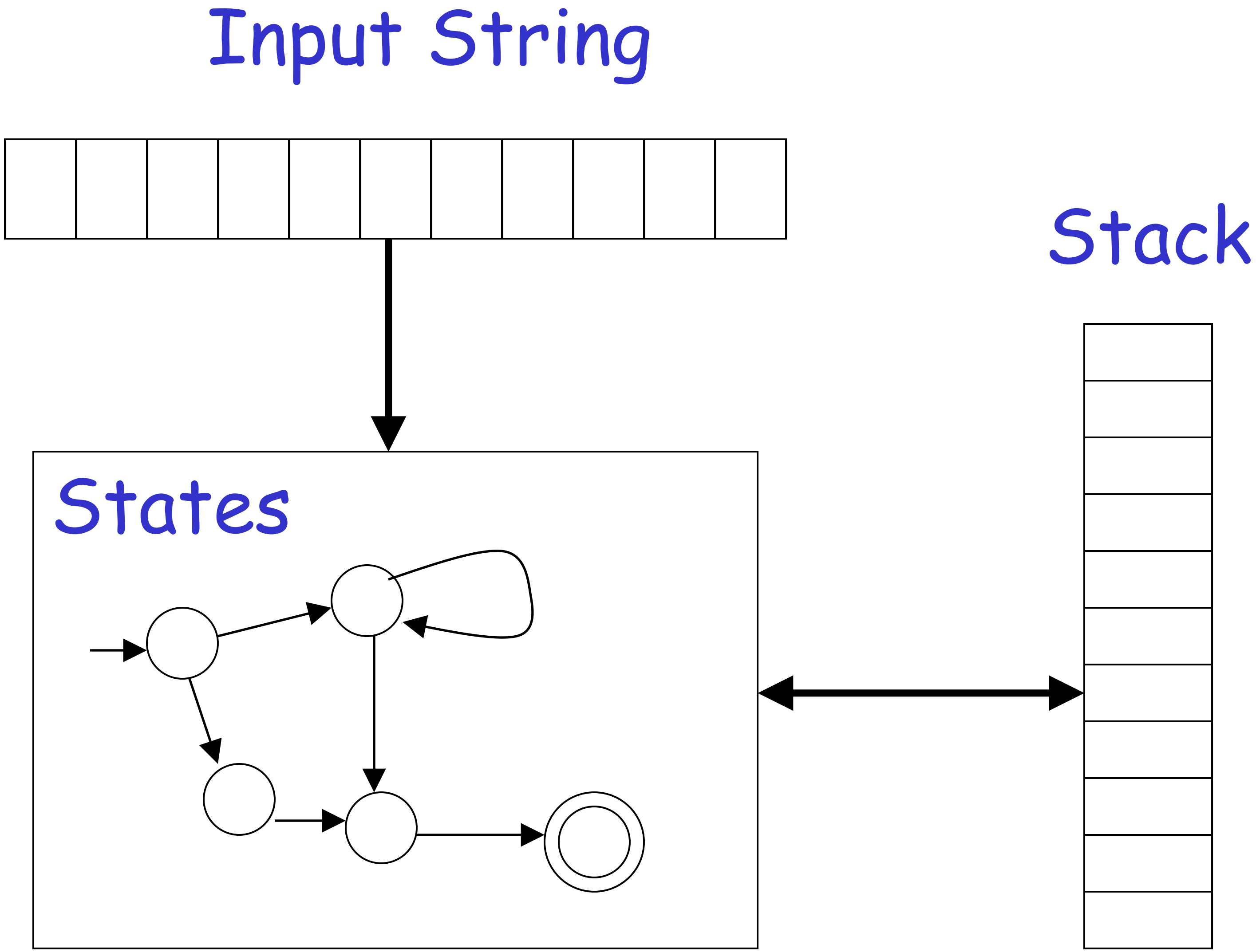
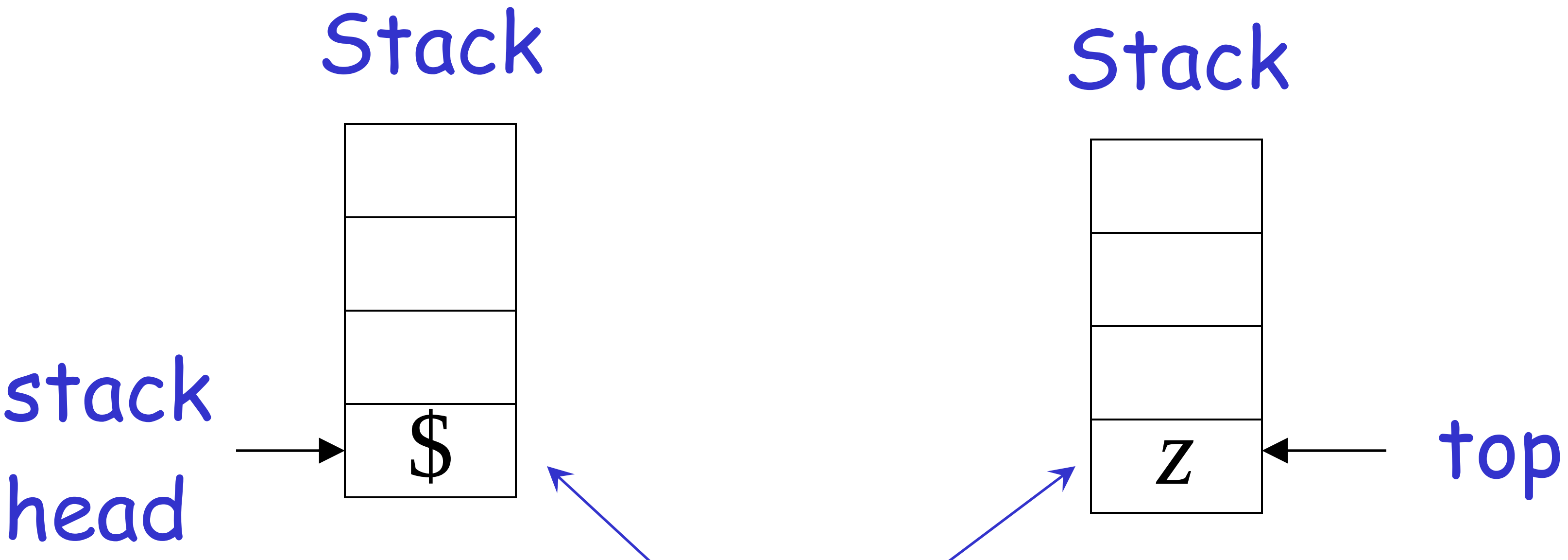
Pushdown Automata
PDAs

Pushdown Automaton -- PDA



Initial Stack Symbol



Stack

Stack

stack head

$\$$

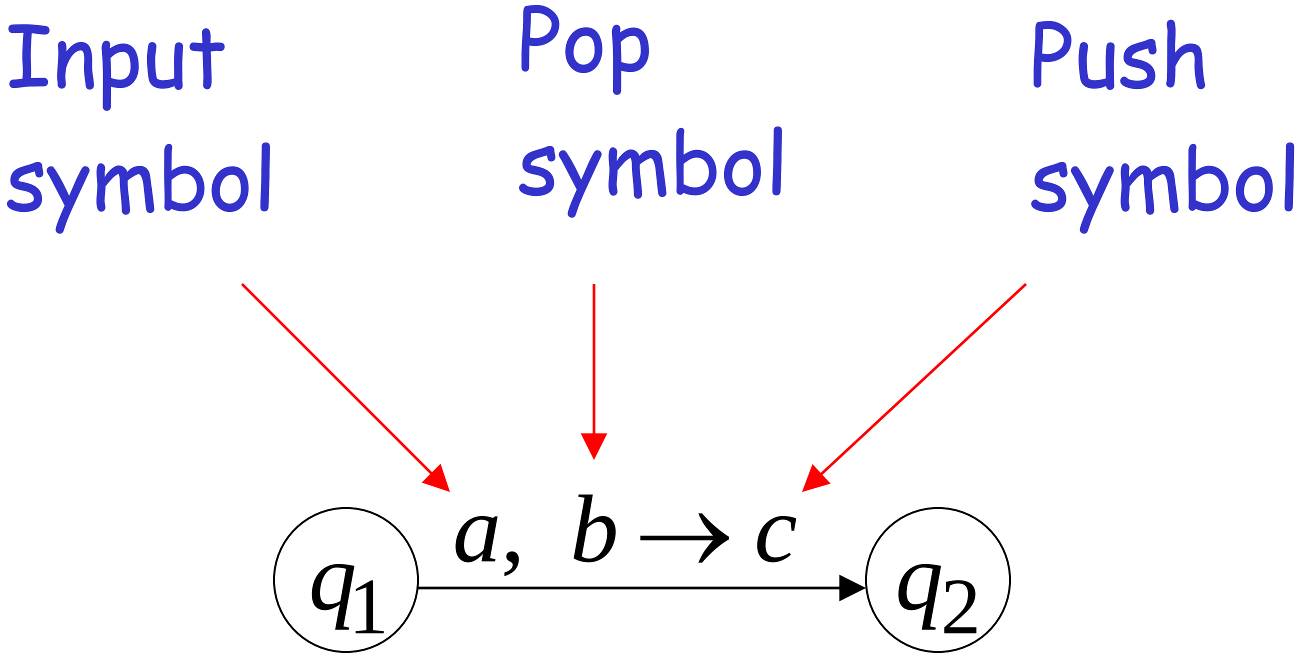
z

top

bottom special symbol

Appears at time 0

The States

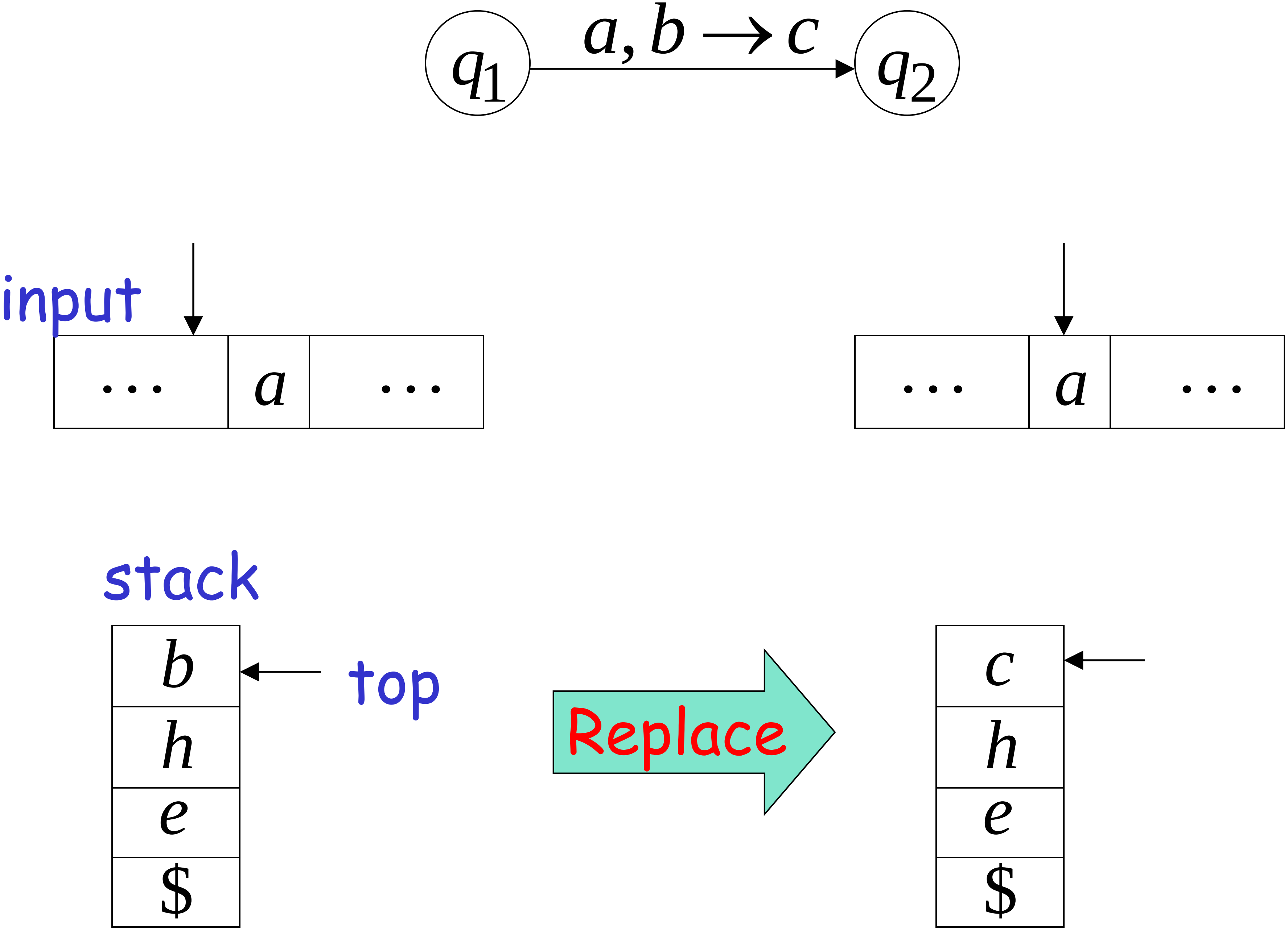


Input symbol

Pop symbol

Push symbol

$a,\ b \to c$



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

input

$\cdots \quad a \quad \cdots$

$\cdots \quad a \quad \cdots$

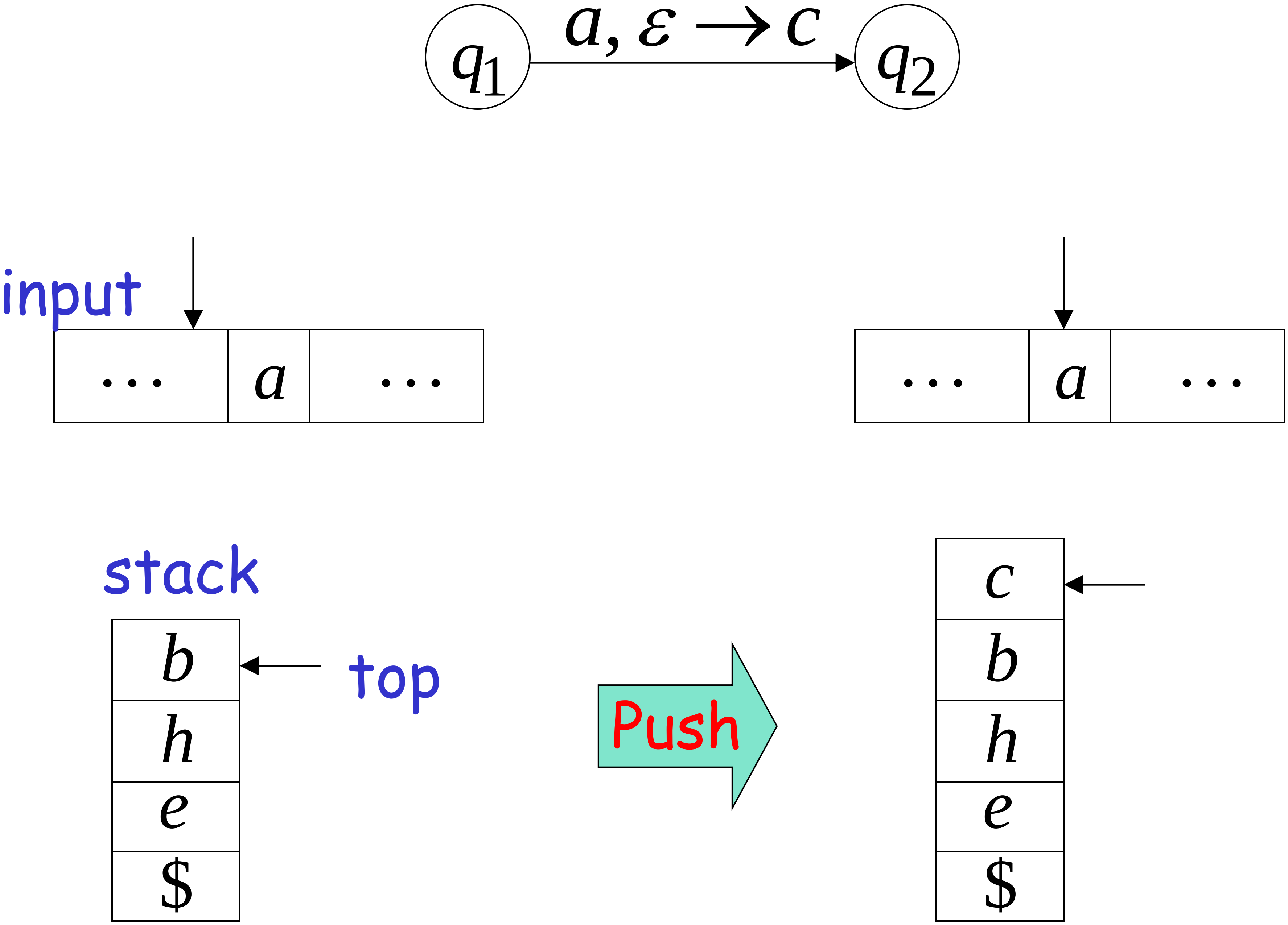
stack

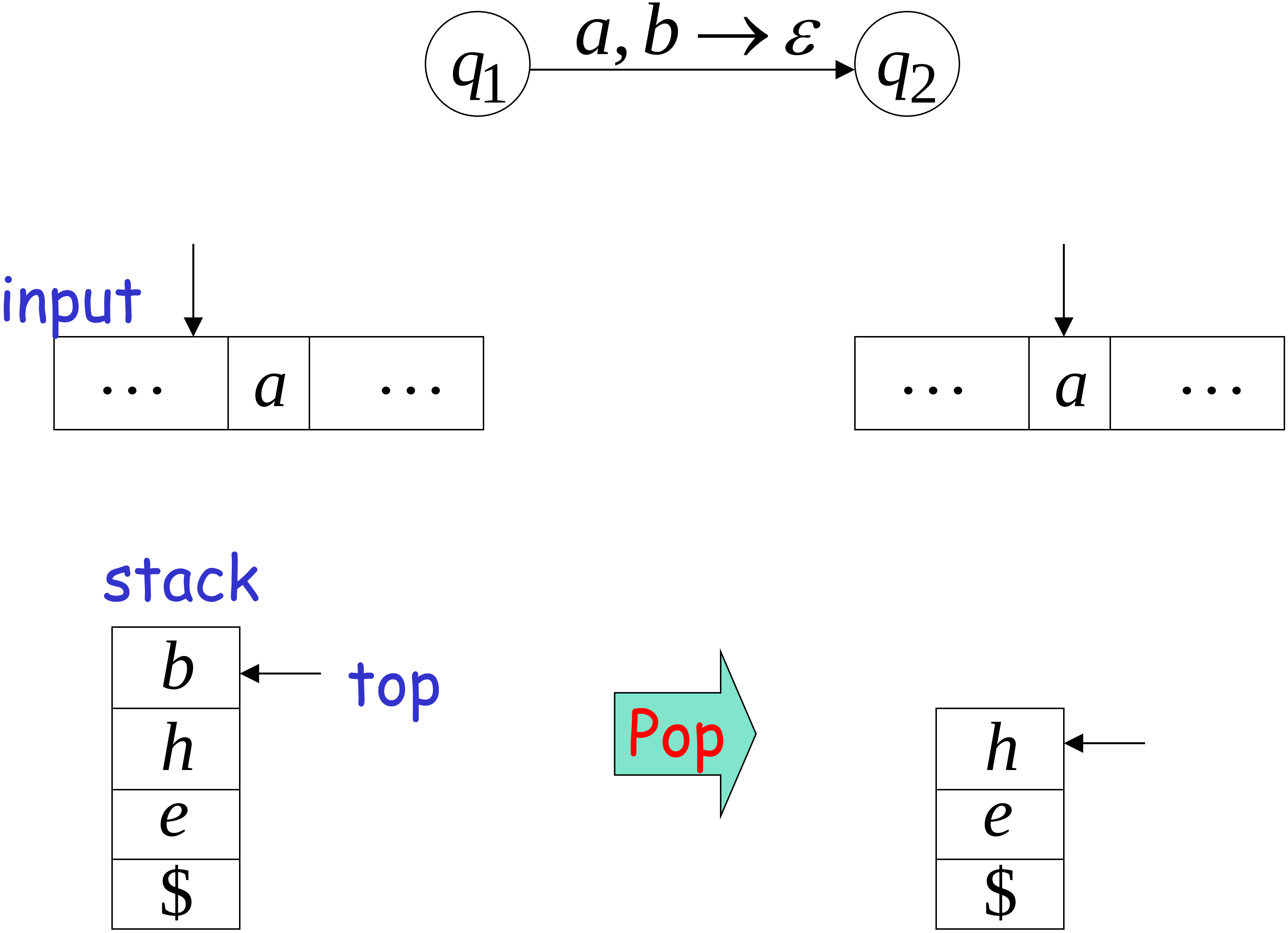
top

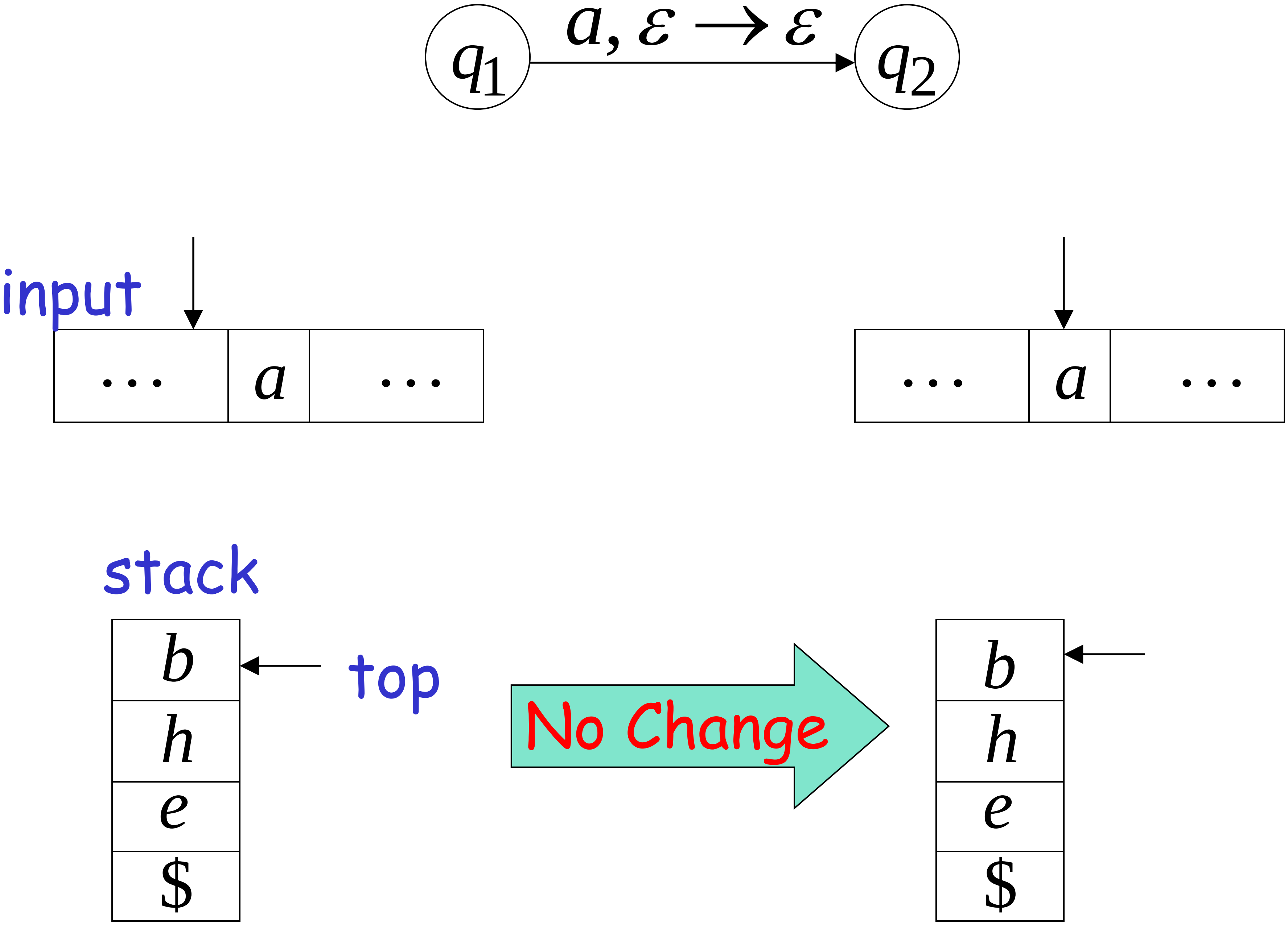
b, h, e, $\$$

Replace

c, h, e, $\$$


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


q_1
$a, b \to \varepsilon$
q_2
input
…
a
…
…
a
…
stack
b
h
e
$
top
Pop
h
e
$



$q_1 \xrightarrow{a, \varepsilon \rightarrow \varepsilon} q_2$

input

$\cdots \; a \; \cdots$

$\cdots \; a \; \cdots$

stack

top

No Change

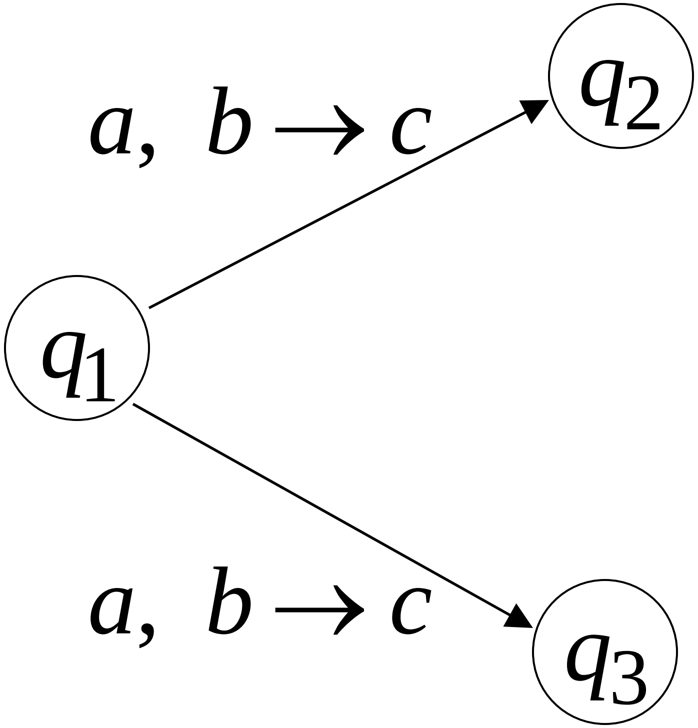
b, h, e, $\$$

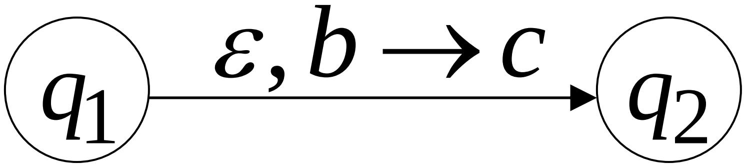
b, h, e, $\$$

Non-Determinism

PDAs are non-deterministic

Allowed non-deterministic transitions

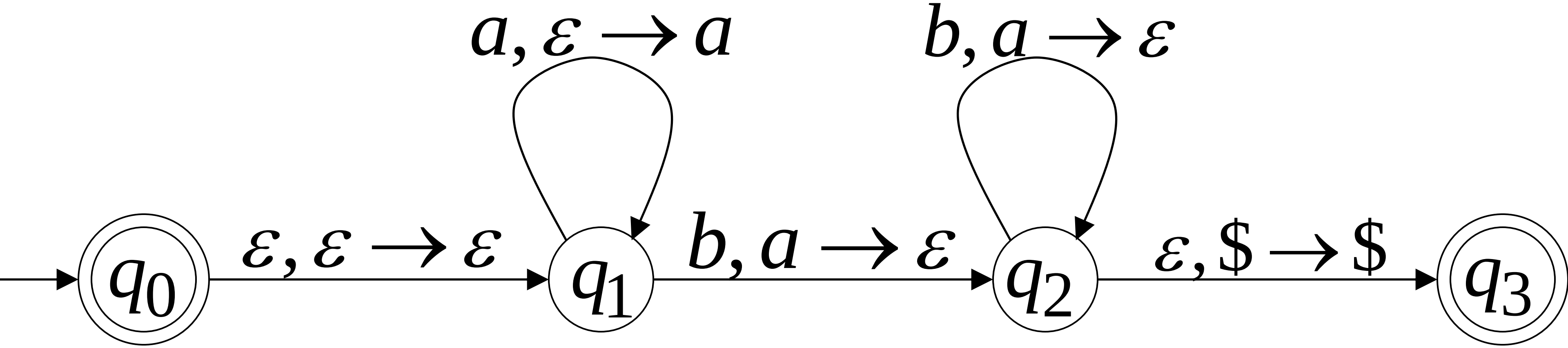




$\varepsilon -$ transition

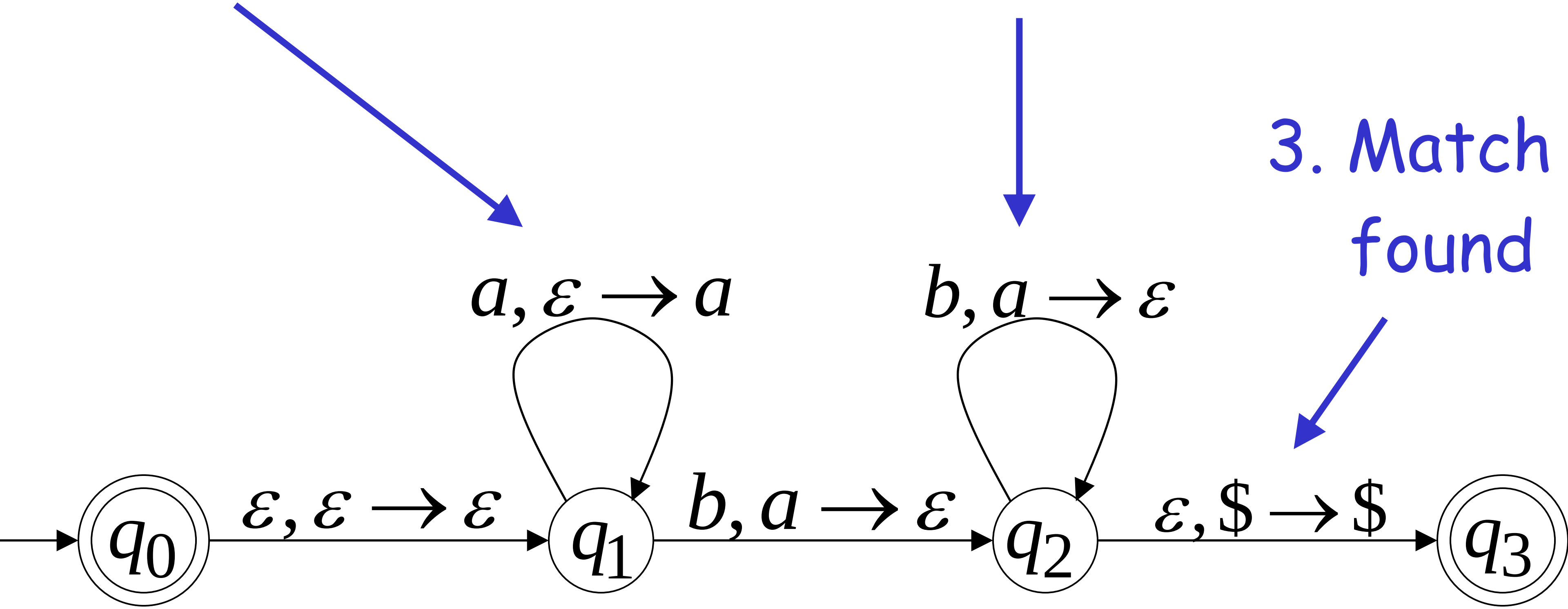
Example PDA

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



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

Basic Idea:



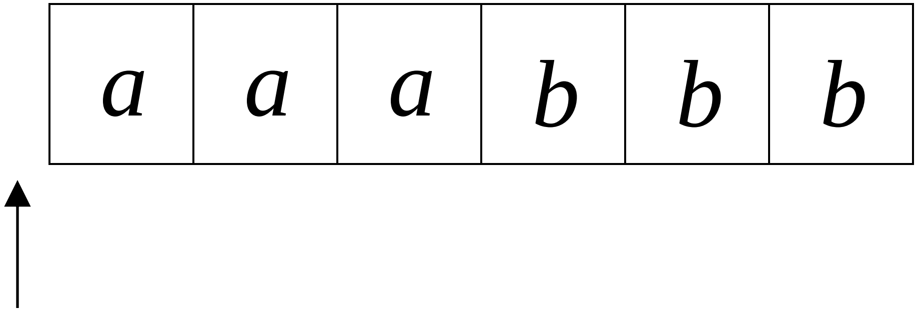
1. Push the a's on the stack

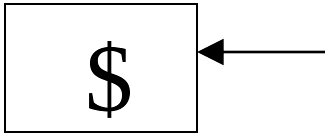
2. Match the b's on input with a's on stack

3. Match found

Execution Example: Time 0

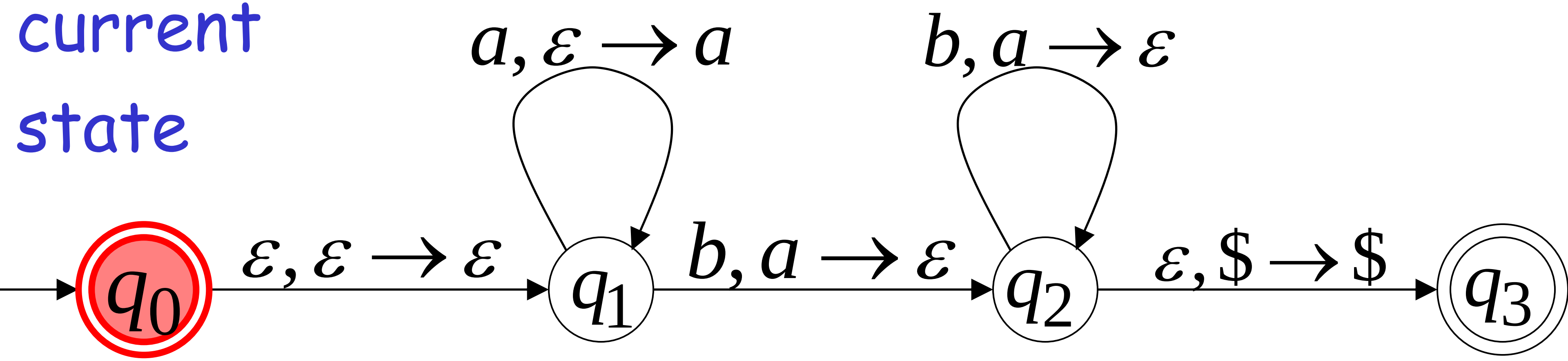
Input



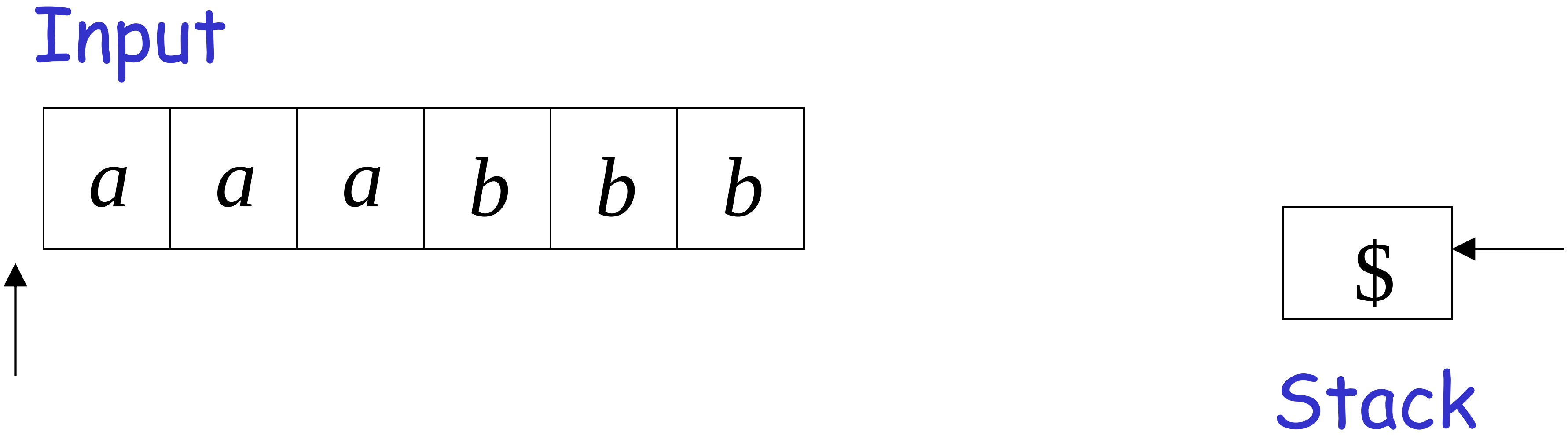


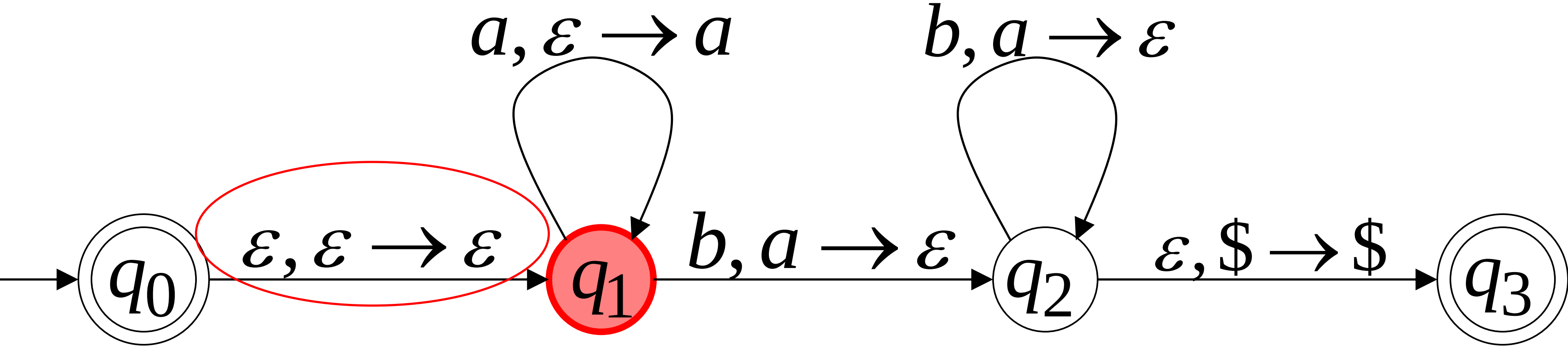
Stack

current state



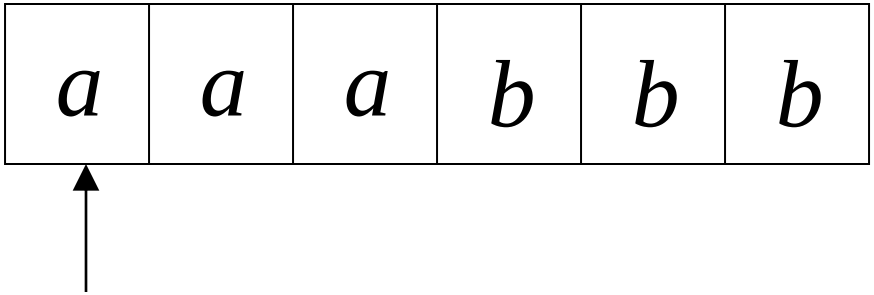
Time 1

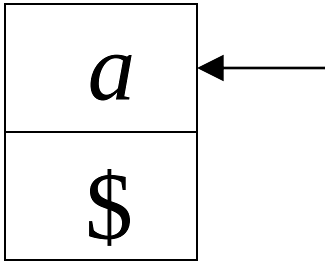

Input
a
a
a
b
b
b
$\$$
Stack


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

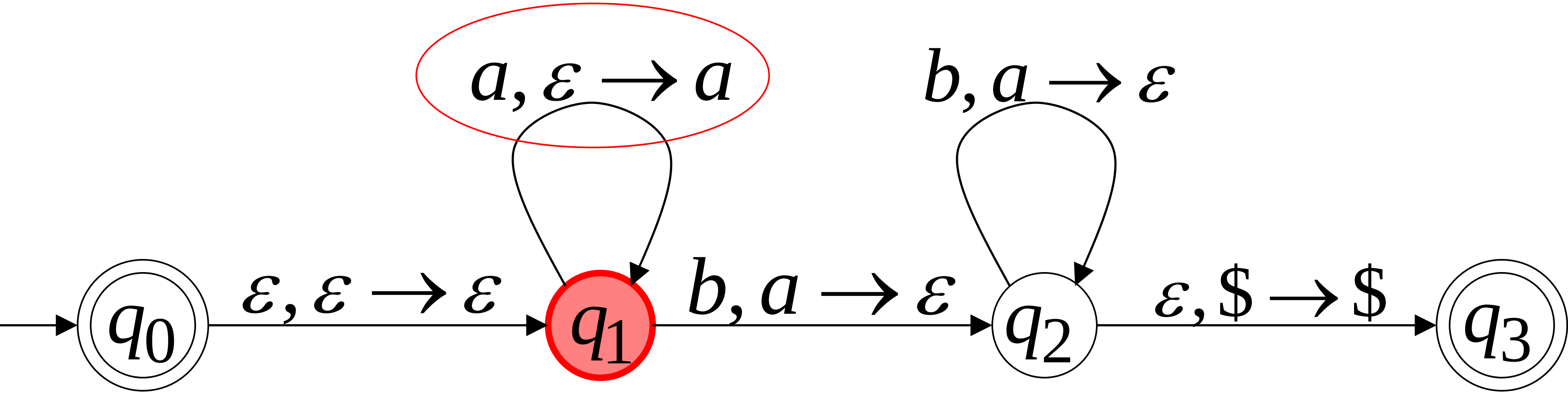
Time 2

Input

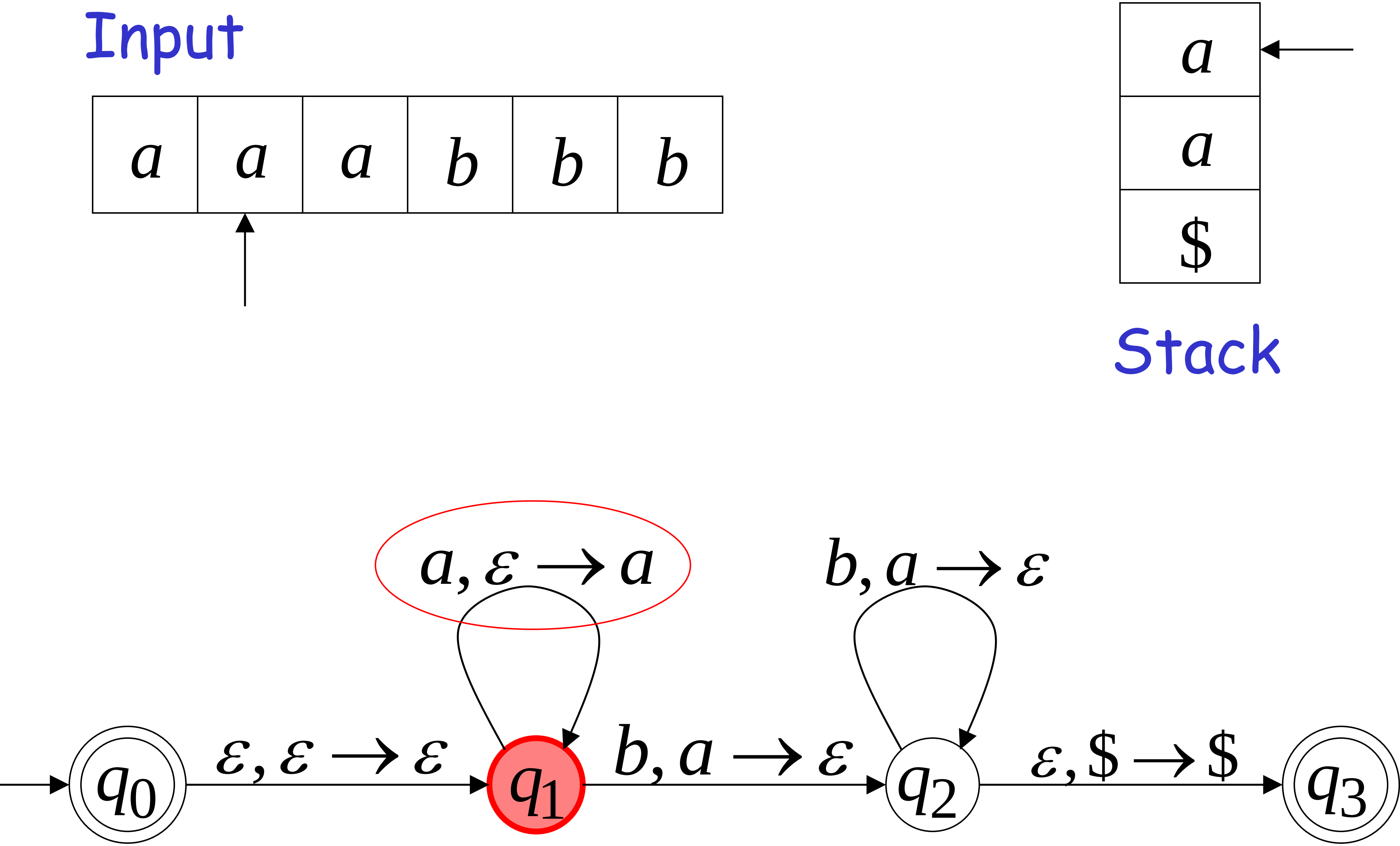




Stack

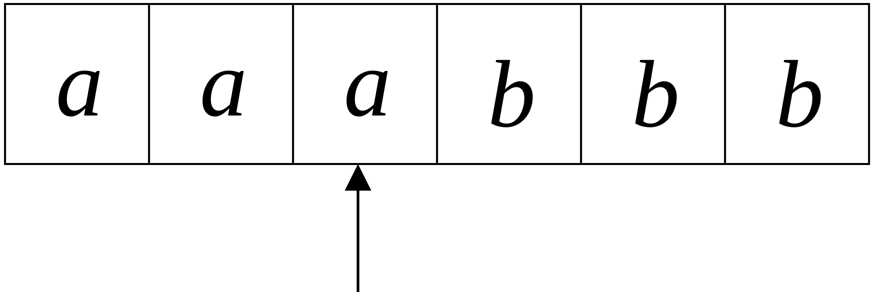


Time 3

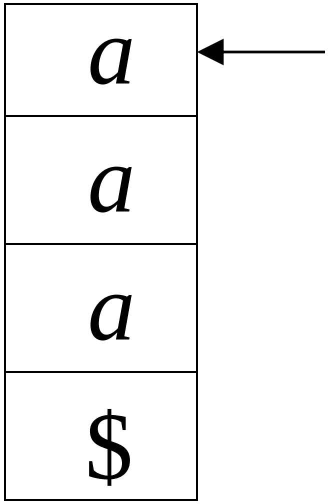

Input
a a a b b b
a
a
$\$$
Stack
$a, \varepsilon \rightarrow a$
$b, a \rightarrow \varepsilon$
$\varepsilon, \varepsilon \rightarrow \varepsilon$
$b, a \rightarrow \varepsilon$
$\varepsilon, \$ \rightarrow \$$
q_0
q_1
q_2
q_3

Time 4

Input

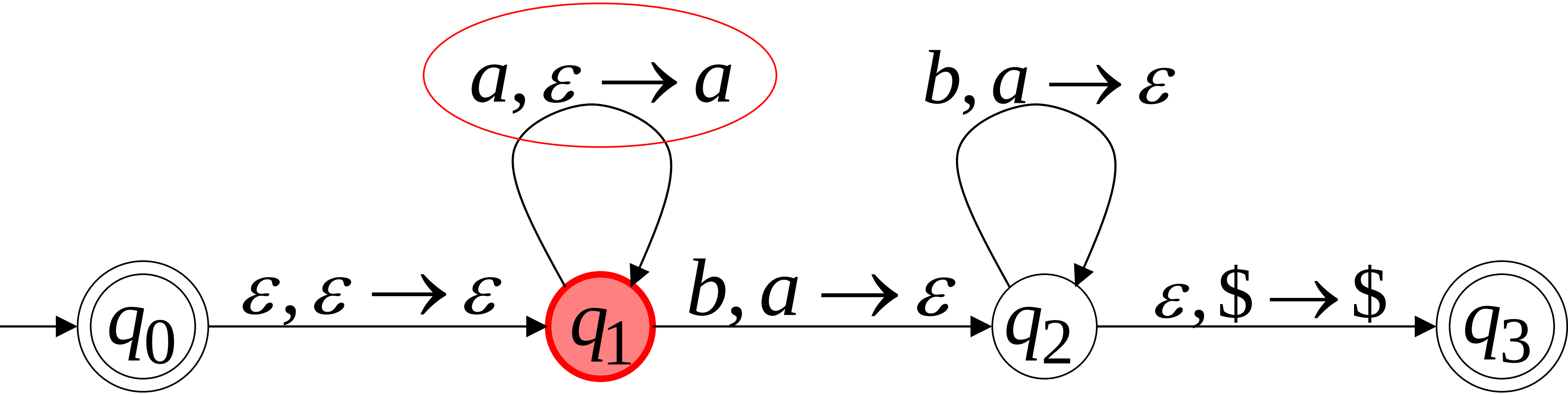


a	a	a	b	b	b



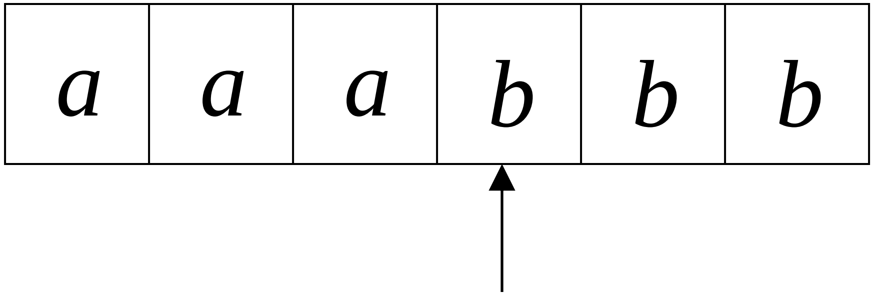
a
a
a
$\$$

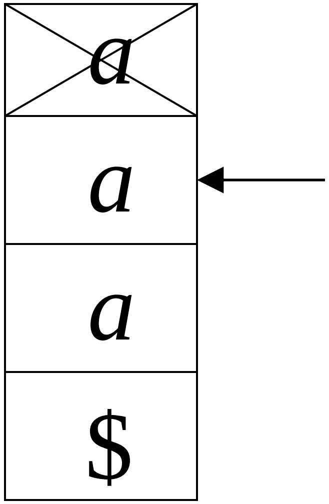
Stack



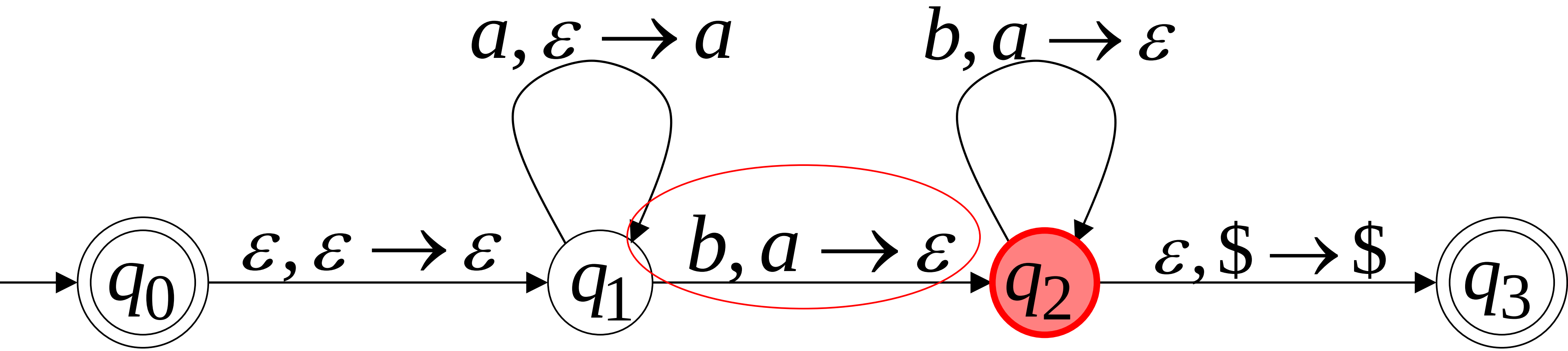
Time 5

Input



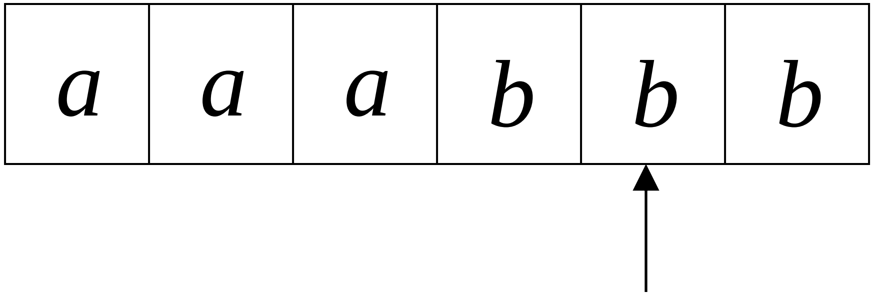


Stack

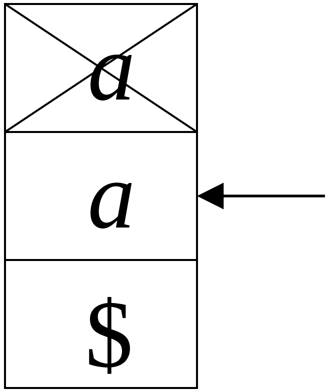


Time 6

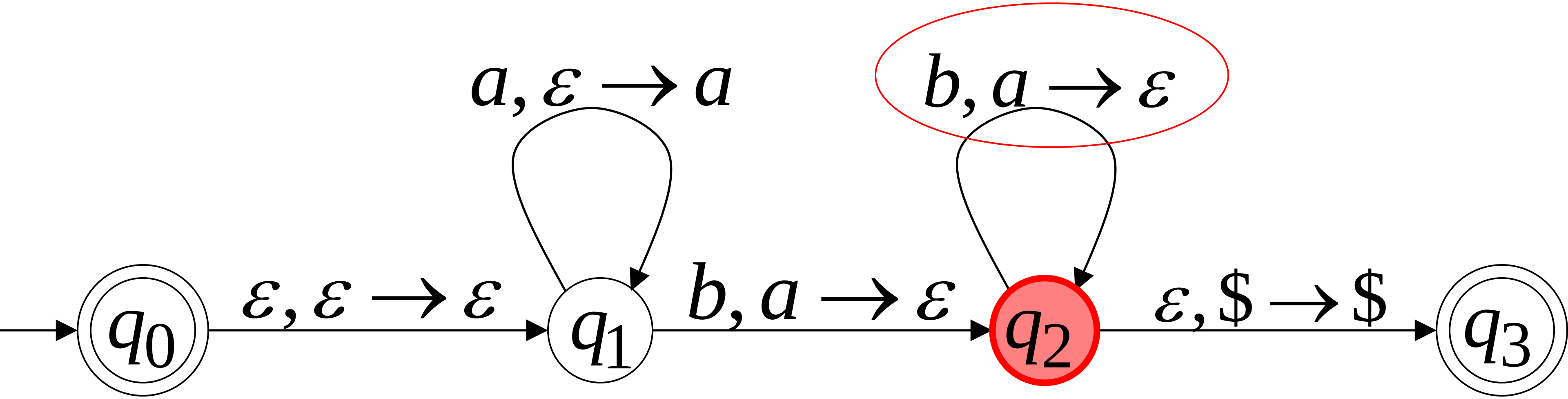
Input



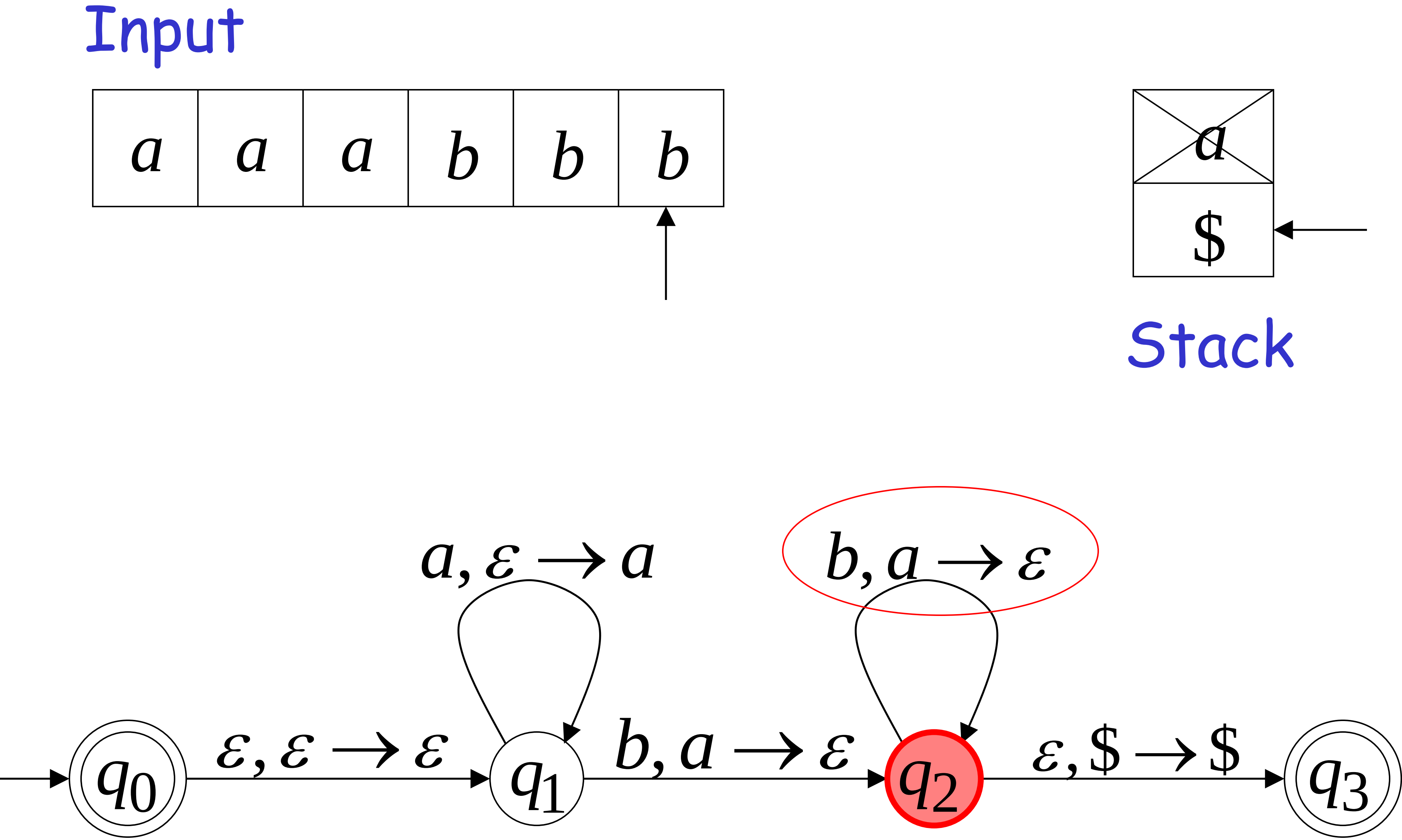
a	a	a	b	b	b



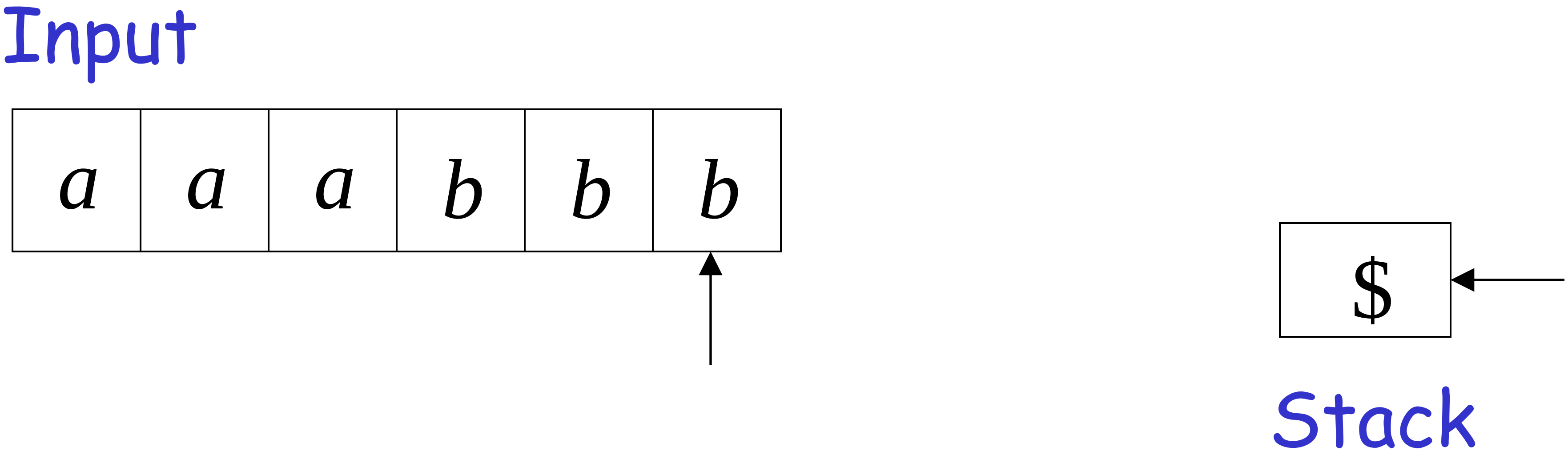
Stack

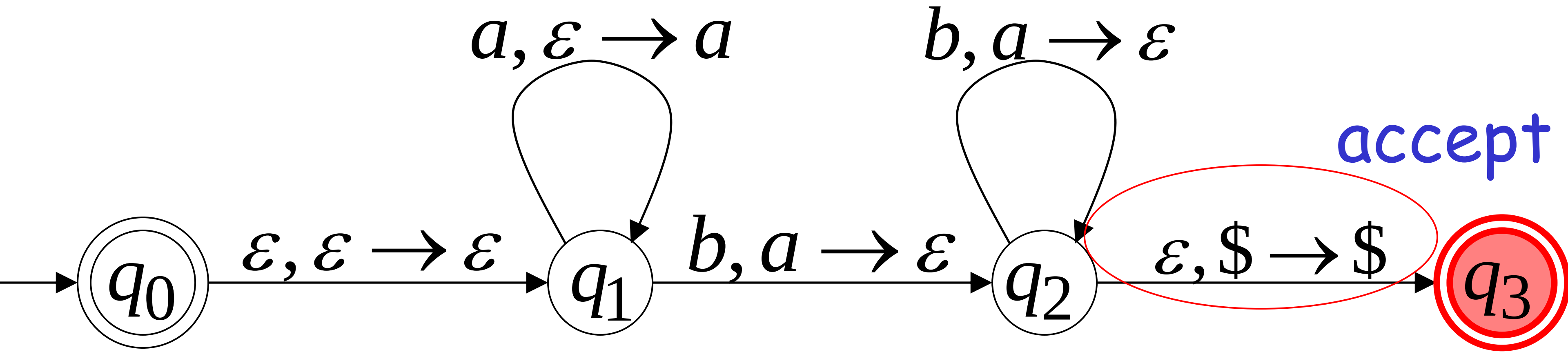


Time 7


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

Time 8


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


$a, \varepsilon \rightarrow a$
$b, a \rightarrow \varepsilon$
accept
q_0
$\varepsilon, \varepsilon \rightarrow \varepsilon$
q_1
$b, a \rightarrow \varepsilon$
q_2
$\varepsilon, \$ \rightarrow \$$
q_3

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

All the input is consumed

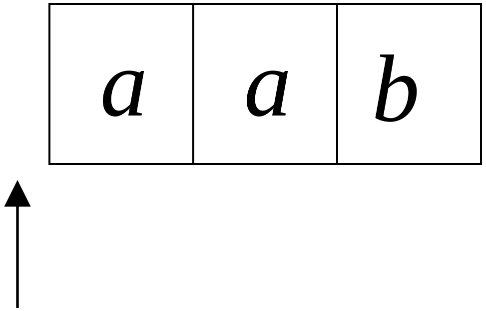
AND

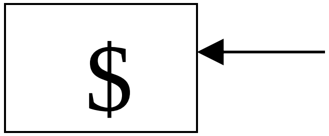
The last state is an accepting state

we do not care about the stack contents at the end of the accepting computation

Rejection Example: Time 0

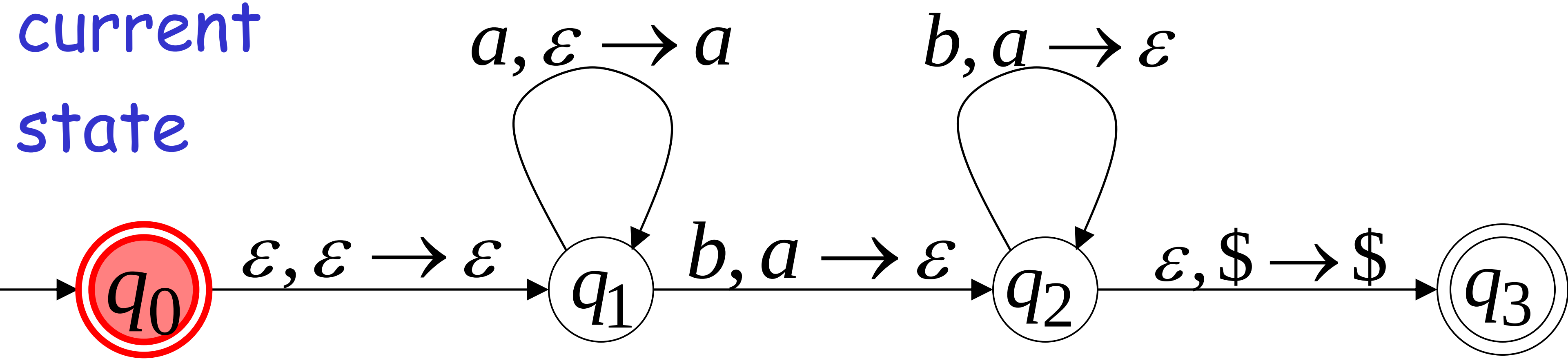
Input



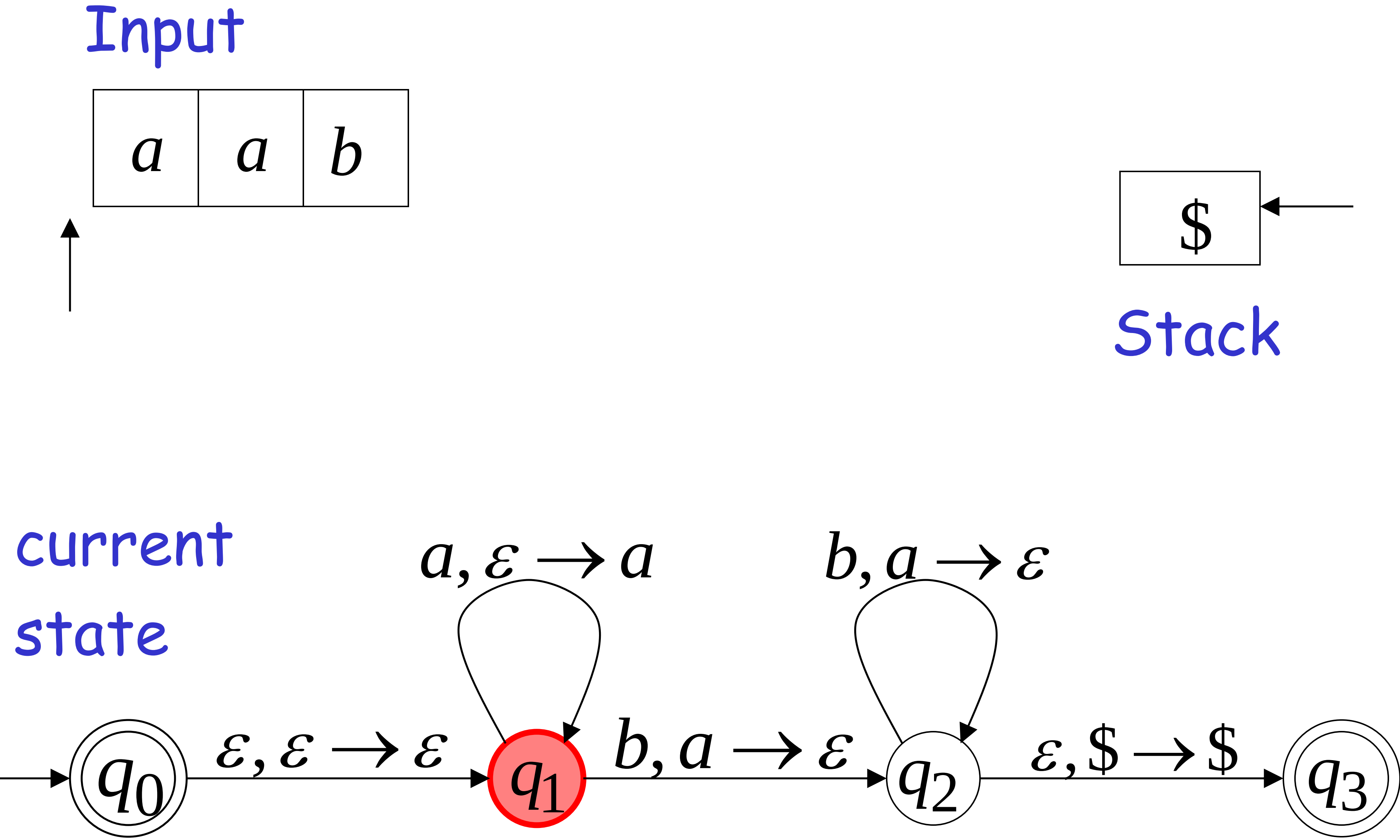


Stack

current state

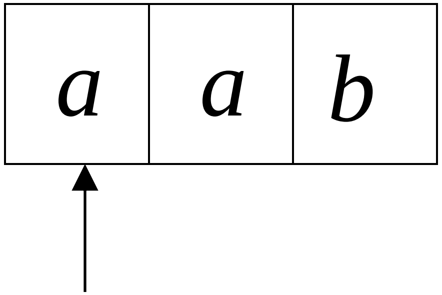


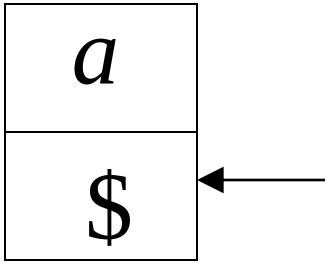
Rejection Example: Time 1



Rejection Example: Time 2

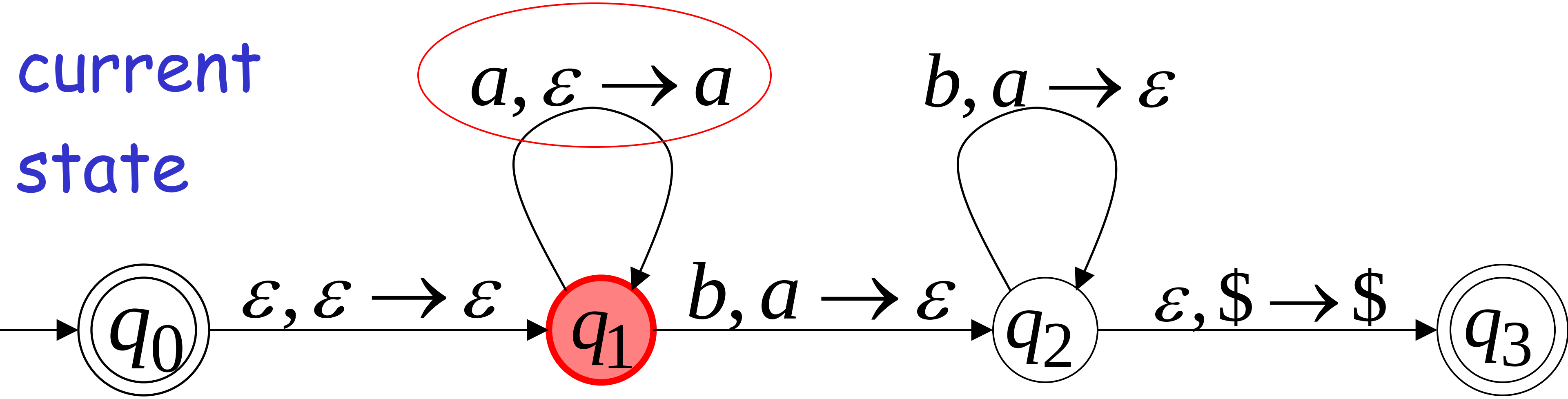
Input





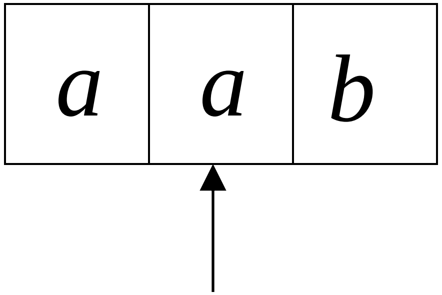
Stack

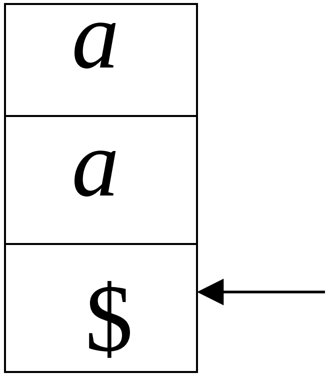
current state



Rejection Example: Time 3

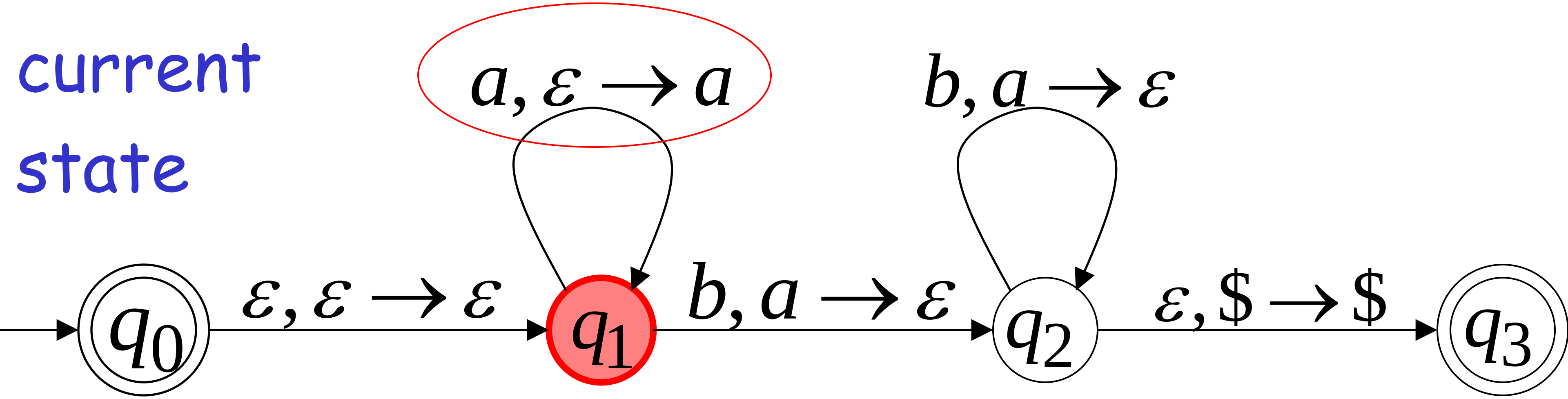
Input





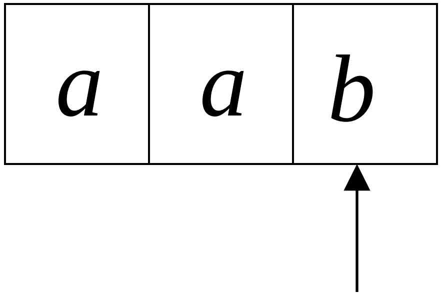
Stack

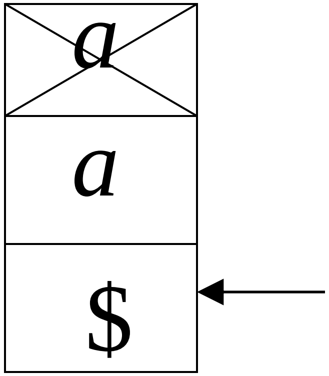
current state



Rejection Example: Time 4

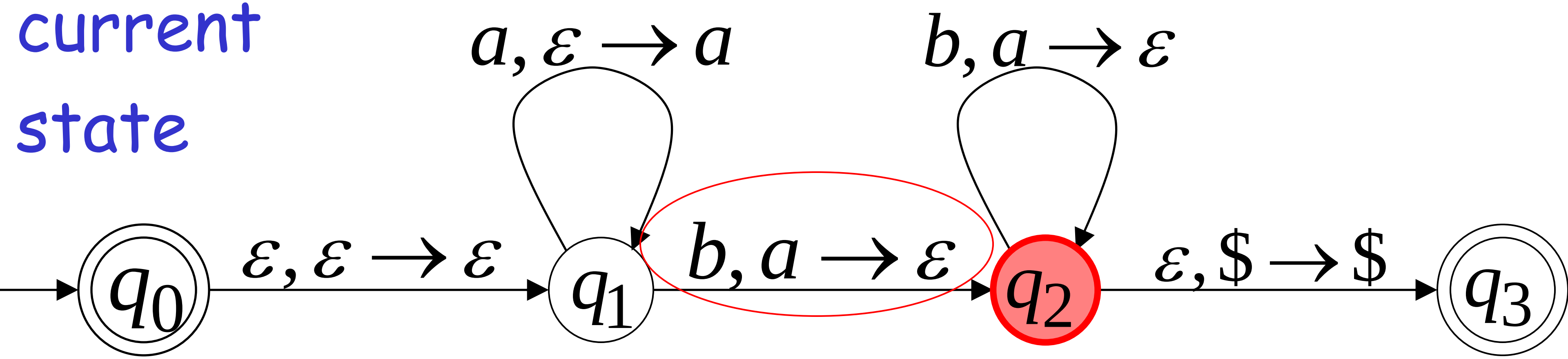
Input





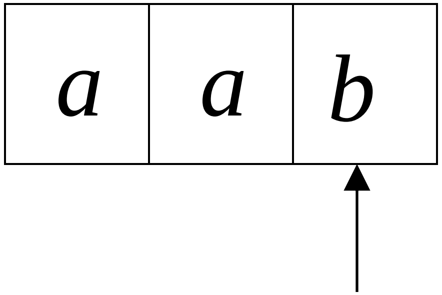
Stack

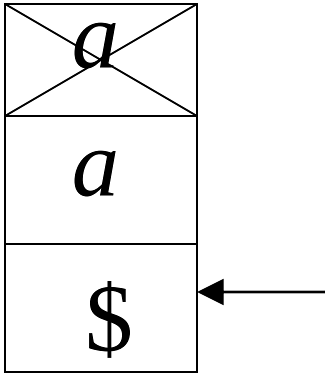
current state



Rejection Example: Time 4

Input

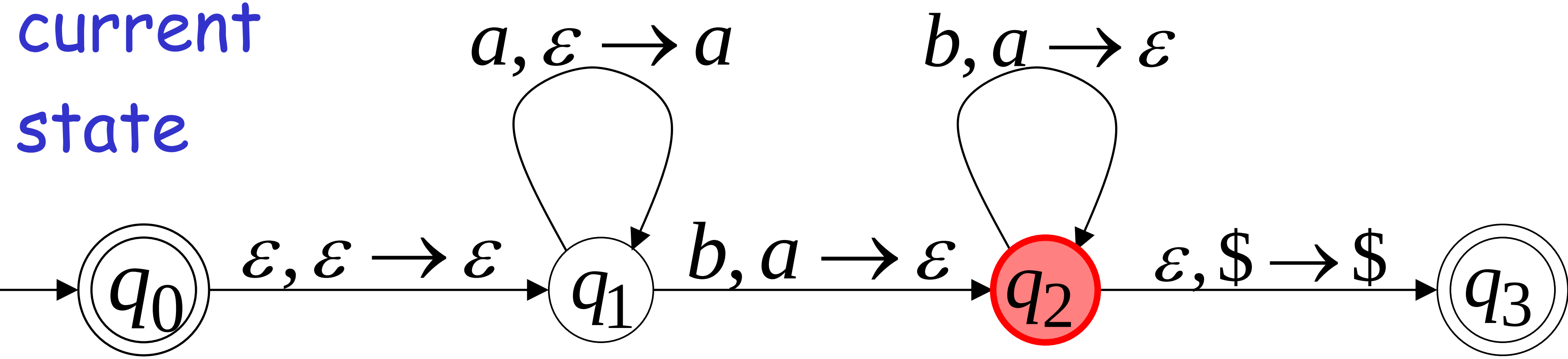




Stack

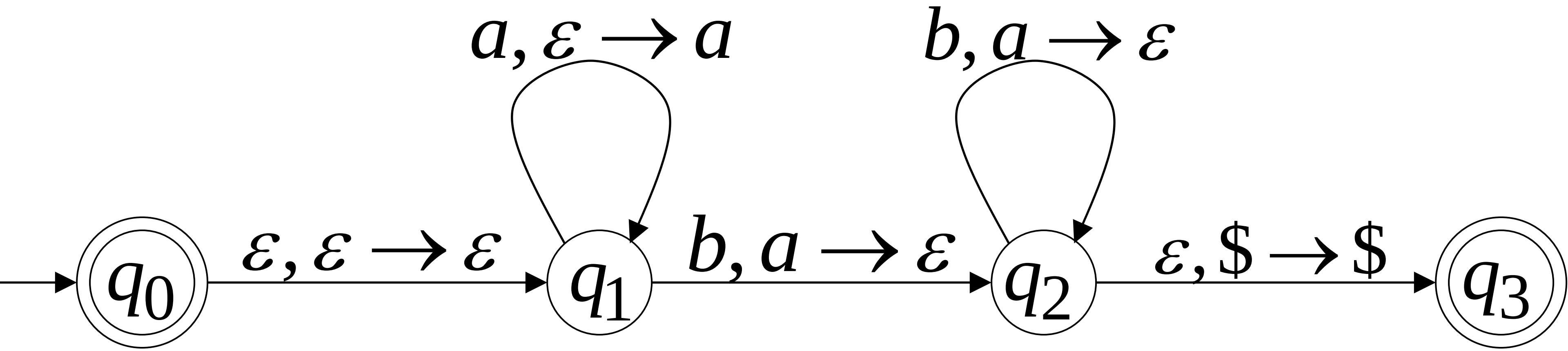
reject

current state



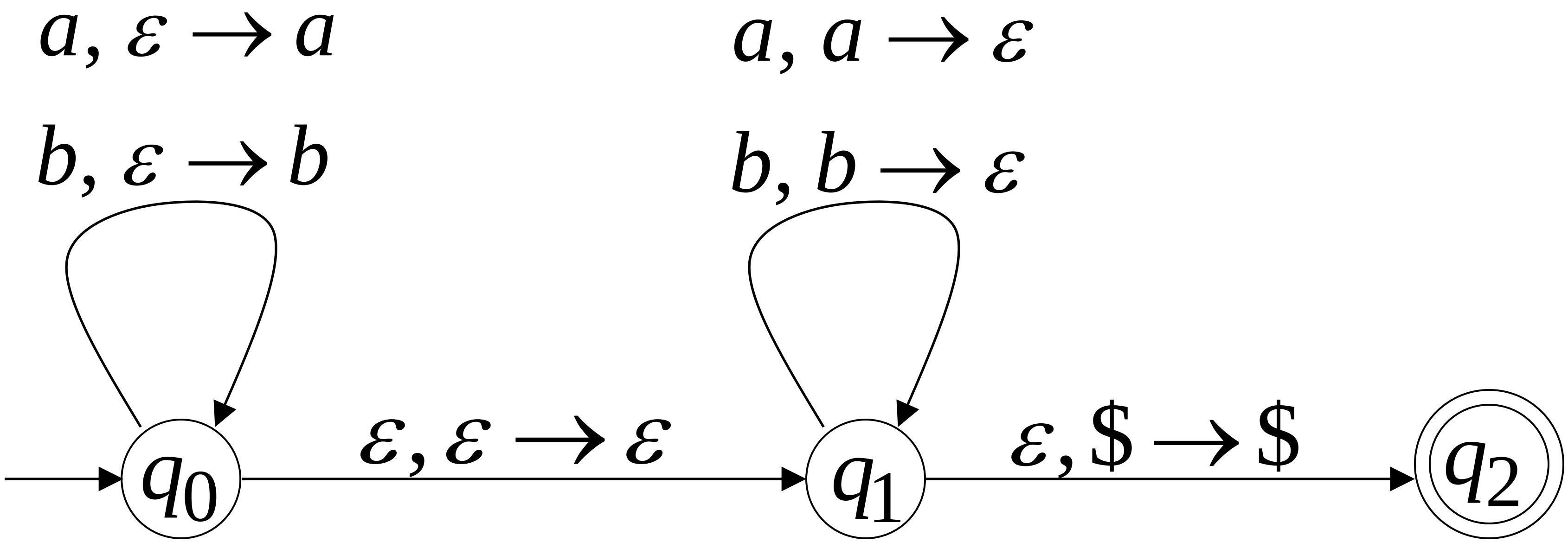
There is no accepting computation for aab

The string aab is rejected by the PDA



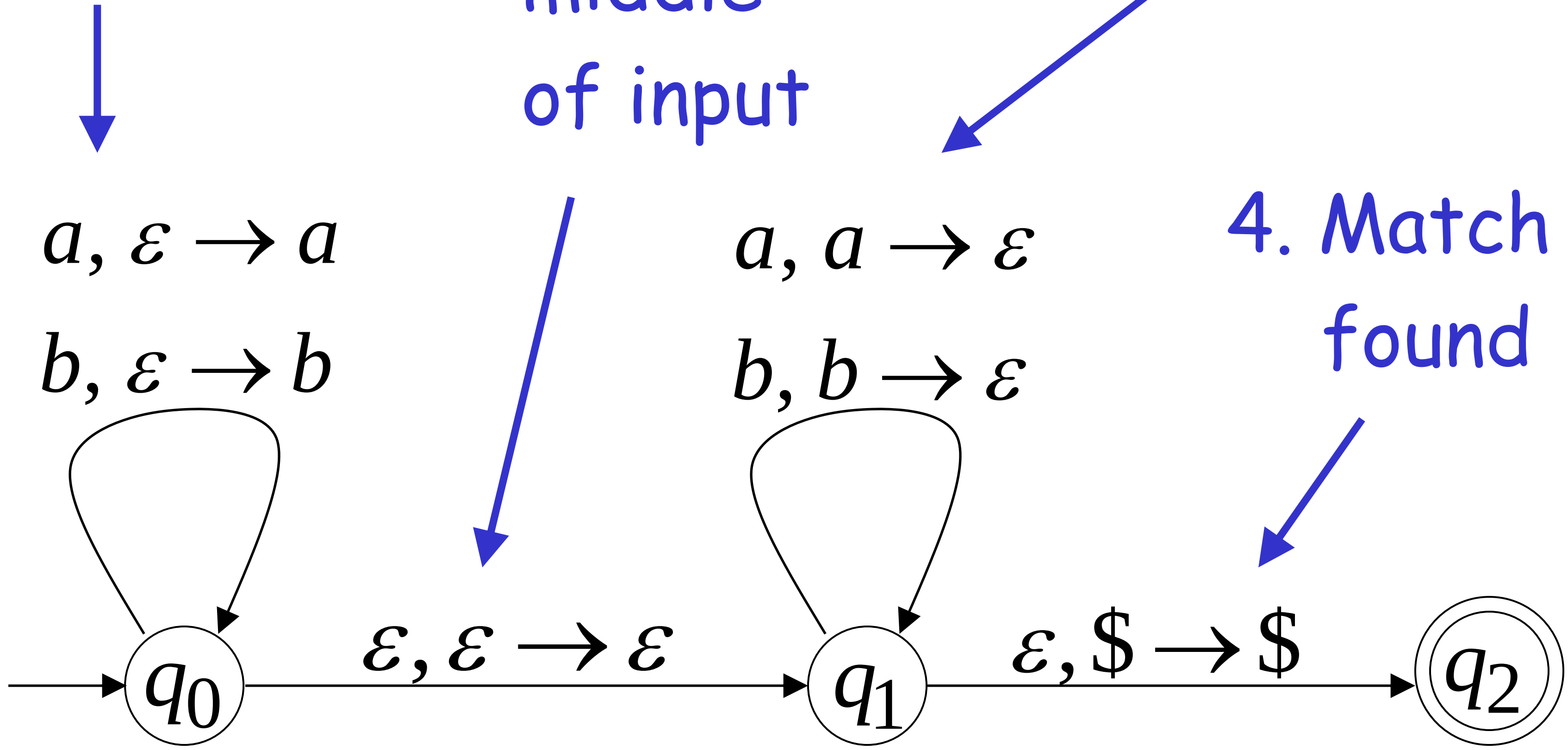
Another PDA example

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



Basic Idea:

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



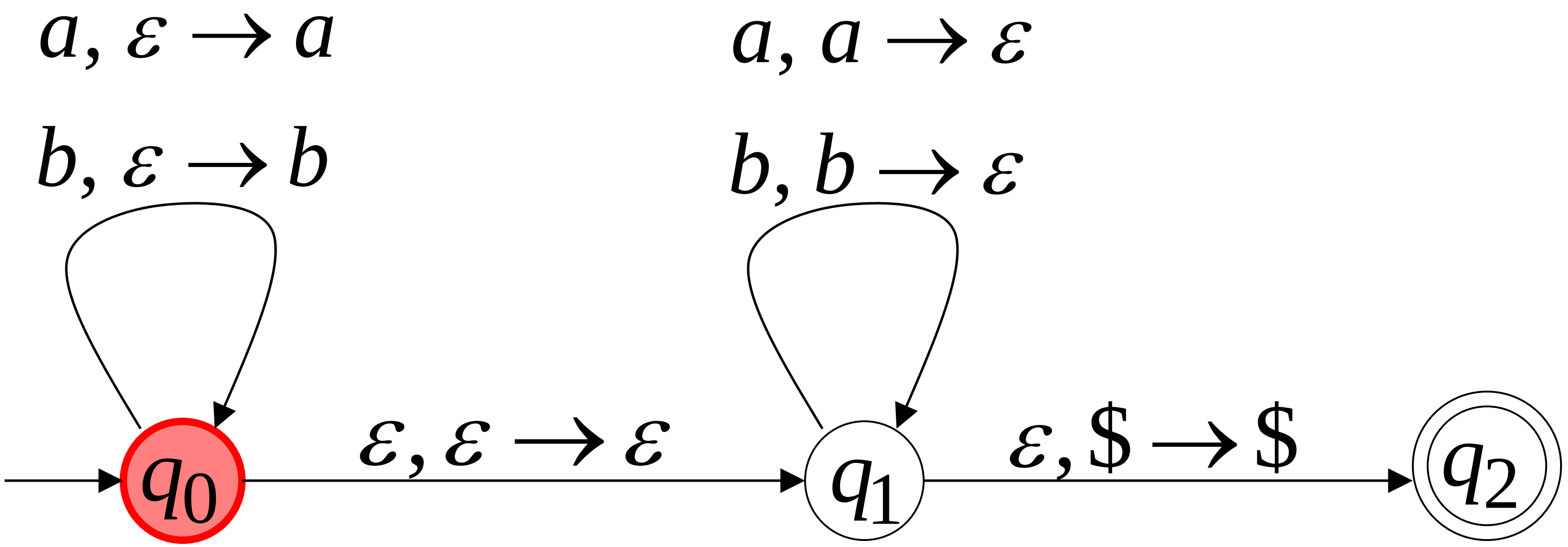
1. Push v on stack

2. Guess middle of input

3. Match v^R on input with v on stack

4. Match found

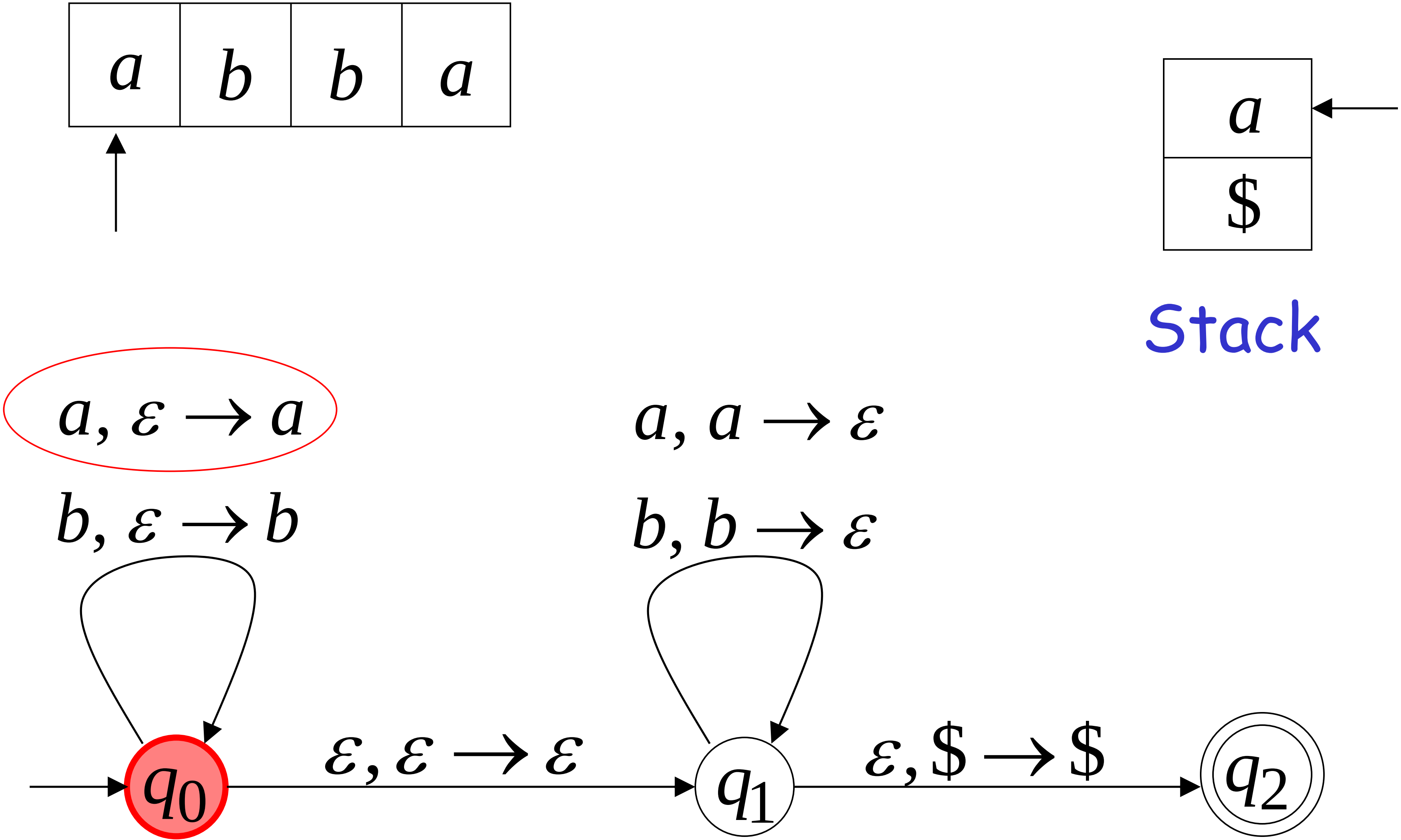
Execution Example: Time 0



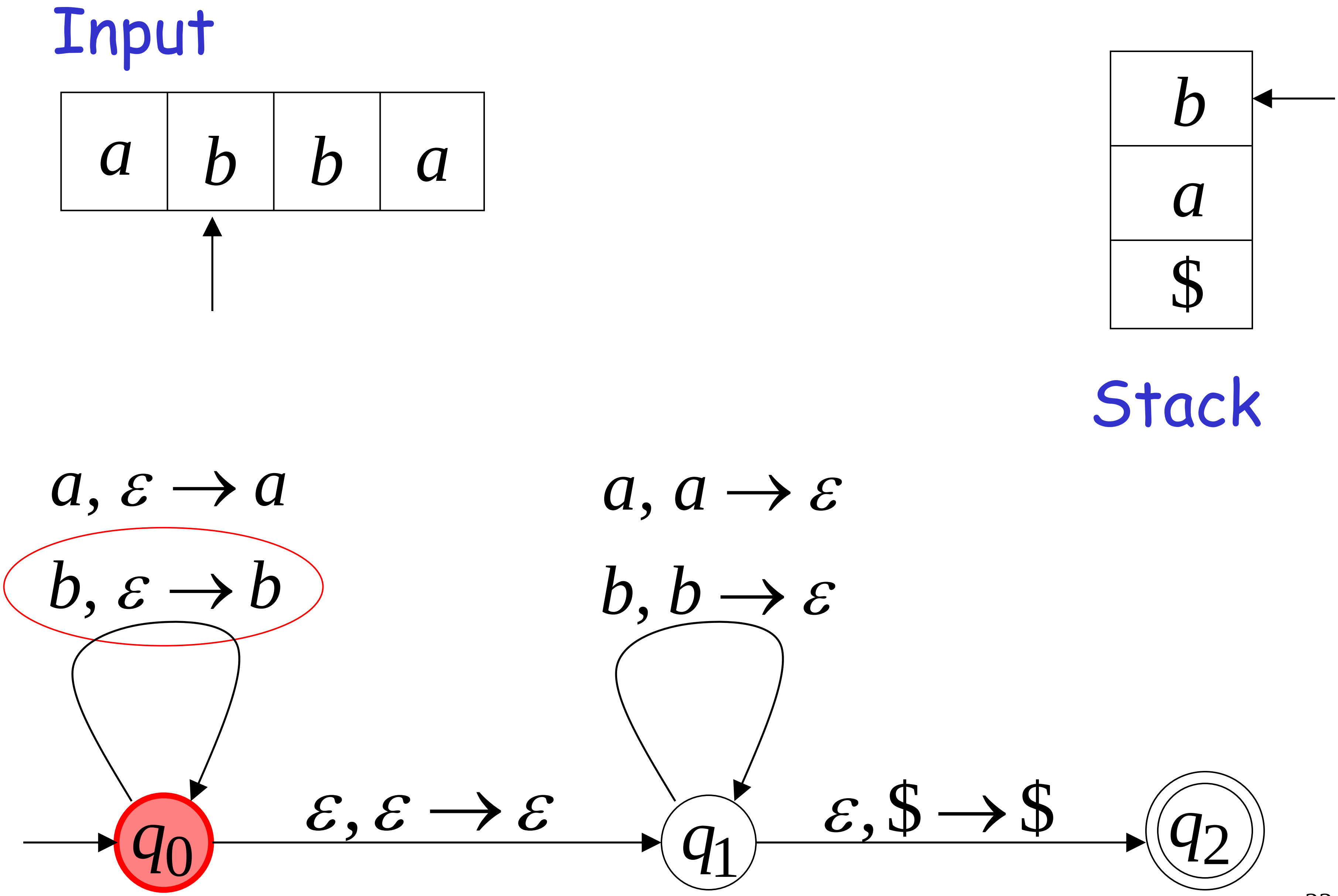
Time 1

Input

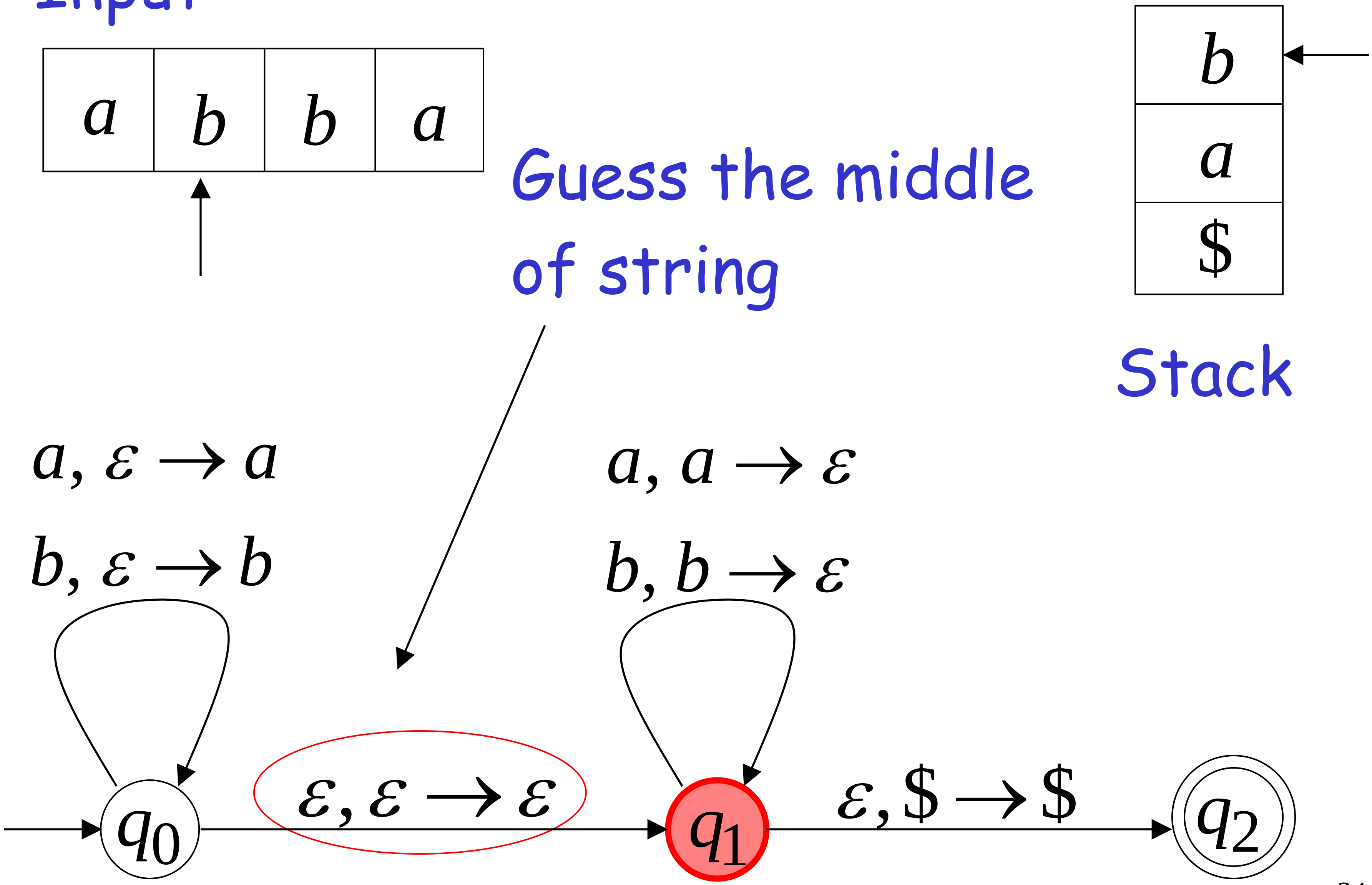
Stack



Time 2


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

Time 3

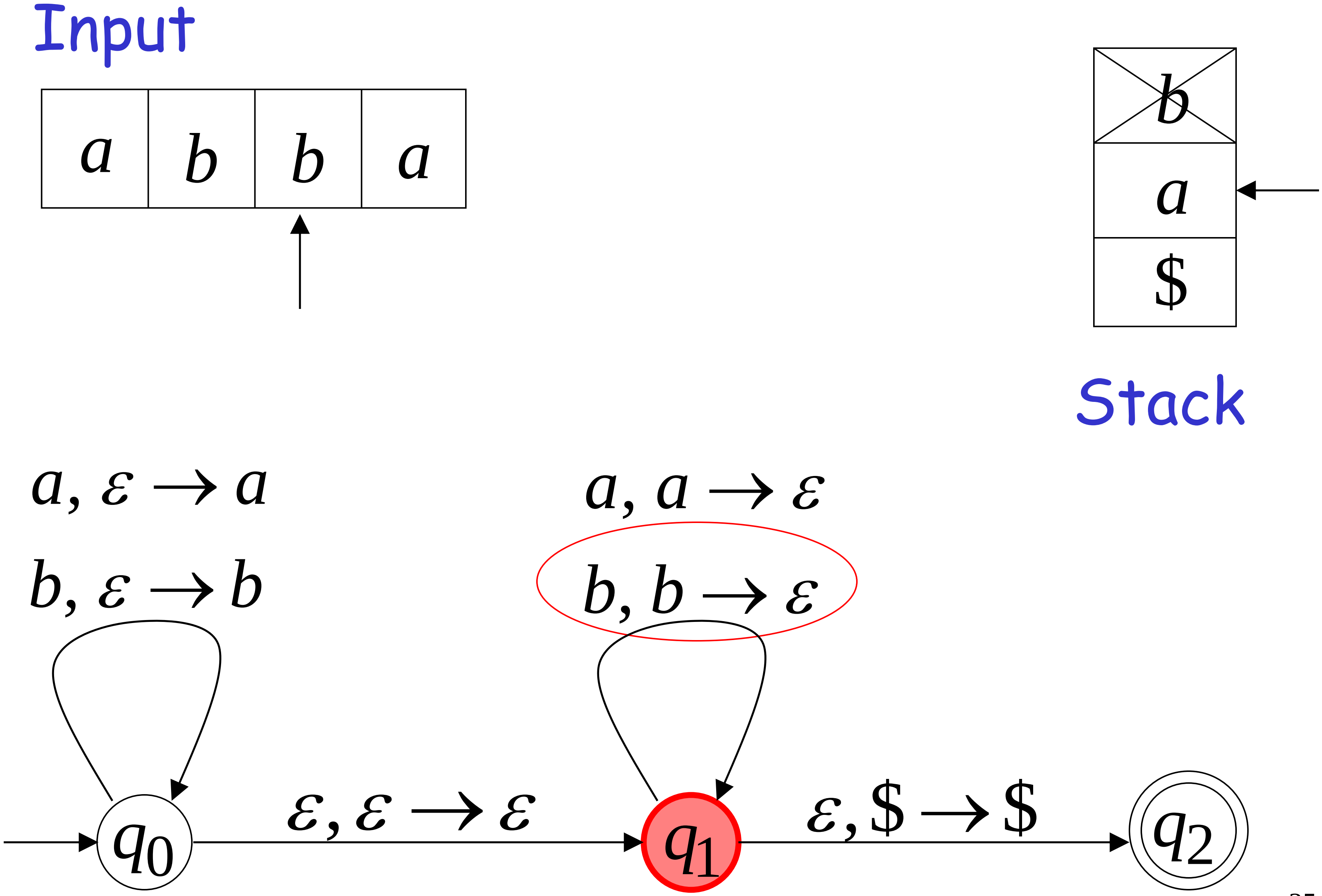


Input

Guess the middle of string

Stack

Time 4



Input

a b b a

$\cancel{b}$

a

$\$$

Stack

$a, \varepsilon \rightarrow a$

$b, \varepsilon \rightarrow b$

$a, a \rightarrow \varepsilon$

$b, b \rightarrow \varepsilon$

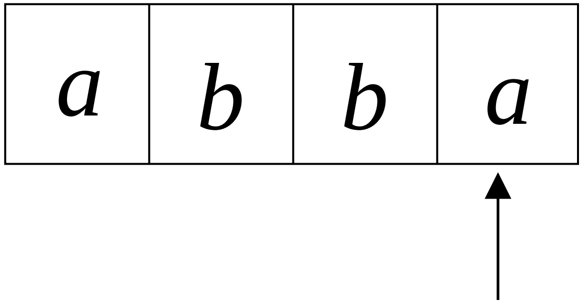
$\varepsilon, \varepsilon \rightarrow \varepsilon$

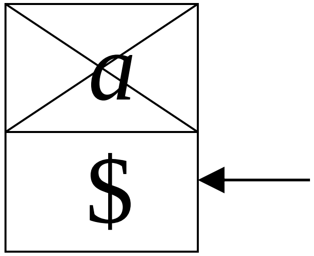
$\varepsilon, \$ \rightarrow \$$

q_0 q_1 q_2

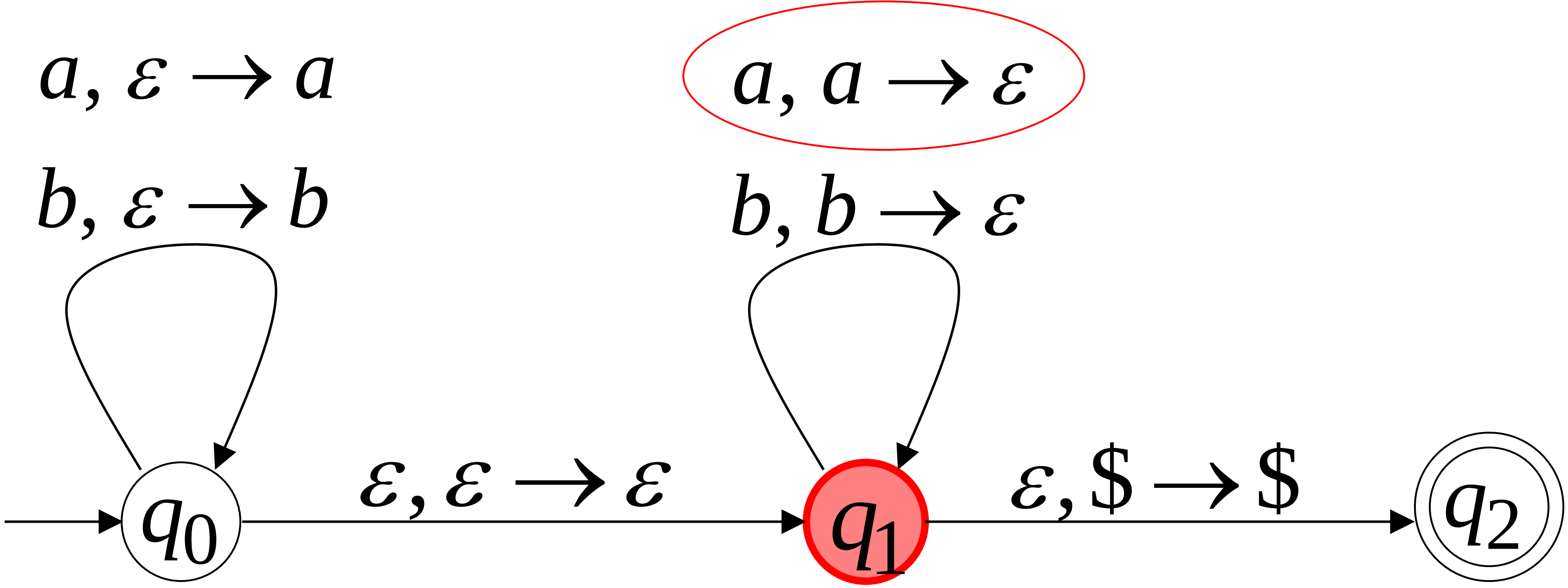
Time 5

Input

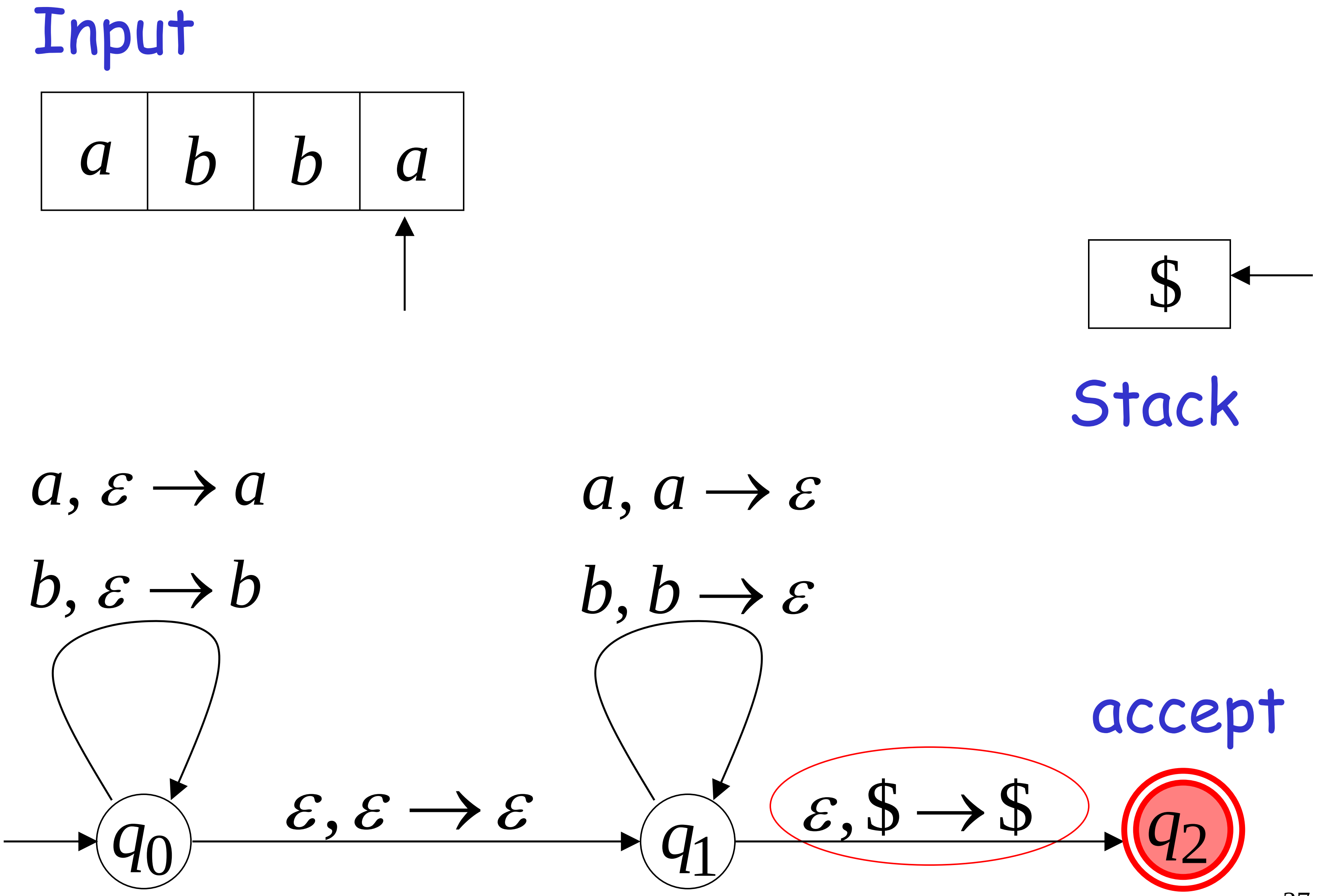




Stack

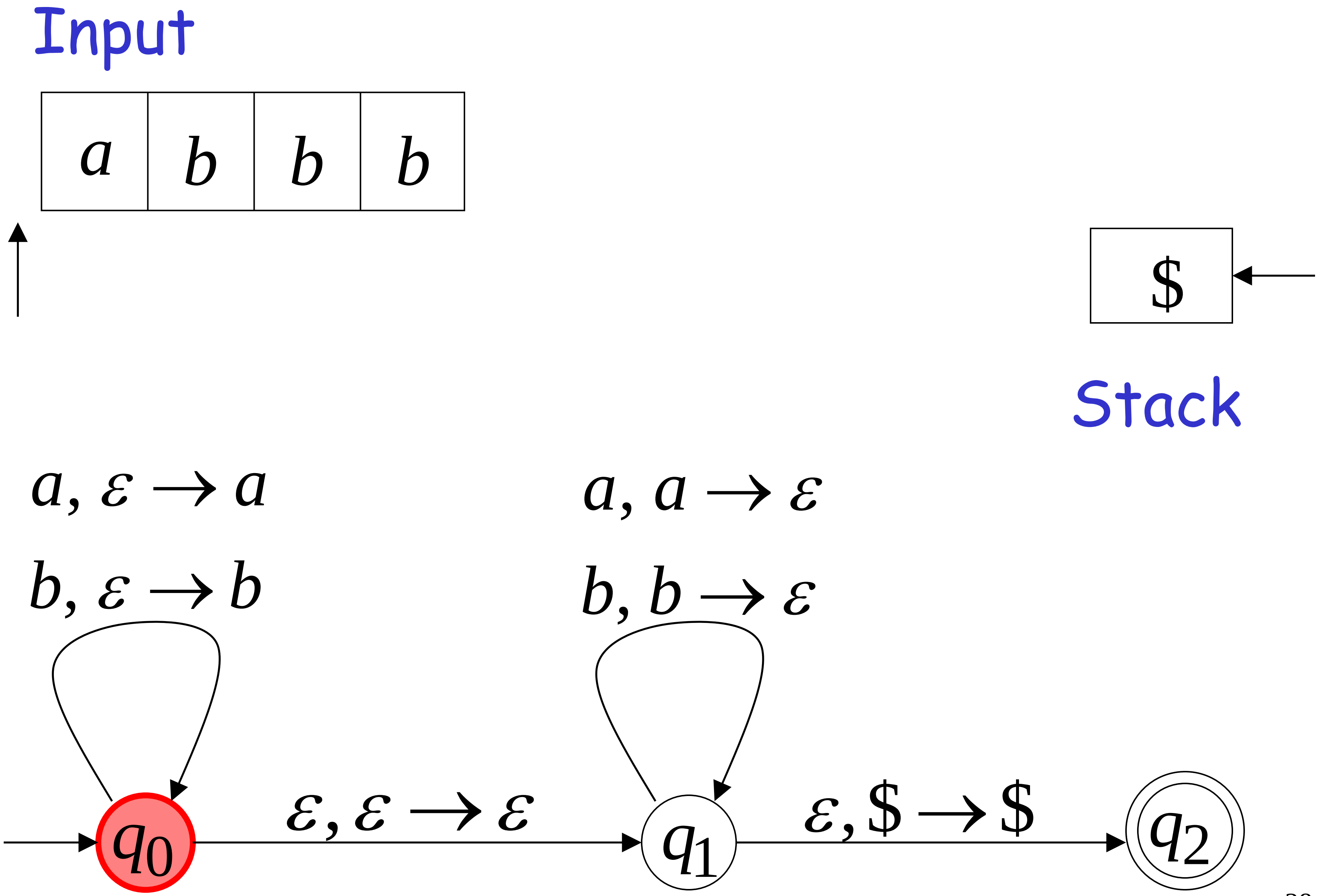


Time 6

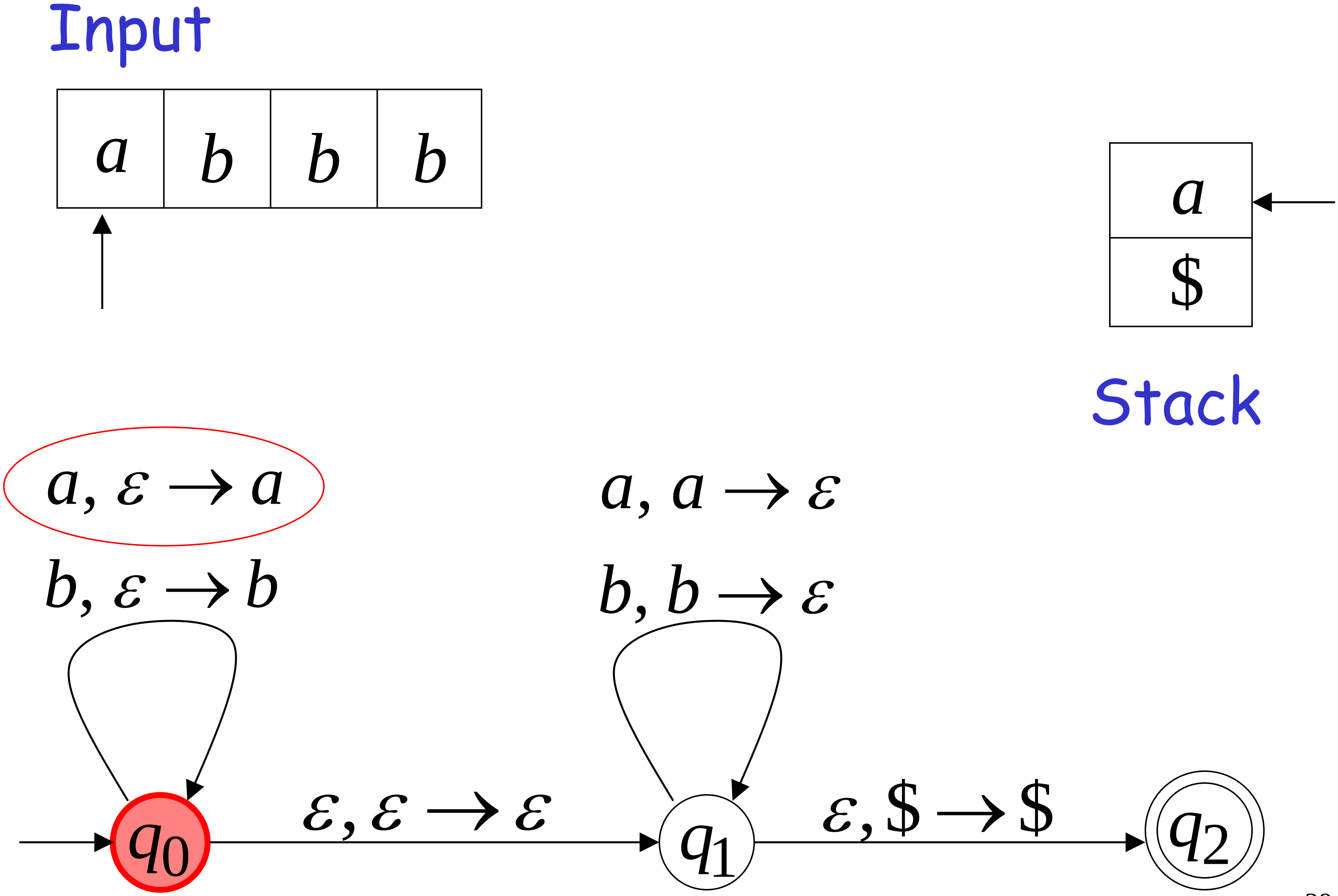

Input
a
b
b
a
$
Stack
$a, \varepsilon \rightarrow a$
$b, \varepsilon \rightarrow b$
$a, a \rightarrow \varepsilon$
$b, b \rightarrow \varepsilon$
q_0
$\varepsilon, \varepsilon \rightarrow \varepsilon$
q_1
$\varepsilon, \$ \rightarrow \$$
q_2
accept

Rejection Example:

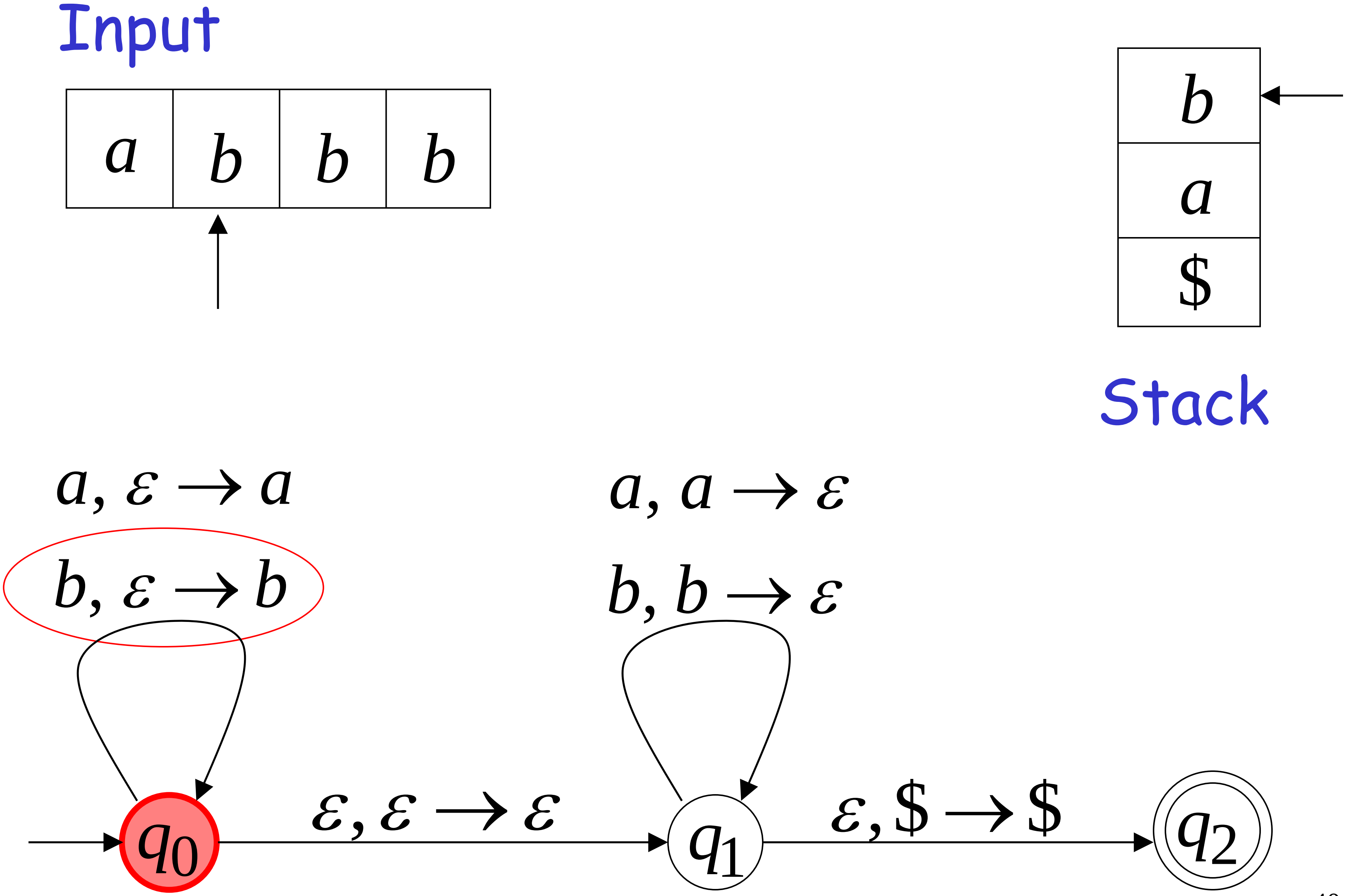
Time 0



Time 1

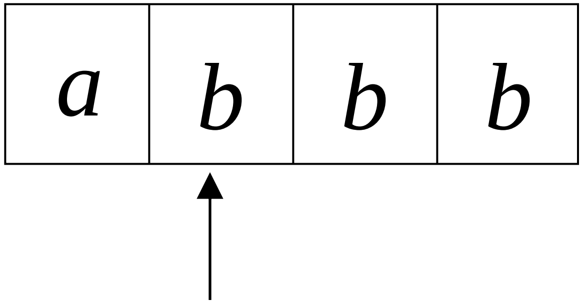

Input
a
b
b
b
a
$
Stack
$a, \varepsilon \to a$
$b, \varepsilon \to b$
$a, a \to \varepsilon$
$b, b \to \varepsilon$
$\varepsilon, \varepsilon \to \varepsilon$
$\varepsilon, \$ \to \$$
q_0
q_1
q_2

Time 2

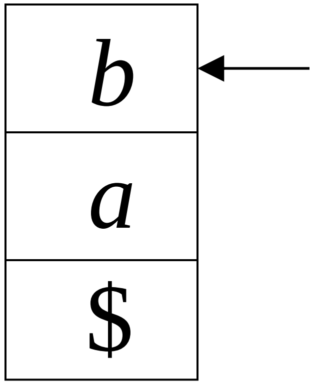

Input
a
b
b
b
b
a
$
Stack
$a, \varepsilon \rightarrow a$
$b, \varepsilon \rightarrow b$
$a, a \rightarrow \varepsilon$
$b, b \rightarrow \varepsilon$
$\varepsilon, \varepsilon \rightarrow \varepsilon$
$\varepsilon, \$ \rightarrow \$$
q_0
q_1
q_2

Time 3

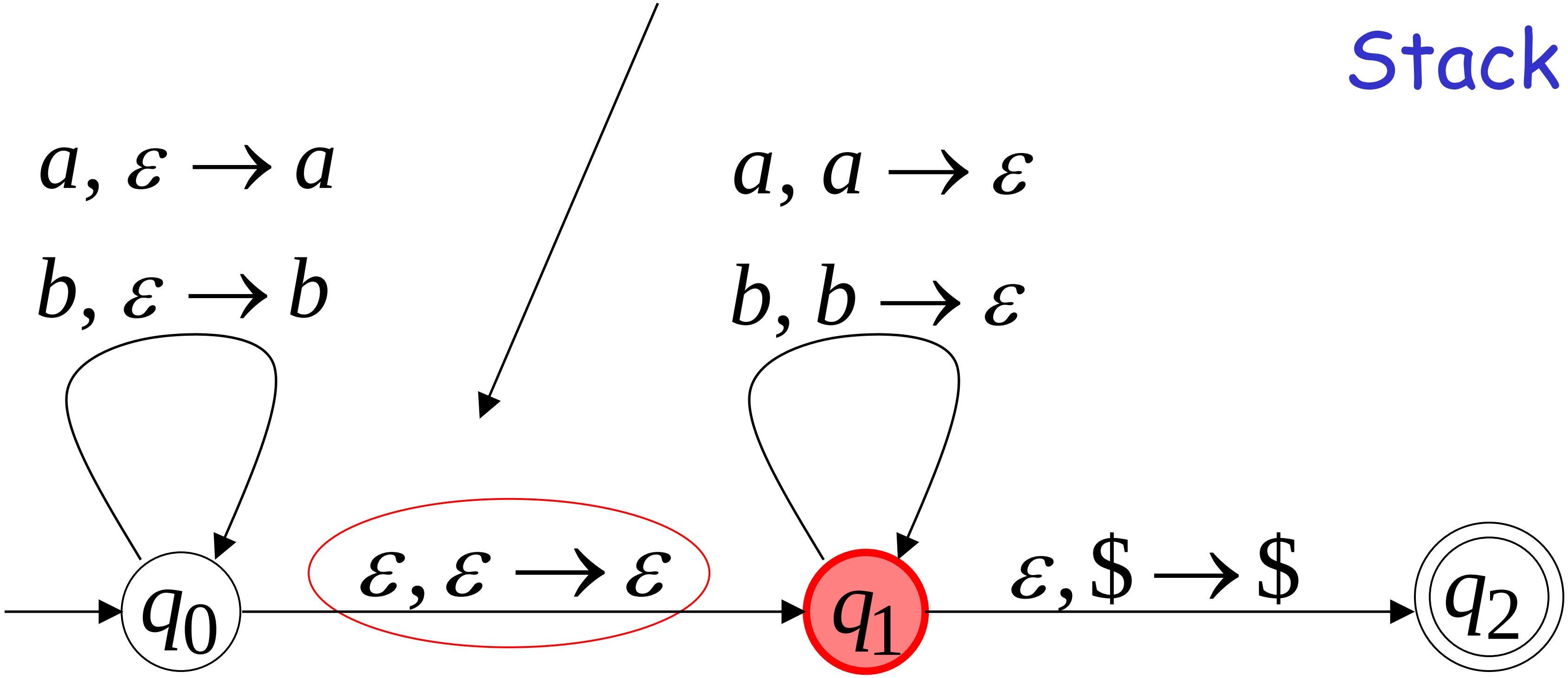
Input



Guess the middle of string

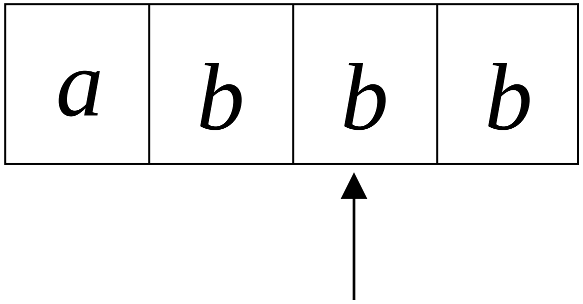


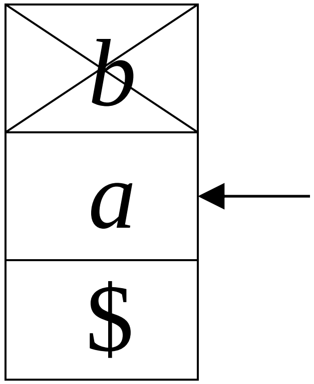
Stack



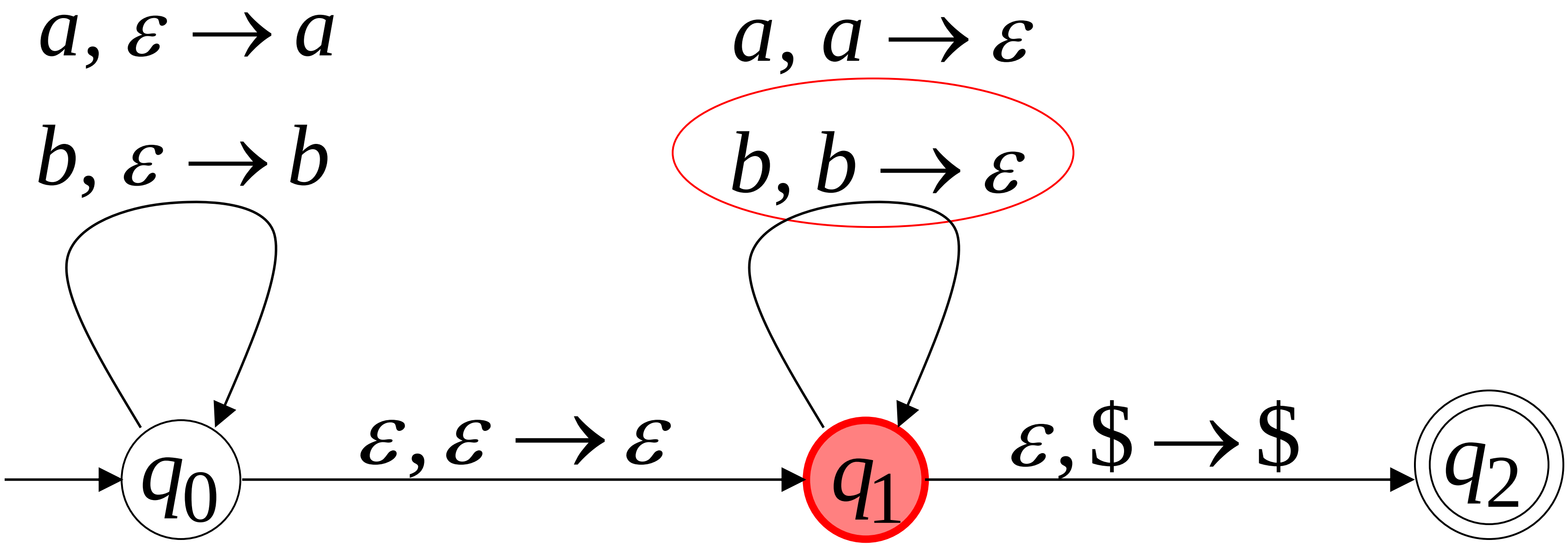
Time 4

Input





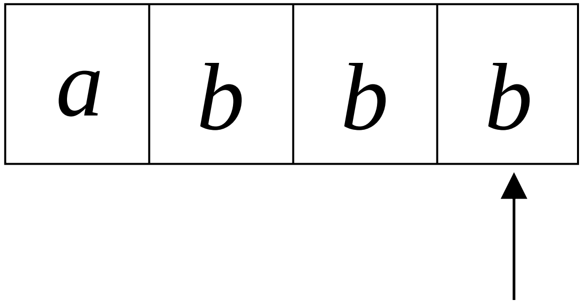
Stack



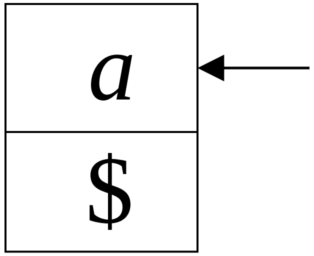
Time 5

There is no possible transition.

