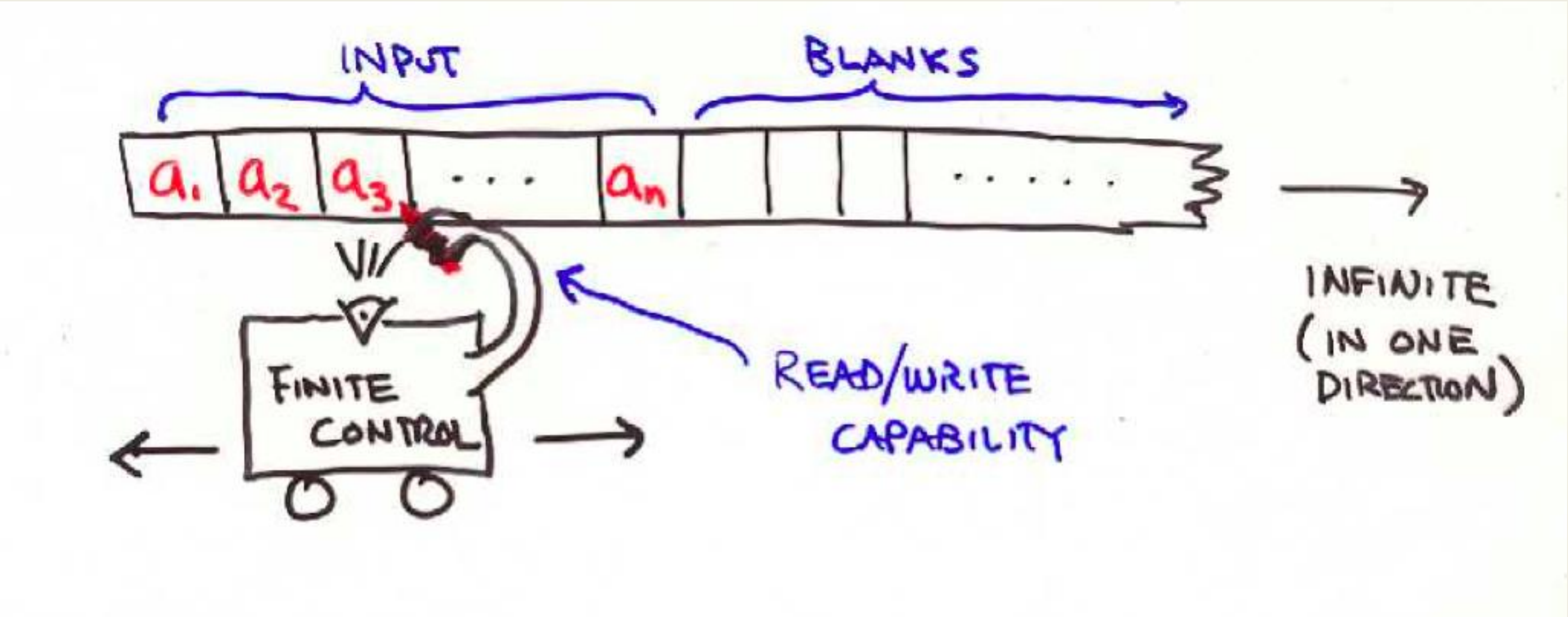
Turing Machine Recap



- DFA with (infinite) tape.
- One move: read, write, move, change state.

High-level Points

- **Church-Turing thesis:** TMs are the most general computing devices. So far no counter example
- **Every TM can be represented as a string**. Think of TM as a program but in a very low-level language.
- **Universal Turing Machine M_u** that can simulate a given M on a given string w

Decision Problems

- A yes/no question over many instances
  - Given grammar G, is G ambiguous?
  - Given a TM M, does $L(M) = \{0,1\}^*$?
  - Given DFAs M_1 and M_2, does $L(M_1) = L(M_2)$?
  - Given a graph G, is G connected?
  - Given a graph G, nodes *s* and *t*, and number *d*, is there a path from *s* to *t* of distance *d* or less?

Equivalently, languages:

- {<G> | <G> encodes an unambiguous grammar}
- {<*M*> | L(*M*) = $\{0,1\}^*$ }
- {<M_1> # <M_2> | DFAs M_1 and M_2, accept the same language}
- {<G> | <G> encodes a connected graph}
- {<G>#*s*#*t*#*d* | <G> encodes a graph with nodes *s* and *t*, there is a path from *s* to *t* of distance *d* or less}

Deciding membership in the language is solving the decision problem

Decidable

- A decision problem (language) is *decidable* if there is a TM that always halts that accepts the language. (The language is recursive.)
- I.e., there is an algorithm that always answers "yes" or "no" correctly.
- Note: since all finite languages are recursive, (they're regular in fact) any decision problem with only a finite number of instances is decidable, and not well-addressed by this theory....

Example 1: decidable or not?

- Is there a substring of exactly 374 consecutive 7's in decimal expansion of π ?
- This is decidable. There is an algorithm which is correct. It is one of these:

Alg 1	Alg 2
Output "yes"	Output "no"

We just don't know which one it is

But, there is an algorithm which will tell us which it is!

Moral

- This is nonsense
- There were no “instances” of the problem.
- It simply asks a single yes/no question.
- Not even clear what “language” corresponds to it
- Remember: decidability is for problems with many possible input instances

Example 2

- Give *n*, is there a substring of exactly *n* consecutive 7's in π ?
- Language: {*n* | decimal expansion of π contains the substring $a7^nb$*,* where *a* and *b* are not 7s}
- Is this language decidable? Is there a halting TM for it?
- Is it r.e.? (recall: a TM that may not halt but accepts if it should)

Example 3

- Give *n*, is there a substring of *at least n* consecutive 7’s in π ?
- Language: L = {*n* | decimal expansion of π contains the substring 7^n}
- Is this language decidable? Is there a halting TM for it?
- In fact, it is regular!

(L is either all of N, or equals {0,1,2,...,*k*} for some fixed *k*.)

Universal TM

- A *single* TM M_u that can compute anything computable!
- Takes as input
  - the ***description*** of some *other* TM *M*
  - data *w* for *M* to run on
- Outputs
  - the results of running *M*(*w*)

Recap: Typical TM code:

1110101000010010011010010000010101 1.....11......11......111

- Begins, ends with 111
- Transitions separated by 11
- Fields within transition separated by 1
- Individual fields represented by 0s
- Note: this can be viewed as a natural number

Recap: Universal TM M_u

We saw a TM M_u such that

$L(M_u) = \{ <M> \# w \mid M \text{ accepts } w\}$

Thus, M_u is a stored-program computer.

