# CS154

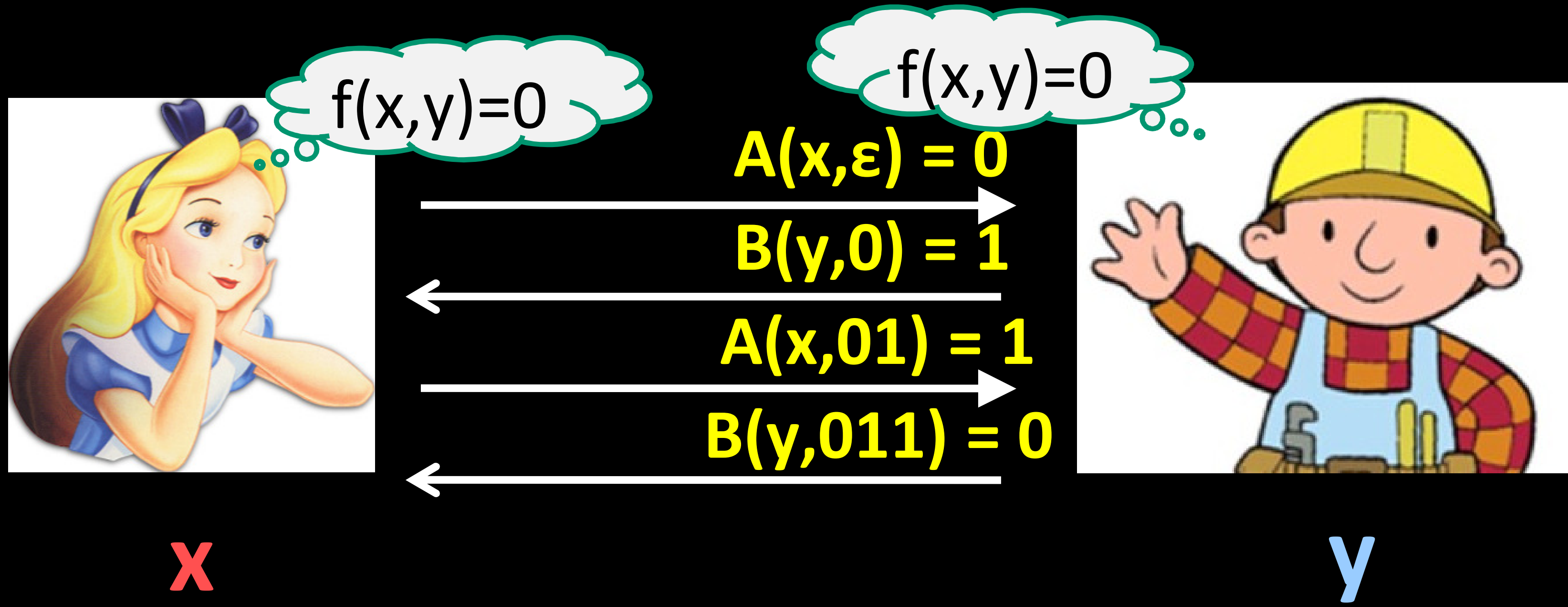
## Lecture 7:
## Finish up Communication,
## Start up Turing Machines

# Communication Complexity

A theoretical model of distributed computing

- **Function $f: \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}$**
  - Two inputs, $x \in \{0,1\}^*$ and $y \in \{0,1\}^*$
  - **We assume $|x|=|y|=n$. Think of $n$ as HUGE**
- **Two computers: Alice** and **Bob**
  - **Alice** *only* knows $x$, **Bob** *only* knows $y$
- **Goal: Compute $f(x, y)$ by communicating as few bits as possible between Alice and Bob**

***We do not count computation cost.*** We *only* care about the number of bits communicated.




**Def.** A *protocol* for a function ***f*** is a pair of functions
A, B : {0,1}* × {0,1}* → {0, 1, STOP} with the semantics:

On input $(x, y)$, let $r$ := 0, $b_0$ = ε.

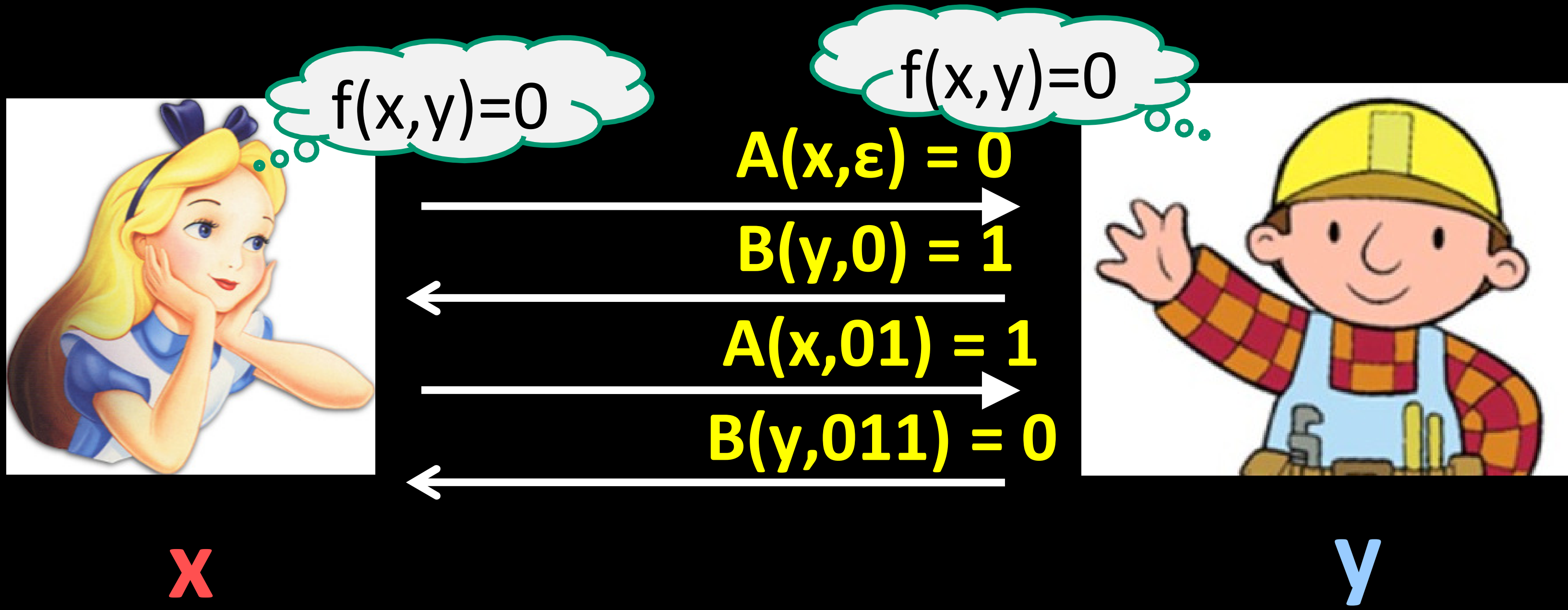
While ($b_r \neq$ STOP),

$r++$

If $r$ is odd, Alice sends $b_r = A(x, b_1 \cdots b_{r-1})$

else Bob sends $b_r = B(y, b_1 \cdots b_{r-1})$

Output $b_{r-1}$. Number of *rounds* $= r - 1$

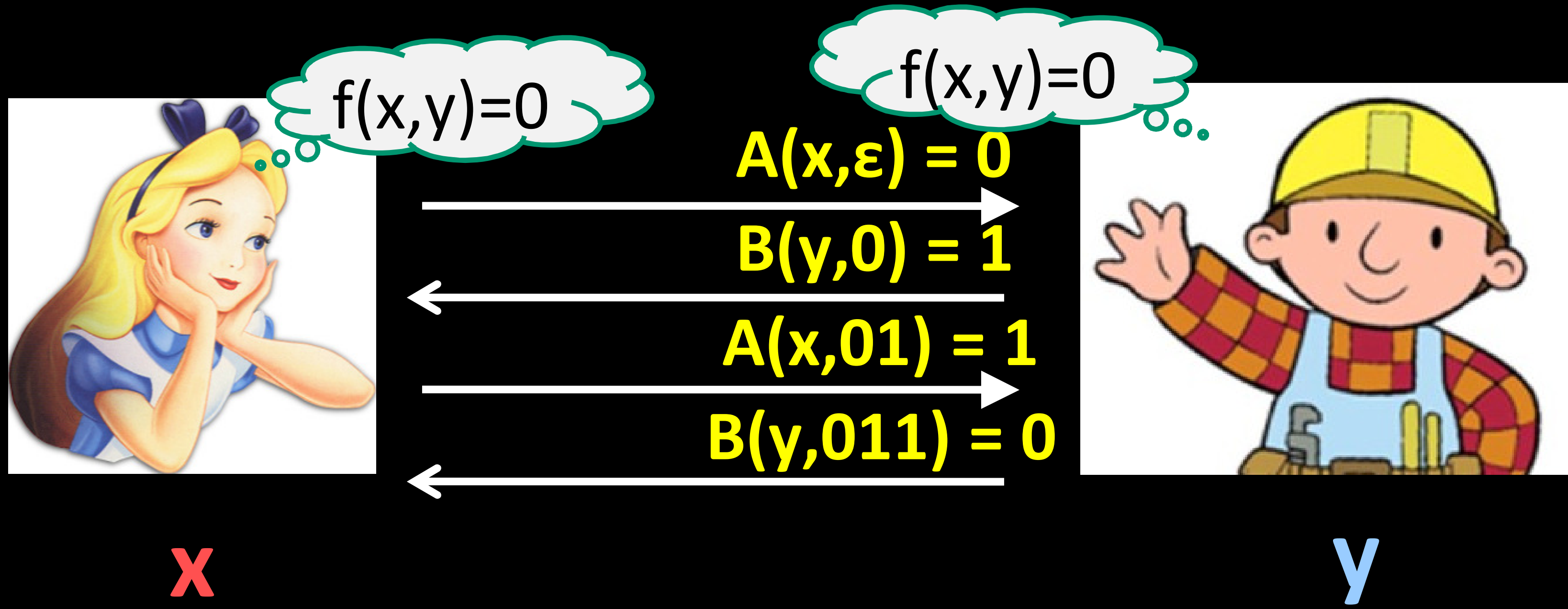



**Def.** The *cost of a protocol* P *for f on $n$-bit strings is*

$$\max_{x,y \in\{0,1\}^n} [\text{number of rounds in P to compute } f(x,y)]$$

The *communication complexity* of *f* on $n$-bit strings is the ***minimum* cost** over all protocols for *f* on $n$-bit strings

