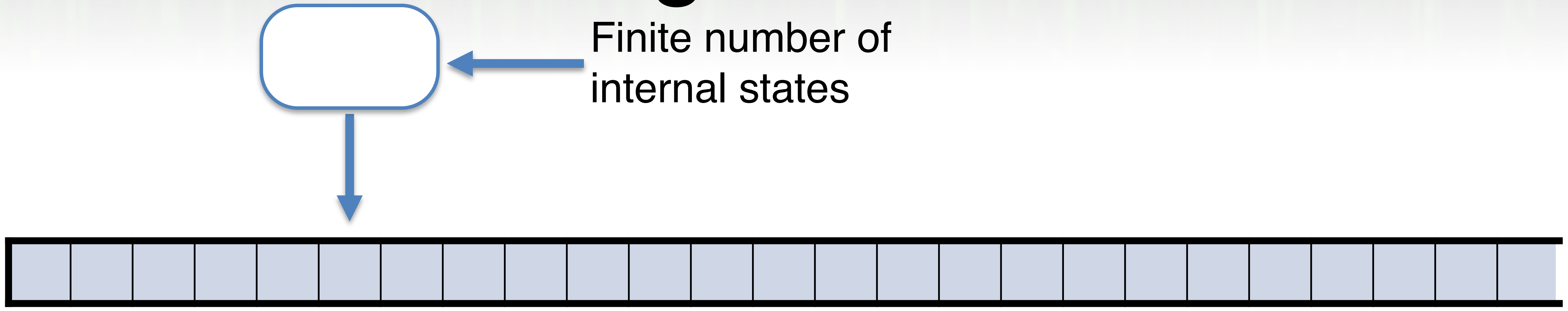
Turing Machines, contd.

Lecture 10

Turing Machine



Finite alphabet

Read

Write

Move +1 or -1

Halt condition

Internal state (finite number)

TM for Decision Problems

$M = (Q, \Sigma, \Gamma, B, \delta, q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}})$:

Γ is a finite tape alphabet.

- B *or* □ is the blank symbol (special symbol)

- Σ is a finite input alphabet $\Sigma \subseteq \Gamma \setminus B$

Q is a finite set of states

$q_{\text{start}} \in Q$ is the initial state

$q_{\text{accept}}, q_{\text{reject}} \in Q$ accept/reject states

Or maybe run forever

Transition function: $\delta : Q \times \Gamma \text{ (read)} \rightarrow Q \times \Gamma \text{ (write)} \times \{ L, R \}$

TMs: what we saw and will see

- They are quite tedious to program, but possible! (it’s the assembly language version)
- They can do anything a computer can do (copy, shift, add…)
- e.g. RAM

TMs: what we saw and will see

- Will see that a TM can simulate itself. Write a TM interpreter in TM!
- Universal TM.

TMs: what we saw and will see

- **Church-Turing Thesis:**

“Any physically realizable model of computation is equivalent to a TM”

- More of a physical law than a math theorem.
- e.g. Python doesn’t have additional power over TM.
- sounds fancy but it says no more than “a Python interpreter can compute anything you can compute in Python”

TMs: what we saw and will see

- **Church-Turing Thesis:**

“Any physically realizable model of computation is equivalent to a TM”

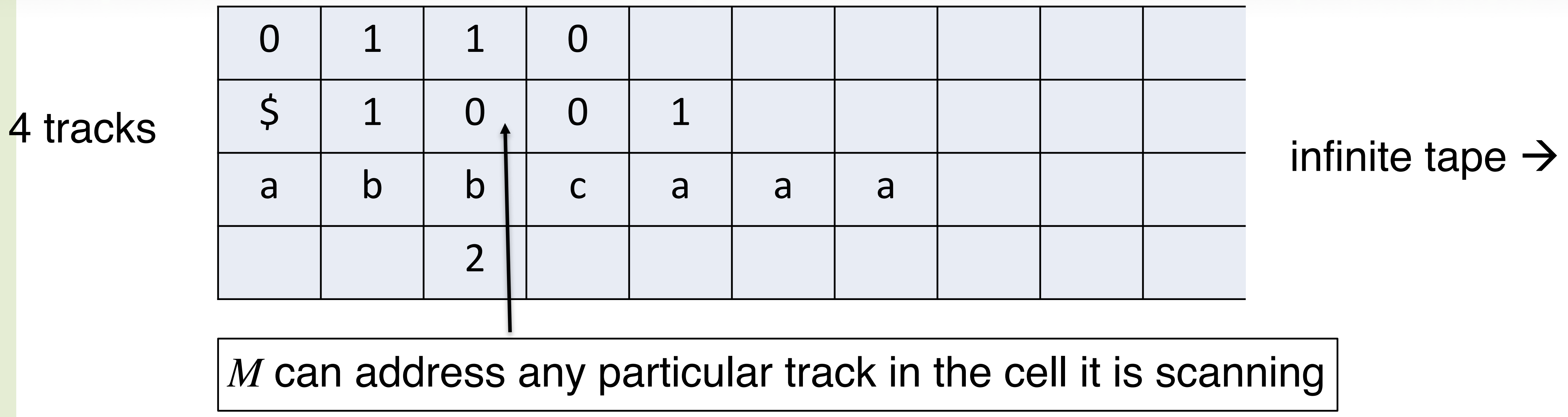
- There are models of computation not equivalent to TM, we won’t see them this semester.

Variants/Extensions

Adding more capabilities to TMs make them easier to program

But doesn't change what TMs can do:
whatever the new variant can do, can be simulated in the original variant
(with a lot more steps, sometimes)

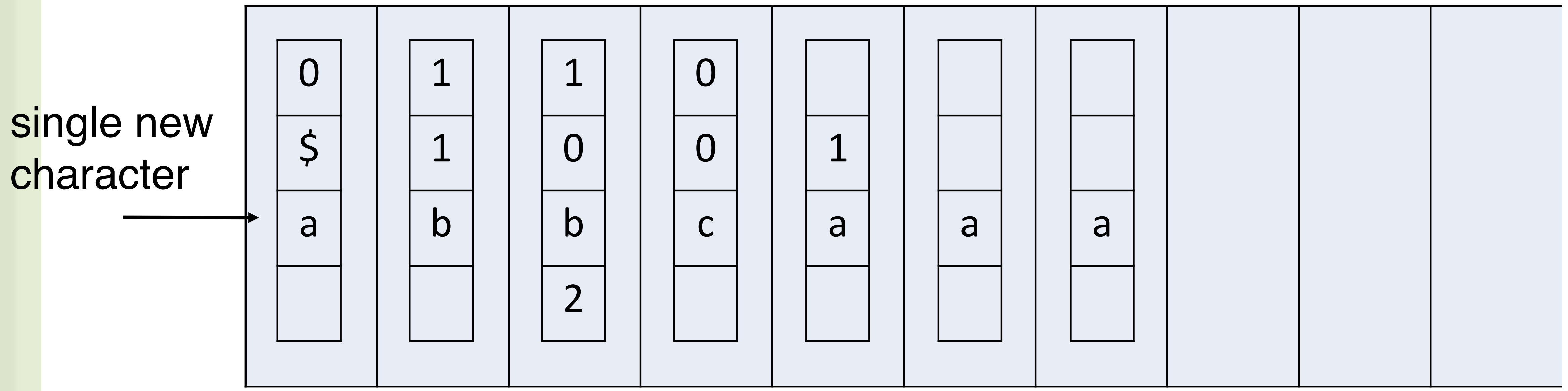
Extension: multiple tracks



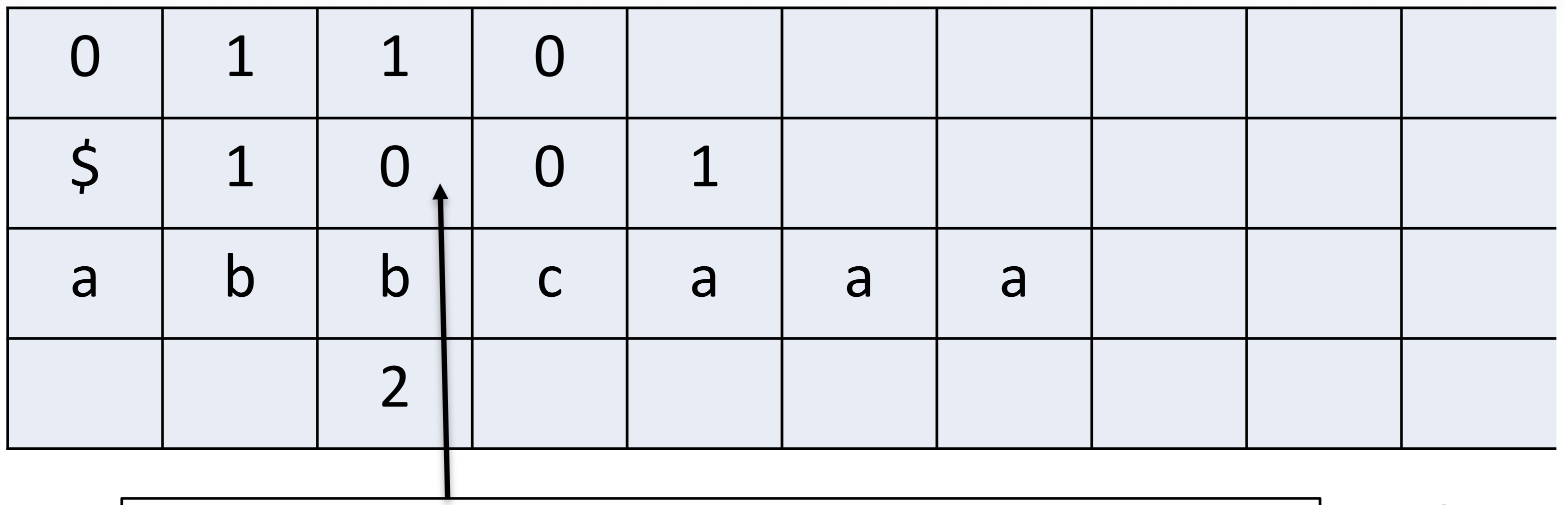
0	1	1	0						
$	1	0	0	1					
a	b	b	c	a	a	a			
		2							

M can address any particular track in the cell it is scanning

Can simulate multiple tracks with a single track machine, using extra "stacked" characters:



Extension: multiple tracks



4 tracks

0	1	1	0						
$	1	0	0	1					
a	b	b	c	a	a	a			
		2							

infinite tape →

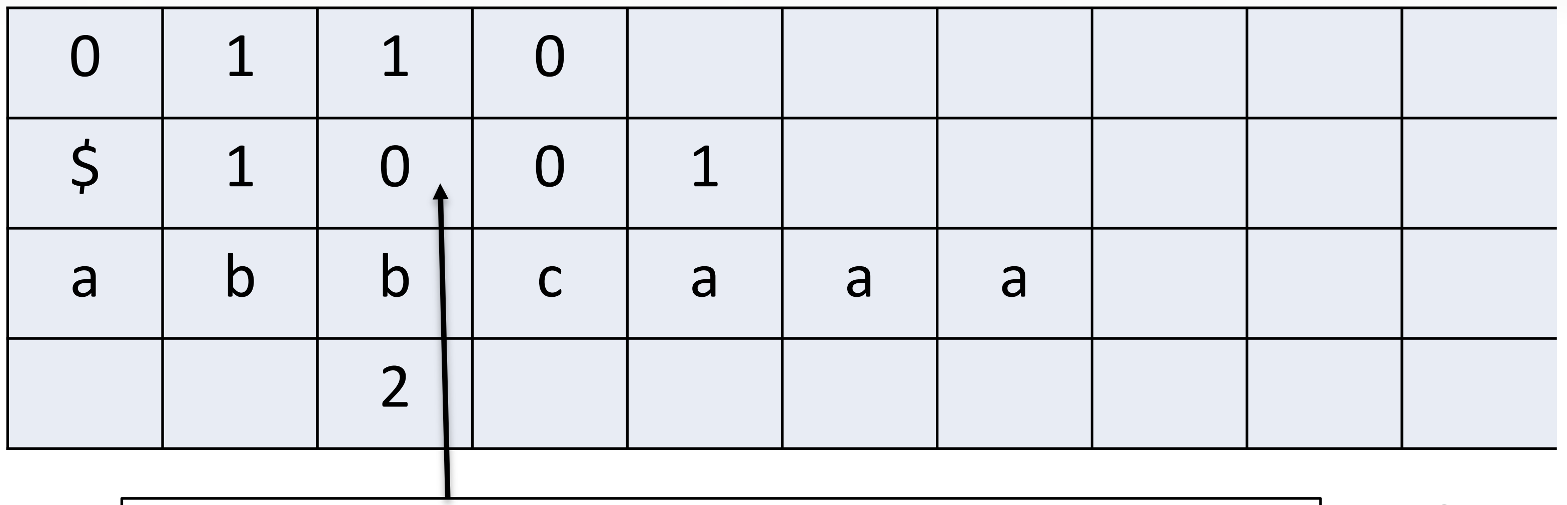
$$M: \delta(q, \text{-}, 0, \text{-}, \text{-}) = (p, \text{-}, \text{-}, \text{-}, 1, R)$$

"If in state *q* reading 0 on second track, then go to state *p*, write 1 on fourth track, and move right"

Then in M' $\delta\left(q, \begin{bmatrix} x \\ 0 \\ y \\ z \end{bmatrix}\right) = \left(p, \begin{bmatrix} x \\ 0 \\ y \\ 1 \end{bmatrix} R\right)$ for every $x, y, z \in \Gamma$

Extension: multiple tracks

4 tracks



0	1	1	0						
\$	1	0	0	1					
a	b	b	c	a	a	a			
		2							

infinite tape →

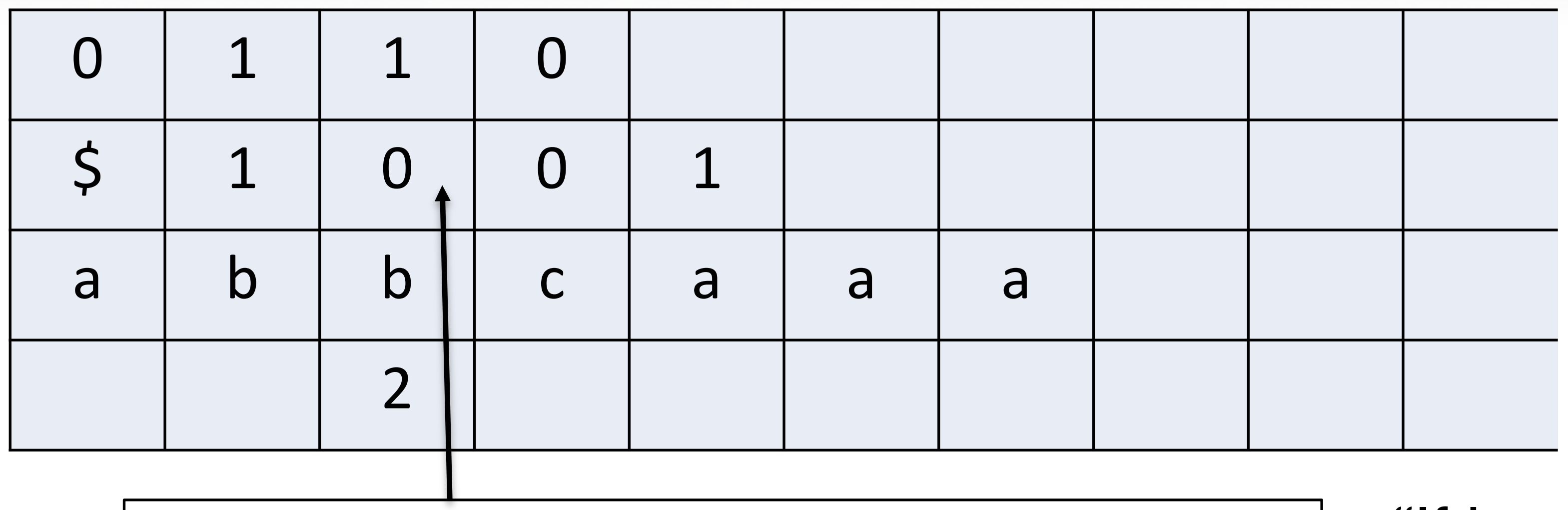
$$\boxed{M\text{: } \delta(q, \text{-},0,\text{-},\text{-}) = (p, \text{-},\text{-},\text{-},1, \mathrm{R})}$$

"If in state *q* reading 0 on second track, then go to state *p*, write 1 on fourth track, and move right"

Transition function:

$$\delta : Q \times \Gamma_1 \times \Gamma_2 \times \Gamma_3 \times \Gamma_4 \to Q \times \Gamma_1 \times \Gamma_2 \times \Gamma_3 \times \Gamma_4 \times \{\, \mathrm{L}, \mathrm{R} \,\}$$

Extension: multiple tracks



4 tracks

0	1	1	0						
$	1	0	0	1					
a	b	b	c	a	a	a			
		2							

infinite tape →

M: $\delta(q, \text{-},0,\text{-},\text{-}) = (p, \text{-},\text{-},\text{-},1, \mathrm{R})$

"If in state *q* reading 0 on second track, then go to state *p*, write 1 on fourth track, and move right"

Transition function:

$$\delta : Q \times (\Gamma_1 \times \Gamma_2 \times \Gamma_3 \times \Gamma_4) \rightarrow Q \times (\Gamma_1 \times \Gamma_2 \times \Gamma_3 \times \Gamma_4) \times \{ \mathrm{L}, \mathrm{R}$$

Extension: multiple tracks

Sometimes intuitively better with multiple tracks
e.g assume I want to copy this string.

0	1	1	0	1	0				
$	$			.					

0	1	1	0	1	0	1			
$	$	$		.					

0	1	1	0	1	0	1	1		
$	$	$	$	.					

0	1	1	0	1	0	1	1	0	
$	$	$	$	$					

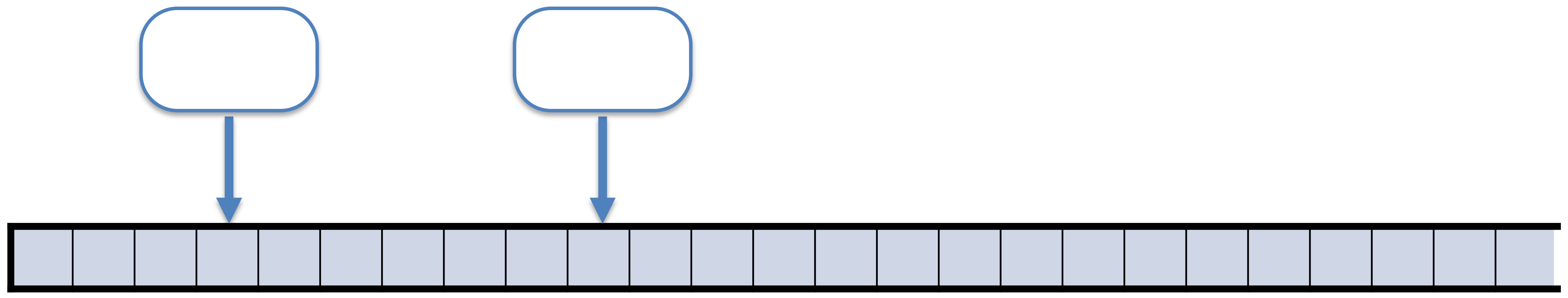
Extension: multiple tracks

Sometimes intuitively better with multiple tracks

e.g assume I want to copy this string.

