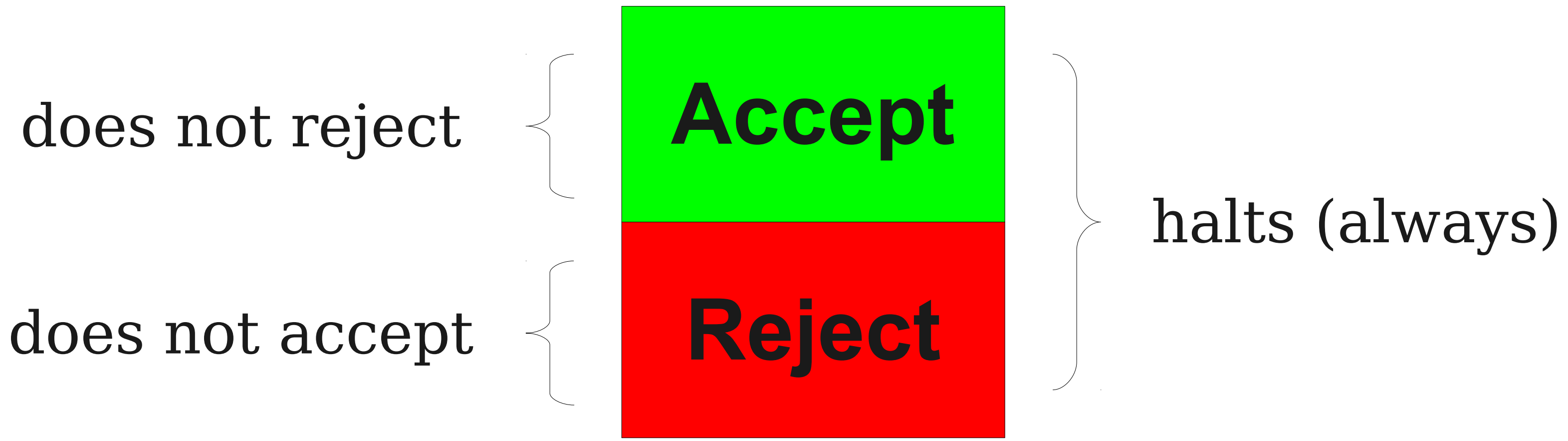
Reducibility

Part I

Deciders

- Some Turing machines always halt; they never go into an infinite loop.
- Turing machines of this sort are called **deciders**.
- For deciders, accepting is the same as not rejecting and rejecting is the same as not accepting.

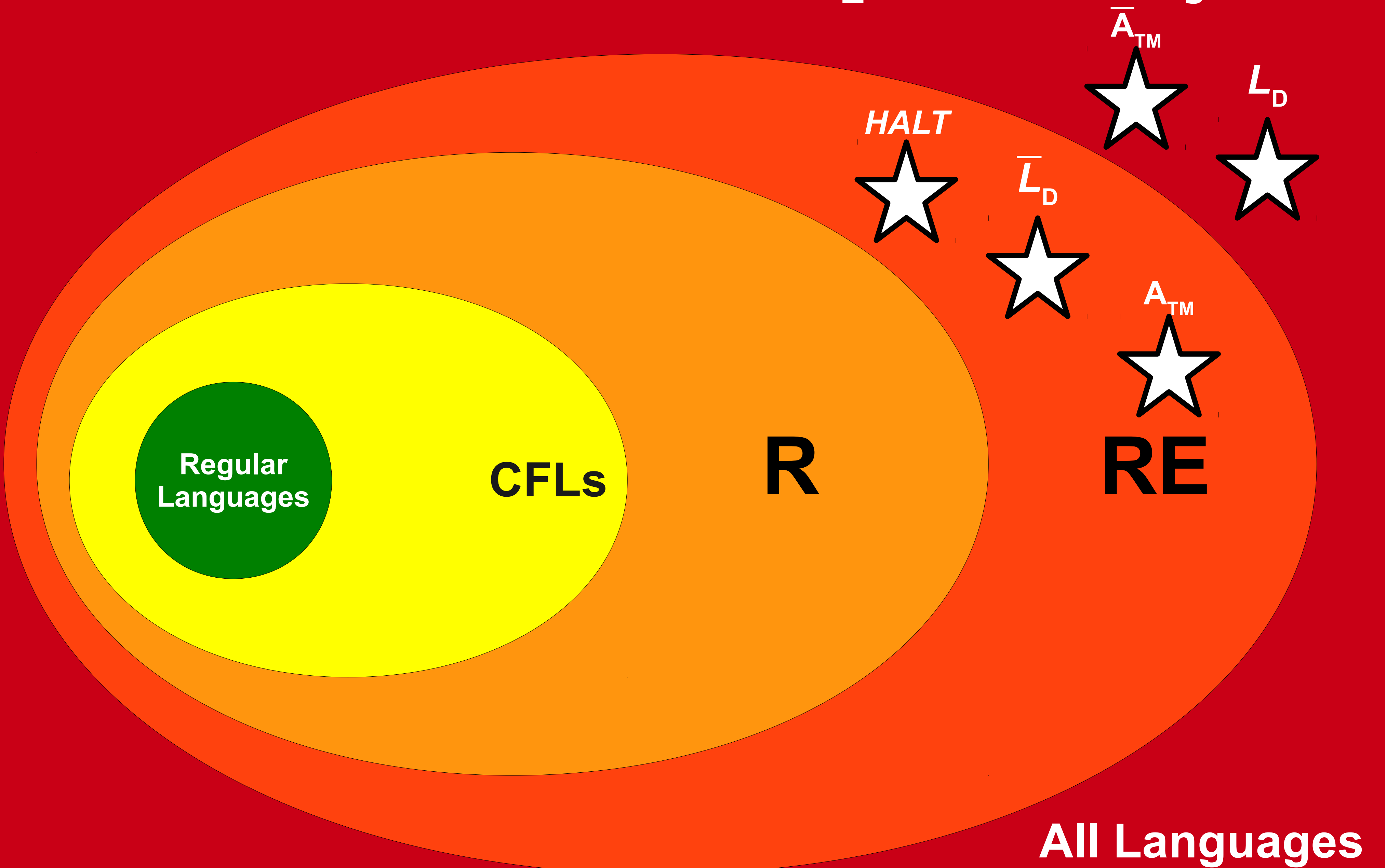


Decidable Languages

- A language L is called **decidable** iff there is a decider M such that $\mathscr{L}(M) = L$.
- Given a decider M, you *can* learn whether or not a string $w \in \mathscr{L}(M)$.
  - Run M on w.
  - Although it might take a staggeringly long time, M will eventually accept or reject w.
- The set $\mathbf{R}$ is the set of all decidable languages.

$$L \in \mathbf{R} \quad \text{iff} \quad L \text{ is decidable}$$

The Limits of Computability



A_{TM} and *HALT*

- Both A_{TM} and *HALT* are undecidable.
    - There is no way to decide whether a TM will accept or eventually terminate.
- However, both A_{TM} and *HALT* are recognizable.
    - We can always run a TM on a string w and accept if that TM accepts or halts.
- **Intuition:** The only general way to learn what a TM will do on a given string is to run it and see what happens.

