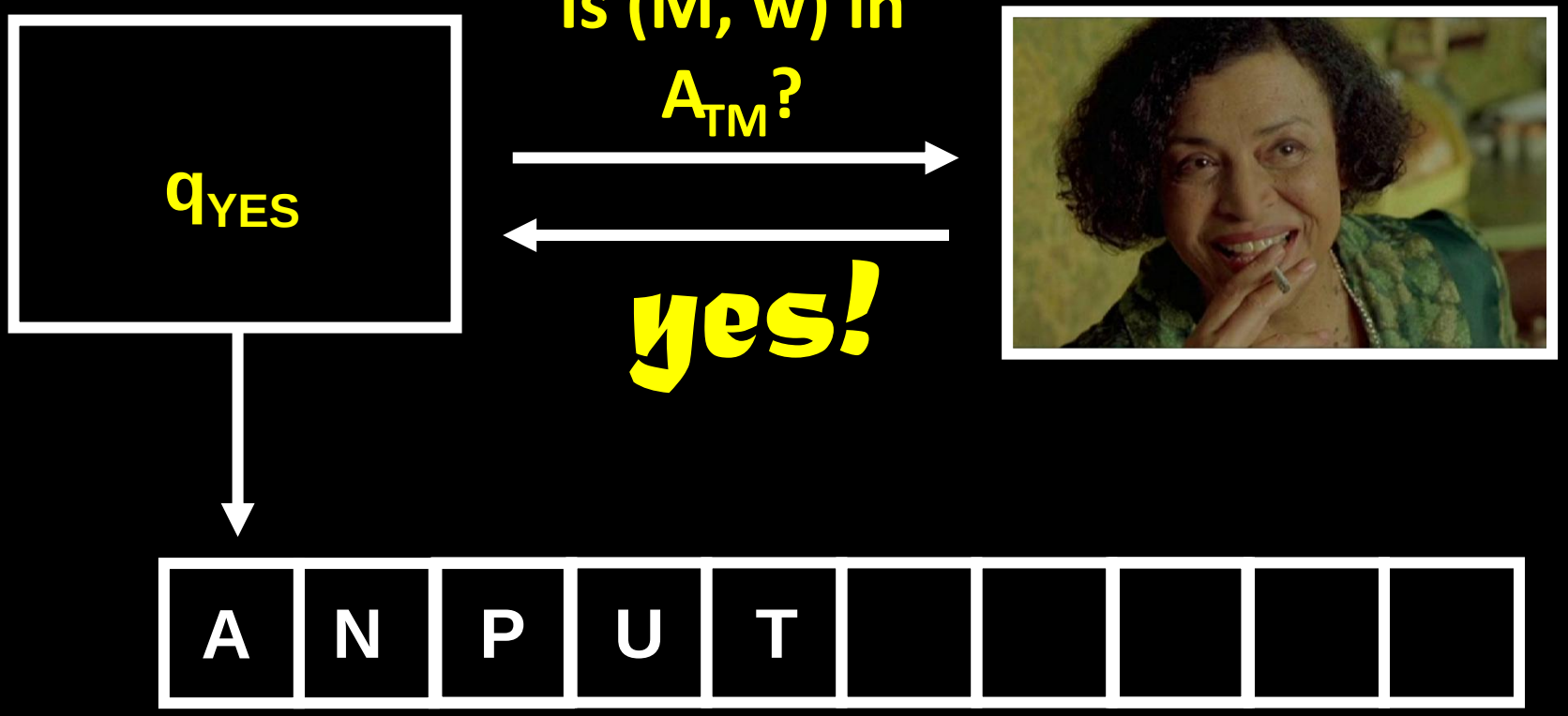
6.1400

Lecture 12
Oracles and Self-Reference:
The Recursion Theorem and
the Fixed-Point Theorem

Computability With Oracles

*We do not condone smoking. Don't do it. It's bad. Kthxbye

Oracle Turing Machines



Now leaving reality for a moment….

Oracle Turing Machines

An oracle Turing machine M is equipped with a set $B \subseteq \Gamma^*$ and a special oracle tape, on which M may ask membership queries about B

Formally, M enters a special state $q_?$ to ask a query

and the TM receives a query answer in one step

[Formally, the transition function on $q_?$ is defined in terms of the *entire oracle tape*:

State $q_?$ changes to q_{YES}
if the string y written on the oracle tape is in B,
else $q_?$ changes to q_{NO}]

This notion makes sense even if B is not decidable!

How to Think about Oracles?

Think in terms of Turing Machine pseudocode!

An **oracle Turing machine M with oracle B $\subseteq \Gamma^*$** lets you include the following kind of if-then statement:

```
"if (z in B) then <do something>
             else <do something else>"
```

where `z` is some string defined earlier in pseudocode. We define the oracle TM to that it can always check the condition `(z in B)` in one step

This notion makes sense even if B is not decidable!

Deciding one problem with another

Definition: A is decidable with B
if there is an *oracle TM M with oracle B*
that accepts strings in A and rejects strings not in A

Language A "Turing-Reduces" to B

$$A \leq_T B$$

A_{TM} is decidable with $HALT_{TM}$ **($A_{TM} \leq_T HALT_{TM}$)**

We can decide if M accepts w
using an oracle for the Halting Problem:

On input (M,w),
If (M,w) is in $HALT_{TM}$ then
run M(w) and output its answer.
else REJECT.

$HALT_{TM}$ is decidable with A_{TM} ($HALT_{TM} \leq_T A_{TM}$)

On input (M,w), decide if M halts on w as follows:

1. If (M,w) is in A_{TM} then ACCEPT

2. Else, swap the accept and reject states of M to get a machine M′. If (M′,w) is in A_{TM} then ACCEPT

3. REJECT

Limitations on Oracle TMs!

The following problem cannot be decided by any TM with an oracle for the Halting Problem:

SUPERHALT = { (M,x) | TM M, with an oracle for the Halting Problem, halts on x}

We can use the original proof by diagonalization!

Assume H (with HALT oracle) decides SUPERHALT

Define D(X) := "if H(X,X) (with HALT oracle) accepts then LOOP, else ACCEPT."

(D uses a HALT oracle to simulate H)

But D(D) halts ⇔ H(D,D) accepts ⇔ D(D) loops…

(by assumption on H) **(by def of D)**

Limitations on Oracle TMs!

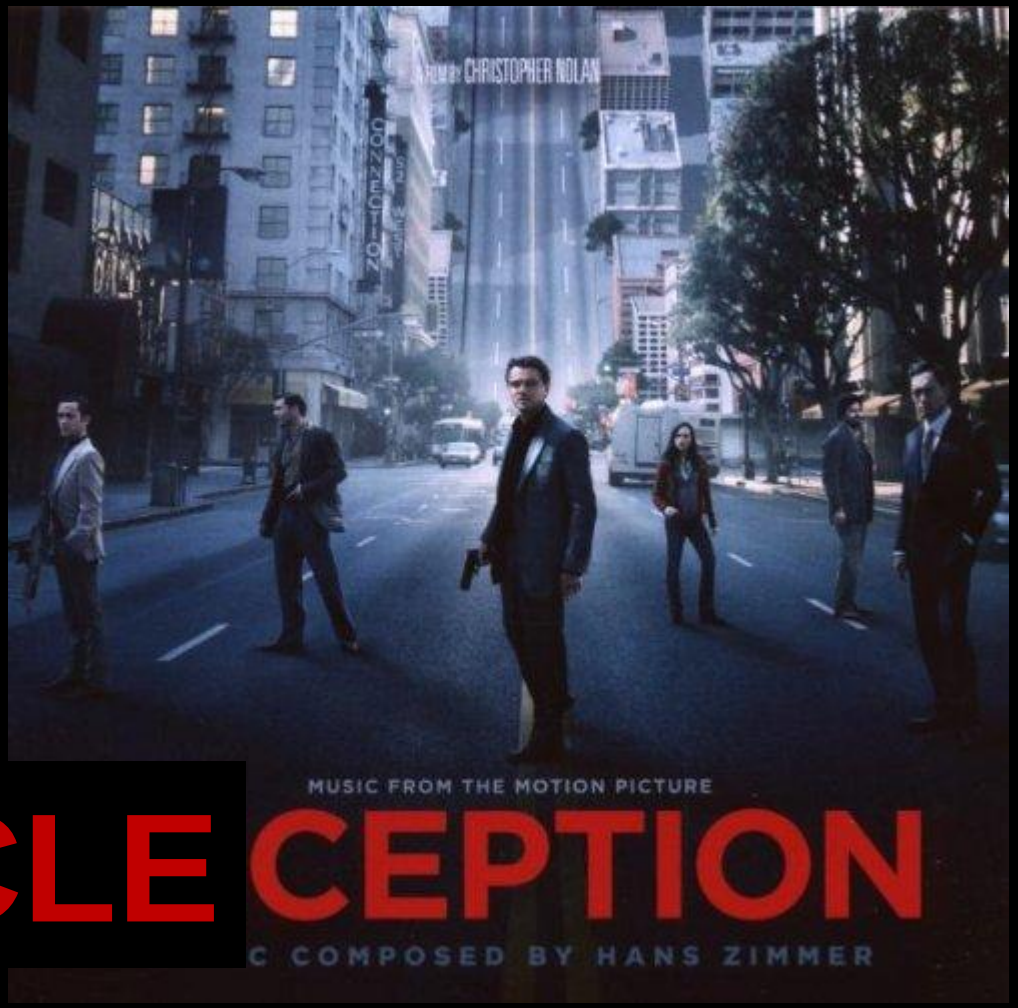
There is an *infinite hierarchy* of unsolvable problems!

Given ANY oracle A, there is always a harder problem that cannot be decided with that oracle A

$\text{SUPERHALT}^0 = \text{HALT} = \{ (M,x) \mid M \text{ halts on } x\}.$

$\text{SUPERHALT}^1 = \{ (M,x) \mid M, \text{ with an oracle for } \text{HALT}_{TM}, \text{ halts on } x\}$

$\text{SUPERHALT}^n = \{ (M,x) \mid M, \text{ with an oracle for } \text{SUPERHALT}^{n-1}, \text{ halts on } x\}$



ORACLE CEPTION

A Puzzle About Oracles

Given three instances
(M_1, w_1), (M_2, w_2), (M_3, w_3)
of the Halting Problem,

It's easy to decide all three of them,
using three oracle calls to HALT.

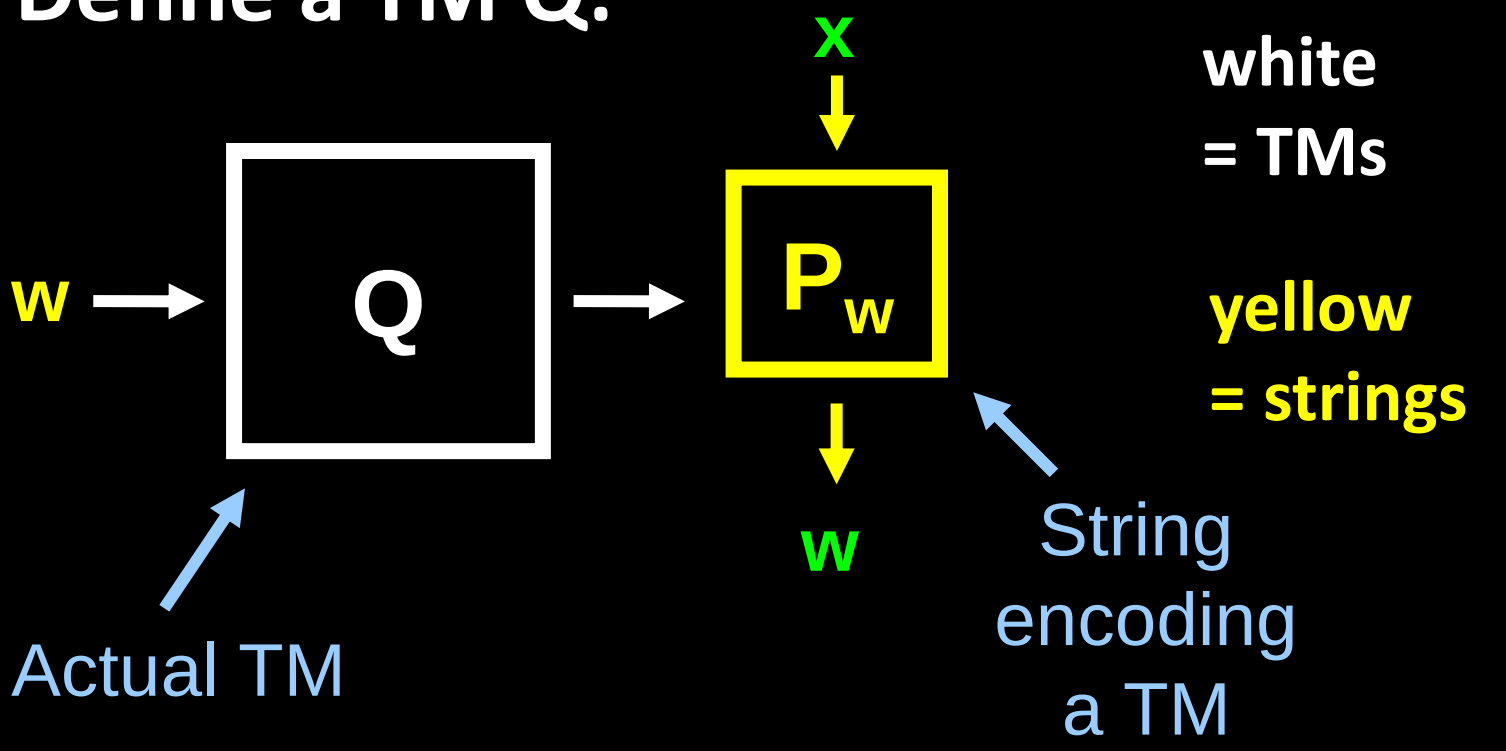
Can you decide $(M_i, w_i) \in$ HALT for all i,
with only TWO oracle calls to HALT?

Self-Reference and the Recursion Theorem

“WLOG, a program can always access its own source code”

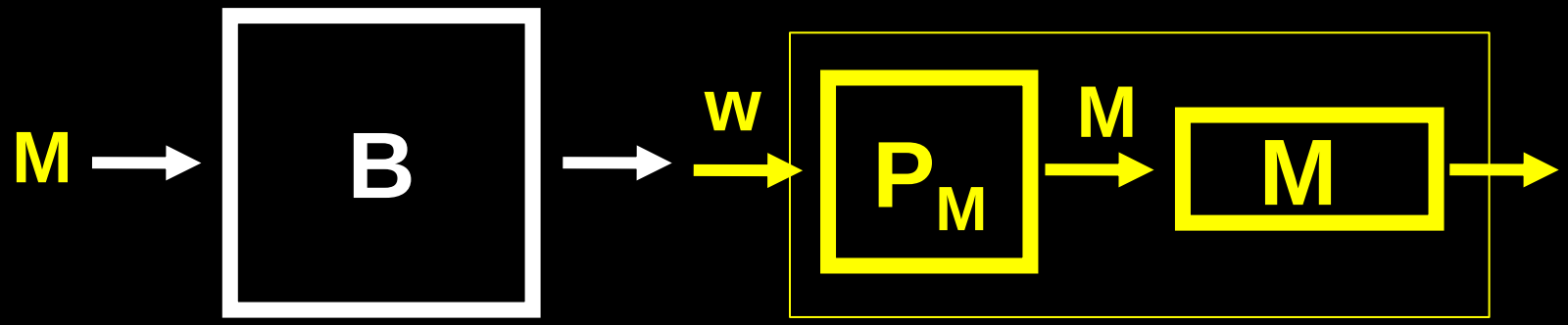
Lemma: There is a computable function $q : \Sigma^* \rightarrow \Sigma^*$ such that for every string w, q(w) is the *description* of a TM P_w that on every input, prints out w and then accepts

"Proof" Define a TM Q:

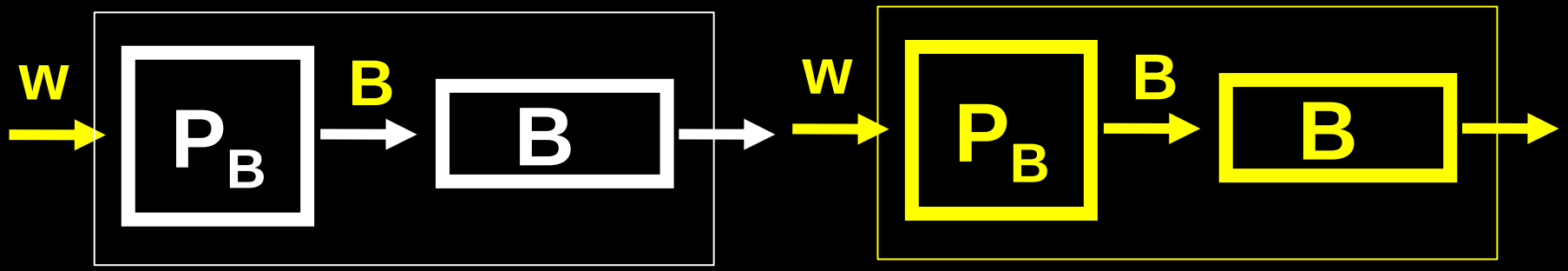


Theorem: There is a Self-Printing TM

Proof: First define a TM B which does this:



Now consider the TM that looks like this:



This is a TM that prints its own description! QED

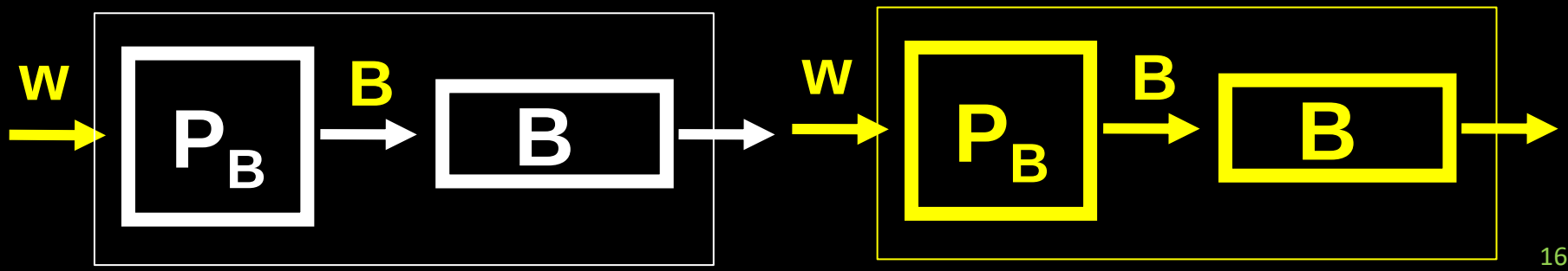
Another Way of Looking At It

Suppose in general we want to design a program that prints its own description. How?

"Print this sentence."

Print two copies of the following, the second copy in quotes: $\approx B$

"Print two copies of the following, the second copy in quotes:" $\approx P_B$



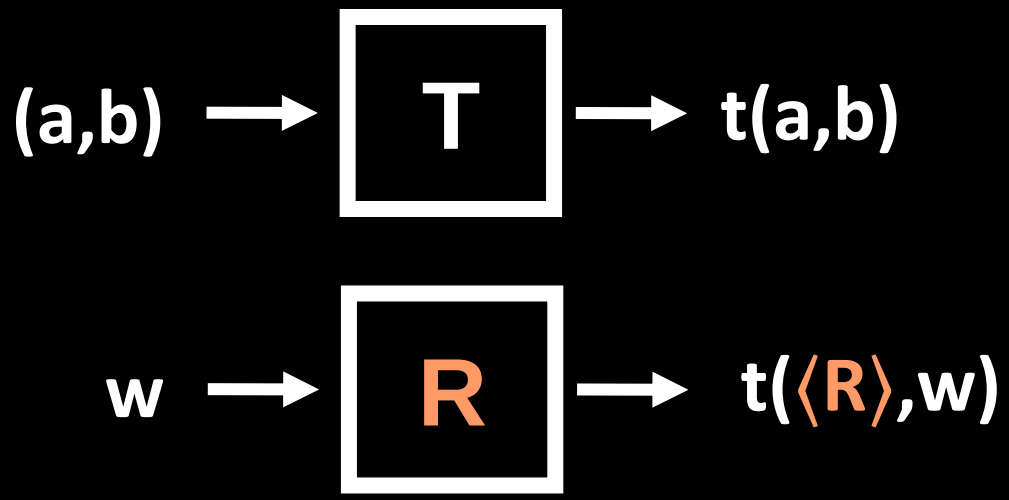
The Recursion Theorem

Theorem: For every TM T computing a function

$$t : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$$

there is a Turing machine R computing a function $R : \Sigma^* \rightarrow \Sigma^*$, such that for every string w,

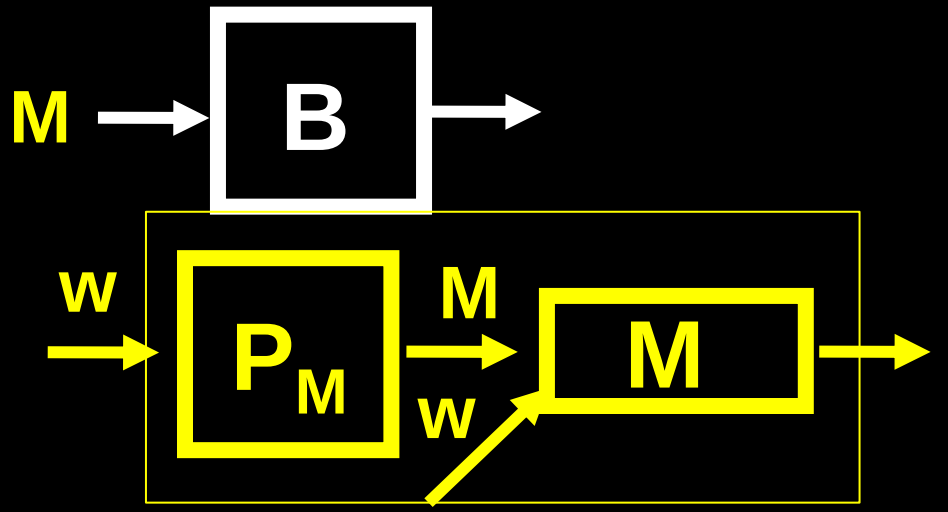
$$R(w) = t(\langle R \rangle, w)$$



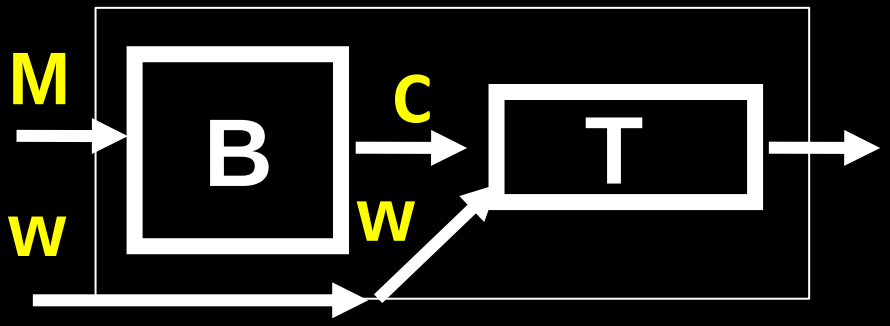
"TMs can implement recursion!"

Proof: (a,b) → T → t(a,b)

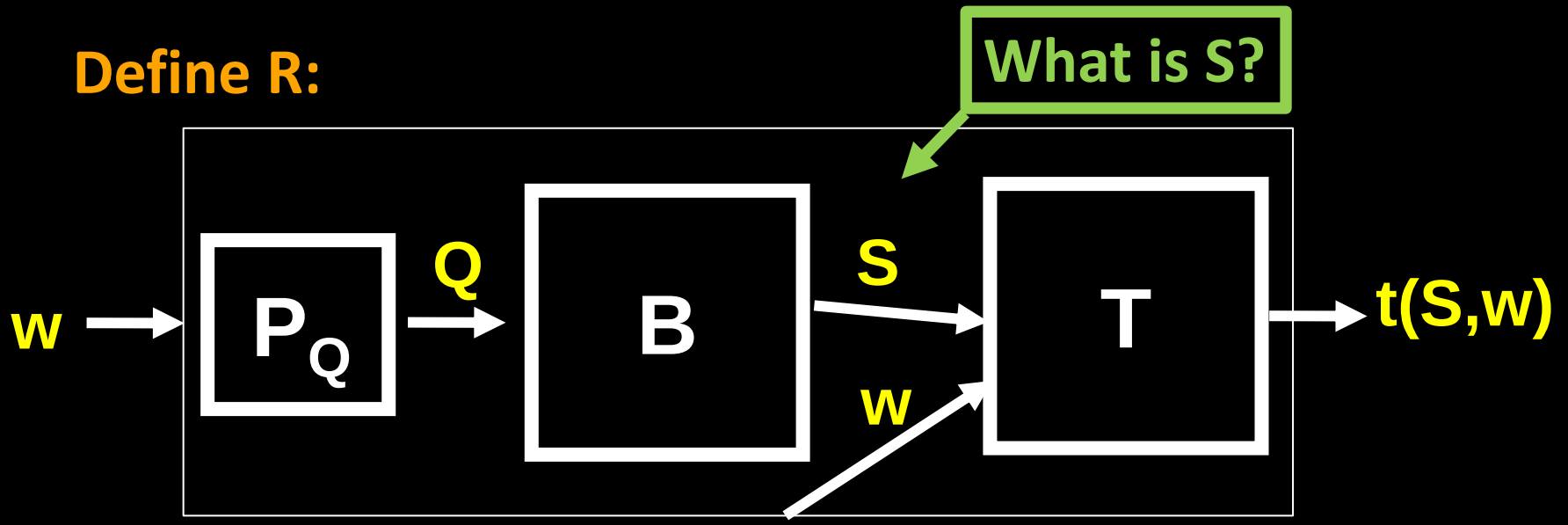
Define TM B:

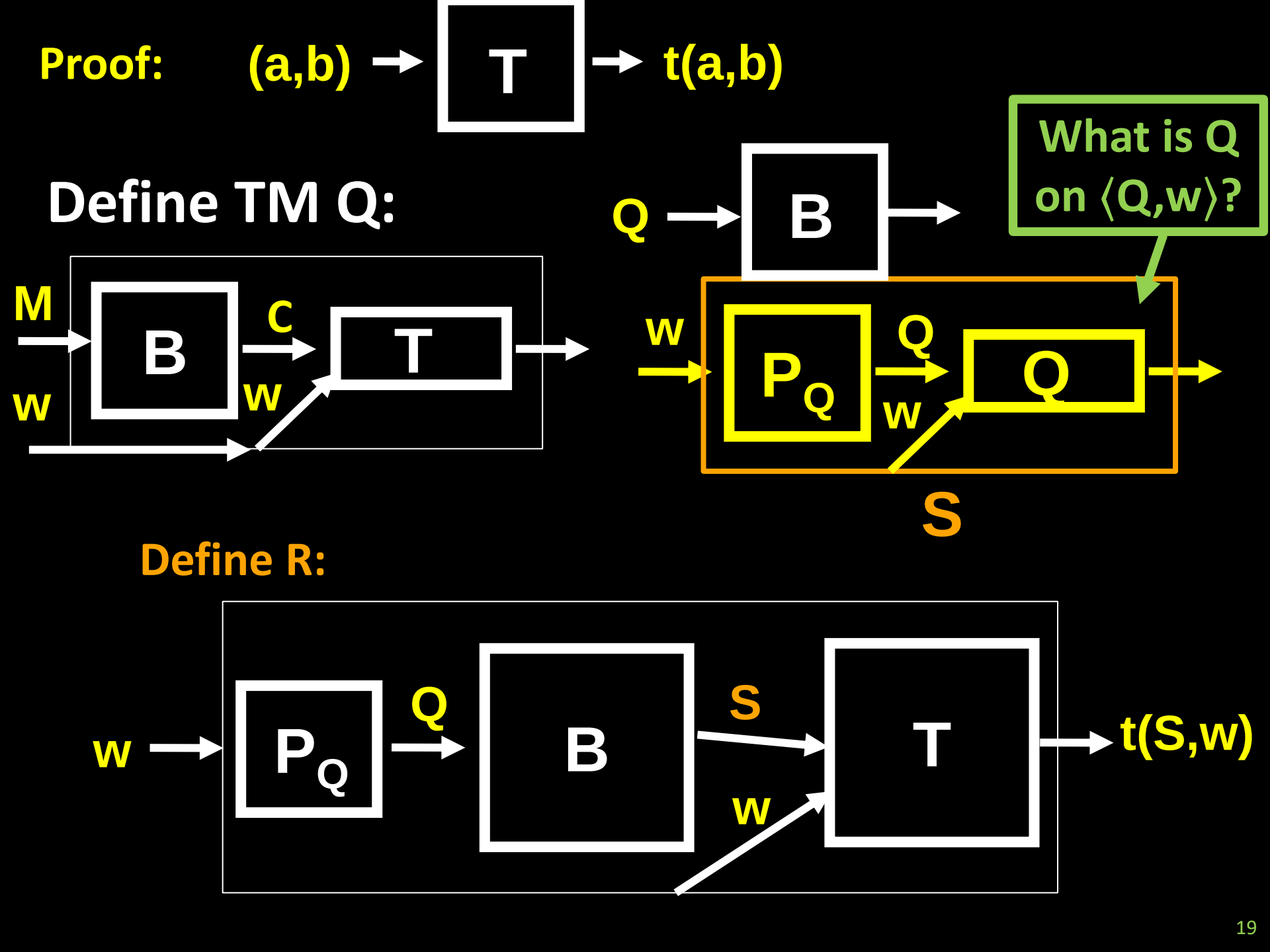


Define TM Q:

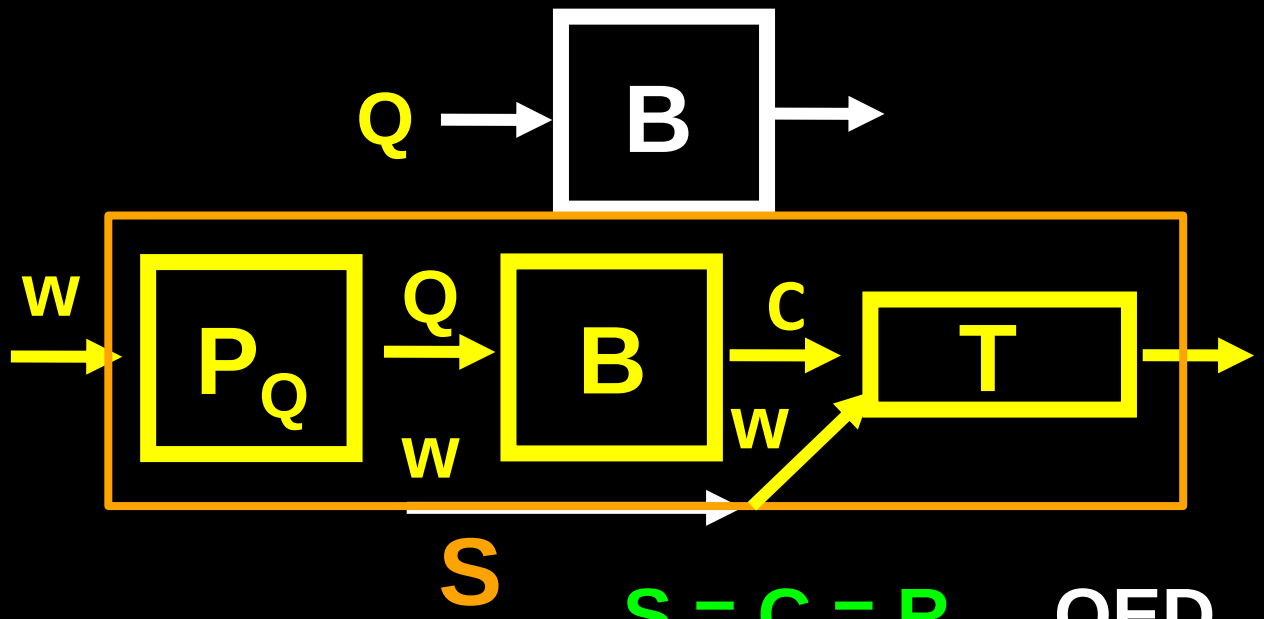


Define R:



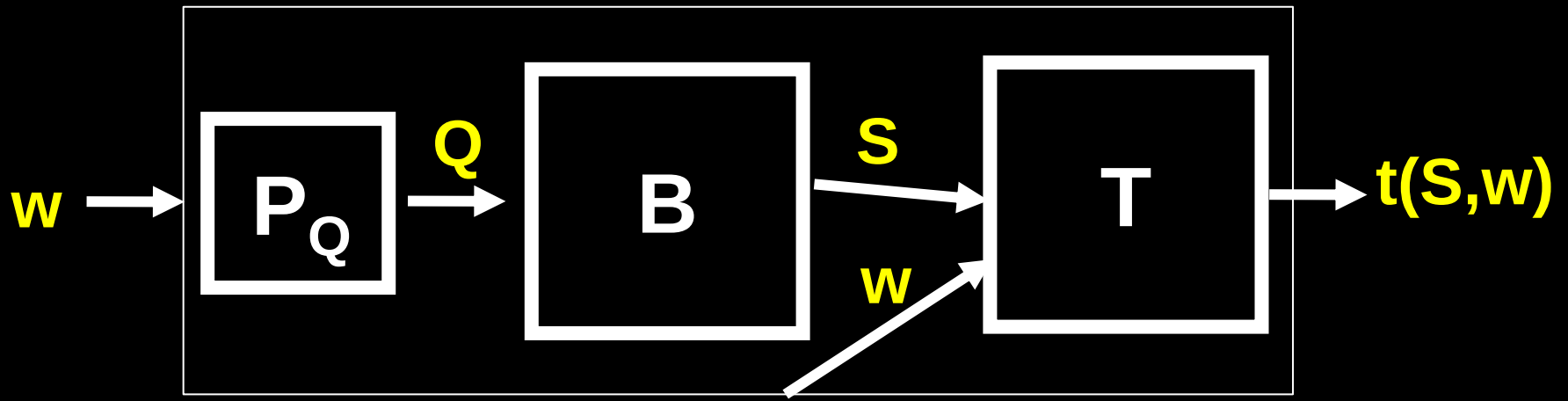

Proof:
(a,b)
T
t(a,b)
Define TM Q:
M
w
B
C
w
T
Q
B
What is Q on ⟨Q,w⟩?
w
P_Q
Q
w
Q
S
Define R:
w
P_Q
Q
B
S
w
T
t(S,w)

Proof: (a,b) → T → t(a,b)



S = C = R. QED

Define R:



$FOO_x(y)$:= Output x and halt.

BAR(⟨M⟩) := Output "N(w) = Run $FOO_{<M>}$ outputting ⟨M⟩.
Run M on (⟨M⟩, w)"

Q(⟨M⟩, w) := Run BAR(⟨M⟩) outputting ⟨N⟩.
Run T on (⟨N⟩, w)

R(w) := Run $FOO_{<Q>}$ outputting ⟨Q⟩.
Run BAR(⟨Q⟩) outputting ⟨N⟩.
Run T on (⟨N⟩, w)

Claim: ⟨N⟩ is a description of R itself!

N(w) = Run $FOO_{<Q>}$ outputting ⟨Q⟩.
Run Q on (⟨Q⟩, w)

$FOO_x(y)$:= Output x and halt.

BAR(⟨M⟩) := Output "N(w) = Run $FOO_{<M>}$ outputting ⟨M⟩.
Run M on (⟨M⟩, w)"

Q(⟨M⟩, w) := Run BAR(⟨M⟩) outputting ⟨N⟩.
Run T on (⟨N⟩, w)

R(w) := Run $FOO_{<Q>}$ outputting ⟨Q⟩.
Run BAR(⟨Q⟩) outputting ⟨N⟩.
Run T on (⟨N⟩, w)

Claim: ⟨N⟩ is a description of R itself!

N(w) = Run $FOO_{<Q>}$ outputting ⟨Q⟩.
Run BAR(⟨Q⟩) outputting ⟨N⟩.
Run T on (⟨N⟩, w)

Therefore R(w) = T(⟨R⟩, w)

For every computable t, there is a computable r such that r(w) = t(⟨R⟩,w) where

R is a Turing machine computing r

Moral: Suppose we can design a TM T of the form

"On input (x,w), do bla bla with x,
do bla bla bla with w, etc. etc."

We can always find a TM R with the *behavior*:

"On input w, do bla bla with the code of R,
do bla bla bla with w, etc. etc."

We can use the operation:

"Obtain your own description"

in Turing machine pseudocode!

Theorem: $A_{TM} = \{\langle M,w\rangle \mid M \text{ accepts } w\}$ **is undecidable**

Proof (using the recursion theorem)

Assume H decides A_{TM} Define a TM T as follows:

T(⟨M⟩,w) := Run H on ⟨M,w⟩. If H accepts, then reject.
If H rejects, then accept.

Recursion Theorem $\Rightarrow$ $\exists$ **TM B such that for all w,**
$B(w) = T(\langle B\rangle,w)$.

Now, running B on w outputs T(⟨B⟩,w), which is the *opposite answer* of H on ⟨B,w⟩. Contradiction!

A formalization of "free will" paradoxes!
No single machine can predict behavior of all others

Theorem: A_{TM} **is undecidable**

Proof (using the recursion theorem)

Assume H decides A_{TM}

Construct machine B such that on input w:

1. **Obtains its own description B**
2. **Runs H on** $\langle B,w \rangle$ **and flips the output**

Running B on any input w always does the *opposite* of what H on $\langle B,w \rangle$ **says B would do! Contradiction!**

Turing Machine Minimization

MIN = {$\langle M \rangle$ | M is a minimal-state TM over Γ={0,1,□}}

Theorem: MIN is undecidable

Proof: Suppose we could recognize MIN with TM M'

$M(x)$:= Obtain $\langle M \rangle$, the description of M.

For k = 1,2,3,…

Run M' on the first k TMs $M_1, \ldots M_k$ for k steps,

Until M' accepts some M_i with *more* states than M

Output M_i on x.

Why does M_i exist?

We have: 1. $L(M) = L(M_i)$ [by construction]

2. M has *fewer* states than M_i

3. M_i is minimal [by definition of MIN]

CONTRADICTION!

The Fixed-Point Theorem

Theorem: Let $t : \Sigma^* \rightarrow \Sigma^*$ be computable. There is a TM F such that $t(\langle F \rangle)$ outputs the description of a TM G such that $L(F)=L(G)$.

Proof: Here is pseudocode for the TM F:

On input w:

1. Obtain $\langle F \rangle$, the description of F
2. Run $t(\langle F \rangle)$ and get an output $\langle G \rangle$. Interpret $\langle G \rangle$ as the description of a TM
3. Accept w if and only if G accepts w

Problem:

REVERSE = { ⟨M⟩ | M is a TM with the property:
for all w, M(w) accepts ⇔ M(w^R) accepts}.

Decidable or not?

REVERSE is undecidable.

Rice's Theorem: *Program Analysis is Hard*

Let P : {Turing Machines} → {0,1}.
(Think of 0=false, 1=true) **Suppose P satisfies:**

1. **(Nontrivial)** There are TMs M_1 and M_0 where $P(M_1) = 1$ and $P(M_0) = 0$

2. **(Semantic)** For all TMs M and M', If L(M) = L(M') then $P(M) = P(M')$

Then, {⟨M⟩ | P(M) = 1} is undecidable.
In other words, function P is undecidable.

A Huge Hammer for Undecidability!

Some Examples and Non-Examples

Semantic Properties *P(M)*

- M accepts 0
- L(M) is empty
- L(M) = Σ^*
- M accepts 61400 strings
- for all w, M(w) accepts ⇔ M(w^R) accepts

Not Semantic!

- M halts and rejects 0
- M has 61400 states
- M halts on all inputs

Rice says: {⟨M⟩ | *P*(M) = 1} is undecidable

***P* is not semantic:**
There are M and M' such that L(M) = L(M') but P(M) ≠ P(M')

Fixed Points and Rice's Theorem

Rice's Theorem: Every nontrivial and semantic property P over Turing machines is undecidable.

Proof: Assume we can decide P. Let $P(\mathrm{M}_0) = 0$ and $P(\mathrm{M}_1) = 1$. The following is computable:

t($\langle M \rangle$) := If ($P(M) = 1$) then output $\langle \mathrm{M}_0 \rangle$
else output $\langle \mathrm{M}_1 \rangle$

For all TMs M, the function t($\langle M \rangle$) always outputs some $\langle N \rangle$ such that L(N) ≠ L(M), by the semantic condition.

This contradicts the fixed-point theorem!