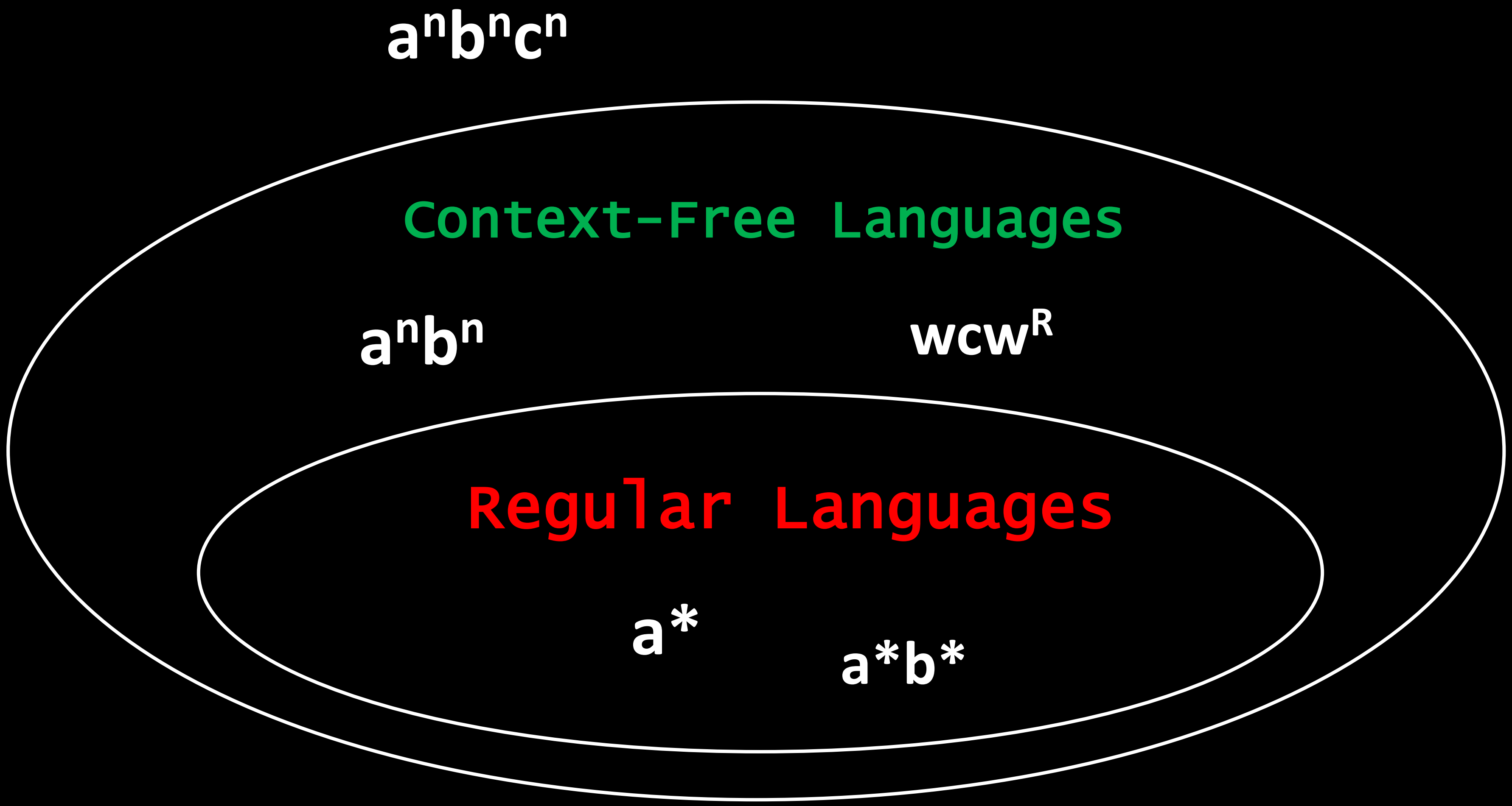
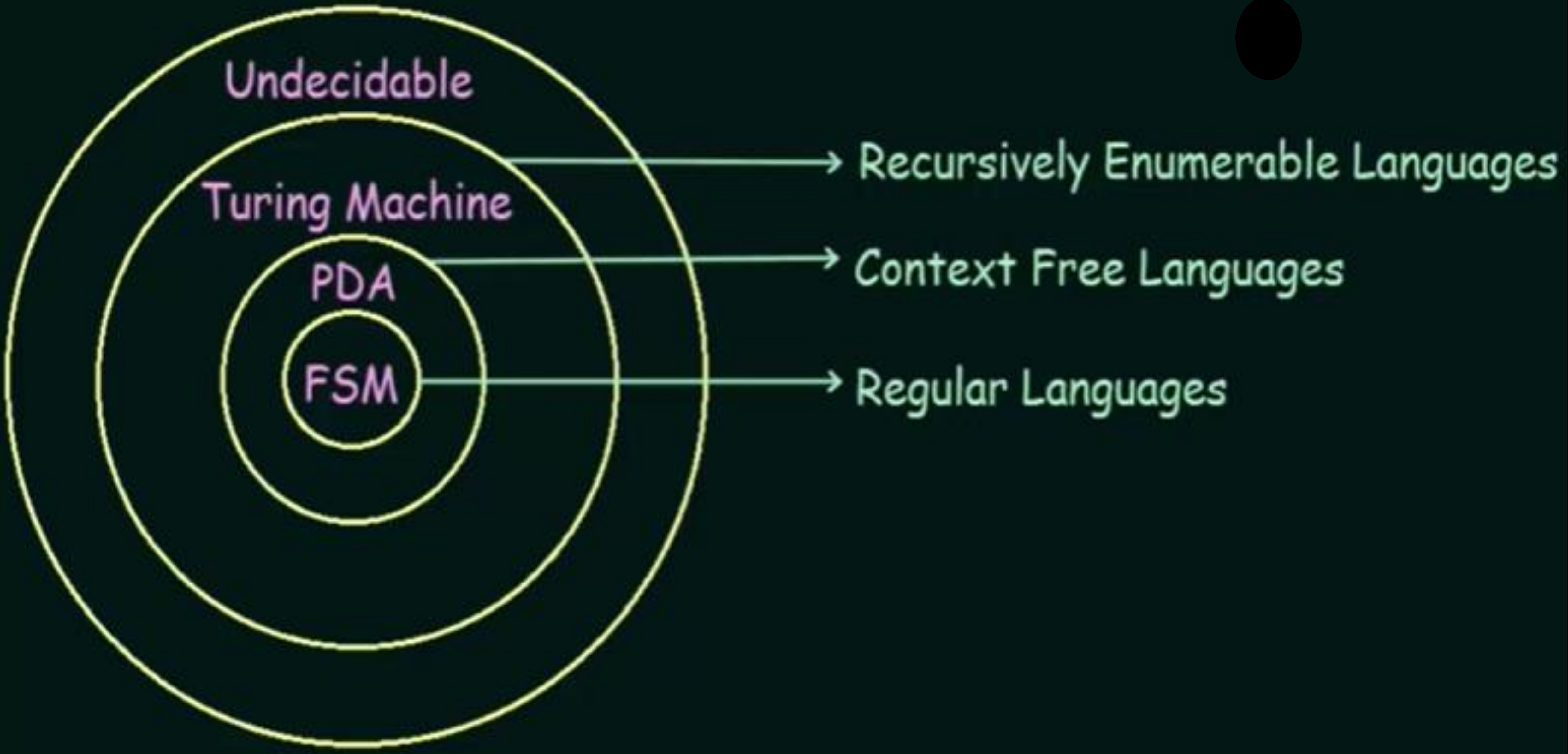

بالعِلم نَرتَقي

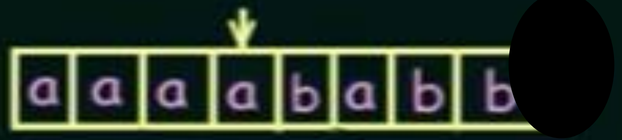
Turing machines

The language hierarchy

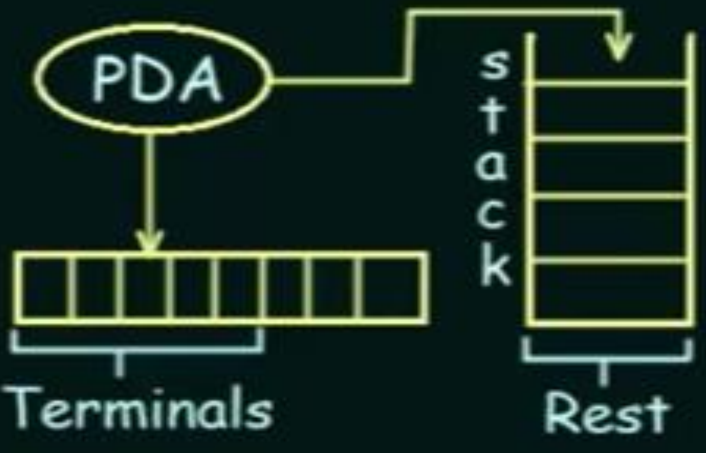



Undecidable
Turing Machine
PDA
FSM
Recursively Enumerable Languages
Context Free Languages
Regular Languages

FSM: The Input String

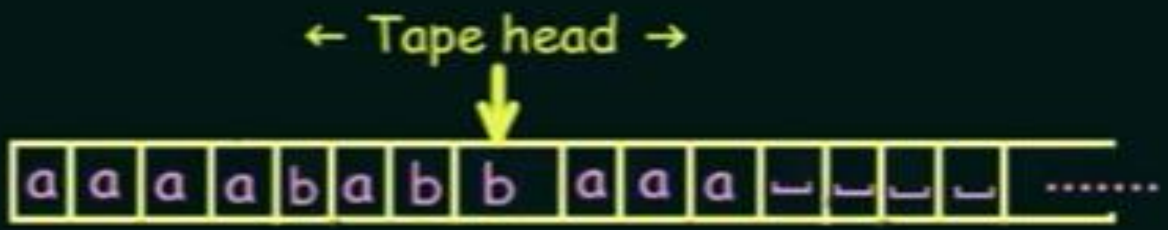


PDA:
-> The Input String
-> A Stack



TURING MACHINE:

-> A Tape



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

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

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

Multitape Turing Machine

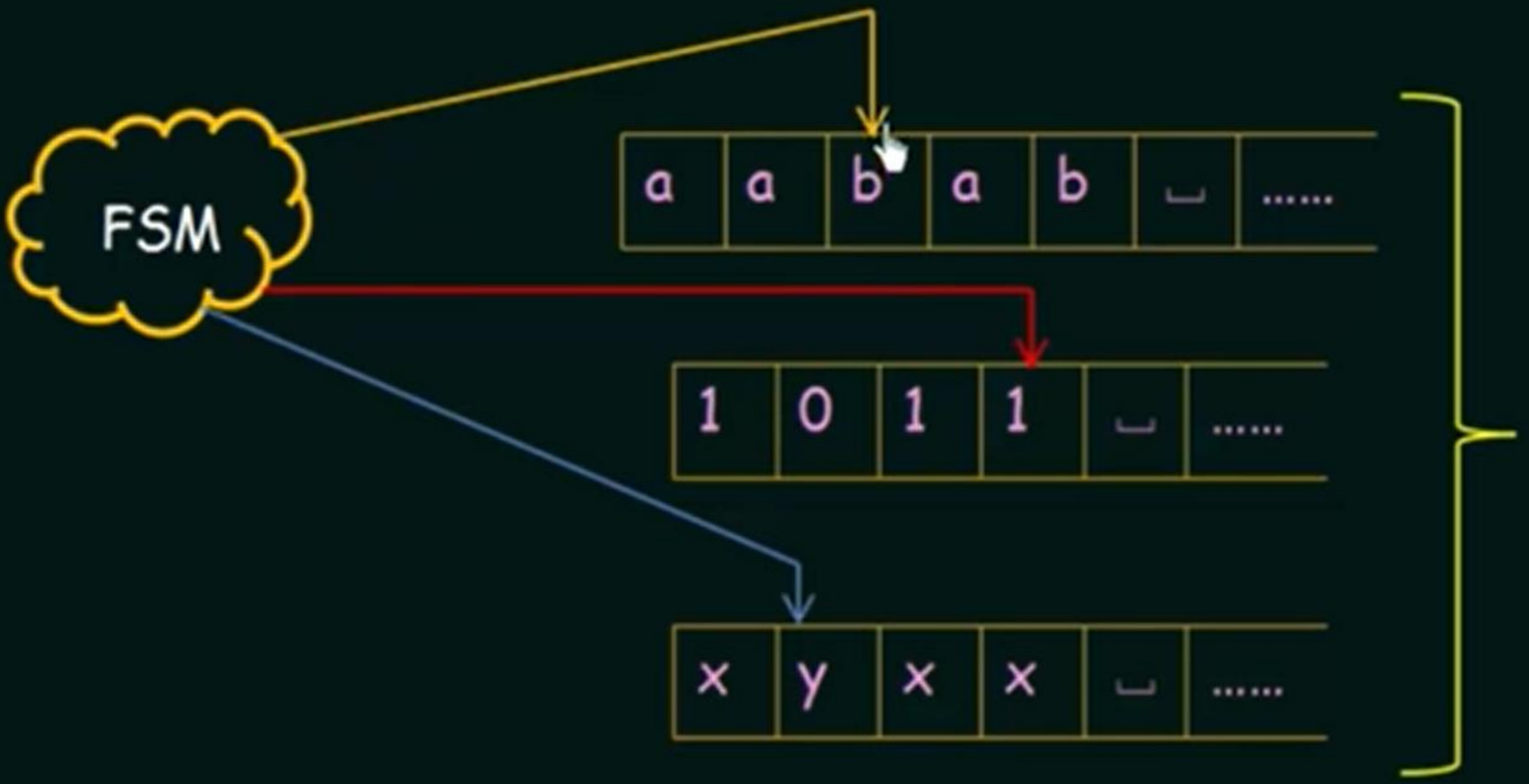
Theorem: Every Multitape Turing Machine has an equivalent Single Tape Turing Machine

Proof

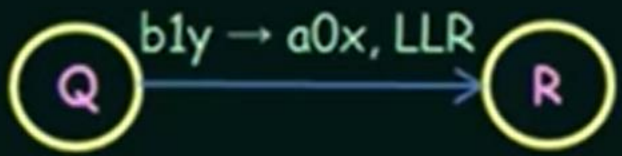
Given a Multitape Turing Machine show how to build a single tape Turing Machine

- Need to store all tapes on a single tape
  Show data representation
- Each tape has a tape head
  Show how to store that info
- Need to transform a move in the Multitape TM into one or moves in the Single Tape TM

Multitape Turing Machine

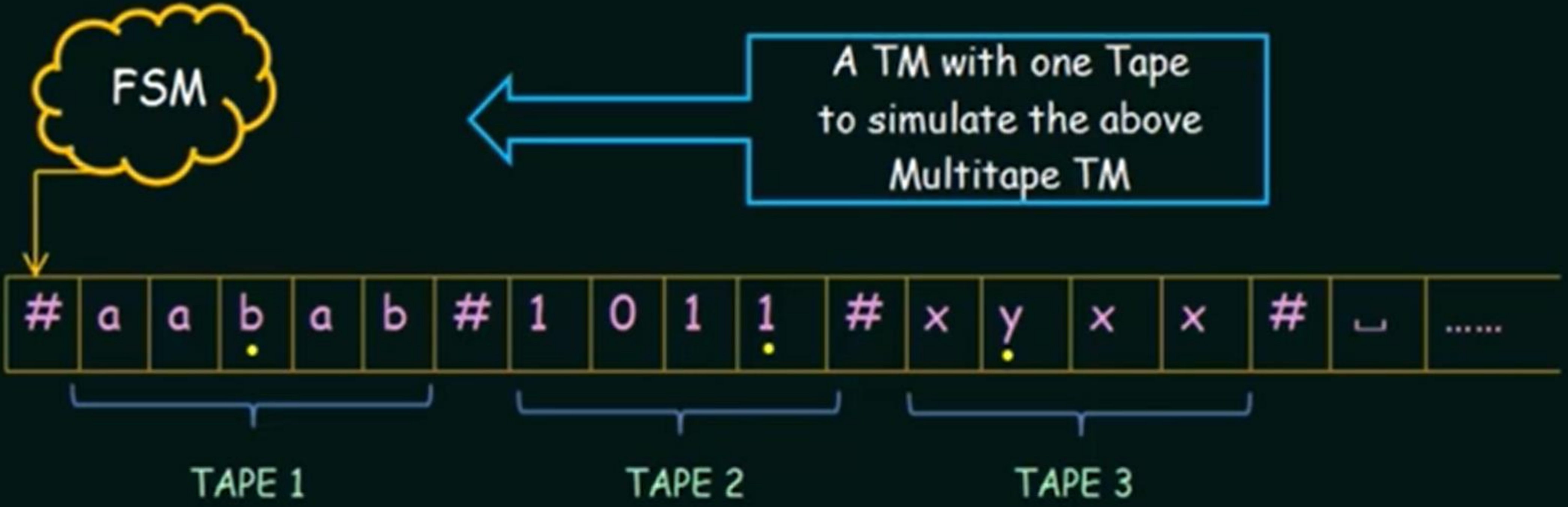


Several Tapes .
Each has its own
TAPE HEAD



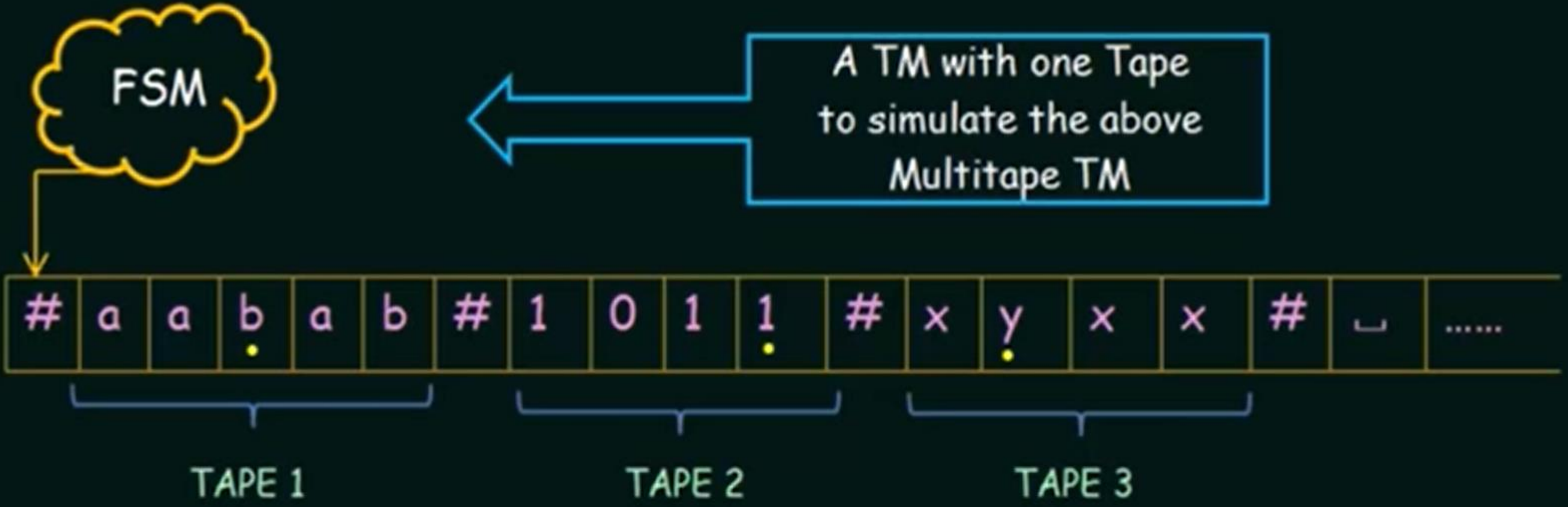
An Example Machine with
K=3 Tapes

Single Tape Turing Machine



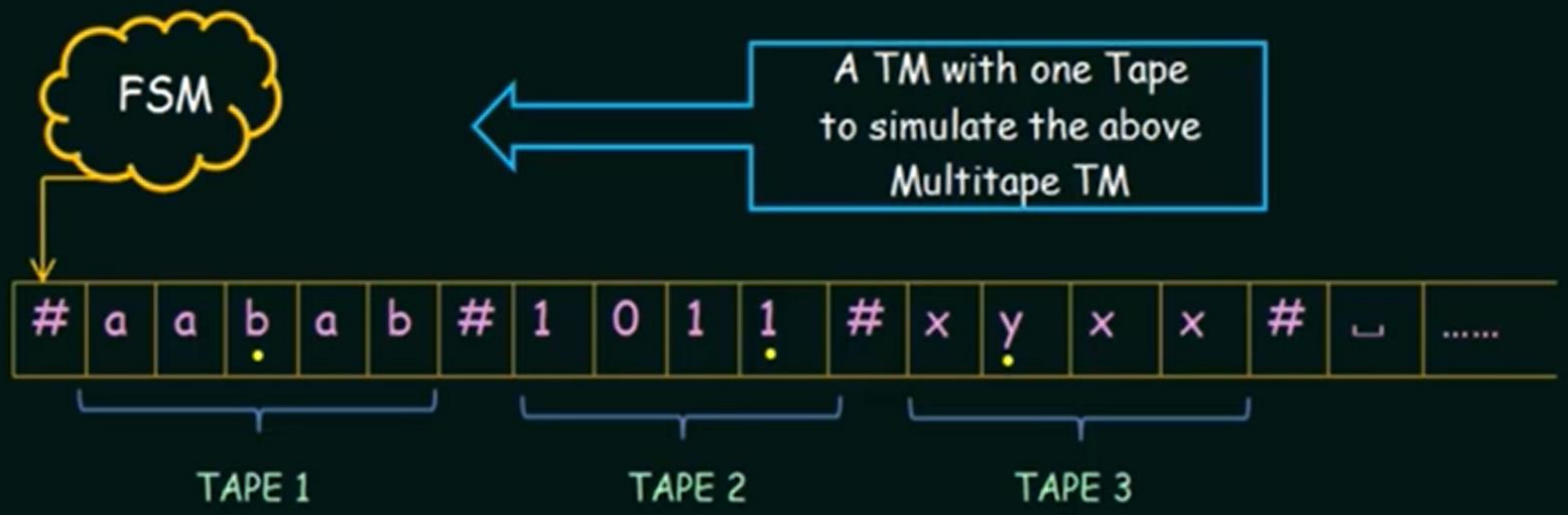
is

Single Tape Turing Machine



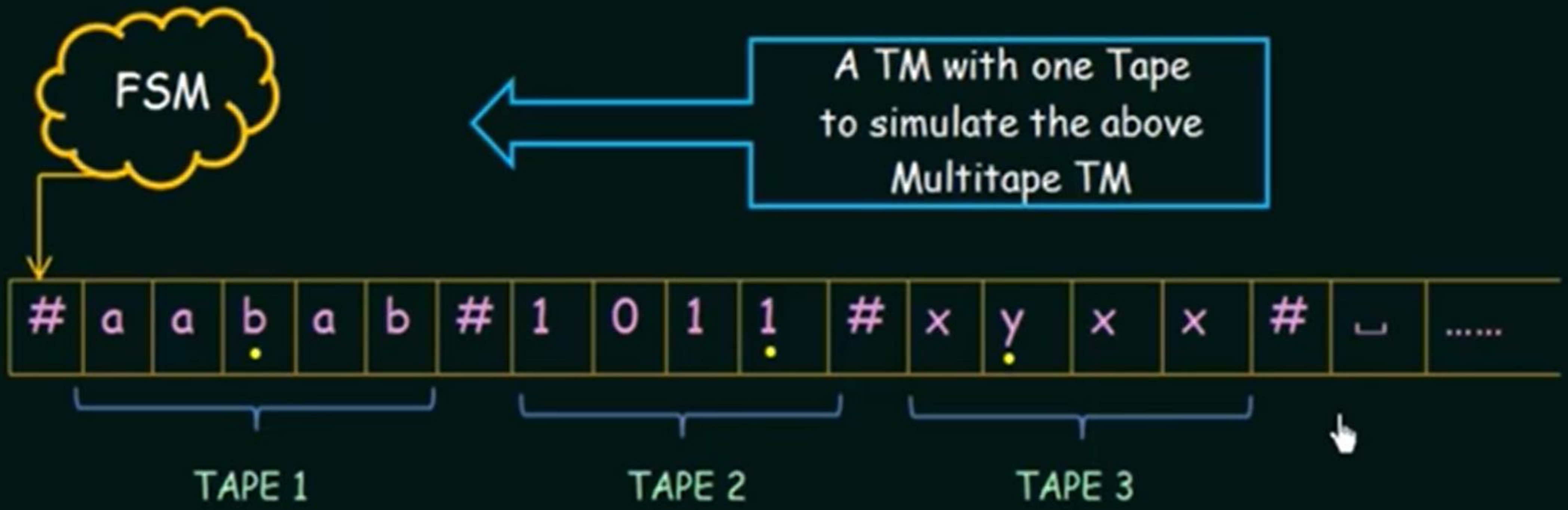
- Add "dots" to show where Head "K" is
- To simulate a transition from state Q, we must scan our Tape to see which symbols are under the K Tape Heads

Single Tape Turing Machine



- Add "dots" to show where Head "K" is
- To simulate a transition from state Q, we must scan our Tape to see which symbols are under the K Tape Heads
- Once we determine this and are ready to MAKE the transition, we must scan across the tape again to update the cells and move the dots

Single Tape Turing Machine



- Add "dots" to show where Head "K" is
- To simulate a transition from state Q, we must scan our Tape to see which symbols are under the K Tape Heads
- Once we determine this and are ready to MAKE the transition, we must scan across the tape again to update the cells and move the dots
- Whenever one head moves off the right end, we must shift our tape so we can insert a ␣

