# Turing Machines

**Md. Khorshed Alam**

**Lecturer**

**CSE, NDUB**

# Introduction

- **Undecidable**: specific problems that cannot be solved using a computer

# Devices of Increasing Computational Power

- So far:
  - Finite Automata – good for devices with small amounts of memory, relatively simple control
  - Pushdown Automata – stack-based automata
- But both have limitations for even simple tasks, too restrictive as general purpose computers
- Enter the **Turing Machine**
  - More powerful than either of the above
  - Essentially a finite automaton but with unlimited memory
  - Although theoretical, can do everything a general purpose computer of today can do
    - If a TM can't solve it, neither can a computer

# Alan Turing

1912 (23 June): Birth, Paddington, London

1954 (7 June): Death (suicide) by cyanide poisoning, Wilmslow, Cheshire

# The Vision

- Alan Turing, regarded by some as the father of computer science, was among the first to envision the power of a computer and the importance of software over hardware
  - He knew the computer would mean unlimited potential in the creation and implementation of artificial intelligence and was determined to be the first to act on this belief.

# The Decision Problem

In 1928 the German mathematician, David Hilbert (1862-1943), asked whether there could be a mechanical way (i.e. by means of a fully specifiable set of instructions) of determining whether some statement in a formal system like arithmetic was provable or not.

In 1936 Turing published a paper the aim of which was to show that there was no such method.

"On computable numbers, with an application to the *Entscheidungs* problem." Proceedings of the London Mathematical Society, 2(42):230-265).

# The Turing Machine

In order to argue for this claim, he needed a clear concept of "mechanical procedure."

His idea — which came to be called the Turing machine — was this:

(1) A tape of infinite length

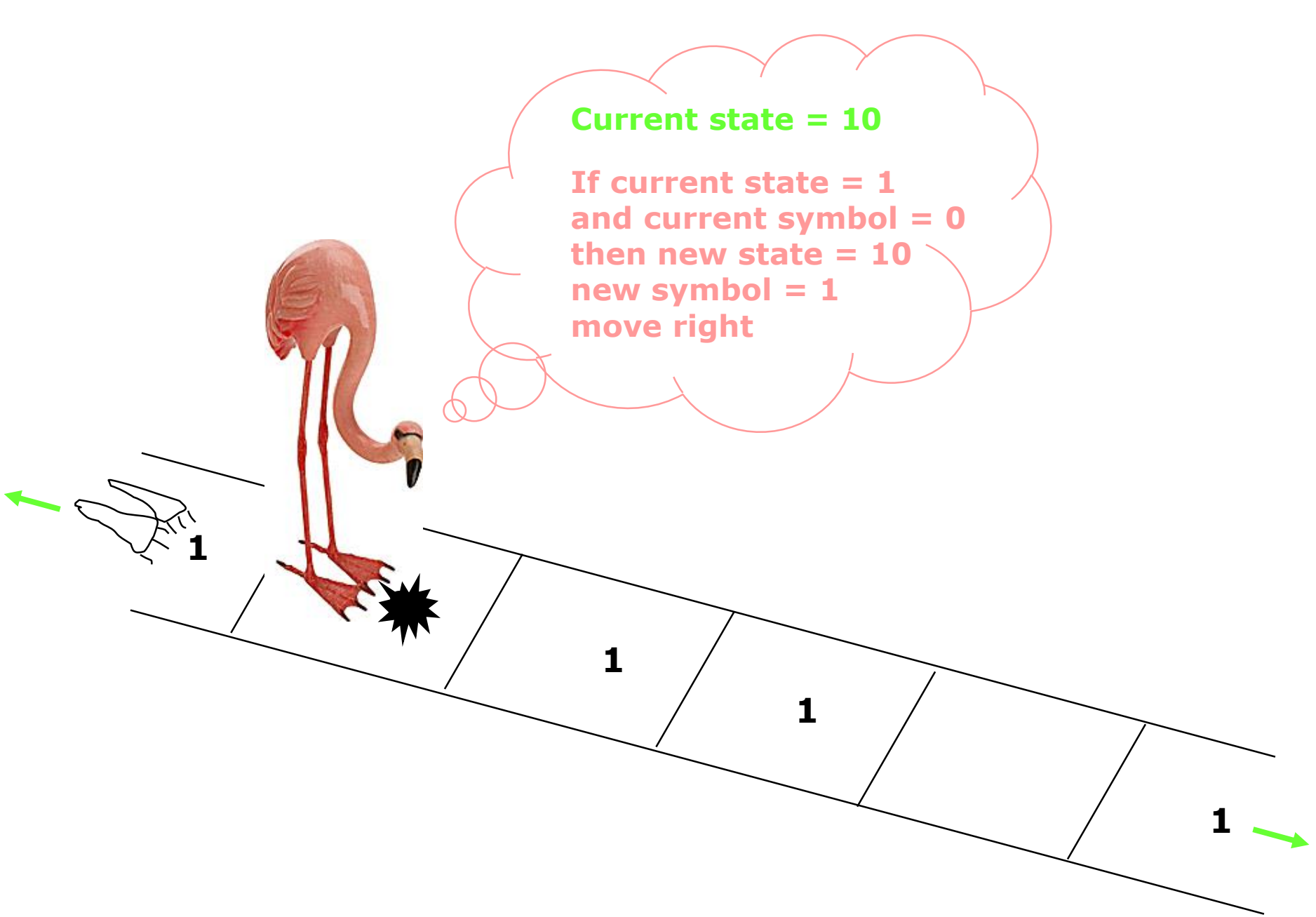
(2) Finitely many squares of the tape have a single symbol from a finite language.

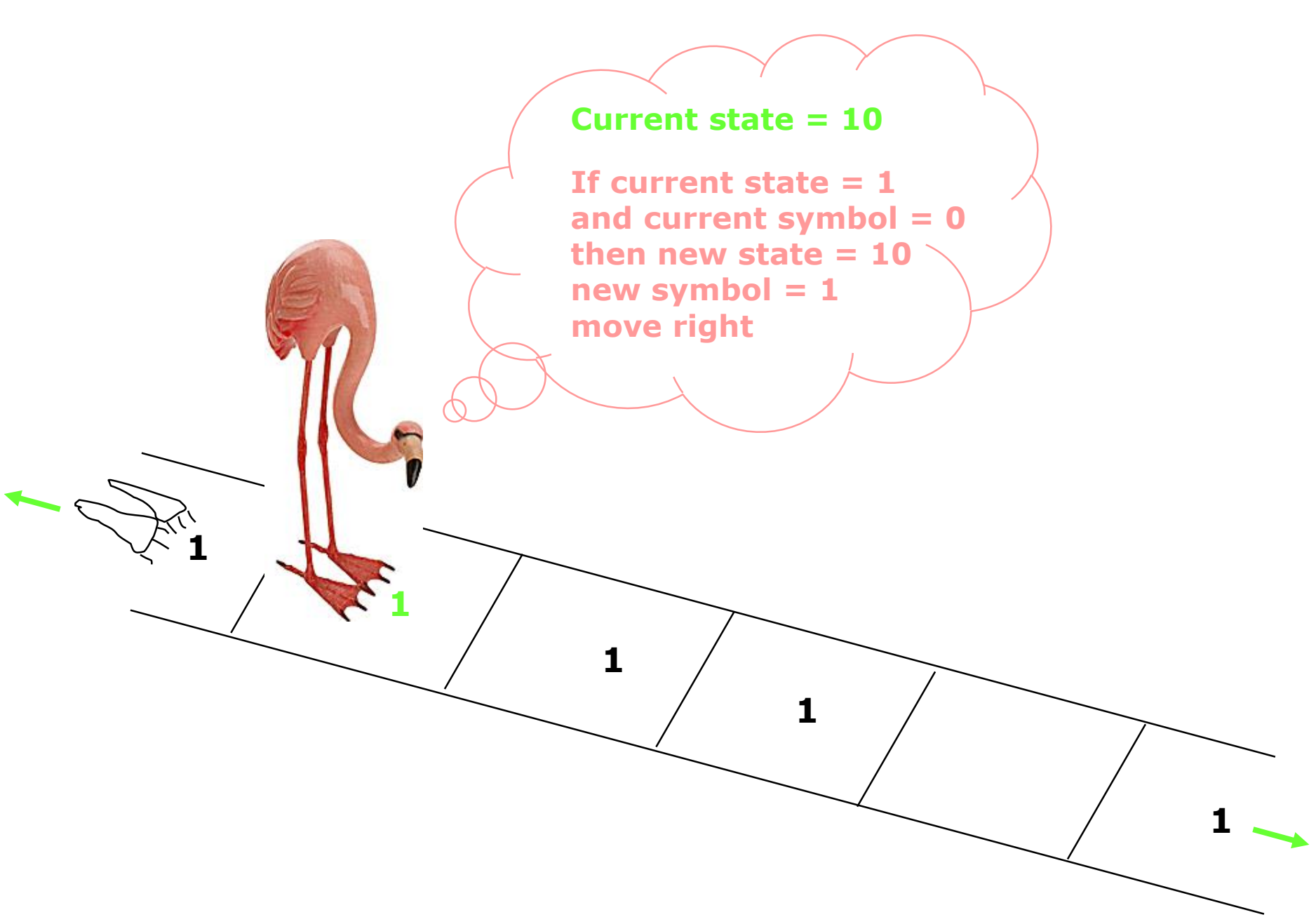
(3) Someone (or something) that can read the squares and write in them.

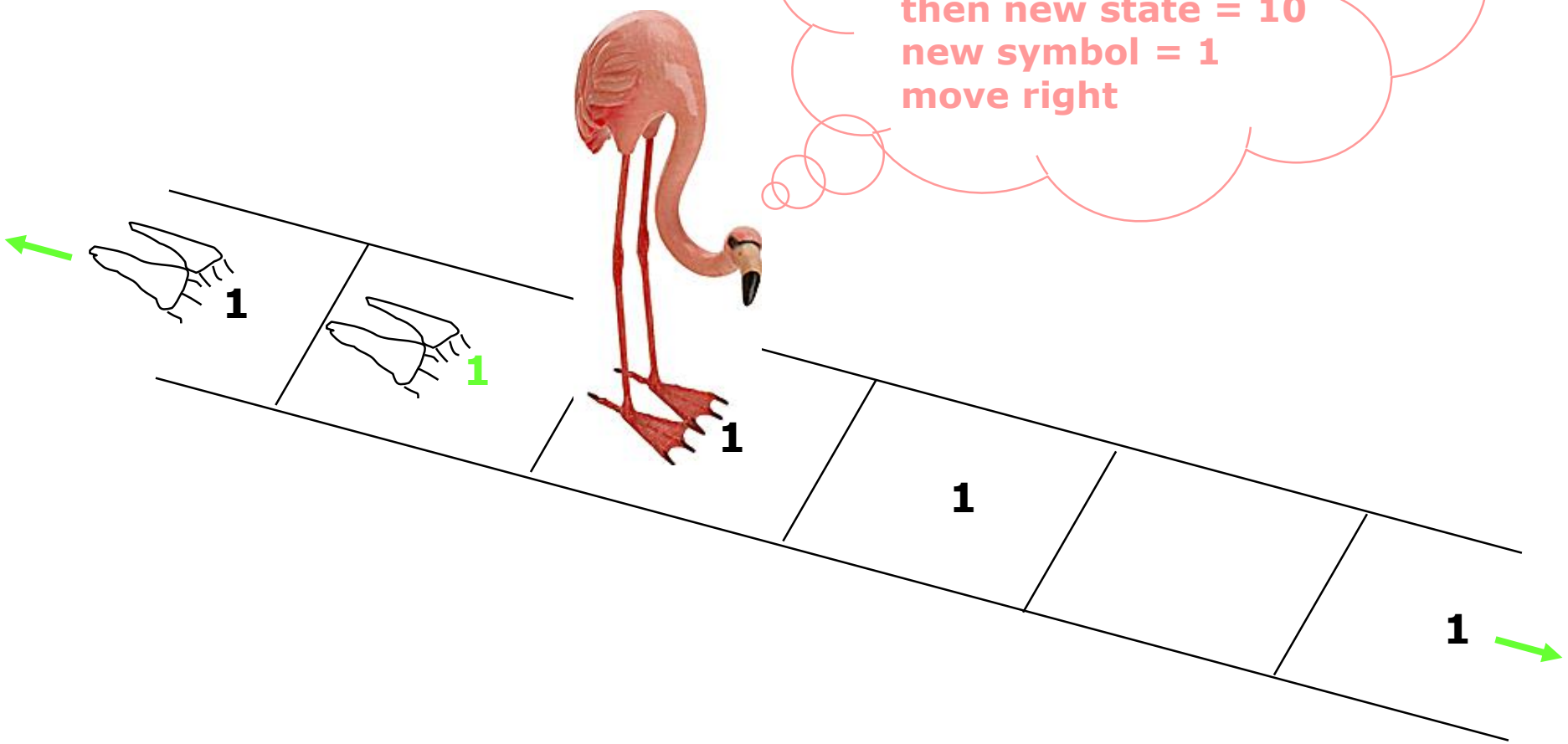
(4) At any time, the machine is in one of a finite number of internal states.

(5) The machine has instructions that determine what it does given its internal state and the symbol it encounters on the tape. It can

- change its internal state;
- change the symbol on the square;
- move forward;
- move backward;
- halt (i.e. stop).


Current state = 10
If current state = 1
and current symbol = 0
then new state = 10
new symbol = 1
move right
1
1
1
1


Current state = 10
If current state = 1
and current symbol = 0
then new state = 10
new symbol = 1
move right
1
1
1
1
1
1


Current state = 10
If current state = 1
and current symbol = 0
then new state = 10
new symbol = 1
move right
1
1
1
1
1

# Functions

It is essential to the idea of a Turing machine that it is not a physical machine, but an abstract one — a set of procedures.

It makes no difference whether the machine is embodied by a person in a boxcar on a track, or a person with a paper and pencil, or
a smart and well-trained flamingo.

# Turing's Theorem

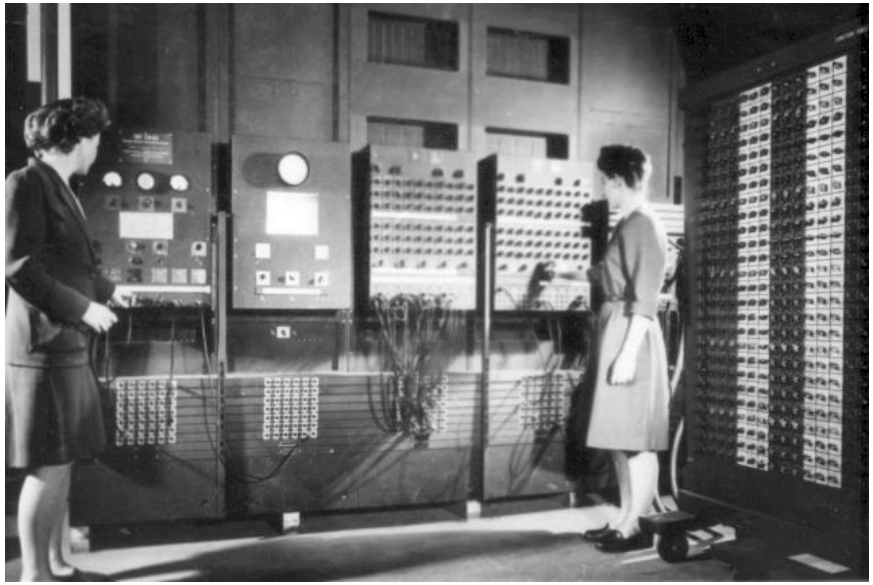
In the 1936 paper Turing proved that there are "general-purpose" Turing machines that can compute whatever any other Turing machine.

This is done by coding the function of the special-purpose machine as instructions of the other machine — that is by "programming" it. This is called Turing's theorem.

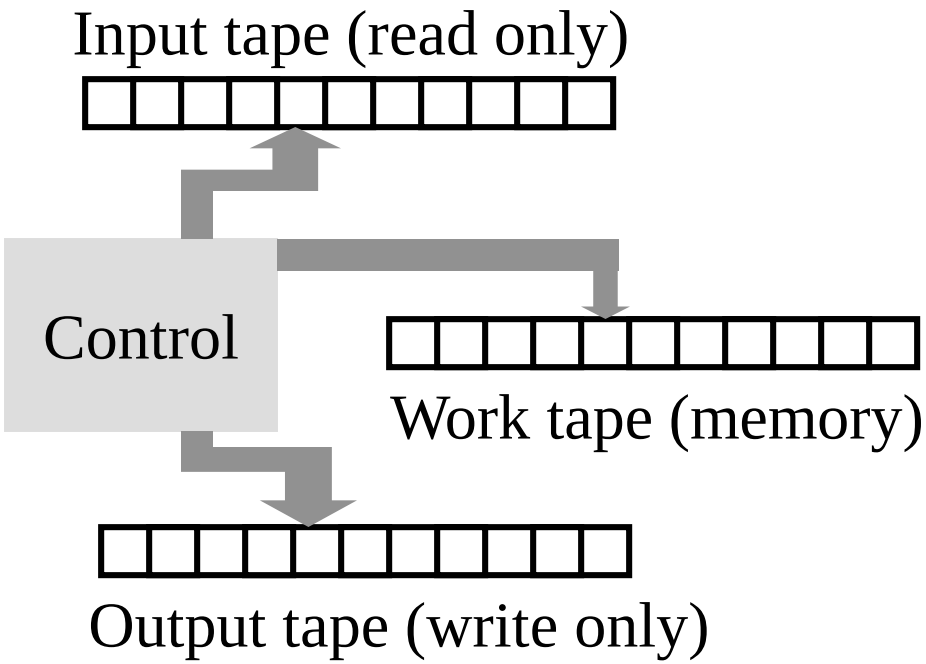
These are universal Turing machines, and the idea of a coding for a particular function fed into a universal Turing machine is basically our conception of a computer and a stored program.

The concept of the universal Turing machine is just the concept of the computer as we know it.

# First computers: custom computing machines



1950 -- Eniac: the control is hardwired manually for each problem.




1940: **VON NEUMANN:**
**DISTINCTION BETWEEN DATA AND INSTRUCTIONS**

# Can Machines Think?

In “Computing machinery and intelligence,” written in 1950, Turing asks whether machines can think.

He claims that this question is too vague, and proposes, instead, to replace it with a different one.

That question is: Can machines pass the “imitation game” (now called the Turing test)? If they can, they are intelligent.

Turing is thus the first to have offered a rigorous test for the determination of intelligence quite generally.

# The Turing Test

The game runs as follows. You sit at a computer terminal and have an electronic conversation. You don't know who is on the other end; it could be a person or a computer responding as it has been programmed to do.

If you can't distinguish between a human being and a computer from your interactions, then the computer is intelligent.

Note that this is meant to be a sufficient condition of intelligence only. There may be other ways to be intelligent.

# Artificial Intelligence

# The Church-Turning Thesis

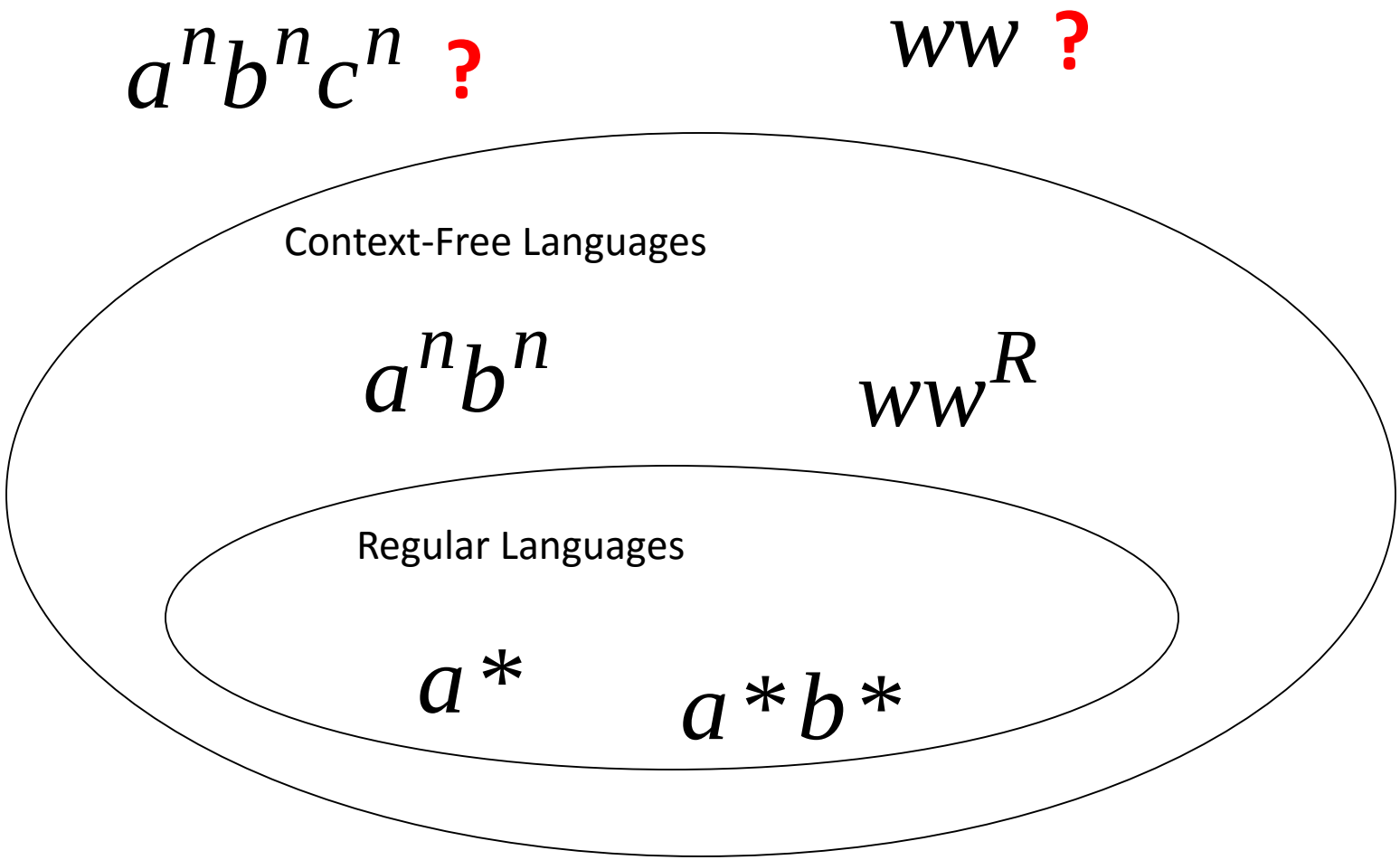
Turing, and a logician called Alonzo Church (1903-1995), independently developed the idea (not yet proven by widely accepted) that whatever can be computed by a mechanical procedure can be computed by a Turing machine.

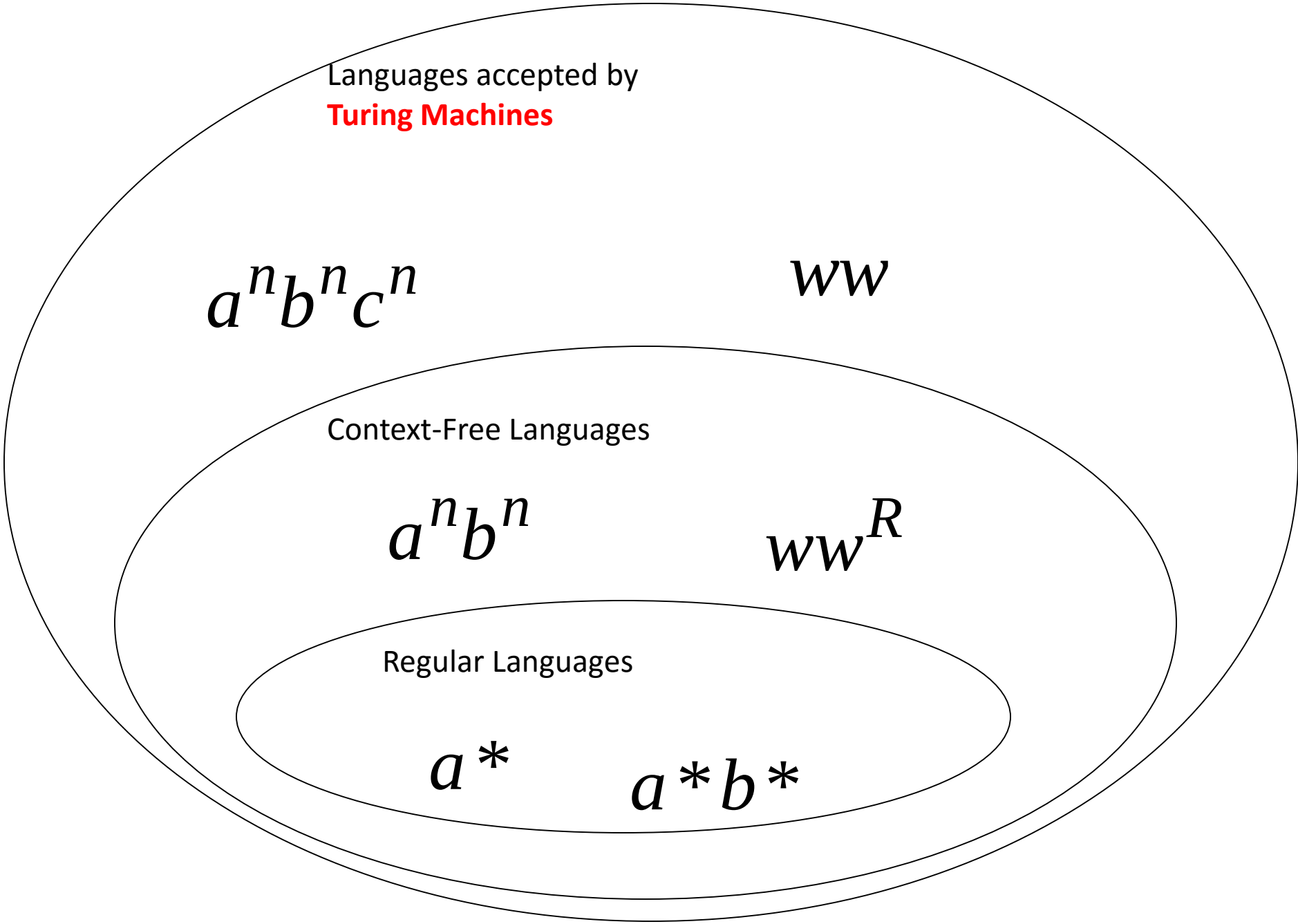
This is known as the Church-Turing thesis.

