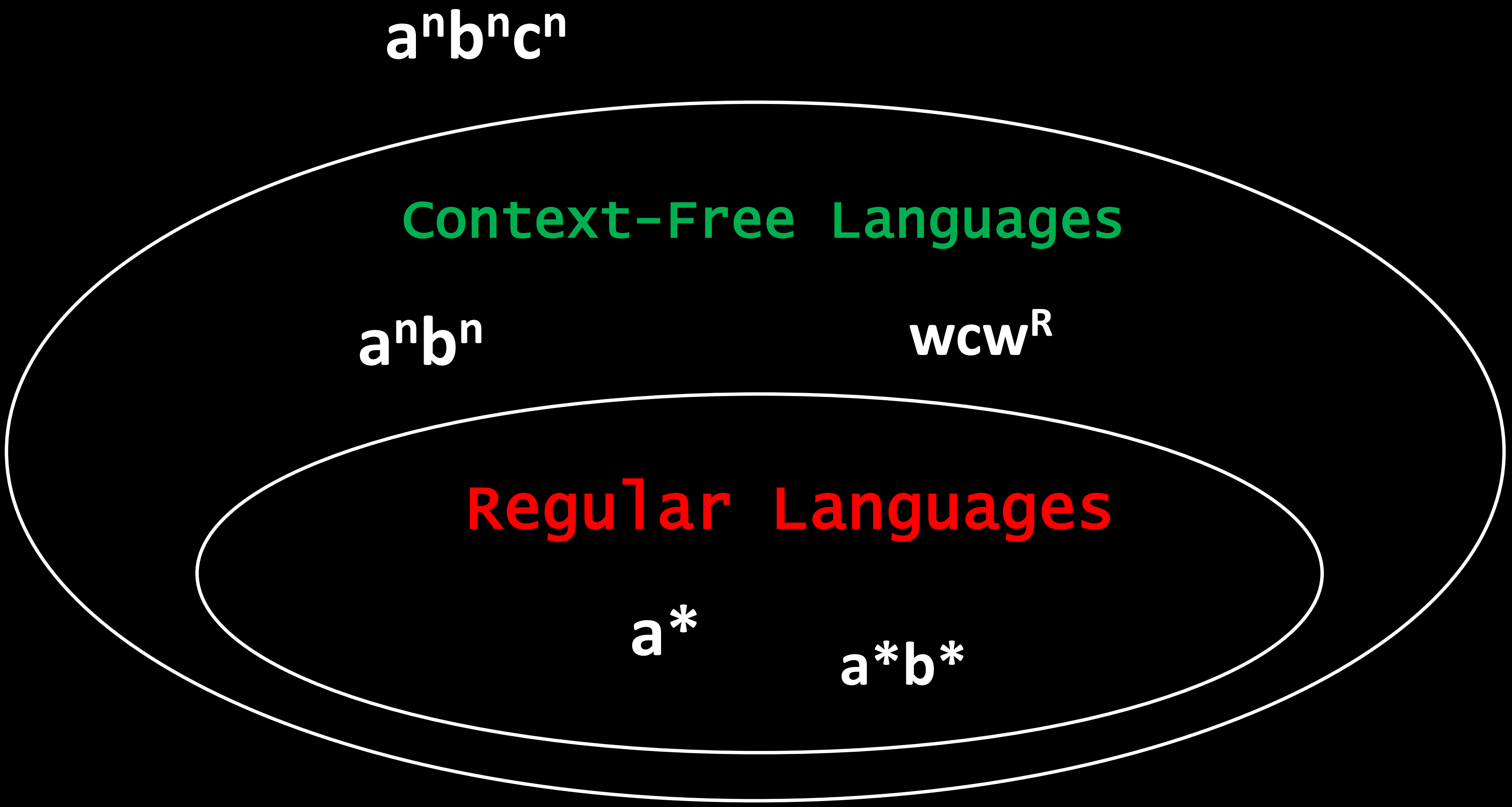
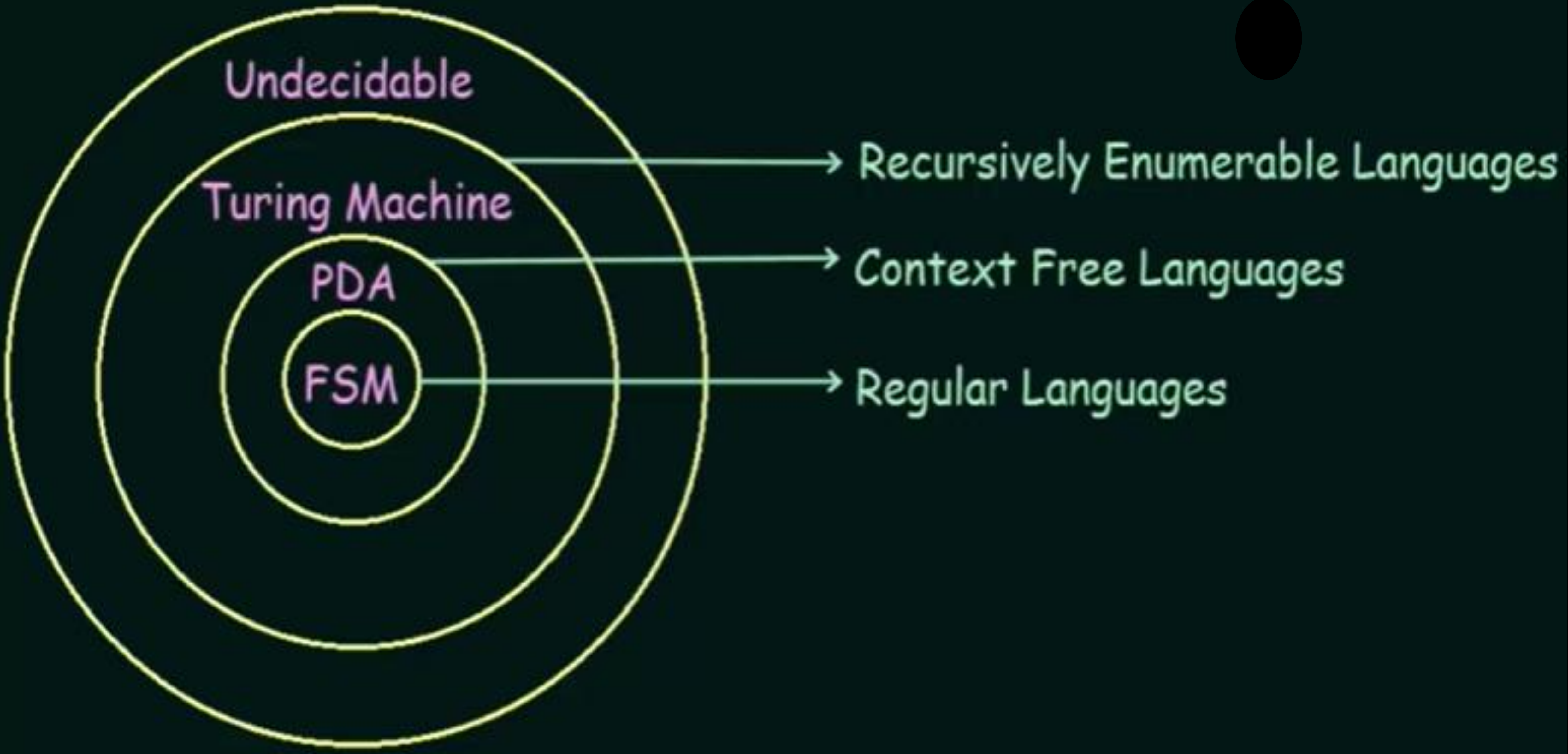