= the minimum number of rounds used in any protocol for computing *f*$(x,y)$ over all $n$-bit $x, y$




**Example.** Let $f$ : {0,1}* × {0,1}* → {0,1} be arbitrary

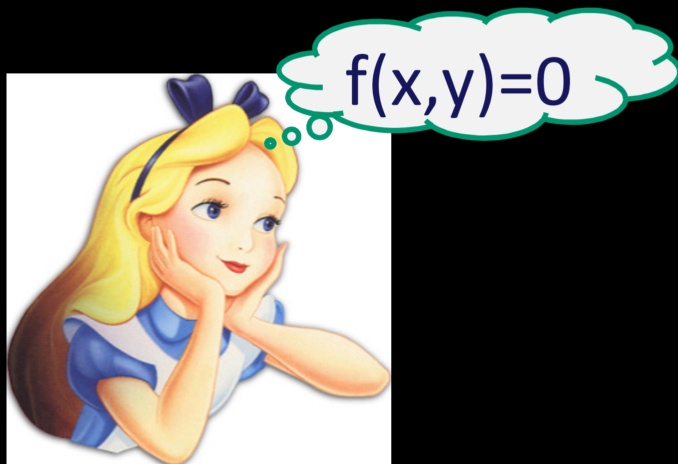
**There is always a "trivial" protocol:**

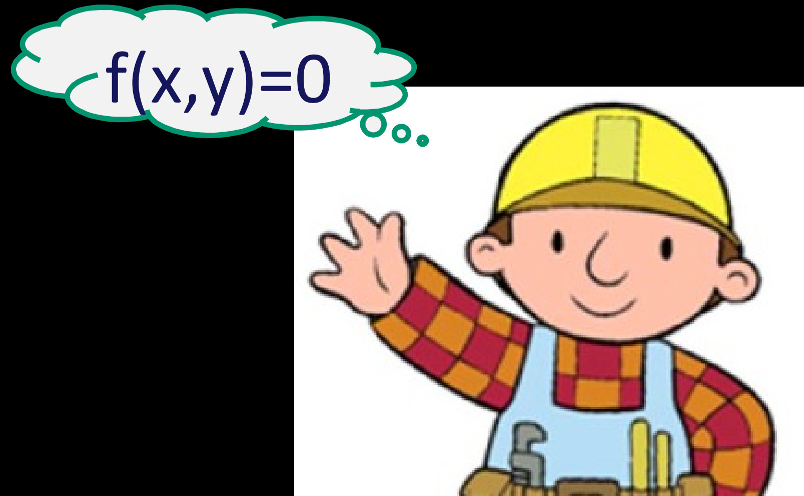
**Alice sends bits of $x$ in odd rounds**

**Bob sends bits of $y$ in even rounds**

**After $2n$ rounds, they both know each other's input!**

***The communication complexity of every $f$ is at most $2n$***




**x** **y**

**Example. EQUALS($x, y$) = 1 $\Leftrightarrow$ $x = y$**

**What's a good protocol for computing EQUALS?**

**????**

***Communication complexity of EQUALS is at most $2n$***

# Connection to Streaming and DFAs

x

y

Let $L \subseteq \{0,1\}^*$

**Def.** $f_L$: $\{0,1\}^* \times \{0,1\}^* \to \{0,1\}$

for $x, y$ with $|x|=|y|$ as:

$$f_L(x, y) = 1 \Leftrightarrow xy \in L$$

**Examples:**

$L$ = { x | x has an odd number of 1s}

$\Rightarrow f_L(x, y)$ = PARITY(x,y) = $\sum_i x_i$ + $\sum_i y_i$ mod 2

$L$ = { x | x has more 1s than 0s}

$\Rightarrow f_L(x, y)$ = MAJORITY(x,y)

$L$ = { xx | x $\in$ {0,1}*}

$\Rightarrow f_L(x, y)$ = EQUALS(x,y)

# Connection to Streaming and DFAs

x

y

Let $L \subseteq \{0,1\}^*$

**Def.** $f_L : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}$
for $x, y$ with $|x|=|y|$ as:

$$f_L(x, y) = 1 \Leftrightarrow xy \in L$$

**Theorem:** If $L$ has a streaming algorithm using $\leq s$ space,
then the comm. complexity of $f_L$ is at most $4s + 5$.

**Proof:** Alice runs streaming algorithm A on $x$.
Sends the *memory content* of A: this is $s$ bits of space
Bob starts up A with that memory content, runs A on $y$.
Gets an output bit, sends to Alice.

*(…why 4s+5 rounds? Can you do better?)*

# Connection to Streaming and DFAs

Let $L \subseteq \{0,1\}^*$  **Def.** $f_L(x, y) = 1 \Leftrightarrow xy \in L$

**Theorem:** If $L$ has a streaming algorithm using $\leq s$ space, then the comm. complexity of $f_L$ is at most $4s + 5$.

**Corollary:** For every regular language $L$, the comm. complexity of $f_L$ is O(1).

**Example:** Comm. Complexity of PARITY is O(1)

**Corollary:** Comm. Complexity of MAJORITY is O(log n)

**What about the Comm. Complexity of EQUALS?**

# Communication complexity of EQUALS

**Theorem:** The comm. complexity of EQUALS is $\Theta(n)$.
In particular, *every* protocol for EQUALS needs $\geq n$ bits of communication.

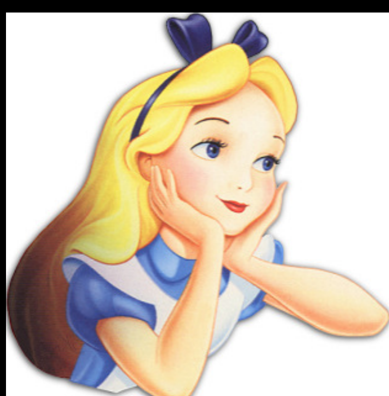
*No communication protocol can do much better than "send your entire input"!*

**Corollary:** $L$ = {ww | w in {0,1}*} is not regular.
*Moreover, every streaming algorithm for $L$ needs $c\ n$ bits of memory, for some constant $c$ > 0!*

# Communication complexity of EQUALS

**Theorem:** The comm. complexity of EQUALS is $\Theta(n)$.
In particular, *every* protocol for EQUALS needs $\geq n$ bits of communication.

**Idea:** Consider all possible ways they can communicate.
**Definition:** The ***communication pattern*** of a protocol on $(x, y)$ is the sequence of bits that Alice & Bob send.



$x$

0

1

1

0

Pattern: 0110

$y$

# Communication complexity of EQUALS

**Theorem:** The comm. complexity of EQUALS is $\Theta(n)$.
In particular, *every* protocol for EQUALS needs $\geq n$ bits of communication.

**Proof:** By contradiction. Suppose CC of EQUALS is $\leq n - 1$.
Then there are $\leq 2^n - 1$ possible communication *patterns* of that protocol, over all pairs of inputs $(x, y)$.

**Claim:** There are $x \neq y$ such that on $(x, x)$ and on $(y, y)$, the protocol uses the *same* pattern $P$.

What is the communication pattern on $(x, y)$?

