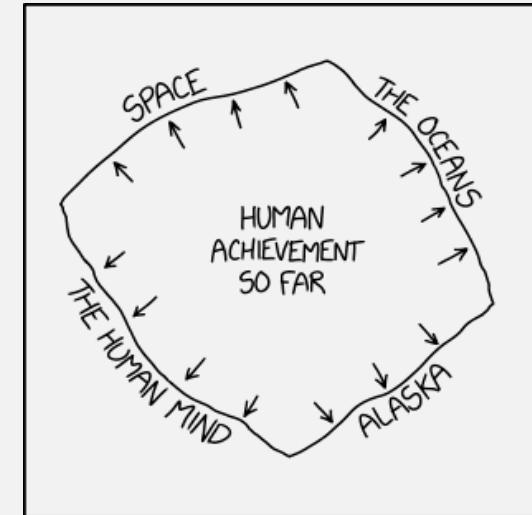


# Space ... and Beyond

Wednesday, May 11, 2022



FINAL REMAINING "FRONTIERS,"  
ACCORDING TO POPULAR USAGE

## *Announcements*

- HW 12 due tonight 11:59pm EST
  - Last HW!
- Last lecture!

# Previously: NP-Completeness

## DEFINITION

---

A language  $B$  is ***NP-complete*** if it satisfies two conditions:

1.  $B$  is in NP, and
2. every  $A$  in NP is polynomial time reducible to  $B$ .

These are the “hardest” problems (in NP) to solve

# NP-Completeness vs NP-Hardness

## DEFINITION

---

A language  $B$  is **NP-complete** if it satisfies two conditions:

1.  $B$  is in NP, and
2. every  $A$  in NP is polynomial time reducible to  $B$ .

“NP-Hard”

“NP-Complete” = in NP + “NP-Hard”

So a language can be NP-hard but not NP-complete!

# *Flashback:* The Halting Problem

$$\text{HALT}_{\text{TM}} = \{\langle M, w \rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$$

Thm:  $\text{HALT}_{\text{TM}}$  is undecidable

Proof, by contradiction:

- Assume  $\text{HALT}_{\text{TM}}$  has *decider*  $R$ ; use it to create decider for  $A_{\text{TM}}$ :
- ...
- But  $A_{\text{TM}}$  is undecidable and has no decider!

# *Flashback:* The Halting Problem

$$\text{HALT}_{\text{TM}} = \{\langle M, w \rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$$

Thm:  $\text{HALT}_{\text{TM}}$  is undecidable

Proof, by contradiction:

- Assume  $\text{HALT}_{\text{TM}}$  has *decider*  $R$ ; use it to create decider for  $A_{\text{TM}}$ :

$S$  = “On input  $\langle M, w \rangle$ , an encoding of a TM  $M$  and a string  $w$ :

1. Run TM  $R$  on input  $\langle M, w \rangle$ .
2. If  $R$  rejects, *reject*.  $\leftarrow$  This means  $M$  loops on input  $w$
3. If  $R$  accepts, simulate  $M$  on  $w$  until it halts.  $\leftarrow$  This step always halts
4. If  $M$  has accepted, *accept*; if  $M$  has rejected, *reject*.”

# *Flashback:* The Halting Problem

$$\text{HALT}_{\text{TM}} = \{\langle M, w \rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$$

Thm:  $\text{HALT}_{\text{TM}}$  is undecidable

Proof, by contradiction:

- Assume  $\text{HALT}_{\text{TM}}$  has *decider*  $R$ ; use it to create decider for  $A_{\text{TM}}$ :

~~$S = \text{"On input } \langle M, w \rangle, \text{ an encoding of a TM } M \text{ and a string } w:$~~

- ~~1. Run TM  $R$  on input  $\langle M, w \rangle$ .~~
- ~~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*.~~

- But  $A_{\text{TM}}$  is undecidable!

- I.e., this decider that we just created cannot exist! So  $\text{HALT}_{\text{TM}}$  is undecidable

# The Halting Problem is **NP**-Hard

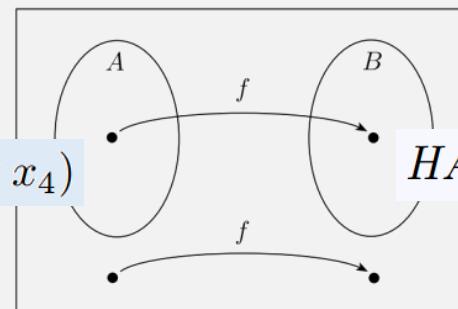
$HALT_{\text{TM}} = \{\langle M, w \rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$

Proof: Reduce  $3SAT$  to the Halting Problem

(Why does this prove that the Halting Problem is **NP**-hard?)

Because  $3SAT$  is **NP**-complete!  
(so every **NP** problem is poly time reducible to  $3SAT$ )

$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4)$



$HALT_{\text{TM}} = \{\langle M, w \rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$

# The Halting Problem is **NP-Hard**

$$HALT_{\text{TM}} = \{\langle M, w \rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$$

Computable function, from  $3SAT \rightarrow HALT_{\text{TM}}$ :

On input  $\phi$ , a formula in 3cnf:

- Construct TM  $M$

$M$  = on input  $\phi$

- Try all assignments
  - If any satisfy  $\phi$ , then accept
  - When all assignments have been tried, start over

This loops when there is no satisfying assignment!

- Output  $\langle M, \phi \rangle$

$\Rightarrow$  If  $\phi$  has a satisfying assignment, then  $M$  halts on  $\phi$   
 $\Leftarrow$  If  $\phi$  has no satisfying assignment, then  $M$  loops on  $\phi$

# Review:

## DEFINITION

A language  $B$  is **NP-complete** if it satisfies two conditions:

- 1.  $B$  is in NP, and
- 2. every  $A$  in NP is polynomial time reducible to  $B$ .

So a language can satisfy condition #2 but not condition #1

But can a language satisfy condition #1 but not condition #2?

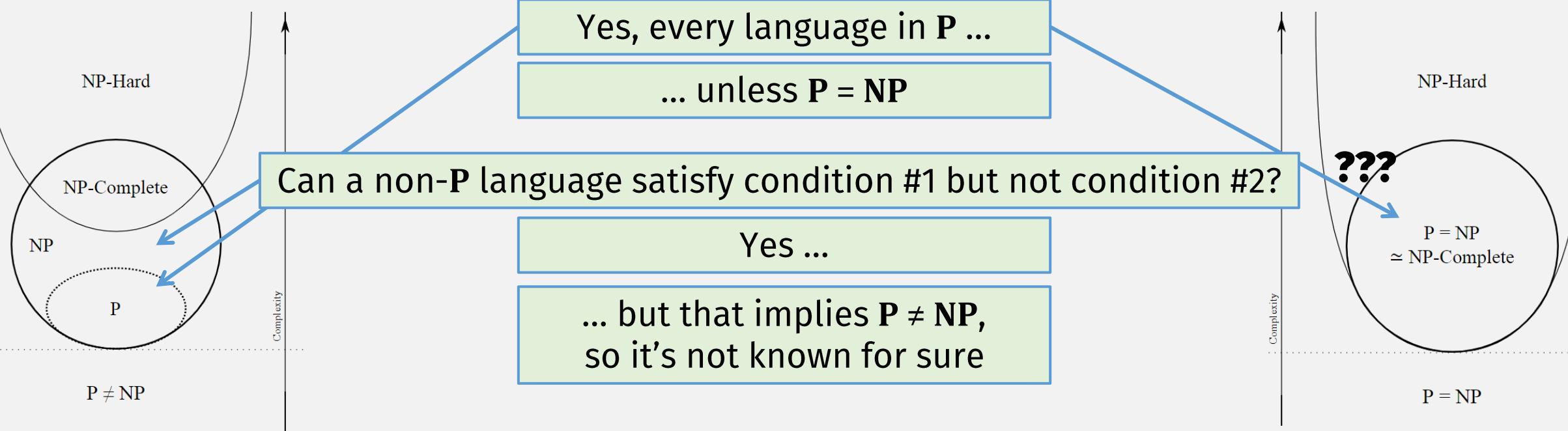
Yes, every language in P ...

... unless  $P = NP$

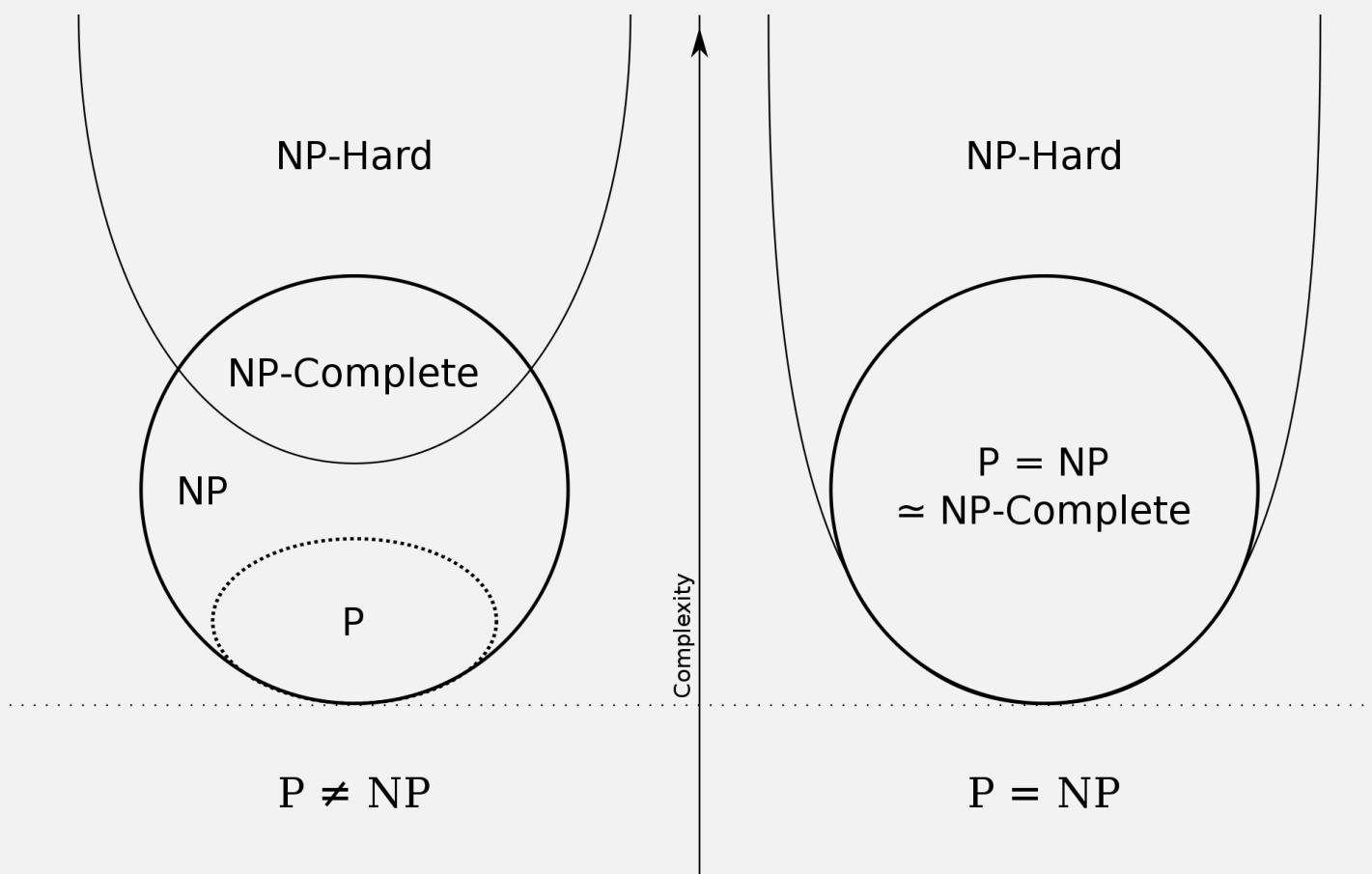
Can a non-P language satisfy condition #1 but not condition #2?

Yes ...

... but that implies  $P \neq NP$ ,  
so it's not known for sure



# NP-Completeness vs NP-Hardness



Space ...



# *Flashback:* Dynamic Programming Example

- Chomsky Grammar  $G$ :
  - $S \rightarrow AB \mid BC$
  - $A \rightarrow BA \mid a$
  - $B \rightarrow CC \mid b$
  - $C \rightarrow AB \mid a$
- Example string: **baaba**
- Store every partial string and their generating variables in a table

We are gaining time ...

... by spending more space!

Substring end char

	b	a	a	b	a
b	vars for “b”	vars for “ba”	vars for “baa”	...	
a		vars for “a”	vars for “aa”	vars for “aab”	
a			...		
b					
a					

Substring  
start char

# Space Complexity, Formally

TMs have a space complexity

## DEFINITION

---

Let  $M$  be a deterministic Turing machine that halts on all inputs. The *space complexity* of  $M$  is the function  $f : \mathcal{N} \rightarrow \mathcal{N}$ , where  $f(n)$  is the maximum number of tape cells that  $M$  scans on any input of length  $n$ . If the space complexity of  $M$  is  $f(n)$ , we also say that  $M$  runs in space  $f(n)$ .

If  $M$  is a nondeterministic Turing machine wherein all branches halt on all inputs, we define its space complexity  $f(n)$  to be the maximum number of tape cells that  $M$  scans on any branch of its computation for any input of length  $n$ .

---

# Space Complexity Classes

TMs have a **space complexity**

Languages are in a **space complexity class**

## DEFINITION

---

Let  $f: \mathcal{N} \rightarrow \mathcal{R}^+$  be a function. The **space complexity classes**,  $\text{SPACE}(f(n))$  and  $\text{NSPACE}(f(n))$ , are defined as follows.

$\text{SPACE}(f(n)) = \{L \mid L \text{ is a language decided by an } O(f(n)) \text{ space deterministic Turing machine}\}.$

$\text{NSPACE}(f(n)) = \{L \mid L \text{ is a language decided by an } O(f(n)) \text{ space nondeterministic Turing machine}\}.$

---

## Compare:

Let  $t: \mathcal{N} \rightarrow \mathcal{R}^+$  be a function. Define the **time complexity class**,  $\text{TIME}(t(n))$ , to be the collection of all languages that are decidable by an  $O(t(n))$  time Turing machine.

$\text{NTIME}(t(n)) = \{L \mid L \text{ is a language decided by an } O(t(n)) \text{ time nondeterministic Turing machine}\}.$

# Example: SAT Space Usage

$$SAT = \{\langle \phi \rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$$

$2^{O(m)}$  exponential  
time machine

$M_1$  = “On input  $\langle \phi \rangle$ , where  $\phi$  is a Boolean formula:

1. For each truth assignment to the variables  $x_1, \dots, x_m$  of  $\phi$ :
2. Evaluate  $\phi$  on that truth assignment.
3. If  $\phi$  ever evaluated to 1, *accept*; if not, *reject*.“

Each loop iteration requires  $O(m)$  space

But the space is re-used on each loop!  
(nothing is stored from the last loop)

So the entire machine only needs  $O(m)$  space!

# Example: Nondeterministic Space Usage

$$ALL_{\text{NFA}} = \{\langle A \rangle \mid A \text{ is an NFA and } L(A) = \Sigma^*\}$$

## Nondeterministic decider for $\overline{ALL}_{\text{NFA}}$

$N =$  “On input  $\langle M \rangle$ , where  $M$  is an NFA:

1. Place a marker on the start state of the NFA.
2. Repeat  $2^q$  times, where  $q$  is the number of states of  $M$ :
3. Nondeterministically select an input symbol and change the positions of the markers on  $M$ 's states to simulate reading that symbol.
4. Accept if stages 2 and 3 reveal some string that  $M$  rejects; that is, if at some point none of the markers lie on accept states of  $M$ . Otherwise, reject.”

Additionally,  
need a **counter**  
to count to  $2^q$  ...

... requires  
 $\log(2^q) = q$   
extra space

Machine tracks  
“current” states of NFA:  
 $q$  states =  $2^q$  possible  
combinations  
(so exponential time)

Each loop uses only  
 $O(q)$  space!

So the whole machine runs in (nondeterministic) linear  $O(q)$  space!

# *Flashback:* TM Variations and Time

- If a multi-tape TM runs in:  $t(n)$  time
- Then an equivalent single-tape TM runs in:  $O(t^2(n))$ 
  - Quadratically slower
- If a non-deterministic TM runs in:  $t(n)$  time
- Then an equivalent deterministic TM runs in:  $2^{O(t(n))}$ 
  - Exponentially slower

What about space?

# TM Variations and Space

## THEOREM

**Savitch's theorem** For any function  $f: \mathcal{N} \rightarrow \mathcal{R}^+$ , where  $f(n) \geq n$ ,

$$\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n)).$$

- If a non-deterministic TM runs in:  $f(n)$  space
- Then an equivalent deterministic TM runs in:  $f^2(n)$  space
  - ~~Exponentially~~ Only Quadratically slower!

# Flashback: Nondet. TM $\rightarrow$ Deterministic TM

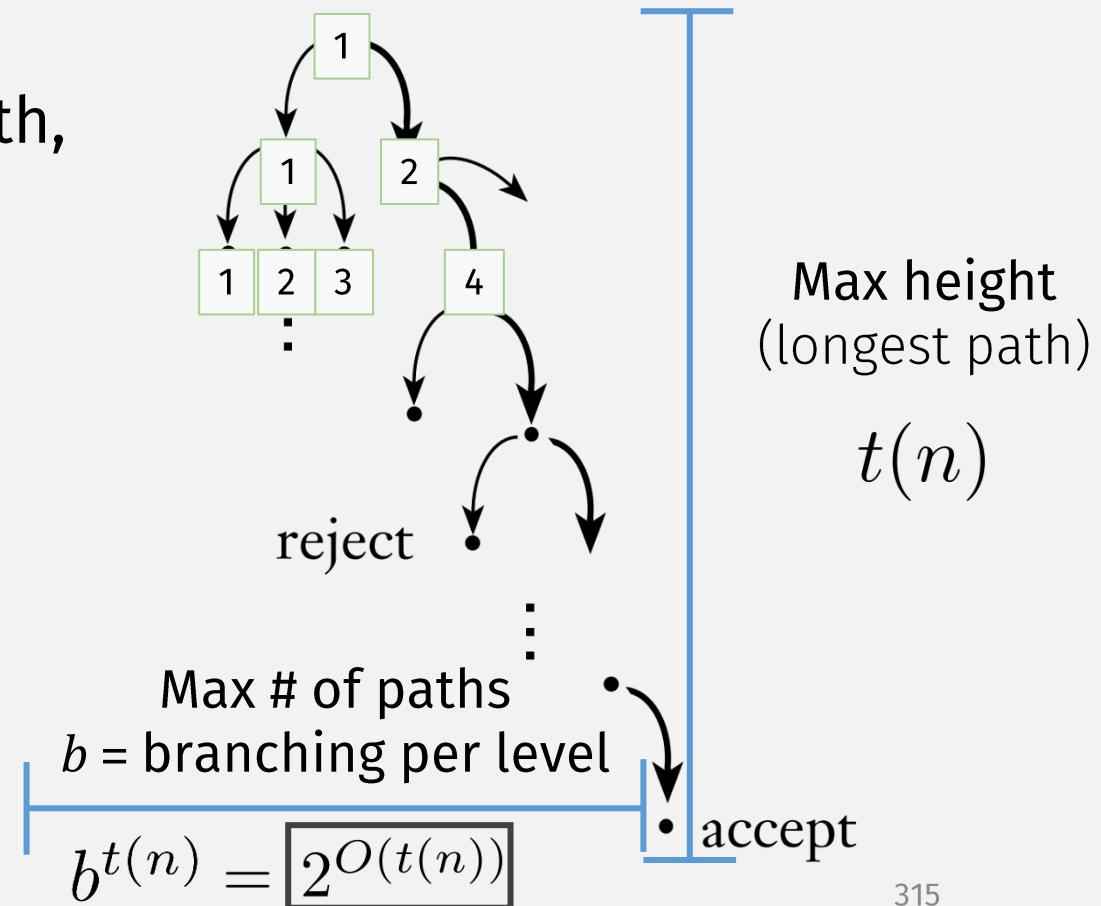
$t(n)$  time  $\xrightarrow{} 2^{O(t(n))}$  time

- Simulate NTM with Det. TM:

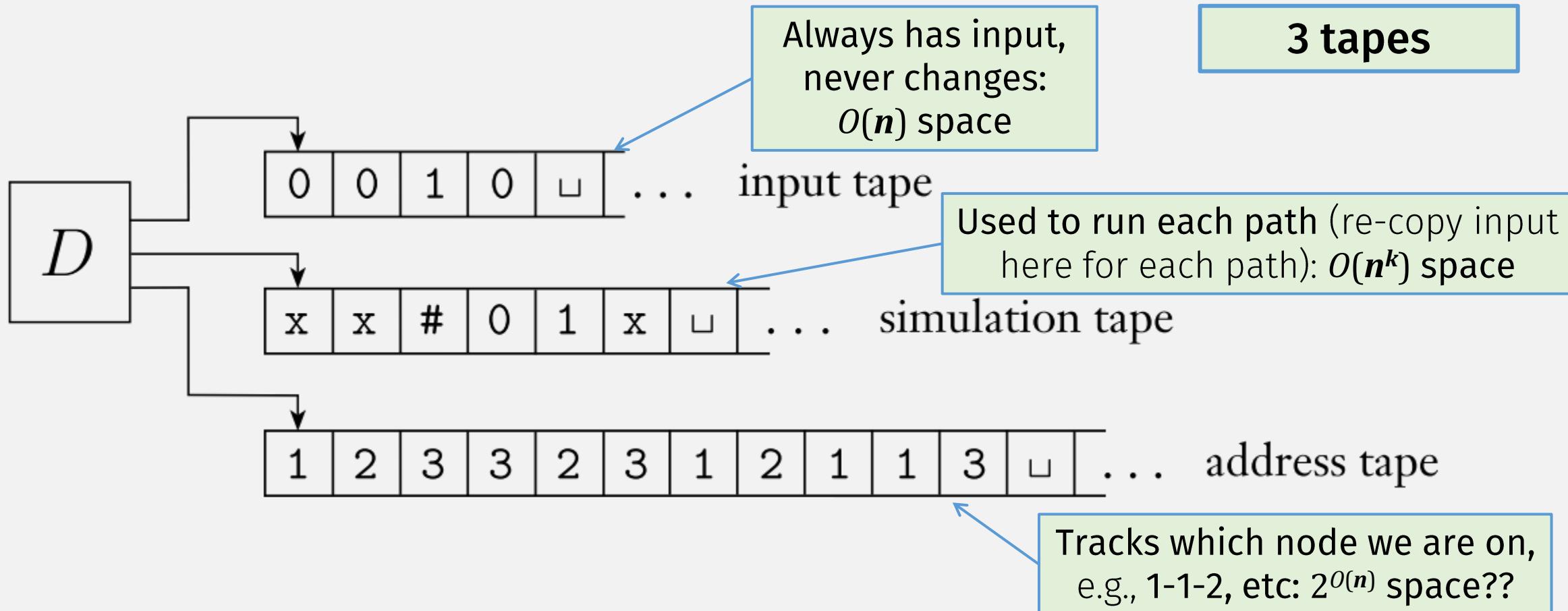
- Number the nodes at each step
- Deterministically check every tree path, in breadth-first order

- 1
- 1-1
- 1-2
- 1-1-1

Nondeterministic computation



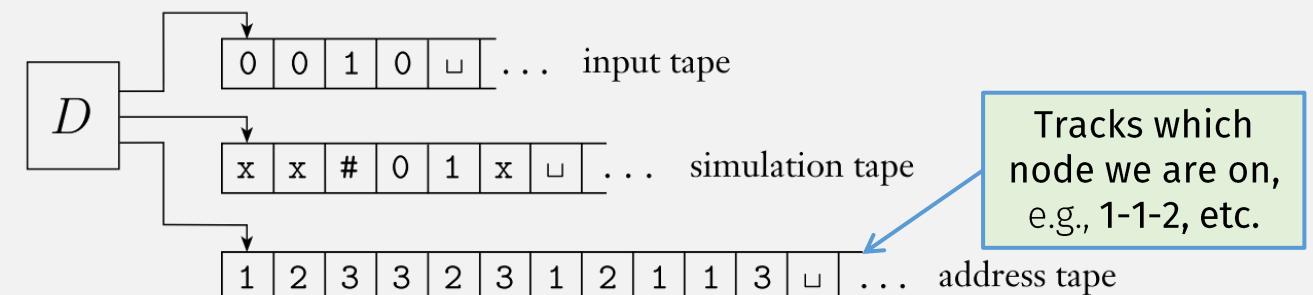
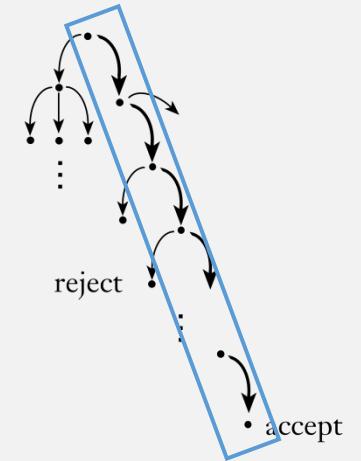
# Flashback: NTM $\rightarrow$ Deterministic



# NTM→Deterministic TM: Space Version

Let  $N$  be an NTM deciding language  $A$  in space  $f(n)$

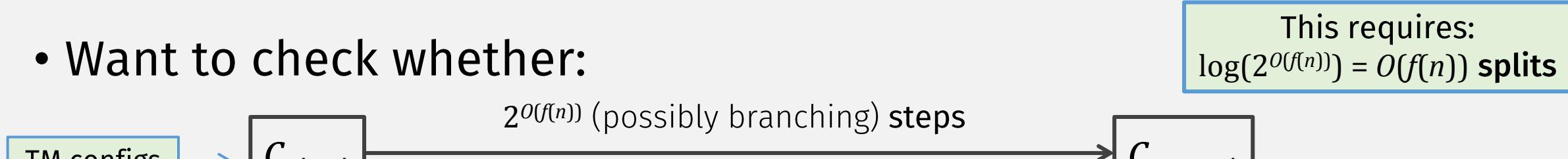
- So a single path could use  $f(n)$  space
- That path could take  $2^{O(f(n))}$  steps
  - (That's the possible ways to fill the space)
  - Where each step could be a branch
- So naïvely tracking these branches requires  $2^{O(f(n))}$  space!



- Instead, let's “divide and conquer” to save space!

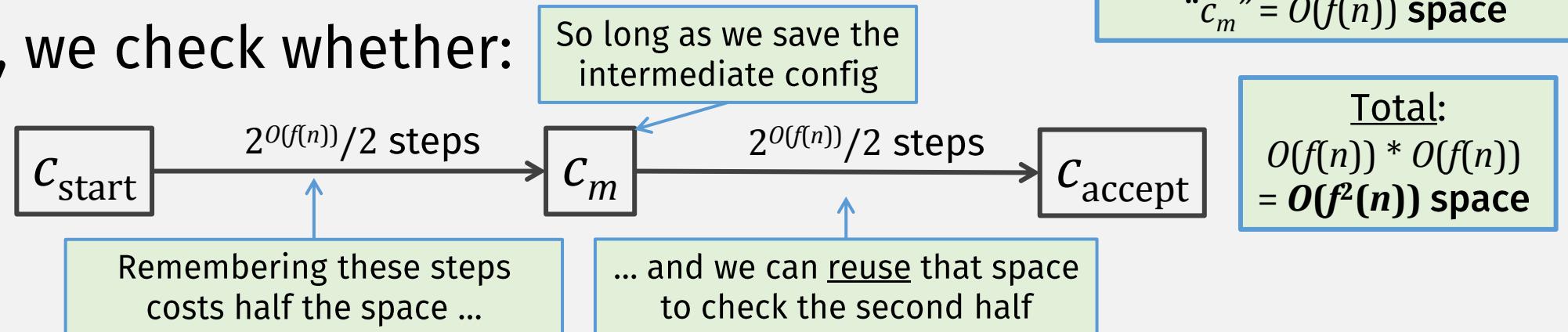
# “Divide and Conquer” TM Config Sequences

- Want to check whether:



This requires:  
 $\log(2^{O(f(n))}) = O(f(n))$  splits

- Instead, we check whether:

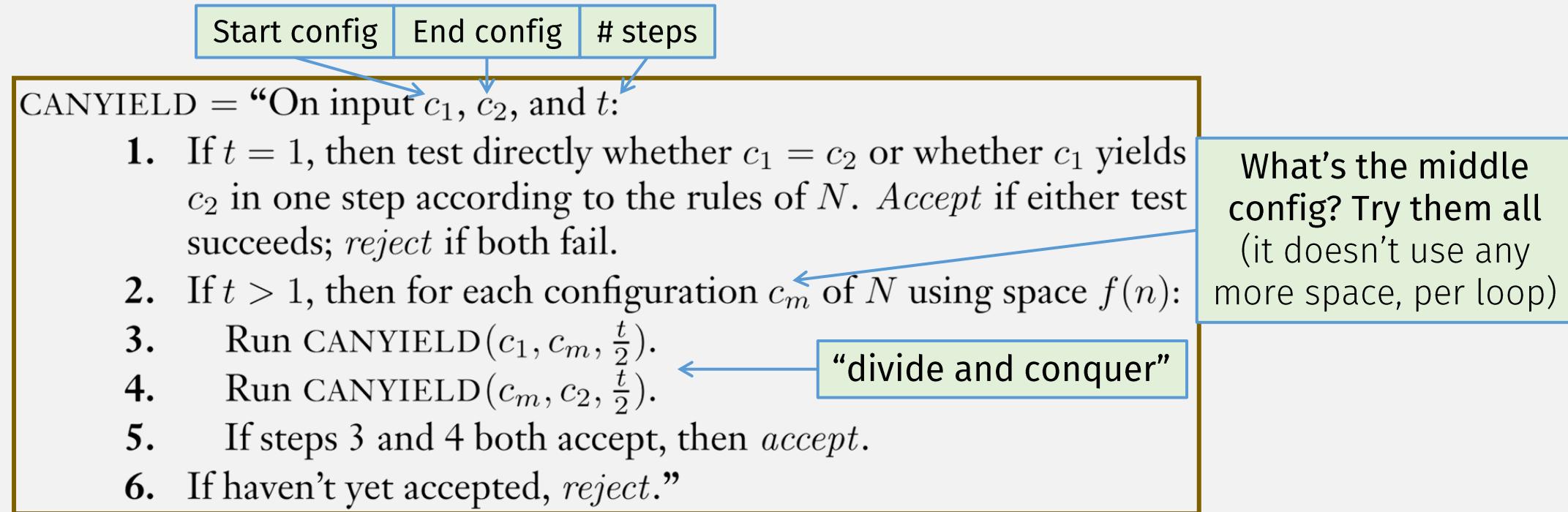


Total:  
 $O(f(n)) * O(f(n))$   
 $= O(f^2(n))$  space

- Keep dividing ...



# Formally: A “Yielding” Algorithm



# Savitch's Theorem: Proof

- Let  $N$  be an NTM deciding language  $A$  in space  $f(n)$
- Construct equivalent deterministic TM  $M$  using  $O(f^2(n))$  space:

$M$  = “On input  $w$ :

1. Output the result of CANYIELD( $c_{\text{start}}, c_{\text{accept}}, 2^{df(n)}$ ).”

Extra  $d$  constant  
depends on size  
of tape alphabet

- $c_{\text{start}}$  = start configuration of  $N$
- $c_{\text{accept}}$  = new accepting config where all  $N$ 's accepting configs go

# PSPACE

## DEFINITION

---

**PSPACE** is the class of languages that are decidable in polynomial space on a deterministic Turing machine. In other words,

$$\text{PSPACE} = \bigcup_k \text{SPACE}(n^k).$$

# NPSPACE

## DEFINITION

---

**NPSPACE** is the class of languages that are decidable in polynomial space on **non** deterministic Turing machine. In other words,

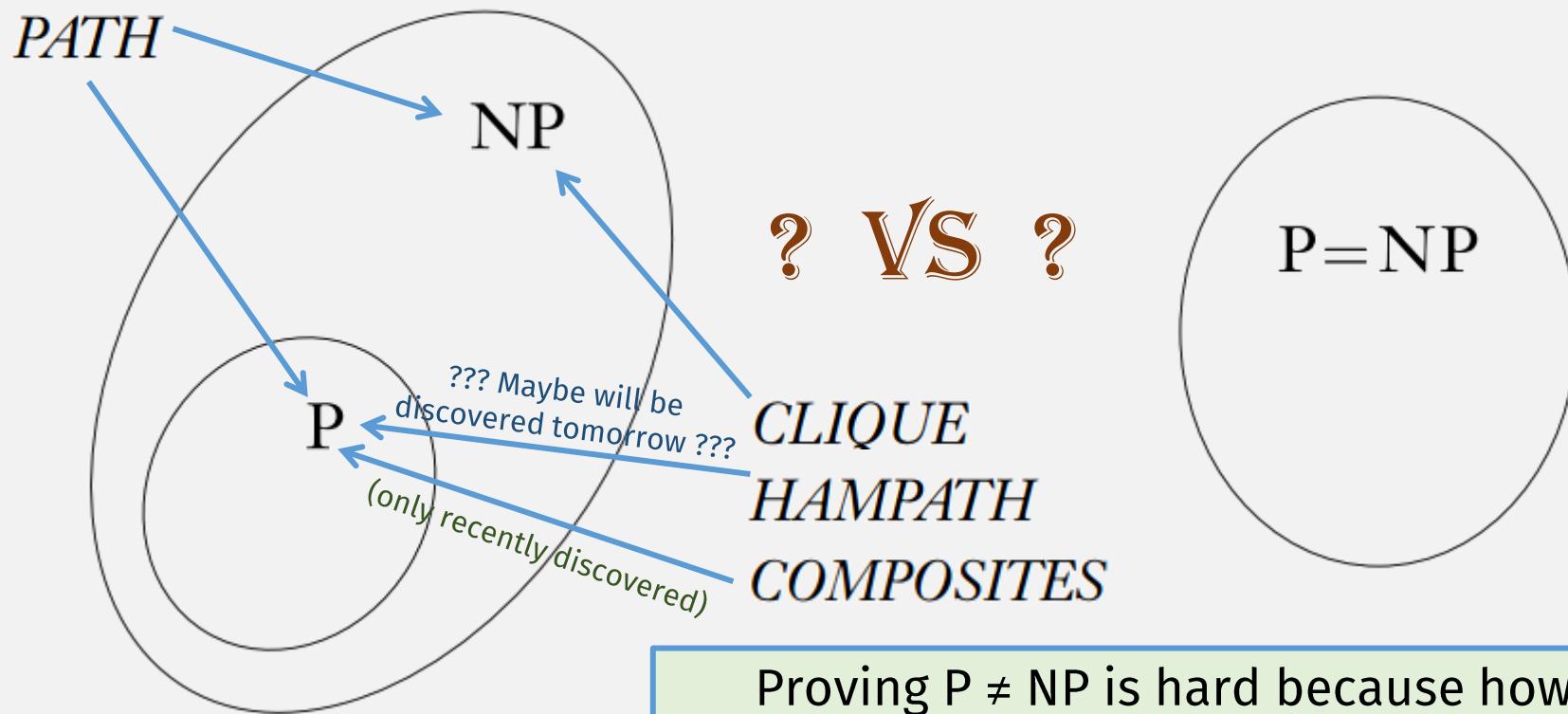
$$\text{NPSPACE} = \bigcup_k \text{NSPACE}(n^k).$$

Analogous to **P** and **NP** for time complexity

# PSPACE vs NPSPACE

- **PSPACE**: langs decidable in poly space on deterministic TM
- **NPSPACE**: langs decidable in poly space on nondeterministic TM

# *Flashback:* Does P = NP?



Proving  $P \neq NP$  is hard because how do you prove an algorithm doesn't have a poly time algorithm?  
(in general it's hard to prove that something doesn't exist)

# PSPACE vs NPSPACE

- **PSPACE**: langs decidable in poly space on deterministic TM
- **NPSPACE**: langs decidable in poly space on nondeterministic TM

Theorem: PSPACE = NPSPACE !!!

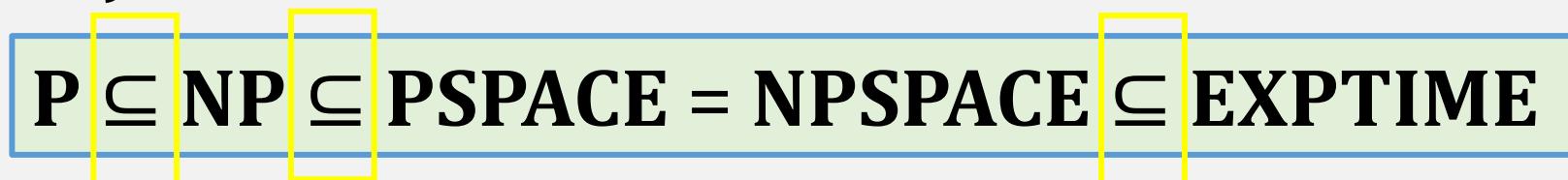
Proof: By Savitch's Theorem!

**THEOREM** .....

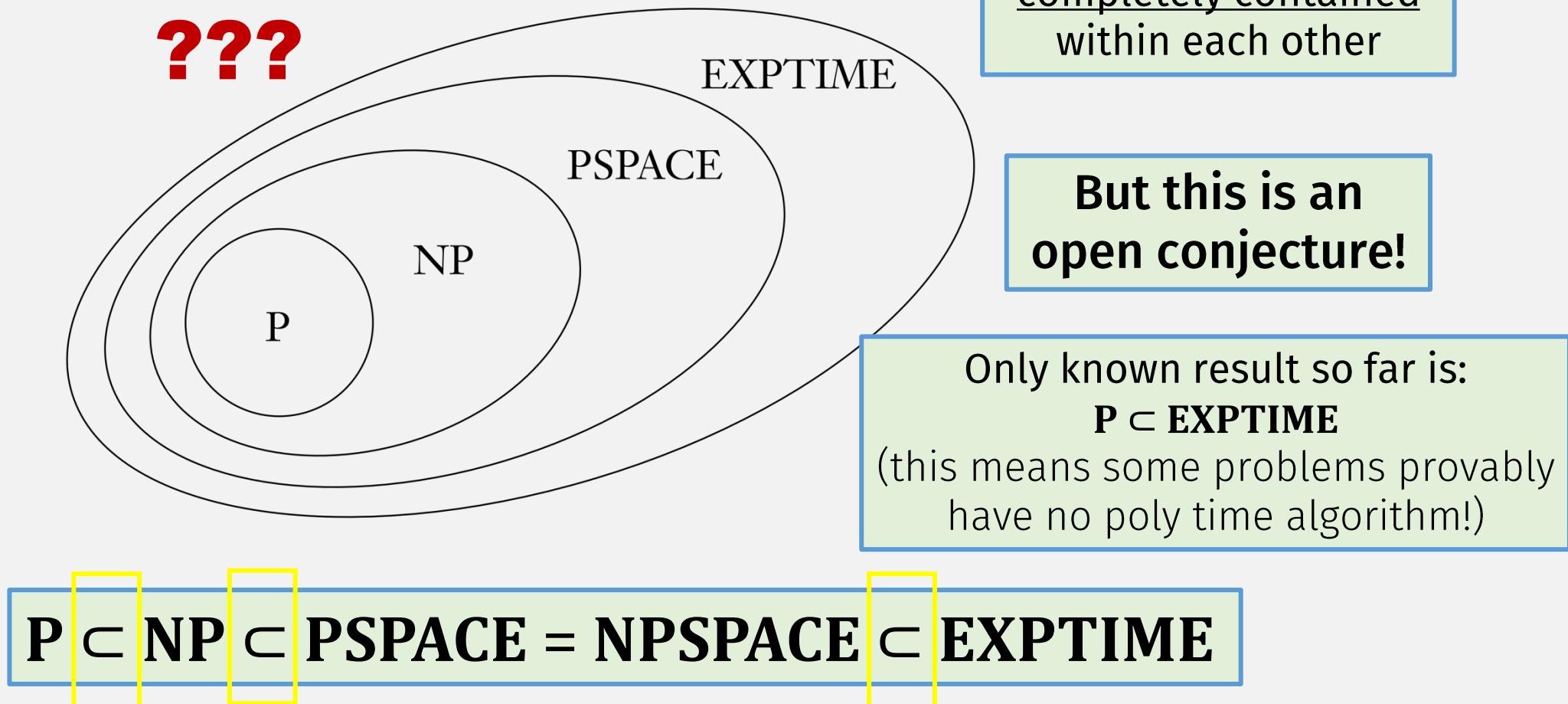
**Savitch's theorem** For any function  $f: \mathcal{N} \rightarrow \mathcal{R}^+$ , where  $f(n) \geq n$ ,  
 $\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n))$ .

# Space vs Time

- **P  $\subseteq$  PSPACE** and **NP  $\subseteq$  NPSPACE**
  - Because each step can use at most one extra tape cell
  - And space can be re-used
- **PSPACE  $\subseteq$  EXPTIME**
  - Because an  $f(n)$  space TM has  $2^{O(f(n))}$  possible configurations
  - And a halting TM cannot repeat a configuration
- We already know **P  $\subseteq$  NP** and **PSPACE = NPSPACE** ... so:



# Space vs Time: Conjecture



# Last Quiz 5/11

In gradescope