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

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

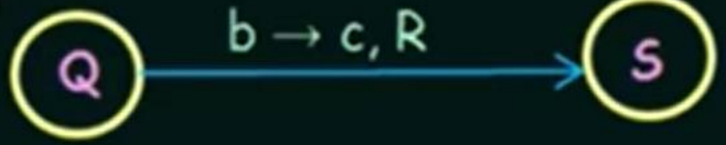
Nondeterminism in Turing Machine (Part-1)

Nondeterministic Turing Machines:

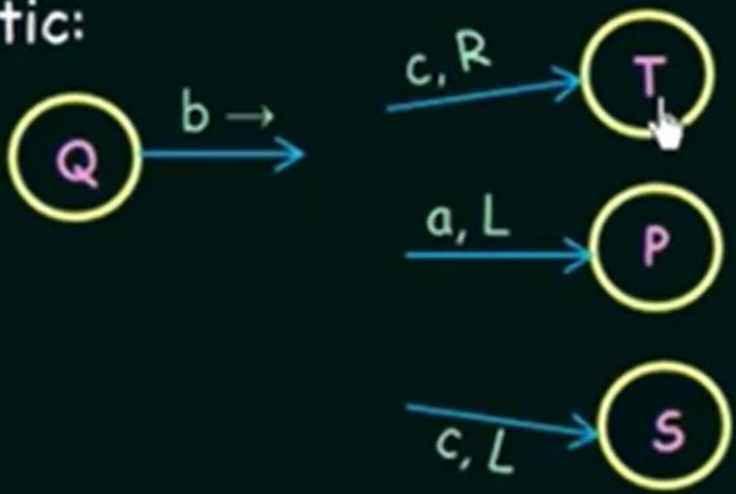
Transition Function:

$\delta : \quad Q \times \Sigma \rightarrow P\{\Gamma \times (R/L) \times Q\}$

Deterministic:

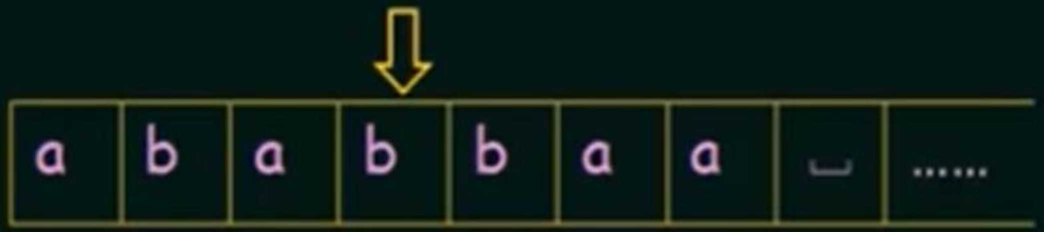


Nondeterministic:



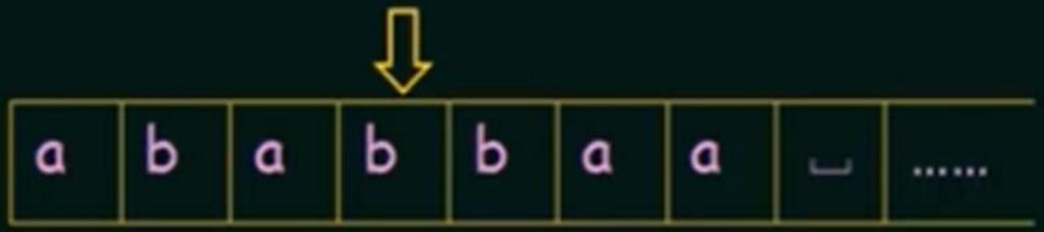
CONFIGURATION

- A way to represent the entire state of a TM at a moment during computation
- A string which captures:
  - The current state
  - The current position of the Head
  - The entire Tape contents



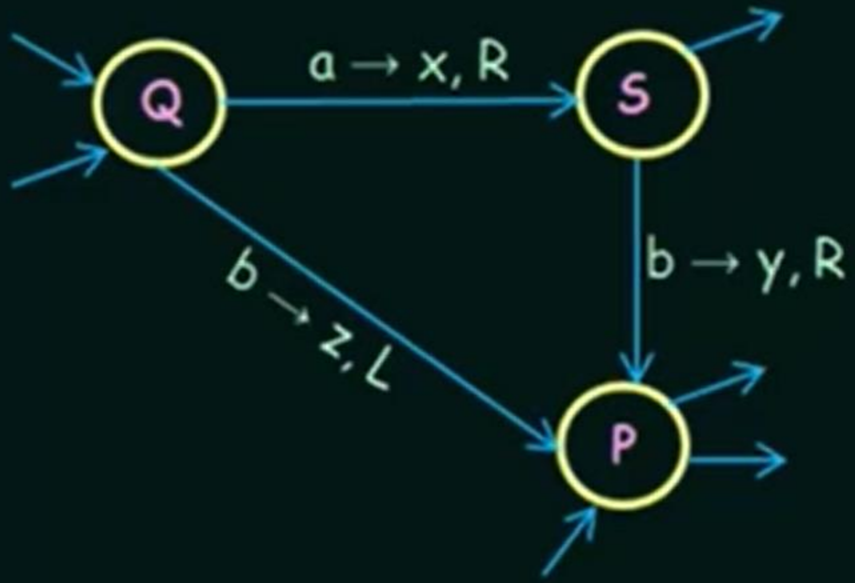
CONFIGURATION

- A way to represent the entire state of a TM at a moment during computation
- A string which captures:
  - The current state
  - The current position of the Head
  - The entire Tape contents

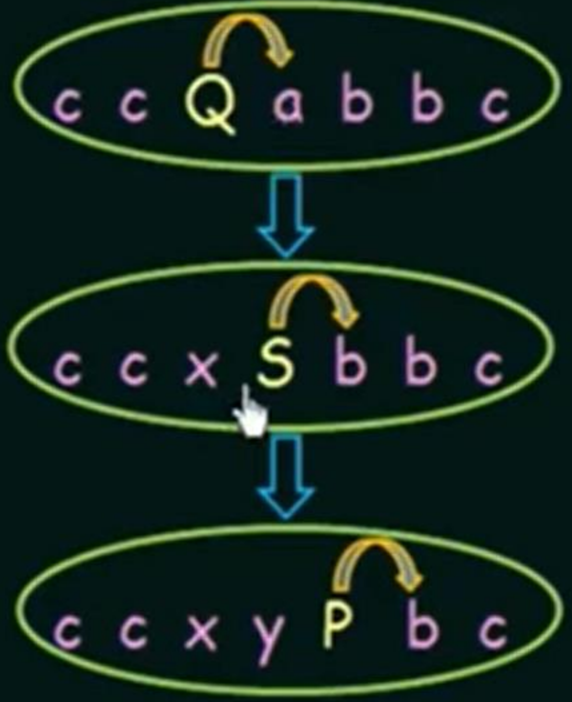


a b a Q b b a a

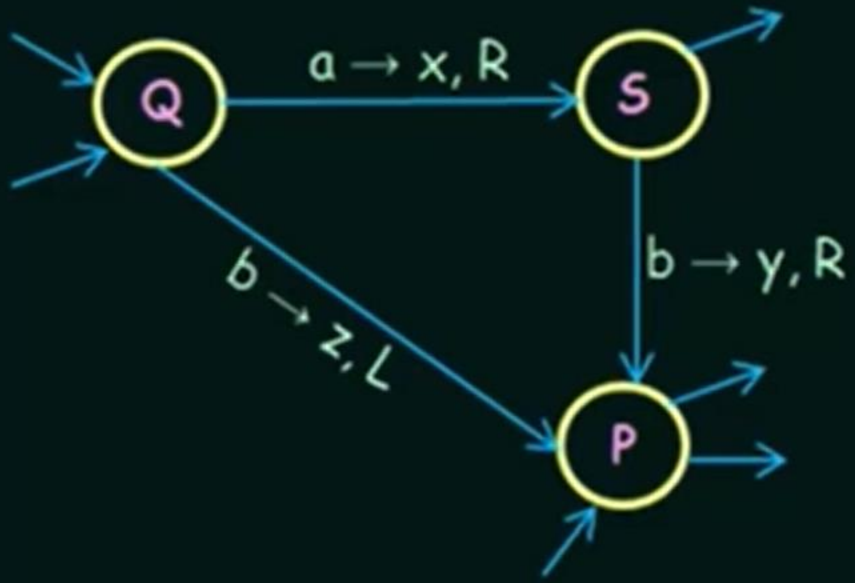
Deterministic TM:



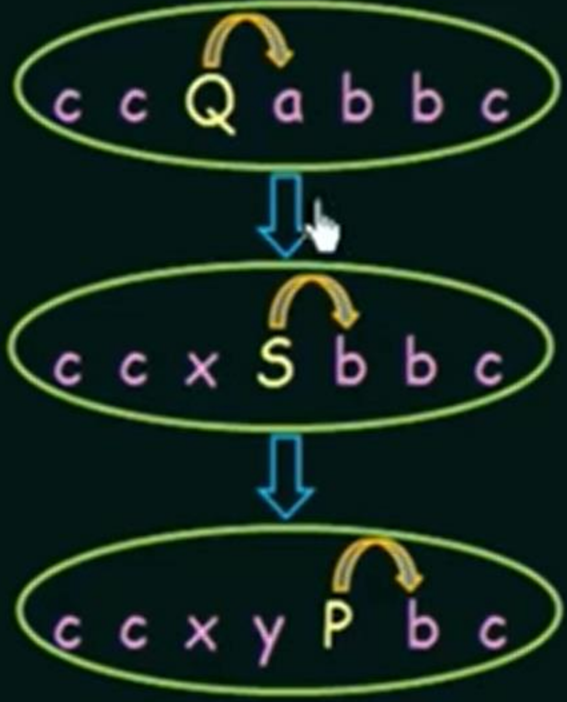
Computation History:



Deterministic TM:



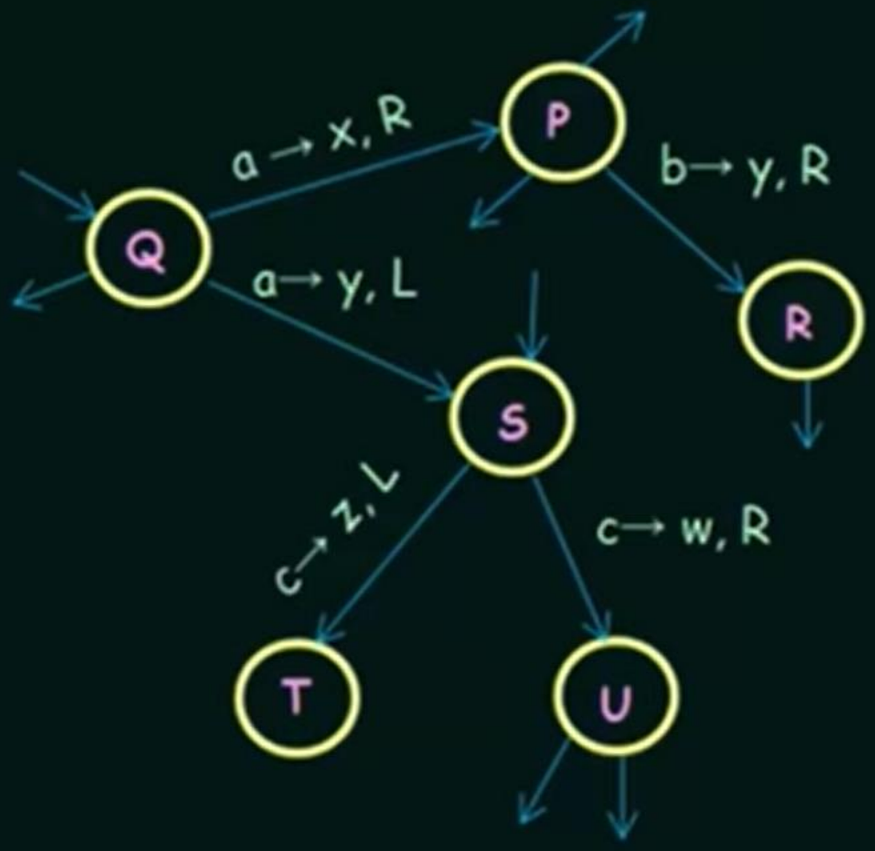
Computation History:



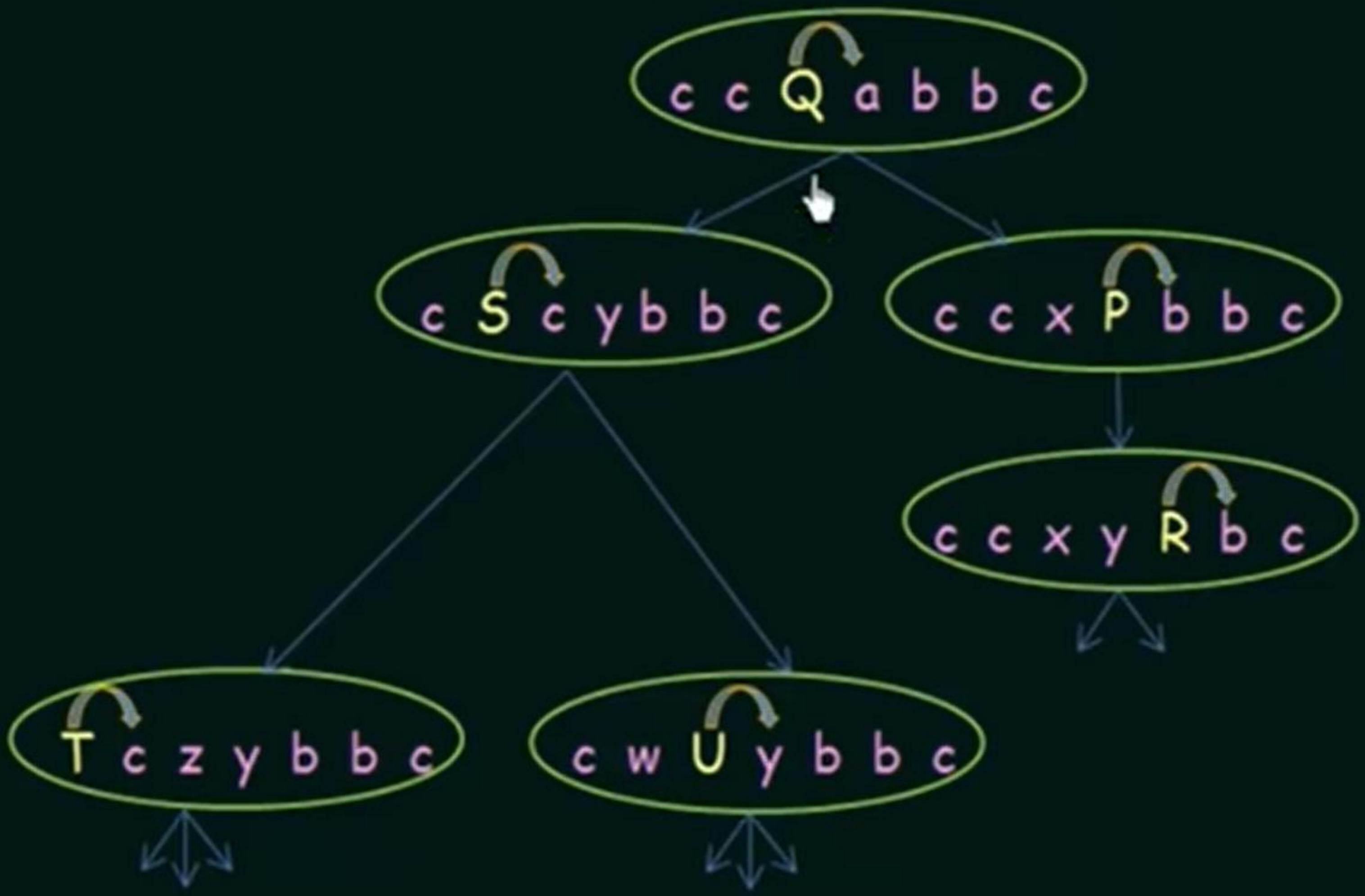
With Nondeterminism:

At each moment in the computation there can be more than one successor configuration

Nondeterministic TM:



Computation History:



Outcomes of a Nondeterministic Computation:

ACCEPT If any branch of the computation accepts, then the nondeterministic TM will Accept.

REJECT If all branches of the computation HALT and REJECT (i.e. no branches accept, but all computations HALT) then the Nondeterministic TM Rejects.

LOOP Computation continues but ACCEPT is never encountered. Some branches in the computation history are infinite.

Example: f(n) = 2n is computable

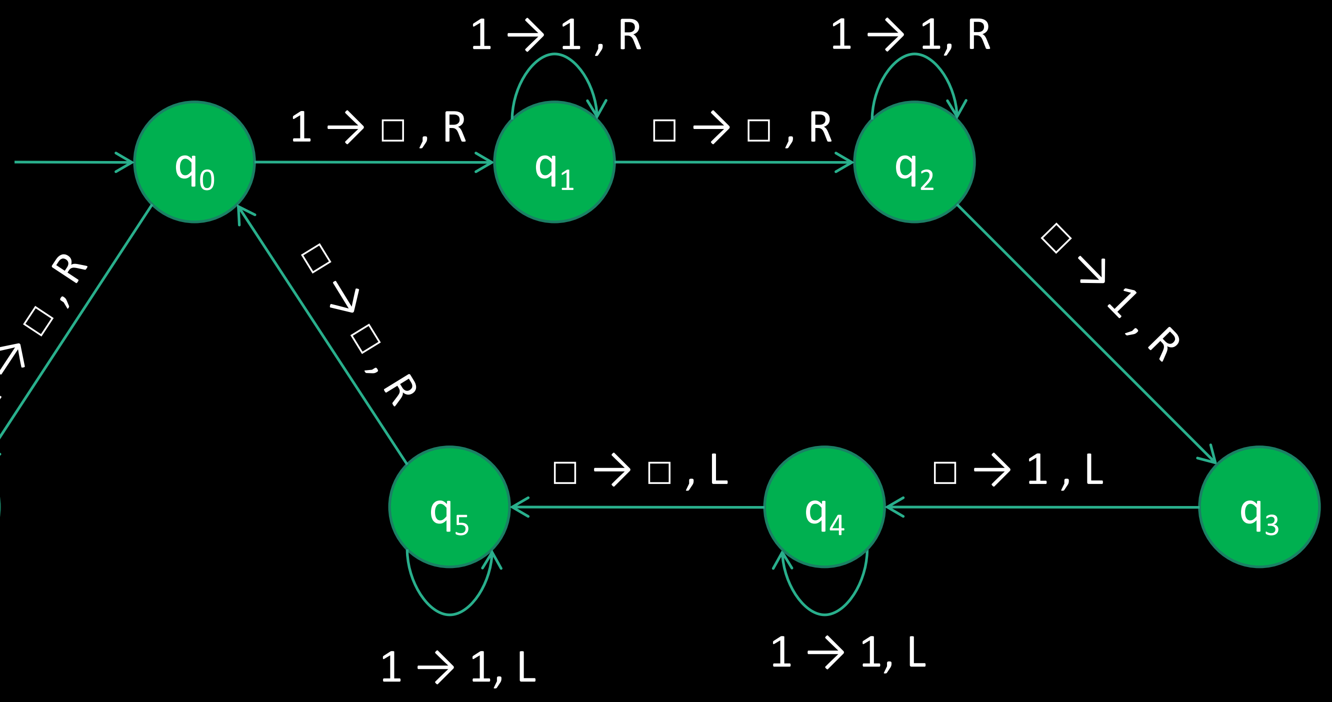
We design a TM that computes f(n).

High Level Program:

- The tape is divided into input and output (output is right after the first blank after the input)
- Repeat:
  - Erase one 1 from the input.
  - Pass along the rest of the input
  - Pass the blank that separates the input from the output.
  - Pass along the output until you reach the end (blank).
  - write two 1s.
  - Go to the beginning of the input.
- Until the input is erased (accept).

