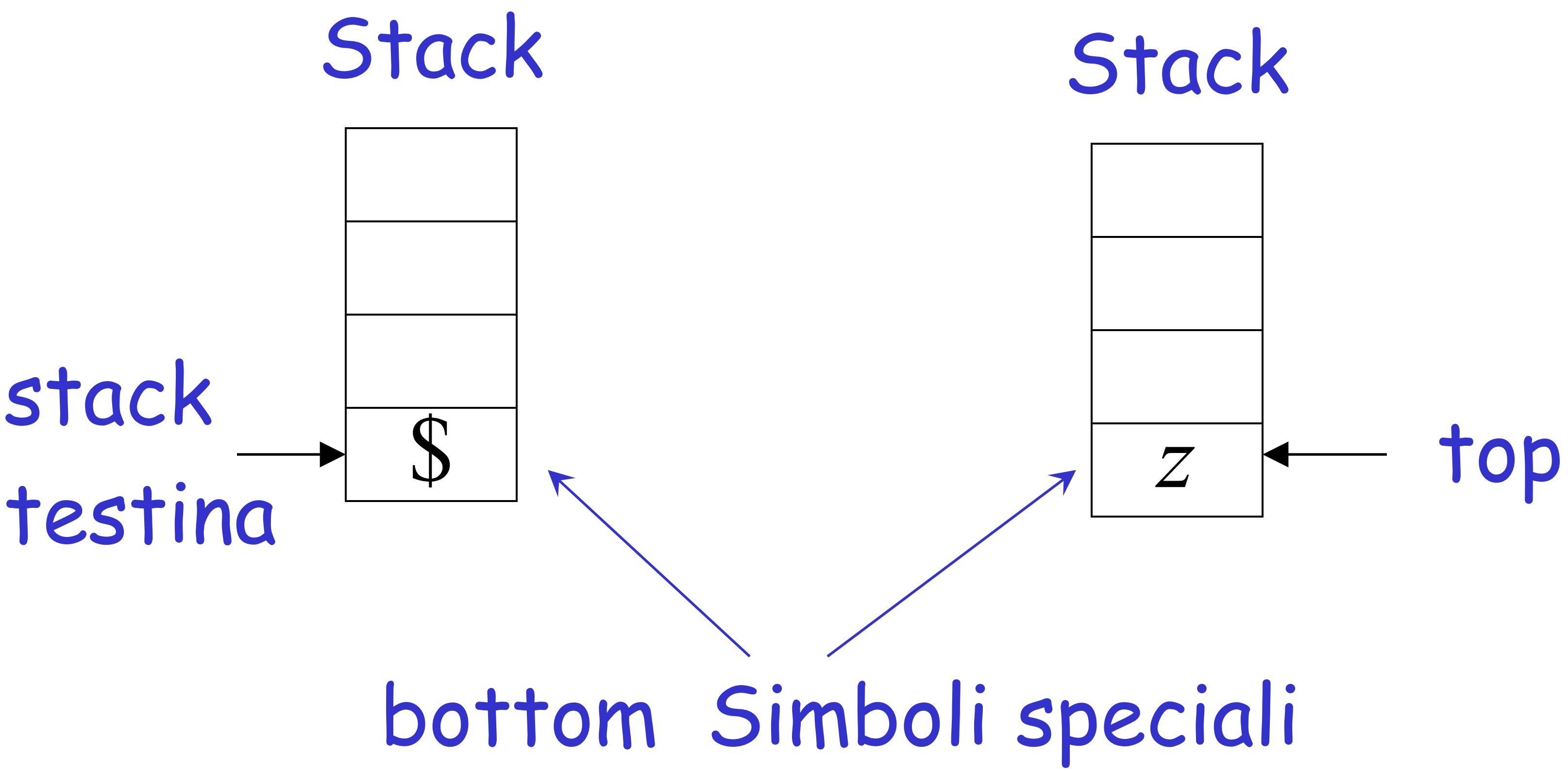
Pushdown Automata
PDA

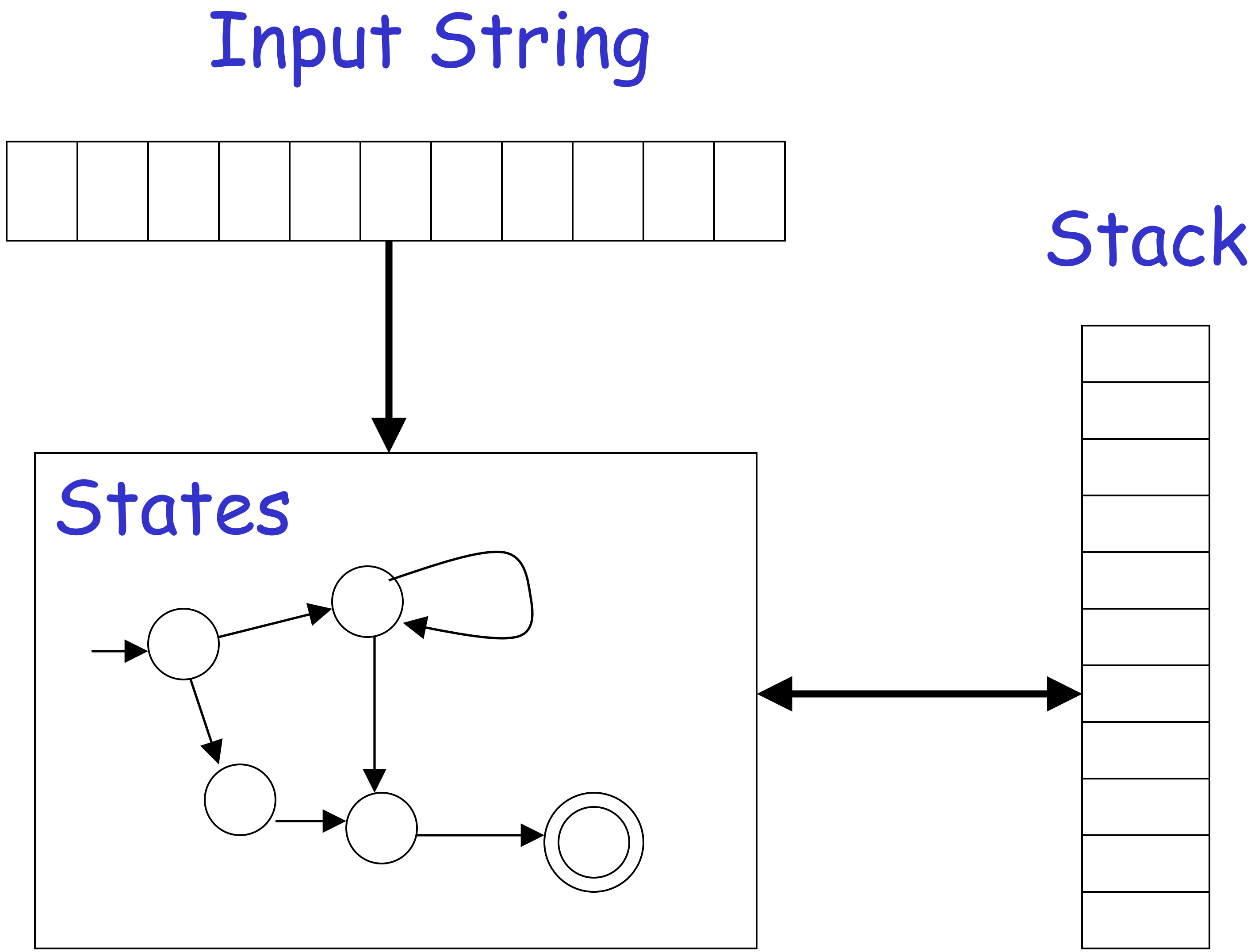
Initial Stack Symbol



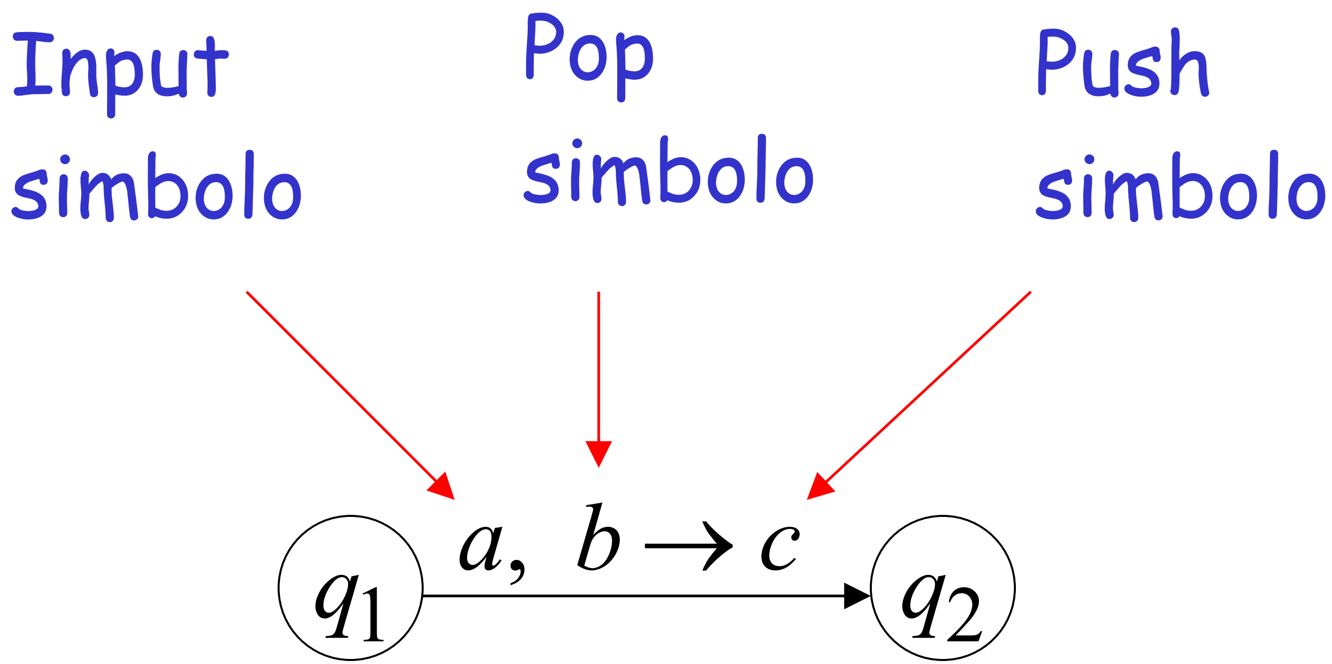
Simboli speciali

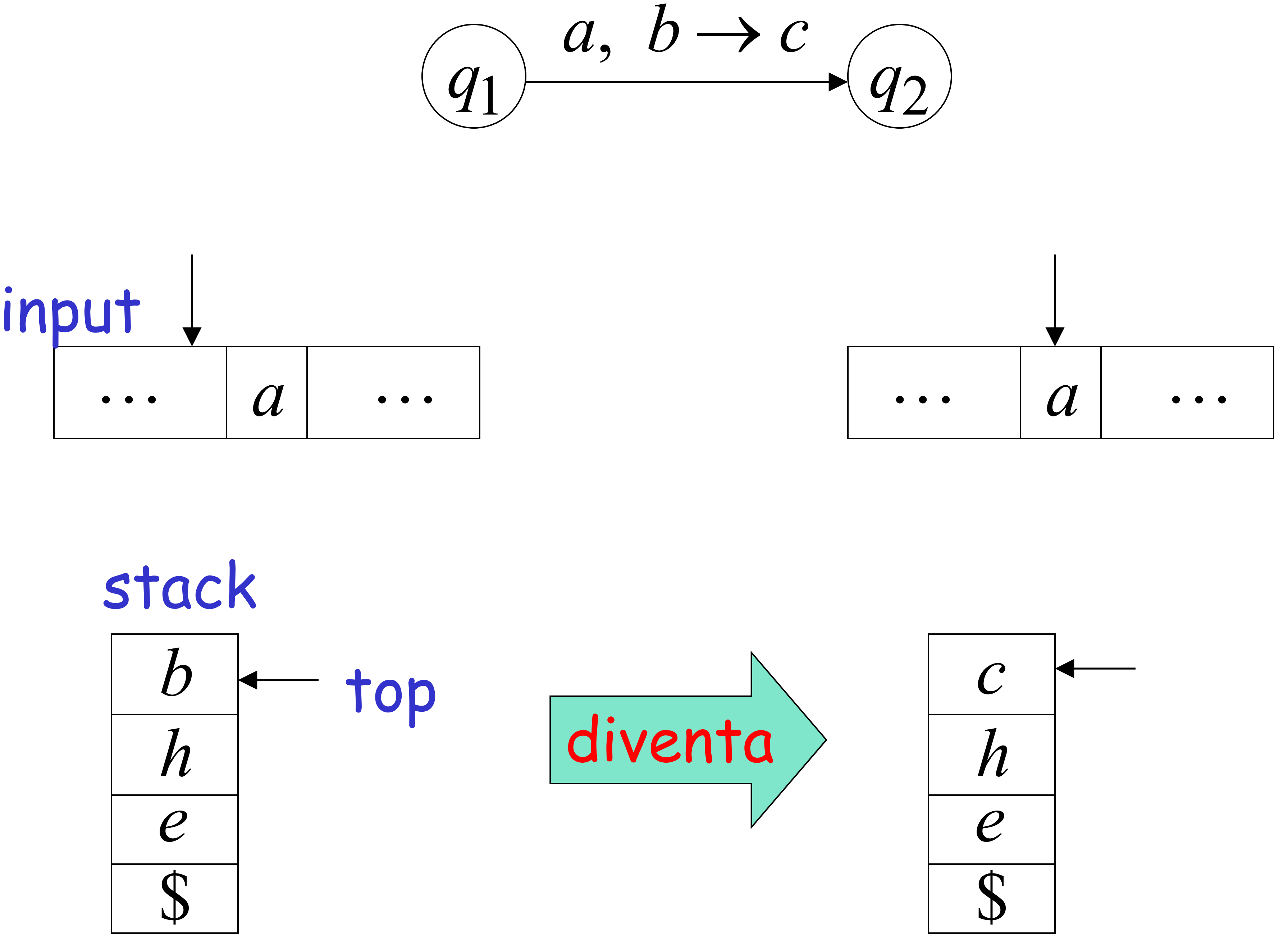
Che appaiono al tempo 0

Pushdown Automaton -- PDA

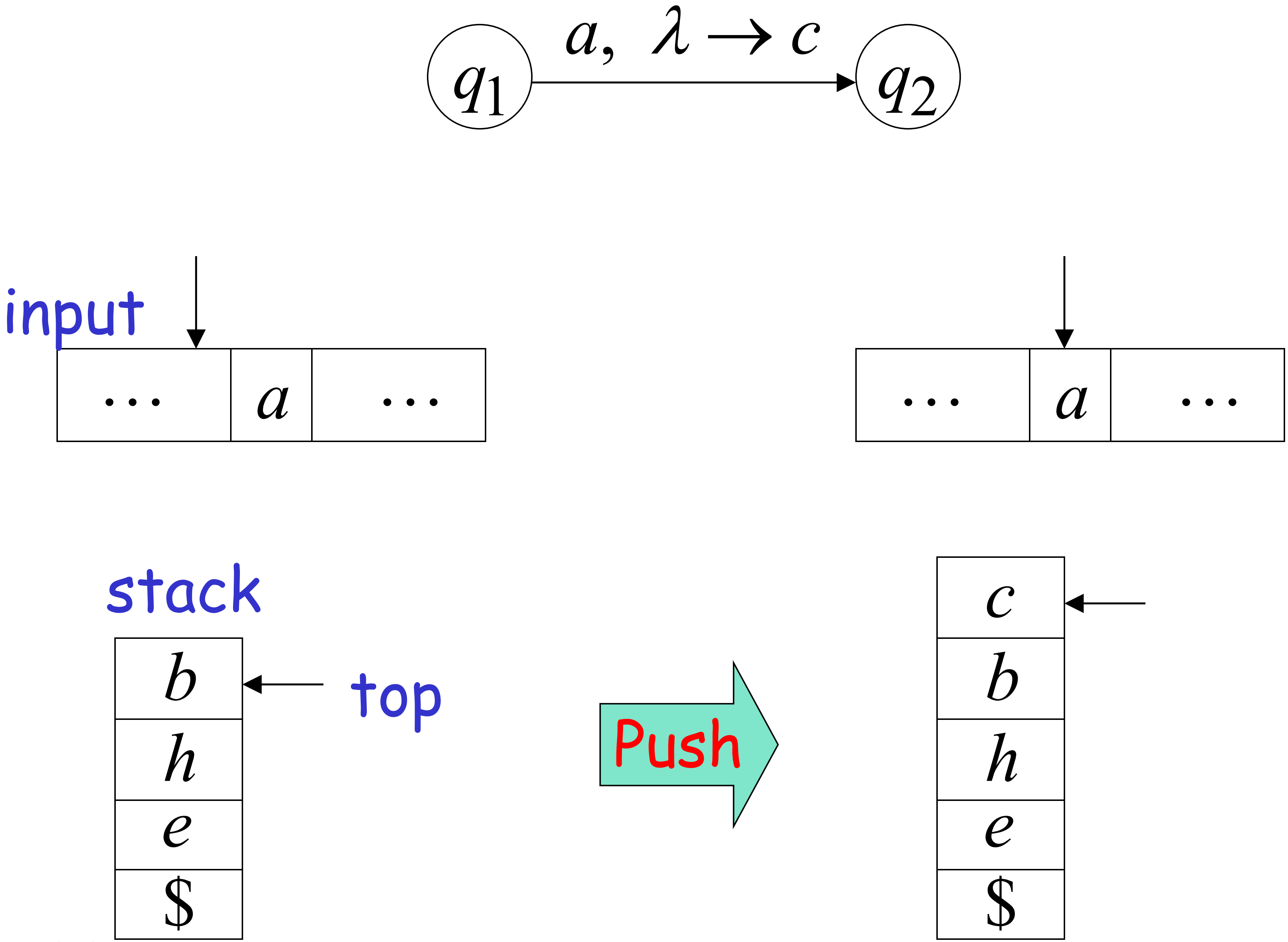


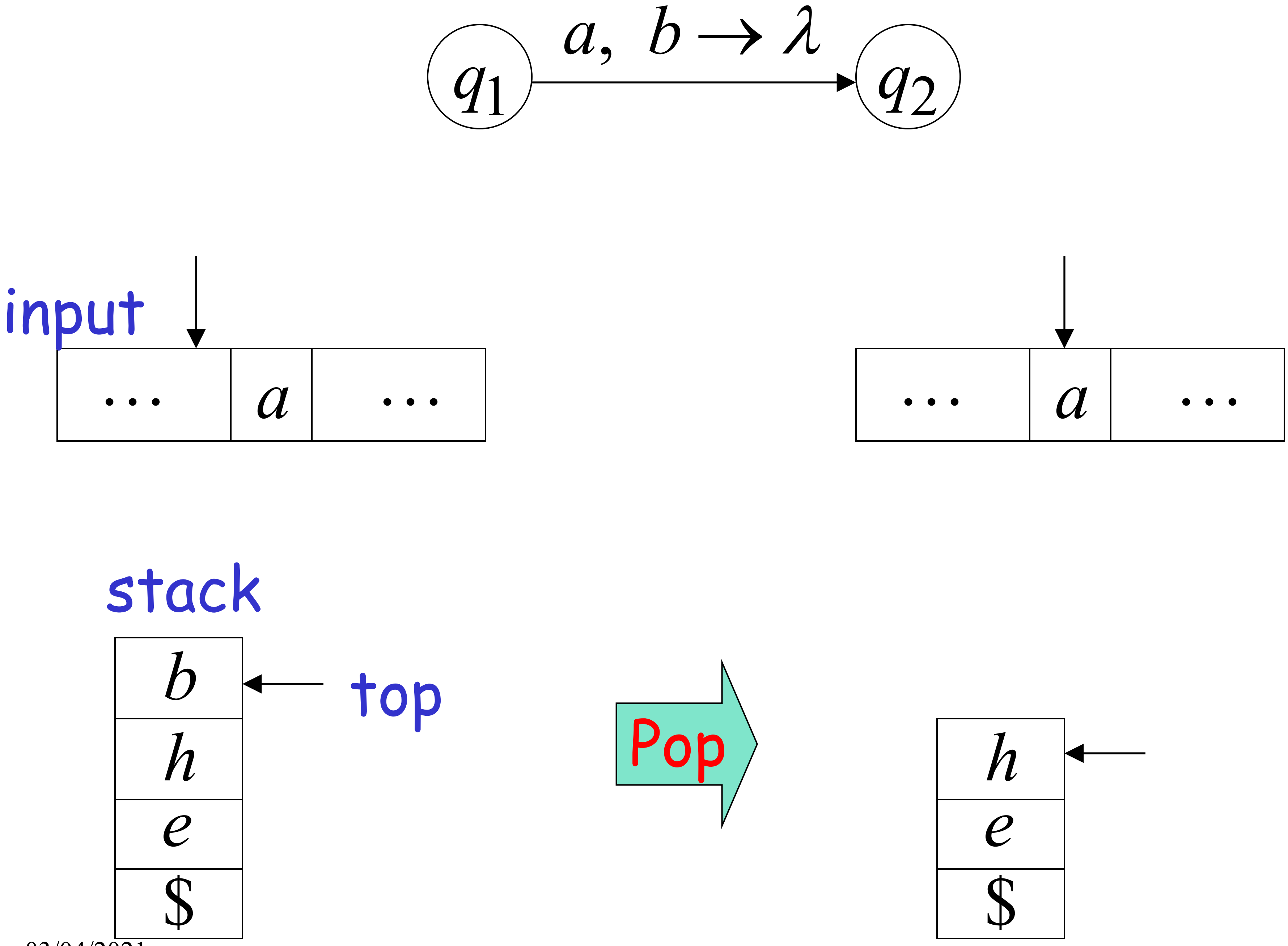
Gli stati

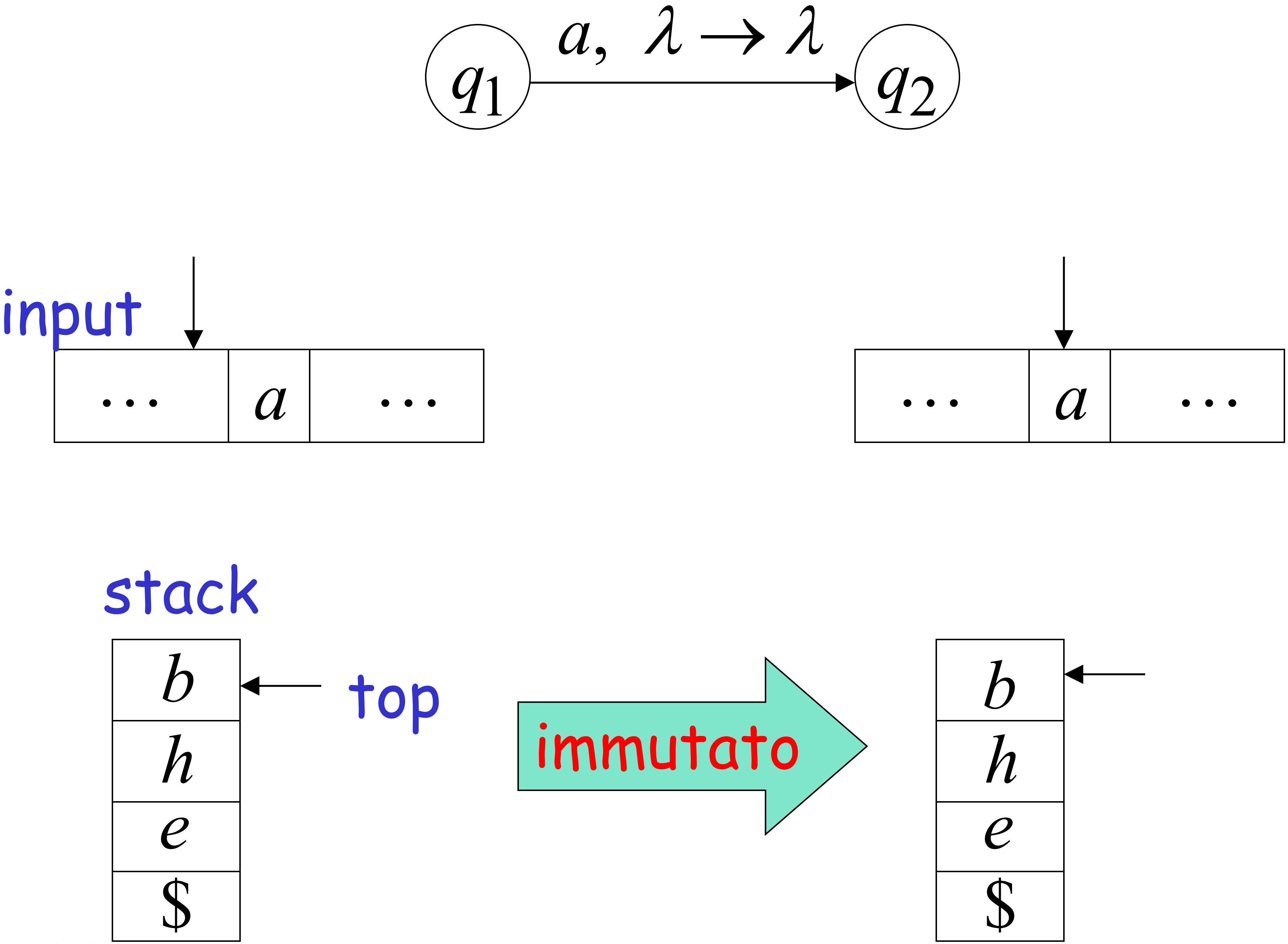

Input simbolo
Pop simbolo
Push simbolo
$a, \; b \rightarrow c$
q_1
q_2



$a, \ b \rightarrow c$
q_1
q_2
input
$\cdots$ a $\cdots$
$\cdots$ a $\cdots$
stack
top
b
h
e
$\$$
diventa
c
h
e
$\$$

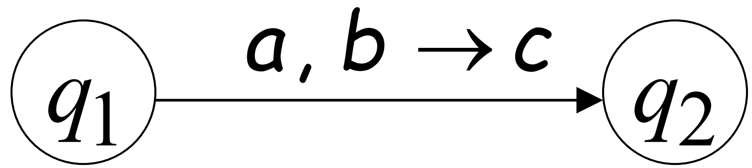

q_1
$a, \lambda \rightarrow c$
q_2
input
$\cdots$ a $\cdots$
$\cdots$ a $\cdots$
stack
b
h
e
$\$$
top
Push
c
b
h
e
$\$$

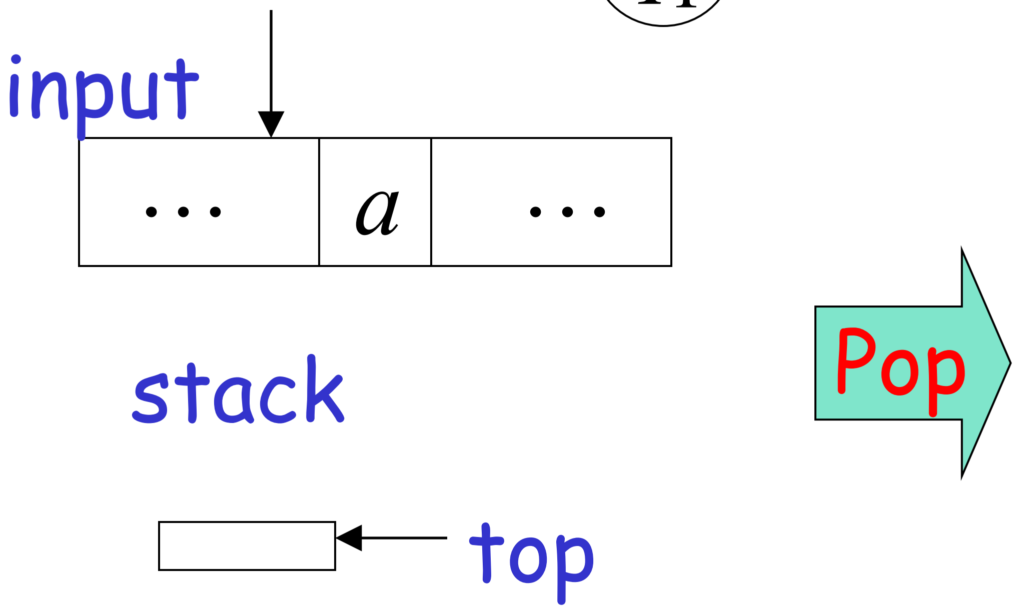

$a, b \to \lambda$
q_1
q_2
input
$\cdots$
a
$\cdots$
$\cdots$
a
$\cdots$
stack
b
h
e
$\$$
top
Pop
h
e
$\$$



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

Pop da uno stack vuoto





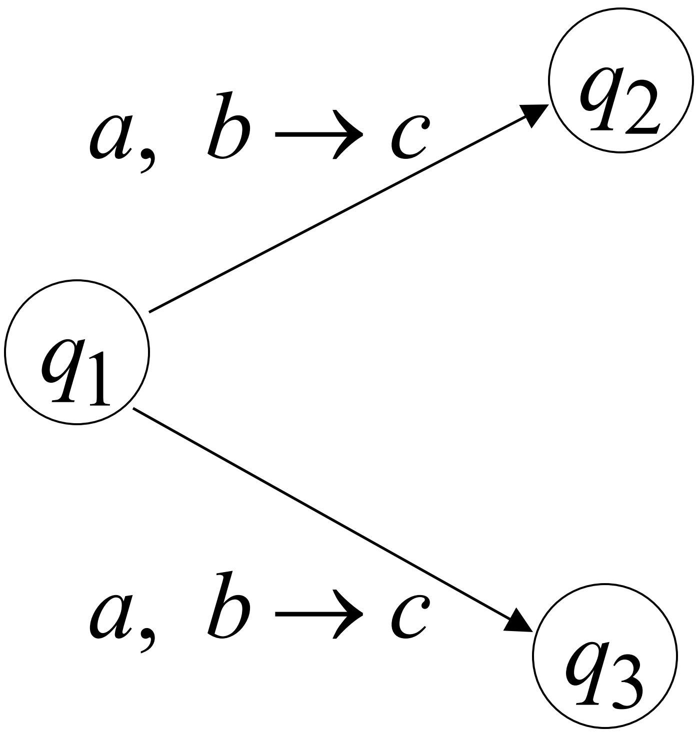
Automa si ferma!

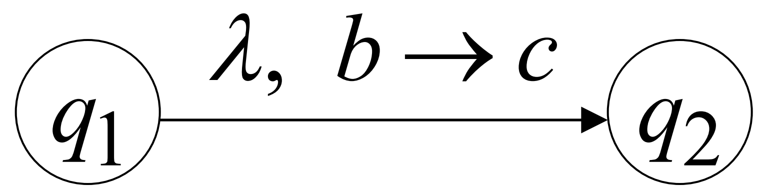
Se l'automata tenta di fare un pop
da uno stack vuoto allora si ferma
la computazione e rigetta l'input

Non-Determinismo

PDAs sono non-deterministici

Permettono transizioni non deterministiche



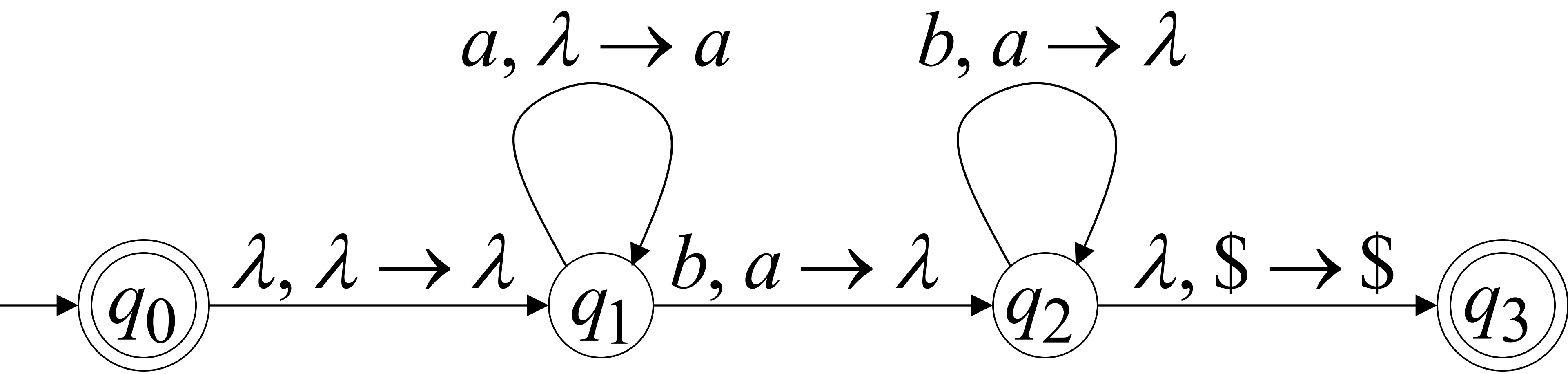


$\lambda -$ transition

Esempio di PDA

PDA M:

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



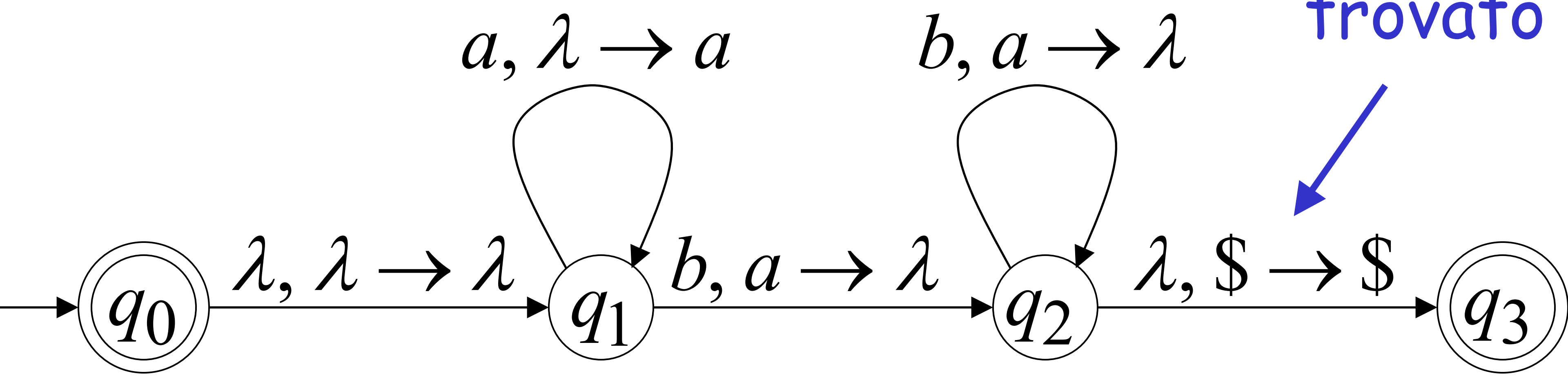
$$L(M) = \{a^n b^n : n \geq 0\}$$

Idea di base:

1. Push le a's nello stack

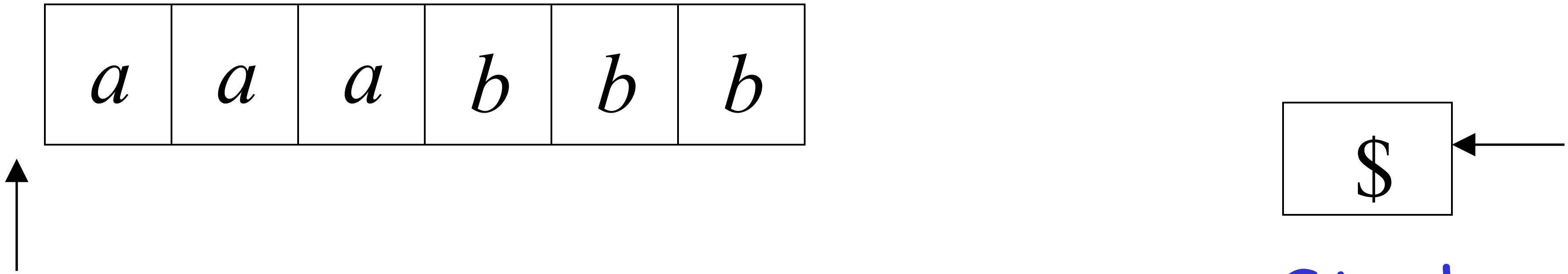
2. Verifica le b's in input con le a's nello stack

