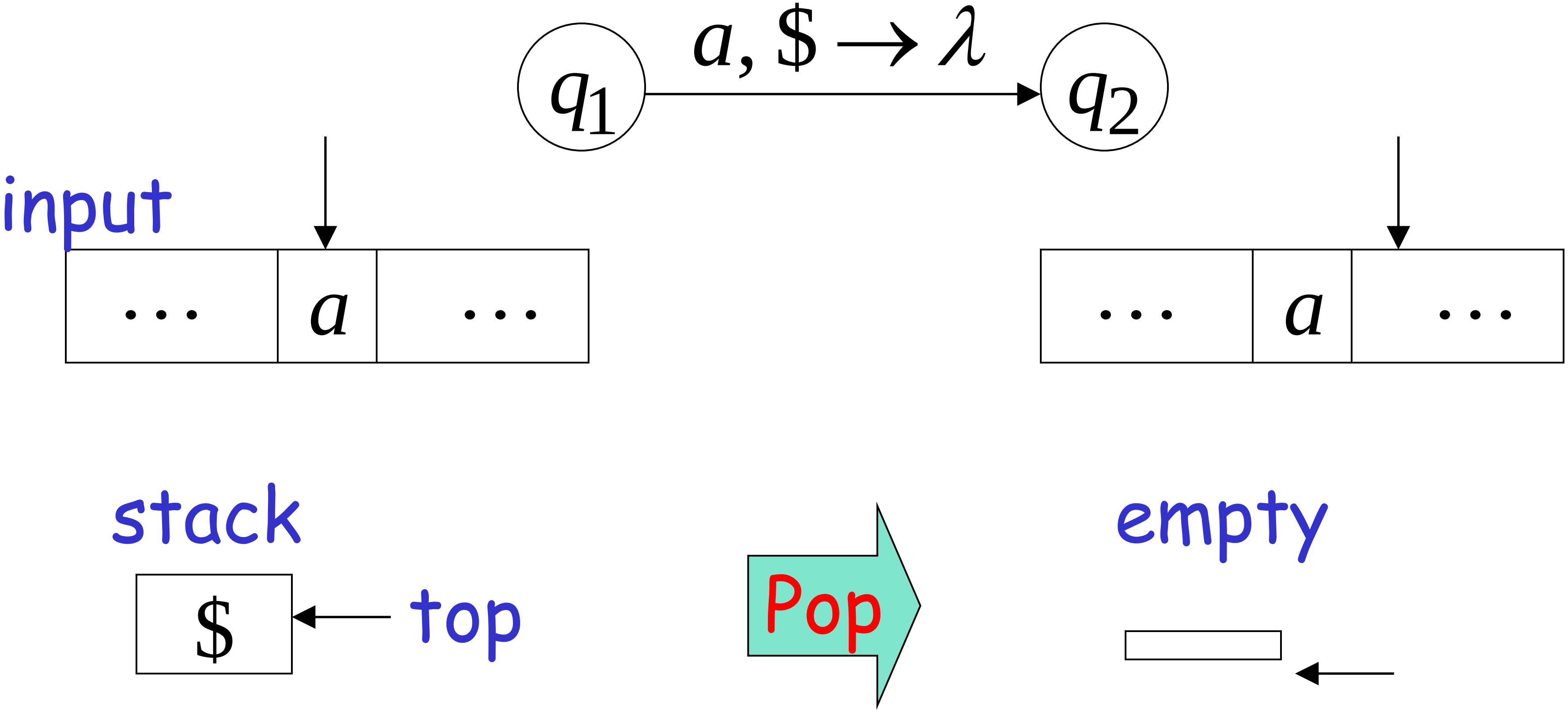
top

No Change

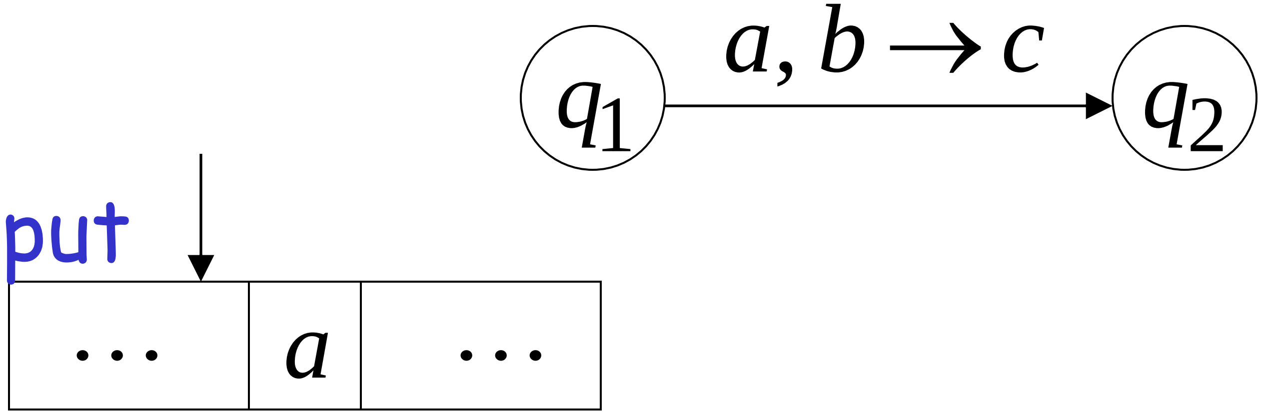
b
h
e
$\$$

b
h
e
$\$$

A Possible Transition



A Bad Transition



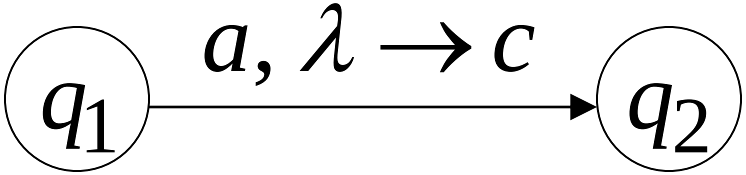
input

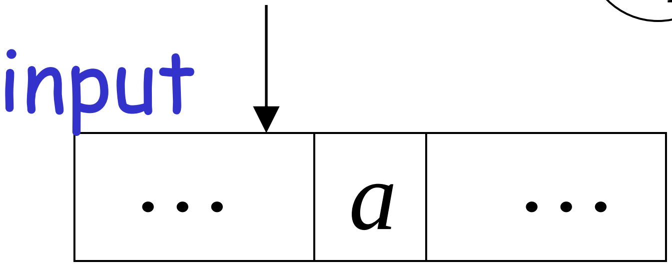
Empty stack

HALT

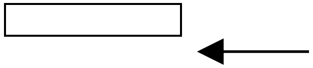
The automaton **Halts** in state q_1
and **Rejects** the input string

A Bad Transition





Empty stack

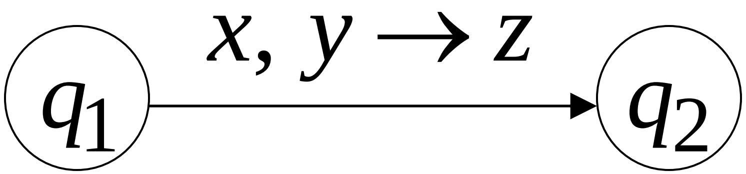


HALT

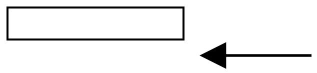
The automaton **Halts** in state q_1

and **Rejects** the input string

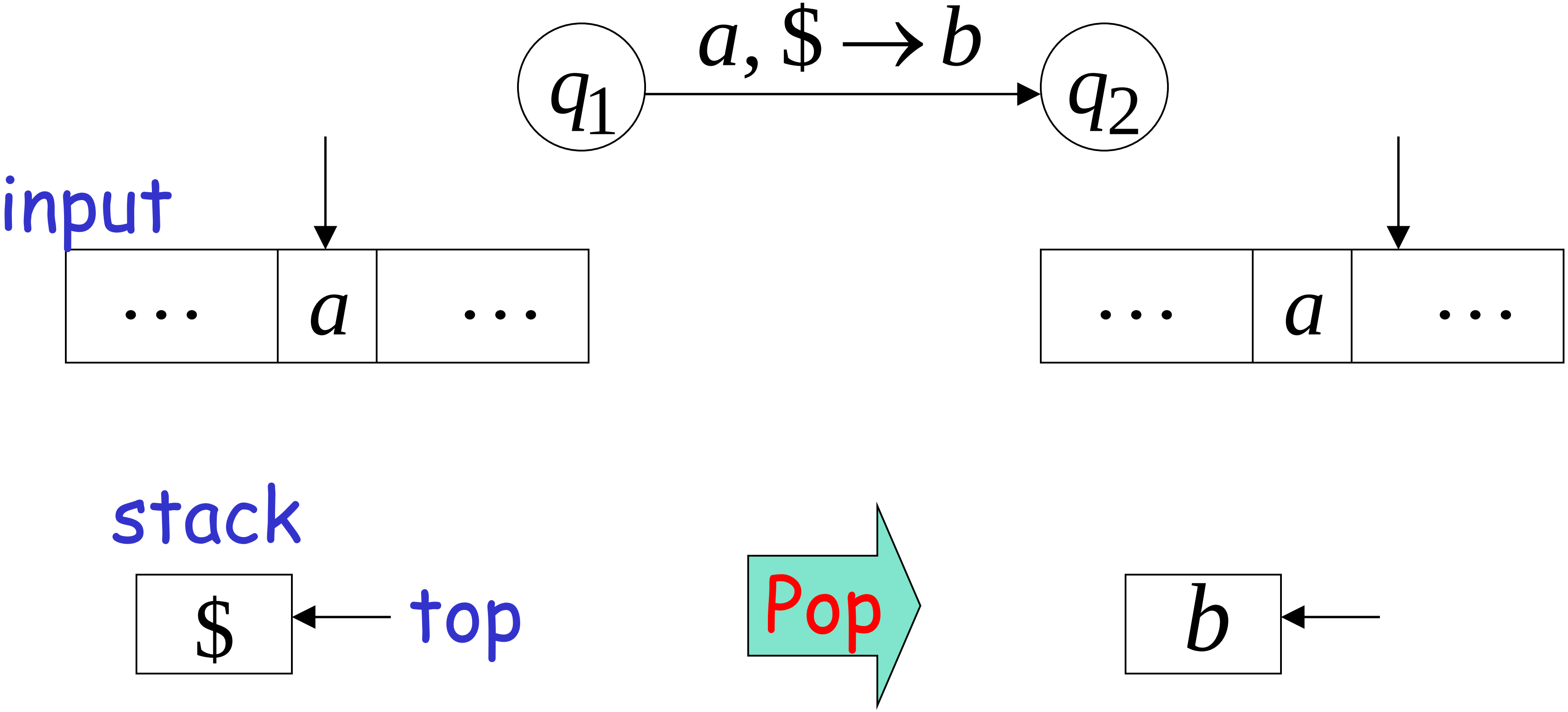
No transition is allowed to be followed When the stack is empty



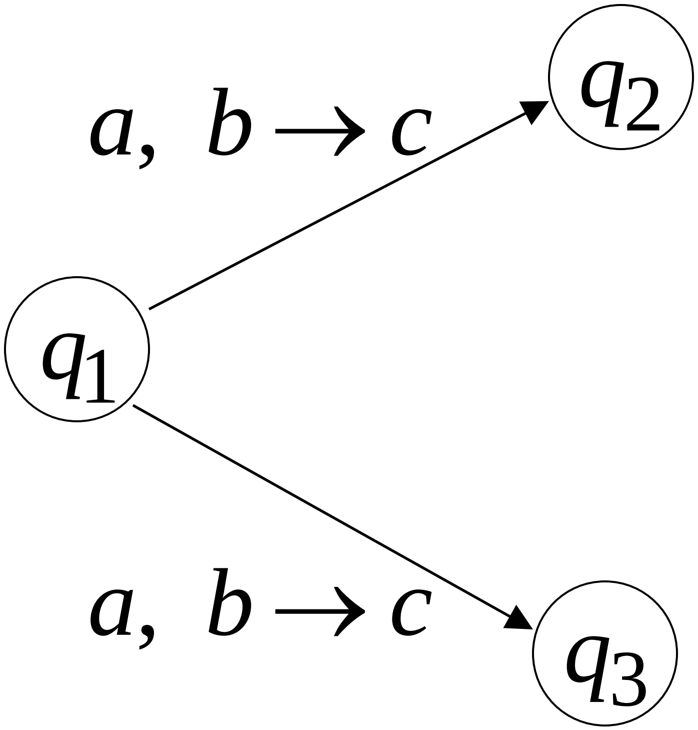
Empty stack

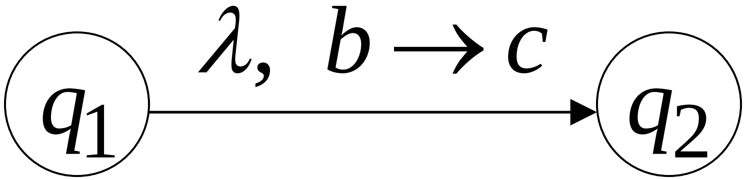


A Good Transition



Non-Determinism



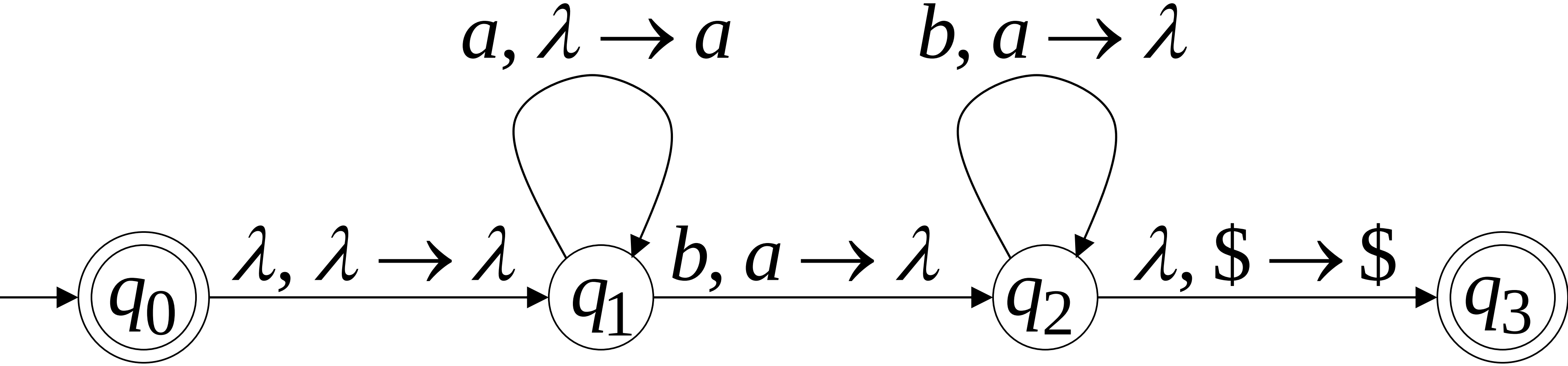


$\lambda -$ transition

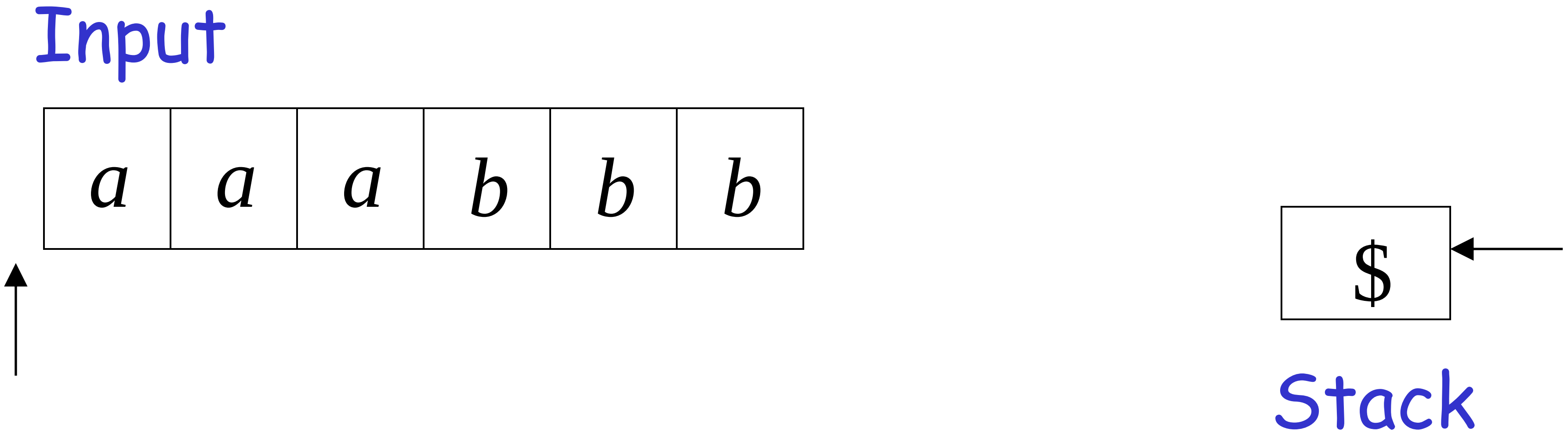
These are allowed transitions in a Non-deterministic PDA (NPDA)

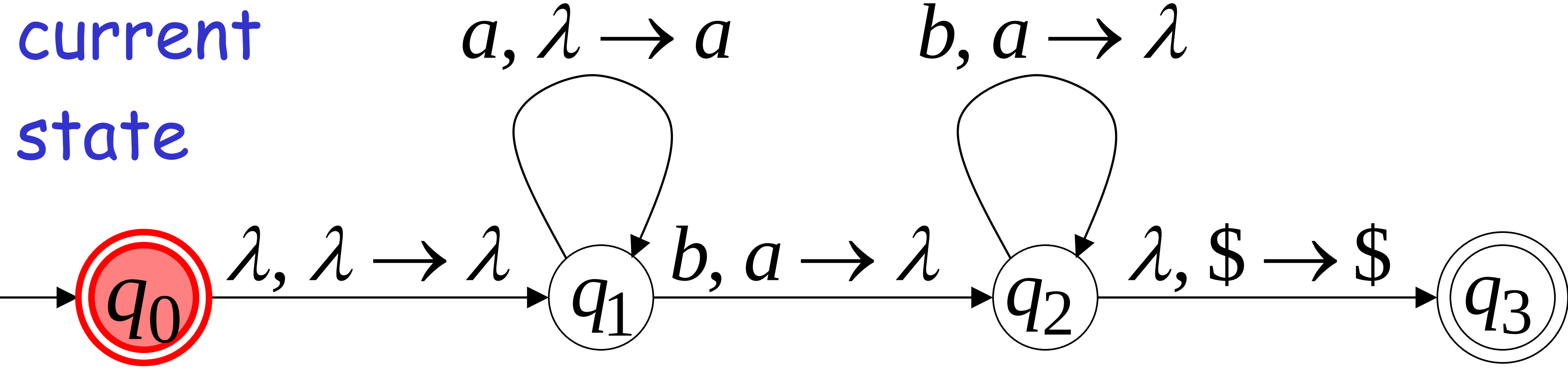
NPDA: Non-Deterministic PDA

Example:



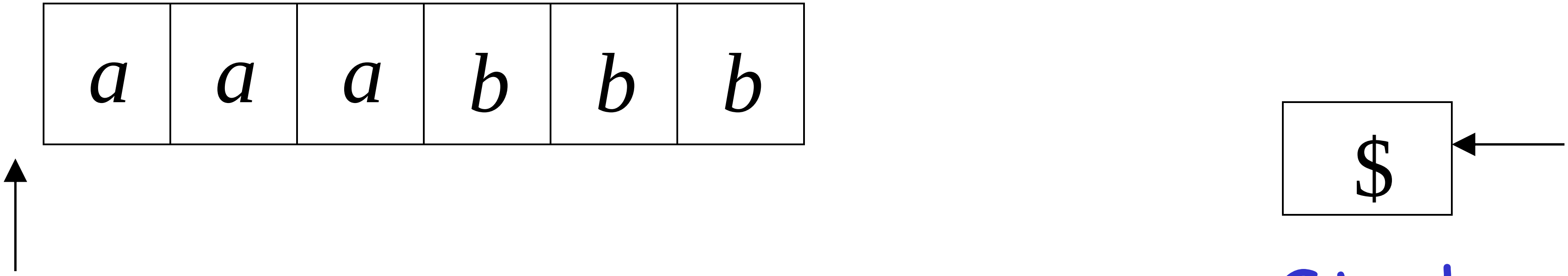
Execution Example: Time 0



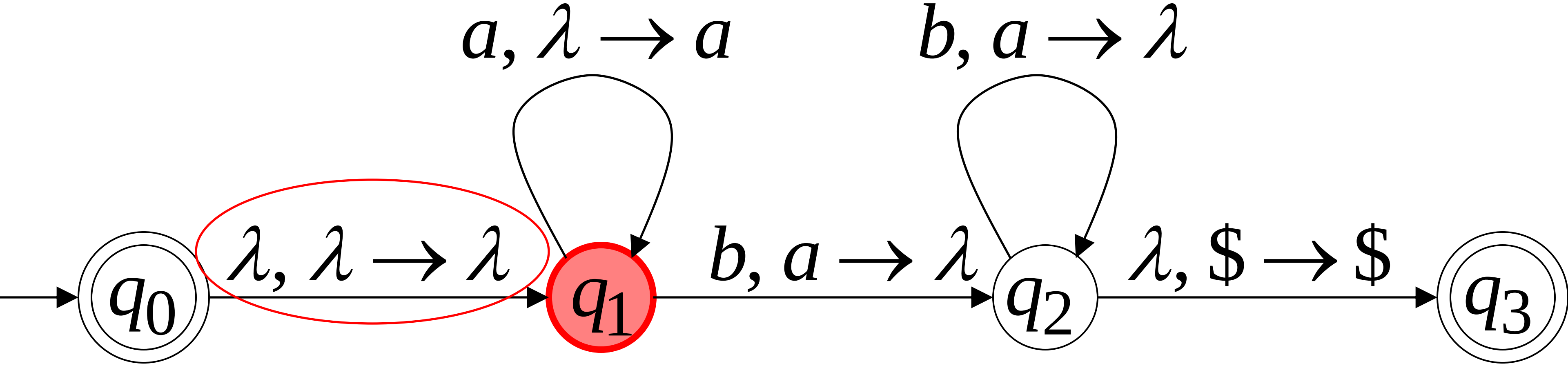


Time 1

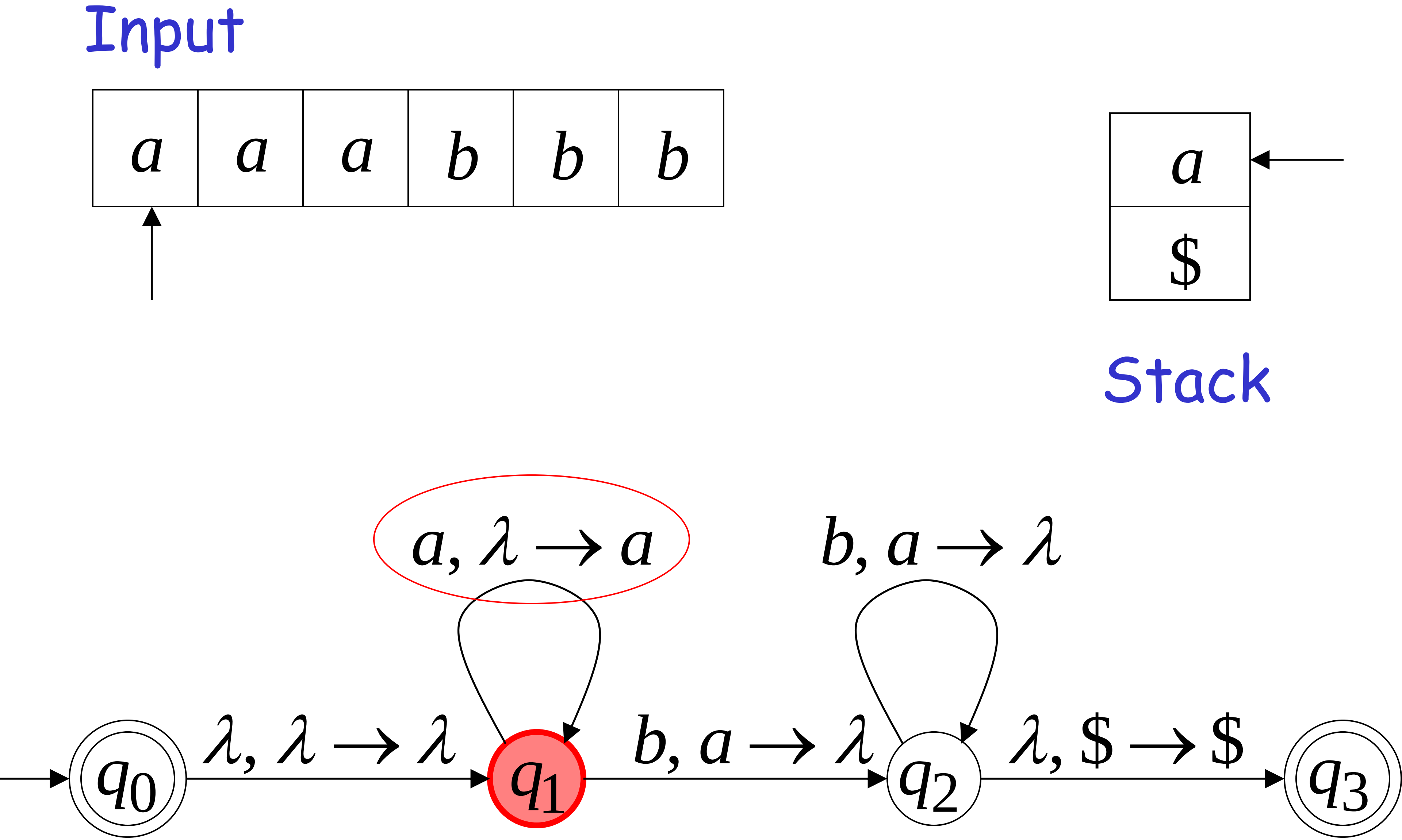
Input



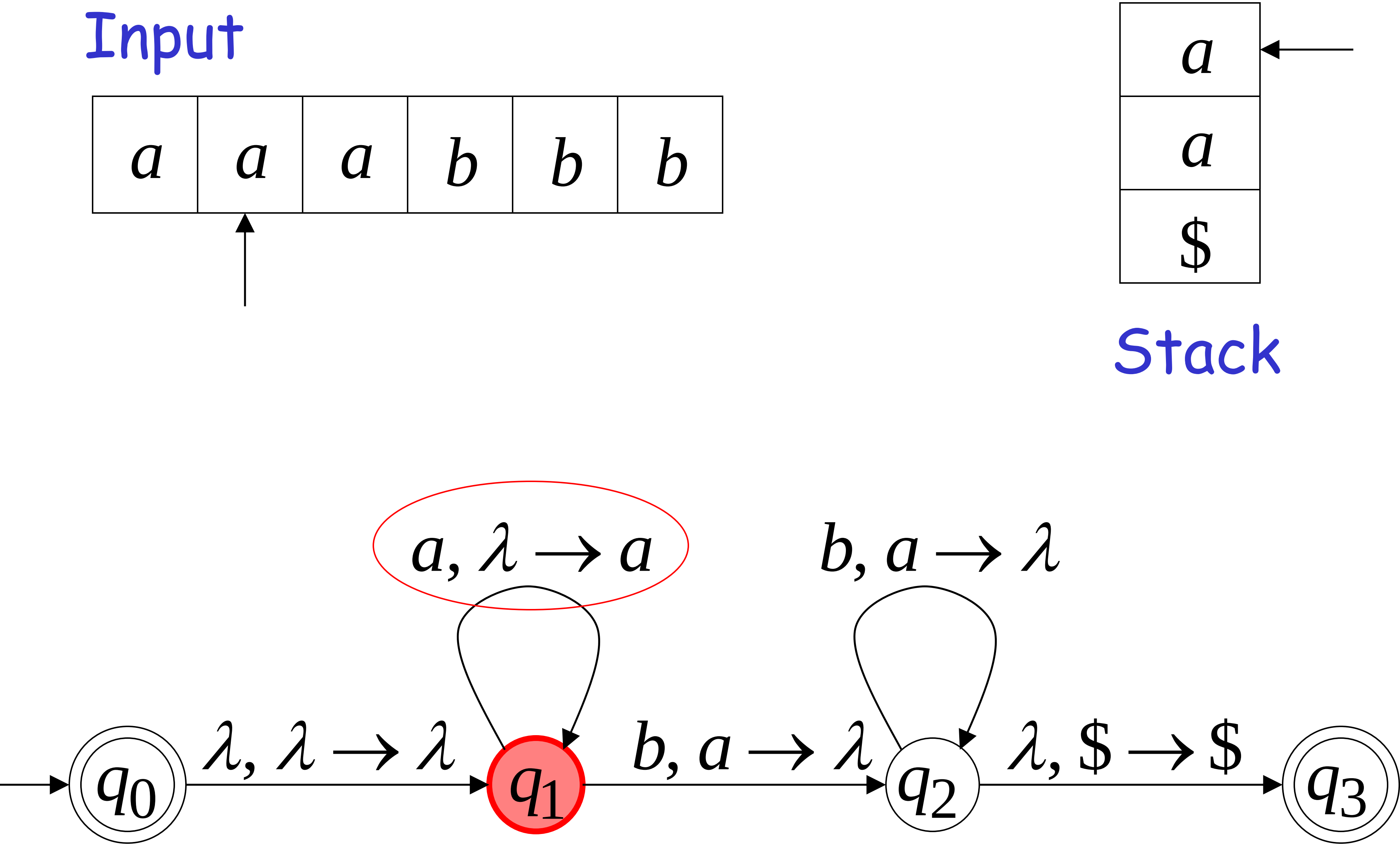
Stack



Time 2

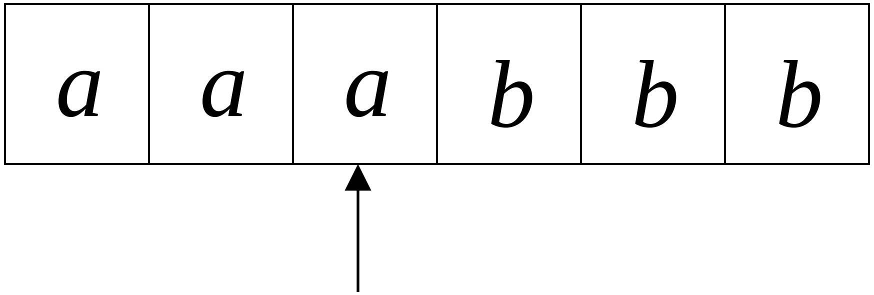

Input
a a a b b b
a
$
Stack
$a, \lambda \rightarrow a$
$b, a \rightarrow \lambda$
$\lambda, \lambda \rightarrow \lambda$
$b, a \rightarrow \lambda$
$\lambda, \$ \rightarrow \$$
q_0
q_1
q_2
q_3

