15-453

FORMAL LANGUAGES, AUTOMATA AND COMPUTABILITY

Problem 1 **DECIDABLE ?**

{ (M, w) | M is a TM that on input w, tries to move its head past the left end of the tape }

Problem 2 **DECIDABLE ?**

{ (M, w) | M is a TM that on input w, moves its head left at least once, at some point }

Problem 1 **UNDECIDABLE**

{ (M, w) | M is a TM that on input w, tries to move its head past the left end of the tape }

Proof: Assume, for a contradiction, that TM T decides the language

We use T to decide A_{TM}

On input (M,w), make a new TM N that on input w marks the leftmost tape cell and then simulates M(w) (as tho the leftmost cell was not there). If M tries to move to the marked cell, N moves the head back to the right. If M accepts, N tries to moves its head past the left end of the tape.

Run T on input (N,w)

Problem 2 **DECIDABLE**

{ (M, w) | M is a TM that on input w, moves its head left at least once, at some point}

On input (M,w), run the machine for $|Q_M| + |w| + 1$ steps:

Accept **If M's head moved left at all**
Reject **Otherwise**

(Why does this work??)

RICE'S THEOREM, THE RECURSION THEOREM, AND THE FIXED-POINT THEOREM

THURSDAY FEB 27

FIN_{TM} = { M | M is a TM and L(M) is finite}

Is **FIN_{TM}** Decidable?

FIN_{TM} = { M | M is a TM and L(M) is finite}

Is FIN_{TM} Decidable?

Note Properties of this language:

- FIN_{TM} is a language of Turing Machines
- If $M_1 \equiv M_2$ (ie $L(M_1) = L(M_2)$), then either both M_1 and M_2 are in FIN_{TM} or both are not.
- There are TMs M_1 and M_2, such that $M_1 \in FIN_{TM}$ and $M_2 \notin FIN_{TM}$

RICE'S THEOREM

Let L be a language over Turing machines.
Assume that L satisfies the following properties:

1. **For TMs M_1 and M_2, if $M_1 \equiv M_2$ then**
$$M_1 \in L \Leftrightarrow M_2 \in L$$

2. **There are TMs M_1 and M_2, such that $M_1 \in L$ and $M_2 \notin L$**

Then L is undecidable

EXTREMELY POWERFUL!

RICE'S THEOREM

Let L be a language over Turing machines.
Assume that L satisfies the following properties:

1. **For TMs M_1 and M_2, if $M_1 \equiv M_2$ then**
$$M_1 \in L \Leftrightarrow M_2 \in L$$

2. **There are TMs M_1 and M_2, such that $M_1 \in L$ and $M_2 \notin L$**

Then L is undecidable

FIN_{TM} = { M | M is a TM and L(M) is finite}

RICE'S THEOREM

Let L be a language over Turing machines.
Assume that L satisfies the following properties:

1. **For TMs M_1 and M_2, if $M_1 \equiv M_2$ then**
$$M_1 \in L \Leftrightarrow M_2 \in L$$

2. **There are TMs M_1 and M_2, such that $M_1 \in L$ and $M_2 \notin L$**

Then L is undecidable

$$E_{TM} = \{ M \mid M \text{ is a TM and } L(M) = \emptyset \}$$

$$REG_{TM} = \{ M \mid M \text{ is a TM and } L(M) \text{ is regular} \}$$

RICE'S THEOREM

Let L be a language over Turing machines.
Assume that L satisfies the following properties:

1. **For TMs M_1 and M_2, if $M_1 \equiv M_2$ then**

$$M_1 \in L \Leftrightarrow M_2 \in L$$

2. **There are TMs M_1 and M_2,**
such that $M_1 \in L$ and $M_2 \notin L$

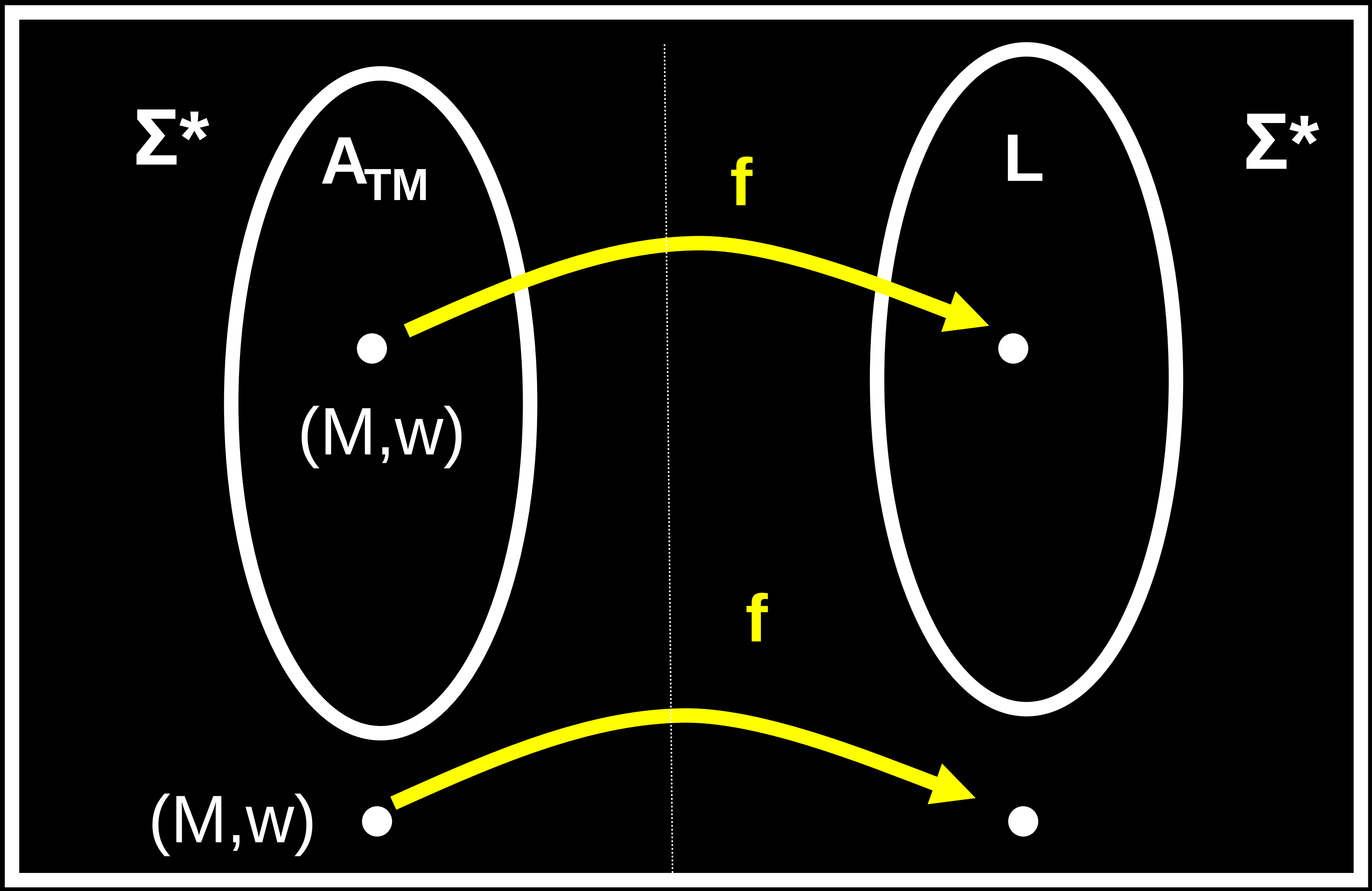
Then L is undecidable

Proof: Will show:

A_{TM} is mapping reducible to L

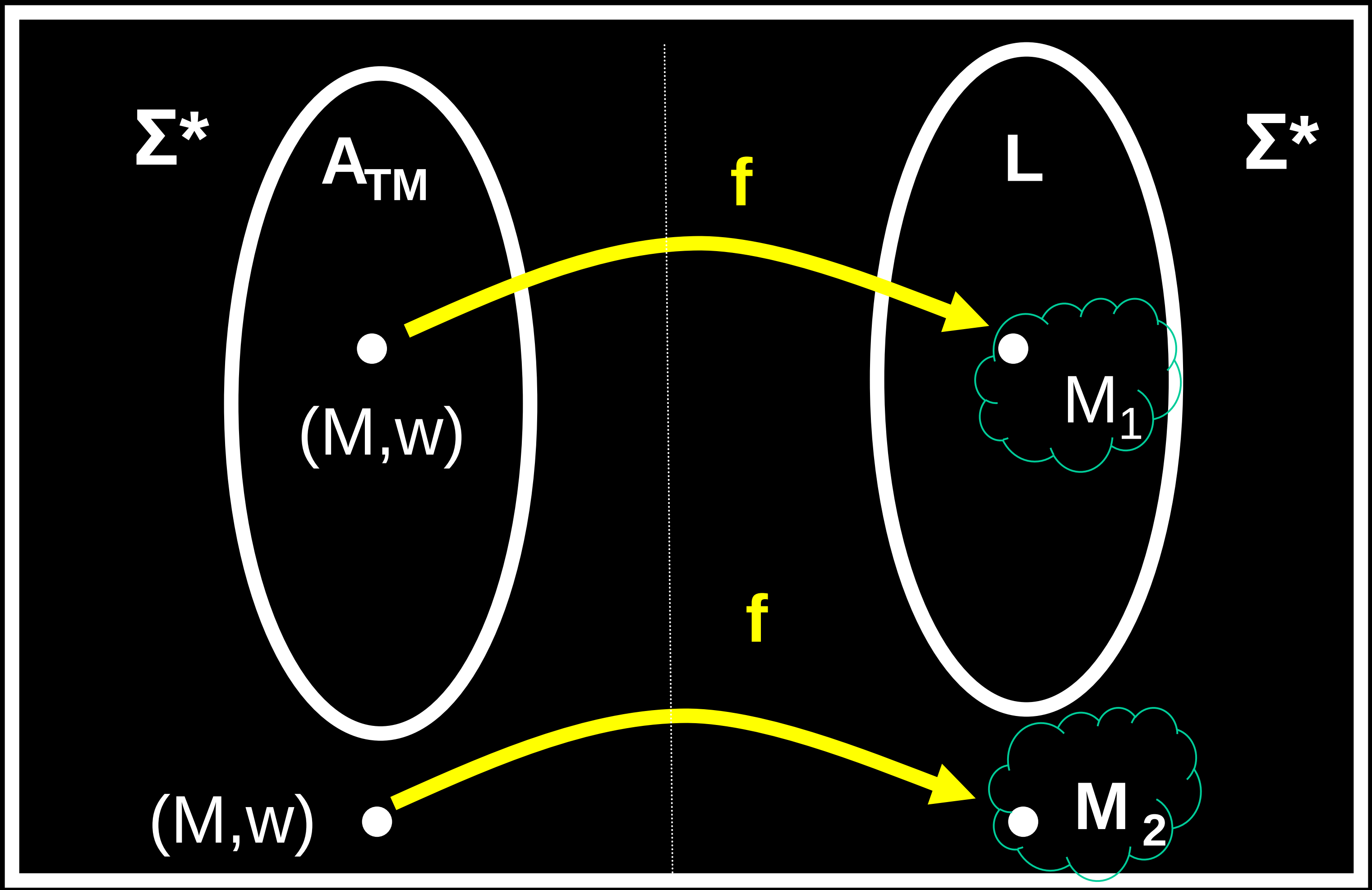
Proof: Show L is undecidable

Show: A_{TM} is mapping reducible to L



Proof: Show L is undecidable

Show: A_{TM} is mapping reducible to L



RICE'S THEOREM

Proof:

Define $M_\emptyset$ to be a TM that never halts

Assume, WLOG, that $M_\emptyset \notin L$ Why?

Let $M_1 \in L$ (such M_1 exists, by assumption)

Show A_{TM} is mapping reducible to L :

RICE'S THEOREM

Proof:

Define $M_\emptyset$ to be a TM that never halts

Assume, WLOG, that $M_\emptyset \notin L$ Why?

Let $M_1 \in L$ (such M_1 exists, by assumption)

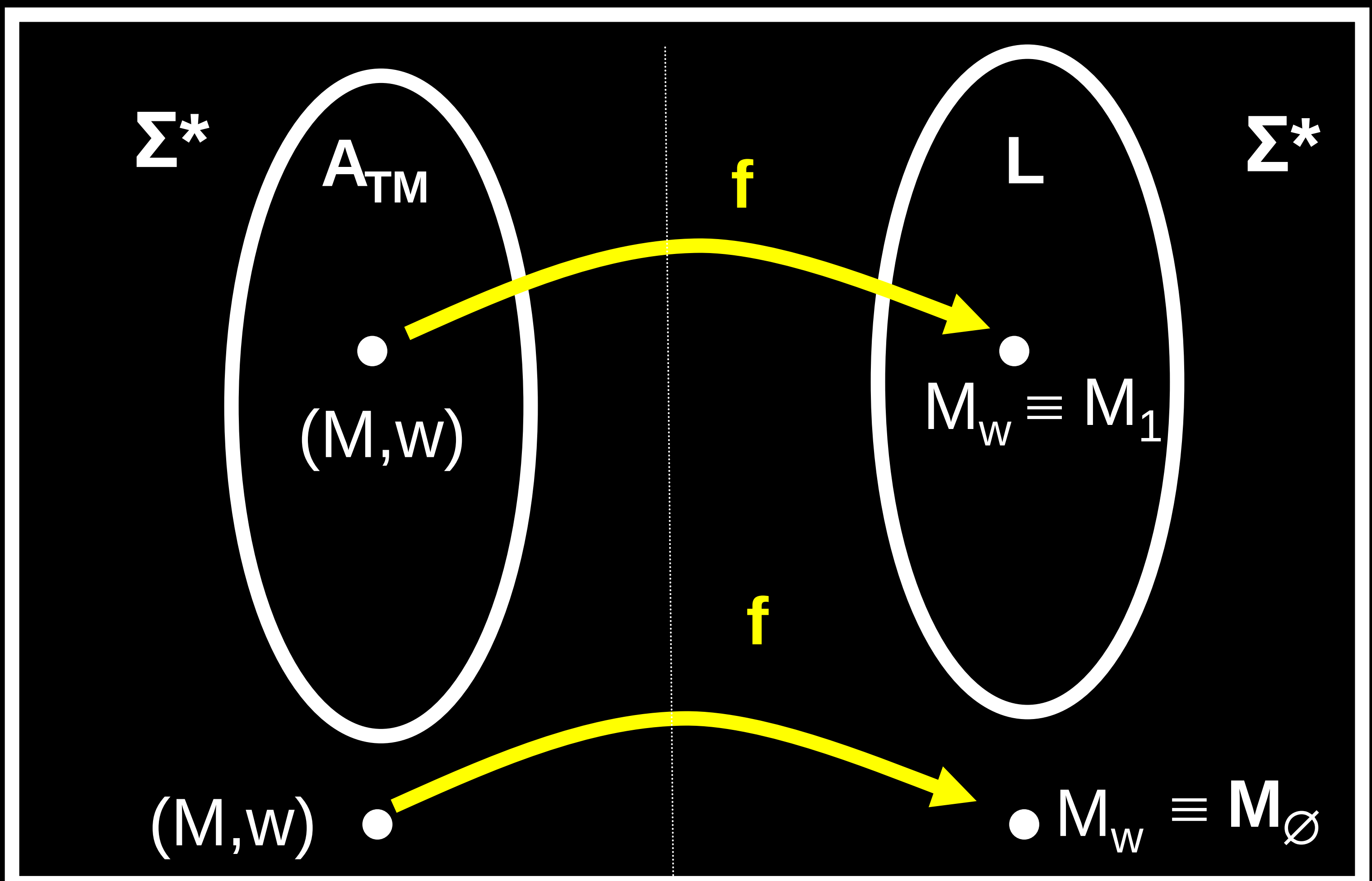
Show A_{TM} is mapping reducible to L :

Map (M, w) → M_w where

M_w(s) = accepts if both M(w) and M_1(s) accept
loops otherwise

What is the language of M_w ?

A_{TM} is mapping reducible to L



QED

Problem

Let S = { M | M is a TM with the property:
for all w, M(w) accepts implies M(w^R) accepts}.

S is undecidable.

A_{TM} = { (M,w) | M is a TM that accepts string w }

$HALT_{TM}$ = { (M,w) | M is a TM that halts on string w }

E_{TM} = { M | M is a TM and L(M) = ∅ }

REG_{TM} = { M | M is a TM and L(M) is regular}

EQ_{TM} = {(M, N) | M, N are TMs and L(M) =L(N)}

ALL_{PDA} = { P | P is a PDA and L(P) = Σ* }

ALL UNDECIDABLE

Where is Rice's Theorm Applicable?

Which are SEMI-DECIDABLE or not?

The rest of the content of today's lecture has been a major source of headaches and misunderstandings

"The recursion theorem is just like tennis. Unless you're exposed to it at age five, you'll never become world class."

-Juris Hartmanis (Turing Award 1993)

(Note: Juris didn't see the recursion theorem until he was in his 20's….)

THE RECURSION THEOREM

Theorem: Let T be a Turing machine that computes a function $t : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$.

Then there is a Turing machine R that computes a function $r : \Sigma^* \rightarrow \Sigma^*$, where for every string w,

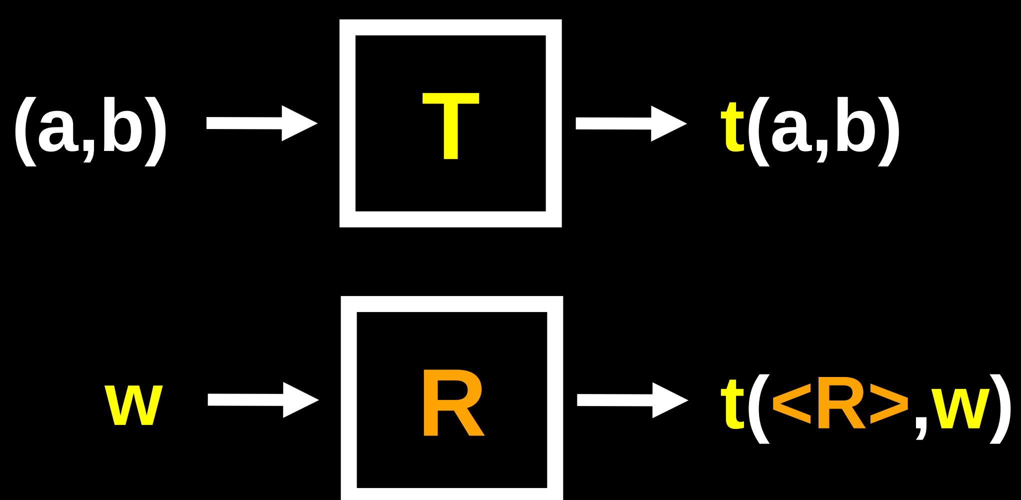
$$r(w) = t(<R>, w)$$

THE RECURSION THEOREM

Theorem: Let **T** be a Turing machine that computes a function $t : \Sigma^* \times \Sigma^* \to \Sigma^*$.

Then there is a Turing machine **R** that computes a function $r : \Sigma^* \to \Sigma^*$, where for every string **w**,

$$r(w) = t(<R>, w)$$



Recursion Theorem says:

A Turing machine can obtain its own description (code), and compute with it

. We can use the operation:

“Obtain your own description”

in pseudocode!

Given a computable **t**, we can get a computable **r** such that **r(w) = t(<R>,w)** where **<R>** is a description **of r**

Recursion Theorem says:

A Turing machine can obtain its own description (code), and compute with it

. We can use the operation:

"Obtain your own description"

in pseudocode!

Given a computable **t**, we can get a computable **r** such that **r(w) = t(<R>,w)** where **<R>** is a description **of r**

INSIGHT: T (or t) is really R (or r)

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

Proof (using the Recursion Theorem)**:**

Assume H decides A_{TM} (Informal Proof)

Construct machine R such that on input w:

1. **Obtains its own description < R>**
2. **Runs H on (<R>, w) and flips the output**

Running R on input w always does the opposite of what H says it should!

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

Proof (using the Recursion Theorem)**:**

Assume H decides A_{TM} (Formal Proof)

Let $T_H(x, w)$ **=**
- **Reject if H (x, w) accepts**
- **Accept if H (x, w) rejects**

(Here x is viewed as a code for a TM)

By the *Recursion Theorem*, there is a **TM R** such that:

$R(w) = T_H(\langle R\rangle, w)$ **=**
- **Reject if H (<R>, w) accepts**
- **Accept if H (<R>, w) rejects**

Contradiction!

MIN_{TM} = {<M>| M is a minimal TM, wrt |<M>|}

Theorem: MIN_{TM} is not RE.

Proof (using the Recursion Theorem)**:**

MIN_{TM} = {<M>| M is a minimal TM, wrt |<M>|}

Theorem: MIN_{TM} is not RE.

Proof (using the Recursion Theorem):

Assume E enumerates MIN_{TM} (Informal Proof)

Construct machine R such that on input w:

1. Obtains its own description <R>
2. Runs E until a machine D appears with a longer description than of R
3. Simulate D on w

Contradiction. Why?

MIN_{TM} = {<M>| M is a minimal TM, wrt |<M>|}

Theorem: MIN_{TM} is not RE.

Proof (using the Recursion Theorem)**:**

Assume E enumerates MIN_{TM} (Formal Proof)

Let T_E(x, w) = D(w) where <D> is first in E's enumeration s.t. |<D>| > |x|

By the *Recursion Theorem*, there is a **TM R** such that:

R(w) = T_E(<R>, w) = D(w)

where <D> is first in E's enumeration s.t. |<D>| > |<R>|

Contradiction. Why?

THE FIXED-POINT THEOREM

Theorem: Let $f : \Sigma^* \rightarrow \Sigma^*$ be a computrable function. There is a TM R such that f(<R>) describes a TM that is *equivalent* to R.

THE FIXED-POINT THEOREM

Theorem: **Let** $f : \Sigma^* \to \Sigma^*$ **be a computrable function. There is a TM R such that f(<R>) describes a TM that is *equivalent* to R.**

Proof: **Pseudocode for the TM R:**

(Informal Proof)

On input w:

1. **Obtain the description <R>**
2. **Let g = f(<R>) and interpret g as a code for a TM G**
3. **Accept w iff G(w) accepts**

THE FIXED-POINT THEOREM

Theorem: **Let $f : \Sigma^* \rightarrow \Sigma^*$ be a computrable function. There is a TM R such that f(<R>) describes a TM that is *equivalent* to R.**

Proof: **Let T_f(x, w) = G(w) where <G> = f (x)**

(Here f(x) is viewed as a code for a TM)

By the *Recursion Theorem*, there is a TM R such that:

R(w) = T_f(<R>, w)

THE FIXED-POINT THEOREM

Theorem: **Let $f : \Sigma^* \to \Sigma^*$ be a computrable function. There is a TM R such that f(<R>) describes a TM that is *equivalent* to R.**

Proof: **Let T_f(x, w) = G(w) where <G> = f (x)**

(Here f(x) is viewed as a code for a TM)

By the *Recursion Theorem*, there is a TM **R** such that:

R(w) = T_f(<R>, w) = G(w) where <G> = f (<R>)

Hence R $\equiv$ G where <G> = f (<R>), ie <R> "$\equiv$" f (<R>)

So R is a fixed point of f !

THE FIXED-POINT THEOREM

Theorem: **Let $f : \Sigma^* \rightarrow \Sigma^*$ be a computrable function. There is a TM R such that f(<R>) describes a TM that is *equivalent* to R.**

Example:

Suppose a virus flips the first bit of each word w in Σ^* (or in each TM).

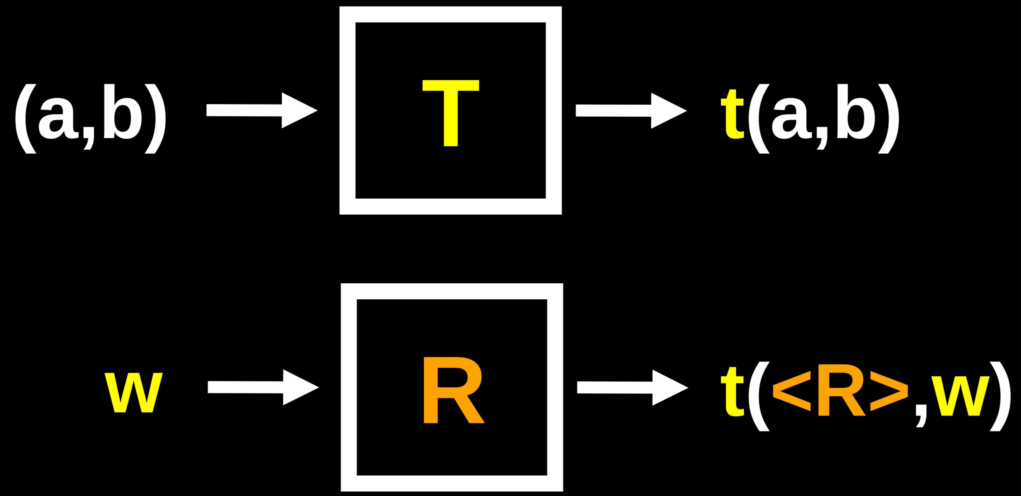
Then there is a TM R that "remains uninfected".

THE RECURSION THEOREM

Theorem: Let T be a Turing machine that computes a function $t : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$.

Then there is a Turing machine R that computes a function $r : \Sigma^* \rightarrow \Sigma^*$, where for every string w,

$$r(w) = t(<R>, w)$$



THE RECURSION THEOREM

Theorem: Let T be a Turing machine that computes a function $t : \Sigma^* \times \Sigma^* \to \Sigma^*$.

Then there is a Turing machine R that computes a function $r : \Sigma^* \to \Sigma^*$, where for every string w,

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

So first, need to show how to construct a TM that computes its own description (ie code).

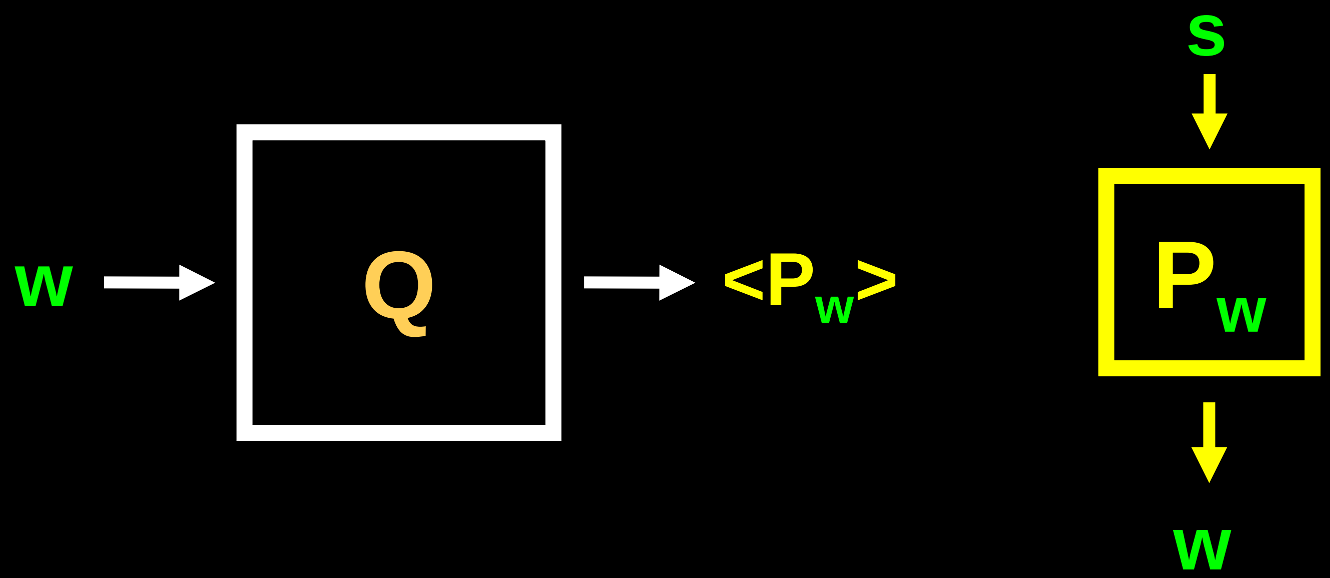
A NOTE ON SELF REFERENCE

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

Print this sentence.

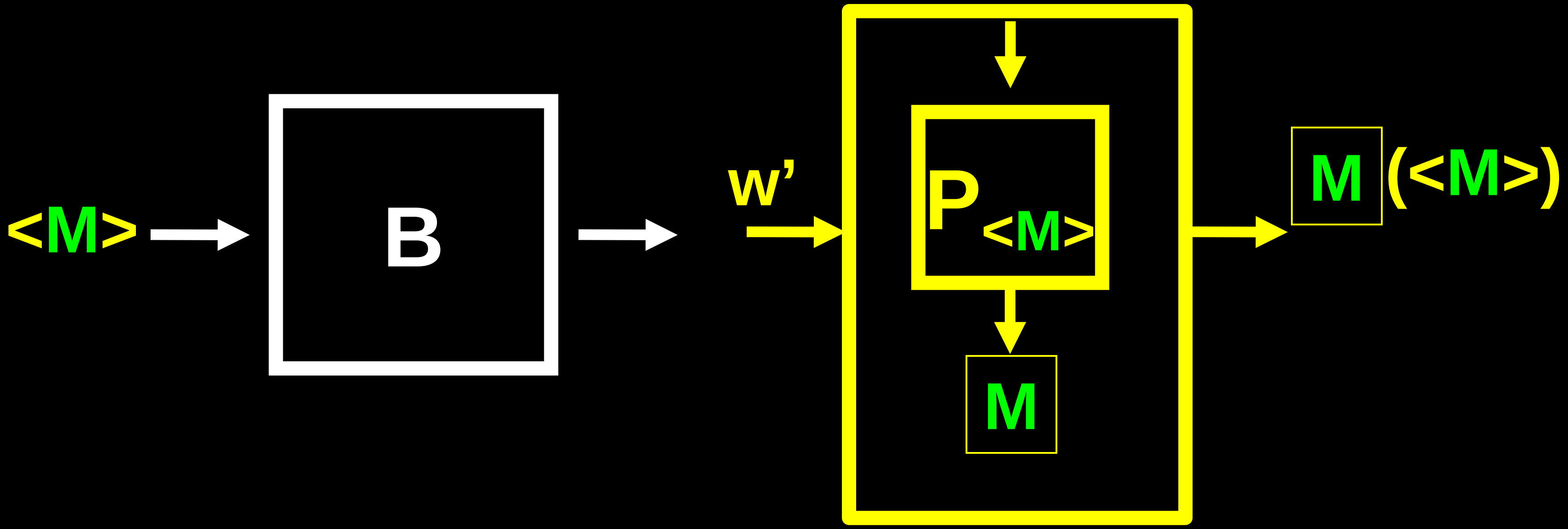
Print two copies of the following (the stuff inside quotes), and put the second copy in quotes:
"Print two copies of the following (the stuff inside quotes), and put the second copy in quotes:"

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



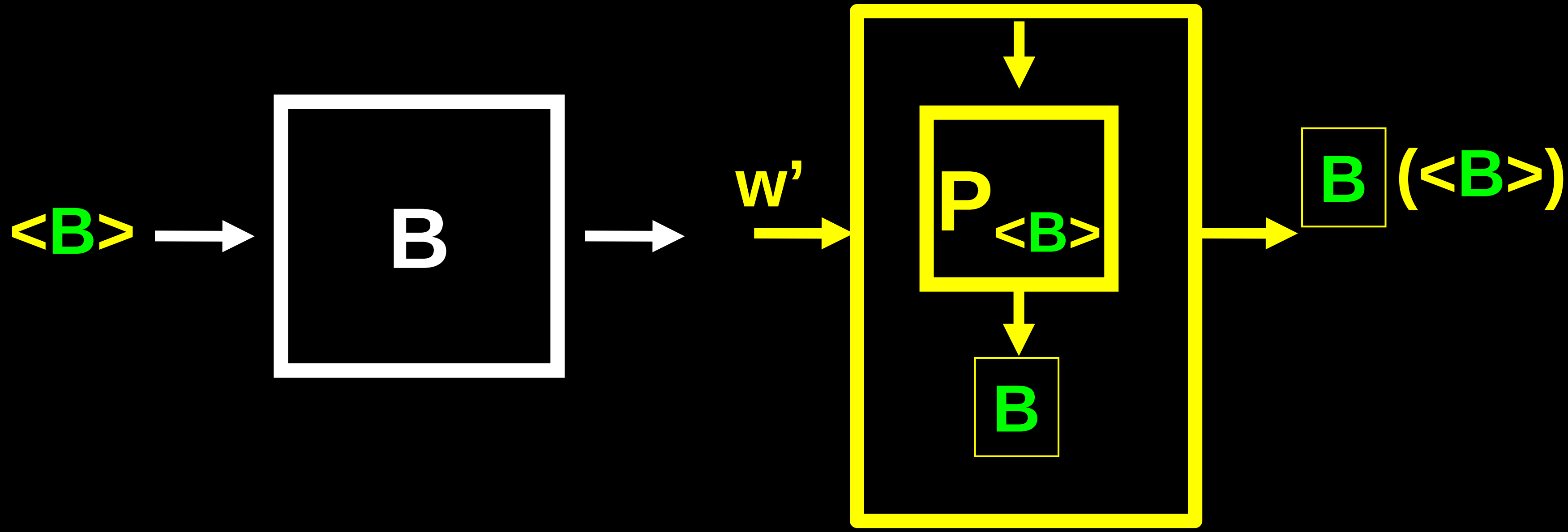
TM Q computes q

A TM SELF THAT PRINTS <SELF>



B (<M>) = < $P_{<M>}$ M> where $P_{<M>}$ M (w') = M (<M>)

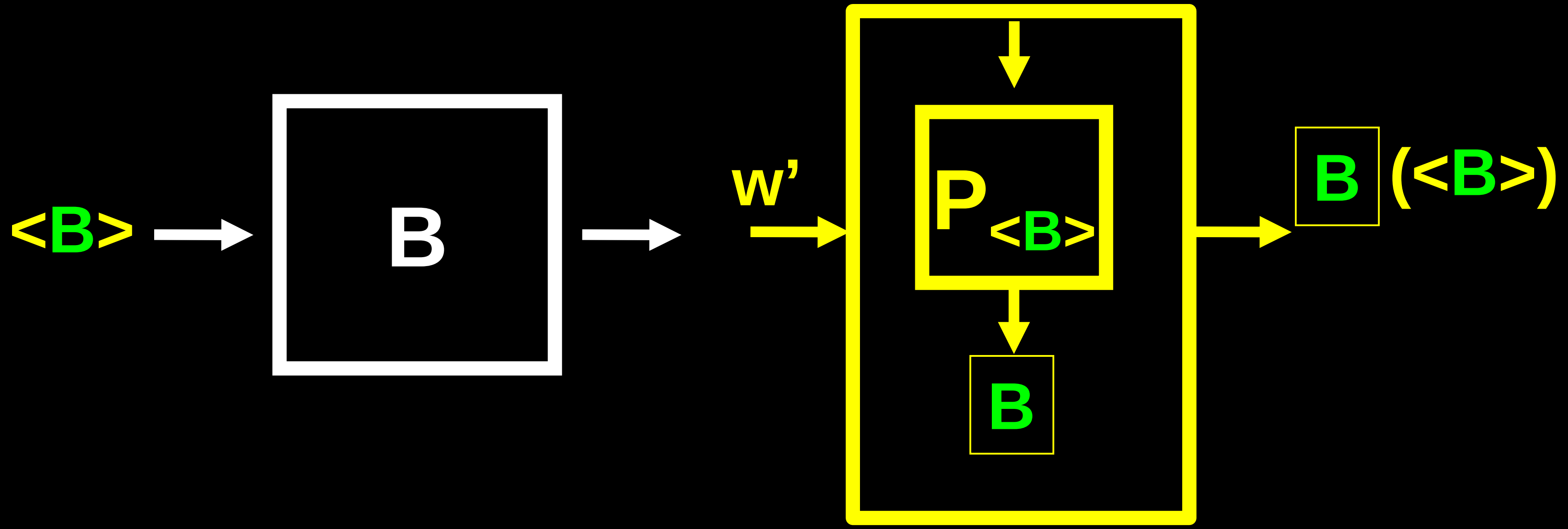
A TM SELF THAT PRINTS <SELF>



$B(\langle M \rangle) = \langle P_{\langle M \rangle} M \rangle$ **where** $P_{\langle M \rangle} M(w') = M(\langle M \rangle)$

So, $B(\langle B \rangle) = \langle P_{\langle B \rangle} B \rangle$ where $P_{\langle B \rangle} B(w') = B(\langle B \rangle)$

A TM SELF THAT PRINTS <SELF>



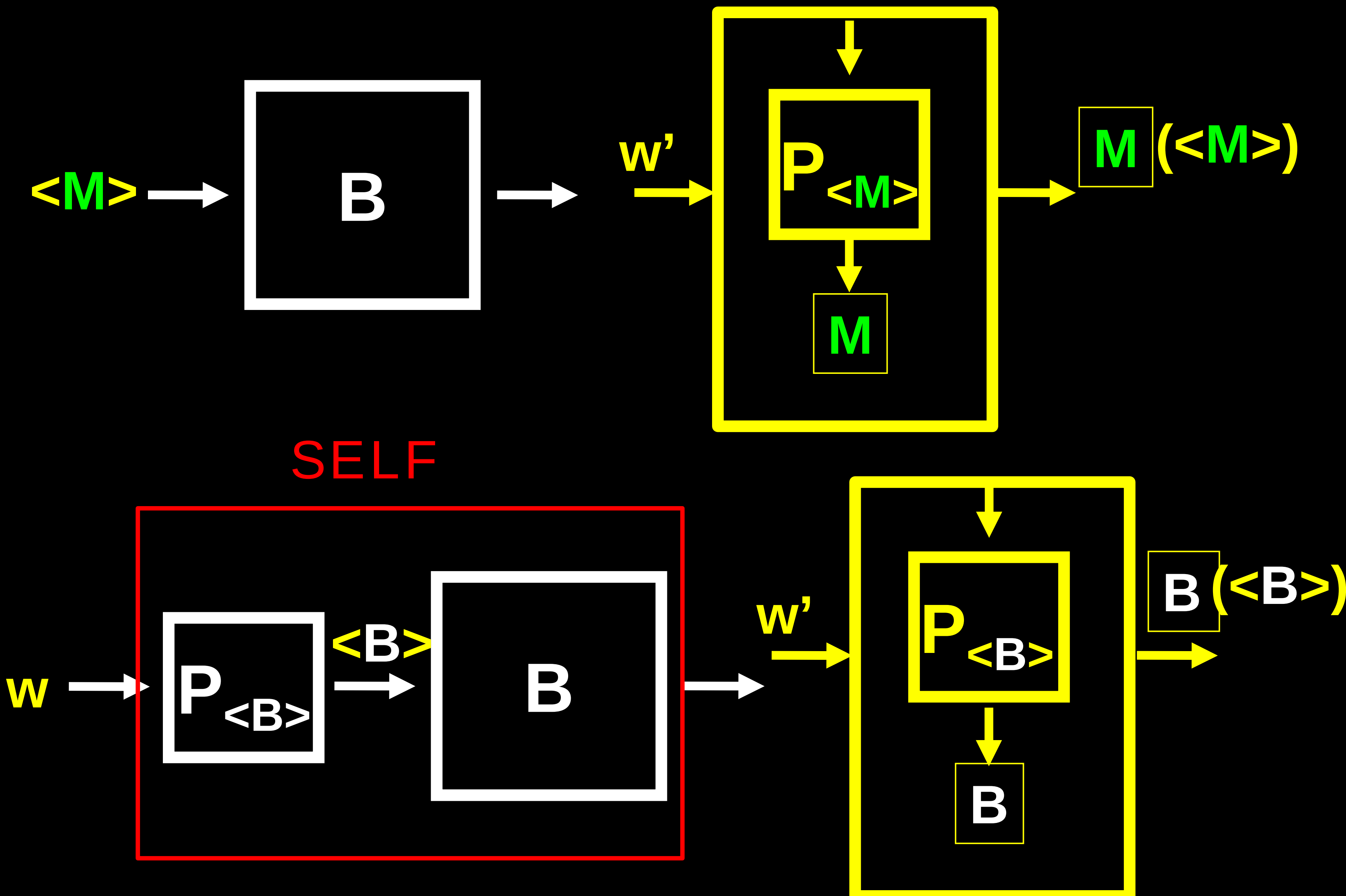
$B(\langle M\rangle) = \langle P_{\langle M\rangle} M\rangle$ **where** $P_{\langle M\rangle} M(w') = M(\langle M\rangle)$

So, $B(\langle B\rangle) = \langle P_{\langle B\rangle} B\rangle$ where $P_{\langle B\rangle} B(w') = B(\langle B\rangle)$

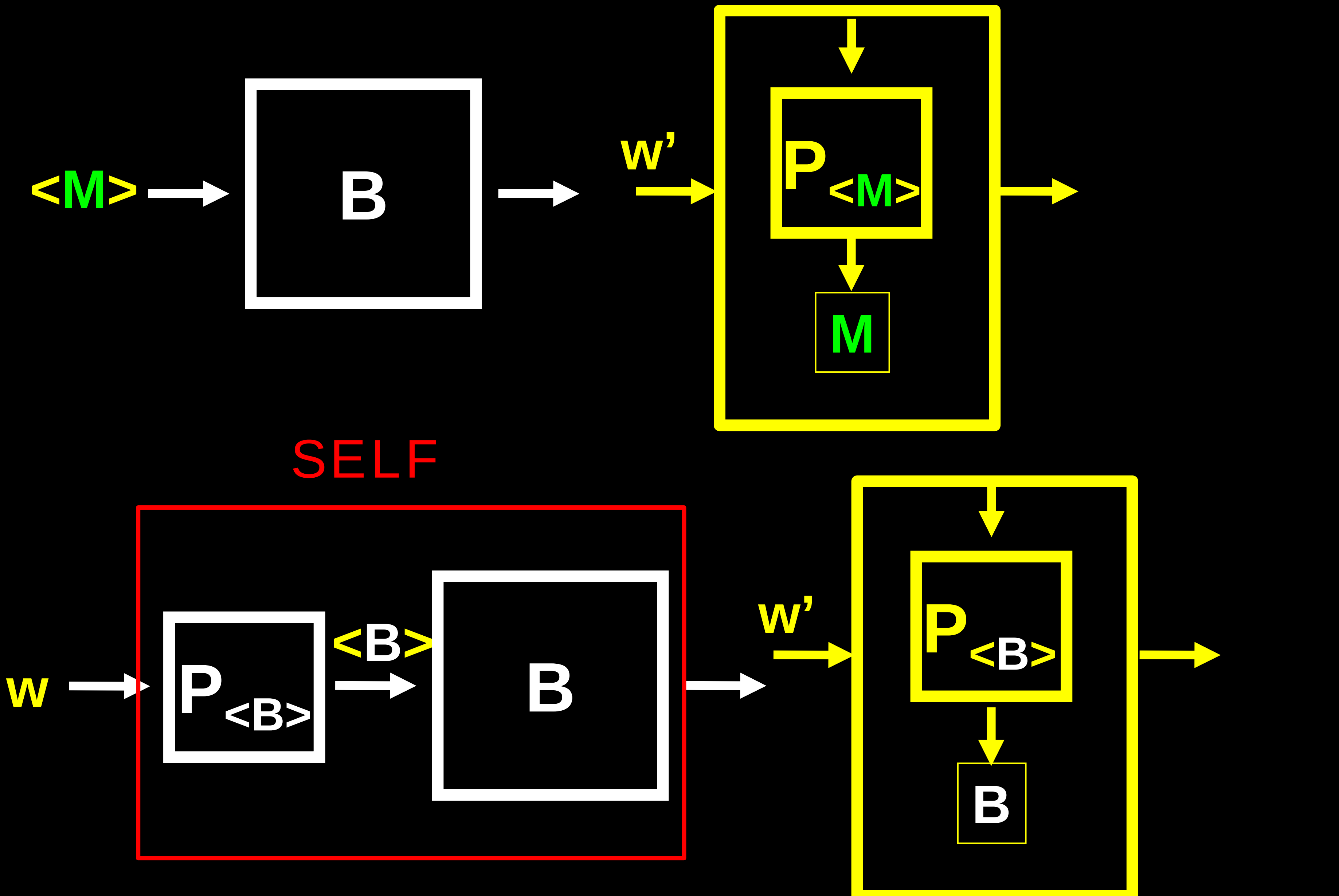
Now, $P_{\langle B\rangle} B(w') = B(\langle B\rangle) = \langle P_{\langle B\rangle} B_{>}\rangle$

So, let **SELF** $= P_{\langle B\rangle} B$

A TM SELF THAT PRINTS <SELF>



A TM SELF THAT PRINTS <SELF>



A NOTE ON SELF REFERENCE

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

Print this sentence.

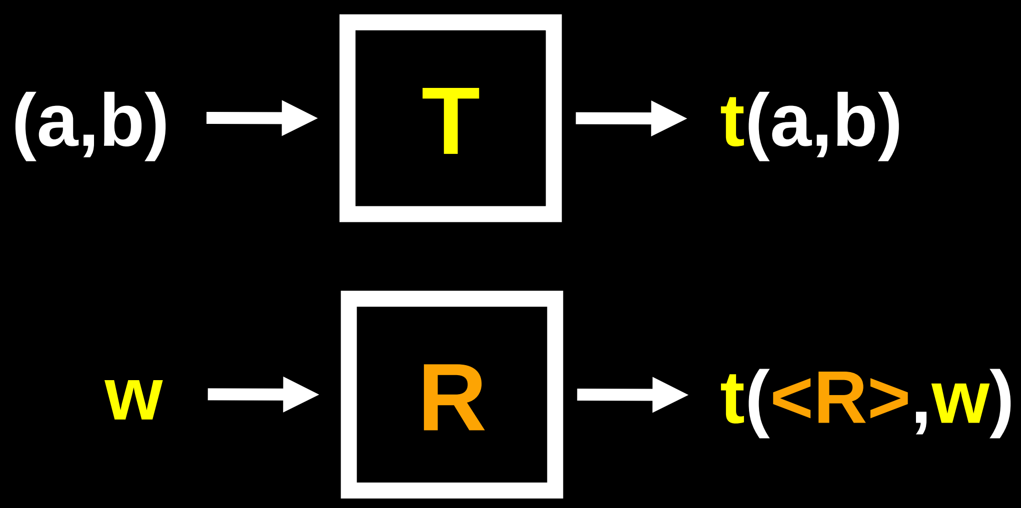
Print two copies of the following (the stuff inside quotes), and put the second copy in quotes: **= B**

"Print two copies of the following (the stuff inside quotes), and put the second copy in quotes:" **$= P_{<B>}$**

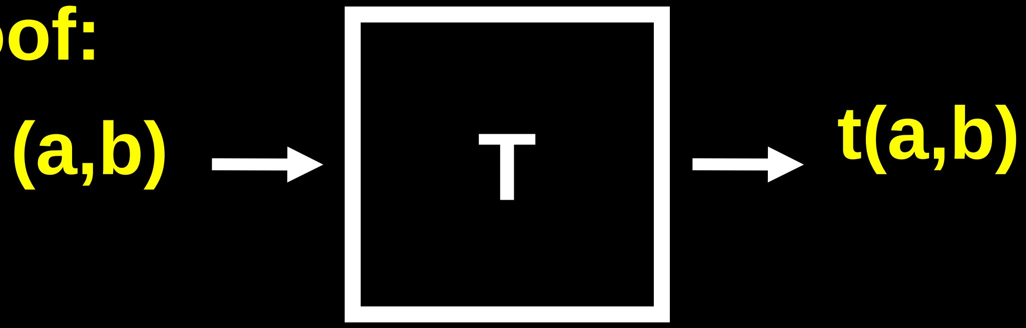
THE RECURSION THEOREM

Theorem: Let T be a Turing machine that computes a function $t : \Sigma^* \times \Sigma^* \to \Sigma^*$. There is a Turing machine R that computes a function $r : \Sigma^* \to \Sigma^*$, where for every string w,

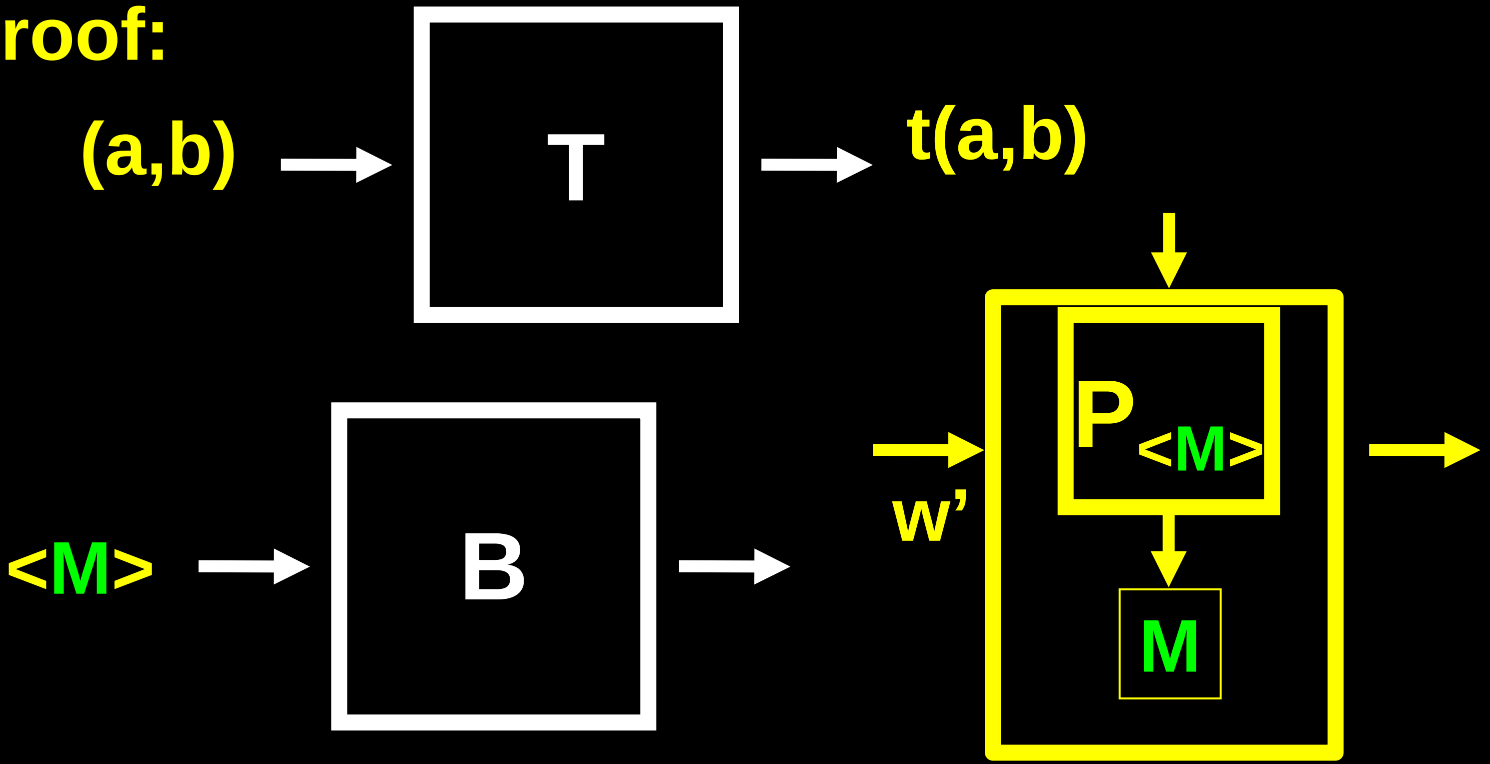
$$r(w) = t(<R>, w)$$



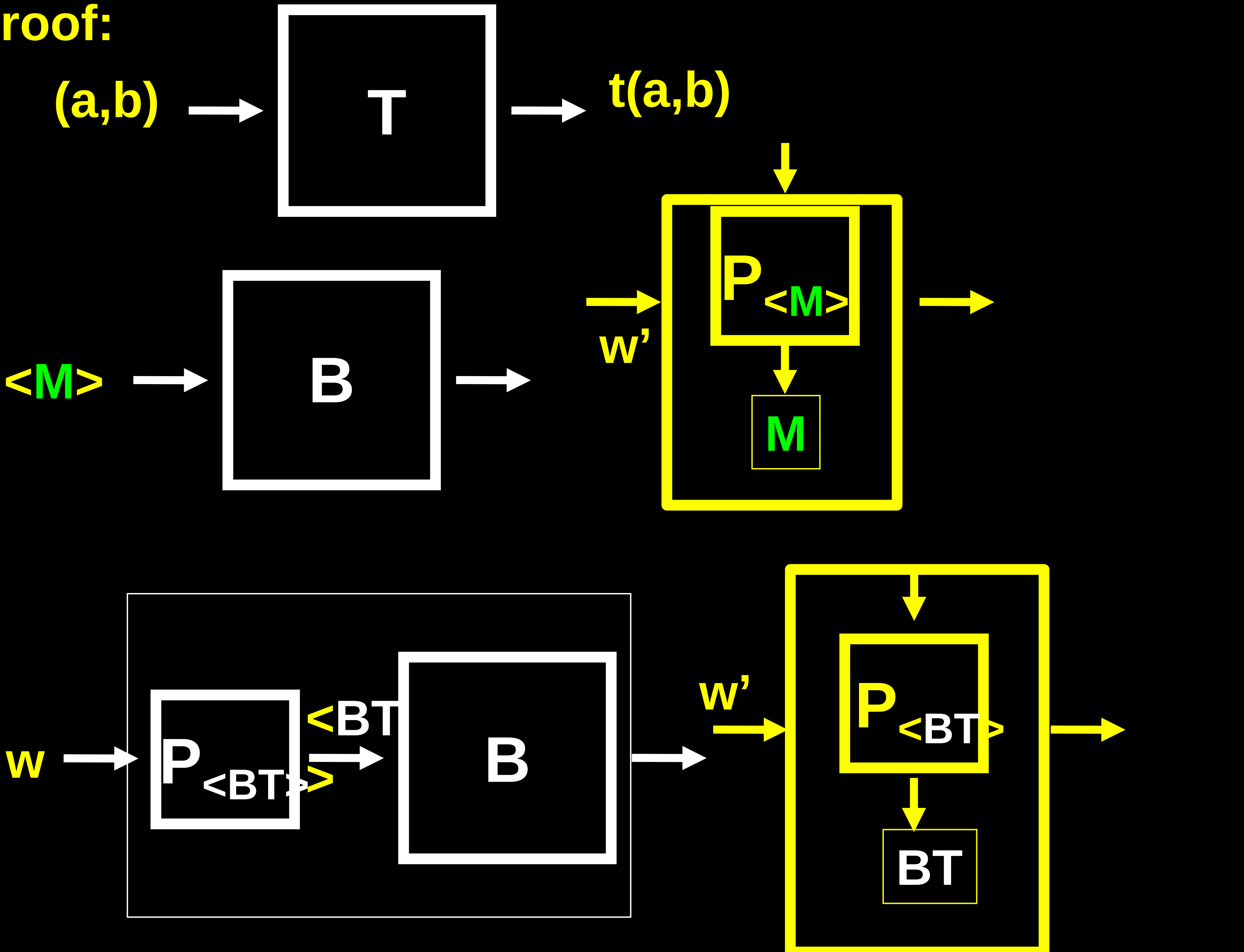
Proof:



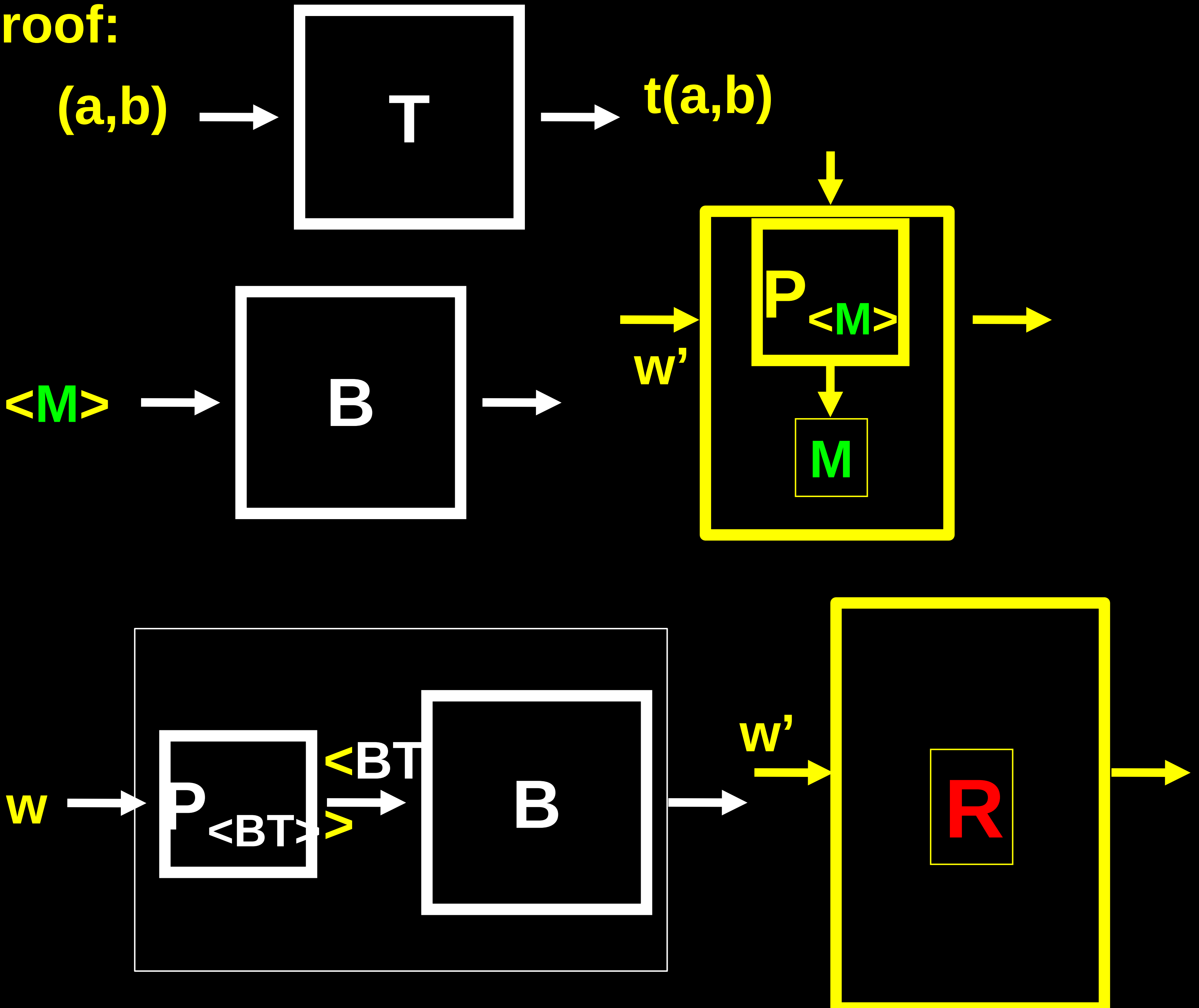
Proof:



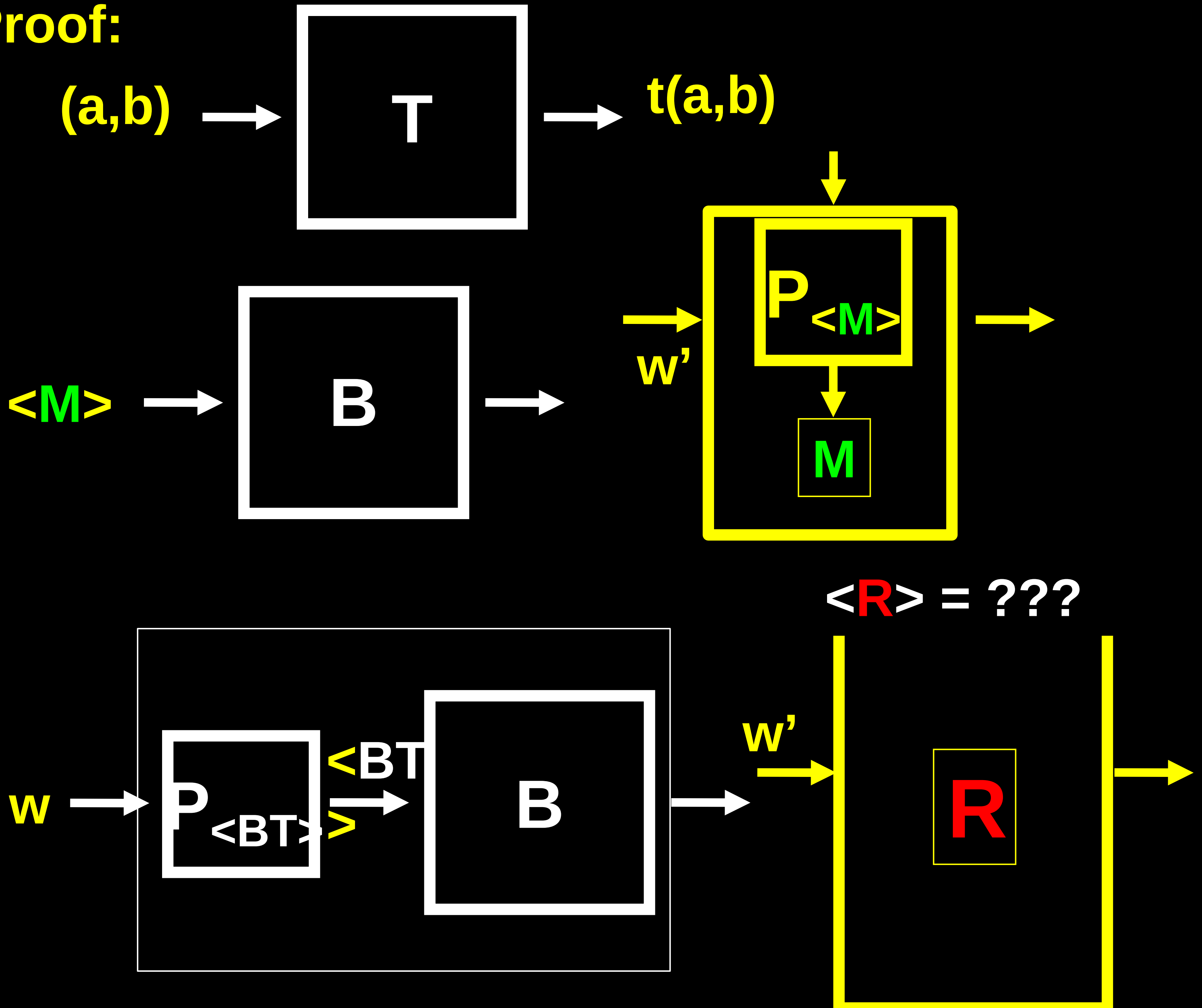
Proof:



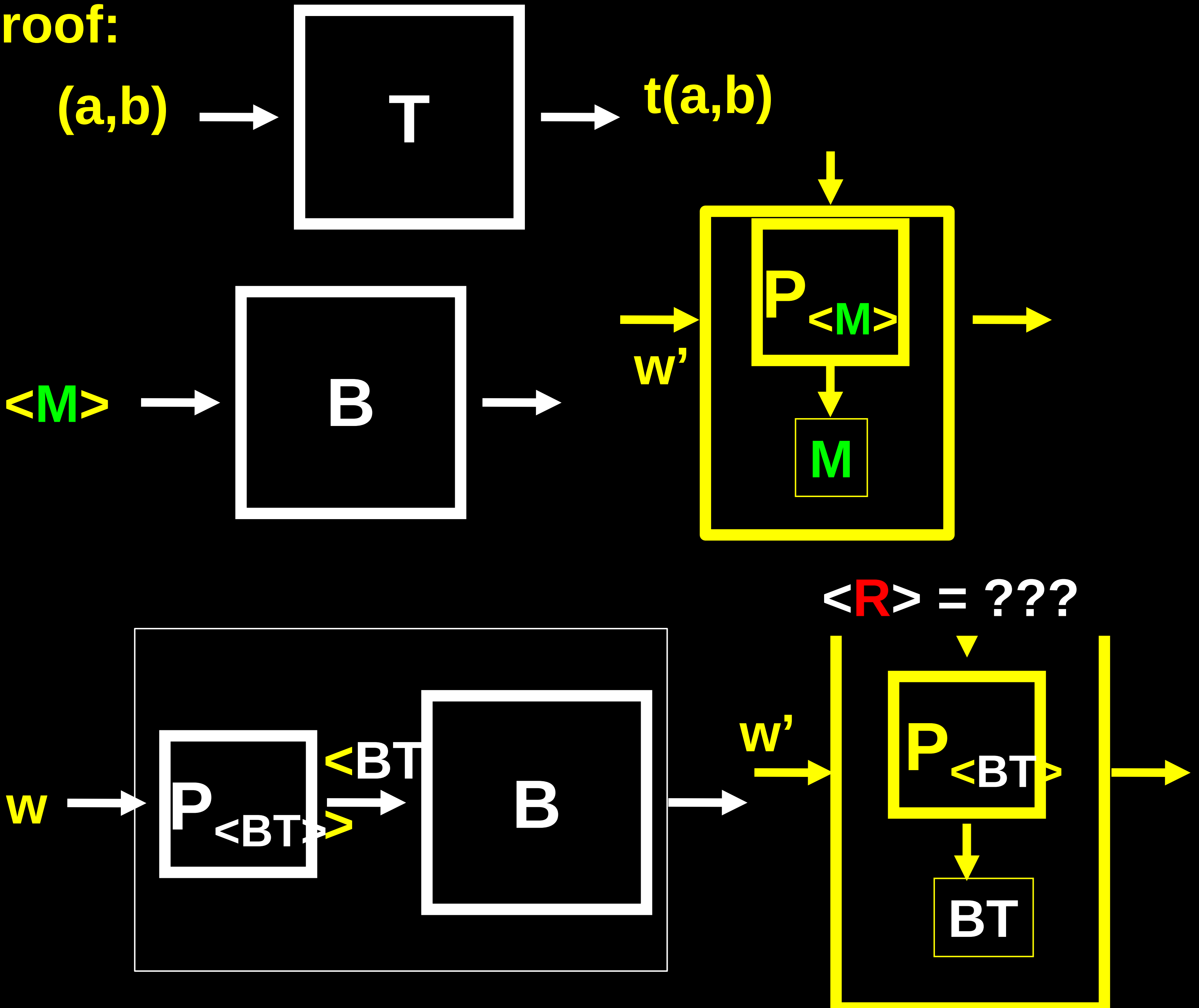
Proof:



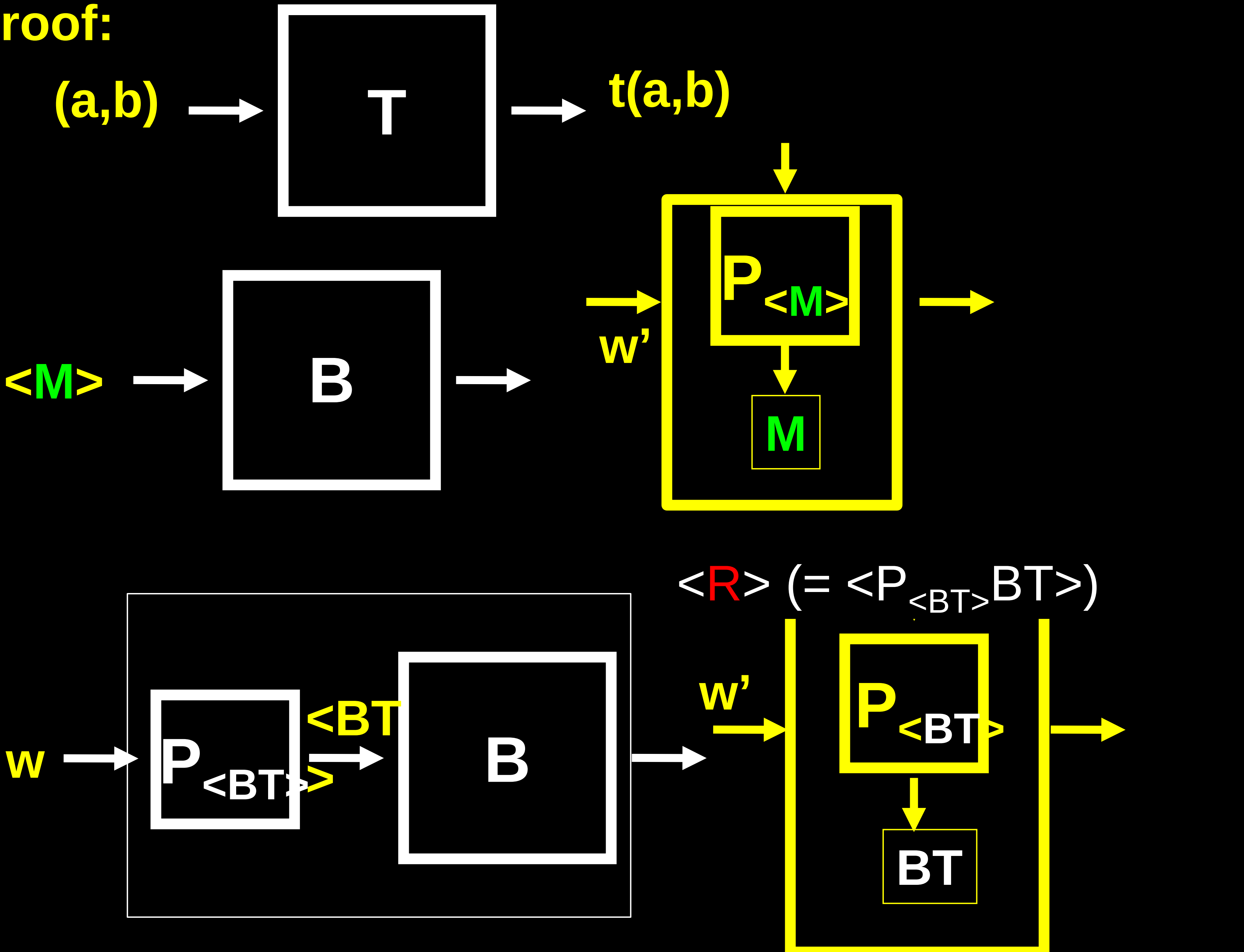
Proof:



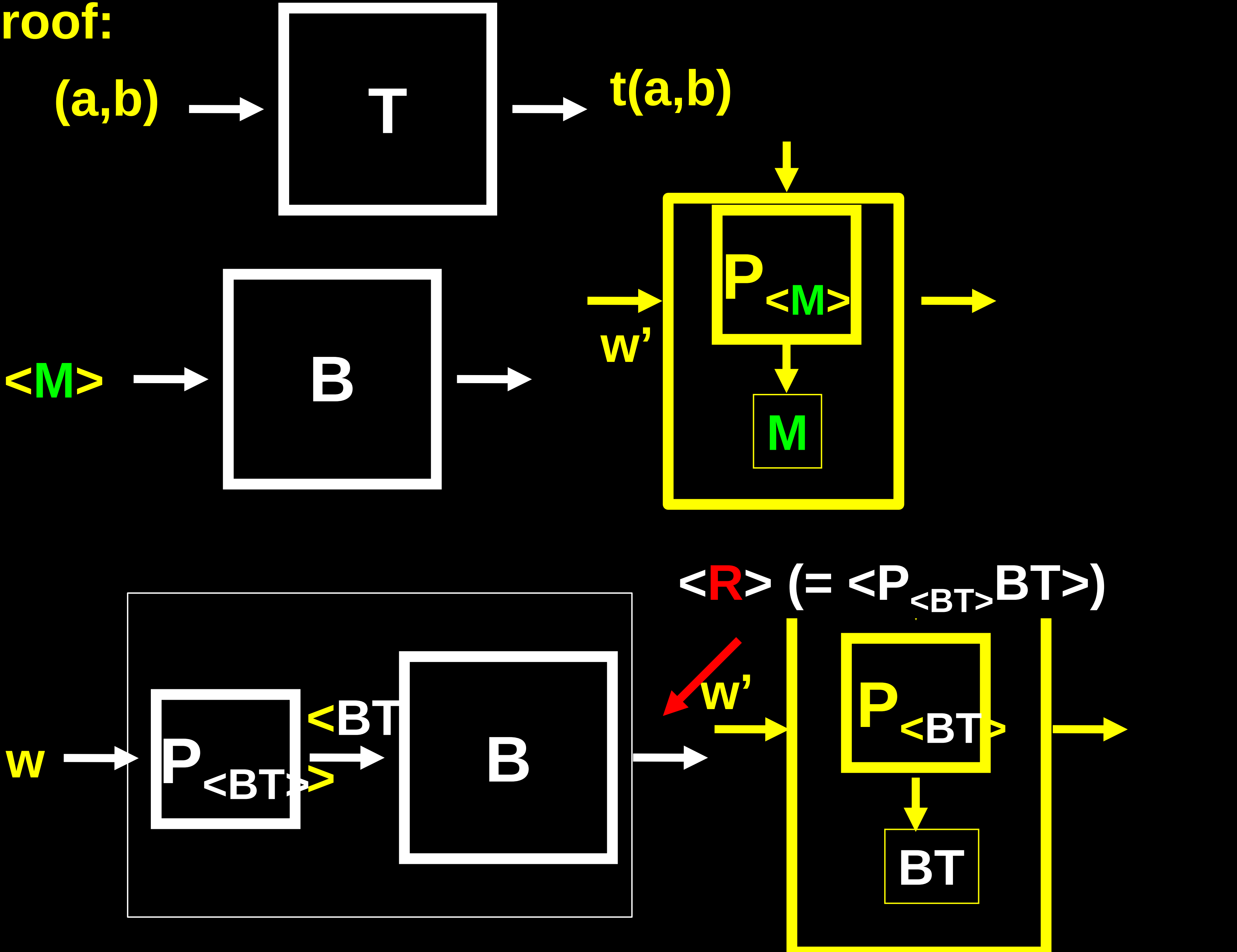
Proof:



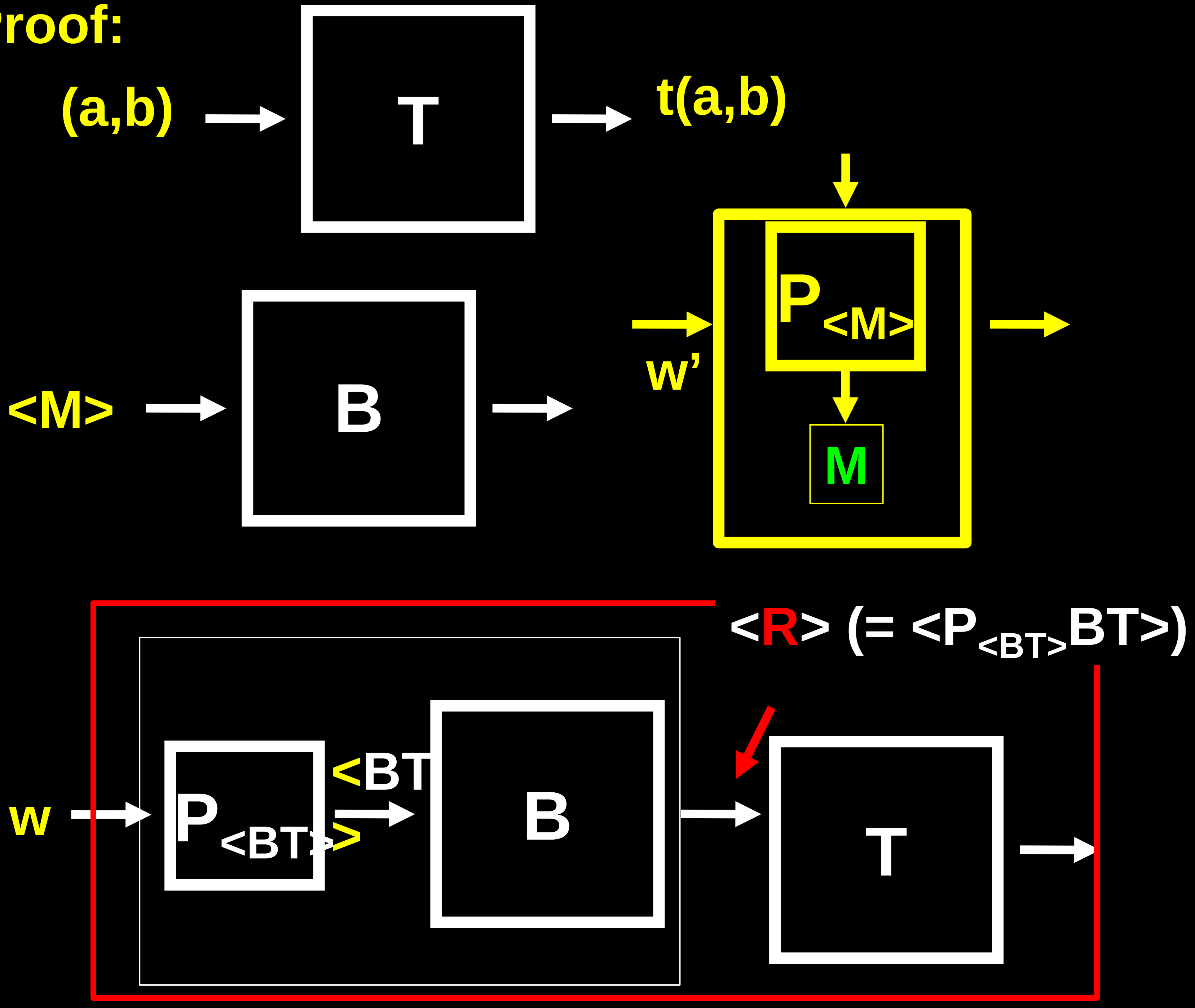
Proof:



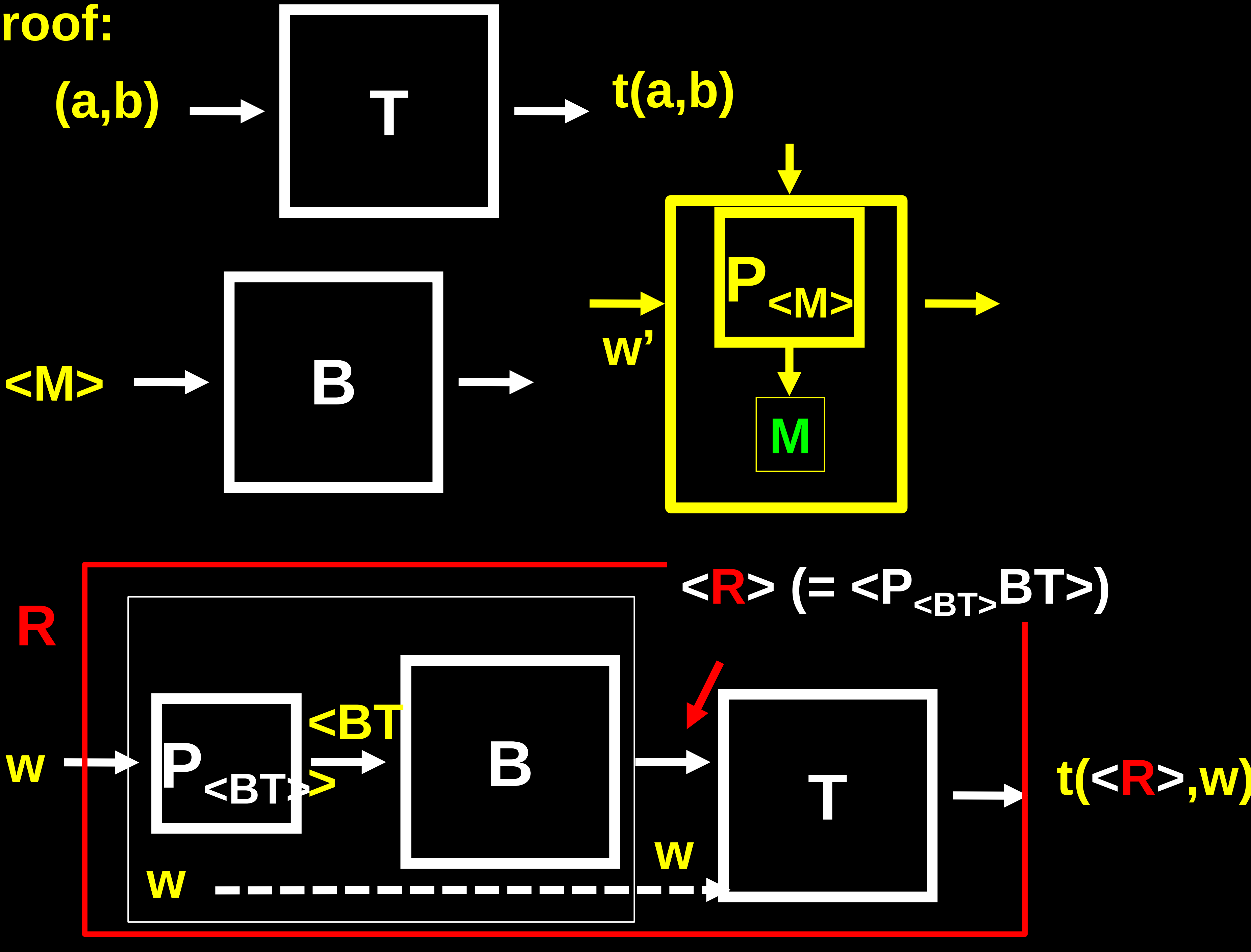
Proof:



Proof:



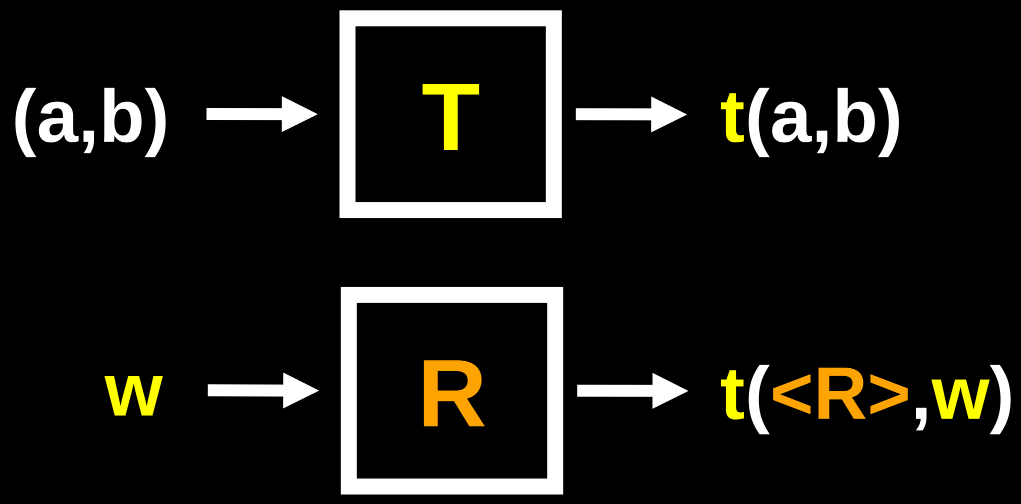
Proof:



THE RECURSION THEOREM

Theorem: **Let T be a Turing machine that computes a function $t : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$. There is a Turing machine R that computes a function $r : \Sigma^* \rightarrow \Sigma^*$, where for every string w,**

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



WWW.FLAC.WS

Read Chapter 6.1 and 6.3 for next time