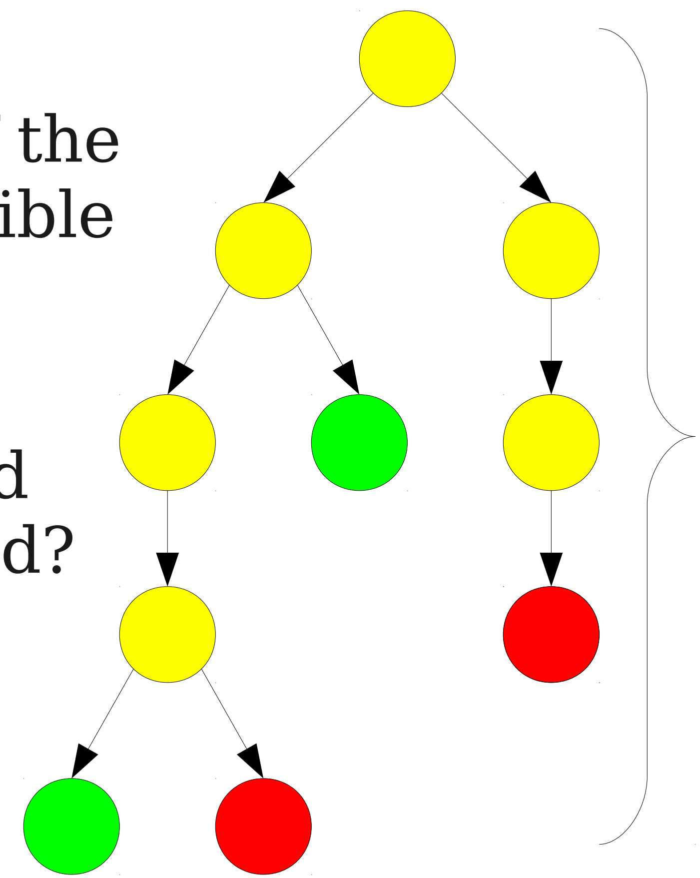
NP-Completeness

Recap from Last Time

Analyzing NTMs

- When discussing deterministic TMs, the notion of time complexity is (reasonably) straightforward.
- **Recall:** One way of thinking about nondeterminism is as a tree.
- The time complexity is the height of the tree (the length of the **longest** possible choice we could make).
- Intuition: If you ran all possible branches in parallel, how long would it take before all branches completed?



The Complexity Class **NP**

- The complexity class **NP** (**nondeterministic polynomial time**) contains all problems that can be solved in polynomial time by an NTM.
- Formally:

NP = { *L* | There is a nondeterministic TM that decides *L* in polynomial time. }

Another View of **NP**

- ***Theorem:*** $L \in$ **NP** iff there is a *deterministic* TM V with the following properties:
  - $w \in L$ iff there is some $c \in \Sigma^*$ such that V accepts $\langle w, c \rangle$.
  - V runs in time polynomial in $|w|$.
- Some terminology:
  - A TM V with the above property is called a **polynomial-time verifier for *L***.
  - The string c is called a **certificate** for w.
  - You can think of V as checking the certificate that proves $w \in L$.

NP and Reductions

Polynomial-Time Reductions

- Suppose that we know that $B \in \mathbf{NP}$.
- Suppose that $A \leq_{\mathrm{P}} B$ and that the reduction f can be computed in time $\mathrm{O}(n^k)$.

A

Solvable?

B

Solvable by NTM in $\mathrm{O}(n^r)$

Polynomial-Time Reductions

- Suppose that we know that $B \in \mathbf{NP}$.
- Suppose that $A \leq_P B$ and that the reduction f can be computed in time $O(n^k)$.

Input size: $\boldsymbol{n}$

A

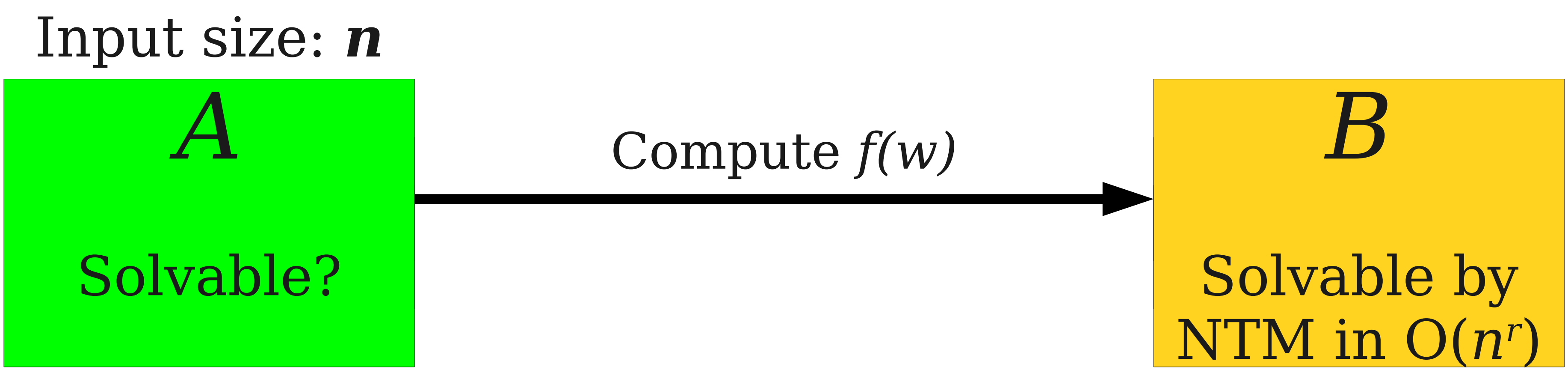
Solvable?

B

Solvable by NTM in $O(n^r)$

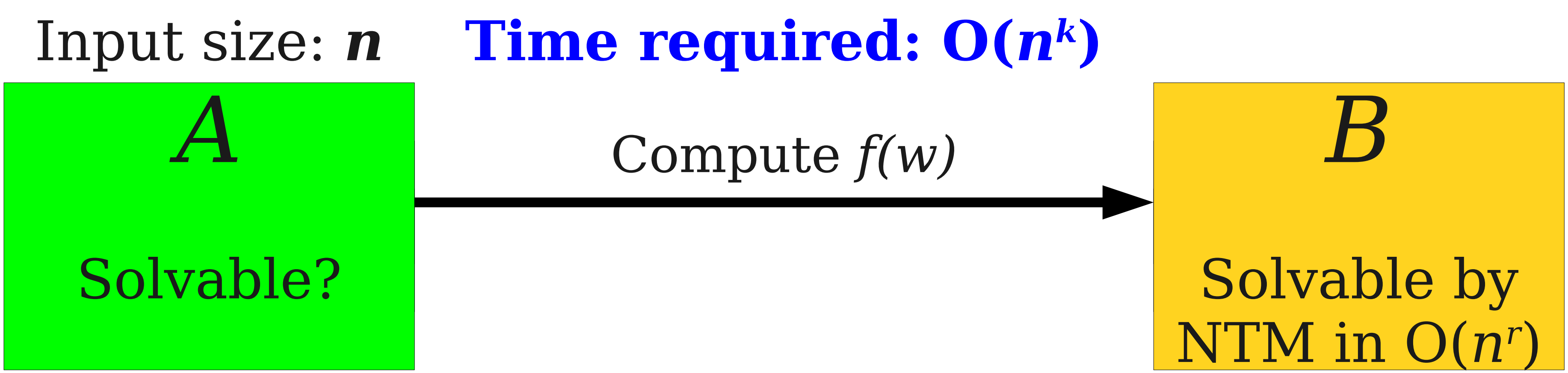
Polynomial-Time Reductions

- Suppose that we know that $B \in \mathbf{NP}$.
- Suppose that $A \leq_{\mathrm{P}} B$ and that the reduction f can be computed in time $O(n^k)$.



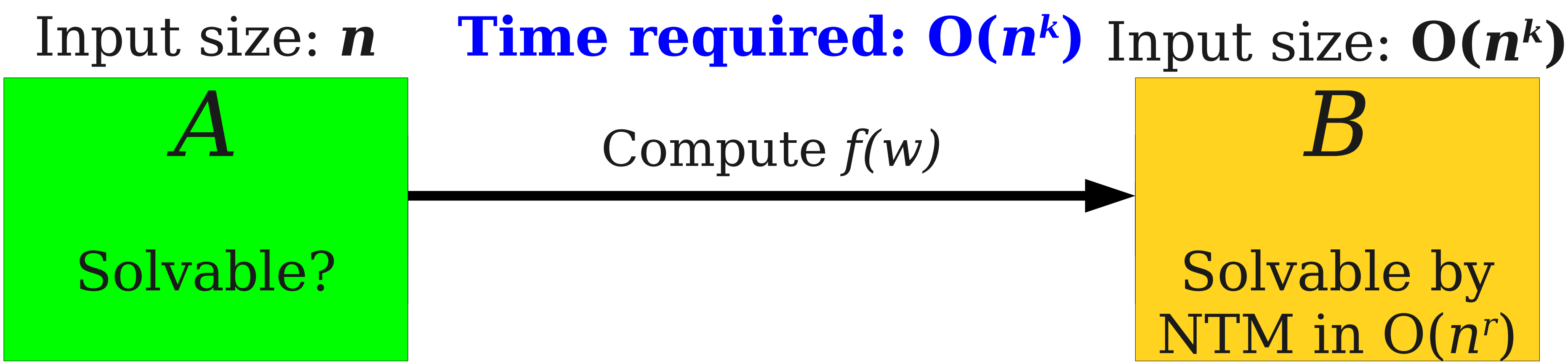
Polynomial-Time Reductions

- Suppose that we know that $B \in \mathbf{NP}$.
- Suppose that $A \leq_{\mathrm{P}} B$ and that the reduction f can be computed in time $O(n^k)$.



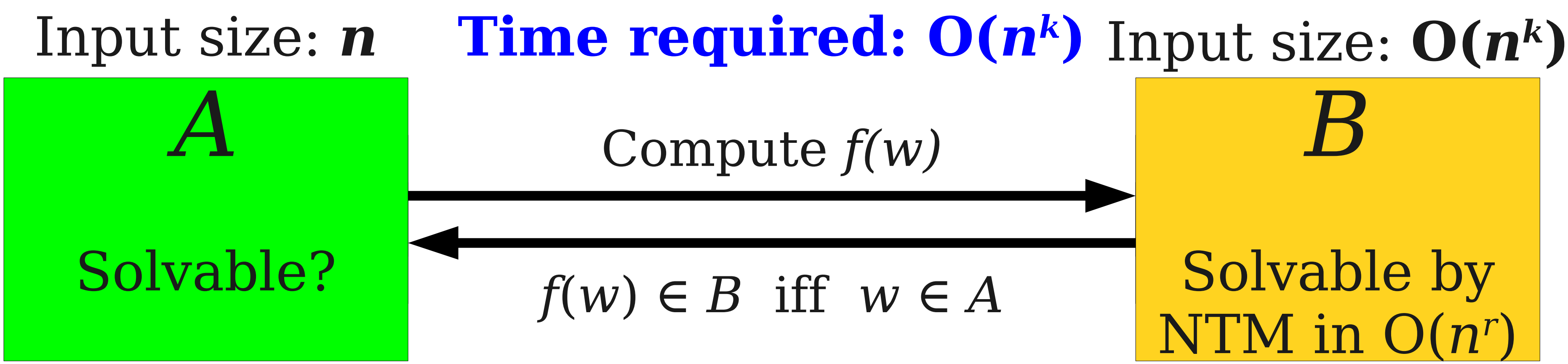
Polynomial-Time Reductions

- Suppose that we know that $B \in \mathbf{NP}$.
- Suppose that $A \leq_P B$ and that the reduction f can be computed in time $O(n^k)$.



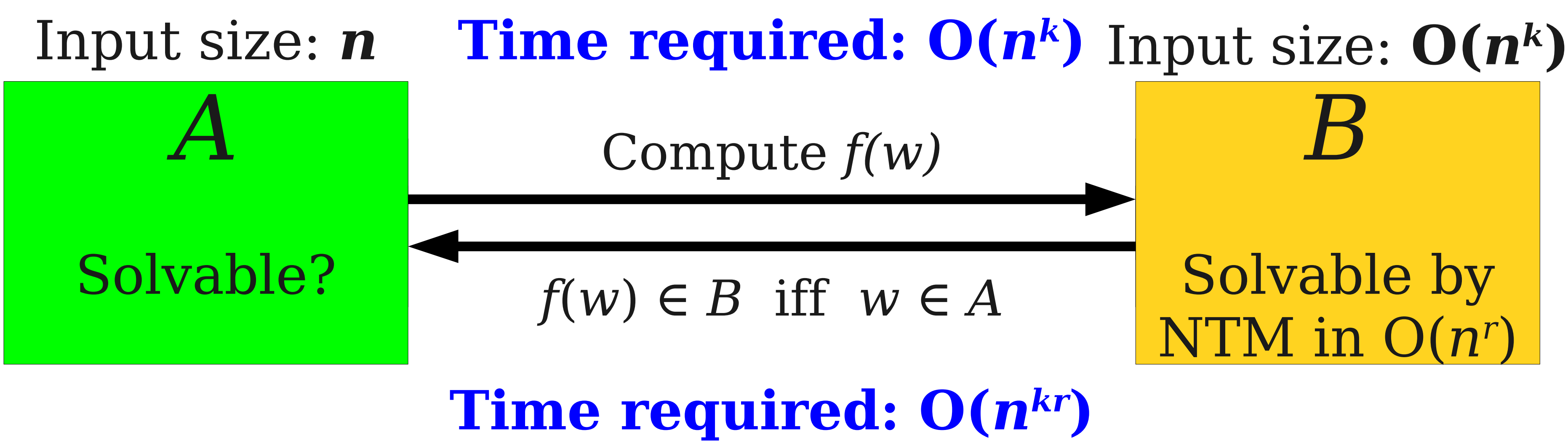
Polynomial-Time Reductions

- Suppose that we know that $B \in \mathbf{NP}$.
- Suppose that $A \leq_P B$ and that the reduction f can be computed in time $O(n^k)$.