This pattern is also $P$! *(WHY?)*

So Alice & Bob *output the same bit* on $(x, y)$ and $(x, x)$.
But EQUALS$(x, y)$ = 0 and EQUALS$(x, x)$ = 1. *Contradiction!*

# Randomized Protocols Help!

**EQUALS needs $c\,n$ bits of communication, but…**

**Theorem:** **For all $(x, y)$ of $n$ bits each, there is a *randomized* protocol for EQUALS$(x, y)$ using only O(log $n$) bits of communication, which works with probability 99.9%!**

# Turing Machines

# Turing Machine

# Turing Machine (1936)

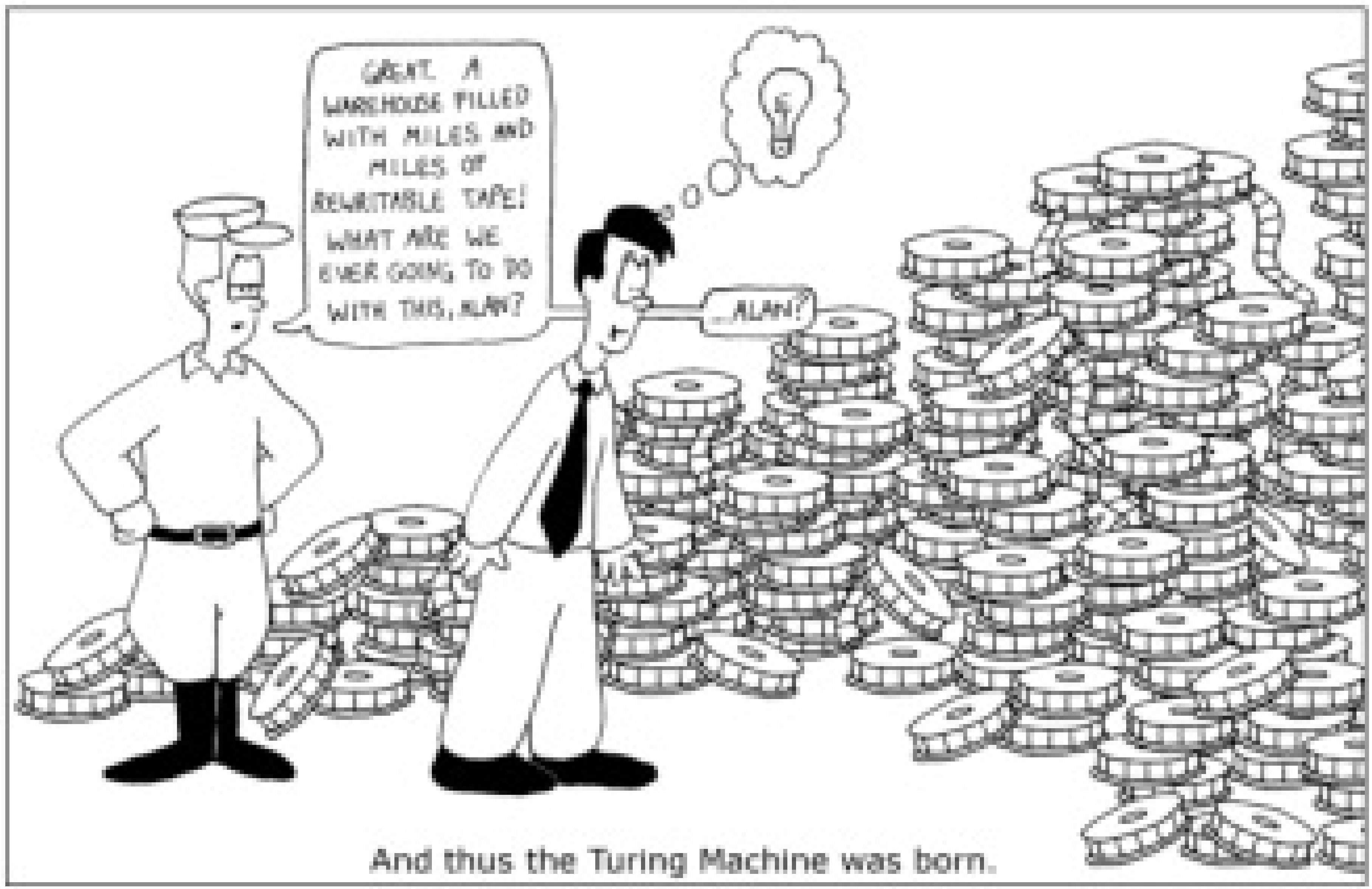


ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

*By* A. M. TURING.

[Received 28 May, 1936.—Read 12 November, 1936.]

The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable *numbers*, it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are, however, the same in each case, and I have chosen the computable numbers for explicit treatment as involving the least cumbrous technique. I hope shortly to give an account of the relations of the computable numbers,




And thus the Turing Machine was born.