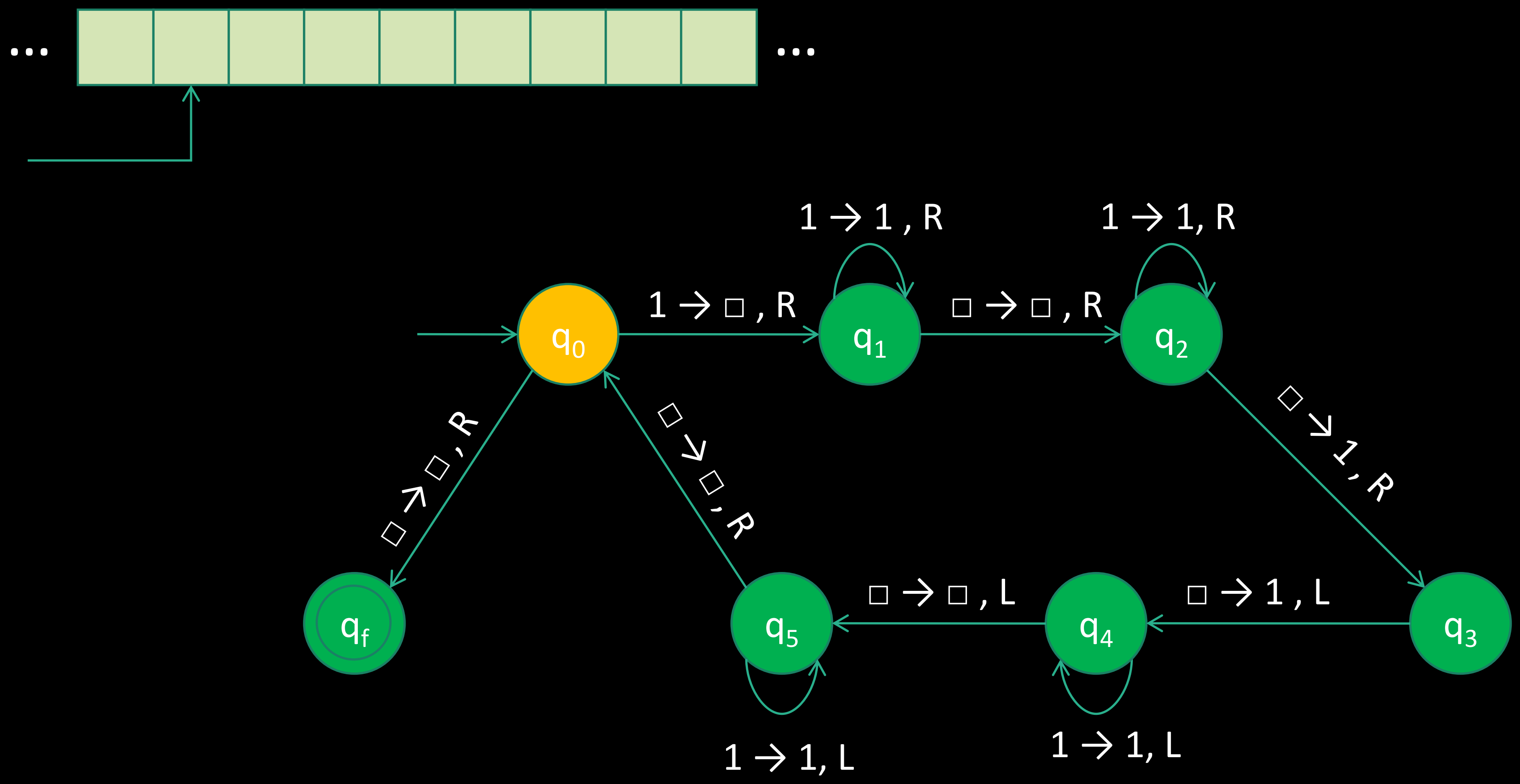
Example: f(n) = 2n is computable

The machine for f(n) = 2n



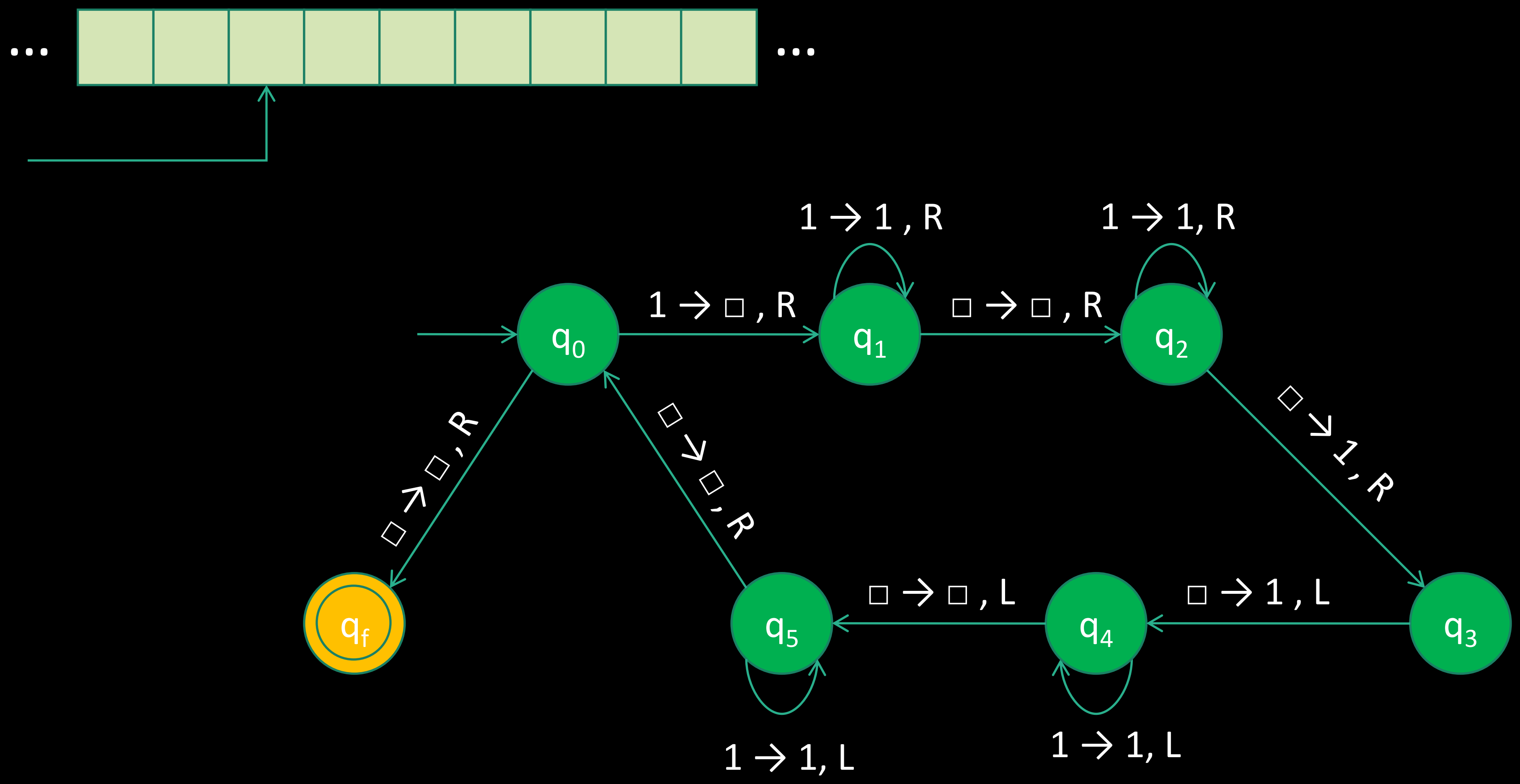
Example: f(n) = 2n is computable

- Test input: ε



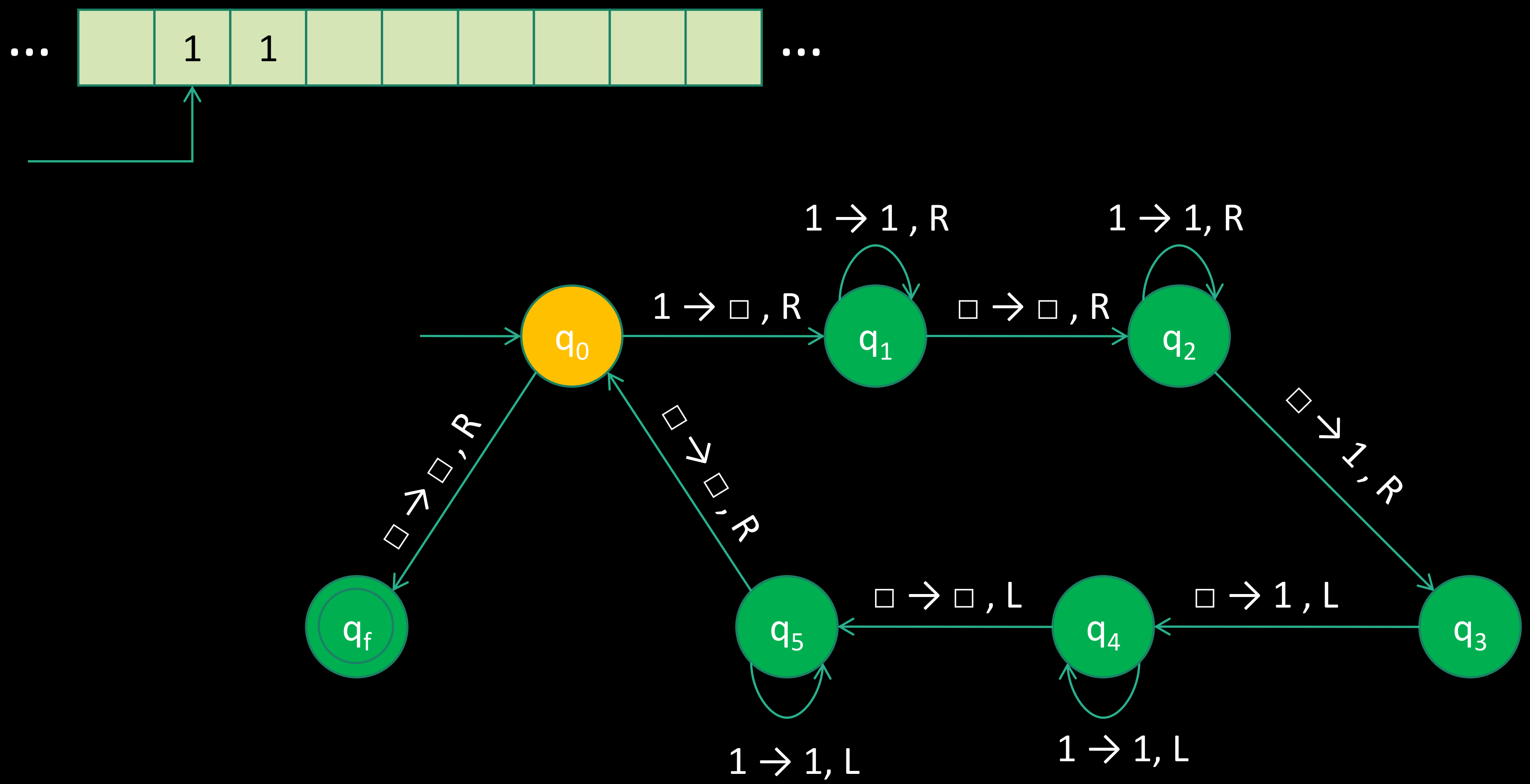
Example: f(n) = 2n is computable

- Test input: ε



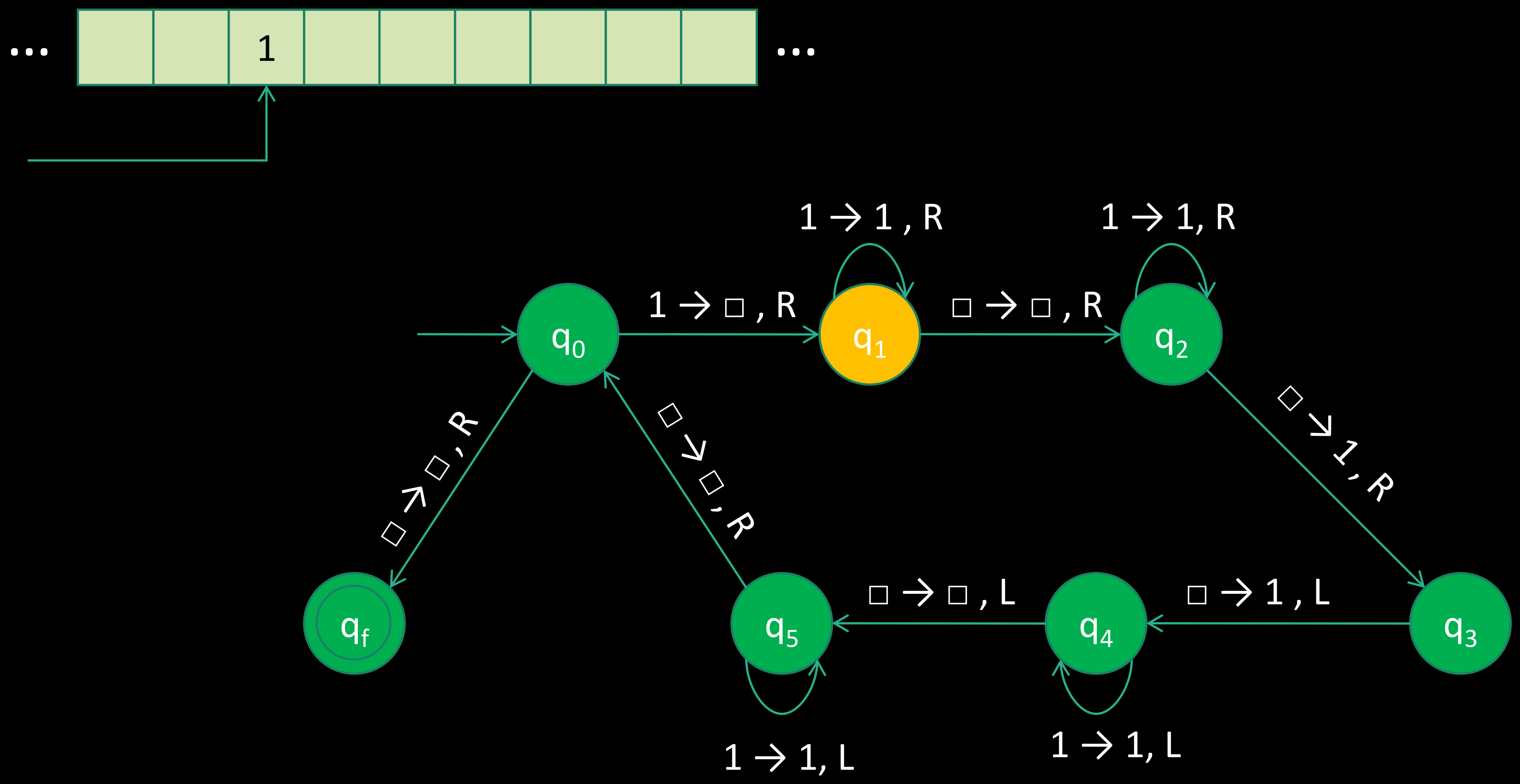
Example: $f(n) = 2n$ is computable

- Test input: 11



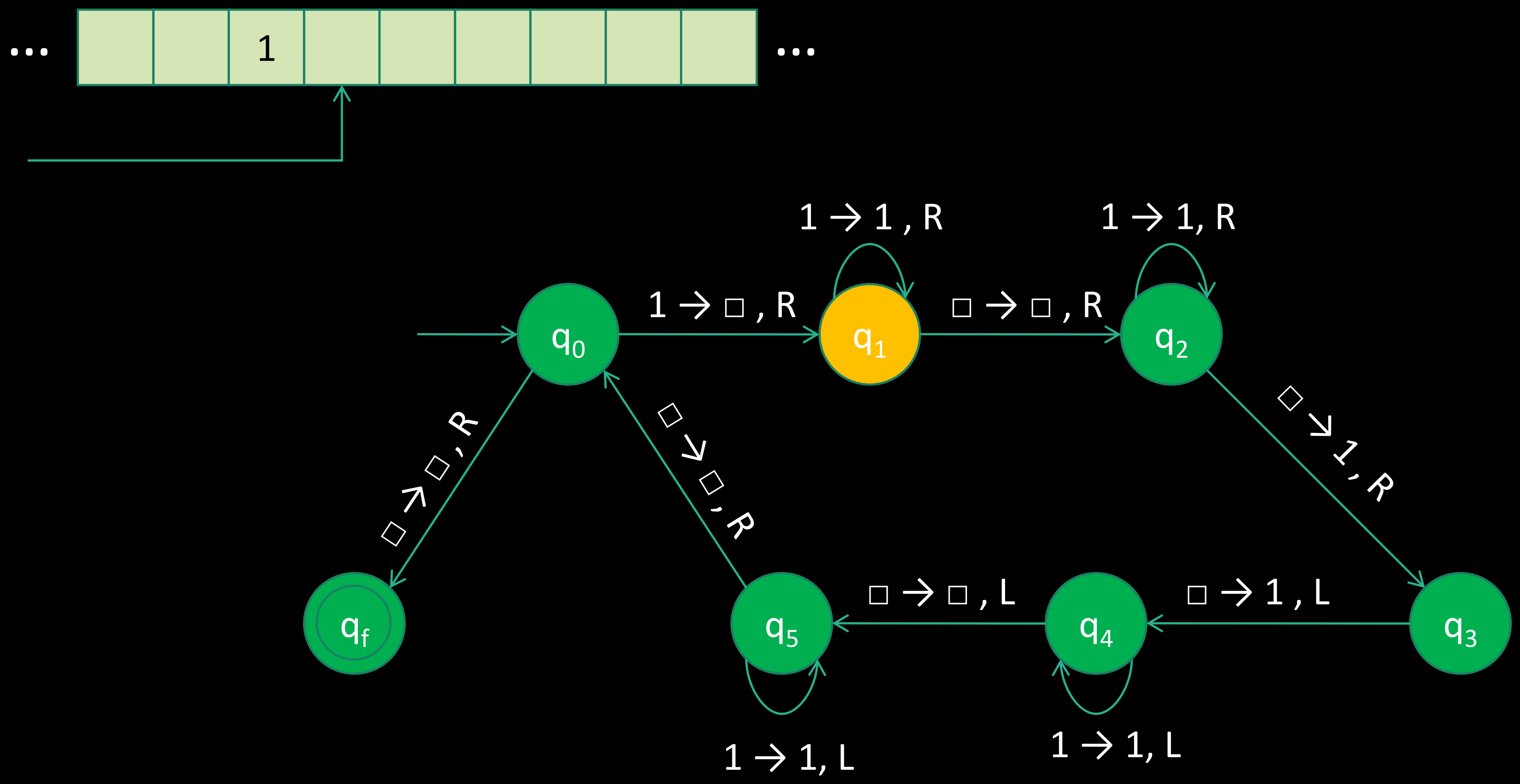
Example: f(n) = 2n is computable

- Test input: 11



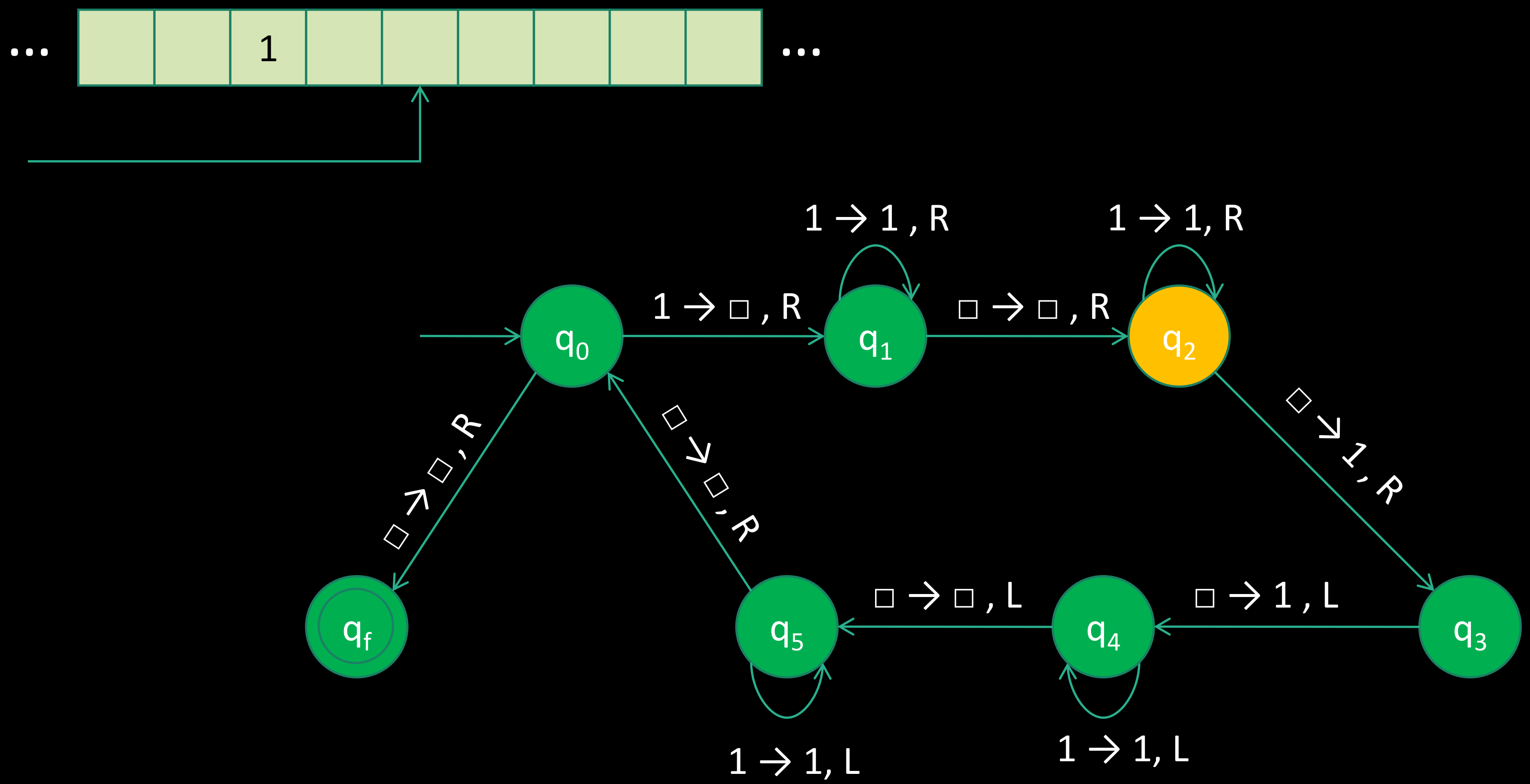
Example: $f(n) = 2n$ is computable

- Test input: 11



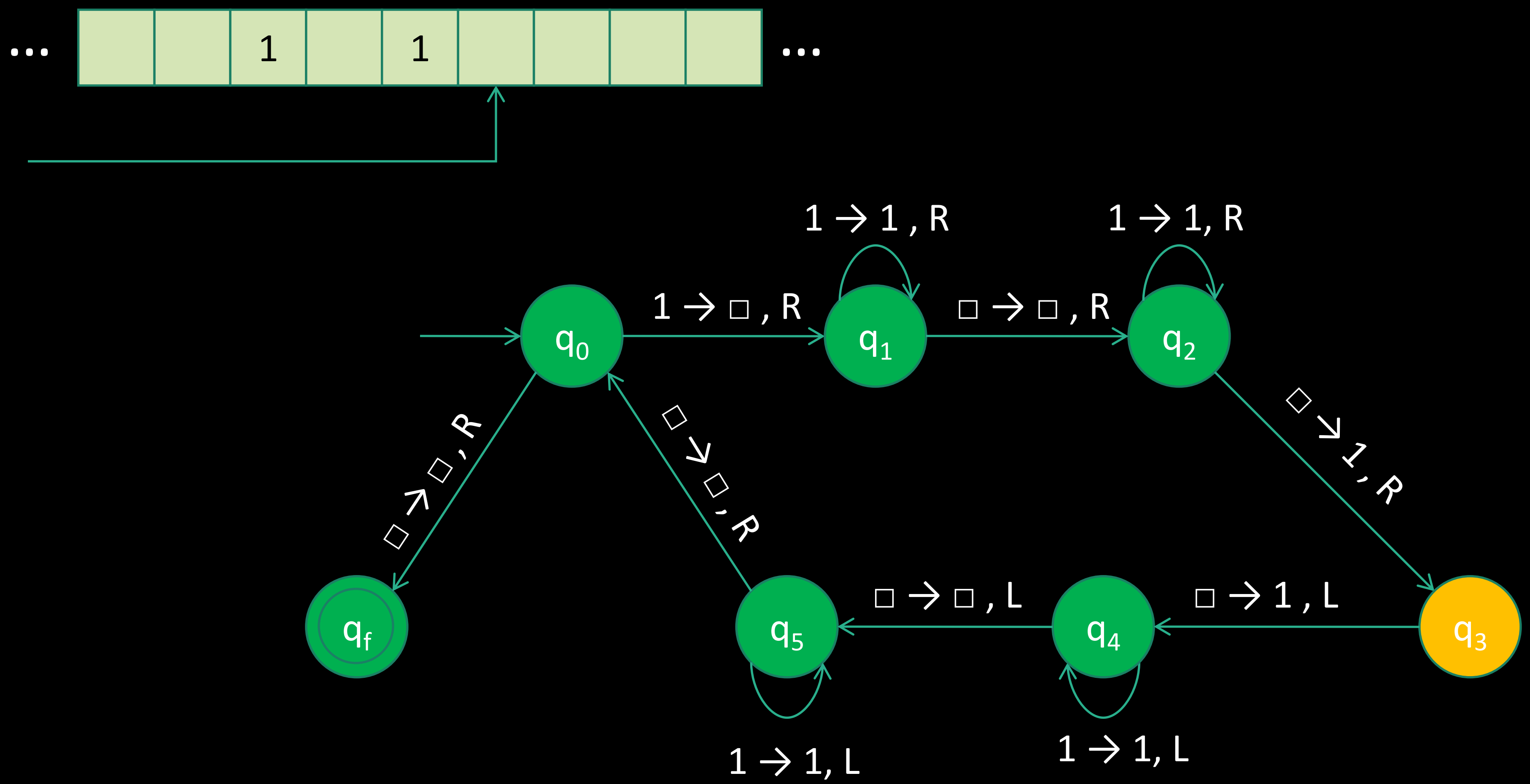
Example: $f(n) = 2n$ is computable

