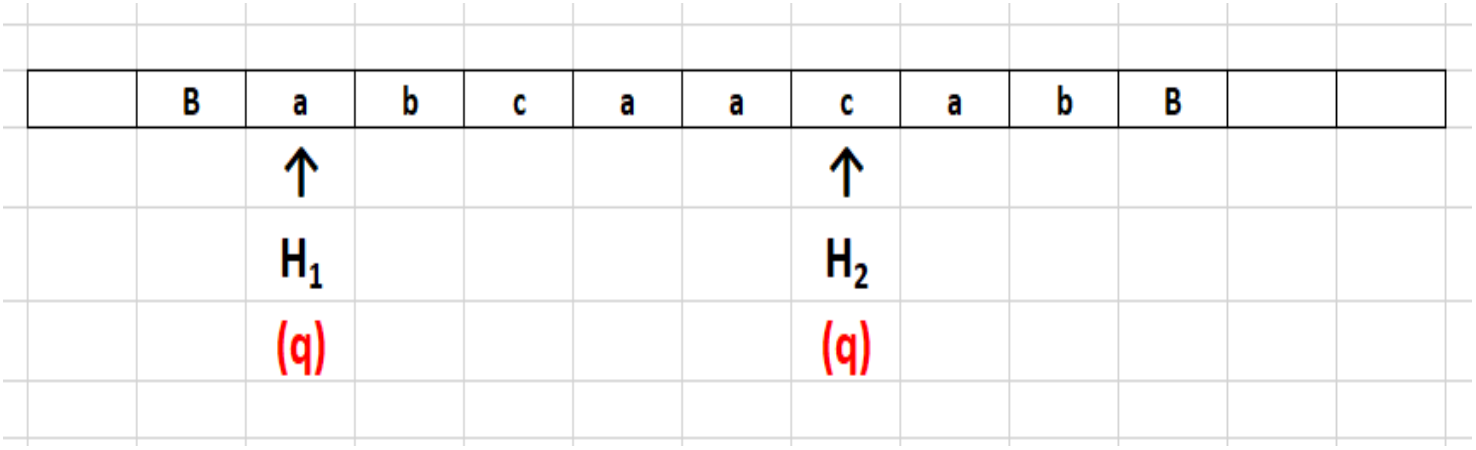
Multi-head Turing Machine

A Turing machine with single tape can have multiple heads.

Let us consider a TM with two heads H_1 and H_2

Each head can capable of performing **read / write / move** operation independently.



The transition behaviour of 2-head one tape Turing machine can be defined as given below:

$$\delta(q, \mathbf{a}, \mathbf{c}) = (p, (\mathbf{X}, R), (\mathbf{Y}, L))$$

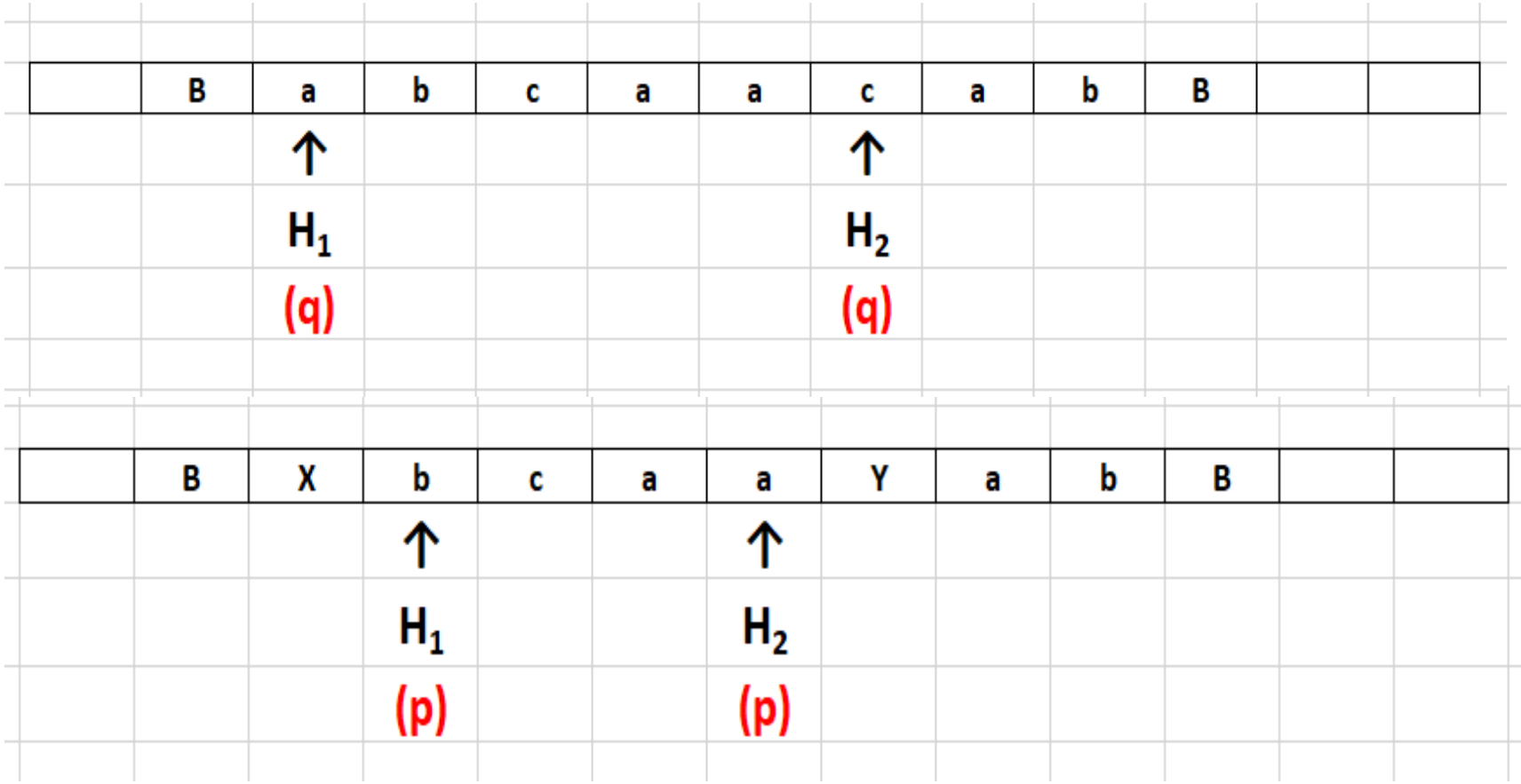
Where

a - symbol under the head $\mathbf{H_1}$

c - symbol under the head $\mathbf{H_2}$

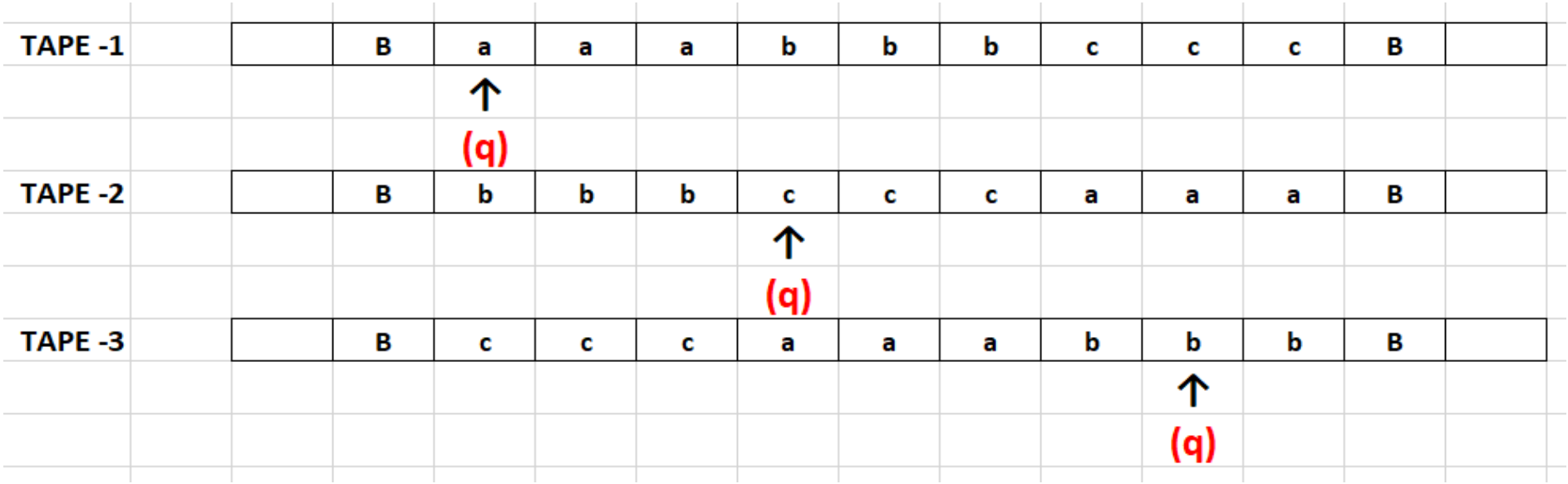
X - symbol to be written in the cell under $\mathbf{H_1}$

Y - symbol to be written in the cell under $\mathbf{H_2}$


B a b c a a c a b B
H1
(q)
H2
(q)
B X b c a a Y a b B
H1
(p)
H2
(p)

Multi-tape Turing Machine

- Multi – tape Turing machine has multiple tapes (more than one but finite number of tapes) with each tape having its own independent head.
- Let us consider the case of three tape TM.



The transition behaviour of 3-tape Turing machine can be defined as given below:

$$\delta(q, \mathbf{a}, \mathbf{c}, \mathbf{b}) = (p, (\mathbf{X}, R), (\mathbf{Y}, L), (\mathbf{Z}, R))$$

