

TURING MACHINE

5/3/2020

Why Turing Machines?

- Observations from the two computation models (FA and PDA):
 - Limited amount of memory (can remember only their current state).
 - The way the memory is accessed (can access only from Top of Stack).

Why Turing Machines?

- This prompts us to introduce a more powerful model of automata approach, which recognizes more languages than a PDA, called *Turing Machine*.
- It was proposed by Alan Turing in 1936.



Alan Turing (1912-1954)
<http://www.turing.org.uk/>

Turing Machine

A Turing Machine (TM) is essentially a finite state machine with the added ability to reread its input and also to erase and write over its input. It also has unlimited auxiliary memory.

Unlimited auxiliary memory makes the Turing Machine a hypothetical machine and not a real device.

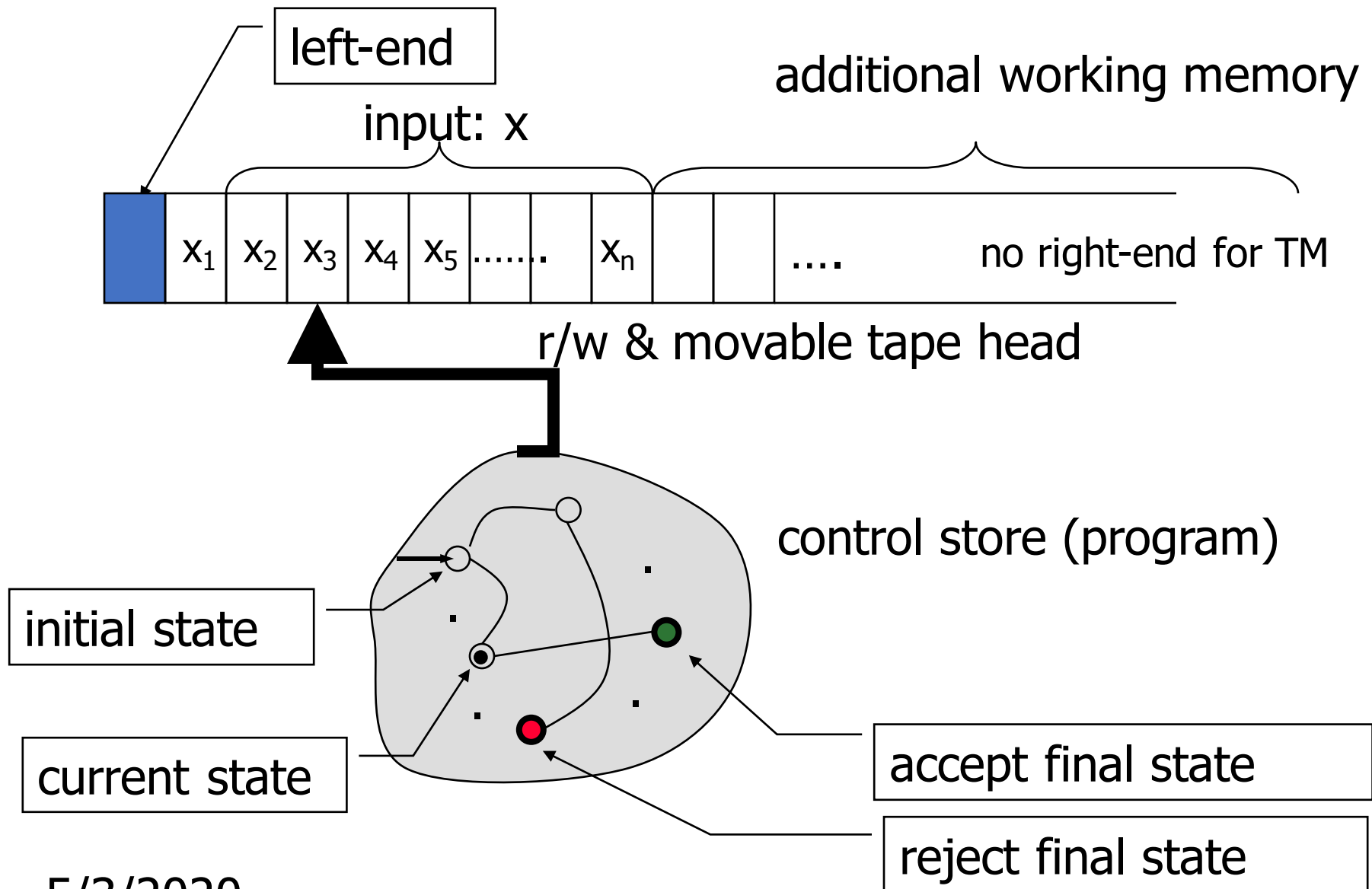


5/3/2020

Comparison with Previous Models

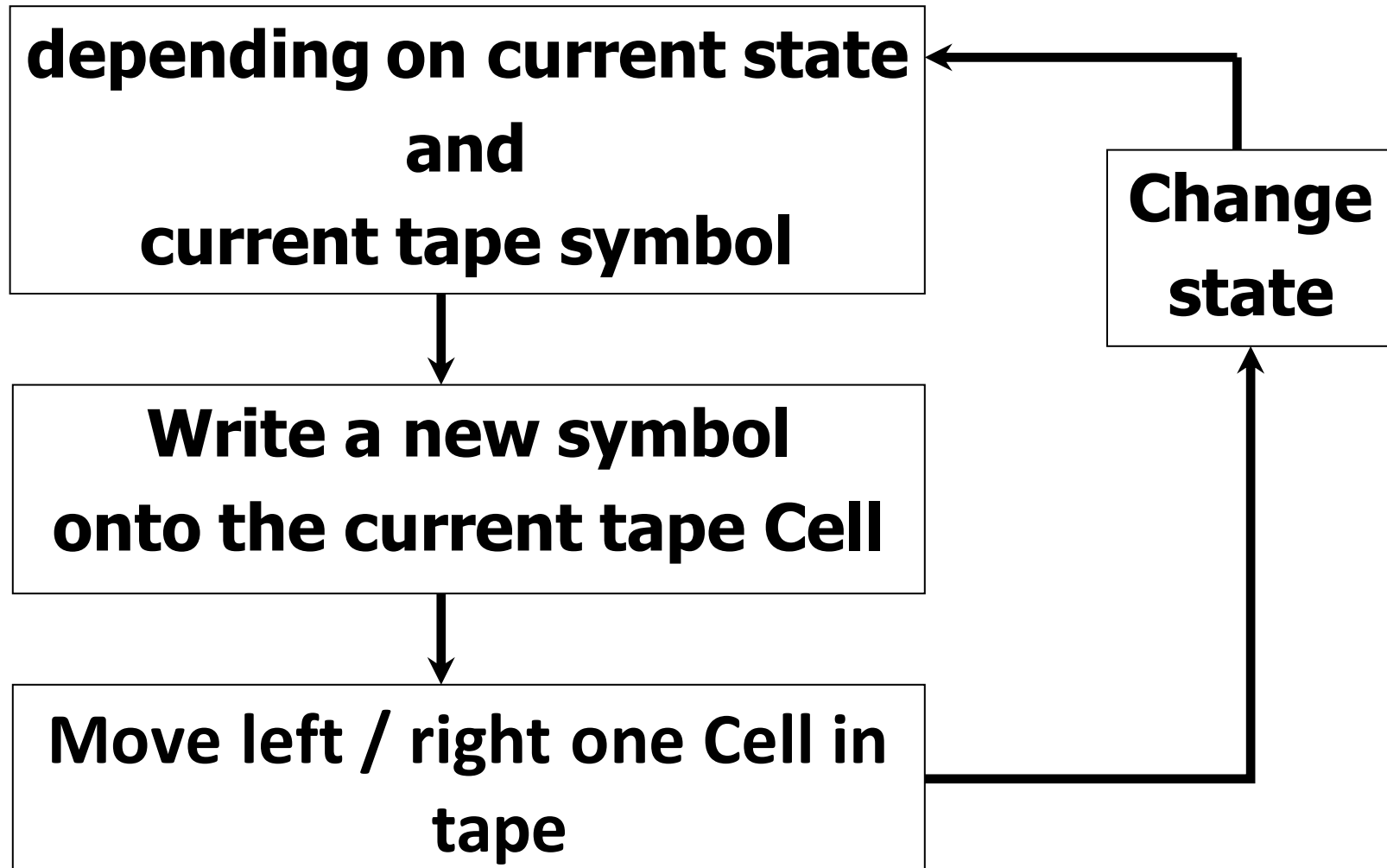
Device	Separate Input?	Read/Write Data Structure	Deterministic by default?
FA	Yes	None	Yes
PDA	Yes	LIFO Stack	No
TM	No	1-way infinite tape. 1 cell access per step.	Yes (but will also allow crashes)