- Test input: 11



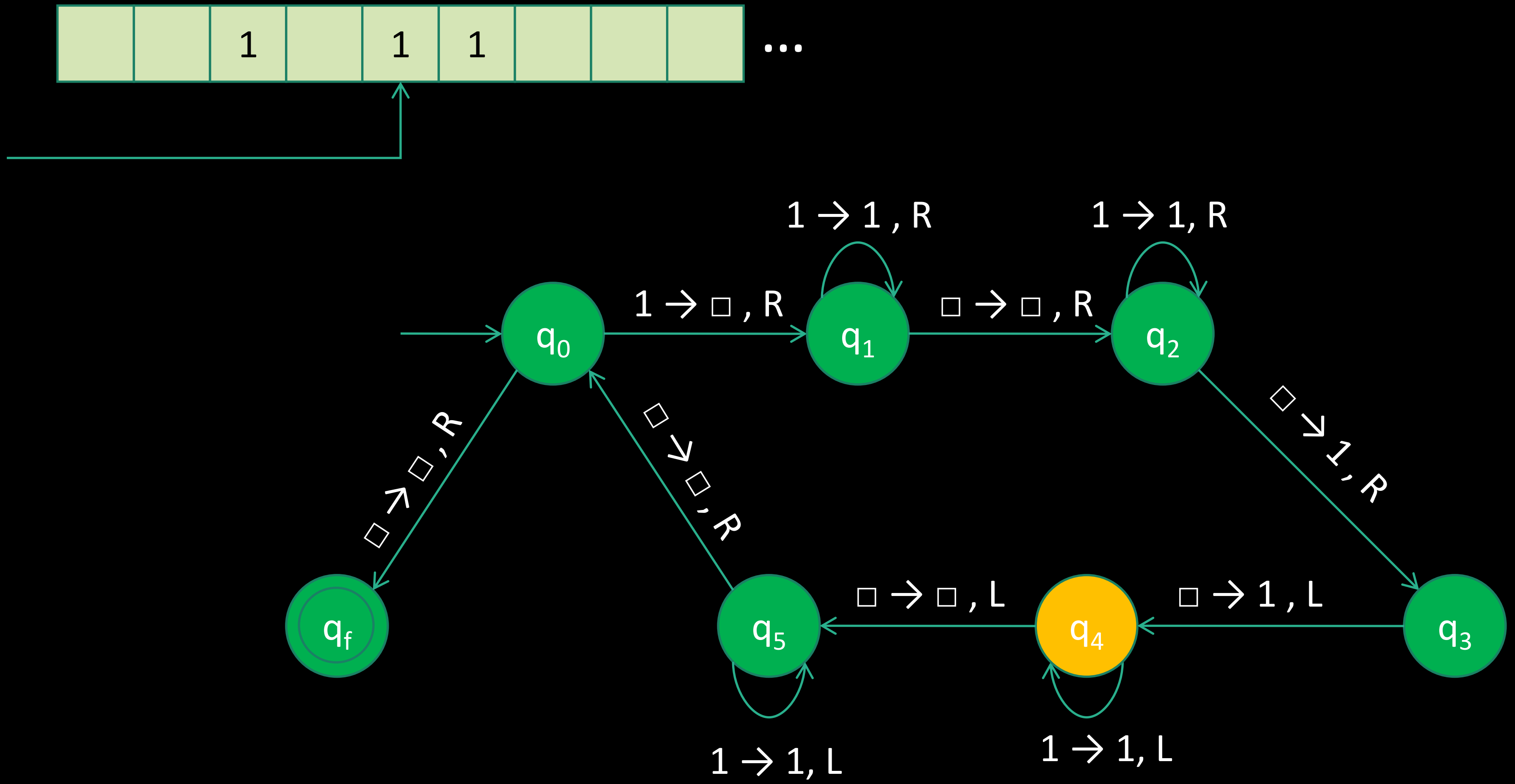
Example: $f(n) = 2n$ is computable

- Test input: 11



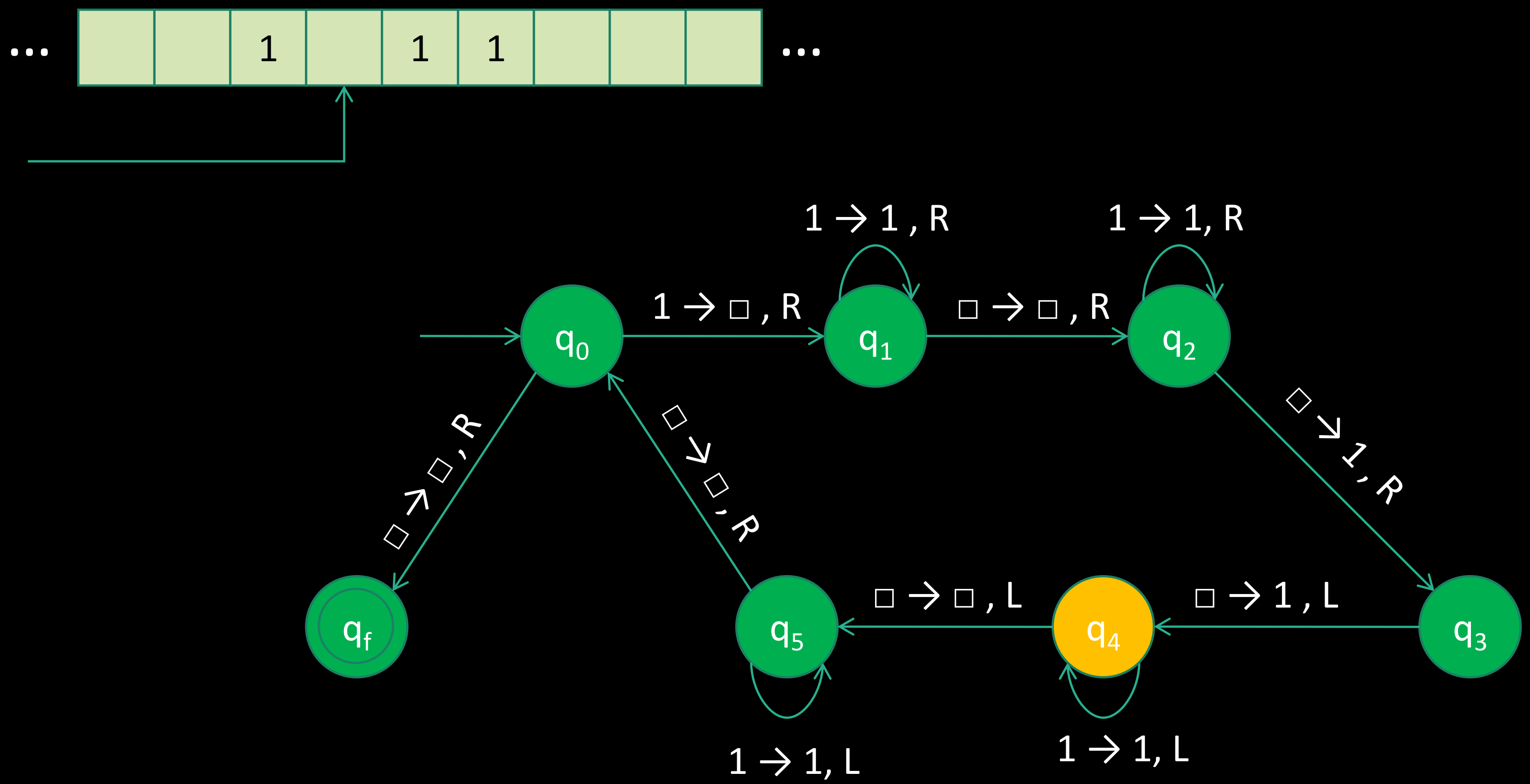
Example: $f(n) = 2n$ is computable

- Test input: 11



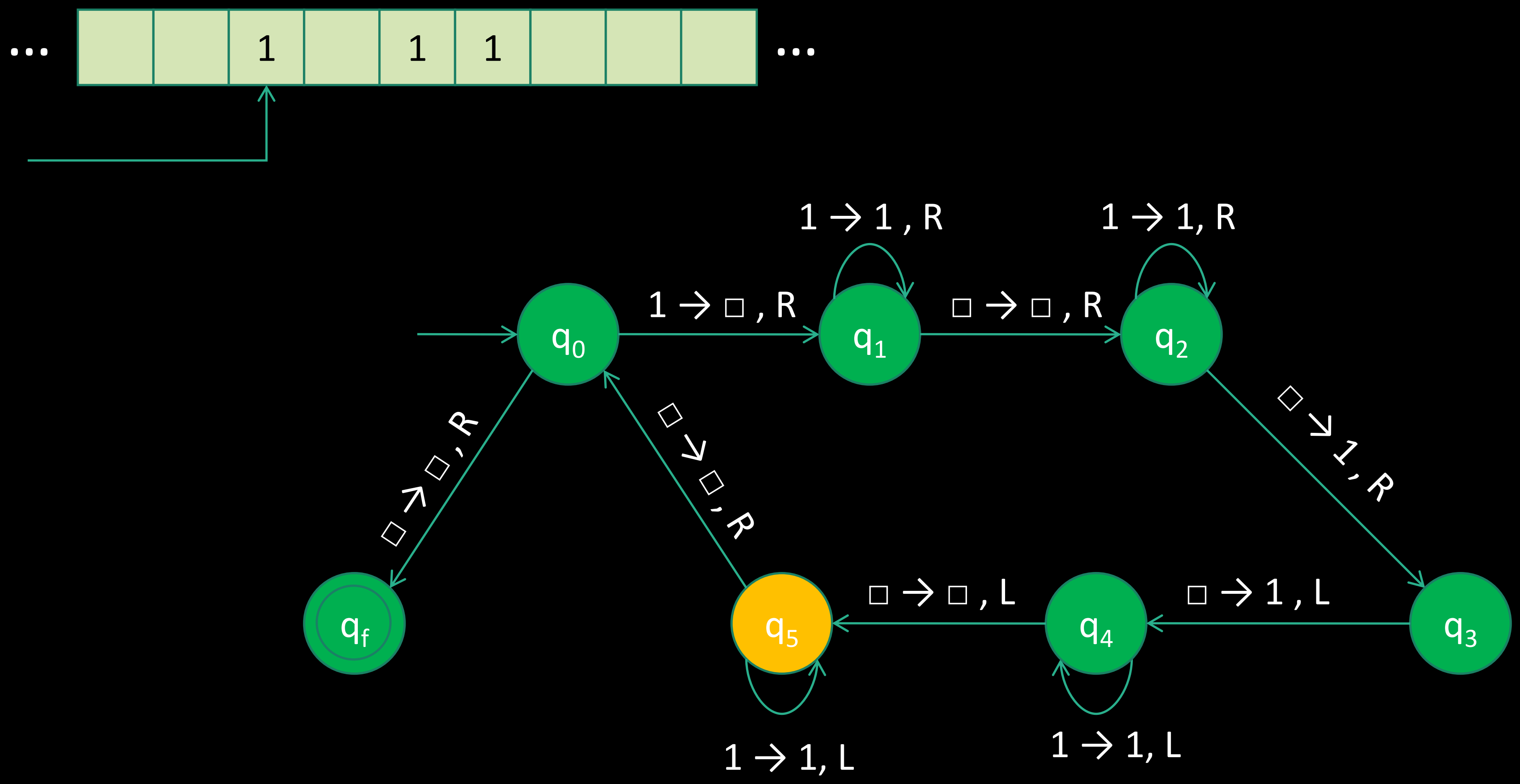
Example: f(n) = 2n is computable

- Test input: 11



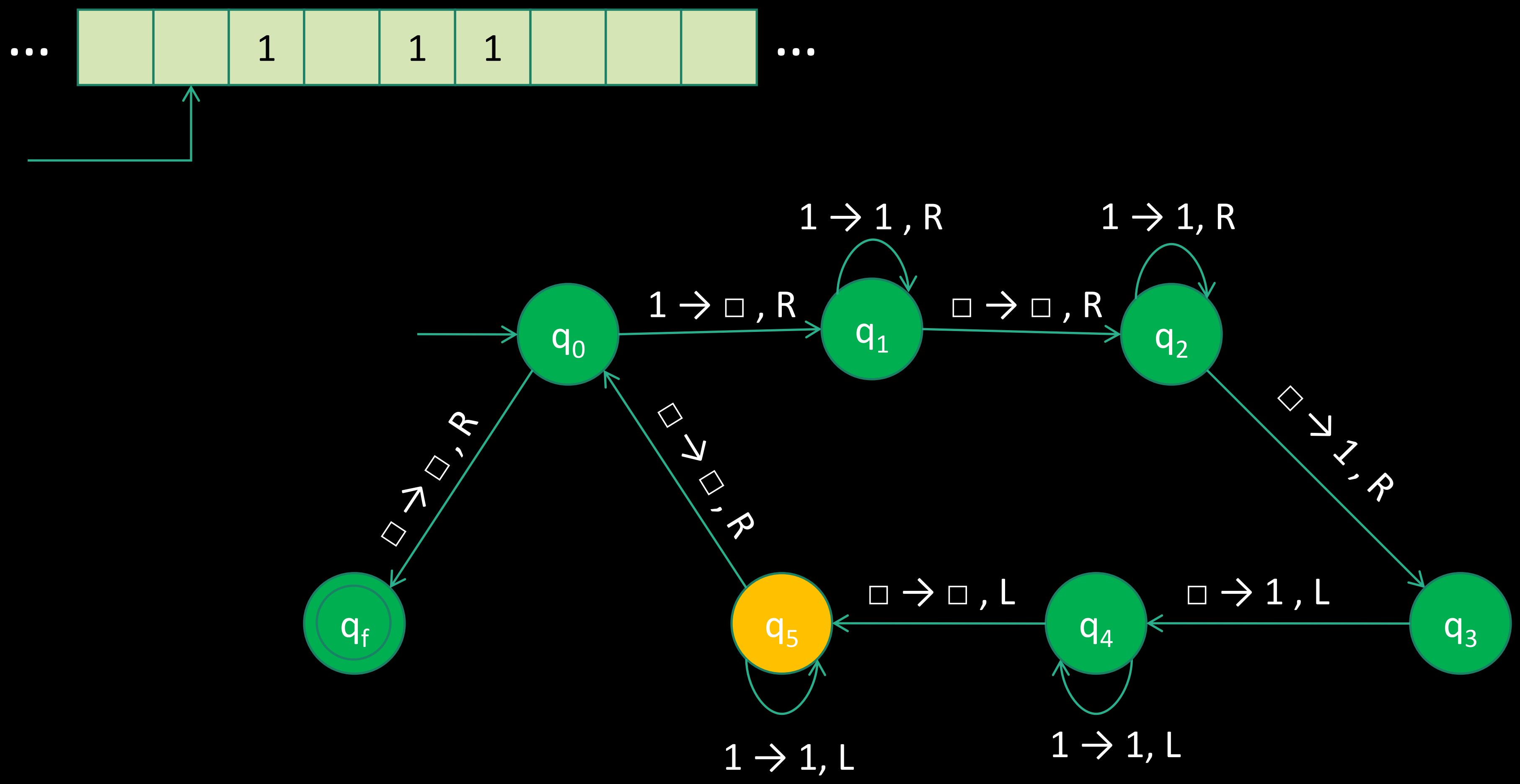
Example: f(n) = 2n is computable

- Test input: 11



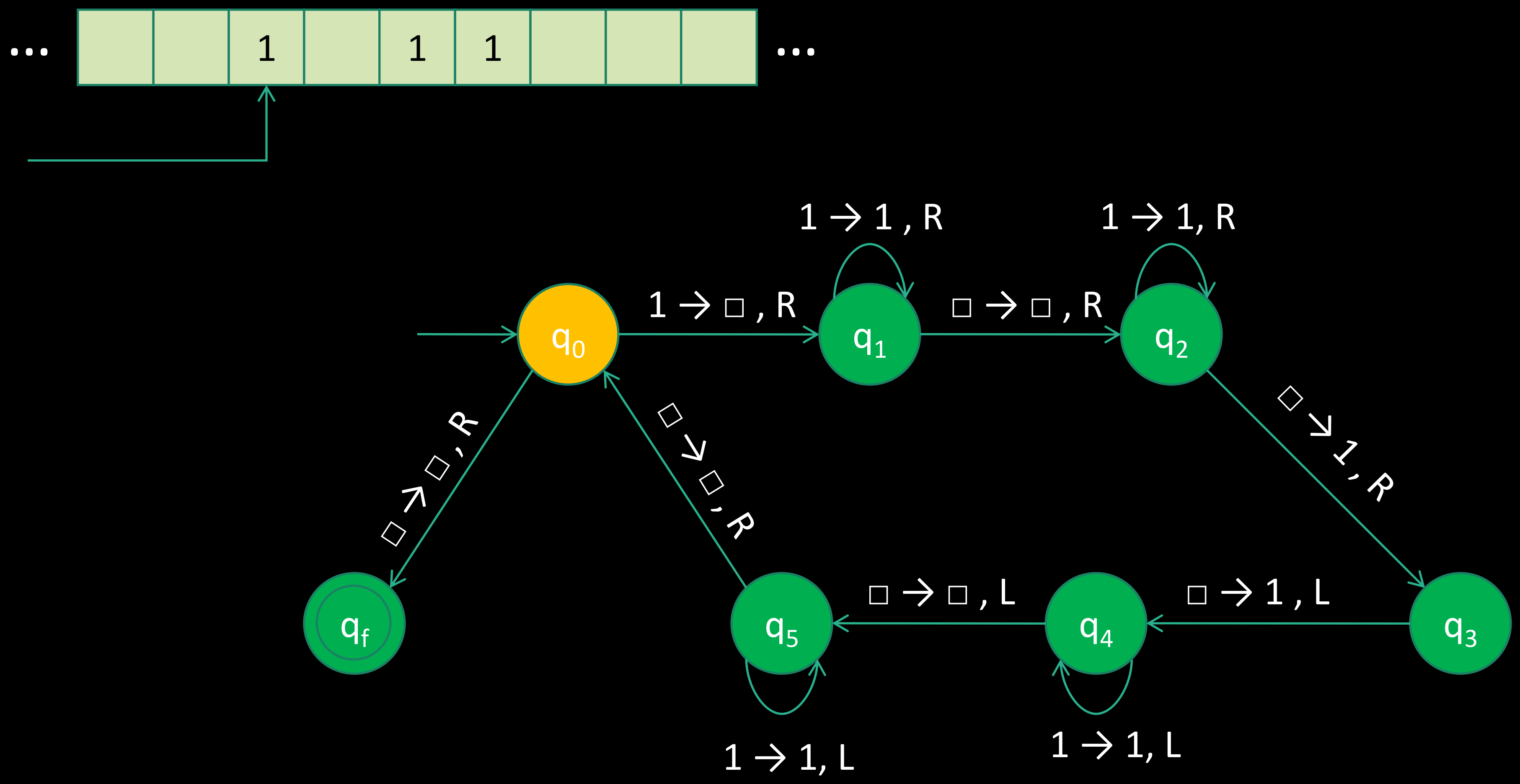
Example: f(n) = 2n is computable

- Test input: 11



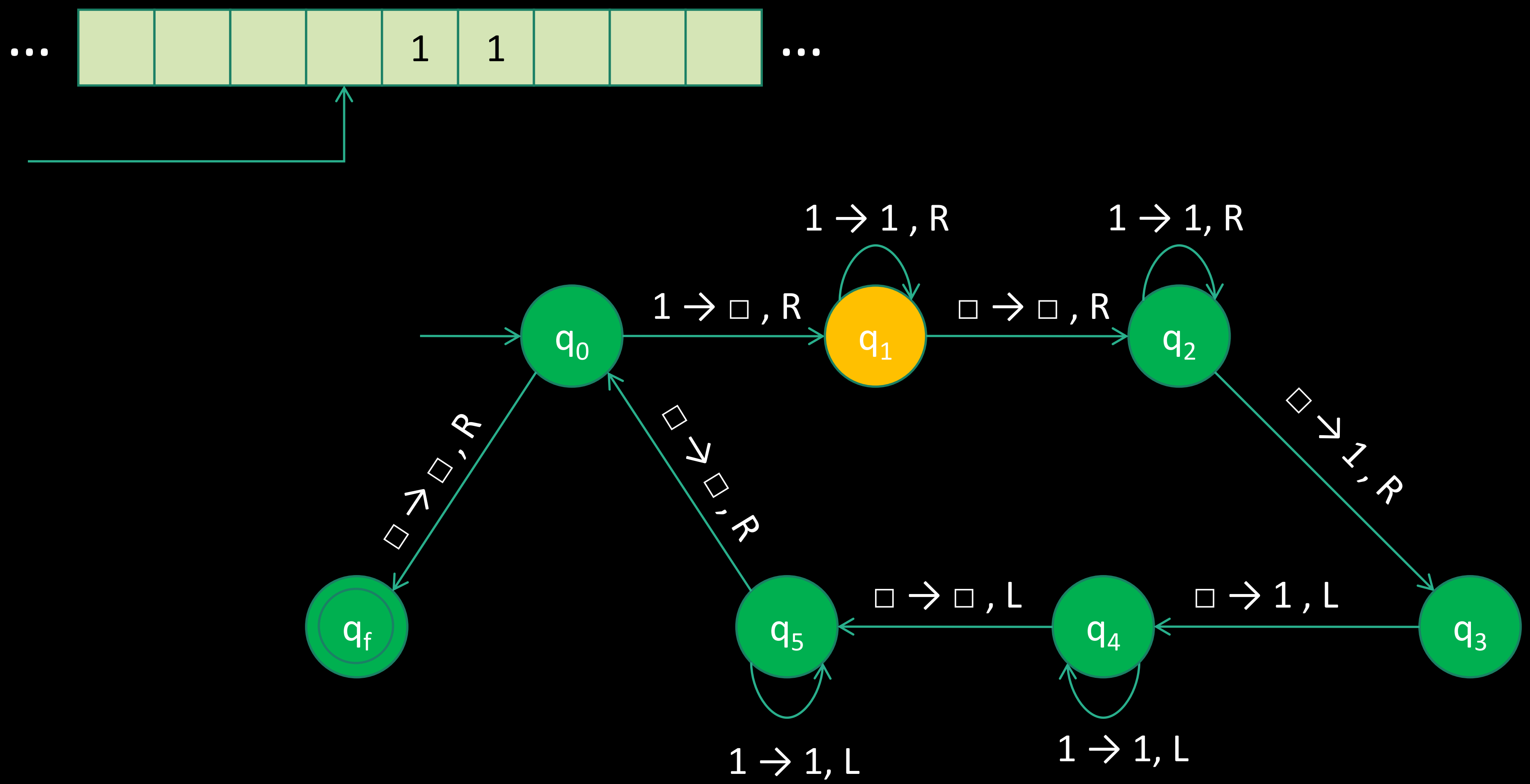
Example: $f(n) = 2n$ is computable

- Test input: 11



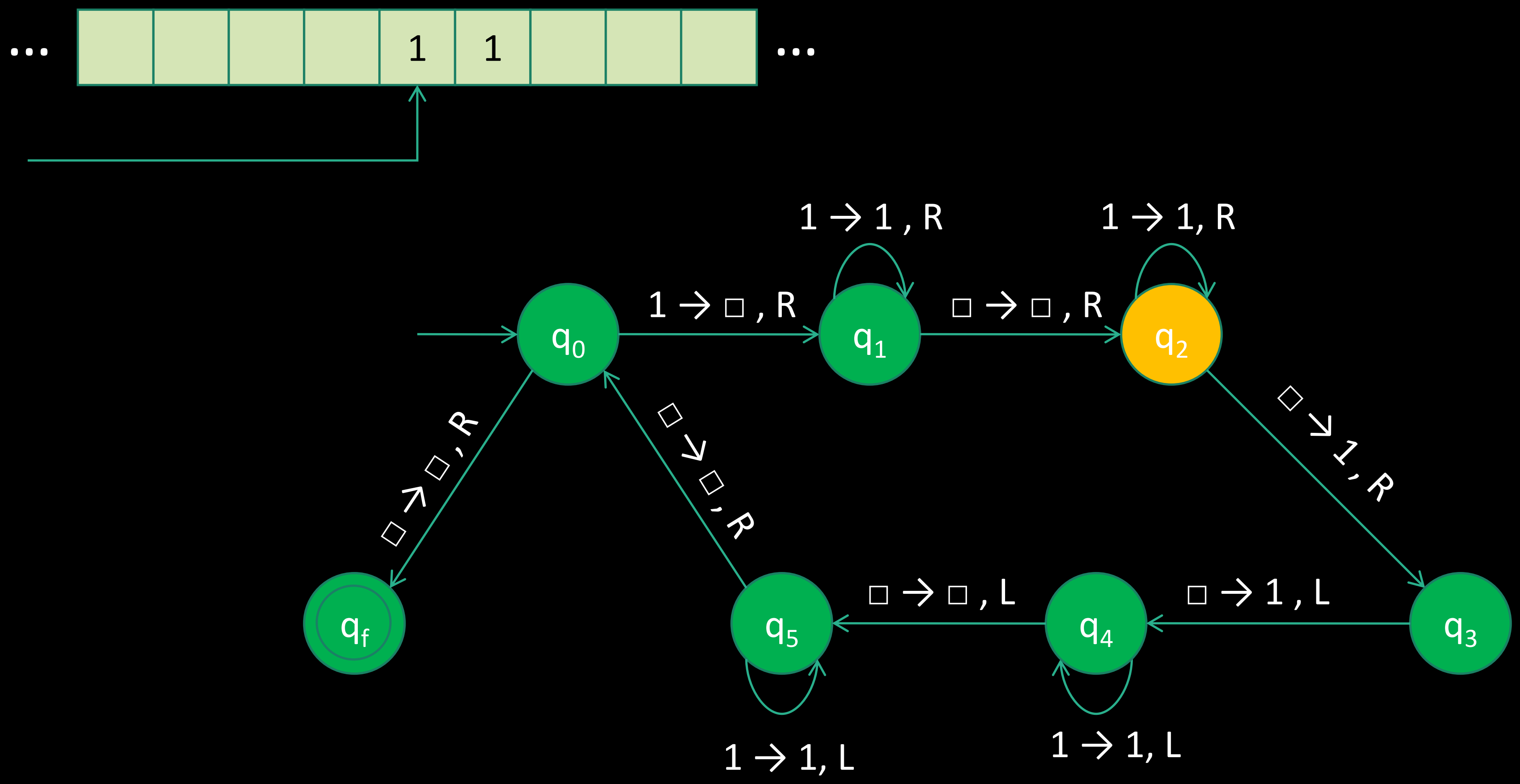
Example: $f(n) = 2n$ is computable