# Turing Machines

- TM's described in 1936
- A TM consists of a finite control (i.e. a finite state automaton) that is connected to an infinite tape.
- There is a tape divided into squares or cells
- Each cell can hold any one of a finite number of symbols

# The Language Hierarchy



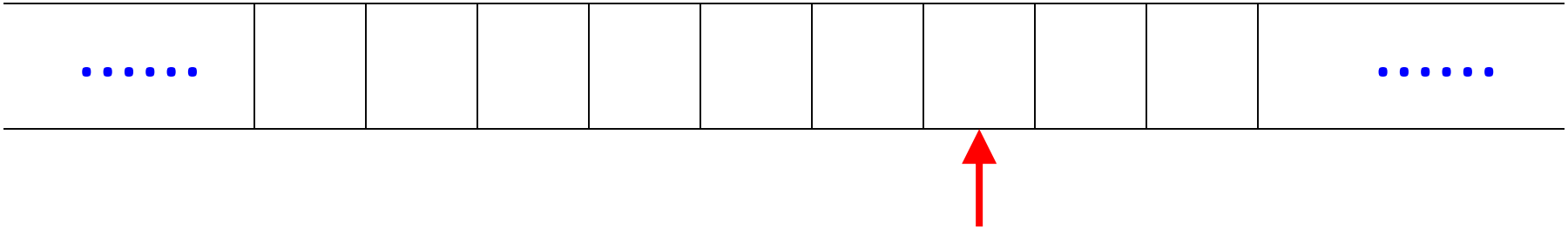

Languages accepted by
Turing Machines
$a^n b^n c^n$
$ww$
Context-Free Languages
$a^n b^n$
$ww^R$
Regular Languages
$a*$
$a*b*$

# Turing Machines

- TM's described in 1936
- A TM consists of a finite control (i.e. a finite state automaton) that is connected to an infinite tape.
- There is a tape divided into squares or cells
    - Each cell can hold any one of a finite number of symbols

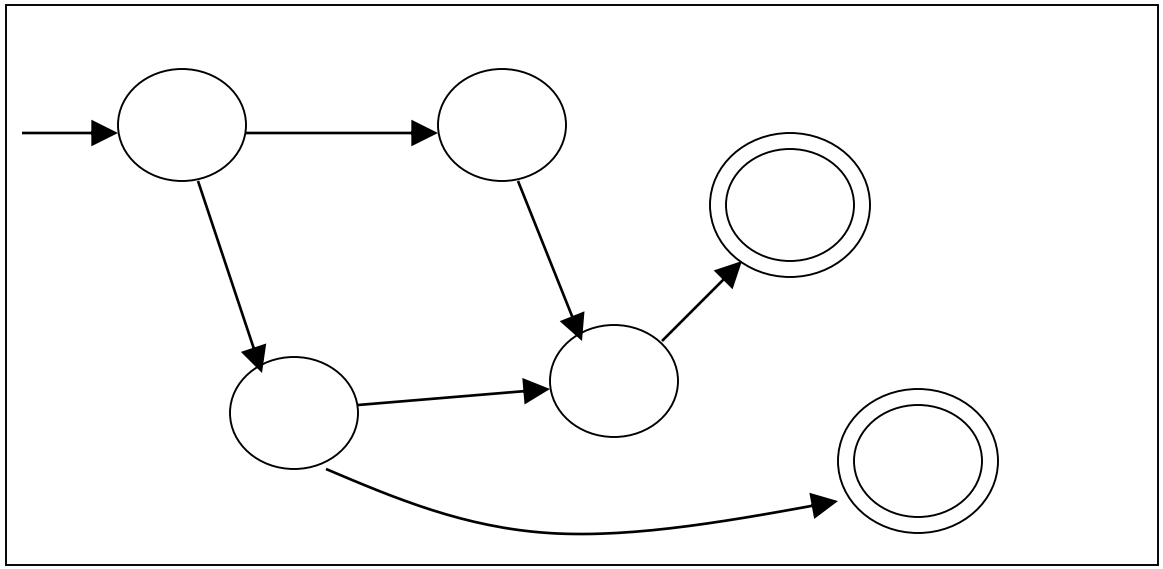
# A Turing Machine

Tape



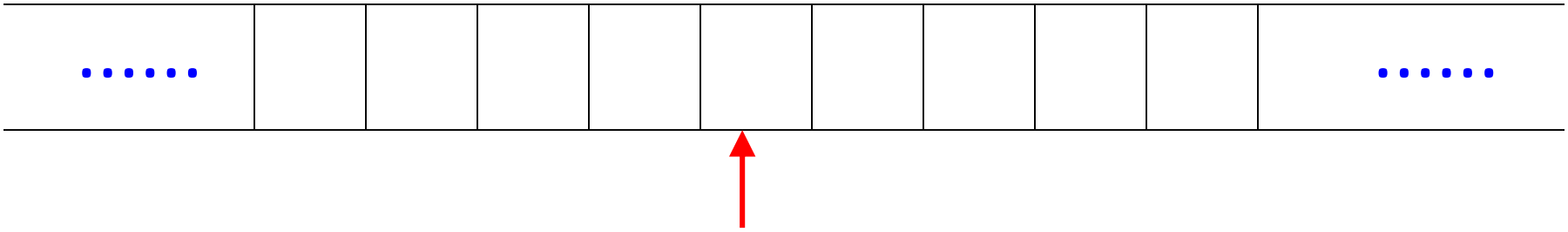

Read-Write head

Control Unit



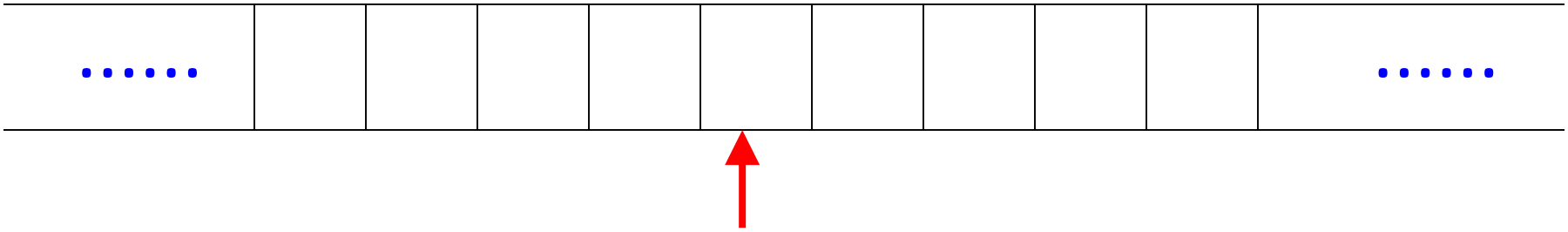
# The Tape

No boundaries -- infinite length




Read-Write head

The head moves Left or Right

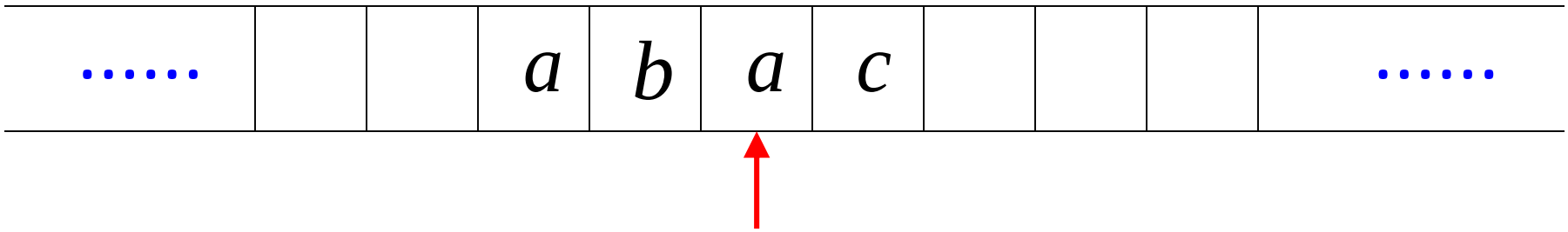



The head at each time step:

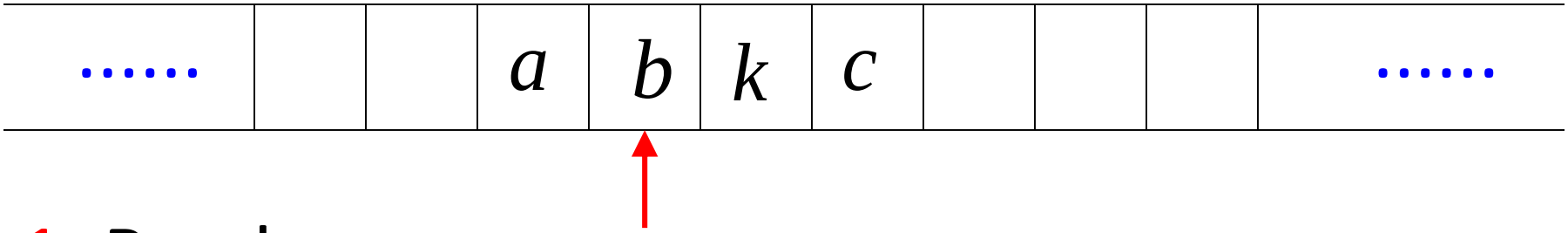
1. Reads a symbol
2. Writes a symbol
3. Moves Left or Right

Example:

Time 0

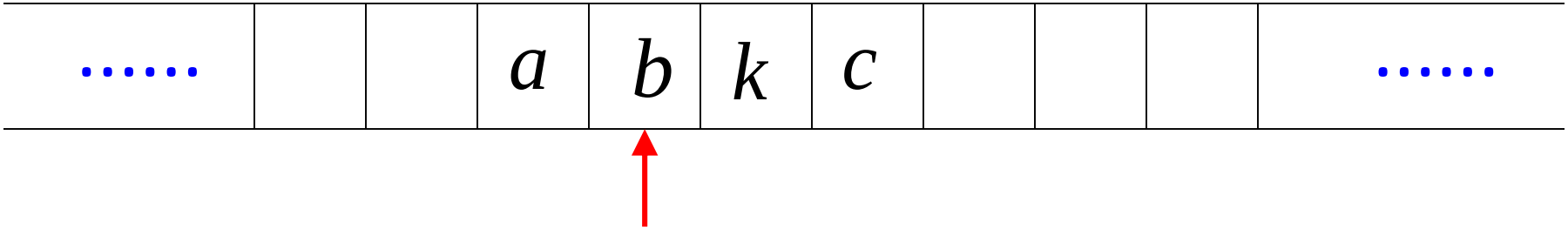



Time 1

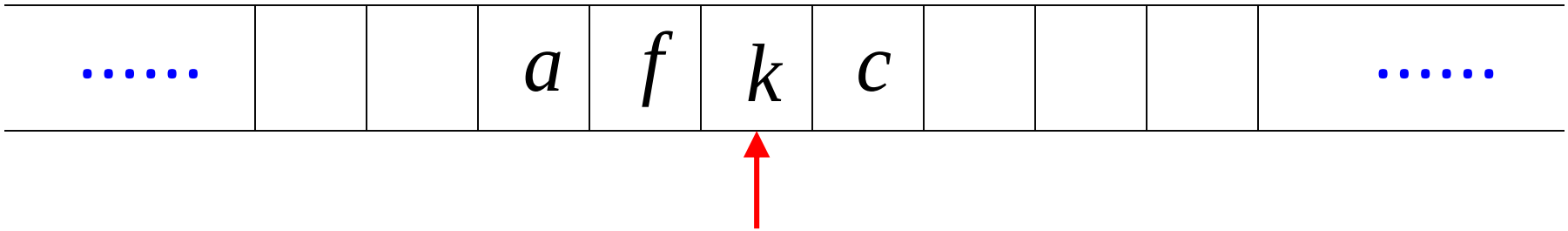



1. Reads $a$
2. Writes $k$
3. Moves Left

Time 1

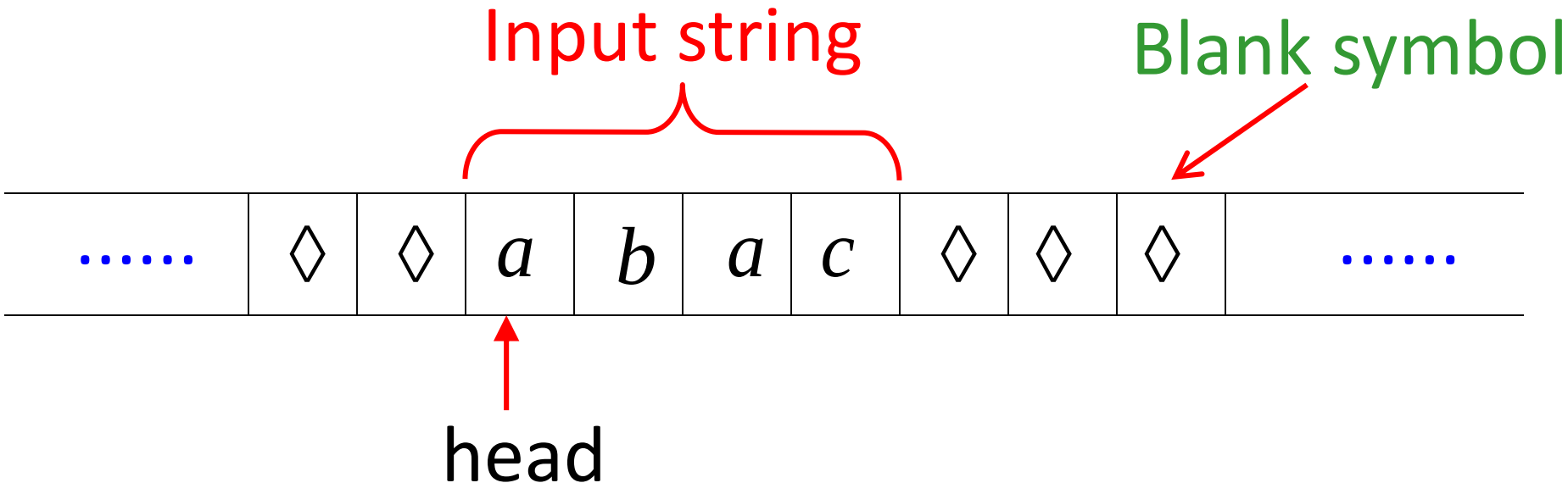



Time 2

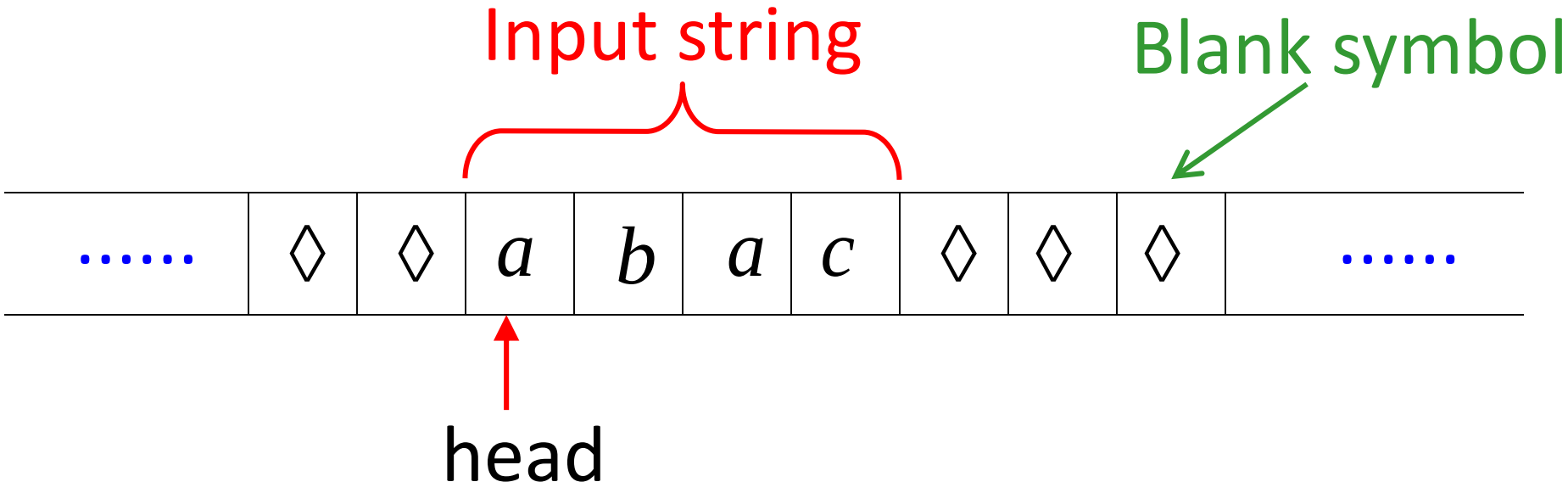



1. Reads $b$
2. Writes $f$
3. Moves Right

# The Input String



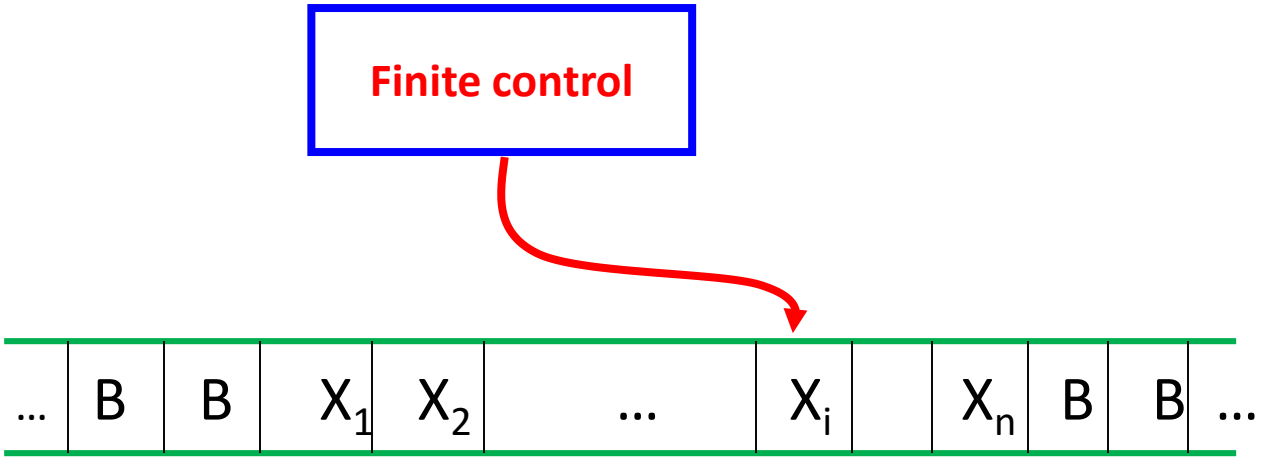

Head starts at the leftmost position of the input string




Remark: the input string is never empty

# Picture of a Turing Machine

- The tape consists of cells where each cell holds a symbol from the tape alphabet.
- Initially the *input* consists of a finite-length string of symbols and is placed on the tape.
- To the left of the input and to the right of the input, extending to infinity, are placed *blanks*.
- The *tape head* is initially positioned at the leftmost cell holding the input.

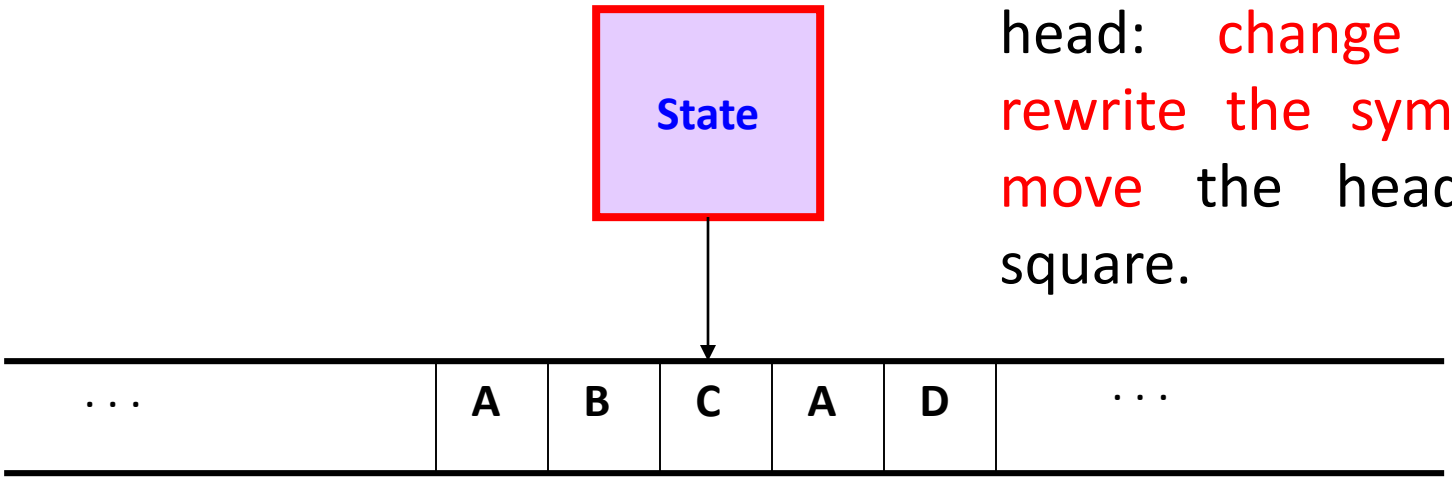


# Picture of a Turing Machine

- The TM is said to be scanning that cell. Initially the tape head is at the leftmost cell that holds the input
- **In one move the TM will:**
  - ❖Change state, which may be the same as the current state
  - ❖Write a tape symbol in the current cell, which may be the same as the current symbol
  - ❖Move the tape head left or right one cell
  - ❖The special states for rejecting and accepting take effect immediately

# Picture of a Turing Machine

**Action:** based on the state & the tape symbol under the head: change state, rewrite the symbol & move the head one square.




Infinite tape with squares containing tape symbols chosen from a finite alphabet

# Turing-Machine Formalism

- A TM is described by 07 tuples:

$$\mathbf{M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)}$$

1. A finite set of *states* (Q, typically).
2. An *input alphabet* ($\Sigma$, typically).
3. A *tape alphabet* ($\Gamma$, typically; contains $\Sigma$).
4. A *transition function* ($\delta$, typically).
5. A *start state* ($q_0$, in Q, typically).
6. A *blank symbol* (B, in $\Gamma - \Sigma$, typically).
   - All tape except for the input is blank initially.
7. A set of *final states* ($F \subseteq Q$, typically).

# Conventions

- a, b, … are input symbols.
- …, X, Y, Z are tape symbols.
- …, w, x, y, z are strings of input symbols.
- $\alpha$, $\beta$,… are strings of tape symbols.

# The Transition Function

- Takes two arguments:
  1. A state, in Q.
  2. A tape symbol in Γ.
- δ(q, Z) is either undefined or a triple of the form (p, Y, D).
  - p is the next state, in Q.
  - Y is the new tape symbol in Γ, written in the cell being scanned, replacing whatever symbol was there.
  - D is a *direction*, L or R (direction of head moves)

# Actions of the PDA

- If $\delta(q, Z) = (p, Y, D)$ then, in state q, scanning Z under its tape head, the TM:
  1. Changes the state to p.
  2. Replaces Z by Y on the tape.
  3. Moves the head one square in direction D.
     - D = L: move left; D = R; move right.

# Example: Turing Machine

- This TM scans its input right, looking for a 1.
- If it finds one, it changes it to a 0, goes to final state f, & halts.
- If it reaches a blank, it changes it to a 1 & moves left.

# Example: Turing Machine

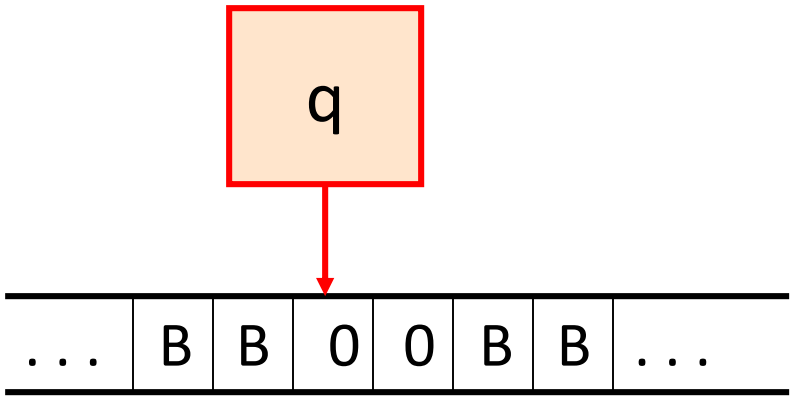
- States = {q (start), f (final)}.
- Input symbols = {0, 1}.
- Tape symbols = {0, 1, B}.
- $\delta(q, 0) = (q, 0, R)$.
- $\delta(q, 1) = (f, 0, R)$.
- $\delta(q, B) = (q, 1, L)$.

# Simulation of TM

$\delta(q, 0) = (q, 0, R)$

$\delta(q, 1) = (f, 0, R)$

$\delta(q, B) = (q, 1, L)$

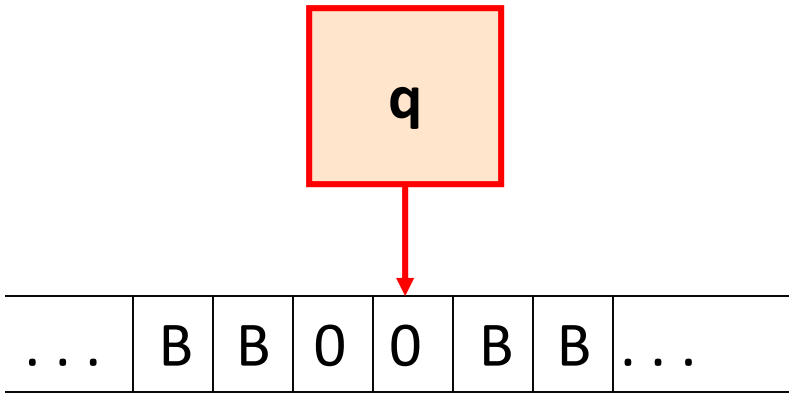


# Simulation of TM

$\delta(q, 0) = (q, 0, R)$

$\delta(q, 1) = (f, 0, R)$

$\delta(q, B) = (q, 1, L)$

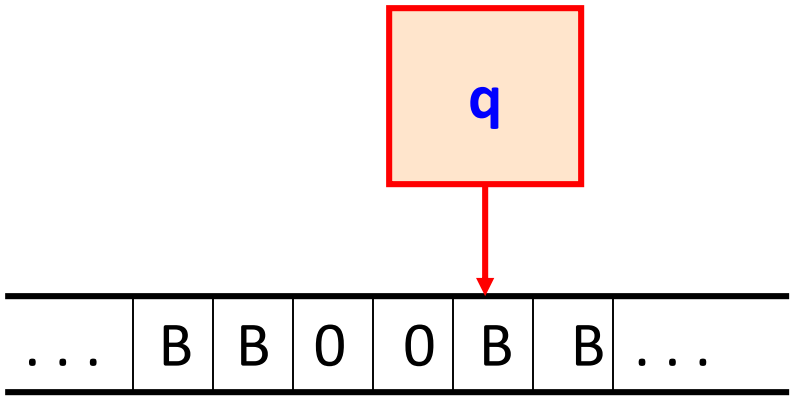


# Simulation of TM

$\delta(q, 0) = (q, 0, R)$

$\delta(q, 1) = (f, 0, R)$

$\delta(q, B) = (q, 1, L)$

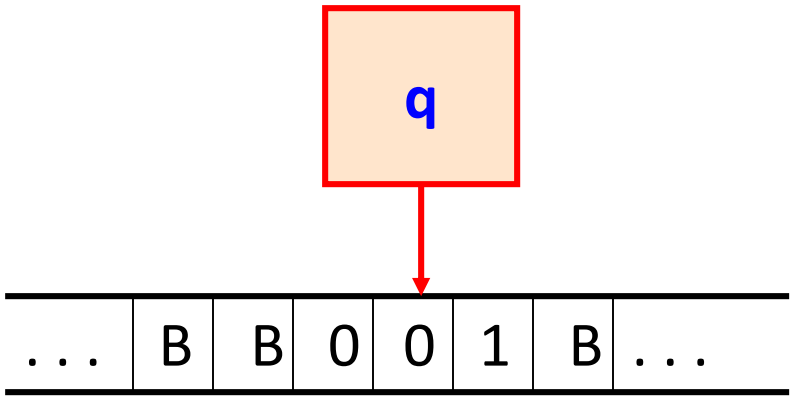


# Simulation of TM

$\delta(q, 0) = (q, 0, R)$

$\delta(q, 1) = (f, 0, R)$

$\delta(q, B) = (q, 1, L)$

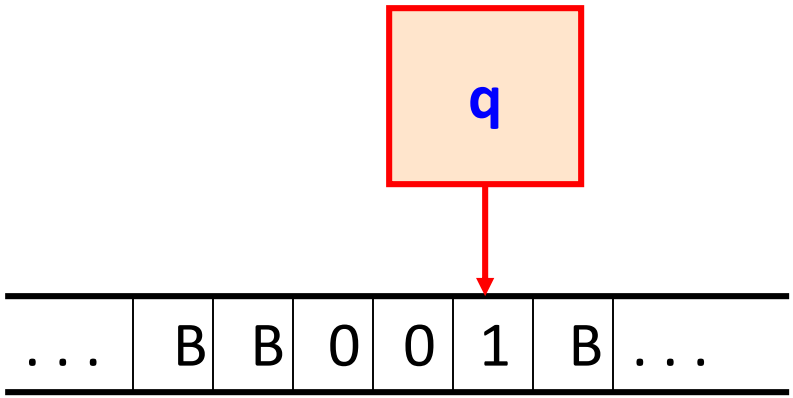


# Simulation of TM

$\delta(q, 0) = (q, 0, R)$

$\delta(q, 1) = (f, 0, R)$

$\delta(q, B) = (q, 1, L)$

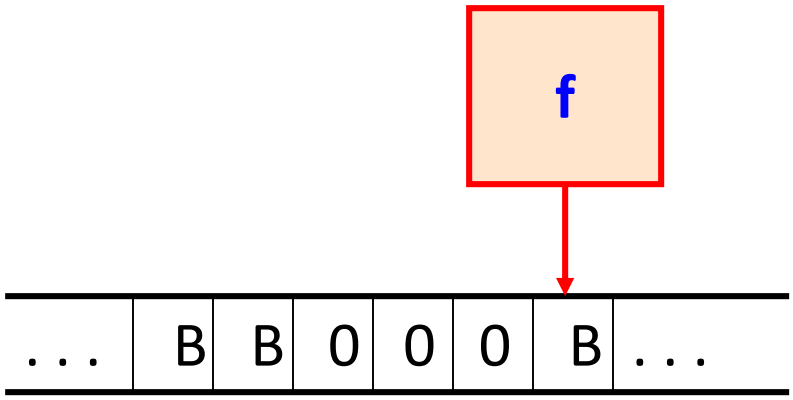


# Simulation of TM

$\delta(q, 0) = (q, 0, R)$

$\delta(q, 1) = (f, 0, R)$

$\delta(q, B) = (q, 1, L)$




No move is possible. The TM halts and accepts.

# Instantaneous Descriptions of a Turing Machine

- Initially, a TM has a tape consisting of a string of input symbols surrounded by an infinity of blanks in both directions.
- The TM is in the start state, and the head is at the leftmost input symbol.

# TM ID's

- An ID is a string $\alpha q \beta$, where $\alpha\beta$ is the tape between the leftmost and rightmost nonblanks (inclusive).
- The state q is immediately to the left of the tape symbol scanned.
- If q is at the right end, it is scanning B.
  - If q is scanning a B at the left end, then consecutive B's at and to the right of q are part of $\alpha$.