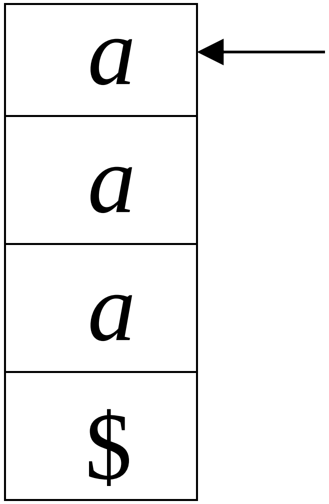
Time 3


Input
a a a b b b
a
a
$
Stack
$a, \lambda \rightarrow a$
$b, a \rightarrow \lambda$
$\lambda, \lambda \rightarrow \lambda$
$b, a \rightarrow \lambda$
$\lambda, \$ \rightarrow \$$
q_0
q_1
q_2
q_3

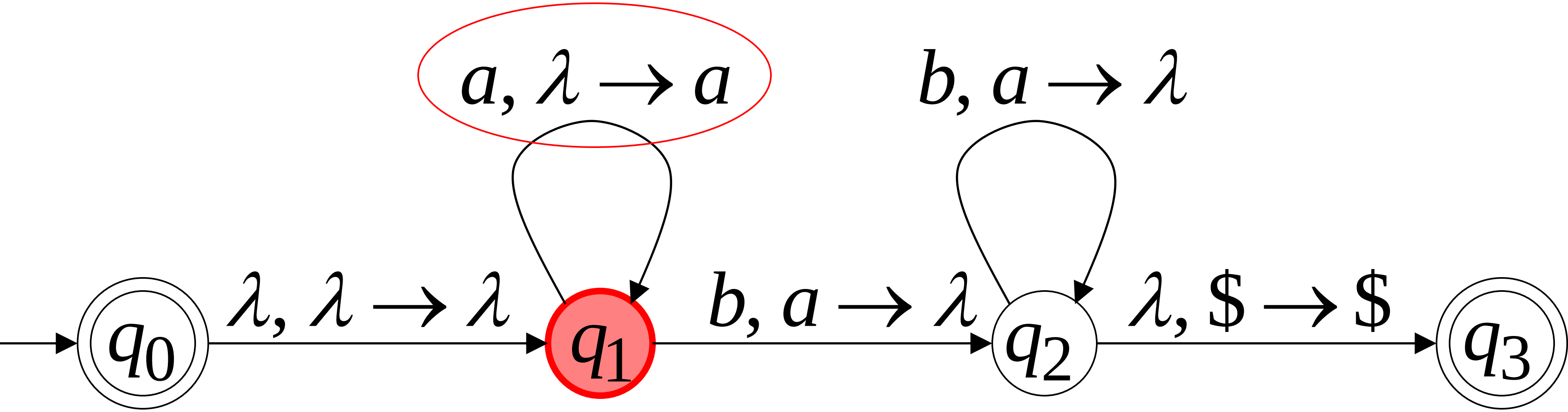
Time 4

Input



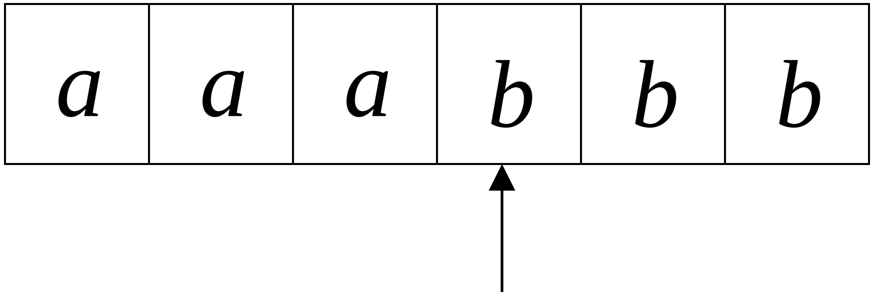


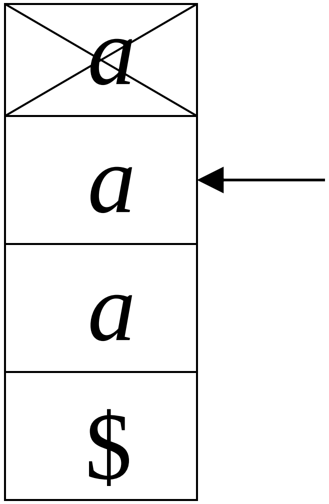
Stack



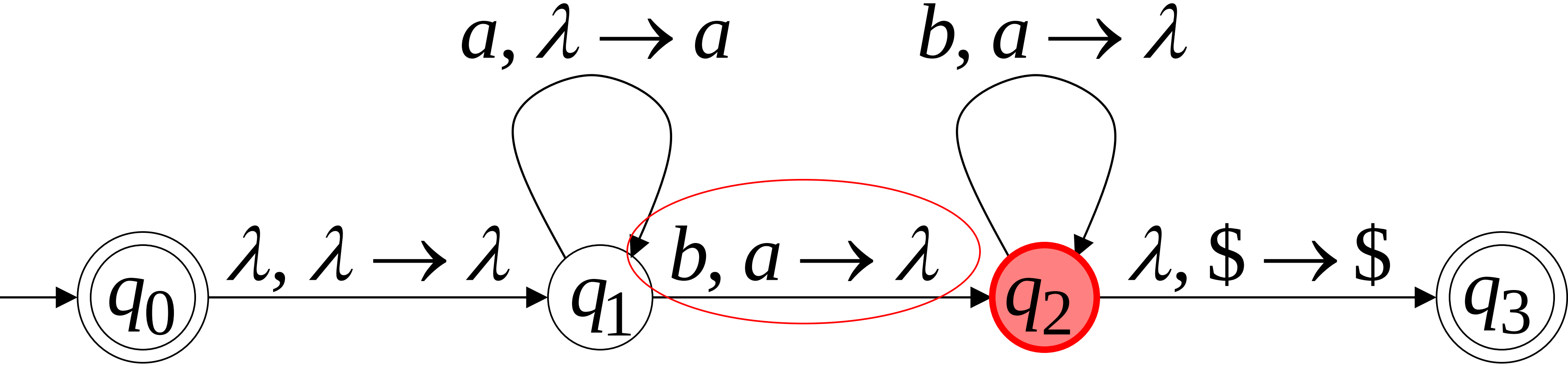
Time 5

Input

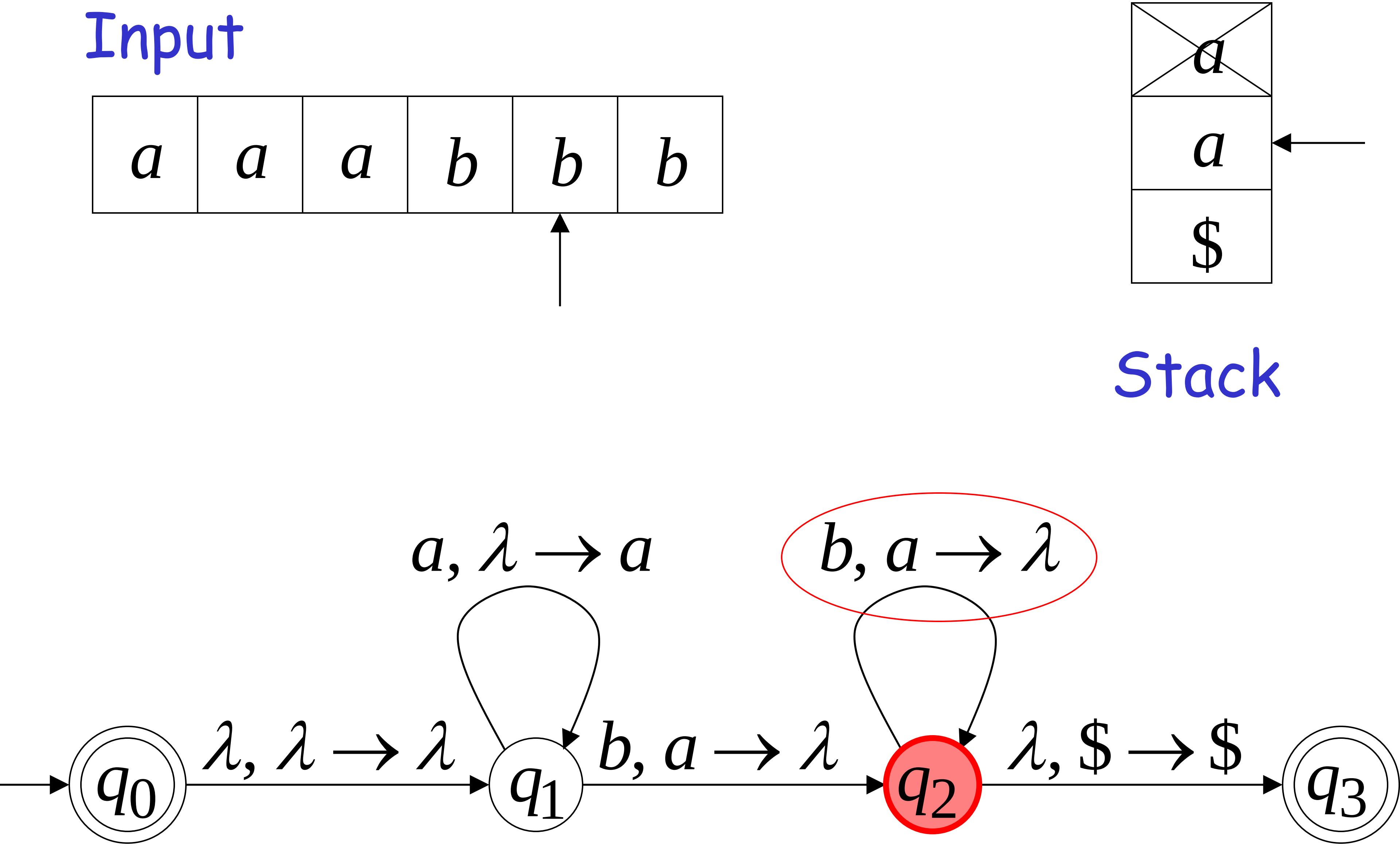




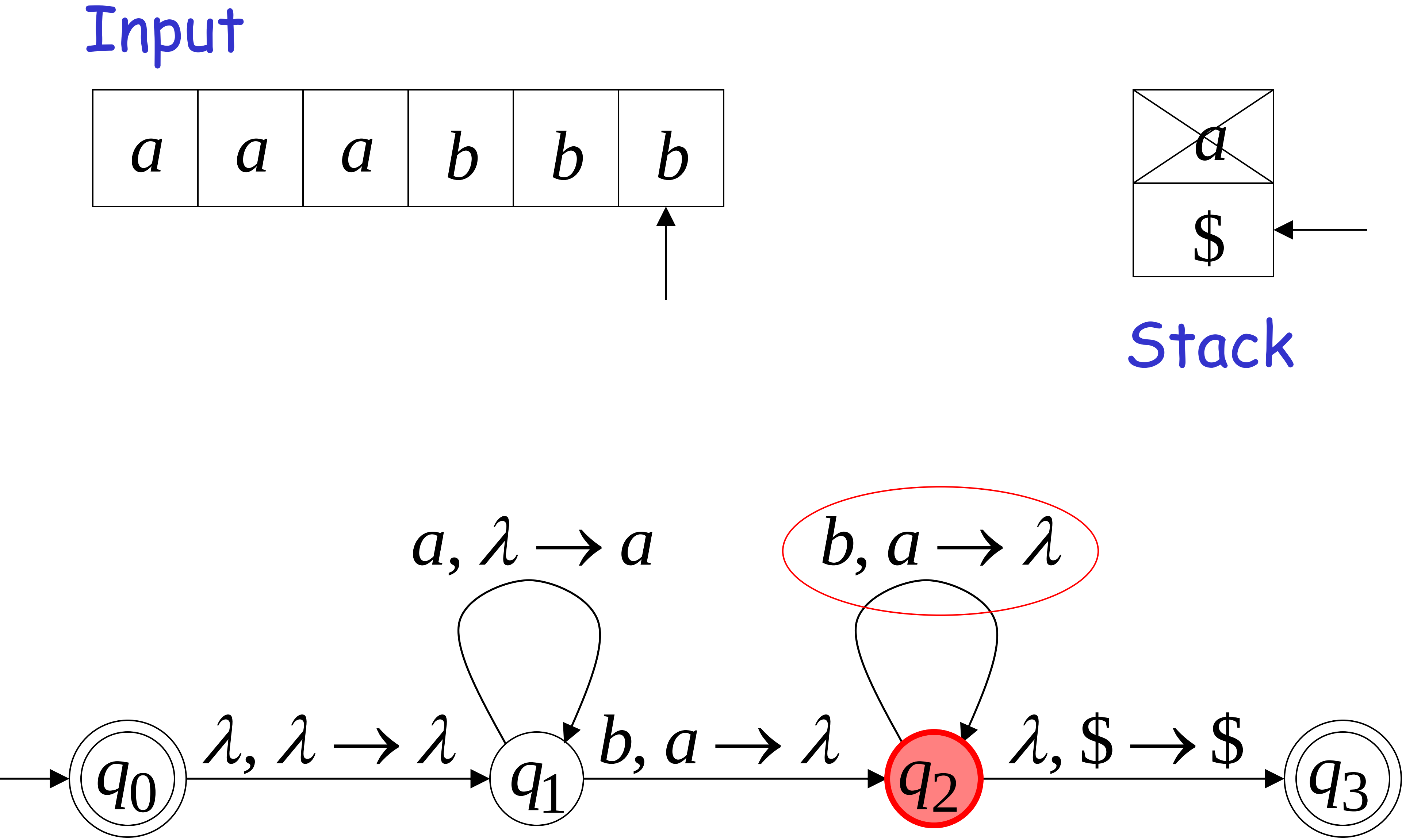
Stack



Time 6

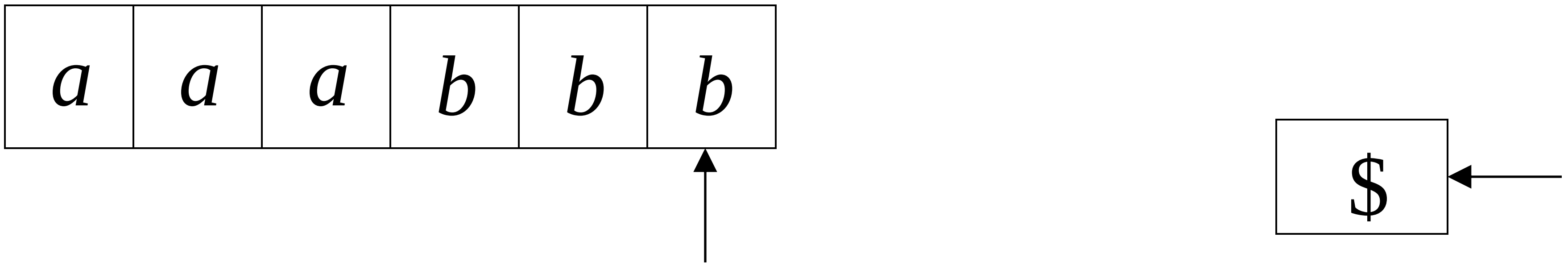

Input
a a a b b b
$\cancel{a}$
a
$
Stack
$a, \lambda \rightarrow a$
$b, a \rightarrow \lambda$
$\lambda, \lambda \rightarrow \lambda$
$b, a \rightarrow \lambda$
$\lambda, \$ \rightarrow \$$
q_0
q_1
q_2
q_3

Time 7

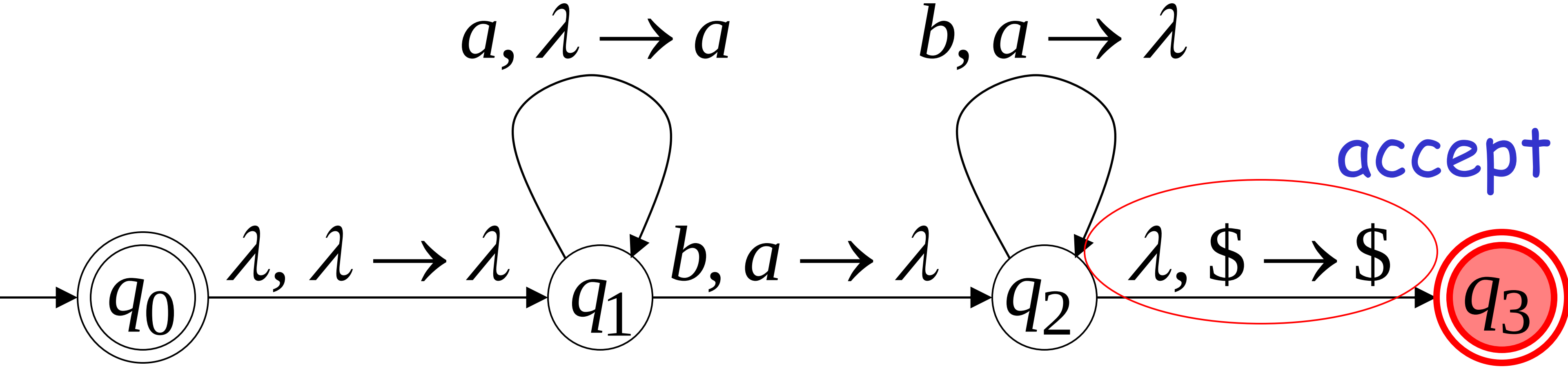

Input
a
a
a
b
b
b
a
$
Stack
$a, \lambda \to a$
$b, a \to \lambda$
$\lambda, \lambda \to \lambda$
$b, a \to \lambda$
$\lambda, \$ \to \$$
q_0
q_1
q_2
q_3

Time 8

Input



Stack



A string is accepted if there is
a computation such that:

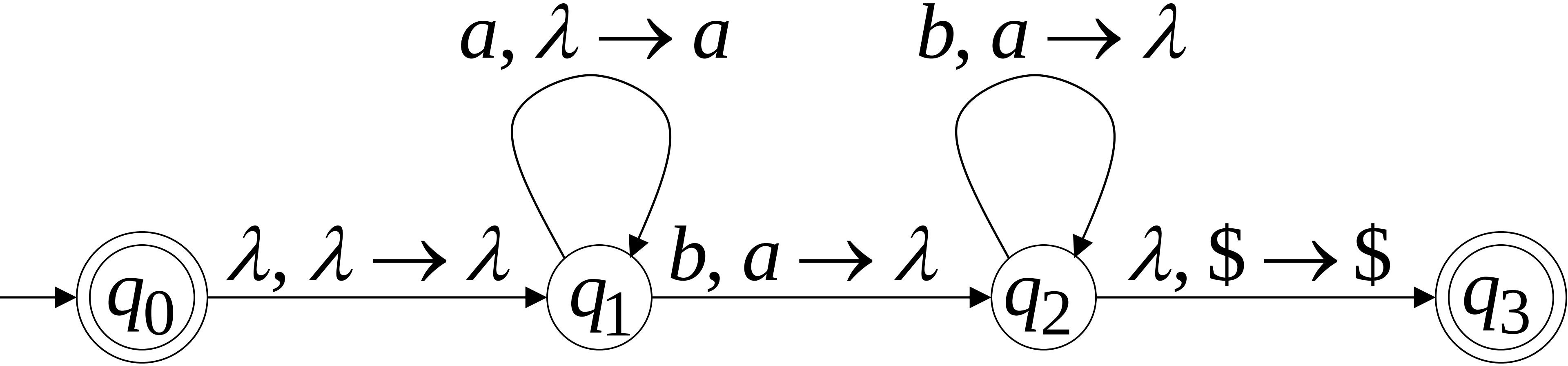
All the input is consumed

AND

The last state is a final state

At the end of the computation,
we do not care about the stack contents

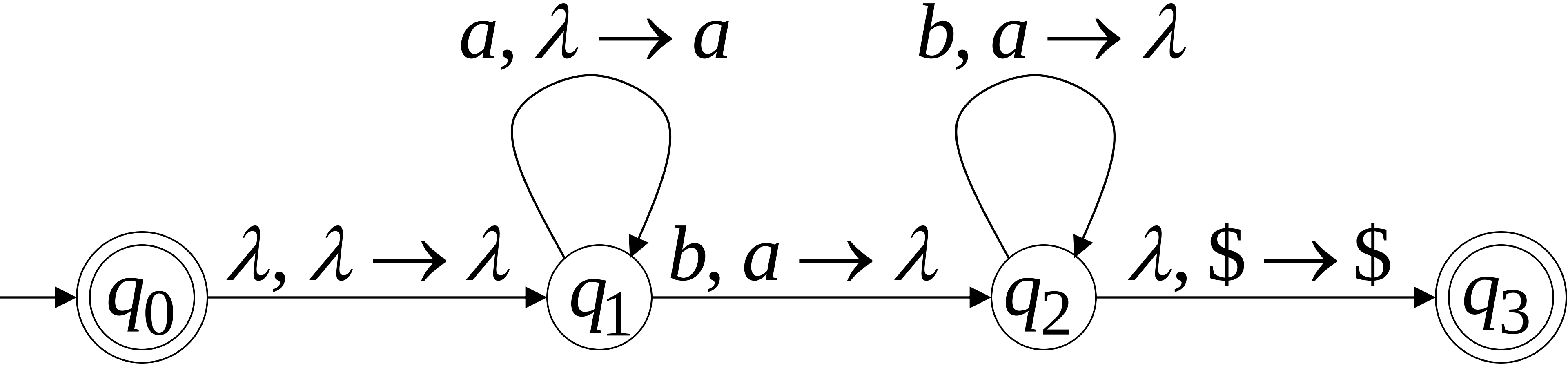
The input string $aaabbb$
is accepted by the NPDA:



In general,

$$L = \{a^n b^n : n \geq 0\}$$

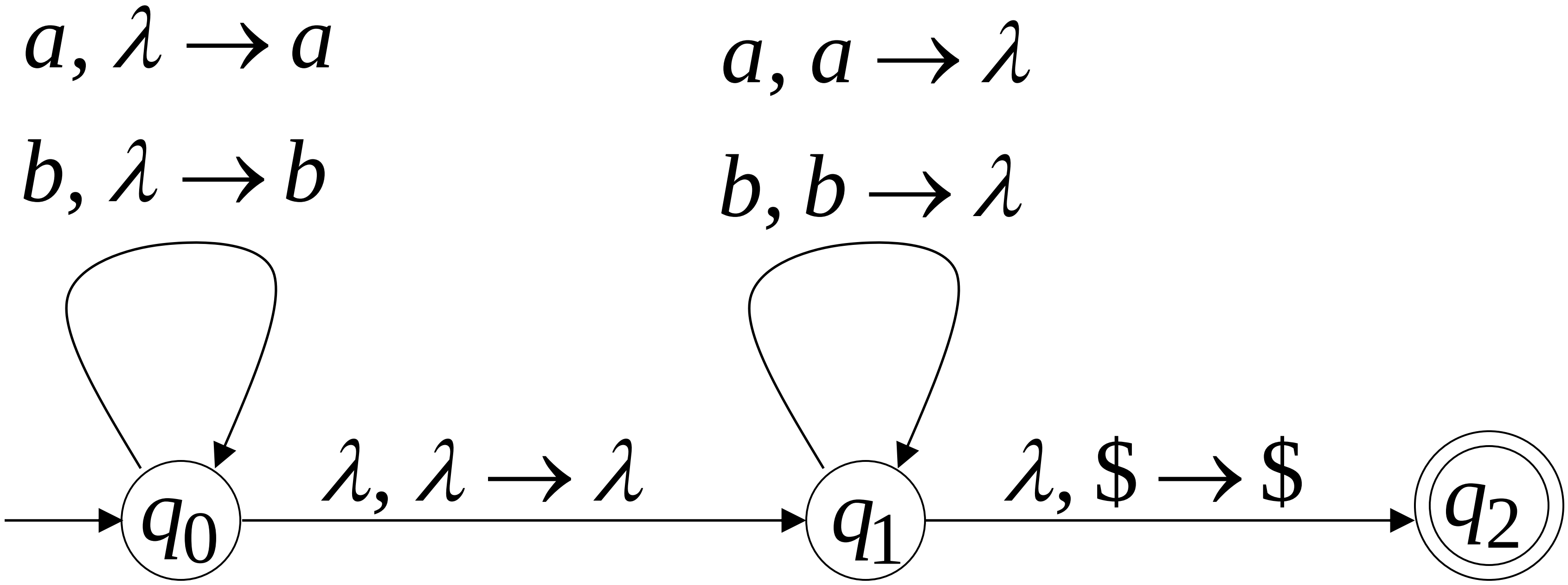
is the language accepted by the NPDA:



Another NPDA example

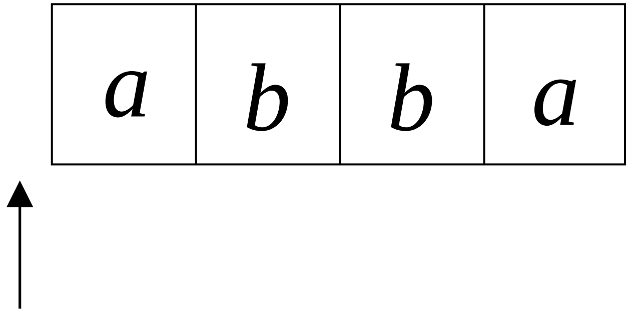
NPDA M

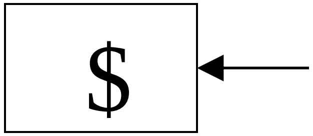
$$L(M) = \{ww^R\}$$



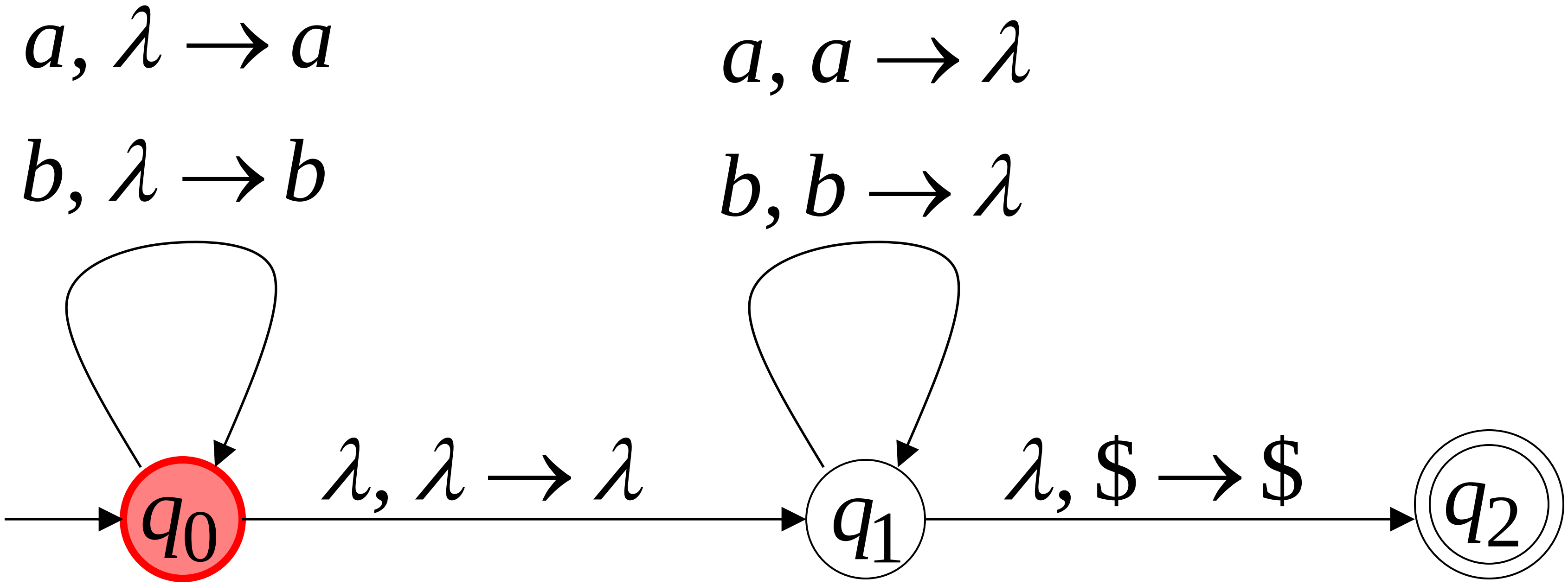
Execution Example: Time 0

Input

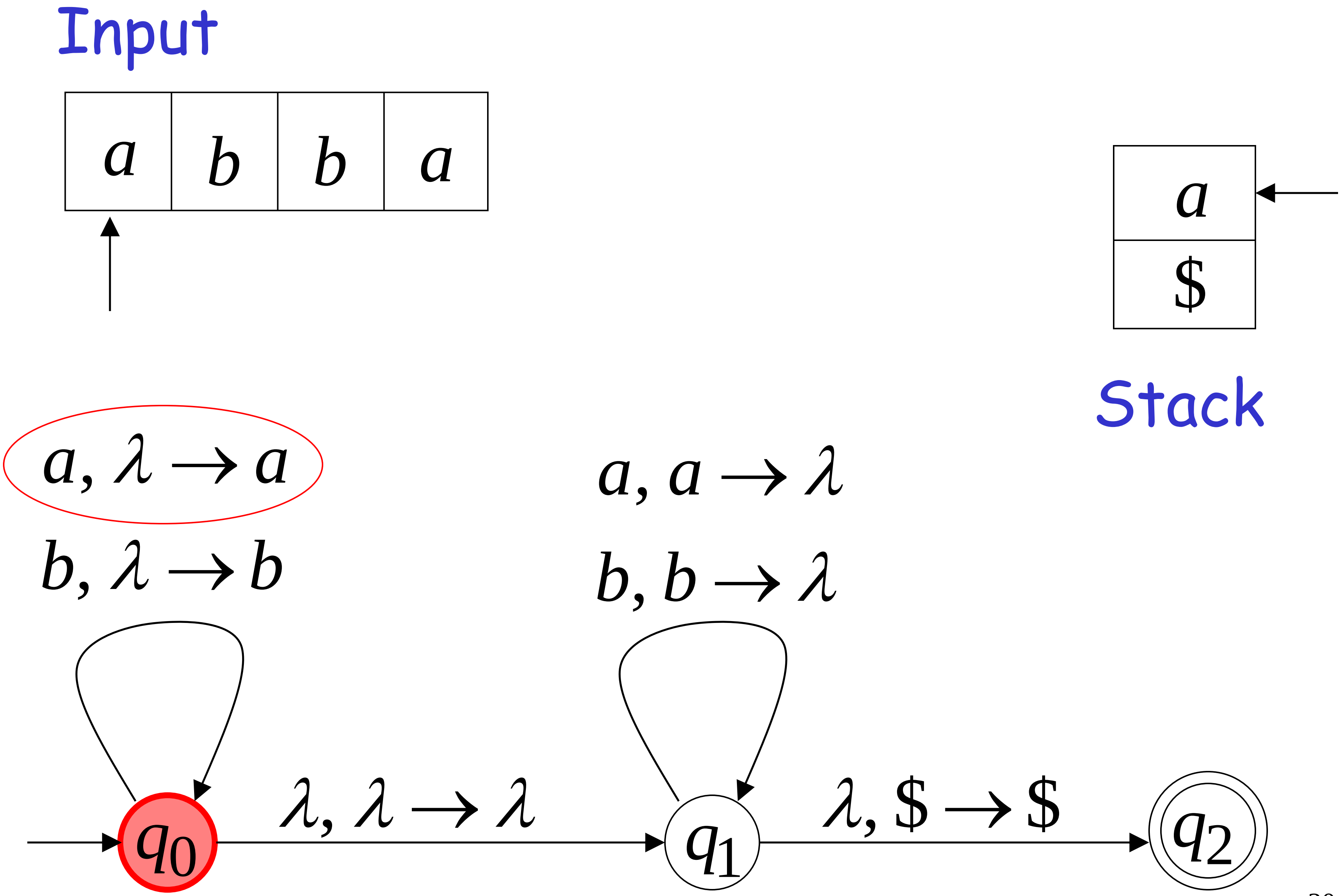




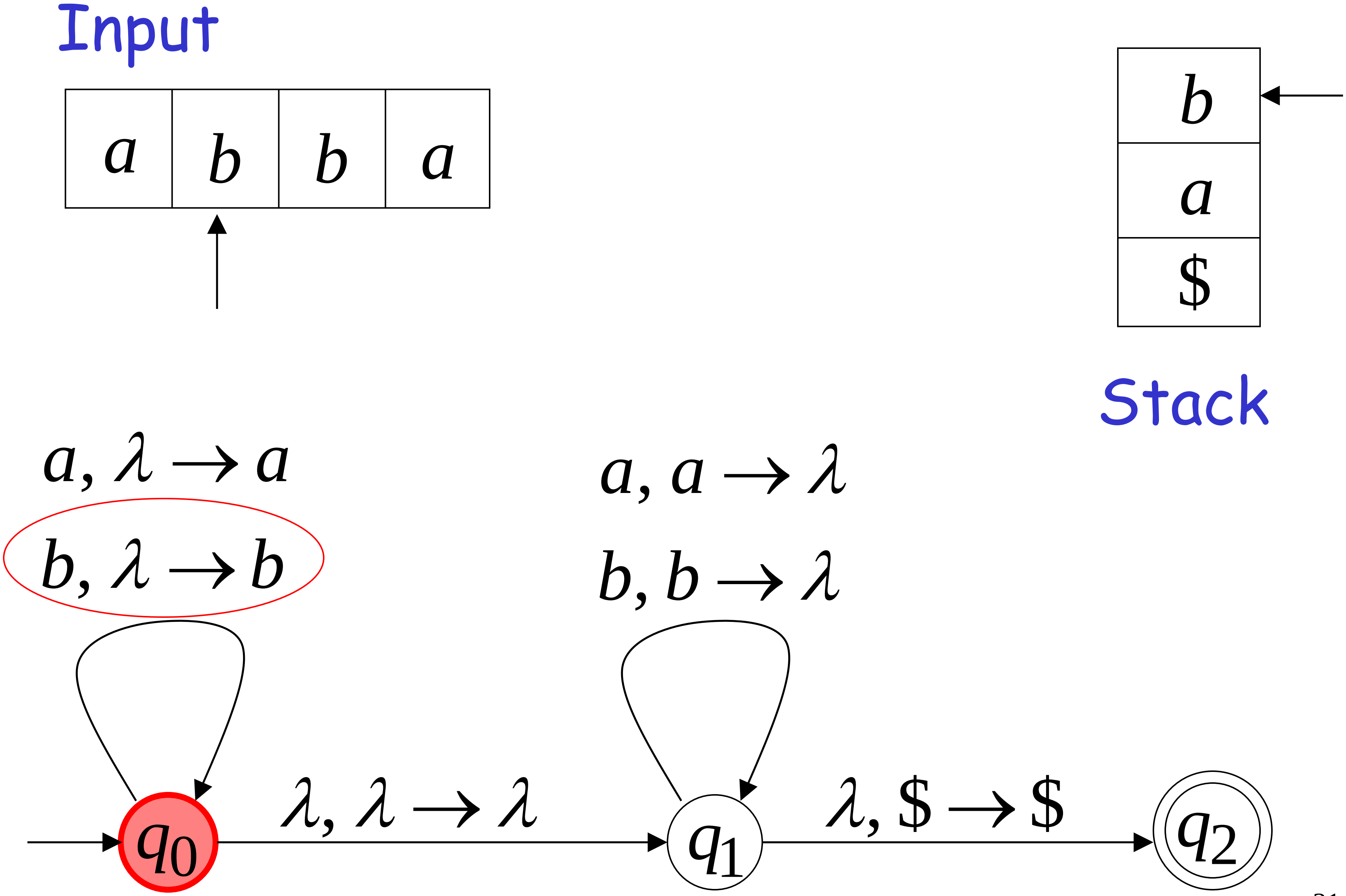
Stack



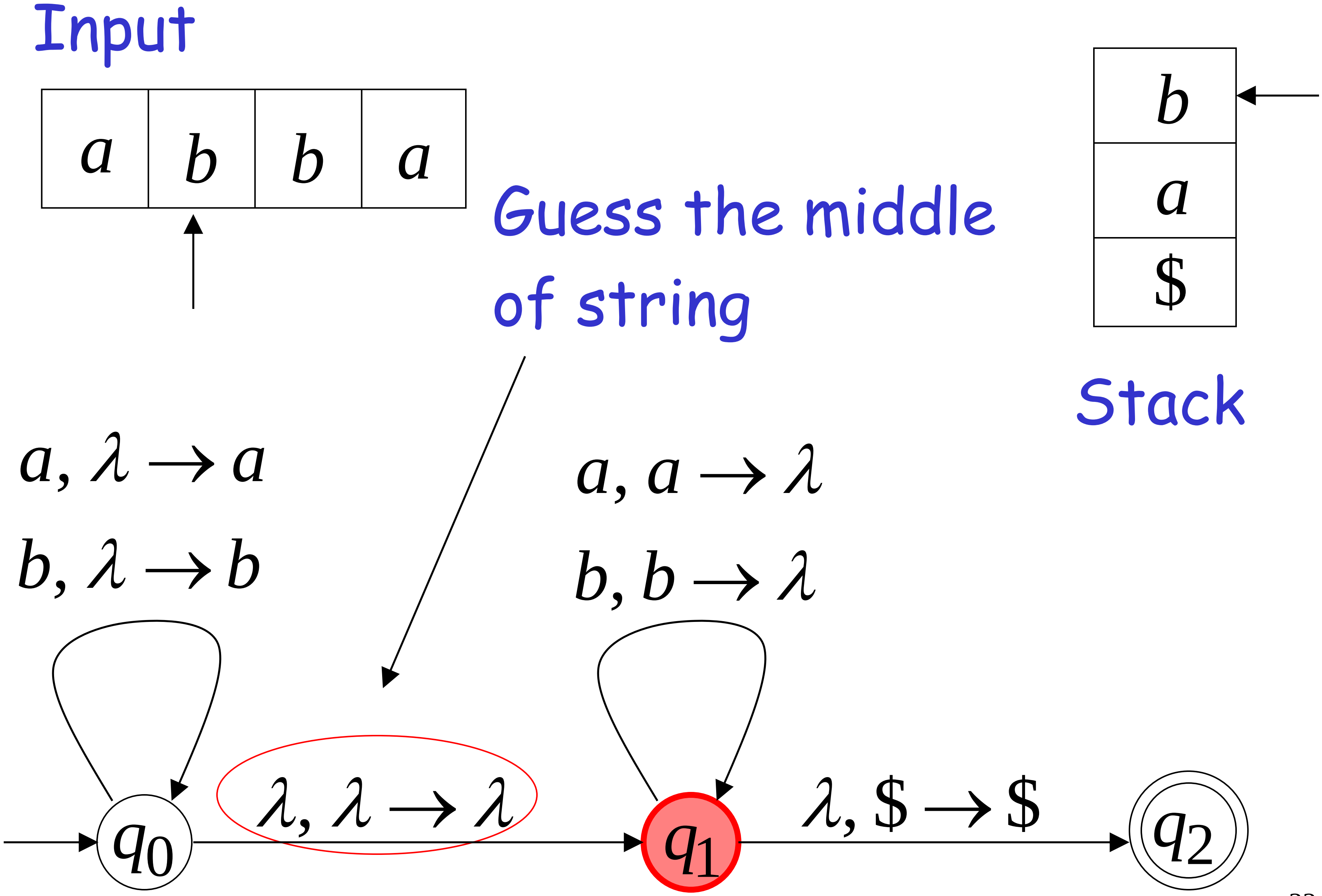
Time 1


Input
a b b a
a
$\$$
Stack
$a, \lambda \rightarrow a$
$b, \lambda \rightarrow b$
$a, a \rightarrow \lambda$
$b, b \rightarrow \lambda$
$\lambda, \lambda \rightarrow \lambda$
$\lambda, \$ \rightarrow \$$
q_0
q_1
q_2

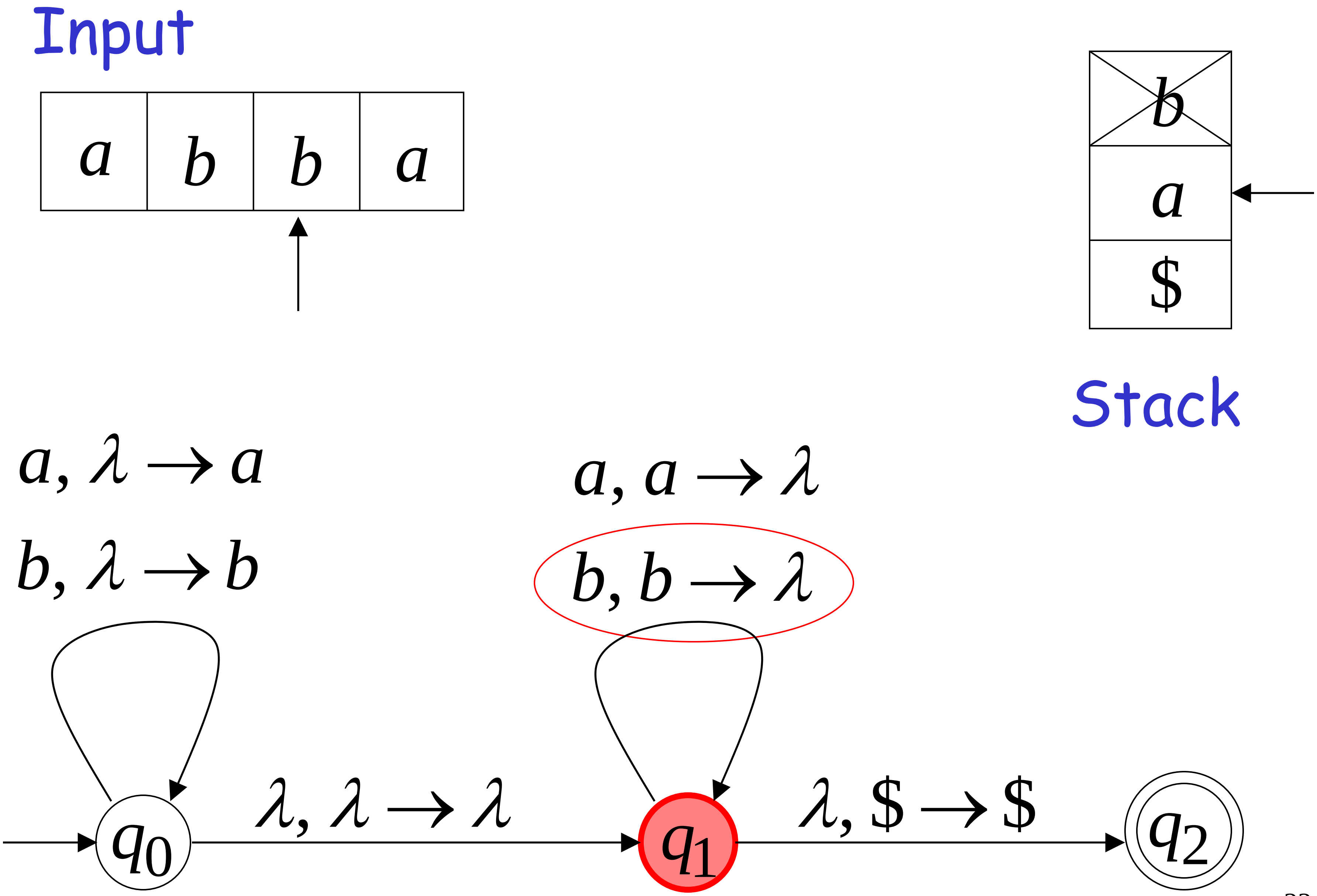
Time 2


Input
a b b a
b
a
$\$$
Stack
$a, \lambda \rightarrow a$
$b, \lambda \rightarrow b$
$a, a \rightarrow \lambda$
$b, b \rightarrow \lambda$
$\lambda, \lambda \rightarrow \lambda$
$\lambda, \$ \rightarrow \$$
q_0
q_1
q_2

Time 3


Input
a
b
b
a
Guess the middle of string
b
a
$\$$
Stack
$a, \lambda \rightarrow a$
$b, \lambda \rightarrow b$
$a, a \rightarrow \lambda$
$b, b \rightarrow \lambda$
$\lambda, \lambda \rightarrow \lambda$
$\lambda, \$ \rightarrow \$$
q_0
q_1
q_2

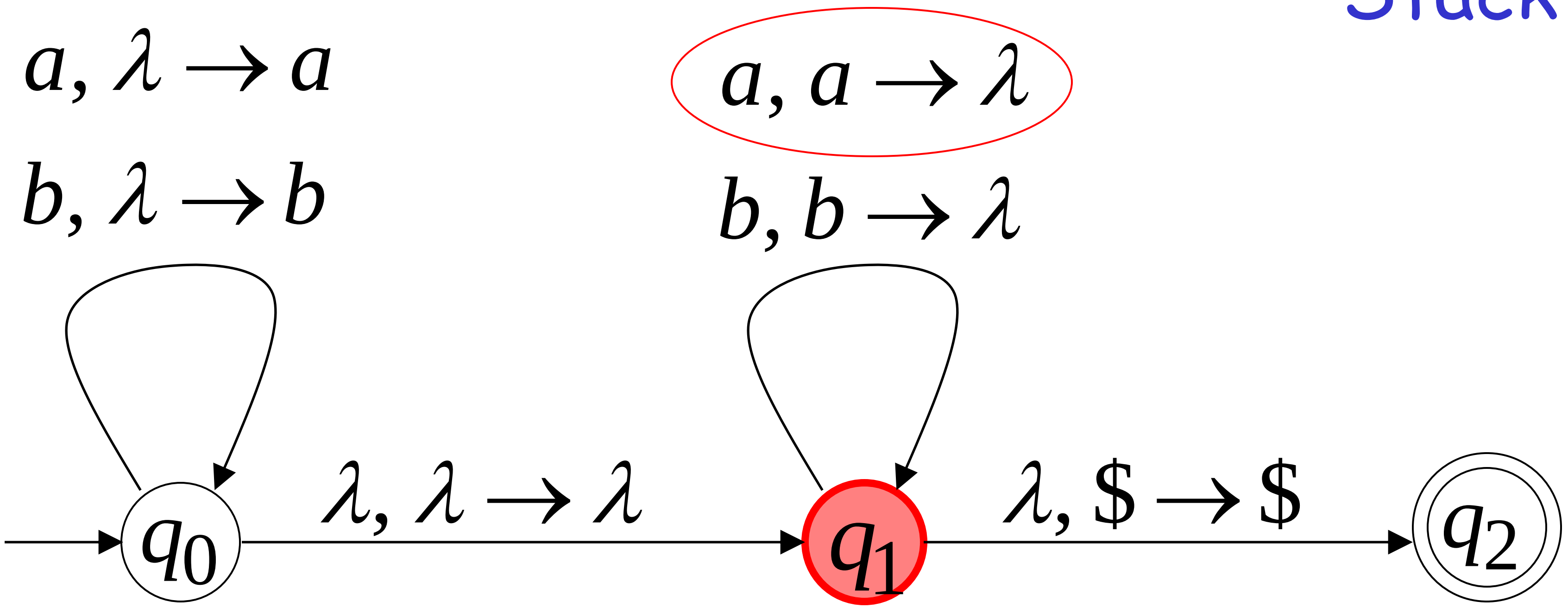
Time 4


Input
a
b
b
a
$\cancel{b}$
a
$
Stack
$a, \lambda \rightarrow a$
$b, \lambda \rightarrow b$
$a, a \rightarrow \lambda$
$b, b \rightarrow \lambda$
$\lambda, \lambda \rightarrow \lambda$
$\lambda, \$ \rightarrow \$$
q_0
q_1
q_2