3. Match trovato



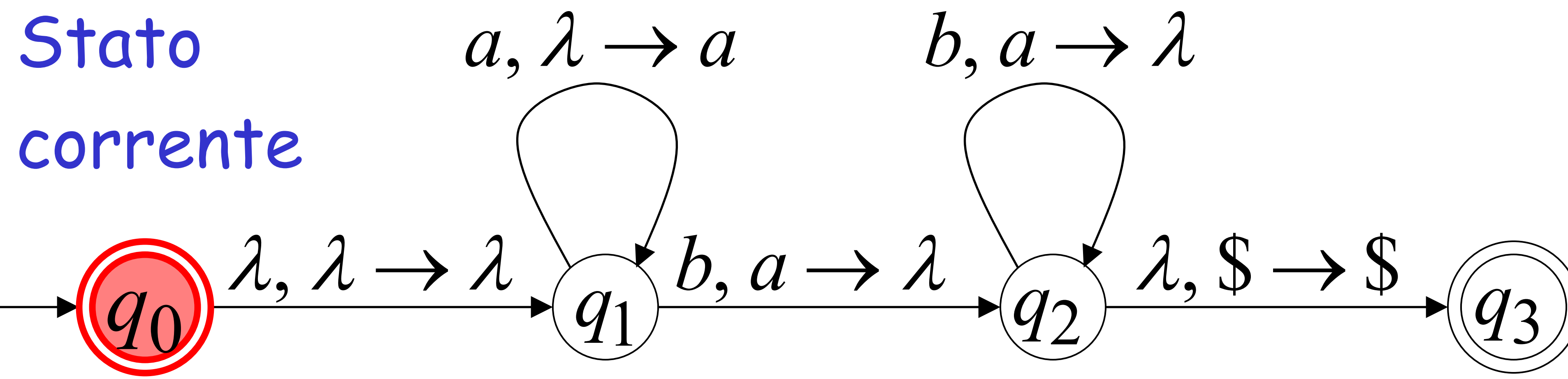
Esempio di esecuzione: Time 0

Input

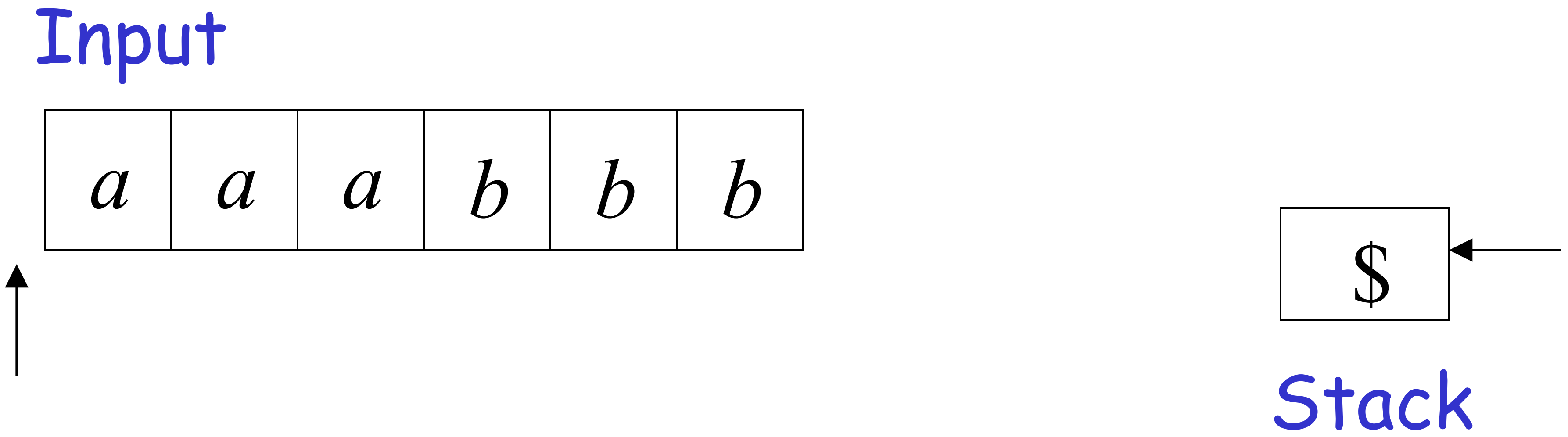


Stack

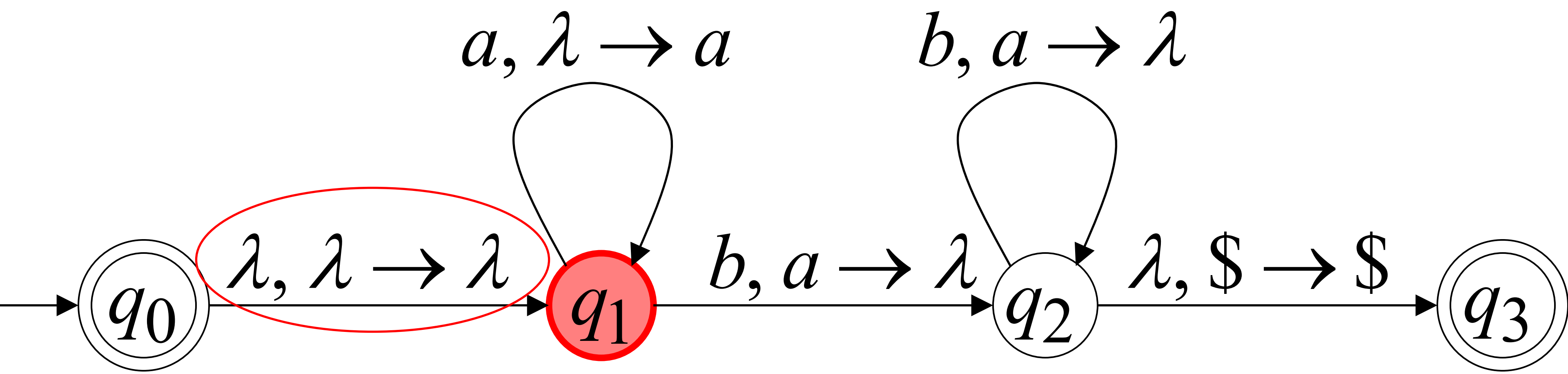
Stato corrente



Time 1



Input
a a a b b b
$\$$
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

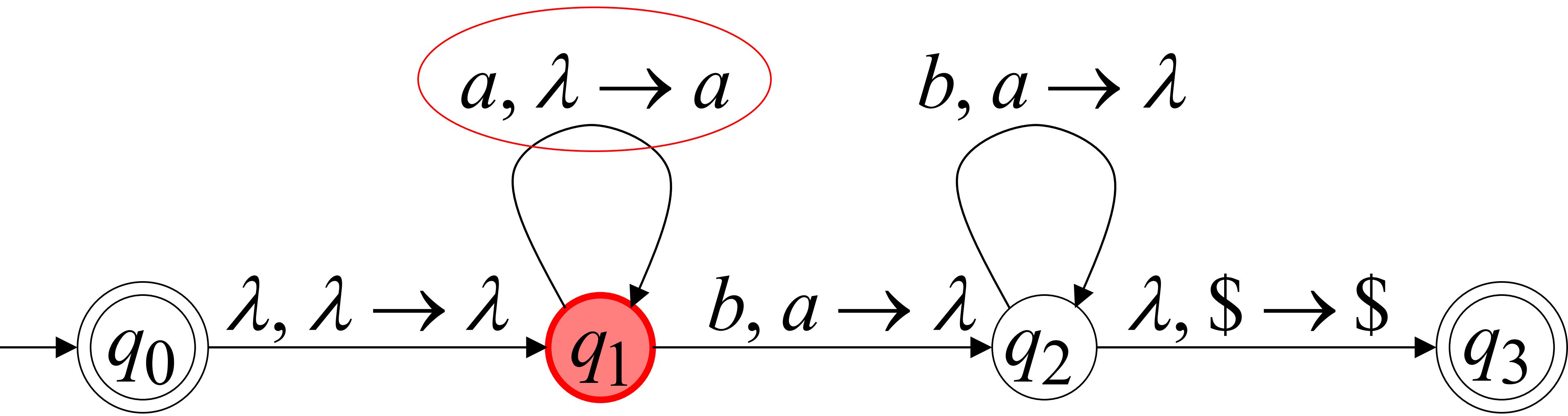
Time 2

Input

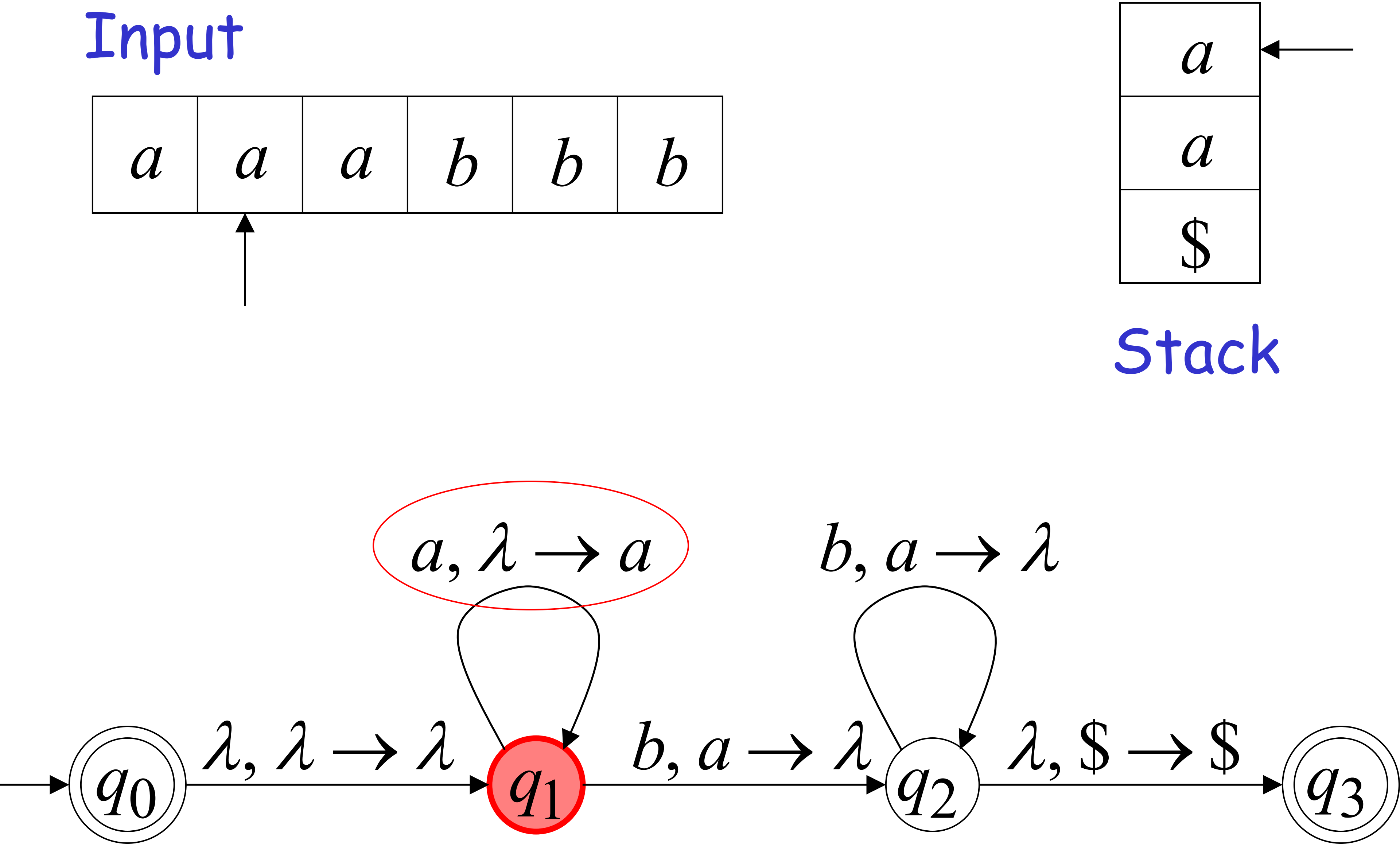
a	a	a	b	b	b

a
$\$$

Stack

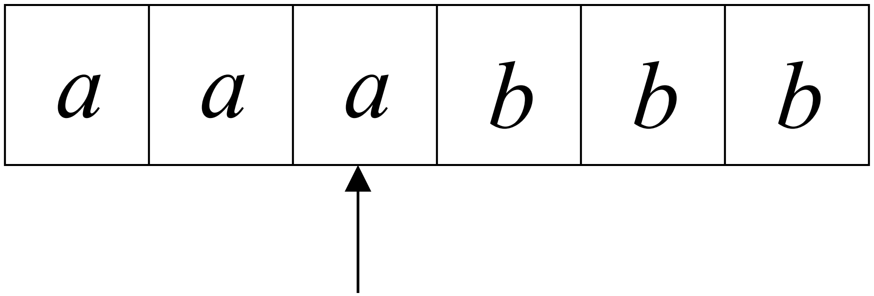


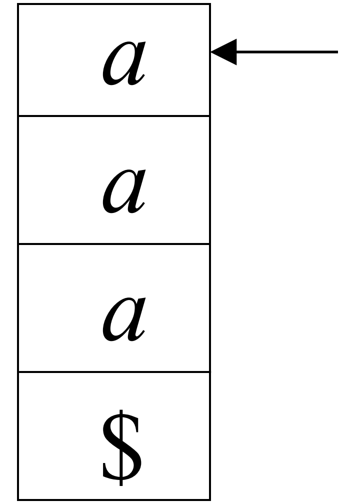
Time 3



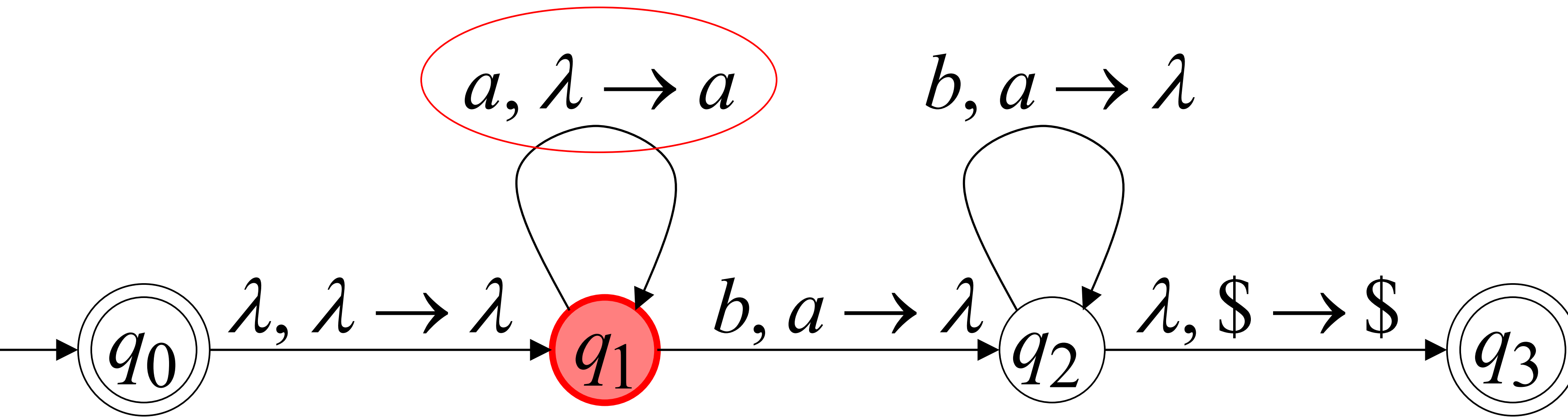
Time 4

Input

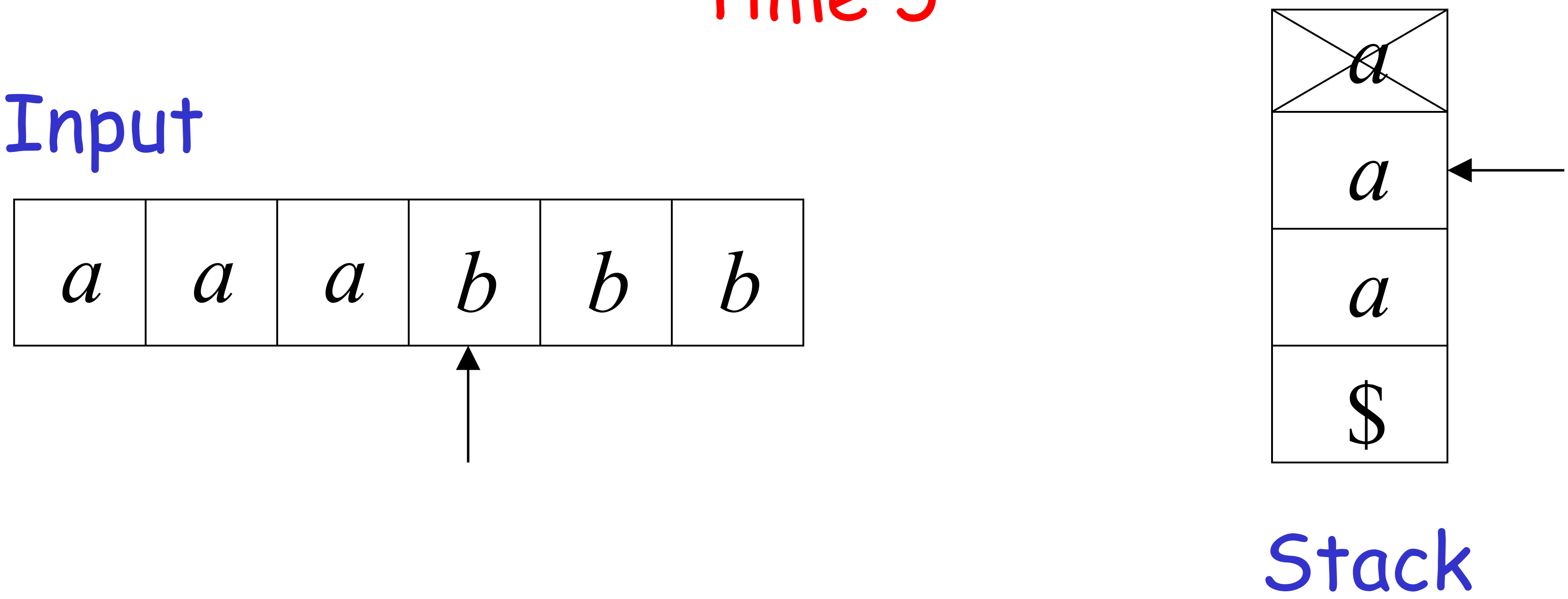


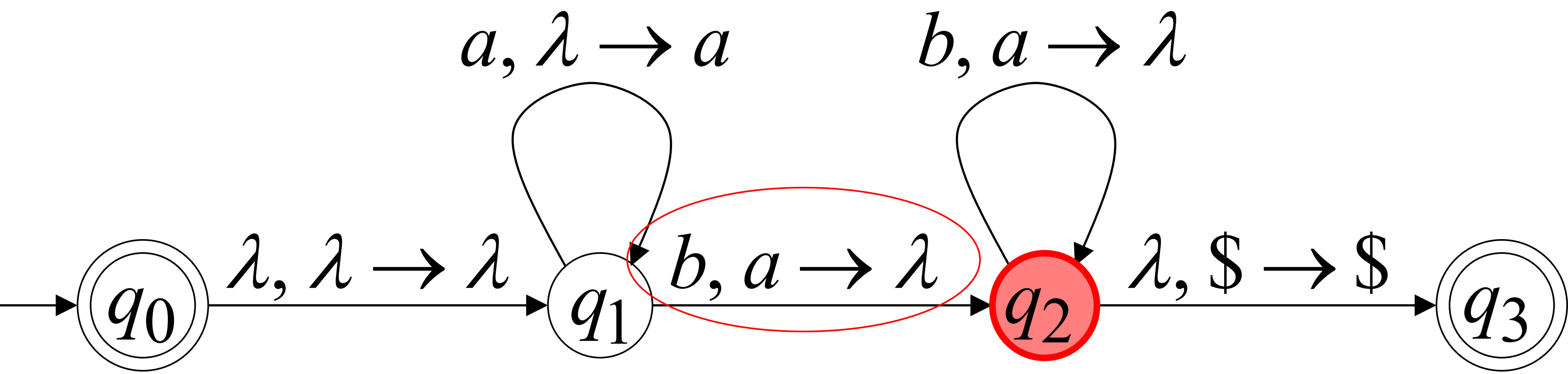


Stack

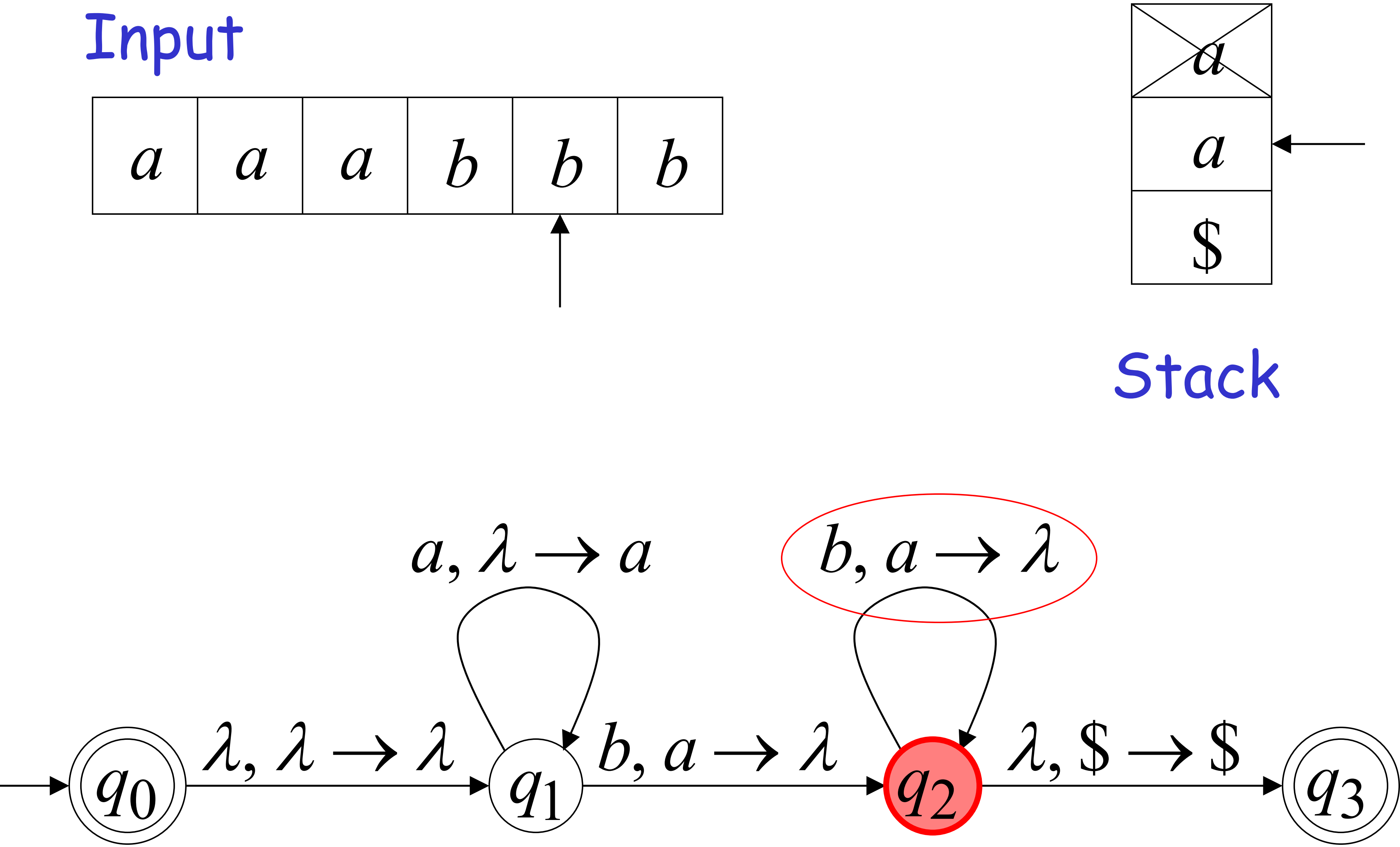


Time 5


Input
a a a b b b
a
a
a
$
Stack

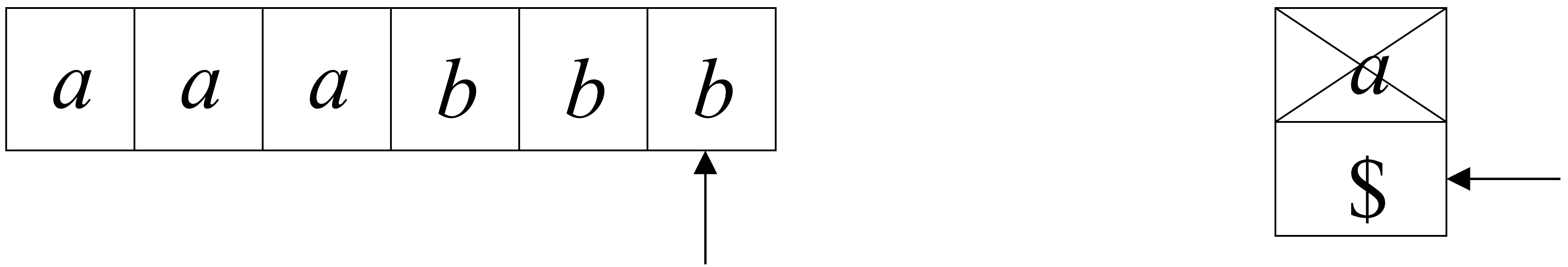

$a, \lambda \to a$
$b, a \to \lambda$
$\lambda, \lambda \to \lambda$
$b, a \to \lambda$
$\lambda, \$ \to \$$
q_0
q_1
q_2
q_3

Time 6

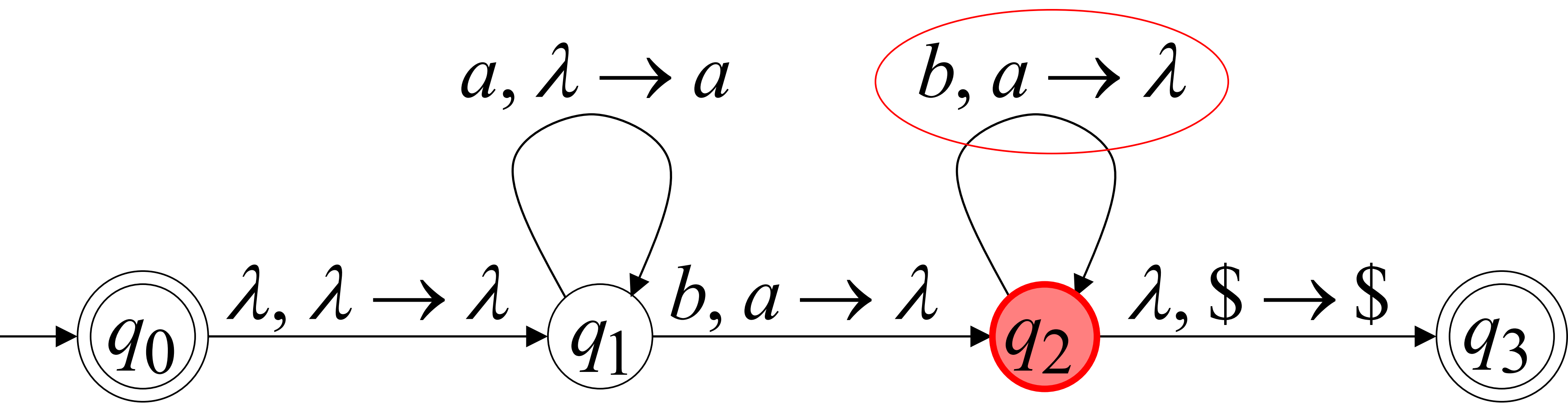

Input
a a a b b b
a
a
$
Stack
$a, \lambda \to a$
$b, a \to \lambda$
$\lambda, \lambda \to \lambda$
$b, a \to \lambda$
$\lambda, \$ \to \$$
q_0
q_1
q_2
q_3

Time 7

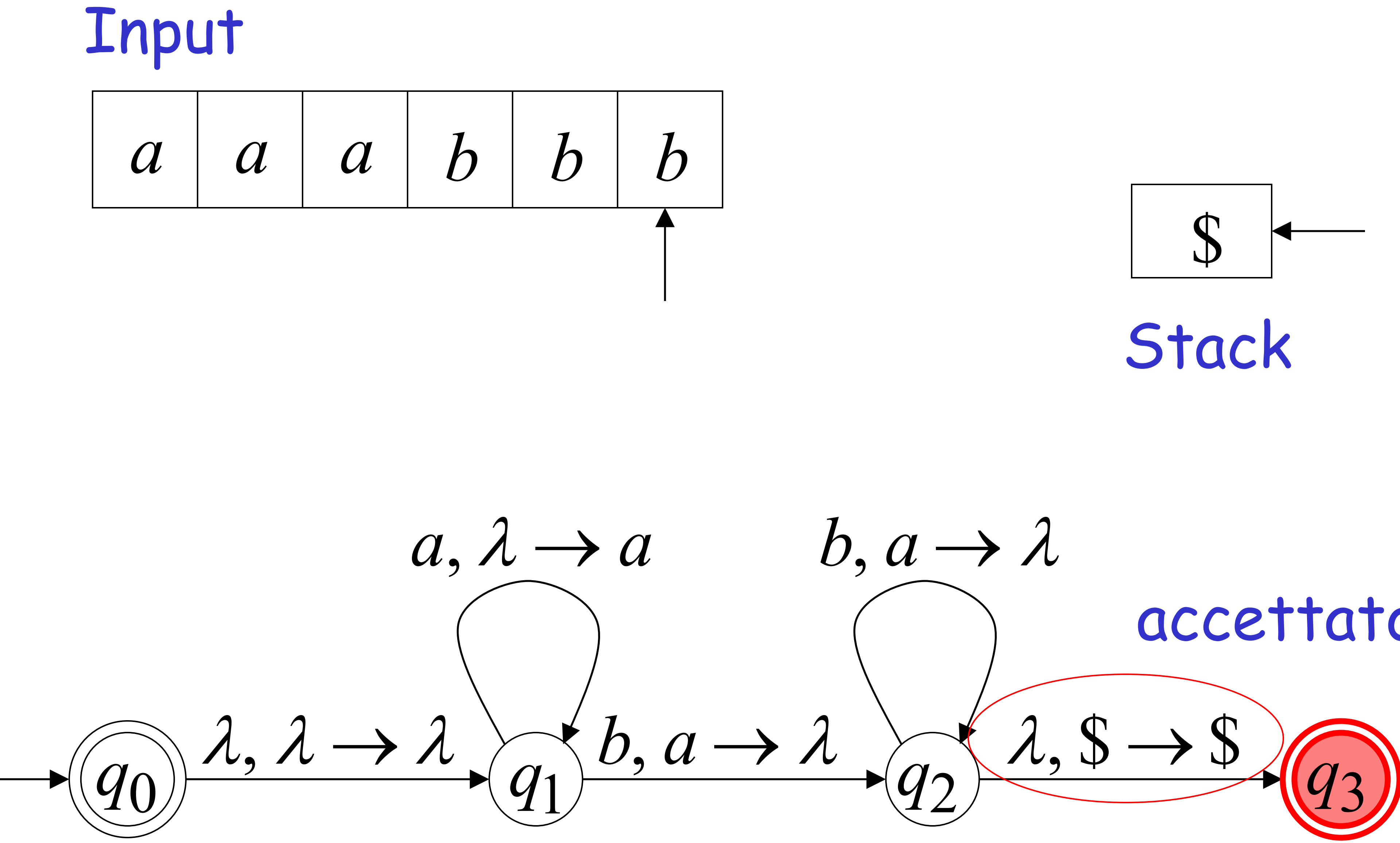
Input



Stack



Time 8



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

Una stringa è accettata se
vi è una computazione tale che:

tutti gli input sono “consumati”

E

lo stato raggiunto è uno stato di accettazione

Non teniamo conto di quello che c’è nello
Stack alla fine dello stato di accettazione

Esempio di non acettazione:

Time 0

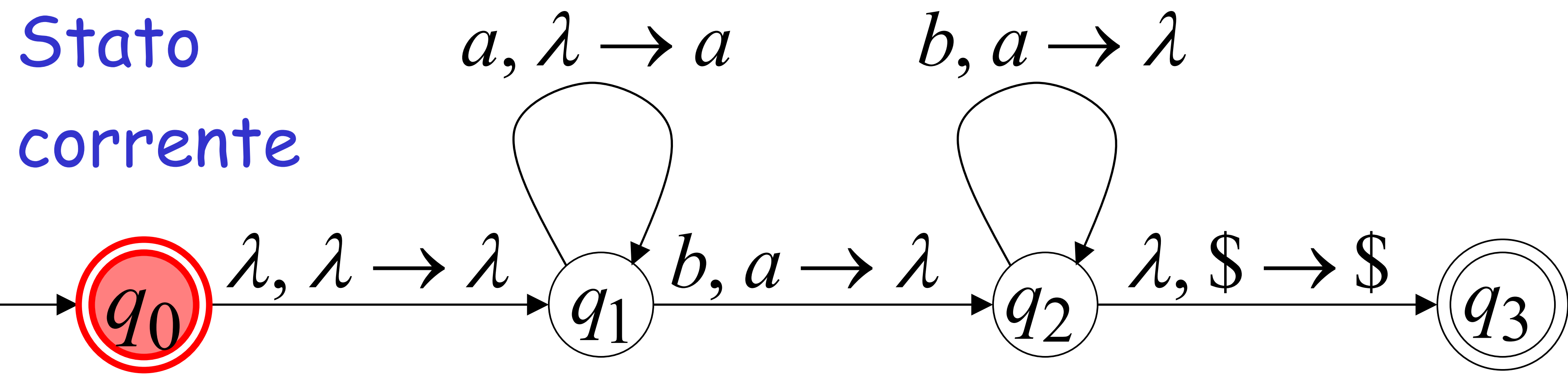
a	a	b

Input

$\$$

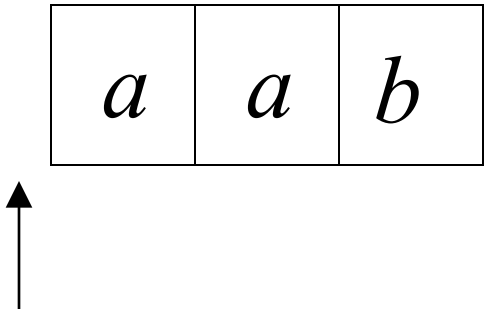
Stack

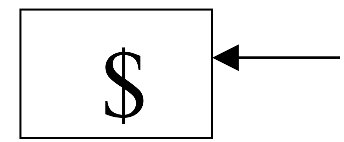
Stato corrente



non acettazione : Time 1

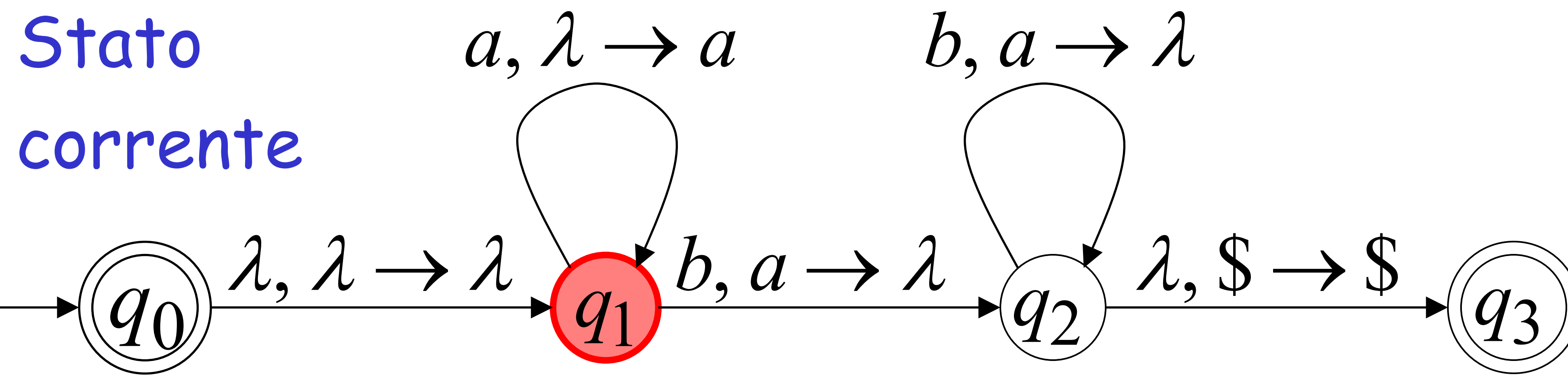
Input





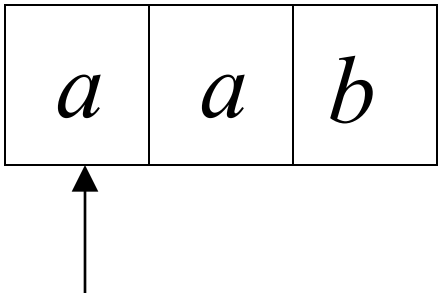
Stack

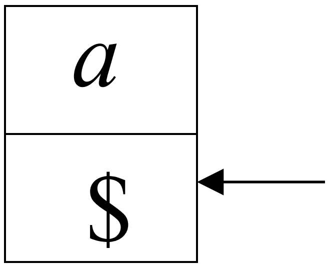
Stato corrente



non acettazione : Time 2

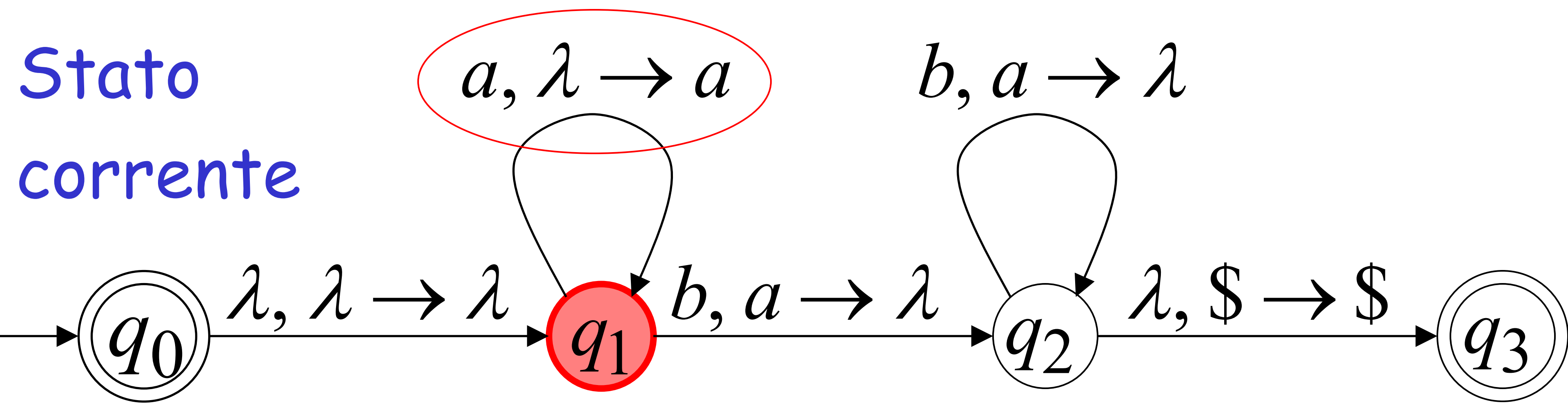
Input





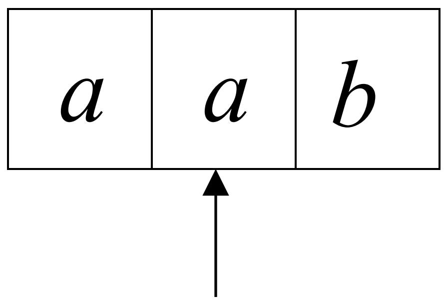
Stack

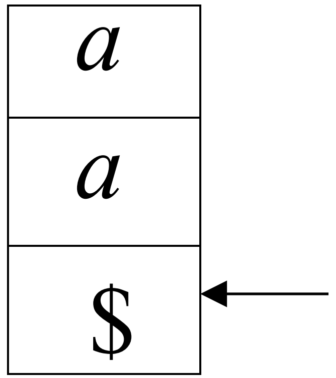
Stato corrente



non acettazione : Time 3

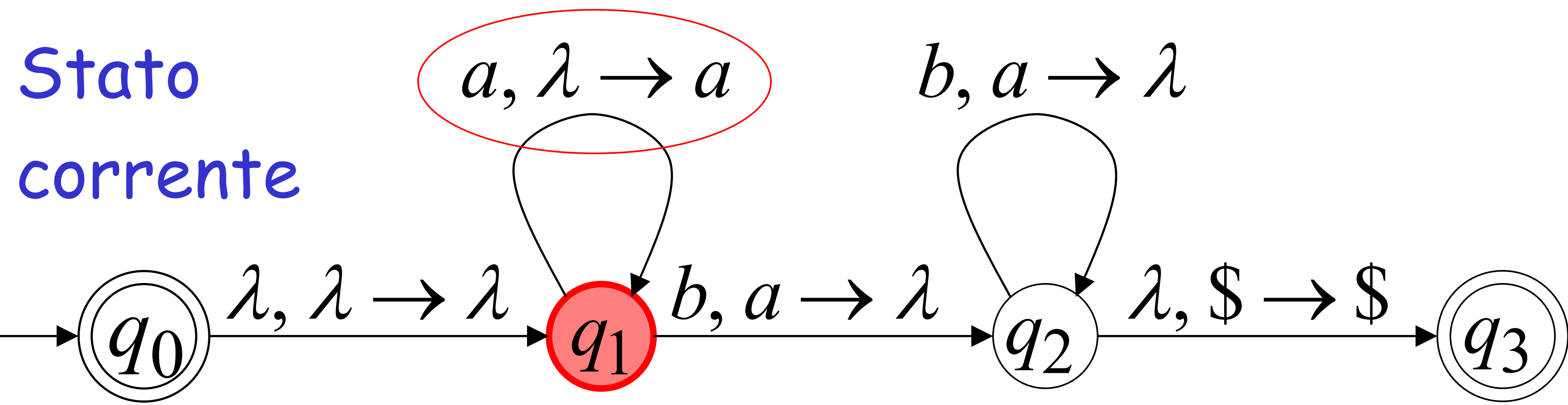
Input



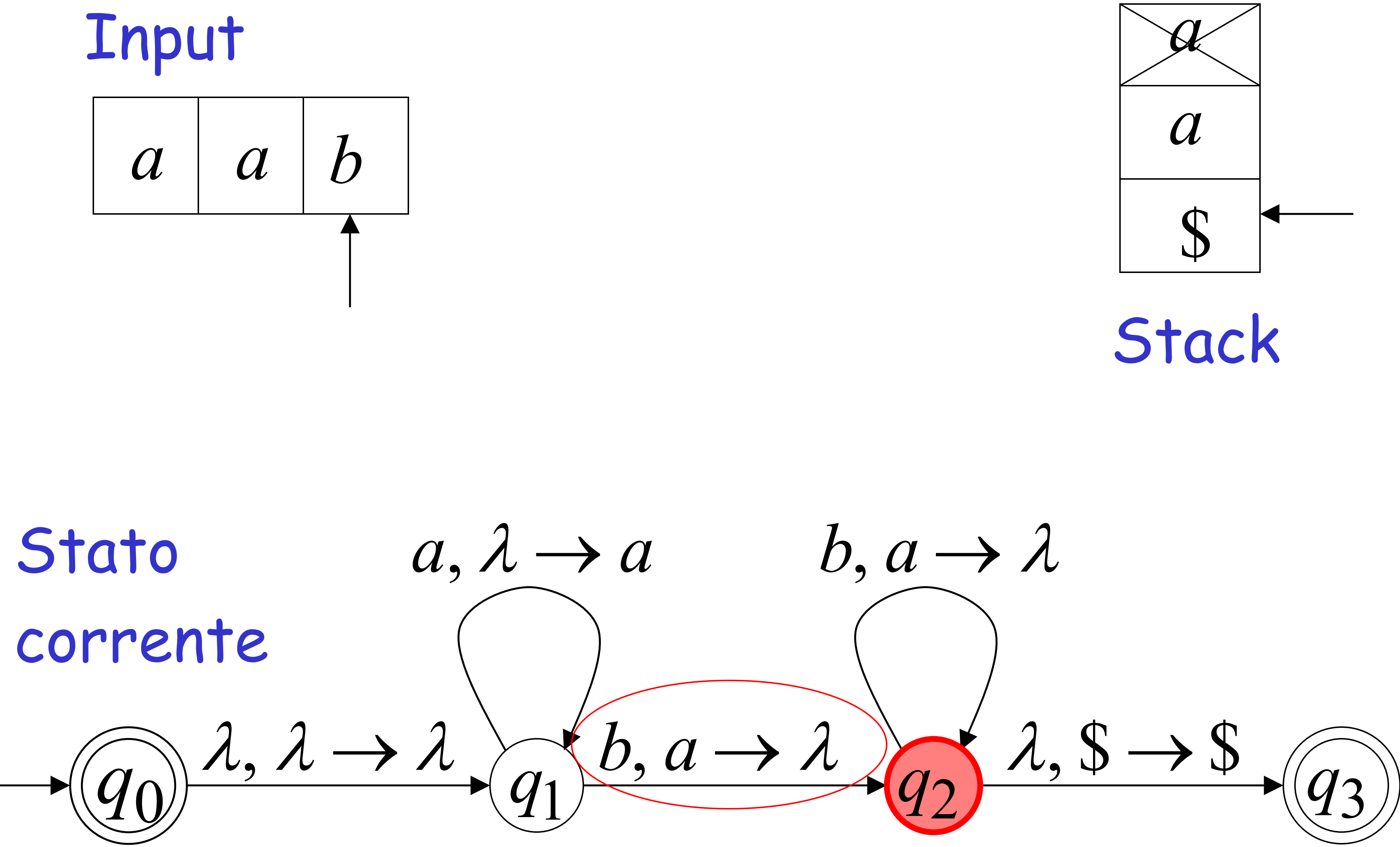


Stack

Stato corrente



non acettazione : Time 4



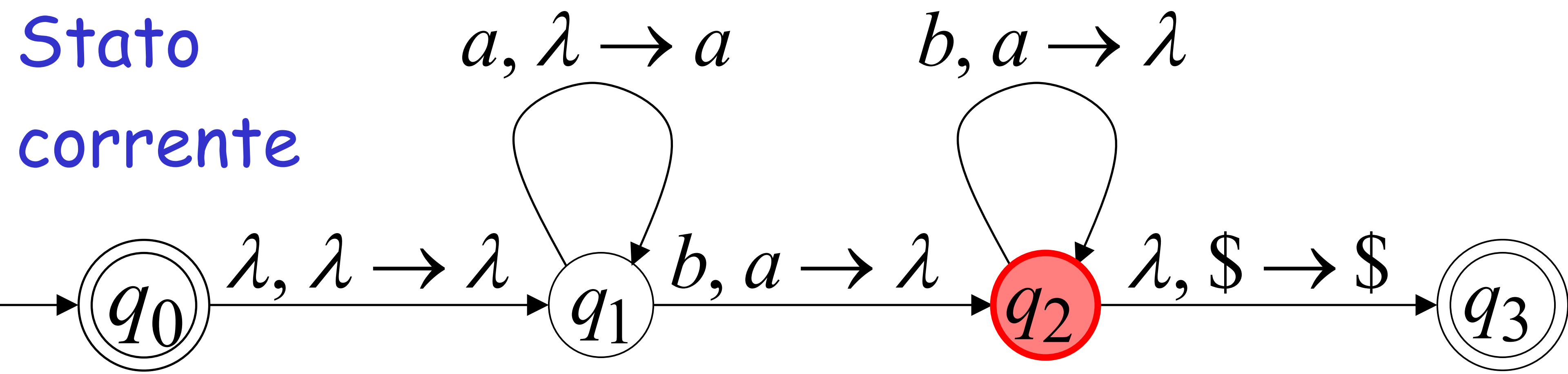
non acettazione : Time 4

Input

Stack

reject

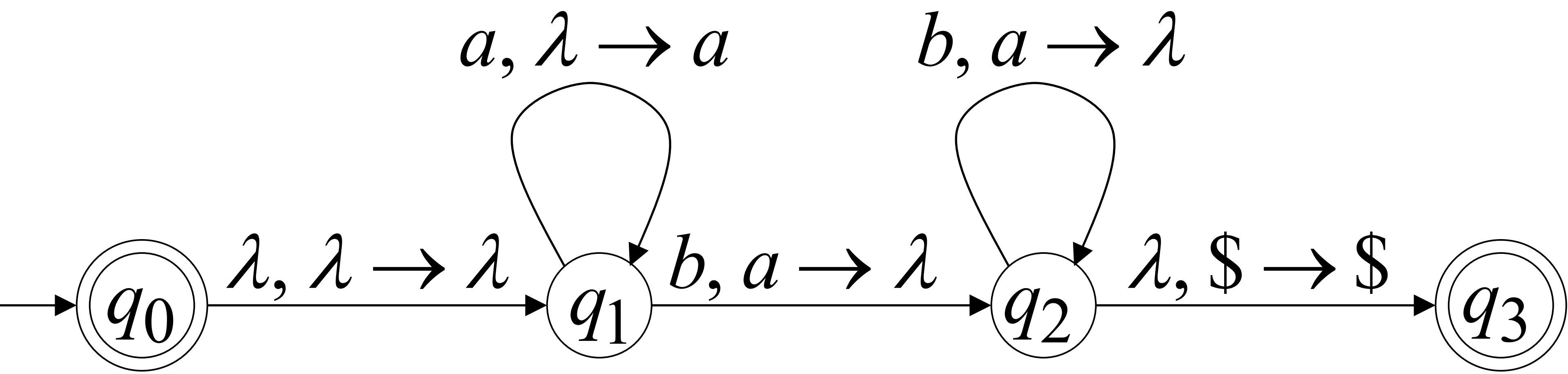
Stato corrente



Non esiste una computazione accettante

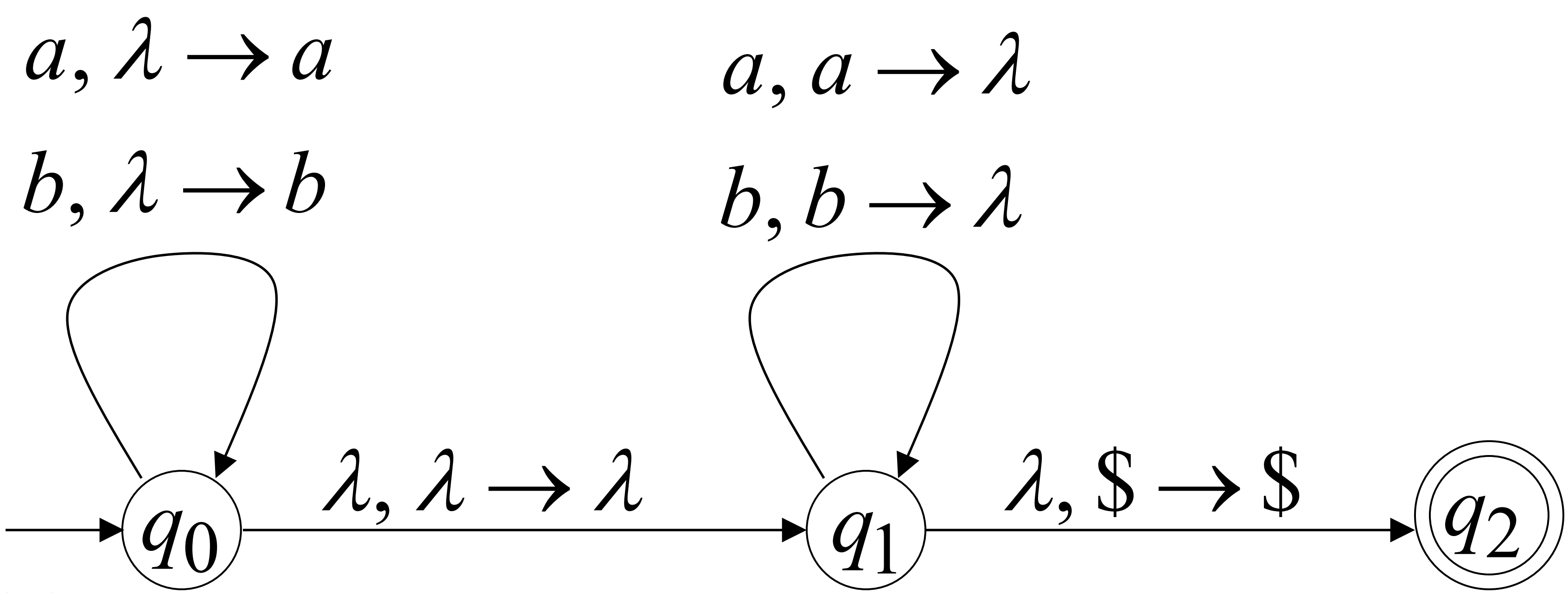
Per aab.

La stringa aab è rigettata dal PDA.



Un altro esempio di PDA

PDA M: $\quad L(M) = \{vv^R : v \in \{a,b\}^*\}$



aby aa y a y aba=xyz

abcy cba

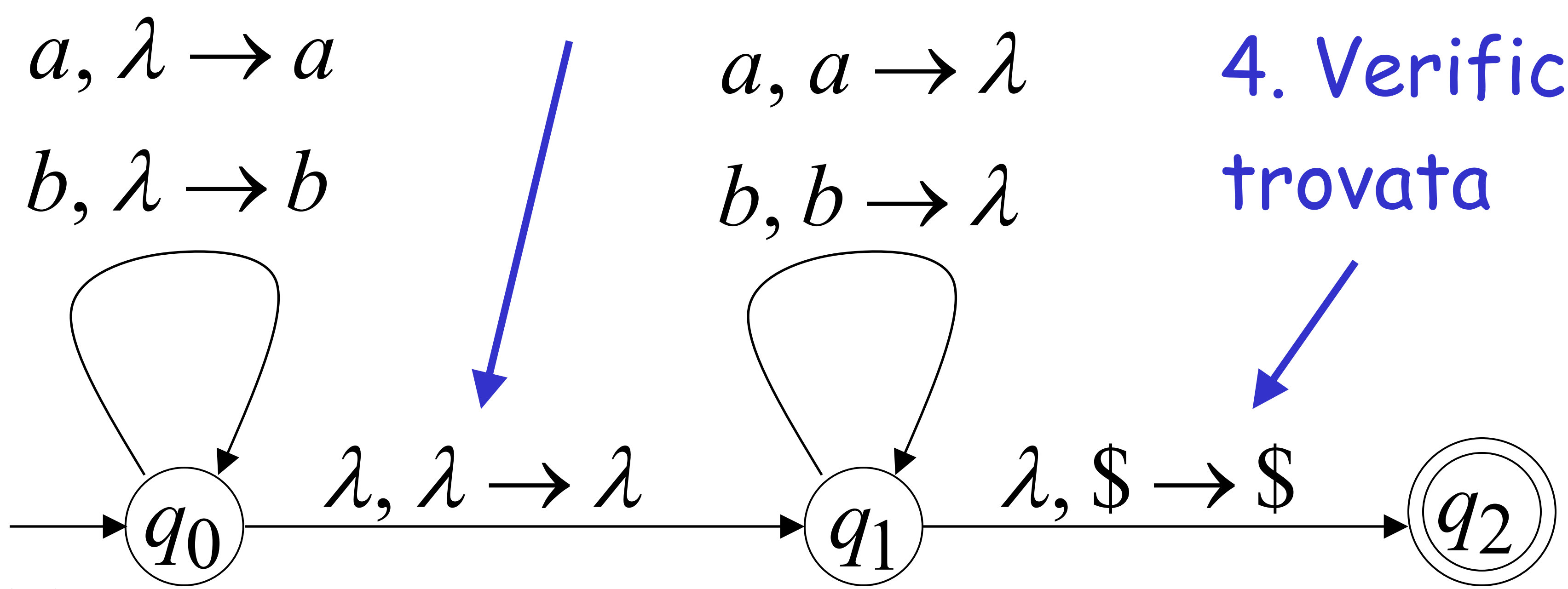
Basic Idea: $L(M) = \{vv^R : v \in \{a,b\}^*\}$

1. Push v Nello stack

2. Congettura metà Dell'input

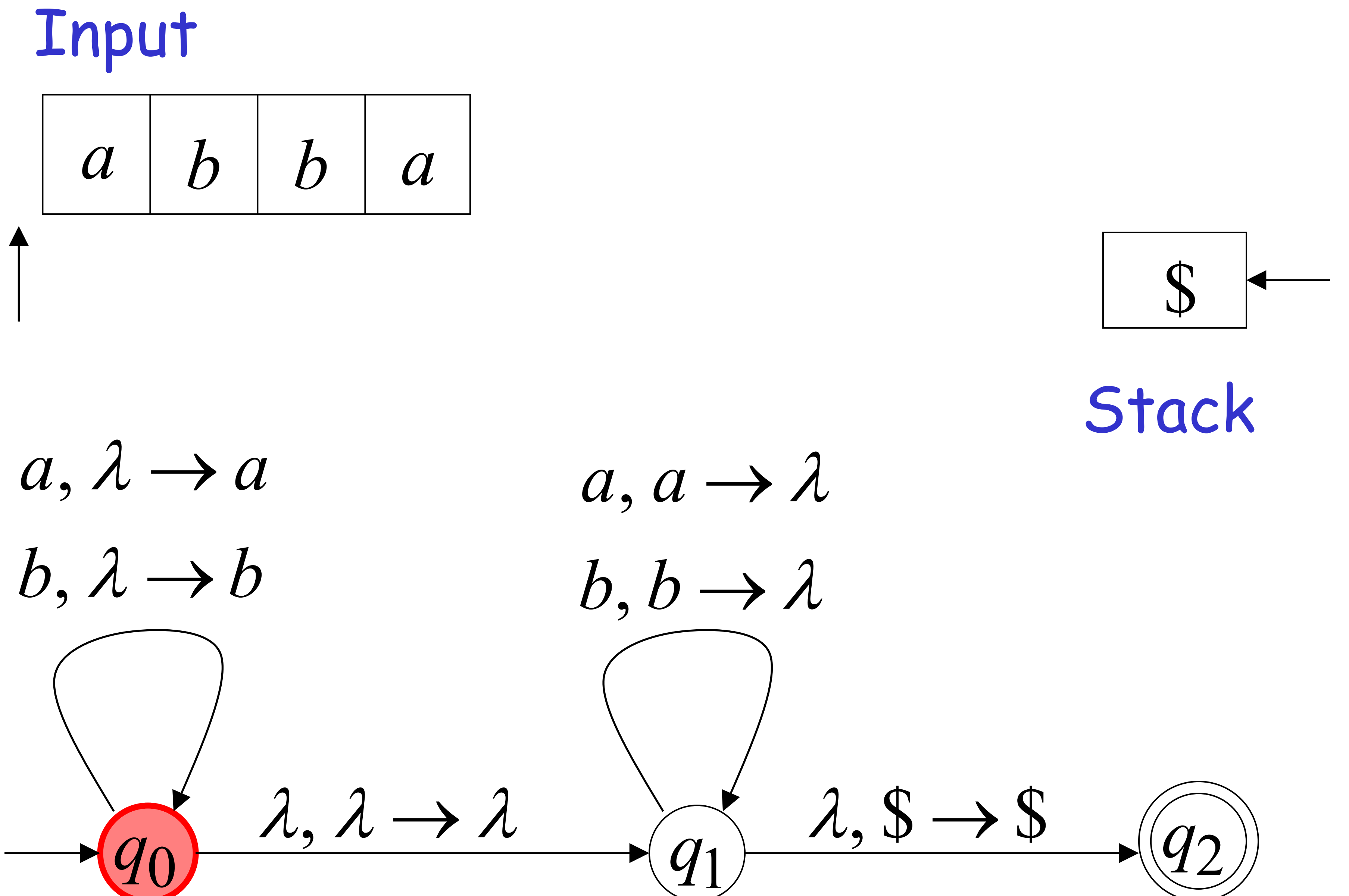
3. verifica v^R in Input con v nello stack

4. Verifica trovata

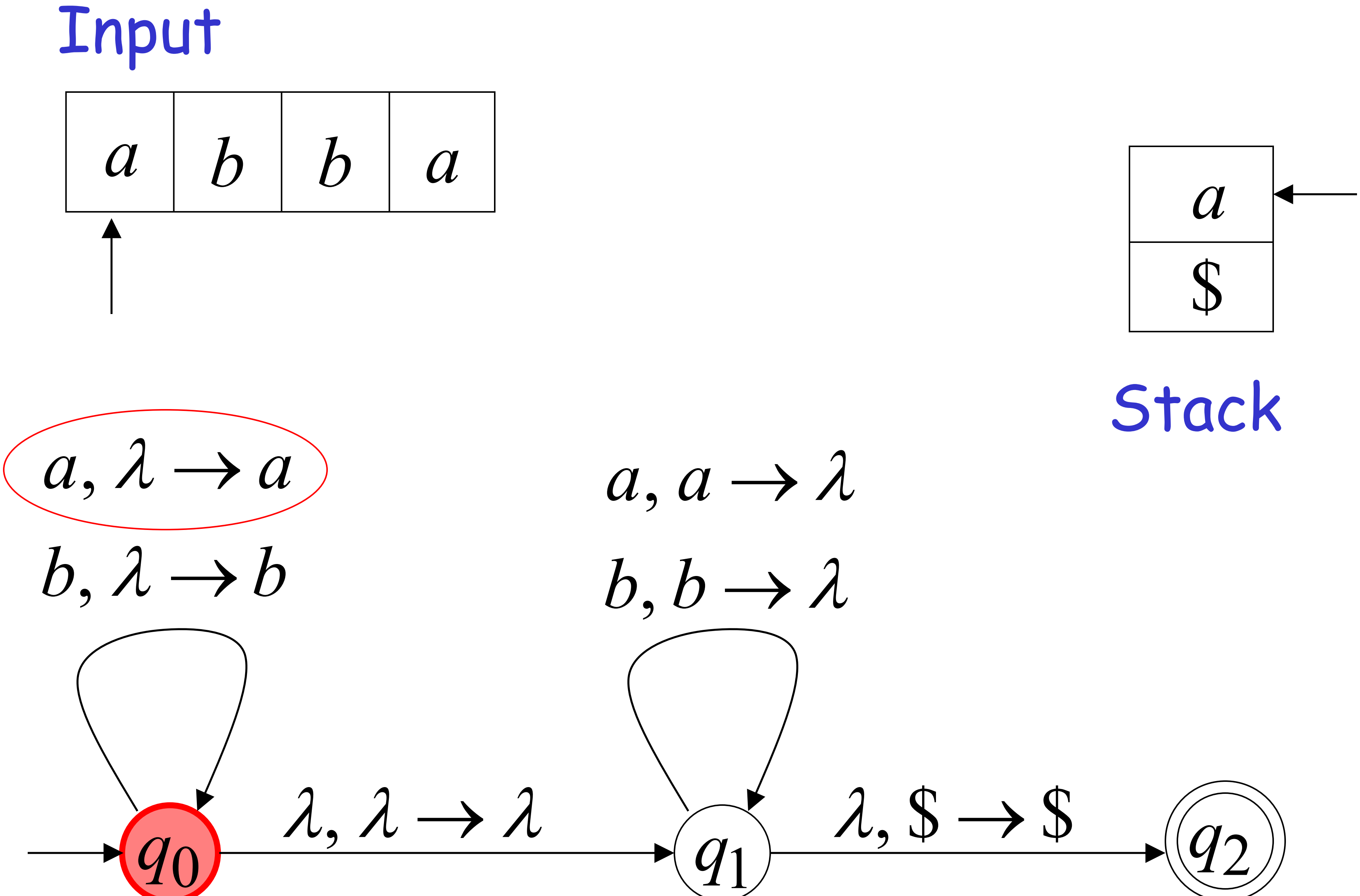


esecuzione:

Time 0

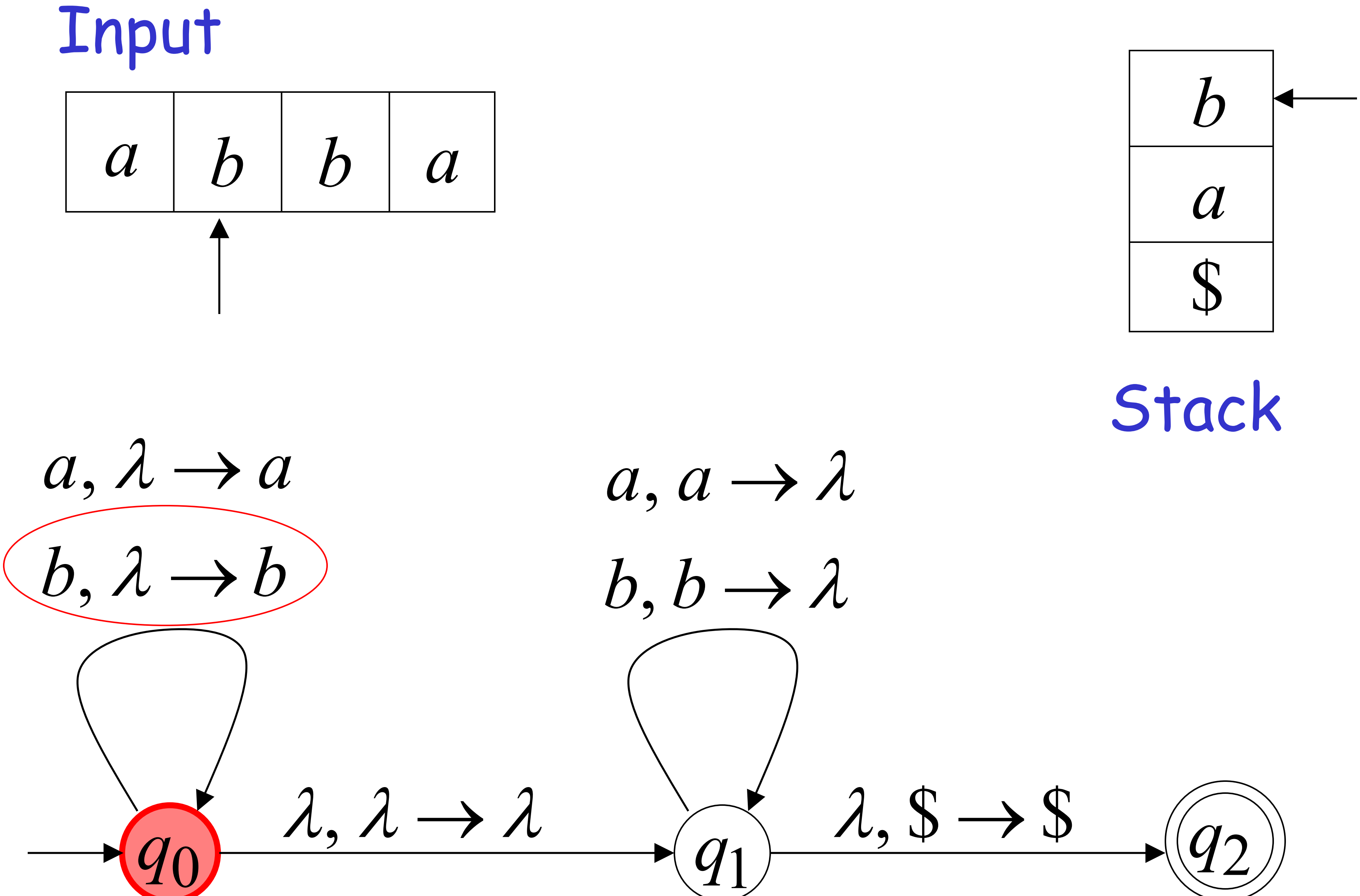


Time 1

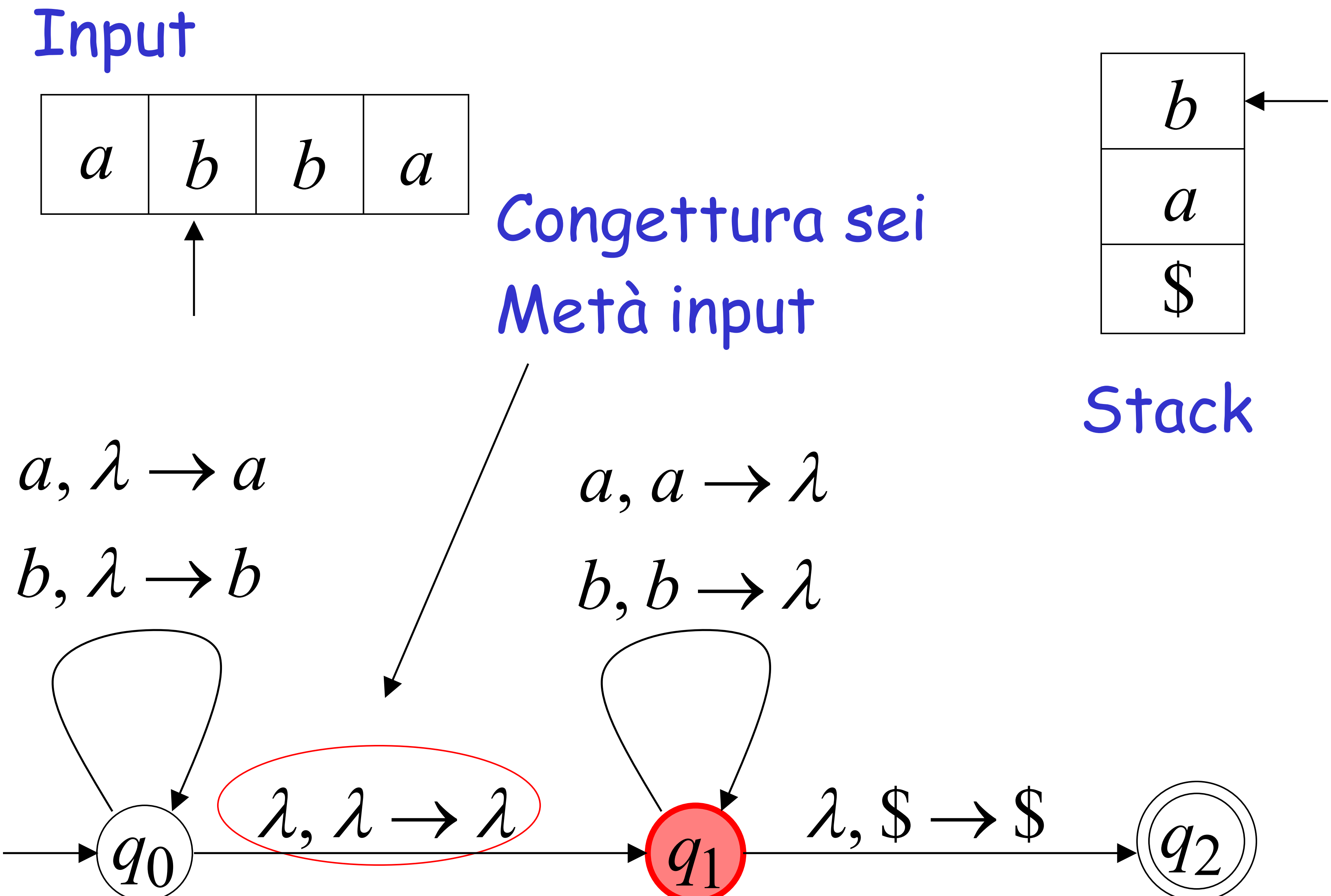


Input
a b b a
a
$
Stack
$a, \lambda \to a$
$b, \lambda \to b$
$a, a \to \lambda$
$b, b \to \lambda$
$\lambda, \lambda \to \lambda$
$\lambda, \$ \to \$$
q_0
q_1
q_2