Where

q - current state

p - next state

a - symbol under the head on tape 1

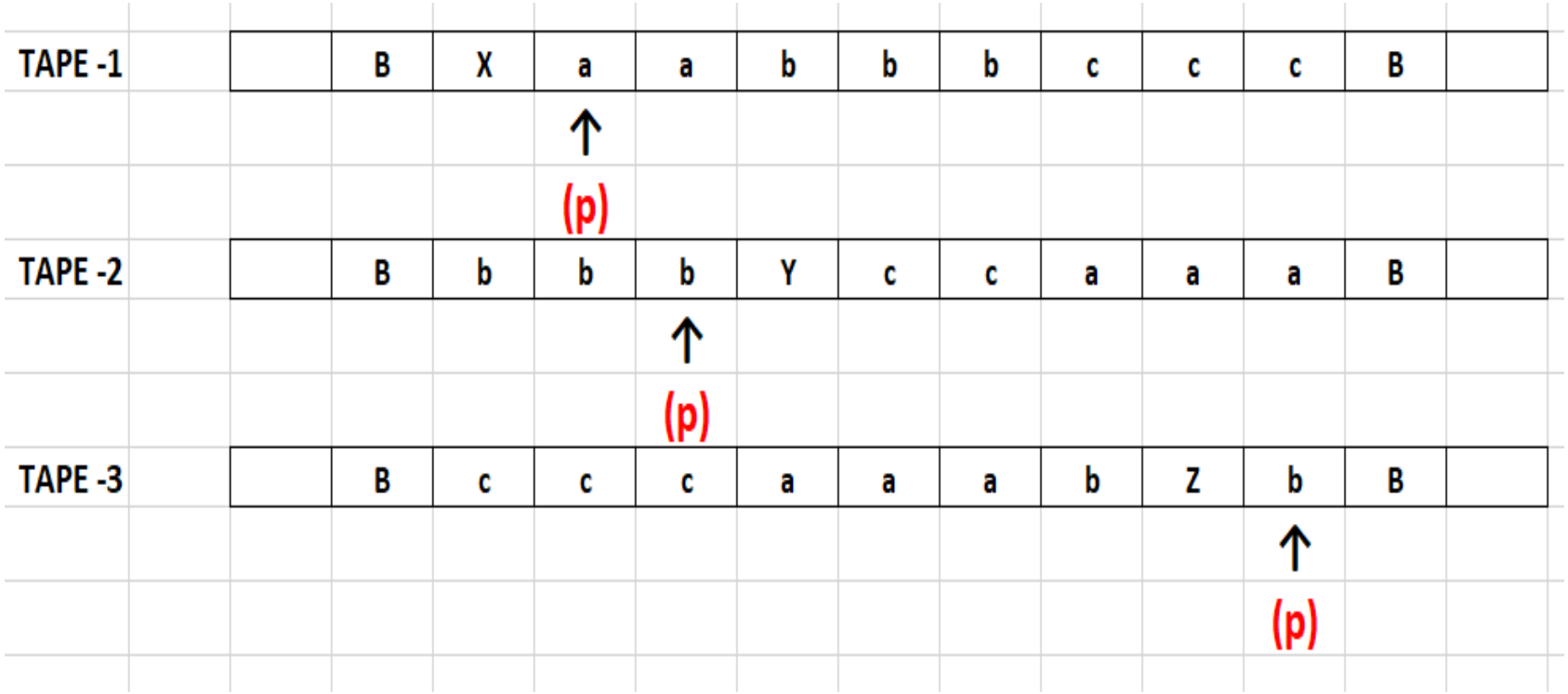
c - symbol under the head on tape 2

b - symbol under the head on tape 3

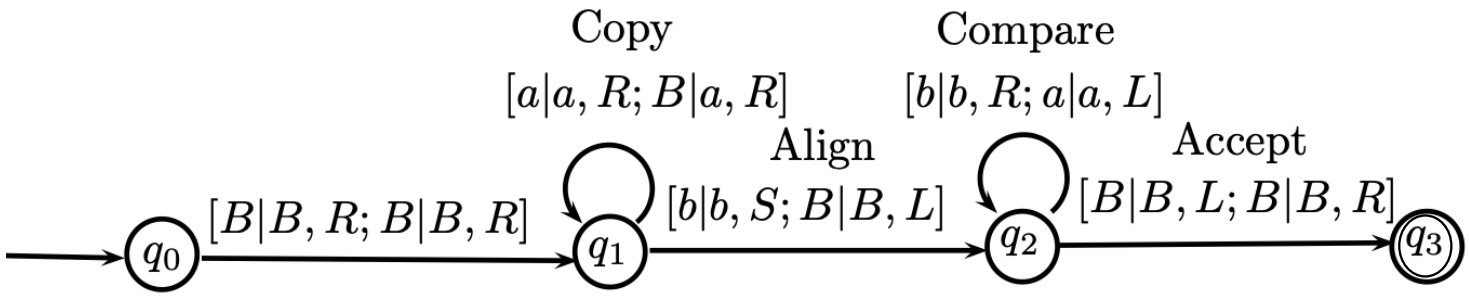
X - symbol to be written in the cell on tape 1