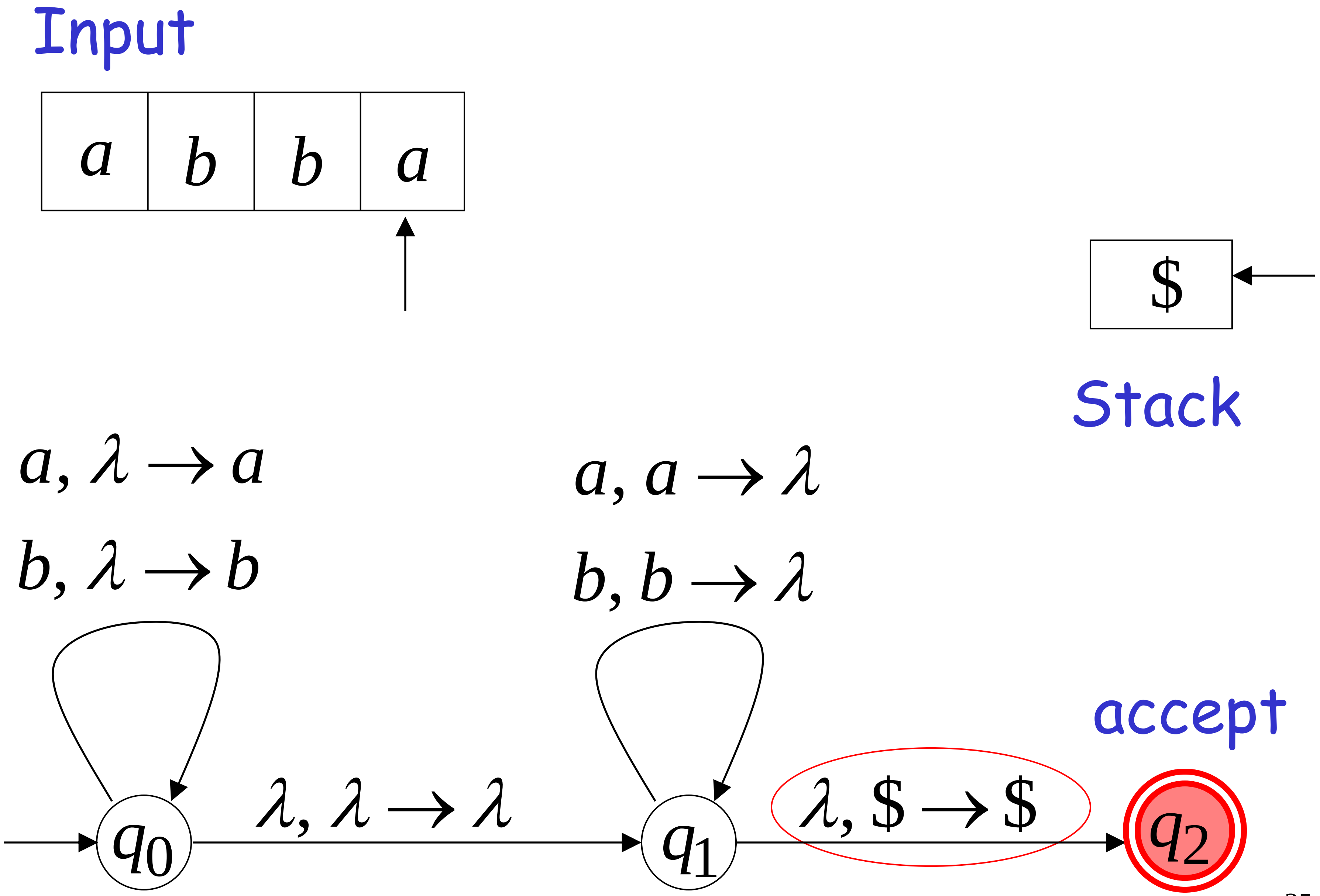
Time 5

Input

Stack



Time 6



Input

a	b	b	a

$\$$

Stack

$a, \lambda \rightarrow a$

$b, \lambda \rightarrow b$

$a, a \rightarrow \lambda$

$b, b \rightarrow \lambda$

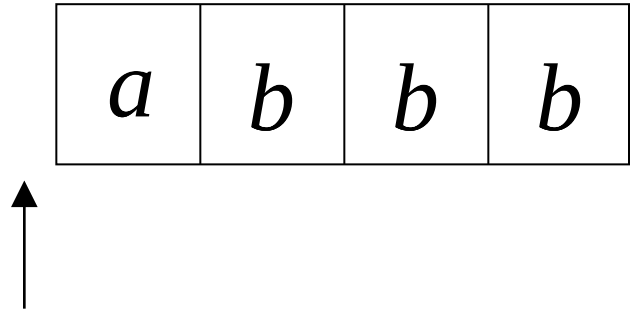
$\lambda, \lambda \rightarrow \lambda$

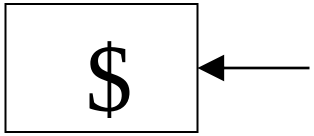
$\lambda, \$ \rightarrow \$$

accept

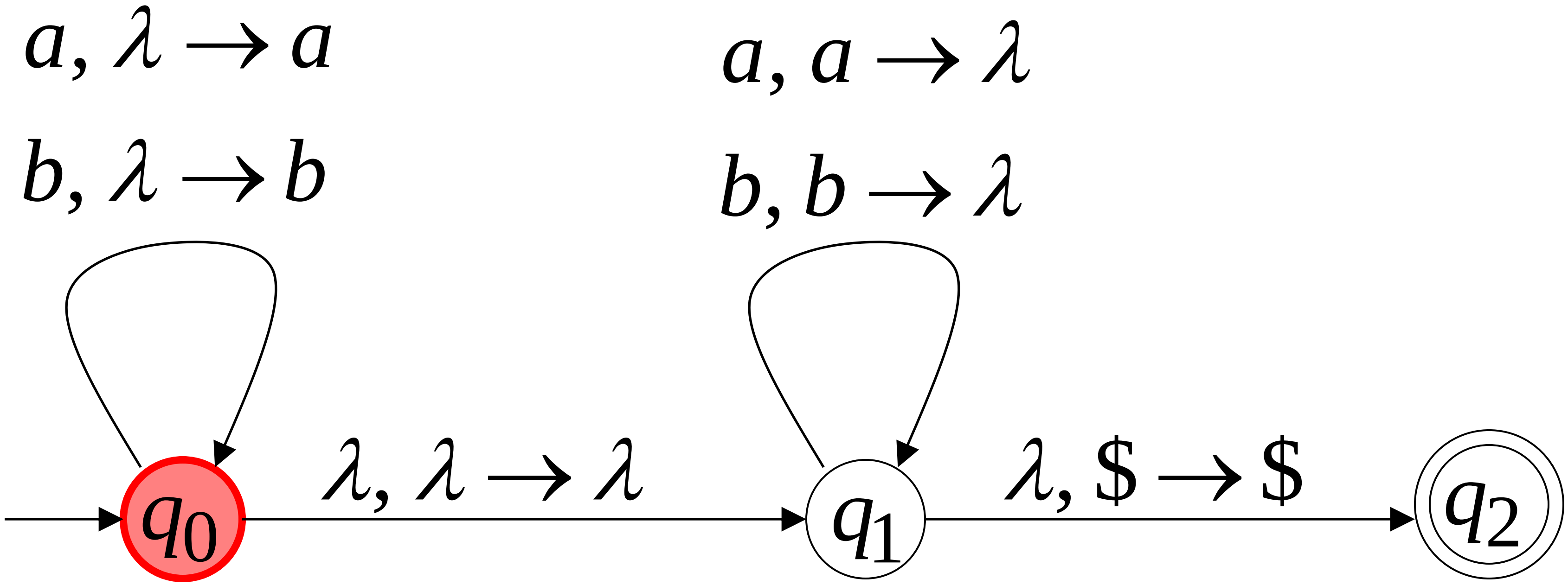
Rejection Example: Time 0

Input



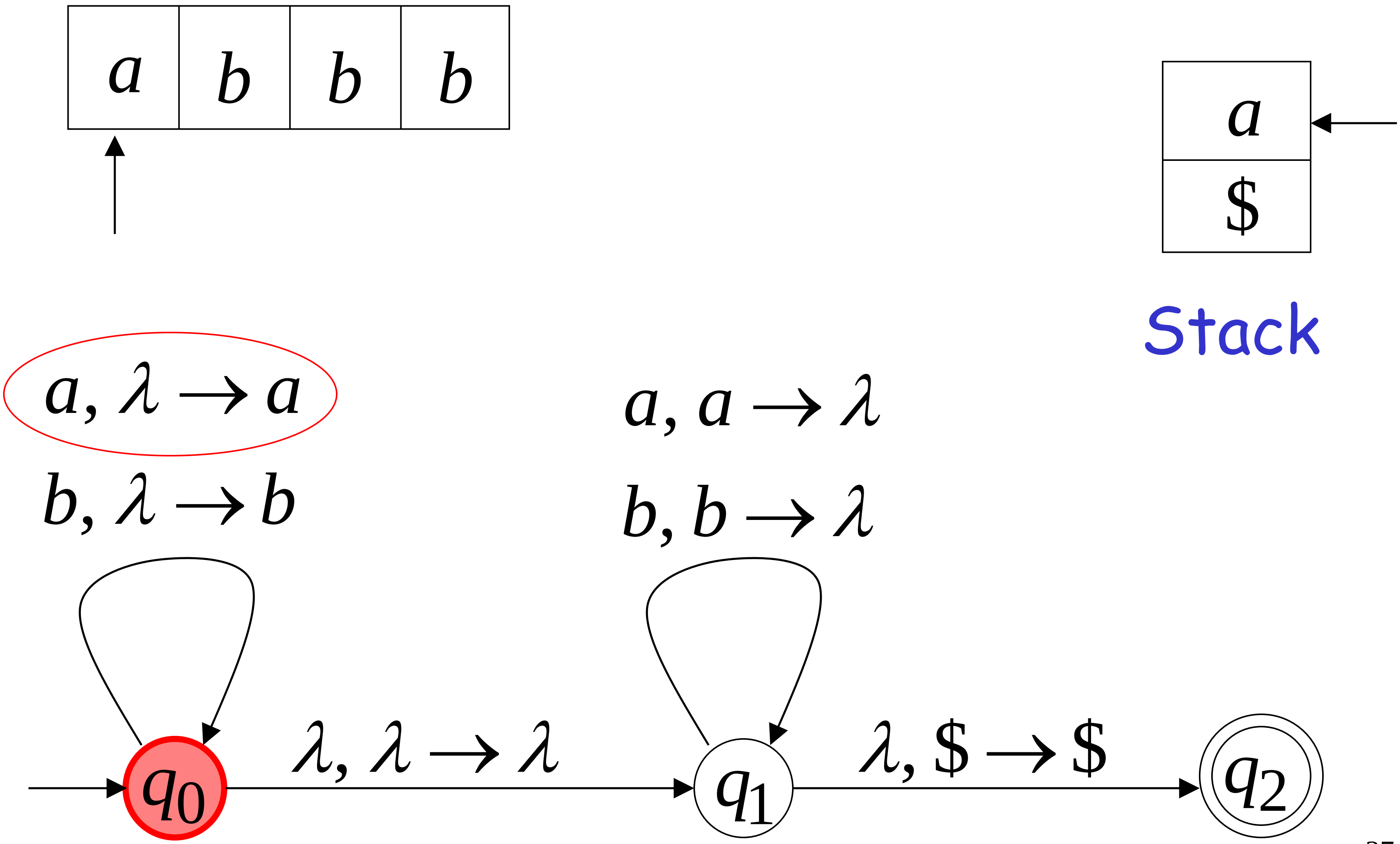


Stack

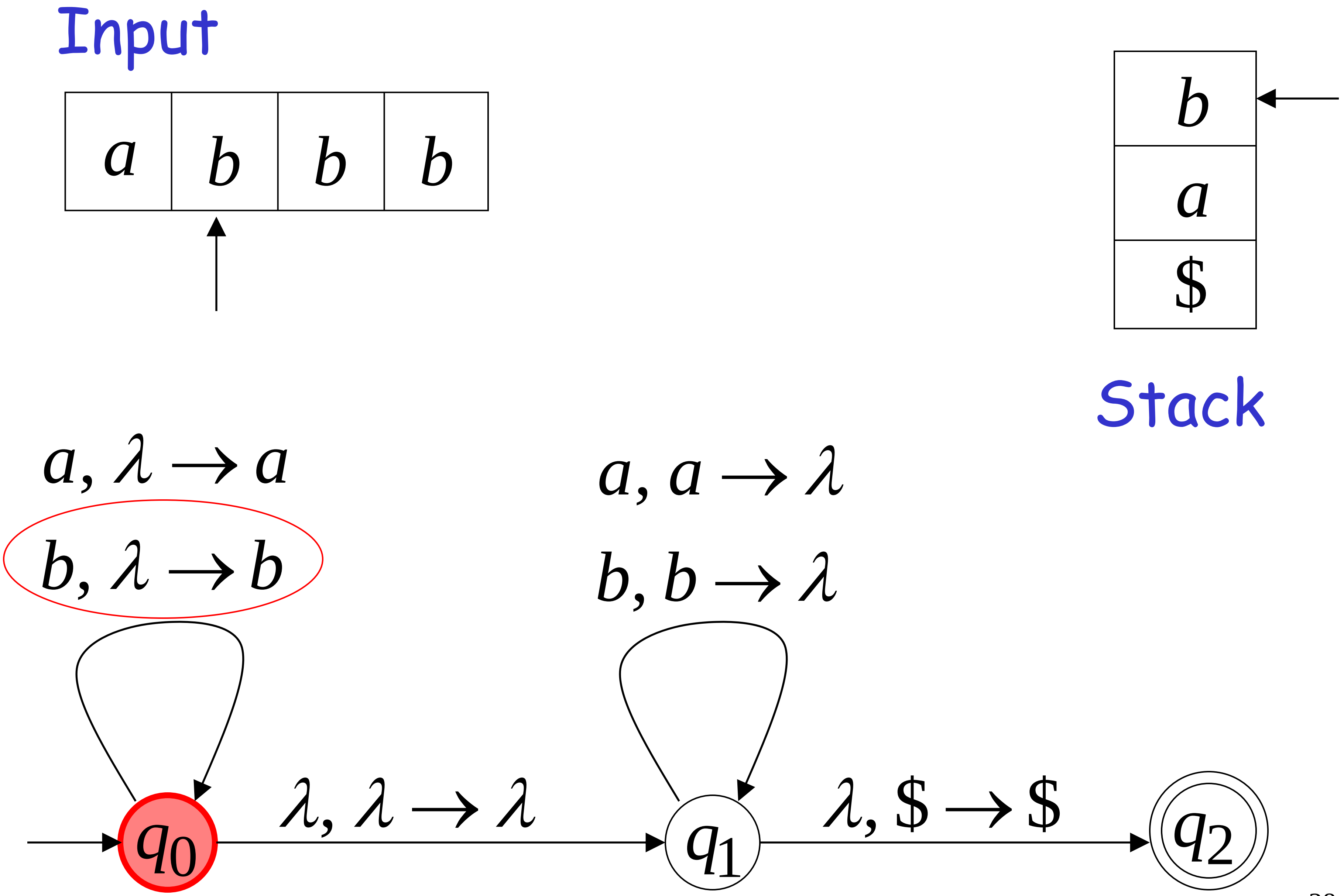


Time 1

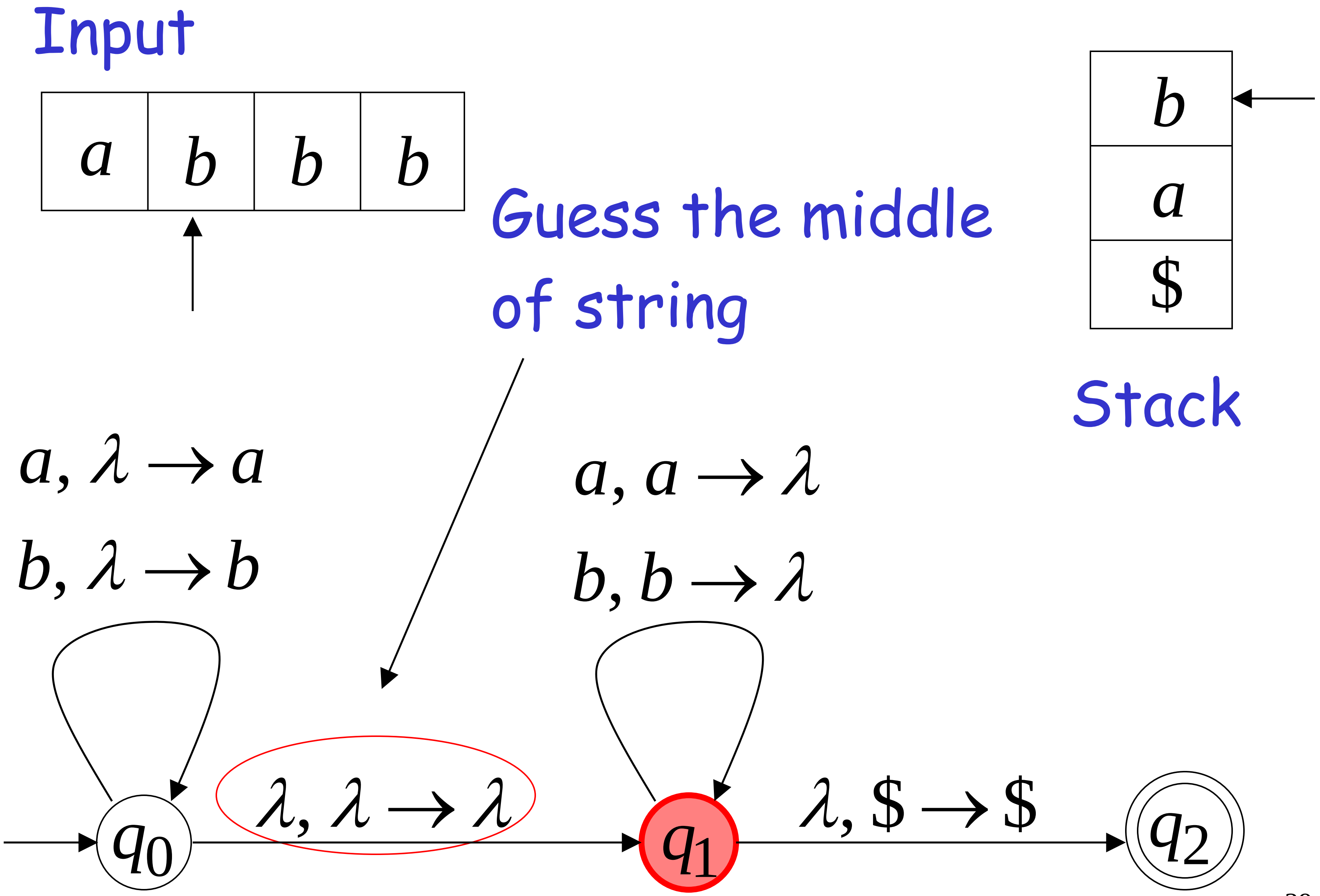
Input


a b b b
a
$
Stack
$a, \lambda \rightarrow a$
$b, \lambda \rightarrow b$
$a, a \rightarrow \lambda$
$b, b \rightarrow \lambda$
$\lambda, \lambda \rightarrow \lambda$
$\lambda, \$ \rightarrow \$$
q_0
q_1
q_2

Time 2

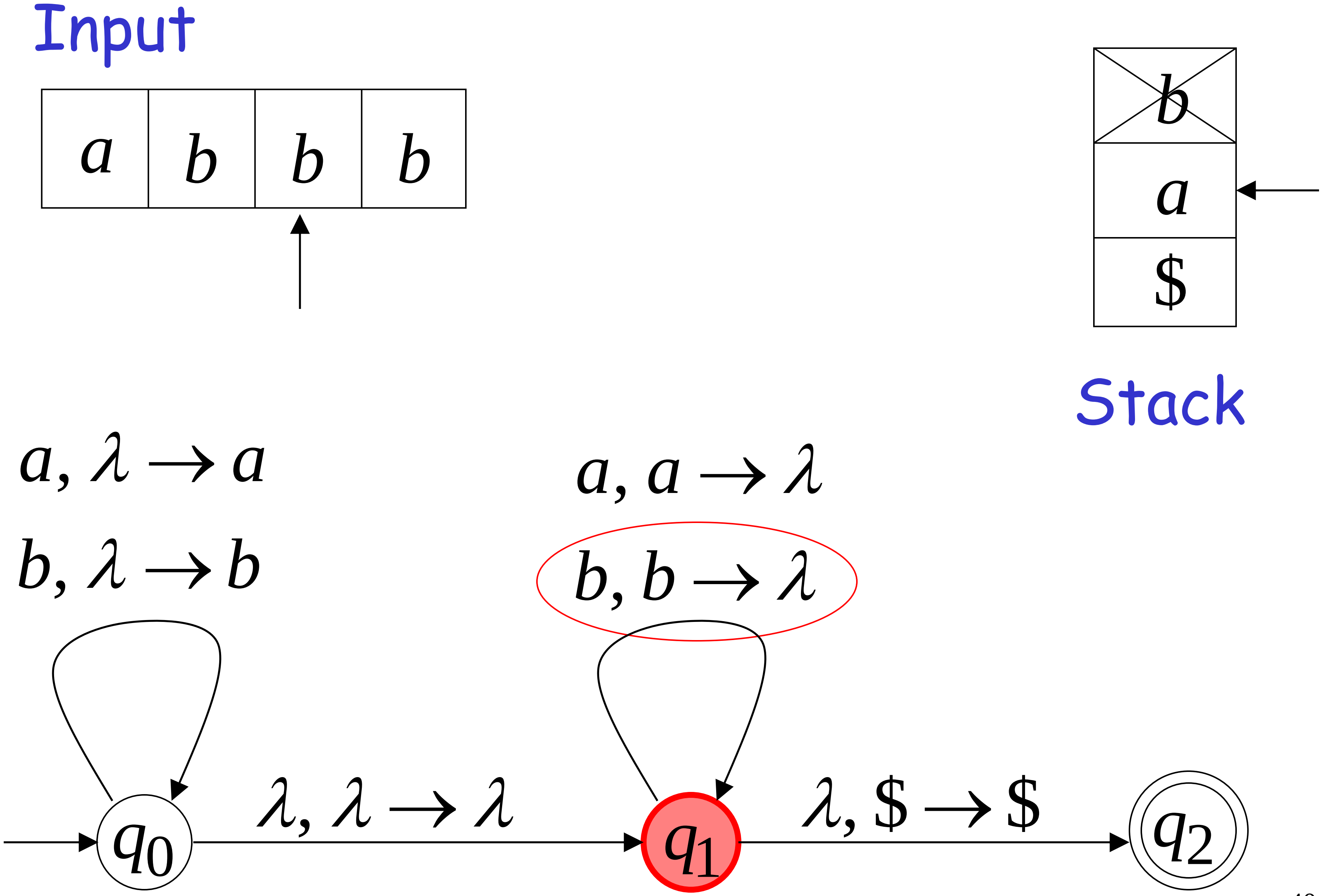

Input
a b b b
b
a
$\$$
Stack
$a, \lambda \rightarrow a$
$b, \lambda \rightarrow b$
$a, a \rightarrow \lambda$
$b, b \rightarrow \lambda$
$\lambda, \lambda \rightarrow \lambda$
$\lambda, \$ \rightarrow \$$
q_0
q_1
q_2

Time 3



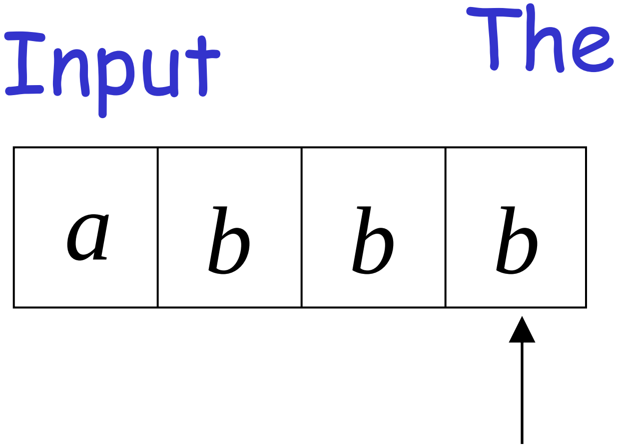
Guess the middle of string

Time 4

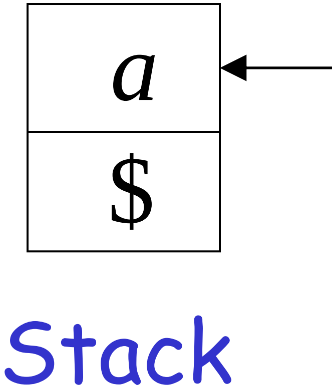

Input
a
b
b
b
$\cancel{b}$
a
$\$$
Stack
$a, \lambda \rightarrow a$
$b, \lambda \rightarrow b$
$a, a \rightarrow \lambda$
$b, b \rightarrow \lambda$
$\lambda, \lambda \rightarrow \lambda$
$\lambda, \$ \rightarrow \$$
q_0
q_1
q_2

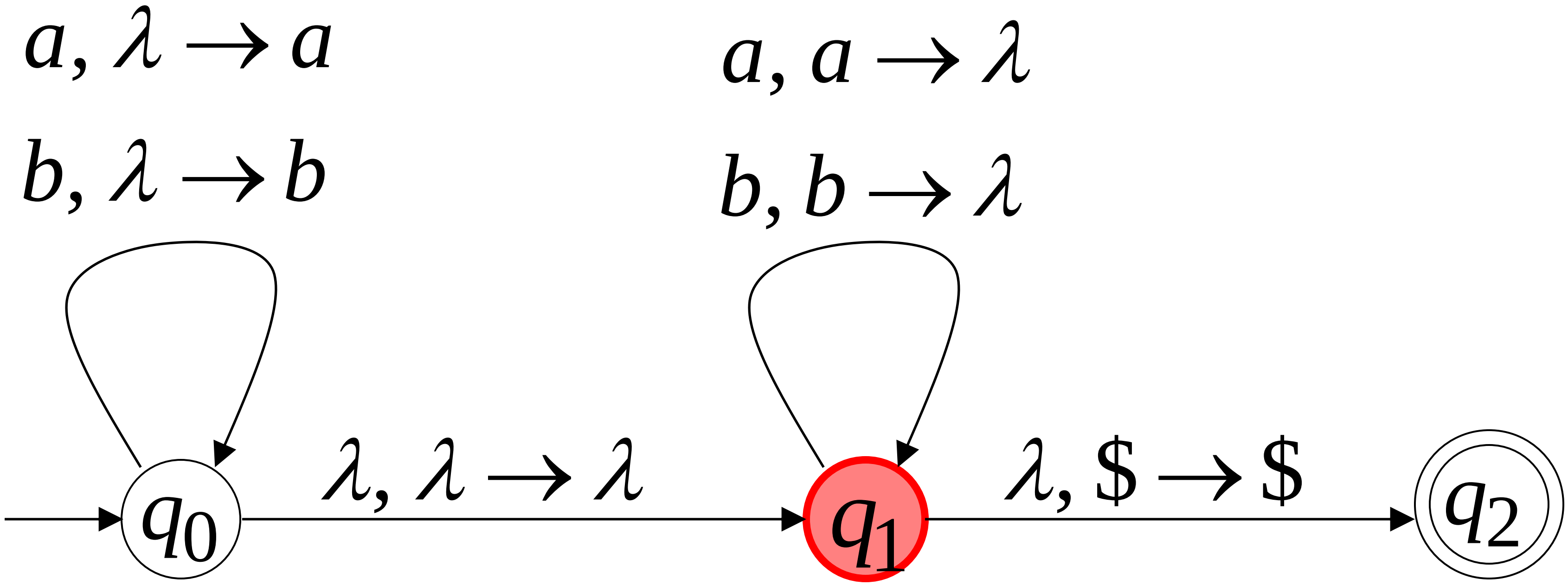
Time 5

There is no possible transition.

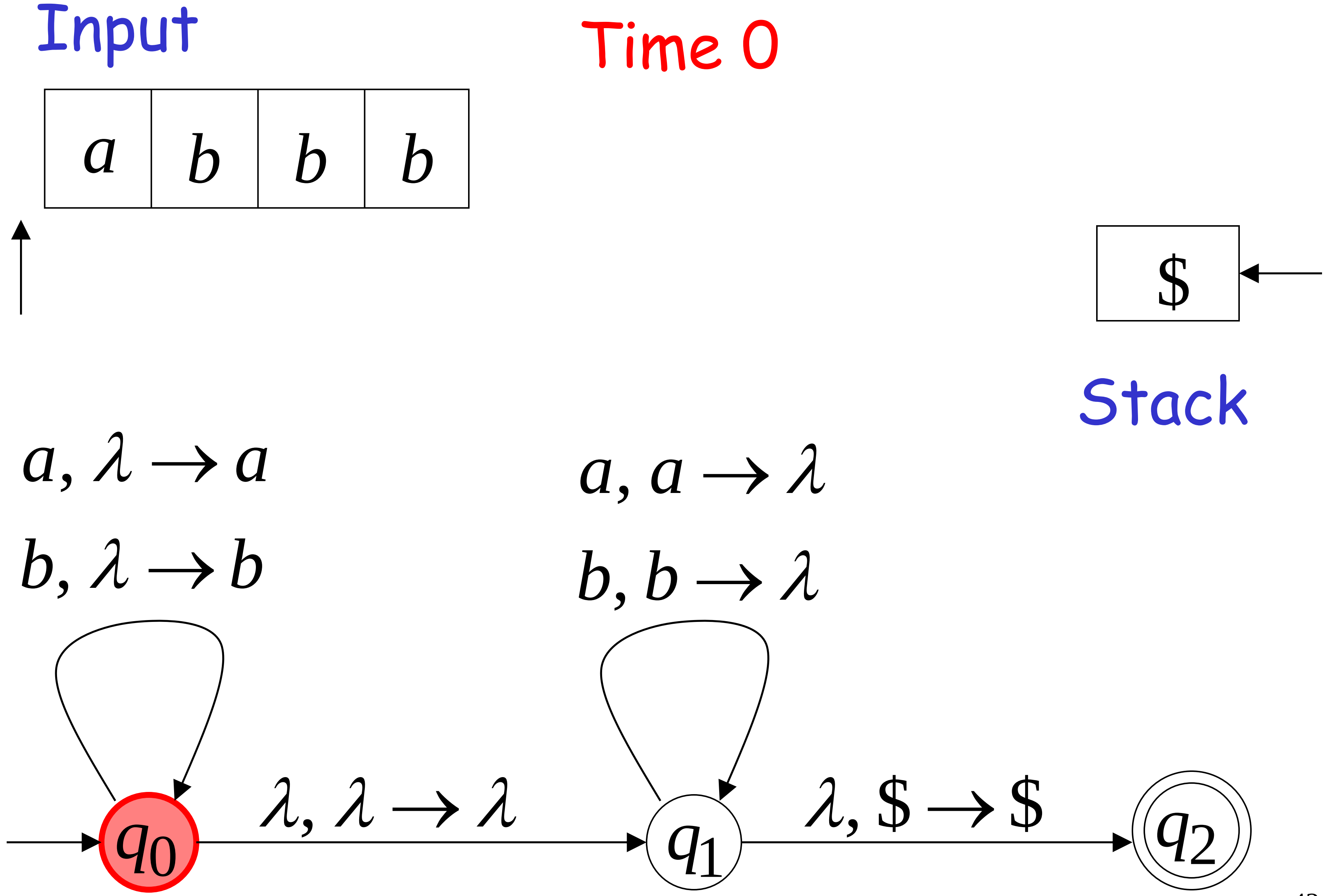


Input is not consumed

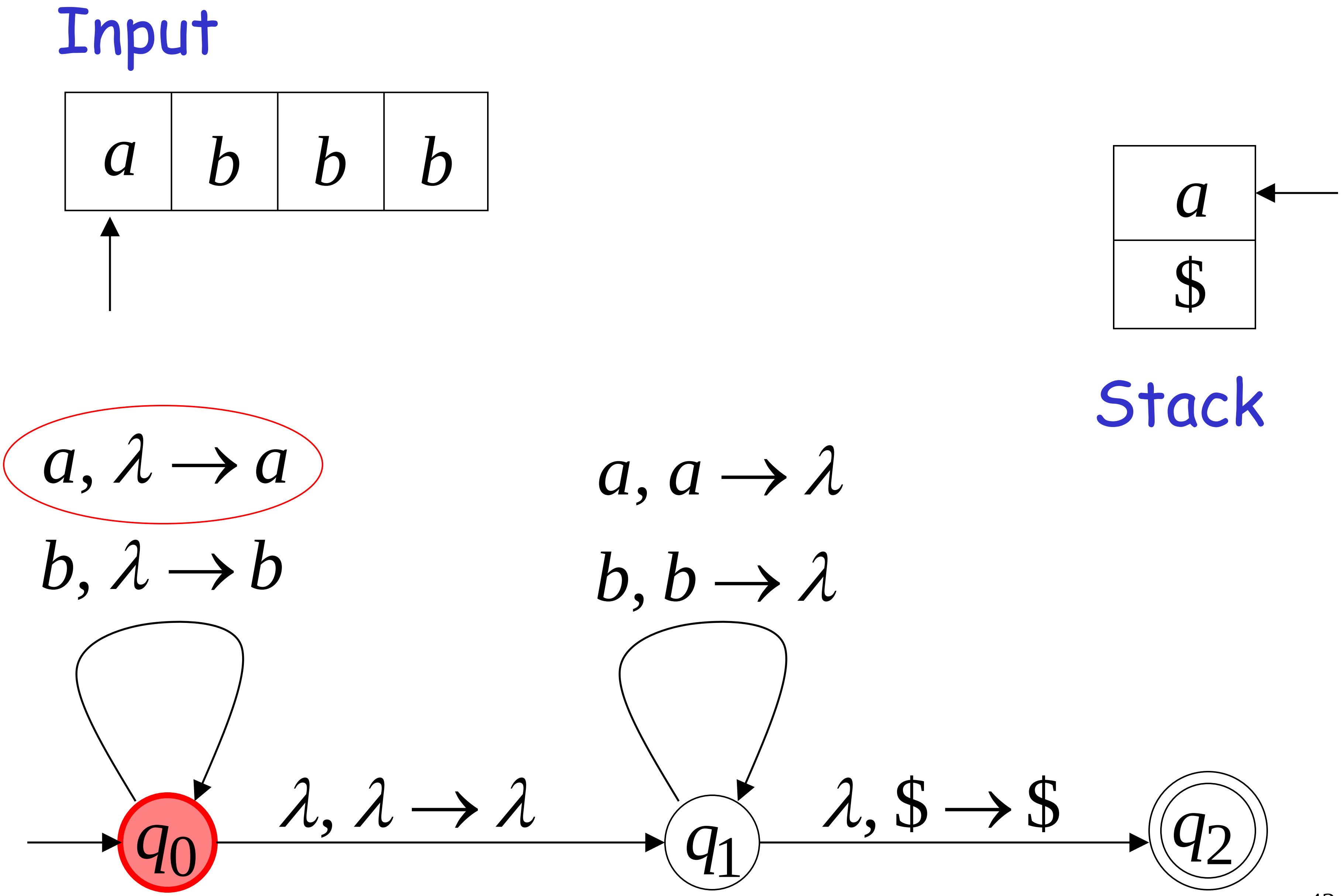




Another computation on same string:

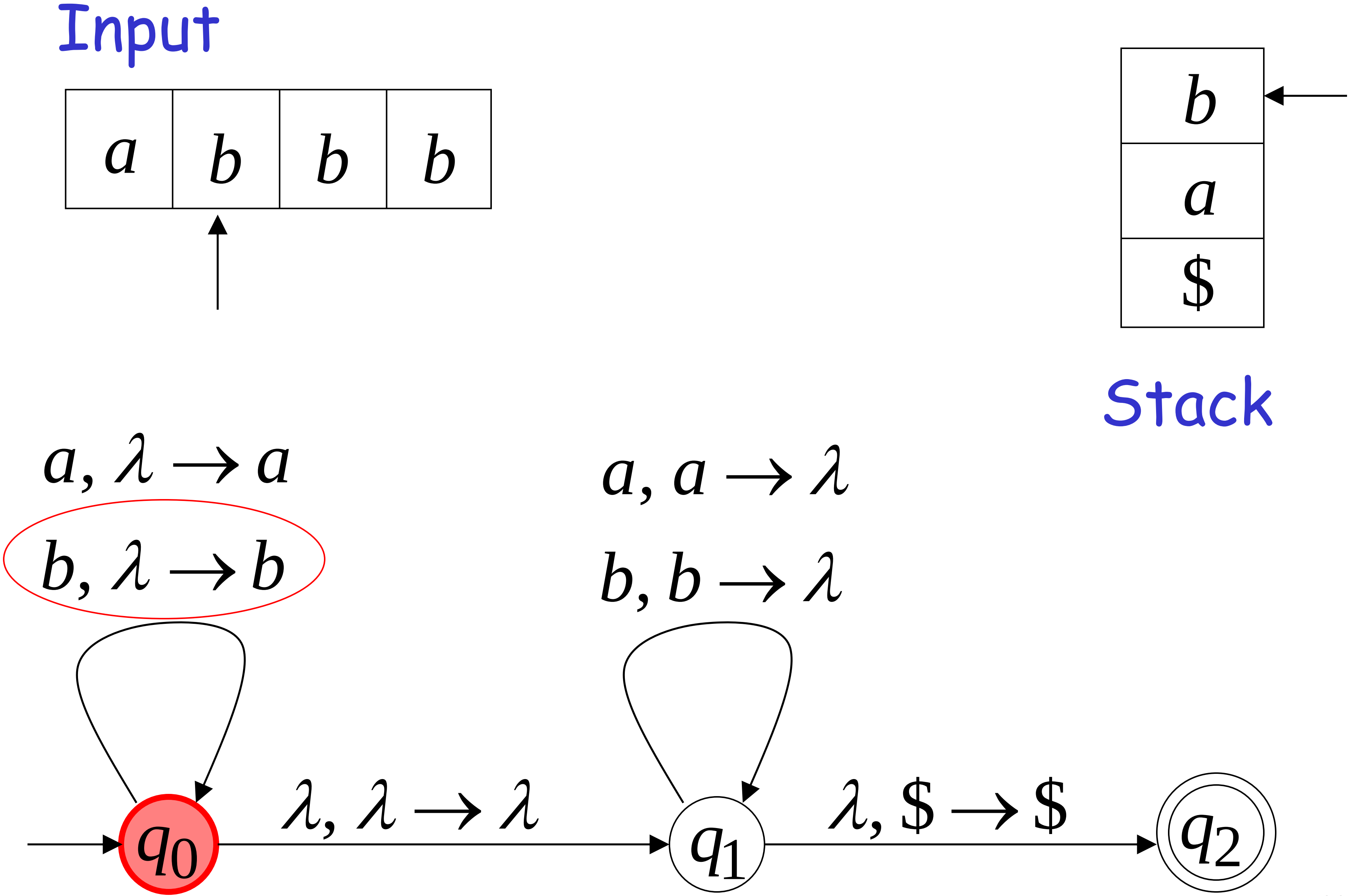


Time 1

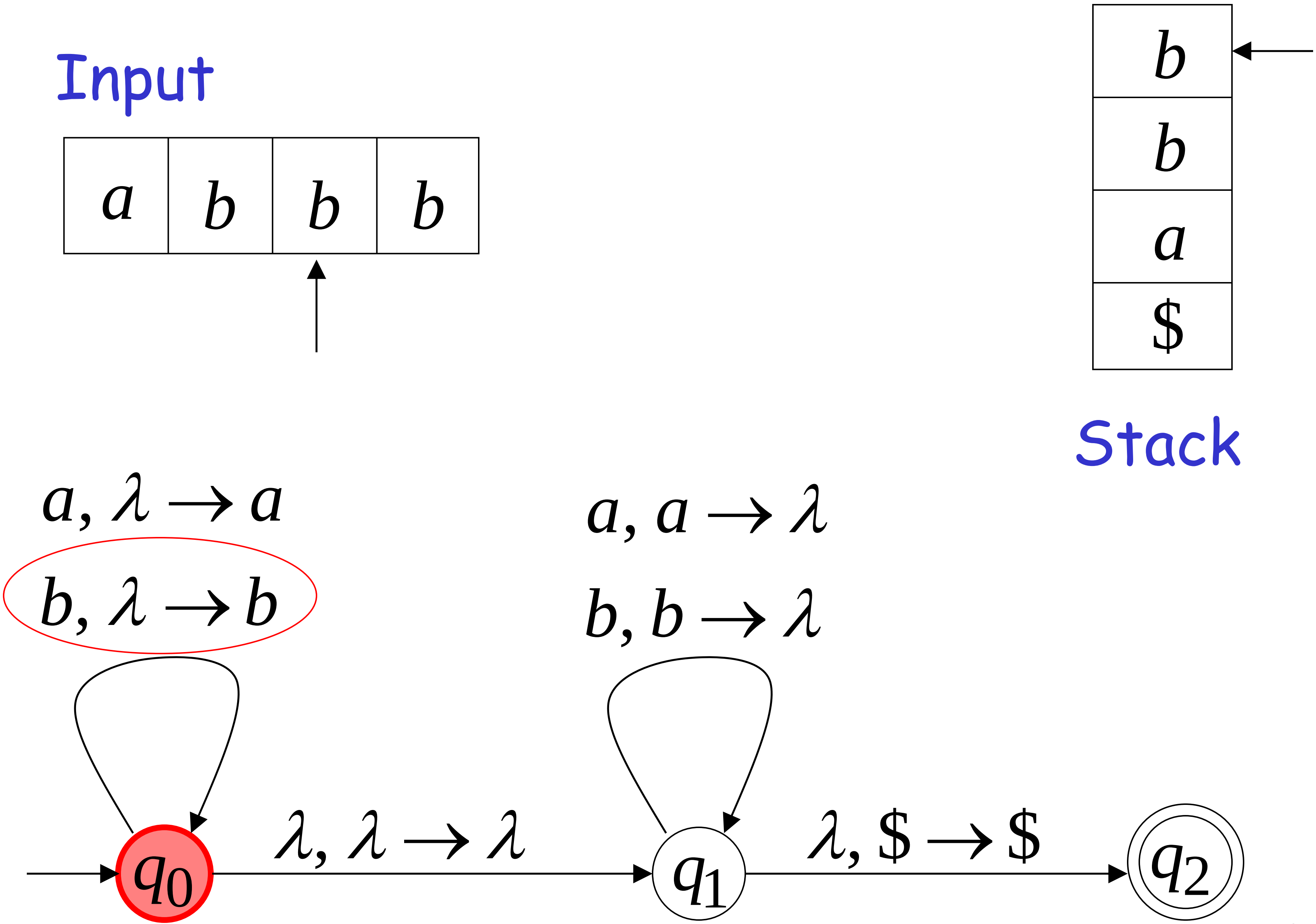


Input
a b b b
a
$\$$
Stack
$a, \lambda \rightarrow a$
$b, \lambda \rightarrow b$
$a, a \rightarrow \lambda$
$b, b \rightarrow \lambda$
$\lambda, \lambda \rightarrow \lambda$
$\lambda, \$ \rightarrow \$$
q_0
q_1
q_2

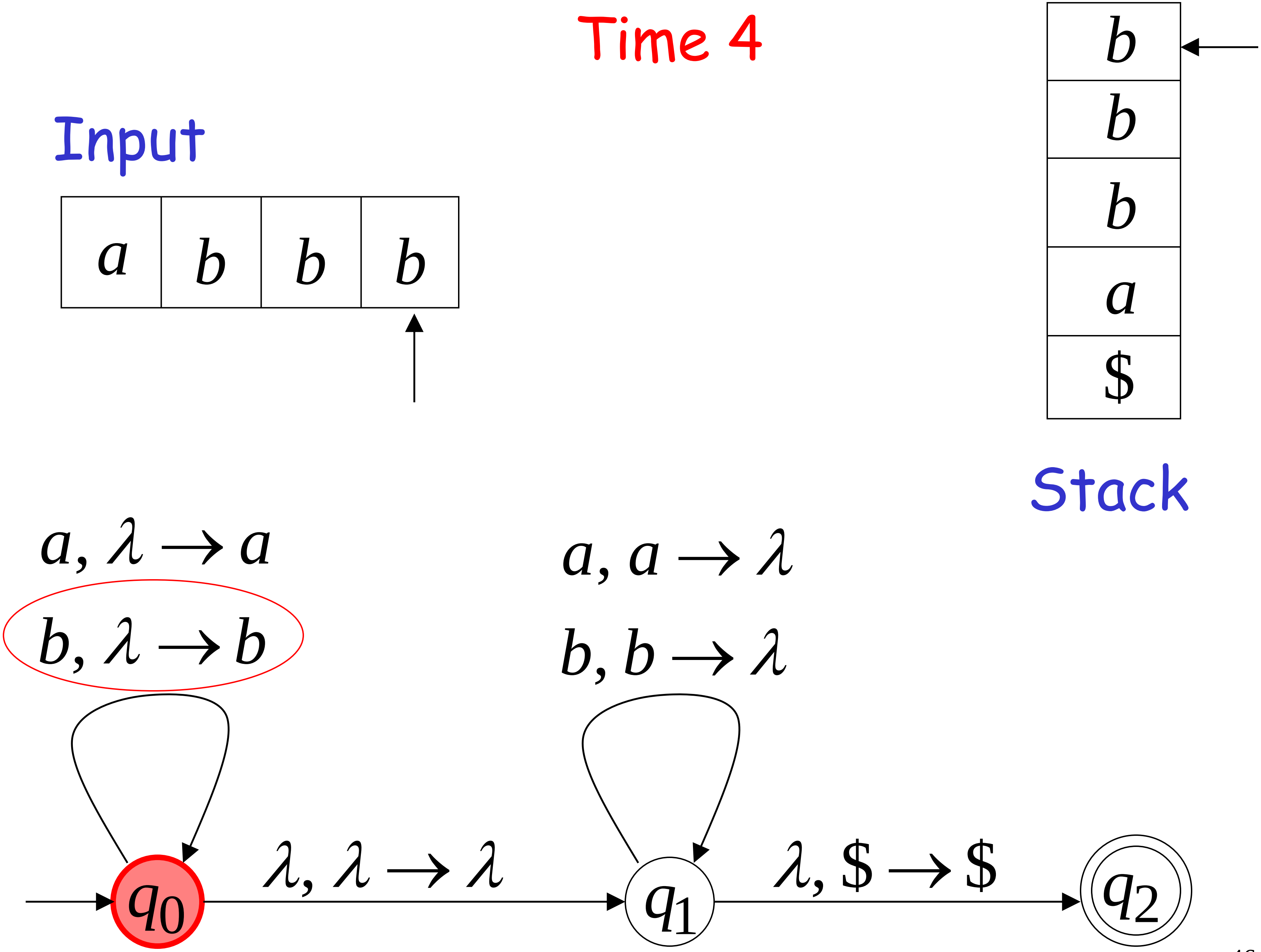
Time 2


Input
a b b b
b
a
$\$$
Stack
$a, \lambda \to a$
$b, \lambda \to b$
$a, a \to \lambda$
$b, b \to \lambda$
$\lambda, \lambda \to \lambda$
$\lambda, \$ \to \$$
q_0
q_1
q_2

Time 3

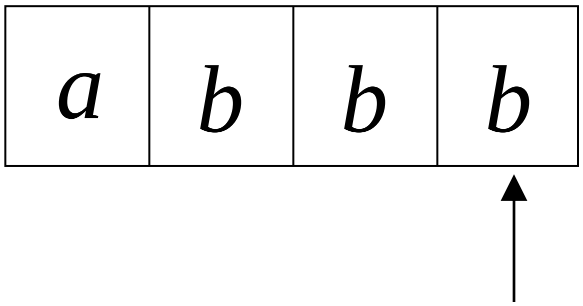

Input
a b b b
b
b
a
$
Stack
$a, \lambda \rightarrow a$
$b, \lambda \rightarrow b$
$a, a \rightarrow \lambda$
$b, b \rightarrow \lambda$
$\lambda, \lambda \rightarrow \lambda$
$\lambda, \$ \rightarrow \$$
q_0
q_1
q_2

Time 4

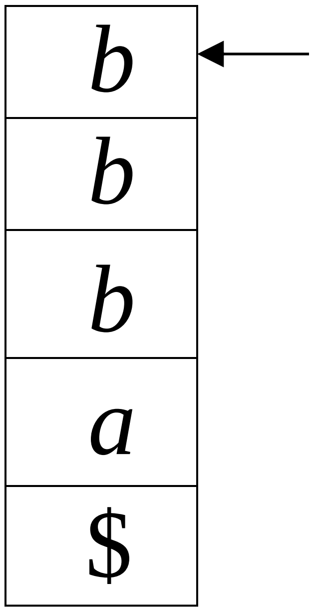

Input
a b b b
b
b
b
a
$\$$
Stack
$a, \lambda \rightarrow a$
$b, \lambda \rightarrow b$
$a, a \rightarrow \lambda$
$b, b \rightarrow \lambda$
$\lambda, \lambda \rightarrow \lambda$
$\lambda, \$ \rightarrow \$$
q_0
q_1
q_2

Time 5

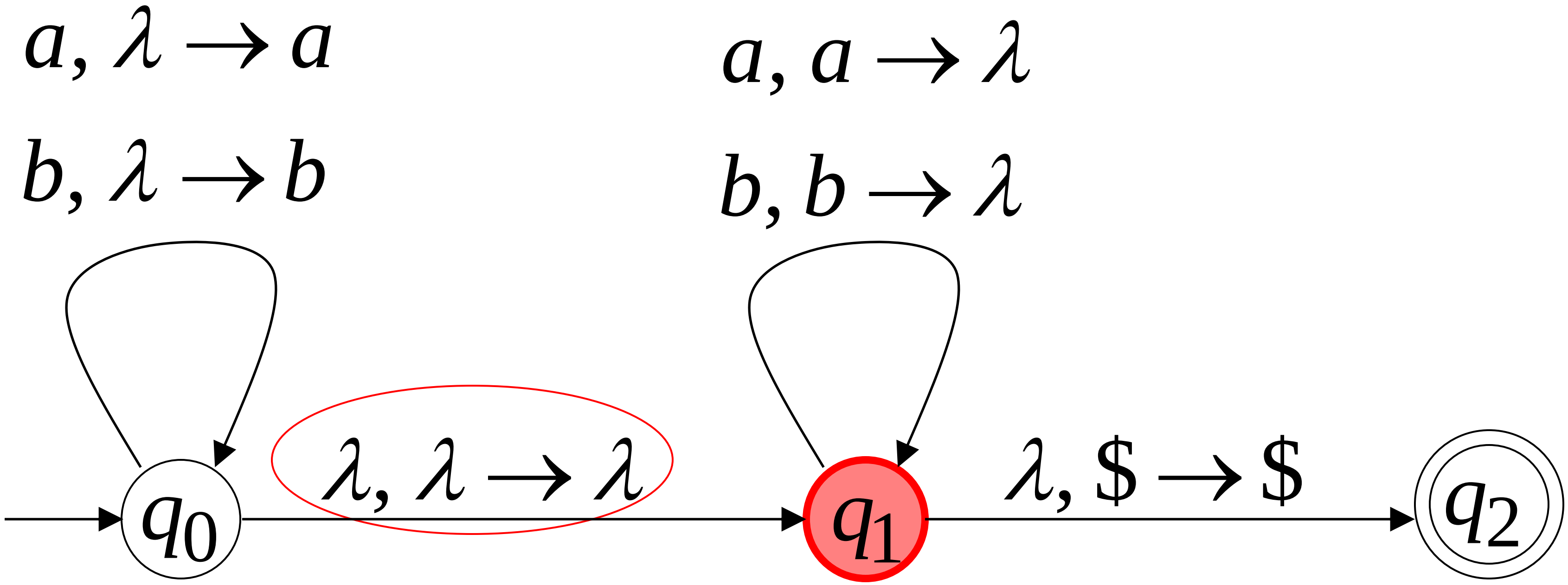
Input



No final state is reached

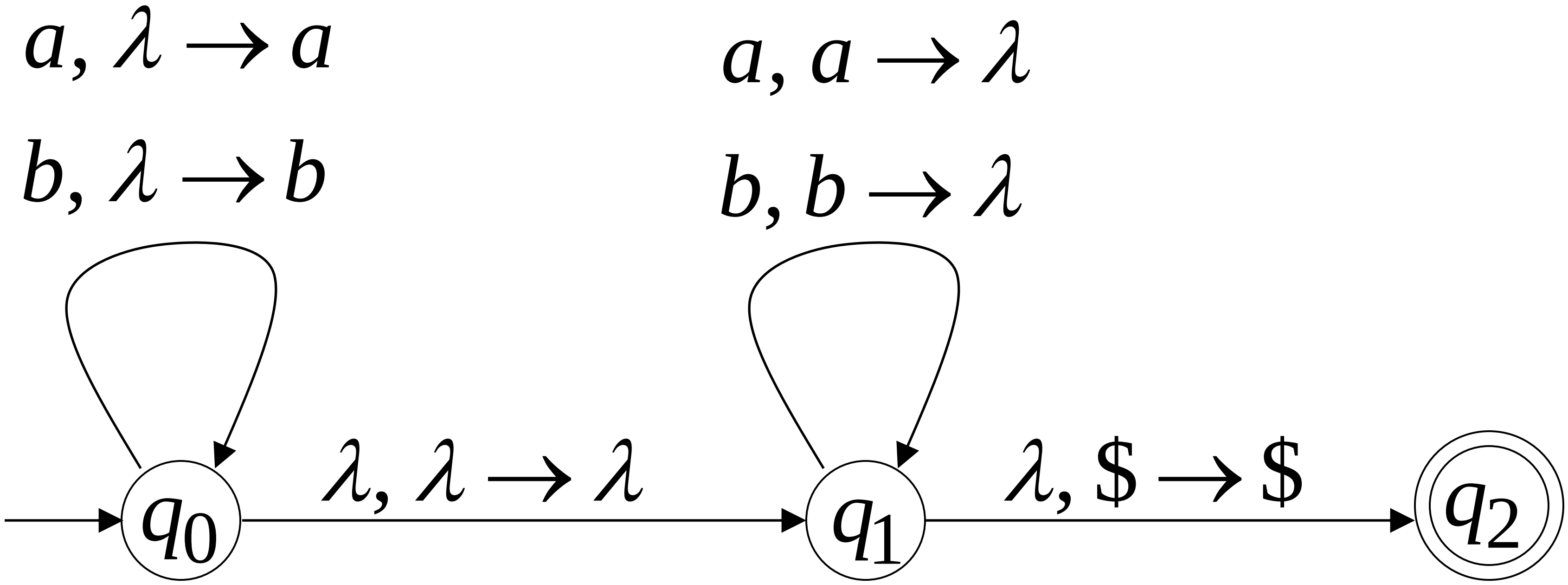


Stack



There is no computation that accepts string $abbb$

$$abbb \notin L(M)$$



A string is rejected if there is no computation such that:

All the input is consumed

AND

The last state is a final state

At the end of the computation, we do not care about the stack contents

In other words, a string is rejected if in every computation with this string:

The input cannot be consumed

OR

The input is consumed and the last state is not a final state

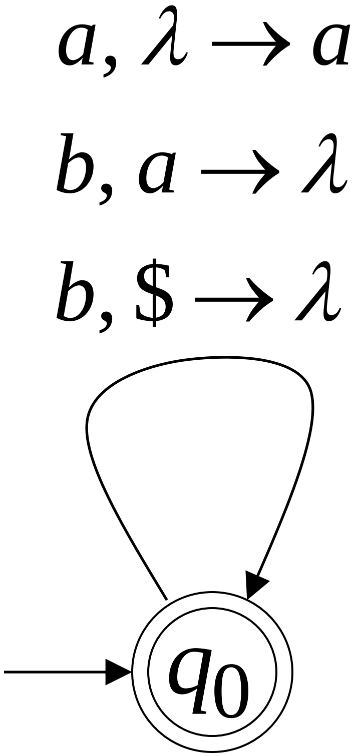
OR

The stack head moves below the bottom of the stack

Another NPDA example

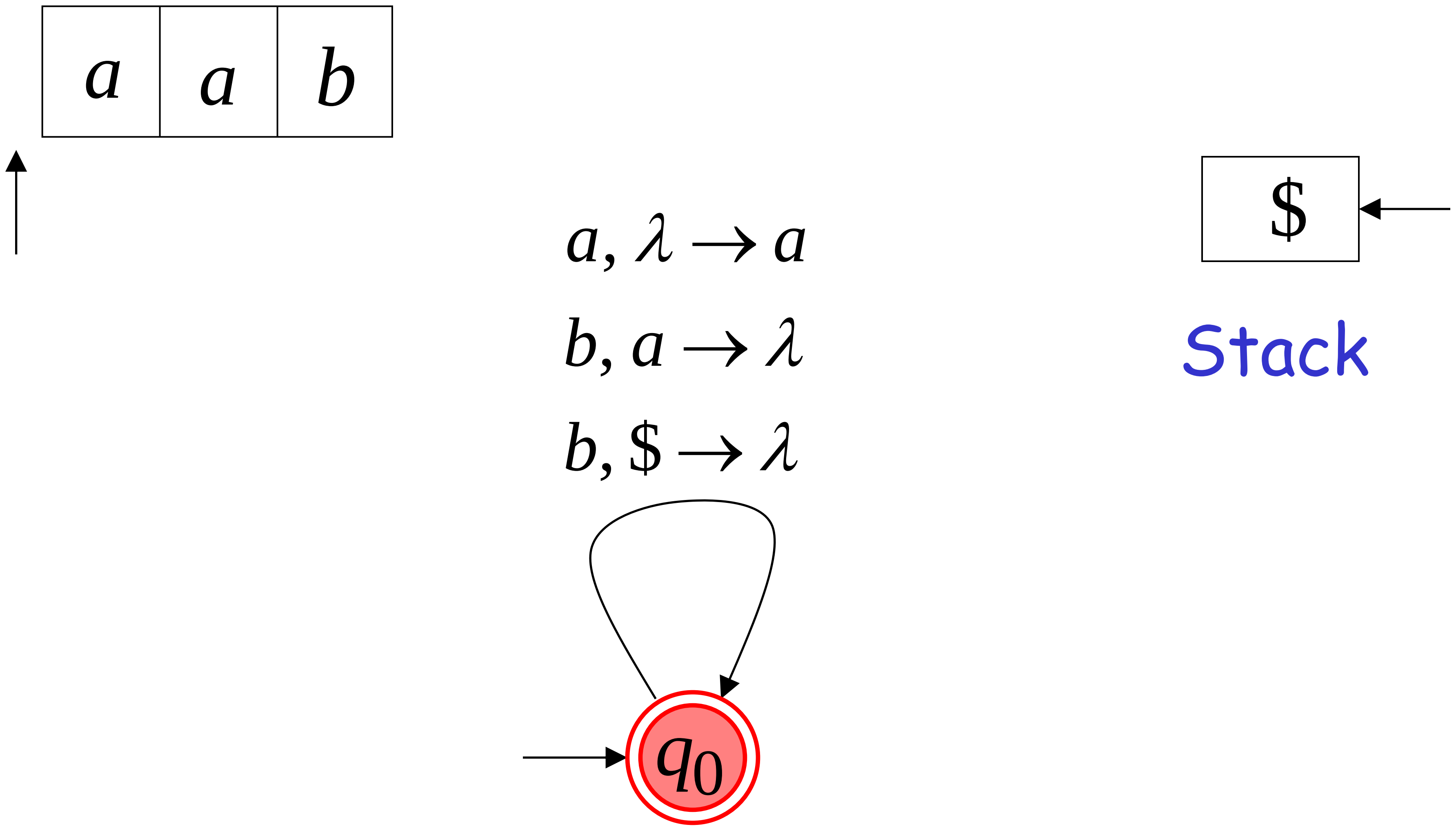
NPDA M

$$L(M) = \{w: \; w \in \{a,b\}^*, n_a \geq n_b \;\; for\ any\ prefix\ of\ \; w\}$$