0	1	1	0	1	0	1	1	0	1
$	$	$	$	$					

Extension: multiple heads

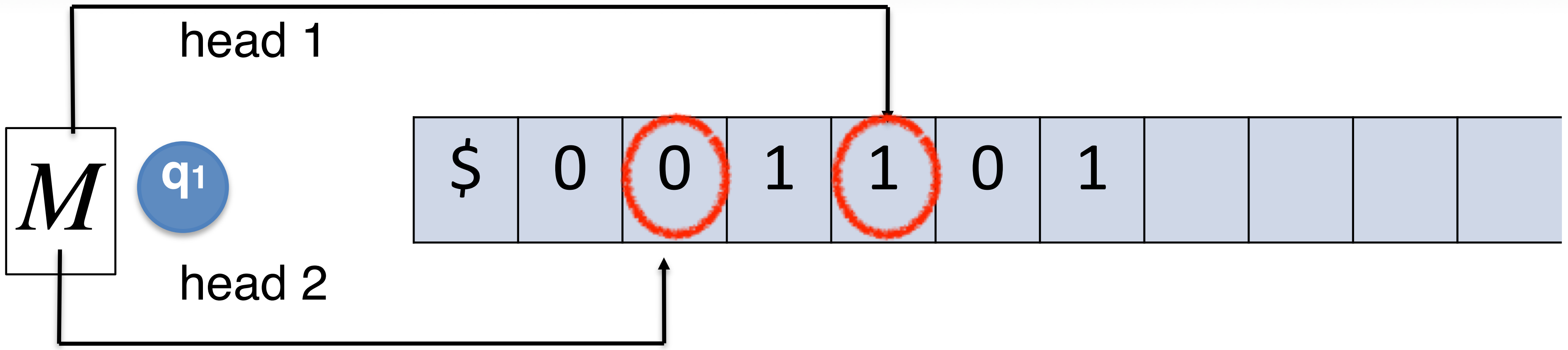


Transition function:

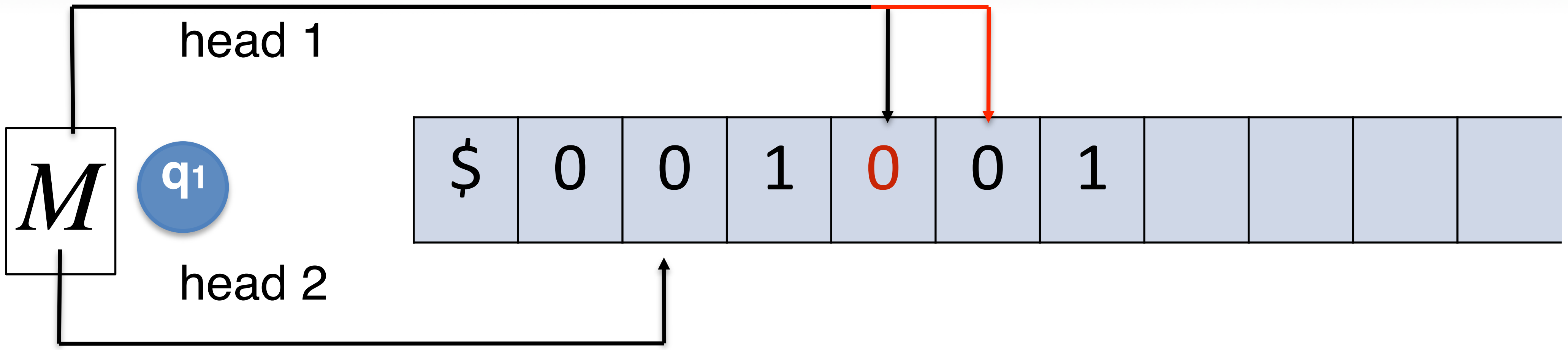
$$\delta : Q \times \Gamma^2 \rightarrow Q \times \Gamma^2 \times \{\mathrm{L}, \mathrm{R}\}^2$$

Snapshot of simulation (2 heads)

Single move:
$\delta(q_1,1,0)$
$= (q_2,0,0,R,L)$



Snapshot of simulation (2 heads)



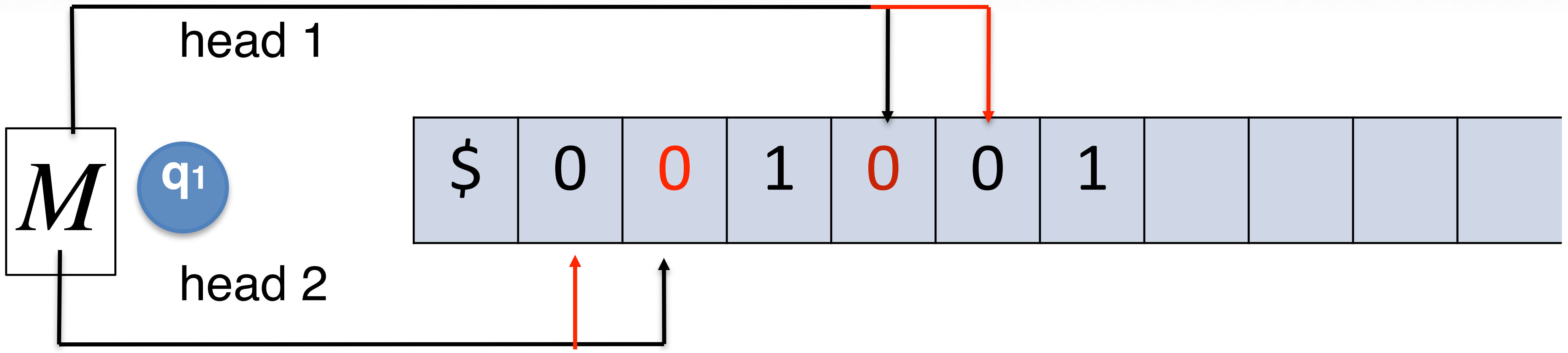
Single move:

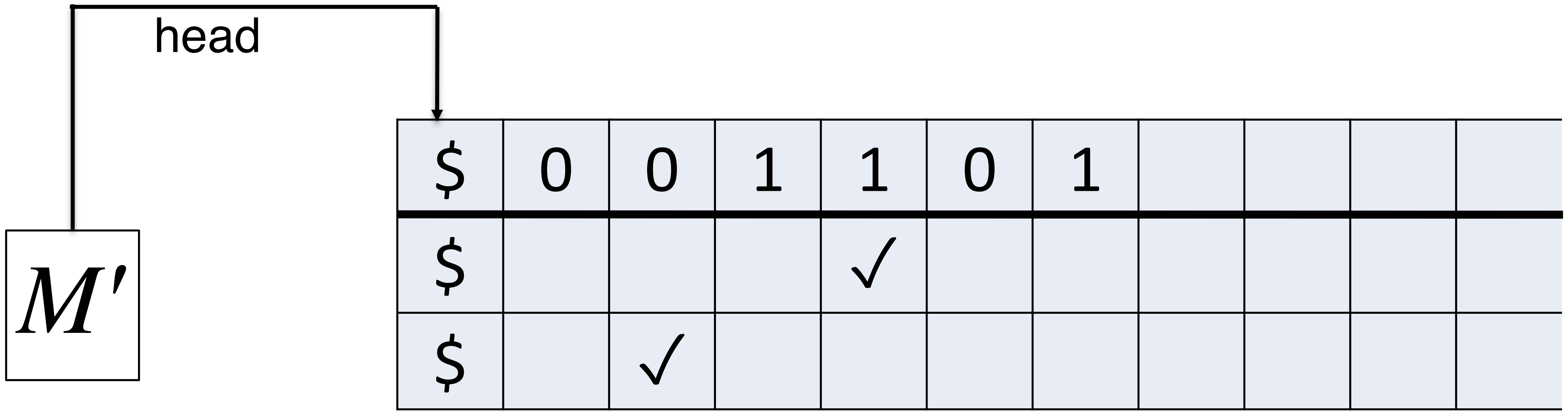
$\delta(q_1,1,0)$

$= (q_2,0,0,R,L)$

Snapshot of simulation (2 heads)

Single move: $\delta(q_1,1,0) = (q_2,0,0,R,L)$



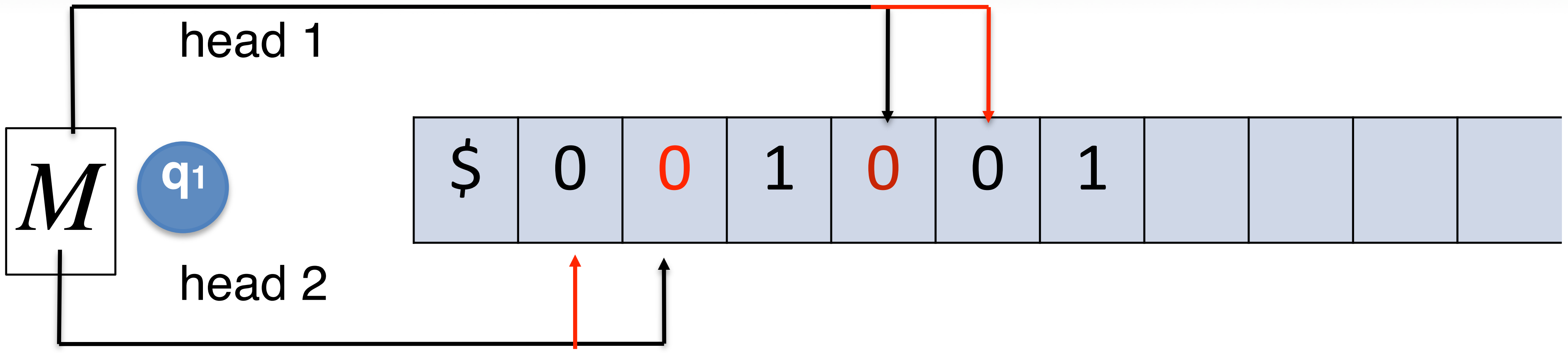


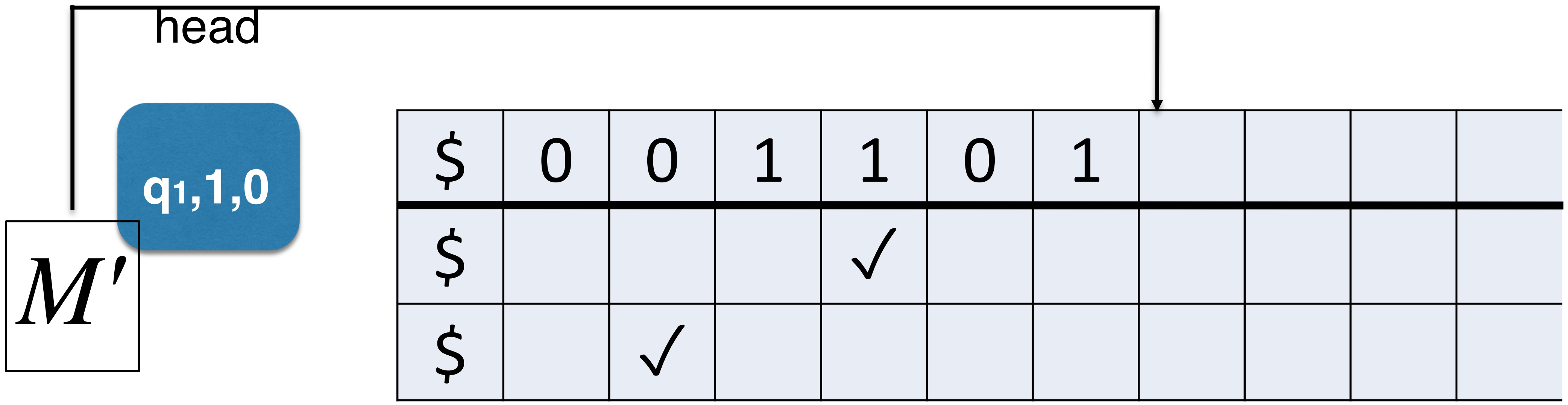
\$	0	0	1	1	0	1				
\$				✓						
\$		✓								

- Simulate with multiple tracks. Special mark on track 1 and 2 for head positions. Track 0 has input.
- Make sweeps over the entire tape