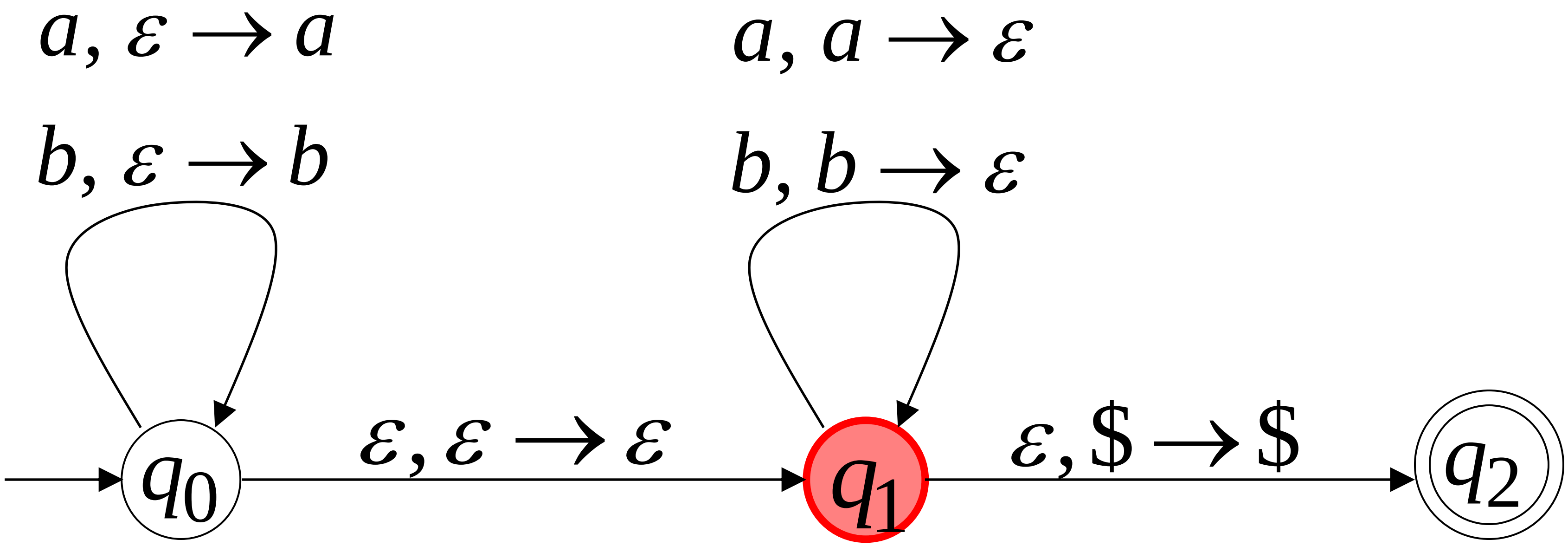
Input



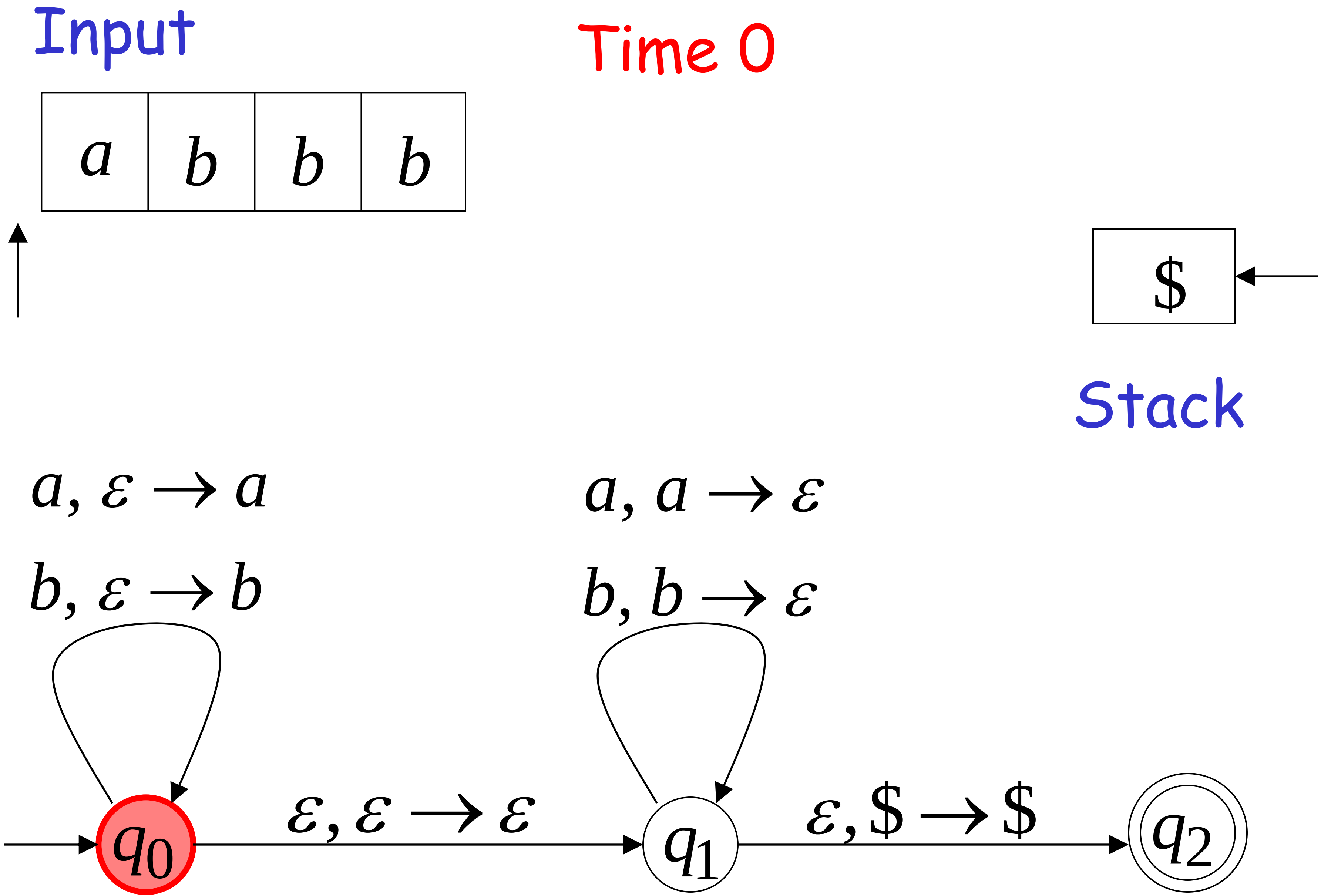
Input is not consumed



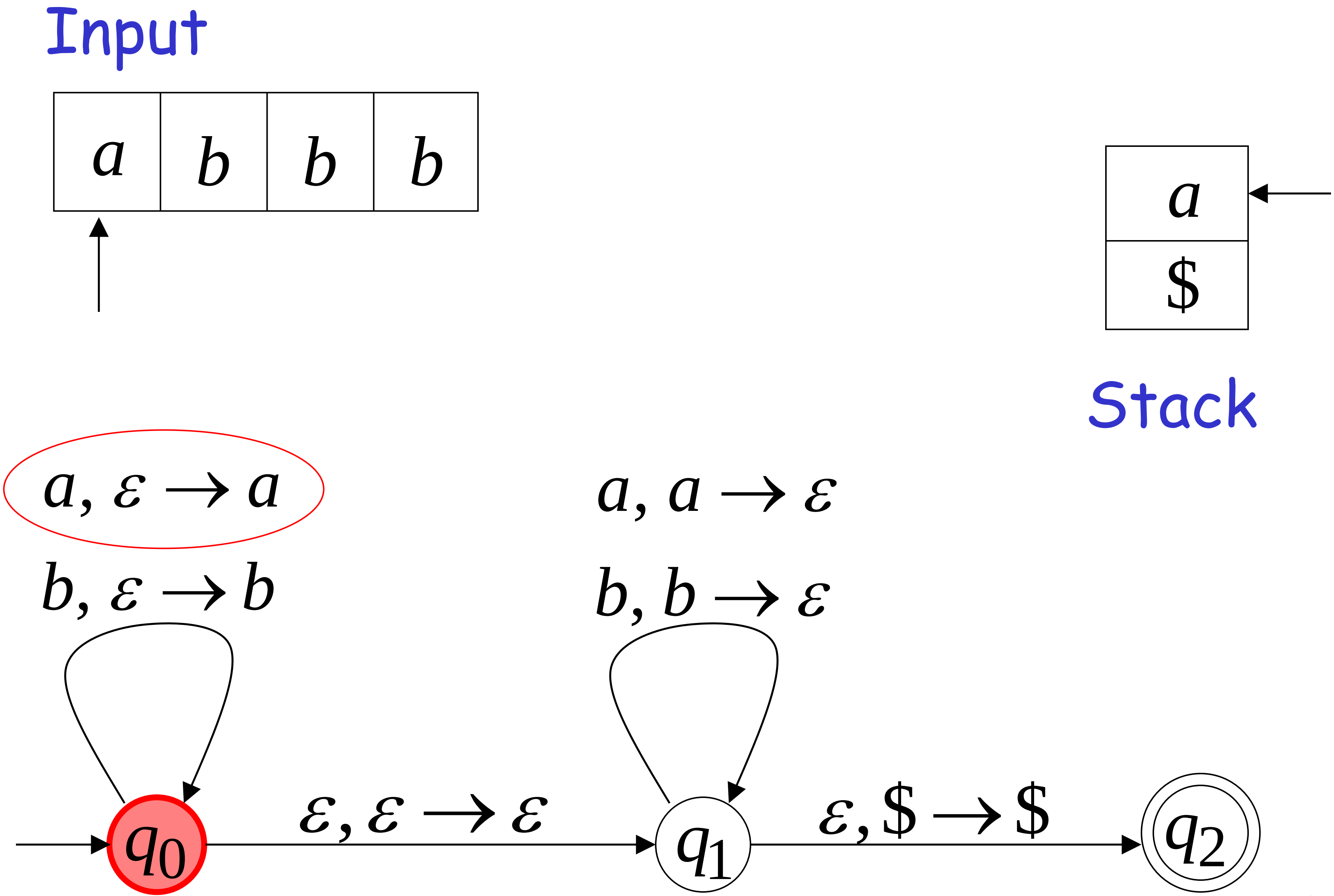
Stack



Another computation on same string:



Time 1



Input

a b b b

Stack

a

$\$$

$a, \varepsilon \rightarrow a$

$b, \varepsilon \rightarrow b$

$a, a \rightarrow \varepsilon$

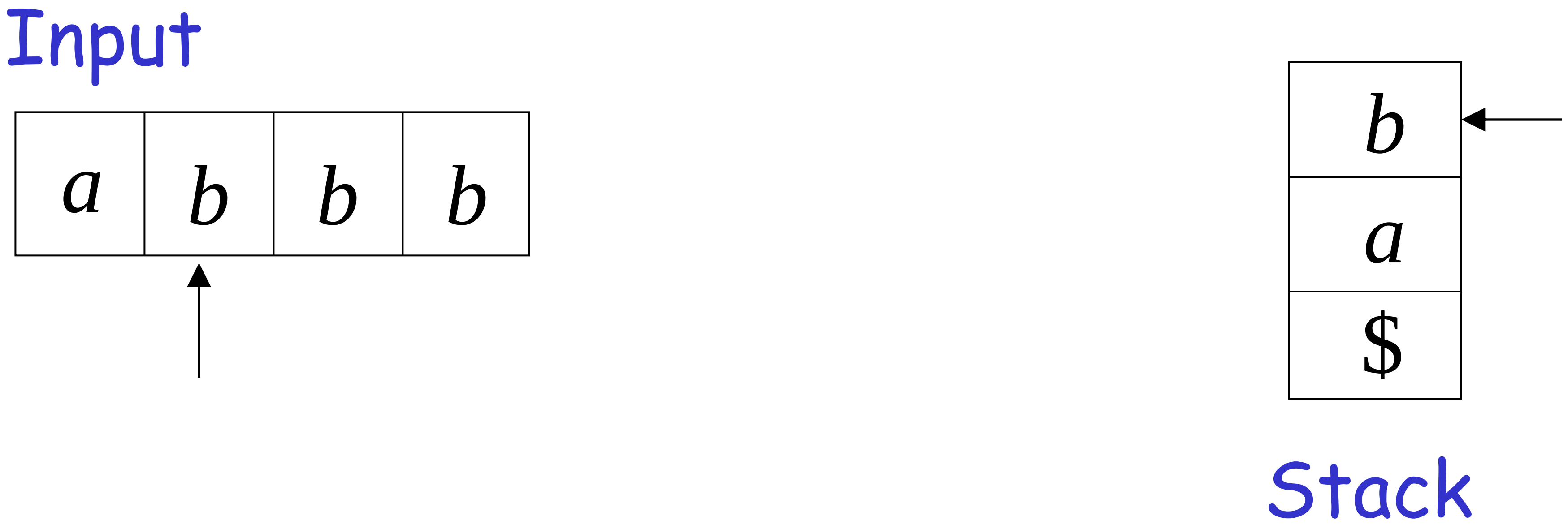
$b, b \rightarrow \varepsilon$

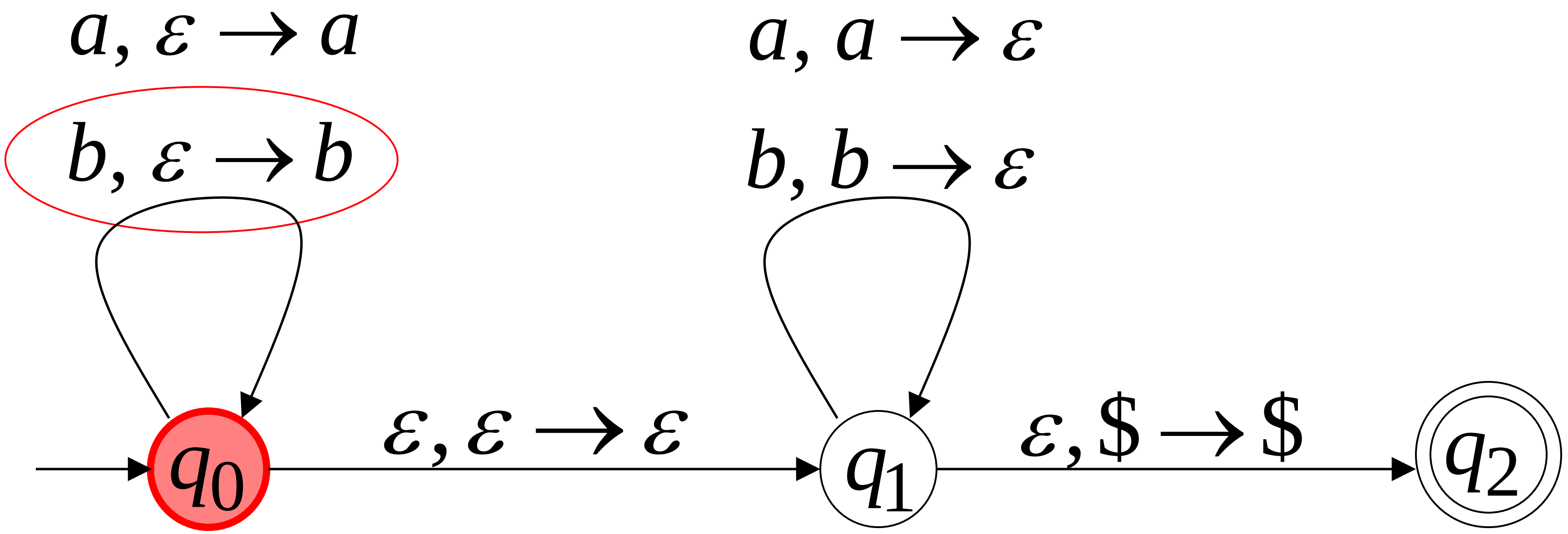
$\varepsilon, \varepsilon \rightarrow \varepsilon$

$\varepsilon, \$ \rightarrow \$$

q_0 q_1 q_2

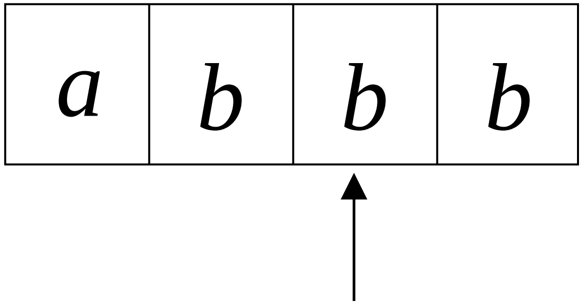
Time 2

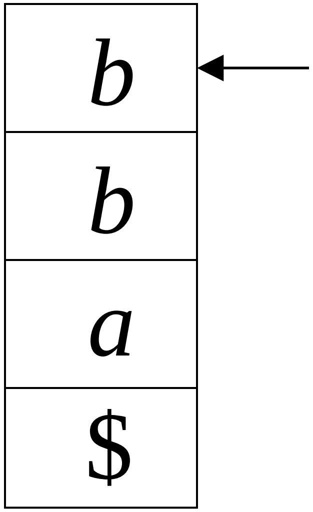

Input
a
b
b
b
b
a
$\$$
Stack


$a, \varepsilon \rightarrow a$
$b, \varepsilon \rightarrow b$
$a, a \rightarrow \varepsilon$
$b, b \rightarrow \varepsilon$
$\varepsilon, \varepsilon \rightarrow \varepsilon$
$\varepsilon, \$ \rightarrow \$$
q_0
q_1
q_2

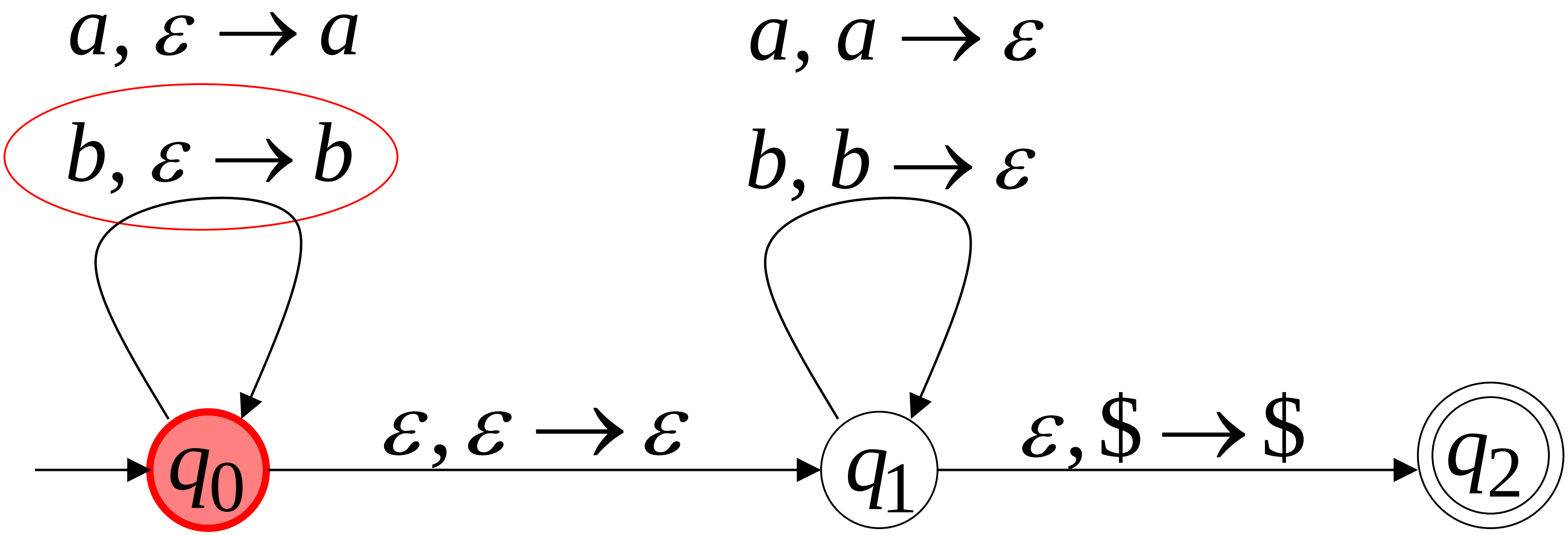
Time 3

Input



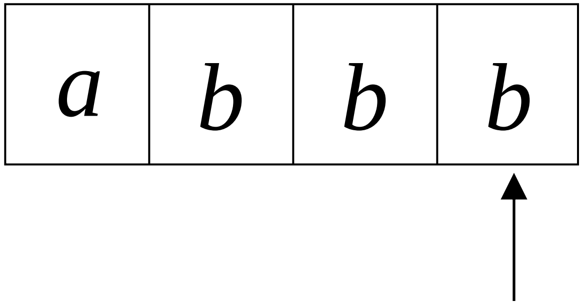


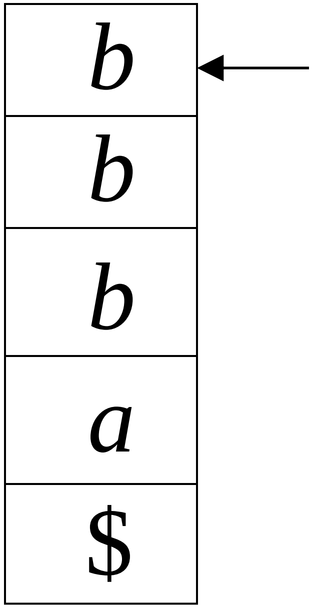
Stack



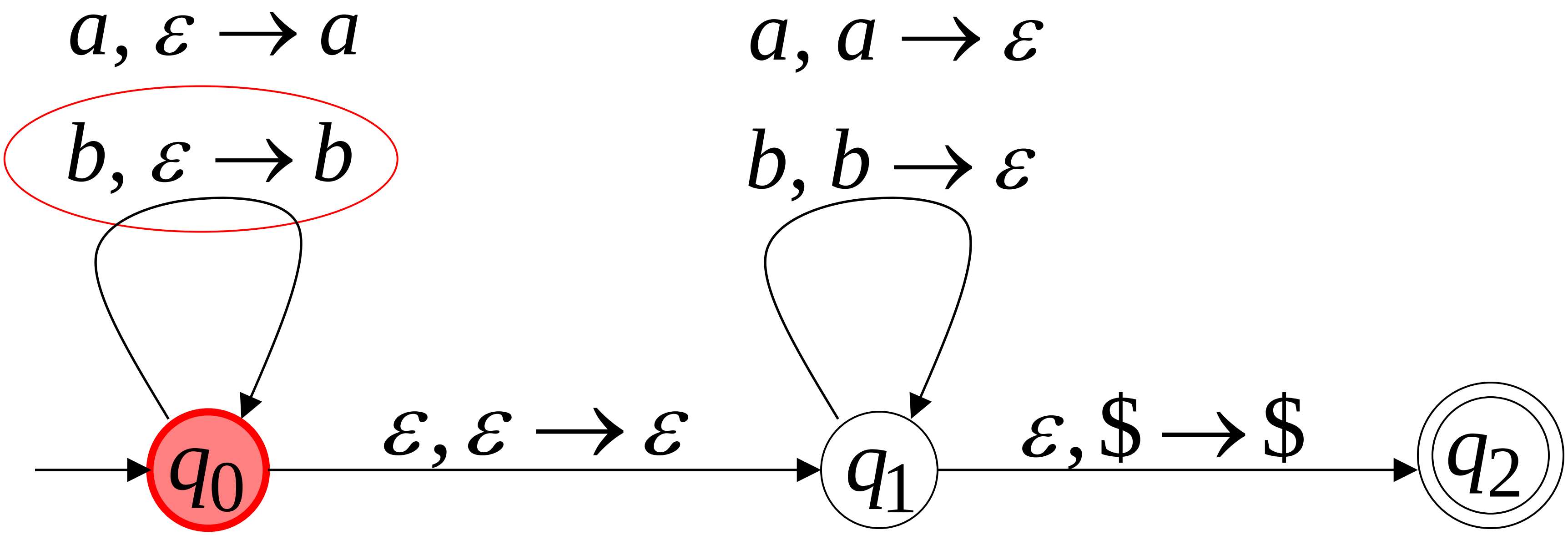
Time 4

Input



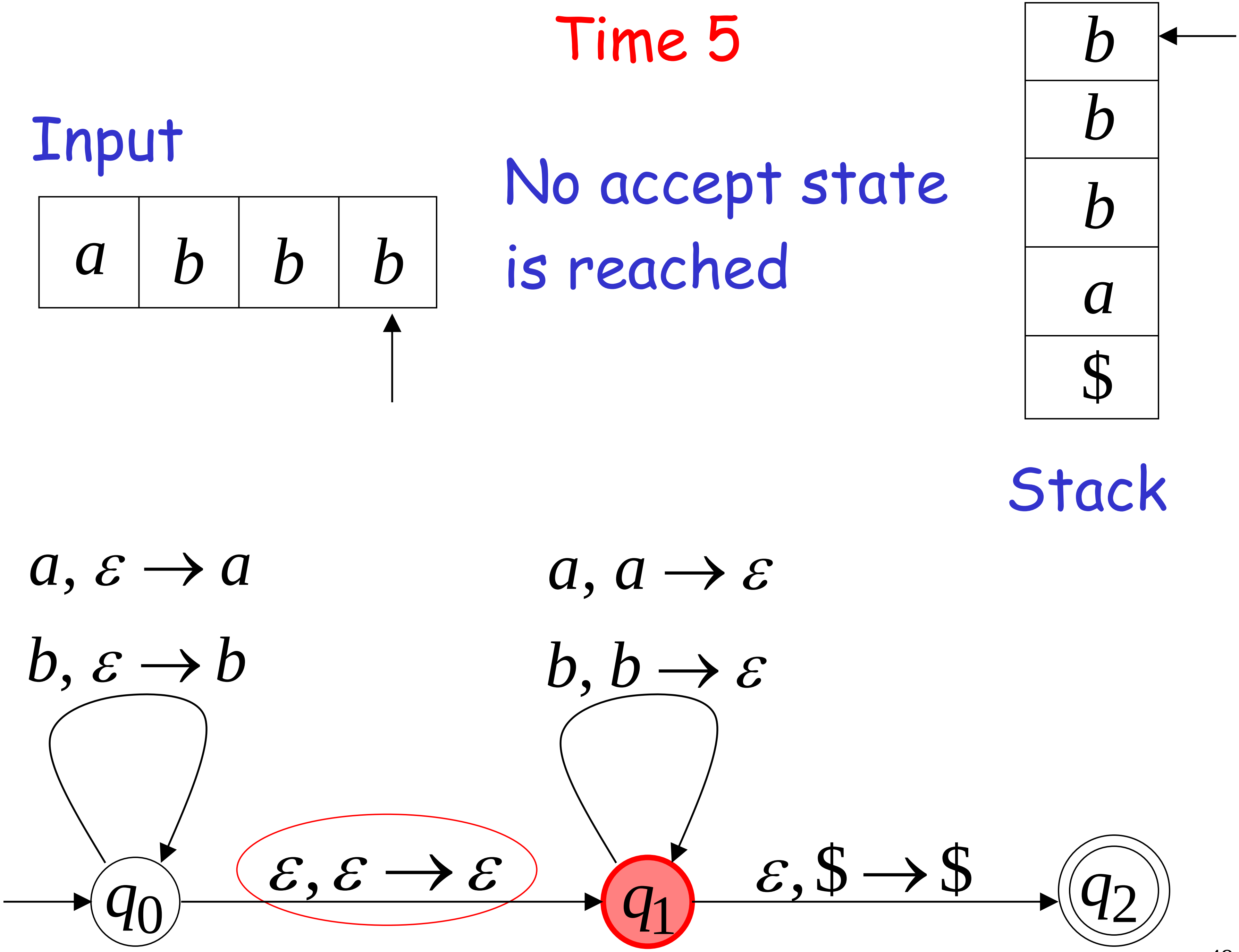


Stack



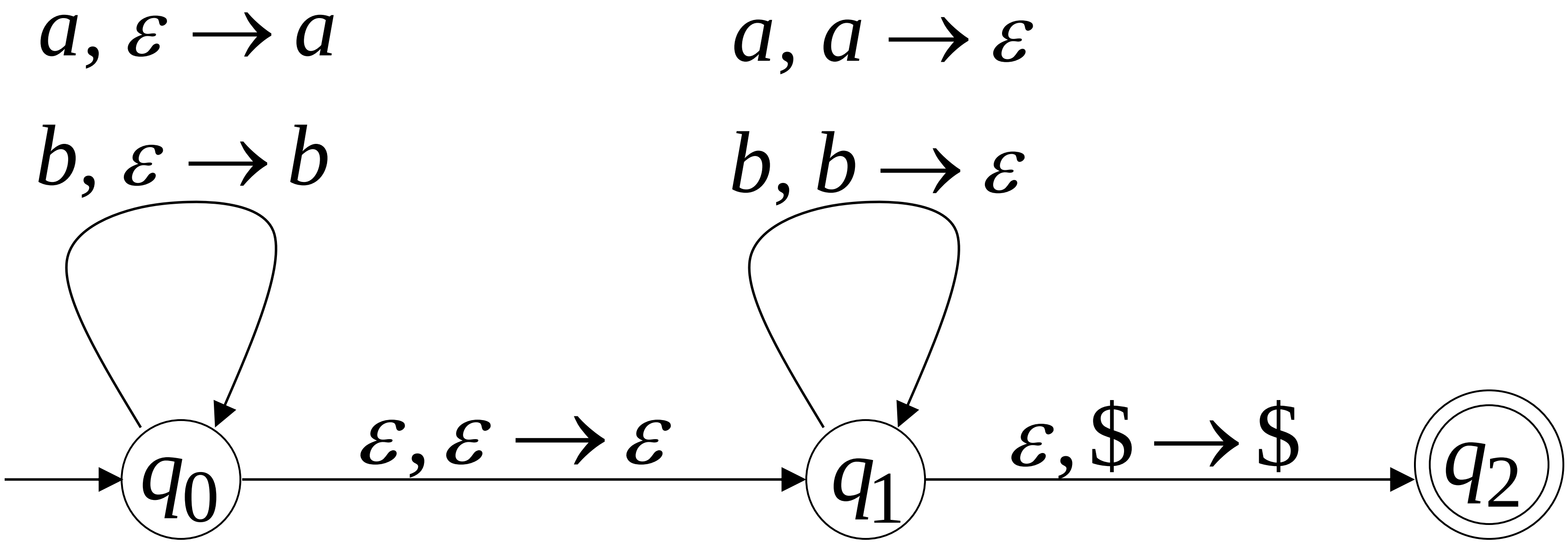
Time 5

No accept state is reached

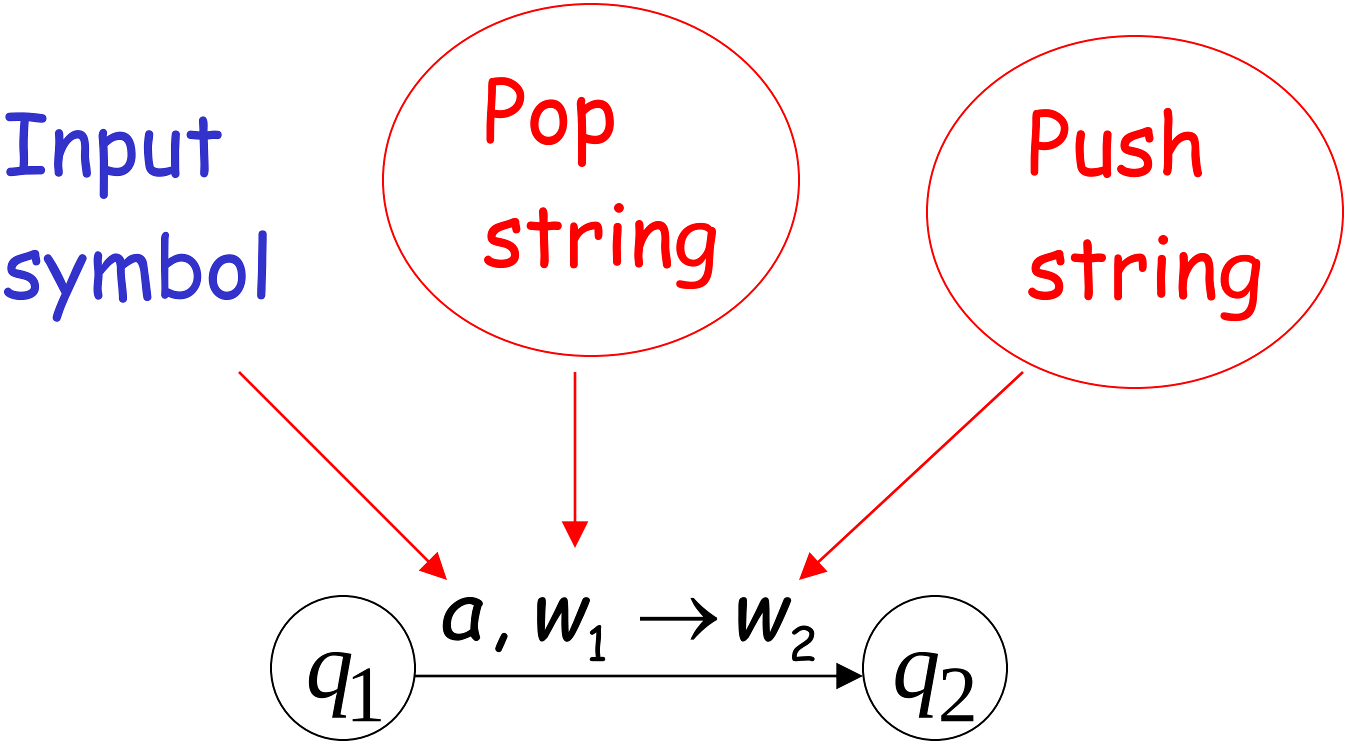


There is no computation
that accepts string $abbb$

$$abbb \notin L(M)$$



Pushing & Popping Strings



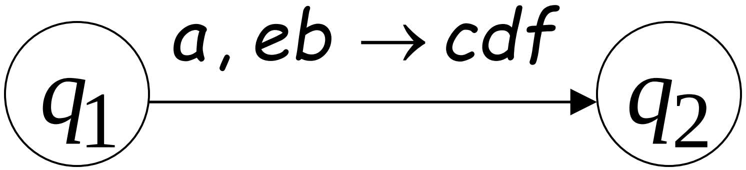
Input symbol

Pop string

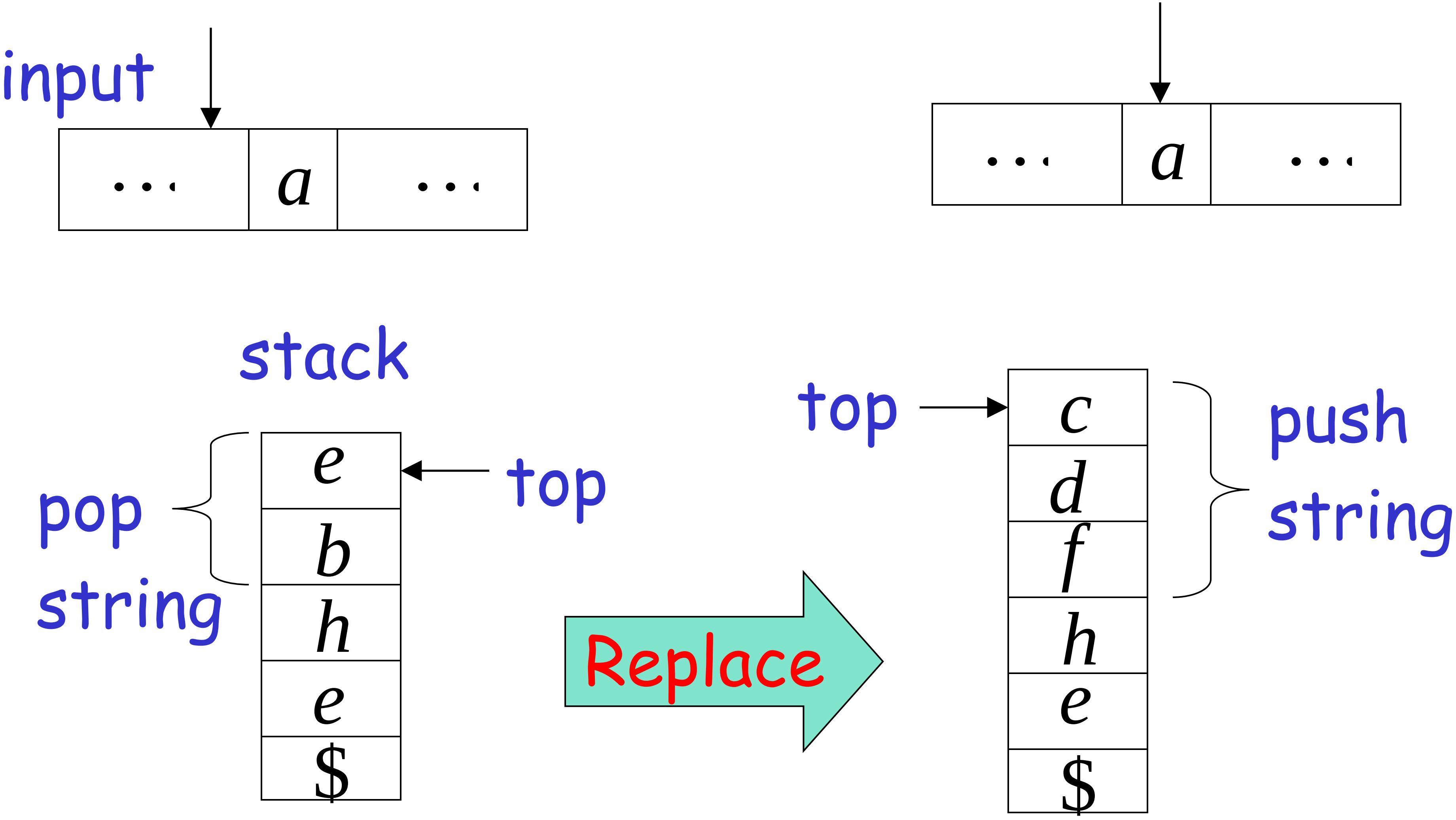
Push string

$a, w_1 \to w_2$

Example:



$q_1 \xrightarrow{a,\, eb \to cdf} q_2$



input

stack

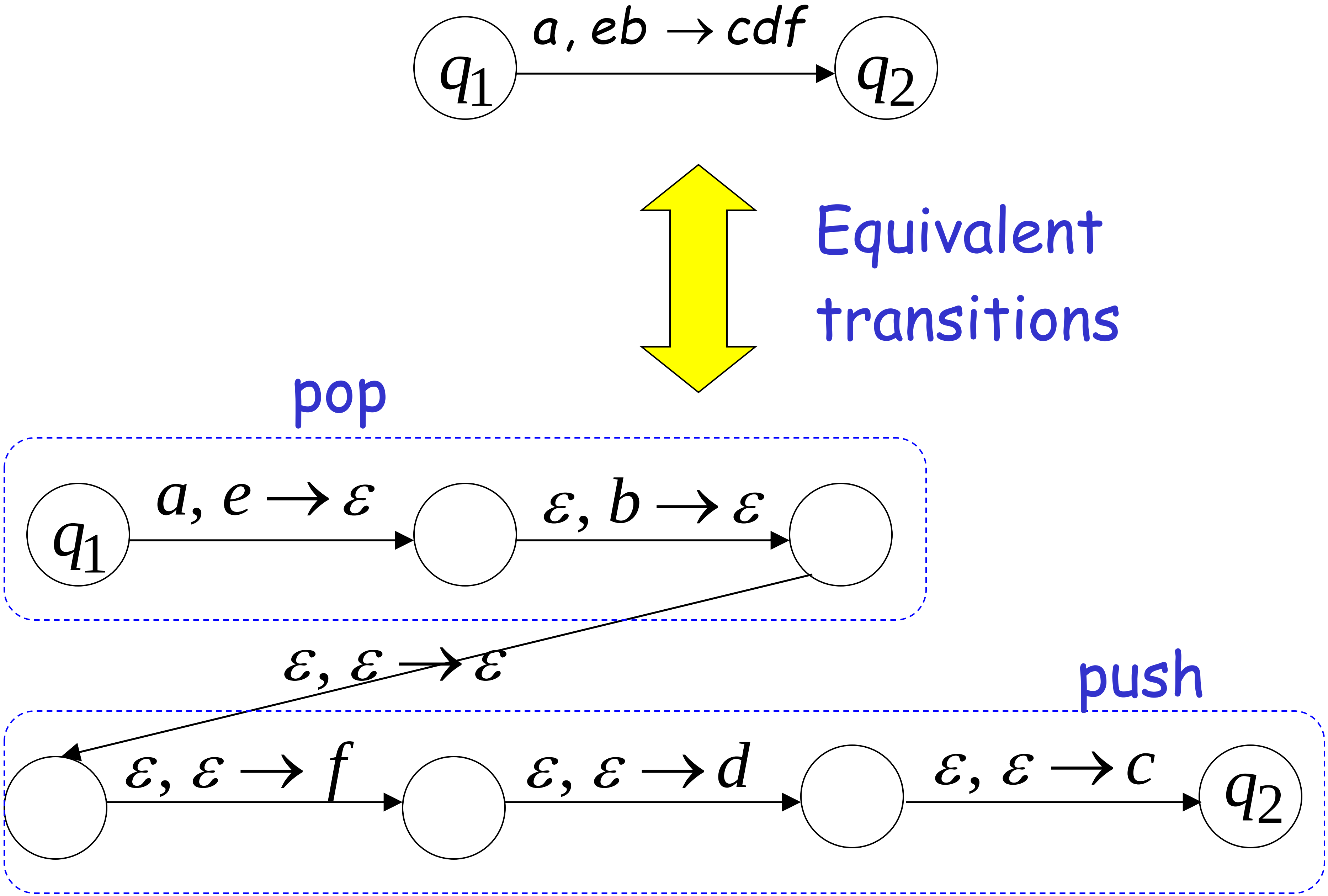
pop string

top

Replace

top

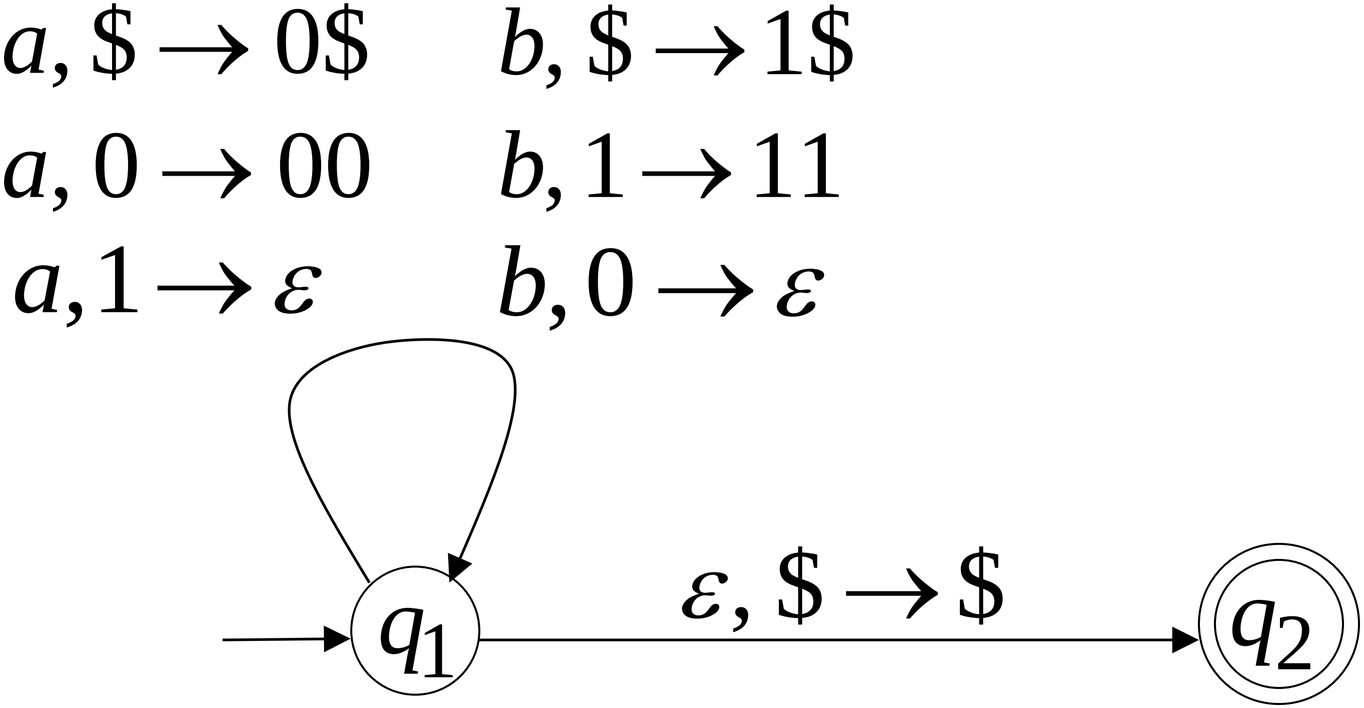
push string


q_1
$a, eb \to cdf$
q_2
Equivalent transitions
pop
q_1
$a, e \to \varepsilon$
$\varepsilon, b \to \varepsilon$
$\varepsilon, \varepsilon \to \varepsilon$
push
$\varepsilon, \varepsilon \to f$
$\varepsilon, \varepsilon \to d$
$\varepsilon, \varepsilon \to c$
q_2

Another PDA example

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

PDA M



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

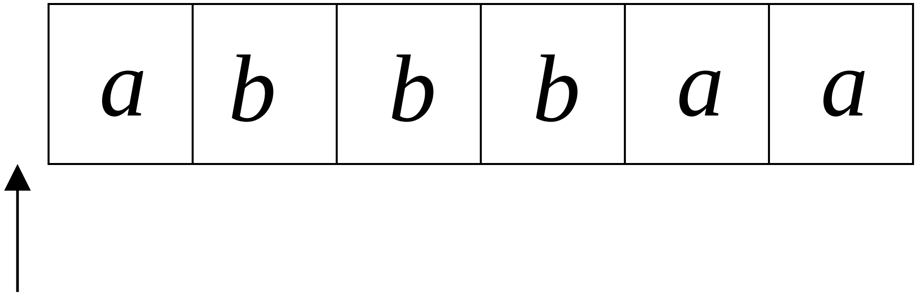
$$a, 0 \rightarrow 00 \qquad b, 1 \rightarrow 11$$

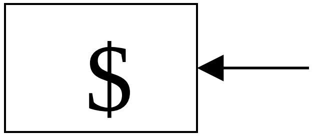
$$a, 1 \rightarrow \varepsilon \qquad b, 0 \rightarrow \varepsilon$$

$$\varepsilon, \$ \rightarrow \$$$

Execution Example: Time 0

Input





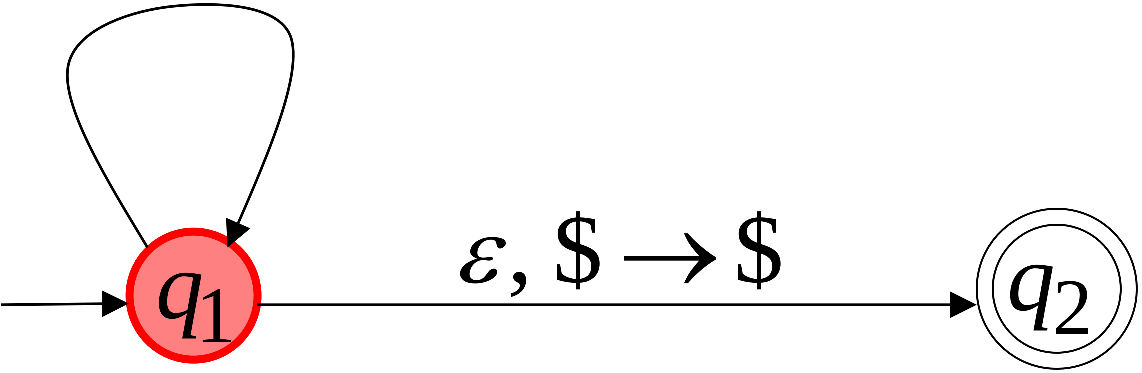
Stack

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

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

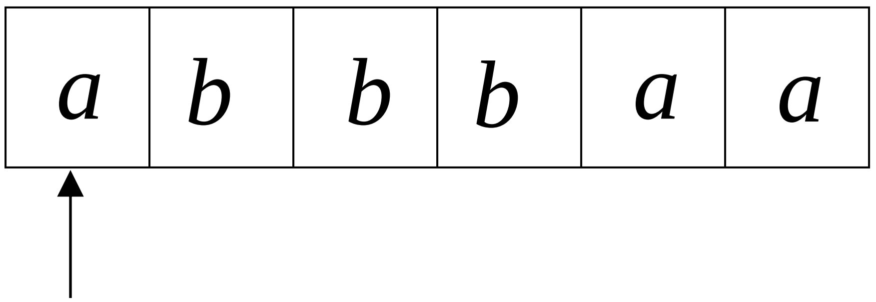
$a, 1 \rightarrow \varepsilon$ $\quad$ $b, 0 \rightarrow \varepsilon$

current state

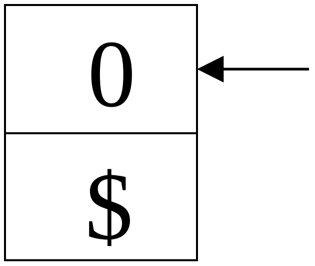


Time 1

Input



a	b	b	b	a	a



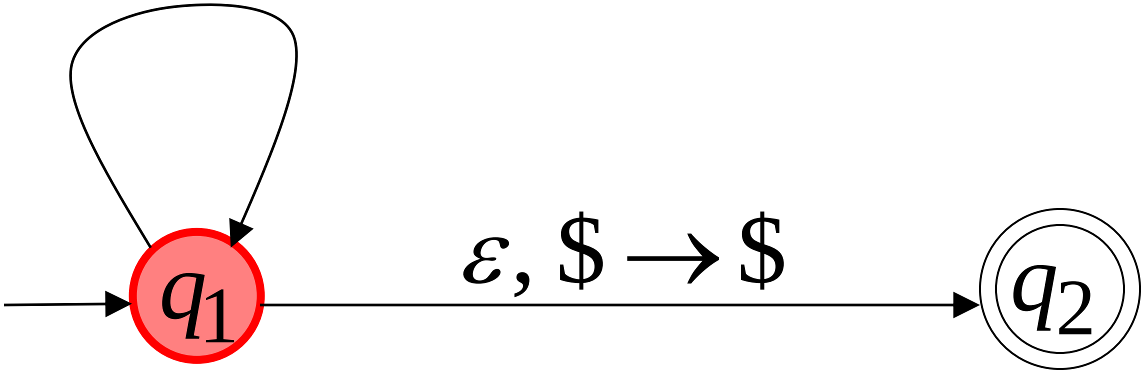
0
\$

Stack

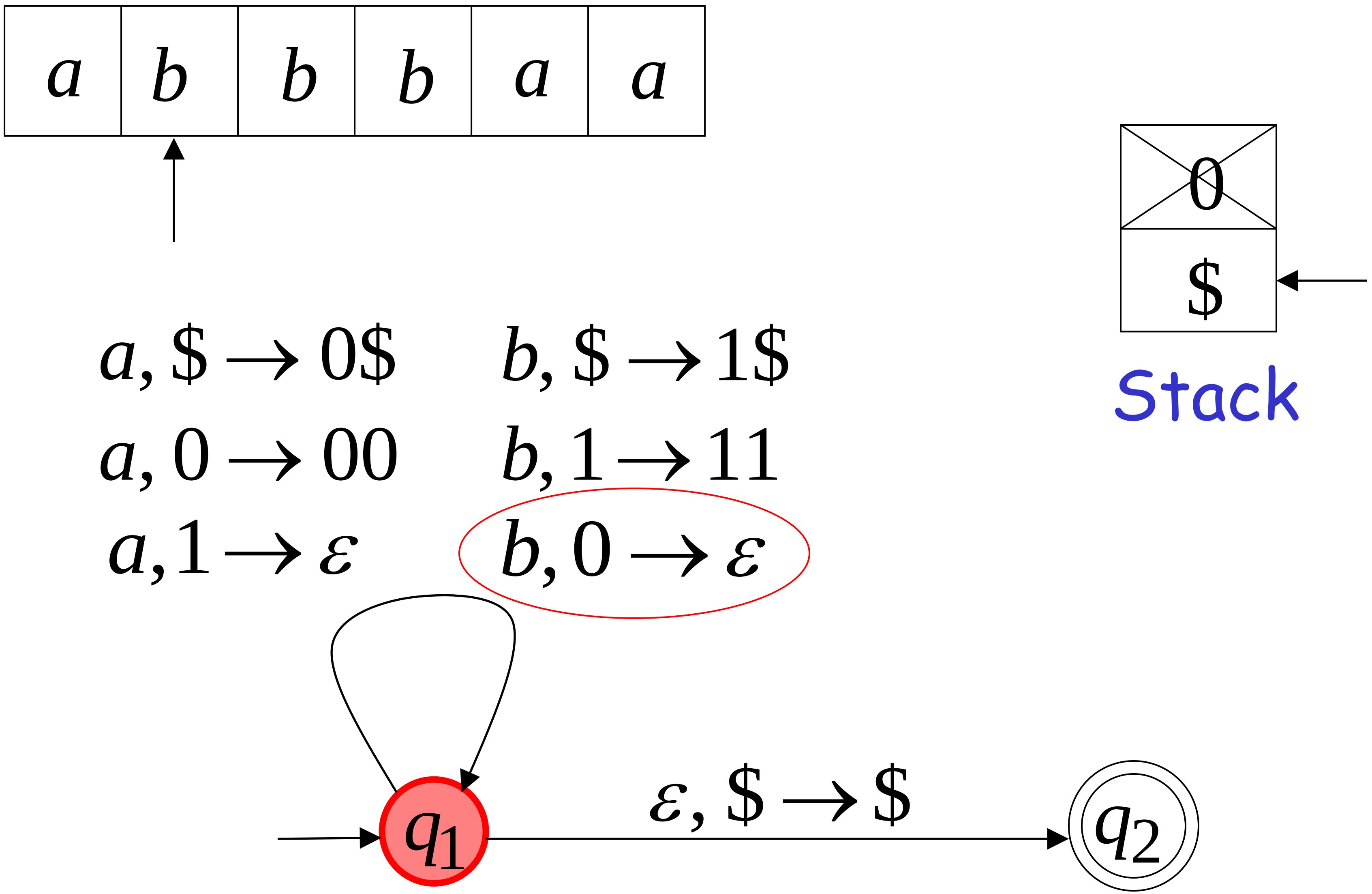
$a, \$ \rightarrow 0\$$ $\quad$ $b, \$ \rightarrow 1\$$

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

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



Time 3



Input

a	b	b	b	a	a

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

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

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

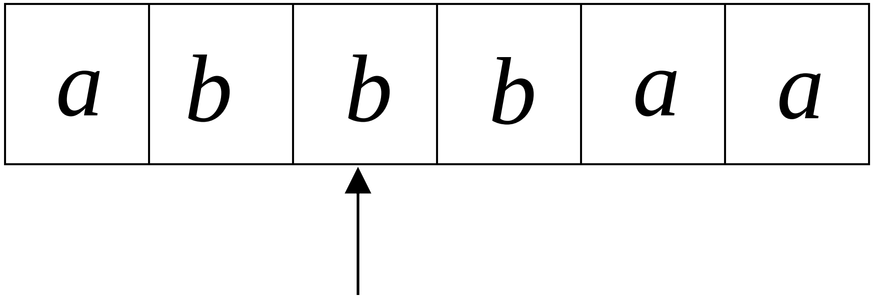
$\xcancel{0}$

$\$$

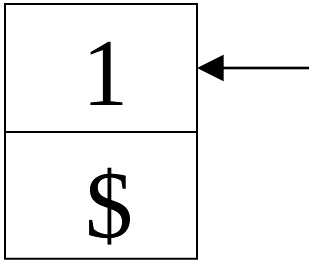
Stack

Time 4

Input



a	b	b	b	a	a



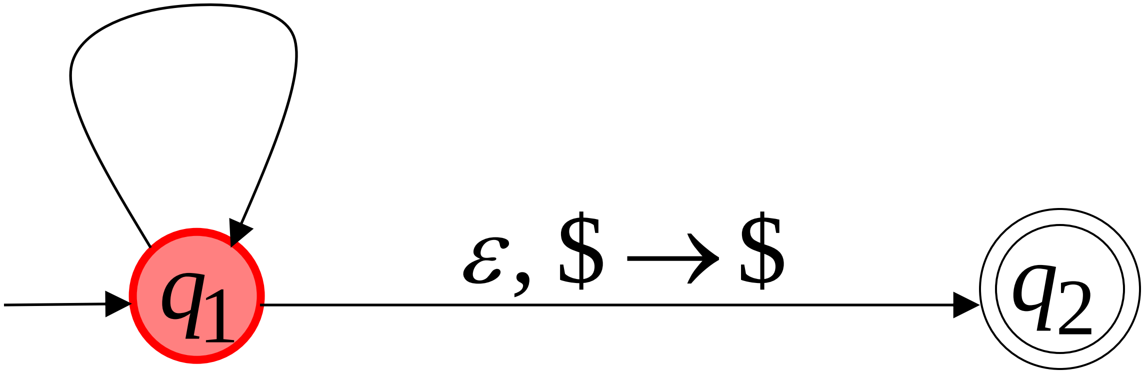
1
$\$$

Stack

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

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

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



$$\varepsilon, \$ \rightarrow \$$$

Time 5

Input

a	b	b	b	a	a

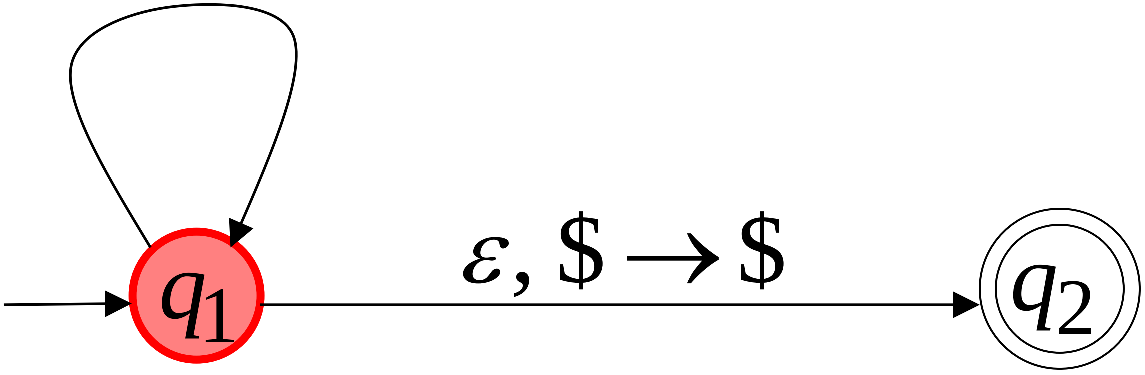
1
1
$

Stack

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

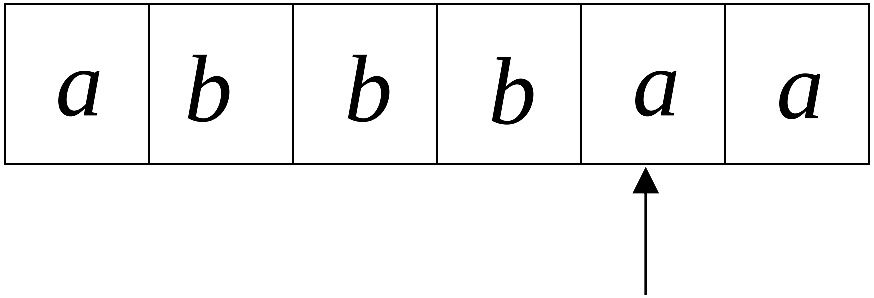
$$a, 0 \rightarrow 00 \qquad b, 1 \rightarrow 11$$

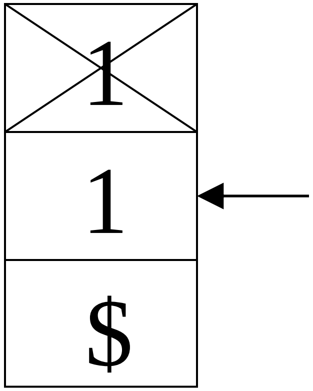
$$a, 1 \rightarrow \varepsilon \qquad b, 0 \rightarrow \varepsilon$$



Time 6

Input



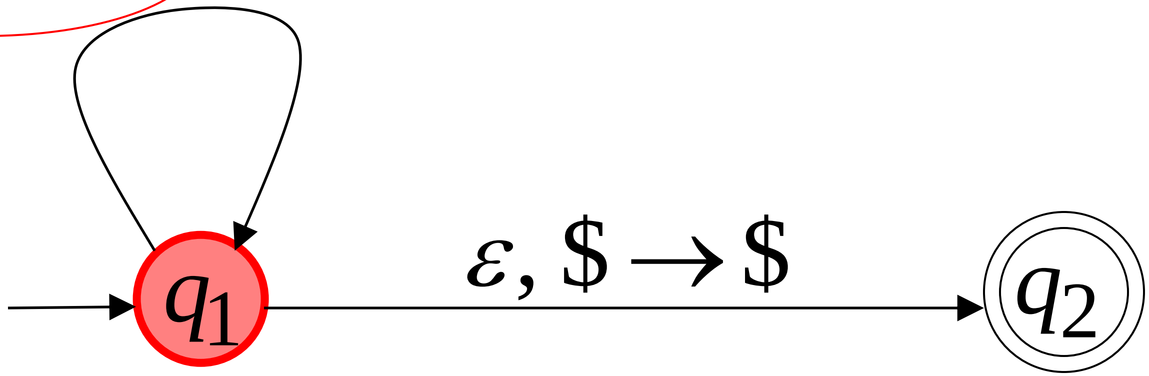


Stack

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

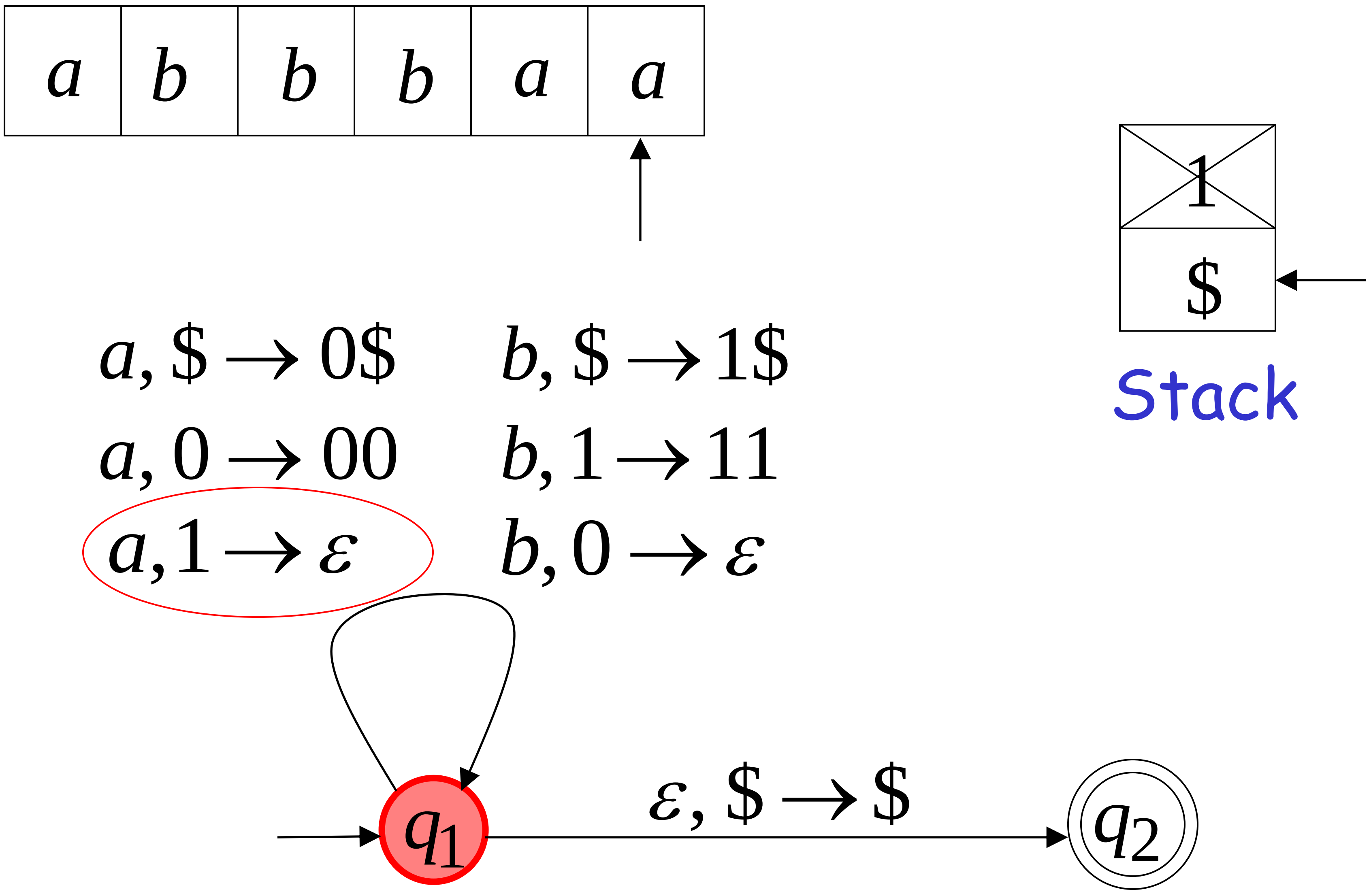
$a, 0 \rightarrow 00$ $\quad$ $b, 1 \rightarrow 11$

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



Time 7

Input



a	b	b	b	a	a

$\cancel{1}$

$\$$

Stack

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

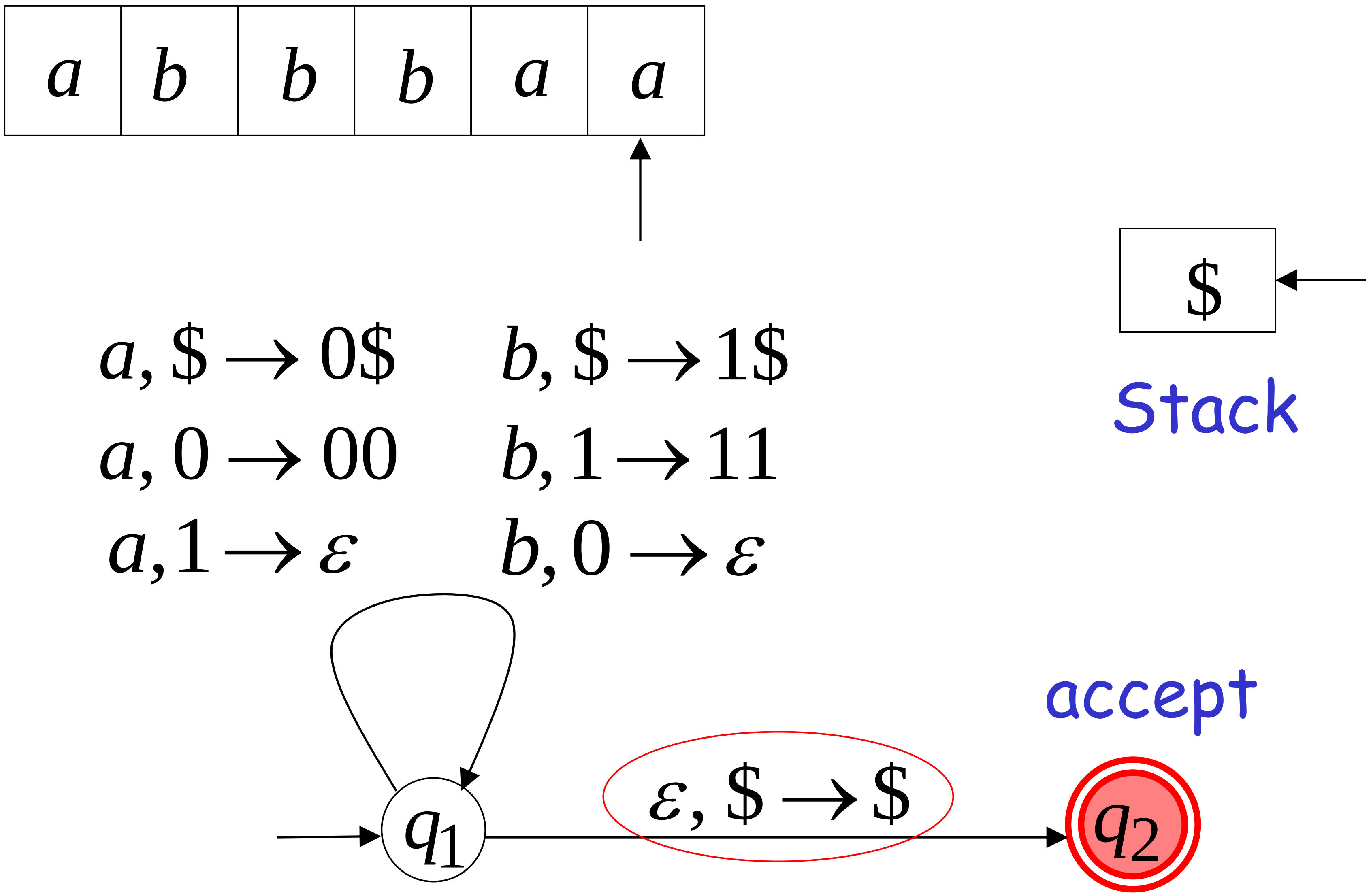
$a, 0 \rightarrow 00$ $\quad$ $b, 1 \rightarrow 11$

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

$\varepsilon, \$ \rightarrow \$$

Time 8

Input



a	b	b	b	a	a

$\$$

Stack

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

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

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

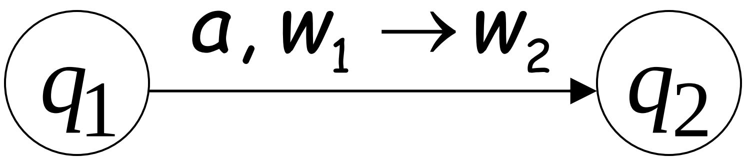
$\varepsilon, \$ \rightarrow \$$

q_1

q_2

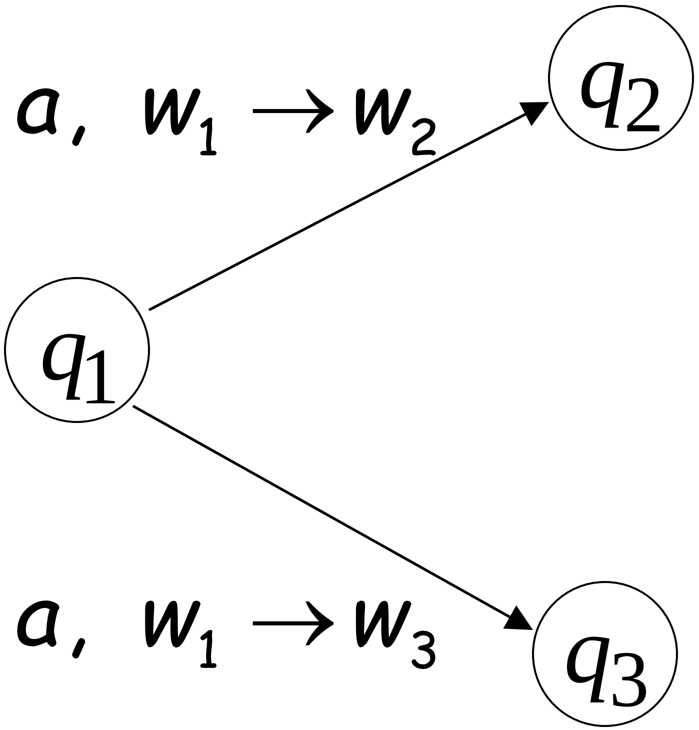
accept

Formalities for PDAs



Transition function:

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



Transition function:

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

Formal Definition

Pushdown Automaton (PDA)

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

States

Input alphabet

Stack alphabet

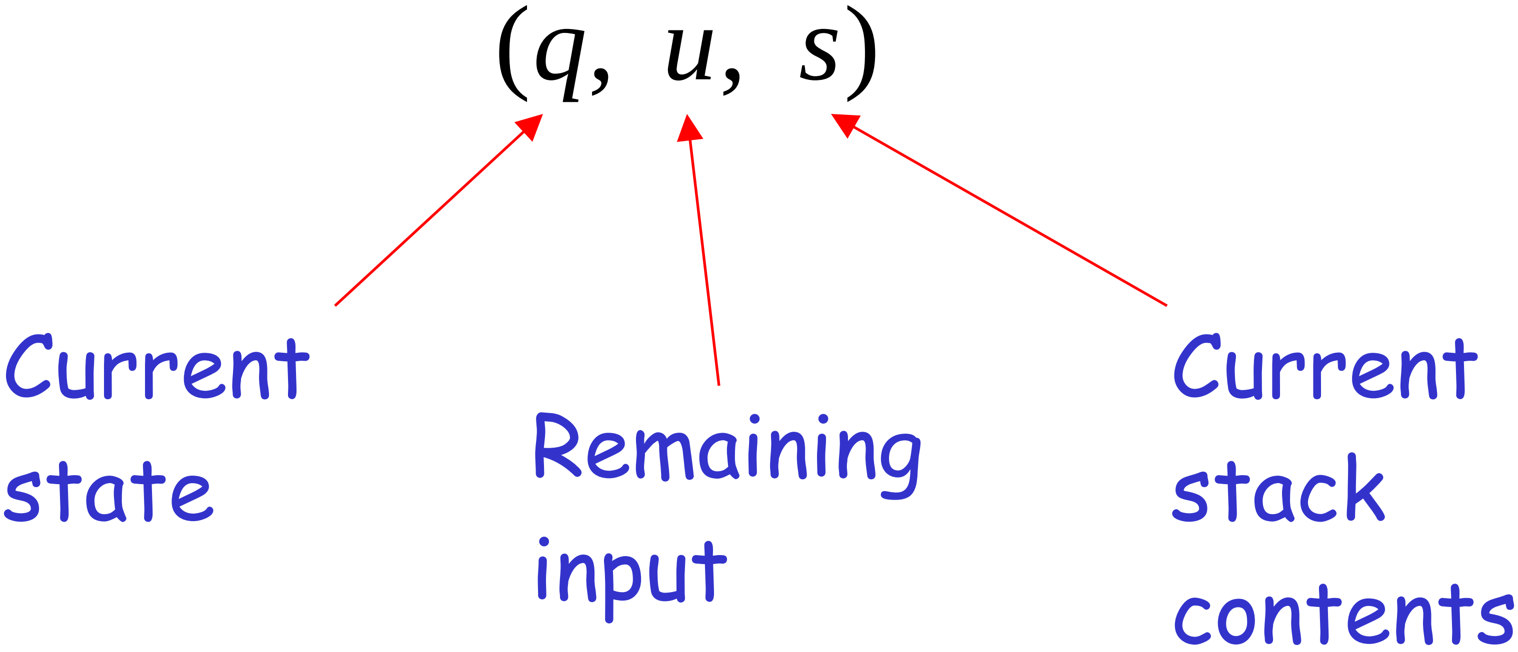
Transition function

Initial state

Stack start symbol

Accept states

Instantaneous Description



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

Current state

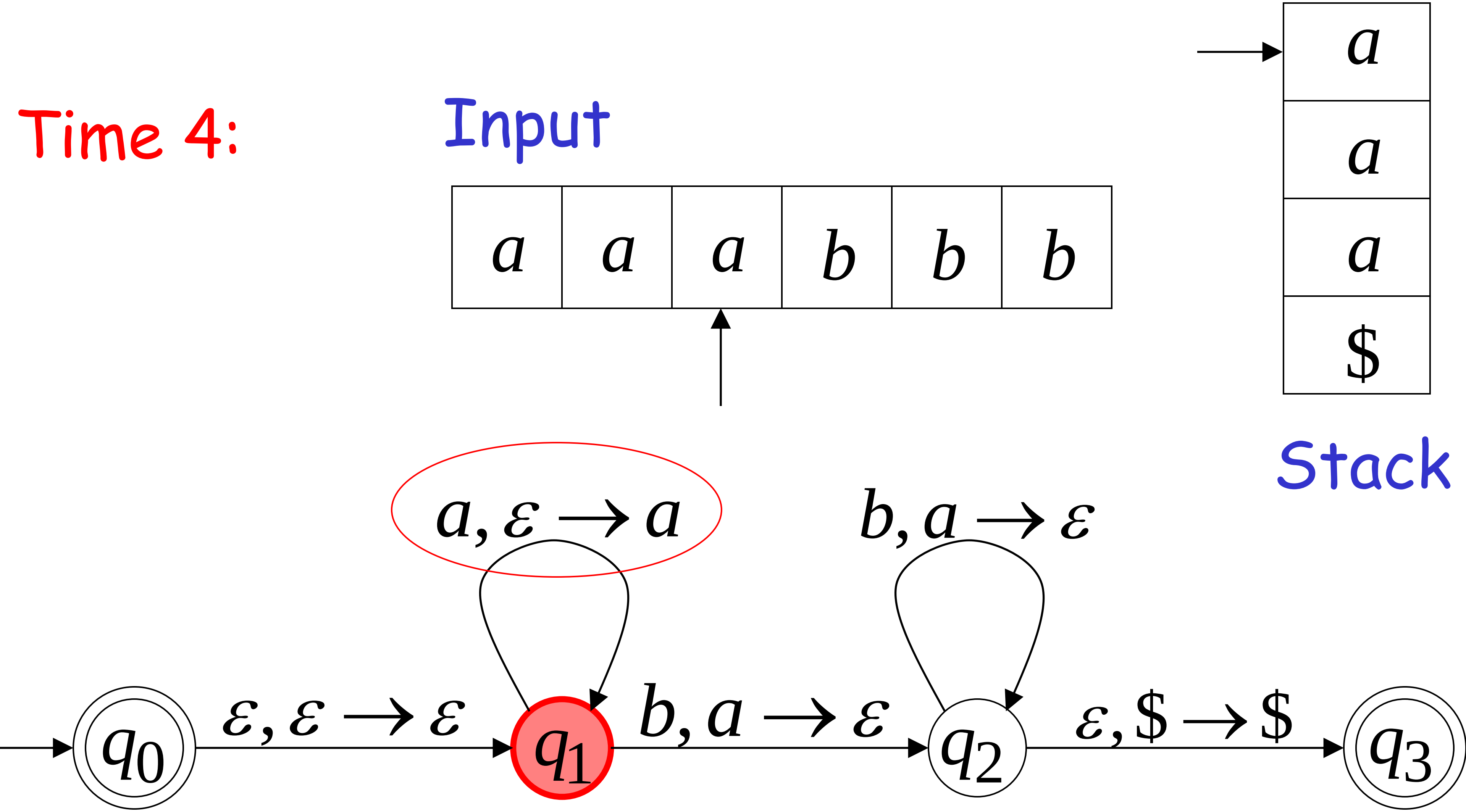
Remaining input

Current stack contents

Example: Instantaneous Description

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

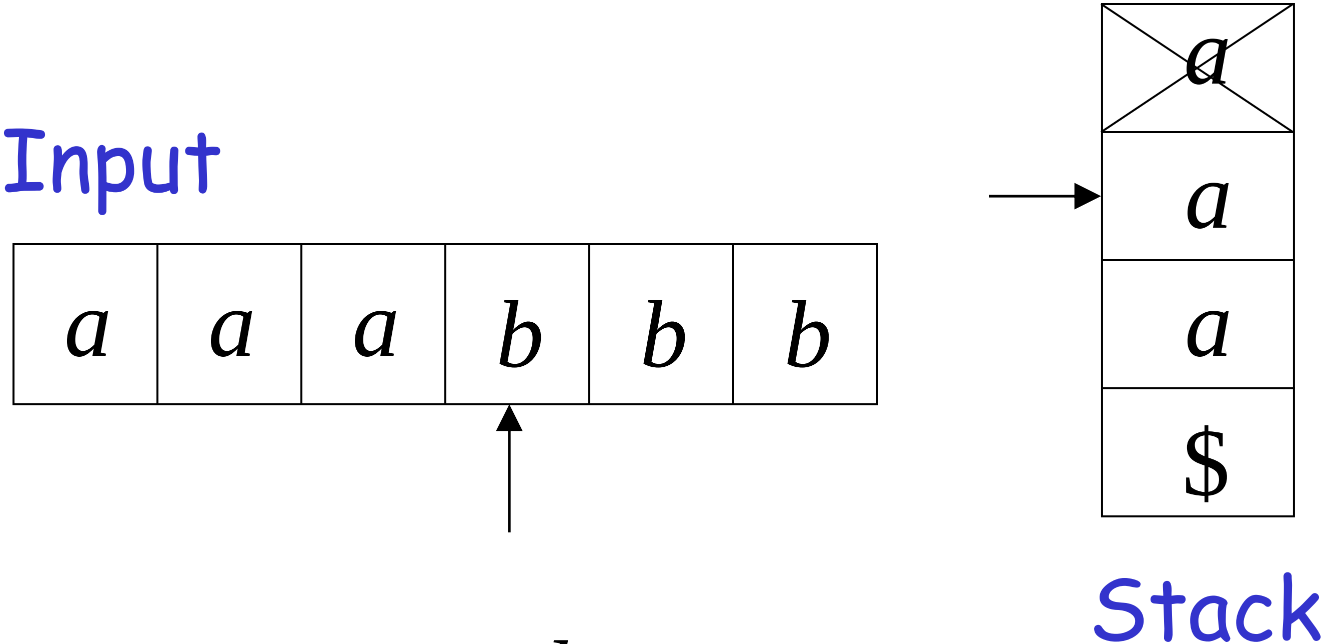
Time 4:

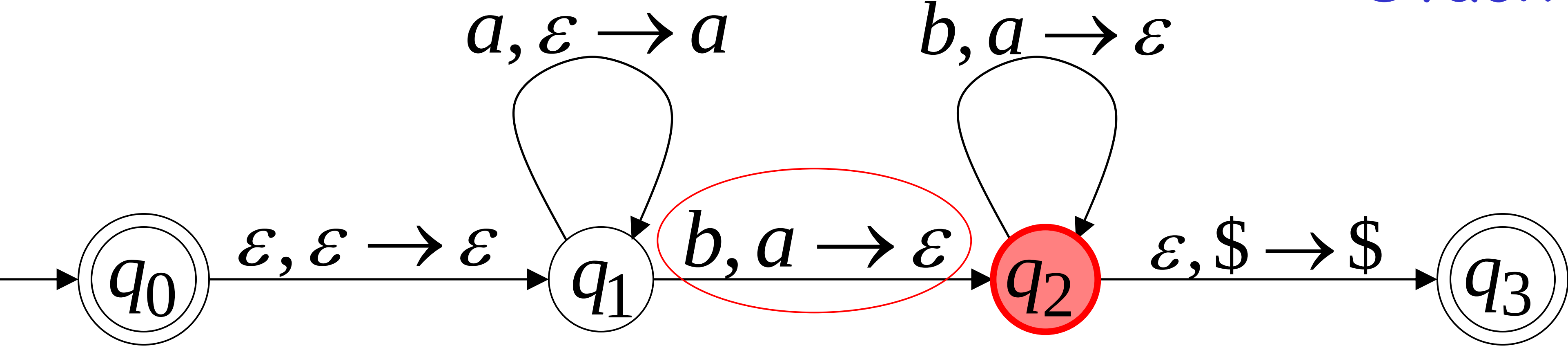


Example: Instantaneous Description

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

Time 5:





We write:

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

Time 4

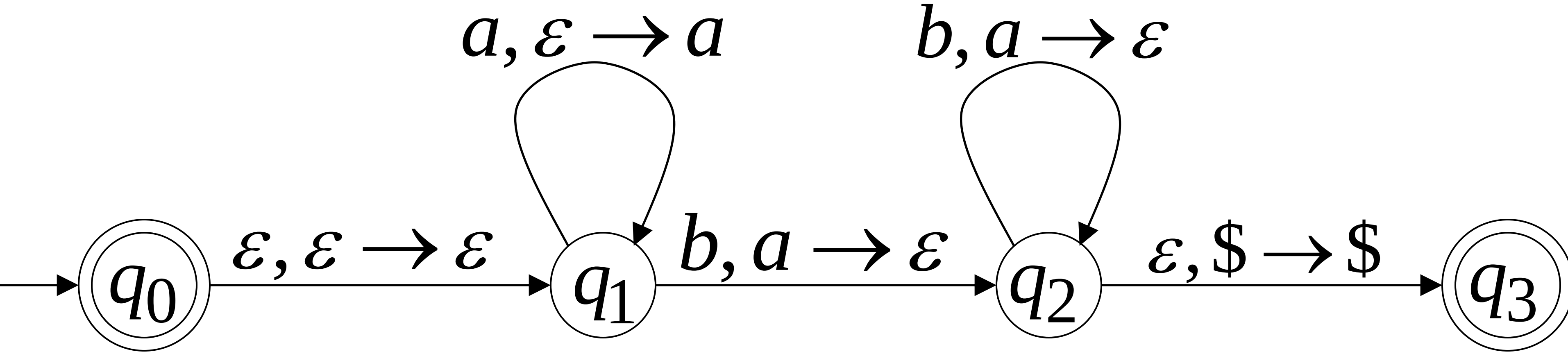
Time 5

A computation:

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

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

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



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

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

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

For convenience we write:

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

Language of PDA

Language $L(M)$ accepted by PDA M :

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

Initial state

Accept state

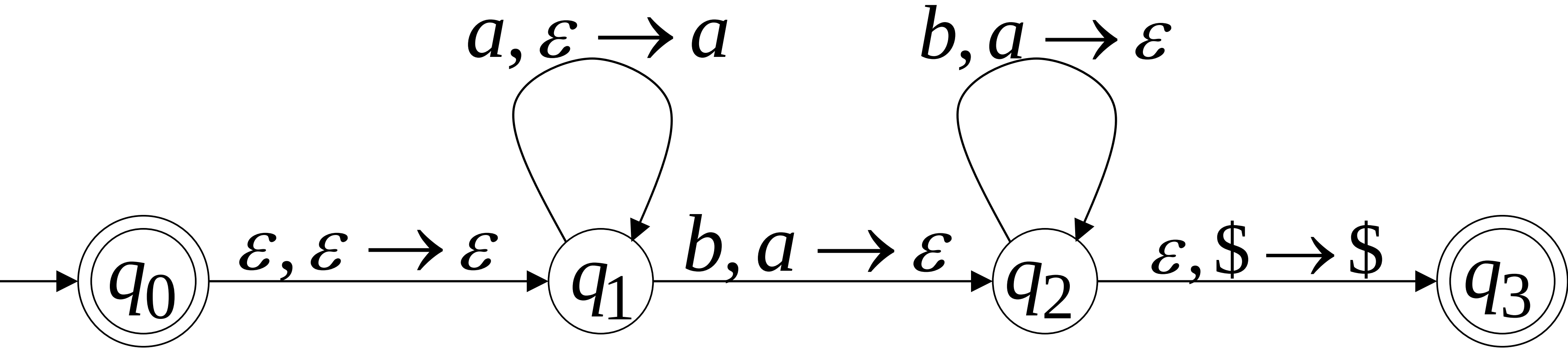
Example:

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

$$\Downarrow$$

$$aaabbb \in L(M)$$

PDA M:

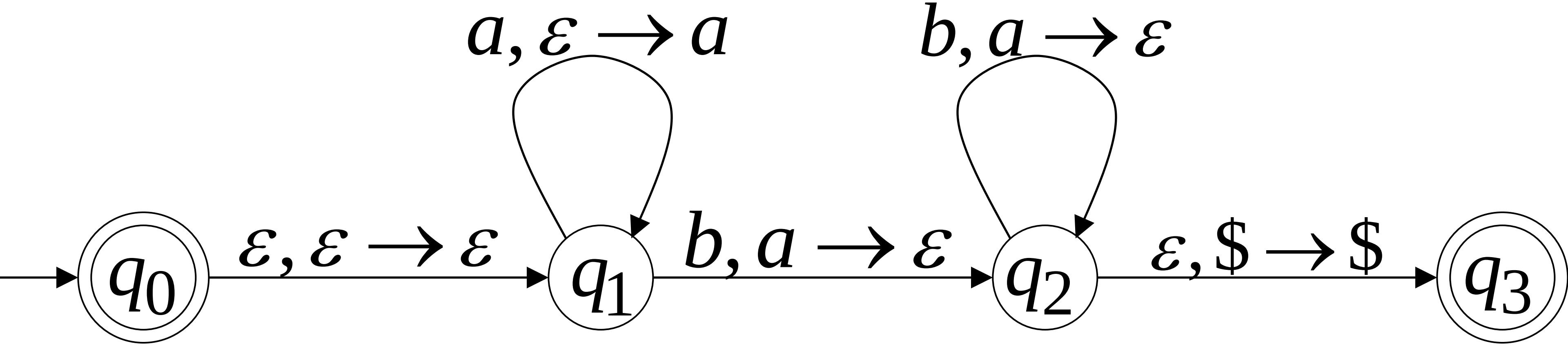


$$(q_0, a^n b^n, \$) \overset{*}{\succ} (q_3, \varepsilon, \$)$$

$$\Downarrow$$

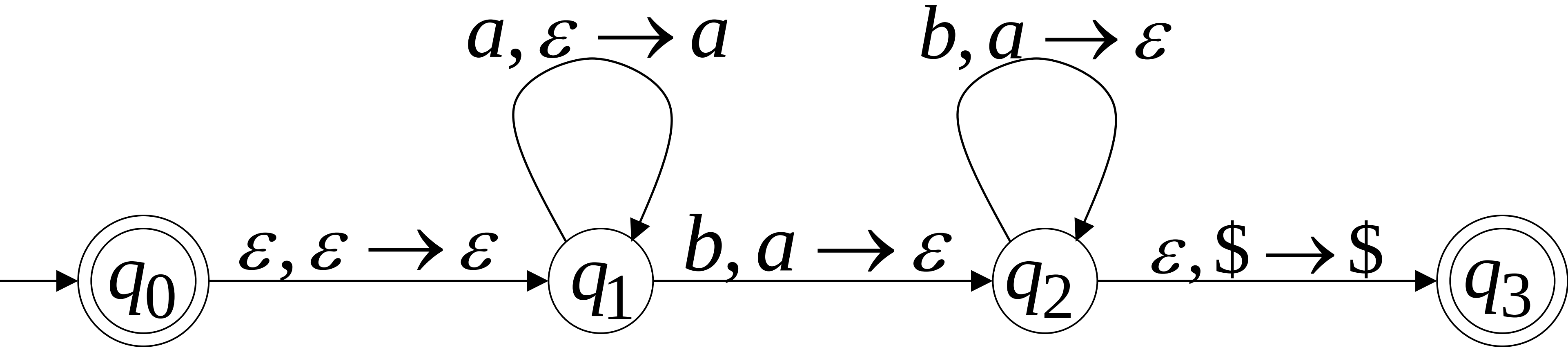
$$a^n b^n \in L(M)$$

PDA M :



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

PDA M :



PDAs Accept Context-Free Languages

Theorem:

$$\left\{\begin{array}{c}\text{Context-Free}\\ \text{Languages}\\ \text{(Grammars)}\end{array}\right\} = \left\{\begin{array}{c}\text{Languages}\\ \text{Accepted by}\\ \text{PDAs}\end{array}\right\}$$

Proof - Step 1:

$$\left\{\begin{array}{c}\text{Context-Free}\\ \text{Languages}\\ \text{(Grammars)}\end{array}\right\} \subseteq \left\{\begin{array}{c}\text{Languages}\\ \text{Accepted by}\\ \text{PDAs}\end{array}\right\}$$

Convert any context-free grammar G to a PDA M with: $L(G) = L(M)$

Proof - Step 2:

$$\left\{\begin{array}{c}\text{Context-Free}\\ \text{Languages}\\ \text{(Grammars)}\end{array}\right\} \supseteq \left\{\begin{array}{c}\text{Languages}\\ \text{Accepted by}\\ \text{PDAs}\end{array}\right\}$$

Convert any PDA M to a context-free grammar G with: $L(G) = L(M)$

Proof - step 1

Convert

Context-Free Grammars
to
PDAs

Take an arbitrary context-free grammar G

We will convert G to a PDA M such that:

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

Conversion Procedure:

For each production in G

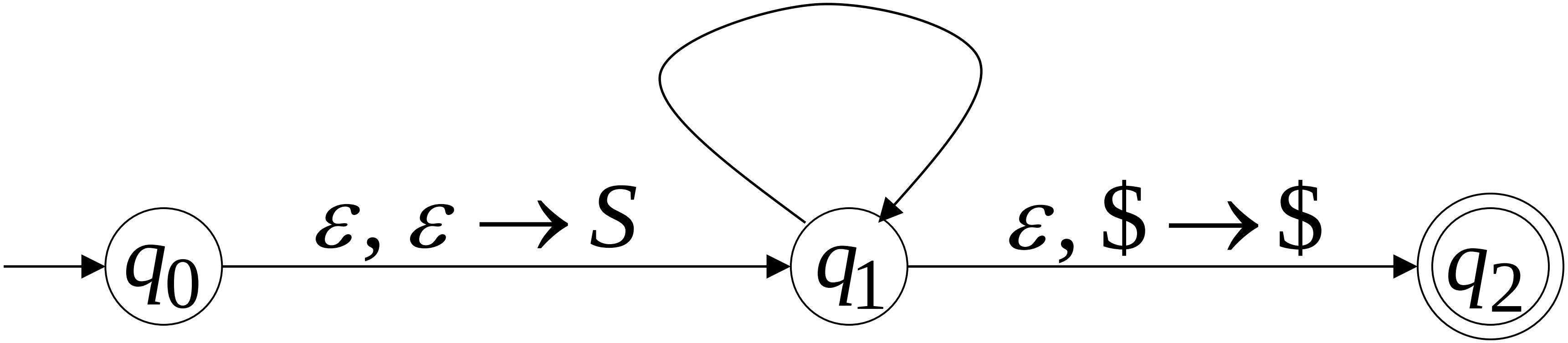
$$A \to w$$

For each terminal in G

$$a$$

Add transitions

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



Example

Grammar

$S \rightarrow aSTb$

$S \rightarrow b$

$T \rightarrow Ta$

$T \rightarrow \varepsilon$

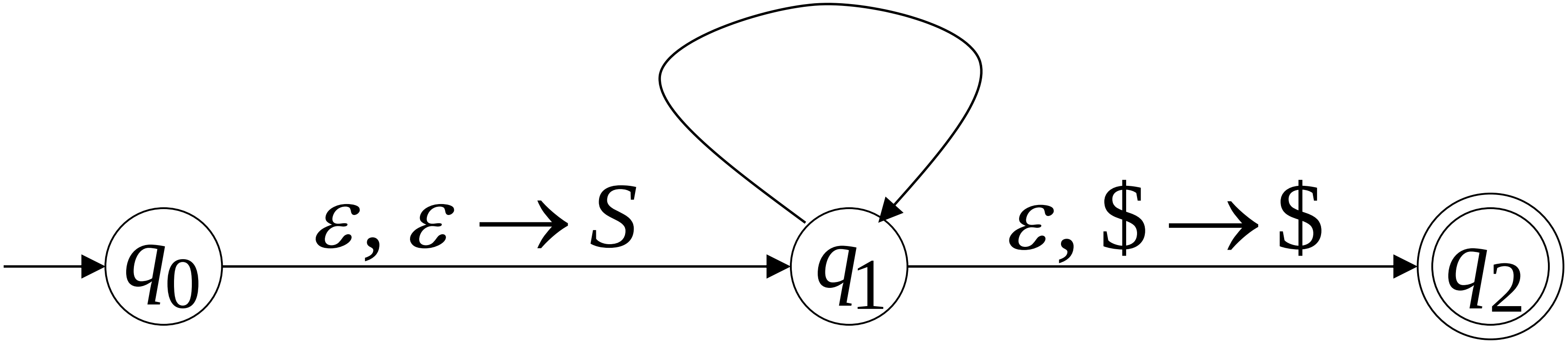
PDA

$\varepsilon, S \rightarrow aSTb$

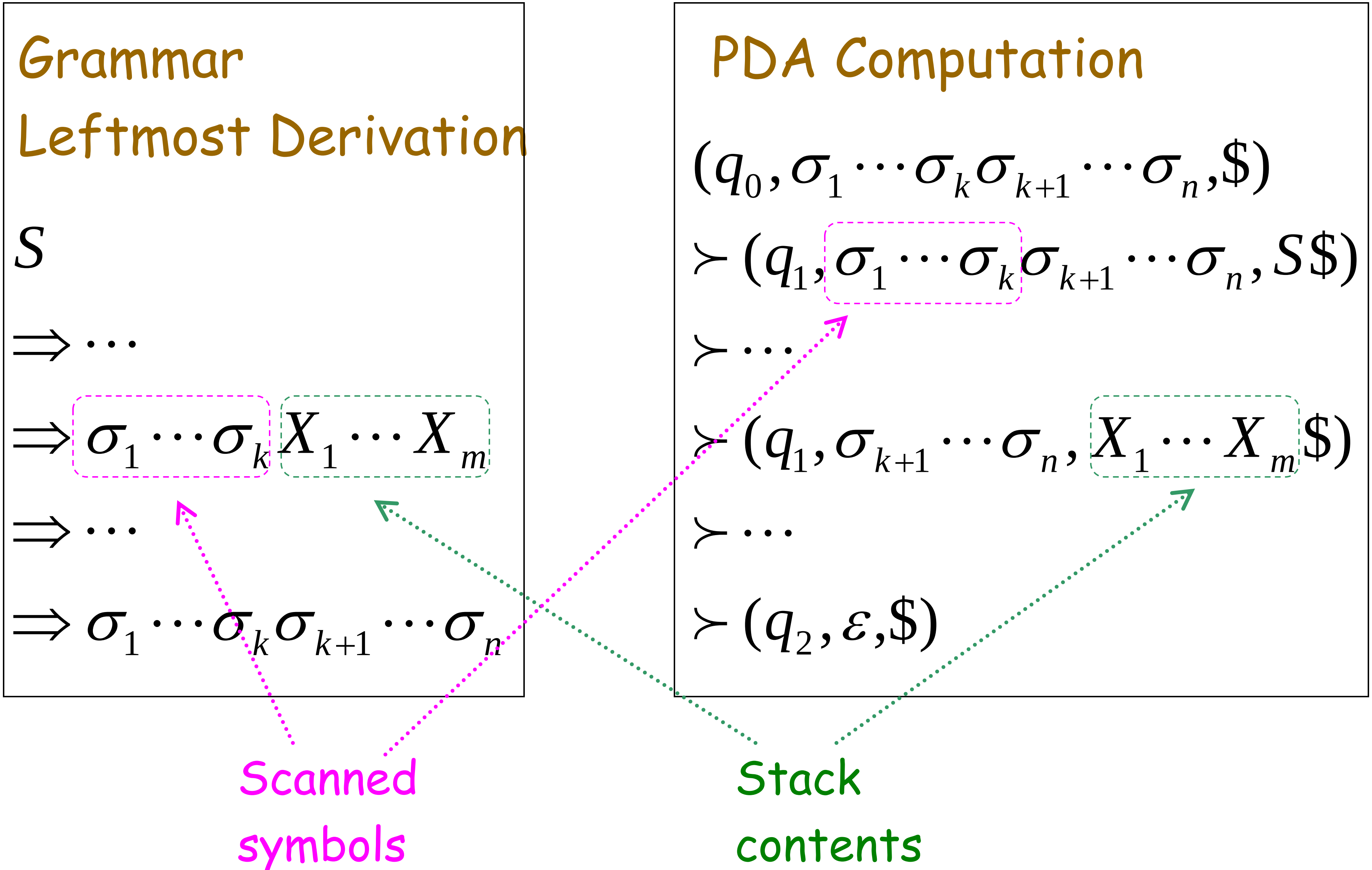
$\varepsilon, S \rightarrow b$

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

$\varepsilon, T \rightarrow \varepsilon$ $\quad$ $b, b \rightarrow \varepsilon$



PDA simulates leftmost derivations



Grammar

Leftmost Derivation

S

$\Rightarrow \cdots$

$\Rightarrow \boxed{\sigma_1 \cdots \sigma_k} \boxed{X_1 \cdots X_m}$

$\Rightarrow \cdots$

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

PDA Computation

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

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

$\succ \cdots$

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

$\succ \cdots$

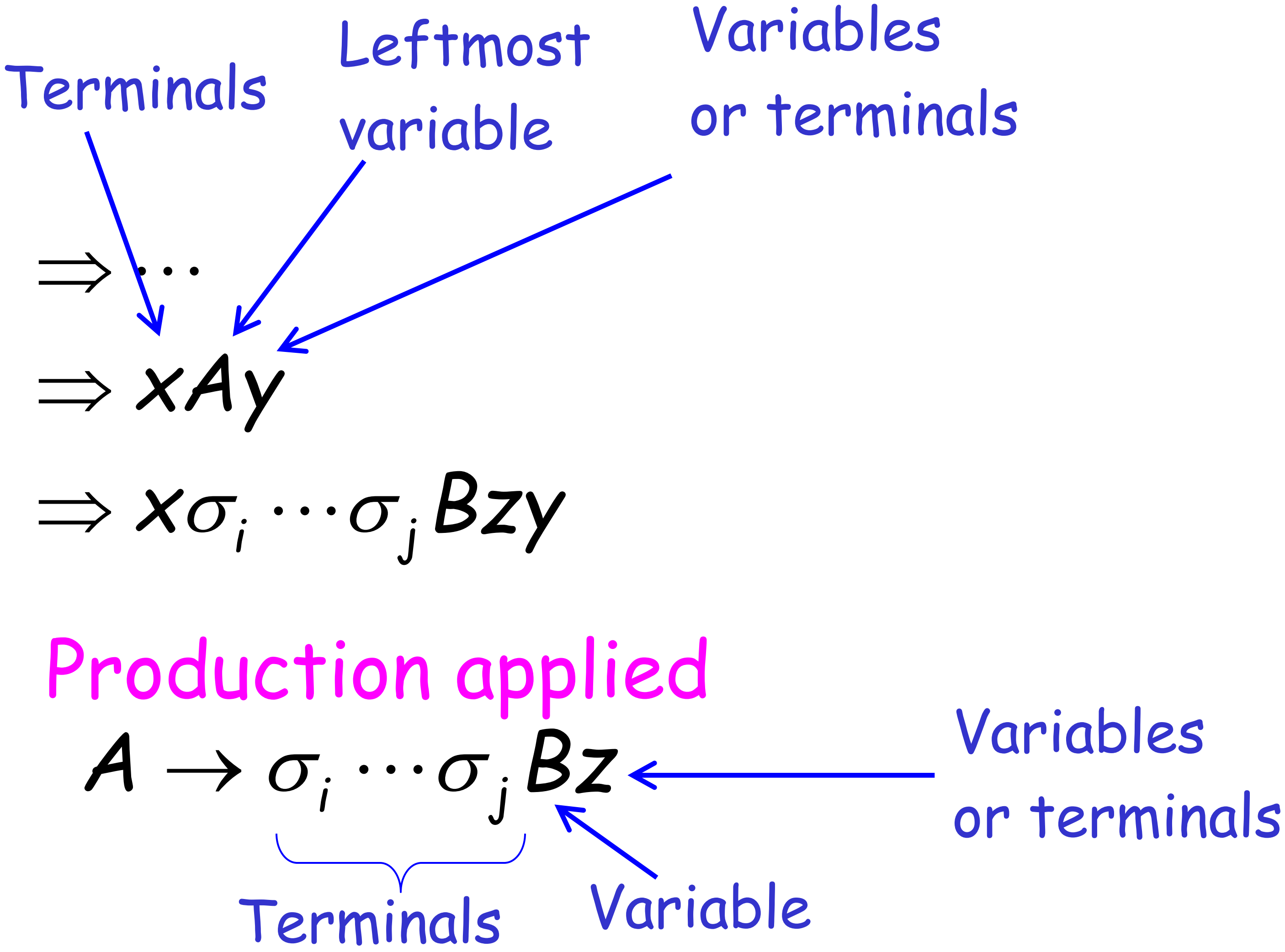
$\succ (q_2, \varepsilon, \$)$

Scanned symbols

Stack contents

Grammar

Leftmost Derivation



Terminals

Leftmost
variable

Variables
or terminals

$$
\begin{aligned}
&\Rightarrow \cdots \\
&\Rightarrow xAy \\
&\Rightarrow x\sigma_i \cdots \sigma_j Bzy
\end{aligned}
$$

Production applied

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

Variables
or terminals

Variable

Grammar
Leftmost Derivation

$$\Rightarrow \cdots$$

$$\Rightarrow xAy$$

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

Production applied

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

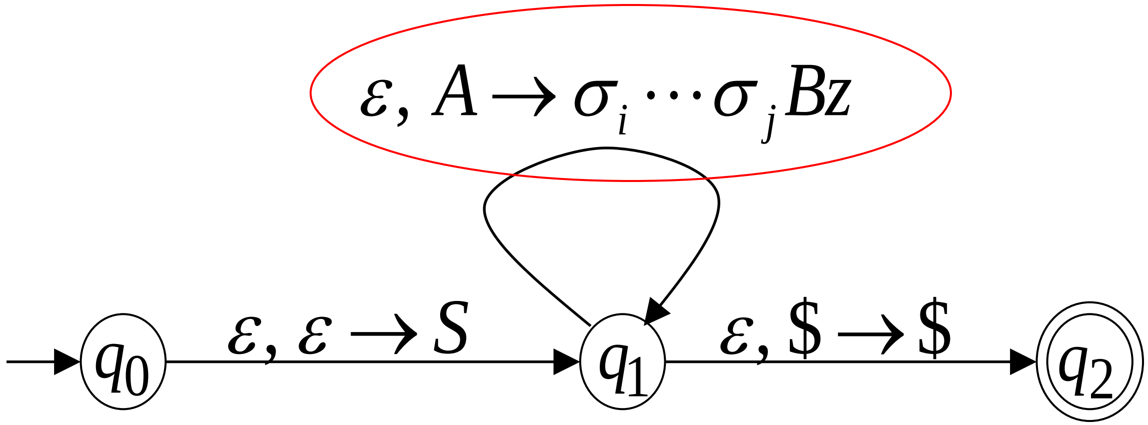
PDA Computation

$$\succ \cdots$$

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

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

Transition applied



Grammar
Leftmost Derivation

PDA Computation

$\Rightarrow \cdots$

$\Rightarrow xAy$

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

$\succ \cdots$

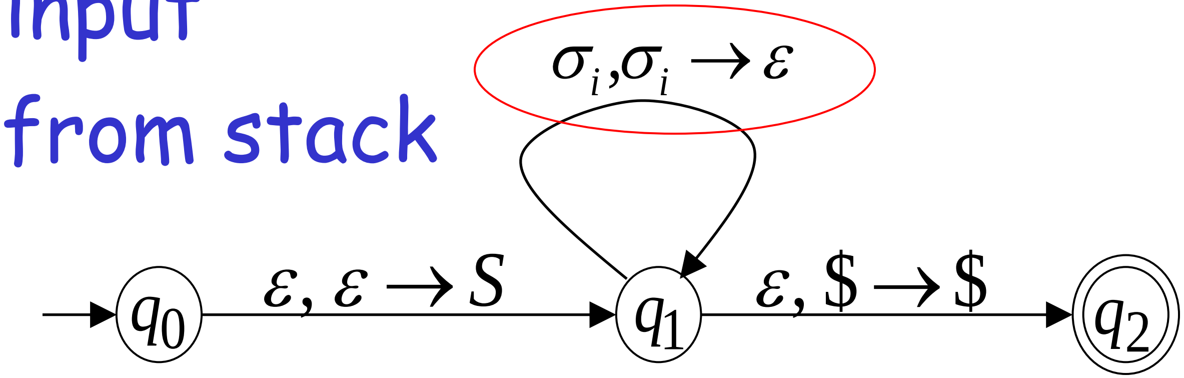
$\succ (q_1, \sigma_i \cdots \sigma_n, Ay\$)$

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

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

Transition applied

Read σ_i from input
and remove it from stack



Grammar

Leftmost Derivation

$\Rightarrow \cdots$

$\Rightarrow xAy$

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

PDA Computation

$\succ \cdots$

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

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

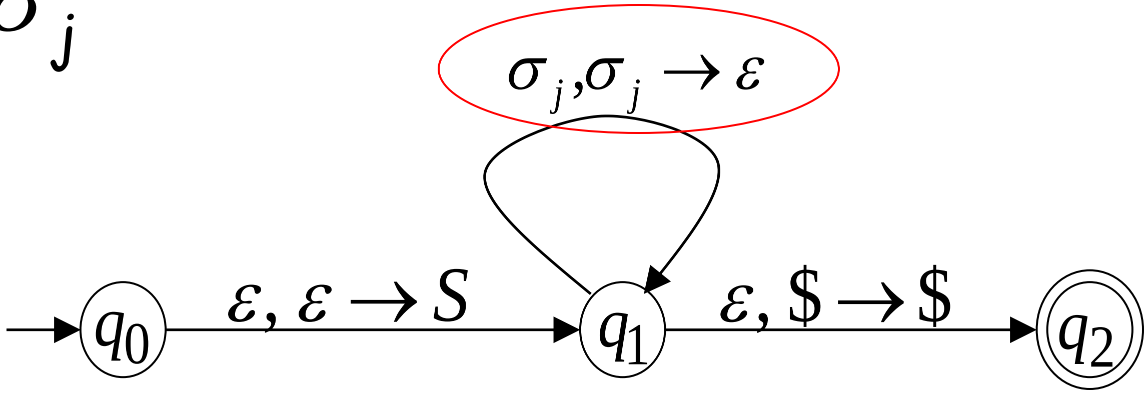
$\succ (q_1, \sigma_{i+1} \cdots \sigma_n, \sigma_{i+1} \cdots \sigma_j Bzy\$)$

$\succ \cdots$

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

Last Transition applied

All symbols $\sigma_i \cdots \sigma_j$ have been removed from top of stack



The process repeats with the next leftmost variable

$$\Rightarrow \cdots$$

$$\Rightarrow xAy$$

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

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

$$\succ \cdots$$

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

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

$$\succ \cdots$$

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

Production applied

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

And so on……

Example:

Input

a	b	a	b

Time 0

$\varepsilon, S \rightarrow aSTb$

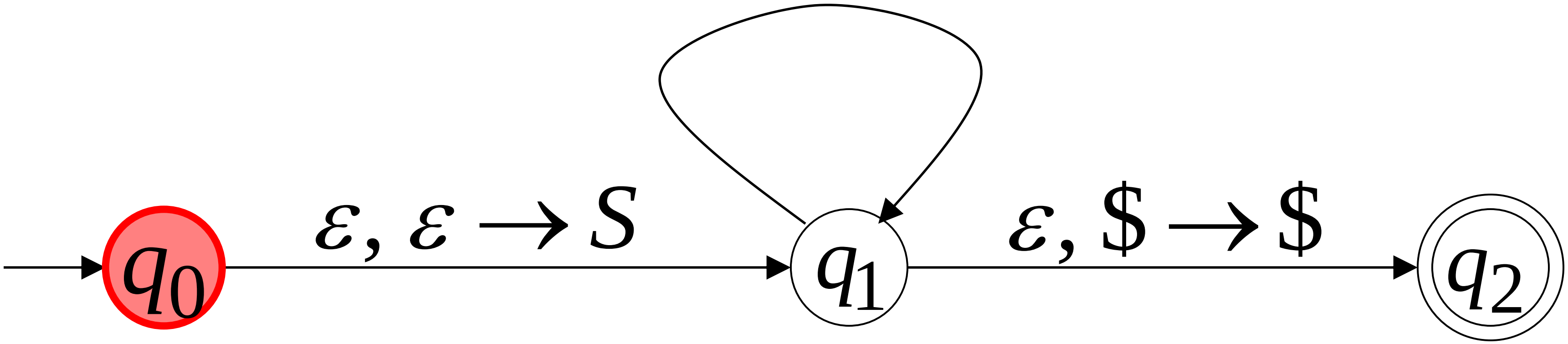
$\varepsilon, S \rightarrow b$

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

$\varepsilon, T \rightarrow \varepsilon$ $\qquad b, b \rightarrow \varepsilon$

$\$$

Stack



Derivation: S

Input

a	b	a	b

Time 1

$\varepsilon, S \to aSTb$

$\varepsilon, S \to b$

$\varepsilon, T \to Ta$

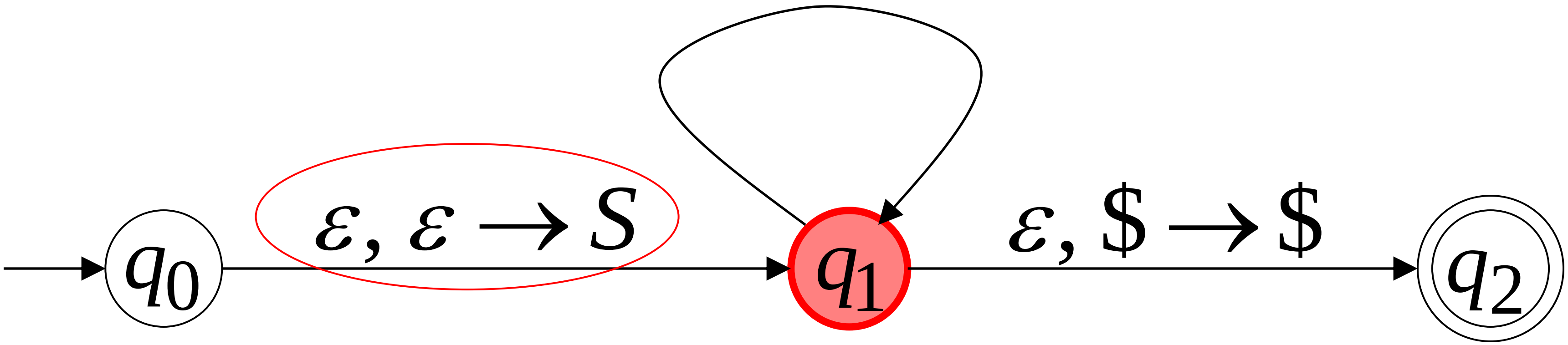
$\varepsilon, T \to \varepsilon$

$a, a \to \varepsilon$

$b, b \to \varepsilon$

S
$\$$

Stack



Derivation: $S \Rightarrow aSTb$

Input

a	b	a	b

Time 2

$\varepsilon, S \rightarrow aSTb$

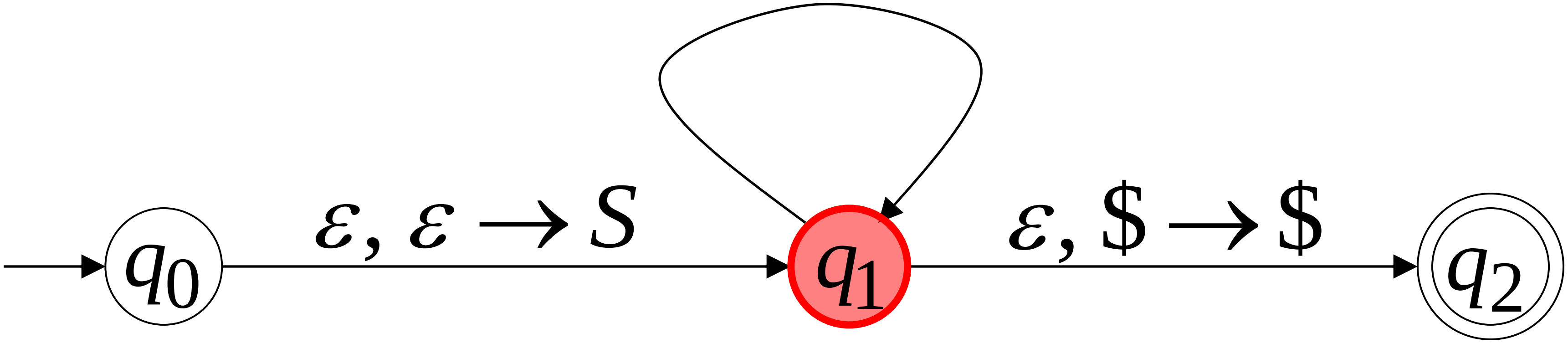
$\varepsilon, S \rightarrow b$

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

$\varepsilon, T \rightarrow \varepsilon$ $\quad b, b \rightarrow \varepsilon$

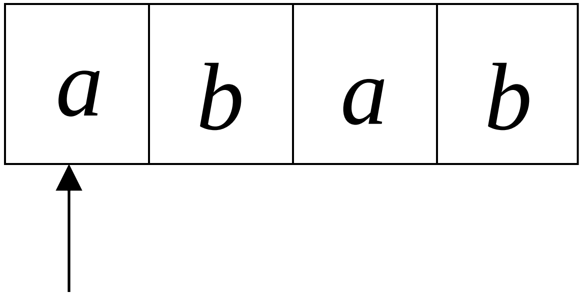
a
S
T
b
$\$$

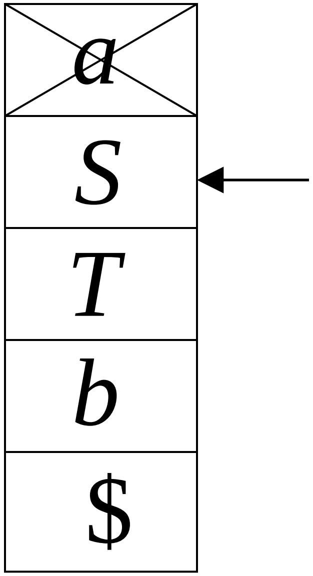
Stack



Derivation: $S \Rightarrow aSTb$

Input





Stack

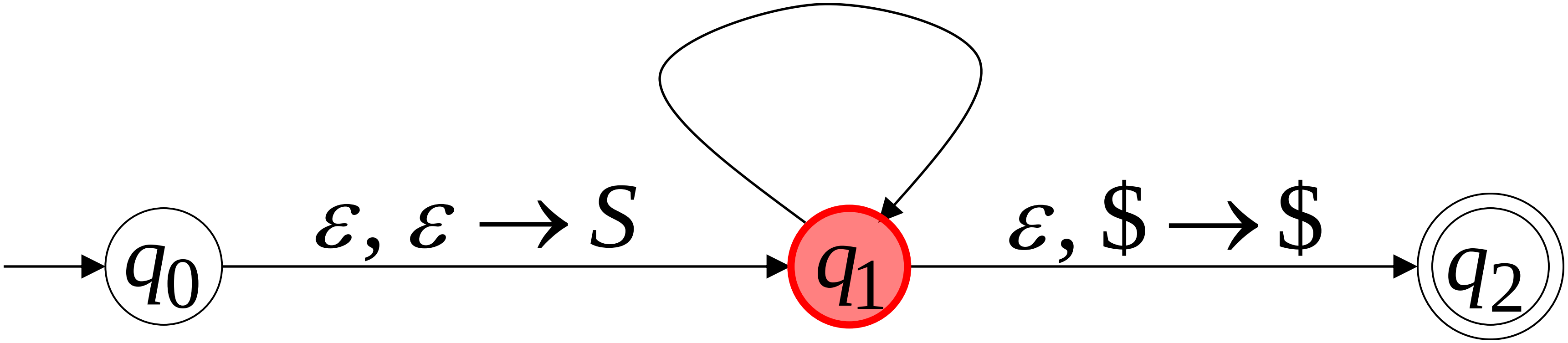
Time 3

$\varepsilon, S \rightarrow aSTb$

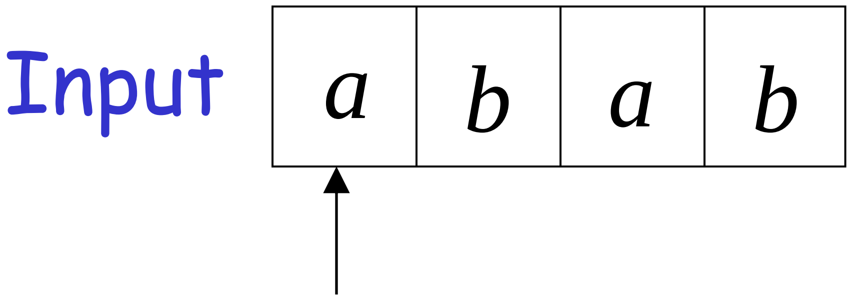
$\varepsilon, S \rightarrow b$

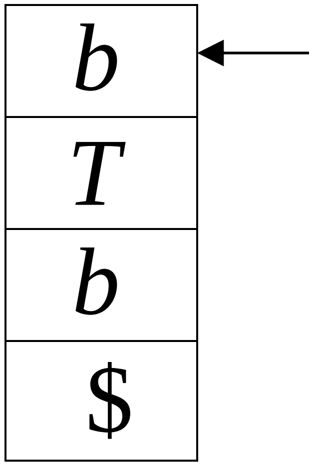
$\varepsilon, T \rightarrow Ta$ $\quad$ $a, a \rightarrow \varepsilon$

$\varepsilon, T \rightarrow \varepsilon$ $\quad$ $b, b \rightarrow \varepsilon$



Derivation: $S \Rightarrow aSTb \Rightarrow abTb$





Stack

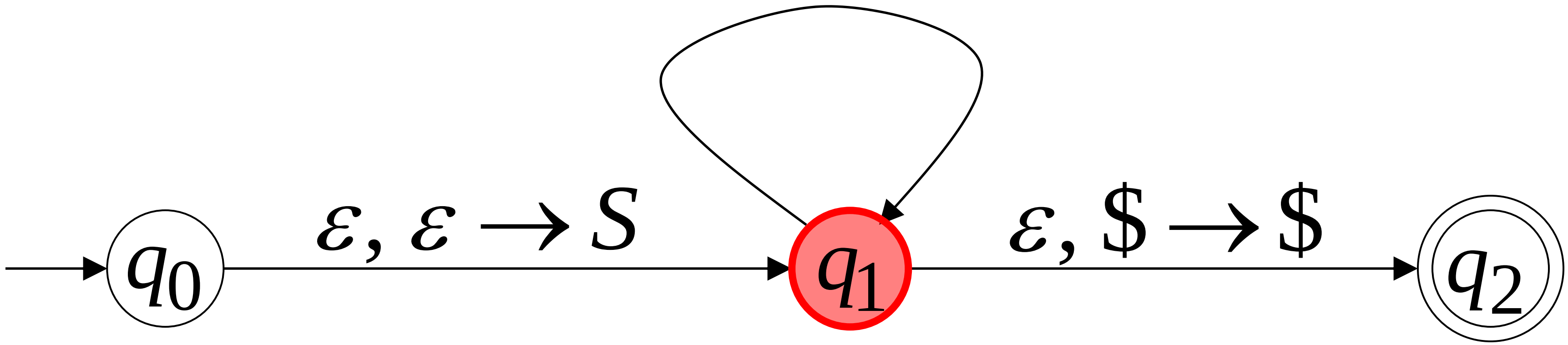
Time 4

$\varepsilon, S \rightarrow aSTb$

$\boxed{\varepsilon, S \rightarrow b}$

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

$\varepsilon, T \rightarrow \varepsilon$ $\qquad b, b \rightarrow \varepsilon$



Derivation: $S \Rightarrow aSTb \Rightarrow abTb$

Input

a	b	a	b

Stack

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

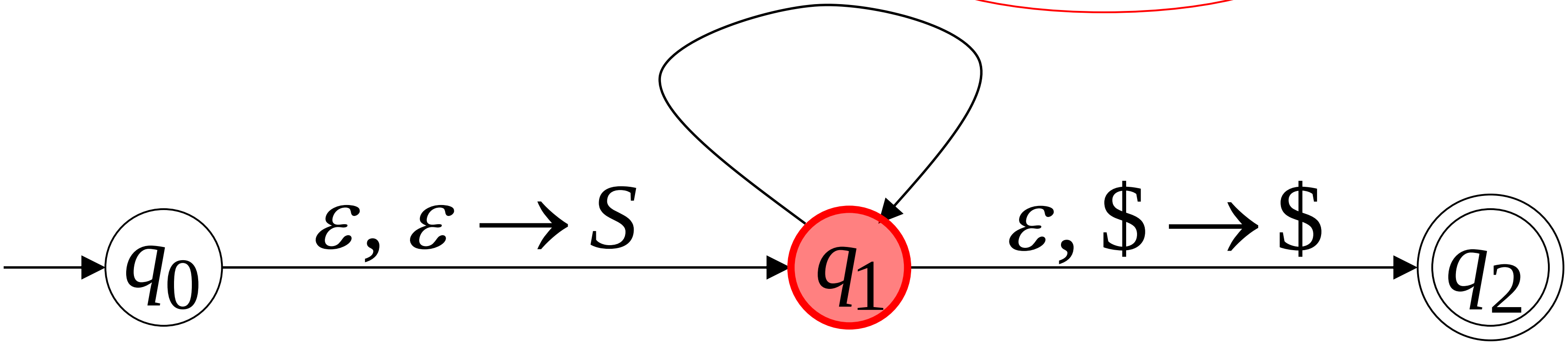
Time 5

$\varepsilon, S \to aSTb$

$\varepsilon, S \to b$

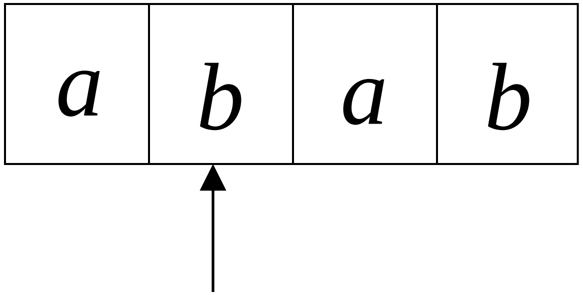
$\varepsilon, T \to Ta$ $\qquad a, a \to \varepsilon$