بالعِلم نَرْتَقي

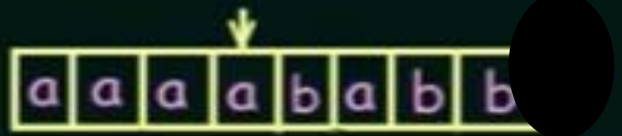
Turing machines

The language hierarchy



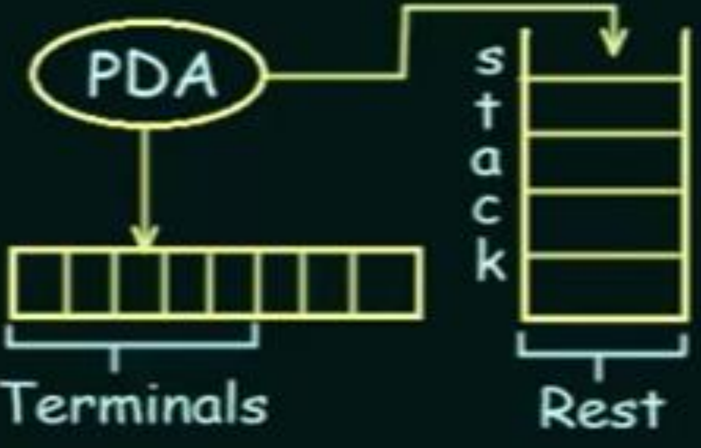

Undecidable
Turing Machine
PDA
FSM
Recursively Enumerable Languages
Context Free Languages
Regular Languages

FSM: The Input String



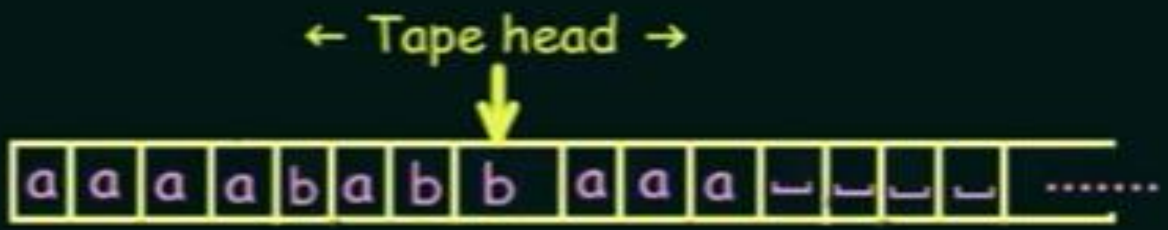
PDA: -> The Input String

-> A Stack



TURING MACHINE:

-> A Tape

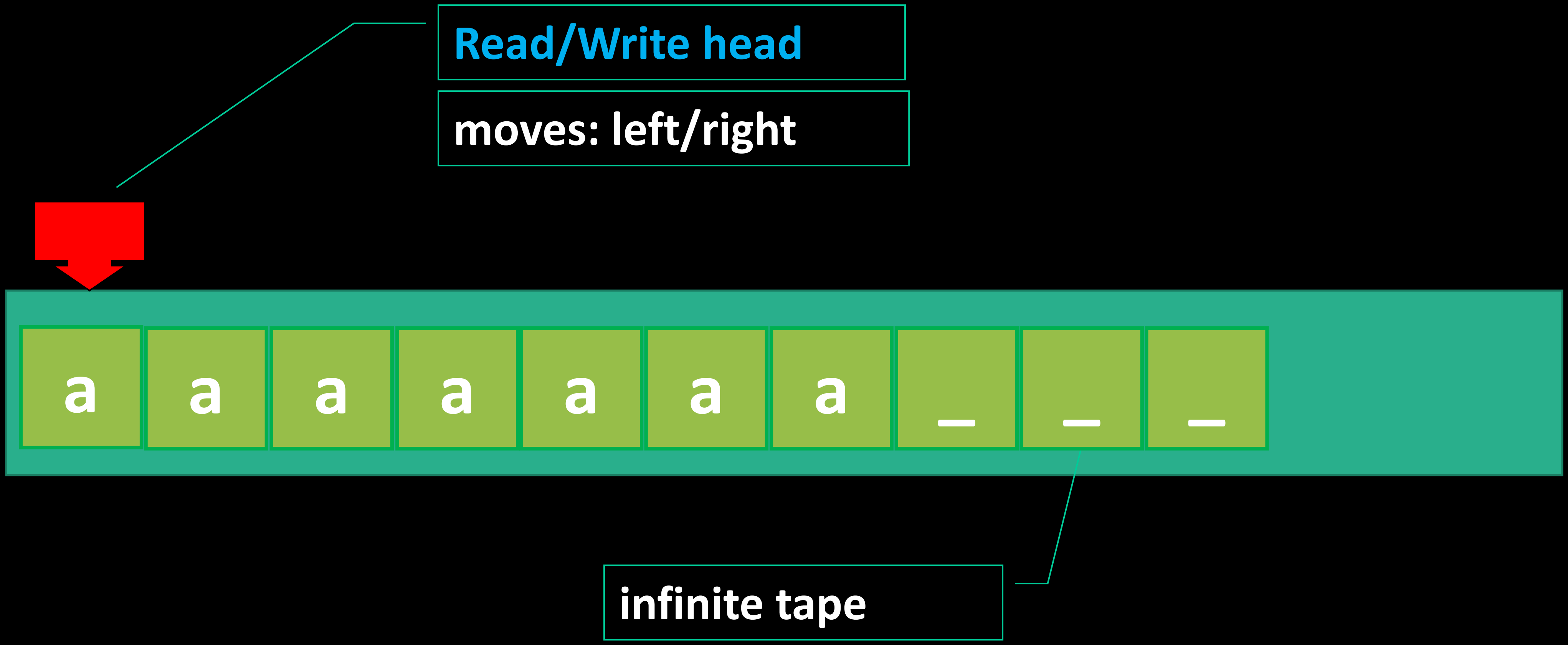


Tape Alphabets: $\Sigma = \{0,1,a,b,x,Z_0\}$

The Blank ␣ is a special symbol. ␣ $\notin \Sigma$

The blank is a special symbol used the fill the infinite tape

Schematic of a Turing Machine



Rules of Operation - 1

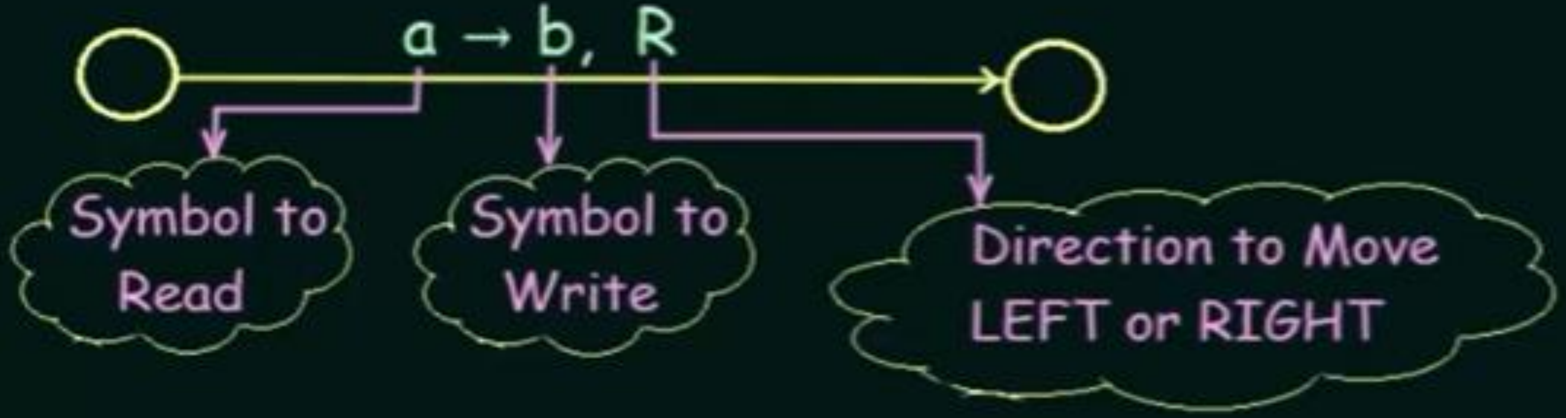
At each step of the computation:

-> Read the currect symbol

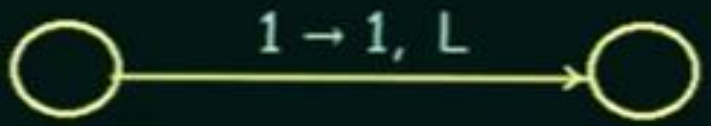
-> Update (i.e. write) the same cell

-> Move exactly one cell either LEFT or RIGHT

If we are at the left end of the tape, and trying to move LEFT, then do not move. Stay at the left end



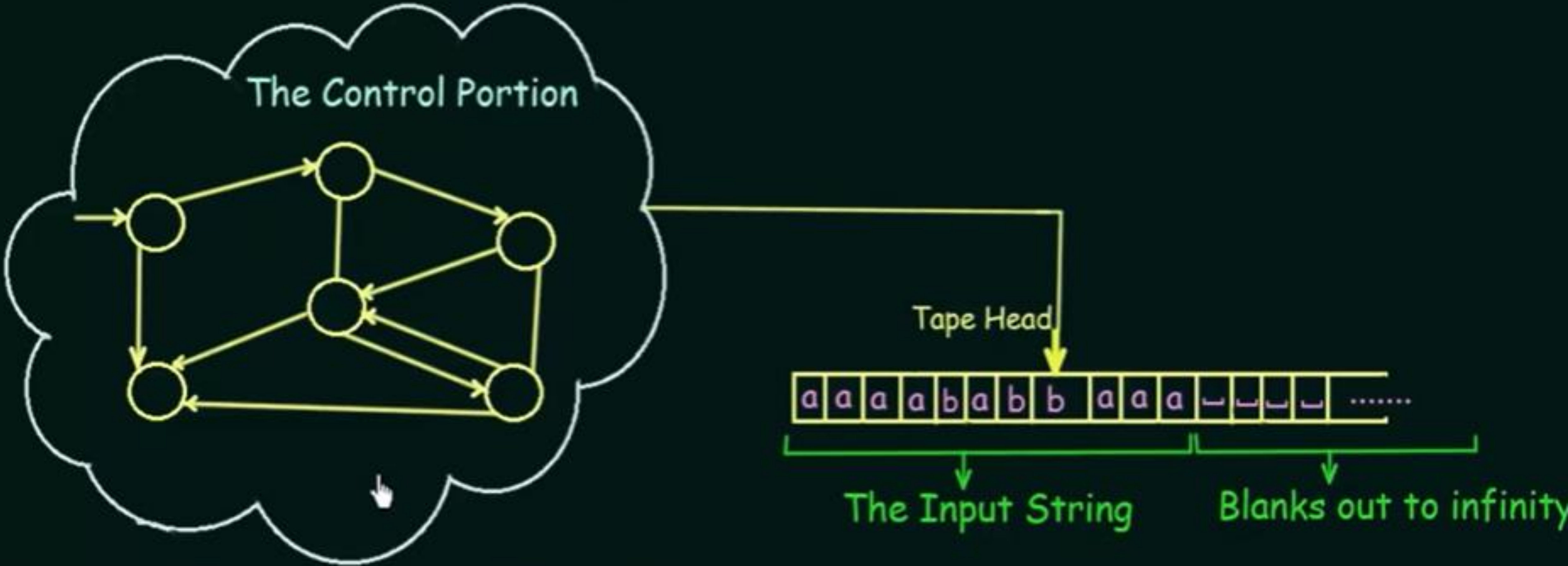
If you dont want to update the cell,
JUST WRITE THE SAME SYMBOL



Abilities of a Turing Machine

- A Turing machine is similar to a DFA or a PDA.
- It has the following abilities.
  - It can **read** or **write** to the tape.
  - It can **move left** or **move right** on the tape.
  - It **halts** as soon as it reaches either the special **accept state** or the special **reject state**.

Turing Machine - Introduction (Part-2)



The Control Portion similar to FSM or PDA

The PROGRAM

It is deterministic

Rules of Operation - 2

- -> Control is with a sort of FSM
- -> Initial State
- -> Final States: (there are two final states)

    1) The ACCEPT STATE

    2) The REJECT STATE

- -> Computation can either

    1) HALT and ACCEPT

    2) HALT and REJECT

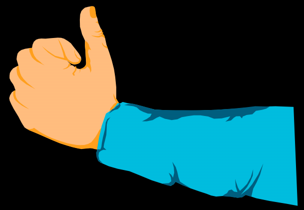
    3) LOOP (the machine fails to HALT)

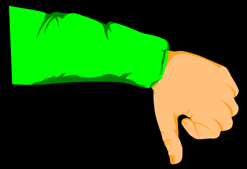
Formal Definition of a TM

A deterministic Turing Machine is a tuple consisting of several objects.

1. Q - a finite set of states.
2. Σ - the input alphabet, finite set not containing the blank symbol .
3. Γ - the tape alphabet, where $\Sigma \subseteq \Gamma$ and $\_ \in \Gamma$.
4. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ - the transition function.
5. q_0 - the start state

6. $q_{accept} \in Q$ - the accept state.





7. $q_{reject} \in Q$ - the reject state. $q_{reject} \neq q_{accept}$.

Turing's Thesis:

Turing's Thesis states that any computation that can be carried out by mechanical means can be performed by some Turing Machine.

Few arguments for accepting this thesis are:

i. Anything that can be done on existing digital computer can also be done by Turing Machine.
ii. No one has yet been able to suggest a problem solvable by what we consider on algorithm, for which a Turing Machine Program cannot be written.

Turing Machine (Formal Definition)

A Turing Machine can be defined as a set of 7 tuples

$$(Q, \Sigma, \Gamma, \delta, q_0, b, F)$$

$Q \rightarrow$ Non empty set of States

$\Sigma \rightarrow$ Non empty set of Symbols

$\Gamma \rightarrow$ Non empty set of Tape Symbols

$\delta \rightarrow$ Transition function defined as

$$Q \times \Sigma \rightarrow \Gamma \times (R/L) \times Q$$

$q_0 \rightarrow$ Initial State

$b \rightarrow$ Blank Symbol

$F \rightarrow$ Set of Final states (Accept state & Reject State)