Time 2

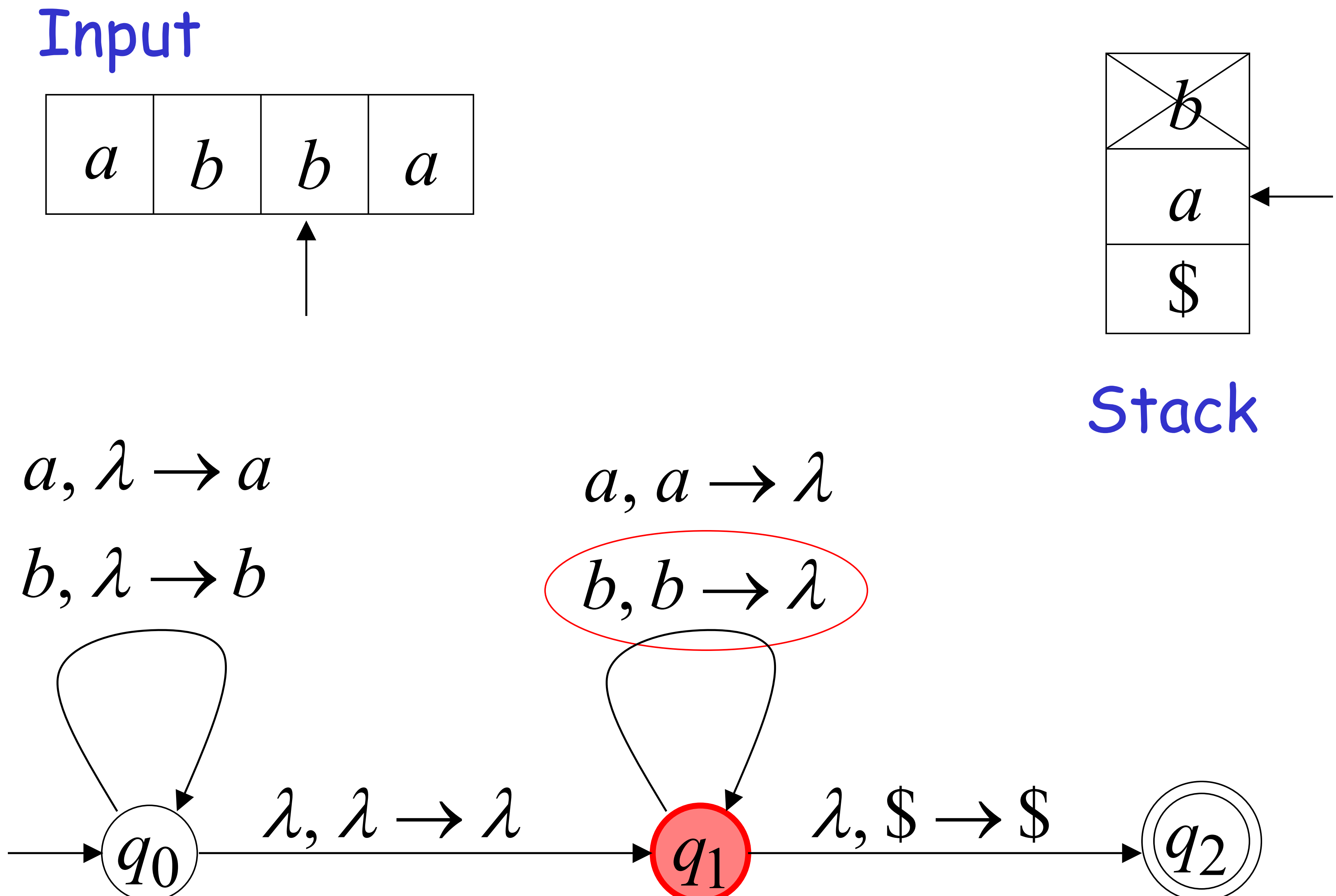

Input
a b b a
b
a
$\$$
Stack
$a, \lambda \to a$
$b, \lambda \to b$
$a, a \to \lambda$
$b, b \to \lambda$
$\lambda, \lambda \to \lambda$
$\lambda, \$ \to \$$
q_0
q_1
q_2

Time 3

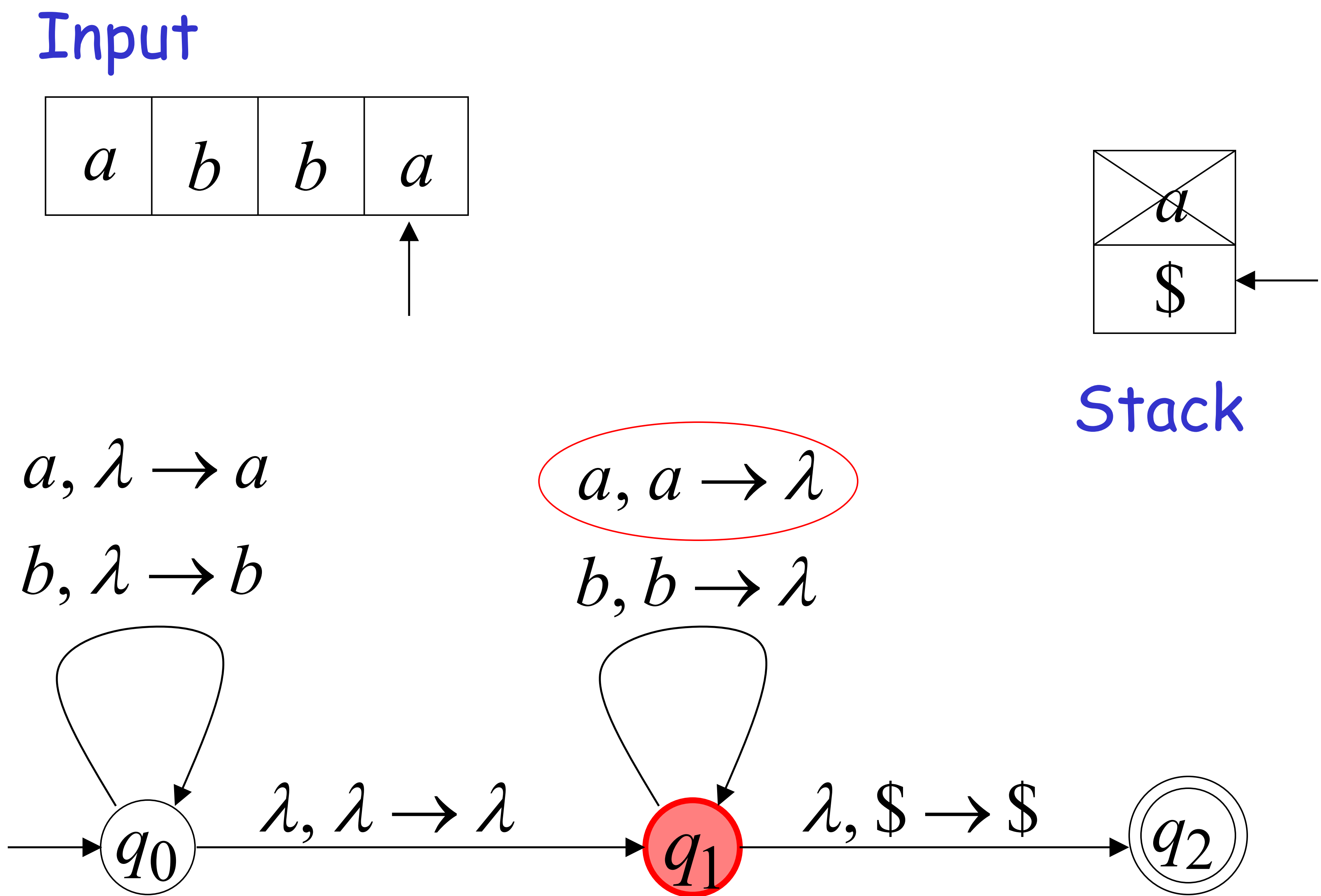


Input
a
b
b
a
Congettura sei
Metà input
b
a
$
Stack
$a, \lambda \to a$
$b, \lambda \to b$
$a, a \to \lambda$
$b, b \to \lambda$
$\lambda, \lambda \to \lambda$
$\lambda, \$ \to \$$
q_0
q_1
q_2

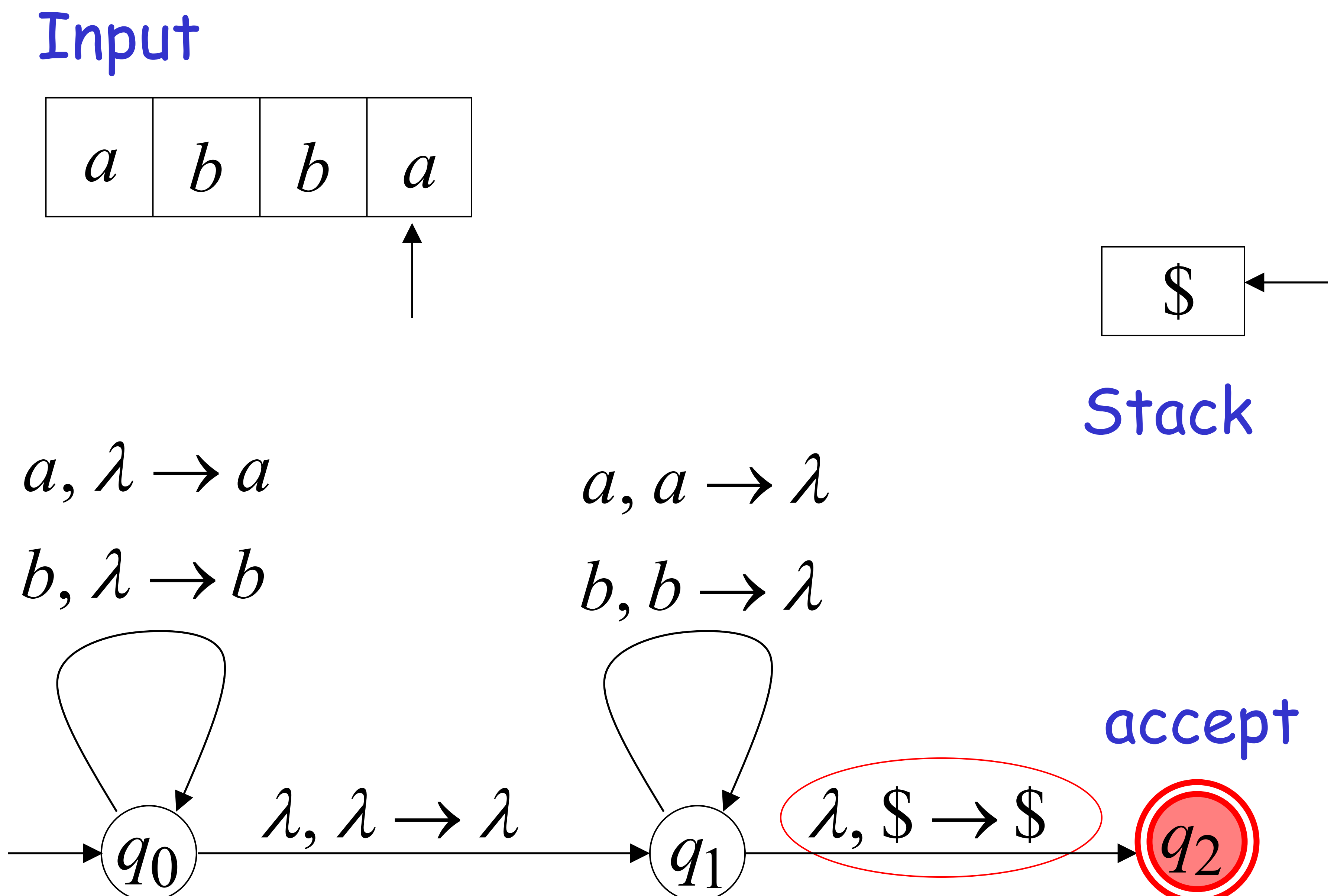
Time 4



Time 5

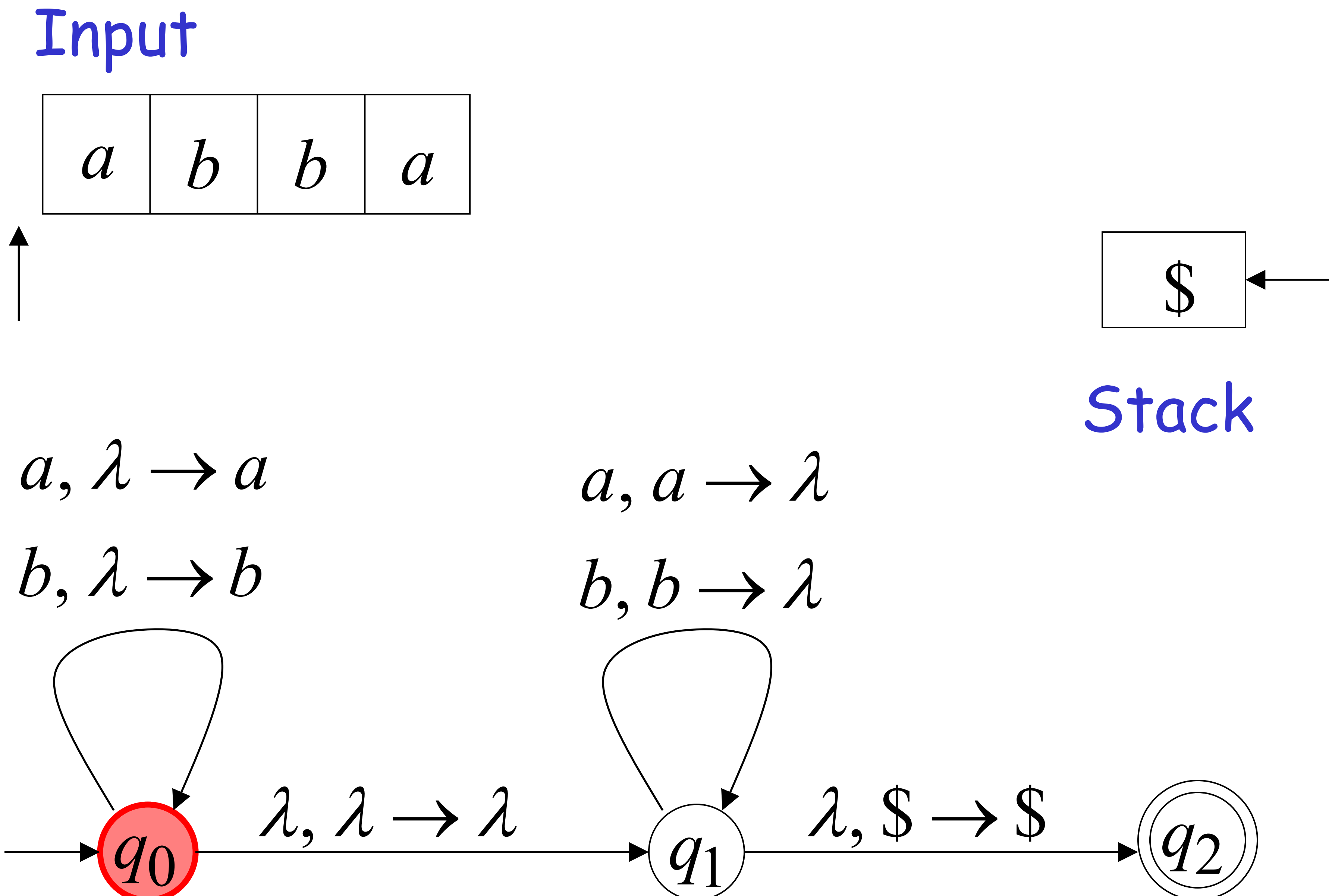

Input
a
b
b
a
$\cancel{a}$
$
Stack
$a, \lambda \to a$
$b, \lambda \to b$
$a, a \to \lambda$
$b, b \to \lambda$
$\lambda, \lambda \to \lambda$
$\lambda, \$ \to \$$
q_0
q_1
q_2

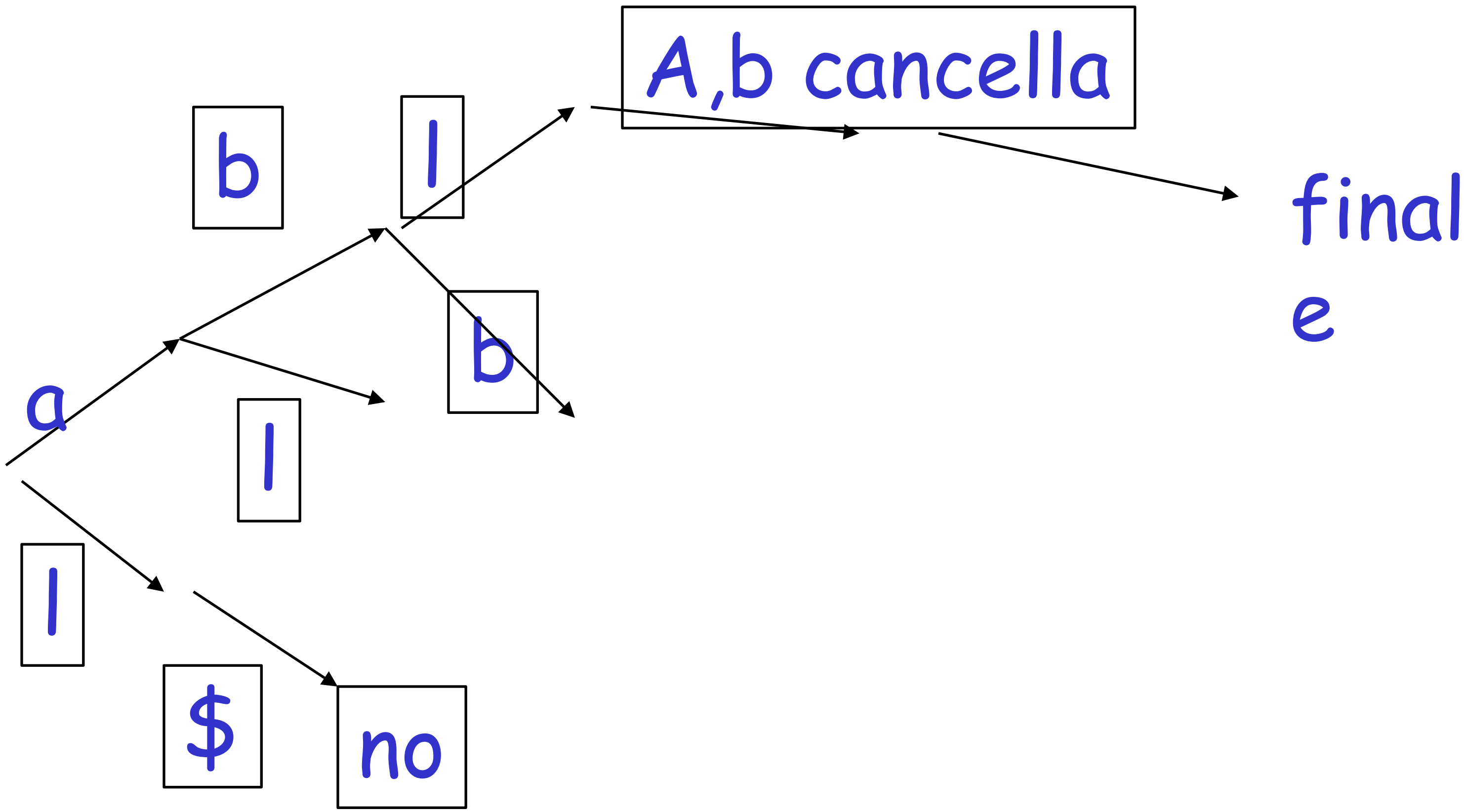
Time 6



esecuzione:

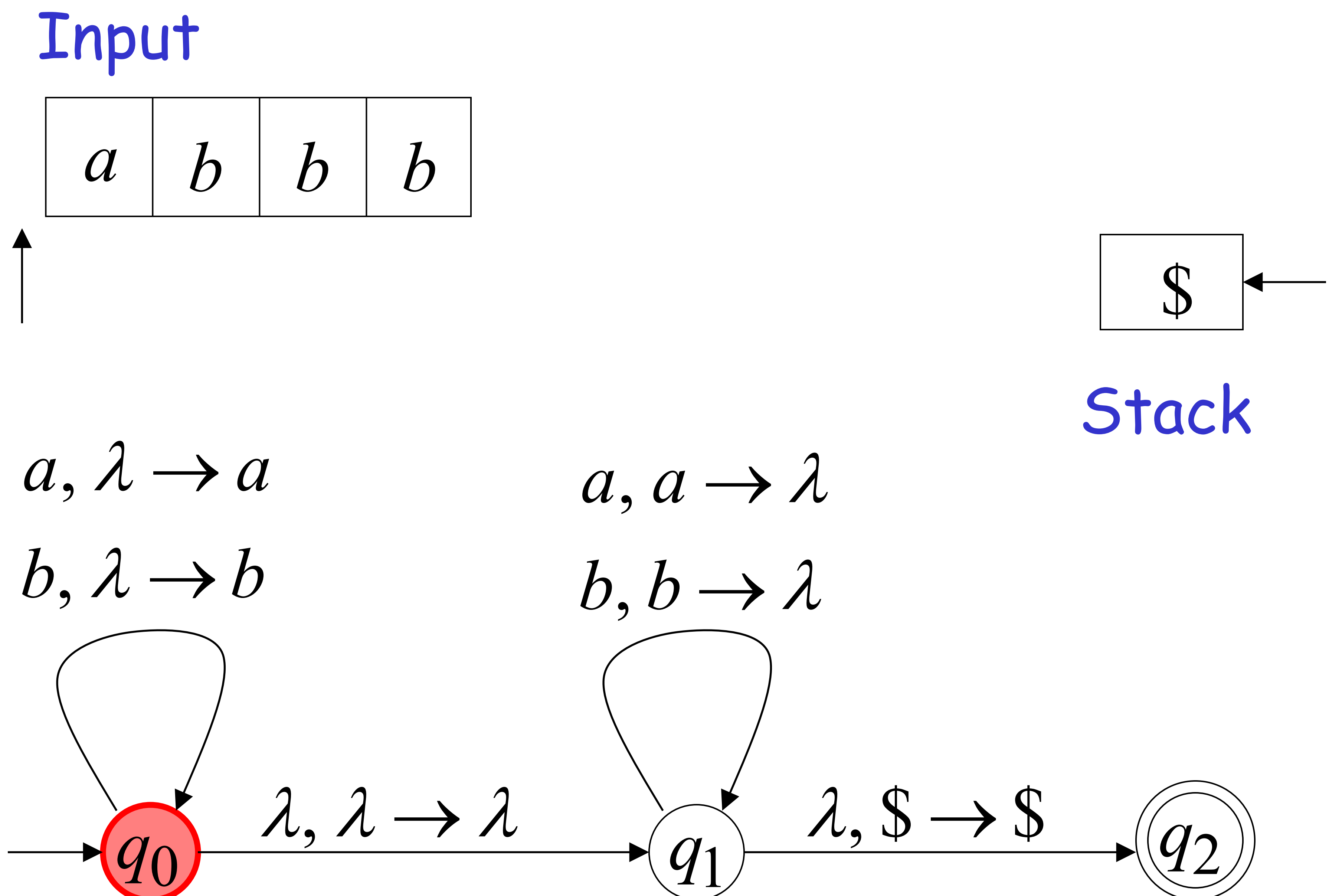
Time 0



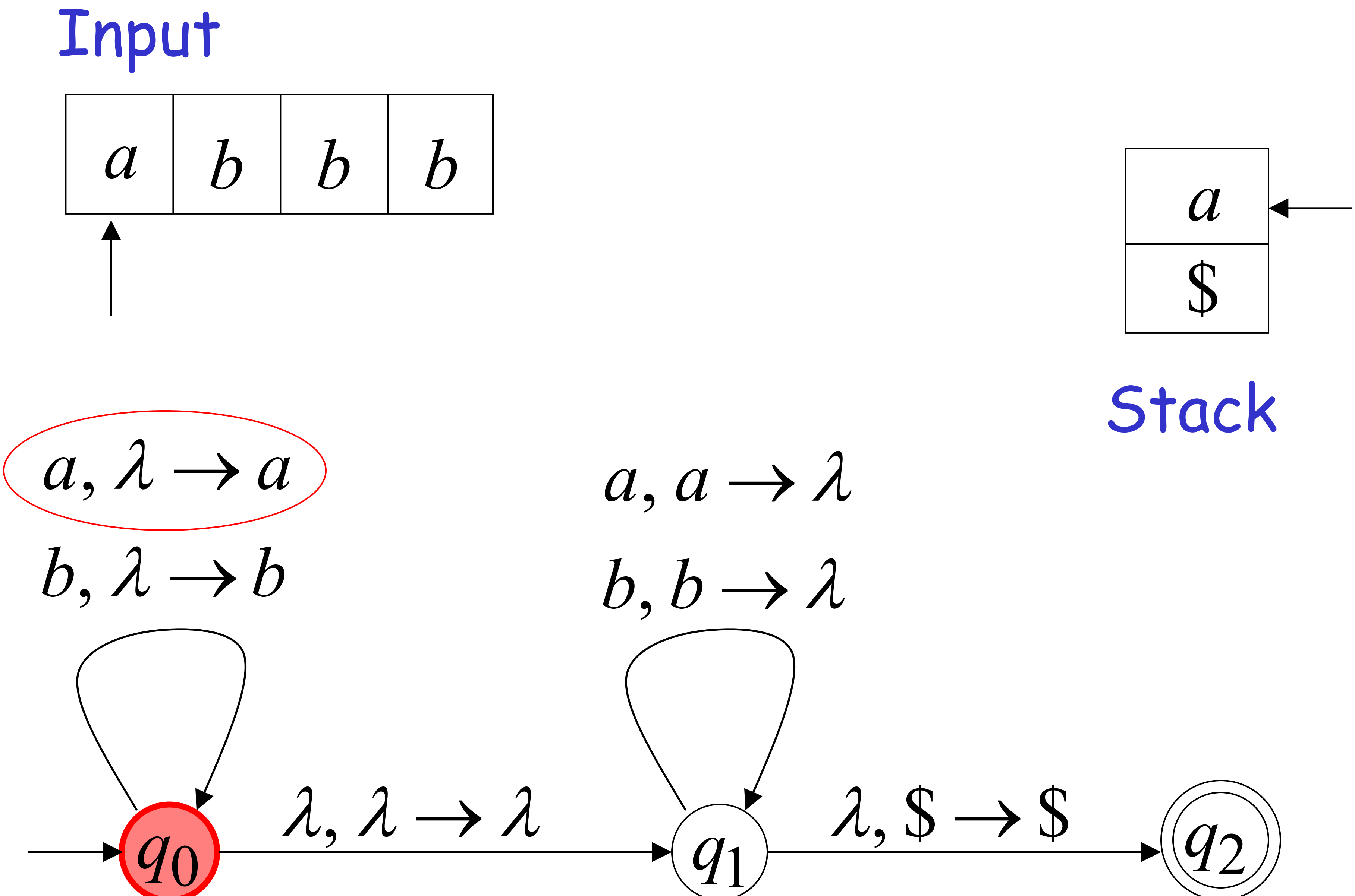

A,b cancella
b
l
final
e
b
a
l
l
$
no

Altro esempio:

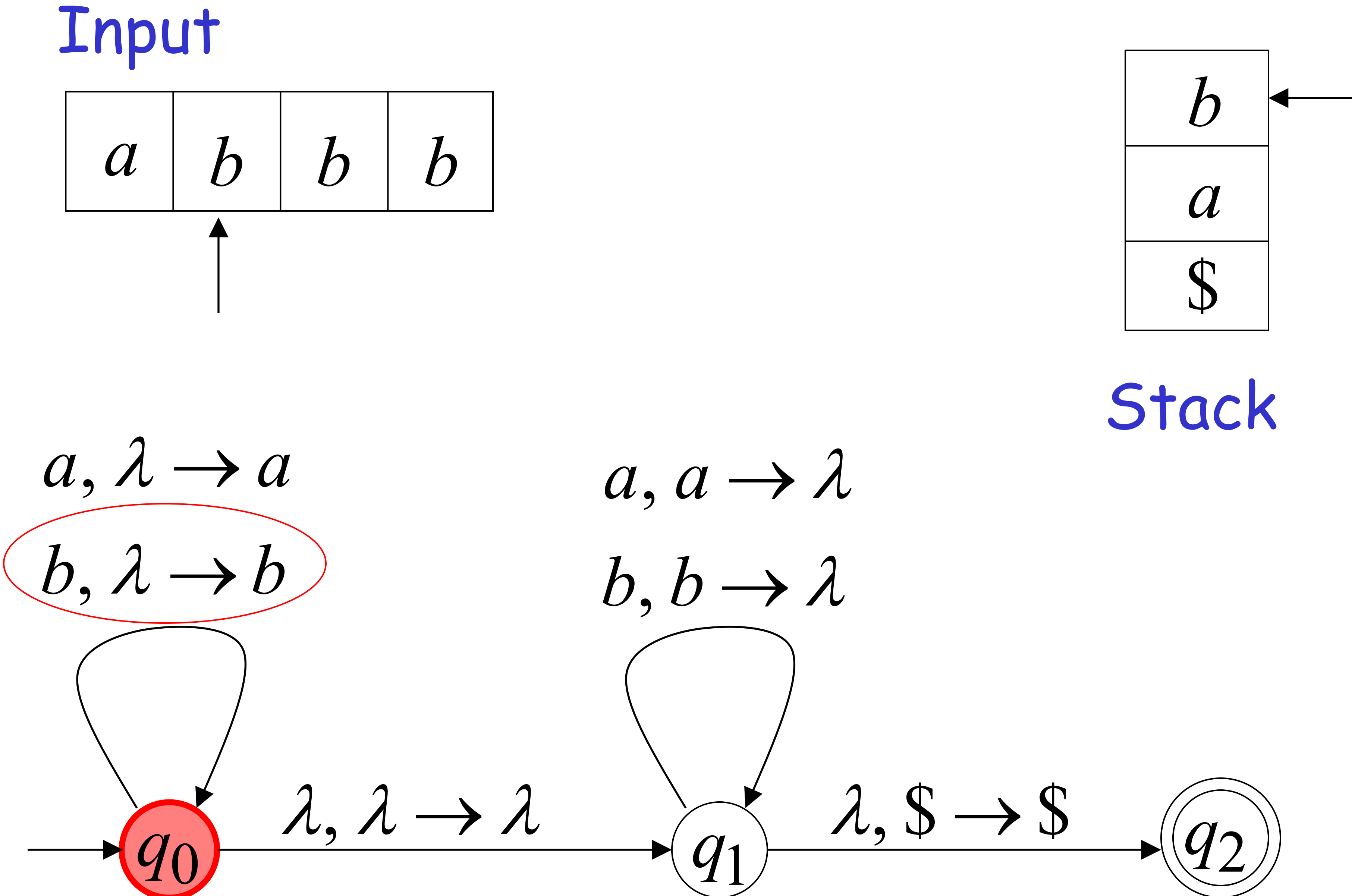
Time 0



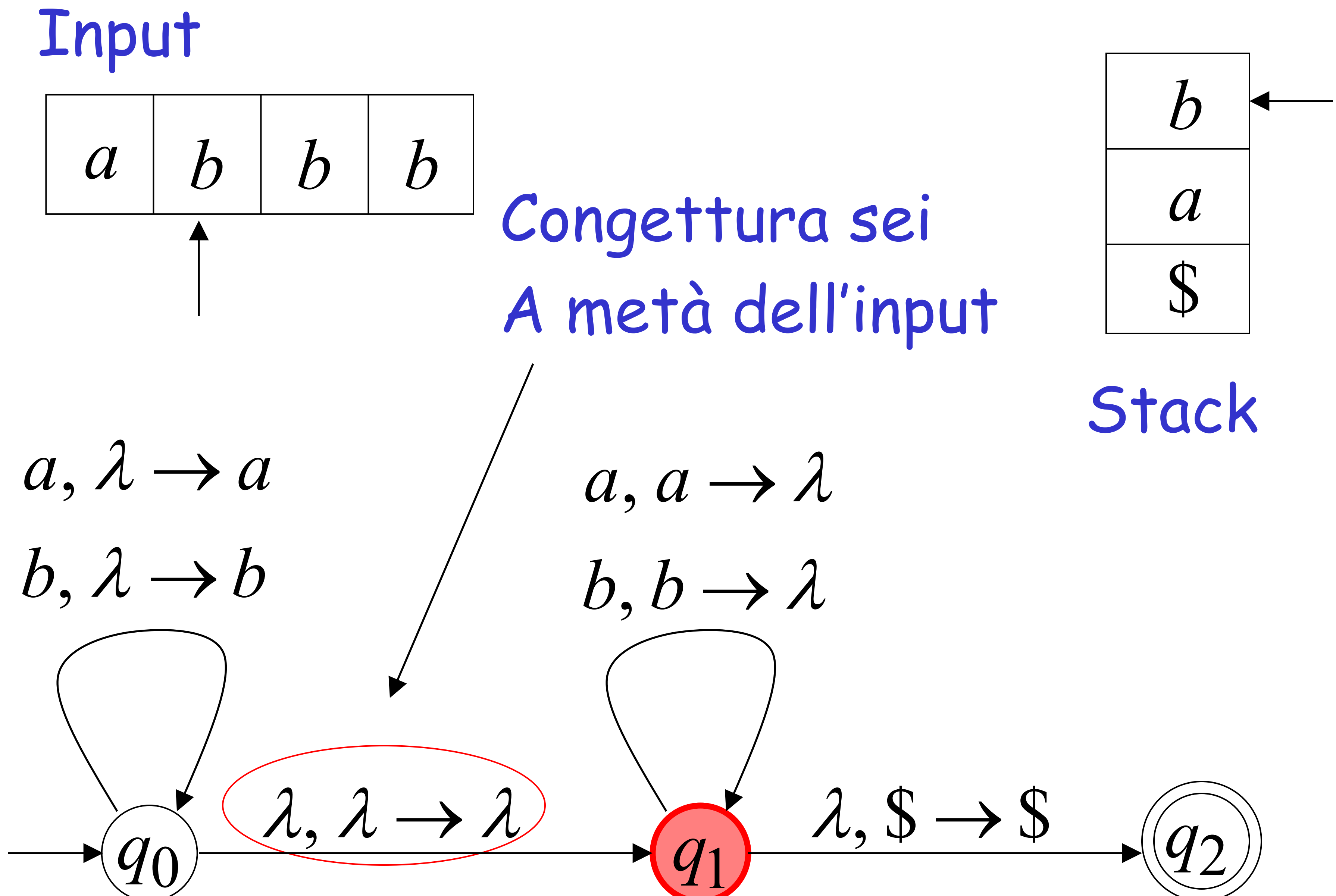
Time 1



Time 2

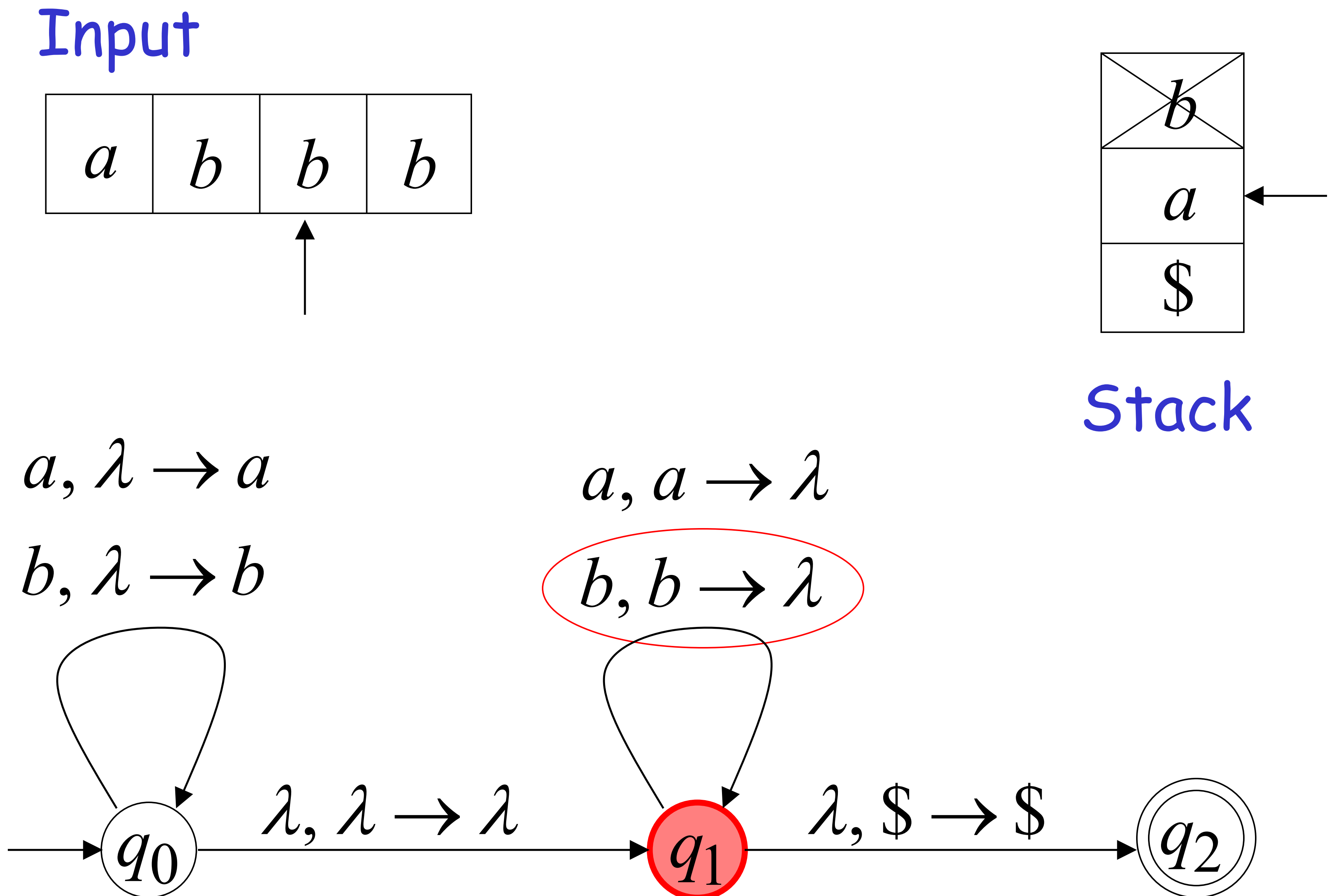


Time 3



Input
a
b
b
b
Congettura sei
A metà dell'input
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
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 5

Input

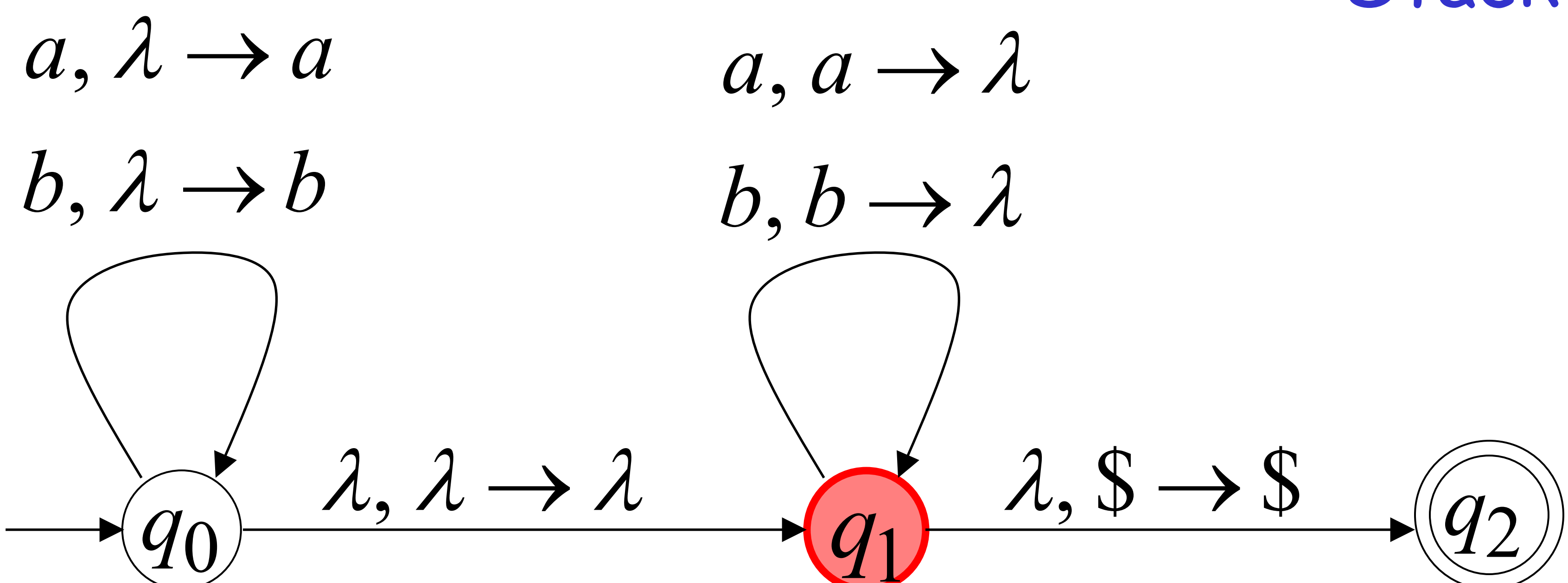
Non ci sono possibili transizioni.

a	b	b	b

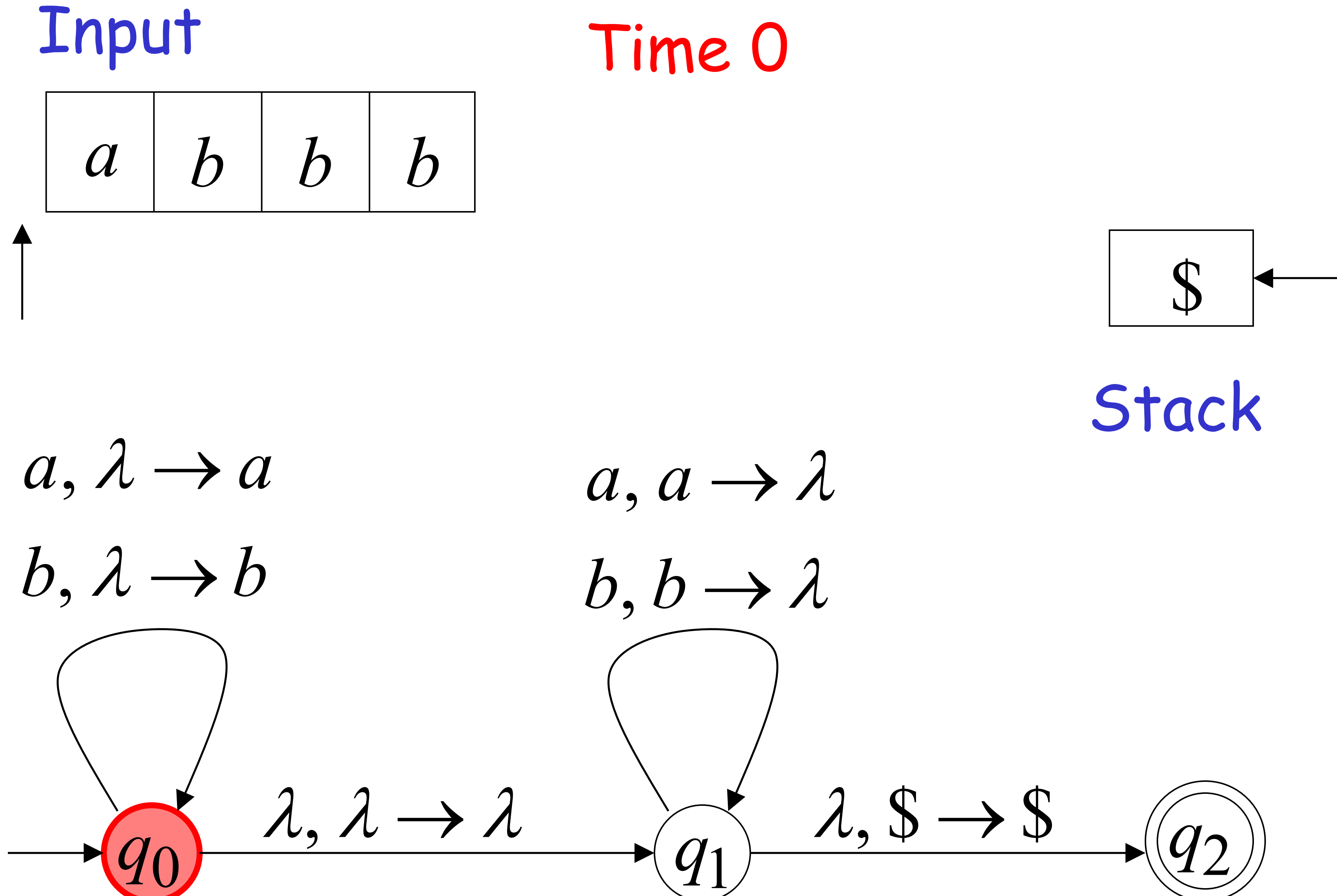
Input non è stato consumato

a
$\$$

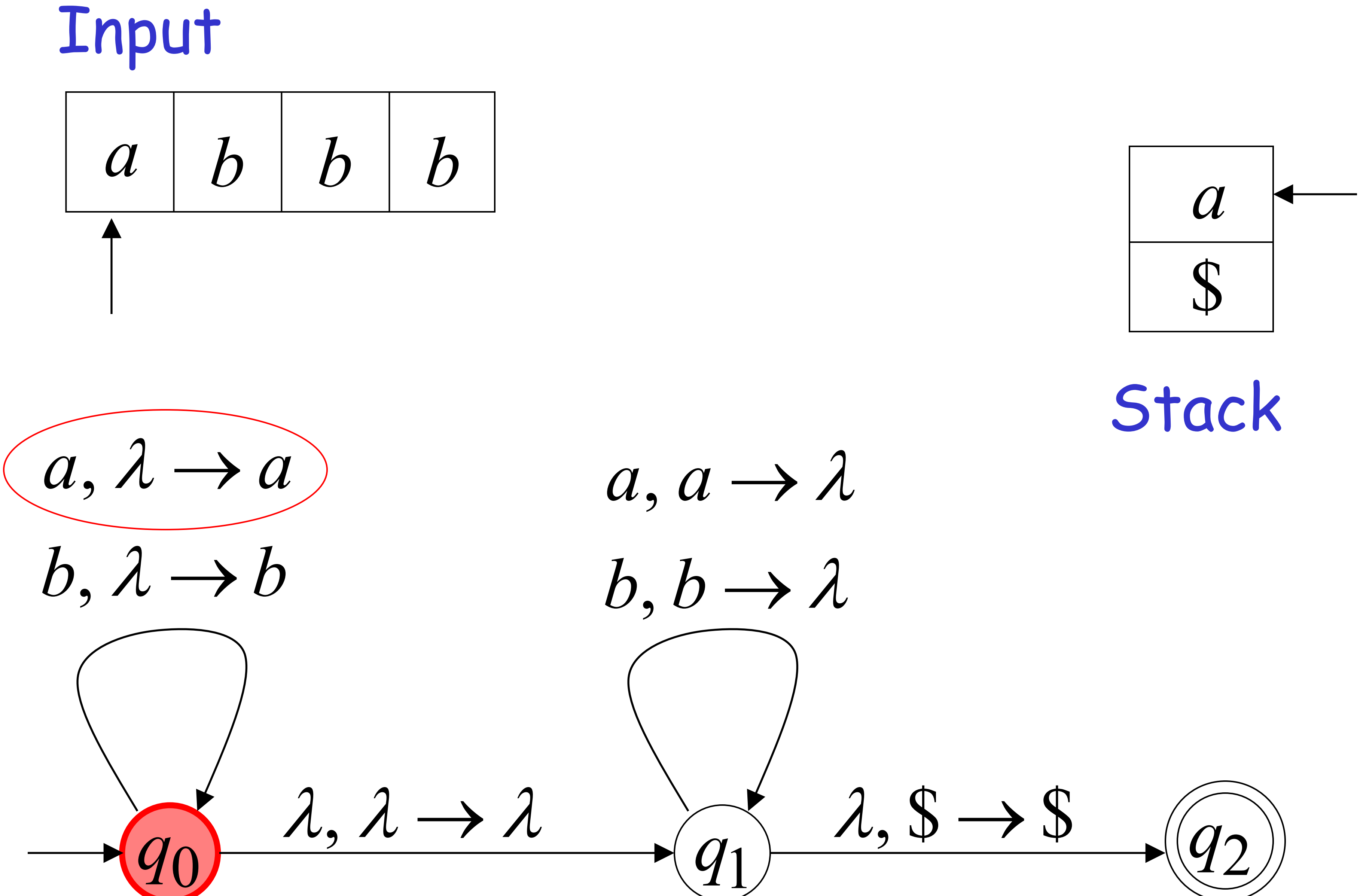
Stack



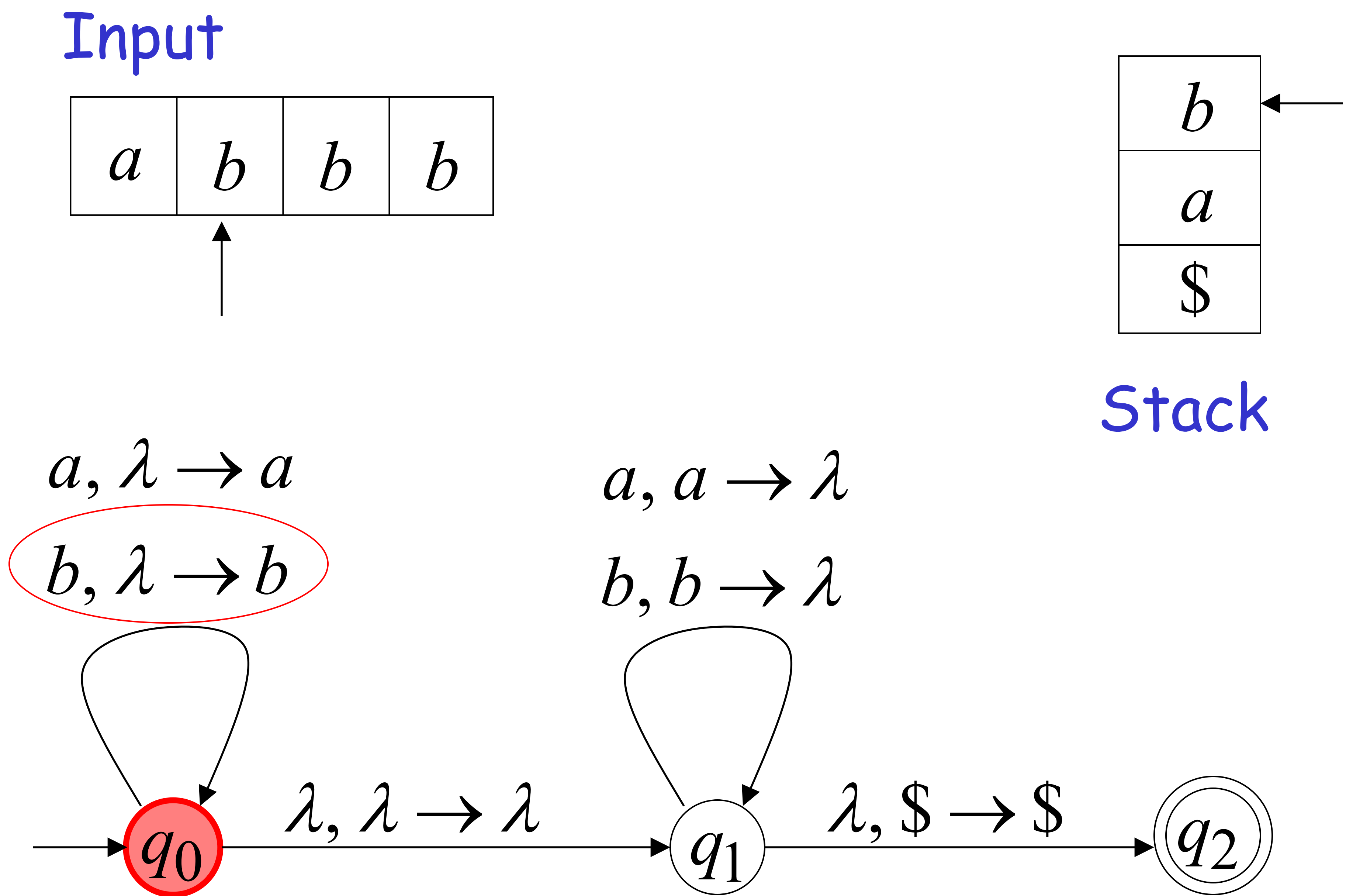
Un altra computazione sulla stessa stringa:



Time 1

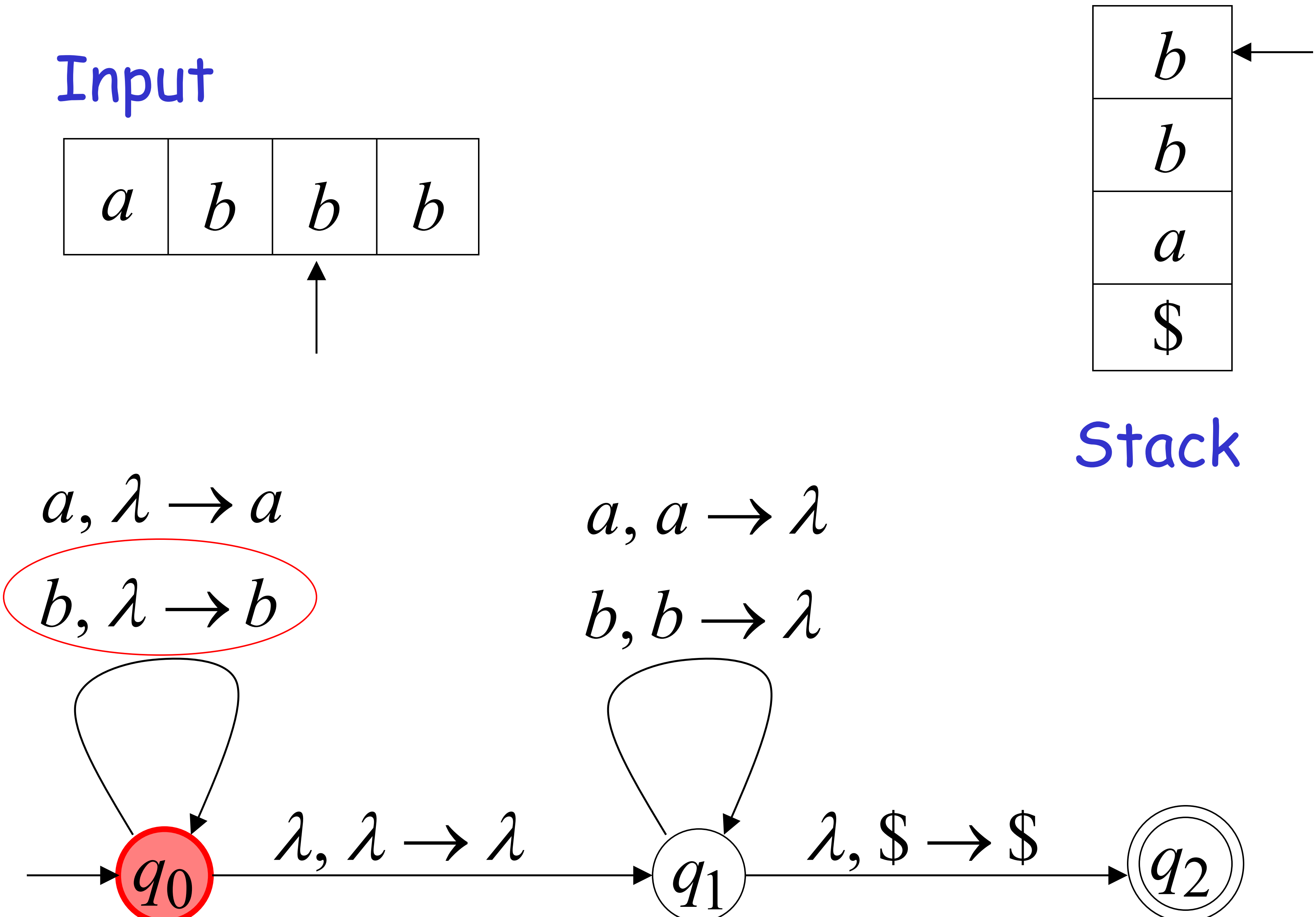


Time 2

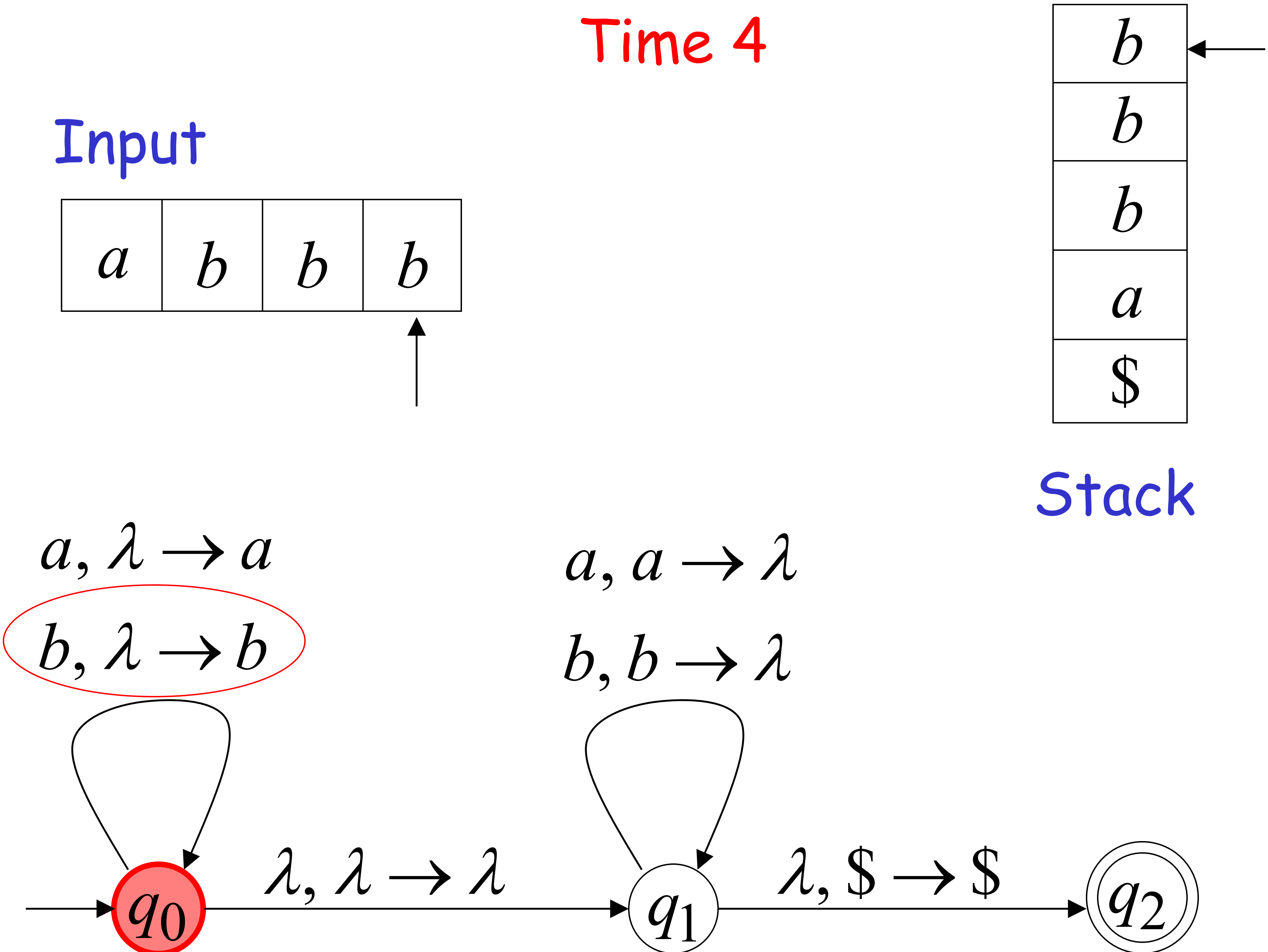


Input
a b b b
b
a
$\$$
Stack
$a, \lambda \rightarrow a$
$b, \lambda \rightarrow b$
$a, a \rightarrow \lambda$
$b, b \rightarrow \lambda$
$\lambda, \lambda \rightarrow \lambda$
$\lambda, \$ \rightarrow \$$
q_0
q_1
q_2

Time 3

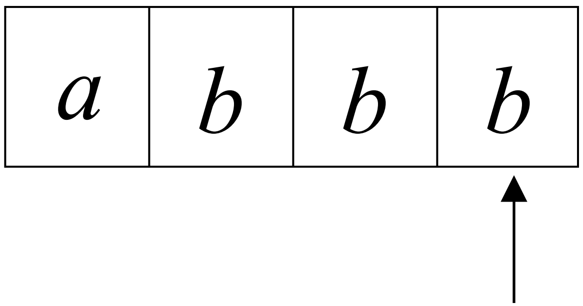

Input
a b b b
b
b
a
$\$$
Stack
$a, \lambda \rightarrow a$
$b, \lambda \rightarrow b$
$a, a \rightarrow \lambda$
$b, b \rightarrow \lambda$
$\lambda, \lambda \rightarrow \lambda$
$\lambda, \$ \rightarrow \$$
q_0
q_1
q_2

Time 4

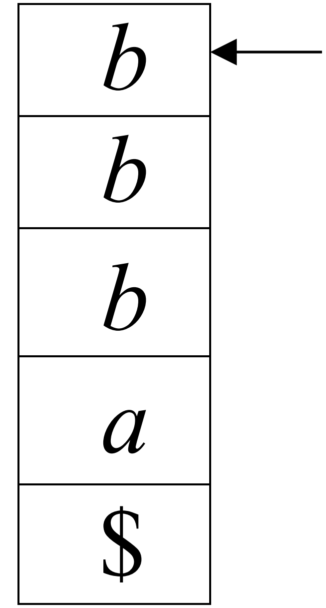


Time 5

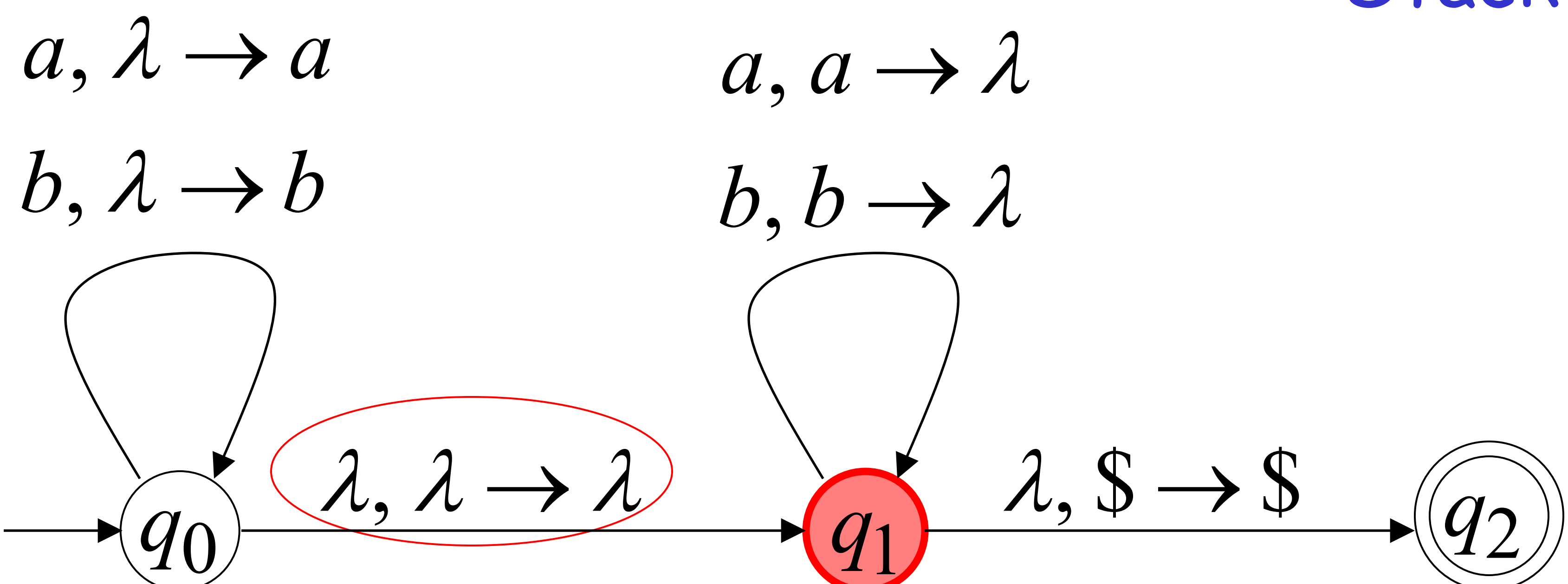
Input



No stato di accettazione

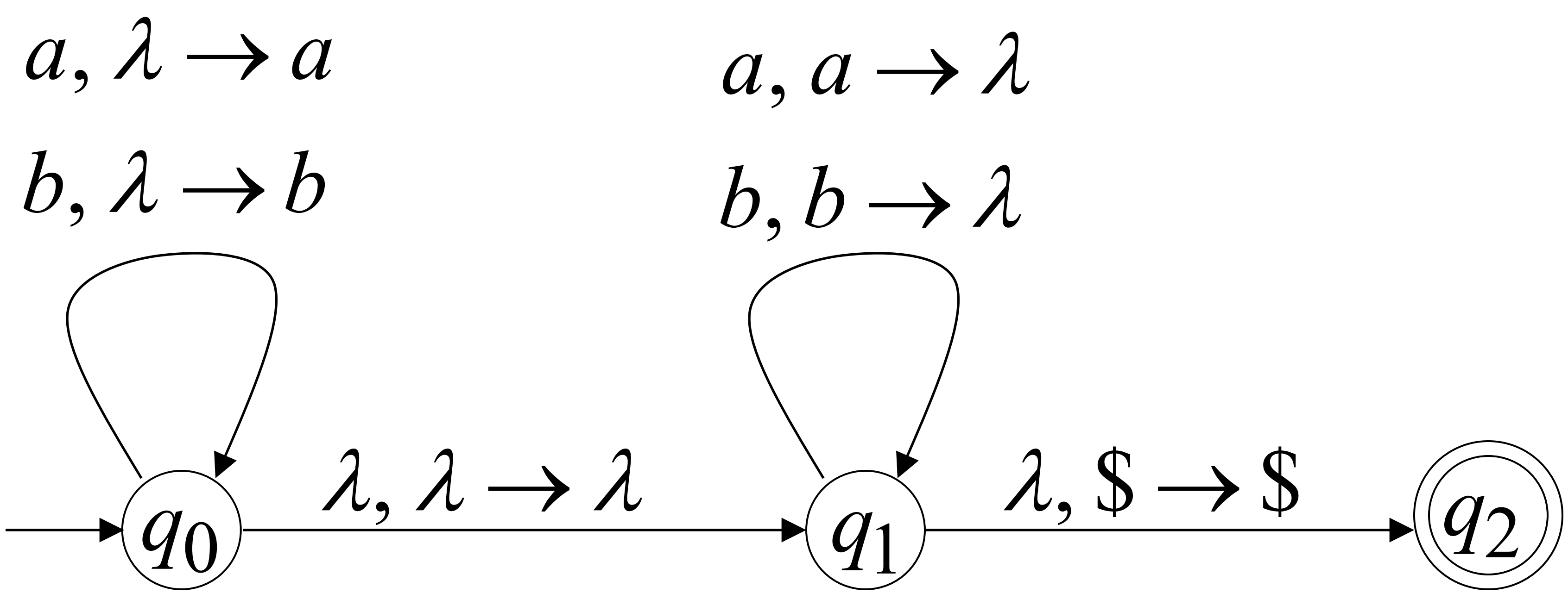


Stack



Non esiste nessuna computazione che accetta $abbb$

$$abbb \notin L(M)$$



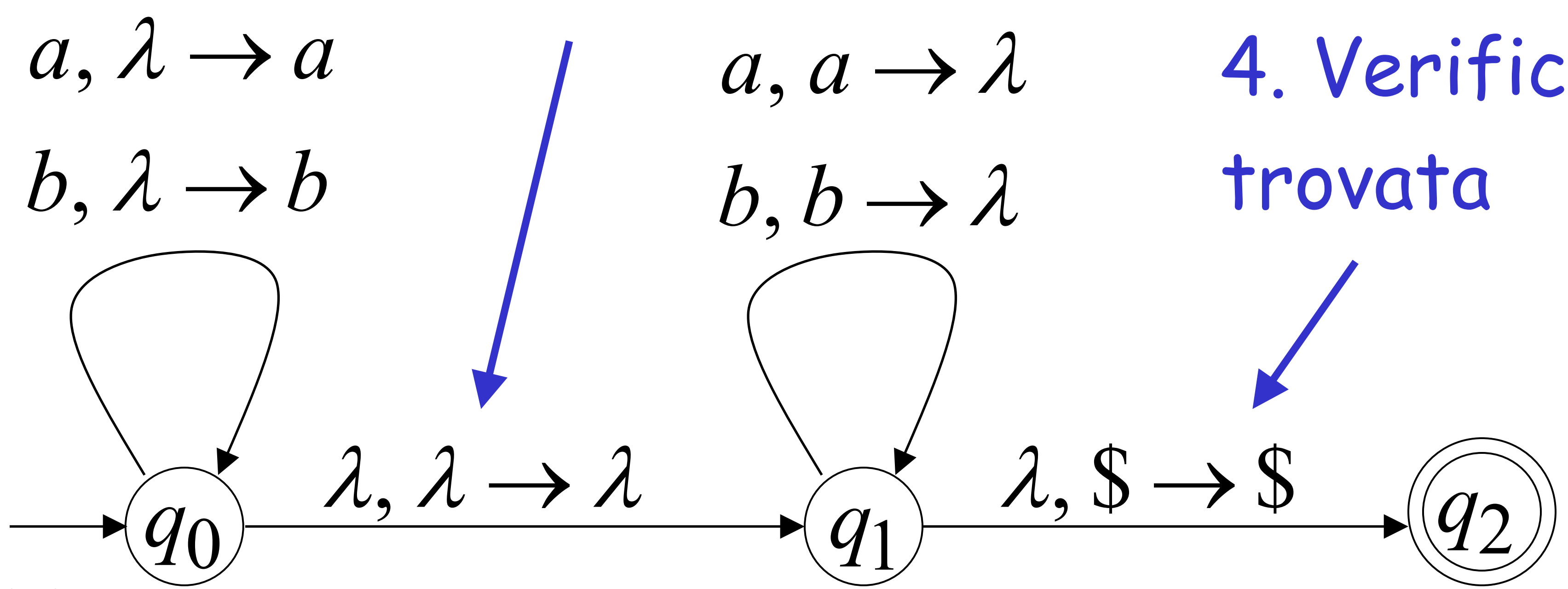
Basic Idea: $L(M) = \{vv^R : v \in \{a,b\}^*\}$

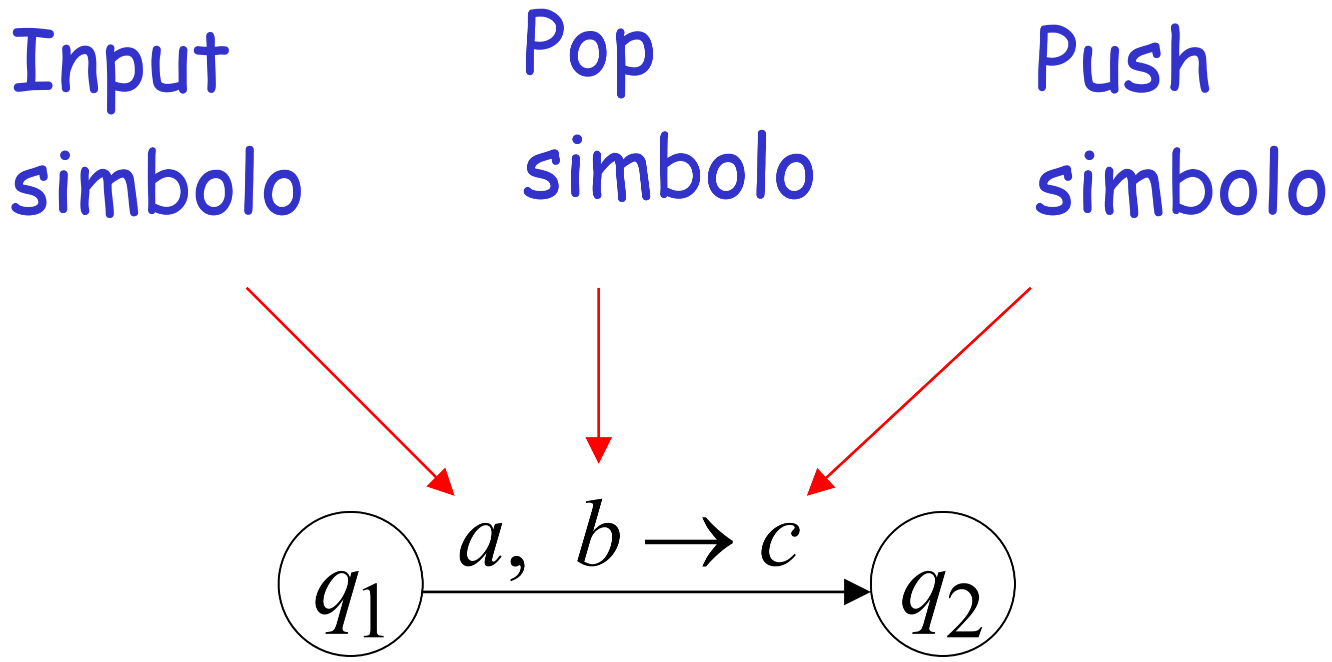
1. Push v Nello stack

2. Congettura metà Dell'input

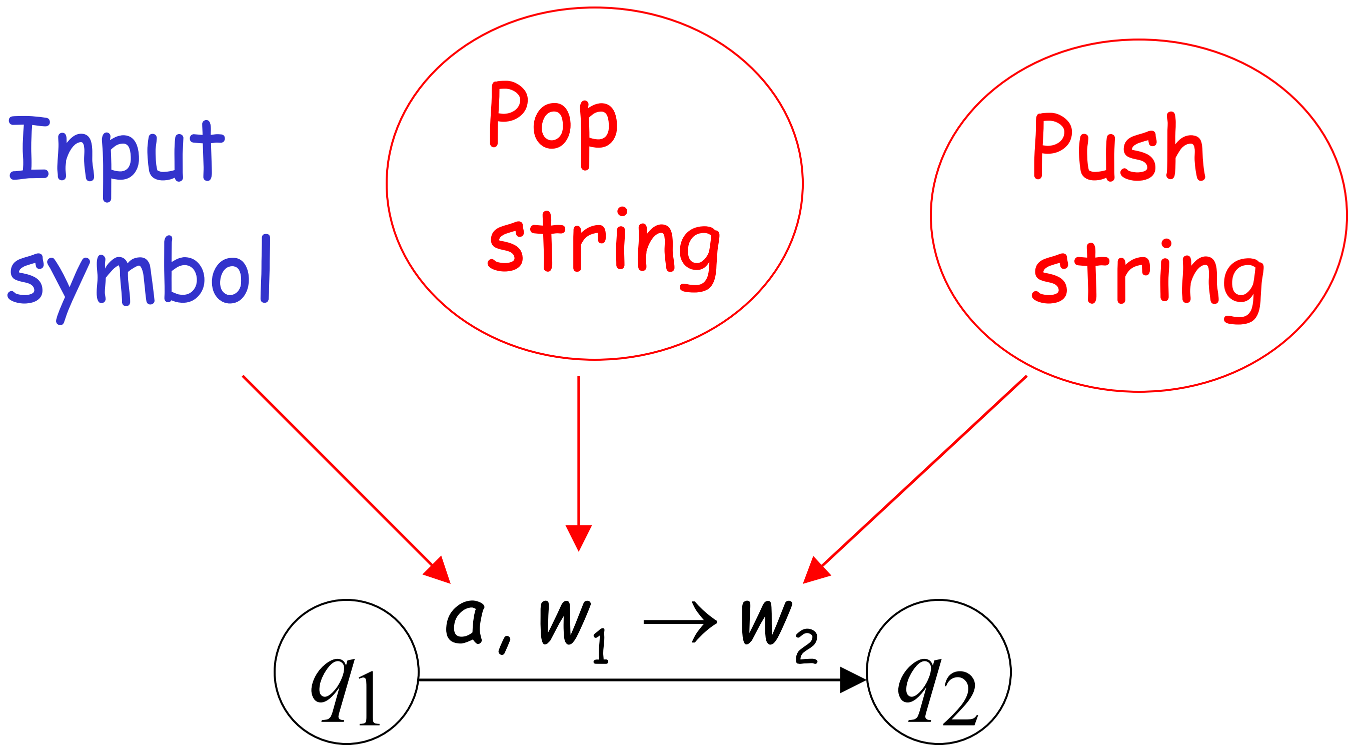
3. verifica v^R in Input con v nello stack

4. Verifica trovata

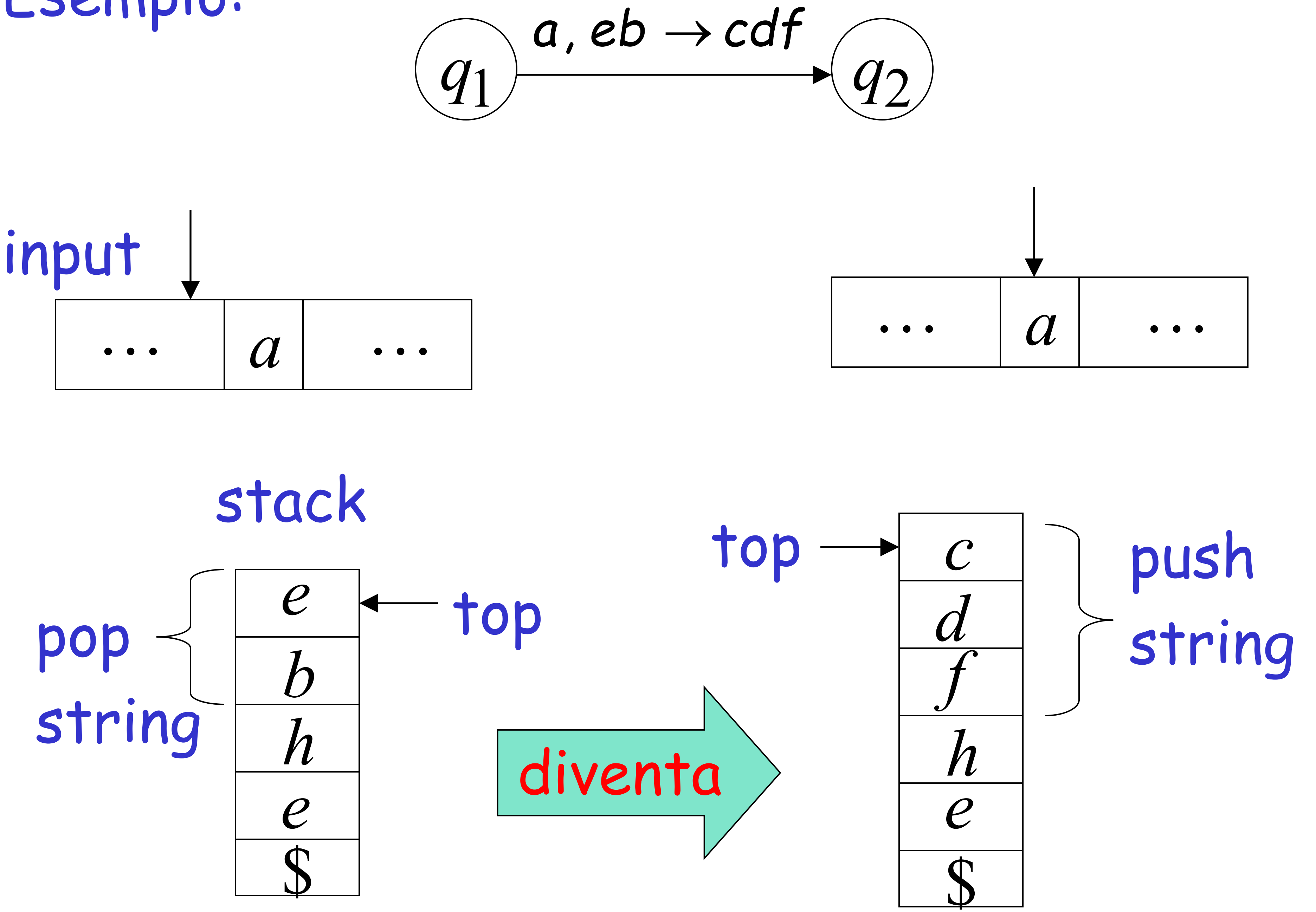



Input simbolo
Pop simbolo
Push simbolo
$a, \ b \rightarrow c$
q_1
q_2

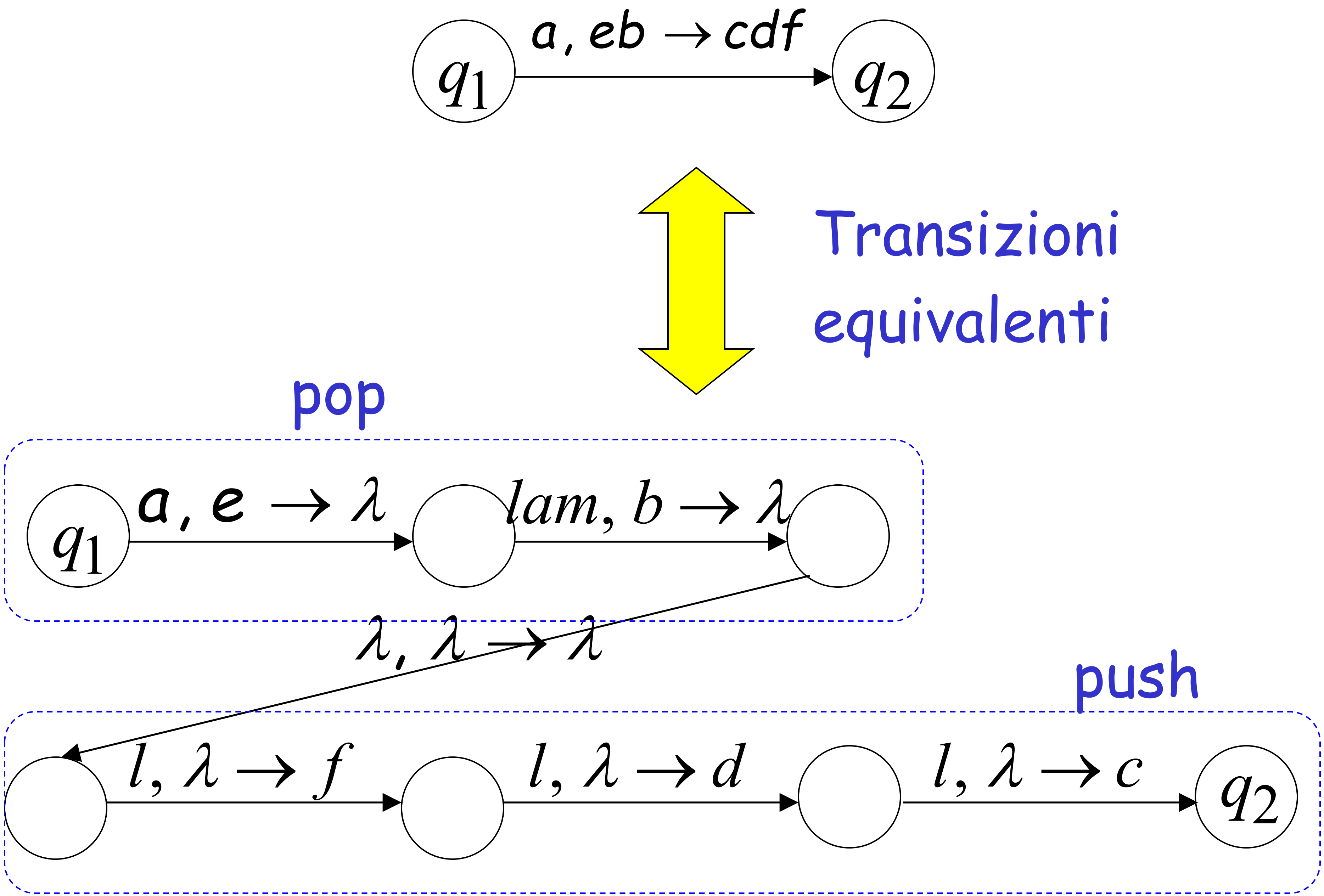
Pushing & Popping Strings



Esempio:



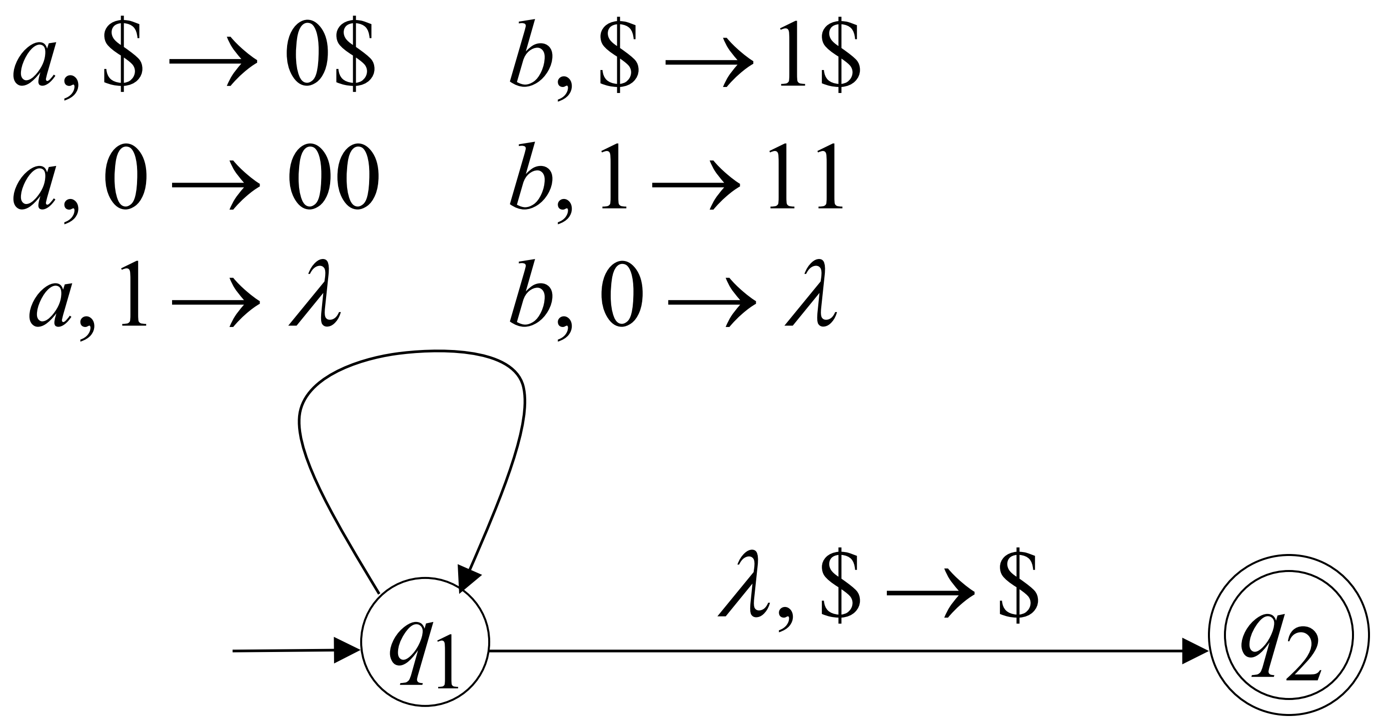
q_1
$a, eb \rightarrow cdf$
q_2
input
$\cdots$
a
$\cdots$
stack
top
pop
string
e
b
h
e
$\$$
diventa
push
string
c
d
f
h
e
$\$$


$a, eb \to cdf$
q_1
q_2
Transizioni equivalenti
pop
$a, e \to \lambda$
$lam, b \to \lambda$
$\lambda, \lambda \to \lambda$
push
$l, \lambda \to f$
$l, \lambda \to d$
$l, \lambda \to c$

altro PDA esempio

$$L(M) = \{w \in \{a,b\}^* : \quad n_a(w) = n_b(w)\}$$

PDA M



esecuzione:

Time 0

Input

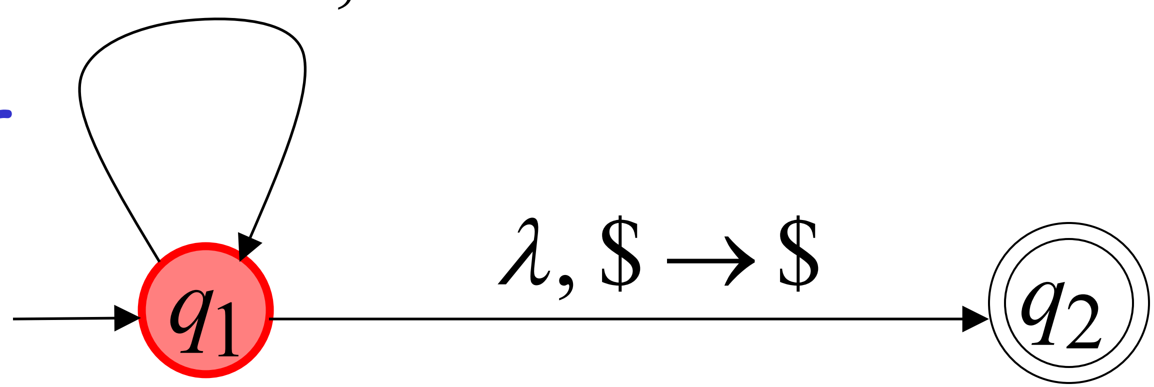
a	b	b	b	a	a

$\$$

Stack

$$a, \$ \to 0\$ \qquad b, \$ \to 1\$$$
$$a, 0 \to 00 \qquad b, 1 \to 11$$
$$a, 1 \to \lambda \qquad b, 0 \to \lambda$$

current state



Time 1

Input

a	b	b	b	a	a

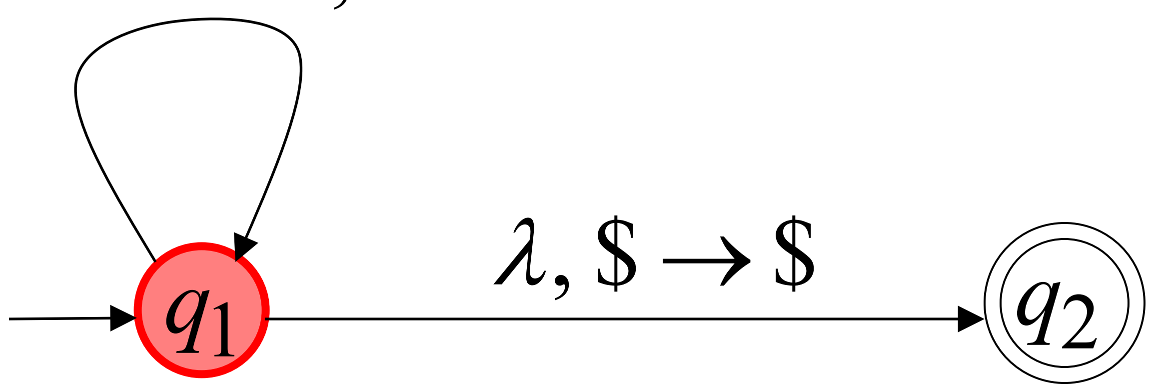
Stack

0
$\$$

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

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

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



Time 3

Input

a	b	b	b	a	a

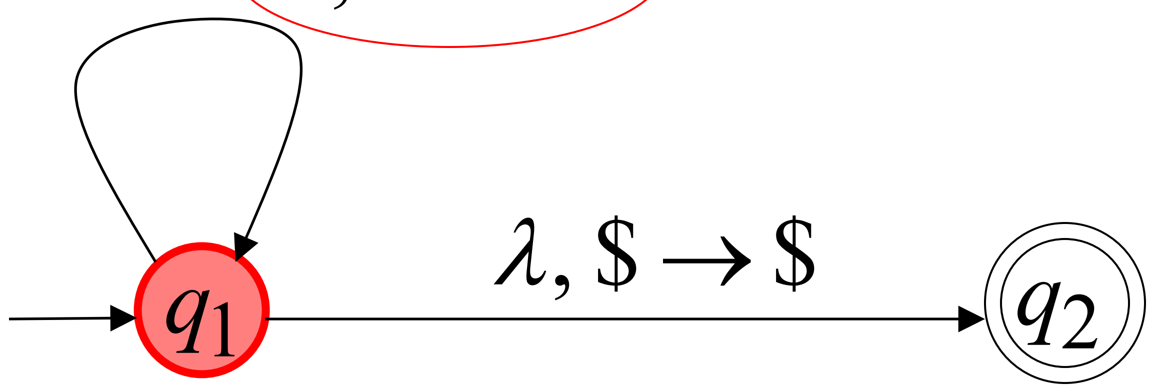
Stack

$\cancel{0}$
$\$$

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

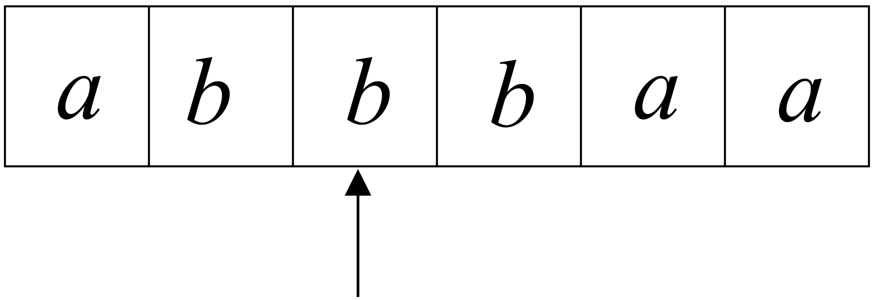
$$a, 0 \rightarrow 00 \qquad b, 1 \rightarrow 11$$

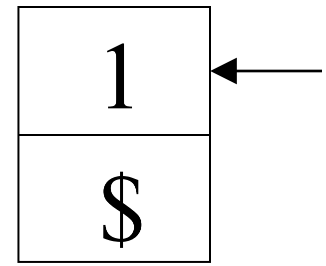
$$a, 1 \rightarrow \lambda \qquad b, 0 \rightarrow \lambda$$



Time 4

Input



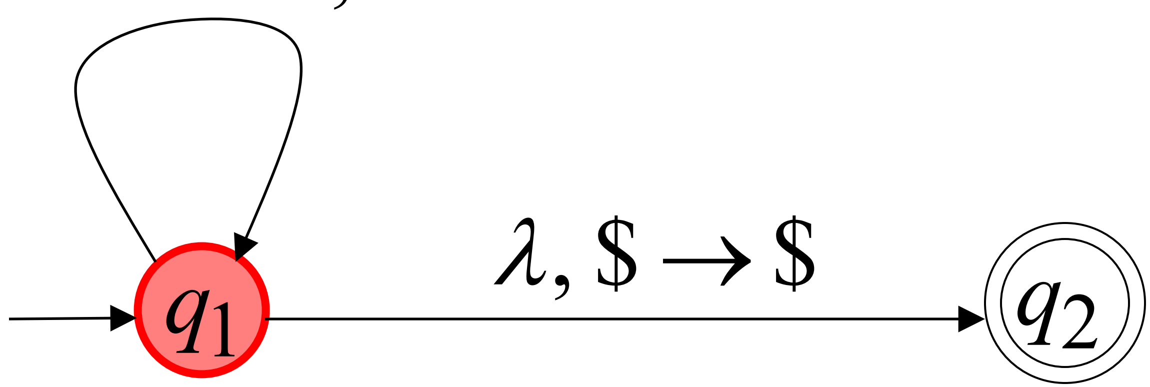


Stack

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

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

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



Time 5

Input

a	b	b	b	a	a

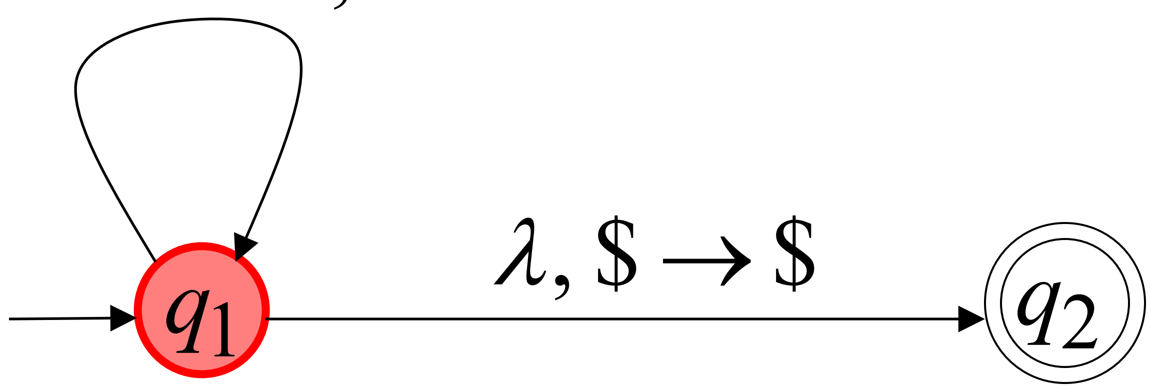
1
1
\$

Stack

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

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

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



Time 6

Input

a	b	b	b	a	a

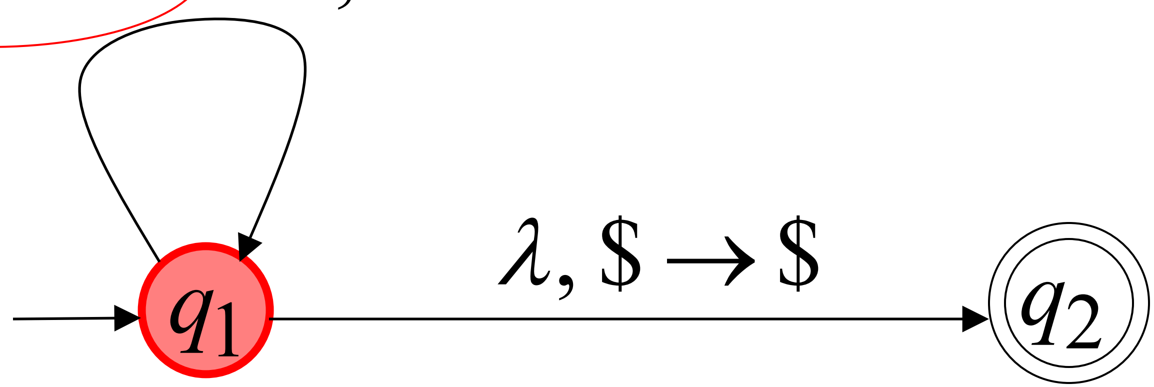
$\cancel{1}$
1
$\$$

Stack

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

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

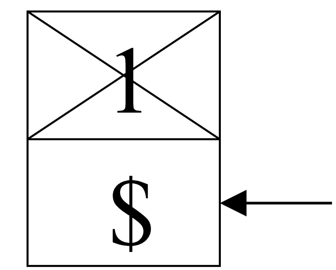
$a, 1 \to \lambda$ $\quad$ $b, 0 \to \lambda$



Time 7

Input

a	b	b	b	a	a

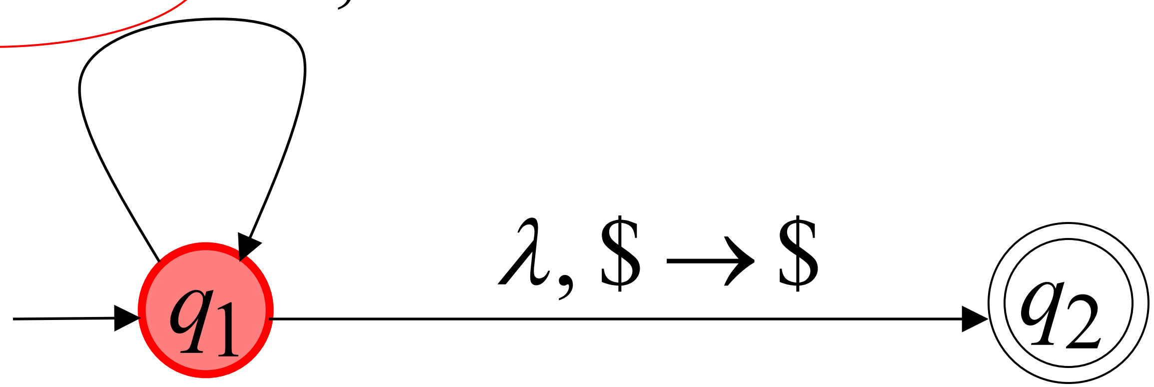


Stack

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

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

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



Time 8

Input

a	b	b	b	a	a

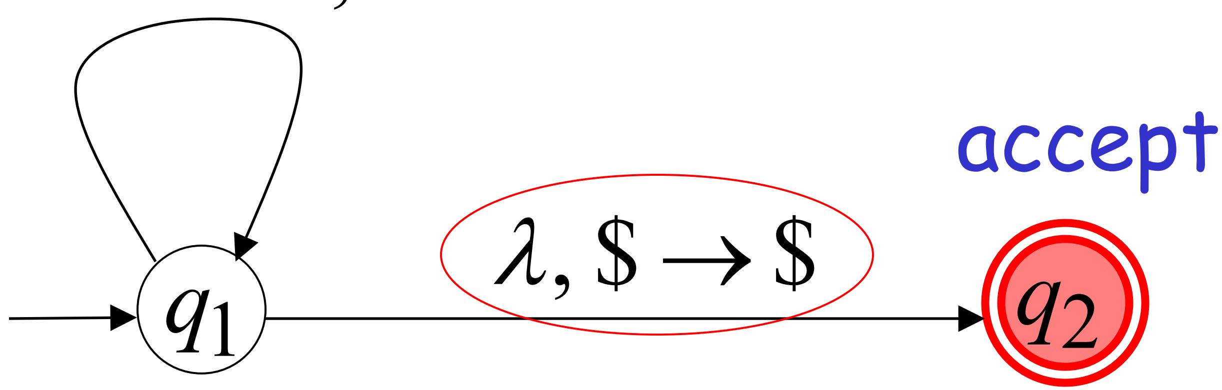
$\$$

Stack

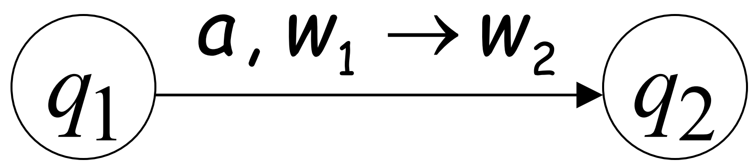
$a, \$ \to 0\$ \qquad b, \$ \to 1\$$

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

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

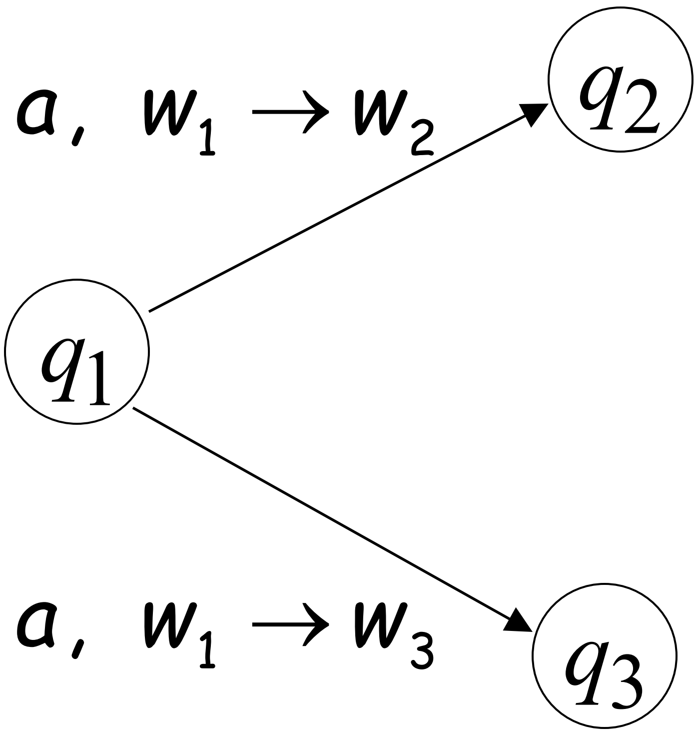


Formalismo per I PDA



Funzione di transizione:

$$\delta(q_1, a, w_1) = \{(q_2, w_2)\}$$



Funzione di transizione :

$$\delta(q_1, a, w_1) = \{(q_2, w_2), \; (q_3, w_3)\}$$

Formal Definition

Pushdown Automaton (PDA)