# TM ID's

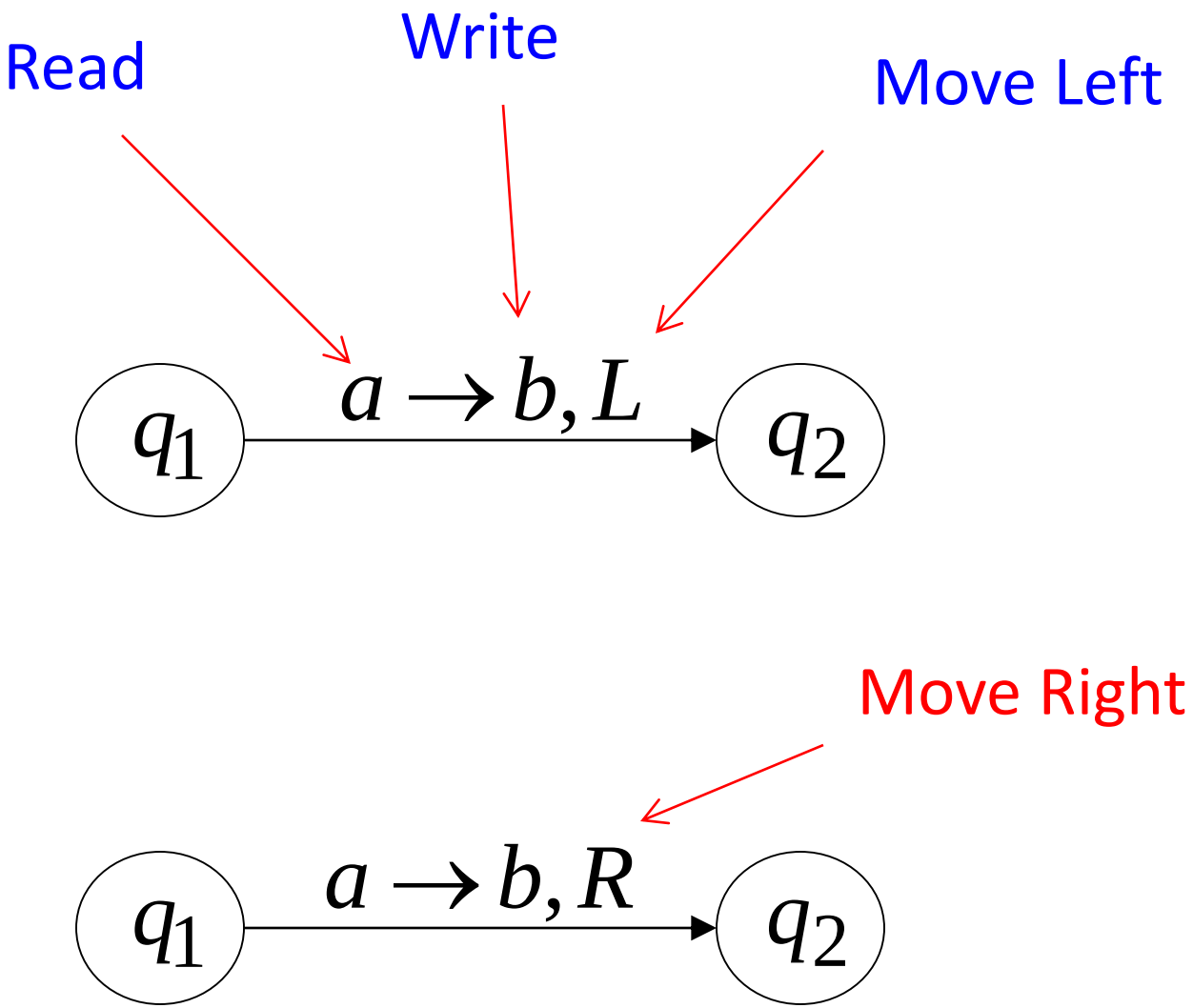
- As for PDA's we may use symbols $\vdash$ and $\vdash^*$ to represent "becomes in one move" & "becomes in zero or more moves,"
- Example: The moves of the previous TM are

**$q00 \vdash 0q0 \vdash 00q \vdash 0q01 \vdash 00q1 \vdash 000f$**

# Formal Definition of Moves

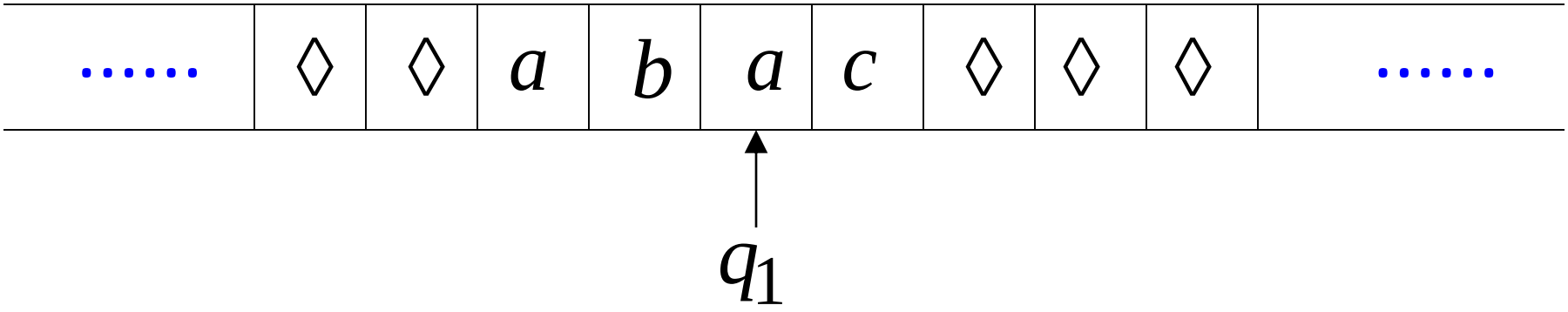
1. If **$\delta(q, Z) = (p, Y, R)$,** then
   - $\alpha qZ\beta \vdash \alpha Yp\beta$
   - If Z is the blank B, then also $\alpha q \vdash \alpha Yp$
2. If $\delta(q, Z) = (p, Y, L)$, then
   - For any X, $\alpha XqZ\beta \vdash \alpha pXY\beta$
   - In addition, $qZ\beta \vdash pBY\beta$

# States & Transitions

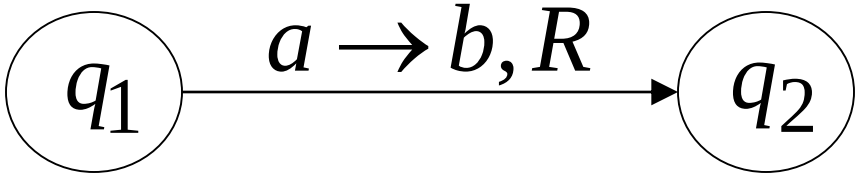


# Example:

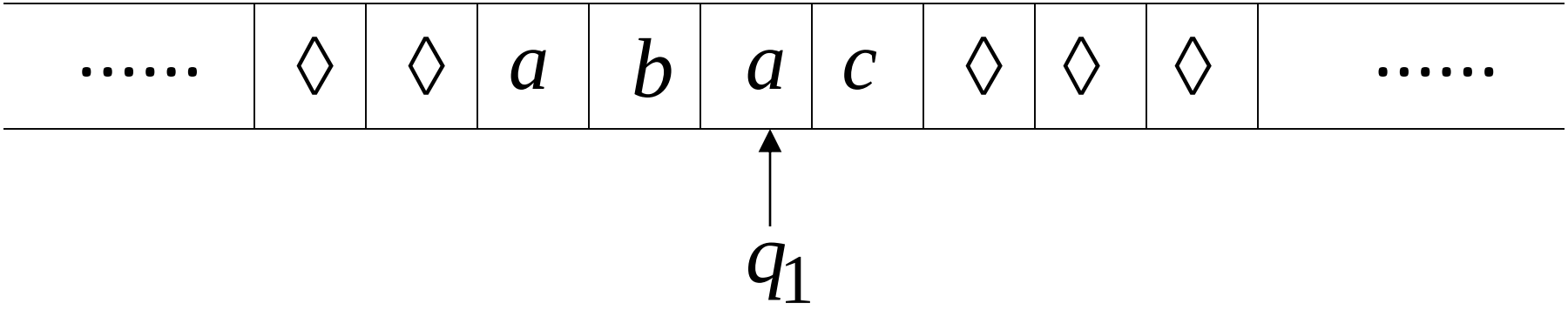
Time 1



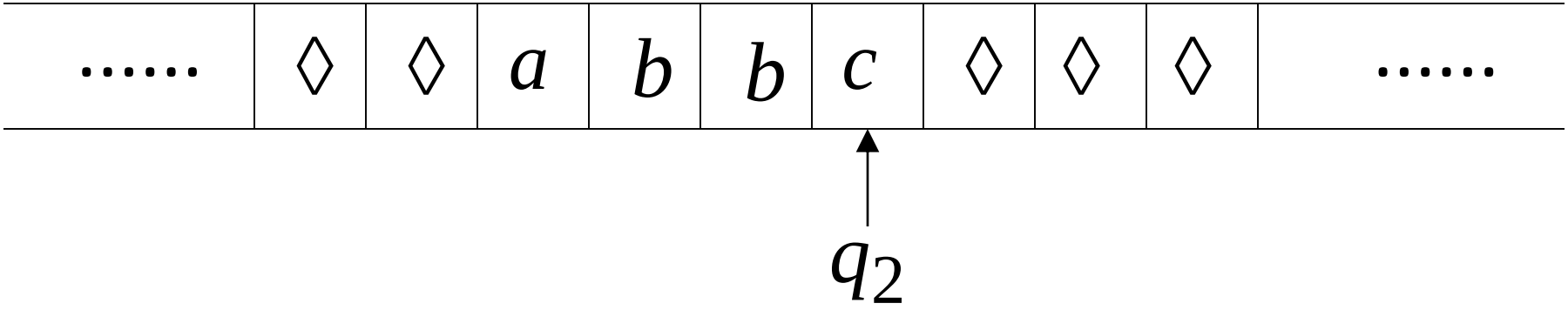

current state

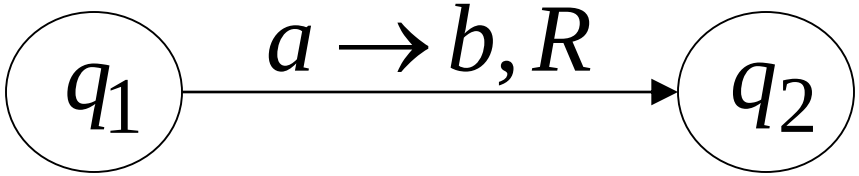


Time 1

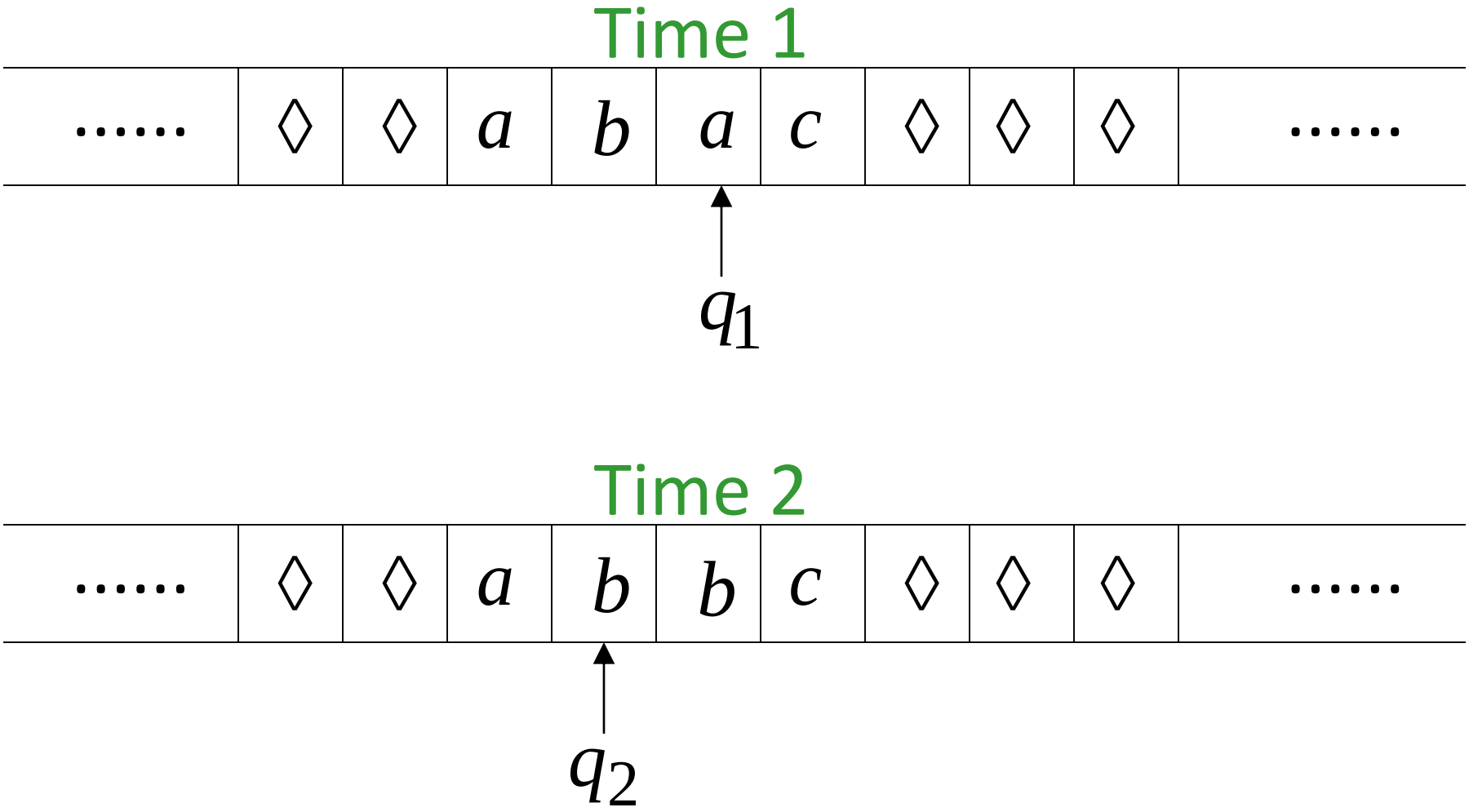


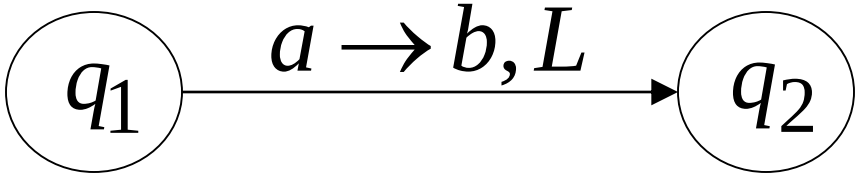

Time 2



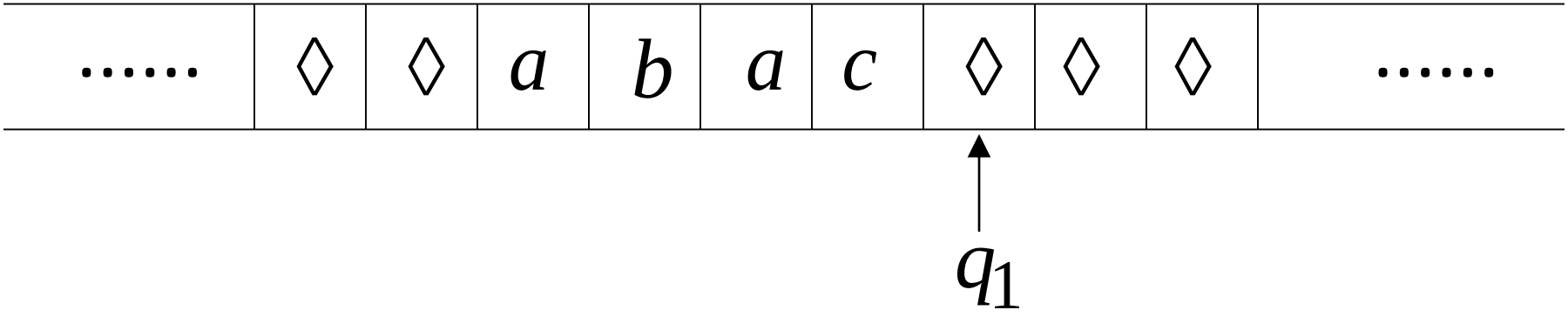
# Example:



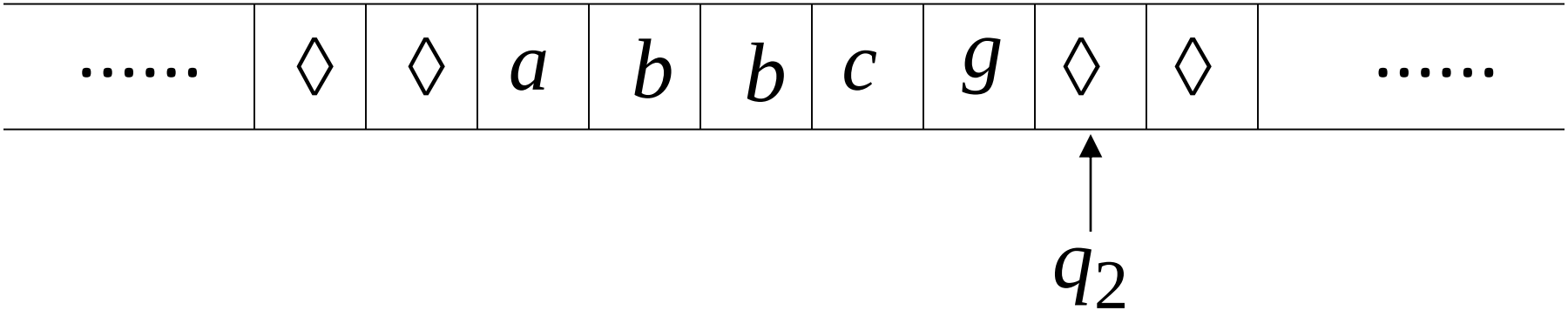



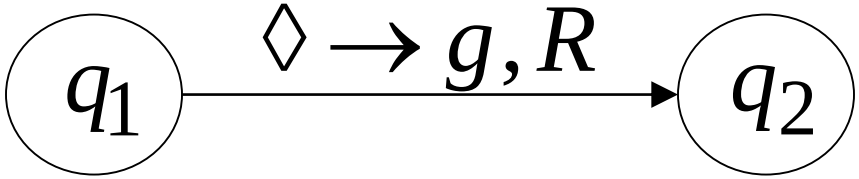
# Example:

Time 1




Time 2

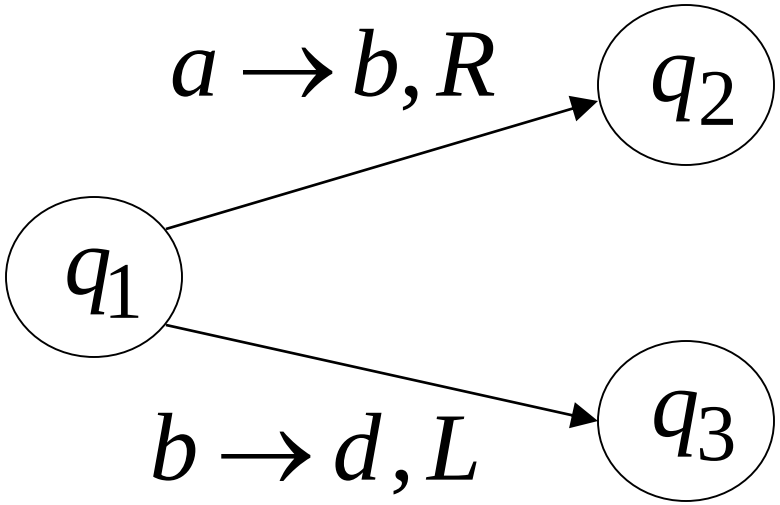


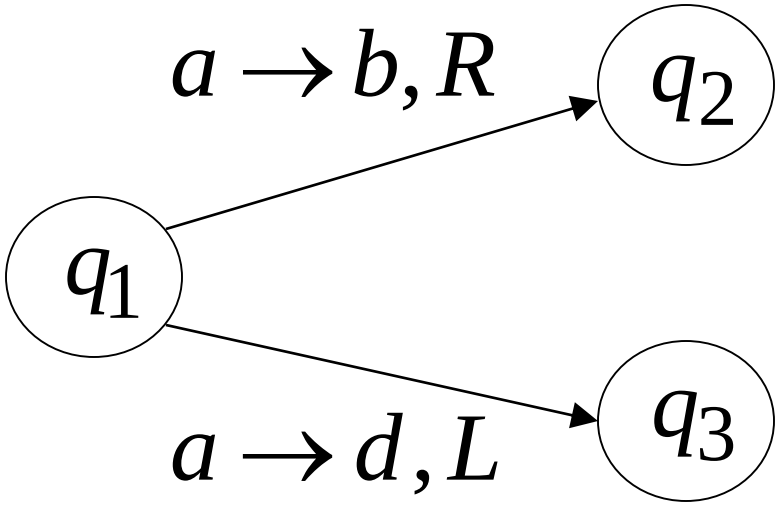
# Determinism

Turing Machines are deterministic

Allowed



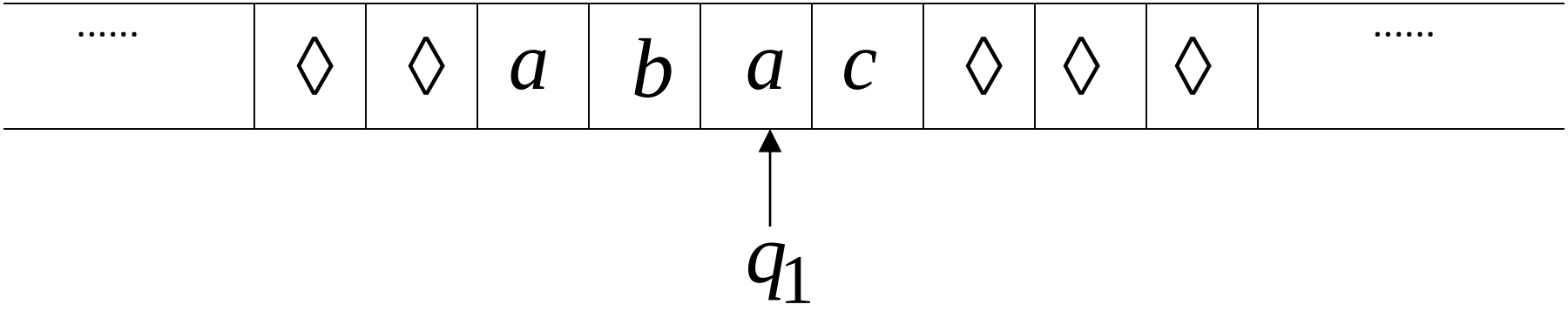

**Not** Allowed

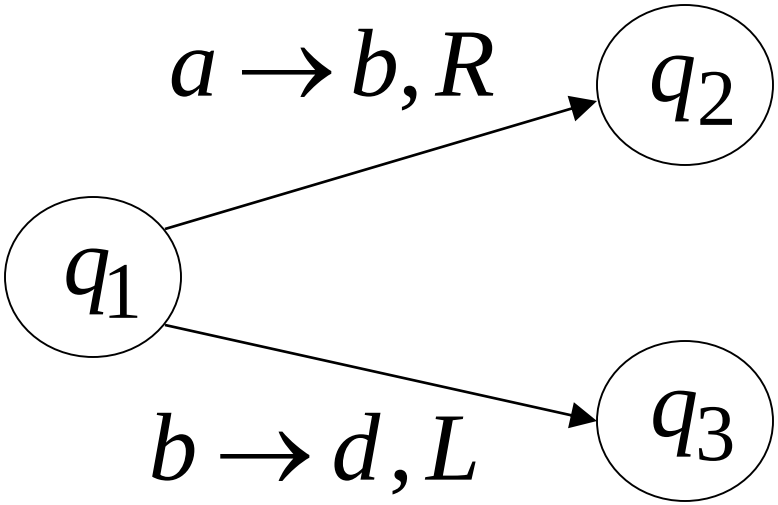



No lambda transitions allowed

# Partial Transition Function

Example:



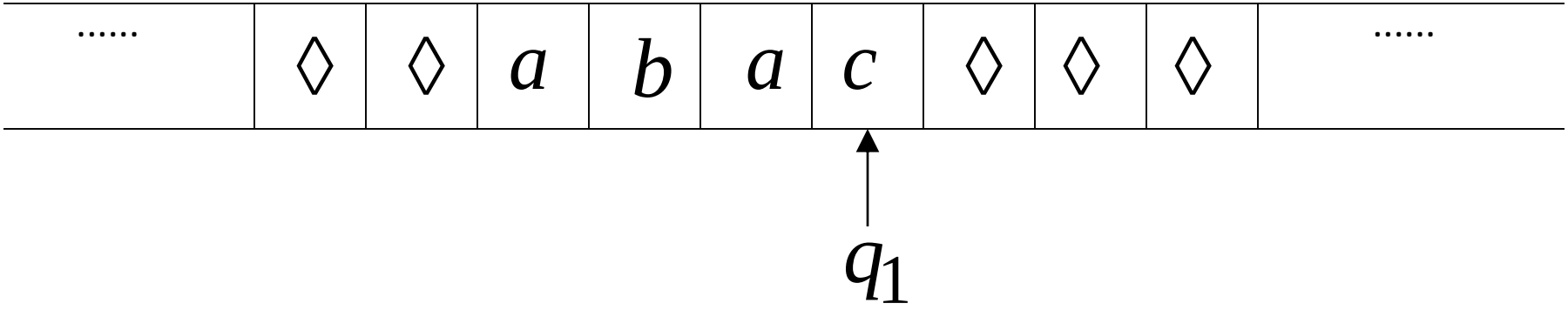

Allowed:

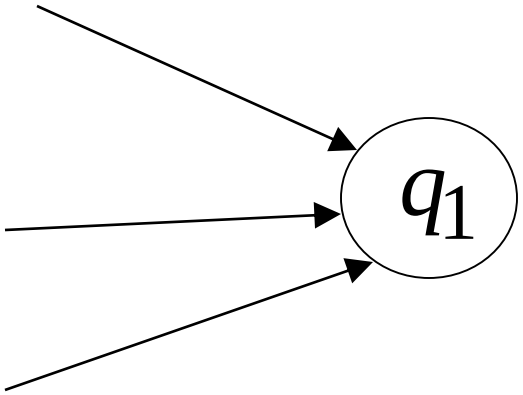
No transition
for input symbol $c$

# Halting

The machine ***halts*** if there are no possible transitions to follow

# Halting Example 1:

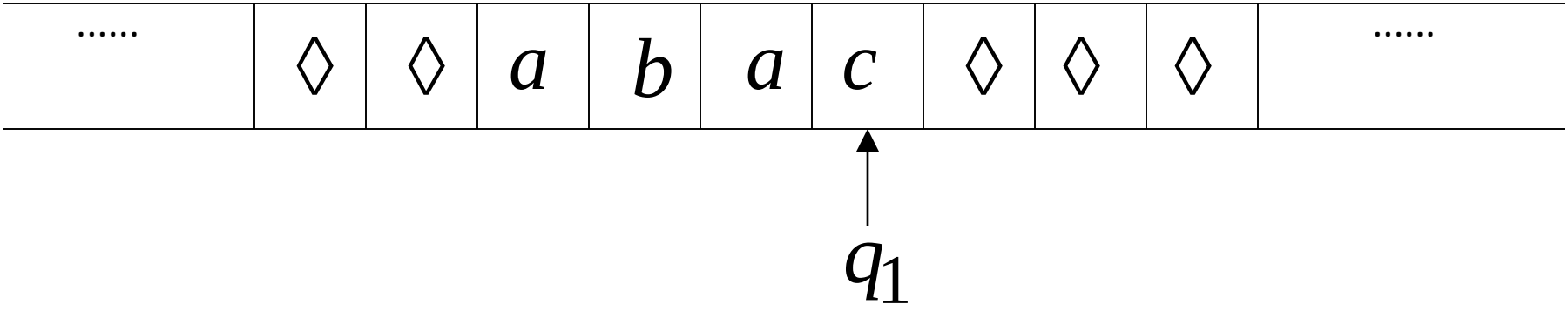


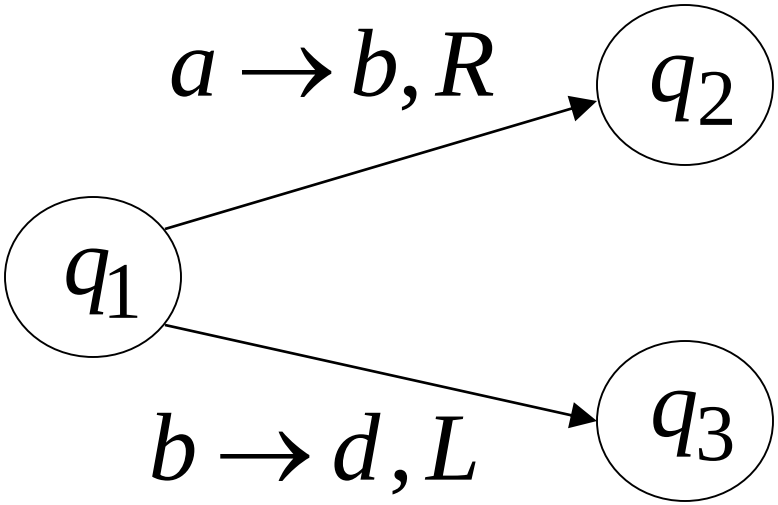




No transition from $q_1$

**HALT!!!**

# Example:

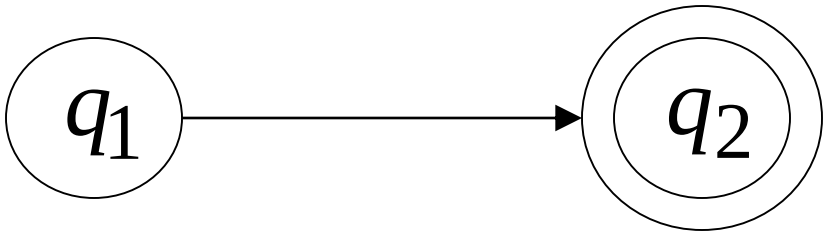


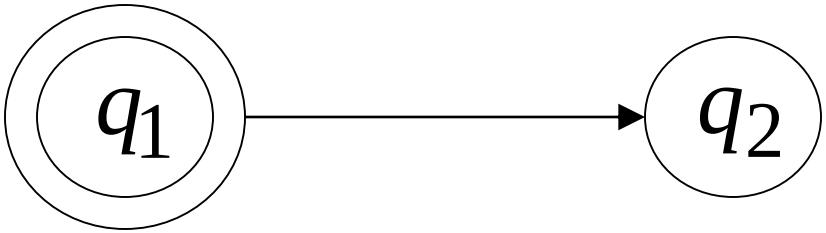

No possible transition

**HALT!!!**

# Final States




Allowed

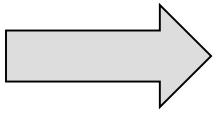



**Not** Allowed

- Final states have no outgoing transitions
- In a final state the machine halts

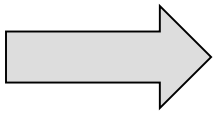
# Acceptance

Accept Input



If machine halts
in a final state

Reject Input



If machine halts
in a non-final state

or

If machine enters
an *infinite loop*

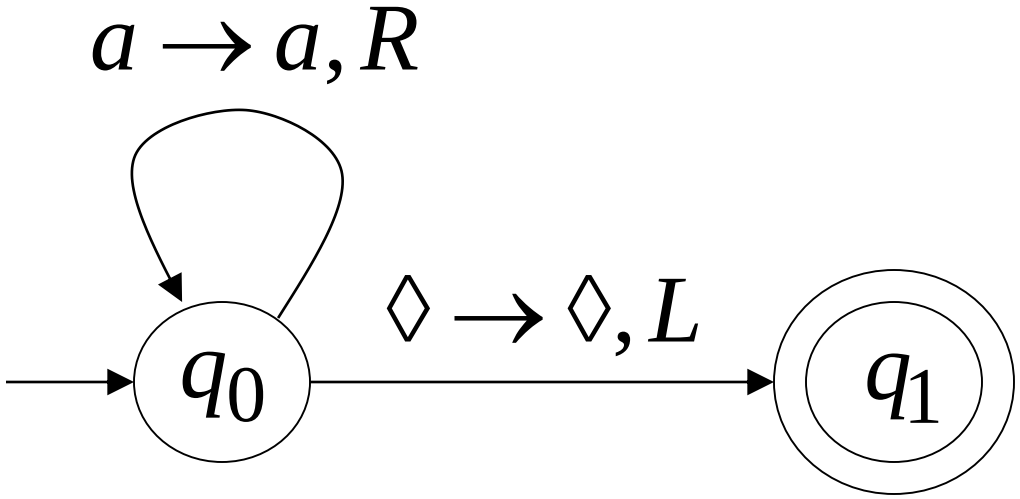
## Observation:

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

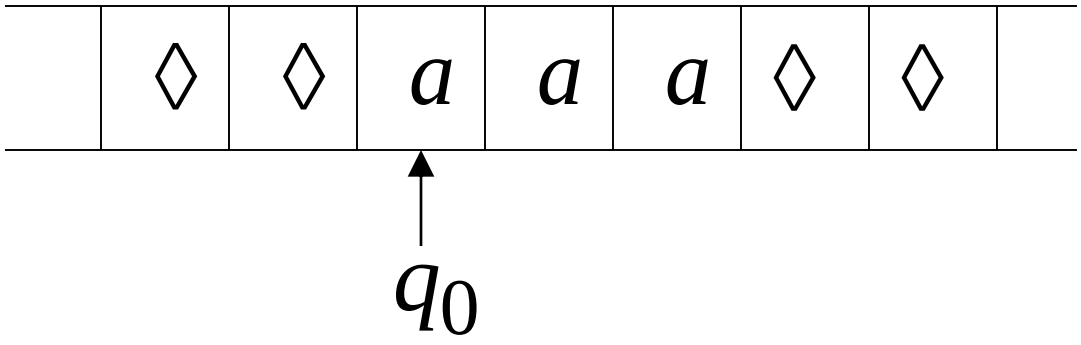
# Turing Machine Example

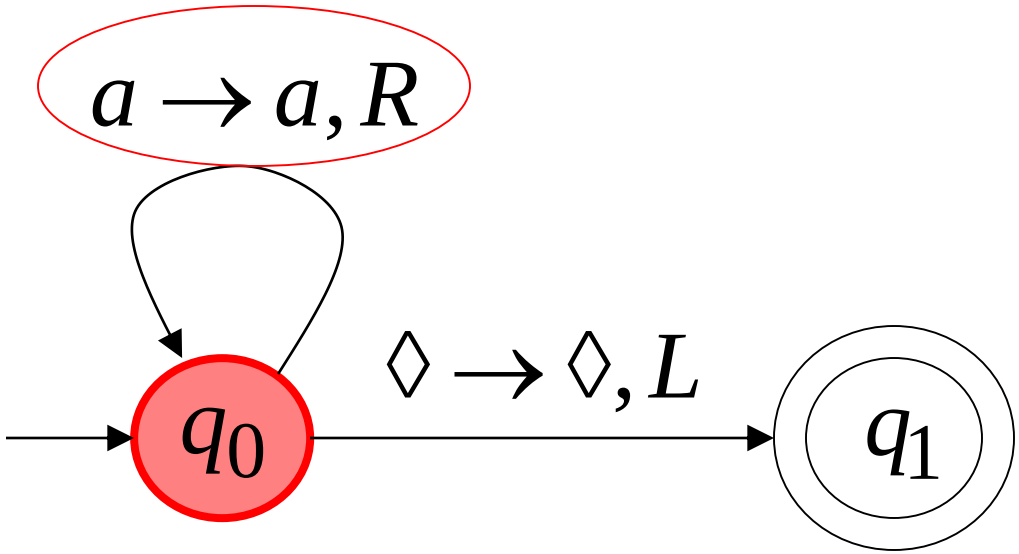
Input alphabet $\Sigma = \{a, b\}$

Accepts the language: $a^*$

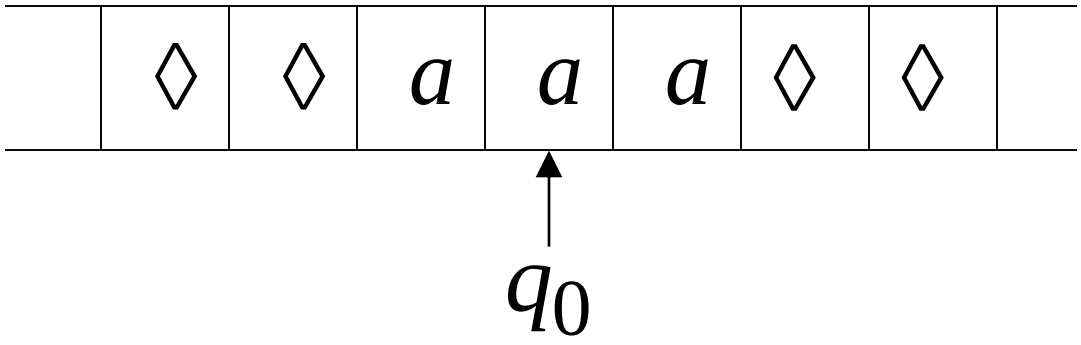


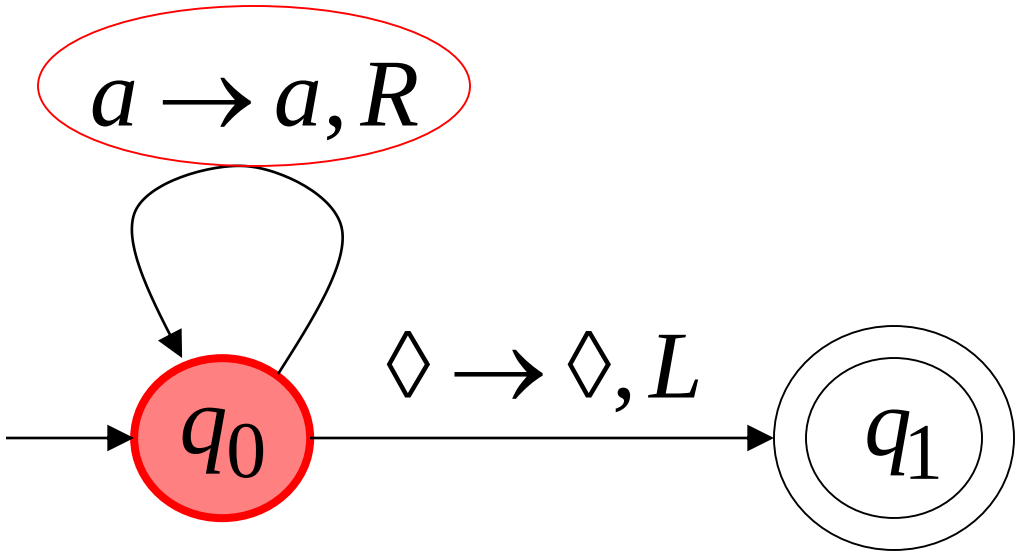
Time 0



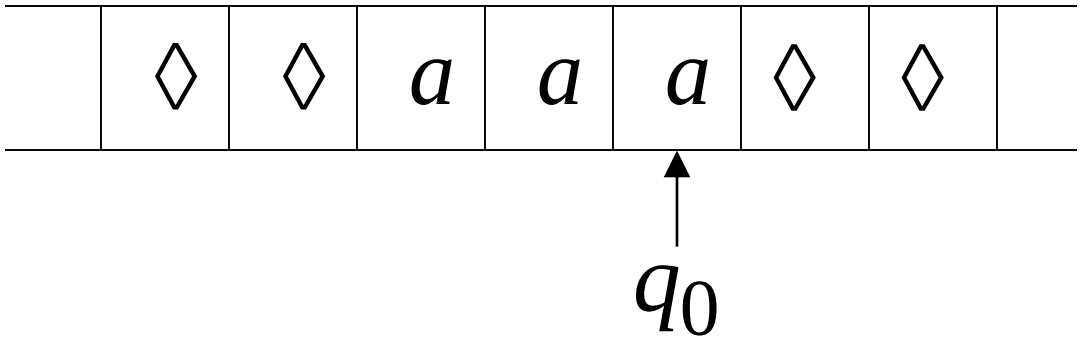



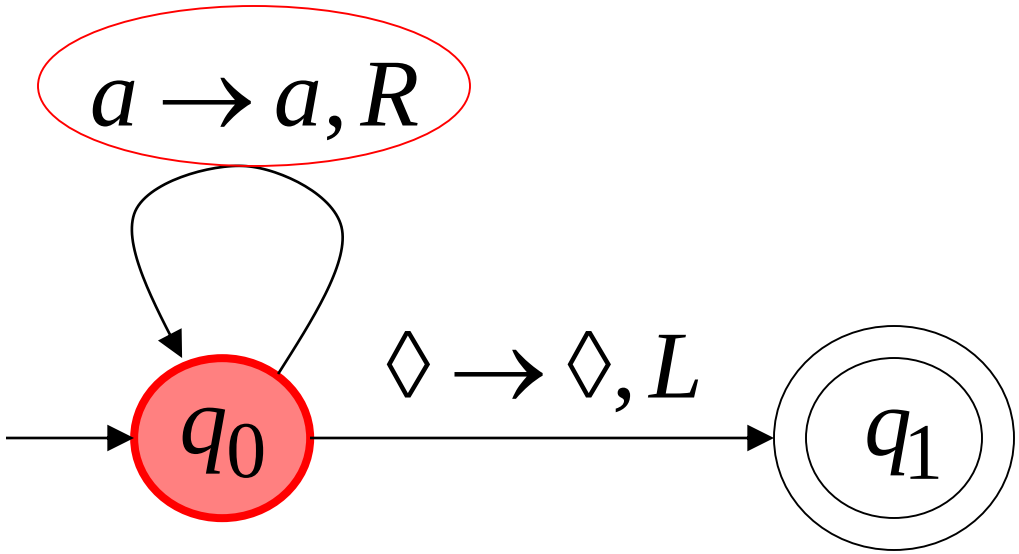
# Time 1



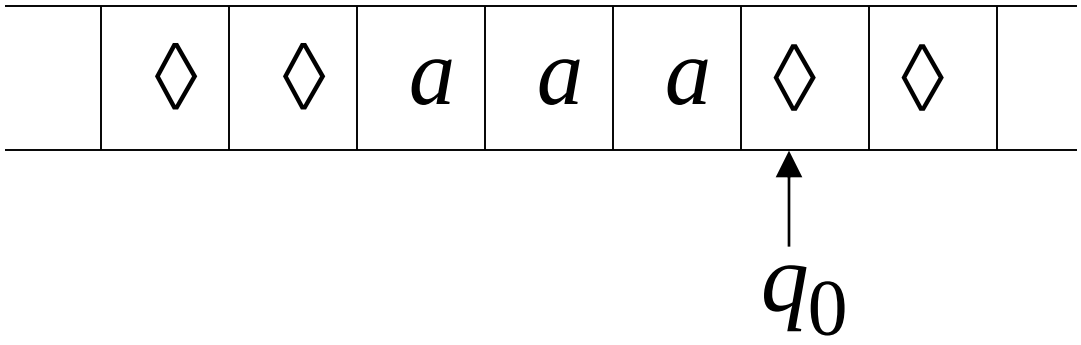



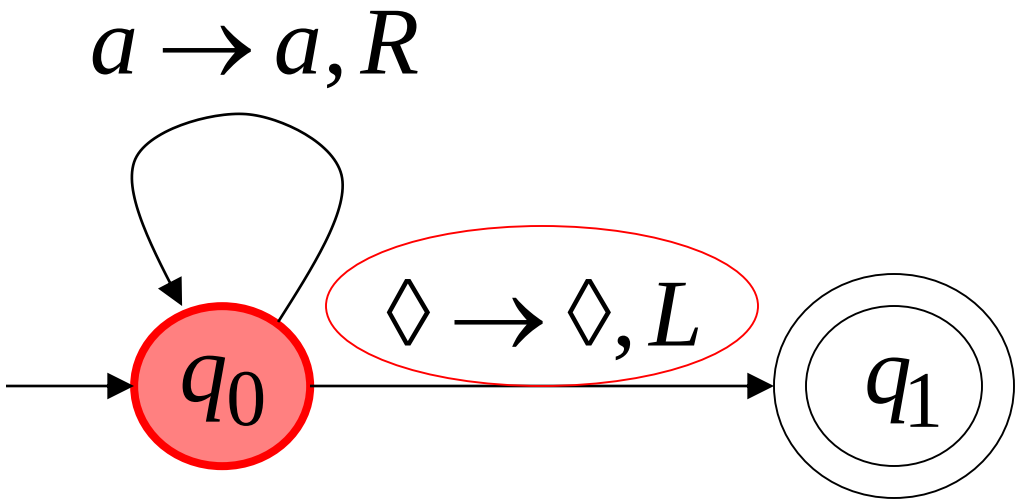
# Time 2



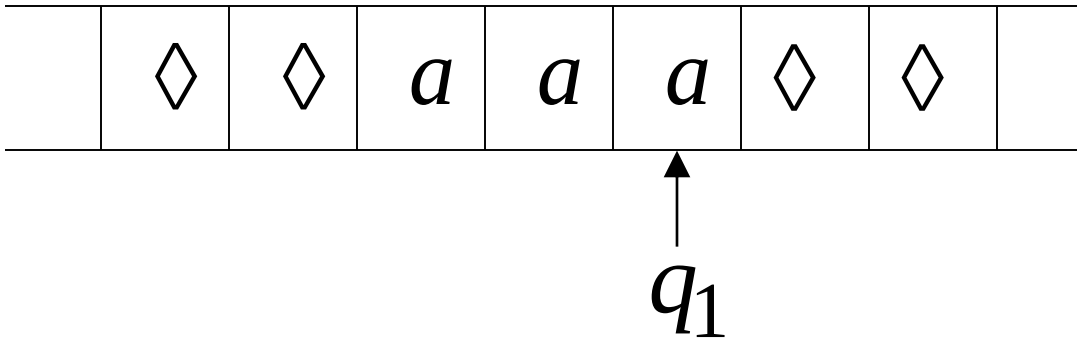



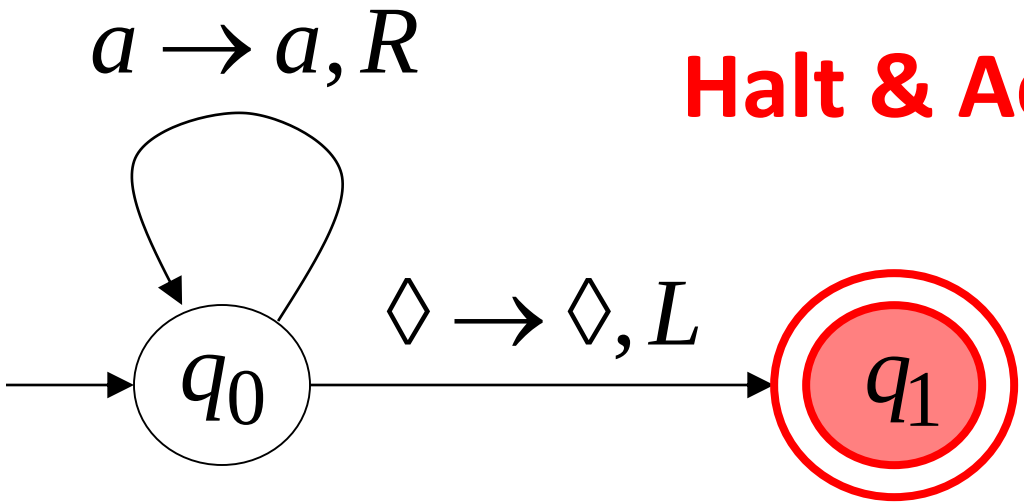
# Time 3



Time 4

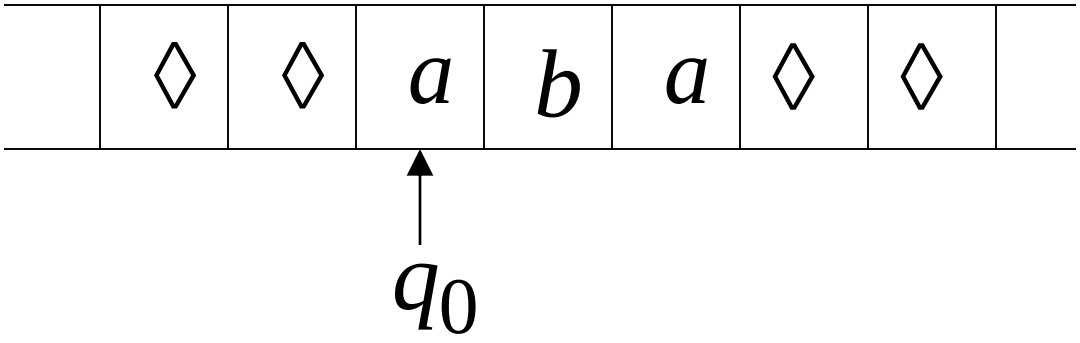


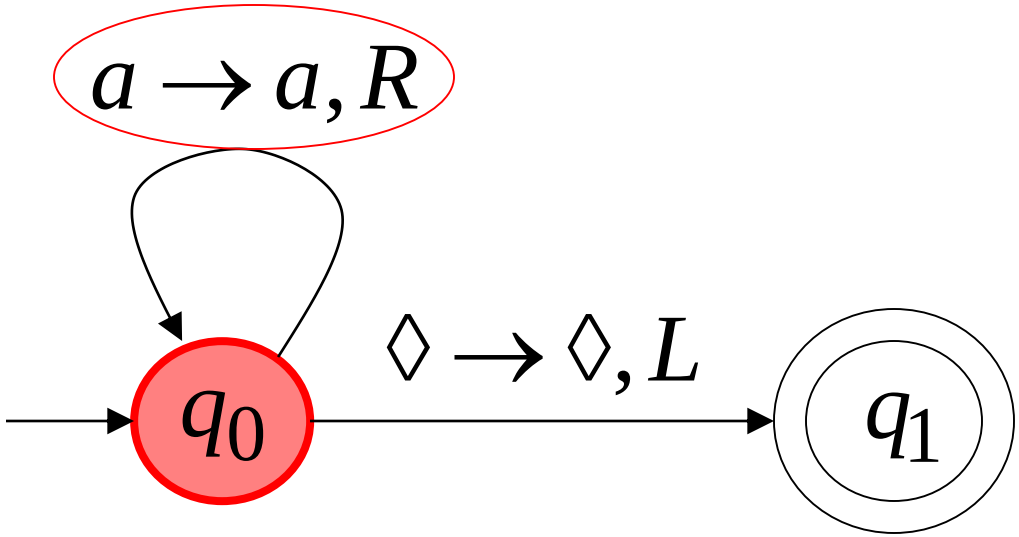




**Halt & Accept**

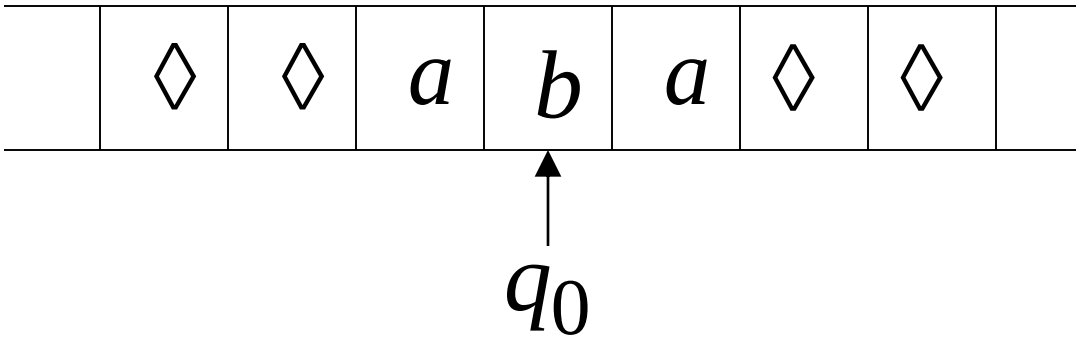
# Rejection Example

Time 0



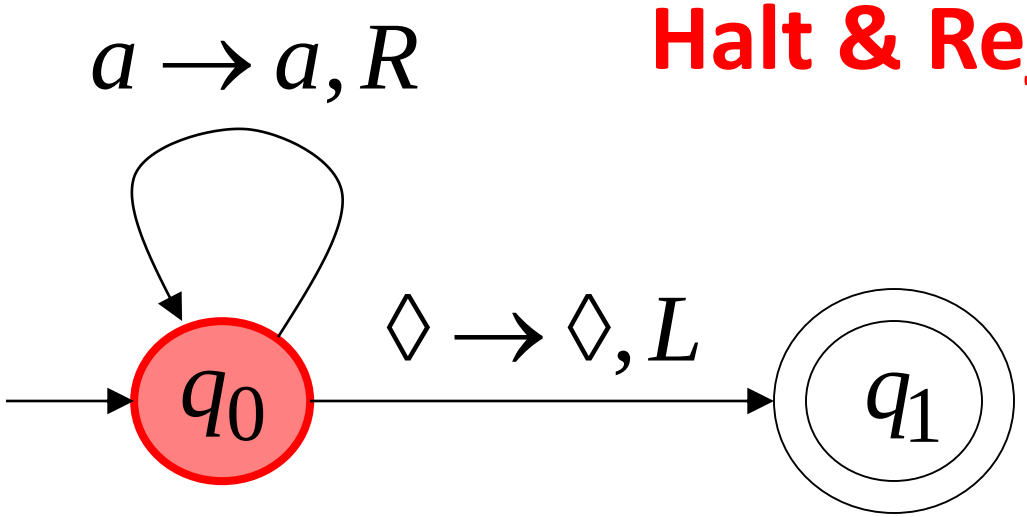



Time 1




No possible Transition

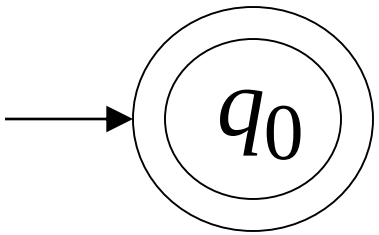
**Halt & Reject**



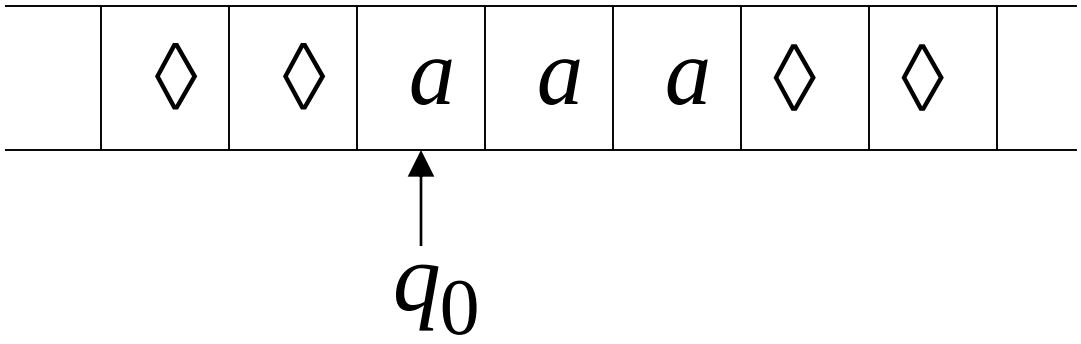
A simpler machine for same language

but for input alphabet $\Sigma = \{a\}$

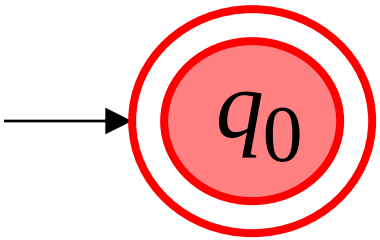
Accepts the language: $a^*$



Time 0



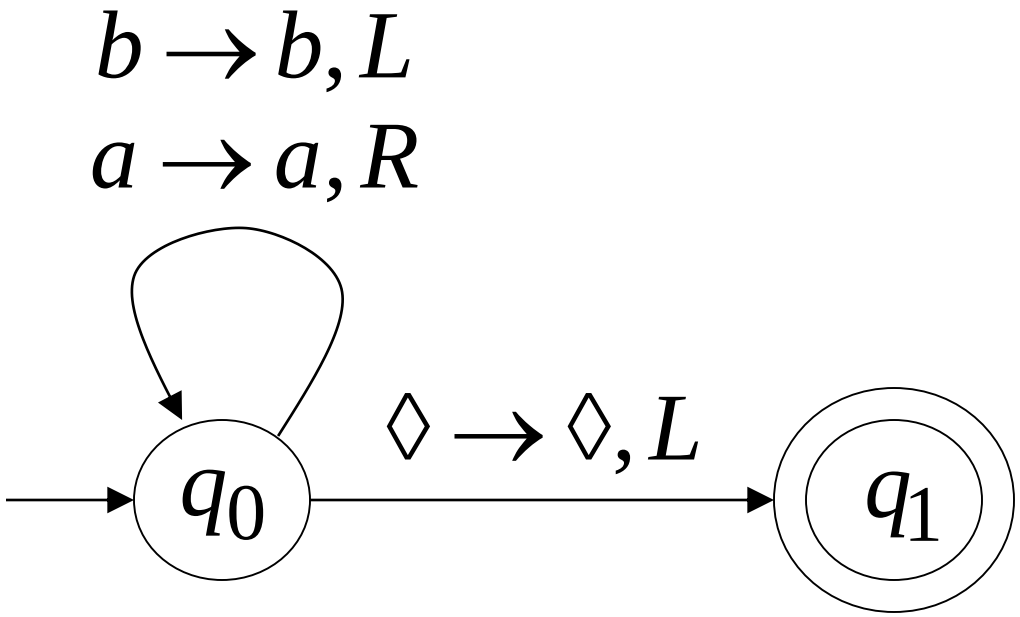

Halt & Accept



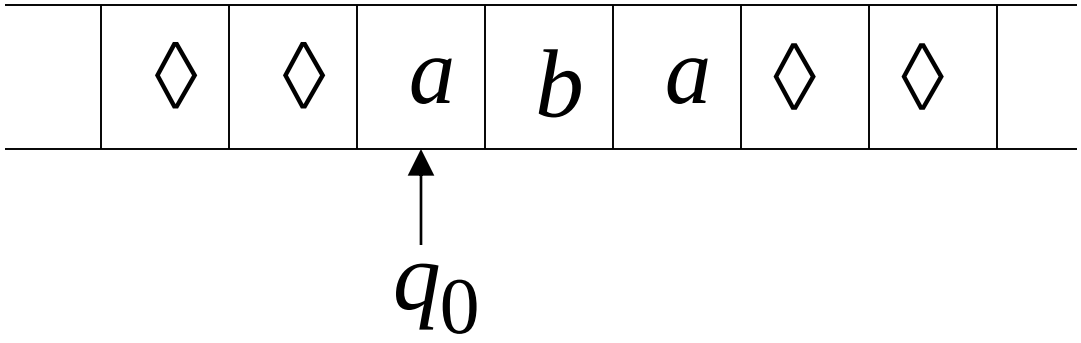

Not necessary to scan input

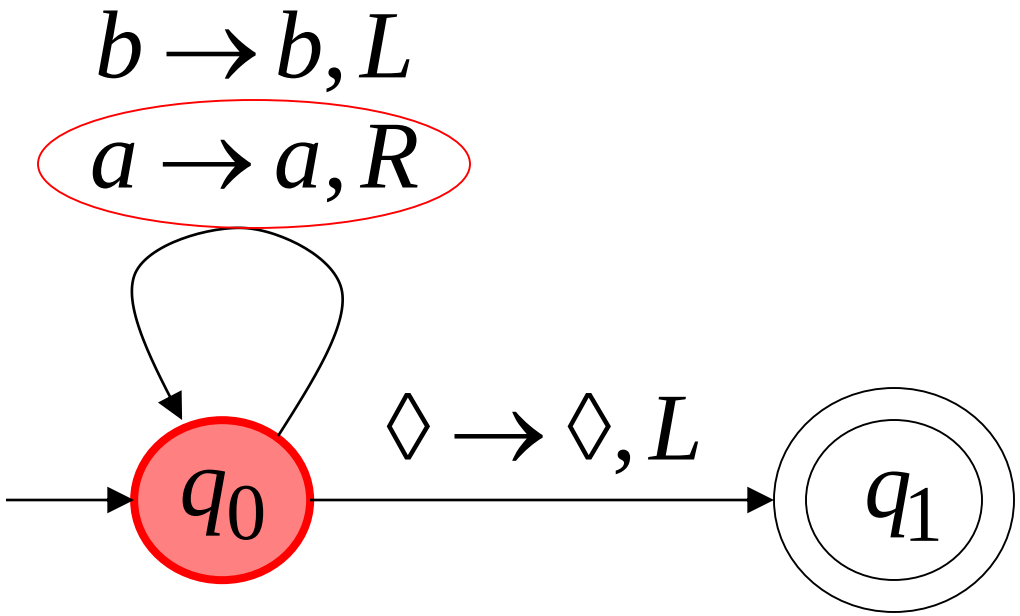
# Infinite Loop Example

A Turing machine for language $a*+b(a+b)*$

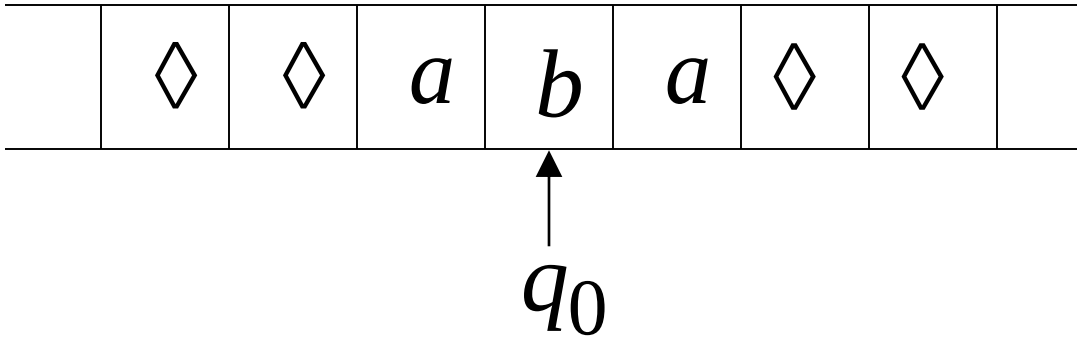


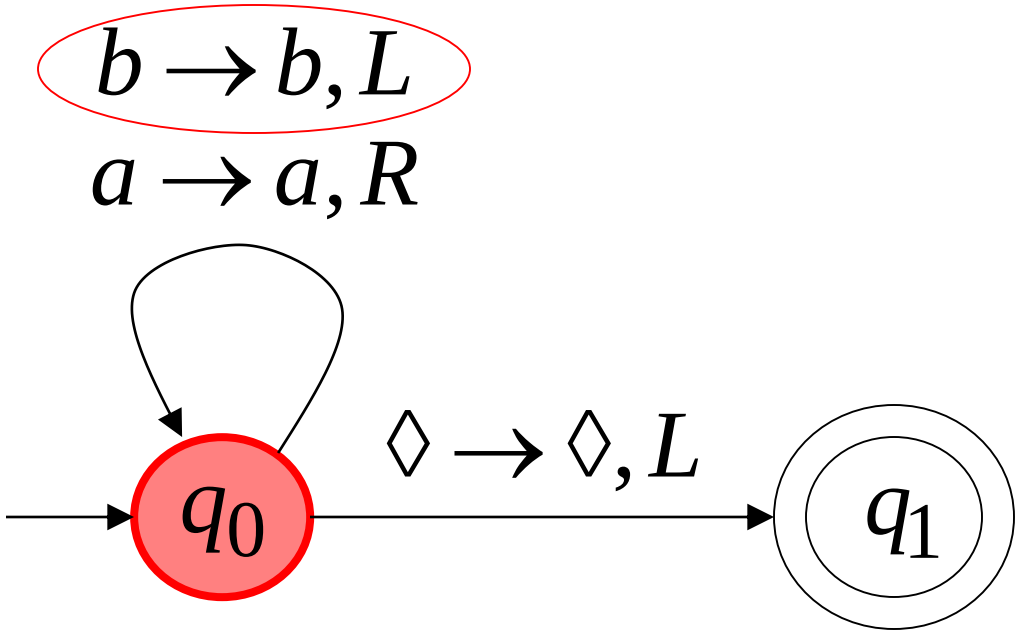
Time 0



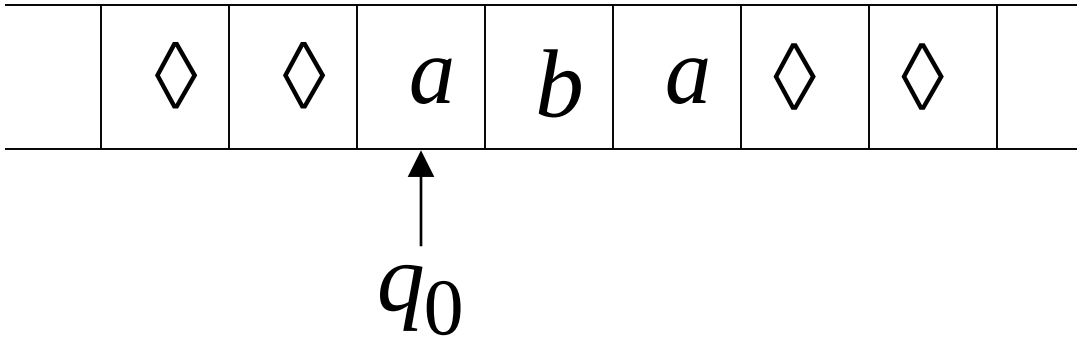



