CS116-Automata Theory and Formal Languages

Lecture 12
Turing Machine Variations and The Universal Turing Machine

Computer Science Department
1st Semester 2025-2026

Turing's thesis (1930):

Any computation carried out
by mechanical means
can be performed by a Turing Machine

Algorithm:

An algorithm for a problem is a Turing Machine which solves the problem

The algorithm describes the steps of the mechanical means

This is easily translated to computation steps of a Turing machine

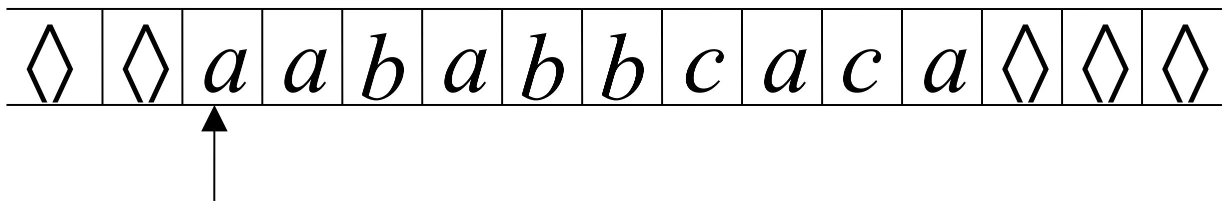
When we say: There exists an algorithm

We mean: There exists a Turing Machine that executes the algorithm

Variations of the Turing Machine

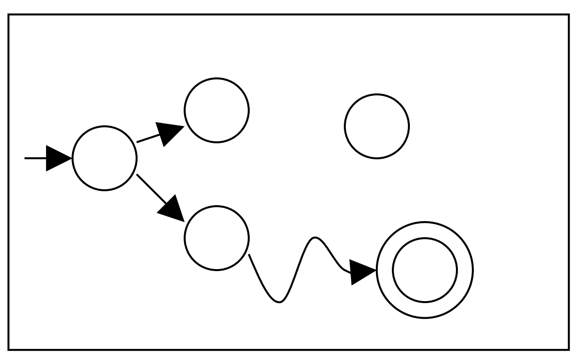
The Standard Model

Infinite Tape



Read-Write Head (Left or Right)

Control Unit



Deterministic

Variations of the Standard Model

Turing machines with:

- Stay-Option
- Semi-Infinite Tape
- Off-Line
- Multitape
- Multidimensional
- Nondeterministic

Different Turing Machine **Classes**

Same Power of two machine classes:

both classes accept the same set of languages

We will prove:

each new class has the same power with Standard Turing Machine

(accept Turing-Recognizable Languages)

Same Power of two classes means:

for any machine M_1 of first class

there is a machine M_2 of second class

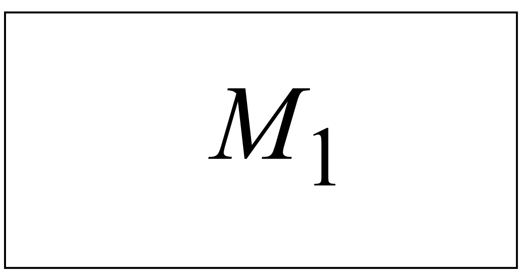
such that: $L(M_1) = L(M_2)$

and vice-versa

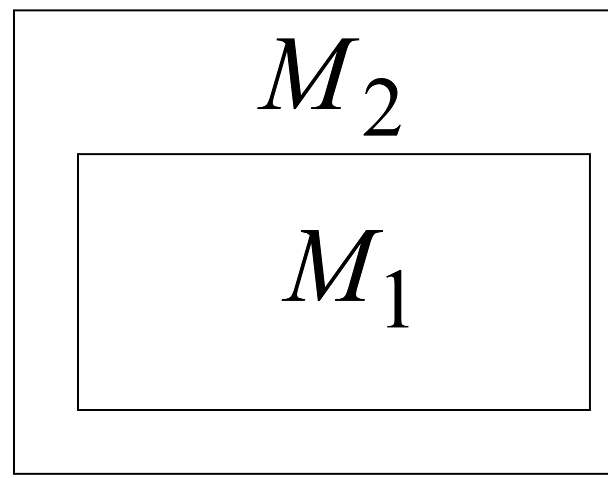
Simulation: A technique to prove same power.

Simulate the machine of one class with a machine of the other class

First Class
Original Machine



Second Class
Simulation Machine



simulates M_1

Configurations in the Original Machine M_1 have corresponding configurations in the Simulation Machine M_2

$$\text{Original Machine:} \quad d_0 \overset{M_1}{\succ} d_1 \succ \cdots \succ d_n$$

$$\updownarrow \qquad \updownarrow \qquad\qquad \updownarrow$$

$$\text{Simulation Machine:} \quad d'_0 \overset{*}{\succ} d'_1 \overset{*}{\underset{M_2}{\succ}} \cdots \overset{*}{\succ} d'_n$$

Accepting Configuration

Original Machine: d_f

$\updownarrow$

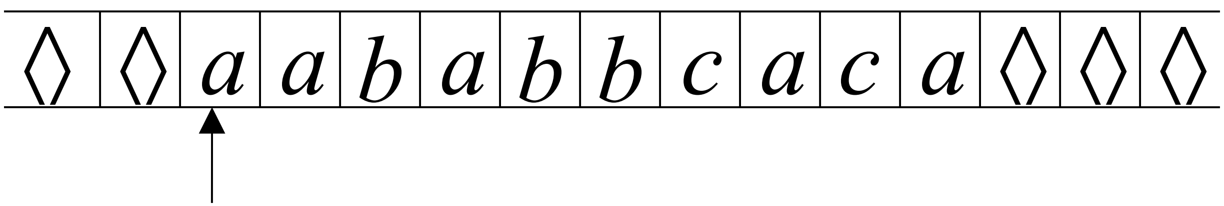
Simulation Machine: d'_f

the Simulation Machine
and the Original Machine
accept the same strings

$$L(M_1) = L(M_2)$$

Turing Machines with Stay-Option

The head can stay in the same position

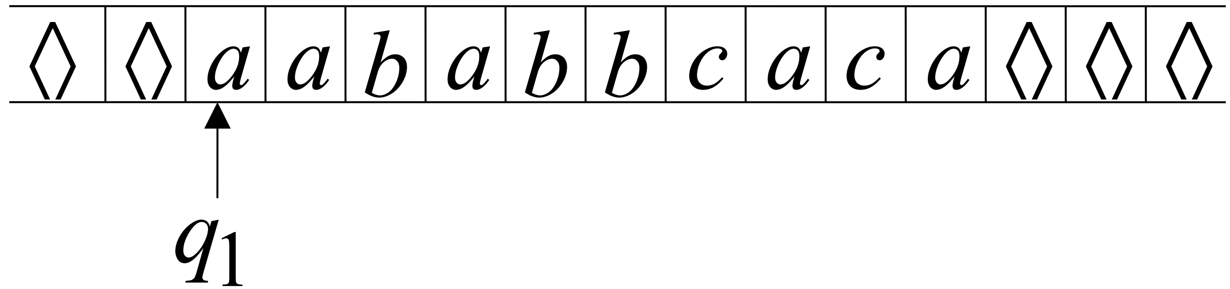


Left, Right, Stay

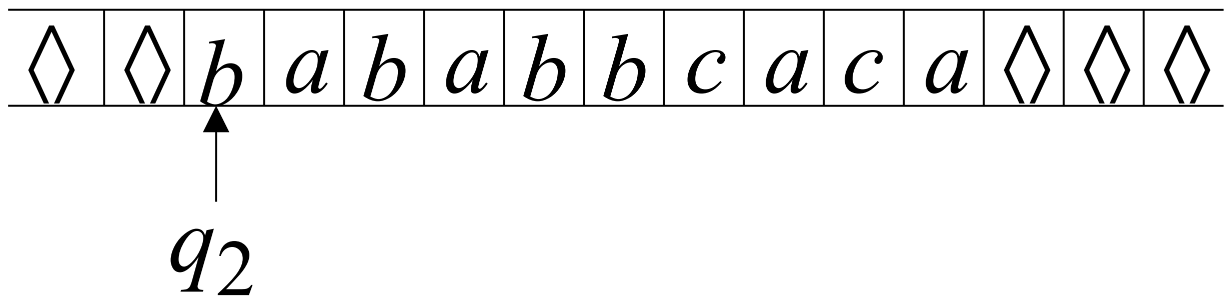
L,R,S: possible head moves

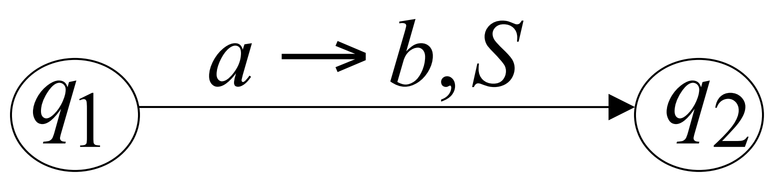
Example:

Time 1



Time 2





Theorem: Stay-Option machines have the same power with Standard Turing machines

Proof:

1. Stay-Option Machines simulate Standard Turing machines

2. Standard Turing machines simulate Stay-Option machines

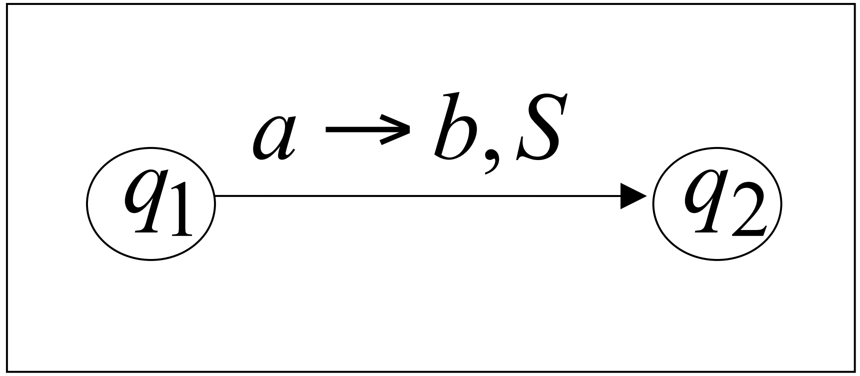
1. Stay-Option Machines
simulate Standard Turing machines

Trivial: any standard Turing machine
is also a Stay-Option machine

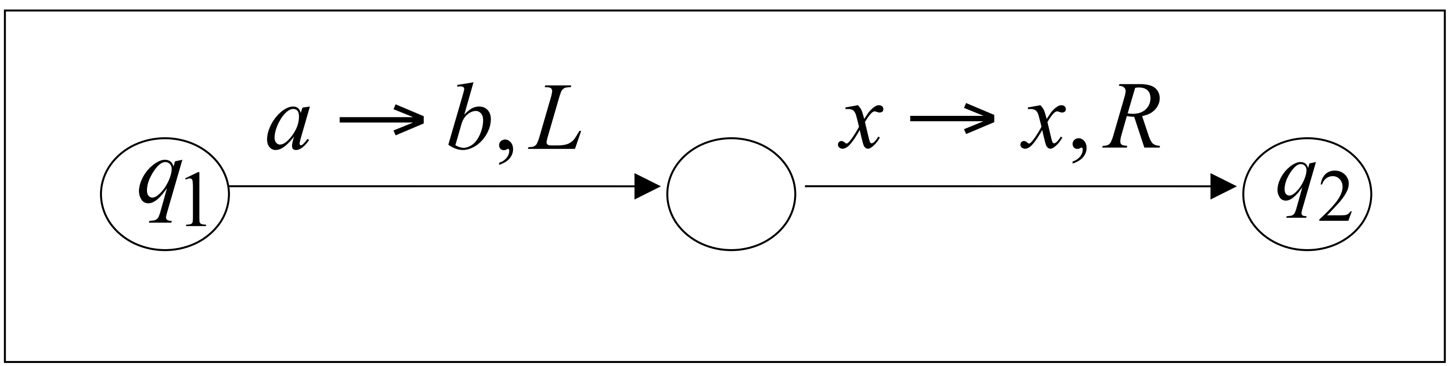
2. Standard Turing machines simulate Stay-Option machines

We need to simulate the stay head option with two head moves, one left and one right