- Test input: 11



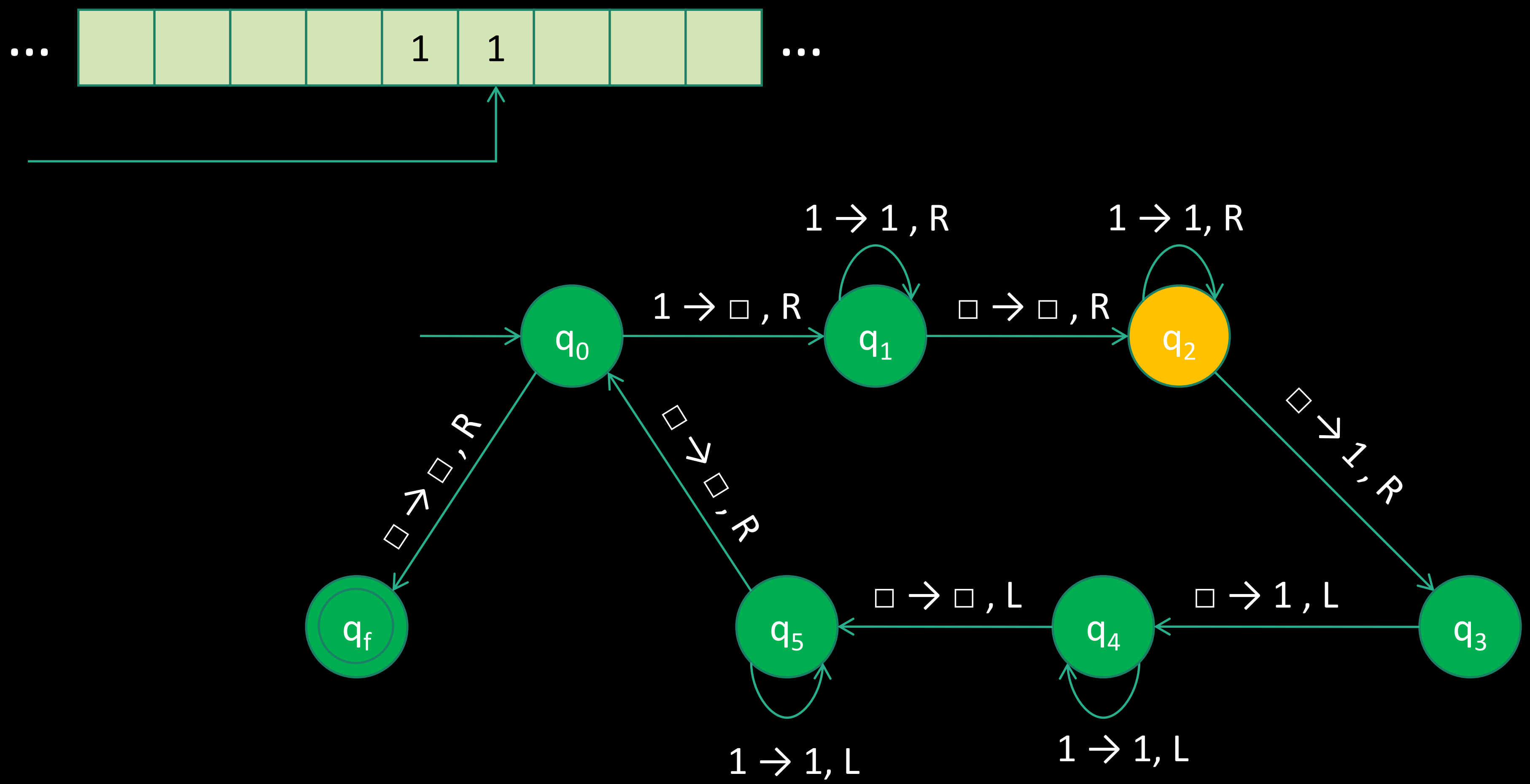
Example: $f(n) = 2n$ is computable

- Test input: 11



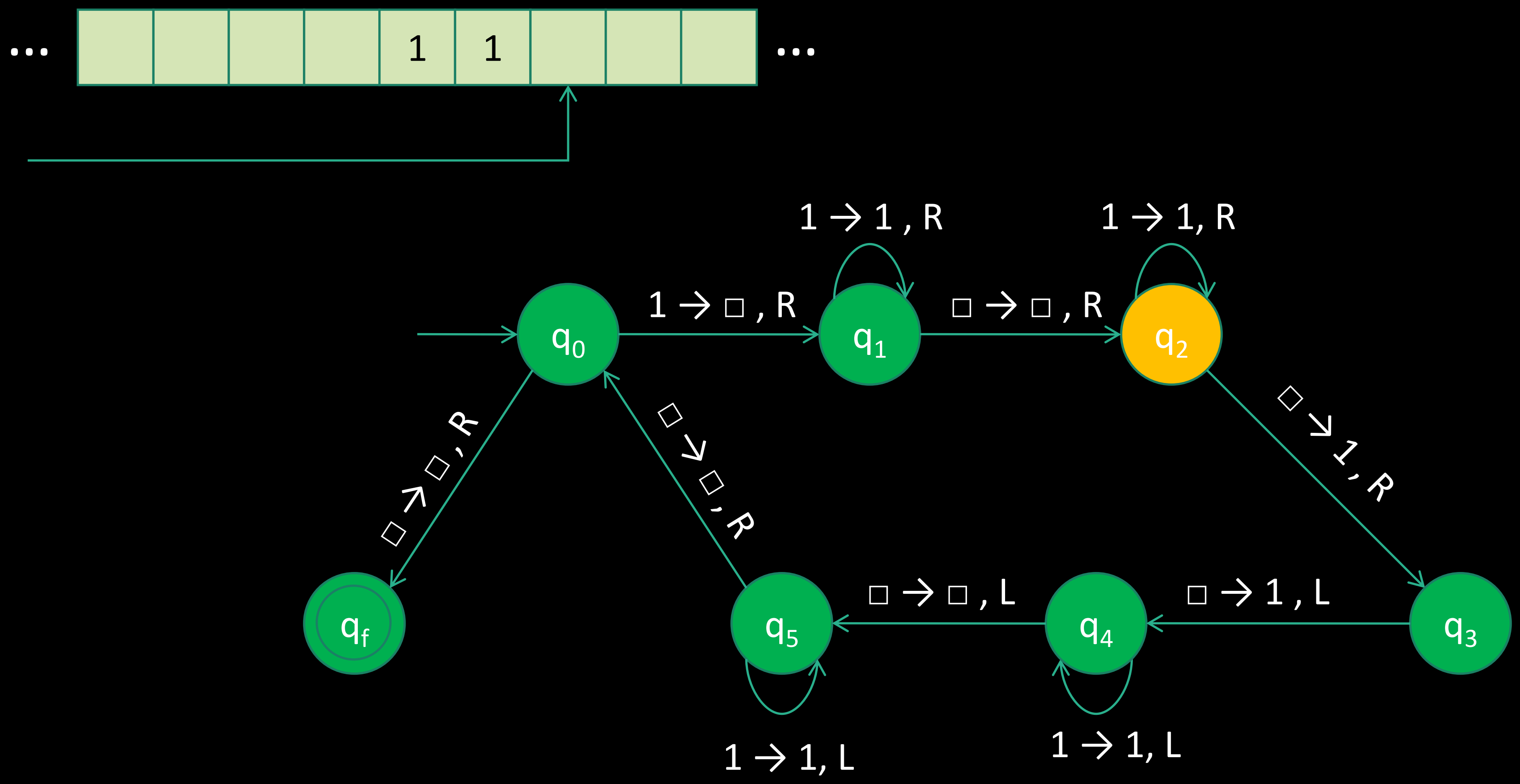
Example: $f(n) = 2n$ is computable

- Test input: 11



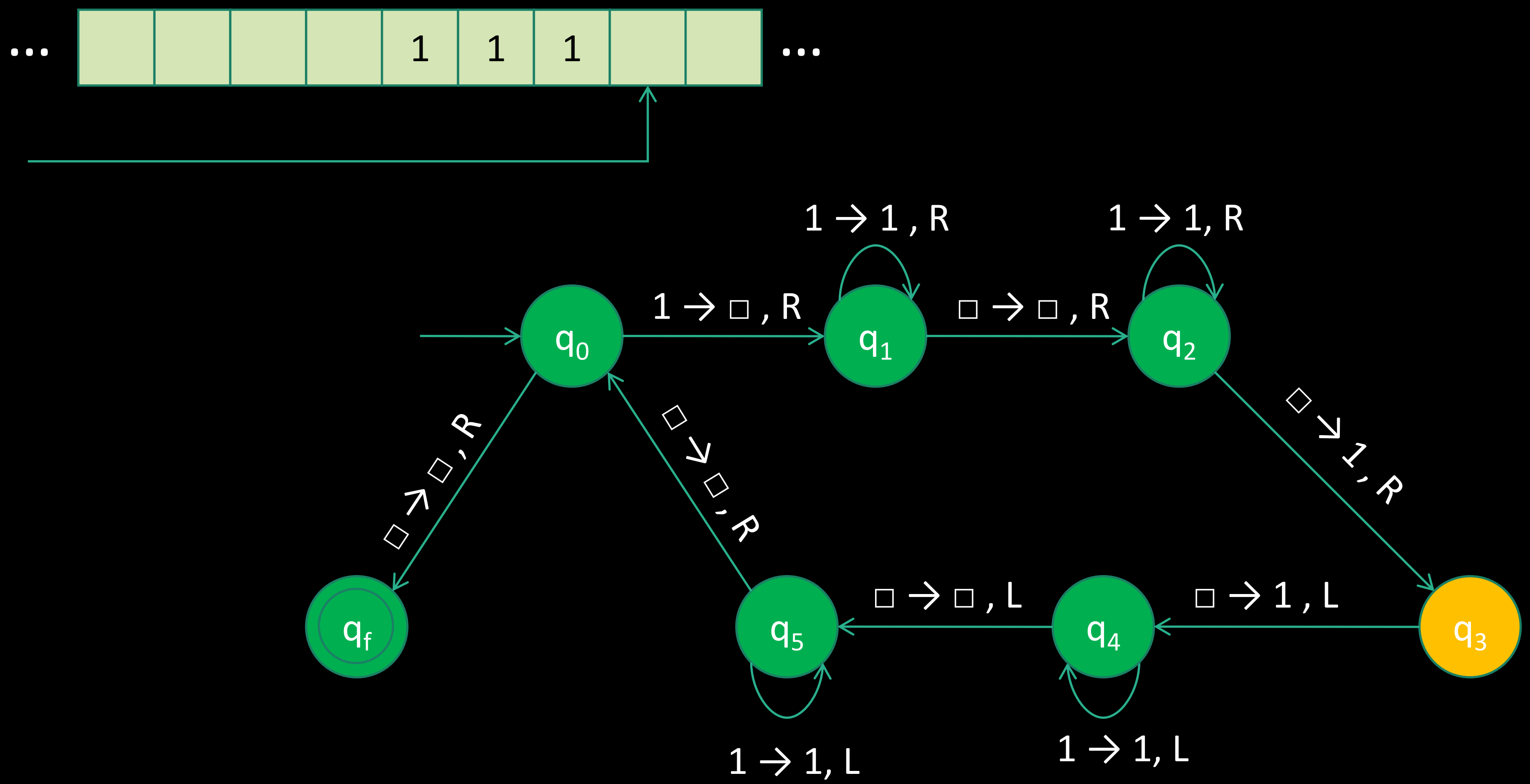
Example: $f(n) = 2n$ is computable

- Test input: 11



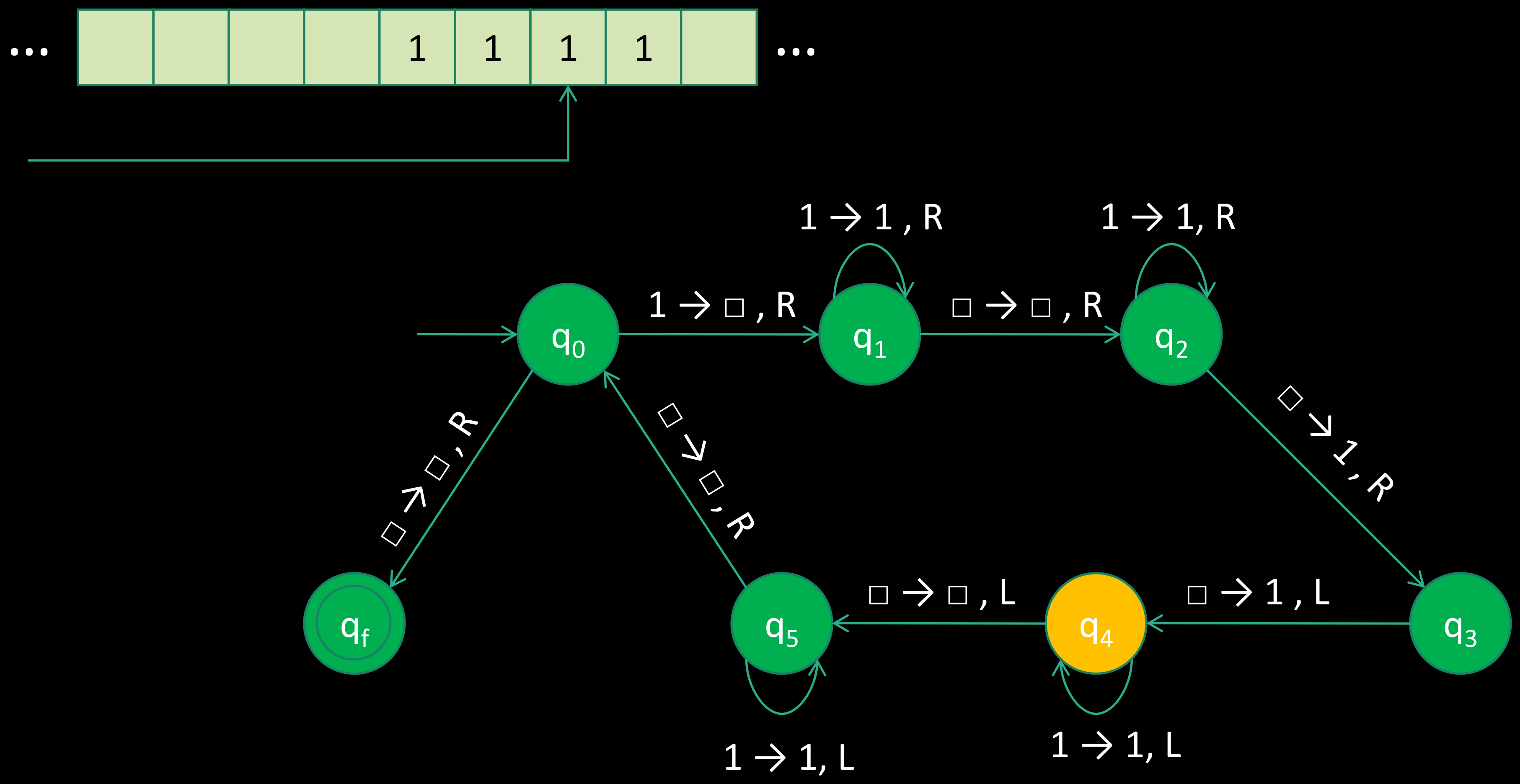
Example: f(n) = 2n is computable

- Test input: 11



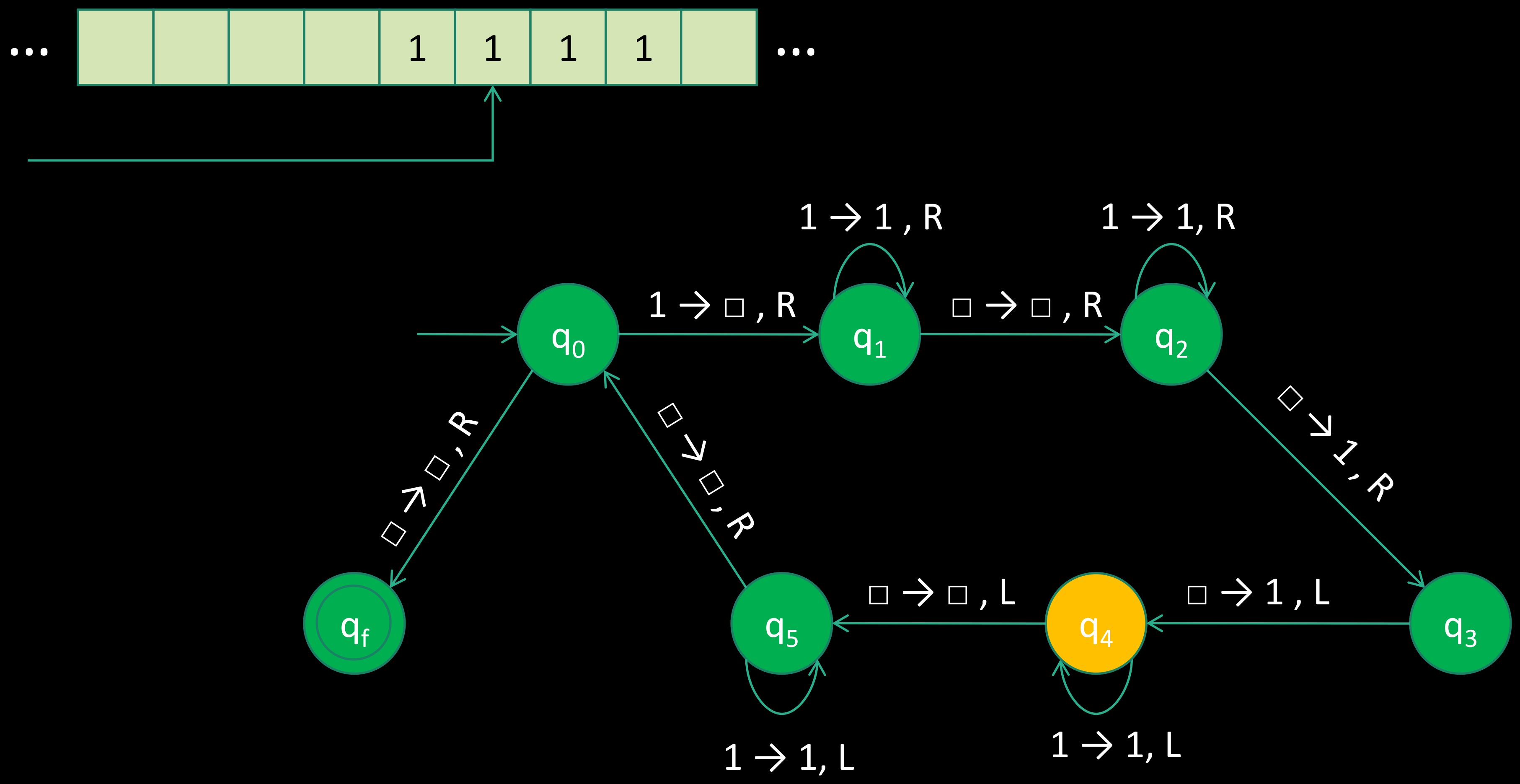
Example: $f(n) = 2n$ is computable

- Test input: 11



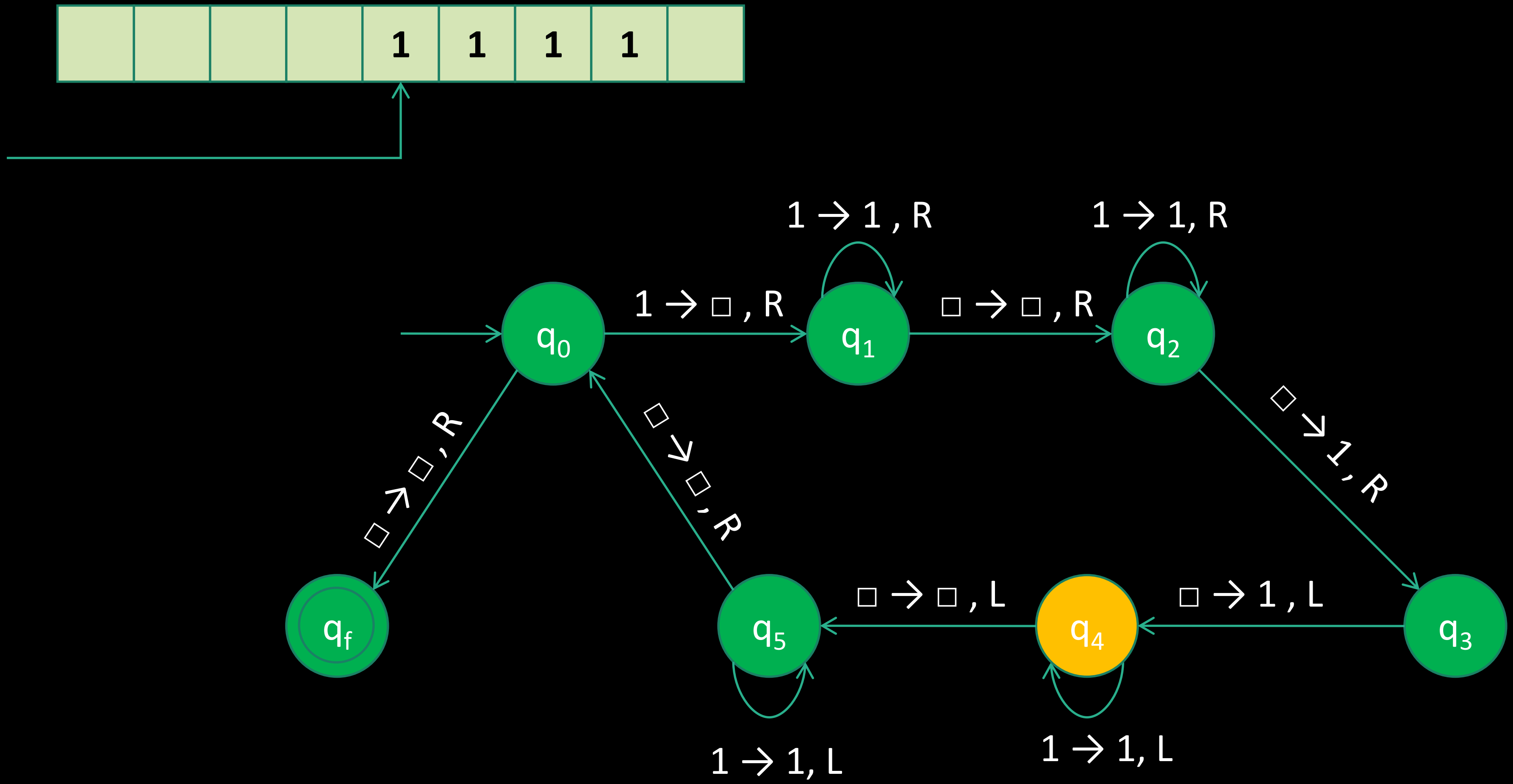
Example: $f(n) = 2n$ is computable

- Test input: 11



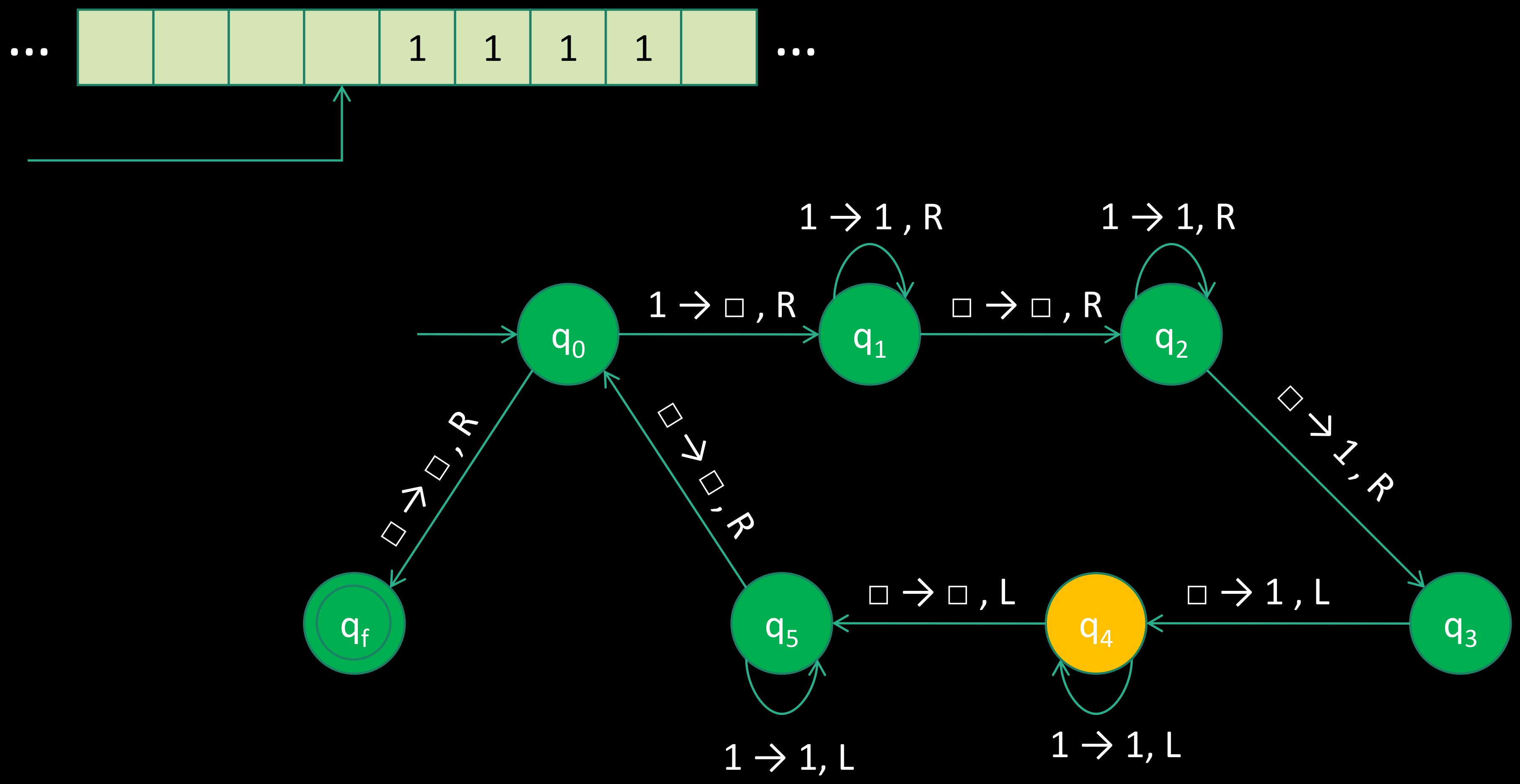
Example: f(n) = 2n is computable