Thus, the Production rule of Turing Machine will be written as

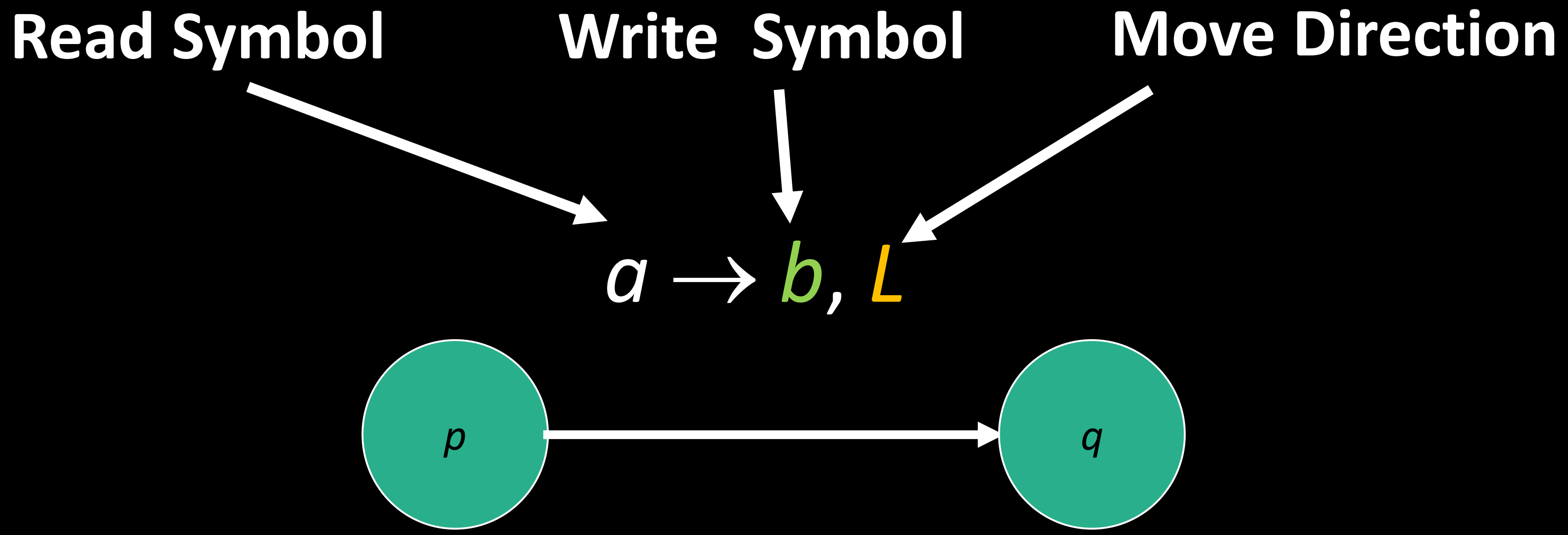
$$\delta\,(q_0, a) \rightarrow (q_1, y, R)$$

Recursively Enumerable Language:

A Language L and $\sum$ is said to be Recursively Enumerable if there exists a Turing Machine that accepts it.

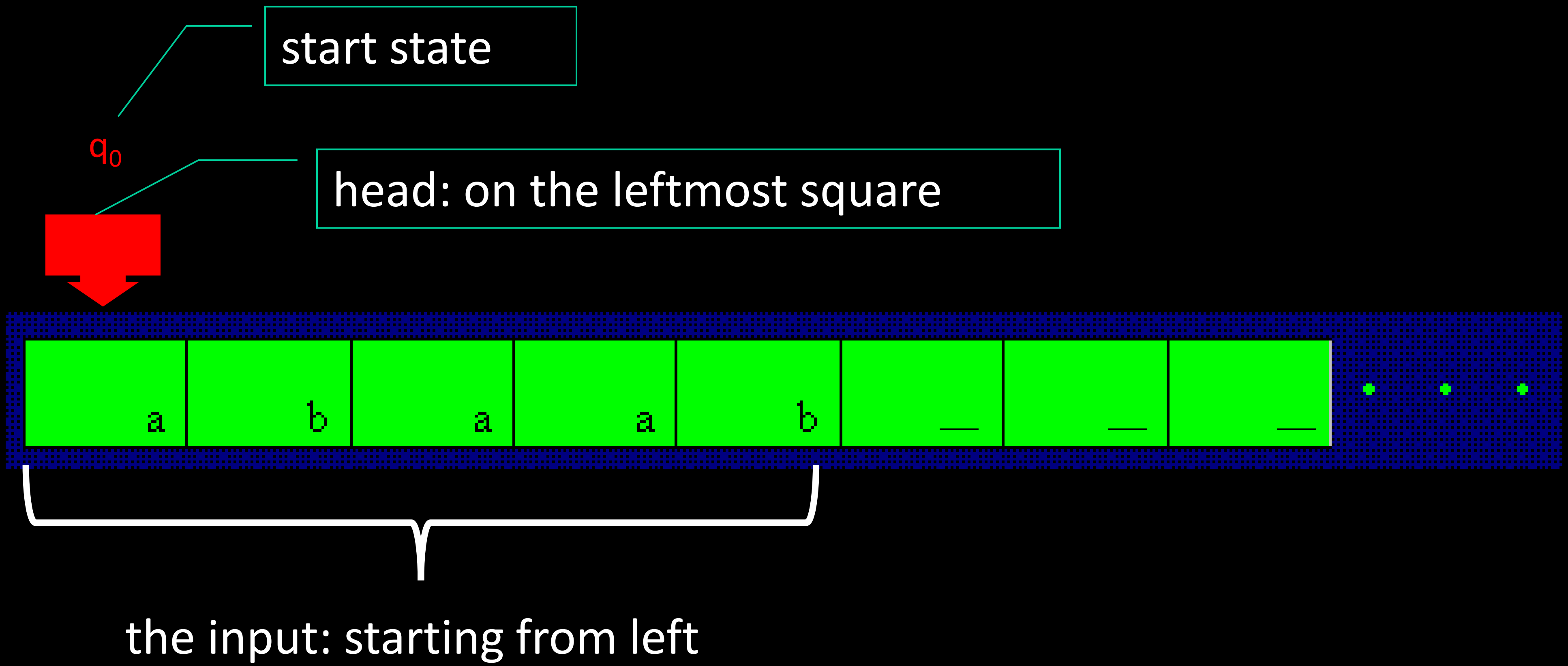
Transitions

- We will represent the transition $\delta(p, a) = (q, b, L)$ as



Computations

The Start Configuration

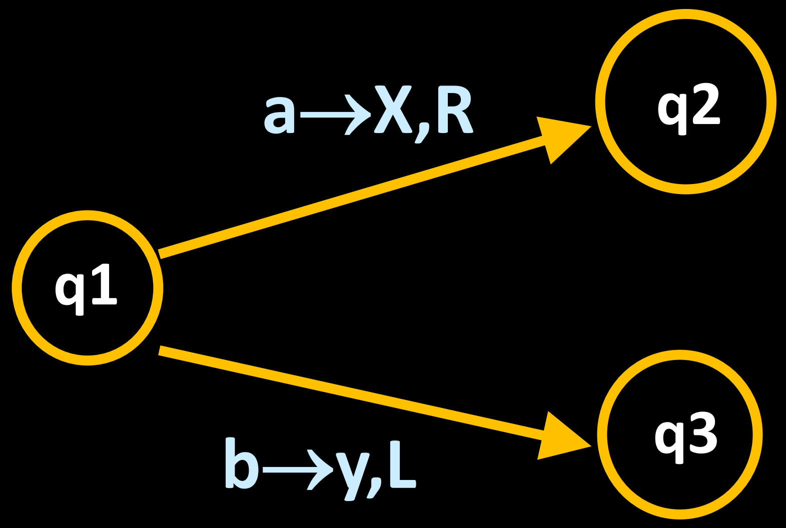


The Language a TM Accepts

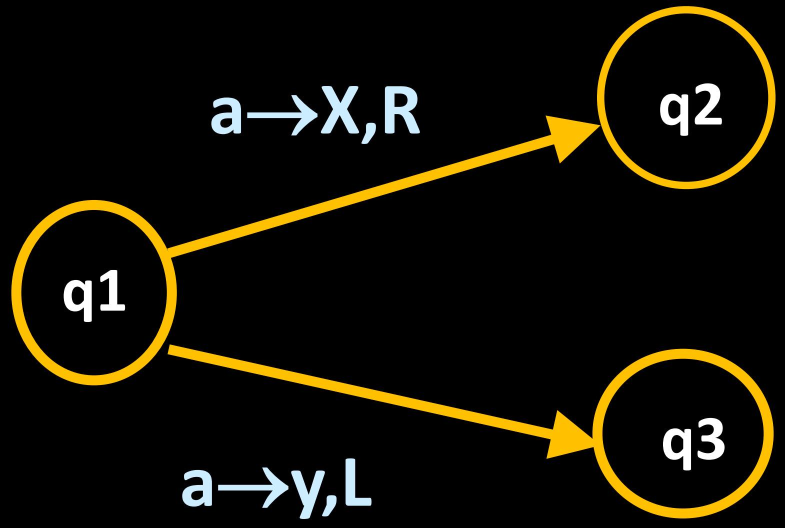
- A Turing Machine accepts its input, if it reaches an accepting configuration.
- The set of inputs it accepts is called its **language**.