Stay-Option Machine



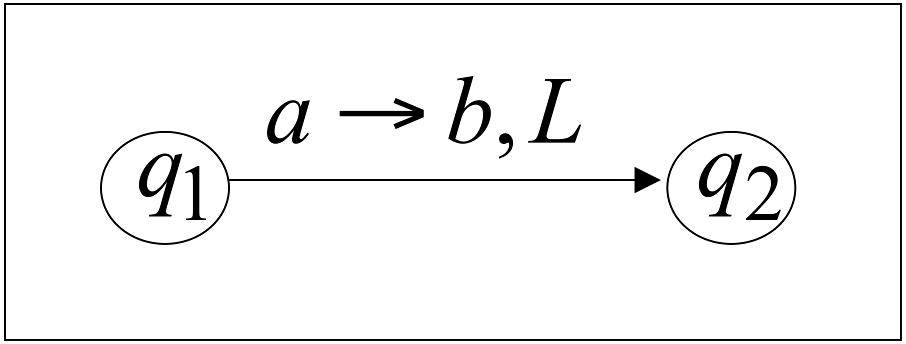
Simulation in Standard Machine



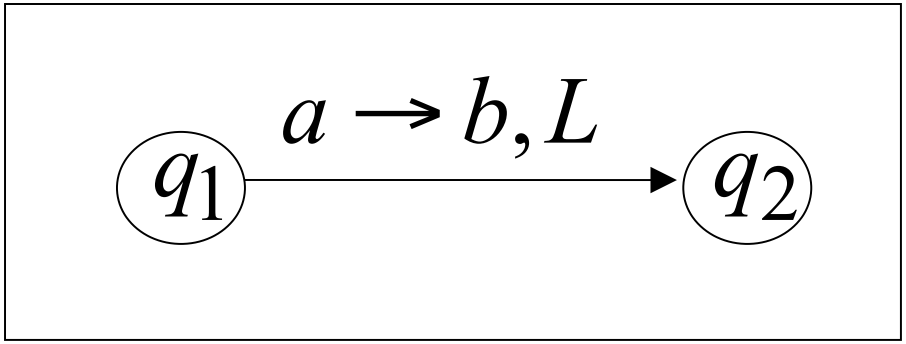
For every possible tape symbol x

For other transitions nothing changes

Stay-Option Machine

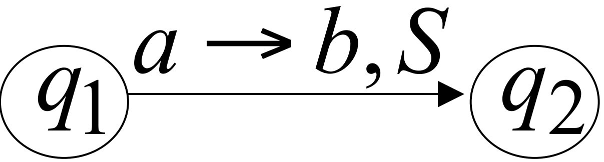


Simulation in Standard Machine

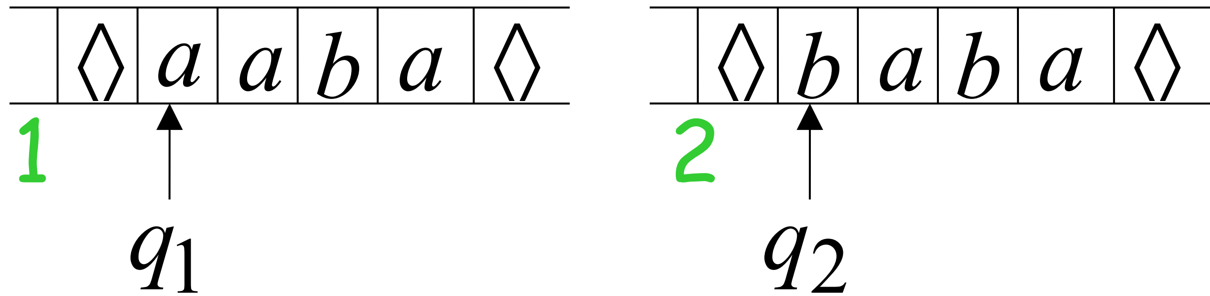


Similar for Right moves

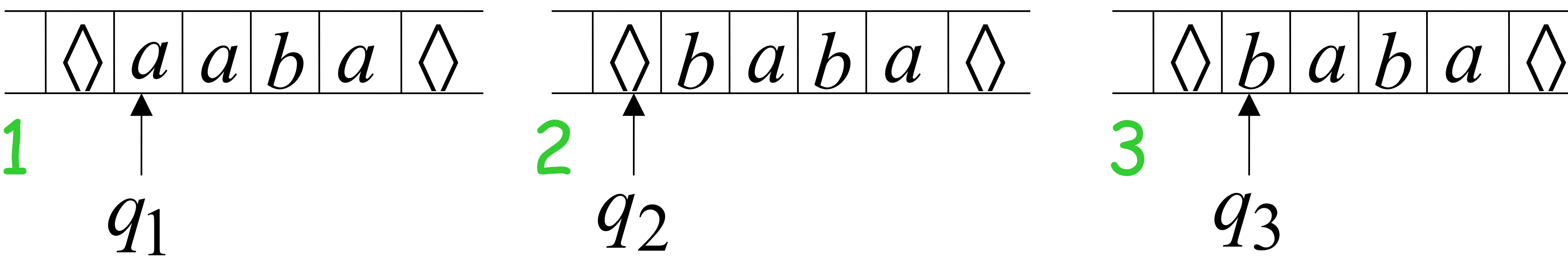
example of simulation



Stay-Option Machine:



Simulation in Standard Machine:

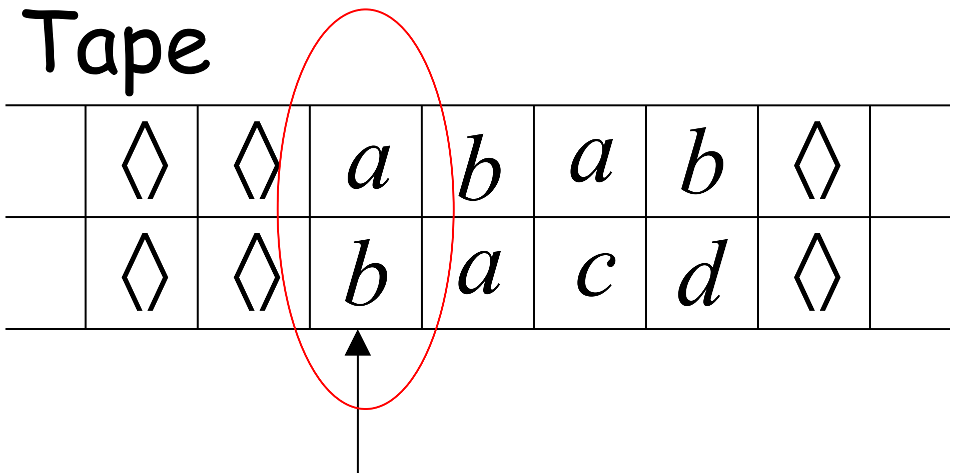


END OF PROOF

Multiple Track Tape

A useful trick to perform more complicated simulations

One Tape

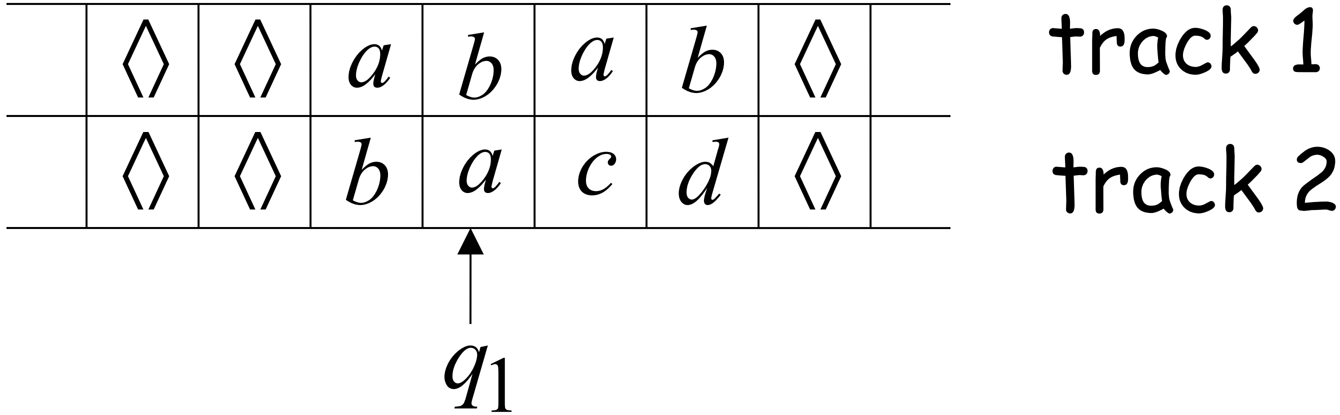


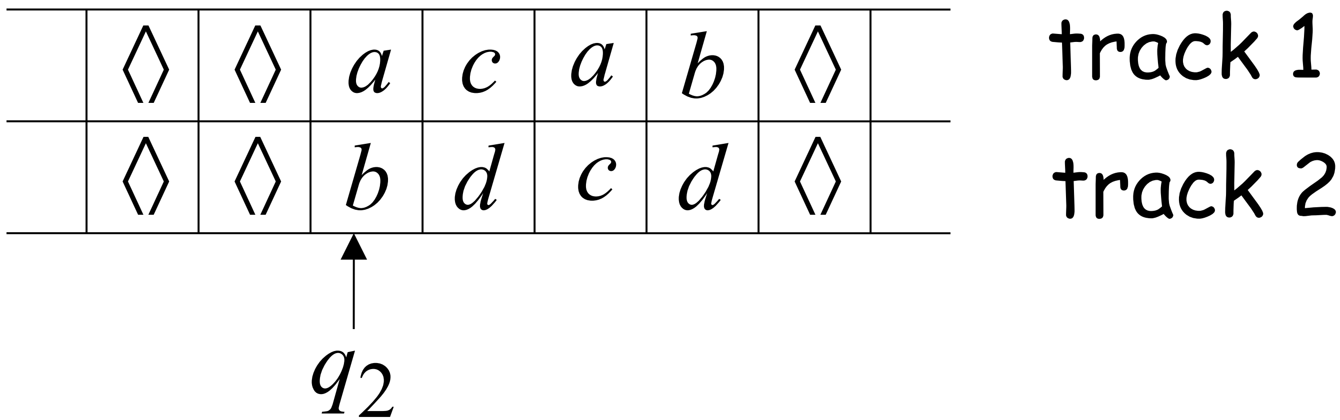
track 1

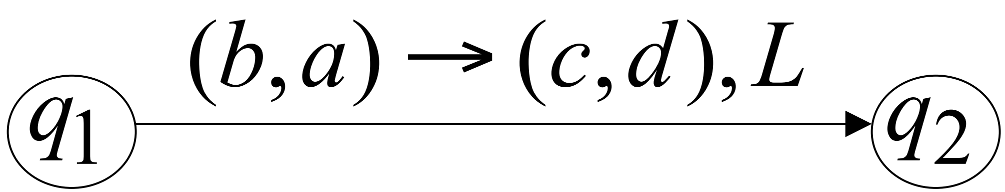
track 2

One head

One symbol (a, b)

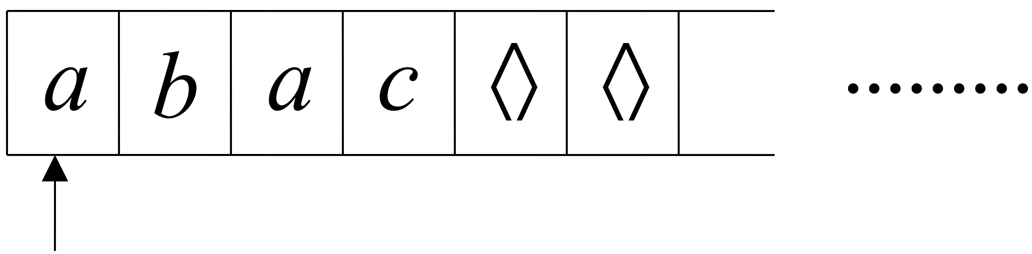

track 1
track 2
q_1


track 1
track 2
q_2


$(b,a) \rightarrow (c,d), L$
q_1
q_2

Semi-Infinite Tape

The head extends infinitely only to the right



- Initial position is the leftmost cell

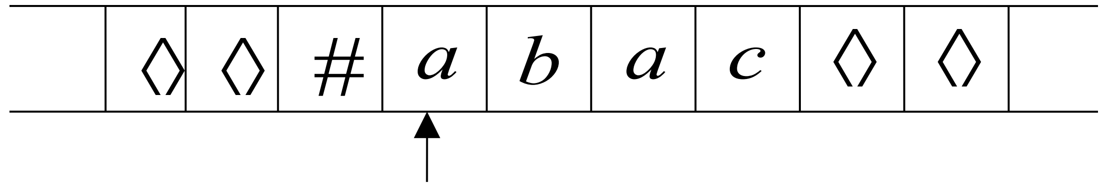
- When the head moves left from the border, it returns to the same position

Theorem: Semi-Infinite machines have the same power with Standard Turing machines

Proof: 1. Standard Turing machines simulate Semi-Infinite machines

2. Semi-Infinite Machines simulate Standard Turing machines

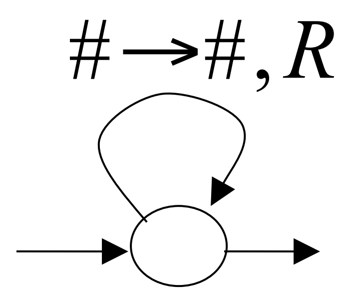
1. Standard Turing machines simulate Semi-Infinite machines:



Standard Turing Machine

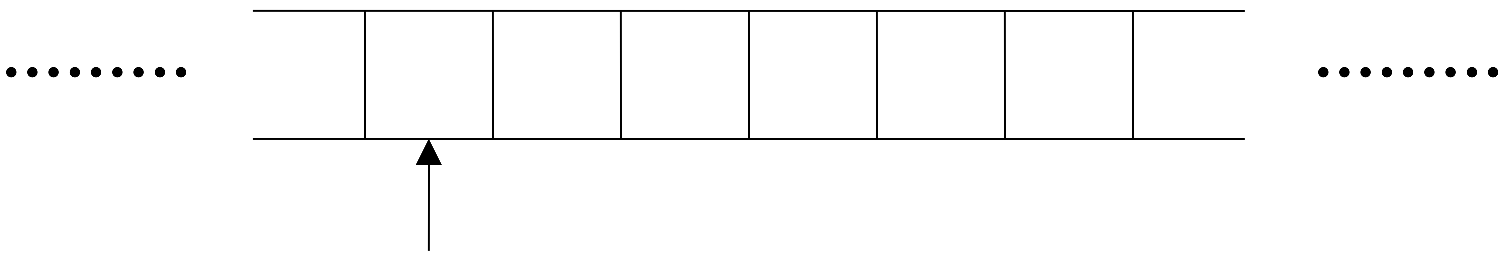
a. insert special symbol $\#$ at left of input string

b. Add a self-loop to every state (except states with no outgoing transitions)

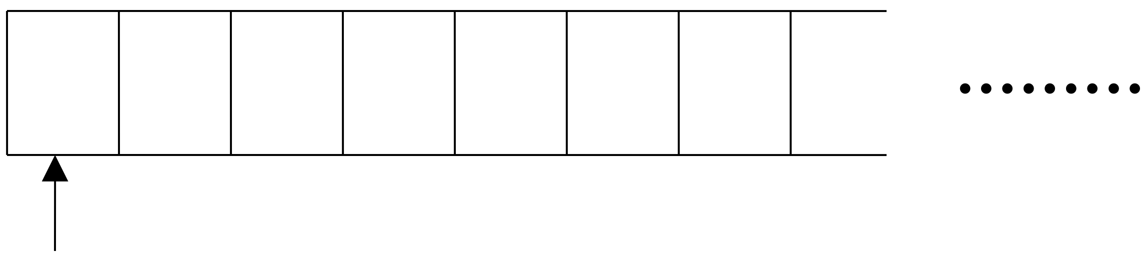


2. Semi-Infinite tape machines simulate Standard Turing machines:

Standard machine

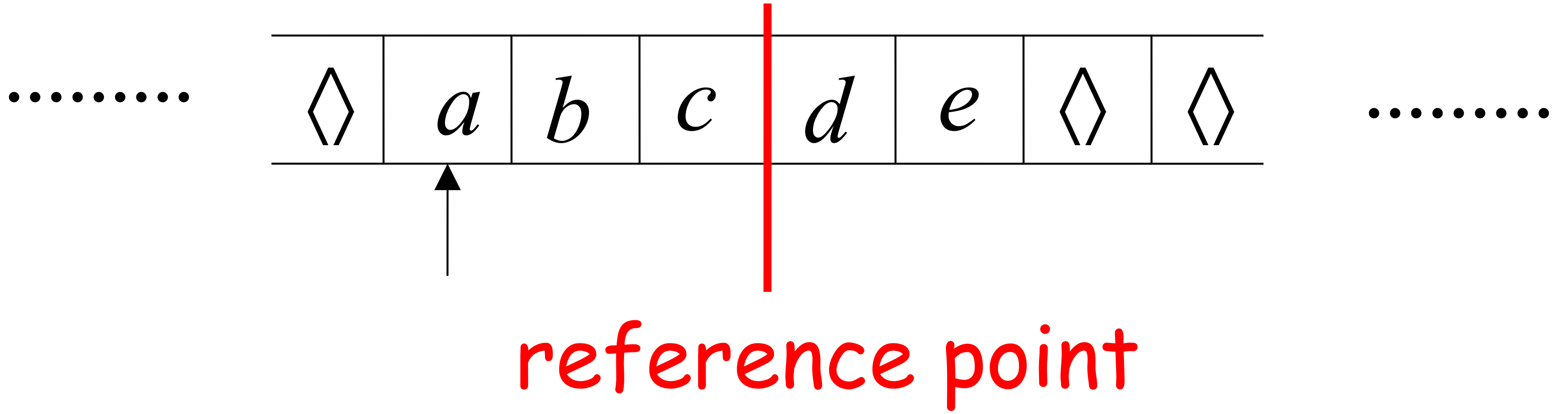


Semi-Infinite tape machine

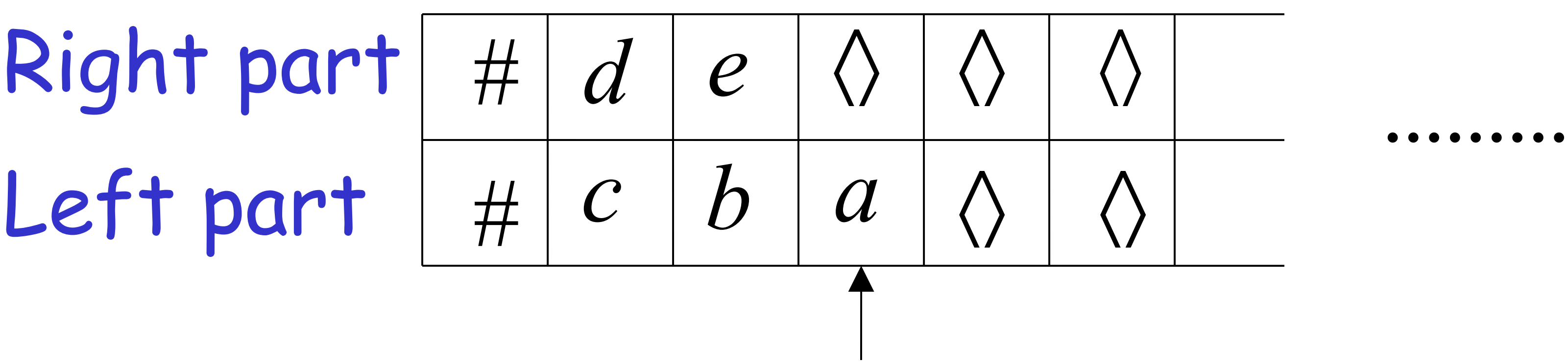


Squeeze infinity of both directions in one direction

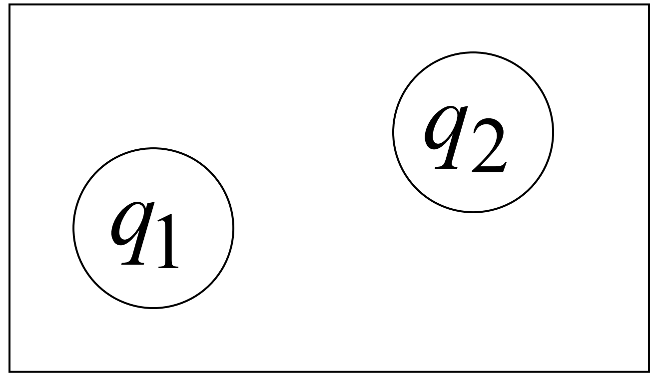
Standard machine



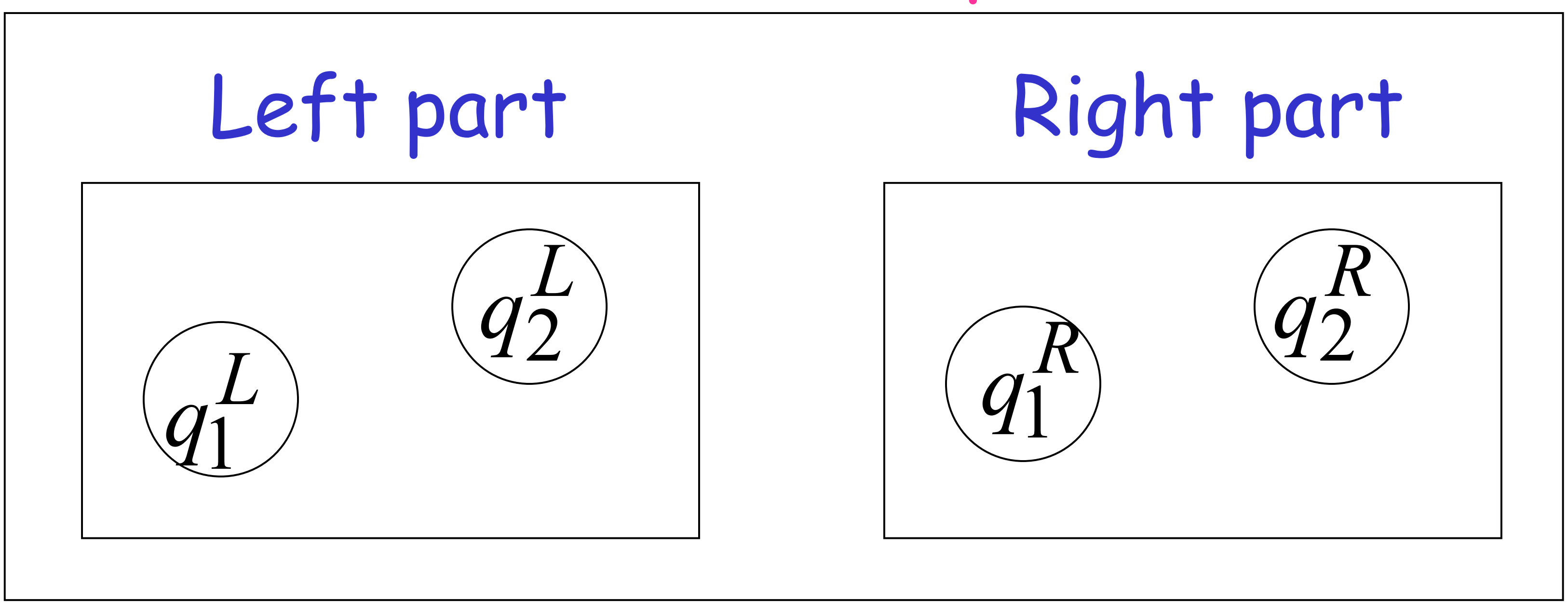
Semi-Infinite tape machine with two tracks