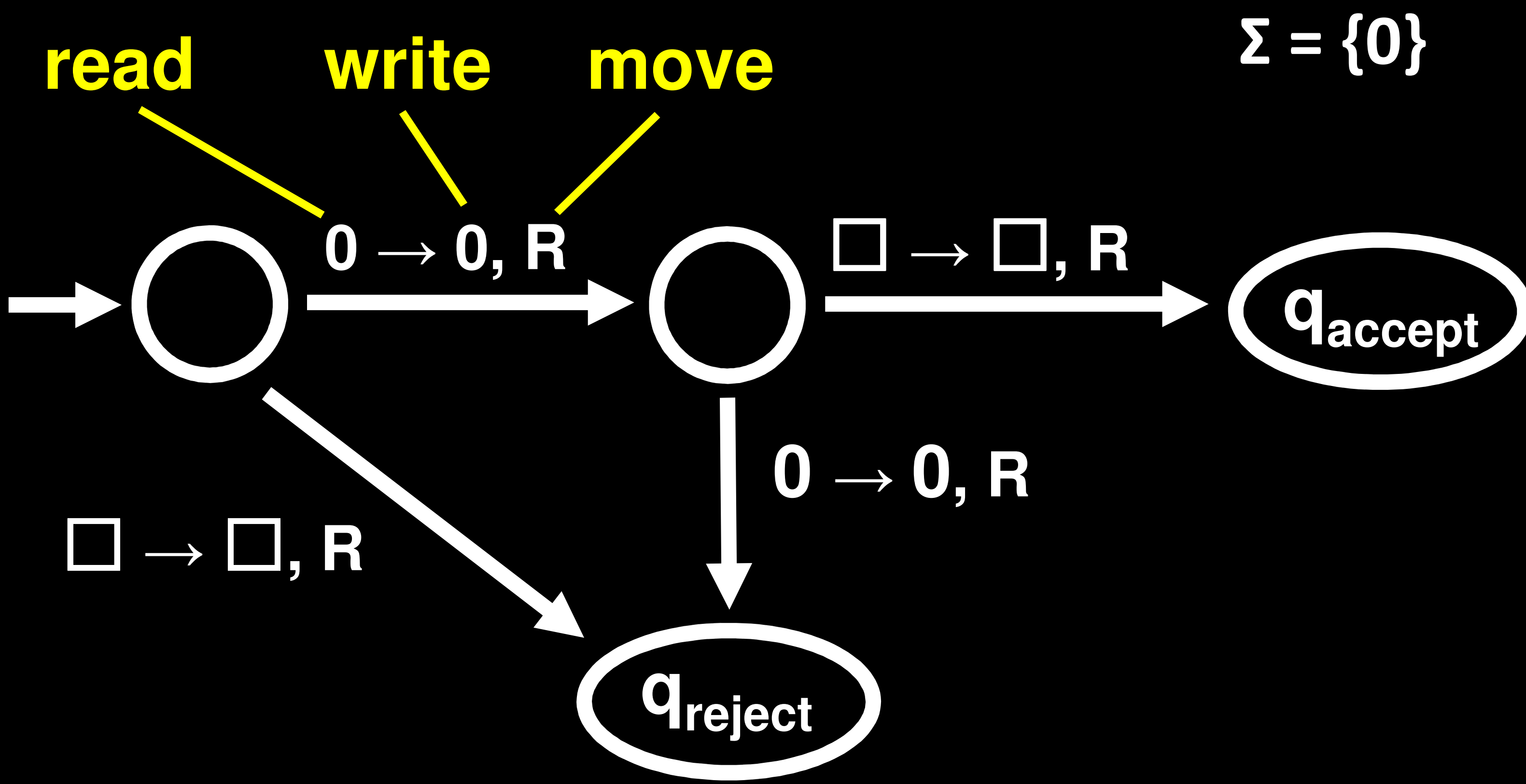
# Turing Machines versus DFAs

TM can both *write to* and *read from* the tape

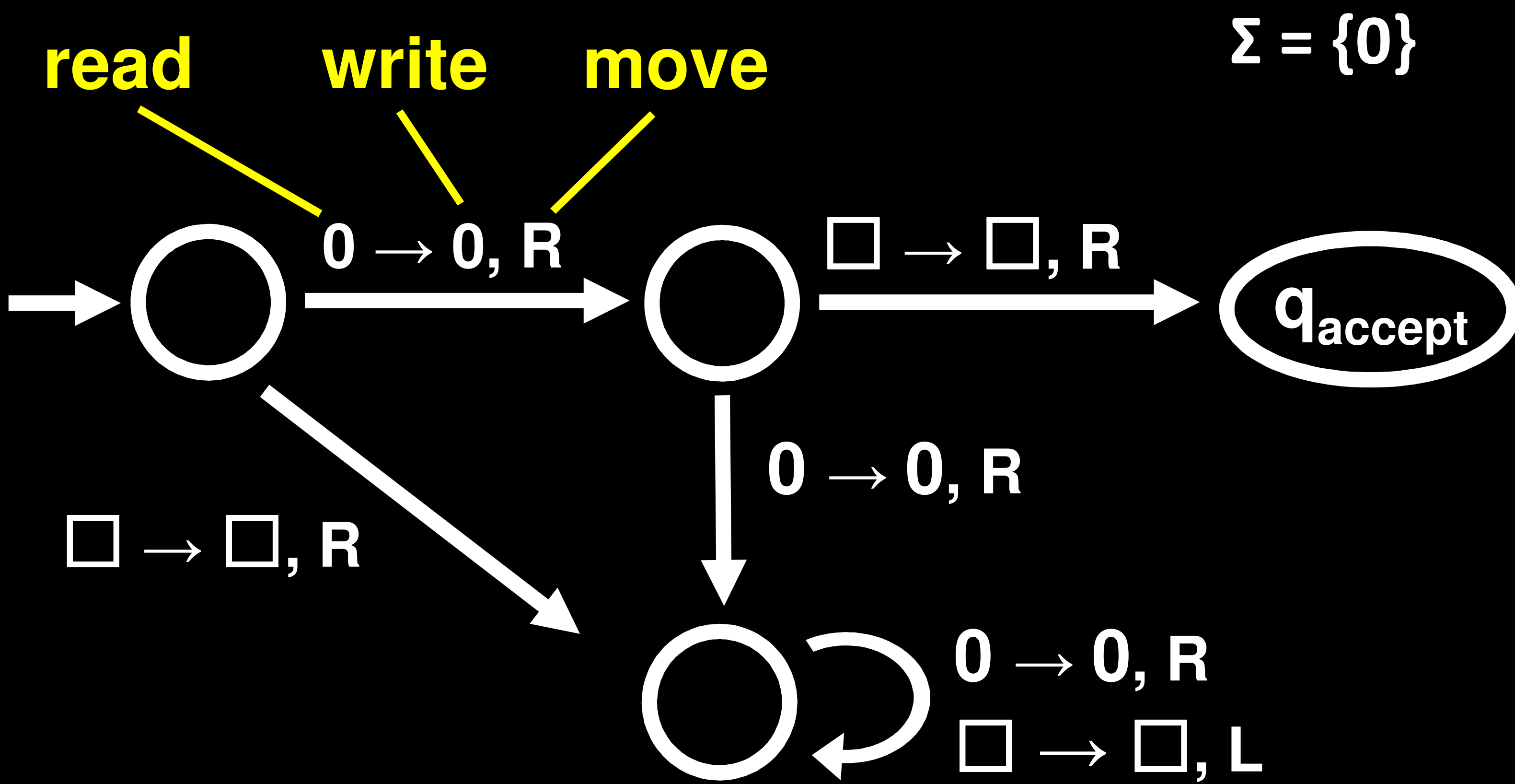
The head can move *left and right*

The input doesn't have to be read entirely,
and the computation can continue further
(even, *forever*) after all input has been read

**Accept** and **Reject** take immediate effect

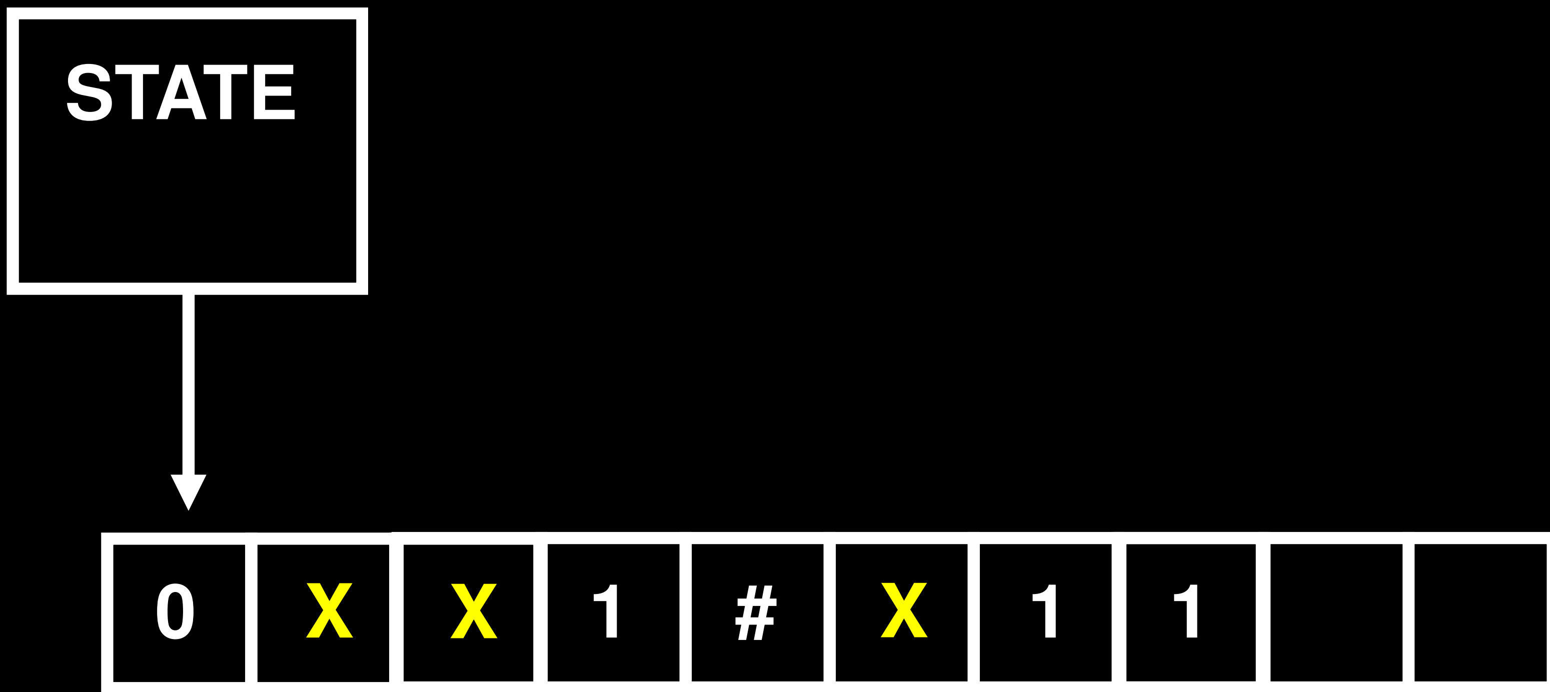



This Turing machine *decides* the language {0}




This Turing machine *recognizes* the language {0}

# Deciding the language **L = { w#w | w ∈ {0,1}* }**




1. If there's no # on the tape (or more than one #), *reject.*
2. While there is a bit to the left of #,

   Replace the first bit with X, and check if the first bit b to the right of the # is identical. (If not, *reject*.)

   Replace that bit b with an X too.
3. If there's a bit to the right of #, then *reject* else *accept*

**Definition:** A Turing Machine is a 7-tuple $T = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where:

$Q$ is a finite set of states

$\Sigma$ is the input alphabet, where $\Box \notin \Sigma$

$\Gamma$ is the tape alphabet, where $\Box \in \Gamma$ and $\Sigma \subseteq \Gamma$

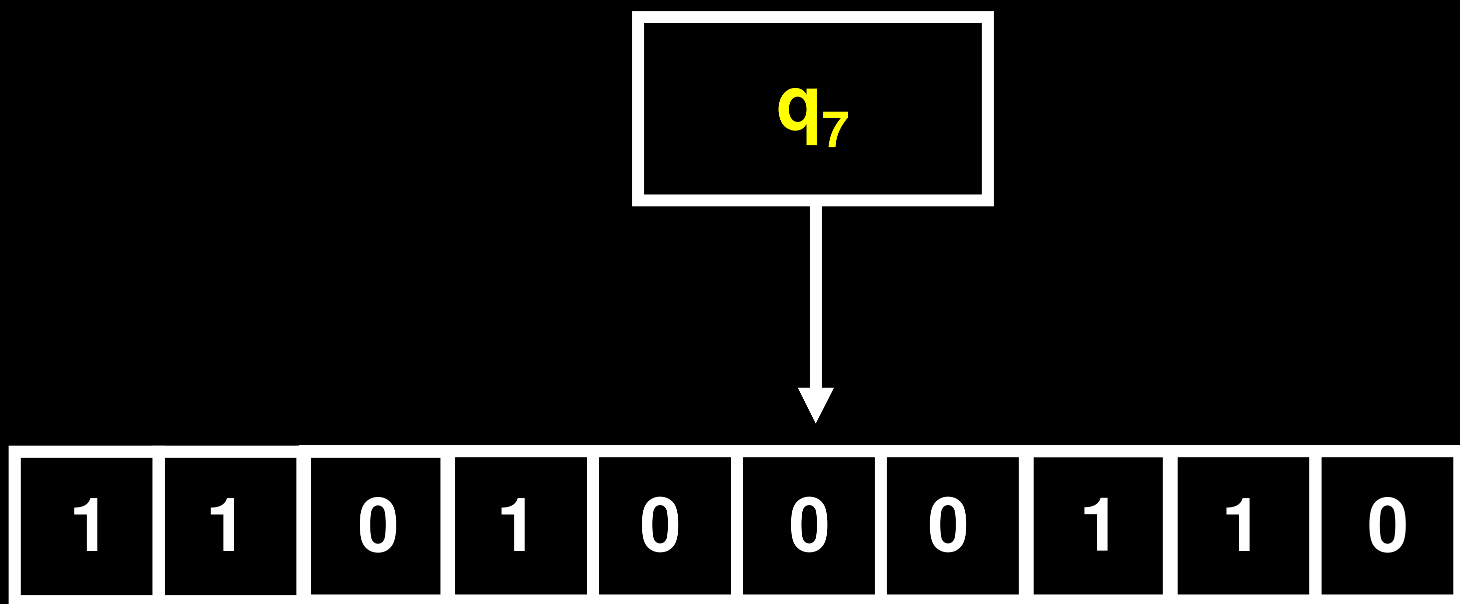
$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$

$q_0 \in Q$ is the start state

$q_{accept} \in Q$ is the accept state

$q_{reject} \in Q$ is the reject state, and $q_{reject} \neq q_{accept}$

# Turing Machine Configurations




corresponds to the *configuration*:

$$11010q_700110 \in \{Q \cup \Gamma\}^*$$

# Defining Acceptance and Rejection for TMs

Let $C_1$ and $C_2$ be configurations of a TM M

**Definition.** $C_1$ *yields* $C_2$ if M is in configuration $C_2$ after running M in configuration $C_1$ for one step

Suppose $\delta(q_1, b) = (q_2, c, L)$

Then $aaq_1bb$ yields $aq_2acb$

Suppose $\delta(q_1, a) = (q_2, c, R)$

Then $cabq_1a$ yields $cabcq_2\square$

Let $w \in \Sigma^*$ and M be a Turing machine

M *accepts* w if there are configs $C_0, C_1, \ldots, C_k$, s.t.