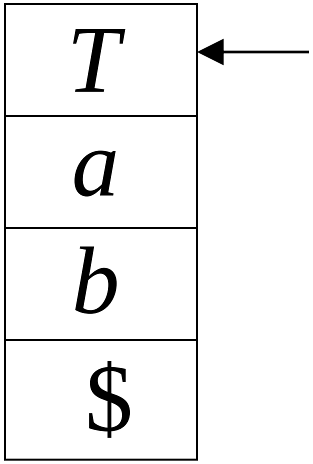
$\varepsilon, T \to \varepsilon$ $\qquad b, b \to \varepsilon$



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

Input





Stack

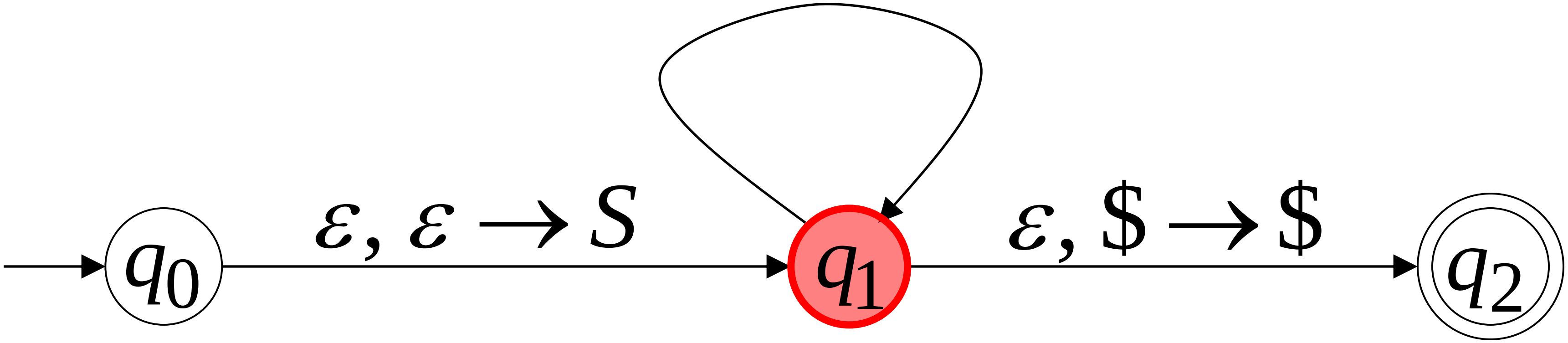
Time 6

$\varepsilon, S \to aSTb$

$\varepsilon, S \to b$

$\boxed{\varepsilon, T \to Ta}$ $\qquad a, a \to \varepsilon$

$\varepsilon, T \to \varepsilon$ $\qquad b, b \to \varepsilon$



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

Input

a	b	a	b

Stack

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

Time 7

$\varepsilon, S \rightarrow aSTb$

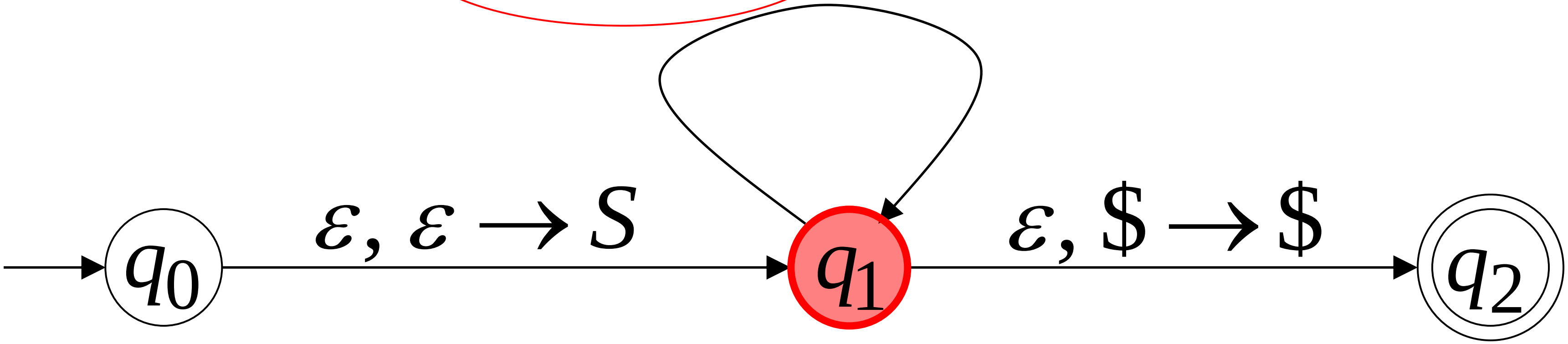
$\varepsilon, S \rightarrow b$

$\varepsilon, T \rightarrow Ta$

$\varepsilon, T \rightarrow \varepsilon$

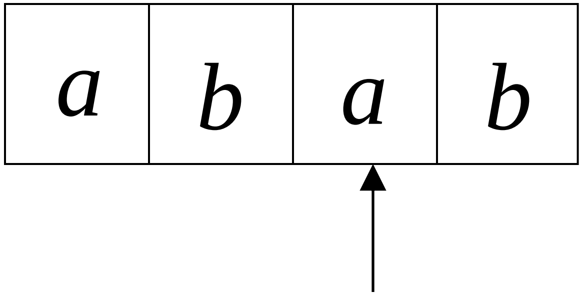
$a, a \rightarrow \varepsilon$

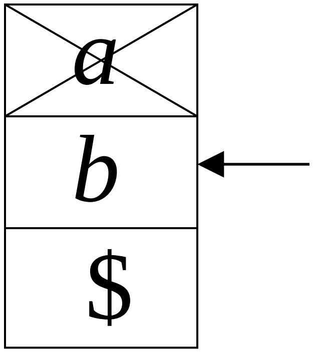
$b, b \rightarrow \varepsilon$



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

Input





Time 8

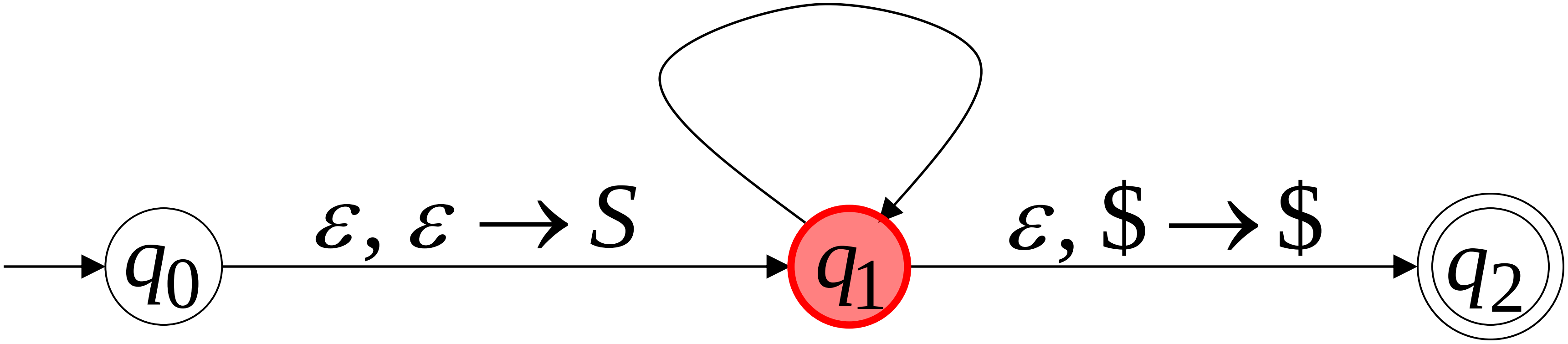
$\varepsilon, S \rightarrow aSTb$

$\varepsilon, S \rightarrow b$

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

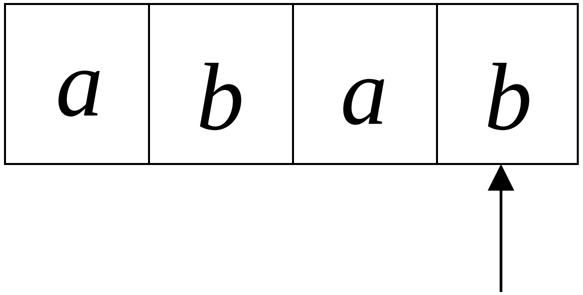
$\varepsilon, T \rightarrow \varepsilon$ $\quad b, b \rightarrow \varepsilon$

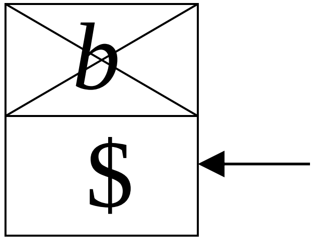
Stack



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

Input





Stack

Time 9

$\varepsilon, S \rightarrow aSTb$

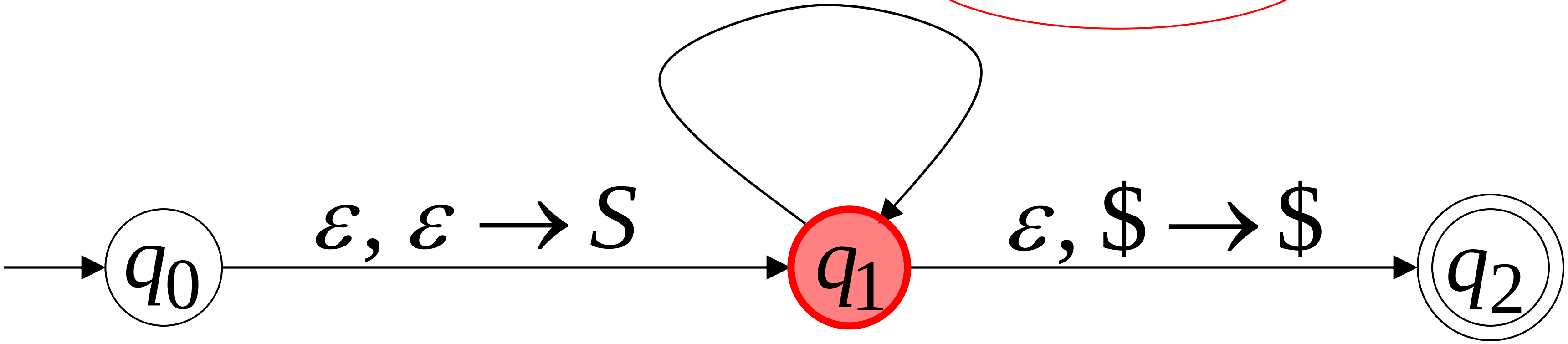
$\varepsilon, S \rightarrow b$

$\varepsilon, T \rightarrow Ta$

$\varepsilon, T \rightarrow \varepsilon$

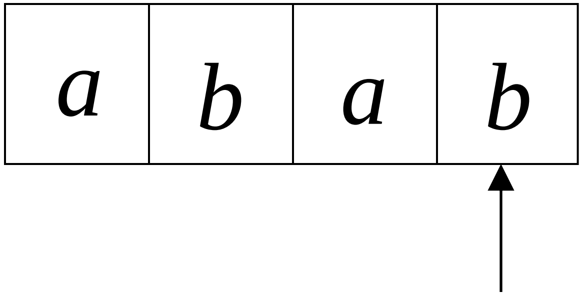
$a, a \rightarrow \varepsilon$

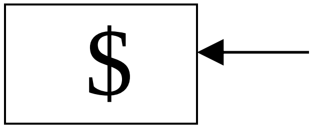
$b, b \rightarrow \varepsilon$



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

Input





Stack

Time 10

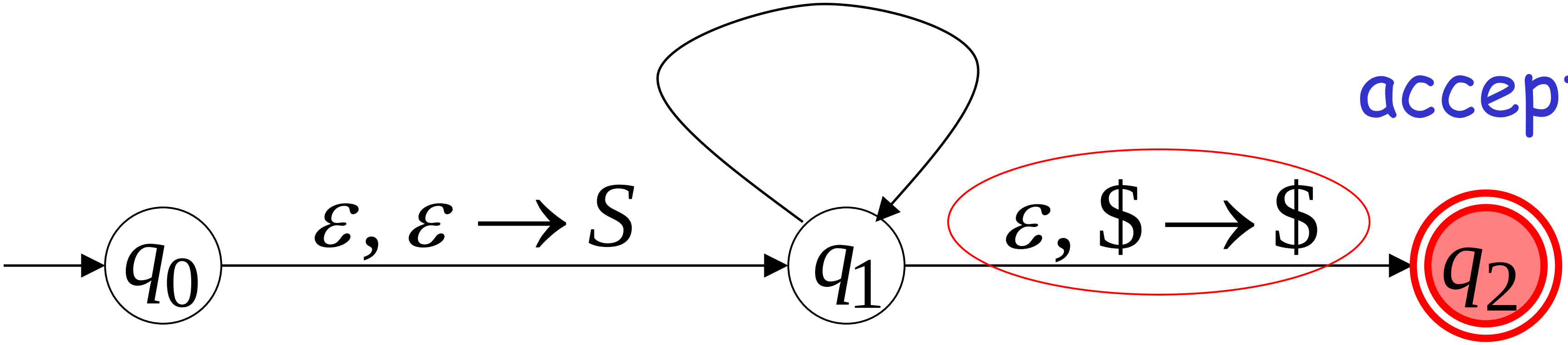
$\varepsilon, S \rightarrow aSTb$

$\varepsilon, S \rightarrow b$

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

$\varepsilon, T \rightarrow \varepsilon$ $\quad b, b \rightarrow \varepsilon$

accept



Grammar Leftmost Derivation	PDA Computation
S	$(q_0, abab, \$)$ $\succ (q_1, abab, S\$)$
$\Rightarrow aSTb$	$\succ (q_1, bab, STb\$)$
$\Rightarrow abTb$	$\succ (q_1, bab, bTb\$)$ $\succ (q_1, ab, Tb\$)$
$\Rightarrow abTab$	$\succ (q_1, ab, Tab\$)$
$\Rightarrow abab$	$\succ (q_1, ab, ab\$)$ $\succ (q_1, b, b\$)$ $\succ (q_1, \varepsilon, \$)$ $\succ (q_2, \varepsilon, \$)$

In general, it can be shown that:

Grammar G
generates
string w

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

If and
Only if

PDA M
accepts w

$$(q_0, w, \$) \overset{*}{\succ} (q_2, \varepsilon, \$)$$

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

Proof - step 2

Convert

PDAs
to
Context-Free Grammars

Take an arbitrary PDA M

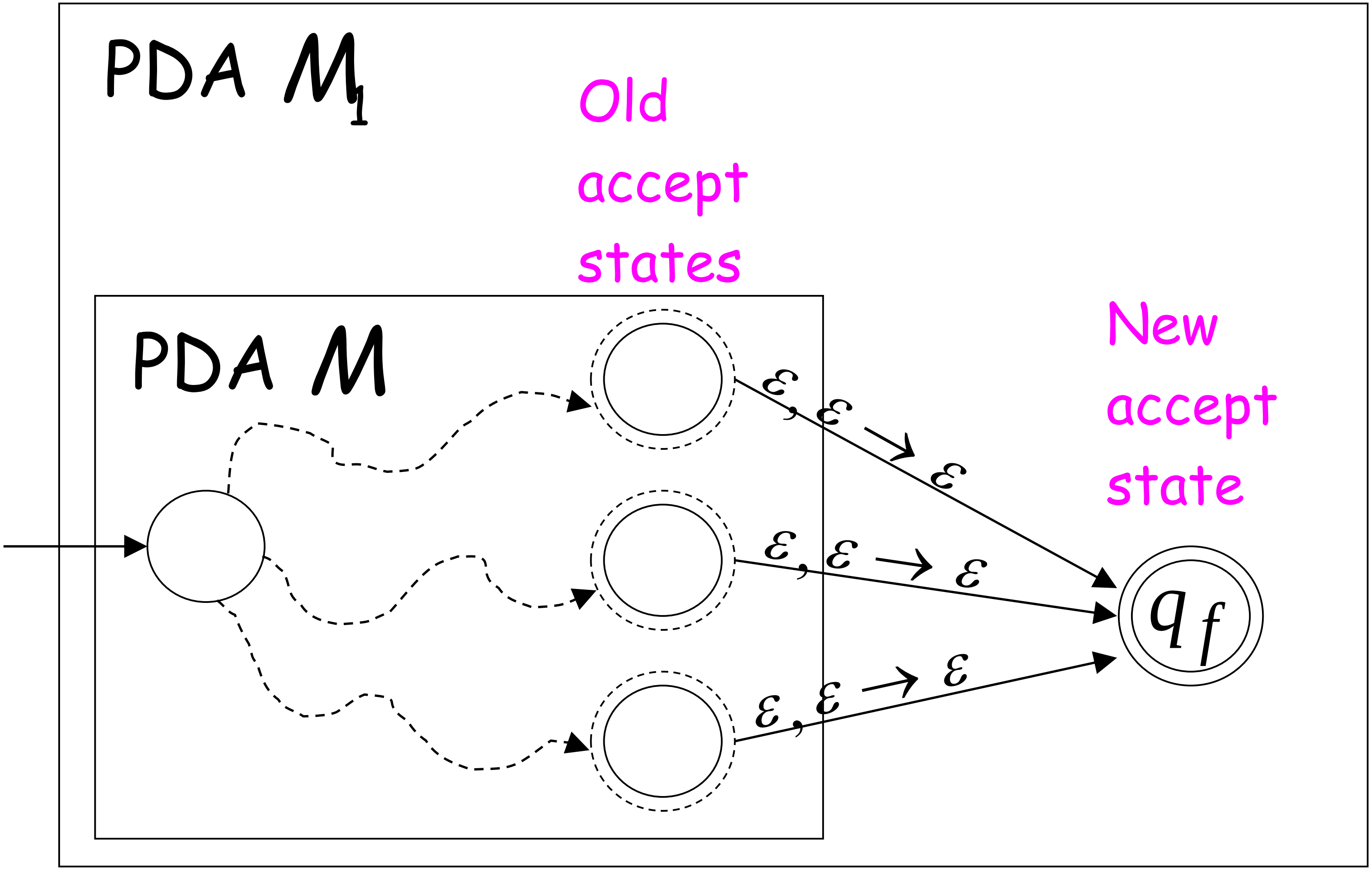
We will convert M
to a context-free grammar G such that:

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

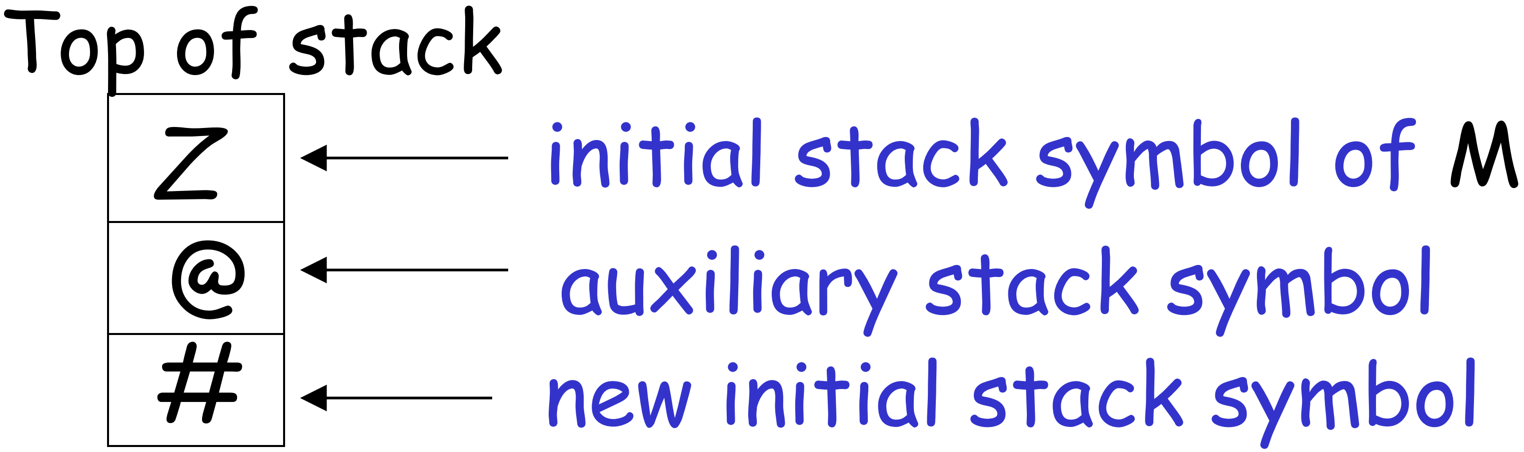
First modify PDA M so that:

1. The PDA has a single accept state
2. Use new initial stack symbol #
3. On acceptance the stack contains only stack symbol # (this symbol is not used in any transition)
4. Each transition either pushes a symbol or pops a symbol but not both together

1. The PDA has a single accept state



2. Use new initial stack symbol

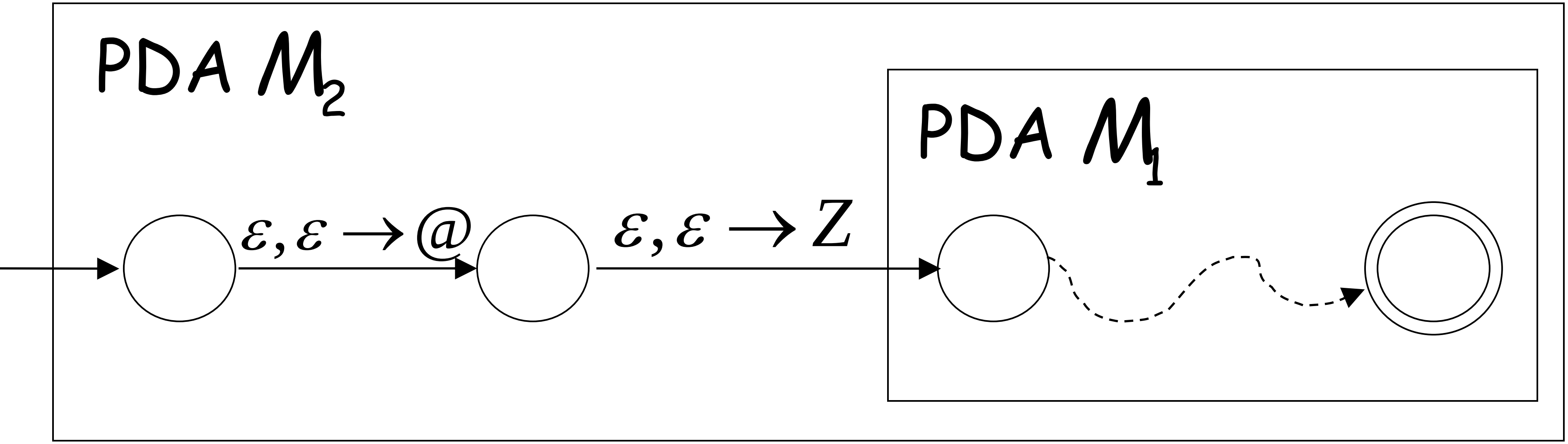


Top of stack

Z ← initial stack symbol of M

@ ← auxiliary stack symbol

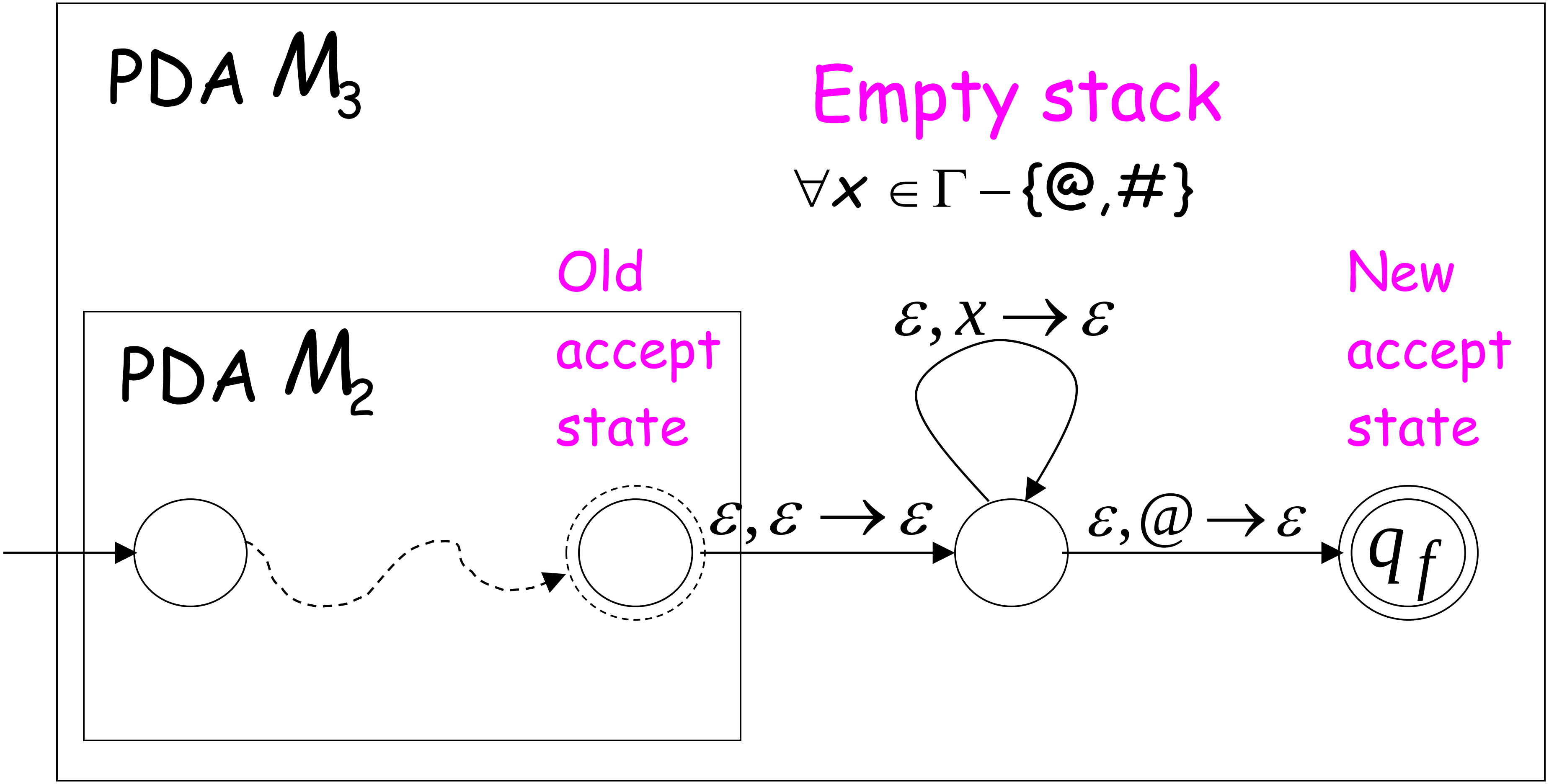
← new initial stack symbol



M_1 still thinks that Z is the initial stack

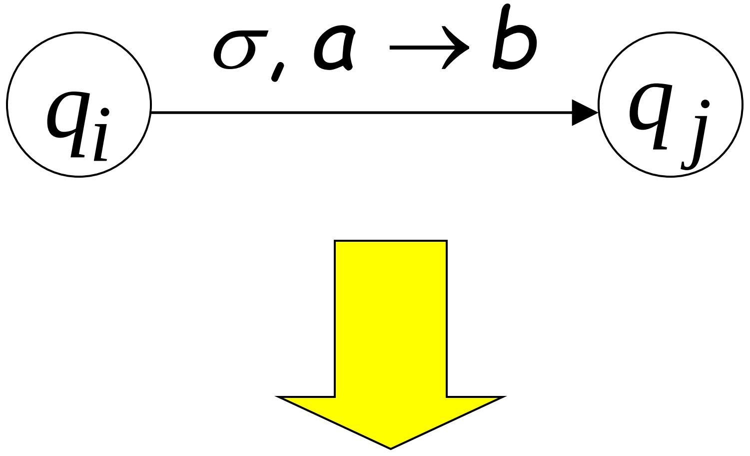
3. On acceptance the stack contains only stack symbol

(this symbol is not used in any transition)

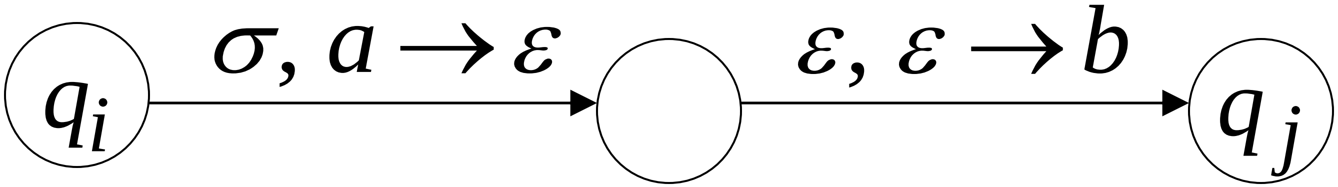


4. Each transition either pushes a symbol or pops a symbol but not both together

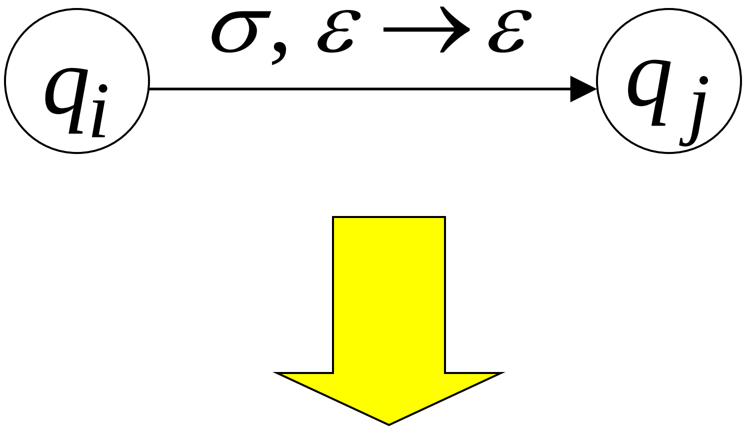
PDA M_3



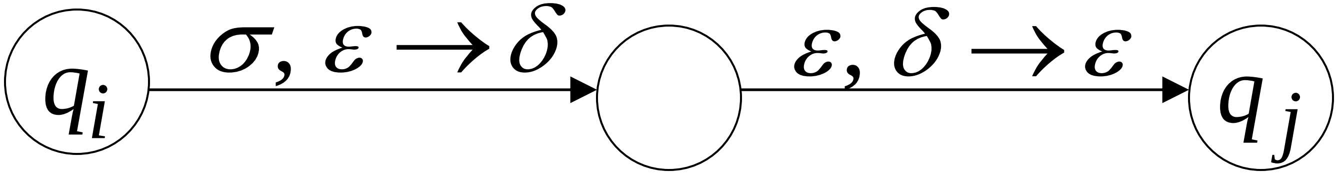
PDA M_4



PDA M_3



PDA M_4

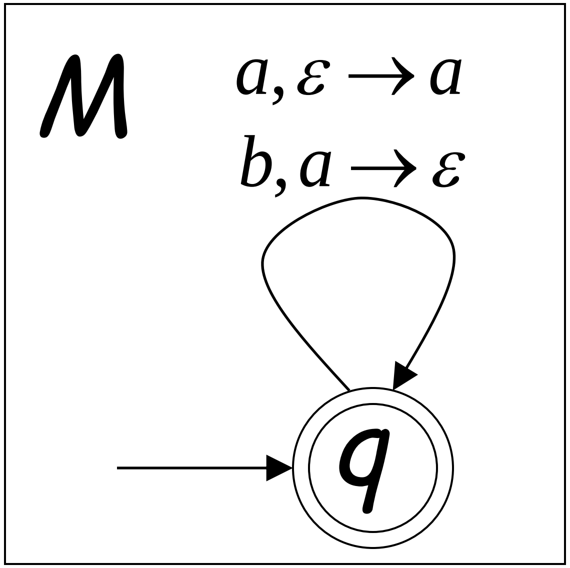


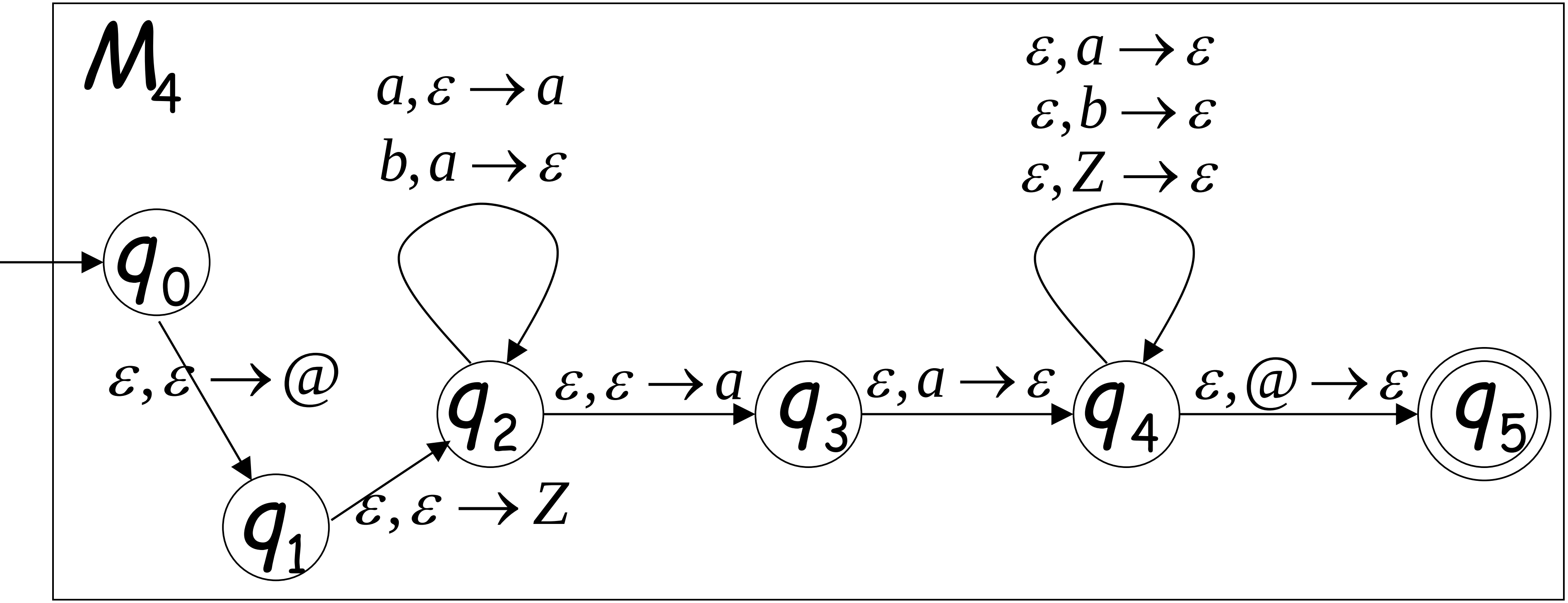
Where δ is a symbol of the stack alphabet

PDA M_4 is the final modified PDA

Note that the new initial stack symbol #
is never used in any transition

Example:


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


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

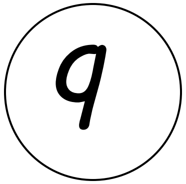
Grammar Construction

Variables: A_{q_i, q_j}

States of PDA

PDA

Kind 1: for each state

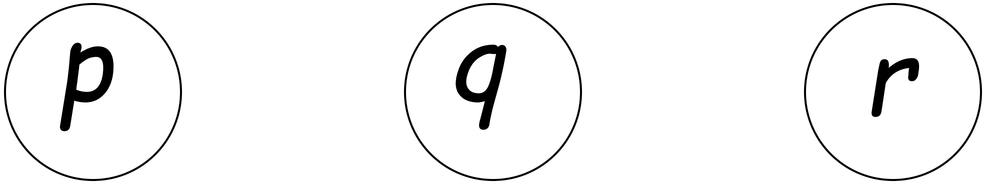


Grammar

$$A_{qq} \rightarrow \varepsilon$$

PDA

Kind 2: for every three states



Grammar

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

PDA

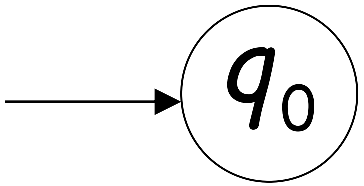
Kind 3: for every pair of such transitions

Grammar

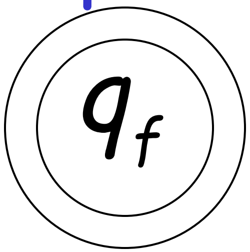
$$A_{pq} \rightarrow aA_{rs}b$$

PDA

Initial state



Accept state

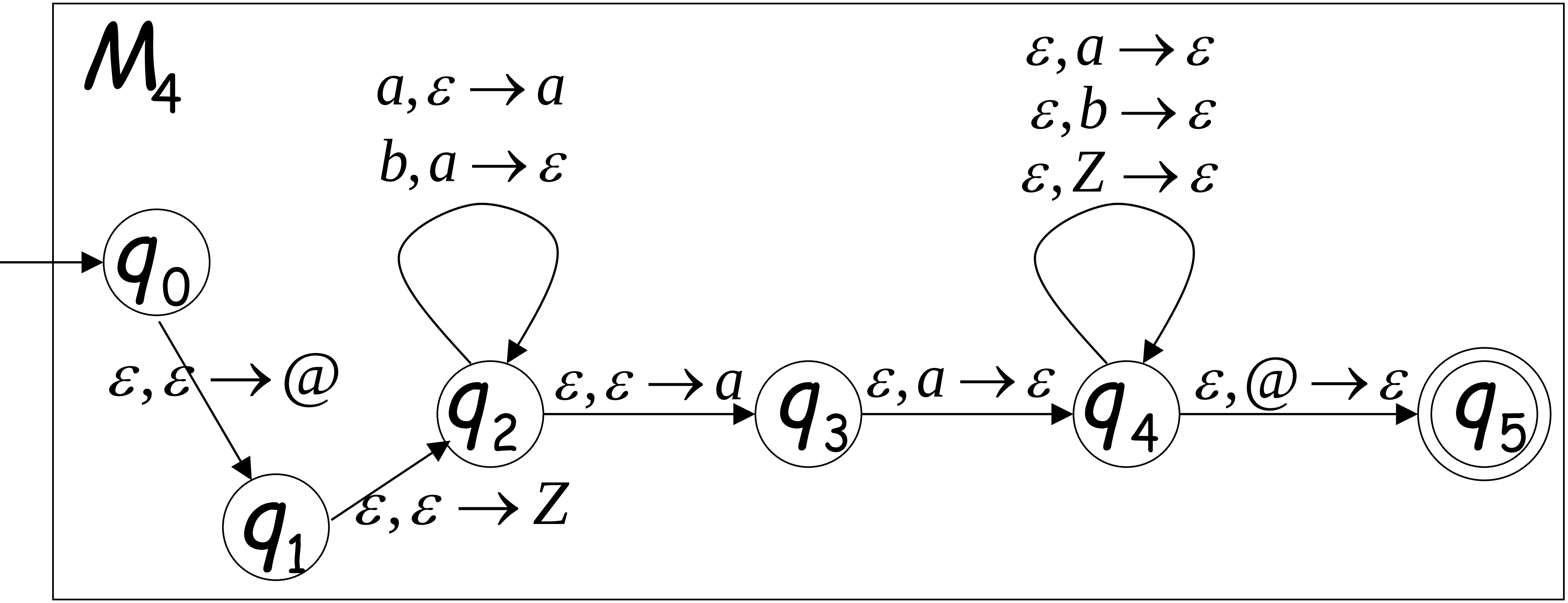


Grammar

Start variable $A_{q_0 q_f}$

Example:

PDA



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

Grammar

Kind 1: from single states

$$A_{q_0q_0} \to \varepsilon$$

$$A_{q_1q_1} \to \varepsilon$$

$$A_{q_2q_2} \to \varepsilon$$

$$A_{q_3q_3} \to \varepsilon$$

$$A_{q_4q_4} \to \varepsilon$$

$$A_{q_5q_5} \to \varepsilon$$

Kind 2: from triplets of states

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

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

$$\vdots$$

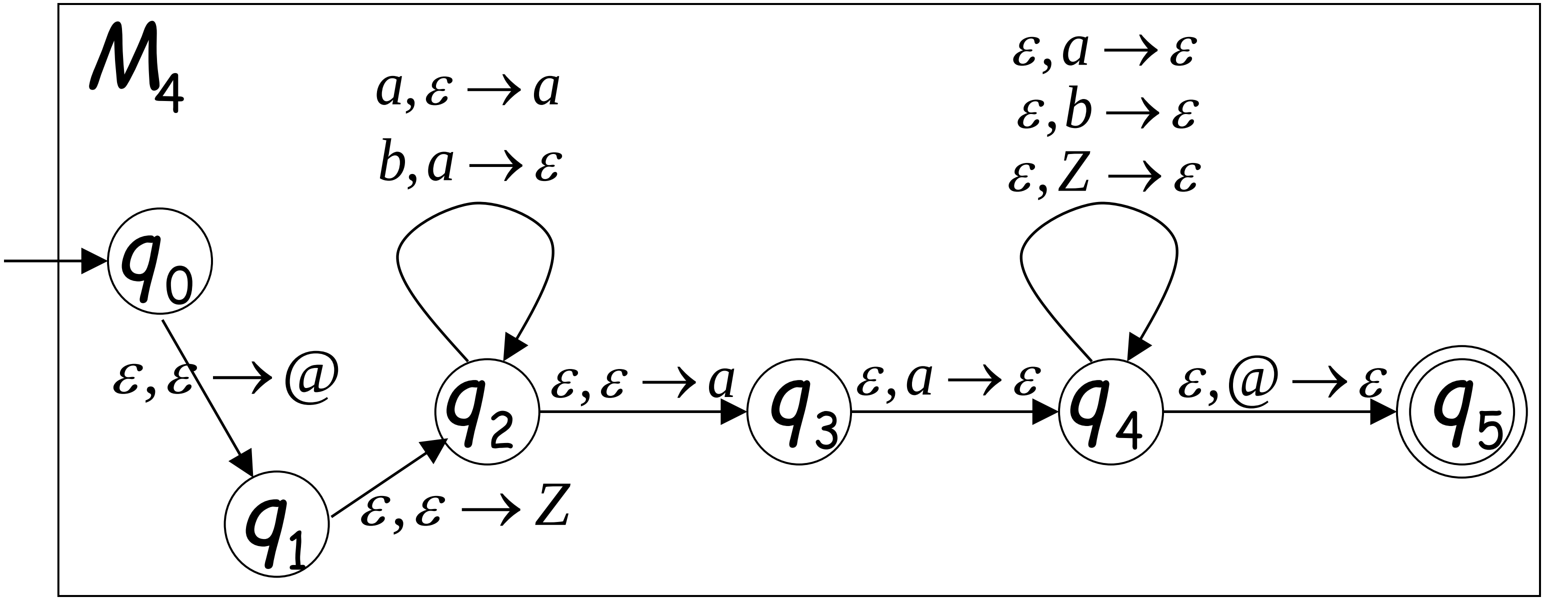
$$A_{q_0q_5} \to A_{q_0q_0}A_{q_0q_5} \mid A_{q_0q_1}A_{q_1q_5} \mid A_{q_0q_2}A_{q_2q_5} \mid A_{q_0q_3}A_{q_3q_5} \mid A_{q_0q_4}A_{q_4q_5} \mid A_{q_0q_5}A_{q_5q_5}$$

$$\vdots$$

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

Start variable $A_{q_0q_5}$

Kind 3: from pairs of transitions



$$A_{q_0q_5} \rightarrow A_{q_1q_4}$$

$$A_{q_1q_4} \rightarrow A_{q_2q_4}$$

$$A_{q_2q_4} \rightarrow aA_{q_2q_4}$$

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

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

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

$$A_{q_2q_4} \rightarrow A_{q_3q_3}$$

$$A_{q_2q_4} \rightarrow A_{q_3q_4}$$

Suppose that a PDA M is converted to a context-free grammar G

We need to prove that $L(G) = L(M)$

or equivalently

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

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

We need to show that if G has derivation:

$$A_{q_0 q_f} \overset{*}{\Rightarrow} w$$ (string of terminals)

Then there is an accepting computation in M :

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

with input string w

We will actually show that if G has derivation:

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

Then there is a computation in M :

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

Therefore:

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

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

Since there is no transition with the # symbol

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

Lemma:

If $A_{pq} \overset{*}{\Rightarrow} w$ (string of terminals)

then there is a computation from state p to state q on string w which leaves the stack empty:

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

Proof Intuition:

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

Type 2

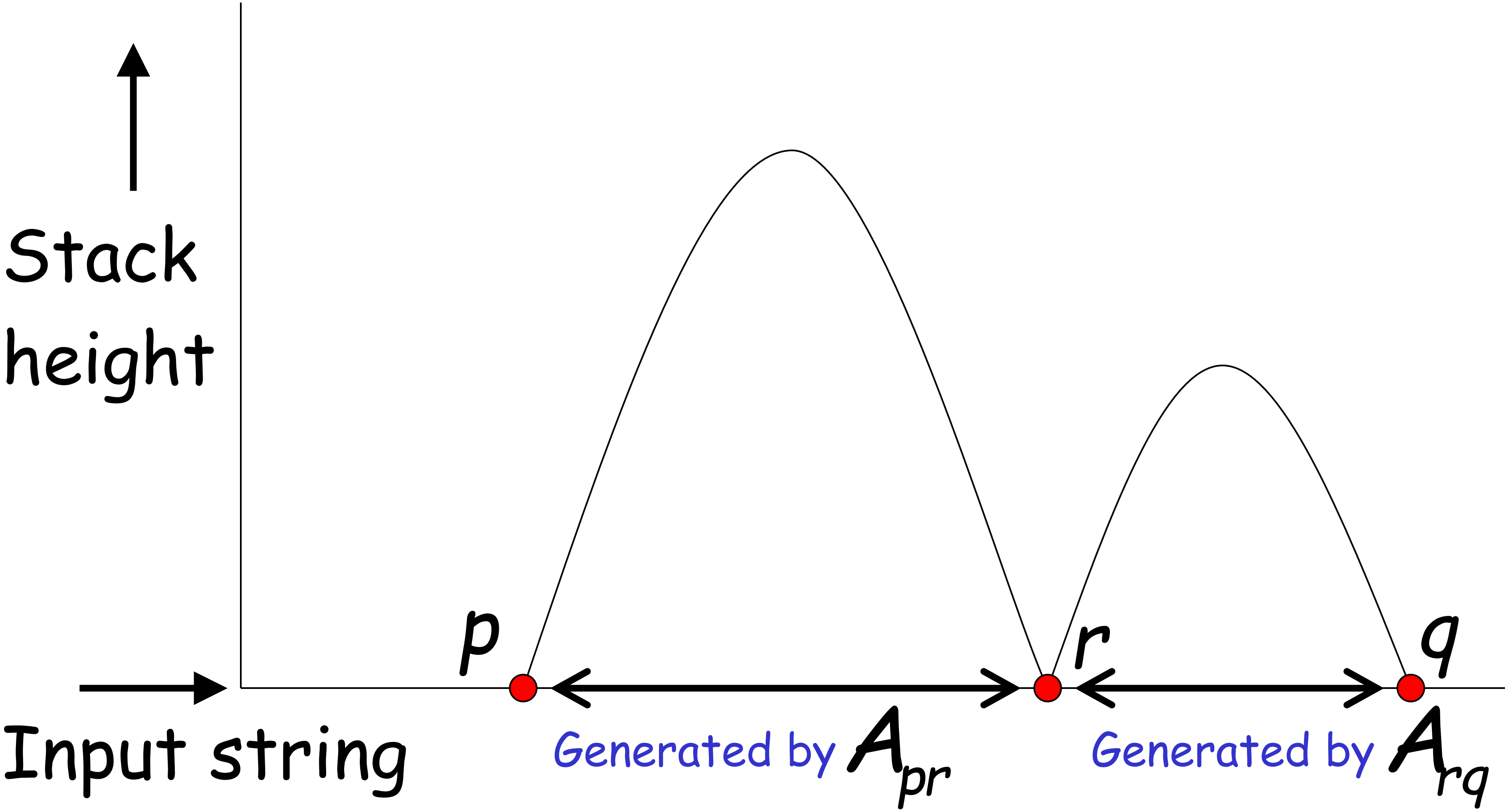
Case 1: $A_{pq} \Rightarrow A_{pr}A_{rq} \Rightarrow \cdots \Rightarrow w$

Type 3

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

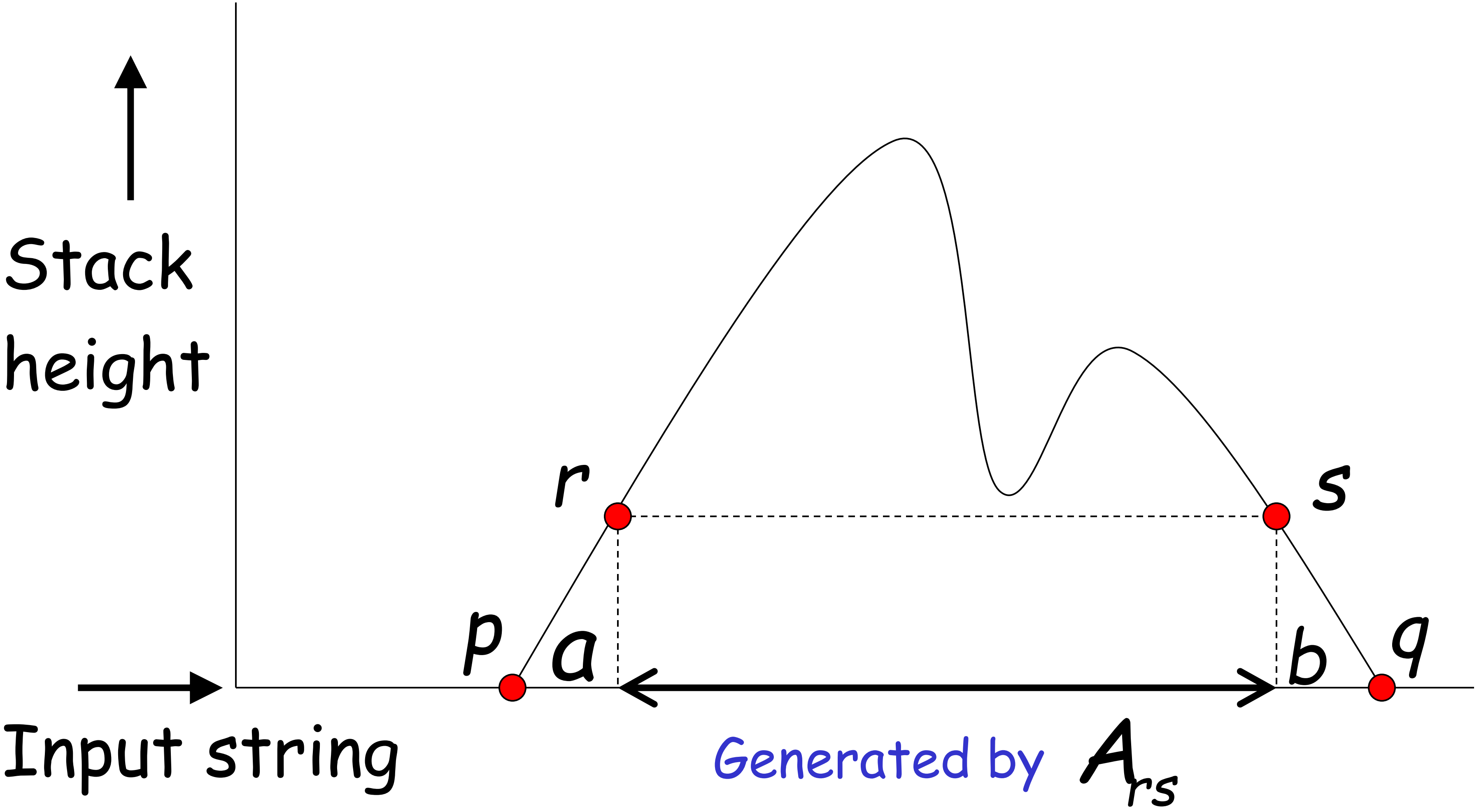
Type 2

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



Type 3

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



Formal Proof:

We formally prove this claim by induction on the number of steps in derivation:

$$A_{pq} \underbrace{\Rightarrow \cdots \Rightarrow}_{\text{number of steps}} w$$

Induction Basis: $A_{pq} \Rightarrow w$

(one derivation step)

A Kind 1 production must have been used:

$$A_{pp} \to \varepsilon$$

Therefore, $p = q$ and $w = \varepsilon$

This computation of PDA trivially exists:

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

Induction Hypothesis:

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

k derivation steps

suppose it holds:

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

Induction Step:

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

We have to show:

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

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

$k+1$ derivation steps

Type 2

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

Type 3

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

Type 2

Case 1:

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

We can write $w = yz$

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

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

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

From induction hypothesis, in PDA:

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

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

From induction hypothesis, in PDA:

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

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

⇓

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

since $w = yz$ ⇓

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

Type 3

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

We can write $w = ayb$

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

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

From induction hypothesis, the PDA has computation:

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

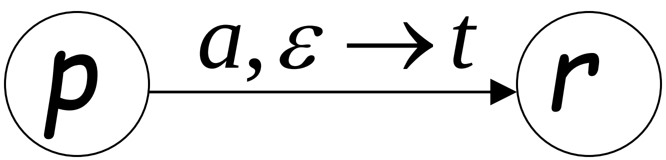
Type 3

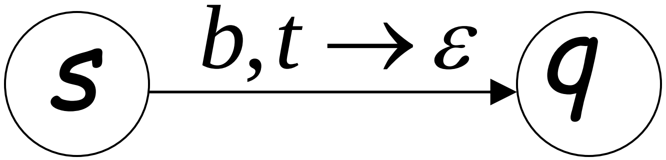
$$A_{pq} \Rightarrow aA_{rs}b \Rightarrow \cdots \Rightarrow w$$

Grammar contains production

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

And PDA Contains transitions





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

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

We know

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

We also know

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

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

Therefore:

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

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

since $w = ayb$

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

END OF PROOF

So far we have shown:

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

With a similar proof we can show

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

Therefore: $L(G) = L(M)$

Properties
of
Context-Free languages

Union

Context-free languages
are closed under: **Union**

L_1 is context free

L_2 is context free

→ $L_1 \cup L_2$ is context-free

Example

Language

Grammar

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

$$S_1 \to aS_1b \mid \lambda$$

$$L_2 = \{ww^R\}$$

$$S_2 \to aS_2a \mid bS_2b \mid \lambda$$

Union

$$L = \{a^n b^n\} \cup \{ww^R\}$$

$$S \to S_1 \mid S_2$$

In general:

For context-free languages $L_1, \; L_2$

with context-free grammars $G_1, \; G_2$

and start variables $S_1, \; S_2$

The grammar of the **union** $L_1 \cup L_2$

has new start variable S

and additional production $S \rightarrow S_1 \mid S_2$

Concatenation

Context-free languages
are closed under: **Concatenation**

L_1 is context free

L_2 is context free

⇒ $L_1 L_2$ is context-free

Example

Language **Grammar**

$L_1 = \{a^n b^n\}$ $S_1 \to aS_1b \mid \lambda$

$L_2 = \{ww^R\}$ $S_2 \to aS_2a \mid bS_2b \mid \lambda$

Concatenation

$L = \{a^n b^n\}\{ww^R\}$ $S \to S_1S_2$

In general:

For context-free languages $L_1, \; L_2$

with context-free grammars $G_1, \; G_2$

and start variables $S_1, \; S_2$

The grammar of the **concatenation** $L_1 L_2$

has new start variable S

and additional production $S \rightarrow S_1 S_2$

Star Operation

Context-free languages
are closed under: **Star-operation**

L is context free ⇒ L^* is context-free

Example

Language

Grammar

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

$$S \rightarrow aSb \mid \lambda$$

Star Operation

$$L = \{a^n b^n\}^*$$

$$S_1 \rightarrow SS_1 \mid \lambda$$

In general:

For context-free language L

with context-free grammar G

and start variable S

The grammar of the **star operation** L^*

has new start variable S_1

and additional production $S_1 \rightarrow SS_1 \mid \lambda$

Negative Properties
of
Context-Free Languages

Intersection

Context-free languages
are **not** closed under: **intersection**

L_1 is context free

L_2 is context free

⇒ $L_1 \cap L_2$

not necessarily
context-free

Example

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

Context-free:

$$S \to AC$$

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

$$C \to cC \mid \lambda$$

$$L_2 = \{a^n b^m c^m\}$$

Context-free:

$$S \to AB$$

$$A \to aA \mid \lambda$$

$$B \to bBc \mid \lambda$$

Intersection

$L_1 \cap L_2 = \{a^n b^n c^n\}$ **NOT** context-free

Complement

Context-free languages
are **not** closed under: **complement**

L is context free $\Rightarrow$ $\overline{L}$ **not** necessarily context-free

Example

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

Context-free:

$$S \to AC$$

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

$$C \to cC \mid \lambda$$

$$L_2 = \{a^n b^m c^m\}$$

Context-free:

$$S \to AB$$

$$A \to aA \mid \lambda$$

$$B \to bBc \mid \lambda$$

Complement

$$\overline{\overline{L_1} \cup \overline{L_2}} = L_1 \cap L_2 = \{a^n b^n c^n\}$$

NOT context-free

Intersection
of
Context-free languages
and
Regular Languages

The intersection of

a context-free language and

a regular language

is a context-free language

L_1 context free

L_2 regular

$\Rightarrow$ $L_1 \cap L_2$

context-free

Machine M_1

NPDA for L_1

context-free

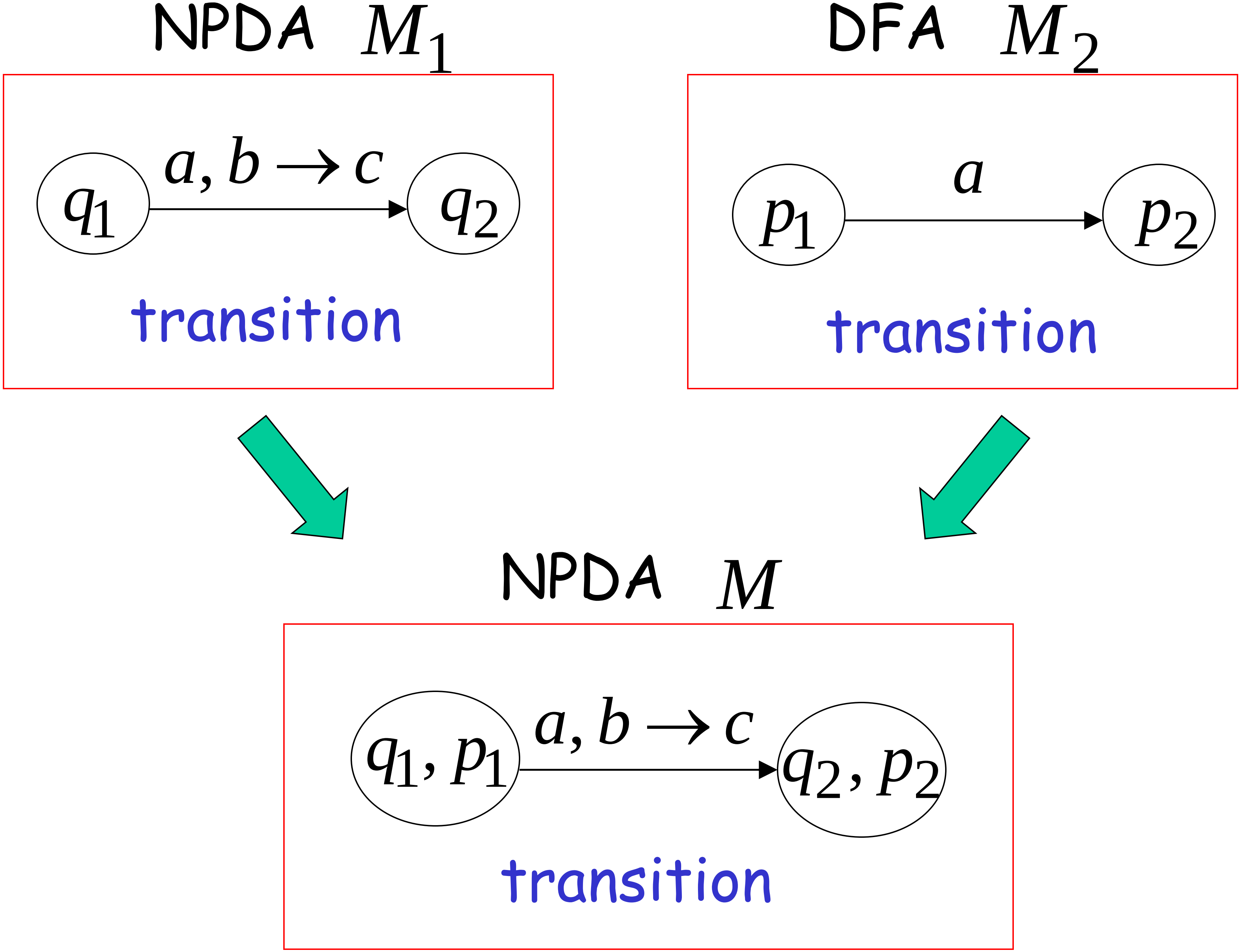
Machine M_2

DFA for L_2

regular

Construct a new NPDA machine M that accepts $L_1 \cap L_2$

M simulates in parallel M_1 and M_2



NPDA M_1

transition

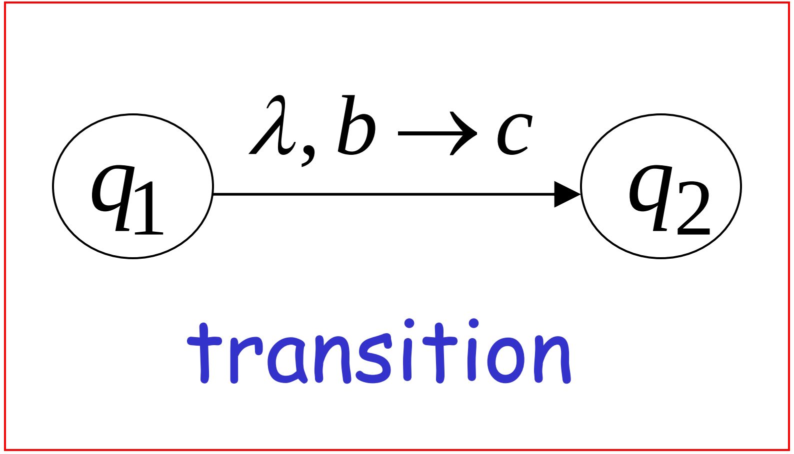
DFA M_2

transition

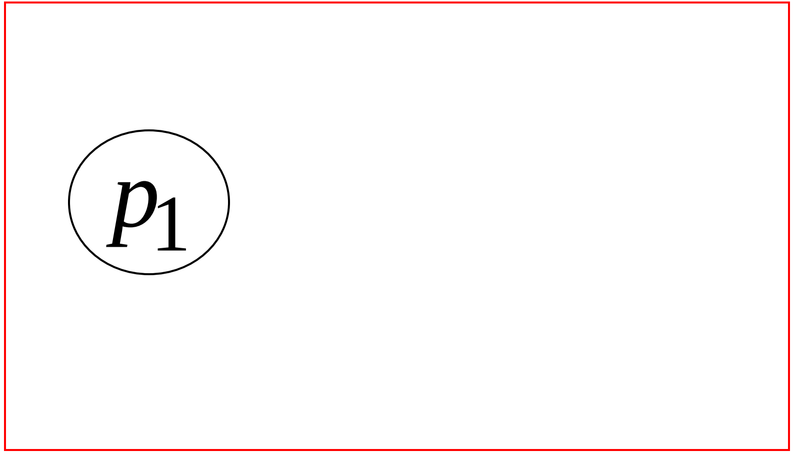
NPDA M

transition

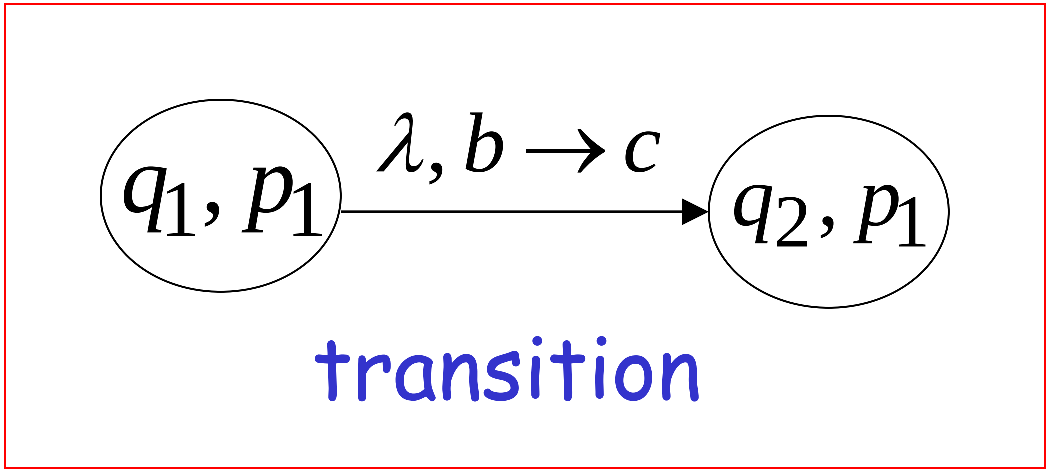
NPDA M_1

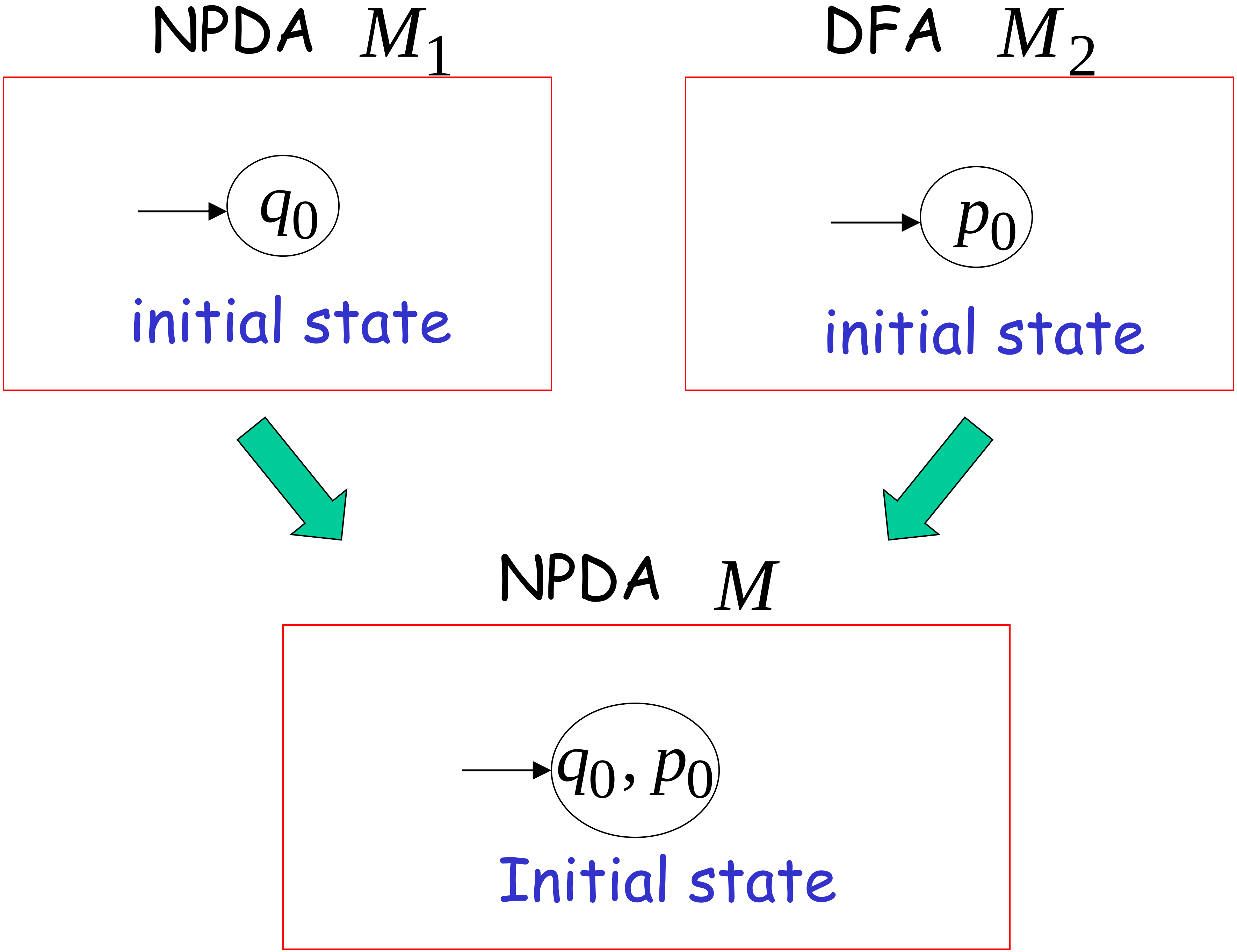


DFA M_2



NPDA M





NPDA M_1

q_0

initial state

DFA M_2

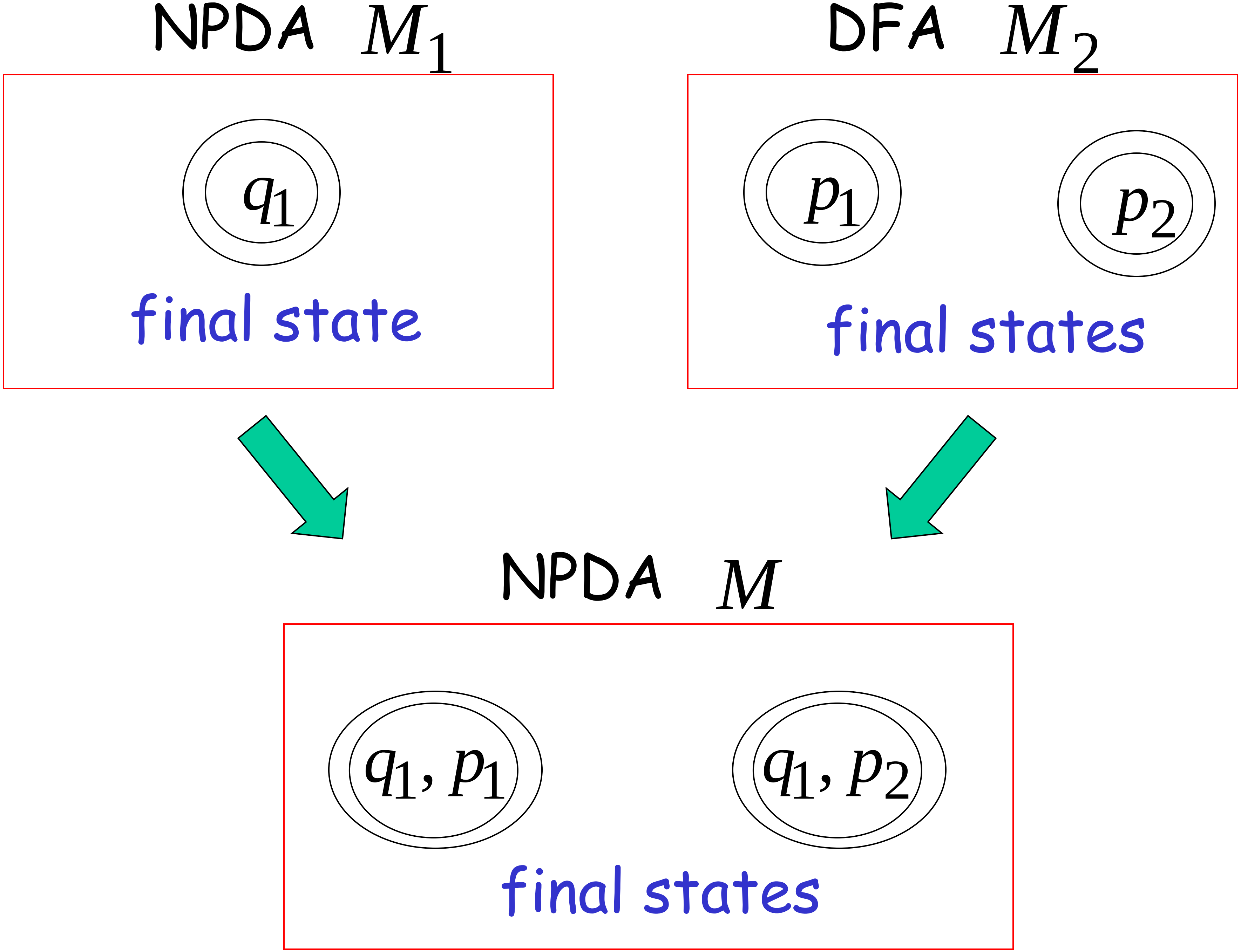
p_0

initial state

NPDA M

q_0, p_0

Initial state



NPDA M_1

q_1

final state

DFA M_2

p_1

p_2

final states

NPDA M

q_1, p_1

q_1, p_2

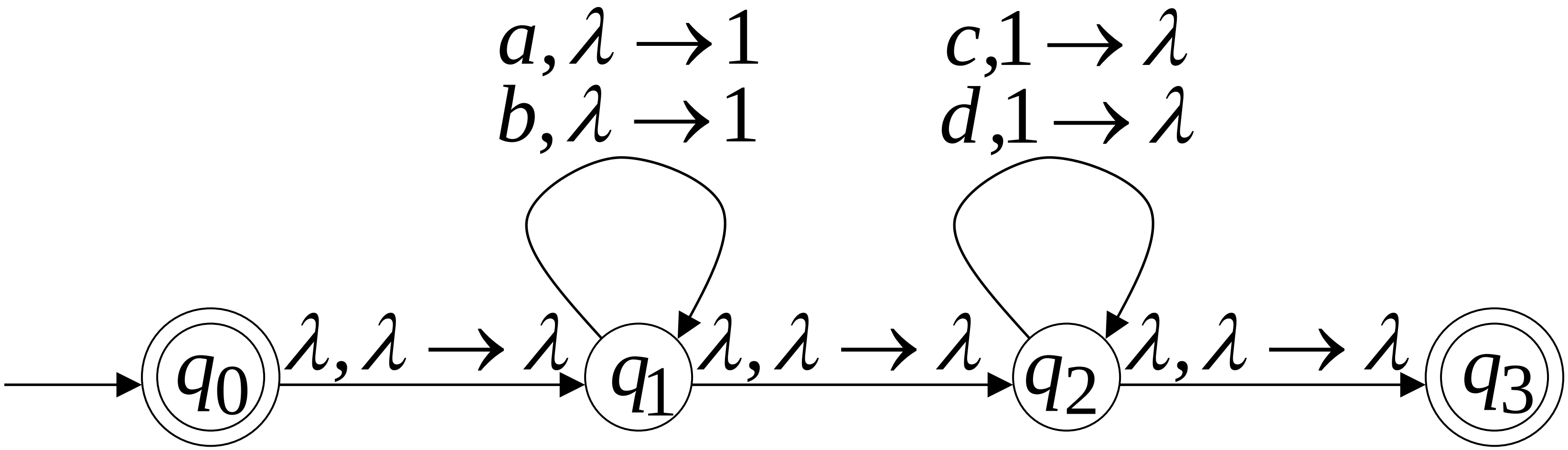
final states

Example:

context-free

$$L_1 = \{w_1 w_2 : |w_1| = |w_2|,\ w_1 \in \{a,b\}^*,\ w_2 \in \{c,d\}^*\}$$

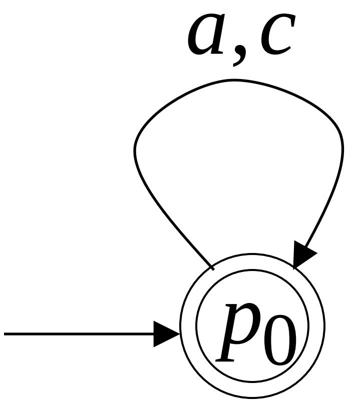
NPDA M_1



regular

$$L_2 = \{a, c\}^*$$

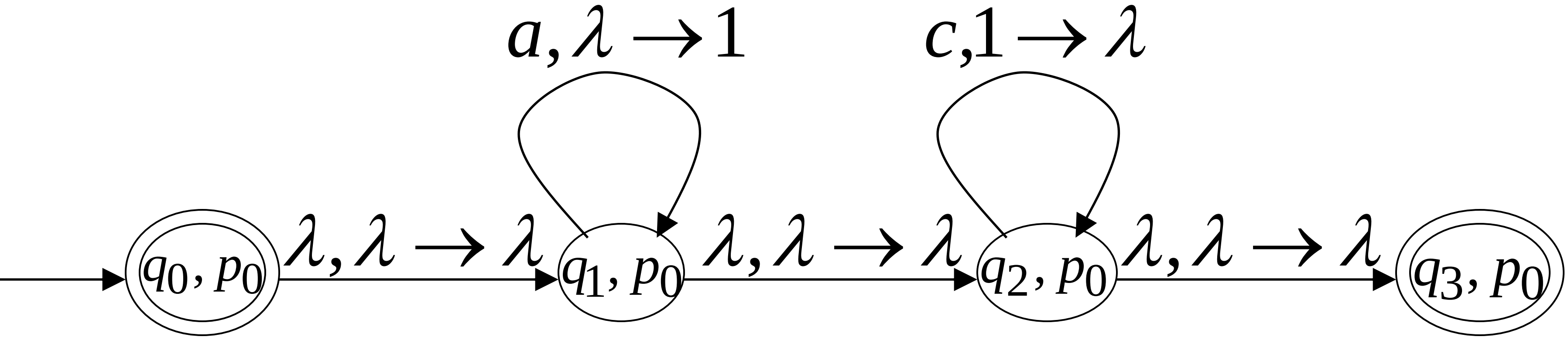
DFA M_2



context-free

Automaton for: $L_1 \cap L_2 = \{a^n c^n : n \geq 0\}$

NPDA M



In General:

M simulates in parallel M_1 and M_2

M accepts string w if and only if

M_1 accepts string w and

M_2 accepts string w

$$L(M) = L(M_1) \cap L(M_2)$$

Therefore:

M is NPDA

$$\Downarrow$$

$L(M_1) \cap L(M_2)$ is context-free

$$\Downarrow$$

$L_1 \cap L_2$ is context-free

Applications
of
Regular Closure

The intersection of

a context-free language and

a regular language

is a context-free language

Regular Closure

L_1 context free

L_2 regular

$L_1 \cap L_2$

context-free

An Application of Regular Closure

Prove that: $L = \{a^n b^n : \quad n \neq 100, n \geq 0\}$

is context-free

We know:

$\{a^n b^n : n \geq 0\}$ is context-free

We also know:

$$L_1 = \{a^{100}b^{100}\}$$ is regular

↓

$$\overline{L_1} = \{(a+b)^*\} - \{a^{100}b^{100}\}$$ is regular

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

context-free

$\overline{L_1} = \{(a+b)^*\} - \{a^{100} b^{100}\}$

regular

(regular closure) $\{a^n b^n\} \cap \overline{L_1}$ context-free

$$\{a^n b^n\} \cap \overline{L_1} = \{a^n b^n : \quad n \neq 100, n \geq 0\} = L$$

is context-free

Another Application of Regular Closure

Prove that: $L = \{w : \ n_a = n_b = n_c\}$

is **not** context-free

If $L = \{w: \ n_a = n_b = n_c\}$ is context-free

(regular closure)

Then $L \cap \{a^*b^*c^*\} = \{a^n b^n c^n\}$

context-free regular context-free

Impossible!!!

Therefore, L is **not** context free

Pumping Lemma
for
Context-free Languages

Take an **infinite** context-free language

Generates an infinite number of different strings

Example:

$$S \to ABE \mid bBd$$

$$A \to Aa \mid a$$

$$B \to bSD \mid cc$$

$$D \to Dd \mid d$$

$$E \to eE \mid e$$

In a derivation of a "long" enough string, variables are repeated

A possible derivation:

$$S \Rightarrow A\boxed{B}E \Rightarrow AaBE \Rightarrow aaBE$$

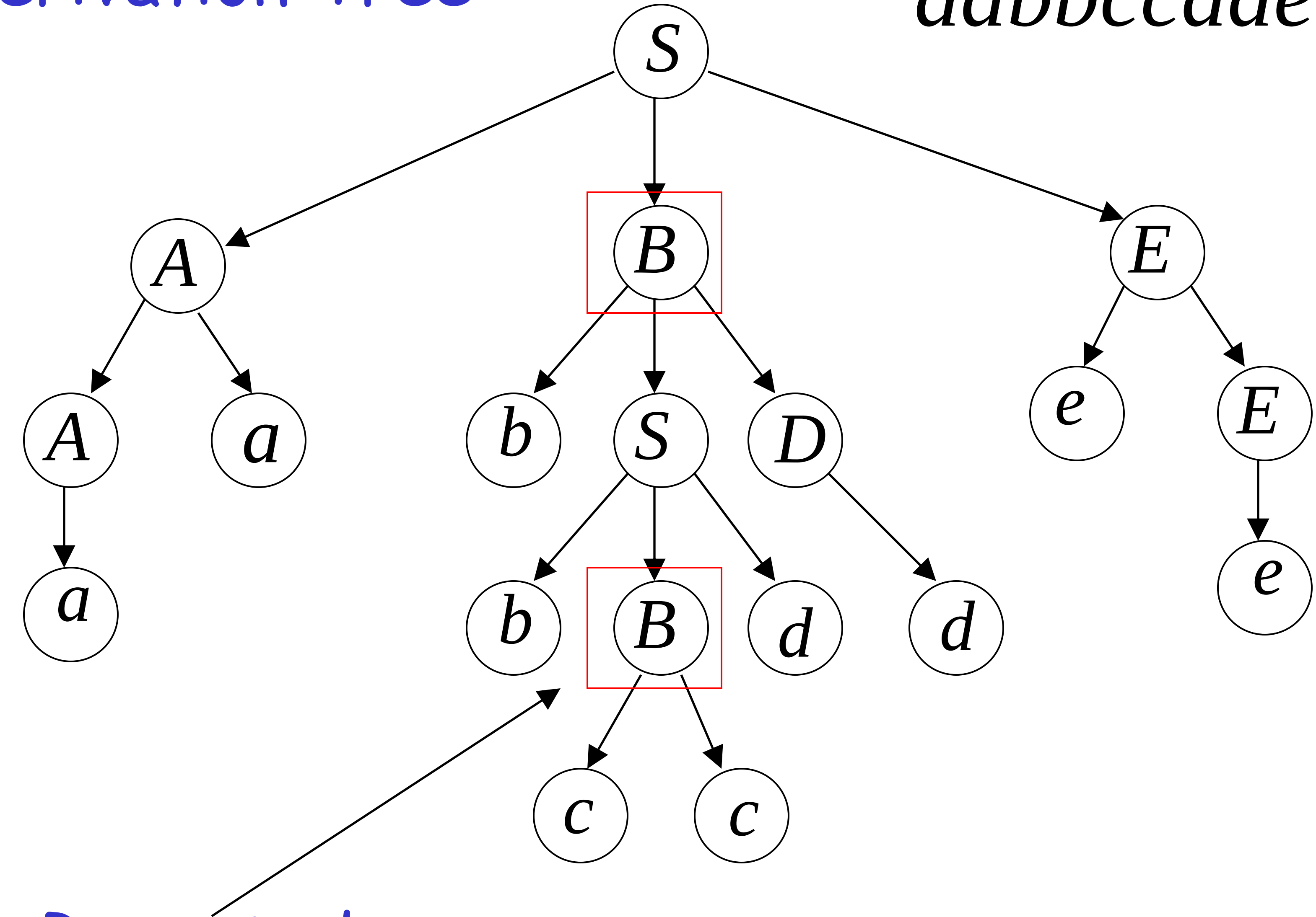
$$\Rightarrow aabSDE \Rightarrow aabb\boxed{B}dDE \Rightarrow$$

$$\Rightarrow aaabbccdDE \Rightarrow aabbccddE$$

$$\Rightarrow aabbccddeE \Rightarrow aabbccddee$$

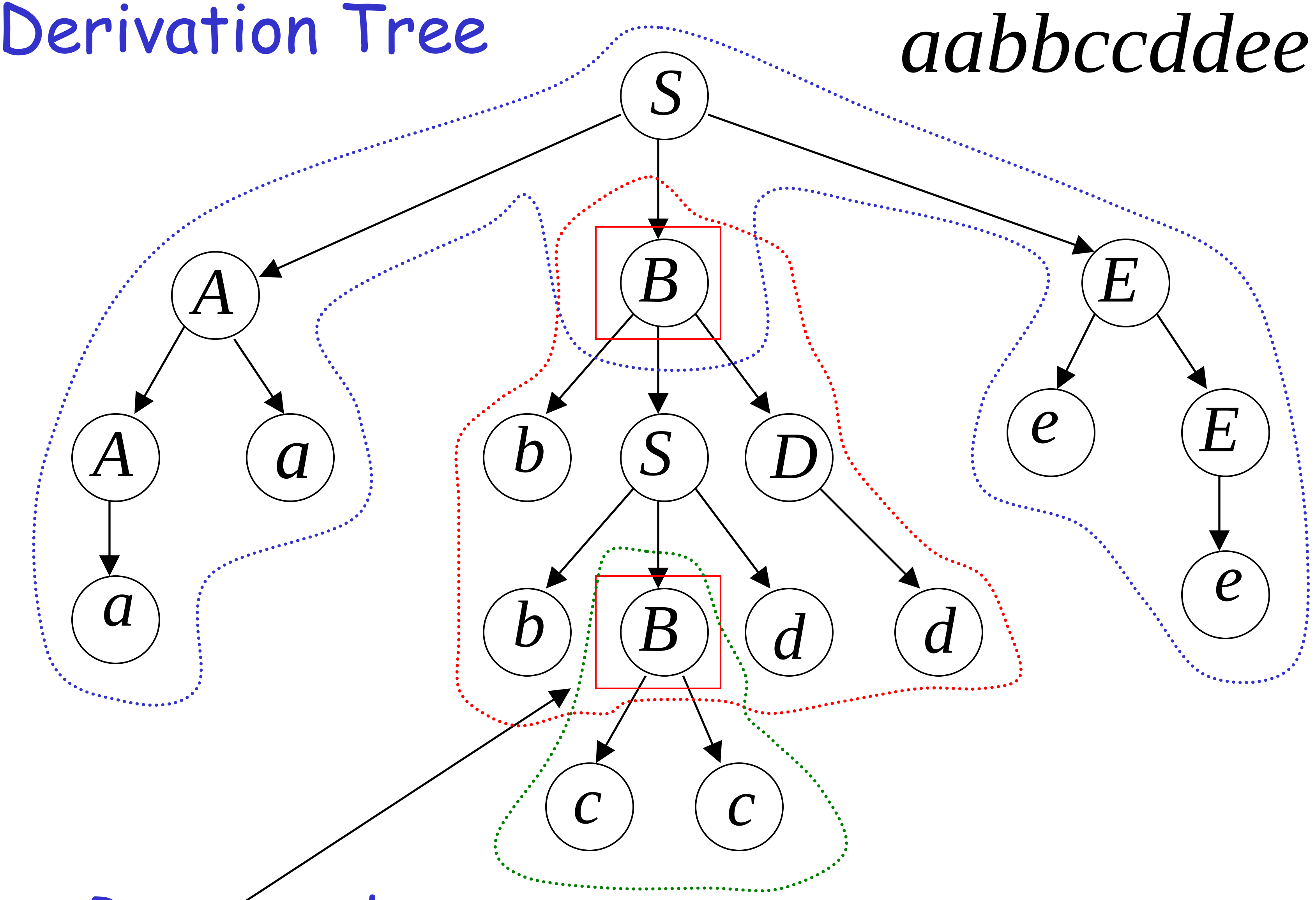
Derivation Tree

aabbccddee



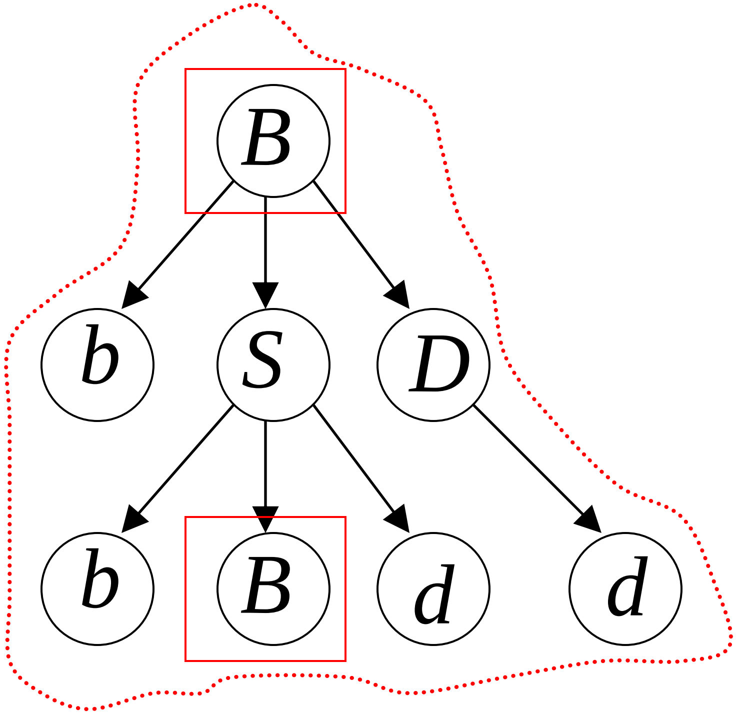
Repeated variable

Derivation Tree



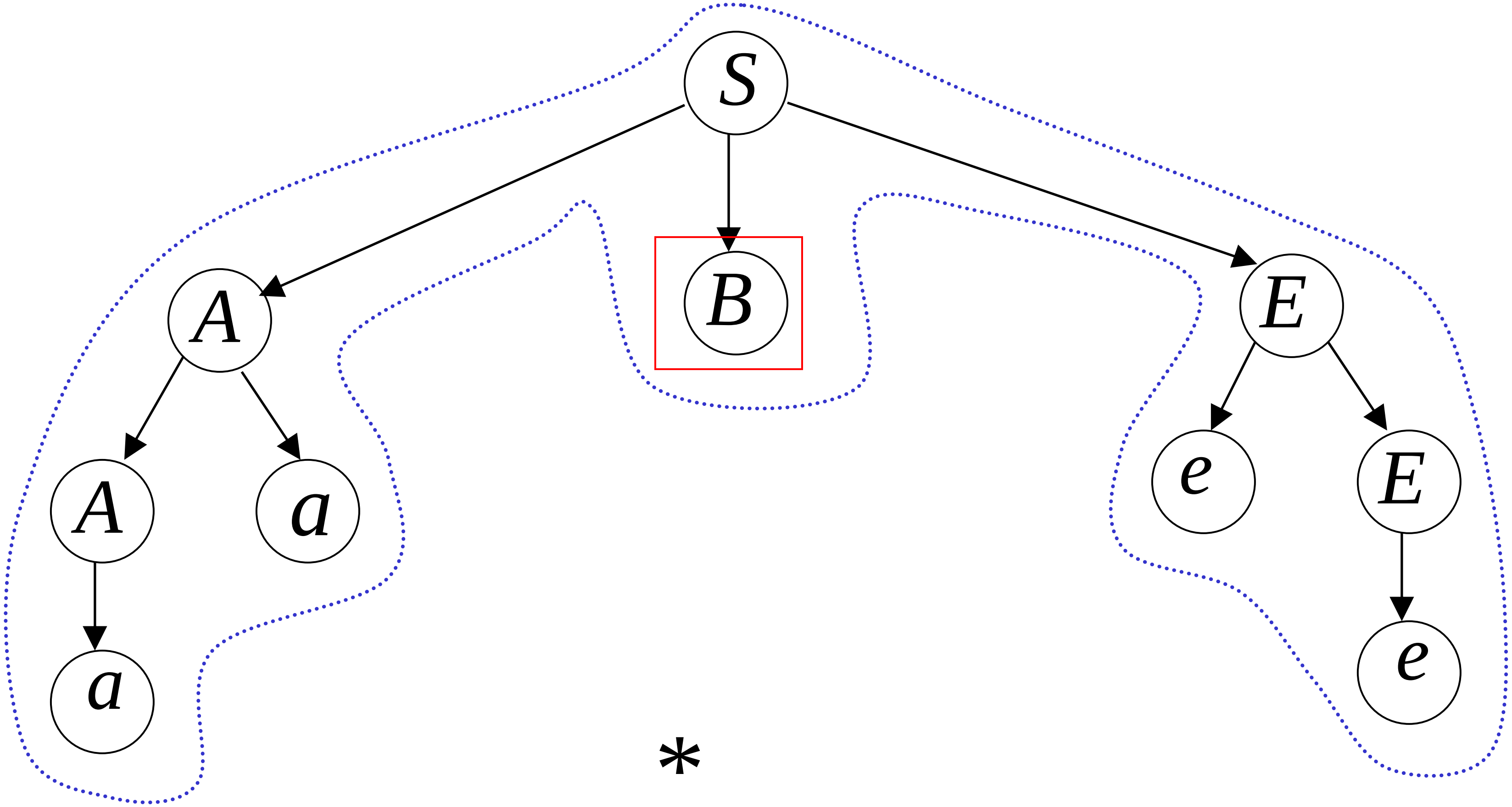
Repeated variable

$$B \Rightarrow bSD \Rightarrow bbBdD \Rightarrow bbBdd$$

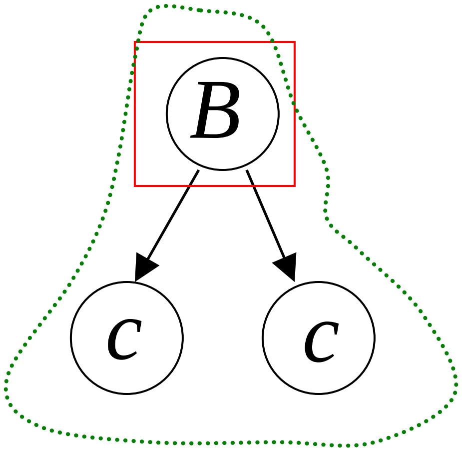


$$B \overset{*}{\Rightarrow} bbBdd$$

$$S \Rightarrow ABE \Rightarrow AaBE \Rightarrow aaBE \Rightarrow aaBeE \Rightarrow aaBee$$

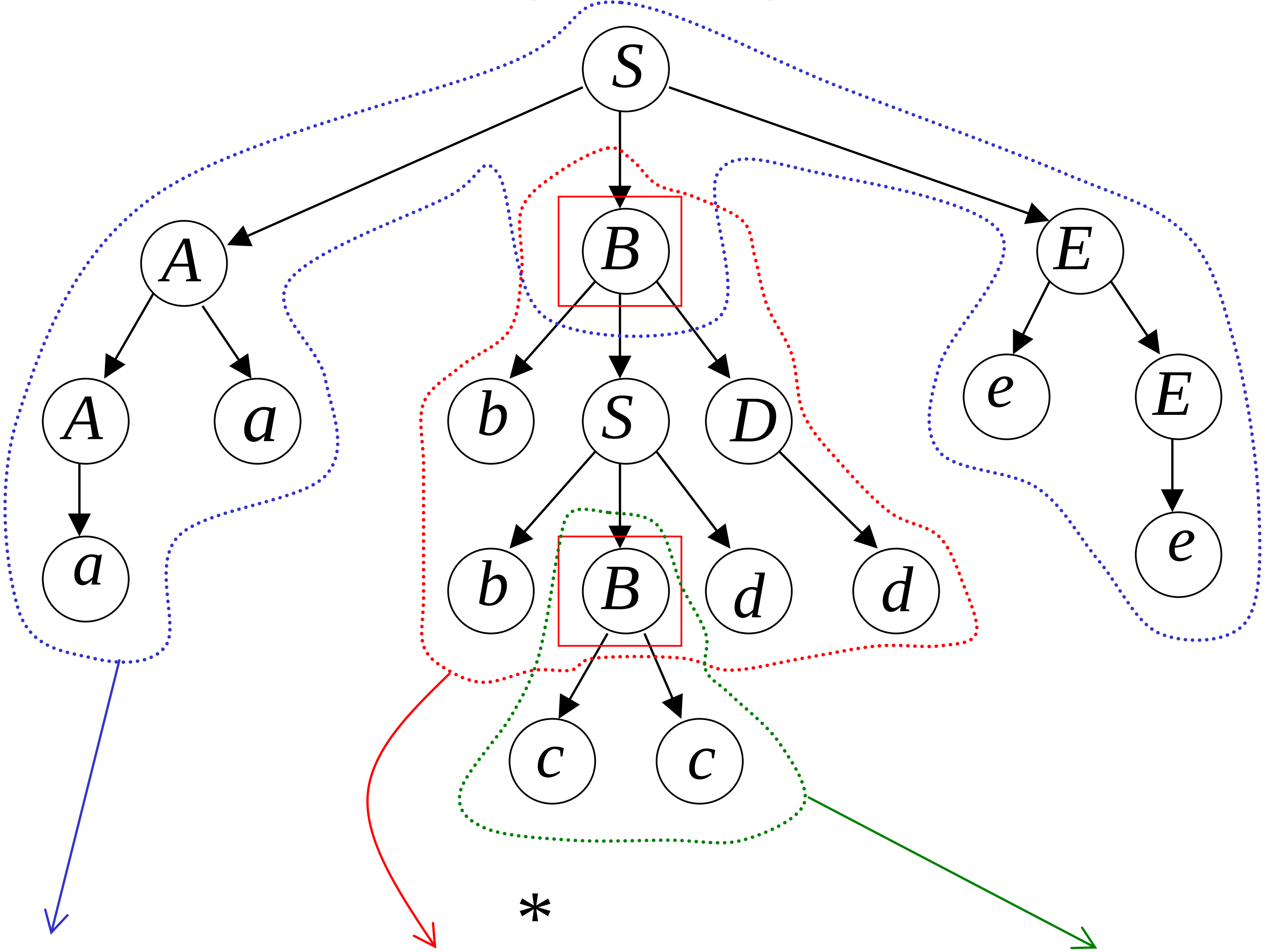


$$S \overset{*}{\Rightarrow} aaBee$$



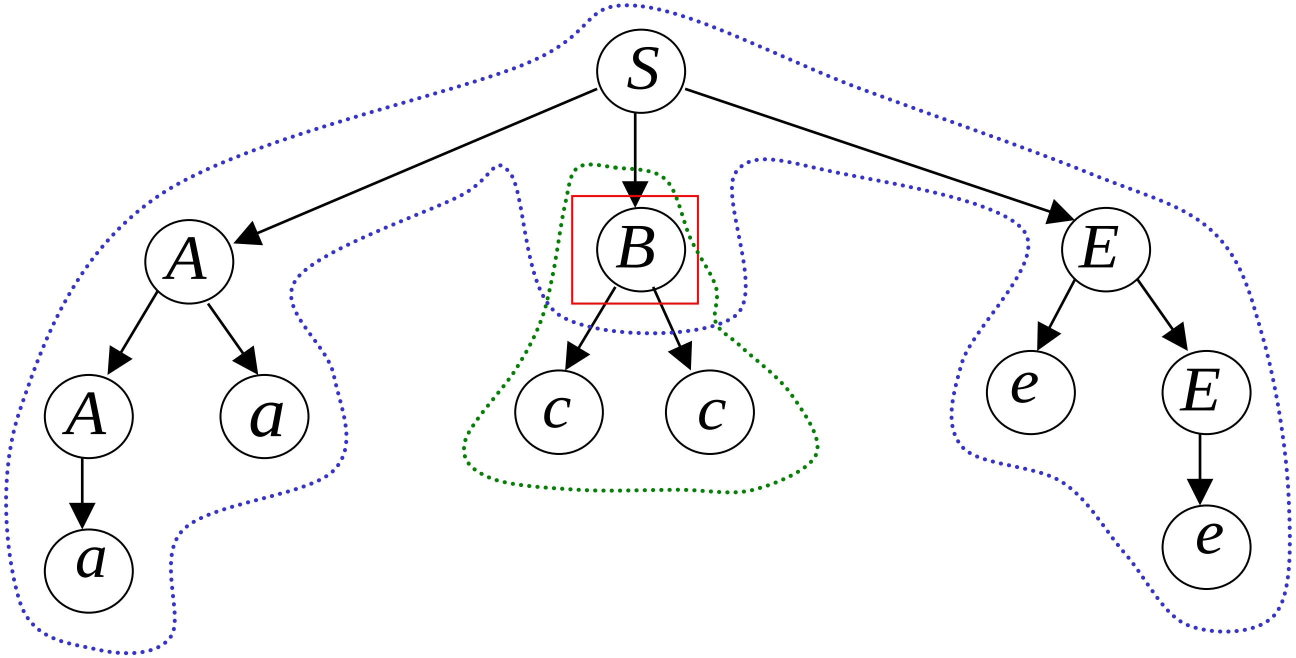
$$B \Rightarrow cc$$

Putting all together



$S \overset{*}{\Rightarrow} aaBee$ $B \overset{*}{\Rightarrow} bbBdd$ $B \Rightarrow cc$

We can remove the middle part



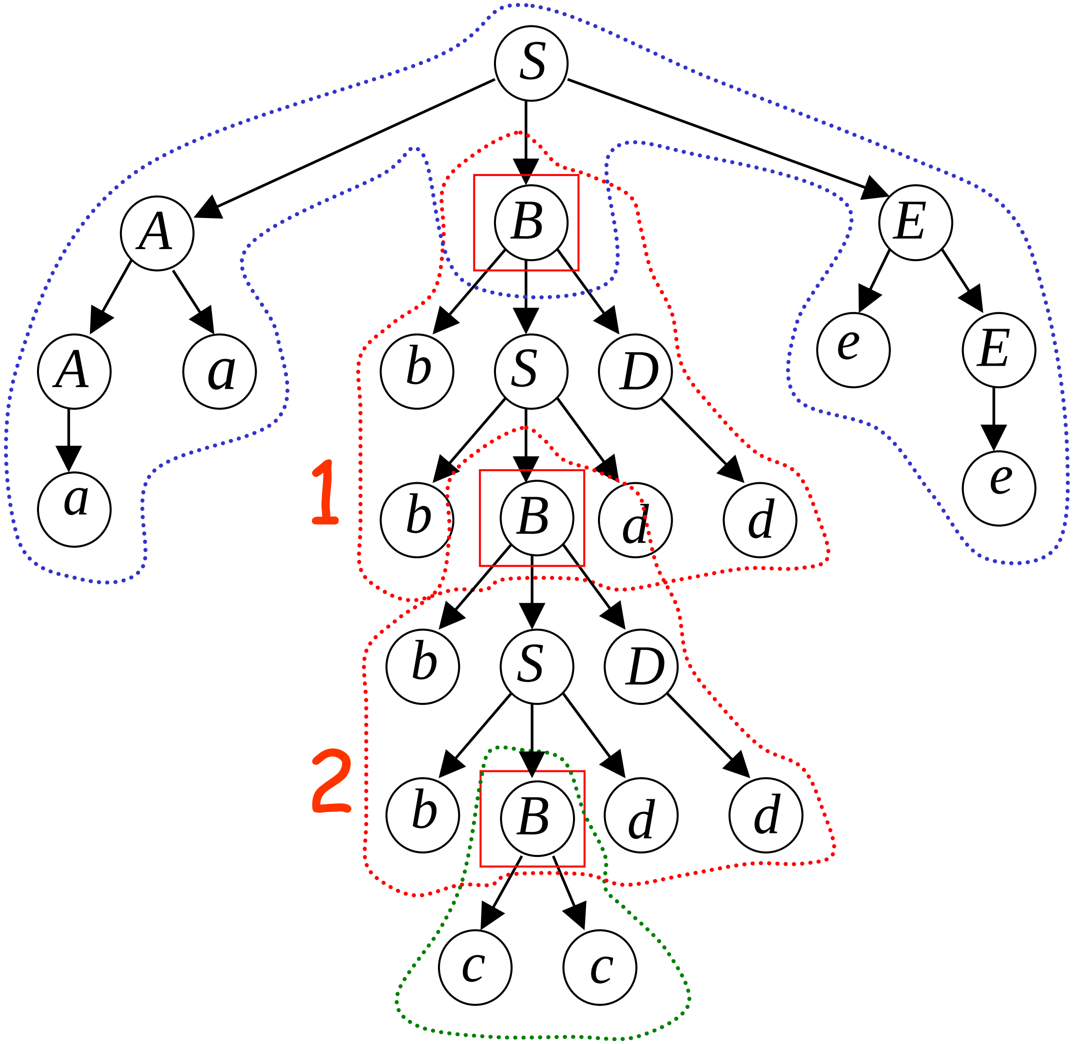
$$S \overset{*}{\Rightarrow} aa(bb)^0 cc(dd)^0 ee$$

$$S \overset{*}{\Rightarrow} aaBee \qquad B \overset{*}{\Rightarrow} bbBdd \qquad B \Rightarrow cc$$

$$S \overset{*}{\Rightarrow} aaBee \overset{*}{\Rightarrow} aaccee \quad = aa(bb)^0 cc(dd)^0 ee$$

$$aa(bb)^0 cc(dd)^0 ee \quad \in L(G)$$

We can repeated middle part two times



$$S \overset{*}{\Rightarrow} aa(bb)^2cc(dd)^2ee$$

$$S \overset{*}{\Rightarrow} aaBee \qquad B \overset{*}{\Rightarrow} bbBdd \qquad B \Rightarrow cc$$

⇩

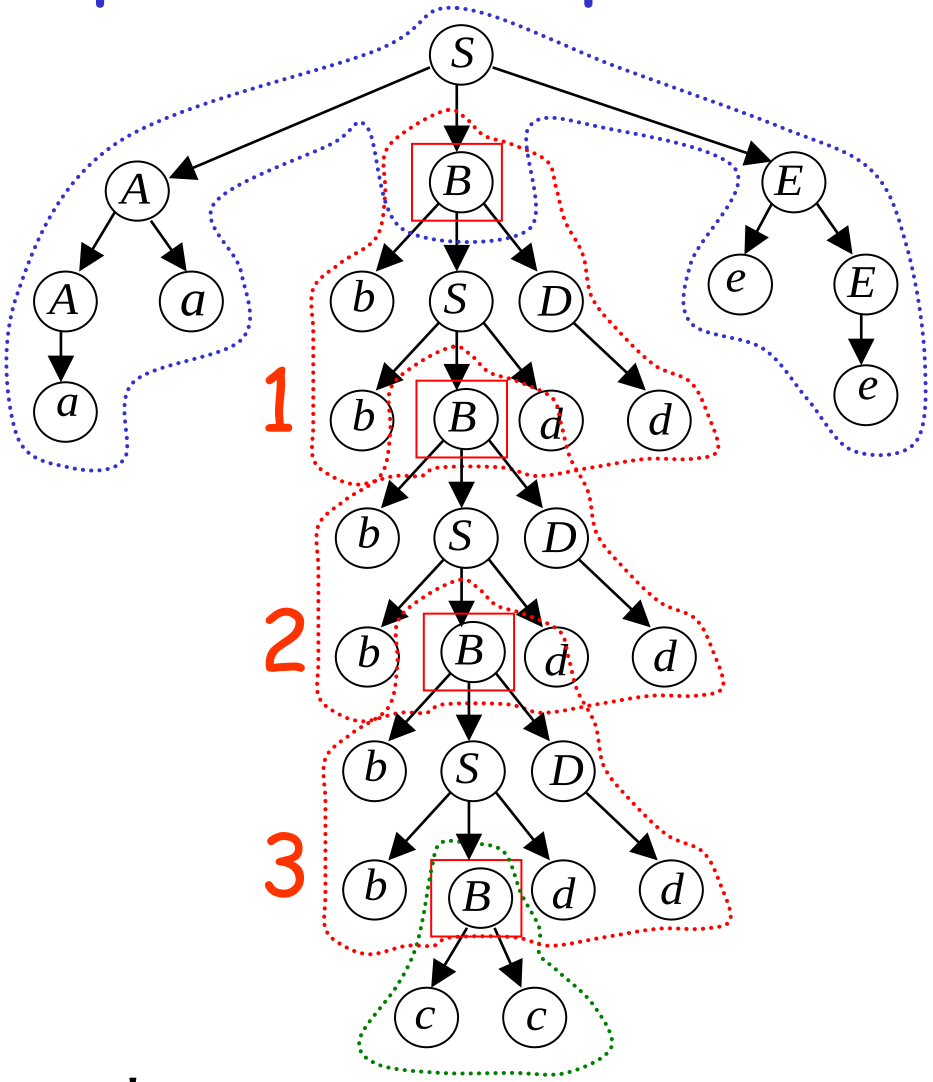
$$S \overset{*}{\Rightarrow} aaBee \overset{*}{\Rightarrow} aabbBddee$$

$$\overset{*}{\Rightarrow} aa(bb)^2 B(dd)^2 ee \overset{*}{\Rightarrow} aa(bb)^2 cc(dd)^2 ee$$

⇩

$$aa(bb)^2 cc(dd)^2 ee \quad \in L(G)$$

We can repeat middle part three times



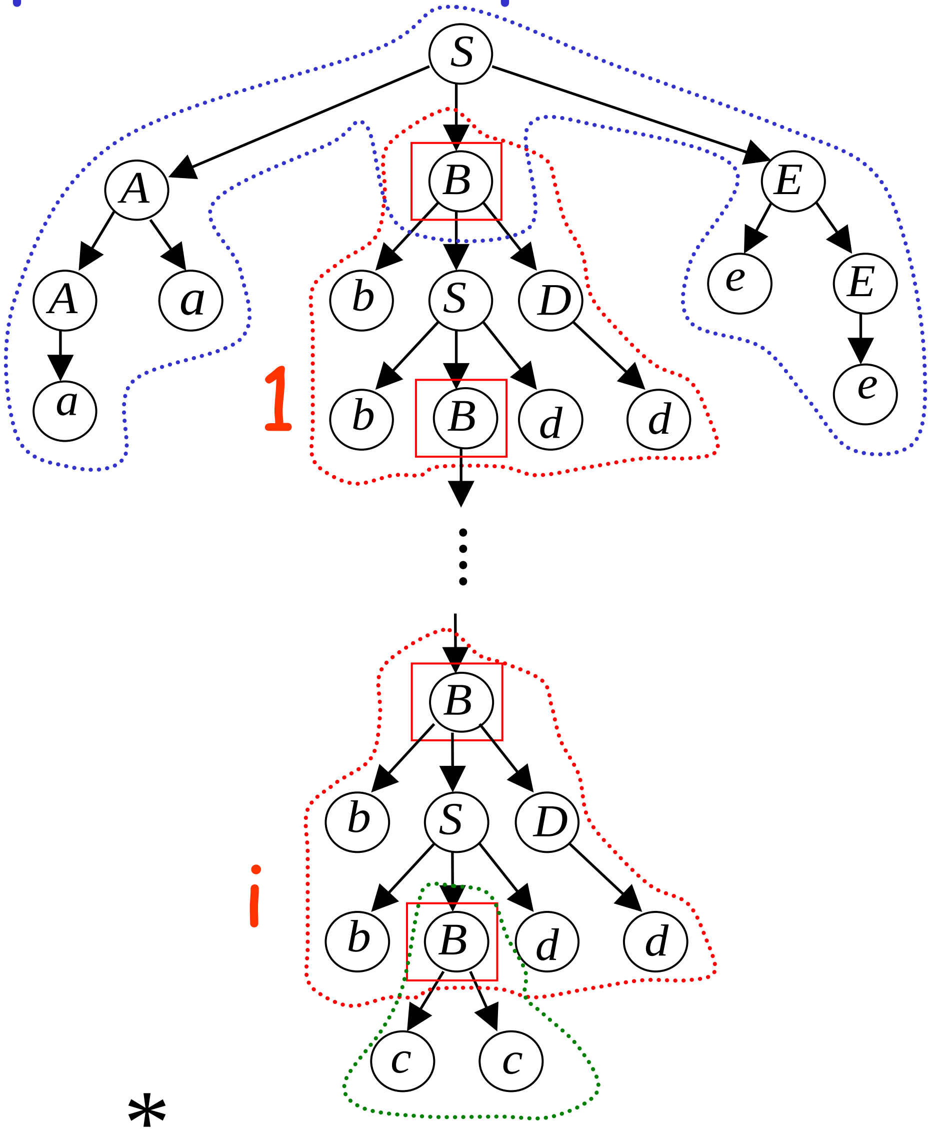
$$S \overset{*}{\Rightarrow} aa(bb)^3cc(dd)^3ee$$

$$S \overset{*}{\Rightarrow} aaBee \qquad B \overset{*}{\Rightarrow} bbBdd \qquad B \Rightarrow cc$$

⇓

$$S \overset{*}{\Rightarrow} aa(bb)^3cc(dd)^3ee \quad \in L(G)$$

Repeat middle part i times



$$S \overset{*}{\Rightarrow} aa(bb)^i cc(dd)^i ee$$

$$S \overset{*}{\Rightarrow} aaBee \qquad B \overset{*}{\Rightarrow} bbBdd \qquad B \Rightarrow cc$$

⬇

$$S \overset{*}{\Rightarrow} aa(bb)^i cc(dd)^i ee \quad \in L(G)$$

For any $i \geq 0$

From Grammar

and given string

$$S \to ABE \mid bBd$$

$$A \to Aa \mid a$$

$$B \to bSD \mid cc$$

$$D \to Dd \mid d$$

$$E \to eE \mid e$$

$$aabbccddee \quad \in L(G)$$

We inferred that a family of strings is in $L(G)$

$$S \overset{*}{\Rightarrow} aa(bb)^i cc(dd)^i ee \quad \in L(G)$$ for any $i \geq 0$

Arbitrary Grammars

Consider now an arbitrary infinite context-free language L

Let G be the grammar of $L-\{\varepsilon\}$

Take G so that it has no unit-productions and no ε-productions

(remove them)

Let r be the number of variables

Let t be the maximum right-hand size of any production

Example:

$$
\begin{aligned}
S &\to ABE \mid \boxed{bBd} \\
A &\to Aa \mid a \\
B &\to bSD \mid cc \\
D &\to Dd \mid d \\
E &\to eE \mid e
\end{aligned}
$$

$r = 5$ (variables)

$t = 3$

Claim:

Take string $w \in L(G)$ with $|w| > t^r$.
Then in the derivation tree of w
there is a path from the root to a leaf
where a variable of G is repeated

Proof:

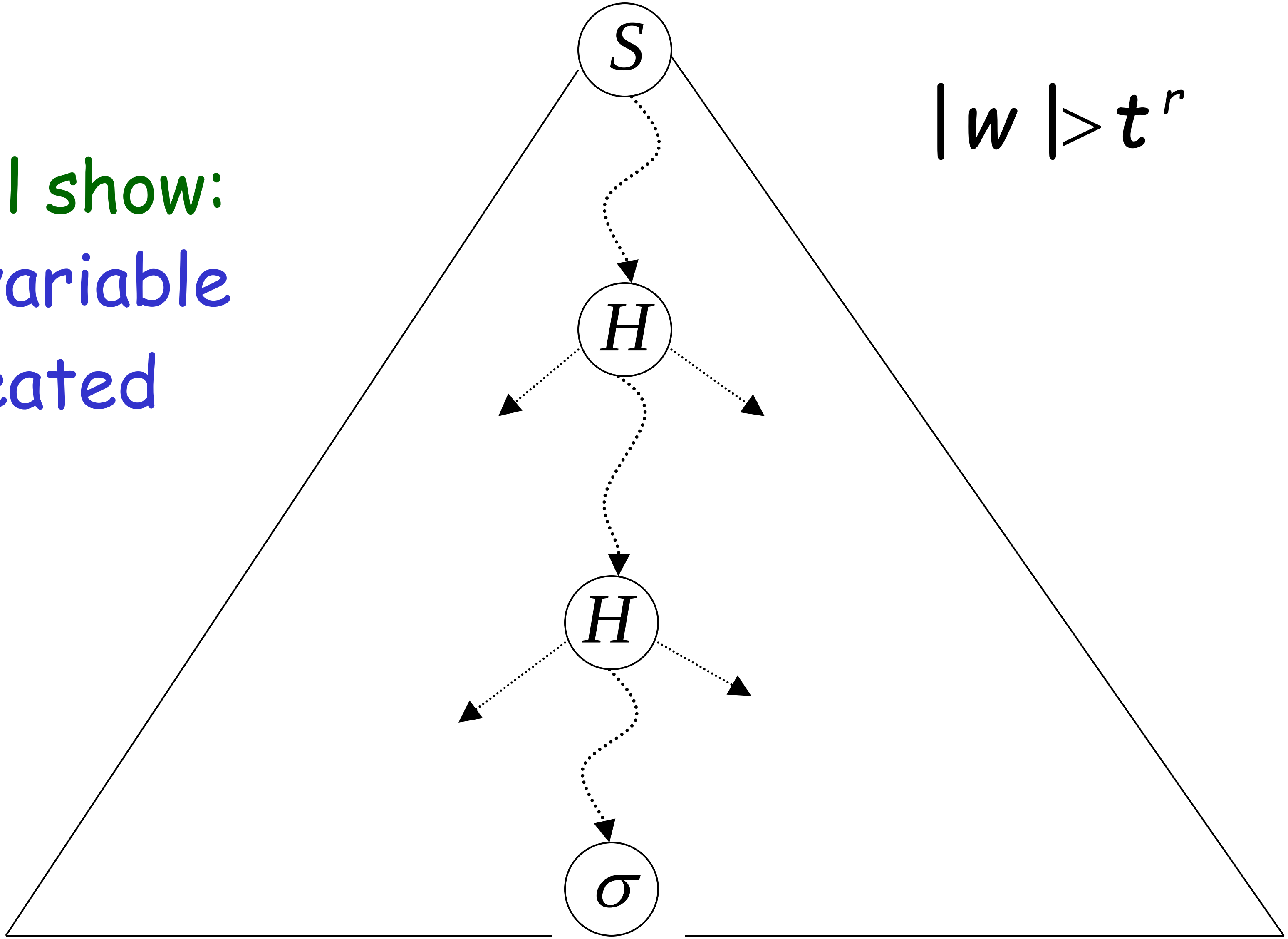
Proof by contradiction

Derivation tree of w

$|w| > t^r$

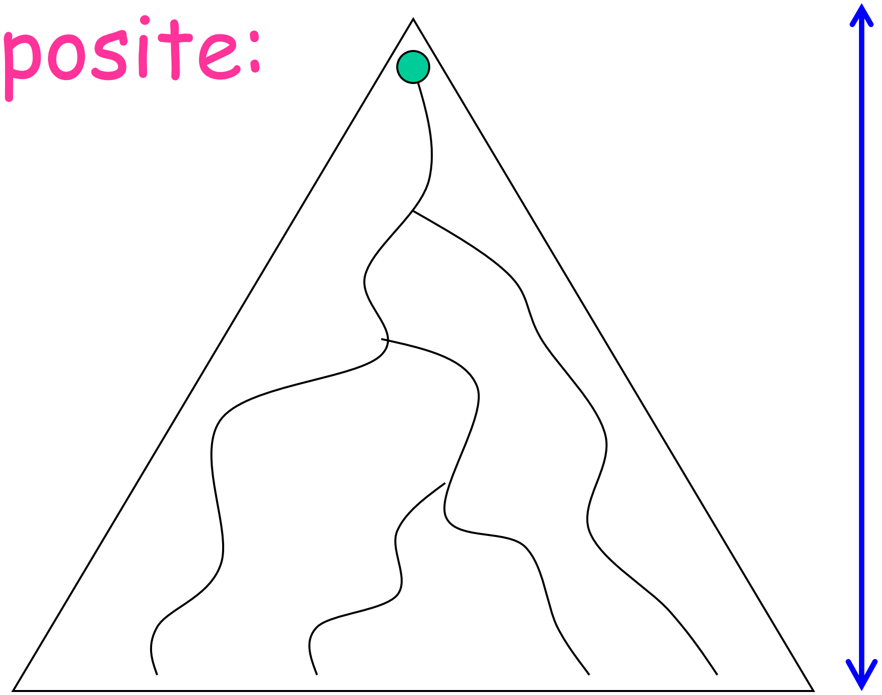
We will show:

some variable is repeated



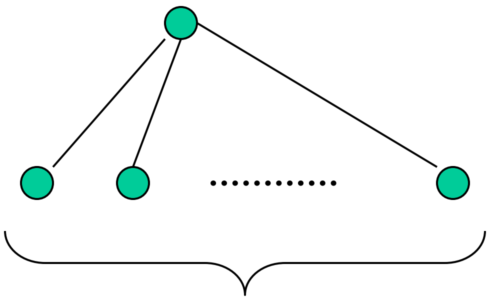
First we show that the tree of w has at least $r+2$ levels of nodes

Suppose the opposite:



At most $r+1$ Levels

Maximum number of nodes per level



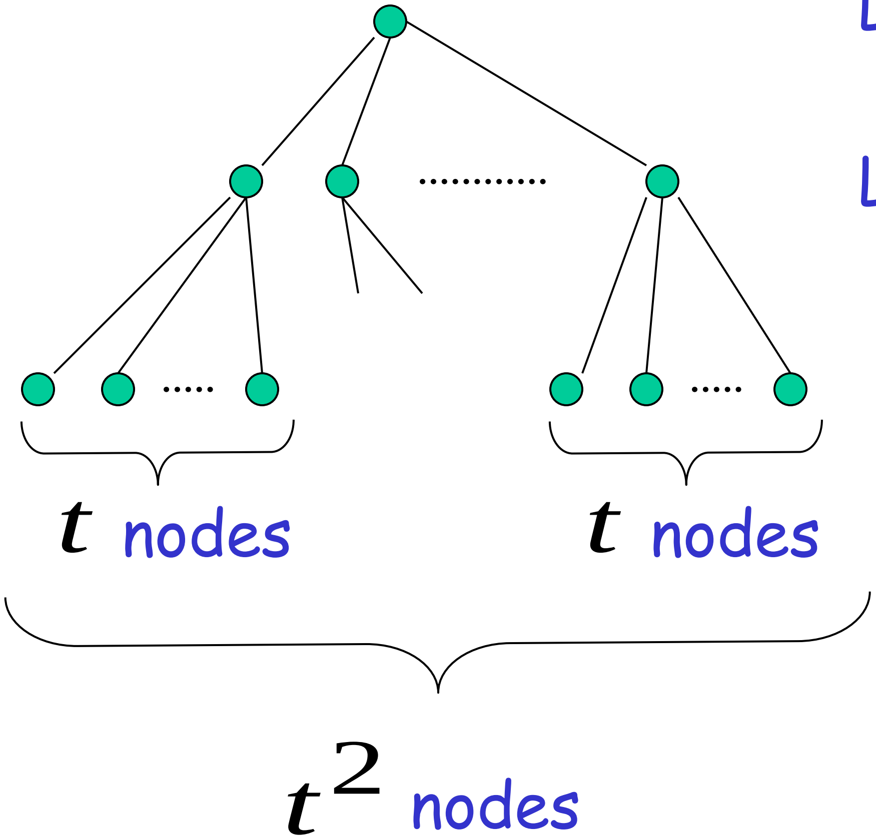
t nodes

Level 0: 1 nodes

Level 1: t nodes

The maximum right-hand side of any production

Maximum number of nodes per level

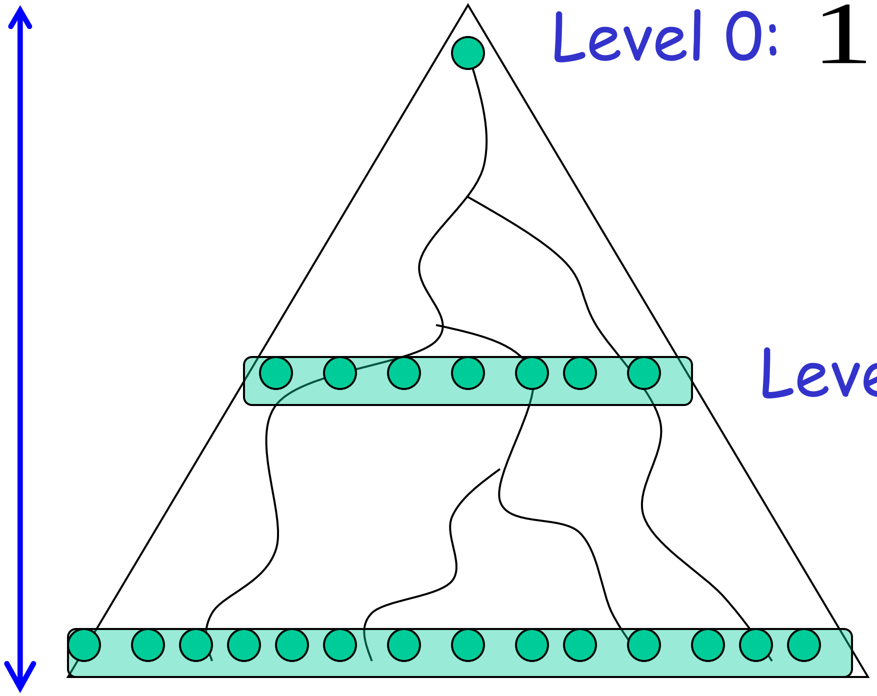


Level 0: 1 nodes

Level 1: t nodes

Level 2: t^2 nodes

Maximum number of nodes per level



At most $r+1$ Levels

Level 0: 1 nodes

Level i : t^{i} nodes

Level r : t^{r} nodes

Maximum possible string length

= max nodes at level r = $\boxed{t^{r}}$

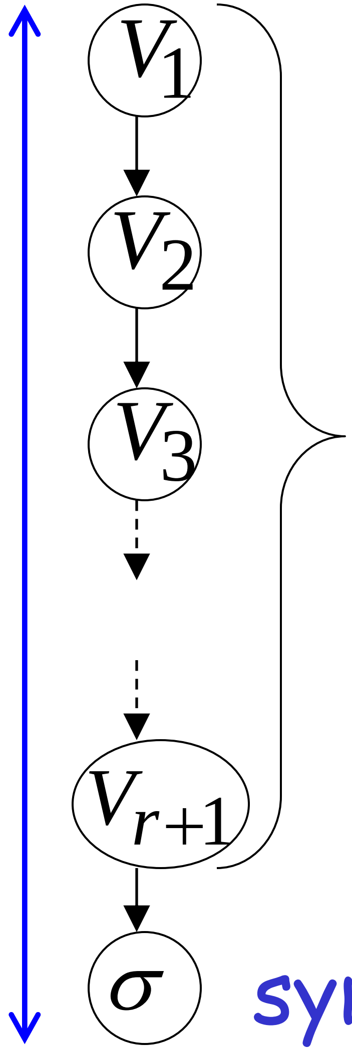
Therefore,
maximum length of string w : $|w| \leq t^r$

However we took, $|w| > t^r$

Contradiction!!!