Standard machine



Semi-Infinite tape machine



Standard machine

$$q_1 \xrightarrow{a \to g, R} q_2$$

Semi-Infinite tape machine

Right part

$$q_1^R \xrightarrow{(a, x) \to (g, x), R} q_2^R$$

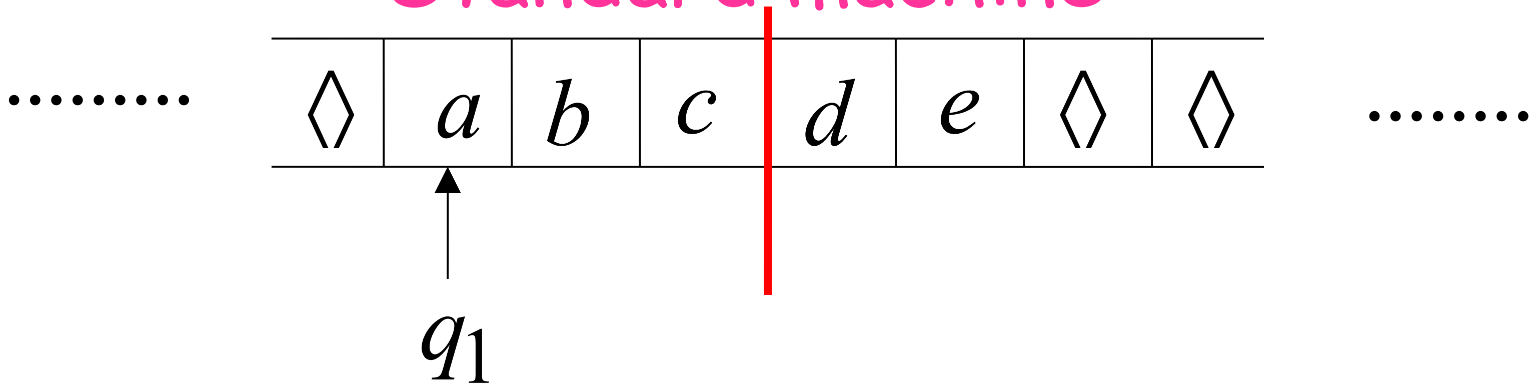
Left part

$$q_1^L \xrightarrow{(x, a) \to (x, g), L} q_2^L$$

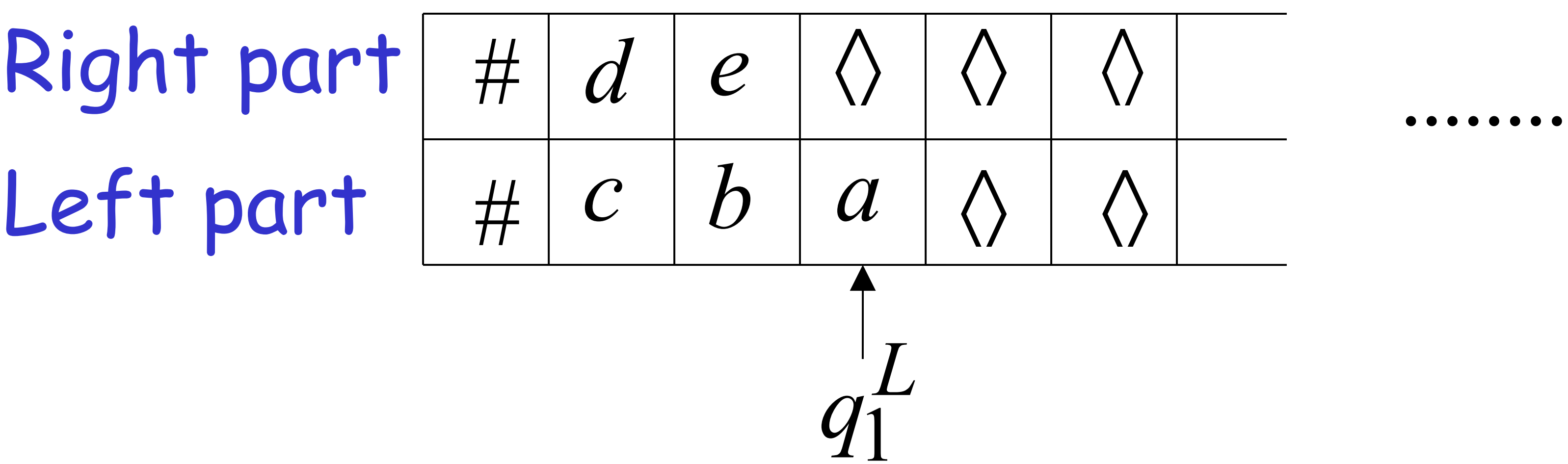
For all tape symbols x

Time 1

Standard machine

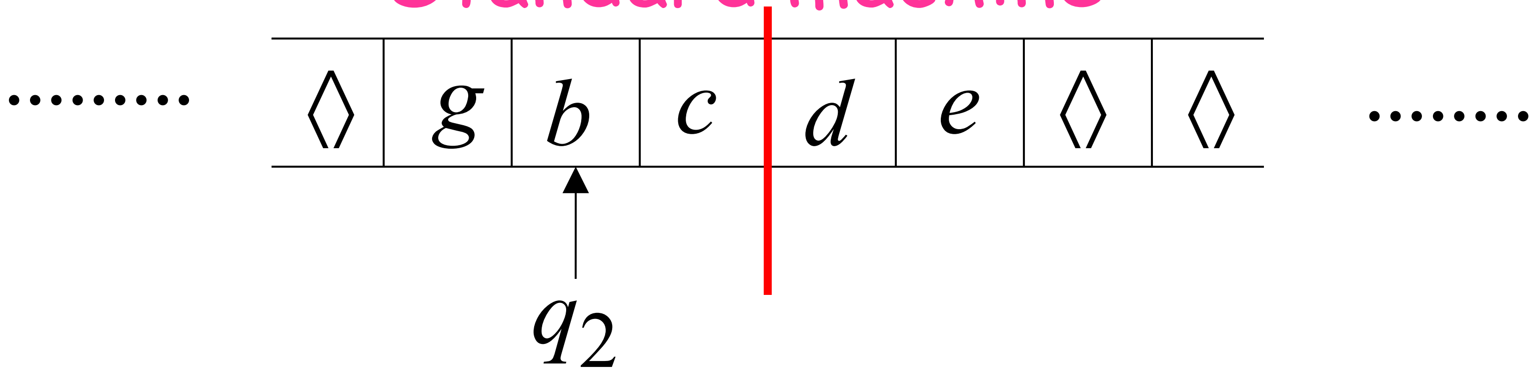


Semi-Infinite tape machine



Time 2

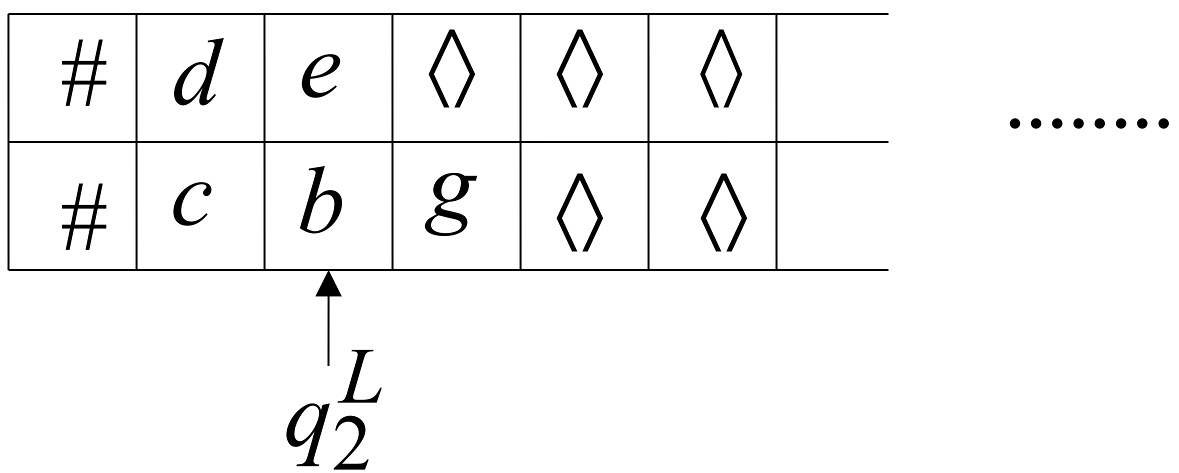
Standard machine



Semi-Infinite tape machine

Right part

Left part



At the border:

Semi-Infinite tape machine

Right part

$$q_1^R \xrightarrow{(\#,\#) \rightarrow (\#,\#),R} q_1^L$$

Left part

$$q_1^L \xrightarrow{(\#,\#) \rightarrow (\#,\#),R} q_1^R$$

Semi-Infinite tape machine

Time 1

Right part	#	d	e	$\Diamond$	$\Diamond$	$\Diamond$	....
Left part	#	c	b	g	$\Diamond$	$\Diamond$	

q_1^L

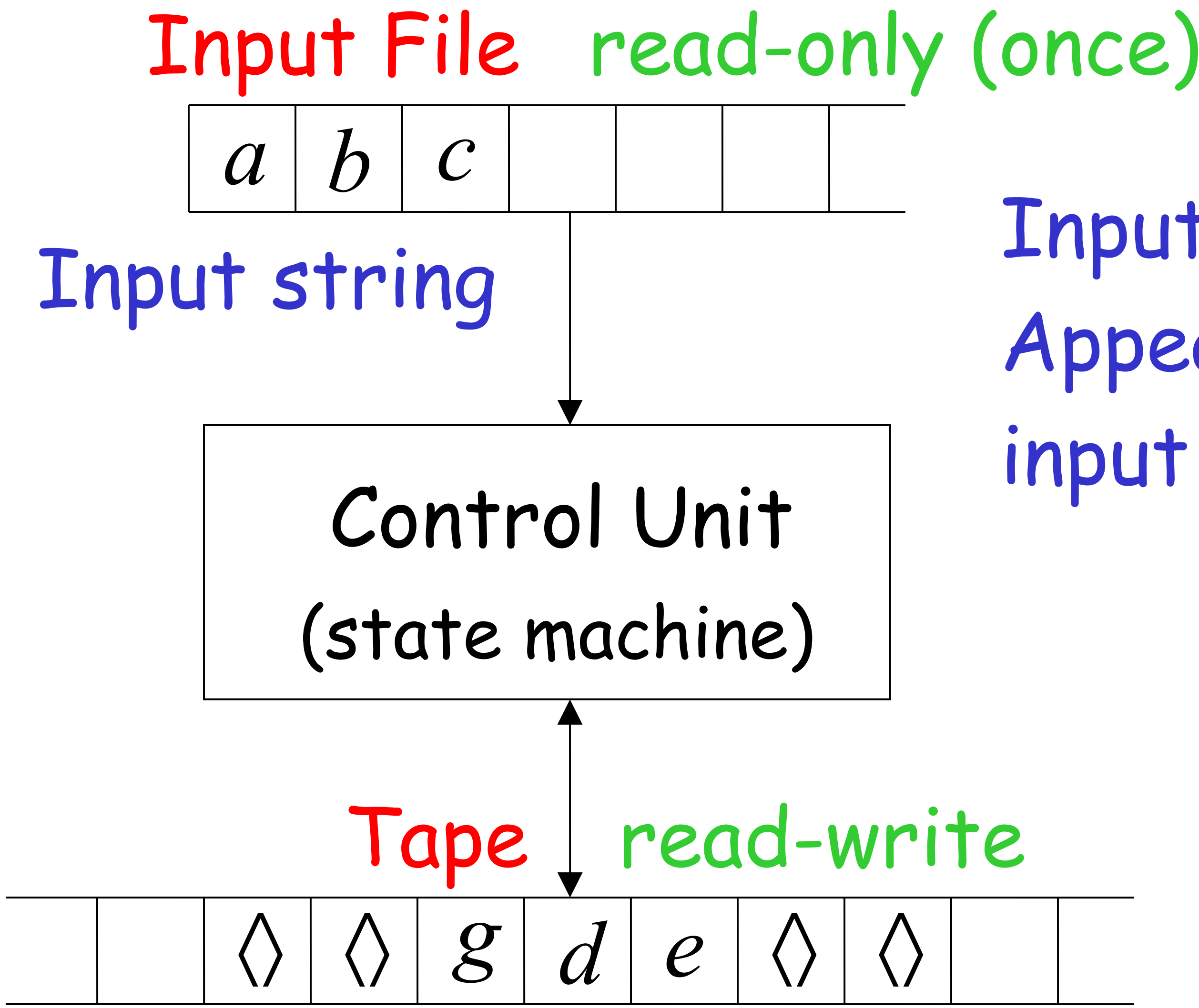
Time 2

Right part	#	d	e	$\Diamond$	$\Diamond$	$\Diamond$	....
Left part	#	c	b	g	$\Diamond$	$\Diamond$	

q_1^R

END OF PROOF

The Off-Line Machine



Input string
Appears on
input file only

Theorem: Off-Line machines have the same power with Standard Turing machines

Proof: 1. Off-Line machines simulate Standard Turing machines

2. Standard Turing machines simulate Off-Line machines

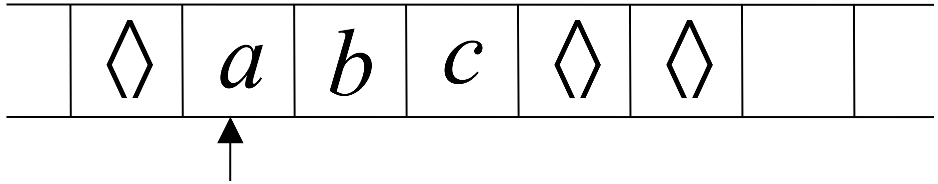
1. Off-line machines simulate Standard Turing Machines

Off-line machine:

1. Copy input file to tape

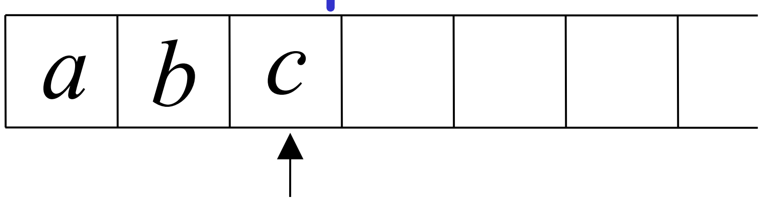
2. Continue computation as in Standard Turing machine