- $C_0 = q_0w$
- $C_i$ yields $C_{i+1}$ for $i = 0, \ldots, k-1$, and
- $C_k$ contains the accept state $q_{accept}$

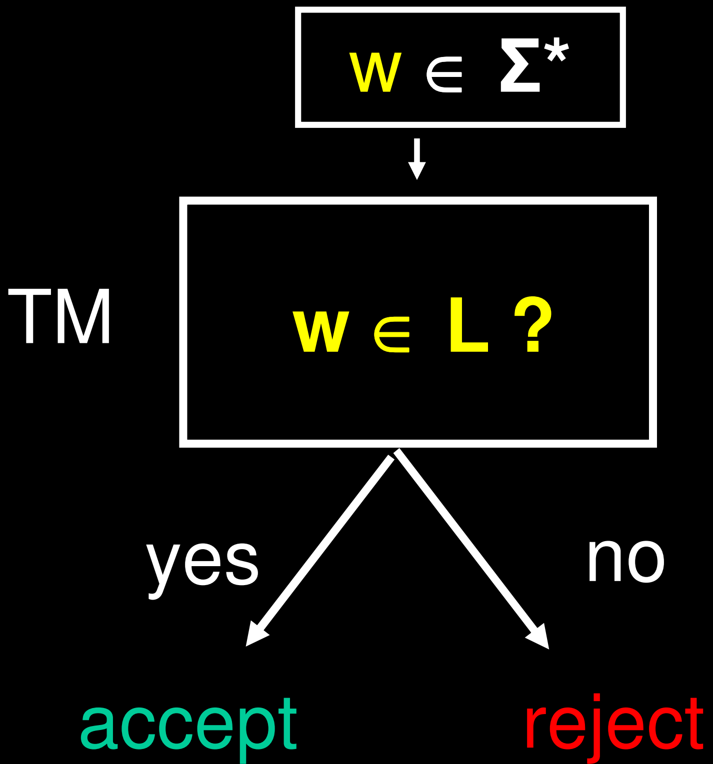
**A TM *M* *recognizes* a language L**
**if *M* accepts exactly those strings in L**

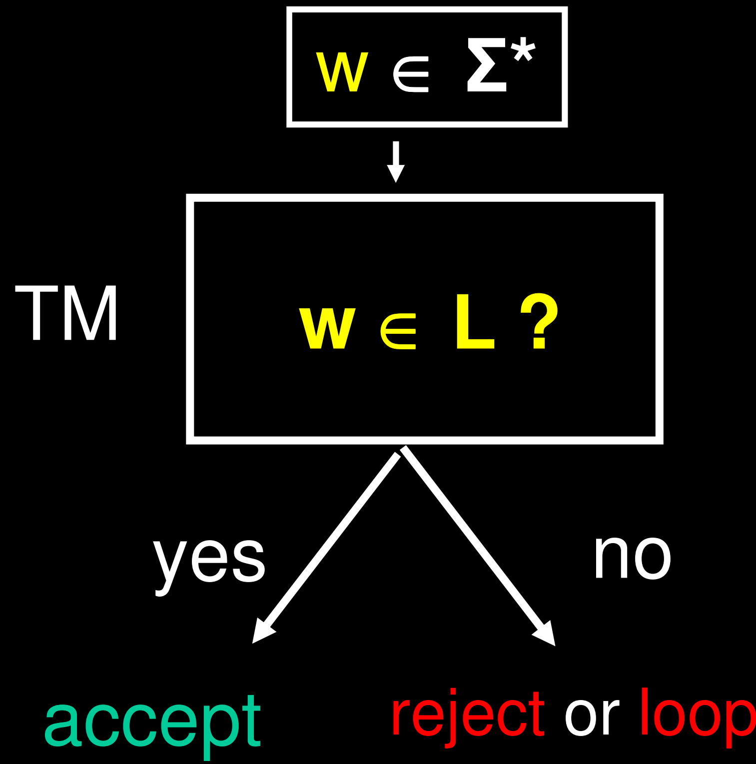
**A language L is *recognizable***
***(a.k.a. recursively enumerable)***
**if some TM recognizes L**

---

**A TM *M* *decides* a language L if *M* accepts all strings in L and rejects all strings not in L**

**A language L is *decidable (a.k.a. recursive)***
**if some TM decides L**




L is decidable
(recursive)

L is recognizable
(recursively enumerable)

# A Turing machine for deciding $\{ 0^{2^n} \mid n \geq 0 \}$

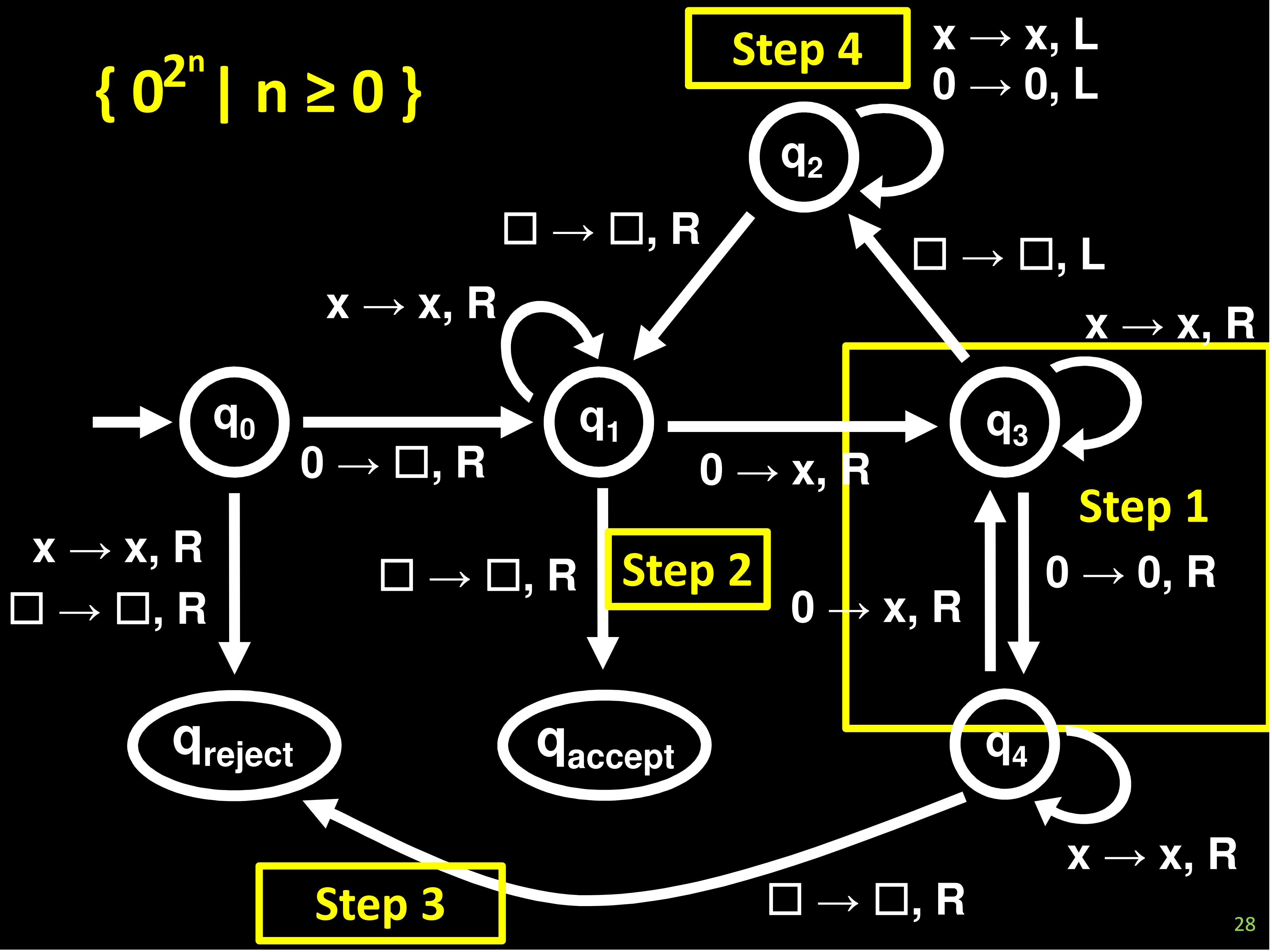
**Turing Machine PSEUDOCODE:**

1. Sweep from left to right, cross out every other **0**
2. If in step 1, the tape had only one **0**, *accept*
3. If in step 1, the tape had an odd number of **0**’s, *reject*
4. Move the head back to the first input symbol.
5. Go to step 1.