Components of Turing Machine



5/3/2020

How Turing Machine operate?



Turing Machine - Formal Definition

- Finite automata is a 7-tuple system defined as:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

Q A finite set of states

Γ A finite tape alphabet

B A distinguished blank symbol, which is in Γ

Σ A finite input alphabet, which is a subset of $\Gamma - \{B\}$

q_0 The initial/starting state, q_0 is in Q

F A set of final states, which is a subset of Q i.e. q_{accept} and q_{reject}

δ A transition function, which is a *mapping* from

$$Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, H\}$$

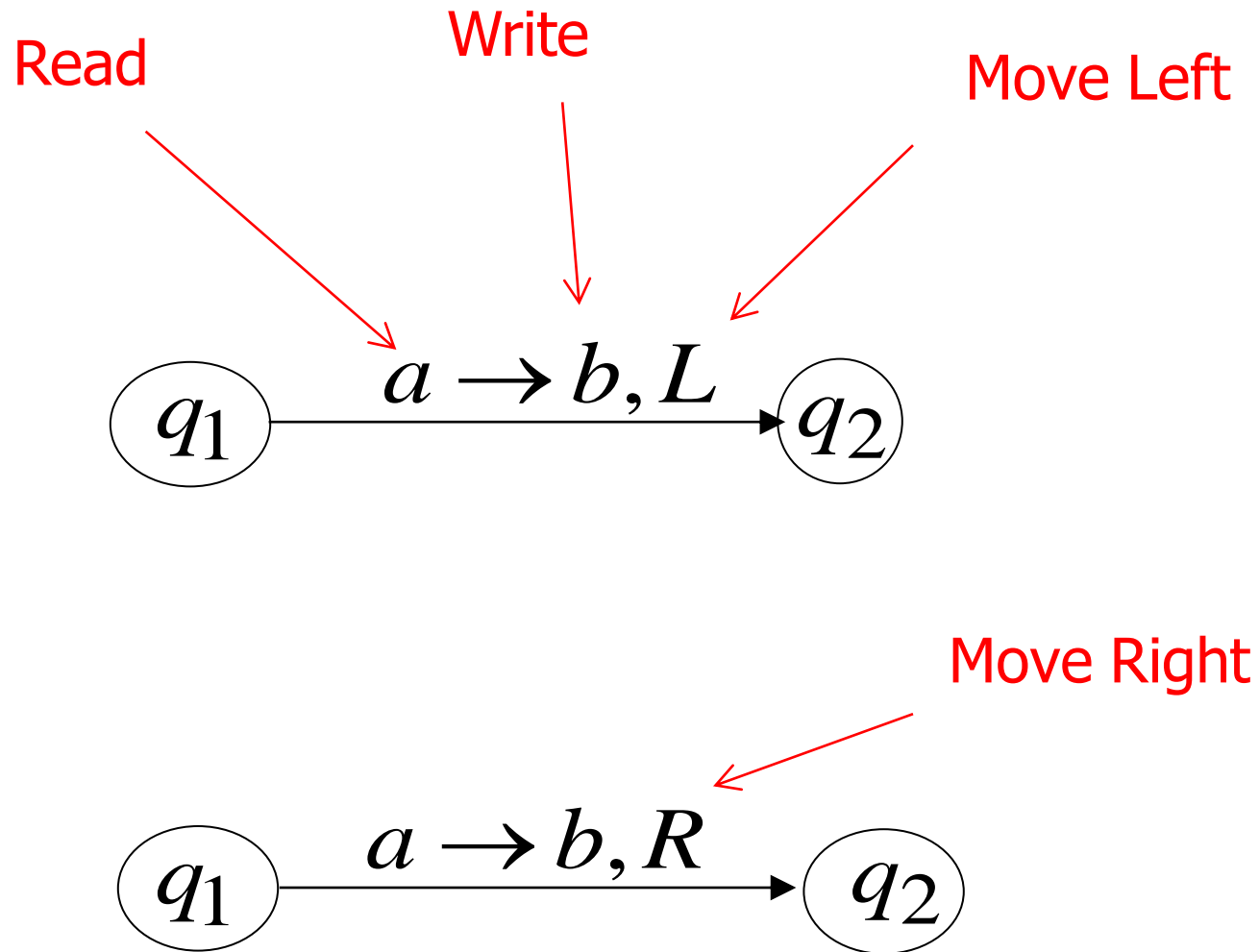
For elements $q, r \in Q$; $X, Y \in \Gamma$; $D \in \{R, L, H\}$; the formula $\delta(q, X) = \{r, Y, D\}$ specifies the next state (r), symbol to be written (Y), and the direction of tape head movement (D) by M after reading symbol X while in state q .