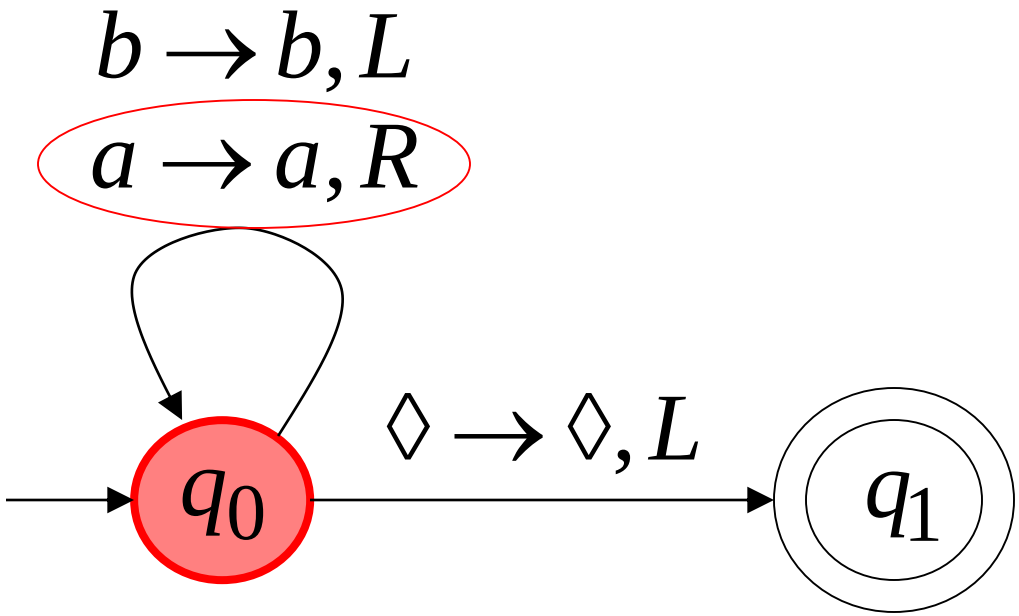
# Time 1



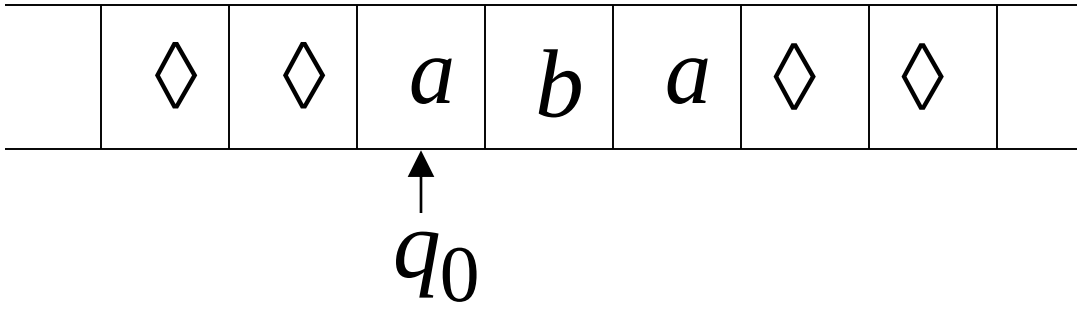



# Time 2

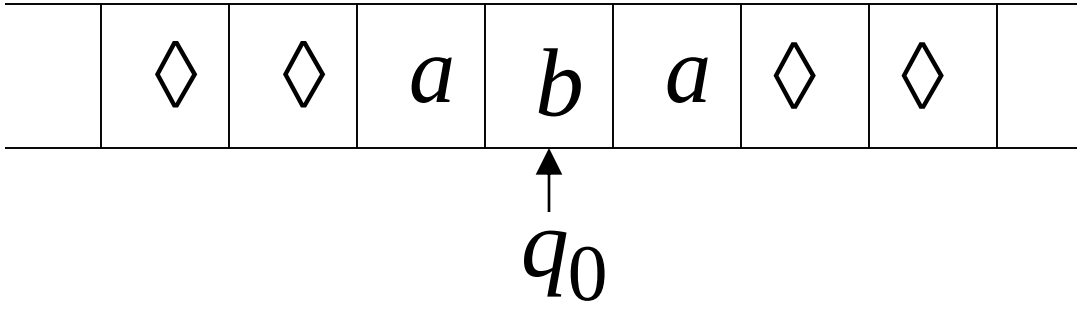


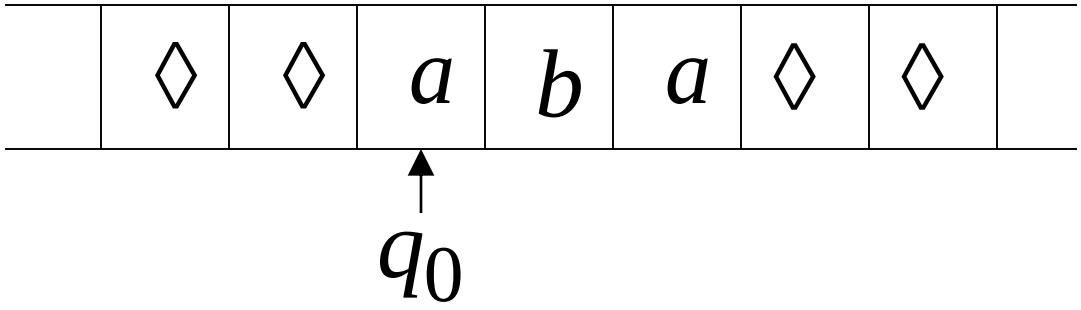
Time 2



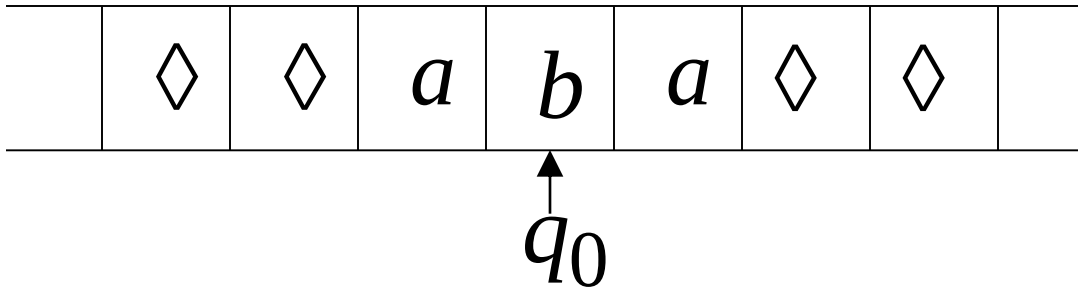

Time 3




Time 4




Time 5



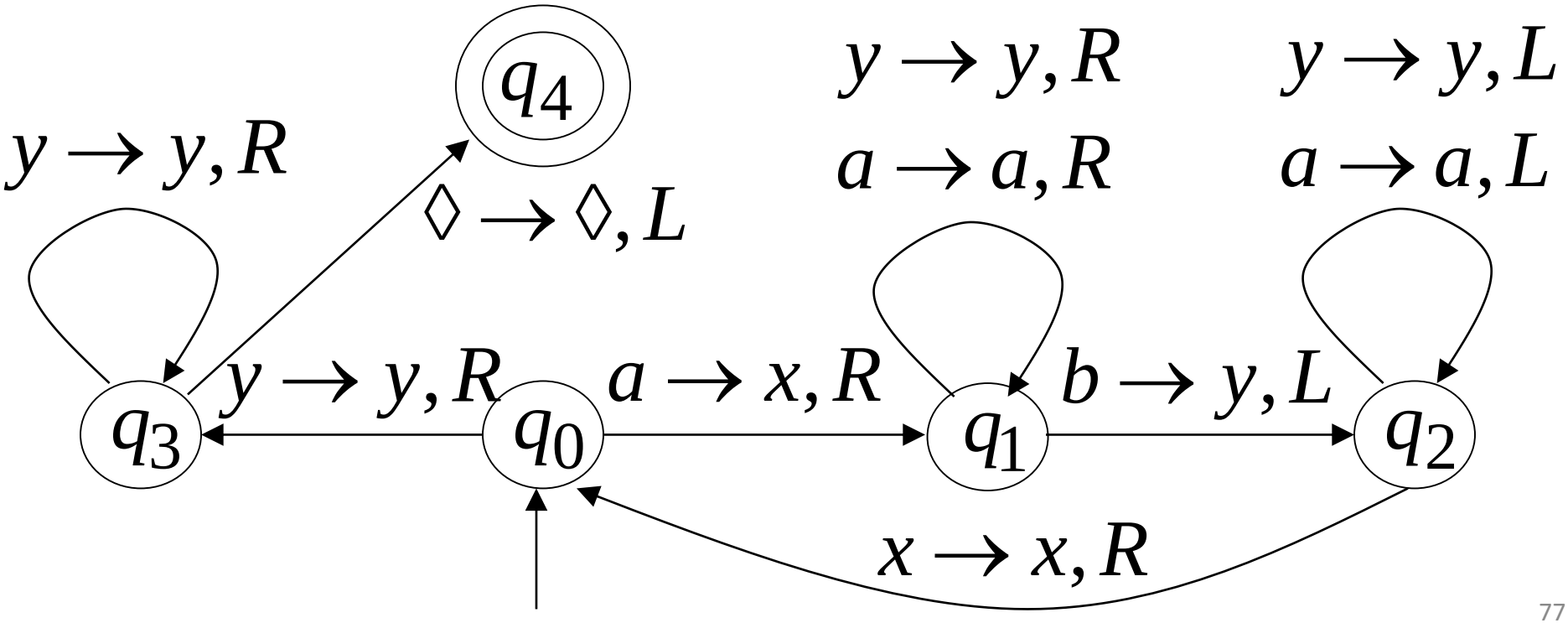

Infinite loop

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 $\{a^n b^n\}$, $n \geq 1$

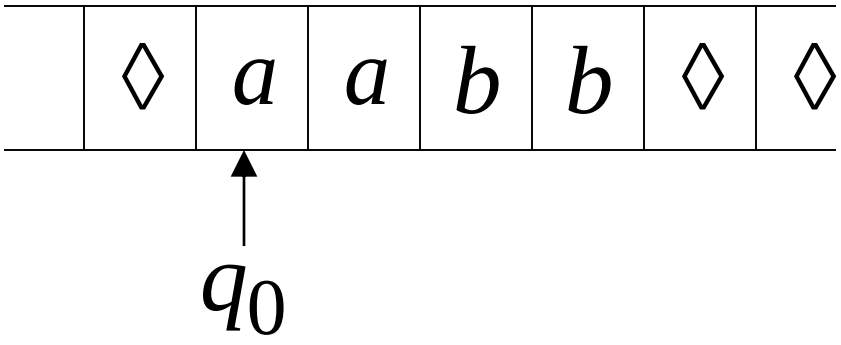


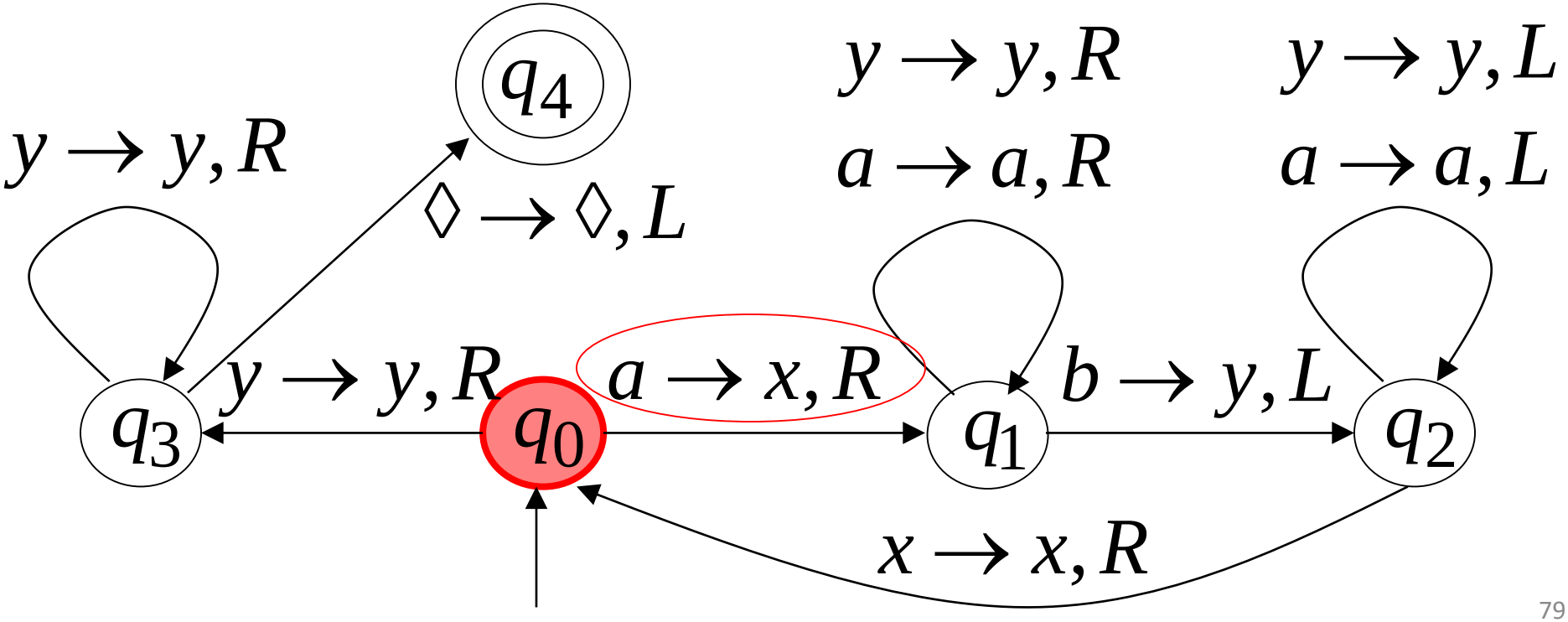
## Basic Idea:

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

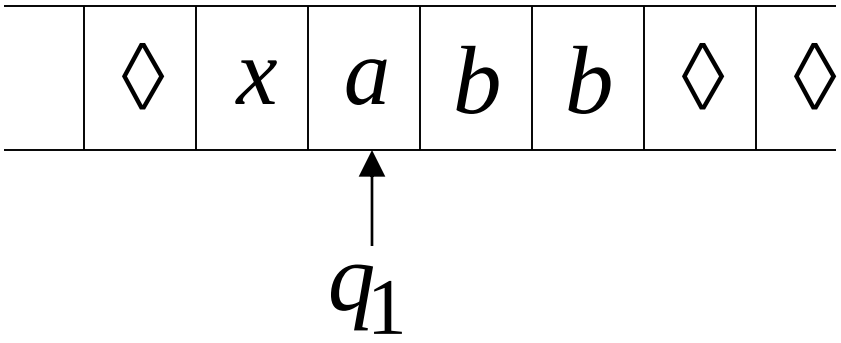
If there is a remaining a or b reject

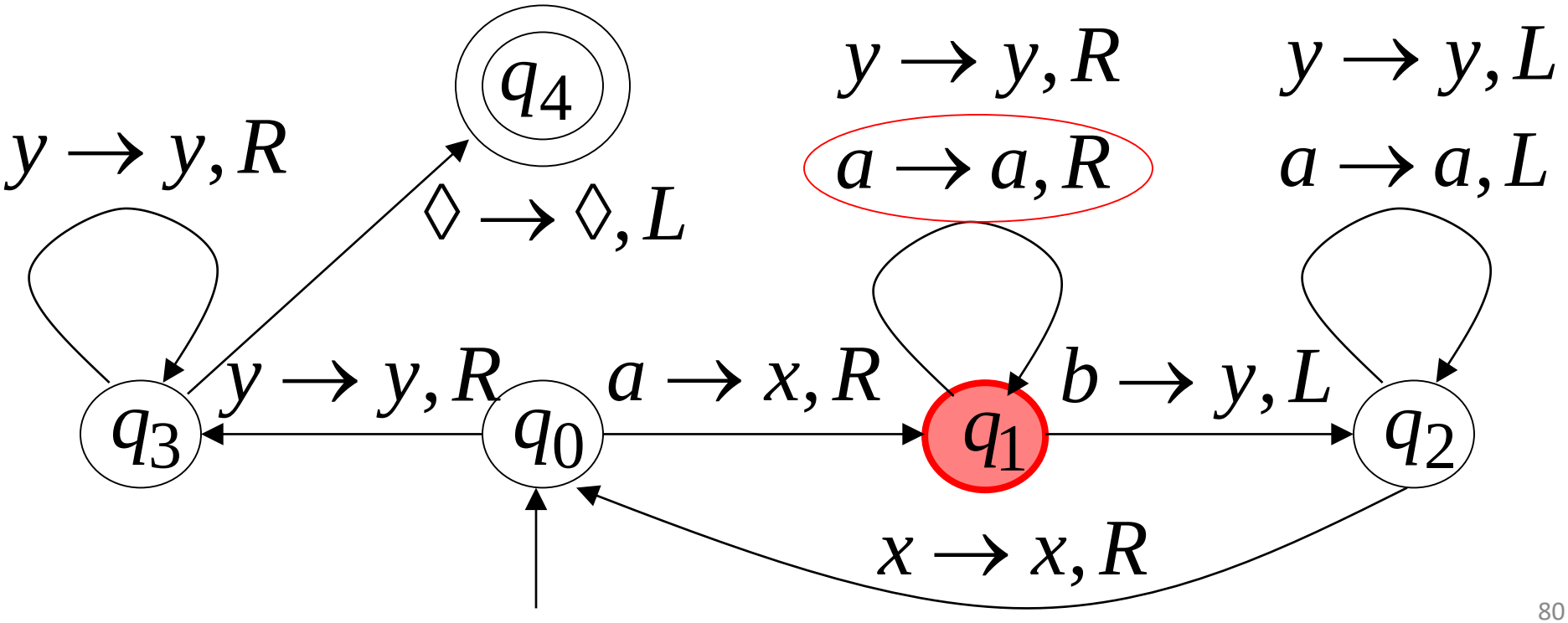
Time 0


◊ a a b b ◊ ◊
$q_0$

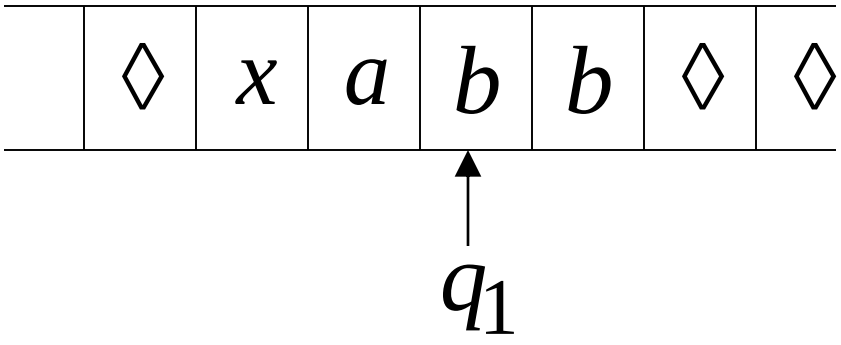

$q_4$
$y \to y, R$
$a \to a, R$
$y \to y, L$
$a \to a, L$
$y \to y, R$
$\Diamond \to \Diamond, L$
$y \to y, R$
$a \to x, R$
$b \to y, L$
$q_3$
$q_0$
$q_1$
$q_2$
$x \to x, R$

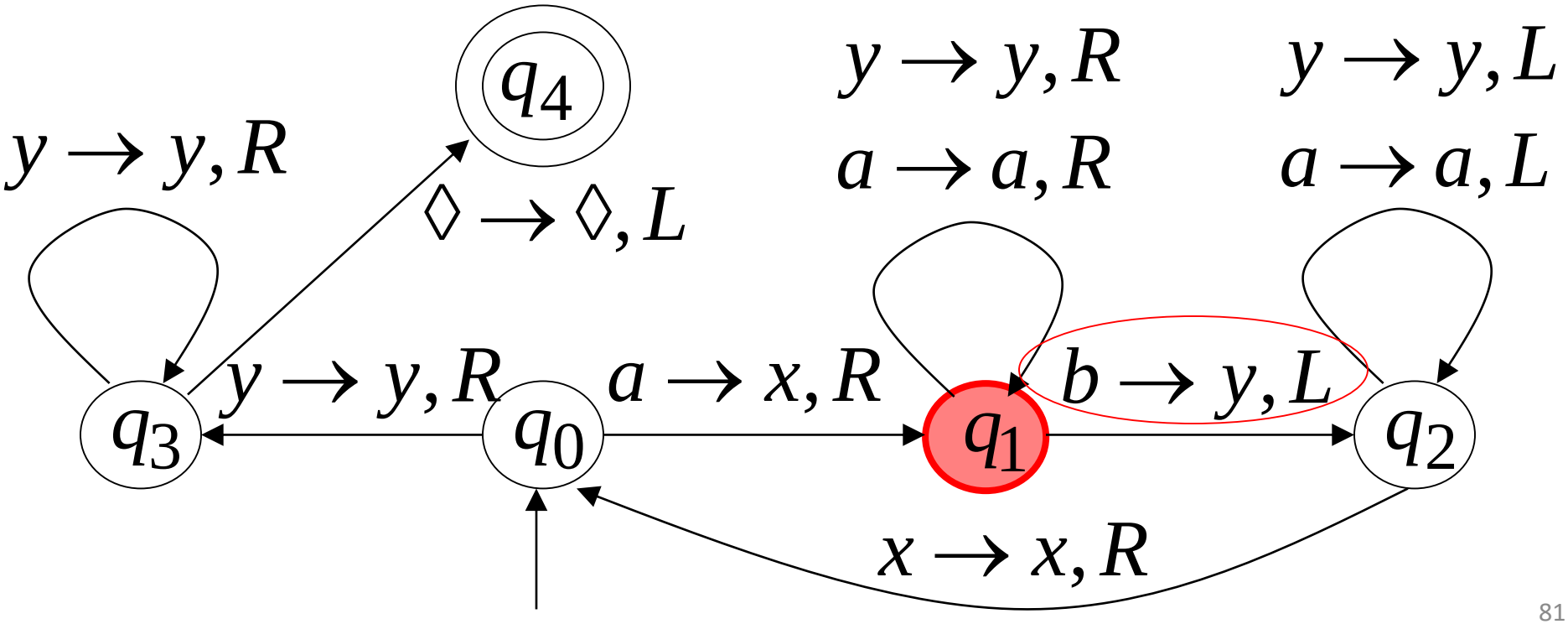
Time 1


◊ x a b b ◊ ◊
$q_1$

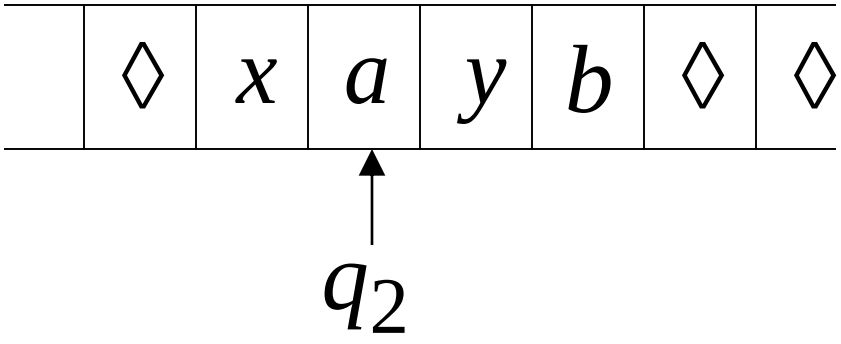

$q_4$
$y \to y, R$
$y \to y, R$
$y \to y, L$
$a \to a, R$
$a \to a, L$
$\Diamond \to \Diamond, L$
$y \to y, R$
$a \to x, R$
$b \to y, L$
$q_3$
$q_0$
$q_1$
$q_2$
$x \to x, R$

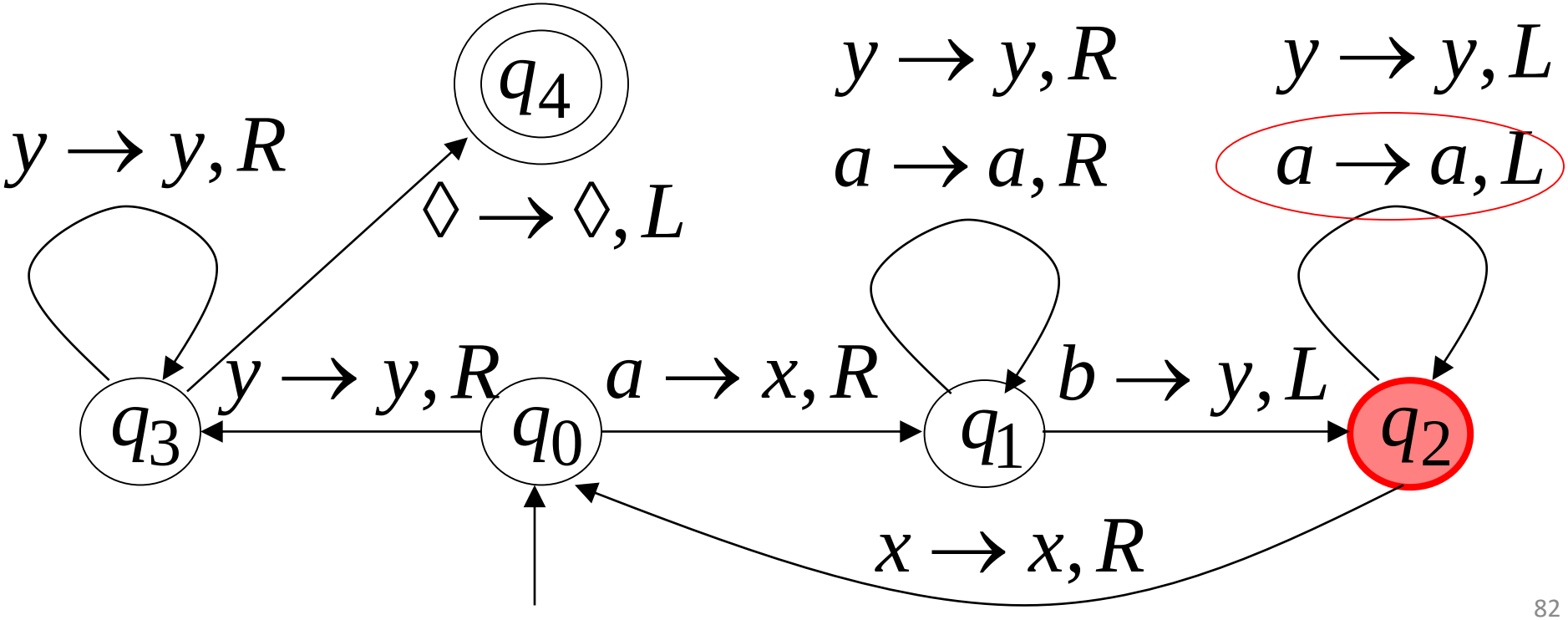
# Time 2


◊ x a b b ◊ ◊
$q_1$

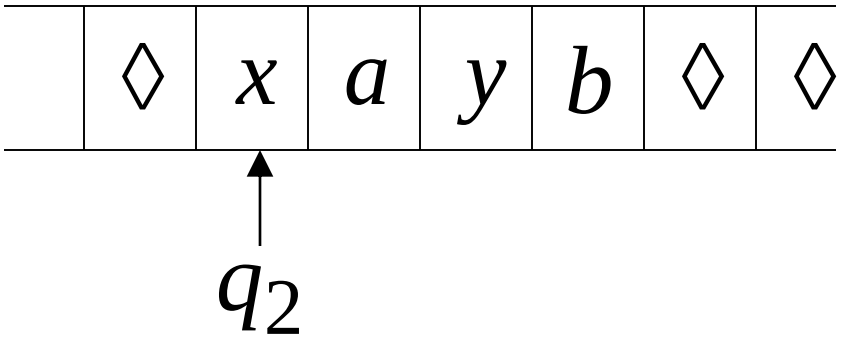

$q_4$
$y \to y, R$
$\lozenge \to \lozenge, L$
$y \to y, R$
$a \to a, R$
$y \to y, L$
$a \to a, L$
$q_3$
$y \to y, R$
$q_0$
$a \to x, R$
$q_1$
$b \to y, L$
$q_2$
$x \to x, R$