Snapshot of simulation (2 heads)

Single move:
$\delta(q_1,1,0)$
$= (q_2,0,0,R,L)$

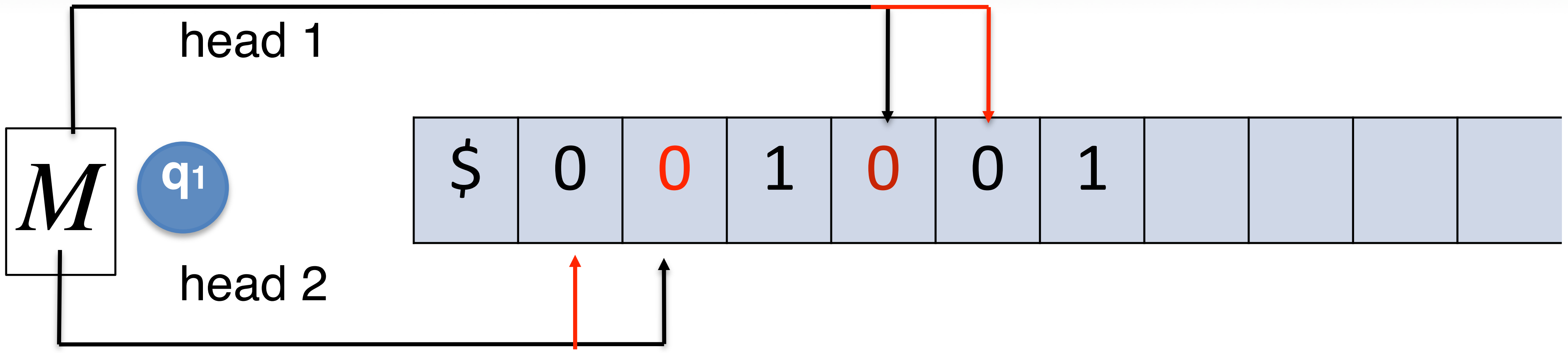


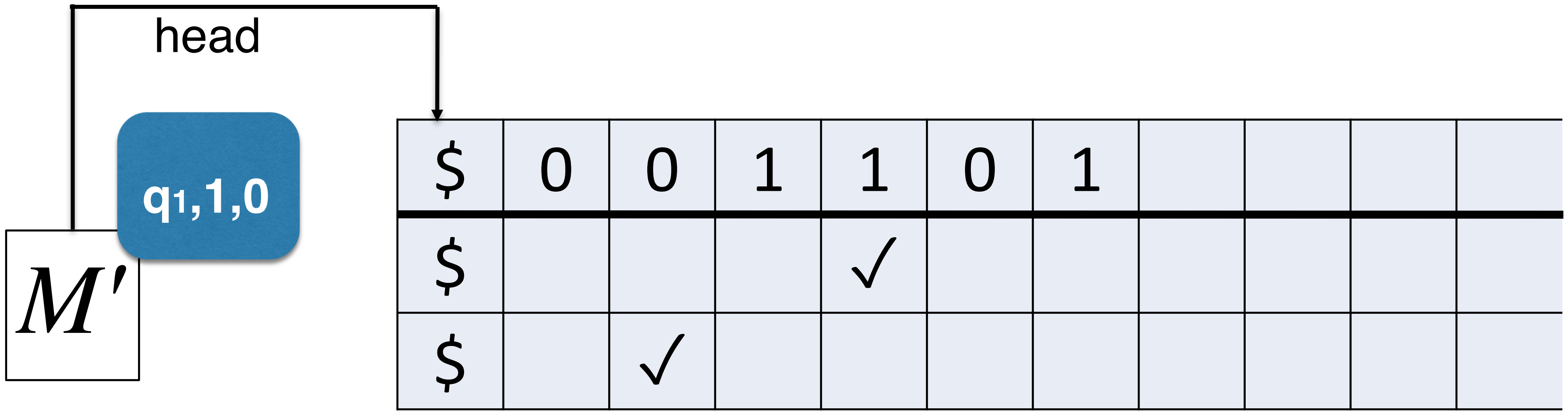


1) Scan to the right to find the mark on track I
read the corresponding symbol from track 0 into our internal state,
and then return to the left end of the tape.

Snapshot of simulation (2 heads)

Single move:
$\delta(q_1,1,0)$
$= (q_2,0,0,R,L)$

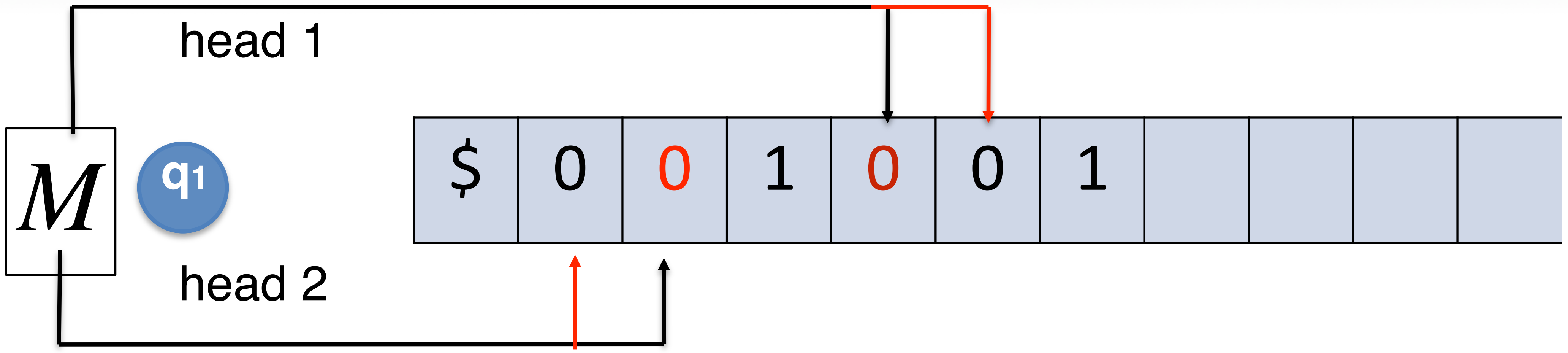


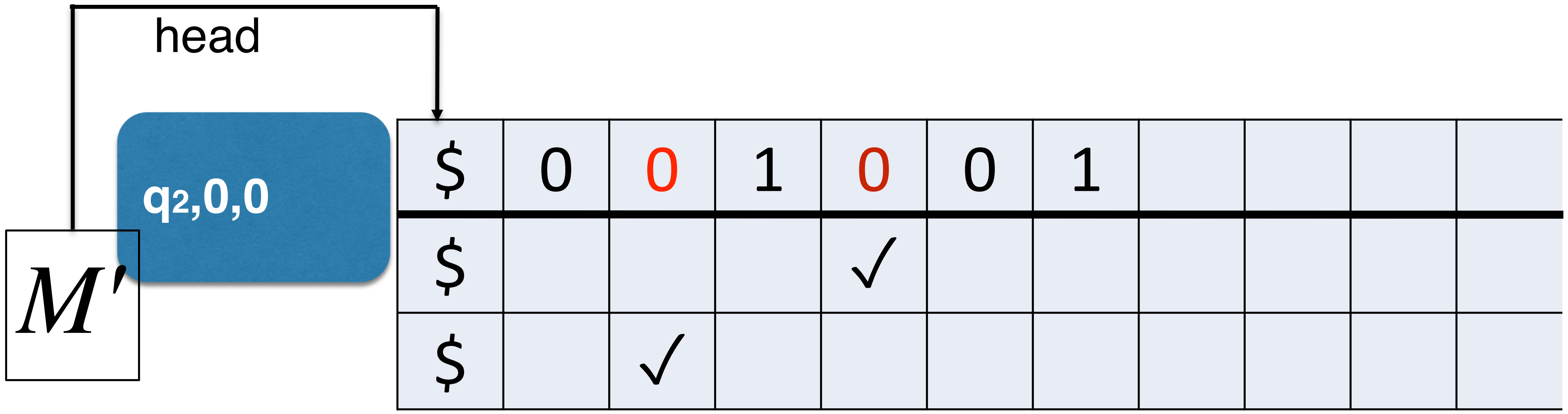


2) Using M's transition function, the internal state records M's next state, the symbol to be written by each head, and the direction to move each head.

Snapshot of simulation (2 heads)