Representation of Turing Machine

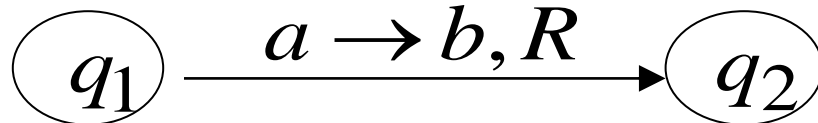
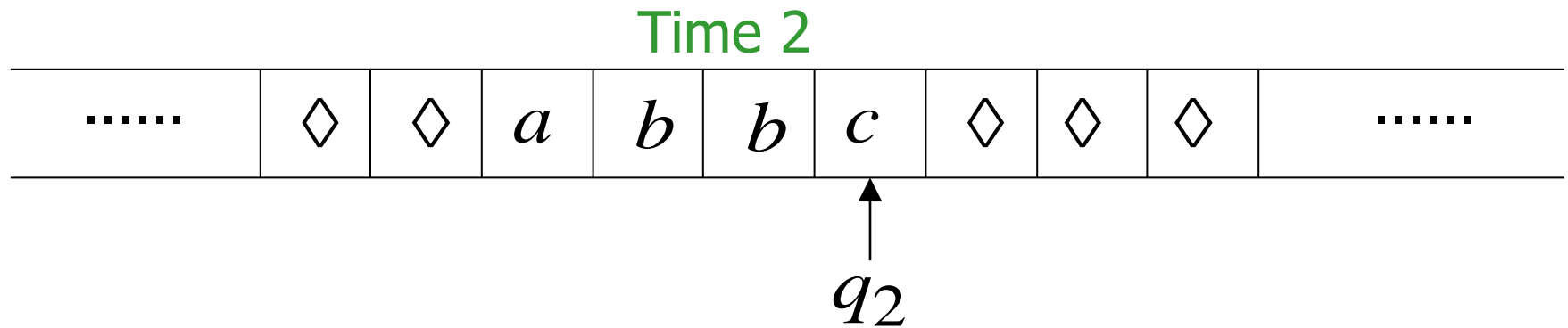
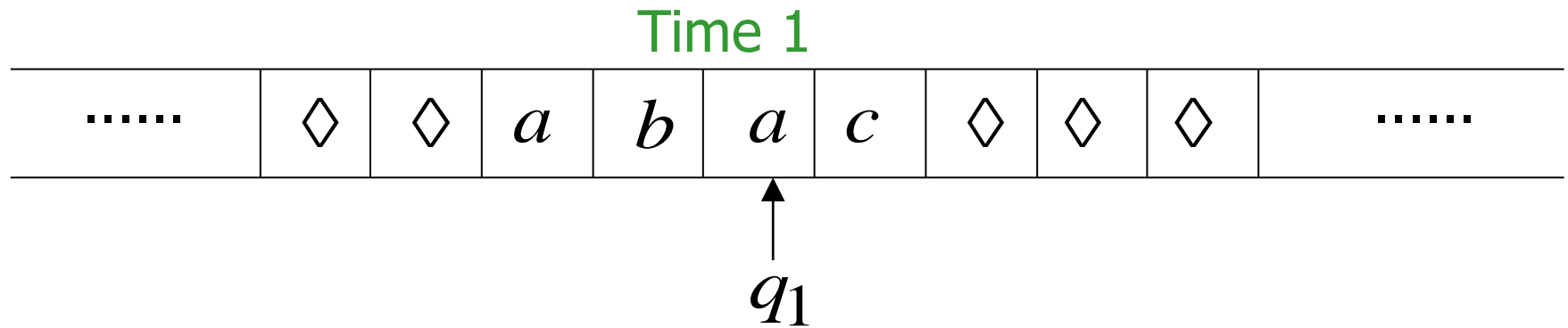
There are two different to describe the task of a Turing Machine:

- Transition Diagram
- Transition Table

Transition Diagram



Example:



Transition Table

state	For all Tape Symbols (a)		
q_i	q_j, b, D		

Where:

q_i is the current state

q_j is the next state

a is the current symbol

b is the symbol to be write

D is the direction of move

Important Definitions

- Instantaneous Description (ID) or Configuration of TM
- Moves of a TM
- Language Accepted by TM

ID of Turing Machine

◆ Let $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ be a TM.

◆ **Definition:** An *instantaneous description* (ID) is a triple

$$X q Y$$

where:

- q , the current state, is in Q
- X, Y is in Γ^* , and is the current tape contents up to the rightmost non-blank symbol, or the symbol to the left of the tape head, whichever is rightmost
- The tape head is currently scanning the first symbol of Y

◆ The start configuration of M on input w : $q_0 w$

◆ The final configuration of M on input w : $w q_{\text{accept}}$ or $w q_{\text{reject}}$

Moves of Turing Machine

◆ Suppose the following is the current ID of a DTM

$$x_1x_2\dots x_{i-1} q x_i x_{i+1} \dots x_n$$

Case 1) $\delta(q, x_i) = (p, y, L)$

(a) if $i = 1$ then $qx_1x_2\dots x_{i-1}x_ix_{i+1}\dots x_n \mid \rightarrow pByx_2\dots x_{i-1}x_ix_{i+1}\dots x_n$

(b) else $x_1x_2\dots x_{i-1}qx_ix_{i+1}\dots x_n \mid \rightarrow x_1x_2\dots x_{i-2}px_{i-1}yx_{i+1}\dots x_n$

■ If any suffix of $x_{i-1}yx_{i+1}\dots x_n$ is blank then it is deleted.

Case 2) $\delta(q, x_i) = (p, y, R)$

$$x_1x_2\dots x_{i-1}qx_ix_{i+1}\dots x_n \mid \rightarrow x_1x_2\dots x_{i-1}ypx_{i+1}\dots x_n$$

■ If $i > n$ then the ID increases in length by 1 symbol

$$x_1x_2\dots x_nq \mid \rightarrow x_1x_2\dots x_nyp$$

Language Acceptance

- **Definition:** Let $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ be a TM, and let w be a string in Σ^* . Then w is *accepted* by M iff

$$q_0 w \vdash^* X p Y$$

where p is in F and X and Y are in Γ^*

- **Definition:** Let $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ be a TM. The *language accepted by M* , denoted $L(M)$, is the set