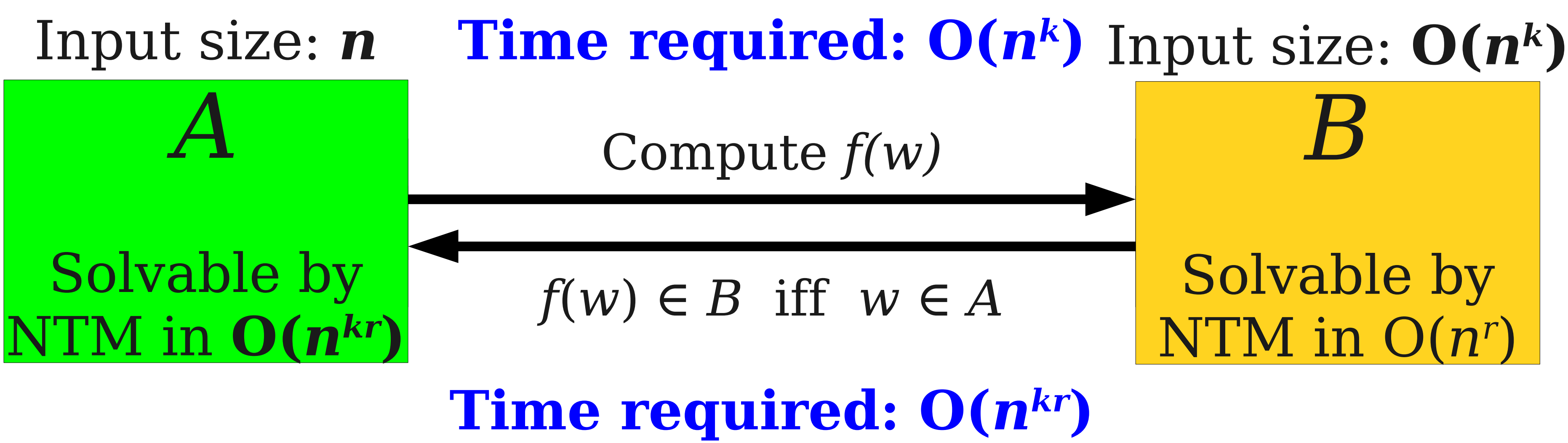
Polynomial-Time Reductions

- Suppose that we know that $B \in \mathbf{NP}$.
- Suppose that $A \leq_P B$ and that the reduction f can be computed in time $O(n^k)$.



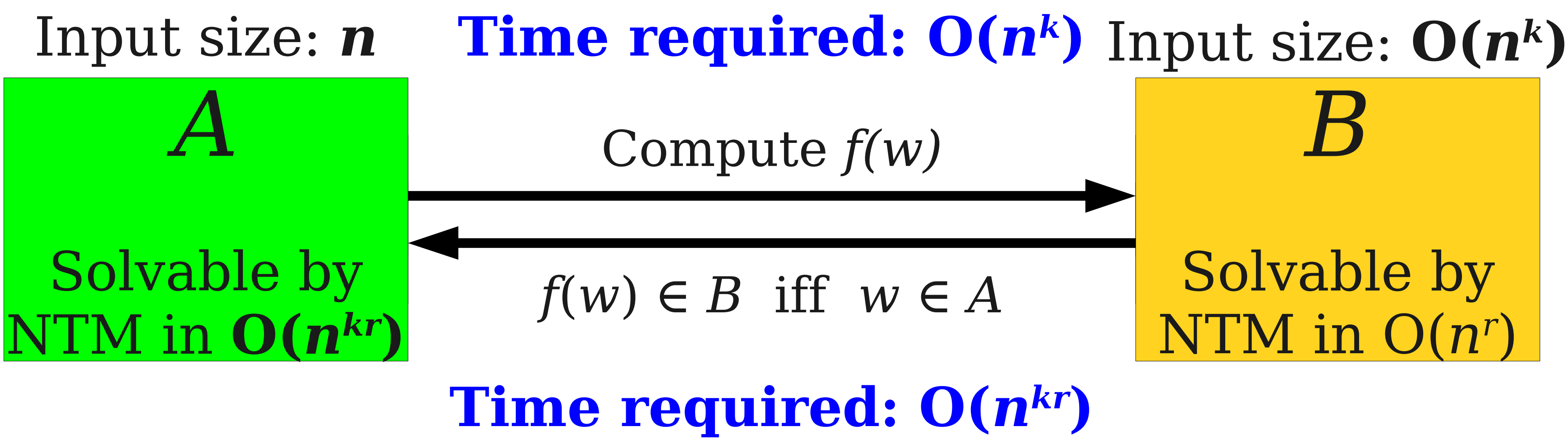
Polynomial-Time Reductions

- Suppose that we know that $B \in \mathbf{NP}$.
- Suppose that $A \leq_{\mathrm{P}} B$ and that the reduction f can be computed in time $\mathrm{O}(n^k)$.



Polynomial-Time Reductions

- Suppose that we know that $B \in \mathbf{NP}$.
- Suppose that $A \leq_P B$ and that the reduction f can be computed in time $O(n^k)$.
- Then $A \in \mathbf{NP}$ as well.



A Sample Reduction

$$U = \{1, 2, 3, 4, 5, 6\}$$

$$S = \left\{ \begin{matrix} \{1, 2, 5\}, \{2, 5\}, \{1, 3, 6\}, \\ \{2, 3, 4\}, \{4\}, \{1, 5, 6\} \end{matrix} \right\}$$

Let U be a set of elements (the **universe**) and $S \subseteq \wp(U)$. An **exact covering** of U is a collection of sets $I \subseteq S$ such that every element of U belongs to exactly one set in I.

$$U = \{1, 2, 3, 4, 5, 6\}$$

$$S = \left\{ \begin{array}{ccc} \{1, 2, 5\}, & \{2, 5\}, & \{1, 3, 6\}, \\ \{2, 3, 4\}, & \{4\}, & \{1, 5, 6\} \end{array} \right\}$$

Let U be a set of elements (the **universe**) and $S \subseteq \wp(U)$. An **exact covering** of U is a collection of sets $I \subseteq S$ such that every element of U belongs to exactly one set in I.

$$U = \{1, 2, 3, 4, 5, 6\}$$

$$S = \left\{ \begin{array}{ccc} \{1, 2, 5\}, & \{2, 5\}, & \{1, 3, 6\}, \\ \{2, 3, 4\}, & \{4\}, & \{1, 5, 6\} \end{array} \right\}$$

Let U be a set of elements (the **universe**) and $S \subseteq \wp(U)$. An **exact covering** of U is a collection of sets $I \subseteq S$ such that every element of U belongs to exactly one set in I.

$$U = \{1, 2, 3, 4, 5, 6\}$$

$$S = \left\{ \begin{array}{l} \{1, 2, 5\}, \{2, 5\}, \{1, 3, 6\}, \\ \{2, 3, 4\}, \{4\}, \{1, 5, 6\} \end{array} \right\}$$

Let U be a set of elements (the **universe**) and $S \subseteq \wp(U)$. An **exact covering** of U is a collection of sets $I \subseteq S$ such that every element of U belongs to exactly one set in I.

$$U = \{1, 2, 3, 4, 5, 6\}$$

$$S = \left\{ \begin{array}{c} \{1, 2, 5\}, \{2, 5\}, \{1, 3, 6\}, \\ \{2, 3, 4\} \; \{4\}, \{1, 5, 6\} \end{array} \right\}$$

Let U be a set of elements (the **universe**) and $S \subseteq \wp(U)$. An **exact covering** of U is a collection of sets $I \subseteq S$ such that every element of U belongs to exactly one set in I.

Exact Covering

- Given a universe U and a set $S \subseteq \wp(U)$, the exact covering problem is

Does S contain an exact covering of U?

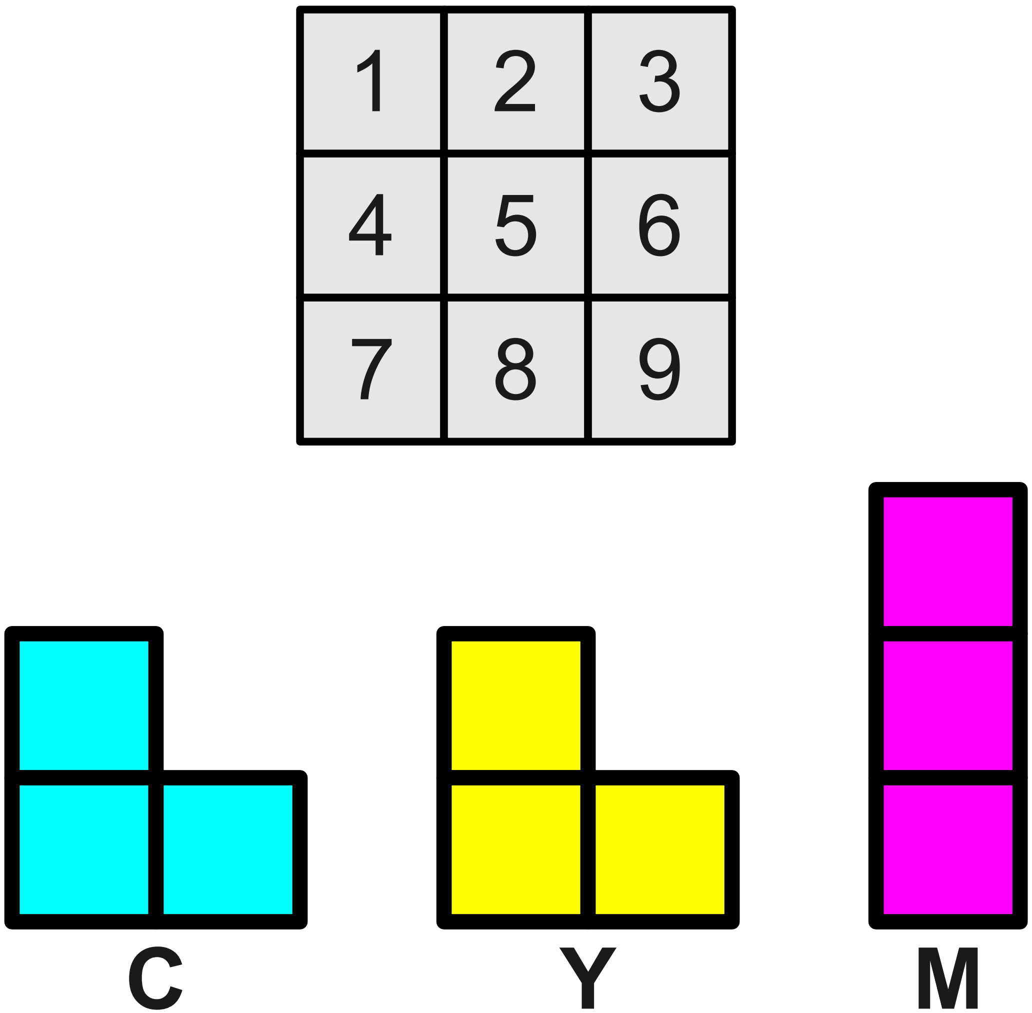
- As a formal language:

$$\textbf{EXACT-COVER} = \{ \langle U, S \rangle \mid S \subseteq \wp(U) \text{ and } S \text{ contains an exact covering of } U \}$$

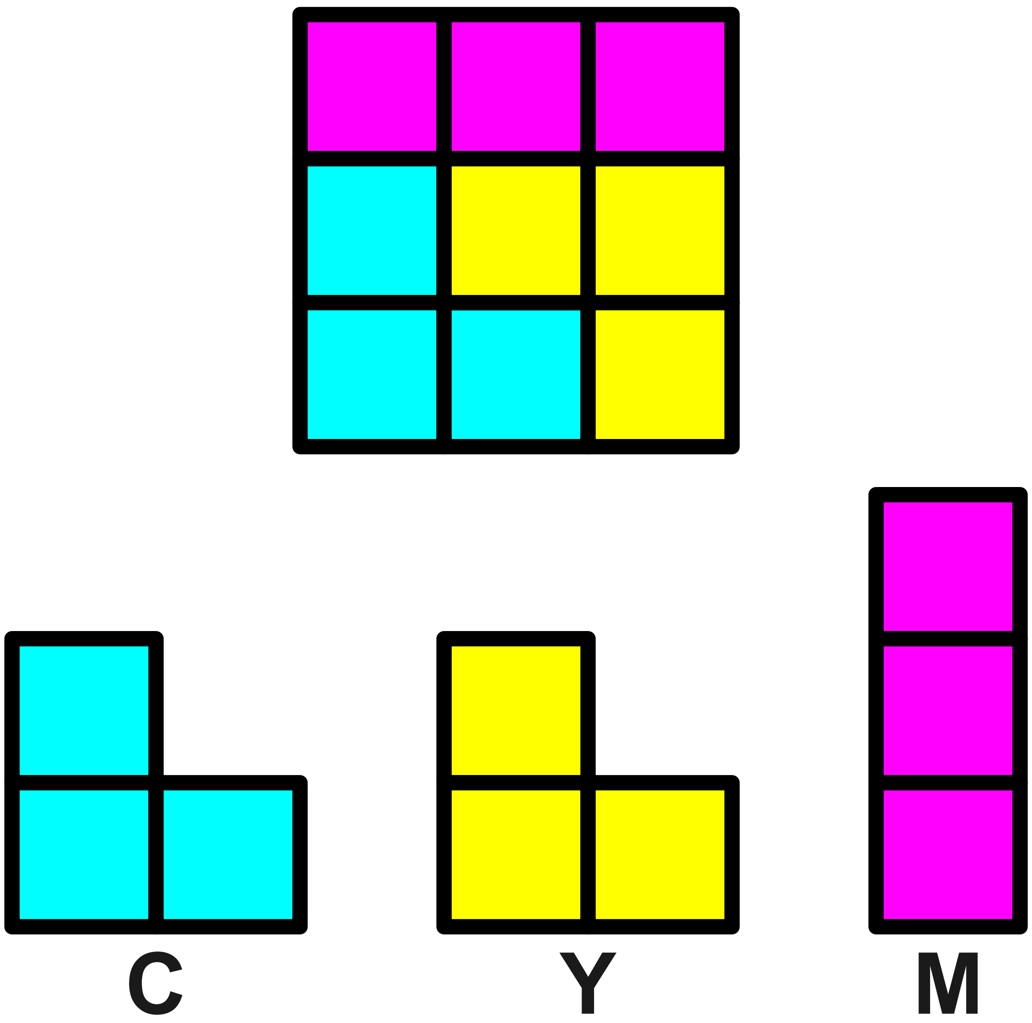
EXACT-COVER $\in$ **NP**

- Here is a polynomial-time verifier for *EXACT-COVER*:
- V = "On input $\langle U, S, I\rangle$, where U, S, and I are sets:
  - Verify that every set in S is a subset of U.
  - Verify that every set in I is an element of S.
  - Verify that every element of U belongs to an element of I.
  - Verify that every element of U belongs to at most one element of I."

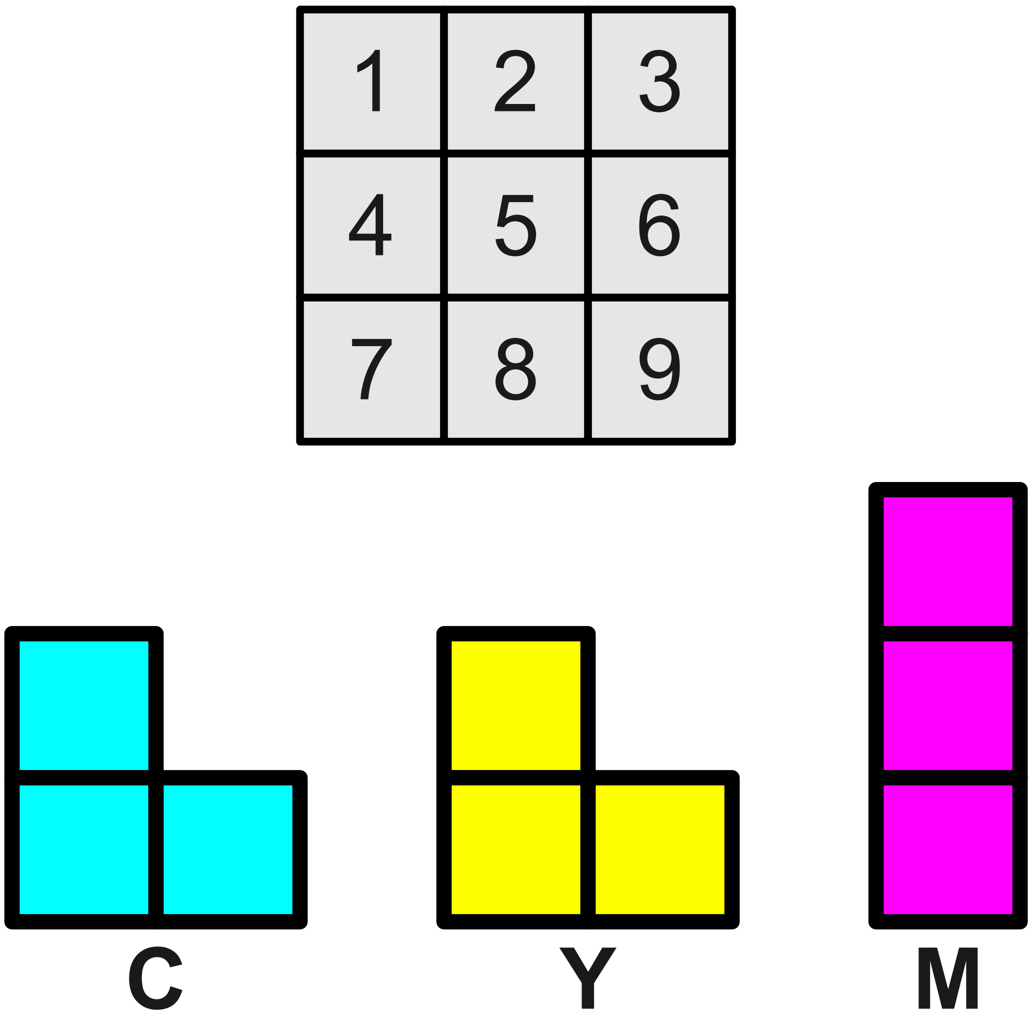
Applications of Exact Covering



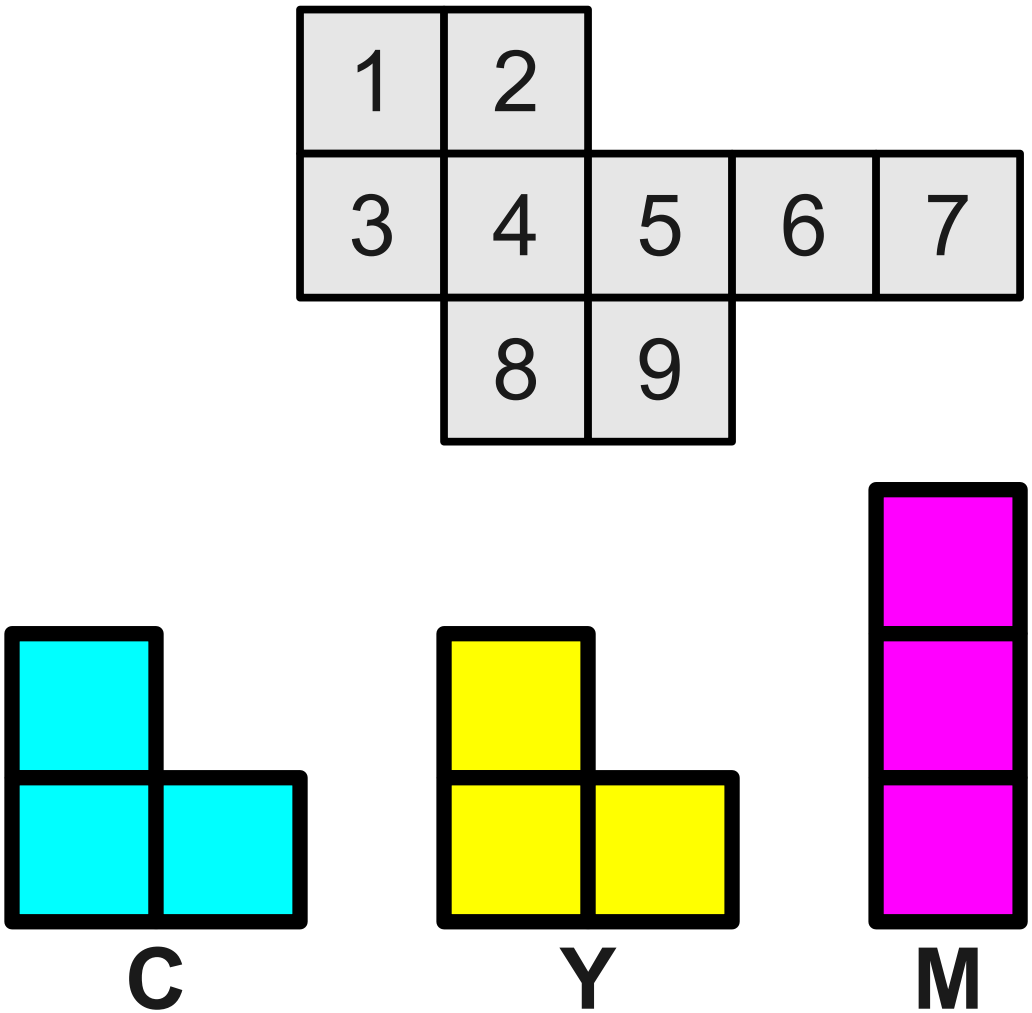
Applications of Exact Covering



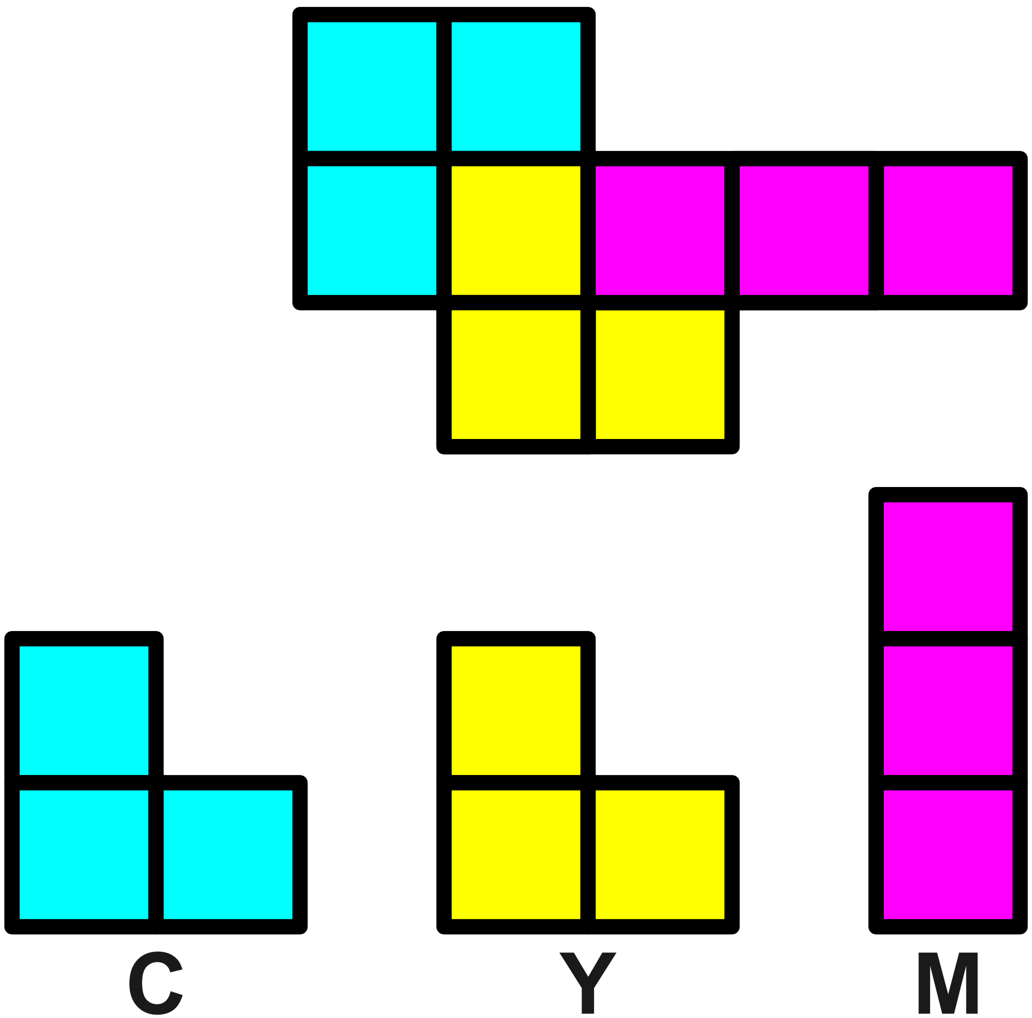
Applications of Exact Covering



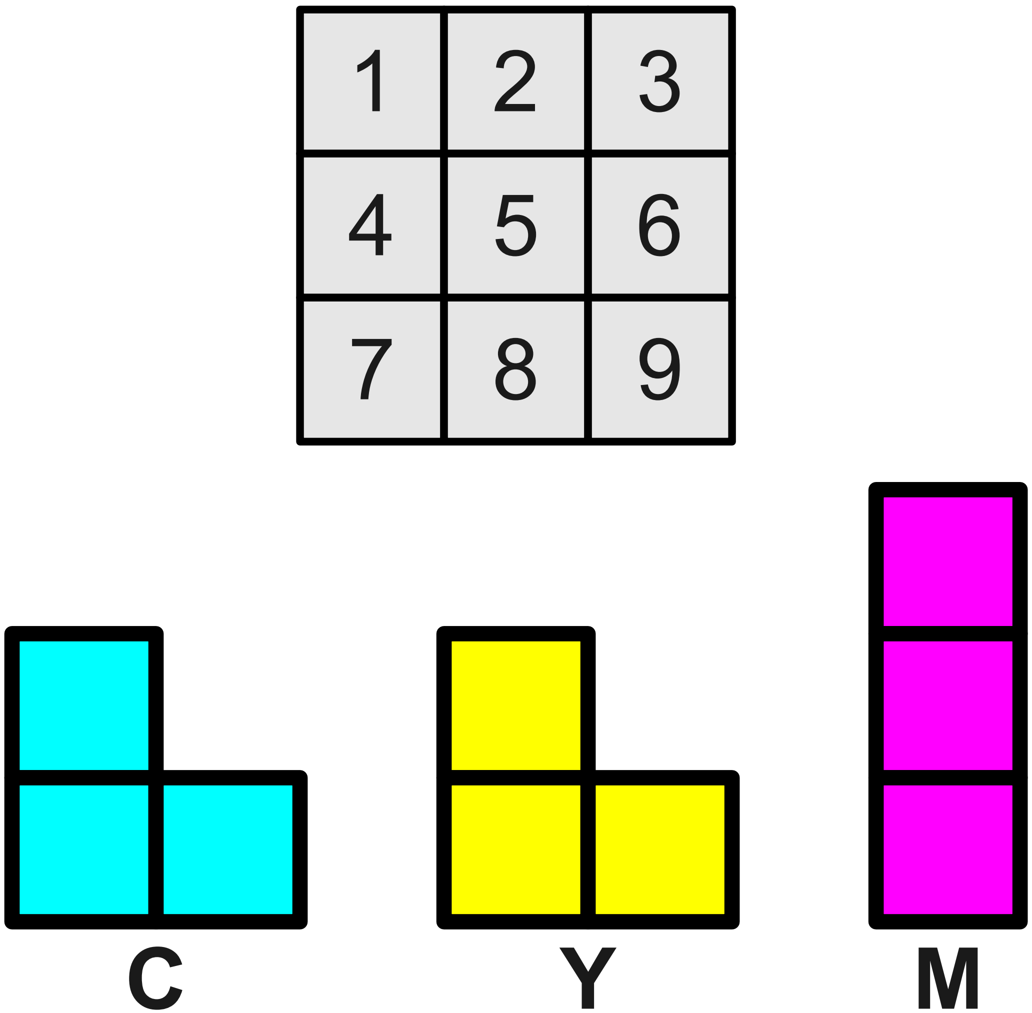
Applications of Exact Covering



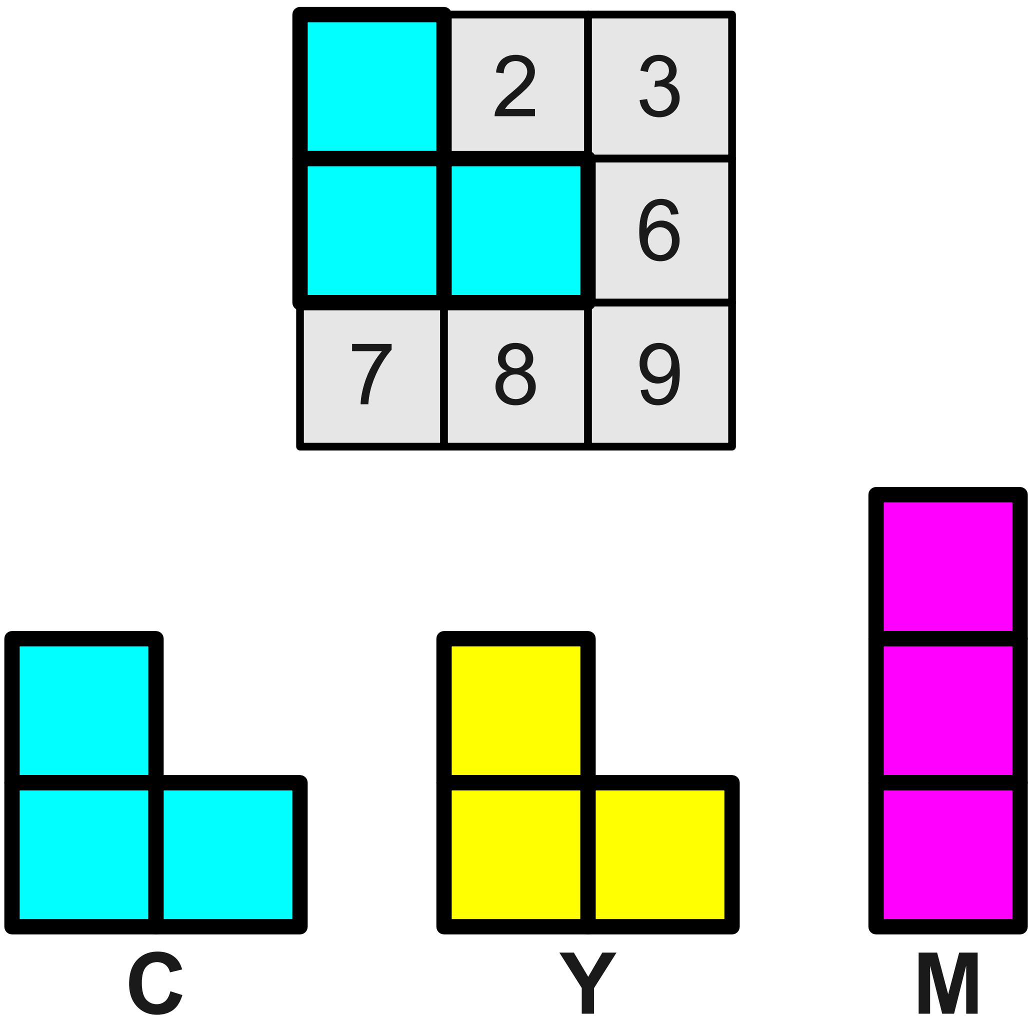
Applications of Exact Covering



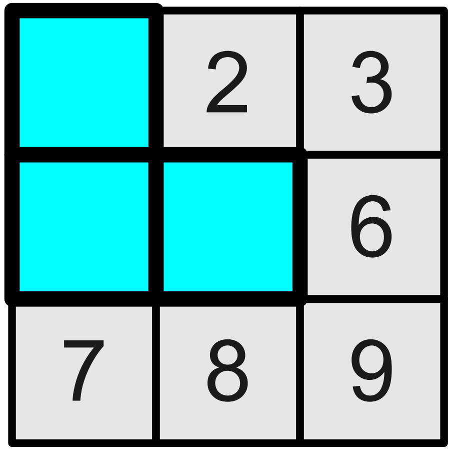
Applications of Exact Covering



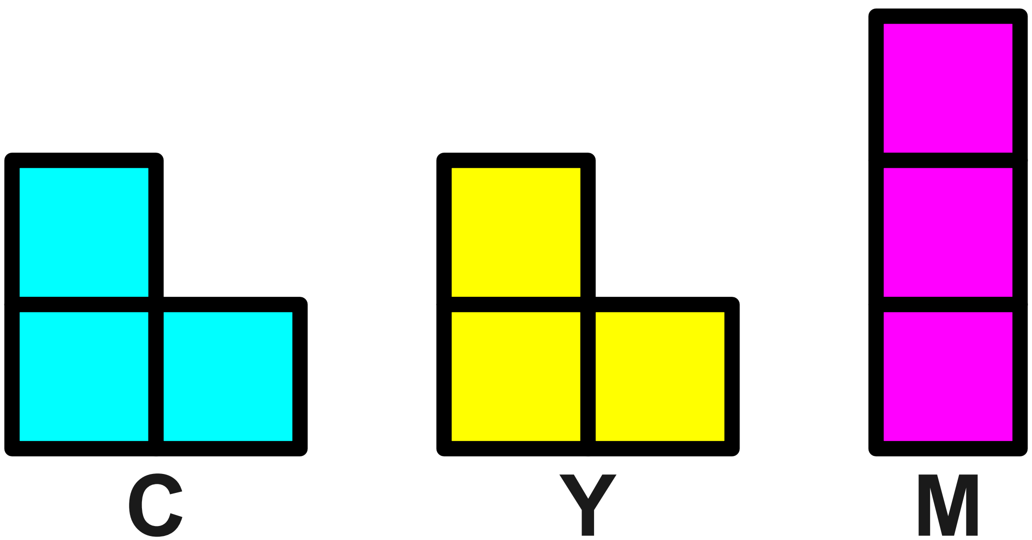
Applications of Exact Covering



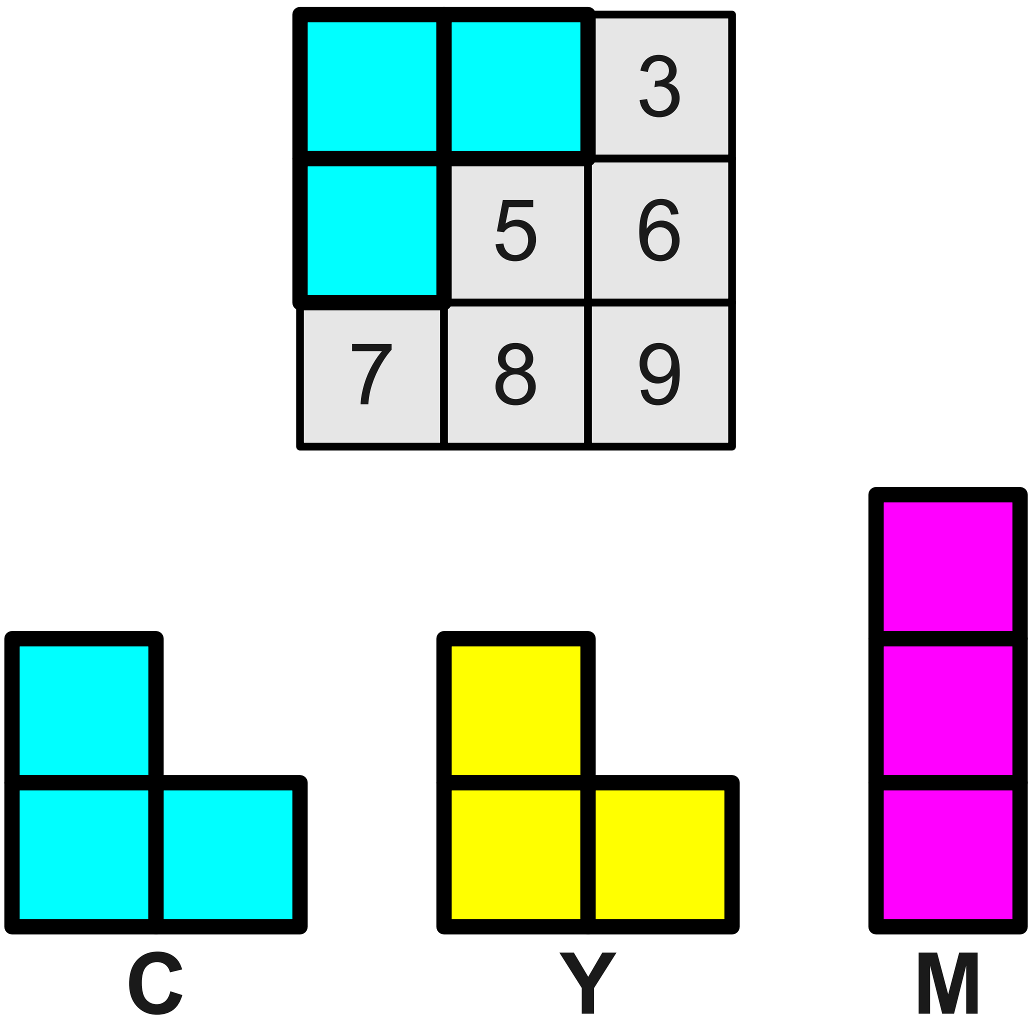
Applications of Exact Covering



{ C, 1, 4, 5 }

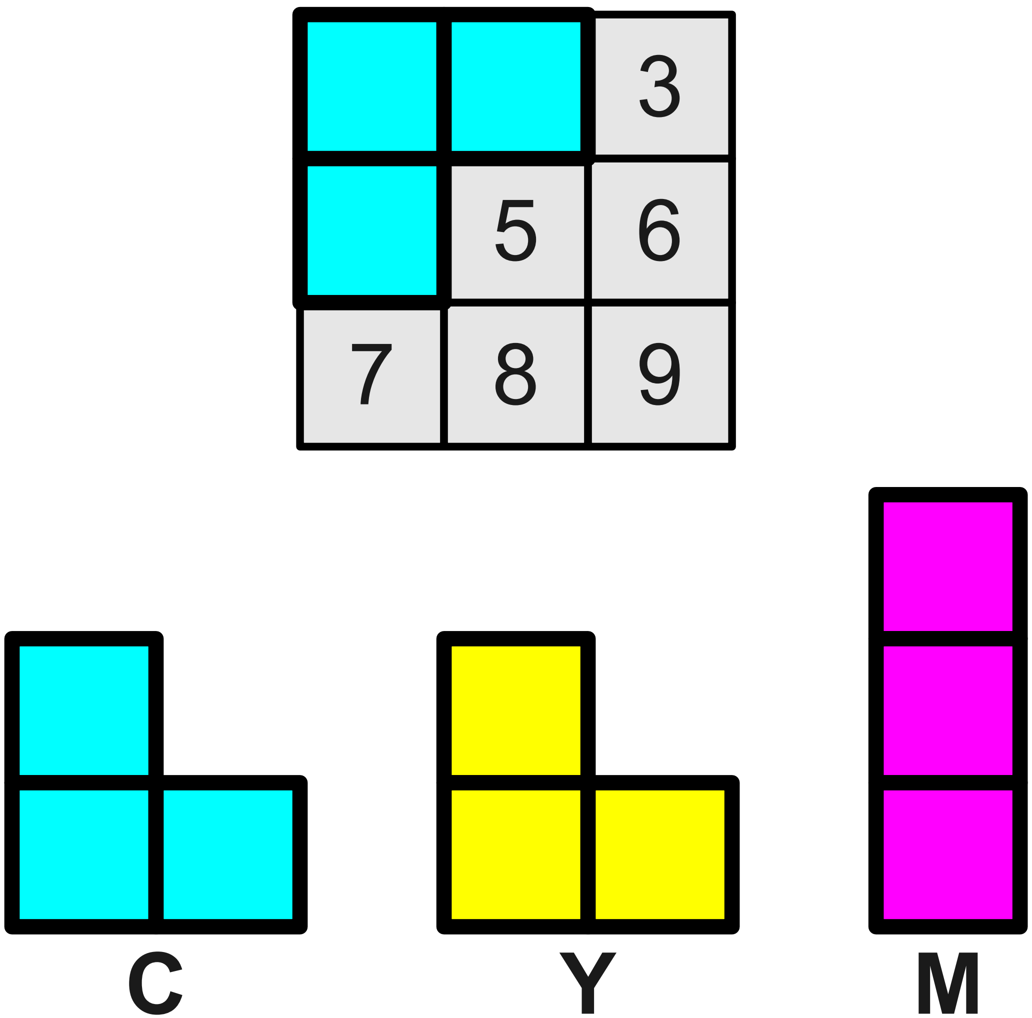


Applications of Exact Covering



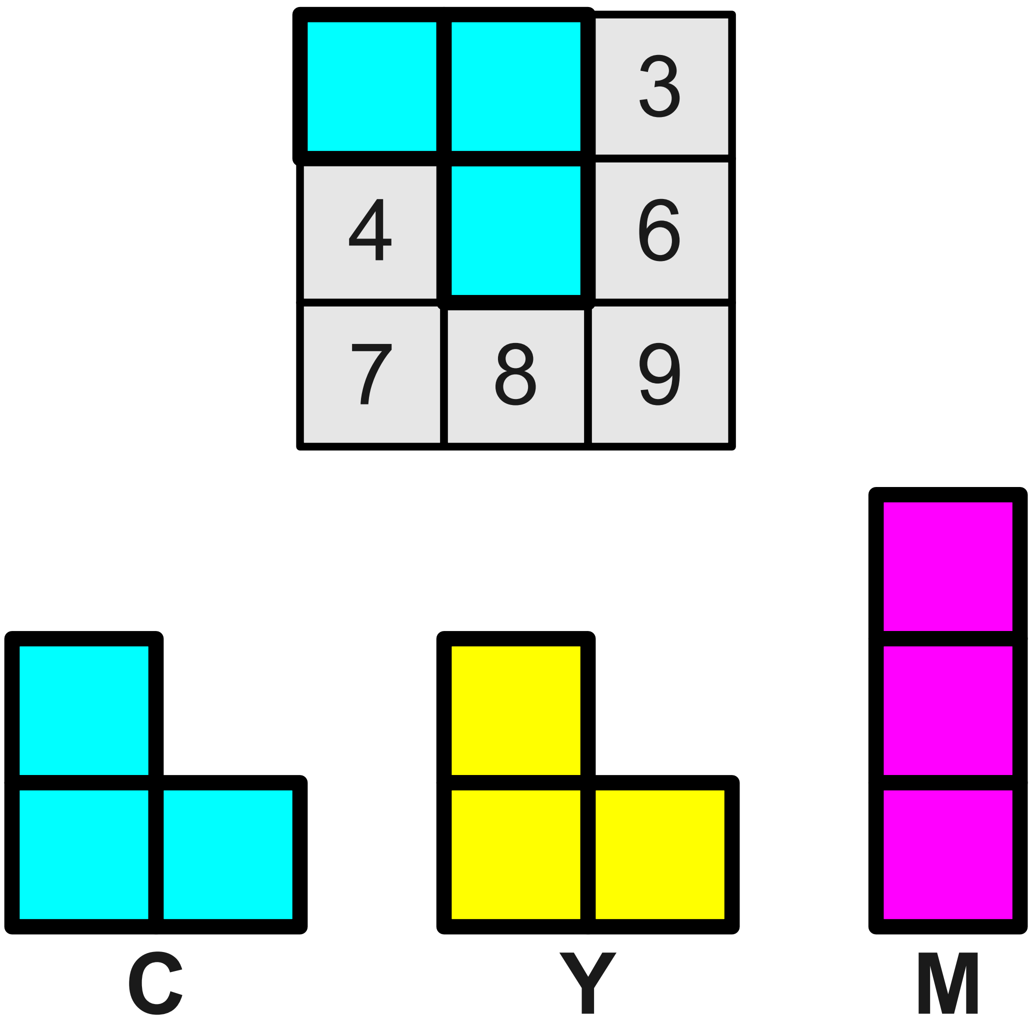
{ C, 1, 4, 5 }

Applications of Exact Covering



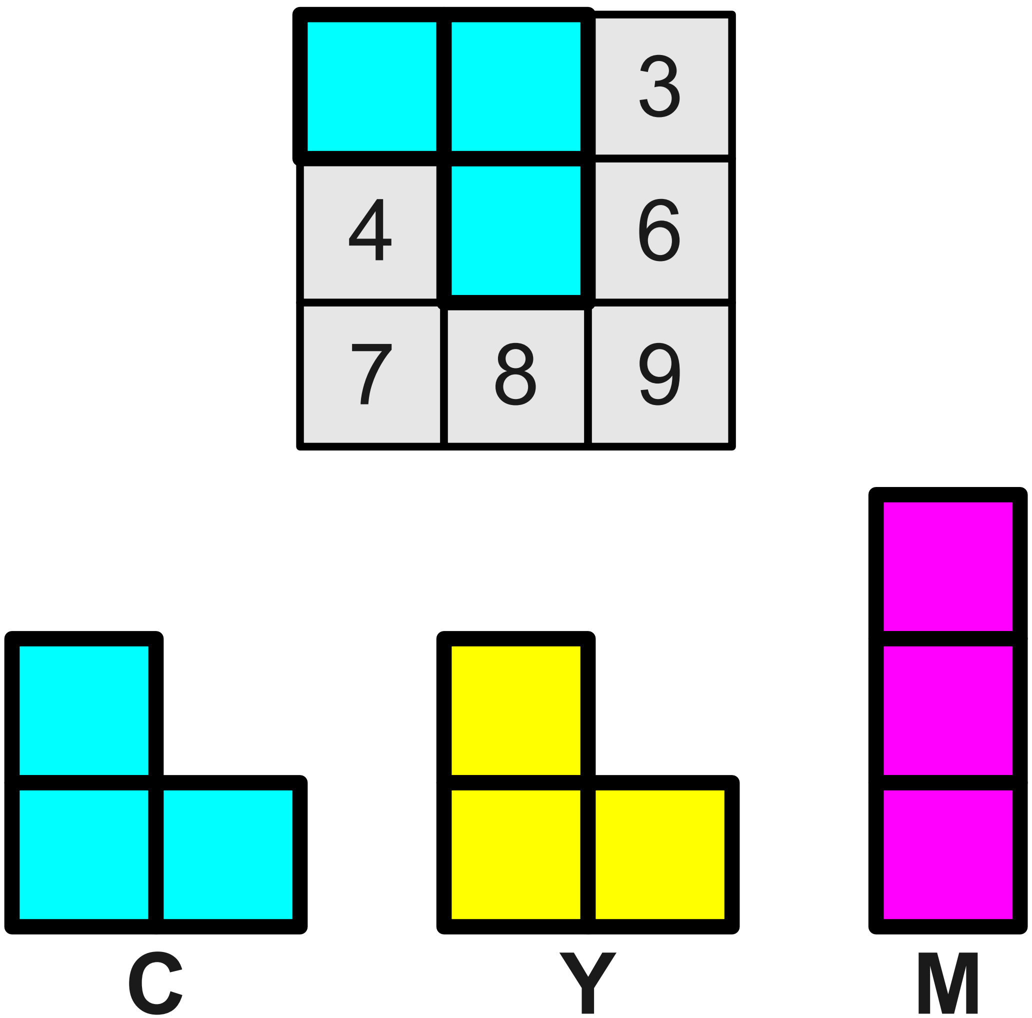
{ C, 1, 4, 5 }
{ C, 1, 2, 4 }

Applications of Exact Covering



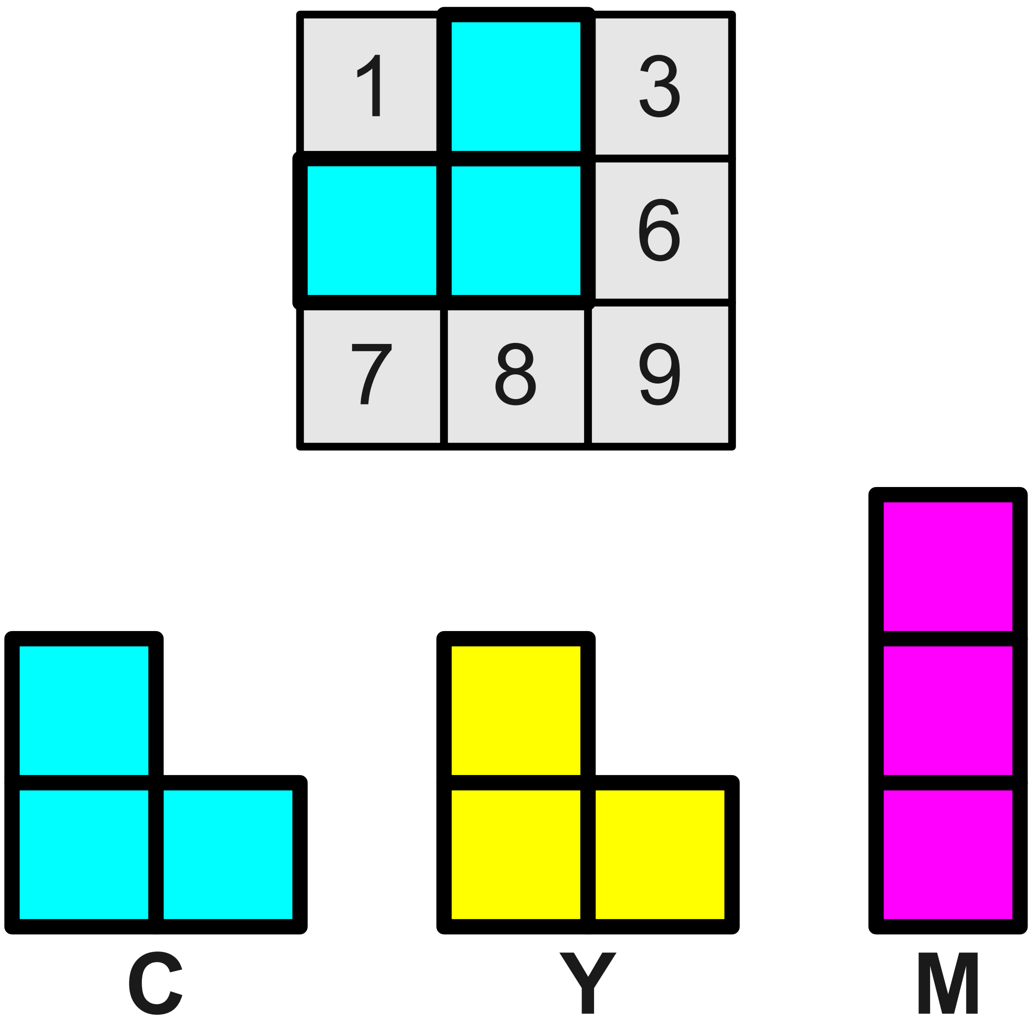
{ C, 1, 4, 5 }
{ C, 1, 2, 4 }

Applications of Exact Covering



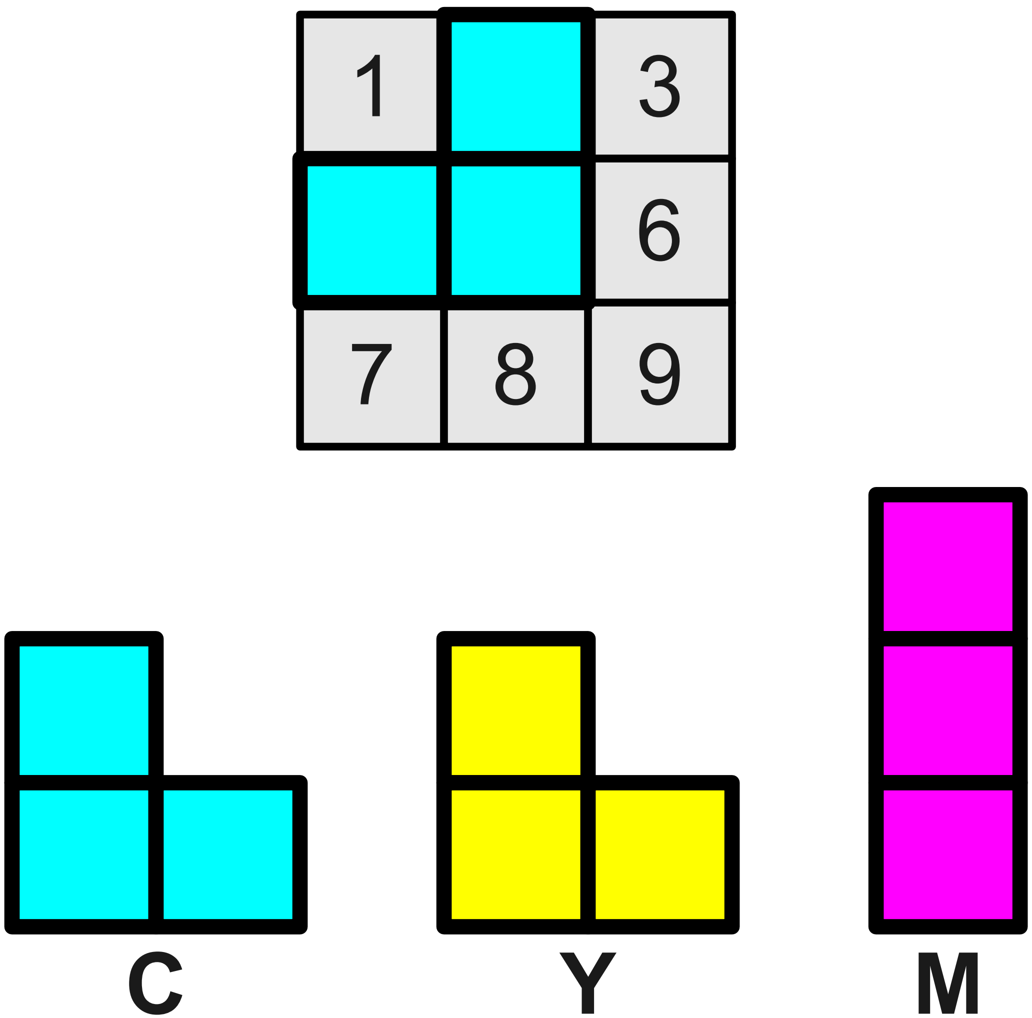
{ C, 1, 4, 5 }
{ C, 1, 2, 4 }
{ C, 1, 2, 5 }

Applications of Exact Covering



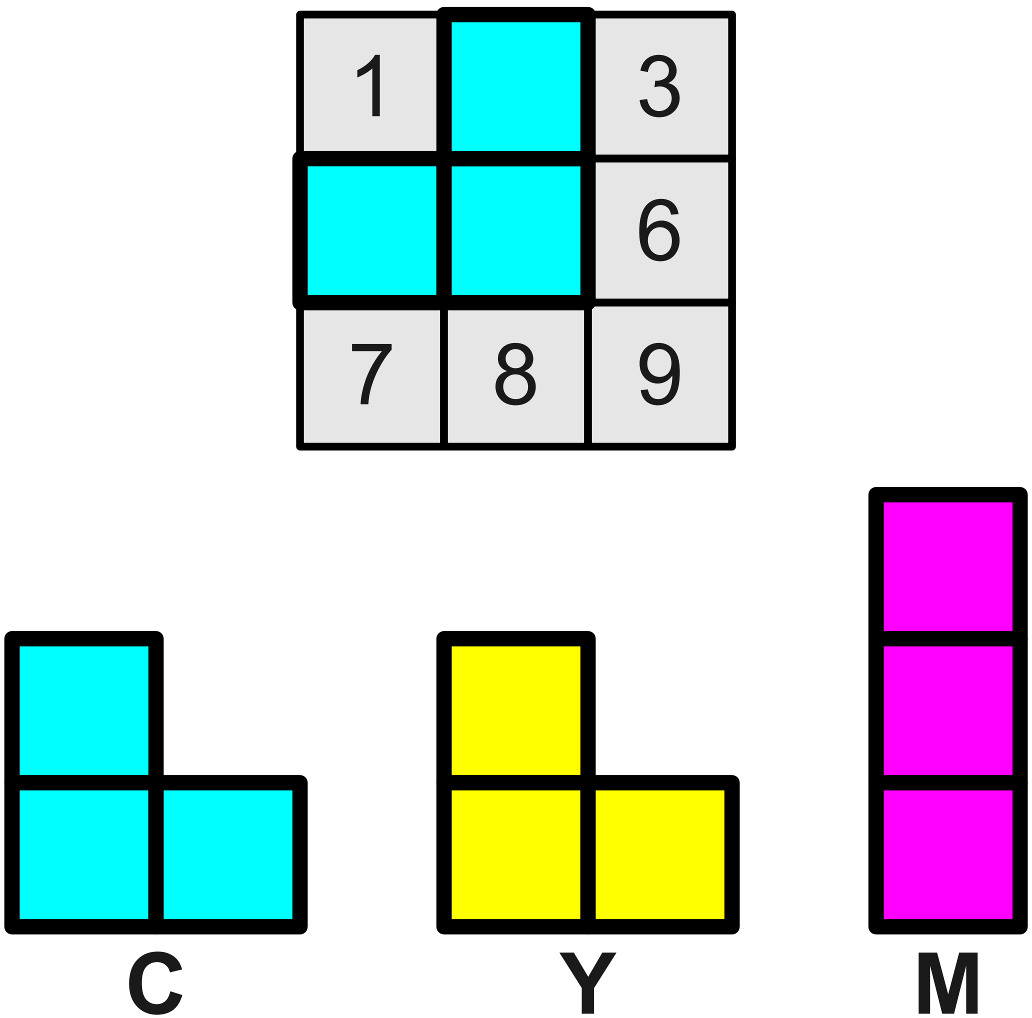
{ C, 1, 4, 5 }
{ C, 1, 2, 4 }
{ C, 1, 2, 5 }

Applications of Exact Covering



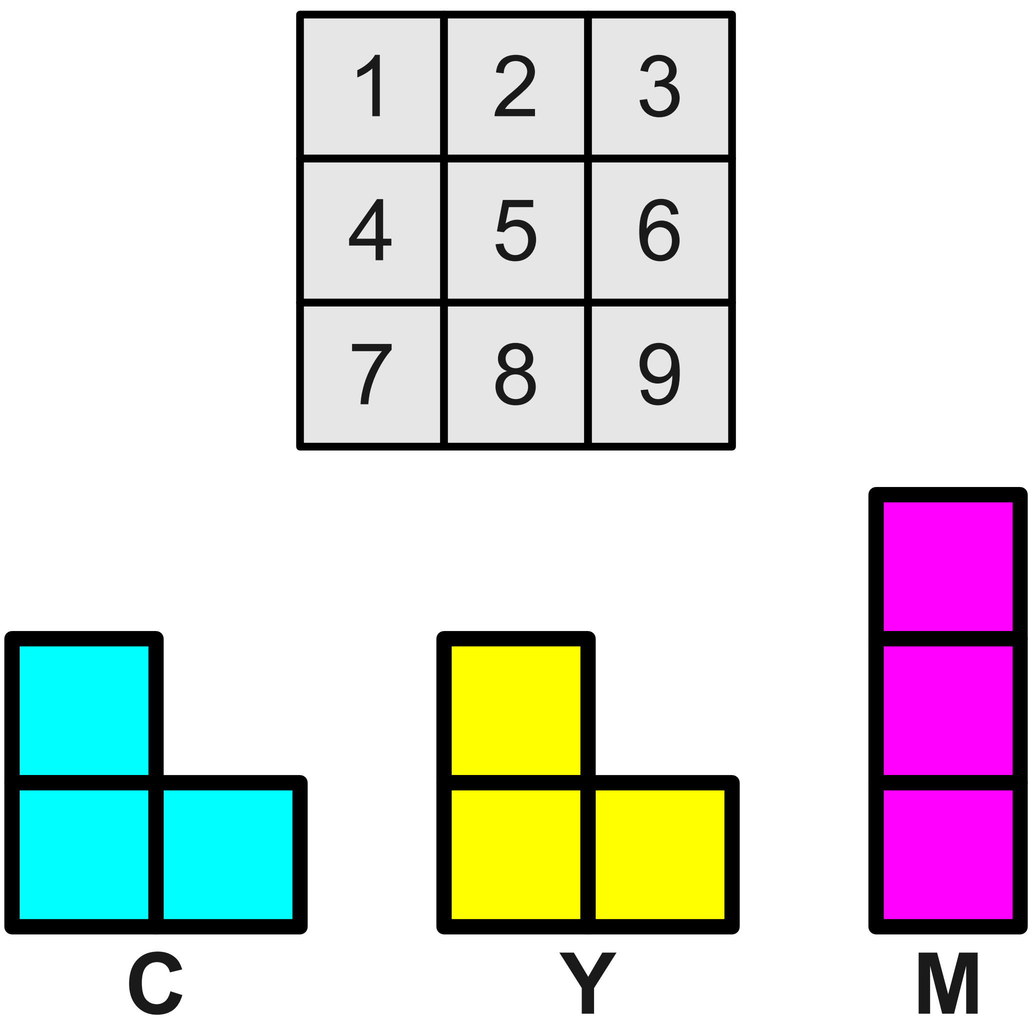
{ C, 1, 4, 5 }
{ C, 1, 2, 4 }
{ C, 1, 2, 5 }
{ C, 2, 4, 5 }

Applications of Exact Covering



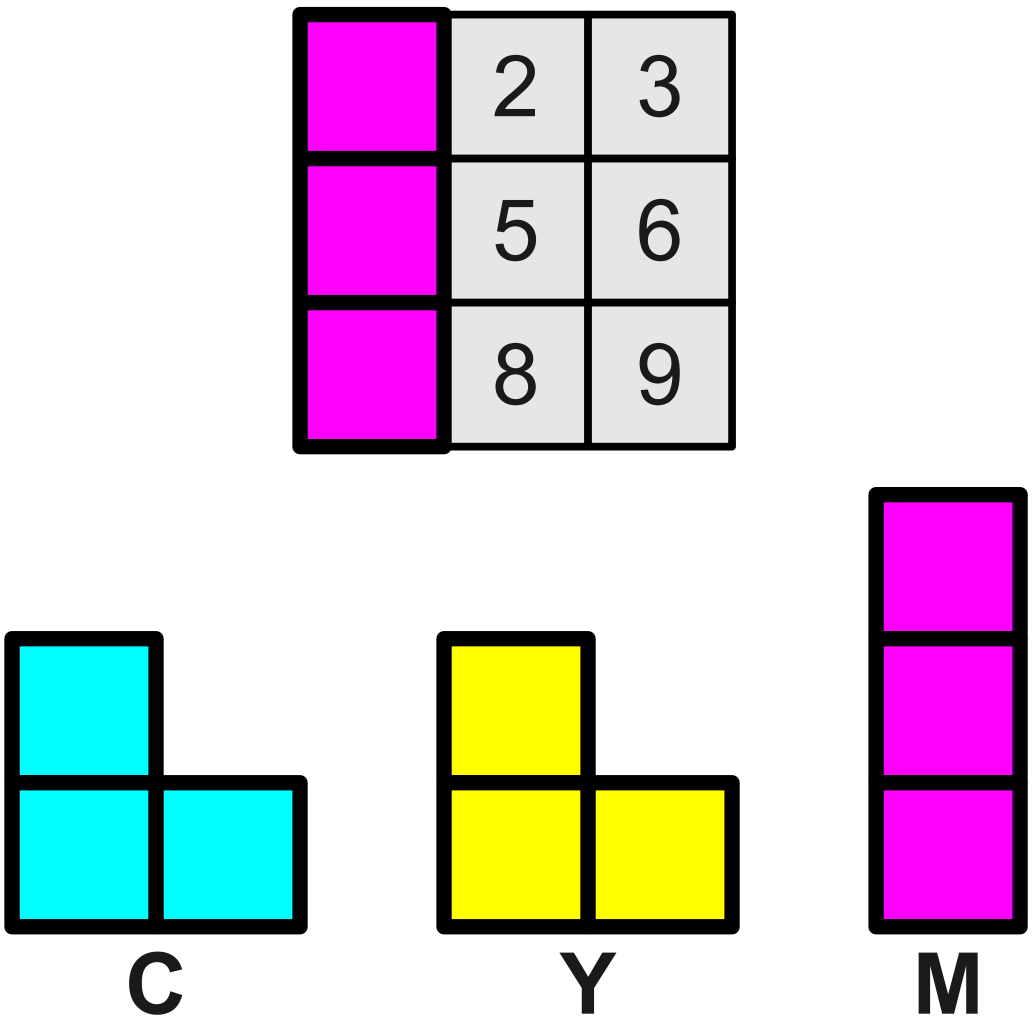
{ C, 1, 4, 5 }
{ C, 1, 2, 4 }
{ C, 1, 2, 5 }
{ C, 2, 4, 5 }
...

Applications of Exact Covering



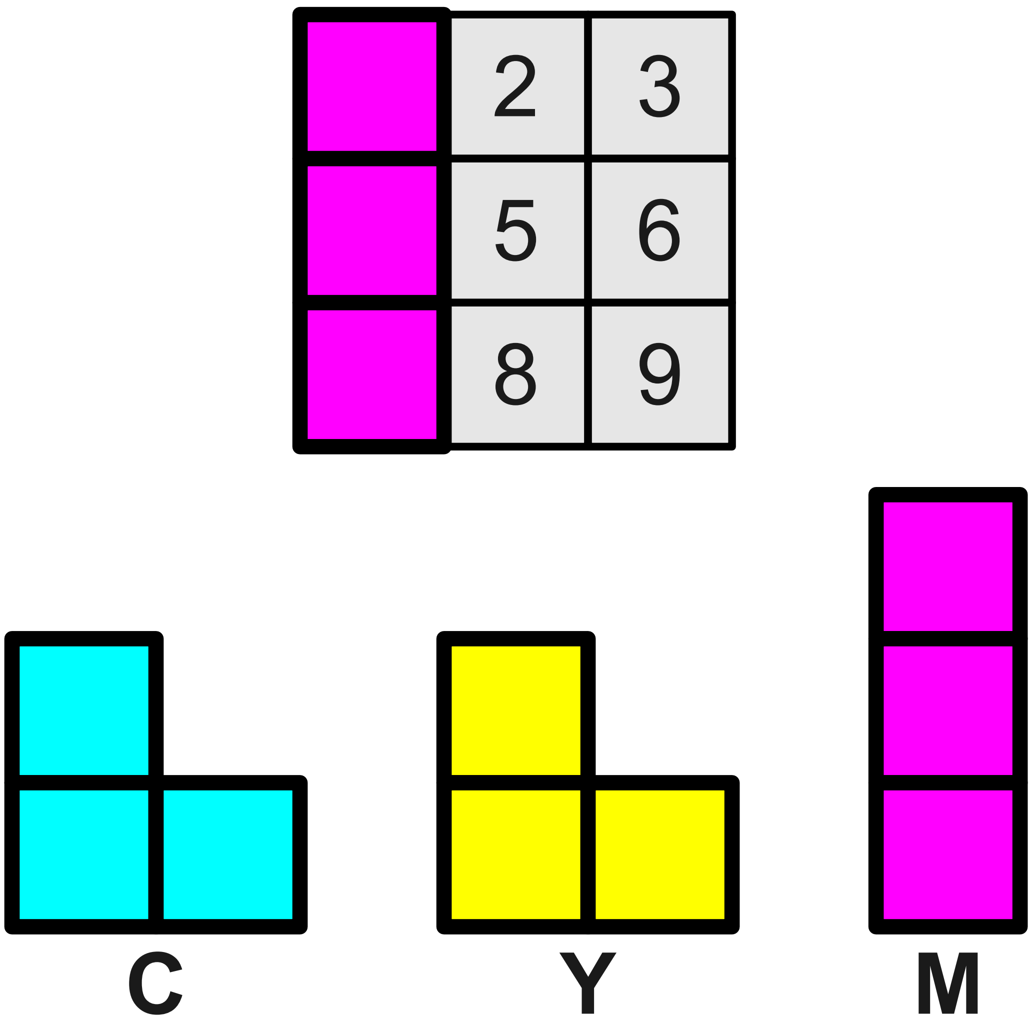
{ C, 1, 4, 5 }
{ C, 1, 2, 4 }
{ C, 1, 2, 5 }
{ C, 2, 4, 5 }
...

Applications of Exact Covering



{ C, 1, 4, 5 }
{ C, 1, 2, 4 }
{ C, 1, 2, 5 }
{ C, 2, 4, 5 }
...

Applications of Exact Covering

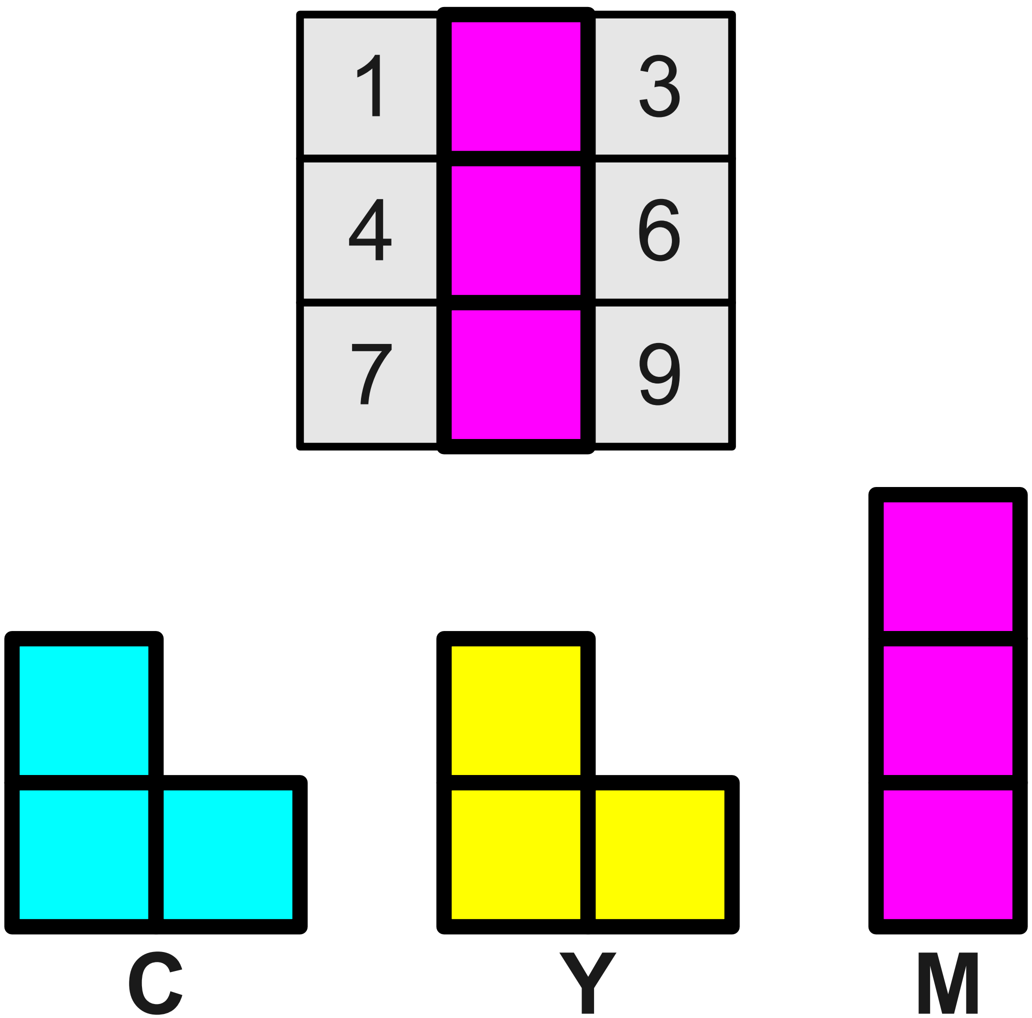


{ C, 1, 4, 5 }
{ C, 1, 2, 4 }
{ C, 1, 2, 5 }
{ C, 2, 4, 5 }

...

{ M, 1, 4, 7 }

Applications of Exact Covering

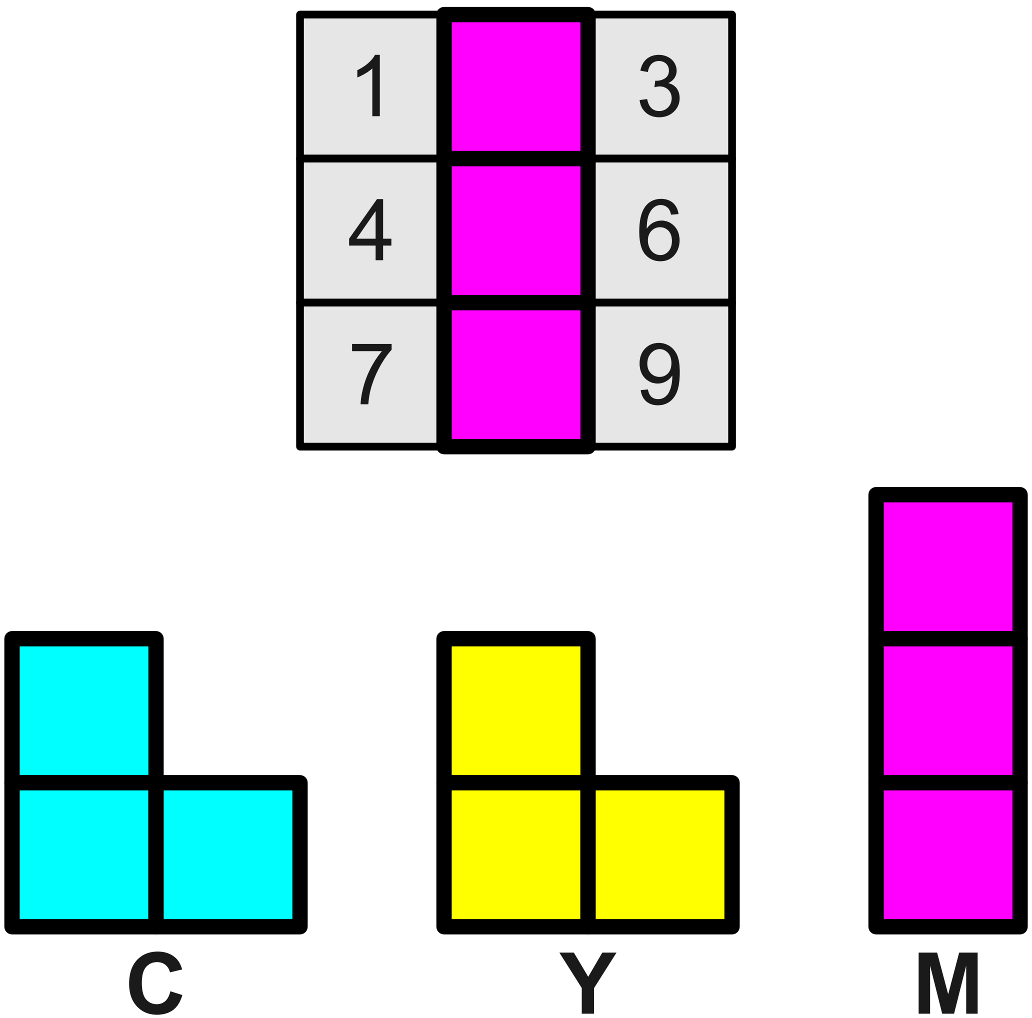


{ C, 1, 4, 5 }
{ C, 1, 2, 4 }
{ C, 1, 2, 5 }
{ C, 2, 4, 5 }

…

{ M, 1, 4, 7 }

Applications of Exact Covering

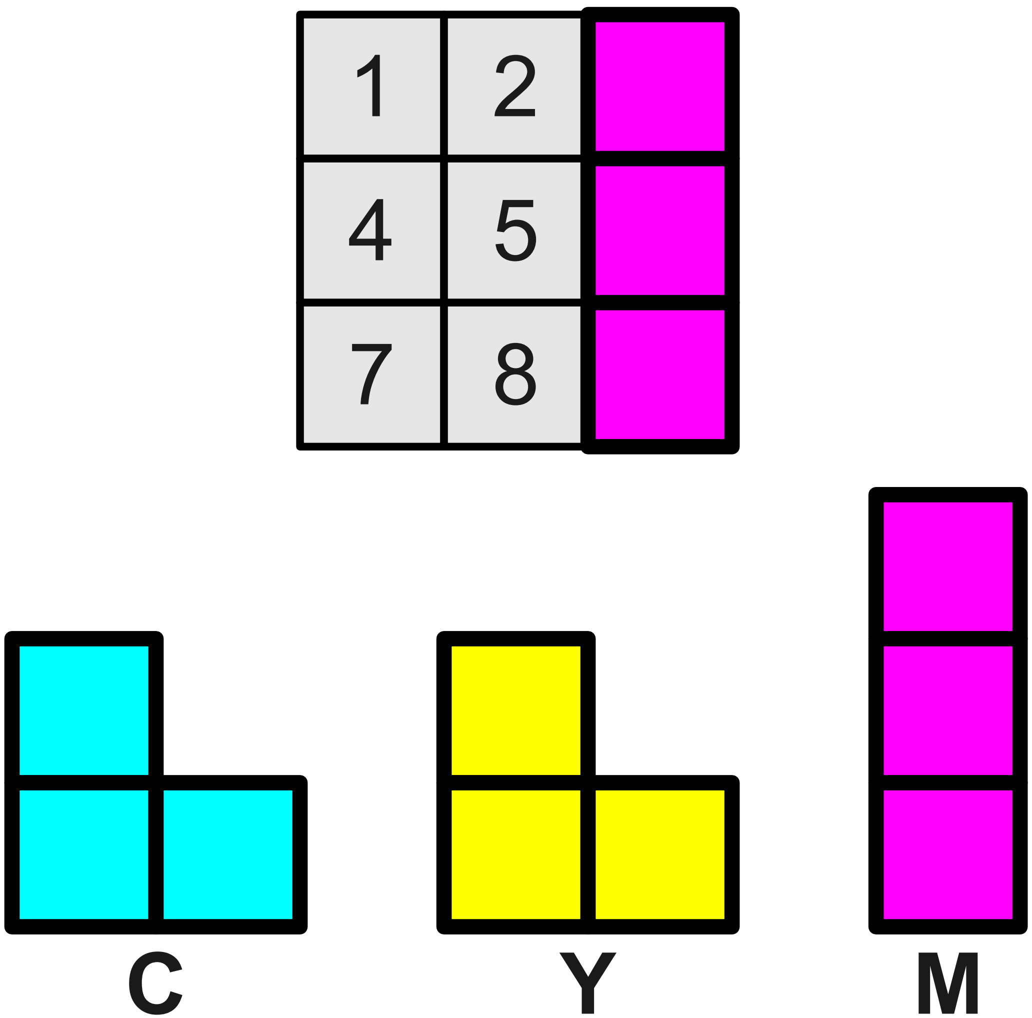


{ C, 1, 4, 5 }
{ C, 1, 2, 4 }
{ C, 1, 2, 5 }
{ C, 2, 4, 5 }

...

{ M, 1, 4, 7 }
{ M, 2, 5, 8 }

Applications of Exact Covering

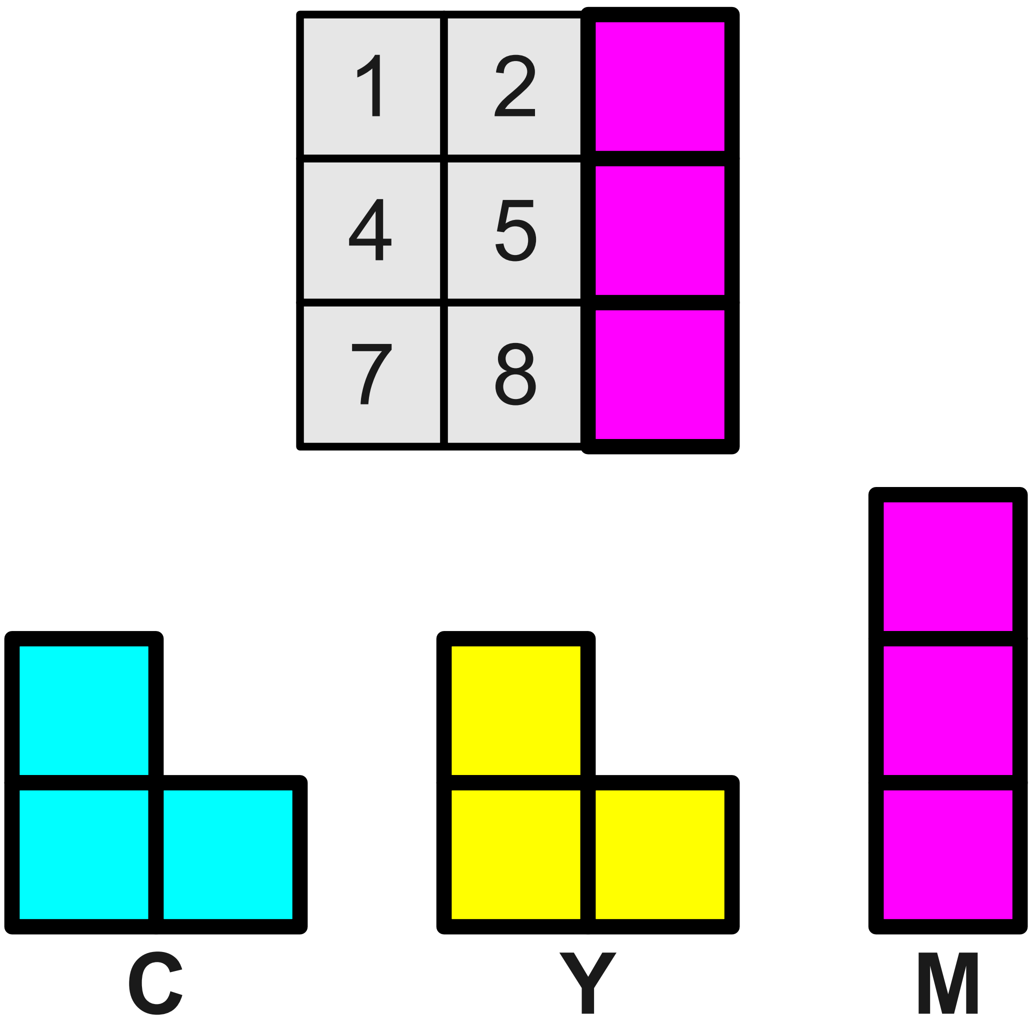


{ C, 1, 4, 5 }
{ C, 1, 2, 4 }
{ C, 1, 2, 5 }
{ C, 2, 4, 5 }

...

{ M, 1, 4, 7 }
{ M, 2, 5, 8 }

Applications of Exact Covering

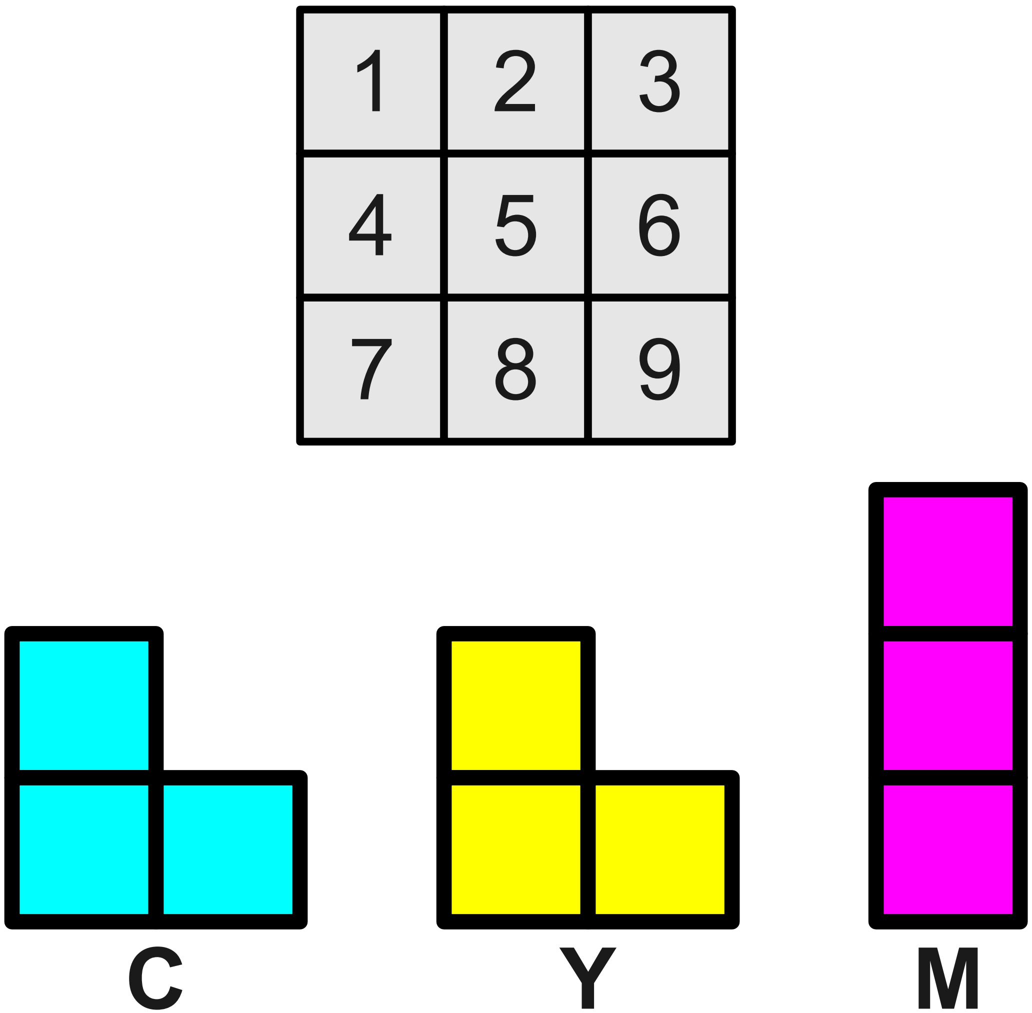


{ C, 1, 4, 5 }
{ C, 1, 2, 4 }
{ C, 1, 2, 5 }
{ C, 2, 4, 5 }

...

{ M, 1, 4, 7 }
{ M, 2, 5, 8 }
{ M, 3, 6, 9 }

Applications of Exact Covering



{ C, 1, 4, 5 }
{ C, 1, 2, 4 }
{ C, 1, 2, 5 }
{ C, 2, 4, 5 }

...

{ M, 1, 4, 7 }
{ M, 2, 5, 8 }
{ M, 3, 6, 9 }

...

Trust me, these reductions matter.

We'll see why in a few minutes.

The

Most Important Question

in

Theoretical Computer Science

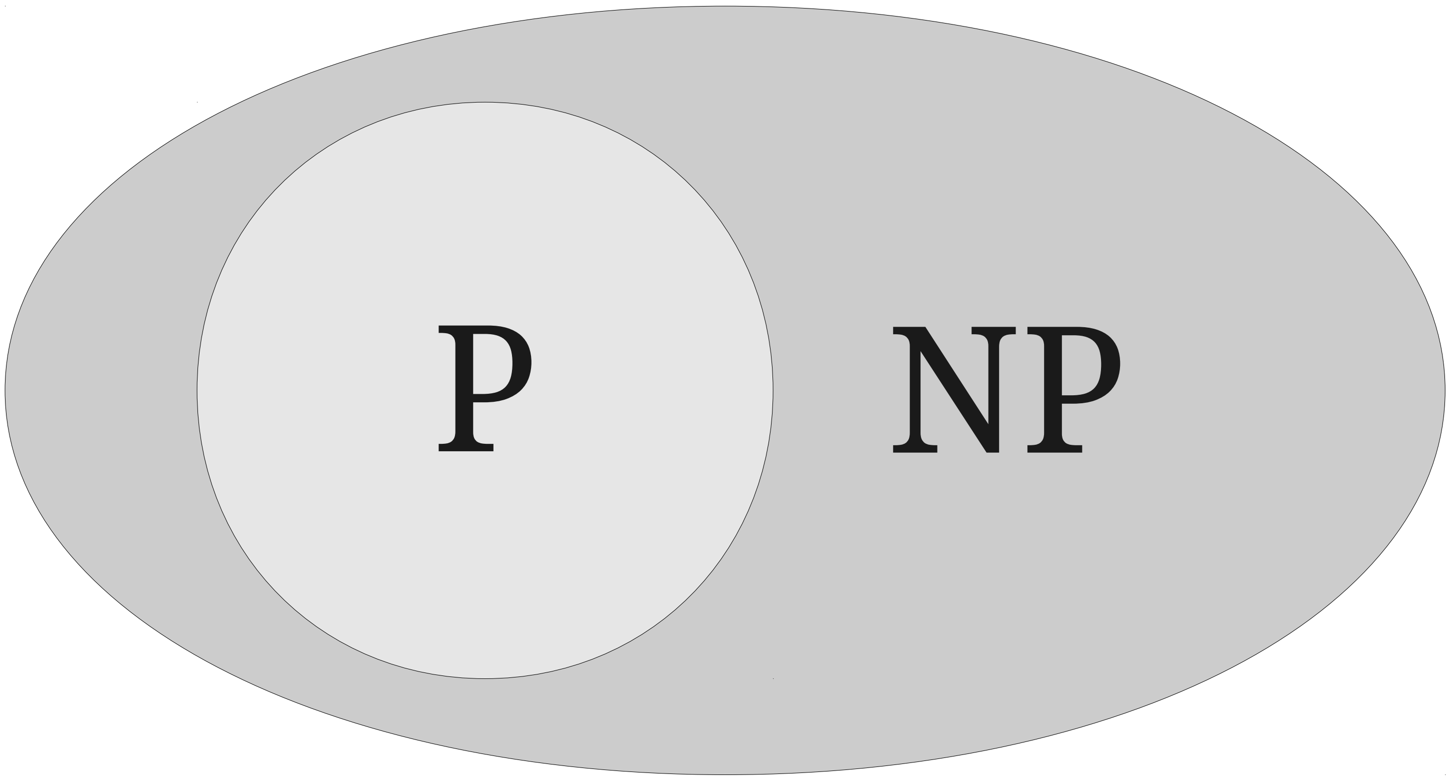
What is the connection between **P** and **NP**?