- Test input: 11



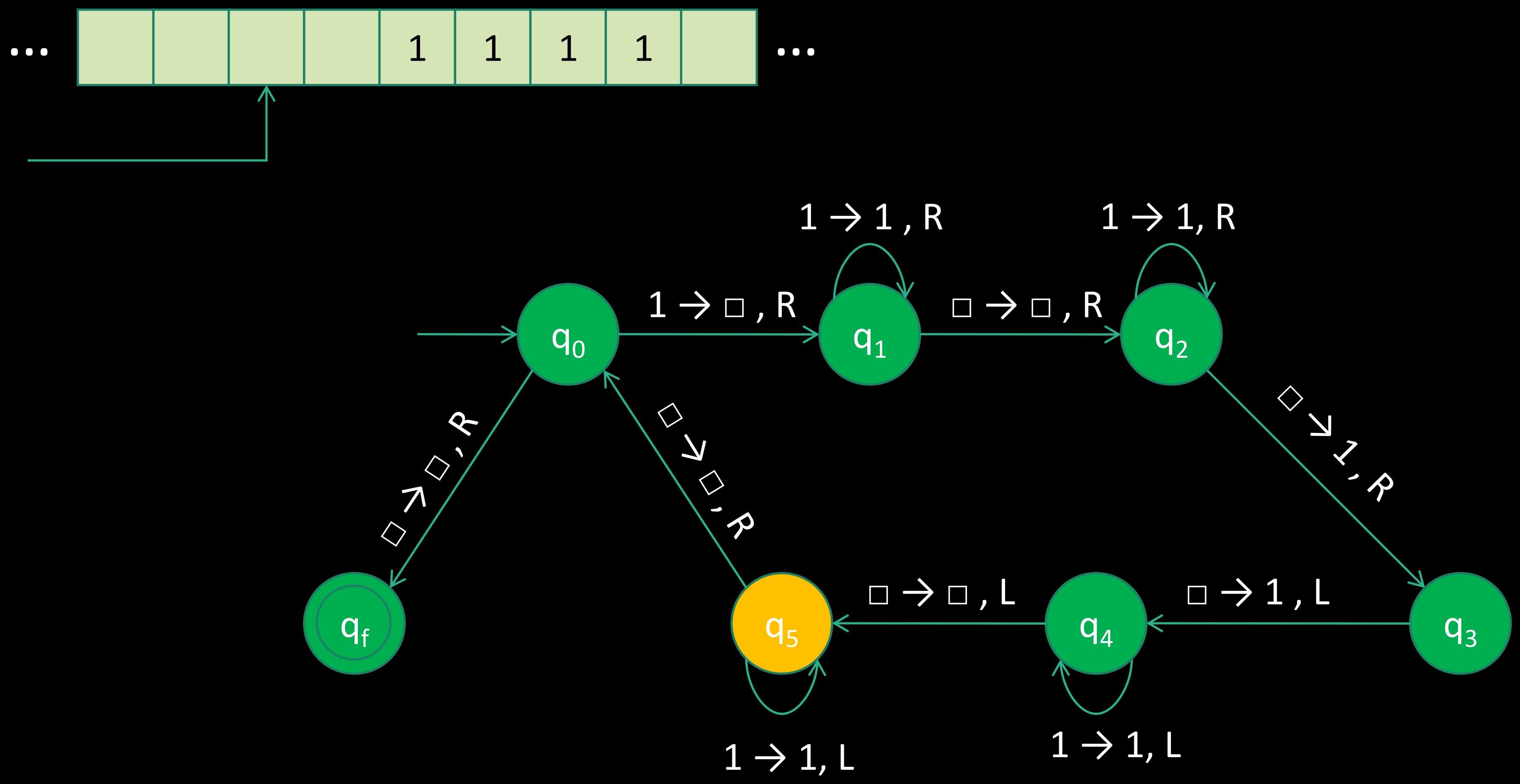
Example: f(n) = 2n is computable

- Test input: 11



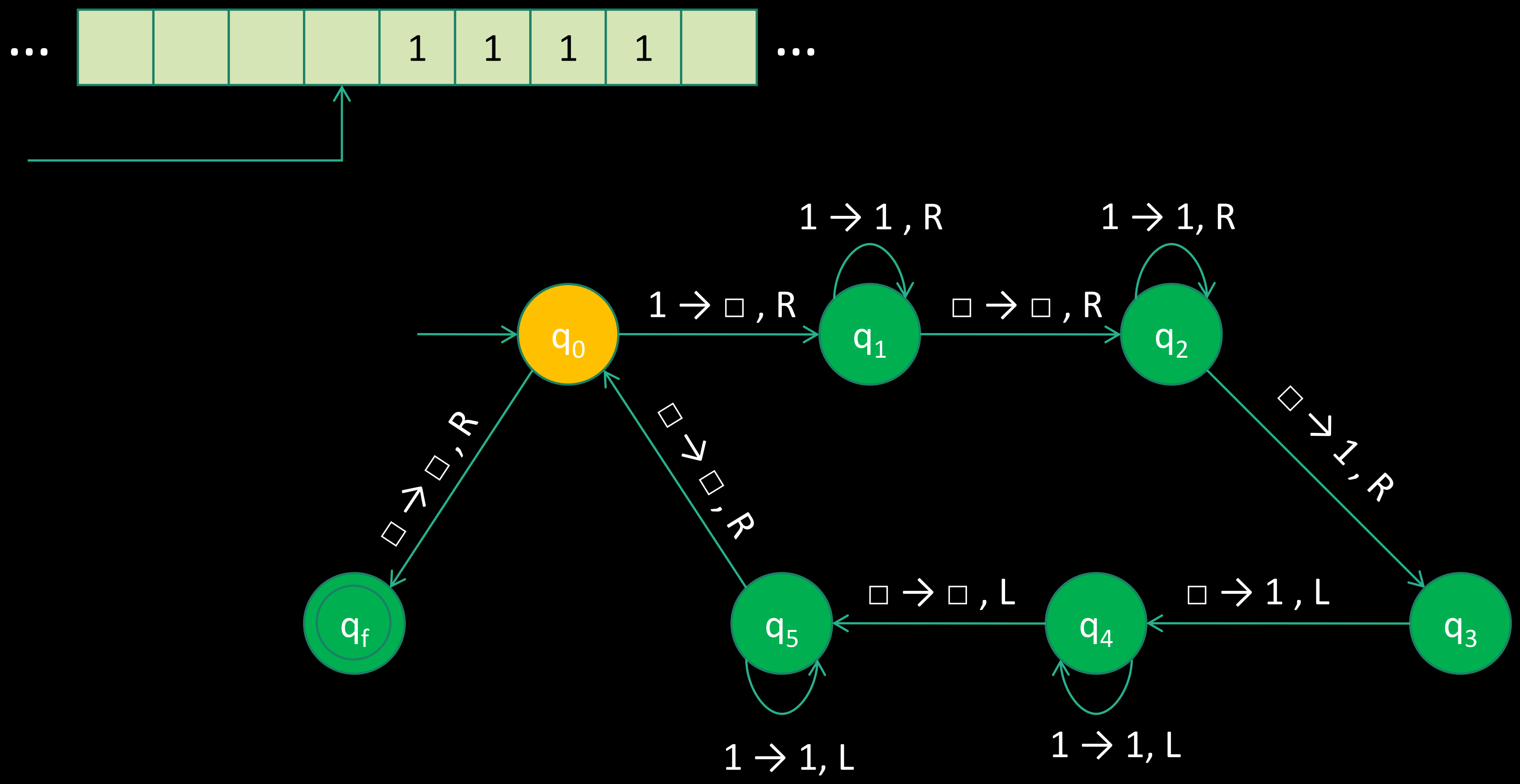
Example: $f(n) = 2n$ is computable

- Test input: 11



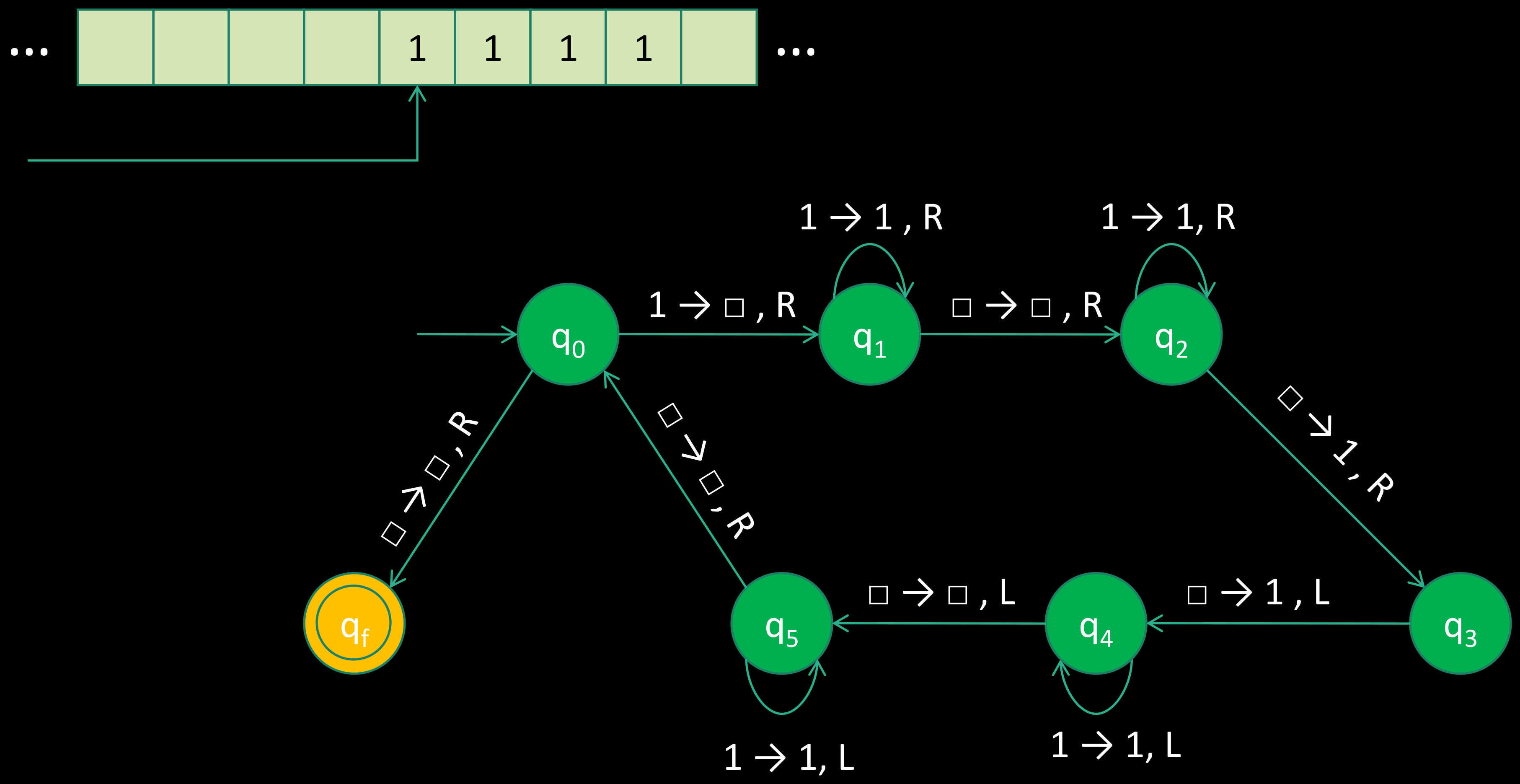
Example: $f(n) = 2n$ is computable

- Test input: 11



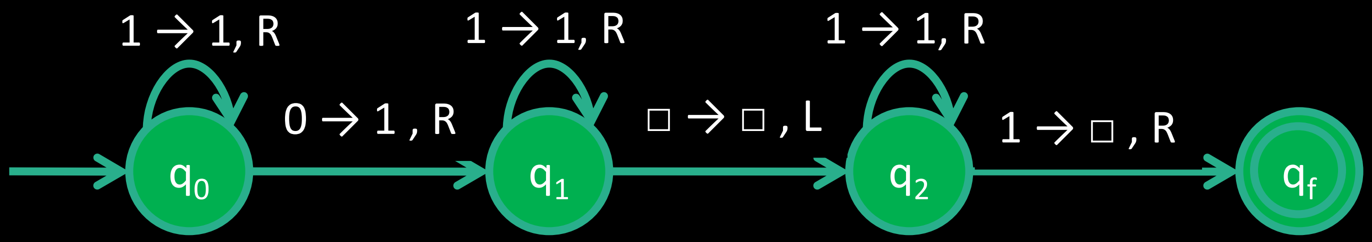
Example: $f(n) = 2n$ is computable

- Test input: 11



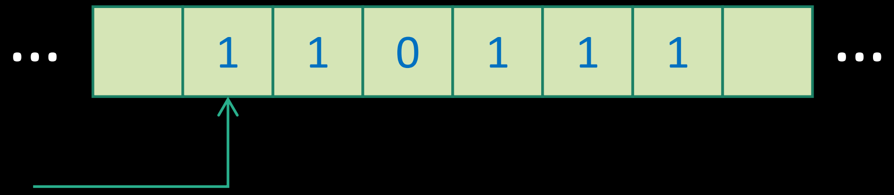
Example: Binary addition is computable

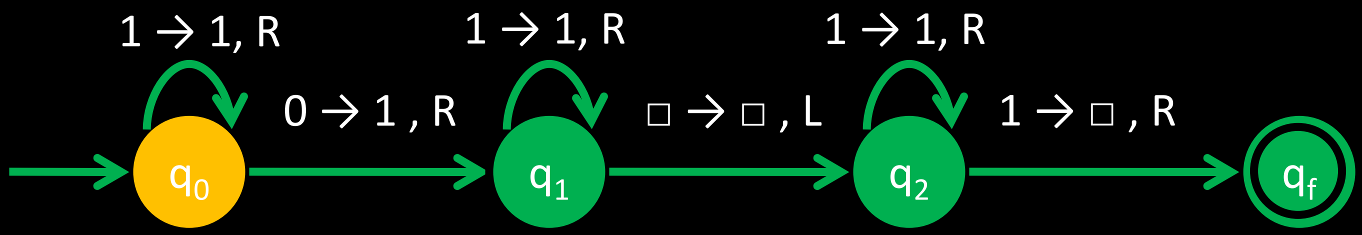
- High level program:
  - Remove the 0 from the middle and make the 1s in the input consecutive.
- TM for this function:



Binary addition is computable

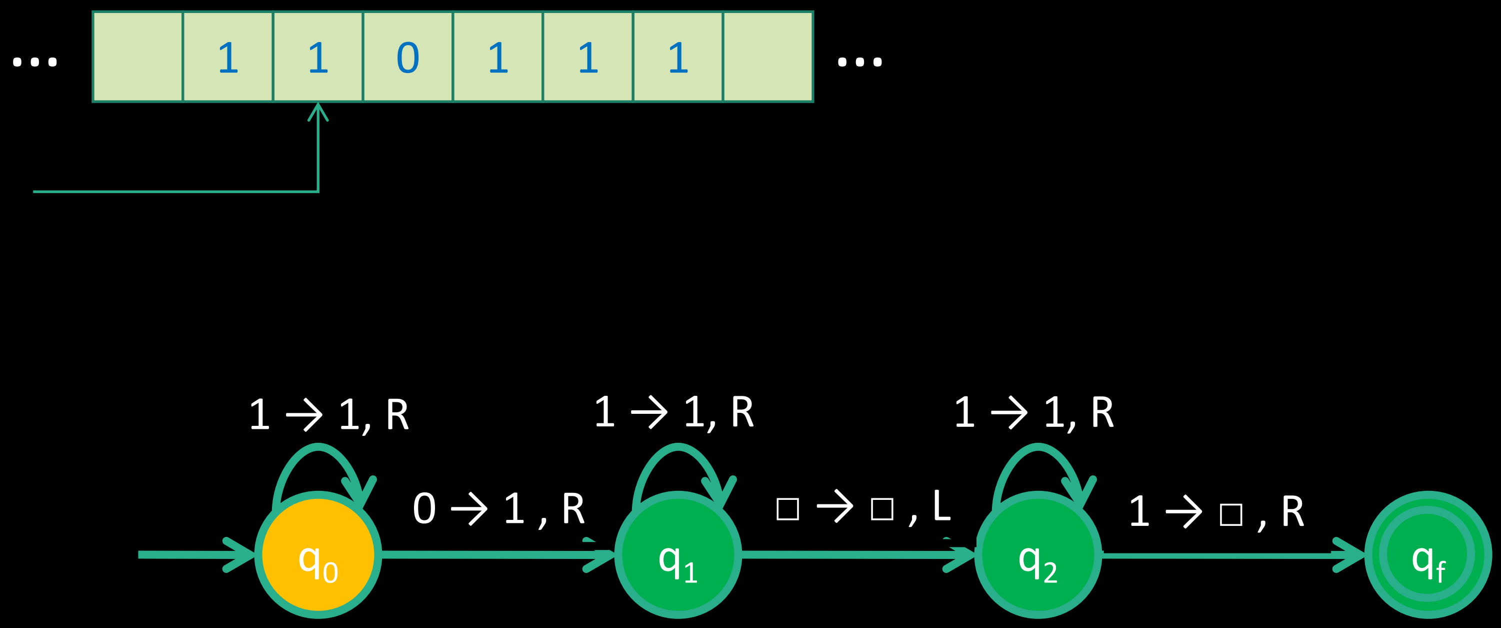
- Test input: (2, 3) = 110111





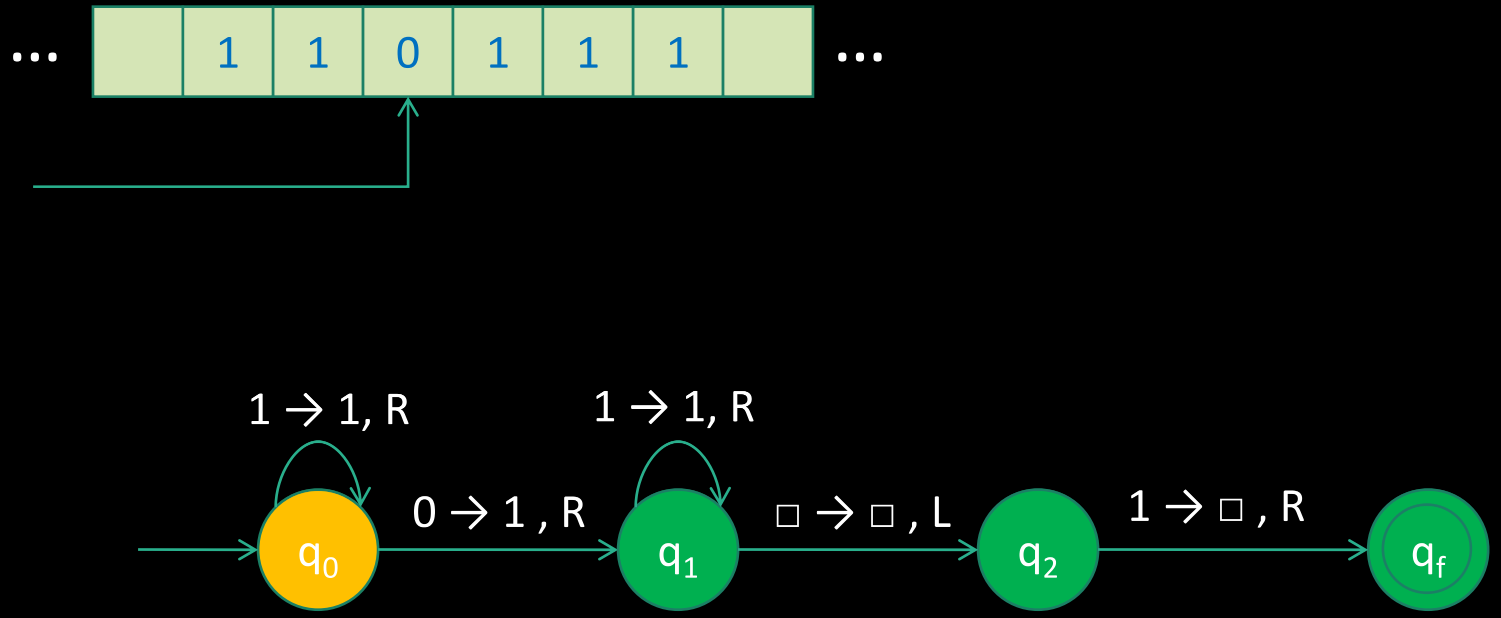
Binary addition is computable

- Test input: (2, 3) = 110111



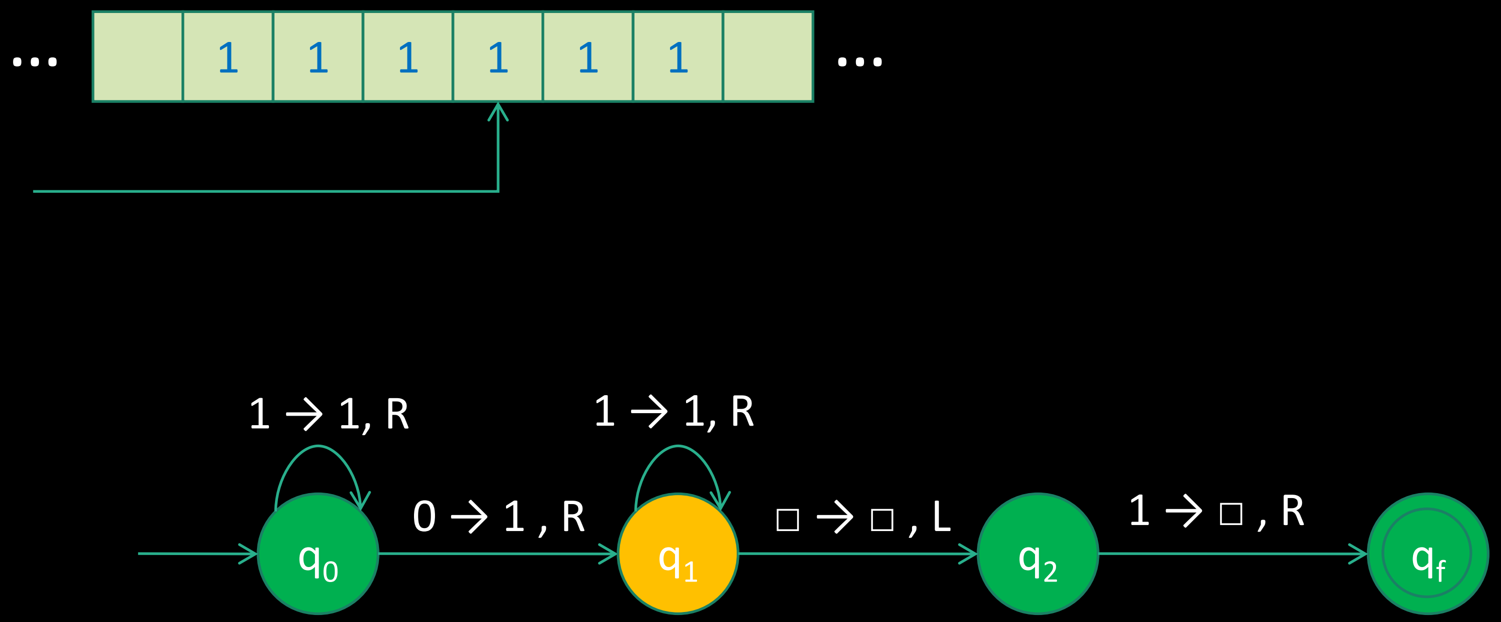
Binary addition is computable

- Test input: (2, 3) = 110111



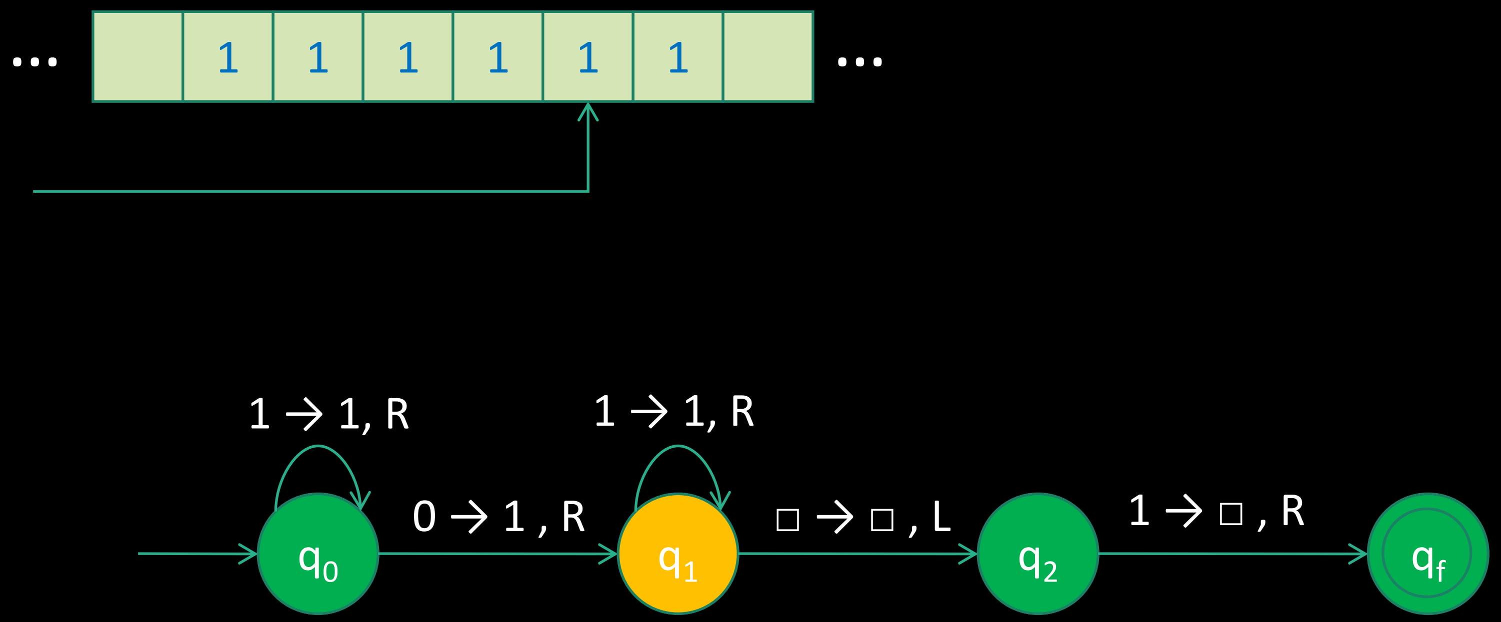
Binary addition is computable

- Test input: (2, 3) = 110111



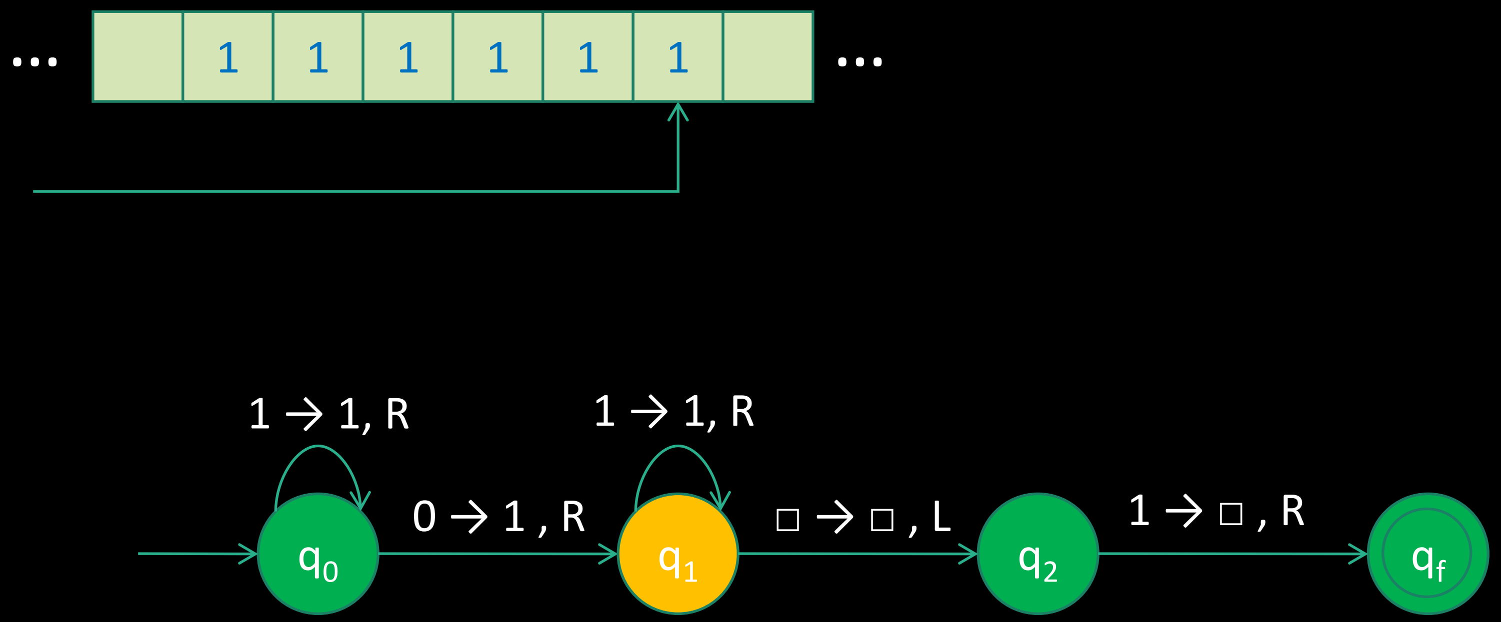
Binary addition is computable

- Test input: (2, 3) = 110111



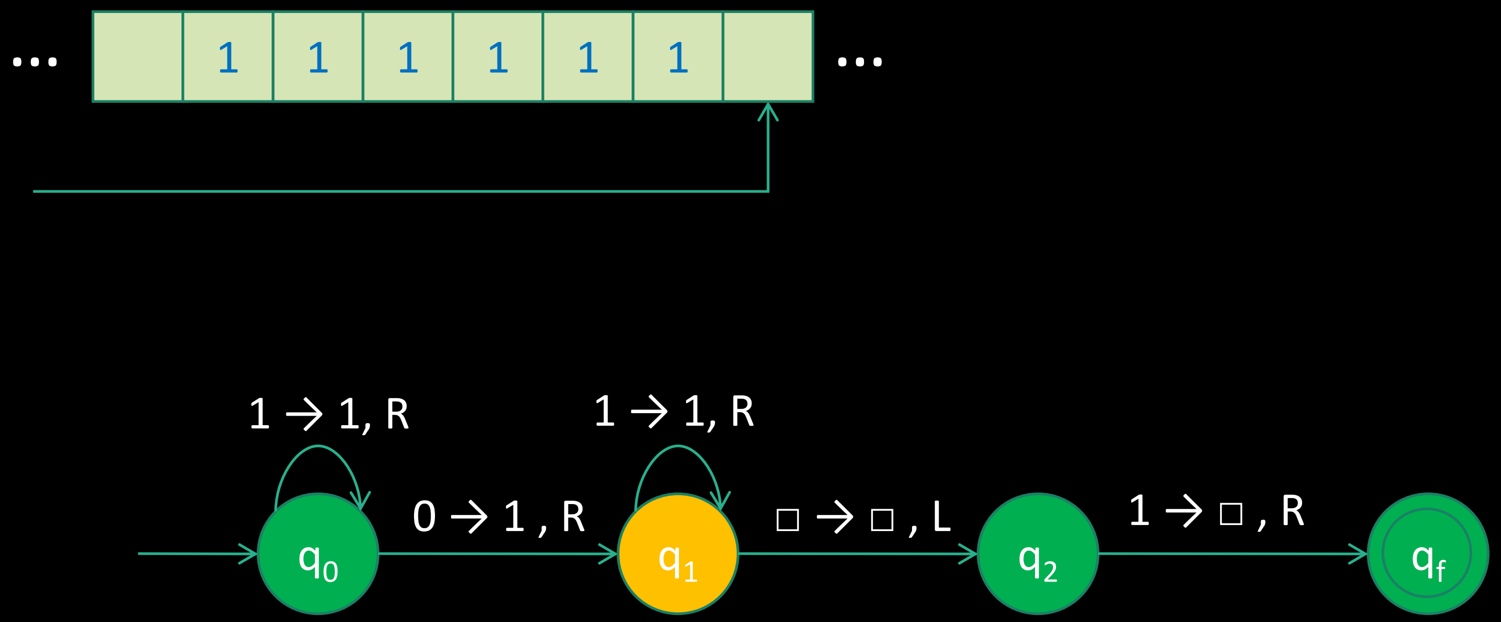
Binary addition is computable

- Test input: (2, 3) = 110111



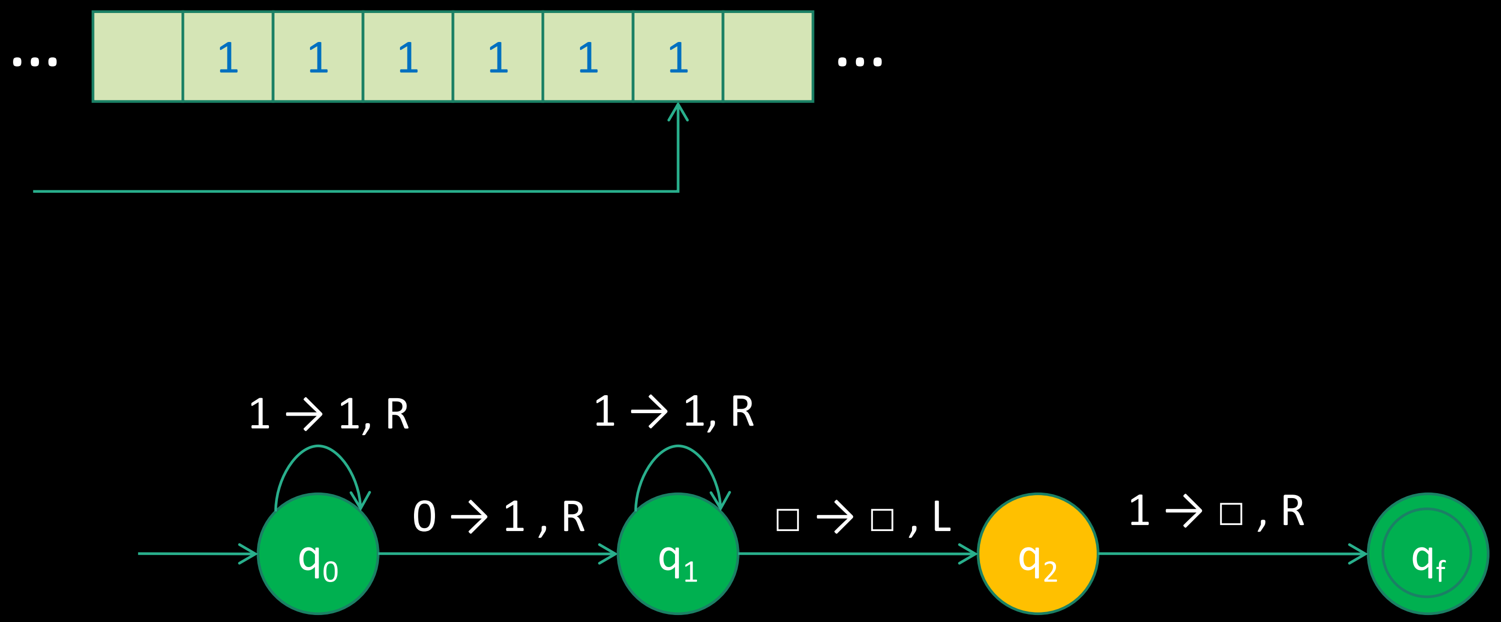
Binary addition is computable

- Test input: (2, 3) = 110111



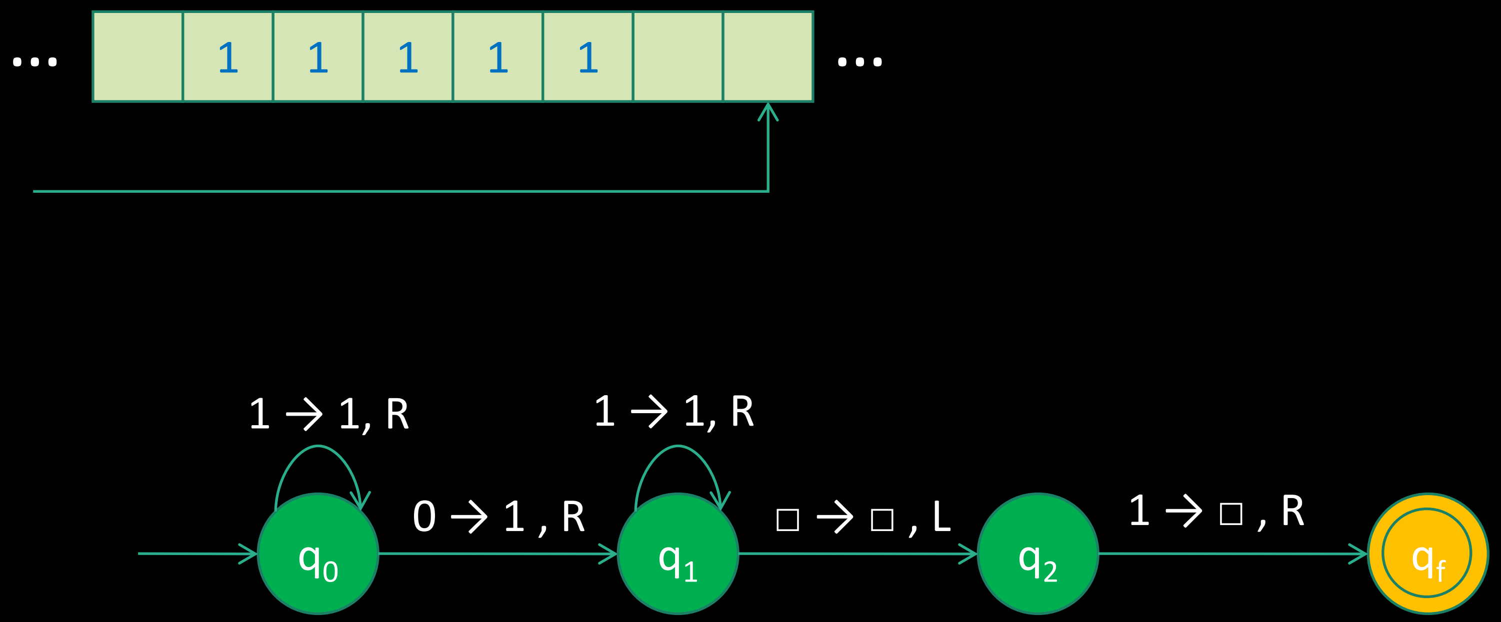
Binary addition is computable

- Test input: (2, 3) = 110111



Binary addition is computable

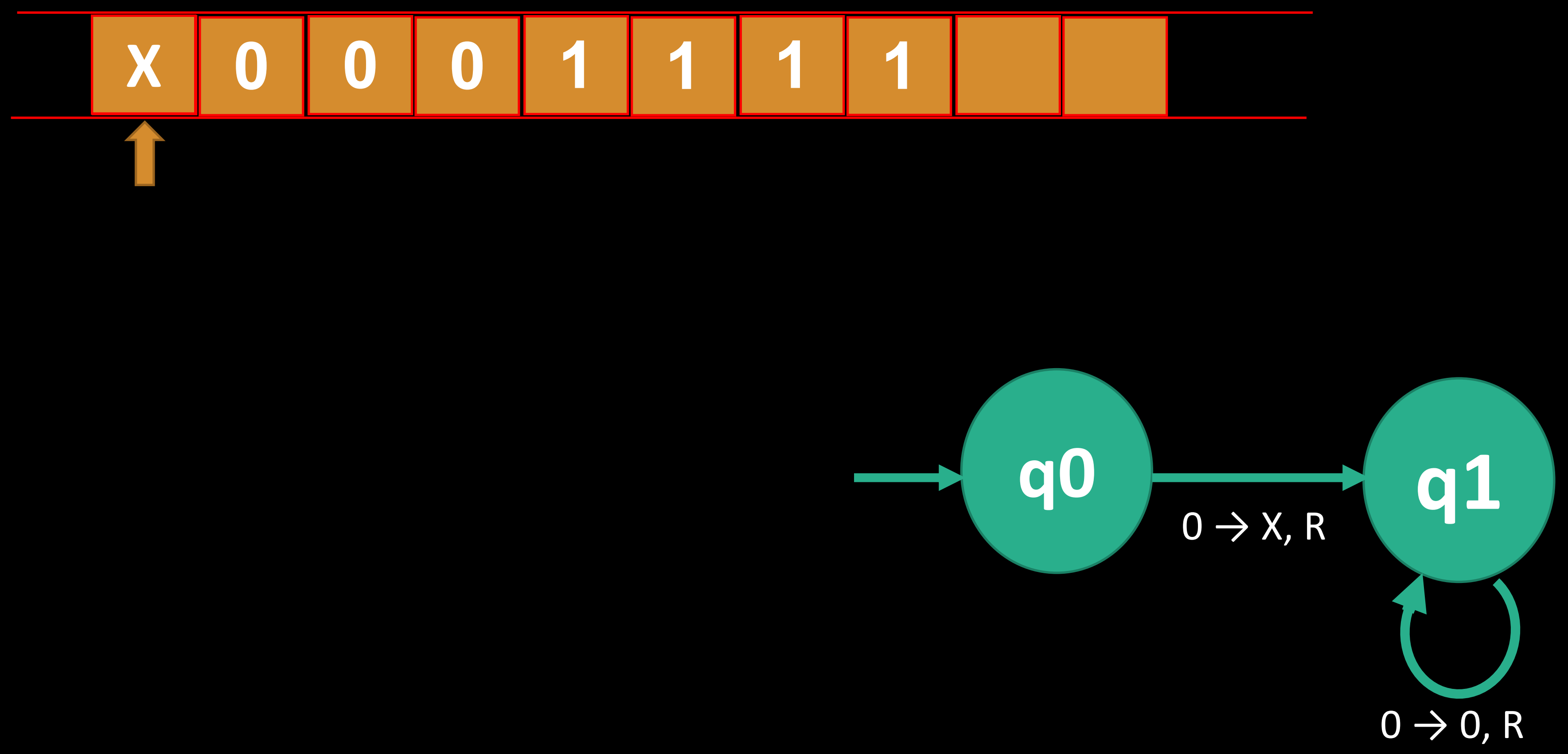
- Test input: (2, 3) = 110111

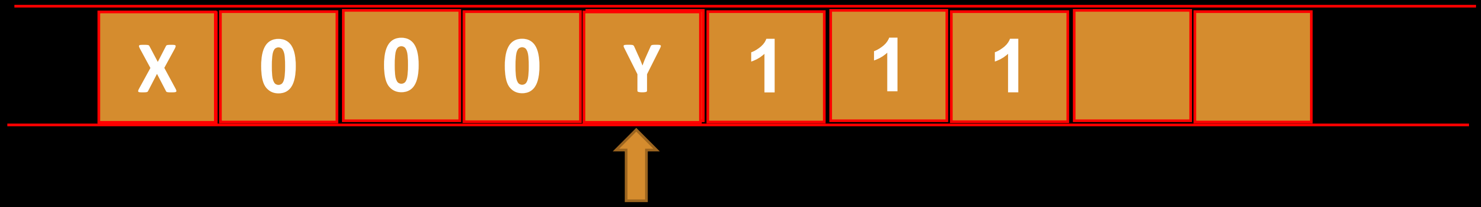


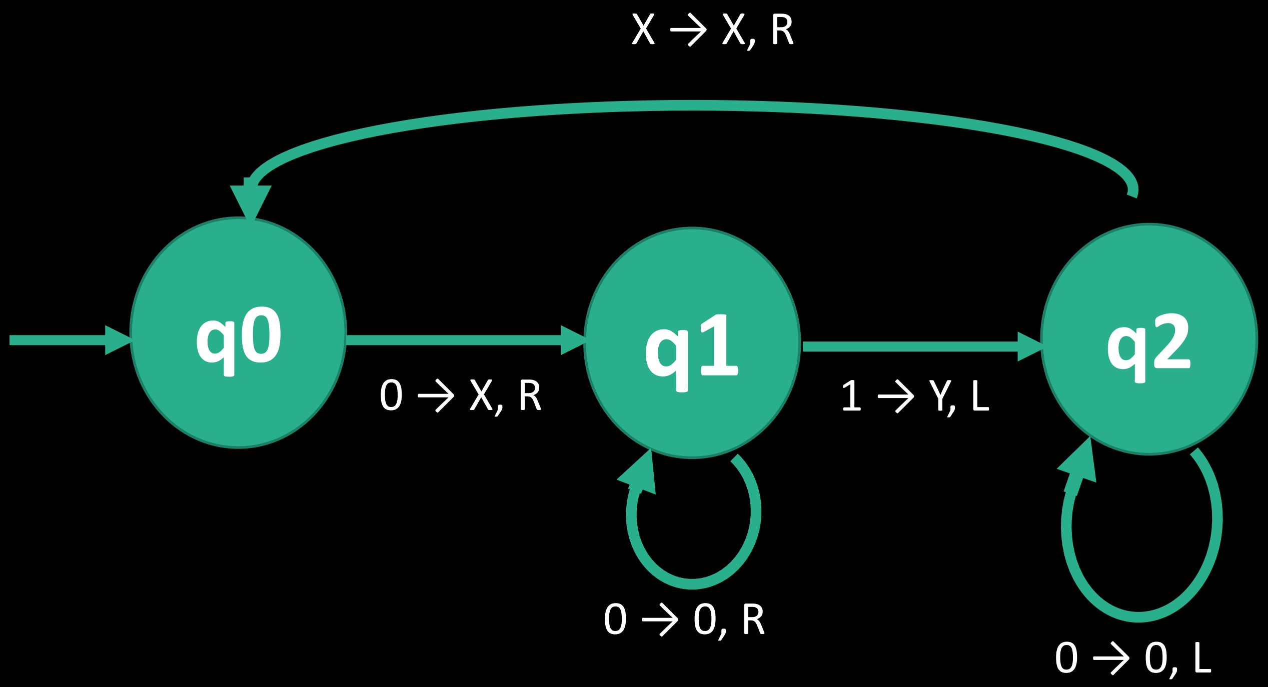
Example

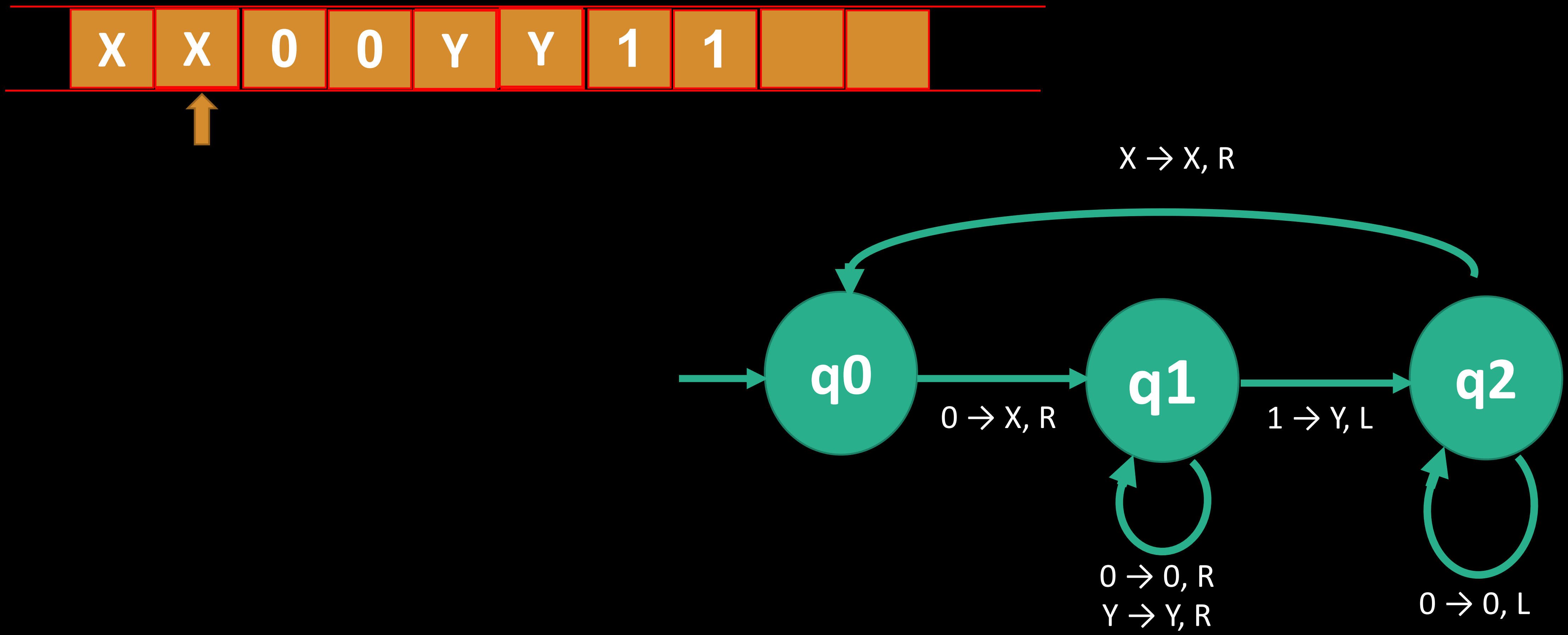
A Turing Machine M that accepts the language