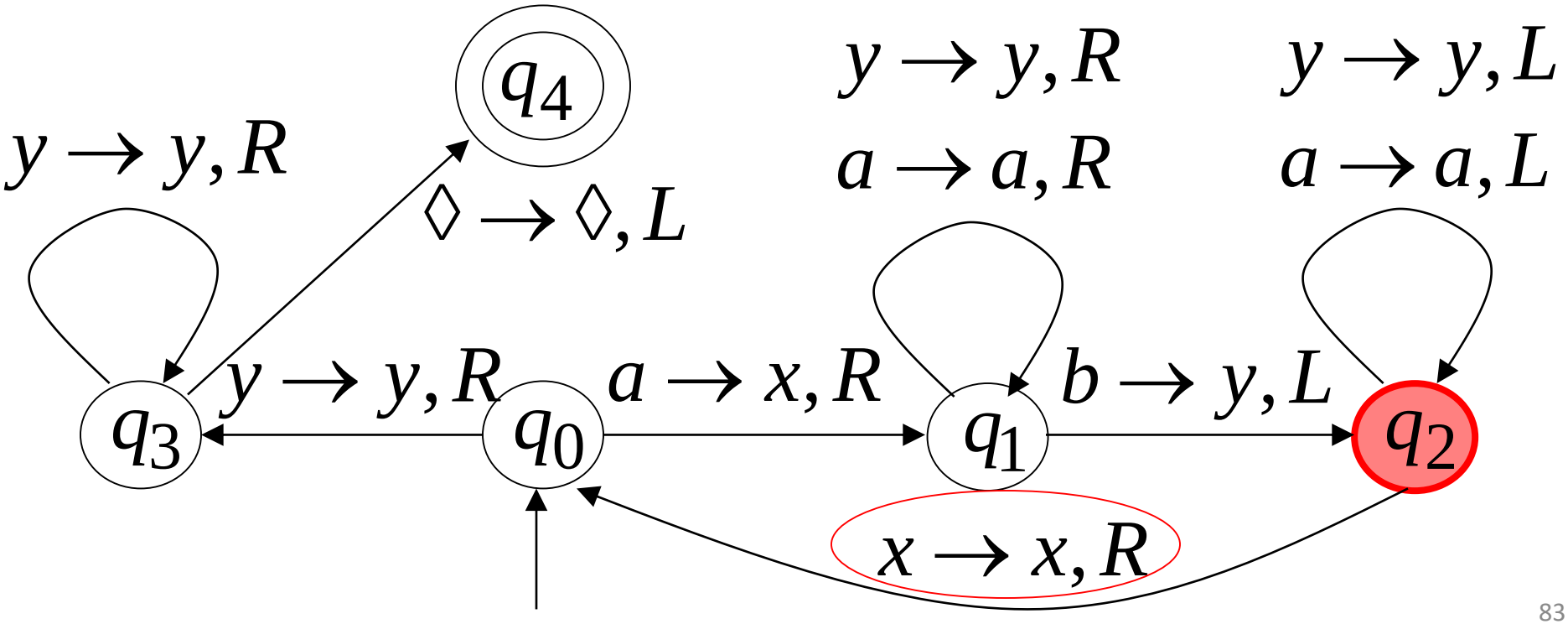
Time 3


◊ x a y b ◊ ◊
$q_2$

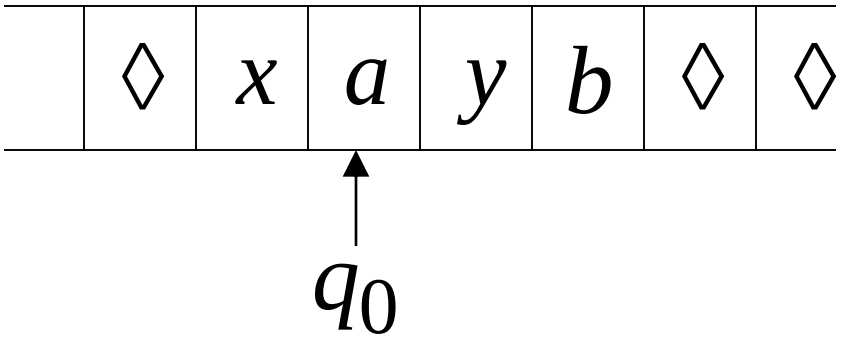

$q_4$
$y \to y, R$
$\lozenge \to \lozenge, L$
$y \to y, R$
$a \to a, R$
$y \to y, L$
$a \to a, L$
$q_3$
$y \to y, R$
$q_0$
$a \to x, R$
$q_1$
$b \to y, L$
$q_2$
$x \to x, R$

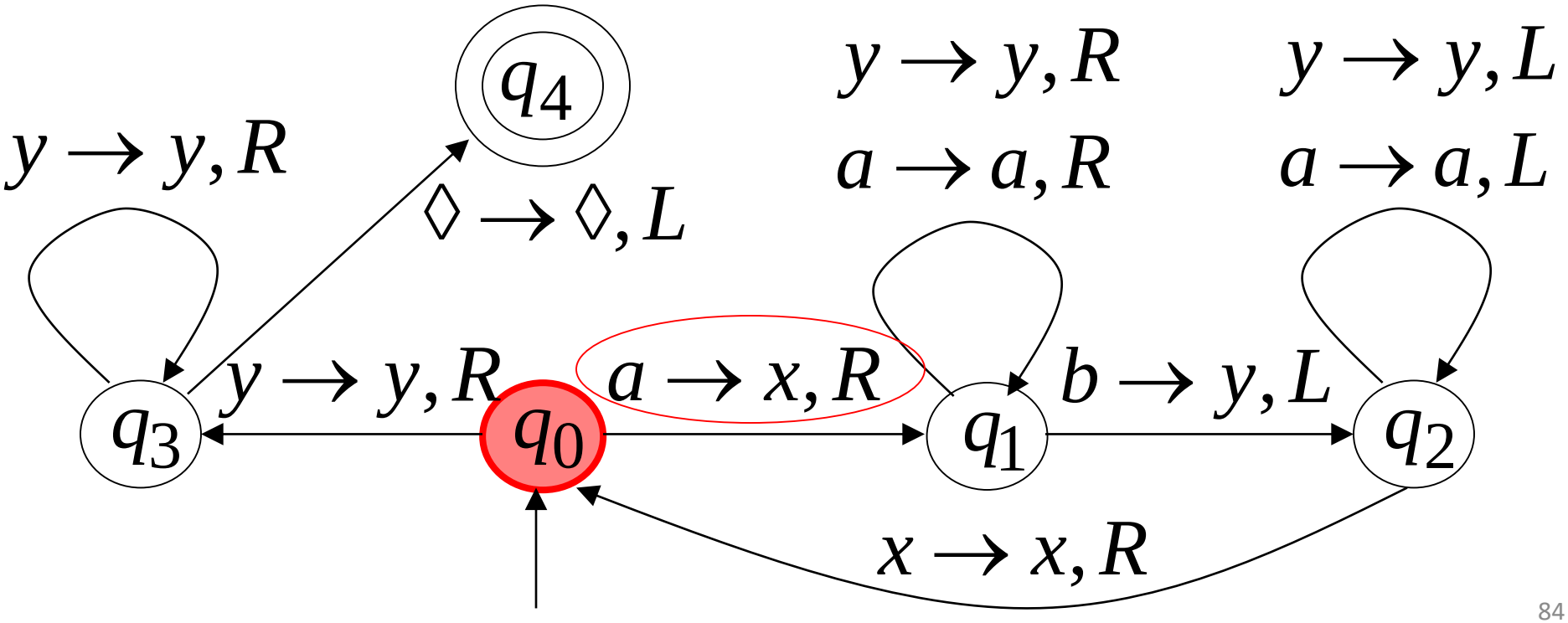
Time 4


◊ x a y b ◊ ◊
$q_2$

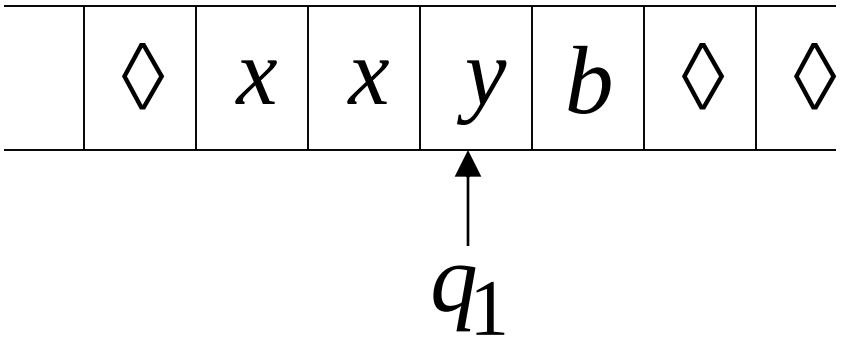

$q_4$
$y \to y, R$
$a \to a, R$
$y \to y, L$
$a \to a, L$
$y \to y, R$
$\Diamond \to \Diamond, L$
$y \to y, R$
$a \to x, R$
$b \to y, L$
$q_3$
$q_0$
$q_1$
$q_2$
$x \to x, R$

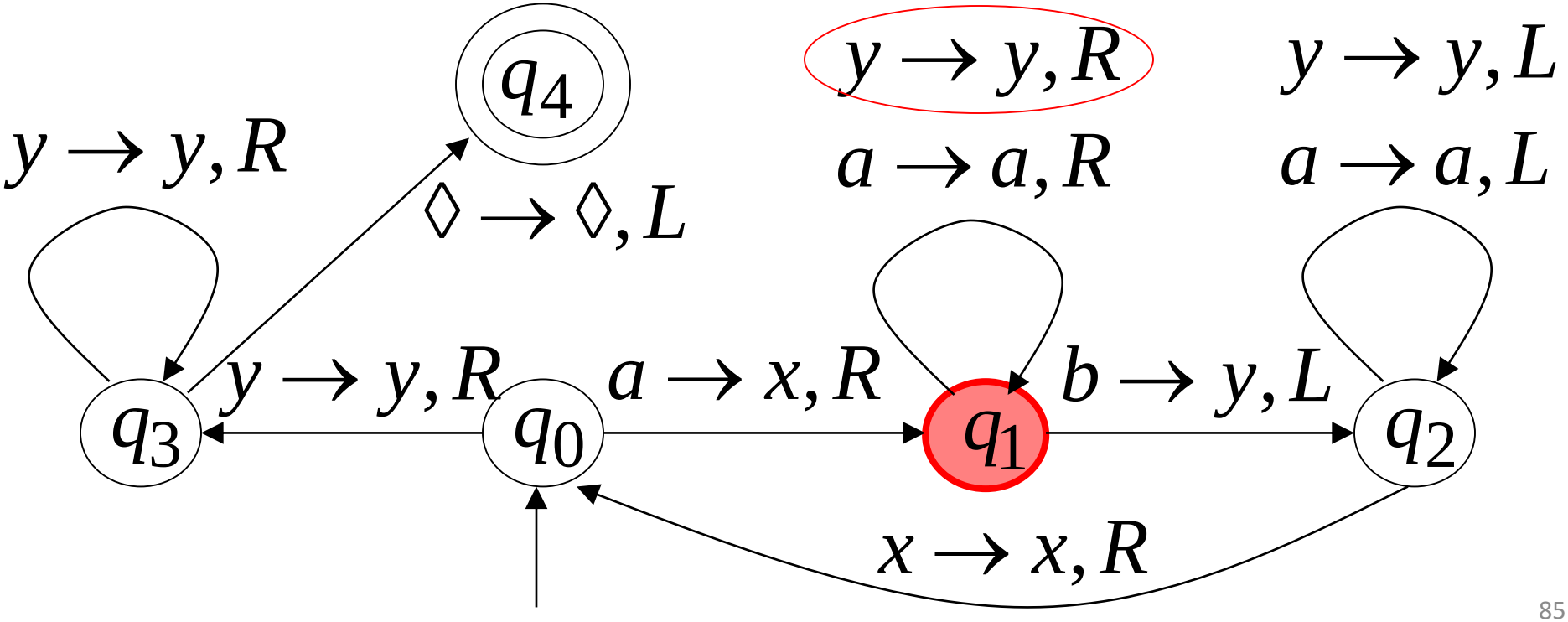
Time 5


◊ x a y b ◊ ◊
$q_0$

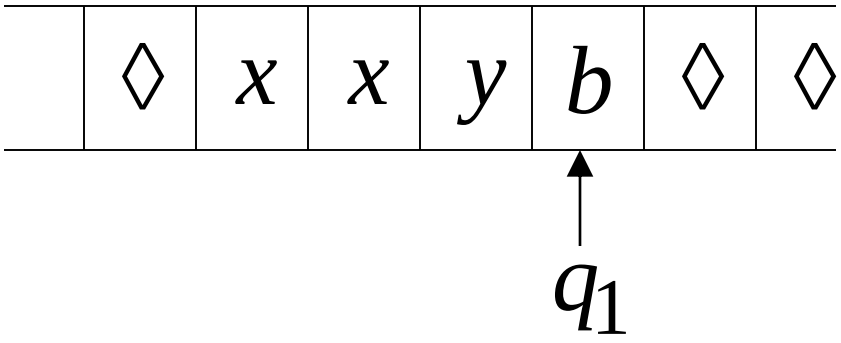

$q_4$
$y \to y, R$
$a \to a, R$
$y \to y, L$
$a \to a, L$
$y \to y, R$
$\Diamond \to \Diamond, L$
$y \to y, R$
$a \to x, R$
$b \to y, L$
$q_3$
$q_0$
$q_1$
$q_2$
$x \to x, R$

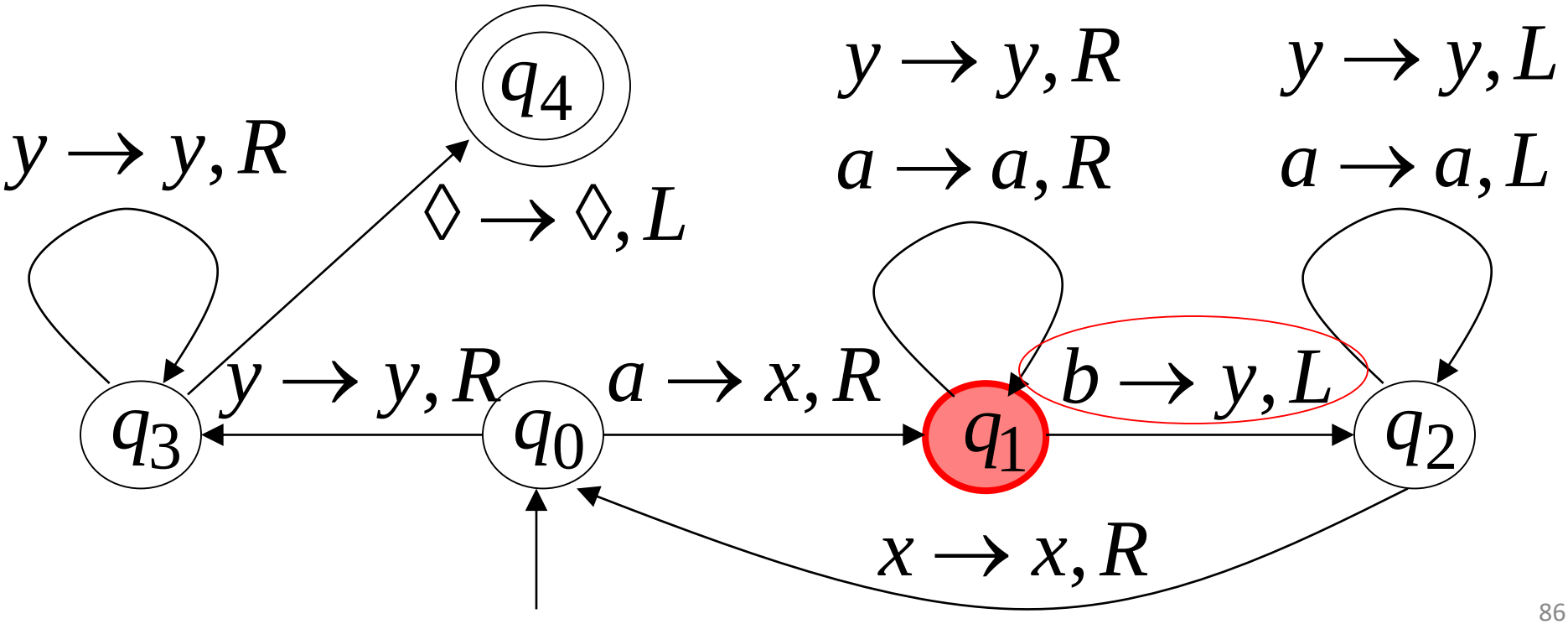
## Time 6


◊ x x y b ◊ ◊
$q_1$

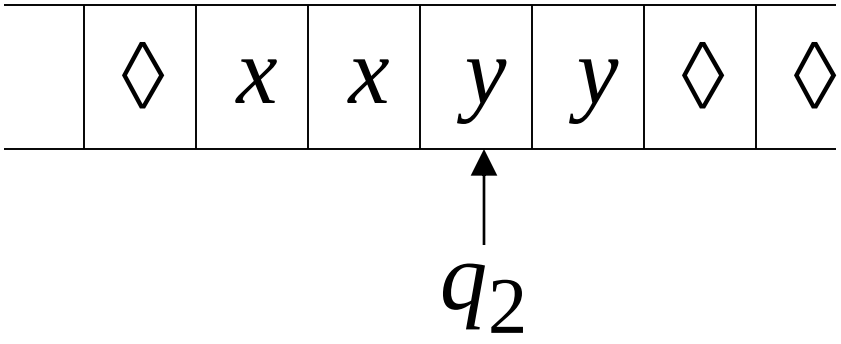

$q_4$
$y \to y, R$
$y \to y, R$
$a \to a, R$
$y \to y, L$
$a \to a, L$
$\Diamond \to \Diamond, L$
$y \to y, R$
$a \to x, R$
$b \to y, L$
$q_3$
$q_0$
$q_1$
$q_2$
$x \to x, R$

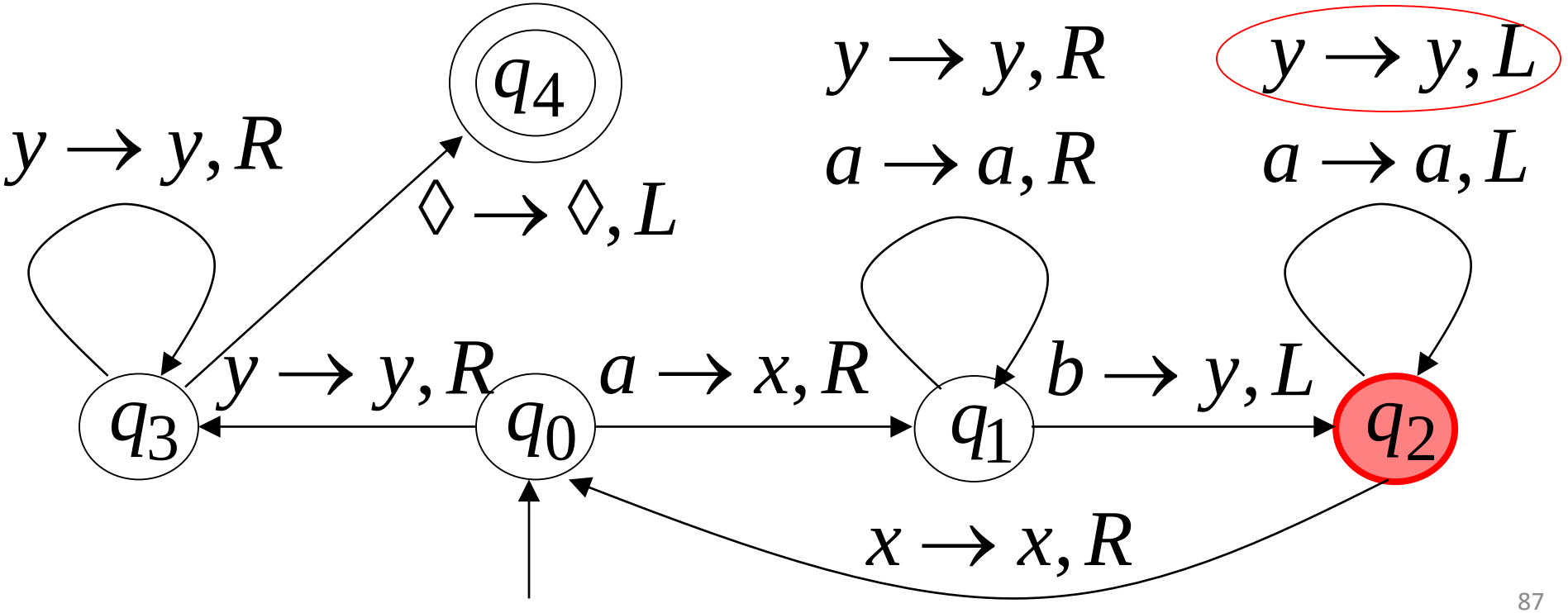
Time 7


◊ x x y b ◊ ◊
$q_1$

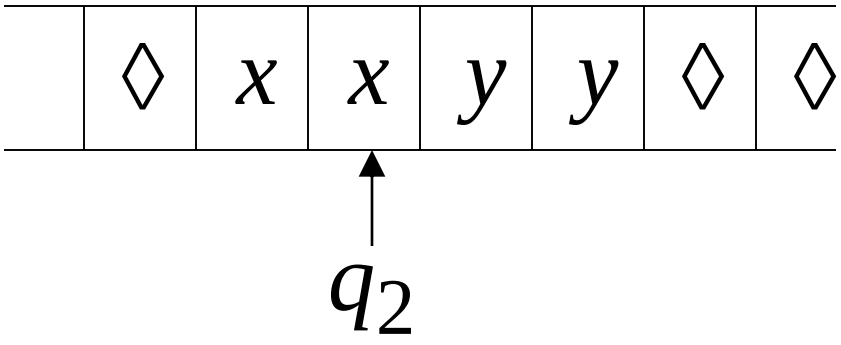

$q_4$
$y \to y, R$
$\lozenge \to \lozenge, L$
$y \to y, R$
$a \to a, R$
$y \to y, L$
$a \to a, L$
$q_3$
$y \to y, R$
$q_0$
$a \to x, R$
$q_1$
$b \to y, L$
$q_2$
$x \to x, R$

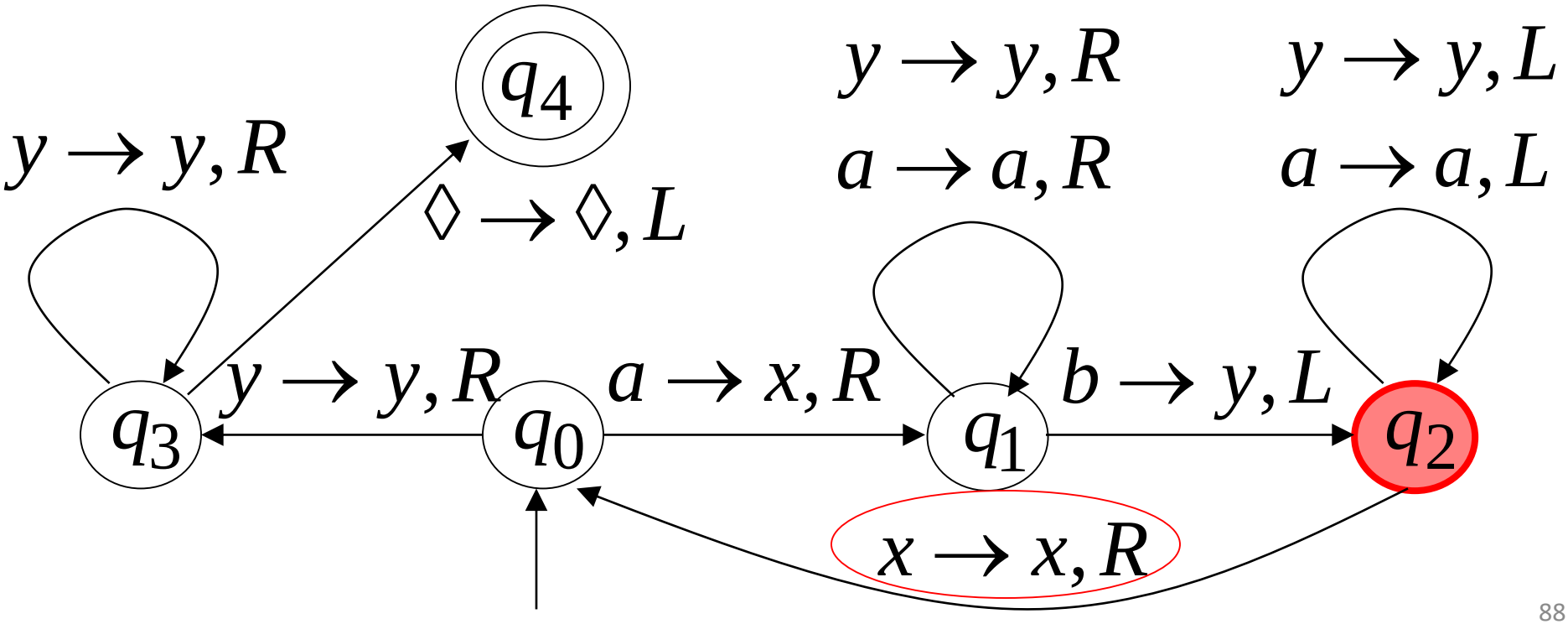
## Time 8


◊ x x y y ◊ ◊
$q_2$

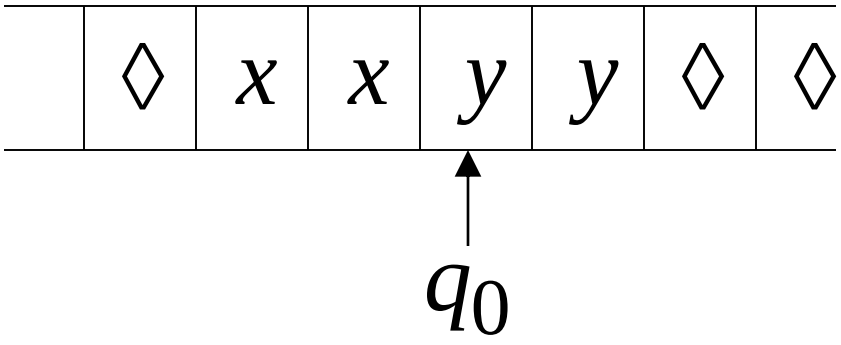

$q_4$
$y \to y, R$
$\Diamond \to \Diamond, L$
$y \to y, R$
$a \to a, R$
$y \to y, L$
$a \to a, L$
$q_3$
$y \to y, R$
$q_0$
$a \to x, R$
$q_1$
$b \to y, L$
$q_2$
$x \to x, R$

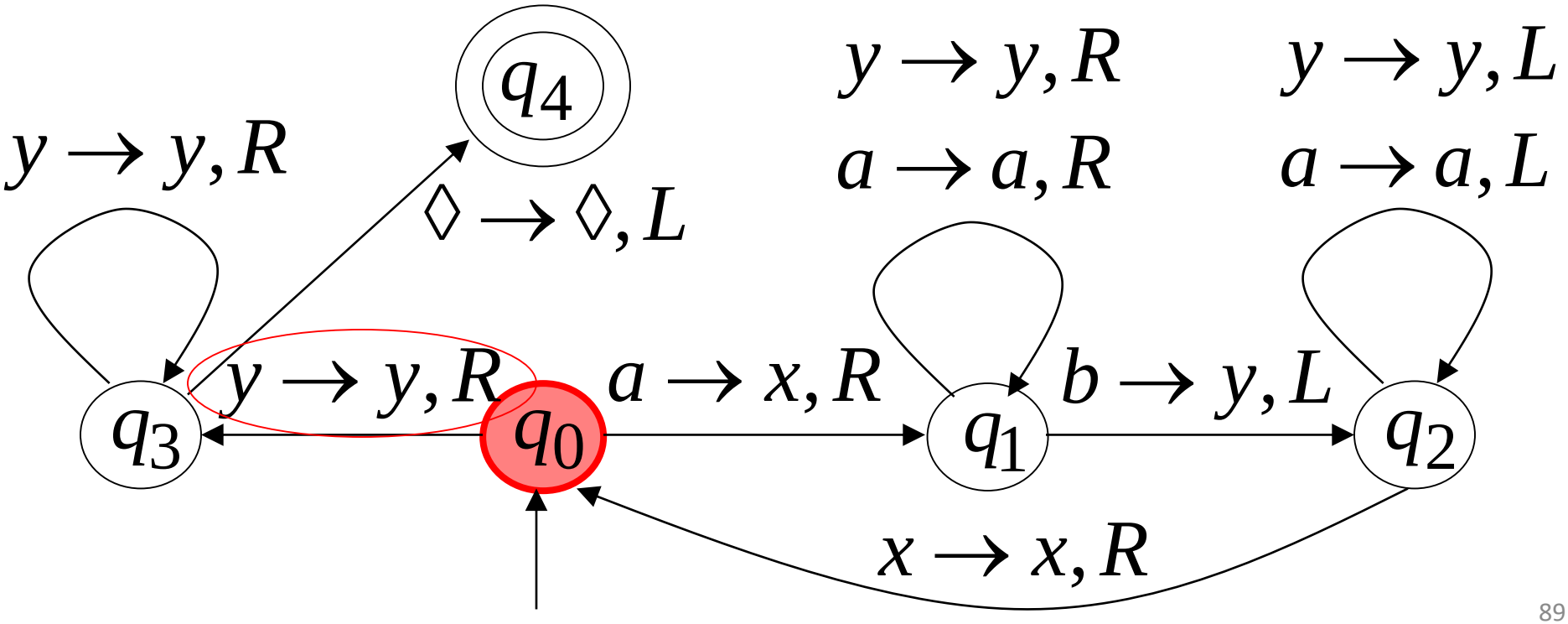
Time 9


◊ x x y y ◊ ◊
$q_2$

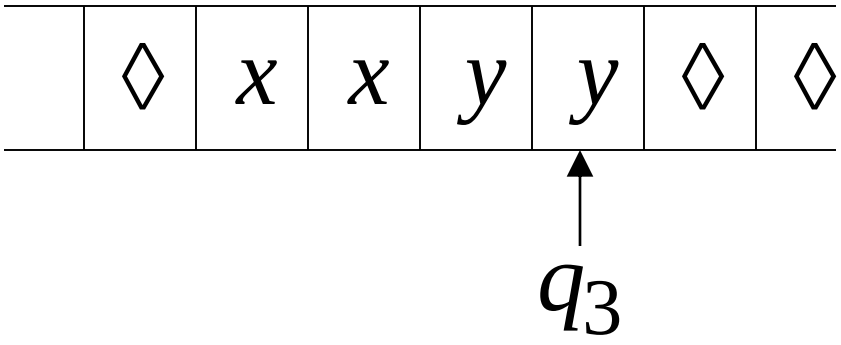

$q_4$
$y \to y, R$
$\lozenge \to \lozenge, L$
$y \to y, R$
$a \to a, R$
$y \to y, L$
$a \to a, L$
$q_3$
$y \to y, R$
$q_0$
$a \to x, R$
$q_1$
$b \to y, L$
$q_2$
$x \to x, R$