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

Accept states

States

Input alphabet

Stack alphabet

Transition function

Initial state

Stack start symbol

Delta

: Stato x Inputx Stack -> P {(Stato, Stack)}

Instantaneous Description

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

Current state

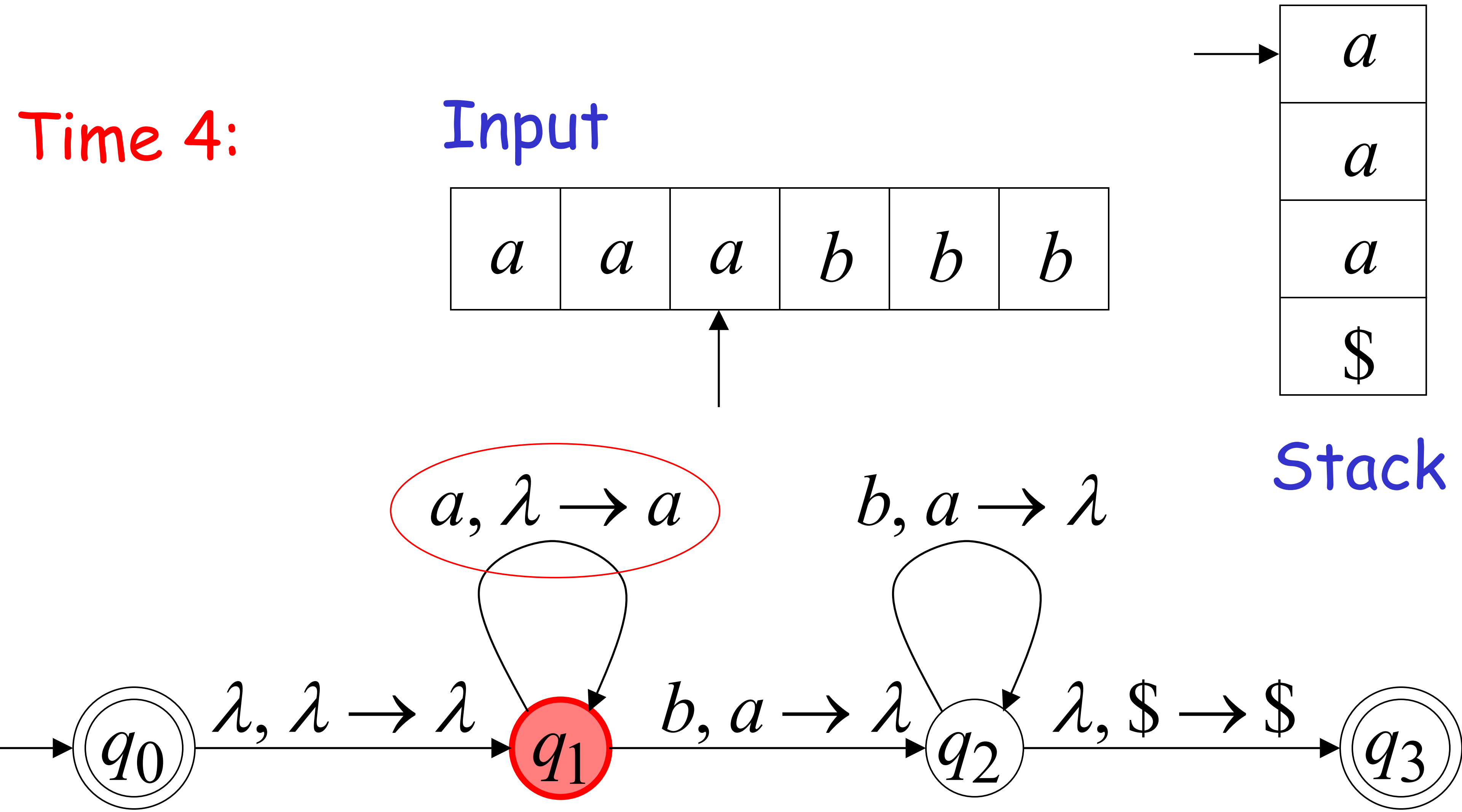
Remaining input

Current stack contents

Example: Instantaneous Description

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

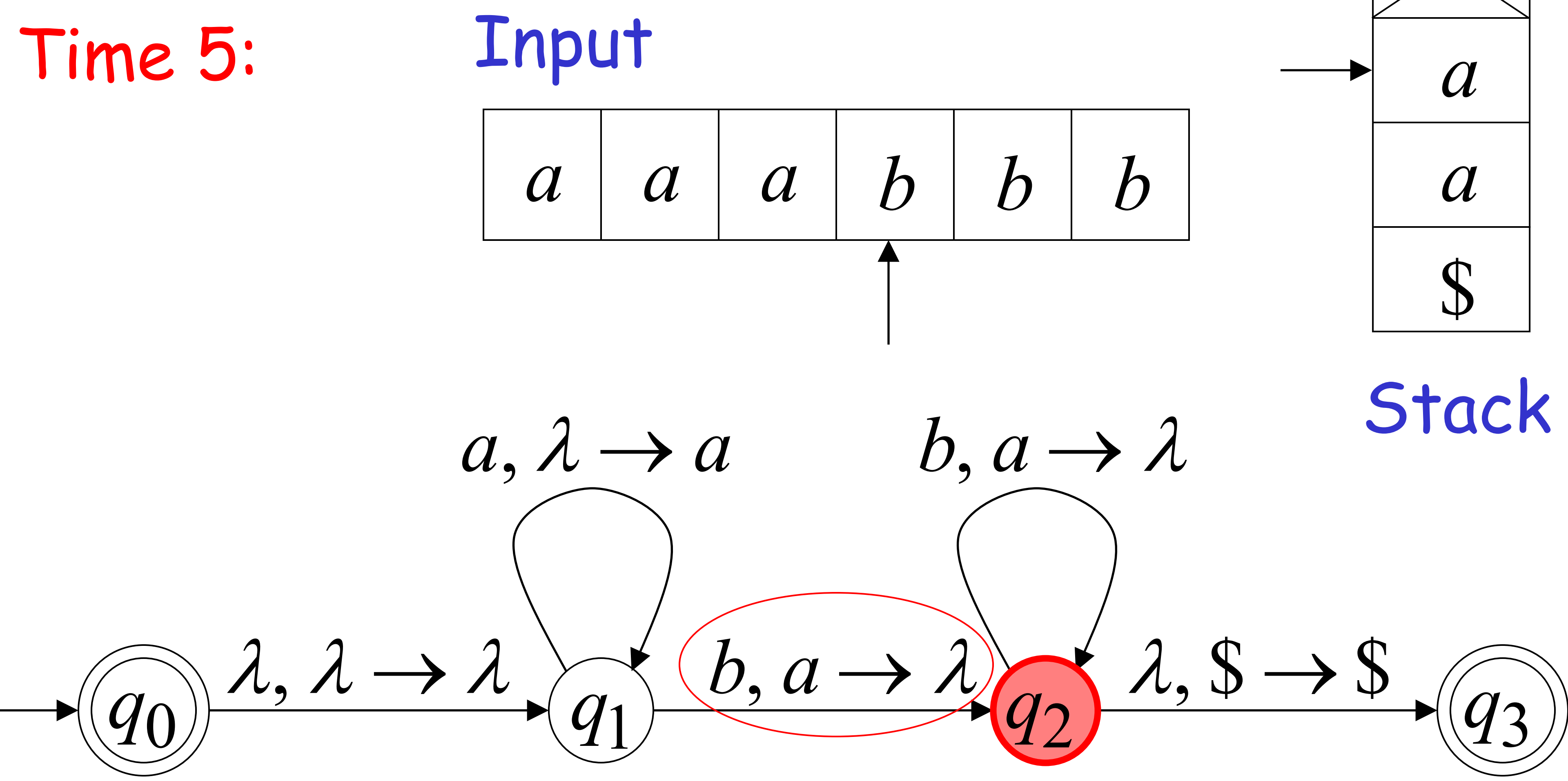
Time 4:



Example: Instantaneous Description

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

Time 5:



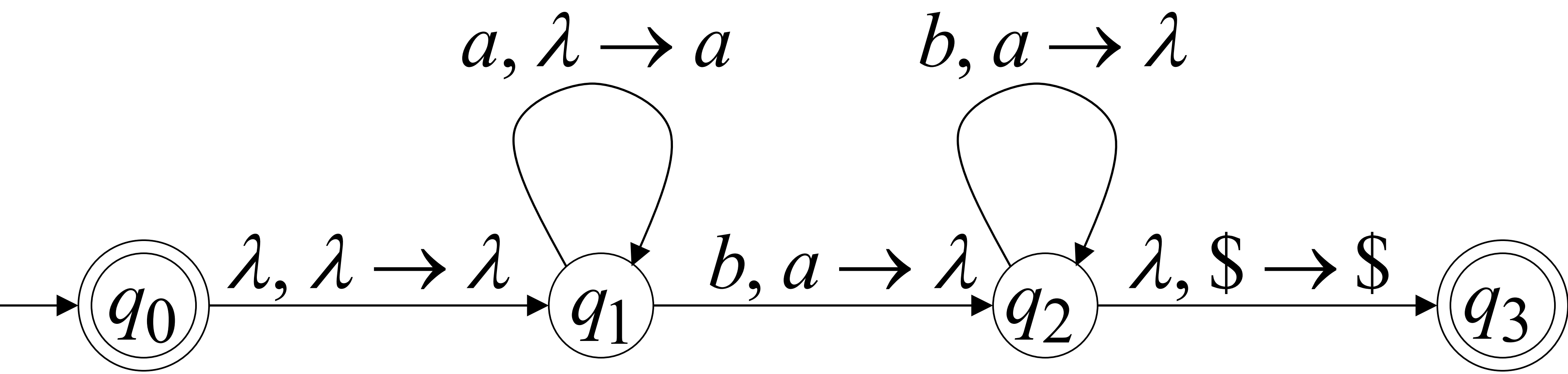
scriviamo:

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

Time 4 Time 5

Una computazione:

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



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

Per convenienza scriviamo:

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

Language of PDA

Linguaggio $L(M)$ accettato da PDA M :

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

Initial state

Accept state

Stack può essere anche non vuoto, quindi s qualsiasi.

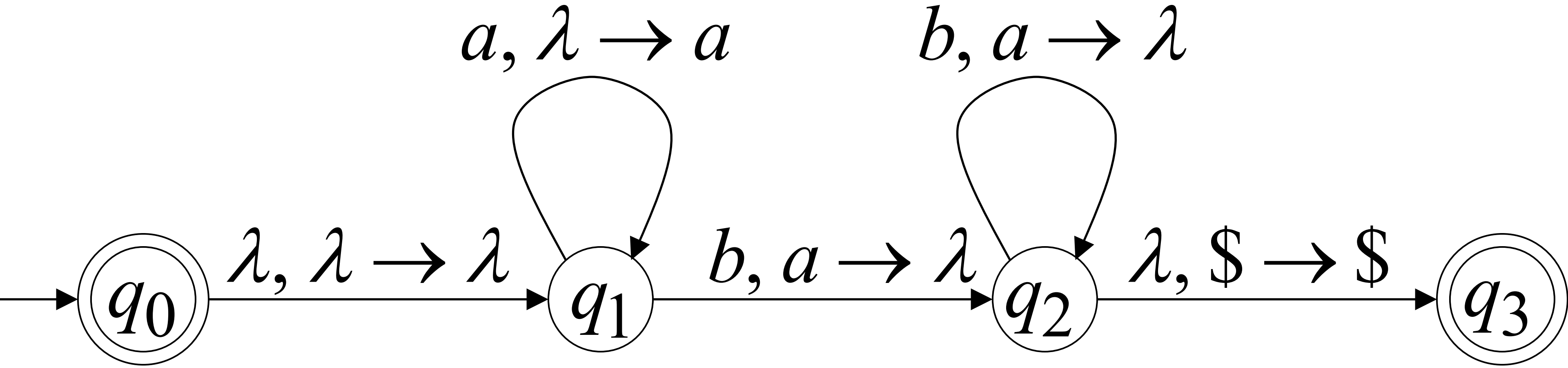
Esempio:

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

$$\Downarrow$$

$$aaabbb \in L(M)$$

PDA M:

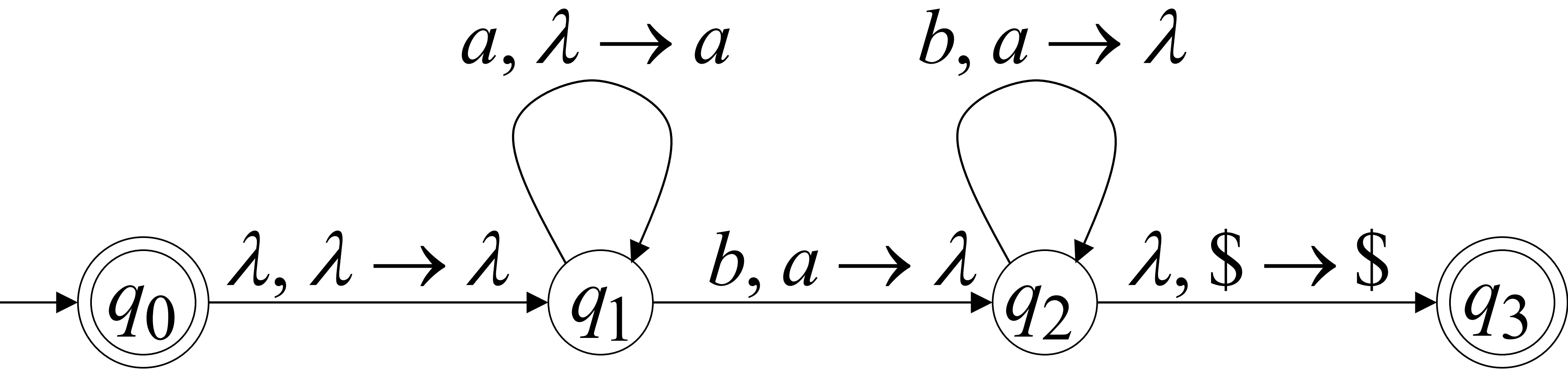


$$(q_0, a^n b^n, \$) \overset{*}{\succ} (q_3, \lambda, \$)$$

$$\Downarrow$$

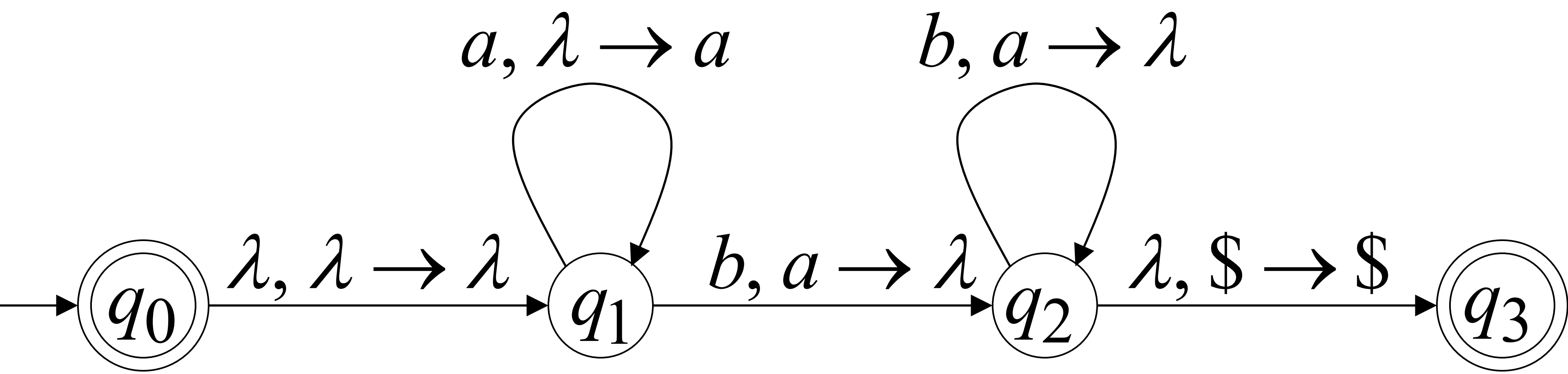
$$a^n b^n \in L(M)$$

PDA M :



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

PDA M :



Normal Forms
per
grammatiche Context-free

Context free

Ogni produzioni ha la forma:

$$A \to stringa$$

variabile

Terminali o costanti

Chomsky Normal Form

Ogni produzioni ha la forma:

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

variabile variabile terminal

esempi:

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

Chomsky
Normal Form

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

Not Chomsky
Normal Form

Conversione nella Chomsky Normal Form

esempio:

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

Not Chomsky Normal Form

Convertiamo questa grammatica nella Chomsky Normal Form

Introduciamo nuove variabili per i terminali:

$$T_a, T_b, T_c$$

$$S \to ABa$$

$$A \to aab$$

$$B \to Ac$$

$$S \to ABT_a$$

$$A \to T_a T_a T_b$$

$$B \to AT_c$$

$$T_a \to a$$

$$T_b \to b$$

$$T_c \to c$$

Introduciamo una nuova variabile intermedia

Per rompere la prima produzione: V_1

$$\boxed{S \to ABT_a}$$
$$A \to T_a T_a T_b$$
$$B \to AT_c$$
$$T_a \to a$$
$$T_b \to b$$
$$T_c \to c$$

$$\boxed{\begin{array}{l} S \to AV_1 \\ V_1 \to BT_a \end{array}}$$
$$A \to T_a T_a T_b$$
$$B \to AT_c$$
$$T_a \to a$$
$$T_b \to b$$
$$T_c \to c$$

Introduciamo la variabile intermedia : V_2

$$
\begin{aligned}
S &\to A V_1 \\
V_1 &\to B T_a \\
&\boxed{A \to T_a T_a T_b} \\
B &\to A T_c \\
T_a &\to a \\
T_b &\to b \\
T_c &\to c
\end{aligned}
\quad\Longrightarrow\quad
\begin{aligned}
S &\to A V_1 \\
V_1 &\to B T_a \\
&\boxed{\begin{aligned} A &\to T_a V_2 \\ V_2 &\to T_a T_b \end{aligned}} \\
B &\to A T_c \\
T_a &\to a \\
T_b &\to b \\
T_c &\to c
\end{aligned}
$$

grammatica in Chomsky Normal Form:

$$S \to AV_1$$

$$V_1 \to BT_a$$

$$A \to T_a V_2$$

$$V_2 \to T_a T_b$$

$$B \to AT_c$$

$$T_a \to a$$

$$T_b \to b$$

$$T_c \to c$$

Iniziale grammatica

$$S \to ABa$$

$$A \to aab$$

$$B \to Ac$$

In generale:

Per ogni grammatica context-free
(che non produce λ)
non in Chomsky Normal Form

Possiamo ottenere:

una grammatica equivalente
in Chomsky Normal Form

La procedura

First remove:

variabili che si possono annulare

(variabili inutili, optional)

Poi, per ogni simbolo $\quad : a$

Nuova variabile: T_a

Nuova produzione $T_a \rightarrow a$

Nelle produzioni con lunghezza maggiore o uguale a due

poni al posto di a il non terminale T_a

produzioni della forma $A \rightarrow a$

Non necessitano di cambio!

Rimpiazza
ogni produzione $A \to C_1 C_2 \cdots C_n$

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

Nuove variabili intermedie : $V_1, V_2, \ldots, V_{n-2}$

Observations

- Chomsky normal forms are good for parsing and proving theorems

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

The Pumping Lemma for CFL's

Statement

Intuition

- Recall the pumping lemma for regular languages.
- It told us that if there was a string long enough to cause a cycle in the DFA for the language, then we could "pump" the cycle and discover an infinite sequence of strings that had to be in the language.

Intuition – (2)

- For CFL's the situation is a little more complicated.
- We can always find two pieces of any sufficiently long string to "pump" in tandem.
  - That is: if we repeat each of the two pieces the same number of times, we get another string in the language.

Statement of the CFL Pumping Lemma

For every context-free language L

There is an integer n, such that

For every string z in L of length $\geq$ n

There exists z = uvwxy such that:

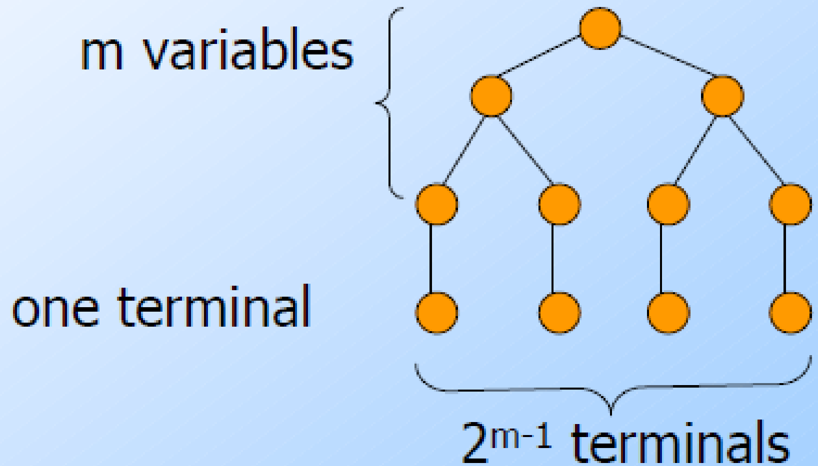
1. |vwx| $\leq$ n.
2. |vx| > 0.
3. For all i $\geq$ 0, uv^iwx^iy is in L.

Proof of the Pumping Lemma

- Start with a CNF grammar for $L - \{\epsilon\}$.
- Let the grammar have m variables.
- Pick $n = 2^m$.
- Let $|z| \geq n$.
- We claim ("*Lemma 1*") that a parse tree with yield z must have a path of length $m+2$ or more.

Proof of Lemma 1

- If all paths in the parse tree of a CNF grammar are of length $\leq$ m+1, then the longest yield has length 2^{m-1}, as in:

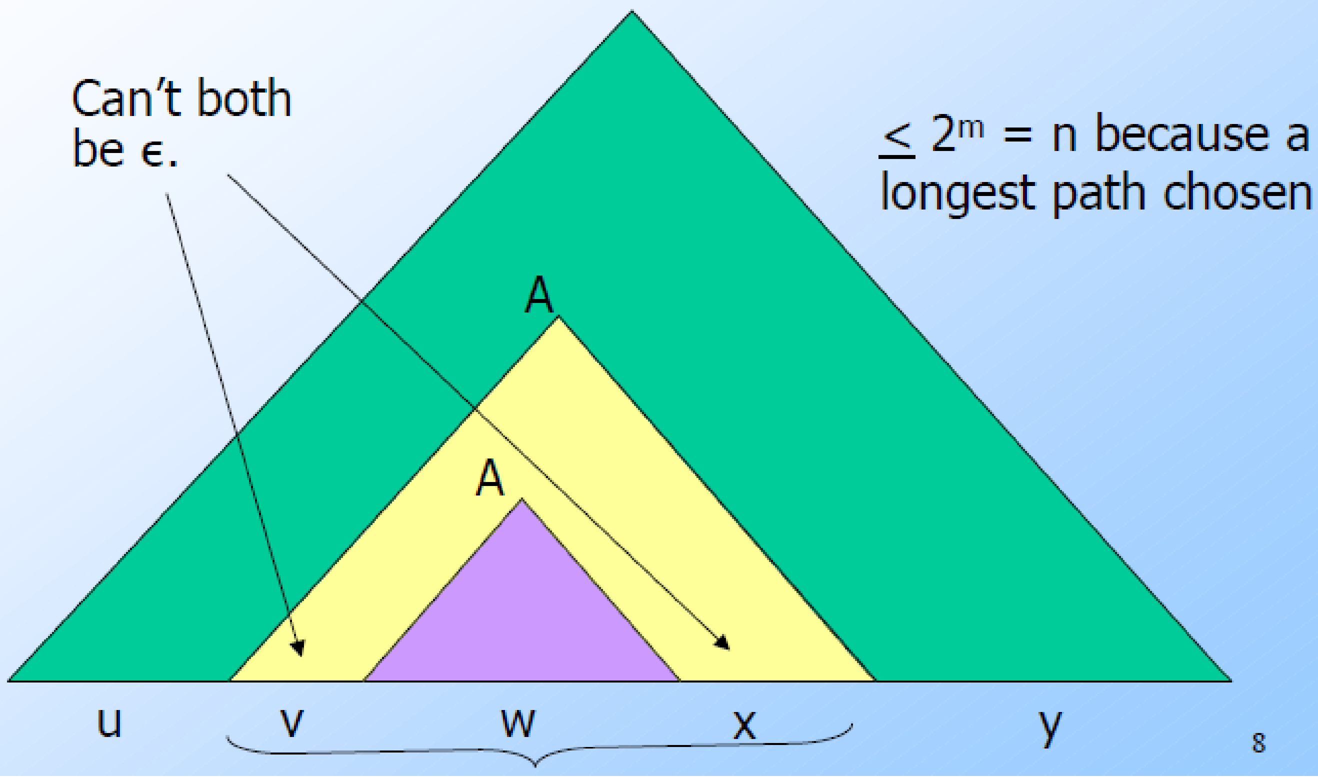


6

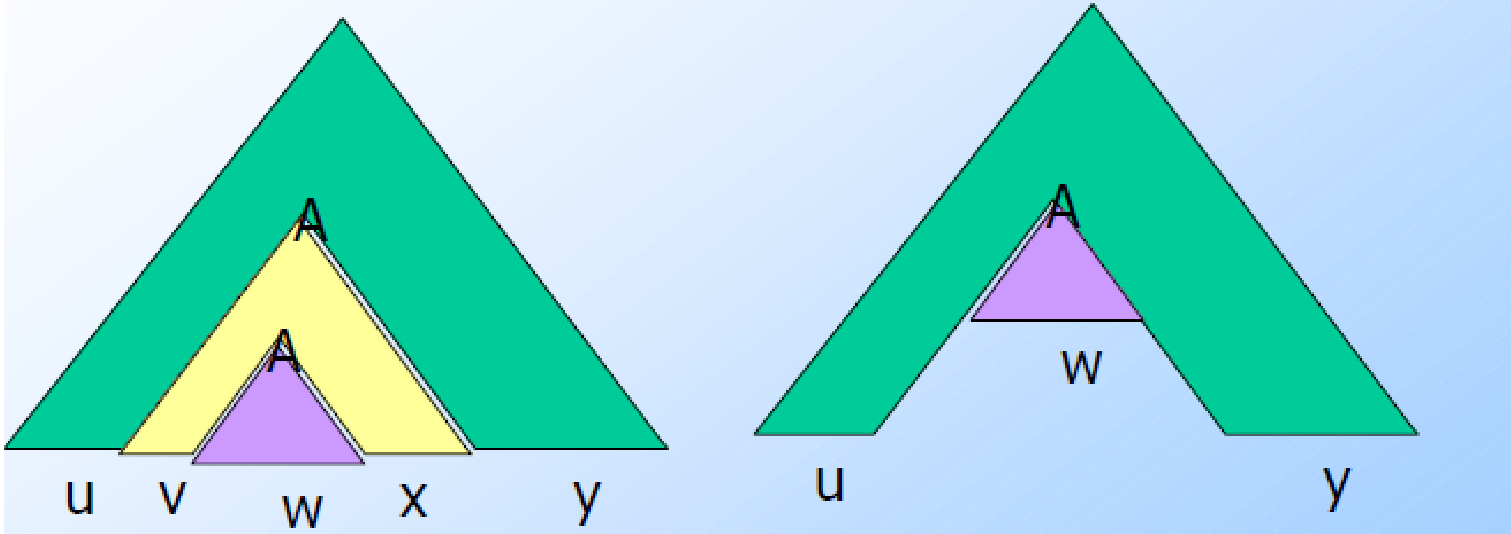
Back to the Proof of the Pumping Lemma

- Now we know that the parse tree for z has a path with at least m+1 variables.
- Consider some longest path.
- There are only m different variables, so among the lowest m+1 we can find two nodes with the same label, say A.
- The parse tree thus looks like:

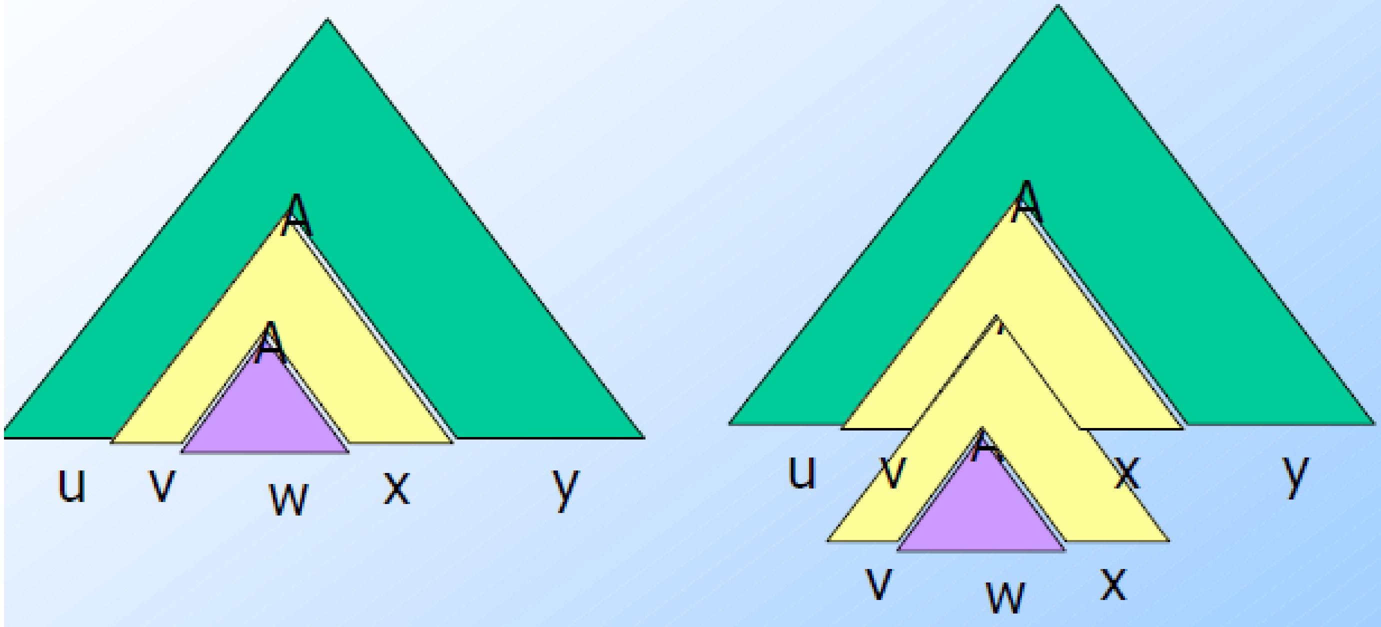
Parse Tree in the Pumping-Lemma Proof



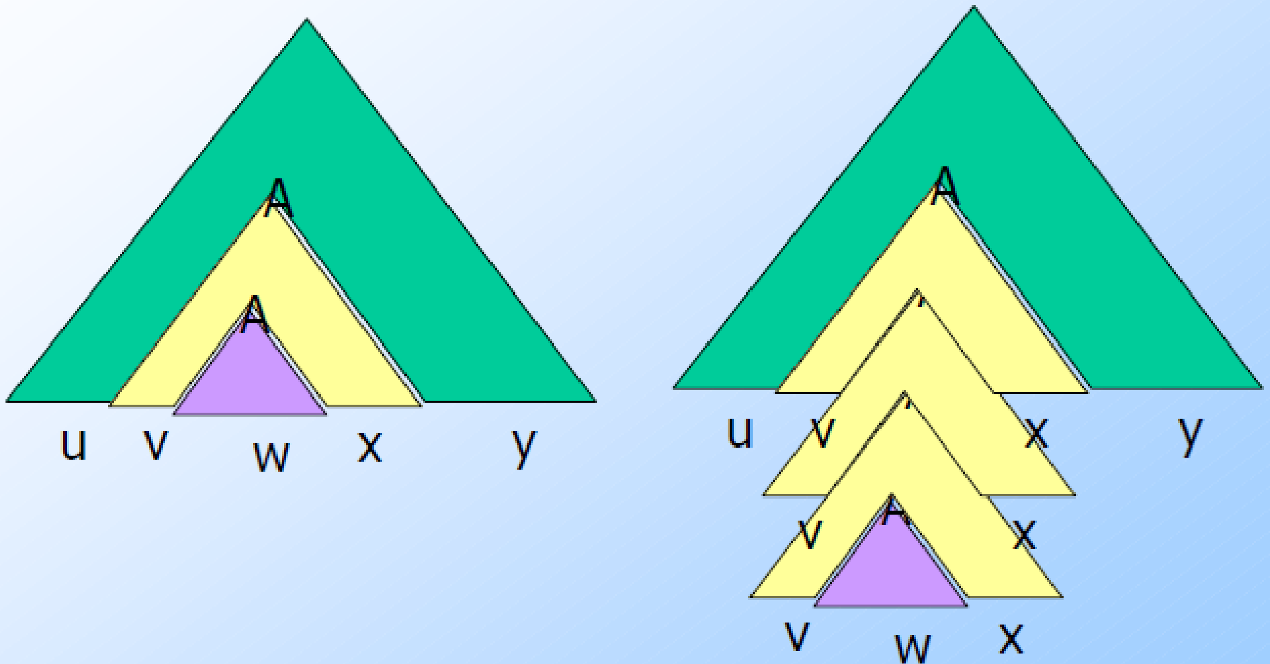
Pump Zero Times



Pump Twice



Pump Thrice Etc., Etc.



applicazioni

del
Pumping Lemma

Gli stati dell'automa , Variabili della grammatica e vale l'opposto.

Le variabili context free sono gli stati del PDA?

W y Z x K, W yy Z xx K, W yyy Z xxx K

Qualcosa di più complicato dobbiamo considerare sia la variabile che genera la y (Y) e sia la variabile che genera la x (X)

Stato YX ?

Quando ho un linguaggio se «provo» che le parole possono essere scritte

X y_i Y

A* a, aa, aaa,

A*B* due pezzi le A e le B indipendenti

W y_i Z x_i K

A_n B_n no regolare

AA BB non indipendenti

Il Pumping Lemma:

- dato un linguaggio regolare infinito L
- esiste un intero m (lunghezza critica)
- Per ogni stringa $w \in L$ con lunghezza $|w| \geq m$
- possiamo scrivere $w = x\, y\, z$
- con $|x\, y| \leq m$ e $|y| \geq 1$
- tale che: $x\, y^i\, z \in L \qquad i = 0, 1, 2, \ldots$

Teorema: Il linguaggio

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

non è regolare

Proof: Usiamo il Pumping Lemma

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

assumiamo per contradizione
che L sia un linguaggio regolare

poichè L è infinito
Possiamo applicare il Pumping Lemma

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

sia m la lunghezza critica per L

Prendiamo una stringa w tale che: $w \in L$

con lunghezza

$$|w| \geq m$$

prendiamo $w = a^m b^m b^m a^m$

Dal Pumping Lemma:

Possiamo scrivere: $w = a^m b^m b^m a^m = x\ y\ z$

con lunghezza: $|x\,y| \le m, \quad |y| \ge 1$

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

(where z spans from the remaining $a\ldots a$ to the end)

allora: $y = a^k, \quad 1 \le k \le m$

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

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

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

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

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

Dal Pumping Lemma: $\quad x\ y^2\ z \ \in\ L$

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

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

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

---

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

$$\Downarrow$$

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

CONTRADIZIONE!!!

Considerare i casi
con y tra le b, tra le
a e tra ab.

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

quindi: L'assunzione che L è un linguaggio regolare non è vera

Conclusione: L Non è un linguaggio regolare

END OF PROOF

Teorema: il linguaggio

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

non è regolare

Proof: Usiamo il Pumping Lemma

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

assumiamo per contradzione
che L sia un linguaggio regolare

poichè L è infinito allora
possiamo applicare il Pumping Lemma

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

sia m la lunghezza critica di L

Prendiamo una stringa w tale che: $w \in L$

e lunghezza $|w| \geq m$

prendiamo $w = a^m b^m c^{2m}$

Dal **Pumping Lemma:**

possiamo scrivere $w = a^m b^m c^{2m} = x\ y\ z$

con lunghezzas $|x\ y| \le m, \quad |y| \ge 1$

$$w = xyz = \underbrace{a\ldots a}_{x}\underbrace{a\ldots a}_{y}\underbrace{a\ldots ab\ldots bc\ldots cc\ldots c}_{z}$$

$$w = \overbrace{a\ldots aa\ldots aa\ldots a}^{m}\overbrace{b\ldots b}^{m}\overbrace{c\ldots cc\ldots c}^{2m}$$

allora: $y = a^k, \quad 1 \le k \le m$

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

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

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

allora: $x\ y^0\ z\ = xz\ \ \square\ \ L$

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

Dal Pumping Lemma: $\quad xz \in L$

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

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

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

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

---

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

$$\Downarrow$$

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

Contradizione.

Vedere gli altri casi. La y tra le b, tra le c, tra ab, tra bc.

La nostra assunzione che L sia un linguaggio regolare non è vera

Conclusione: L non è un linguaggio regolare

END OF PROOF

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

Non è regolare

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

dimostrazione: Usiamo il Pumping Lemma

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

assumiamo che L
sia un linguaggio regolare

poichè L è infinito
Possiamo applicare il Pumping Lemma

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

sia m la lunghezza critica of L

prendiamo una stringa w tale che: $w \in L$

lunghezza $|w| \geq m$

$$\boxed{\text{prendiamo } w = a^{m!}}$$

Dal Pumping Lemma:

Possiamo scrivere $w = a^{m!} = x\ y\ z$

con lunghezza $|x\ y| \leq m, \quad |y| \geq 1$

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

(where z is everything after y: $z = a\ldots aa\ldots aa\ldots a$)

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

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

Dal Pumping Lemma:

$$x\ y^i\ z \ \in\ L$$

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

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

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

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

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

(with z = $a\ldots aa\ldots aa\ldots a$, the part after the second y)

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

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

poichè: $L = \{a^{n!} : n \geq 0\}$

$\Downarrow$

Deve esistere p tale che:

$$m! + k = p!$$

ovvero: $m!+k \le m!+m$ per $m>1$

$$\le m!+m!$$

$$< m!m + m!$$

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

$$= (m+1)!$$

$$\Downarrow$$

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

$$\Downarrow$$

$m!+k \neq p!$ Per ogni p

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

---

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

$$\Downarrow$$

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

contradizione

quindi: La nostra assunzione che L
È un linguaggio regolare
Non è vera

Conclusione: L Non è un linguaggio regolare

END OF PROOF

Applicazioni del Pumping Lemma context free

The Pumping Lemma:

Per un linguaggio infinito context-free L

Esiste un intero m tale che

per ogni stringa $w \in L, \quad |w| \geq m$

possiamo scrivere $w = uvxyz$

Con lunghezze $|vxy| \leq m$ and $|vy| \geq 1$

E deve essere:

$$uv^i xy^i z \in L, \quad \text{for all } i \geq 0$$

m>=
(2_numero delle variabili della grammatica)-1

Perché?

linguaggi Non-context free

$$\{a^n b^n c^n : n \geq 0\} \qquad \{vv : v \in \{a,b\}\}$$

linguaggi Context-free

$$\{a^n b^n : n \geq 0\} \qquad \{ww^R : w \in \{a,b\}^*\}$$

Perché?

Teorema: il linguaggio

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

non è context free

Dim.: Usiamo il Pumping Lemma
Per i linguaggi context-free

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

Assumiamo per assurdo che L

è context-free

poichè L è context-free e infinito

Possiamo applicare il pumping lemma

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

Pumping Lemma ci dà un magico numero m tale che da li in poi due pezzi della stringa si ripetono.

Prendiamo
una stringa di L con lunghezza almeno m

sia: $a^m b^m a^m b^m \in L$

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

possiamo scrivere: $a^m b^m a^m b^m = uvxyz$

con lunghezze $|vxy| \leq m$ e $|vy| \geq 1$

Pumping Lemma dice:

$uv^i xy^i z \in L$ per tutti $i \geq 0$

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

$$a^m b^m a^m b^m = uvxyz \qquad |vxy| \leq m \qquad |vy| \geq 1$$

Esaminiamo tutti i possibili “posti”
Dove la stringa vxy può essere
in $a^m b^m a^m b^m$

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

$$a^m b^m a^m b^m = uvxyz \qquad |vxy| \leq m \qquad |vy| \geq 1$$

Case 1: vxy È nel primo a^m

$$v = a^{k_1} \qquad y = a^{k_2} \qquad k_1 + k_2 \geq 1$$

$$\overbrace{a\ldots\ldots a}^{m}\ \overbrace{b\ldots\ldots b}^{m}\ \overbrace{a\ldots\ldots a}^{m}\ \overbrace{b\ldots\ldots b}^{m}$$

$$u\ v\ x\ y \qquad z$$

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

$$a^m b^m a^m b^m = uvxyz \qquad |vxy| \le m \qquad |vy| \ge 1$$

Case 1: vxy è nel primo a^m

$$v = a^{k_1} \qquad y = a^{k_2} \qquad k_1 + k_2 \ge 1$$

$$\overbrace{a \ldots\ldots\ldots\ldots\ldots\ldots a}^{m+k_1+k_2}\ \overbrace{b \ldots\ldots b}^{m}\ \overbrace{a \ldots\ldots a}^{m}\ \overbrace{b \ldots\ldots b}^{m}$$

$$u \quad v^2 \quad x \quad y^2 \quad z$$

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

$$a^m b^m a^m b^m = uvxyz \qquad |vxy| \leq m \qquad |vy| \geq 1$$

Case 1: vxy è nel primo a^m

$$a^{m+k_1+k_2} b^m a^m b^m = uv^2xy^2z \quad \notin L$$

$$k_1 + k_2 \geq 1$$

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

$$a^m b^m a^m b^m = uvxyz \qquad |vxy|\leq m \qquad |vy|\geq 1$$

$\boxed{\textbf{Case 1:} \quad vxy \text{ è nel primo } \quad a^m}$

$$a^{m+k_1+k_2} b^m a^m b^m = uv^2xy^2z \quad \notin L$$

Ma dal pumping lemma abbiamo: $uv^2xy^2z \in L$

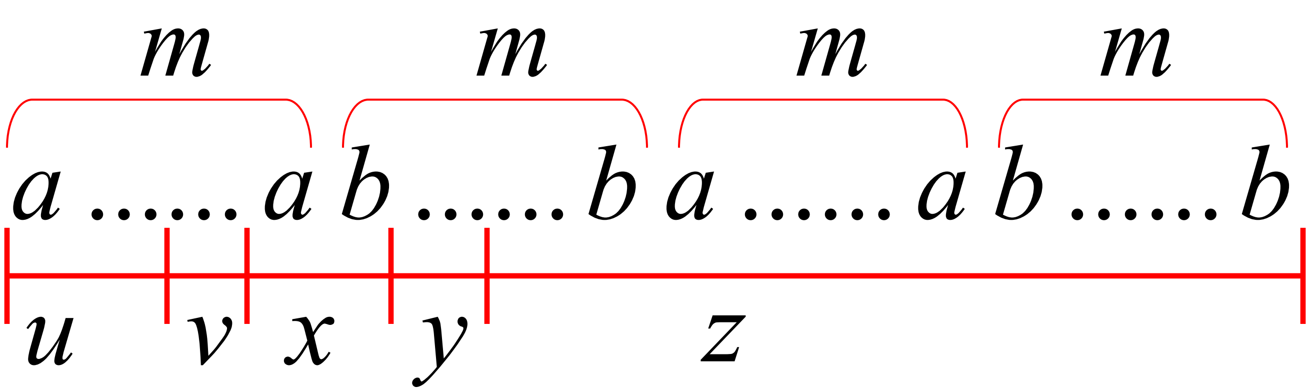
Contradizione!!!

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

$$a^m b^m a^m b^m = uvxyz \qquad |vxy| \le m \qquad |vy| \ge 1$$

Case 2: v È nel primo a^m
y È nel primo b^m

$$v = a^{k_1} \qquad y = b^{k_2} \qquad k_1 + k_2 \ge 1$$



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

$$a^m b^m a^m b^m = uvxyz \qquad |vxy| \leq m \qquad |vy| \geq 1$$

Case 2: v è nel primo a^m
y è nel primo b^m

$$v = a^{k_1} \qquad y = b^{k_2} \qquad k_1 + k_2 \geq 1$$

$$\overbrace{a \ldots\ldots\ldots\ldots a}^{m+k_1}\ \overbrace{b \ldots\ldots\ldots\ldots b}^{m+k_2}\ \overbrace{a \ldots\ldots a}^{m}\ \overbrace{b \ldots\ldots b}^{m}$$

$$u \quad v^2 \quad x \quad y^2 \quad z$$

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

$$a^m b^m a^m b^m = uvxyz \qquad |vxy| \leq m \qquad |vy| \geq 1$$

Case 2: v è nel primo a^m

y è nel primo b^m

$$a^{m+k_1} b^{m+k_2} a^m b^m = uv^2xy^2z \quad \notin L$$

$$k_1 + k_2 \geq 1$$

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

$$a^m b^m a^m b^m = uvxyz \qquad |vxy| \leq m \qquad |vy| \geq 1$$

Case 2: v È nel primo a^m

y È nel primo b^m

$$a^{m+k_1} b^{m+k_2} a^m b^m = uv^2xy^2z \quad \notin L$$

Dal Pumping Lemma: $uv^2xy^2z \in L$

Contradizione!!!

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

$$a^m b^m a^m b^m = uvxyz \qquad |vxy| \le m \qquad |vy| \ge 1$$

Case 3: v Sovrappone sul primo $a^m b^m$

y È nel primo b^m

$$v = a^{k_1} b^{k_2} \qquad y = b^{k_3} \qquad k_1, k_2 \ge 1$$

$$\overbrace{a \ldots\ldots a}^{m}\ \overbrace{b \ldots\ldots b}^{m}\ \overbrace{a \ldots\ldots a}^{m}\ \overbrace{b \ldots\ldots b}^{m}$$

$$u \qquad v \qquad x \quad y \qquad z$$

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

$$a^m b^m a^m b^m = uvxyz \qquad |vxy| \le m \qquad |vy| \ge 1$$

Case 3: v Sovrappone sul primo $a^m b^m$

y è nel primo b^m

$$v = a^{k_1} b^{k_2} \qquad y = b^{k_3} \qquad k_1, k_2 \ge 1$$

$$\overbrace{a \ldots\ldots a}^{m}\ \overbrace{b \ldots b}^{k_2}\ \overbrace{a \ldots a}^{k_1}\ \overbrace{b \ldots\ldots\ldots b}^{m+k_3}\ \overbrace{a \ldots\ldots a}^{m}\ \overbrace{b \ldots\ldots b}^{m}$$

$$u \qquad v^2 \qquad x \qquad y^2 \qquad z$$

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

$$a^m b^m a^m b^m = uvxyz \qquad |vxy| \leq m \qquad |vy| \geq 1$$

Case 3: v Sovrappone sul primo $a^m b^m$

y È nel primo b^m

$$a^m b^{k_2} a^{k_1} b^{m+k_3} a^m b^m = uv^2xy^2z \quad \notin L$$

$$k_1, k_2 \geq 1$$

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

$$a^m b^m a^m b^m = uvxyz \qquad |vxy| \le m \qquad |vy| \ge 1$$

Case 3: v Sovrappone sul primo $a^m b^m$

y È nel primo b^m

$$a^m b^{k_2} a^{k_1} b^{k_3} a^m b^m = uv^2xy^2z \quad \notin L$$

dal Pumping Lemma: $uv^2xy^2z \in L$

assurdo!!!

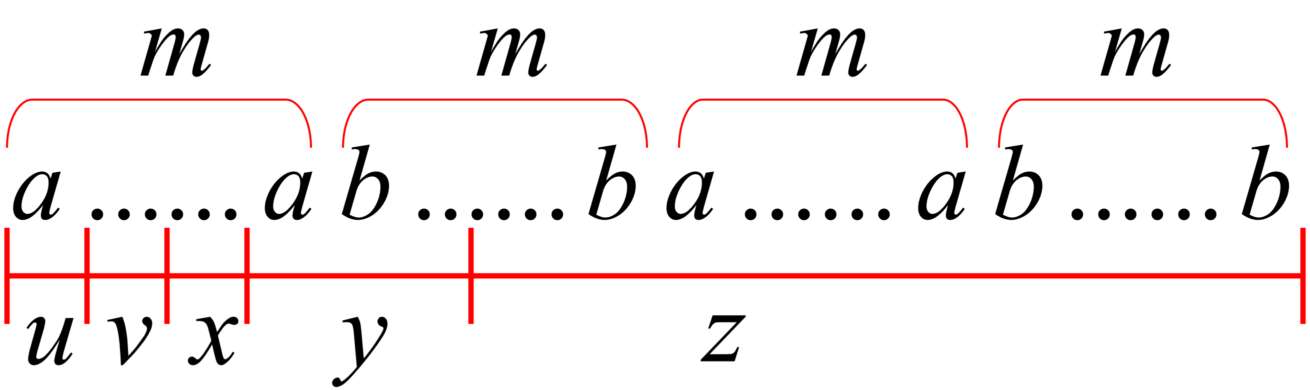
$$L = \{vv : v \in \{a,b\}^*\}$$

$$a^m b^m a^m b^m = uvxyz \qquad |vxy| \leq m \qquad |vy| \geq 1$$

Case 4: v È nel primo a^m

y Sovrappone $a^m b^m$

Analisi simile al caso 3



Altri casi: vxy È dentro $a^m \boxed{b^m} a^m b^m$

or

$$a^m b^m \boxed{a^m} b^m$$

or

$$a^m b^m a^m \boxed{b^m}$$

Analisi simile al caso 1:

$$\boxed{a^m} b^m a^m b^m$$

Altri casi: vxy sovrappone $a^m \boxed{b^m a^m} b^m$

or

$a^m b^m \boxed{a^m b^m}$

Analisi simile ai casi 2,3,4:

$\boxed{a^m b^m} a^m b^m$

Vi sono altri casi da considerare

Poichè $|vxy| \leq m$, è impossibile vxy

Sovrapporre a

$$\boxed{a^m b^m a^m} b^m$$

O, esclusivo

$$a^m \boxed{b^m a^m b^m}$$

O, esclusivo

$$\boxed{a^m b^m a^m b^m}$$

In tutti i casi raggiungiamo un assurdo

quindi: Il punto di partenza che

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

è context-free è sbagliato

Conclusione: L Non è context-free

Linguaggi Non-context free

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

$$\{ww : w \in \{a,b\}\}$$

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

linguaggi Context-free

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

$$\{ww^R : w \in \{a,b\}^*\}$$

linguaggi Non-context free

$$\{a^n b^n c^n : n \ge 0\}$$

$$\{ww : w \in \{a,b\}\}$$

$$\{a^{n^2} b^n : n \ge 0\}$$

$$\{a^{n!} : n \ge 0\}$$

linguaggi Context-free

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

$$\{ww^R : w \in \{a,b\}^*\}$$

Teorema: il linguaggio

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

non è context free

Dim: Usiamo il Pumping Lemma per linguaggi context-free

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

assumiamo per assurdo che L

è context-free

poichè L è context-free ed è infinito

possiamo applicare il pumping lemma

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

Pumping Lemma ci da m

Prendiamo una stringa di L
Con lunghezza almeno m

sia: $a^{m^2} b^m \in L$

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

Prendiamo m: $a^{m^2} b^m = uvxyz$

con lunghezze $|vxy| \leq m$ e $|vy| \geq 1$

Pumping Lemma dice:

$uv^i xy^i z \in L$ Per tutte le $i \geq 0$

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

$$a^{m^2} b^m = uvxyz \qquad |vxy| \leq m \qquad |vy| \geq 1$$

Esaminiamo tutte le possibili posizioni

Della stringa vxy in $a^{m^2} b^m$

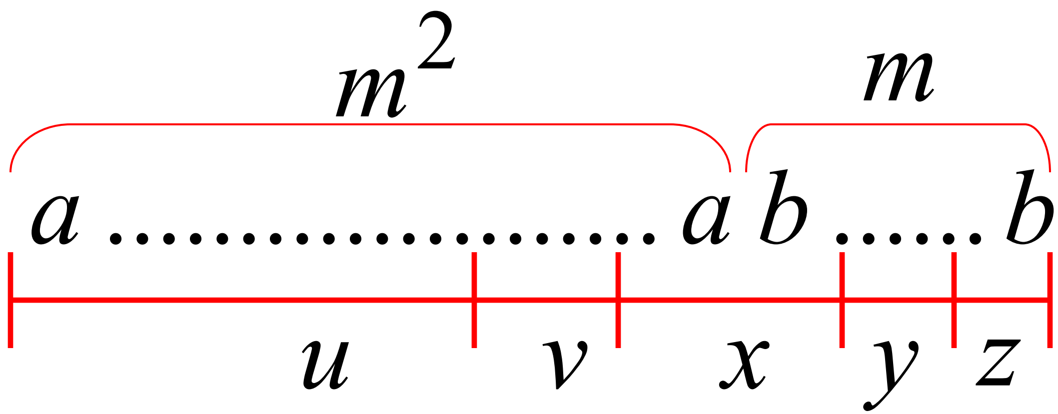
$$L = \{a^{n^2} b^n : n \geq 0\}$$

$$a^{m^2} b^m = uvxyz \qquad |vxy| \leq m \qquad |vy| \geq 1$$

Caso interessante: v è in a^m

y è in b^m

sia $v = a^{k_1}$ $y = b^{k_2}$ con $1 \leq k_1 + k_2 \leq m$



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

$$a^{m^2} b^m = uvxyz \qquad |vxy| \leq m \qquad |vy| \geq 1$$

Sotto caso in cui: $v = a^{k_1}$ $y = b^{k_2}$

Con $1 \leq k_1 + k_2 \leq m$

Sia i=0
abbiamo:

$$\overbrace{a \ldots\ldots\ldots\ldots a}^{m^2 - k_1}\ \overbrace{b \ldots b}^{m - k_2}$$

$$u \quad v^0 \quad x \quad y^0 \quad z$$

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

$$a^{m^2} b^m = uvxyz \qquad |vxy| \leq m \qquad |vy| \geq 1$$

Sia i=0:

$$v = a^{k_1} \qquad y = b^{k_2} \qquad 1 \leq k_1 + k_2 \leq m$$

$$a^{m^2 - k_1} b^{m - k_2} = uv^0 xy^0 z$$

Vediamo il rapporto tra il numero di a e di b quando i=0

$$(m-k_2)^2 \le (m-1)^2$$

$$= m^2 - 2m + 1$$

$$< m^2 - k_1$$

$$\Downarrow$$

$$m^2 - k_1 \neq (m-k_2)^2$$

$$m^2 - k_1 \neq (m - k_2)^2$$

$$\Downarrow$$

$$a^{m^2 - k_1} b^{m - k_2} = uv^0xy^0z \quad \notin L$$

Ma via PL:

$$uv^0xy^0z \quad \in L$$

assurdo!!!

