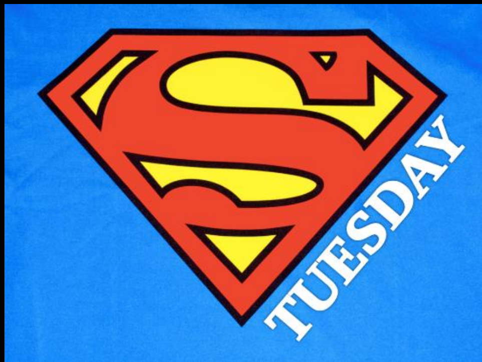


https://www.innovairre.com/super-tuesday/

6.045

Lecture 8:
Communication Complexity,
Start up Turing Machines

L has a streaming alg using $\leq s(n)$ bits of space

means:

Give an algorithm A and prove that on all inputs x, A determines $x \in$ L correctly and uses $\leq s(|x|)$ bits of memory

Give an upper bound!

Every streaming alg for L needs $\geq s(n)$ bits of space

means:

For any n, give a streaming distinguisher S for L (a set of strings such that all pairs can be distinguished in L) where $|S| \geq 2^{s(n)}$

Give a lower bound!

6.045

Announcements:

- Pest 3 is due tomorrow
- Midterm: March 19

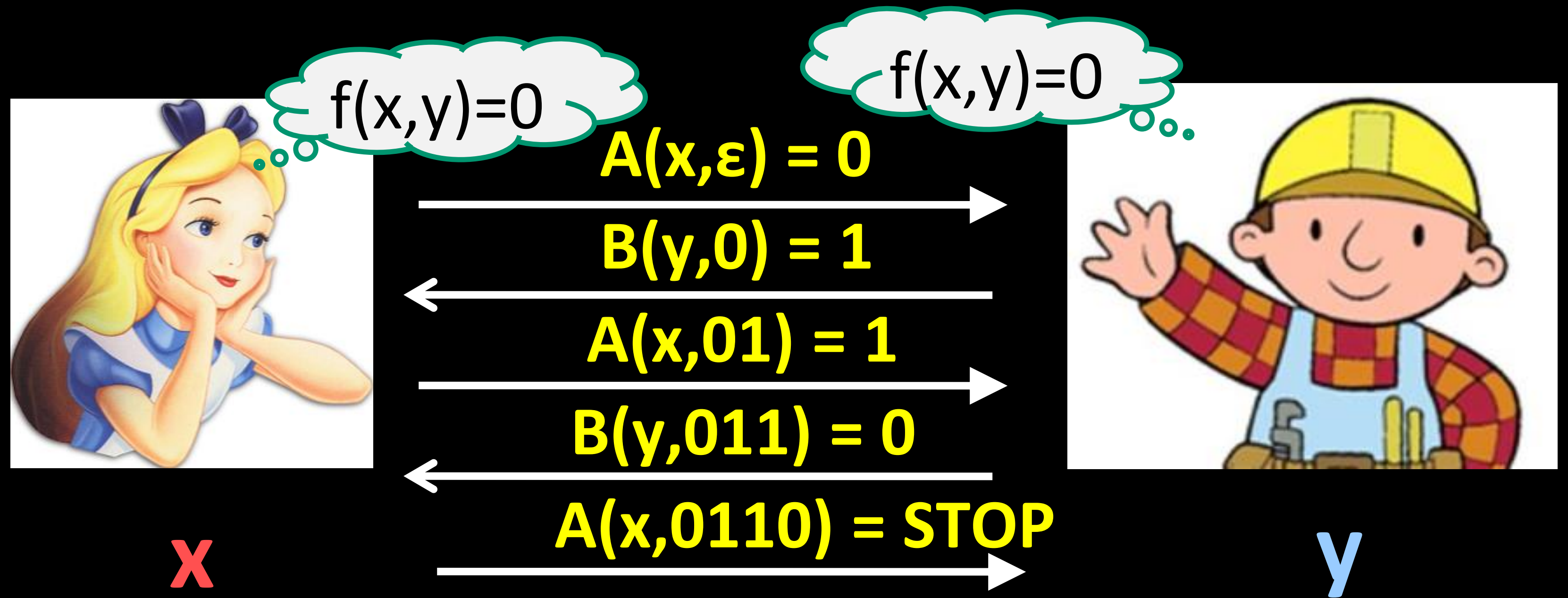
Communication Complexity

A theoretical model of distributed computing

- **Function $f: \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$**
  - 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.

Alice and Bob Have a Conversation

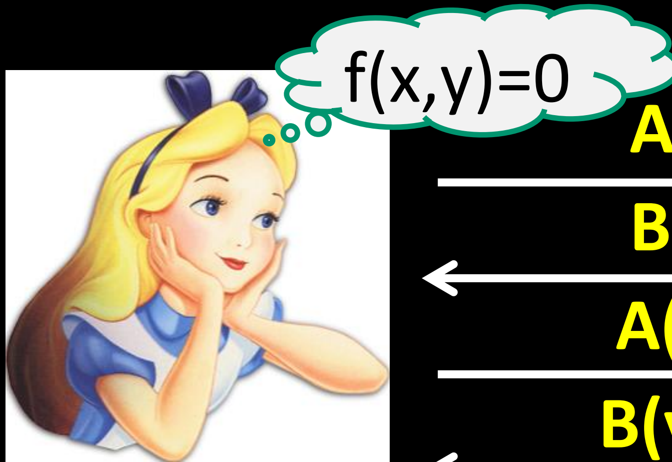


In every step: A **bit** or **STOP** is sent, which is a function of the party's input and all the bits communicated so far.

Communication cost = number of bits communicated

= 4 (in the example)

We assume Alice and Bob alternate in communicating, and the last BIT sent is the value of *f*(x,y)

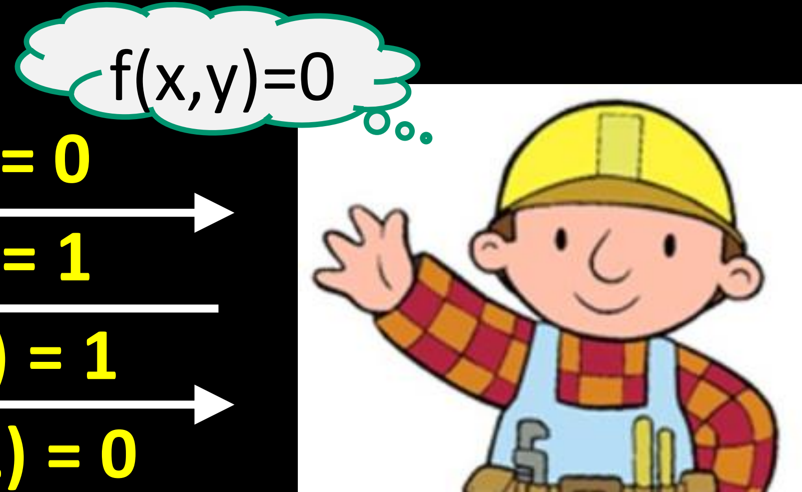


A(x,ε) = 0 →

← B(y,0) = 1

A(x,01) = 1 →

← B(y,011) = 0



x **y**

Def. A *protocol* computing f is a pair of functions

$A, B : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0, 1, \text{STOP}\}$ with the semantics:

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

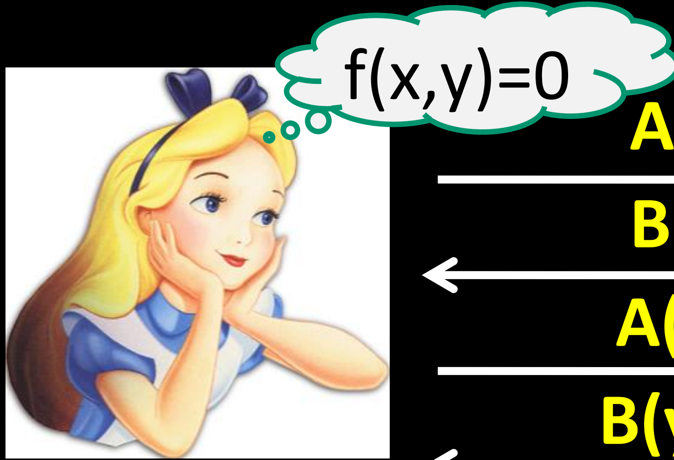
While ($b_r \neq$ STOP),

$r++$

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 $b_{r-1} = f(x, y)$. Number of *rounds* $= r - 1$

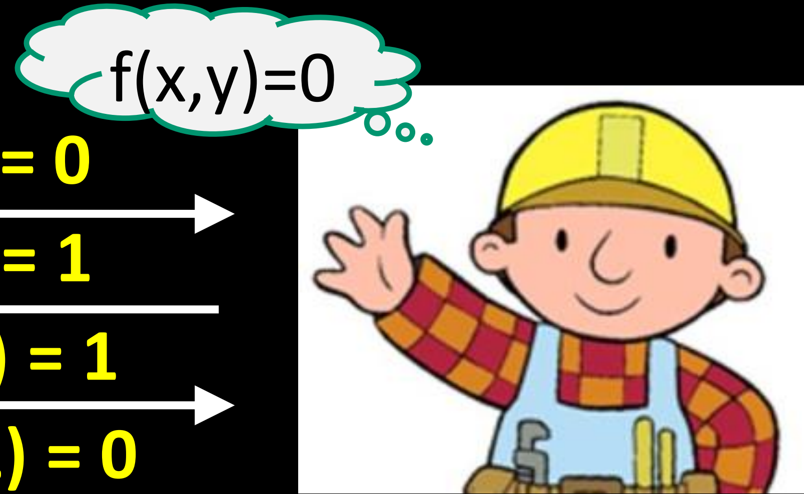


A(x,ε) = 0 →

← B(y,0) = 1

A(x,01) = 1 →

← B(y,011) = 0



x

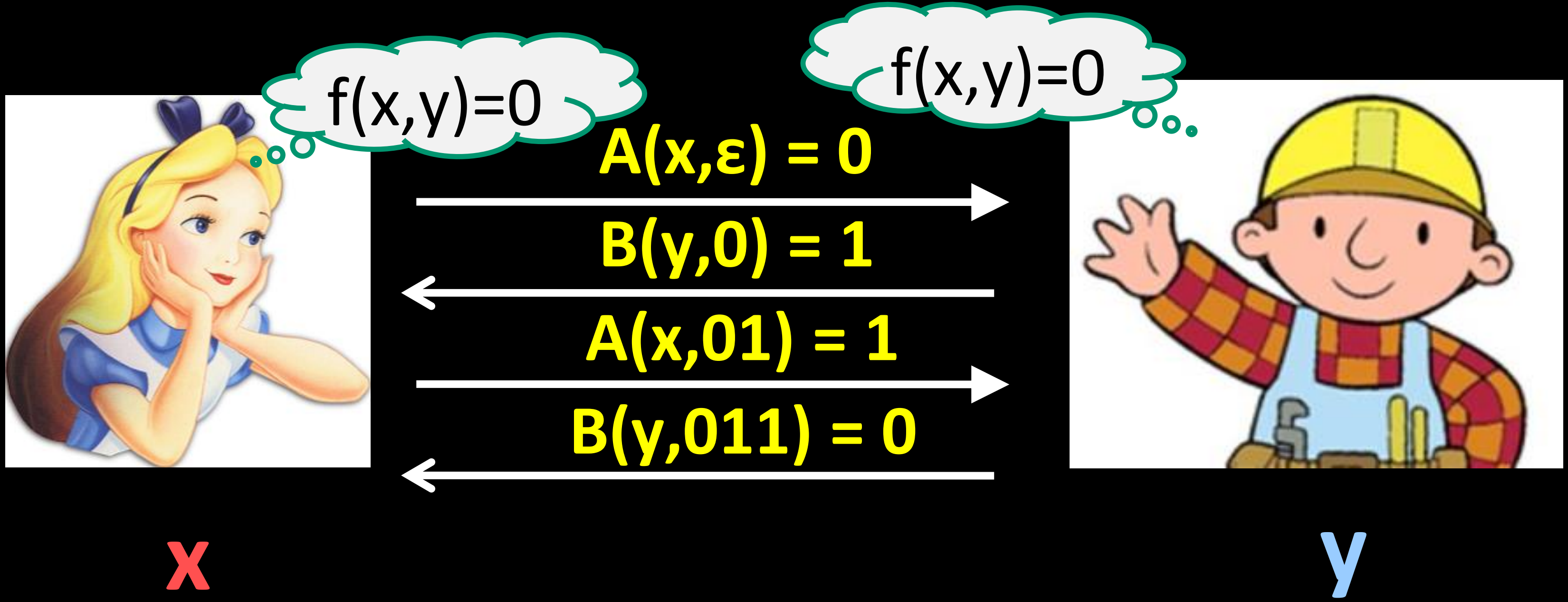
y

Def. The *cost of a protocol* (A,B) *on n-bit strings is*

$$\max_{x,y\,\in\{0,1\}^n}$$ [number of rounds taken by (A,B) on (x, y)]

The *communication complexity* of *f* on n-bit strings, *cc(f)*, is *min* cost over *all protocols* computing *f* on n-bit strings

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



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

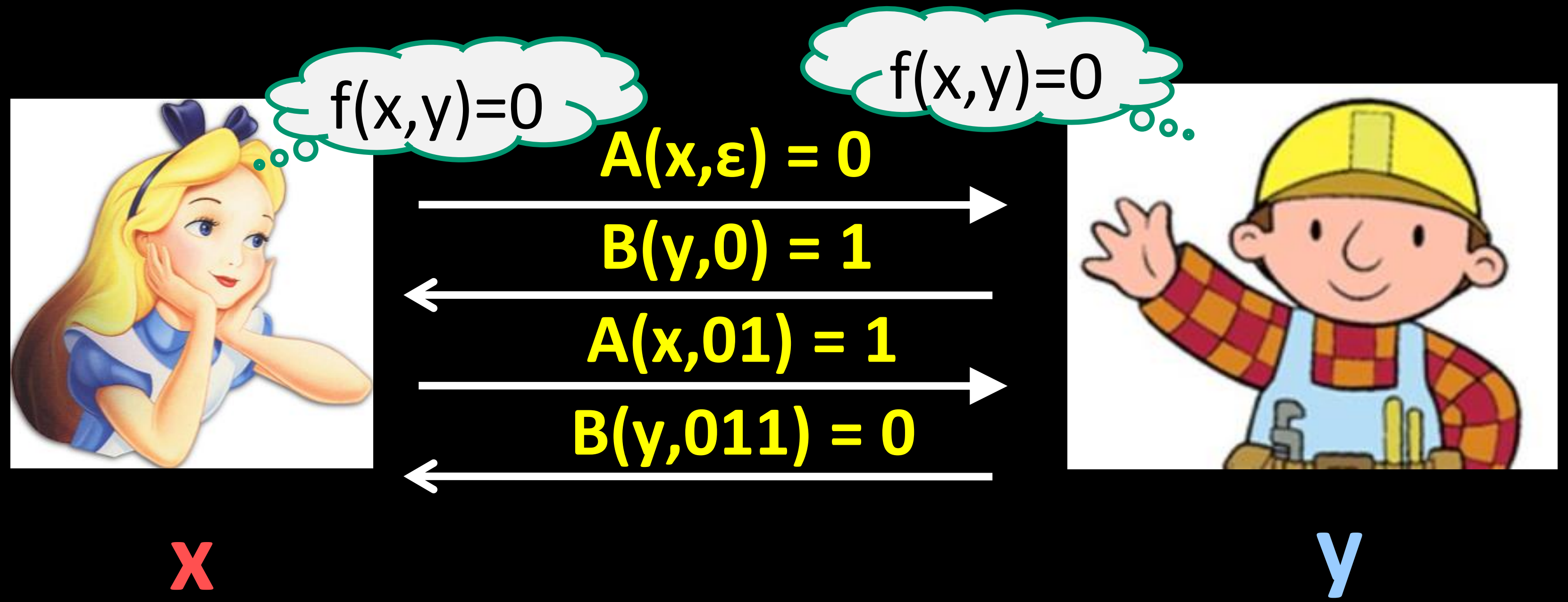
There is always a "trivial" protocol for f:

Alice sends the bits of her x in odd-numbered rounds

Bob sends whatever bit in even rounds

After $2n-1$ rounds, Bob knows x and can send $f(x,y)$

Proposition: For every f, cc(f) $\leq 2n$

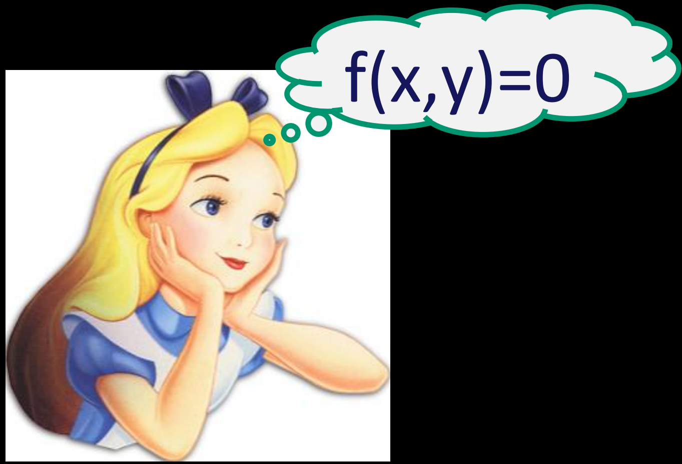


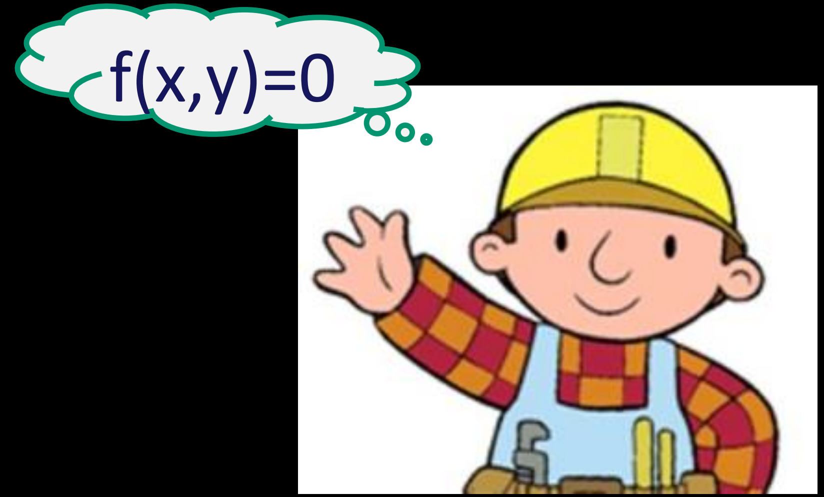
Example. PARITY(x, y) = $\sum_i x_i + \sum_i y_i$ mod 2.

What's a good protocol for computing PARITY?

Alice sends b_1 = ($\sum_i x_i$ mod 2)
Bob sends b_2 = ($b_1 + \sum_i y_i$ mod 2). Alice stops.

Proposition: cc(PARITY) = 2





x **y**

Example. MAJORITY(x, y) = most frequent bit in xy

Models voting in two "remote" locations; they want to determine a winner

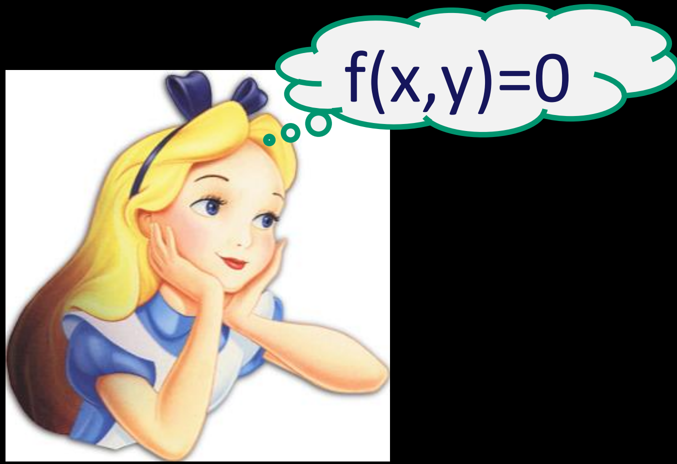
What's a good protocol for computing MAJORITY?

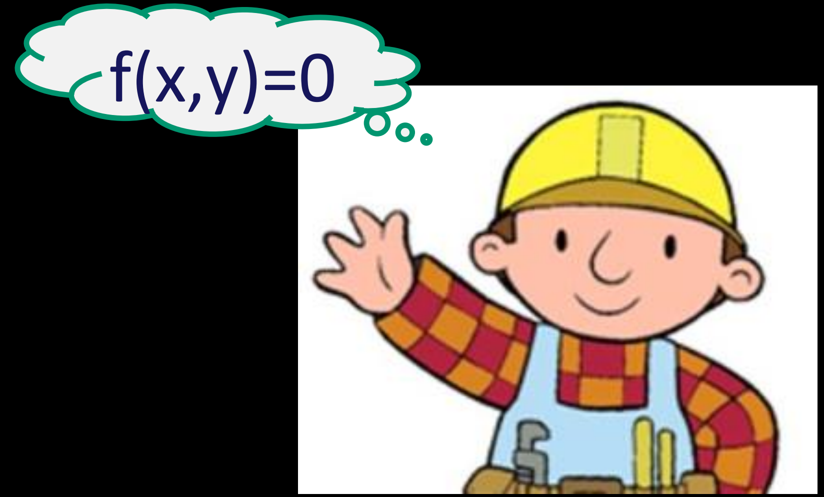
Alice sends N_x $=$ number of 1s in x

Bob computes N_y $=$ number of 1s in y,

sends 1 iff $N_x + N_y$ is greater than $(|x|+|y|)/2 = n$

Proposition: cc(MAJORITY) $\leq$ O(log n)





x **y**

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

Useful for checking consistency of two far-apart databases!

What's a good protocol for computing EQUALS?

????

Connection to Streaming Algs 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:

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

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 at least as many 1s as 0s}

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

L = { xx | x ∈ {0,1}*}

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

Connection to Streaming Algs 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 alg using $\leq s(m)$ space on inputs of length $\leq 2m$, then $\text{cc}(f_L) \leq O(s(n))$.

Proof Idea: Alice runs streaming algorithm A on x, reaches a memory state m. She sends m to Bob in $O(s(n))$ rounds. Then Bob starts up A from state m, runs A on y. Gets an output bit, sends bit to Alice.

Connection to Streaming Algs and DFAs

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

Theorem: If L has a streaming alg using $\leq s(m)$ space on inputs of length $\leq 2m$, then $\text{cc}(f_L) \leq O(s(n))$.

Corollary: For every regular L, $\text{cc}(f_L) \leq O(1)$.

Example: cc(PARITY) = 2

Corollary: cc(MAJORITY) $\leq$ O(log n),

because there's a streaming algorithm for $\{x : x$ has more 1's than 0's$\}$ with O(log n) space

What about the Comm. Complexity of EQUALS?

Communication Complexity of EQUALS

Theorem: cc(EQUALS) = $\Theta(n)$.
In particular, *every* communication protocol for EQUALS must send $\geq n$ bits between Alice and Bob.

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

Corollary: L = {xx | x in {0,1}*} is not regular.

***Corollary:** Every streaming algorithm for L needs $c\ n$ bits of memory, for some constant $c > 0$!*

$\Omega(n)$

Communication Complexity of EQUALS

Theorem: cc(EQUALS) = $\Theta(n)$. In particular, *every* protocol for EQUALS needs $\geq n$ bits of communication!

Idea: Consider all possible ways A & B can communicate.

Definition: The ***communication pattern*** of a protocol on inputs (x, y) is the sequence of bits Alice & Bob send.

x

0

1

1

0

Pattern: 0110

y

Key Lemma: If (x, y) and (x', y') have the same pattern P in a protocol, then (x, y') and (x', y) also have pattern P

x

A(x,ε) = 0 →

← B(y,0) = 1

A(x,01) = 1 →

← B(y,011) = 0

y

x'

A(x’,ε) = 0 →

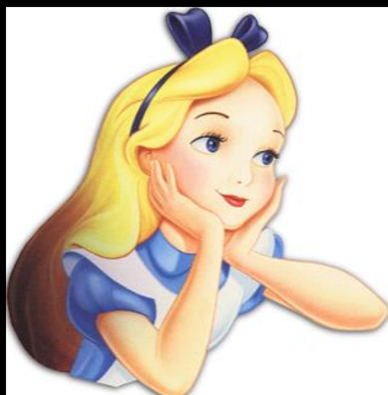
← B(y’,0) = 1

A(x’,01) = 1 →

← B(y’,011) = 0

y'

Key Lemma: If (x, y) and (x', y') have the same pattern P in a protocol, then (x, y') and (x', y) also have pattern P



A(x,ε) = 0 →

← B(y',0) = 1

A(x,01) = 1 →

← B(y',011) = 0

x y

A(x',ε) = 0 →

← B(y',0) = 1

A(x',01) = 1 →

← B(y',011) = 0

x' y'

Key Lemma: If (x, y) and (x', y') have the same pattern P in a protocol, then (x, y') and (x', y) also have pattern P

A(x,ε) = 0 →

B(y,0) = 1 ←

A(x,01) = 1 →

B(y,011) = 0 ←

x y

A(x,ε) = 0 →

B(y’,0) = 1 ←

A(x,01) = 1 →

B(y’,011) = 0 ←

x' 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(EQUALS) $\leq n - 1$.
Then there are $\leq 2^n - 1$ possible communication *patterns* of that protocol, over all pairs of inputs (x, y) with n bits each.

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

By the Key Lemma, (x, y) and (y, x) also use pattern P

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

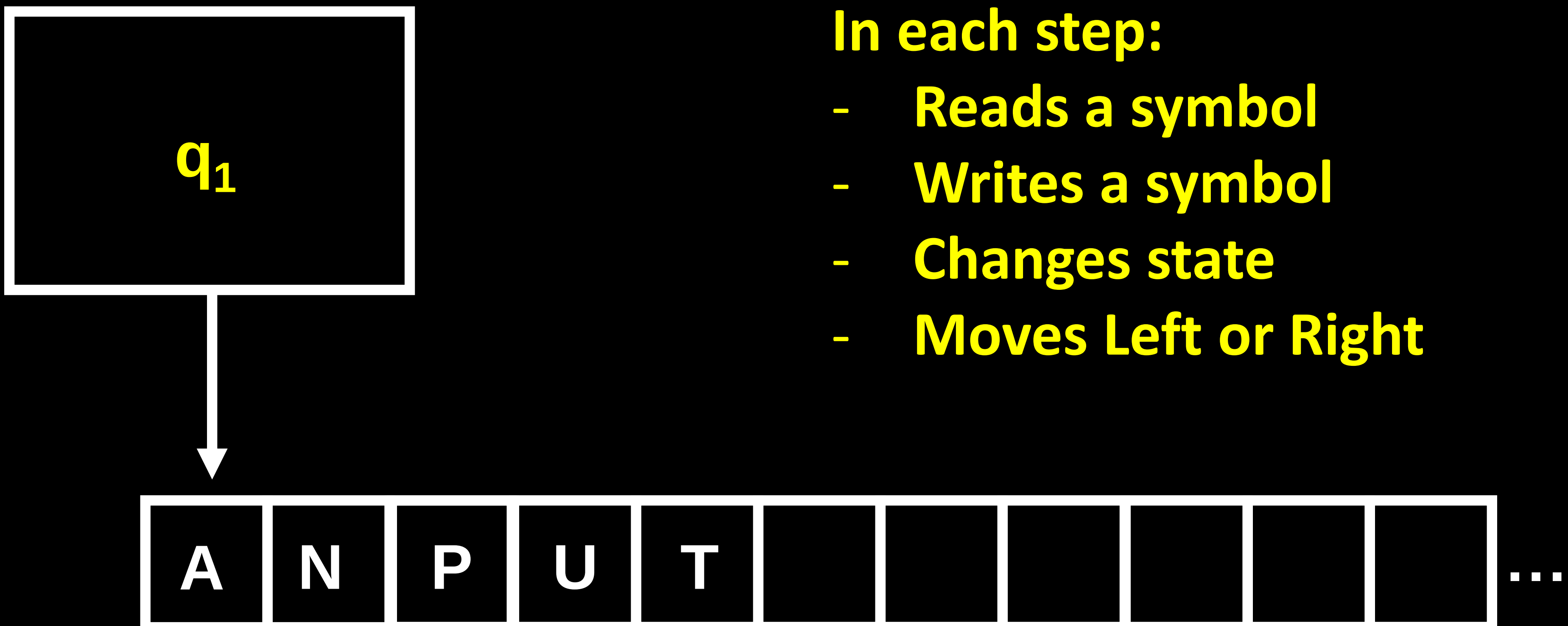
Randomized Protocols Help!

EQUALS needs $\geq n$ bits of communication, but…

Theorem: **There is a *randomized* protocol for computing EQUALS(x, y) using only O(log n) bits of communication, which is correct with probability 99.9%!**

Turing Machines

Turing Machine (1936)

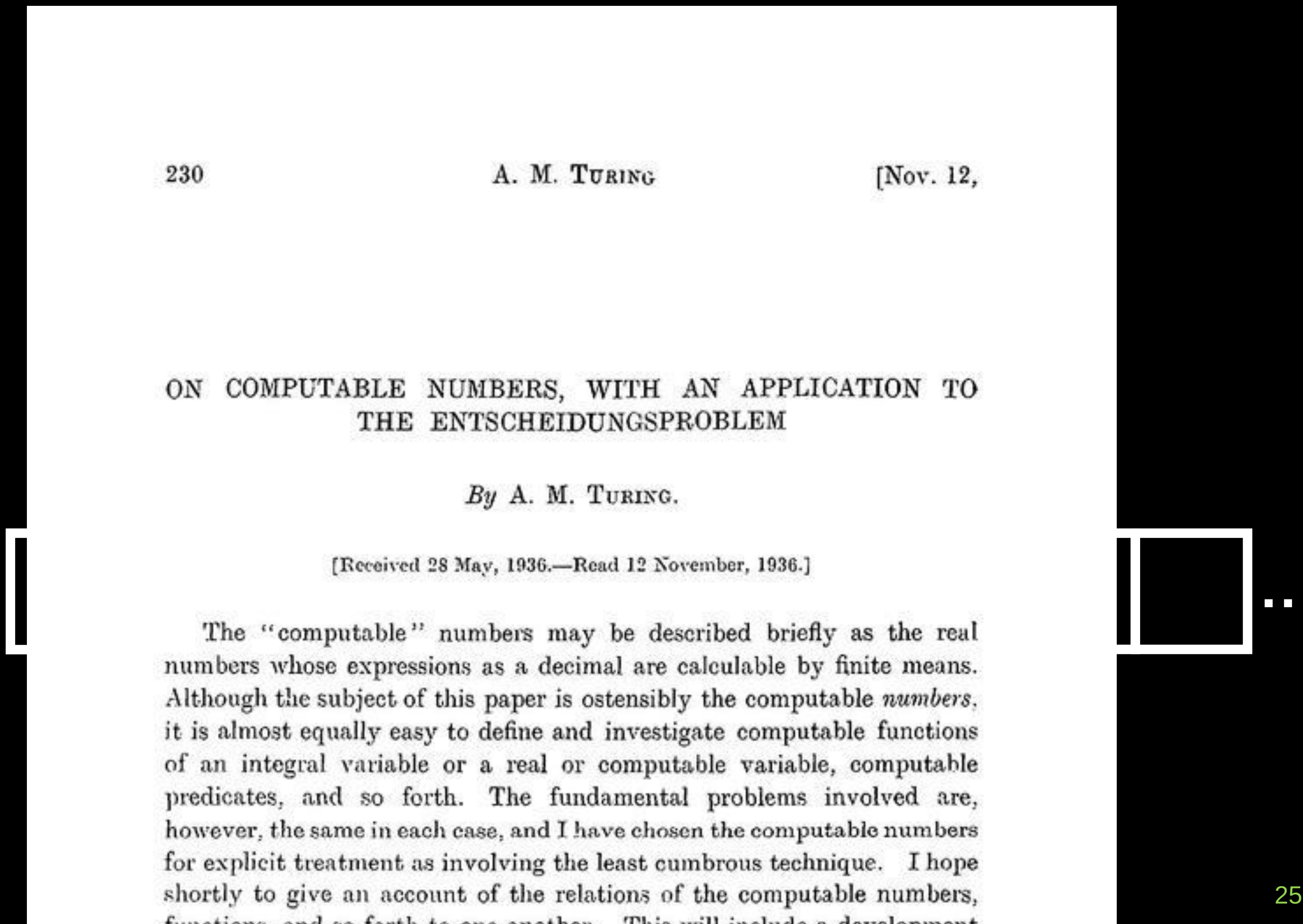


In each step:

- Reads a symbol
- Writes a symbol
- Changes state
- Moves Left or Right

INFINITE REWRITABLE TAPE

Turing Machine (1936)

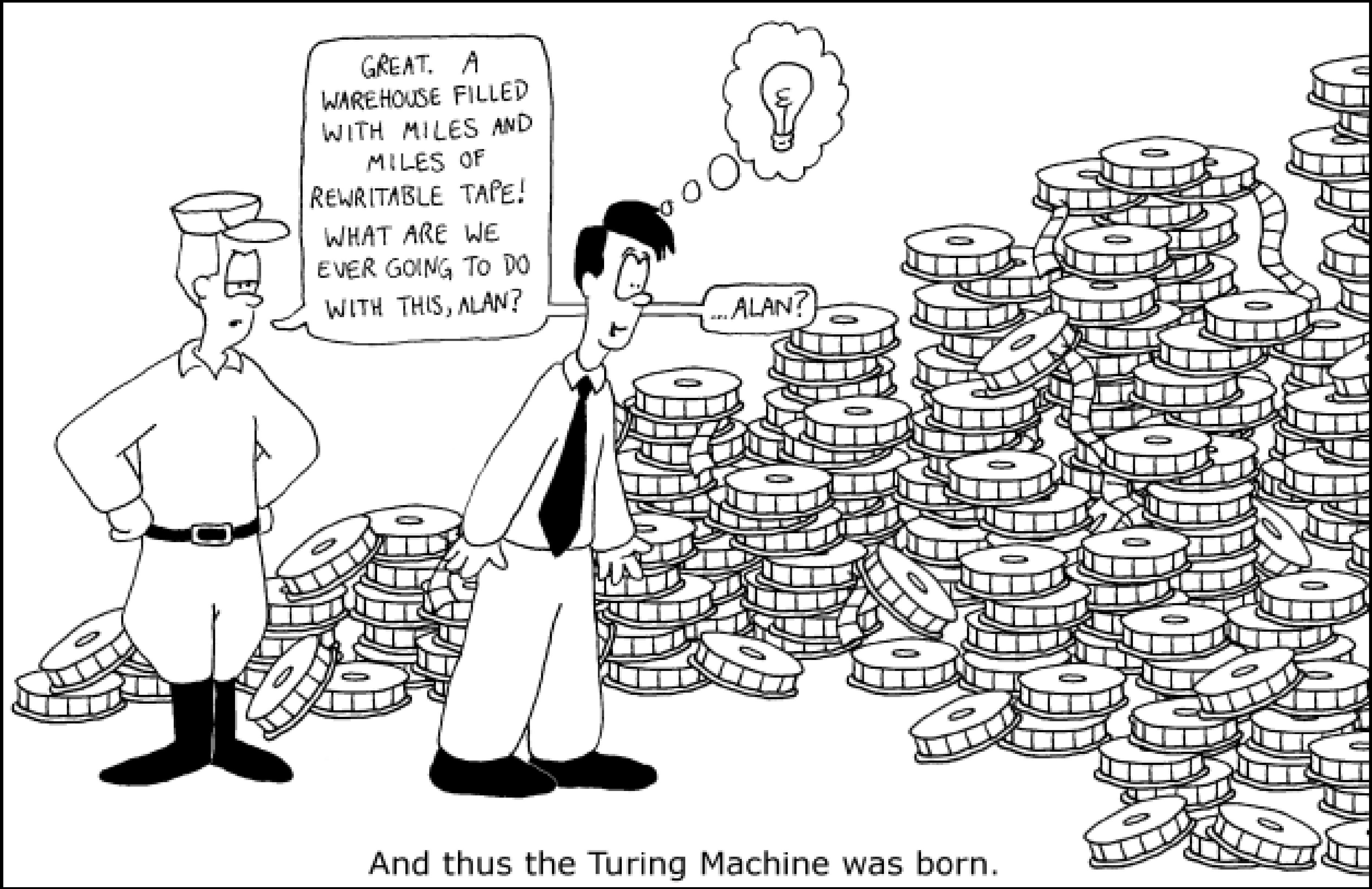


ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

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,



https://www.cs.utah.edu/~draperg/cartoons/2005/turing.html

Turing Machines versus DFAs

The input is written on an infinite tape
with "blank" symbols after the input

The "tape head" can move *right and left*

The TM can both *write to* and *read from* the tape,
and can write symbols that aren't part of input

Accept and Reject take immediate effect

A TM for L = { w#w | w $\in$ {0,1}* } over Σ={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 **b** with **X**, and check if the first bit b' to the right of the # is identical to **b**. (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:

$\square$ = "blank"

Q is a finite set of states

Σ is the input alphabet, where $\square \notin \Sigma$

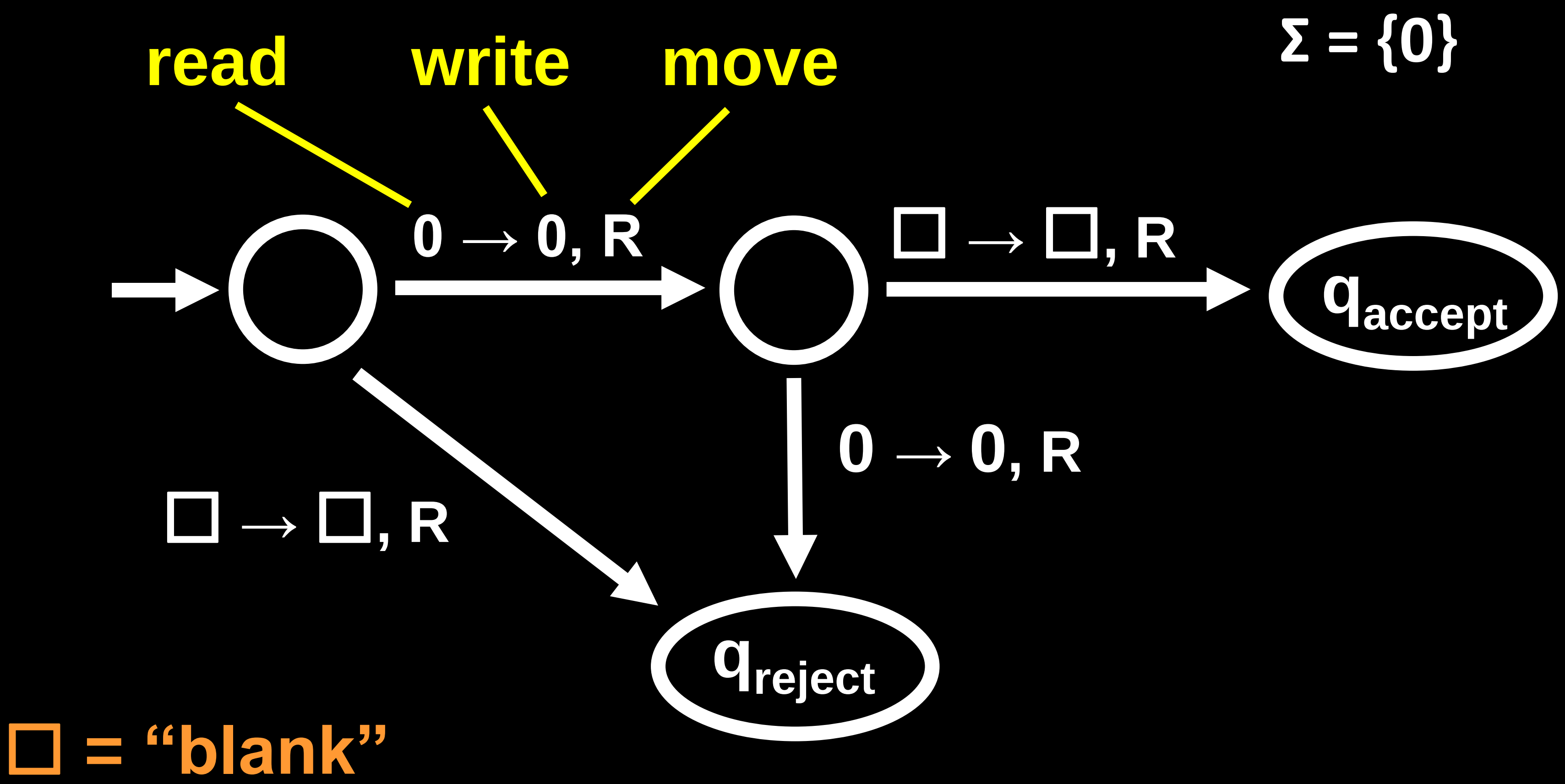
Γ is the tape alphabet, where $\square \in \Gamma$ and $\Sigma \subseteq \Gamma$

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

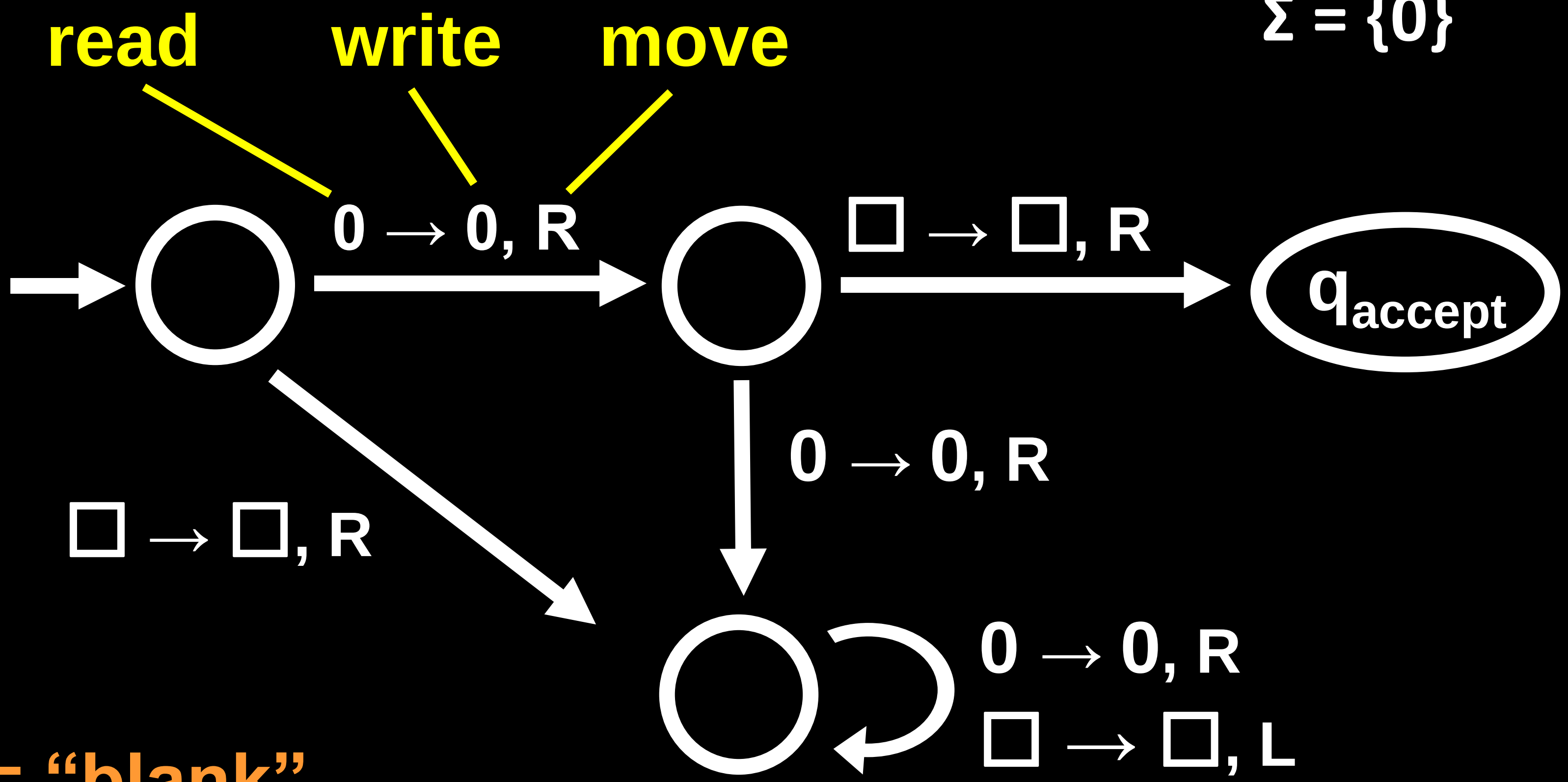
$q_0 \in Q$ is the start state

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

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

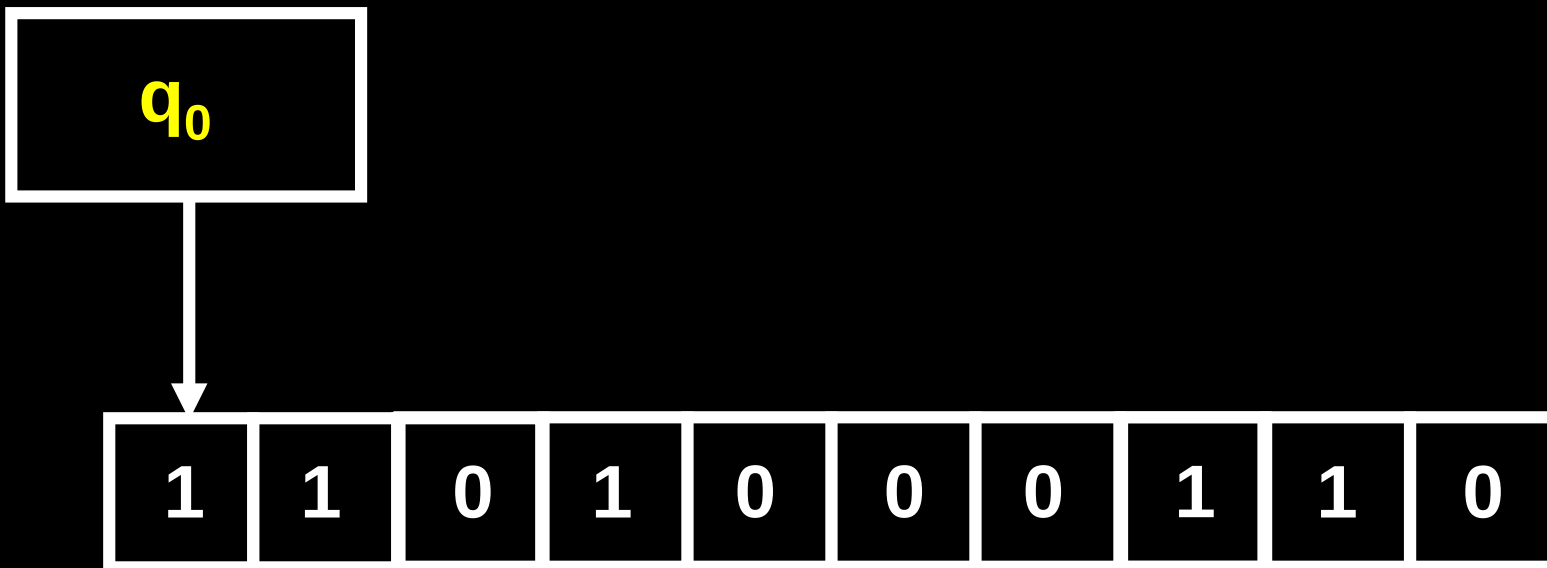

read
write
move
Σ = {0}
0 → 0, R
□ → □, R
q_accept
0 → 0, R
□ → □, R
q_reject
□ = "blank"

$\Sigma = \{0\}$



□ = "blank"

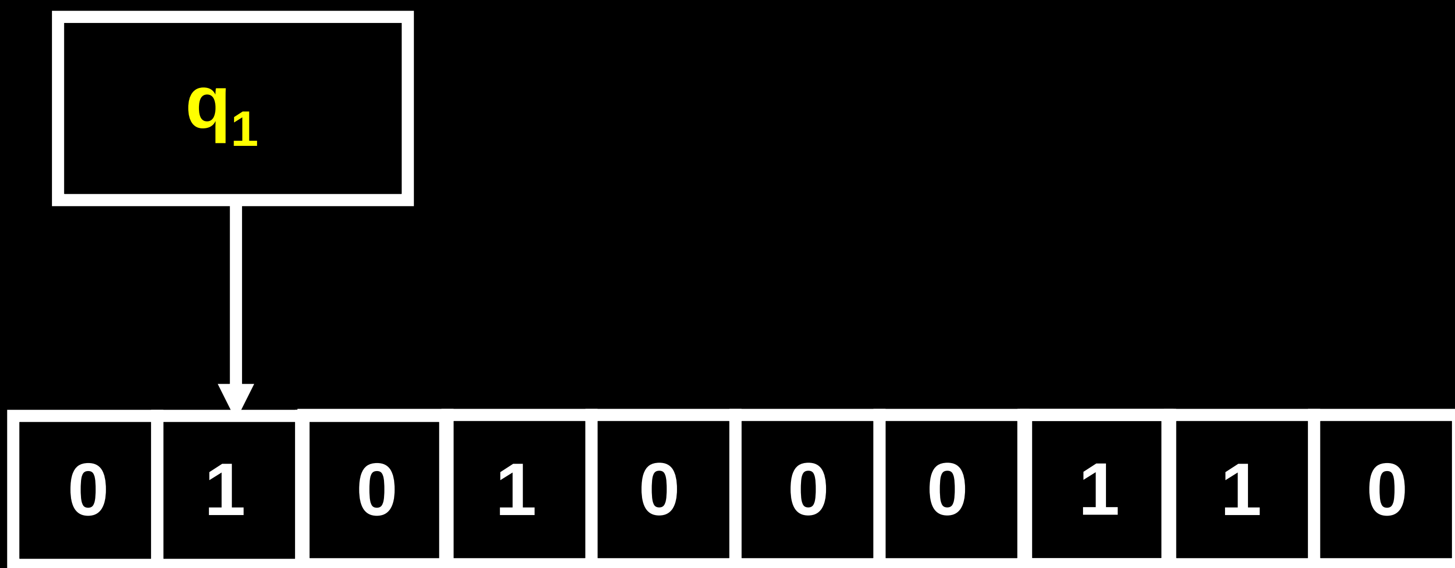
Turing Machine Configurations



corresponds to the *configuration*:

$$q_0 1101000110 \in (Q \cup \Gamma)^*$$

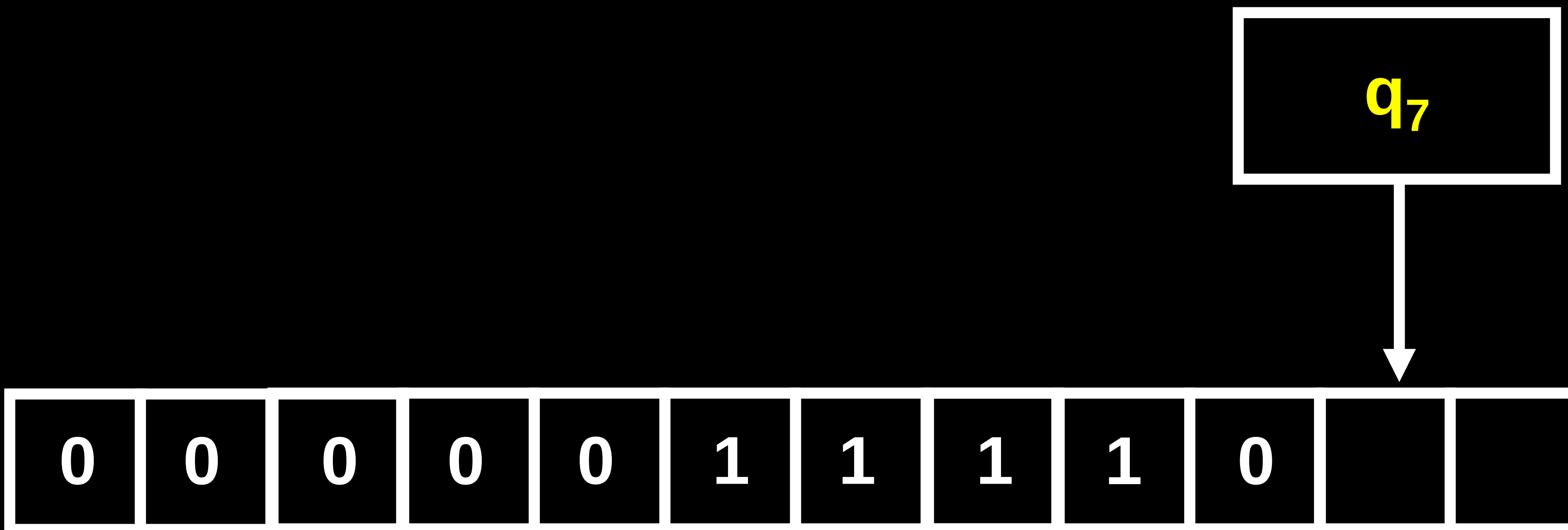
Turing Machine Configurations



corresponds to the *configuration*:

$$0q_1101000110 \in (Q \cup \Gamma)^*$$

Turing Machine Configurations



corresponds to the *configuration*:

$$0000011110q_7\square \in (Q \cup \Gamma)^*$$

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

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

Then aq_1bb yields q_2acb

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

Then abq_1a yields $abcq_2\square$

accepting computation history of M on x

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.

- $C_0 = q_0w$ [the initial configuration]
- C_i yields C_{i+1} for i = 0, ..., 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

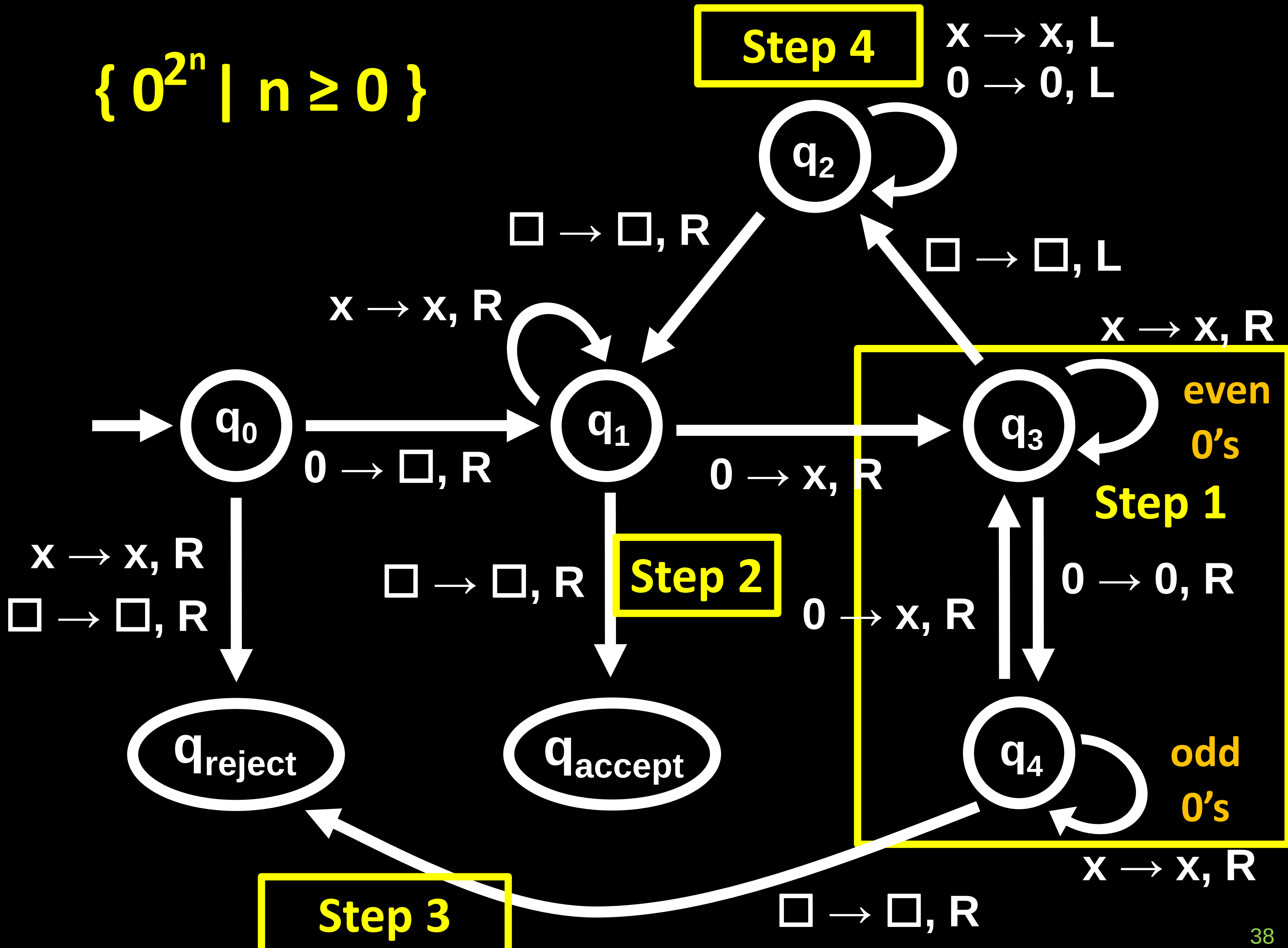
A language L is ***decidable (a.k.a. recursive)***
if some TM **decides** L

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

Turing Machine PSEUDOCODE:

1. Sweep from left to right, **x**-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 left to the first input symbol.
5. Go to step 1.

Why does this work?


{ 0^(2^n) | n ≥ 0 }
Step 4
x → x, L
0 → 0, L
q2
□ → □, R
□ → □, L
x → x, R
q0
0 → □, R
q1
0 → x, R
q3
x → x, R
even
0’s
Step 1
x → x, R
□ → □, R
□ → □, R
Step 2
0 → x, R
0 → 0, R
qreject
qaccept
q4
odd
0’s
x → x, R
Step 3
□ → □, R