Turing Machines are deterministic

Allowed

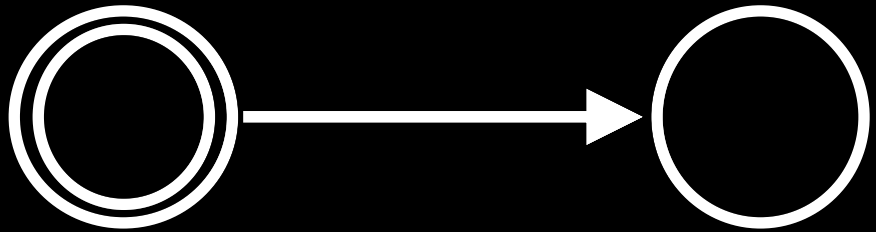


Not Allowed



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

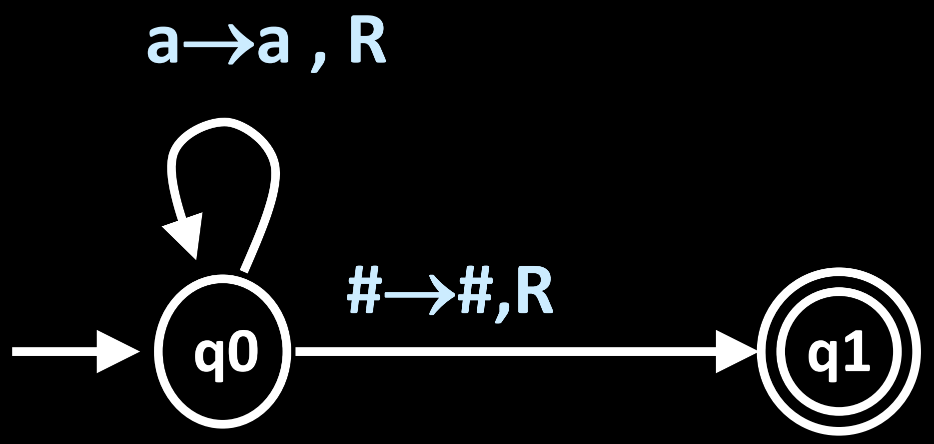
Reject Input ▶ If machine halts in a non-final state