Therefore,
the tree must have at least $r+2$ levels

Thus, there is a path from the root to a leaf with at least $r+2$ nodes



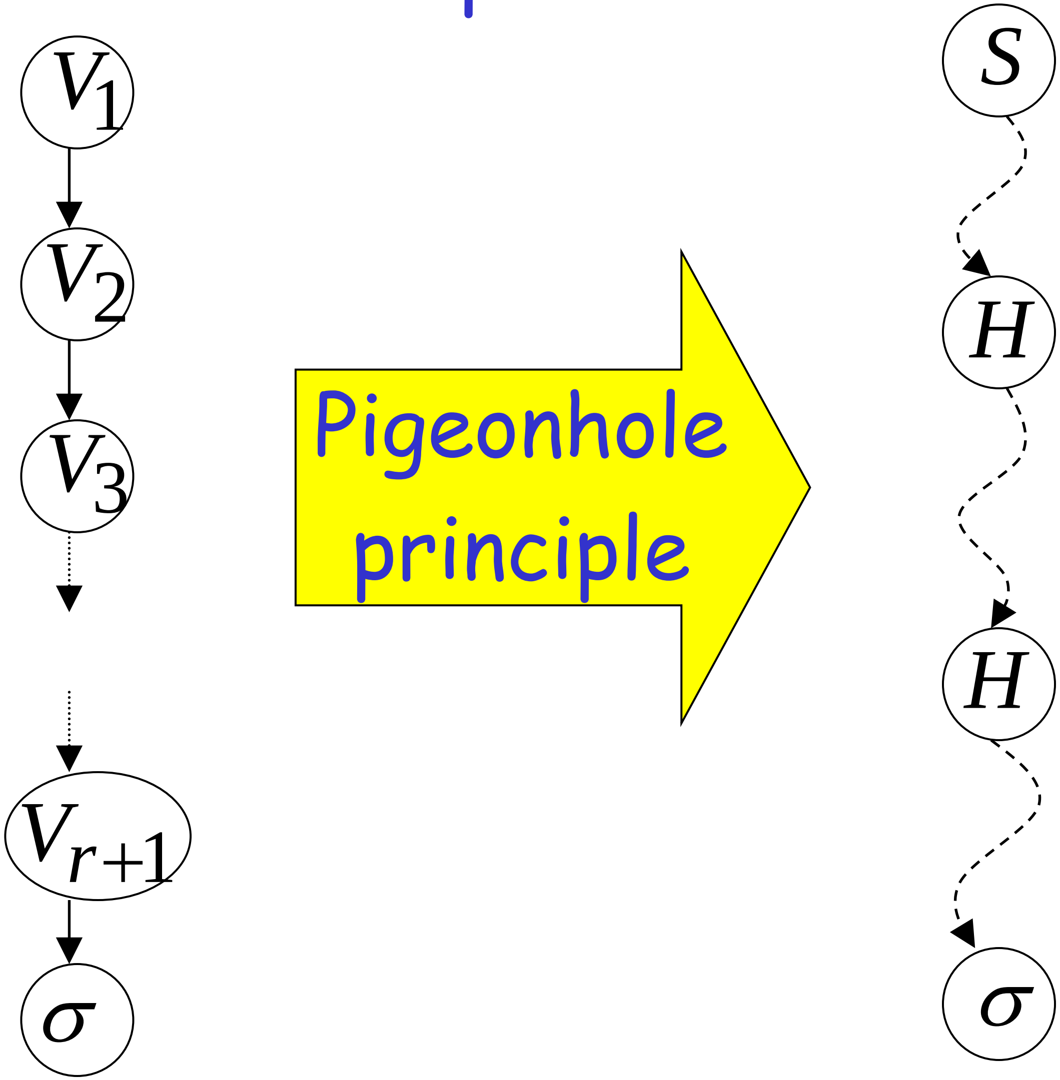
At least
$r+2$
Levels

$V_1 = S$ (root)

$r+1$ Variables

symbol (leaf)

Since there are at most r different variables, some variable is repeated



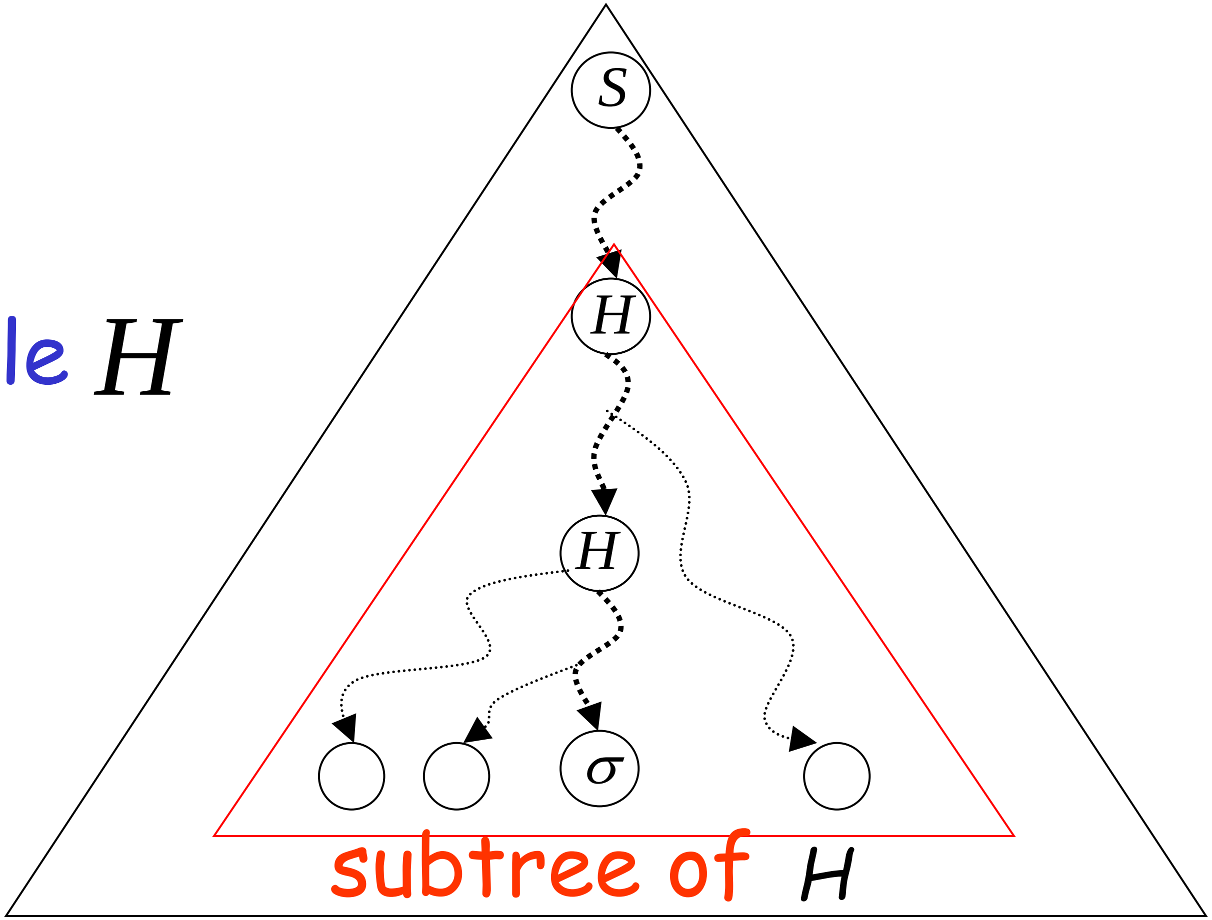
Pigeonhole principle

END OF CLAIM PROOF

Take now a string w with $|w| > t^r$

From claim:

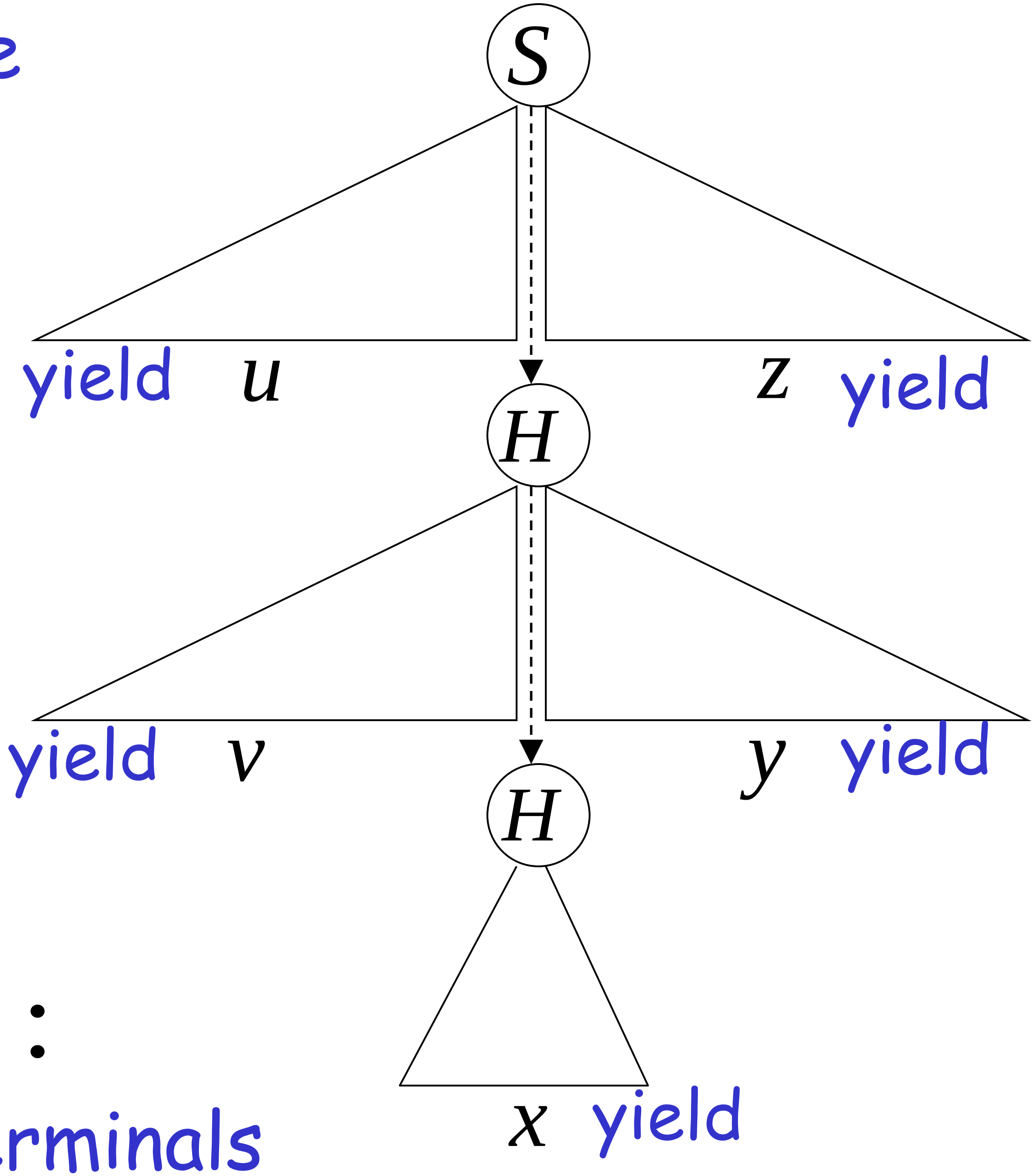
some variable H is repeated



Take H to be the deepest, so that only H is repeated in subtree

We can write

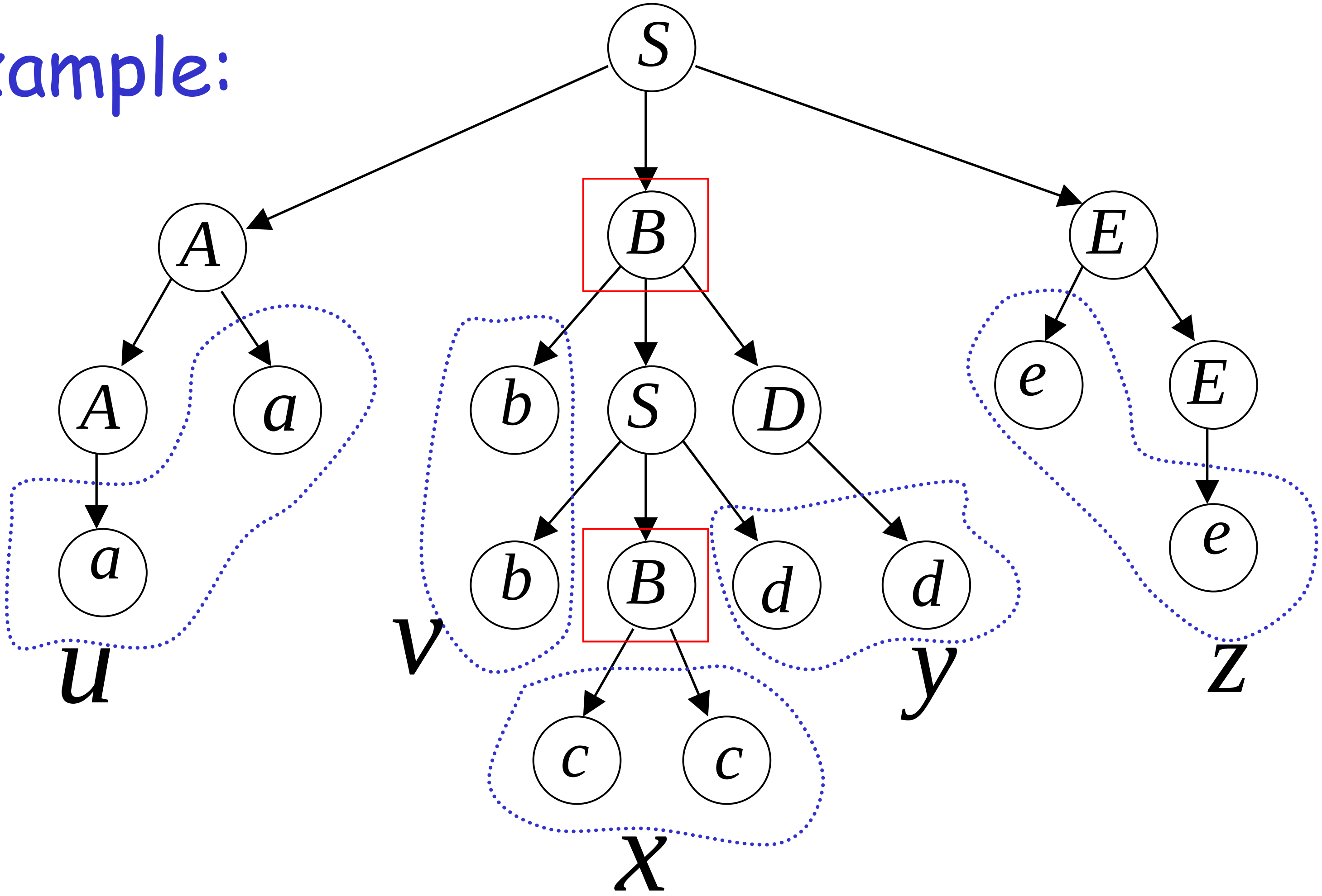
$w = uvxyz$



$u, v, x, y, z:$

Strings of terminals

Example:



$u = aa$

$v = bb$

$x = cc$

$y = dd$

$z = ee$

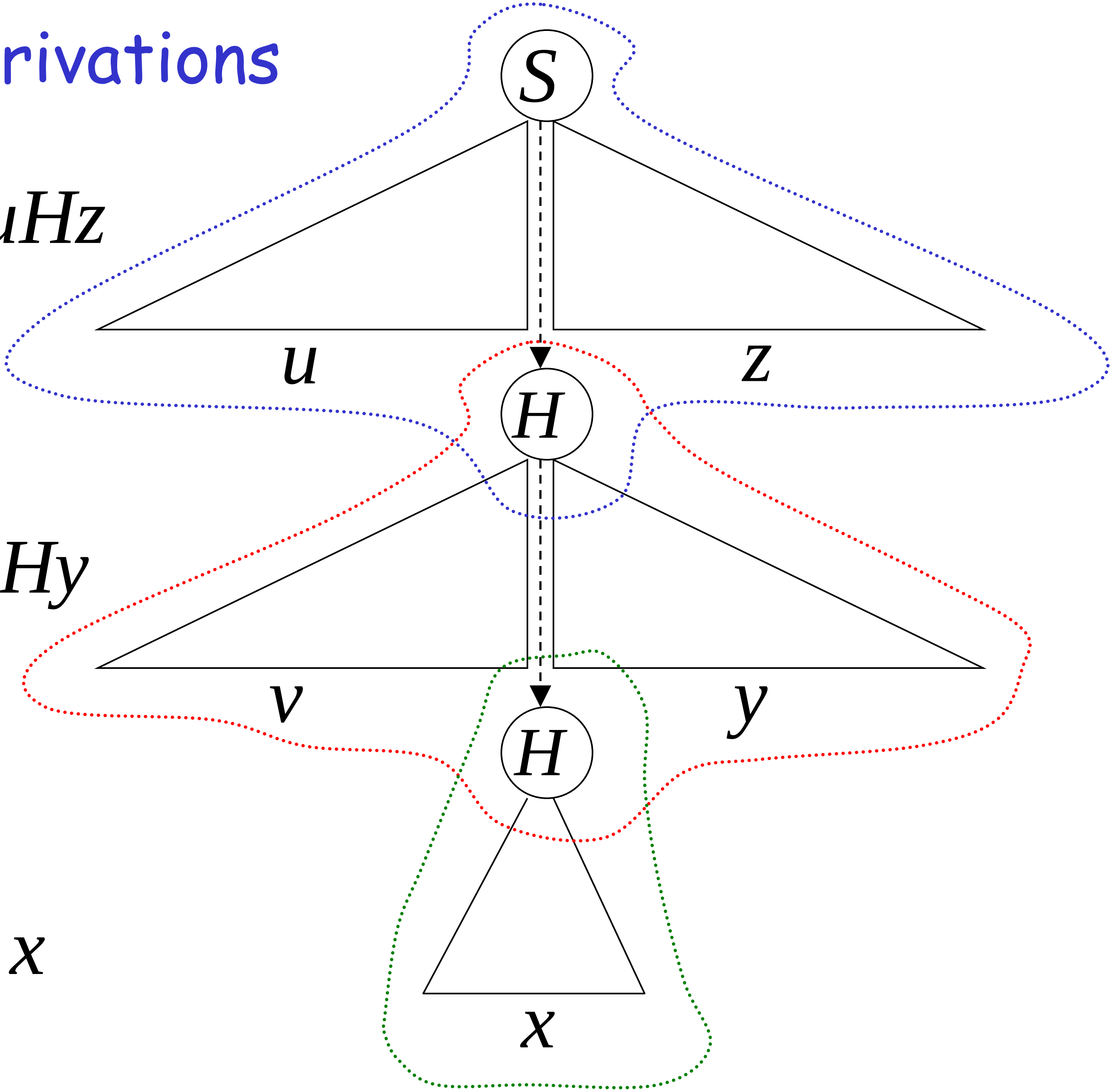
B corresponds to H

Possible derivations

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

$$H \overset{*}{\Rightarrow} vHy$$

$$H \overset{*}{\Rightarrow} x$$



Example:

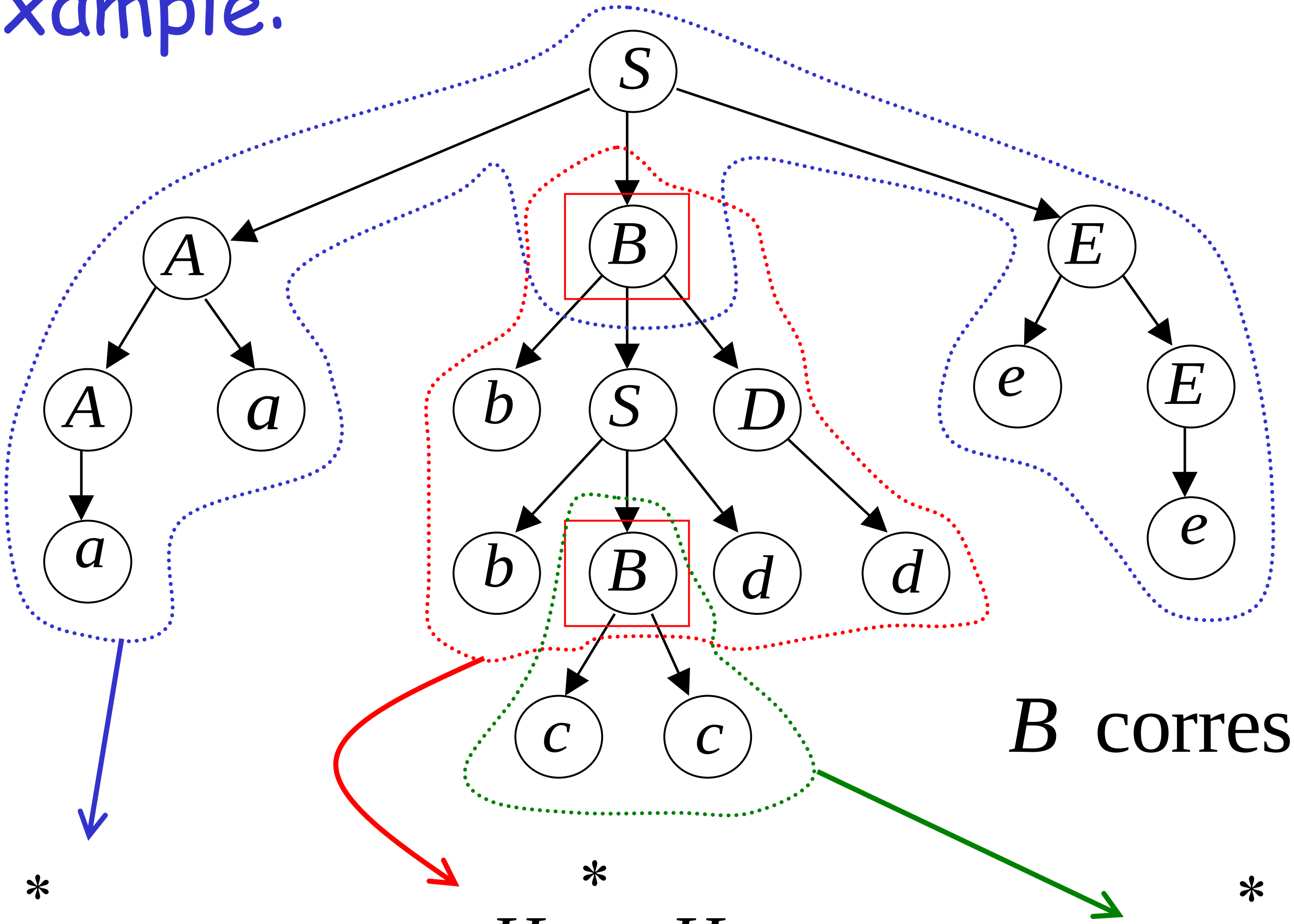
$u = aa$

$v = bb$

$x = cc$

$y = dd$

$z = ee$



B corresponds to H

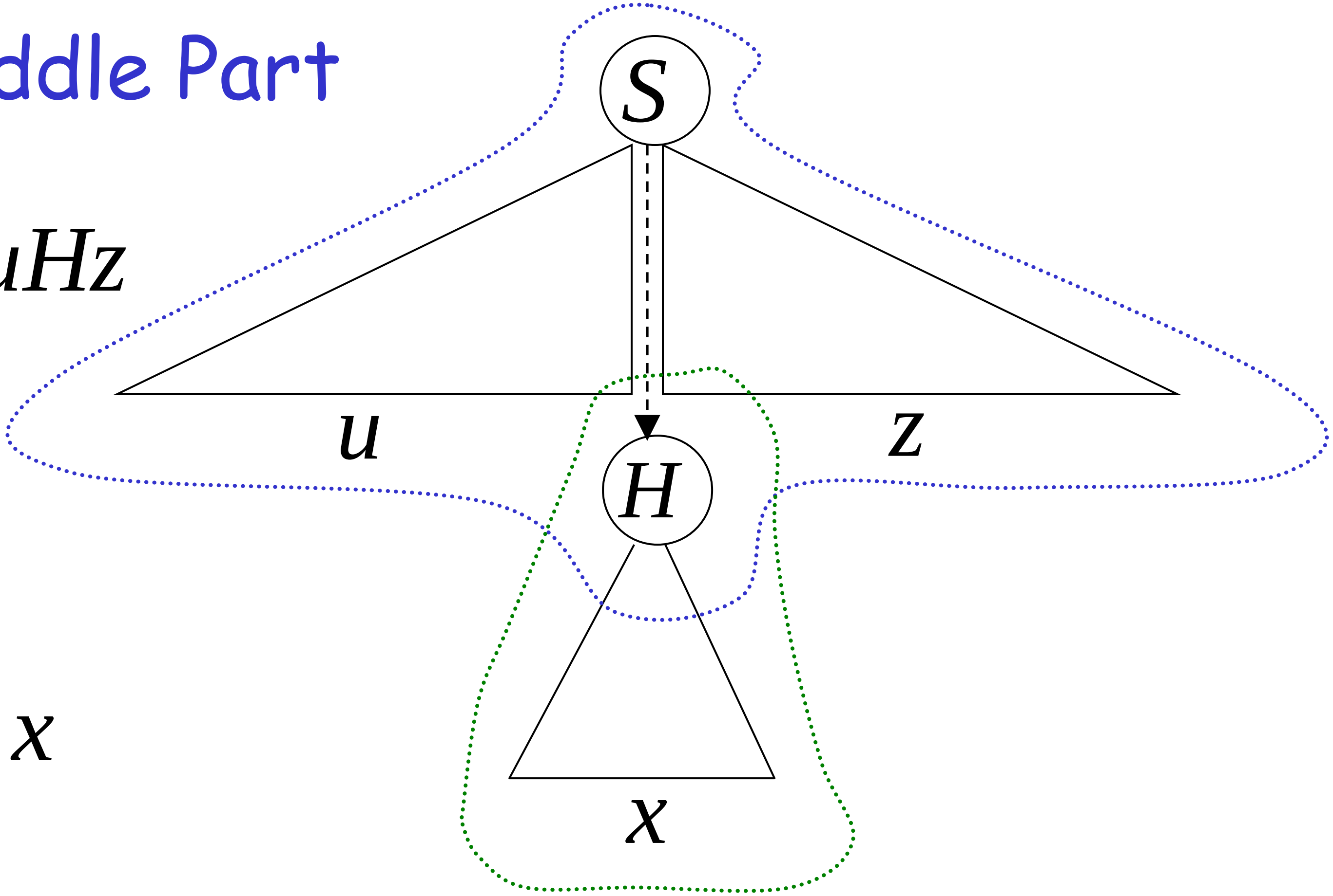
$S \overset{*}{\Rightarrow} uHz$ $\quad$ $H \overset{*}{\Rightarrow} vHy$ $\quad$ $H \overset{*}{\Rightarrow} x$

$S \overset{*}{\Rightarrow} aaBee$ $\quad$ $B \overset{*}{\Rightarrow} bbBdd$ $\quad$ $B \Rightarrow cc$

Remove Middle Part

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

$$H \overset{*}{\Rightarrow} x$$



Yield: $uxz = uv^0xy^0z$

$$S \overset{*}{\Rightarrow} uHz \overset{*}{\Rightarrow} uxz \quad = uv^0xy^0z \quad \in L(G)$$

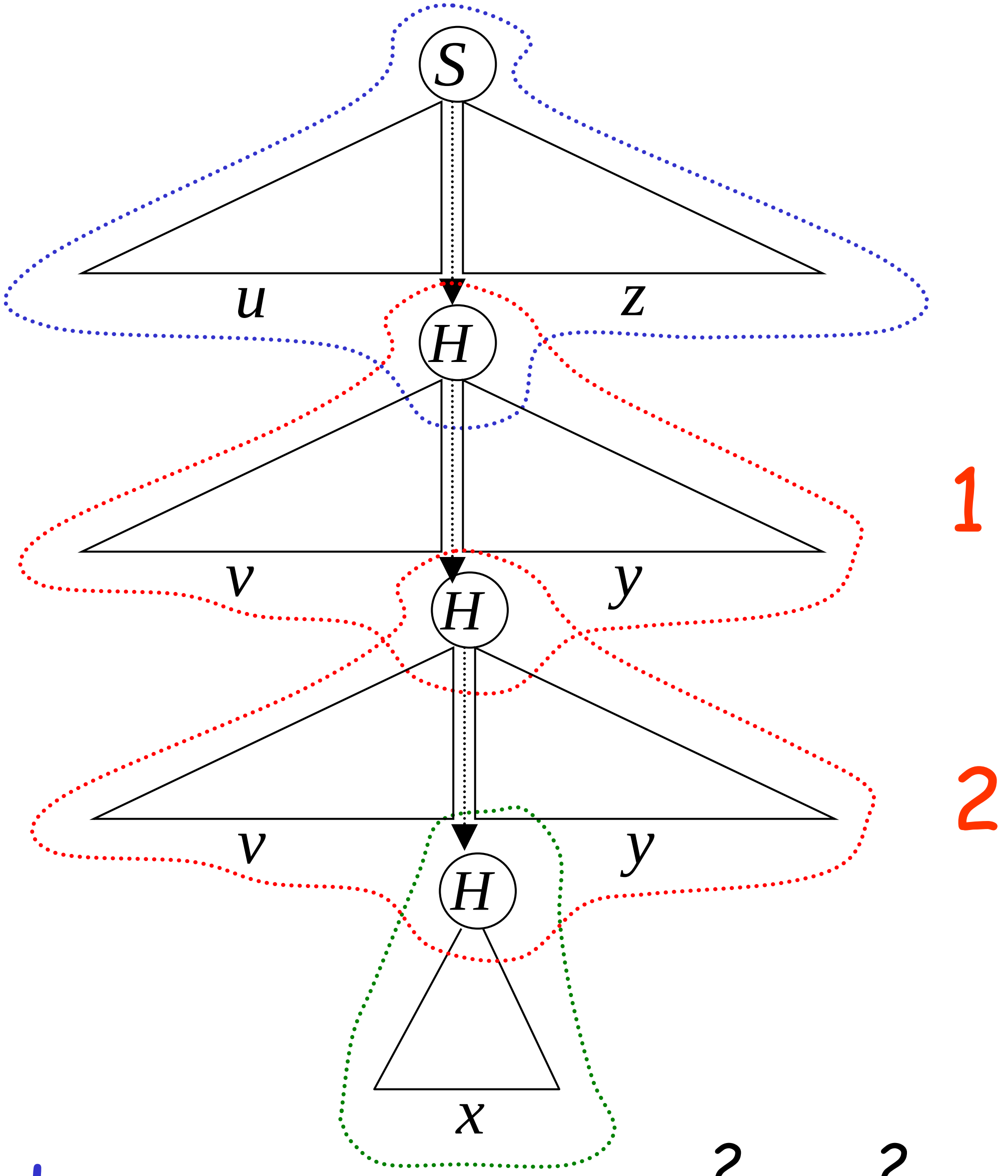
Repeat Middle part two times

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

$$H \overset{*}{\Rightarrow} vHy$$

$$H \overset{*}{\Rightarrow} vHy$$

$$H \overset{*}{\Rightarrow} x$$



Yield: $uvvxyyz = uv^2xy^2z$

$$S \overset{*}{\Rightarrow} uHz \qquad H \overset{*}{\Rightarrow} vHy \qquad H \overset{*}{\Rightarrow} x$$

$$\Big\Downarrow$$

$$S \overset{*}{\Rightarrow} uHz \overset{*}{\Rightarrow} uvHyz \overset{*}{\Rightarrow} uvvHyyz$$

$$\overset{*}{\Rightarrow} uvvxyyz \quad = uv^2xy^2z \quad \in L(G)$$

Repeat Middle part i times

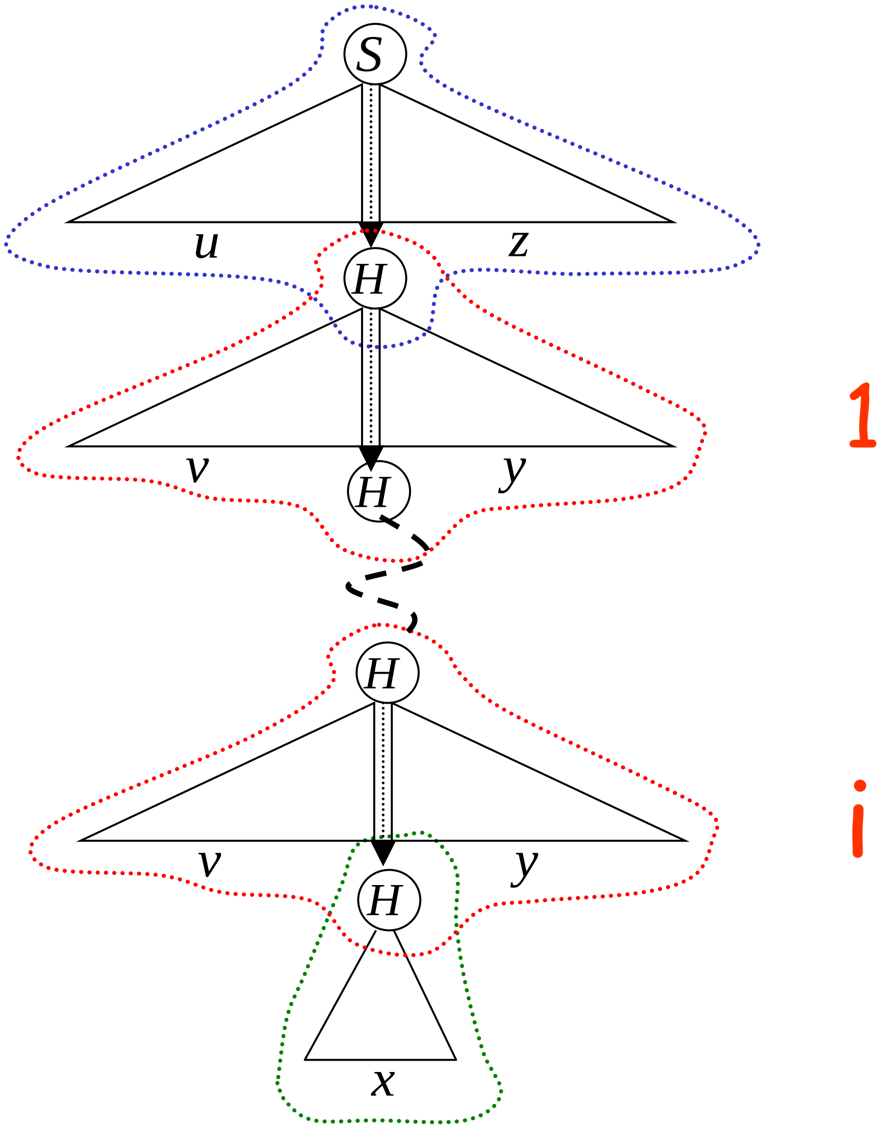
$$S \overset{*}{\Rightarrow} uHz$$

$$H \overset{*}{\Rightarrow} vHy$$

⋮

$$H \overset{*}{\Rightarrow} vHy$$

$$H \overset{*}{\Rightarrow} x$$



Yield: $uv^i xy^i z$

$$S \overset{*}{\Rightarrow} uHz \qquad H \overset{*}{\Rightarrow} vHy \qquad H \overset{*}{\Rightarrow} x$$

$$S \overset{*}{\Rightarrow} uHz \overset{*}{\Rightarrow} uvHyz \overset{*}{\Rightarrow} uvvHyyz \overset{*}{\Rightarrow}$$

$$\overset{*}{\Rightarrow} \ldots$$

$$\overset{*}{\Rightarrow} uv^{i}Hy^{i}z \overset{*}{\Rightarrow} uv^{i}xy^{i}z \in L(G)$$

Therefore,

$$|w| > t^r$$

If we know that: $w = uvxyz \quad \in L(G)$

then we also know: $uv^i xy^i z \quad \in L(G)$

For all $\boldsymbol{i} \geq \mathbf{0}$

since

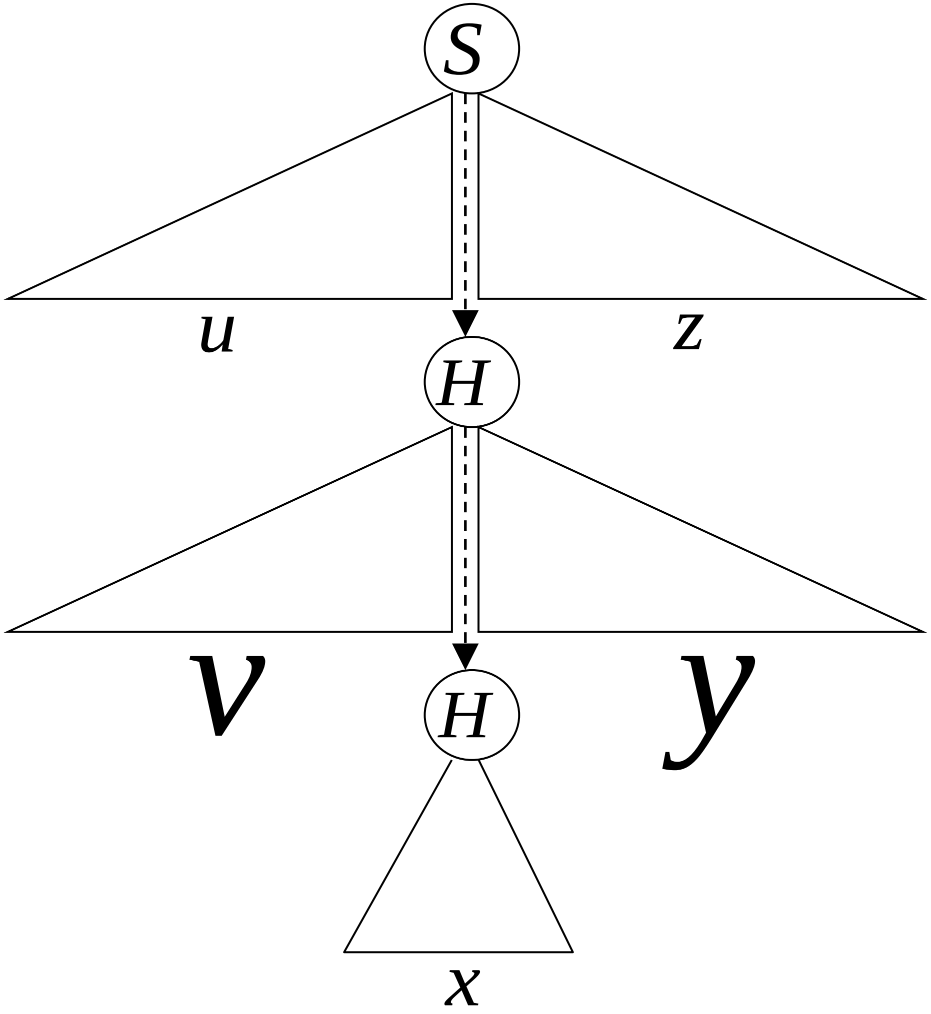
$L(G) = L - \{\lambda\}$

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

Observation 1:

$$|vy| \geq 1$$

Since G has no unit and ε-productions

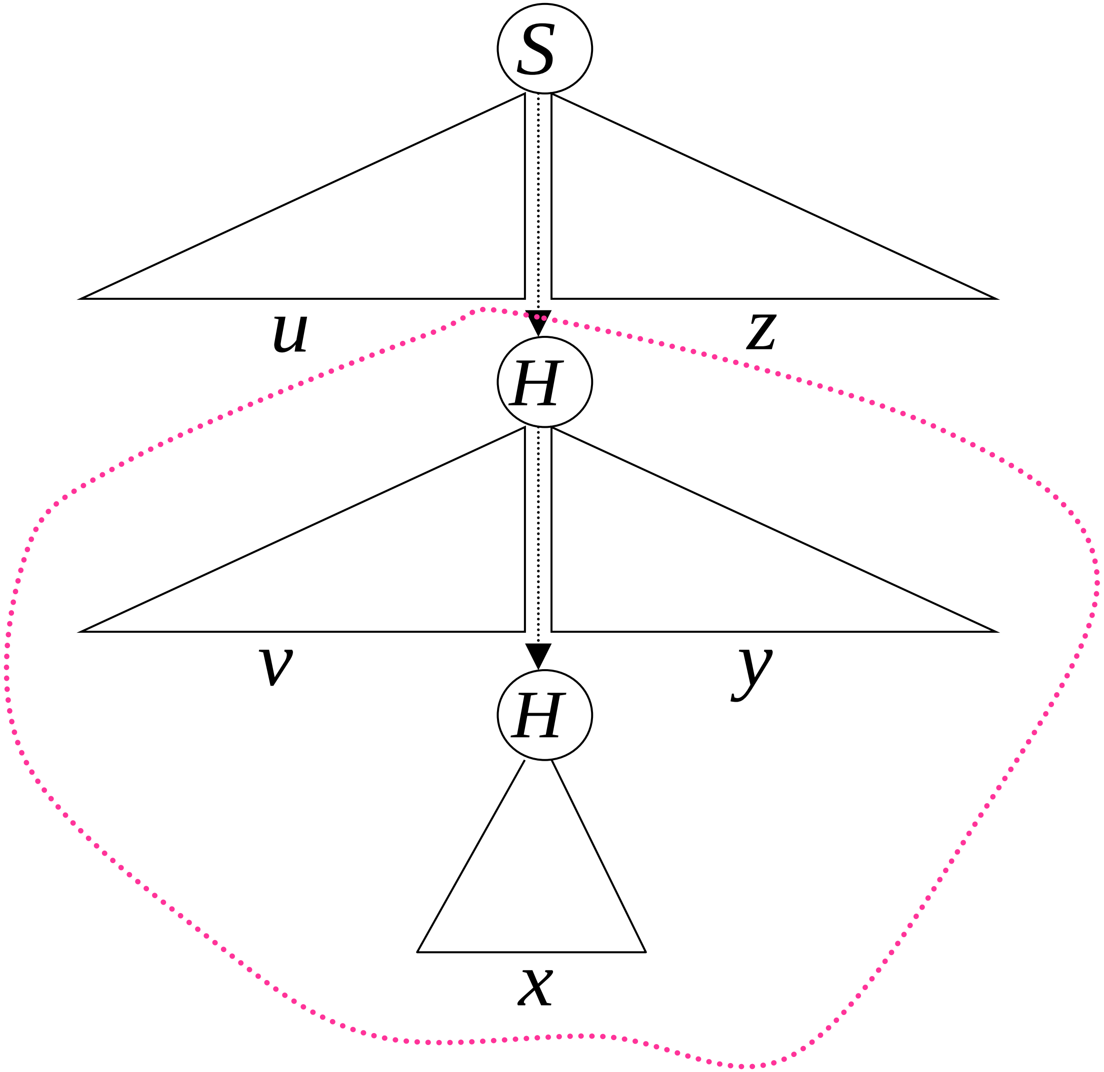


At least one of v or y is not ε

Observation 2:

$$|vxy| \leq t^{r+1}$$

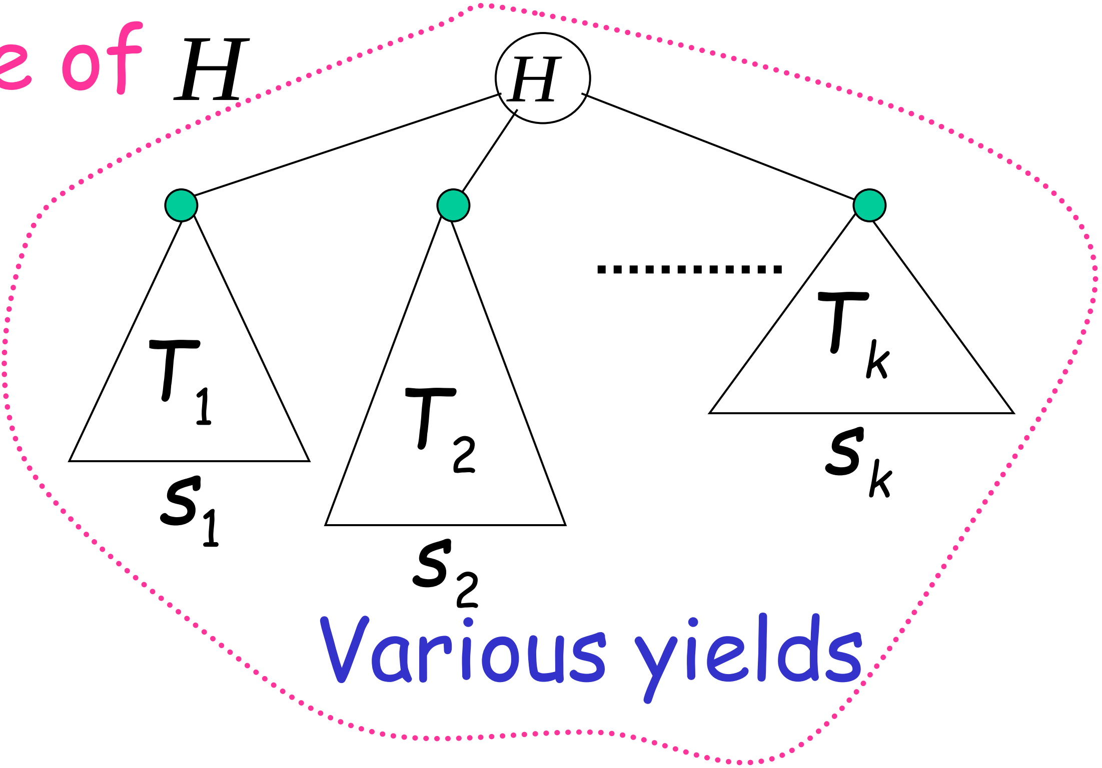
since in subtree only variable H is repeated



subtree of H

Explanation follows....

subtree of H



Various yields

$$vxy = s_1 s_2 \cdots s_k$$

$$|s_j| \le t^r$$ since no variable is repeated in T_j

$$|vxy| = \sum_{j=1}^{k} |s_j| \le k \cdot t^r \le t \cdot t^r = t^{r+1}$$

Maximum right-hand side of any production

Thus, if we choose critical length

$$p = t^{r+1} > t^{r}$$

then, we obtain the pumping lemma for context-free languages

The Pumping Lemma:

For any infinite context-free language L

there exists an integer p such that

for any string $w \in L, \quad |w| \geq p$

we can write $w = uvxyz$

with lengths $|vxy| \leq p$ and $|vy| \geq 1$

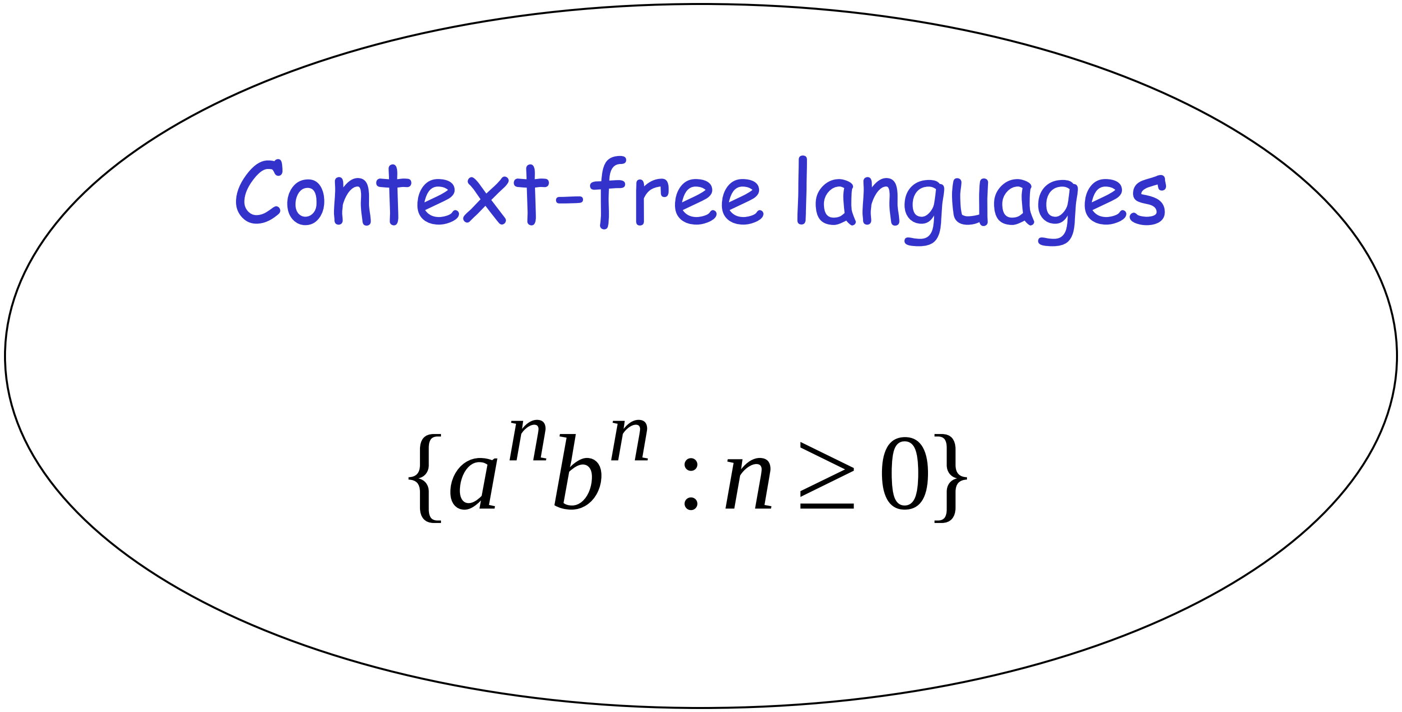
and it must be that:

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

Applications
of
The Pumping Lemma

Non-context free languages

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



Context-free languages

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

Theorem: The language

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

is **not** context free

Proof: Use the Pumping Lemma for context-free languages

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

Assume for contradiction that L is context-free

Since L is context-free and infinite we can apply the pumping lemma

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

Let p be the critical length of the pumping lemma

Pick any string $w \in L$ with length $|w| \geq p$

We pick: $w = a^p b^p c^p$

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

$$w = a^p b^p c^p$$

From pumping lemma:

we can write: $w = uvxyz$

with lengths $|vxy| \leq p$ and $|vy| \geq 1$

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

$$w = a^p b^p c^p$$

$w = uvxyz$ $\qquad |vxy| \leq p$ $\qquad |vy| \geq 1$

Pumping Lemma says:

$uv^i xy^i z \in L$ for all $i \geq 0$

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

$$w = a^p b^p c^p$$

$$w = uvxyz \qquad |vxy| \leq p \qquad |vy| \geq 1$$

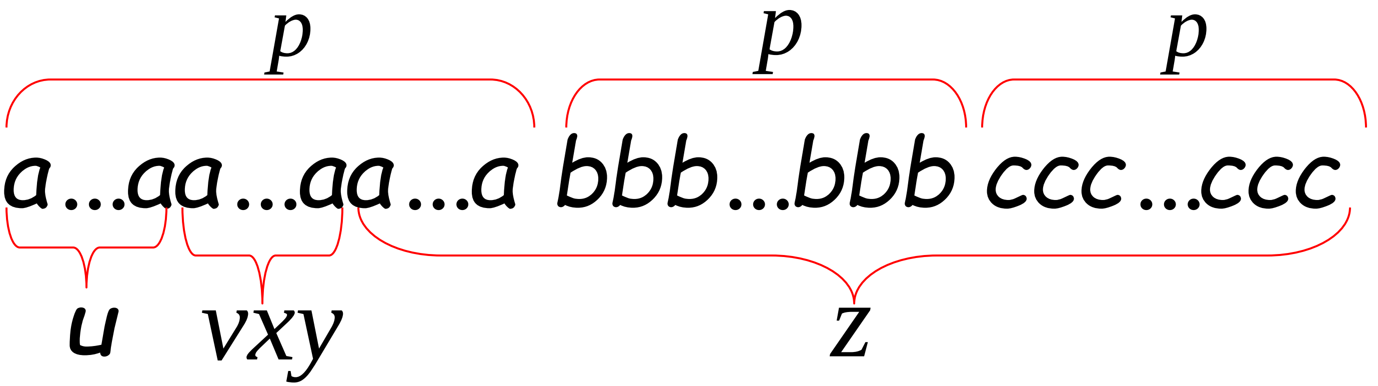
We examine **all** the possible locations of string vxy in w

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

$$w = a^p b^p c^p$$

$$w = uvxyz \qquad |vxy| \leq p \qquad |vy| \geq 1$$

Case 1: vxy is in a^p



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

$$w = a^p b^p c^p$$

$$w = uvxyz \qquad |vxy| \le p \qquad \boxed{|vy| \ge 1}$$

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

$$\overbrace{\underbrace{a\ldots a}_{u}\underbrace{a\ldots a}_{v}\underbrace{a\ldots a}_{x}\underbrace{a\ldots a}_{y}\mathrlap{\underbrace{}_{z}}a\ldots a}^{p}\;\overbrace{bbb\ldots bbb}^{p}\;\overbrace{ccc\ldots ccc}^{p}$$

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

$$w = a^p b^p c^p$$

$$w = uvxyz \qquad |vxy| \leq p \qquad |vy| \geq 1$$

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

$$\mathrlap{\overbrace{}^{p+k_1+k_2}}\underbrace{a...a}_{u}\underbrace{a...a}_{v^2}\underbrace{a...a}_{x}\underbrace{a...a}_{y^2}\underbrace{a...a\ \overbrace{bbb...bbb}^{p}\ \overbrace{ccc...ccc}^{p}}_{z}$$

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

$$w = a^p b^p c^p$$

$$w = uvxyz \qquad |vxy| \leq p \qquad |vy| \geq 1$$

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

$$k_1 + k_2 \geq 1$$

However: $uv^2xy^2z = a^{p+k_1+k_2} b^p c^p \notin L$

Contradiction!!!

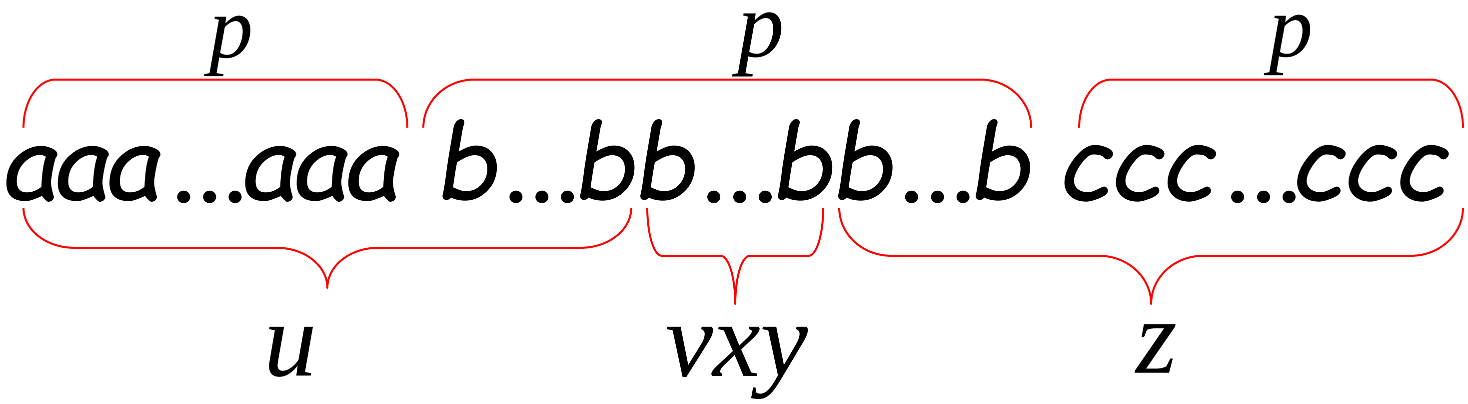
$$L = \{a^n b^n c^n : n \geq 0\}$$

$$w = a^p b^p c^p$$

$$w = uvxyz \qquad |vxy| \leq p \qquad |vy| \geq 1$$

Case 2: vxy is in b^p

Similar to case 1



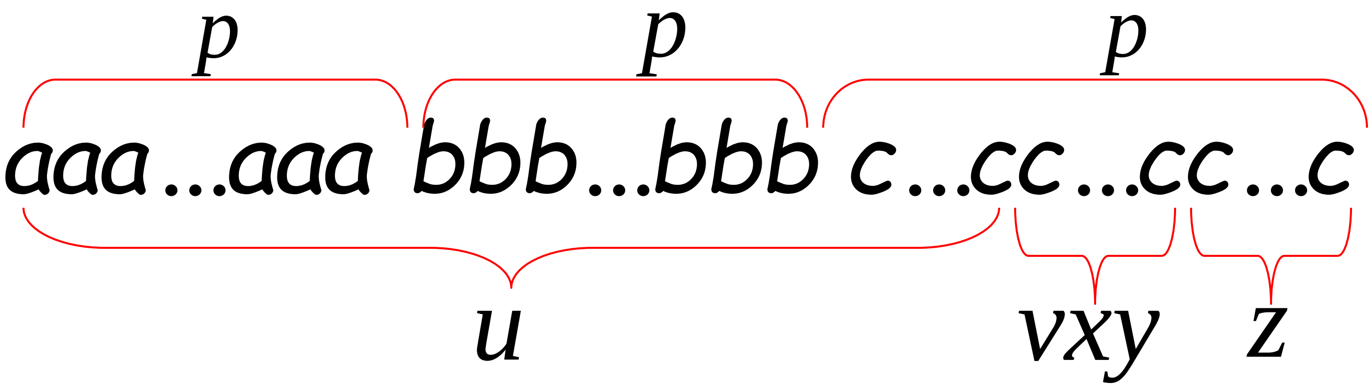
$$L=\{a^n b^n c^n : n \geq 0\}$$

$$w = a^p b^p c^p$$

$w = uvxyz$ $\qquad |vxy| \leq p$ $\qquad |vy| \geq 1$

Case 3: vxy is in c^p

Similar to case 1

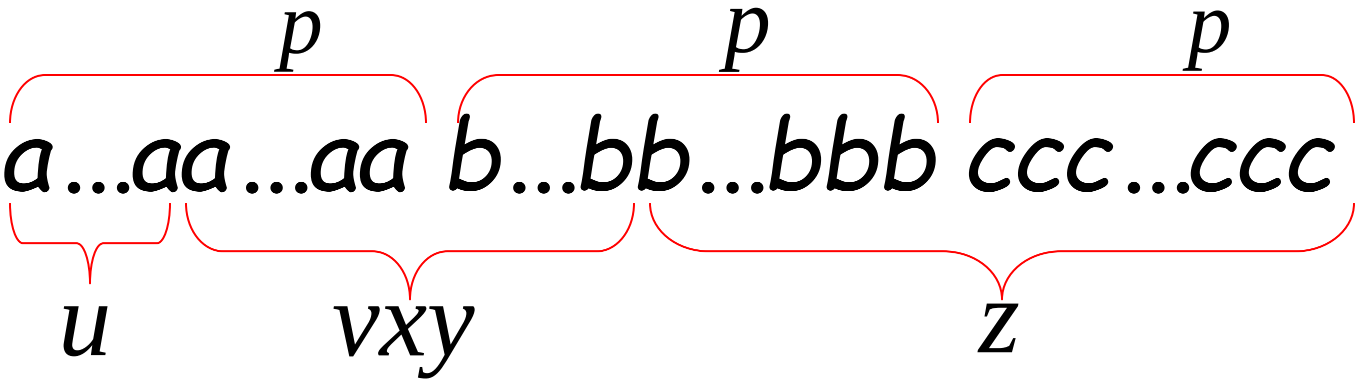


$$L = \{a^n b^n c^n : n \geq 0\}$$

$$w = a^p b^p c^p$$

$w = uvxyz$ $\quad$ $|vxy| \leq p$ $\quad$ $|vy| \geq 1$

Case 4: vxy overlaps a^p and b^p



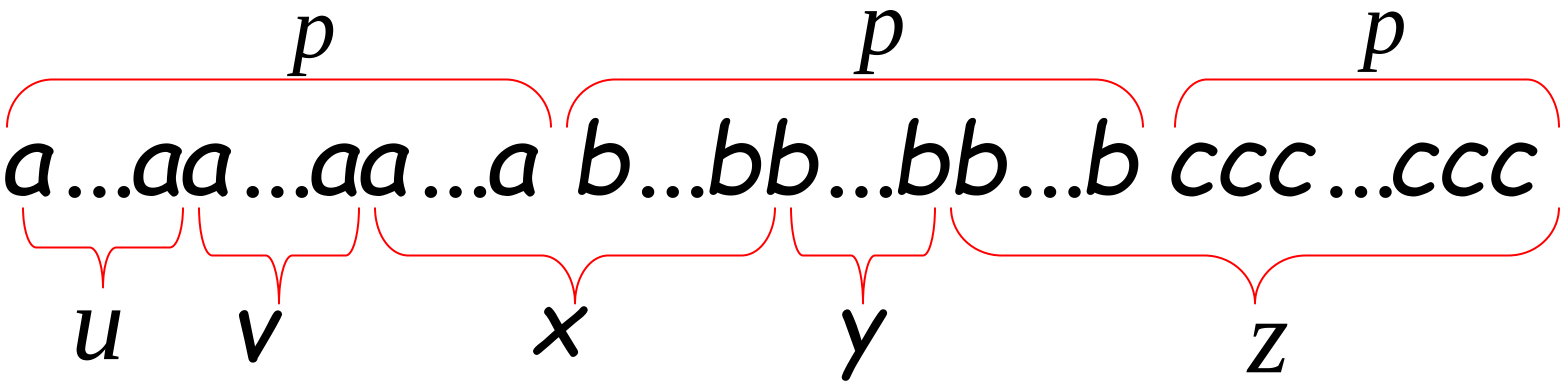
$$L=\{a^nb^nc^n : n \geq 0\}$$

$$w=a^pb^pc^p$$

$w=uvxyz$ $|vxy| \leq p$ $|vy| \geq 1$

Sub-case 1: v contains only a

y contains only b



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

$$w = a^p b^p c^p$$

$$w = uvxyz \qquad |vxy| \leq p \qquad |vy| \geq 1$$

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

$$\overbrace{a\ldots a\,a\ldots a\,a\ldots a}^{p}\ \overbrace{b\ldots b\,b\ldots b\,b\ldots b}^{p}\ \overbrace{ccc\ldots ccc}^{p}$$

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

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

$$w=a^p b^p c^p$$

$$w=uvxyz \qquad |vxy| \leq p \qquad |vy| \geq 1$$

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

$$\overbrace{a\ldots aa\ldots aa\ldots a}^{p+k_1}\;\overbrace{b\ldots bb\ldots bb\ldots b}^{p+k_2}\;\overbrace{ccc\ldots ccc}^{p}$$

$$\underbrace{a\ldots a}_{u}\underbrace{a\ldots a}_{v^2}\underbrace{a\ldots a\;b\ldots b}_{x}\underbrace{b\ldots b}_{y^2}\underbrace{b\ldots b\;ccc\ldots ccc}_{z}$$

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

$$w = a^p b^p c^p$$

$w = uvxyz$ $\quad |vxy| \leq p$ $\quad |vy| \geq 1$

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

$$k_1 + k_2 \geq 1$$

However: $uv^2xy^2z = a^{p+k_1} b^{p+k_2} c^p \notin L$

Contradiction!!!

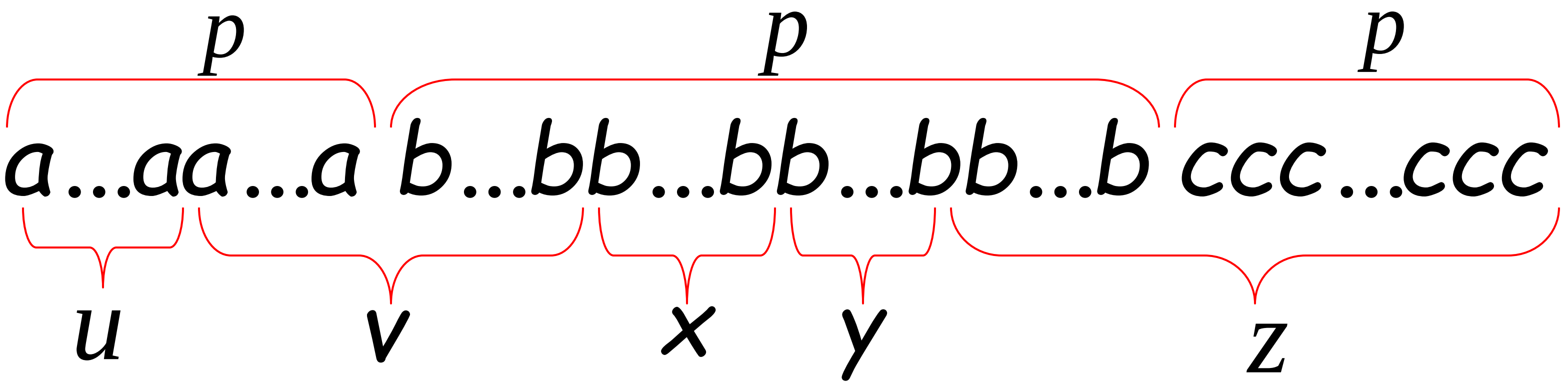
$$L = \{a^n b^n c^n : n \geq 0\}$$

$$w = a^p b^p c^p$$

$w = uvxyz$ $\qquad |vxy| \leq p$ $\qquad |vy| \geq 1$

Sub-case 2: v contains a and b

y contains only b



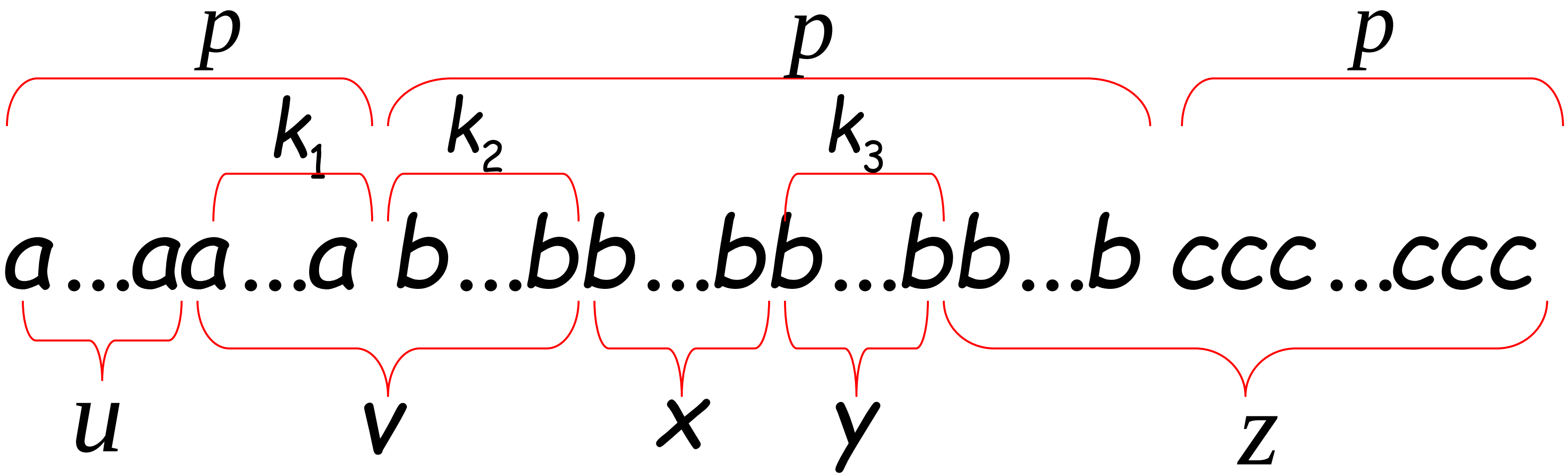
$$L = \{a^n b^n c^n : n \geq 0\}$$

$$w = a^p b^p c^p$$

$$w = uvxyz \qquad |vxy| \leq p \qquad |vy| \geq 1$$

By assumption

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



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

$$w = a^p b^p c^p$$

$$w = uvxyz \qquad |vxy| \leq p \qquad |vy| \geq 1$$

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

$$\overbrace{a\ldots a\overbrace{a\ldots a}^{k_1}}^{p}\overbrace{b\ldots b}^{k_2}\overbrace{a\ldots a}^{k_1}\overbrace{\overbrace{b\ldots b}^{k_2}b\ldots b\overbrace{b\ldots b}^{2k_3}b\ldots b}^{p+k_3}\ \overbrace{ccc\ldots ccc}^{p}$$

$$\underbrace{a\ldots a}_{u}\underbrace{a\ldots ab\ldots ba\ldots ab\ldots b}_{v^2}\underbrace{b\ldots b}_{x}\underbrace{b\ldots b}_{y^2}\underbrace{b\ldots b\ ccc\ldots ccc}_{z}$$

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

$$w = a^p b^p c^p$$

$w = uvxyz$ $|vxy| \leq p$ $|vy| \geq 1$

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

$k_1, k_2 \geq 1$

However: $uv^2xy^2z = a^p b^{k_2} a^{k_1} b^{p+k_3} c^p \notin L$

Contradiction!!!

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

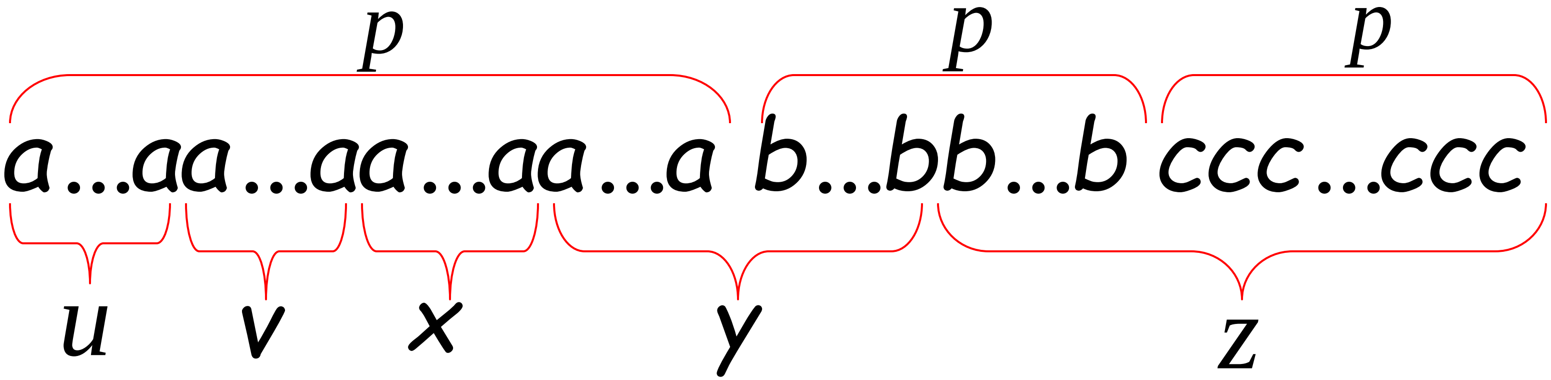
$$w = a^p b^p c^p$$

$$w = uvxyz \qquad |vxy| \leq p \qquad |vy| \geq 1$$

Sub-case 3: v contains only a

y contains a and b

Similar to sub-case 2



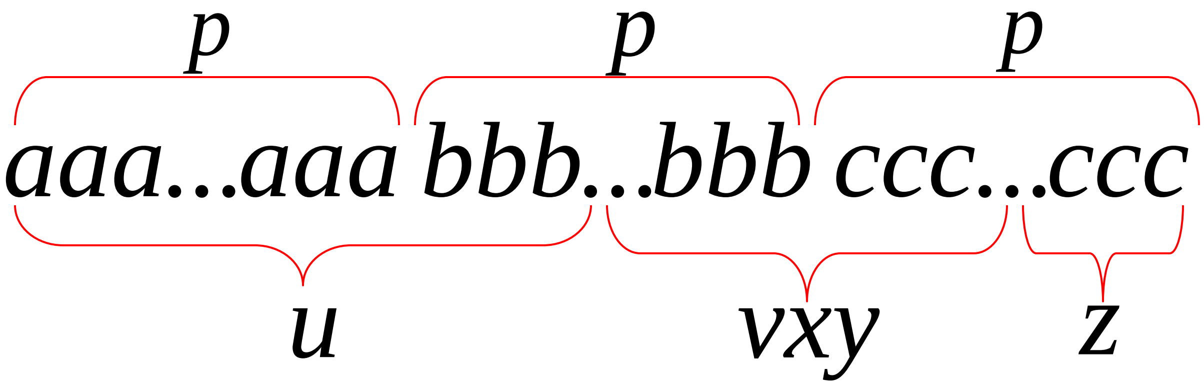
$$L = \{a^n b^n c^n : n \geq 0\}$$

$$w = a^p b^p c^p$$

$$w = uvxyz \qquad |vxy| \leq p \qquad |vy| \geq 1$$

Case 5: vxy overlaps b^p and c^p

Similar to case 4



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

$$w = a^p b^p c^p$$

$$w = uvxyz \qquad |vxy| \leq p \qquad |vy| \geq 1$$

Case 6: vxy overlaps a^p, b^p and c^p

Impossible!

$$\underbrace{aaa\ldots}_{u}\underbrace{aaa\;\overbrace{bbb\ldots bbb}^{p}\;ccc\ldots}_{vxy}\underbrace{ccc}_{z}$$

In all cases we obtained a contradiction

Therefore: the original assumption that

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

is context-free must be wrong

Conclusion: L is not context-free

More Applications of The Pumping Lemma

The Pumping Lemma:

For infinite context-free language L

there exists an integer m such that

for any string $w \in L, \quad |w| \geq m$

we can write $w = uvxyz$

with lengths $|vxy| \leq m$ and $|vy| \geq 1$

and it must be:

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

Non-context free languages

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

Context-free languages

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

Theorem: The language

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

is **not** context free

Proof: Use the Pumping Lemma for context-free languages

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

Assume for contradiction that L is context-free

Since L is context-free and infinite we can apply the pumping lemma

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

Pumping Lemma gives a magic number m such that:

Pick any string of L with length at least m

we pick: $a^m b^m a^m b^m \in L$

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

We can write: $a^m b^m a^m b^m = uvxyz$

with lengths $|vxy| \leq m$ and $|vy| \geq 1$

Pumping Lemma says:

$uv^i xy^i z \in L$ for all $i \geq 0$

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

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

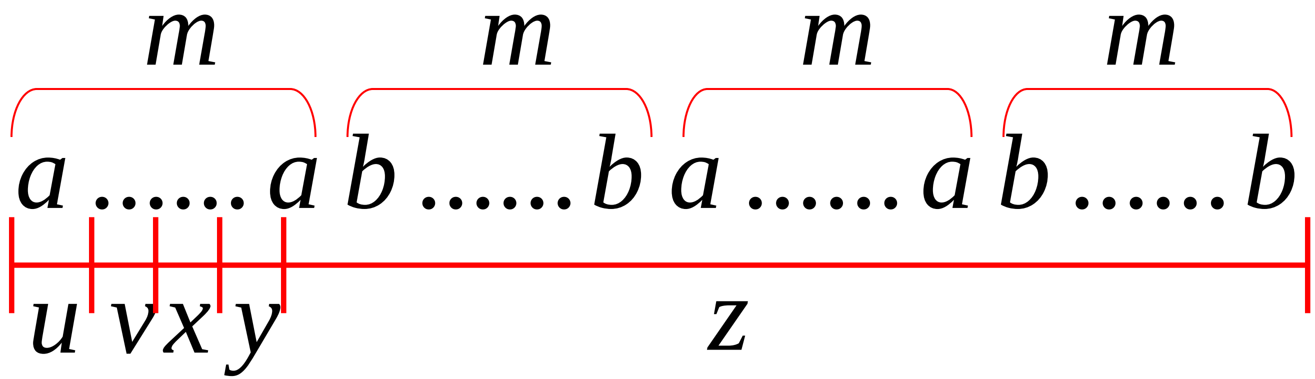
We examine **all** the possible locations of string vxy in $a^m b^m a^m b^m$

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

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

Case 1: vxy is within the first a^m

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

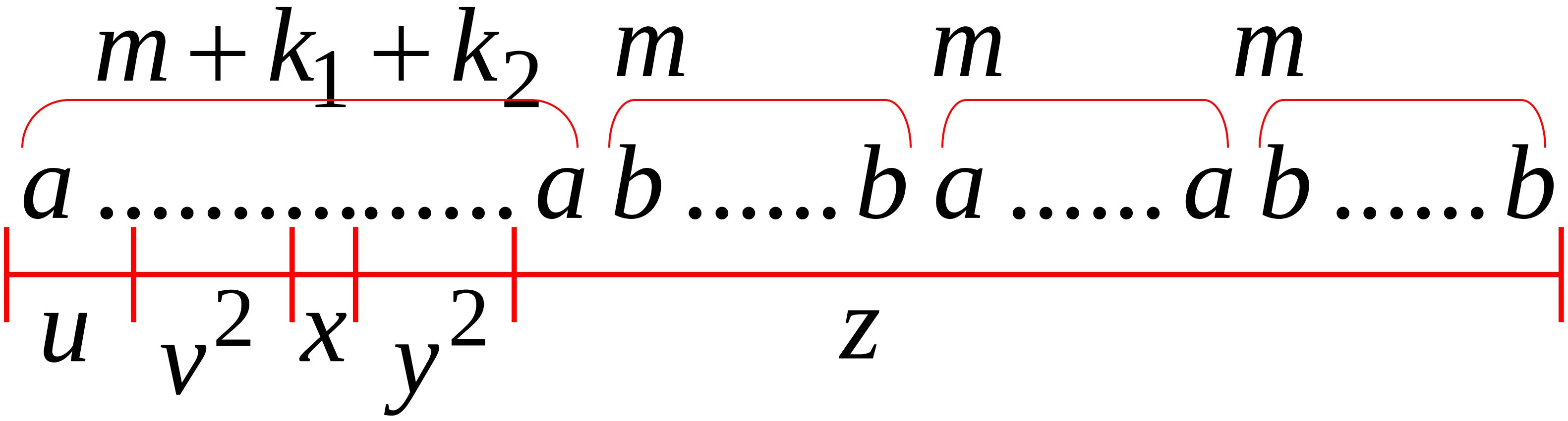


$$L = \{vv : v \in \{a,b\}^*\}$$

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

Case 1: vxy is within the first a^m

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



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

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

Case 1: vxy is within the first a^m

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

$$k_1 + k_2 \geq 1$$

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

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

Case 1: vxy is within the first a^m

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

However, from Pumping Lemma: $uv^2xy^2z \in L$

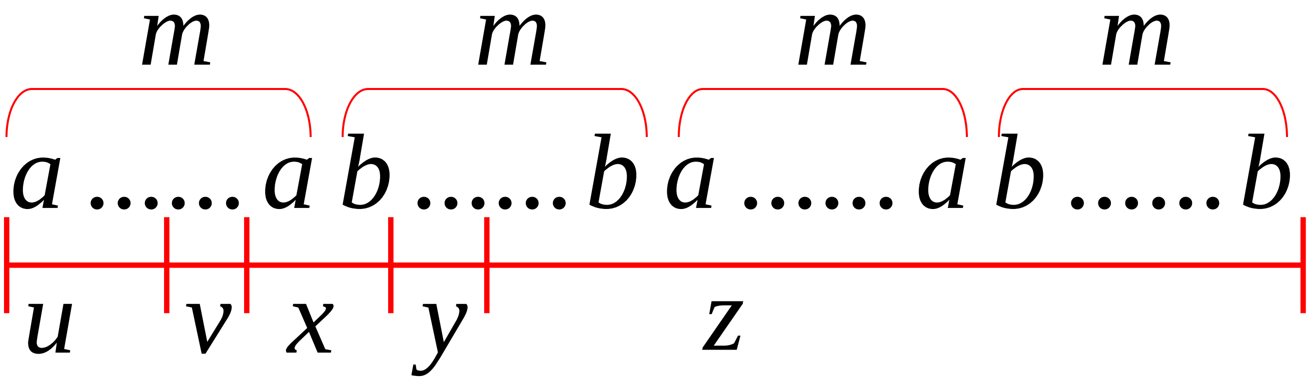
Contradiction!!!

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

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

Case 2: v is in the first a^m
y is in the first b^m

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



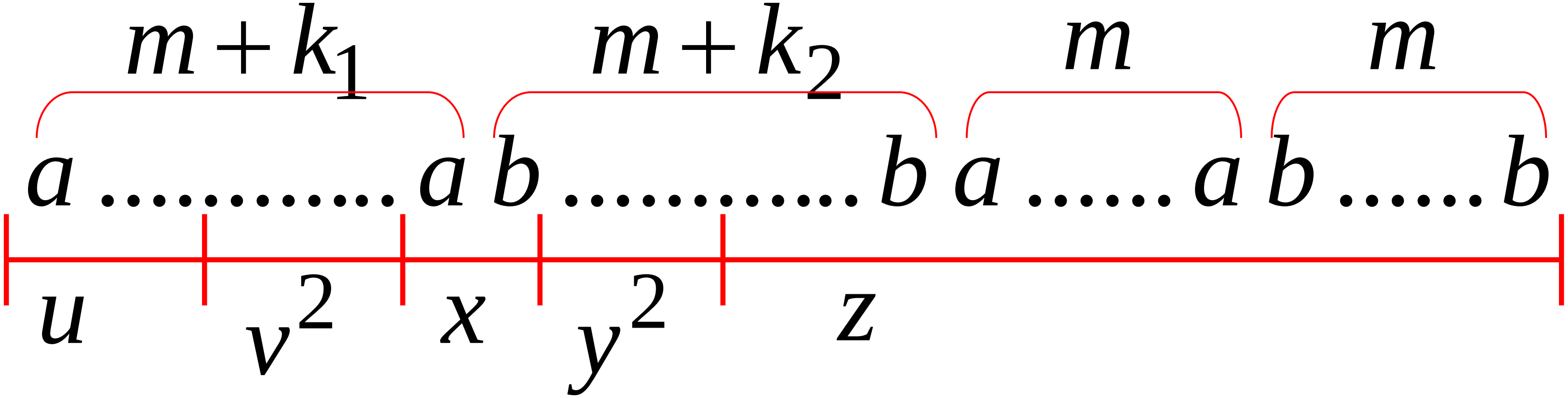
$$L = \{vv : v \in \{a,b\}^*\}$$

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

Case 2: v is in the first a^m

y is in the first b^m

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



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

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

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

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

Case 2: v is in the first a^m
y is in the first 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 is in the first a^m

y is in the first b^m

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

However, from Pumping Lemma: $uv^2xy^2z \in L$

Contradiction!!!

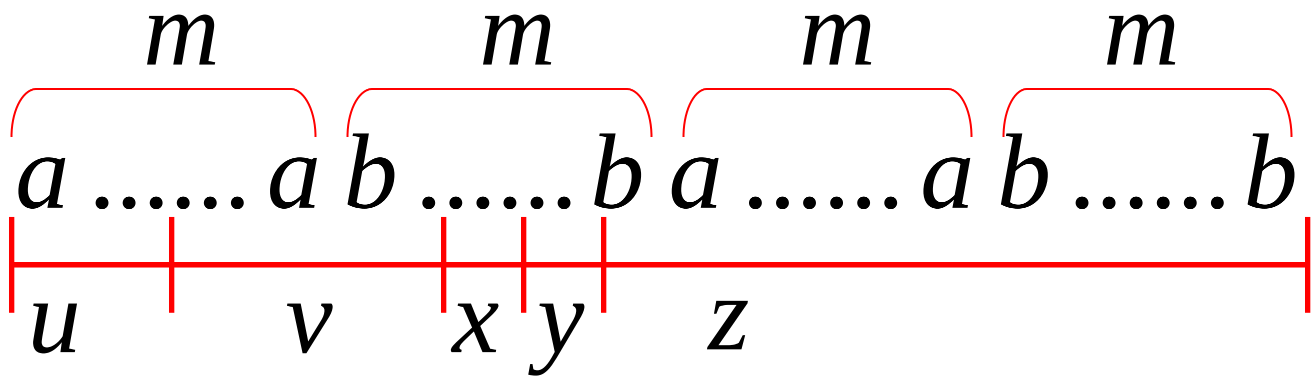
$$L = \{vv : v \in \{a,b\}^*\}$$

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

Case 3: v overlaps the first $a^m b^m$

y is in the first b^m

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



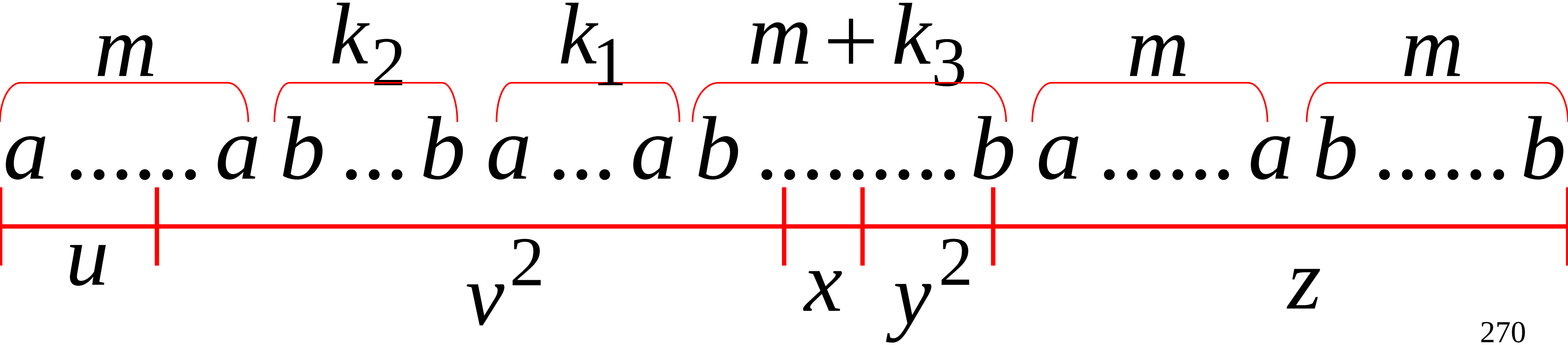
$$L = \{vv : v \in \{a,b\}^*\}$$

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

Case 3: v overlaps the first $a^m b^m$

y is in the first b^m

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



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

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

Case 3: v overlaps the first $a^m b^m$

y is in the first b^m

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

$$k_1, k_2 \geq 1$$

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

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

Case 3: v overlaps the first $a^m b^m$

y is in the first b^m

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

However, from Pumping Lemma: $uv^2xy^2z \in L$

Contradiction!!!

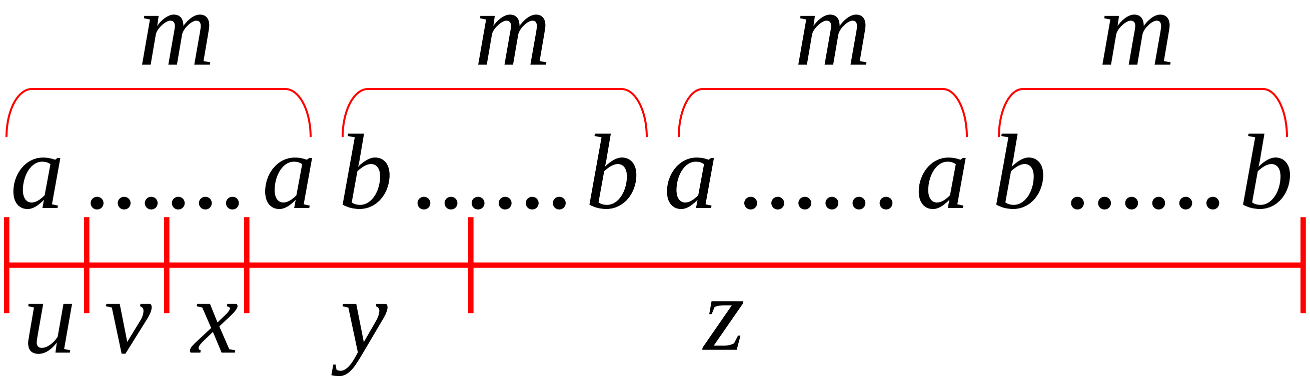
$$L = \{vv : v \in \{a,b\}^*\}$$

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

Case 4: v in the first a^m

y Overlaps the first $a^m b^m$

Analysis is similar to case 3



Other cases: vxy is within $a^m\boxed{b^m}a^mb^m$

or

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

or

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

Analysis is similar to case 1:

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

More cases:

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

or

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

Analysis is similar to cases 2,3,4:

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

There are no other cases to consider

Since $|vxy| \leq m$, it is impossible vxy to overlap:

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

nor

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

nor

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

In all cases we obtained a contradiction

Therefore: The original assumption that

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

is context-free must be wrong

Conclusion: L is not context-free

Non-context free languages

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

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

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

Context-free languages

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

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

Theorem: The language

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

is **not** context free

Proof: Use the Pumping Lemma for context-free languages

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

Assume for contradiction that L is context-free

Since L is context-free and infinite we can apply the pumping lemma

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

Pumping Lemma gives a magic number m such that:

Pick any string of L with length at least m

we pick: $a^{m!} \in L$

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

We can write: $a^{m!} = uvxyz$

with lengths $|vxy| \leq m$ and $|vy| \geq 1$

Pumping Lemma says:

$uv^i xy^i z \in L$ for all $i \geq 0$

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

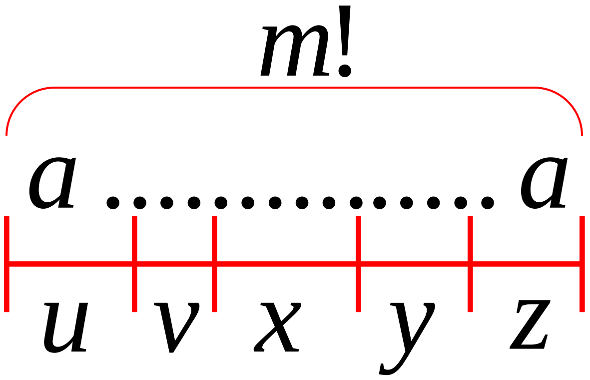
$$a^{m!} = uvxyz \qquad |vxy| \leq m \qquad |vy| \geq 1$$

We examine **all** the possible locations of string vxy in $a^{m!}$

There is only one case to consider

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

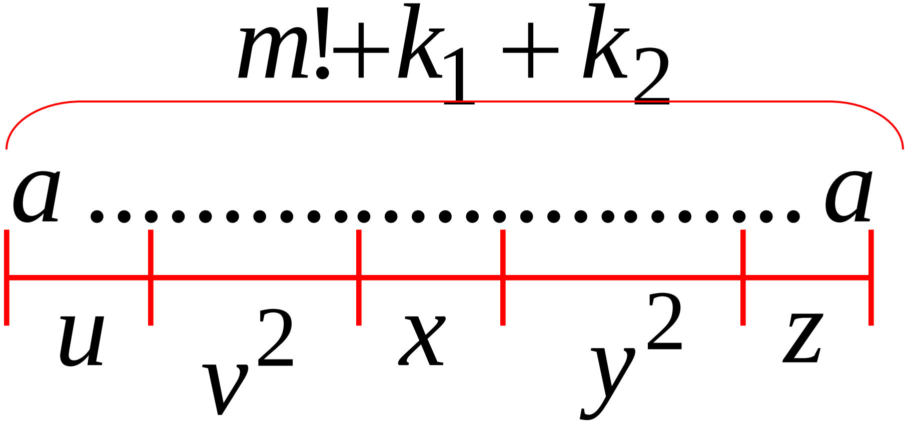
$$a^{m!} = uvxyz \qquad |vxy| \leq m \qquad |vy| \geq 1$$



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

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

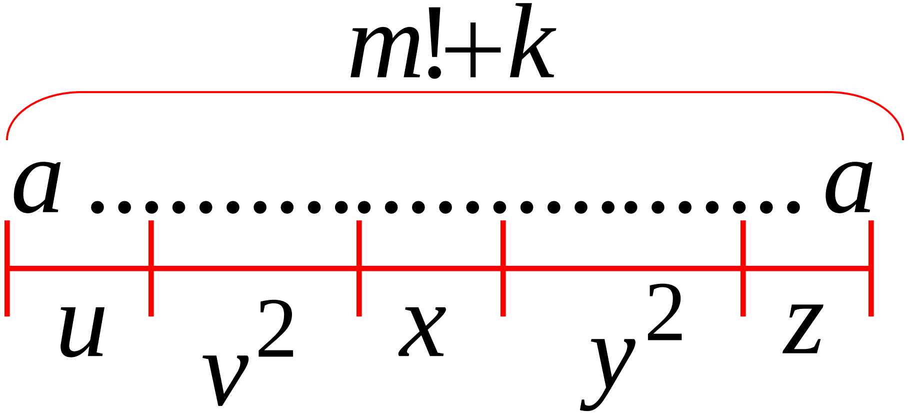
$$a^{m!} = uvxyz \qquad |vxy| \leq m \qquad |vy| \geq 1$$



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

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

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



$$\overbrace{a \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots a}^{m!+k}$$

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

$$k = k_1 + k_2$$

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

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

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

$$a^{m!+k} = uv^2xy^2z$$

$$1 \leq k \leq m$$

Since $1 \le k \le m$, for $m \ge 2$ we have:

$$
\begin{aligned}
m! + k &\le m! + m \\
&< m! + m!m \\
&= m!(1+m) \\
&= (m+1)!
\end{aligned}
$$

$$\Downarrow$$

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

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

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

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

$$\Downarrow$$

$$a^{m!+k} = uv^2xy^2z \quad \notin L$$

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

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

However, from Pumping Lemma: $uv^2xy^2z \in L$

$$a^{m!+k} = uv^2xy^2z \quad \notin L$$

Contradiction!!!

We obtained a contradiction

Therefore: The original assumption that

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

is context-free must be wrong

Conclusion: L is not context-free

Non-context free languages

$\{a^n b^n c^n : n \geq 0\}$ $\{ww : w \in \{a,b\}\}$

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

Context-free languages

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

Theorem: The language

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

is **not** context free

Proof: Use the Pumping Lemma for context-free languages

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

Assume for contradiction that L is context-free

Since L is context-free and infinite we can apply the pumping lemma

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

Pumping Lemma gives a magic number m such that:

Pick any string of L with length at least m

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

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

We can write: $a^{m^2} b^m = uvxyz$

with lengths $|vxy| \leq m$ and $|vy| \geq 1$

Pumping Lemma says:

$uv^i xy^i z \in L$ for all $i \geq 0$

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

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

We examine **all** the possible locations

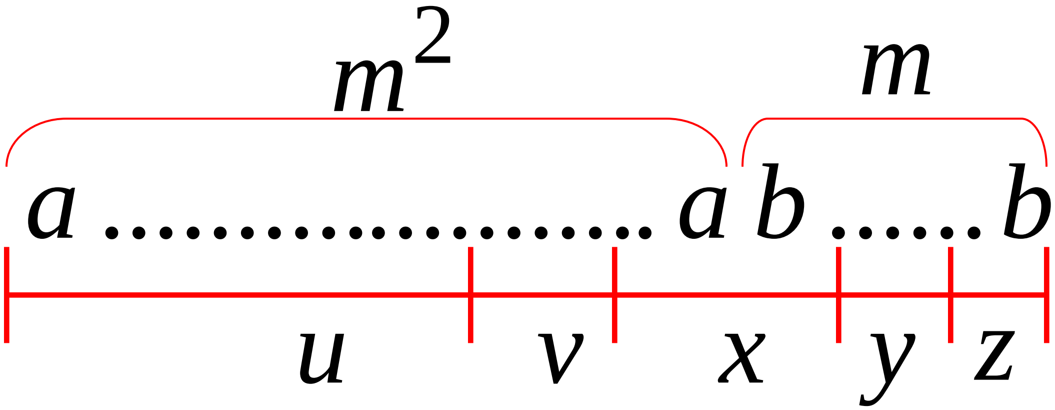
of string vxy in $a^{m^2}b^m$

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

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

Most complicated case: v is in a^m

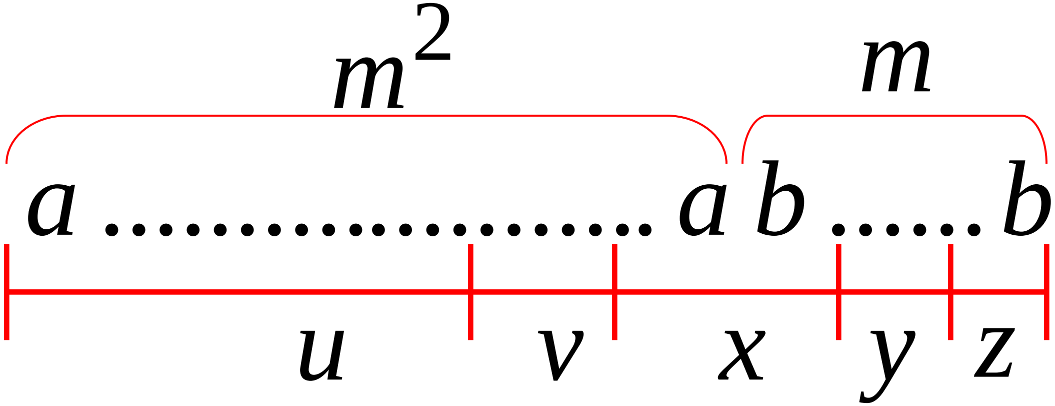
y is in b^m



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

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

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

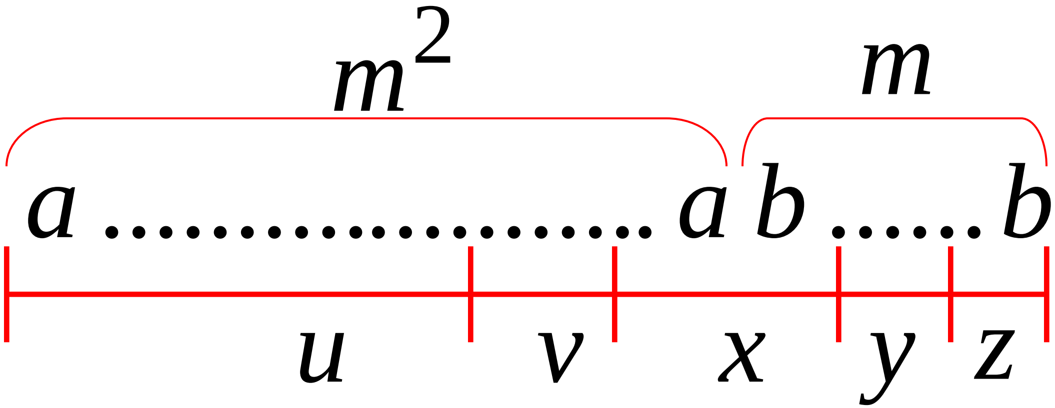


$$L = \{a^{n^2} b^n : n \geq 0\}$$

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

Most complicated sub-case: $k_1 \neq 0$ and $k_2 \neq 0$

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

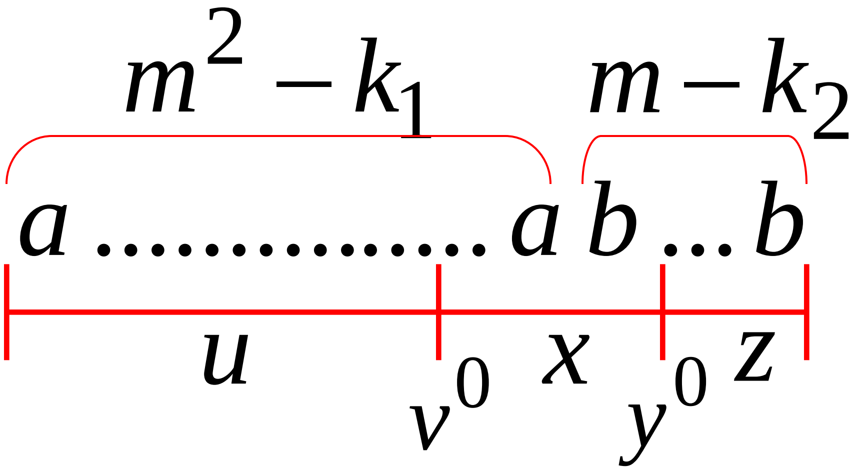


$$L = \{a^{n^2} b^n : n \geq 0\}$$

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

Most complicated sub-case: $k_1 \neq 0$ and $k_2 \neq 0$

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



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

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

Most complicated sub-case: $k_1 \neq 0$ and $k_2 \neq 0$

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

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

$k_1 \neq 0$ and $k_2 \neq 0$ $\quad\quad 1 \leq k_1 + k_2 \leq m$

$$\Downarrow$$

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

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

$$< m^2 - k_1$$

$$\Downarrow$$

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

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

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

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

$$\Downarrow$$

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

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

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

However, from Pumping Lemma: $uv^0xy^0z \in L$

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

Contradiction!!!

When we examine the rest of the cases we also obtain a contradiction

In all cases we obtained a contradiction

Therefore: The original assumption that

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

is context-free must be wrong

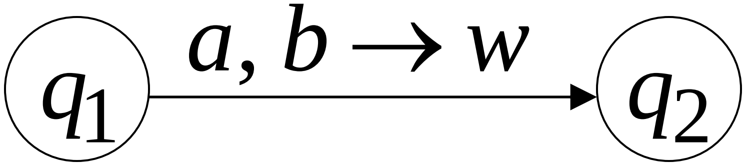
Conclusion: L is not context-free

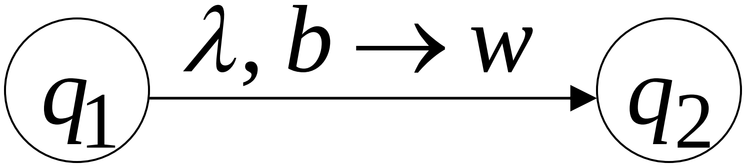
DPDA

Deterministic PDA

Deterministic PDA: DPDA

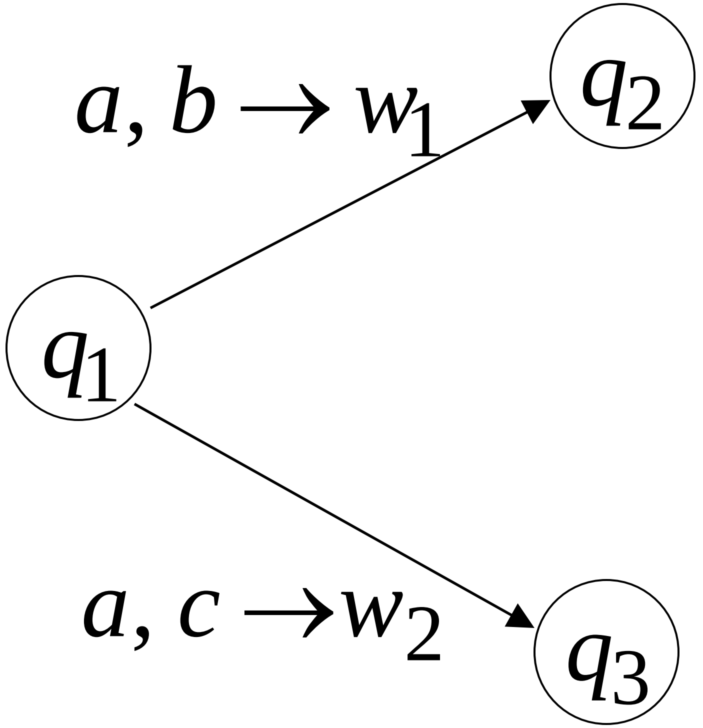
Allowed transitions:

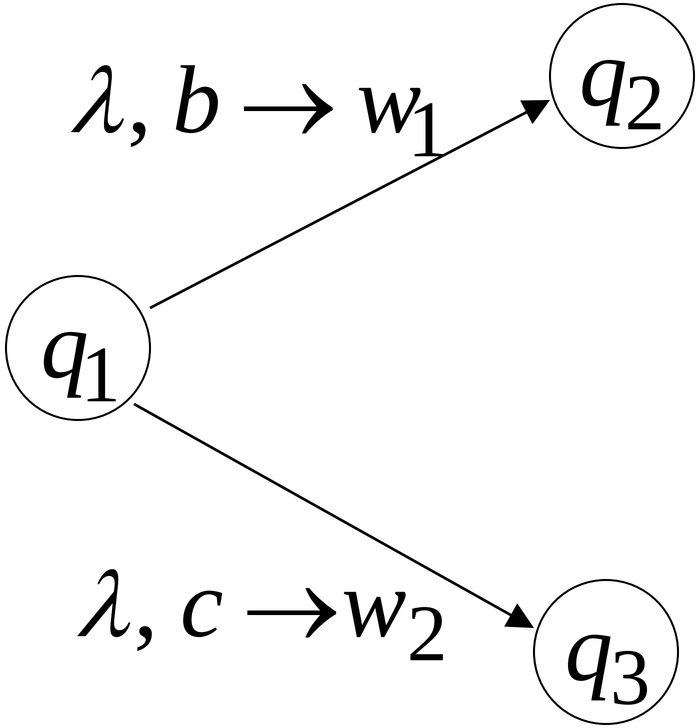




(deterministic choices)

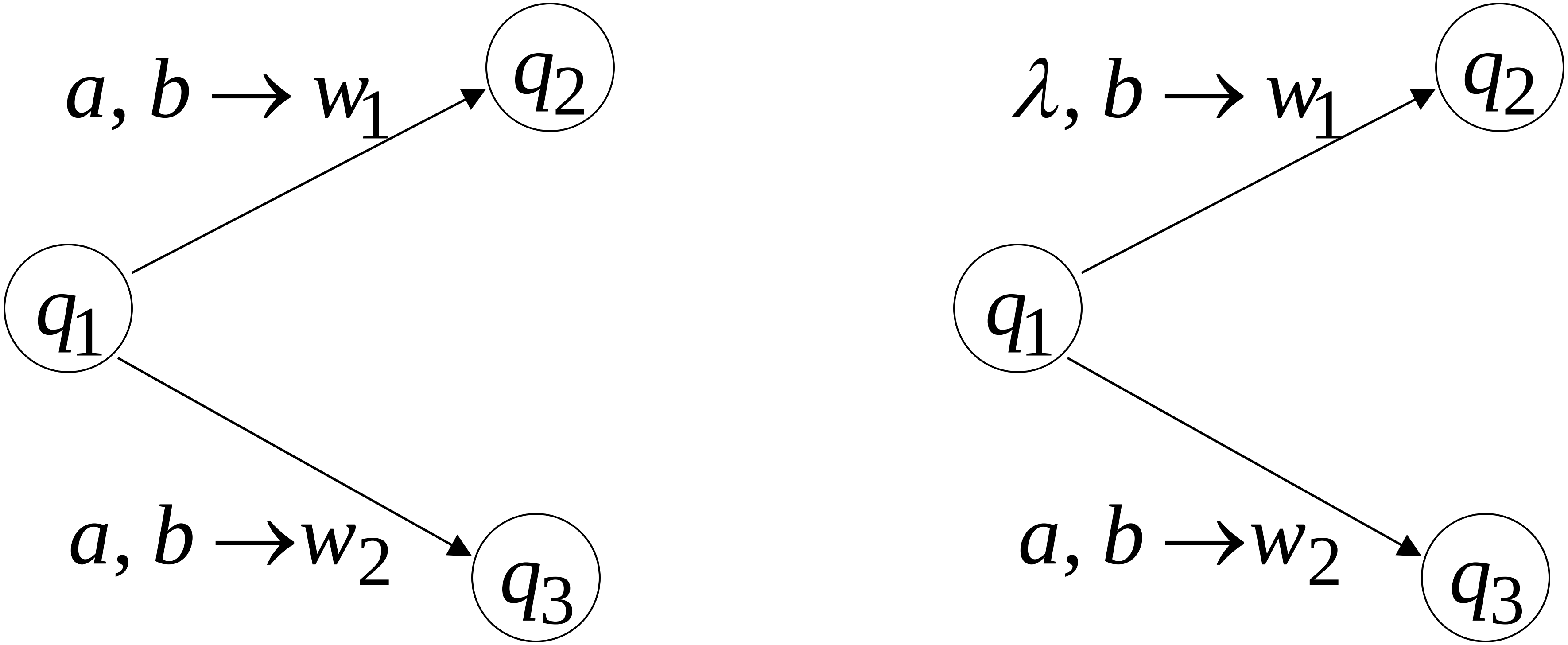
Allowed transitions:





(deterministic choices)

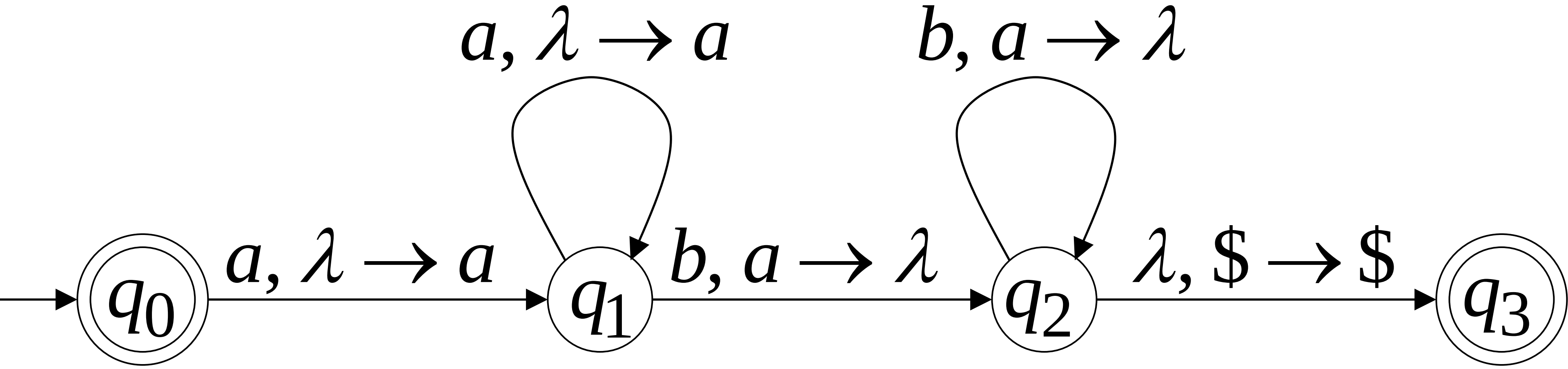
Not allowed:



(non deterministic choices)

DPDA example

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



Definition:

A language L is **deterministic context-free** if there exists some DPDA that accepts it

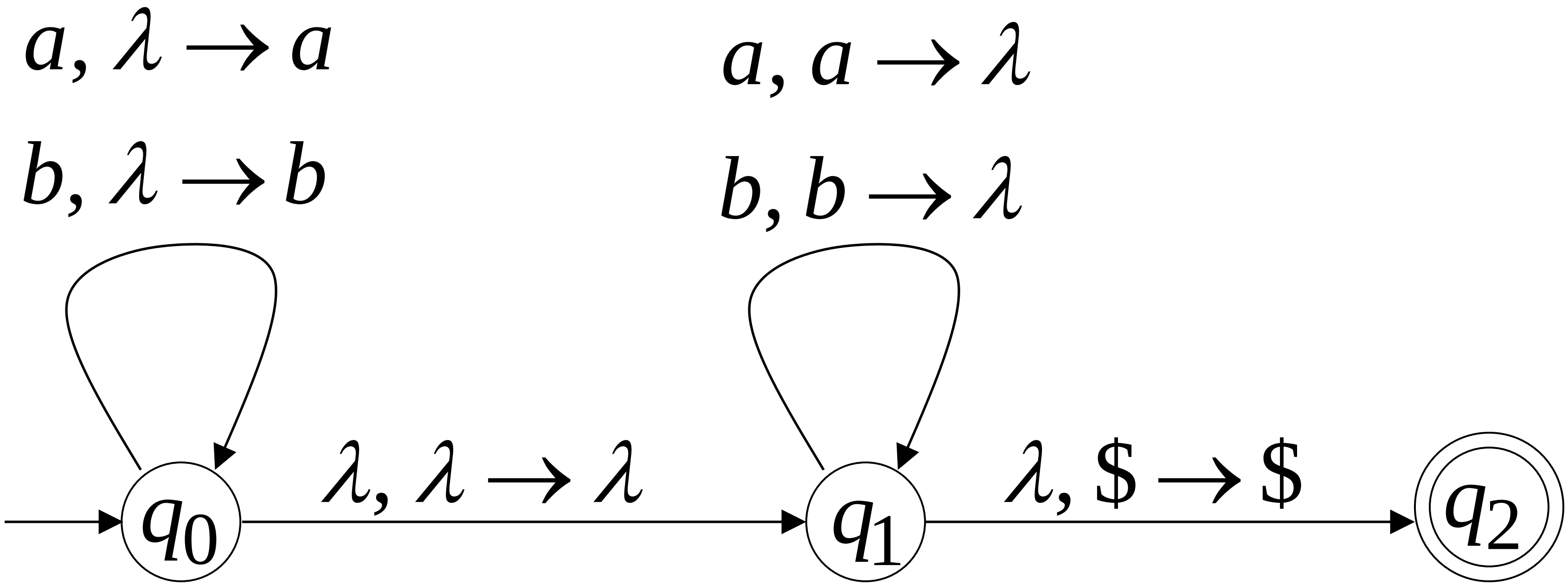
Example:

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

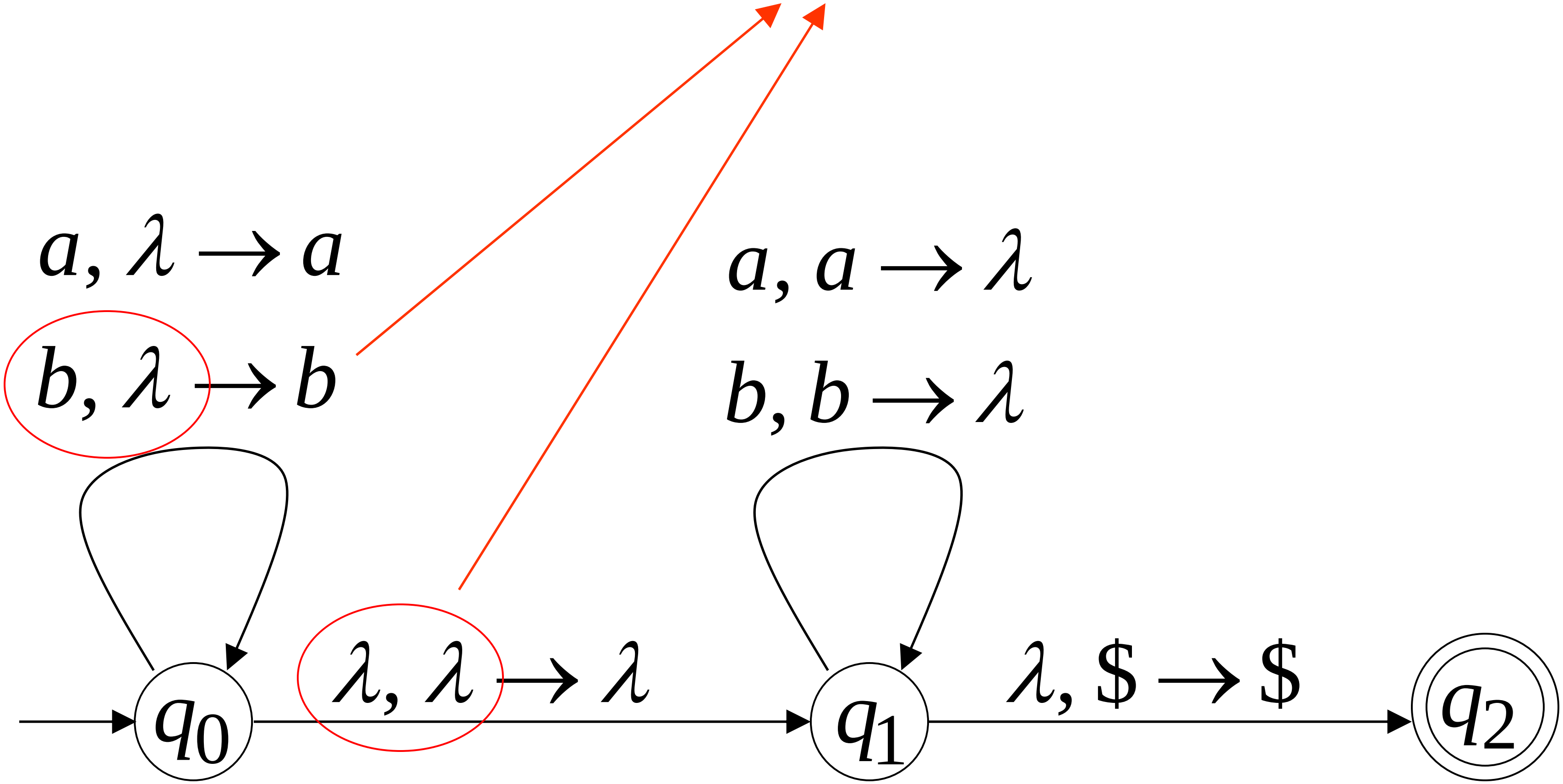
is **deterministic context-free**

Example of Non-DPDA (PDA)

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



Not allowed in DPDAs



PDAs

Have More Power than

DPDAs

It holds that:

$$\left\{\begin{array}{c}\text{Deterministic}\\ \text{Context-Free}\\ \text{Languages}\\ \text{(DPDA)}\end{array}\right\} \subseteq \left\{\begin{array}{c}\text{Context-Free}\\ \text{Languages}\\ \text{PDAs}\end{array}\right\}$$

Since every DPDA is also a PDA

We will actually show:

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

We will show that there exists a context-free language L which is not accepted by any DPDA

The language is:

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

We will show:

- L is context-free
- L is **not** deterministic context-free

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

Language L is context-free

Context-free grammar for L :

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

$$S_1 \to aS_1b \mid \lambda \qquad \{a^n b^n\}$$

$$S_2 \to aS_2bb \mid \lambda \qquad \{a^n b^{2n}\}$$

Theorem:

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

is **not** deterministic context-free

(there is **no** DPDA that accepts L)

Proof: Assume for contradiction that

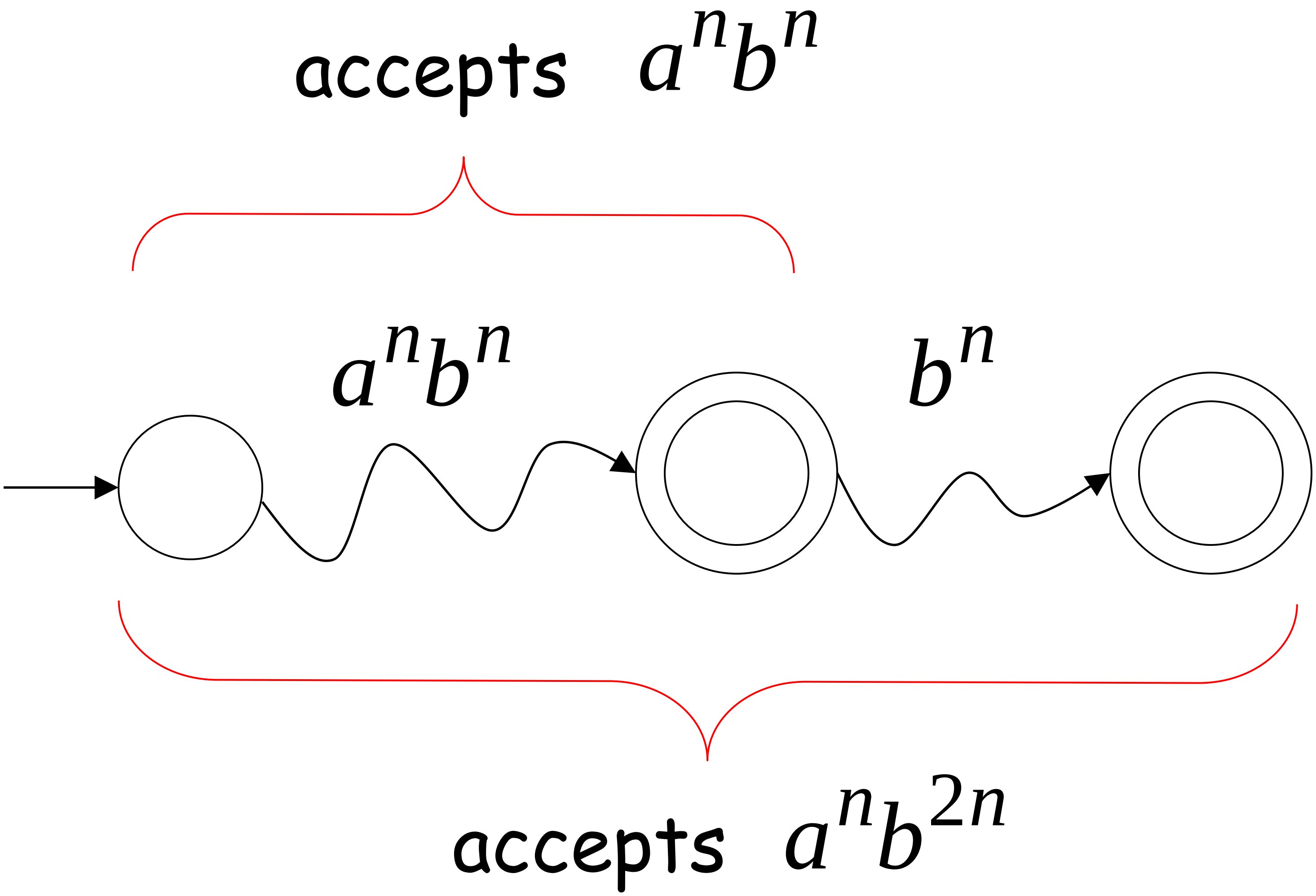
$$L = \{a^n b^n\} \cup \{a^n b^{2n}\}$$

is deterministic context free

Therefore:

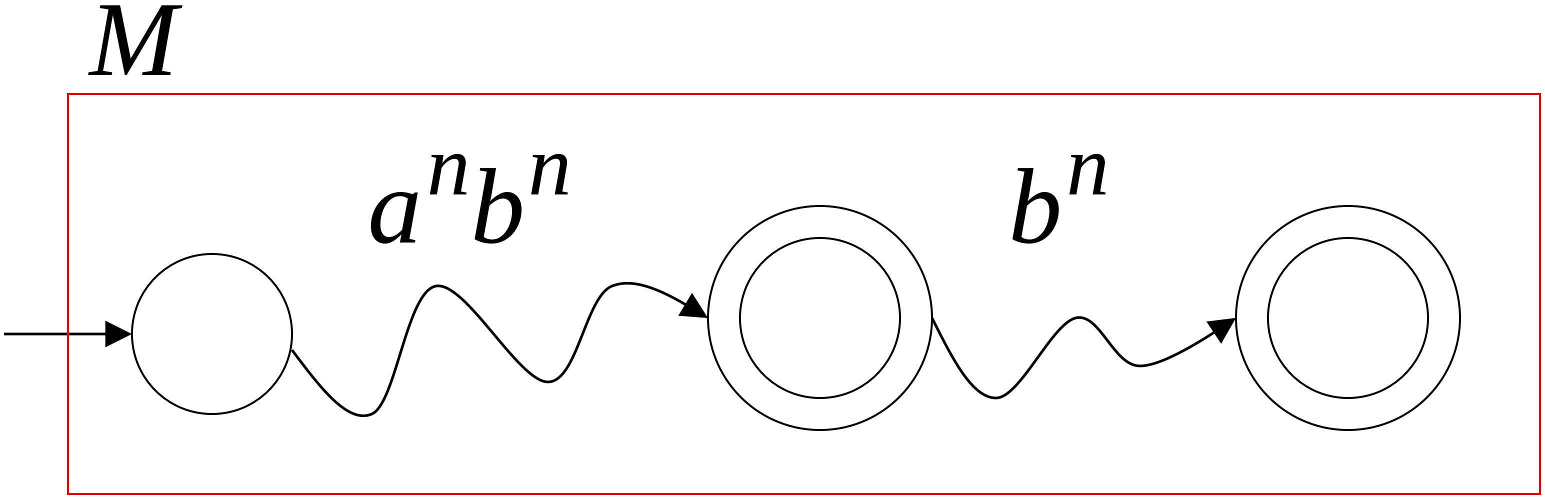
there is a DPDA M that accepts L

DPDA M with $L(M)=\{a^nb^n\}\cup\{a^nb^{2n}\}$

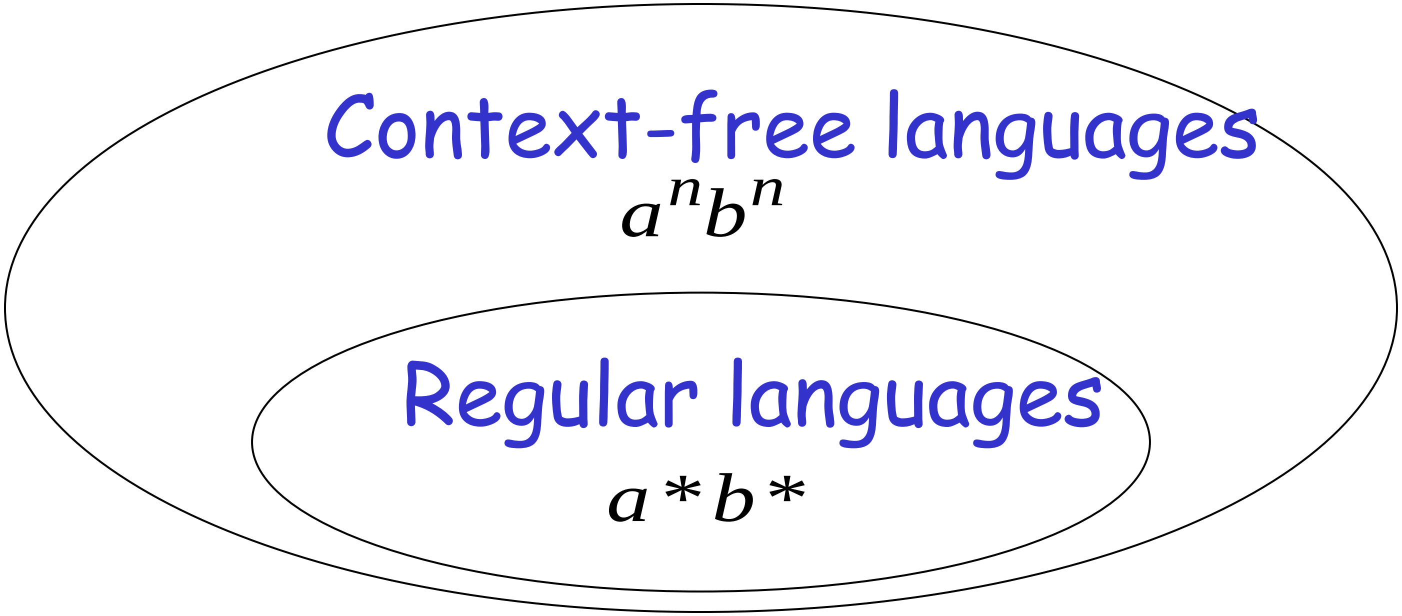


DPDA M with $L(M)=\{a^nb^n\}\cup\{a^nb^{2n}\}$

Such a path exists due to determinism



Fact 1: The language $\{a^n b^n c^n\}$ is **not** context-free


Context-free languages

$a^n b^n$

Regular languages

$a * b *$

Fact 2: The language $L \cup \{a^n b^n c^n\}$ is not context-free

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

(we can prove this using pumping lemma for context-free languages)

We will construct a PDA that accepts:

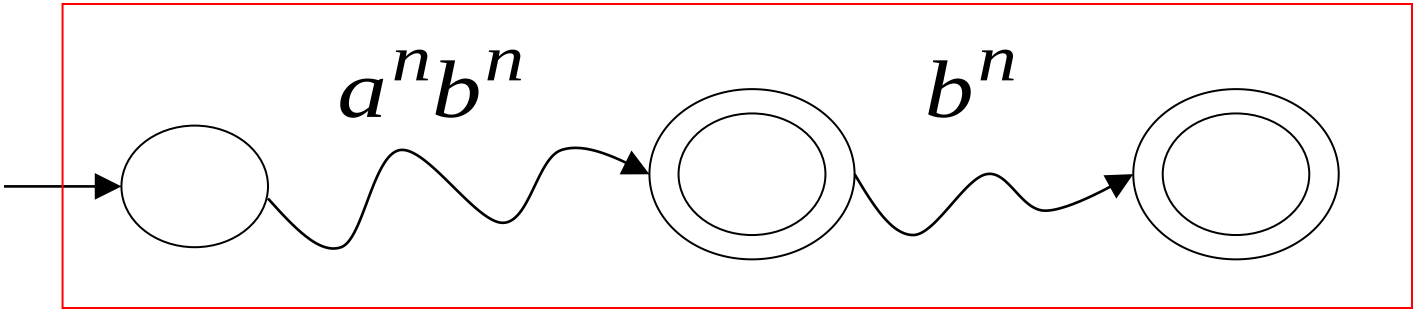
$$L \cup \{a^n b^n c^n\}$$

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

which is a contradiction!

DPDA M

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

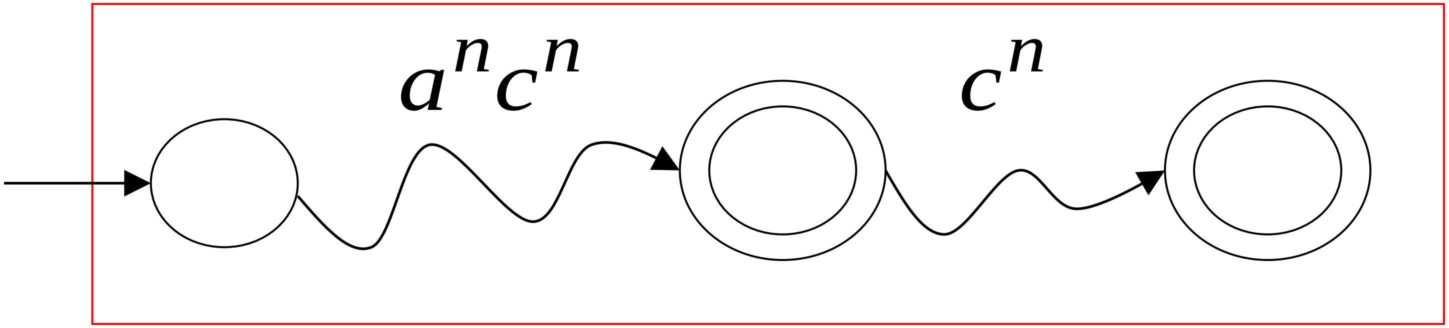


Modify M

Replace b with c

DPDA M'

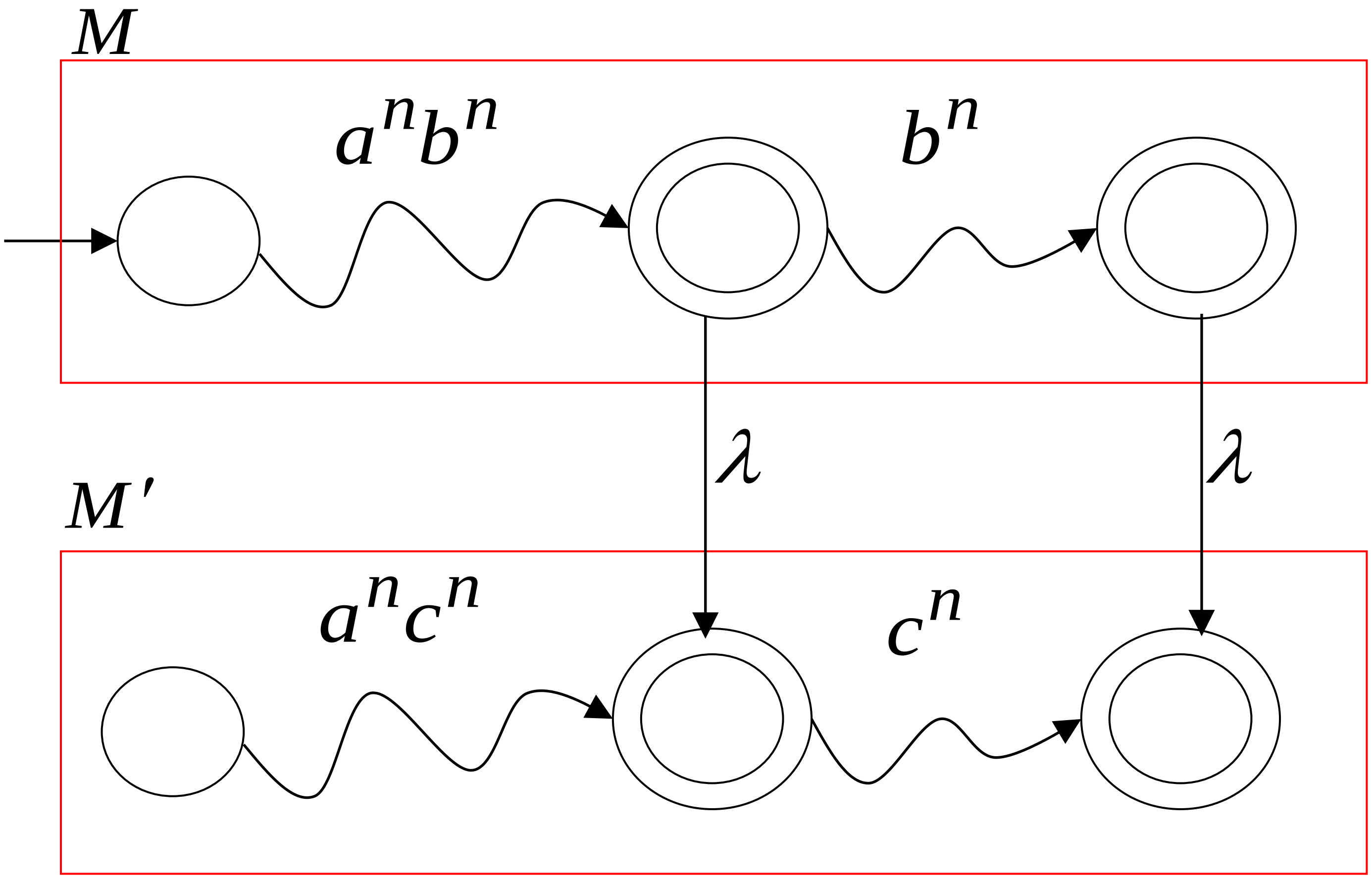
$$L(M')=\{a^nc^n\}\cup\{a^nc^{2n}\}$$



A PDA that accepts $L \cup \{a^n b^n c^n\}$

Connect the final states of M

with the final states of M'



Since $L \cup \{a^n b^n c^n\}$ is accepted by a PDA

it is context-free

Contradiction!