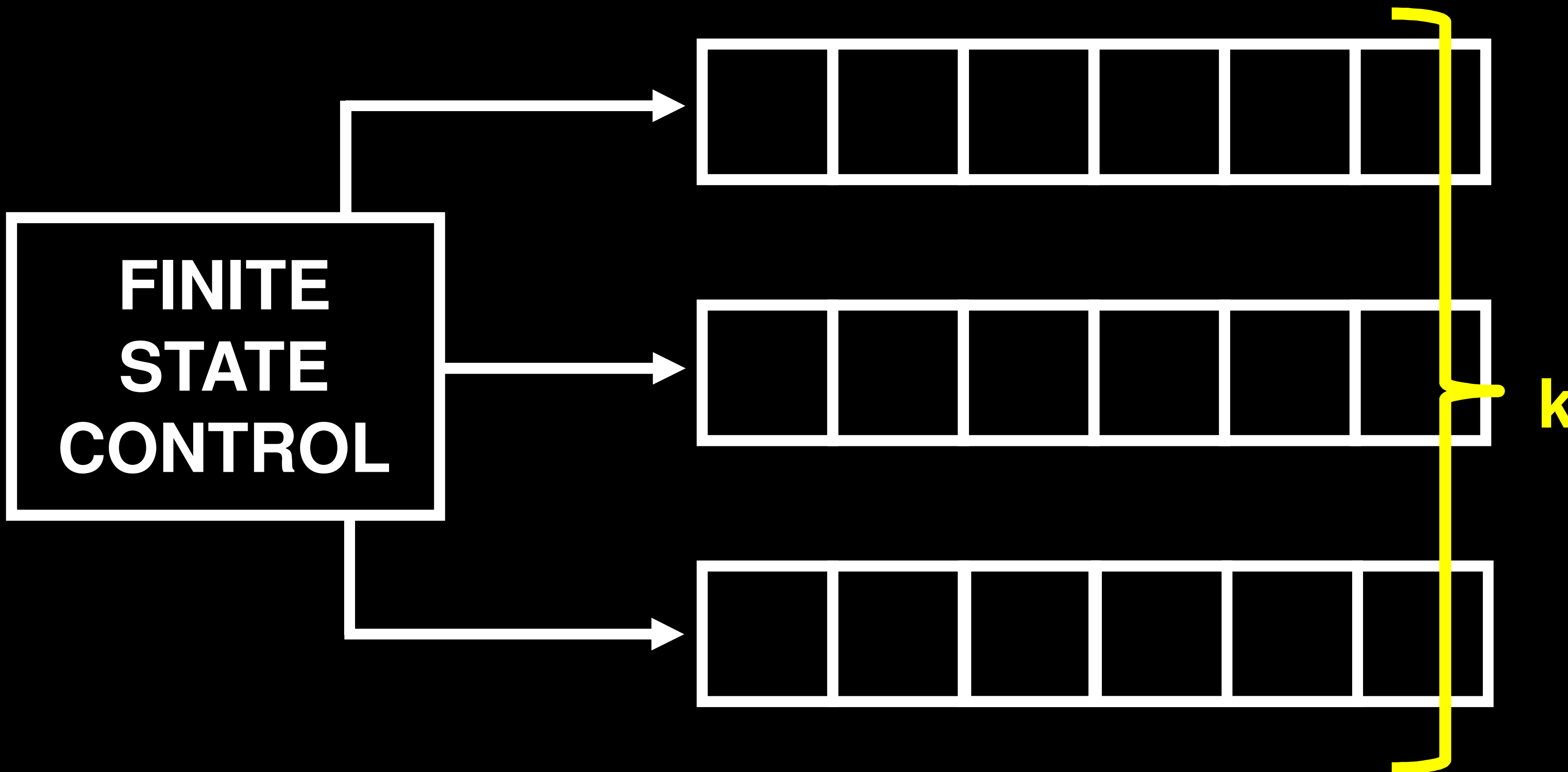
***Why does this work?***

**Idea:** **Every time we return to stage 1, the number of 0’s on the tape has been halved.**

$\{ 0^{2^n} \mid n \geq 0 \}$

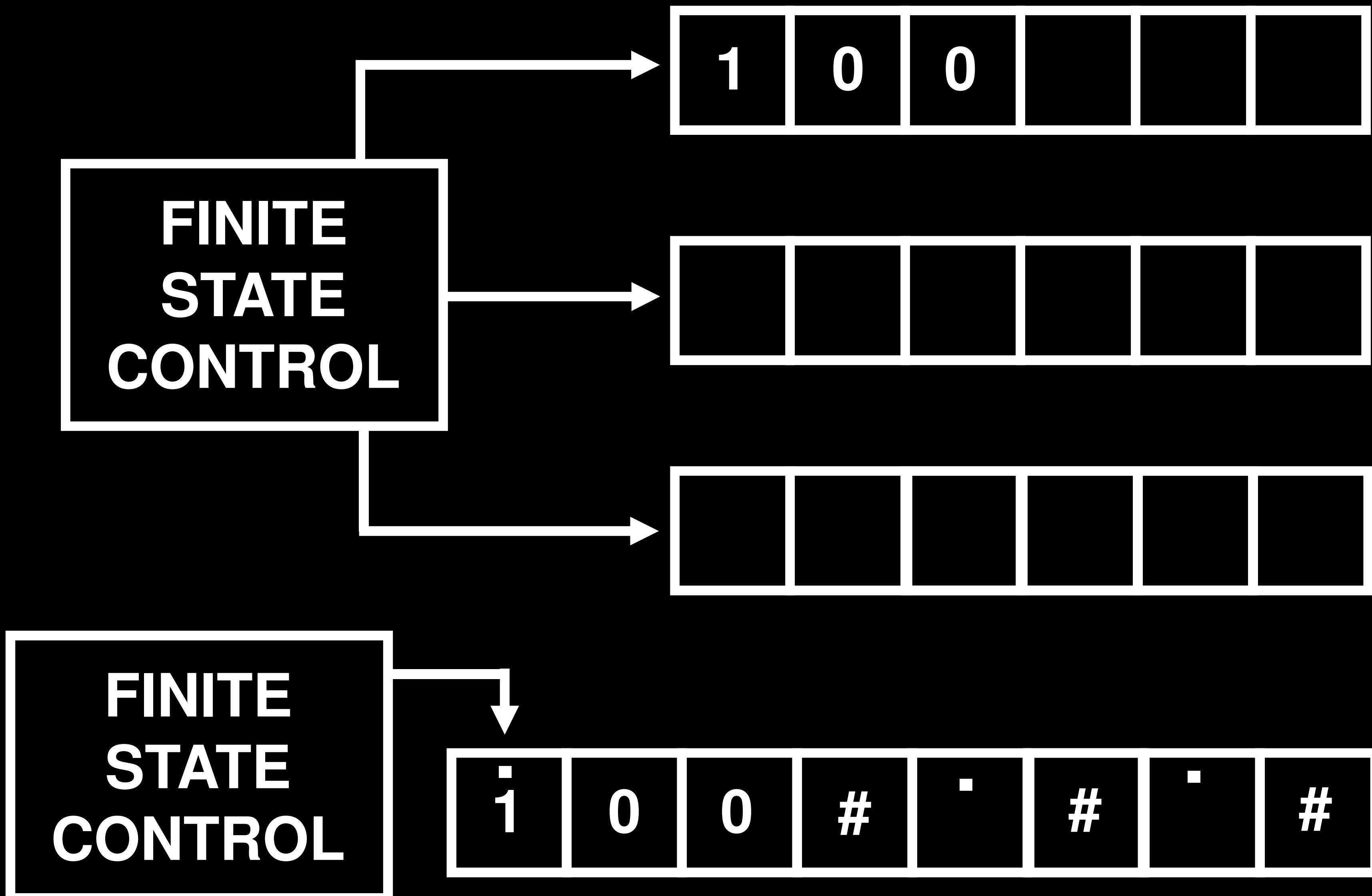


# Multitape Turing Machines

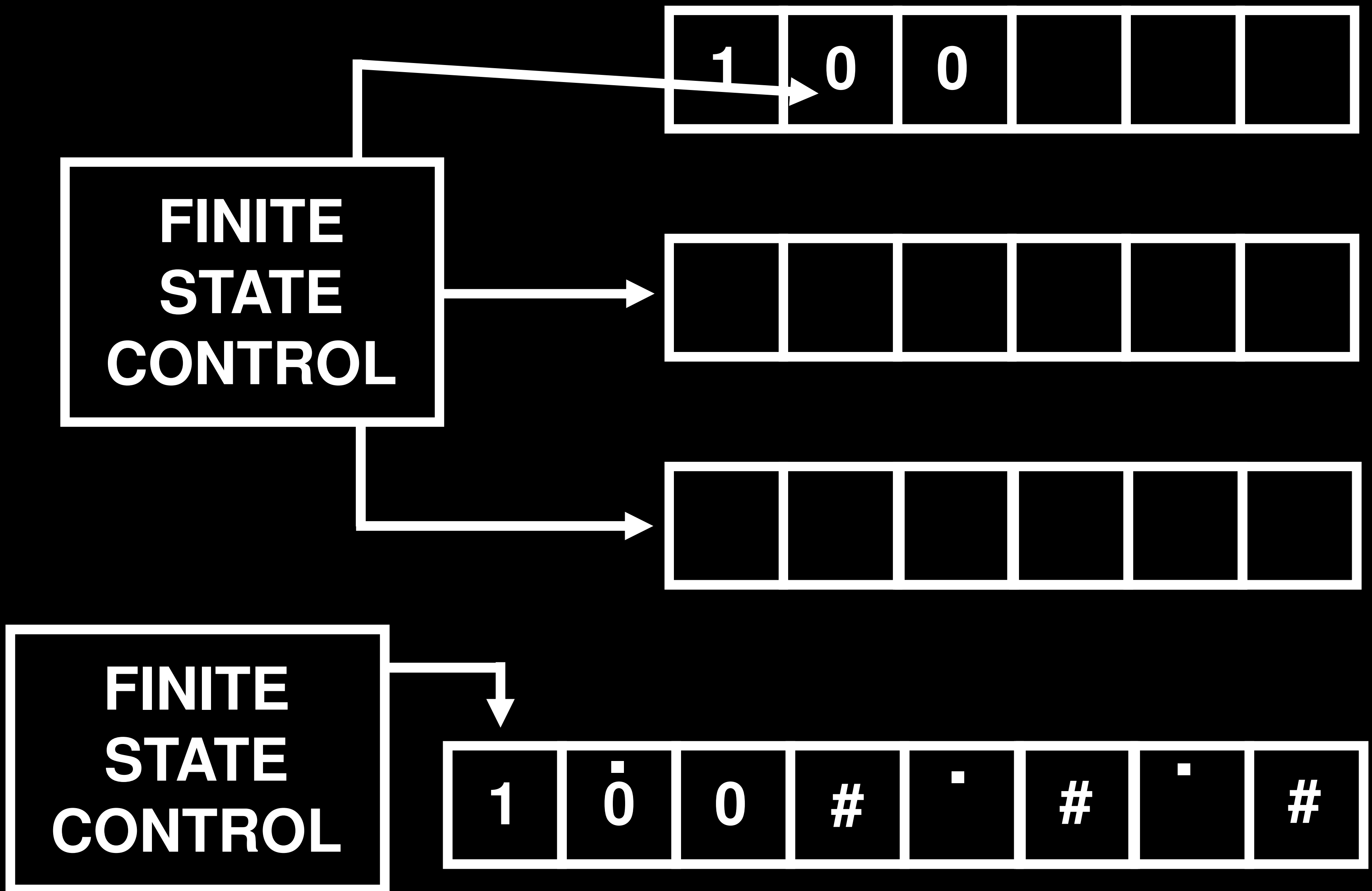