or

If machine enters an *infinite loop*

Turing machine example

A Turing machine that accepts language a^*



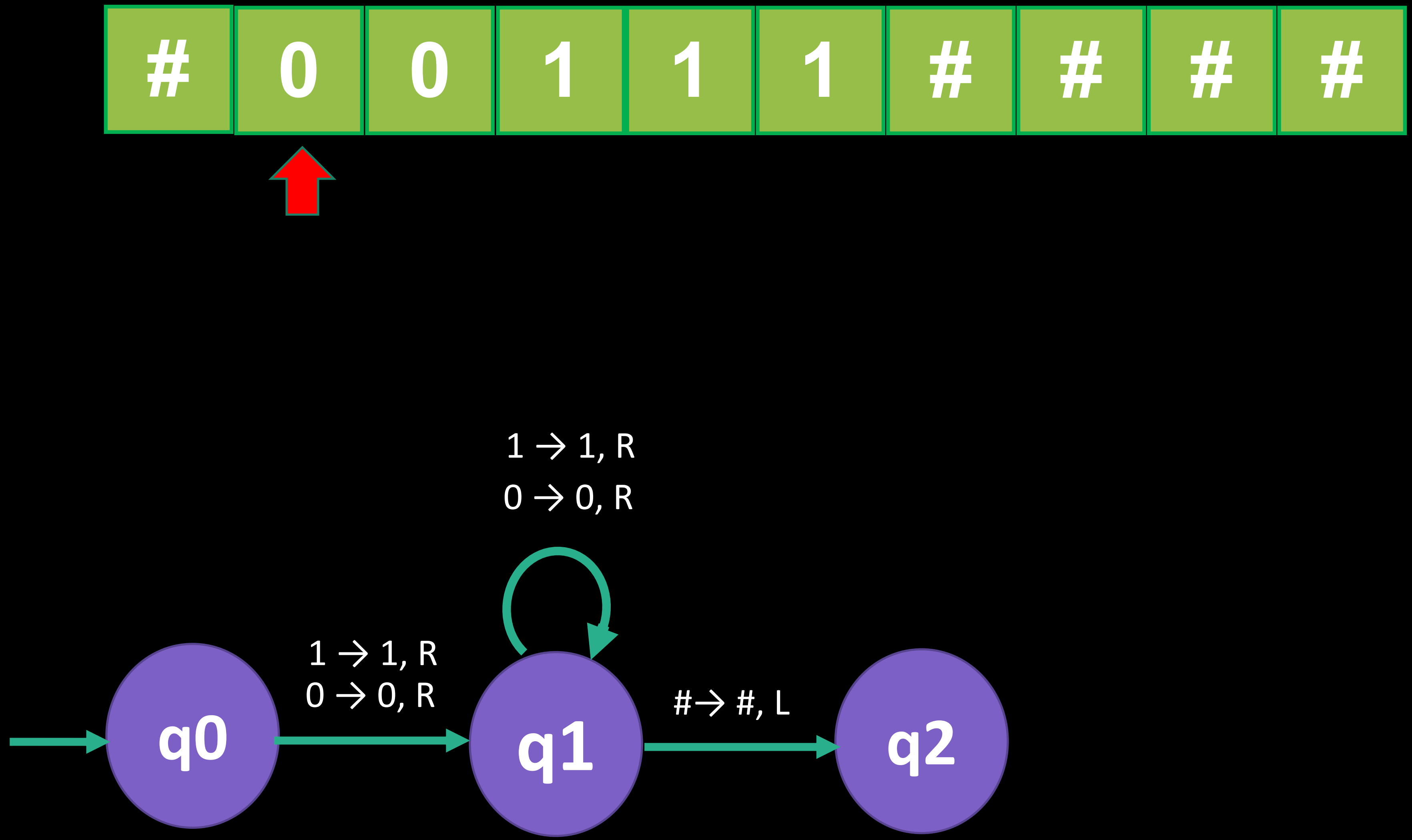
a	a	a	a	a	a	a	#	#	#

Example

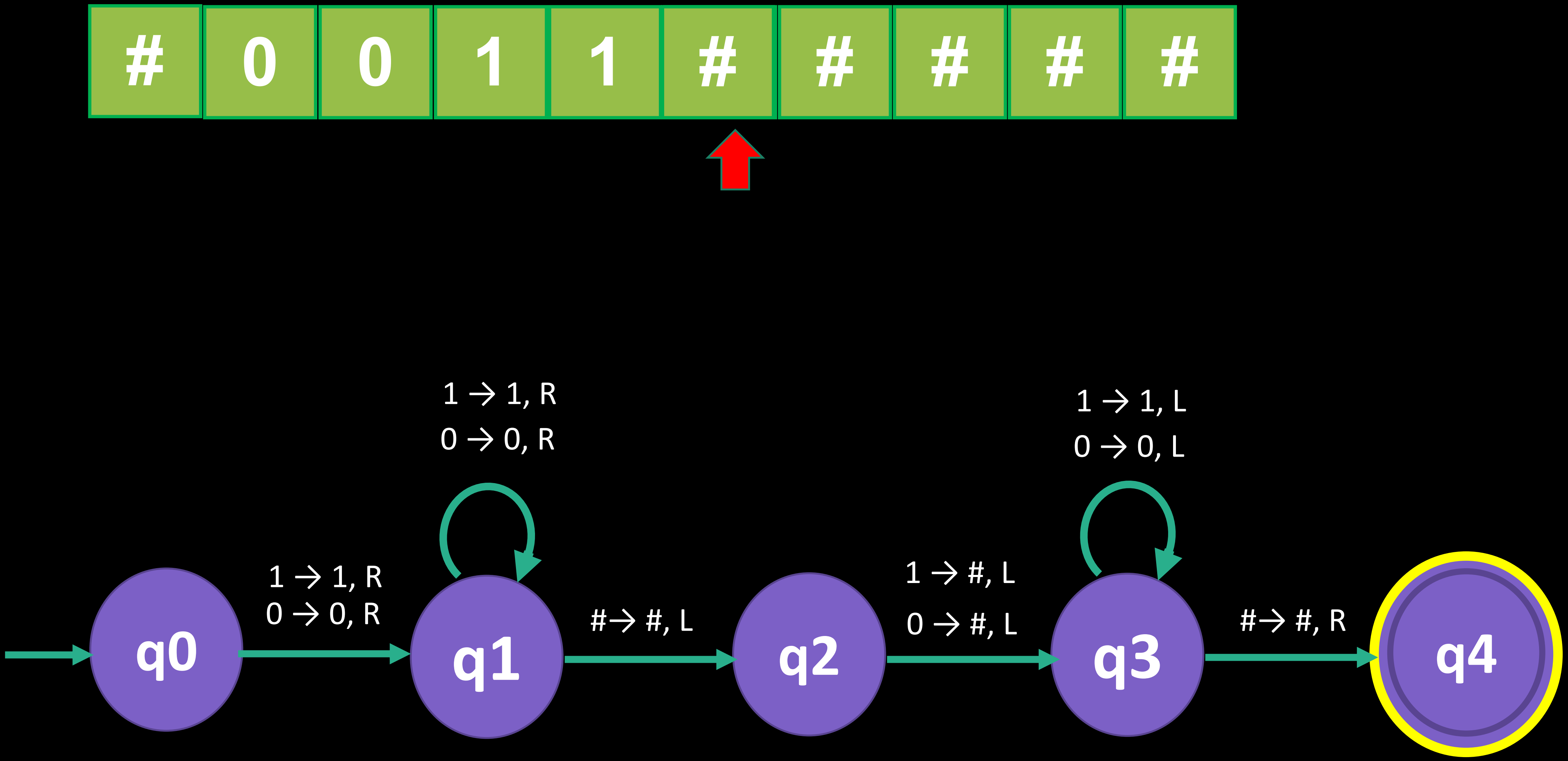
Turing Machine to erase the first symbol in the right side, where $\Sigma = \{0, 1\}$

1. Move the head to the right direction
2. Read symbols without change
3. When read the symbol # move to the left
4. Change the symbol into # and move to the left
5. Move the head to the right direction and read without change
6. When reach the first symbol in the left move to the right and stop.

#	0	0	1	1	1	#	#	#	#



0 0 1 1 1 # # #
1 → 1, R
0 → 0, R
1 → 1, R
0 → 0, R
#→ #, L
q0
q1
q2

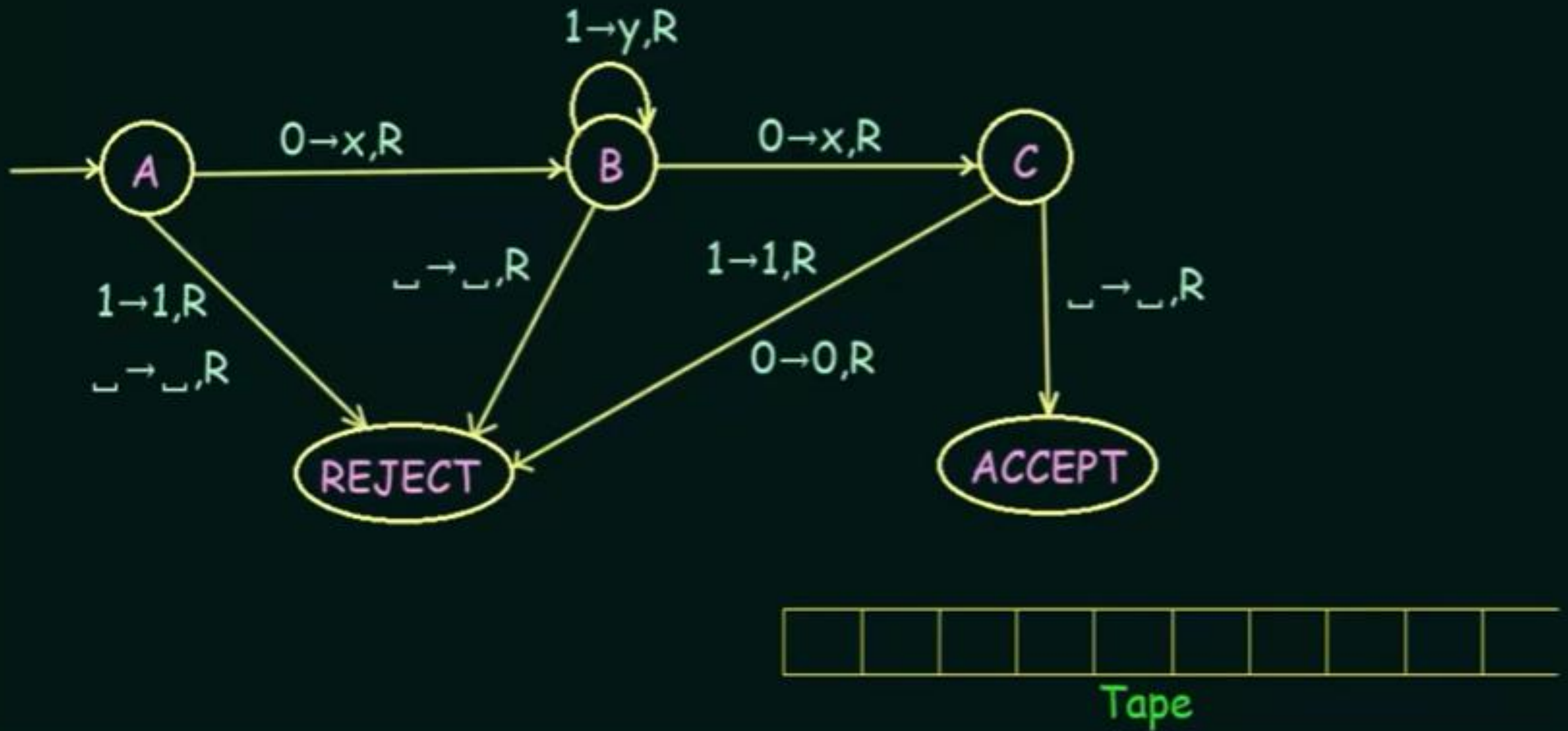


0 0 1 1 # # # #
1 → 1, R
0 → 0, R
1 → 1, L
0 → 0, L
1 → 1, R
0 → 0, R
#→ #, L
1 → #, L
0 → #, L
#→ #, R
q0
q1
q2
q3
q4

Turing Machine - Example (Part-1)

Design a Turing Machine which recognizes the language

$L = 01^*0$



DEFINITION

Call a language *Turing-recognizable* if some Turing machine recognizes it.[1]

- It is called a ***recursively enumerable language*** in some other textbooks.

DEFINITION

Call a language *Turing-decidable* or simply *decidable* if some Turing machine decides it.[2]

- It is called a ***recursive language*** in some other textbooks.

- Every decidable language is Turing-recognizable.

Turing Machine - Example (Part-2)

Design a Turing Machine which recognizes the language $L = 0^N 1^N$

0	0	0	0	1	1	1	1	␣	␣	...