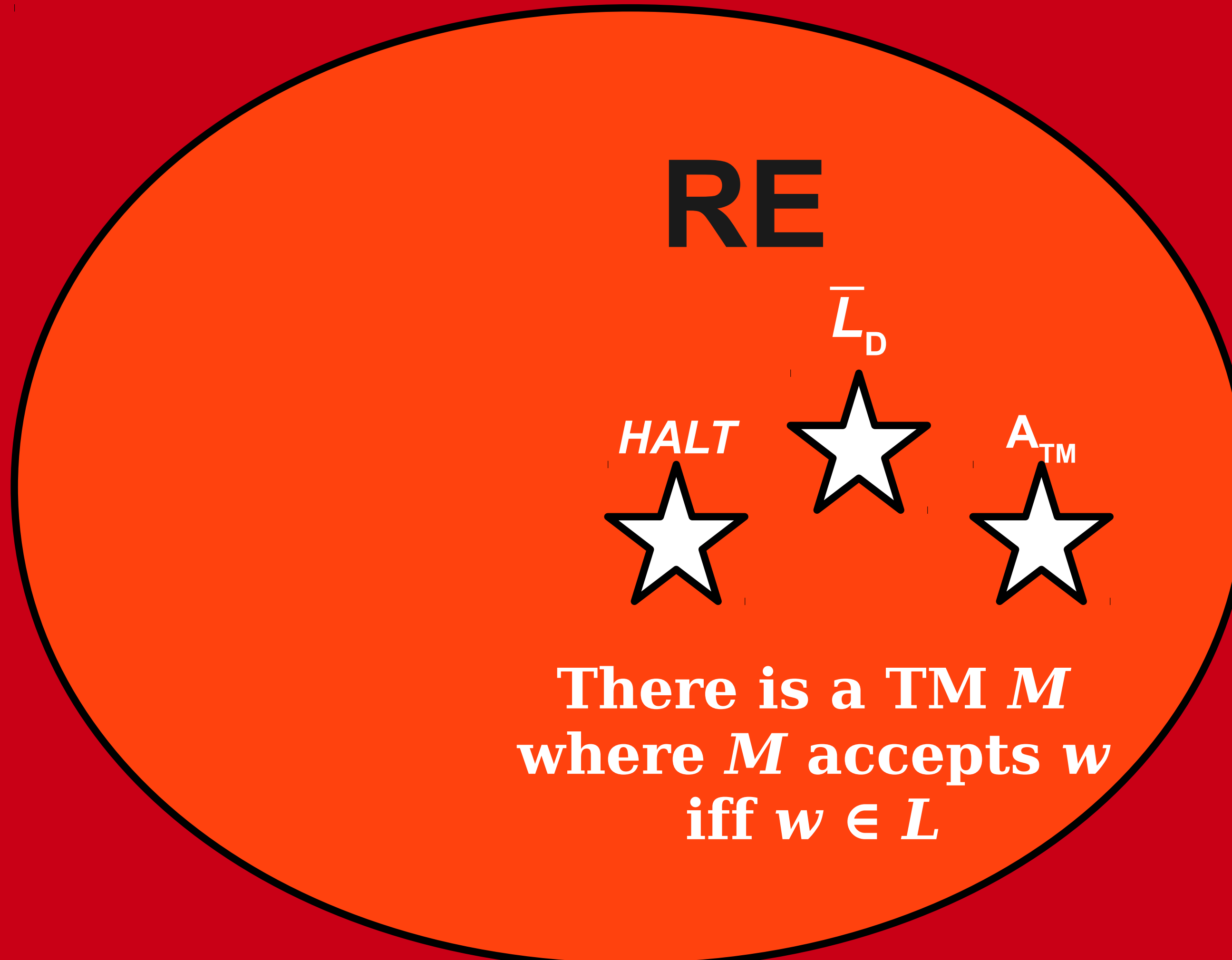
Resolving an Asymmetry

The Limits of Computability

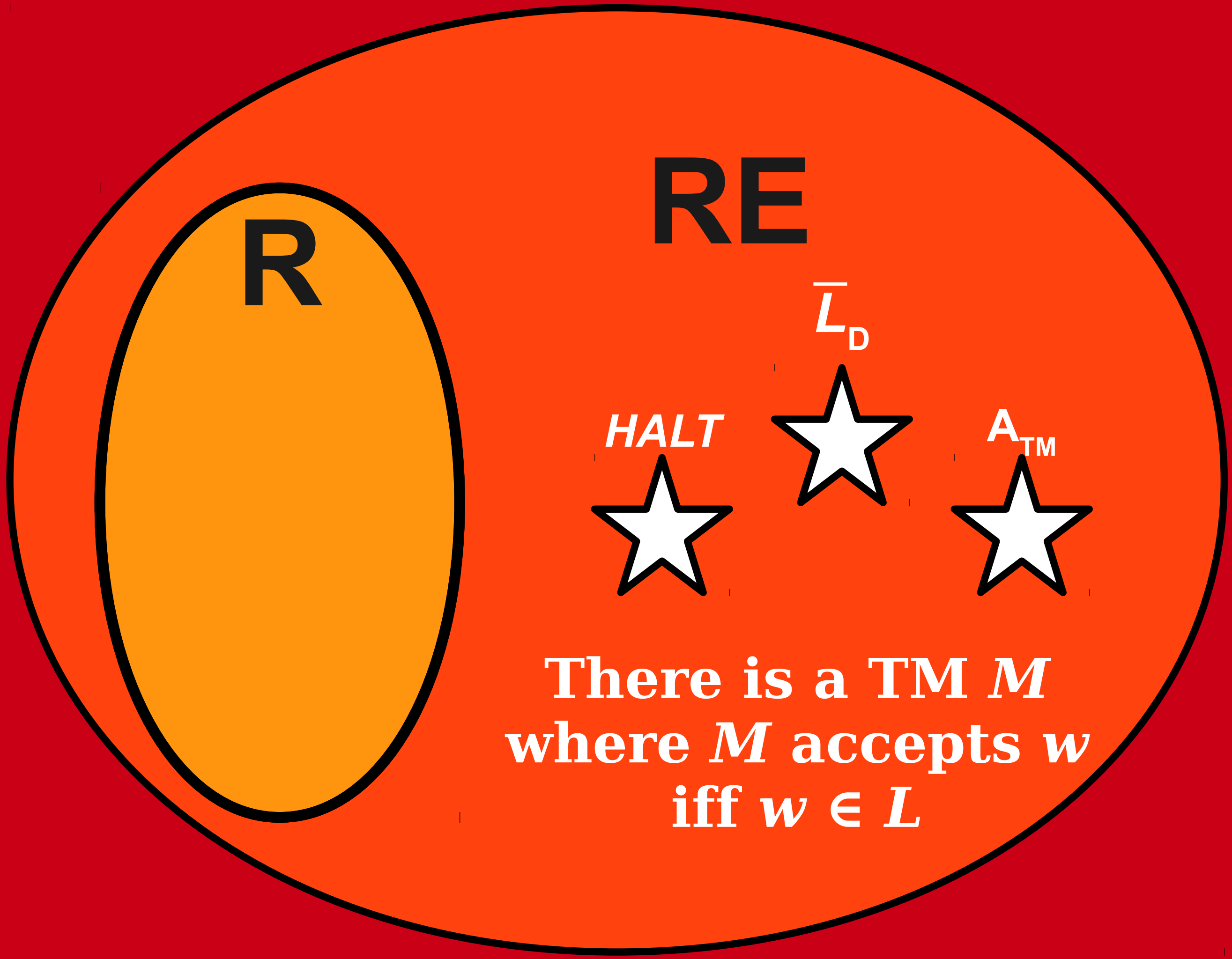
The Limits of Computability

The Limits of Computability

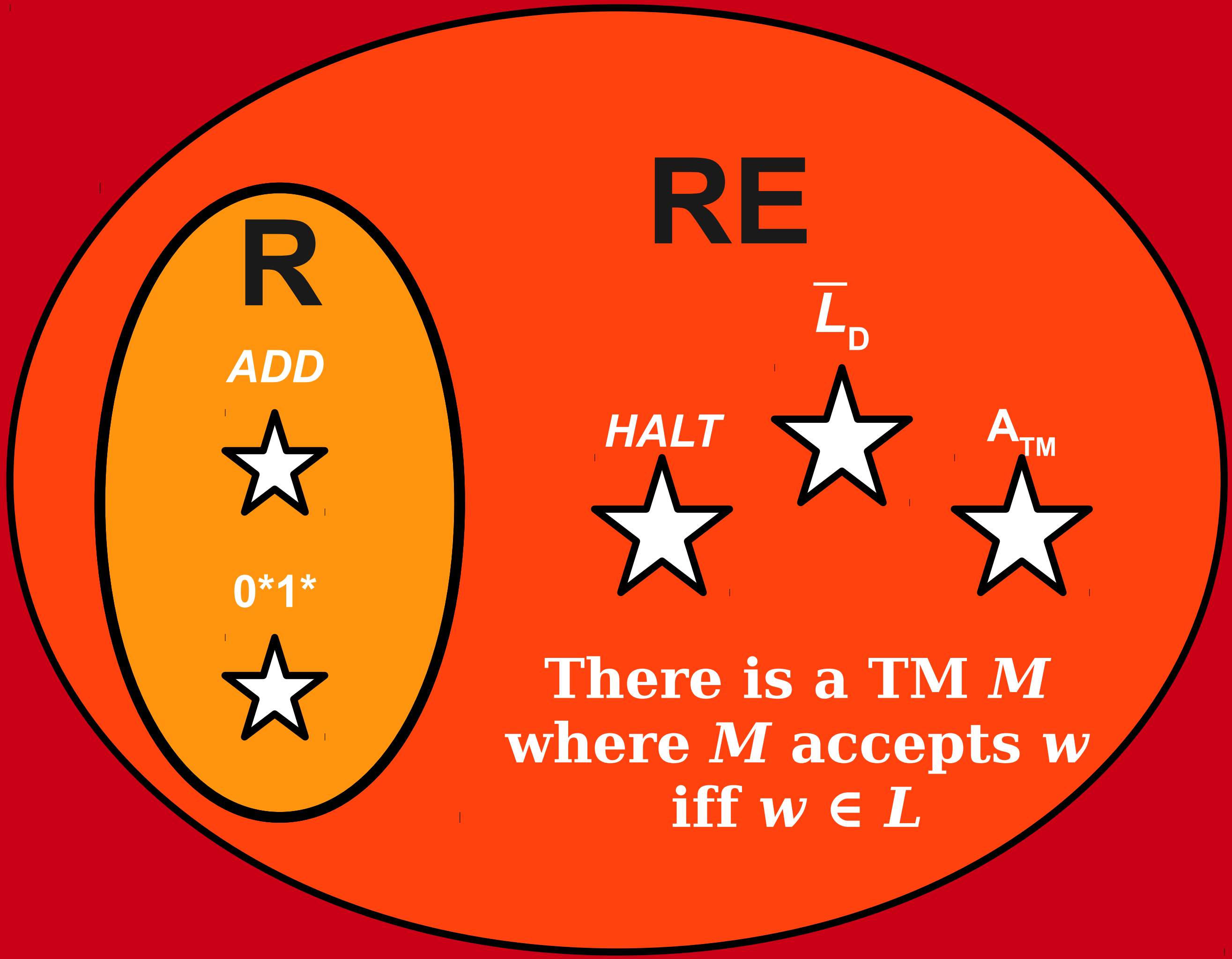
The Limits of Computability



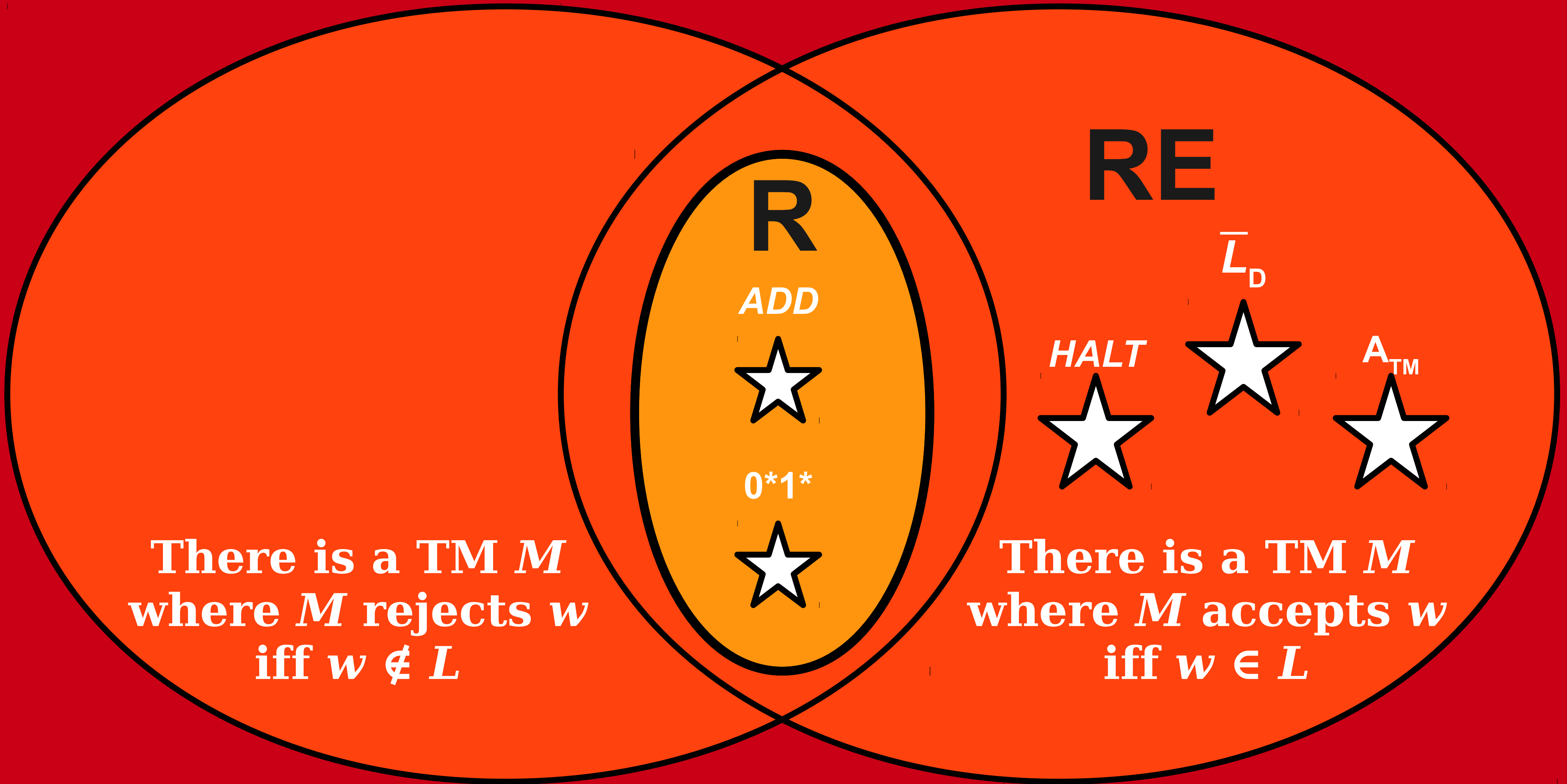
The Limits of Computability



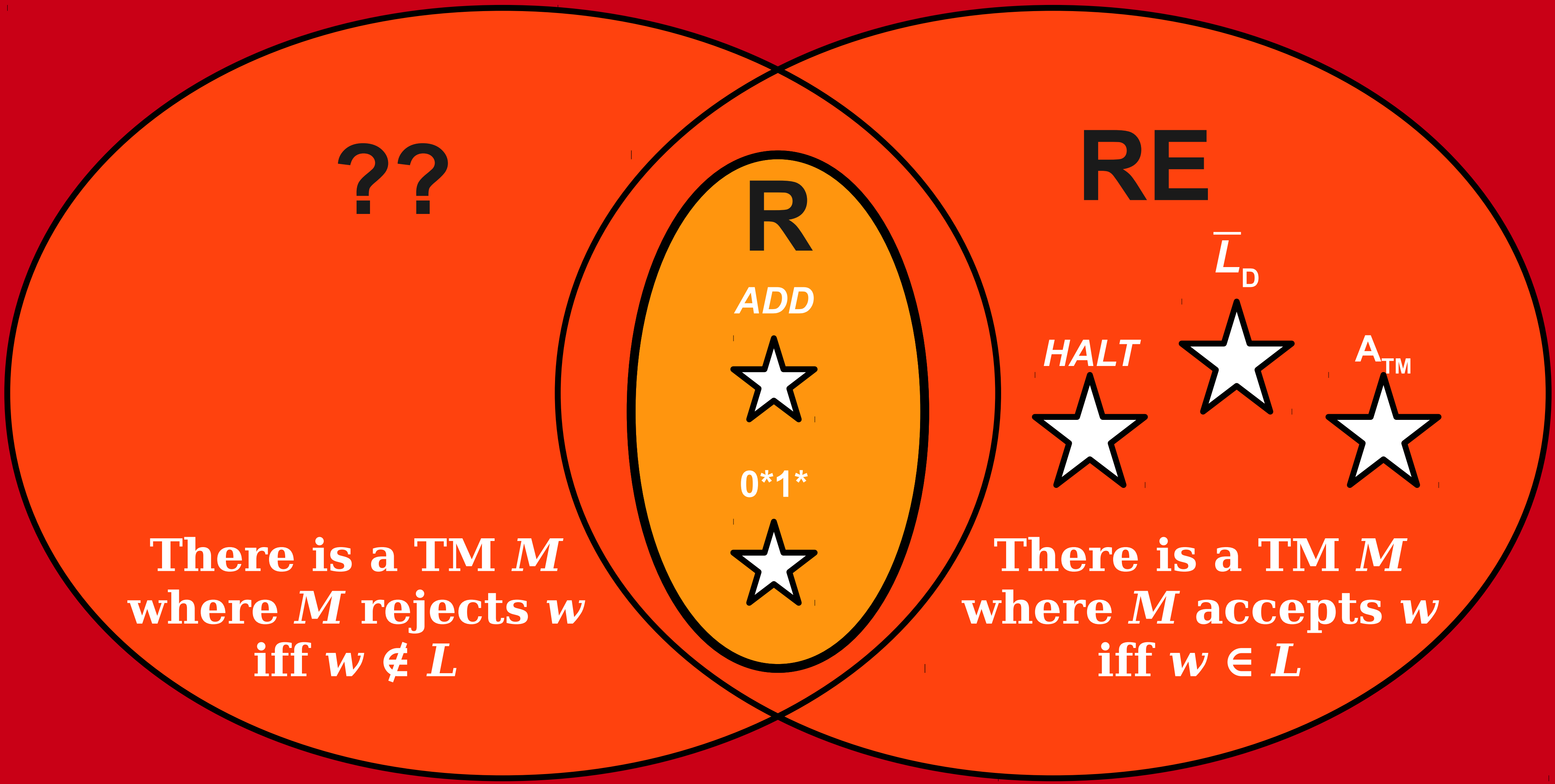
The Limits of Computability



The Limits of Computability



The Limits of Computability



A New Complexity Class

- A language L is in **RE** iff there is a TM M such that
  - if $w \in L$, then M accepts w.
  - if $w \notin L$, then M does not accept w.
- A TM M of this sort is called a *recognizer*, and L is called *recognizable*.
- A language L is in co-**RE** iff there is a TM M such that
  - if $w \in L$, then M does not reject w.
  - if $w \notin L$, then M rejects w.
- A TM M of this sort is called a ***co-recognizer***, and L is called ***co-recognizable***.

RE and co-**RE**

- Intuitively, **RE** consists of all problems where a TM can exhaustively search for **proof** that $w \in L$.
  - If $w \in L$, the TM will find the proof.
  - If $w \notin L$, the TM cannot find a proof.
- Intuitively, co-**RE** consists of all problems where a TM can exhaustively search for a **disproof** that $w \in L$.
  - If $w \in L$, the TM cannot find the disproof.
  - If $w \notin L$, the TM will find the disproof.

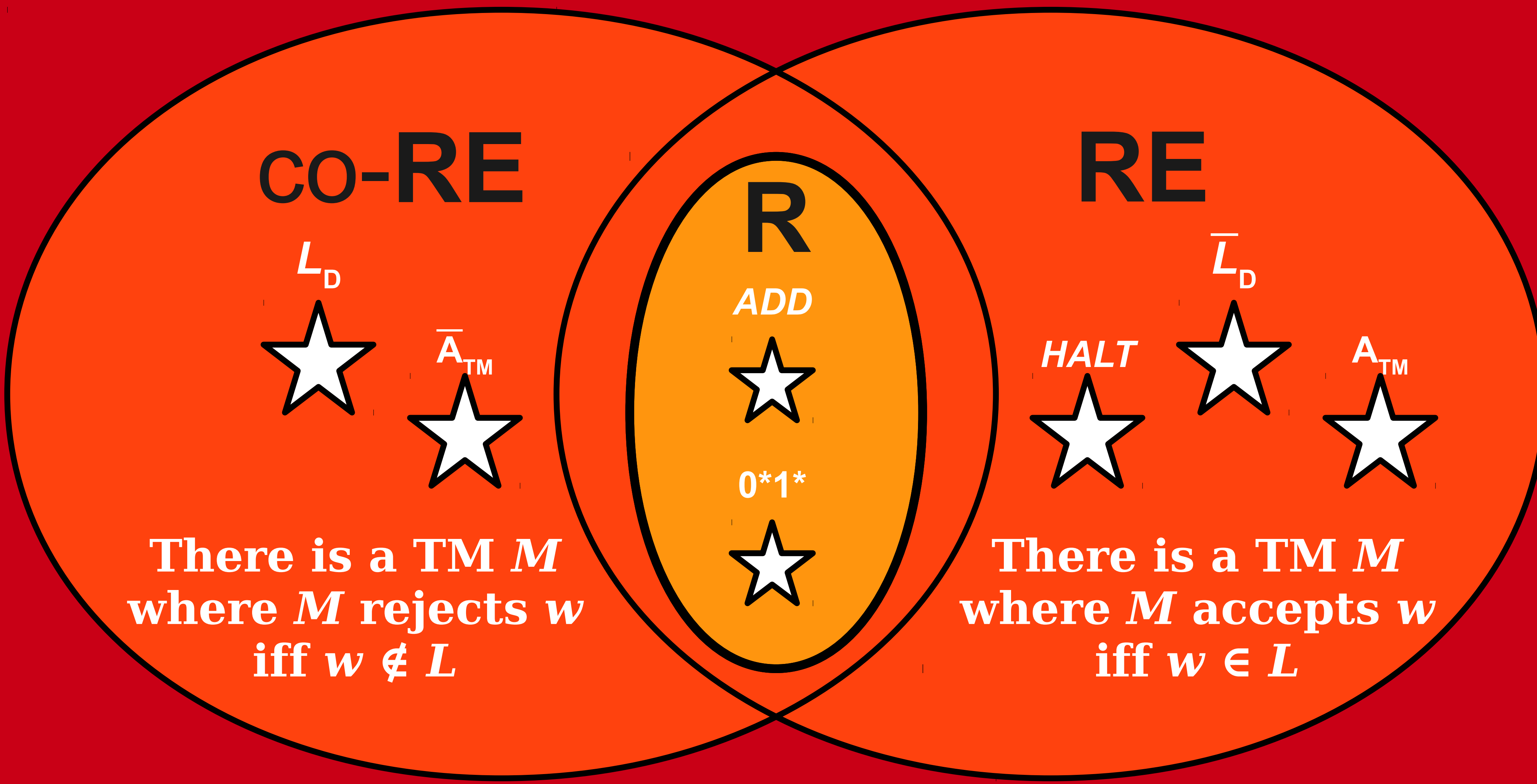
RE and co-**RE** Languages

- A_{TM} is an **RE** language:
  - Simulate the TM *M* on the string *w*.
  - If you find that *M* accepts *w*, accept.
  - If you find that *M* rejects *w*, reject.
  - (If *M* loops, we implicitly loop forever)
- $\overline{A}_{TM}$ is a co-**RE** language:
  - Simulate the TM *M* on the string *w*.
  - If you find that *M* accepts *w*, reject.
  - If you find that *M* rejects *w*, accept.
  - (If *M* loops, we implicitly loop forever)

RE and co-**RE** Languages

- $\overline{L}_{\mathrm{D}}$ is an **RE** language.
  - Simulate M on $\langle M \rangle$.
  - If you find that M accepts $\langle M \rangle$, accept.
  - If you find that M rejects $\langle M \rangle$, reject.
  - (If M loops, we implicitly loop forever)
- L_{D} is a co-**RE** language.
  - Simulate M on $\langle M \rangle$.
  - If you find that M accepts $\langle M \rangle$, reject.
  - If you find that M rejects $\langle M \rangle$, accept.
  - (If M loops, we implicitly loop forever)

The Limits of Computability



RE and co-**RE**

Theorem: $L \in \mathbf{RE}$ iff $\overline{L} \in \text{co-}\mathbf{RE}$.

RE and co-**RE**

Theorem: $L \in \mathbf{RE}$ iff $\overline{L} \in \text{co-}\mathbf{RE}$.

Proof Sketch: Start with a recognizer M for L.

RE and co-**RE**

Theorem: $L \in$ **RE** iff $\overline{L} \in$ co-**RE**.

Proof Sketch: Start with a recognizer M for L. Then, flip its accepting and rejecting states to make machine M'.

RE and co-**RE**

Theorem: $L \in$ **RE** iff $\overline{L} \in$ co-**RE**.

Proof Sketch: Start with a recognizer M for L. Then, flip its accepting and rejecting states to make machine M'. Then

M' rejects w

RE and co-**RE**

Theorem: $L \in$ **RE** iff $\overline{L} \in$ co-**RE**.

Proof Sketch: Start with a recognizer M for L. Then, flip its accepting and rejecting states to make machine M'. Then

M' rejects w
iff M accepts w

RE and co-**RE**

Theorem: $L \in$ **RE** iff $\overline{L} \in$ co-**RE**.

Proof Sketch: Start with a recognizer M for L. Then, flip its accepting and rejecting states to make machine M'. Then

M' rejects w
iff M accepts w
iff $w \in L$

RE and co-**RE**

Theorem: $L \in \mathbf{RE}$ iff $\overline{L} \in$ co-**RE**.

Proof Sketch: Start with a recognizer M for L. Then, flip its accepting and rejecting states to make machine M'. Then

M' rejects w
iff M accepts w
iff $w \in L$
iff $w \notin \overline{L}$.

RE and co-**RE**

Theorem: $L \in \mathbf{RE}$ iff $\overline{L} \in$ co-$\mathbf{RE}$.

Proof Sketch: Start with a recognizer M for L. Then, flip its accepting and rejecting states to make machine M'. Then

M' rejects w
iff M accepts w
iff $w \in L$
iff $w \notin \overline{L}$.

M' does not reject w

RE and co-**RE**

Theorem: $L \in$ **RE** iff $\overline{L} \in$ co-**RE**.

Proof Sketch: Start with a recognizer M for L. Then, flip its accepting and rejecting states to make machine M'. Then

M' rejects w
iff M accepts w
iff $w \in L$
iff $w \notin \overline{L}$.

M' does not reject w
iff M' accepts w or M' loops on w

RE and co-**RE**

Theorem: $L \in$ **RE** iff $\overline{L} \in$ co-**RE**.

Proof Sketch: Start with a recognizer M for L. Then, flip its accepting and rejecting states to make machine M'. Then

M' rejects w
iff M accepts w
iff $w \in L$
iff $w \notin \overline{L}$.

M' does not reject w
iff M' accepts w or M' loops on w
iff M rejects w or M loops on w

RE and co-**RE**

Theorem: $L \in$ **RE** iff $\overline{L} \in$ co-**RE**.

Proof Sketch: Start with a recognizer M for L. Then, flip its accepting and rejecting states to make machine M'. Then

M' rejects w
iff M accepts w
iff $w \in L$
iff $w \notin \overline{L}$.

M' does not reject w
iff M' accepts w or M' loops on w
iff M rejects w or M loops on w
iff $w \notin L$

RE and co-**RE**

Theorem: $L \in$ **RE** iff $\overline{L} \in$ co-**RE**.

Proof Sketch: Start with a recognizer M for L. Then, flip its accepting and rejecting states to make machine M'. Then

M' rejects w
iff M accepts w
iff $w \in L$
iff $w \notin \overline{L}$.

M' does not reject w
iff M' accepts w or M' loops on w
iff M rejects w or M loops on w
iff $w \notin L$
iff $w \in \overline{L}$.

RE and co-**RE**

Theorem: $L \in$ **RE** iff $\overline{L} \in$ co-**RE**.

Proof Sketch: Start with a recognizer M for L. Then, flip its accepting and rejecting states to make machine M'. Then

M' rejects w
iff M accepts w
iff $w \in L$
iff $w \notin \overline{L}$.

M' does not reject w
iff M' accepts w or M' loops on w
iff M rejects w or M loops on w
iff $w \notin L$
iff $w \in \overline{L}$.

RE and co-**RE**

Theorem: $L \in \mathbf{RE}$ iff $\overline{L} \in$ co-**RE**.

Proof Sketch: Start with a recognizer M for L. Then, flip its accepting and rejecting states to make machine M'. Then

M' rejects w
iff M accepts w
iff $w \in L$
iff $w \notin \overline{L}$.

M' does not reject w
iff M' accepts w or M' loops on w
iff M rejects w or M loops on w
iff $w \notin L$
iff $w \in \overline{L}$.

RE and co-**RE**

Theorem: $L \in$ **RE** iff $\overline{L} \in$ co-**RE**.

Proof Sketch: Start with a recognizer M for L. Then, flip its accepting and rejecting states to make machine M'. Then

M' rejects w
iff M accepts w
iff $w \in L$
iff $w \notin \overline{L}$.