Y - symbol to be written in the cell on tape 2

Z - symbol to be written in the cell on tape 3

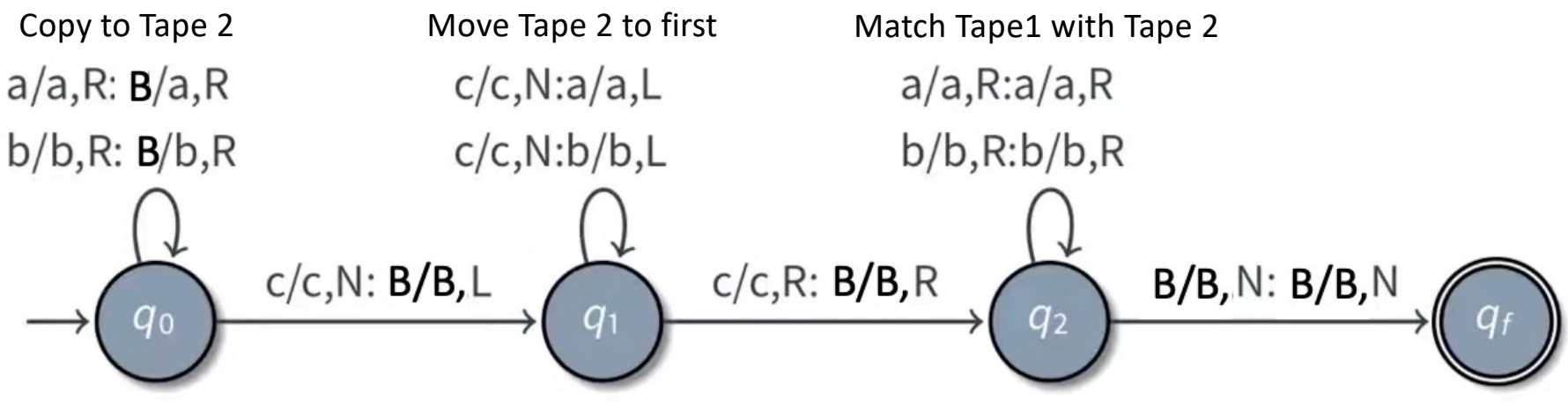

TAPE -1
B X a a b b b c c c B
(p)
TAPE -2
B b b b Y c c a a a B
(p)
TAPE -3
B c c c a a a b Z b B
(p)

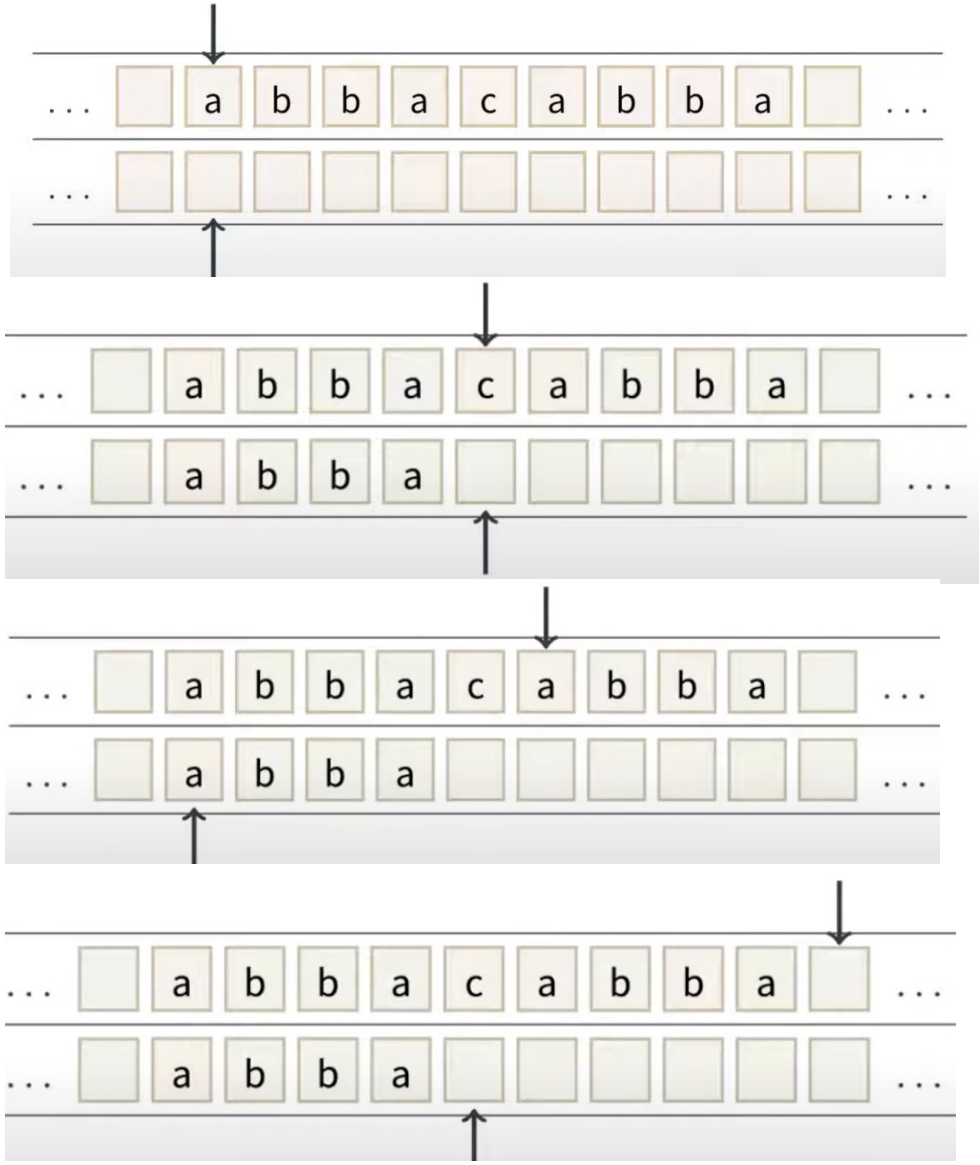
Construct a Two-tape TM to recognize language L = {$a^n b^n$ | n ≥ 0}



1. *Copy:* Copies the string of a's on tape-1 to tape-2.
2. *Align:* Moves head 1 to begin of b's on tape-1, and head-2 on last a on tape-2.
3. *Compare:* Compares number of b's tape-1 with number of a's on tape-2 by moving heads in opposite directions.