Turing Machine - Example (Part-2)

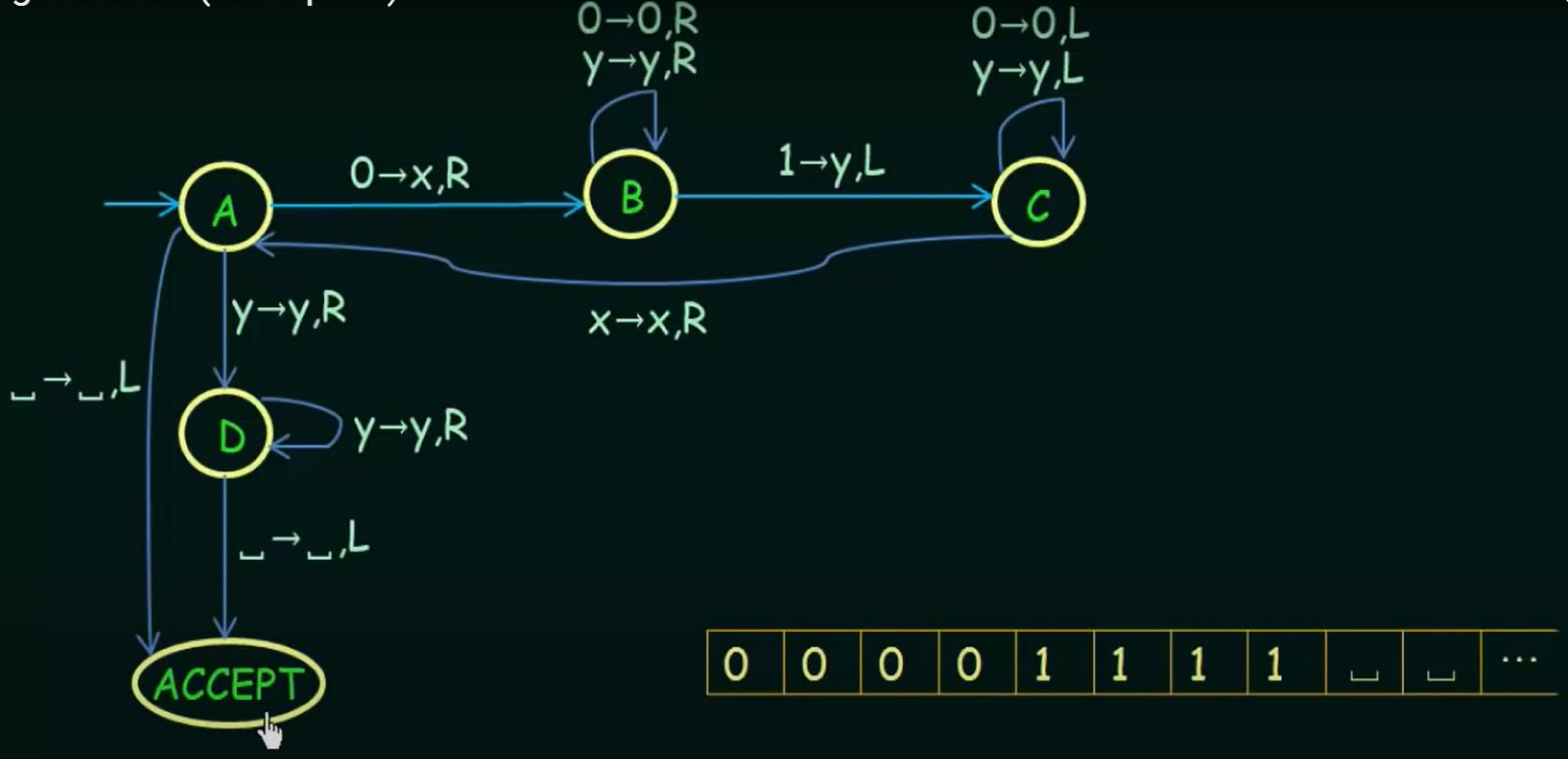
Design a Turing Machine which recognizes the language $L = 0^N 1^N$

0	0	0	0	1	1	1	1	␣	␣	...

Algorithm:

- Change "0" to "x"
- Move RIGHT to First "1"
  - If None: REJECT
- Change "1" to "y"
- Move LEFT to Leftmost "0"
- Repeat the above steps until no more "0"s
- Make sure no more "1"s remain

Turing Machine (Example 2)

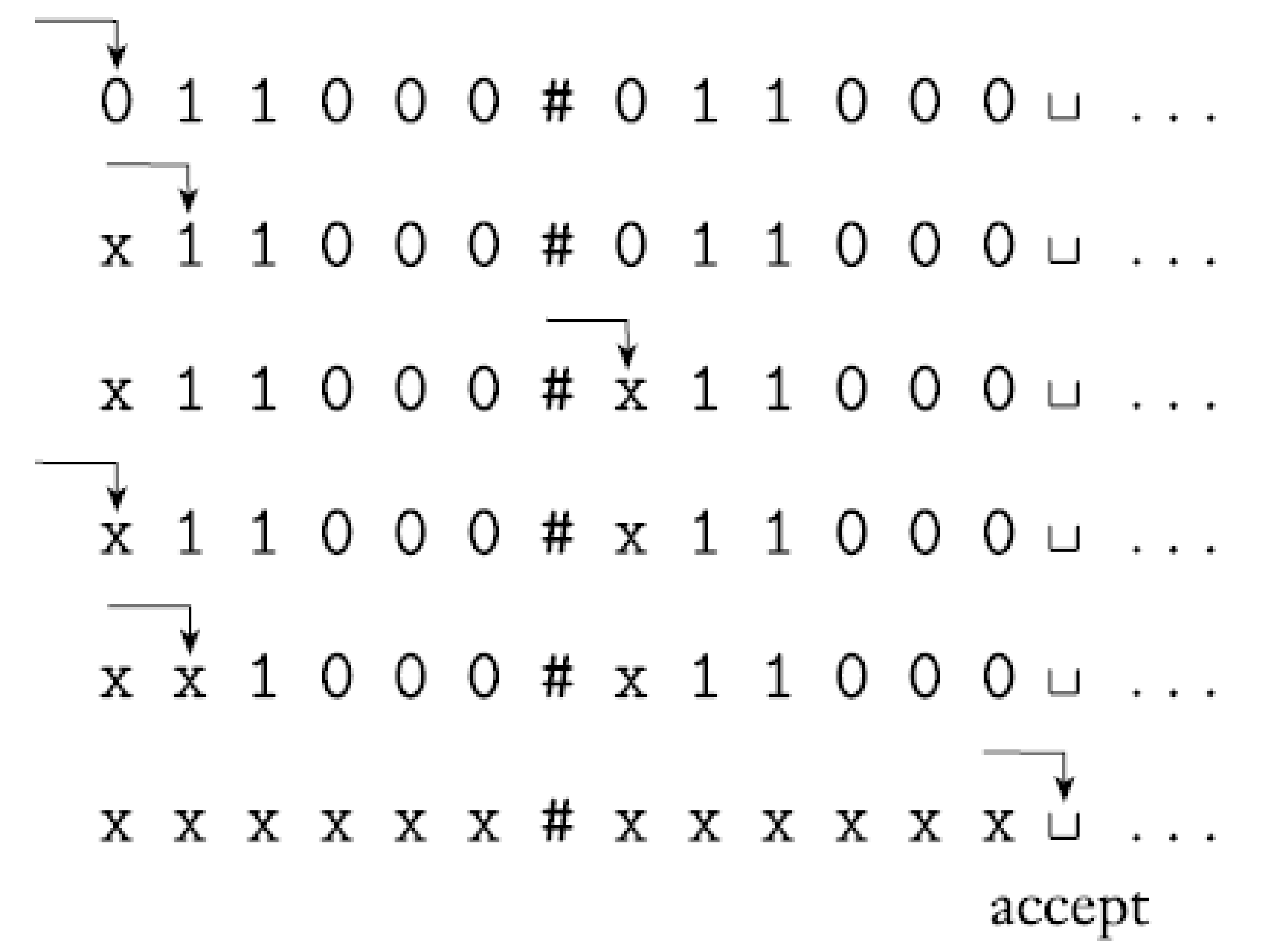


Example

- Let's introduce a Turing machine M1 for testing membership in the language $B = \{w \# w \mid w \in \{0,1\}^*\}$. We want M1 to accept if its input is a member of B and to reject otherwise.