Single move:
$\delta(q_1,1,0)$
$= (q_2,0,0,R,L)$

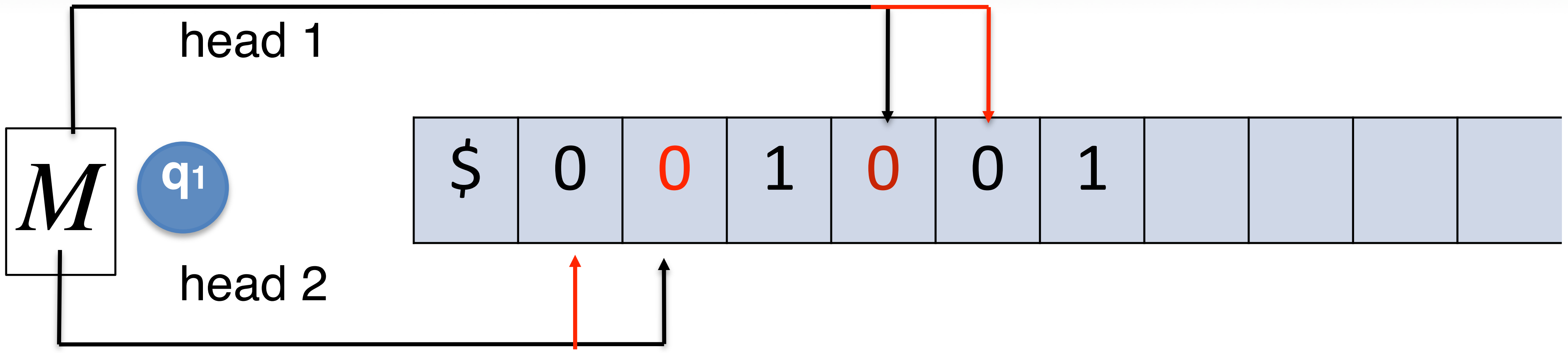


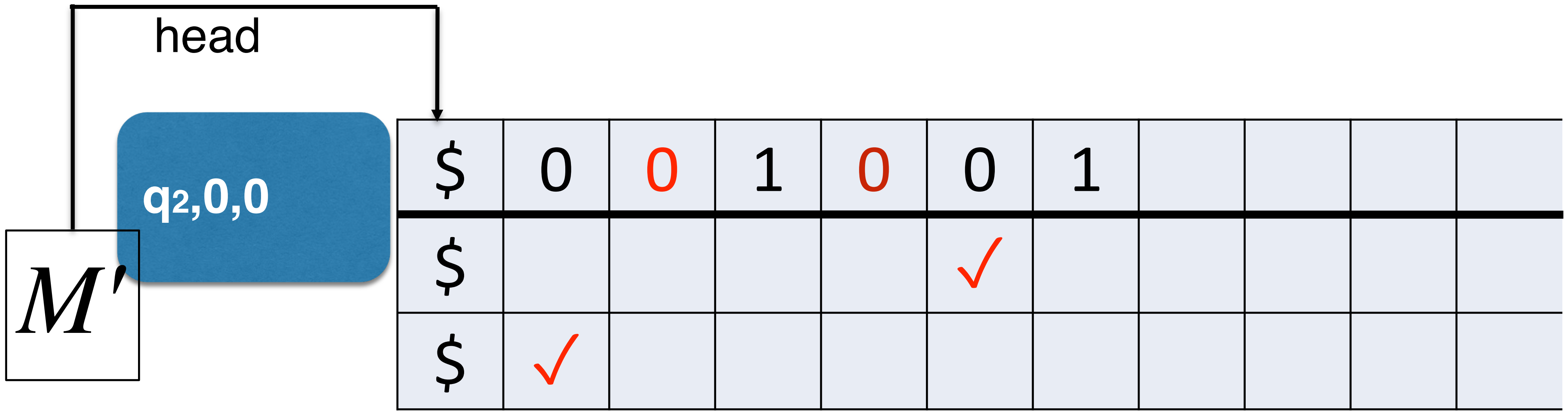


3) Scan to the right to find the mark on track i, write the correct symbol onto on track 0, move the mark on track i one step left or right, and then return to the left end of the tape.

Snapshot of simulation (2 heads)

Single move:
$\delta(q_1,1,0)$
$= (q_2,0,0,R,L)$

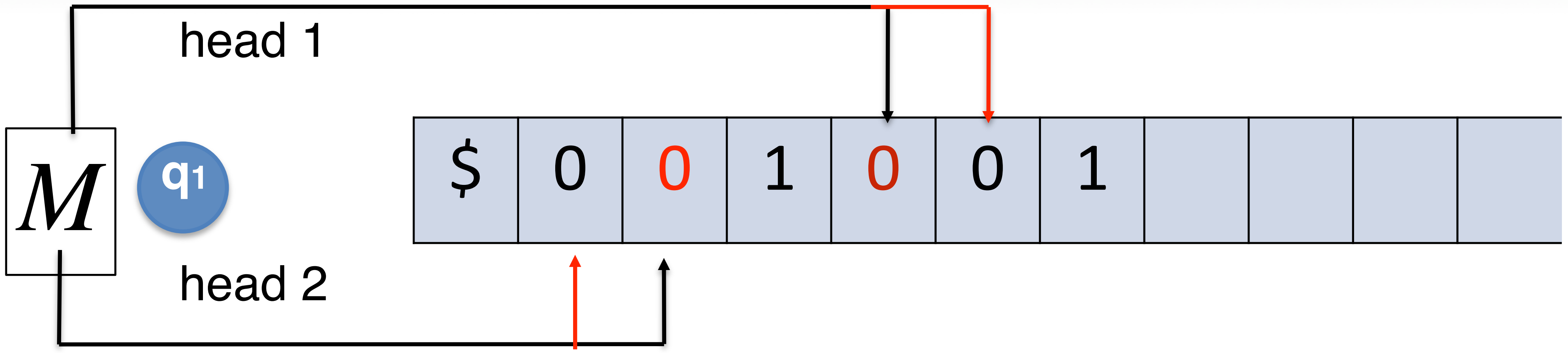


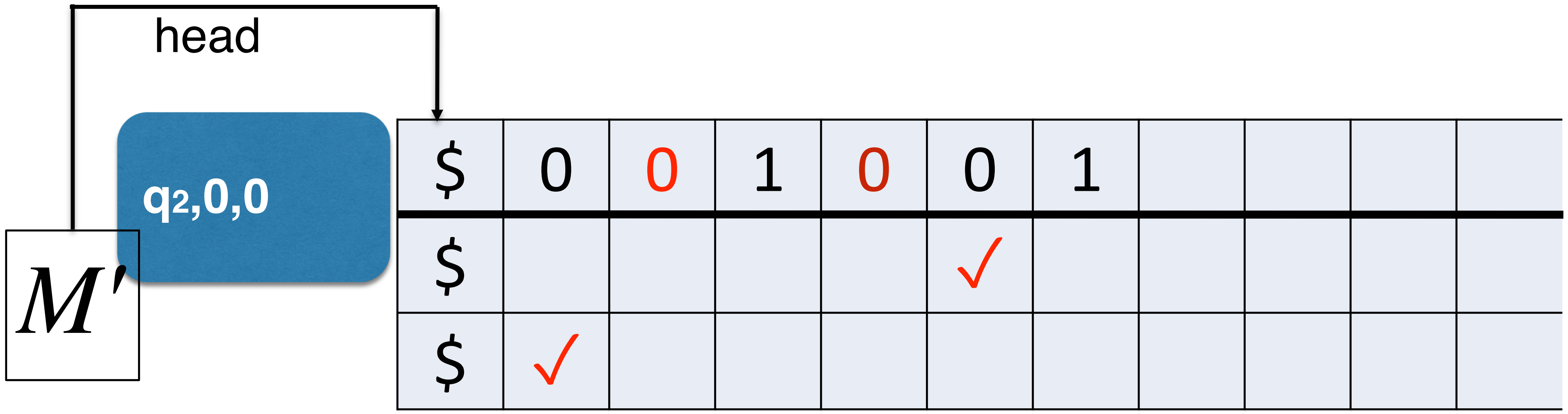


3) Scan to the right to find the mark on track i, write the correct symbol onto on track 0, move the mark on track i one step left or right, and then return to the left end of the tape.

Snapshot of simulation (2 heads)

Single move:
$\delta(q_1,1,0)$
$= (q_2,0,0,R,L)$





- Subroutine!
- However, seriously slows down the process but we don’t care about running time right now

Extension: multiple tapes

k-tape TM

k different (2-way infinite) tapes

k different independently controllable heads

input initially on tape 1; tapes 2, 3, ..., k, blank.

Single move:

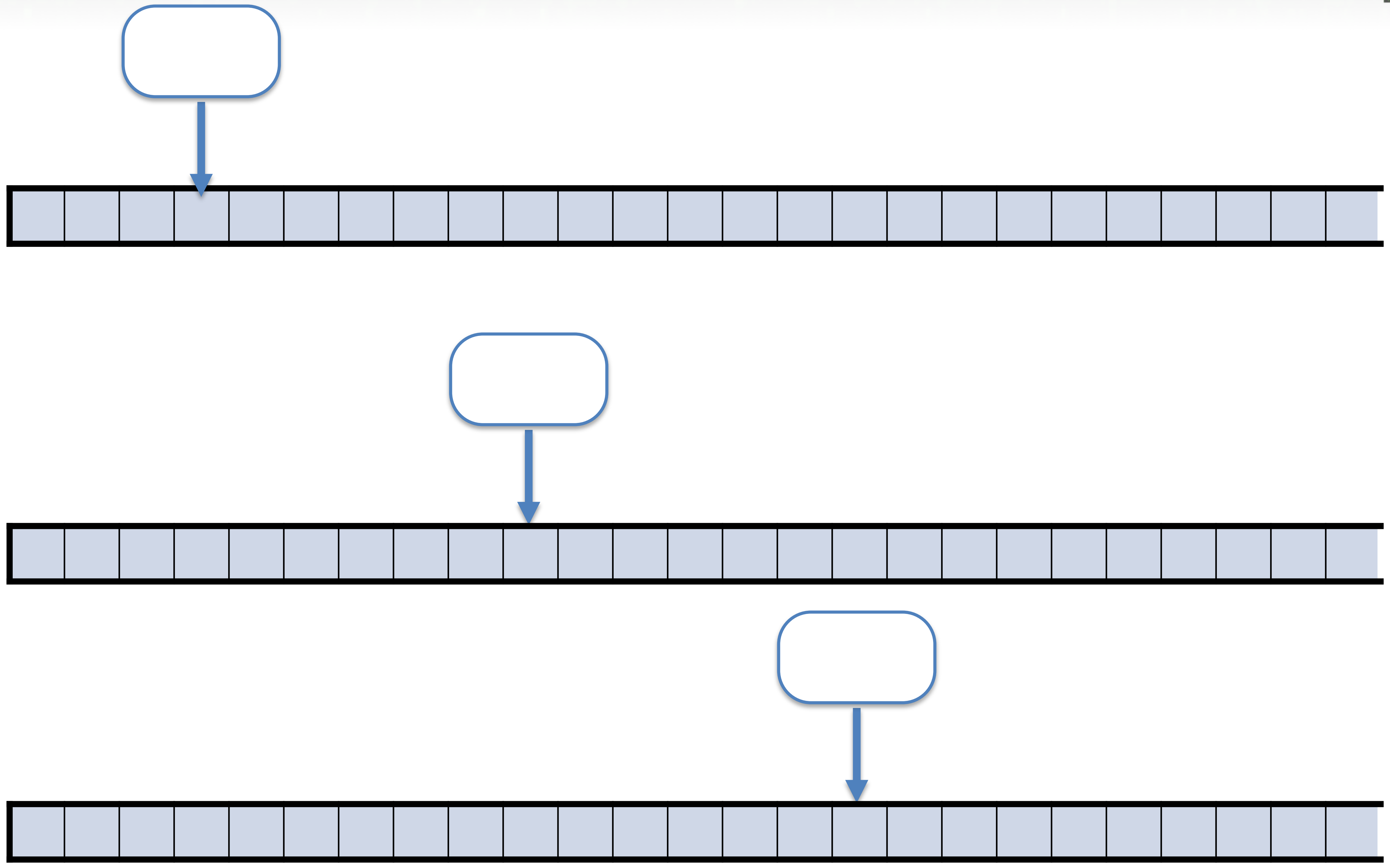
read symbols under all heads

print (possibly different) symbols under heads

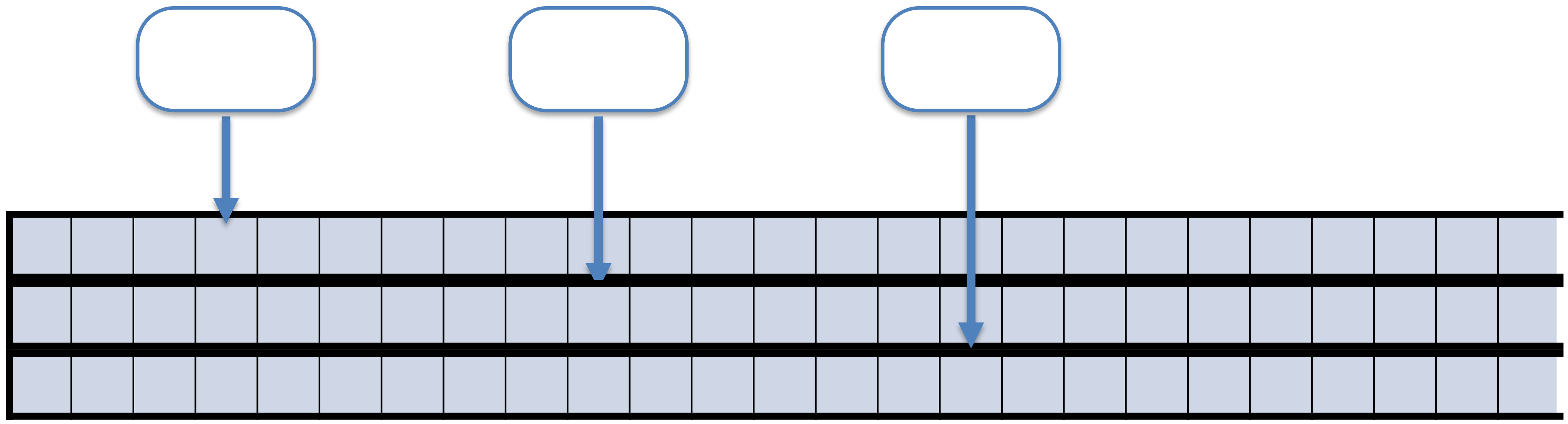
move all heads (possibly in different directions)

go to new state

Extension: multiple tapes



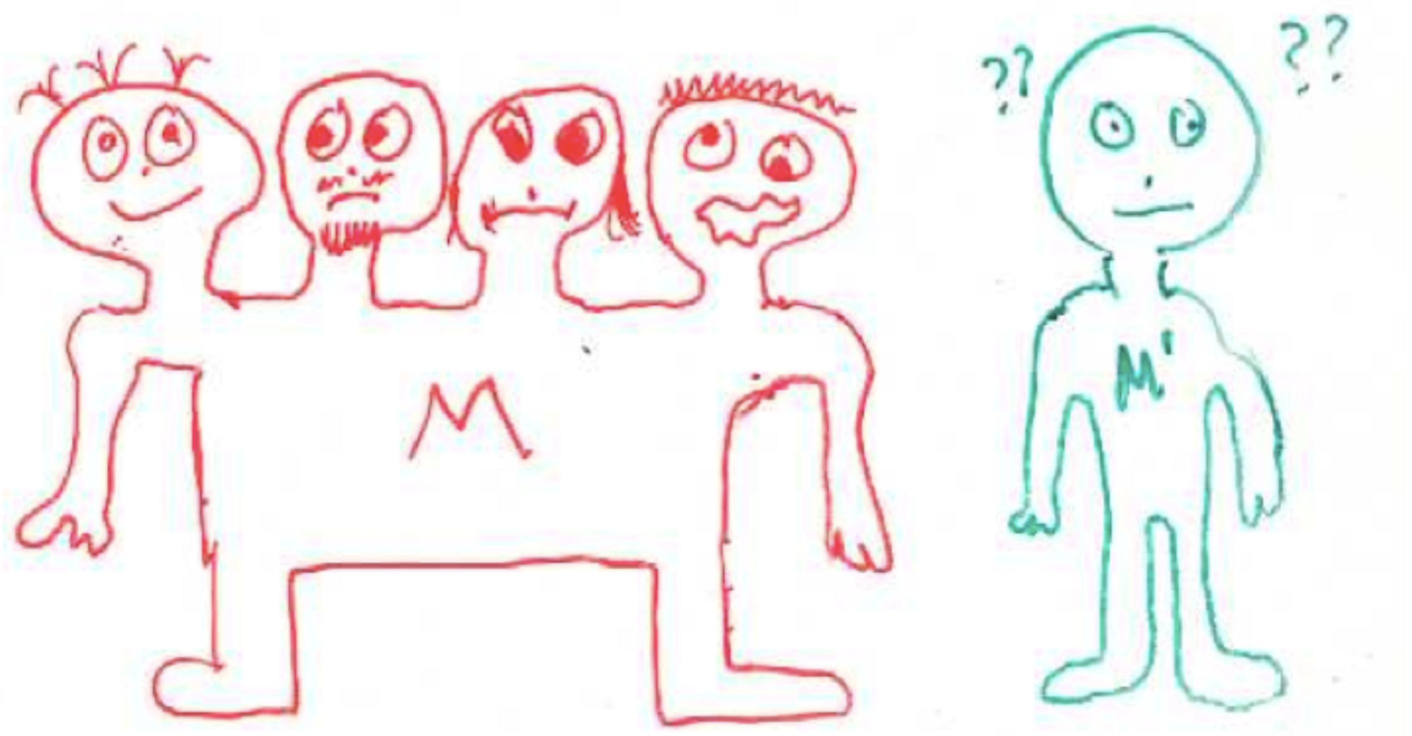
Extension: multiple tapes



Can't compute more with k tapes

Theorem: If L is accepted by a k-tape TM M, then L is accepted by some 1-tape TM M'.

Idea: M' uses k tracks to simulate tapes of M

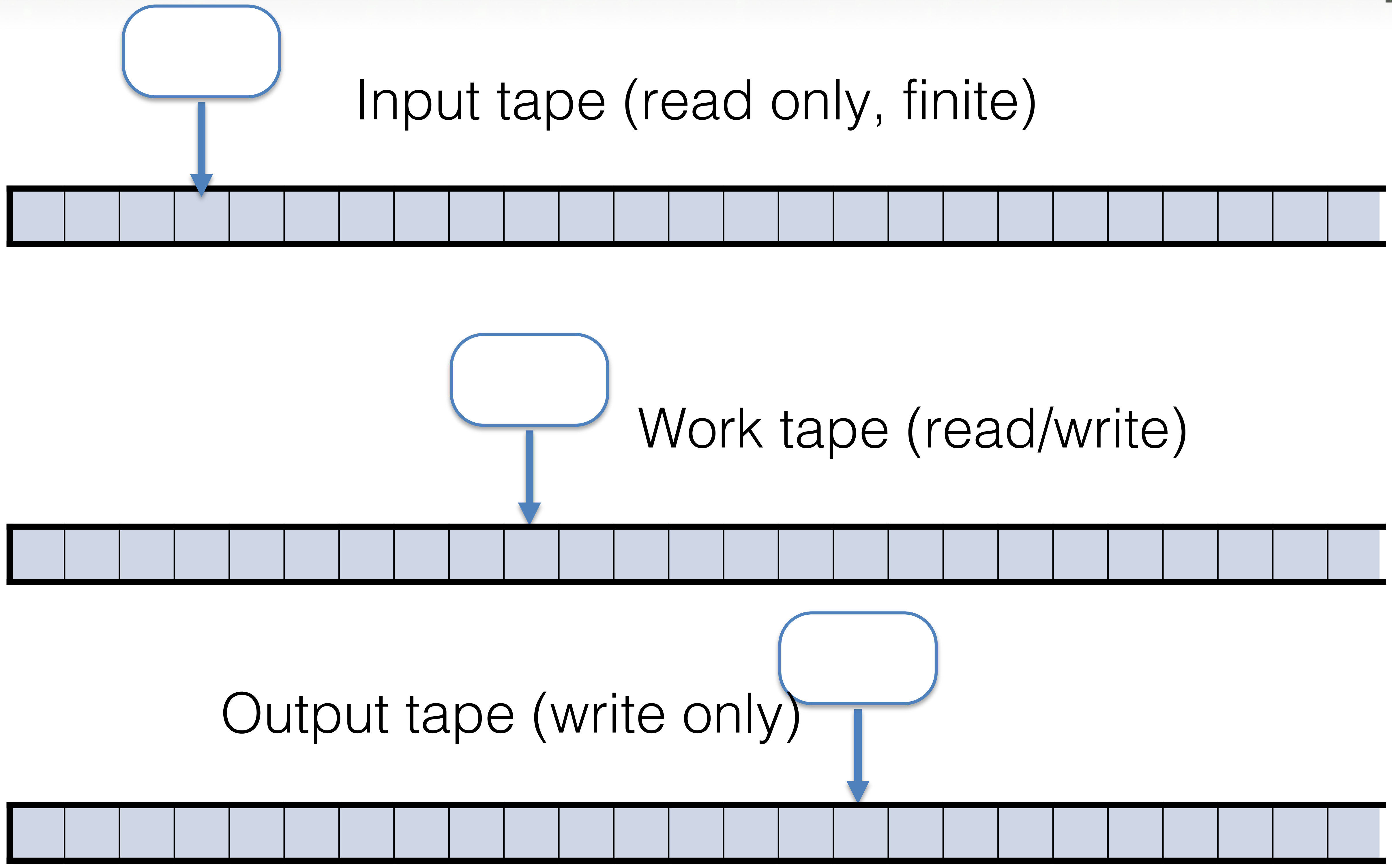


BUT....
M has k heads!

How can M′ be in k places at once?