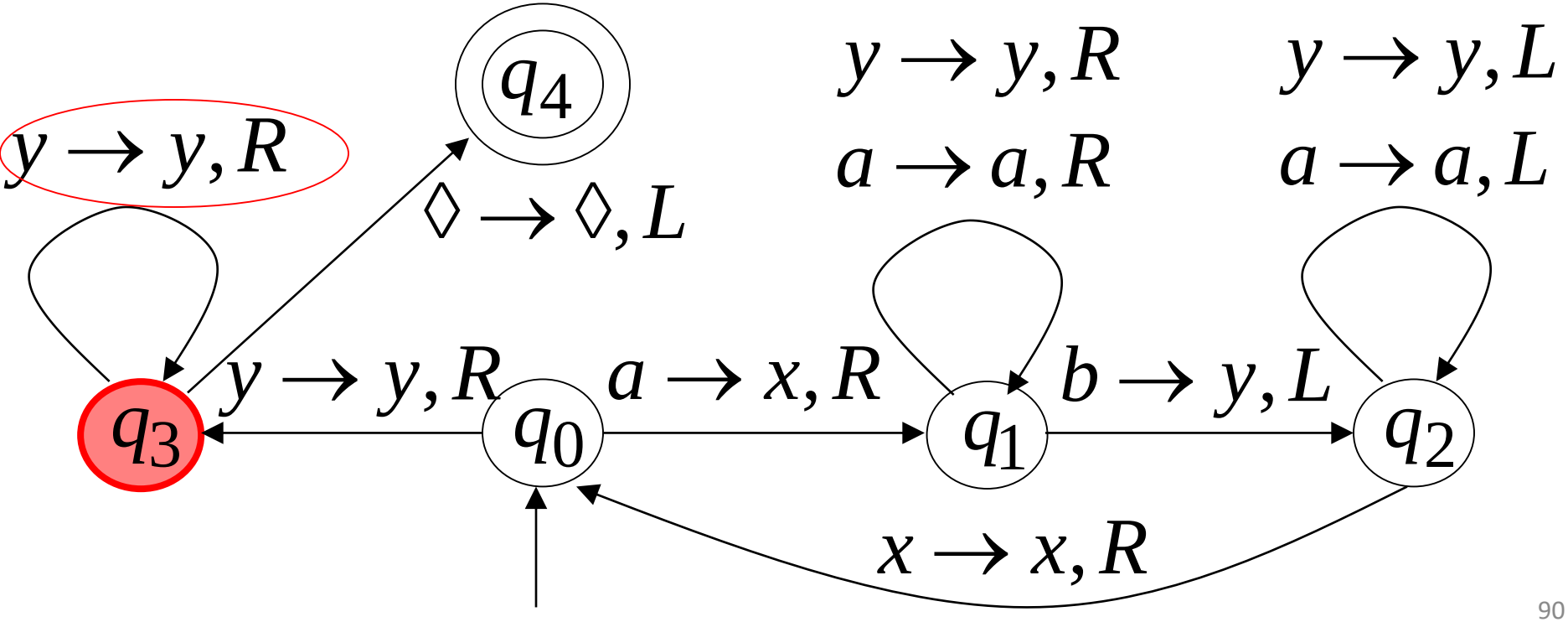
Time 10


◊ x x y y ◊ ◊
$q_0$

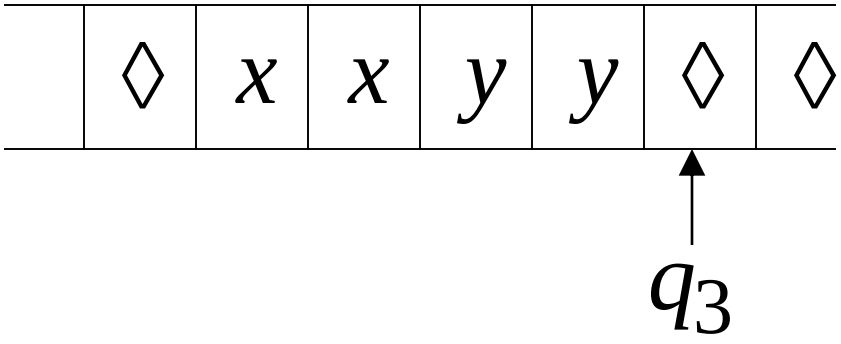

$q_4$
$y \to y, R$
$a \to a, R$
$y \to y, L$
$a \to a, L$
$y \to y, R$
$\Diamond \to \Diamond, L$
$y \to y, R$
$a \to x, R$
$b \to y, L$
$q_3$
$q_0$
$q_1$
$q_2$
$x \to x, R$

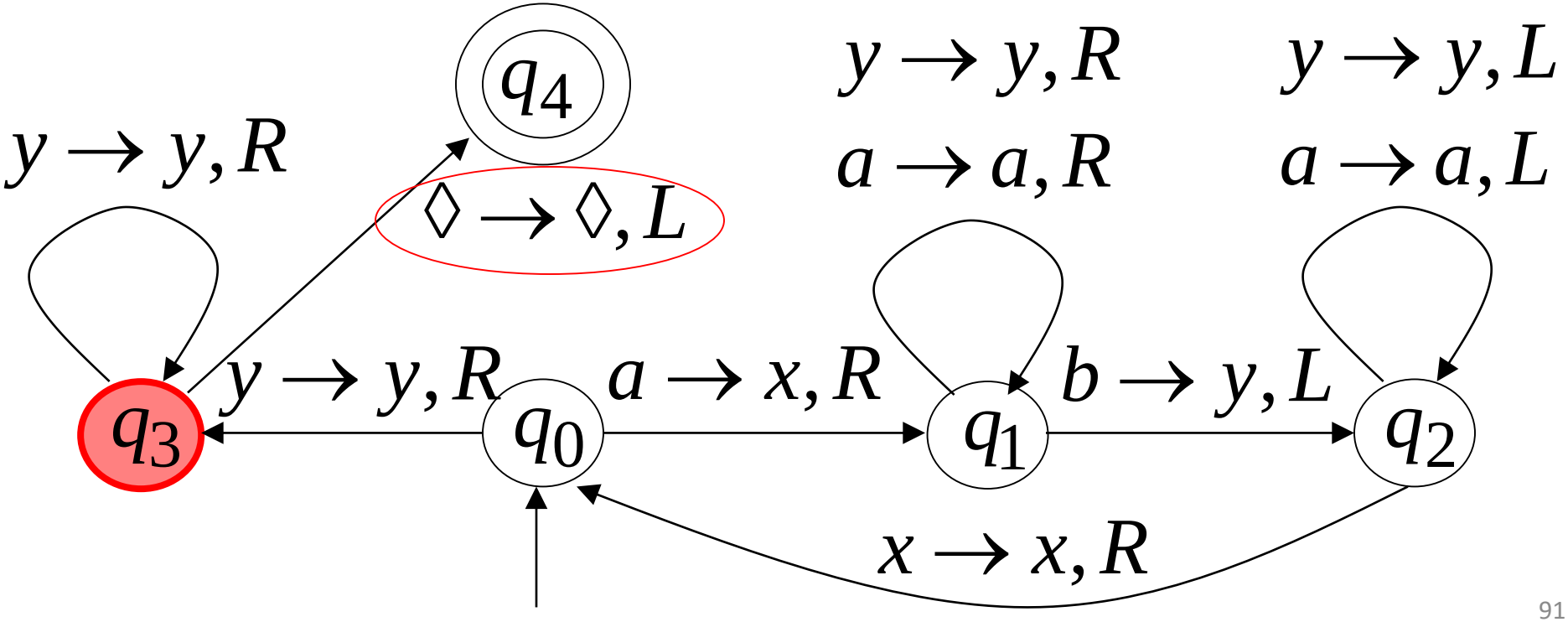
Time 11


◊ x x y y ◊ ◊
$q_3$

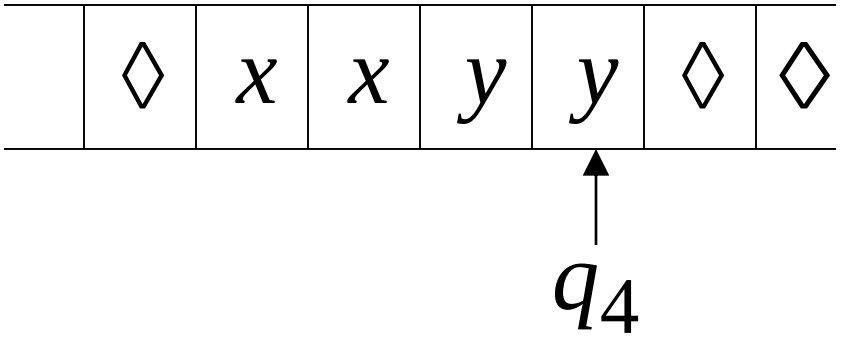

$q_4$
$y \to y, R$
$a \to a, R$
$y \to y, L$
$a \to a, L$
$y \to y, R$
$\Diamond \to \Diamond, L$
$y \to y, R$
$a \to x, R$
$b \to y, L$
$q_3$
$q_0$
$q_1$
$q_2$
$x \to x, R$

# Time 12

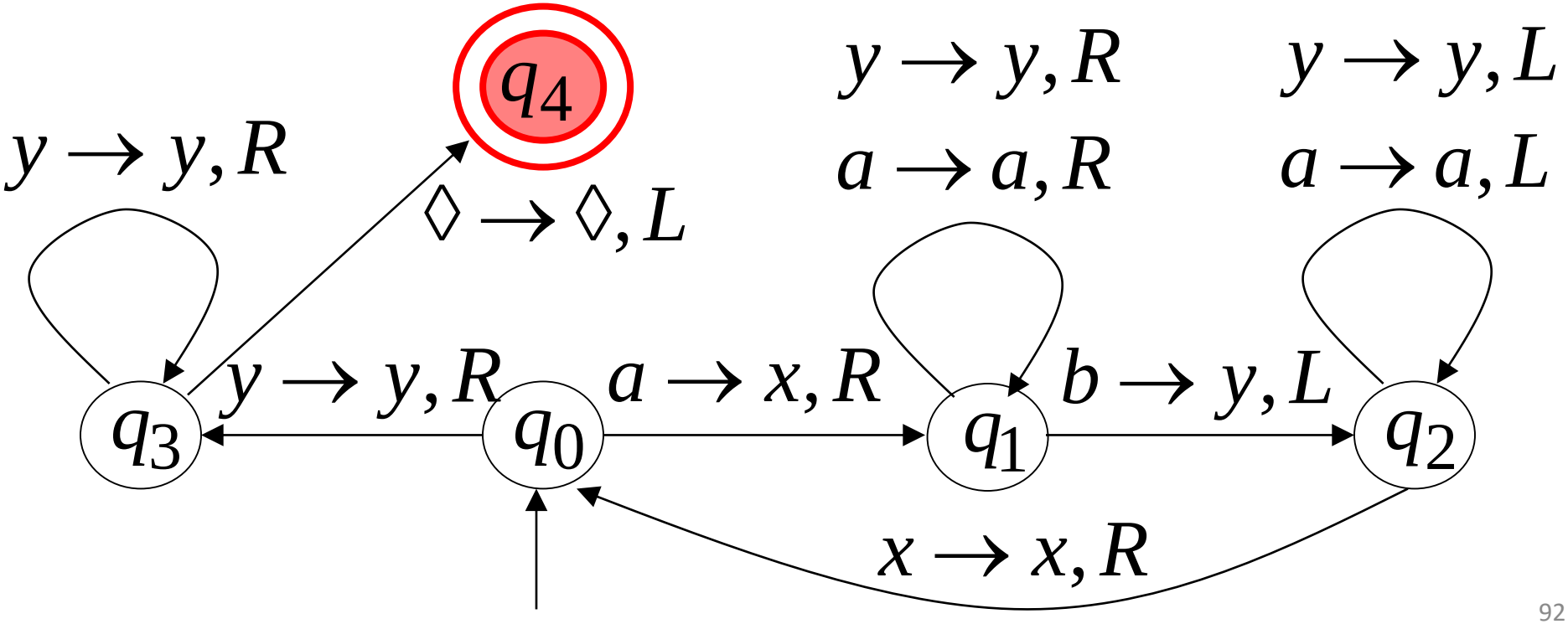

◊ x x y y ◊ ◊
$q_3$


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

Time 13




**Halt & Accept**



- **M = ({$q_0$,$q_1$, $q_2$, $q_3$, $q_4$}, {0, 1}, {0, 1, x, y, B}, δ, $q_0$, B, {$q_4$})**
- **M = ({$q_0$,$q_1$, $q_2$, $q_3$, $q_4$}, {a, b}, {a, b, x, y, B}, δ, $q_0$, B, {$q_4$})**
- **a=0, b=1**
- **Accepted**
- **ID:** $q_0 0011 \vdash x q_1 011 \vdash x 0 q_1 11 \vdash x q_2 0 Y 1 \vdash q_2 x 0 y 1 \vdash x q_0 0 y 1 \vdash x x q_1 y 1 \vdash x x y q_1 1 \vdash x x q_2 y y \vdash x q_2 x y y \vdash x x q_0 y y \vdash x x y q_3 y \vdash x x y y q_3 B \vdash x x y q_4 y B$
- **Rejected**
- **ID:** $q_0 0010 \vdash x q_1 010 \vdash x 0 q_1 10 \vdash x q_2 0 Y 0 \vdash q_2 x 0 y 0 \vdash x x q_1 y 0 \vdash x x y q_1 0 \vdash x x y 0 q_1 B$

  ❑**In state $q_1$, M has no move on tape symbol B**

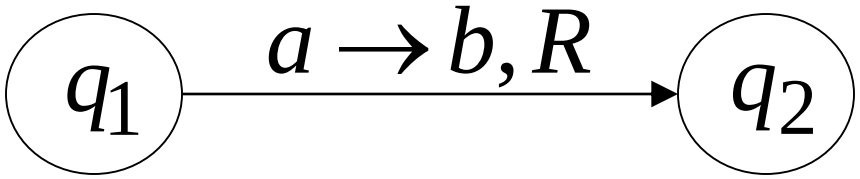
## Observation:

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

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

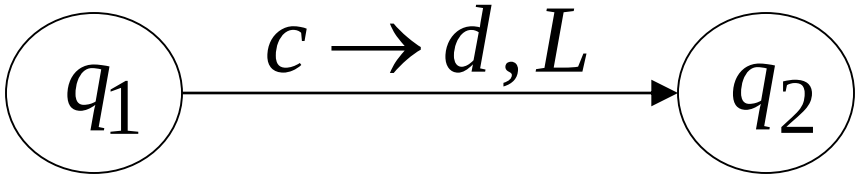
# Formal Definitions for Turing Machines

# Transition Function



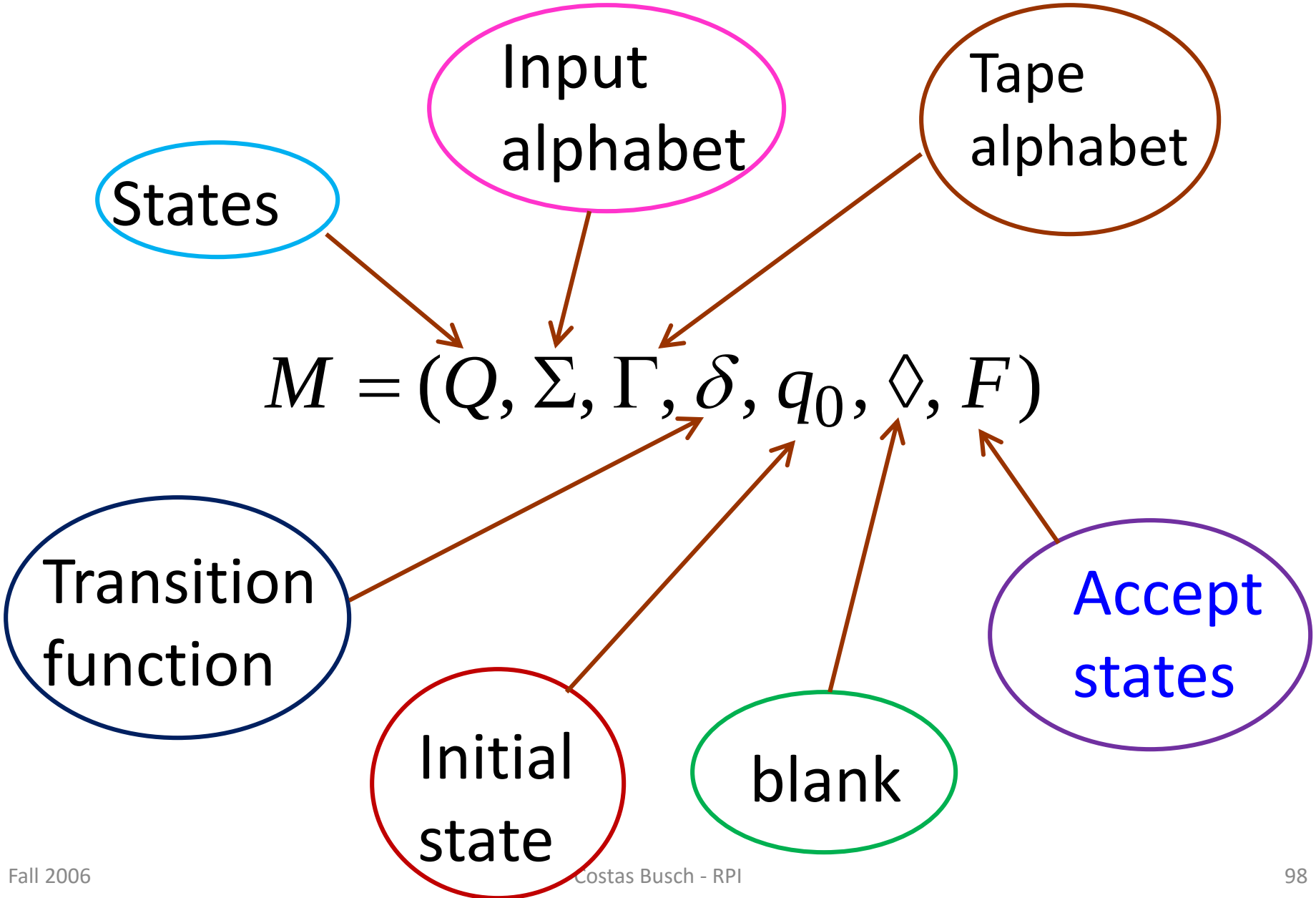

$$\delta(q_1, a) = (q_2, b, R)$$

# Transition Function

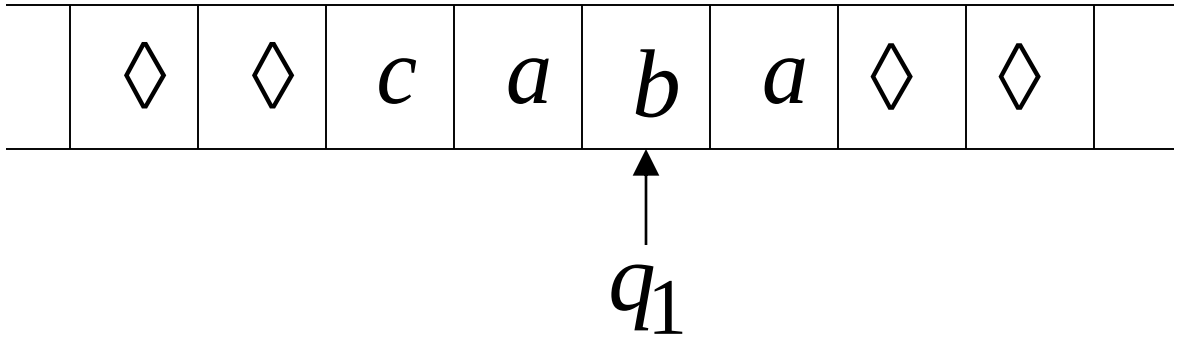



$$\delta(q_1, c) = (q_2, d, L)$$

# Turing Machine:

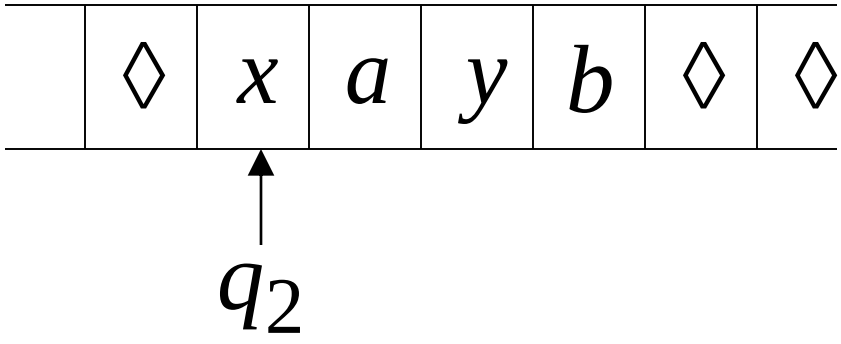


# Configuration

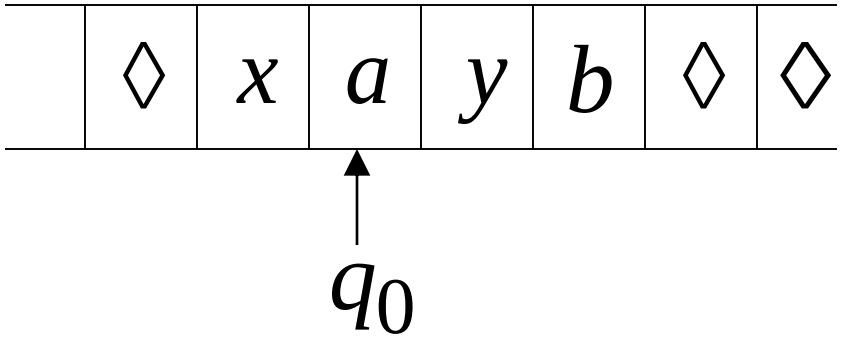



Instantaneous description: $ca\ q_1\ ba$

Time 4



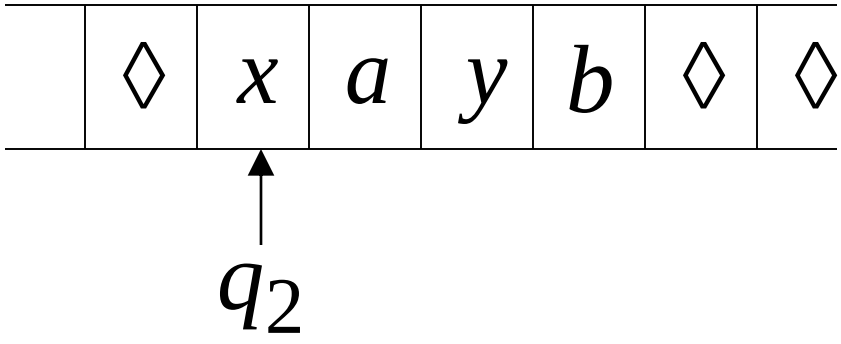

Time 5



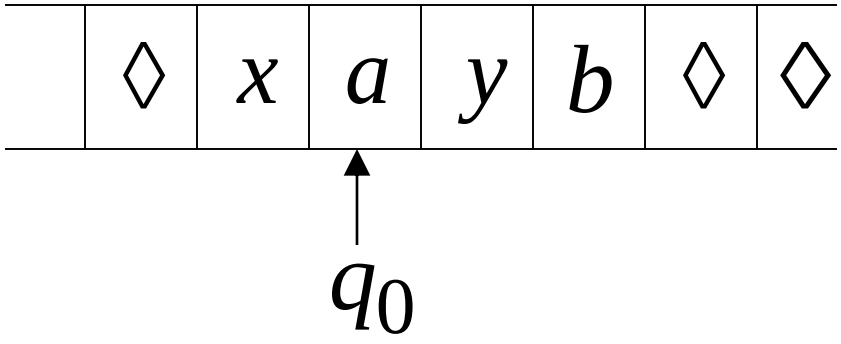

A Move: $q_2\ xayb \succ x\ q_0\ ayb$

(yields in one mode)

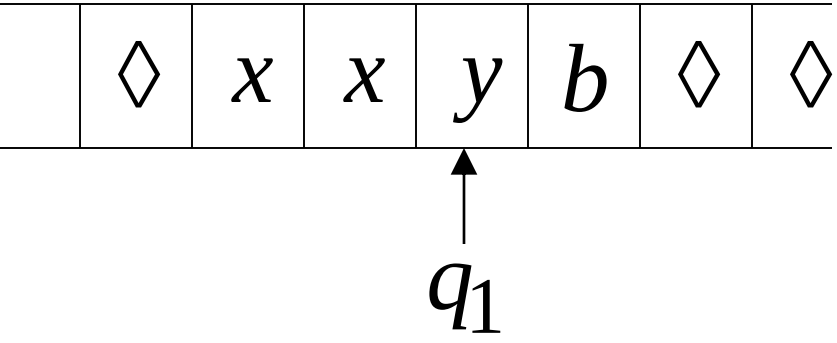
Time 4



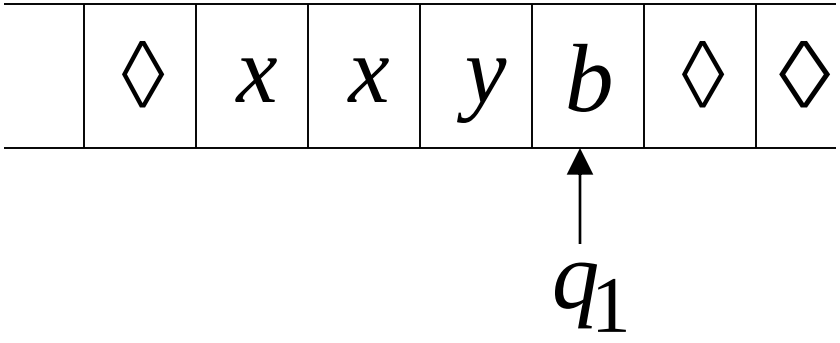

Time 5




Time 6




Time 7




A computation

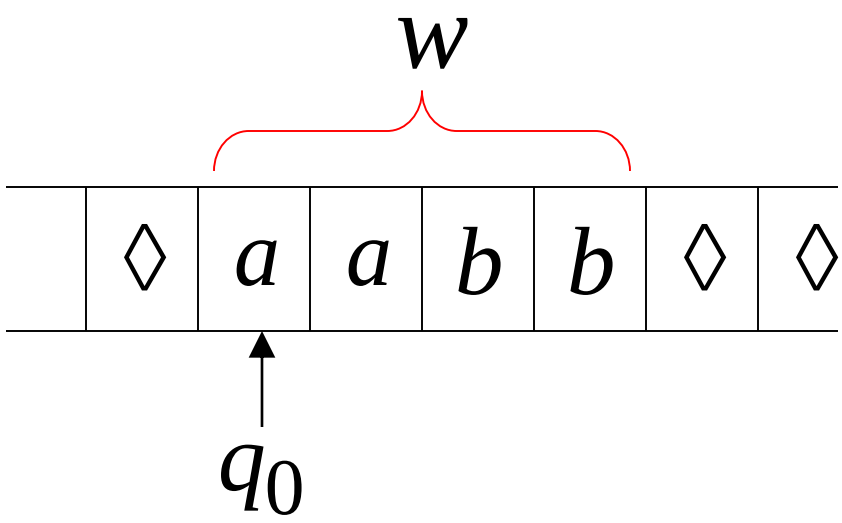
$$q_2\ xayb \succ x\, q_0\, ayb \succ xx\, q_1\, yb \succ xxy\, q_1\, b$$

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

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 $M$

$$L(M) = \{w : \quad q_0\, w \overset{*}{\succ} x_1\, q_f\, x_2\}$$

Initial state

Accept state

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

# TM: Computes the proper-subtraction function

$M = (q_0, q_1, \ldots, q_6\}, \{0,1\}, \{0, 1, B\}, \delta, q_0, B)$

| State | Symbols | | |
|---|---|---|---|
| | 0 | 1 | B |
| $q_0$ | $(q_1, B, R)$ | $(q_5, B, R)$ | - |
| $q_1$ | $(q_1, 0, R)$ | $(q_2, 1, R)$ | - |
| $q_2$ | $(q_3, 1, L)$ | $(q_2, 1, R)$ | $(q_4, B, L)$ |
| $q_3$ | $(q_3, 0, L)$ | $(q_3, 1, L)$ | $(q_0, B, R)$ |
| $q_4$ | $(q_4, 0, L)$ | $(q_4, B, L)$ | $(q_6, 0, R)$ |
| $q_5$ | $(q_5, B, R)$ | $(q_5, B, R)$ | $(q_6, B, R)$ |
| $q_6$ | - | - | - |

- This TM is not used to accept inputs, we have omitted the $7^{th}$ component (accepting states

# Transition Diagram

- See page 325, Figure 8.12

# Programming Techniques for TM

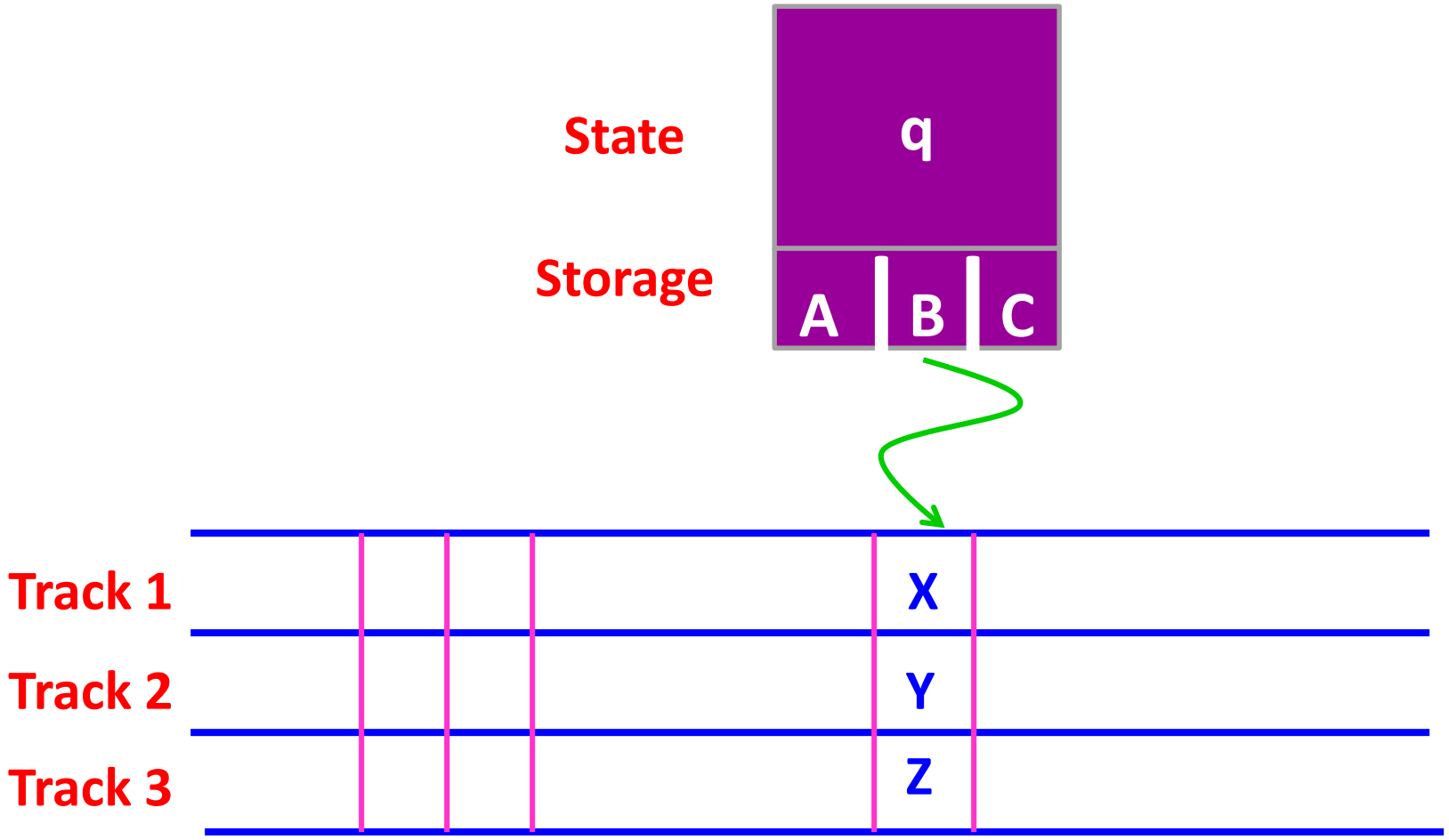
- Storage  in the state
- Multiple Tracks
- Subroutines

# Storage in the state

- Finite control can use
  - ❑not only to represent a position in the "program" of the TM,
  - ❑but to hold a finite amount of data.