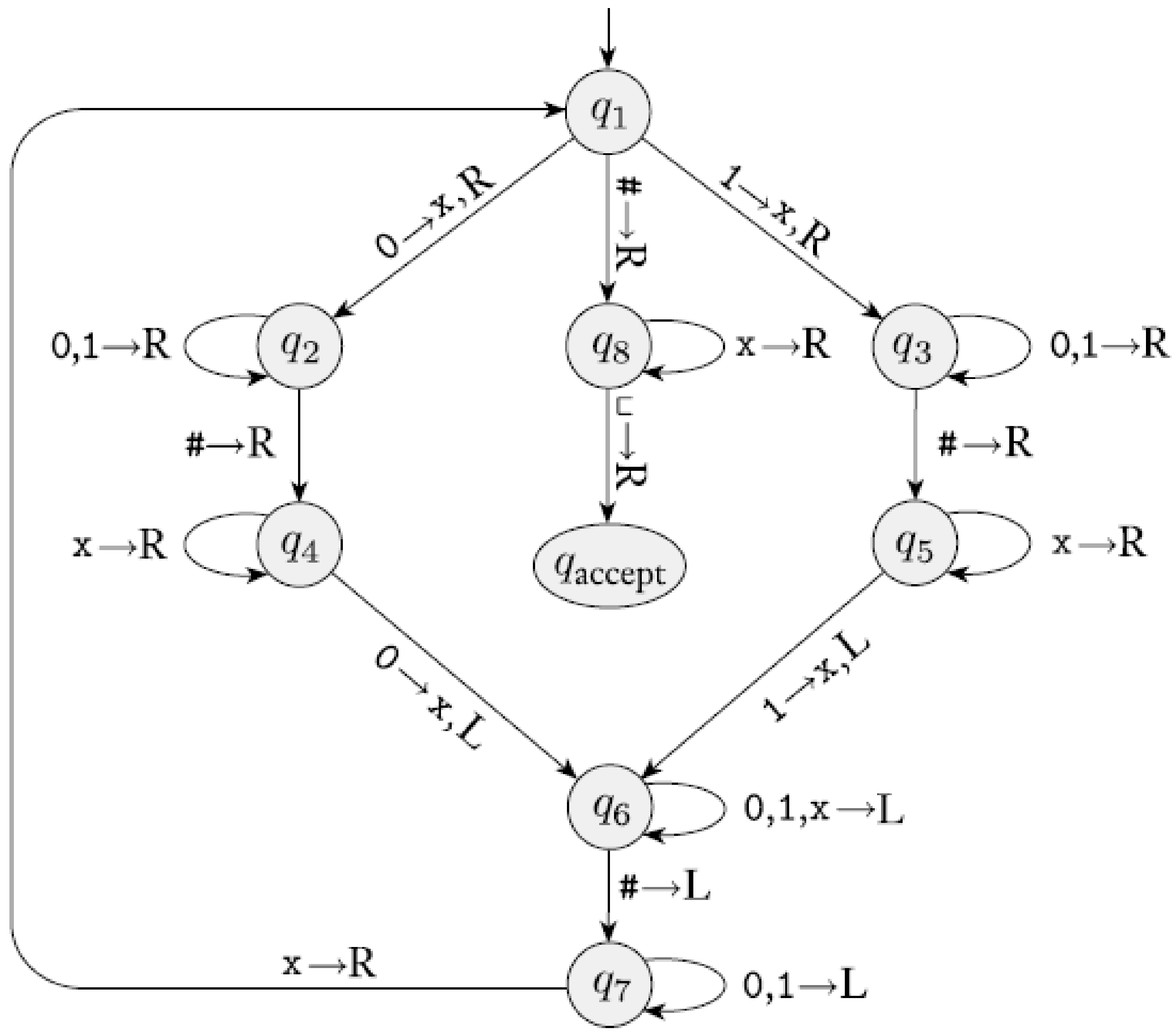
$M_1 =$ "On input string w:

1. Zig-zag across the tape to corresponding positions on either side of the # symbol to check whether these positions contain the same symbol. If they do not, or if no # is found, *reject*. Cross off symbols as they are checked to keep track of which symbols correspond.
2. When all symbols to the left of the # have been crossed off, check for any remaining symbols to the right of the #. If any symbols remain, *reject*; otherwise, *accept*."


0 1 1 0 0 0 # 0 1 1 0 0 0 ␣ ...
x 1 1 0 0 0 # 0 1 1 0 0 0 ␣ ...
x 1 1 0 0 0 # x 1 1 0 0 0 ␣ ...
x 1 1 0 0 0 # x 1 1 0 0 0 ␣ ...
x x 1 0 0 0 # x 1 1 0 0 0 ␣ ...
x x x x x x # x x x x x x ␣ ...
accept

The following is a formal description of $M_1 = (Q, \Sigma, \Gamma, \delta, q_1, q_{\text{accept}}, q_{\text{reject}})$, the Turing machine that we informally described ` for deciding the language $B = \{w\#w \mid w \in \{0,1\}^*\}$.

- $Q = \{q_1, \ldots, q_8, q_{\text{accept}}, q_{\text{reject}}\}$,
- $\Sigma = \{0,1,\#\}$, and $\Gamma = \{0,1,\#,x,\sqcup\}$.
- We describe δ with a state diagram (see the following figure).
- The start, accept, and reject states are q_1, q_{accept}, and q_{reject}, respectively.



q1
q2
q3
q4
q5
q6
q7
q8
qaccept
0→x,R
#→R
1→x,R
0,1→R
x→R
0,1→R
#→R
␣→R
#→R
x→R
x→R
0→x,L
1→x,L
0,1,x→L
#→L
x→R
0,1→L

Turing Machine Programming Techniques (Part-3)

COMPARING TWO STRINGS

A Turing Machine to decide $\{ w \# w \mid w \in \{a,b,c\}^* \}$

COMPARING TWO STRINGS

A Turing Machine to decide $\{ w \# w \mid w \in \{a,b,c\}^* \}$

Solution:

- Use a new symbol such as 'x'
- Replace each symbol into an x after it has been examined

Solution:

- Use a new symbol such as 'x'
- Replace each symbol into an x after it has been examined

a b b a c # a b b a c

$\Downarrow$

x b b a c # x b b a c

$\Downarrow$

x x b a c # x x b a c

$\Downarrow$

x x x a c # x x x a c

$\Downarrow$

x x x x c # x x x x c

$\Downarrow$

x x x x x # x x x x x

Solution:

- Use a new symbol such as 'x'
- Replace each symbol into an x after it has been examined

a b b a c # a b b a c

⇓

x b b a c # x b b a c

⇓

x x b a c # x x b a c

⇓

x x x a c # x x x a c

⇓

x x x x c # x x x x c

⇓

x x x x x # x x x x x

Problem:

Can we do it non-destructively? i.e. without loosing the original strings?

Solution:

Replace each unique symbol with another unique symbol instead of replacing all with the same symbol

Problem:

Can we do it non-destructively? i.e. without loosing the original strings?

Solution:

Replace each unique symbol with another unique symbol instead of replacing all with the same symbol

Eg. $a \rightarrow p$

$b \rightarrow q$

$c \rightarrow r$

a b b a c # a b b a c

⇩

p q q p r # p q q p r

Problem:

Can we do it non-destructively? i.e. without loosing the original strings?

Solution:

Replace each unique symbol with another unique symbol instead of replacing all with the same symbol

Eg. $a \rightarrow p$
$b \rightarrow q$
$c \rightarrow r$

a b b a c # a b b a c

p q q p r # p q q p r

Restore the original strings if required