M' will use $2k$ tracks to simulate tapes+heads of M

Convention for TM

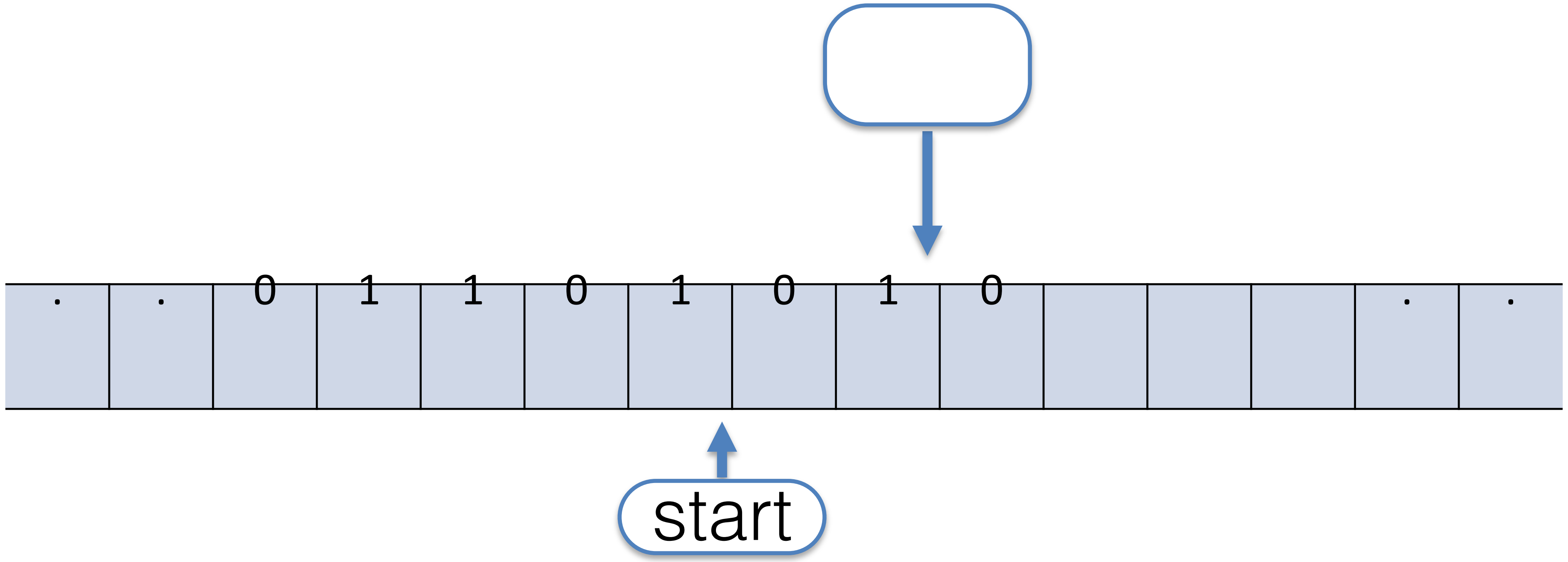


Convention for TM

More convenient!

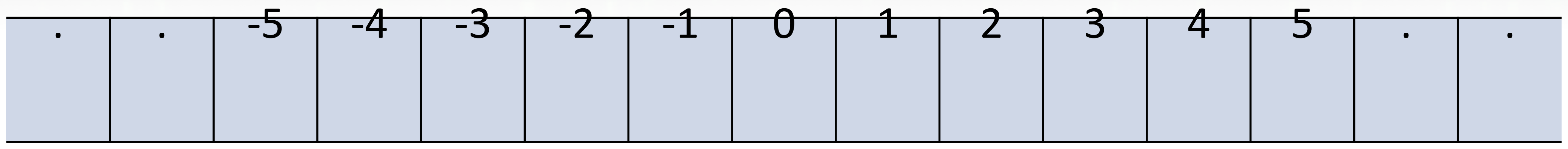
- Output doesn’t clash with input
- Don’t have to clean work tape
- Just remember to copy what I need to output tape

Extension: 2-Way Infinite Tape

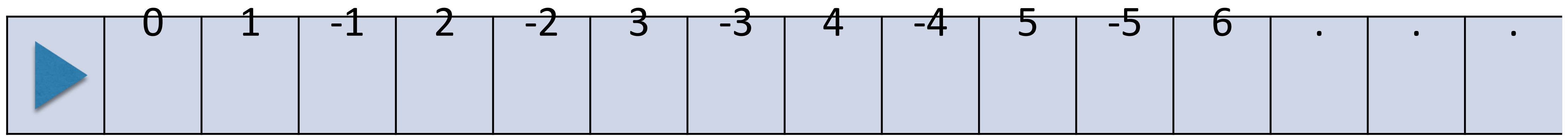


How to do it with one infinite direction?

2-Way Infinite Tape: Folding



Simulating it in the original TM variant:

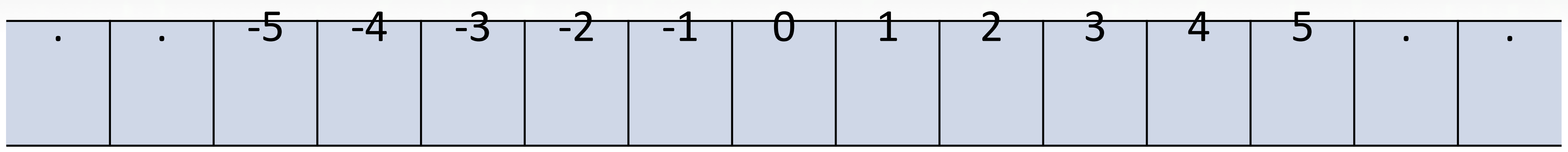


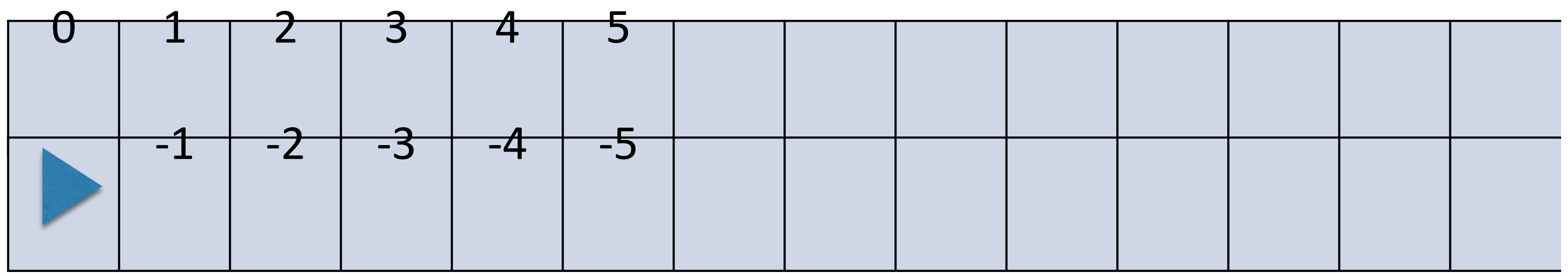
Modify transitions:
Remember in control if +ve or -ve side of tape
(contents of 0 cell will be marked).

If positive: R → RR & L → LL
If negative: R → LL & L → RR
At 0: R → R & L → RR

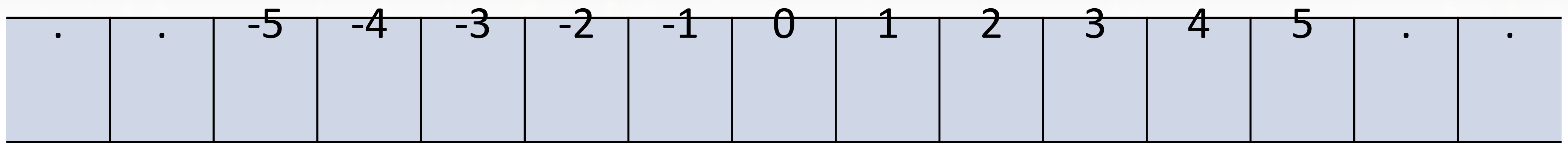
At 1?