(since $L \cup \{a^n b^n c^n\}$ is not context-free)

Therefore:

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

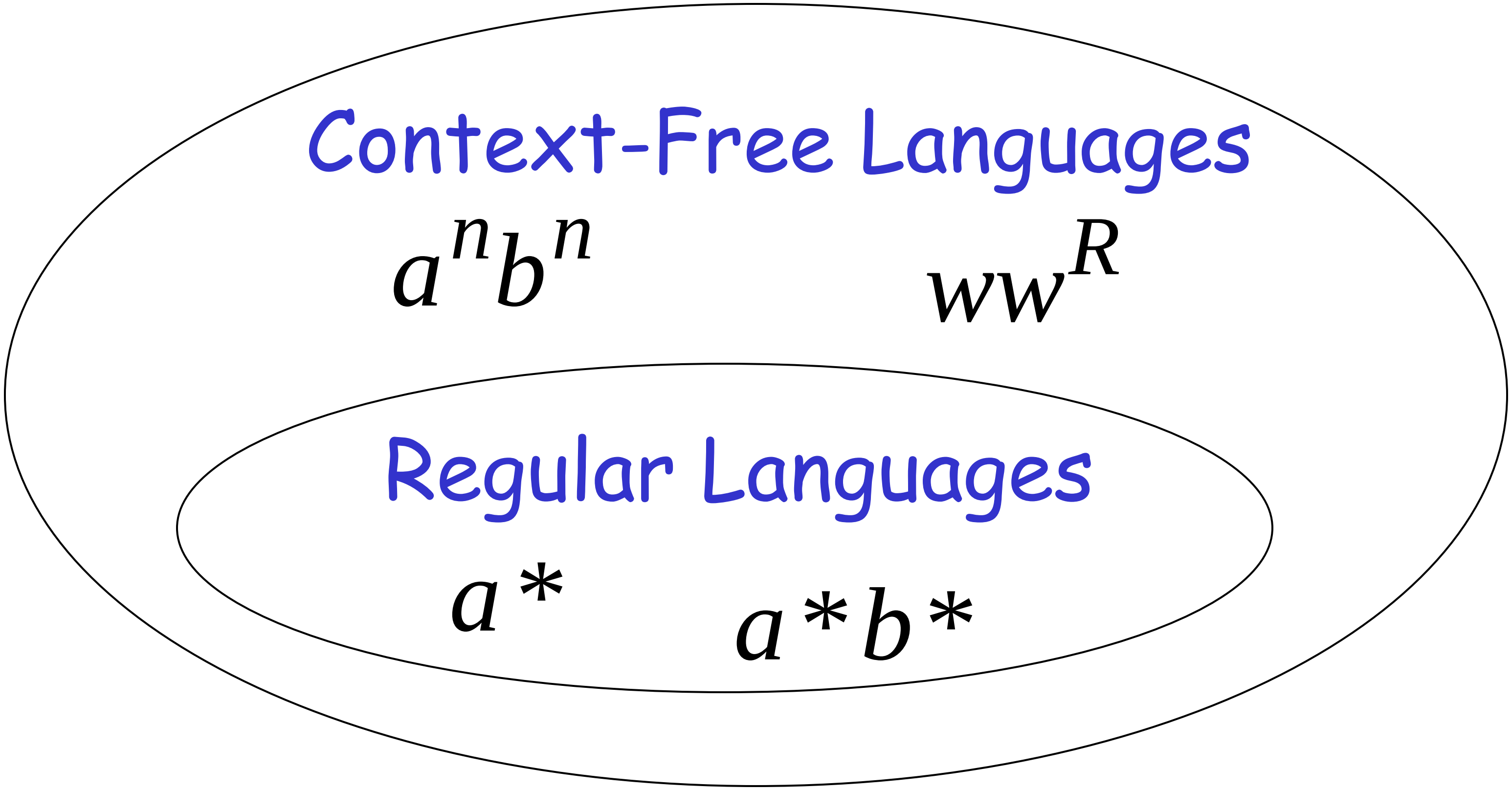
Is not deterministic context free

There is **no** DPDA that accepts it

End of Proof

Turing Machines

The Language Hierarchy



$a^n b^n c^n$?

ww ?

Context-Free Languages

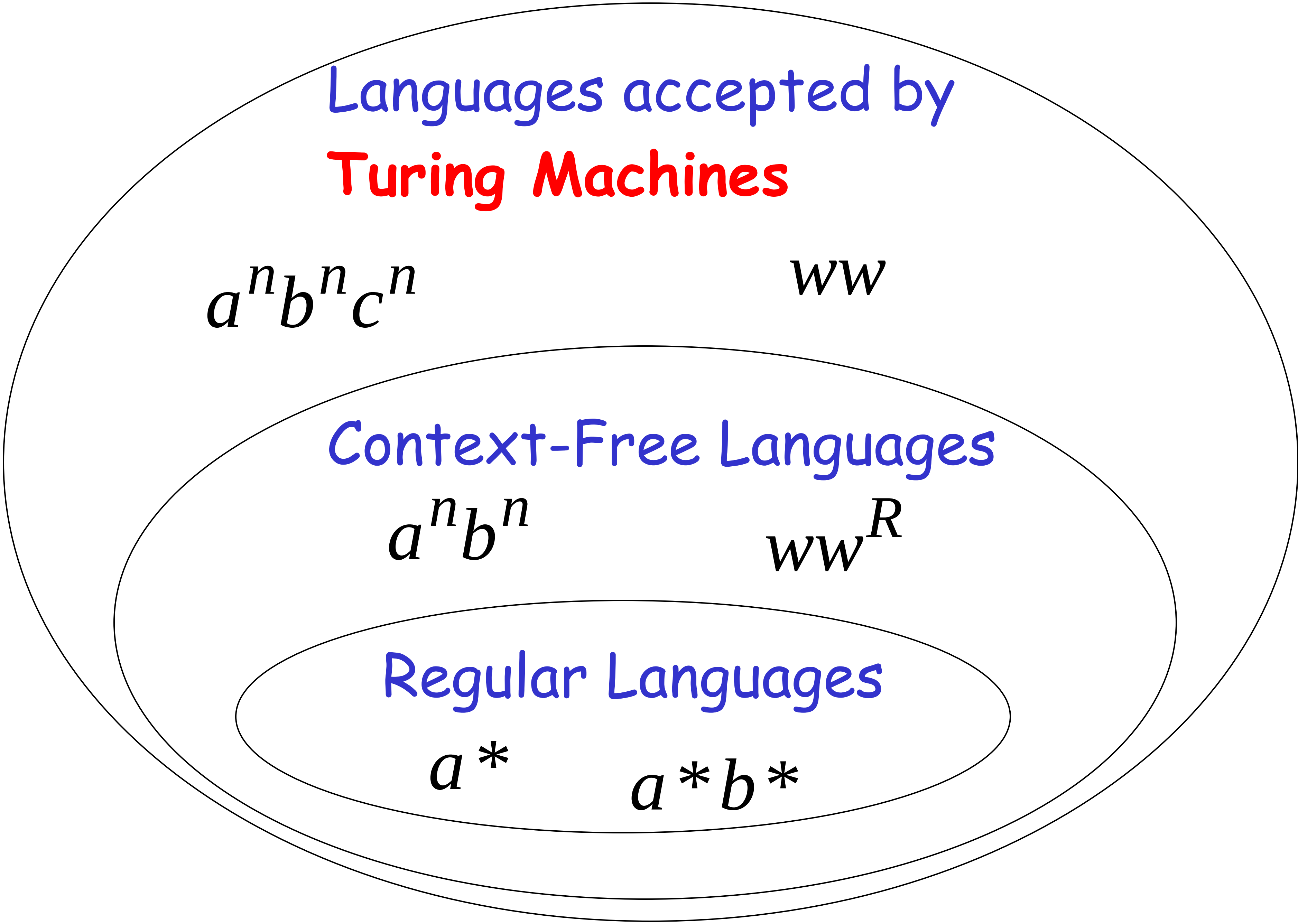
$a^n b^n$

ww^R

Regular Languages

a^*

$a^* b^*$



Languages accepted by **Turing Machines**

$a^n b^n c^n$ ww

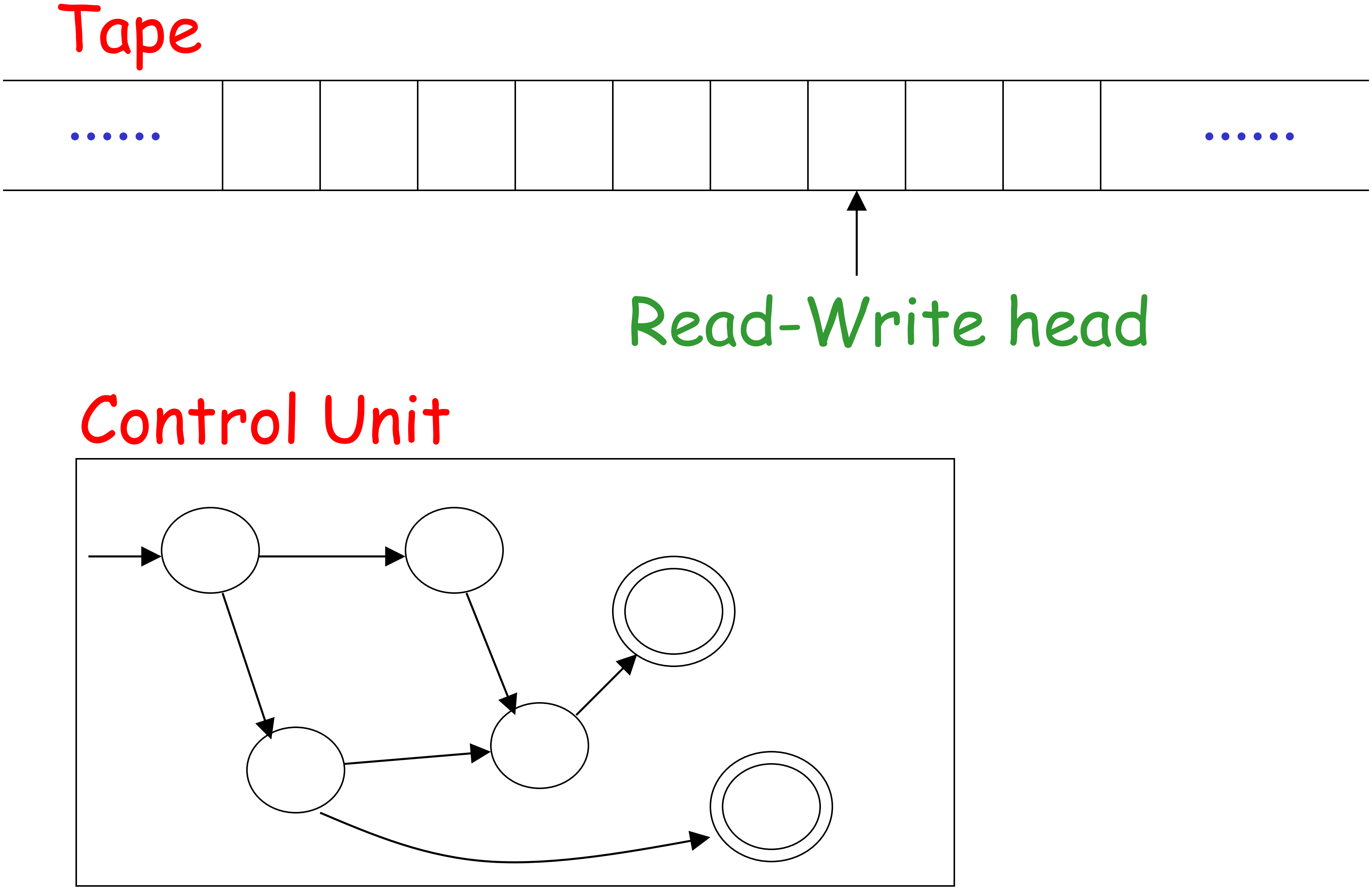
Context-Free Languages

$a^n b^n$ ww^R

Regular Languages

a^* a^*b^*

A Turing Machine



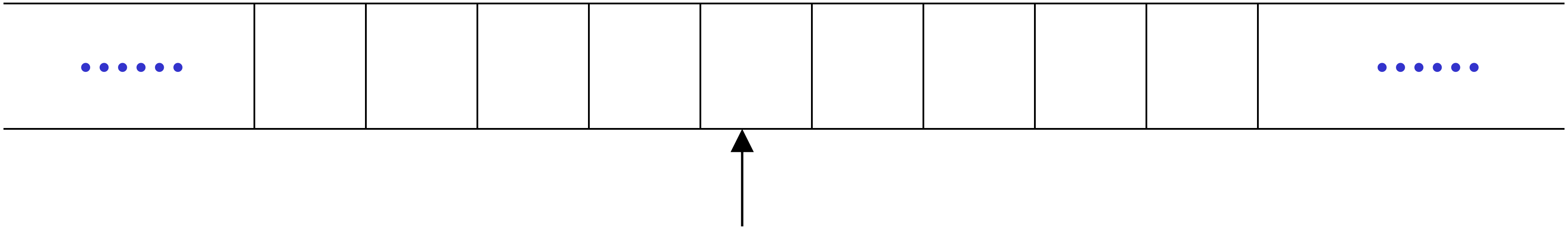
Tape

Read-Write head

Control Unit

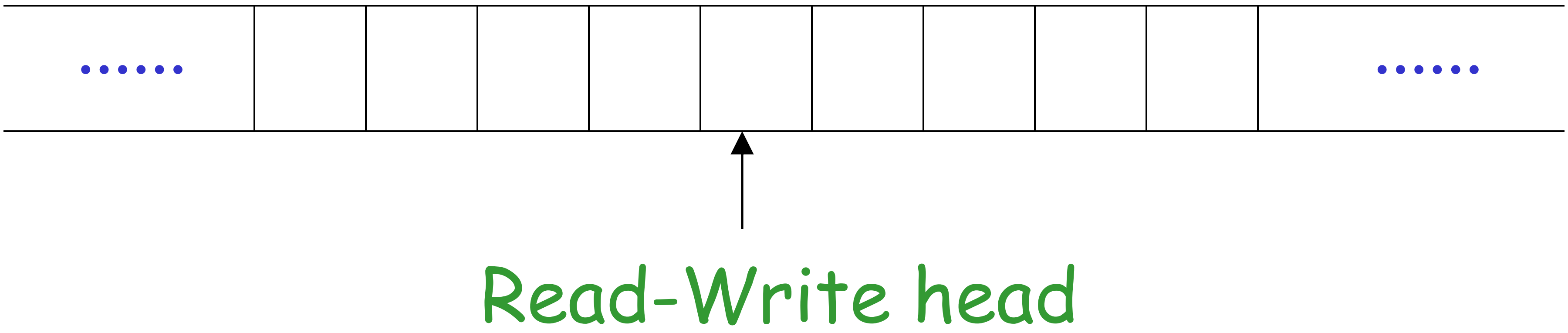
The Tape

No boundaries -- infinite length



Read-Write head

The head moves Left or Right

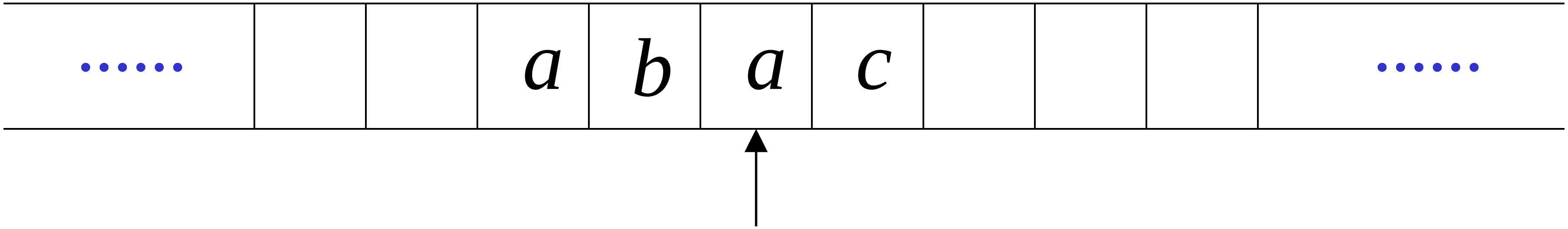


The head at each transition (time step):

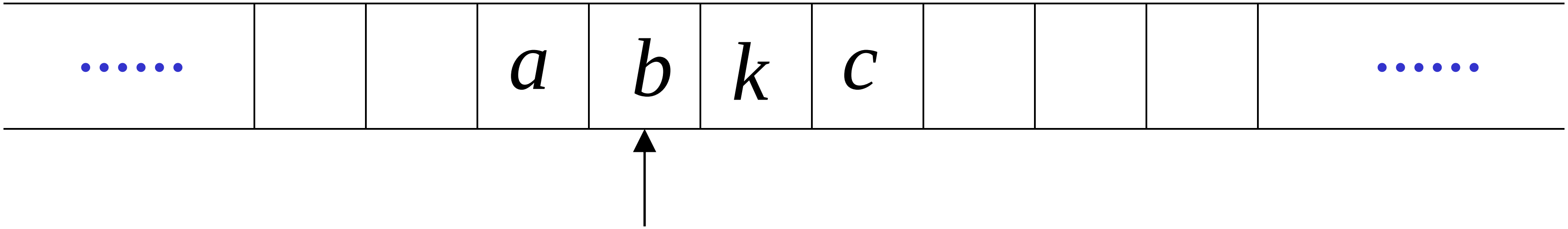
1. Reads a symbol
2. Writes a symbol
3. Moves Left or Right

Example:

Time 0

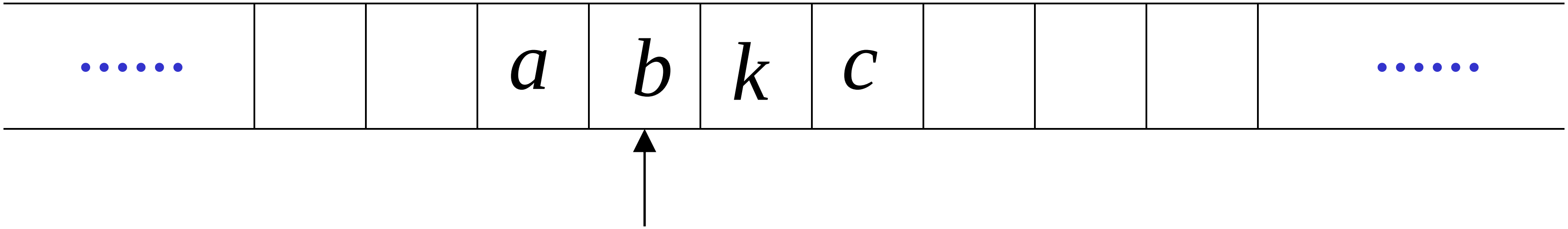


Time 1

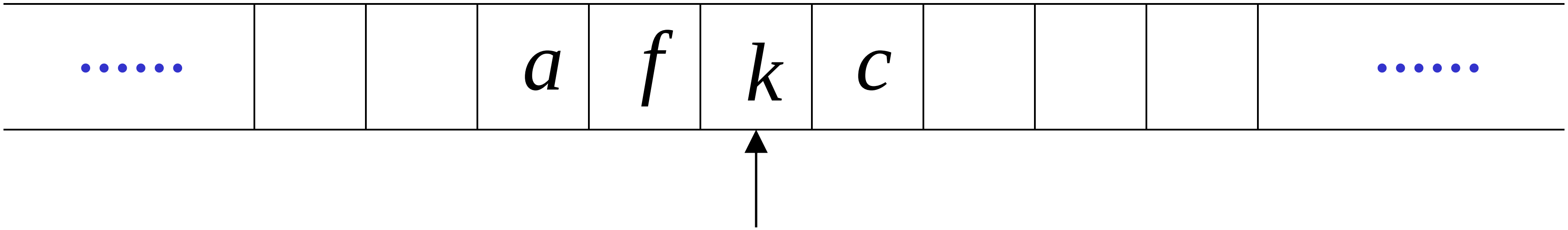


1. Reads a
2. Writes k
3. Moves Left

Time 1

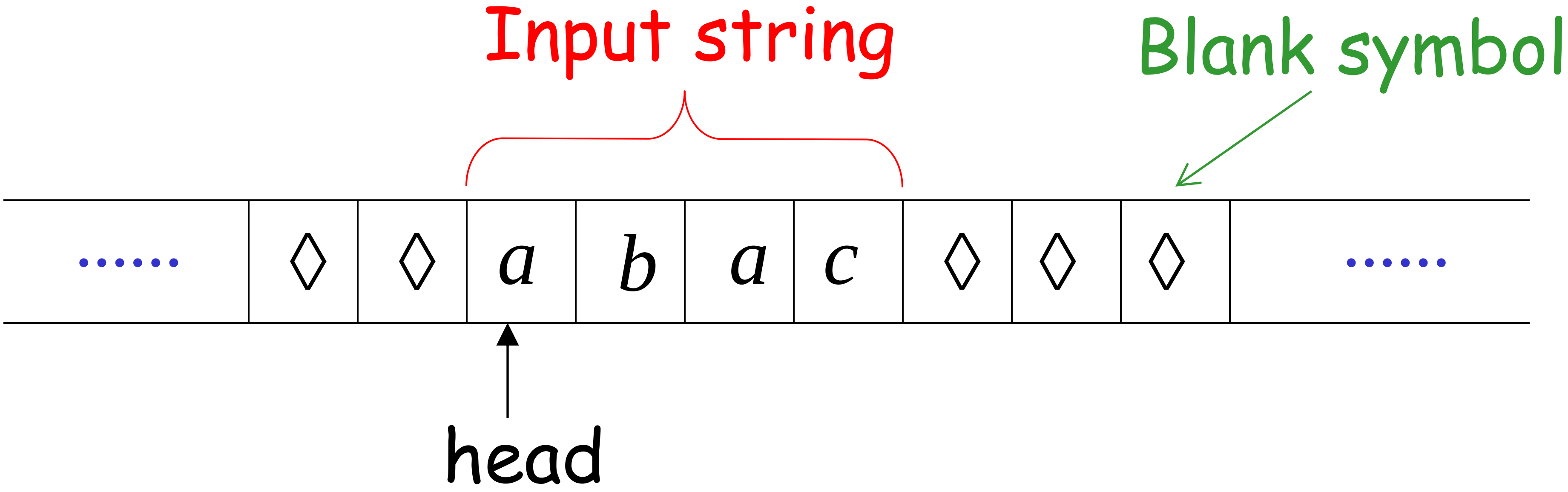


Time 2



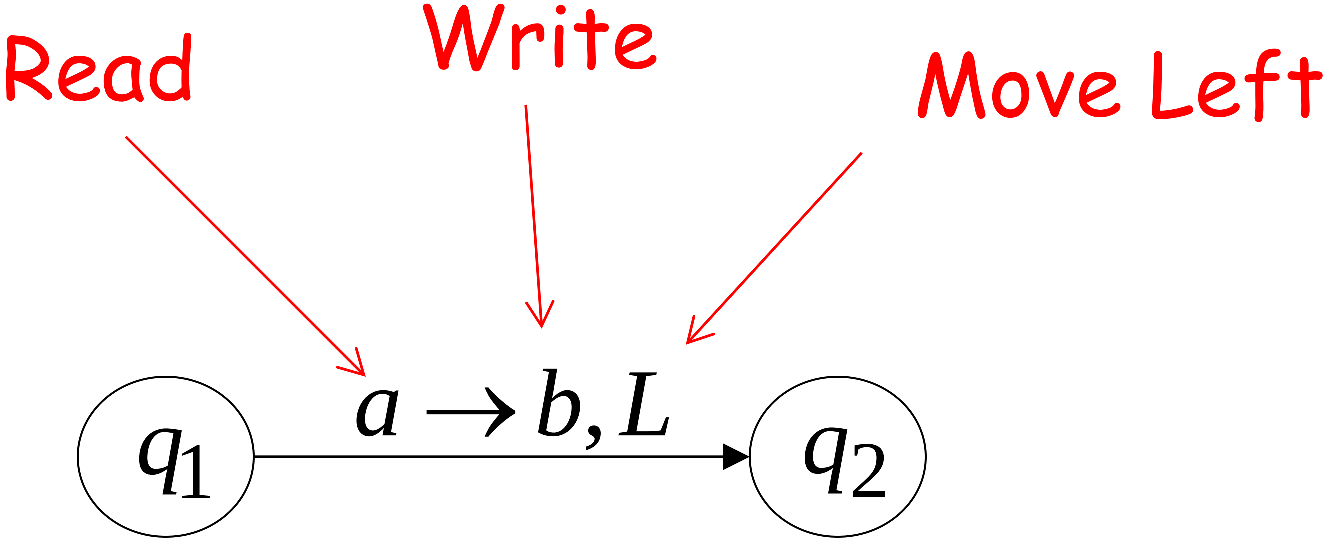
1. Reads b
2. Writes f
3. Moves Right

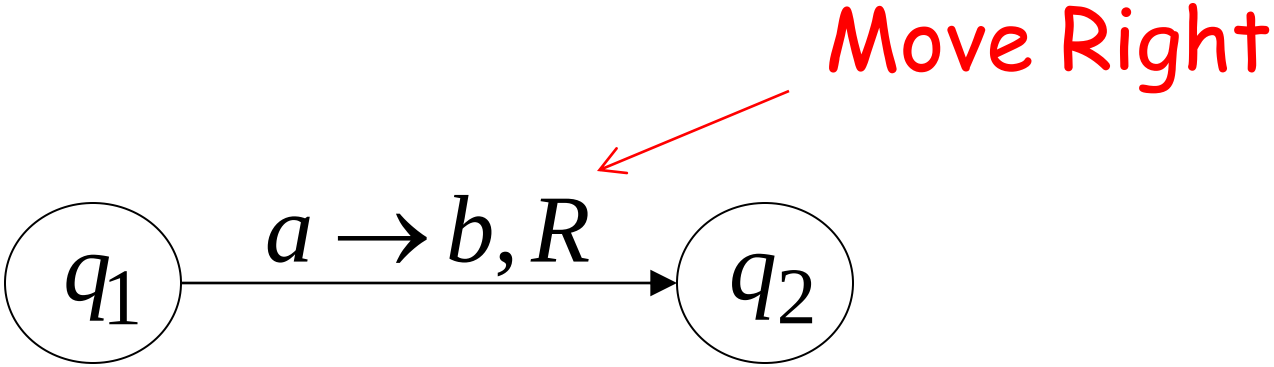
The Input String



Head starts at the leftmost position of the input string

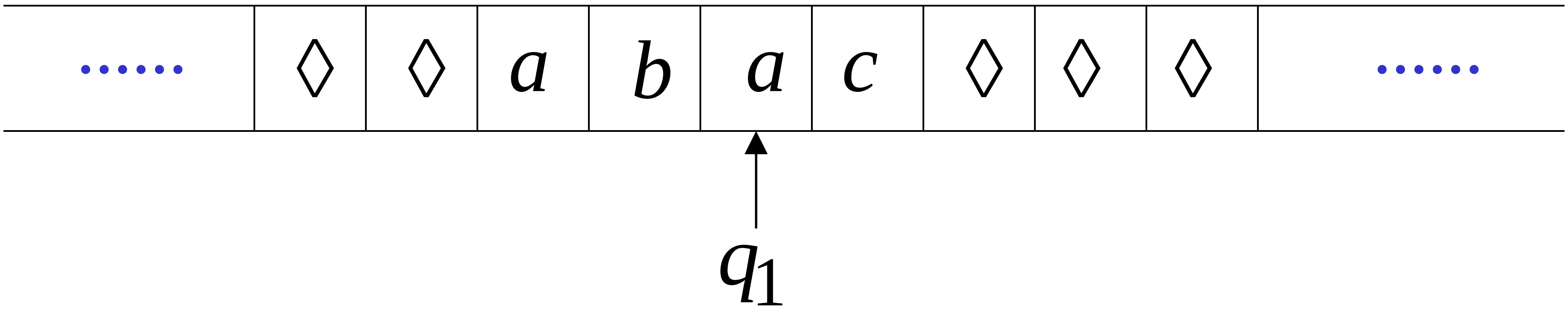
States & Transitions



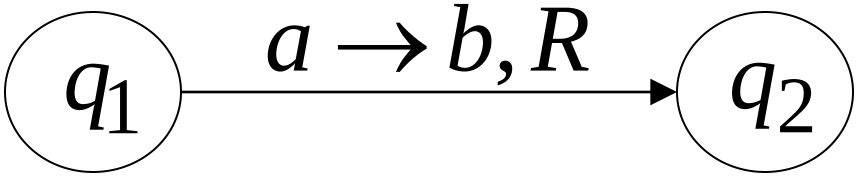


Example:

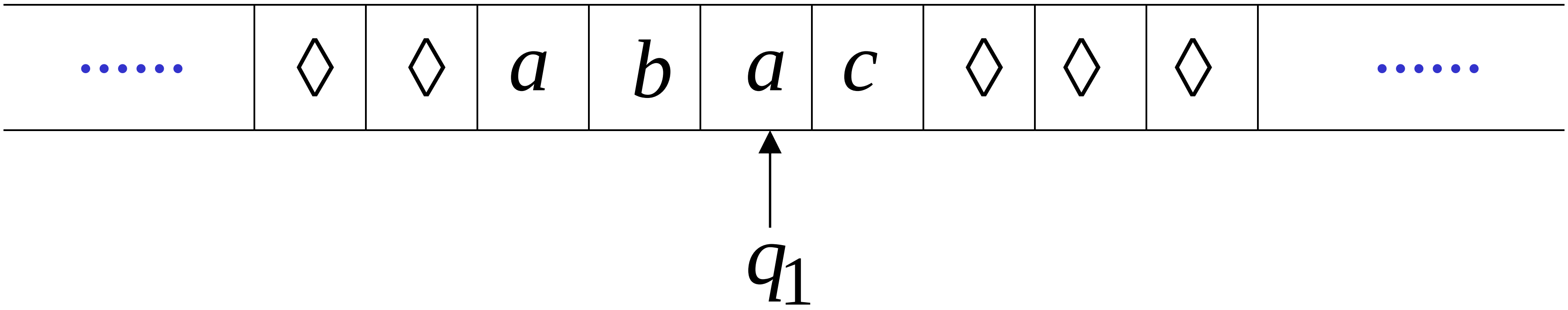
Time 1



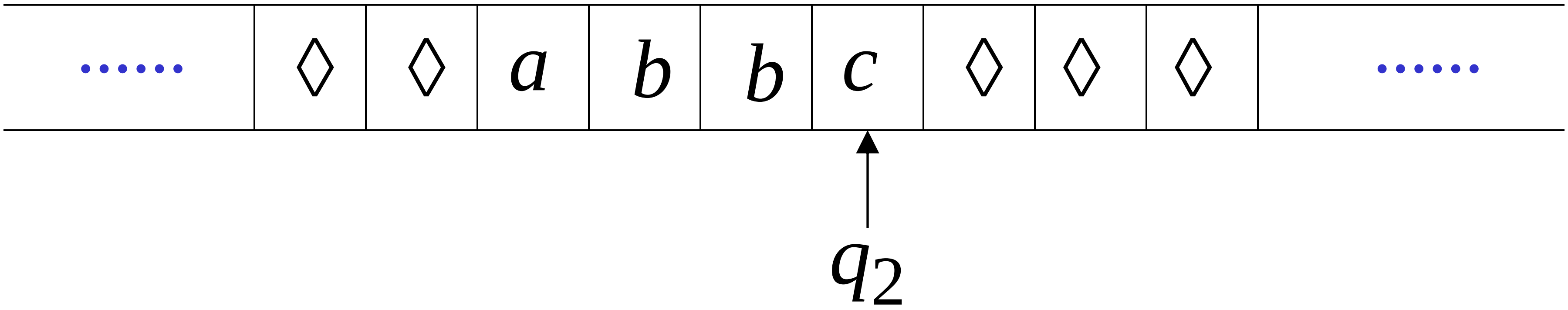
current state

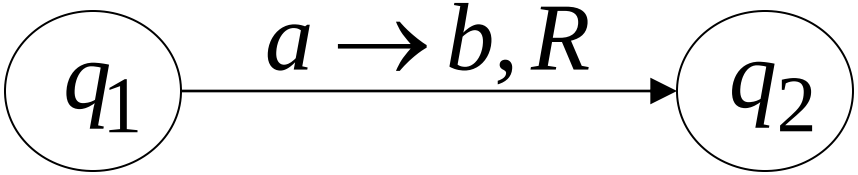


Time 1



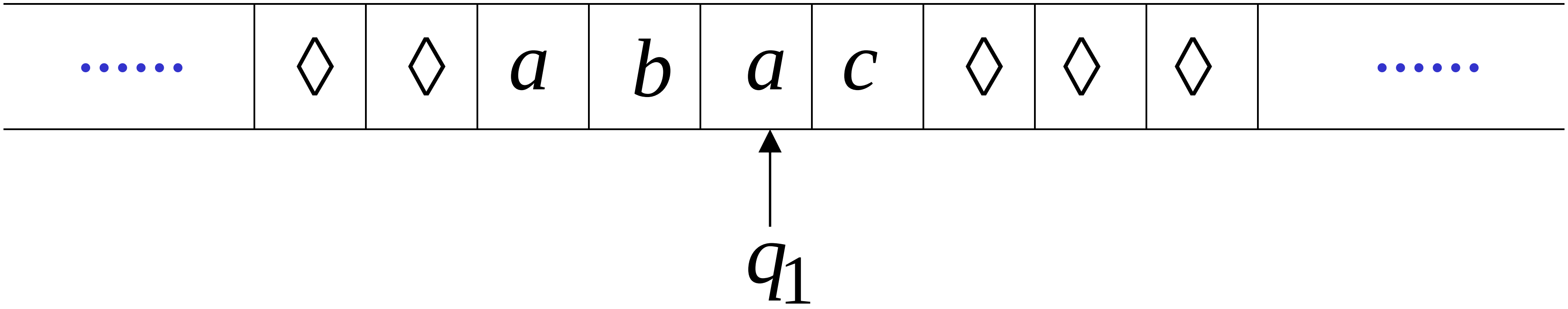
Time 2



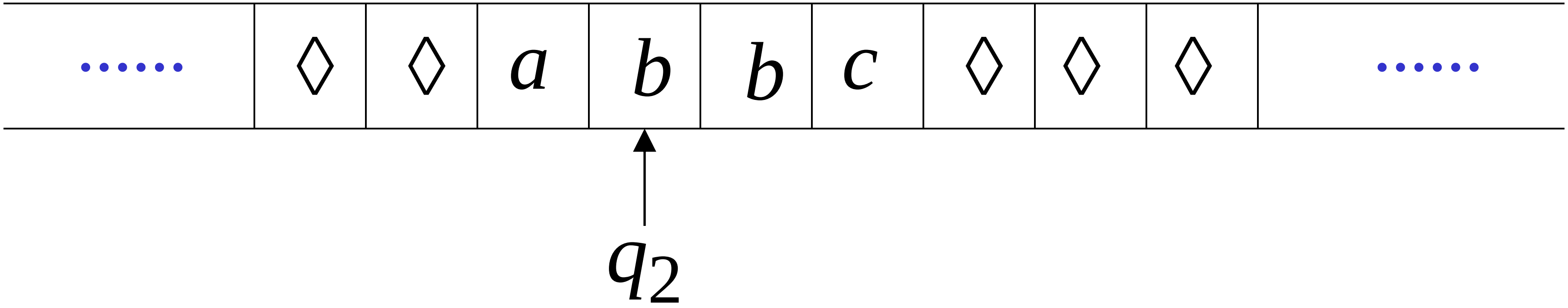


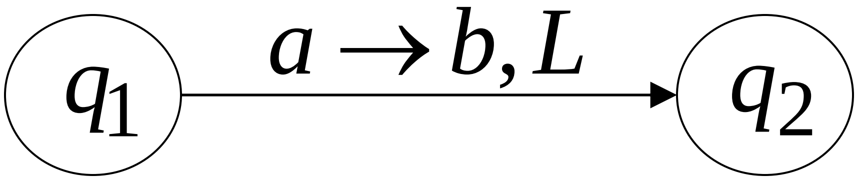
Example:

Time 1



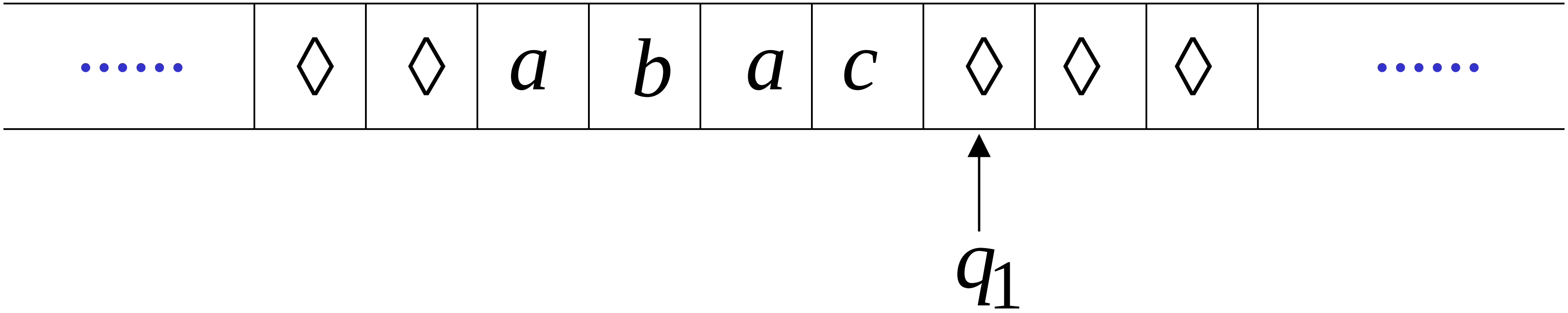
Time 2



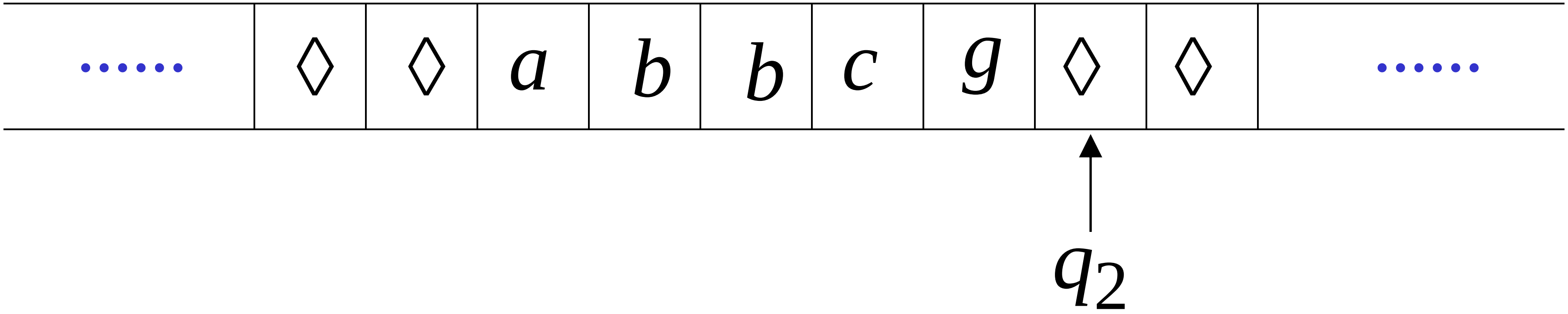


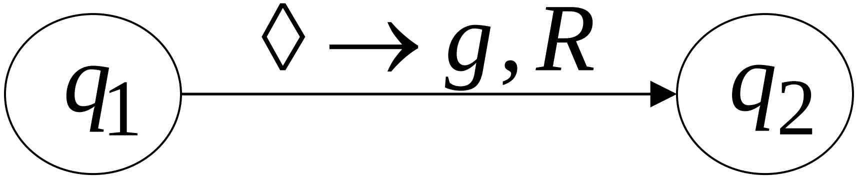
Example:

Time 1



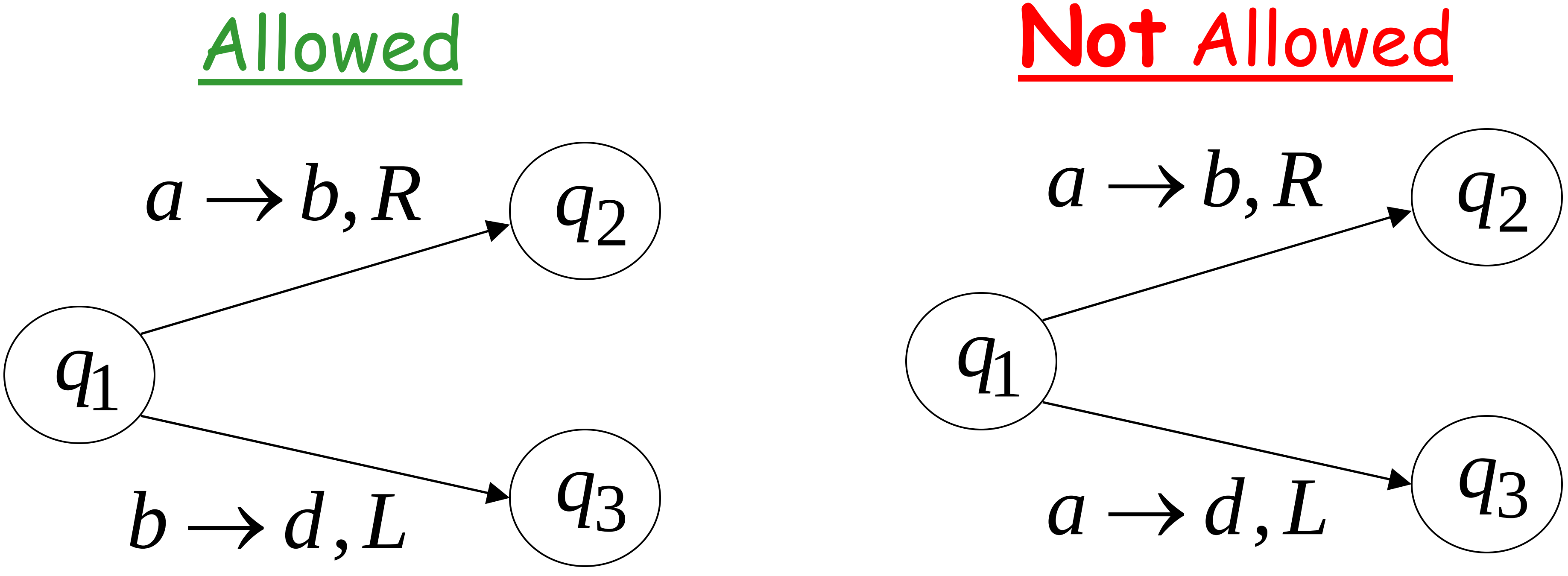
Time 2





Determinism

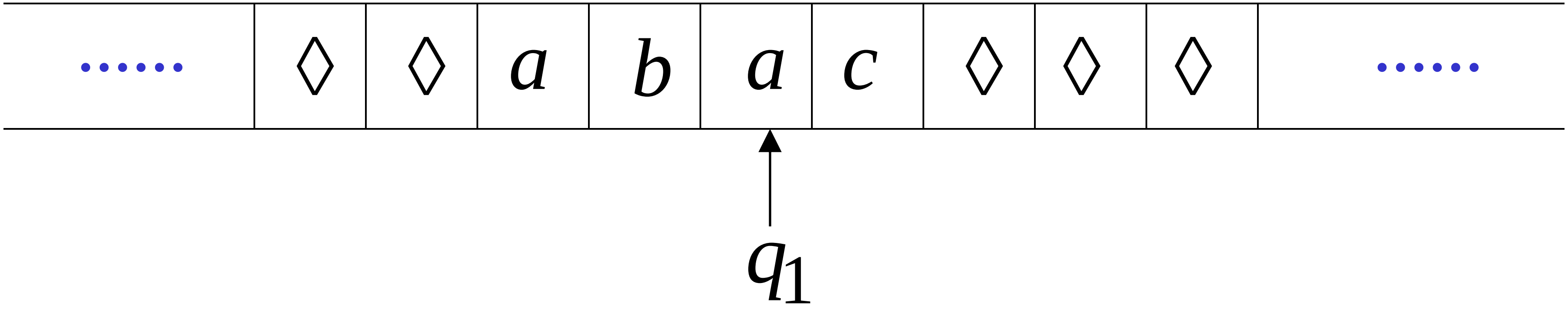
Turing Machines are deterministic

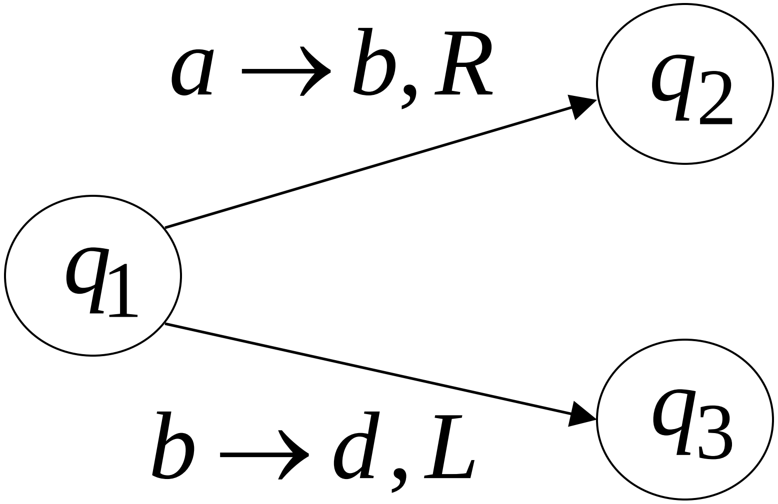


No ϵ-transitions allowed

Partial Transition Function

Example:





<u>Allowed:</u>

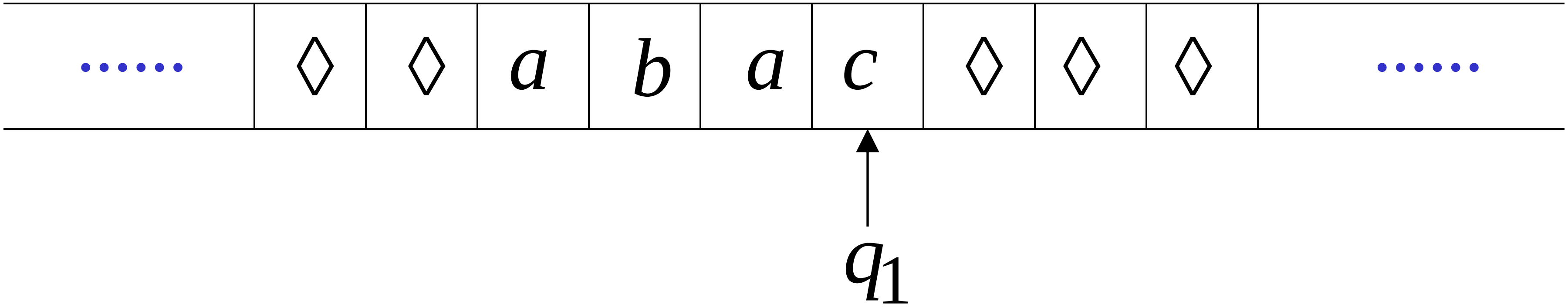
No transition

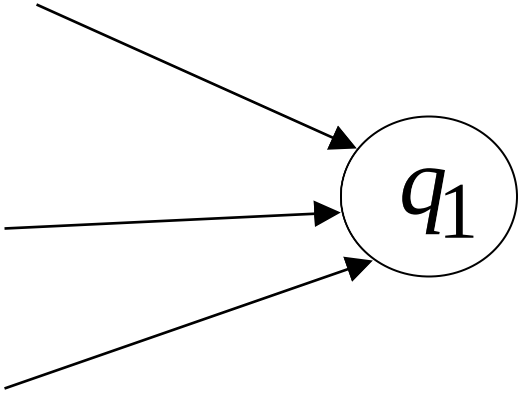
for input symbol c

Halting

The machine ***halts*** in a state if there is no transition to follow

Halting Example 1:

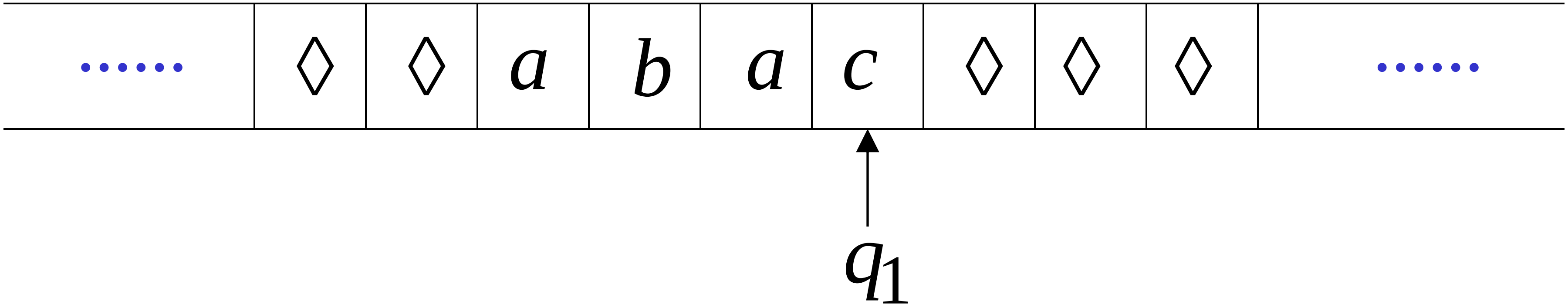


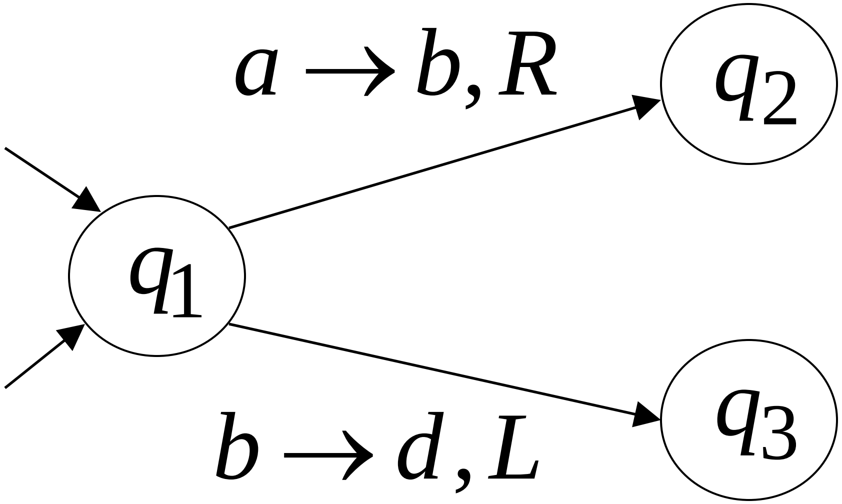


No transition from q_1

HALT!!!

Halting Example 2:

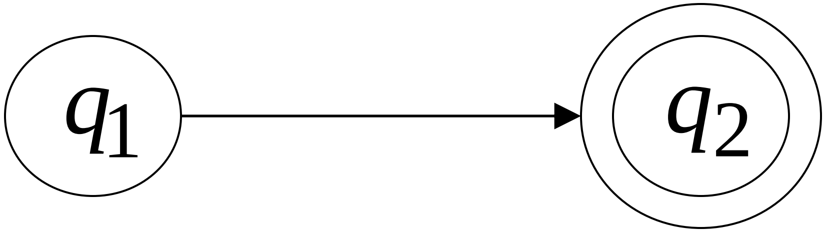




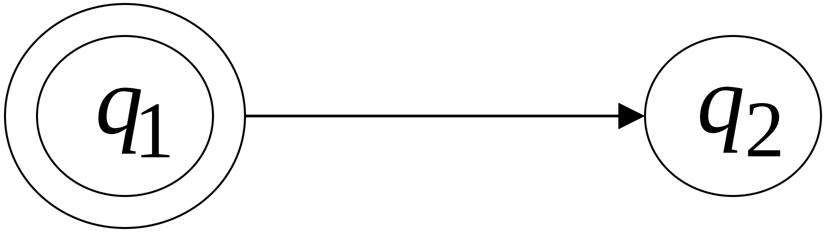
No possible transition from q_1 and symbol c

HALT!!!

Accepting States



Allowed



Not Allowed

- Accepting states have no outgoing transitions
- The machine halts and accepts

Acceptance

Accept Input string → If machine halts in an accept state

Reject Input string → If machine halts in a non-accept state

or

If machine enters an *infinite loop*

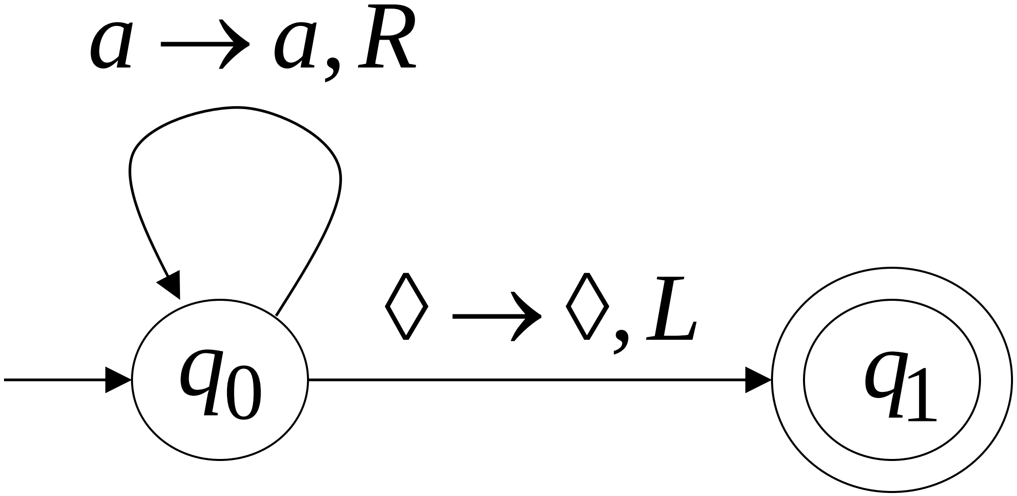
Observation:

In order to accept an input string, it is not necessary to scan all the symbols of the input string

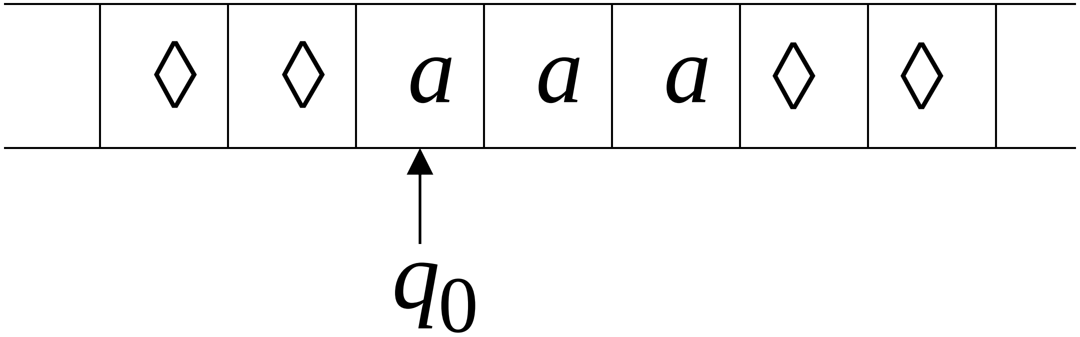
Turing Machine Example

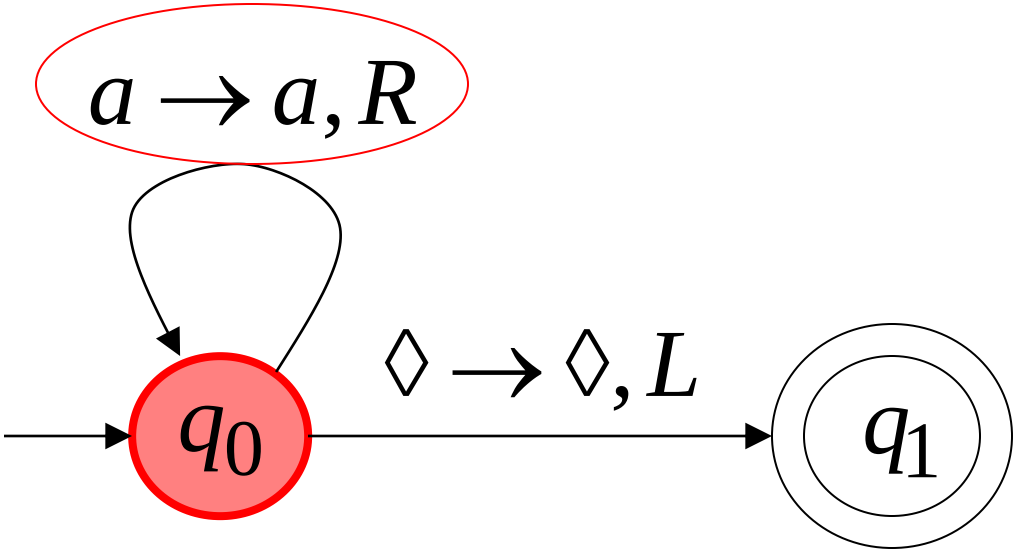
Input alphabet $\Sigma = \{a, b\}$

Accepts the language: a^*

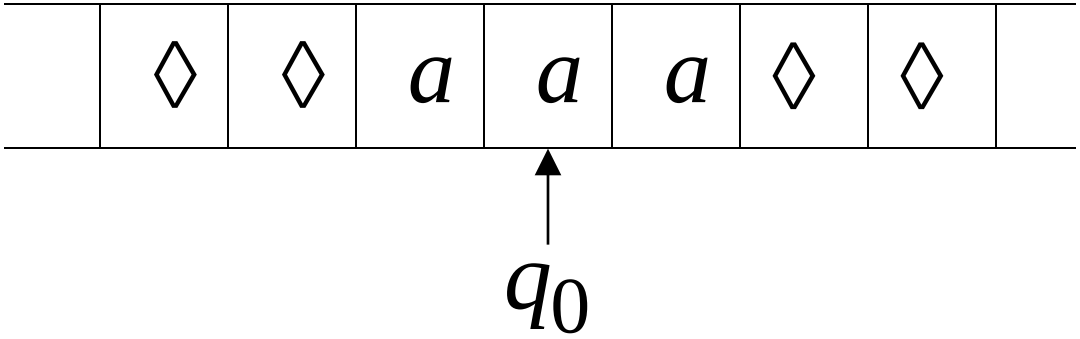


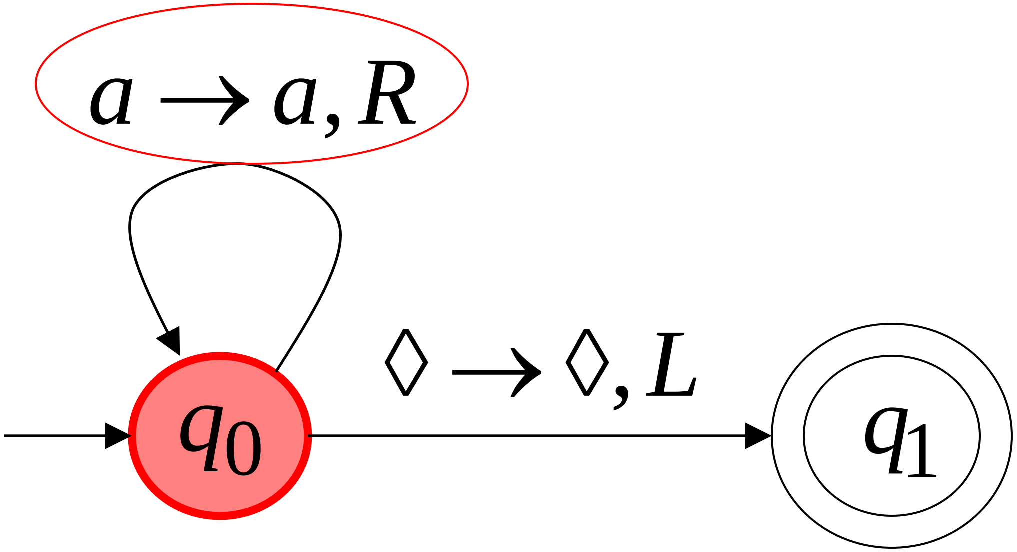
Time 0


$\Diamond$
$\Diamond$
a
a
a
$\Diamond$
$\Diamond$
q_0

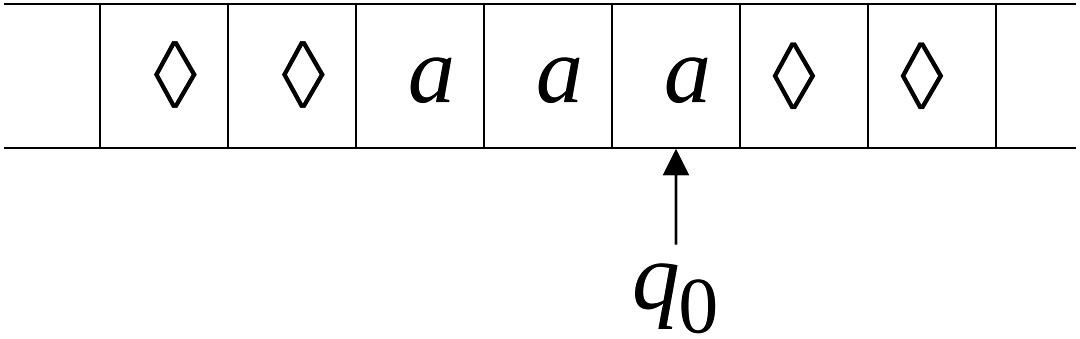

$a \rightarrow a, R$
$\Diamond \rightarrow \Diamond, L$
q_0
q_1

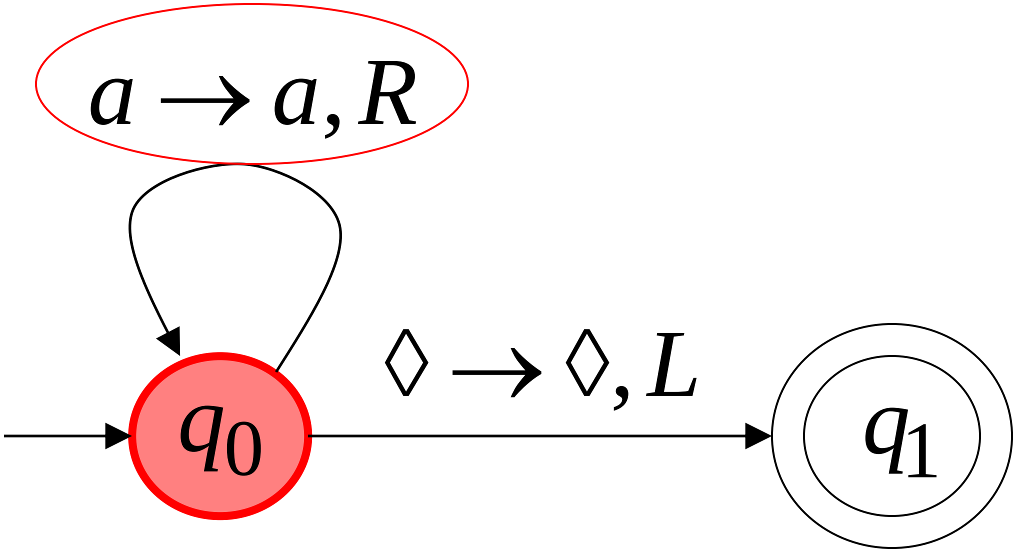
Time 1



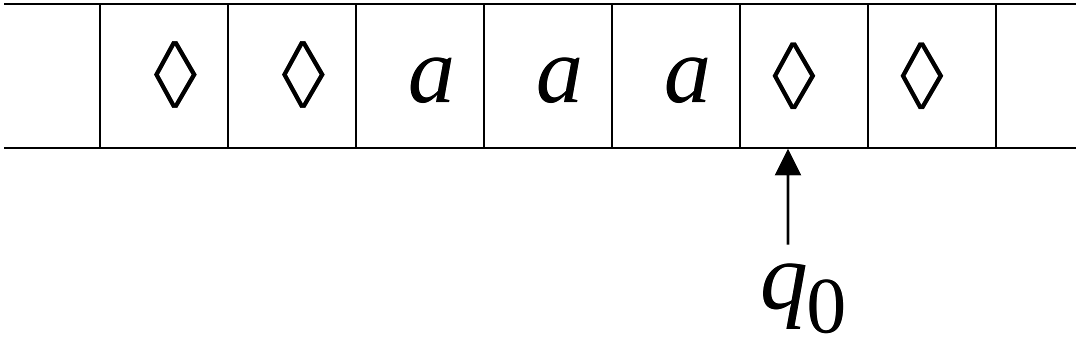


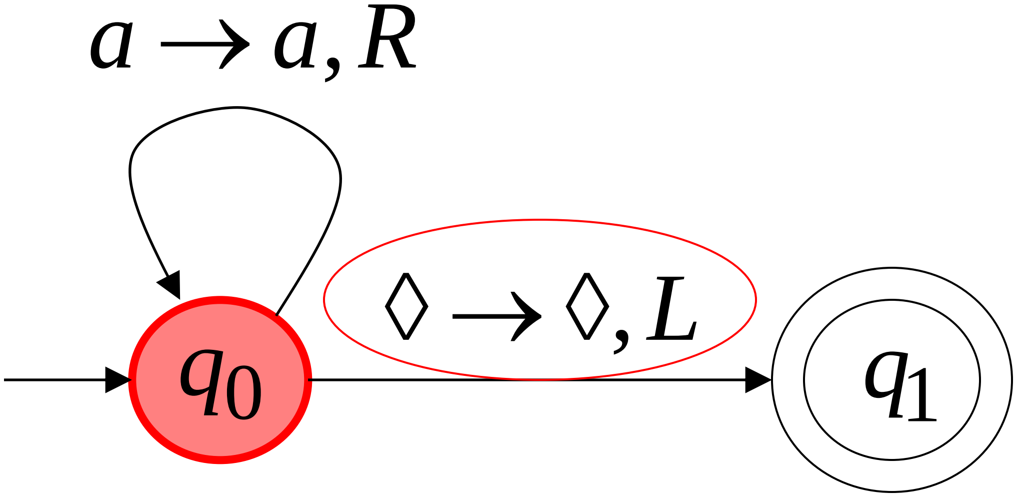
Time 2



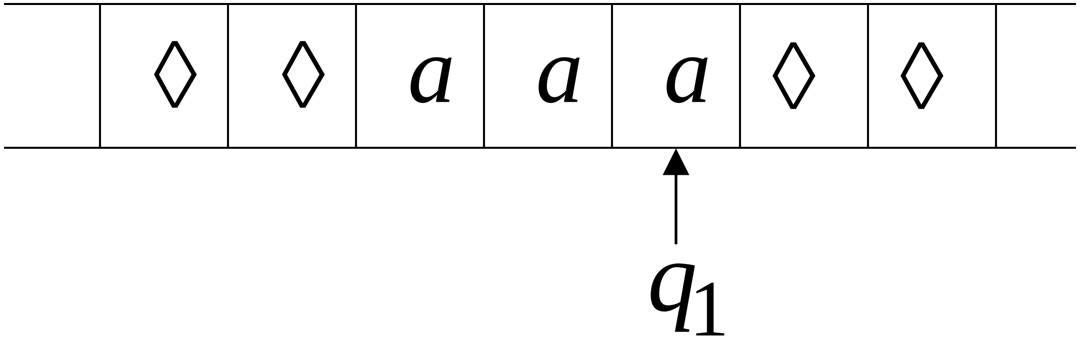


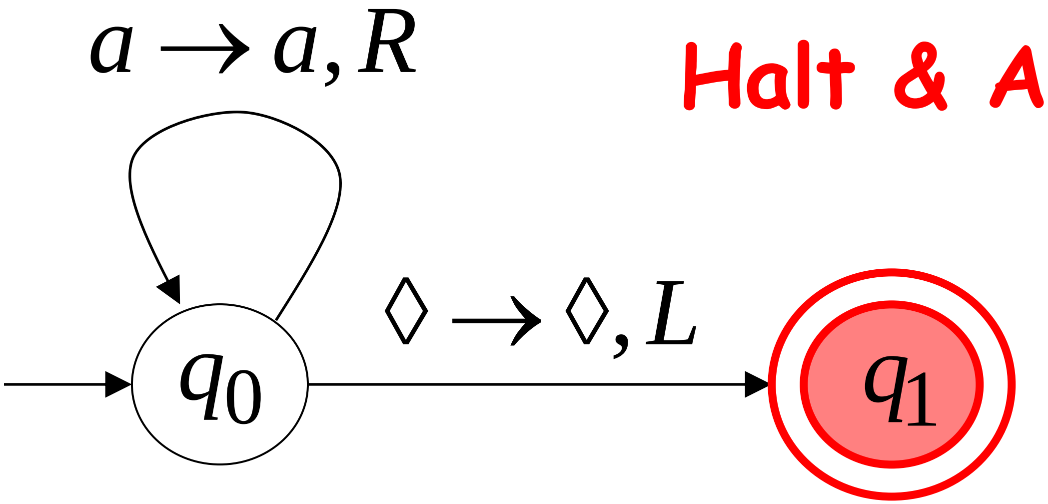
Time 3





Time 4

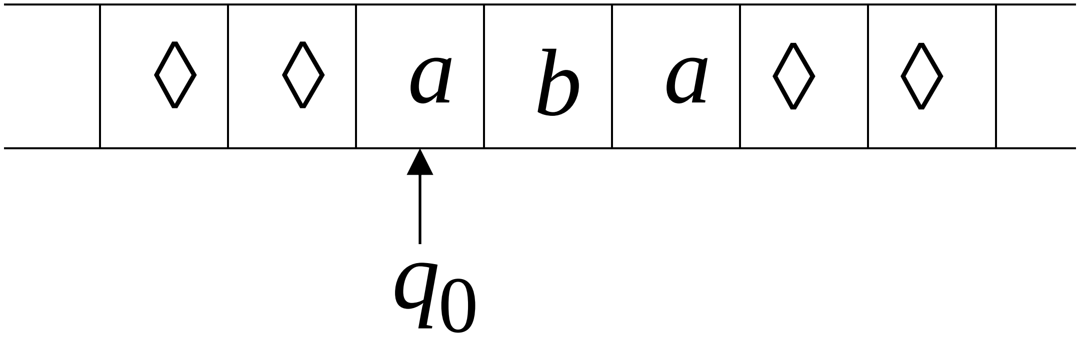


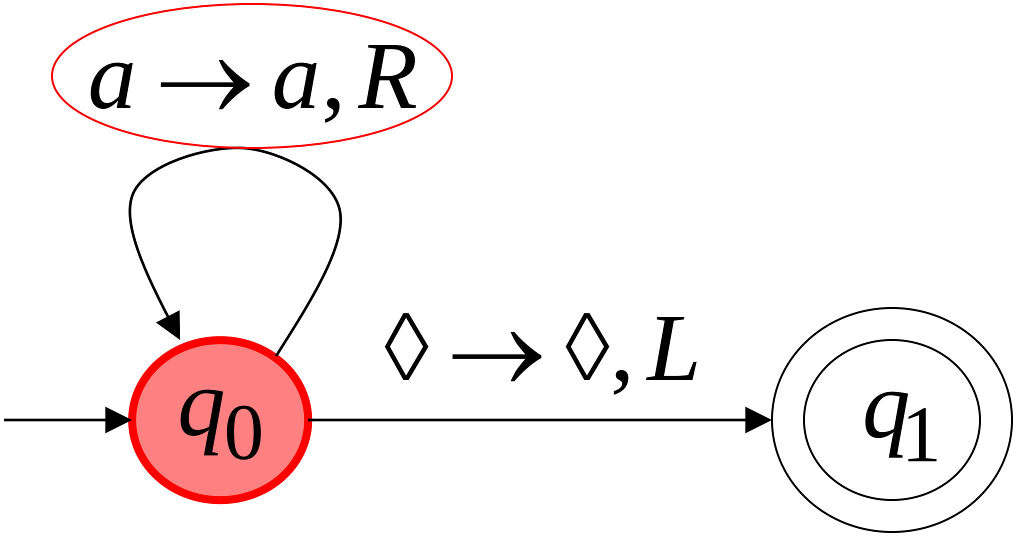


Halt & Accept

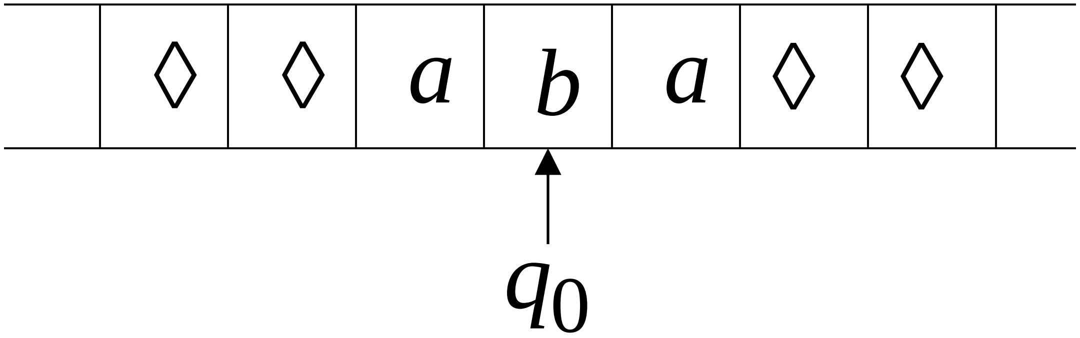
Rejection Example

Time 0



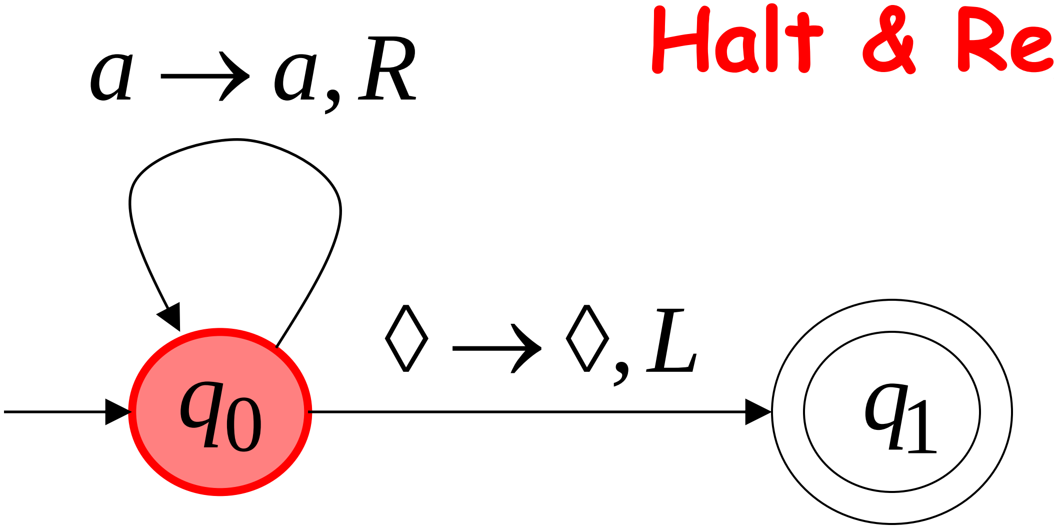


Time 1



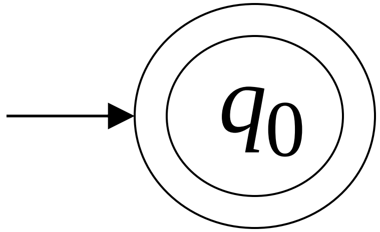
No possible Transition

Halt & Reject

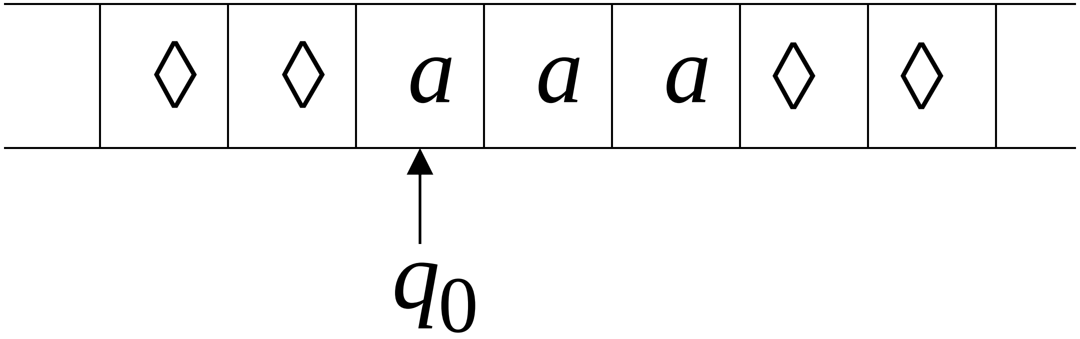


A simpler machine for same language but for input alphabet $\Sigma = \{a\}$

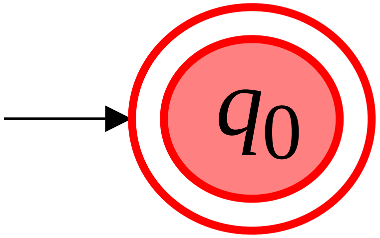
Accepts the language: a^*



Time 0



Halt & Accept

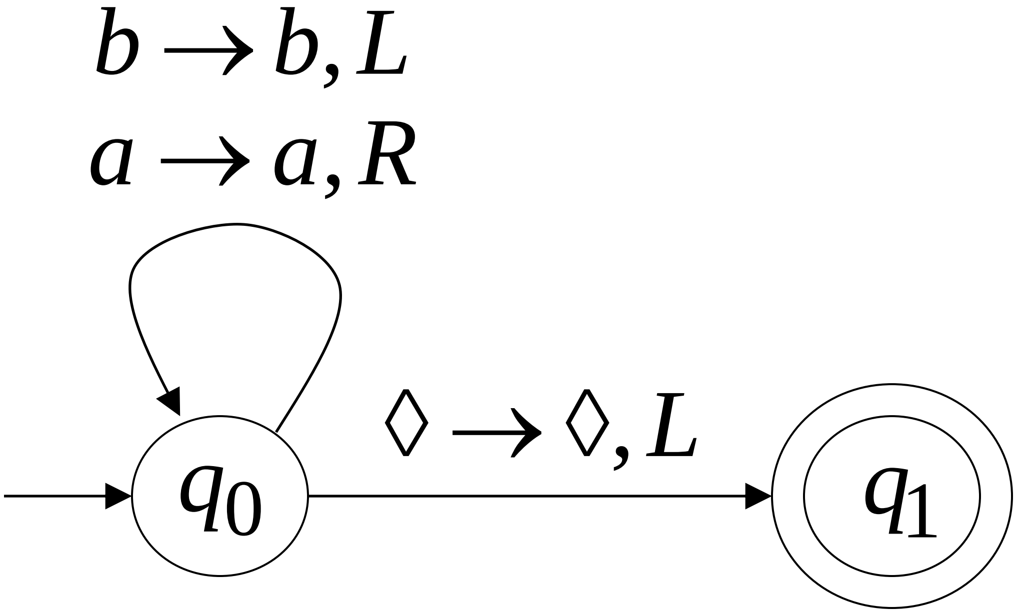


Not necessary to scan input

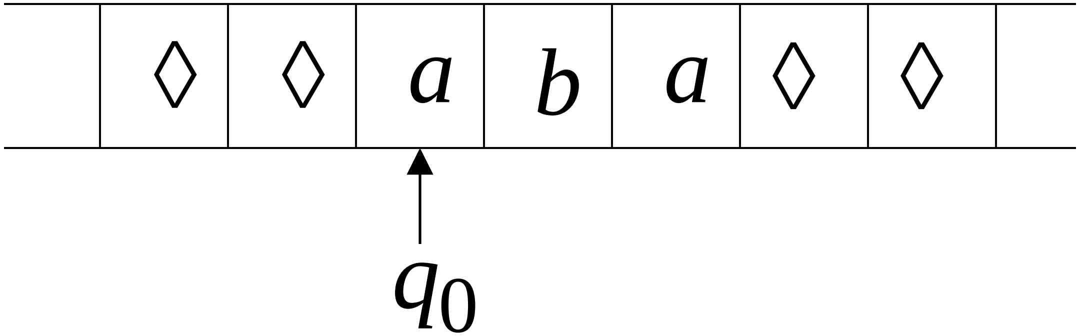
Infinite Loop Example

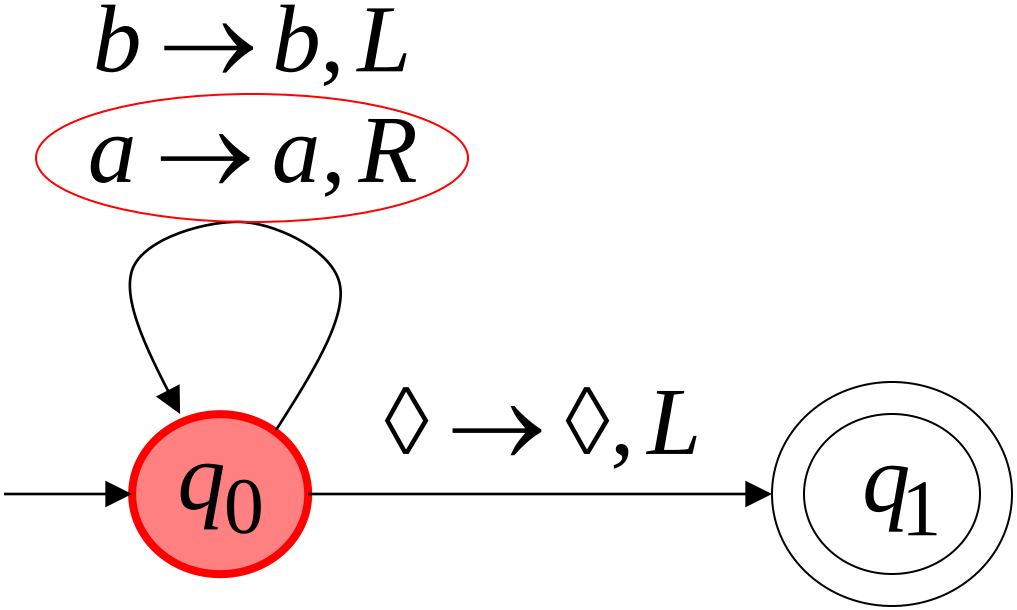
A Turing machine

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

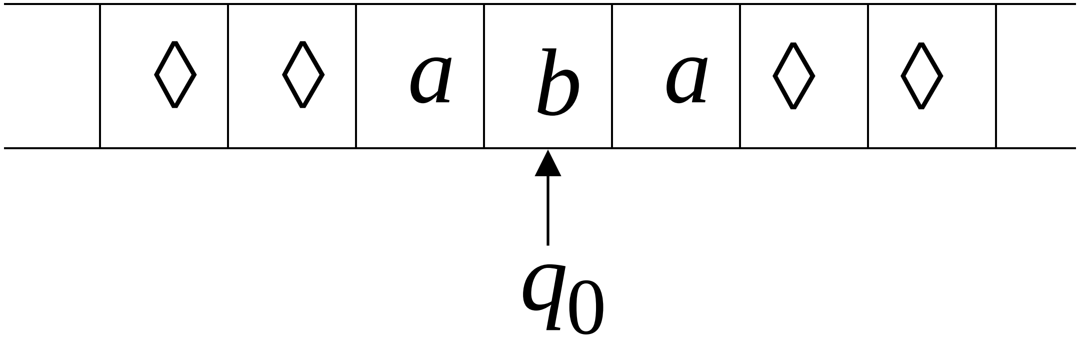


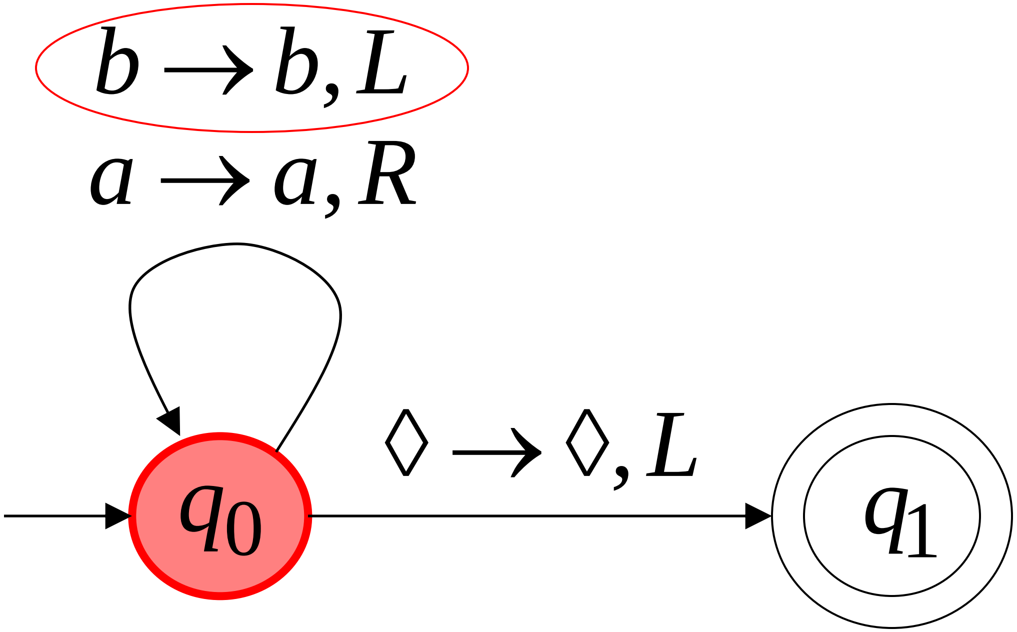
Time 0



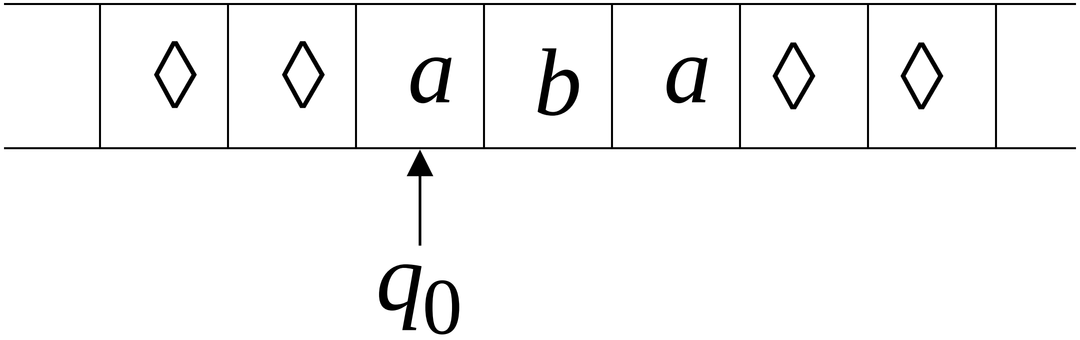


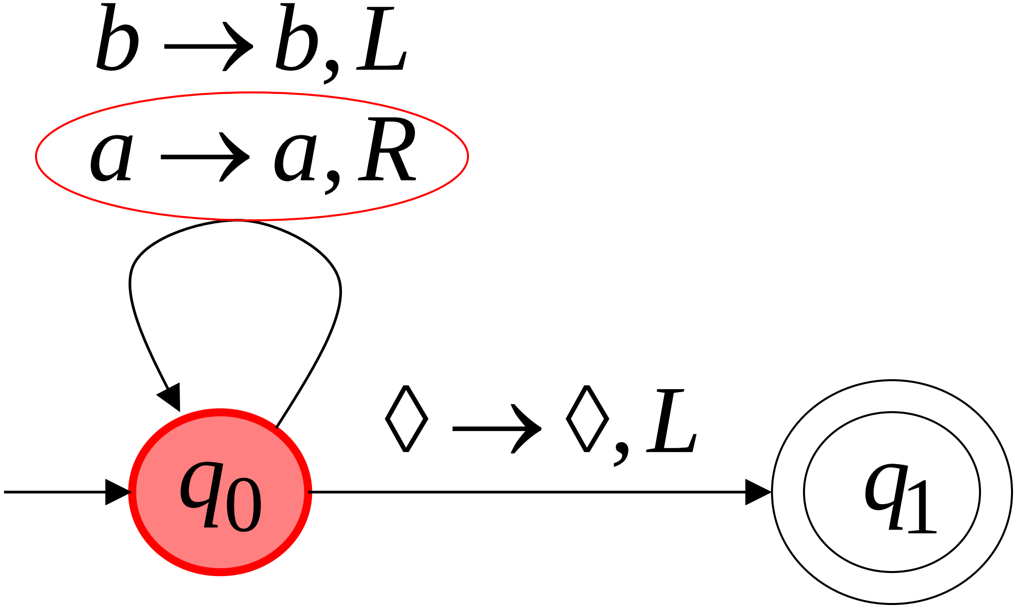
Time 1



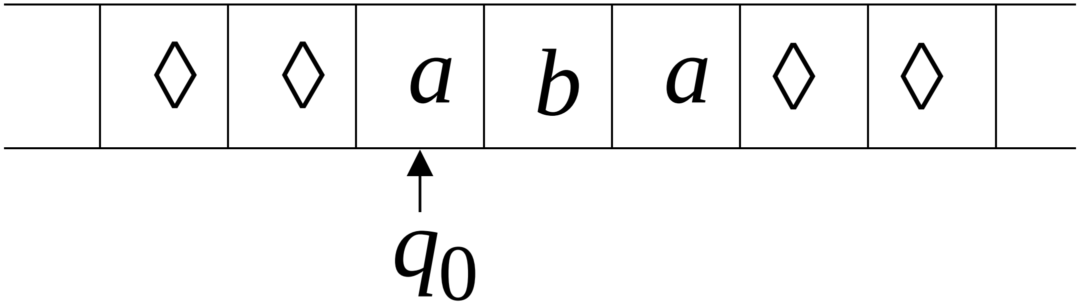


Time 2

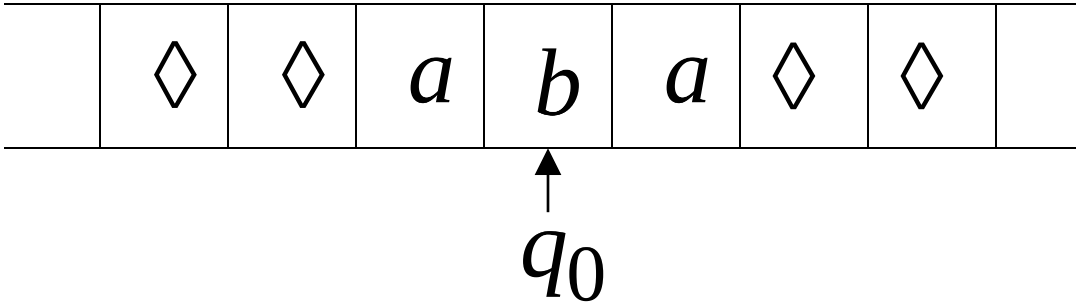

◊ ◊ a b a ◊ ◊
q_0


$b \rightarrow b, L$
$a \rightarrow a, R$
$\lozenge \rightarrow \lozenge, L$
q_0
q_1

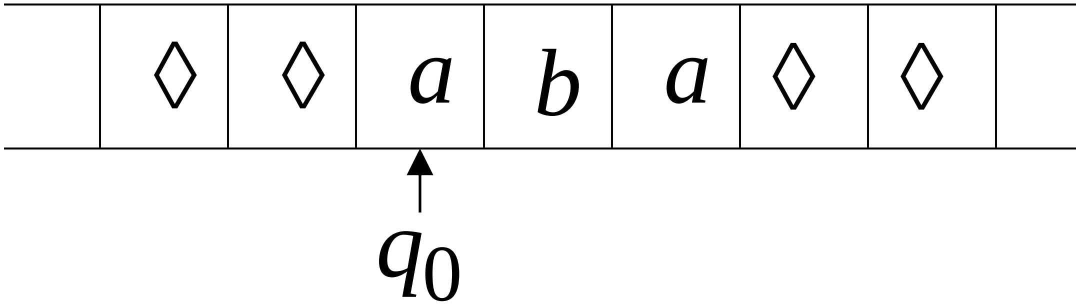
Time 2



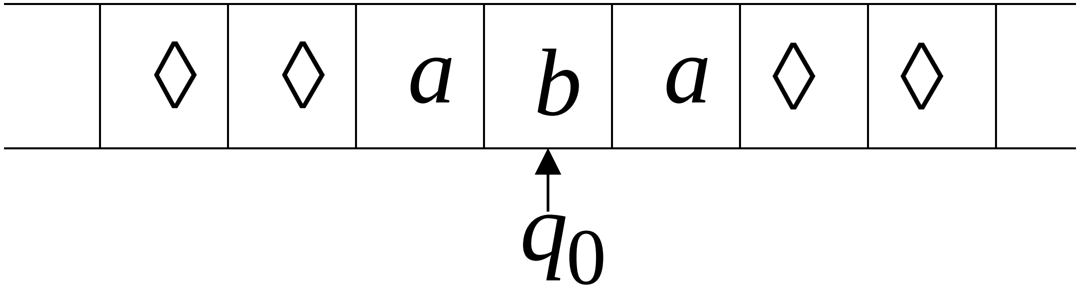
Time 3



Time 4



Time 5



Infinite loop

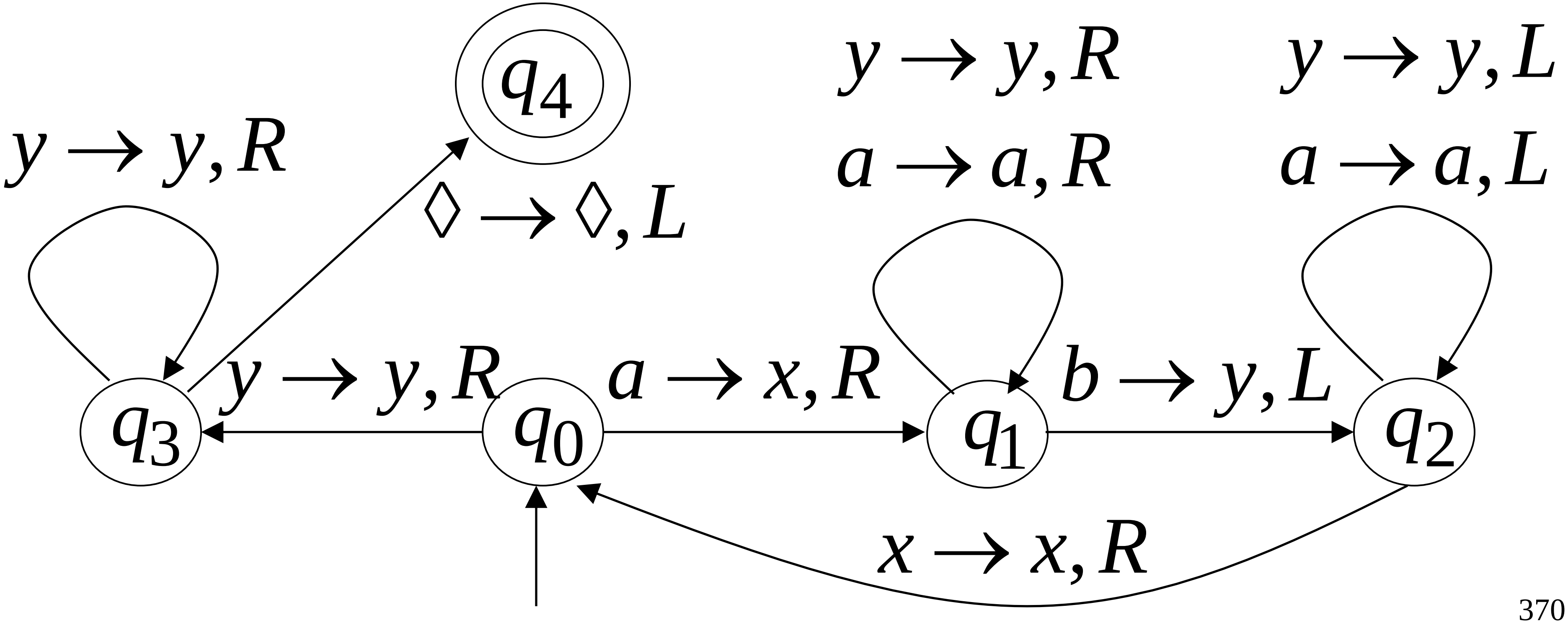
Because of the infinite loop:

- The accepting state cannot be reached
- The machine never halts
- The input string is rejected

Another Turing Machine Example

Turing machine for the language $\{a^n b^n\}$

$n \geq 1$



Basic Idea:

Match a's with b's:

Repeat:

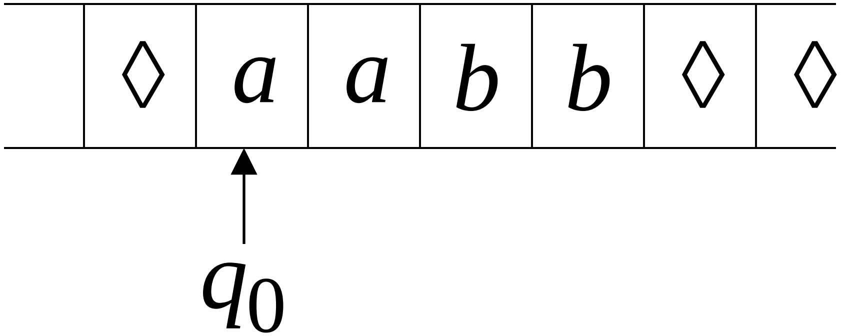
replace leftmost a with x

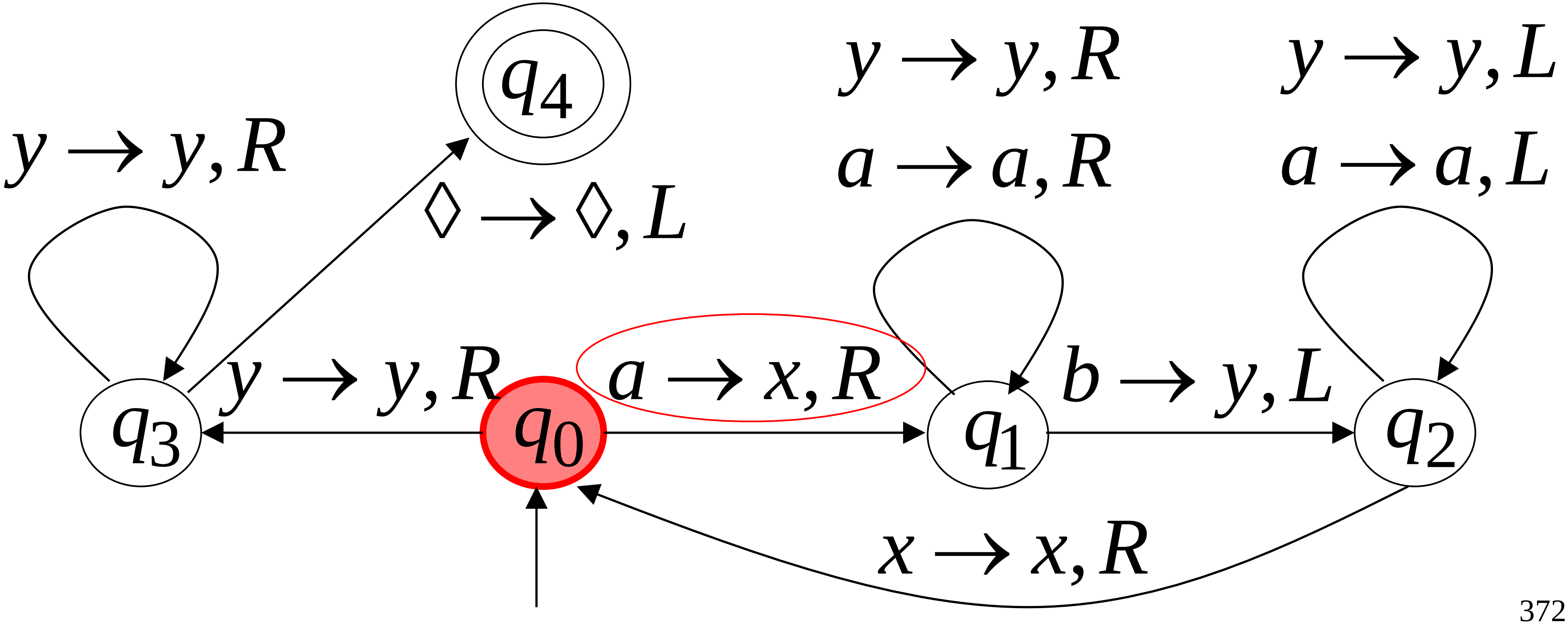
find leftmost b and replace it with y

Until there are no more a's or b's

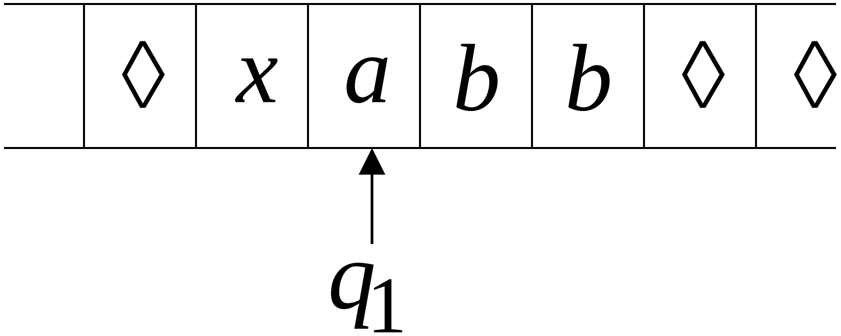
If there is a remaining a or b reject

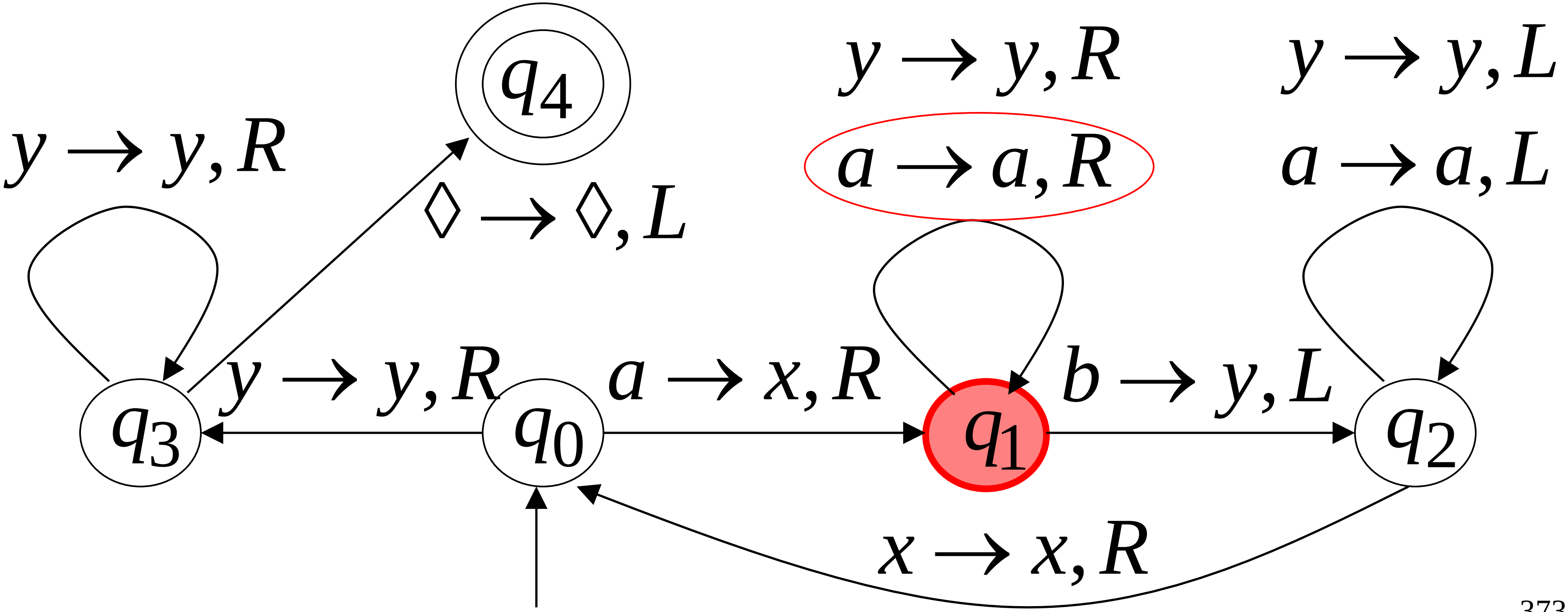
Time 0


◊
a
a
b
b
◊
◊
q_0

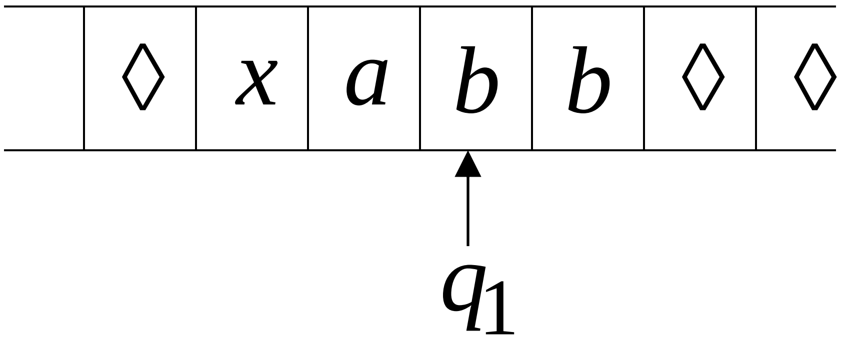

q_4
$y \to y, R$
$\Diamond \to \Diamond, L$
$y \to y, R$
$a \to a, R$
$y \to y, L$
$a \to a, L$
q_3
$y \to y, R$
q_0
$a \to x, R$
q_1
$b \to y, L$
q_2
$x \to x, R$

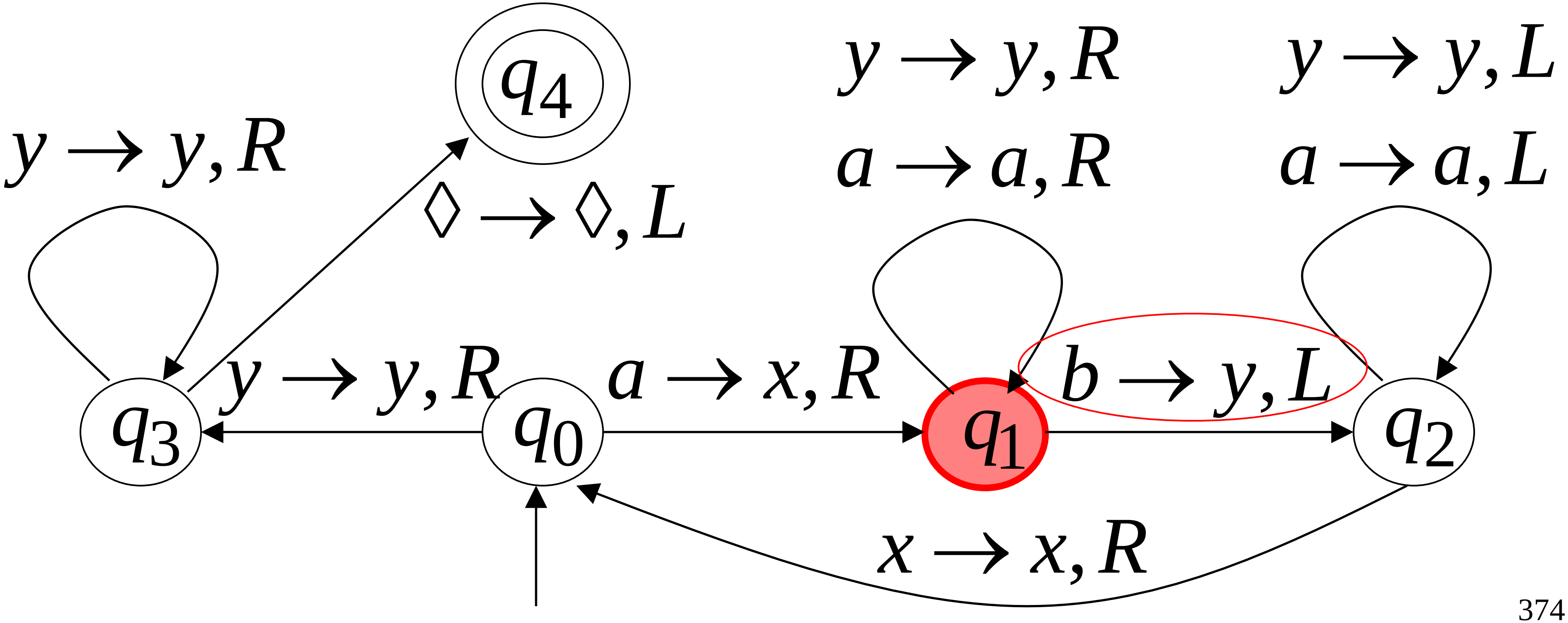
Time 1


◊
x
a
b
b
◊
◊
q_1

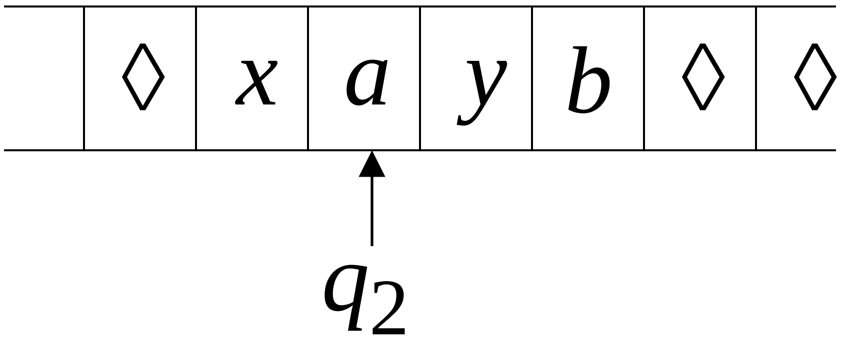

q_4
$\Diamond \rightarrow \Diamond, L$
$y \rightarrow y, R$
q_3
$y \rightarrow y, R$
q_0
$a \rightarrow x, R$
$y \rightarrow y, R$
$a \rightarrow a, R$
q_1
$b \rightarrow y, L$
$y \rightarrow y, L$
$a \rightarrow a, L$
q_2
$x \rightarrow x, R$

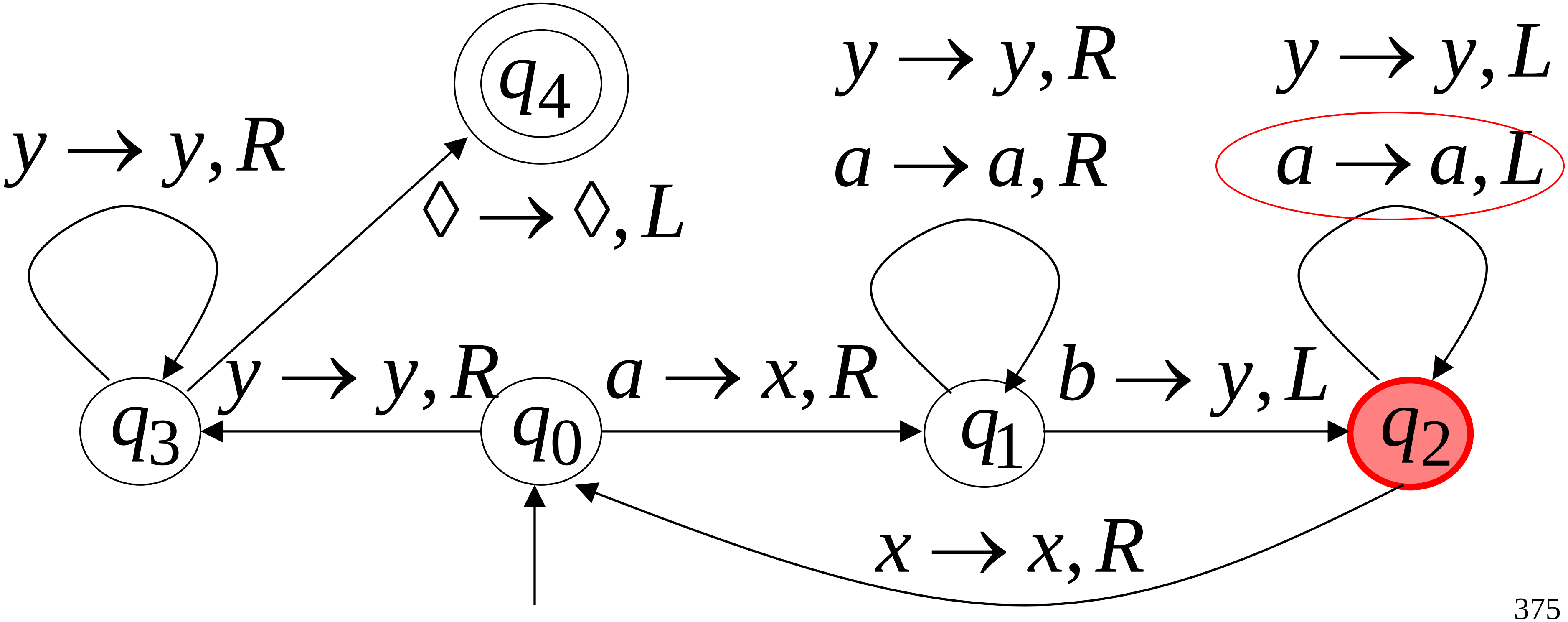
Time 2


$\Diamond$
x
a
b
b
$\Diamond$
$\Diamond$
q_1

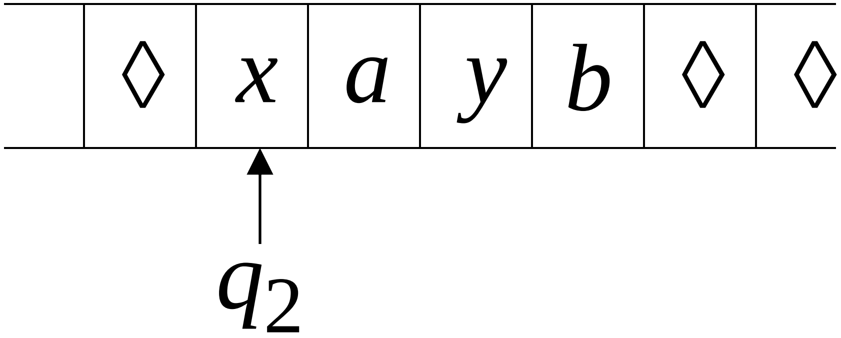

q_4
$y \rightarrow y, R$
$\Diamond \rightarrow \Diamond, L$
$y \rightarrow y, R$
$a \rightarrow a, R$
$y \rightarrow y, L$
$a \rightarrow a, L$
$y \rightarrow y, R$
$a \rightarrow x, R$
$b \rightarrow y, L$
q_3
q_0
q_1
q_2
$x \rightarrow x, R$

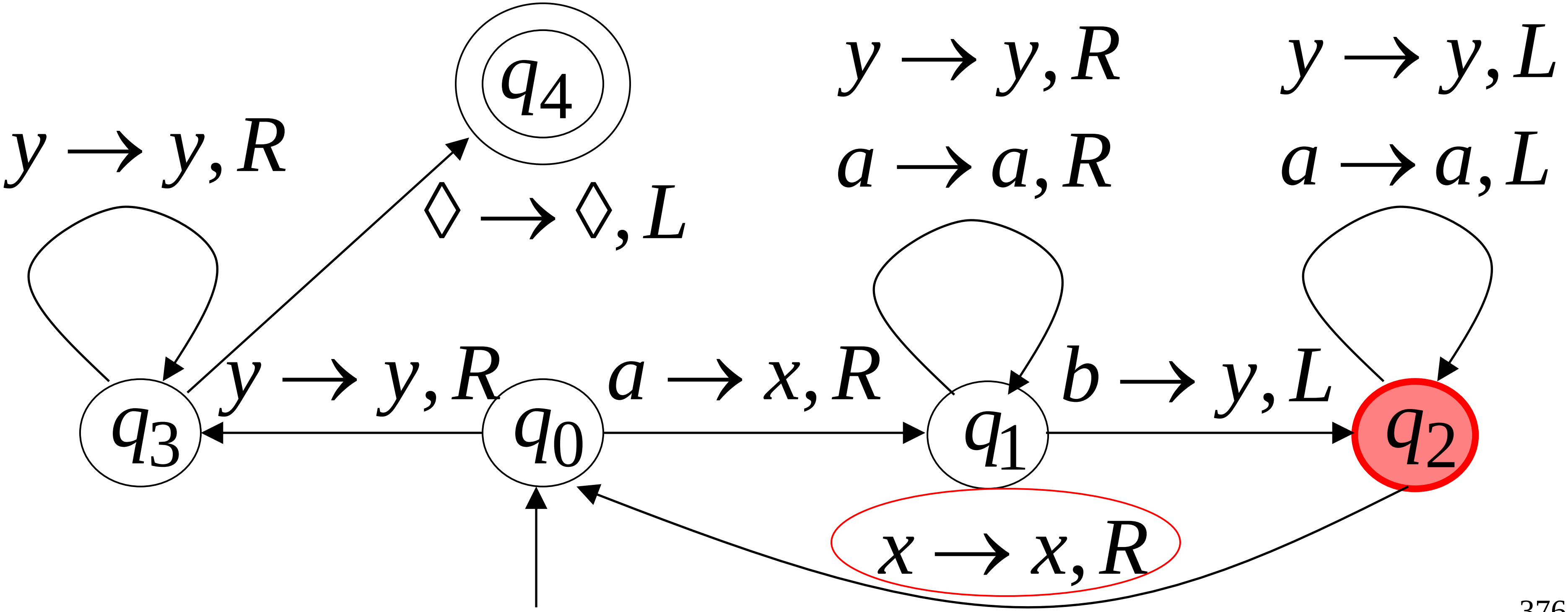
Time 3


◊
x
a
y
b
◊
◊
q_2

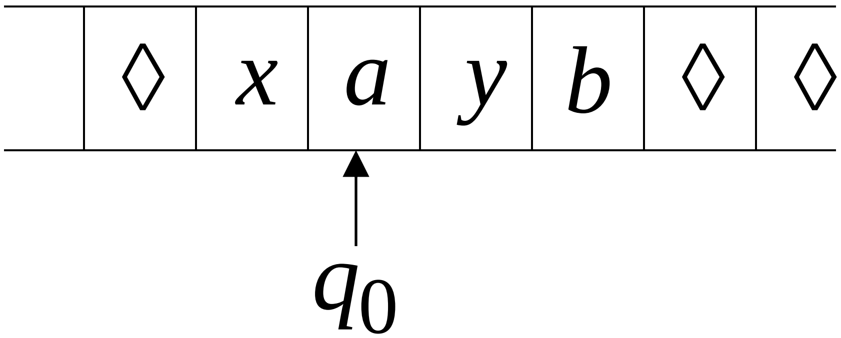

q_4
$y \rightarrow y, R$
$◊ \rightarrow ◊, L$
$y \rightarrow y, R$
$a \rightarrow a, R$
$y \rightarrow y, L$
$a \rightarrow a, L$
$y \rightarrow y, R$
$a \rightarrow x, R$
$b \rightarrow y, L$
q_3
q_0
q_1
q_2
$x \rightarrow x, R$

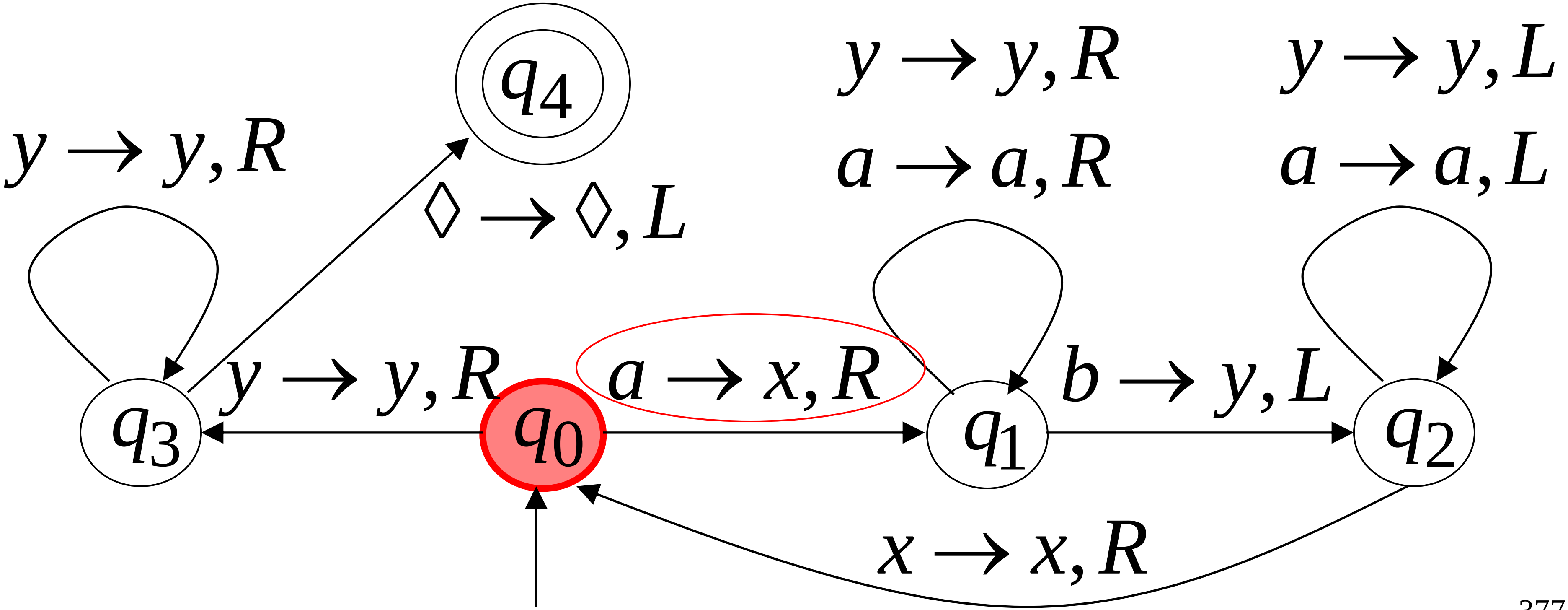
Time 4


◊
x
a
y
b
◊
◊
q_2

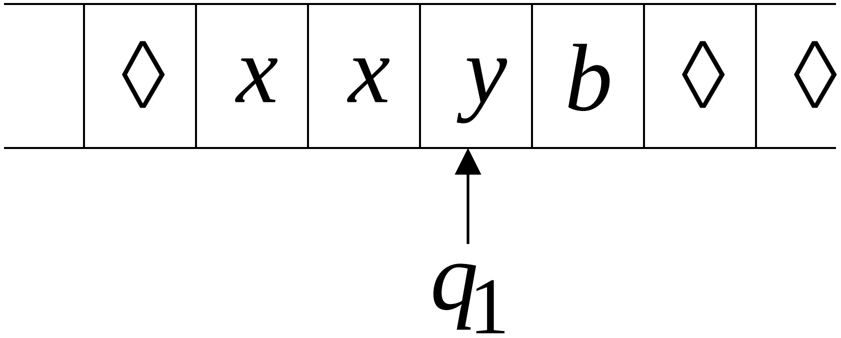

q_4
$\Diamond \to \Diamond, L$
$y \to y, R$
q_3
$y \to y, R$
q_0
$a \to x, R$
$y \to y, R$
$a \to a, R$
q_1
$b \to y, L$
$y \to y, L$
$a \to a, L$
q_2
$x \to x, R$

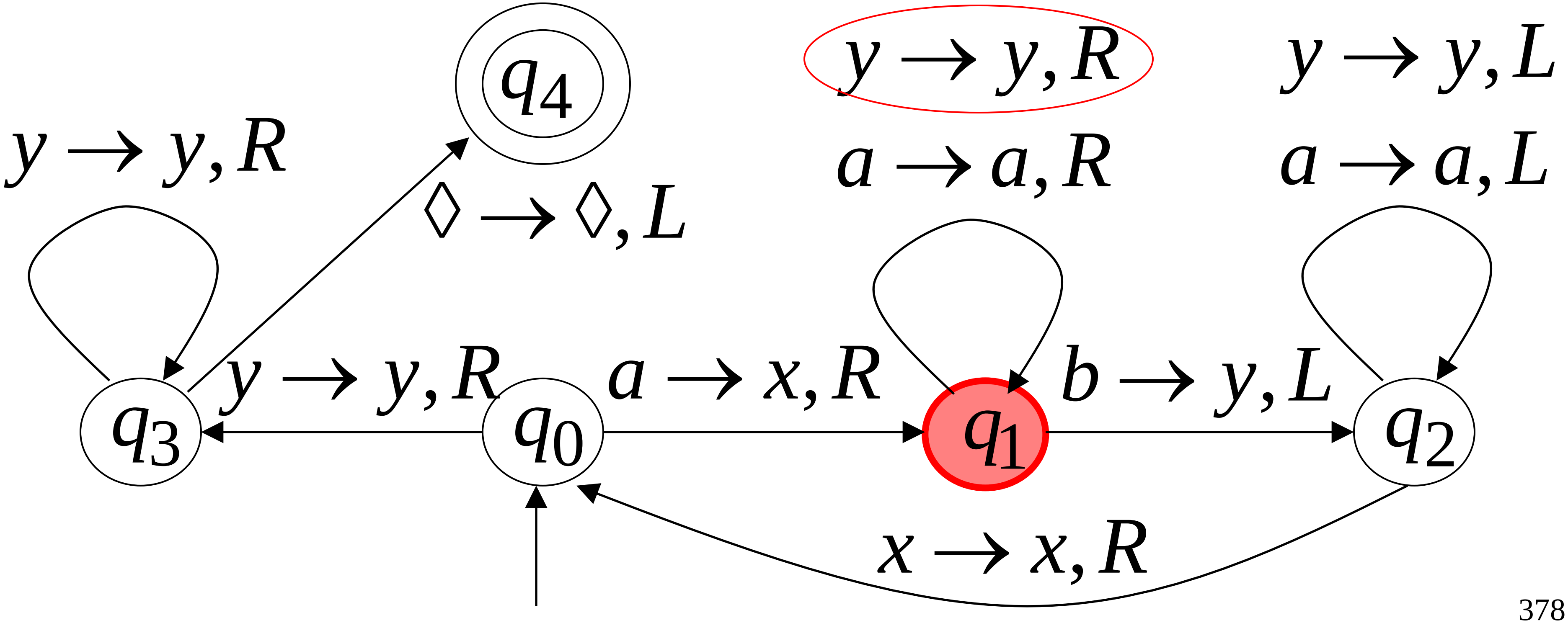
Time 5


◊
x
a
y
b
◊
◊
q_0

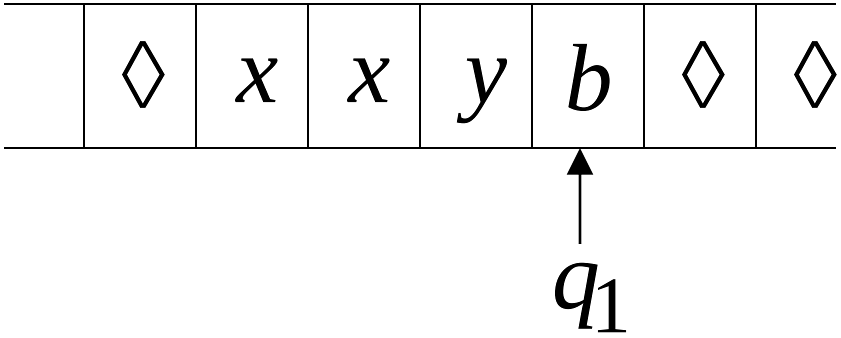

q_4
$y \to y, R$
$\Diamond \to \Diamond, L$
$y \to y, R$
$a \to a, R$
$y \to y, L$
$a \to a, L$
q_3
$y \to y, R$
q_0
$a \to x, R$
q_1
$b \to y, L$
q_2
$x \to x, R$

