بسم الله الرحمن الرحیم

فصل دوازدهم

حدود محاسبات الگوریتمی (۱)

Limits of Algorithmic Computation (1)

کاظم فولادی
kazim@fouladi.ir
دانشکده‌ی مهندسی برق و کامپیوتر
دانشگاه تهران

Decidability

Consider problems with answer YES or NO

Examples:

- Does Machine M have three states ?

- Is string w a binary number?

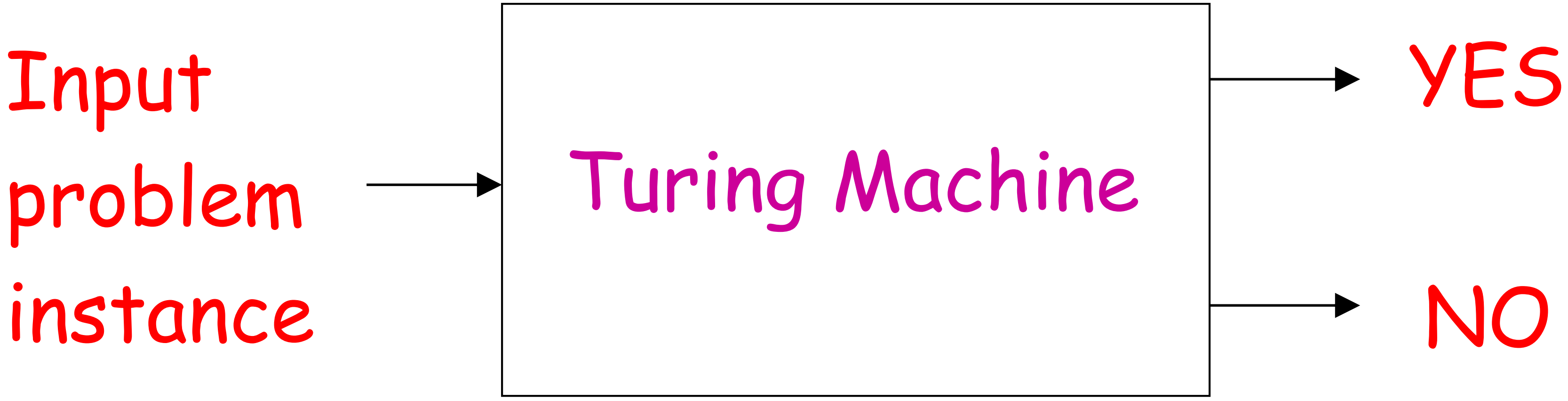
- Does DFA M accept any input?

A problem is decidable if some Turing machine Solves (decides) the problem

Decidable problems:

- Does Machine M have three states ?
- Is string w a binary number?
- Does DFA M accept any input?

The Turing machine that solves a problem answers YES or NO for each instance



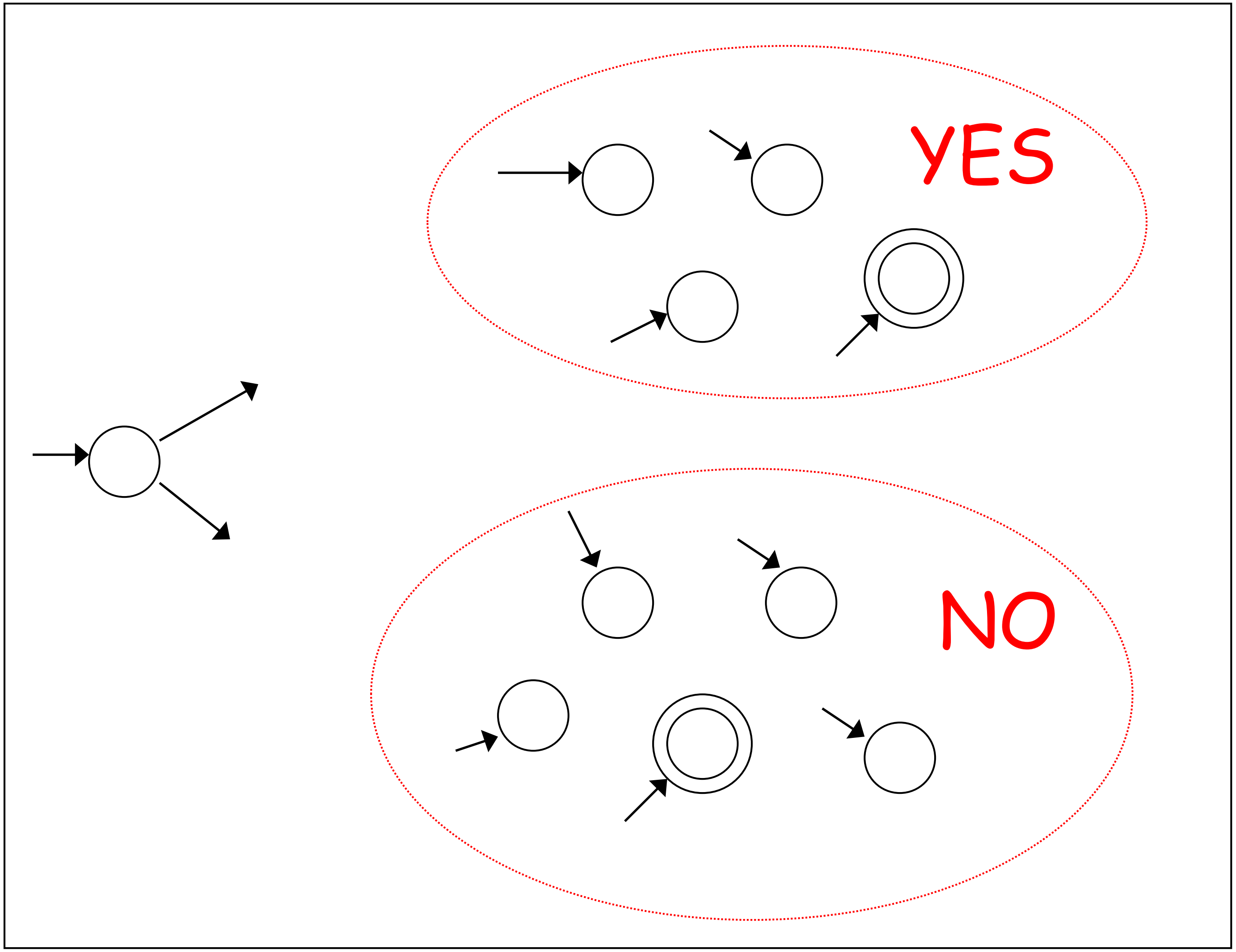
The machine that decides a problem:

- If the answer is YES
  then halts in a <u>yes state</u>

- If the answer is NO
  then halts in a <u>no state</u>

These states may not be final states

Turing Machine that decides a problem



YES and NO states are halting states

Difference between
Recursive Languages and Decidable problems

For decidable problems:

The YES states may not be final states

Some problems are undecidable:

There is no Turing Machine that solves all instances of the problem

A simple undecidable problem:

The membership problem

The Membership Problem

Input:

- Turing Machine M
- String w

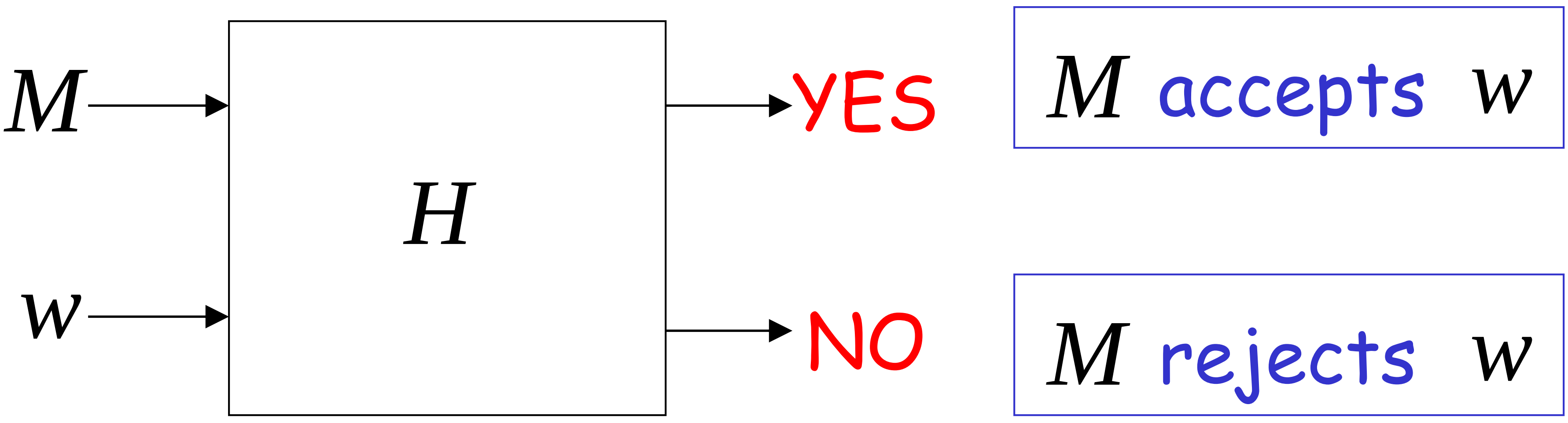
Question: Does M accept w ?

Theorem:

The membership problem is undecidable

Proof: Assume for contradiction that the membership problem is decidable

There exists a Turing Machine H
that solves the membership problem



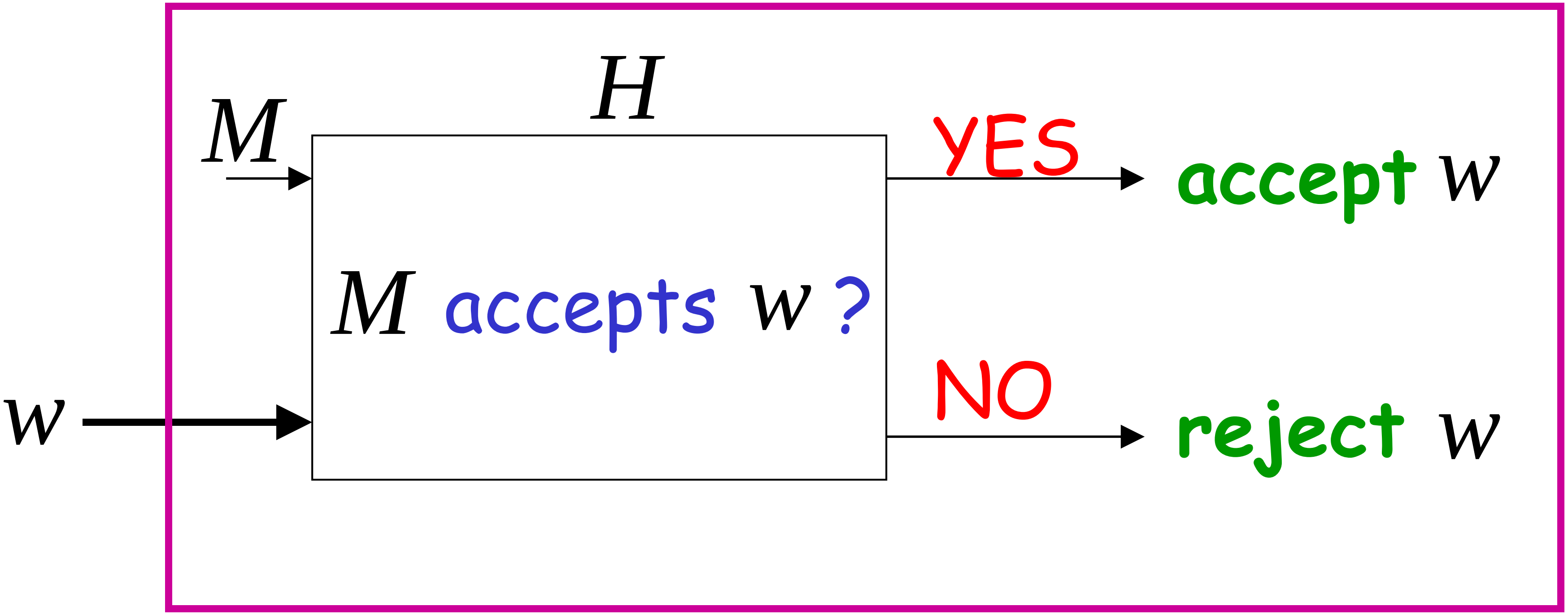
Let L be a recursively enumerable language

Let M be the Turing Machine that accepts L

We will prove that L is also recursive:

we will describe a Turing machine that accepts L and halts on any input

Turing Machine that accepts L and halts on any input



Therefore, L is recursive

Since L is chosen arbitrarily, we have proven that every recursively enumerable language is also recursive

But there are recursively enumerable languages which are not recursive

Contradiction!!!!

Therefore, the membership problem is undecidable

END OF PROOF

A famous undecidable problem:

The halting problem

The Halting Problem

Input:
- Turing Machine M
- String w

Question: Does M halt on w ?

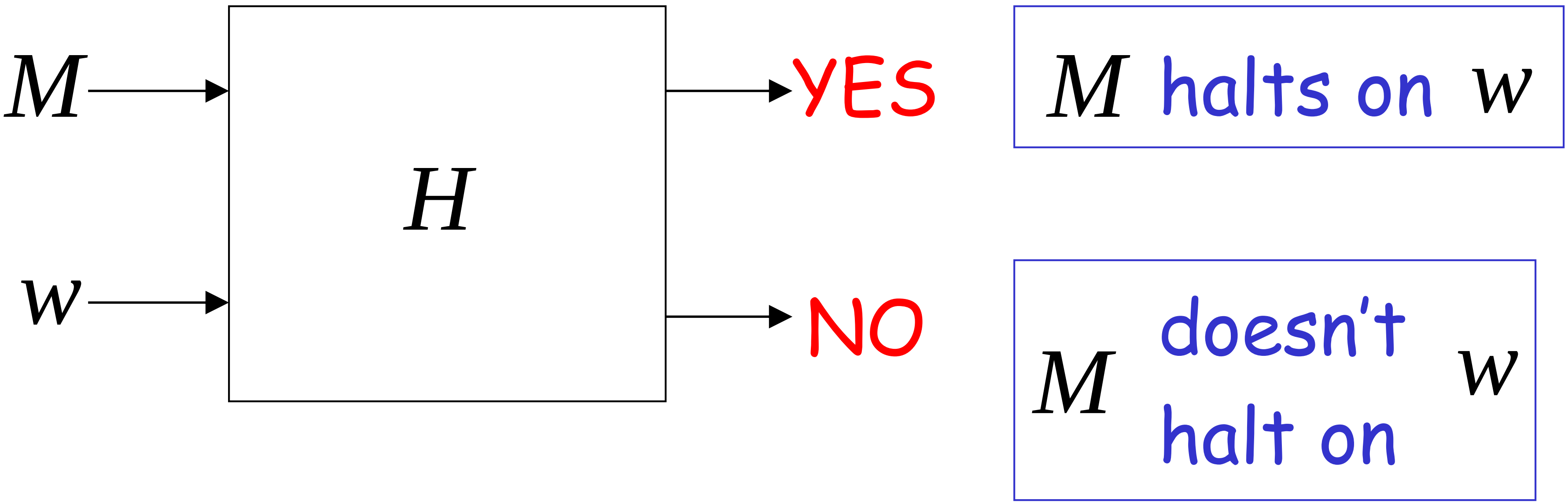
Theorem:

The halting problem is undecidable

Proof: Assume for contradiction that the halting problem is decidable

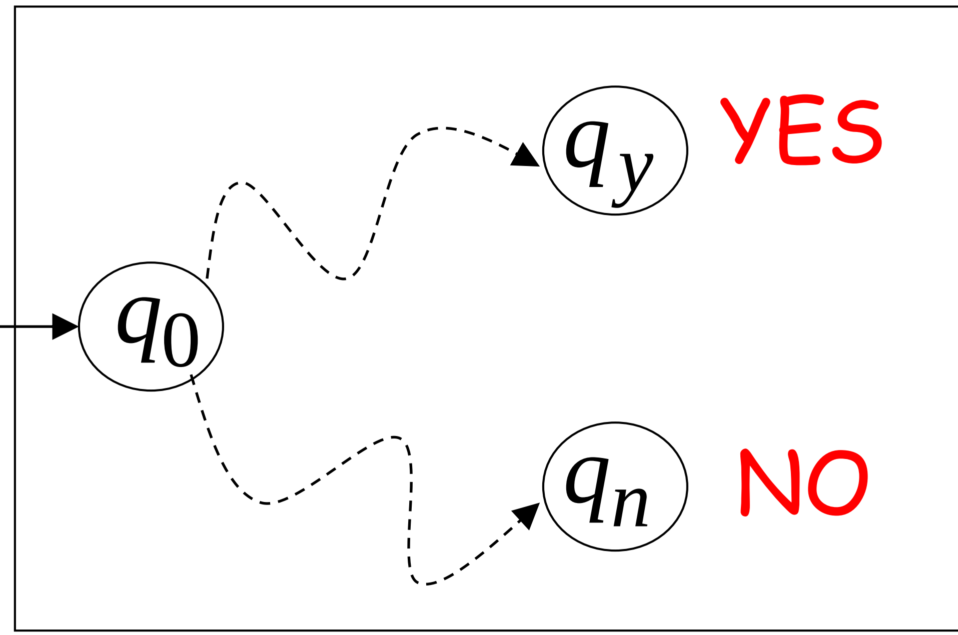
There exists Turing Machine H

that solves the halting problem



Construction of H

H



Input:
initial tape contents

$w_M \quad w$

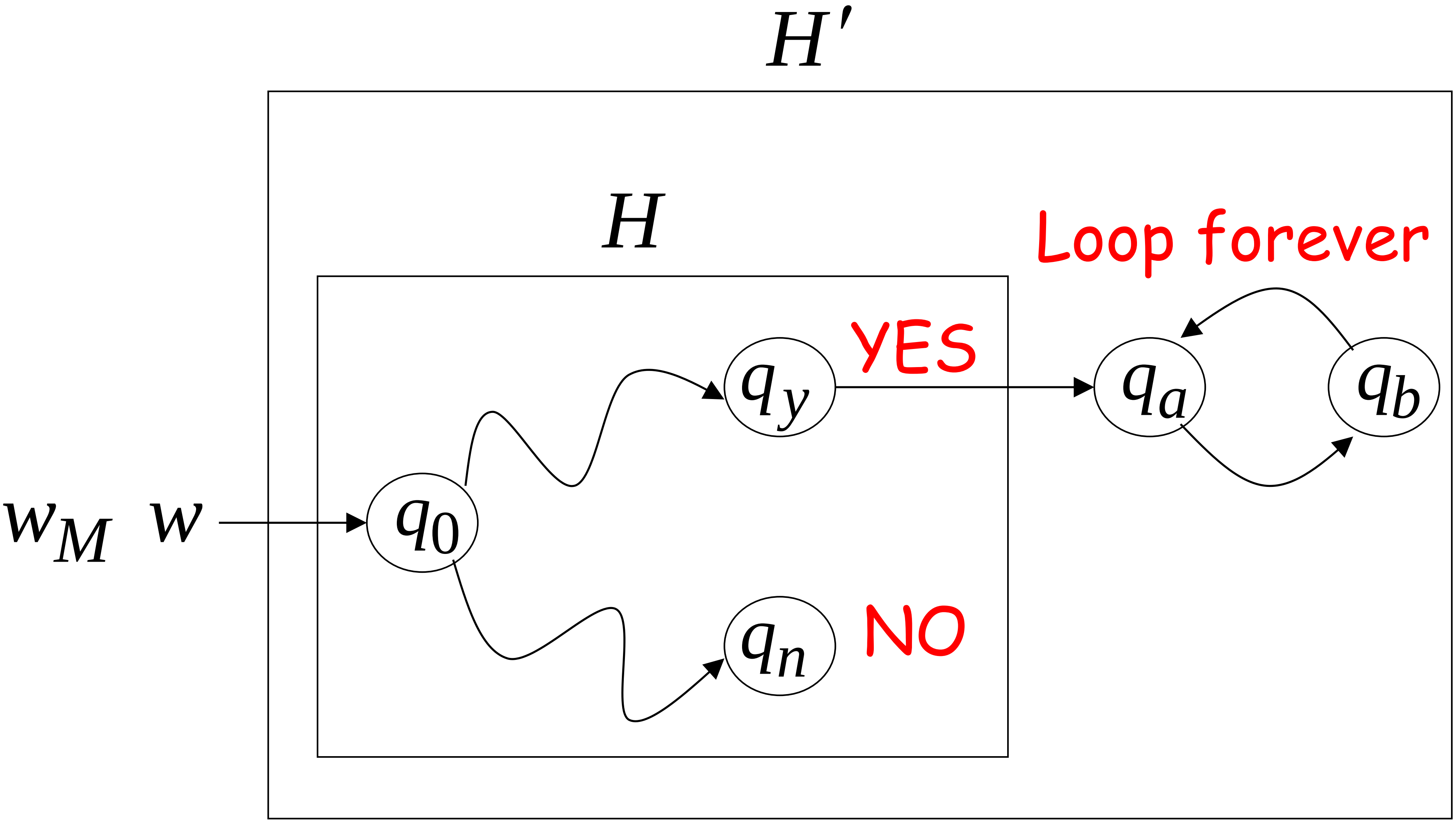
Encoding of M

String w

Construct machine H' :

If H returns YES then loop forever

If H returns NO then halt


H'
H
Loop forever
YES
NO
q_y
q_a
q_b
q_0
q_n
w_M w

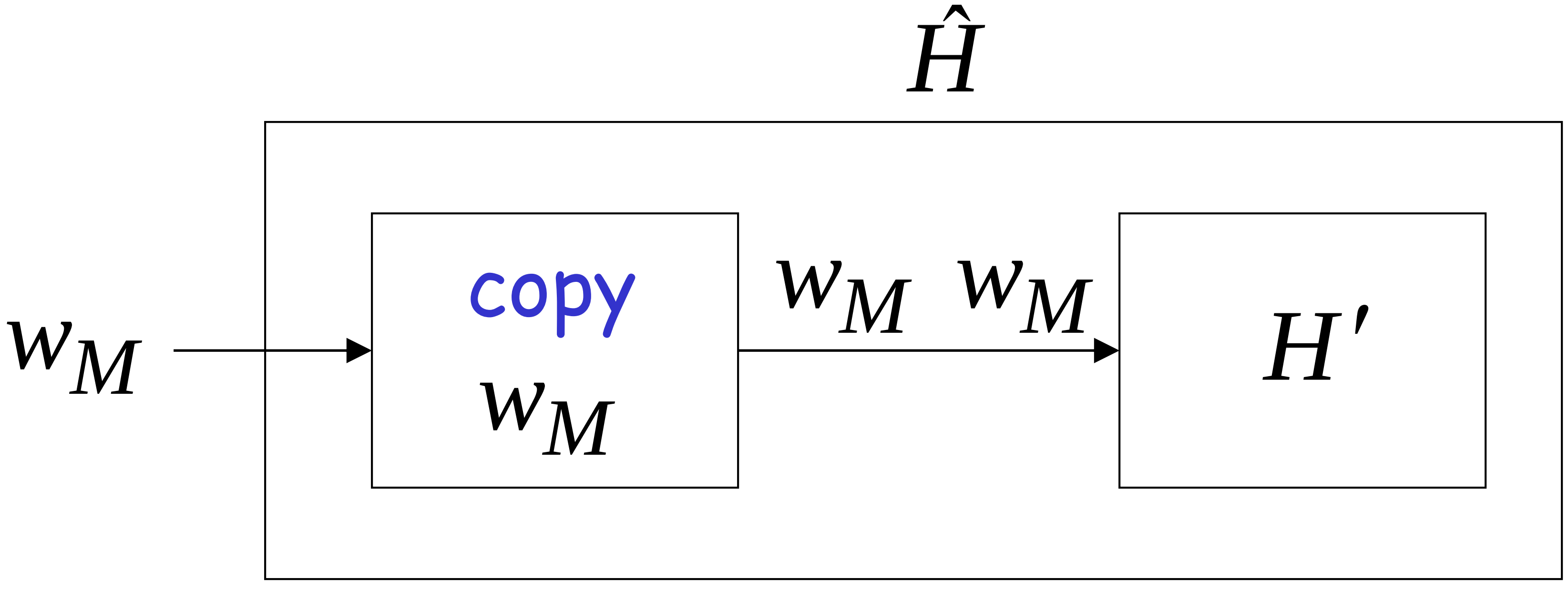
Construct machine $\hat{H}$:

Input: w_M (machine M)

If M halts on input w_M

<u>**Then**</u> loop forever

<u>**Else**</u> halt


$\hat{H}$
w_M
copy
w_M
w_M w_M
H'

Run machine $\hat{H}$ with input itself:

Input: $w_{\hat{H}}$ (machine $\hat{H}$)

If $\hat{H}$ halts on input $w_{\hat{H}}$

<u>Then</u> loop forever

<u>Else</u> halt

$\hat{H}$ on input $w_{\hat{H}}$:

If $\hat{H}$ halts then loops forever

If $\hat{H}$ doesn't halt then it halts

NONSENSE !!!!!

Therefore, we have contradiction

The halting problem is undecidable

END OF PROOF

Another proof of the same theorem:

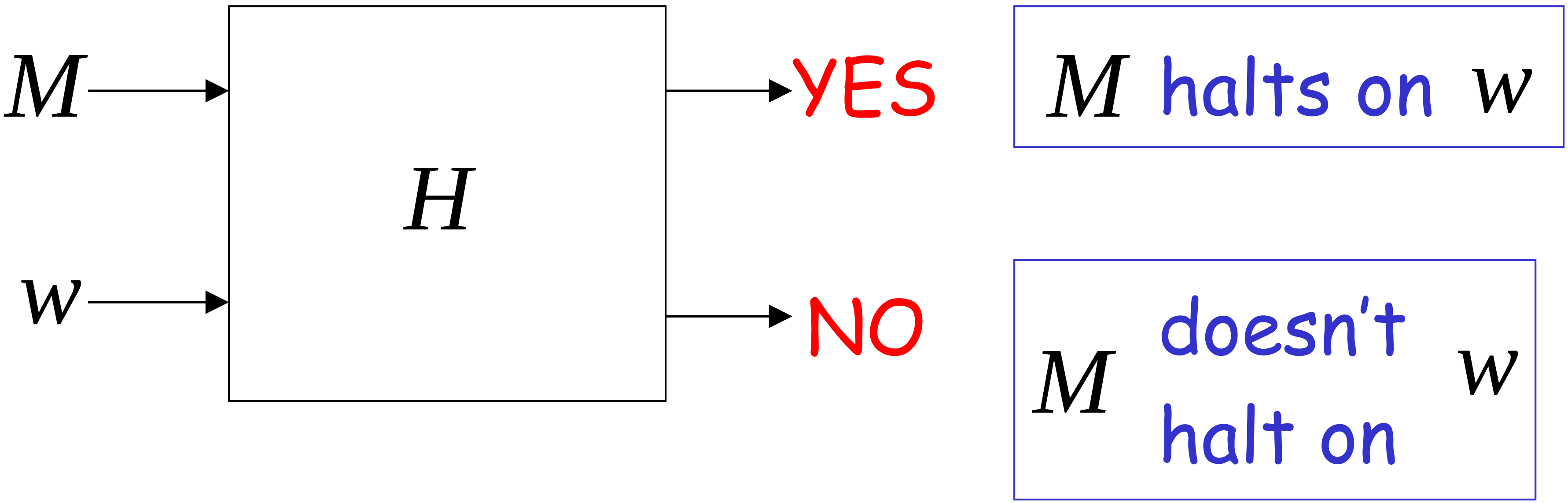
If the halting problem was decidable then every recursively enumerable language would be recursive

Theorem:

The halting problem is undecidable

Proof: Assume for contradiction that the halting problem is decidable

There exists Turing Machine H that solves the halting problem



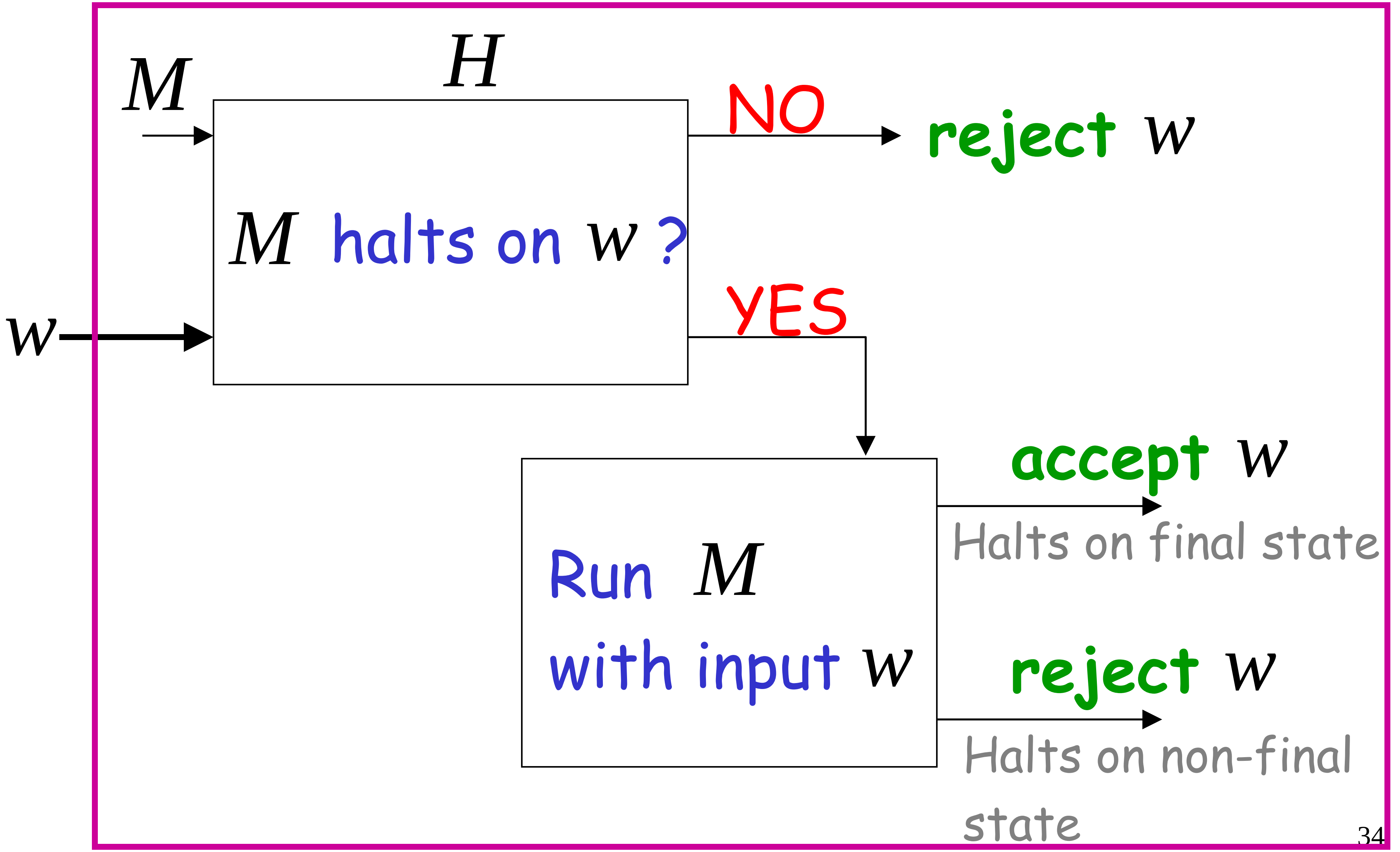
Let L be a recursively enumerable language

Let M be the Turing Machine that accepts L

We will prove that L is also recursive:

we will describe a Turing machine that accepts L and halts on any input

Turing Machine that accepts L and halts on any input



Therefore L is recursive

Since L is chosen arbitrarily, we have proven that every recursively enumerable language is also recursive

But there are recursively enumerable languages which are not recursive

Contradiction!!!!

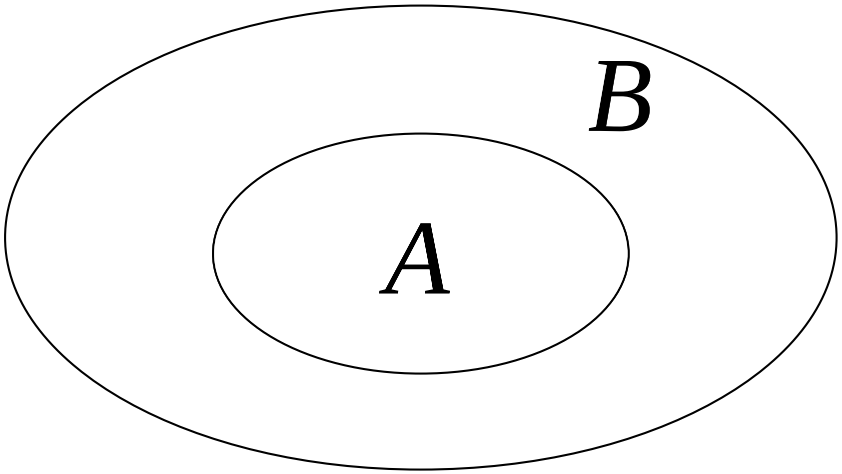
Therefore, the halting problem is undecidable

END OF PROOF

Reducibility

Problem A is reduced to problem B

If we can solve problem B then we can solve problem A



Problem A is reduced to problem B

If B is decidable then A is decidable

If A is undecidable then B is undecidable

Example: the halting problem

is reduced to

the state-entry problem

The state-entry problem

Inputs:

- Turing Machine M
- State q
- String w

Question: Does M enter state q on input w ?

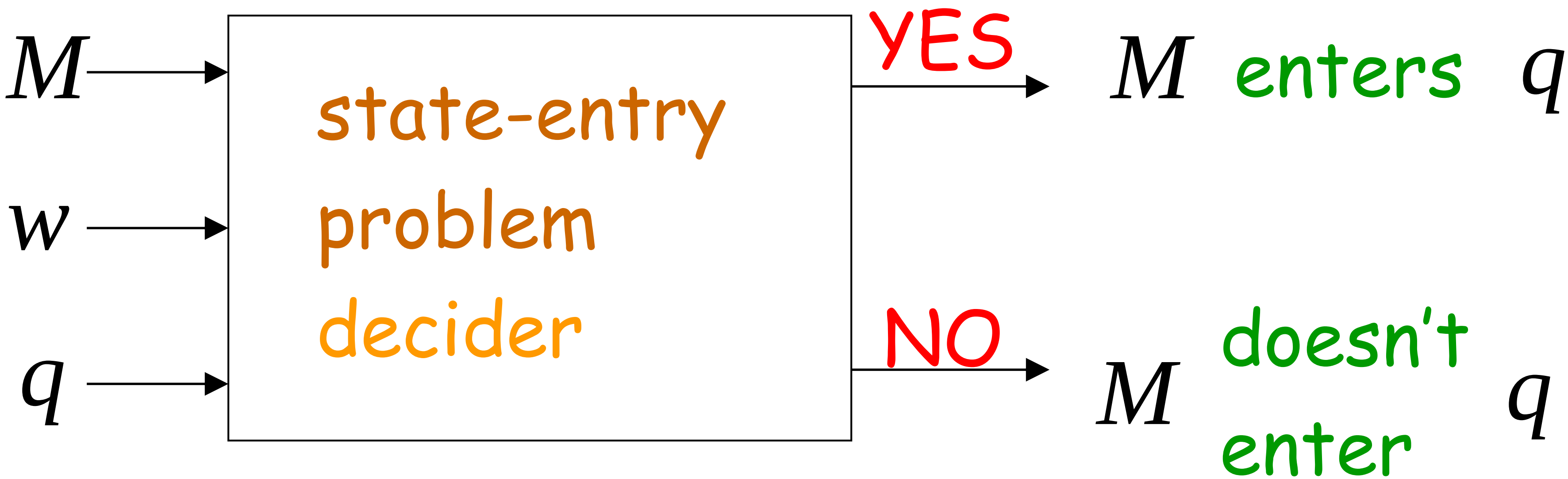
Theorem:

The state-entry problem is undecidable

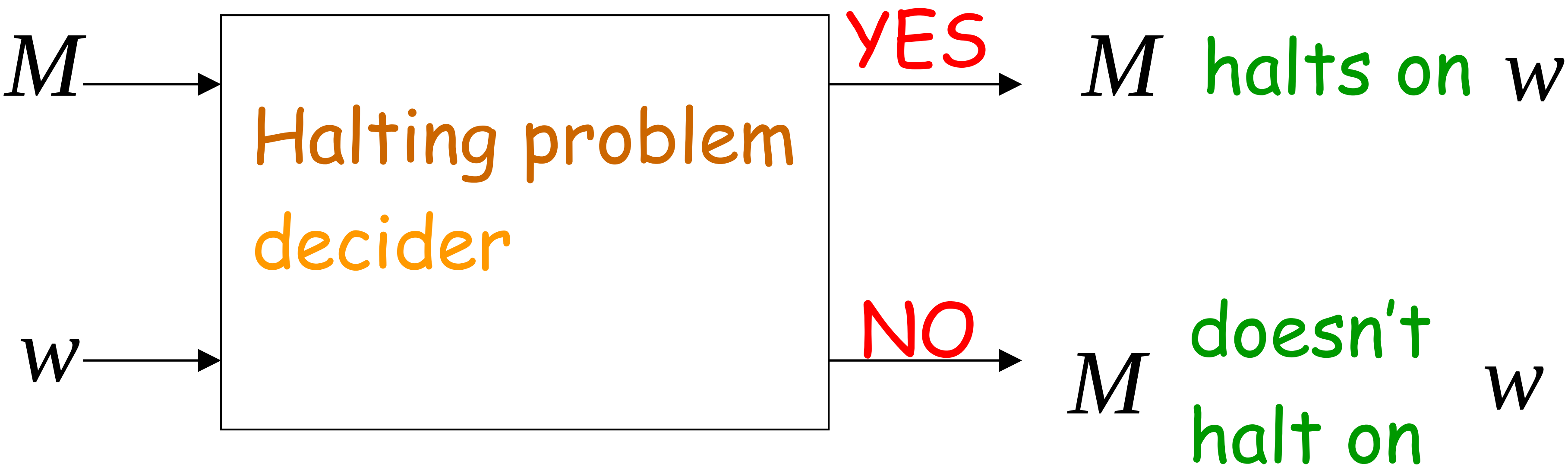
Proof: Reduce the halting problem to

the state-entry problem

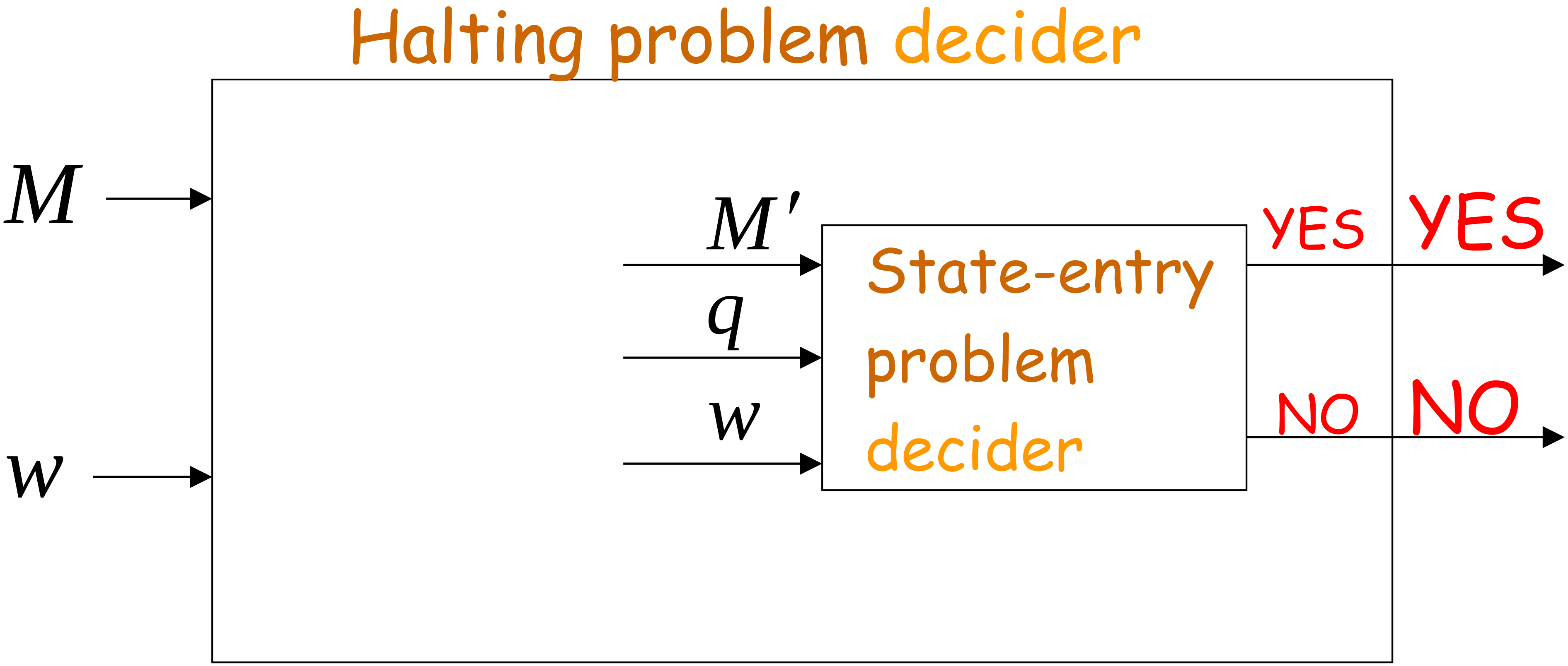
Suppose we have a Decider
for the state-entry algorithm:



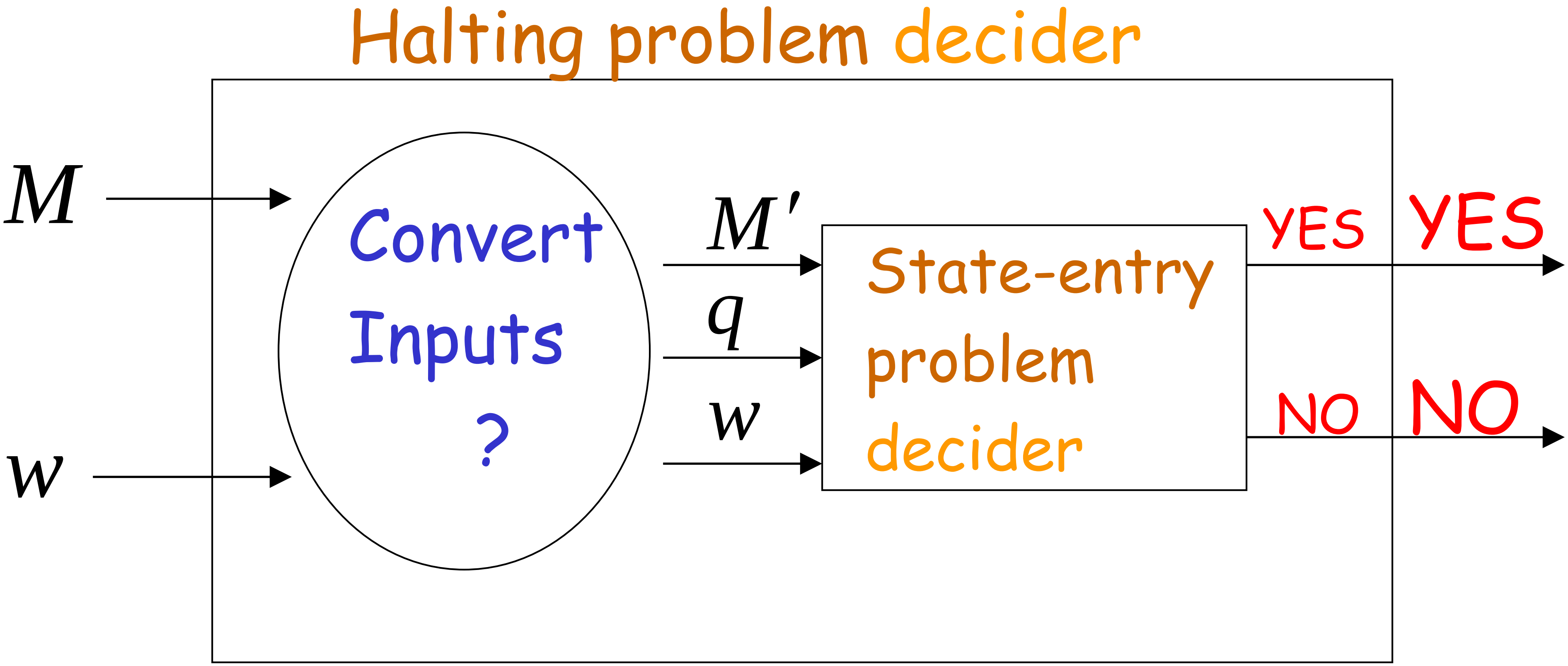
We want to build a decider for the halting problem:



We want to reduce the halting problem to the state-entry problem:



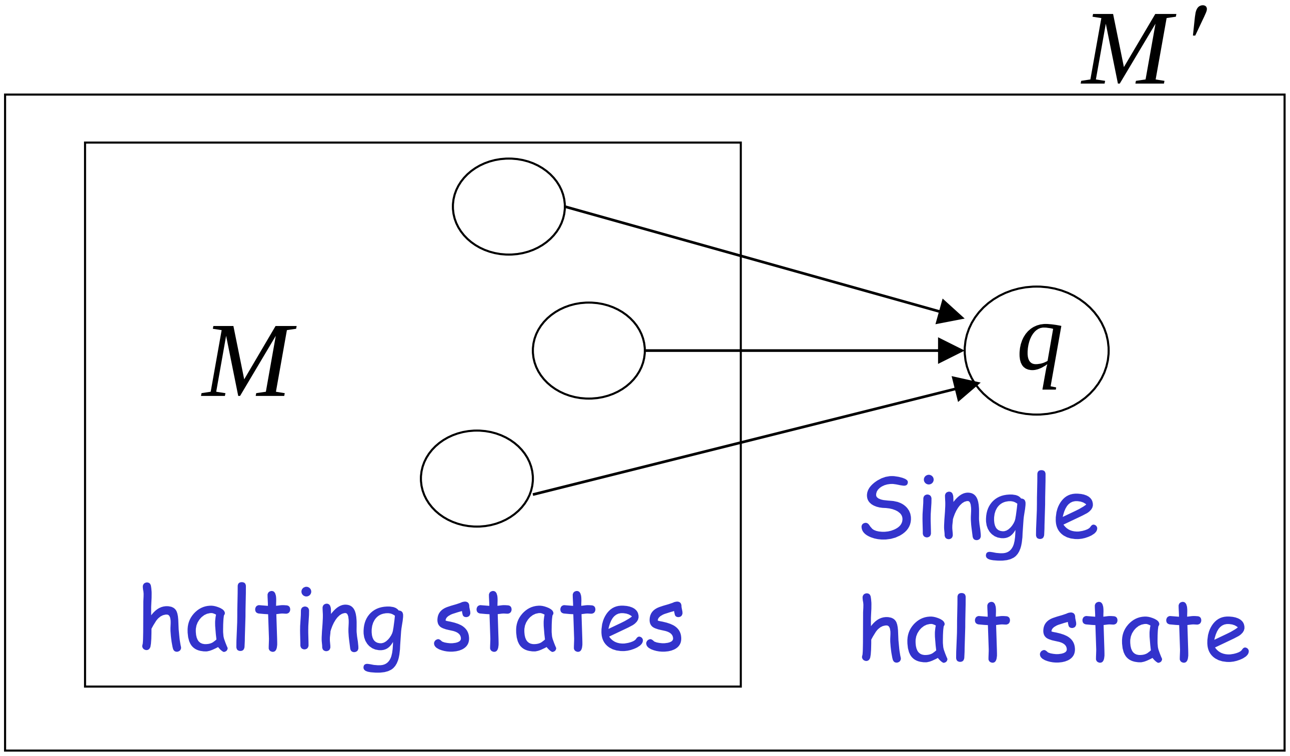
We need to convert one problem instance to the other problem instance



Convert M to M' :

- Add new state q

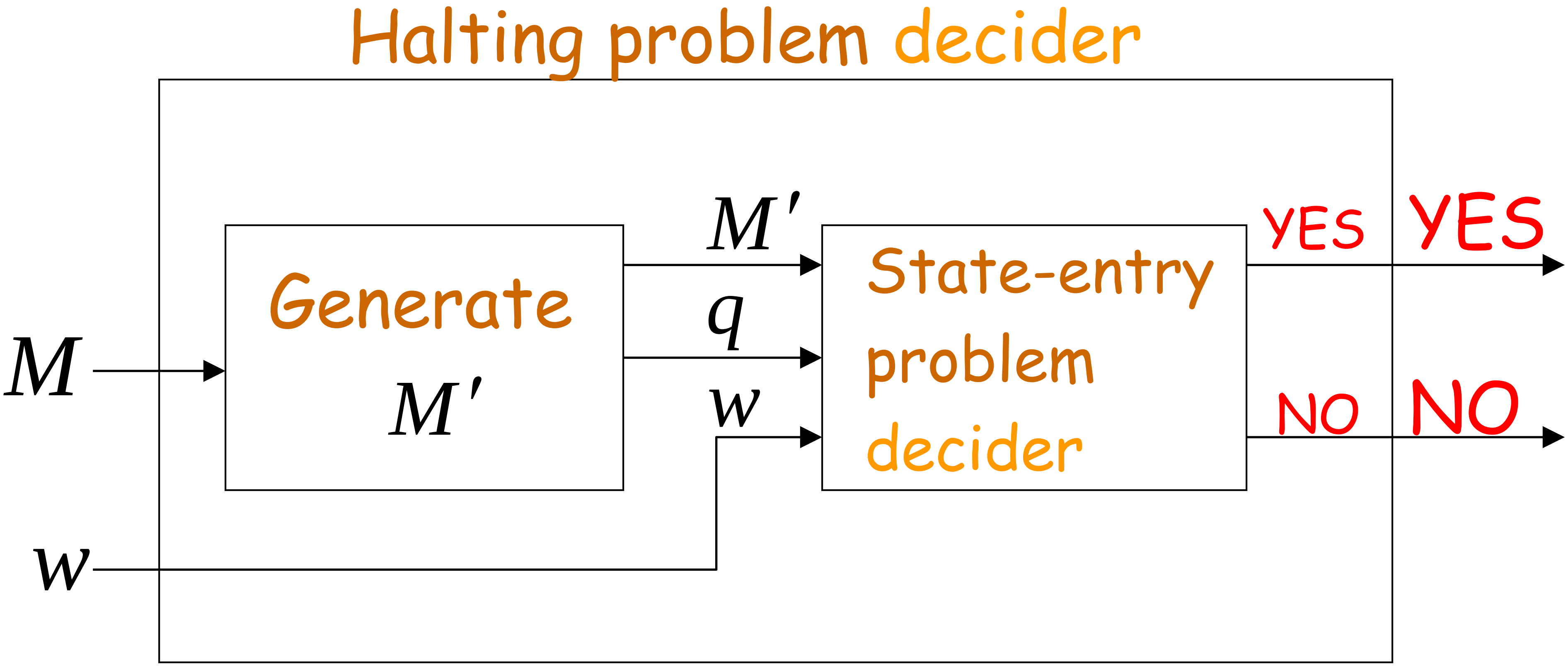
- From any halting state of M add transitions to q



M halts on input w

if and only if

M' halts on state q on input w


Halting problem decider
M
w
Generate
M'
M'
q
w
State-entry
problem
decider
YES
NO
YES
NO

We reduced the halting problem
to the state-entry problem

Since the halting problem is undecidable,
the state-entry problem is undecidable

END OF PROOF

Another example:

the halting problem

is reduced to

the blank-tape halting problem

The blank-tape halting problem

Input: Turing Machine M

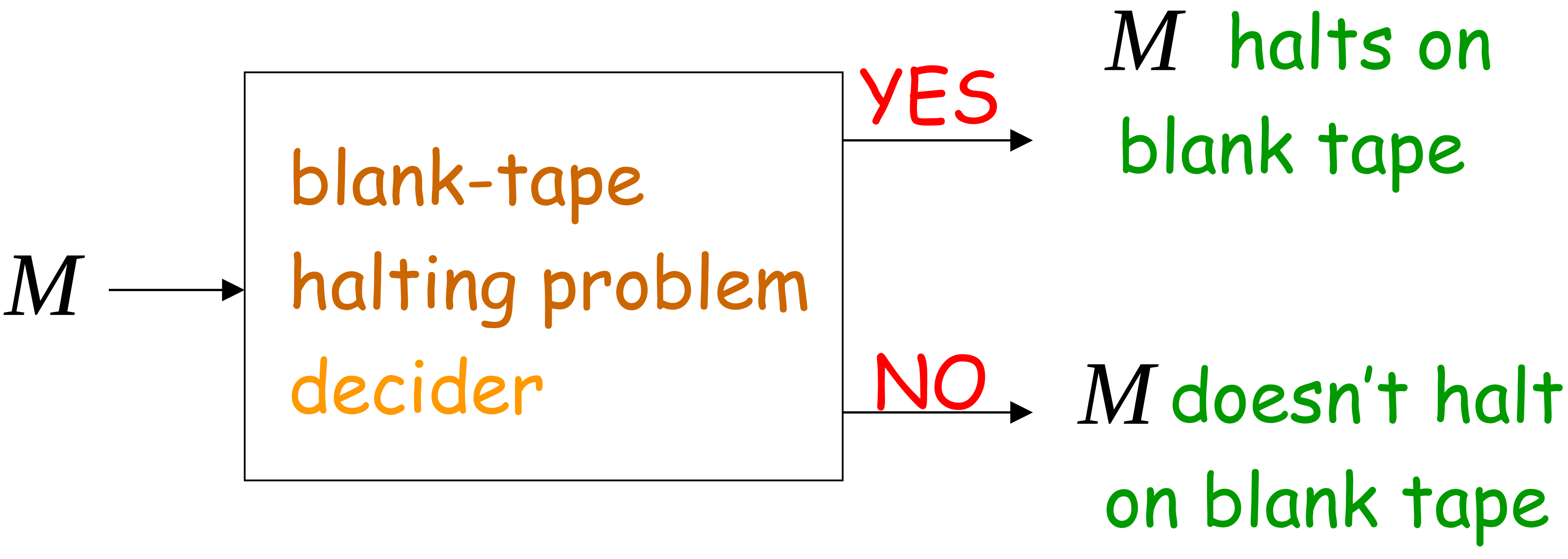
Question: Does M halt when started with a blank tape?

Theorem:

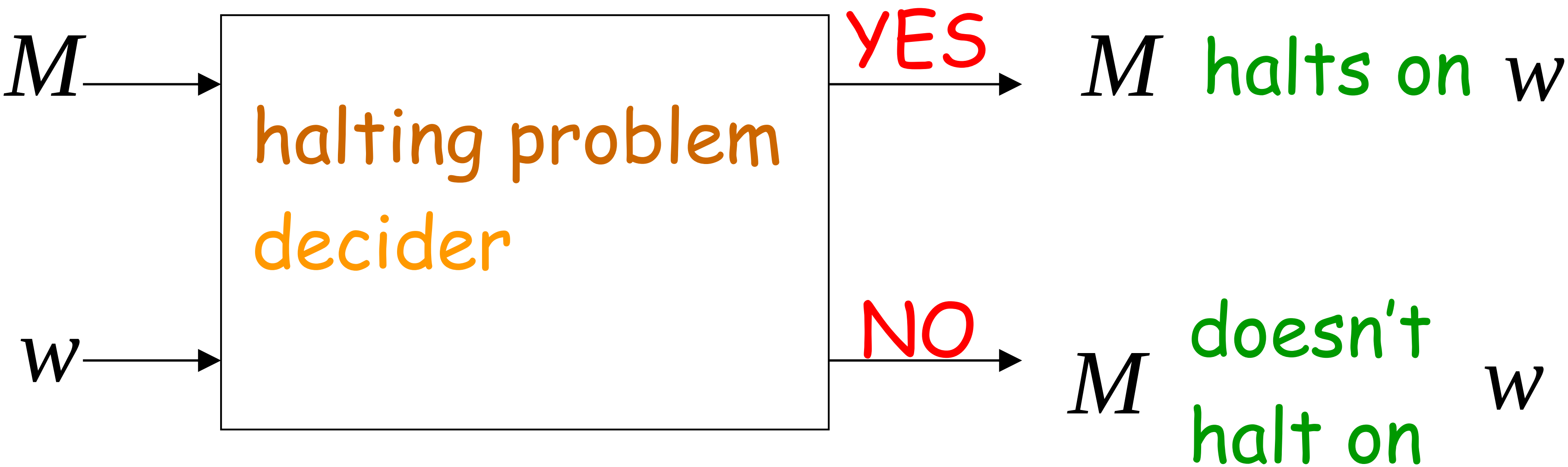
The blank-tape halting problem is undecidable

Proof: Reduce the halting problem to the blank-tape halting problem

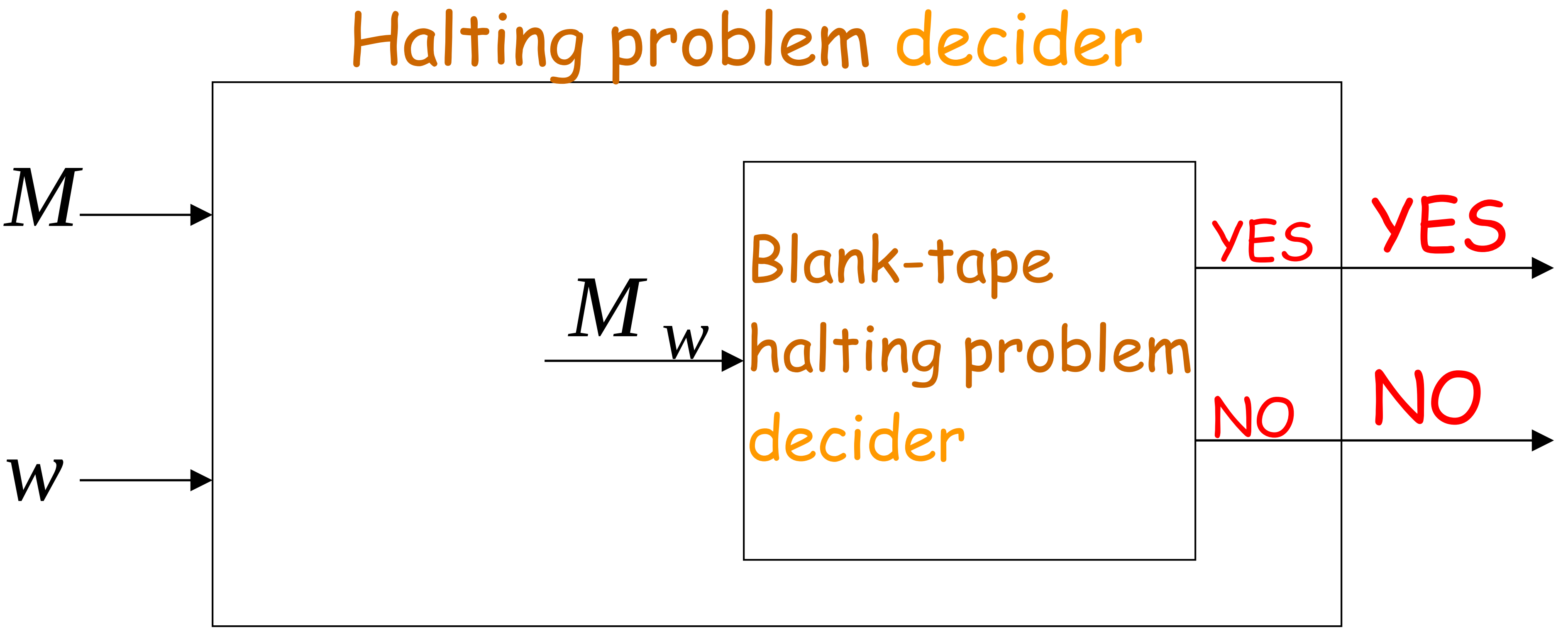
Suppose we have a decider for the blank-tape halting problem:



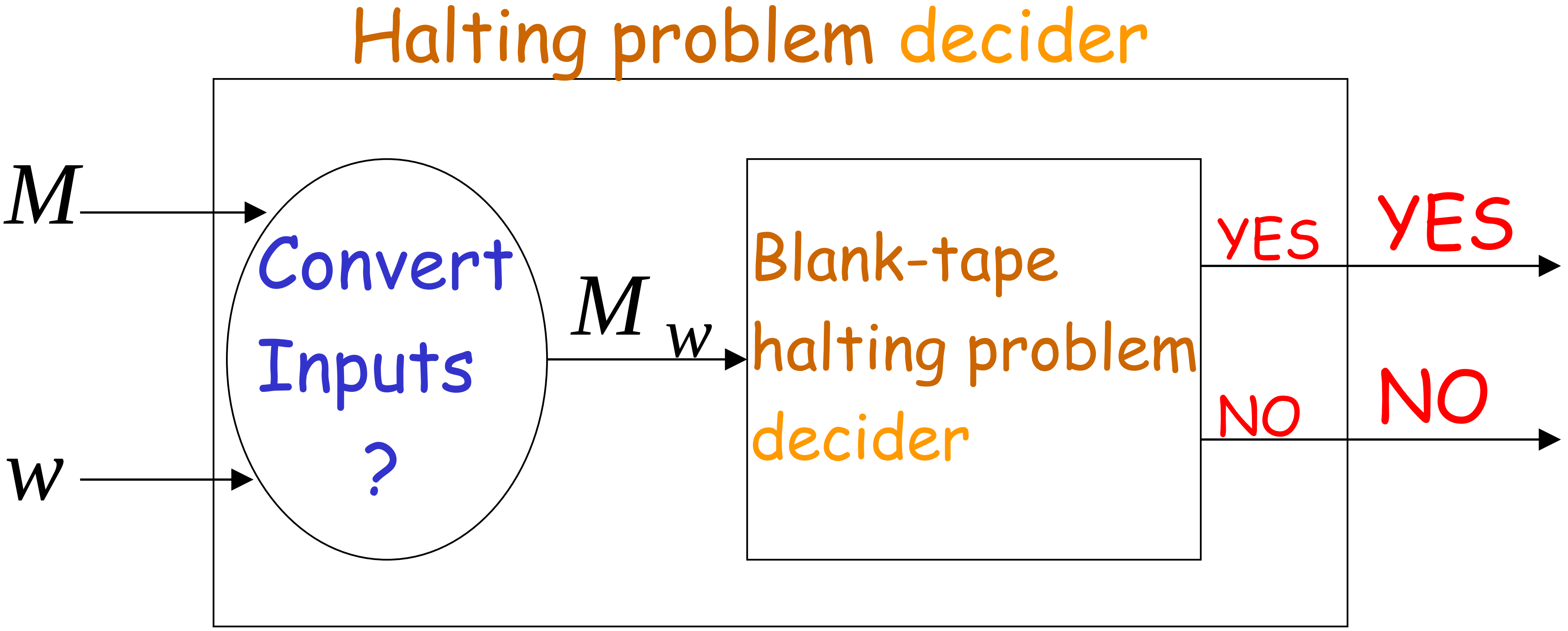
We want to build a decider for the halting problem:



We want to reduce the halting problem to the blank-tape halting problem:



We need to convert one problem instance to the other problem instance



Construct a new machine M_w

- When started on blank tape, writes w

- Then continues execution like M

M_w

step 1

if blank tape

then write w

step2

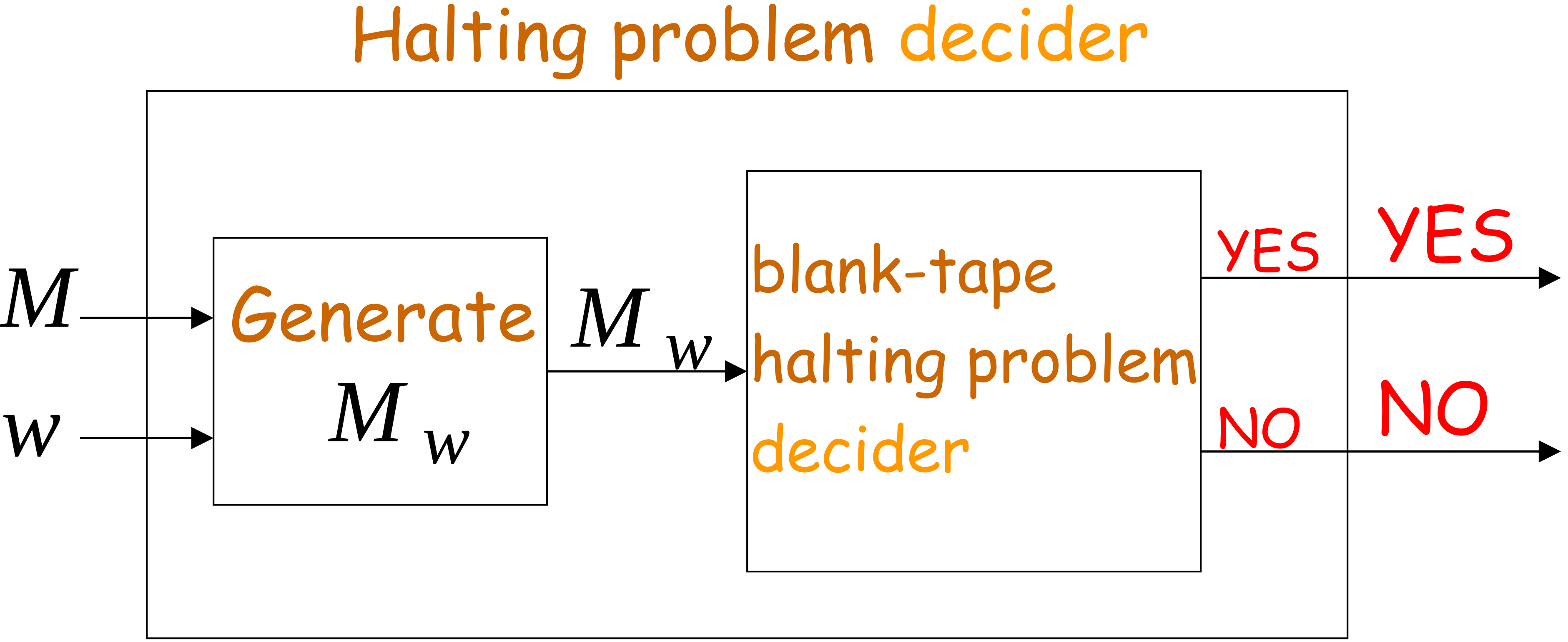
execute M

with input w

M halts on input string w

if and only if

M_w halts when started with blank tape


Halting problem decider
M
w
Generate
M_w
M_w
blank-tape
halting problem
decider
YES
NO
YES
NO

We reduced the halting problem
to the blank-tape halting problem

Since the halting problem is undecidable,
the blank-tape halting problem is undecidable

END OF PROOF

Summary of Undecidable Problems

Halting Problem:

Does machine M halt on input w?

Membership problem:

Does machine M accept string w?

Blank-tape halting problem:

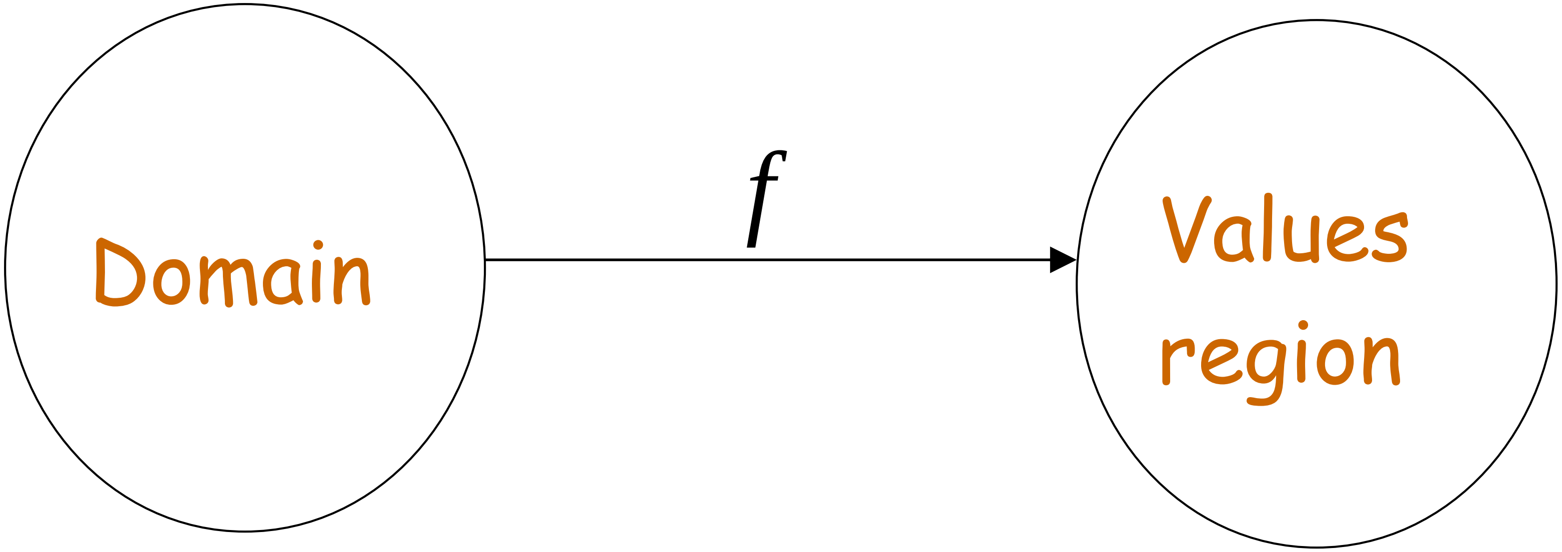
Does machine M halt when starting on blank tape?

State-entry Problem:

Does machine M enter state q on input w ?

Uncomputable Functions

Uncomputable Functions



A function is uncomputable if it cannot be computed for all of its domain

An uncomputable function:

$$f(n) = \begin{cases} \text{maximum number of moves until} \\ \text{any Turing machine with } n \text{ states} \\ \text{halts when started with the blank tape} \end{cases}$$

Theorem: Function $f(n)$ is uncomputable

Proof: Assume for contradiction that $f(n)$ is computable

Then the blank-tape halting problem is decidable

Decider for blank-tape halting problem:

Input: machine M

1. Count states of M : m

2. Compute $f(m)$

3. Simulate M for $f(m)$ steps starting with empty tape

   If M halts then return YES
   otherwise return NO

Therefore, the blank-tape halting problem is decidable

However, the blank-tape halting problem is undecidable

Contradiction!!!

Therefore, function $f(n)$ in uncomputable

END OF PROOF

Undecidable Problems
for
Recursively Enumerable Languages

Take a recursively enumerable language L

Decision problems:

- L is empty?
- L is finite?
- L contains two different strings of the same length?

All these problems are undecidable

Theorem:

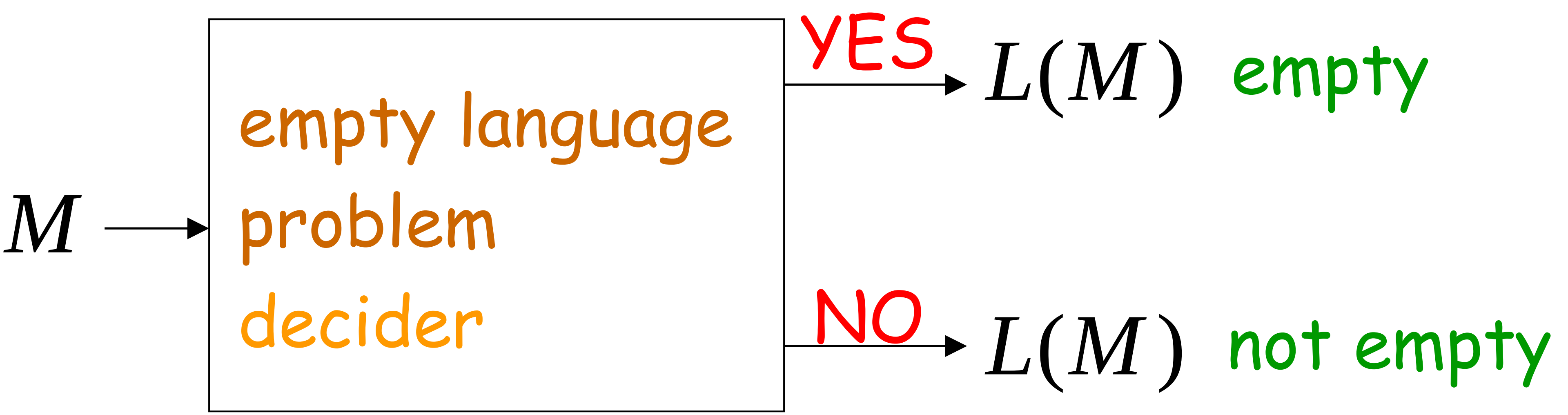
For any recursively enumerable language L it is undecidable to determine whether L is empty

Proof:

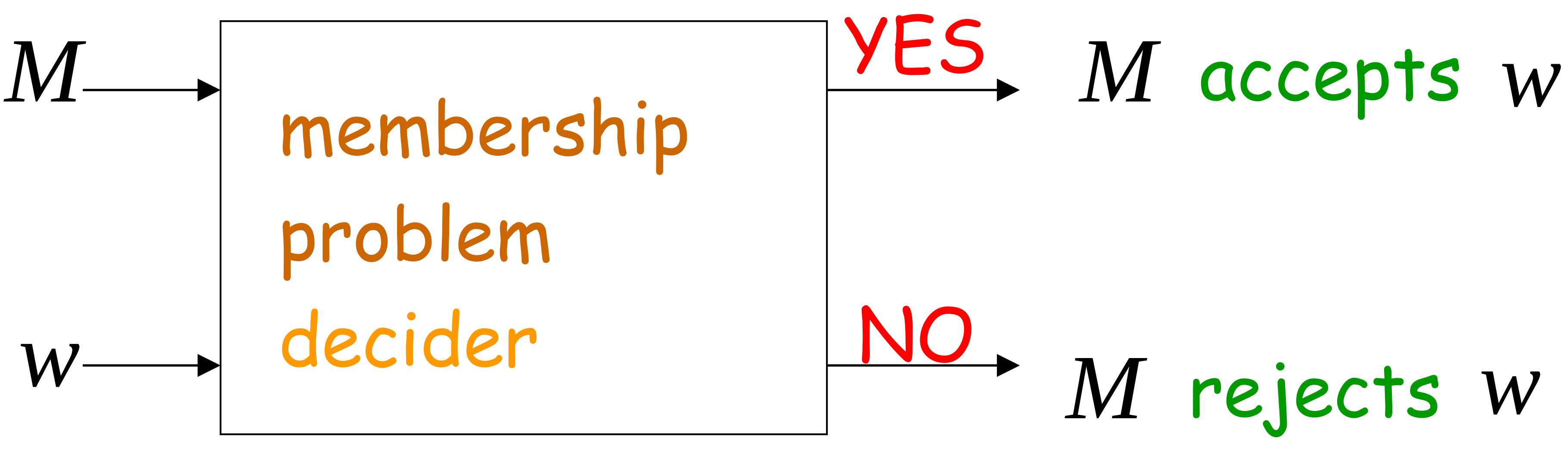
We will reduce the membership problem to this problem

Let M be the TM with $L(M) = L$

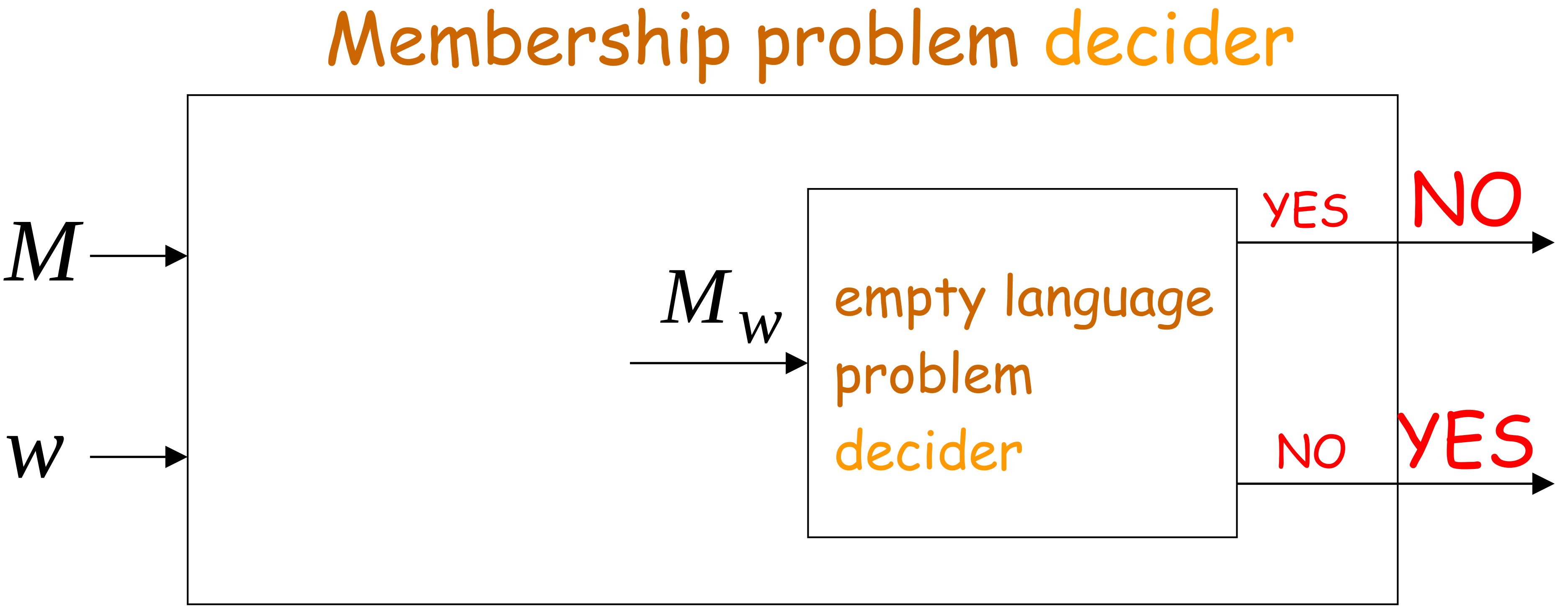
Suppose we have a decider for the empty language problem:



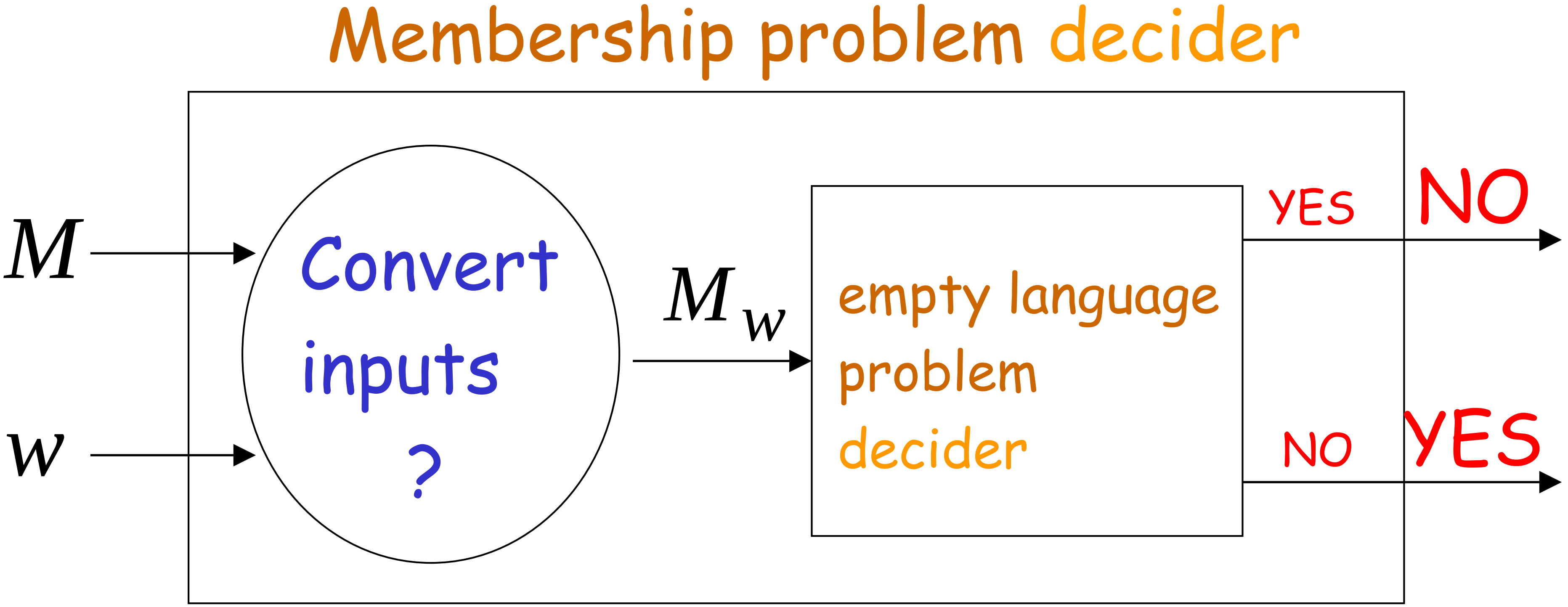
We will build the decider for the membership problem:



We want to reduce the membership problem to the empty language problem:



We need to convert one problem instance to the other problem instance



Construct machine M_w :

On arbitrary input string s

M_w executes the same as with M

When M enters a final state,
compare s with w

Accept only if $s = w$

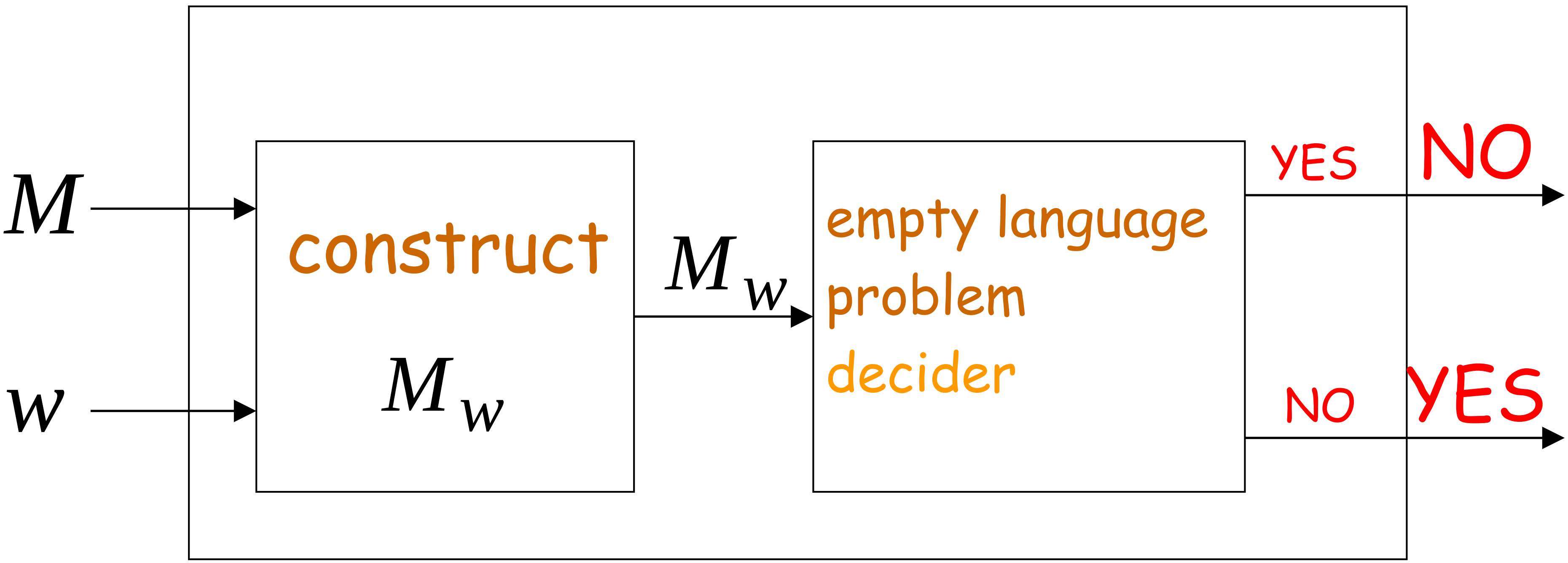
$$w \in L$$

if and only if

$L(M_w)$ is not empty

$$L(M_w) = \{w\}$$

Membership problem decider



END OF PROOF