**Techniques**

- ❑Storage
- ❑Multiple track




- **Finite control**: state q ++ data elements (A, B, C)
- The technique requires no extension to the TM Model;
- Think state as a **tuple = [q, A, B, C]**

# Multiple Tracks

- Tape composed of several tracks
- Each track can hold

➢ one symbol

➢ Tape alphabet of the TM consists of tuples, With one component for each track

- The cell scanned by the tape head contains the symbol **[X, Y, Z]**

# Subroutines

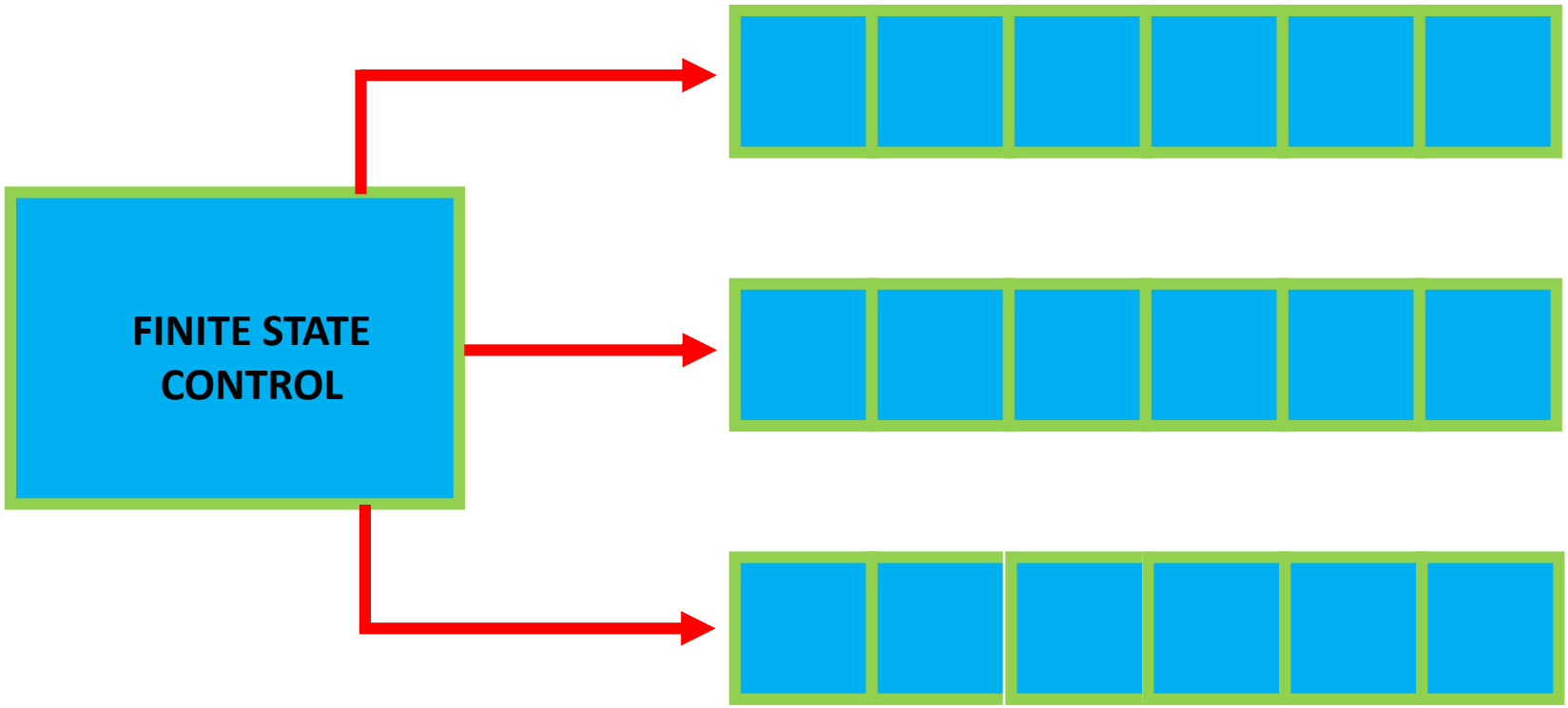
- TM build from a collection of interacting components/subroutines
- A TM subroutine is a set of states that perform some useful process
- This set of states includes 02 states:
  - a start state
  - another state that temporarily has no moves, and that serves as the return state to pass control to whatever other set of states called the subroutine
- The call of a subroutine occurs whenever there is a transition to its initial state
- Since the TM has no mechanisms for remembering a return address,
  - that is, a state to go to after it finishes,
- a TM call for one subroutine to be called from several states,
- we can make copies of the subroutine, using a new set of states for each copy
- The calls are made to the start states of different copies of the subroutine, and each copy returns to a different state.

# Extensions to the Basic TM

- Multi-tape TM: simulate real computers
- Nondeterministic TM: allow to make any of a finite set of choices of move in a given situation

# Multi-tape TM

- A finite control
- Some finite number of tapes
- Each tapes is divided into cells & each cell can hold any symbol of the finite tape alphabet
- The set of tape symbols includes
  - a blank
  - has a subset called the input symbols,
- The set of states includes: initial state and some accepting states.


FINITE STATE
CONTROL

# Initial condition

1. The input, a finite sequence of input symbol, is placed on the 1st tape
2. All other cells of all the tapes hold B
3. The finite control is in the initial state
4. The head of the 1st tape is at the left end of the input
5. All other tape heads are at some arbitrary cell. Since tapes other than the 1st tape are completely B, it does not matter where the head is placed initially; all cells of these tapes "look" the same.

# Moves

- **Depends**

- The state
- The symbol scanned by each of the heads

**Actions:**

1. The control enters a new state
2. On each tape, a new tape symbol is written on the cell scanned
3. Each of the tape heads makes a move, which can be either left, right, or stationary. The heads move independently, so different heads may move in different directions, & some may not move at all.

**Theorem: Every Multitape Turing Machine can be transformed into a single tape Turing Machine**

Say that M has k tapes.

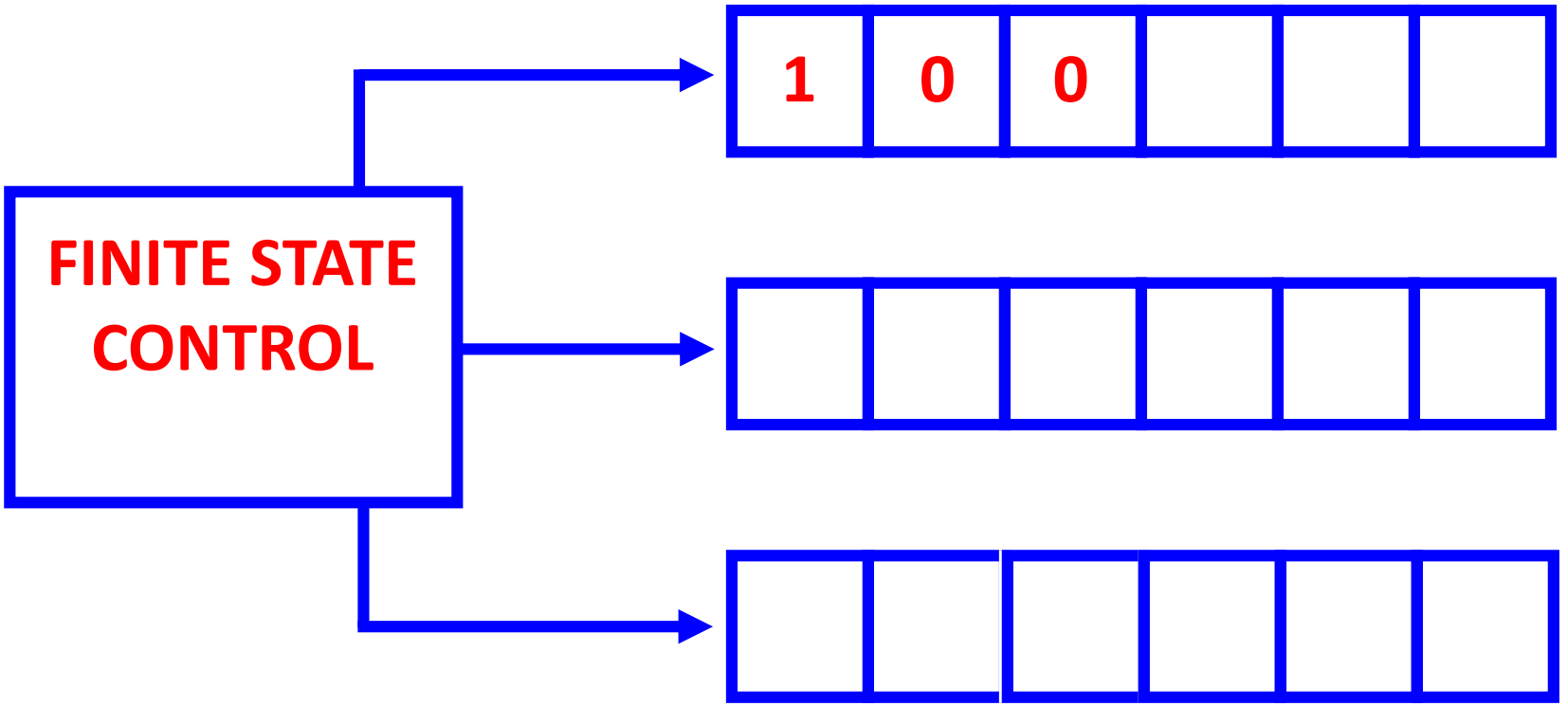
- ❑Create the TM S to simulate having k tapes by interleaving the information on each of the k tapes on its single tape
  - ➢Use a new symbol # as a delimiter to separate the contents of each tape
  - ➢S must also keep track of the location on each of the simulated heads

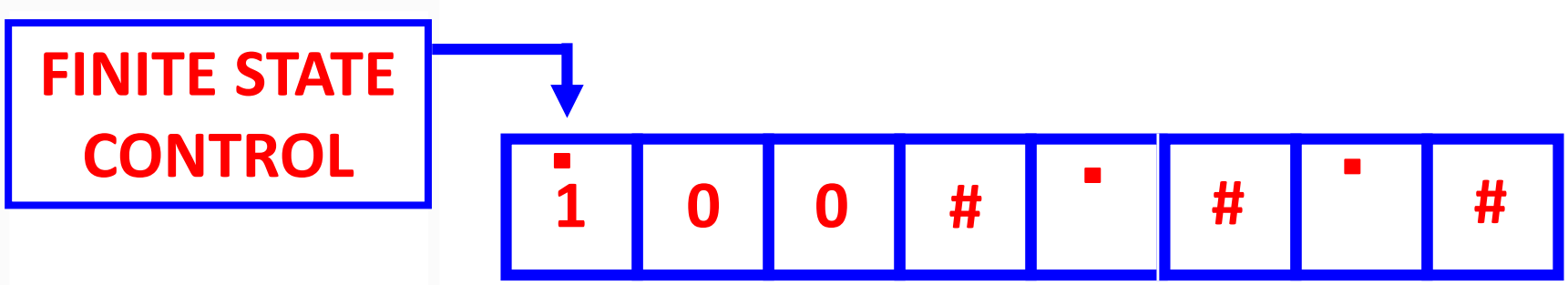
❑We write down the contents of all these tapes in order, & put a # between two tapes.

✓To keep track of where all the heads of the multitape machine are, we double the size of the alphabet to include

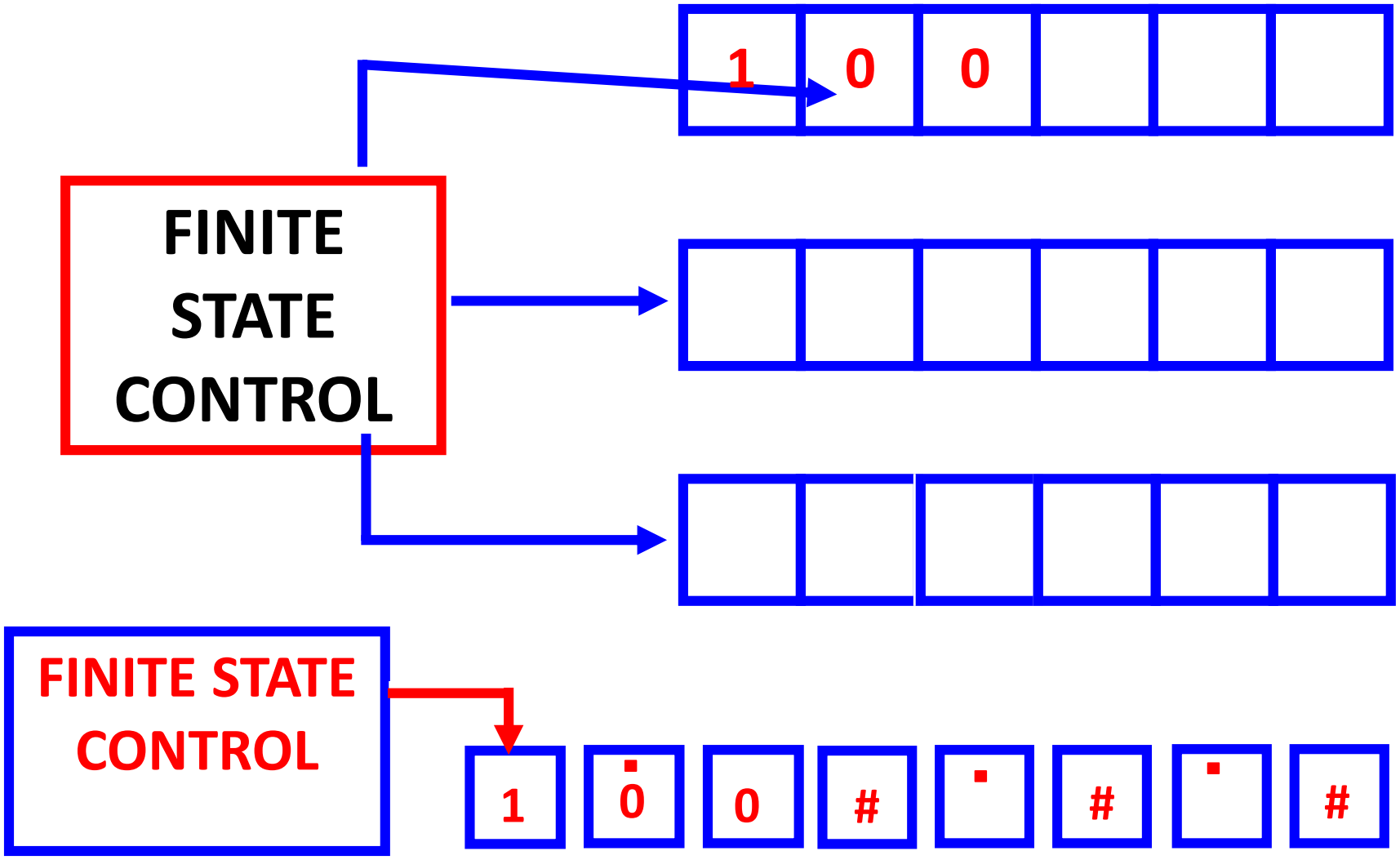
✓"marked" symbols, & a mark above a symbol indicates the head position

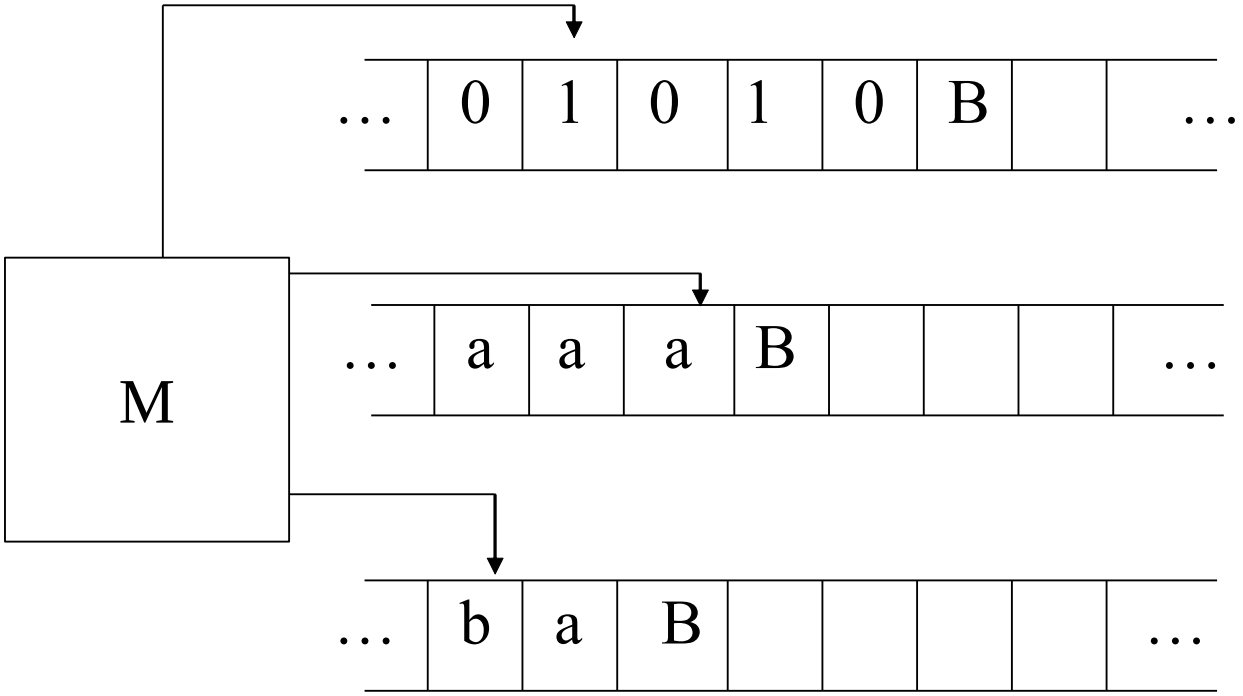
# Multitape Machine



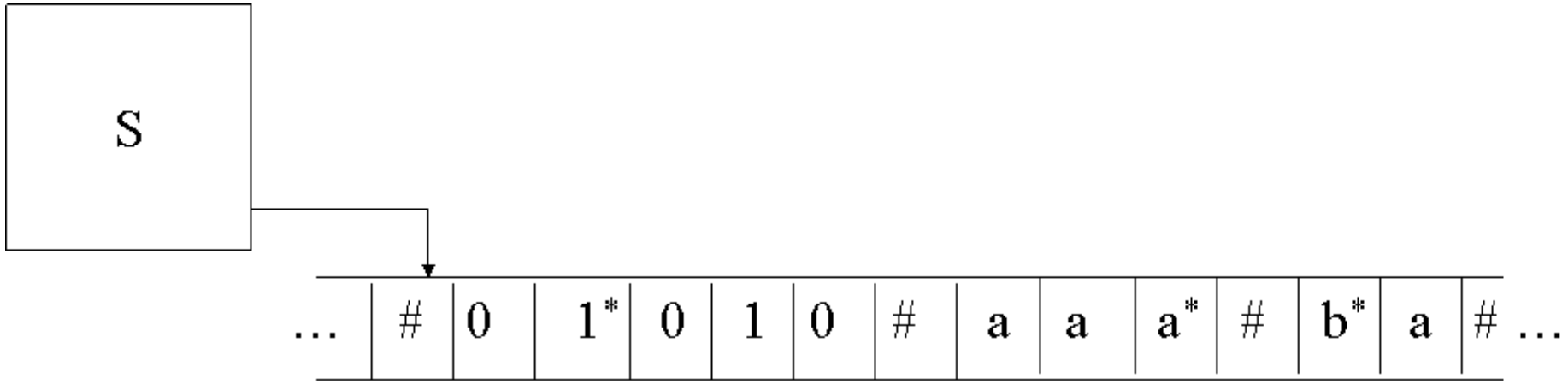



- When some tape head moves, we move the mark.
- When the multi-tape machine makes one of its tape contents "longer" (by writing over some blanks), the simulation of it moves all the tape content beyond over by one.
  - If at any point S moves one of the virtual tape heads onto a #, then this action signifies that M has moved the corresponding head onto the previously unread blank portion of that tape.
  - To accommodate this situation, S writes a blank symbol on this tape cell and shifts the tape contents to the rightmost # by one, adds a new #, and then continues back where it left off


1
0
0
FINITE
STATE
CONTROL
FINITE STATE
CONTROL
1
0
0
#
#
#




## Equivalent Single Tape Machine:



# Multistack Machines

**Based on PDA**

- TM can accept languages that are not accepted by any PDA with one stack
- The PDA two stacks, it can accept any language that a TM can accept

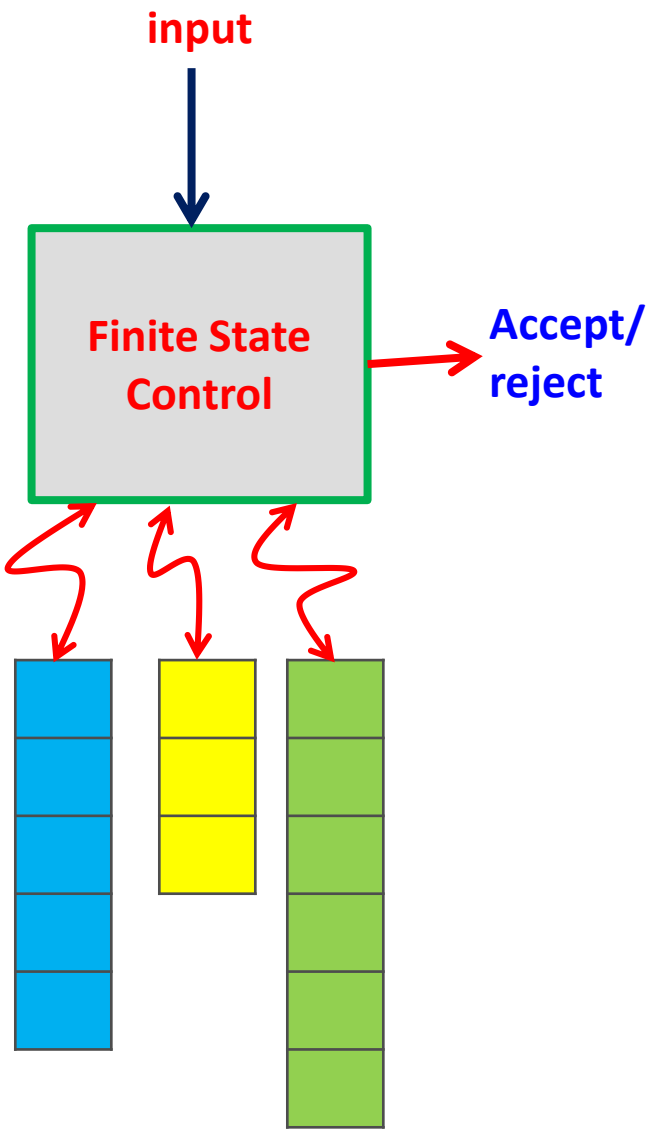
- "**Counter machines**"
- Only ability
  - to store a finite number of integers ("counters")
  - to make different moves depending on which if any of the counters are currently 0.

# Multistack Machines

- The counter machine can only add/subtract one from the Counter, & cannot tell 02 different non-zero counts from each other
- In effect, a counter is like a stack on which we can place only 02 symbols:
  - **a bottom-of-stack marker that appears only at the bottom,**
  - **one other symbol that may be pushed & popped from the stack**

# Multistack Machines

- **k- stack machine**: deterministic PDA with k-stacks
  - It obtain its input, from an input source
  - The multistack machine has a finite control, which is in one of a finite set of states.
  - It has a finite stack alphabet, which it uses for all its stacks



# Actions

1. The state of the finite control
2. The input symbol read, which is chosen from the finite input alphabet. Alternatively, the multistack machine can make a move using $\varepsilon$ input, but to make the machine deterministic, there cannot be a choice of and $\varepsilon$-move or a non-$\varepsilon$-move in any situation
3. The top stack symbol on each of its stacks

# In one move

a) Change to a new state

b) Replace the top symbol of each stack with a string of zero or more stack symbols. There can be a different replacement string for each stack

$$\delta (q, a, X_1, X_2, \ldots\ldots, X_k) = (p, Y_1, Y_2, \ldots\ldots, Y_k)$$

# Counter Machines

- A counter machine (Two ways):

1. The counter machine has the same as the multistack machine but in place of each stack is a counter

-counter hold any non-negative integer, but we can only distinguish between zero & non-zero counters

-the move of the counter machine depends on its state, input symbol & counters (which are zero)

- In one move

a) Change state

b) Add/subtract 1 from any of its counters, independently. However, a counter is not allowed to become negative, so it cannot subtract 1 from a counter that is currently 0

- 2. A counter machine may also be regarded as a restricted multistack machine.
- **Restrictions:**

a) There are only 02 stack symbols: $Z_0$ & X

b) $Z_0$ is initially on each stack

c) Replace $Z_0$ only by a string of the form $X^iZ_0$, for some $i \geq 0$

d) Replace X only by $X^i$ for some $i \geq 0$. $Z_0$ appears only on the bottom of each stack, & all other stack symbols, if any, are X

# Simulating a TM by Computer

- Given a TM M, we write a program that acts like M
- Since there are only a finite number of states & a finite number of transitions rules, program can encode states as character strings & use a table of transitions, which it looks up to determine each move.
- Likewise, the tape symbols can be encoded as character strings of a fixed length, since there are only finite number of tape symbols.

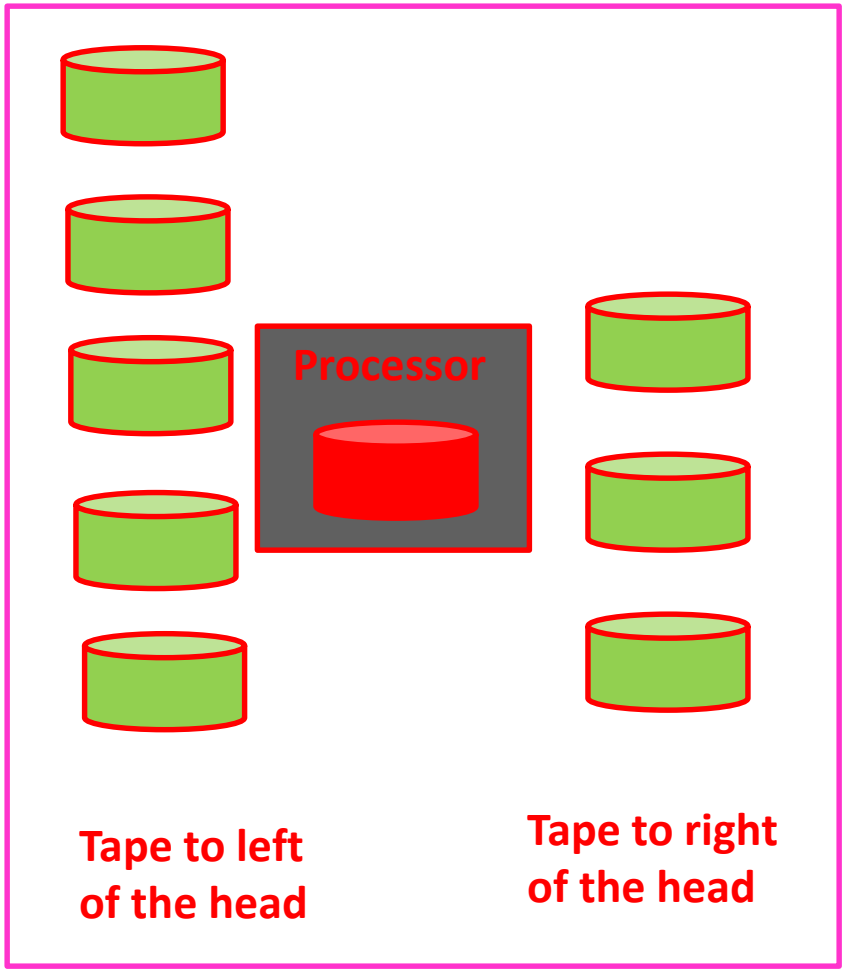
# How program is to simulate the TM tape?

- Tape grow infinitely long, but the computer's memory-main memory, disk & other storage devices---are finite
- Can we simulate an infinite tape with a fixed amount of memory?

❑If there are no opportunity to replace storage devices, then in fact we cannot; a computer would then be a finite automation & only languages it could accept would be regular

- However, common computers have swappable storage devices, perhaps a “Zip” disk.
- In fact, the typical hard disk is removable & can be replaced by an empty, but otherwise identical disk
- **Assume**: as many disks as the computer needs is available
- Arrange that the disks are placed in 02 stacks

- One stack holds the data in cells of the TM tape that are located significantly to the left of the tape head
- The other stack holds data significantly to the right of the tape head
- The further down the stacks, the further away from the tape head the data is



- If the tape head of TM move sufficiently far to the left that it reaches cells that are not represented by the disk currently mounted in the computer, then it prints a message, "swap left"
- The currently mounted disk is removed by a human operator & placed on the top of the right stack
- The disk on top of the left stack is mounted in the computer & computation resumes

- Similarly, if TM's tape head reaches cells so far to the right that these cells are not represented by the mounted disk, the a "swap right" message is printed
- The human operator moves the currently mounted disk to the top of the left stack, & mounts the disk on top of the right stack in the computer
- If either stack is empty when the computer asks, that a disk from that stack be mounted, then the TM has entered an all-blank region of the tape
- In that case, the human operator must go to the store & buy afresh disk to mount.

# Comparing the running times of computers & Turing machines

- **See textbook: P#361**