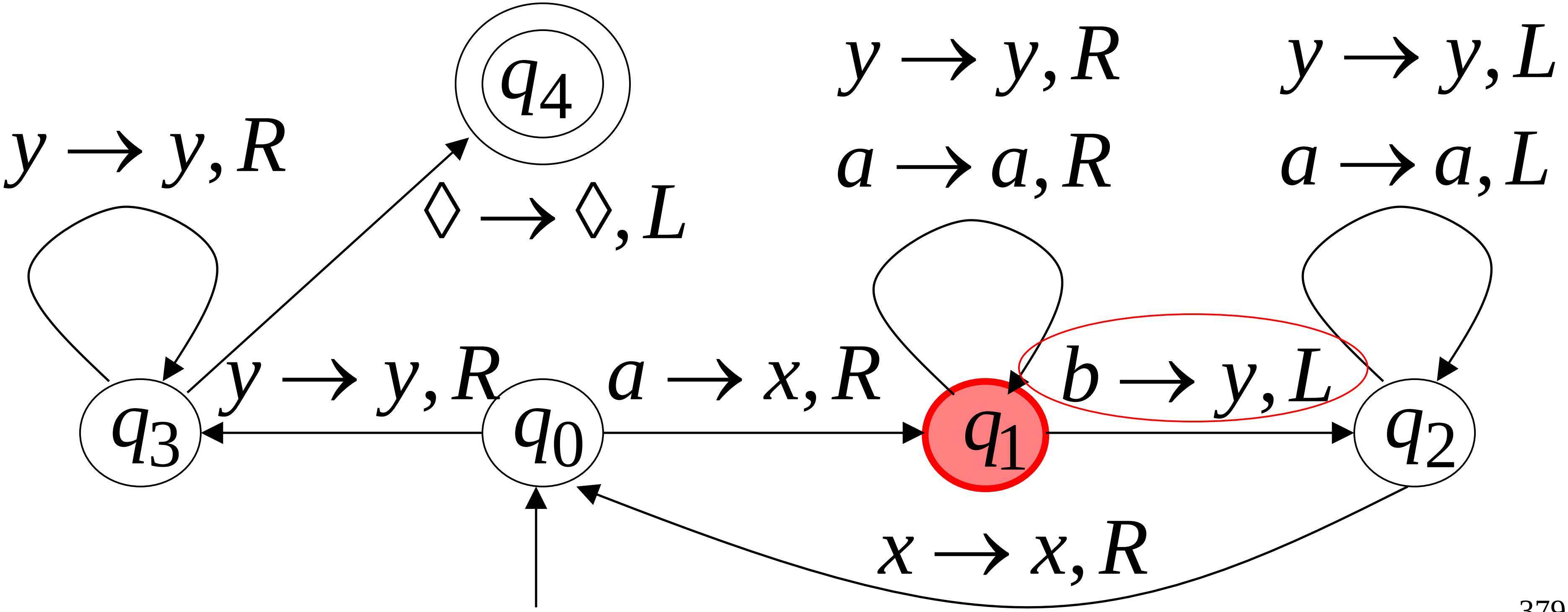
Time 6


◊
x
x
y
b
◊
◊
q_1

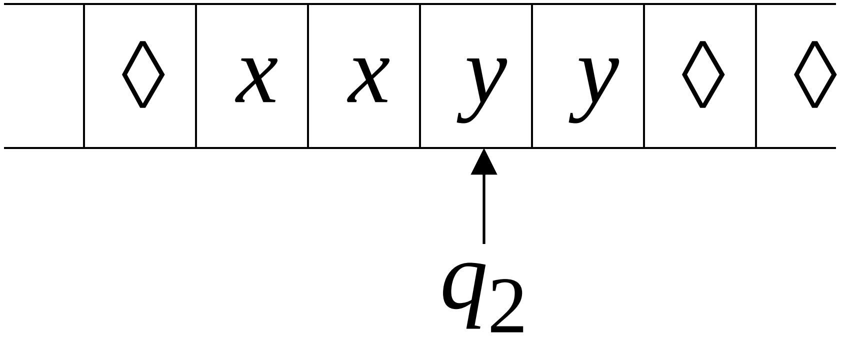

q_4
$y \rightarrow y, R$
◊ → ◊, L
$y \rightarrow y, R$
$a \rightarrow a, R$
$y \rightarrow y, L$
$a \rightarrow a, L$
q_3
$y \rightarrow y, R$
q_0
$a \rightarrow x, R$
q_1
$b \rightarrow y, L$
q_2
$x \rightarrow x, R$

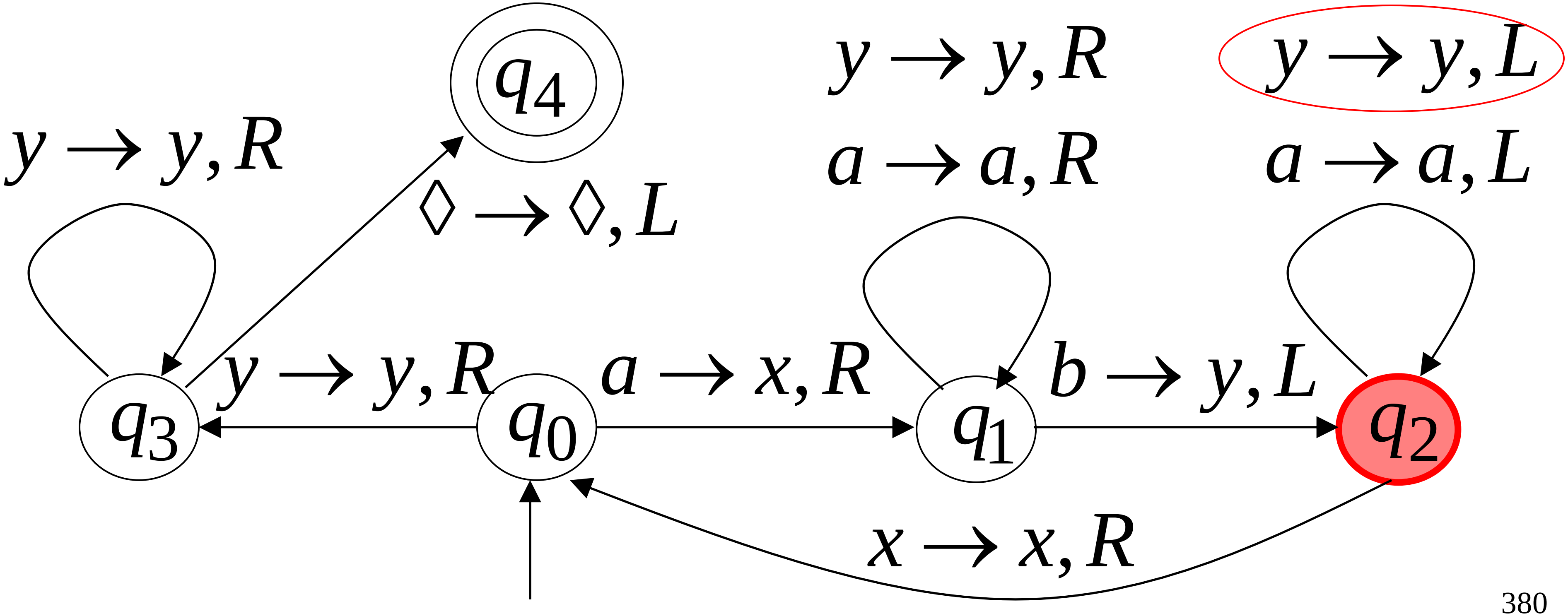
Time 7


◊
x
x
y
b
◊
◊
q_1

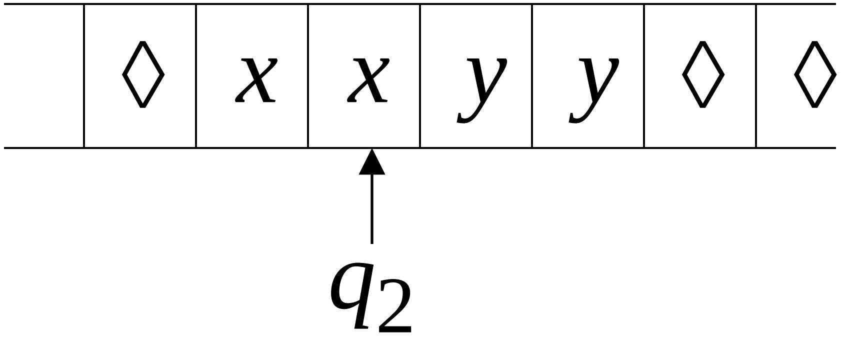

q_4
$y \rightarrow y, R$
$\lozenge \rightarrow \lozenge, L$
$y \rightarrow y, R$
$a \rightarrow a, R$
$y \rightarrow y, L$
$a \rightarrow a, L$
$y \rightarrow y, R$
$a \rightarrow x, R$
$b \rightarrow y, L$
q_3
q_0
q_1
q_2
$x \rightarrow x, R$

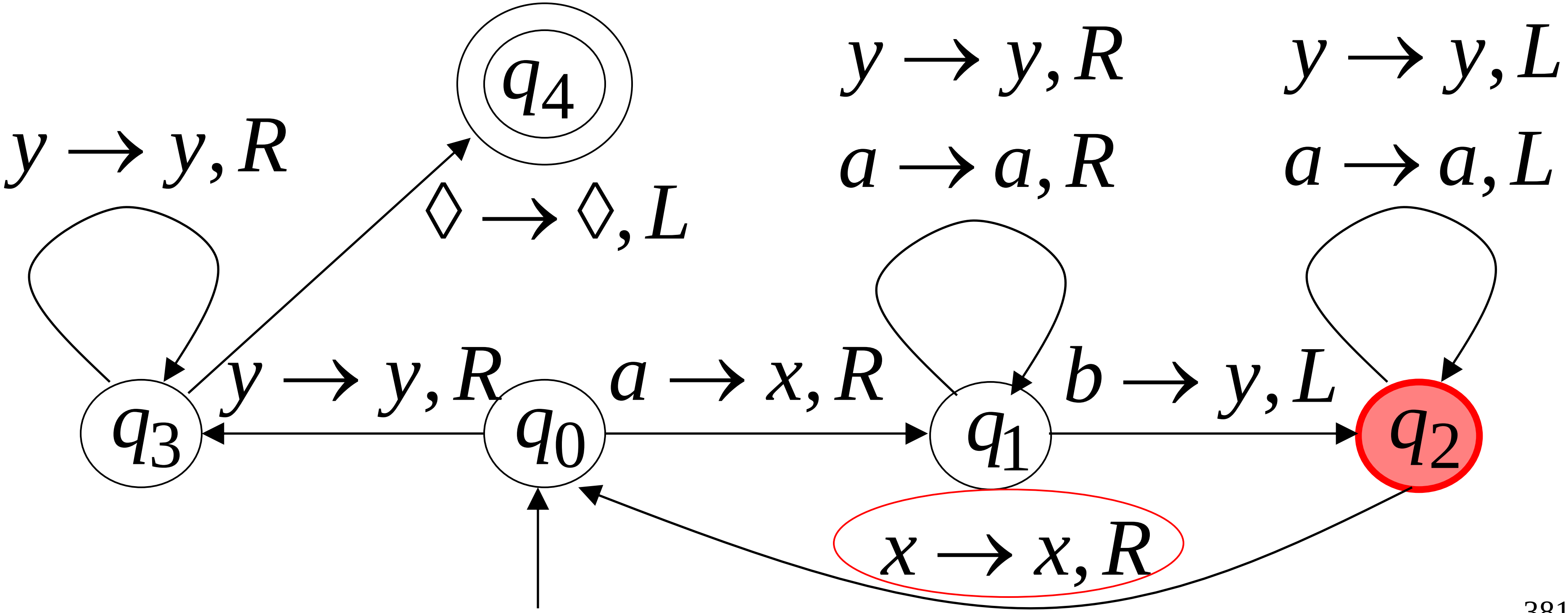
Time 8



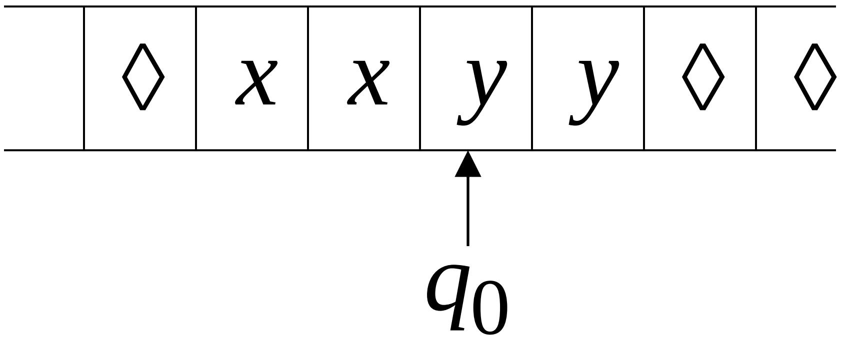


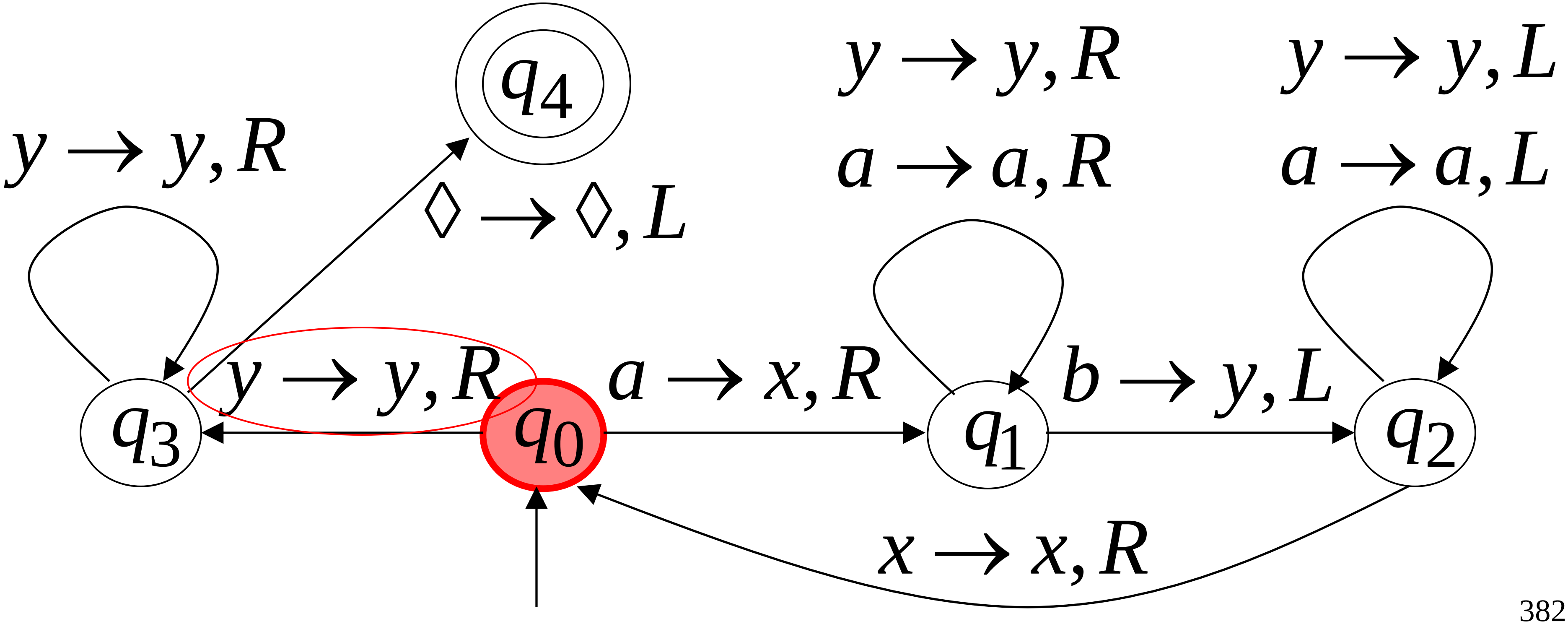
Time 9


◊
x
x
y
y
◊
◊
q_2

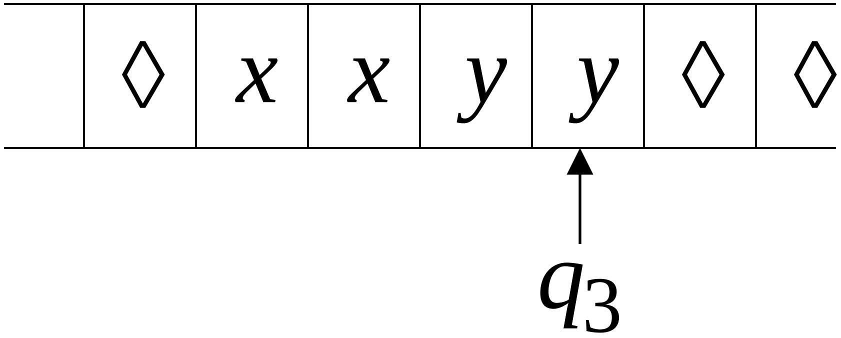

q_4
$y \rightarrow y, R$
q_3
$◊ \rightarrow ◊, L$
$y \rightarrow y, R$
q_0
$a \rightarrow x, R$
$y \rightarrow y, R$
$a \rightarrow a, R$
q_1
$b \rightarrow y, L$
$y \rightarrow y, L$
$a \rightarrow a, L$
q_2
$x \rightarrow x, R$

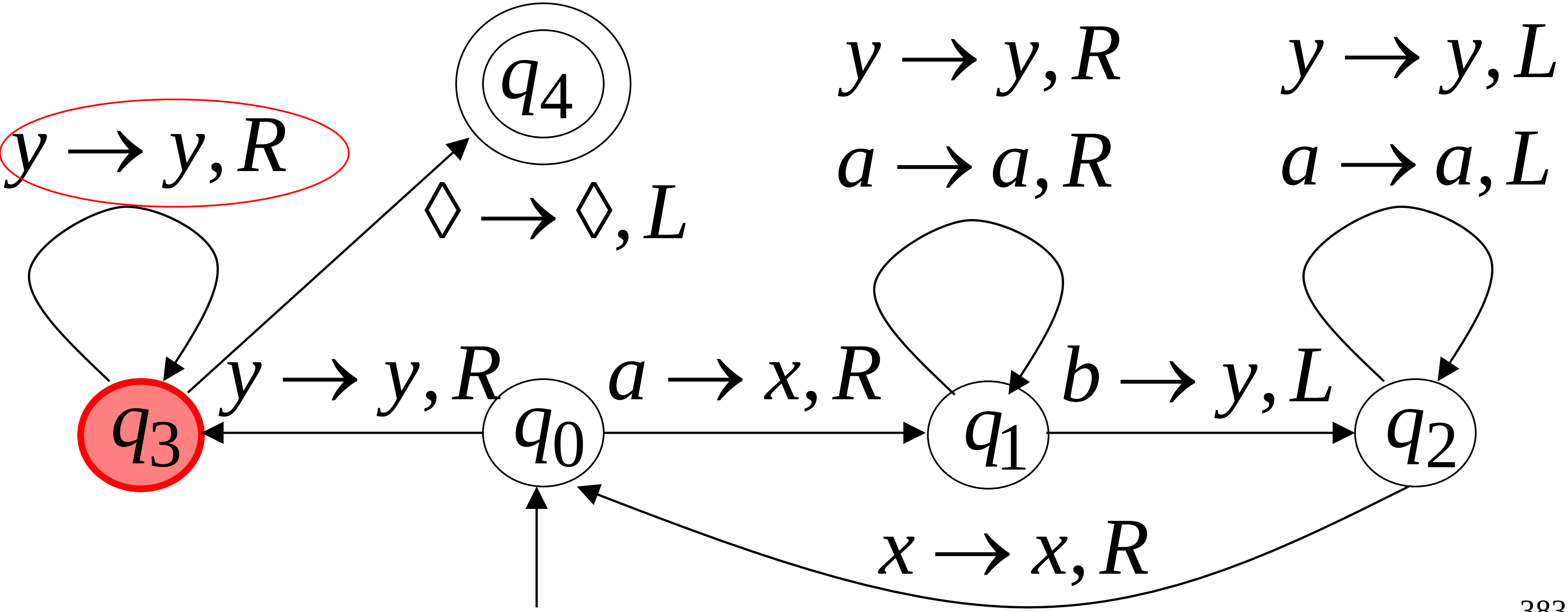
Time 10


◊
x
x
y
y
◊
◊
q_0

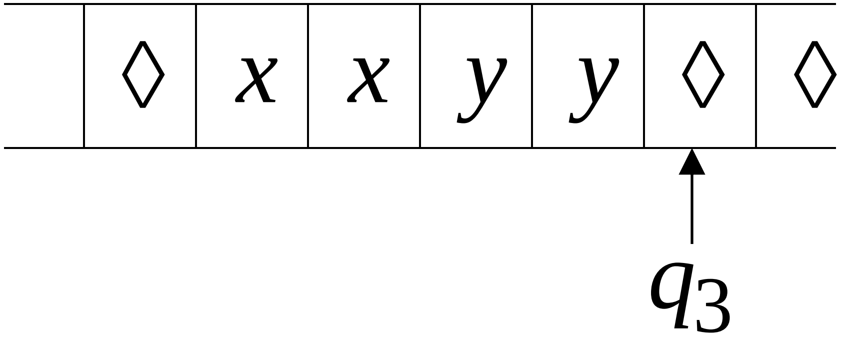

q_4
$y \rightarrow y, R$
◊ → ◊, L
$y \rightarrow y, R$
$a \rightarrow a, R$
$y \rightarrow y, L$
$a \rightarrow a, L$
q_3
$y \rightarrow y, R$
q_0
$a \rightarrow x, R$
q_1
$b \rightarrow y, L$
q_2
$x \rightarrow x, R$

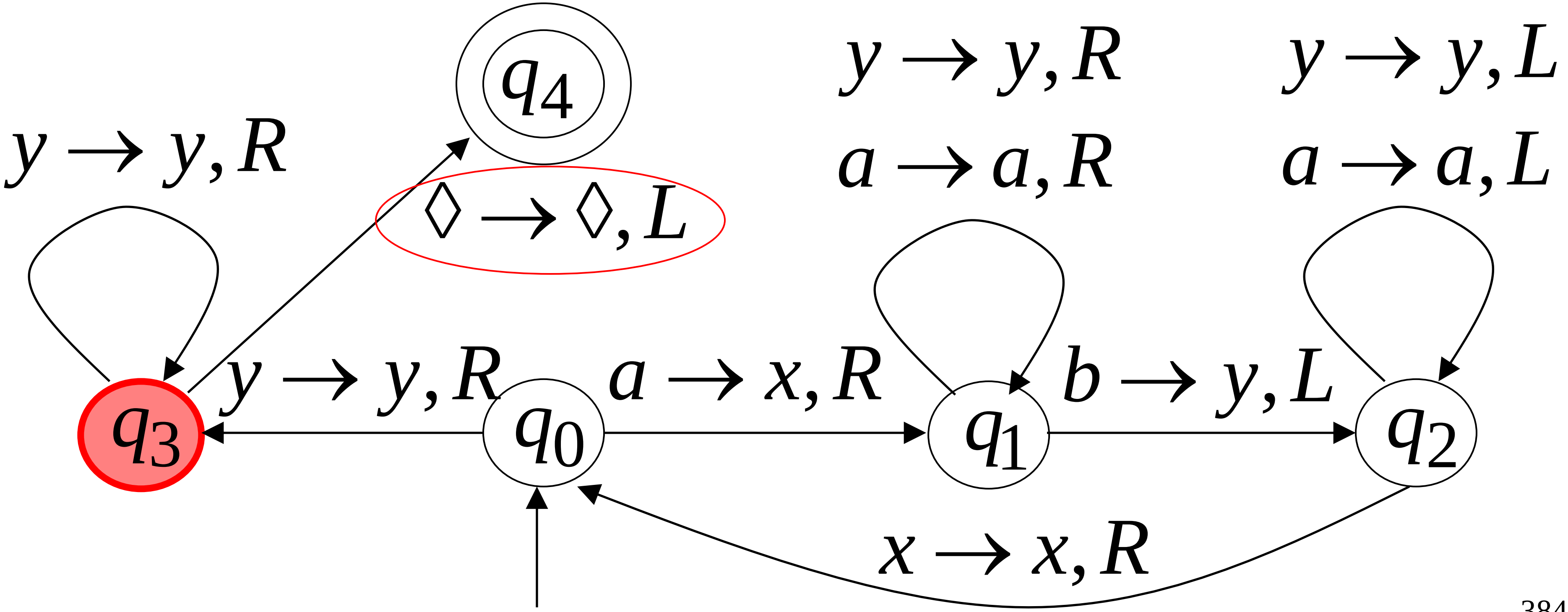
Time 11


◊
x
x
y
y
◊
◊
q_3

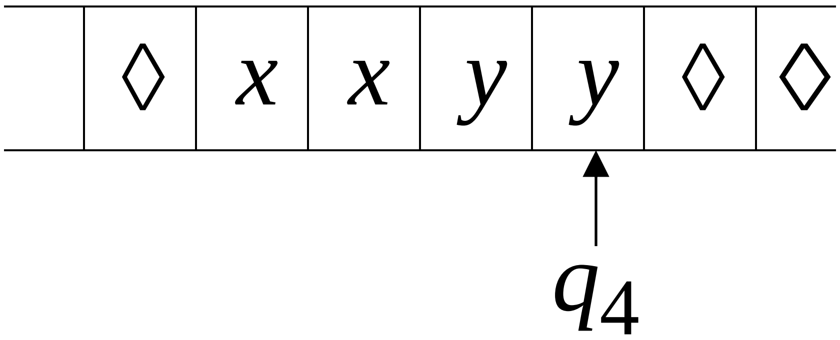

q_4
$y \rightarrow y, R$
$◊ \rightarrow ◊, L$
$y \rightarrow y, R$
$a \rightarrow a, R$
$y \rightarrow y, L$
$a \rightarrow a, L$
q_3
$y \rightarrow y, R$
q_0
$a \rightarrow x, R$
q_1
$b \rightarrow y, L$
q_2
$x \rightarrow x, R$

Time 12

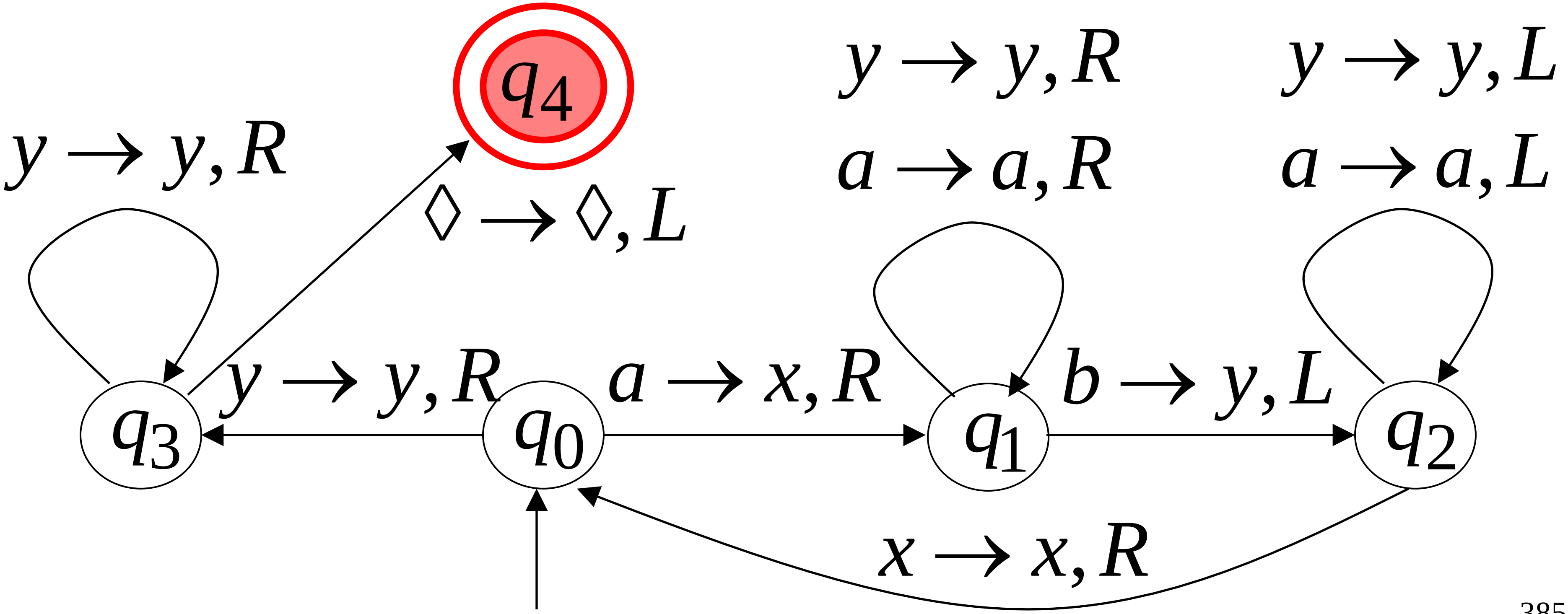

◊
x
x
y
y
◊
◊
q_3


q_4
$\Diamond \rightarrow \Diamond, L$
$y \rightarrow y, R$
$y \rightarrow y, R$
$a \rightarrow a, R$
$y \rightarrow y, L$
$a \rightarrow a, L$
$y \rightarrow y, R$
$a \rightarrow x, R$
$b \rightarrow y, L$
q_3
q_0
q_1
q_2
$x \rightarrow x, R$

Time 13



Halt & Accept



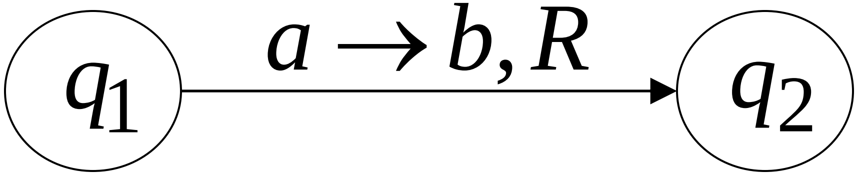
Observation:

If we modify the
machine for the language $\{a^n b^n\}$

we can easily construct
a machine for the language $\{a^n b^n c^n\}$

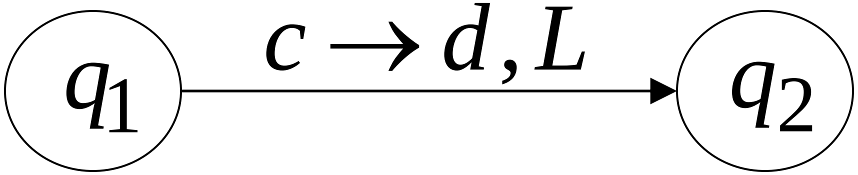
Formal Definitions for Turing Machines

Transition Function



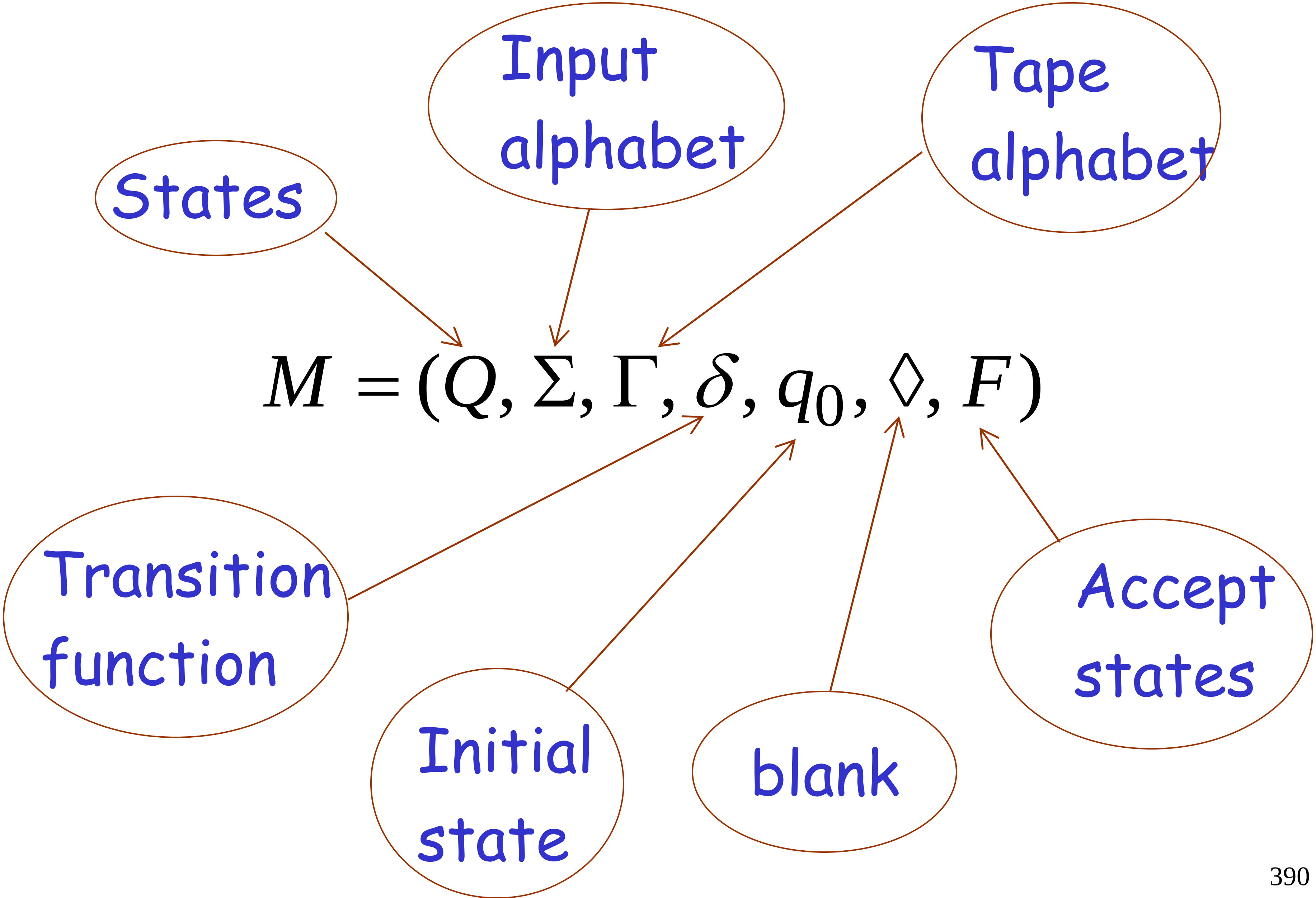
$$\delta(q_1, a) = (q_2, b, R)$$

Transition Function



$$\delta(q_1, c) = (q_2, d, L)$$

Turing Machine:



States

Input alphabet

Tape alphabet

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

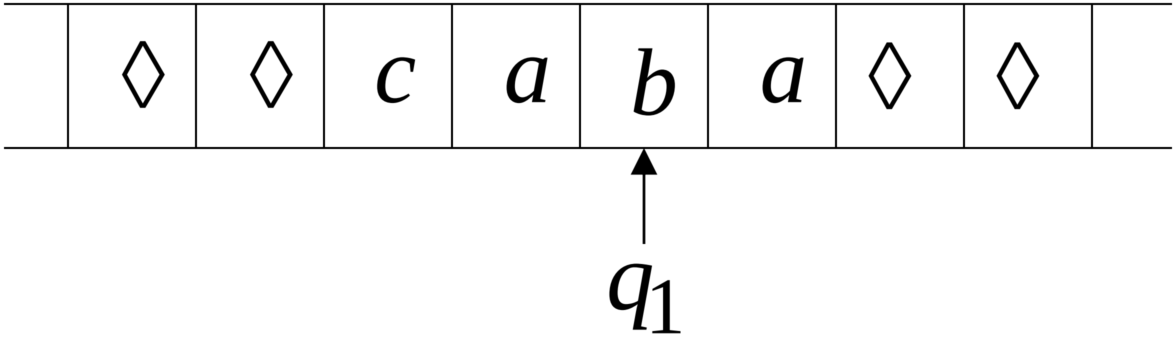
Transition function

Initial state

blank

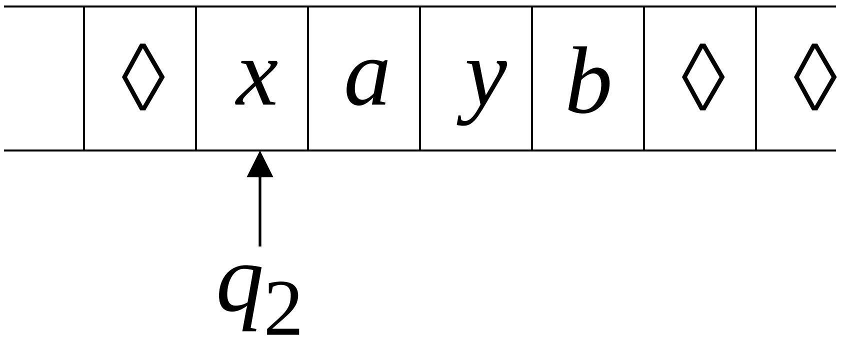
Accept states

Configuration

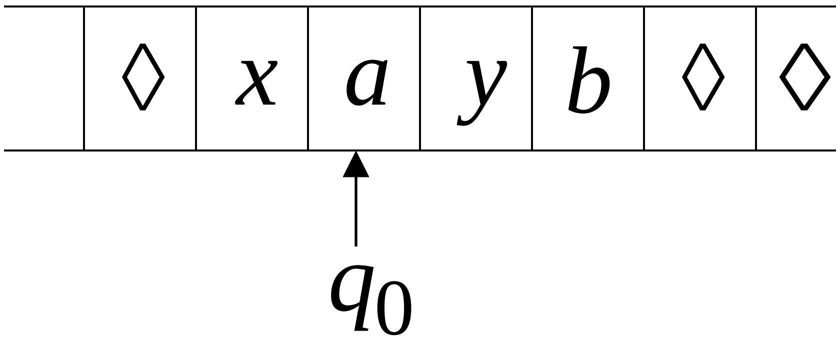


Instantaneous description: $ca\, q_1\, ba$

Time 4



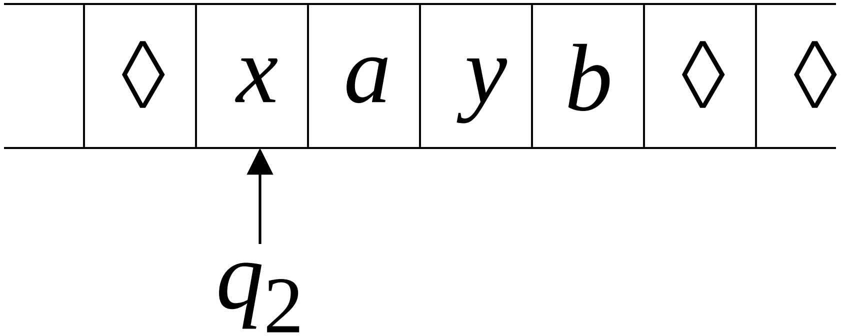
Time 5



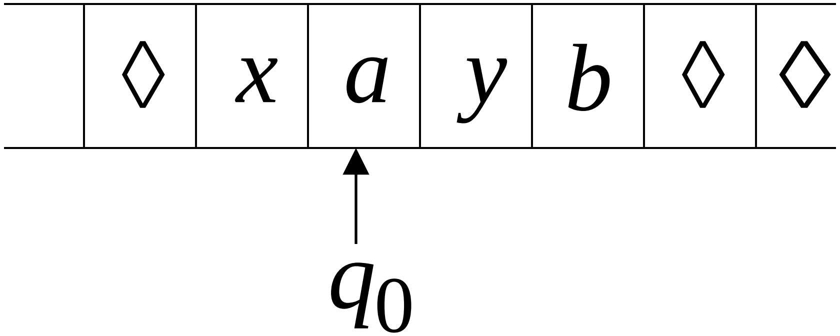
A Move: $$q_2\ xayb \succ x\ q_0\ ayb$$

(yields in one move)

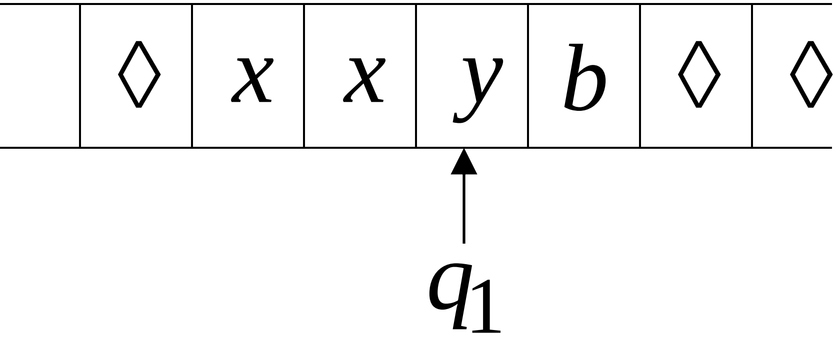
Time 4



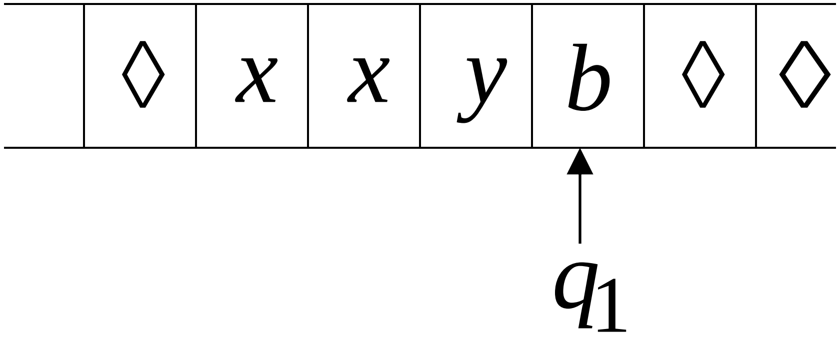
Time 5



Time 6



Time 7



A computation

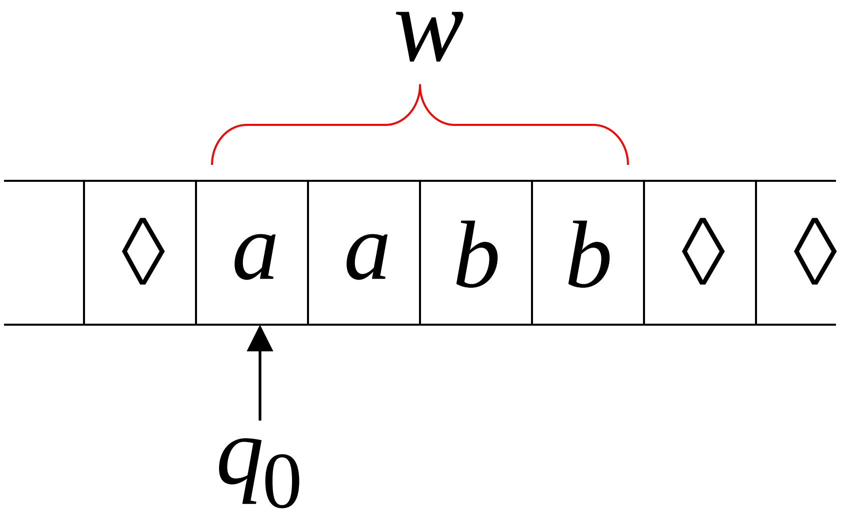
$$q_2\ xayb \succ x\ q_0\ ayb \succ xx\ q_1\ yb \succ xxy\ q_1\ b$$

$$q_2\ xayb \succ x\,q_0\ ayb \succ xx\,q_1\ yb \succ xxy\,q_1\,b$$

Equivalent notation: $q_2\ xayb \overset{*}{\succ} xxy\,q_1\,b$

Initial configuration: $q_0 \, w$

Input string



The Accepted Language

For any Turing Machine M

$$L(M)=\{w: \underset{\substack{\uparrow \\ \text{Initial state}}}{q_0}\, w \overset{*}{\succ} x_1\, \underset{\substack{\uparrow \\ \text{Accept state}}}{q_f}\, x_2\}$$

If a language L is accepted by a Turing machine M then we say that L is:

- Turing Recognizable

Other names used:

- Turing Acceptable
- Recursively Enumerable

Turing's Thesis

Turing's thesis (1930):

Any computation carried out
by mechanical means
can be performed by a Turing Machine

Algorithm:

An algorithm for a problem is a
Turing Machine which solves the problem

The algorithm describes the steps of
the mechanical means

This is easily translated to computation steps
of a Turing machine

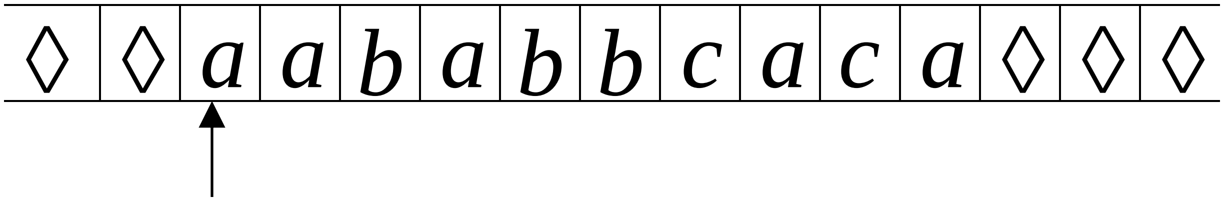
When we say: There exists an algorithm

We mean: There exists a Turing Machine that executes the algorithm

Variations of the Turing Machine

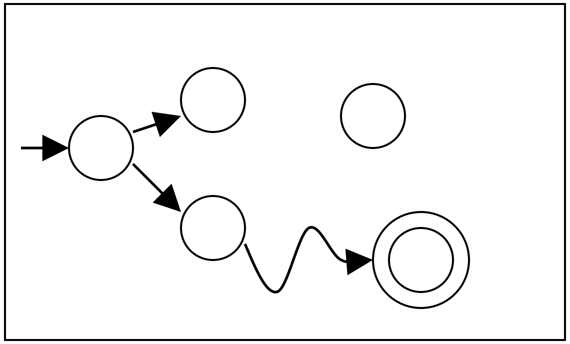
The Standard Model

Infinite Tape



Read-Write Head (Left or Right)

Control Unit



Deterministic

Variations of the Standard Model

Turing machines with:

- Stay-Option
- Semi-Infinite Tape
- Multitape
- Multidimensional
- Nondeterministic

Different Turing Machine **Classes**

Same Power of two machine classes:

both classes accept the
same set of languages

We will prove:

each new class has the same power
with Standard Turing Machine

(accept Turing-Recognizable Languages)

Same Power of two classes means:

for any machine M_1 of first class

there is a machine M_2 of second class

such that: $L(M_1) = L(M_2)$

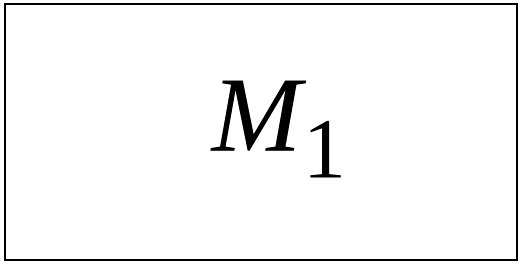
and vice-versa

Simulation: A technique to prove same power.

Simulate the machine of one class with a machine of the other class

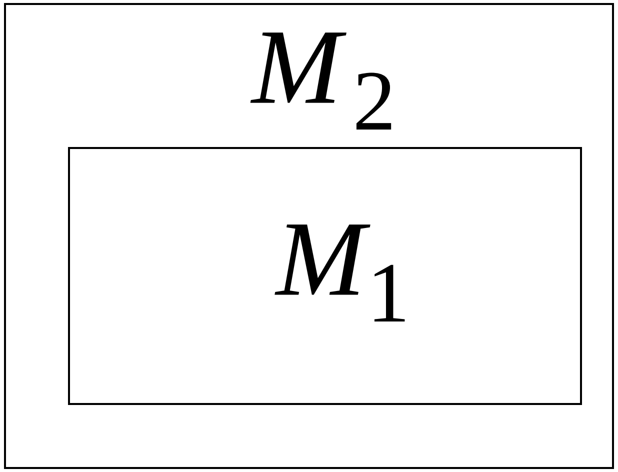
First Class

Original Machine



Second Class

Simulation Machine



simulates M_1

Configurations in the Original Machine M_1 have corresponding configurations in the Simulation Machine M_2

$$\begin{tikzcd}
 & & & & M_1 & & & \\
\text{Original Machine:} & d_0 \arrow[d, leftrightarrow] & \succ & d_1 \arrow[d, leftrightarrow] & \succ & \cdots & \succ & d_n \arrow[d, leftrightarrow] \\
\text{Simulation Machine:} & d'_0 & \overset{*}{\succ} & d'_1 & \overset{*}{\succ} & \cdots & \overset{*}{\succ} & d'_n \\
 & & & & M_2 & & &
\end{tikzcd}$$

Accepting Configuration

Original Machine: d_f

$$\updownarrow$$

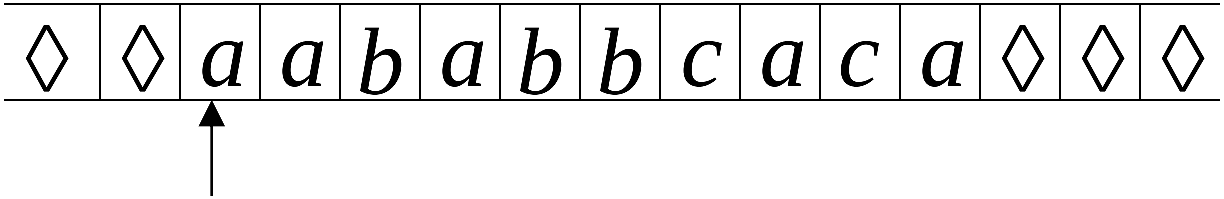
Simulation Machine: d'_f

the Simulation Machine
and the Original Machine
accept the same strings

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

Turing Machines with Stay-Option

The head can stay in the same position



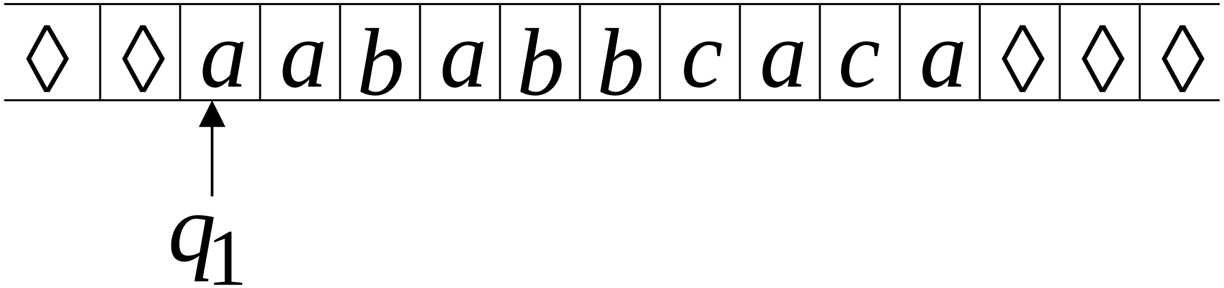
◊	◊	a	a	b	a	b	b	c	a	c	a	◊	◊	◊

Left, Right, Stay

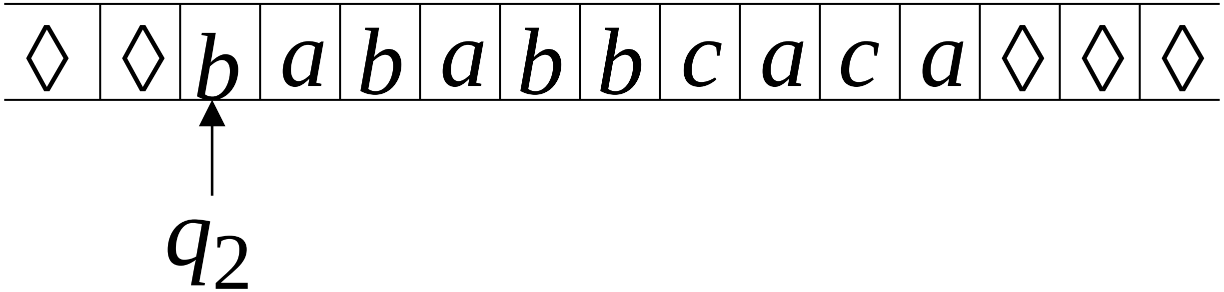
L,R,S: possible head moves

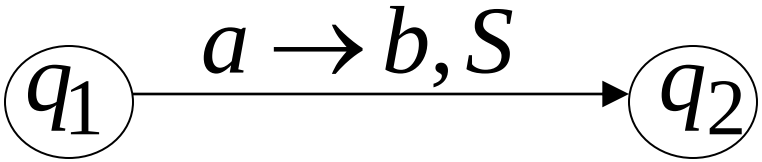
Example:

Time 1



Time 2





Theorem: Stay-Option machines
have the same power with
Standard Turing machines

Proof: 1. Stay-Option Machines
simulate Standard Turing machines

2. Standard Turing machines
simulate Stay-Option machines

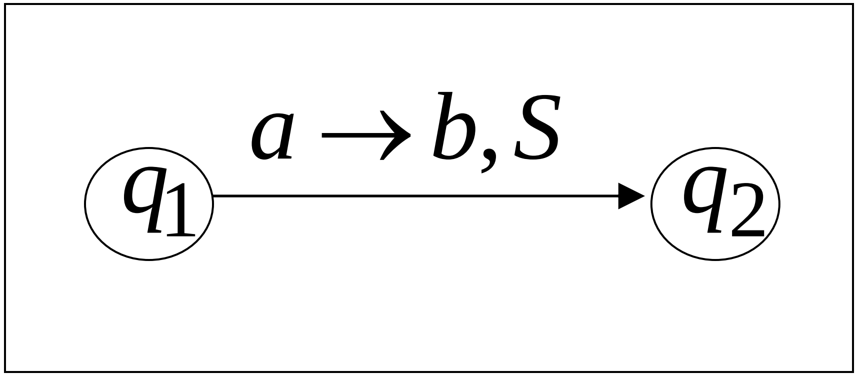
1. Stay-Option Machines simulate Standard Turing machines

Trivial: any standard Turing machine
is also a Stay-Option machine

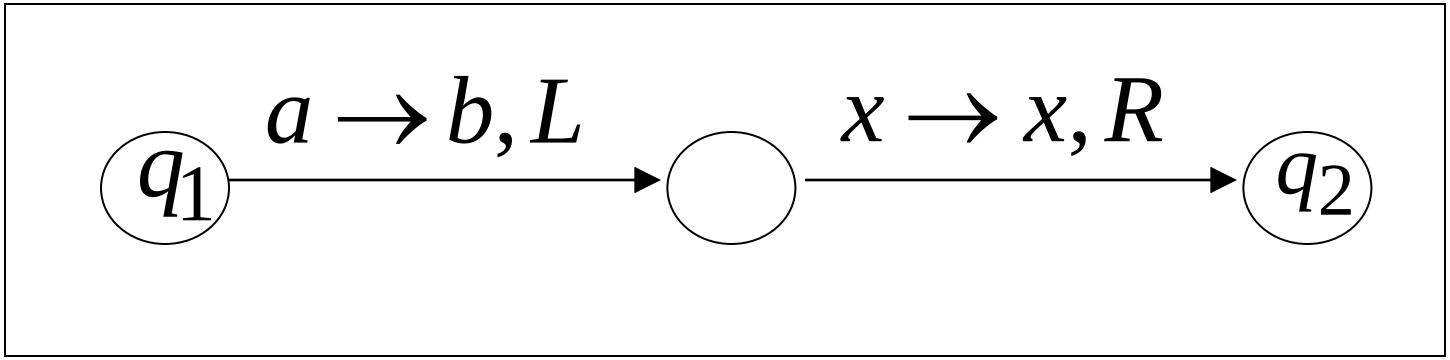
2. Standard Turing machines simulate Stay-Option machines

We need to simulate the stay head option with two head moves, one left and one right

Stay-Option Machine



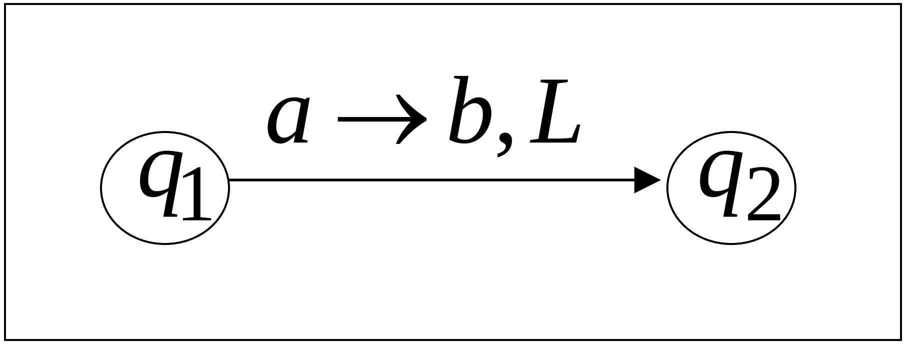
Simulation in Standard Machine



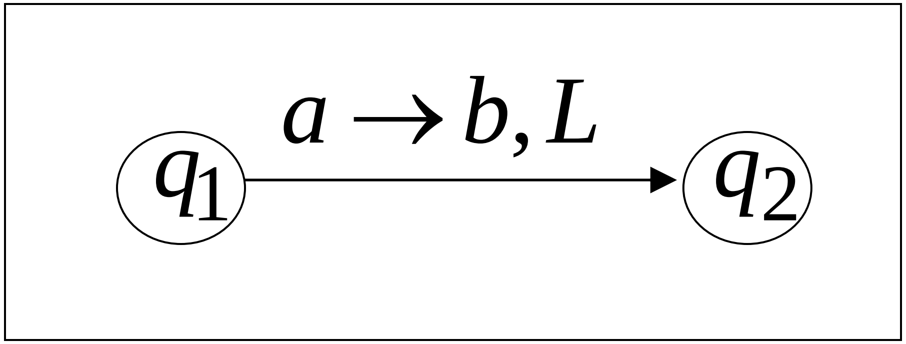
For every possible tape symbol x

For other transitions nothing changes

Stay-Option Machine



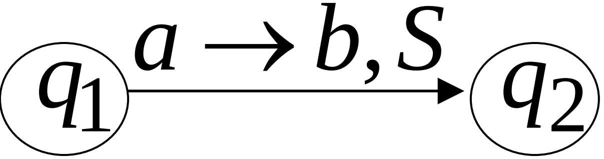
Simulation in Standard Machine

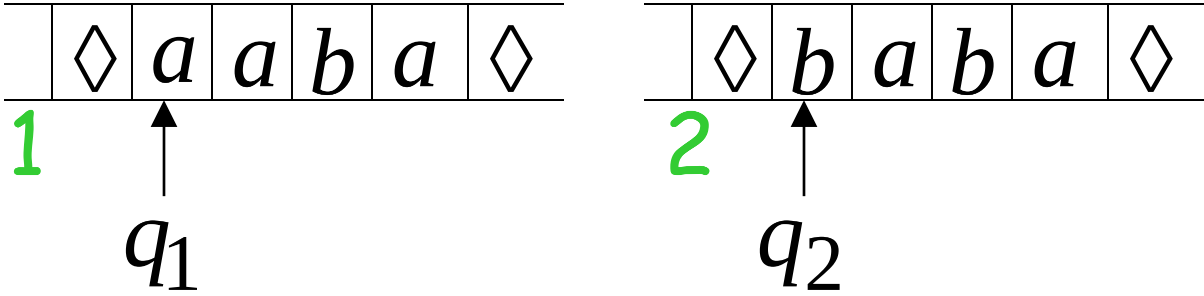


Similar for Right moves

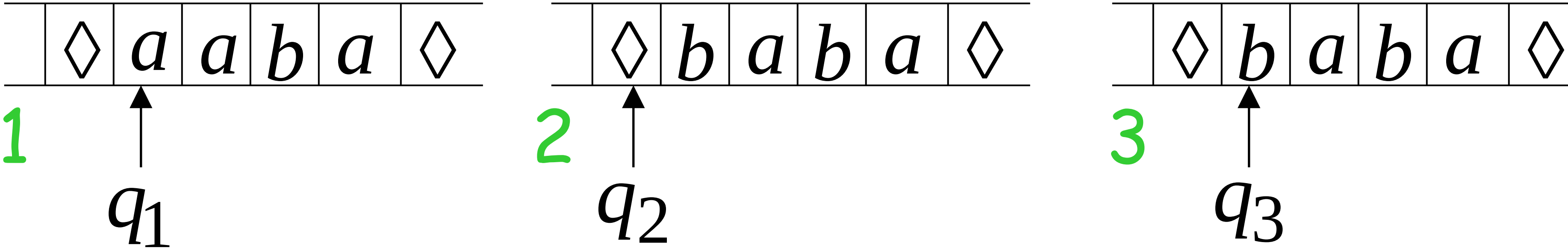
example of simulation

Stay-Option Machine:





Simulation in Standard Machine:

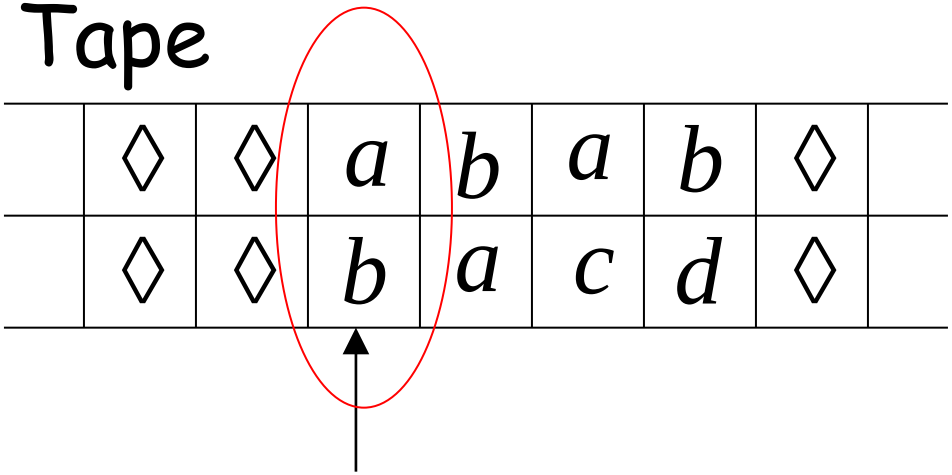


END OF PROOF

A useful trick: Multiple Track Tape

helps for more complicated simulations

One Tape

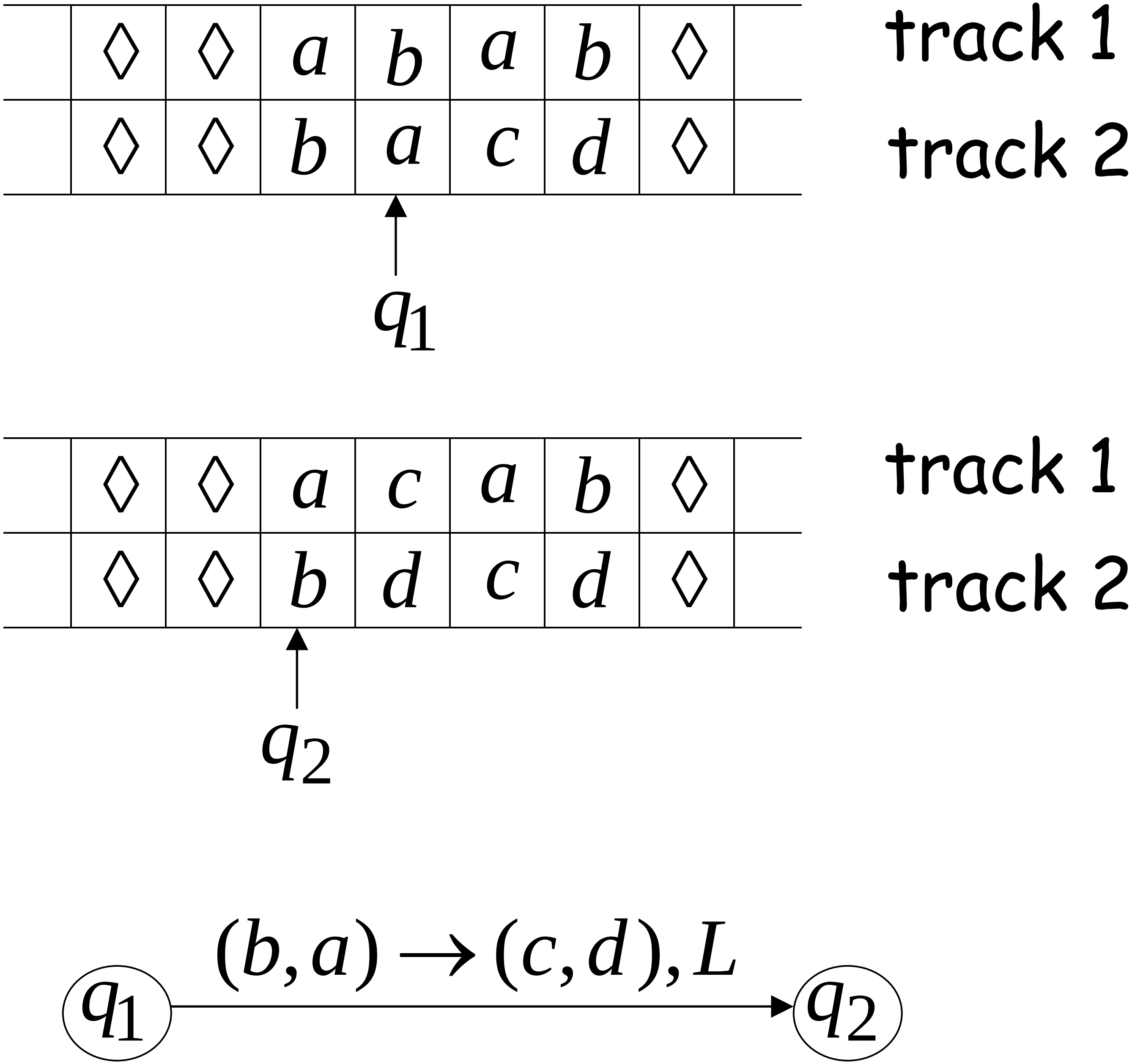


	◊	◊	a	b	a	b	◊		track 1
	◊	◊	b	a	c	d	◊		track 2

One head

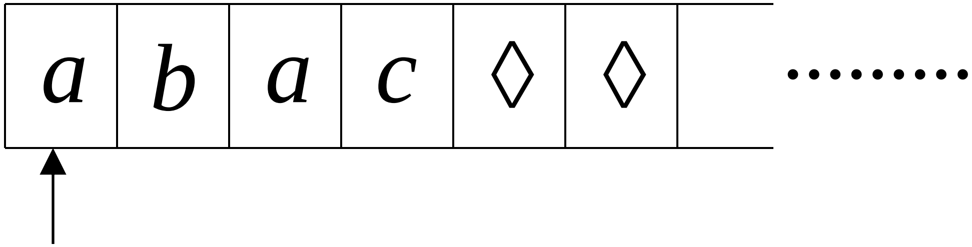
One symbol (a,b)

It is a standard Turing machine, but each tape alphabet symbol describes a pair of actual useful symbols


◊ ◊ a b a b ◊
track 1
◊ ◊ b a c d ◊
track 2
q_1
◊ ◊ a c a b ◊
track 1
◊ ◊ b d c d ◊
track 2
q_2
$(b,a) \rightarrow (c,d), L$
q_1
q_2

Semi-Infinite Tape

The head extends infinitely only to the right



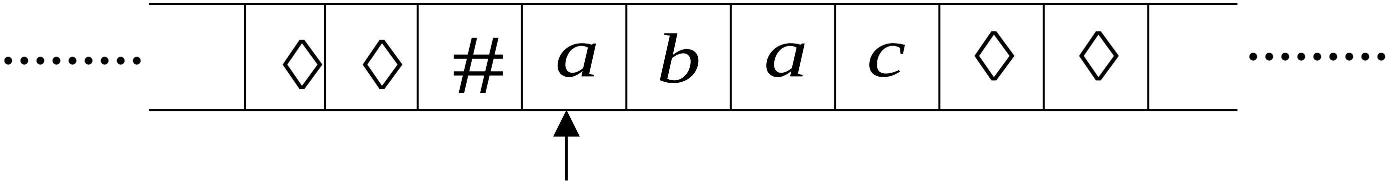
- Initial position is the leftmost cell
- When the head moves left from the border, it returns back to leftmost position

Theorem: Semi-Infinite machines have the same power with Standard Turing machines

Proof:

1. Standard Turing machines simulate Semi-Infinite machines
2. Semi-Infinite Machines simulate Standard Turing machines

1. Standard Turing machines simulate Semi-Infinite machines:

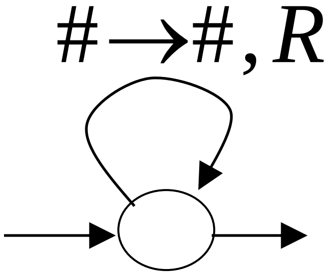


Standard Turing Machine

Semi-Infinite machine modifications

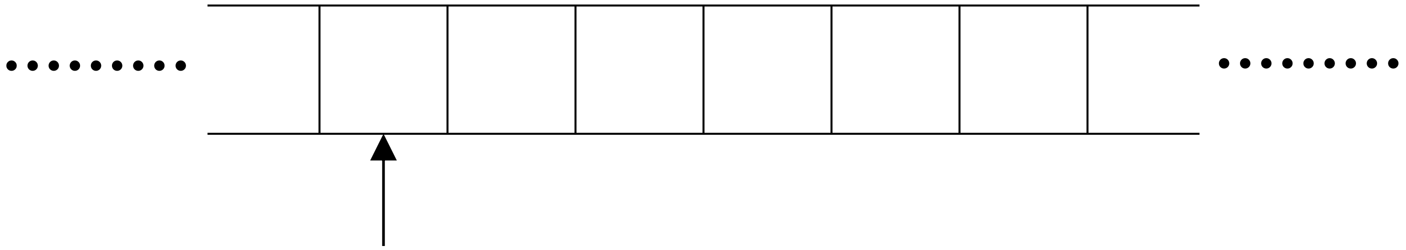
a. insert special symbol #
at left of input string

b. Add a self-loop
to every state
(except states with no
outgoing transitions)

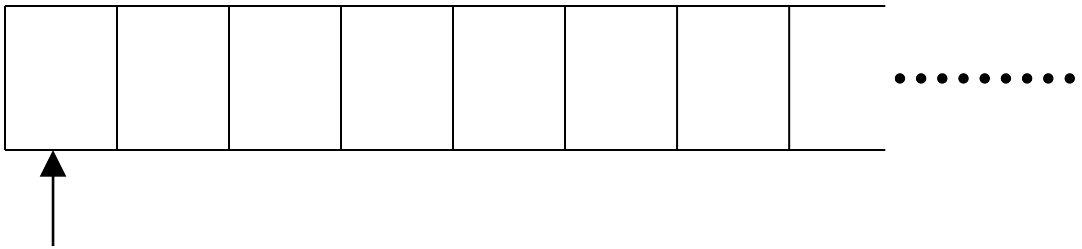


2. Semi-Infinite tape machines simulate Standard Turing machines:

Standard machine

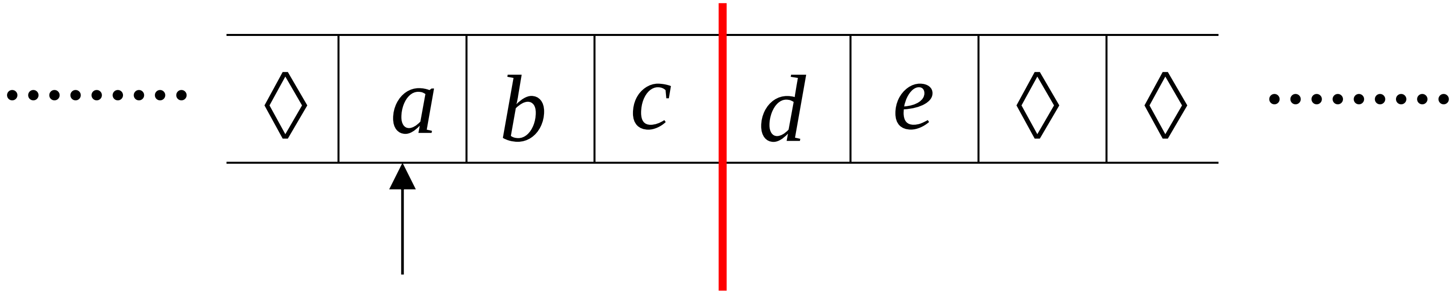


Semi-Infinite tape machine



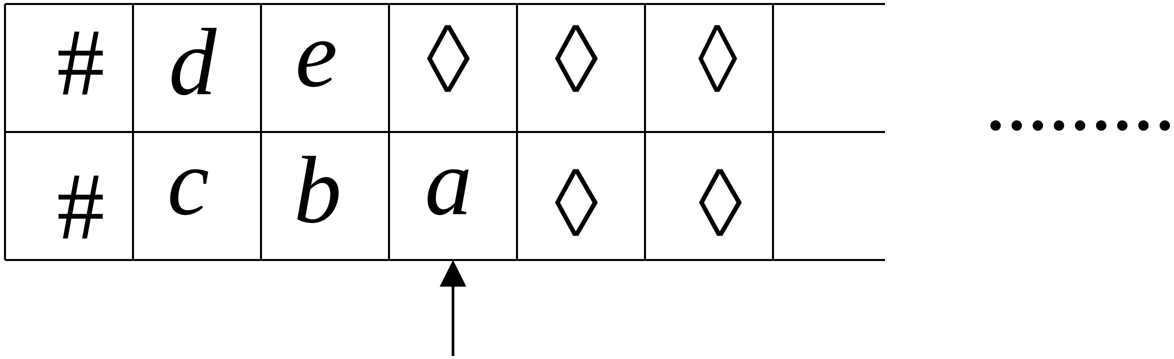
Squeeze infinity of both directions to one direction

Standard machine



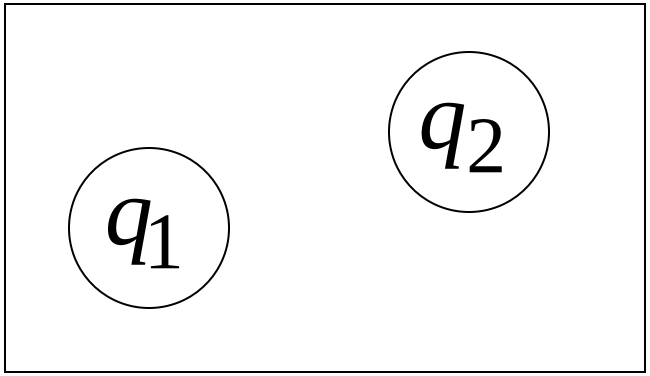
reference point

Semi-Infinite tape machine with two tracks

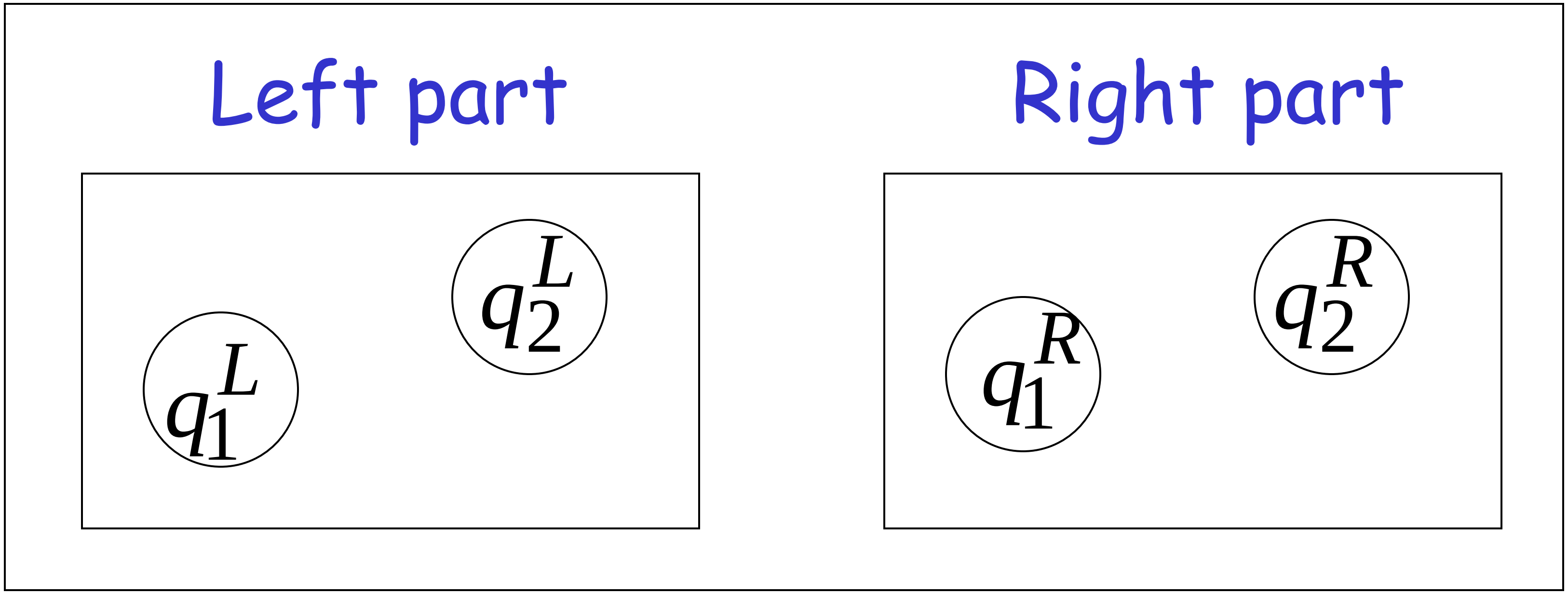


Right part	#	d	e	◊	◊	◊	
Left part	#	c	b	a	◊	◊	

Standard machine

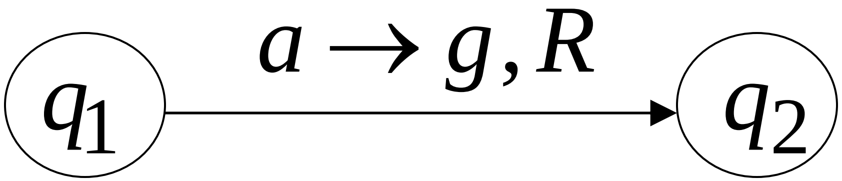


Semi-Infinite tape machine


Left part

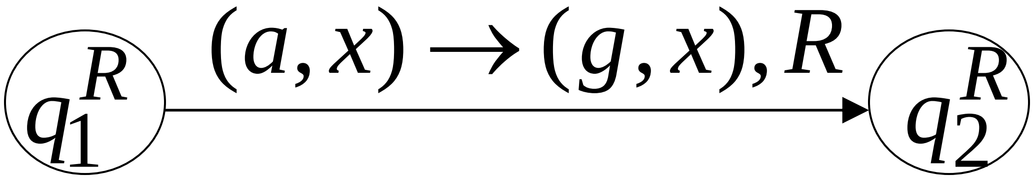
Right part

Standard machine

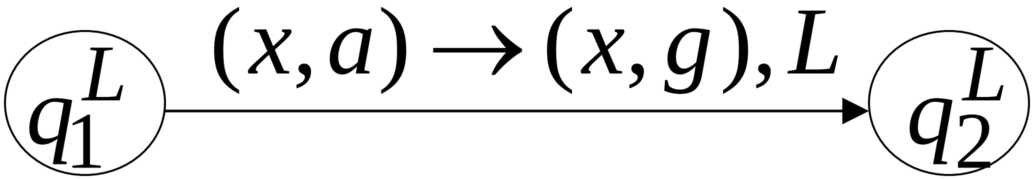


Semi-Infinite tape machine

Right part



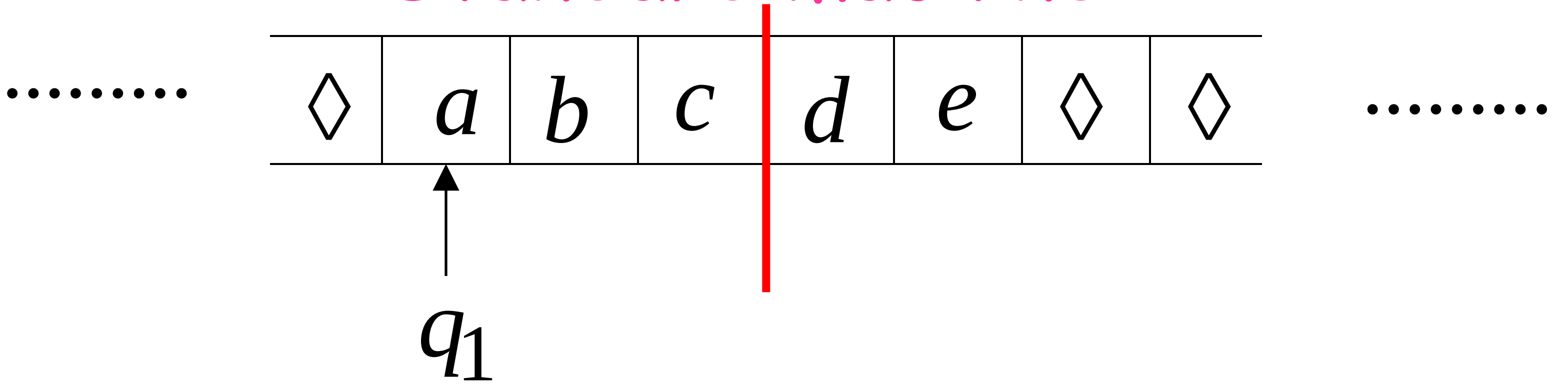
Left part



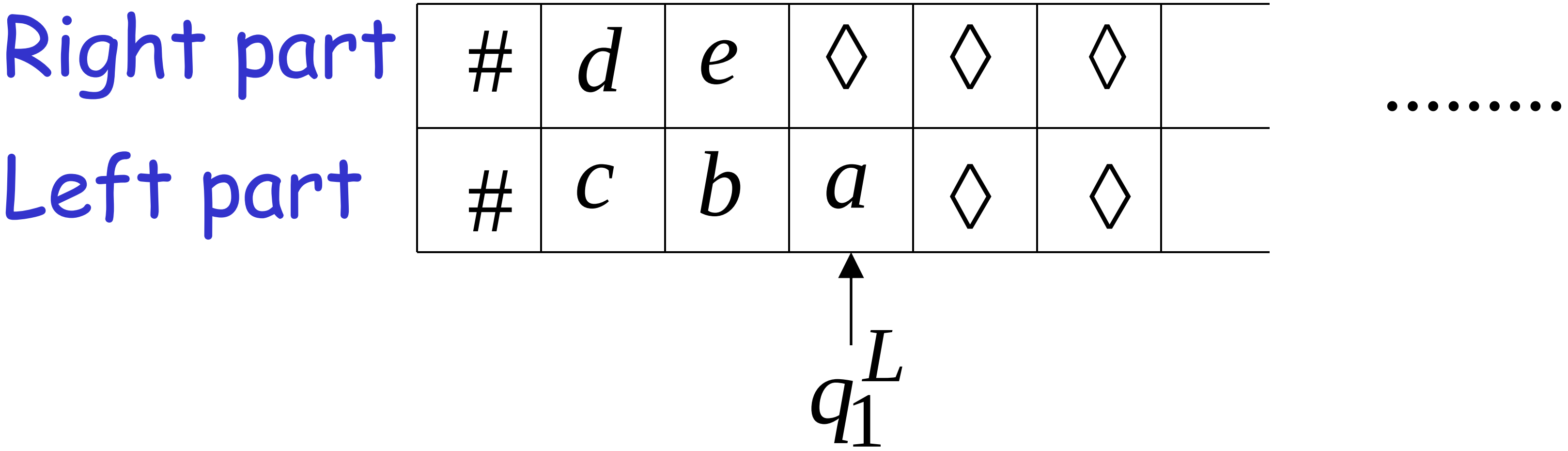
For all tape symbols x

Time 1

Standard machine

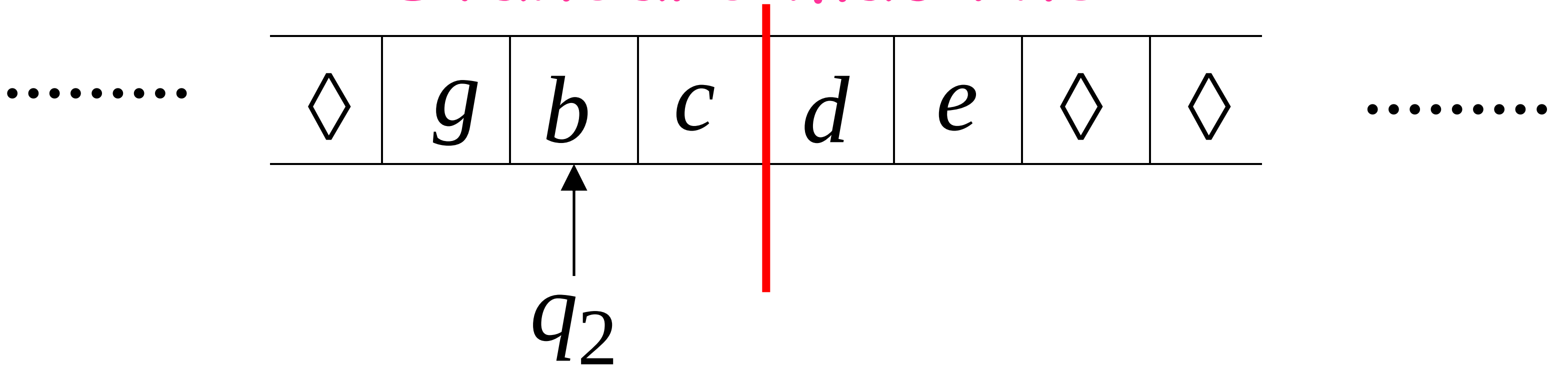


Semi-Infinite tape machine

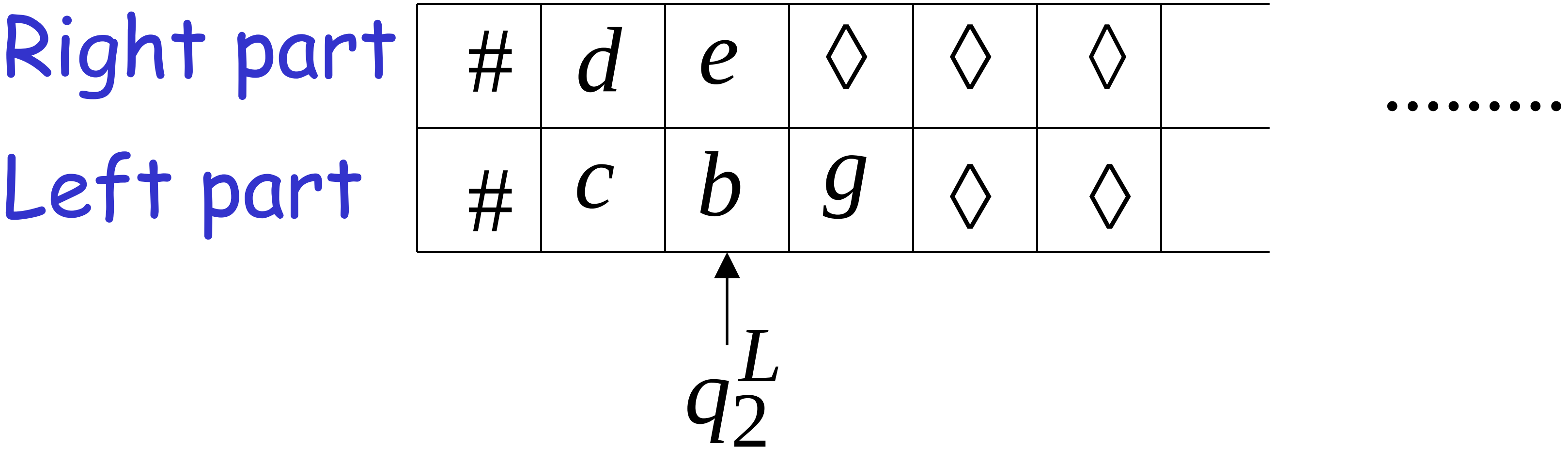


Time 2

Standard machine



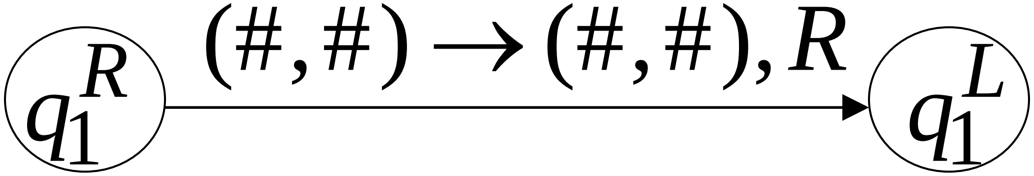
Semi-Infinite tape machine



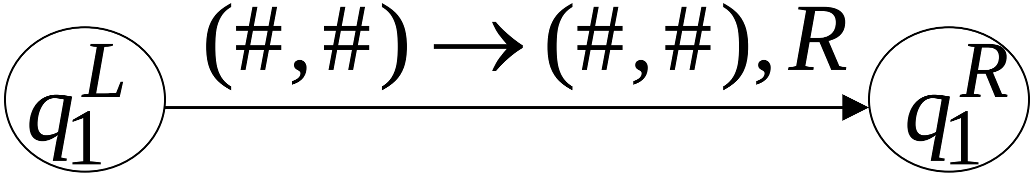
At the border:

Semi-Infinite tape machine

Right part

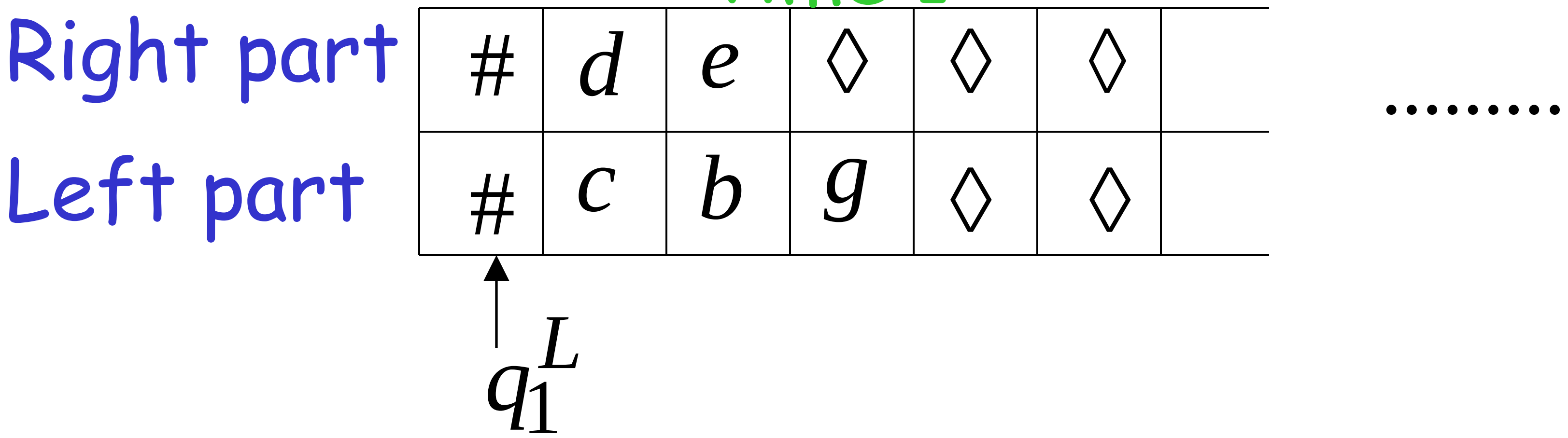


Left part



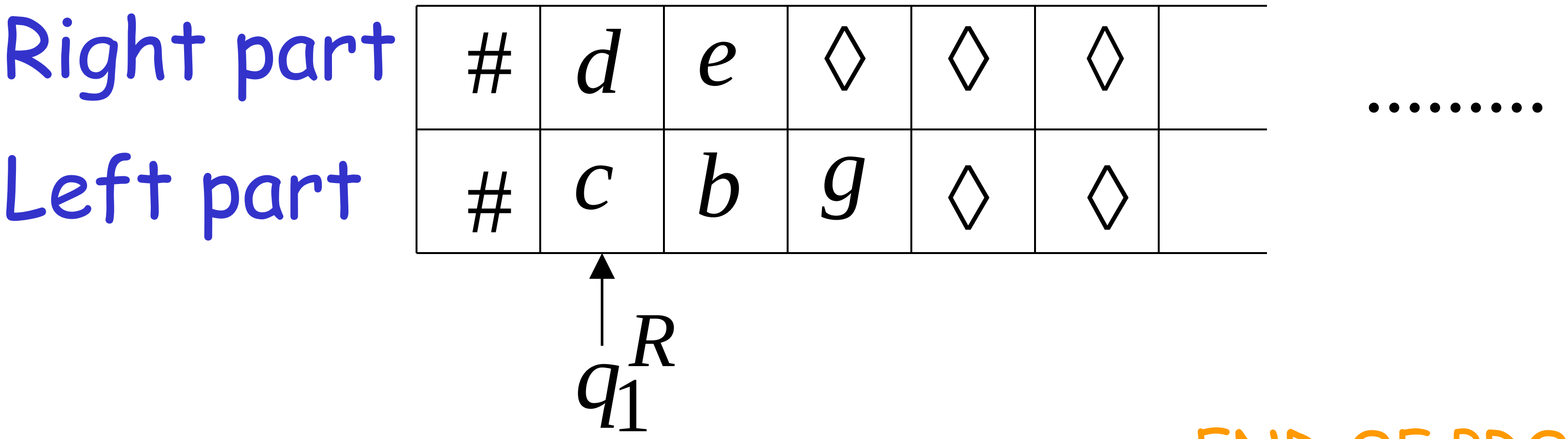
Semi-Infinite tape machine

Time 1



Right part	#	d	e	◊	◊	◊		....
Left part	#	c	b	g	◊	◊		

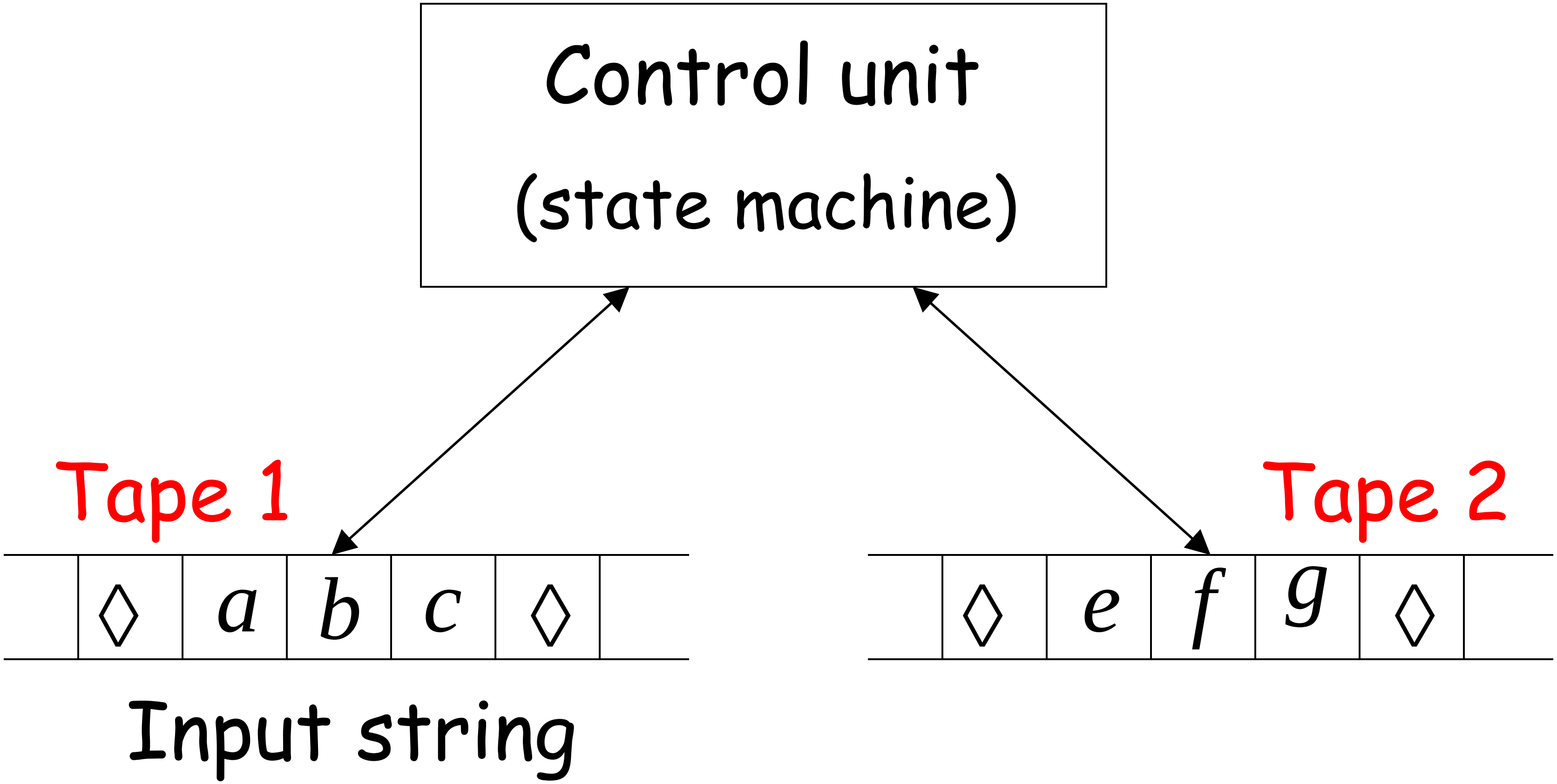
Time 2



Right part	#	d	e	◊	◊	◊		....
Left part	#	c	b	g	◊	◊		

END OF PROOF

Multi-tape Turing Machines



Control unit

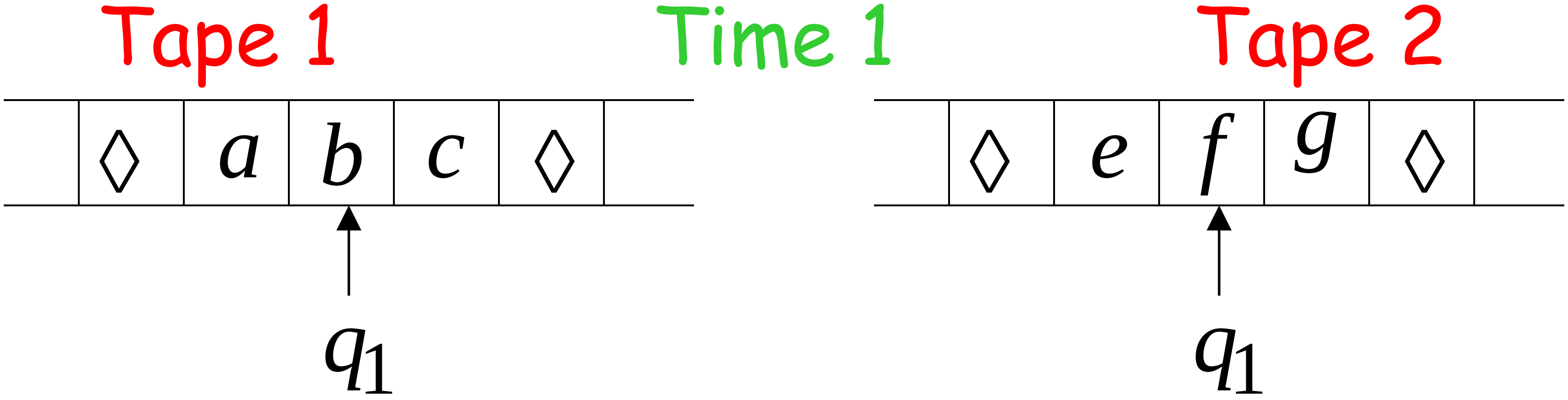
(state machine)

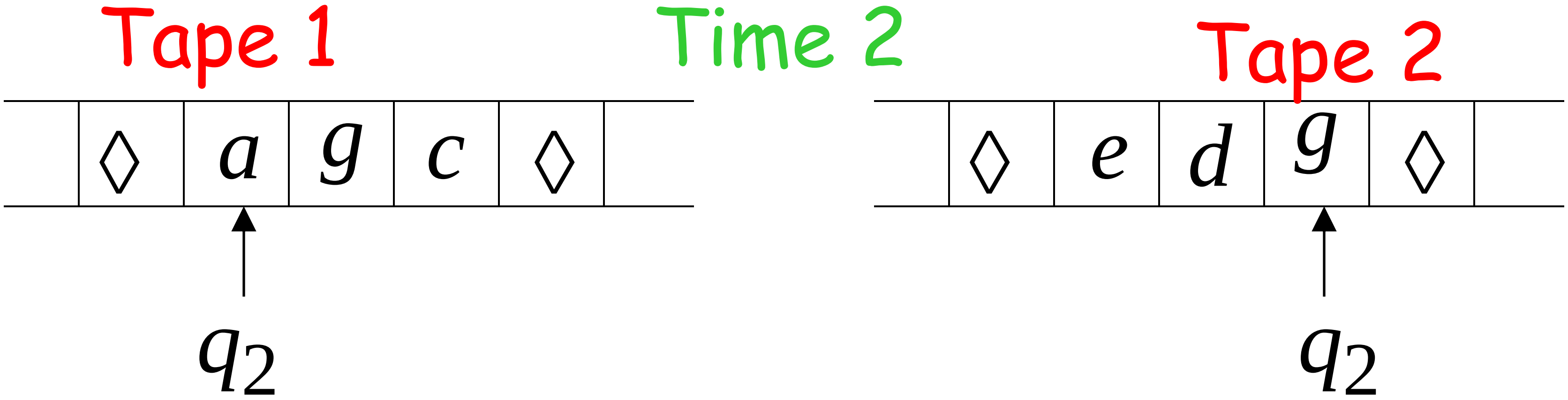
Tape 1

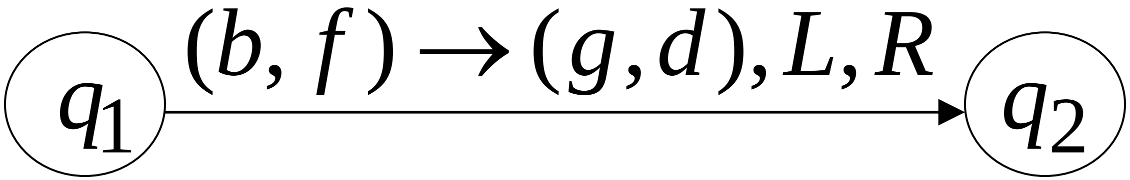
Tape 2

Input string

Input string appears on Tape 1


Tape 1
Time 1
Tape 2
◊
a
b
c
◊
q_1
◊
e
f
g
◊
q_1


Tape 1
Time 2
Tape 2
◊
a
g
c
◊
q_2
◊
e
d
g
◊
q_2


q_1
$(b, f) \rightarrow (g, d), L, R$
q_2

Theorem: Multi-tape machines
have the same power with
Standard Turing machines

Proof: 1. Multi-tape machines
simulate Standard Turing machines

2. Standard Turing machines
simulate Multi-tape machines

1. Multi-tape machines simulate Standard Turing Machines:

Trivial: Use one tape

2. Standard Turing machines simulate Multi-tape machines:

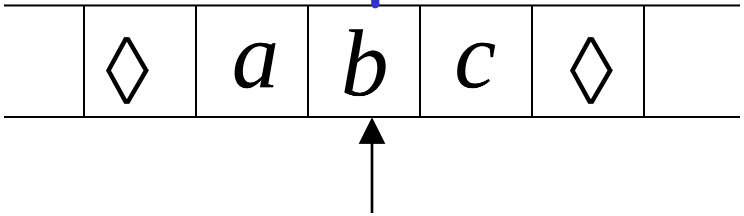
Standard machine:

- Uses a multi-track tape to simulate the multiple tapes

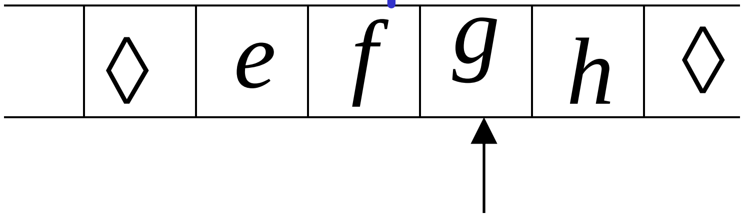
- A tape of the Multi-tape machine corresponds to a pair of tracks

Multi-tape Machine

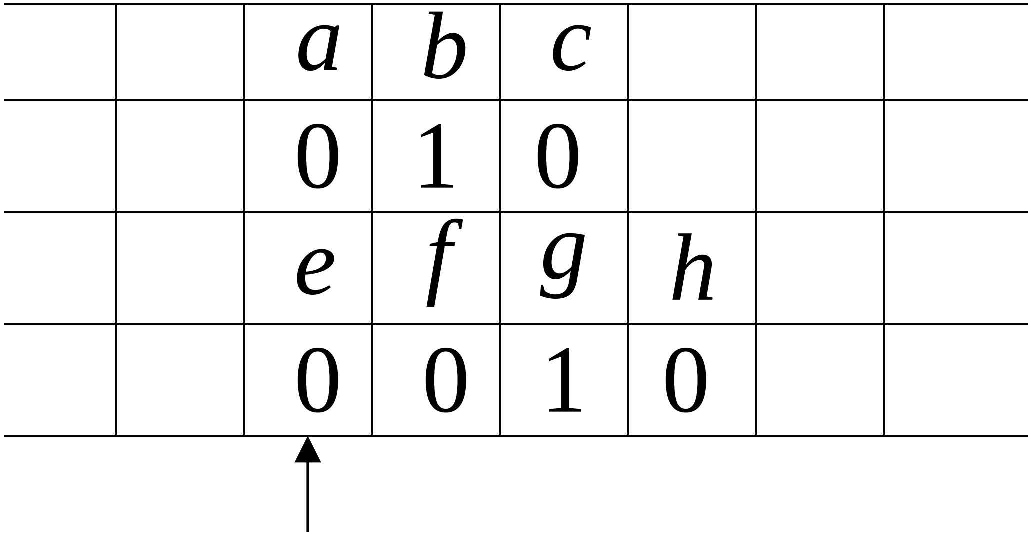
Tape 1



Tape 2



Standard machine with four track tape



		a	b	c			
		0	1	0			
		e	f	g	h		
		0	0	1	0		

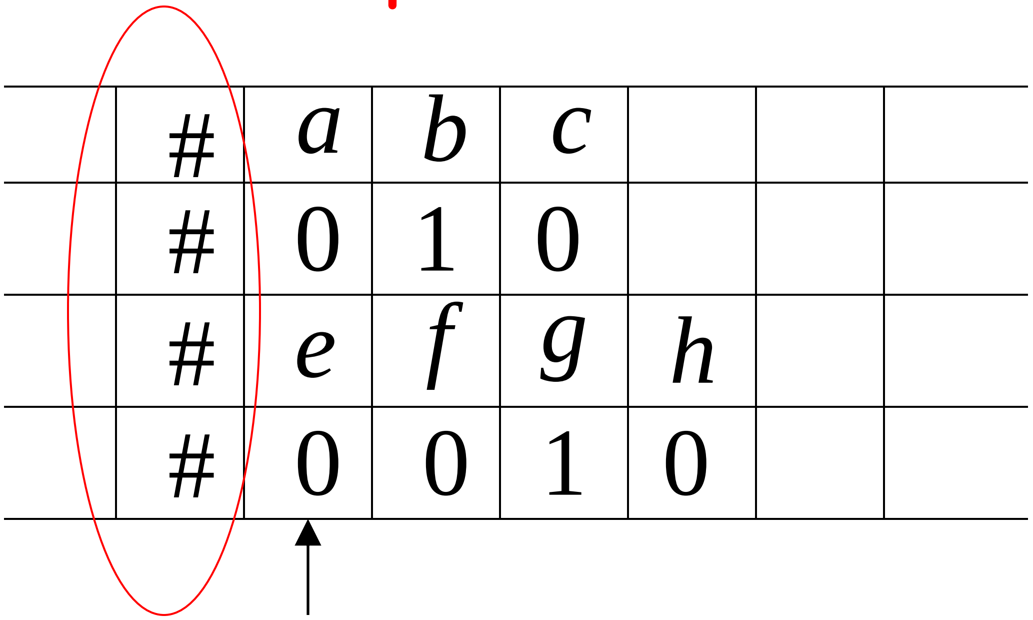
Tape 1

head position

Tape 2

head position

Reference point



	#	a	b	c			
	#	0	1	0			
	#	e	f	g	h		
	#	0	0	1	0		

Tape 1

head position

Tape 2

head position

Repeat for each Multi-tape state transition:

1. Return to reference point
2. Find current symbol in Track 1 and update
3. Return to reference point
4. Find current symbol in Tape 2 and update

END OF PROOF

Same power doesn't imply same speed:

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

Standard Turing machine: $O(n^2)$ time

Go back and forth $O(n^2)$ times
to match the a's with the b's

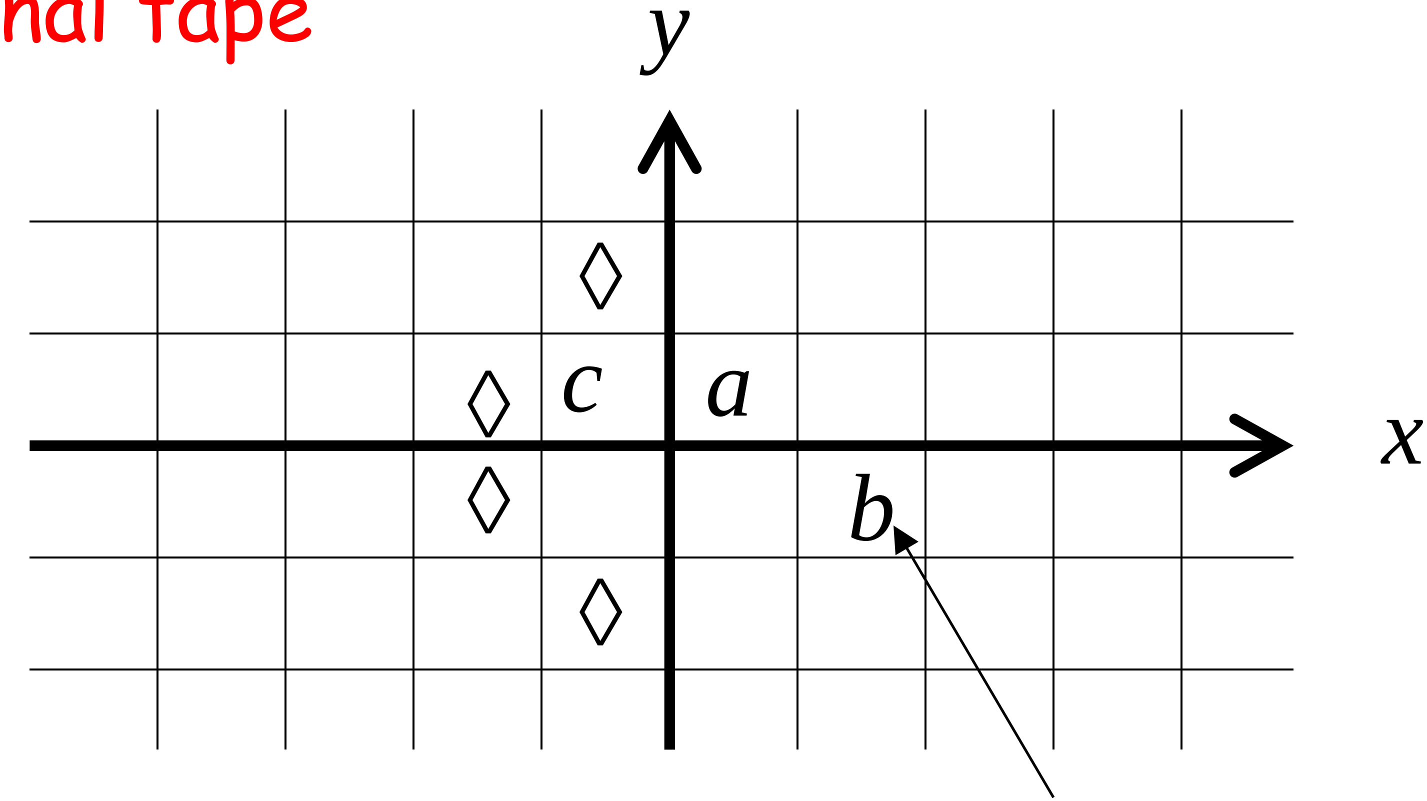
2-tape machine: $O(n)$ time

1. Copy b^n to tape 2 ($O(n)$ steps)

2. Compare a^n on tape 1
and b^n on tape 2 ($O(n)$ steps)

Multidimensional Turing Machines

2-dimensional tape



MOVES: L,R,U,D

U: up D: down

HEAD

Position: +2, -1

Theorem: Multidimensional machines have the same power with Standard Turing machines

Proof:

1. Multidimensional machines simulate Standard Turing machines

2. Standard Turing machines simulate Multi-Dimensional machines

1. Multidimensional machines simulate Standard Turing machines

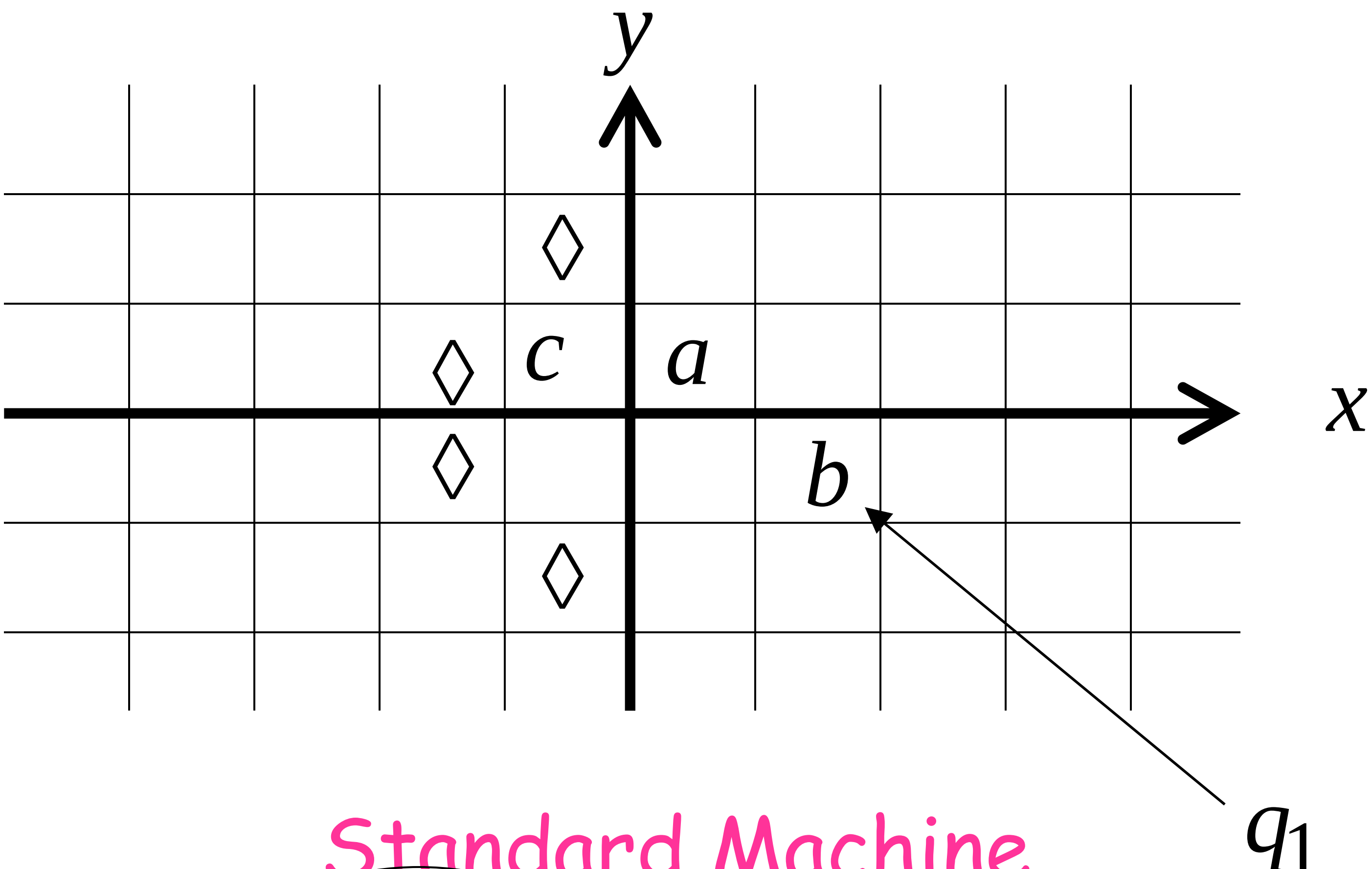
Trivial: Use one dimension

2. Standard Turing machines simulate Multidimensional machines

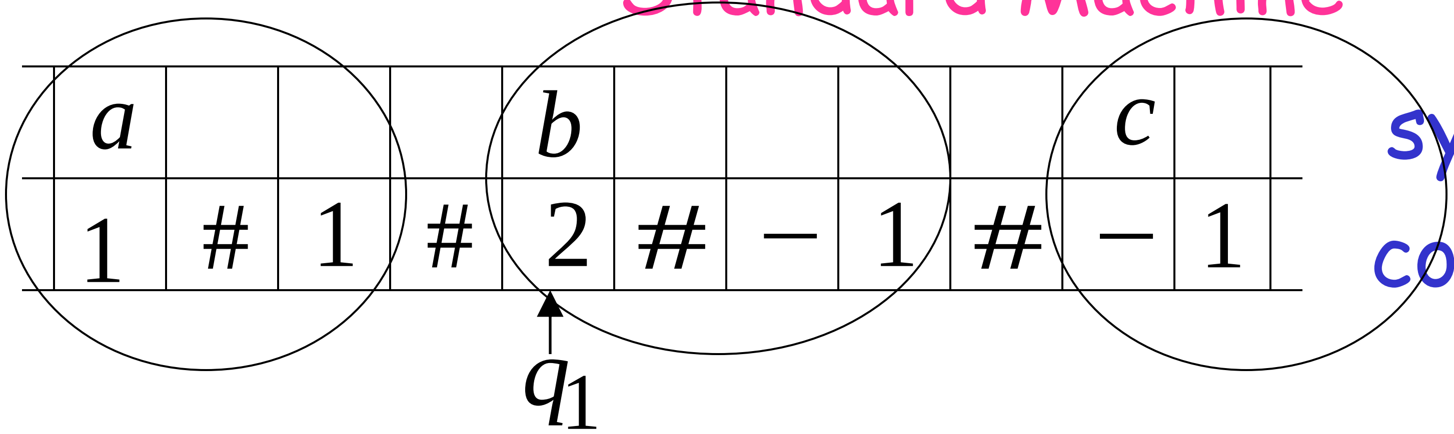
Standard machine:

- Use a two track tape
- Store symbols in track 1
- Store coordinates in track 2

2-dimensional machine



Standard Machine



a				b					c	
1	#	1	#	2	#	−	1	#	−	1

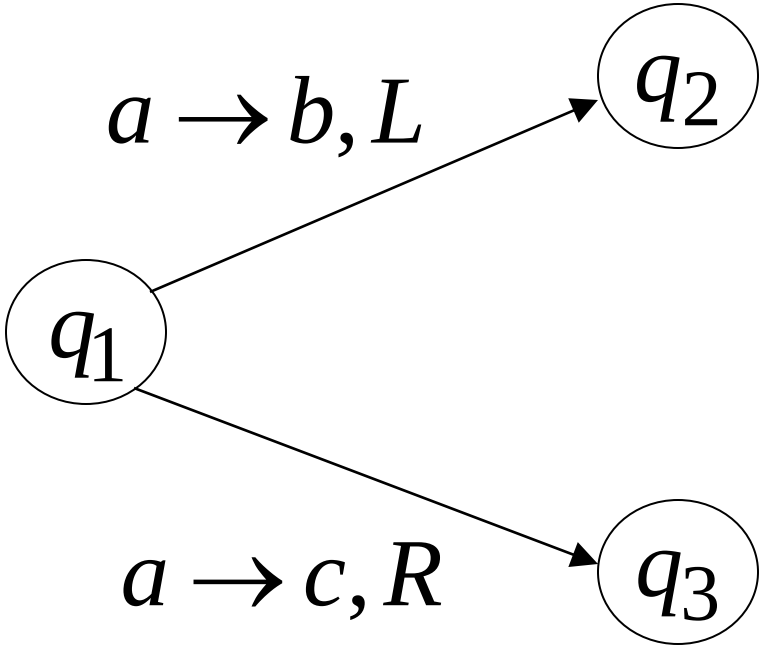
Standard machine:

Repeat for each transition followed in the 2-dimensional machine:

1. Update current symbol
2. Compute coordinates of next position
3. Find next position on tape

END OF PROOF

Nondeterministic Turing Machines

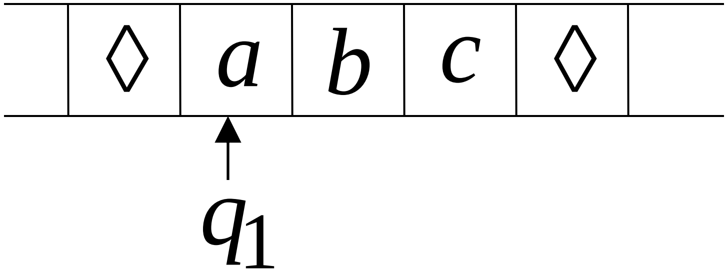


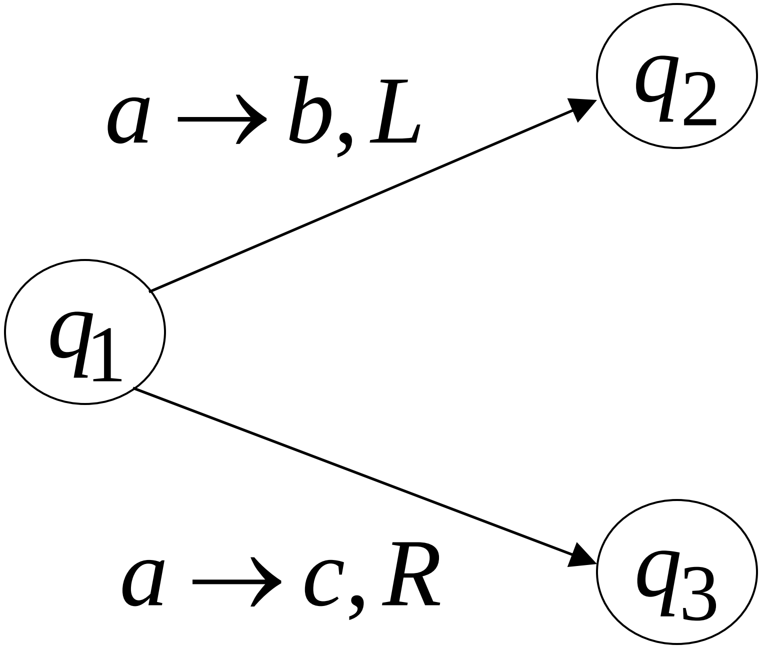
Choice 1

Choice 2

Allows Non Deterministic Choices

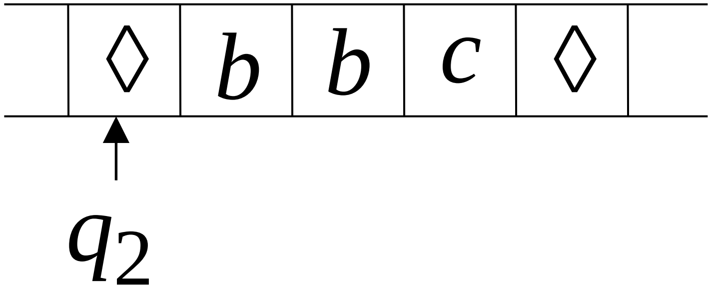
Time 0



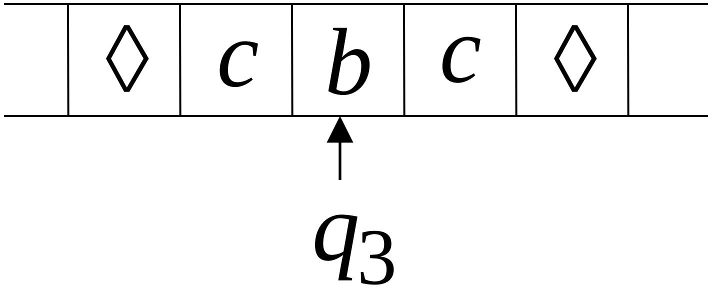


Time 1

Choice 1



Choice 2



Input string w is accepted if there is a computation:

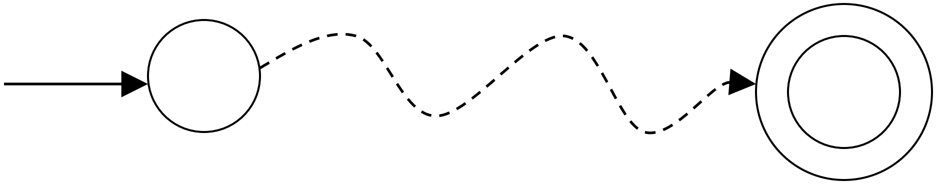
$$q_0 w \overset{*}{\succ} x q_f y$$

Initial configuration

Any accept state

Final Configuration

There is a computation:



Theorem: Nondeterministic machines have the same power with Standard Turing machines

Proof: 1. Nondeterministic machines simulate Standard Turing machines

2. Standard Turing machines simulate Nondeterministic machines

1. Nondeterministic Machines simulate Standard (deterministic) Turing Machines

Trivial: every deterministic machine is also nondeterministic

2. Standard (deterministic) Turing machines simulate Nondeterministic machines:

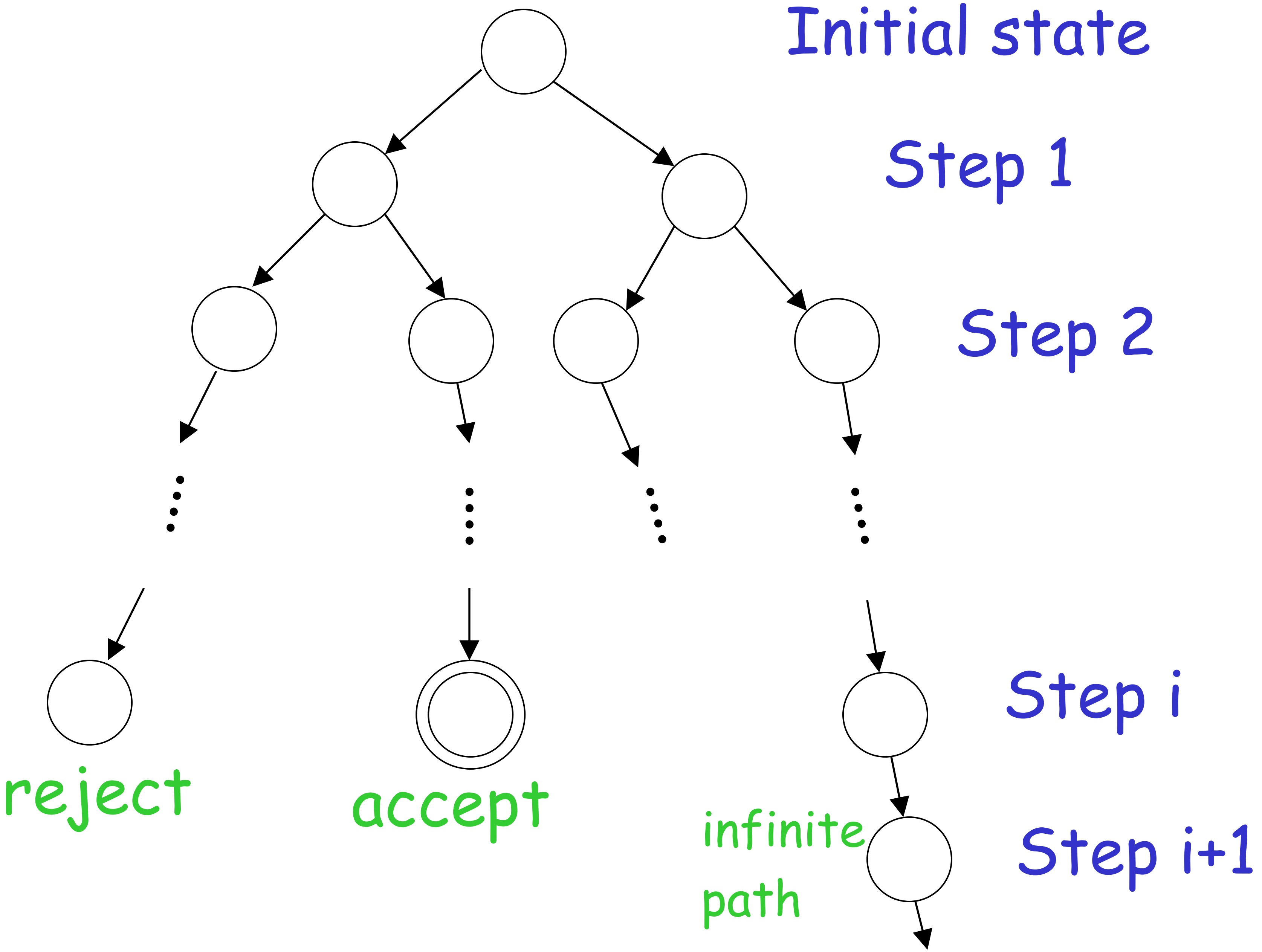
Deterministic machine:

- Uses a 2-dimensional tape

  (equivalent to standard Turing machine with one tape)

- Stores all possible computations of the non-deterministic machine on the 2-dimensional tape