L = {wcw |w belongs to {a, b}





L = {wcw |w belongs to {a, b}

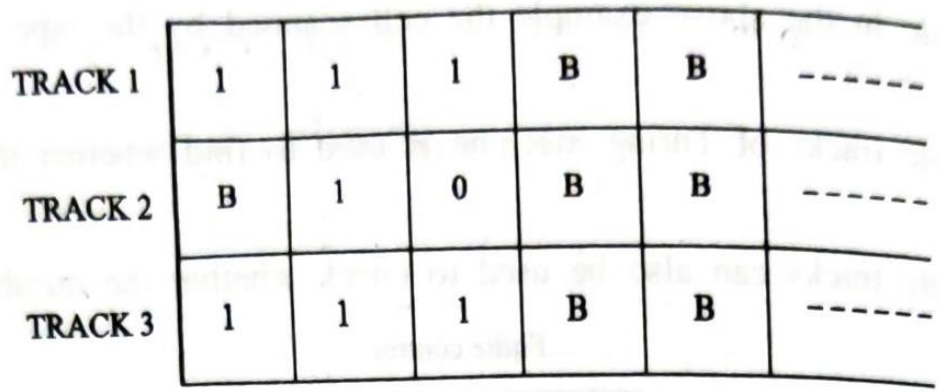
Problems

1. **Design a Turing Machine using multiple tracks to check whether the given input number is prime or not.**

Solution:

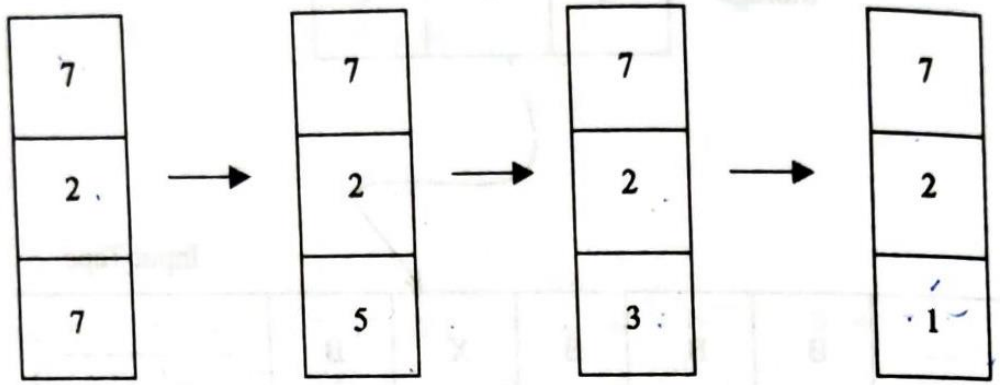
Let us use multiple track of Turing machine to find that the input number is a prime number or not. The idea to design this Turing machine is that let us store the input symbol in the first track of input tape. Let us store the number 2 is binary in the second track of input tape. Let us copy the input in the third track also. All the symbols in the three tracks of the Turing machine are in binary form. Now subtract the second track from the third track until we get '0' or any remainder.

- If the remainder is zero, then the number is not prime, since the prime number is one which is divided by 1 and itself.
- If the remainder is non zero value, then the second track value is incremented by 1 and again the subtraction procedure is continued.
- If the value of the second and first track is equal, then the number is prime number. Let us take an input value 7 and it is stored as,

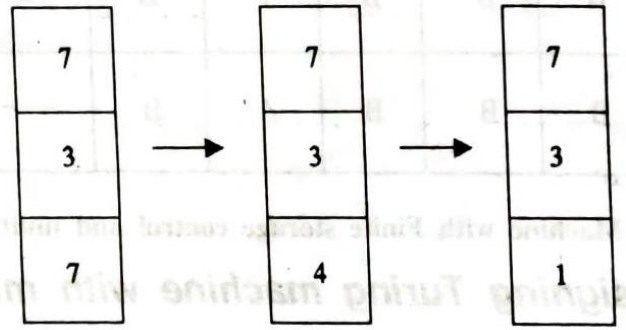


Now process is as follows,

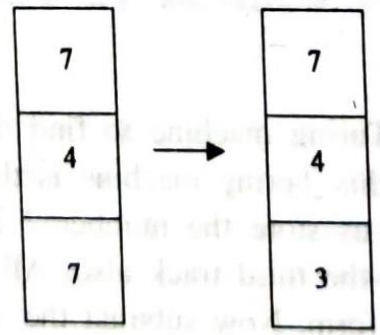
Subtracting 2 from 7, we get



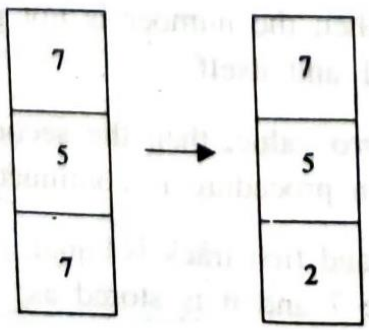
The Remainder is 1, so increment the value of second track by 1