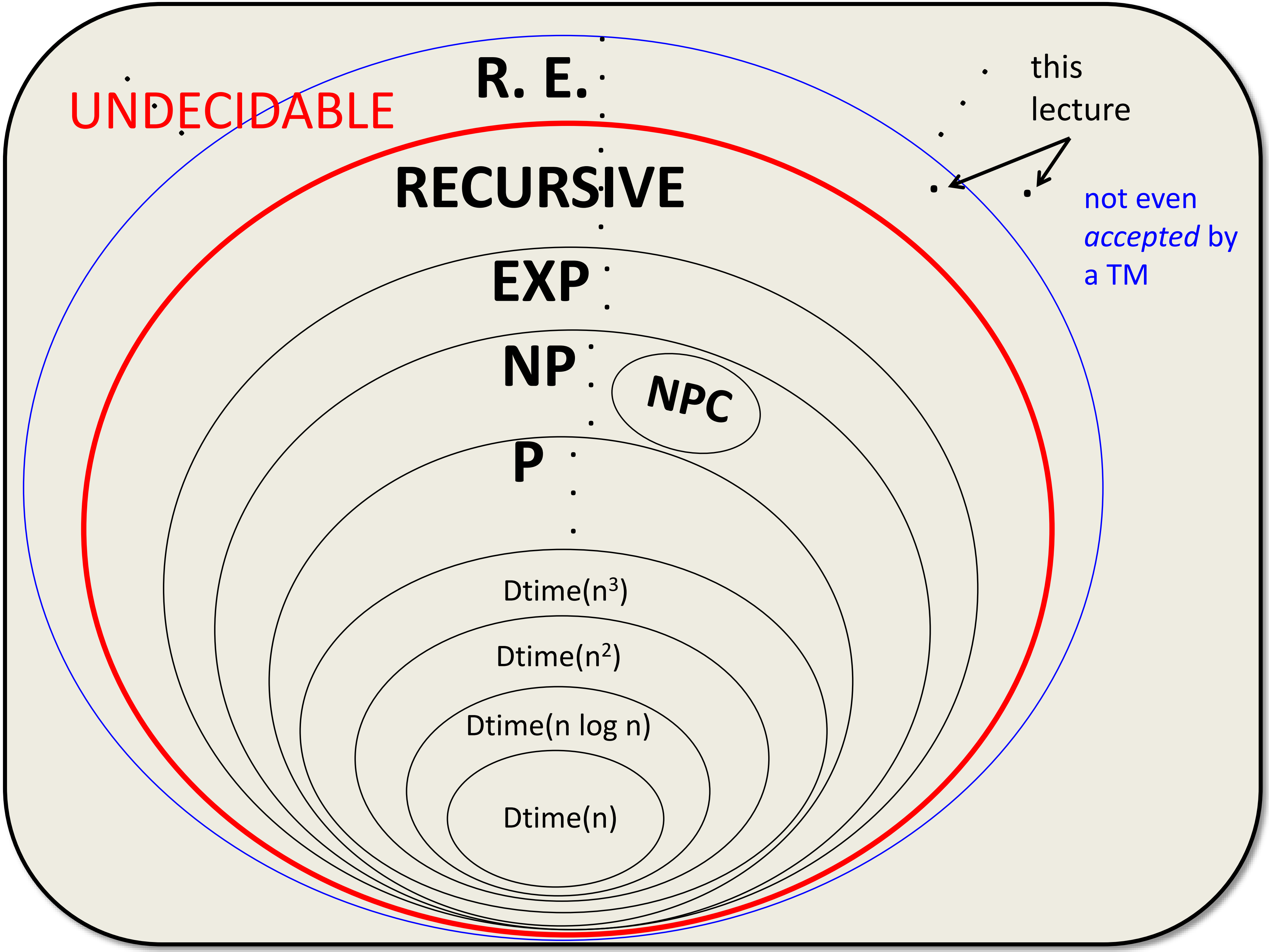
It reads a program $<M>$ and executes it on data w

$L_u = L(M_u) = \{ <M> \# w \mid M \text{ accepts } w\}$ is r.e.

High-level Points

- **Church-Turing thesis:** TMs are the most general computing devices. So far no counter example
- **Every TM can be represented as a string**. Think of TM as a program but in a very low-level language.
- **Universal Turing Machine M_u** that can simulate a given M on a given string w

Undecidability



R. E.
UNDECIDABLE
this
lecture
RECURSIVE
not even
accepted by
a TM
EXP
NP
NPC
P
Dtime(n³)
Dtime(n²)
Dtime(n log n)
Dtime(n)

Undecidable Languages: Counting Argument

- Are there undecidable languages?
- Most languages are undecidable!
- Simple proof:
  - \# of TMs/algorithms is **countably infinite** since each TM can be represented as a natural number (it's description is a unique binary number)
  - \# of languages is **uncountably infinite**

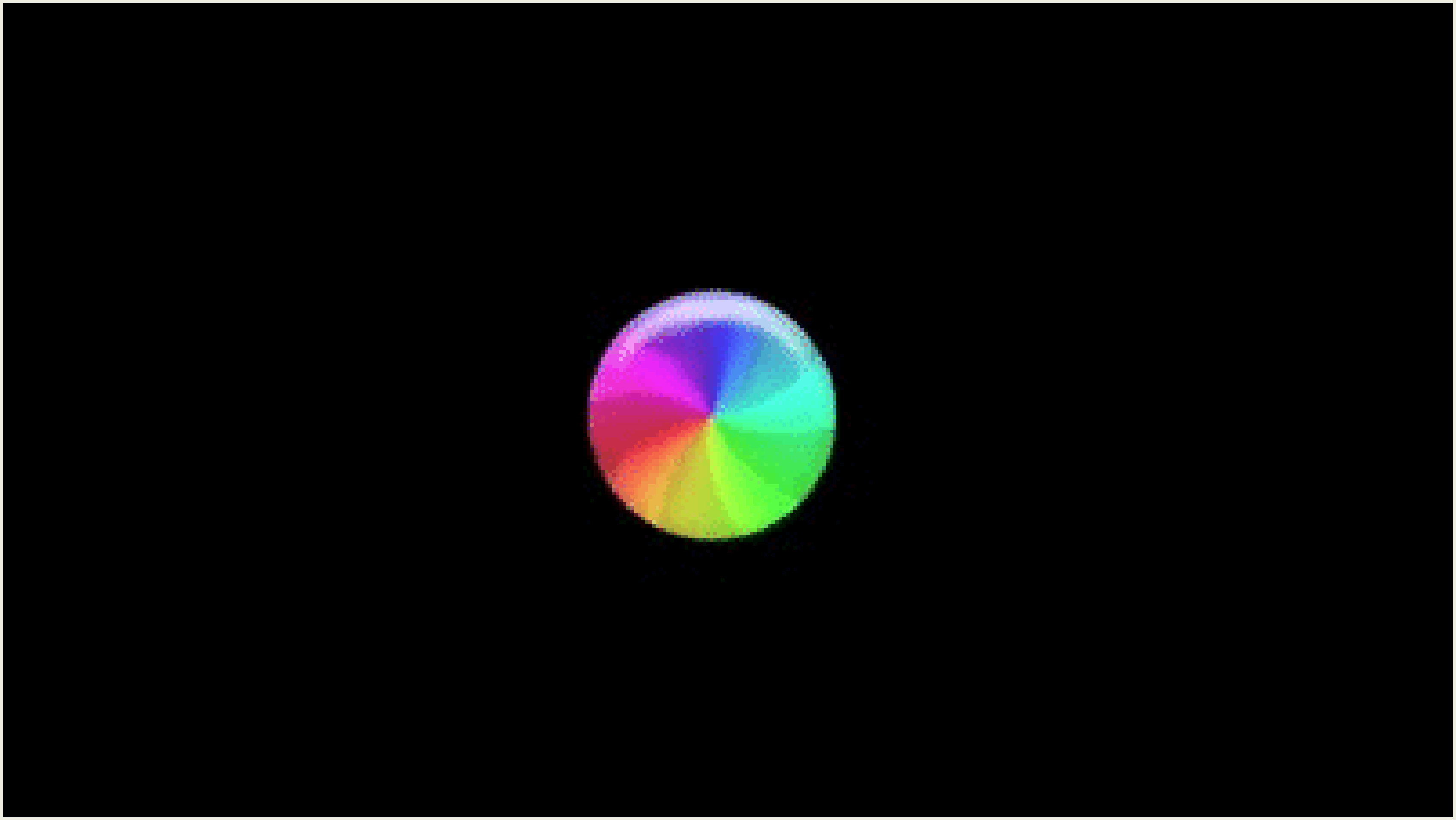
Is L_u decidable?

- Counting argument does not directly tell us about undecidablity of specific interesting languages
- Recall L_u = { <M>#w | M accepts w } is r.e.
- Is L_u decidable?

Halting Problem

- Does given *M* halt when run on *blank input*?
- L_{halt} = {<*M*> | *M* halts when run on blank input}
- Is L_{halt} decidable?

Who cares about halting TMs?



Who cares about halting TMs?

- Remember, TMs = programs
- Debugging is an important problem in CS
- Furthermore, *virtually all math conjectures can be expressed as a halting-TM question*.

Example: Goldbach's conjecture:

Every even number > 2 is the sum of two primes.

Program Goldbach

```
is-sum-of-two-primes(n): boolean
    FOR p ≤ q < n
            IF p,q, prime AND p+q=n THEN RETURN TRUE
    RETURN FALSE

goldbach()
    n = 4
    WHILE is-sum-of-two-primes(n)
        n = n+2
    HALT
```

goldbach() halts iff Goldbach's conjecture is false

CS 125 assignment:

- Write a program that outputs “Hello world”.

```
main()
{   printf(“Hello world”);
}
```

- Can you write an auto-grader?
- If so; you can solve Goldbach’s conjecture...

```
goldbach()
    n = 4
    WHILE is-sum-of-two-primes(n)
        n = n+2
    HALT
```

```
is-sum-of-two-primes(n): boolean
    FOR p ≤ q < n
        IF p,q, prime AND p+q=n
            THEN RETURN TRUE
    RETURN FALSE
```

```
main()

    { goldbach();
     printf("Hello world");
    }
```

main() → AUTOGRADER → CORRECT / INCORRECT

So, deciding if a program prints "Hello world" is solving goldbach's conjecture

Deciding halting problem

- Given program <*M*>, to determine if *M* halts, do the following:

So, deciding if a program prints "Hello world" is solving the halting problem

```
main()

    { M()
     printf("Hello world");
    }
```

→ AUTOGRADER → CORRECT / INCORRECT

Using same ideas, we can show that deciding anything about code behavior is not possible

L_u *is not recursive*

Two proofs

- Slick proof
- Slow proof via diagonalization and reduction

L_u is not decidable

Warm-up: Self-reference leads to paradox

- In a town there is a barber who shaves all and only those who do not shave themselves

  Who shaves the barber?

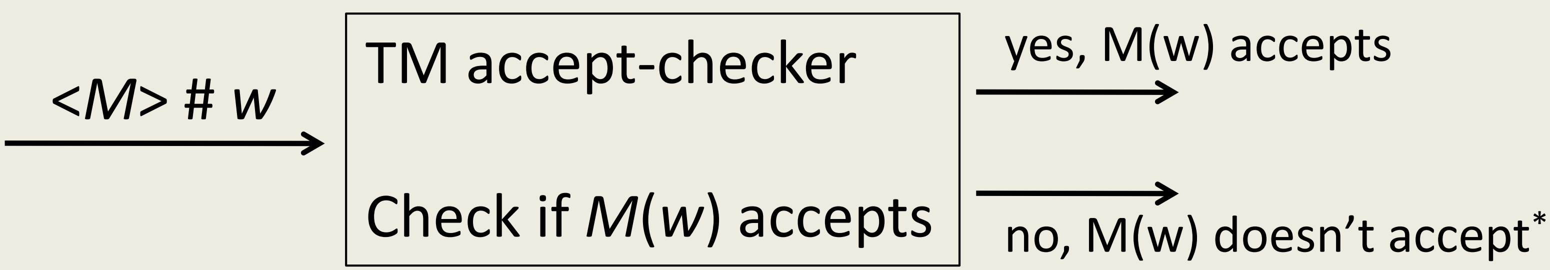
- Homogenous words: self-describing
  - English, short, polysyllabic

  Heterogenous words: non-self-describing
  - Spanish, long, monosyllabic

What kind of word is "heterogenous" ?

L_u is not decidable

- Proof by contradiction
- Suppose there was an algorithm (TM) that always halted, as follows:

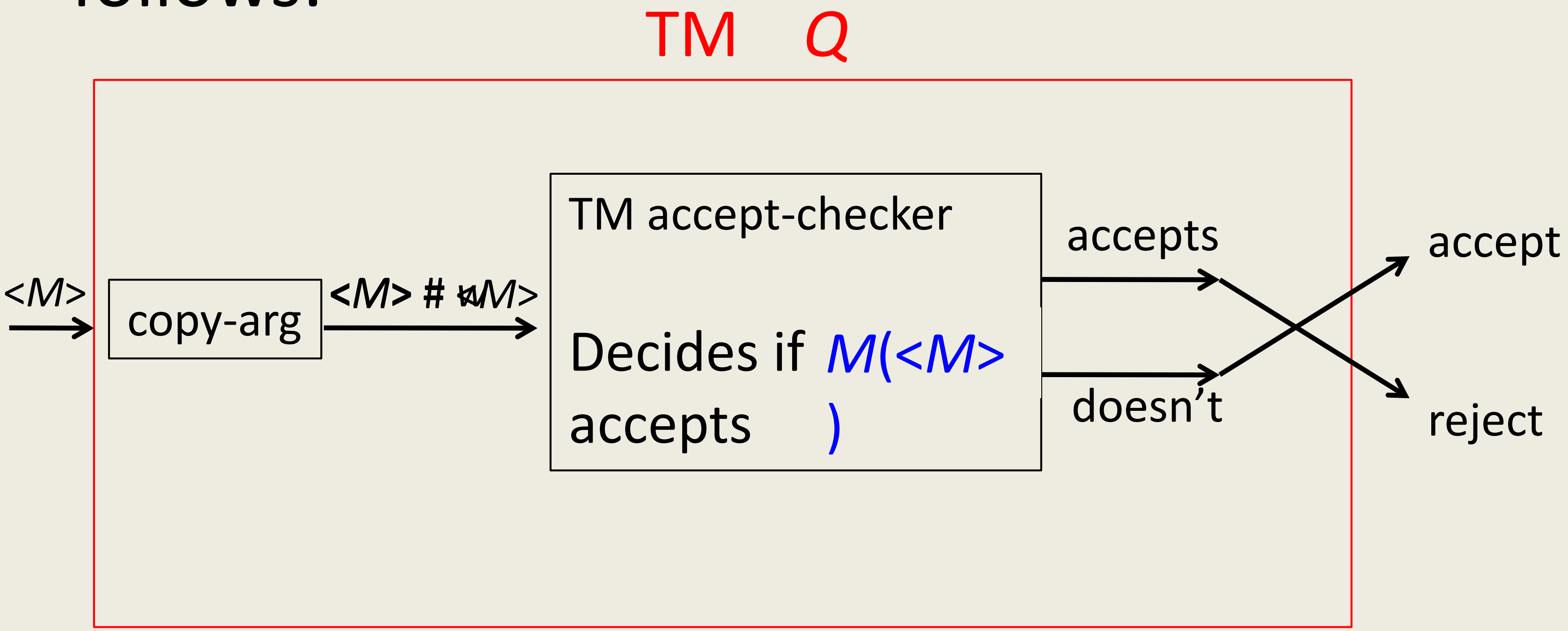


* remember – *M*(*w*) may not halt – which is why this may be difficult

We'll show how to use this as a subroutine to get a contradiction

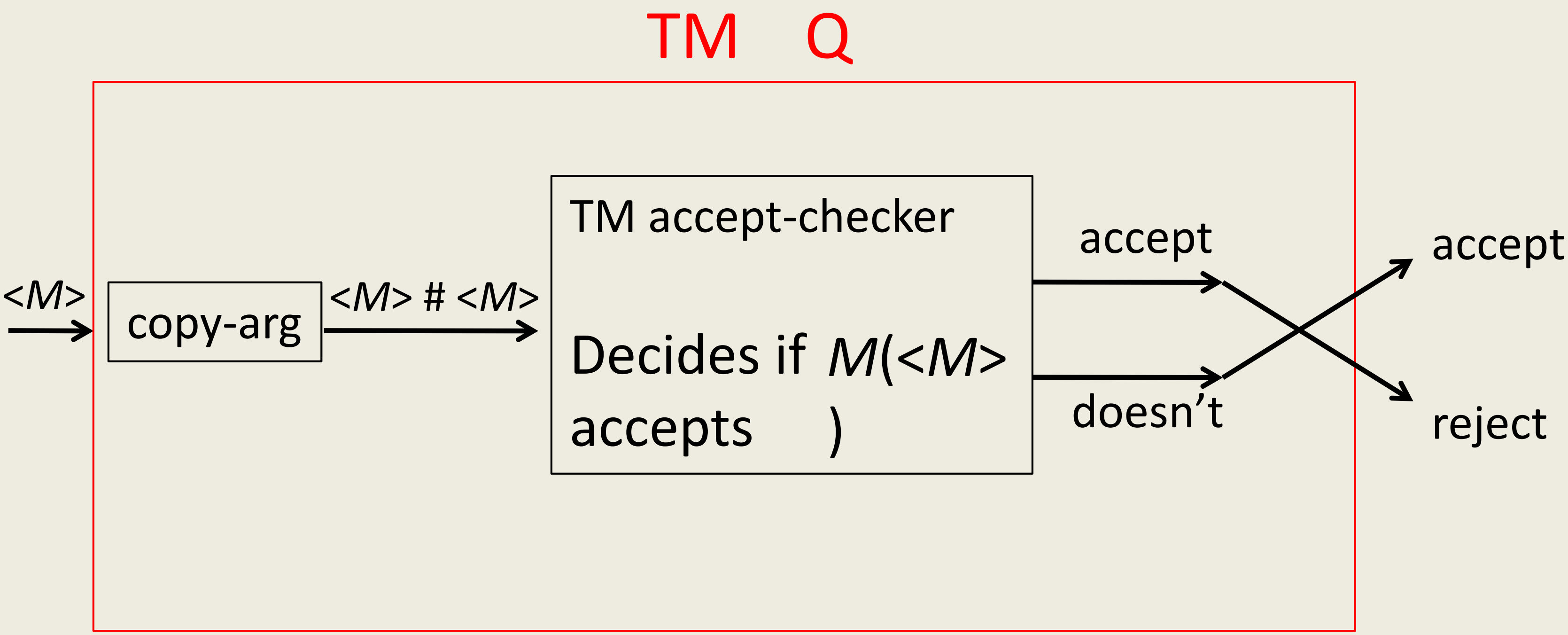
L_u *is not decidable*

- Proof by contradiction
- Suppose there was an algorithm (TM) as follows:



Q(<*M*>) accepts iff *M*(<*M*>) doesn't accept
Q(<*M*>) rejects iff *M*(<*M*>) accepts

L_u is not decidable



Q(<*M*>) accepts iff *M*(<*M*>) doesn't accept

Q(<*M*>) rejects iff *M*(<*M*>) accepts

Does Q(<Q>) accept or reject?

either way, a contradiction, so assumption that accept-checker existed was wrong

L_u *is not decidable: Slow proof*

- Use diagonalization to prove that a specific language L_d is not r.e
- Show that if L_u is decidable then L_d is decidable which leads to contradiction

Diagonalization

- Fix alphabet to be {0,1}
- Recall that {0,1}* is countable: we can enumerate strings as w_0, w_1, w_2,...
- Recall that we established a correspondence between TMs and binary numbers hence TMs can be enumerated as M_0, M_1, M_2, …
- A language L is a subset of {0,1}*

List of all r.e. languages

	w_0	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	...
M_0	no	no	no	no	no	no	no	no	no	no	...
M_1	yes	no	no	yes	no	yes	yes	yes	yes	no	...
M_2	no	yes	yes	no	no	yes	no	yes	no	no	...
M_3	no	yes	no	yes	no	yes	no	yes	no	yes	...
M_4	yes	yes	yes	yes	no	no	no	no	no	no	...
M_5	no	no	no	no	no	no	no	no	no	no	...
M_6	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	...
M_7	yes	yes	no	no	yes	yes	yes	no	no	yes	...
M_8	no	yes	no	no	yes	no	yes	yes	yes	no	...
M_9	no	no	no	yes	yes	no	yes	no	yes	yes	...
...	...	...	...	...	...	...	...	...	...	...	...

List of all r.e. languages

	w_0	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	...
M_0	no	no	no	no	no	no	no	no	no	no	...
M_1	yes	no	no	yes	no	yes	yes	yes	yes	no	...
M_2	no	yes	yes	no	no	yes	no	yes	no	no	...
M_3	no	yes	no	yes	no	yes	no	yes	no	yes	...
M_4	yes	yes	yes	yes	no	no	no	no	no	no	...
M_5	no	no	no	no	no	no	no	no	no	no	...
M_6	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	...
M_7	yes	yes	no	no	yes	yes	yes	no	no	yes	...
M_8	no	yes	no	no	yes	no	yes	yes	yes	no	...
M_9	no	no	no	yes	yes	no	yes	no	yes	yes	...
...	...	...	...	...	...	...	...	...	...	...	...

Consider for each *i*, whether or not M_i *accepts* w_i

List of all r.e. languages

	w_0	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	...
M_0	yes	no	no	no	no	no	no	no	no	no	...
M_1	yes	yes	no	yes	no	yes	yes	yes	yes	no	...
M_2	no	yes	no	no	no	yes	no	yes	no	no	...
M_3	no	yes	no	no	no	yes	no	yes	no	yes	...
M_4	yes	yes	yes	yes	yes	no	no	no	no	no	...
M_5	no	no	no	no	no	yes	no	no	no	no	...
M_6	yes	yes	yes	yes	yes	yes	no	yes	yes	yes	...
M_7	yes	yes	no	no	yes	yes	yes	yes	no	yes	...
M_8	no	yes	no	no	yes	no	yes	yes	no	no	...
M_9	no	no	no	yes	yes	no	yes	no	yes	no	...
...	...	...	...	...	...	...	...	...	...	...	...

Flip “yes” and “no”, defining $L_d = \{w_i \mid w_i \text{ not in } L(M_i)\}$

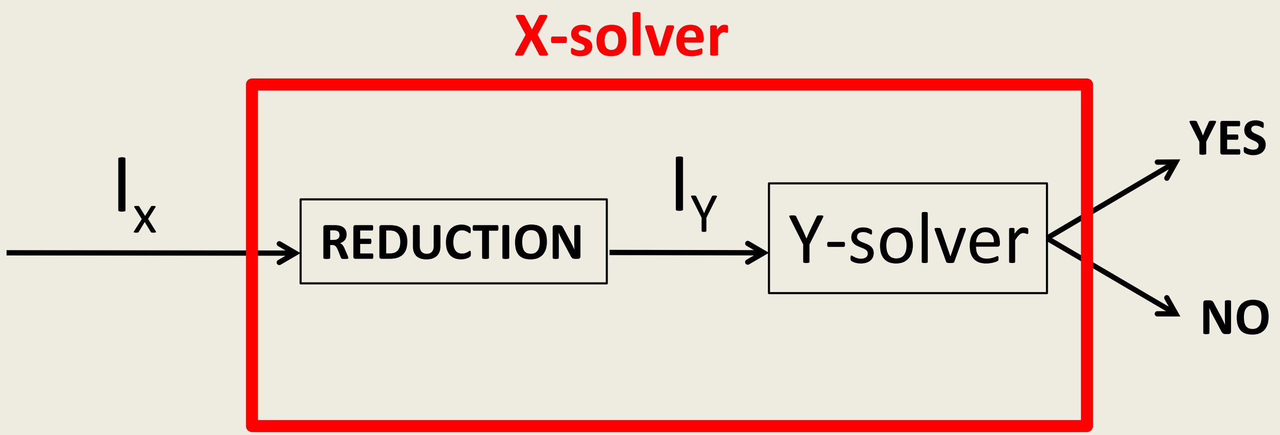
$L_d = \{w_i \mid w_i \text{ not in } L(M_i)\}$

L_d is not r.e. (Why not?)

- if it were, it would be accepted by some TM M_k
- but L_d contains w_k iff $L(M_k)$ does not contain w_k
- so $L_d \neq L(M_k)$ for *any* k
- so L_d is not r.e.

Reduction

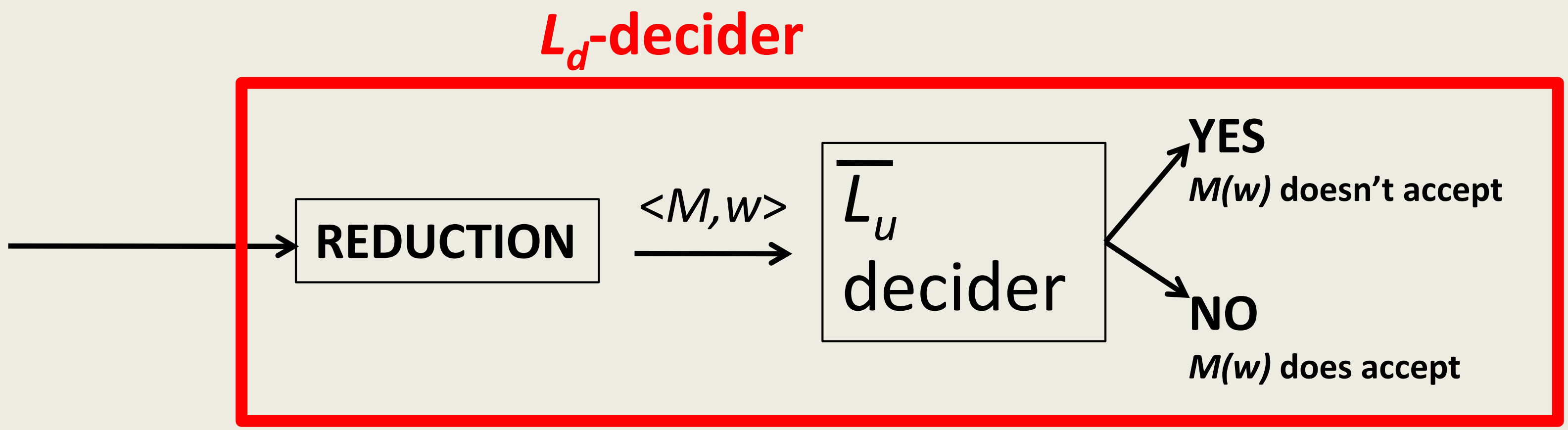
$X \leq Y$ "X reduces to Y"



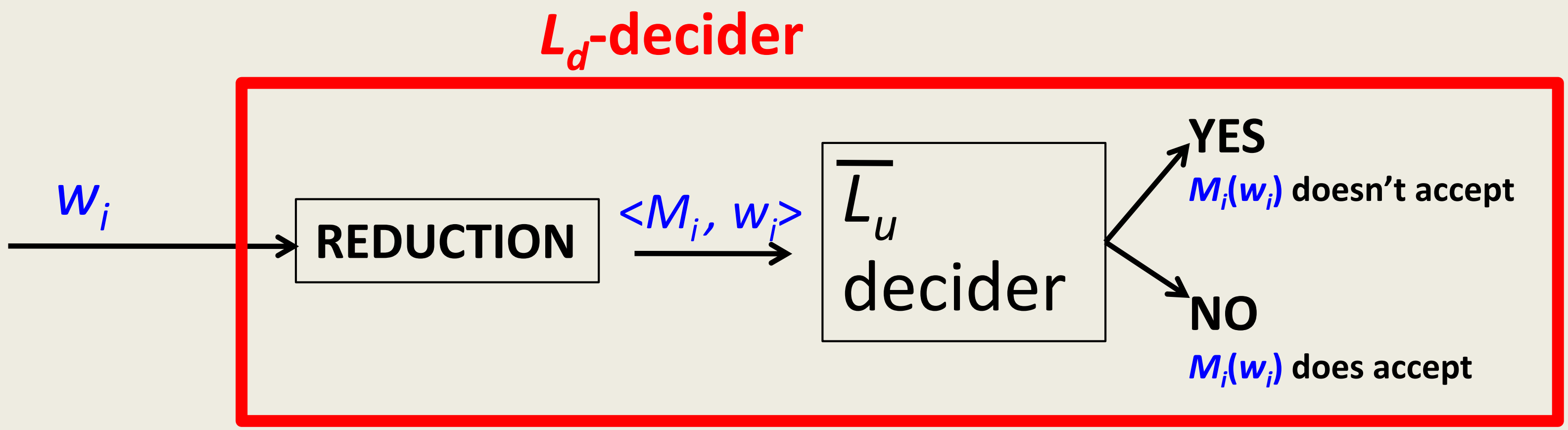
If Y can be decided, then X can be decided.

If X can't be decided, then Y can't be decided

$$L_d \leq \overline{L_u}$$



$L_d \leq \overline{L_u}$



- The above is a reduction from L_d to complement of L_u
- Note that a language L is decidable iff $\overline{L}$ is decidable
- Hence L_u is decidable iff $\overline{L_u}$ decidable

L_u is not decidable

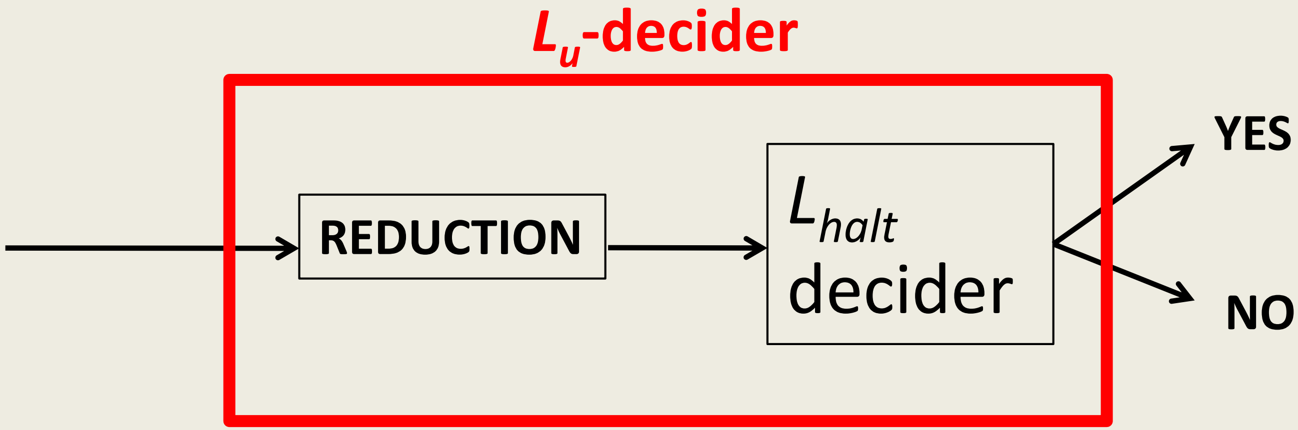
- L_d is not r.e. by diagonalization
- Suppose L_u is decidable
- Then $\overline{L_u}$ is also decidable
- We have shown $L_d \leq \overline{L_u}$ which implies L_d is decidable, a contradiction
- Therefore L_u is **not** decidable (undecidable)
- No algorithm for L_u

Using Reductions

- Once we have some seed problems such as L_d and L_u we can use reductions to prove that more problems are undecidable

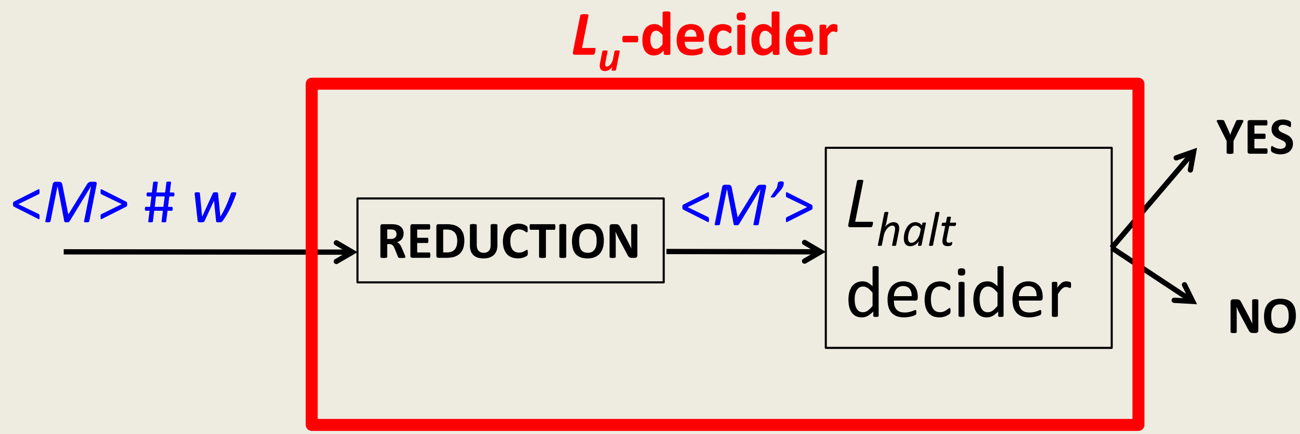
Halting Problem

- Does given M halt when run on *blank input*?
- L_{halt} = {<M> | M halts when run on blank input}
- Show L_{halt} is undecidable by showing $L_u \leq L_{halt}$



What are input and output of the reduction?

$L_u \le L_{halt}$



Need: *M'* halts on blank input iff *M*(*w*) accepts

<u>TM *M'*</u>

  const *M*

  const *w*

run *M*(*w*) and halt if it accepts

The REDUCTION **doesn't run *M* on *w***. It produces code for *M'* !

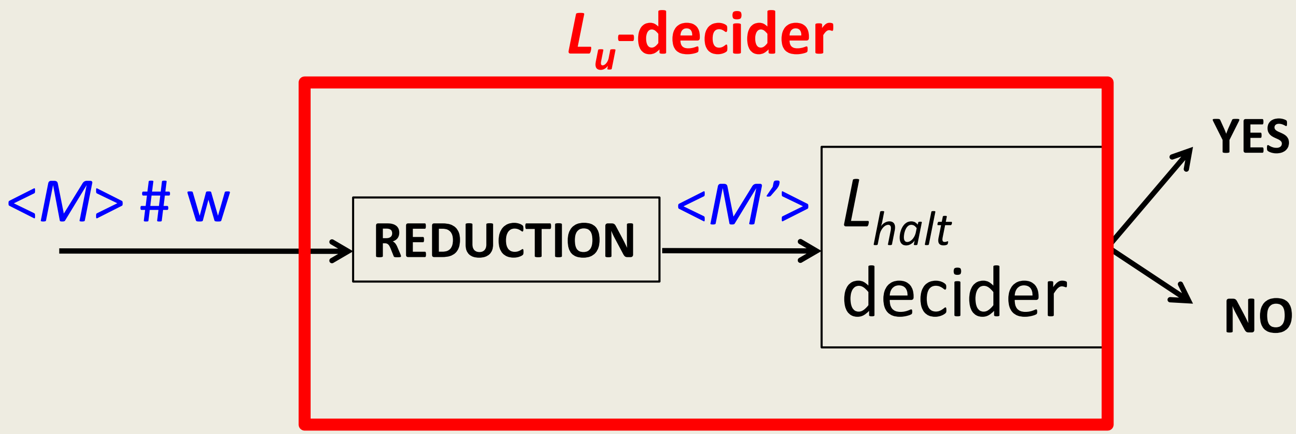
Example

- Suppose we have the code for a program isprime() and we want to check if it accepts the number 13
- The reduction creates new program to give to decider for L_{halt}: note that the reduction only creates the code, does not run any program itself.

```
main() {
        If (isprime(13)) then
            HALT
        else
            LOOP FOREVER
    }

 boolean isprime(int i) {
     …
   }
```

$L_u \leq L_{halt}$



Need: *M'* halts on blank input iff *M*(*w*) accepts

<u>TM *M'*</u>

const *M*

const *w*

run *M*(*w*) and halt if it accepts

Correctness: L_u-decider say "yes" iff *M'* halts on blank input

iff *M*(*w*) accepts

iff <*M*>#*w* is in L_u

More reductions about languages

- We'll show other languages involving program behavior are undecidable:
- $L_{374} = \{<M> \mid L(M) = \{0^{374}\}\}$
- $L_{\neq \emptyset} = \{<M> \mid L(M) \text{ is nonempty}\}$
- $L_{pal} = \{<M> \mid L(M) = \text{palindromes}\}$
- *many many* others

$L_{374} = \{\langle M \rangle \mid L(M) = \{0^{374}\}\}$ is undecidable

- Given a TM M, telling whether it accepts only the string 0^{374} is not possible
- Proved by showing $L_u \leq L_{374}$

$\langle M \rangle \# w$ → instance of L_u

REDUCTION: BUILD M'

$\langle M' \rangle =$ → instance of L_{374}

M': constants: M, w

On input x,

0. if $x \neq 0^{374}$, reject
1. if $x = 0^{374}$, then run $M(w)$

   accept x iff $M(w)$ ever accepts w

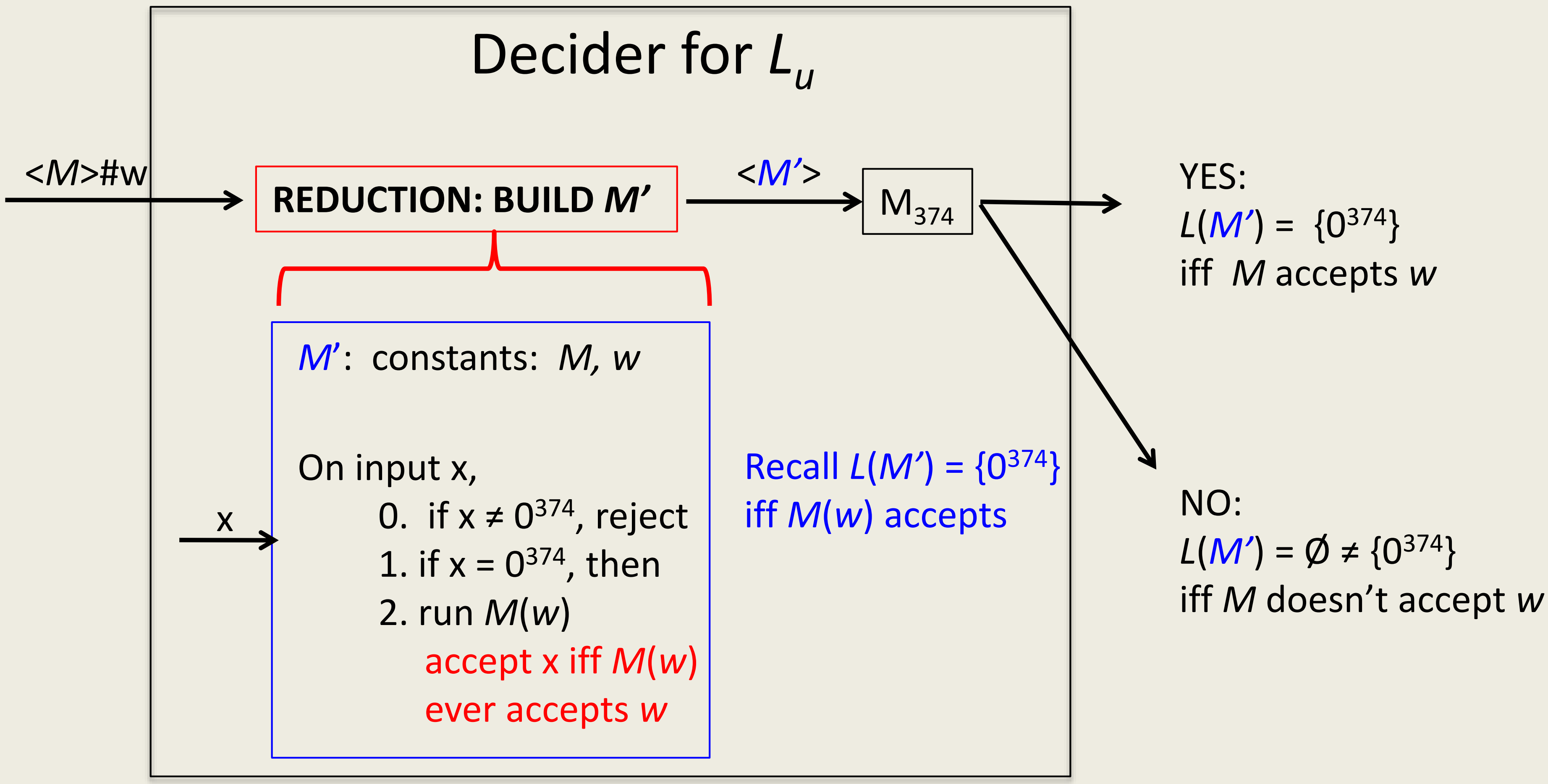
What is $L(M')$?

- If $M(w)$ accepts, $L(M') = \{0^{374}\}$
- If $M(w)$ doesn't $L(M') = \emptyset$

Q: How does the reduction know whether or not $M(w)$ accepts ?

A: It doesn't have to. It just *builds* (code for) *M'*.

If there is a decider M_{374} to tell if a TM accepts the language $\{0^{374}\}$...



Since L_u is not decidable, M_{374} doesn't exist, and L_{374} is undecidable

L_{374} = {<M> | $L(M)$ = {0^{374}} } is undecidable

- What about $L_{accepts\text{-}374}$ = {<M> | M accepts 0^{374}}
- Is this easier?
  – in fact, yes, since L_{374} isn't even r.e., but $L_{accepts\text{-}374}$ is
  – but no, $L_{accepts\text{-}374}$ is not decidable either
- The same reduction works:
  – If $M(w)$ accepts, $L(M') = \{0^{374}\}$, so M' accepts 0^{374}
  – If $M(w)$ doesn't, $L(M') = \emptyset$, so M' doesn't accept 0^{374}
- More generally, telling whether or not a machine accepts any fixed string is undecidable

$L_{\neq\emptyset} = \{\langle M \rangle \mid L(M)$ is nonempty$\}$ is undecidable

- Given a TM M, telling whether it accepts *any* string is undecidable
- Proved by showing $L_u \leq L_{\neq\emptyset}$

$\langle M \rangle$ # w → instance of L_u

REDUCTION: BUILD M'

$\langle M' \rangle$ = → instance of $L_{\neq\emptyset}$

M': constants: M, w

On input x,

Run $M(w)$
Accept x if $M(w)$ accepts

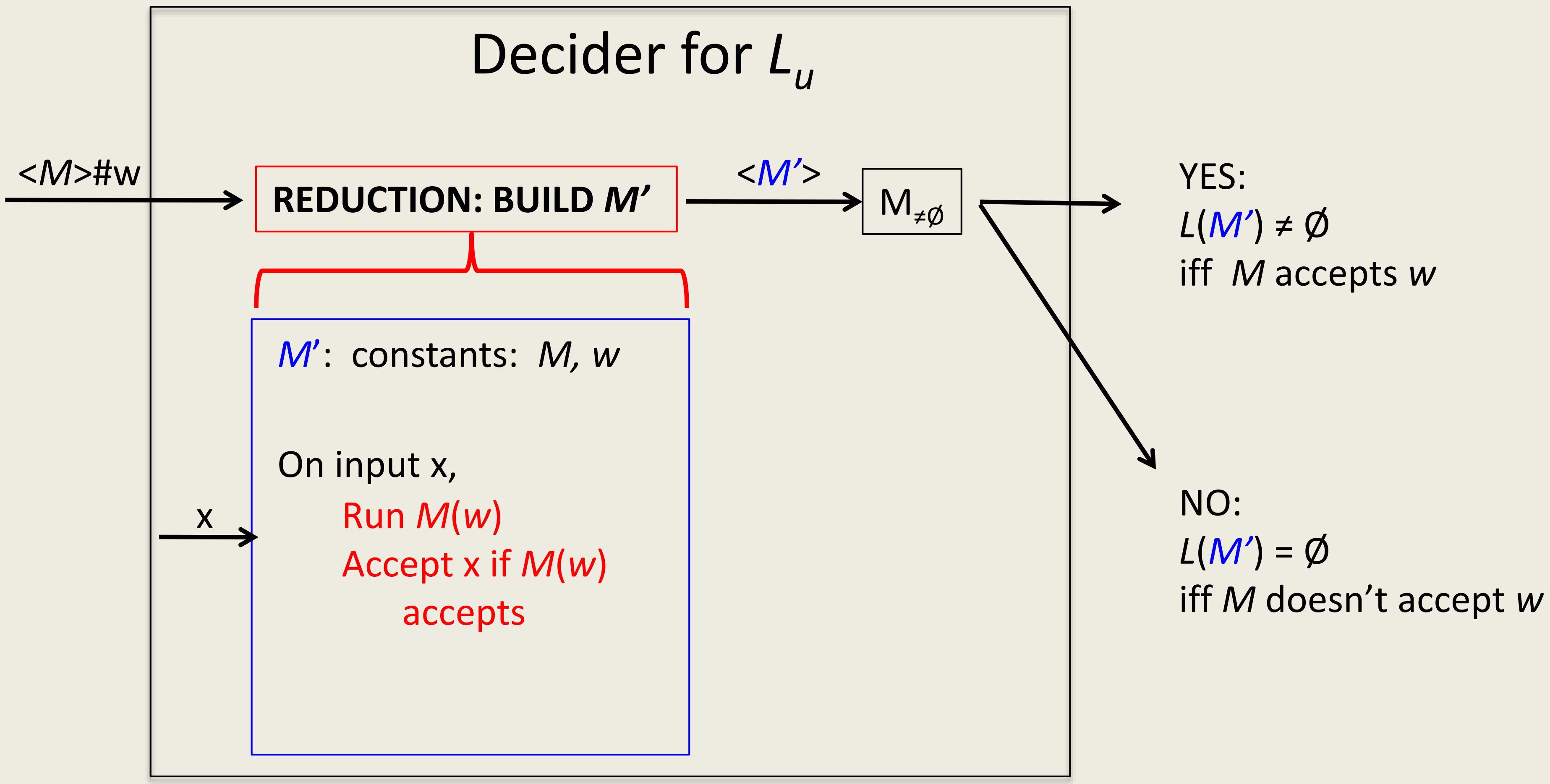
We want M' to satisfy:

- If $M(w)$ accepts, $L(M')$ $\neq \emptyset$
- If $M(w)$ doesn't $L(M')$ $= \emptyset$

What is L(M')?

If $M(w)$ accepts, $L(M') = \Sigma^*$ hence $\neq \emptyset$
If $M(w)$ doesn't, $L(M') = \emptyset$

If there is a decider $M_{\neq\emptyset}$ to tell if a TM accepts a nonempty language...



Since L_u is not decidable, $M_{\neq\emptyset}$ doesn't exist, and $L_{\neq\emptyset}$ is undecidable

L_{pal} = {<*M*> | $L(M)$ = palindromes} is undecidable

- Given a TM *M*, telling whether it accepts the set of palindromes is undecidable
- Proved by showing $L_u \leq L_{pal}$

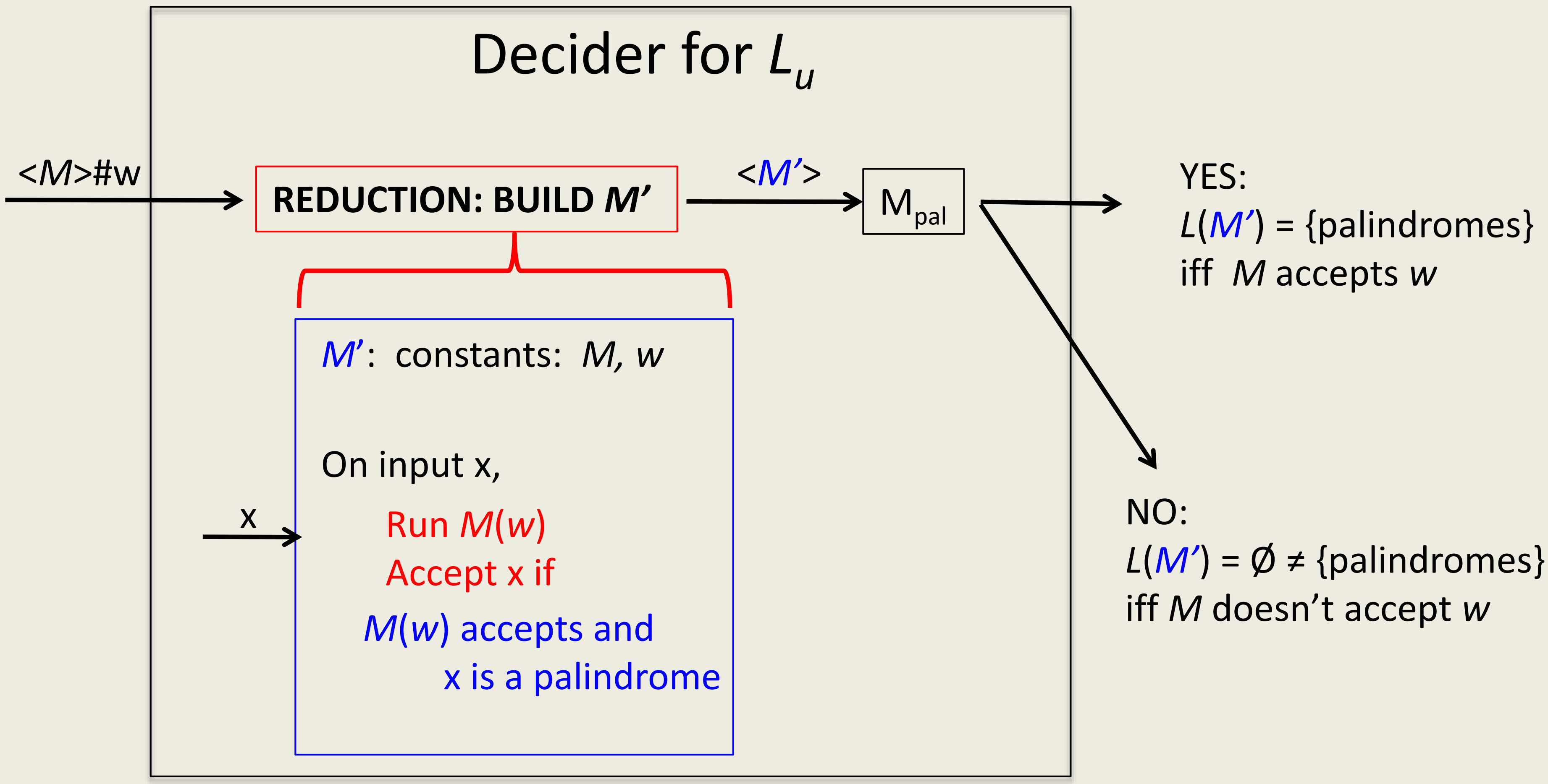
M’: constants: *M, w*

On input x,

Run *M*(*w*)
Accept x if
M(*w*) accepts and
x is a palindrome

We want *M’* to satisfy:
- If *M*(*w*) accepts, *L*(*M’*) = {palindromes}
- If *M*(*w*) doesn’t *L*(*M’*) ≠ {palindromes}

If there is a decider M_{pal} to tell if a TM accepts the set of palindromes

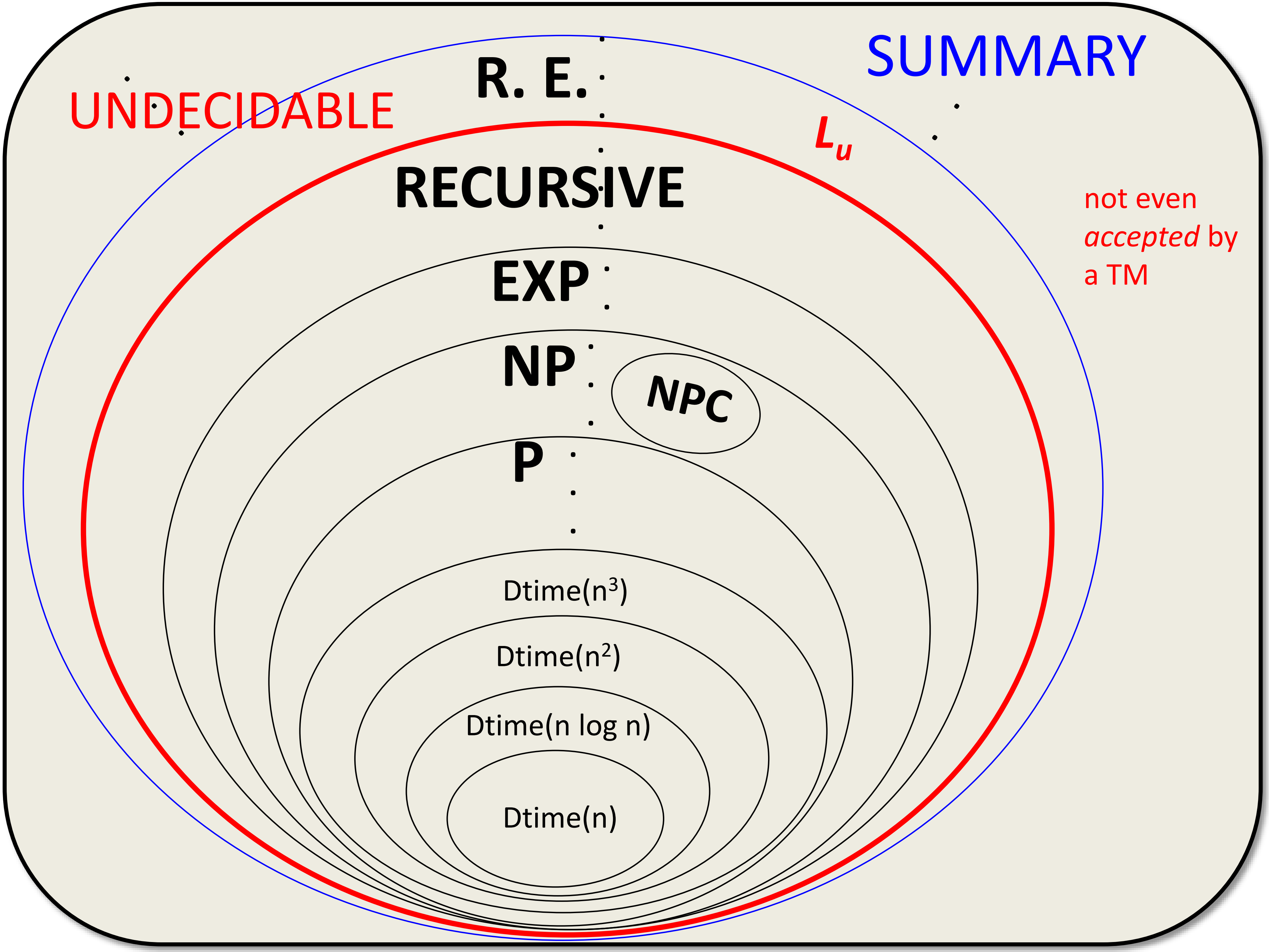


Since L_u is not decidable, M_{pal} *doesn't* exist, and L_{pal} is undecidable

Lots of undecidable problems about languages accepted by programs

- Given M, is $L(M)$ = {palindromes}?
- Given M, is $L(M) \neq \emptyset$?
- Given M, is $L(M) = \{0^{374}\}$ [redacted]
- Given M, does $L($ [redacted]
- Given M, is [redacted] e}?
- Given [redacted] n any prime?
- [redacted] contain *any word?*
- [redacted] s $L(M)$ meet these formal specs?
- G [redacted], does $L(M) = \Sigma^*$?

UNDECIDABLE


SUMMARY
UNDECIDABLE
R. E.
L_u
not even *accepted* by a TM
RECURSIVE
EXP
NP
NPC
P
Dtime(n³)
Dtime(n²)
Dtime(n log n)
Dtime(n)