All possible computation paths

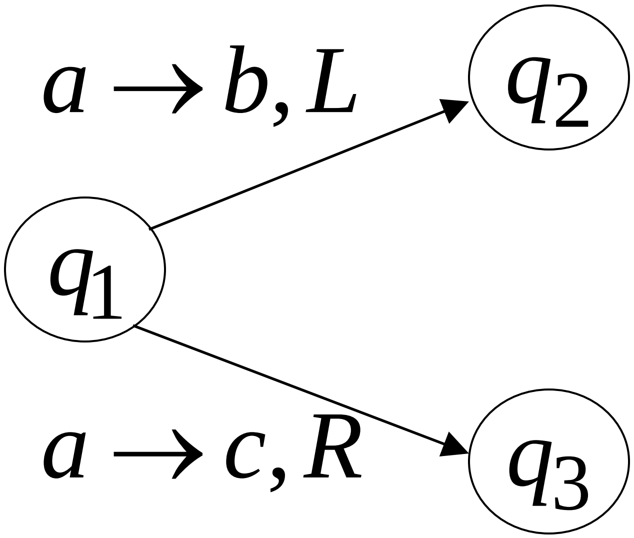


The Deterministic Turing machine simulates all possible computation paths:

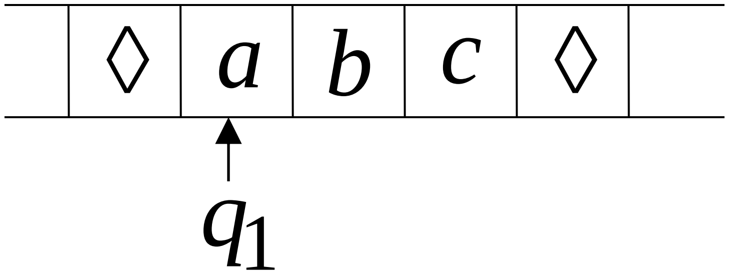
- simultaneously
- step-by-step
- with breadth-first search

depth-first may result getting stuck at exploring
an infinite path before discovering the accepting path

NonDeterministic machine



Time 0



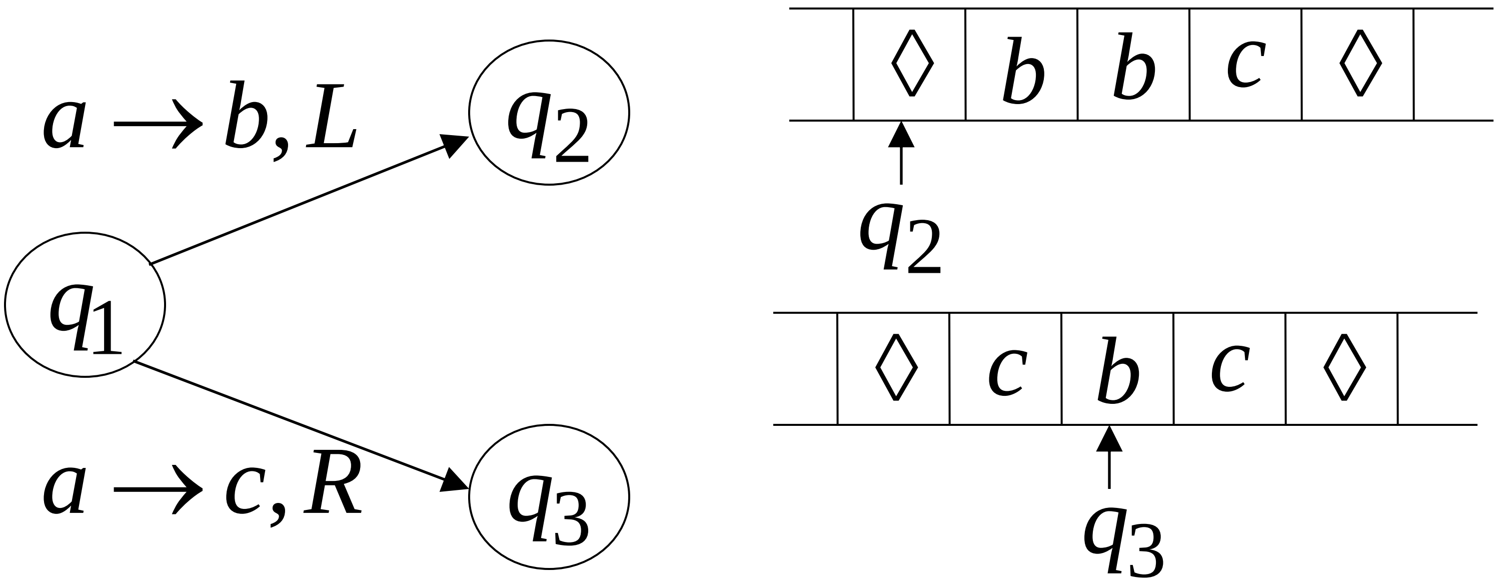
Deterministic machine

	#	#	#	#	#	#	
	#	a	b	c	#		
	#	q_1			#		
	#	#	#	#	#		

current configuration

NonDeterministic machine

Time 1



Choice 1

Choice 2

Deterministic machine

	#	#	#	#	#	#	
#		b	b	c	#		
#	q_2				#		
#		c	b	c	#		
#			q_3		#		

Computation 1

Computation 2

Deterministic Turing machine

```
Repeat
    For each configuration in current step
    of non-deterministic machine,
    if there are two or more choices:
        1. Replicate configuration
        2. Change the state in the replicas
Until either the input string is accepted
    or rejected in all configurations
```

If the non-deterministic machine accepts the input string:

The deterministic machine accepts and halts too

The simulation takes in the worst case exponential time compared to the shortest length of an accepting path

If the non-deterministic machine does not accept the input string:

1. The simulation halts if all paths reach a halting state

OR

2. The simulation never terminates if there is a never-ending path (infinite loop)

In either case the deterministic machine rejects too (1. by halting or 2. by simulating the infinite loop)

END OF PROOF

A Universal Turing Machine

A limitation of Turing Machines:

Turing Machines are "hardwired"

they execute
only one program

Real Computers are re-programmable

Solution: Universal Turing Machine

Attributes:

- Reprogrammable machine
- Simulates any other Turing Machine

Universal Turing Machine

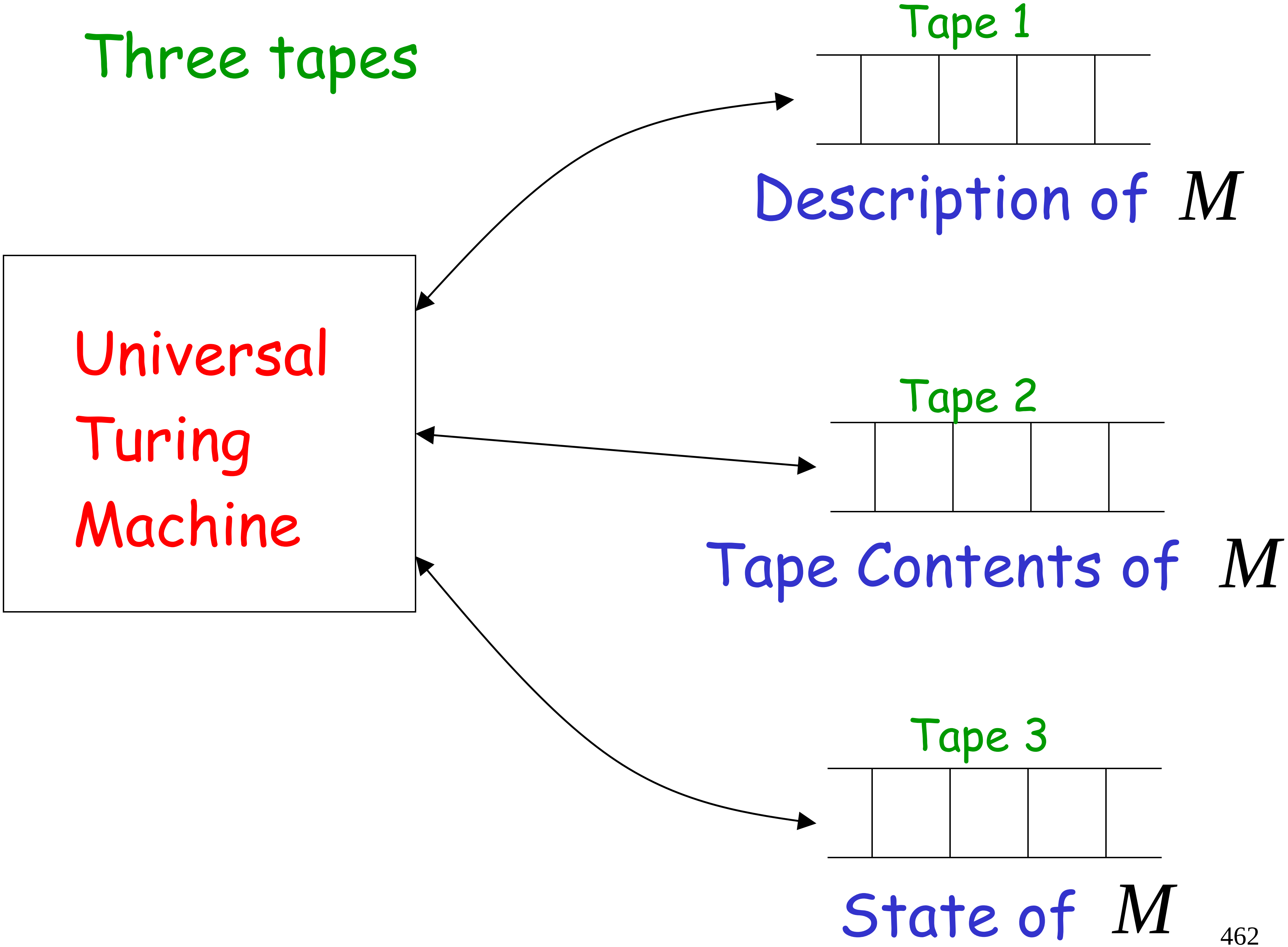
simulates any Turing Machine M

Input of Universal Turing Machine:

Description of transitions of M

Input string of M

Three tapes



Universal Turing Machine

Tape 1

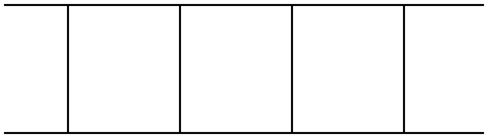
Description of M

Tape 2

Tape Contents of M

Tape 3

State of M



Tape 1

Description of M

We describe Turing machine M
as a string of symbols:

We encode M as a string of symbols

Alphabet Encoding

Symbols:	a	b	c	d	$\cdots$
	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	
Encoding:	1	11	111	1111	

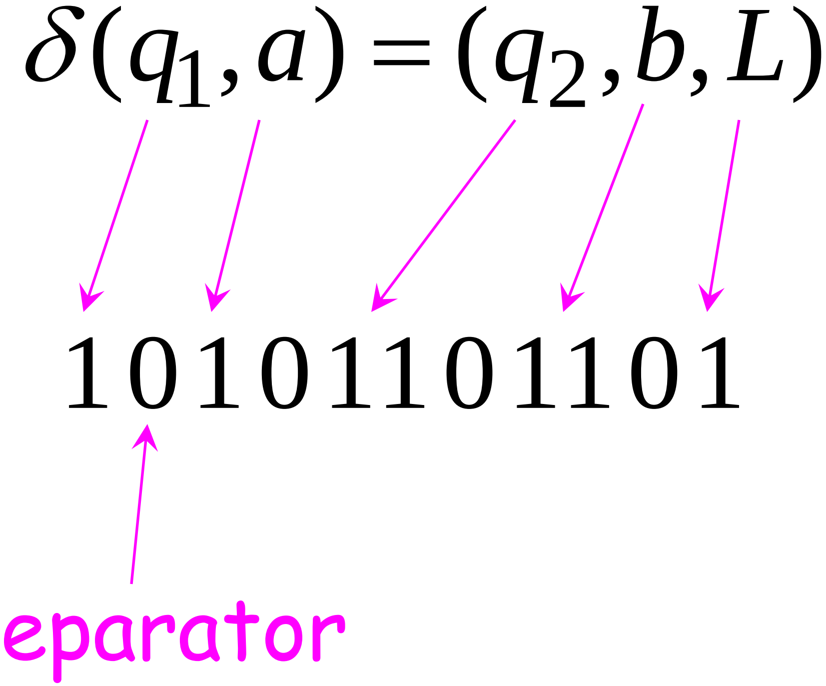
State Encoding

States:	q_1	q_2	q_3	q_4	$\cdots$
Encoding:	1	11	111	1111	

Head Move Encoding

Move:	L	R
Encoding:	1	11

Transition Encoding



Transition: $\delta(q_1, a) = (q_2, b, L)$

Encoding: $1\,0\,1\,0\,1\,1\,0\,1\,1\,0\,1$

separator

Turing Machine Encoding

Transitions:

$$\delta(q_1, a) = (q_2, b, L) \qquad \delta(q_2, b) = (q_3, c, R)$$

Encoding:

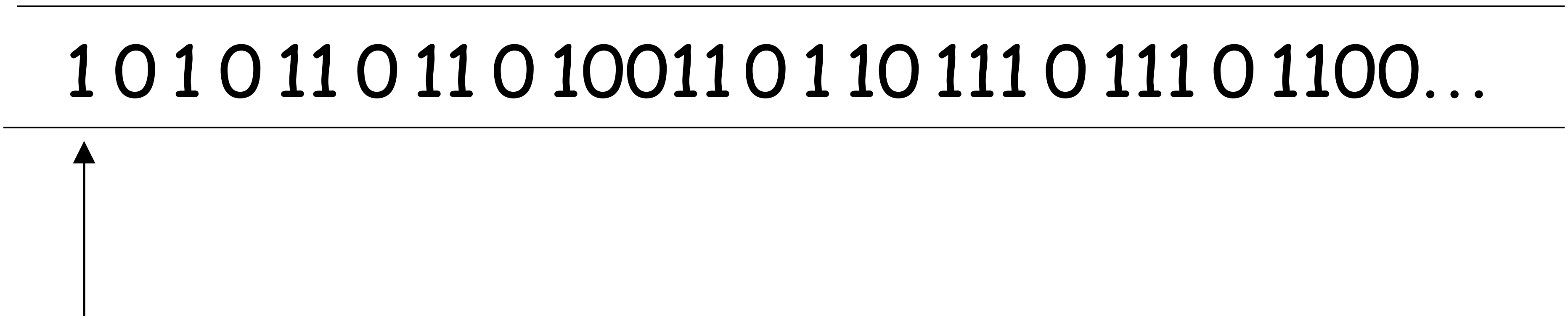
$$10101101101 \; 00 \; 1101101110111011$$

separator

Tape 1 contents of Universal Turing Machine:

binary encoding
of the simulated machine M

Tape 1



1 0 1 0 11 0 11 0 10011 0 1 10 111 0 111 0 1100…

A Turing Machine is described
with a binary string of 0's and 1's

Therefore:

The set of Turing machines
forms a language:

each string of this language is
the binary encoding of a Turing Machine

Language of Turing Machines

L = { 1010110101, (Turing Machine 1)

101011101011, (Turing Machine 2)

11101011110101111, ……

…… }

Countable Sets

Infinite sets are either:

Countable

or

Uncountable

Countable set:

There is a one to one correspondence (injection)

of

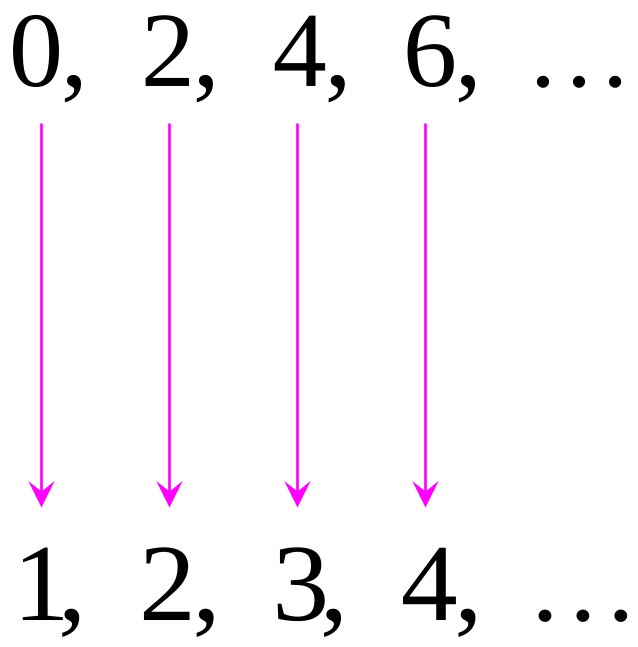
elements of the set

to

Positive integers (1,2,3,…)

Every element of the set is mapped to a positive number such that no two elements are mapped to same number

Example: The set of even integers is countable



Even integers: $0, \; 2, \; 4, \; 6, \; \ldots$

(positive)

Correspondence:

Positive integers: $1, \; 2, \; 3, \; 4, \; \ldots$

$2n$ corresponds to $n+1$

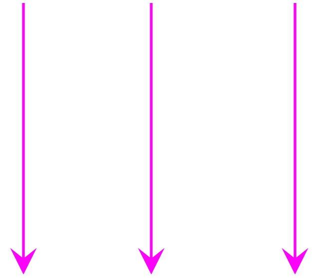
Example: The set of rational numbers is countable

Rational numbers: $\frac{1}{2}, \frac{3}{4}, \frac{7}{8}, \ldots$

Naïve Approach

Nominator 1

Rational numbers: $\frac{1}{1}, \frac{1}{2}, \frac{1}{3}, \ldots$



Correspondence:

Positive integers: $1, 2, 3, \ldots$

Doesn't work:

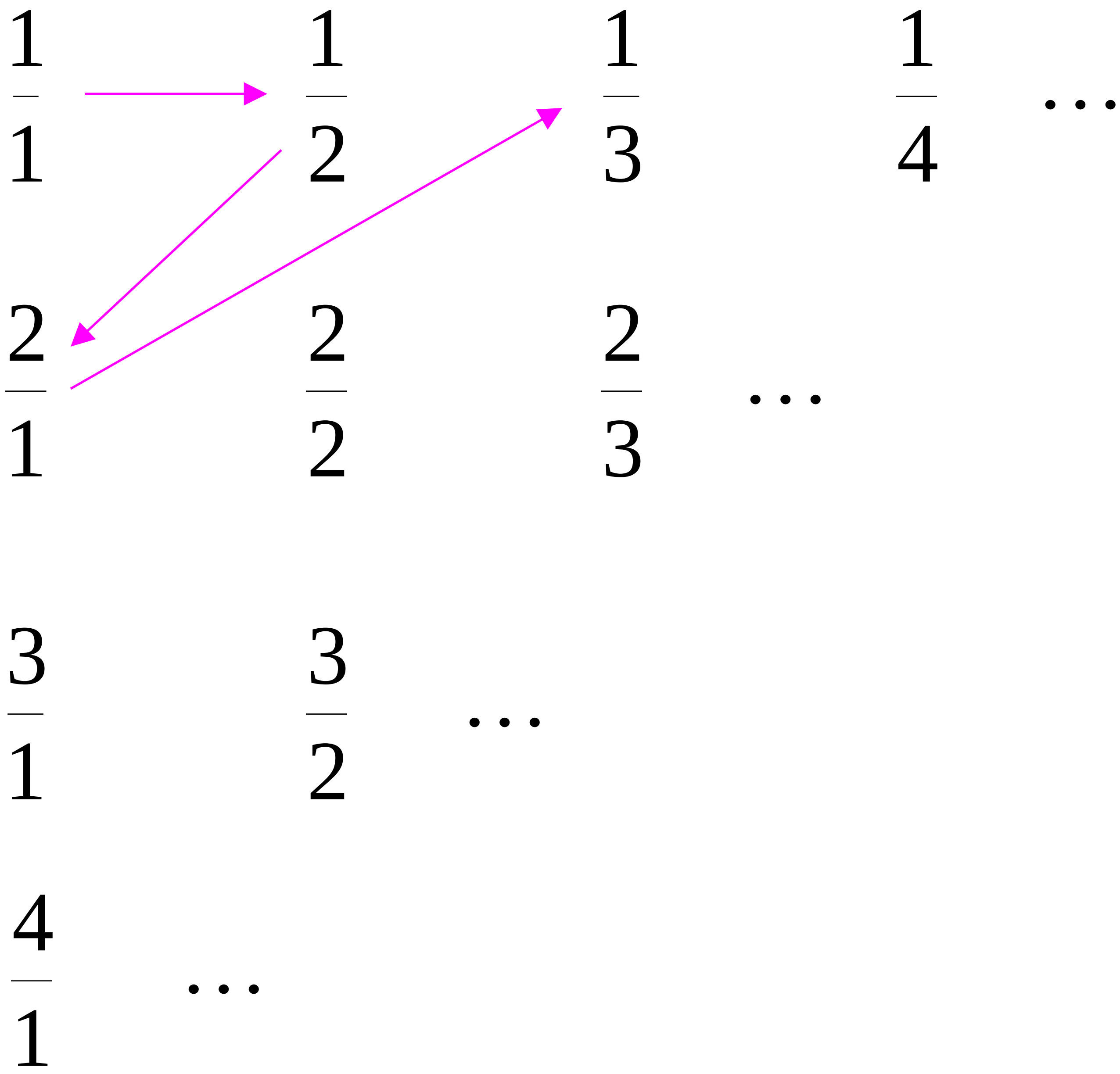
we will never count
numbers with nominator 2: $\frac{2}{1}, \frac{2}{2}, \frac{2}{3}, \ldots$

Better Approach

$$\begin{array}{ccccc}
\frac{1}{1} & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \cdots \\
\frac{2}{1} & \frac{2}{2} & \frac{2}{3} & \cdots & \\
\frac{3}{1} & \frac{3}{2} & \cdots & & \\
\frac{4}{1} & \cdots & & &
\end{array}$$

$$\begin{tikzcd}
\frac{1}{1} \arrow[r] & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \cdots \\
\frac{2}{1} & \frac{2}{2} & \frac{2}{3} & \cdots & \\
\frac{3}{1} & \frac{3}{2} & \cdots & & \\
\frac{4}{1} & \cdots & & &
\end{tikzcd}$$

$$\begin{tikzcd}
\frac{1}{1} \arrow[r] & \frac{1}{2} \arrow[dl] & \frac{1}{3} & \frac{1}{4} & \cdots \\
\frac{2}{1} & \frac{2}{2} & \frac{2}{3} & \cdots & \\
\frac{3}{1} & \frac{3}{2} & \cdots & & \\
\frac{4}{1} & \cdots & & &
\end{tikzcd}$$



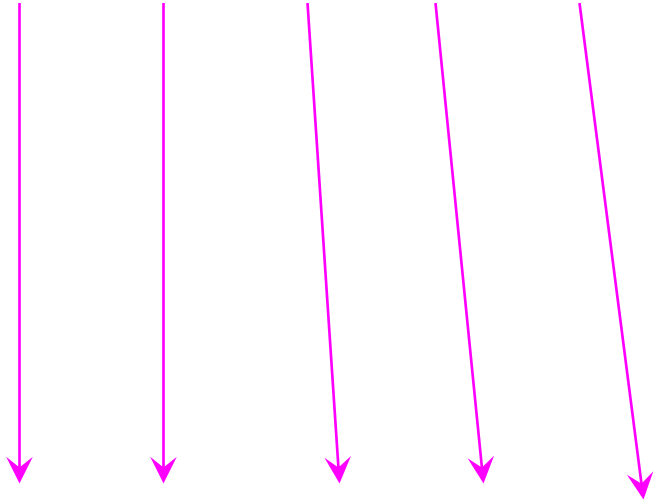
$$\begin{array}{cccccc} \frac{1}{1} & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \cdots \\ \frac{2}{1} & \frac{2}{2} & \frac{2}{3} & \cdots & \\ \frac{3}{1} & \frac{3}{2} & \cdots & & \\ \frac{4}{1} & \cdots & & & \end{array}$$

$$\begin{tikzcd}
\frac{1}{1} \arrow[r] & \frac{1}{2} \arrow[dl] & \frac{1}{3} \arrow[dl] & \frac{1}{4} & \cdots \\
\frac{2}{1} \arrow[urr] & \frac{2}{2} & \frac{2}{3} & \cdots & \\
\frac{3}{1} & \frac{3}{2} & \cdots & & \\
\frac{4}{1} & \cdots & & &
\end{tikzcd}$$

$$\begin{tikzcd}
\frac{1}{1} \arrow[r] & \frac{1}{2} \arrow[dl] & \frac{1}{3} \arrow[dl] & \frac{1}{4} \arrow[dl] & \cdots \\
\frac{2}{1} \arrow[urr] & \frac{2}{2} \arrow[dl] & \frac{2}{3} \arrow[dl] & \cdots & \\
\frac{3}{1} \arrow[uurrr] & \frac{3}{2} \arrow[dl] & \cdots & & \\
\frac{4}{1} & \cdots & & &
\end{tikzcd}$$

Rational Numbers: $\frac{1}{1}, \frac{1}{2}, \frac{2}{1}, \frac{1}{3}, \frac{2}{2}, \ldots$

Correspondence:



Positive Integers: $1, \ 2, \ 3, \ 4, \ 5, \ \ldots$

We proved:

the set of rational numbers is countable
by describing an enumeration procedure
(enumerator)
for the correspondence to natural numbers

Definition

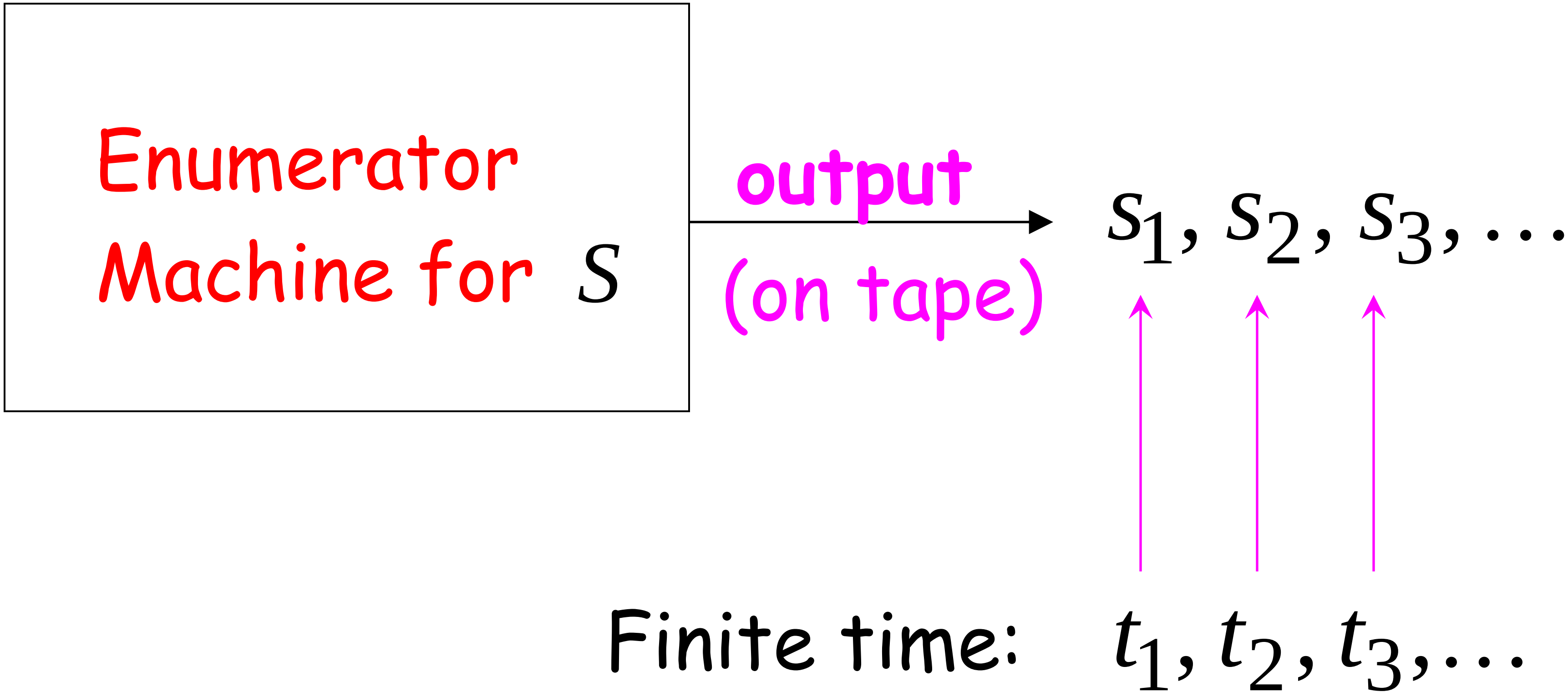
Let S be a set of strings (Language)

An **enumerator** for S is a Turing Machine that generates (prints on tape) all the strings of S one by one

and

each string is generated in finite time

strings $s_1, s_2, s_3, \ldots \in S$



Enumerator Machine for S

output (on tape)

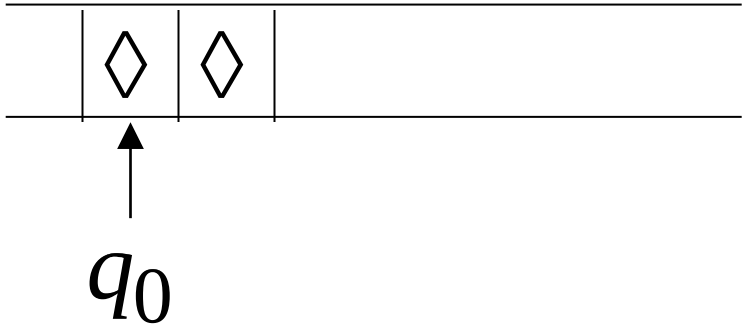
$s_1, s_2, s_3, \ldots$

Finite time: $t_1, t_2, t_3, \ldots$

Enumerator Machine

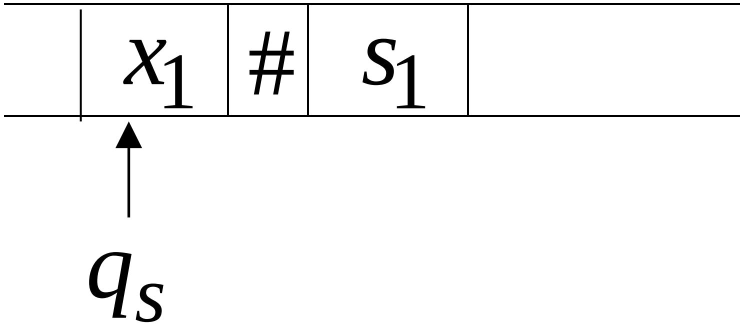
Configuration

Time 0

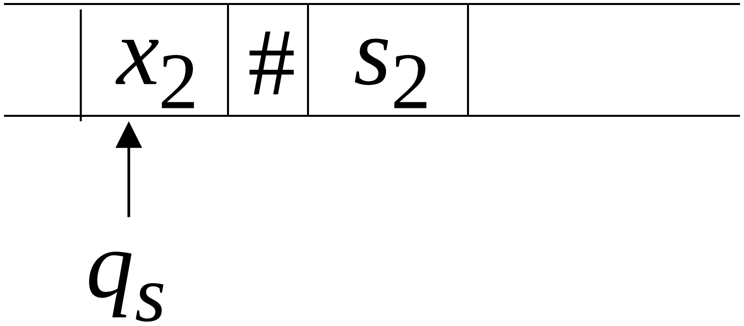


prints s_1

Time t_1

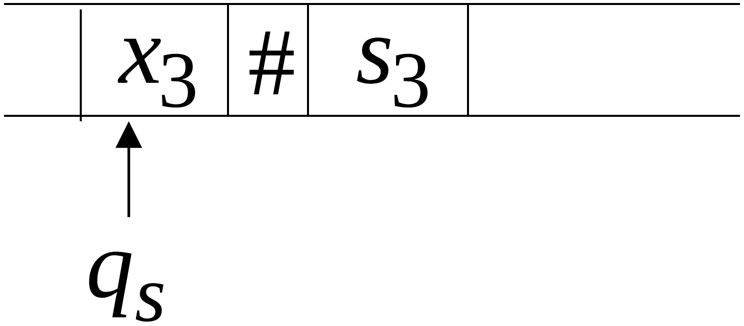


Time t_2



prints s_2

Time t_3



prints s_3

Observation:

If for a set S there is an enumerator, then the set is countable

The enumerator describes the correspondence of S to natural numbers

Example: The set of strings $S = \{a,b,c\}^{+}$ is countable

Approach:

We will describe an enumerator for S

Naive enumerator:

Produce the strings in lexicographic order:

$$s_1 = a$$

$$s_2 = aa$$

$$\vdots \quad aaa$$

$$aaaa$$

$$\ldots\ldots$$

Doesn't work:

strings starting with b
will never be produced

Better procedure: **Proper Order** (Canonical Order)

1. Produce all strings of length 1

2. Produce all strings of length 2

3. Produce all strings of length 3

4. Produce all strings of length 4

⋮

Produce strings in

Proper Order:

$$\left.\begin{array}{ll} s_1 = & a \\ s_2 = & b \\ \vdots & c \end{array}\right\} \text{length 1}$$

$$\left.\begin{array}{l} aa \\ ab \\ ac \\ ba \\ bb \\ bc \\ ca \\ cb \\ cc \end{array}\right\} \text{length 2}$$

$$\left.\begin{array}{l} aaa \\ aab \\ aac \\ \ldots\ldots \end{array}\right\} \text{length 3}$$

Theorem: The set of all Turing Machines is countable

Proof: Any Turing Machine can be encoded with a binary string of 0's and 1's

Find an enumeration procedure for the set of Turing Machine strings

Enumerator:

Repeat

1. Generate the next binary string of 0's and 1's in proper order

2. Check if the string describes a Turing Machine

    if **YES:** print string on output tape

    if **NO:** ignore string

Binary strings		Turing Machines
0	ignore	
1	ignore	
00	ignore	
01		
⋮		
1 0 1 0 11 0 11 0 0		
1 0 1 0 11 0 11 0 1	$\xrightarrow{s_1}$	1 0 1 0 11 0 11 0 1
⋮		
1 0 11 0 10100101011101	$\xrightarrow{s_2}$	1 0 11 0 10100101011101
⋮		

End of Proof

Simpler Proof:

Each Turing machine binary string is mapped to the number representing its value

Uncountable Sets

We will prove that there is a language L which is not accepted by any Turing machine

Technique:

Turing machines are countable

Languages are uncountable

(there are more languages than Turing Machines)

Theorem:

If S is an infinite countable set, then

the powerset 2^S of S is uncountable.

The powerset 2^S contains all possible subsets of S

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

Proof:

Since S is countable, we can list its elements in some order

$$S = \{s_1, s_2, s_3, \ldots\}$$

Elements of S

Elements of the powerset 2^S have the form:

$$\emptyset$$

$$\{s_1, s_3\}$$

$$\{s_5, s_7, s_9, s_{10}\}$$

$$\vdots$$

They are subsets of S

We encode each subset of S
with a binary string of 0's and 1's

	Binary encoding				
Subset of S	s_1	s_2	s_3	s_4	$\cdots$
$\{s_1\}$	1	0	0	0	$\cdots$
$\{s_2, s_3\}$	0	1	1	0	$\cdots$
$\{s_1, s_3, s_4\}$	1	0	1	1	$\cdots$

Every infinite binary string corresponds to a subset of S:

Example: $1\,0\,0\,1\,1\,1\,0\ \cdots$

Corresponds to: $\{s_1, s_4, s_5, s_6, \ldots\} \in 2^S$

Let's assume (for contradiction) that the powerset 2^S is countable

Then: we can list the elements of the powerset in some order

$$2^S = \{t_1, t_2, t_3, \ldots\}$$

Subsets of S

Powerset element	Binary encoding example					
t_1	1	0	0	0	0	$\cdots$
t_2	1	1	0	0	0	$\cdots$
t_3	1	1	0	1	0	$\cdots$
t_4	1	1	0	0	1	$\cdots$
$\cdots$			$\cdots$			

$\boldsymbol{t}$ = the binary string whose bits are the complement of the diagonal

$$\begin{array}{llllll} t_1 & 1 & 0 & 0 & 0 & 0 & \cdots \\ t_2 & 1 & 1 & 0 & 0 & 0 & \cdots \\ t_3 & 1 & 1 & 0 & 1 & 0 & \cdots \\ t_4 & 1 & 1 & 0 & 0 & 1 & \cdots \end{array}$$

Binary string: $\boldsymbol{t} = 0011\cdots$

(birary complement of diagonal)

The binary string $\quad t = 0011\ldots$

corresponds
to a subset of S: $\quad t = \{s_3, s_4, \ldots\} \in 2^S$

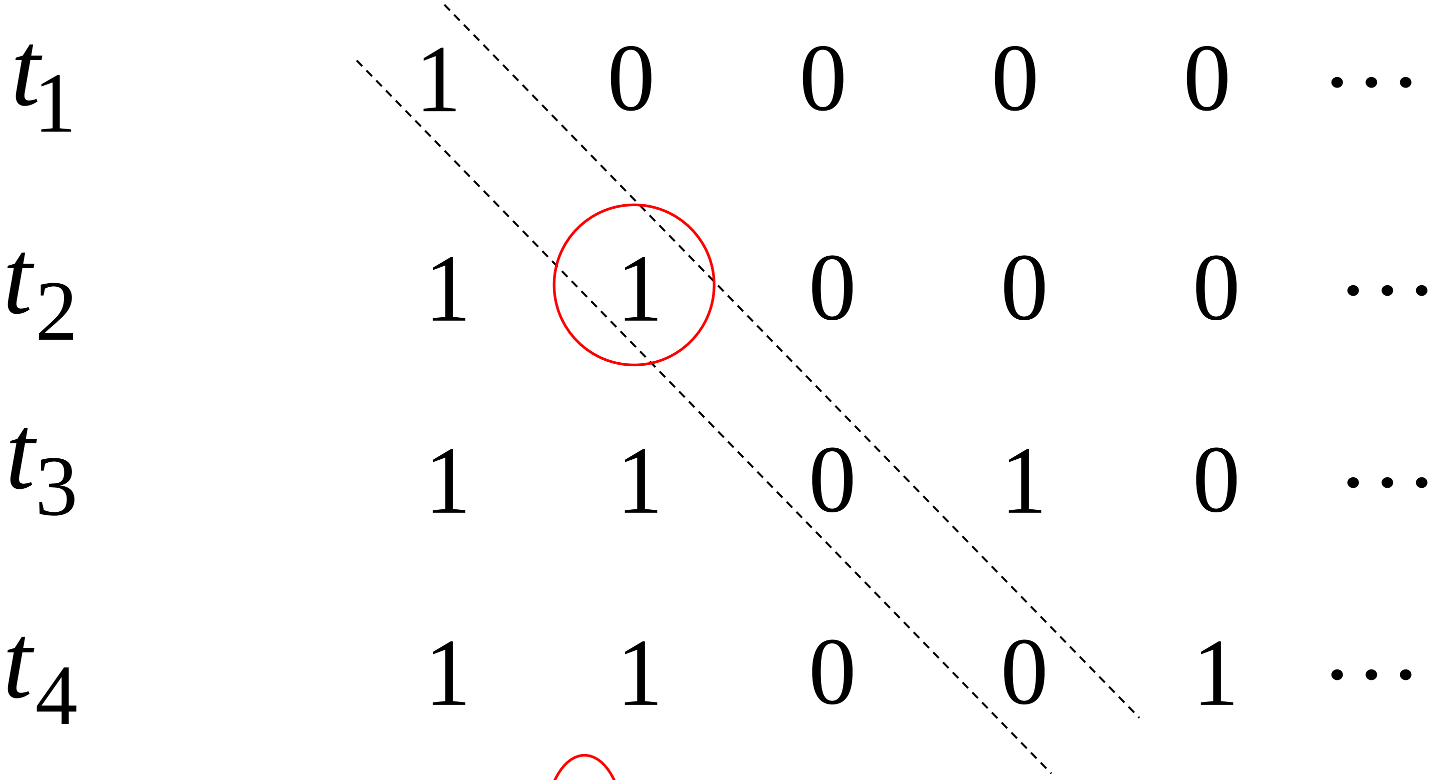
$t =$ the binary string whose bits are the complement of the diagonal

$$\begin{array}{lcccccc}
t_1 & 1 & 0 & 0 & 0 & 0 & \cdots \\
t_2 & 1 & 1 & 0 & 0 & 0 & \cdots \\
t_3 & 1 & 1 & 0 & 1 & 0 & \cdots \\
t_4 & 1 & 1 & 0 & 0 & 1 & \cdots
\end{array}$$

$$t = 0011\cdots$$

Question: $t = t_1$? **NO:** differ in 1st bit

$t =$ the binary string whose bits are the complement of the diagonal

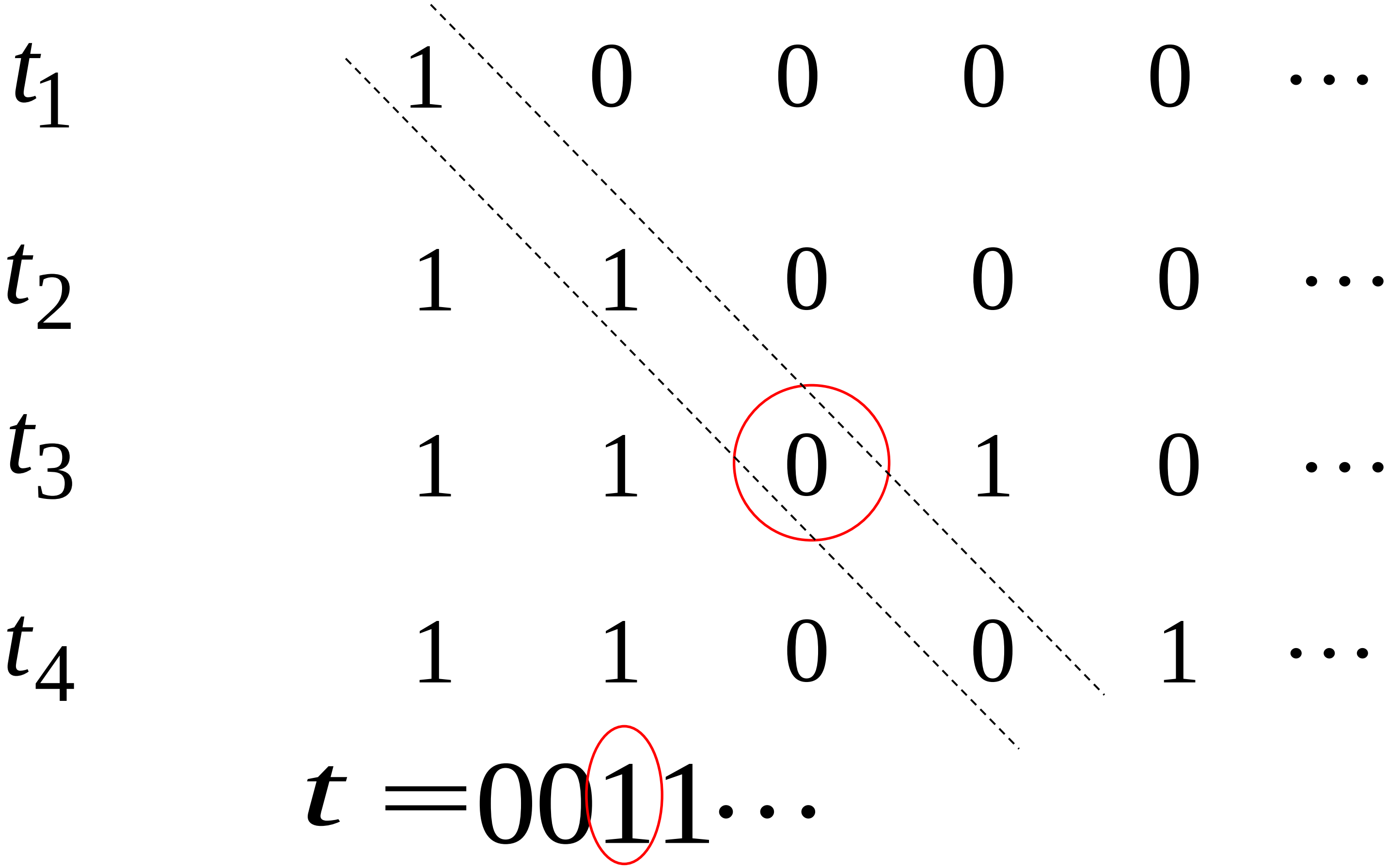


$$\begin{array}{c|cccccc} t_1 & 1 & 0 & 0 & 0 & 0 & \cdots \\ t_2 & 1 & 1 & 0 & 0 & 0 & \cdots \\ t_3 & 1 & 1 & 0 & 1 & 0 & \cdots \\ t_4 & 1 & 1 & 0 & 0 & 1 & \cdots \end{array}$$

$$t = 0011\cdots$$

Question: $t = t_2$? NO: differ in 2nd bit

$t =$ the binary string whose bits are the complement of the diagonal



$$\begin{array}{llllll} t_1 & 1 & 0 & 0 & 0 & 0 & \cdots \\ t_2 & 1 & 1 & 0 & 0 & 0 & \cdots \\ t_3 & 1 & 1 & 0 & 1 & 0 & \cdots \\ t_4 & 1 & 1 & 0 & 0 & 1 & \cdots \end{array}$$

$$t = 0011\cdots$$

Question: $t = t_3$? **NO:** differ in 3rd bit

Thus: $t \neq t_i$ for every i

since they differ in the ith bit

However, $t \in 2^S \Rightarrow t = t_i$ for some i

Contradiction!!!

Therefore the powerset 2^S is uncountable

End of proof

An Application: Languages

Consider Alphabet : $A = \{a, b\}$

The set of all strings:

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

infinite and countable

because we can enumerate
the strings in proper order

Consider Alphabet : $A = \{a, b\}$

The set of all strings:

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

infinite and countable

Any language is a subset of S :

$$L = \{aa, ab, aab\}$$

Consider Alphabet : $A = \{a, b\}$

The set of all Strings:

$$S = A^* = \{a, b\}^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}$$

infinite and countable

The powerset of S contains all languages:

$$2^S = \{\emptyset, \{\varepsilon\}, \{a\}, \{a, b\}, \{aa, b\}, \ldots, \{aa, ab, aab\}, \ldots\}$$

uncountable

Consider Alphabet : $A = \{a, b\}$

countable

Turing machines:

Languages accepted By Turing Machines:

$$\begin{tikzcd} M_1 \arrow[d, "\text{accepts}"'] & M_2 \arrow[d] & M_3 \arrow[d] & \cdots \\ L_1 & L_2 & L_3 & \cdots \end{tikzcd}$$

countable

Denote: $X = \{L_1, L_2, L_3, \ldots\}$

countable

Note: $X \subseteq 2^S$

$(S = \{a, b\}^*)$

Languages accepted
by Turing machines: X countable

All possible languages: 2^S uncountable

Therefore: $X \neq 2^S$

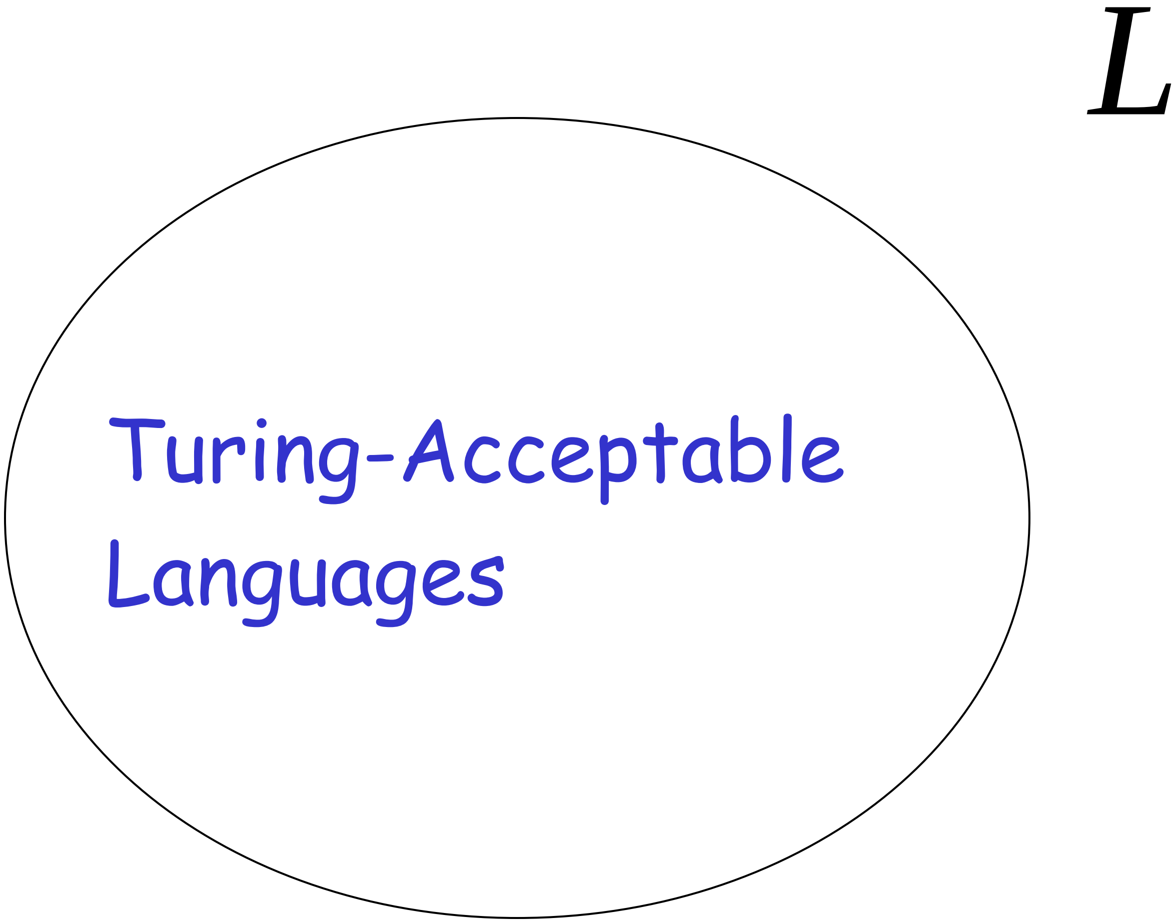
$$\left(\text{since } X \subseteq 2^S, \text{ we get } X \subset 2^S\right)$$

Conclusion:

There is a language L not accepted by any Turing Machine:

$$X \subset 2^{S} \quad \Longrightarrow \quad \exists L \in 2^{S} \text{ and } L \notin X$$

Non Turing-Acceptable Languages



Note that: $X = \{L_1, L_2, L_3, \ldots\}$

is a *multi-set* (elements may repeat) since a language may be accepted by more than one Turing machine

However, if we remove the repeated elements, the resulting set is again countable since every element still corresponds to a positive integer

Decidable Languages

Recall that:

A language L is **Turing-Acceptable**
if there is a Turing machine M
that accepts L

Also known as: *Turing-Recognizable*
or
Recursively-enumerable
languages

Turing-Acceptable

For any input string w :

$w \in L$ ⇒ M halts in an accept state

$w \notin L$ ⇒ M halts in a non-accept state

or loops forever

Definition:

A language L is **decidable**
if there is a Turing machine (decider) M
which accepts L
and halts on every input string

Also known as *recursive languages*

Turing-Decidable

For any input string w :

$w \in L$ ⟹ M halts in an accept state

$w \notin L$ ⟹ M halts in a non-accept state

Observation:

Every decidable language is Turing-Acceptable

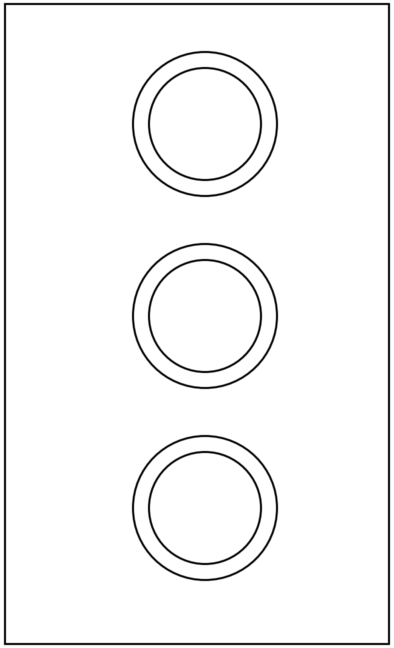
Sometimes, it is convenient to have Turing machines with single accept and reject states

These are the only halting states

That result to possible halting configurations

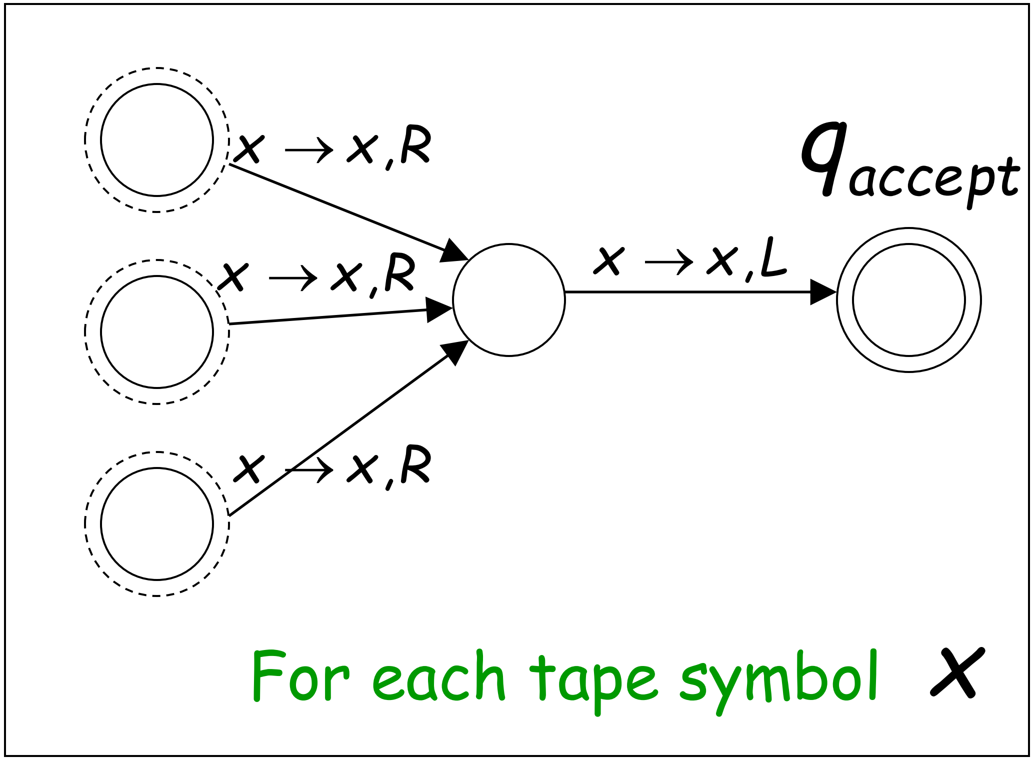
We can convert any Turing machine to have single accept and reject states

Old machine



Multiple accept states

New machine

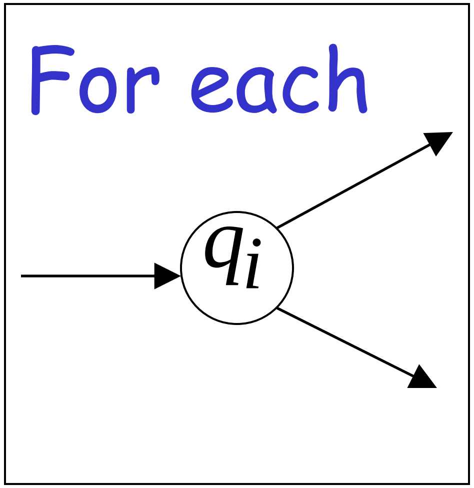


For each tape symbol x

One accept state

Do the following for each possible halting state:

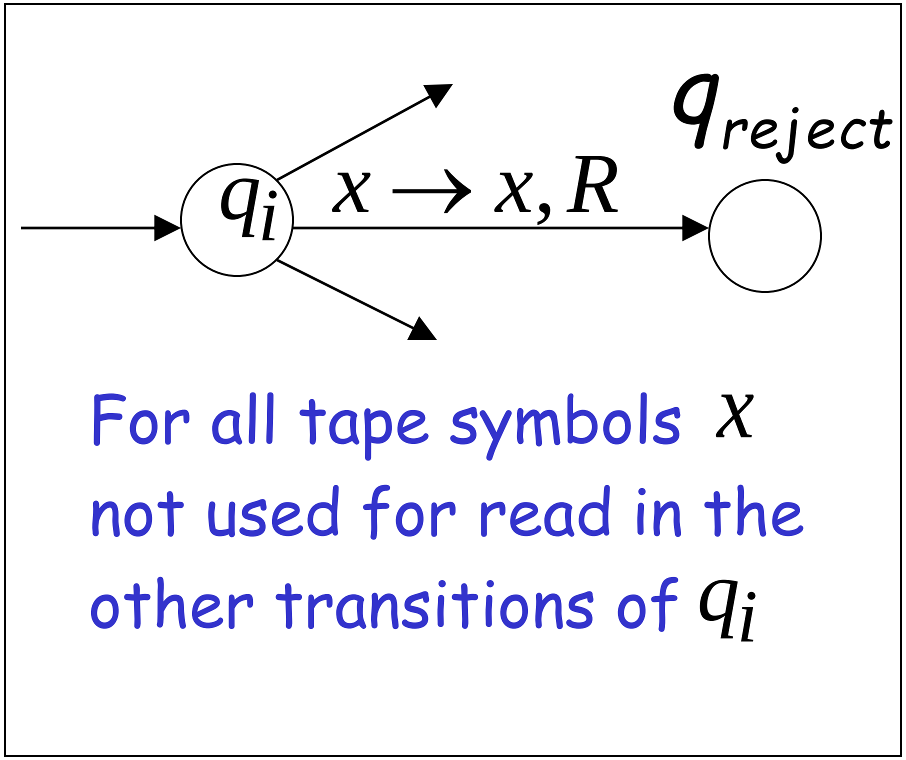
Old machine



For each

Multiple reject states

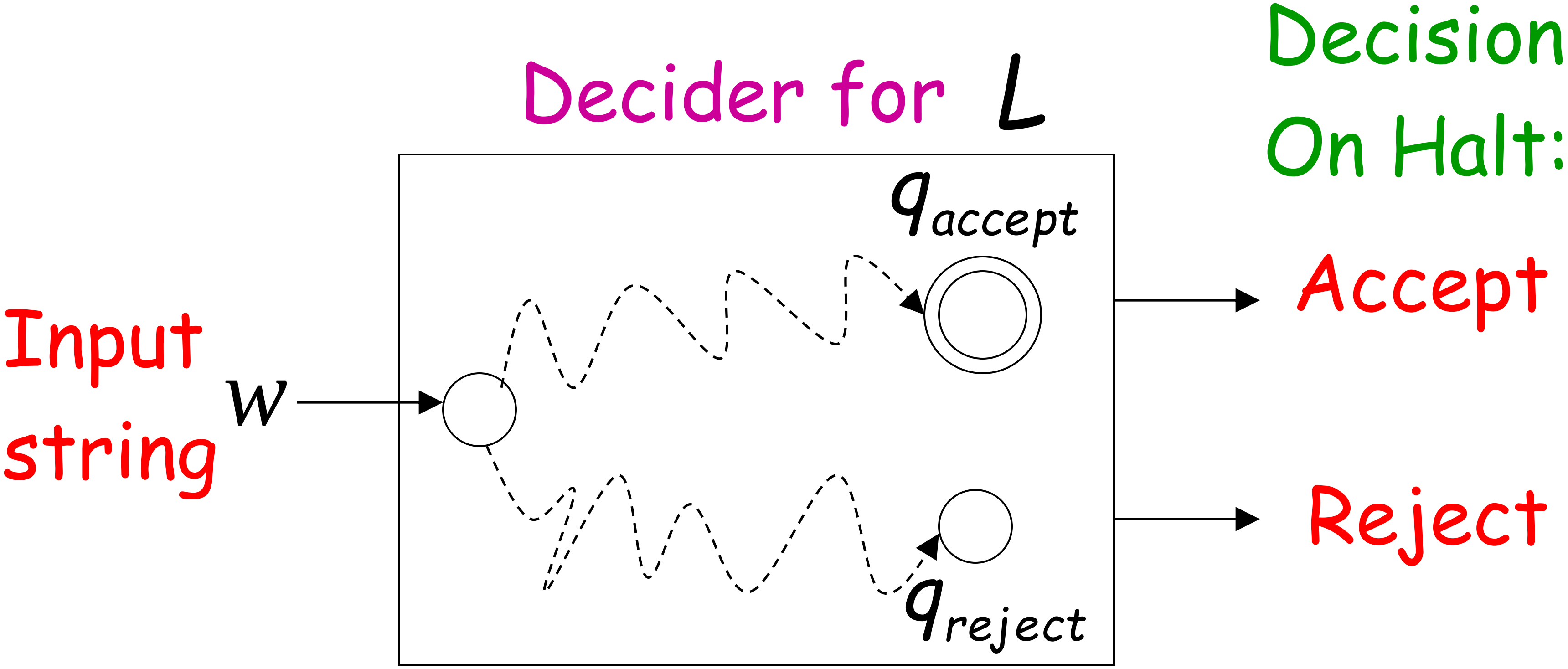
New machine



For all tape symbols x not used for read in the other transitions of q_i

One reject state

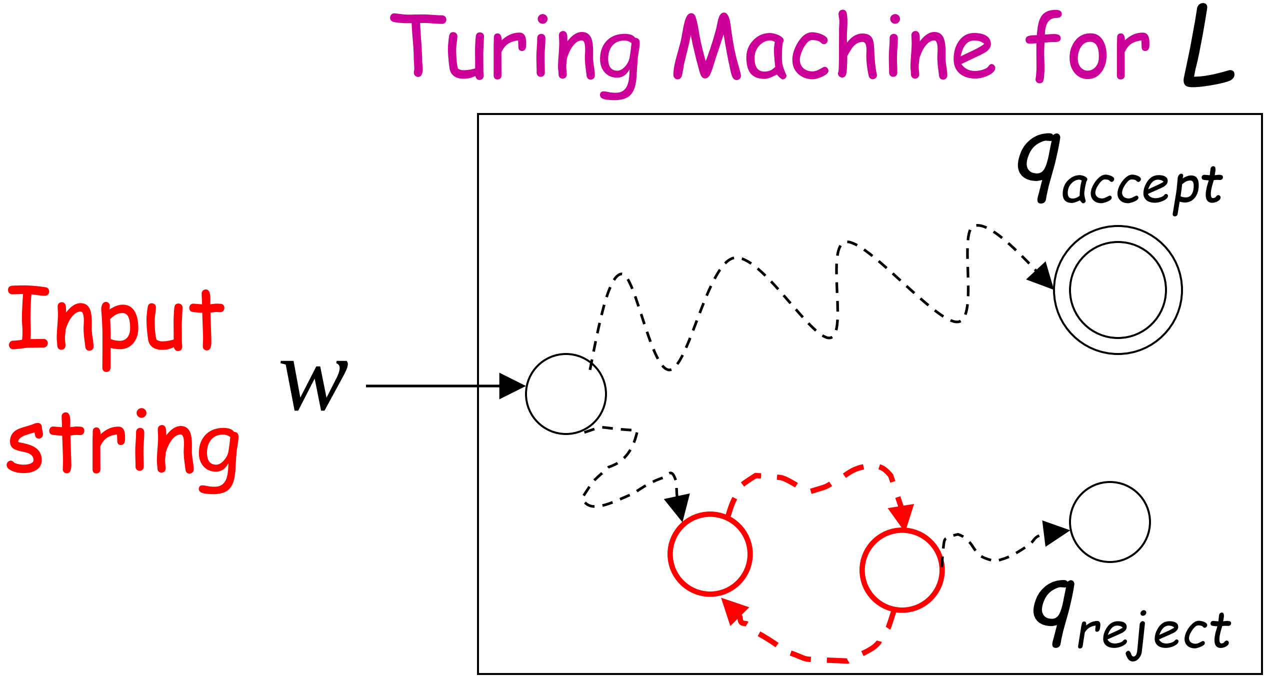
For a decidable language L :



For each input string, the computation halts in the accept or reject state

For a Turing-Acceptable language L :

Turing Machine for L



It is possible that for some input string the machine enters an infinite loop

A computational problem is decidable
if the corresponding language is decidable

We also say that the problem is solvable

Problem: Is number x prime?

Corresponding language:

$$PRIMES = \{1, 2, 3, 5, 7, \ldots\}$$

We will show it is decidable

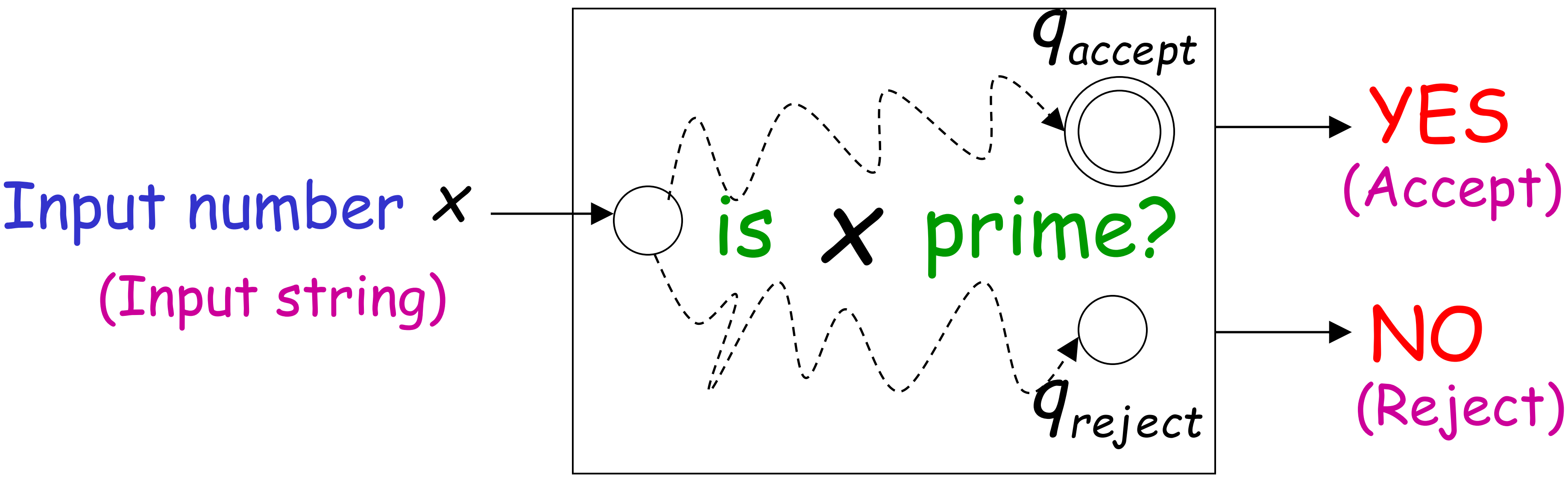
Decider for *PRIMES* :

```
On input number $x$ :
    Divide $x$ with all possible numbers
    between $2$ and $\sqrt{x}$

    If any of them divides $x$
    Then reject
    Else accept
```

the decider Turing machine can be designed based on the algorithm

Decider for *PRIMES*



Problem: Does DFA M accept the empty language $L(M) = \emptyset$?

Corresponding Language: (Decidable)

$$EMPTY_{DFA} =$$

$$\{\langle M \rangle : M \text{ is a DFA that accepts empty language } \emptyset\}$$

Description of DFA M as a string

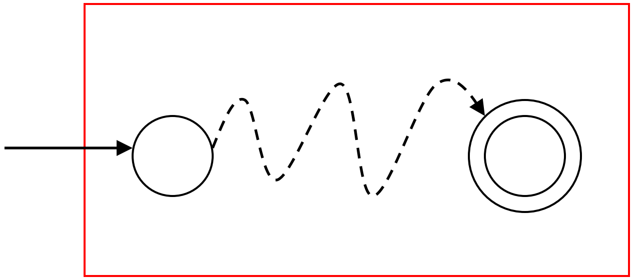
(For example, we can represent M as a binary string, as we did for Turing machines)

Decider for $EMPTY_{DFA}$:

On input $\langle M \rangle$:

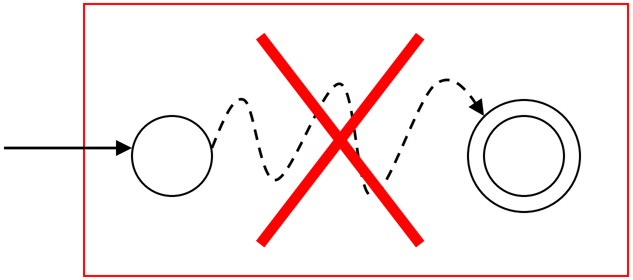
Determine whether there is a path from the initial state to any accepting state

DFA M



$L(M) \neq \varnothing$

DFA M



$L(M) = \varnothing$

Decision: Reject $\langle M \rangle$ Accept $\langle M \rangle$

Problem: Does DFA M accept a finite language?

Corresponding Language: (Decidable)

$$FINITE_{DFA} =$$

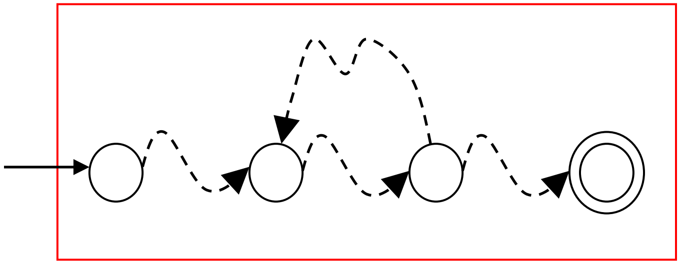
$$\{\langle M \rangle : M \text{ is a DFA that accepts a finite language}\}$$

Decider for $FINITE_{DFA}$:

On input $\langle M \rangle$:

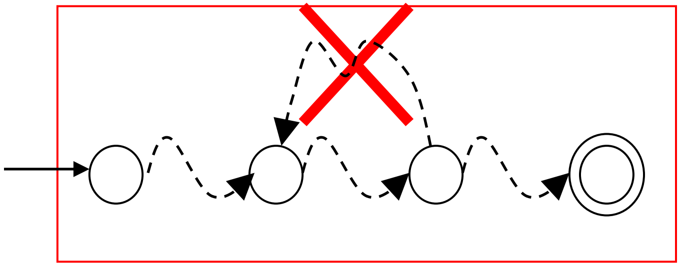
Check if there is a walk with a cycle from the initial state to an accepting state

DFA M



infinite

DFA M



finite

Decision: Reject $\langle M \rangle$ (NO) Accept $\langle M \rangle$ (YES)

Problem: Does DFA M accept string w ?

Corresponding Language: (Decidable)

$$A_{DFA} =$$

$$\{\langle M, w \rangle : M \text{ is a DFA that accepts string } w\}$$

Decider for A_{DFA} :

```
On input string $\langle M,w \rangle$:

    Run DFA $M$ on input string $w$

    If $M$ accepts $w$
    Then accept $\langle M,w \rangle$ (and halt)
    Else reject $\langle M,w \rangle$ (and halt)
```

Problem: Do DFAs M_1 and M_2 accept the same language?

Corresponding Language: (Decidable)

$$EQUAL_{DFA} =$$

$$\{\langle M_1, M_2\rangle : M_1 \text{ and } M_2 \text{ are DFAs that accept the same languages}\}$$

Decider for $EQUAL_{DFA}$:

On input $\langle M_1, M_2 \rangle$:

Let L_1 be the language of DFA M_1

Let L_2 be the language of DFA M_2

Construct DFA M such that:

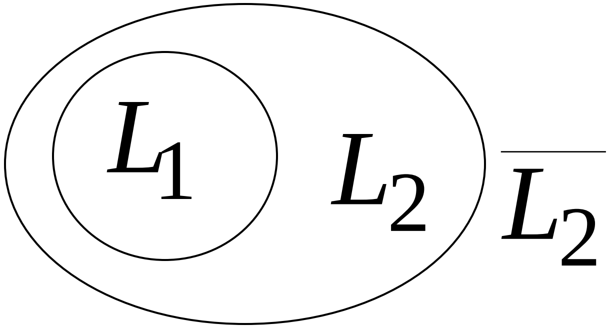
$$L(M) = (L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2)$$

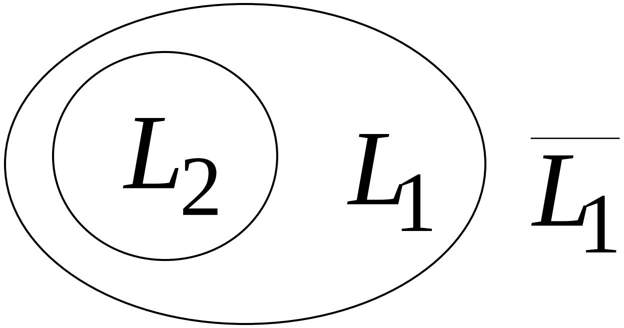
(combination of DFAs)

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

$$\Downarrow$$

$L_1 \cap \overline{L_2} = \emptyset$ **and** $\overline{L_1} \cap L_2 = \emptyset$





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

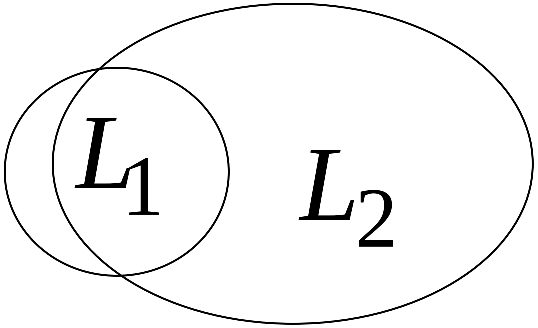
$$\Downarrow$$

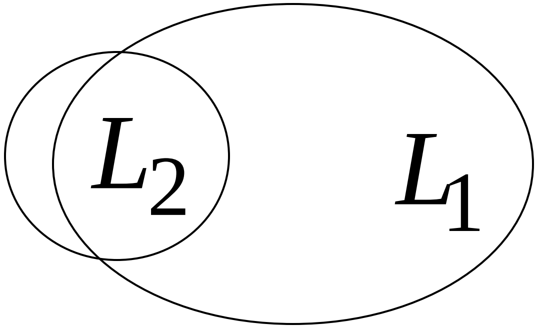
$$L_1 = L_2$$

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

↓

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





$L_1 \not\subset L_2$

$L_2 \not\subset L_1$

↓

$$L_1 \neq L_2$$

Therefore, we only need to determine whether

$$L(M) = (L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2) = \emptyset$$

which is a solvable problem for DFAs:

$EMPTY_{DFA}$

Theorem:

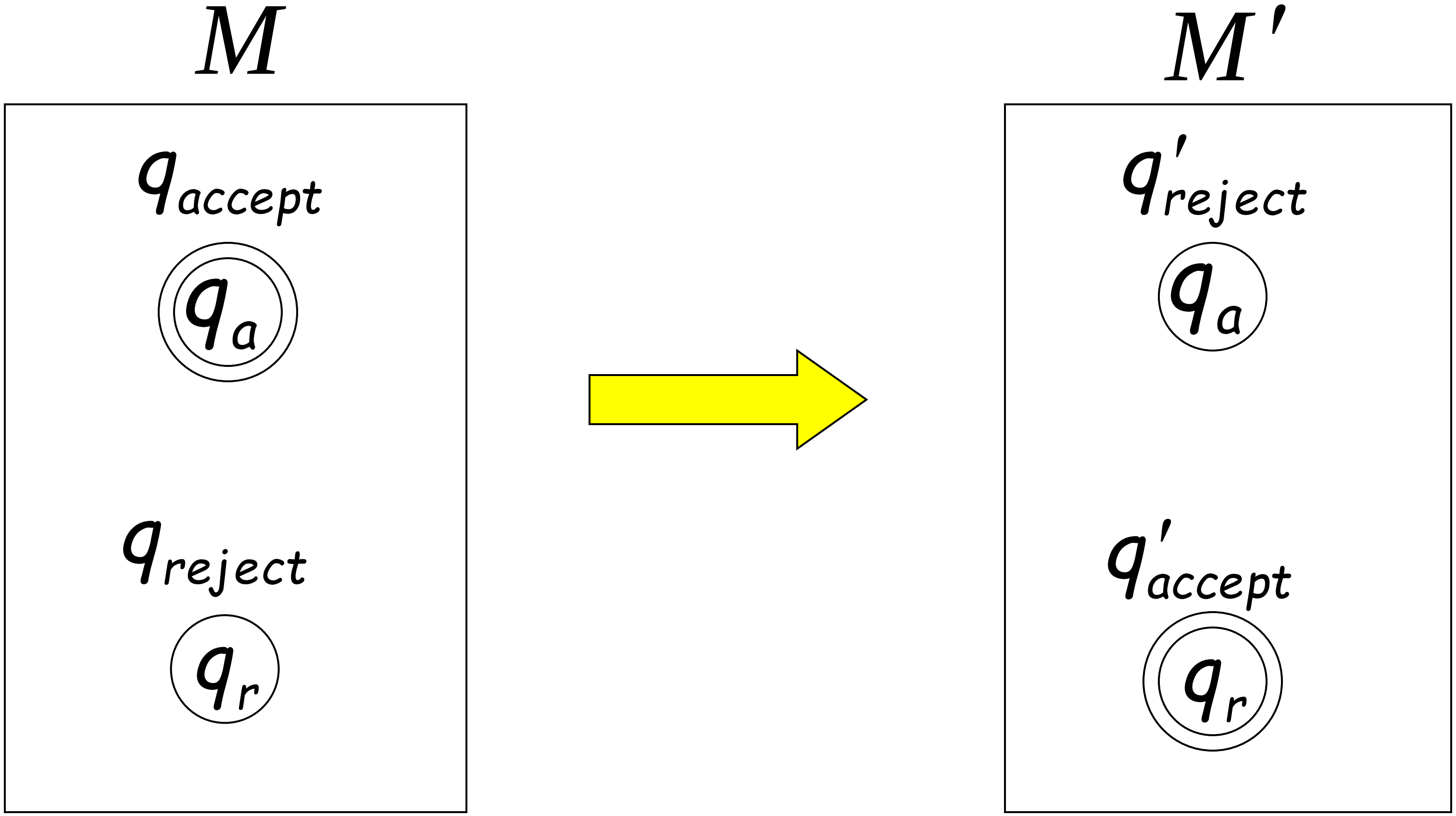
If a language L is decidable, then its complement $\overline{L}$ is decidable too

Proof:

Build a Turing machine M' that accepts $\overline{L}$ and halts on every input string

(M' is decider for $\overline{L}$)

Transform accept state to reject and vice-versa



Turing Machine M'

On each input string w do:

1. Let M be the decider for L

2. Run M with input string w

   If M accepts then reject

   If M rejects then accept

Accepts $\overline{L}$ and halts on every input string

END OF PROOF

Undecidable Languages

Undecidable Languages

An undecidable language has no decider:

Any Turing machine that accepts L does not halt on some input string

We will show that:

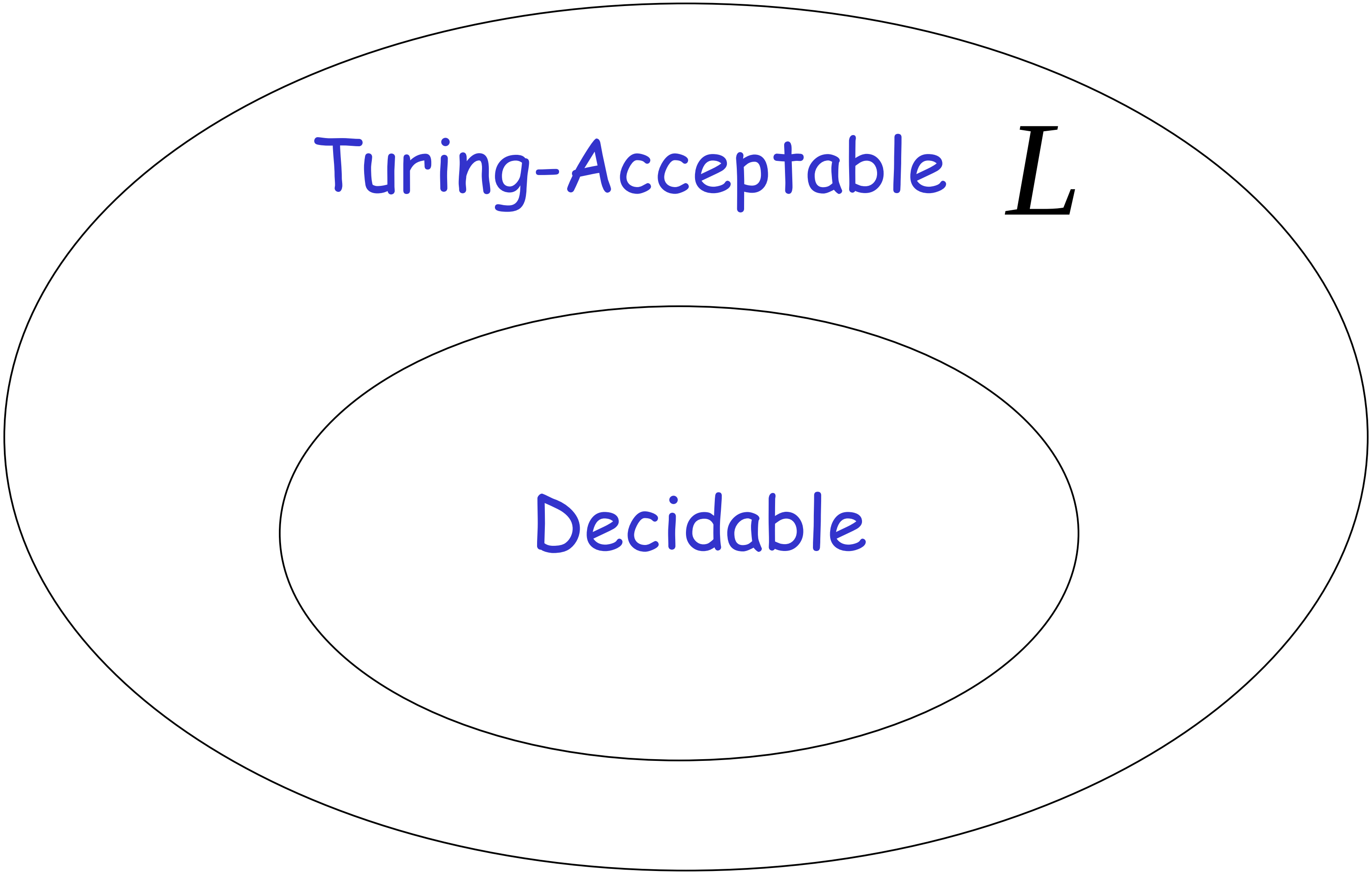
There is a language which is Turing-Acceptable and undecidable

We will prove that there is a language L:

- L is Turing-acceptable
- $\overline{L}$ is **not** Turing-acceptable
  (not accepted by any Turing Machine)

the complement of a decidable language is decidable

Therefore, L is undecidable



Non Turing-Acceptable $\overline{L}$

Turing-Acceptable L

Decidable

Consider alphabet $\{a\}$

Strings of $\{a\}^+$:

$$a, \ aa, \ aaa, \ aaaa, \ \ldots$$

$$a^1 \ a^2 \quad a^3 \quad\quad a^4 \quad \ldots$$

Consider Turing Machines
that accept languages over alphabet $\{a\}$

They are countable:

$$M_1, \; M_2, \; M_3, \; M_4, \; \ldots$$

(There is an enumerator that generates them)

Each machine accepts some language over $\{a\}$

$$\begin{tikzcd}
M_1, \arrow[d] & M_2, \arrow[d] & M_3, \arrow[d] & M_4, \arrow[d] & \ldots \\
L(M_1), & L(M_2), & L(M_3), & L(M_4), & \ldots
\end{tikzcd}$$

Note that it is possible to have

$$L(M_i) = L(M_j) \quad \text{for} \quad i \neq j$$

Since, a language could be accepted by more than one Turing machine

Example language accepted by M_i

$$L(M_i) = \{aa, aaaa, aaaaaa\}$$

$$L(M_i) = \{a^2, a^4, a^6\}$$

Binary representation

	a^1	a^2	a^3	a^4	a^5	a^6	a^7	$\cdots$
$L(M_i)$	0	1	0	1	0	1	0	$\cdots$

Example of binary representations

	a^1	a^2	a^3	a^4	$\cdots$
$L(M_1)$	0	1	0	1	$\cdots$
$L(M_2)$	1	0	0	1	$\cdots$
$L(M_3)$	0	1	1	1	$\cdots$
$L(M_4)$	0	0	0	1	$\cdots$

Consider the language

$$L = \{a^i : a^i \in L(M_i)\}$$

L consists of the **1's** in the diagonal

$$\begin{array}{c|ccccc}
 & a^1 & a^2 & a^3 & a^4 & \cdots \\
\hline
L(M_1) & 0 & 1 & 0 & 1 & \cdots \\
L(M_2) & 1 & 0 & 0 & 1 & \cdots \\
L(M_3) & 0 & 1 & \enclose{circle}{1} & 1 & \cdots \\
L(M_4) & 0 & 0 & 0 & \enclose{circle}{1} & \cdots
\end{array}$$

$$L = \{a^3, a^4, \ldots\}$$

Consider the language $\overline{L}$

$$\overline{L} = \{a^i : a^i \notin L(M_i)\}$$

$$L = \{a^i : a^i \in L(M_i)\}$$

$\overline{L}$ consists of the **0's** in the diagonal

$$
\begin{array}{c|ccccc}
 & a^1 & a^2 & a^3 & a^4 & \cdots \\
\hline
L(M_1) & \textcircled{0} & 1 & 0 & 1 & \cdots \\
L(M_2) & 1 & \textcircled{0} & 0 & 1 & \cdots \\
L(M_3) & 0 & 1 & 1 & 1 & \cdots \\
L(M_4) & 0 & 0 & 0 & 1 & \cdots
\end{array}
$$

$$\overline{L} = \{a^1, a^2, \ldots\}$$

Theorem:

Language $\overline{L}$ is not Turing-Acceptable

Proof:

Assume for contradiction that

$\overline{L}$ is Turing-Acceptable

Let M_k be the Turing machine that accepts $\overline{L}$: $L(M_k) = \overline{L}$

	a^1	a^2	a^3	a^4	$\cdots$
$L(M_1)$	0	1	0	1	$\cdots$
$L(M_2)$	1	0	0	1	$\cdots$
$L(M_3)$	0	1	1	1	$\cdots$
$L(M_4)$	0	0	0	1	$\cdots$

$$L(M_k) = \overline{L} = 1100\cdots$$

Question: $M_k = M_1$?

$$\begin{array}{c|ccccc}
 & a^1 & a^2 & a^3 & a^4 & \cdots \\
\hline
L(M_1) & 0 & 1 & 0 & 1 & \cdots \\
L(M_2) & 1 & 0 & 0 & 1 & \cdots \\
L(M_3) & 0 & 1 & 1 & 1 & \cdots \\
L(M_4) & 0 & 0 & 0 & 1 & \cdots
\end{array}$$

$$L(M_k) = \overline{L} = 1100 \cdots$$

$$a^1 \in L(M_k)$$

Answer: $M_k \neq M_1$

$$a^1 \notin L(M_1)$$

	a^1	a^2	a^3	a^4	$\cdots$
$L(M_1)$	0	1	0	1	$\cdots$
$L(M_2)$	1	0	0	1	$\cdots$
$L(M_3)$	0	1	1	1	$\cdots$
$L(M_4)$	0	0	0	1	$\cdots$

$$L(M_k) = \overline{L} = 1100\cdots$$

Question: $M_k = M_2$?

	a^1	a^2	a^3	a^4	$\cdots$
$L(M_1)$	0	1	0	1	$\cdots$
$L(M_2)$	1	0	0	1	$\cdots$
$L(M_3)$	0	1	1	1	$\cdots$
$L(M_4)$	0	0	0	1	$\cdots$

$$L(M_k) = \overline{L} = 1100\cdots$$

$$a^2 \in L(M_k)$$

Answer: $M_k \neq M_2$

$$a^2 \notin L(M_2)$$

$$\begin{array}{c|ccccc} & a^1 & a^2 & a^3 & a^4 & \cdots \\ \hline L(M_1) & 0 & 1 & 0 & 1 & \cdots \\ L(M_2) & 1 & 0 & 0 & 1 & \cdots \\ L(M_3) & 0 & 1 & 1 & 1 & \cdots \\ L(M_4) & 0 & 0 & 0 & 1 & \cdots \end{array}$$

$$L(M_k) = \overline{L} = 1100\cdots$$

Question: $M_k = M_3$?

	a^1	a^2	a^3	a^4	$\cdots$
$L(M_1)$	0	1	0	1	$\cdots$
$L(M_2)$	1	0	0	1	$\cdots$
$L(M_3)$	0	1	1	1	$\cdots$
$L(M_4)$	0	0	0	1	$\cdots$

$$L(M_k) = \overline{L} = 1100\cdots$$

$$a^3 \notin L(M_k)$$

Answer: $M_k \neq M_3$

$$a^3 \in L(M_3)$$

Similarly: $M_k \neq M_i$ for any i

Because either:

$a^i \in L(M_k)$

$a^i \notin L(M_i)$

or

$a^i \notin L(M_k)$

$a^i \in L(M_i)$

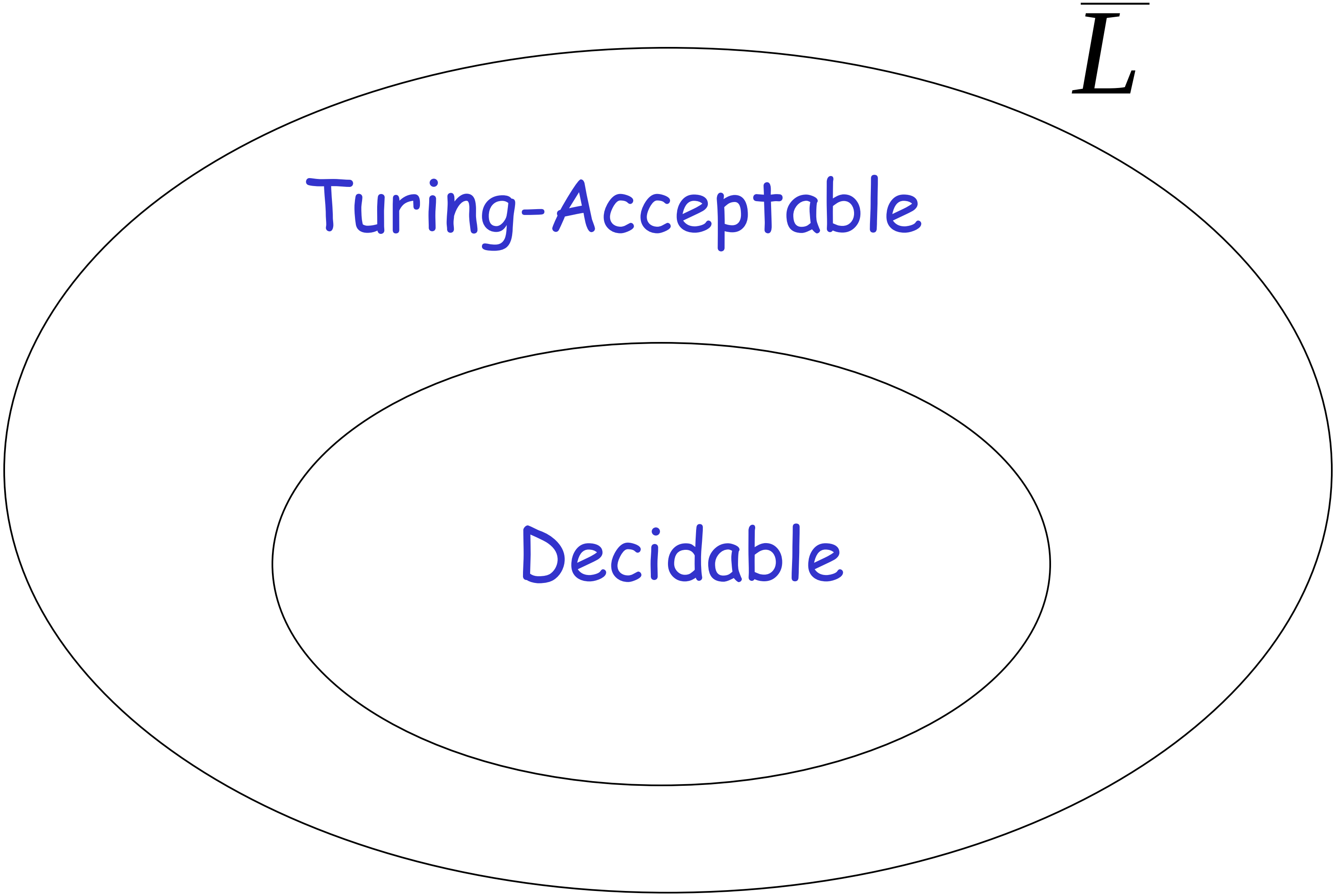
⇓

the machine M_k cannot exist

⇓

$\overline{L}$ is not Turing-Acceptable

End of Proof



Non Turing-Acceptable

$\overline{L}$

Turing-Acceptable

Decidable

We will prove that the language

$$L = \{a^i : a^i \in L(M_i)\}$$

Is Turing-Acceptable

There is a
Turing machine
that accepts L

Undecidable

Each machine
that accepts L
doesn't halt
on some input string

Theorem: The language

$$L = \{a^i : a^i \in L(M_i)\}$$

Is Turing-Acceptable

Proof: We will give a Turing Machine that accepts L

Turing Machine that accepts L

For any input string w

- Suppose $w = a^i$
- Find Turing machine M_i

  (using the enumerator for Turing Machines)
- Simulate M_i on input string a^i
- If M_i accepts, then accept w

End of Proof

Therefore:

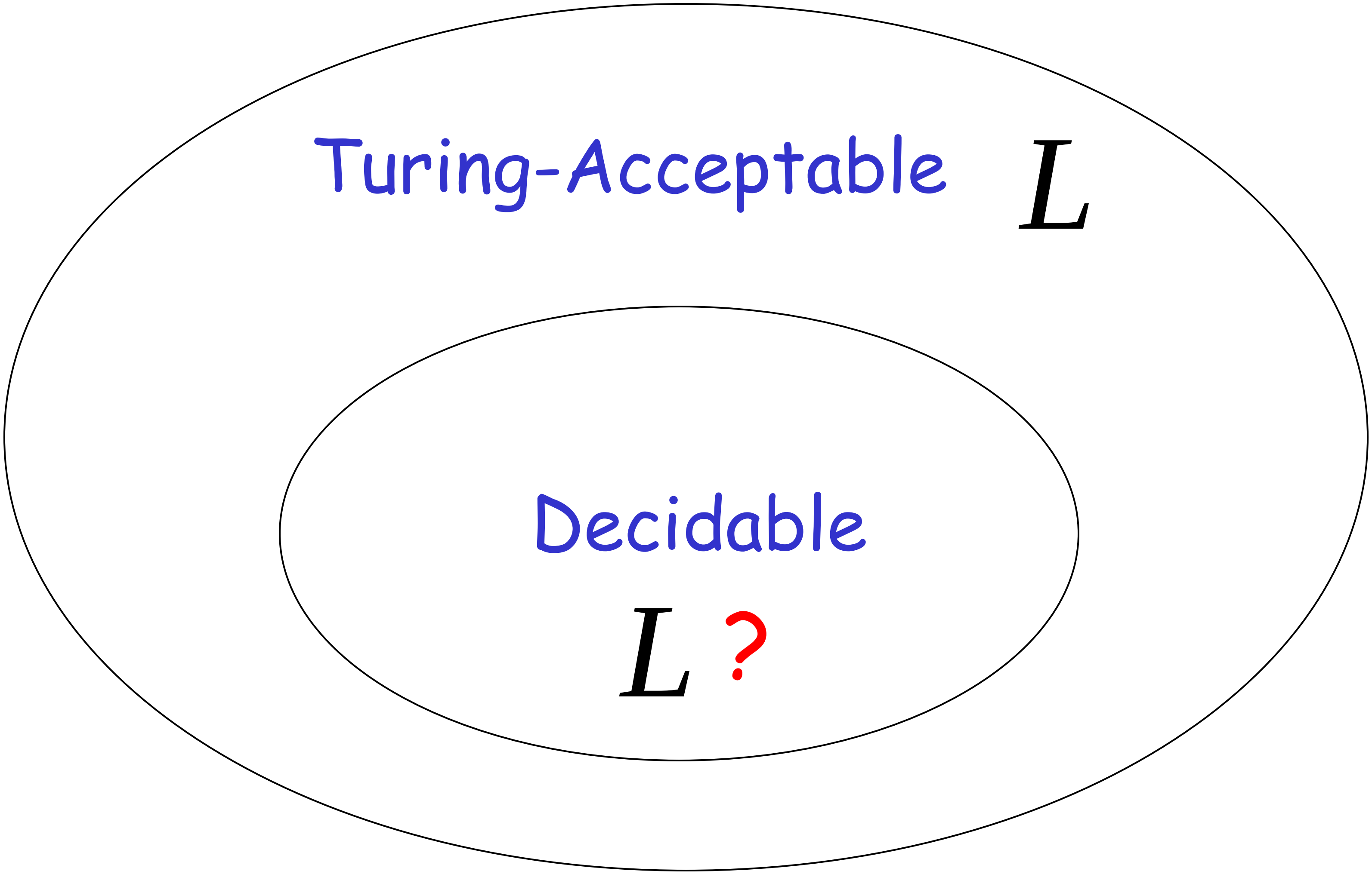
Turing-Acceptable

$$L = \{a^i : a^i \in L(M_i)\}$$

Not Turing-acceptable

$$\overline{L} = \{a^i : a^i \notin L(M_i)\}$$

Non Turing-Acceptable $\overline{L}$



Turing-Acceptable L

Decidable

L ?

Theorem: $L = \{a^i : a^i \in L(M_i)\}$

is undecidable

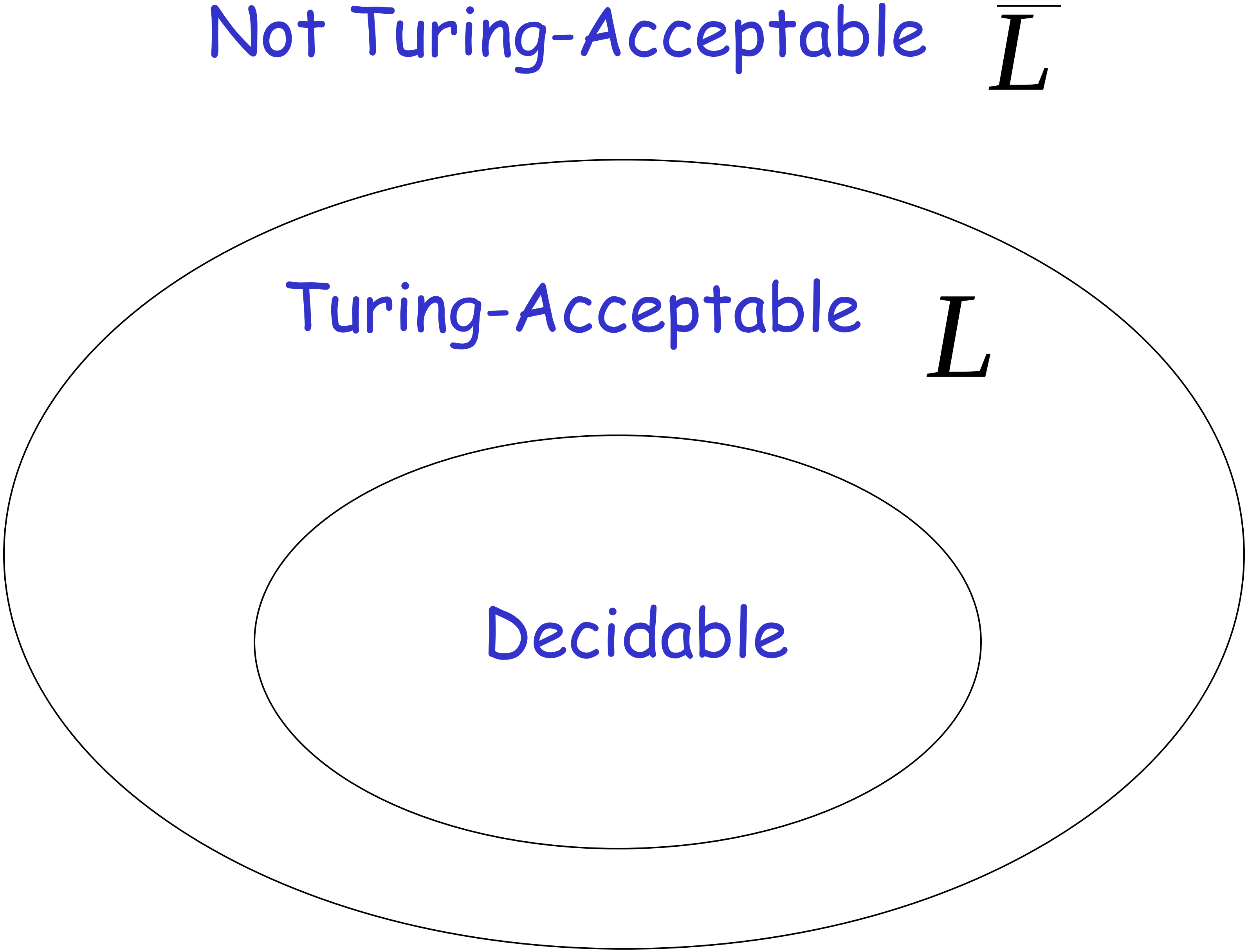
Proof: If L is decidable

the complement of a decidable language is decidable

Then $\overline{L}$ is decidable

However, $\overline{L}$ is not Turing-Acceptable!

Contradiction!!!!



Not Turing-Acceptable $\overline{L}$

Turing-Acceptable L

Decidable

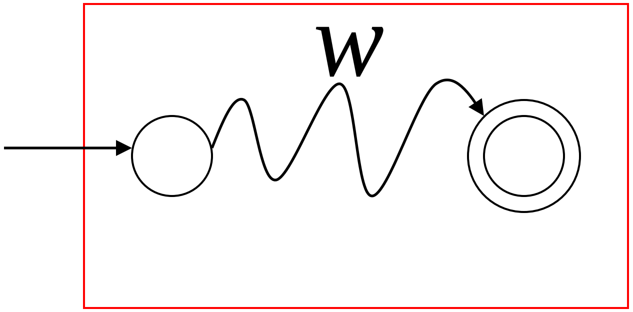
Decidable Problems
of
Regular Languages

Membership Question

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

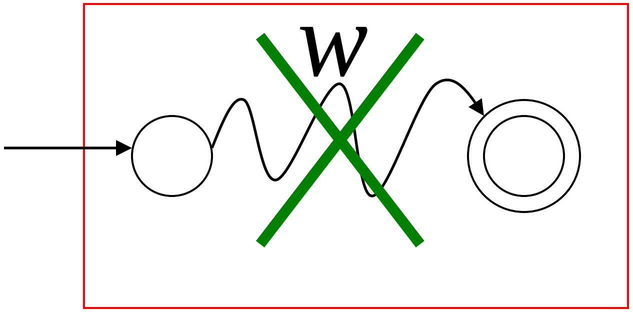
Answer: Take the DFA that accepts L
and check if w is accepted

DFA



$w \in L$

DFA



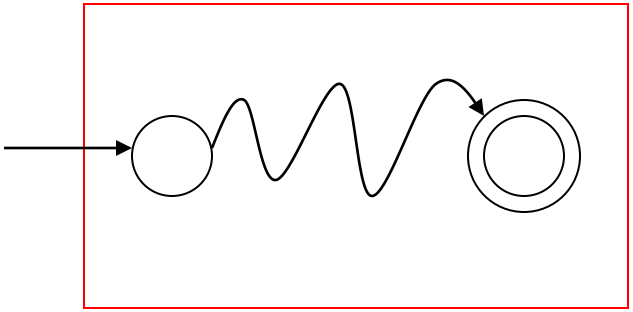
$w \notin L$

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

Answer: Take the DFA that accepts L

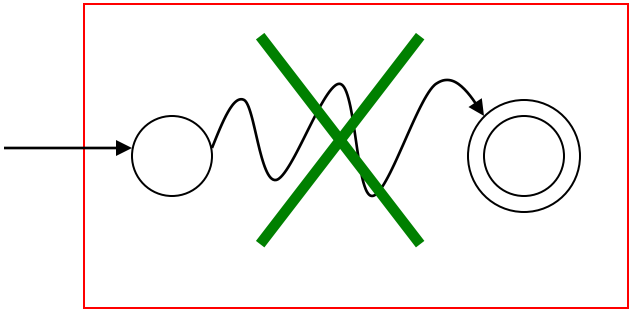
Check if there is any path from
the initial state to an accepting state

DFA



$L \neq \emptyset$

DFA



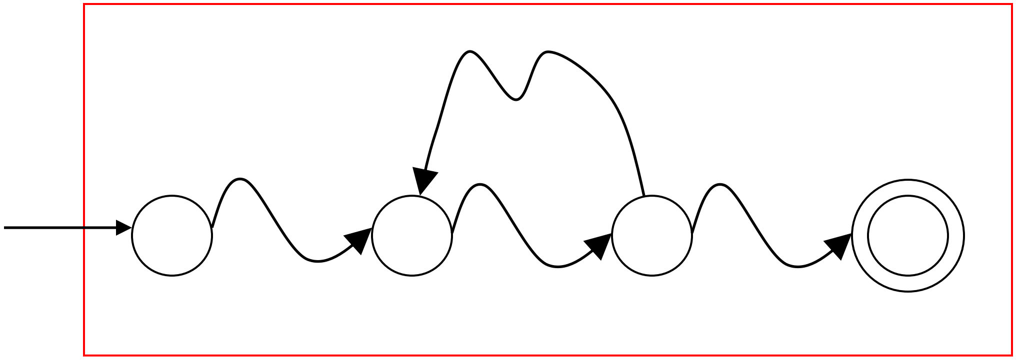
$L = \emptyset$

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

Answer: Take the DFA that accepts L

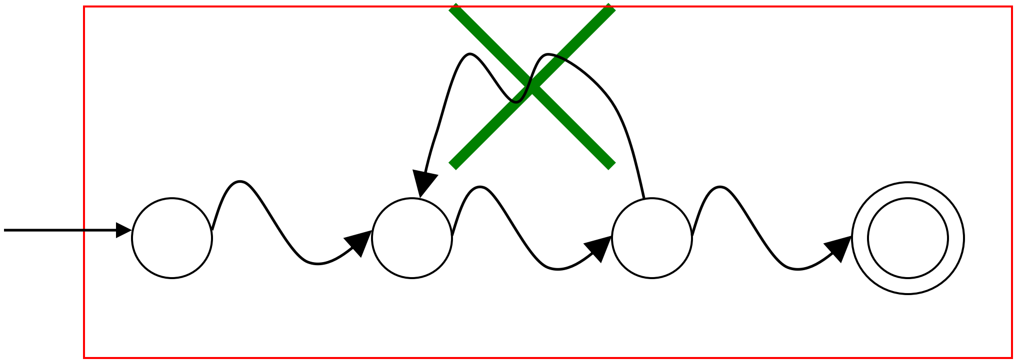
Check if there is a walk with cycle from the initial state to a final state

DFA



L is infinite

DFA



L is finite

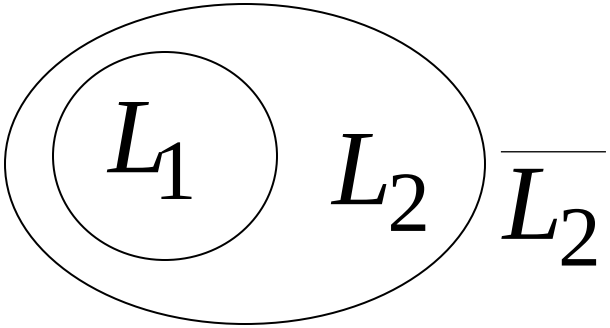
Question: Given regular languages L_1 and L_2 how can we check if $L_1 = L_2$?

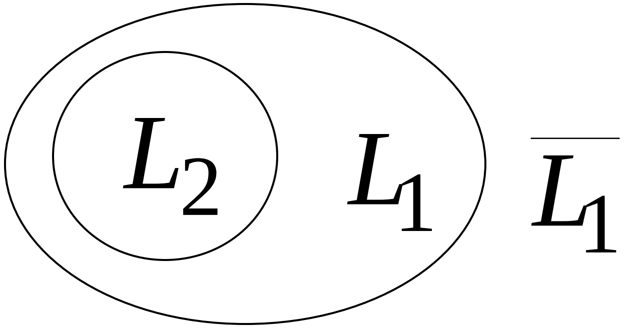
Answer: Find if $(L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2) = \emptyset$

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

$$\Downarrow$$

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





$L_1 \subseteq L_2$

$L_2 \subseteq L_1$

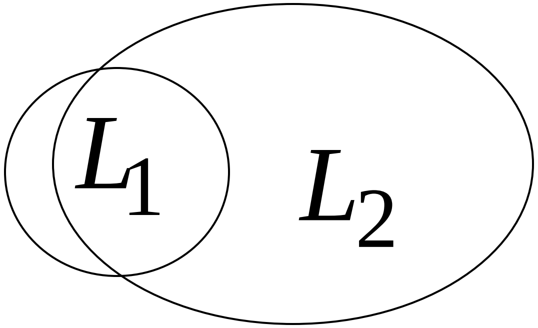
$$\Downarrow$$

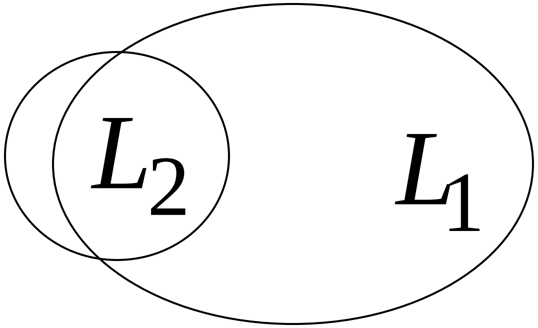
$$L_1 = L_2$$

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

⬇

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





$L_1 \not\subset L_2$ $L_2 \not\subset L_1$

⬇

$$L_1 \neq L_2$$

Decidable Problems of Context-Free Languages

Membership Question:

for context-free grammar G

find if string $w \in L(G)$

Membership Algorithms: Parsers

- Exhaustive search parser
- **CYK** parsing algorithm

Empty Language Question:

for context-free grammar G

find if $L(G) = \emptyset$

Algorithm:

1. Remove useless variables

2. Check if start variable S is useless

Infinite Language Question:

for context-free grammar G

find if $L(G)$ is infinite

Algorithm:

1. Remove useless variables

2. Remove unit and λ productions

3. Create dependency graph for variables

4. If there is a loop in the dependency graph then the language is infinite

Example:

$$S \to AB$$

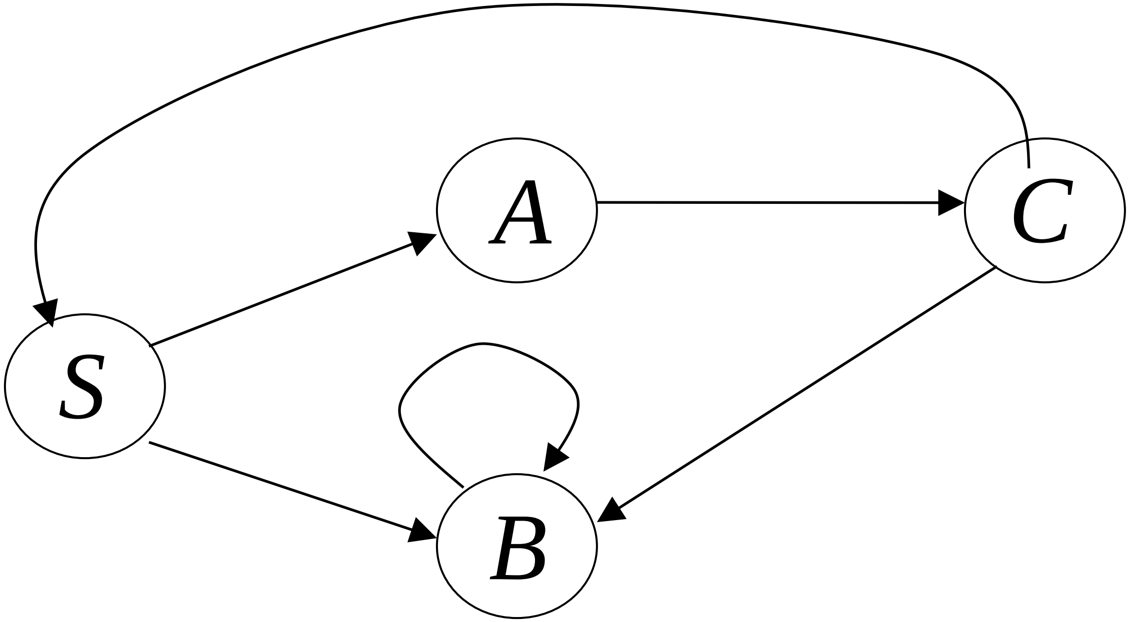
$$A \to aCb \mid a$$

$$B \to bB \mid bb$$

$$C \to cBS$$

Dependency graph

Infinite language



$$S \rightarrow AB$$

$$A \rightarrow aCb \mid a$$

$$B \rightarrow bB \mid bb$$

$$C \rightarrow cBS$$

$$S \Rightarrow AB \Rightarrow aCbB \Rightarrow acBSbB \Rightarrow acbbSbbb$$

$$S \overset{*}{\Rightarrow} acbbSbbb \overset{*}{\Rightarrow} (acbb)^2 S (bbb)^2$$

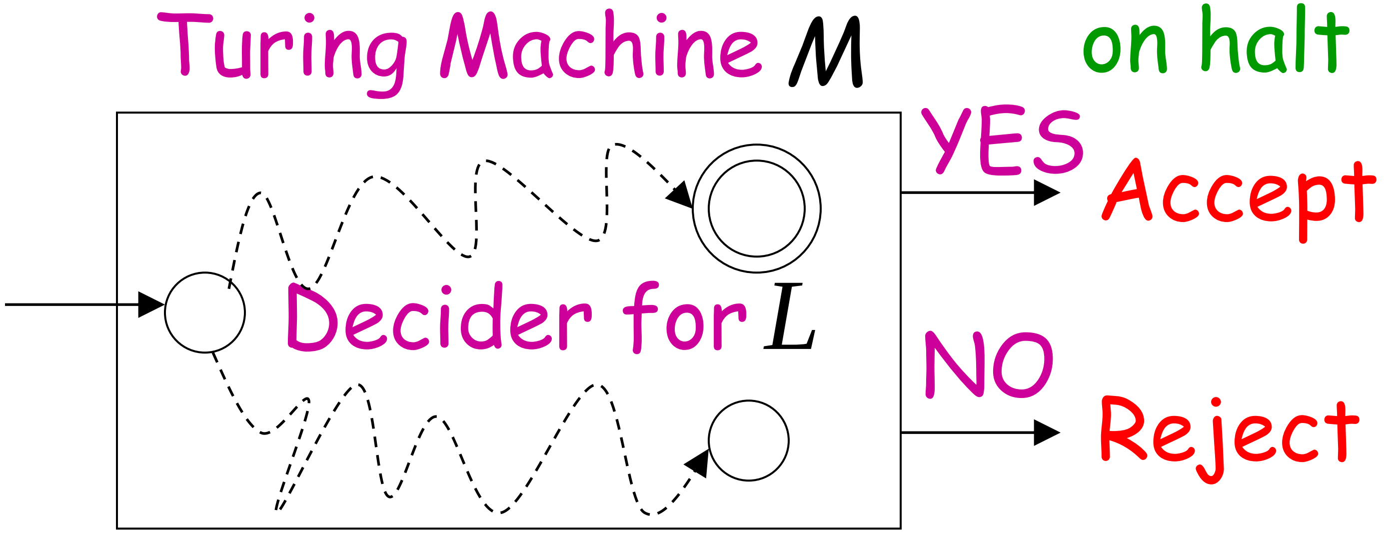
$$\overset{*}{\Rightarrow} (acbb)^i S (bbb)^i$$

Undecidable Problems

Recall that:

A language L is decidable,
if there is a Turing machine M (decider)
that accepts L and halts on every input string

Decision
on halt



Undecidable Language L

There is no decider for L:

there is no Turing Machine
which accepts L
and halts on every input string

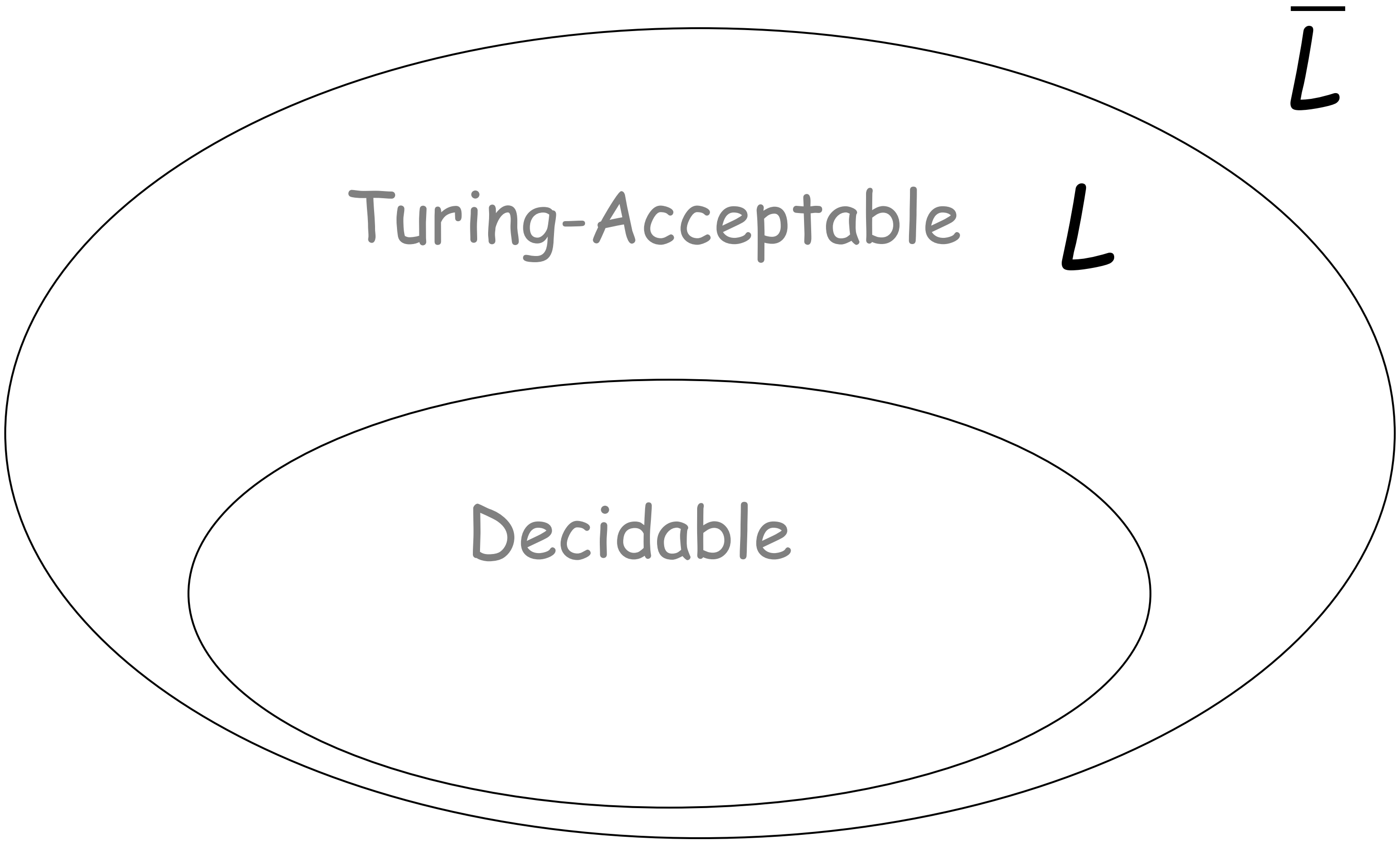
(the machine may halt and decide for some input strings)

For an undecidable language,
the corresponding problem is
undecidable (unsolvable):

there is no Turing Machine (Algorithm)
that gives an answer (yes or no)
for every input instance

(answer may be given for some input instances)

We have shown before that there are undecidable languages:



L is Turing-Acceptable and undecidable

We will prove that two particular problems are unsolvable:

Membership problem

Halting problem

Membership Problem

Input:
- Turing Machine M
- String w

Question: Does M accept w ?

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

Corresponding language:

$$A_{TM} = \{\langle M, w \rangle : M \text{ is a Turing machine that accepts string } w\}$$

Theorem: A_{TM} is undecidable

(The membership problem is unsolvable)

Proof:

Basic idea:

We will assume that A_{TM} is decidable;

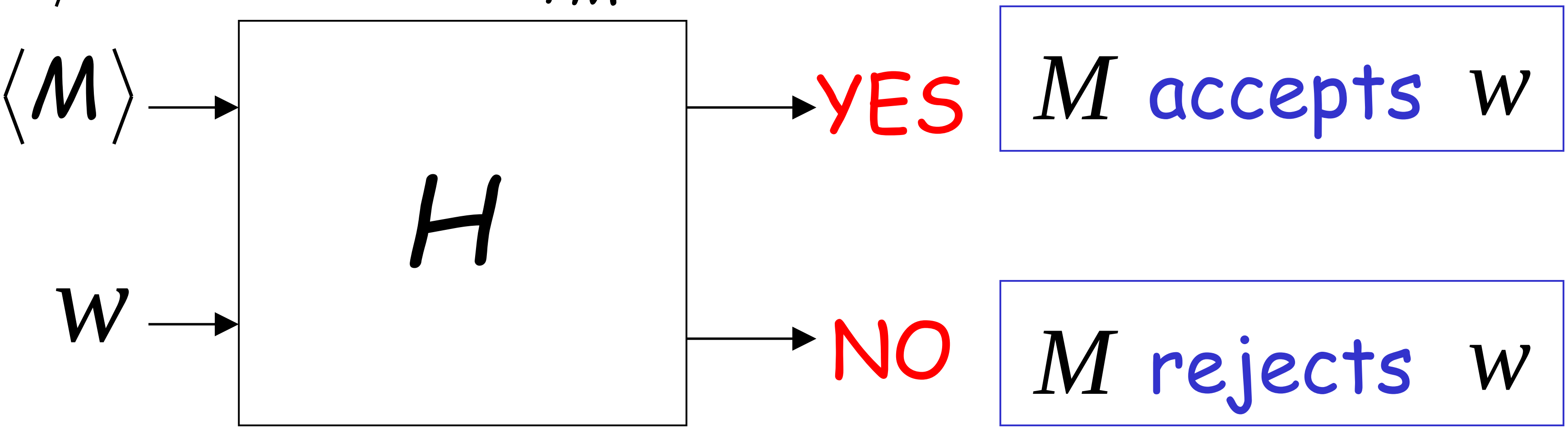
We will then prove that

every Turing-acceptable language

is also decidable

A contradiction!

Suppose that A_{TM} is decidable



Input string $\langle M,w \rangle$

Decider for A_{TM}

$\langle M \rangle$

w

H

YES → M accepts w

NO → M rejects w

Let L be a Turing recognizable language

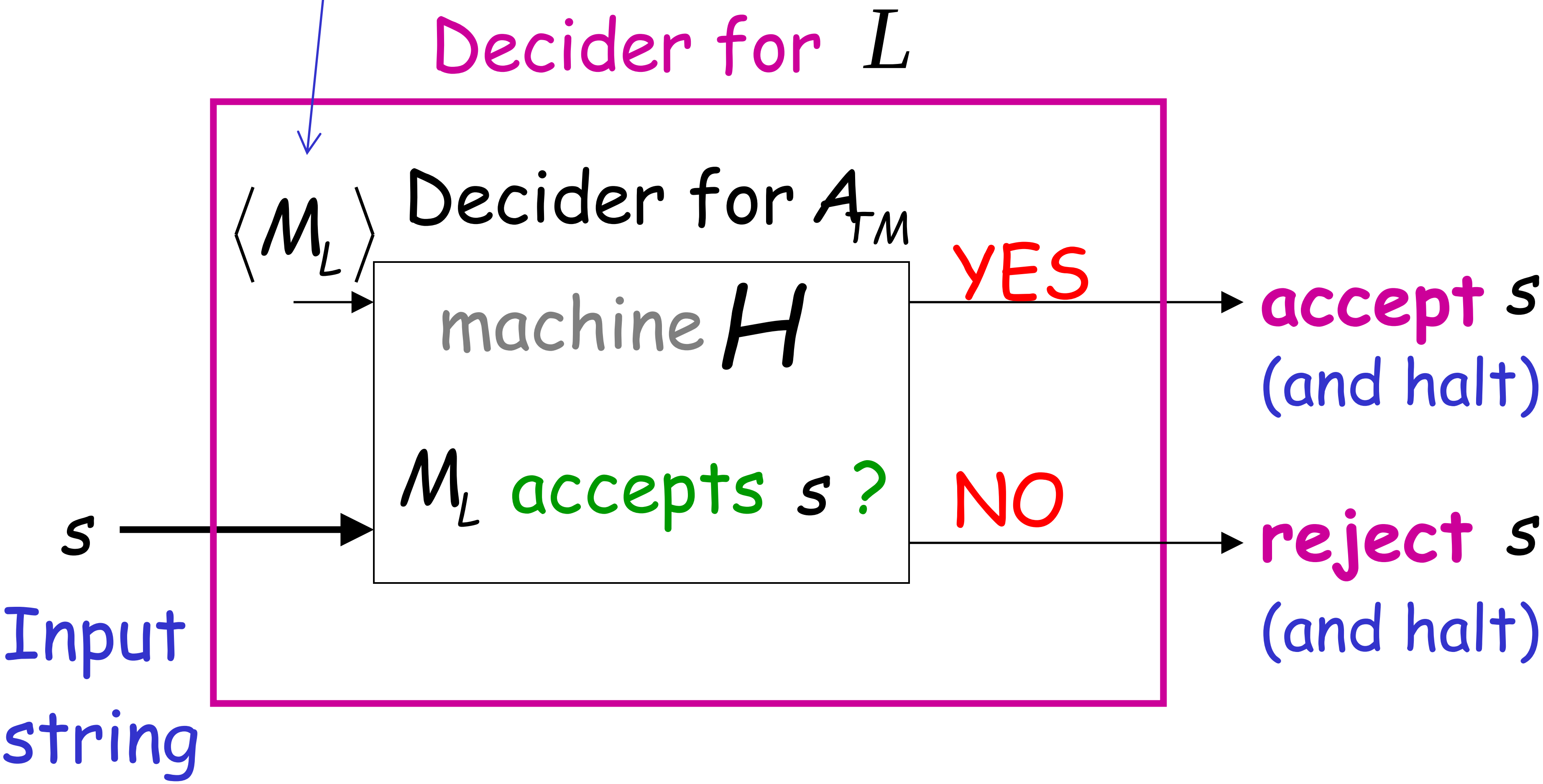
Let M_L be the Turing Machine that accepts L

We will prove that L is also decidable:

we will build a decider for L

String description of M_L

This is hardwired and copied on the tape next to input string **s**, and then the pair $\langle M_L, s \rangle$ is input to H



Decider for L

$\langle M_L \rangle$

Decider for A_{TM}

machine H

M_L accepts s ?

YES

NO

accept s
(and halt)

reject s
(and halt)

s

Input string

Therefore, L is decidable

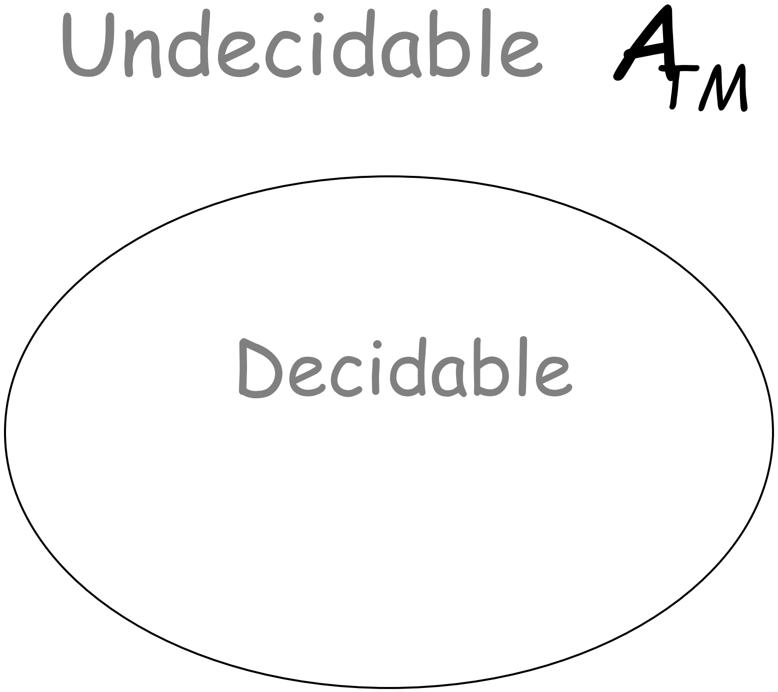
Since L is chosen arbitrarily, every Turing-Acceptable language is decidable

But there is a Turing-Acceptable language which is undecidable

Contradiction!!!!