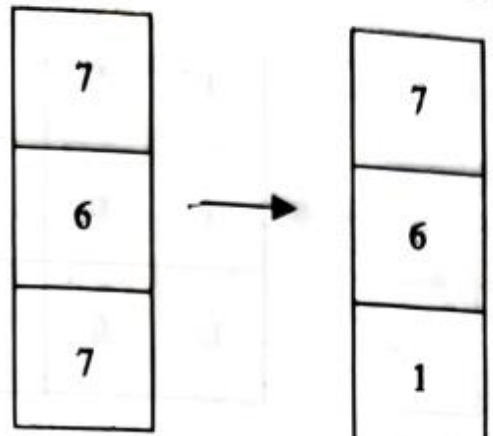
The Remainder is 1, so increment the value of second track by 1



The Remainder is 3, so increment the value of second track by 1



The Remainder is 2, so increment the value of second track by 1



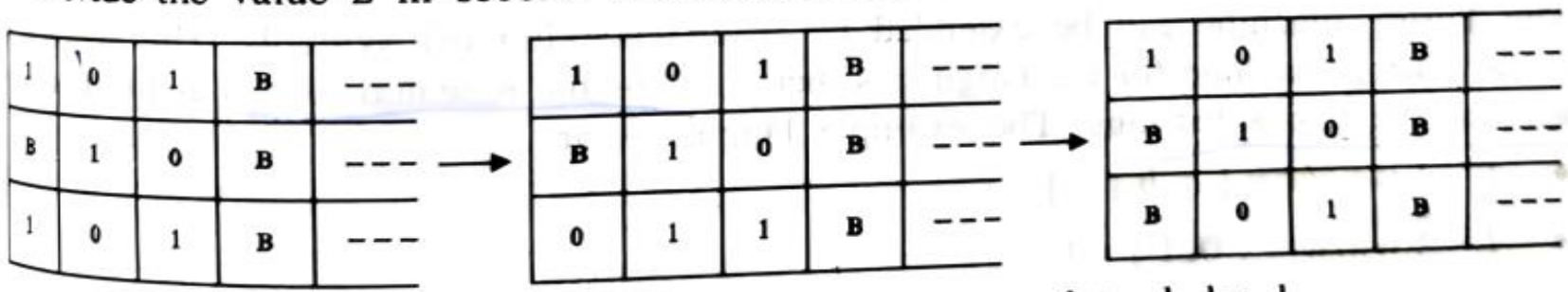
The Remainder is 1, so increment the value of second track by 1

Now the value of first and second track is equal, so the number 7 is a prime number.