Casi

Caso 1. v e y sono una serie di a, quindi pumping v e y aumentano le a ma non le b

Caso 2. v e y sono una serie di b, quindi pumping v e y aumentano le b ma non le a.

Caso 3 (interessante) visto prima

v è una serie di a e y è una serie di b.

Si potrebbe pensare che crescono secondo le regole del linguaggio. Ma le a dovrebbero crescere rispetto alle brispettando il fatto che le tutte le a sono di lunghezza quadratic rispetto al numero delle b. Questo non è possibile Perche le v, quindi le a, crescono linearmente (allo stesso modo) delle y , ovvero delle b.

dal Pumping Lemma: $uv^0xy^0z \quad \in L$

$$uv^0xy^0z \quad \in L$$

$$a^{m^2-k_1}b^{m-k_2} = uv^0xy^0z \quad \notin L$$

In tutti i casi otteniamo un assurdo

quindi: L'assunzione che

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

è context-free è sbagliata

Conclusion: L Non è context-free

PDA sono equivalenti ai linguaggi Context-Free

Teorema:

$$\left\{\begin{array}{c}\text{Context-Free}\\ \text{linguaggi}\\ \text{(grammatiche)}\end{array}\right\} = \left\{\begin{array}{c}\text{linguaggi}\\ \text{accettati da}\\ \text{PDA}\end{array}\right\}$$

dimostrazione - Step 1:

$$\left\{\begin{array}{c}\text{Context-Free}\\ \text{linguaggi}\\ \text{(grammatiche)}\end{array}\right\} \subseteq \left\{\begin{array}{c}\text{linguaggi}\\ \text{accettati by}\\ \text{PDAs}\end{array}\right\}$$

Traduci ogni grammatica context-free G
In un PDA M con: $L(G) = L(M)$

dimostrazione - step 1

trasforma le

grammatiche Context-Free
in
PDAs

Prendiamo una grammatica context-free G

Tradurremo G in un PDA M tale che:

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

Def. : Una derivazione $S \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \ldots \Rightarrow \alpha_{n-1} \Rightarrow \alpha_n$ si dice leftmost (sinistra) se:

$\forall i = 1,\ldots,n-1 \quad si\ ha \quad \alpha_i = uX\beta_i \quad e \quad \alpha_{i+1} = u\gamma\beta_i, \quad con\ u \in \Sigma^*, X \in N, (X \to \gamma)\ in\ P$

[redacted]

Nel primo caso si scrive : $\alpha_i \underset{L}{\Longrightarrow} \alpha_j \quad (i < j)$.

[redacted]

Useremo solo leftmost

Grammatica consideriamo le Derivazioni Leftmost

Terminali

Leftmost variabile

Variabile o terminali

$$\Rightarrow \cdots$$

$$\Rightarrow xAy$$

$$\Rightarrow \sigma_i \cdots \sigma_j Bzy$$

produzione Applicata

$$A \to \underbrace{\sigma_i \cdots \sigma_j}_{\text{Terminali}} Bz$$

Variabile

Variabili o Terminali

Procedura di conversione:

per ogni
produzione in G

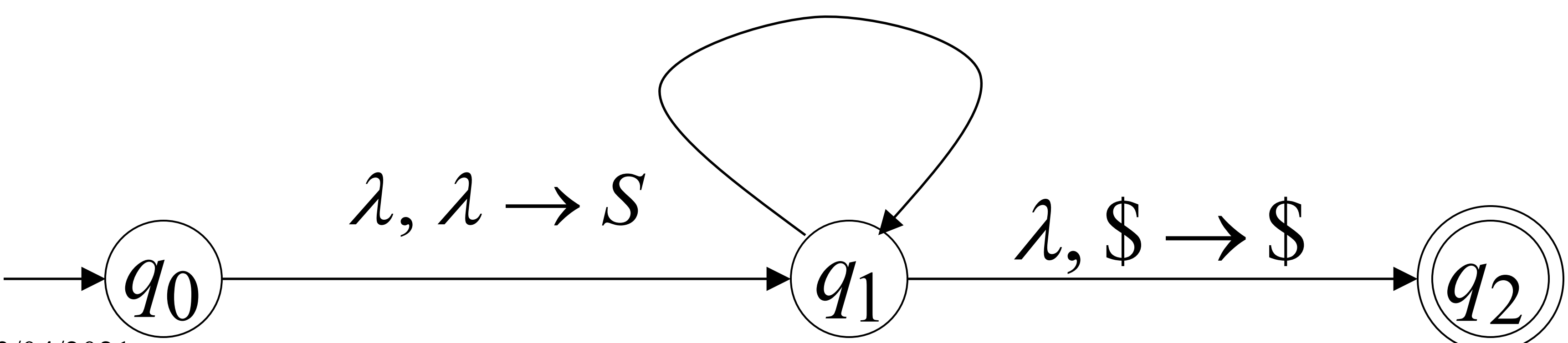
$$A \to w$$

per ogni
terminale in G

$$a$$

Addiziona
le transizioni

$$\lambda, A \to w \qquad a, a \to \lambda$$



esempio

grammatica

$$S \to aSTb$$

$$S \to b$$

$$T \to Ta$$

$$T \to \lambda$$

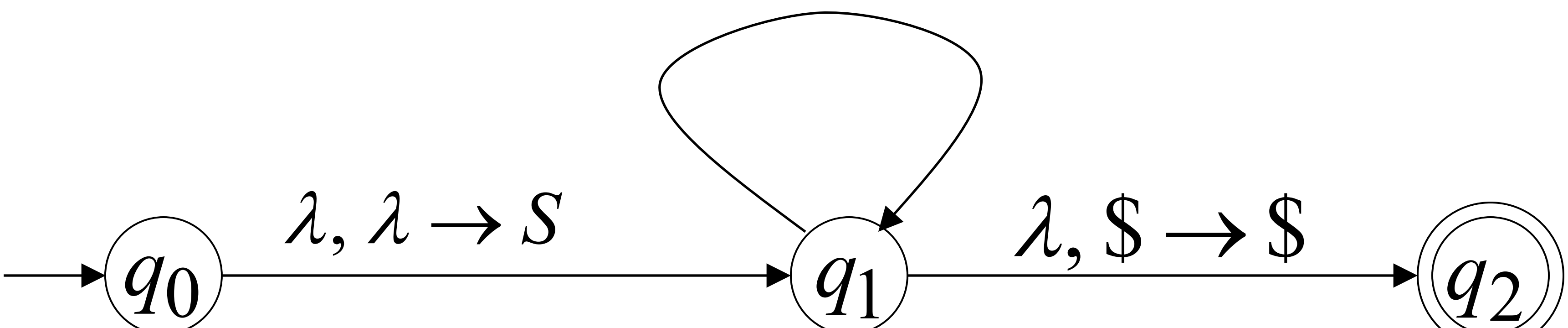
PDA

$$\lambda, S \to aSTb$$

$$\lambda, S \to b$$

$$\lambda, T \to Ta \qquad a, a \to \lambda$$

$$\lambda, T \to \lambda \qquad b, b \to \lambda$$



Esempio:

Input

a	b	a	b

Time 0

$$\lambda, S \rightarrow aSTb$$

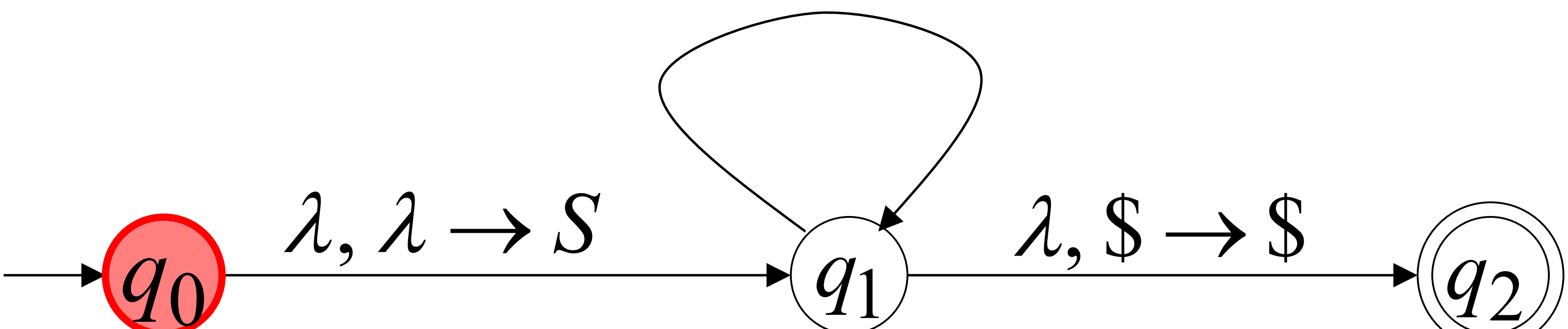
$$\lambda, S \rightarrow b$$

$$\lambda, T \rightarrow Ta \qquad a, a \rightarrow \lambda$$

$$\lambda, T \rightarrow \lambda \qquad b, b \rightarrow \lambda$$

$\$$

Stack



derivazione: S

Input

a	b	a	b

Time 1

$\lambda, S \rightarrow aSTb$

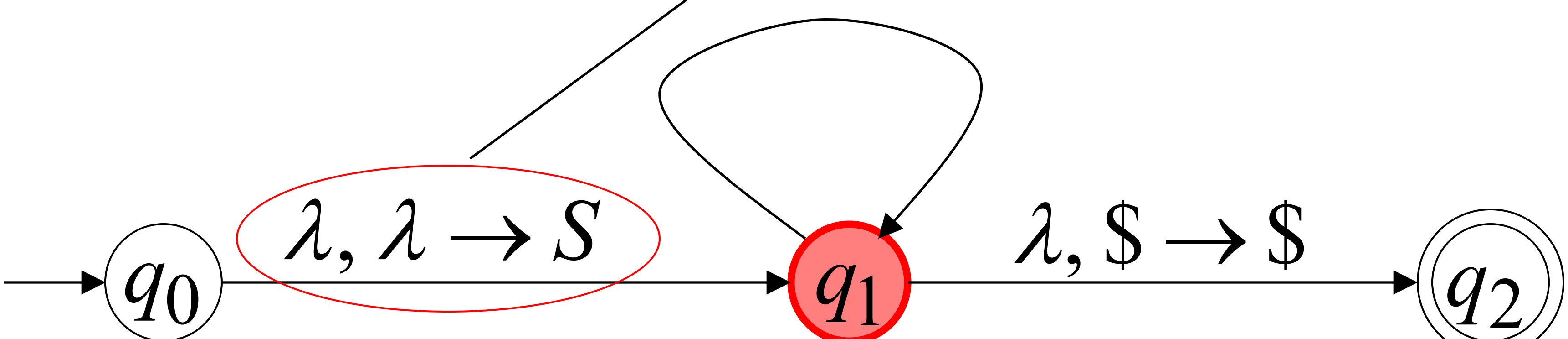
$\lambda, S \rightarrow b$

$\lambda, T \rightarrow Ta$ $\quad a, a \rightarrow \lambda$

$\lambda, T \rightarrow \lambda$ $\quad b, b \rightarrow \lambda$

S
$\$$

Stack



derivazione: $S \Rightarrow aSTb$

Input

a	b	a	b

Time 2

$\lambda, S \rightarrow aSTb$

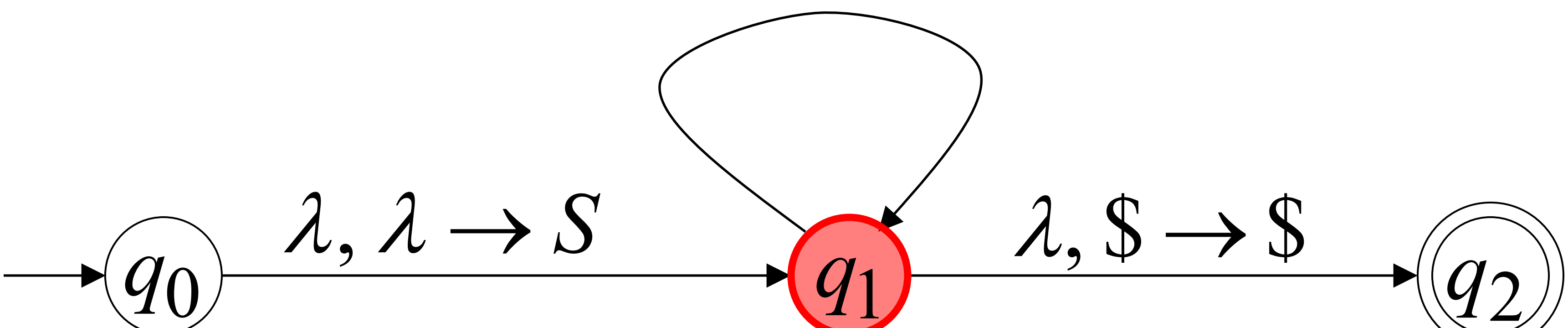
$\lambda, S \rightarrow b$

$\lambda, T \rightarrow Ta$ $\qquad a, a \rightarrow \lambda$

$\lambda, T \rightarrow \lambda$ $\qquad b, b \rightarrow \lambda$

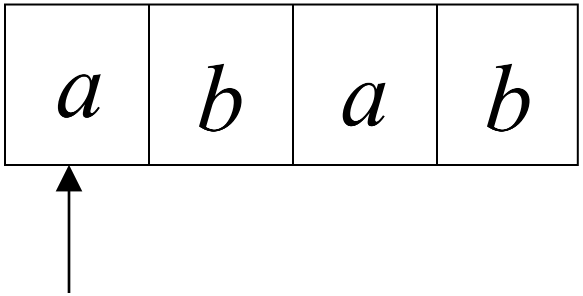
Stack

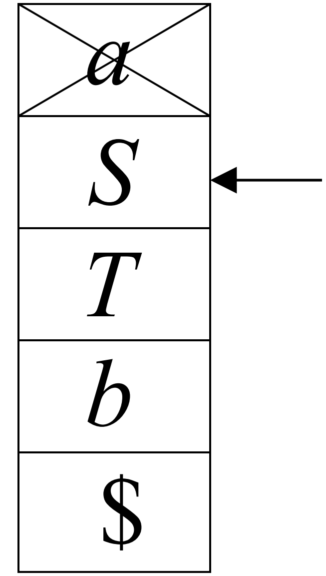
a
S
T
b
$\$$



derivazione: $S \Rightarrow aSTb$

Input





Stack

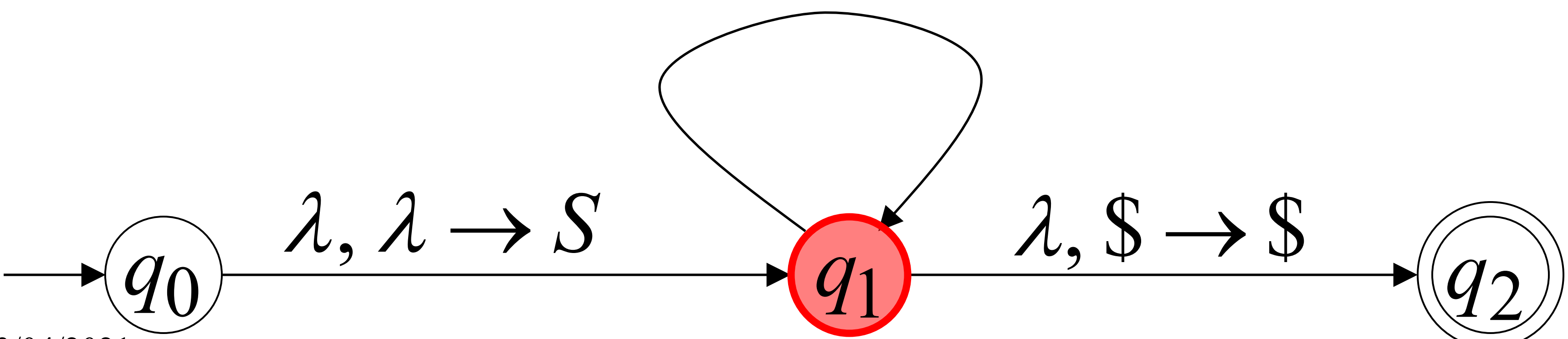
Time 3

$$\lambda, S \rightarrow aSTb$$

$$\lambda, S \rightarrow b$$

$$\lambda, T \rightarrow Ta \qquad a, a \rightarrow \lambda$$

$$\lambda, T \rightarrow \lambda \qquad b, b \rightarrow \lambda$$



derivazione: $S \Rightarrow aSTb \Rightarrow abTb$

Input

a	b	a	b

Time 4

$\lambda, S \to aSTb$

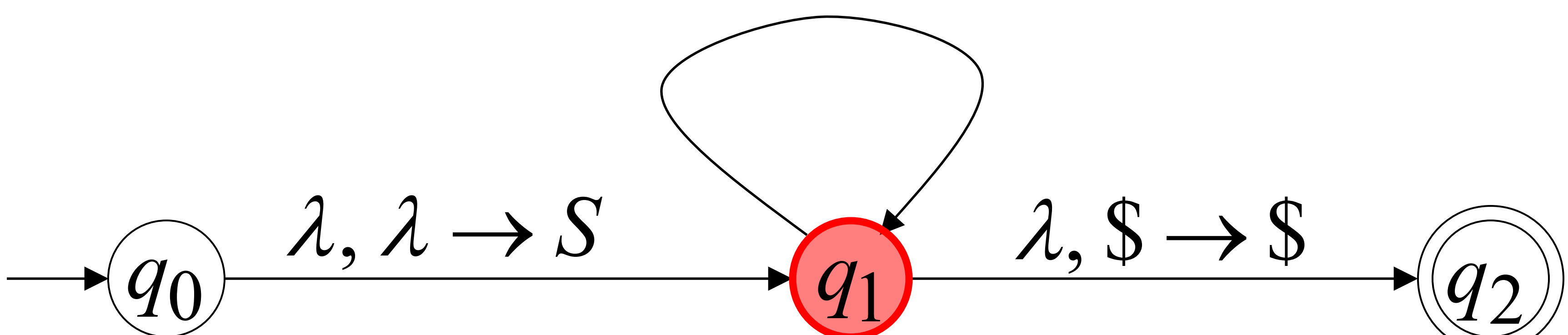
$\lambda, S \to b$

$\lambda, T \to Ta$ $\quad$ $a, a \to \lambda$

$\lambda, T \to \lambda$ $\quad$ $b, b \to \lambda$

Stack

b
T
b
$\$$



derivazione: $S \Rightarrow aSTb \Rightarrow abTb$

Input

a	b	a	b

Time 5

$\lambda, S \rightarrow aSTb$

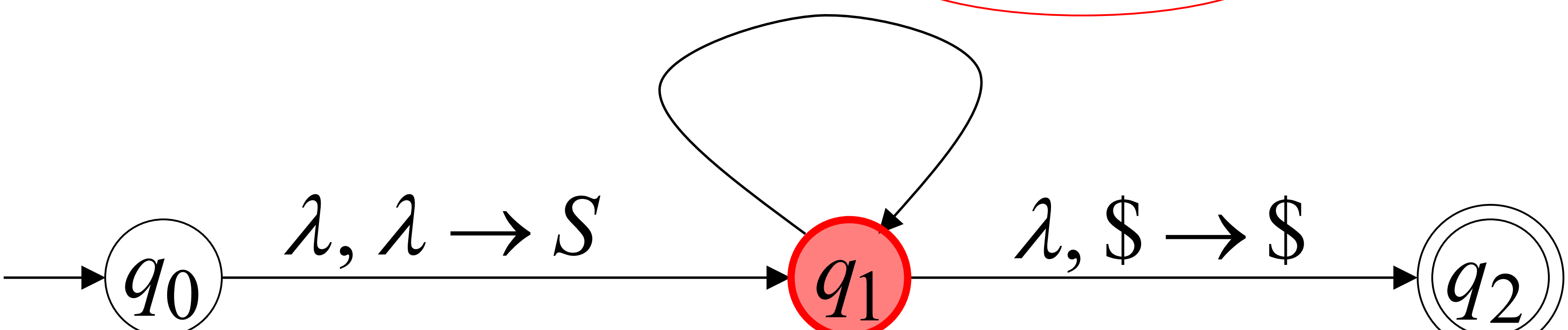
$\lambda, S \rightarrow b$

$\lambda, T \rightarrow Ta$ $\quad a, a \rightarrow \lambda$

$\lambda, T \rightarrow \lambda$ $\quad b, b \rightarrow \lambda$

$\cancel{b}$
T
b
$\$$

Stack



derivazione: $S \Rightarrow aSTb \Rightarrow abTb \Rightarrow abTab$

Input

a	b	a	b

Time 6

$\lambda, S \rightarrow aSTb$

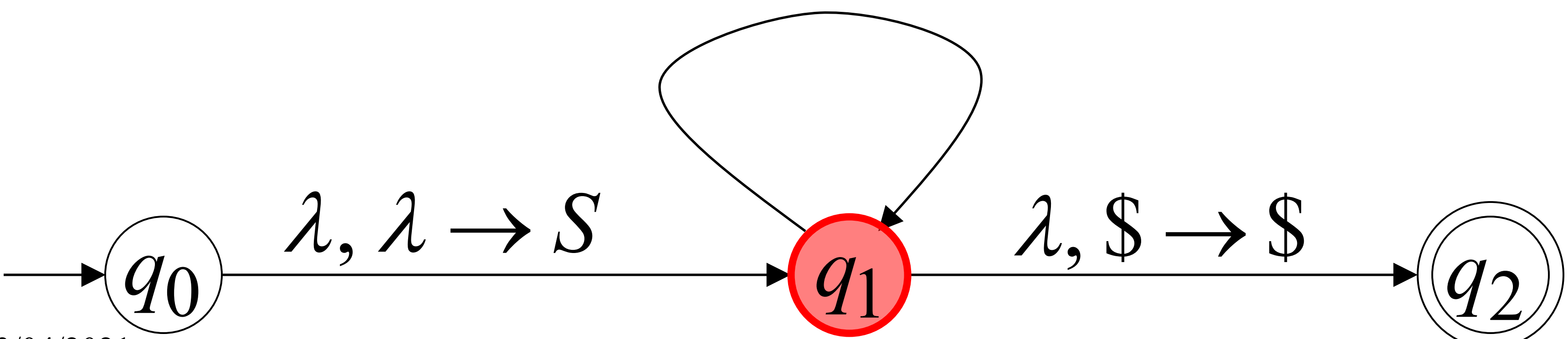
$\lambda, S \rightarrow b$

$\lambda, T \rightarrow Ta$ $\quad a, a \rightarrow \lambda$

$\lambda, T \rightarrow \lambda$ $\quad b, b \rightarrow \lambda$

Stack

T
a
b
$\$$



derivazione: $\underline{S \Rightarrow aSTb \Rightarrow abTb \Rightarrow abTab} \Rightarrow abab$

Input

a	b	a	b

Stack

$\cancel{T}$
a
b
$\$$

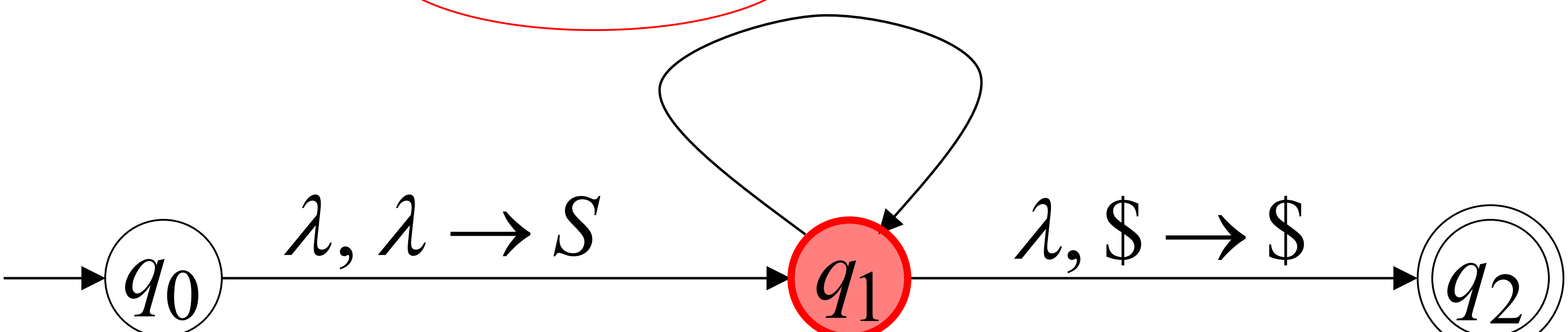
Time 7

$\lambda, S \rightarrow aSTb$

$\lambda, S \rightarrow b$

$\lambda, T \rightarrow Ta$ $\quad a, a \rightarrow \lambda$

$\lambda, T \rightarrow \lambda$ $\quad b, b \rightarrow \lambda$



derivazione: $S \Rightarrow aSTb \Rightarrow abTb \Rightarrow abTab \Rightarrow abab$

Input

a	b	a	b

Stack

$\cancel{a}$
b
$\$$

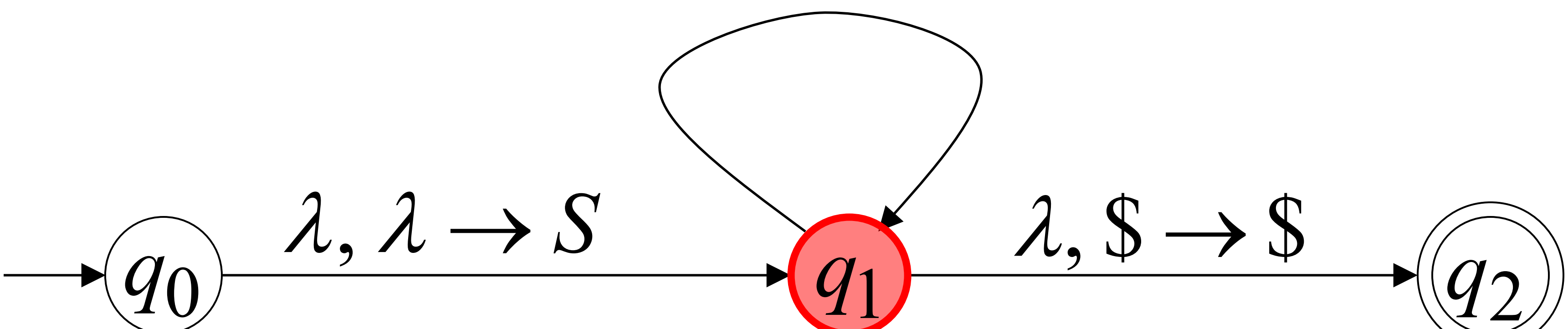
Time 8

$\lambda, S \rightarrow aSTb$

$\lambda, S \rightarrow b$

$\lambda, T \rightarrow Ta$ $\quad a, a \rightarrow \lambda$

$\lambda, T \rightarrow \lambda$ $\quad b, b \rightarrow \lambda$



derivazione: $S \Rightarrow aSTb \Rightarrow abTb \Rightarrow abTab \Rightarrow abab$

Input

a	b	a	b

Stack

$\cancel{b}$
$\$$

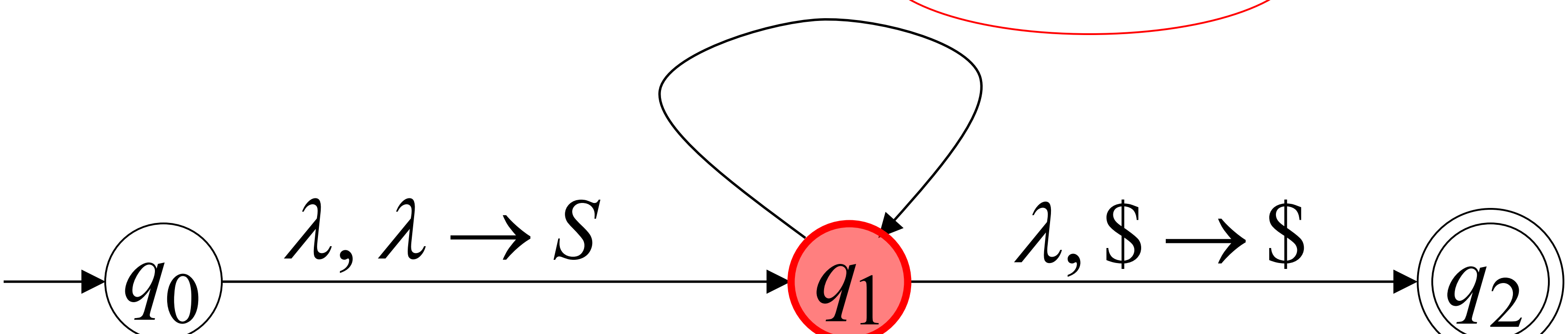
Time 9

$\lambda, S \rightarrow aSTb$

$\lambda, S \rightarrow b$

$\lambda, T \rightarrow Ta$ $\quad a, a \rightarrow \lambda$

$\lambda, T \rightarrow \lambda$ $\quad b, b \rightarrow \lambda$



derivazione: $S \Rightarrow aSTb \Rightarrow abTb \Rightarrow abTab \Rightarrow abab$

Input

a	b	a	b

$\lambda, S \rightarrow aSTb$

$\lambda, S \rightarrow b$

$\lambda, T \rightarrow Ta$ $\quad a, a \rightarrow \lambda$

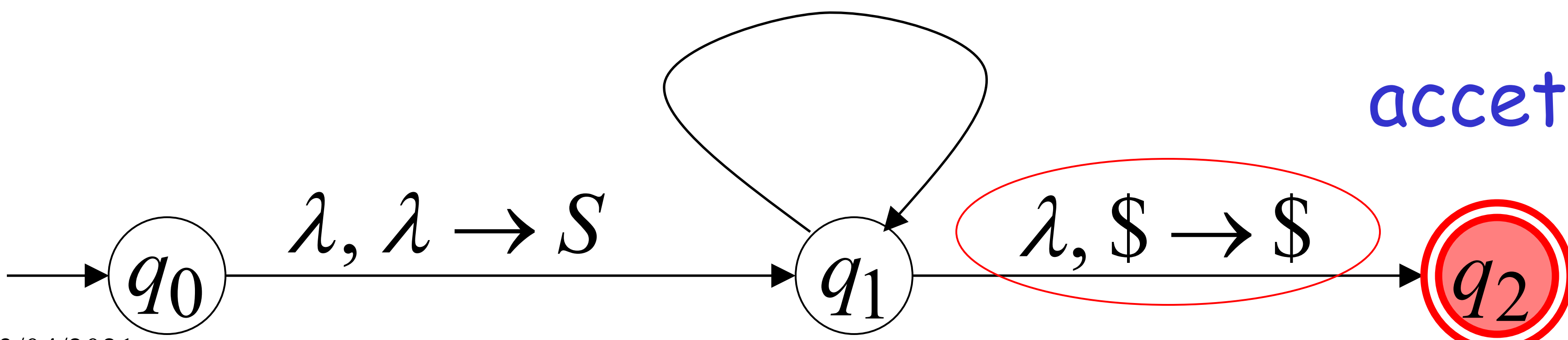
$\lambda, T \rightarrow \lambda$ $\quad b, b \rightarrow \lambda$

Time 10

$\$$

Stack

accettaz



Procedura di conversione:

per ogni produzione in G

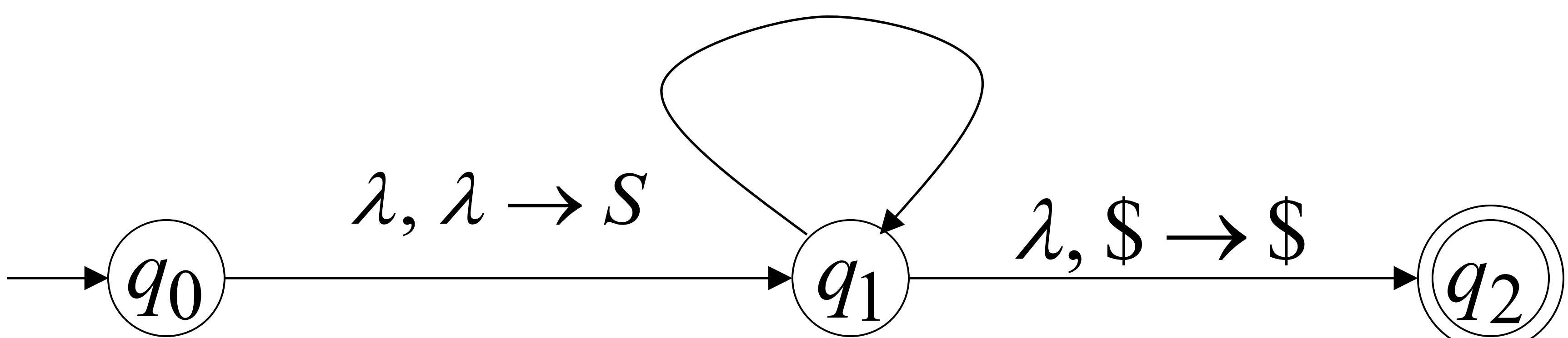
$$A \rightarrow w$$

per ogni terminale in G

$$a$$

Addiziona le transizioni

$$\lambda, A \rightarrow w \qquad a, a \rightarrow \lambda$$



esempio

grammatica

$$S \to aSTb$$

$$S \to b$$

$$T \to Ta$$

$$T \to \lambda$$

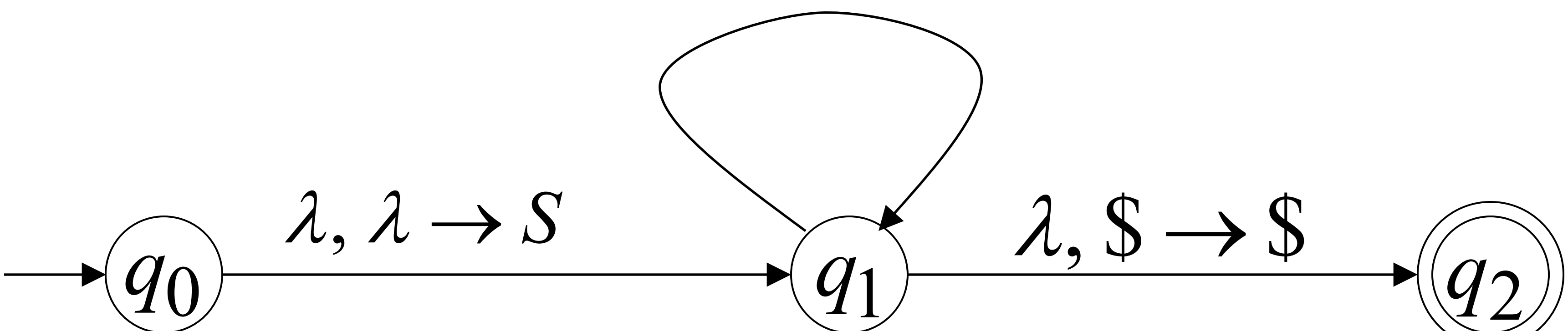
PDA

$$\lambda, S \to aSTb$$

$$\lambda, S \to b$$

$$\lambda, T \to Ta \qquad a, a \to \lambda$$

$$\lambda, T \to \lambda \qquad b, b \to \lambda$$



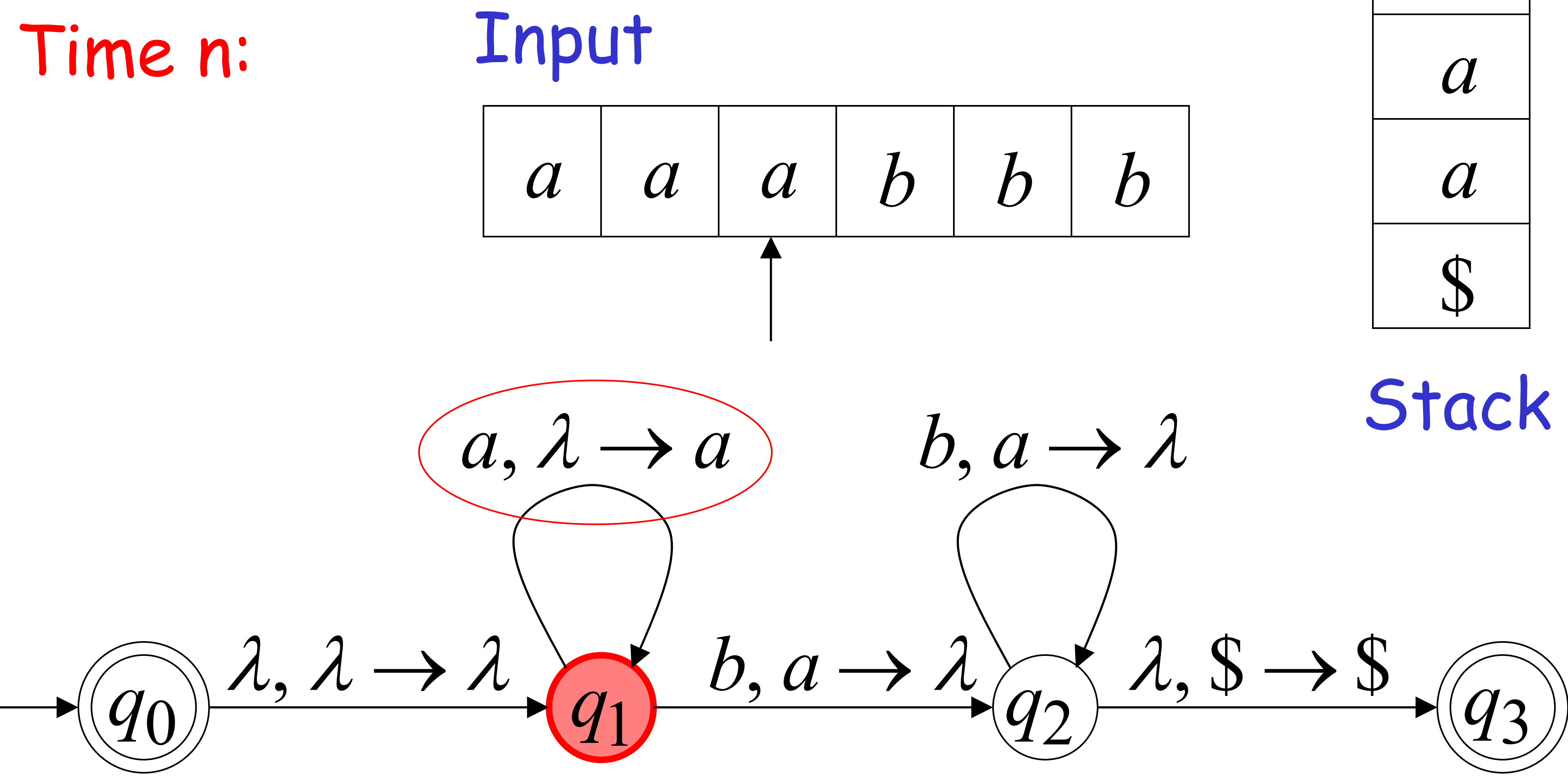
Proviamo a dimostrare il teorema

Ricordiamo la notazione di configurazione istantanea e di passo di computazione.

configurazione istantanea definizione:

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

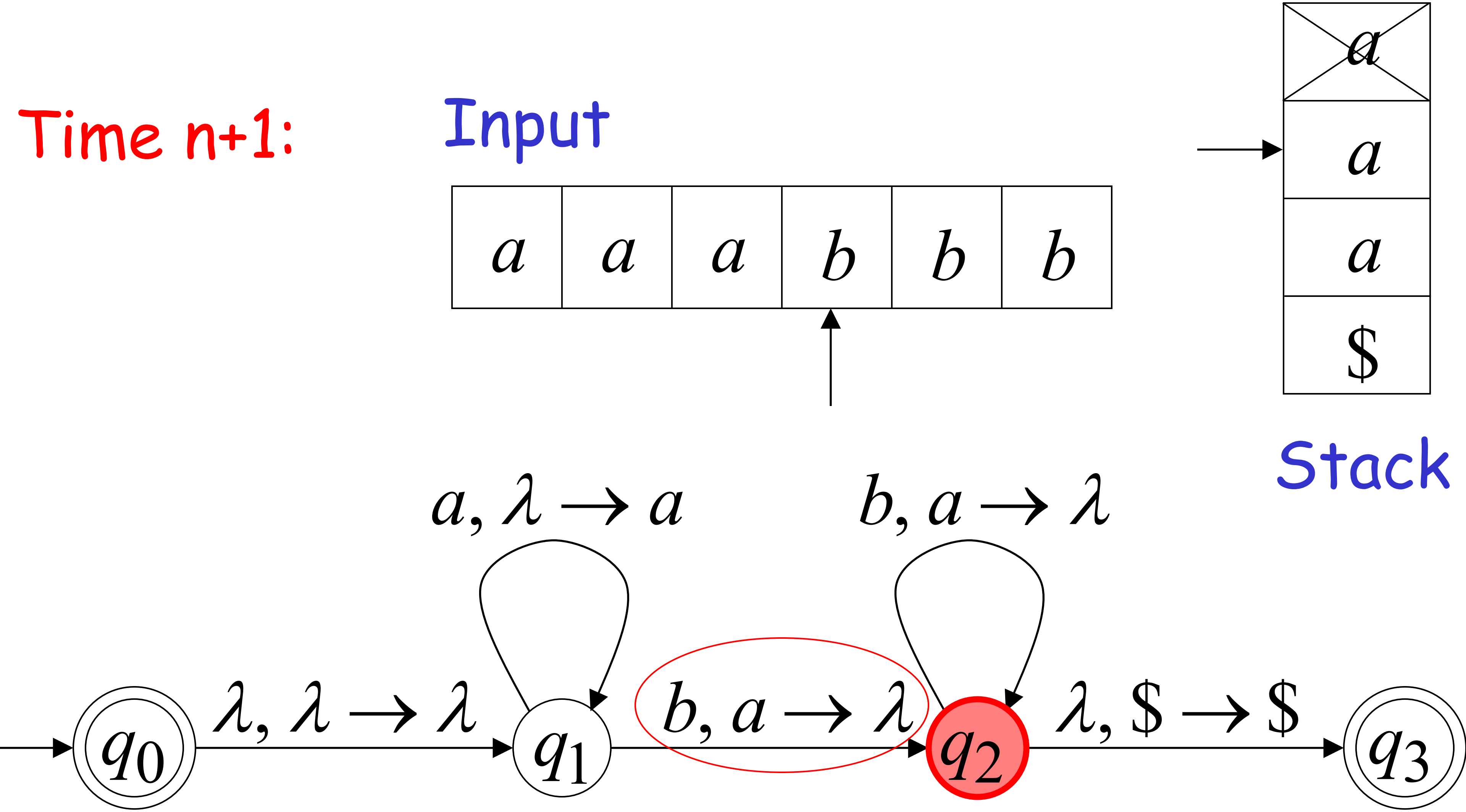
Time n:



configurazione istantanea

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

Time n+1:



passo di computazione.

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

Time n

Time n+1

PDA simula le derivazioni leftmost

grammatica

Leftmost derivazione

$$S$$

$$\Rightarrow \cdots$$

$$\Rightarrow \sigma_1 \cdots \sigma_k X_1 \cdots X_m$$

$$\Rightarrow \cdots$$

$$\Rightarrow \sigma_1 \cdots \sigma_k \sigma_{k+1} \cdots \sigma_n$$

PDA calcolo

$$(q_0, \sigma_1 \cdots \sigma_k \sigma_{k+1} \cdots \sigma_n, \$)$$

$$\succ (q_1, \sigma_1 \cdots \sigma_k \sigma_{k+1} \cdots \sigma_n, S\$)$$

$$\succ \cdots$$

$$\succ (q_1, \sigma_{k+1} \cdots \sigma_n, X_1 \cdots X_m \$)$$

$$\succ \cdots$$

$$\succ (q_2, \lambda, \$)$$

simboli esaminati

Contenuti Dello Stack

Procedura di conversione:

per ogni produzione in G

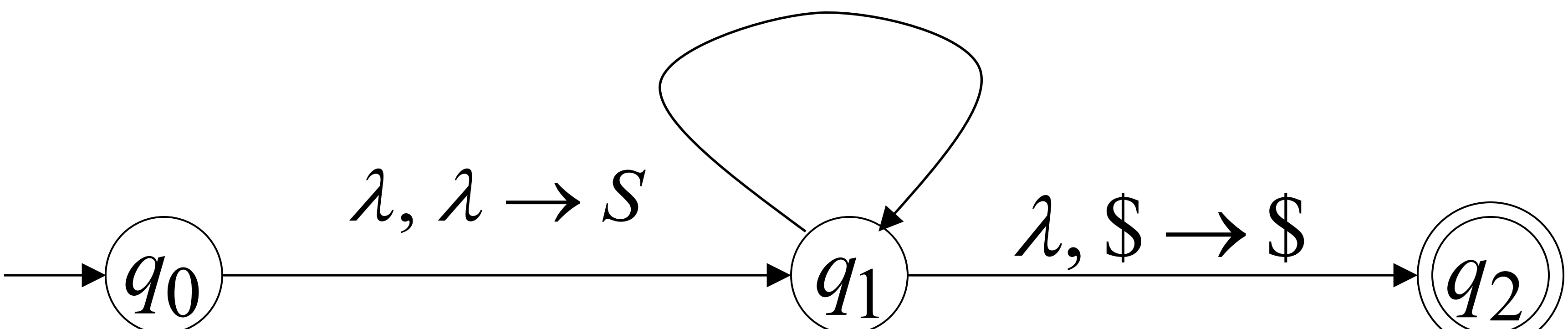
$$A \rightarrow w$$

per ogni terminale in G

$$a$$

Addiziona le transizioni

$$\lambda, A \rightarrow w \qquad a, a \rightarrow \lambda$$



esempio

grammatica

$S \to aSTb$

$S \to b$

$T \to Ta$

$T \to \lambda$

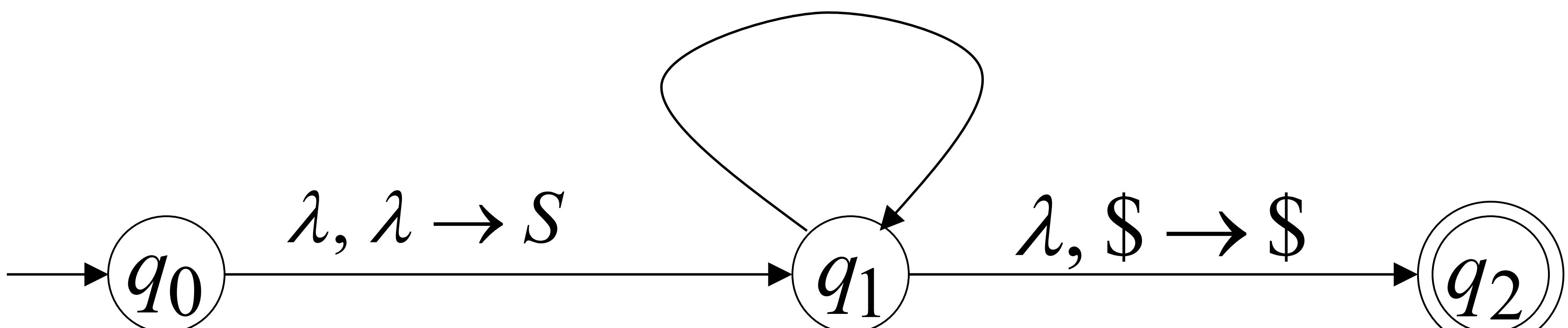
PDA

$\lambda, S \to aSTb$

$\lambda, S \to b$

$\lambda, T \to Ta$ $\quad a, a \to \lambda$

$\lambda, T \to \lambda$ $\quad b, b \to \lambda$



grammatica
Derivazione Leftmost

Calcolo PDA

$$S$$

$$(q_0, abab, \$)$$
$$\succ (q_1, abab, S\$)$$

$$\Rightarrow aSTb$$

$$\succ (q_1, bab, STb\$)$$

$$\Rightarrow abTb$$

$$\succ (q_1, bab, bTb\$)$$
$$\succ (q_1, ab, Tb\$)$$

$$\Rightarrow abTab$$

$$\succ (q_1, ab, Tab\$)$$

$$\Rightarrow abab$$

$$\succ (q_1, ab, ab\$)$$
$$\succ (q_1, b, b\$)$$
$$\succ (q_1, \lambda, \$)$$
$$\succ (q_2, \lambda, \$)$$

grammatica
Leftmost derivazione

calcolo del PDA

$$\Rightarrow \cdots \qquad \succ \cdots$$

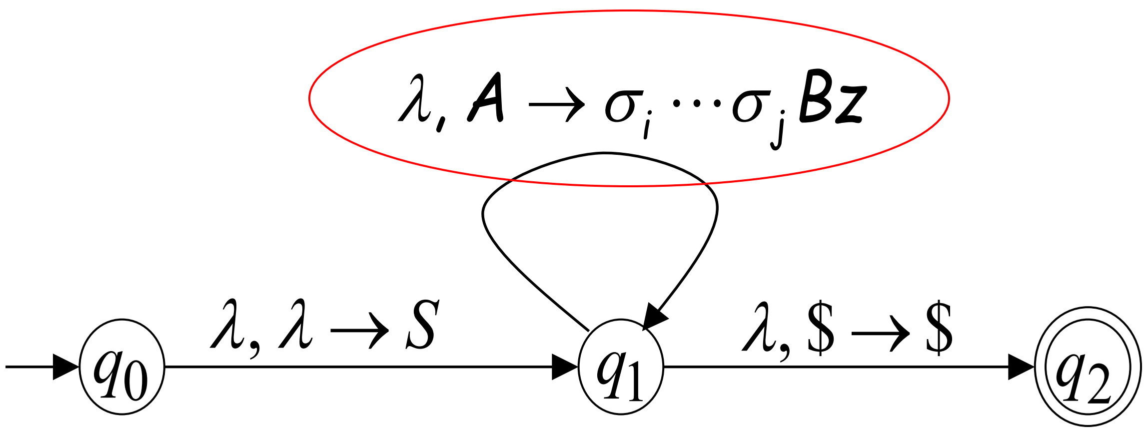
$$\Rightarrow xAy \qquad \succ (q_1, \sigma_i \cdots \sigma_n, Ay\$)$$

$$\Rightarrow \sigma_i \cdots \sigma_j Bzy \qquad \succ (q_1, \sigma_i \cdots \sigma_n, \sigma_i \cdots \sigma_j Bzy\$)$$

Produzione Applicata

$$A \rightarrow \sigma_i \cdots \sigma_j Bz$$

Transizione applicata



grammatica
Leftmost derivazione

calcolo del PDA

$$\Rightarrow \cdots \qquad \succ \cdots$$

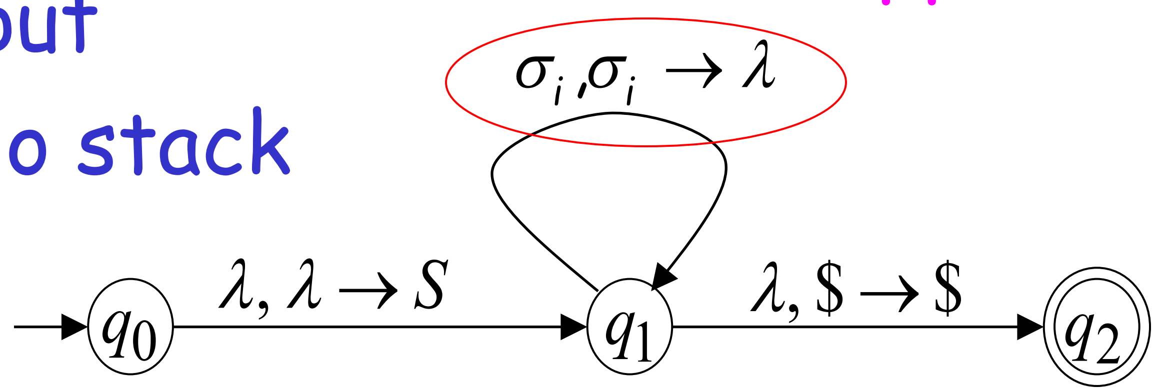
$$\Rightarrow xAy \qquad \succ (q_1, \sigma_i \cdots \sigma_n, Ay\$)$$

$$\Rightarrow \sigma_i \cdots \sigma_j Bzy \qquad \succ (q_1, \sigma_i \cdots \sigma_n, \sigma_i \cdots \sigma_j Bzy\$)$$

$$\succ (q_1, \sigma_{i+1} \cdots \sigma_n, \sigma_{i+1} \cdots \sigma_j Bzy\$)$$

