Context-Free Grammars

Formal Languages and Abstract Machines

Week 06

Baris E. Suzek, PhD

Outline

- Last week
- Context-free Grammars and Definitions
- Simplifications of Context-free Grammars
- Normal forms for Context-free Grammars
- CYK Parser
- Pushdown automata

Regular Languages

Regular Grammars

$$S \rightarrow aA \mid B$$

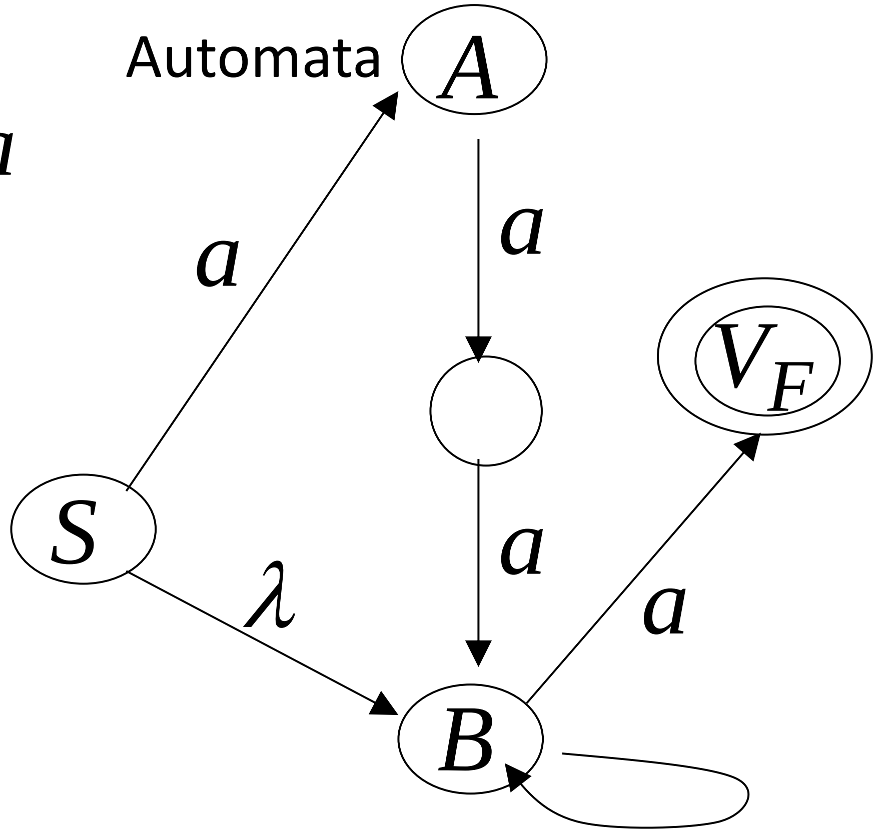
$$A \rightarrow aa\ B$$

$$B \rightarrow bB \mid a$$

Regular Expressions

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

Finite Automata



Another Example

- Grammar:

$$S \to aSb$$
$$S \to \lambda$$

- Derivation of sentence : ab

$$S \Rightarrow aSb \Rightarrow ab$$

$$S \to aSb \qquad S \to \lambda$$

More Notation

- <u>Grammar:</u> $G = (V, T, S, P)$

$V:$ Set of variables

$T:$ Set of terminal symbols

$S:$ Start variable

$P:$ Set of production rules

More Derivations

$$S \to Ab$$
$$A \to aAb$$
$$A \to \lambda$$

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

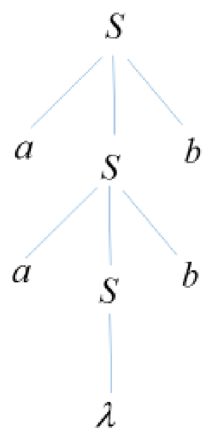
$$S \overset{*}{\Rightarrow} aaaabbbbb$$

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

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

Example

$$S \to aSb$$
$$S \to \lambda$$



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

Right-Linear Grammars

- All productions have form: $A \to xB$

  or

  $A \to x$

  string of terminals

- Example: $S \to abS$

  $S \to a$

Left-Linear Grammars

- All production rules have form: $A \to Bx$

  or

  $A \to x$

  string of terminals

- Example: $S \to Aab$

  $A \to Aab \mid B$

  $B \to a$

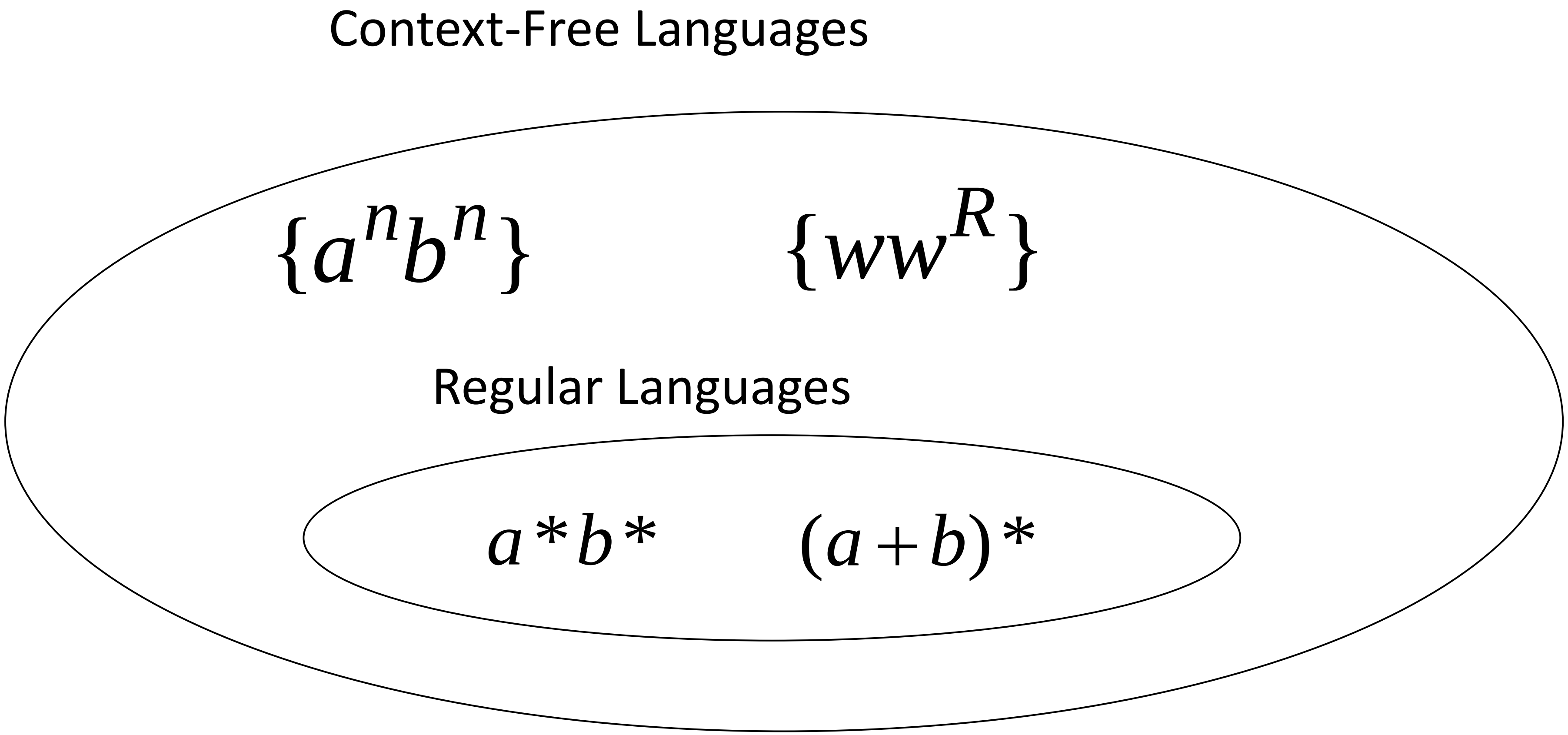
Regular Grammars

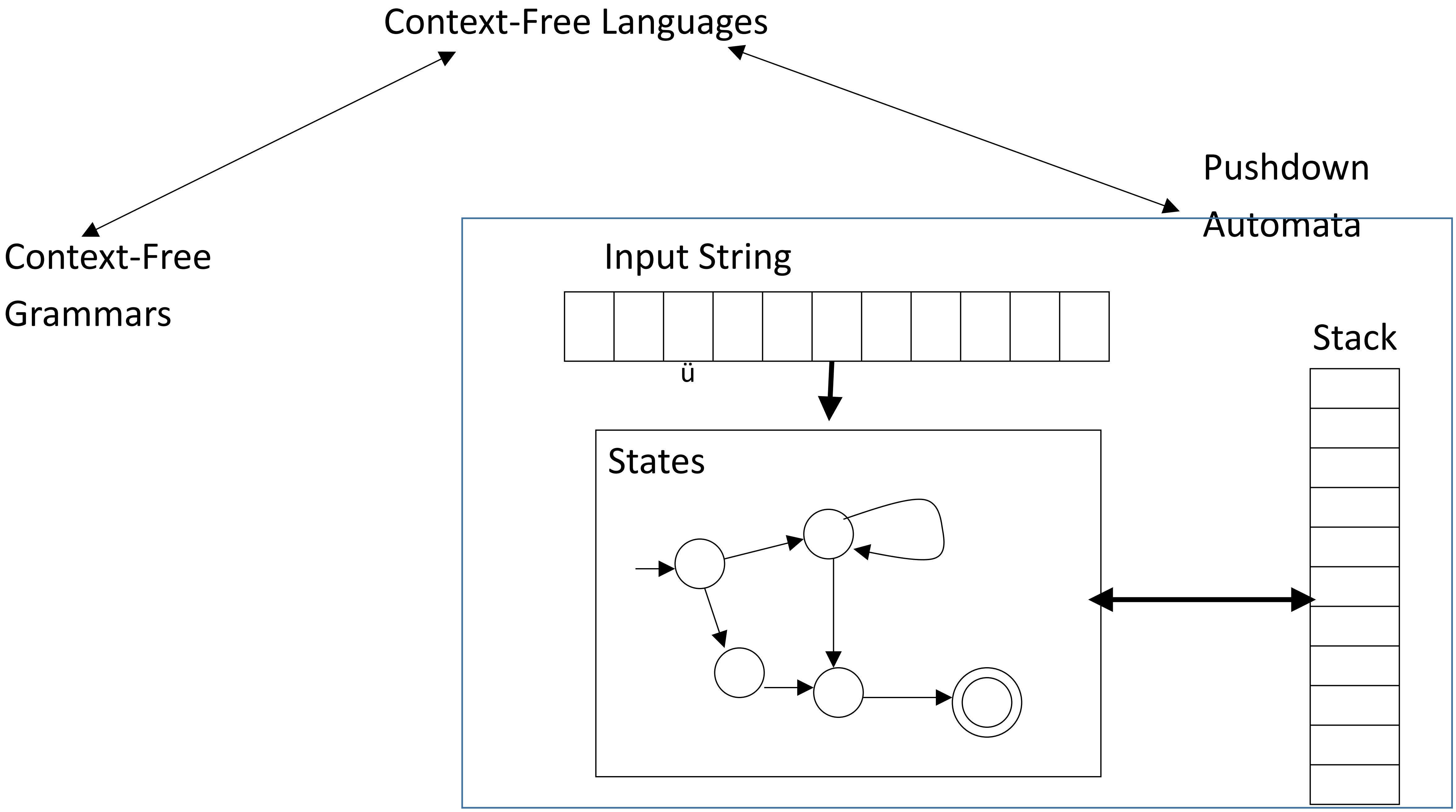
- A regular grammar is any right-linear or left-linear grammar

- Examples:

G_1	G_2
$S \to abS$	$S \to Aab$
$S \to a$	$A \to Aab \mid B$
	$B \to a$

Context-Free and Regular Languages




Context-Free Languages
Context-Free
Grammars
Pushdown
Automata
Input String
ü
States
Stack

Example

A context-free grammar G :

$$S \rightarrow aSb$$

$$S \rightarrow \lambda$$

Example derivations:

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

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

Describes parentheses in format: (((( ))))

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

Note: There is no constraint on linear-ness

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

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

Derivation Order

- $1.\ S \to AB$

$2.\ A \to aaA$ $\quad$ $4.\ B \to Bb$

$3.\ A \to \lambda$ $\quad$ $5.\ B \to \lambda$

Leftmost derivation:

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

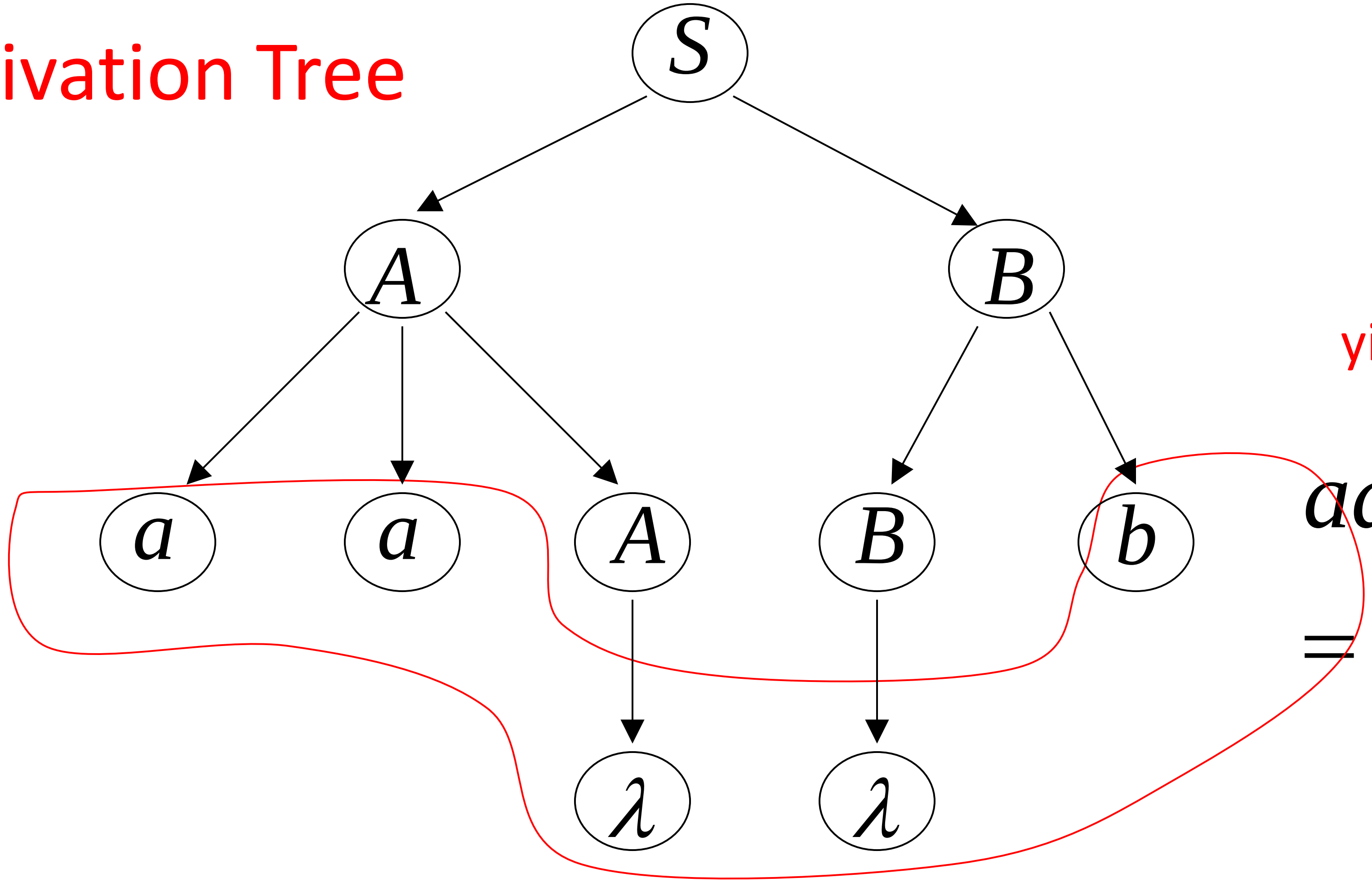
Rightmost derivation:

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

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

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

Derivation Tree



yield

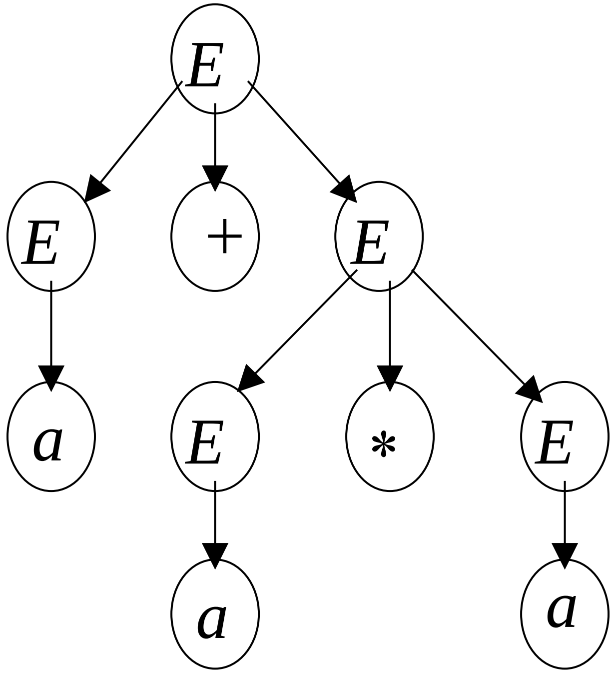
$aa\lambda\lambda b$

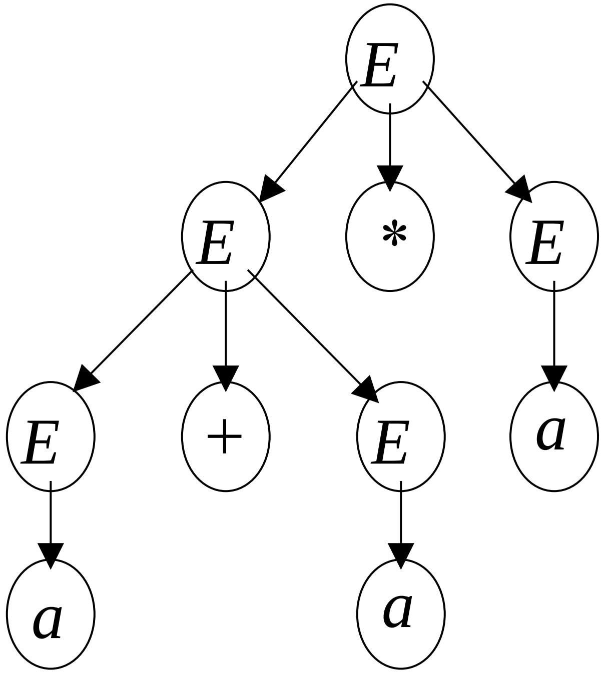
$= aab$

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

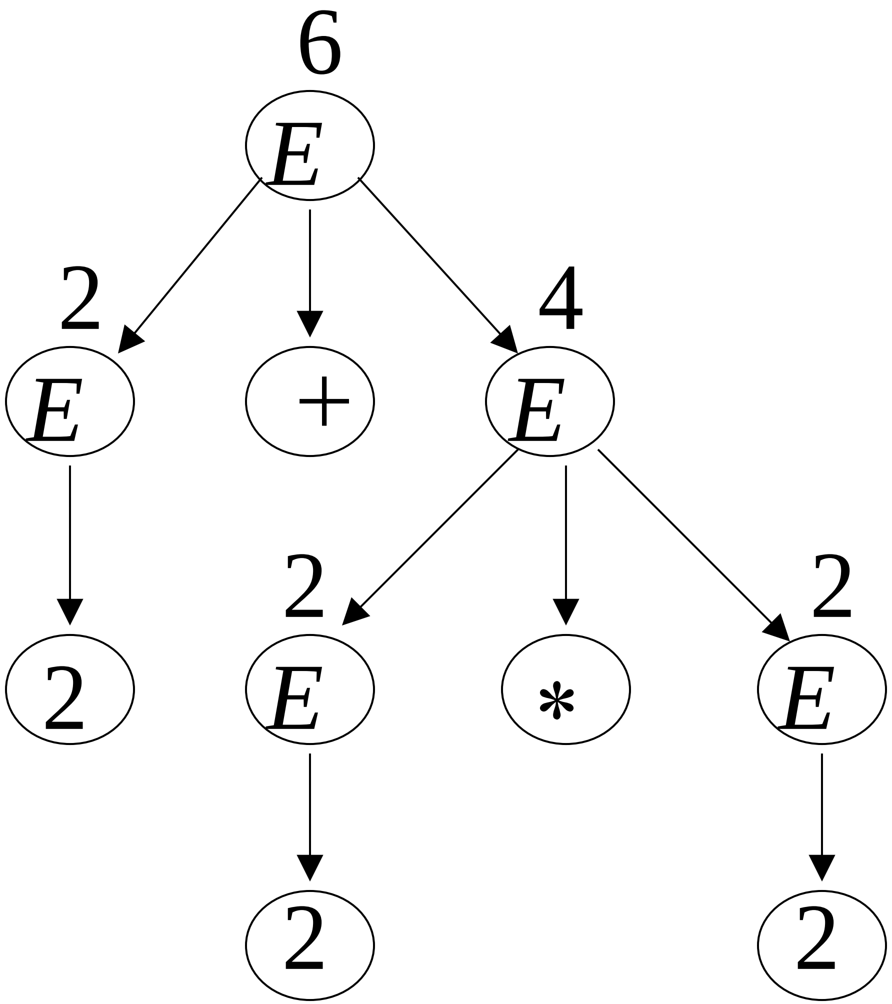
is ambiguous:

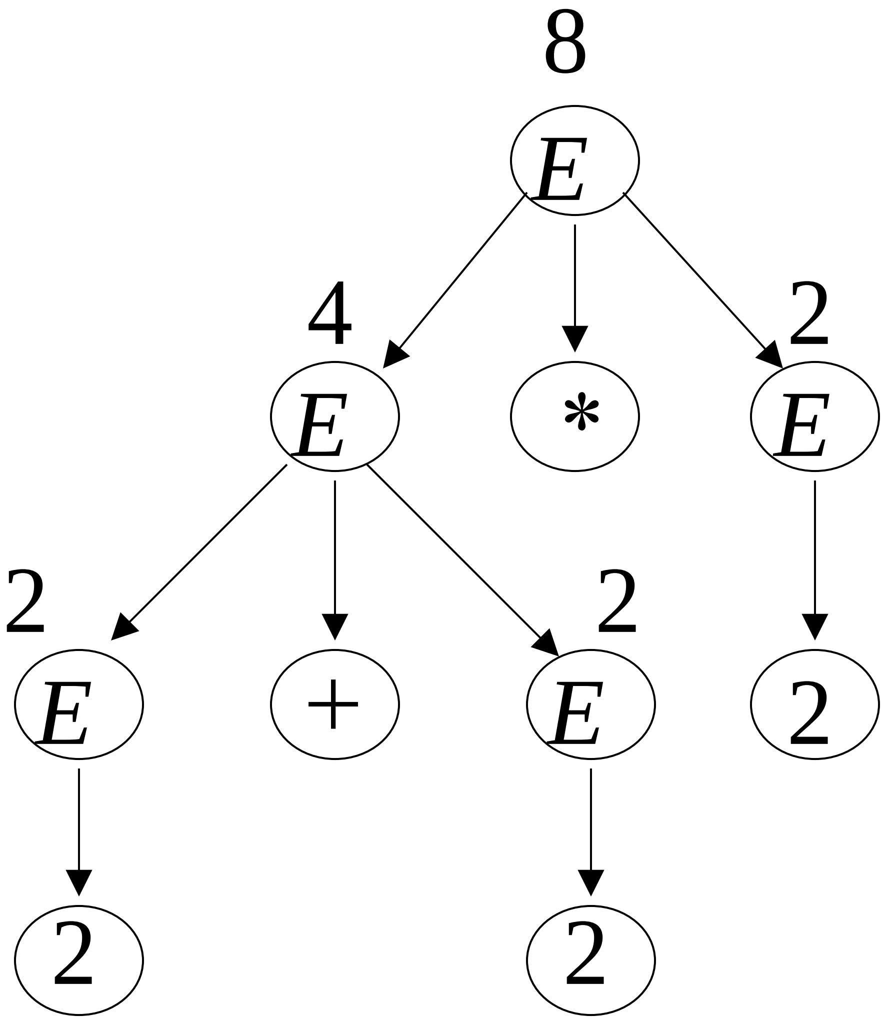
string $a + a * a$ has two derivation trees





$$2+2*2=6 \qquad\qquad 2+2*2=8$$





We fix the ambiguous grammar:

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

New non-ambiguous grammar:

$$E \to E + T$$

$$E \to T$$

$$T \to T * F$$

$$T \to F$$

$$F \to (E)$$

$$F \to a$$

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

Outline

- Last week
- Context-free Grammars and Definitions
- Simplifications of Context-free Grammars
- Normal forms for Context-free Grammars
- CYK Parser
- Pushdown automata

Parser

Example:

parser

$$S \to SS$$

$$S \to aSb$$

$$S \to bSa$$

$$S \to \lambda$$

input

$aabb$

derivation

?

Exhaustive Search

$$S \to SS \mid aSb \mid bSa \mid \lambda$$

Find derivation of $aabb$

Phase 1:

$S \Rightarrow SS$ $\quad$ $S \Rightarrow SS$

$S \Rightarrow aSb$ $\quad$ $S \Rightarrow aSb$

$S \Rightarrow bSa$ $\quad$ $\cancel{S \Rightarrow bSa}$

$S \Rightarrow \lambda$ $\quad$ $\cancel{S \Rightarrow \lambda}$

All possible derivations of length 1

Phase 2 $S \to SS \mid aSb \mid bSa \mid \lambda$

$S \Rightarrow SS \Rightarrow SSS$ $aabb$

$S \Rightarrow SS \Rightarrow aSbS$

$\cancel{S \Rightarrow SS \Rightarrow bSaS}$

Phase 1

$S \Rightarrow SS$ $S \Rightarrow SS \Rightarrow S$

$S \Rightarrow aSb$ $S \Rightarrow aSb \Rightarrow aSSb$

$S \Rightarrow aSb \Rightarrow aaSbb$

$\cancel{S \Rightarrow aSb \Rightarrow abSab}$

$\cancel{S \Rightarrow aSb \Rightarrow ab}$

Phase 2

$$S \rightarrow SS \mid aSb \mid bSa \mid \lambda$$

$$aabb$$

$$S \Rightarrow SS \Rightarrow SSS$$

$$S \Rightarrow SS \Rightarrow aSbS$$

$$S \Rightarrow SS \Rightarrow S$$

$$S \Rightarrow aSb \Rightarrow aSSb$$

Phase 3

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

Final result of exhaustive search (top-down parsing)

Input

$aabb$

Parser

$$S \to SS$$
$$S \to aSb$$
$$S \to bSa$$
$$S \to \lambda$$

Derivation

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

Time complexity of exhaustive search

Suppose there are no productions of the form

$$A \rightarrow \lambda$$

$$A \rightarrow B$$

Number of phases for string w : $|w|$

For grammar with k rules

Time for phase 1: k possible derivations

Time for phase 2: k^2 possible derivations

Time for phase $|w|$: $k^{|w|}$ possible derivations

Total time needed for string w:

$$k + k^2 + \cdots + k^{|w|}$$

phase 1

phase 2

phase |w|

Pretty bad!!!

There exist faster algorithms for specialized grammars

S-grammar:

$$A \rightarrow ax$$

symbol

string
of variables

Pair (A, a) appears once

S-grammar example:

$$S \to aS$$

$$S \to bSS$$

$$S \to c$$

Each string has a unique derivation

$$S \Rightarrow aS \Rightarrow abSS \Rightarrow abcS \Rightarrow abcc$$

For S-grammars:

In the exhaustive search parsing there is only one choice in each phase

Time for a phase: 1

Total time for parsing string w : $|w|$

For general context-free grammars:

There exists a parsing algorithm that parses a string $|w|$ in time $|w|^3$

Outline

- Last week
- Context-free Grammars and Definitions
- Simplifications of Context-free Grammars
- Normal forms for Context-free Grammars
- CYK Parser
- Pushdown automata

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

$$S \rightarrow aB \mid ab$$

$$A \rightarrow aaA$$

$$A \rightarrow abBc \mid abbc$$

$$B \rightarrow aA$$

Substitute

$$B \rightarrow aA$$

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

$$A \rightarrow aaA$$

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

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

$$S \to aMb$$

$$M \to aMb$$

$$\cancel{M \to \lambda}$$

Substitute

$$M \to \lambda$$

Final Grammar

$$S \to aMb$$

$$S \to ab$$

$$M \to aMb$$

$$M \to ab$$

Unit-Productions

Unit Production: $A \rightarrow B$

(a single variable in both sides)

Example Grammar:

$$S \to aA$$
$$A \to a$$
$$\cancel{A \to B}$$
$$B \to A$$
$$B \to bb$$

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
$$B \to B$$

$$S \to aA \mid aB$$
$$A \to a$$
$$B \to A$$
$$B \to bb$$

$$S \to aA \mid aB$$

$$A \to a$$

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

$$B \to bb$$

Substitute
$B \to A$

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

$$A \to a$$

$$B \to bb$$

Remove repeated productions

$$S \to aA \mid aB \mid \cancel{aA}$$
$$A \to a$$
$$B \to bb$$

Final grammar

$$S \to aA \mid aB$$
$$A \to a$$
$$B \to bb$$

Useless Productions

Some derivations never terminate...

$$S \to aSb$$

$$S \to \lambda$$

$$S \to A$$

$$A \to aA$$

Useless Production

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

Another grammar:

Some rules not reachable from S

$$S \to A$$

$$A \to aA$$

$$A \to \lambda$$

$$B \to bA$$

Useless Production

In general:

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

$w \in L(G)$ contains only terminals

then variable A is useful

otherwise, variable A is useless

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

Variables		Productions
	$S \to aSb$	
	$S \to \lambda$	
	$S \to A$	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

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

$$A \to a$$

$$B \to aa$$

$$C \to aCb$$

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)

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

$$A \to a$$

$$B \to aa$$

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

$\Rightarrow$

$$S \to aS \mid A$$

$$A \to a$$

$$B \to aa$$

Remove useless productions

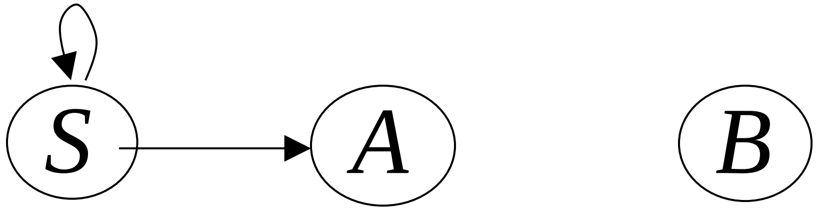
Second: Find all variables reachable from S

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)

Final Grammar

$$S \to aS \mid A$$
$$A \to a$$
$$\cancel{B \to aa}$$

$$S \to aS \mid A$$
$$A \to a$$

Remove useless productions

Removing All

- **Step 1:** Remove Nullable Variables
- **Step 2:** Remove Unit-Productions
- **Step 3:** Remove Useless Variables

Outline

- Last week
- Context-free Grammars and Definitions
- Simplifications of Context-free Grammars
- Normal forms for Context-free Grammars
- CYK Parser
- Pushdown automata

Chomsky Normal Form

Each productions has form:

$$A \to BC \qquad \text{or} \qquad A \to a$$

variable variable terminal

and no useless productions.

Examples:

$$S \to AS$$
$$S \to a$$
$$A \to SA$$
$$A \to b$$

Chomsky
Normal Form

$$S \to AS$$
$$S \to AAS$$
$$A \to SA$$
$$A \to aa$$

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

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

$$\Rightarrow$$

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

Final grammar in Chomsky Normal Form:

Initial grammar

$$S \to ABa$$
$$A \to aab$$
$$B \to Ac$$

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

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 $T_a \to a$

In productions: replace a with T_a

New variable: T_a

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

with
$$A \to C_1 V_1$$
$$V_1 \to C_2 V_2$$
$$\cdots$$
$$V_{n-2} \to C_{n-1} C_n$$

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

Exercise

S	$\longrightarrow$	aA
A	$\longrightarrow$	AbB
B	$\longrightarrow$	Bb
B	$\longrightarrow$	b

T(a)-> a

T(b)-> b

S -> T(a)A

A -> AT(b)B

B -> BT(b)

B -> b

S -> T(a)A

A -> AV(1)

V(1) -> T(b)B

B -> BT(b)

B -> b

T(a)-> a

T(b)-> b

Greinbach Normal Form

All productions have form:

$$A \to a\, V_1 V_2 \cdots V_k \qquad\qquad k \geq 0$$

symbol

variables

Examples:

$$S \to cAB$$
$$A \to aA \mid bB \mid b$$
$$B \to b$$

Greinbach
Normal Form

$$S \to abSb$$
$$S \to aa$$

Not Greinbach
Normal Form

Conversion to Greinbach Normal Form:

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

$$S \to abSb$$

$$S \to aa$$

$$S \to aT_bST_b$$

$$S \to aT_a$$

$$T_a \to a$$

$$T_b \to b$$

Greinbach
Normal Form

Observations

- Normal forms are very good for parsing

- It is hard to find the Greinbach normal form of any context-free grammar

Outline

- Last week
- Context-free Grammars and Definitions
- Simplifications of Context-free Grammars
- Normal forms for Context-free Grammars
- CYK Parser
- Pushdown automata

The CYK(Cocke–Younger–Kasami) Membership Algorithm

Input: Grammar G in Chomsky Normal Form

String w

Output: Find if string $w \in L(G)$

The Algorithm

Input example:

- Grammar G :

$$S \to AB$$

$$A \to BB$$

$$A \to a$$

$$B \to AB$$

$$B \to b$$

- String w :

$$aabbb$$

aabbb

a	a	b	b	b
aa	ab	bb	bb	
aab	abb	bbb		
aabb	abbb			
aabbb				

$$S \to AB$$

$$A \to BB$$

$$A \to a$$

$$B \to AB$$

$$B \to b$$

a	a	b	b	b
A	A	B	B	B
aa	ab	bb	bb	
aab	abb	bbb		
aabb	abbb			
aabbb				

$$S \to AB$$

$$A \to BB$$

$$A \to a$$

$$B \to AB$$

$$B \to b$$

a A	a A	b B	b B	b B
aa	ab S,B	bb A	bb A	
aab	abb	bbb		
aabb	abbb			
aabbb				

$$S \rightarrow AB$$

$$A \rightarrow BB$$

$$A \rightarrow a$$

$$B \rightarrow AB$$

$$B \rightarrow b$$

a	a	b	b	b
A	A	B	B	B
aa	ab	bb	bb	
	S,B	A	A	
aab	abb	bbb		
S,B	A	S,B		
aabb	abbb			
A	S,B			
aabbb				
S,B				

What is the computation complexity?

Therefore: $aabbb \in L(G)$

Time Complexity: $|w|^3$

Observation: The CYK algorithm can be easily converted to a parser (bottom up parser)

Exercise

S	$\longrightarrow$	AB
A	$\longrightarrow$	AA
B	$\longrightarrow$	AS
A	$\longrightarrow$	a
B	$\longrightarrow$	b

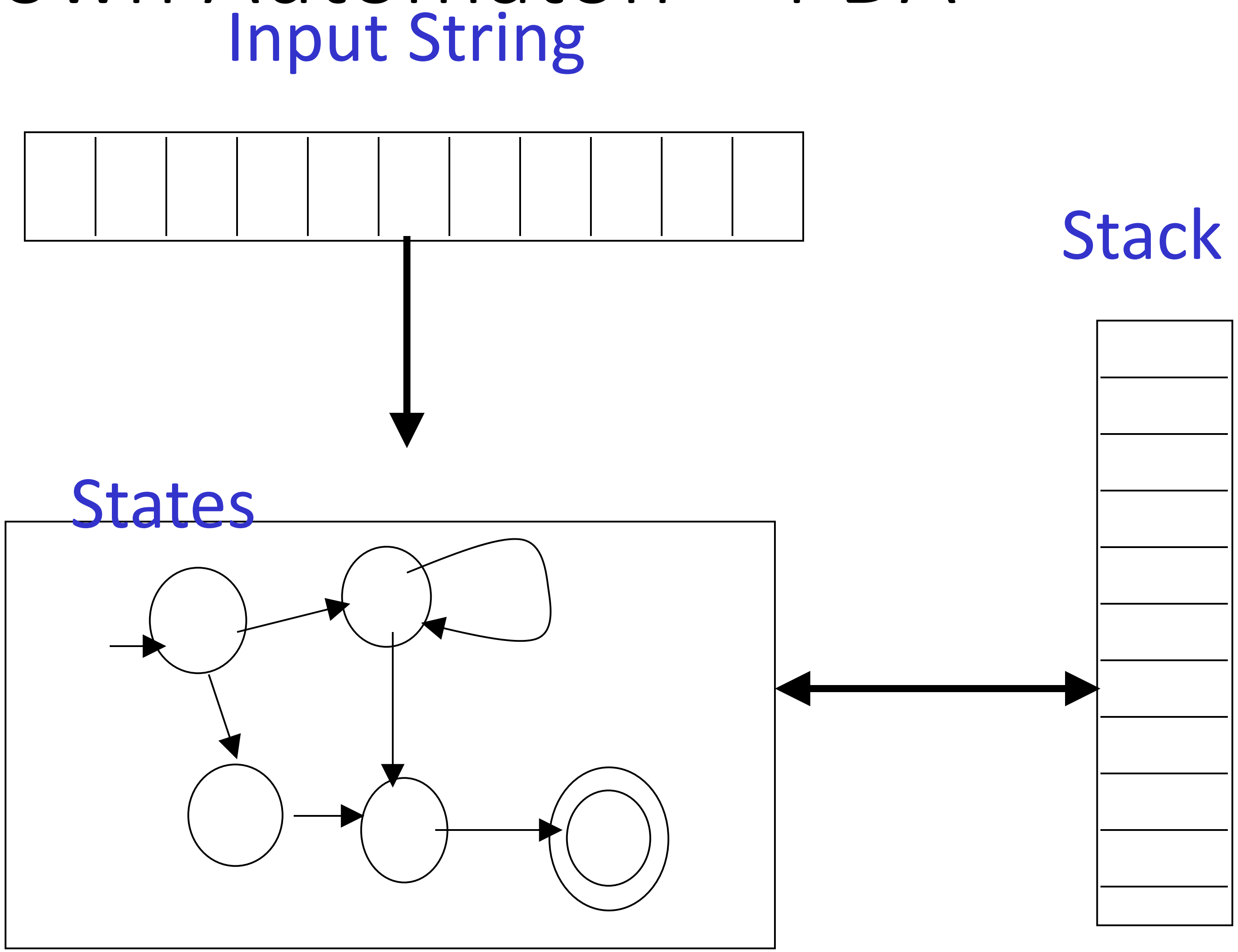
Input: aaaab

Size of the table?

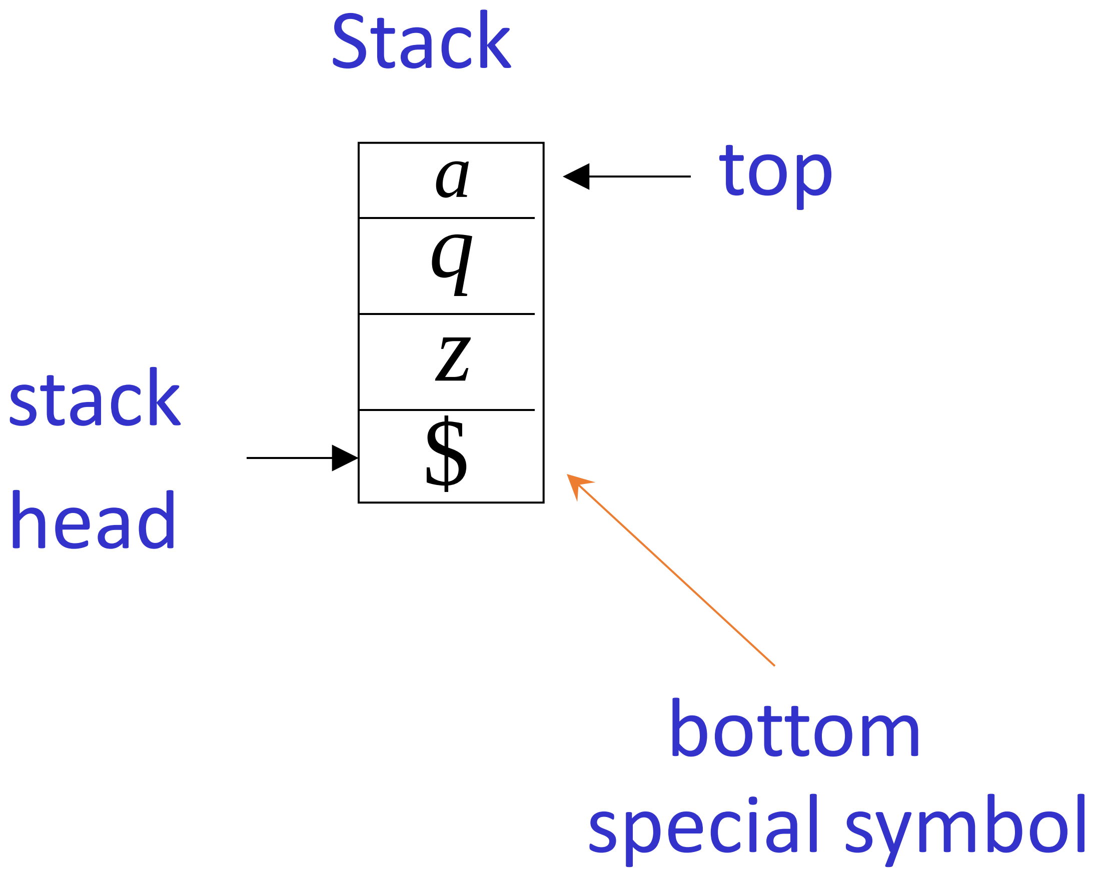
Outline

- Last week
- Context-free Grammars and Definitions
- Simplifications of Context-free Grammars
- Normal forms for Context-free Grammars
- CYK Parser
- Pushdown automata

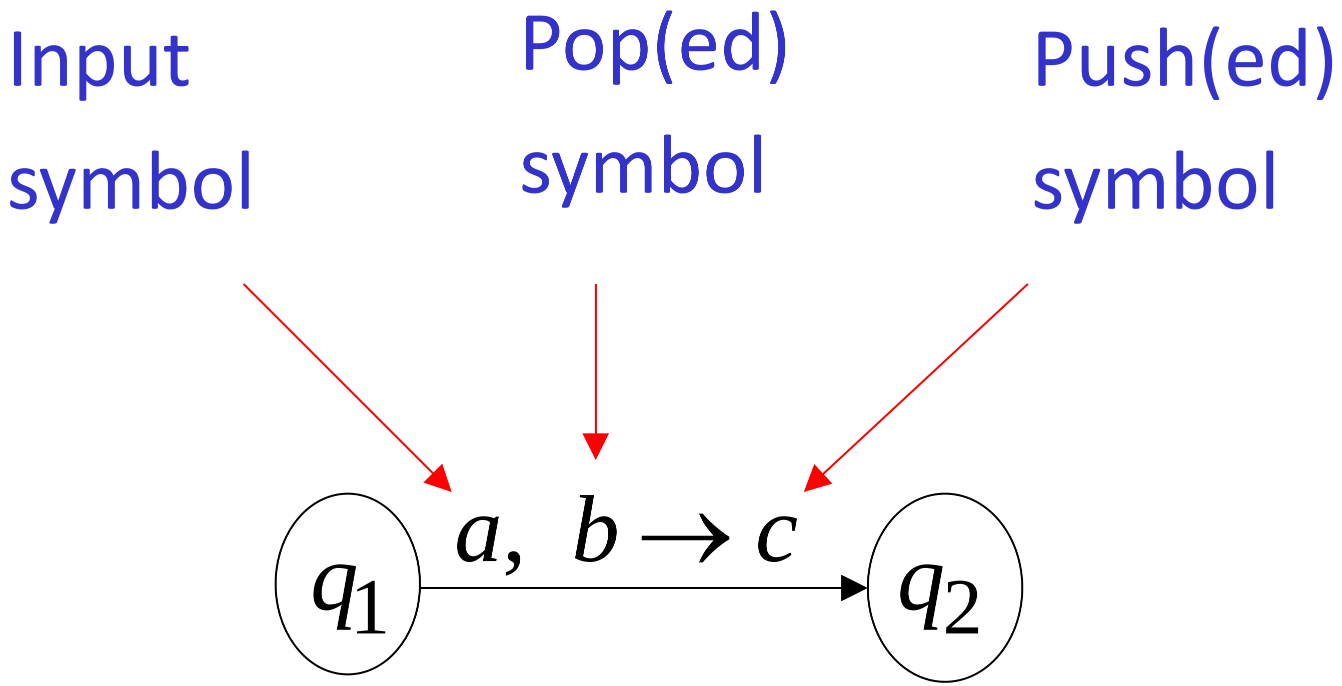
Pushdown Automaton -- PDA

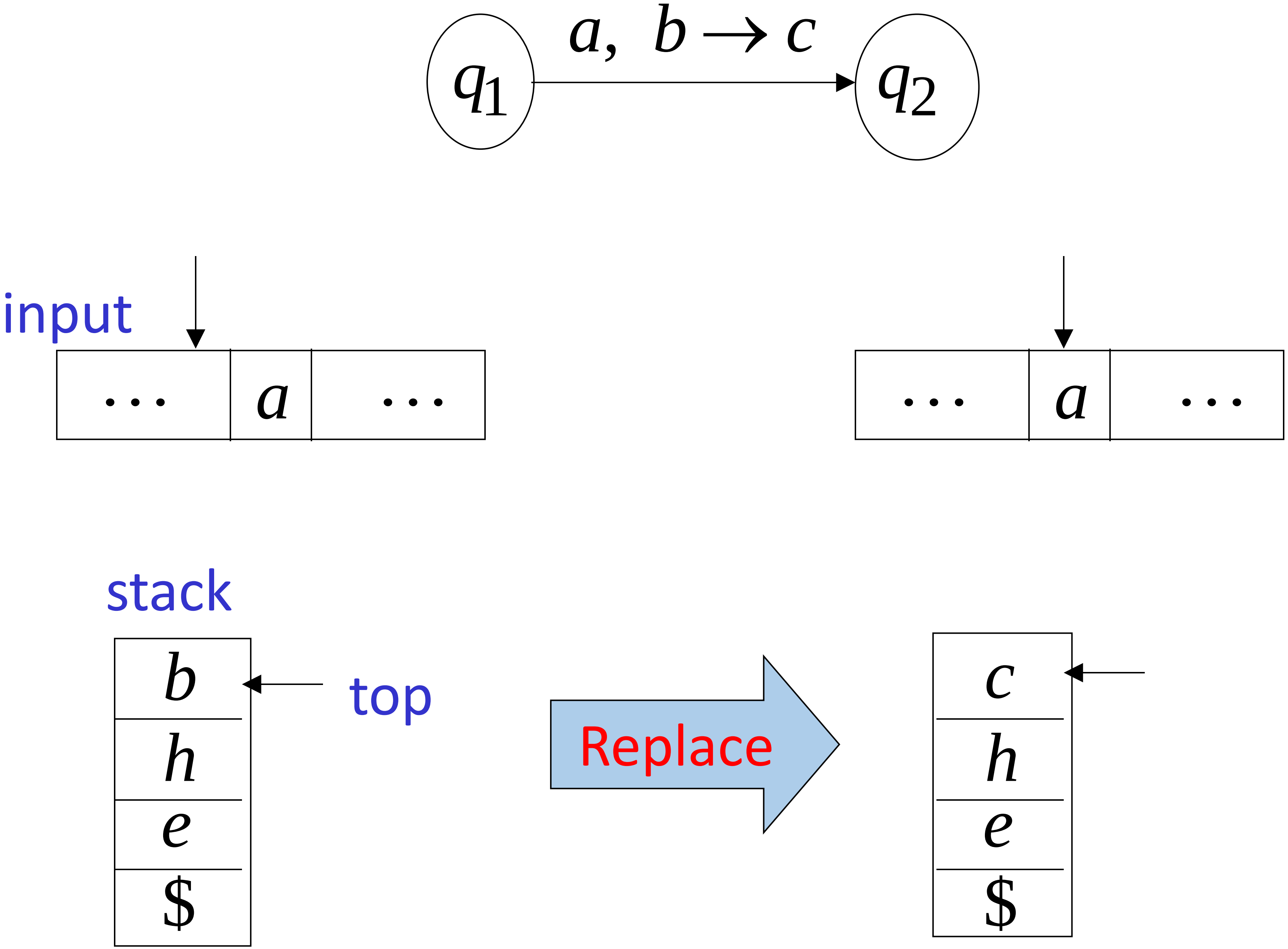


Initial Stack Symbol

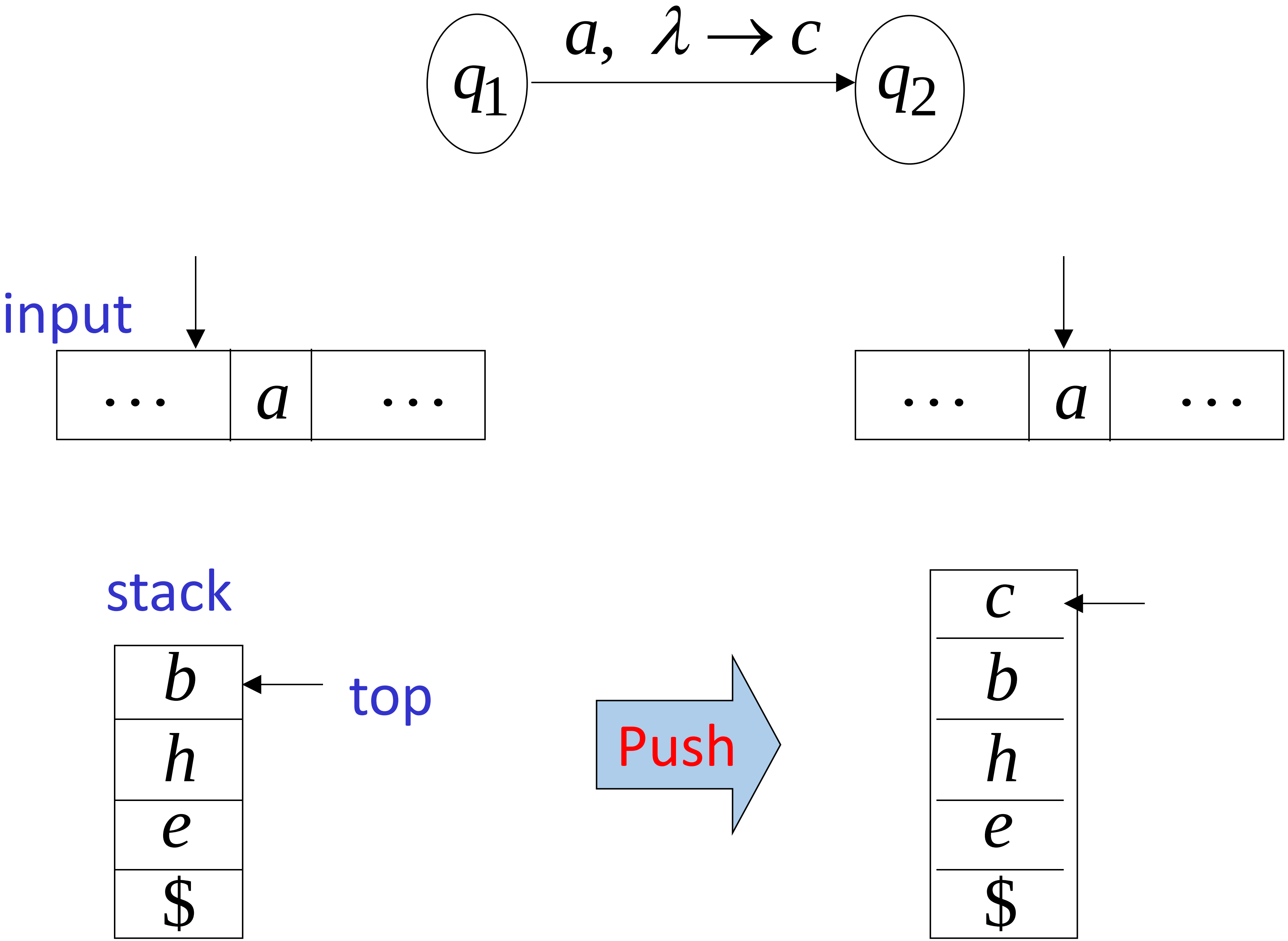


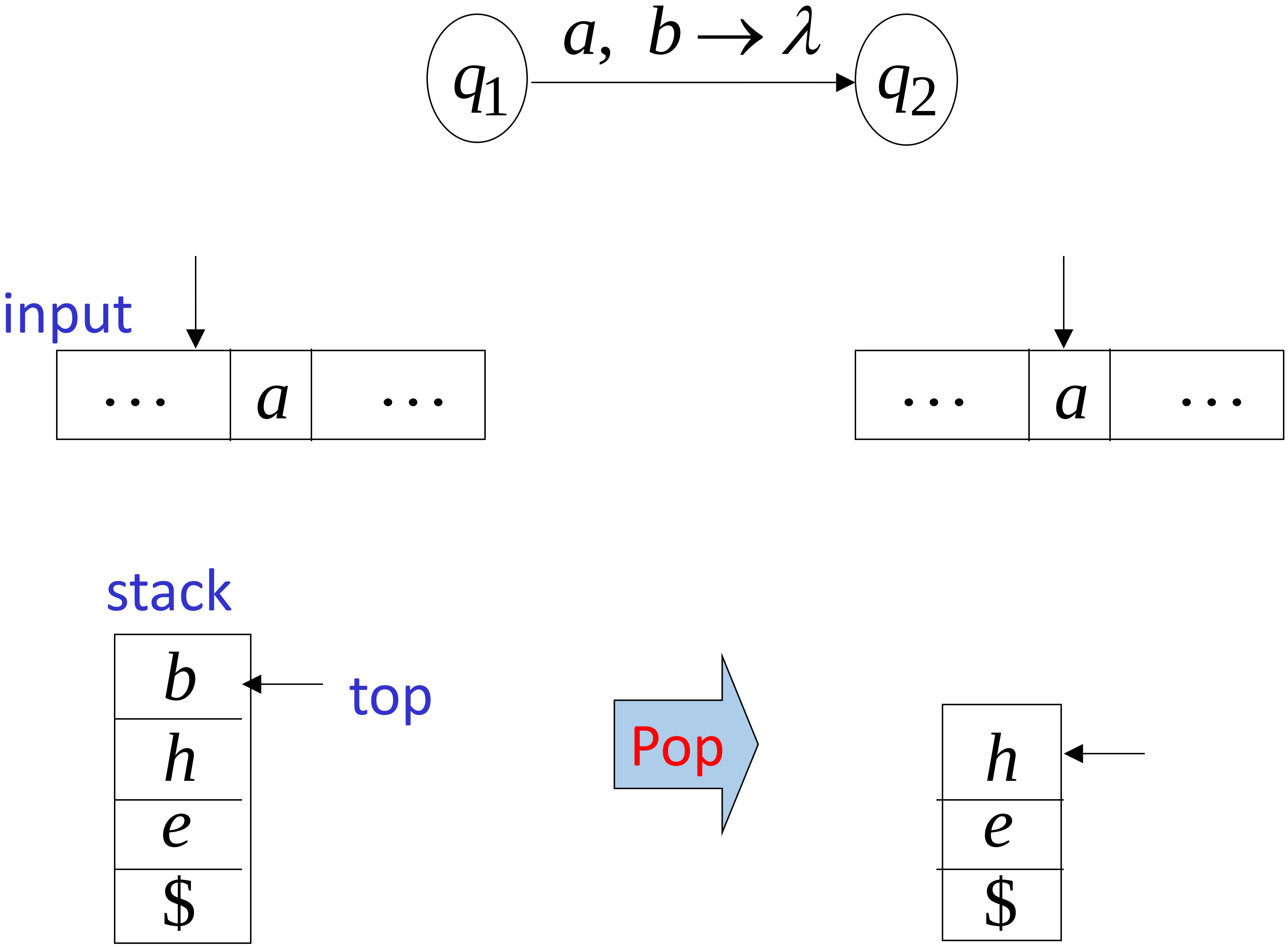
The States

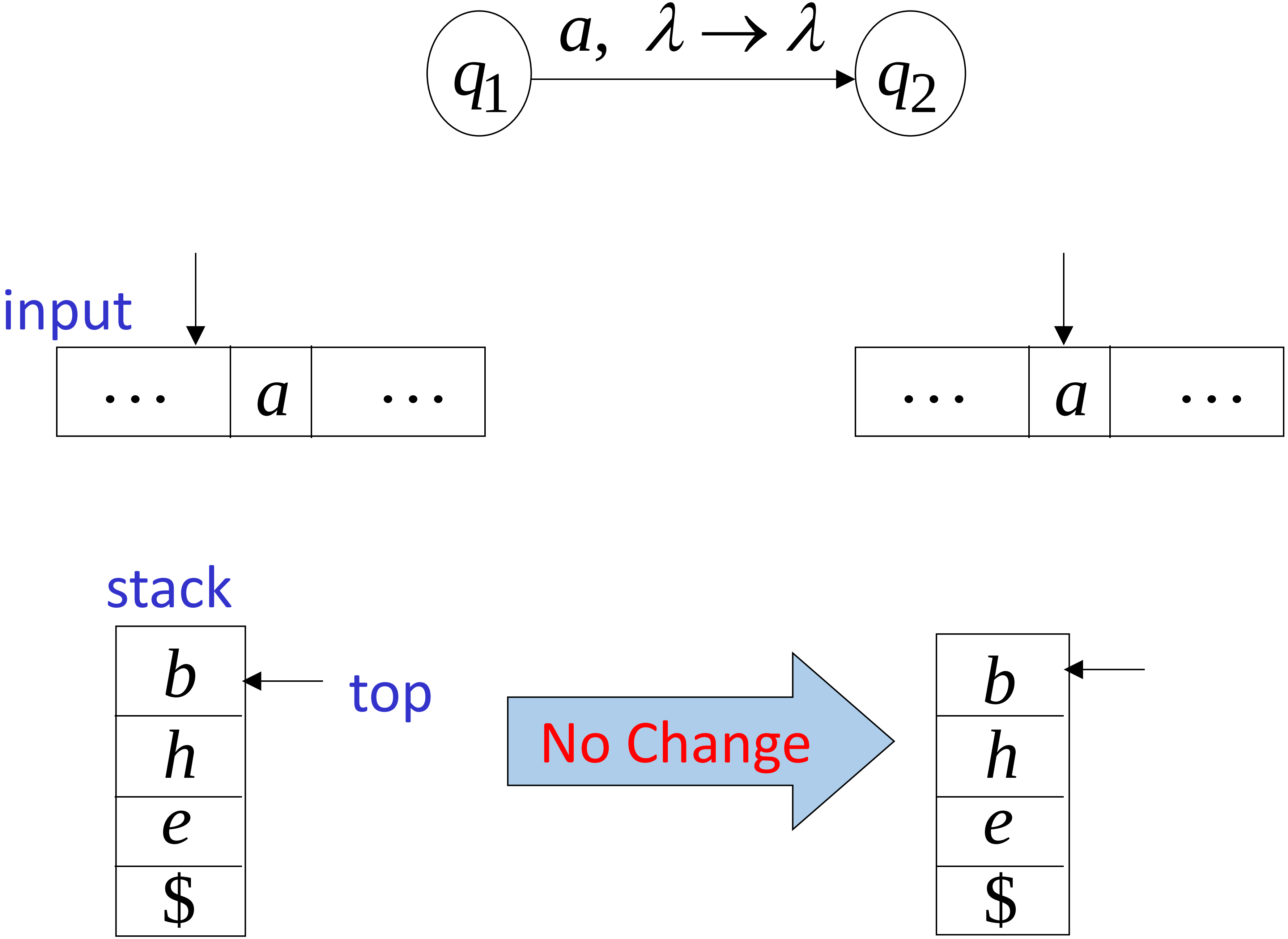




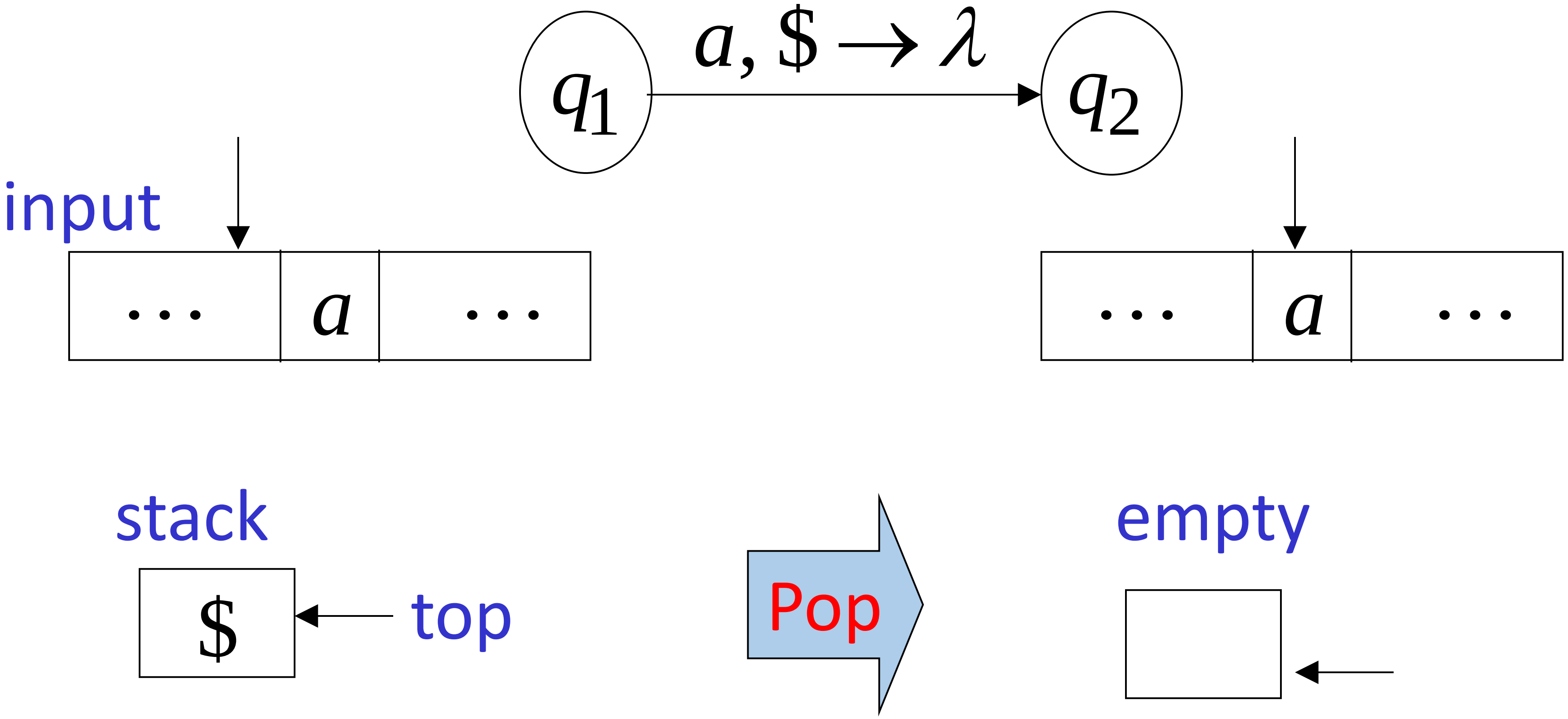
$a, \; b \to c$
q_1
q_2
input
$\cdots$
a
$\cdots$
$\cdots$
a
$\cdots$
stack
b
h
e
$\$$
top
Replace
c
h
e
$\$$


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

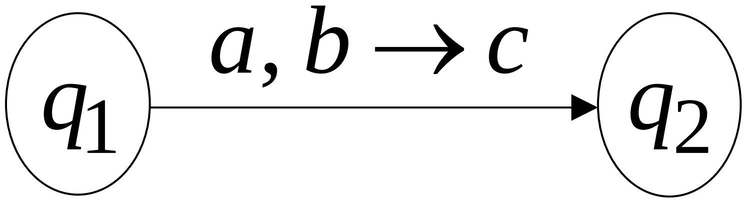

q_1
$a, \ b \to \lambda$
q_2
input
$\cdots$ a $\cdots$
$\cdots$ a $\cdots$
stack
b
h
e
$\$$
top
Pop
h
e
$\$$


q_1
$a, \lambda \rightarrow \lambda$
q_2
input
$\cdots$ a $\cdots$
$\cdots$ a $\cdots$
stack
b
h
e
$\$$
top
No Change
b
h
e
$\$$

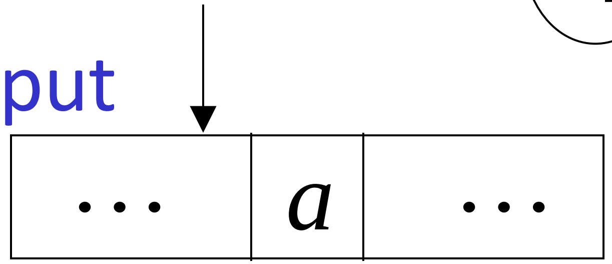
A Possible Transition



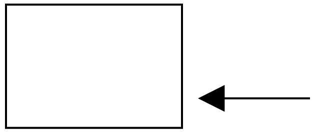
A Bad Transition



input



Empty stack

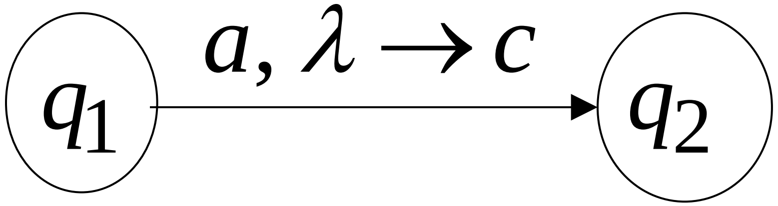


HALT

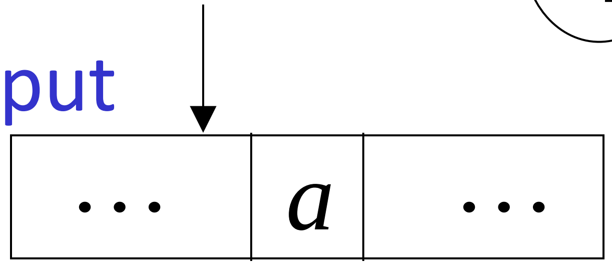
The automaton **Halts** in state q_1

and **Rejects** the input string

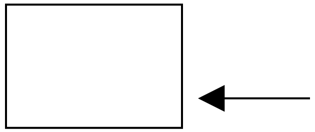
A Bad Transition

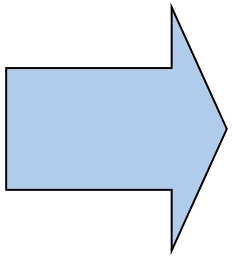


input



Empty stack





HALT

The automaton **Halts** in state q_1

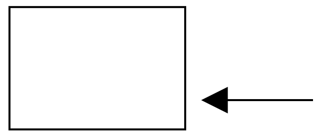
and **Rejects** the input string

No transition is allowed to be followed

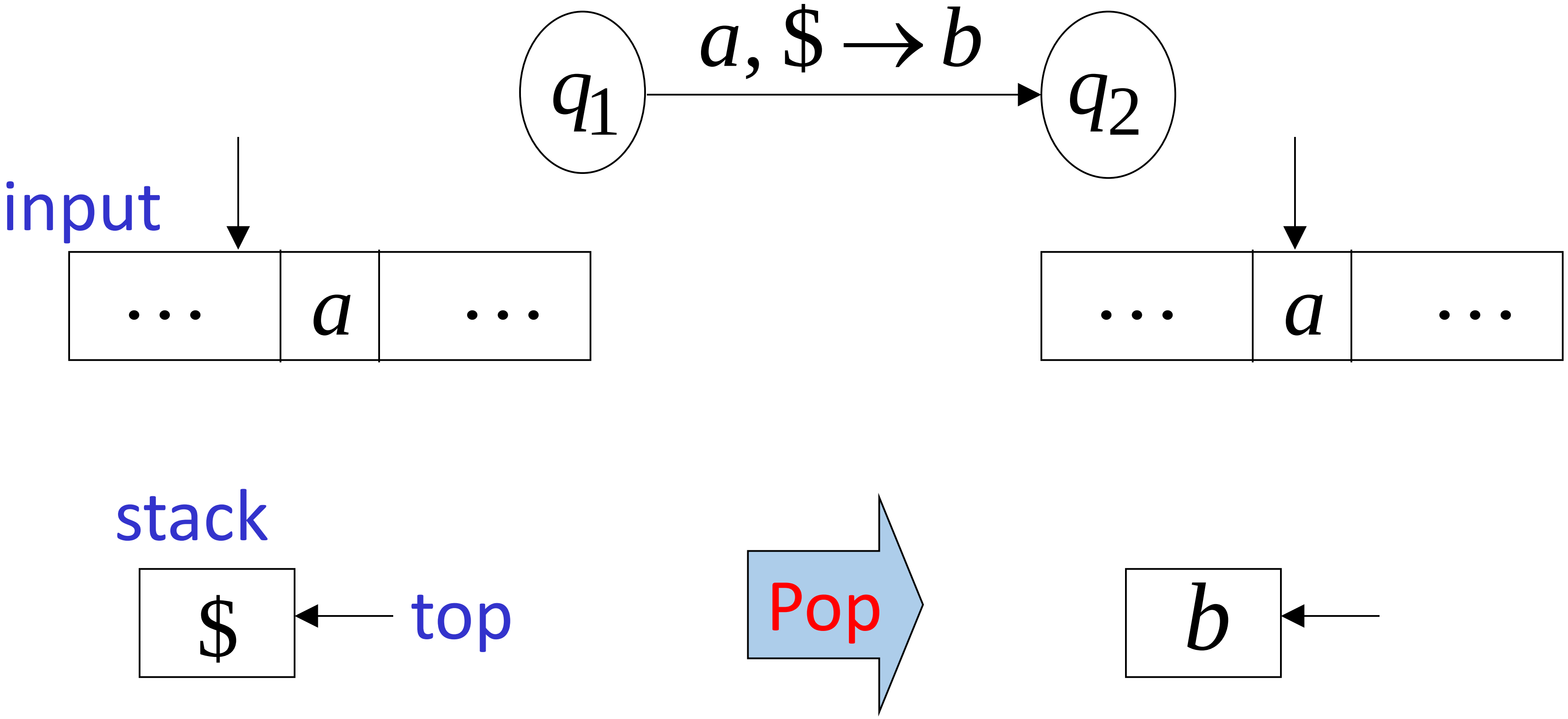
When the stack is empty

$$q_1 \xrightarrow{x,\ y \to z} q_2$$

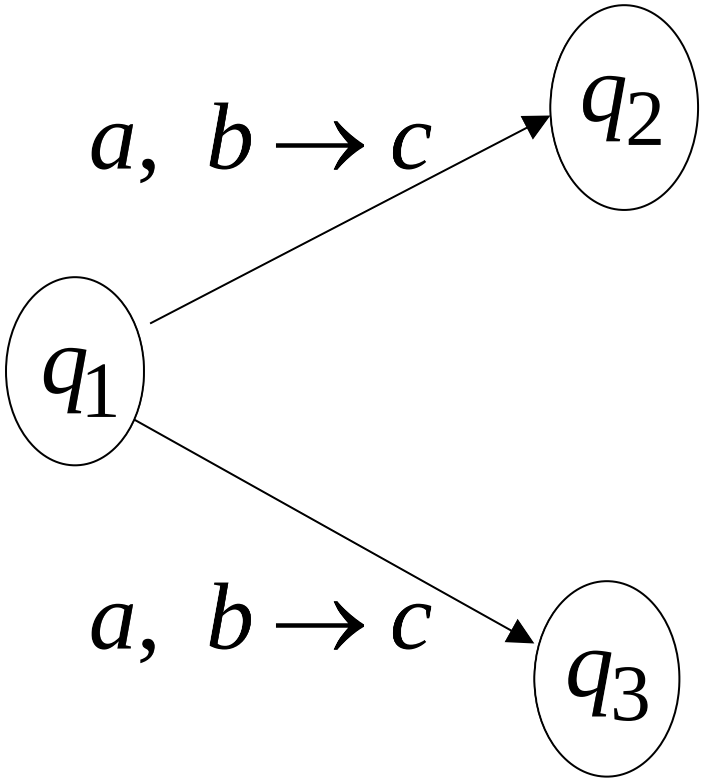
Empty stack

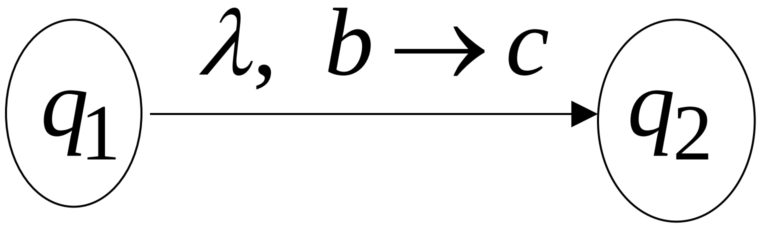


A Good Transition



Non-Determinism





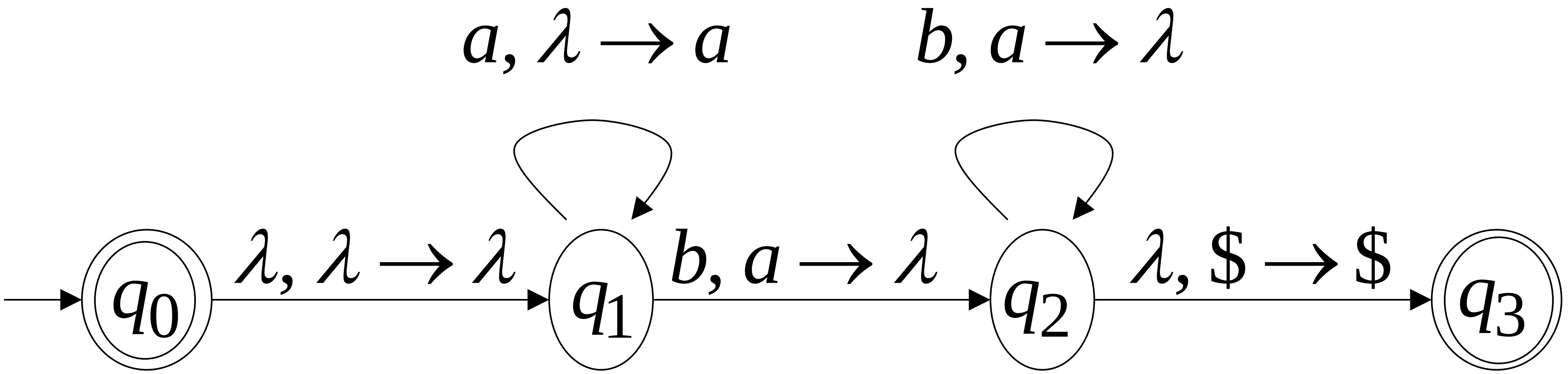
$\lambda -$ transition

These are allowed transitions in a

Non-deterministic PDA (NPDA)

NPDA: Non-Deterministic PDA

Example:



Execution Example: Time 0

Input

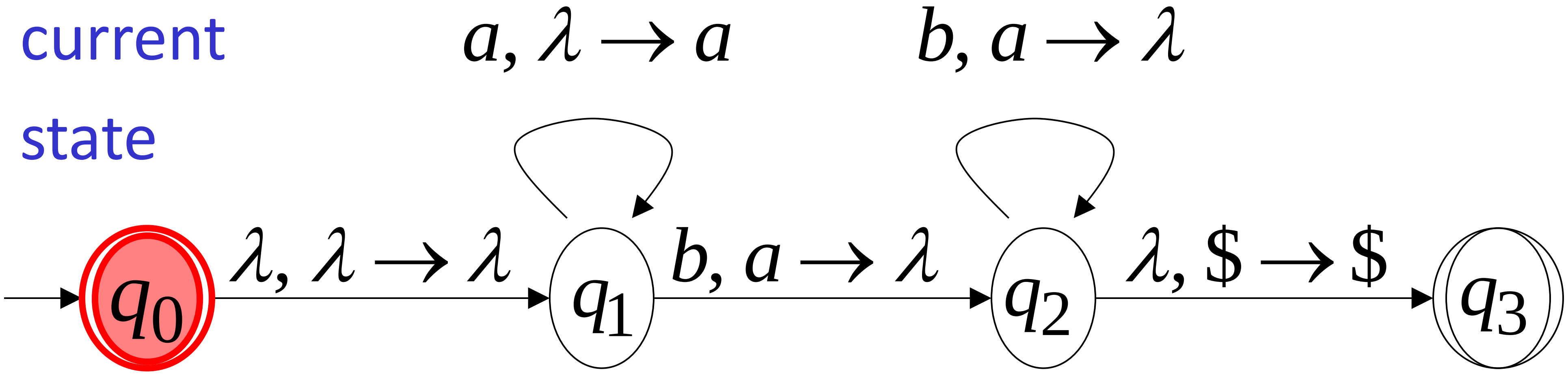
a	a	a	b	b	b

$\$$

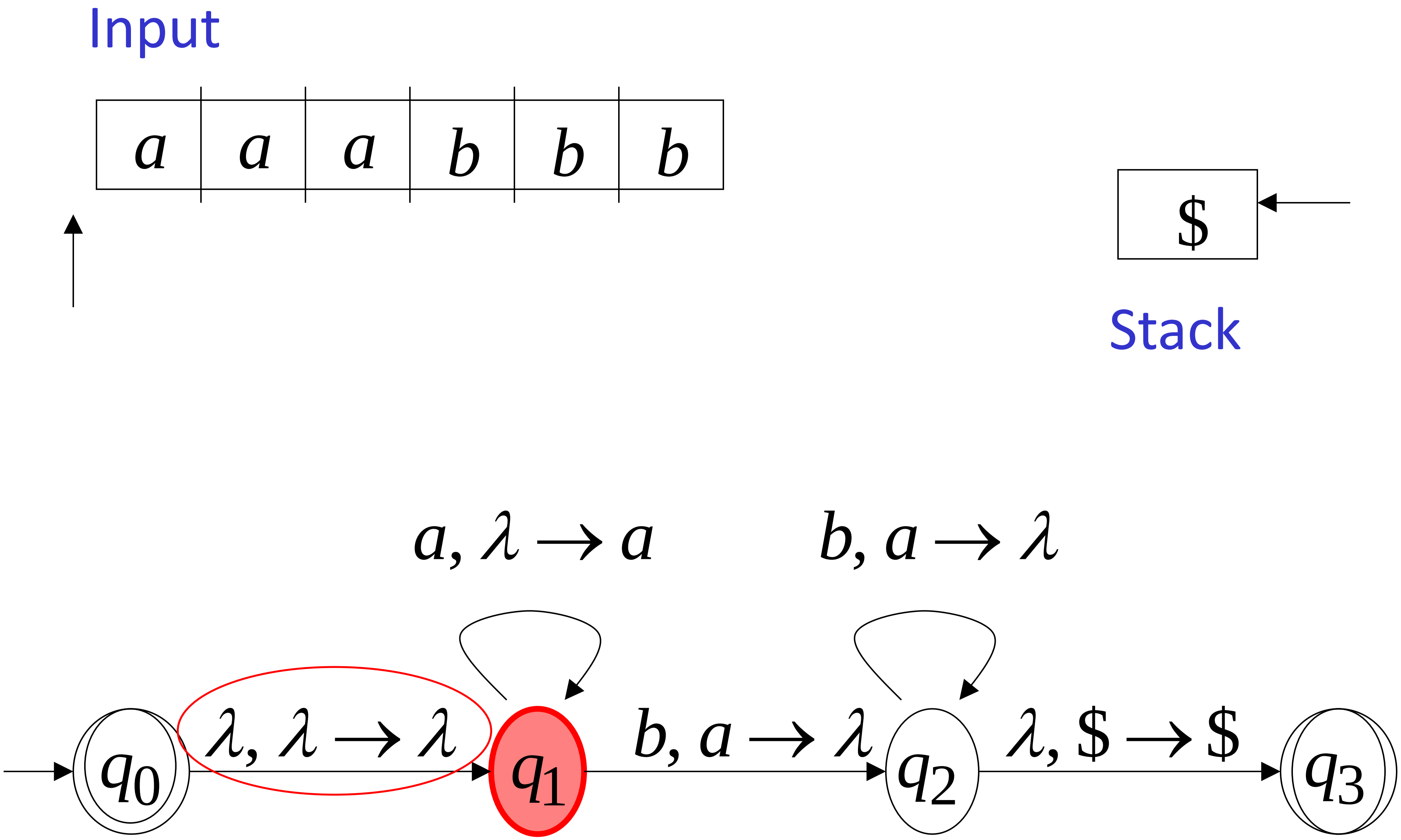
Stack

current

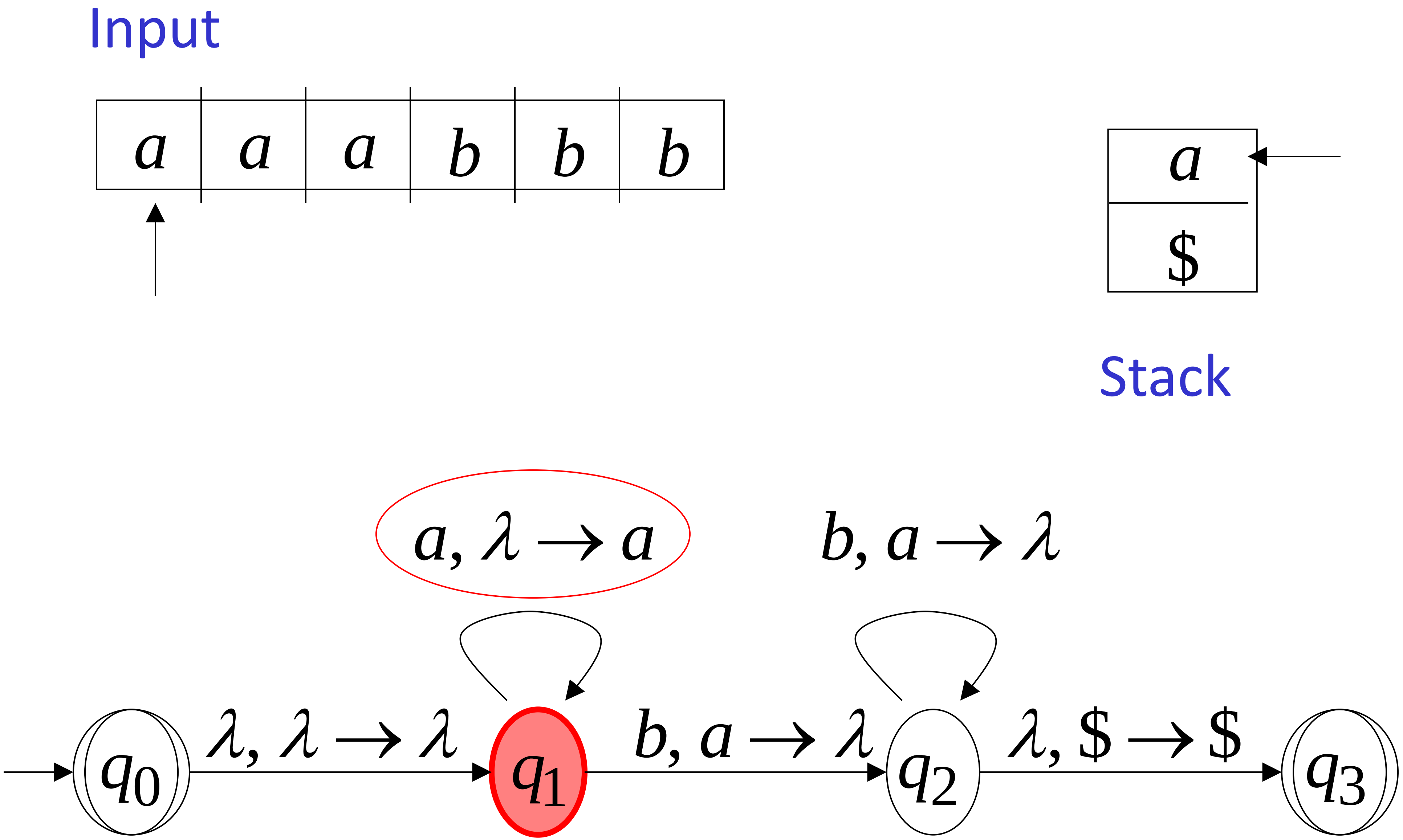
state

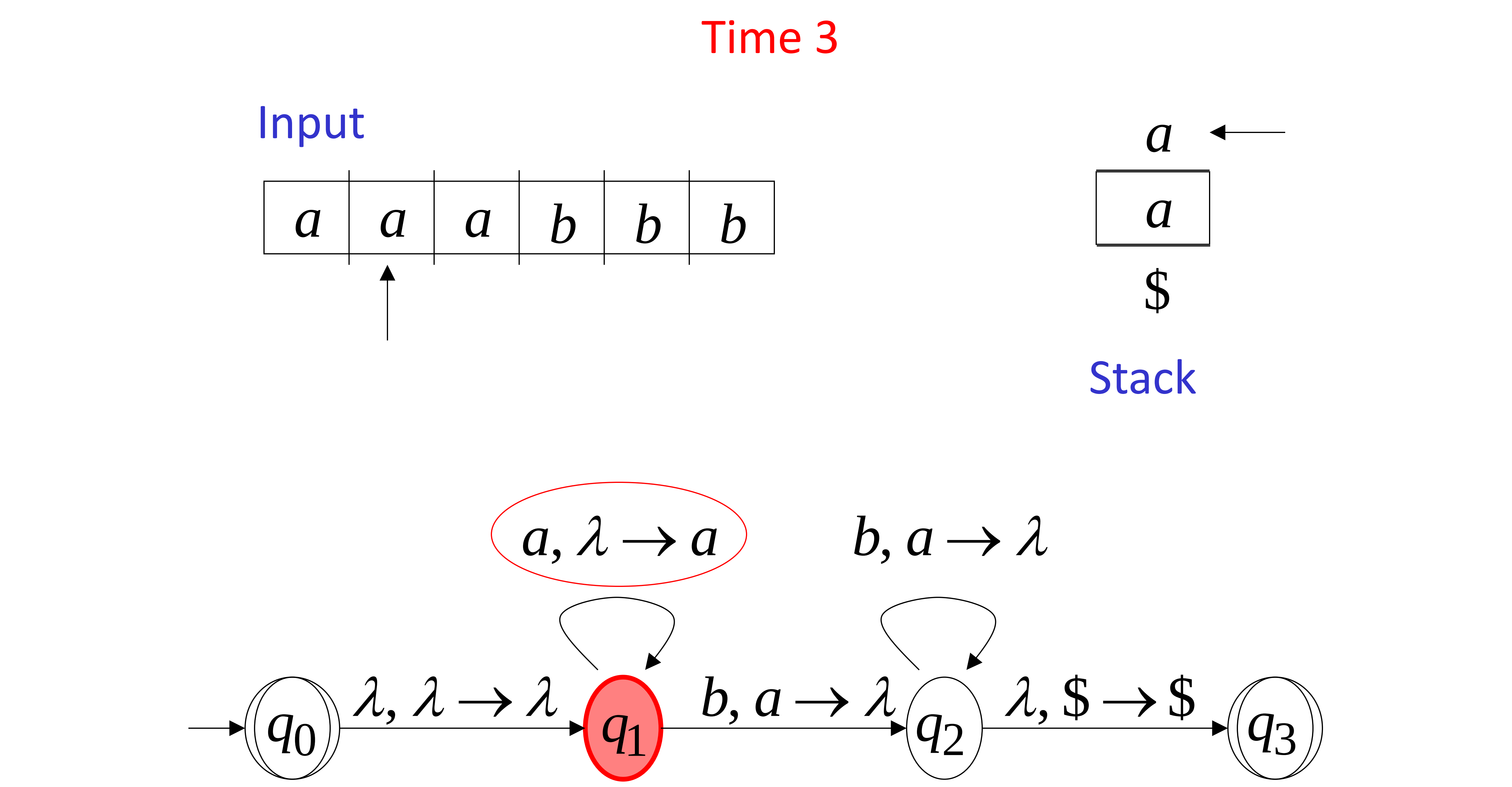


Time 1



Time 2


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 3
Input
a a a b b b
a
a
$
Stack
$a, \lambda \rightarrow a$
$b, a \rightarrow \lambda$
q_0
$\lambda, \lambda \rightarrow \lambda$
q_1
$b, a \rightarrow \lambda$
q_2
$\lambda, \$ \rightarrow \$$
q_3

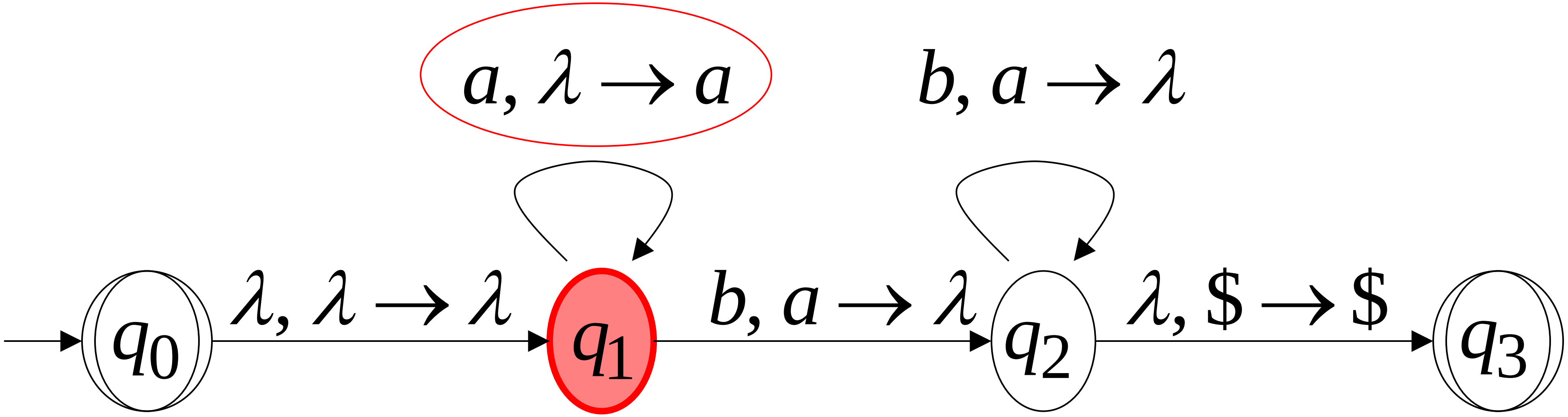
Time 4

Input

a	a	a	b	b	b

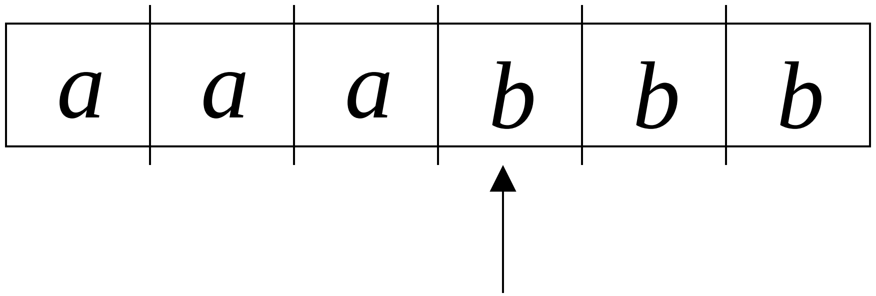
a
a
a
$\$$

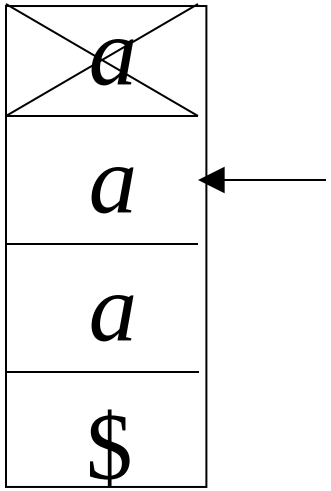
Stack



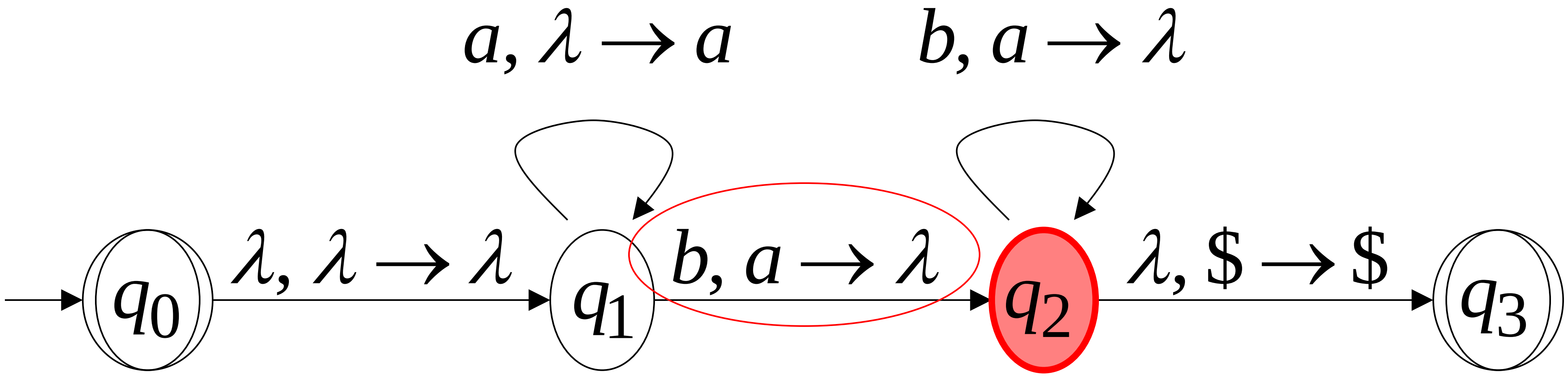
Time 5

Input



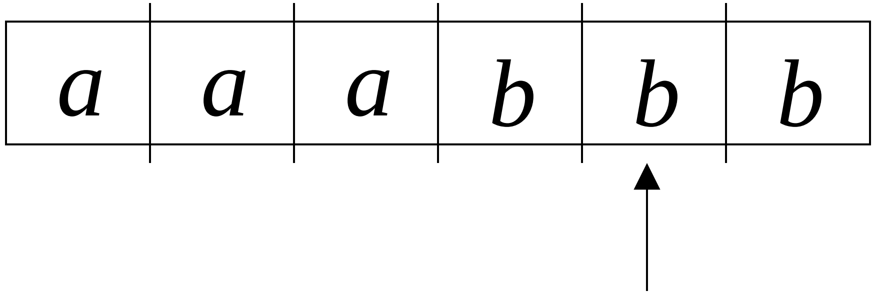


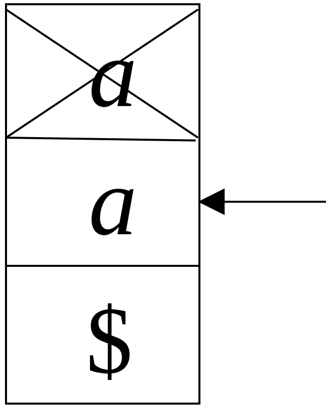
Stack



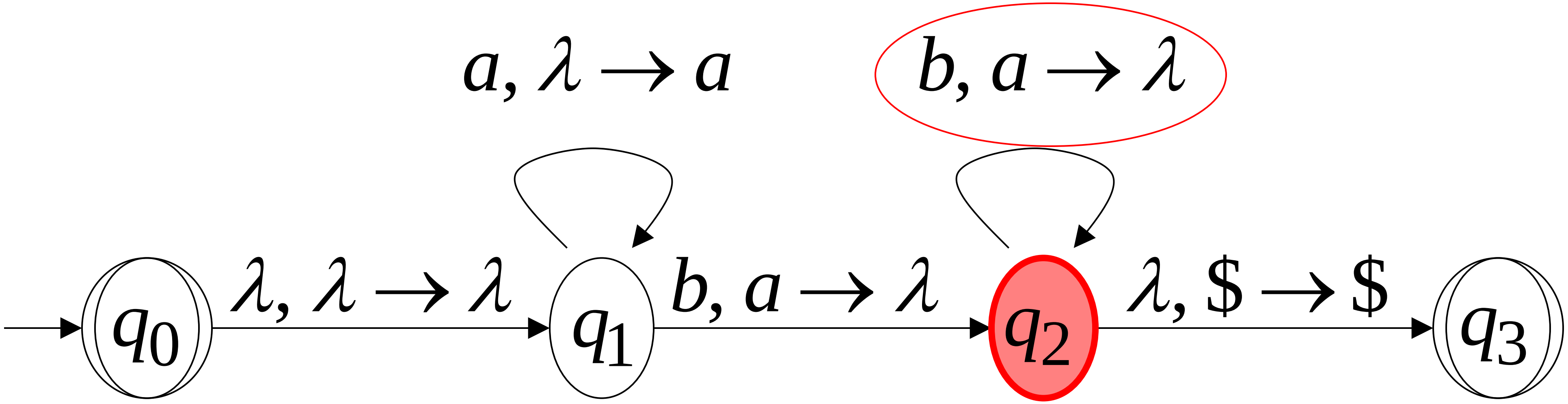
Time 6

Input

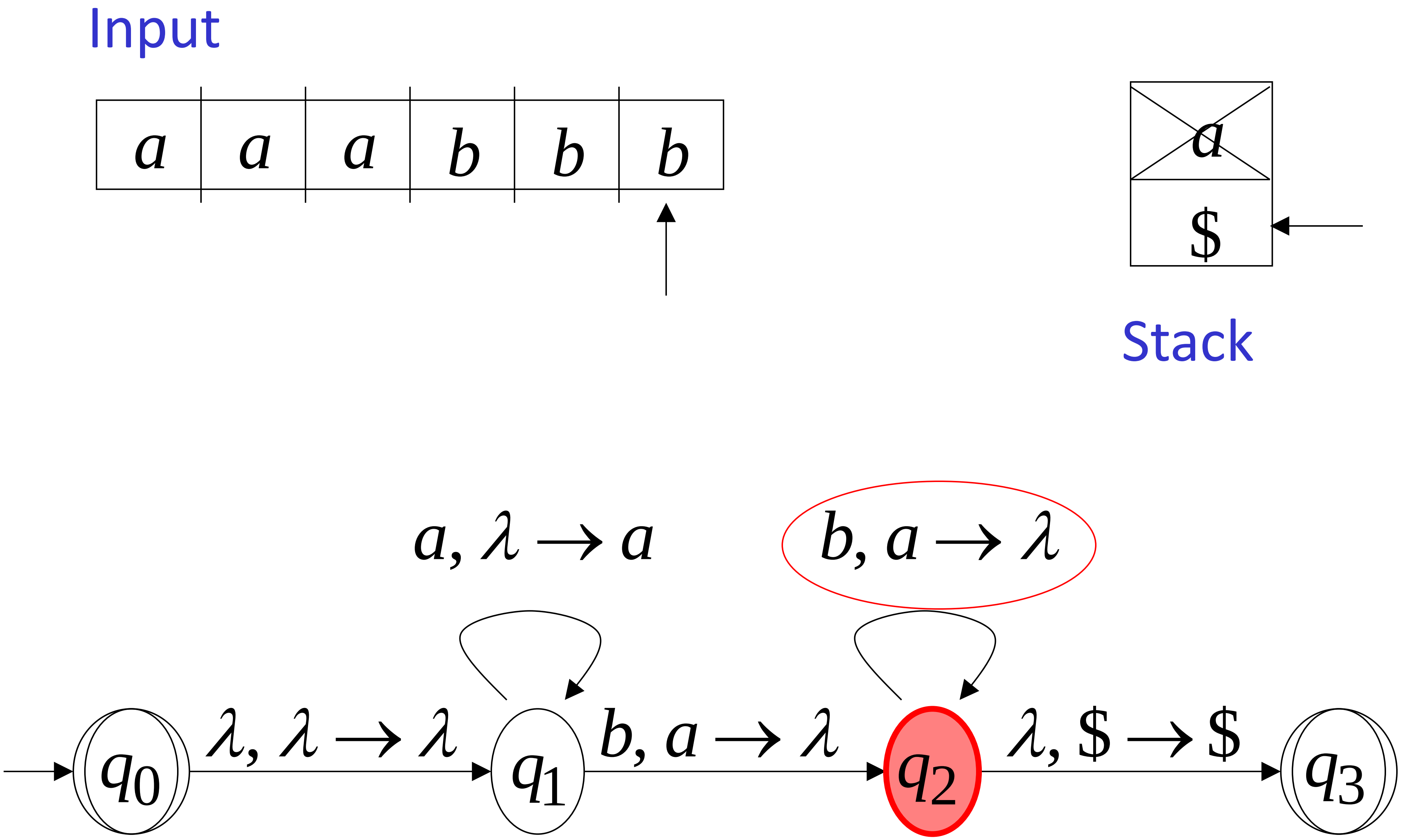




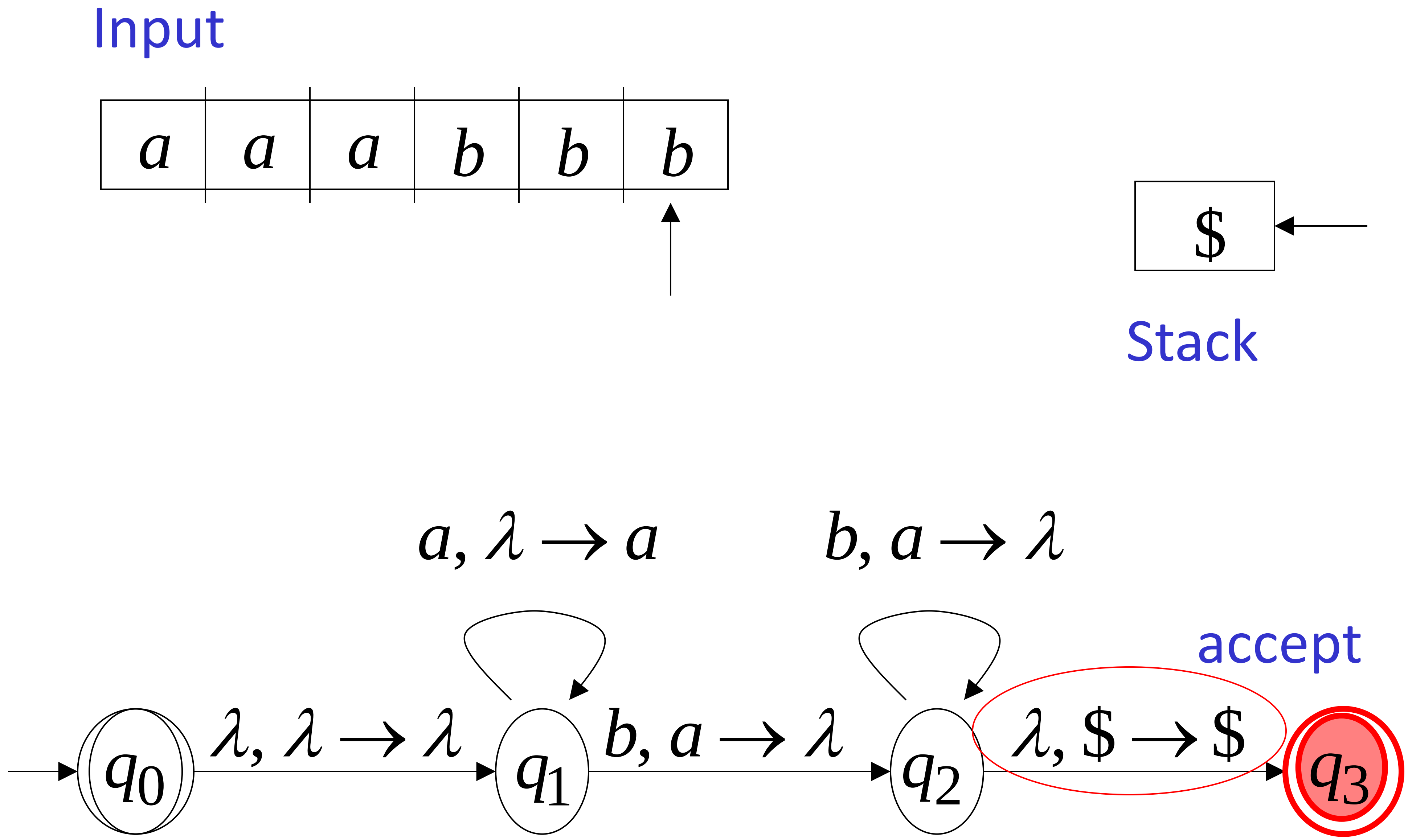
Stack



Time 7


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

Time 8


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

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

All the input is consumed

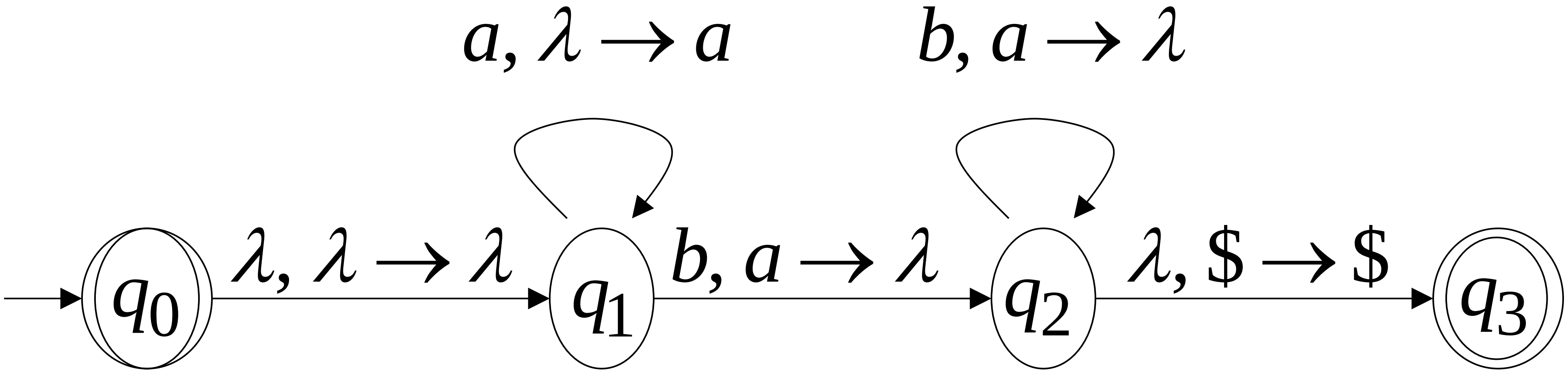
AND

The last state is a final state

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

The input string $aaabbb$

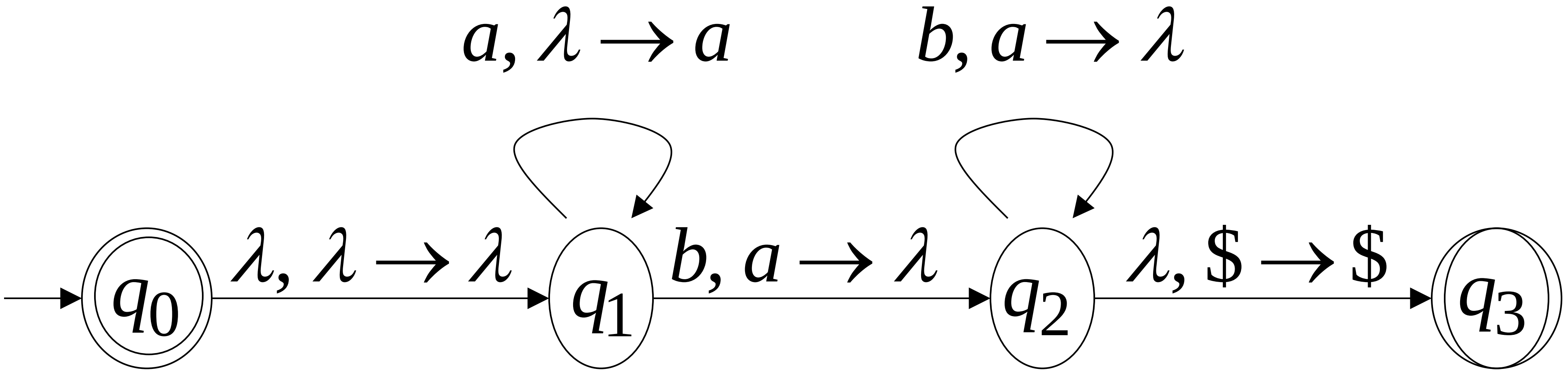
is accepted by the NPDA:



In general,

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

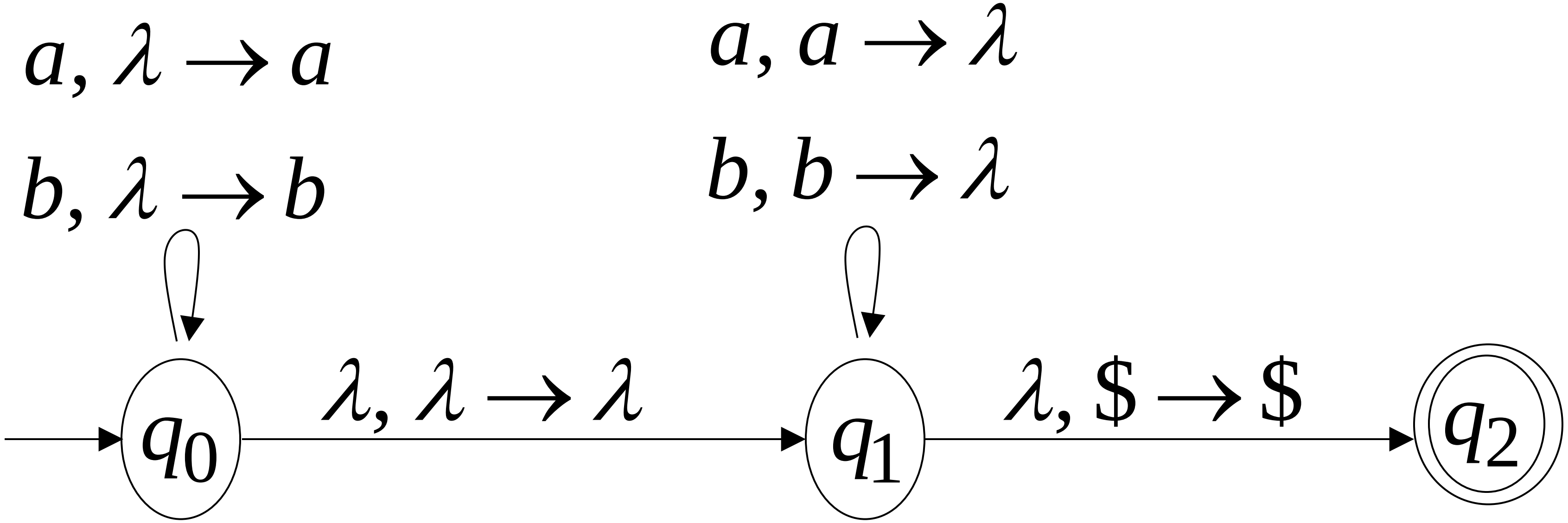
is the language accepted by the NPDA:



Another NPDA example

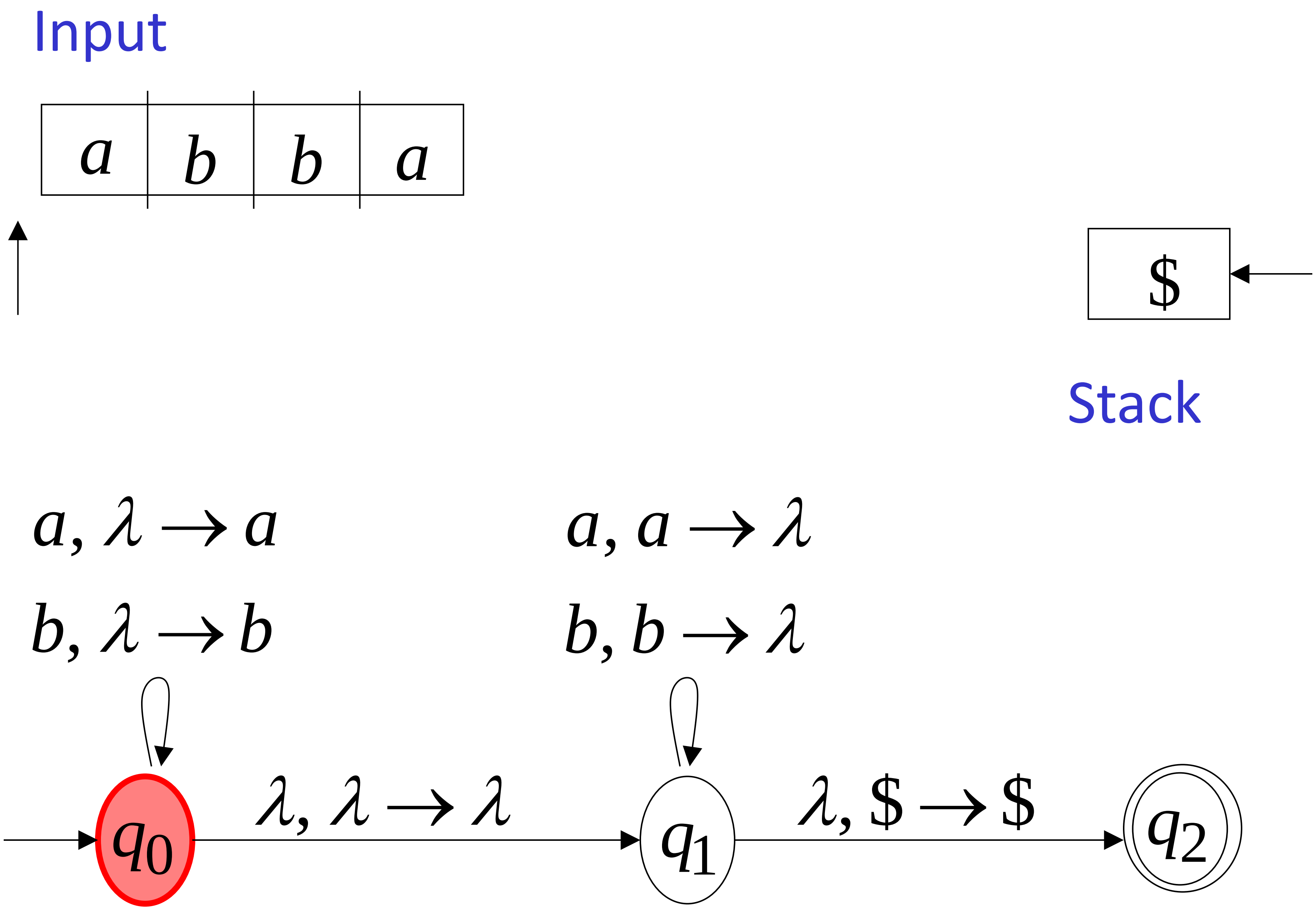
NPDA M

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



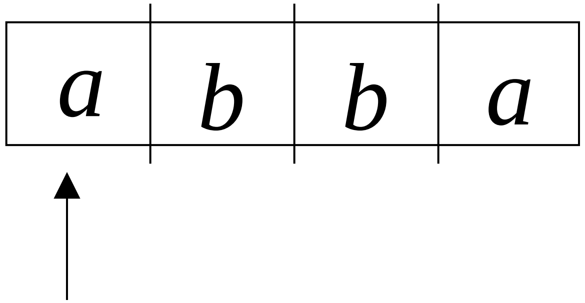
Execution Example:

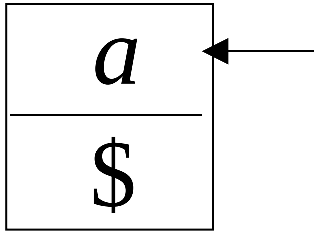
Time 0



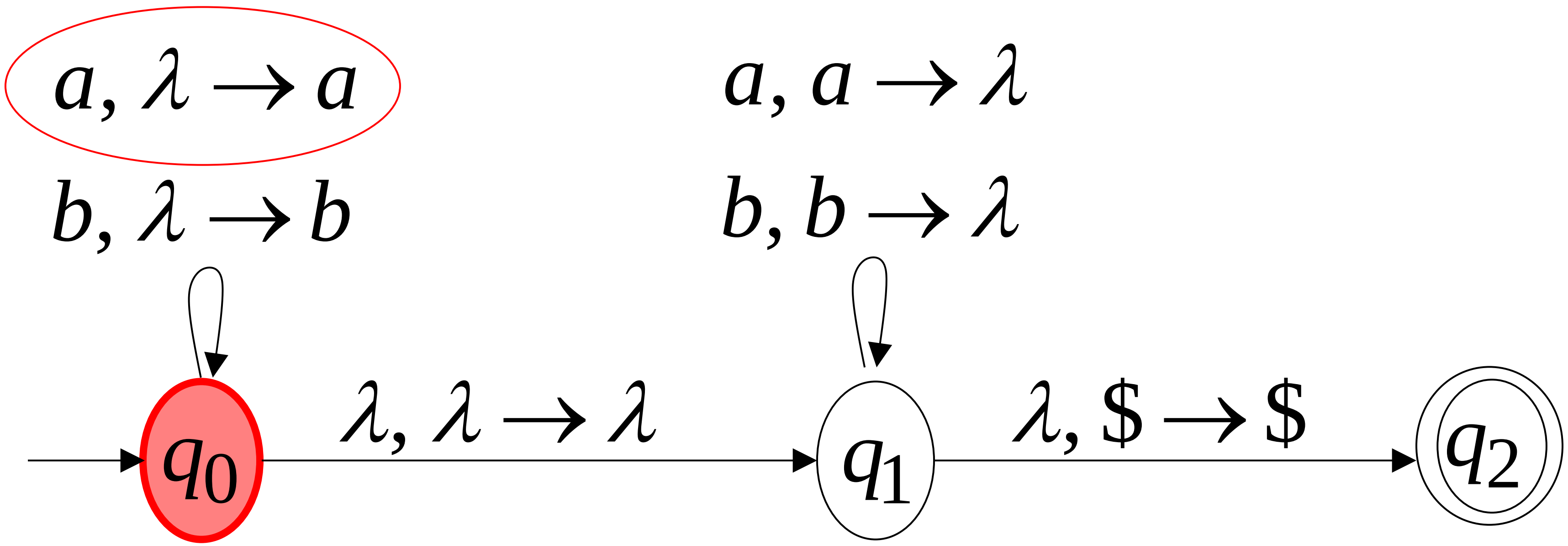
Time 1

Input

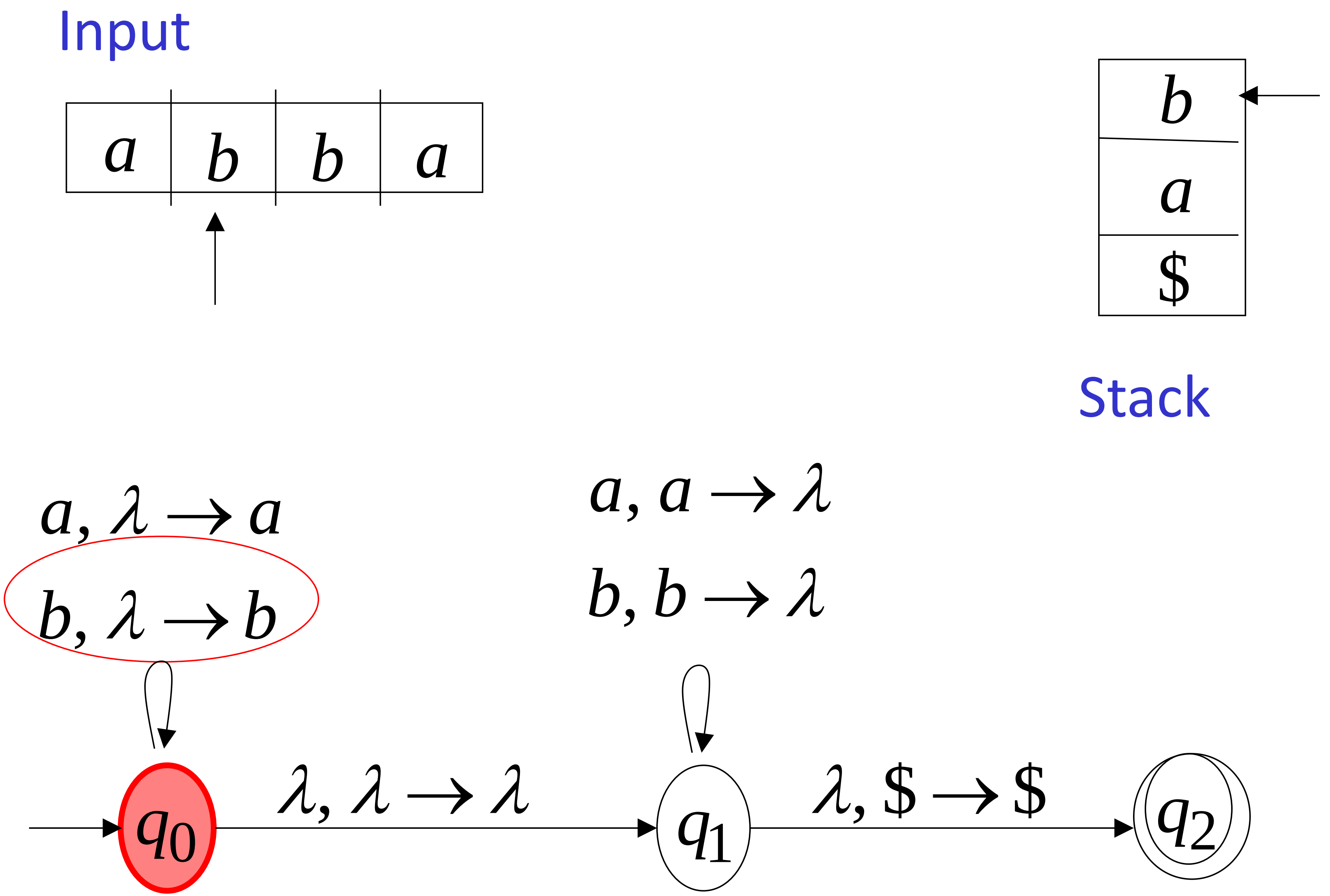




Stack

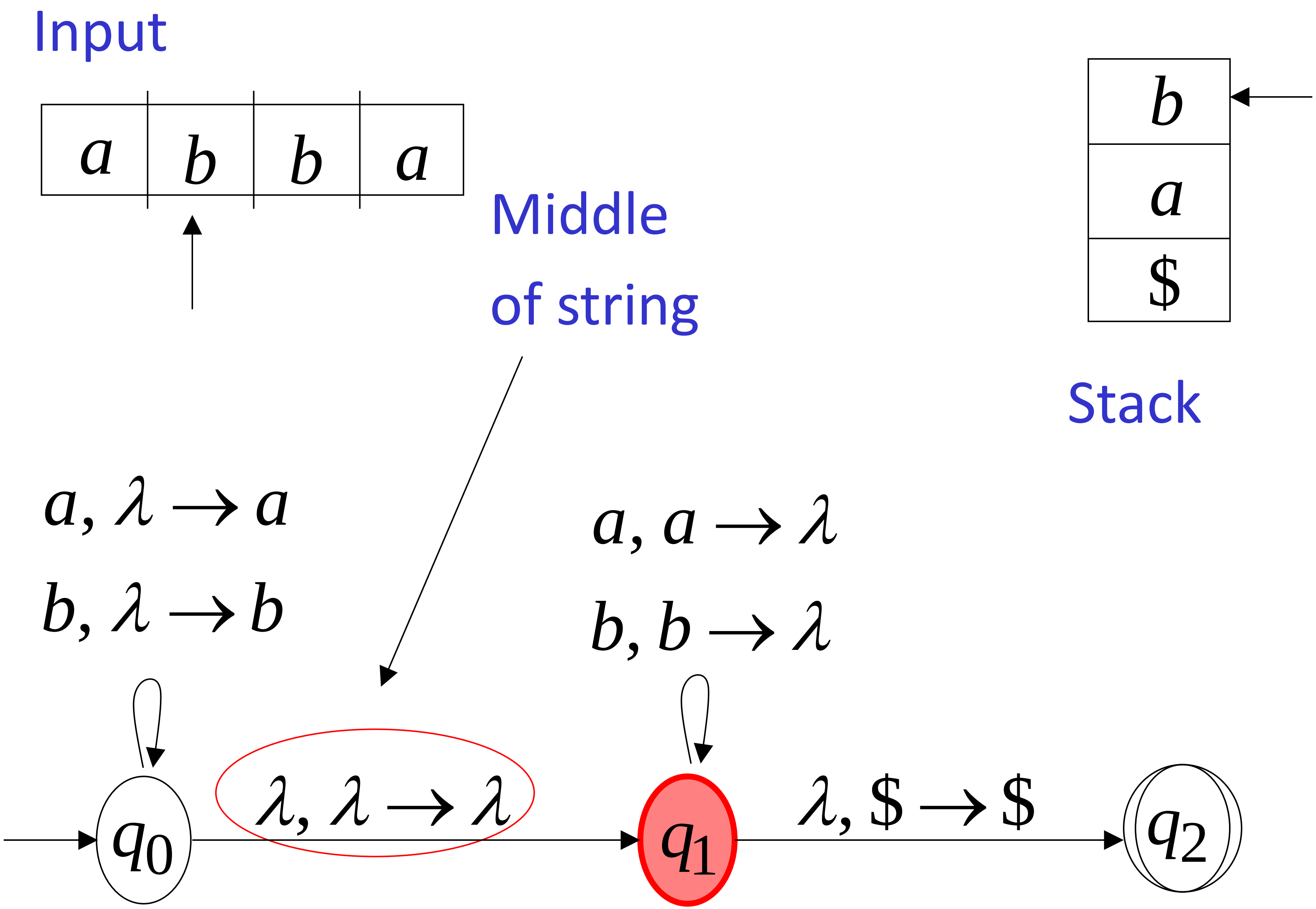


Time 2



Input
a
b
b
a
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

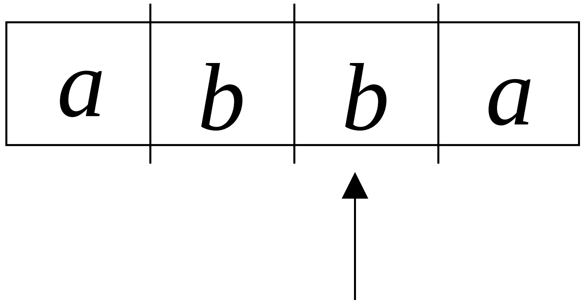
Time 3

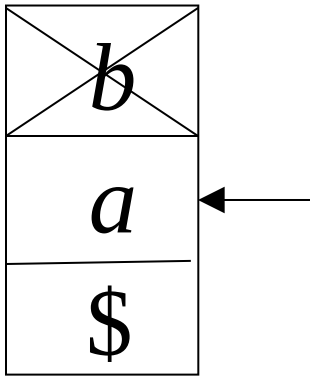


Input
a b b a
Middle of string
b
a
$
Stack
$a, \lambda \rightarrow a$
$b, \lambda \rightarrow b$
$a, a \rightarrow \lambda$
$b, b \rightarrow \lambda$
q_0
$\lambda, \lambda \rightarrow \lambda$
q_1
$\lambda, \$ \rightarrow \$$
q_2

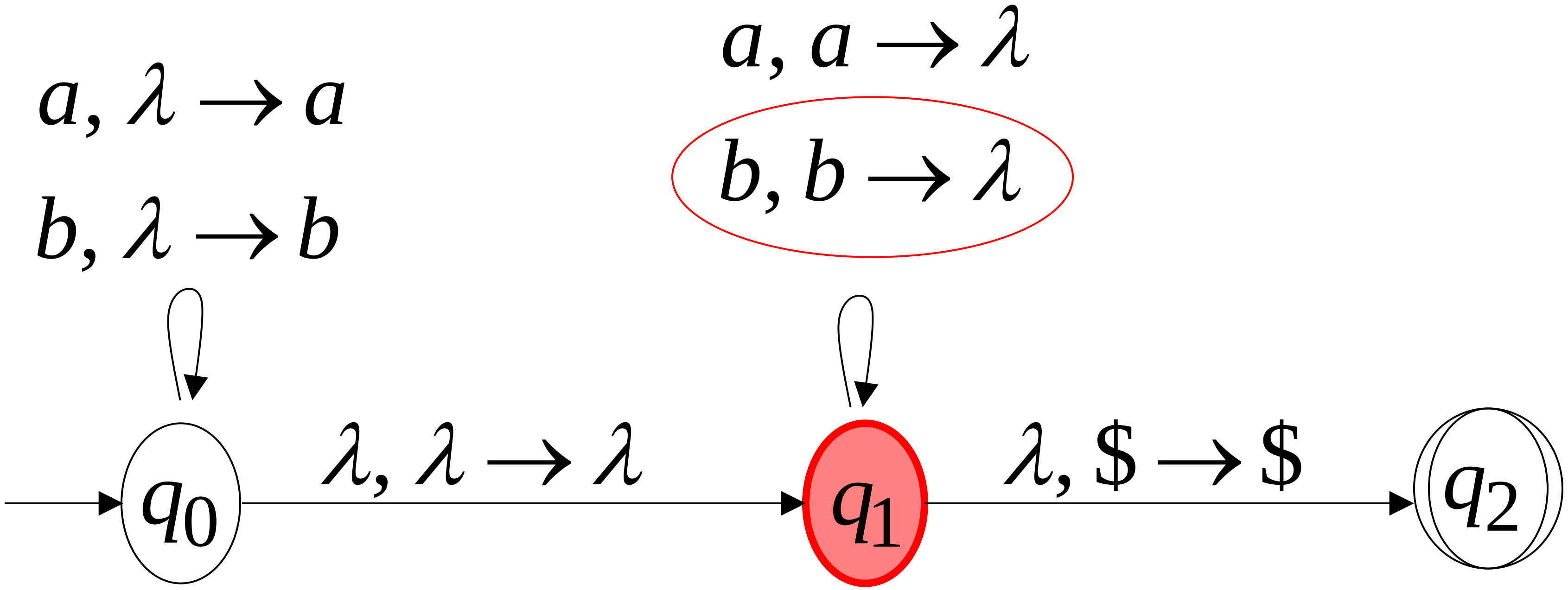
Time 4

Input



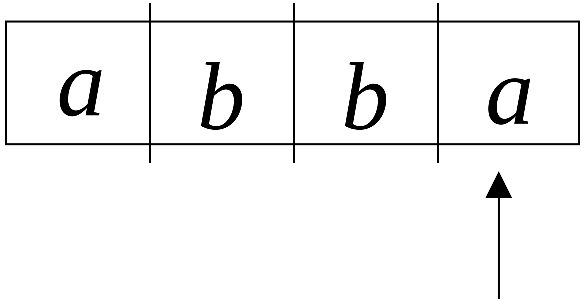


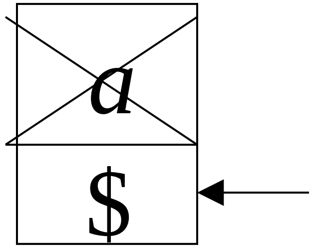
Stack



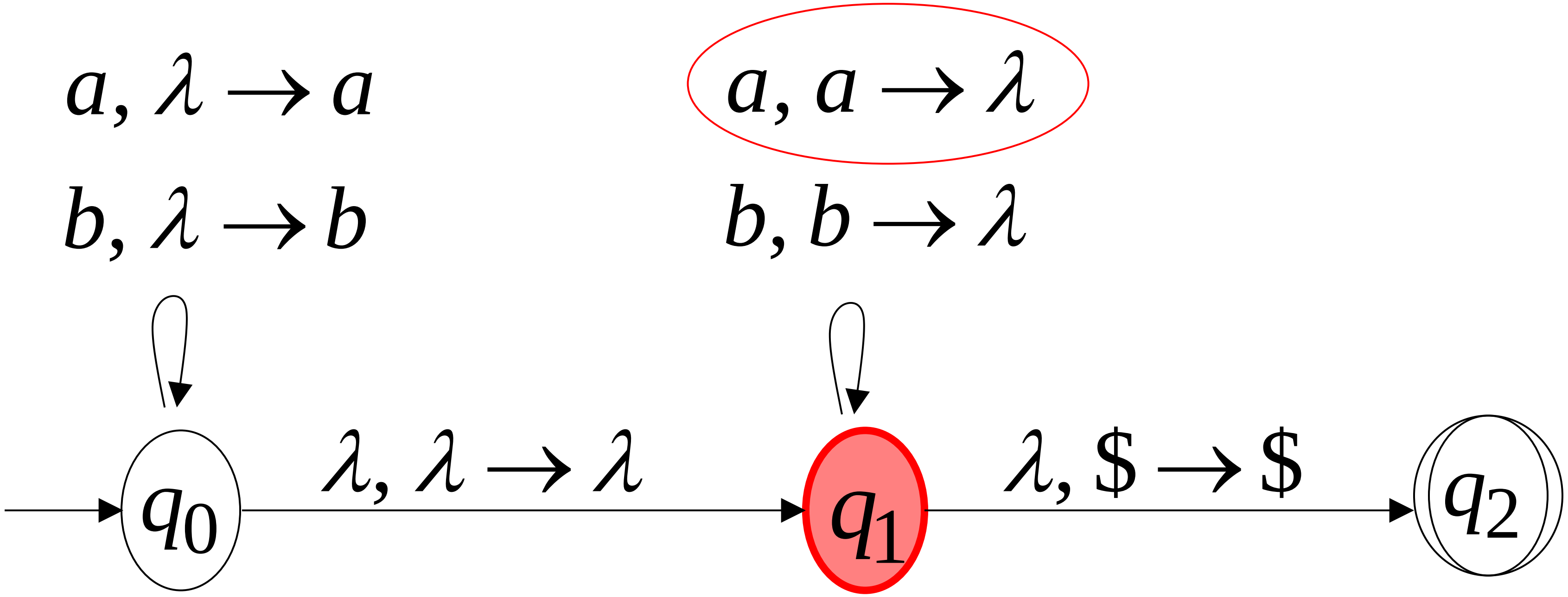
Time 5

Input

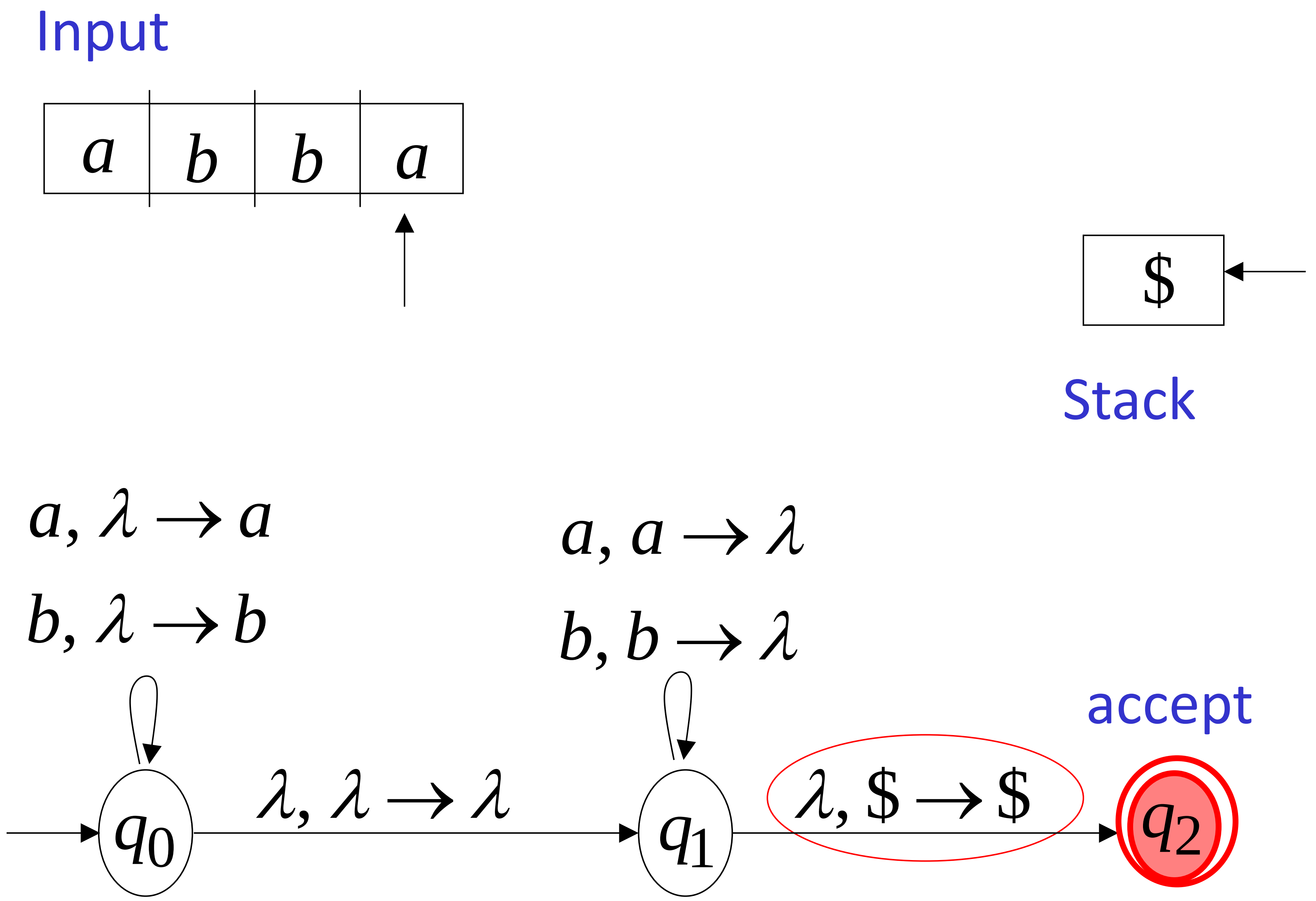




Stack

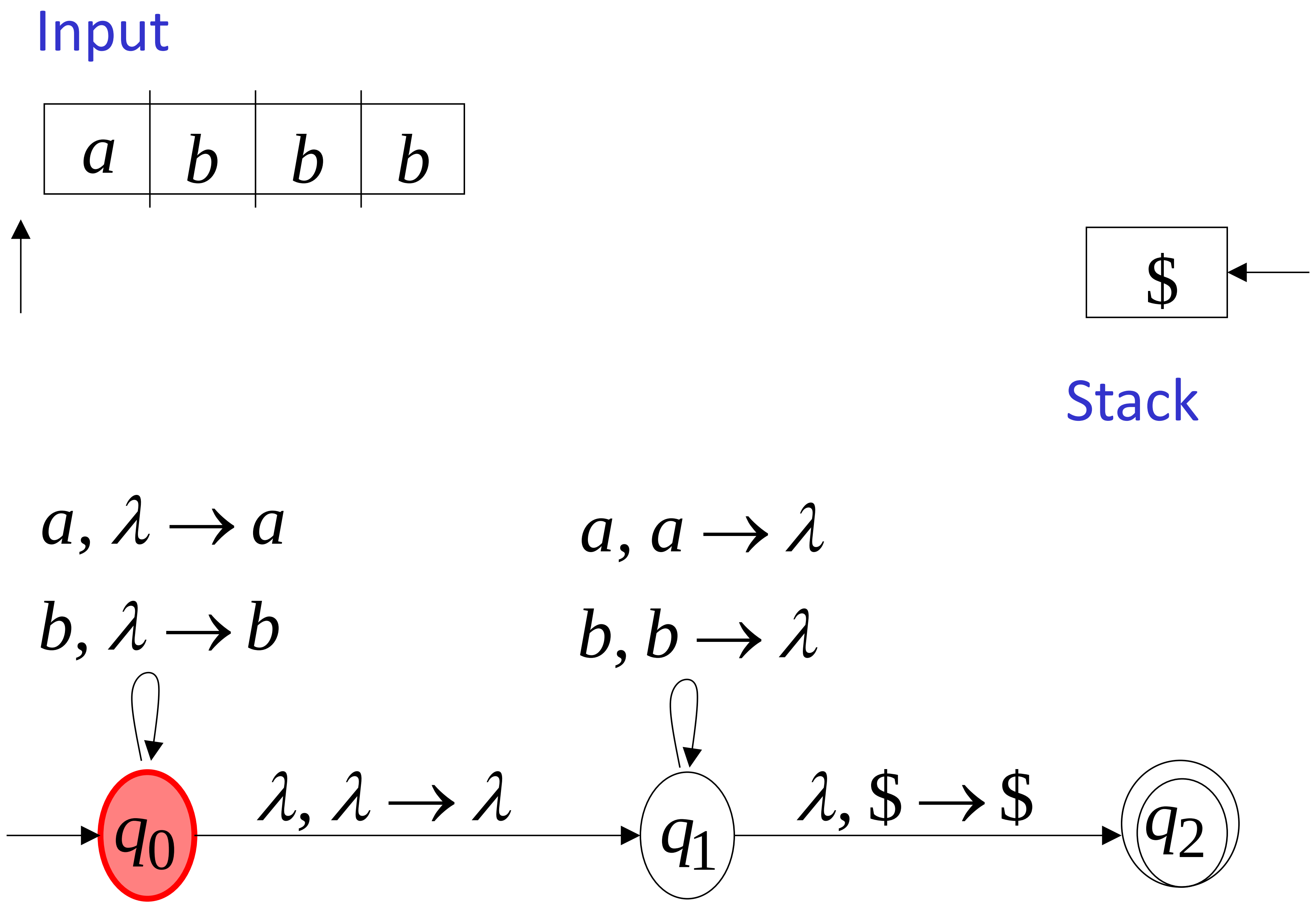


Time 6

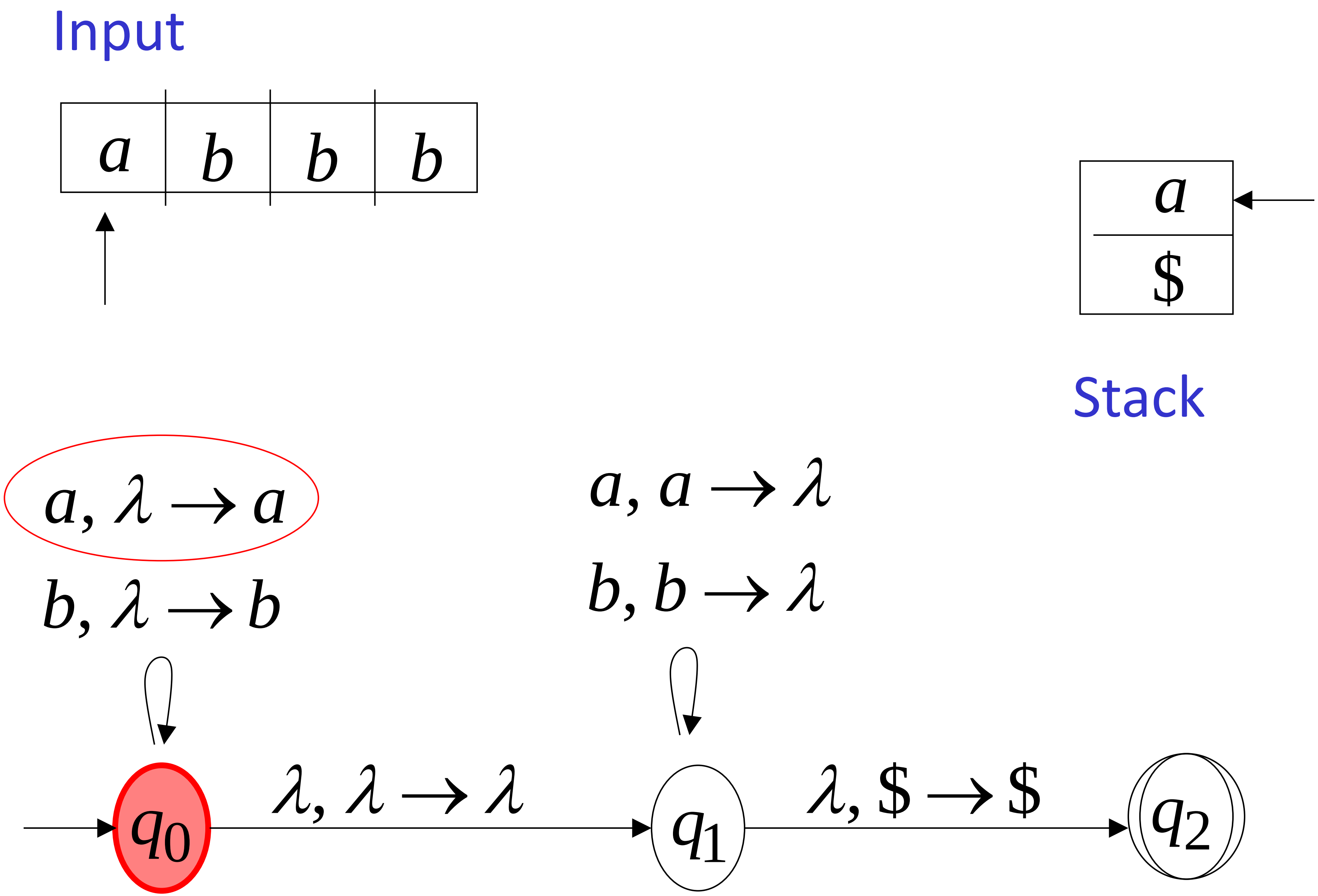

Input
a b b a
$
Stack
$a, \lambda \to a$
$b, \lambda \to b$
$a, a \to \lambda$
$b, b \to \lambda$
accept
q_0
$\lambda, \lambda \to \lambda$
q_1
$\lambda, \$ \to \$$
q_2

Rejection Example:

Time 0

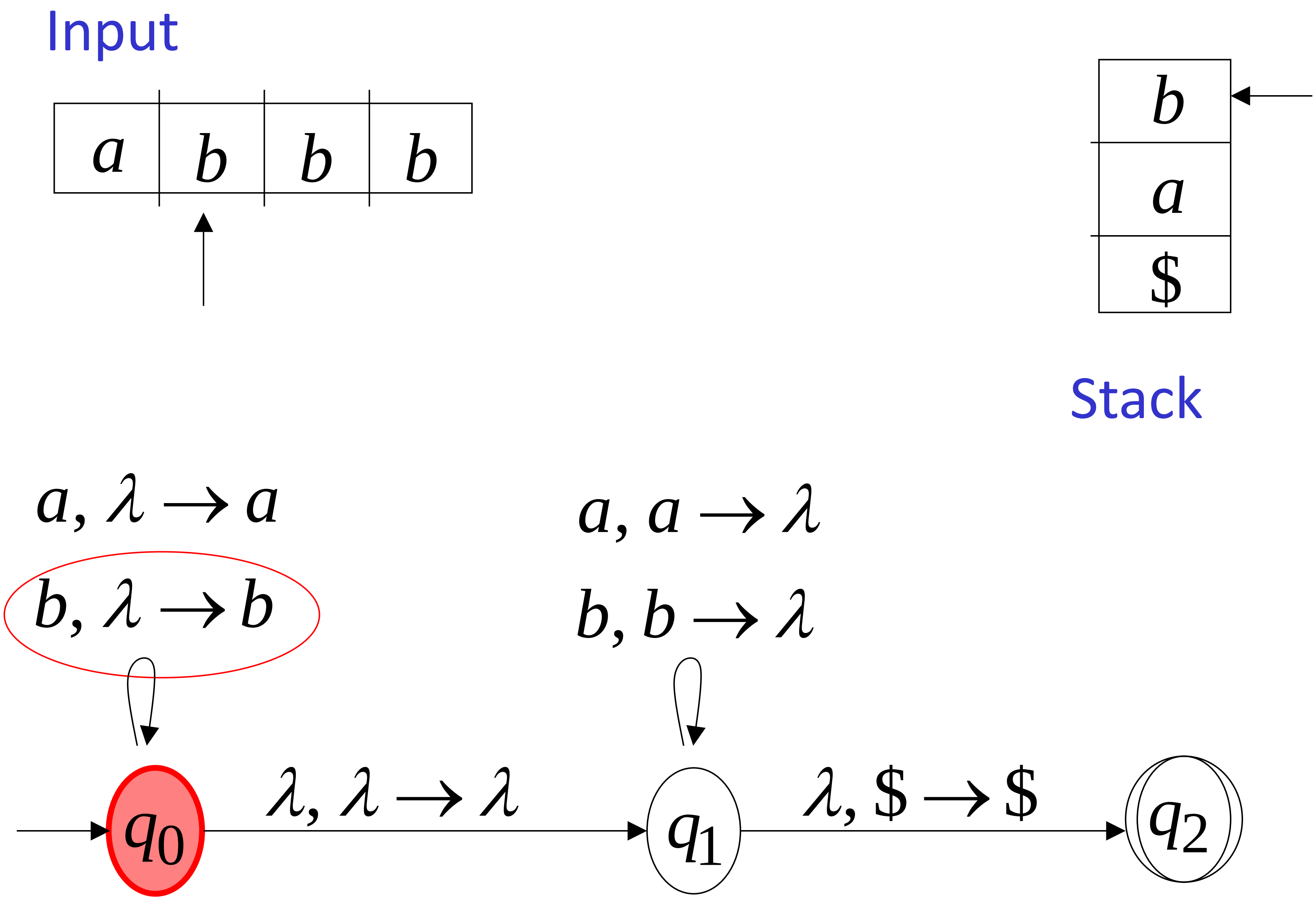


Time 1



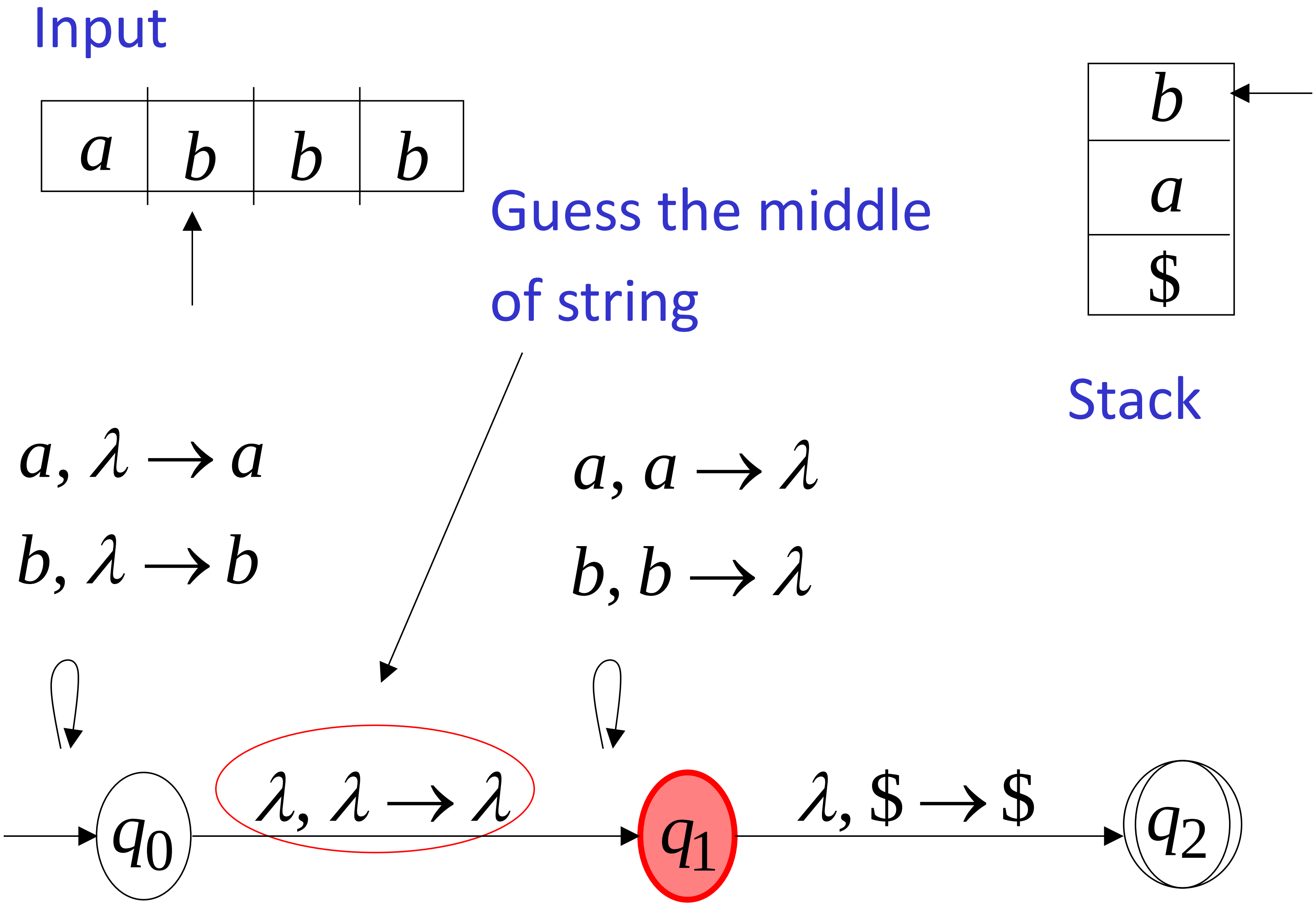
Input
a
b
b
b
a
$
Stack
$a, \lambda \to a$
$b, \lambda \to b$
$a, a \to \lambda$
$b, b \to \lambda$
$\lambda, \lambda \to \lambda$
$\lambda, \$ \to \$$
q_0
q_1
q_2

Time 2

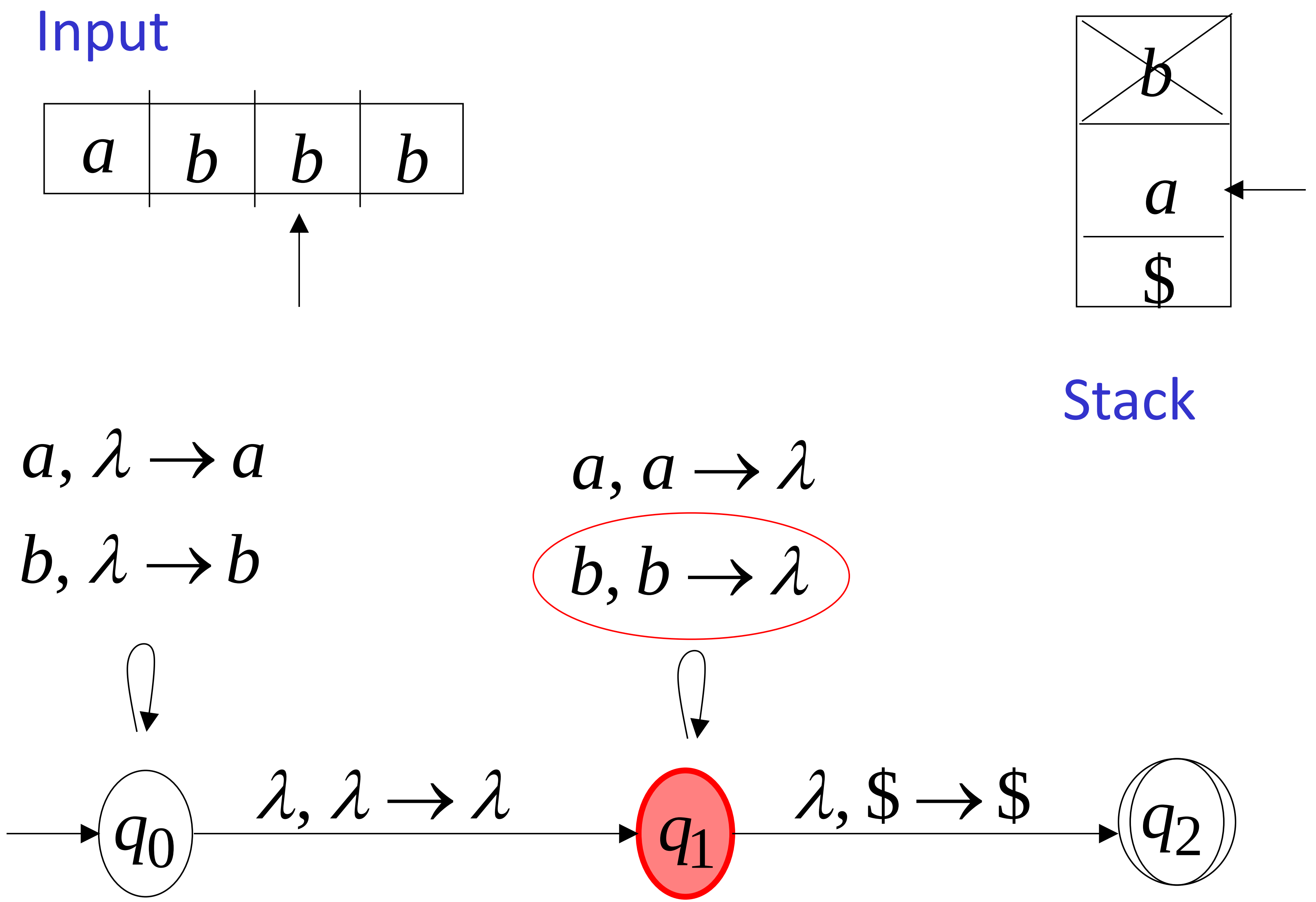


Input
a b b b
b
a
$
Stack
$a, \lambda \rightarrow a$
$b, \lambda \rightarrow b$
$a, a \rightarrow \lambda$
$b, b \rightarrow \lambda$
$\lambda, \lambda \rightarrow \lambda$
$\lambda, \$ \rightarrow \$$
q_0
q_1
q_2

Time 3


Input
a b b b
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

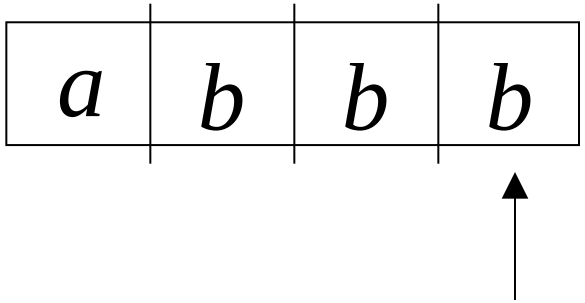
Time 4



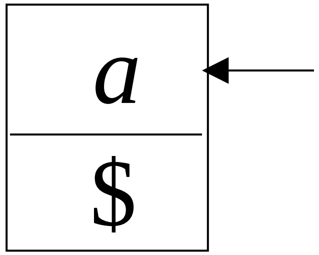
Time 5

There is no possible transition.

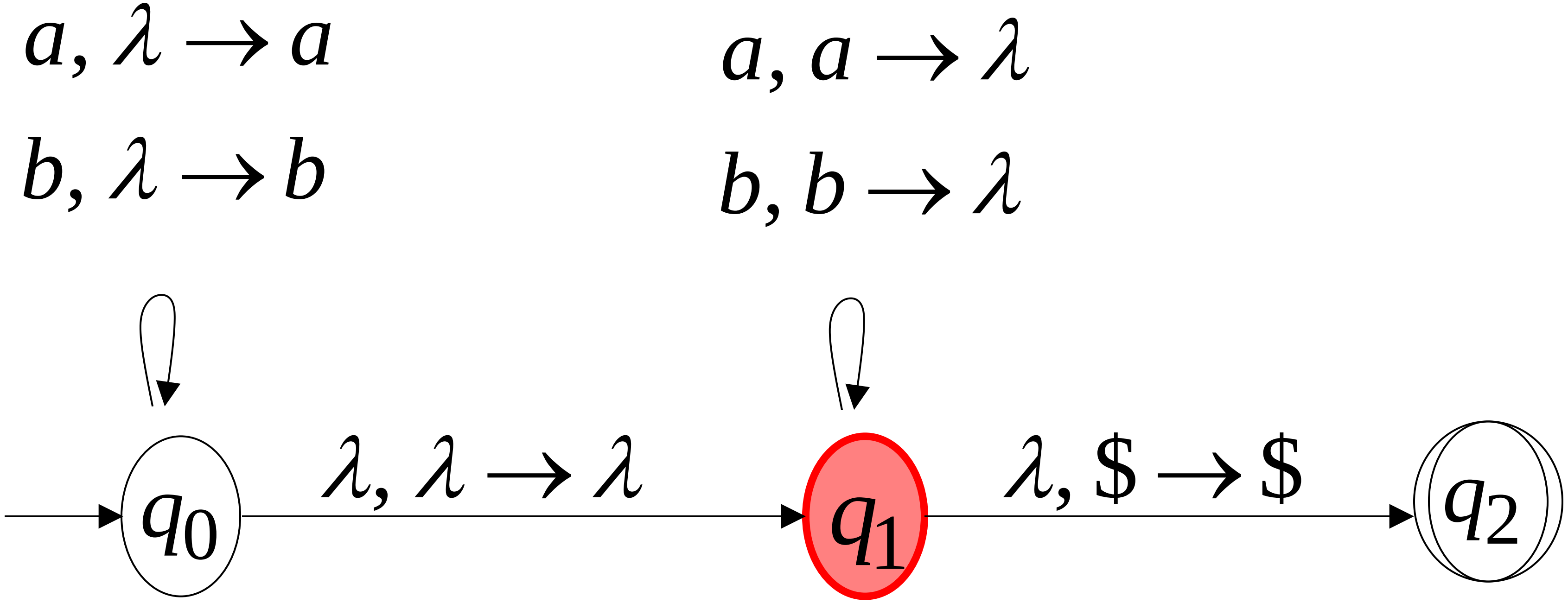
Input



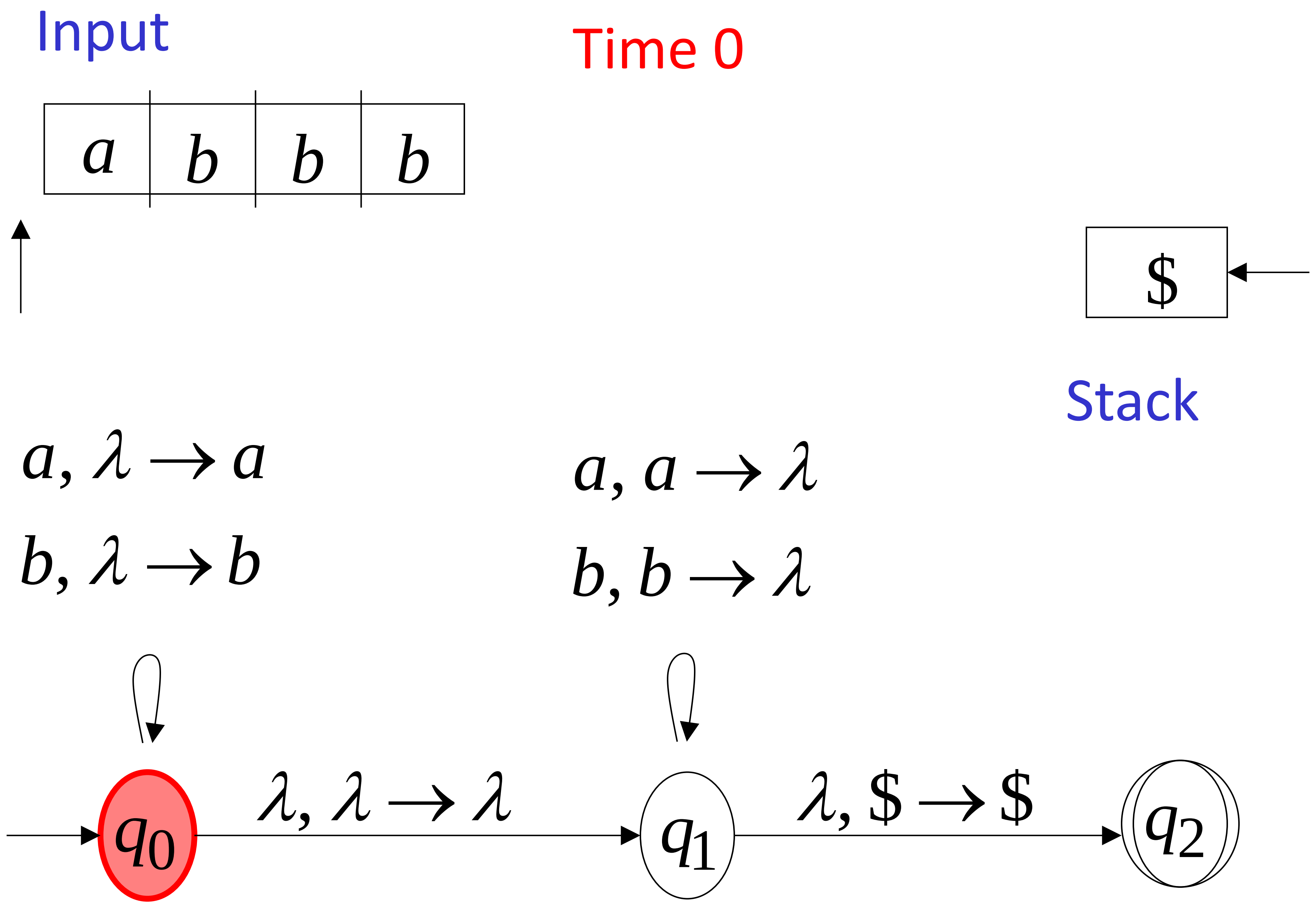
Input is not consumed



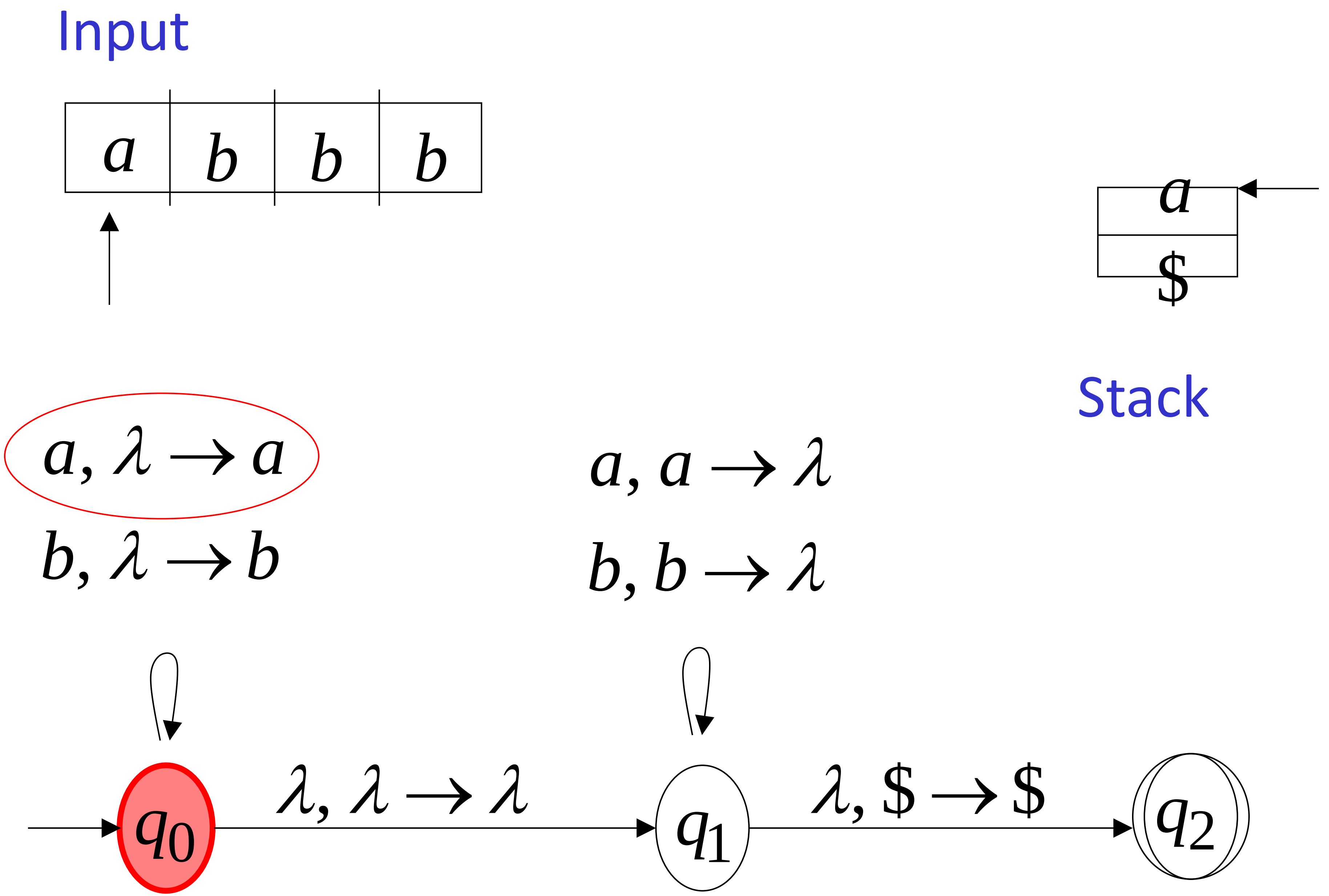
Stack



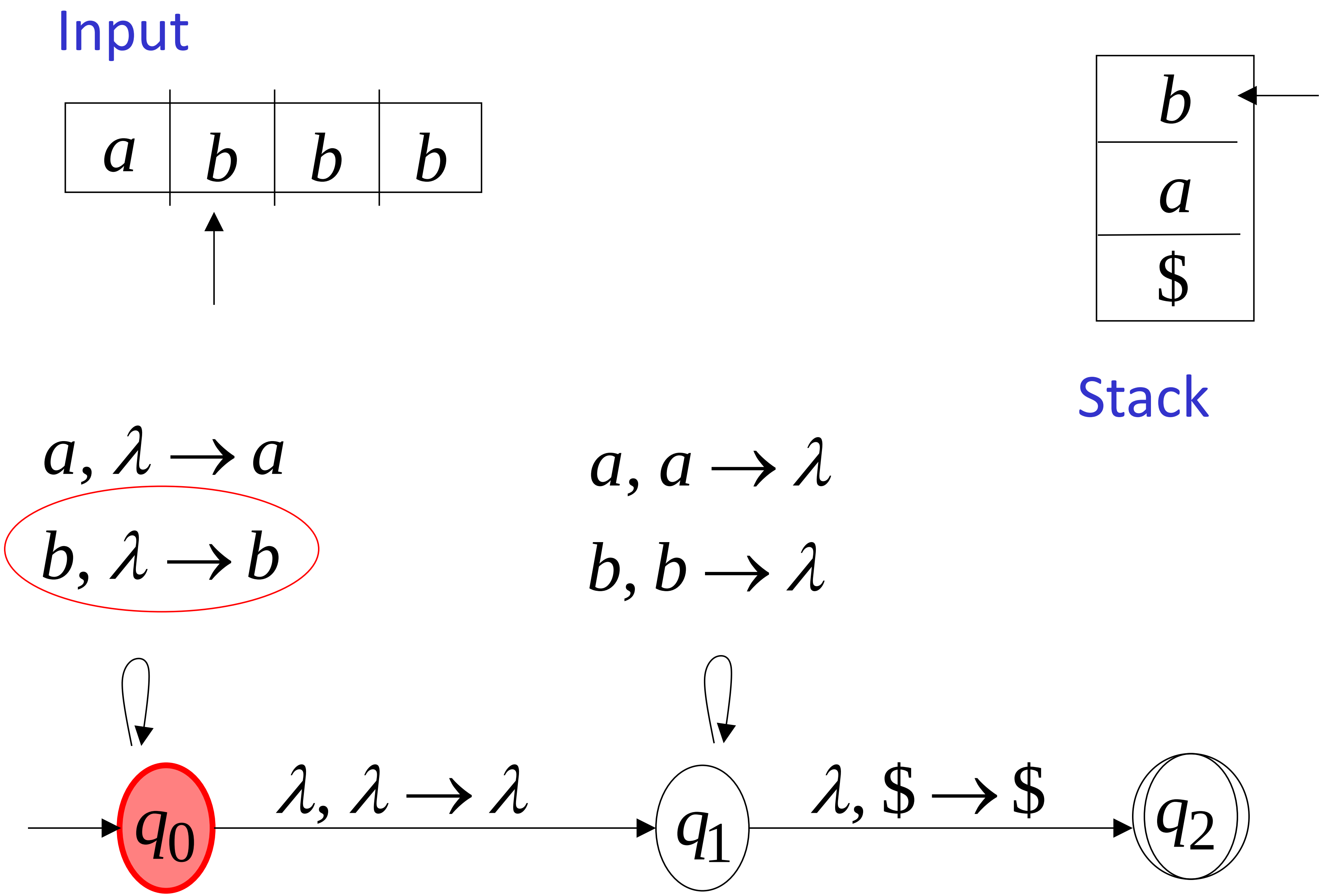
Another computation on same string:



Time 1

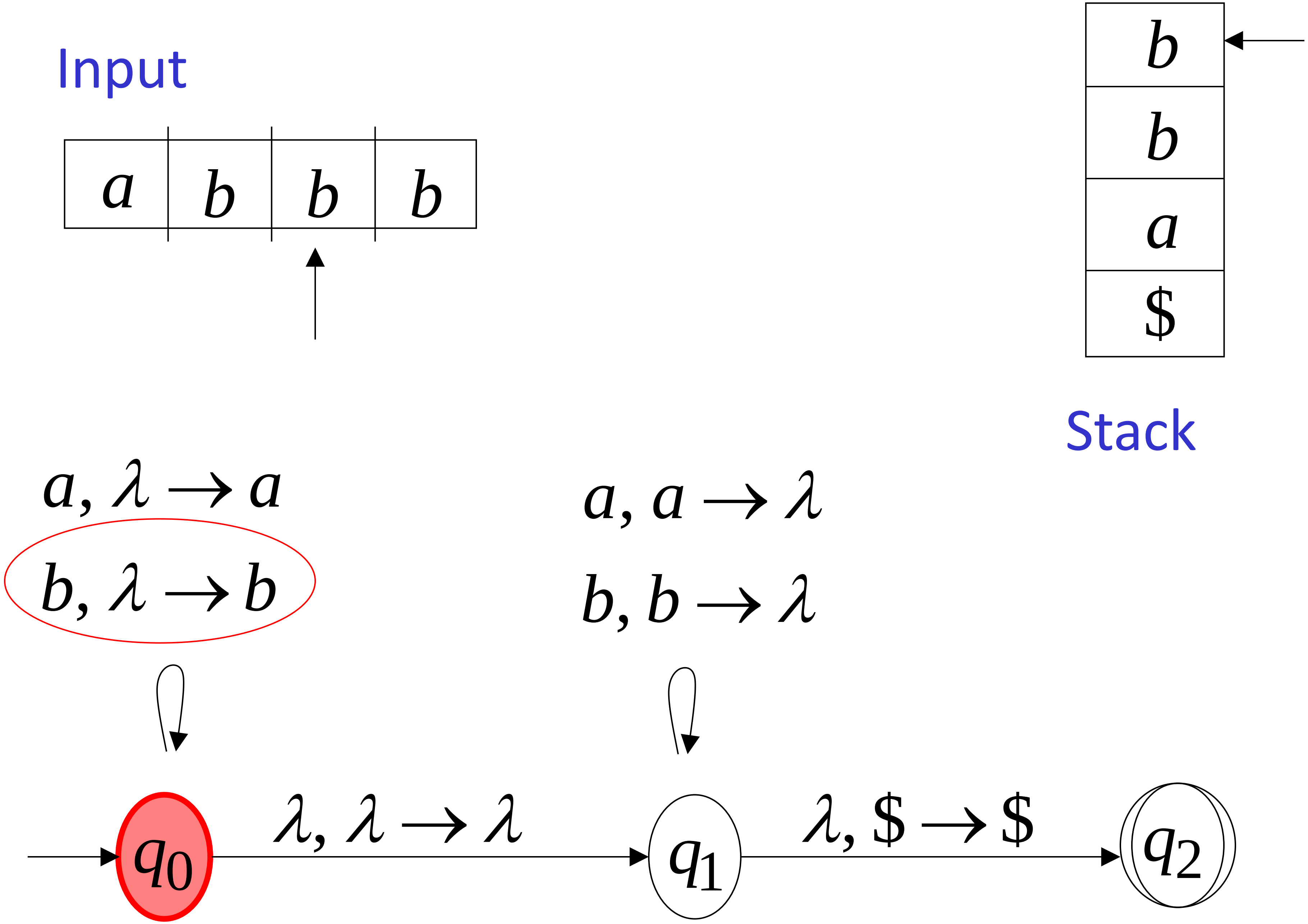

Input
a b b b
a
$\$$
Stack
$a, \lambda \rightarrow a$
$b, \lambda \rightarrow b$
$a, a \rightarrow \lambda$
$b, b \rightarrow \lambda$
q_0
$\lambda, \lambda \rightarrow \lambda$
q_1
$\lambda, \$ \rightarrow \$$
q_2

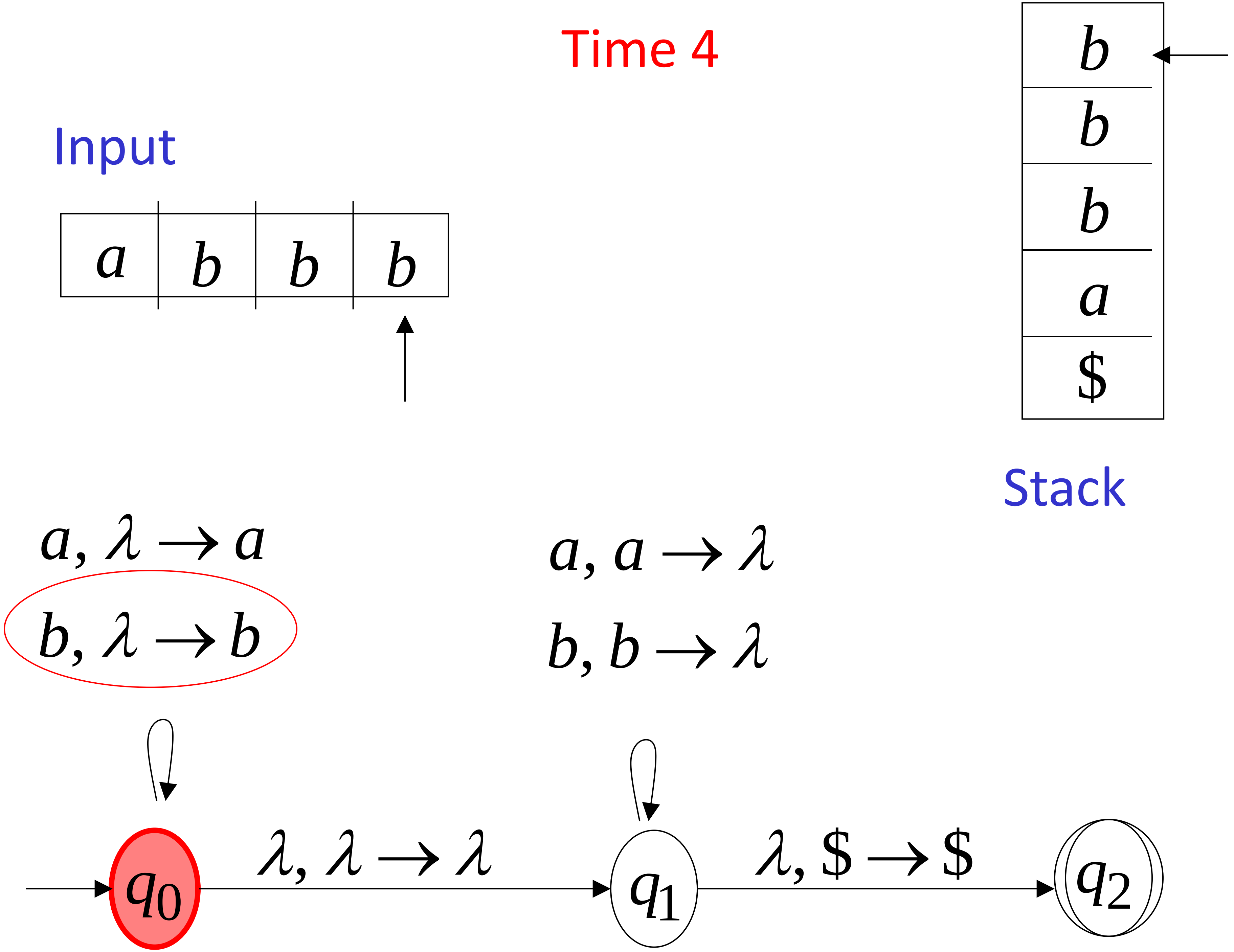
Time 2



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


Time 4
Input
a b b b
Stack
b
b
b
a
$
a, λ → a
b, λ → b
a, a → λ
b, b → λ
λ, λ → λ
λ, $ → $
q0
q1
q2

Time 5

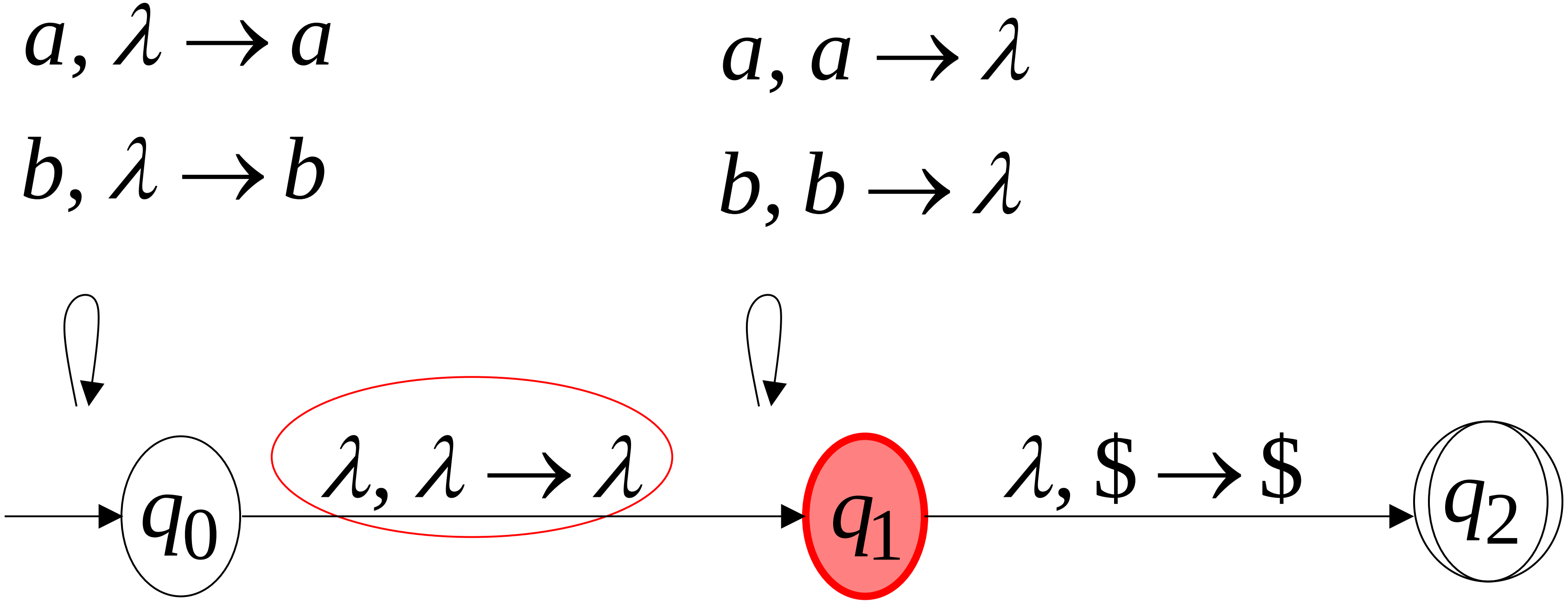
Input

a	b	b	b

No final state is reached

b
b
b
a
$\$$

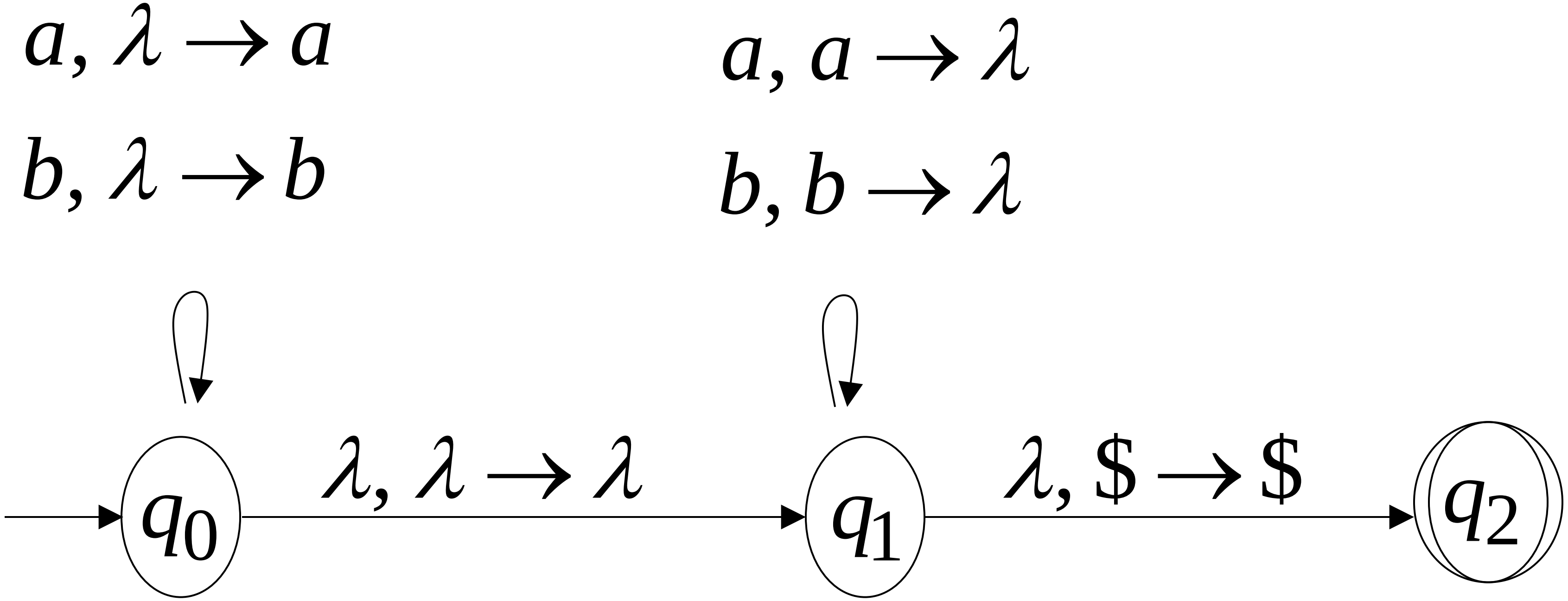
Stack



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

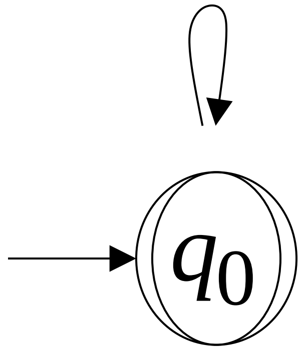
NPDA M

$$L(M) = \{a^n b^m \ : \ n \geq m-1\}$$

$$a, \lambda \rightarrow a$$

$$b, a \rightarrow \lambda$$

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



Execution Example:

Time 0

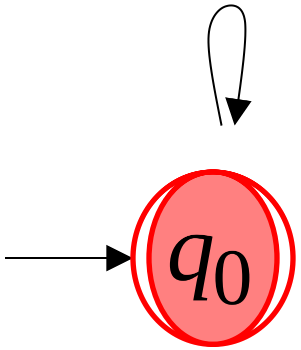
Input

a	a	b

$$a, \lambda \rightarrow a$$

$$b, a \rightarrow \lambda$$

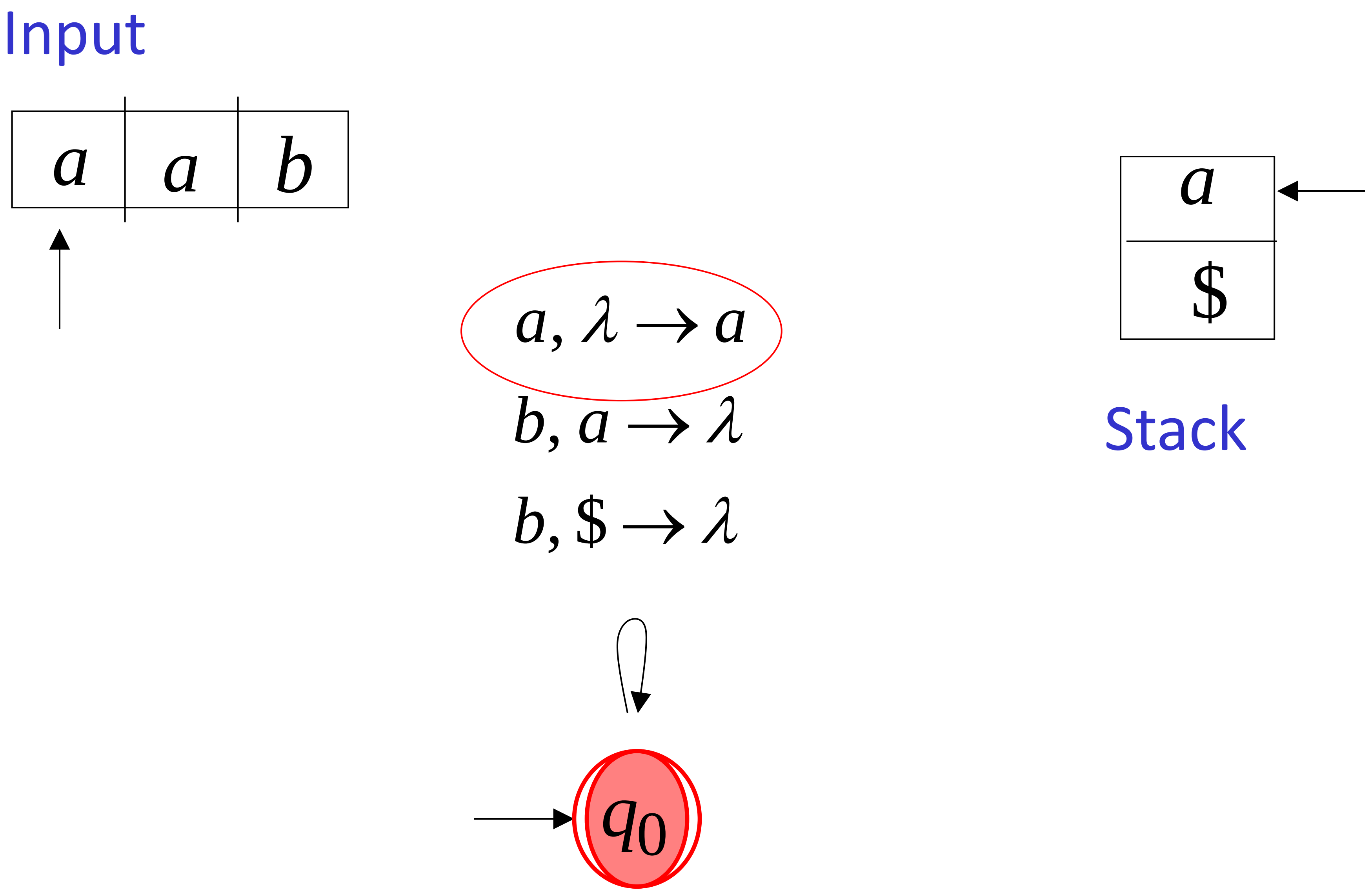
$$b, \$ \rightarrow \lambda$$



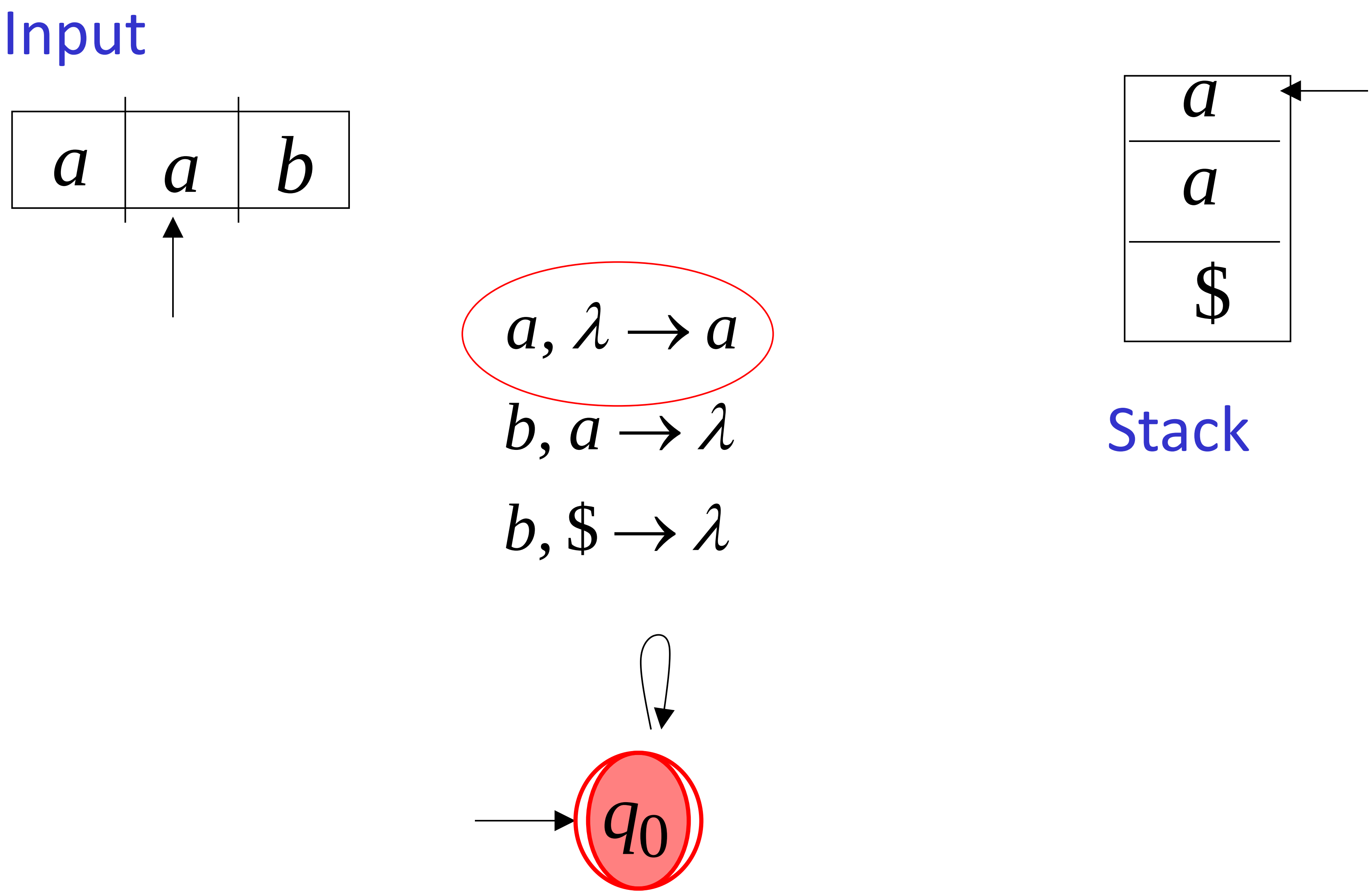
$\$$

Stack

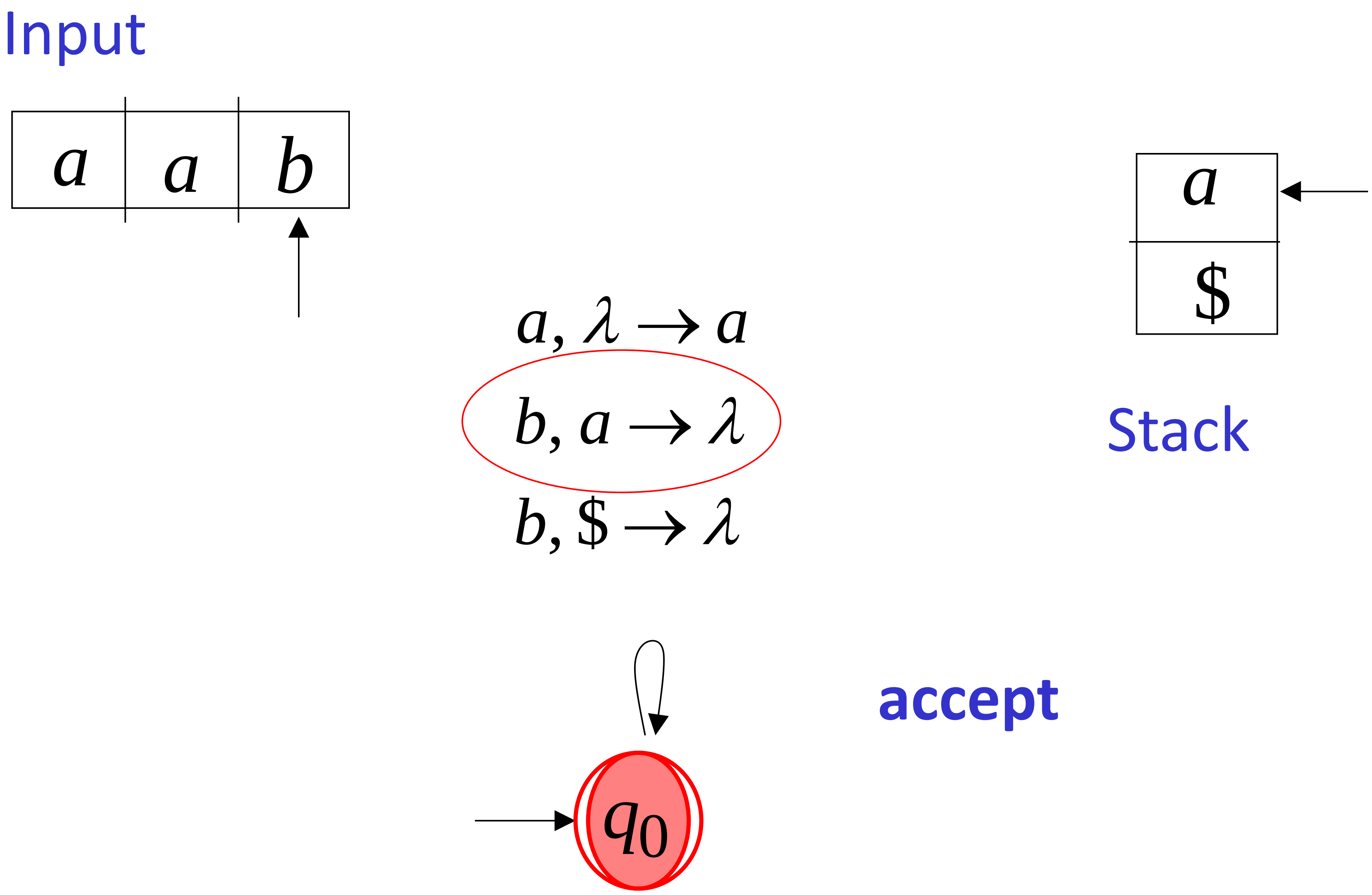
Time 1



Time 2

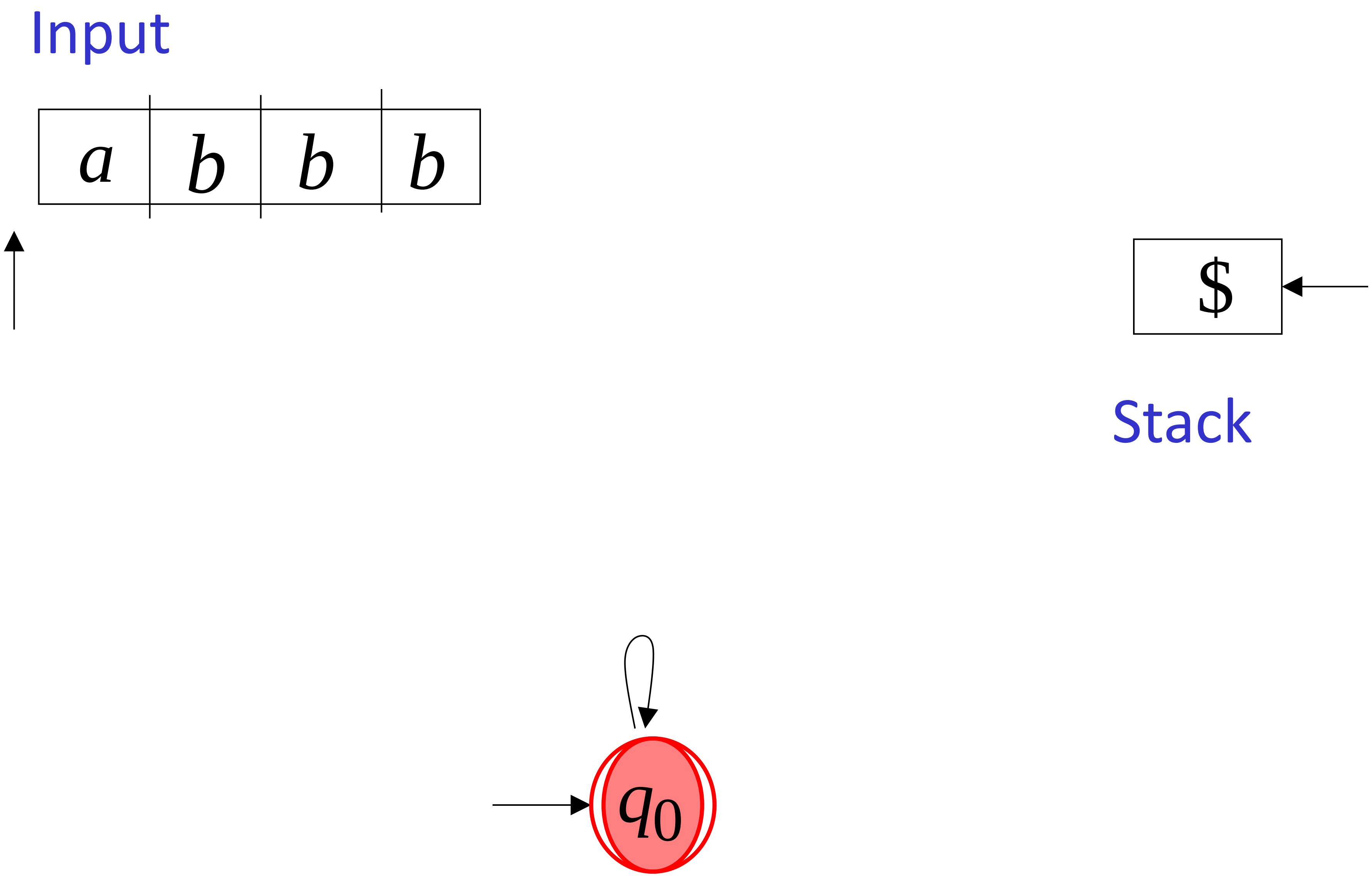


Time 3

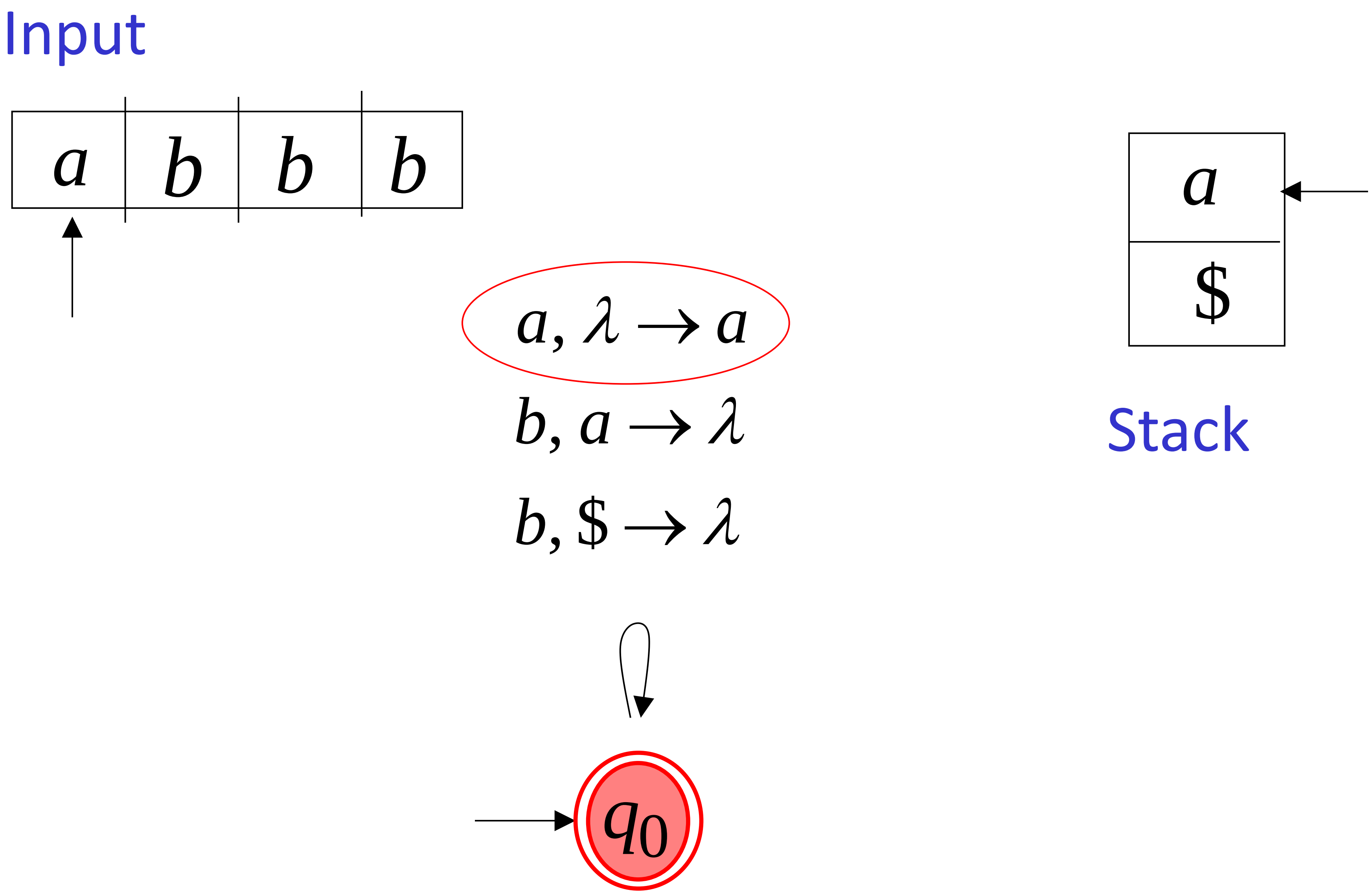

Input
a a b
$a, \lambda \rightarrow a$
$b, a \rightarrow \lambda$
$b, \$ \rightarrow \lambda$
q_0
a
$
Stack
accept

Rejection example:

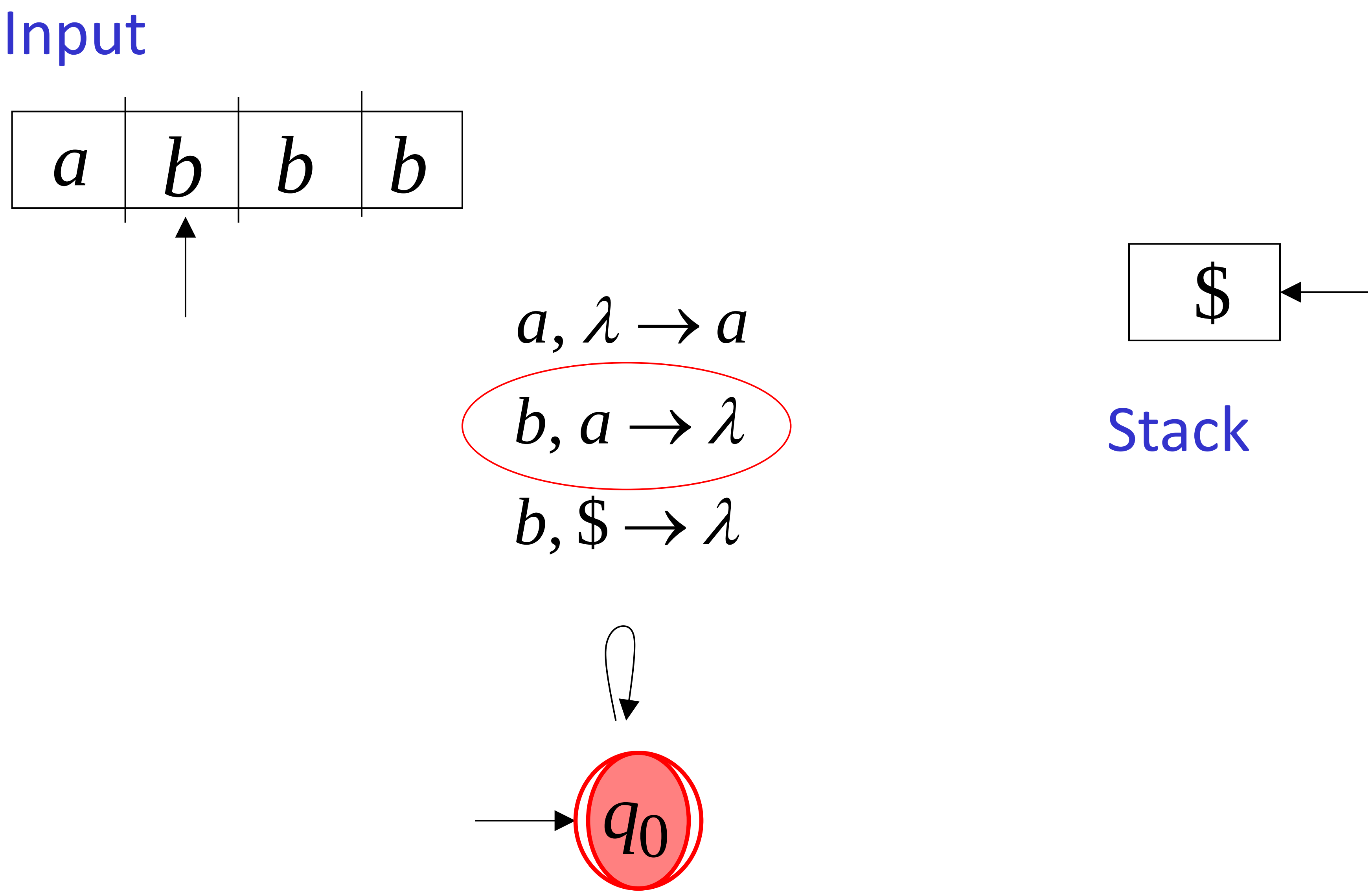
Time 0



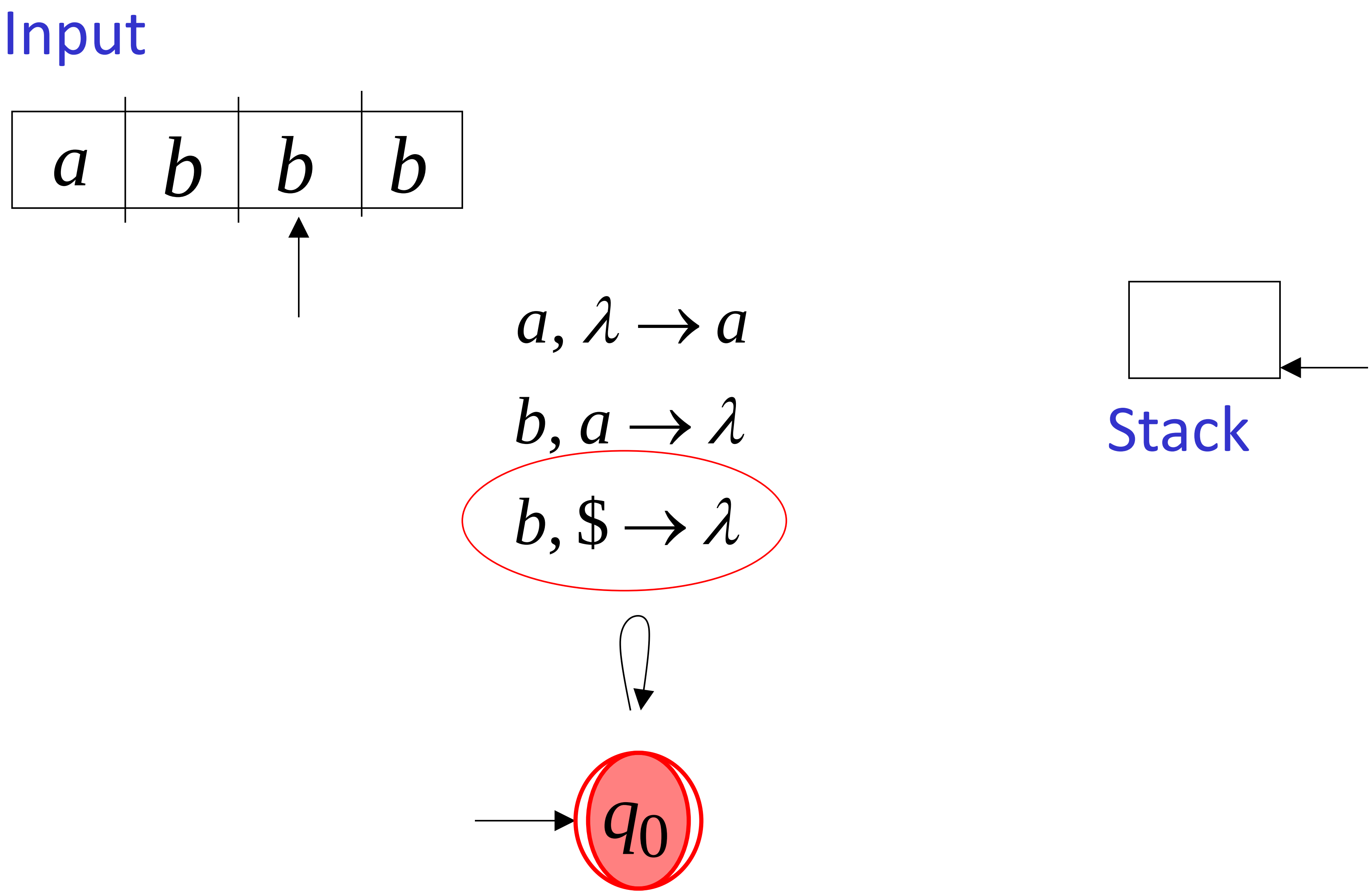
Time 1



Time 2



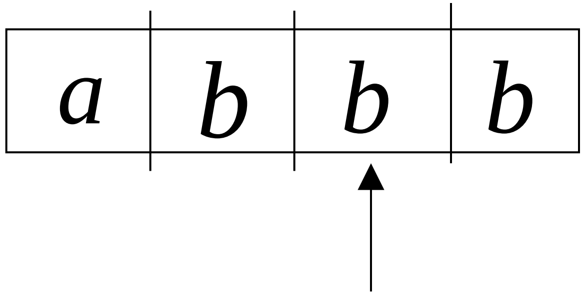
Time 3

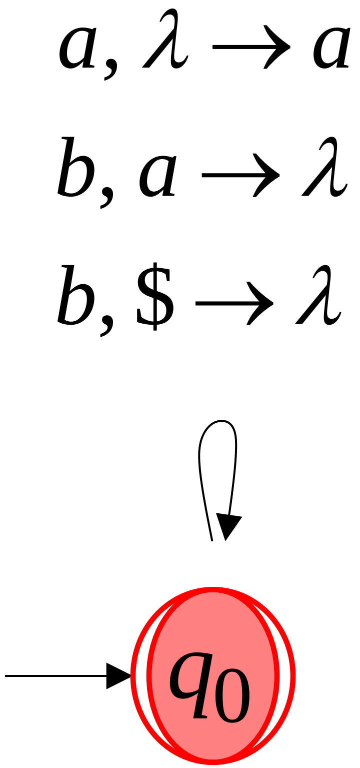


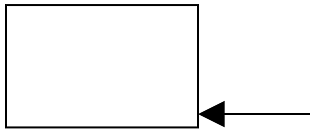
Input
a b b b
$a, \lambda \rightarrow a$
$b, a \rightarrow \lambda$
$b, \$ \rightarrow \lambda$
q_0
Stack

Time 4

Input







Stack

Halt and Reject