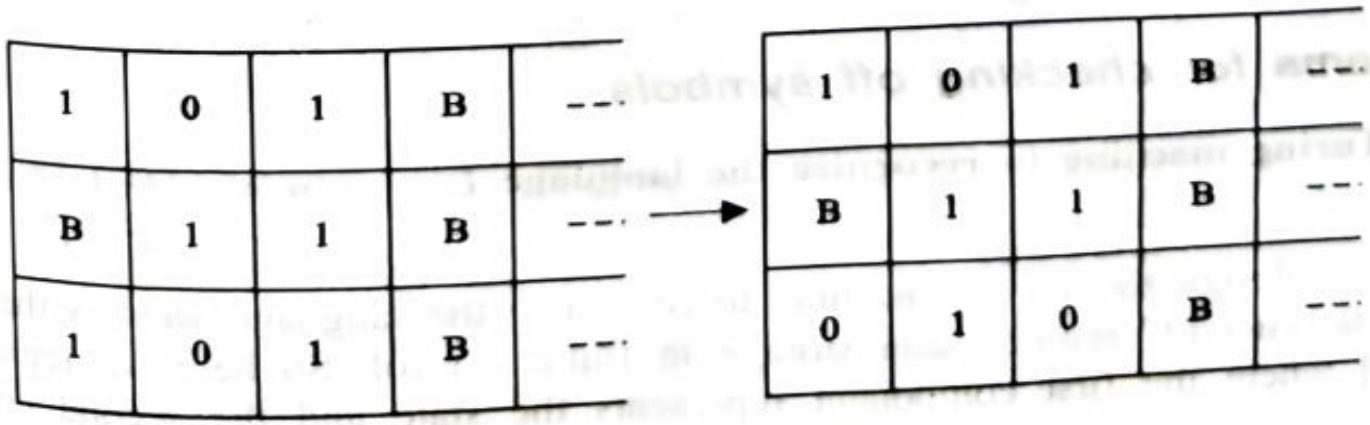
Input string = 5

1	0	1	B	----
B	1	0	B	----
1	0	1	B	----

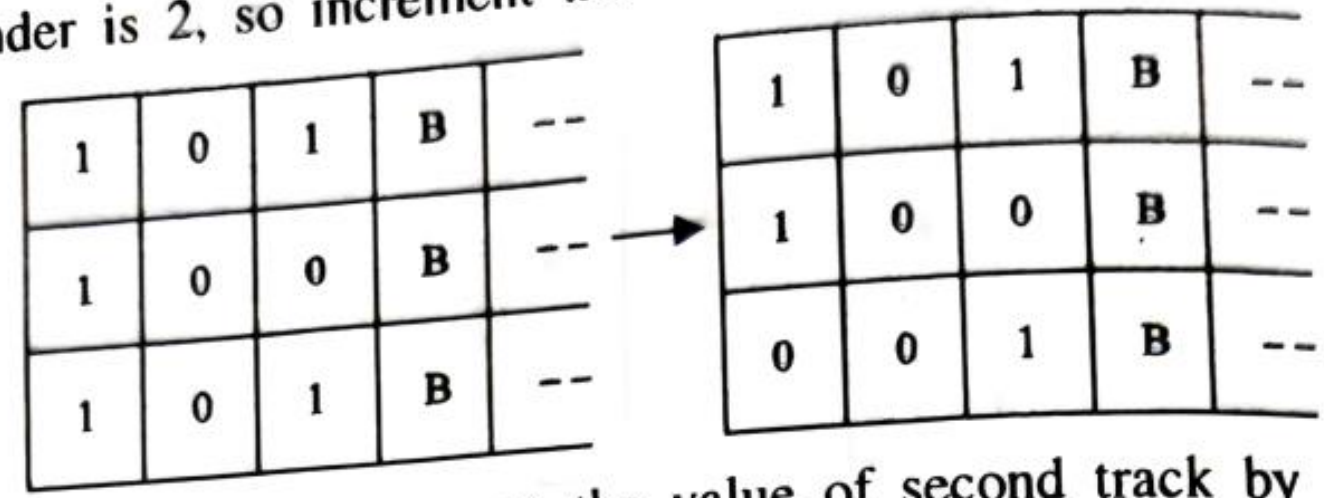
Divide the value 2 in second track from value 5 in the third track.



The Remainder is 1, so increment the value of second track by 1



The Remainder is 2, so increment the value of second track by 1



1	0	1	B	--
1	0	0	B	--
1	0	1	B	--

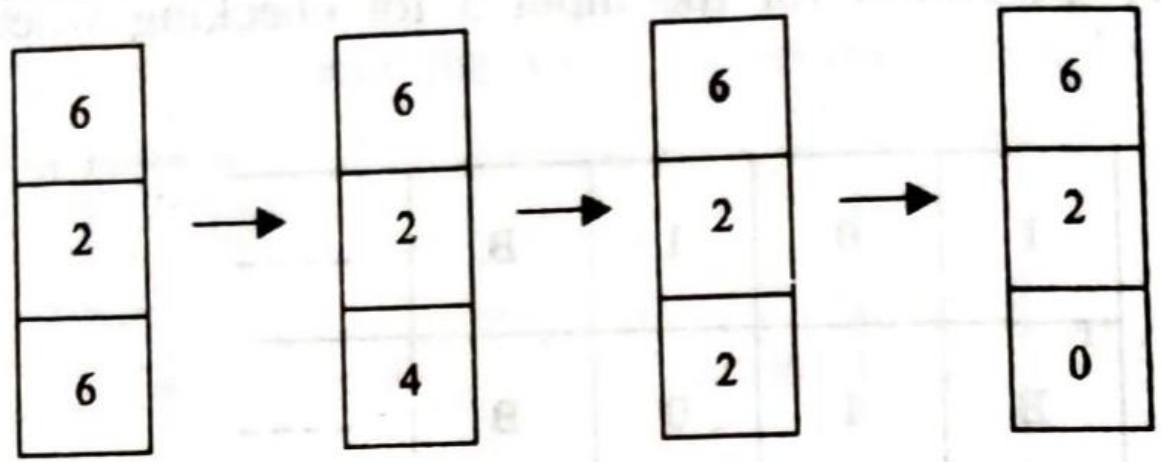
1	0	1	B	--
1	0	0	B	--
0	0	1	B	--

The Remainder is 1, so increment the value of second track by 1

1	0	1	B	--
1	0	1	B	--
1	0	1	B	--

Now the value of first and second track is equal, so the number 5 is a prime number.

Now let us take the number as 6.