Transizione applicata

leggi σ_i dall'input
E rimuovilo dallo stack



grammatica

Leftmost derivazione

$\Rightarrow \cdots$

$\Rightarrow xAy$

$\Rightarrow \sigma_i \cdots \sigma_j Bzy$

PDA calcolo

$\succ \cdots$

$\succ (q_1, \sigma_i \cdots \sigma_n, Ay\$)$

$\succ (q_1, \sigma_i \cdots \sigma_n, \sigma_i \cdots \sigma_j Bzy\$)$

$\succ (q_1, \sigma_{i+1} \cdots \sigma_n, \sigma_{i+1} \cdots \sigma_j Bzy\$)$

$\succ \cdots$

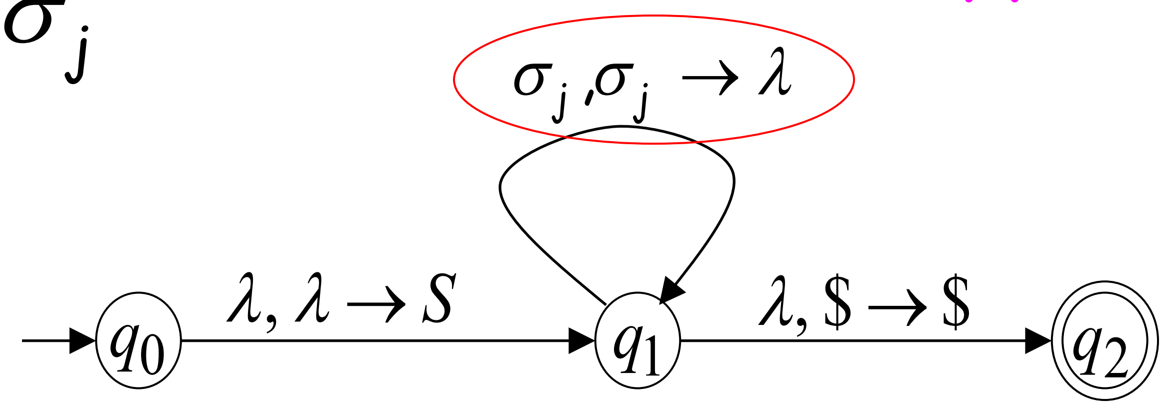
$\succ (q_1, \sigma_{j+1} \cdots \sigma_n, Bzy\$)$

ultima Transizione applicat

tutti simboli $\sigma_i \cdots \sigma_j$

Sono stati rimossi

Dal top dello stack



Il processo viene ripetuto con successiva variabile leftmost

$$\Rightarrow \cdots$$
$$\Rightarrow xAy$$
$$\Rightarrow \sigma_i \cdots \sigma_j Bzy$$
$$\Rightarrow \sigma_i \cdots \sigma_j \sigma_{j+1} \cdots \sigma_k Cpzy$$

$$\succ \cdots$$
$$\succ (q_1, \sigma_{j+1} \cdots \sigma_n, Bzy\$)$$
$$\succ (q_1, \sigma_{j+1} \cdots \sigma_n, \sigma_{j+1} \cdots \sigma_k Cpzy\$)$$
$$\succ \cdots$$
$$\succ (q_1, \sigma_{k+1} \cdots \sigma_n, Cpzy\$)$$

Produzione applicata

$$B \rightarrow \sigma_{j+1} \cdots \sigma_k Cp$$

E cosi via

dimostrato che:

grammatica G
Genera la
stringa w

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

Se e solo se

PDA M
accetta w

$$(q_0, w, \$) \succ (q_2, \lambda, \$)$$

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

Proof - step 2

Tradurre
i PDA
in
grammatiche Context-Free

Prendi un qualsiasi PDA M

Traduremmo M

In una grammatica G context-free

tale che: $L(M) = L(G)$

Prima di tutto modifica PDA M tale che:

1. PDA ha un solo stato di accettazione

2. Svuota tutta la pila prima di accettare

3. Per ogni transizione o

push un simbolo

oppure

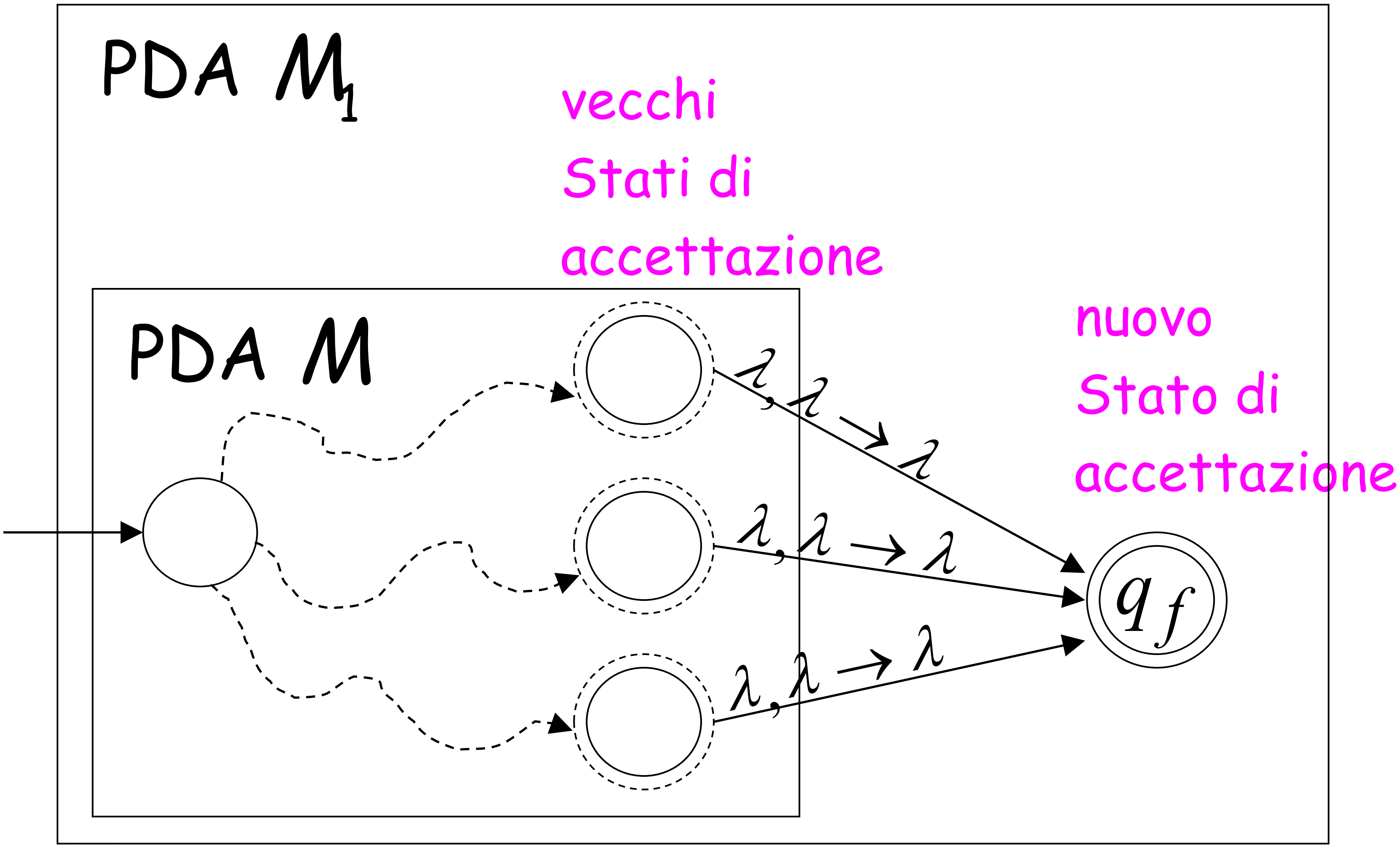
pop un simbolo

ma non entrambe le cose

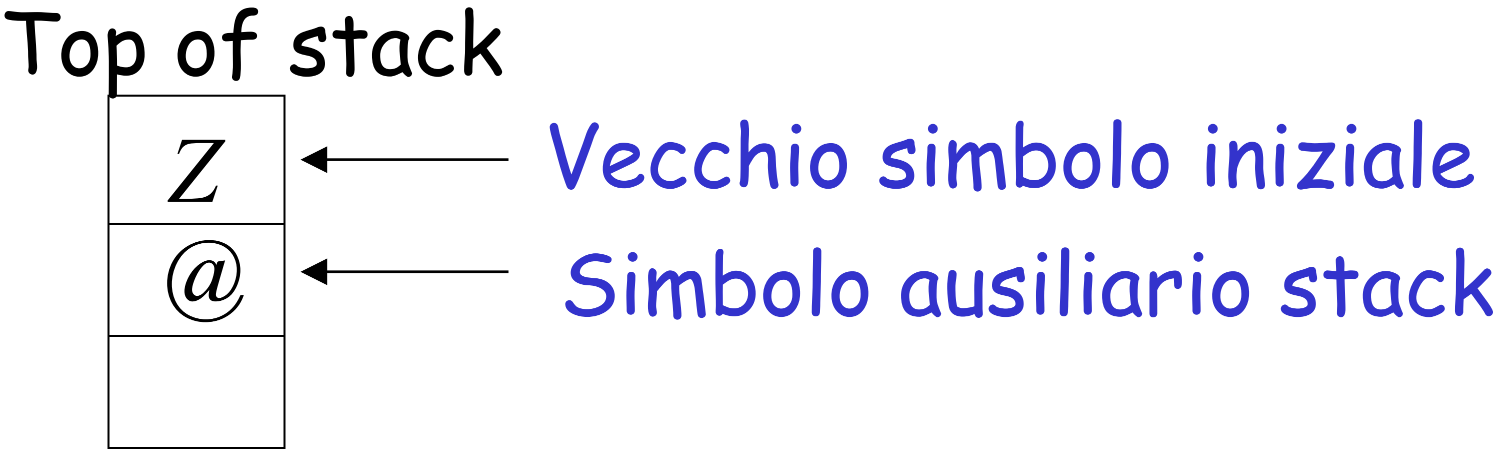
Inoltre abbiamo:

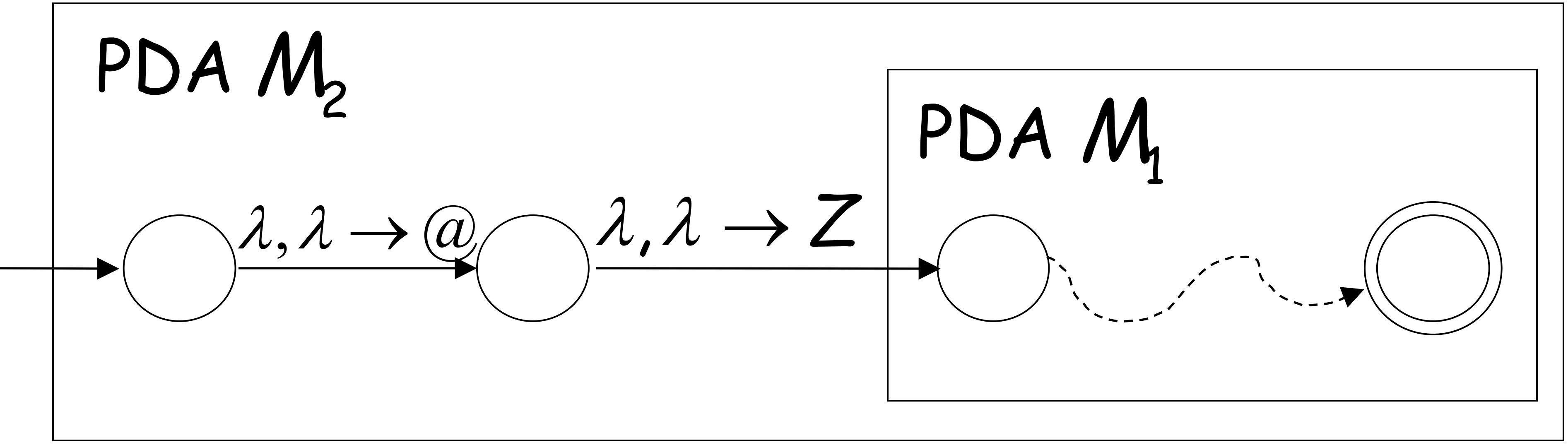
- - nuovo simbolo iniziale @
- - stato di accettazione nello stack solo @

1. il PDA deve avere un solo stato di accettazione



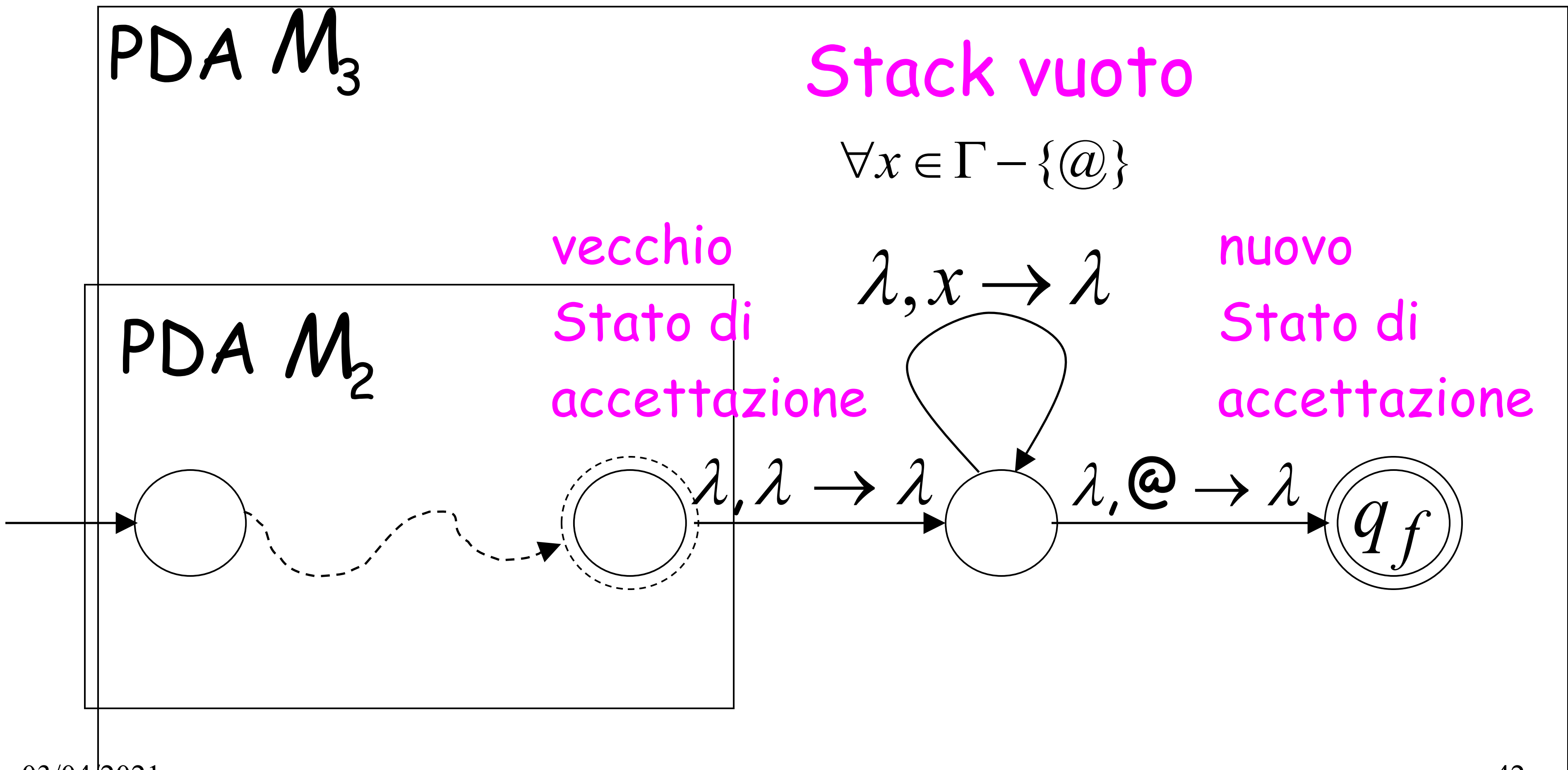
2. Nuovo simbolo iniziale dello stack





M_1 pensa ancora che Z è il simbolo iniziale

3. Nello stato di accetazione lo stack contiene solo il simbolo @

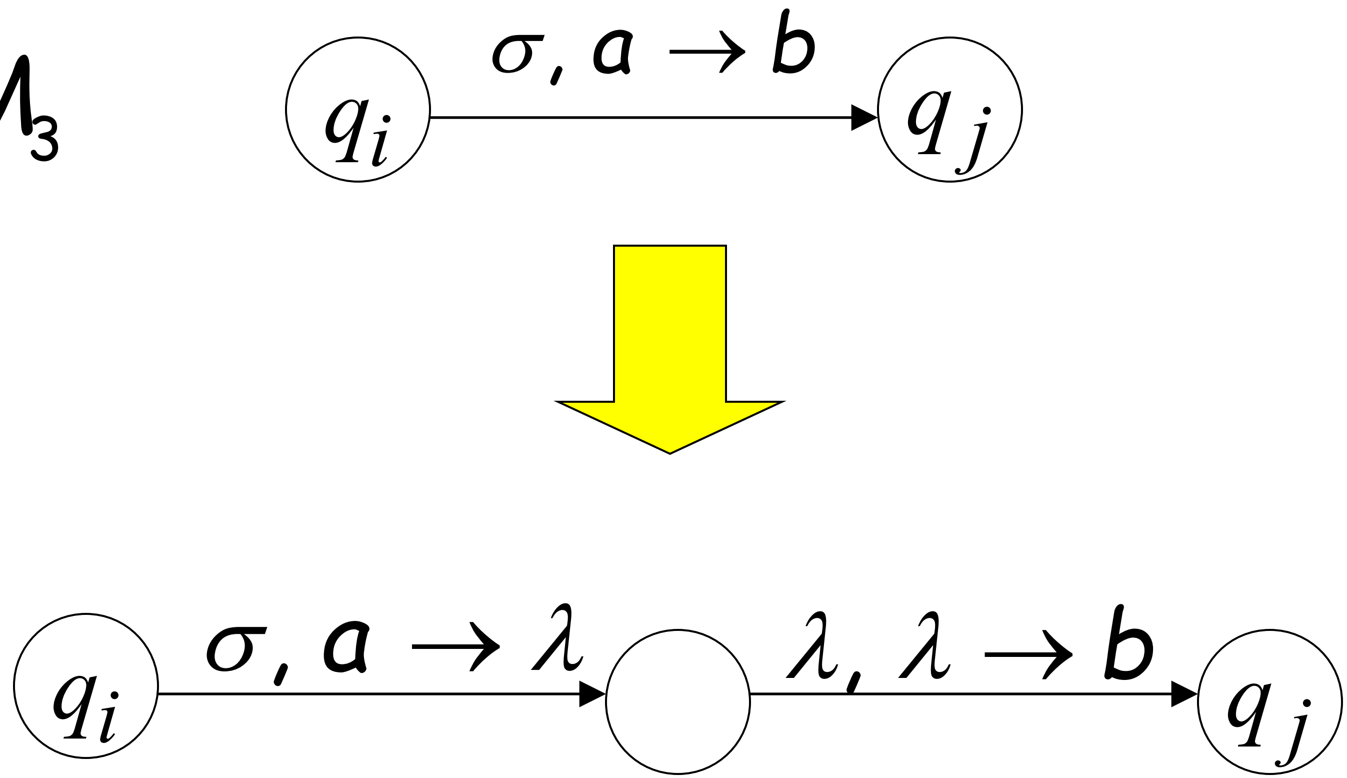


4. Ogni transizione ha un push oppure un pop

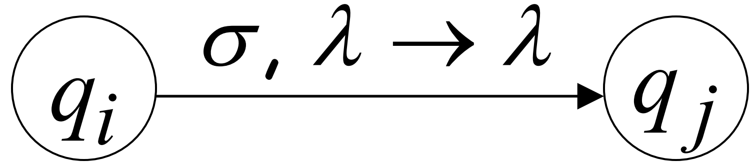
Mai le due cose insieme.

PDA M_3

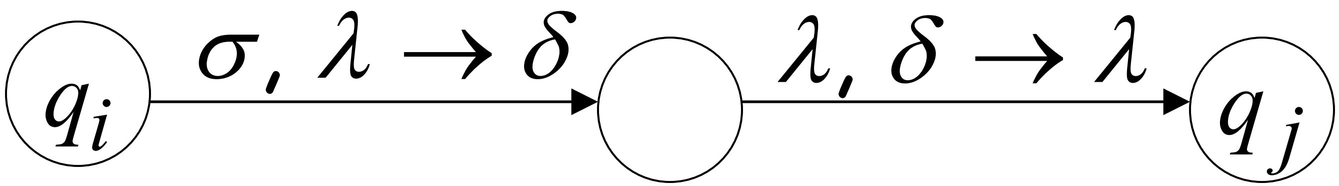
PDA M_4



PDA M_3



PDA M_4

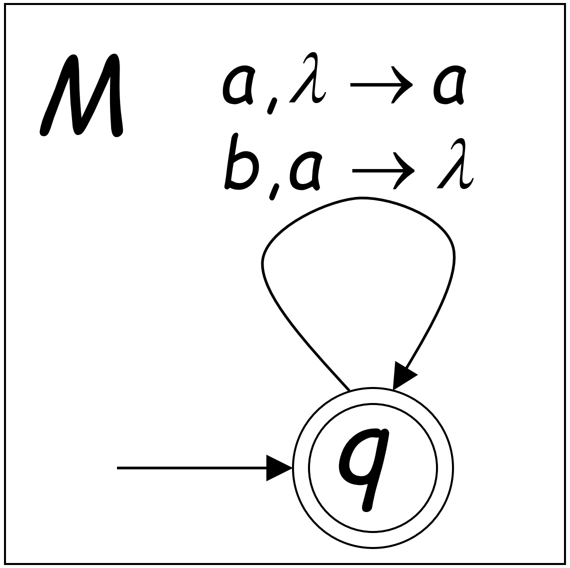


dove δ è un qualsiasi simbolo dell'alfabeto di input

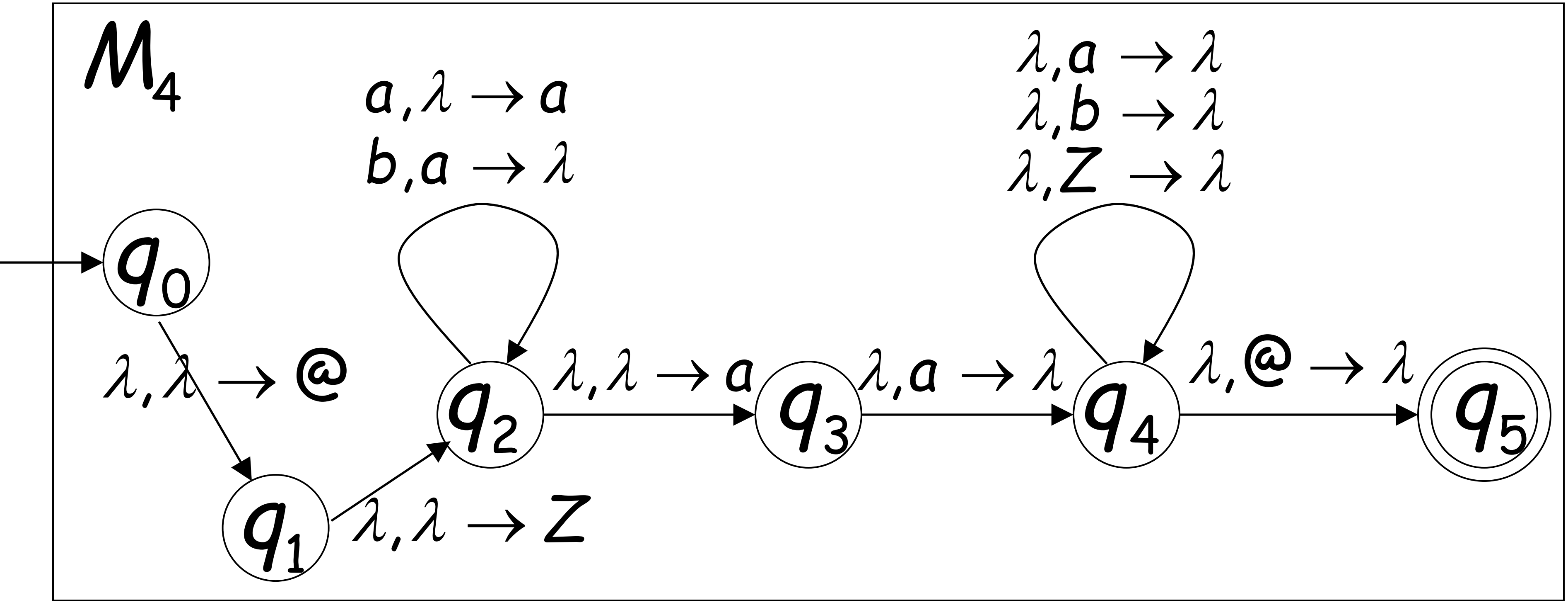
PDA M_4 è il PDA completamente modificato secondo le regole precedenti

Notiamo che il nuovo simbolo iniziale @ non è mai usato in nessuna transizione

Esempio:



M
$a, \lambda \rightarrow a$
$b, a \rightarrow \lambda$
q



M_4
q_0
$\lambda, \lambda \rightarrow @$
q_1
$\lambda, \lambda \rightarrow Z$
$a, \lambda \rightarrow a$
$b, a \rightarrow \lambda$
q_2
$\lambda, \lambda \rightarrow a$
q_3
$\lambda, a \rightarrow \lambda$
$\lambda, a \rightarrow \lambda$
$\lambda, b \rightarrow \lambda$
$\lambda, Z \rightarrow \lambda$
q_4
$\lambda, @ \rightarrow \lambda$
q_5

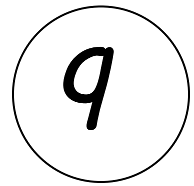
Costruzione della grammatica

variabili: A_{q_i, q_j}

stati del PDA

PDA

caso 1: per ogni stato

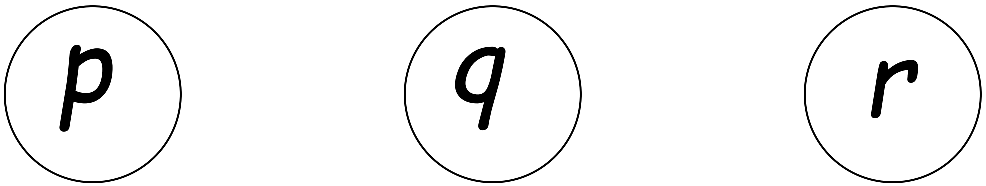


grammatica

$$A_{qq} \rightarrow \lambda$$

PDA

caso 2: per ogni tre stati



---

grammatica

$$A_{pq} \to A_{pr} A_{rq}$$

PDA

caso 3: per ogni coppia di transizioni

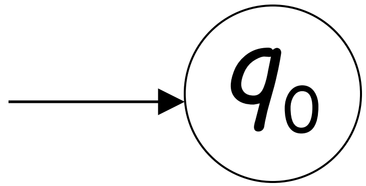
$$p \xrightarrow{a,\lambda \to t} r \qquad\qquad s \xrightarrow{b,t \to \lambda} q$$

grammatica

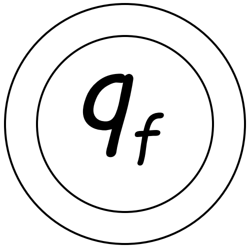
$$A_{pq} \to aA_{rs}b$$

PDA

Stato Iniziale



Stato accettazione



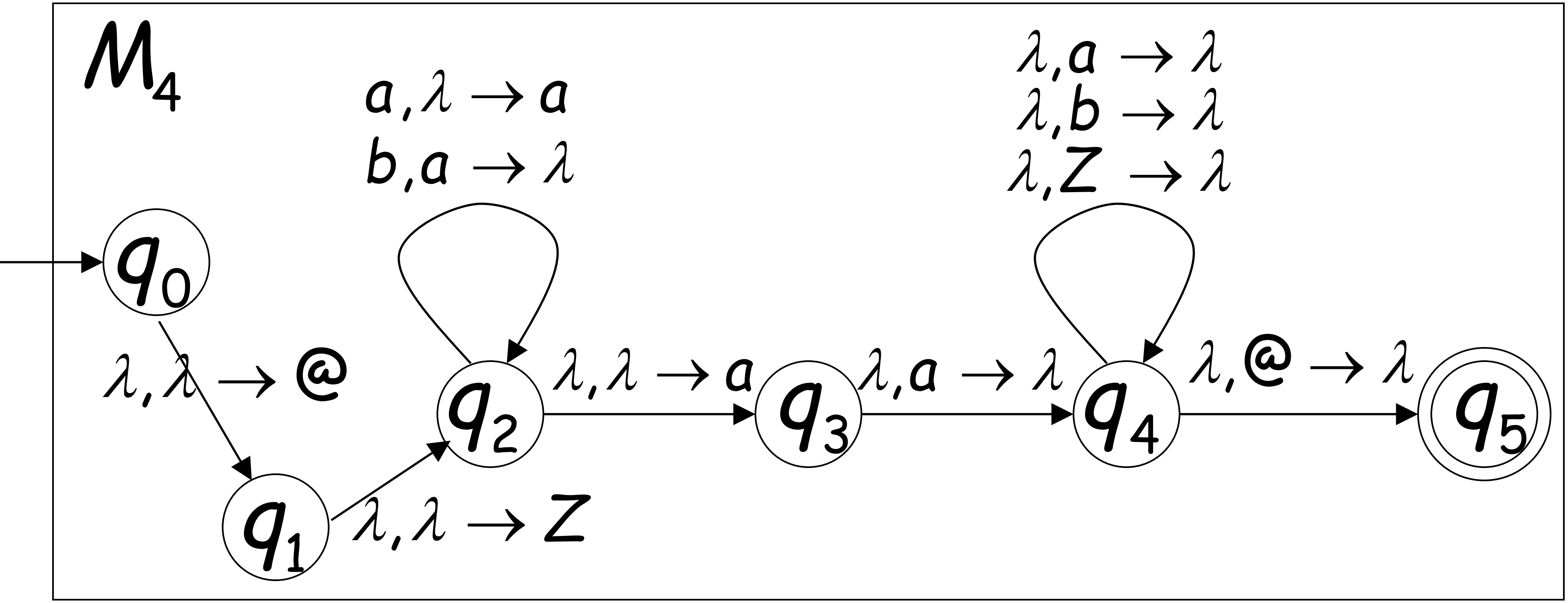
---

grammatica

Variabile Start $A_{q_0 q_f}$

Esempio:

PDA



M_4
$a, \lambda \rightarrow a$
$b, a \rightarrow \lambda$
$\lambda, a \rightarrow \lambda$
$\lambda, b \rightarrow \lambda$
$\lambda, Z \rightarrow \lambda$
q_0
$\lambda, \lambda \rightarrow @$
q_1
$\lambda, \lambda \rightarrow Z$
q_2
$\lambda, \lambda \rightarrow a$
q_3
$\lambda, a \rightarrow \lambda$
q_4
$\lambda, @ \rightarrow \lambda$
q_5

grammatica

caso 1: da stati singoli

$$A_{q_0q_0} \to \lambda$$

$$A_{q_1q_1} \to \lambda$$

$$A_{q_2q_2} \to \lambda$$

$$A_{q_3q_3} \to \lambda$$

$$A_{q_4q_4} \to \lambda$$

$$A_{q_5q_5} \to \lambda$$

caso 2: da triple di stati

$$A_{q_0q_0} \to A_{q_0q_0}A_{q_0q_0} \mid A_{q_0q_1}A_{q_1q_0} \mid A_{q_0q_2}A_{q_2q_0} \mid A_{q_0q_3}A_{q_3q_0} \mid A_{q_0q_4}A_{q_4q_0} \mid A_{q_0q_5}A_{q_5q_0}$$

$$A_{q_0q_1} \to A_{q_0q_0}A_{q_0q_1} \mid A_{q_0q_1}A_{q_1q_1} \mid A_{q_0q_2}A_{q_2q_1} \mid A_{q_0q_3}A_{q_3q_1} \mid A_{q_0q_4}A_{q_4q_1} \mid A_{q_0q_5}A_{q_5q_1}$$

$$\vdots$$

$$A_{q_0q_5} \to A_{q_0q_0}A_{q_0q_5} \mid A_{q_0q_1}A_{q_1q_5} \mid A_{q_0q_2}A_{q_2q_5} \mid A_{q_0q_3}A_{q_3q_5} \mid A_{q_0q_4}A_{q_4q_5} \mid A_{q_0q_5}A_{q_5q_5}$$

$$\vdots$$

$$A_{q_5q_5} \to A_{q_5q_0}A_{q_0q_5} \mid A_{q_5q_1}A_{q_1q_5} \mid A_{q_5q_2}A_{q_2q_5} \mid A_{q_5q_3}A_{q_3q_5} \mid A_{q_5q_4}A_{q_4q_5} \mid A_{q_5q_5}A_{q_5q_5}$$

Variabile Start $A_{q_0q_5}$

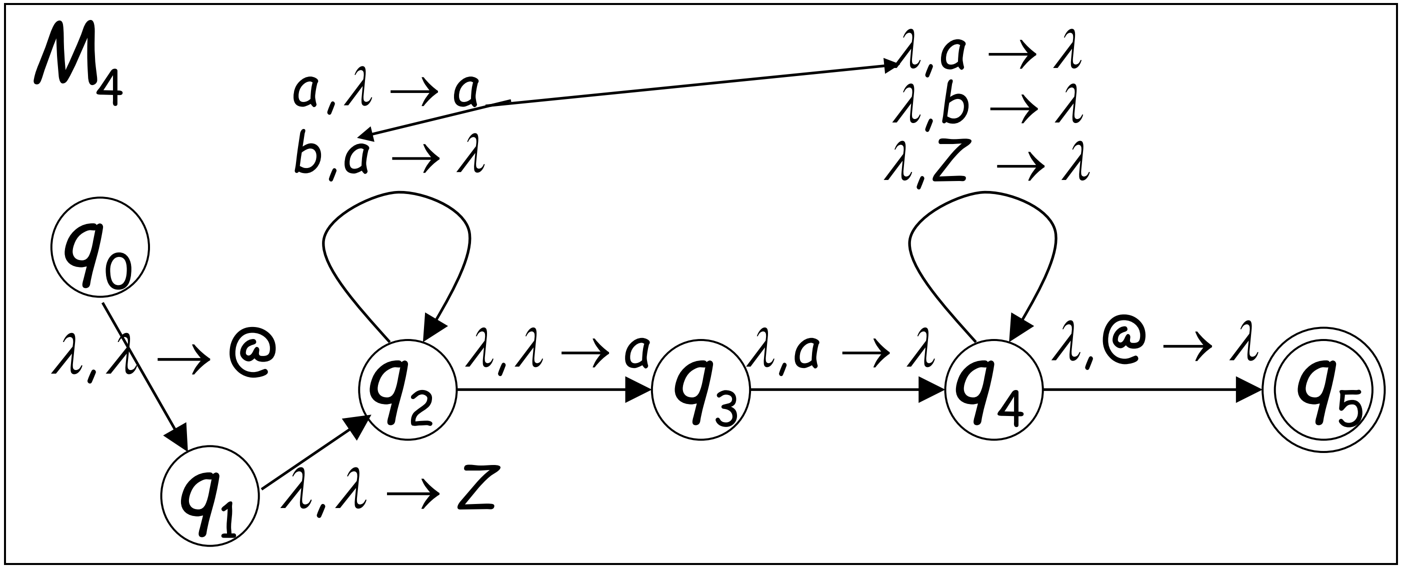
PDA

caso 3: <u>per ogni coppia di transizioni</u>

grammatica

$$A_{pq} \to a A_{rs} b$$

caso 3: da coppie di transizioni



$$A_{q_0q_5} \to A_{q_1q_4}$$

$$A_{q_1q_4} \to A_{q_2q_4}$$

$$A_{q_2q_4} \to aA_{q_2q_4}$$

$$A_{q_2q_2} \to aA_{q_2q_2}b$$

$$A_{q_2q_4} \to aA_{q_2q_3}$$

$$A_{q_2q_2} \to A_{q_3q_2}b$$

$$A_{q_2q_4} \to A_{q_3q_3}$$

$$A_{q_2q_4} \to A_{q_3q_4}$$

Supponiamo che il PDA M è stato tradotto
In una grammatica context-free G

Dobbiamo provare $L(G) = L(M)$

O in modo equivalente

$L(G) \subseteq L(M)$ $L(G) \supseteq L(M)$

$L(G) \subseteq L(M)$

Dobbiamo mostrare che se G ha una derivazione:

$$A_{q_0 q_f} \stackrel{*}{\Longrightarrow} w \qquad \text{(stringa di terminali)}$$

Allora vi è un calcolo in M che accetta w :

$$(q_0, w, @) \stackrel{*}{\succ} (q_f, \lambda, @)$$

partiamo con una p e una q qualsiasi.

Se in G vi è una derivazione:

$$A_{pq} \overset{*}{\Rightarrow} w$$

Allora vi è un calcolo in M :

$$(p, w, \lambda) \overset{*}{\succ} (q, \lambda, \lambda)$$

Dopo aver provato
il passo precedente
abbiamo:

$$A_{q_0 q_f} \overset{*}{\Rightarrow} w$$

$$\Downarrow$$

$$(q_0, w, \lambda) \overset{*}{\succ} (q_f, \lambda, \lambda)$$

$$\Downarrow$$

Poichè non c'è nessuna
transizione
Con il simbolo @

$$(q_0, w, @) \overset{*}{\succ} (q_f, \lambda, @)$$

Lemma:

se $A_{pq} \stackrel{*}{\Rightarrow} w$ (stringa di terminali)

Allora vi è un calcolo

dallo stato p allo stato q

sulla stringa w

Che lascia lo stack vuoto:

$$(p, w, \lambda) \stackrel{*}{\succ} (q, \lambda, \lambda)$$

Dim intuitiva:

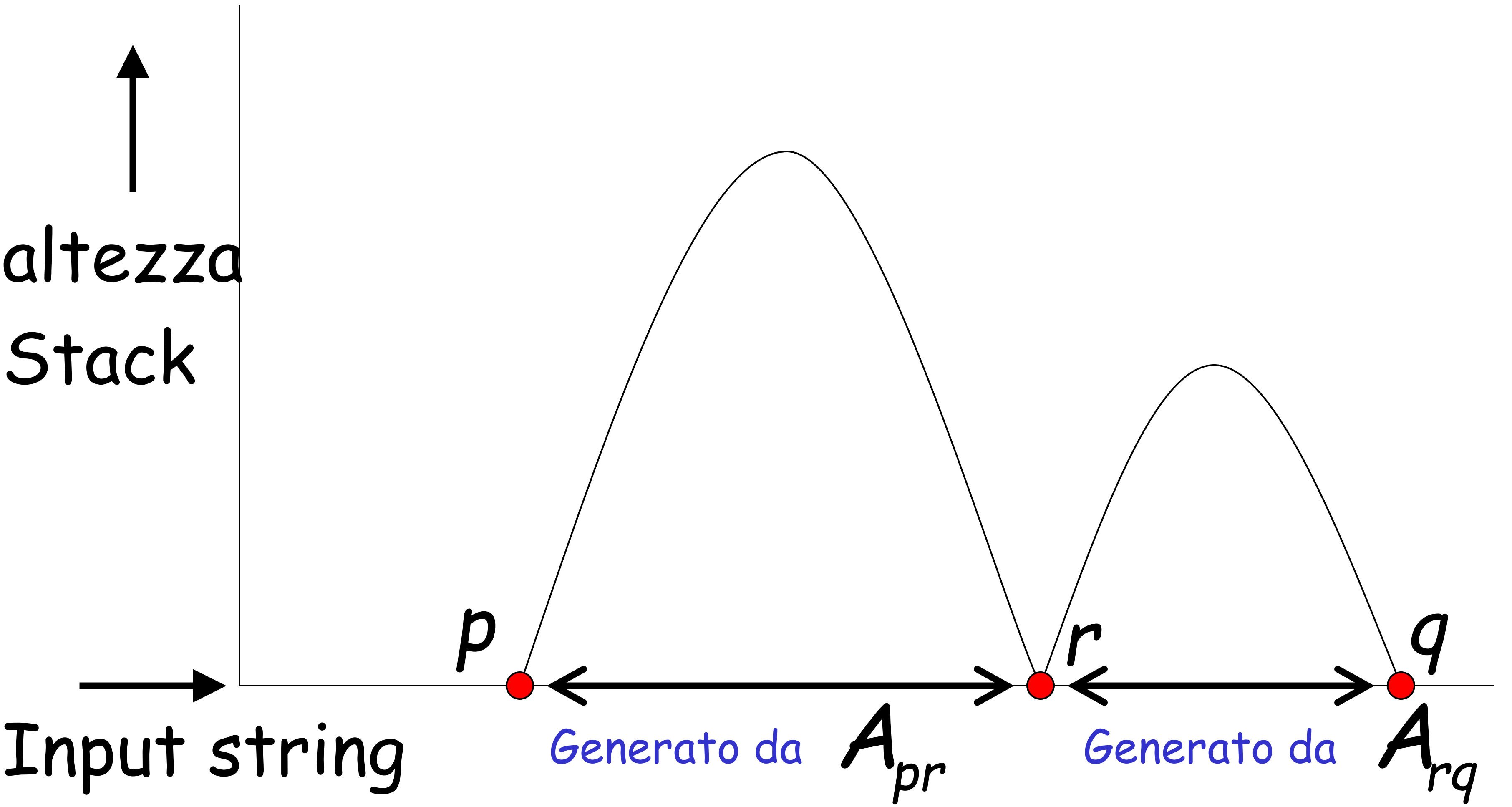
$$A_{pq} \Rightarrow \cdots \Rightarrow w$$

Case 1: $A_{pq} \Rightarrow A_{pr} A_{rq} \Rightarrow \cdots \Rightarrow w$

Case 2: $A_{pq} \Rightarrow a A_{rs} b \Rightarrow \cdots \Rightarrow w$

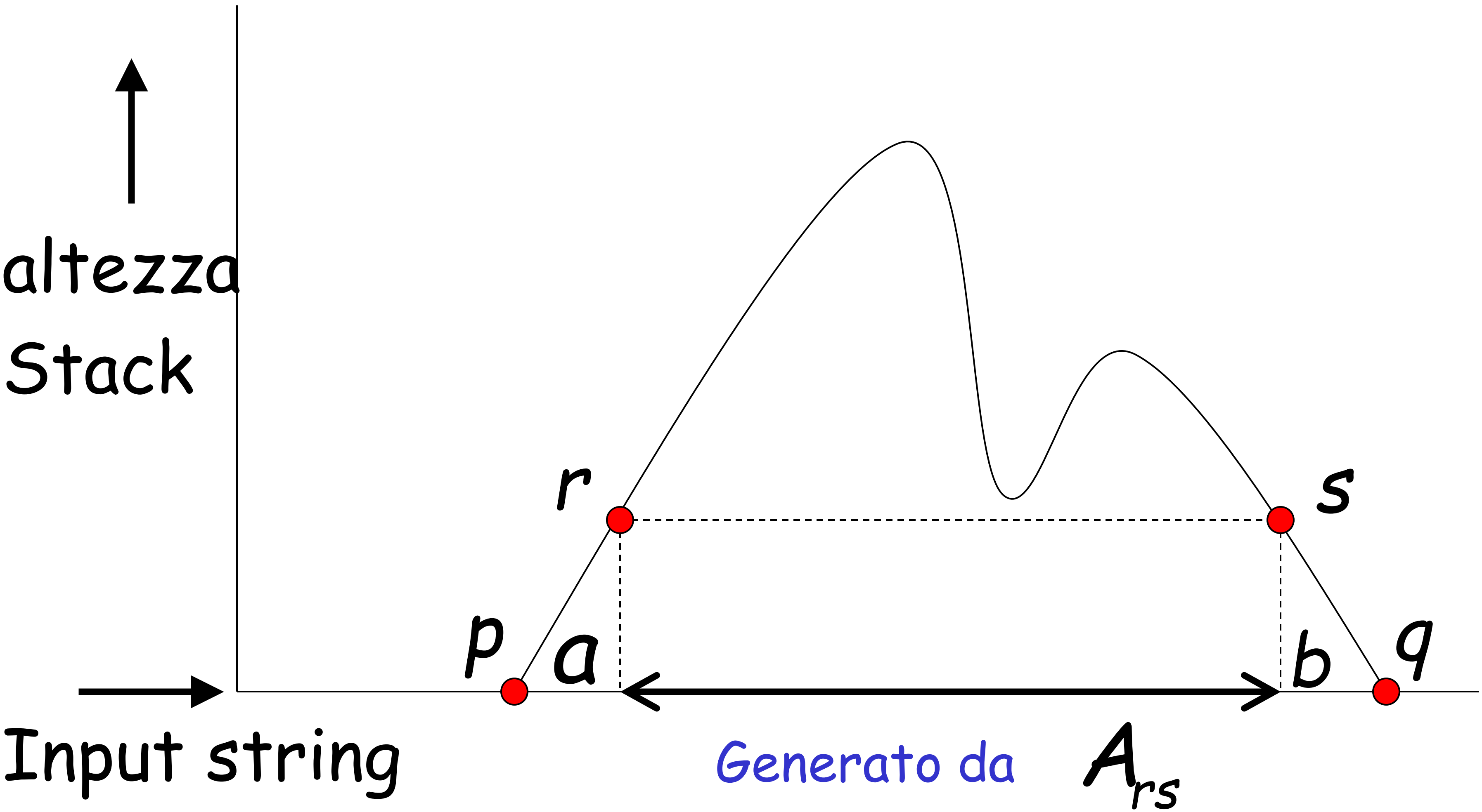
Type 2

Case 1: $A_{pq} \Rightarrow A_{pr} A_{rq} \Rightarrow \cdots \Rightarrow w$



Type 3

Case 2: $A_{pq} \Rightarrow aA_{rs}b \Rightarrow \cdots \Rightarrow w$



Formale :

Proviamo l'asserto per induzione
Sul numero di step della derivazione:

$$A_{pq} \underbrace{\Rightarrow \cdots \Rightarrow}_{\text{Numero di step}} w$$

base: $$A_{pq} \Rightarrow w$$

(Uno step di derivazione)

caso 1 produzione che deve essere usata è:

$$A_{pp} \rightarrow \lambda$$

Quindi $p = q$ e $w = \lambda$

Questo calcolo nel PDA esiste (banale):

$$(p,\lambda,\lambda) \overset{*}{\succ} (p,\lambda,\lambda)$$

Ipotesi induttiva:

$$A_{pq} \underbrace{\Rightarrow \cdots \Rightarrow}_{k} w$$

con k Step di derivazione

Quindi abbiamo, per ipotesi induttiva:

$$(p, w, \lambda) \stackrel{*}{\succ} (q, \lambda, \lambda)$$

Step induttivo

Data:

$$\underbrace{A_{pq} \Rightarrow \cdots \Rightarrow w}_{k+1}$$

con $k+1$ step di derivazione

Dobbiamo provare:

$$(p, w, \lambda) \overset{*}{\succ} (q, \lambda, \lambda)$$

$$A_{pq} \underbrace{\Rightarrow \cdots \Rightarrow}_{k+1} w$$

$k+1$ derivazione steps

Case 1: $A_{pq} \Rightarrow A_{pr}A_{rq} \Rightarrow \cdots \Rightarrow w$

Case 2: $A_{pq} \Rightarrow aA_{rs}b \Rightarrow \cdots \Rightarrow w$

Case 1: $A_{pq} \underbrace{\Rightarrow A_{pr} A_{rq} \Rightarrow \cdots \Rightarrow}_{k+1 \text{ steps}} w$

Sia: $w = yz$

$A_{pr} \underbrace{\Rightarrow \cdots \Rightarrow}_{\text{Massimo } k \text{ steps}} y$

$A_{rq} \underbrace{\Rightarrow \cdots \Rightarrow}_{\text{Massimo } k \text{ steps}} z$

$$A_{pr} \underbrace{\Rightarrow \cdots \Rightarrow}_{\text{massimo } k \text{ steps}} y$$

$$A_{rq} \underbrace{\Rightarrow \cdots \Rightarrow}_{\text{massimo } k \text{ steps}} z$$

Per ipotesi induttiva in PDA:

$$(p, y, \lambda) \overset{*}{\succ} (r, \lambda, \lambda)$$

Per ipotesi induttiva, in PDA:

$$(r, z, \lambda) \overset{*}{\succ} (q, \lambda, \lambda)$$

$$(p,y,\lambda) \overset{*}{\succ} (r,\lambda,\lambda) \qquad (r,z,\lambda) \overset{*}{\succ} (q,\lambda,\lambda)$$

$\Downarrow$

$$(p,yz,\lambda) \overset{*}{\succ} (r,z,\lambda) \overset{*}{\succ} (q,\lambda,\lambda)$$

poichè $w = yz$ $\Downarrow$

$$(p,w,\lambda) \overset{*}{\succ} (q,\lambda,\lambda)$$

Case 2: $A_{pq} \underbrace{\Rightarrow aA_{rs}b \Rightarrow \cdots \Rightarrow}_{k+1 \text{ steps}} w$

Possiamo scrivere

$$w = ayb$$

$$A_{rs} \underbrace{\Rightarrow \cdots \Rightarrow}_{\text{Massimo } k \text{ steps}} y$$

$$A_{rs} \underbrace{\Rightarrow \cdots \Rightarrow}_{k} y$$

Massimo k steps

Per ipotesi induttiva ,

il PDA calcola:

$$(r, y, \lambda) \overset{*}{\succ} (s, \lambda, \lambda)$$

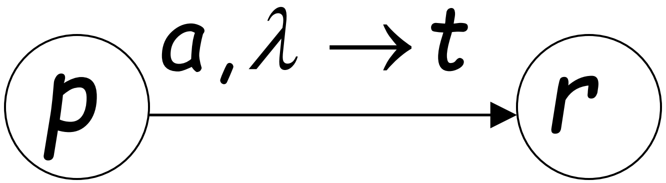
$$A_{pq} \Rightarrow aA_{rs}b \Rightarrow \cdots \Rightarrow w$$

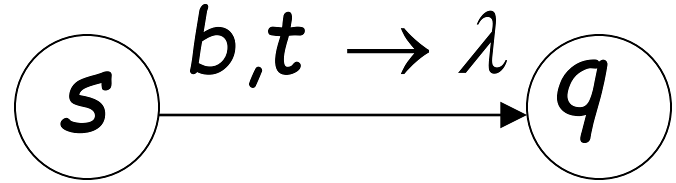
La grammatica contiene la produzione

$$A_{pq} \rightarrow aA_{rs}b$$

e il PDA contiene la transizione

$$p \xrightarrow{a,\lambda \rightarrow t} r \qquad\qquad s \xrightarrow{b,t \rightarrow \lambda} q$$





$$(p, ayb, \lambda) \succ (r, yb, t)$$

$$(s, b, t) \succ (q, \lambda, \lambda)$$

sappiamo

$$(r,y,\lambda) \overset{*}{\succ} (s,\lambda,\lambda) \quad \Rightarrow \quad (r,yb,t) \overset{*}{\succ} (s,b,t)$$

Inoltre sappiamo

$$(p,ayb,\lambda) \succ (r,yb,t)$$

$$(s,b,t) \succ (q,\lambda,\lambda)$$

---

quindi:

$$(p,ayb,\lambda) \succ (r,yb,t) \overset{*}{\succ} (s,b,t) \succ (q,\lambda,\lambda)$$

$$(p,ayb,\lambda) \succ (r,yb,t) \overset{*}{\succ} (s,b,t) \succ (q,\lambda,\lambda)$$

poichè $\quad w = ayb$

$$(p,w,\lambda) \overset{*}{\succ} (q,\lambda,\lambda)$$

Fine

Determinismo vs non Determinismo nei push down.

deterministico PDA: DPDA

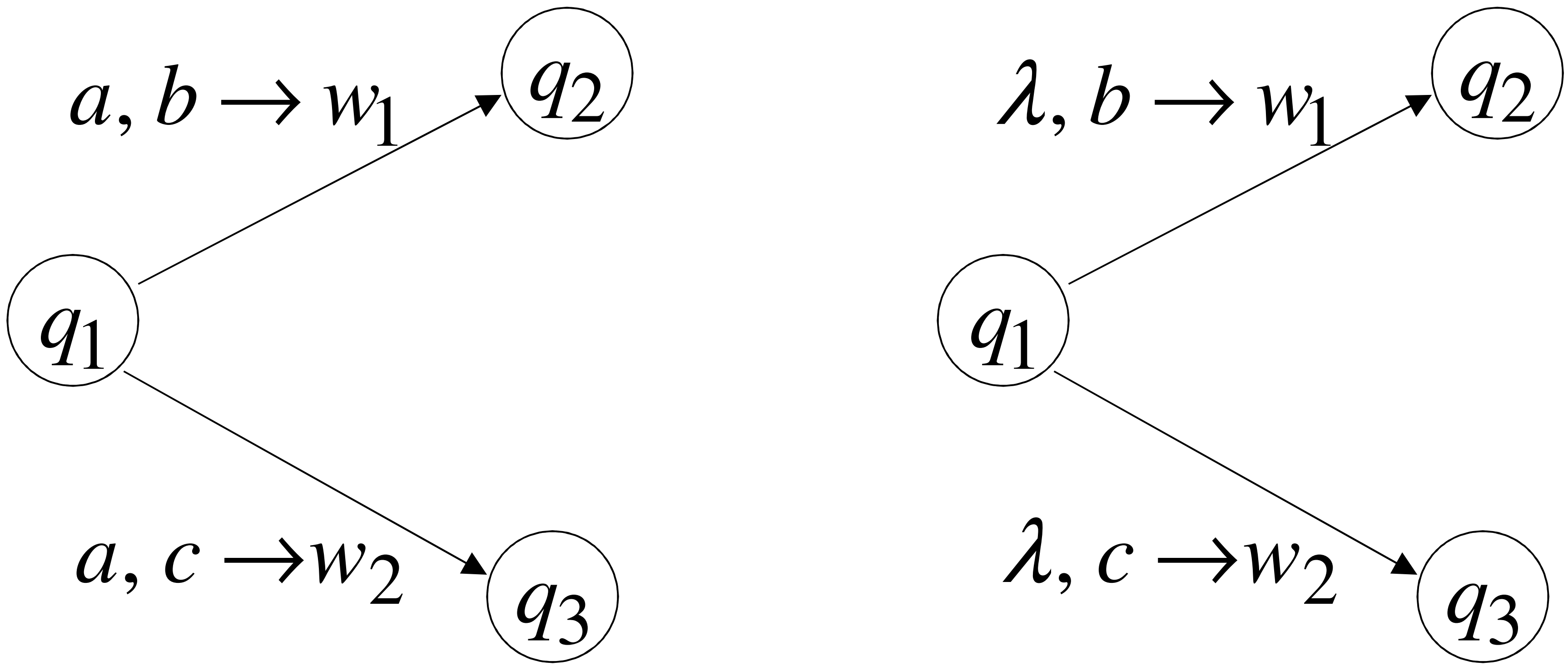
Transizioni permesse:

$$q_1 \xrightarrow{a,\, b \to w} q_2$$

$$q_1 \xrightarrow{\lambda,\, b \to w} q_2$$

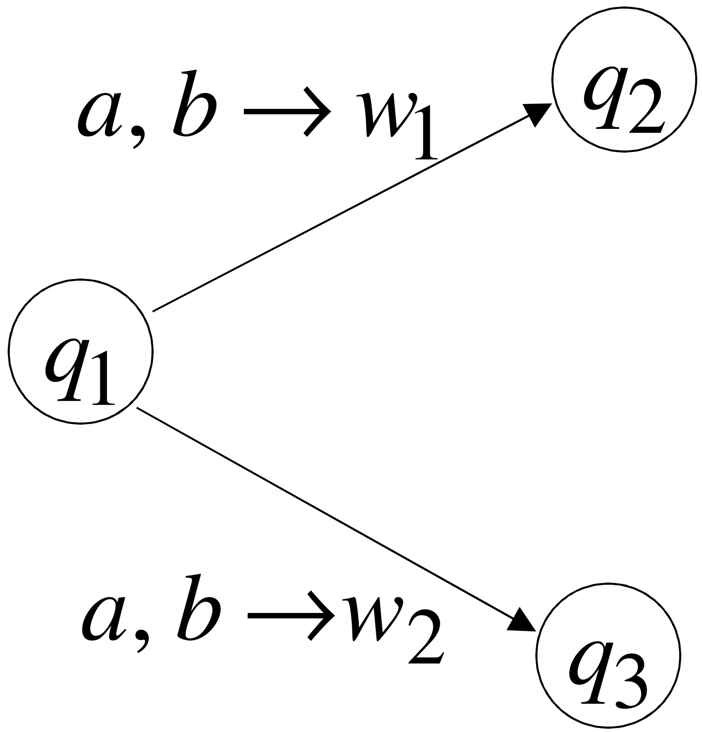
(scelte deterministiche)