Execution Example: Time 0

Input



$a, \lambda \to a$

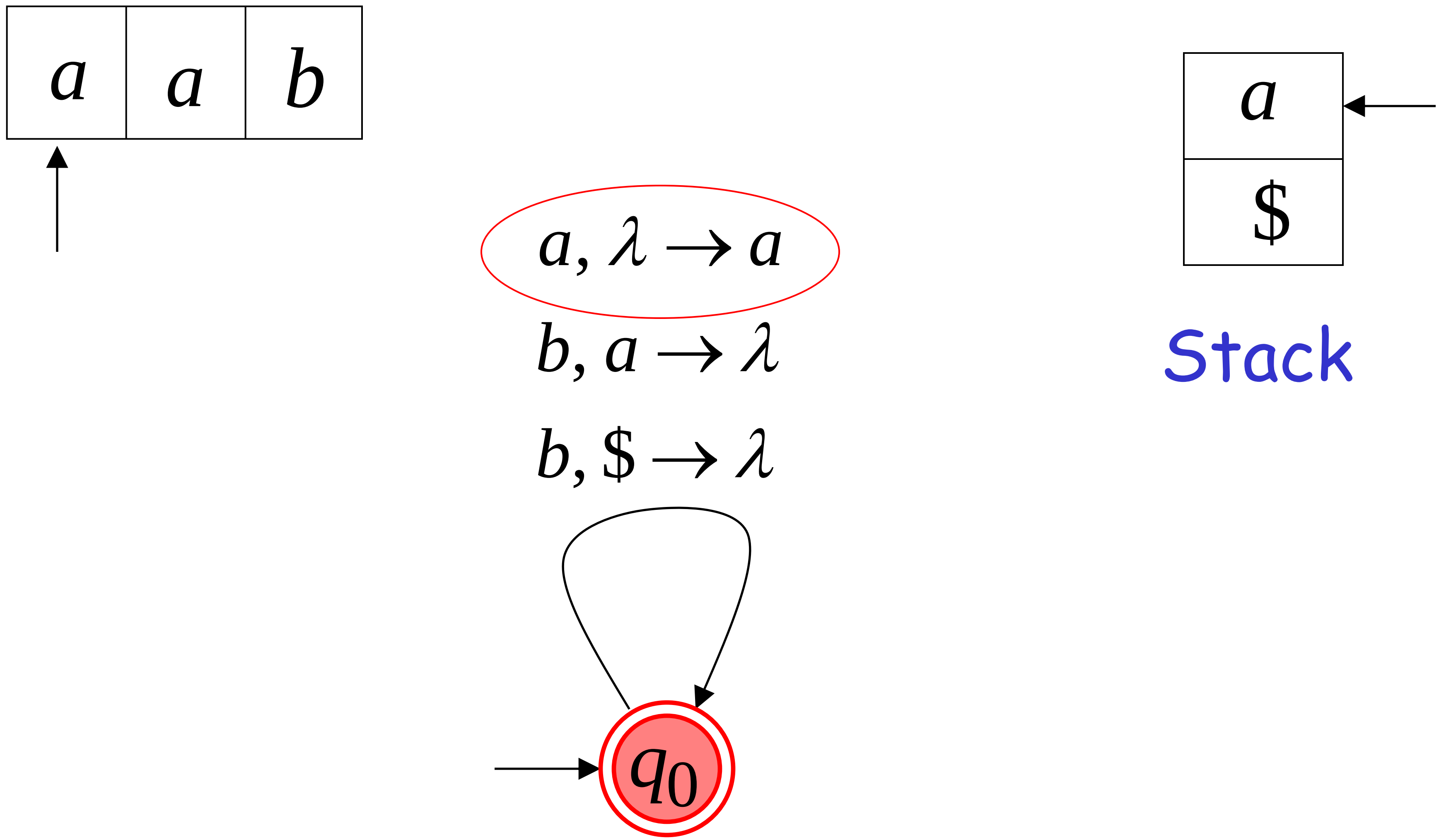
$b, a \to \lambda$

$b, \$ \to \lambda$

Stack

Time 1

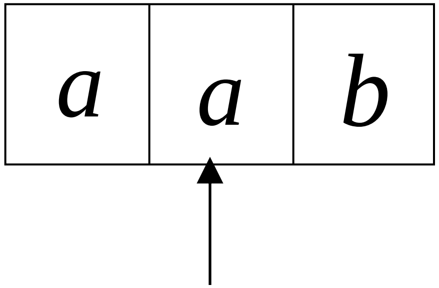
Input



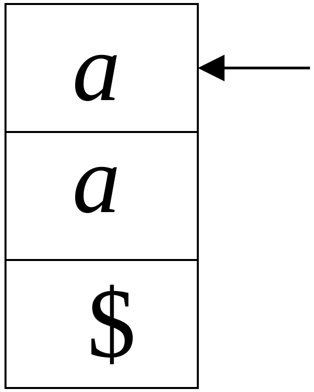
Stack

Time 2

Input

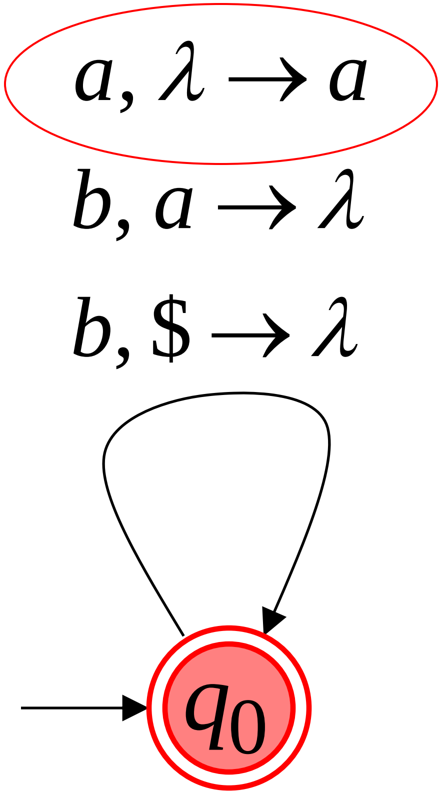


a	a	b



a
a
$\$$

Stack

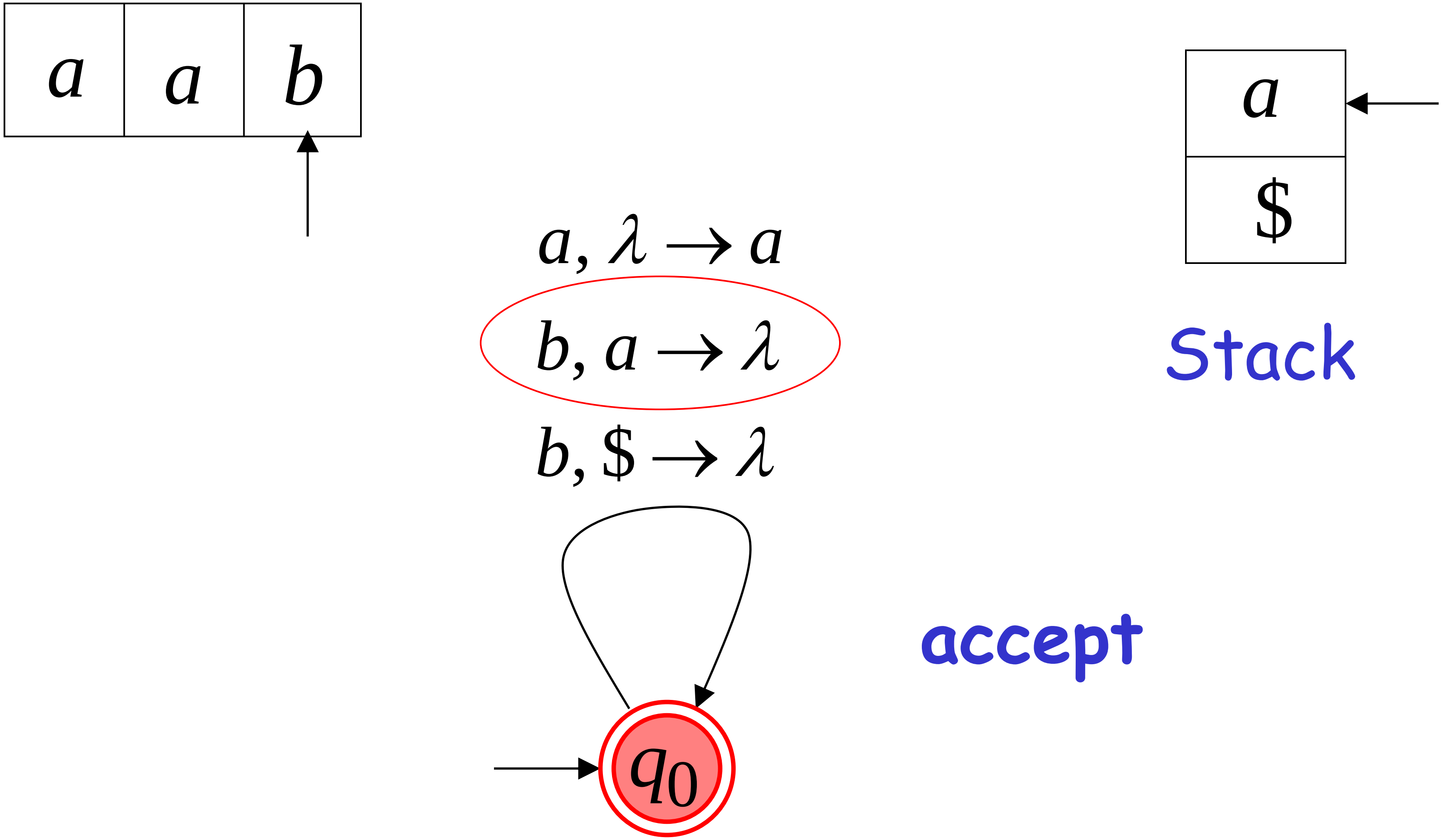


$a, \lambda \rightarrow a$

$b, a \rightarrow \lambda$

$b, \$ \rightarrow \lambda$

Time 3



Input

a	a	b

$a, \lambda \rightarrow a$

$b, a \rightarrow \lambda$

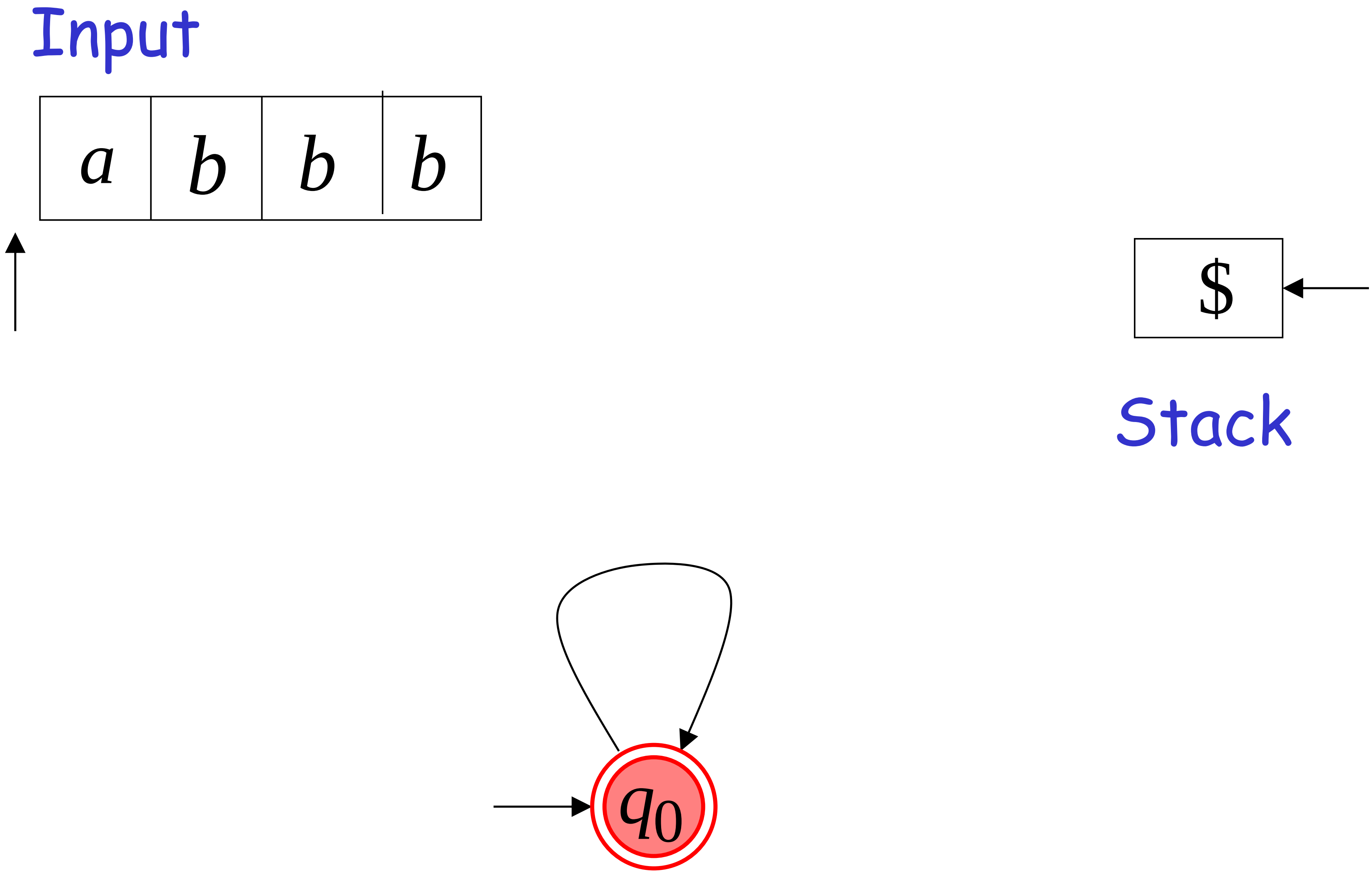
$b, \$ \rightarrow \lambda$

a
$\$$

Stack

accept

Rejection example: Time 0



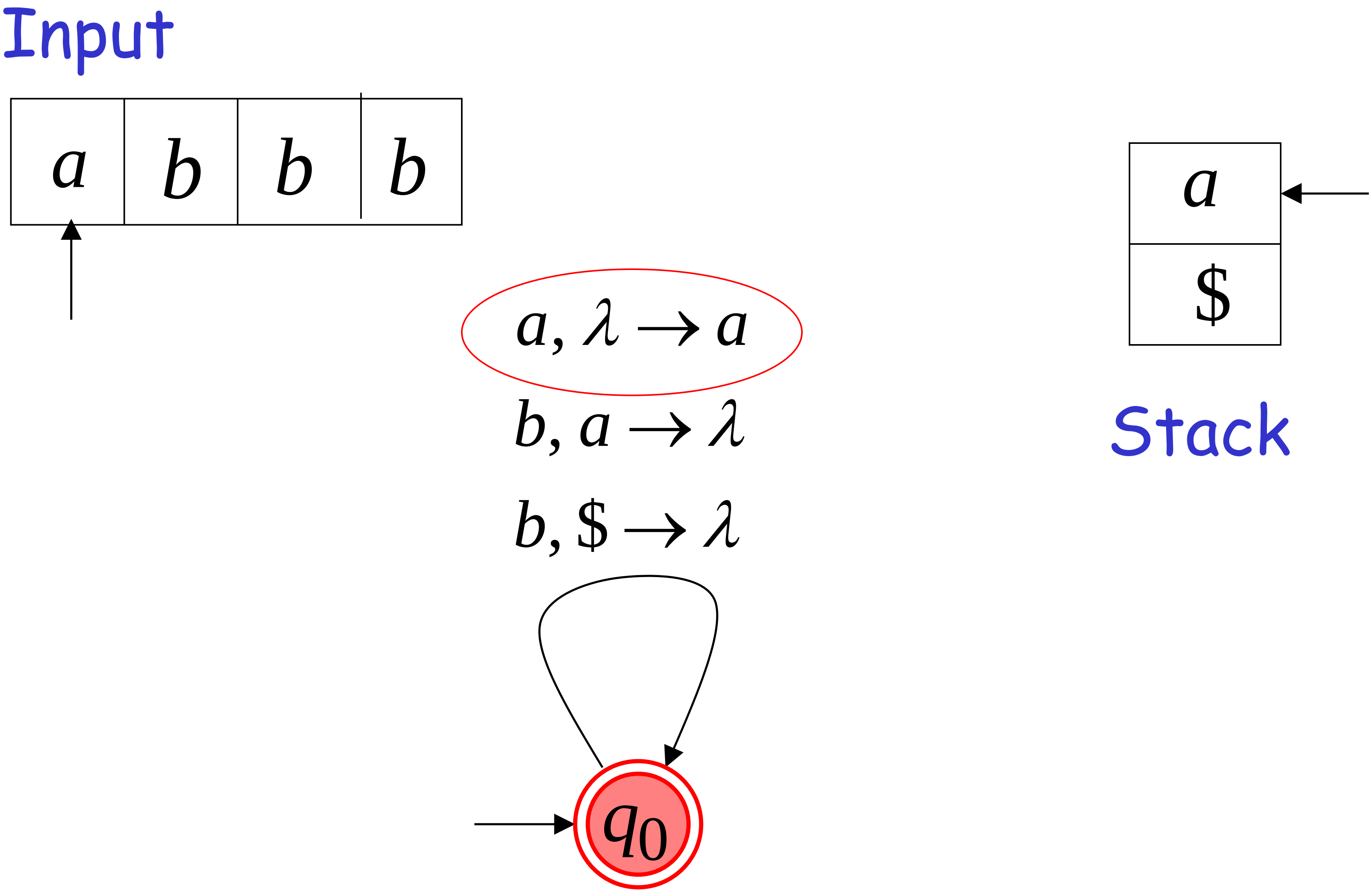
Input

a	b	b	b

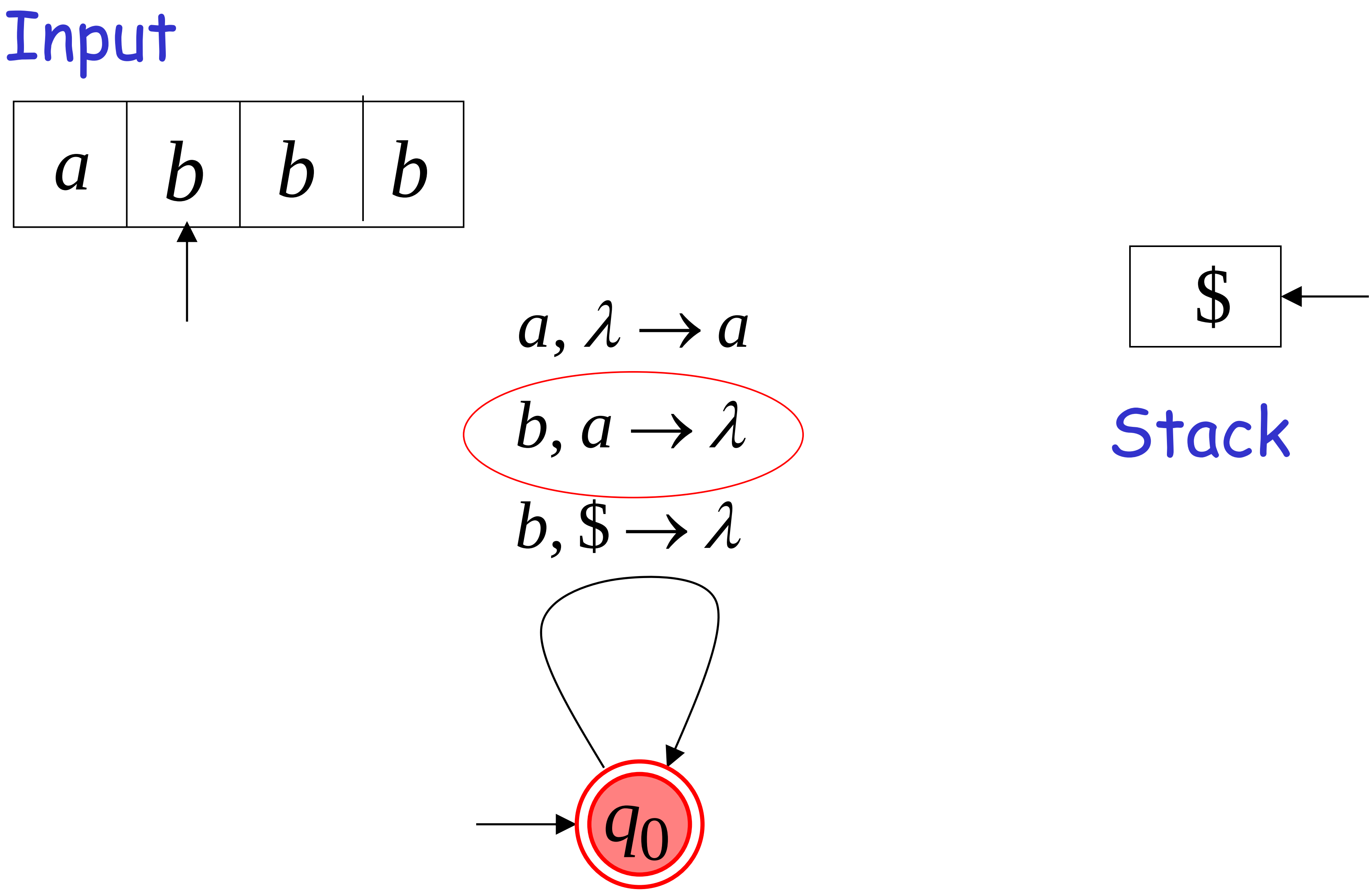
$\$$

Stack

Time 1


Input
a
b
b
b
$a, \lambda \rightarrow a$
$b, a \rightarrow \lambda$
$b, \$ \rightarrow \lambda$
q_0
a
$
Stack