Standard machine

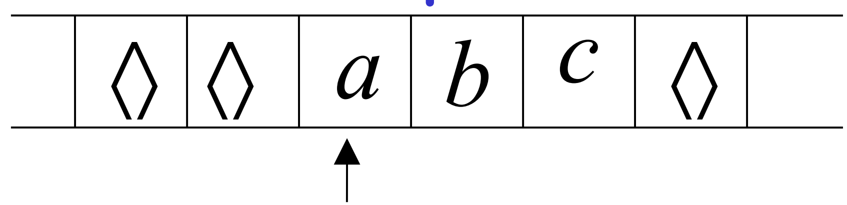


Off-line machine

Input File

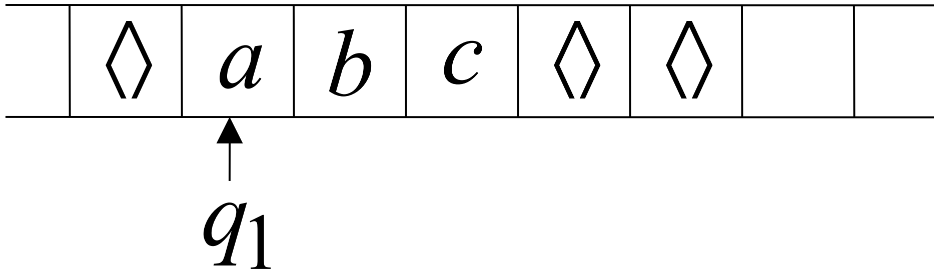


Tape



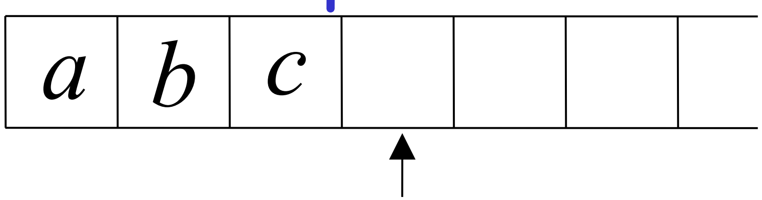
1. Copy input file to tape

Standard machine

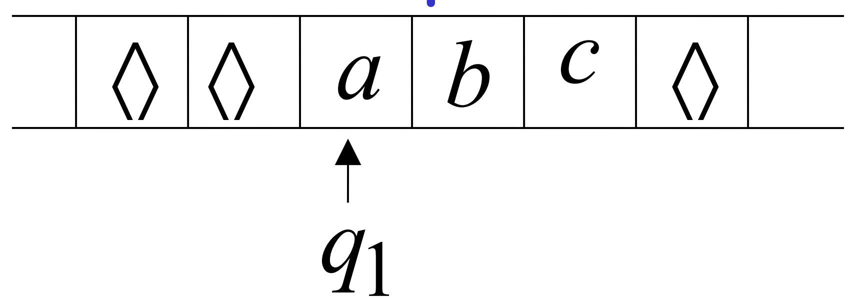


Off-line machine

Input File



Tape



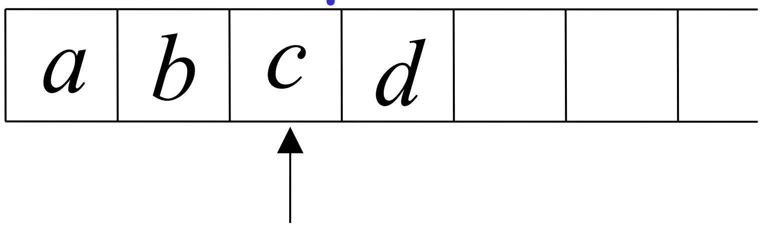
2. Do computations as in Turing machine

2. Standard Turing machines simulate Off-Line machines:

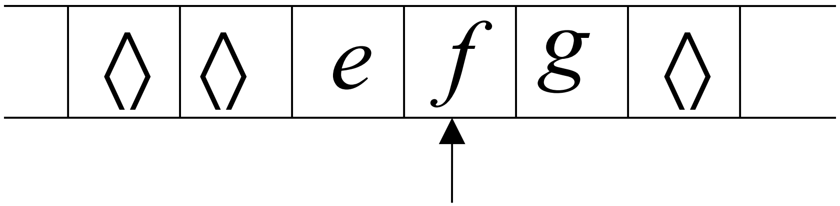
Use a Standard machine with
a four-track tape to keep track of
the Off-line input file and tape contents

Off-line Machine

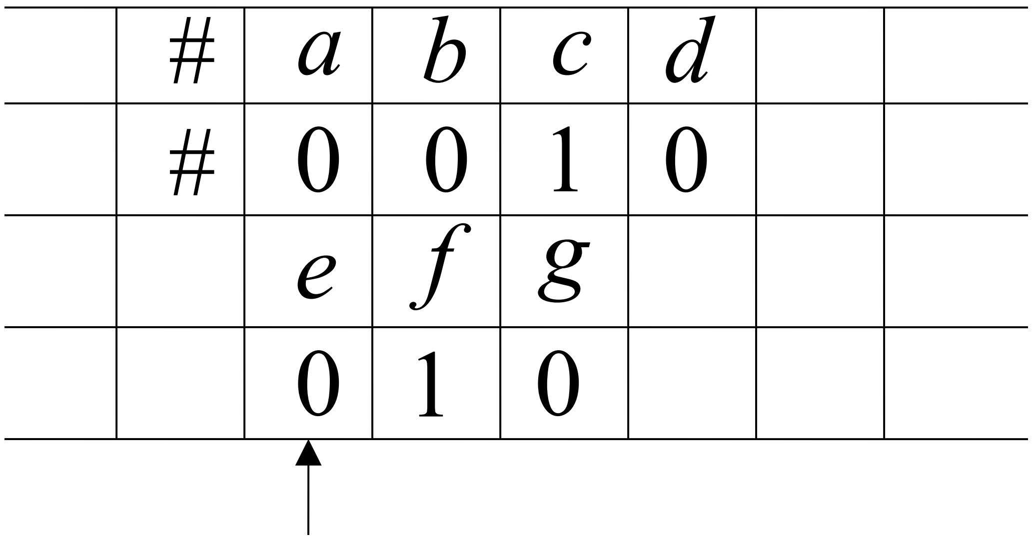
Input File



Tape



Standard Machine -- Four track tape



Input File

head position

Tape

head position

Reference point (uses special symbol #)

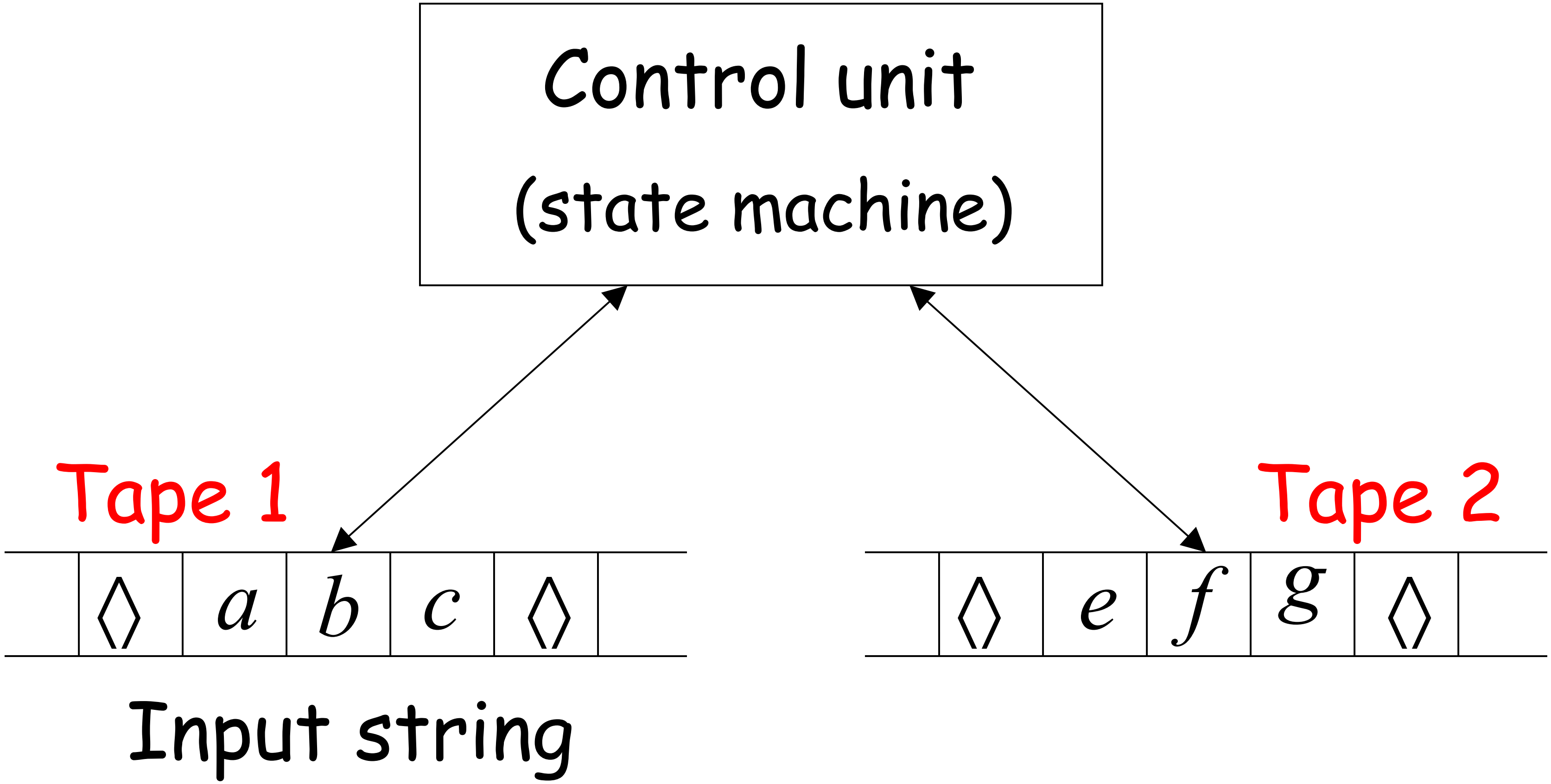
	#	a	b	c	d			Input File
	#	0	0	1	0			head position
	#	e	f	g				Tape
	#	0	1	0				head position

Repeat for each state transition:

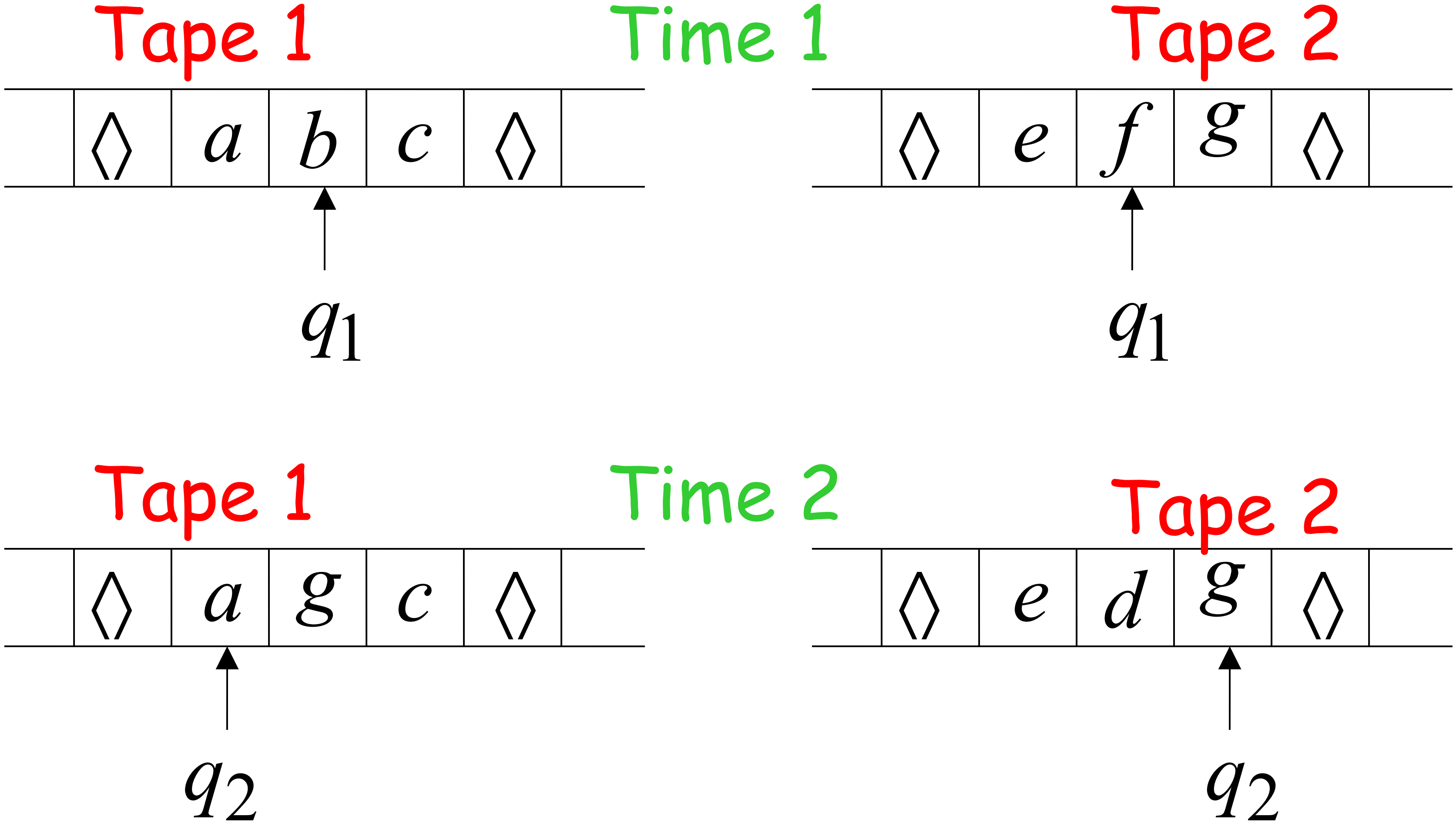
1. Return to reference point
2. Find current input file symbol
3. Find current tape symbol
4. Make transition

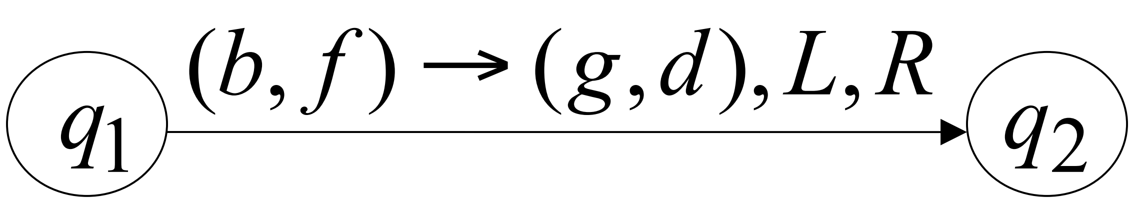
END OF PROOF

Multi-tape Turing Machines



Input string appears on Tape 1


Tape 1
Time 1
Tape 2
◊ a b c ◊
◊ e f g ◊
q_1
q_1
Tape 1
Time 2
Tape 2
◊ a g c ◊
◊ e d g ◊
q_2
q_2


q_1
$(b, f) \rightarrow (g, d), L, R$
q_2

Theorem: Multi-tape machines have the same power with Standard Turing machines

Proof: 1. Multi-tape machines simulate Standard Turing machines

2. Standard Turing machines simulate Multi-tape machines

1. Multi-tape machines simulate Standard Turing Machines:

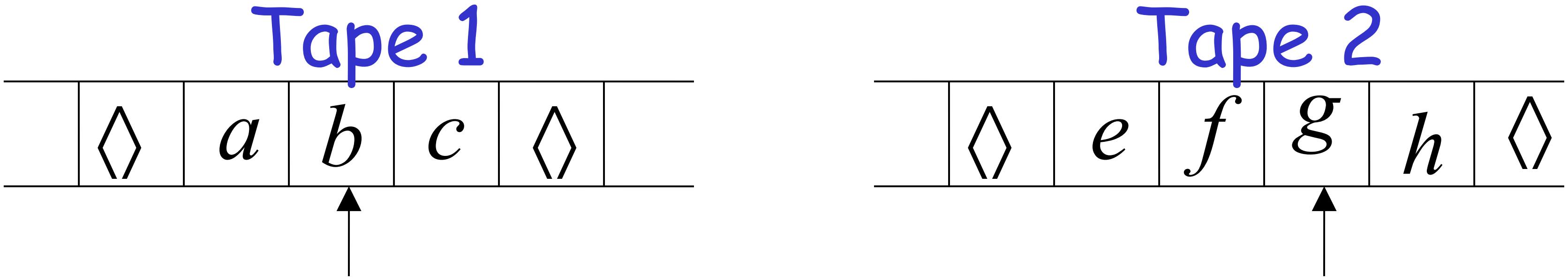
Trivial: Use just one tape

2. Standard Turing machines simulate Multi-tape machines:

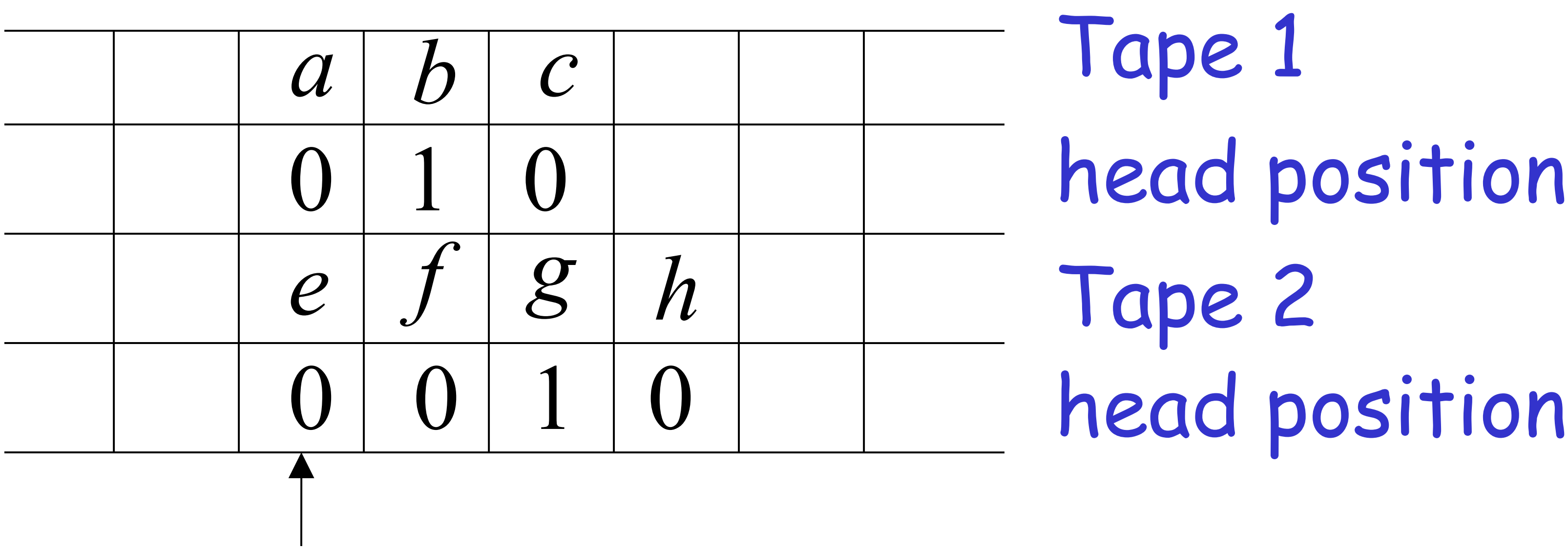
Standard machine:

- Uses a multi-track tape to simulate the multiple tapes
- A tape of the Multi-tape machine corresponds to a pair of tracks

Multi-tape Machine



Standard machine with four track tape



Reference point

	#	a	b	c			
	#	0	1	0			
	#	e	f	g	h		
	#	0	0	1	0		

Tape 1

head position

Tape 2

head position

Repeat for each state transition:

1. Return to reference point
2. Find current symbol in Tape 1
3. Find current symbol in Tape 2
4. Make transition

END OF PROOF

Same power doesn't imply same speed:

$$L = \{a^n b^n\}$$

Standard Turing machine: $O(n^2)$ time

Go back and forth $O(n^2)$ times
to match the a's with the b's

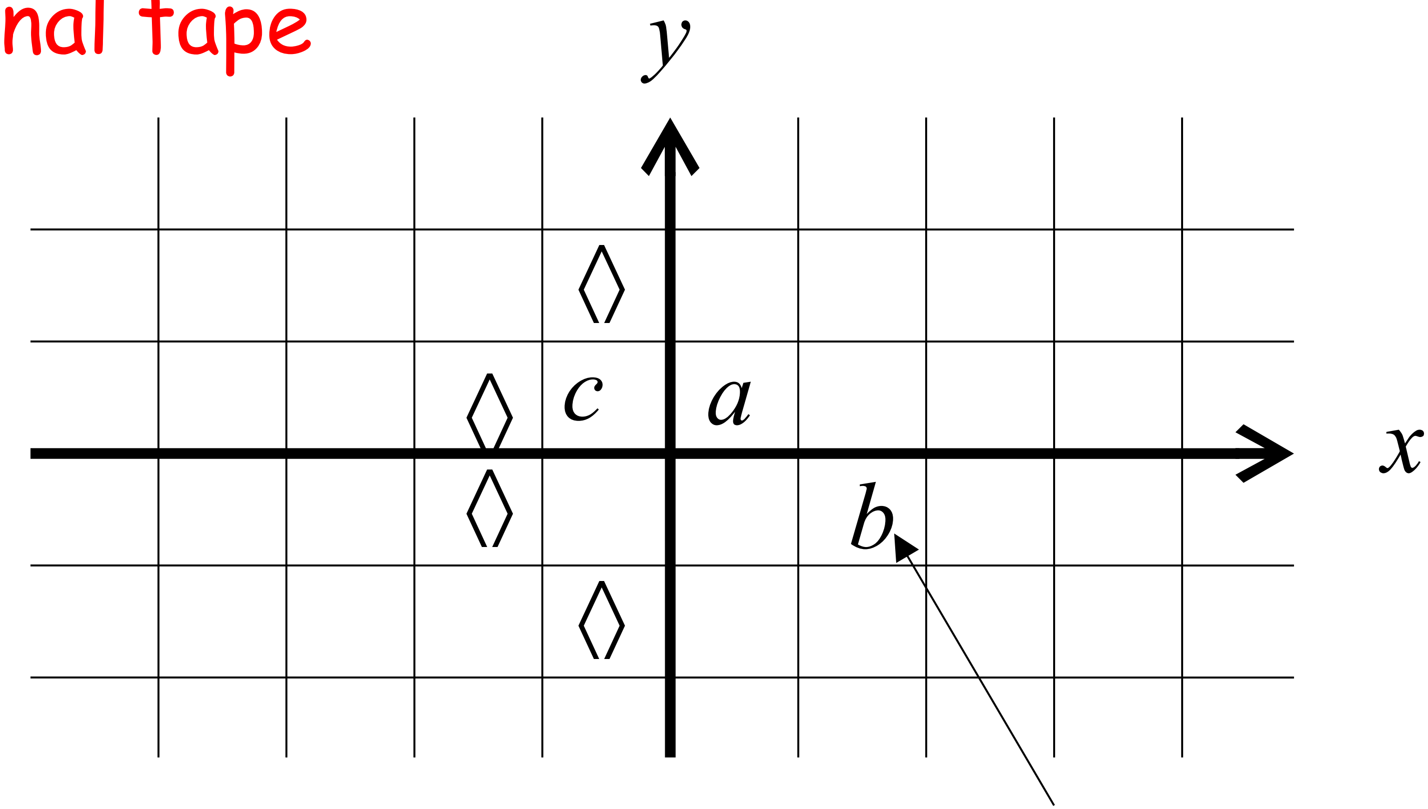
2-tape machine: $O(n)$ time

1. Copy b^n to tape 2 ($O(n)$ steps)

2. Compare a^n on tape 1
and b^n tape 2 ($O(n)$ steps)

Multidimensional Turing Machines

2-dimensional tape



MOVES: L,R,U,D

U: up D: down

HEAD

Position: +2, -1

Theorem: Multidimensional machines have the same power with Standard Turing machines

Proof:

1. Multidimensional machines simulate Standard Turing machines

2. Standard Turing machines simulate Multi-Dimensional machines

1. Multidimensional machines simulate Standard Turing machines

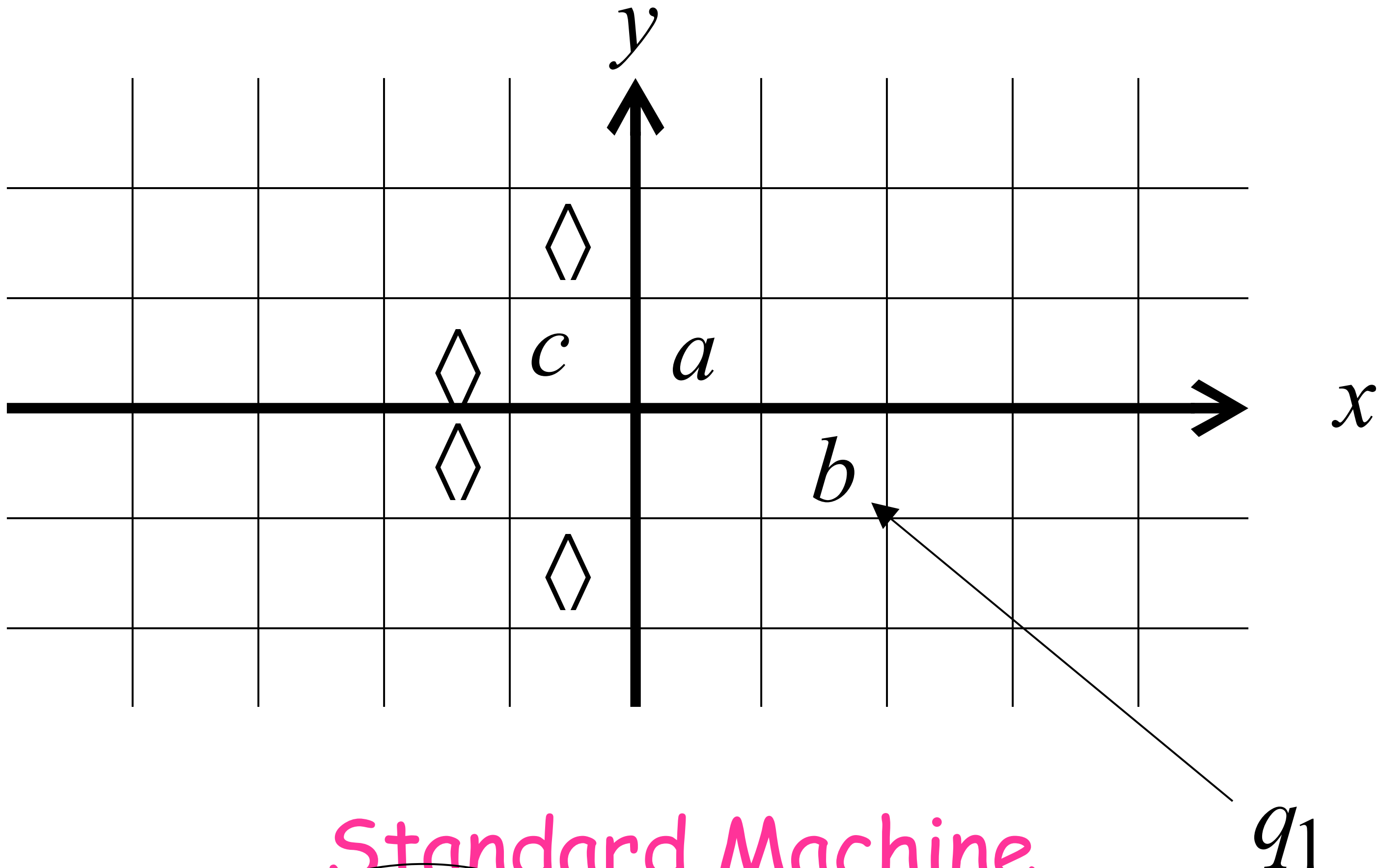
Trivial: Use one dimension

2. Standard Turing machines simulate Multidimensional machines

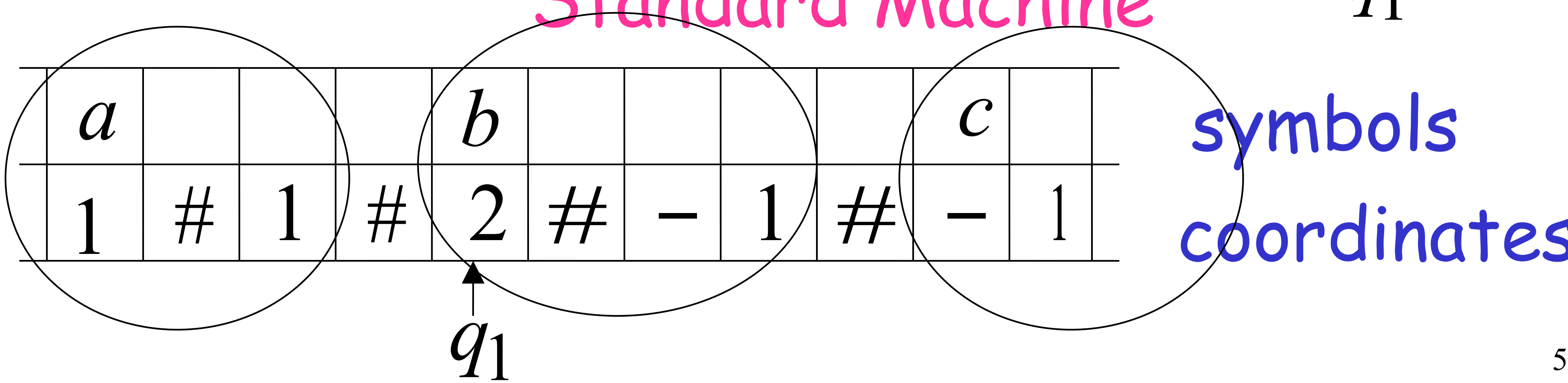
Standard machine:

- Use a two track tape
- Store symbols in track 1
- Store coordinates in track 2

2-dimensional machine



Standard Machine



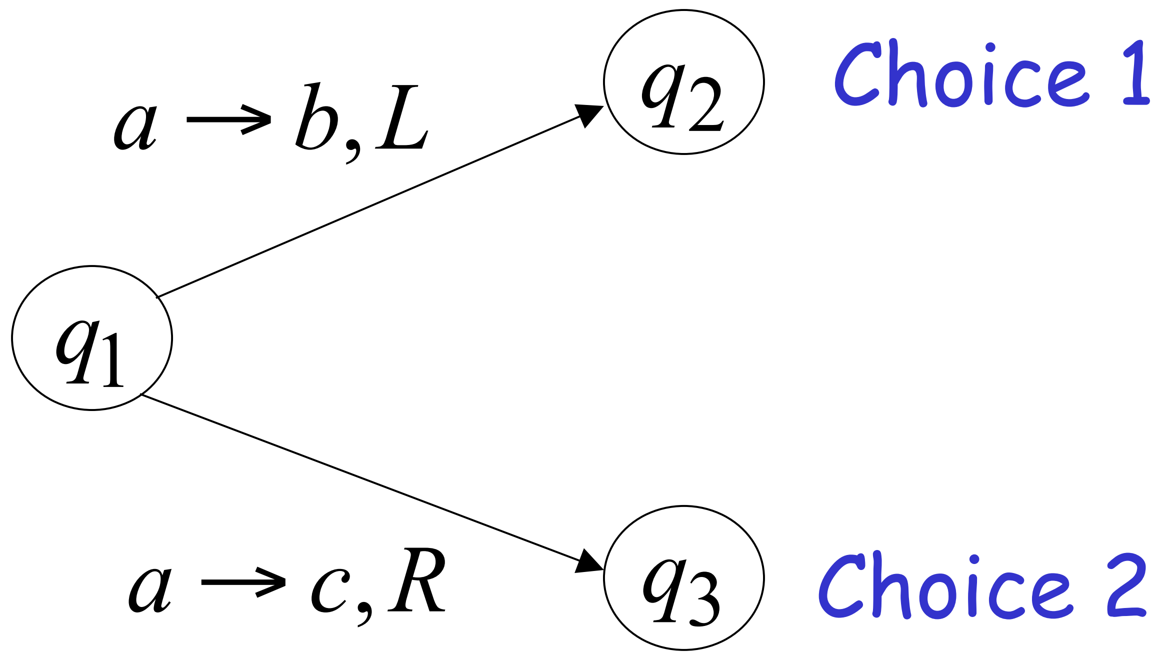
Standard machine:

Repeat for each transition followed
in the 2-dimensional machine:

1. Update current symbol
2. Compute coordinates of next position
3. Go to new position

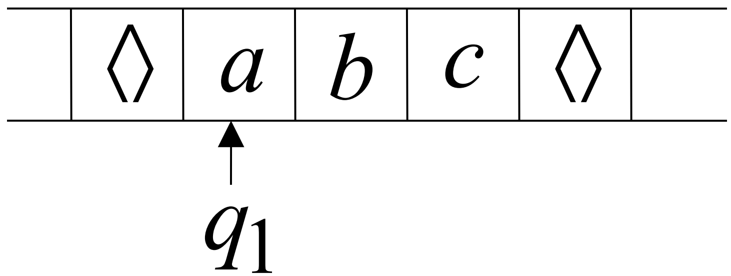
END OF PROOF

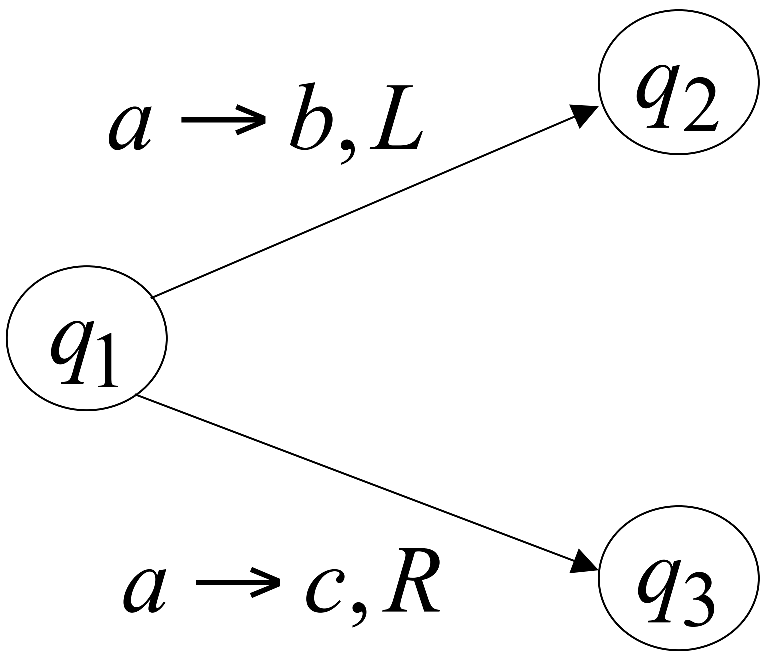
Nondeterministic Turing Machines



Allows Non Deterministic Choices

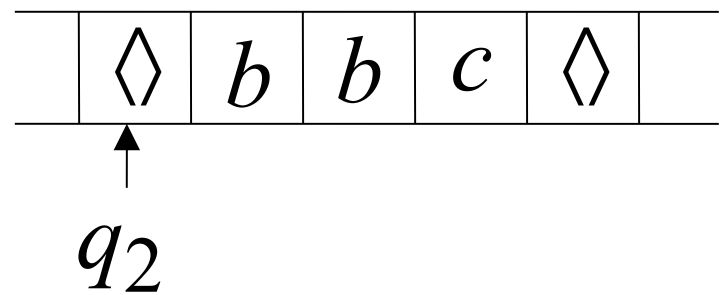
Time 0



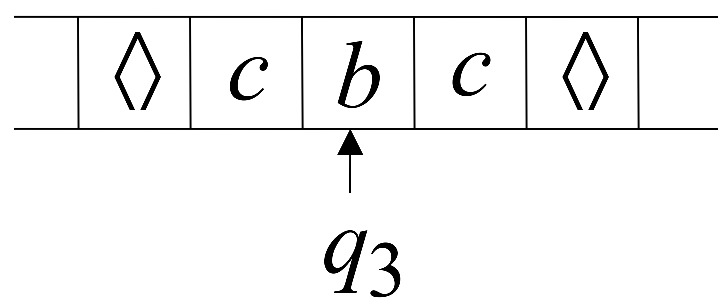


Time 1

Choice 1



Choice 2



Input string w is accepted if there is a computation:

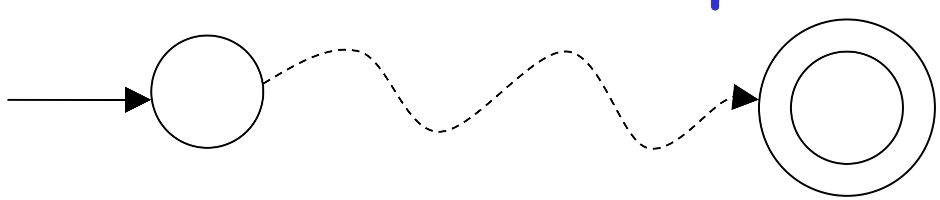
$$q_0 w \overset{*}{\succ} x\, q_f\, y$$

Initial configuration

Final Configuration

Any accept state

There is a computation:



Theorem: Nondeterministic machines have the same power with Standard Turing machines

Proof:

1. Nondeterministic machines simulate Standard Turing machines

2. Standard Turing machines simulate Nondeterministic machines

1. Nondeterministic Machines simulate Standard (deterministic) Turing Machines

Trivial: every deterministic machine is also nondeterministic

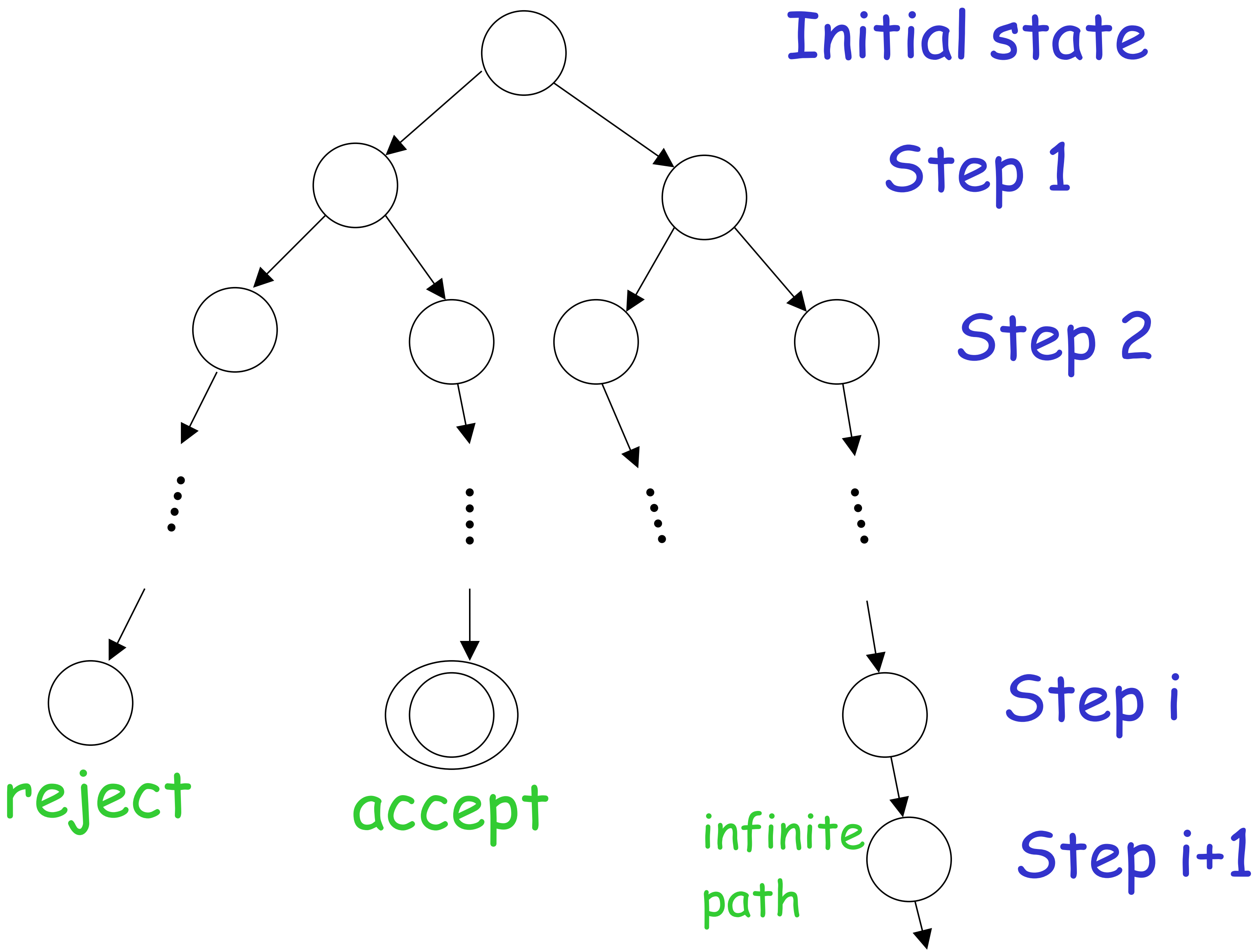
2. Standard (deterministic) Turing machines simulate Nondeterministic machines:

Deterministic machine:

- Uses a 2-dimensional tape
  (which is equivalent to 1-dimensional tape)

- Stores all possible computations of the non-deterministic machine on the 2-dimensional tape

All possible computation paths



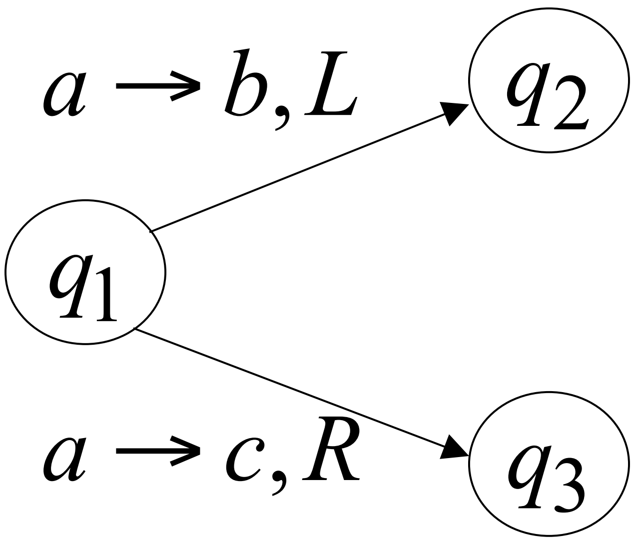
The Deterministic Turing machine simulates all possible computation paths:

- simultaneously

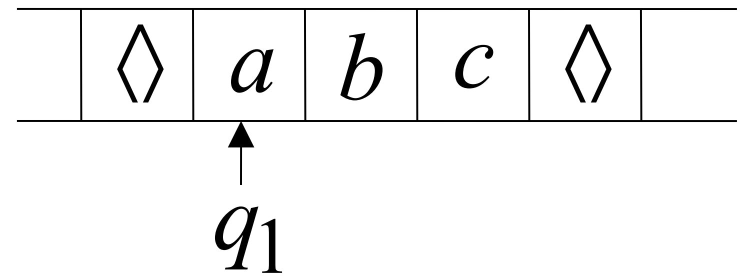
- step-by-step

- in a breadth-first search fashion

NonDeterministic machine



Time 0

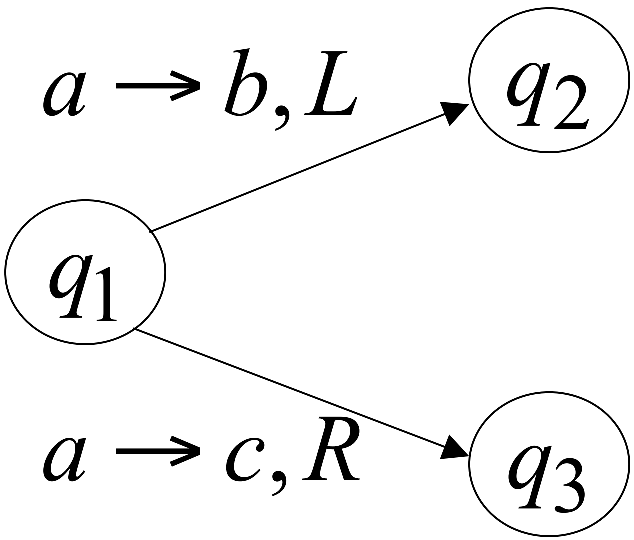


Deterministic machine

	#	#	#	#	#	#
	#	a	b	c	#	
	#	q_1			#	
	#	#	#	#	#	

current configuration

NonDeterministic machine



Time 1

	◊	b	b	c	◊	
	$\uparrow q_2$					

Choice 1

	◊	c	b	c	◊	
			$\uparrow q_3$			

Choice 2

Deterministic machine

	#	#	#	#	#	#		
#		b	b	c	#			Computation 1
#	q_2				#			
#		c	b	c	#			Computation 2
#			q_3		#			

Deterministic Turing machine

Repeat

For each configuration in current step of non-deterministic machine,

if there are two or more choices:

1. Replicate configuration
2. Change the state in the replicas

Until either the input string is accepted or rejected in all configurations

END OF PROOF

Remark:

The simulation takes in the worst case exponential time compared to the shortest accepting path length of the nondeterministic machine

Universal Turing Machine (UTM)

A limitation of Turing Machines:

Turing Machines are "hardwired"

they execute only one program

Real Computers are re-programmable

Solution: Universal Turing Machine

Attributes:

- Reprogrammable machine
- Simulates any other Turing Machine

Universal Turing Machine

simulates any Turing Machine M

Input of Universal Turing Machine:

Description of transitions of M

Input string of M

How a Universal Turing Machine Works

A UTM takes two things as input:

1. **Encoding of another TM** (let's call it TM_1)
    - This encoding includes TM_1's states, transition rules, alphabets, etc.
    - Think of it as TM_1's "program code."
2. **Input string** for TM_1
    - What TM_1 is supposed to process.

The UTM then simulates TM_1 **step-by-step**, exactly as TM_1 would do.

Your laptop is a Universal Turing Machine

Your laptop:

- doesn't only run one program
- can run Chrome, Word, VLC, games, compilers, etc.
- because each program is **just data** stored in memory

A Universal Turing Machine:

- doesn't only compute one function
- it can compute *any* computable function
- because each TM description is **just data** written on its tape

This "program-as-data" concept is the foundation of all modern software systems.

Why Universal Turing Machines Matter

1. **Basis of the stored-program architecture**

   Modern CPUs use memory to store:
   - instructions, and
   - data

   This is exactly what a UTM does.

2. **Shows that one machine can do anything computable**

   You don't need infinite machines for infinite tasks—just **one universal one**.

3. **Leads to the definition of computability**

   Anything that any TM can do
   → can be done by a UTM.

4. **Foundation for programming languages**

   Anything you can program in C, Python, Java, etc.
   → can be simulated by a Universal Turing Machine.

Structure of a Universal Turing Machine

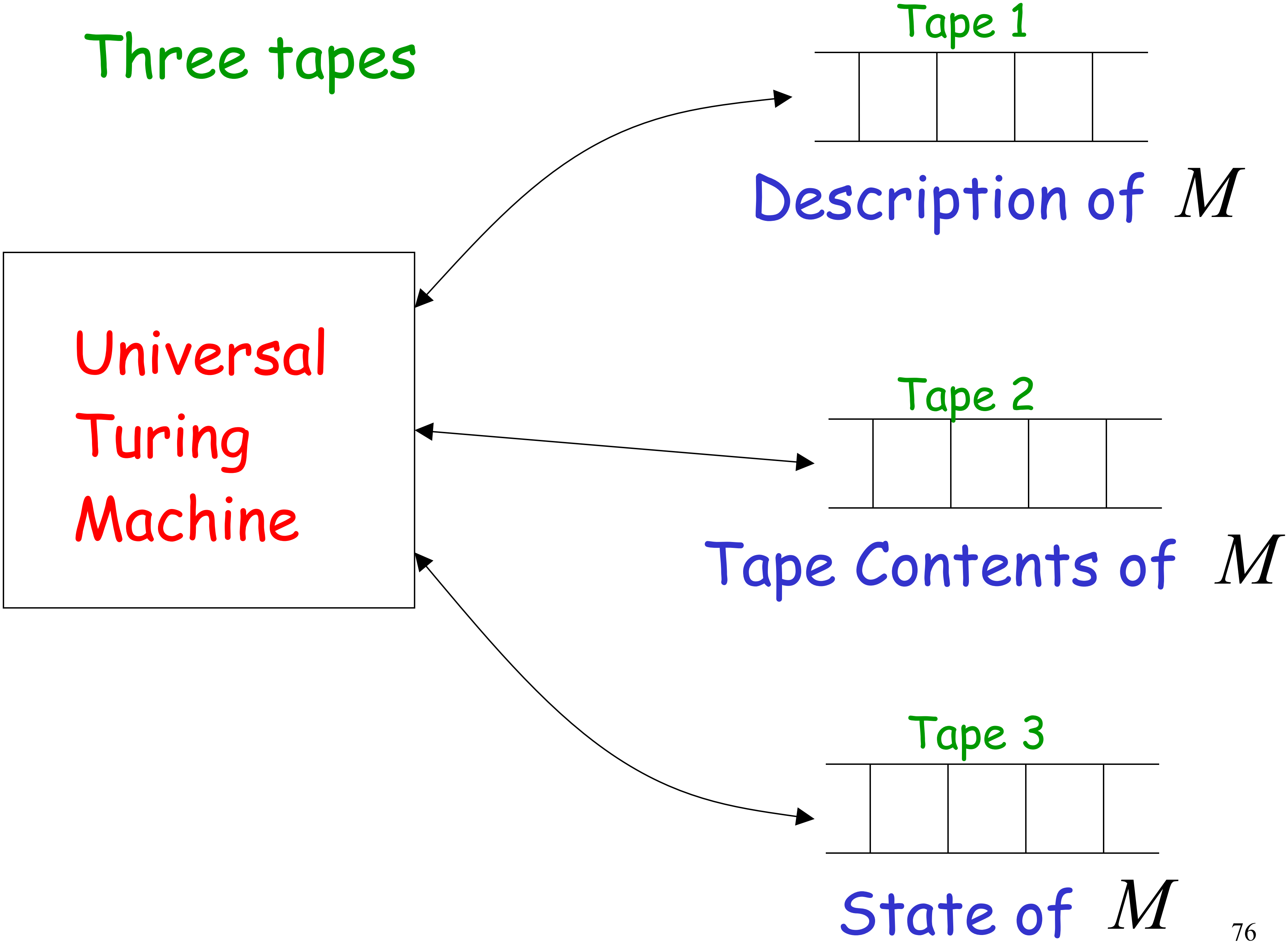
A UTM typically has:

- **multiple tracks** or encoded regions of tape
- one area stores the *program* (the description of TM_1)
- another stores the *input*
- another stores TM_1's *current simulated configuration*

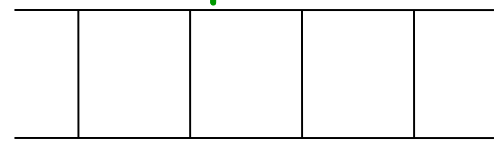
At each step, the UTM:

1. looks up the transition rules of TM_1
2. applies the rule
3. updates the simulated tape and head positions
4. moves to the next simulated step

It's like writing an interpreter for another language.


Three tapes
Universal Turing Machine
Tape 1
Description of M
Tape 2
Tape Contents of M
Tape 3
State of M

Tape 1



Description of M

We describe Turing machine M
as a string of symbols:

We encode M as a string of symbols

Alphabet Encoding

Symbols:	a	b	c	d	$\cdots$
Encoding:	1	11	111	1111	

State Encoding

States:	q_1	q_2	q_3	q_4	$\cdots$
Encoding:	1	11	111	1111	

Head Move Encoding

Move:	L	R
Encoding:	1	11

Transition Encoding

Transition: $\delta(q_1, a) = (q_2, b, L)$

Encoding: $1\,0\,1\,0\,11\,0\,11\,0\,1$

separator

Turing Machine Encoding

Transitions:

$$\delta(q_1, a) = (q_2, b, L) \qquad \delta(q_2, b) = (q_3, c, R)$$

Encoding:

$$1\,0\,1\,0\,11\,0\,11\,0\,1 \;\; 00 \;\; 11\,0\,11\,0\,111\,0\,111\,0\,11$$

separator

Tape 1 contents of Universal Turing Machine:

binary encoding
of the simulated machine M

Tape 1

1 0 1 0 11 0 11 0 10011 0 1 10 111 0 111 0 1100…

A Turing Machine is described with a binary string of 0's and 1's

Therefore:

The set of Turing machines forms a language:

each string of this language is the binary encoding of a Turing Machine

Language of Turing Machines

L = { 010100101, (Turing Machine 1)

00100100101111, (Turing Machine 2)

111010011110010101, ...

...... }