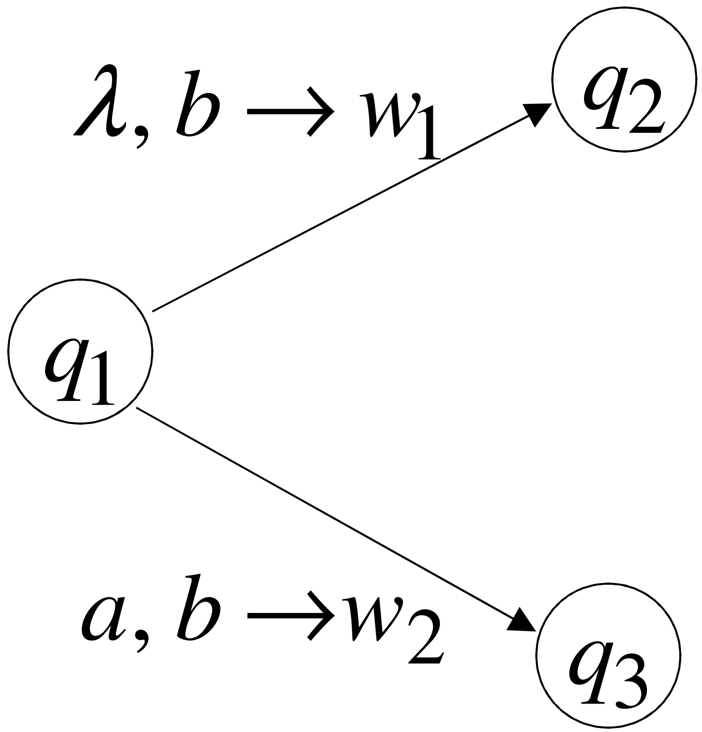
Transizioni permesse:



scelte deterministiche

Non permesse:

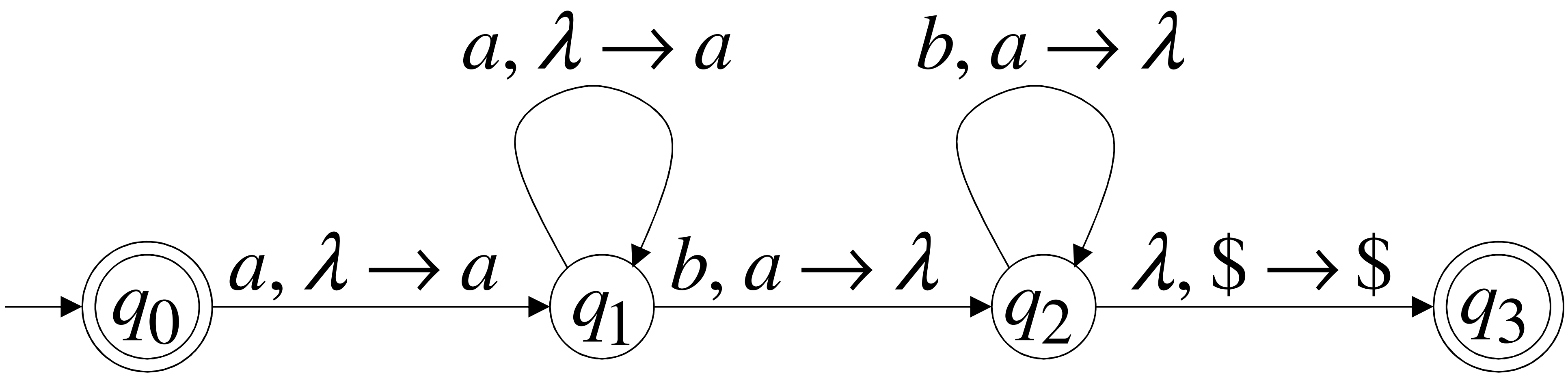




(scelte non deterministiche)

Deterministico PDA
esempio

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



Definition:

Un linguaggio L è **deterministico context-free**

Se esiste un DPDA che lo accetta

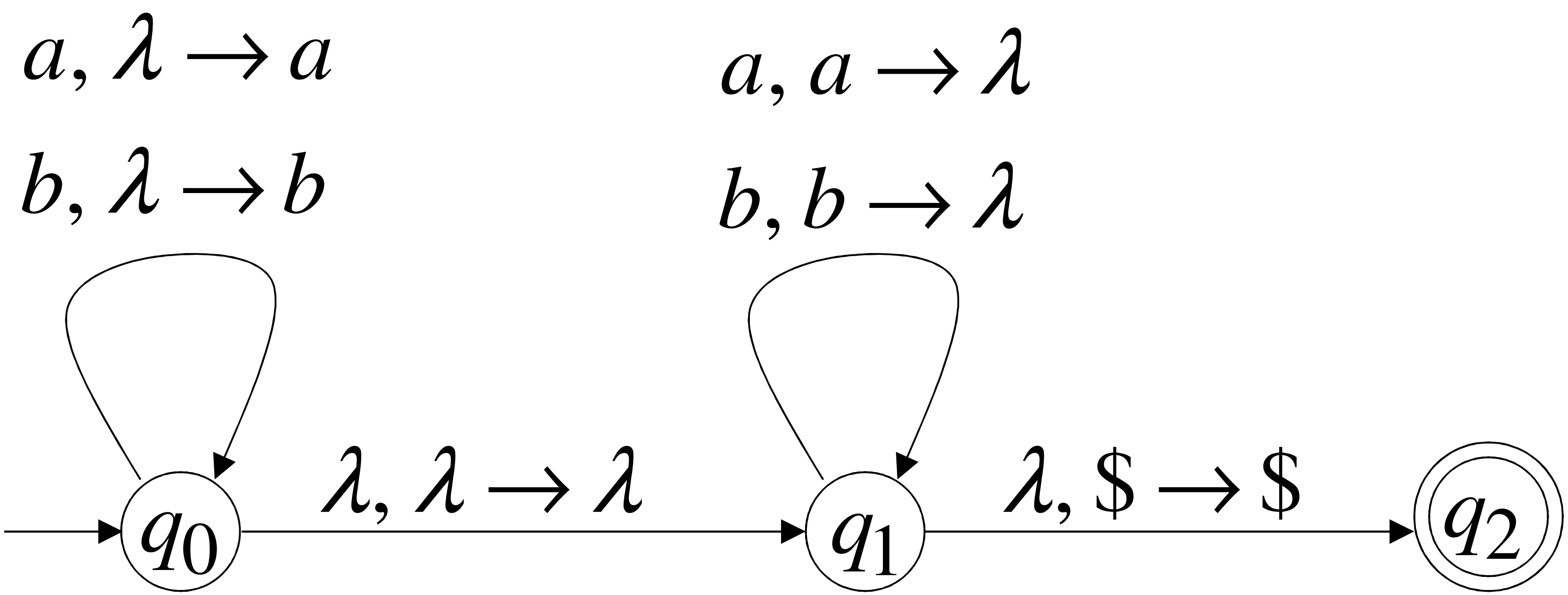
Esempio:

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

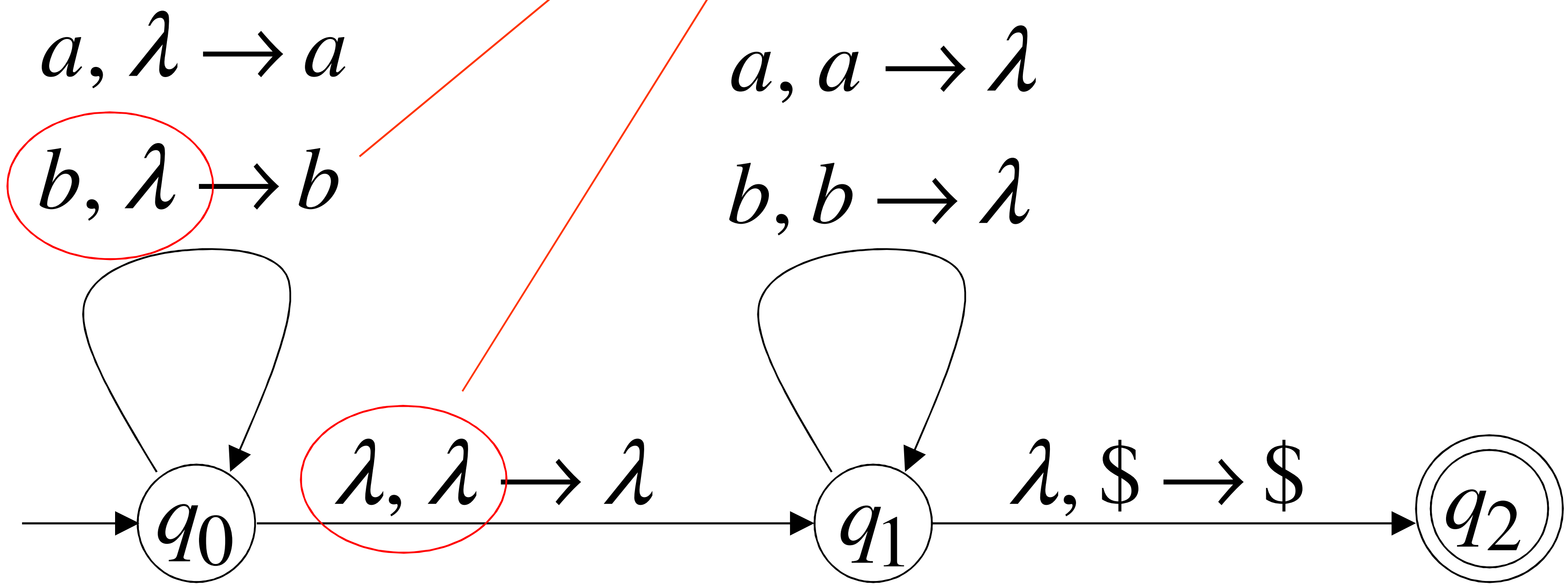
è **deterministico context-free**

Esempio di Non-DPDA (PDA)

$$L(M)=\{vv^R:\ v\in\{a,b\}^*\}$$



Non è permesso in DPDA



I PDA

hanno più potere dei

DPDA

Vale la relazione:

$$\left\{ \begin{array}{c} \text{deterministico} \\ \text{Context-Free} \\ \text{linguaggi} \\ \text{(DPDA)} \end{array} \right\} \subseteq \left\{ \begin{array}{c} \text{Context-Free} \\ \text{linguaggi} \\ \text{PDA} \end{array} \right\}$$

Poichè ogni DPDA è anche un PDA

Dimostriamo che :

$$L \notin \left\{ \begin{array}{c} \text{deterministico} \\ \text{Context-Free} \\ \text{Languages} \\ \text{(DPDA)} \end{array} \right\} \subset \left\{ \begin{array}{c} \text{Context-Free} \\ \text{Languages} \\ \text{(PDA)} \end{array} \right\} \ni L$$

Definiremo un linguaggio context-free L che non è Accettato da un DPDA

Il linguaggio è :

$$L = \{a^n b^n\} \cup \{a^n b^{2n}\} \qquad n \geq 0$$

Dobbiamo dimostrare che :

- L è context free

- L **non è** deterministico context-free

$$L = \{a^n b^n\} \cup \{a^n b^{2n}\}$$

Il linguaggio L è context-free

Grammatica Context-free per : L

$$S \to S_1 \mid S_2 \qquad \{a^n b^n\} \cup \{a^n b^{2n}\}$$

$$S_1 \to a S_1 b \mid \lambda \qquad \{a^n b^n\}$$

$$S_2 \to a S_2 bb \mid \lambda \qquad \{a^n b^{2n}\}$$

Teorema:

Il linguaggio $L = \{a^n b^n\} \cup \{a^n b^{2n}\}$

non è deterministico context-free

(**nessun** DPDA accetta L)

Dim : Assumiamo per assurdo che

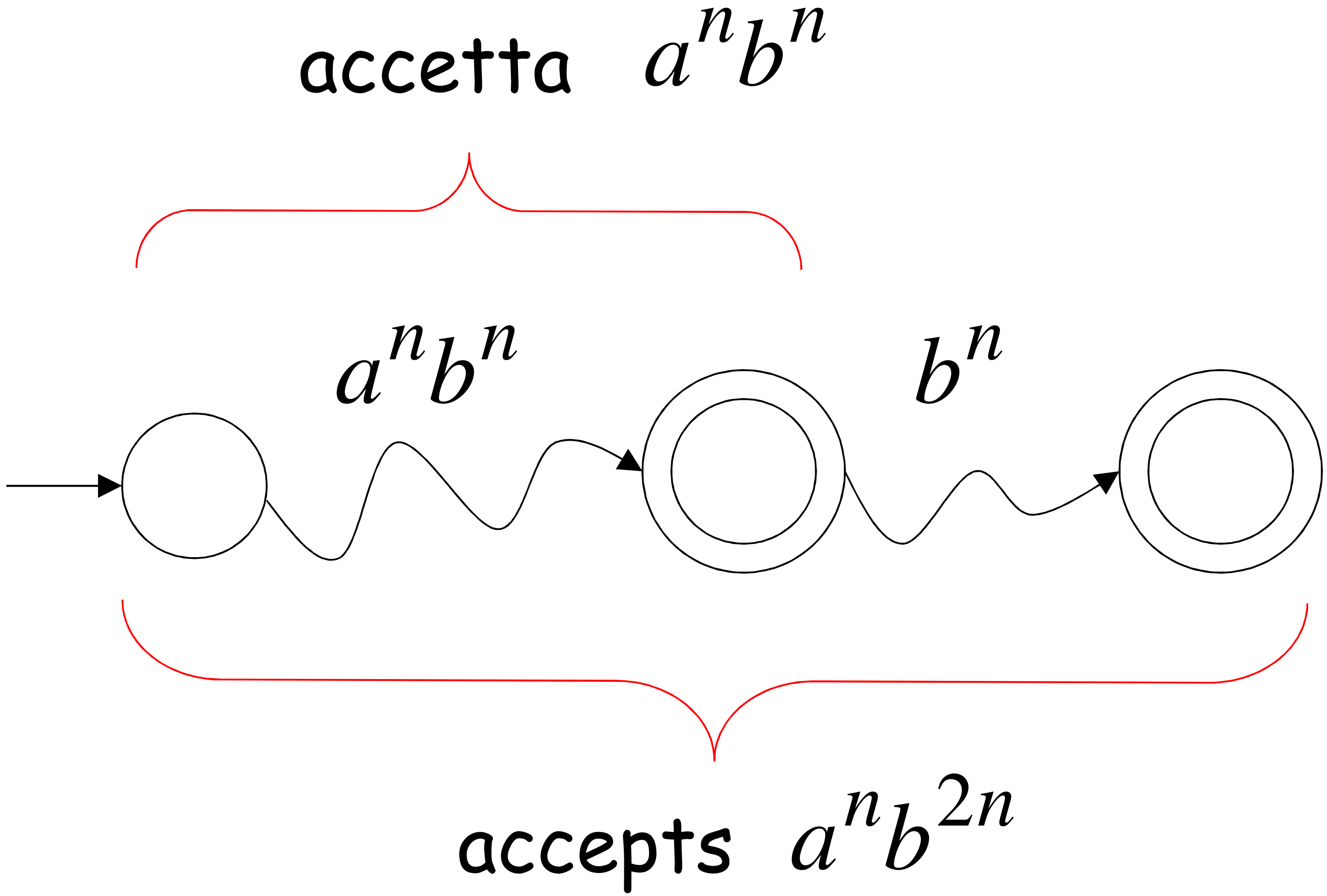
$$L = \{a^n b^n\} \cup \{a^n b^{2n}\}$$

è deterministico context free

quindi:

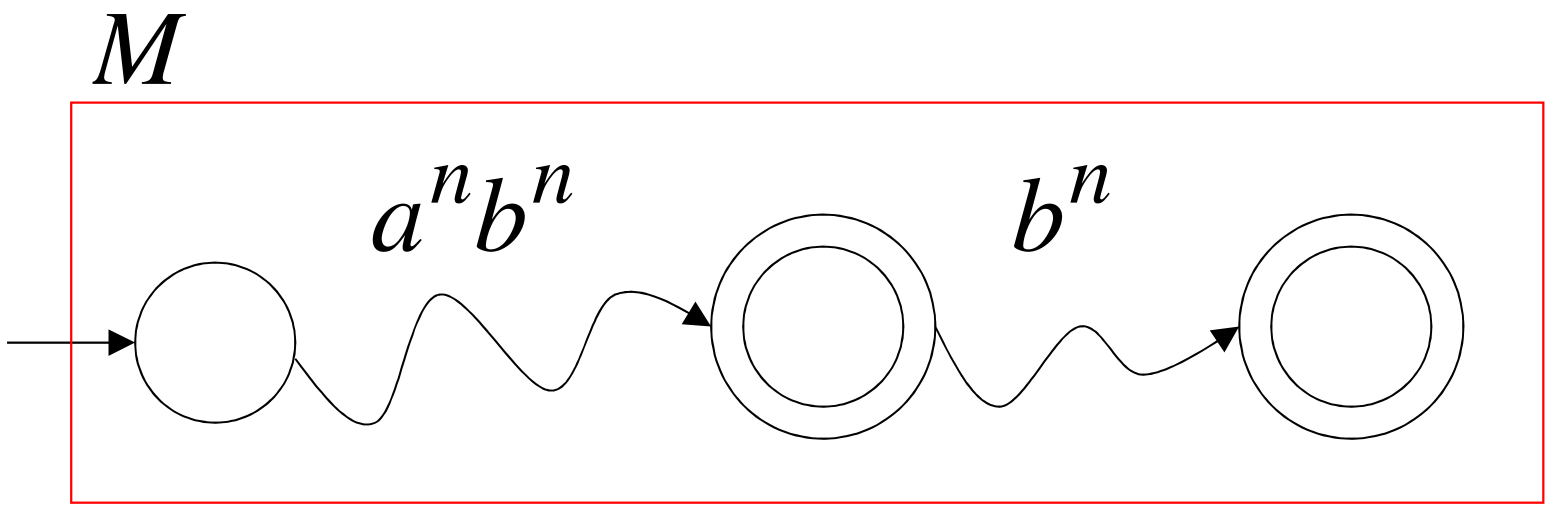
Esiste un DPDA M che accetta L

DPDA M con $L(M) = \{a^n b^n\} \cup \{a^n b^{2n}\}$



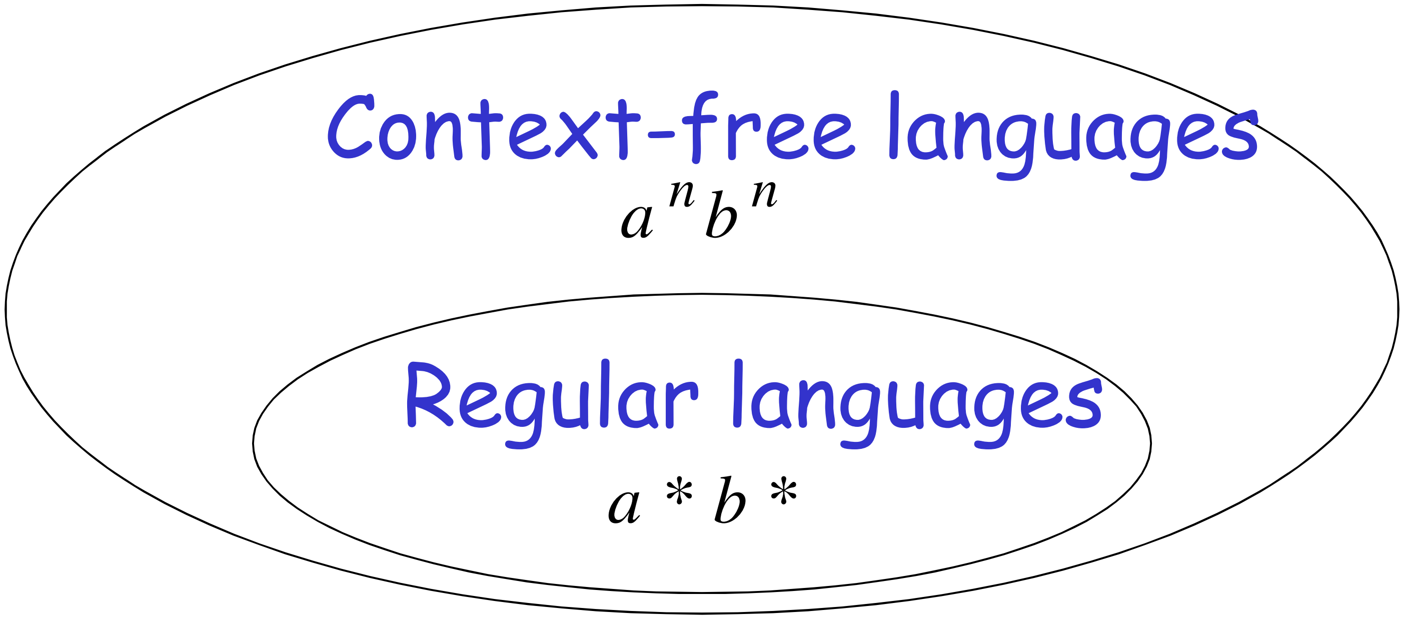
DPDA M con $L(M) = \{a^n b^n\} \cup \{a^n b^{2n}\}$

Un tale cammino
deve esistere a causa del determinismo



Fatto 1: Il linguaggio $\{a^n b^n c^n\}$

not è context-free



(si prova per pumping lemma "per i" context free)

Fatto 2: Il linguaggio $L \cup \{a^n b^n c^n\}$

non è context-free

$$(L = \{a^n b^n\} \cup \{a^n b^{2n}\})$$

(usando pumping lemma per linguaggi context-free)

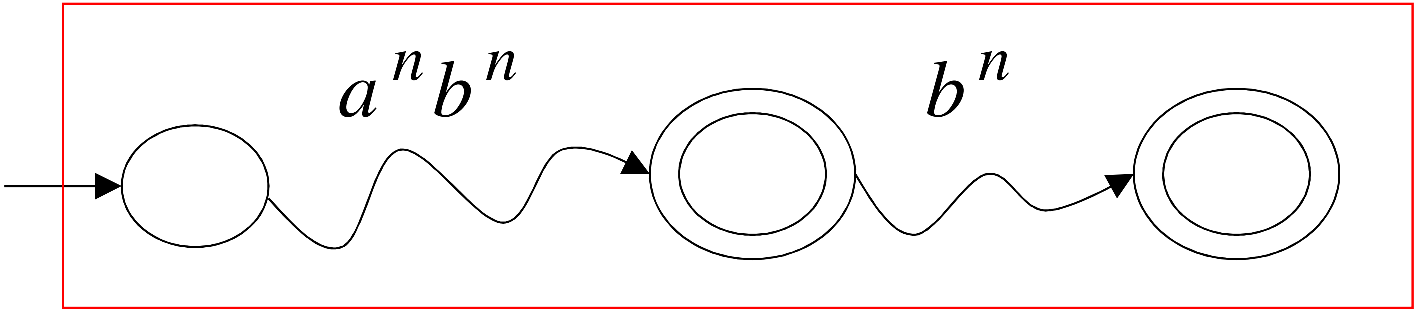
Ora costruiamo un PDA che accetta:

$$L \cup \{a^n b^n c^n\}$$

$$(L = \{a^n b^n\} \cup \{a^n b^{2n}\})$$

Che è una contraddizione !

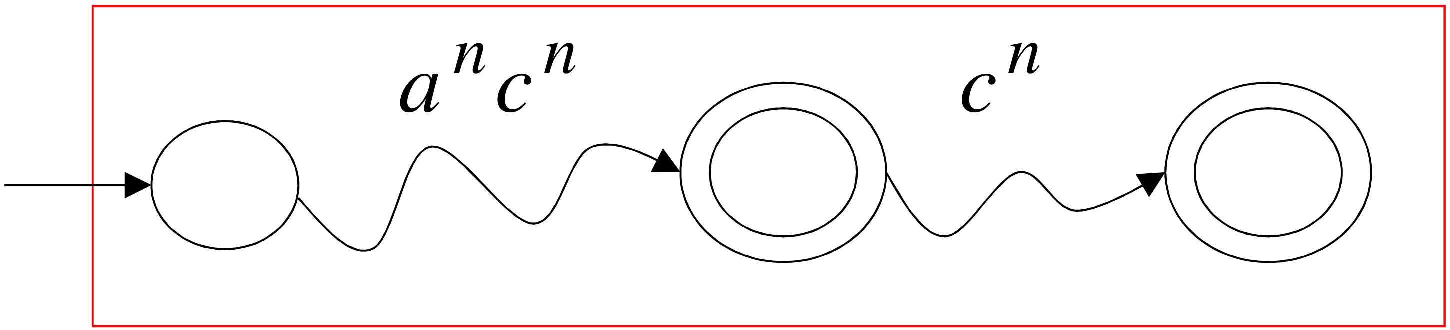
DPDA M $\qquad L(M)=\{a^n b^n\}\cup\{a^n b^{2n}\}$



modifichiamo M

Al posto di b poniamo c

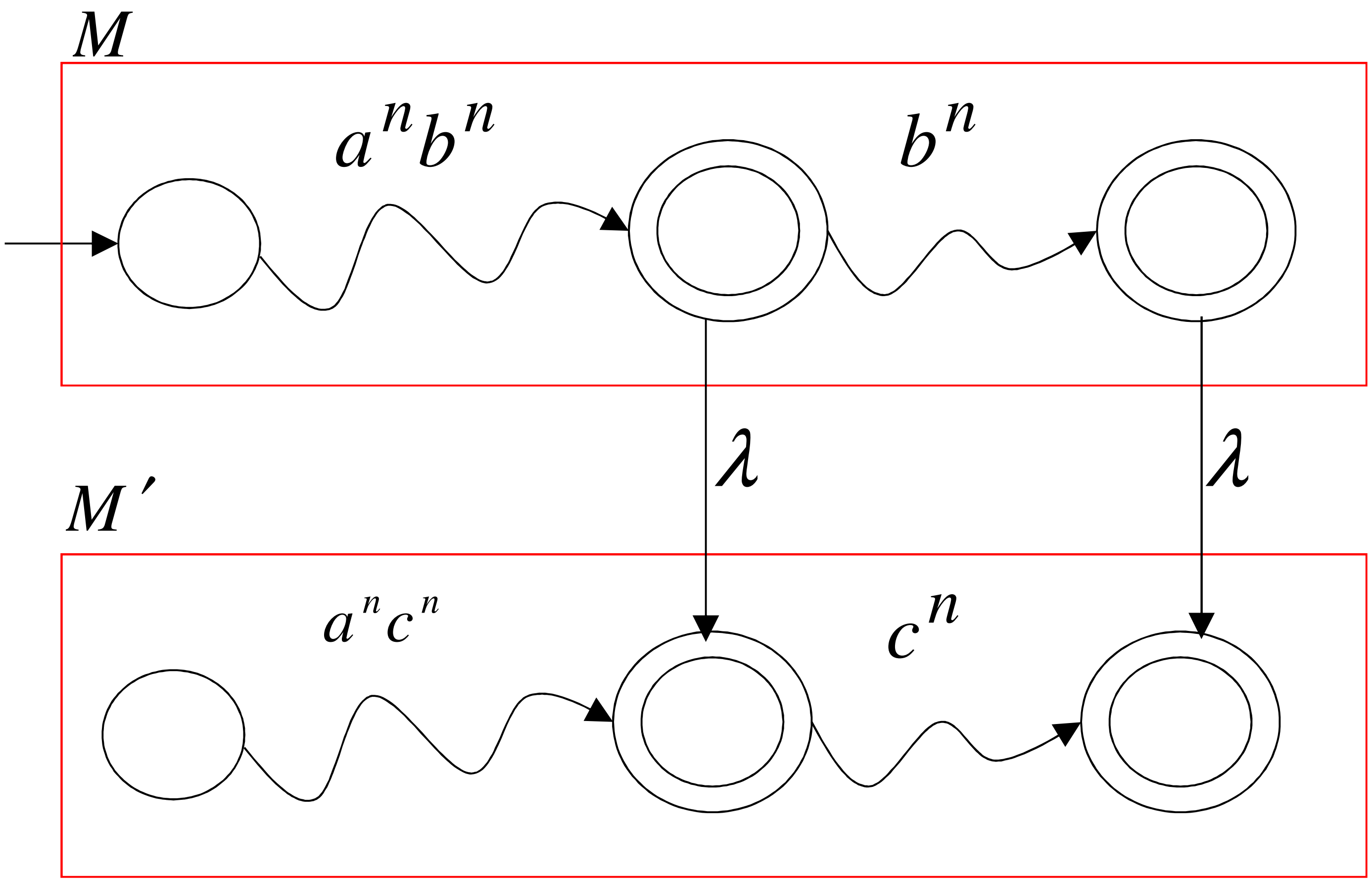
DPDA M' $\qquad L(M')=\{a^n c^n\}\cup\{a^n c^{2n}\}$



un PDA che accetta $L \cup \{a^n b^n c^n\}$

Connettiamo lo stato finale M

Con lo stato finale di M'



poichè $L \cup \{a^n b^n c^n\}$ è accettato da PDA

È context-free

Contradizione!

(poichè $L \cup \{a^n b^n c^n\}$ non è context-free) quindi:

$$L = \{a^n b^n\} \cup \{a^n b^{2n}\}$$

Non è deterministico context free

non esiste DPDA che lo accetta

Fine context free

The CYK Algorithm

- J. Cocke
- D. Younger,
- T. Kasami

The CYK Algorithm

- *Il problema dell'appartenenza*:
  - Problema:
    - Data una grammatica grammar **G** e una stringa **w**
      - **G** = (V, ∑ ,P , S) dove
        - » V insieme finito di variabili
        - » ∑ (alfabeto) insieme finito di simboli terminali-
        - » P insieme finito di produzioni
        - » S simbolo iniziale (elemento distintivo di V)
        - » V e ∑ sono insiemi disgiunti
      - **G** genera un linguaggio, **L**(**G**),
  - Domanda :
    - **w** appartiene al **L**(**G**)?

The CYK Algorithm

– La grammatica è scritta in Chomsky Normal Form
– Viene usata una tecnica chiamata “dynamic programming” o “table-filling algorithm”

Chomsky Normal Form

- *Normal Form è descritta da un insieme di condizioni che ogni regola della grammatica deve soddisfare.*
- Context-free grammar è in CNF, ogni regola ha la seguente forma:
  - $A \rightarrow BC$ al massimo due simboli sul lato destro
  - $A \rightarrow a$ a simbolo terminale
  - $S \rightarrow \lambda$ stringa vuota

  Dove $B, C \in V - \{S\}$

Costruire una Triangular Table

$X_{5,1}$				
$X_{4,1}$	$X_{4,2}$			
$X_{3,1}$	$X_{3,2}$	$X_{3,3}$		
$X_{2,1}$	$X_{2,2}$	$X_{2,3}$	$X_{2,4}$	
$X_{1,1}$	$X_{1,2}$	$X_{1,3}$	$X_{1,4}$	$X_{1,5}$
w_1	w_2	w_3	w_4	w_5

Tavola per una stringa ‘**w**’ che ha lunghezza 5

Esempio CYK Algorithm

- Prendiamo la seguente grammatica:
  - CNF grammatica **G**
    - S → AB | BC
    - A → BA | a
    - B → CC | b
    - C → AB | a
  - **w** sia baaba
  - E' **baaba** in L(G)?

Constructing The Triangular Table

S → AB | BC
A → BA | a
B → CC | b
C → AB | a

{B}	**{A, C}**	**{A, C}**	**{B}**	**{A, C}**
b	**a**	**a**	**b**	**a**

Calcolare la riga più bassa

Costruire la Triangular Table

- $\mathbf{X_{2,1} = (X_{1,1}\ ,X_{1,2})}$
- ➔ {B}{A,C} = {BA, BC}
- Step:
  - Trovare, se esistono, le regole che producono BA or BC
  - Sono due : S e A
  - $\mathbf{X_{2,1} = \{S, A\}}$

S → AB | BC
A → BA | a
B → CC | b
C → AB | a

{S, A}				
{B}	**{A, C}**	**{A, C}**	**{B}**	**{A, C}**
b	**a**	**a**	**b**	**a**

Constructing The Triangular Table

- $\mathbf{X_{2\,,\,2} = (X_{1\,,\,2}\ \ , X_{1\,,\,3})}$
- ➔ {A, C}{A,C} = {AA, AC, CA, CC} = Y
- Step:
  - Trovare, se esistono, le regole che producono Y
  - Esiste una : B
  - $\mathbf{X_{2\,,\,2} = \{B\}}$

S → AB | BC
A → BA | a
B → CC | b
C → AB | a

{S, A}	**{B}**			
{B}	**{A, C}**	**{A, C}**	**{B}**	**{A, C}**
b	**a**	**a**	**b**	**a**

- **$X_{2,3} = (X_{1,3}\ , X_{1,4})$**
- ➔ {A, C}{B} = {AB, CB} = Y
- Steps:
  - Trovare, se esistono, le regole che producono Y
  - sono: S e C
  - **$X_{2,3} = \{S, C\}$**

S → AB | BC
A → BA | a
B → CC | b
C → AB | a

{S, A}	**{B}**	**{S, C}**		
{B}	**{A, C}**	**{A, C}**	**{B}**	**{A, C}**
b	**a**	**a**	**b**	**a**

- $\mathbf{X_{2,4} = (X_{1,4}\ , X_{1,5})}$
- ➔ {B}{A, C} = {BA, BC} = Y
- Steps:
  - Trovare, se esistono, le regole che producono Y
  - Cono: S and A
  - $\mathbf{X_{2,4} = \{S, A\}}$

S → AB | BC
A → BA | a
B → CC | b
C → AB | a

Constructing The Triangular Table

{S, A}	**{B}**	**{S, C}**	**{S, A}**	
{B}	**{A, C}**	**{A, C}**	**{B}**	**{A, C}**
b	**a**	**a**	**b**	**a**

- $\mathbf{X_{3,1} = (X_{1,1}, X_{2,2}), (X_{2,1}, X_{1,3})}$
- ➔ {B}{B} **U** {S, A}{A, C}= {BB, SA, SC, AA, AC} = Y
- Steps:
  - Trovare, se esistono, le regole che producono Y
  - Nessuna
  - $\mathbf{X_{3,1} = \emptyset}$
  - **Nessun elemento in questo insieme**

S → AB | BC
A → BA | a
B → CC | b
C → AB | a

Ø				
{S, A}	{B}	{S, C}	{S, A}	
{B}	{A, C}	{A, C}	{B}	{A, C}
b	a	a	b	a

- $\mathbf{X_{3,2} = (X_{1,2}\ , X_{2,3})\ , (X_{2,2}\ , X_{1,3})}$
- ➔ {A, C}{S, C} **U** {B}{B}= {AS, AC, CS, CC, BB} = Y
- Step:
  - Trovare, se esistono, le regole che producono Y
  - una: B
  - $\mathbf{X_{2,4} = \{B\}}$

S → AB | BC
A → BA | a
B → CC | b
C → AB | a

Ø	{B}			
{S, A}	{B}	{S, C}	{S, A}	
{B}	{A, C}	{A, C}	{B}	{A, C}
b	a	a	b	a

- $\mathbf{X_{3,3} \quad = (X_{1,3}\ , X_{2,4})\ , (X_{2,3}\ , X_{1,5})}$
- ➔ {A,C}{S,A} **U** {S,C}{A,C}

  = {AS, AA, CS, CA, SA, SC, CA, CC} = Y
- Step:
  – Trovare, se esistono, le regole che producono Y
  – una: B
  – $\mathbf{X_{3,5} = \{B\}}$

S → AB | BC
A → BA | a
B → CC | b
C → AB | a

Ø	{B}	{B}		
{S, A}	{B}	{S, C}	{S, A}	
{B}	{A, C}	{A, C}	{B}	{A, C}
b	a	a	b	a

- $\mathbf{X_{4,1} = (X_{1,1}, X_{3,2}), (X_{2,1}, X_{2,3})}$
- $\mathbf{(X_{3,1}, X_{1,4})}$
- Step:
  - Trovare, se esistono, le regole che producono Y
  - una: B
  - $\mathbf{X_{4,1} = \{?\}}$

S → AB | BC
A → BA | a
B → CC | b
C → AB | a

- $X_{4,2} = (X_{1,2}, X_{3,3}), (X_{2,2}, X_{2,4})$
- $(X_{3,2}, X_{1,5})$
- Step:
  - Trovare, se esistono, le regole che producono Y
  - una: B
  - $X_{4,1} = \{?\}$

S → AB | BC
A → BA | a
B → CC | b
C → AB | a

Finale Triangular Table

{S, A, C}	**← $X_{5, 1}$**			
Ø	**{S, A, C}**			
Ø	**{B}**	**{B}**		
{S, A}	**{B}**	**{S, C}**	**{S, A}**	
{B}	**{A, C}**	**{A, C}**	**{B}**	**{A, C}**
b	**a**	**a**	**b**	**a**

- Tavola per la stringa ‘**w**’ ha lunghezza 5
- The algorithm popola la triangular table

domanda

- sia **G la grammatical CNF**
  - S → AB | BC
  - A → BA | a
  - B → CC | b
  - C → AB | a
- **w** is ababa
- Domanda: **ababa** è in L(G)?

- E’ baaba in L(G)?

Si

Possiamo vedere che S è nell’insieme X_{1n} dove ‘n’= 5

la cella X_{51} = (S, A, C) allora

S Є X_{15} allora baaba Є L(G)

Construire una Triangular Table

- Ogni riga corrisponde a una lunghezza delle sottostringhe.
  - La riga più in basso – Stringhe di lng 1
  - Seconda riga – Stringhe di lng 2

    .

    .
  - Riga più in alto – la stringa ‘w’

Costruire una Triangular Table

- $\mathbf{X_{i,\,i}}$ è l'insieme delle variabili tale che A è A → w_i una produzione di G
- Comparare al massimo n coppie di insieme calcolati in precedenza

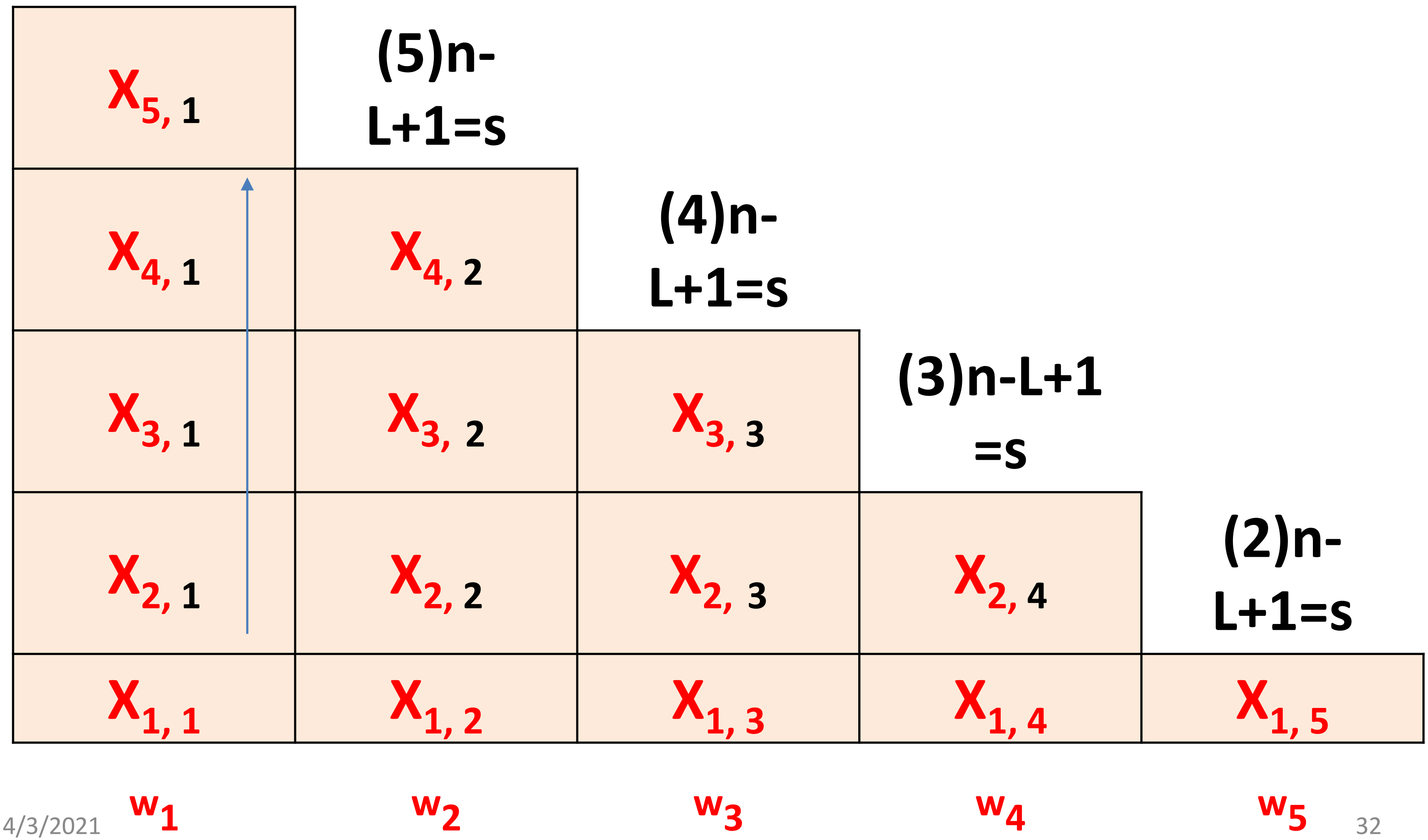
$$(\mathbf{X_{i,\,i}}, \mathbf{X_{i+1,\,j}}), (\mathbf{X_{i,\,i+1}}, \mathbf{X_{i+2,\,j}}) \ldots (\mathbf{X_{i,\,j-1}}, \mathbf{X_{j,\,j}})$$

teorema

- The CYK Algorithm calcola correttamente $X_{i\,j}$ per tutti i e j; allora w è in L(G) iff S è in X_{1n}.
- Perchè? Spiegazione, esercizio scrivere la dimostrazione.
- Complessità $O(n^3)$.

- **let** the input be a string L consisting of n characters: a_1 ... a_n.
- **let** the grammar contain r nonterminal symbols R_1 ... R_r , with start symbol R_1.
- **let** $P[n,n,r]$ be an array of booleans. Initialize all elements of P to false.
- **for each** s = 1 to n
- **for each** unit production R_v -> a_s
- set $P[1,s,v]$ = true ; Generata la prima riga-
- **for each** L = 2 to n
- **for each** s = 1 to n-L+1
- **for each** p = 1 to L-1
- **for each** production R_a -> R_b R_c
- **if** $P[p,s,b]$ and $P[L\text{-}p,s+p,c]$ **then** set $P[L,s,a]$ = true
- **if** $P[n,1,1]$ is true **then** L is member of language
- **else** L is not member of language

{S, A, C}	← $X_{5, 1}$			
Ø	{S, A, C}			
Ø	{B}	{B}		
{S, A}	{B}	{S, C}	{S, A}	
{B}	{A, C}	{A, C}	{B}	{A, C}
b	a	a	b	a



$X_{5,1}$
(5)n-L+1=s
$X_{4,1}$
$X_{4,2}$
(4)n-L+1=s
$X_{3,1}$
$X_{3,2}$
$X_{3,3}$
(3)n-L+1 =s
$X_{2,1}$
$X_{2,2}$
$X_{2,3}$
$X_{2,4}$
(2)n-L+1=s
$X_{1,1}$
$X_{1,2}$
$X_{1,3}$
$X_{1,4}$
$X_{1,5}$
w_1
w_2
w_3
w_4
w_5

for each L = 2 to n
  for each s = 1 to n-L+1
    for each p = 1 to L-1
      for each production R_a -> R_b R_c
        if $P[p,s,b]$ and $P[L-p,s+p,c]$ **then** set $P[L,s,a]$ = true

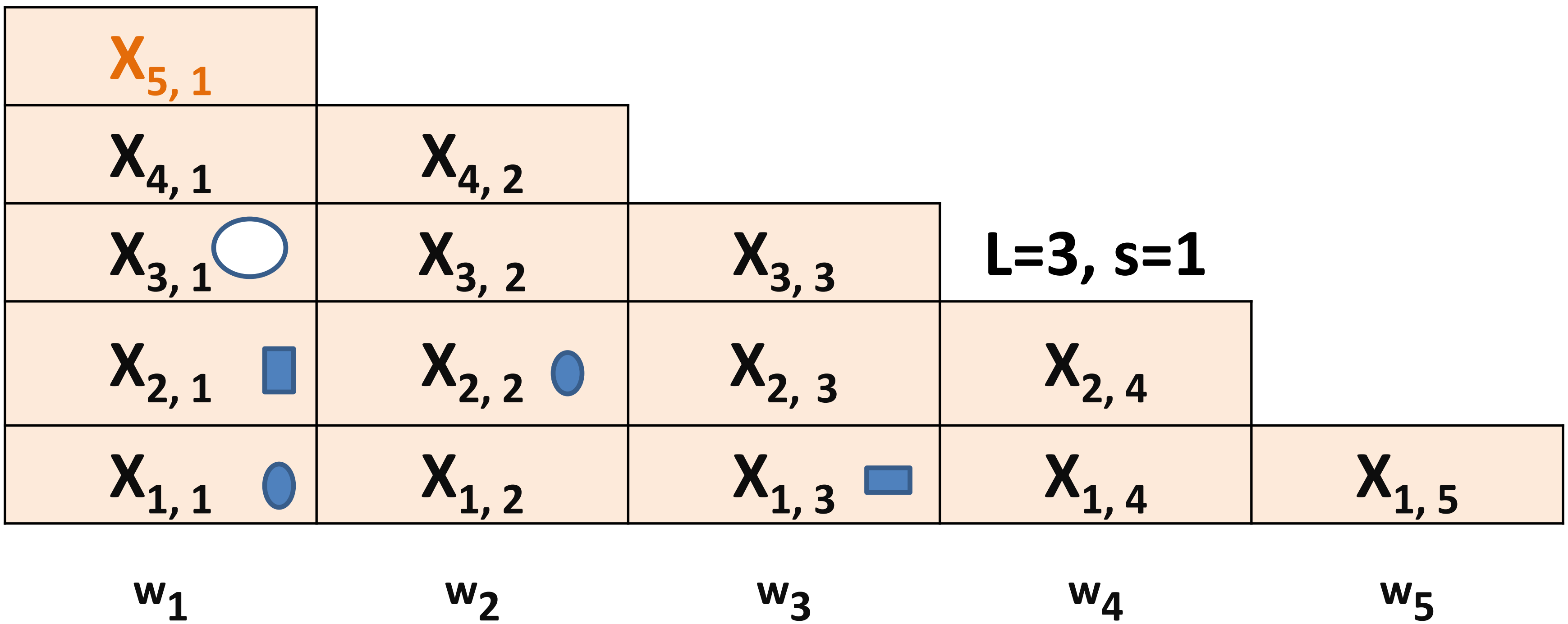


Tavola per una stringa ‘**w**’ che ha lunghezza 5

for each $L = 2$ **to** n
    for each $s = 1$ **to** $n-L+1$
        for each $p = 1$ **to** $L-1$
            for each production $R_a \rightarrow R_b\ R_c$
                if $P[p,s,b]$ and $P[L-p,s+p,c]$ **then** set $P[L,s,a]$ = true

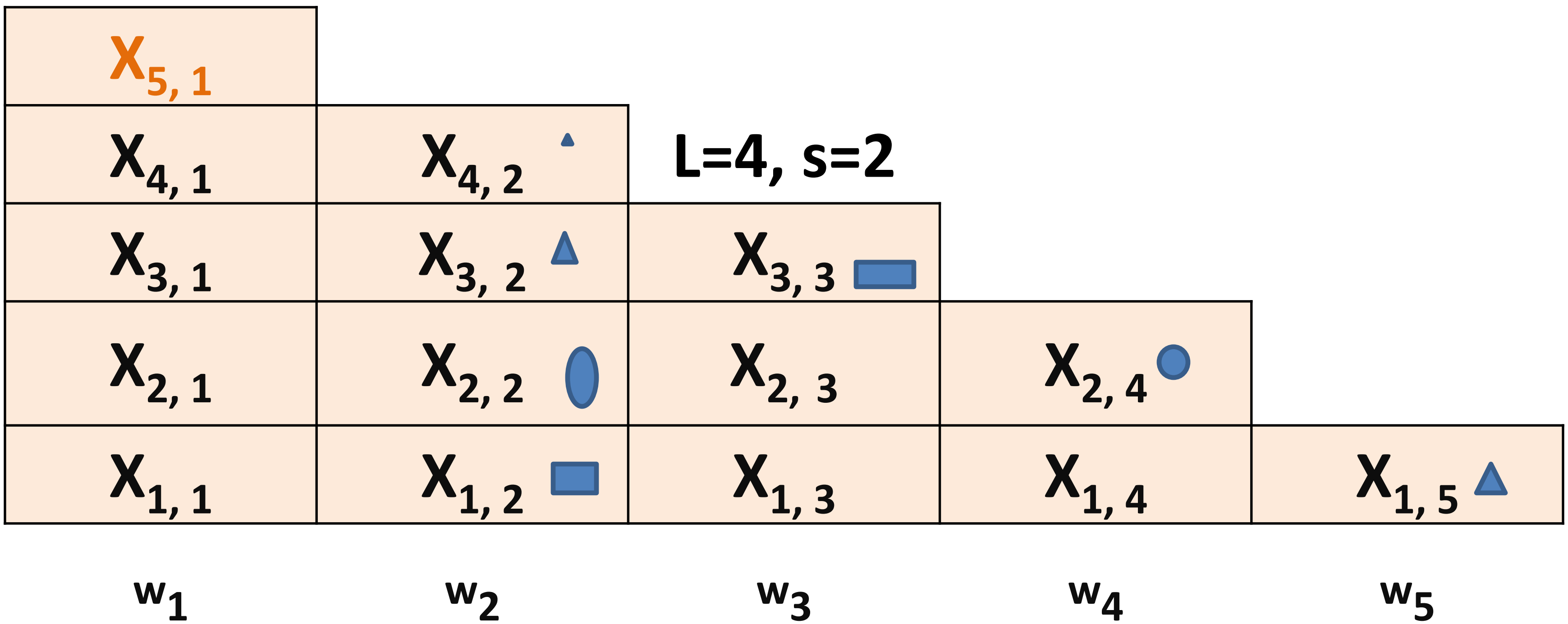


Tavola per una stringa ‘**w**’ che ha lunghezza 5

for each L **= 2 to** n
for each s **= 1 to** n**-**L**+1**
for each p **= 1 to** L**-1**
for each production $R_a \rightarrow R_b\ R_c$
if $P[p,s,b]$ and $P[L-p,s+p,c]$ **then** set $P[L,s,a]$ = true

$X_{5,1}$				
$X_{4,1}$	$X_{4,2}$			
$X_{3,1}$	$X_{3,2}$	$X_{3,3}$		
$X_{2,1}$	$X_{2,2}$	$X_{2,3}$	$X_{2,4}$	
$X_{1,1}$	$X_{1,2}$	$X_{1,3}$	$X_{1,4}$	$X_{1,5}$
w_1	w_2	w_3	w_4	w_5

Tavola per una stringa ‘**w**’ che ha lunghezza 5