M' does not reject w
iff M' accepts w or M' loops on w
iff M rejects w or M loops on w
iff $w \notin L$
iff $w \in \overline{L}$.

The same approach works if we flip the accept and reject states of a co-recognizer for $\overline{L}$.

RE and co-**RE**

Theorem: $L \in$ **RE** iff $\overline{L} \in$ co-**RE**.

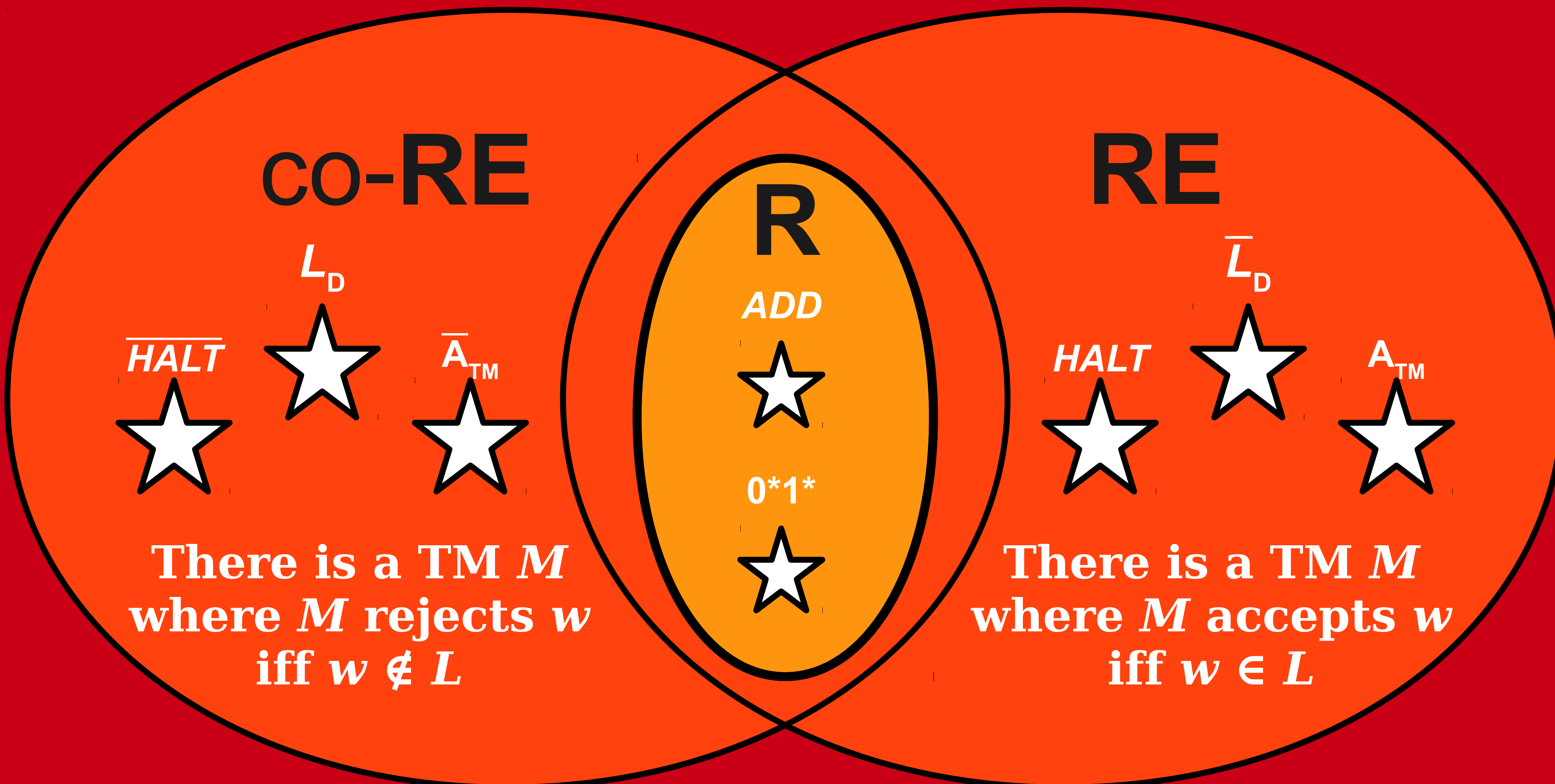
Proof Sketch: Start with a recognizer M for L. Then, flip its accepting and rejecting states to make machine M'. Then

M' rejects w
iff M accepts w
iff $w \in L$
iff $w \notin \overline{L}$.

M' does not reject w
iff M' accepts w or M' loops on w
iff M rejects w or M loops on w
iff $w \notin L$
iff $w \in \overline{L}$.

The same approach works if we flip the accept and reject states of a co-recognizer for $\overline{L}$. ■

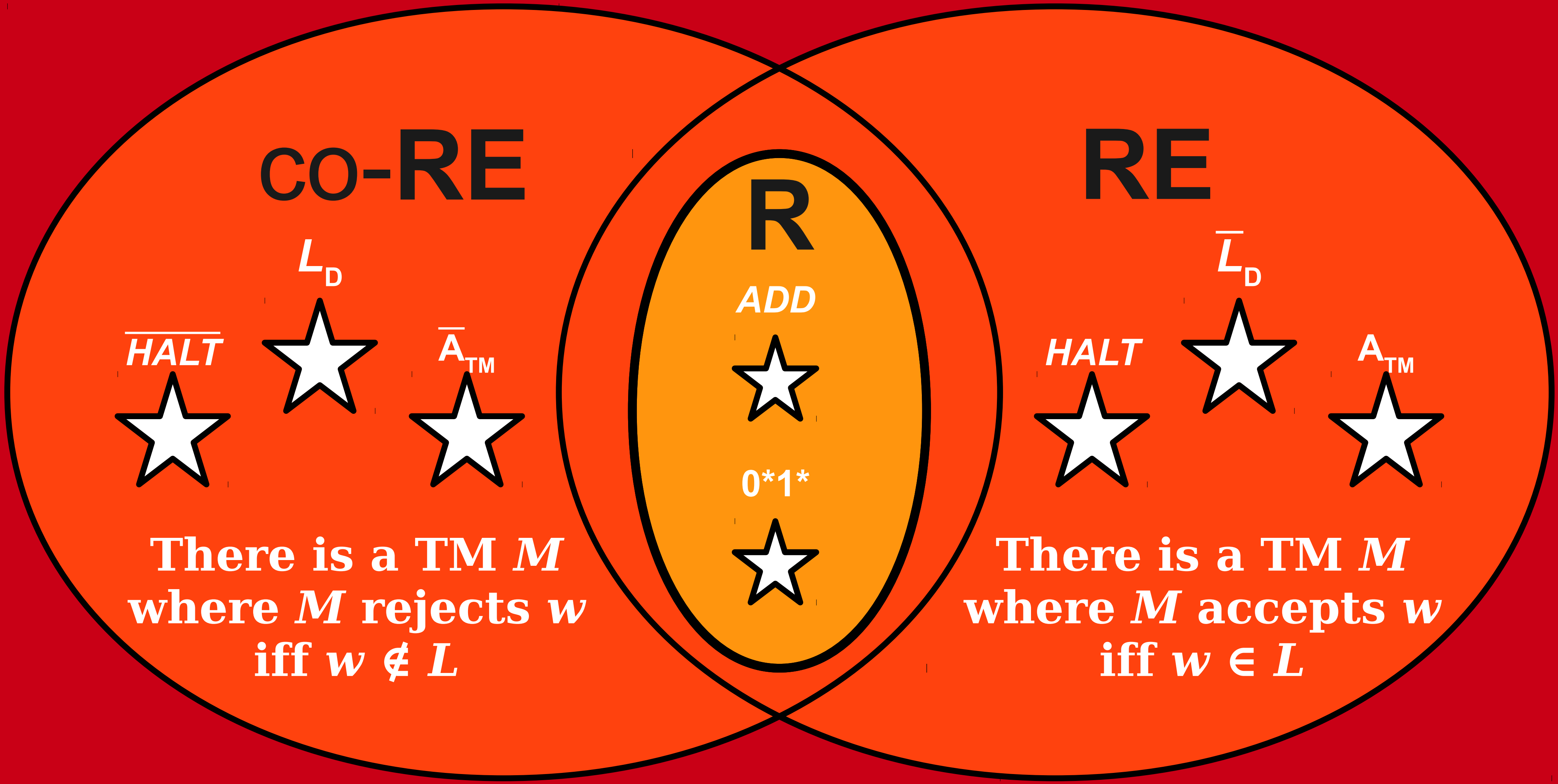
The Limits of Computability



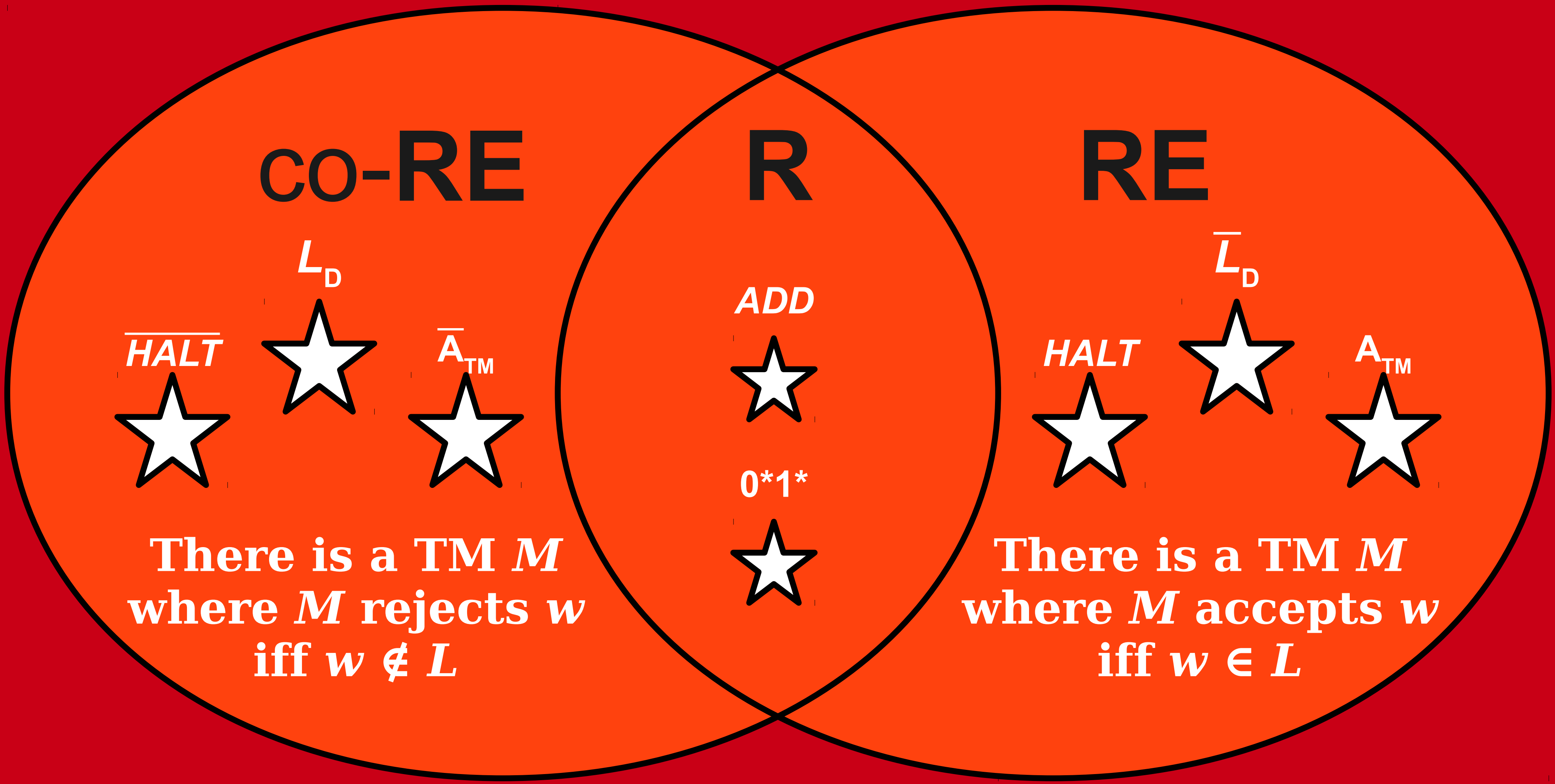
R, **RE**, and co-**RE**

- Every language in **R** is in both **RE** and co-**RE**.
- Why?
  - A decider for L accepts all $w \in L$ and rejects all $w \notin L$.
- In other words, $\mathbf{R} \subseteq \mathbf{RE} \cap \text{co-}\mathbf{RE}$.
- **Question:** Does $\mathbf{R} = \mathbf{RE} \cap \text{co-}\mathbf{RE}$?

Which Picture is Correct?



Which Picture is Correct?



R, **RE**, and co-**RE**

- ***Theorem:*** If $L \in \mathbf{RE}$ and $L \in \text{co-}\mathbf{RE}$, then $L \in \mathbf{R}$.

$\mathbf{R}$, $\mathbf{RE}$, and co-$\mathbf{RE}$

- ***Theorem:*** If $L \in \mathbf{RE}$ and $L \in \text{co-}\mathbf{RE}$, then $L \in \mathbf{R}$.
- ***Proof sketch:*** Since $L \in \mathbf{RE}$, there is a recognizer M for it.

R, **RE**, and co-**RE**

- ***Theorem:*** If $L \in$ **RE** and $L \in$ co-**RE**, then $L \in$ **R**.
- ***Proof sketch:*** Since $L \in$ **RE**, there is a recognizer M for it. Since $\overline{L} \in$ co-**RE**, there is a co-recognizer $\overline{M}$ for it.

R, **RE**, and co-**RE**

- ***Theorem:*** If $L \in \mathbf{RE}$ and $L \in \text{co-}\mathbf{RE}$, then $L \in \mathbf{R}$.
- ***Proof sketch:*** Since $L \in \mathbf{RE}$, there is a recognizer M for it. Since $L \in \text{co-}\mathbf{RE}$, there is a co-recognizer $\overline{M}$ for it.

  This TM D is a decider for L:

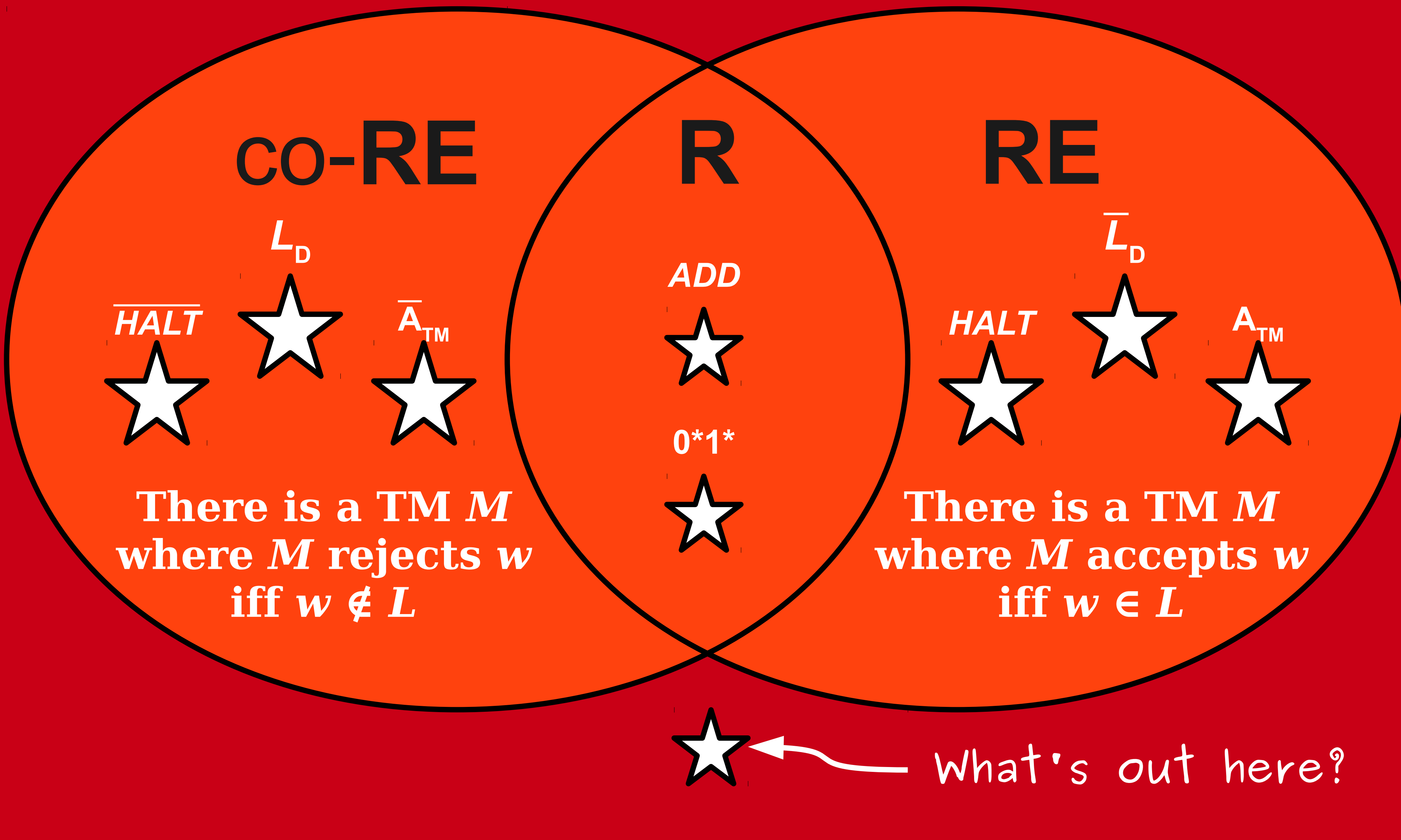
R, **RE**, and co-**RE**

- ***Theorem:*** If $L \in \mathbf{RE}$ and $L \in$ co-$\mathbf{RE}$, then $L \in \mathbf{R}$.
- ***Proof sketch:*** Since $L \in \mathbf{RE}$, there is a recognizer M for it. Since $L \in$ co-$\mathbf{RE}$, there is a co-recognizer $\overline{M}$ for it.

  This TM D is a decider for L:

> D = "On input w:
> Run M on w and $\overline{M}$ on w in parallel.
> If M accepts w, accept.
> If $\overline{M}$ rejects w, reject.

The Limits of Computability



Time-Out For Announcements!