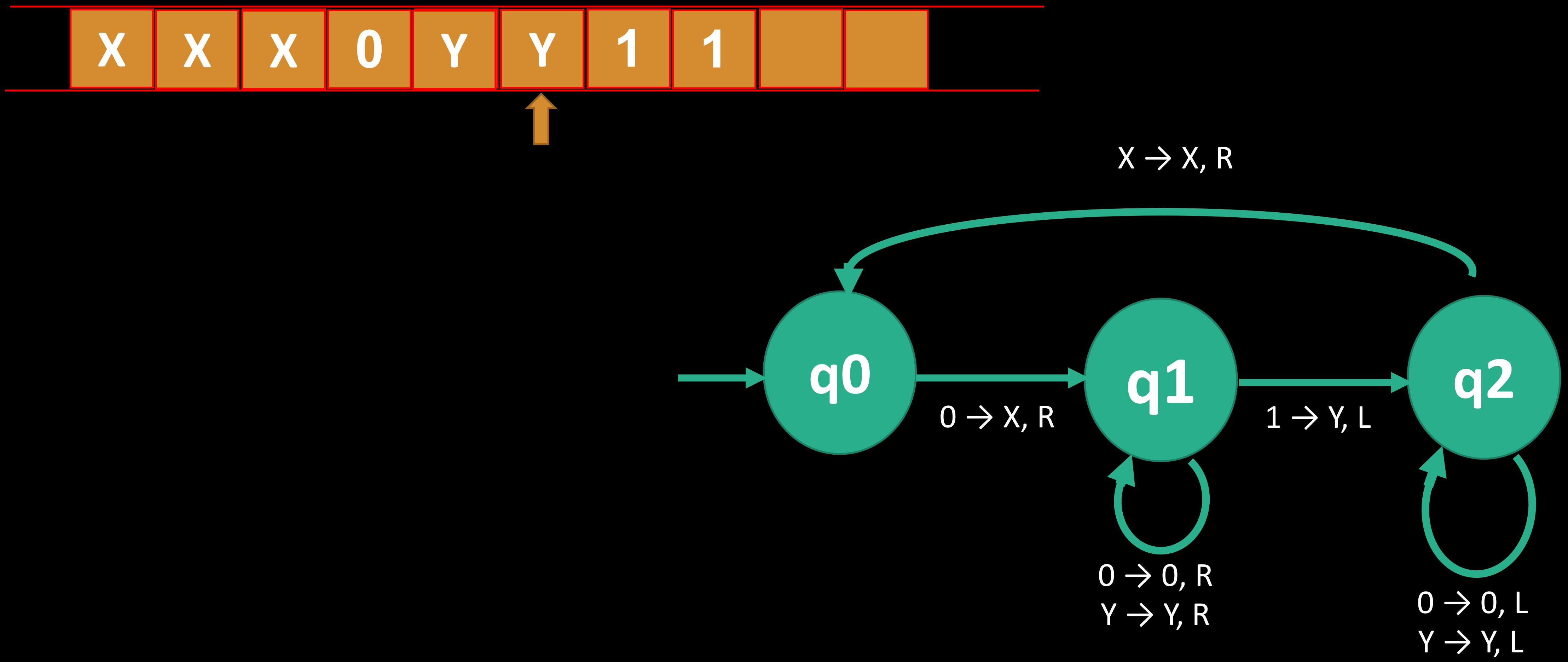
$\{ 0^n1^n \mid n \geq 0 \}$


X
0
0
0
1
1
1
1
q0
0 → X, R
q1
0 → 0, R

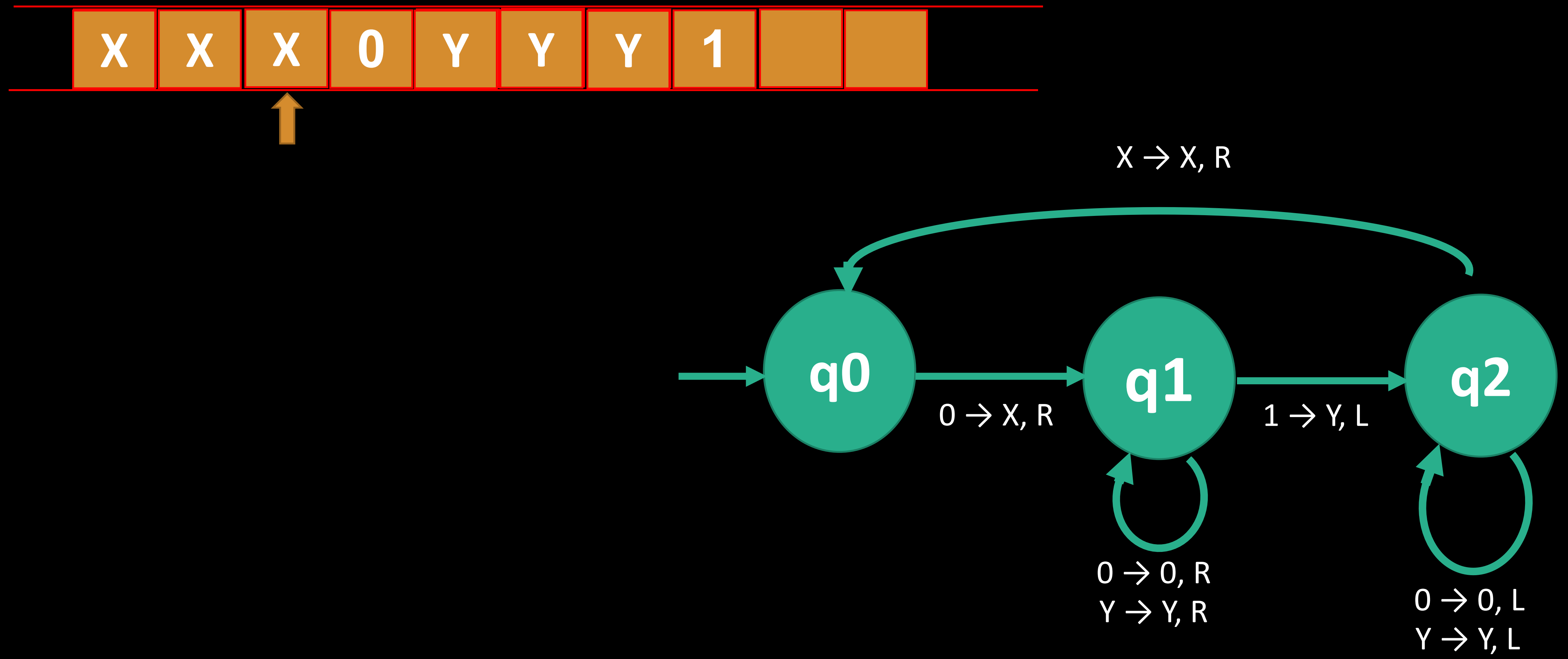

X
0
0
0
Y
1
1
1

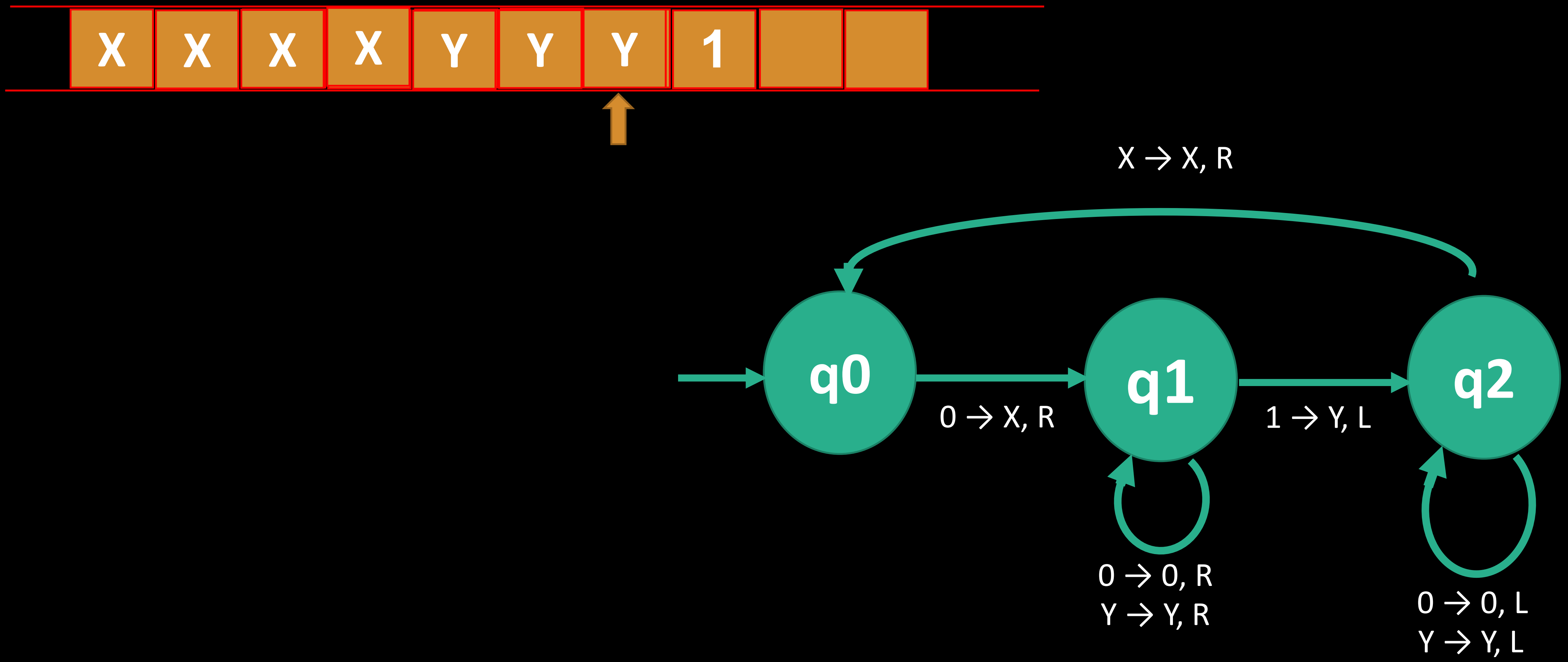

X → X, R
q0
q1
q2
0 → X, R
1 → Y, L
0 → 0, R
0 → 0, L


X
X
0
0
Y
Y
1
1
X → X, R
q0
0 → X, R
q1
1 → Y, L
q2
0 → 0, R
Y → Y, R
0 → 0, L

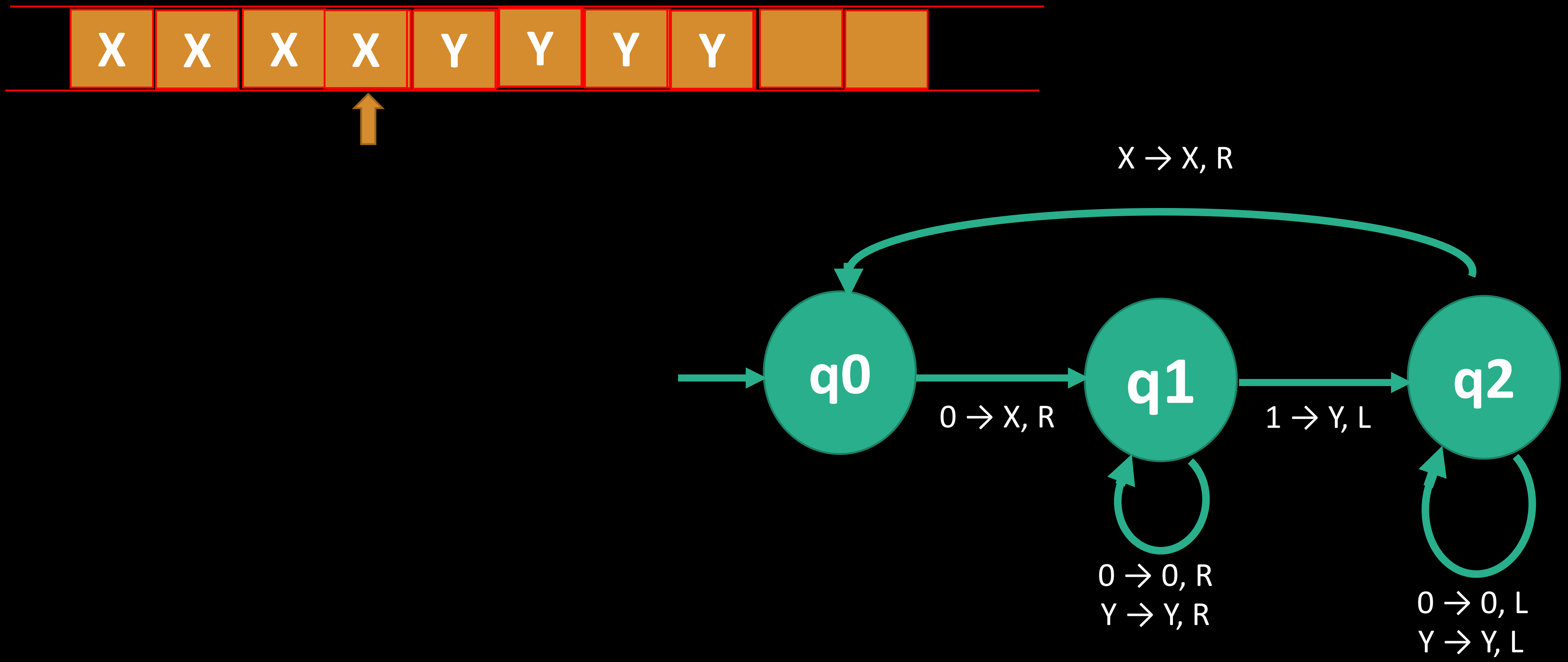


X X X 0 Y Y 1 1
X → X, R
q0
q1
q2
0 → X, R
1 → Y, L
0 → 0, R
Y → Y, R
0 → 0, L
Y → Y, L

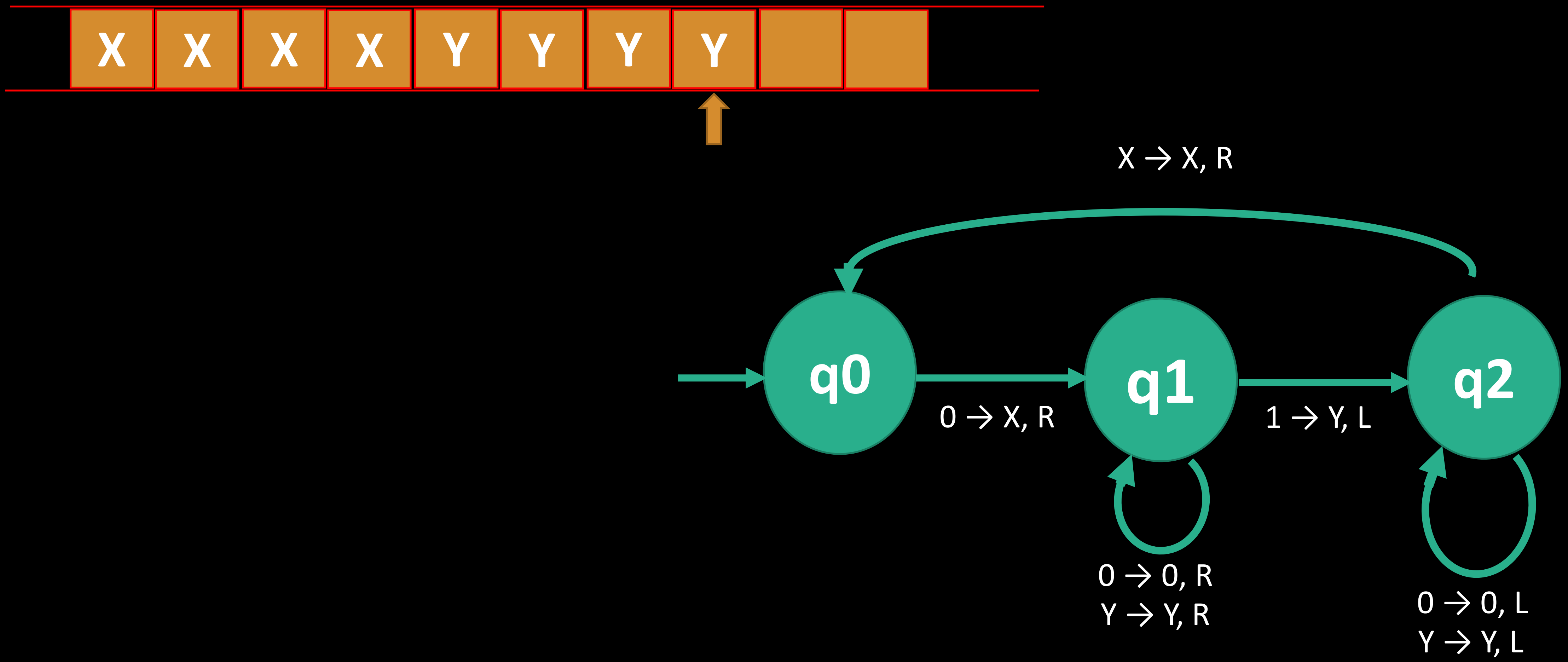

X
X
X
0
Y
Y
Y
1
X → X, R
q0
0 → X, R
q1
1 → Y, L
q2
0 → 0, R
Y → Y, R
0 → 0, L
Y → Y, L



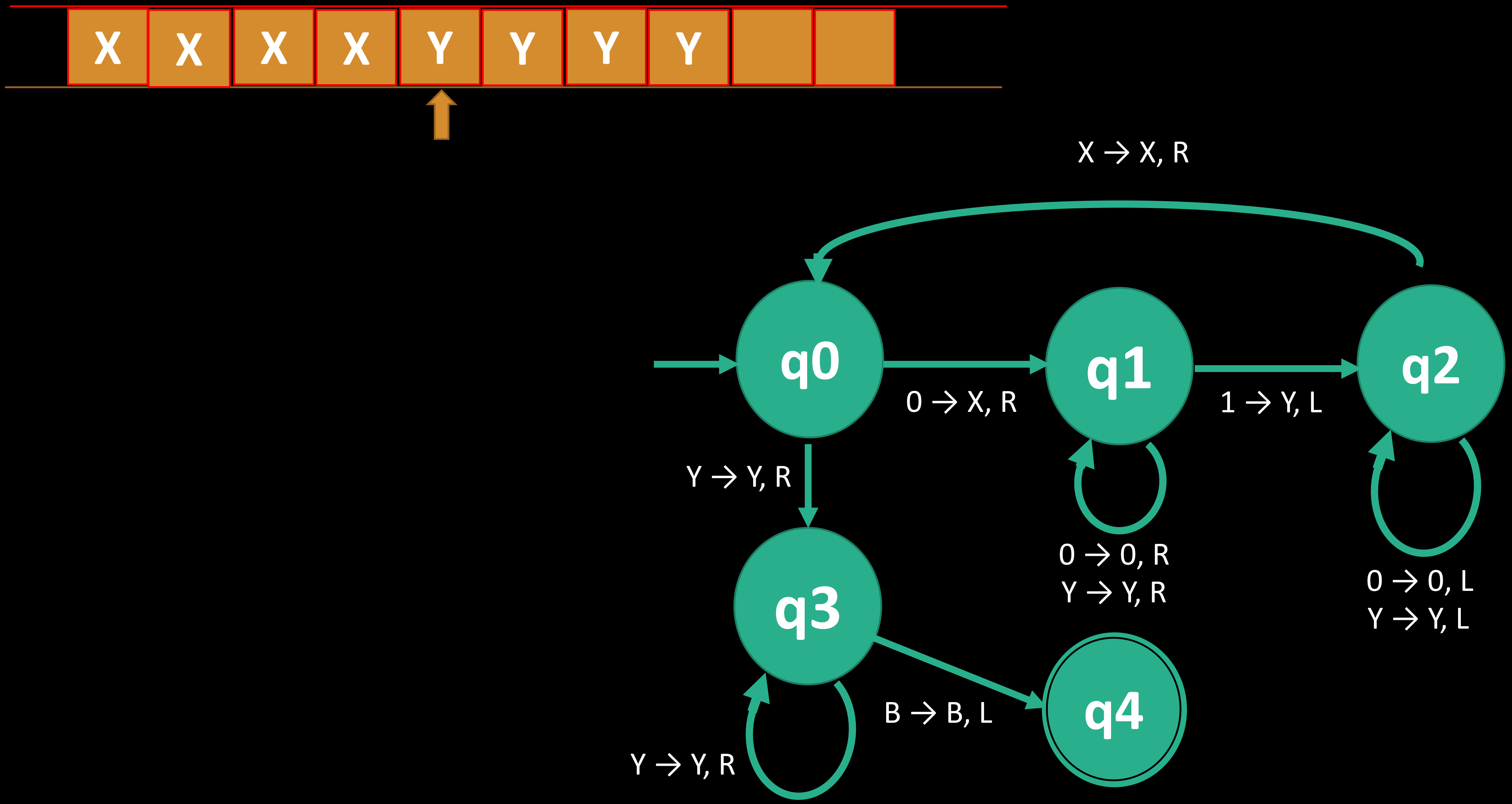
X X X X Y Y Y 1
X → X, R
q0
q1
q2
0 → X, R
1 → Y, L
0 → 0, R
Y → Y, R
0 → 0, L
Y → Y, L



X X X X Y Y Y Y
X → X, R
q0
0 → X, R
q1
1 → Y, L
q2
0 → 0, R
Y → Y, R
0 → 0, L
Y → Y, L



X
X
X
X
Y
Y
Y
Y
X → X, R
q0
q1
q2
0 → X, R
1 → Y, L
0 → 0, R
Y → Y, R
0 → 0, L
Y → Y, L


X
X
X
X
Y
Y
Y
Y
X → X, R
q0
q1
q2
0 → X, R
1 → Y, L
Y → Y, R
0 → 0, R
Y → Y, R
0 → 0, L
Y → Y, L
q3
q4
B → B, L
Y → Y, R

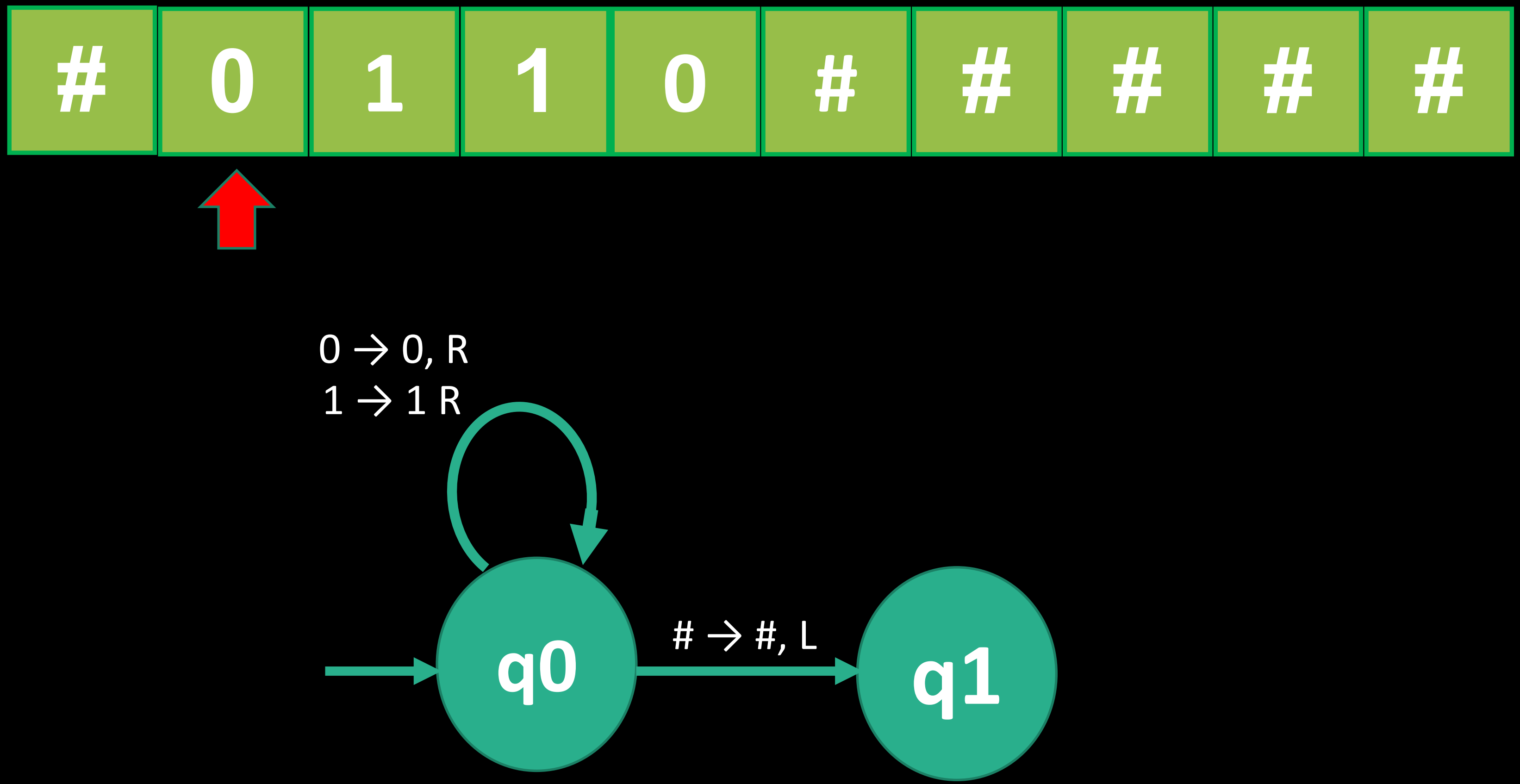
Example

Design a TM to add 1 to a binary number X.

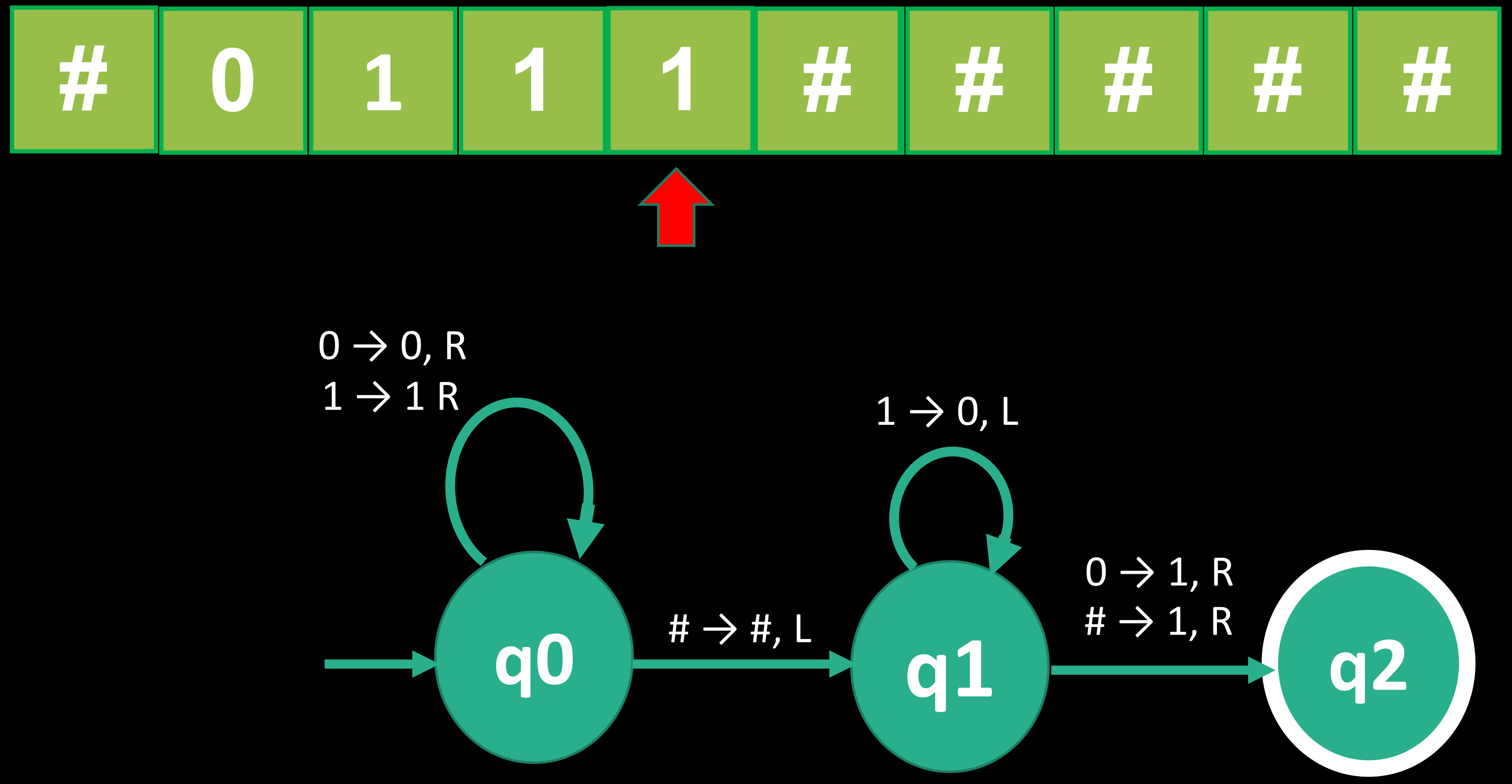
Carry	0	1	1	0
X	0	0	1	1
				1
Result	0	1	0	0

الية العمل:

1. تقرأ الالة رمزا دون تغييره وتتحرك نحو اليمين.
2. تكرر الخطوة 1 حتى تصل الى الرمز #.
3. تتحرك خطوة واحدة نحو اليسار. اي انها تقف عند اول مرتبة من جهة اليمين **least** significant **bit**.
4. اذا كان الرمز 0 غيره الى 1 و توقف.
5. اذا كان الرمز 1 غيره الى 0 وانتقل خطوة الى اليسار
6. كرر الخطوات 4 و 5 حتى تصل الى الرمز # الذي يعني انتهاء العملية والتوقف عندها.

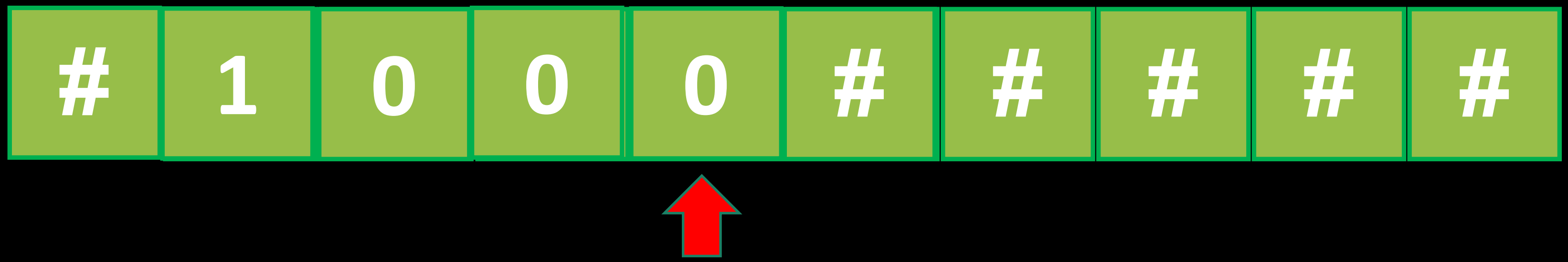


0 1 1 0 # # # #
0 → 0, R
1 → 1 R
q0
→ #, L
q1



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

X = 0111



1 0 0 0 # # # #

eND