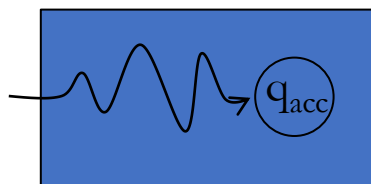
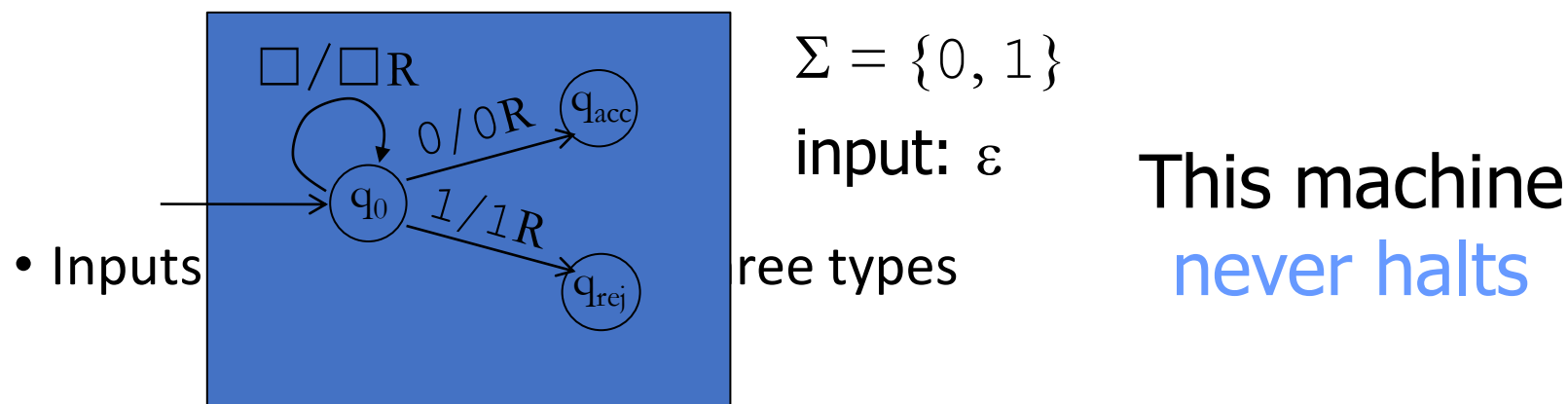
$$\{w \mid w \text{ is in } \Sigma^* \text{ and } w \text{ is accepted by } M\}$$

Language Acceptance

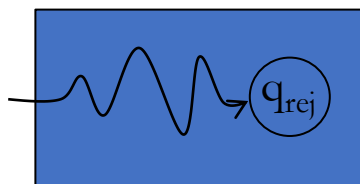
- **Turing Acceptable Language:** A Language over some alphabet is said to be Turing Acceptable, if there exists a TM M such that $L=L(M)$.
- **Turing Decidable Language:** A Language over some alphabet is said to be Turing Decidable, if both the language and its complement are Turing acceptable.

Important observations on TM

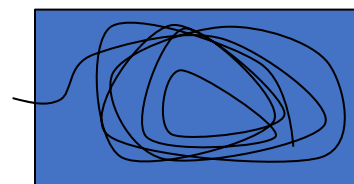
- Something strange can happen in a Turing Machine:



5/3/2020 accept



reject



loop

Role of Turing Machine

- Functions of a Turing Machine
 - **Function #1**
As a Language Acceptor / Recognizer
 - **Function #2**
As a Computing Device (Transducer)

Function 1: Language Acceptor

Three possibilities occur on a given input w :

1. The TM M eventually enters q_{acc} and therefore halts and accepts. ($w \in L(M)$)
2. The TM M eventually enters q_{rej} *or* crashes somewhere. M **rejects** w . ($w \notin L(M)$)
3. Neither occurs! I.e., M never halts its computation and is caught up in an ***infinite loop***, never reaching q_{acc} or q_{rej} . In this case w is neither accepted nor rejected. However, any string not explicitly accepted is considered to be outside the accepted language. ($w \notin L(M)$)