Friday Four Square!

Today at 4:15PM outside Gates

Two Handouts Online

- **24: Additional Proofs on TMs**
  - See alternate proofs of why various languages are or are not **R**, **RE**, or co-**RE**.
- **25: Extra Practice Problems**
  - By popular demand, extra questions on topics you'd like some more practice with!
  - Solutions released Monday.

Picking up Problem Sets

- If you pick up problem sets from the filing cabinet,

  please put all other papers back into the filing cabinet when you're done!

- If you don't:
  - they get mixed with problem sets from other classes and lost,
  - it causes a fire hazard, and
  - I get flak from the building managers about making a mess.

Your Questions

“Can you recommend software for designing and / or simulating Turing machines?”

http://www.jflap.org/

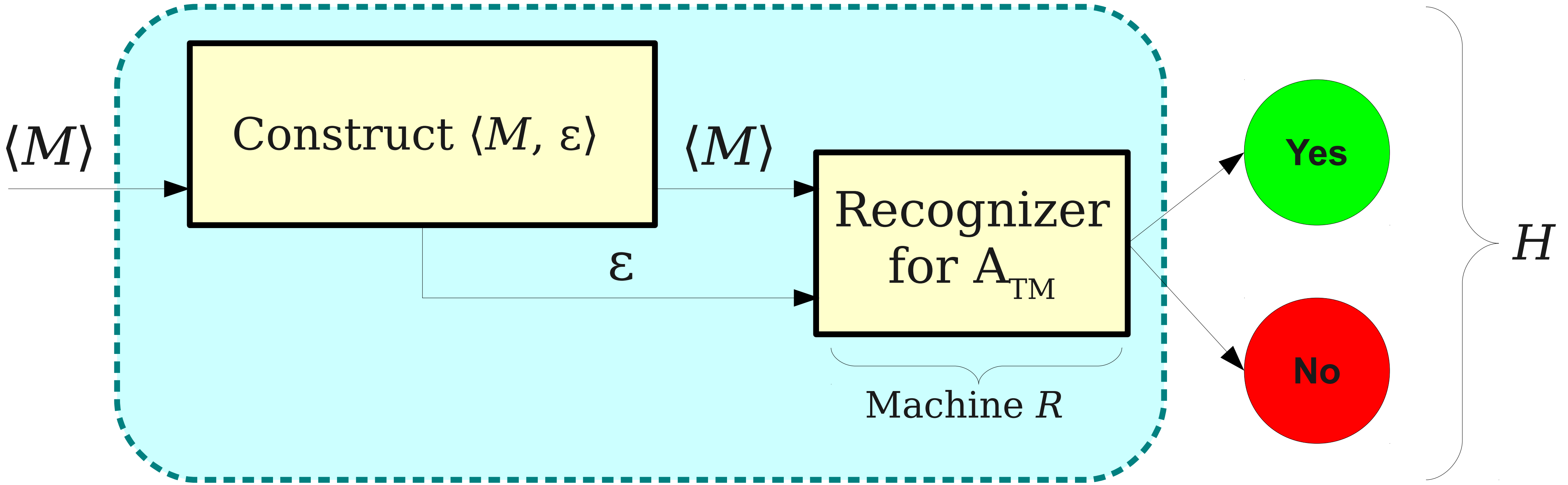
“Is there a difference between when a TM “runs” another TM as a subroutine vs. when it “simulates running” another TM?”

“Sometime my brain is stuck and I make silly and stupid mistakes [...]. What [do] you do when you are stuck on a problem?”

Back to CS103!

A Repeating Pattern

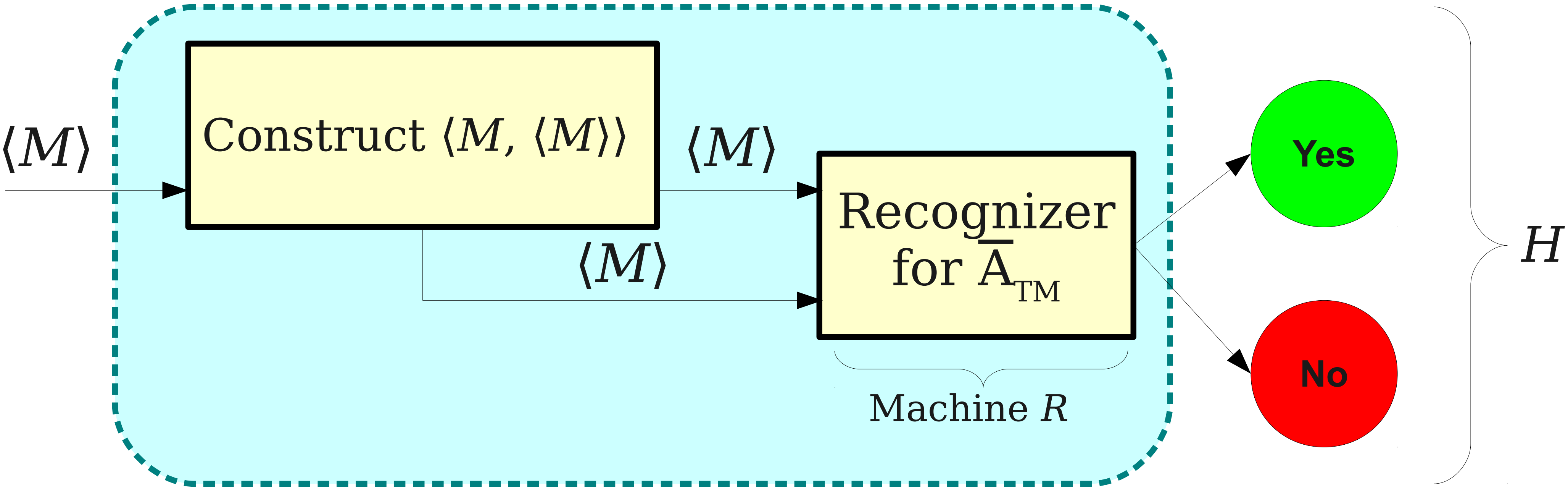
$L = \{ \langle M \rangle \mid M$ is a TM that accepts $\varepsilon \}$



H = "On input $\langle M \rangle$:

- Construct the string $\langle M, \varepsilon \rangle$.
- Run R on $\langle M, \varepsilon \rangle$.
- If R accepts $\langle M, \varepsilon \rangle$, then H accepts $\langle M, \varepsilon \rangle$.
- If R rejects $\langle M, \varepsilon \rangle$, then H rejects $\langle M, \varepsilon \rangle$."

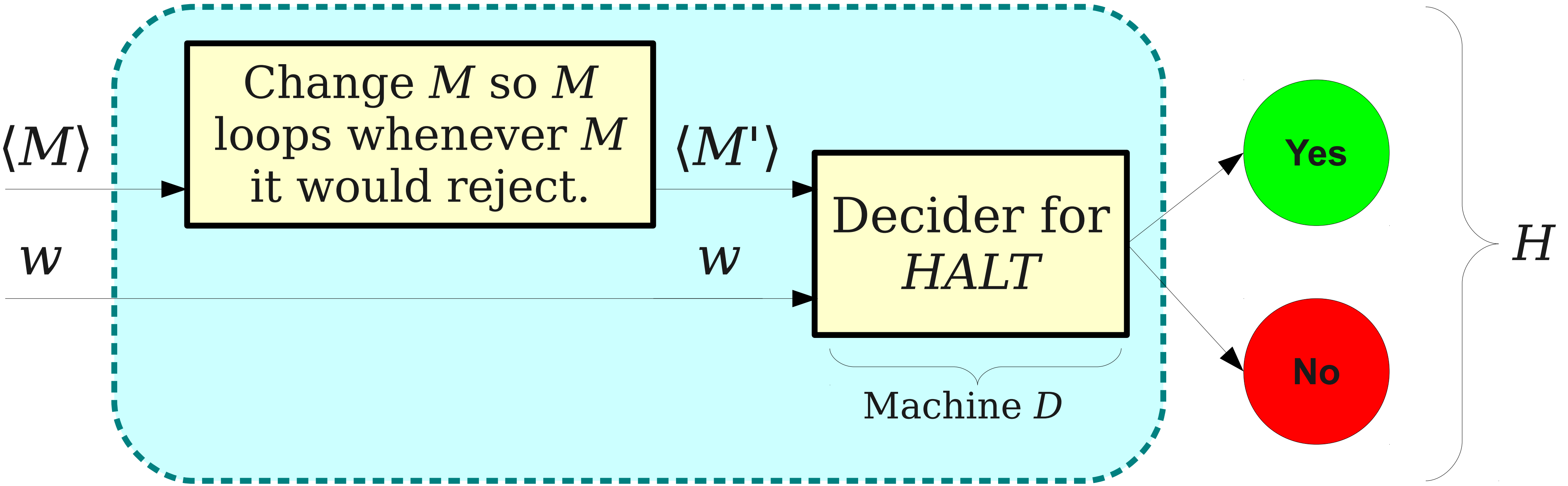
From $\overline{\text{A}}_{\text{TM}}$ to L_{D}



H = "On input $\langle M \rangle$:

- Construct the string $\langle M, \langle M \rangle \rangle$.
- Run R on $\langle M, \langle M \rangle \rangle$.
- If R accepts $\langle M, \langle M \rangle \rangle$, then H accepts $\langle M, \langle M \rangle \rangle$.
- If R rejects $\langle M, \langle M \rangle \rangle$, then H rejects $\langle M, \langle M \rangle \rangle$."

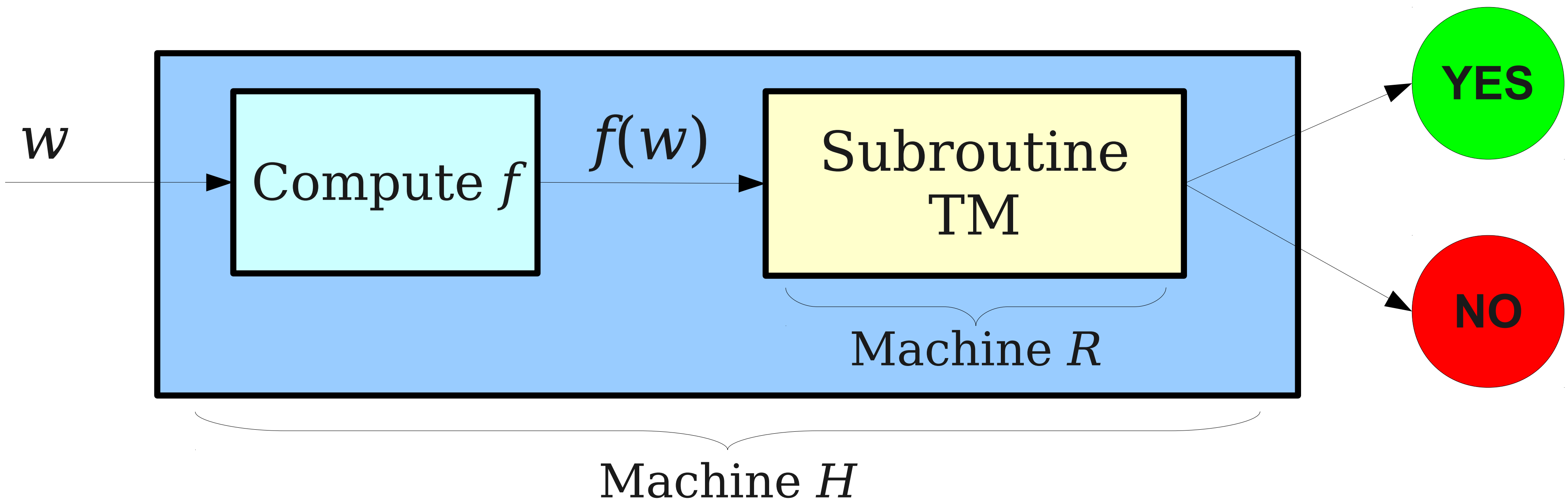
From *HALT* to A_{TM}



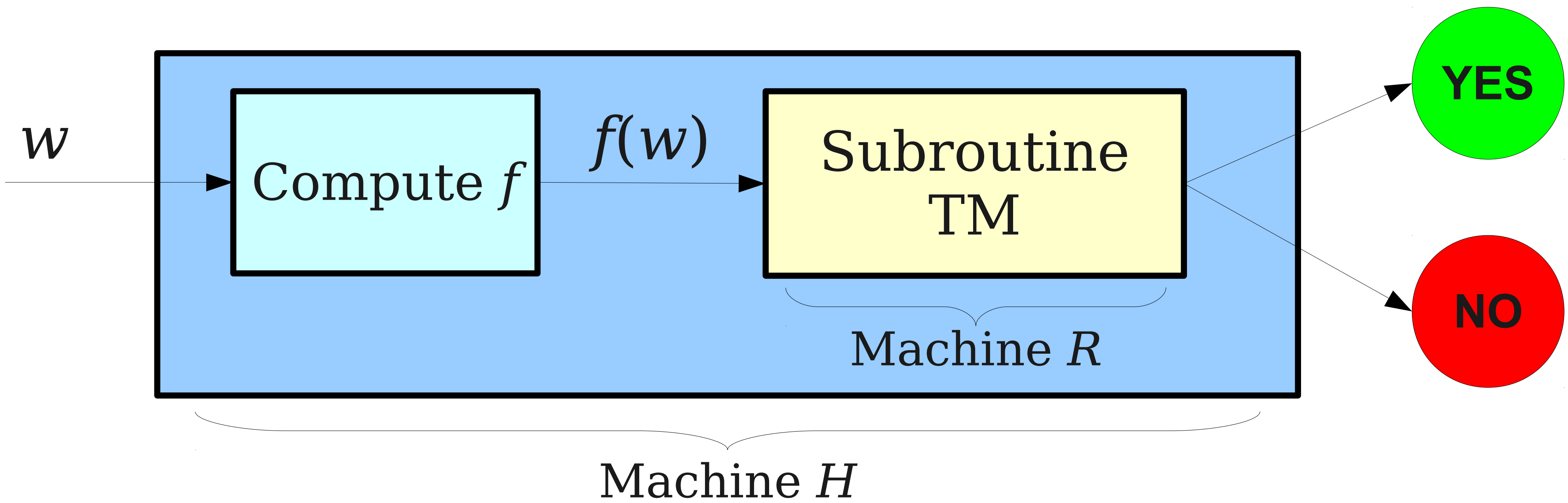
H = “On input ⟨*M*, *w*⟩:

- Build *M* into *M*' so *M*' loops when *M* rejects.
- Run *D* on ⟨*M*', *w*⟩.
- If *D* accepts ⟨*M*', *w*⟩, then *H* accepts ⟨*M*, *w*⟩.
- If *D* rejects ⟨*M*', *w*⟩, then *H* rejects ⟨*M*, *w*⟩.”

The General Pattern



The General Pattern

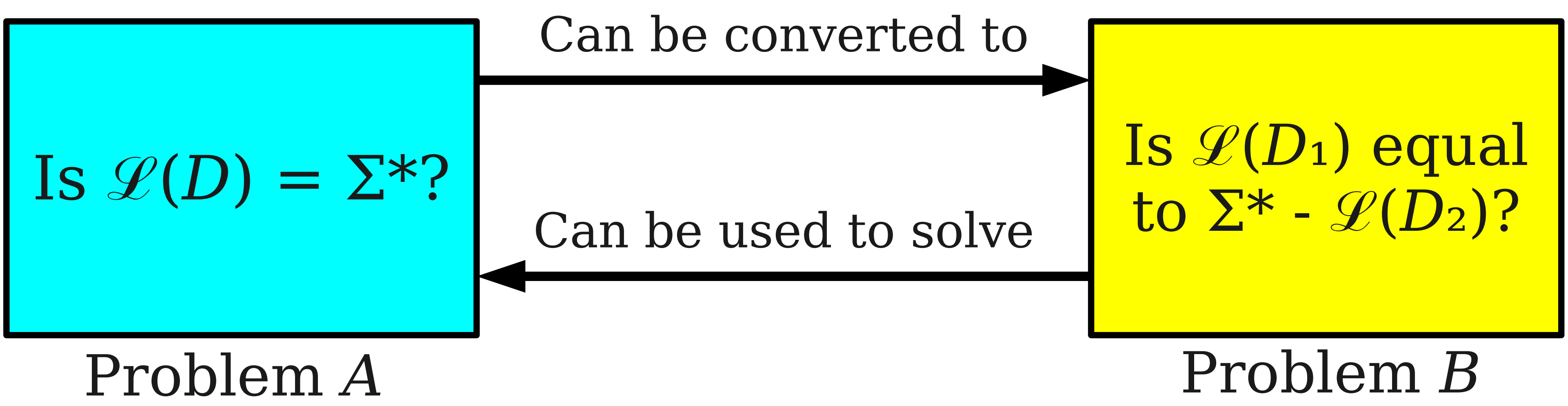


H = "On input w:

- Transform the input w into $f(w)$.
- Run machine R on $f(w)$.
- If R accepts $f(w)$, then H accepts w.
- If R rejects $f(w)$, then H rejects w."

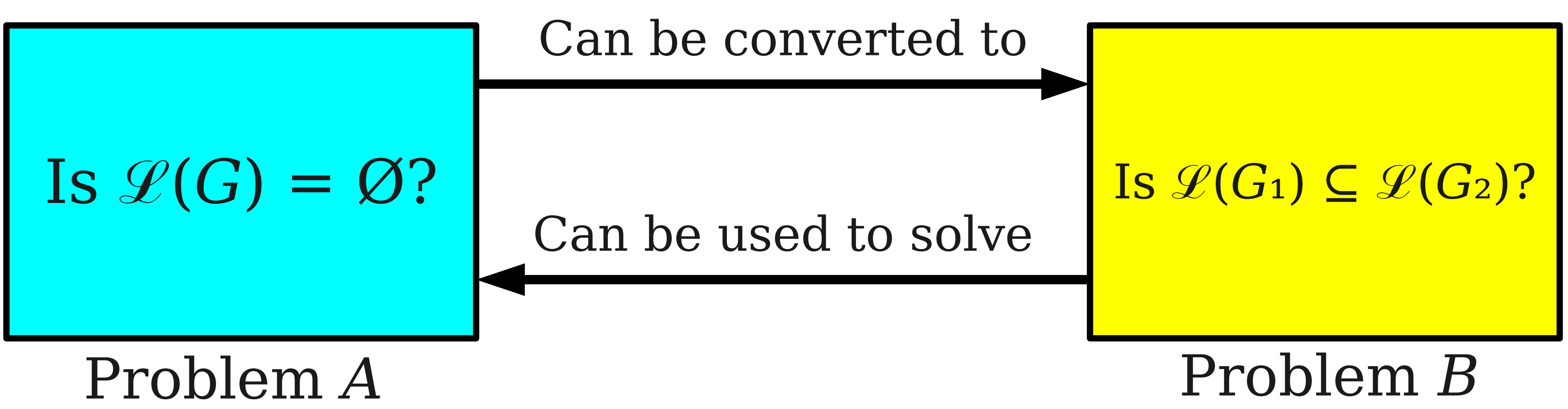
Reductions

- Intuitively, problem A **reduces** to problem B iff a solver for B can be used to solve problem A.



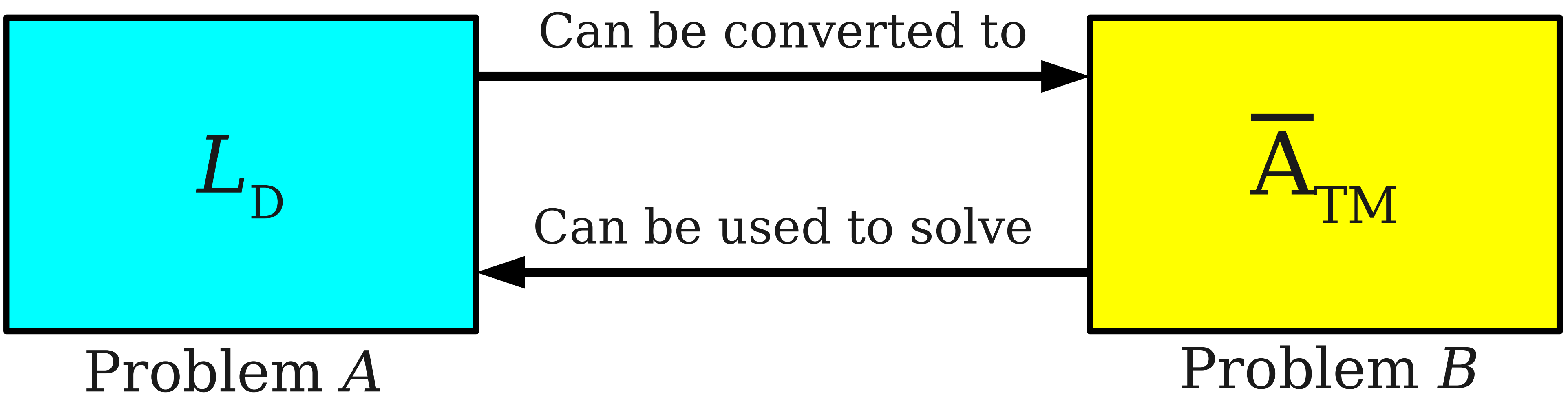
Reductions

- Intuitively, problem A **reduces** to problem B iff a solver for B can be used to solve problem A.



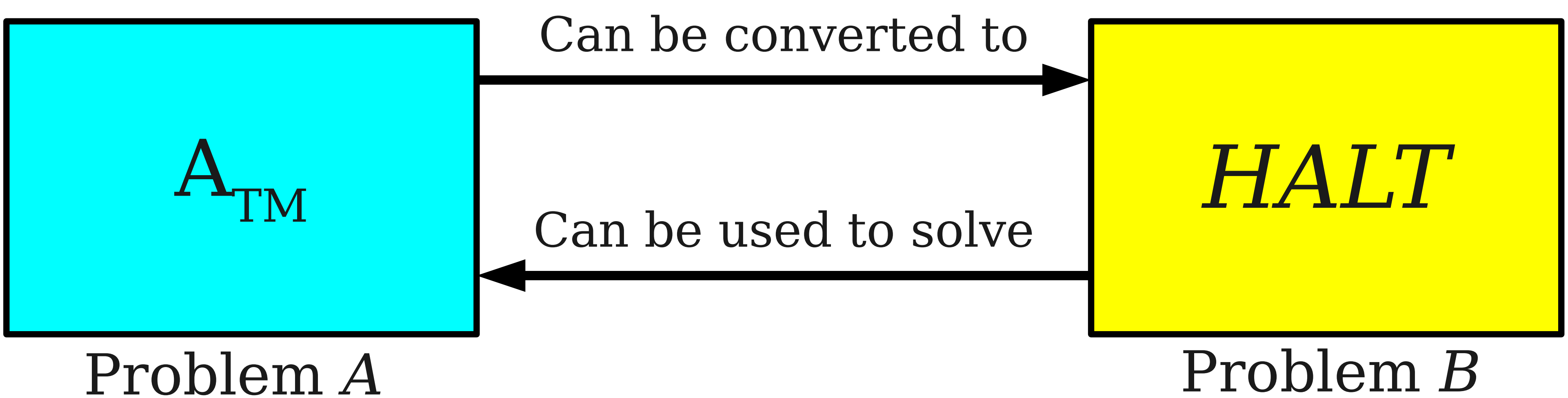
Reductions

- Intuitively, problem A **reduces** to problem B iff a solver for B can be used to solve problem A.



Reductions

- Intuitively, problem *A* **reduces** to problem *B* iff a solver for *B* can be used to solve problem *A*.



Reductions

- Intuitively, problem A **reduces** to problem B iff a solver for B can be used to solve problem A.
- Reductions can be used to show certain problems are "solvable:"

If A reduces to B and B is "solvable," then A is "solvable."

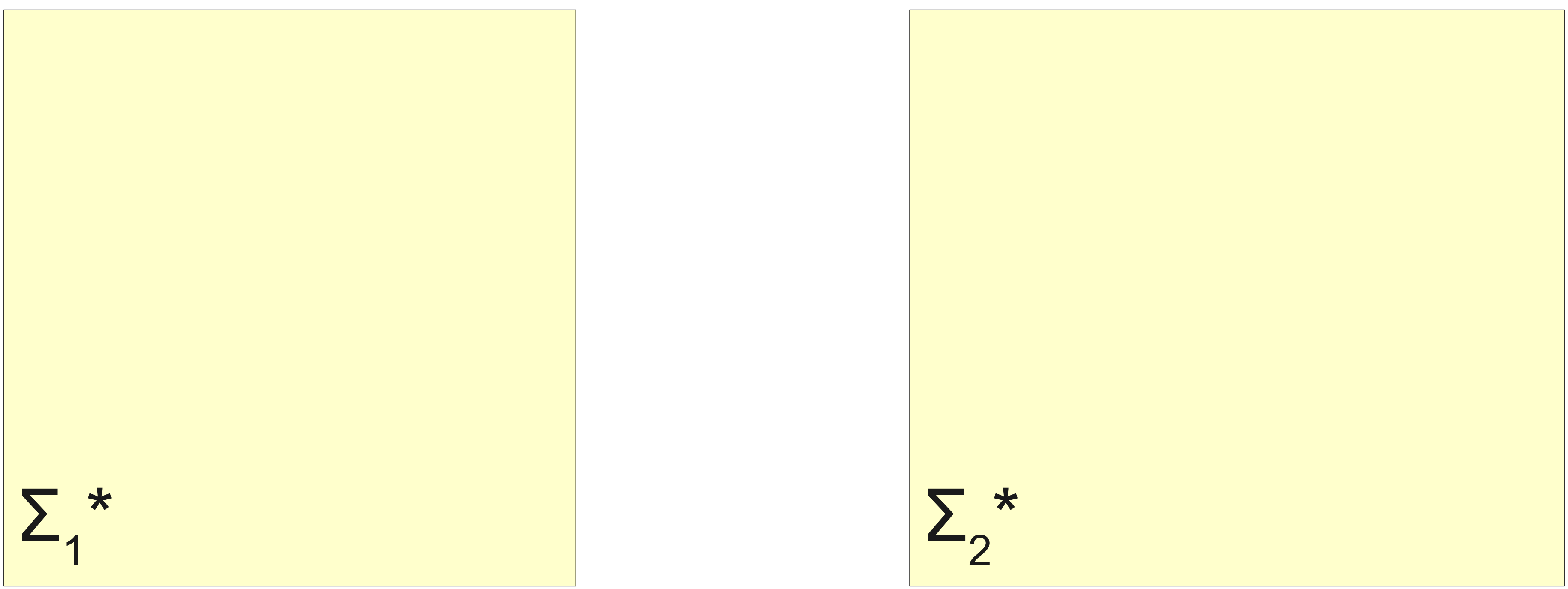
Formalizing Reductions

- In order to make the previous intuition more rigorous, we need to formally define reductions.
- There are many ways to do this; we'll explore two:
  - **Mapping reducibility** (today / Monday), and
  - **Polynomial-time reducibility** (next week).

Defining Reductions

- A **reduction** from A to B is a function $f : \Sigma_1^* \to \Sigma_2^*$ such that

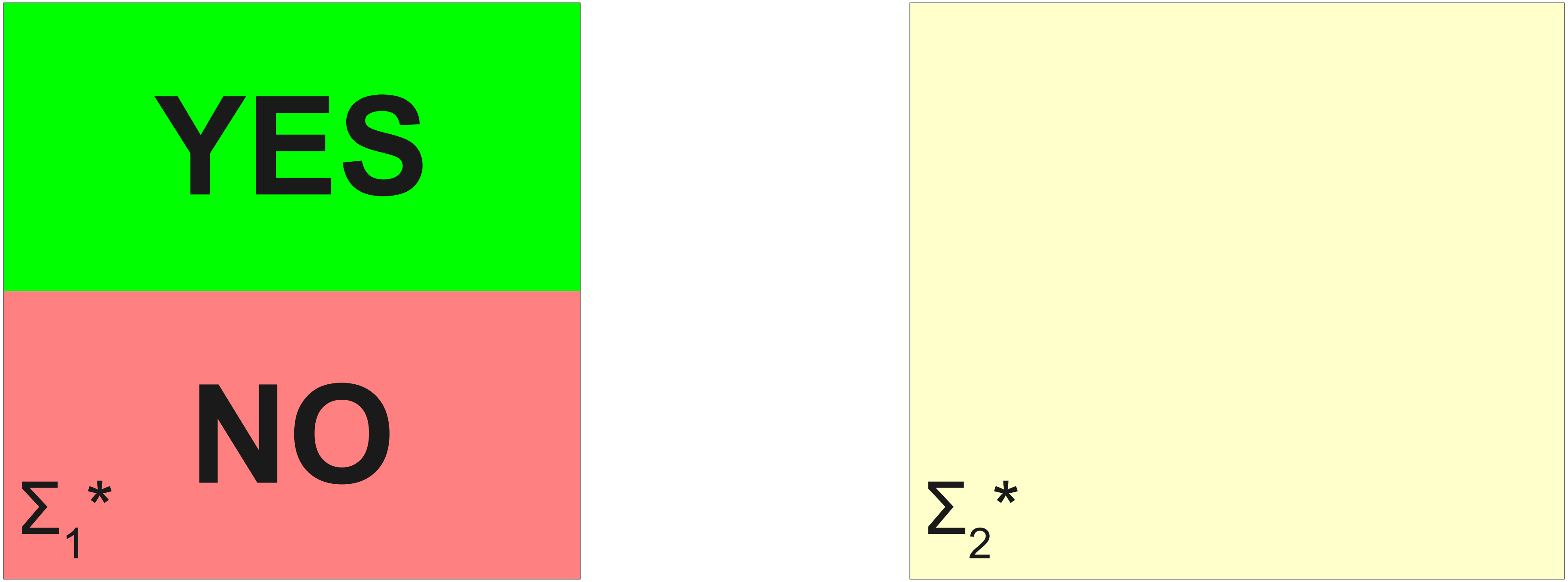
For any $w \in \Sigma_1^*$, $w \in A$ iff $f(w) \in B$



Defining Reductions

- A **reduction** from A to B is a function $f : \Sigma_1^* \to \Sigma_2^*$ such that

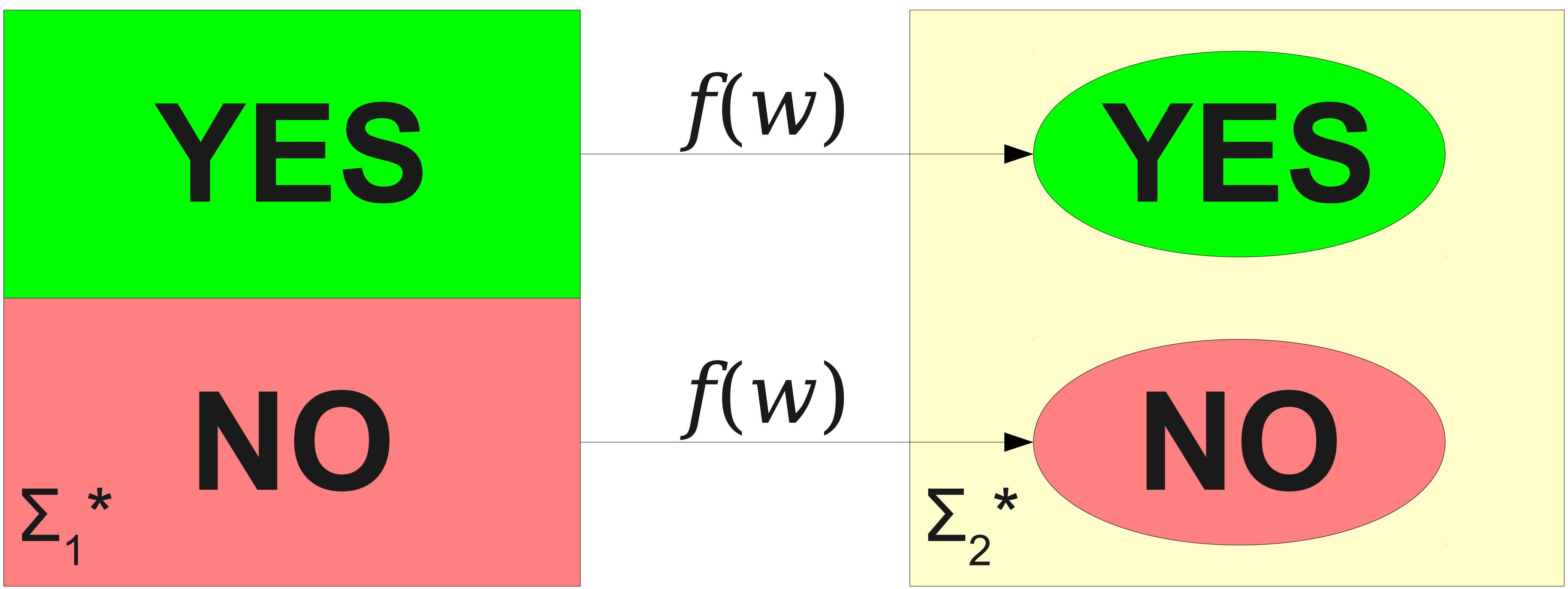
For any $w \in \Sigma_1^*$, $w \in A$ iff $f(w) \in B$



Defining Reductions

- A **reduction** from A to B is a function $f : \Sigma_1^* \to \Sigma_2^*$ such that

For any $w \in \Sigma_1^*$, $w \in A$ iff $f(w) \in B$



Defining Reductions

- A **reduction** from A to B is a function $f : \Sigma_1^* \to \Sigma_2^*$ such that

For any $w \in \Sigma_1^*$, $w \in A$ iff $f(w) \in B$

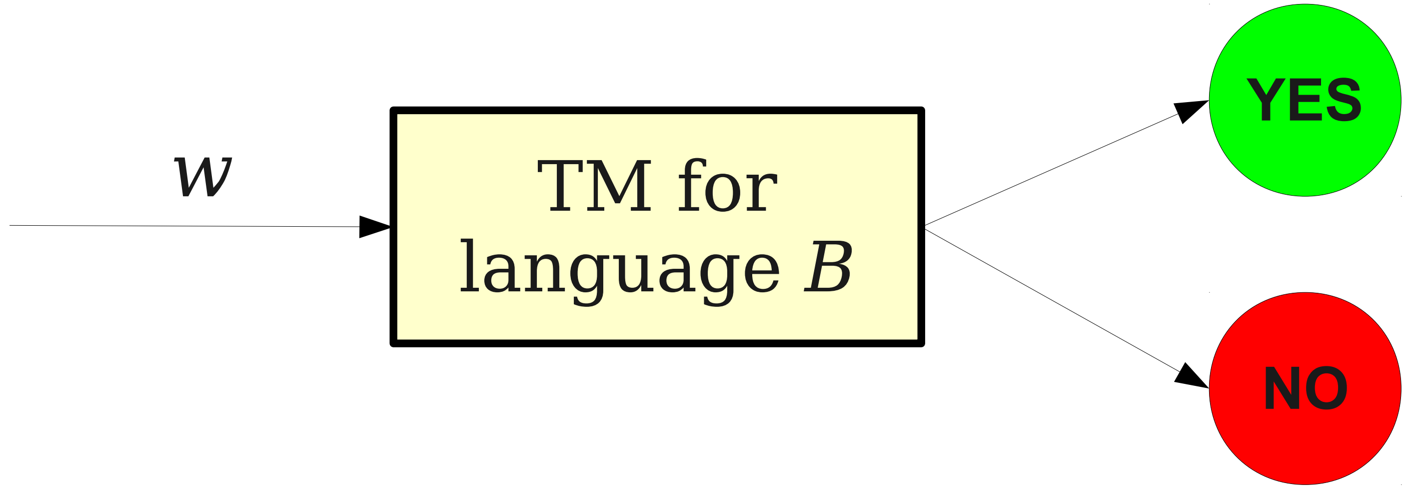
- Every $w \in A$ maps to some $f(w) \in B$.
- Every $w \notin A$ maps to some $f(w) \notin B$.
- f does not have to be injective or surjective.

Why Reductions Matter

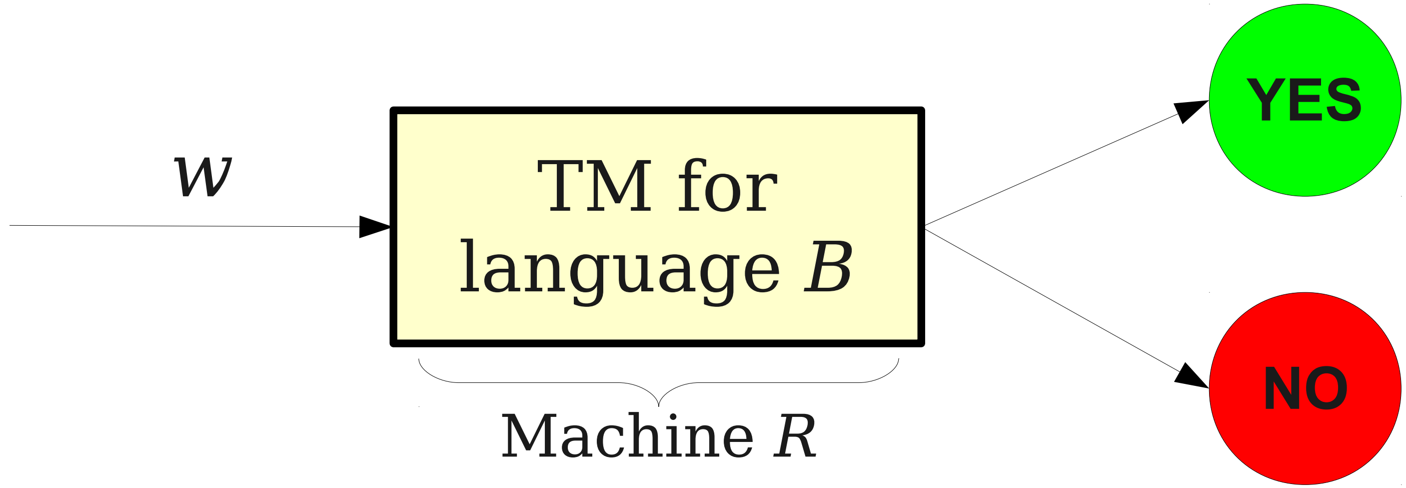
- If language A reduces to language B, we can use a recognizer / co-recognizer / decider for B to recognize / co-recognize / decide problem A.
  - (There's a slight catch – we'll talk about this in a second).
- How is this possible?

$$w \in A \quad \text{iff} \quad f(w) \in B$$

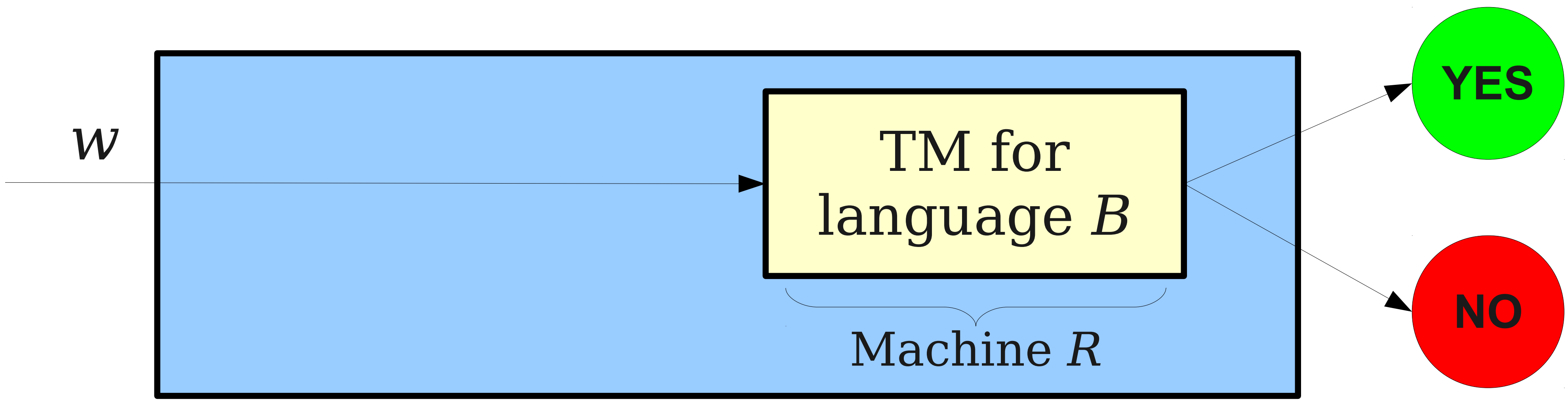
$$w \in A \quad \text{iff} \quad f(w) \in B$$



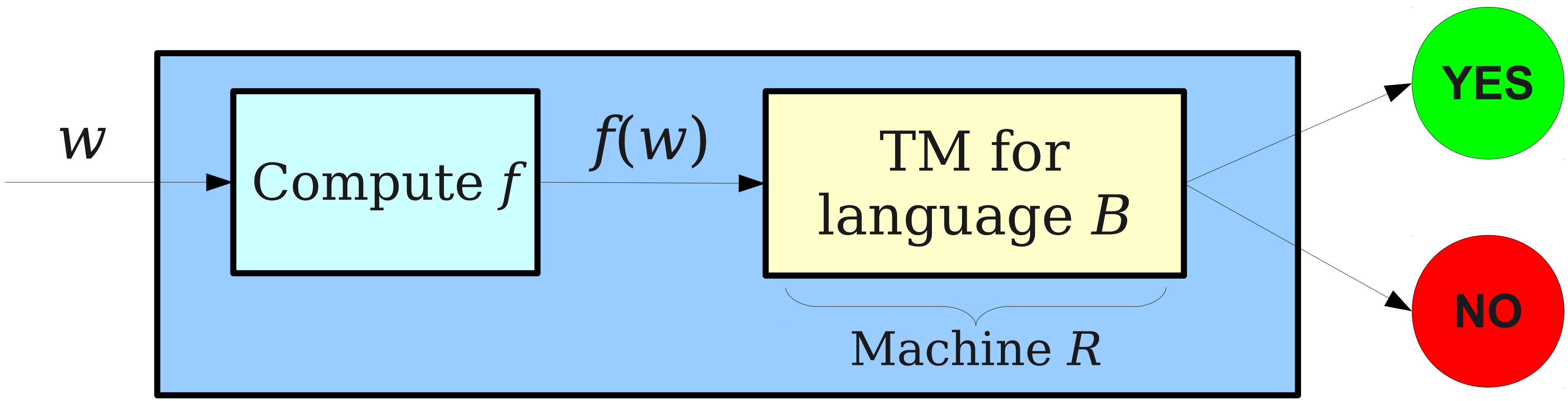
$$w \in A \quad \text{iff} \quad f(w) \in B$$



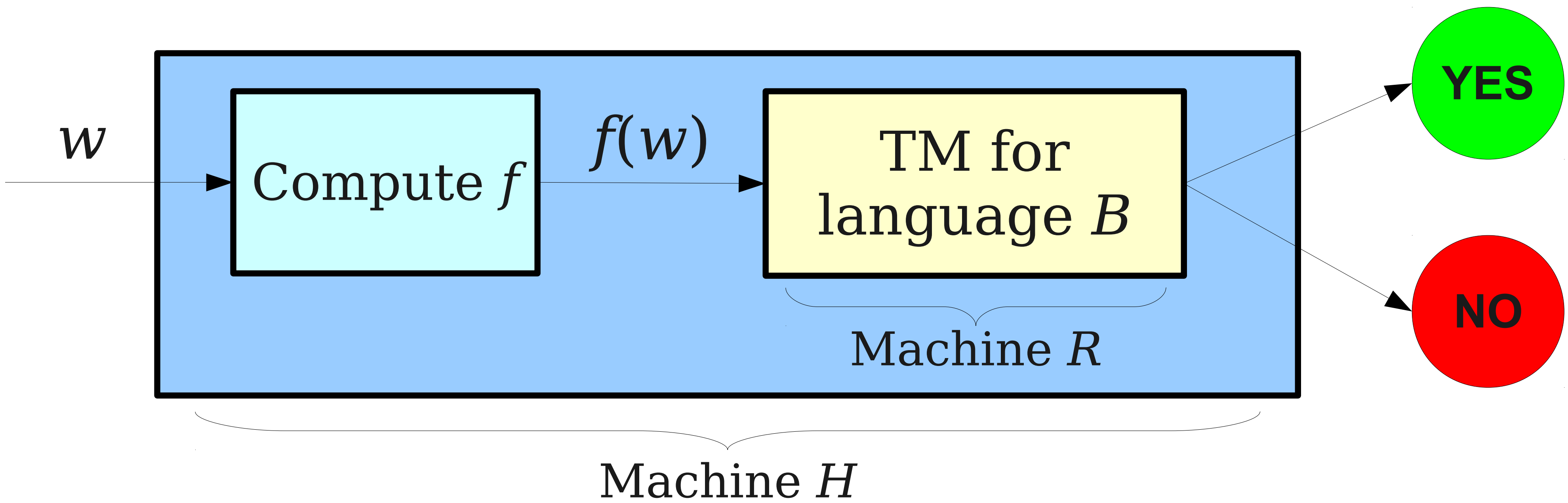
$$w \in A \quad \text{iff} \quad f(w) \in B$$



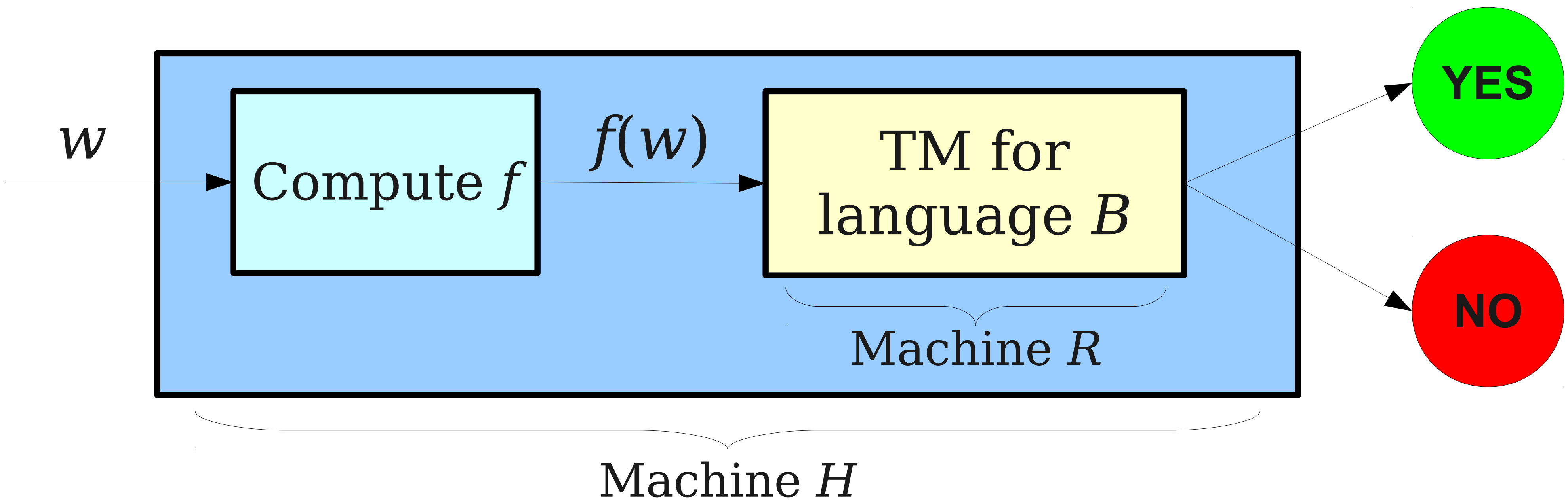
$$w \in A \quad \text{iff} \quad f(w) \in B$$



$$w \in A \quad \text{iff} \quad f(w) \in B$$



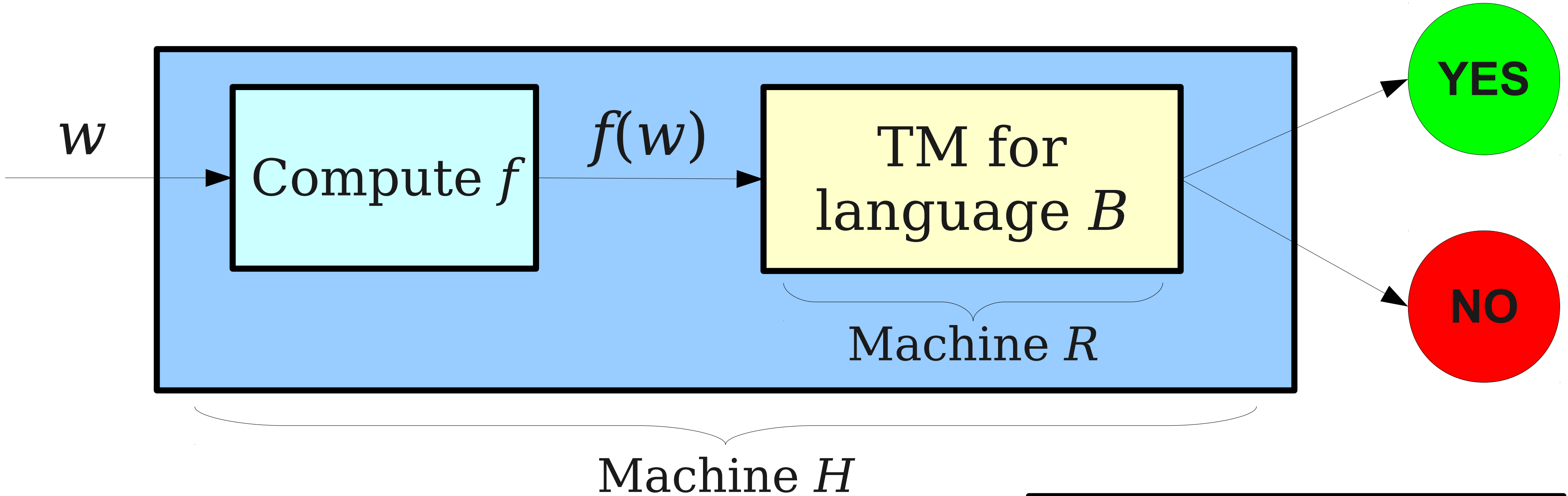
$$w \in A \quad \text{iff} \quad f(w) \in B$$



H = "On input w:

- Transform the input w into $f(w)$.
- Run machine R on $f(w)$.
- If R accepts $f(w)$, then H accepts w.
- If R rejects $f(w)$, then H rejects w."

$$\boldsymbol{w \in A \quad \text{iff} \quad f(w) \in B}$$

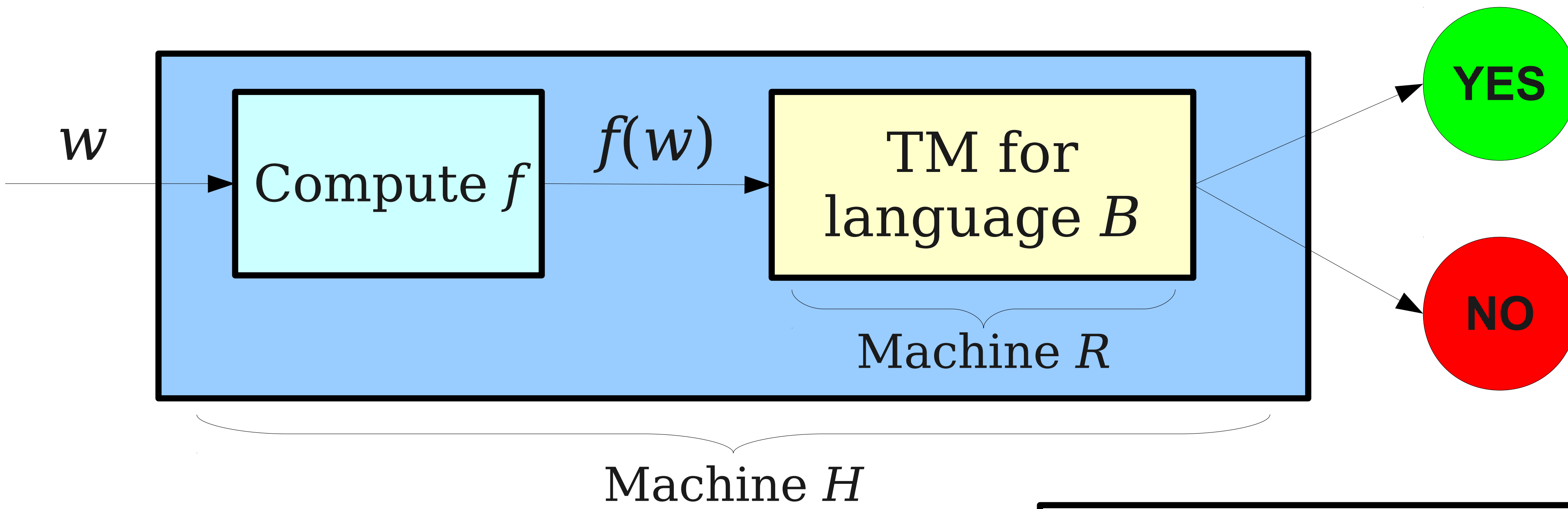


H = "On input w:

- Transform the input w into $f(w)$.
- Run machine R on $f(w)$.
- If R accepts $f(w)$, then H accepts w.
- If R rejects $f(w)$, then H rejects w."

H* accepts *w

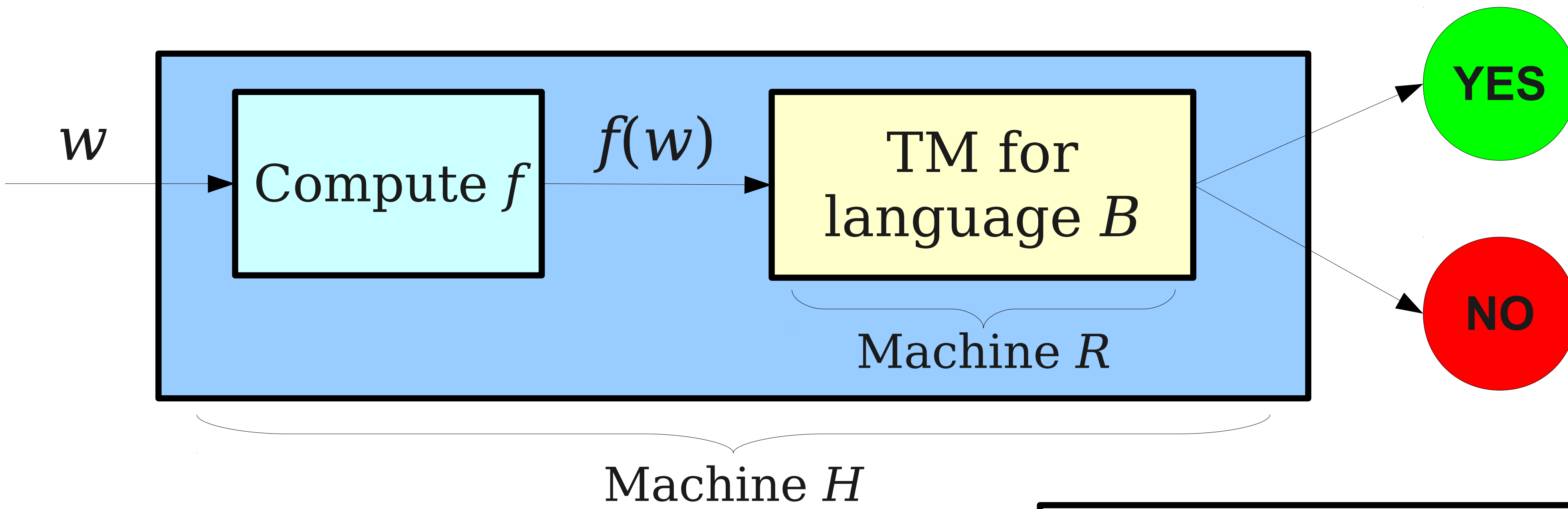
$w \in A$ iff $f(w) \in B$



H = "On input w:
- Transform the input w into $f(w)$.
- Run machine R on $f(w)$.
- If R accepts $f(w)$, then H accepts w.
- If R rejects $f(w)$, then H rejects w."

H* accepts *w
iff
***R* accepts *f*(*w*)**

$$\boldsymbol{w \in A \quad \text{iff} \quad f(w) \in B}$$



H = "On input w:

- Transform the input w into $f(w)$.
- Run machine R on $f(w)$.
- If R accepts $f(w)$, then H accepts w.
- If R rejects $f(w)$, then H rejects w."

H* accepts *w

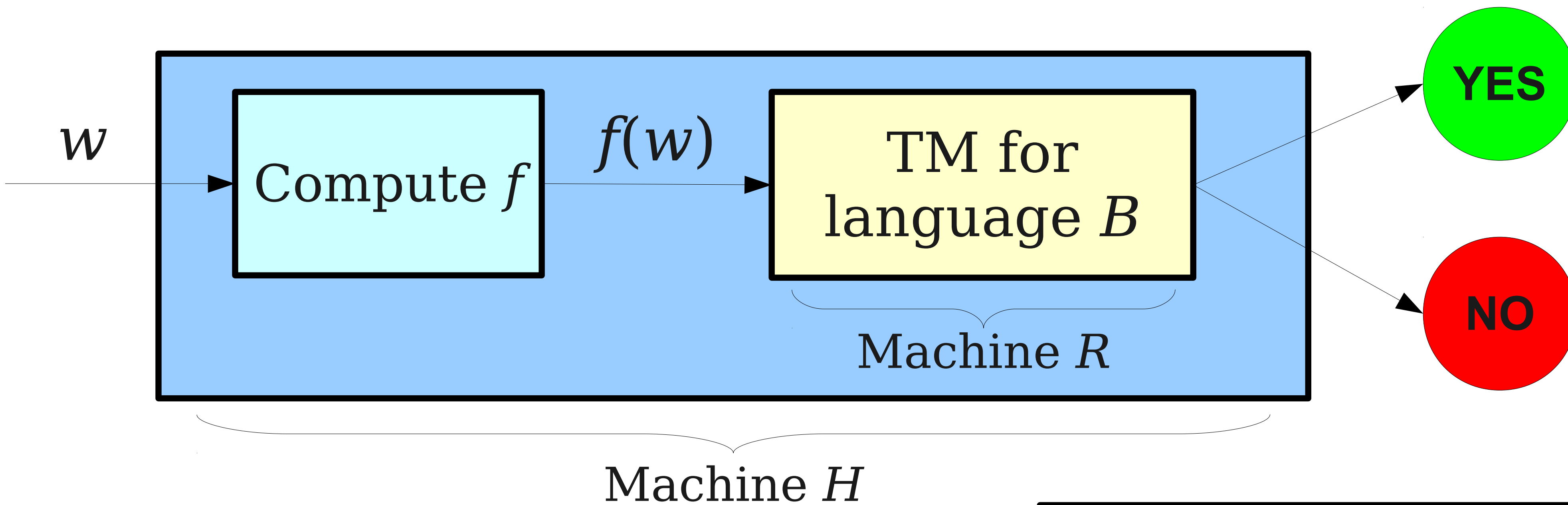
iff

***R* accepts *f*(*w*)**

iff

$\boldsymbol{f(w) \in B}$

$w \in A$ iff $f(w) \in B$



H = "On input w:

- Transform the input w into $f(w)$.
- Run machine R on $f(w)$.
- If R accepts $f(w)$, then H accepts w.
- If R rejects $f(w)$, then H rejects w."

H accepts w

iff

R accepts $f(w)$

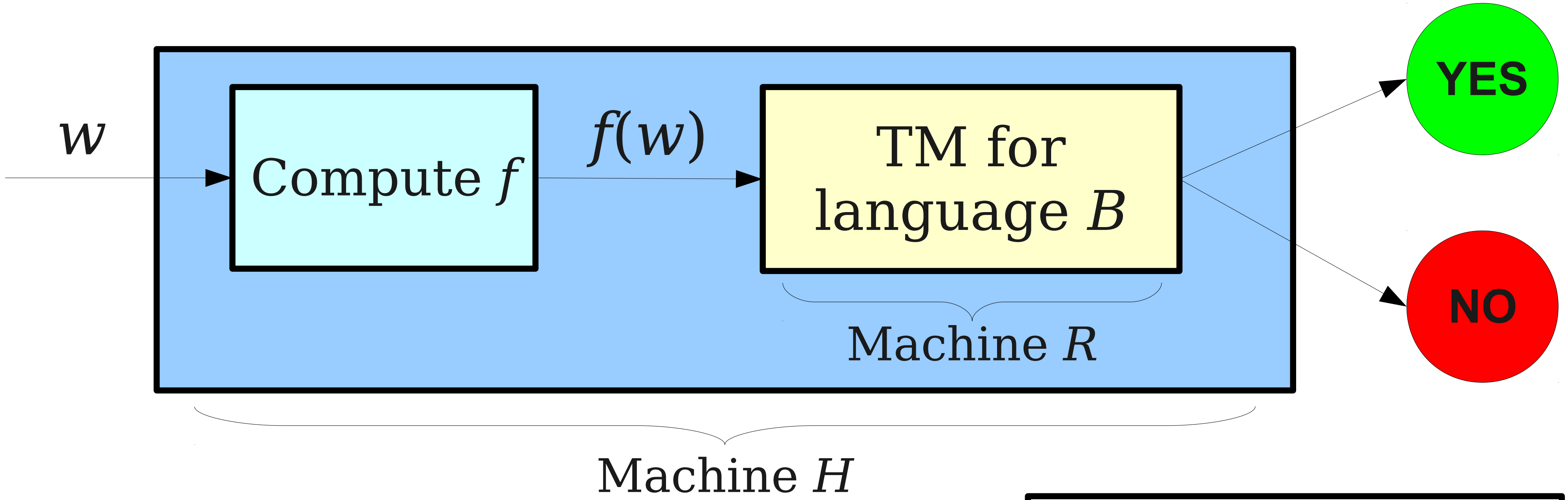
iff

$f(w) \in B$

iff

$w \in A$

$$w \in A \quad \text{iff} \quad f(w) \in B$$



H = "On input w:

- Transform the input w into $f(w)$.
- Run machine R on $f(w)$.
- If R accepts $f(w)$, then H accepts w.
- If R rejects $f(w)$, then H rejects w."

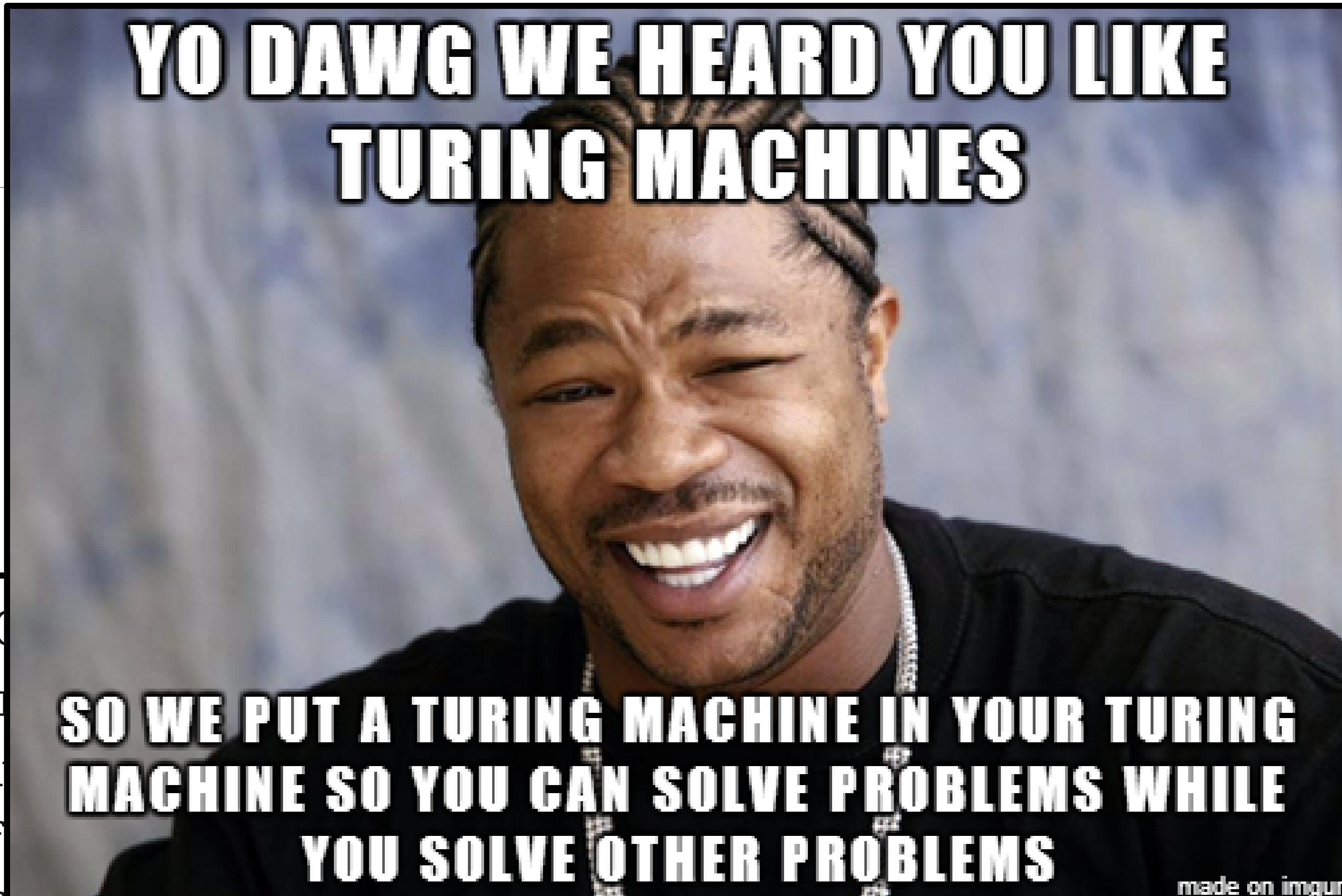
$$\mathscr{L}(H) = A$$

$w \in A$ iff $f(w) \in B$

w

YES

NO



H = "(

- T
- R
- If
- If

Thanks to Alan Kaptanoglu for this image.

A Problem

- Recall: f is a reduction from A to B iff

$$\boldsymbol{w \in A \quad \text{iff} \quad f(w) \in B}$$

- Under this definition, *any* language A reduces to *any* language B unless $B = \emptyset$ or Σ^*.
- Since $B \neq \emptyset$ and $B \neq \Sigma^*$, there is some $w_{yes} \in B$ and some $w_{no} \notin B$.
- Define $f : \Sigma_1^* \rightarrow \Sigma_2^*$ as follows:

$$f(w) = \begin{cases} w_{yes} & \text{if } w \in A \\ w_{no} & \text{if } w \notin A \end{cases}$$

- Then f is a reduction from A to B.

A Problem

- Example: let's reduce L_D to `0`*`1`*.
- Take w_{yes} = `01`, w_{no} = `10`.
- Then $f(w)$ is defined as

$$f(w) = \begin{cases} 01 & \text{if } w \in L_D \\ 10 & \text{if } w \notin L_D \end{cases}$$

- There is no TM that can actually evaluate the function $f(w)$ on all inputs, since no TM can decide whether or not $w \in L_D$.

- E
- T
- T

$$10 \quad \textit{if } w \notin L_D$$

- There is no TM that can actually evaluate the function $f(w)$ on all inputs, since no TM can decide whether or not $w \in L_D$.

Computable Functions

- This general reduction is mathematically well-defined, but might be impossible to actually compute!
- To fix our definition, we need to introduce the idea of a computable function.
- A function $f : \Sigma_1^* \rightarrow \Sigma_2^*$ is called a **computable function** if there is some TM M with the following behavior:

  "On input w:

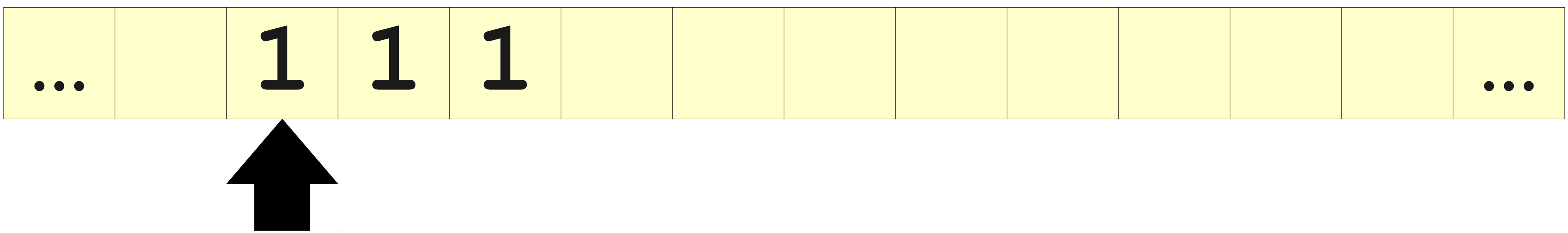
  Compute $f(w)$ and write it on the tape.

  Move the tape head to the start of $f(w)$.

  Halt."

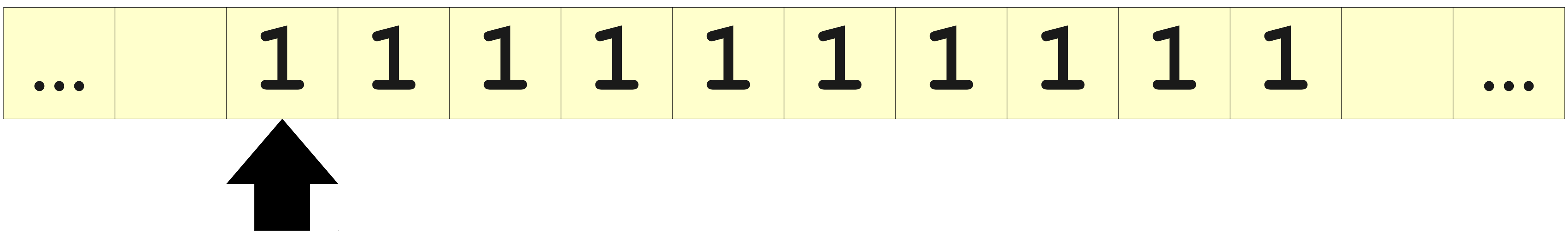
Computable Functions

$$f(\mathbf{1}^n) = \mathbf{1}^{3n+1}$$



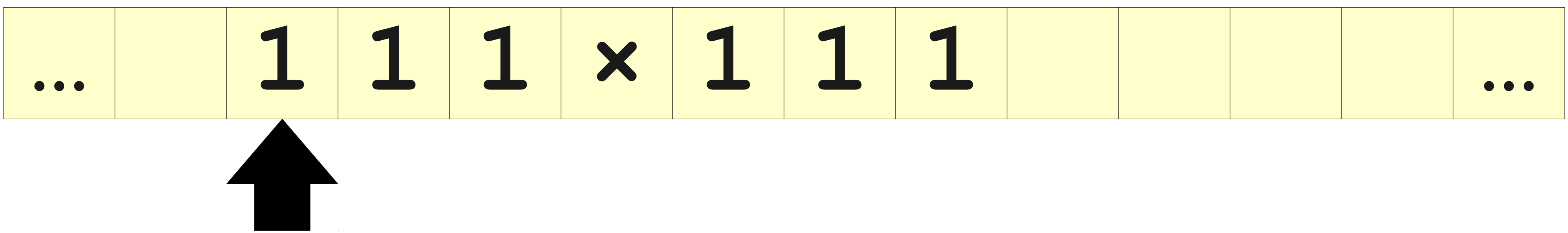
Computable Functions

$$f(\mathbf{1}^n) = \mathbf{1}^{3n+1}$$



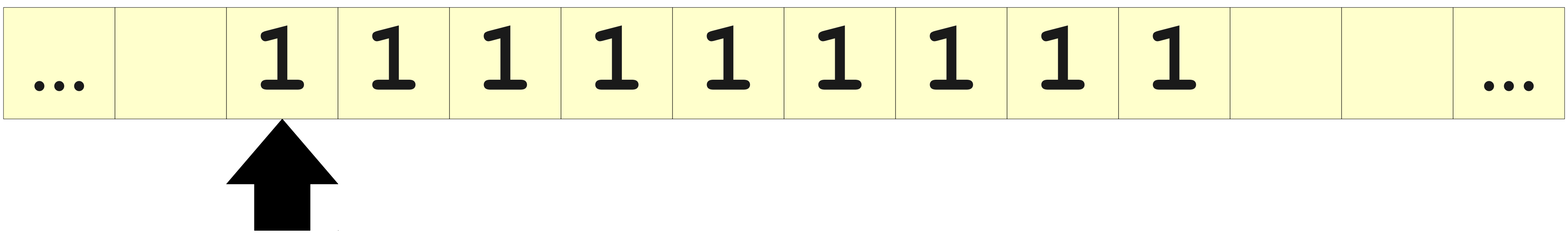
Computable Functions

$$f(w) = \begin{cases} 1^{mn} & \text{if } w = 1^n \times 1^m \\ \varepsilon & \text{otherwise} \end{cases}$$



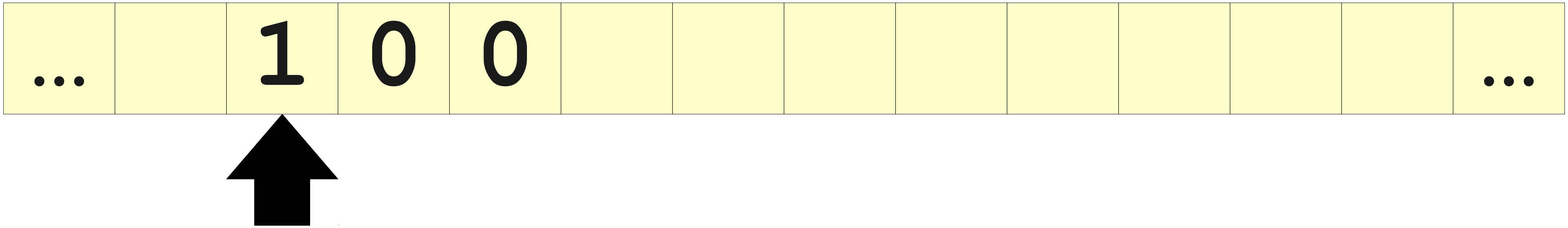
Computable Functions

$$f(w) = \begin{cases} 1^{mn} & \textbf{if } w = 1^n \times 1^m \\ \varepsilon & \textbf{otherwise} \end{cases}$$



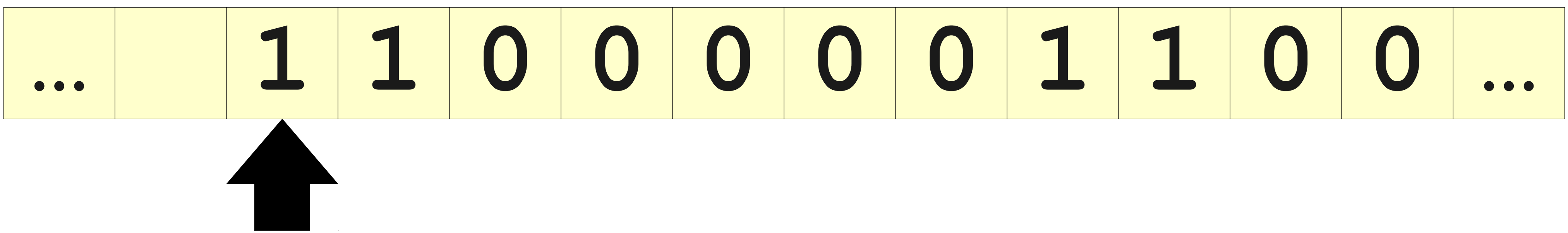
Computable Functions

$$f(\langle M \rangle) = \langle M, \langle M \rangle \rangle$$



Computable Functions

$$f(\langle M \rangle) = \langle M, \langle M \rangle \rangle$$



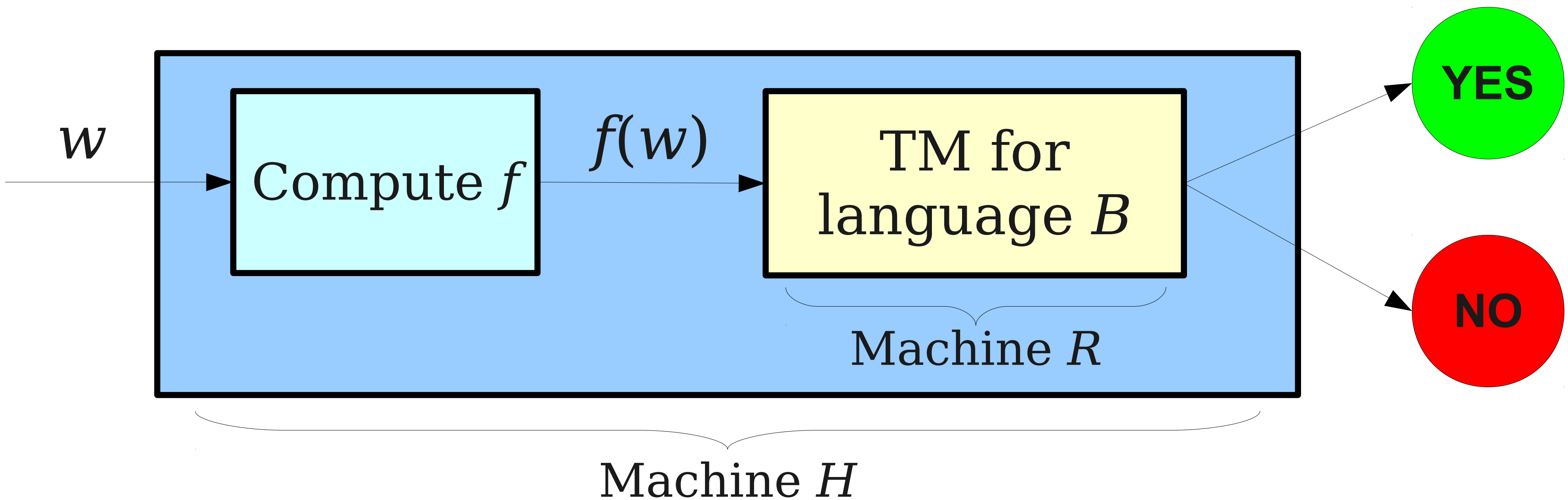
Mapping Reductions

- A function $f : \Sigma_1^* \to \Sigma_2^*$ is called a **mapping reduction** from A to B iff
  - For any $w \in \Sigma_1^*$, $w \in A$ iff $f(w) \in B$.
  - f is a computable function.
- Intuitively, a mapping reduction from A to B says that a computer can transform any instance of A into an instance of B such that the answer to B is the answer to A.

Mapping Reducibility

- If there is a mapping reduction from language A to language B, we say that language A is **mapping reducible** to language B.
- Notation: $\boldsymbol{A \leq_{\mathrm{M}} B}$ iff language A is mapping reducible to language B.
- Note that we reduce ***languages***, not ***machines***.

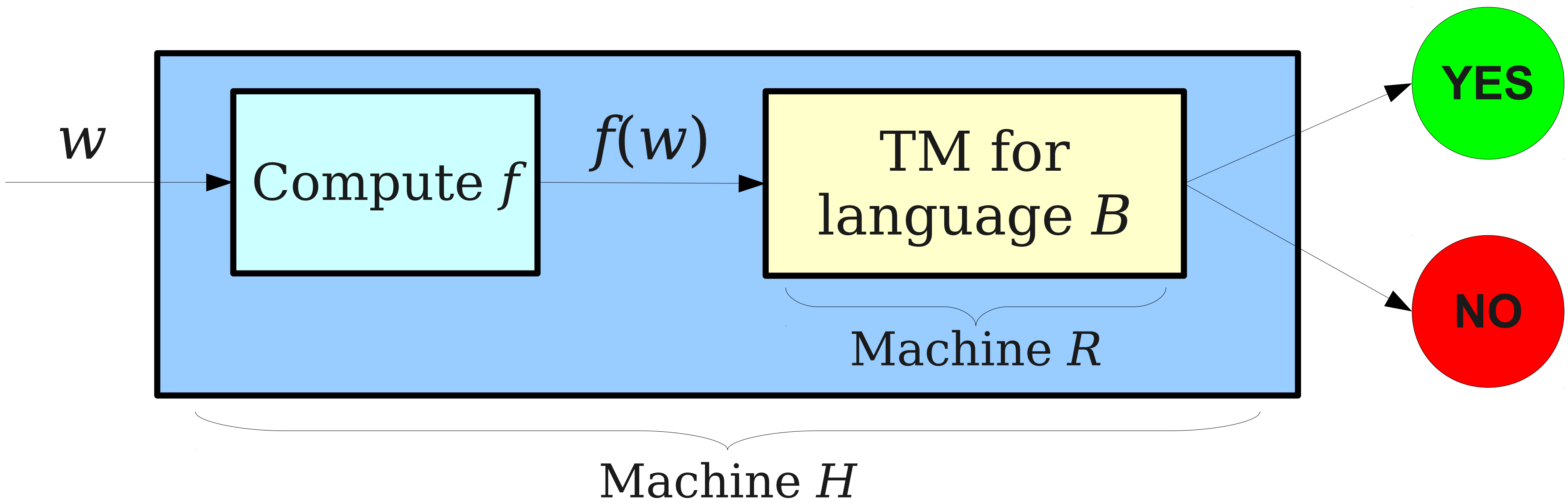
$A \leq_M B$



H = "On input w:

- Compute $f(w)$.
- Run machine R on $f(w)$.
- If R accepts $f(w)$, then H accepts w.
- If R rejects $f(w)$, then H rejects w."

$A \leq_M B$

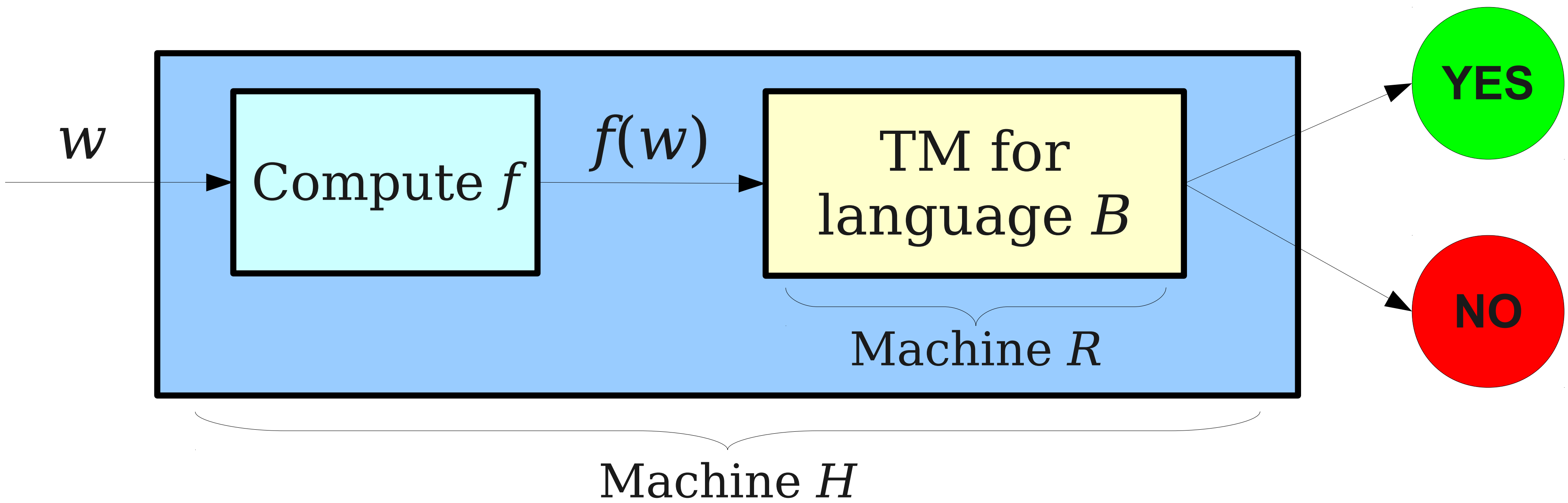


H = "On input w:

- Compute $f(w)$.
- Run machine R on $f(w)$.
- If R accepts $f(w)$, then H accepts w.
- If R rejects $f(w)$, then H rejects w."

If R is a decider for B, then H is a decider for A.

$A \leq_M B$



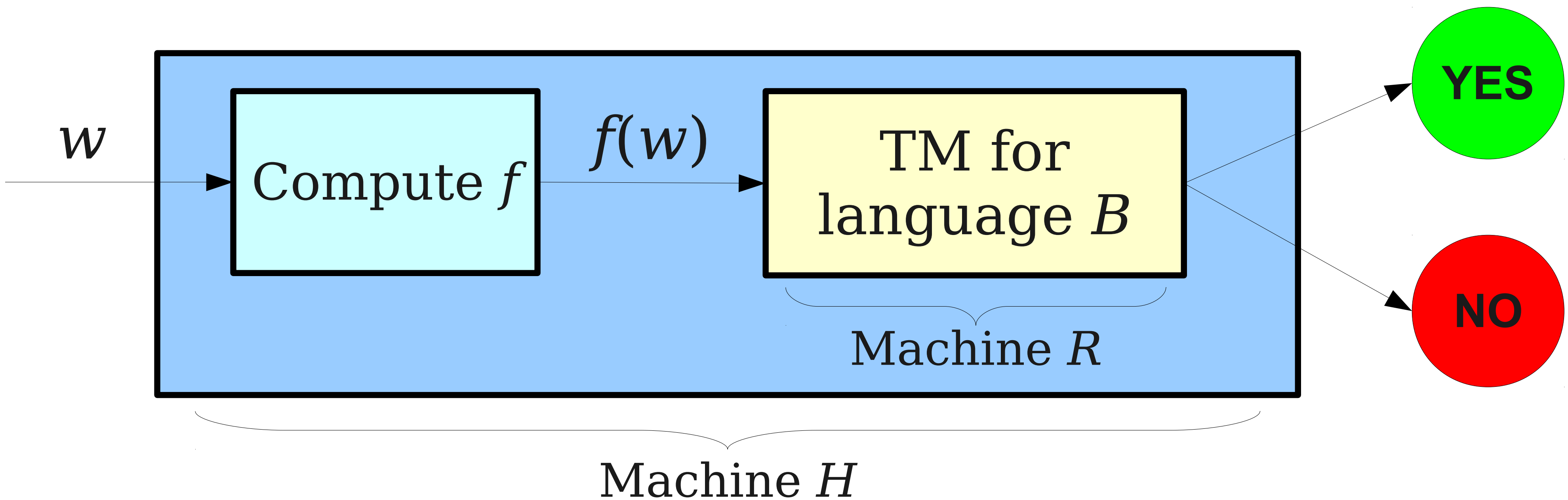
H = "On input w:

- Compute $f(w)$.
- Run machine R on $f(w)$.
- If R accepts $f(w)$, then H accepts w.
- If R rejects $f(w)$, then H rejects w."

If R is a decider for B, then H is a decider for A.

If R is a recognizer for B, then H is a recognizer for A.

$A \leq_M B$



H = "On input w:

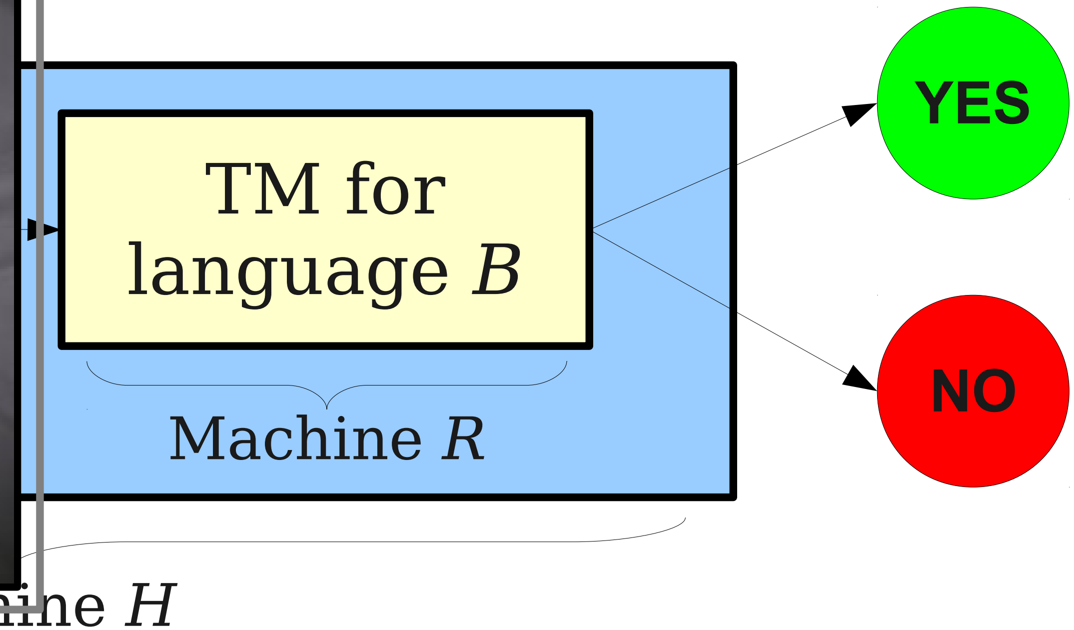
- Compute $f(w)$.
- Run machine R on $f(w)$.
- If R accepts $f(w)$, then H accepts w.
- If R rejects $f(w)$, then H rejects w."

If R is a decider for B, then H is a decider for A.

If R is a recognizer for B, then H is a recognizer for A.

If R is a co-recognizer for B, then H is a co-recognizer for A.

$\leq_M B$



H = "On input w:

- Compute $f(w)$.
- Run machine R on $f(w)$.
- If R accepts $f(w)$, then H accepts w.
- If R rejects $f(w)$, then H rejects w."

If R is a decider for B, then H is a decider for A.

If R is a recognizer for B, then H is a recognizer for A.

If R is a co-recognizer for B, then H is a co-recognizer for A.

Why Mapping Reducibility Matters

- **Theorem**: If $B \in \mathbf{R}$ and $A \leq_{\mathrm{M}} B$, then $A \in \mathbf{R}$.
- **Theorem**: If $B \in \mathbf{RE}$ and $A \leq_{\mathrm{M}} B$, then $A \in \mathbf{RE}$.
- **Theorem**: If $B \in \text{co-}\mathbf{RE}$ and $A \leq_{\mathrm{M}} B$, then $A \in \text{co-}\mathbf{RE}$.
- *Intuitively:* $A \leq_{\mathrm{M}} B$ means "A is not harder than B."

Why Mapping Reducibility Matters

- **Theorem**: If $A \notin \mathbf{R}$ and $A \leq_{\mathrm{M}} B$, then $B \notin \mathbf{R}$.
- **Theorem**: If $A \notin \mathbf{RE}$ and $A \leq_{\mathrm{M}} B$, then $B \notin \mathbf{RE}$.
- **Theorem**: If $A \notin \text{co-}\mathbf{RE}$ and $A \leq_{\mathrm{M}} B$, then $B \notin \text{co-}\mathbf{RE}$.
- *Intuitively:* $A \leq_{\mathrm{M}} B$ means "B is at at least as hard as A."

Why Mapping Reducibility Matters

If this one is "easy" (R, RE, co-RE)…

$$A \leq_{\mathrm{M}} B$$

… then this one is "easy" (R, RE, co-RE) too.

Why Mapping Reducibility Matters

If this one is "hard" (not **R**, not **RE**, or not co-**RE**)…

$$A \leq_{\mathrm{M}} B$$

… then this one is "hard" (not **R**, not **RE**, or not co-**RE**) too.