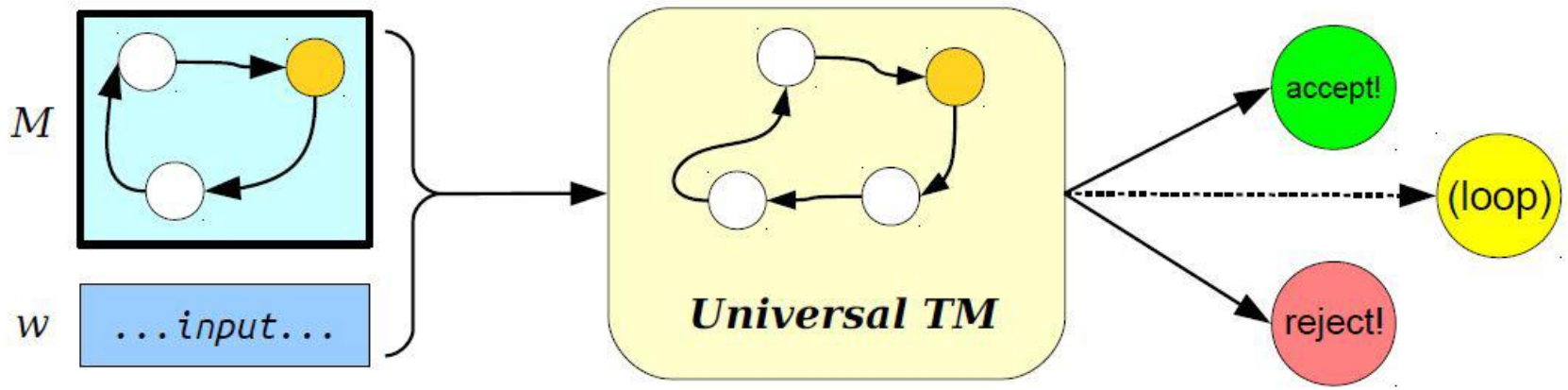


On dividing 2 from 6, we get 4 then by subtracting 4 by 2, we get 2 and again by subtracting 2 by 2 we get 0. Since the remainder is 0, the number 6 is not a prime number

The Universal Turing Machine (UTM)

- A single computing machine can capture / compute any computable function.
- A UTM, M_u simulate the computation of any TM, M on a given input, w

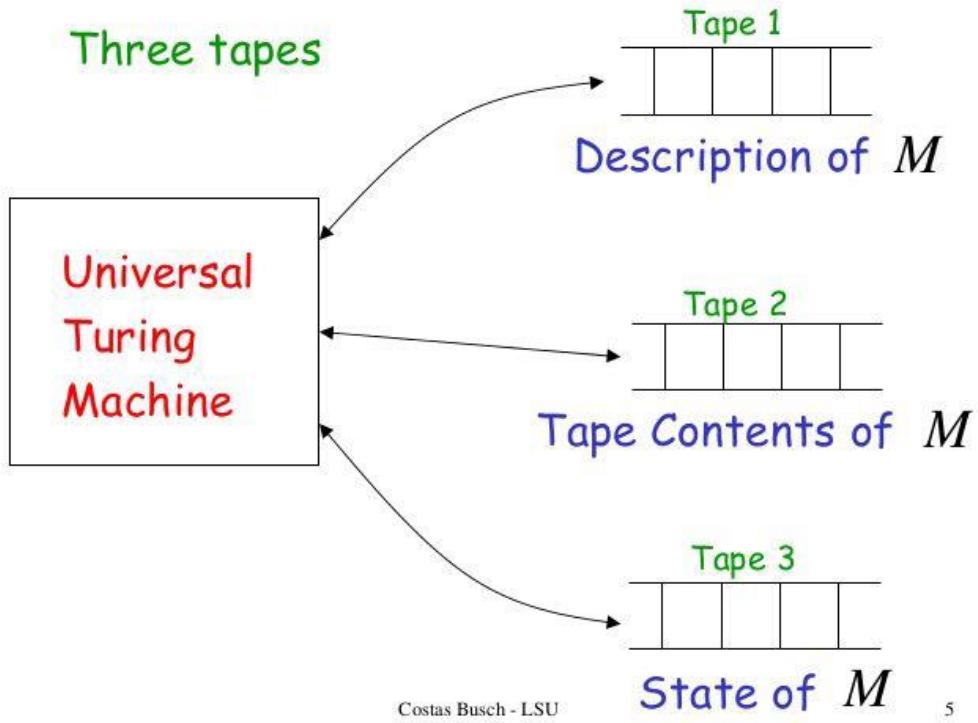
- The observable behavior of U_{TM} is the following:
  - If M accepts w, then U_{TM} accepts $\langle M, w \rangle$.
  - If M rejects w, then U_{TM} rejects $\langle M, w \rangle$.
  - If M loops on w, then U_{TM} loops on $\langle M, w \rangle$.
- **U_{TM} accepts $\langle M, w \rangle$ if and only if M accepts w.**



Cont…

- M_u is a three tape machine (3 read/write heads)
  - Tape 1: Transitions of TM + Input string
  - Tape 2: Compiled program tape contains encoding of the TM, M
  - Tape 3: Working memory tape, contains current state of TM, M being simulated

Encode transition of M as a binary string



Working of M_u

1. Look at current state in working memory and current symbol on input-output tape
2. With information of current state and current symbol, scan the compiled program tape to find a transition for the given configuration.
3. Execute the transition i.e. write new symbol on input-output tape and move its read/write head
4. Repeat above steps until there are no more transitions in compiled program
5. It halts and accepts (OR) halts and rejects.

Cont…

- Turing's idea of storing an encoded table of instructions (i.e. program) of a TM on one of the tapes of UTM, made it possible for us to build modern computers that can compute practically anything that is computable.

- Also, Turing machines for simple computable functions can be combined to compute more complex functions.
  - Can easily concatenate encoded programs of multiple Turing machines to create a program for a complex Turing machine.
  - Similar to modularity in programming