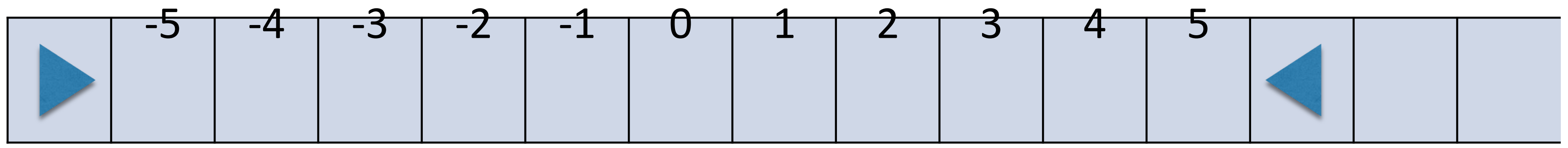
2-Way Infinite Tape: multiple tracks



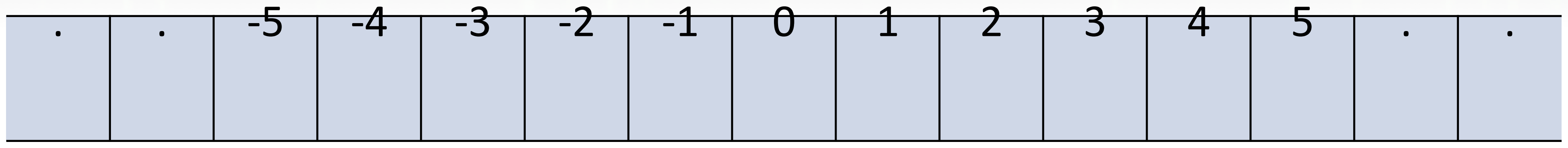


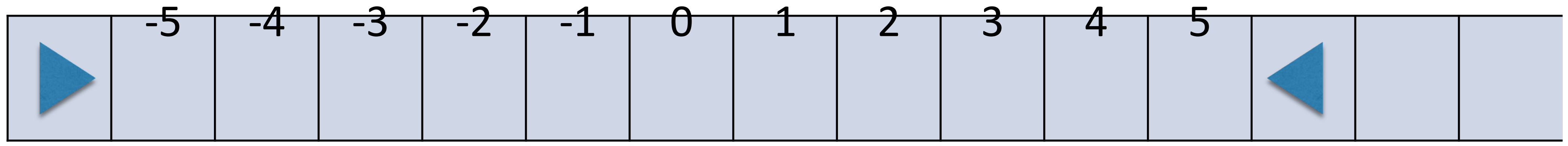
2-Way Infinite Tape: shifting





2-Way Infinite Tape: shifting

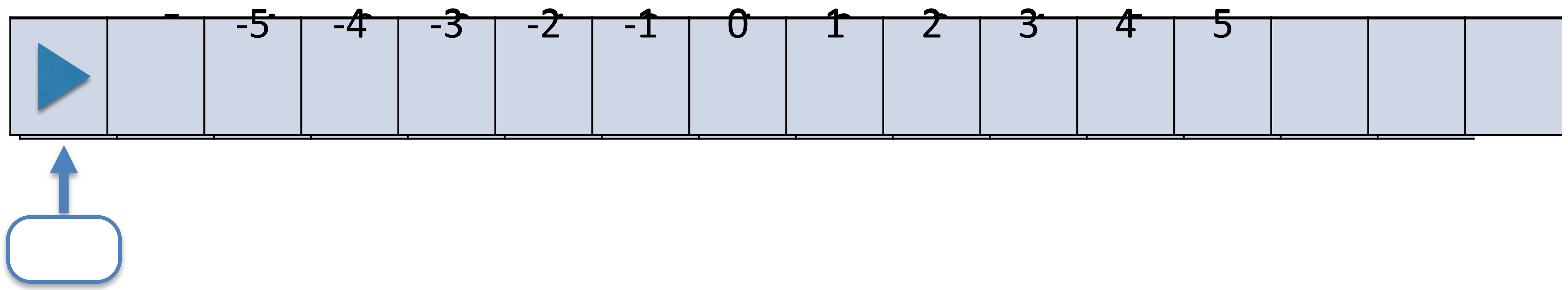




When the machine reads ◀ write a blank, move right, write a ◀, move right and proceed as if we had read a blank.

When the machine reads ▶ shift the entire contents of the tape to the right. Move back to the ▶, move right, write a blank and proceed as if we had read a blank.

2-Way Infinite Tape: shifting



When the machine reads ▶ shift the entire contents of the tape to the right. Move back to the ▶, move right, write a blank and proceed as if we had read a blank.

Subroutine calls

Mechanism for M_1 to "call" M_2 on an argument

Goal:

I need to be able to do two things:

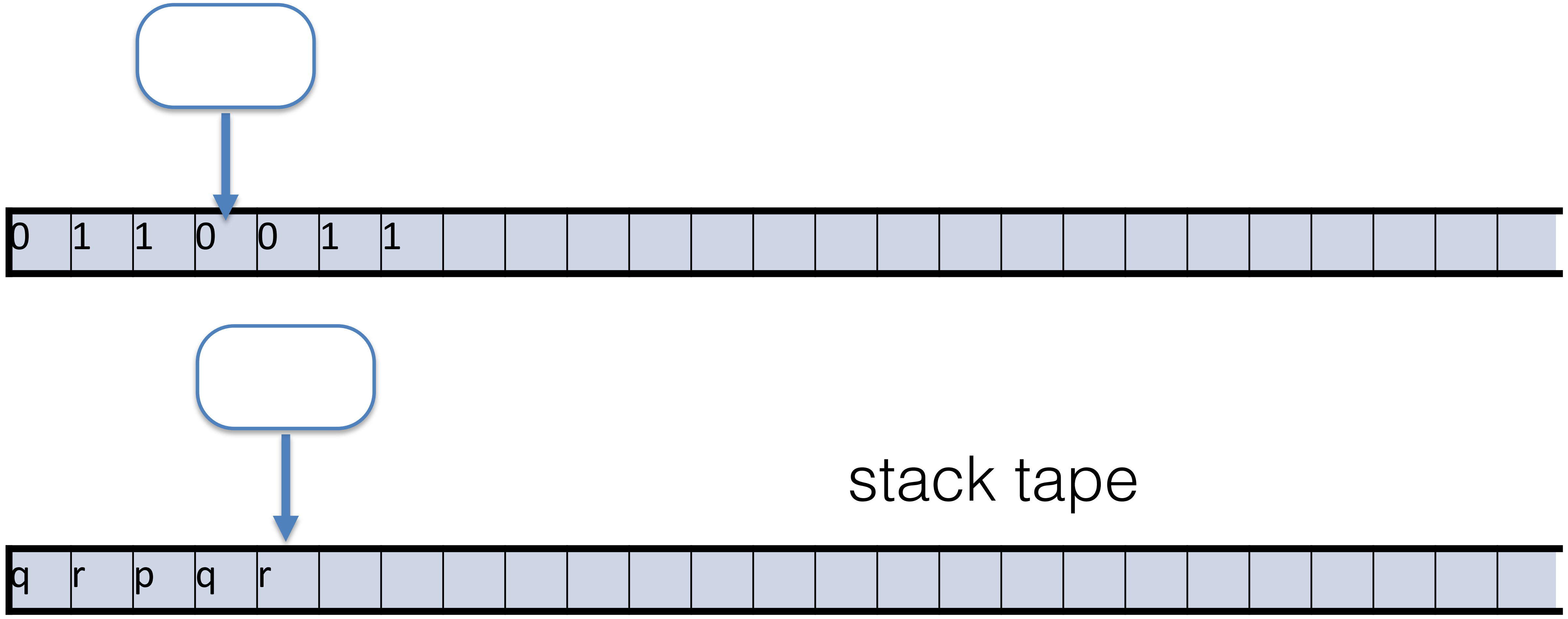
call

- push(q) :push the state in some stack, save it.
- pop(q): pop whatever state is on top of stack and make it current state.

return

Subroutine calls

Implement the Stack with a new tape



- For push, write a new symbol to stack and move R
- For pop read symbol, write blank, move head L

Subroutine calls

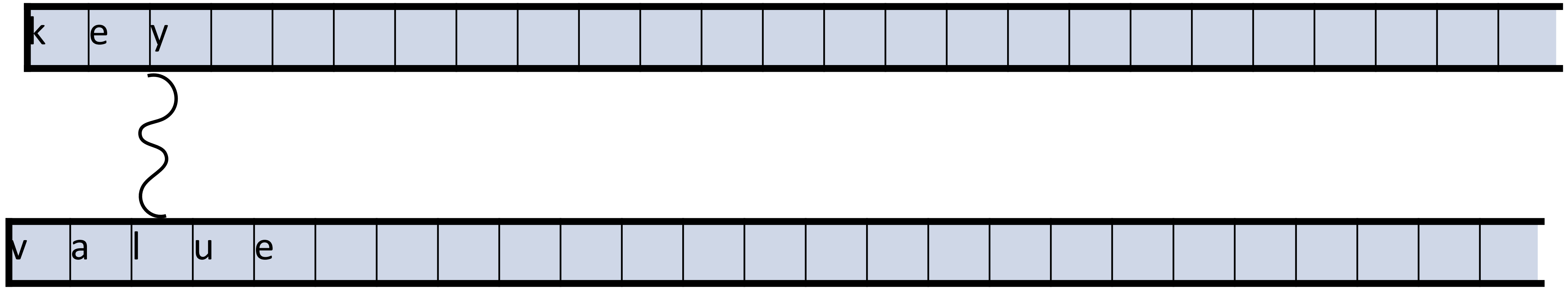
- Recursion (e.g. Fibonacci)
- Can take existing TMs and call them as subroutines.
- Call = jump to start state of the TM subroutine
- Halt = return

Random Access Memory (RAM)

- By definition can only access memory directly under the head.
- How to do associative memory?
- Memory is made up from pairs [key,value]
- key $\in \{0,1\}^*$, value $\in \{0,1\}^*$

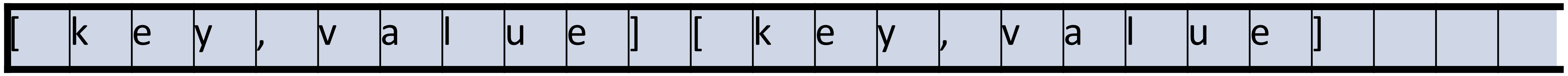
Random Access Memory (RAM)

- Would like a subroutine that starts with "key" written at the beginning of a tape and ends with "value" written at the beginning of the same tape

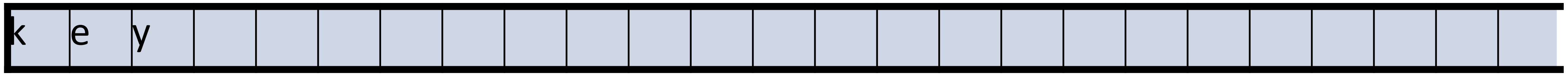


for any key a most one value

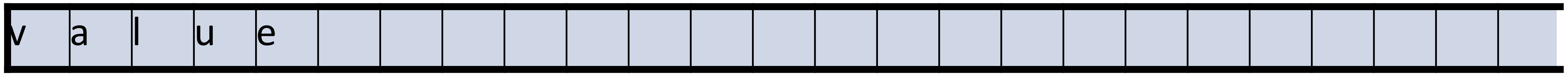
Random Access Memory (RAM)



Ram tape $\Sigma = \{[\ ]\ ,0\ 1\}$



Address tape



Value tape

Random Access Memory (RAM)

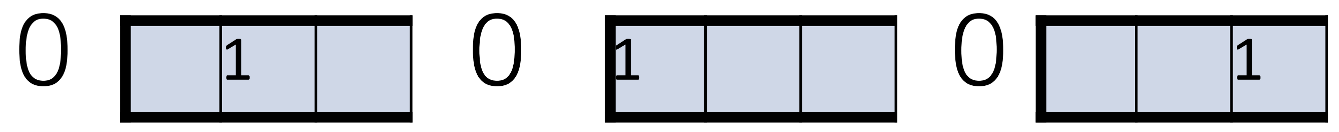
- If I have an RAM also that runs in time T(n), I can simulate it in one tape, one head,one track TM in time $T(n)^2$

Universal Turing Machine

- "Turing machine interpreter written in Turing machine".
- Just as the input to a Python interpreter is a string of Python source code, the input to our universal Turing machine U is a string 〈M,w〉 that encodes an arbitrary Turing machine M and a string w in the input alphabet of M.
- Given these encodings, U simulates the execution of M on input w; in particular,
- U accepts 〈M,w〉 if and only if M accepts w.
- U rejects 〈M,w〉 if and only if M rejects w.

Universal Turing Machine

- How to encode a Turing Machine as a binary string:
  - $01^{|\Gamma|}01^{|\Sigma|}01^{|Q|}0$[….] where [….] is some encoding (brute force) of all possible transitions as pattern of bits.
  - Encode the tape as a bit string: (e.g. tape alphabet of 3 symbols {a,b,c})



:tape was bac