Time 2



Input

a	b	b	b

$a, \lambda \to a$

$b, a \to \lambda$

$b, \$ \to \lambda$

q_0

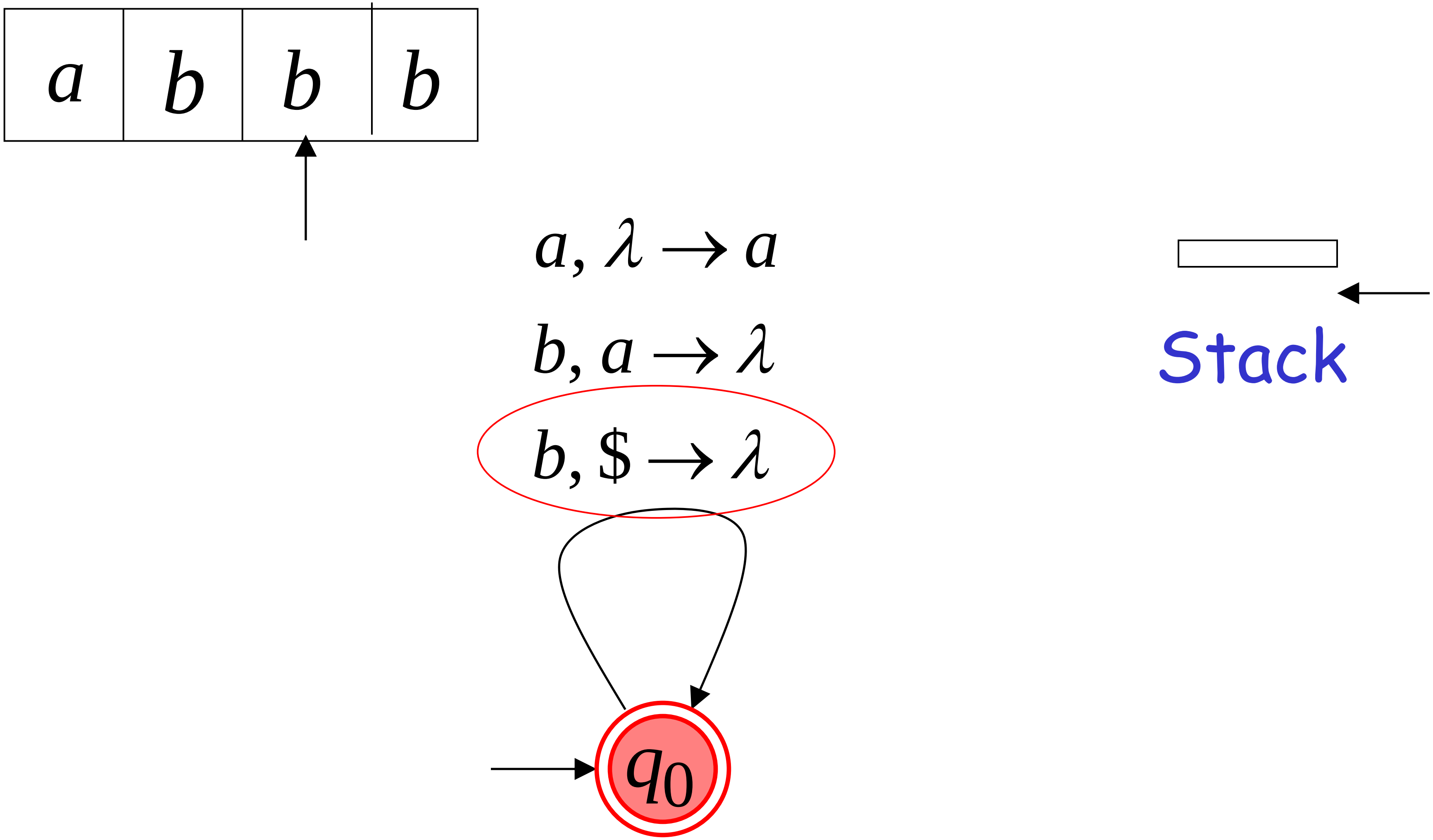
$\$$

Stack

Time 3

Input

Stack



$a, \lambda \rightarrow a$

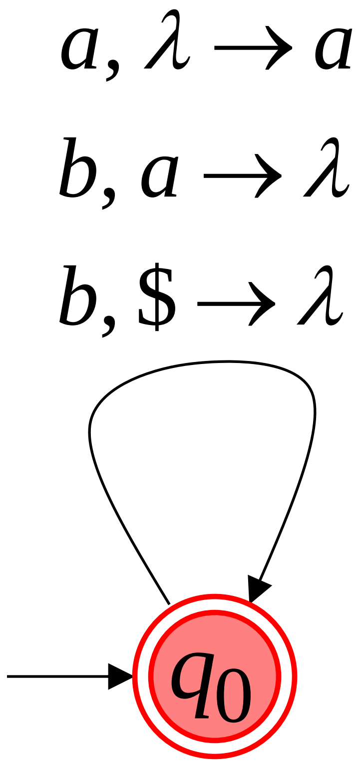
$b, a \rightarrow \lambda$

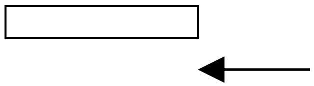
$b, \$ \rightarrow \lambda$

Time 4

Input

a	b	b	b

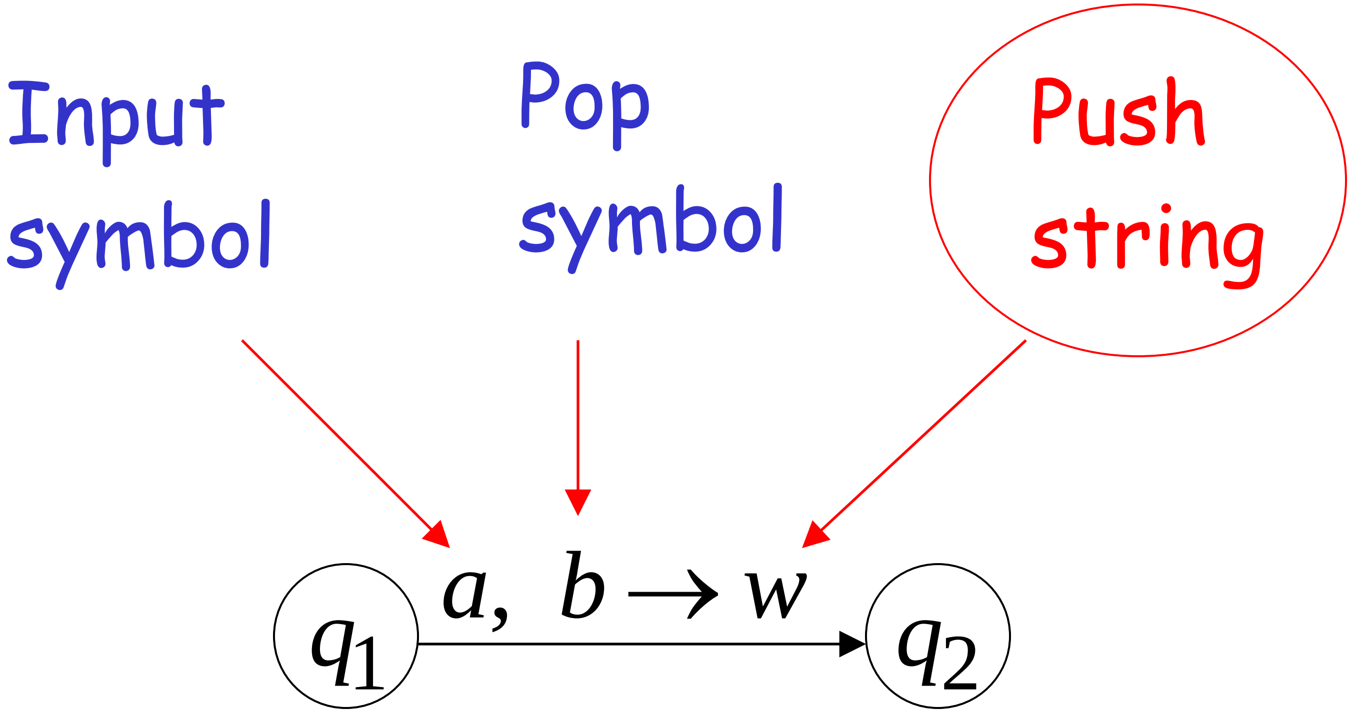




Stack

Halt and Reject

Pushing Strings

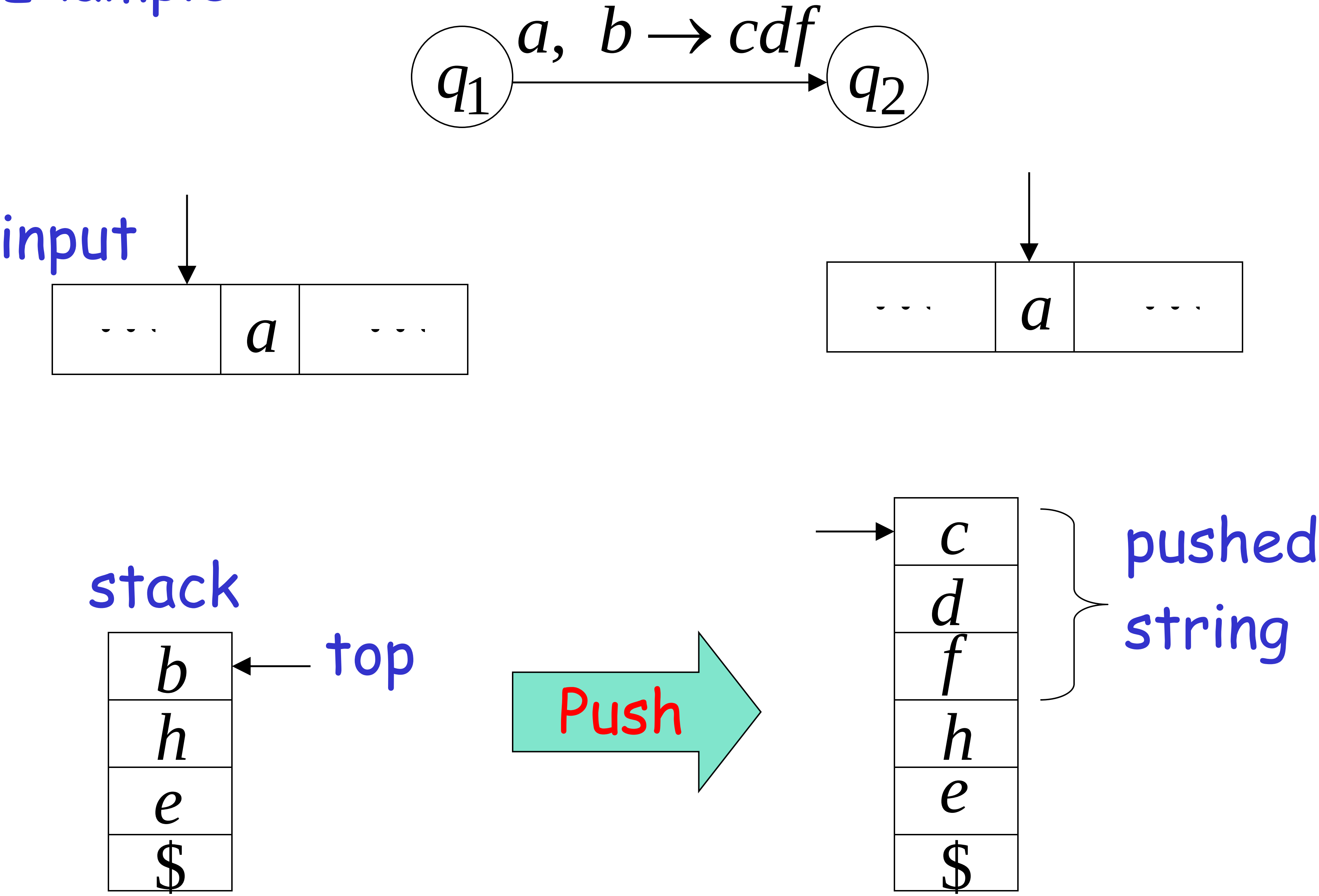


Input symbol

Pop symbol

Push string

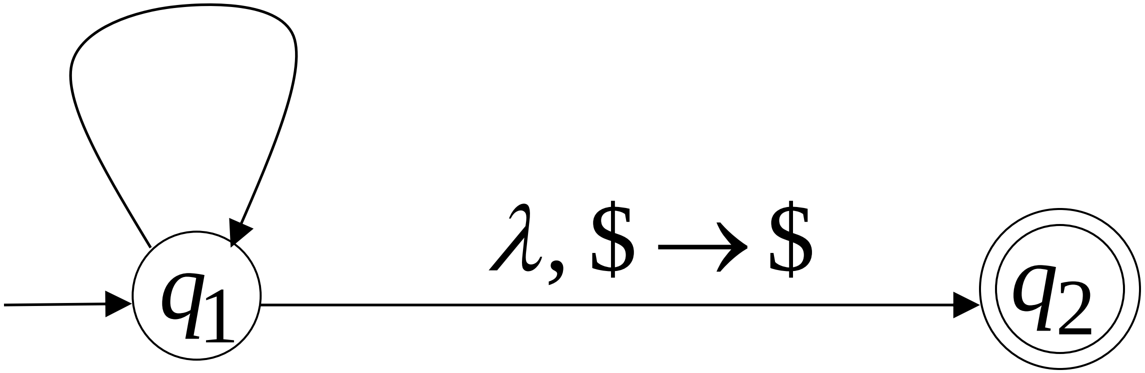
Example:


$a, \; b \to cdf$
q_1
q_2
input
a
a
stack
top
b
h
e
$\$$
Push
c
d
f
h
e
$\$$
pushed
string

Another NPDA example

NPDA M

$$L(M) = \{w: \quad n_a = n_b\}$$



$a, \$ \rightarrow 0\$$ $\quad$ $b, \$ \rightarrow 1\$$

$a, 0 \rightarrow 00$ $\quad$ $b, 1 \rightarrow 11$

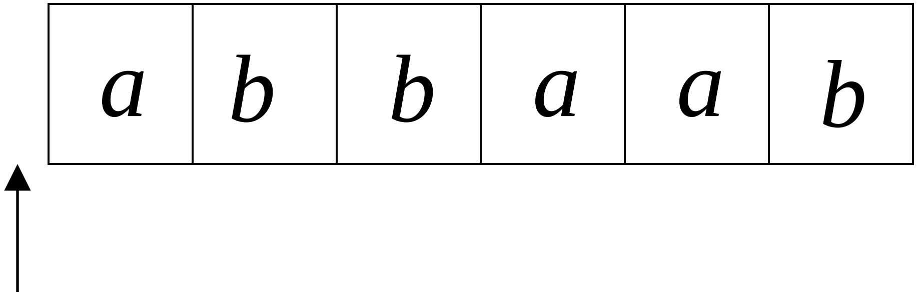
$a, 1 \rightarrow \lambda$ $\quad$ $b, 0 \rightarrow \lambda$

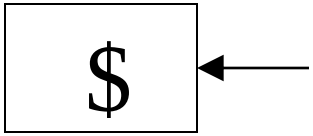
$\lambda, \$ \rightarrow \$$

q_1 $\quad$ q_2

Execution Example: Time 0

Input





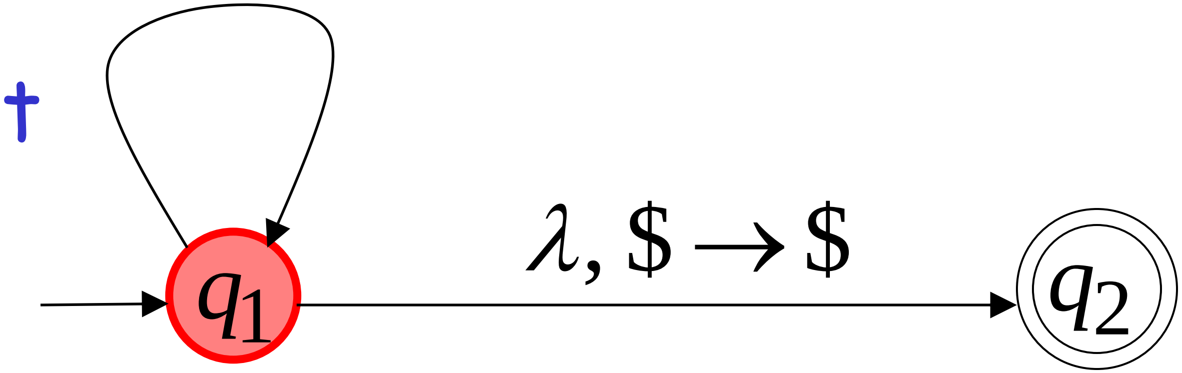
Stack

$a, \$ \rightarrow 0\$$ $b, \$ \rightarrow 1\$$

$a, 0 \rightarrow 00$ $b, 1 \rightarrow 11$

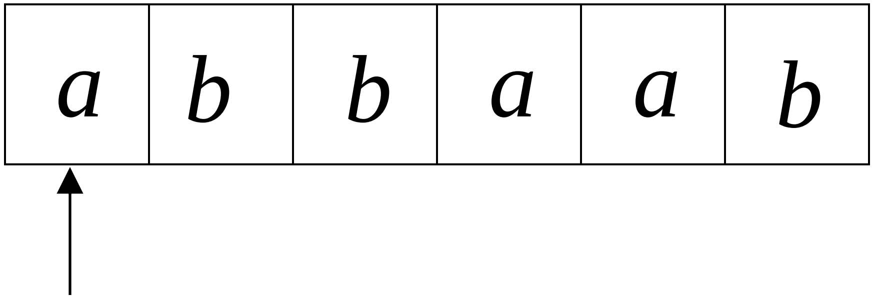
$a, 1 \rightarrow \lambda$ $b, 0 \rightarrow \lambda$

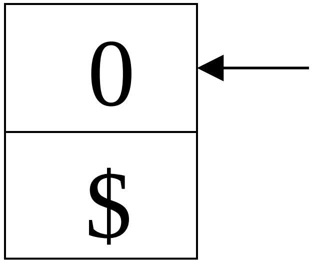
current state



Time 1

Input



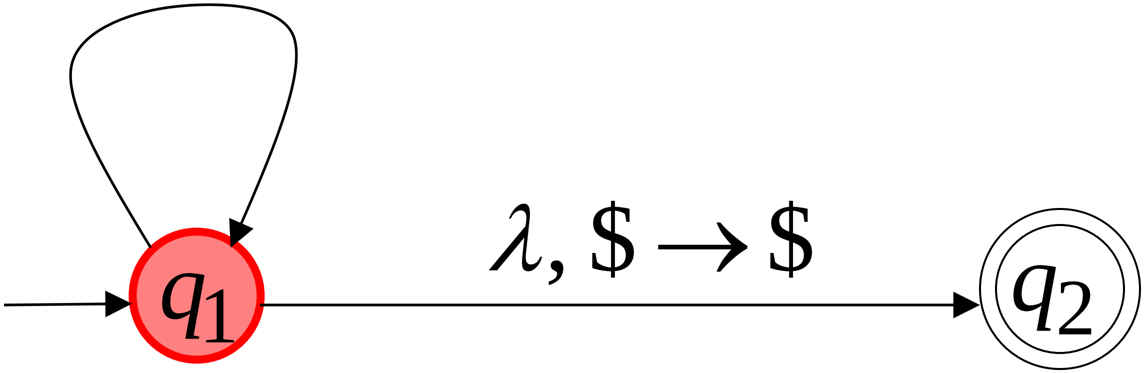


Stack

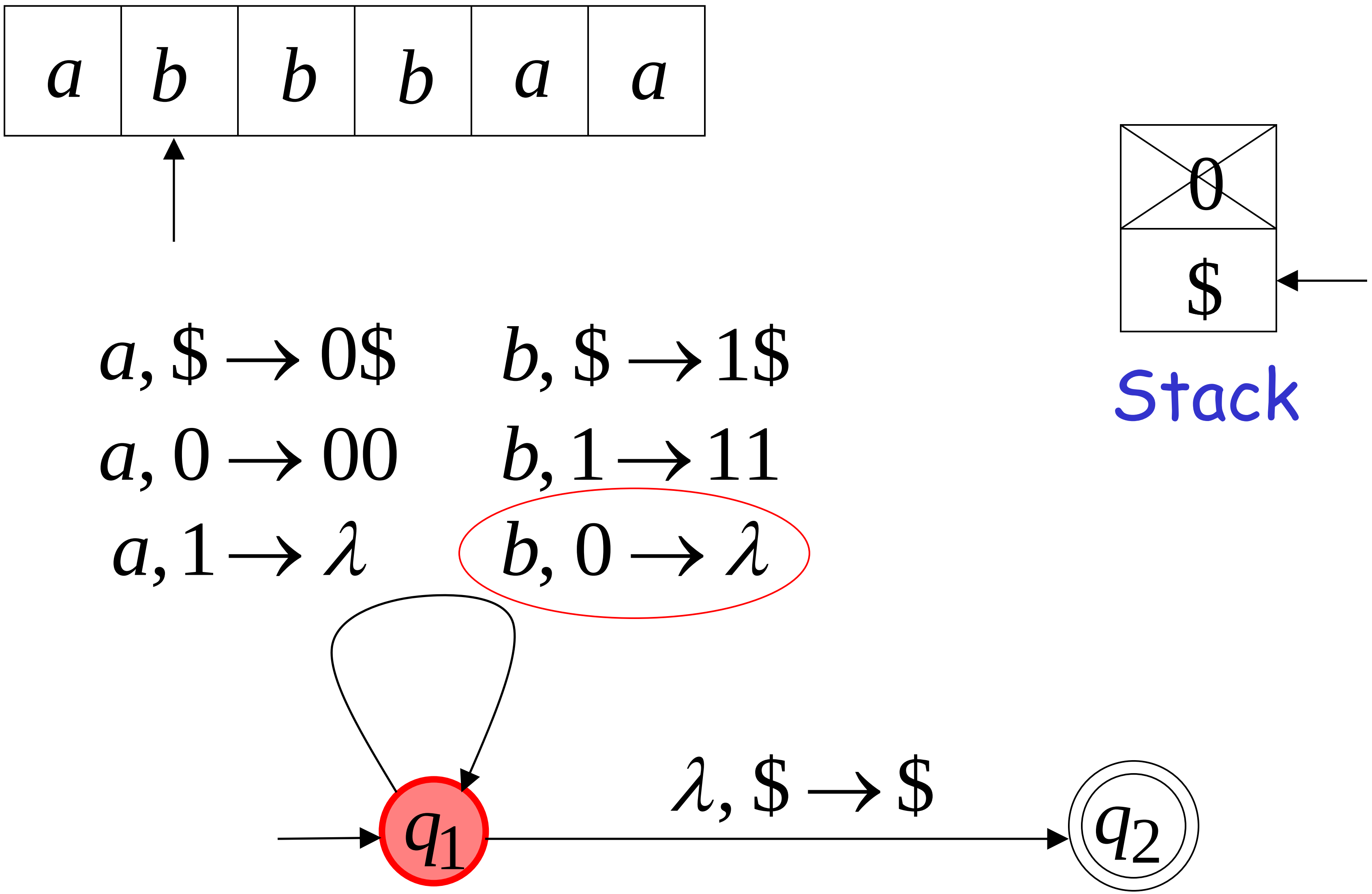
$$a, \$ \rightarrow 0\$ \qquad b, \$ \rightarrow 1\$$$

$$a, 0 \rightarrow 00 \qquad b, 1 \rightarrow 11$$

$$a, 1 \rightarrow \lambda \qquad b, 0 \rightarrow \lambda$$



Time 3



Input

a	b	b	b	a	a

Stack

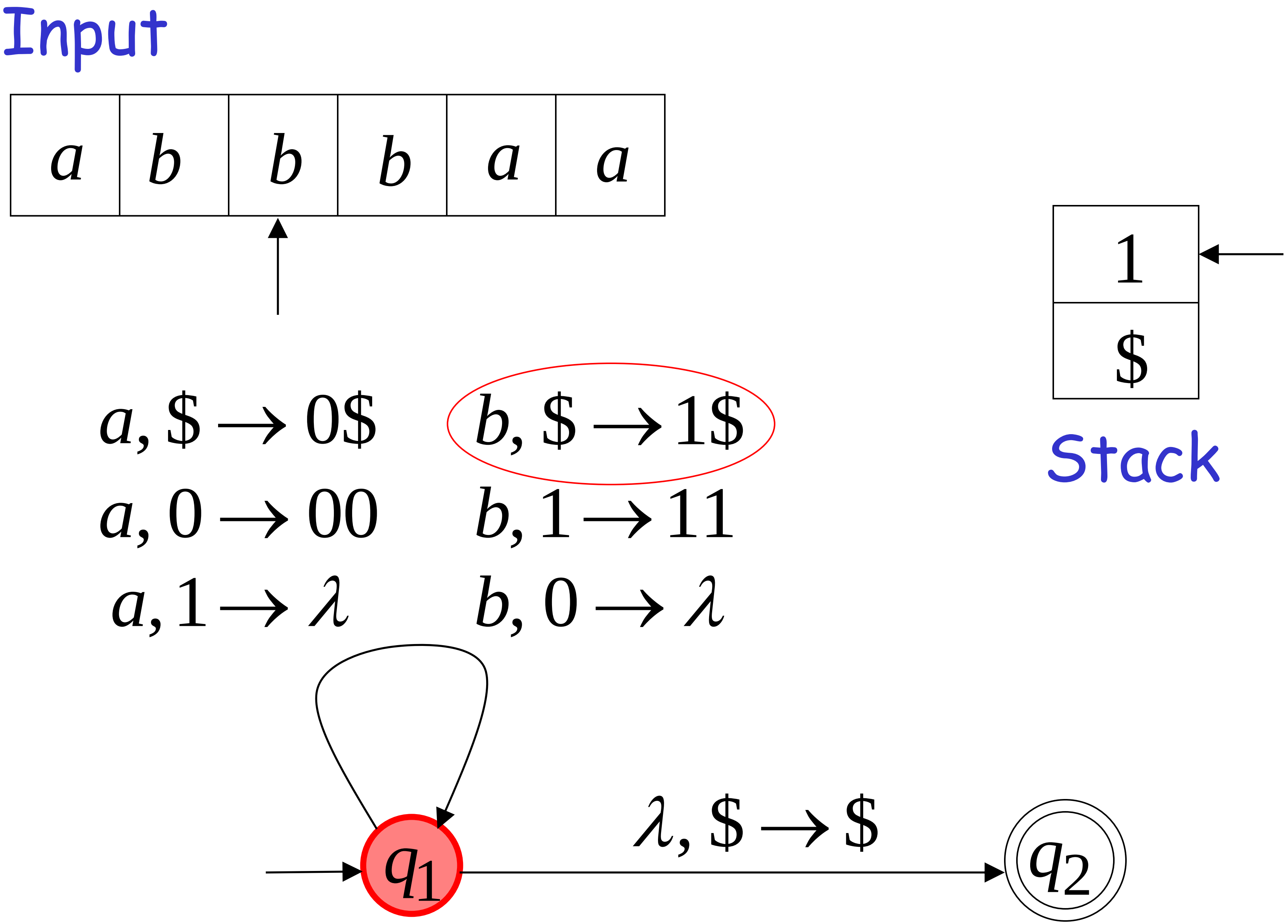
$\cancel{0}$
$\$$

$a, \$ \rightarrow 0\$$ $\quad$ $b, \$ \rightarrow 1\$$

$a, 0 \rightarrow 00$ $\quad$ $b, 1 \rightarrow 11$

$a, 1 \rightarrow \lambda$ $\quad$ $b, 0 \rightarrow \lambda$

Time 4



Input

a	b	b	b	a	a

1
$

Stack

$$a, \$ \rightarrow 0\$ \qquad b, \$ \rightarrow 1\$$$

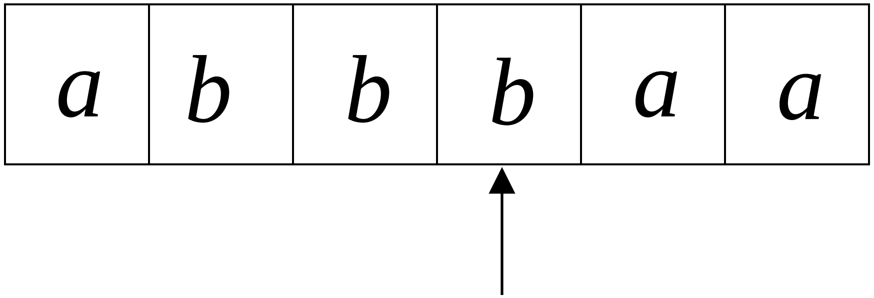
$$a, 0 \rightarrow 00 \qquad b, 1 \rightarrow 11$$

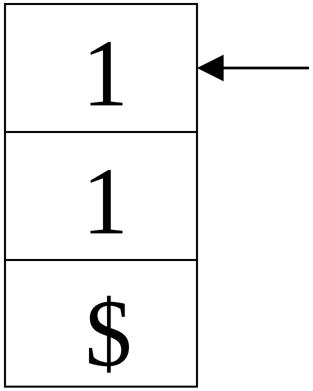
$$a, 1 \rightarrow \lambda \qquad b, 0 \rightarrow \lambda$$

$$\lambda, \$ \rightarrow \$$$

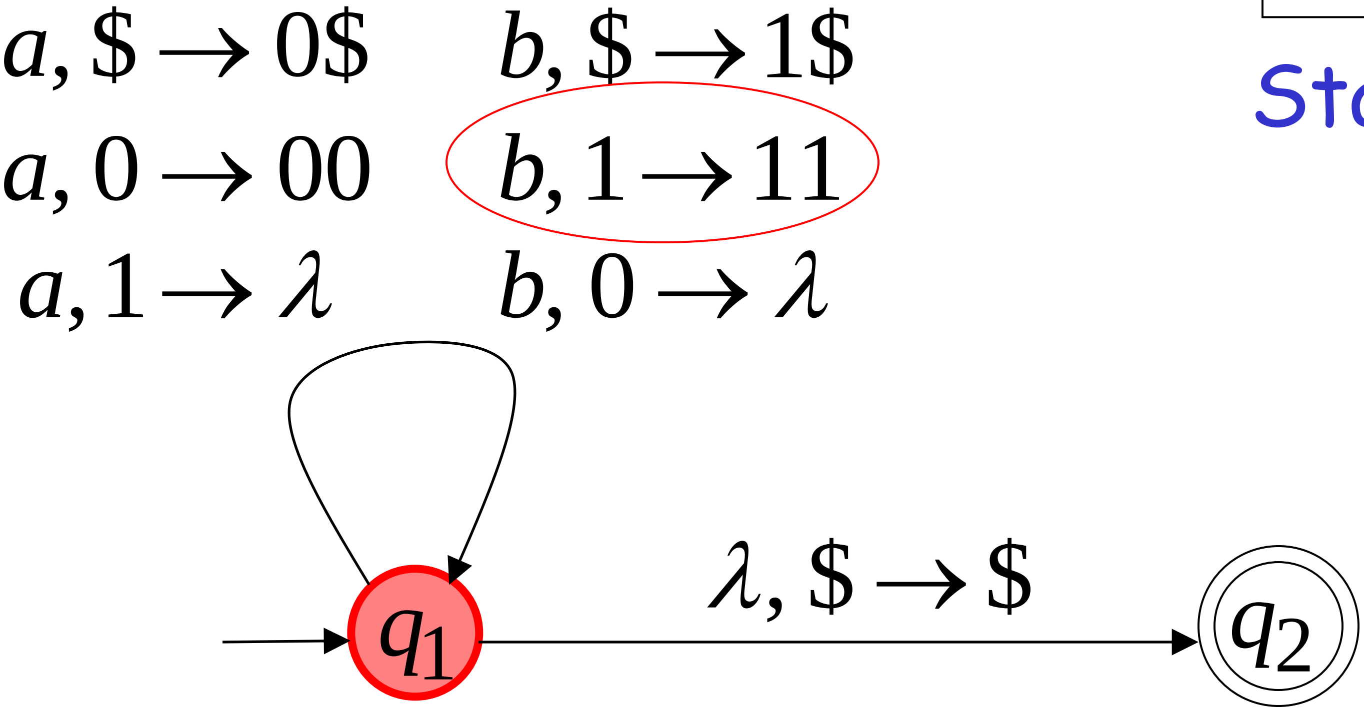
Time 5

Input

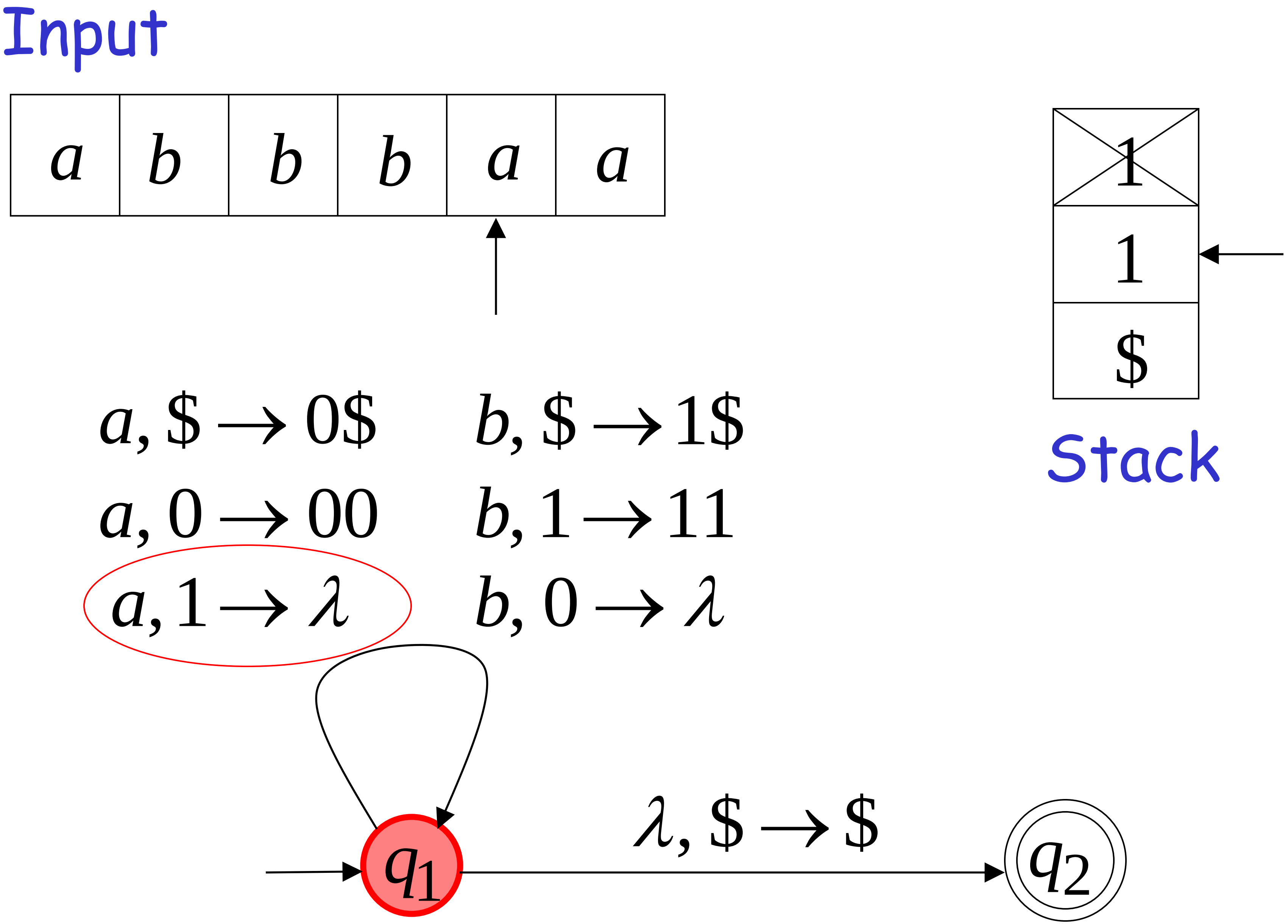




Stack



Time 6


Input
a b b b a a
$\xcancel{1}$
1
$\$$
Stack
$a, \$ \rightarrow 0\$$
$b, \$ \rightarrow 1\$$
$a, 0 \rightarrow 00$
$b, 1 \rightarrow 11$
$a, 1 \rightarrow \lambda$
$b, 0 \rightarrow \lambda$
$\lambda, \$ \rightarrow \$$
q_1
q_2