$$\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L,R\}^k$$

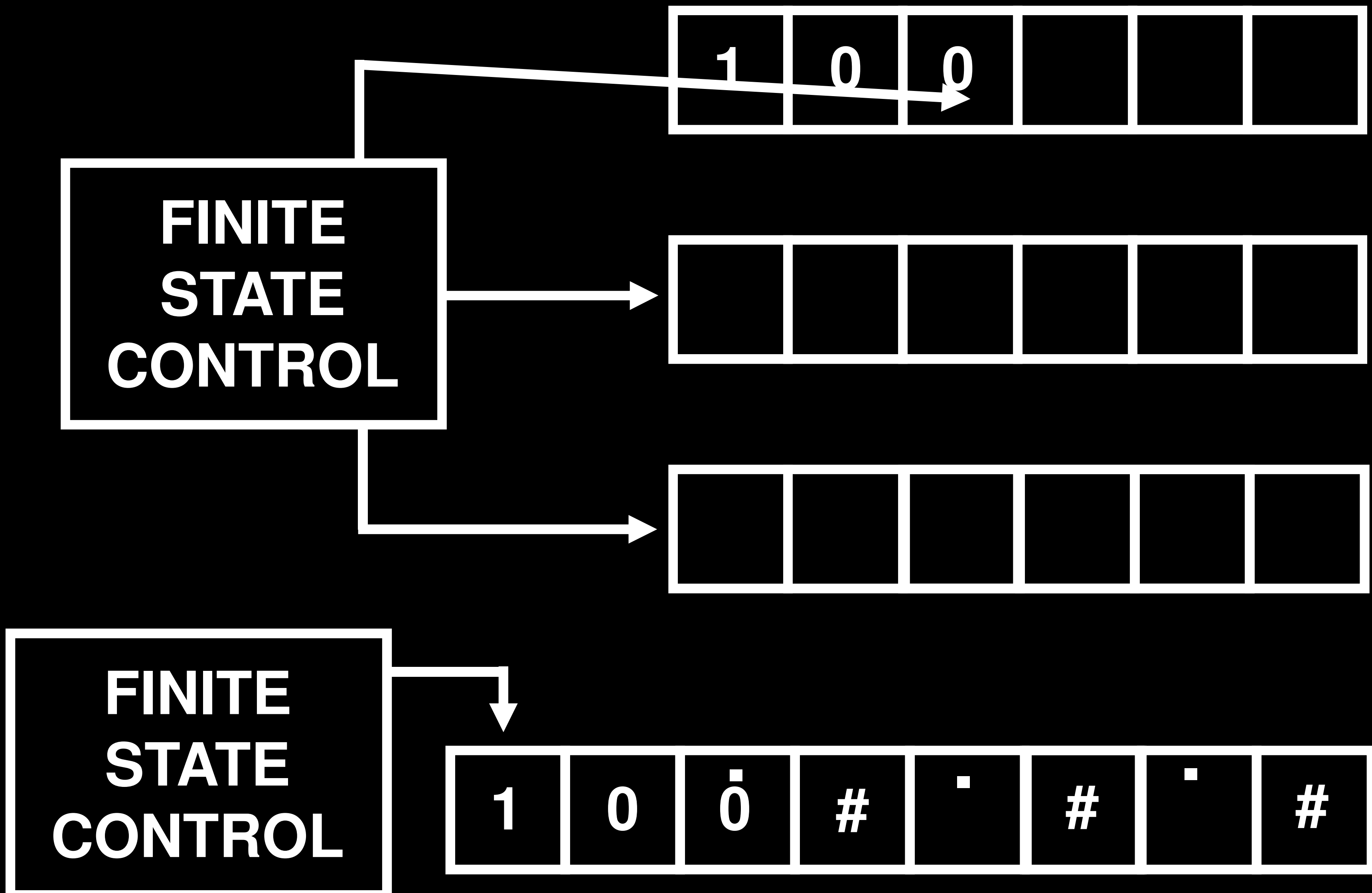
**Theorem:** Every Multitape Turing Machine can be transformed into a single tape Turing Machine



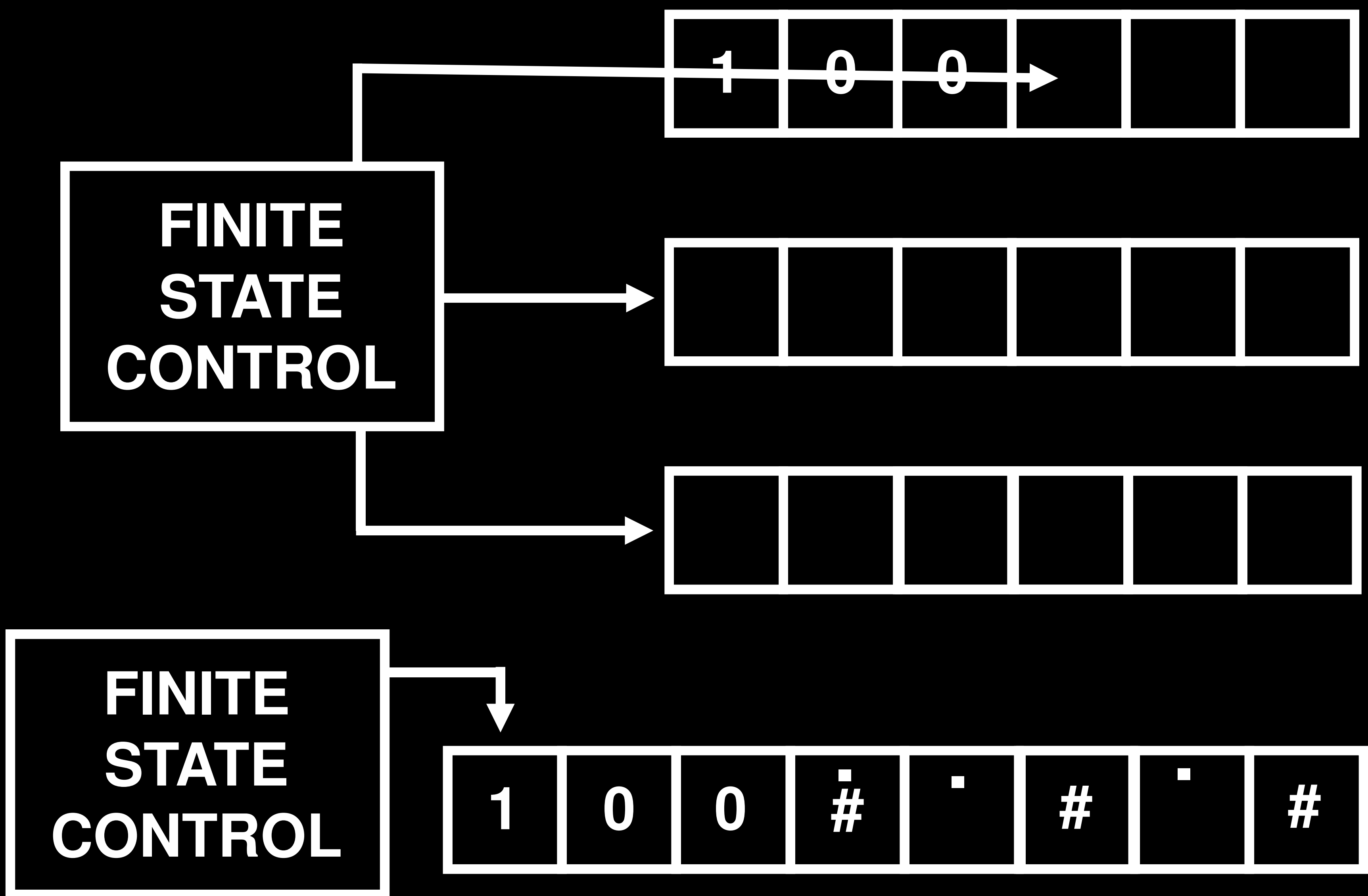
**Theorem:** Every Multitape Turing Machine can be transformed into a single tape Turing Machine



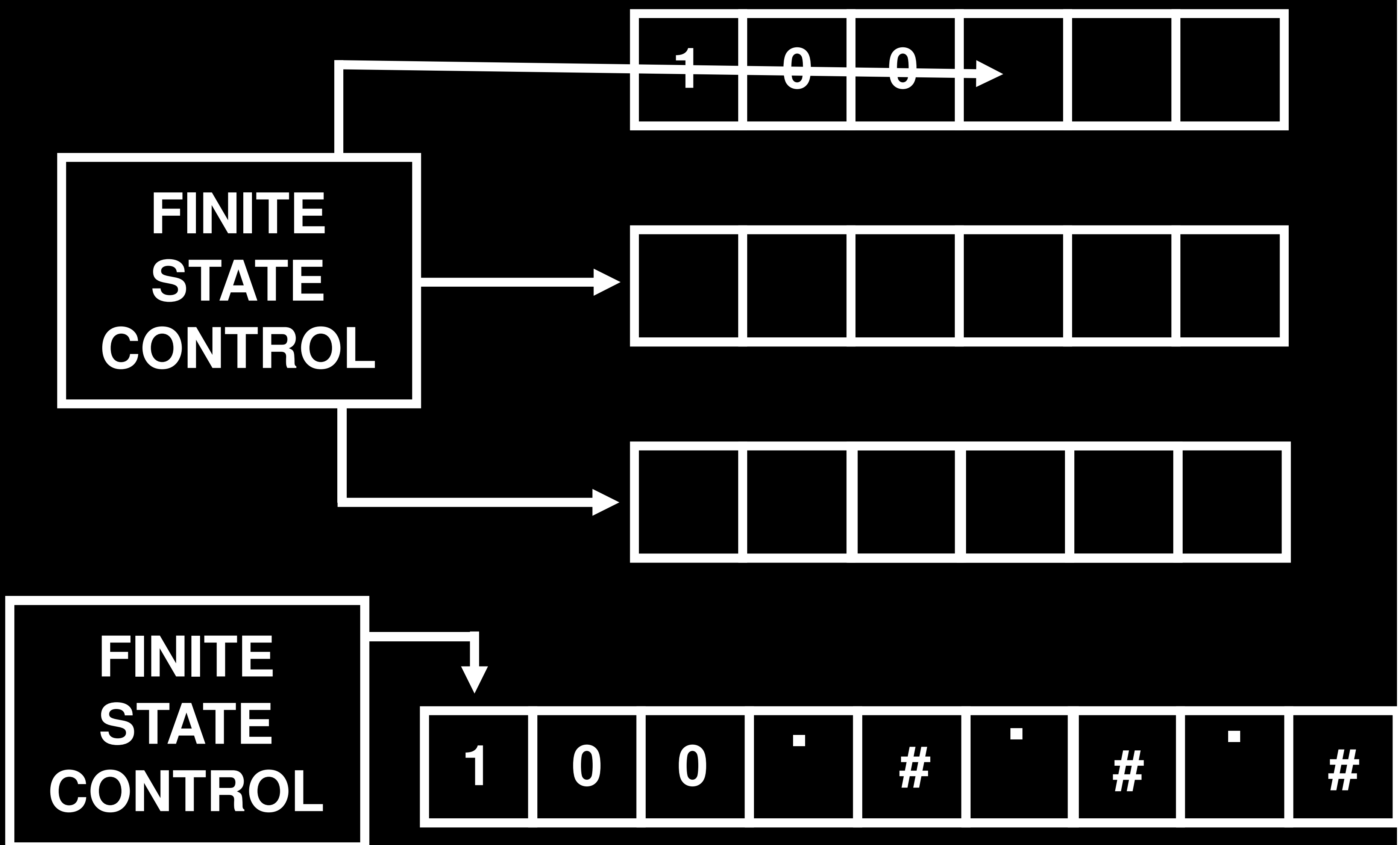
**Theorem:** Every Multitape Turing Machine can be transformed into a single tape Turing Machine

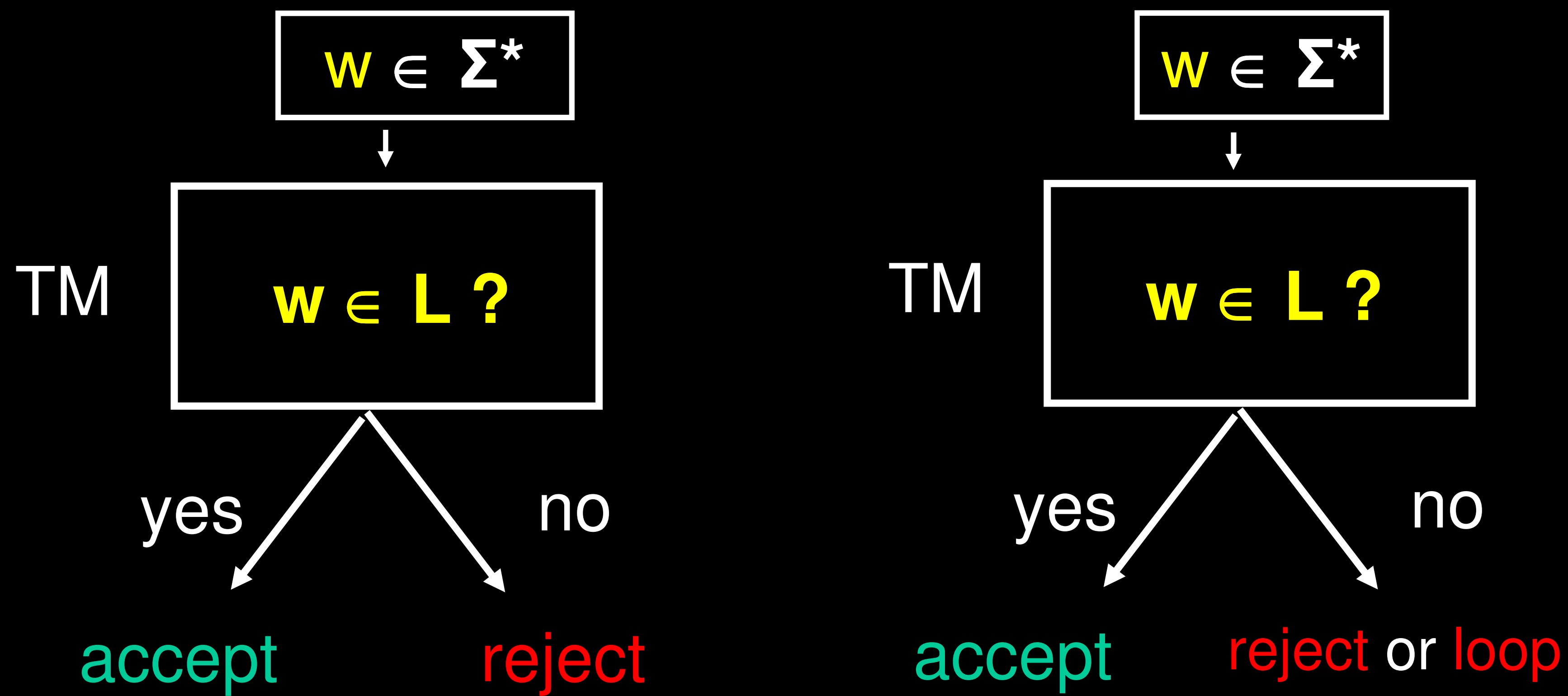


**Theorem:** Every Multitape Turing Machine can be transformed into a single tape Turing Machine



## Theorem: Every Multitape Turing Machine can be transformed into a single tape Turing Machine






L is decidable
(recursive)

L is recognizable
(recursively enumerable)

**Theorem: L is decidable iff both L and ¬L are recognizable**