Description of Turing Machine for $L_1 = \{w\#w: w \in \{a, b\}^*\}$

1 **Until** you reach #

2 **Read** and remember entry

3 **Write** x

4 **Move** right past # and past all xs

5 **If** this entry is different, **reject**

Otherwise

6 **Write** x

7 **Move** left past # and to right of first x

8 **If** you see only xs followed by \square , accept

xbbaa#xbbaa

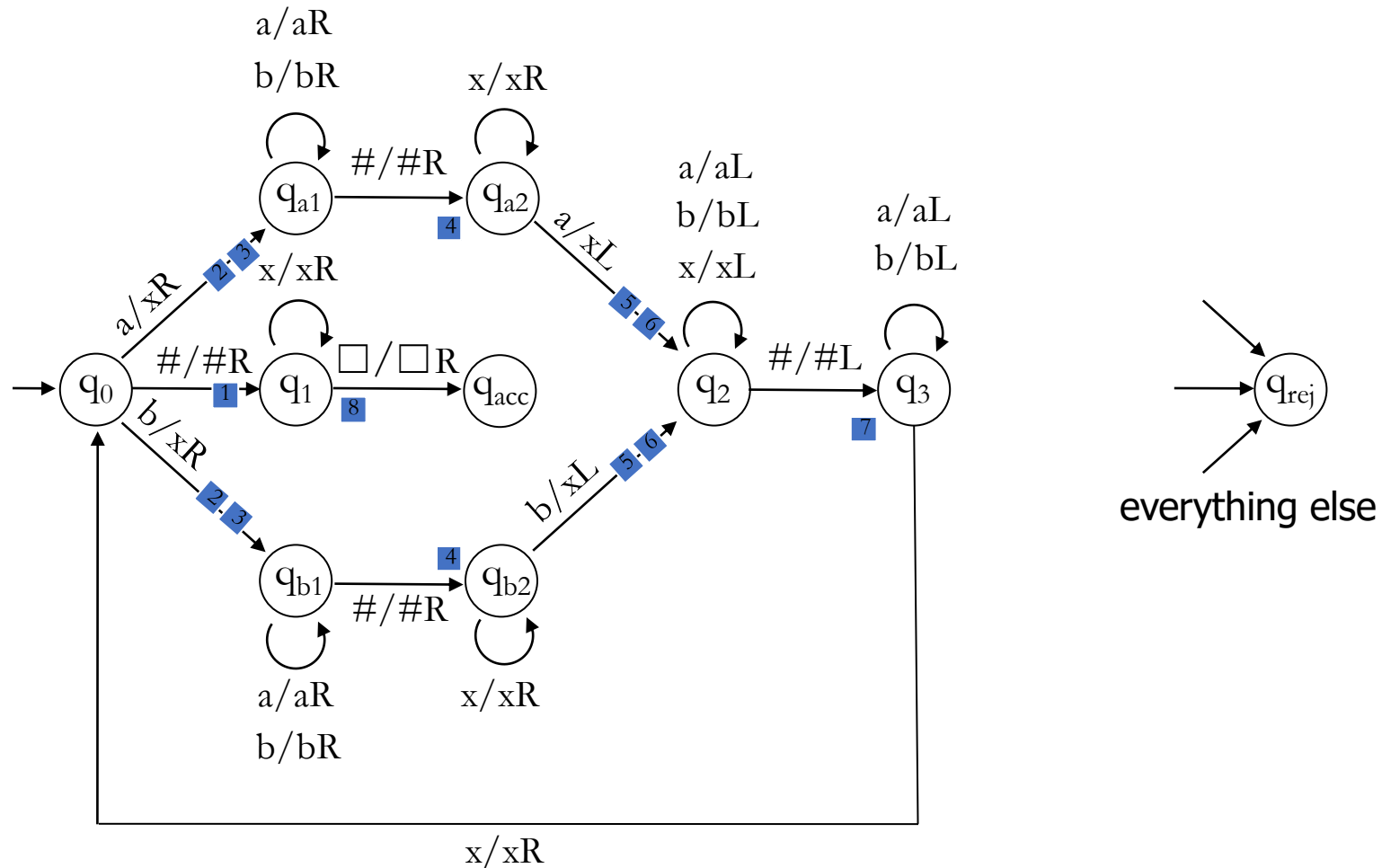
xxbaa#xbbaa

xxbaa#xbbaa

xxbaa#xxbaa

xxbaa#xxbaa

Description of Turing Machine for $L_1 = \{w\#w: w \in \{a, b\}^*\}$



Function 2: Computing Device

- A function $f: N \rightarrow N$ is said to be computable function of k arguments if there exists a Turing machine M halts with a tape consisting of 0^m for some m , where $f(i_1, i_2, \dots, i_k) = m$.
- A TM may compute a function of one argument, a different function of two arguments and so on. If $f(i_1, i_2, \dots, i_k)$ is defined for all (i_1, i_2, \dots, i_k) then we say that f is a **total recursive function**.

Description of Turing Machine for $f(x)=x+1$

Unary representation is used in order to do any arithmetic operations on a TM.

Unary representation looks as follows

$0 \rightarrow 1$
 $1 \rightarrow 11$
 $2 \rightarrow 111$
 $3 \rightarrow 1111$
 $4 \rightarrow 11111$

Example: Design a Turing Machine to add 1 to any number
start in state 1

if the state is 1 and current input is 1, write 1 and move right and stay in state 1

if the current state is 1 and current input is b, write 1 and move to state 2 and move right and HALT

TM instructions:

$(1, 1, 1, R, 1)$

$(1, b, 1, R, 2)$

state 2 does not exist, so halt

Let the initial configuration be: $\dots b 1 1 1 b \dots$

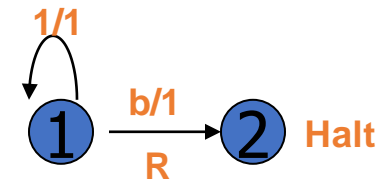
s1 $\dots b \underline{1} 1 1 b \dots$ 2 in unary representation

s1 $\dots b 1 \underline{1} 1 b \dots$

s1 $\dots b 1 1 \underline{1} b \dots$

s1 $\dots b 1 1 1 \underline{b} \dots$

s2 $\dots b 1 1 1 1 \underline{b} \dots$ 3 in unary representation



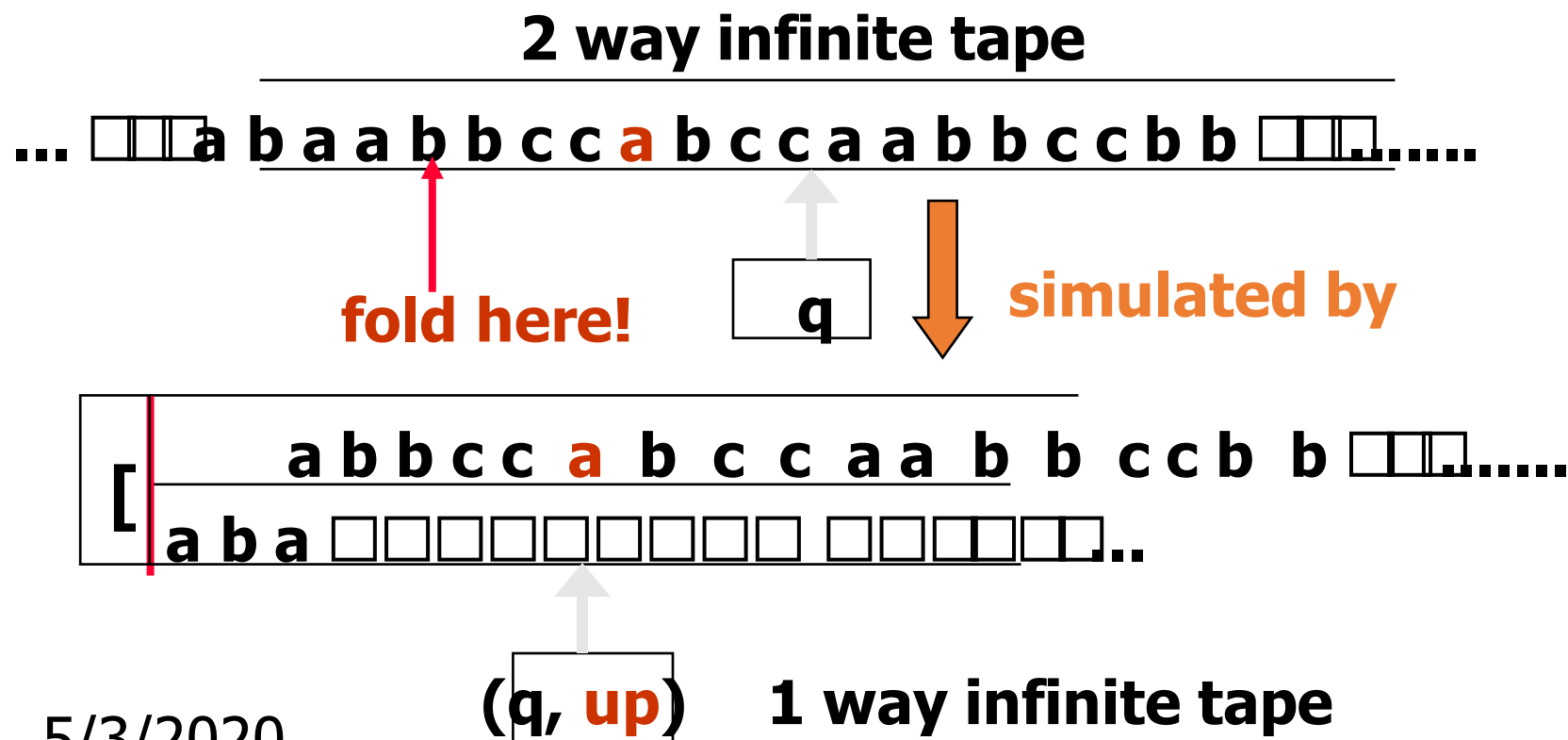
Turing Machines Variants

Variants of TM

- Two-way Infinite tape TM
- Multi-tape TM
- Multi-track TM
- Non-deterministic TM

Two-way Infinite tape TM

A TM in which there is an infinite number of sequences of blanks on either side of the tape is said to be two-way infinite tape TM.



Non-Deterministic TM's

A non-Deterministic Turing Machine N allows more than one possible action per given state-tape symbol pair.

A string w is **accepted** by N if after being put on the tape and letting N run, N eventually enters q_{acc} on *some computation branch*.

If, on the other hand, given any branch, N eventually enters q_{rej} **or** crashes **or** enters an infinite loop on, w is not accepted.

Symbolically as before:

$$L(N) = \{ x \in \Sigma^* \mid \exists \text{ accepting config. } y, q_0 x \Rightarrow^* y \}$$

(No change needed as \Rightarrow need not be function)

Non-Deterministic TM's Recognizers vs. Deciders

N is always called a ***non-deterministic recognizer*** and is said to *recognize* $L(N)$; furthermore, if in addition for all inputs and all computation branches, N always halts, then N is called a ***non-deterministic decider*** and is said to *decide* $L(N)$.

Non-Deterministic TM

Example

Consider the non-deterministic method:

```
void nonDeterministicCrossOut(char c)
    while()
        if (read blank) go left
        else
            if (read c)
                cross out, go right, return
            OR go right    // even when reading c
            OR go left     // even when reading c
```

Non-Deterministic TM

Example

Using randomCross() put together a non-deterministic program:

1. while(some character not crossed out)
 - nonDeterministicCrossOut('0')
 - nonDeterministicCrossOut('1')
 - nonDeterministicCrossOut('2')
2. ACCEPT

Q: What language does this non-deterministic program recognize ?

Non-Deterministic TM

Example

A: $\{x \in \{0,1,2\}^* \mid x \text{ has the same no. of 0's as 1's as 2's} \}$

Q: Suppose q is the state of the TM while running inside `nonDeterministicCrossOut('1')` and q' is the state of the TM inside `nonDeterministicCrossOut('2')`.

Suppose that current configuration is

$$u = 0XX1Xq12X2$$

For which v do we have $u \Rightarrow v$?

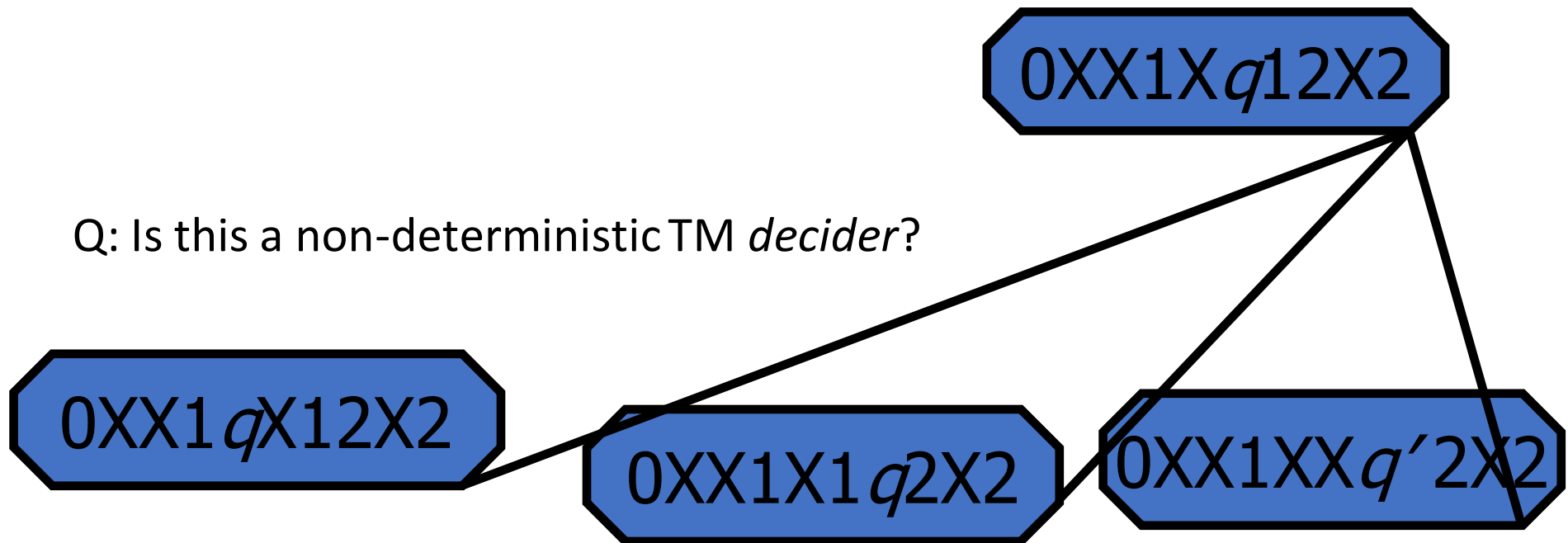
Non-Deterministic TM

Example

A: $0XX1Xq12X2 \Rightarrow$

$0XX1qX12X2 \mid 0XX1X1q2X2 \mid 0XX1XXq'2X2$

These define 3 branches of computation tree:



Q: Is this a non-deterministic TM *decider*?

Non-Deterministic TM

Example

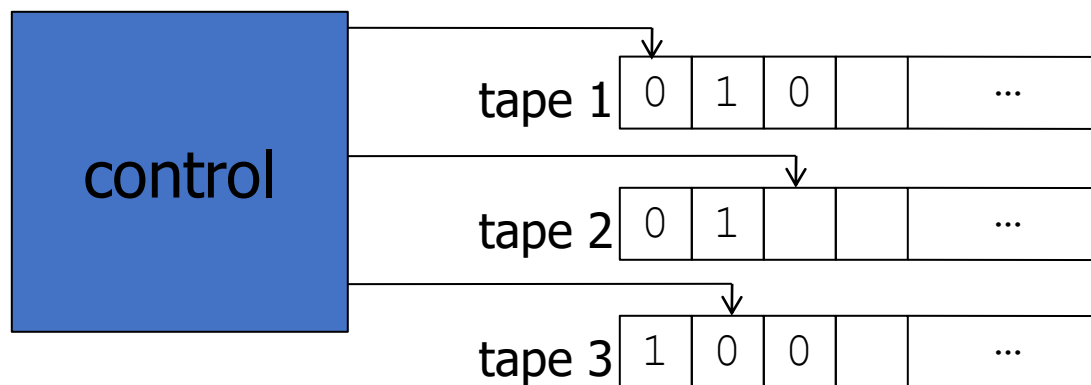
A: No. This is a TM recognizer, but not a decider.

nonDeterministicCrossOut() often enters an infinite branch of computation since can see-saw from right to left to right, etc. ad infinitum without ever crossing out anything. I.e., computation tree is infinite!

Note: If you draw out state-diagrams, you will see that the NTM is more compact, than TM version so there are some advantages to non-determinism! Later, will encounter examples of “efficient” nondeterministic programs for practically important problems, with no known efficient counterpart: The **P** vs. **NP** Problem.

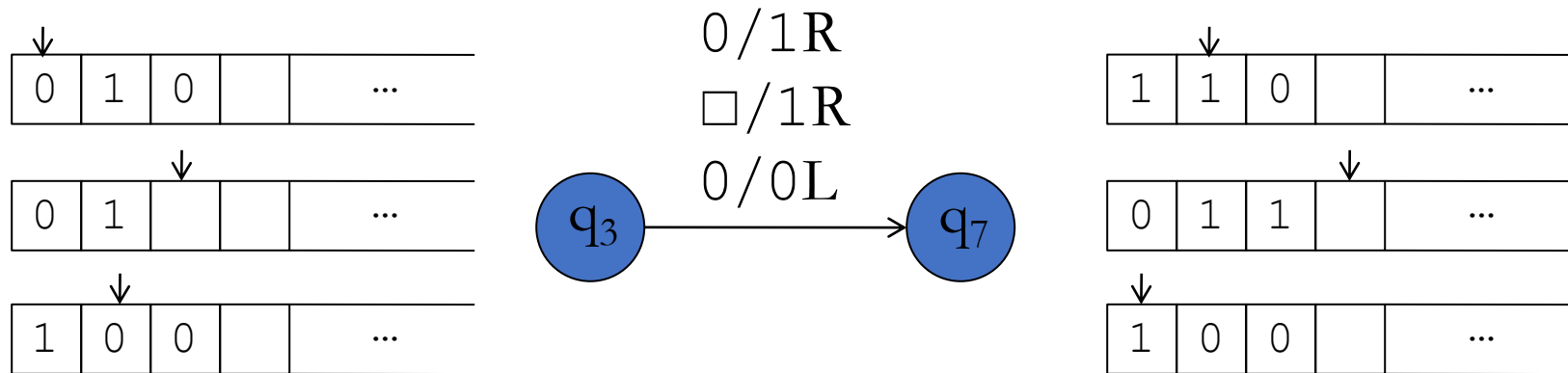
Multi-tape TM's

A TM with more than one tape and each tape having its own independent head is said to be a multi-tape TM.



- ◆ The transition may depend on the contents of all the cells
- ◆ Different tape heads can be moved independently

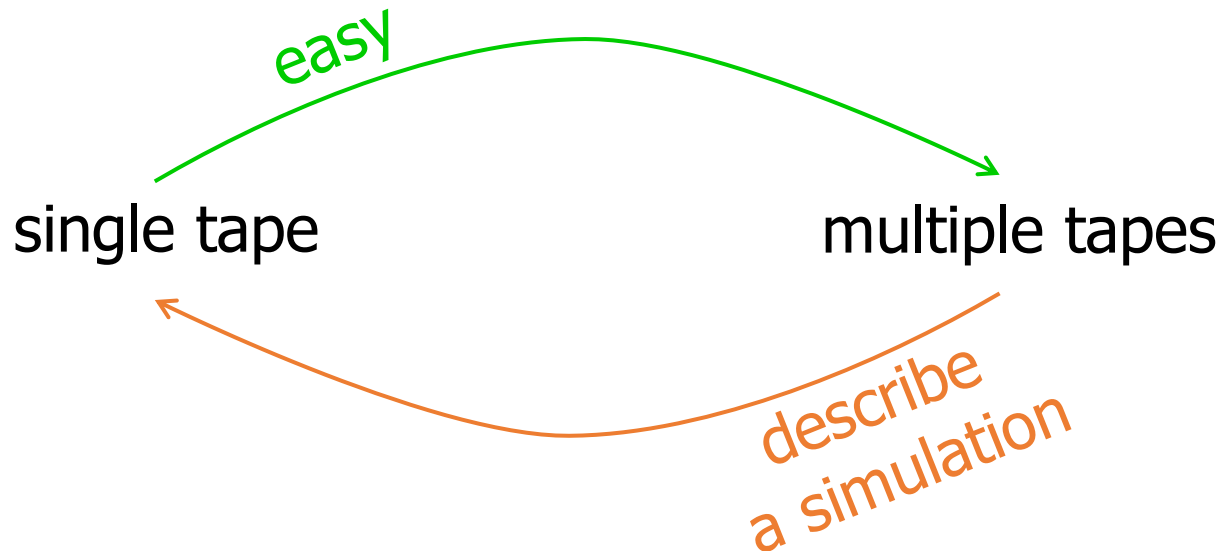
Multi-tape TM's



Multiple tapes are convenient,
e.g. one can serve as temporary storage

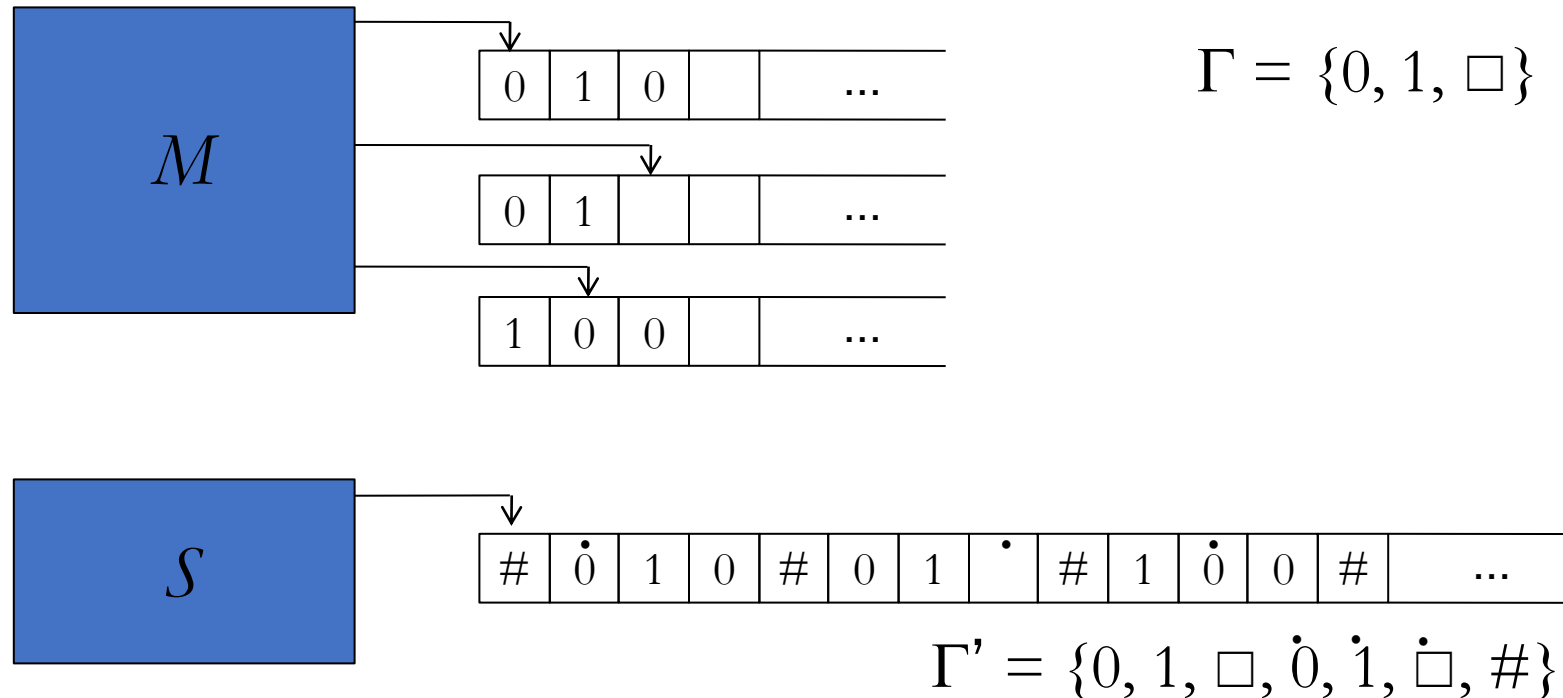
Equivalence of MTM and STM

Multi-tape Turing Machines are **equivalent** to single-tape Turing Machines



Equivalence of MTM and STM

Multitape Turing Machines are **equivalent** to single-tape Turing Machines



Multitape TM's

Formal Notation

NOTE: Sipser's multitape machines cannot pause on one of the tapes as above example. This isn't a problem since pausing 1-tape machines can simulate pausing k -tape machines, and non-pausing 1-tape machines can simulate 1-tape pausing machines by adding dummy R-L moves for each pause.

Formally, the δ -function of a k -tape machine:

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

Multitape TM's Conventions

- Input always put on the first tape
- If I/O machine, output also on first tape
- Can consider machines as “string-vector” generators. E.g., a 4 tape machine could be considered as outputting in $(\Sigma^*)^4$

A Universal Turing Machine

A limitation of Turing Machines:

Turing Machines are “hardwired”



they execute only one program

Solution: UTM

- Reprogrammable machine
- Simulates any other Turing Machine

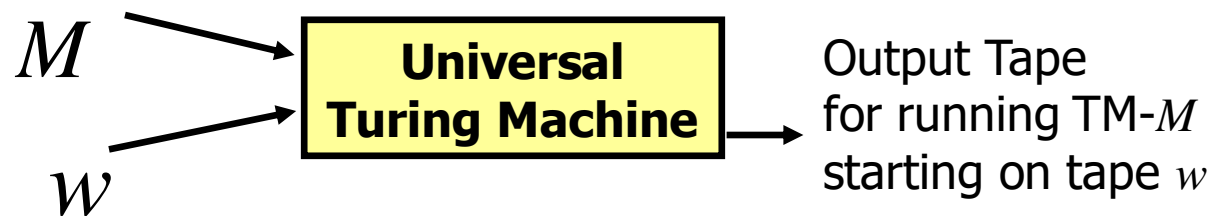
Universal Turing Machine

Universal Turing Machine simulates any Turing Machine M

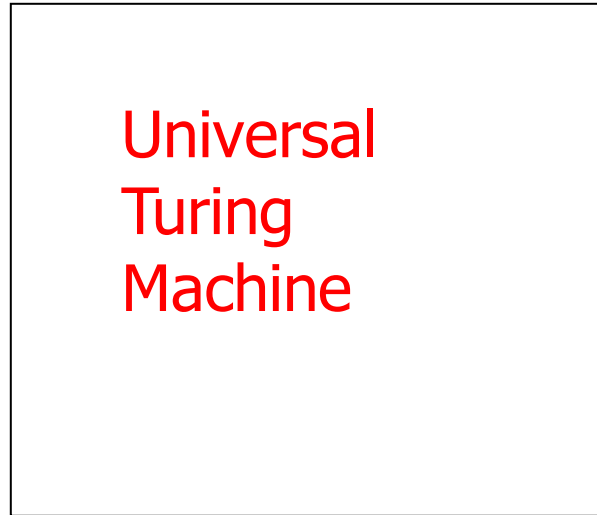
Input of Universal Turing Machine:

Description of transitions of M

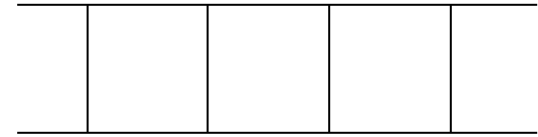
Input string of M



Three tapes



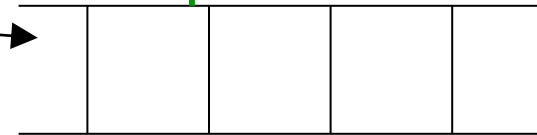
Tape 1



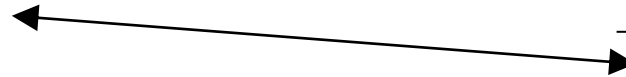
Description of M



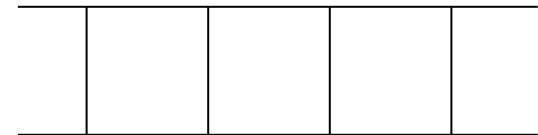
Tape 2



Tape Contents of M







Tape 3



State of M



Alphabet Encoding

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

State Encoding

States:

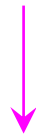
q_1

q_2

q_3

q_4

\dots



Encoding:

1

11

111

1111

Head Move Encoding

Move:

L

R



Encoding:

1

11

5/3/2020

Transition Encoding

Transition:

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

Encoding:

1 0 1 0 1 1 0 1 1 0 1

separator

Turing Machine Encoding

Transitions:

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

$$\delta(q_2, b) = (q_3, c, R)$$

Encoding:

1 0 1 0 1 1 0 1 1 0 1 0 0 1 1 0 1 1 1 0 1 1 1 0 1 1

separator

5/3/2020

Tape 1 contents of Universal Turing Machine: M
binary encoding
of the simulated machine

Tape 1

1 0 1 0 11 0 11 0 10011 0 1 10 111 0 111 0 1100...



5/3/2020

Language of Turing Machines

$L = \{ 010100101,$

$00100100101111,$

$111010011110010101,$

$\dots \}$

(Turing Machine 1)

(Turing Machine 2)

.....