Time 7

Input

a	b	b	b	a	a

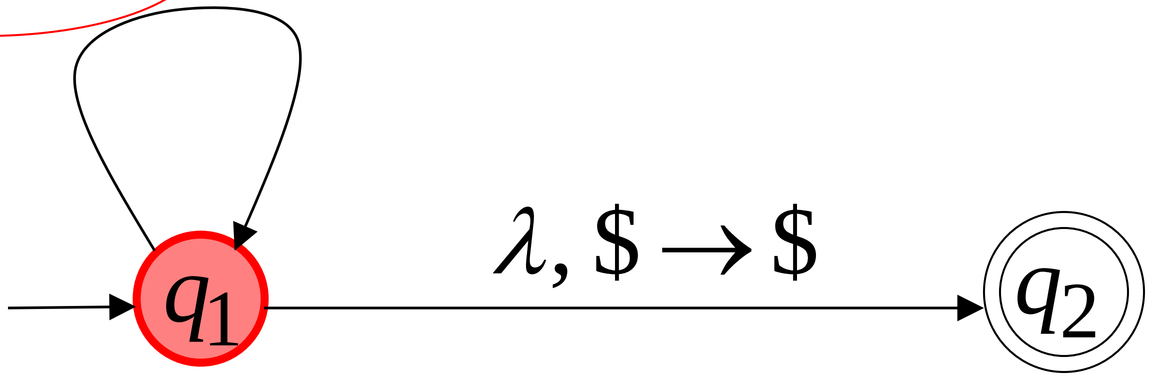
$\cancel{1}$
$\$$

Stack

$$a, \$ \rightarrow 0\$ \qquad b, \$ \rightarrow 1\$$$

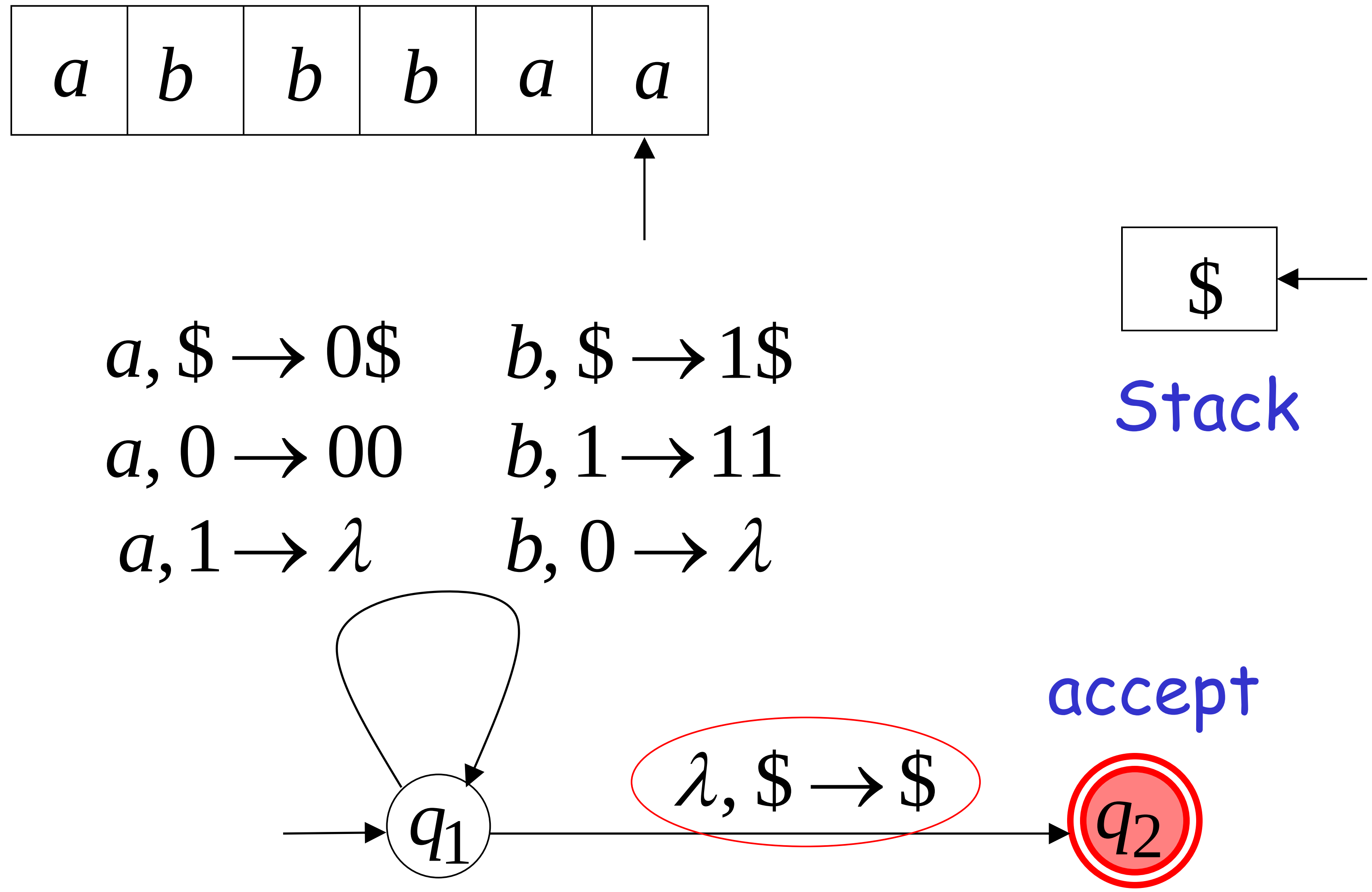
$$a, 0 \rightarrow 00 \qquad b, 1 \rightarrow 11$$

$$\boxed{a, 1 \rightarrow \lambda} \qquad b, 0 \rightarrow \lambda$$



Time 8

Input

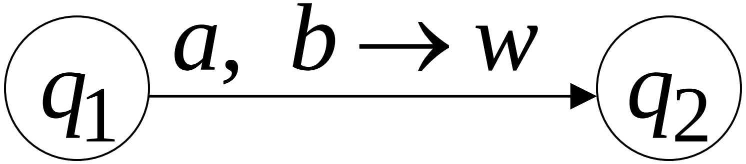


$a, \$ \to 0\$$ $\quad$ $b, \$ \to 1\$$

$a, 0 \to 00$ $\quad$ $b, 1 \to 11$

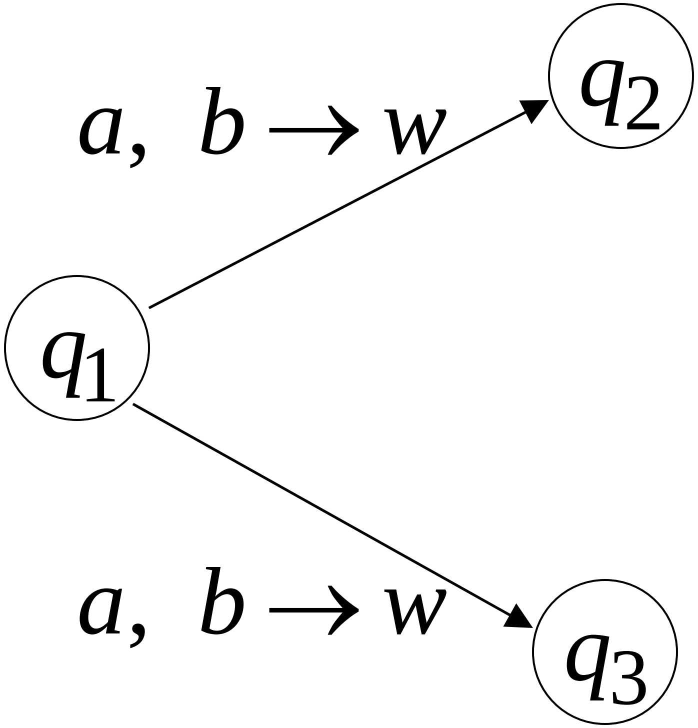
$a, 1 \to \lambda$ $\quad$ $b, 0 \to \lambda$

Formalities for NPDAs



Transition function:

$$\delta(q_1, a, b) = \{(q_2, w)\}$$



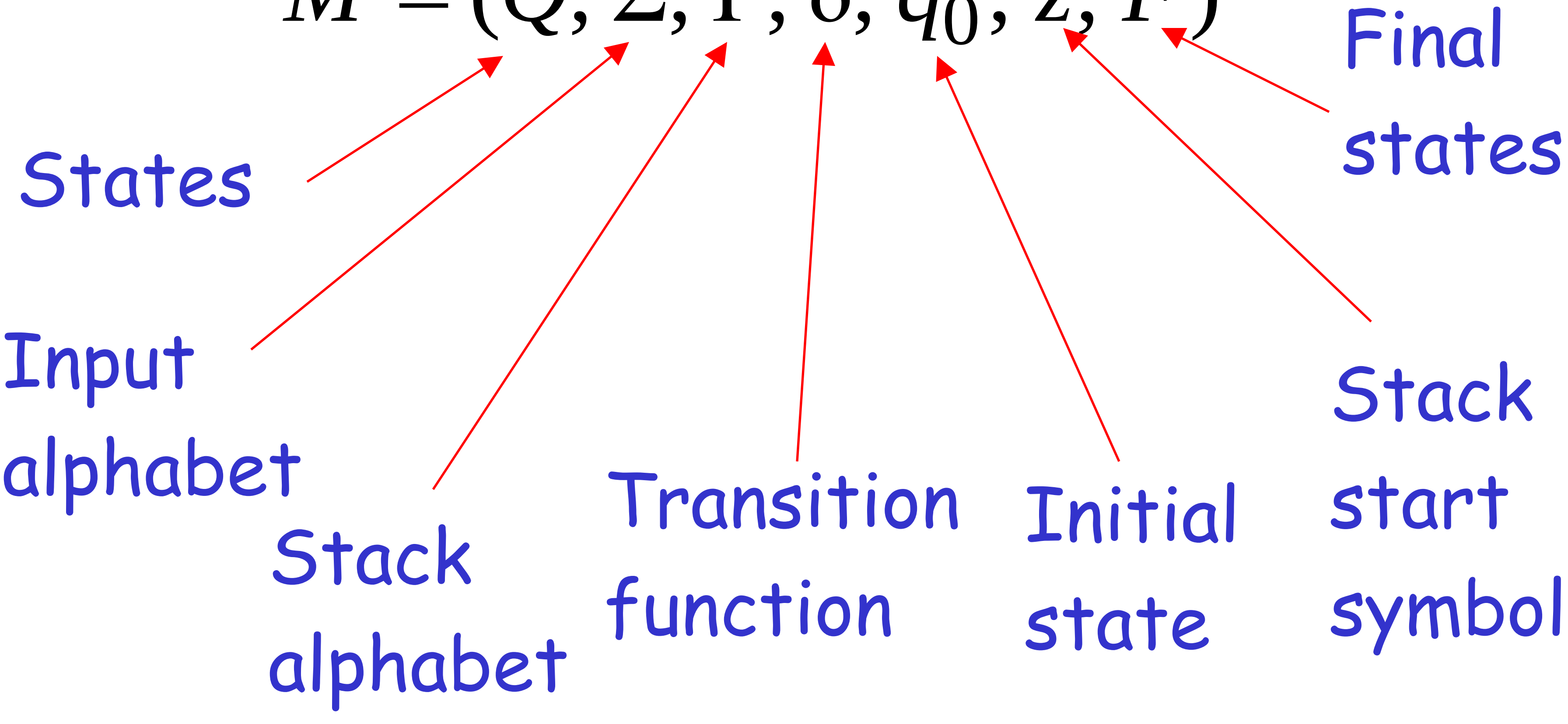
Transition function:

$$\delta(q_1, a, b) = \{(q_2, w), \ (q_3, w)\}$$

Formal Definition

Non-Deterministic Pushdown Automaton
NPDA

$$M = (Q, \Sigma, \Gamma, \delta, q_0, z, F)$$



Instantaneous Description

$$(q,\ u,\ s)$$

Current state

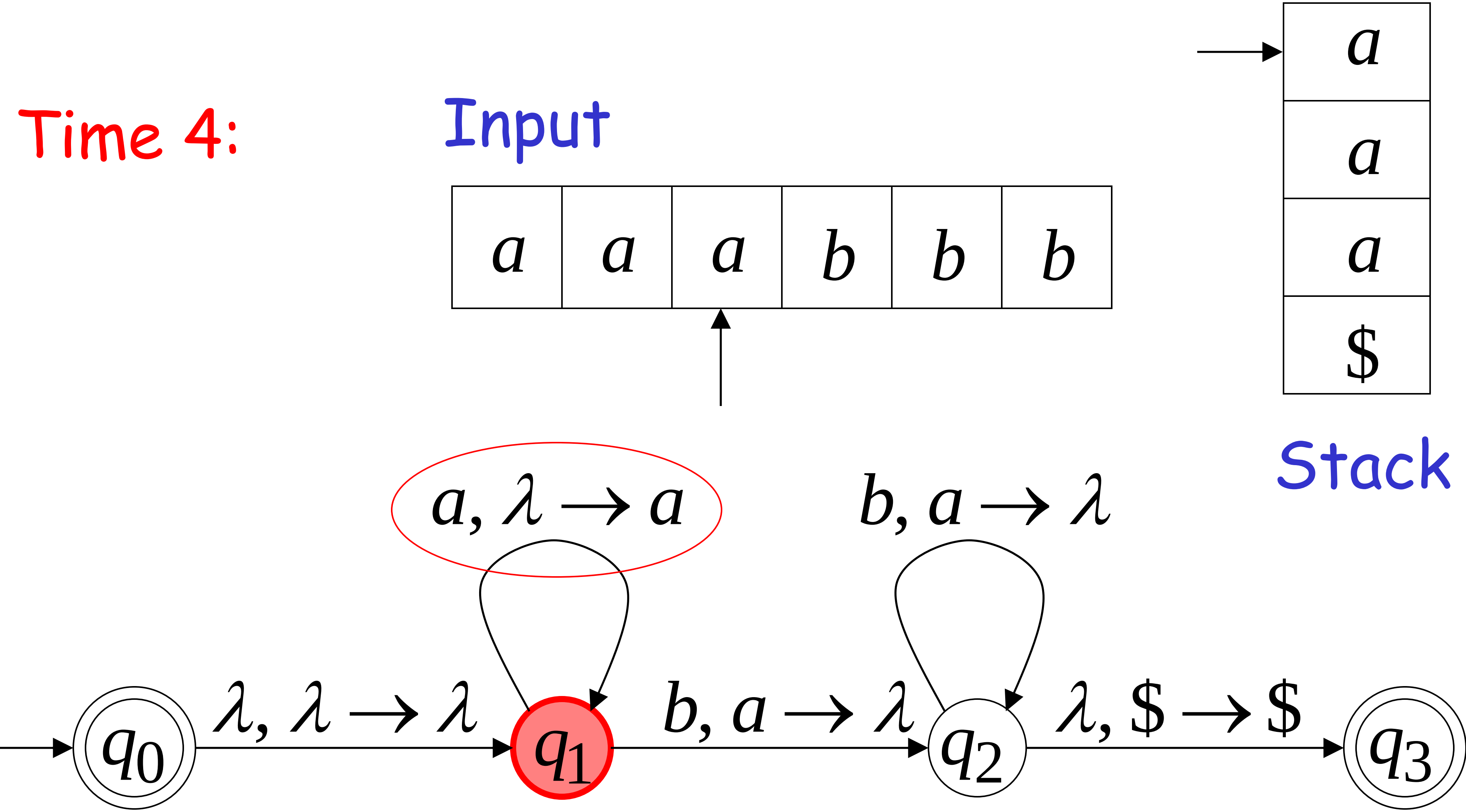
Remaining input

Current stack contents

Example: Instantaneous Description

$$(q_1, bbb, aaa\$)$$

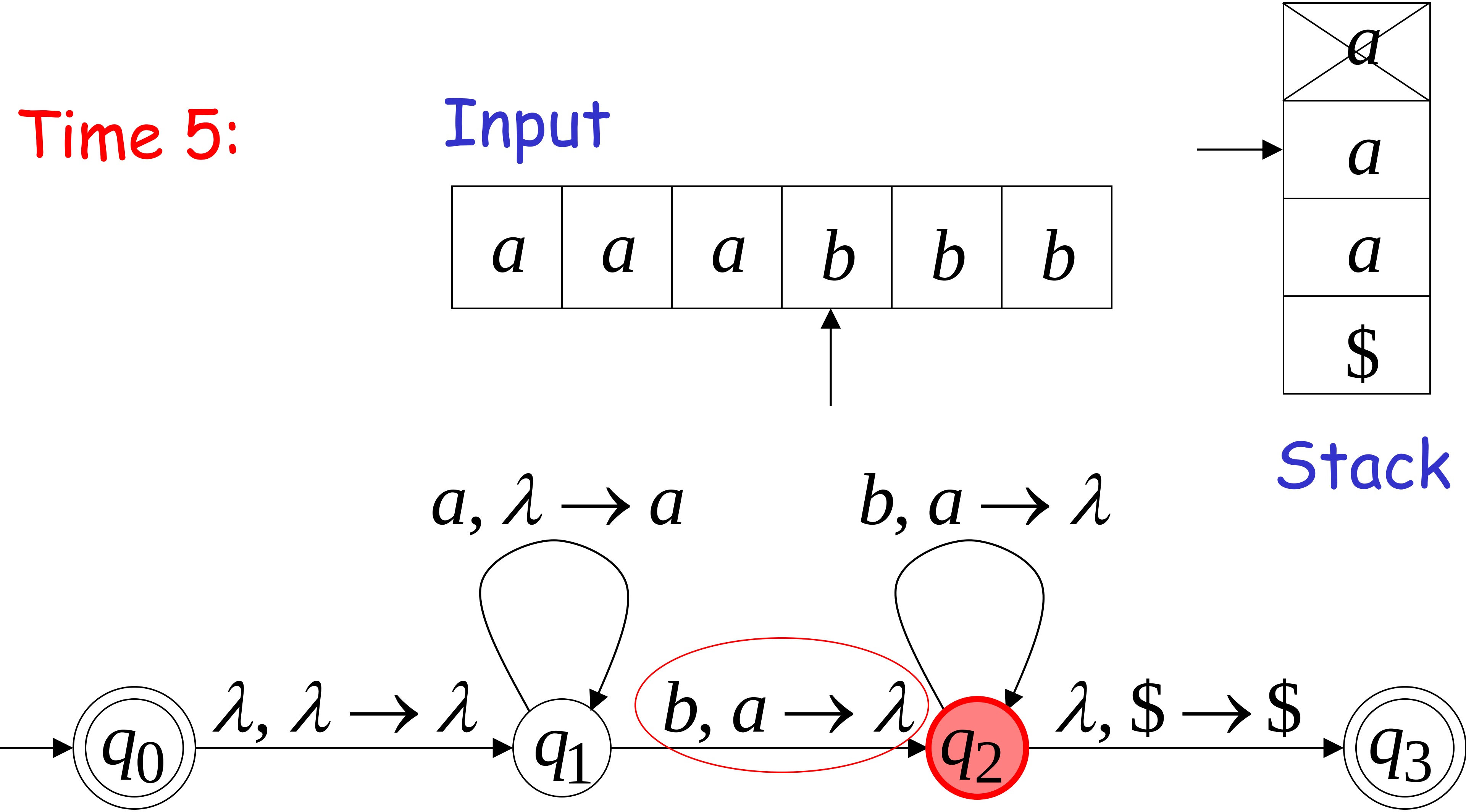
Time 4:



Example: **Instantaneous Description**

$$(q_2, bb, aa\$)$$

Time 5:



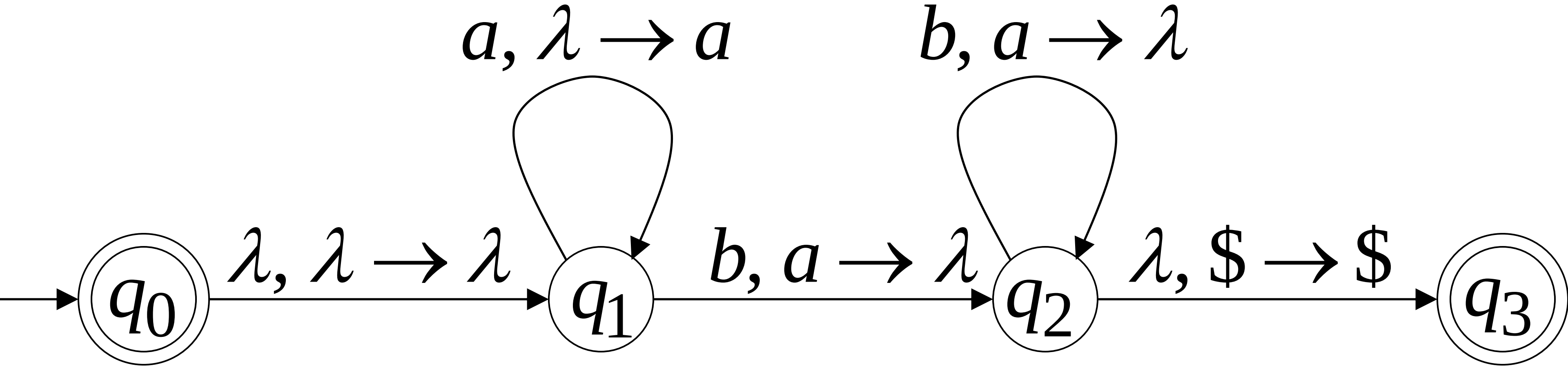
We write:

$$(q_1, bbb, aaa\$) \succ (q_2, bb, aa\$)$$

Time 4 Time 5

A computation:

$$(q_0, aaabbb, \$) \succ (q_1, aaabbb, \$) \succ$$
$$(q_1, aabbb, a\$) \succ (q_1, abbb, aa\$) \succ (q_1, bbb, aaa\$) \succ$$
$$(q_2, bb, aa\$) \succ (q_2, b, a\$) \succ (q_2, \lambda, \$) \succ (q_3, \lambda, \$)$$



$$(q_0, aaabbb,\$) \succ (q_1, aaabbb,\$) \succ$$

$$(q_1, aabbb, a\$) \succ (q_1, abbb, aa\$) \succ (q_1, bbb, aaa\$) \succ$$

$$(q_2, bb, aa\$) \succ (q_2, b, a\$) \succ (q_2, \lambda,\$) \succ (q_3, \lambda,\$)$$

For convenience we write:

$$(q_0, aaabbb,\$) \overset{*}{\succ} (q_3, \lambda,\$)$$

Formal Definition

Language $L(M)$ of NPDA M:

$$L(M)=\{w:\ (q_0,w,s)\overset{*}{\succ}(q_f,\lambda,s')\}$$

Initial state

Final state

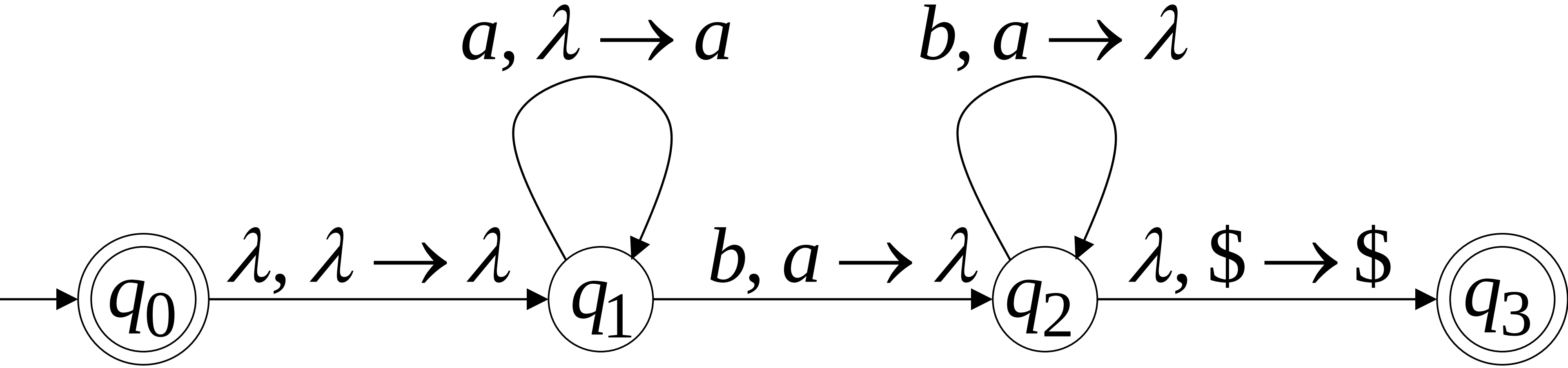
Example:

$$(q_0, aaabbb, \$) \overset{*}{\succ} (q_3, \lambda, \$)$$

⇓

$$aaabbb \in L(M)$$

NPDA M :

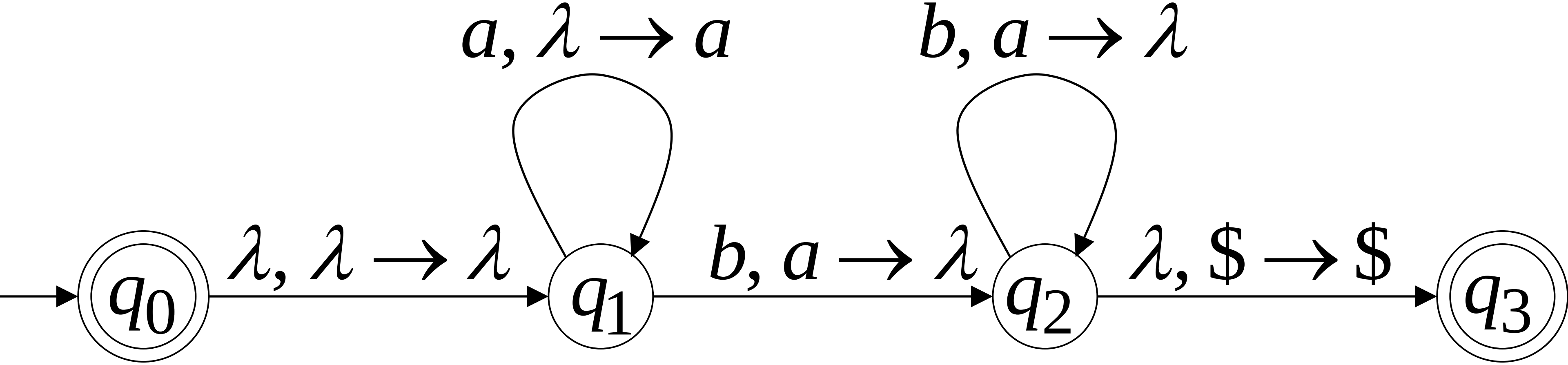


$$(q_0, a^n b^n, \$) \overset{*}{\succ} (q_3, \lambda, \$)$$

$$\Downarrow$$

$$a^n b^n \in L(M)$$

NPDA M :



Therefore: $L(M) = \{a^n b^n : n \geq 0\}$

NPDA M: