CSCI 361 Lecture 18: NP hardness Reductions

Shikha Singh

Announcements & Logistics

- **HW 9** will be released today
  - Can work in pairs and submit a single write up
  - Due **next Thursday Nov 21** at 10 pm
- Reminder: I am out of town next week, Sam will cover for me
- Created virtual office hour 10 min slots you can reserve
  - Tues/Wed/Thur 2-3 pm EST: https://calendar.app.google/oLLLwVw4uMjrwFxy7
  - Please reserve 1 hr ahead of the meeting slot
  - You can also ask Sam questions at the end of lecture

Last Time: Recall NP Definitions

- **Class P.** A language is in P iff there exists a deterministic TM that decides it in time $O(n^k)$ for some constant k.
- **Class NP (Definition 1).** A language is in NP iff there exists a non-deterministic TM that decides it in time $O(n^k)$ for some constant k.
- **Class NP (Definition 2).** A language A is in NP iff it has a polynomial-time verifier V, that is,
  - For each input w, there exists a polynomial-size certificate c s.t. V accepts $\langle w, c \rangle$ iff $w \in A$ and V runs in polynomial time

Last Time: P versus NP restated

- **Turing machine language.** For every language that can be decided in polynomial time by an NTM, can it be decided in polynomial time by a DTM?
  - Only method we know how to simulate a NTM using a DTM, we incur an exponential blow up in running time
- **Algorithms language.** Can every decision problem which can be verified efficiently (in poly time), can it also be solved quickly?
- Researchers believe that

Recall: Status of What We Know

- P: class of languages whose membership can be ***decided*** quickly
- NP: class of languages whose membership can be ***verified*** quickly

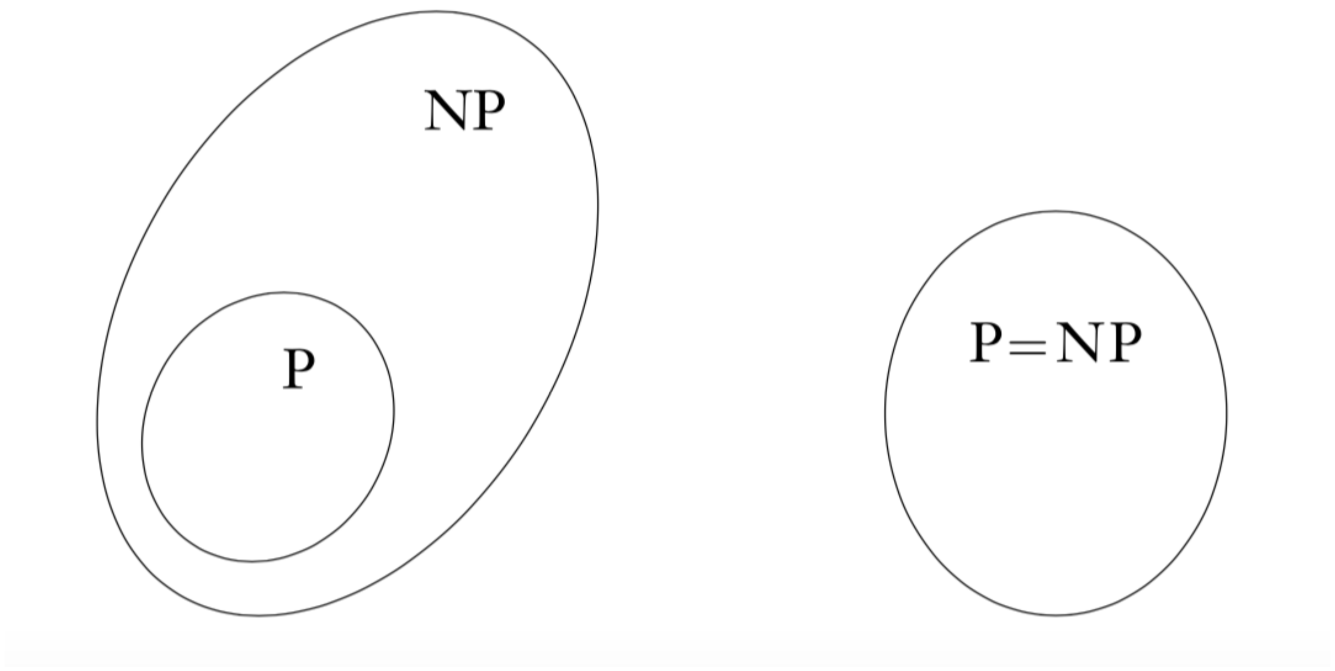
- **Theorem.** $\mathrm{P} \subseteq \mathrm{NP}$

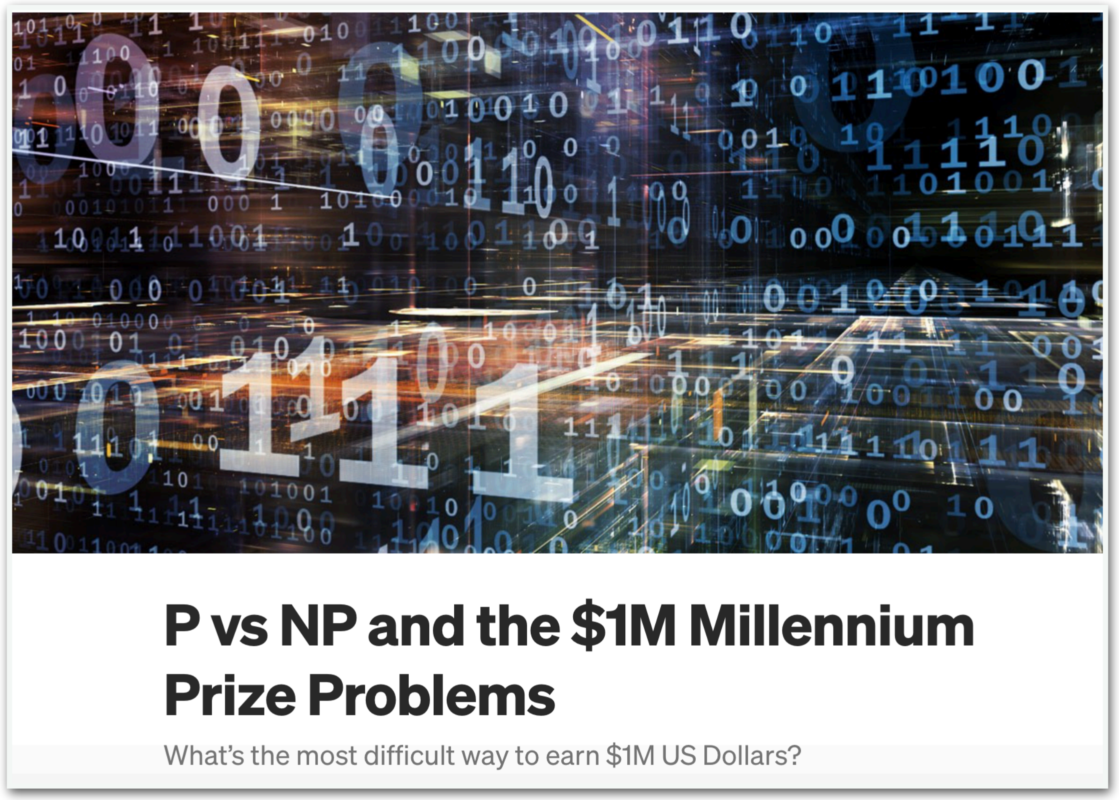
- **Theorem.** $\mathrm{NP} \subseteq \bigcup_{k} \mathrm{TIME}(2^{n^k}) = \mathrm{EXPTIME}$

- We don't know whether either of these containment are **strict**

What We Don't Know: P = NP?

- A problem that can be verified quickly, can it also be decided quickly?
- Is **P = NP** ?
  - Million Dollar Question





P vs NP and the $1M Millennium Prize Problems

What's the most difficult way to earn $1M US Dollars?

"*None of us truly understand the P versus NP problem, we have only begun to peel the layers around this increasingly complex question. Perhaps we will see a resolution of the P versus NP problem in the near future but I almost hope not. The P versus NP problem continues to inspire and boggle the mind and continued exploration of this problem will lead us to yet even new complexities in that truly mysterious process we call computation.*"
- Fortnow 2024

Recent CACM article by Lance Fortnow: https://www.cs.cmu.edu/~15326-f24/CACM-Fortnow.pdf

Hardest Problem in NP

- **SAT.** Consider a Boolean formulae ϕ with **variables** or their negations connected by AND ($\wedge$) and OR ($\vee$) and NOT. Here a **literal** is a variable or its negation. The language is defined as follows:

  SAT = $\{\phi \mid \phi$ has a satisfying assignment$\}$

- E.g. $\phi = (\overline{x_1} \vee x_2, \vee x_3) \wedge (x_1, \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$

  $\phi \in$ SAT because it has a satisfying assignment
  $x_1 = 1,\ x_2 = 1,\ x_3 = 0,\ x_4 = 0$

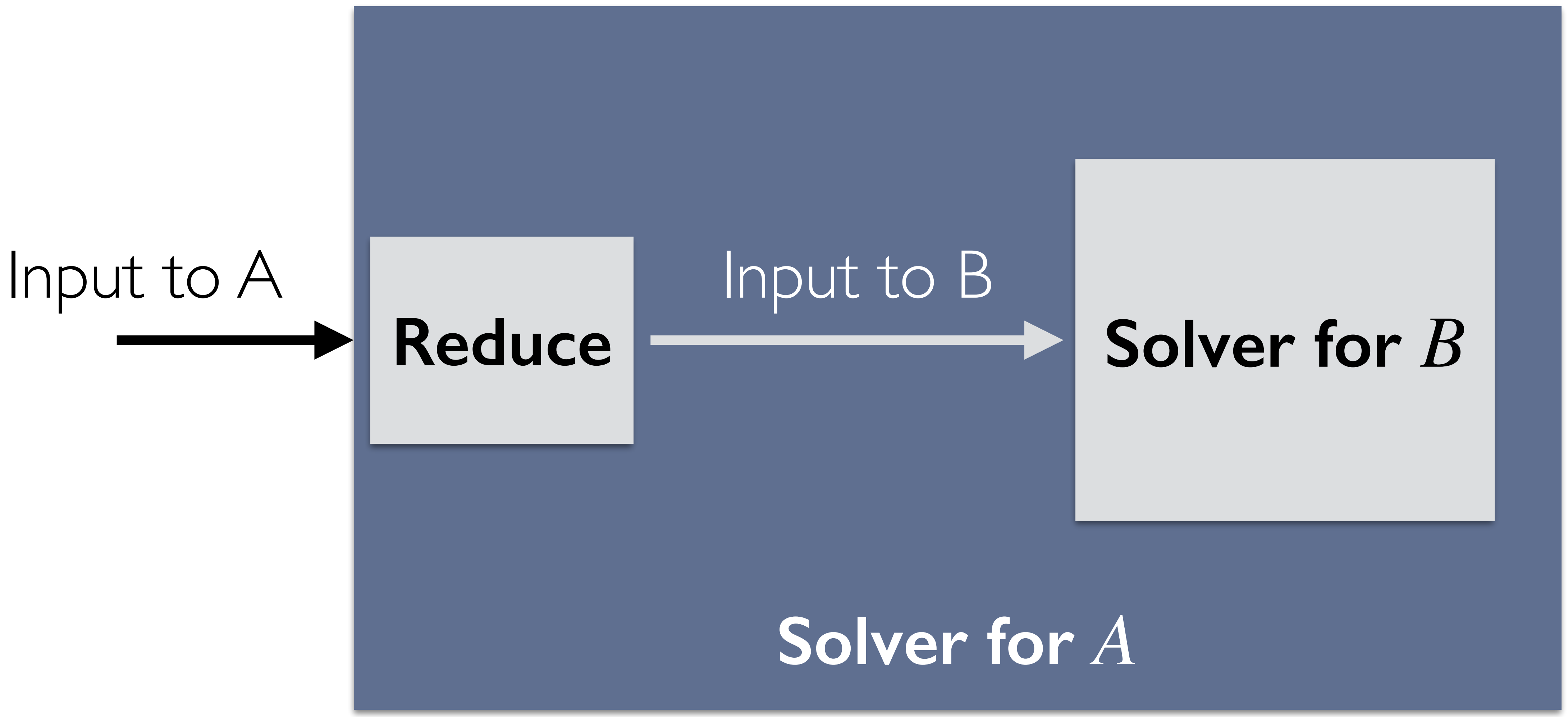
- **Cook-Levin Theorem.** If SAT $\in$ P, then **P** $=$ **NP**

Today: Classifying Problems

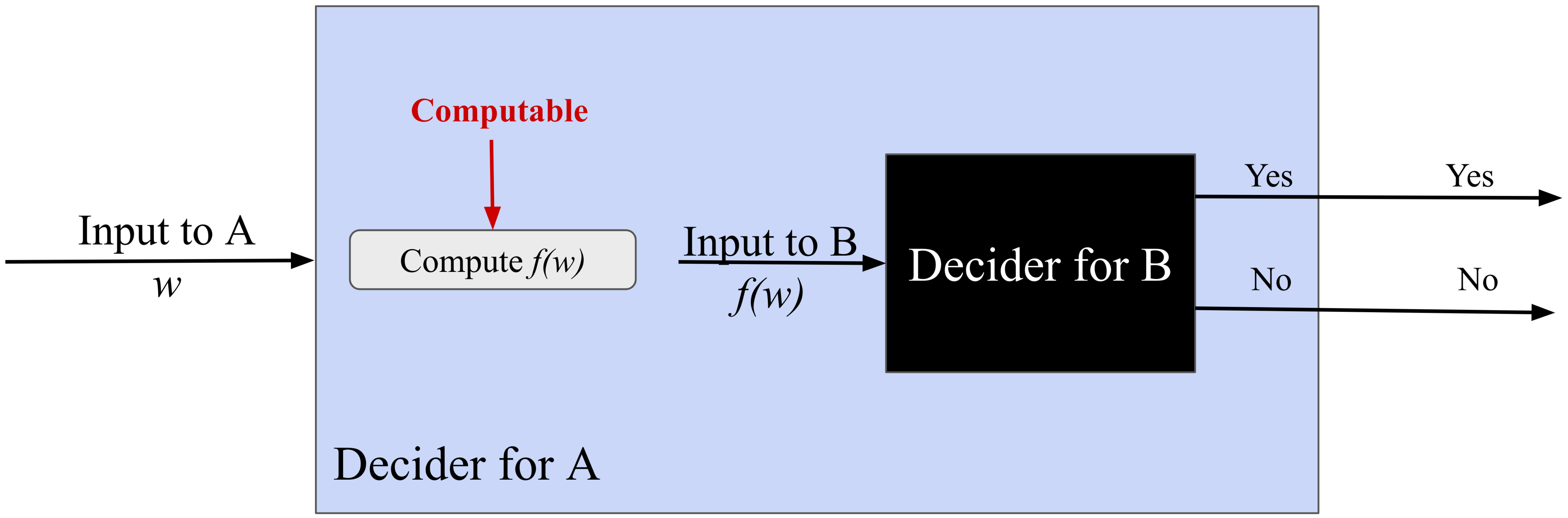
- Instead, we say problem X is likely very hard to solve by comparing it to a known hard-to-solve problem
- This is similar to how we proved undecidability of languages

Recall Reductions

- If a problem $\boldsymbol{A}$ reduces to problem $\boldsymbol{B}$, then solving $\boldsymbol{A}$ is no harder than solving $\boldsymbol{B}$

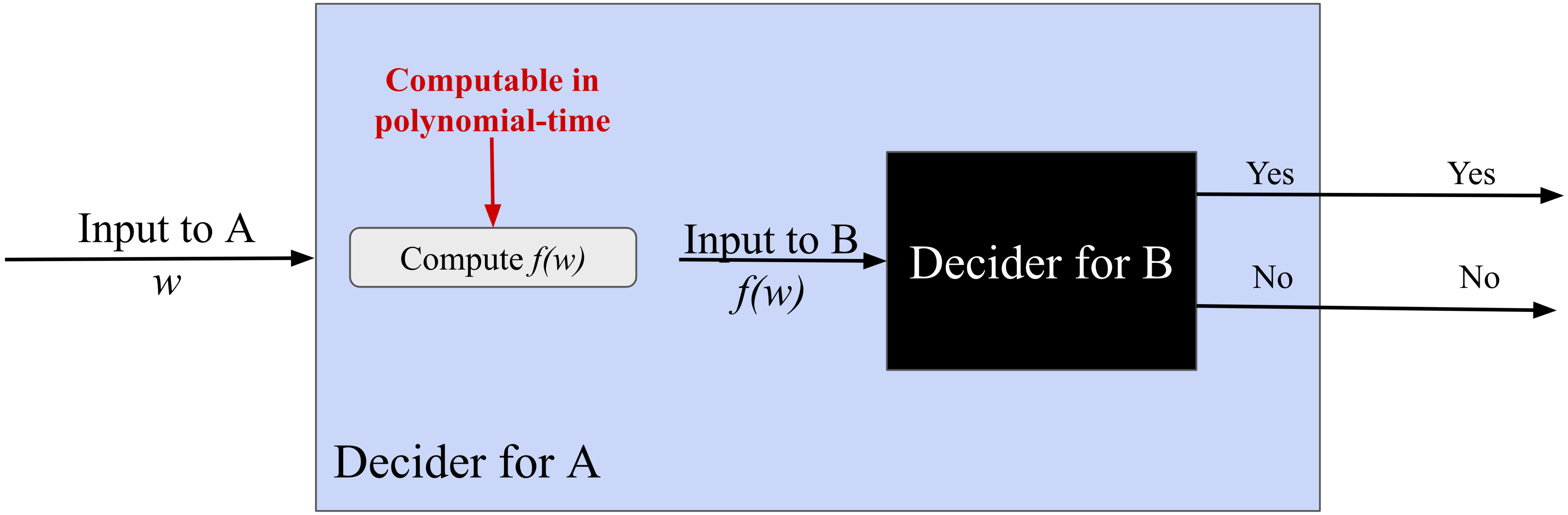


So Far: Mapping Reductions



$$A \leq_m B$$

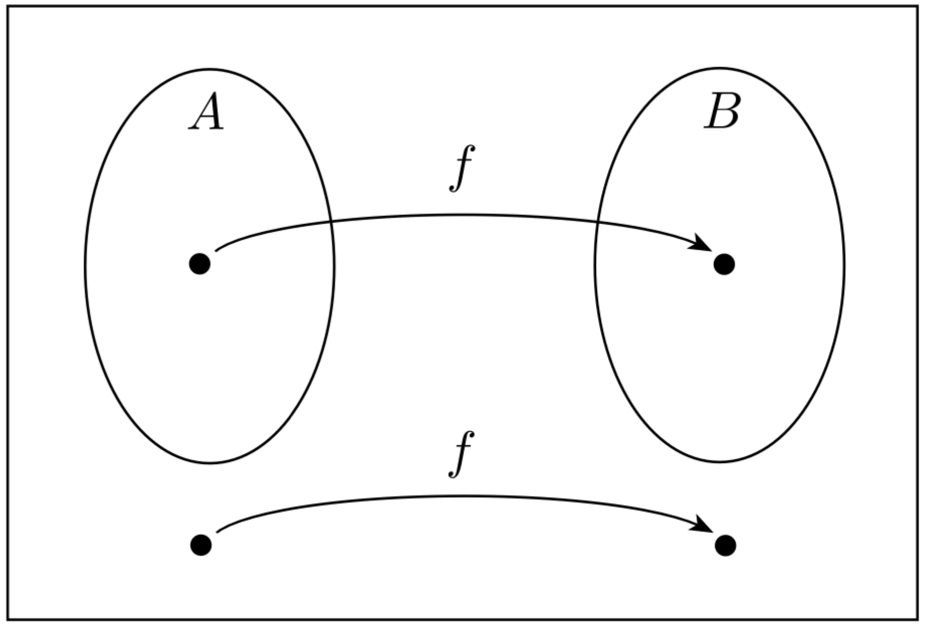
Now: Polynomial-Time Reductions



$$A \leq_p B$$

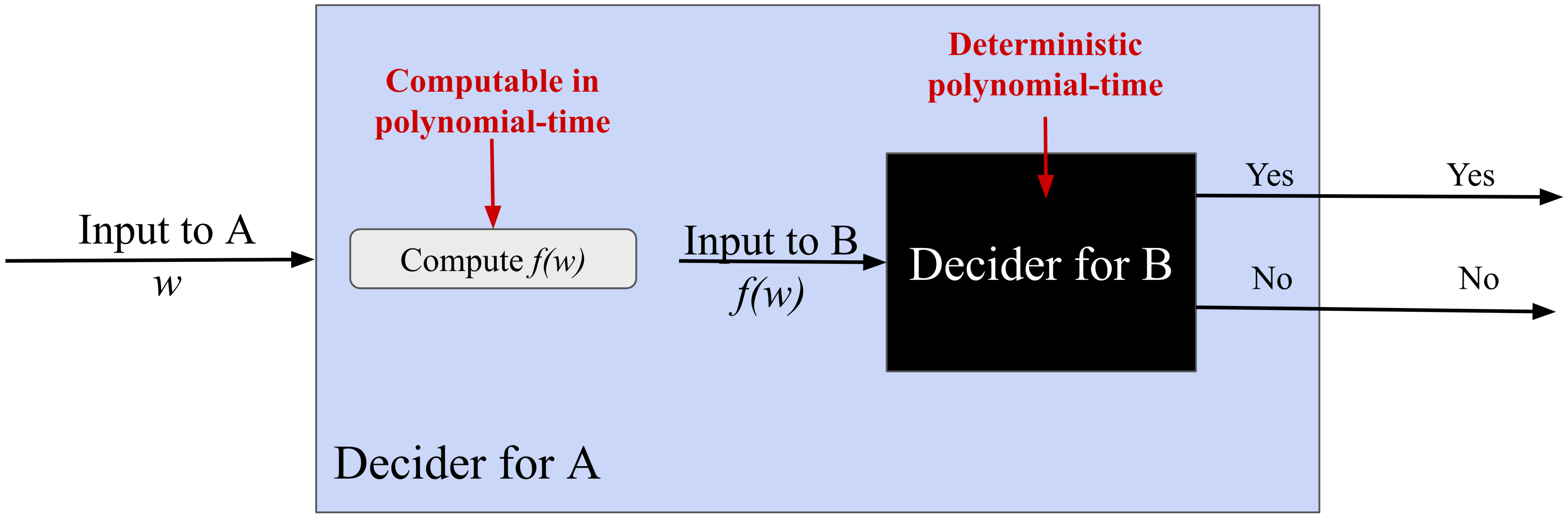
Polynomial-time Reductions

- Similar to mapping reductions but now we care about efficiency
- **Definition.** Language A is polynomial-time reducible to language B, denoted $A \leq_p B$, if there exists a ***polynomial-time computable*** function $f : \Sigma^* \to \Sigma^*$, such that
  $$w \in A \iff f(w) \in B \quad \text{for every } w$$
- The function f is the **polynomial-time reduction** from A to B



Using Reductions: Positive Use

- Using reductions to prove **membership** in P:
  - **Theorem.** If $A \leq_p B$ and $B \in$ P, then $A \in$ P.

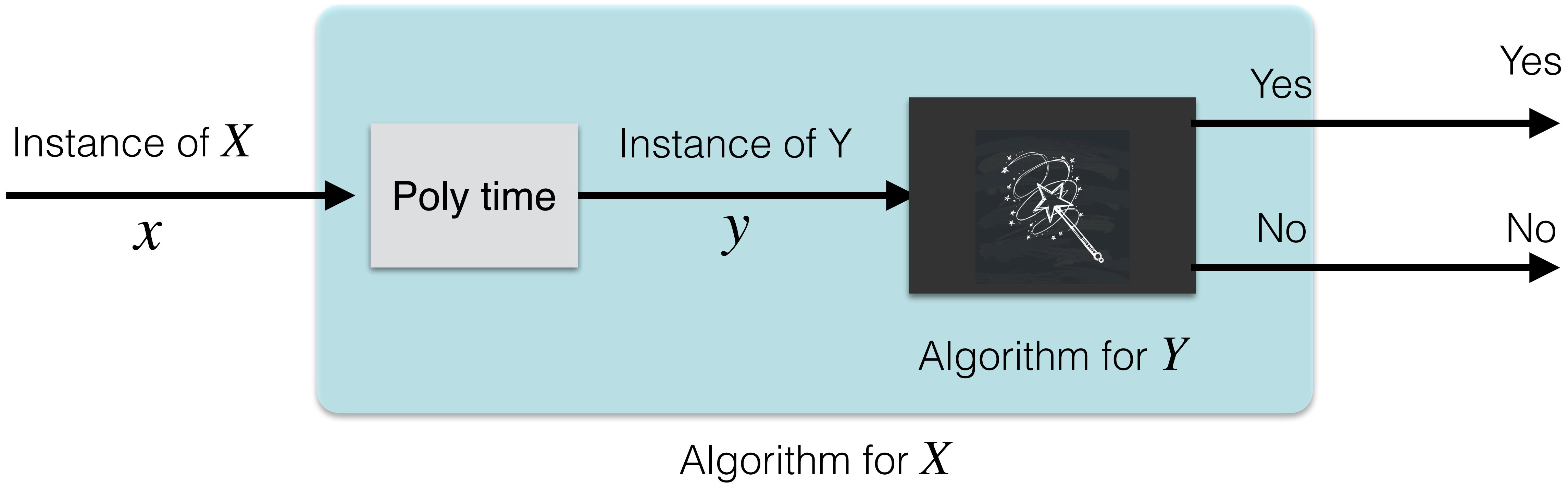


$$A \leq_p B$$

Polynomial-Time Reductions

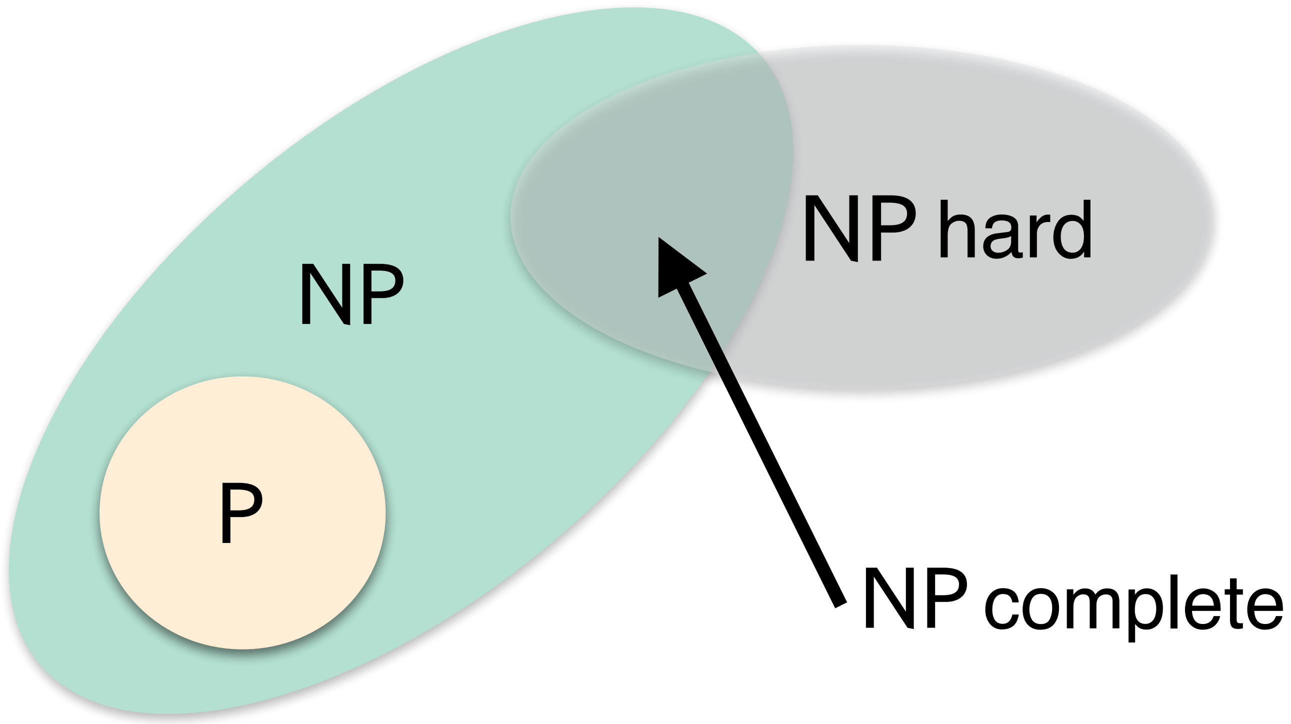
If a language X is poly-time reducible to a language Y ($X \leq_p Y$):

- Solving X is no harder than solving Y: if we have an algorithm for Y, we can use it + poly time reduction to solve X



NP Hard and NP Complete

- **Definition. (NP Hard).** A language B is **NP hard** if every language in NP is polynomial-time reducible to B.
- **Definition. (NP Complete).** A language B is **NP complete** if $B \in$ NP **and** B is NP hard.

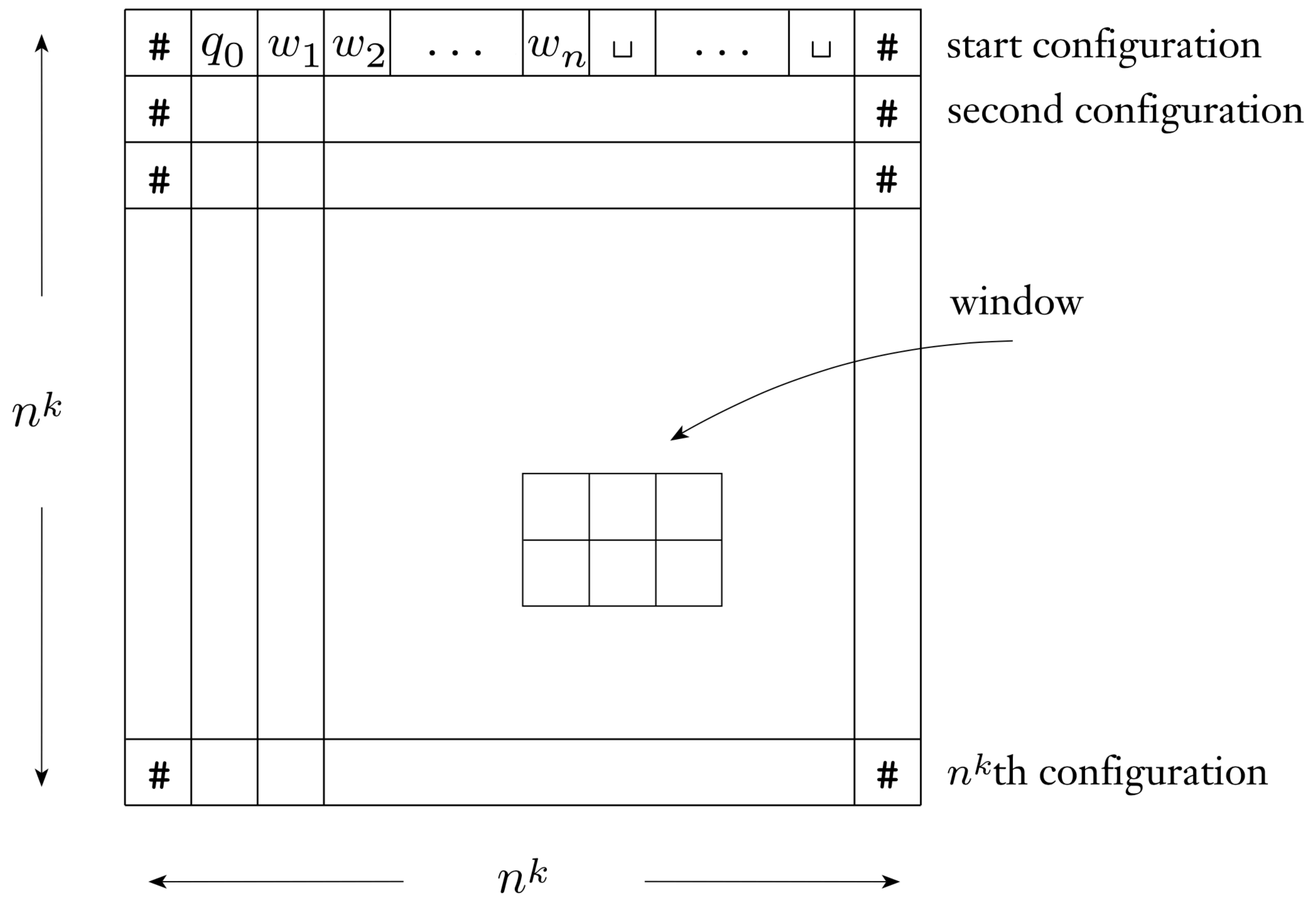


SAT is NP Complete

- Cook Levin Theorem restated:
  - SAT is in **NP**
  - Consider an arbitrary $L \in$ **NP**, then need to show $L \leq_p$ SAT
- **Proof intuition**. Computation history of a NTM can be encoded as a Boolean formula ϕ s.t. an accepting history exists iff ϕ is satisfiable.
- Let's come back to this

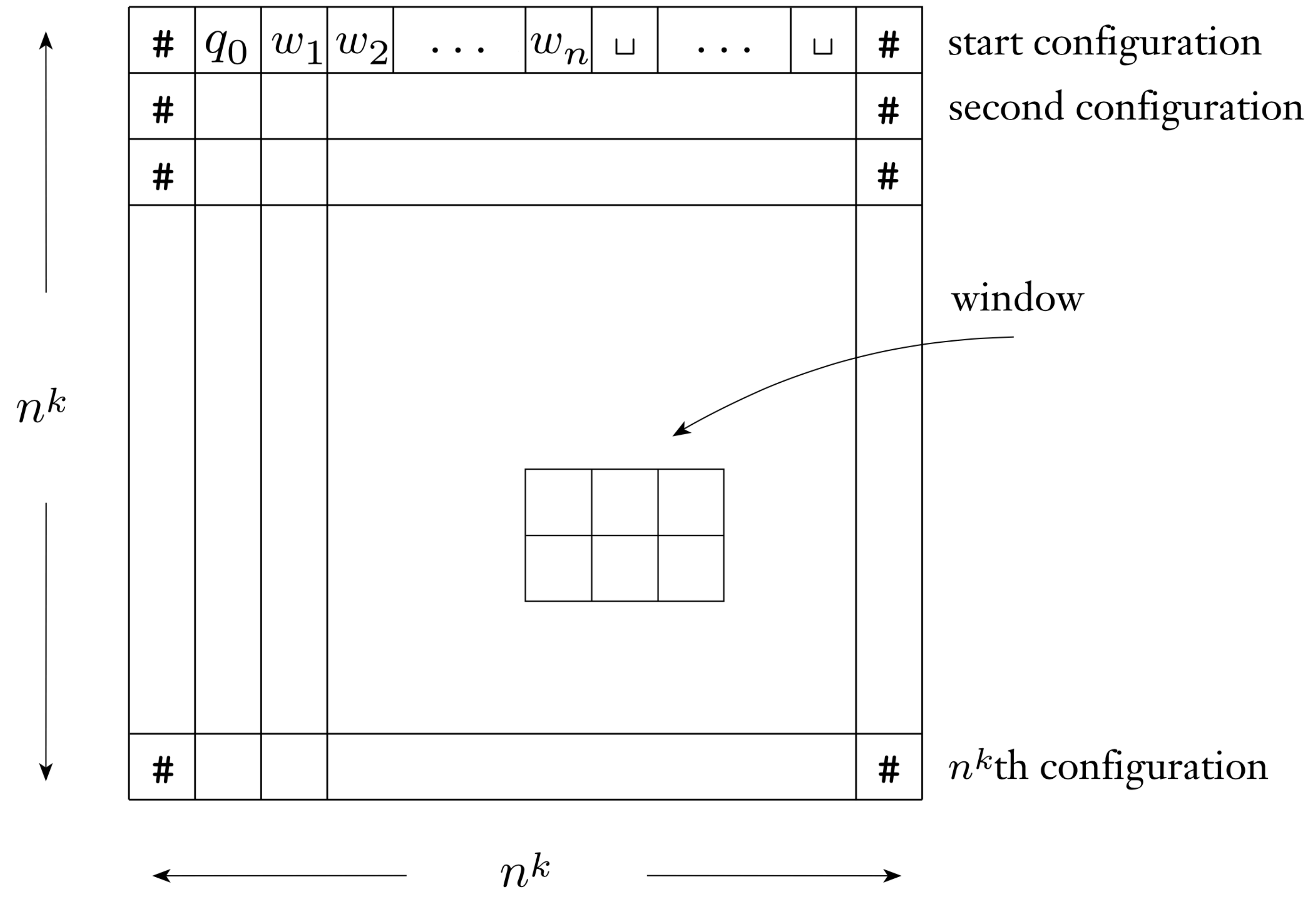
Proof Outline

- There exists a NTM N that runs in time n^k that decides L
- Consider the computation history table of N on an input $w \in L$



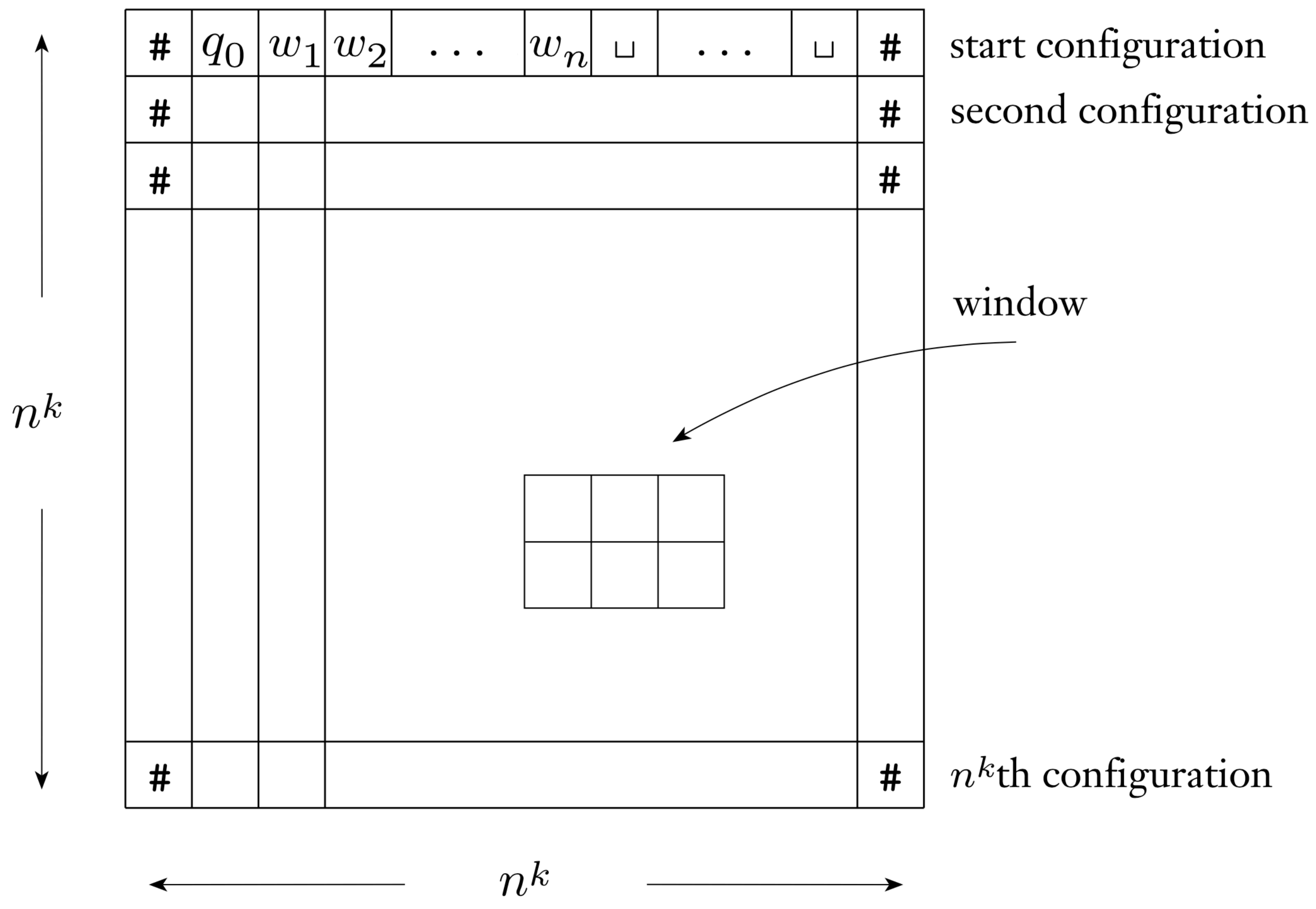
Proof Outline

- Given w, determine if $w \in L$ $\iff$ Given w, does there exist an **accepting** computation history table



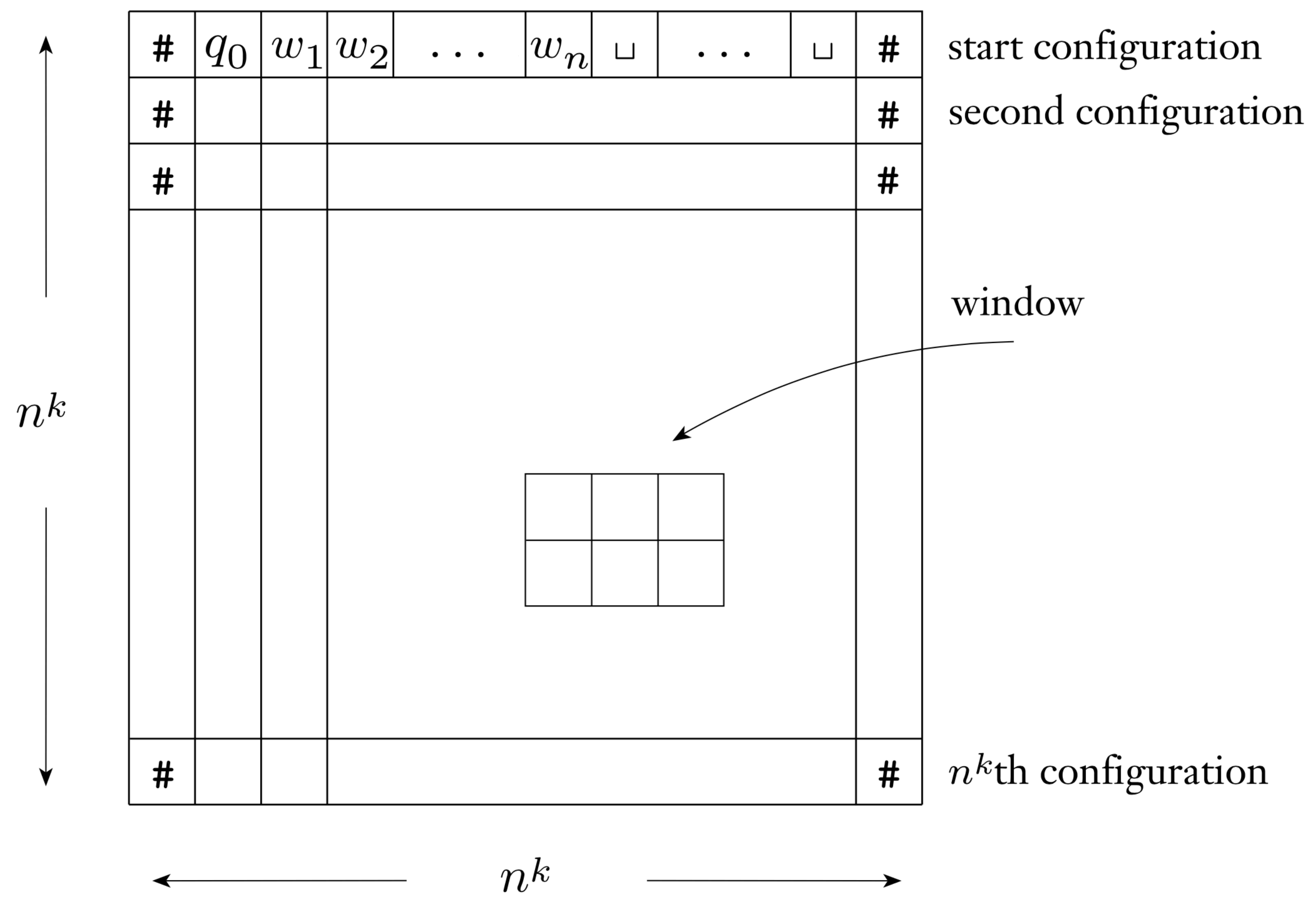
Proof Outline

- **Goal**: Given w and N s.t. $L(N) = L$, construct ϕ such that there exist an **accepting** computation history table of N on w iff $\phi \in$ SAT



Proof Outline

- **Idea**: Encode N's legal computation on w in a SAT instance and ensure it is satisfied iff N has an accepting computation table

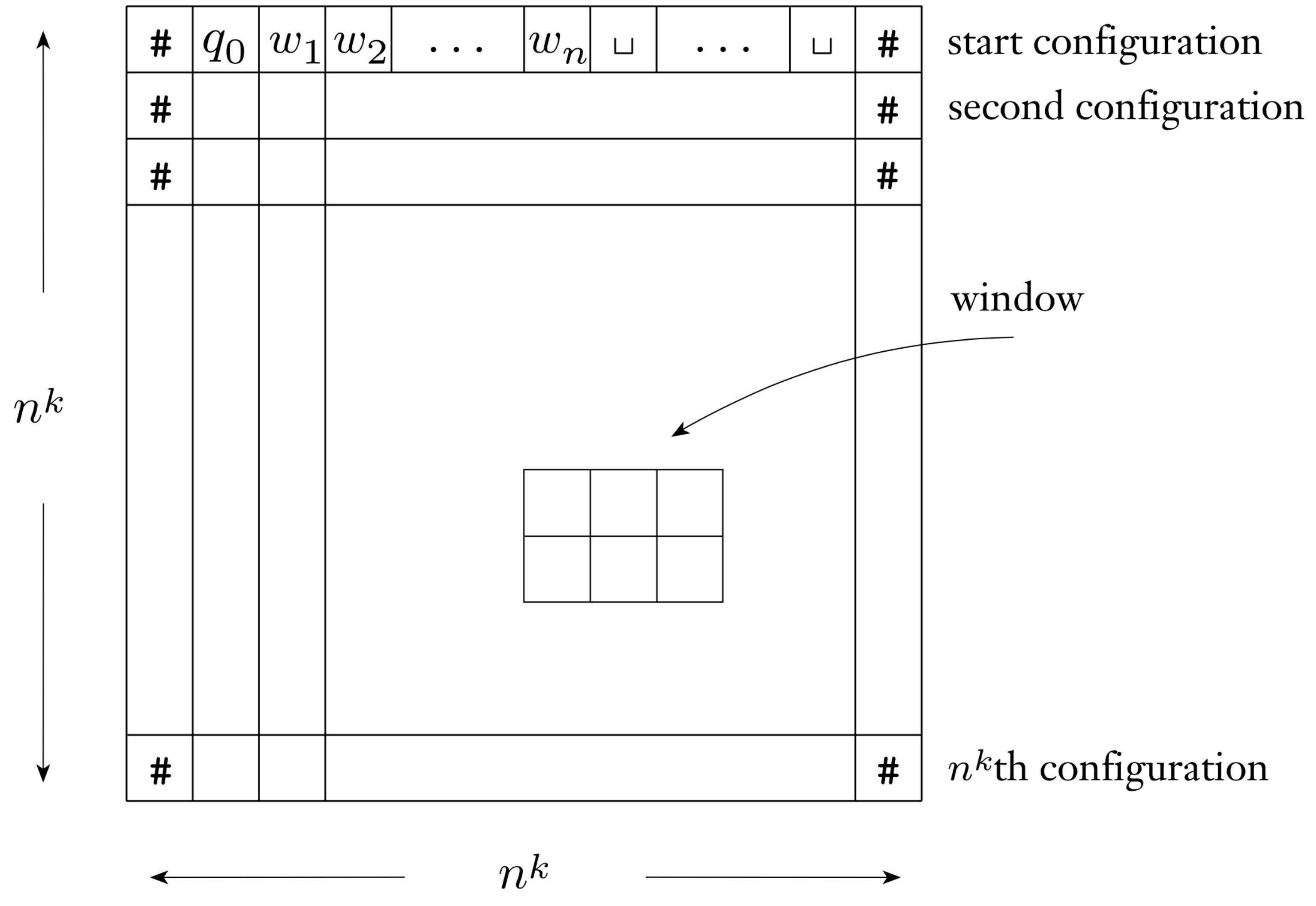


Creating ϕ: Variables

- **Variables of ϕ:**

  $x_{i,j,s}$ where (i, j) is an index of the table and $s \in \{Q \cup \Gamma \cup \#\}$

here $x_{i,j,s} = 1$ means s would be written in cell (i, j) in N's computation



Creating ϕ: Clauses

- Contains several parts enforcing various constraints on the table
- Condition for a valid table:
    - Each cell should only have exactly 1 variable turned "on"

For each cell (i, j), at least one variable is turned on

For each pair of variables s, t, at least one is turned off at (i, j)

$$\phi_{\text{cell}} = \bigwedge_{1 \le i,j \le n^k} \left[\left(\bigvee_{s \in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{\substack{s,t \in C \\ s \neq t}} \left(\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}} \right) \right) \right]$$

$\phi_{\text{cell}} = 1 \iff$ the assignment to variables maps to a feasible computation table

Creating ϕ: Clauses

- Starting configuration to be valid
- ϕ_{start} is True iff the starting row corresponds to a correct starting configuration

$$\begin{aligned}\phi_{\text{start}} = {} & x_{1,1,\#} \wedge x_{1,2,q_0} \wedge \\ & x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge \ldots \wedge x_{1,n+2,w_n} \wedge \\ & x_{1,n+3,\sqcup} \wedge \ldots \wedge x_{1,n^k-1,\sqcup} \wedge x_{1,n^k,\#}\,.\end{aligned}$$

#	q_0	w_1	w_2	$\ldots$	w_n	$\sqcup$	$\ldots$	$\sqcup$	#	start configuration

Creating ϕ: Clauses

- Want the final configuration to be accepting one
- Here ϕ_{accept} is True iff the final configuration has q_{accept}

$$\phi_{\text{accept}} = \bigvee_{1 \leq i,j \leq n^k} x_{i,j,q_{\text{accept}}}.$$

ϕ_{move} Construction

- This is the crux of the reduction!
  - Recall how we encoded TM computation in a PCP instance
  - Only symbols adjacent to head position change

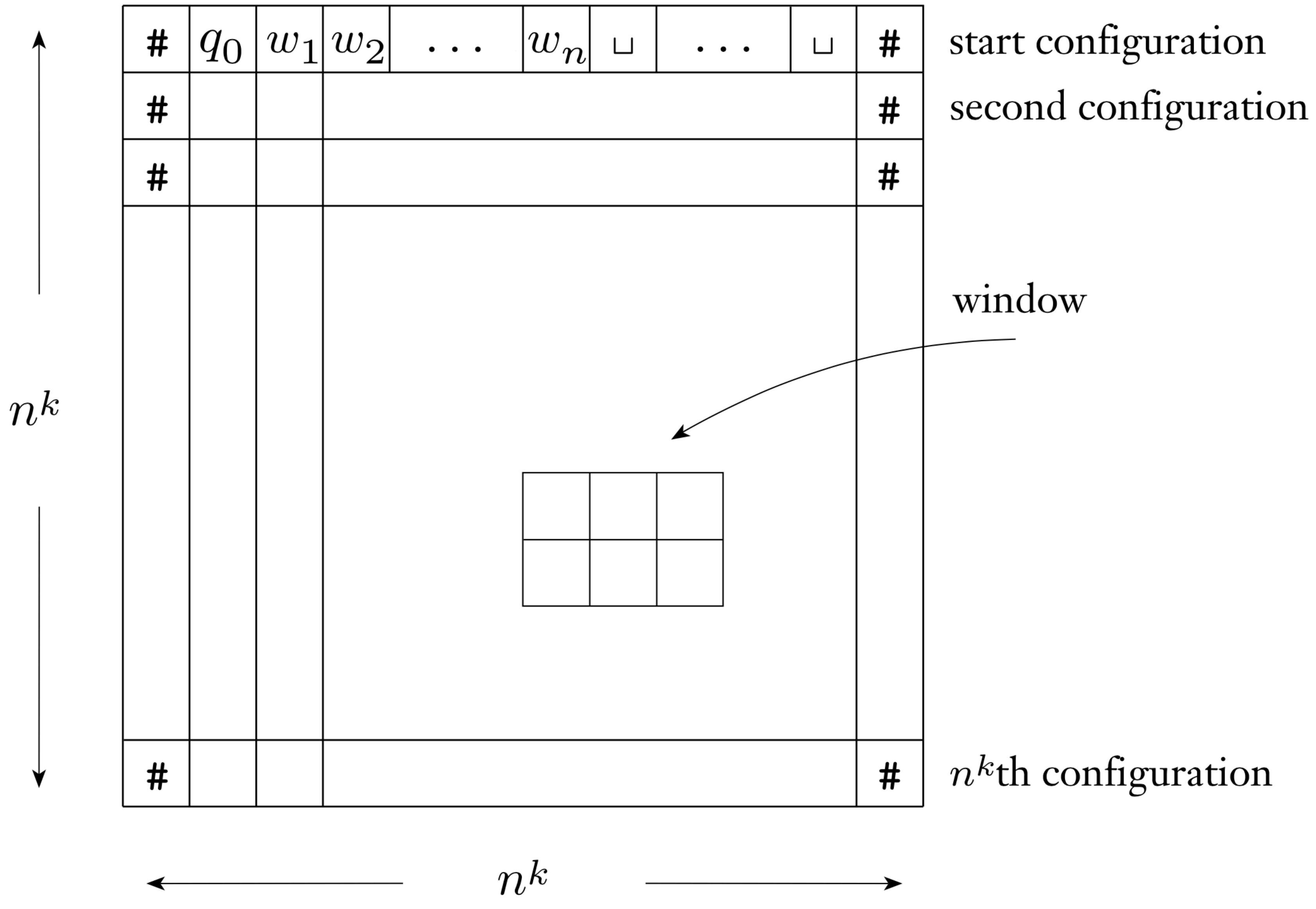
    $\delta(q, a) = (q', b, R): \quad uqav \rightarrow ubq'v$

    $\delta(q, a) = (q', b, L): \quad ucqav \rightarrow uq'cbv$

ϕ_{move} Construction

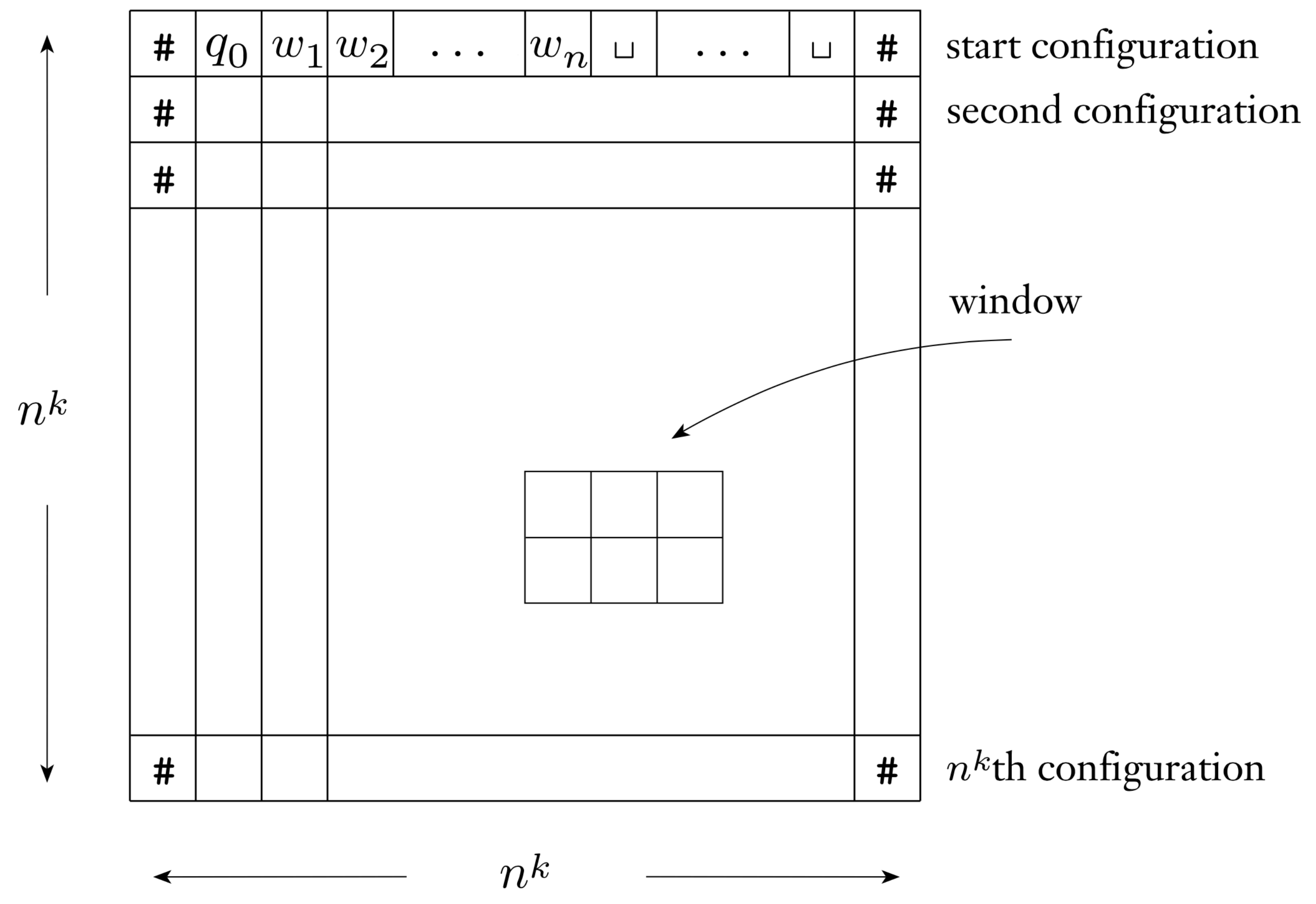
- $\delta(q,a) = (q', b, R)$: $uqav \to ubq'v$

  $\delta(q,a) = (r, b, L)$: $ucqav \to uq'cbv$



ϕ_{move} Construction

- Can check each 2×3 window of the table and make sure it is consistent with legal transitions



$$\phi_{\text{move}} = \bigwedge_{1 \le i < n^k,\ 1 < j < n^k} \big(\text{the } (i,j)\text{-window is legal}\big).$$

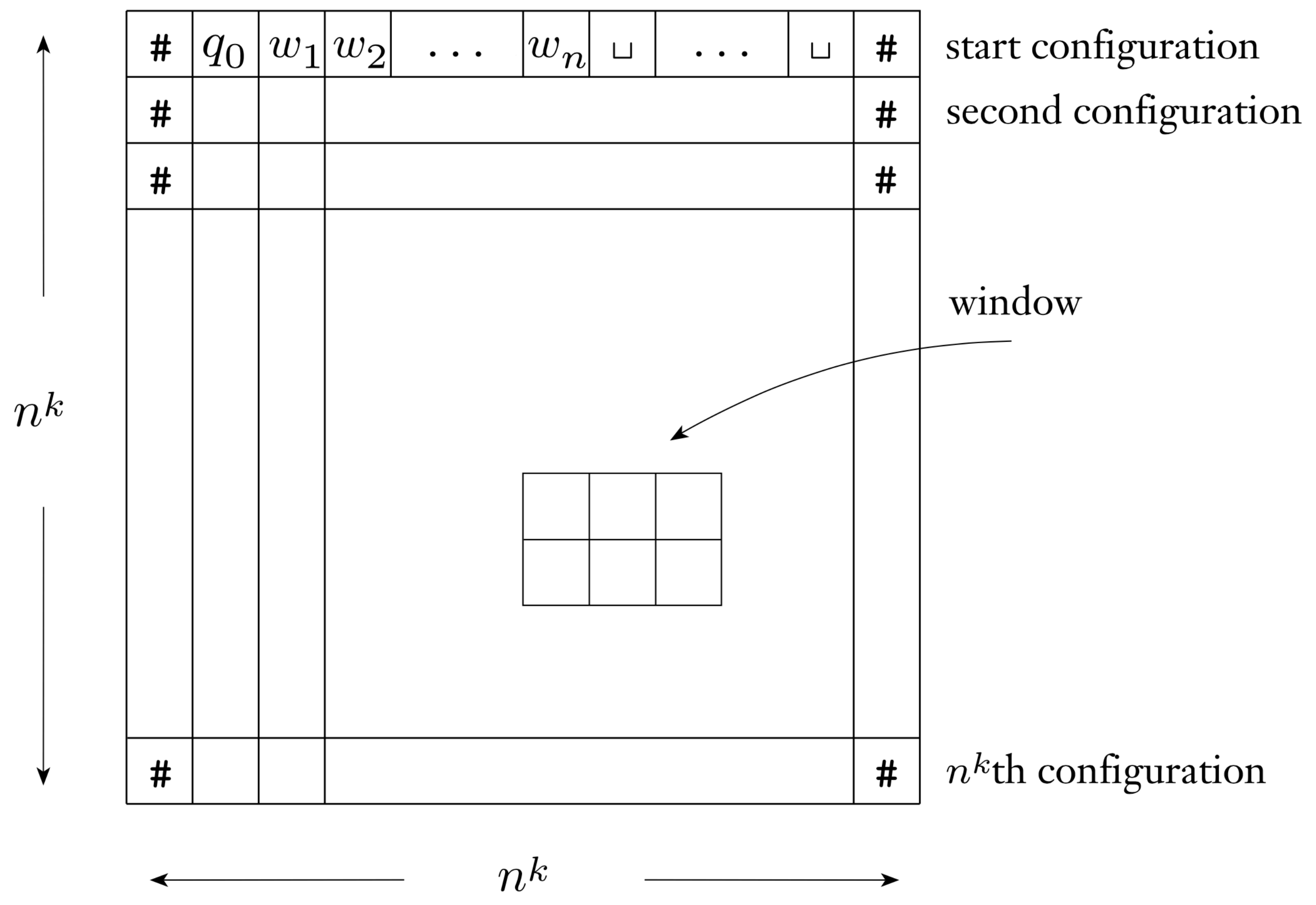
ϕ Overall

- $\phi = \phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}}$
- **Summary.** Given w and NTM N for L, constructed ϕ such that
    - An assignment to ϕ is satisfiable iff the corresponding (valid) computation history table of N on w is an accepting one

Polynomial-time Reduction

- # variables $O(n^{2k})$ each part of ϕ has size $O(n^{2k})$
- Reduction complexity it is linear-time in size of ϕ

$$\phi = \phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}}$$



To Conclude

- We showed that SAT is in NP, and,
- Any language $L \in \mathsf{NP}$ can be reduced in polynomial time to SAT
- Thus, SAT is NP complete

3SAT is NP Complete

- A specialized instance of SAT which is in
    - conjunctive-normal-form (CNF): consists of AND of clauses
    - each clause is an OR of 3 literals
- Note that 3SAT is in NP (can verify assignment is correct in poly time)

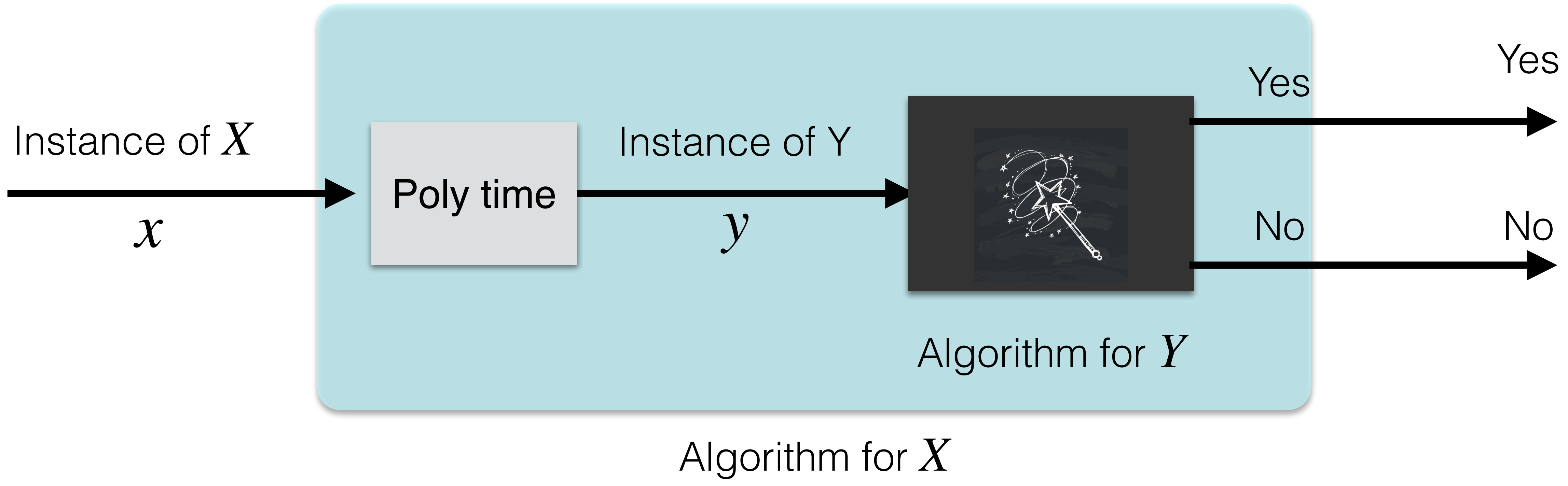
$$\phi = (\overline{x_1} \vee x_2, \vee x_3) \wedge (x_1, \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$$

3SAT is NP Complete

- To show every language in NP reduces to 3SAT in poly time, it is easy to modify the previous construction
  - Replace OR of ANDs with AND of ORs (distributive law)
  - Any clause with more than 3 literals, can be broken up using additional variables, e.g. $(a_1 \vee a_2 \vee a_3 \vee a_4)$ can be replaced with $(a_1 \vee a_2 \vee z) \wedge (\bar{z} \vee a_3 \vee a_4)$
- **Claim**. $(a_1 \vee a_2 \vee a_3 \vee a_4)$ is satisfiable iff $(a_1 \vee a_2 \vee z) \wedge (\bar{z} \vee a_3 \vee a_4)$ is satisfiable.
- Why is this true?

Reductions to Prove NP hardness

- To prove a language Y is NP hard, we is hard to prove that every language in **NP** reduces to Y
- Instead, show that a known-NP-hard problem X reduces to Y
- **Takeaway**. If Y was solvable in poly time, then X would be as well



CLIQUE is NP Complete

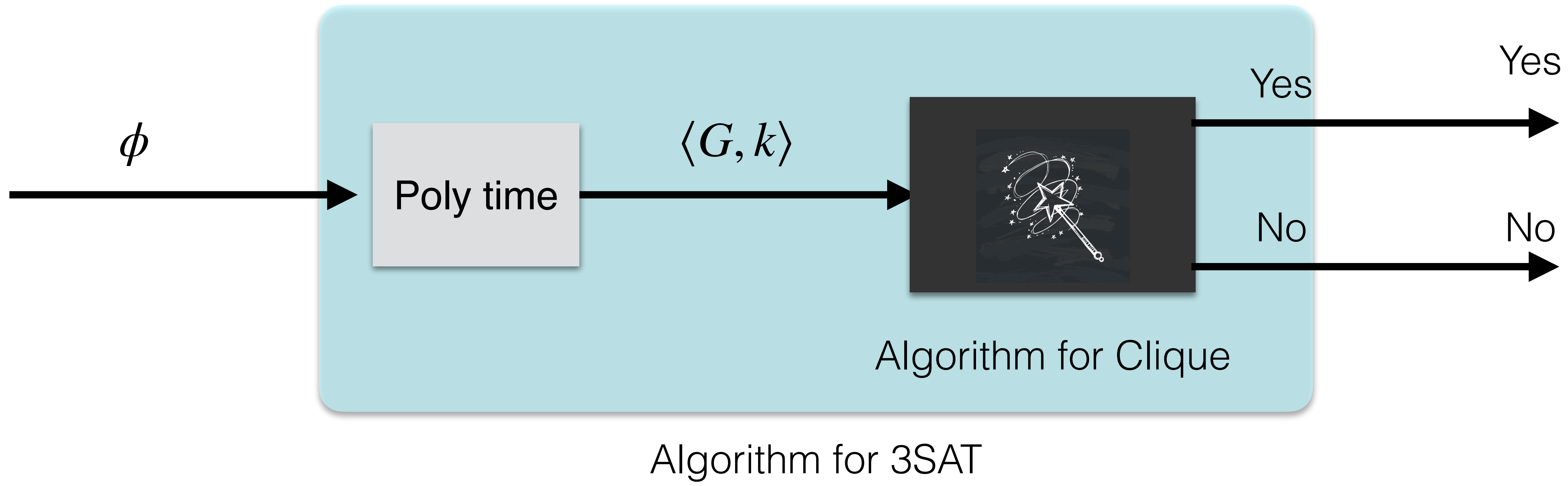
- A clique C in a graph $G = (V, E)$ is a subset of vertices such that every pair $u, v \in C$ has an edge between it. A k-clique is a clique containing k vertices.
- CLIQUE = $\{\langle G, k\rangle \mid G$ is an undirected graph with a k-clique$\}$
- First, show Clique is in NP

Step 1: CLIQUE is in NP

- Poly-time verifier V for Clique:
  - $V =$ "On input $\langle\langle G, k\rangle, C\rangle$:
    - If C is not a valid subset of V or does not contain k node, reject
    - If every pair of nodes in C is not adjacent to each other, reject
    - Otherwise, C is a k-clique, accept.
- Alternatively, a poly-time NTM for CLIQUE:
    - $N =$ "On input $\langle G, k\rangle$,
    - Non-deterministically select a subset C of V
    - Verify (as above) if C is a k-clique; if not reject.
    - Otherwise, accept.

Step 2: CLIQUE is NP Hard

$$\boxed{\text{3SAT} \leq_p \text{CLIQUE}}$$



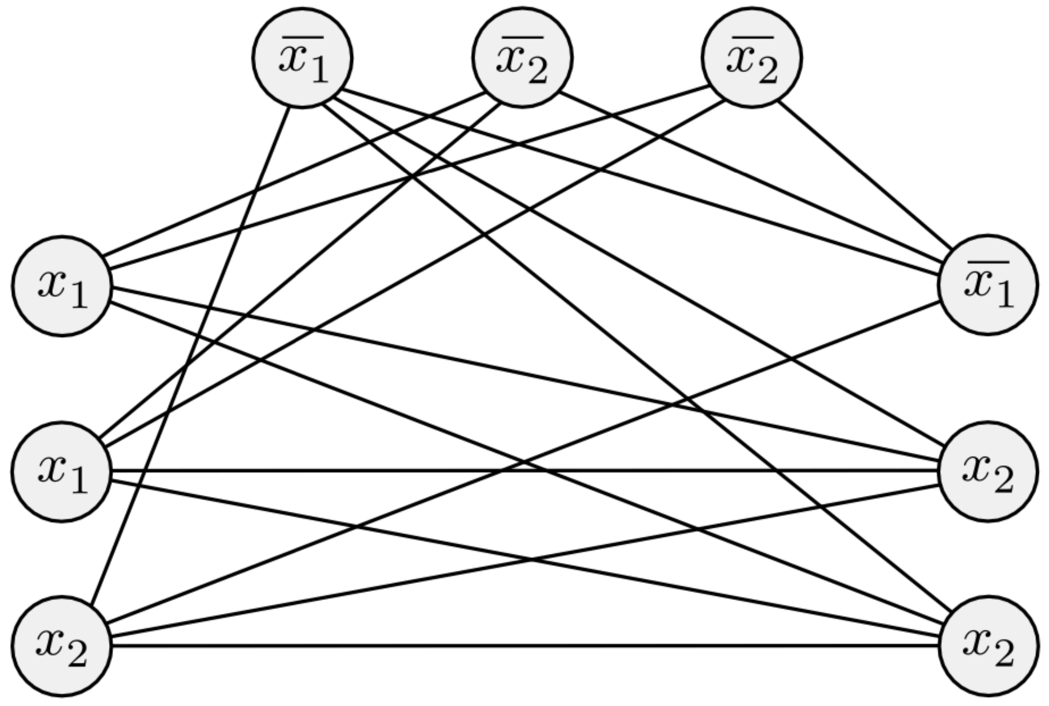
Map the Problems

3SAT		**CLIQUE**
	What is an input?	
Variables and clauses		Vertices and edges
	What is a possible solution?	
A selection of vertices set to TRUE		A selection of vertices in the Clique
	What is the requirement?	
Each clause has at least 1 True literal		Selected vertices form a k-clique

3SAT Reduction to Clique

- **Reduction.** Given a 3SAT instance ϕ with k clauses, create graph G with $3k$ nodes:
  - A node for each literal, organized in k groups of 3
  - Add an edge between every pair of node across groups except x_i and $\overline{x_i}$ (no edge between nodes in the same group)

$$\phi = (x_1 \lor x_1 \lor x_2) \land (\overline{x_1} \lor \overline{x_2} \lor \overline{x_2}) \land (\overline{x_1} \lor x_2 \lor x_2)$$



Correctness of Reduction

- **Prove:** ϕ is satisfiable iff G has a clique of size k.
- ($\Rightarrow$) If ϕ has a satisfying assignment, then G has a k-clique
  - Each clause has at least one true literal
  - Pick the vertex for that literal to be in the clique
  - If more than one literal is true, choose any one of them
- **Question.** Why is this a k-clique?
  - k vertices: one from each clause triple
  - Each pair has an edge between them since they can't be from the same triple, or between x_i and $\overline{x_i}$ (both can't be true)

Correctness of Reduction

- **Prove:** ϕ is satisfiable iff G has a clique of size k.
- $(\Leftarrow)$ G has a k-clique, then ϕ has a satisfying assignment
    - Must have one vertex per clause triple, why?
    - Set corresponding literal in each clause to True
    - Why is this a valid assignment?

Correctness of Reduction

- **Prove:** ϕ is satisfiable iff G has a clique of size k.
- Suppose ϕ has a satisfying assignment, what is the k-clique in G?
    - Each clause has at least one true literal
    - Pick the vertex for that literal to be in the clique
    - If more than one literal is true, choose any one of them
    - Why is this a k-clique?
- Suppose G has a k clique, what is the satisfying assignment for ϕ?

Class Exercises:

- An **independent set** is a subset of vertices $S \subseteq V$ such that no two of them are adjacent: for any $x, y \in S,\ (x, y) \notin E$
    - **IND-SET Problem.** Given a graph $G = (V, E)$ and an integer k, does G have an independent set of size at least k?
    - Show that CLIQUE $\leq_p$ IND-SET
- A ***Hamiltonian cycle*** in a graph G is a cycle that visits every vertex of G exactly once. Deciding whether an arbitrary graph contains a Hamiltonian cycle is NP-hard. A ***tonian cycle*** in a graph G is a cycle that visits at least half of the vertices of G. Prove that deciding whether a graph contains a tonian cycle is NP-hard.

Class Exercises:

- A **vertex cover** is a subset of vertices $T \subseteq V$ such that for every edge $e = (u, v) \in E$, either $u \in T$ or $v \in T$.
    - **VERTEX-COVER Problem.** Given a graph $G = (V, E)$ and an integer k, does G have a vertex cover of size at most k?