END OF PROOF

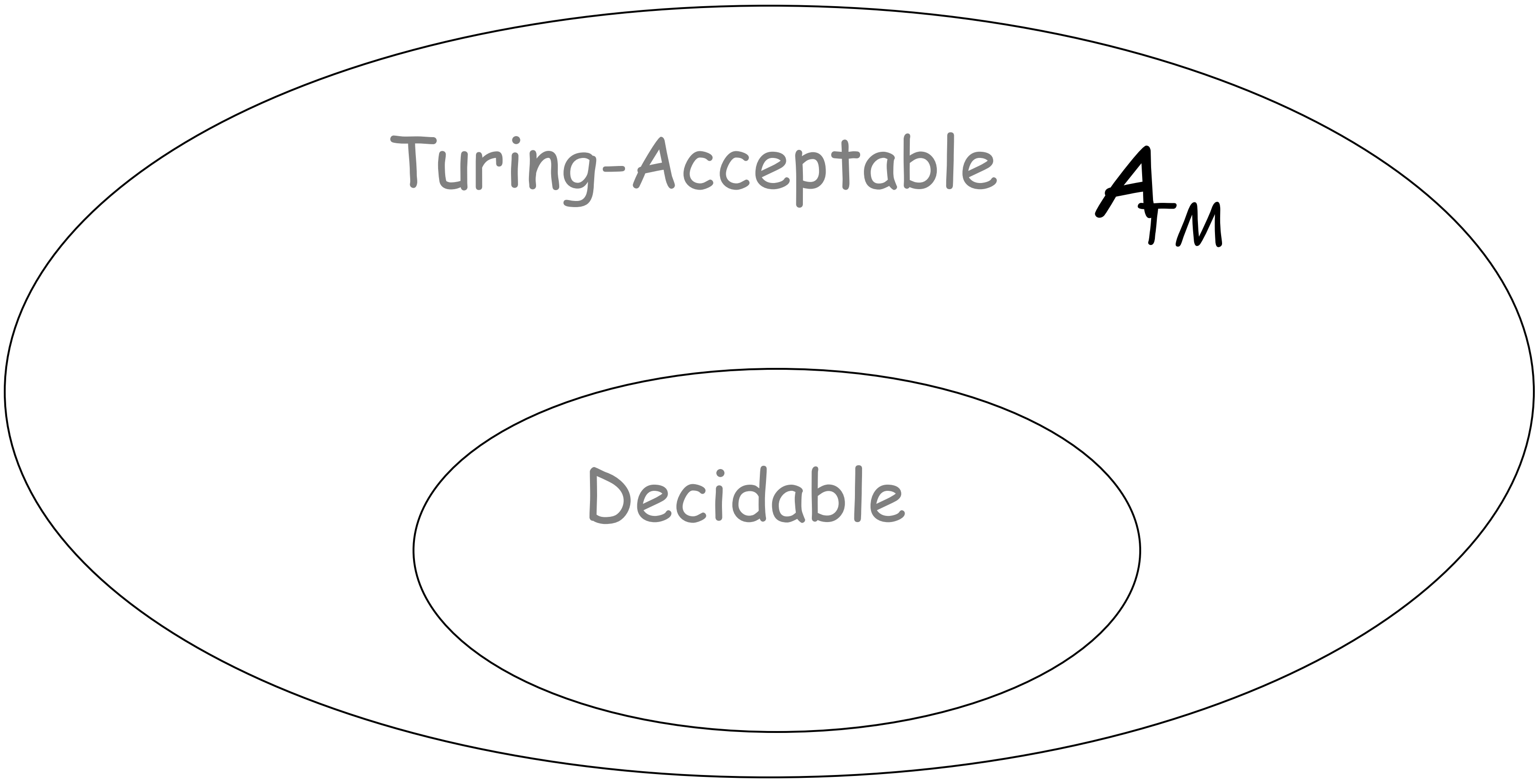
We have shown:



Undecidable A_{TM}

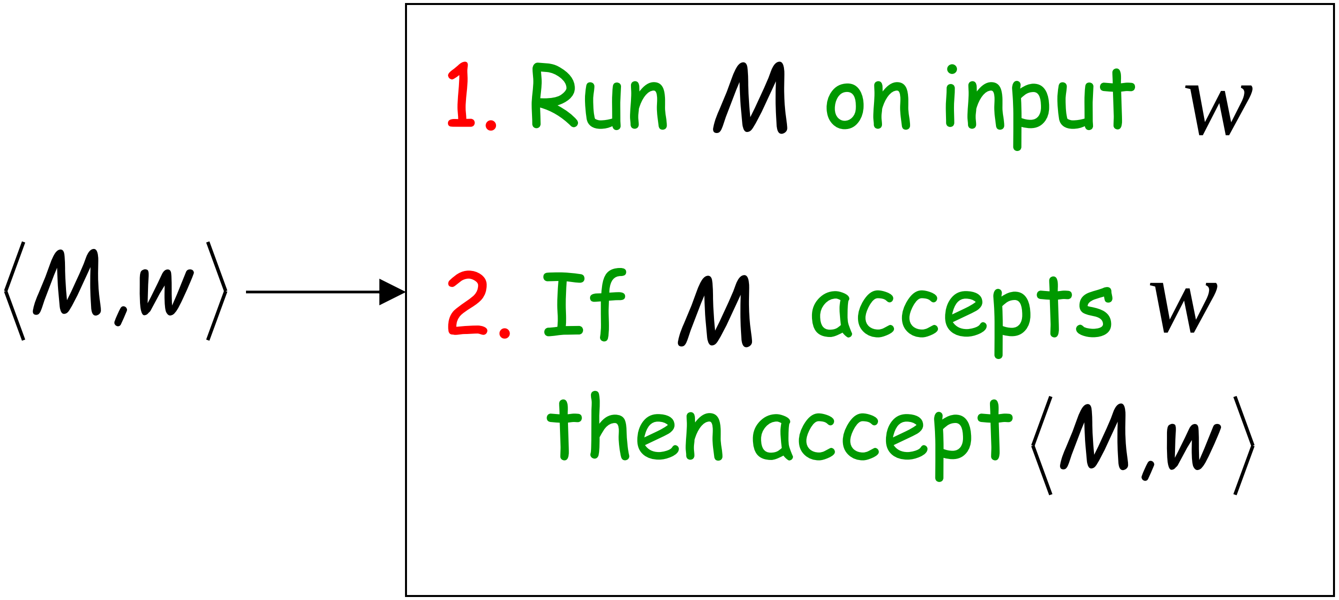
Decidable

We can actually show:



A_{TM} is Turing-Acceptable

Turing machine that accepts A_{TM} :



$\langle M,w \rangle$

1. Run M on input w

2. If M accepts w
   then accept $\langle M,w \rangle$

Halting Problem

Input:
- Turing Machine M
- String w

Question: Does M halt while processing input string w ?

Corresponding language:

$$HALT_{TM} = \{\langle M, w \rangle : M \text{ is a Turing machine that halts on input string } w\}$$

Theorem: $HALT_{TM}$ is undecidable

(The halting problem is unsolvable)

Proof:

Basic idea:

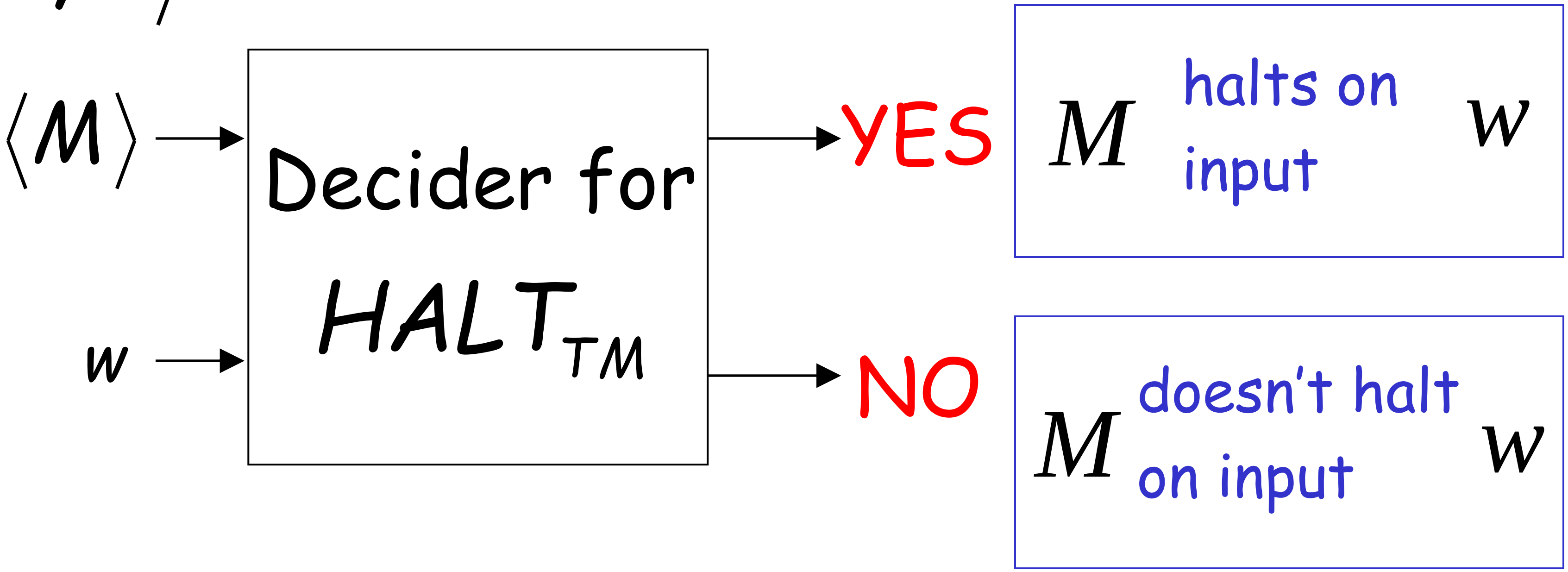
Suppose that $HALT_{TM}$ is decidable;
we will prove that
every Turing-acceptable language
is also decidable

A contradiction!

Suppose that $HALT_{TM}$ is decidable

Input string

$\langle M,w \rangle$



$\langle M \rangle$

w

Decider for $HALT_{TM}$

YES

NO

M halts on input w

M doesn't halt on input w

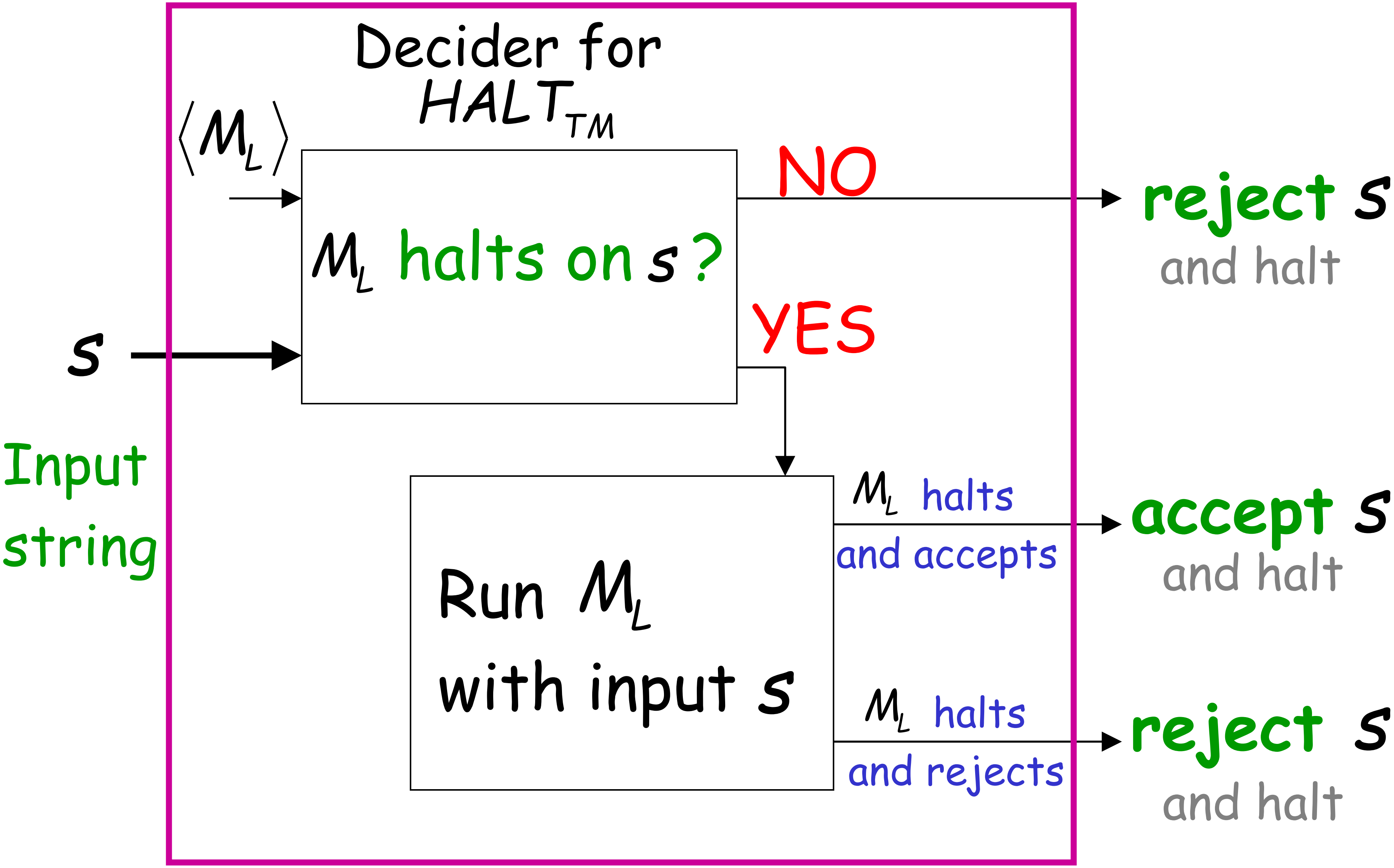
Let L be a Turing-Acceptable language

Let M_L be the Turing Machine that accepts L

We will prove that L is also decidable:

we will build a decider for L

Decider for L



Decider for $HALT_{TM}$

$\langle M_L \rangle$

M_L halts on s ?

NO

YES

reject s

and halt

s

Input string

Run M_L with input s

M_L halts and accepts

accept s

and halt

M_L halts and rejects

reject s

and halt

Therefore, L is decidable

Since L is chosen arbitrarily, every Turing-Acceptable language is decidable

But there is a Turing-Acceptable language which is undecidable

Contradiction!!!!

END OF PROOF

An alternative proof

Theorem: $HALT_{TM}$ is undecidable

(The halting problem is unsolvable)

Proof:

Basic idea:

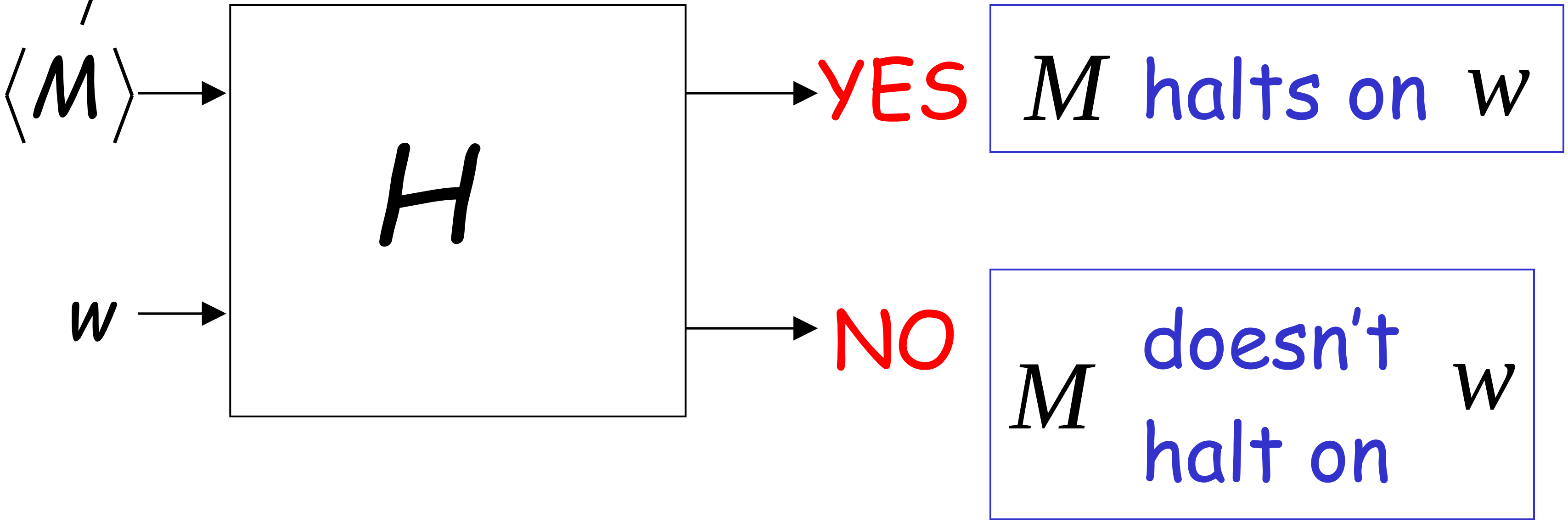
Assume for contradiction that the halting problem is decidable;

we will obtain a contradiction using a diagonilization technique

Suppose that $HALT_{TM}$ is decidable

Input string $\langle M,w \rangle$

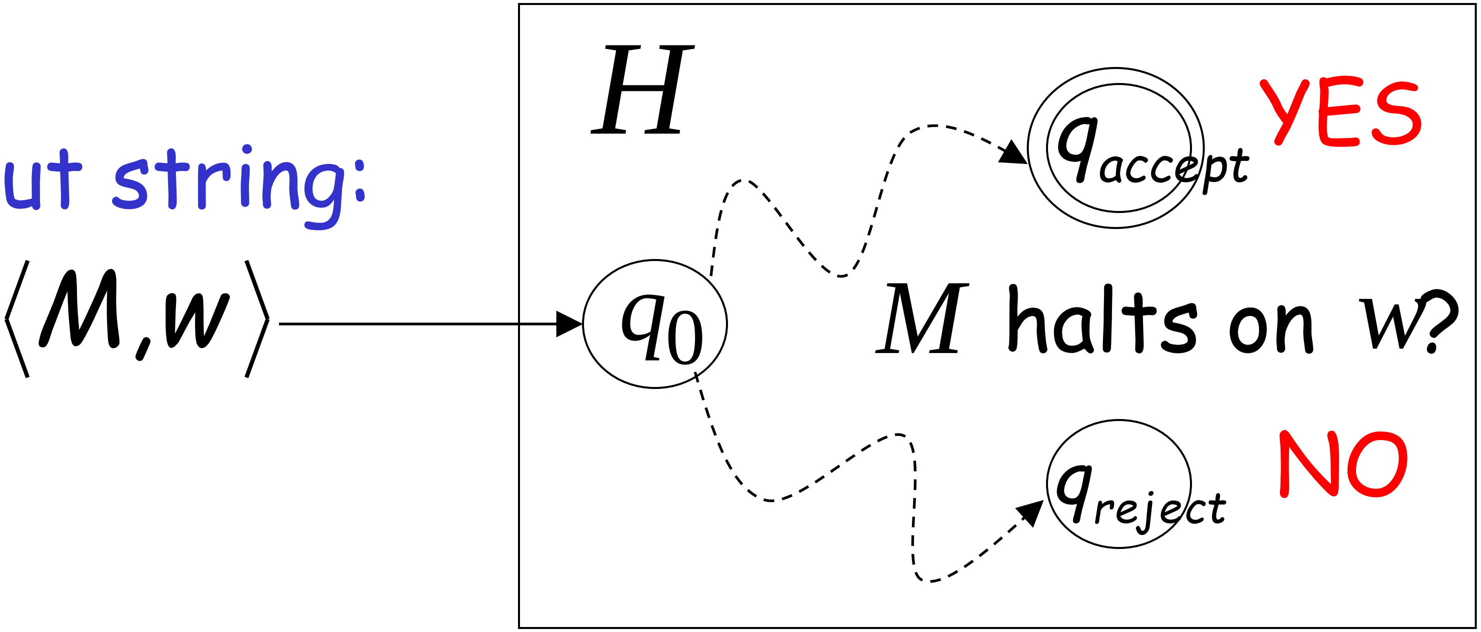
Decider for $HALT_{TM}$



Looking inside H

Decider for $HALT_{TM}$

Input string:

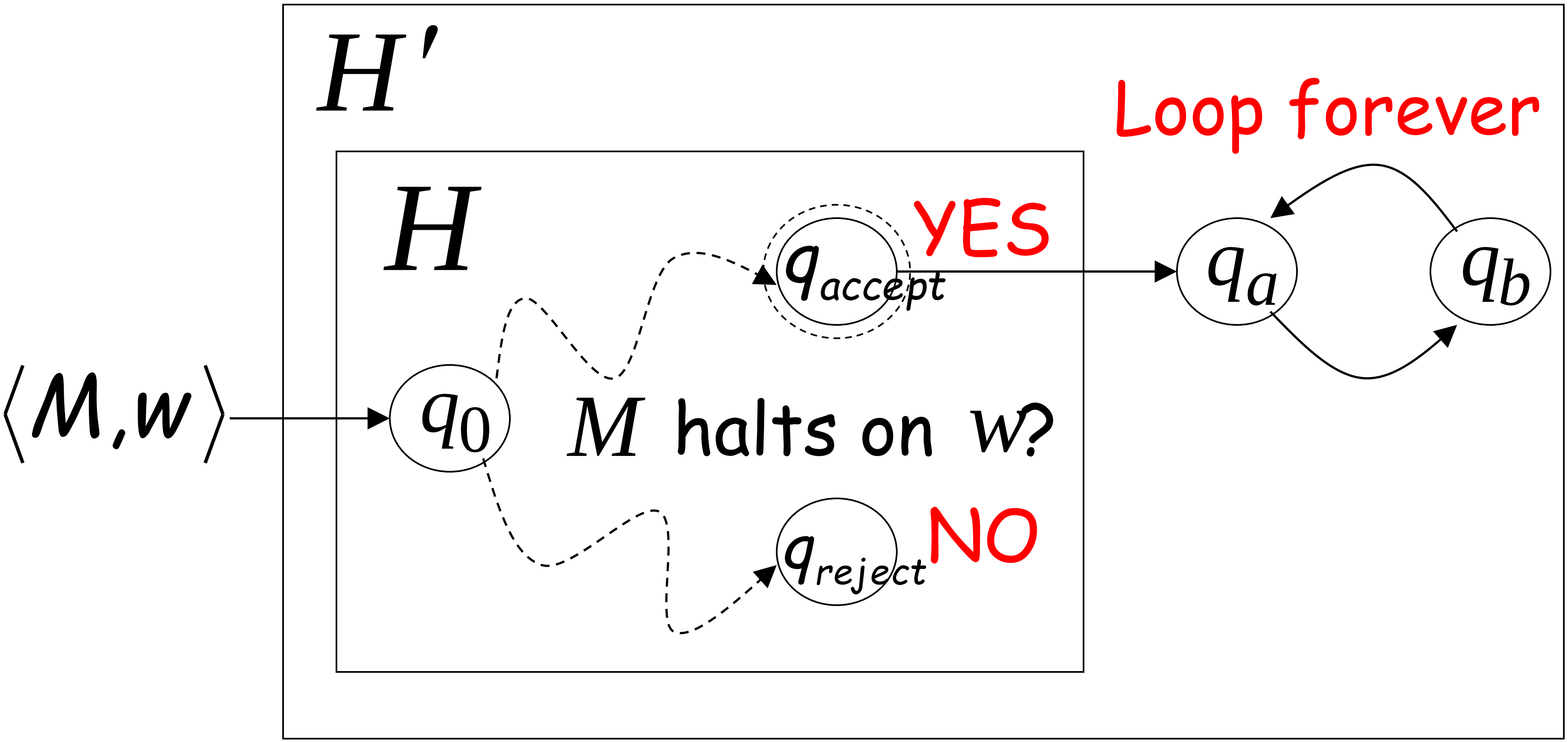


$\langle M,w \rangle$

H

M halts on w?

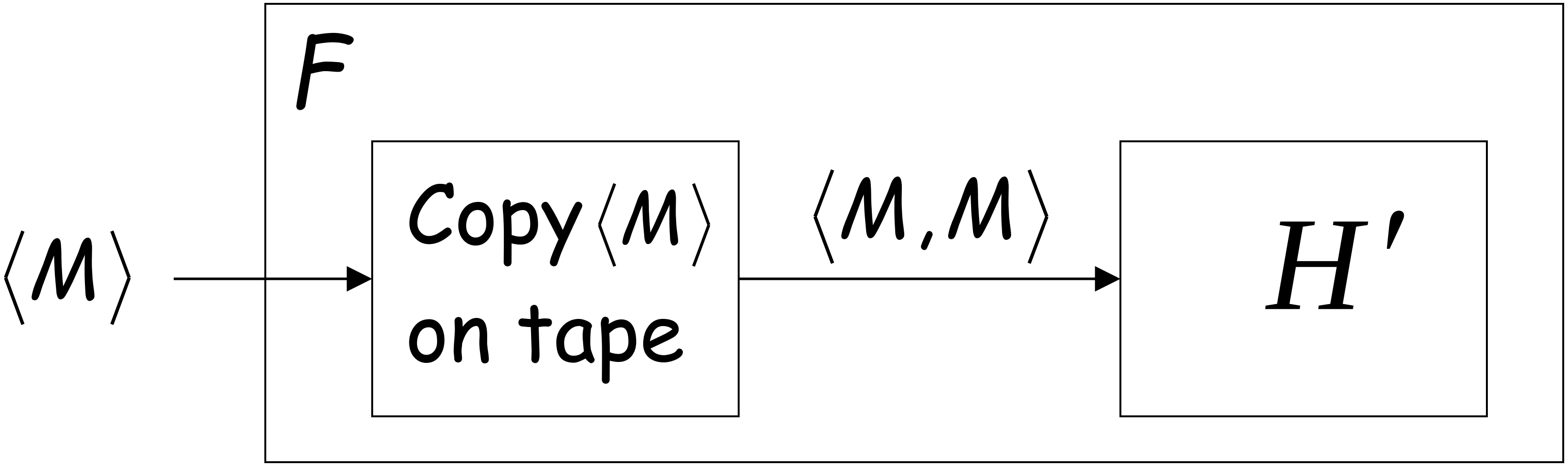
Construct machine H' :



If M halts on input w Then Loop Forever

Else Halt

Construct machine F :



$\langle M \rangle$

F

Copy $\langle M \rangle$ on tape

$\langle M, M \rangle$

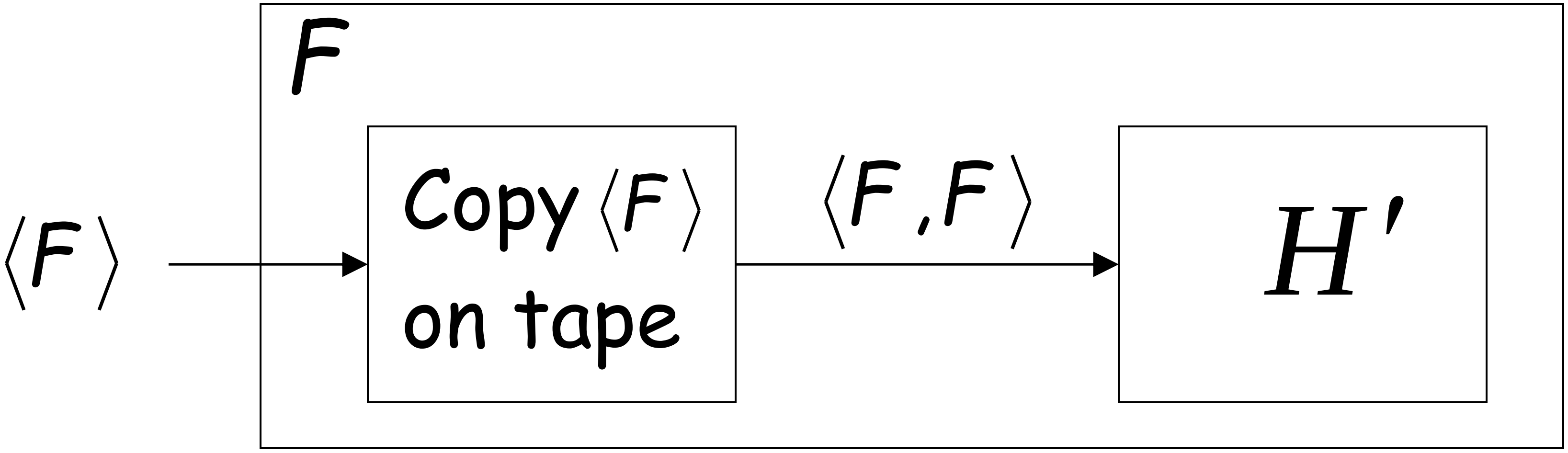
H'

If M halts on input $\langle M \rangle$

Then loop forever

Else halt

Run F with input itself



F

$\langle F \rangle$

Copy $\langle F \rangle$ on tape

$\langle F, F \rangle$

H'

If F halts on input $\langle F \rangle$

Then F loops forever on input $\langle F \rangle$

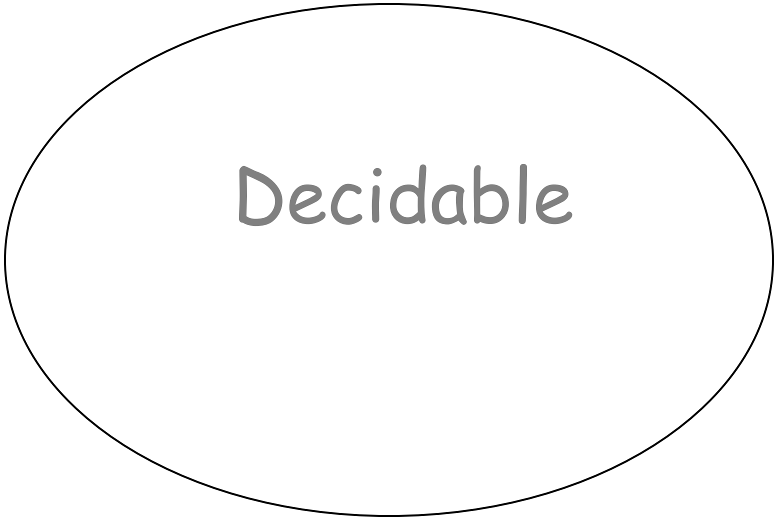
Else F halts on input $\langle F \rangle$

CONTRADICTION!!!

END OF PROOF

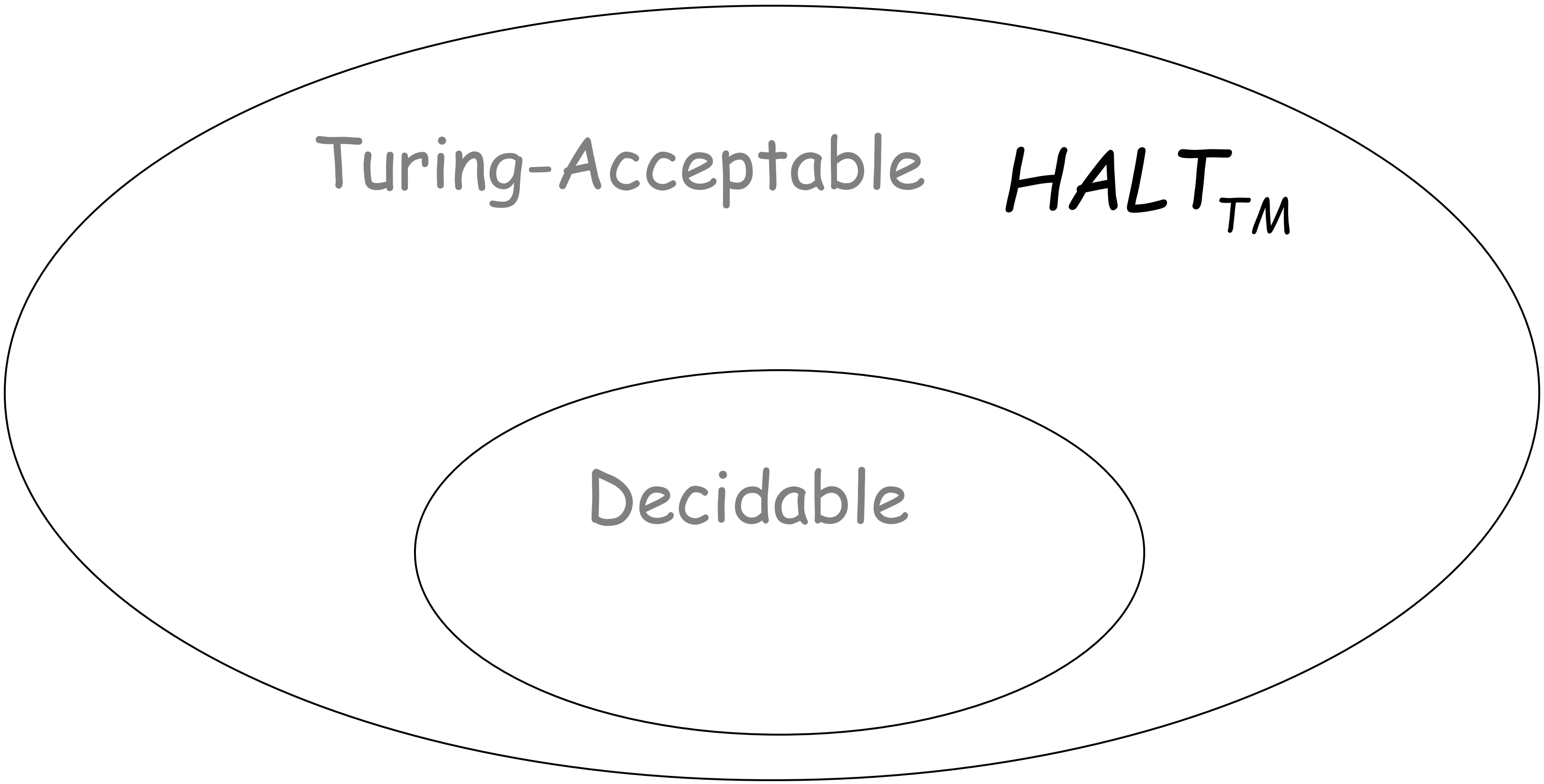
We have shown:

Undecidable $HALT_{TM}$



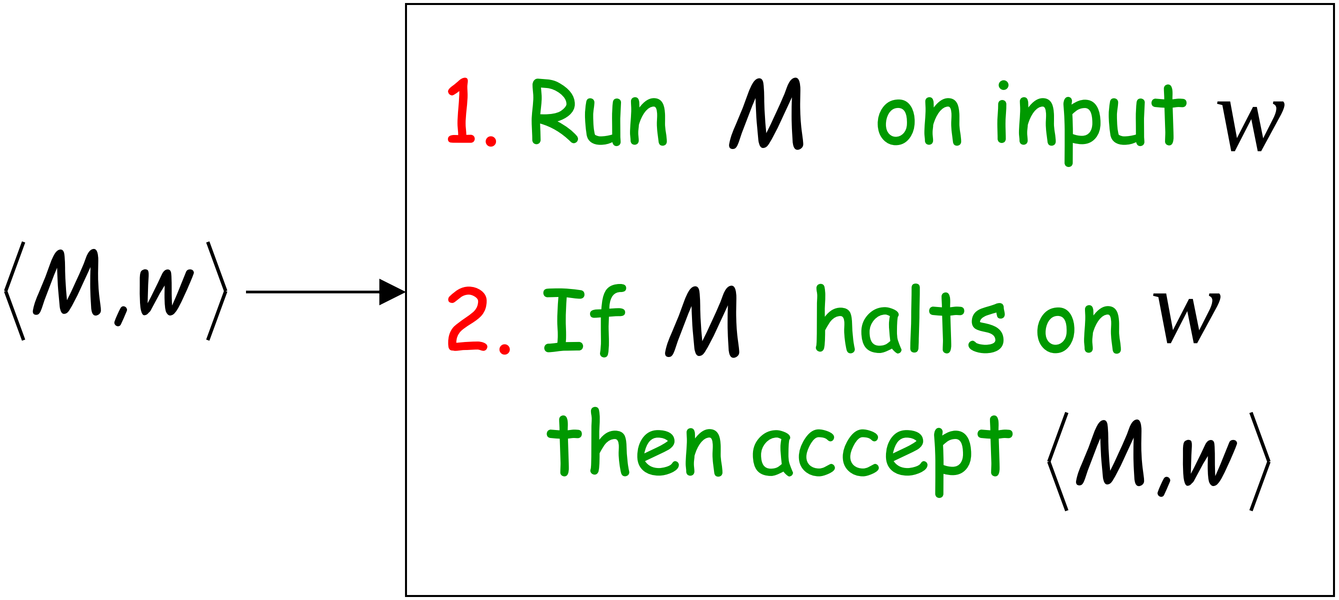
Decidable

We can actually show:



$HALT_{TM}$ is Turing-Acceptable

Turing machine that accepts $HALT_{TM}$:

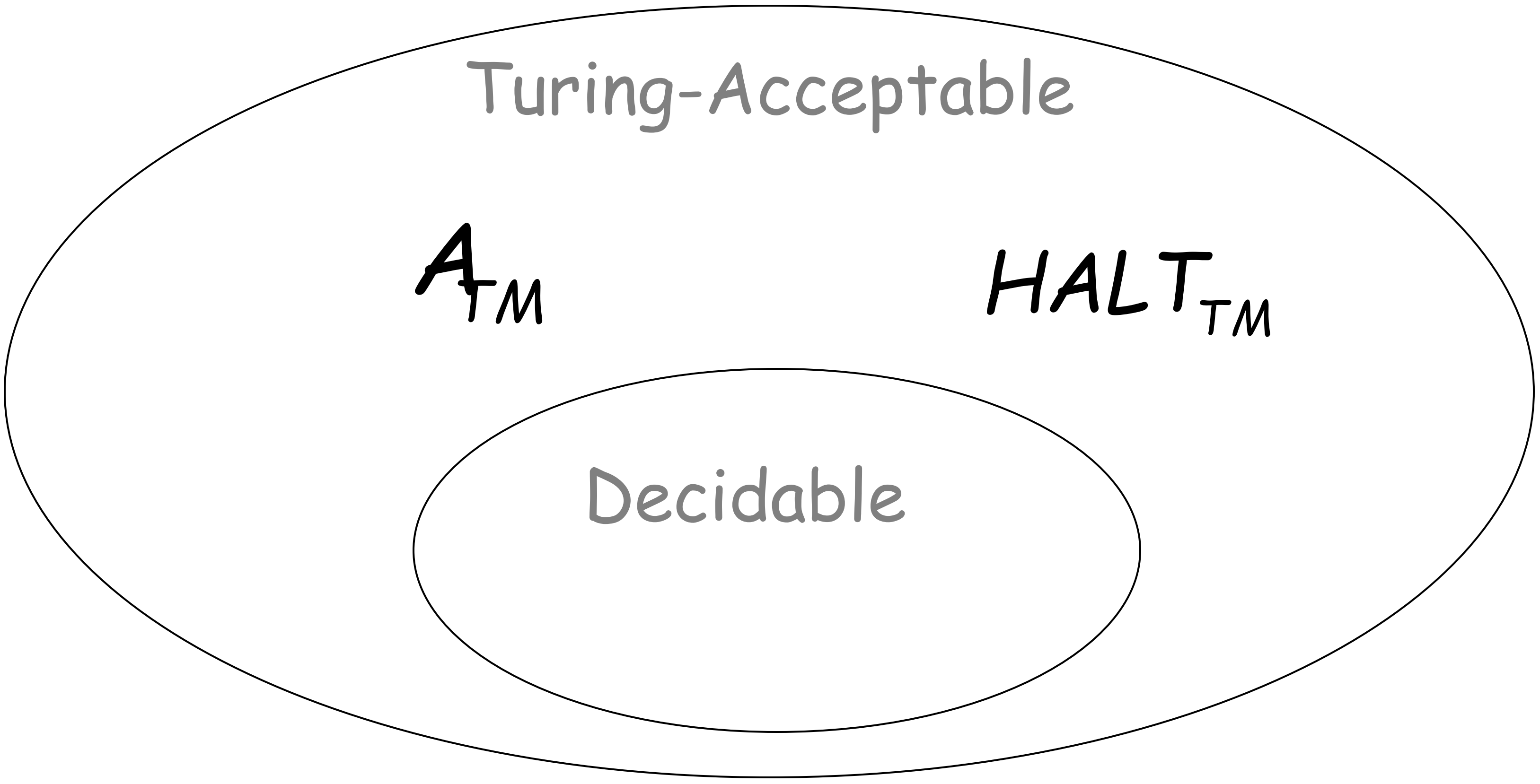


$\langle M,w \rangle$

1. Run M on input w

2. If M halts on w
   then accept $\langle M,w \rangle$

We showed:



Turing-Acceptable

A_{TM}

$HALT_{TM}$

Decidable

Reductions

Computable function f :

There is a deterministic Turing machine M which for any input string w computes $f(w)$ and writes it on the tape

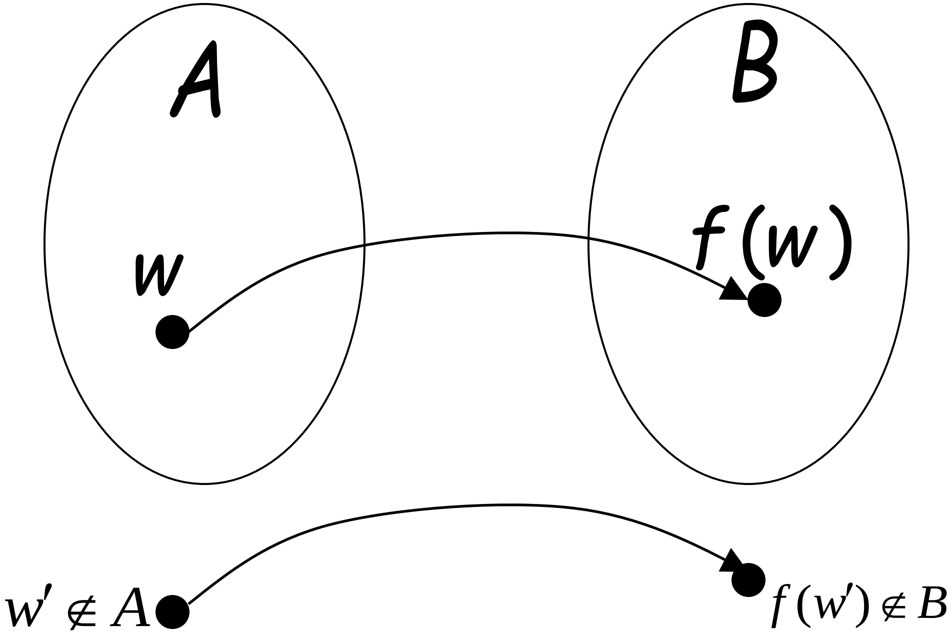
Problem X is reduced to problem Y

⇓

If we can solve problem Y
then we can solve problem X

Definition:

Language A
is reduced to
language B



There is a computable function f (*reduction*) such that:

$$w \in A \Leftrightarrow f(w) \in B$$

Theorem 1:

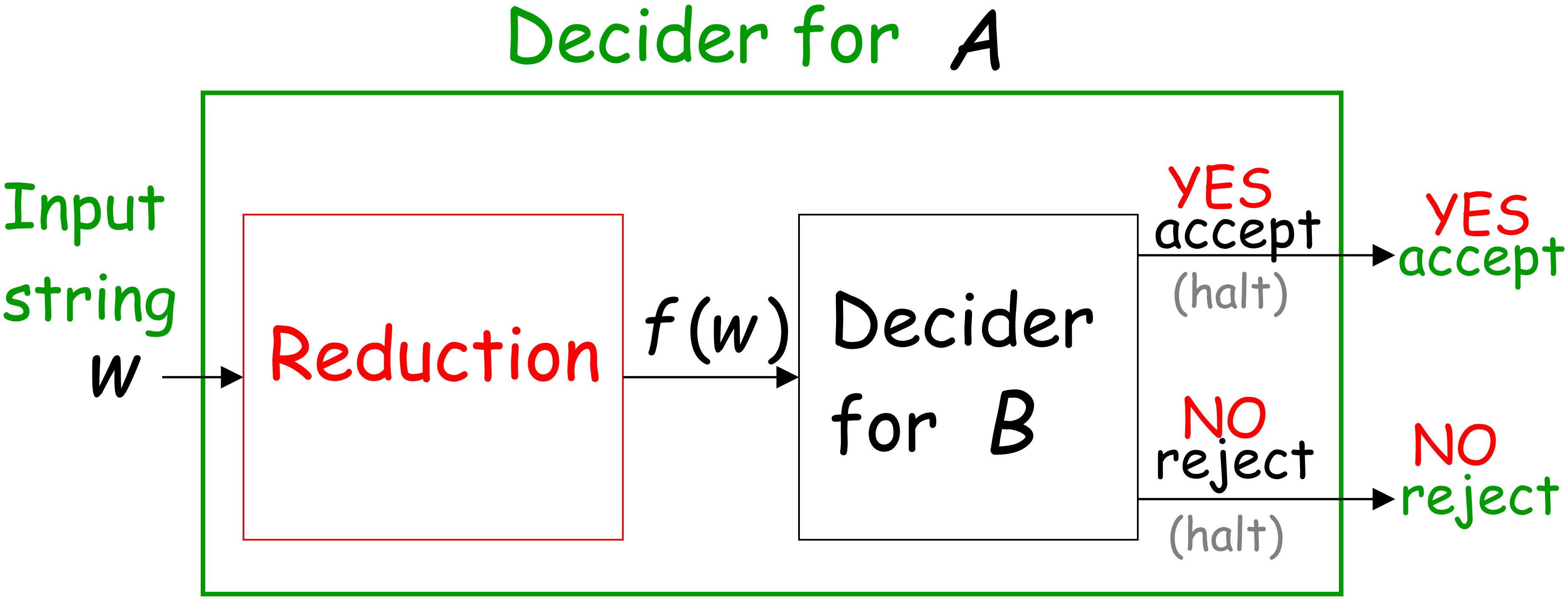
If: Language A is reduced to B
and language B is decidable

Then: A is decidable

Proof:

Basic idea:

Build the decider for A
using the decider for B



Decider for A

Input string w

Reduction

$f(w)$

Decider for B

YES accept (halt)

NO reject (halt)

YES accept

NO reject

From reduction: $w \in A \iff f(w) \in B$

END OF PROOF

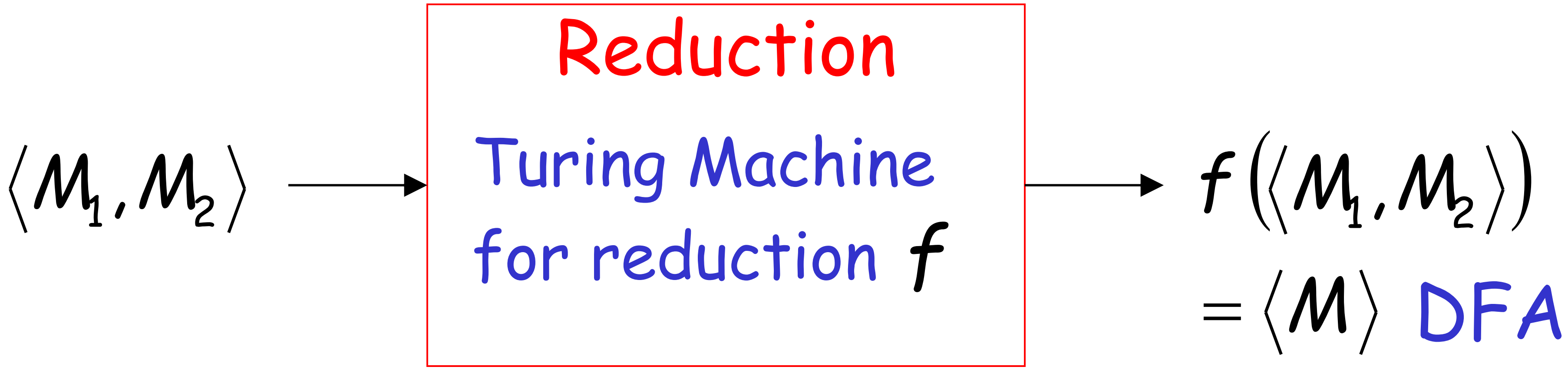
Example:

$$EQUAL_{DFA} = \{\langle M_1, M_2 \rangle : M_1 \text{ and } M_2 \text{ are DFAs that accept the same languages}\}$$

is reduced to:

$$EMPTY_{DFA} = \{\langle M \rangle : M \text{ is a DFA that accepts the empty language } \emptyset\}$$

We only need to construct:



$\langle M_1, M_2 \rangle \longrightarrow$

Reduction

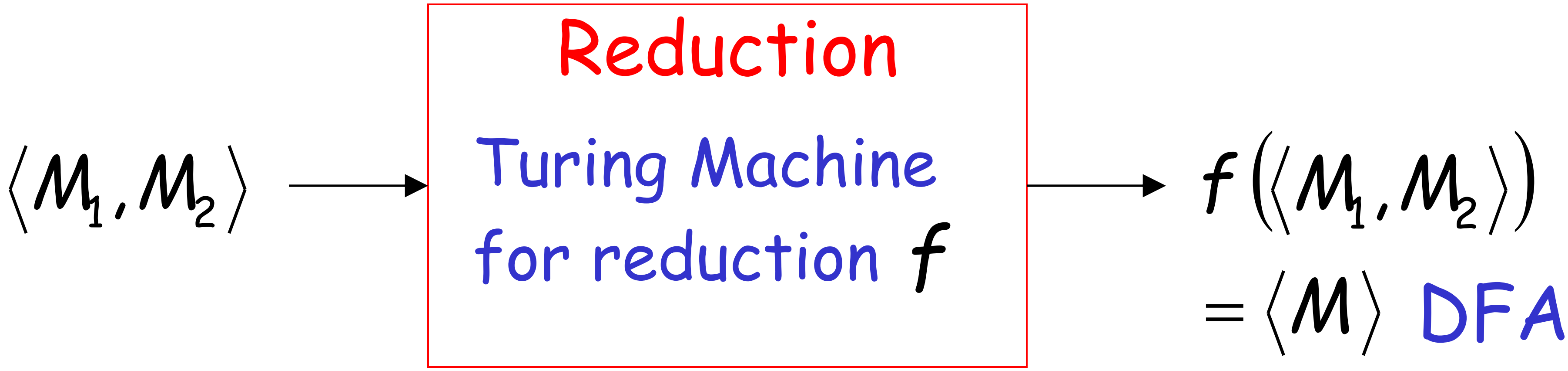
Turing Machine
for reduction f

$\longrightarrow f(\langle M_1, M_2 \rangle)$
$= \langle M \rangle$ DFA

$$\langle M_1, M_2 \rangle \in EQUAL_{DFA} \quad \Leftrightarrow \quad \langle M \rangle \in EMPTY_{DFA}$$

Let L_1 be the language of DFA M_1

Let L_2 be the language of DFA M_2



$\langle M_1, M_2 \rangle$

Reduction

Turing Machine

for reduction f

$f(\langle M_1, M_2 \rangle)$

$= \langle M \rangle$ DFA

construct DFA M

by combining M_1 and M_2 so that:

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

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

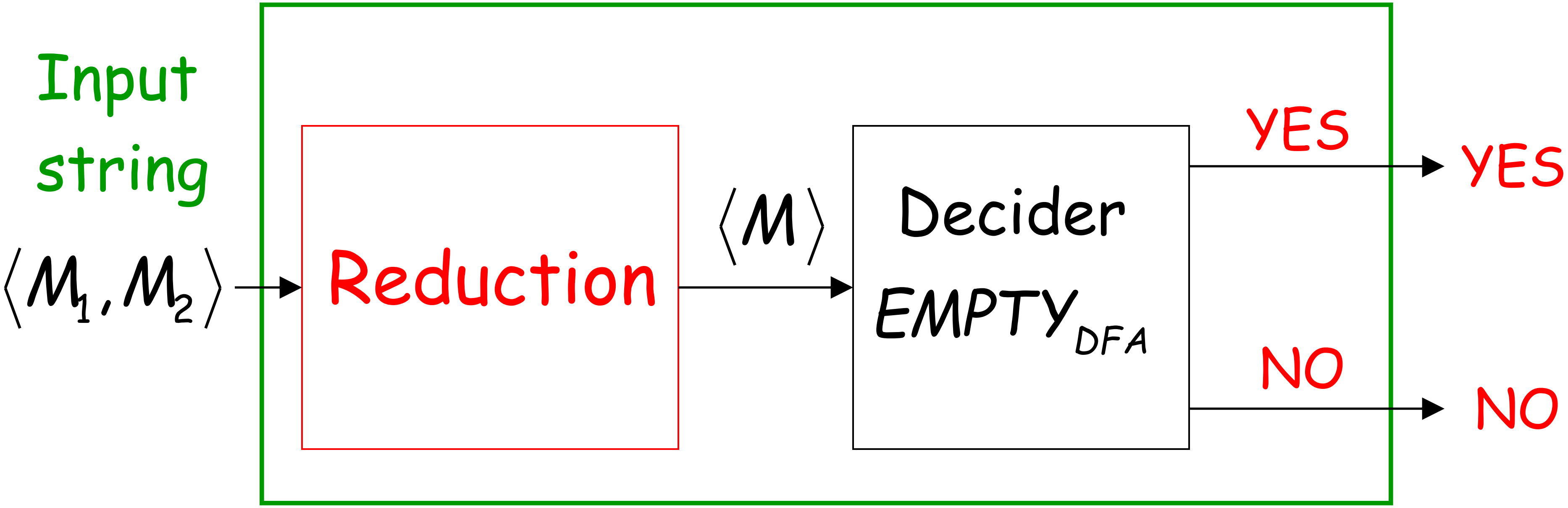
$$\Updownarrow$$

$$L_1 = L_2 \quad \Leftrightarrow \quad L(M) = \varnothing$$

$$\Updownarrow$$

$$\langle M_1, M_2 \rangle \in EQUAL_{DFA} \quad \Leftrightarrow \quad \langle M \rangle \in EMPTY_{DFA}$$

Decider for $EQUAL_{DFA}$



Input string

$\langle M_1, M_2 \rangle$

Reduction

$\langle M \rangle$

Decider $EMPTY_{DFA}$

YES

YES

NO

NO

Theorem 2:

If: Language A is reduced to B
and language A is undecidable

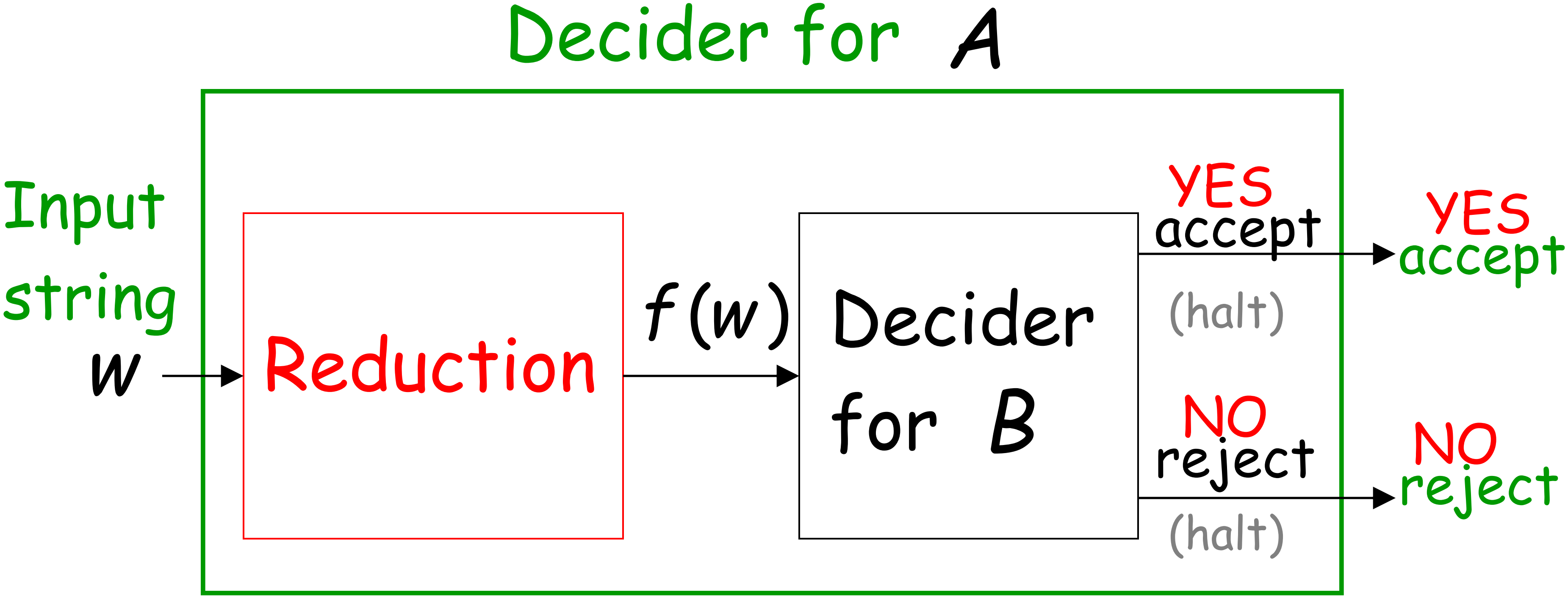
Then: B is undecidable

Proof: Suppose B is decidable

Using the decider for B
build the decider for A

Contradiction!

If B is decidable then we can build:



$$w \in A \iff f(w) \in B$$

CONTRADICTION!

END OF PROOF

Observation:

To prove that language B is undecidable
we only need to reduce
a known undecidable language A to B

State-entry problem

Input:
- Turing Machine M
- State q
- String w

Question: Does M enter state q while processing input string w ?

Corresponding language:

$$STATE_{TM} = \{\langle M, w, q \rangle : M \text{ is a Turing machine that enters state } q \text{ on input string } w\}$$

(while processing)

Theorem: $STATE_{TM}$ is undecidable

(state-entry problem is unsolvable)

Proof: Reduce

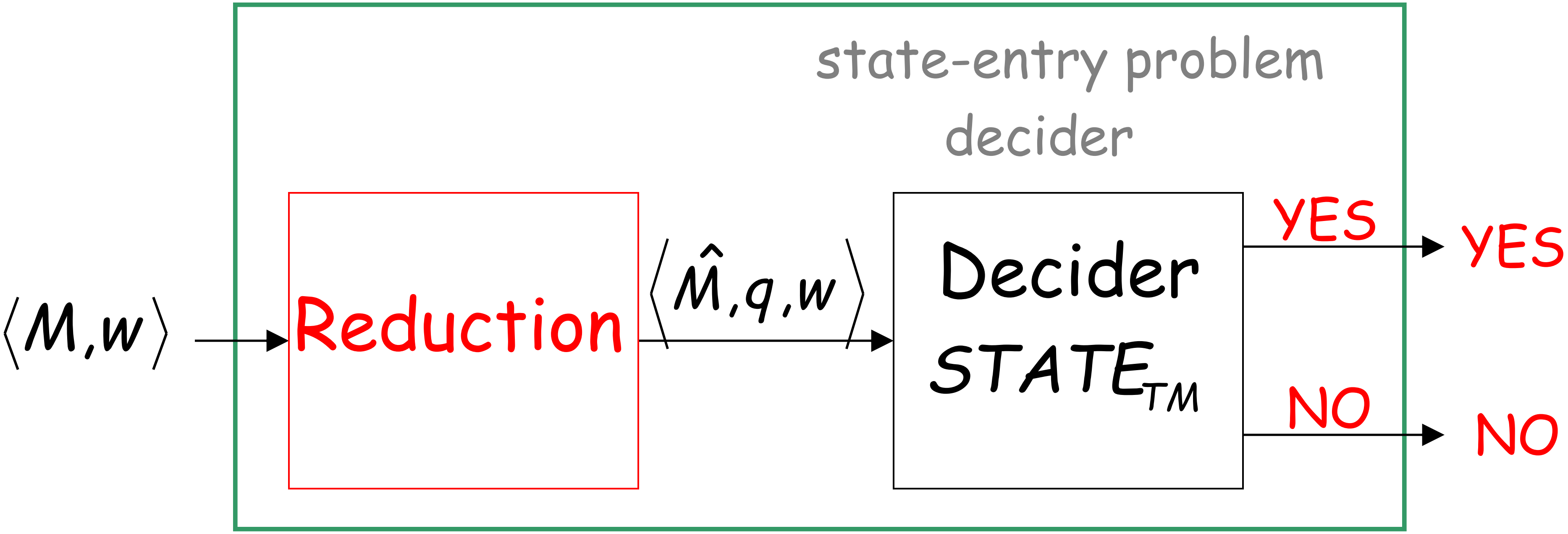
$HALT_{TM}$ (halting problem)

to

$STATE_{TM}$ (state-entry problem)

Halting Problem Decider

Decider for $HALT_{TM}$



state-entry problem decider

$\langle M, w \rangle$

Reduction

$\langle \hat{M}, q, w \rangle$

Decider $STATE_{TM}$

YES

YES

NO

NO

Given the reduction,
if $STATE_{TM}$ is decidable,
then $HALT_{TM}$ is decidable

A contradiction!
since $HALT_{TM}$
is undecidable

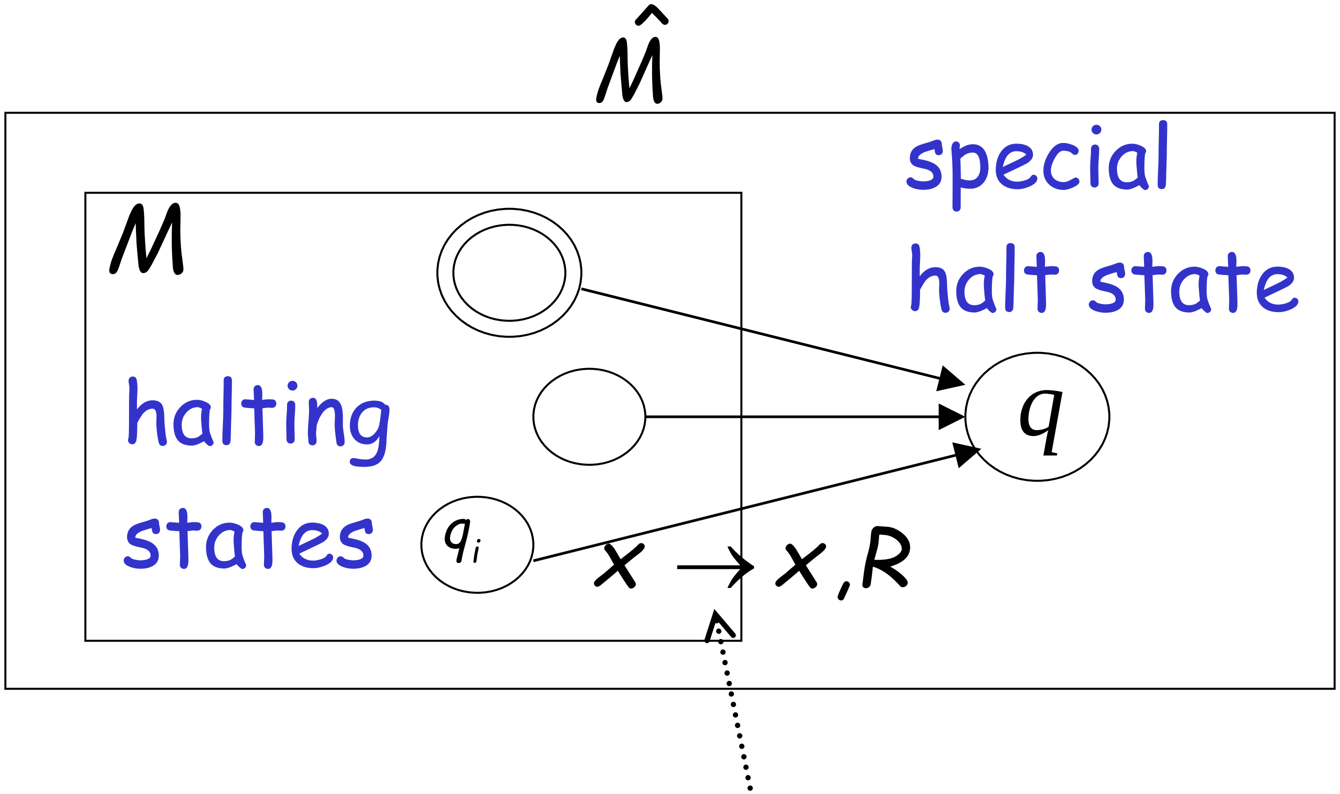
We only need to build the reduction:

$$\langle M,w\rangle \longrightarrow \boxed{\text{Reduction}} \longrightarrow \left\langle \hat{M},q,w \right\rangle$$

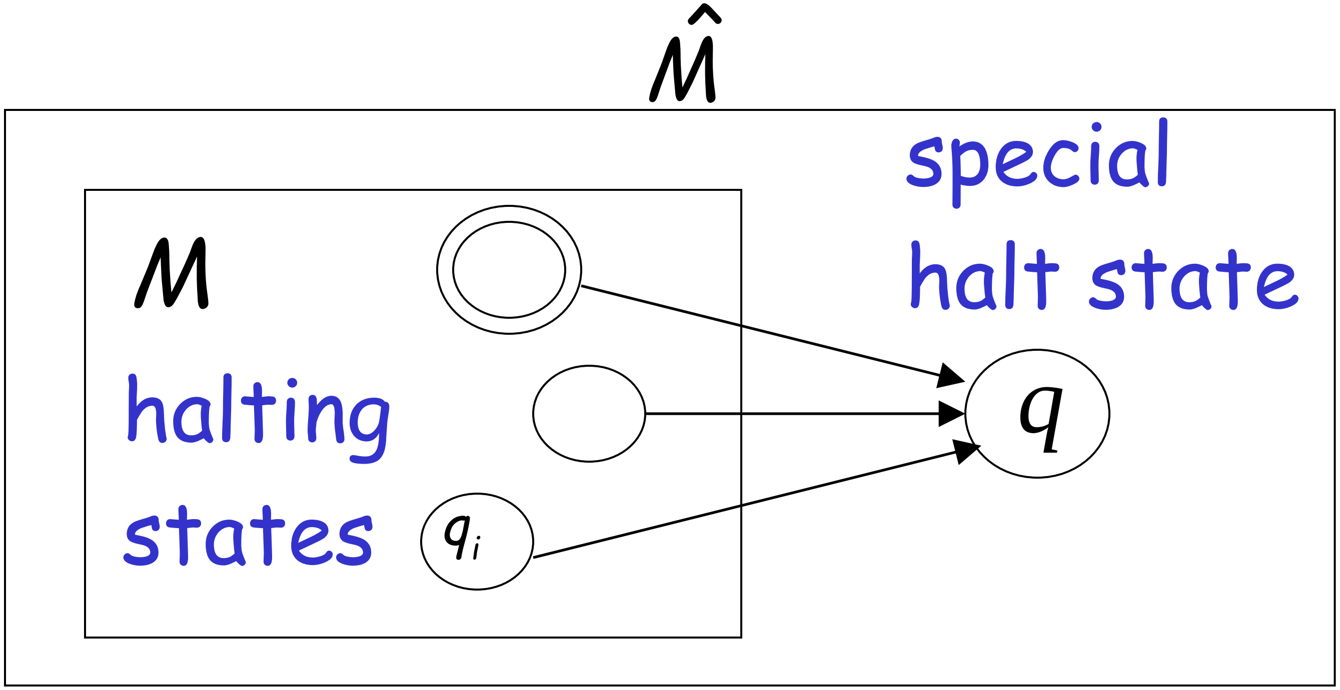
So that:

$$\langle M,w\rangle \in HALT_{TM} \quad \Longleftrightarrow \quad \left\langle \hat{M},w,q \right\rangle \in STATE_{TM}$$

For the reduction, construct $\hat{M}$ from M :



A transition for every unused tape symbol x of q_i



halting states

special halt state

M halts $\Longleftrightarrow$ $\hat{M}$ halts on state q

Therefore: M halts on input w

$$\Updownarrow$$

$\hat{M}$ halts on state q on input w

Equivalently:

$$\langle M,w\rangle \in HALT_{TM} \quad \Longleftrightarrow \quad \left\langle \hat{M},w,q\right\rangle \in STATE_{TM}$$

END OF PROOF

Blank-tape halting problem

Input: Turing Machine M

Question: Does M halt when started with a blank tape?

Corresponding language:

$$BLANK_{TM} = \{\langle M \rangle : M \text{ is a Turing machine that halts when started on blank tape}\}$$

Theorem: $BLANK_{TM}$ is undecidable

(blank-tape halting problem is unsolvable)

Proof: Reduce

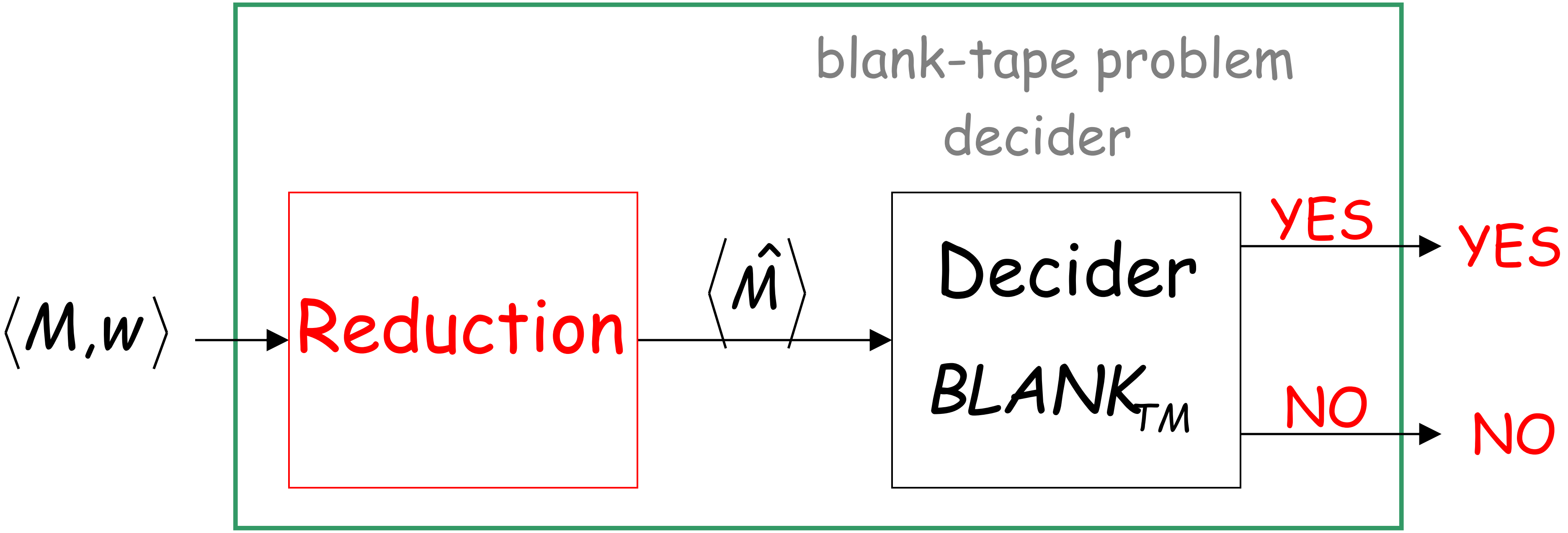
$HALT_{TM}$ (halting problem)

to

$BLANK_{TM}$ (blank-tape problem)

Halting Problem Decider

Decider for $HALT_{TM}$



Given the reduction,
If $BLANK_{TM}$ is decidable,
then $HALT_{TM}$ is decidable

A contradiction!
since $HALT_{TM}$
is undecidable

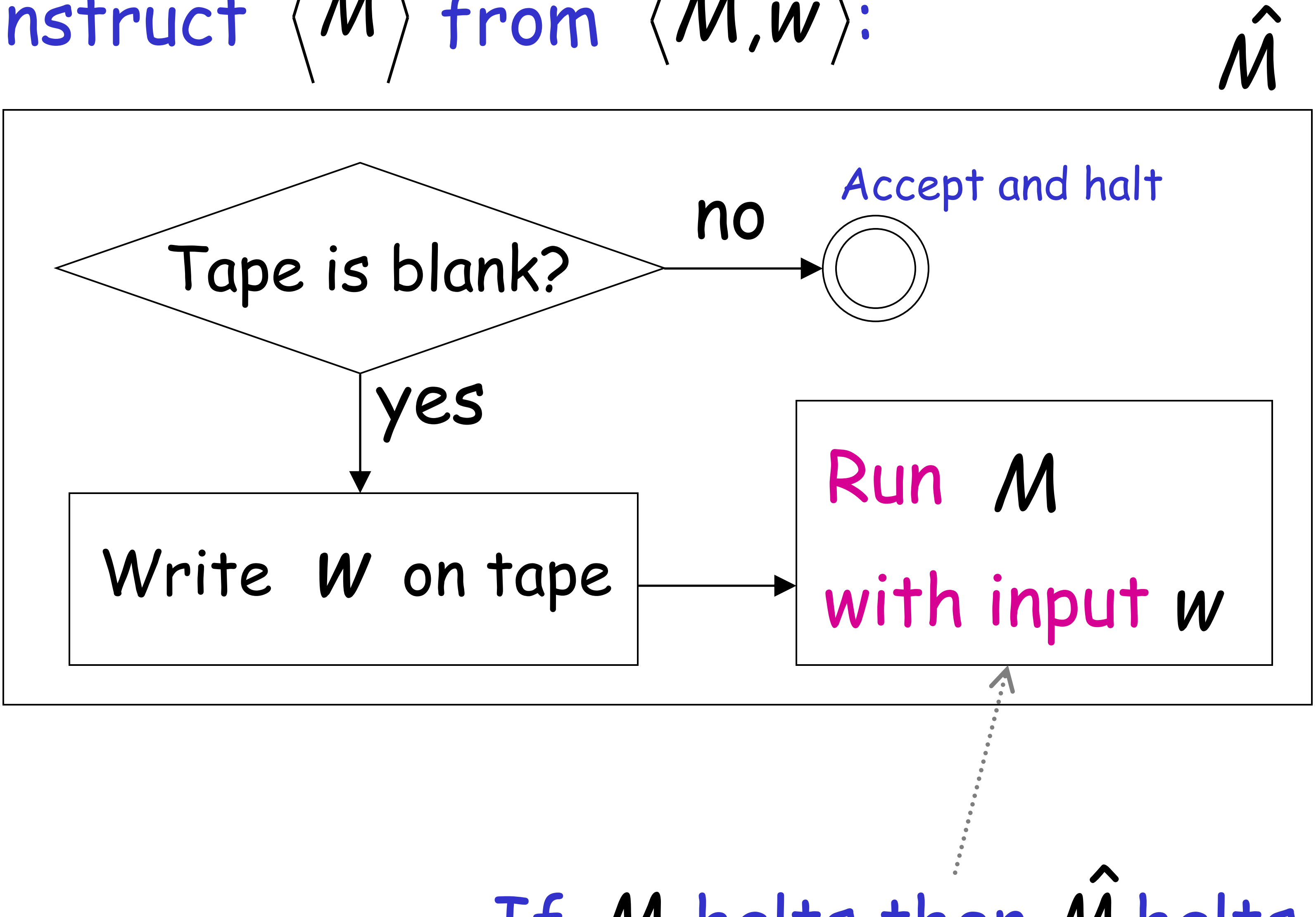
We only need to build the reduction:

$$\langle M,w \rangle \longrightarrow \boxed{\text{Reduction}} \longrightarrow \langle \hat{M} \rangle$$

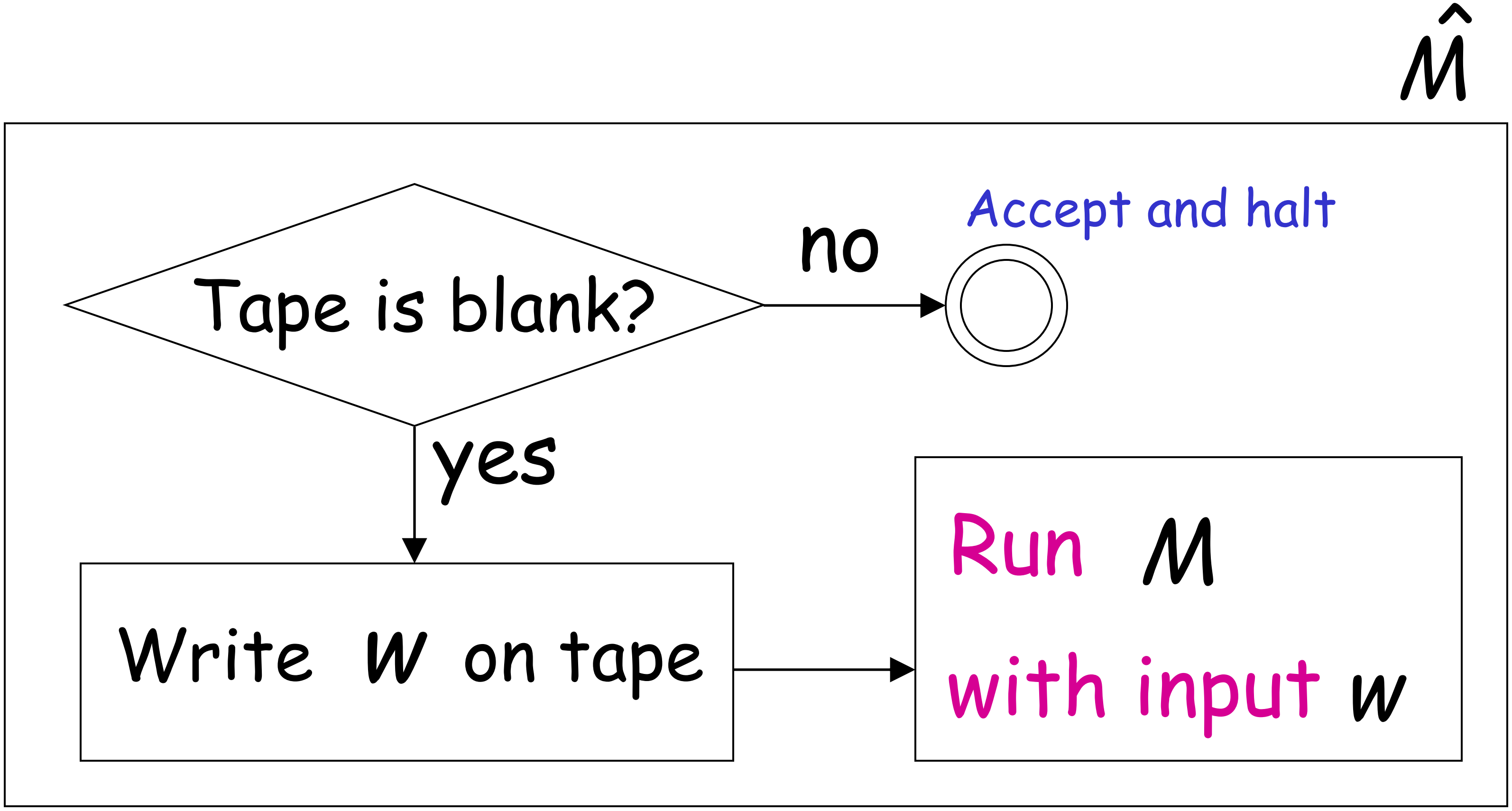
So that:

$$\langle M,w \rangle \in HALT_{TM} \Longleftrightarrow \langle \hat{M} \rangle \in BLANK_{TM}$$

Construct $\langle \hat{M} \rangle$ from $\langle M, w \rangle$:



If M halts then $\hat{M}$ halts too



M halts on input w

⇕

$\hat{M}$ halts when started on blank tape

M halts on input w

$$\Updownarrow$$

$\hat{M}$ halts when started on blank tape

Equivalently:

$$\langle M, w \rangle \in HALT_{TM} \quad \Longleftrightarrow \quad \langle \hat{M} \rangle \in BLANK_{TM}$$

END OF PROOF

Theorem 3:

If: Language A is reduced to $\overline{B}$
and language A is undecidable

Then: B is undecidable

Proof: Suppose B is decidable

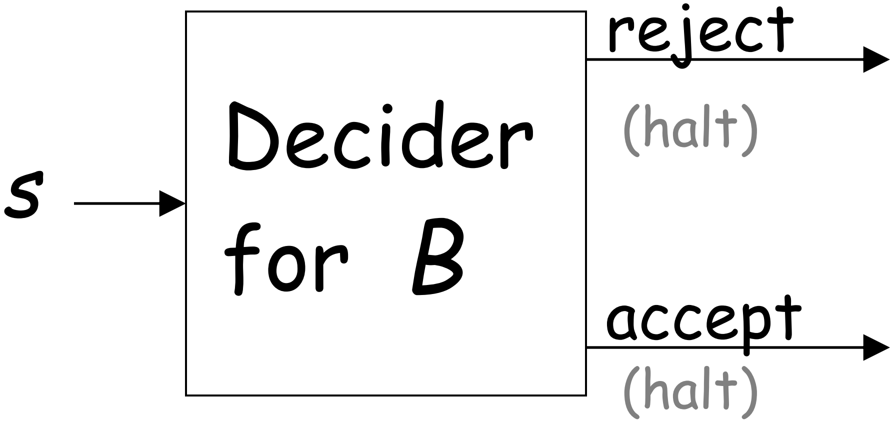
Then $\overline{B}$ is decidable

Using the decider for $\overline{B}$

build the decider for A

Contradiction!

Suppose B is decidable



s

Decider for B

reject

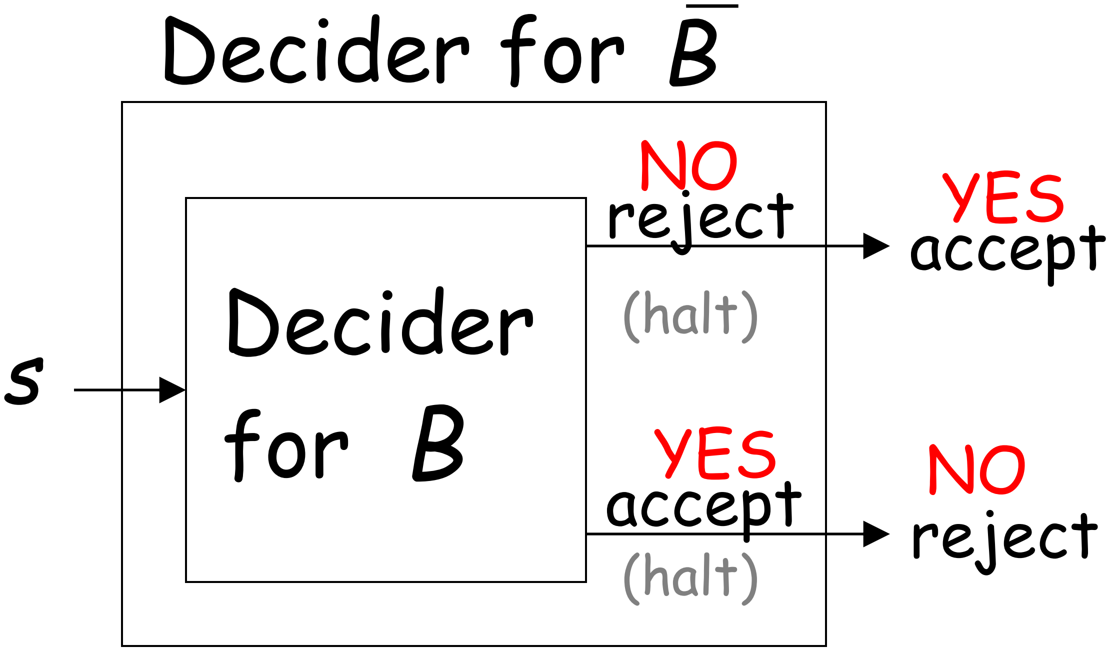
(halt)

accept

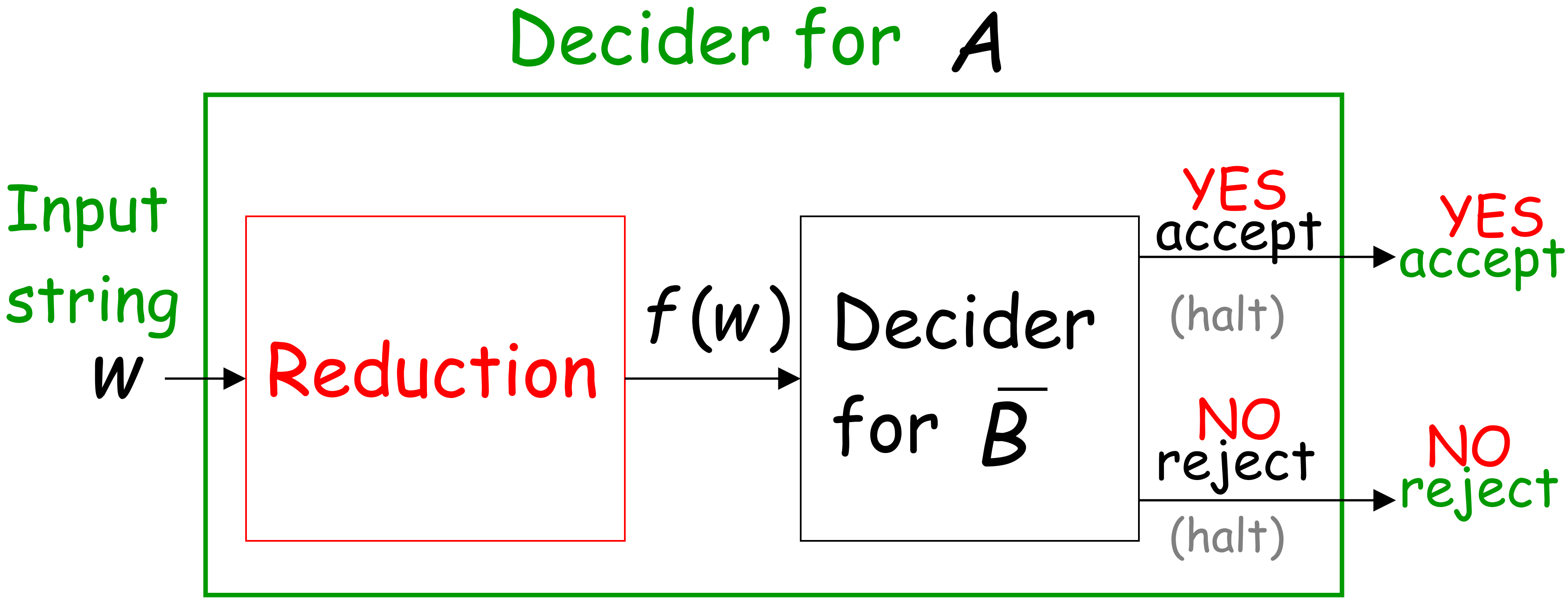
(halt)

Suppose B is decidable

Then $\overline{B}$ is decidable



If $\overline{B}$ is decidable then we can build:



Decider for A

Input string w

Reduction

$f(w)$

Decider for $\overline{B}$

YES accept (halt)

NO reject (halt)

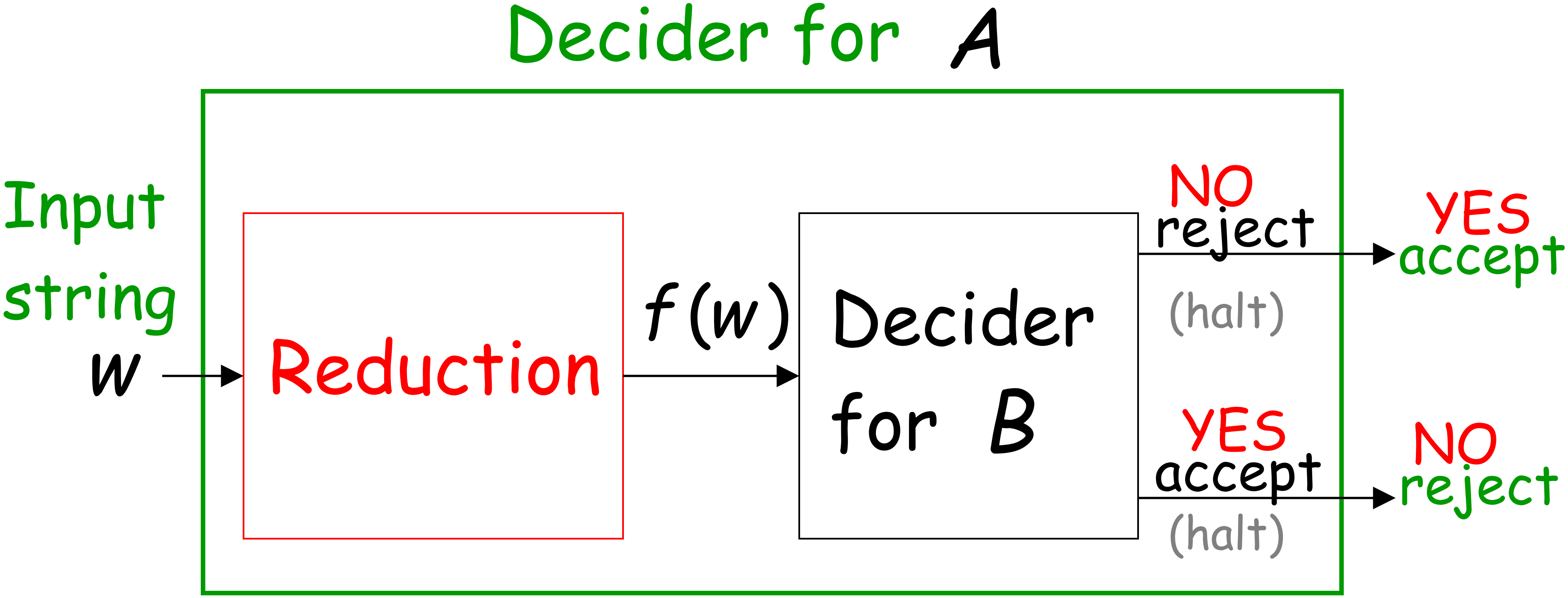
YES accept

NO reject

$$w \in A \Leftrightarrow f(w) \in \overline{B}$$

CONTRADICTION!

Alternatively:



$$w \in A \iff f(w) \notin B$$

CONTRADICTION!

END OF PROOF

Observation:

To prove that language B is undecidable
we only need to reduce
a known undecidable language A
to B (Theorem 2)
or $\overline{B}$ (Theorem 3)

Undecidable Problems for Turing Recognizable languages

Let L be a Turing-acceptable language

- L is empty?
- L is regular?
- L has size 2?

All these are undecidable problems

Let L be a Turing-acceptable language

- L is empty?
- L is regular?
- L has size 2?

Empty language problem

Input: Turing Machine M

Question: Is $L(M)$ empty? $L(M) = \emptyset ?$

Corresponding language:

$$EMPTY_{TM} = \{\langle M \rangle : M \text{ is a Turing machine that accepts the empty language } \emptyset\}$$

Theorem: $EMPTY_{TM}$ is undecidable

(empty-language problem is unsolvable)

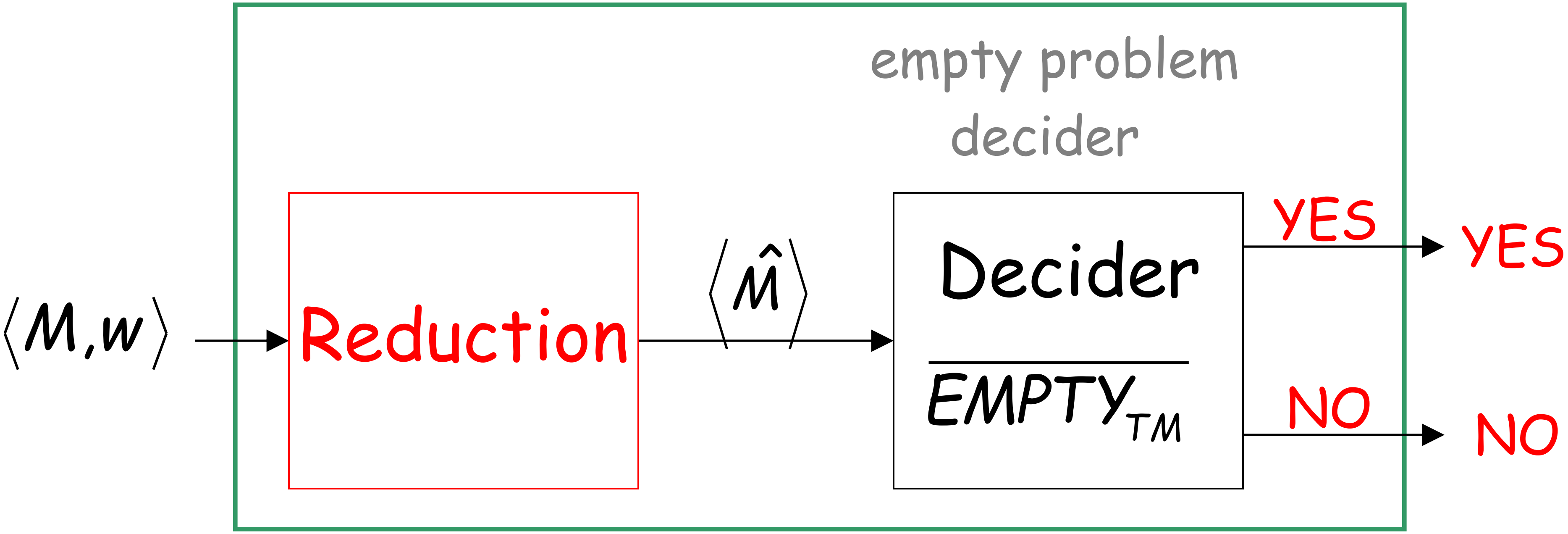
Proof: Reduce

A_{TM} (membership problem)

to

$\overline{EMPTY_{TM}}$ (empty language problem) membership problem decider

Decider for A_{TM}



empty problem decider

$\langle M, w \rangle$

Reduction

$\langle \hat{M} \rangle$

Decider

$\overline{EMPTY_{TM}}$

YES

YES

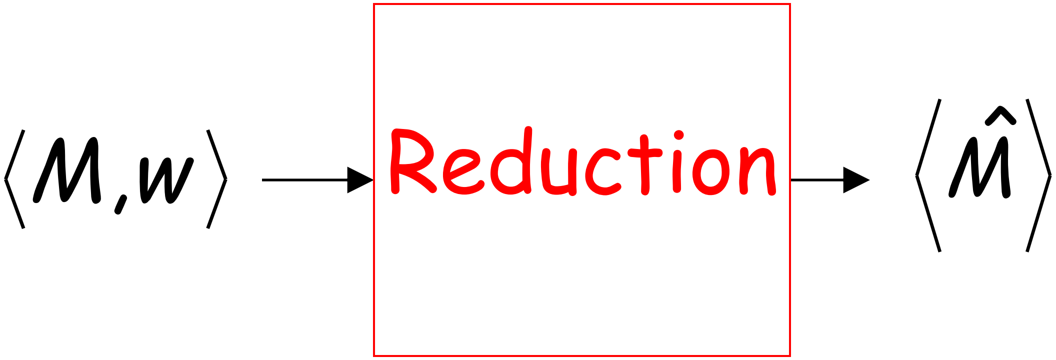
NO

NO

Given the reduction,
if $\overline{EMPTY_{TM}}$ is decidable,
then A_{TM} is decidable

A contradiction!
since A_{TM}
is undecidable

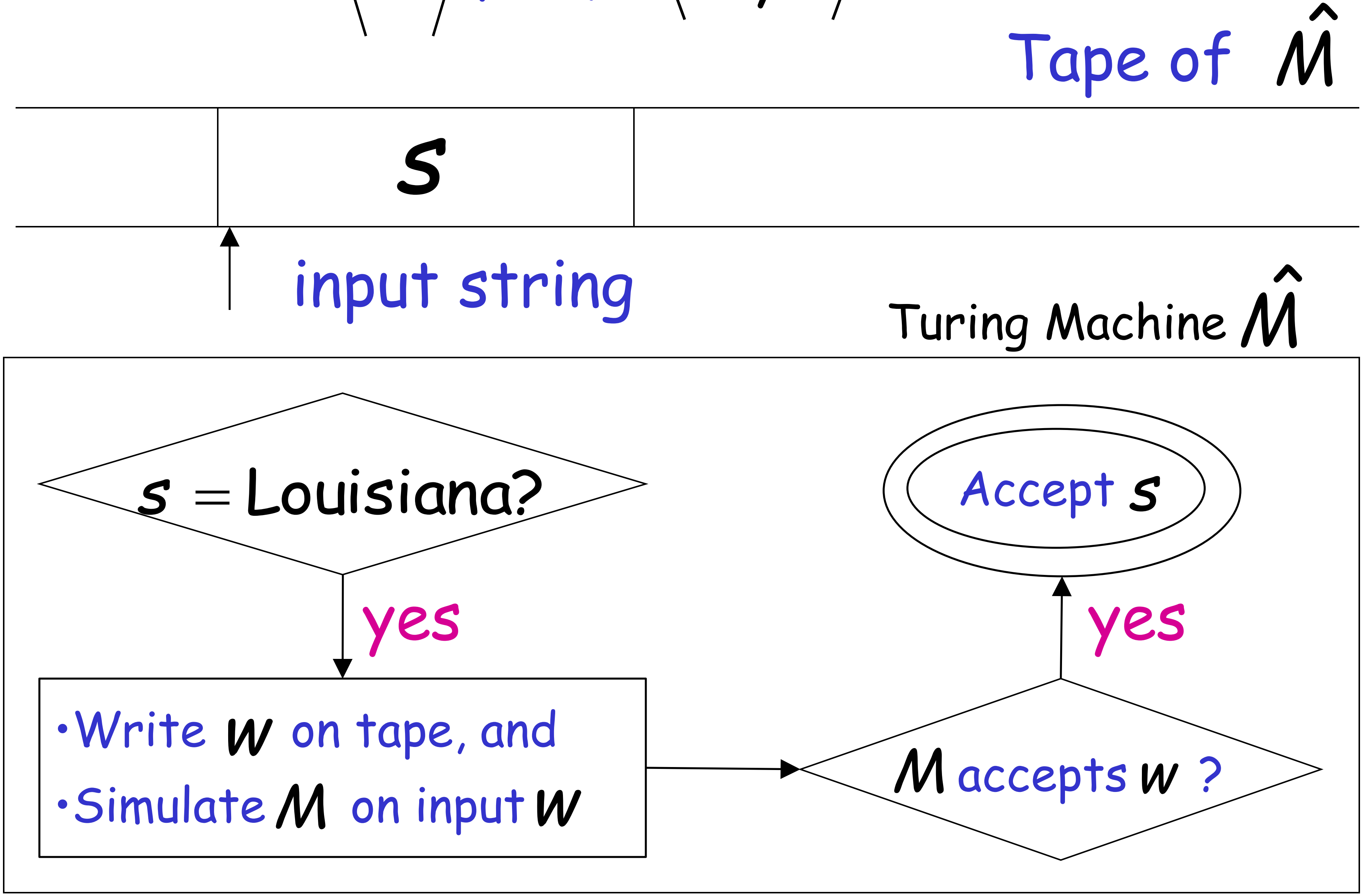
We only need to build the reduction:



So that:

$$\langle M, w\rangle \in A_{TM} \quad \Longleftrightarrow \quad \langle \hat{M}\rangle \in \overline{EMPTY_{TM}}$$

Construct $\langle \hat{M} \rangle$ from $\langle M, w \rangle$:



Tape of $\hat{M}$

s

input string

Turing Machine $\hat{M}$

$s =$ Louisiana?

yes

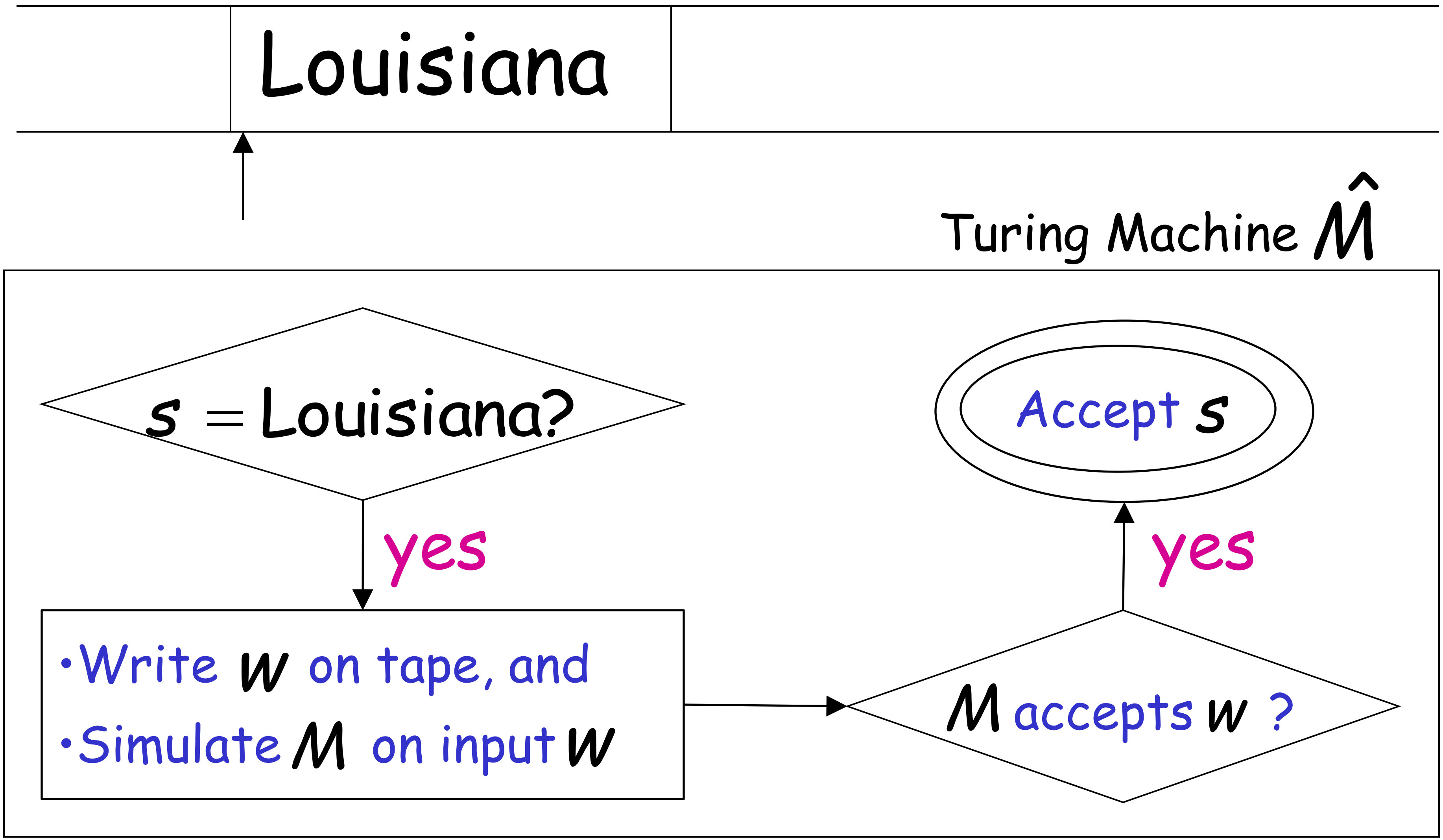
- Write w on tape, and
- Simulate M on input w

M accepts w ?

yes

Accept s

The only possible accepted string s



Louisiana

Turing Machine $\hat{M}$

$s =$ Louisiana?

yes

- Write w on tape, and
- Simulate M on input w

M accepts w ?

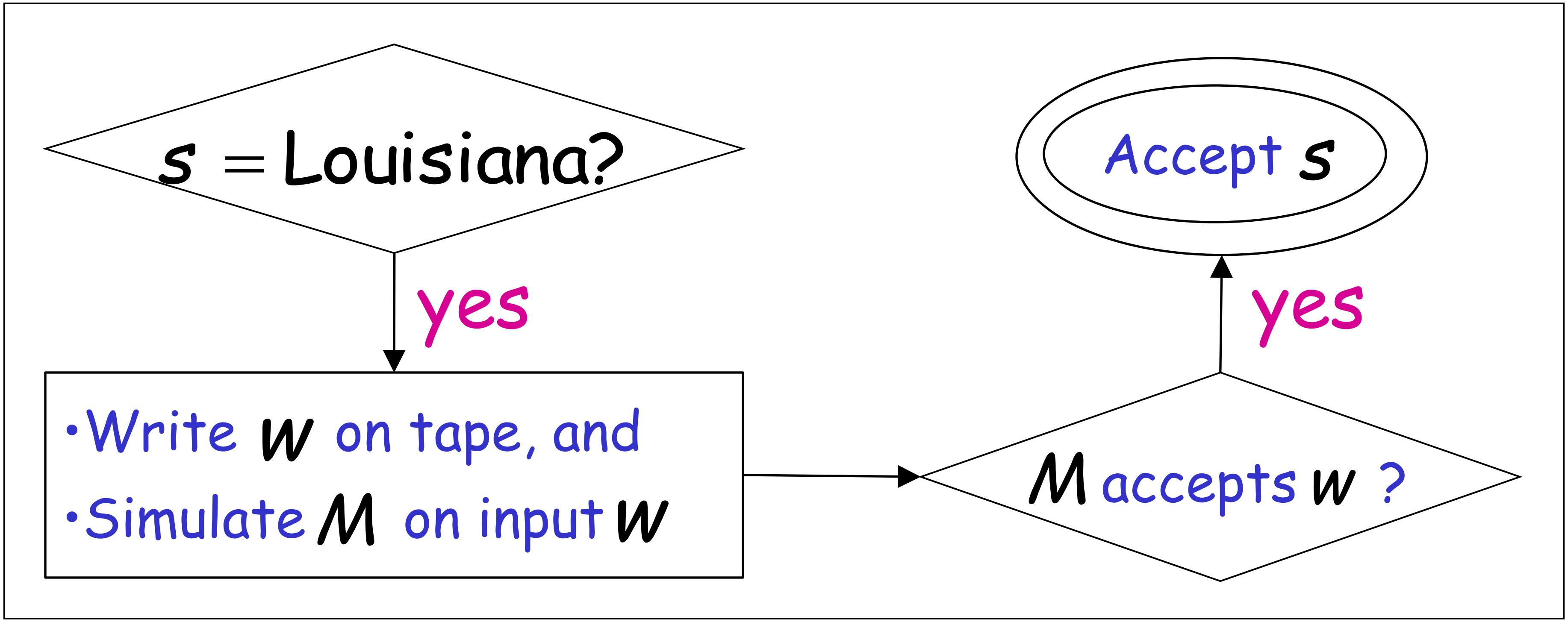
yes

Accept s

M accepts w ⇒ $L(\hat{M}) = \{\text{Louisiana}\} \neq \emptyset$

M does not accept w ⇒ $L(\hat{M}) = \emptyset$

Turing Machine $\hat{M}$



$s =$ Louisiana?

yes

- Write w on tape, and
- Simulate M on input w

M accepts w ?

yes

Accept s

Therefore:

$$M \text{ accepts } w \iff L(\hat{M}) \neq \emptyset$$

Equivalently:

$$\langle M, w \rangle \in A_{TM} \iff \langle \hat{M} \rangle \in \overline{EMPTY_{TM}}$$

END OF PROOF

Let L be a Turing-acceptable language

- L is empty?
- L is regular?
- L has size 2?

Regular language problem

Input: Turing Machine M

Question: Is $L(M)$ a regular language?

Corresponding language:

$$REGULAR_{TM} = \{\langle M \rangle : M \text{ is a Turing machine that accepts a regular language}\}$$

Theorem: $REGULAR_{TM}$ is undecidable

(regular language problem is unsolvable)

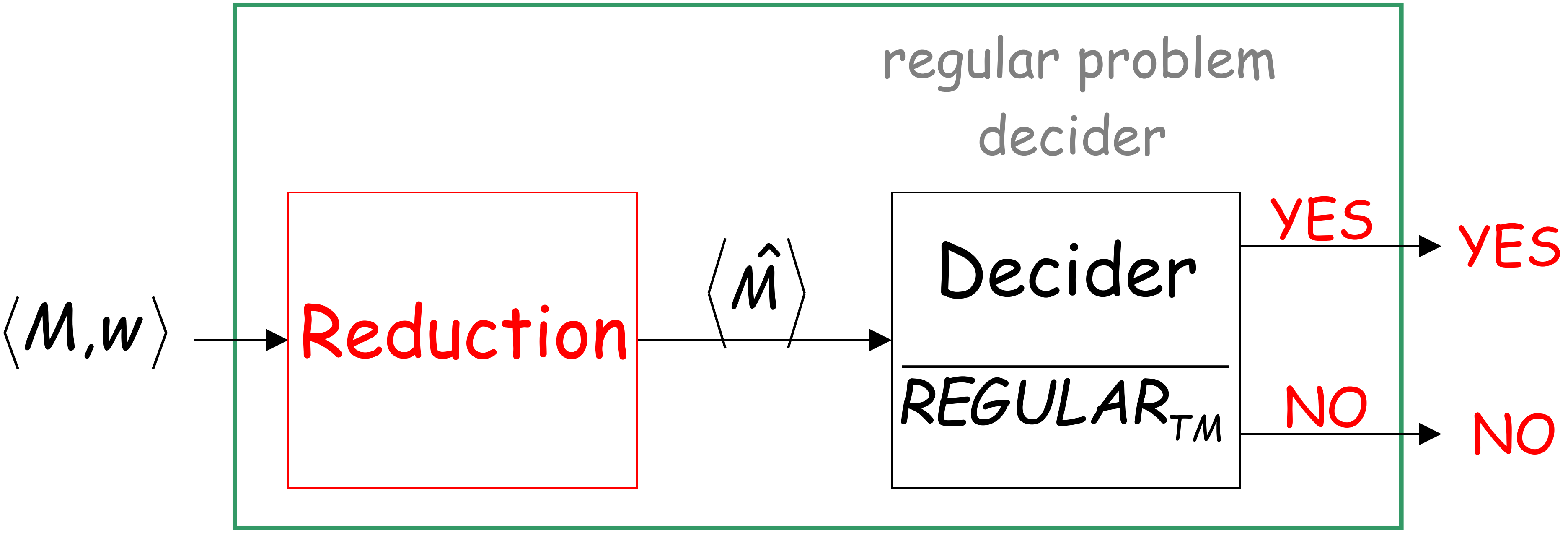
Proof: Reduce

A_{TM} (membership problem)

to

$\overline{REGULAR_{TM}}$ (regular language problem) membership problem decider

Decider for A_{TM}



regular problem decider

$\langle M,w \rangle$

Reduction

$\langle \hat{M} \rangle$

Decider

$\overline{REGULAR_{TM}}$

YES

NO

YES

NO

Given the reduction,
If $\overline{REGULAR_{TM}}$ is decidable,
then A_{TM} is decidable

A contradiction!
since A_{TM}
is undecidable

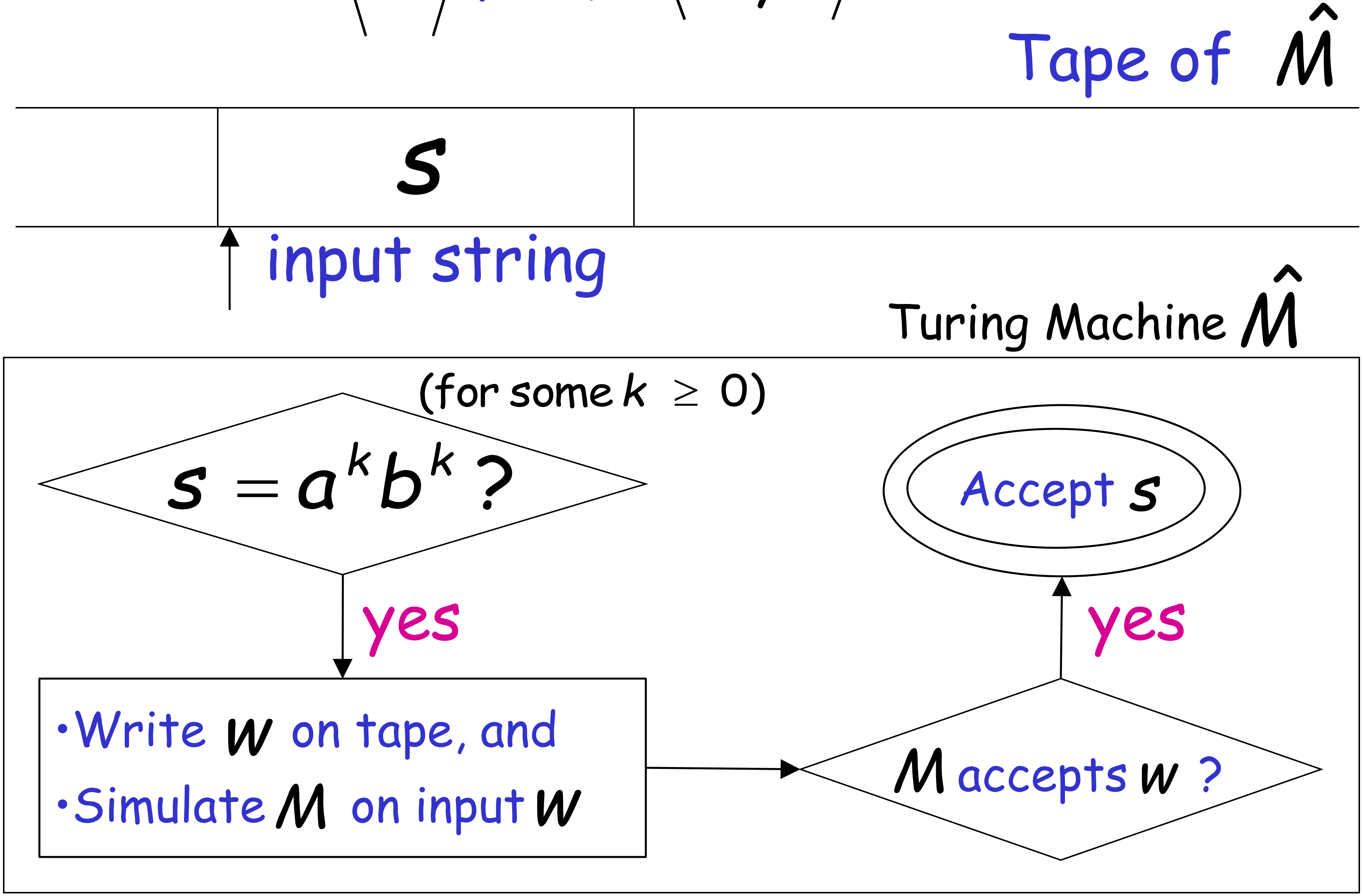
We only need to build the reduction:

$$\langle M,w\rangle \longrightarrow \boxed{\text{Reduction}} \longrightarrow \langle \hat{M}\rangle$$

So that:

$$\langle M,w\rangle \in A_{TM} \Longleftrightarrow \langle \hat{M}\rangle \in \overline{REGULAR_{TM}}$$

Construct $\langle \hat{M} \rangle$ from $\langle M, w \rangle$:



Tape of $\hat{M}$

s

input string

Turing Machine $\hat{M}$

(for some $k \geq 0$)

$s = a^k b^k$?

yes

- Write w on tape, and
- Simulate M on input w

M accepts w ?

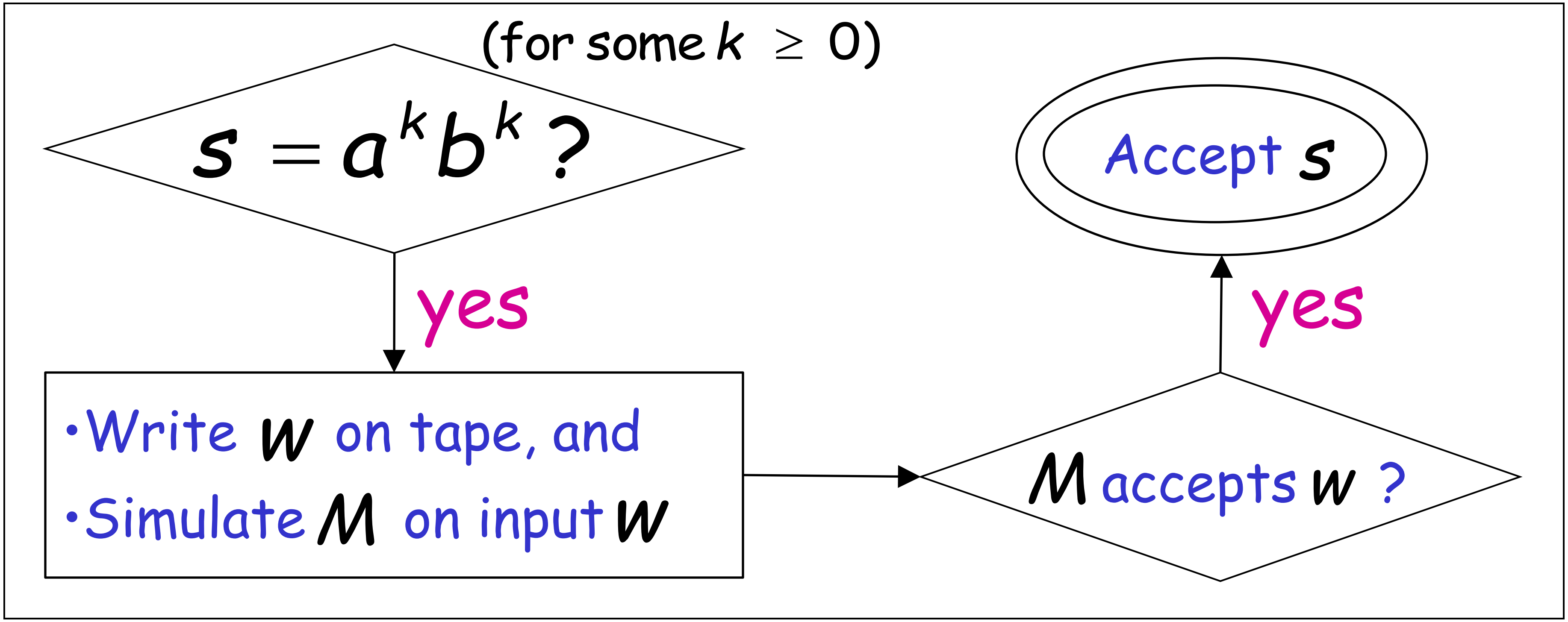
yes

Accept s

M accepts w $\Rightarrow$ $L(\hat{M}) = \{a^n b^n : n \geq 0\}$ not regular

M does not accept w $\Rightarrow$ $L(\hat{M}) = \emptyset$ regular

Turing Machine $\hat{M}$



Therefore:

M accepts w $\Longleftrightarrow$ $L(\hat{M})$ is not regular

Equivalently:

$\langle M, w \rangle \in A_{TM}$ $\Longleftrightarrow$ $\langle \hat{M} \rangle \in \overline{REGULAR_{TM}}$

END OF PROOF

Let L be a Turing-acceptable language

- L is empty?
- L is regular?
- L has size 2?

Size2 language problem

Input: Turing Machine M

Question: Does $L(M)$ have size 2 (two strings)?

$$|L(M)| = 2?$$

Corresponding language:

$$SIZE2_{TM} = \{\langle M \rangle : M \text{ is a Turing machine that accepts exactly two strings}\}$$

Theorem: $SIZE2_{TM}$ is undecidable

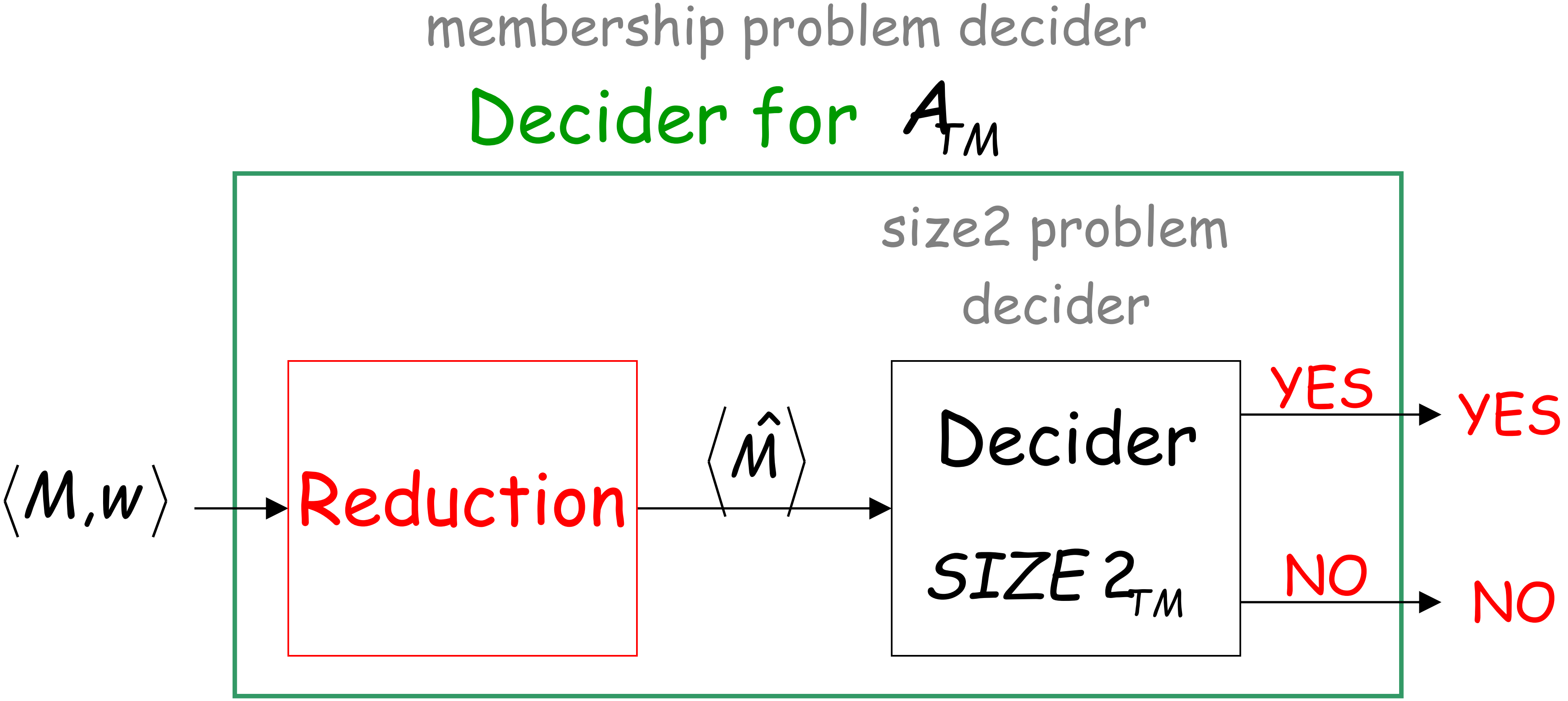
(size2 language problem is unsolvable)

Proof: Reduce

A_{TM} (membership problem)

to

$SIZE2_{TM}$ (size 2 language problem)



Given the reduction,
If $SIZE2_{TM}$ is decidable,
then A_{TM} is decidable

A contradiction!
since A_{TM}
is undecidable

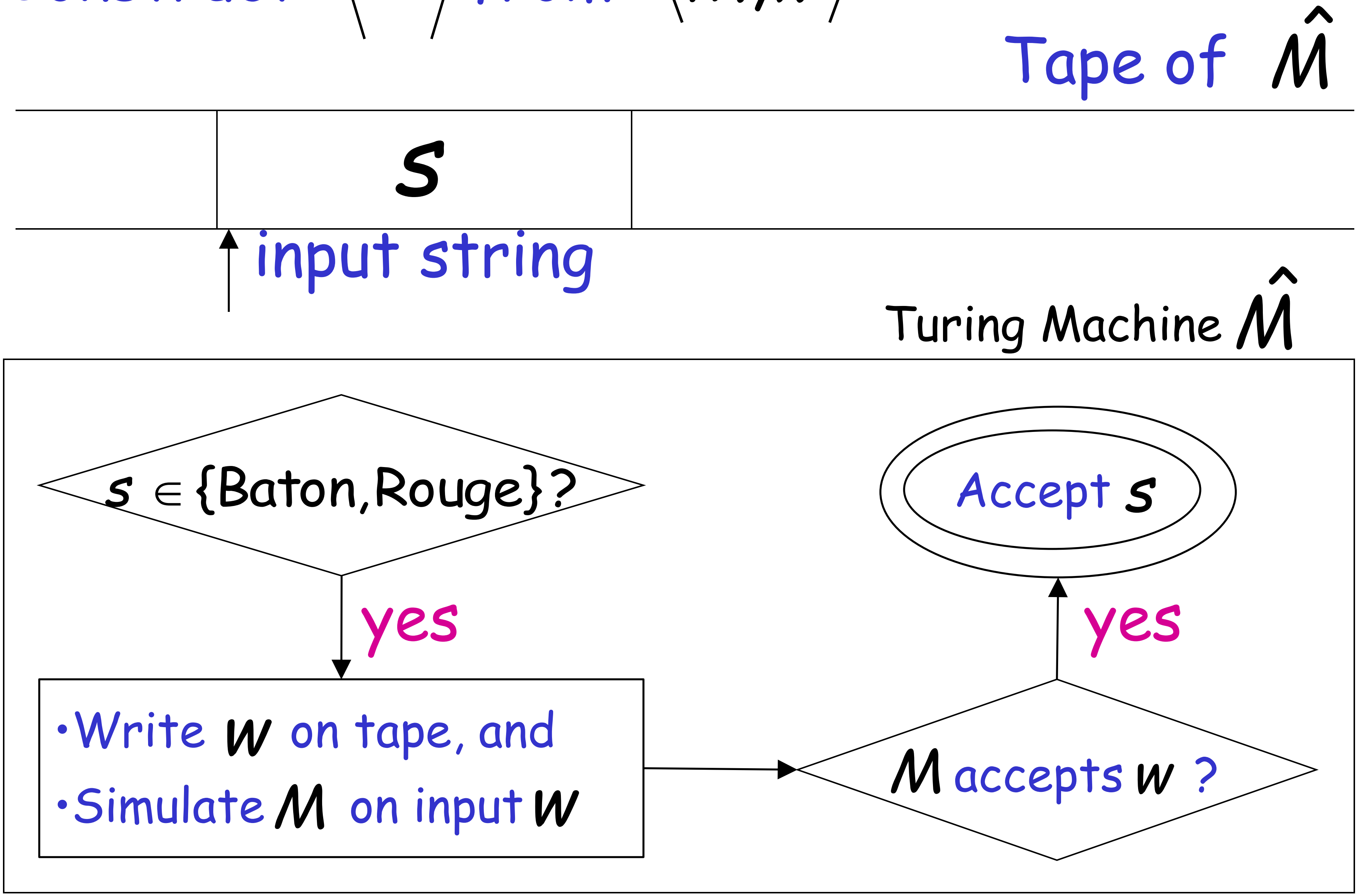
We only need to build the reduction:

$$\langle M,w\rangle \longrightarrow \boxed{\text{Reduction}} \longrightarrow \langle \hat{M}\rangle$$

So that:

$$\langle M,w\rangle \in A_{TM} \Longleftrightarrow \langle \hat{M}\rangle \in SIZE2_{TM}$$

Construct $\langle \hat{M} \rangle$ from $\langle M, w \rangle$:



Tape of $\hat{M}$

s

input string

Turing Machine $\hat{M}$

$s \in$ {Baton,Rouge}?

yes

- Write w on tape, and
- Simulate M on input w

M accepts w ?

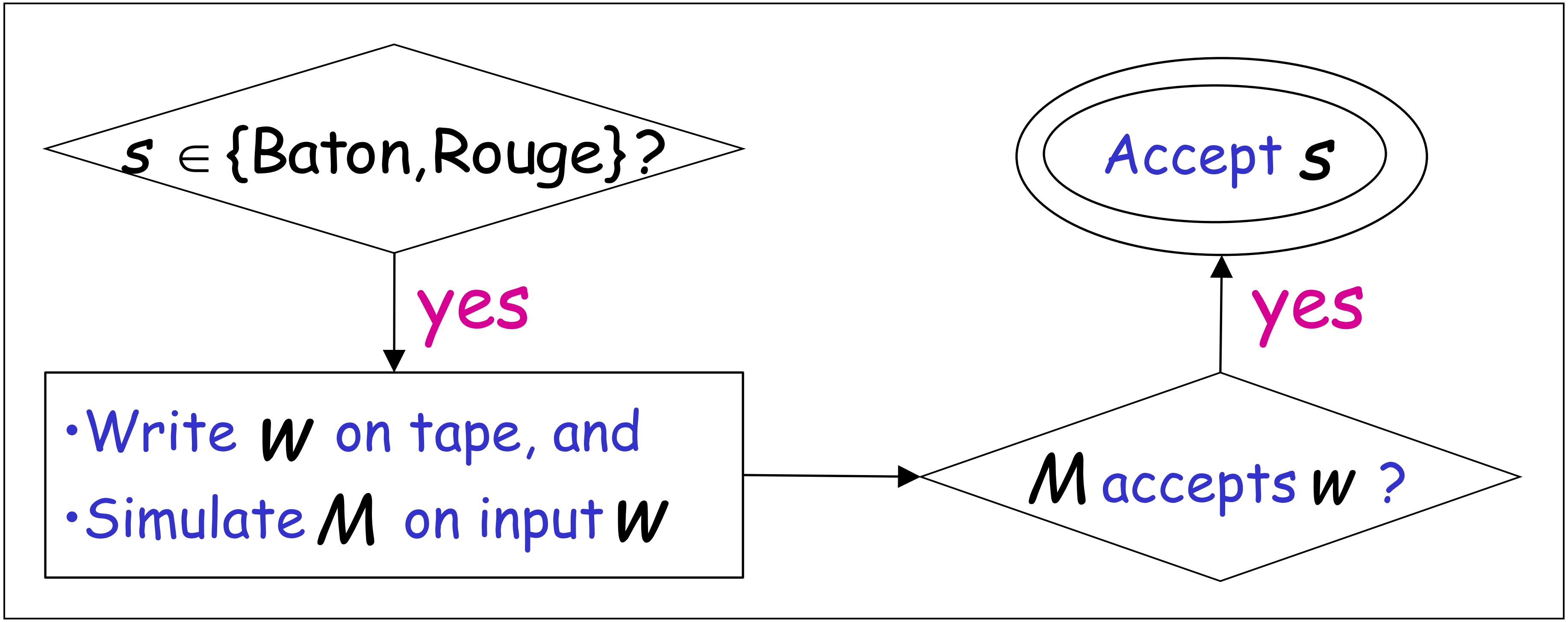
yes

Accept s

M accepts w ⇒ $L(\hat{M}) = \{Baton, Rouge\}$ 2 strings

M does not accept w ⇒ $L(\hat{M}) = \emptyset$ 0 strings

Turing Machine $\hat{M}$



$s \in \{Baton, Rouge\}$?

yes

- Write w on tape, and
- Simulate M on input w

M accepts w ?

yes

Accept s

Therefore:

M accepts w $\Longleftrightarrow$ $L(\hat{M})$ has size 2

Equivalently:

$$\langle M, w \rangle \in A_{TM} \Longleftrightarrow \langle \hat{M} \rangle \in SIZE2_{TM}$$

END OF PROOF