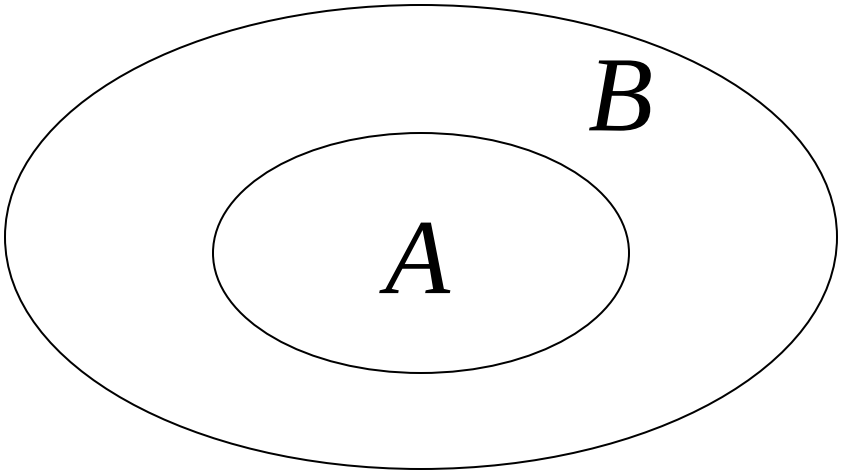
Reducibility

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

The basic approach in following slides

- Assume that TM R can decide the problem;
- Then, by using R we can construct a TM S which can decide A_{TM};
- However, we know membership problem A_{TM} is undecidable.
- Therefore, the assumption is false.

A_{TM} problem is reduced to Halting problem

Assume that TM R decides $HALT_{TM}$.
We construct TM S to decide A_{TM}

S=On input <M, w>, an encoding of a TM M and a string w:

1. Run TM R on input <M, w>.

A_{TM} problem is reduced to Halting problem

2. If R rejects , *reject*.

3. If R accepts , simulate M on w until it halts

4. If M has accepted, *accept* ; if M has rejected ,*reject*.

A_{TM} problem is reduced to E_{TM} problem

E_{TM}={<M>|M is a TM and L(M)=null}

We assume that TM R can decide E_{TM}

We don't know if TM M accepts the particular string w when R rejects E_{TM}

Therefore, we need to construct a TM M_1 which only accepts w when TM R rejects E_{TM}

A_{TM} problem is reduced to E_{TM} problem

M_1 = "On input x:

If $x \neq w$, *reject*.

If $x = w$, run M on input w and
accept if M does."

S = "On input <M, w>, an encoding of a TM M and a string w:

1. Use the description of M and w to construct the TM M_1 described above

2. Run R on input $<M_1>$

3. If R accepts, *reject*; if R rejects, *accept*."

A_{TM} is reduced to $REGULAR_{TM}$

$REGULAR_{TM}$ = {<M>| M is a TM and L(M) is a regular language}

M_2 = "On input x:

1. If x has the form 0^n1^n, *accept*.

2. If x does not have this form, run M on input w and *accept* if M accepts w."

A_{TM} is reduced to $REGULAR_{TM}$

S = "On input <M, w>, where M is a TM and w is a string:

1. Construct the TM M_2.
2. Run R on input $<M_2>$.
3. If R accepts, *accept*; if R rejects, *reject*."

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 ?

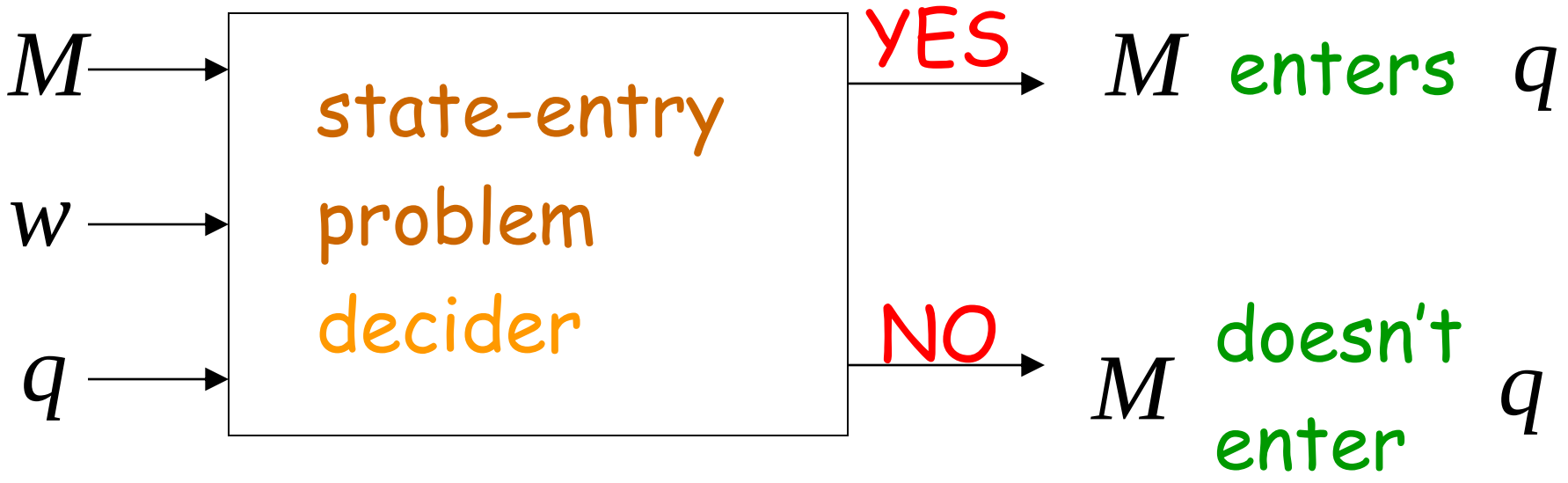
Theorem:

The state-entry problem is undecidable

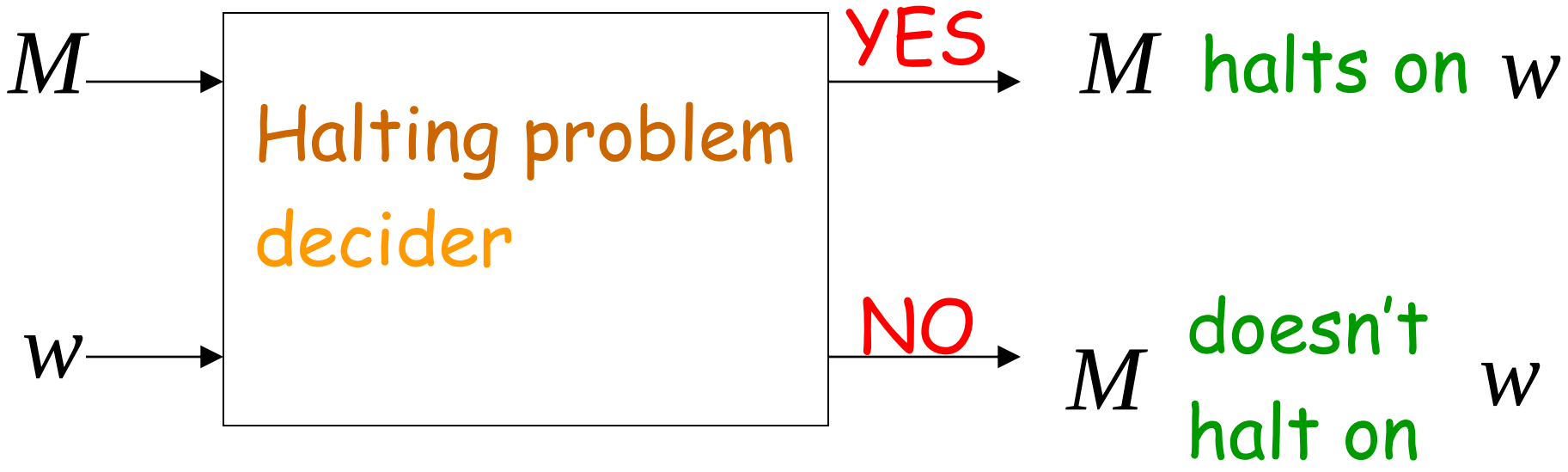
Proof: Reduce the halting problem to

the state-entry problem

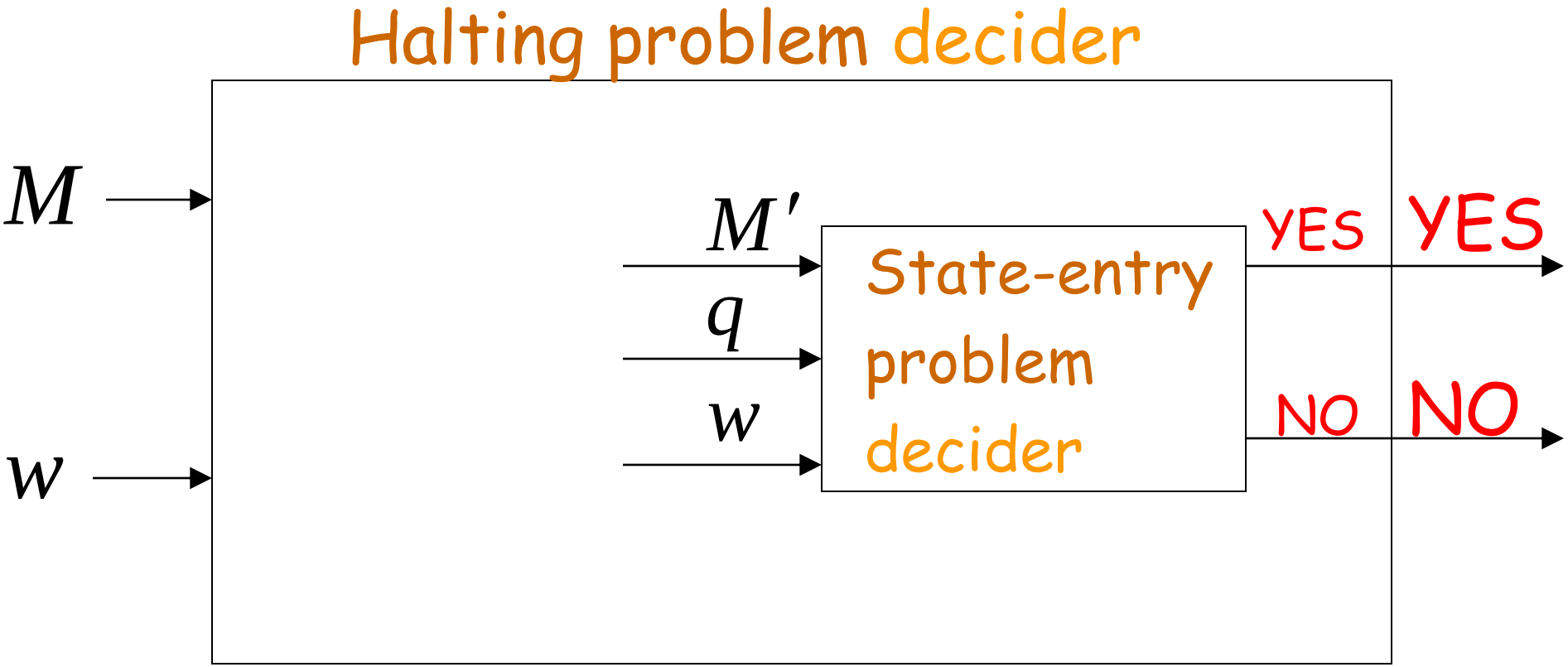
Suppose we have a Decider
for the state-entry algorithm:



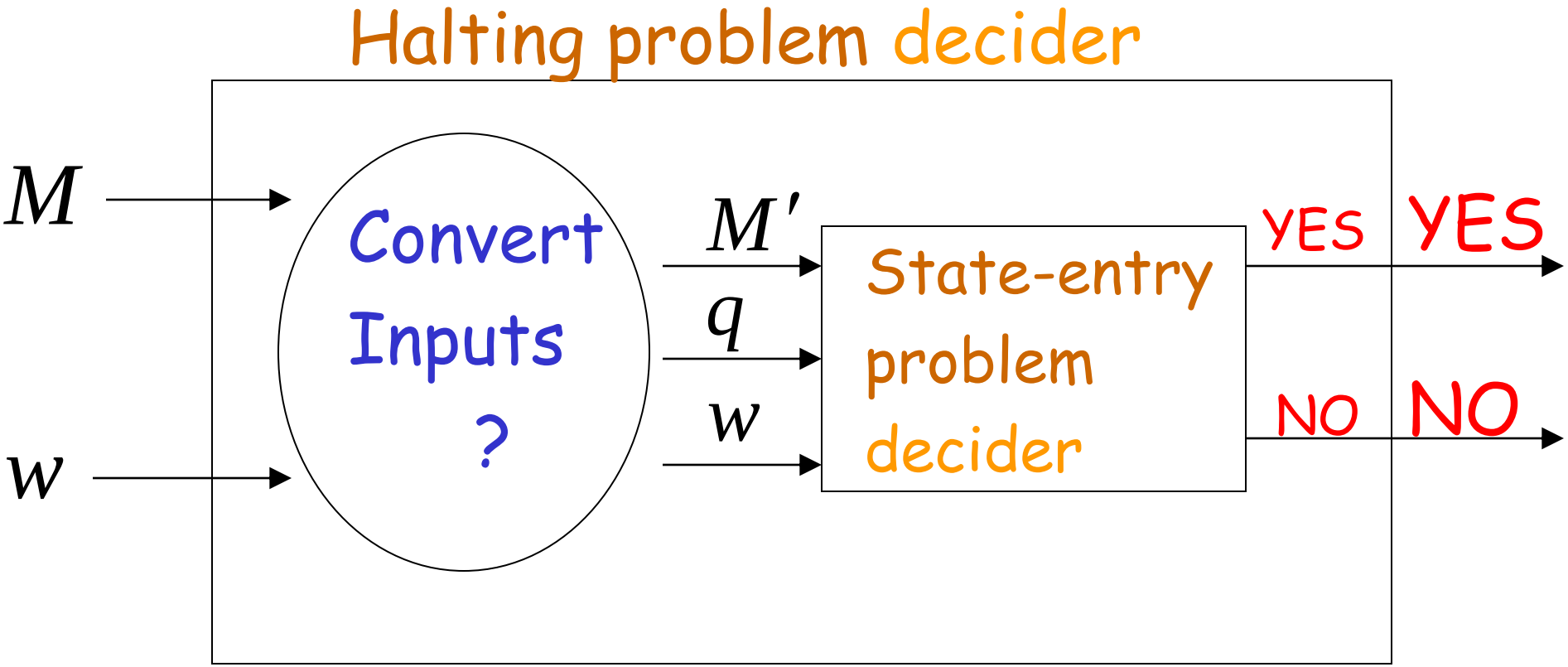
We want to build a decider for the halting problem:



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

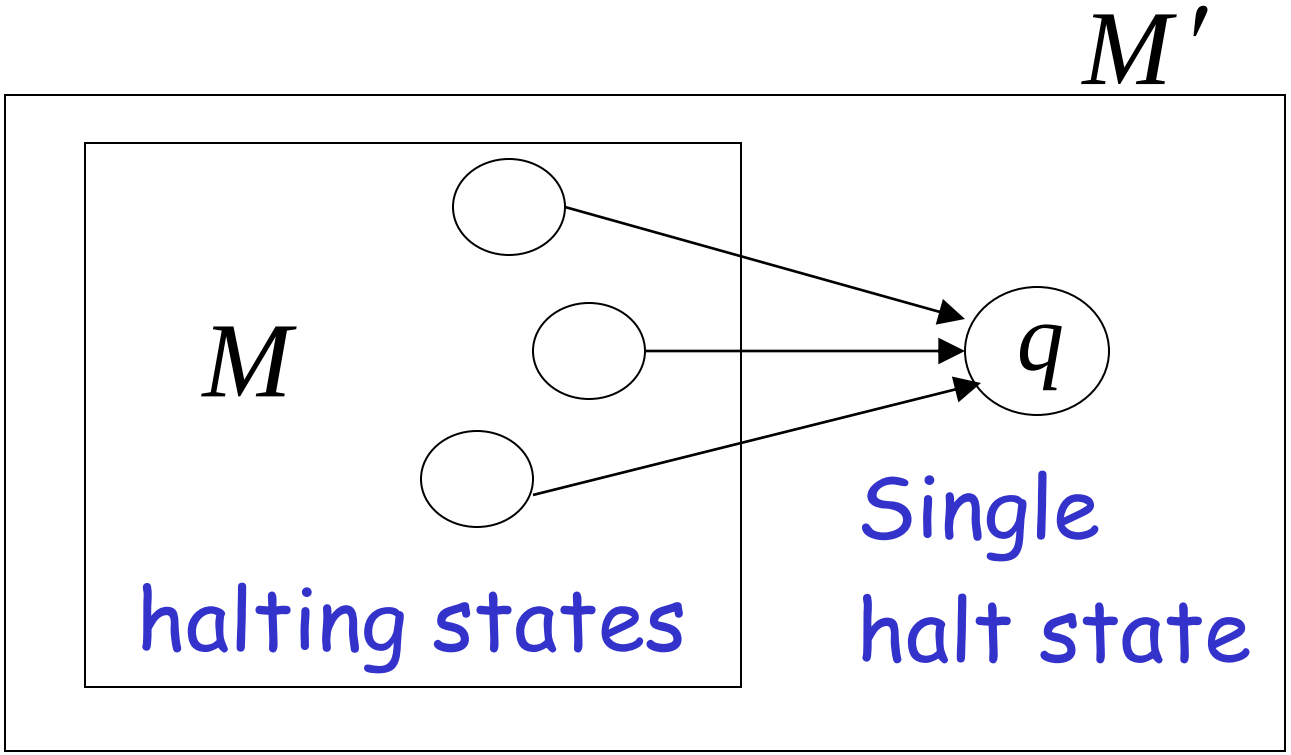


We need to convert one problem instance to the other problem instance

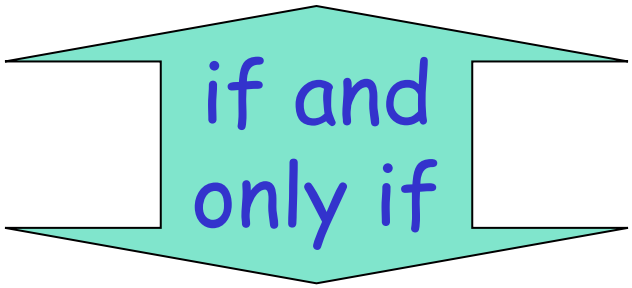


Convert M to M' :

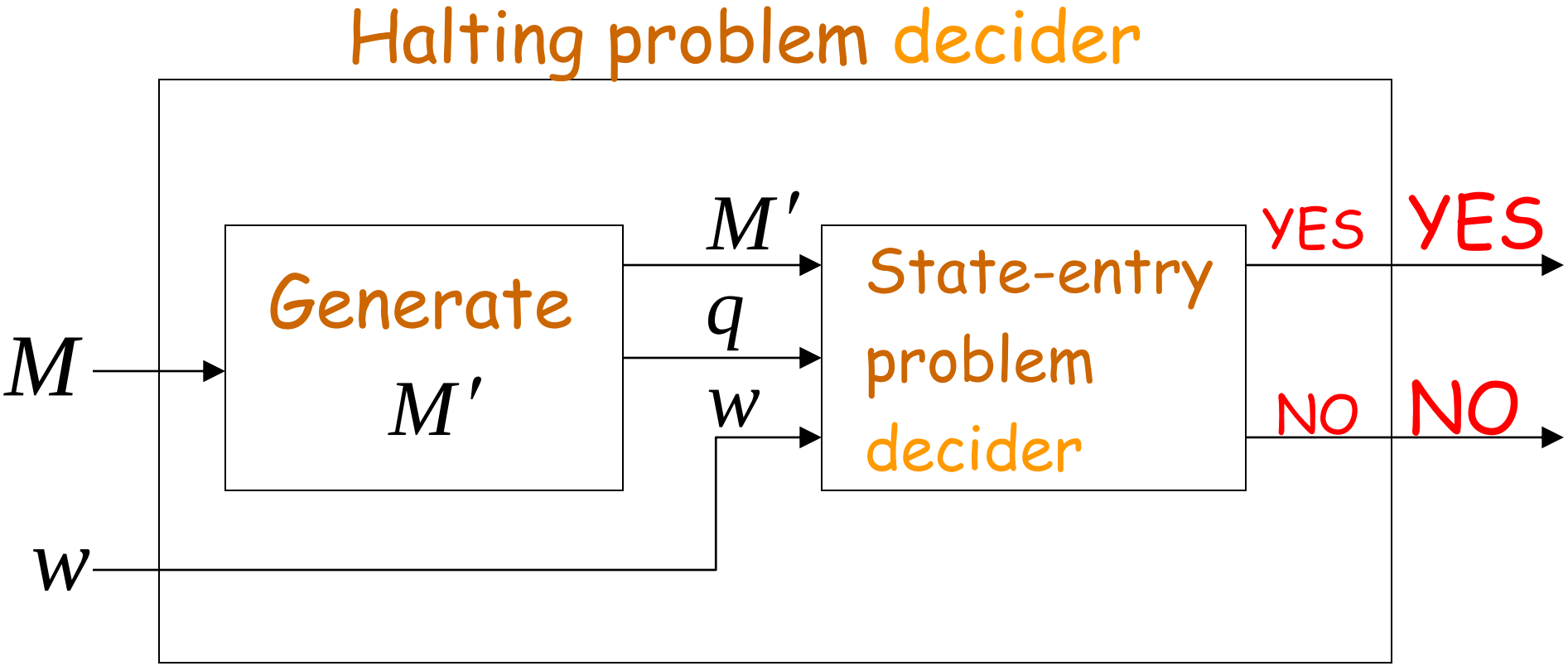
- Add new state q
- From any halting state of M add transitions to q



M halts on input w



M' halts on state q on input w


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

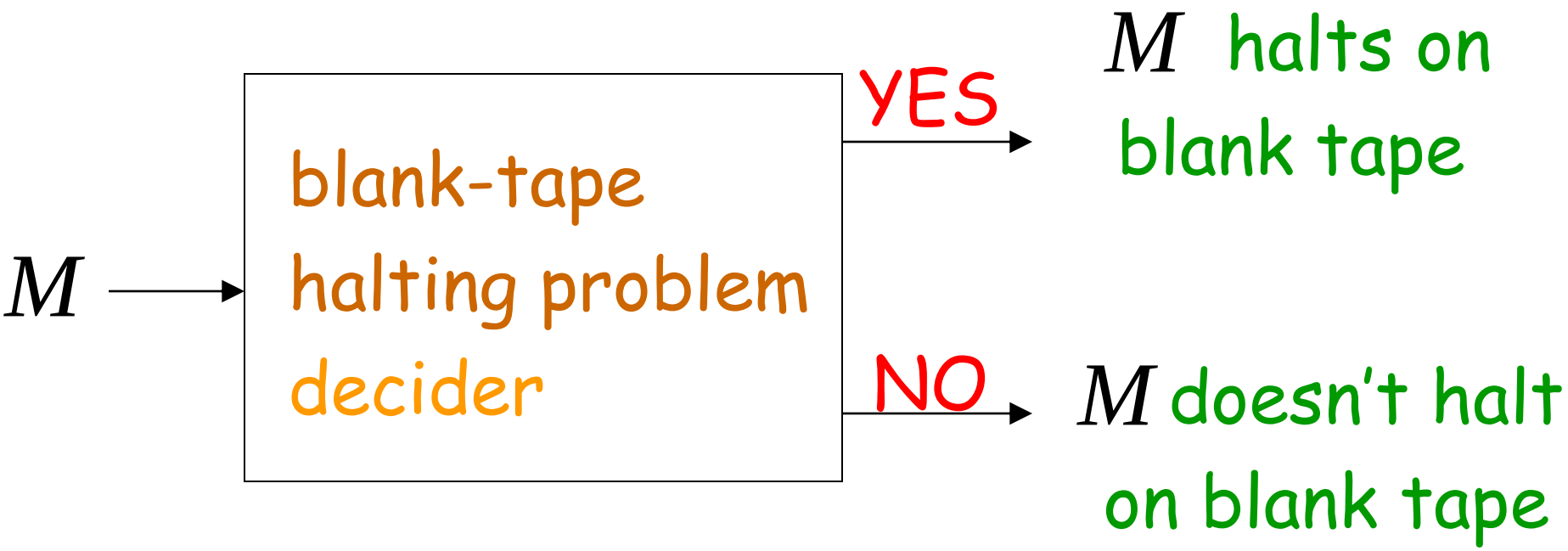
Theorem:

The blank-tape halting problem is undecidable

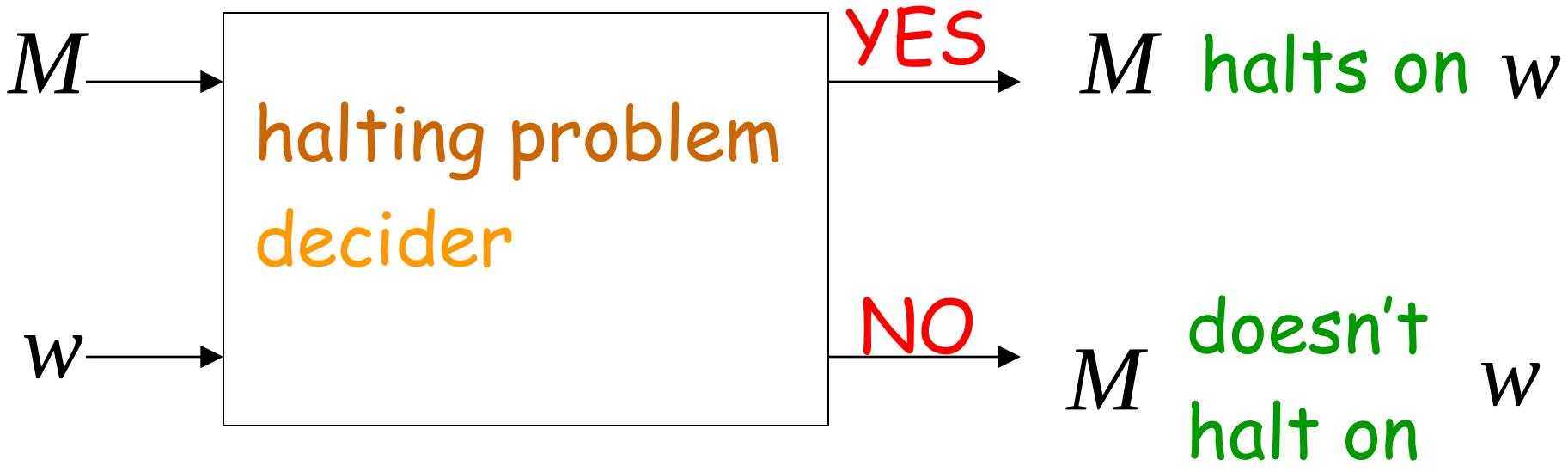
Proof: Reduce the halting problem to the

blank-tape halting problem

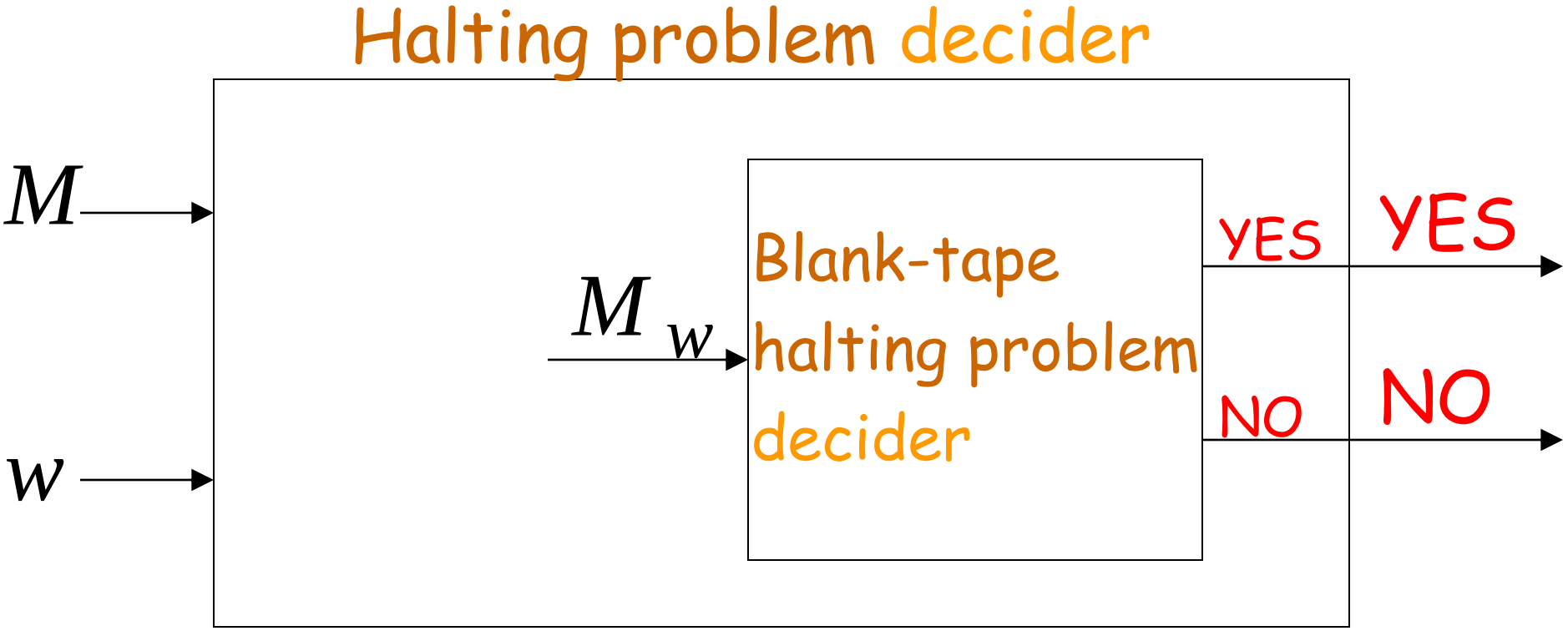
Suppose we have a decider for the blank-tape halting problem:



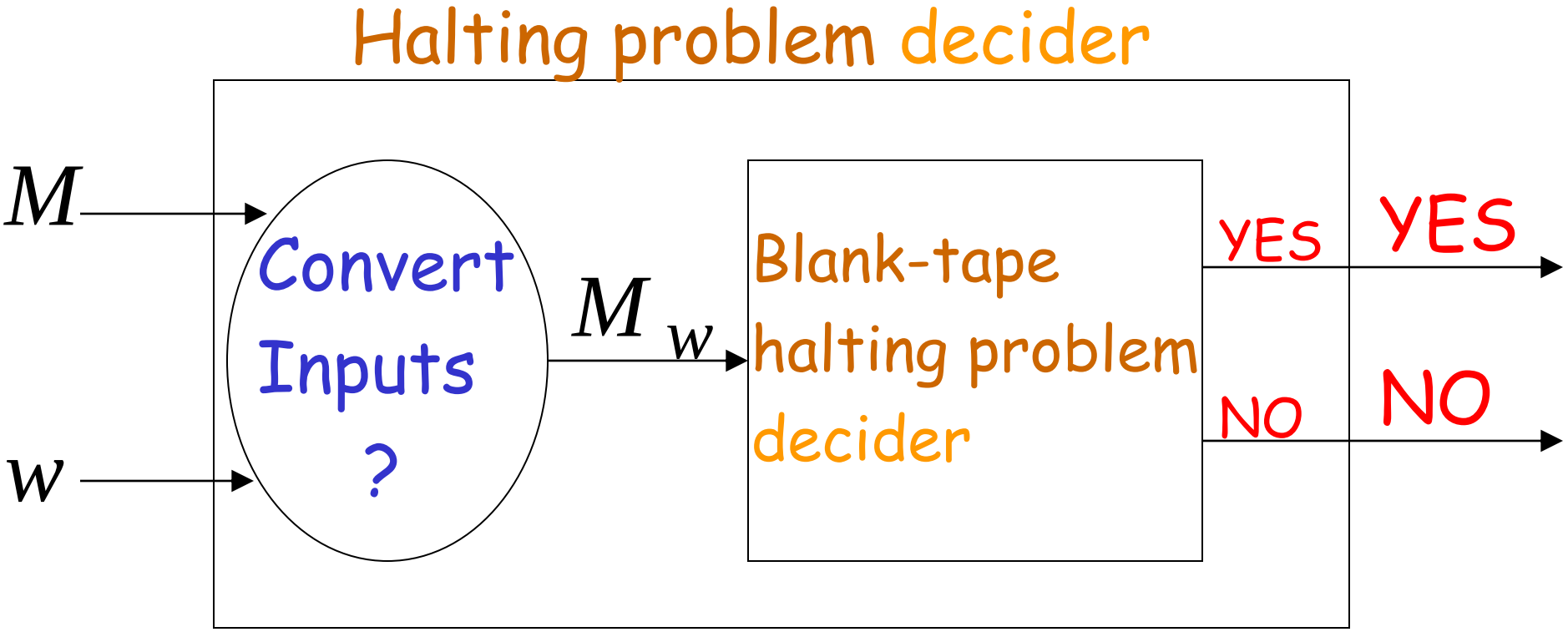
We want to build a decider for the halting problem:



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



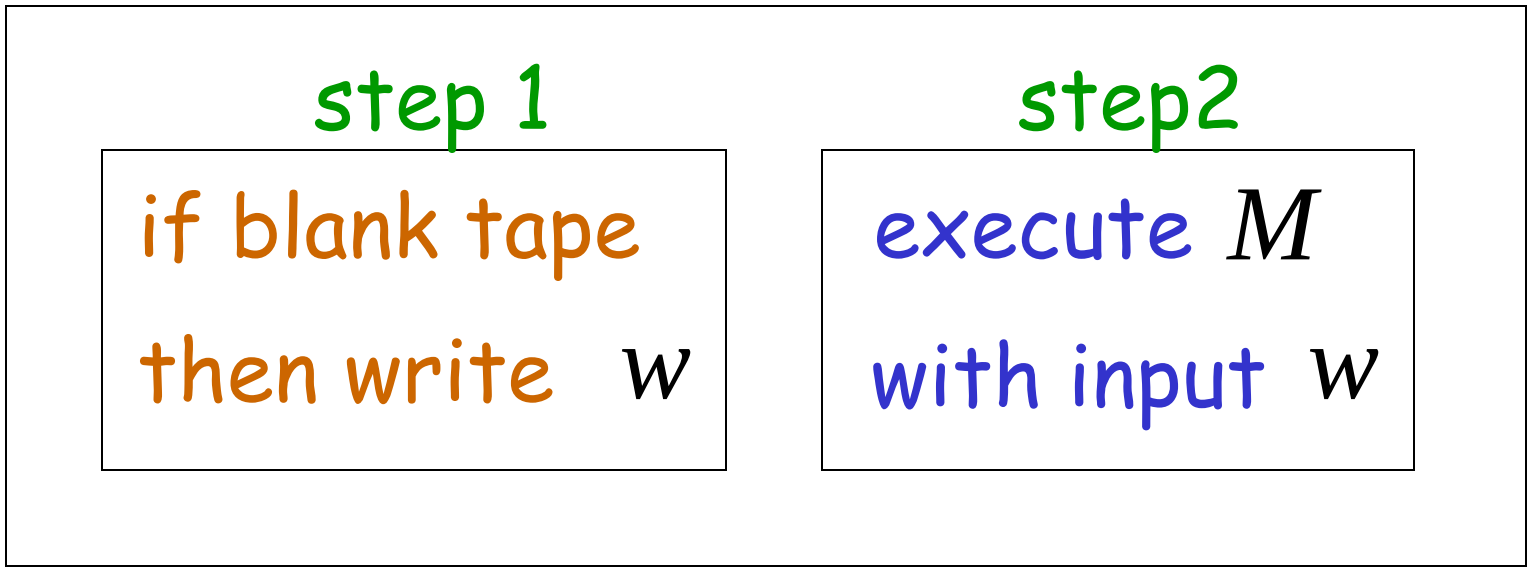
We need to convert one problem instance to the other problem instance



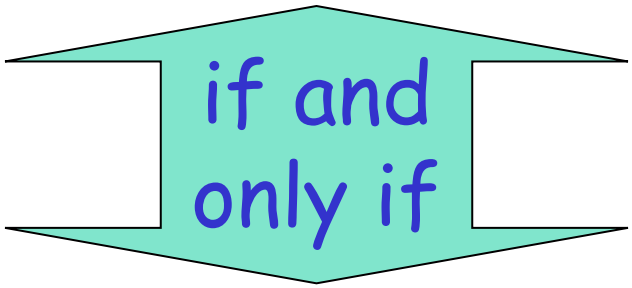
Construct a new machine M_w

- When started on blank tape, writes w
- Then continues execution like M

M_w

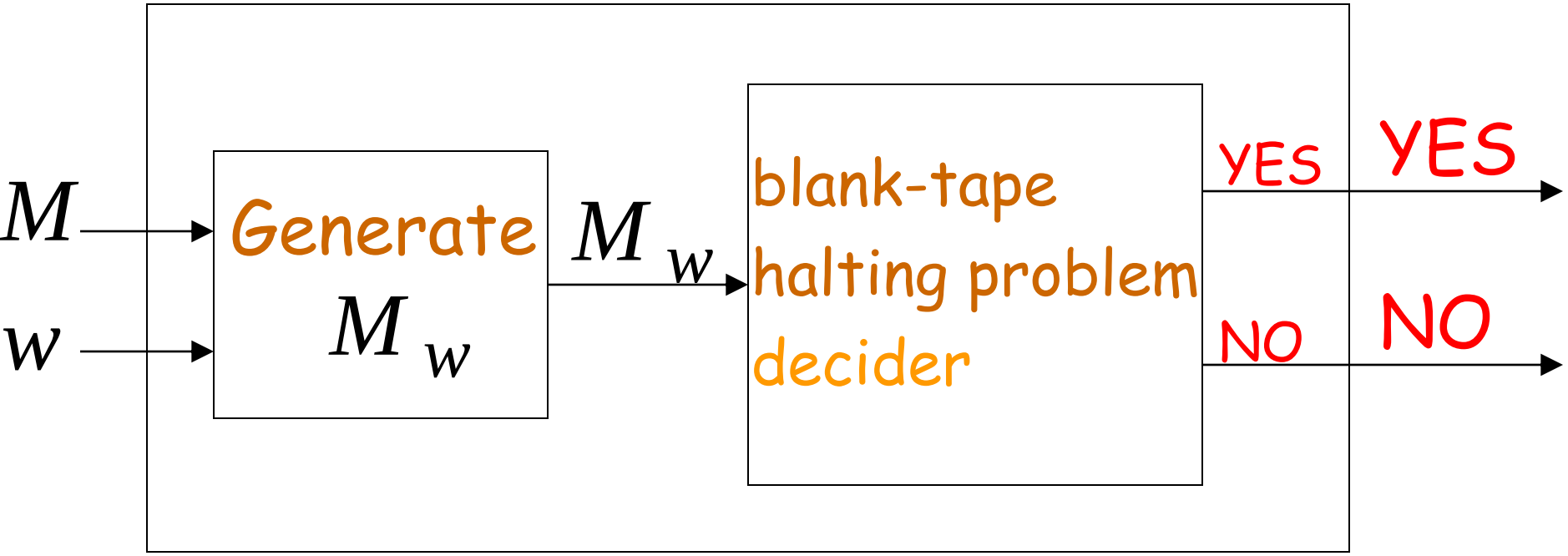


M halts on input string w



M_w halts when started with blank tape

Halting problem decider



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 ?

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:

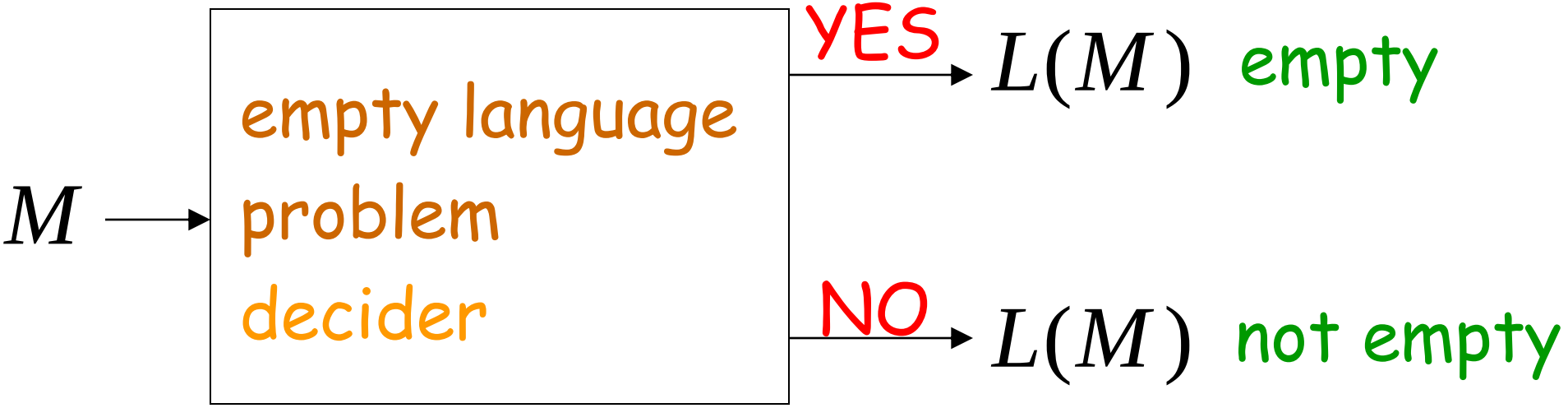
For any recursively enumerable language L it is undecidable to determine whether L is empty

Proof:

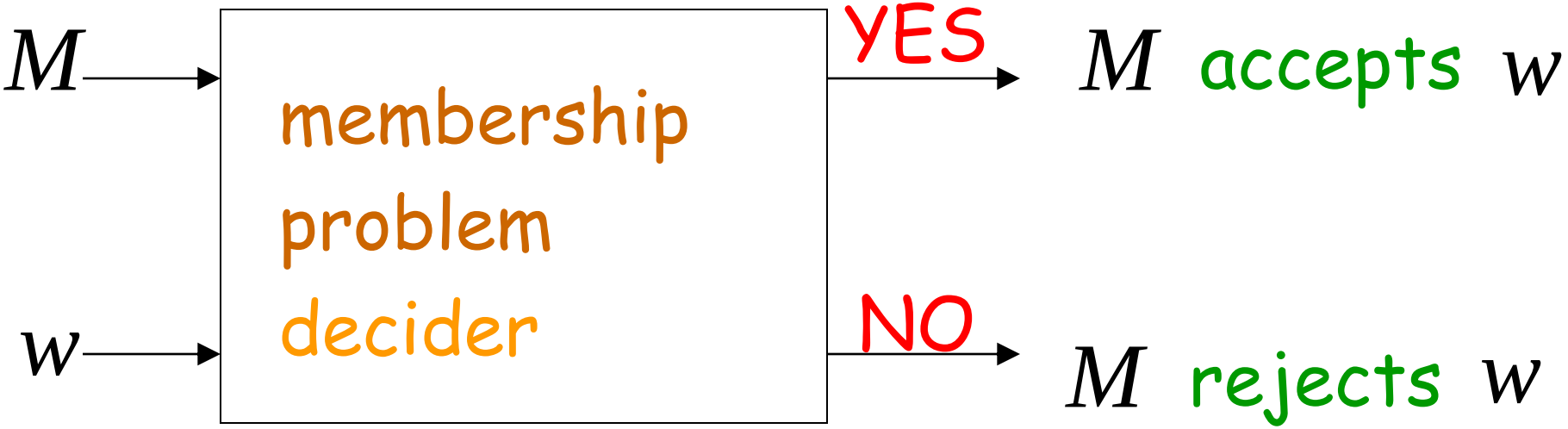
We will reduce the membership problem to this problem

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

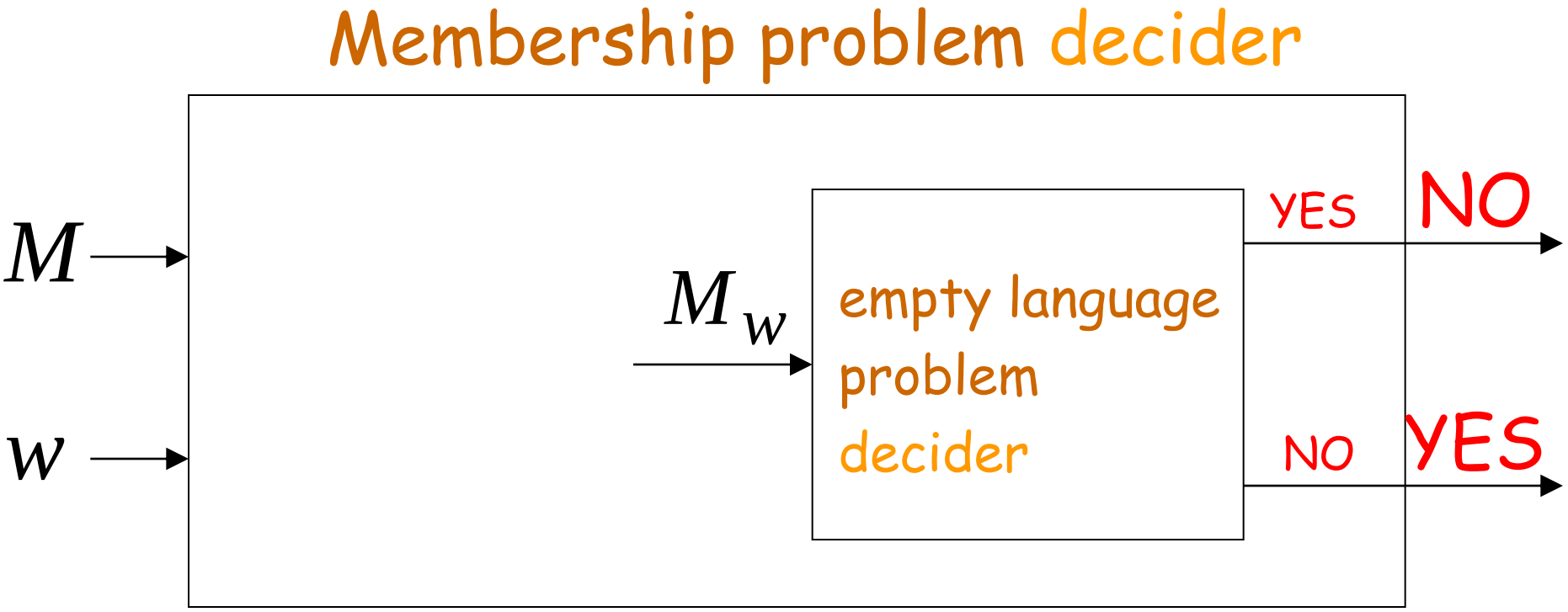
Suppose we have a decider for the empty language problem:



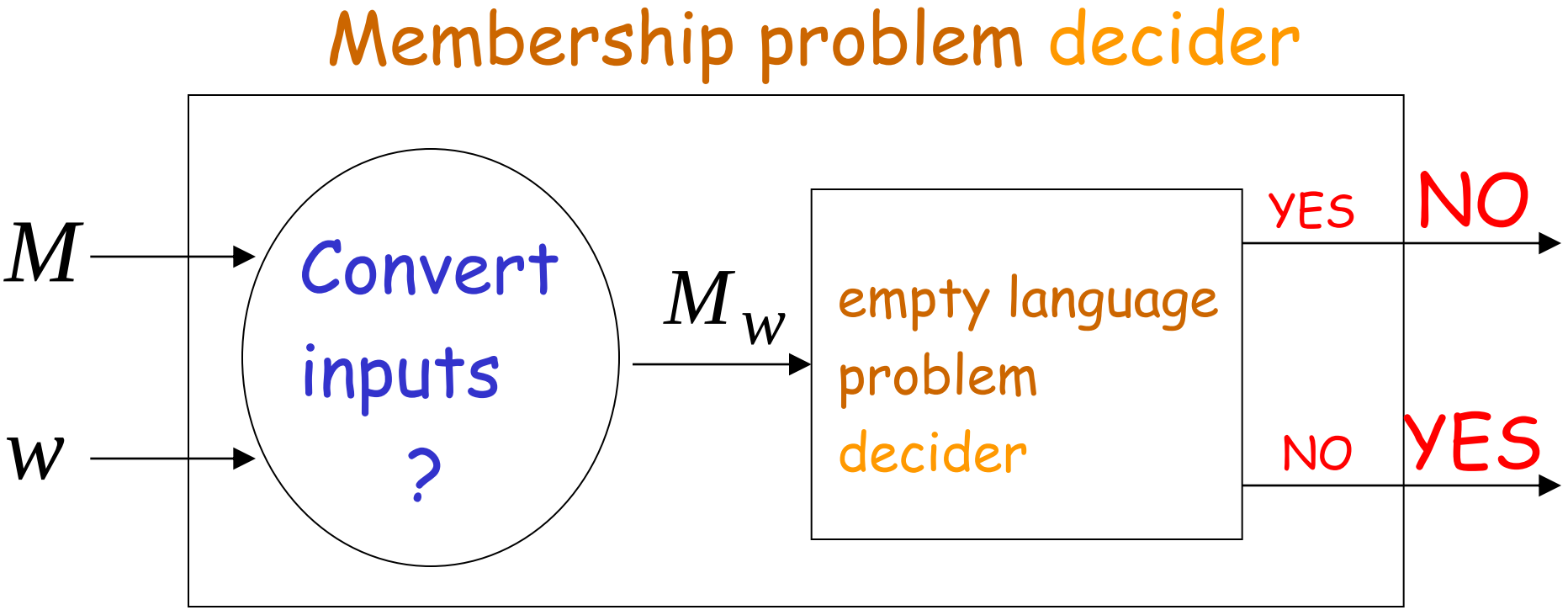
We will build the decider for the membership problem:



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:

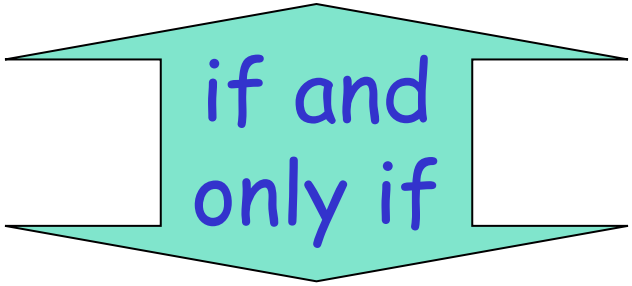
On arbitrary input string s

M_w executes the same as with M

When M enters a final state,
compare s with w

Accept only if $s = w$

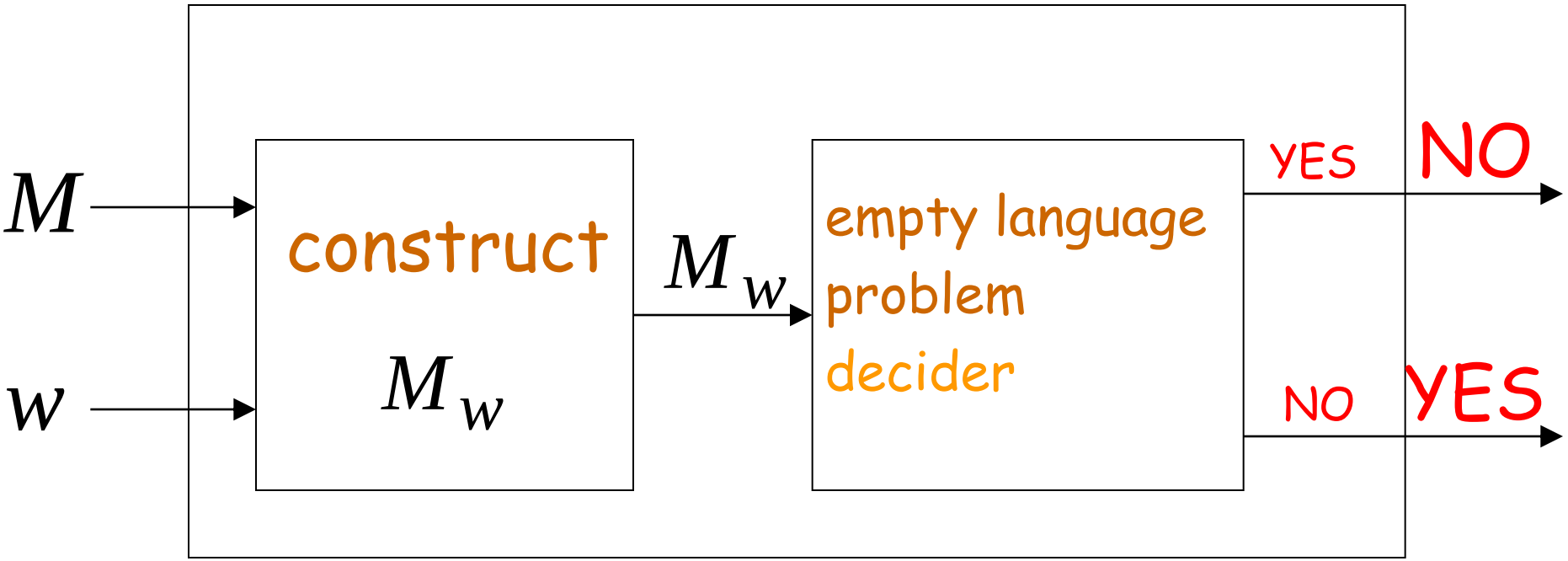
$$w \in L$$



$L(M_w)$ is not empty

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

Membership problem decider



END OF PROOF