$$\mathbf{P} = \{\ L \mid \text{There is a polynomial-time decider for } L\ \}$$

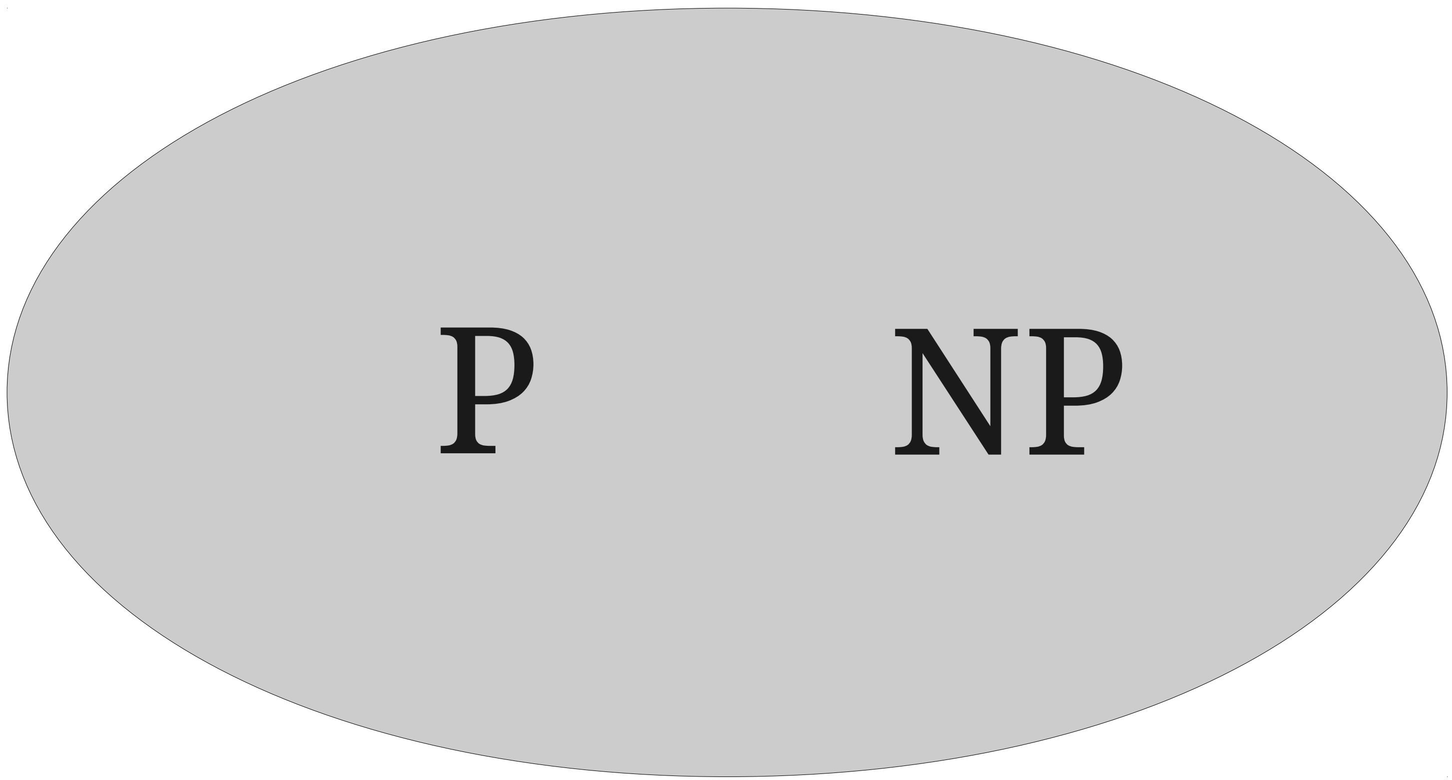
$$\mathbf{NP} = \{\ L \mid \text{There is a nondeterministic polynomial-time decider for } L\ \}$$

$$\mathbf{P} \subseteq \mathbf{NP}$$

Which Picture is Correct?



Which Picture is Correct?



Does **P** = **NP**?

$\mathbf{P} \stackrel{?}{=} \mathbf{NP}$

- The $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ question is the most important question in theoretical computer science.
- With the verifier definition of **NP**, one way of phrasing this question is

If a solution to a problem can be **verified** efficiently, can that problem be **solved** efficiently?

- An answer either way will give fundamental insights into the nature of computation.

Why This Matters

- The following problems are known to be efficiently verifiable, but have no known efficient solutions:
  - Determining whether an electrical grid can be built to link up some number of houses for some price (Steiner tree problem).
  - Determining whether a simple DNA strand exists that multiple gene sequences could be a part of (shortest common supersequence).
  - Determining the best way to assign hardware resources in a compiler (optimal register allocation).
  - Determining the best way to distribute tasks to multiple workers to minimize completion time (job scheduling).
  - **And many more**.
- If **P = NP**, **all** of these problems have efficient solutions.
- If **P ≠ NP**, **none** of these problems have efficient solutions.

Why This Matters

- If **P = NP**:
  - A huge number of seemingly difficult problems could be solved efficiently.
  - Our capacity to solve many problems will scale well with the size of the problems we want to solve.
- If **P ≠ NP**:
  - Enormous computational power would be required to solve many seemingly easy tasks.
  - Our capacity to solve problems will fail to keep up with our curiosity.

What We Know

- Resolving $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ has proven ***extremely difficult.***
- In the past 35 years:
  - Not a single correct proof either way has been found.
  - Many types of proofs have been shown to be insufficiently powerful to determine whether $\mathbf{P} = \mathbf{NP}$.
  - A majority of computer scientists believe $\mathbf{P} \neq \mathbf{NP}$, but this isn't a large majority.
- Interesting read: Interviews with leading thinkers about $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$:
  - http://web.ing.puc.cl/~jabaier/iic2212/poll-1.pdf

The Million-Dollar Question

The Clay Mathematics Institute has offered a **$1,000,000 prize** to anyone who proves or disproves **P = NP**.

The Million-Dollar Question

The Clay Mathematics Institute has offered a **$1,000,000 prize** to anyone who proves or disproves **P = NP**.

Time-Out For Announcements

Please evaluate this course in Axess.

Your feedback really does make a difference.

Final Exam Logistics

- Final exam is this upcoming Monday, December 9th from 12:15PM – 3:15PM.
- Room information TBA; we're still finalizing everything.
- Exam is cumulative, but focuses primarily on material from DFAs onward.
  - Take a look a the practice exams for a sense of what the coverage will be like.

Practice Finals

- We have three practice exams available right now:
  - An **extra credit** practice exam worth +5 EC points.
  - Two actual final exams from previous quarters, which are good for studying but not worth any extra credit.
- Solutions to the two additional practice finals will be released Wednesday.
- **Please take the additional final exams under realistic conditions** so that you can get a sense of where you stand. Most of the problems are “nondeterministically trivial.”

A Note on Honesty and Integrity

Review Sessions

- We will be holding at least one final exam review session later this week.
- We will announce date and time information once it's finalized.
- Feel free to show up with any questions you'd like answered!

Casual CS Dinner

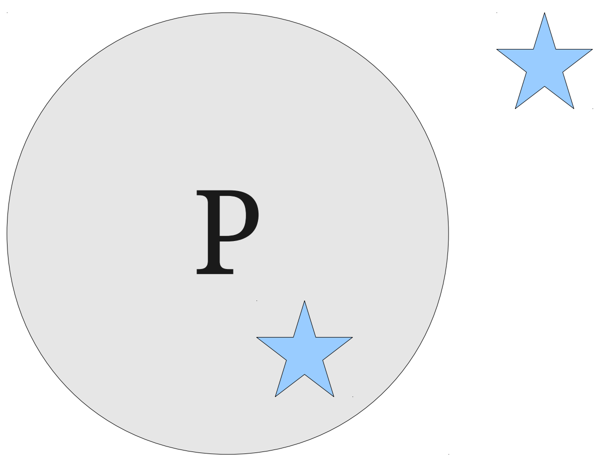
- The second biquarterly Casual CS Dinner for Women in CS is **tonight** at 6PM on the fifth floor of Gates.
- Everyone is welcome!
- RSVP appreciated; check the email sent to the CS103 list.

Back to CS103!

NP-Completeness

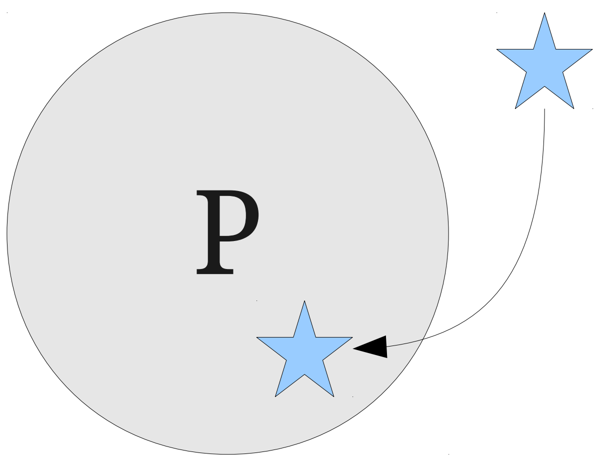
Polynomial-Time Reductions

- If $L_1 \leq_P L_2$ and $L_2 \in \mathbf{P}$, then $L_1 \in \mathbf{P}$.



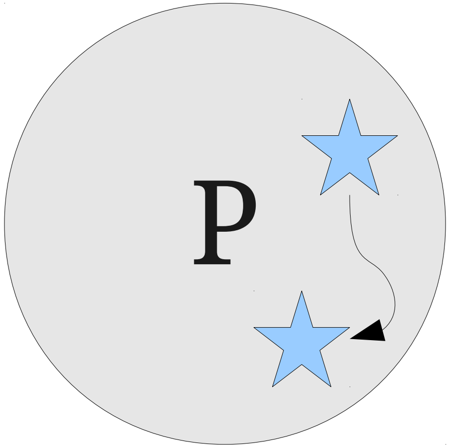
Polynomial-Time Reductions

- If $L_1 \leq_P L_2$ and $L_2 \in \mathbf{P}$, then $L_1 \in \mathbf{P}$.



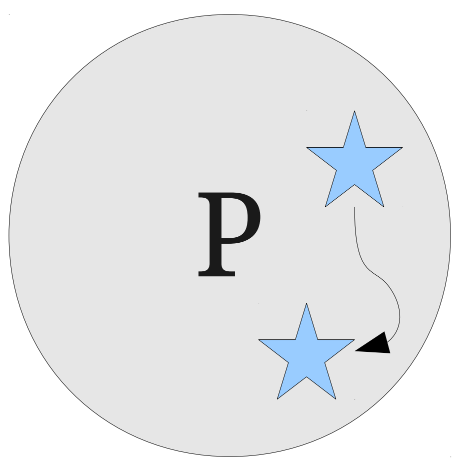
Polynomial-Time Reductions

- If $L_1 \leq_{\text{P}} L_2$ and $L_2 \in \mathbf{P}$, then $L_1 \in \mathbf{P}$.



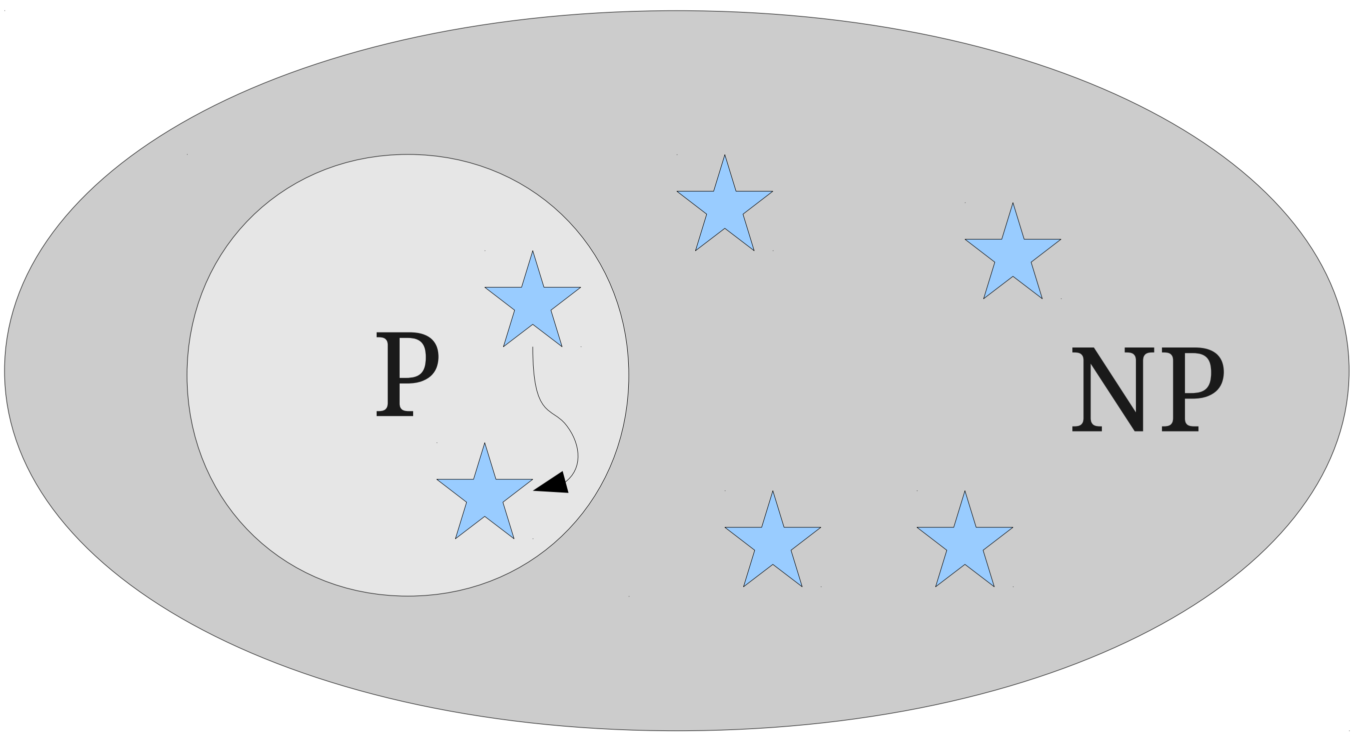
Polynomial-Time Reductions

- If $L_1 \leq_{\mathrm{P}} L_2$ and $L_2 \in \mathbf{P}$, then $L_1 \in \mathbf{P}$.
- If $L_1 \leq_{\mathrm{P}} L_2$ and $L_2 \in \mathbf{NP}$, then $L_1 \in \mathbf{NP}$.



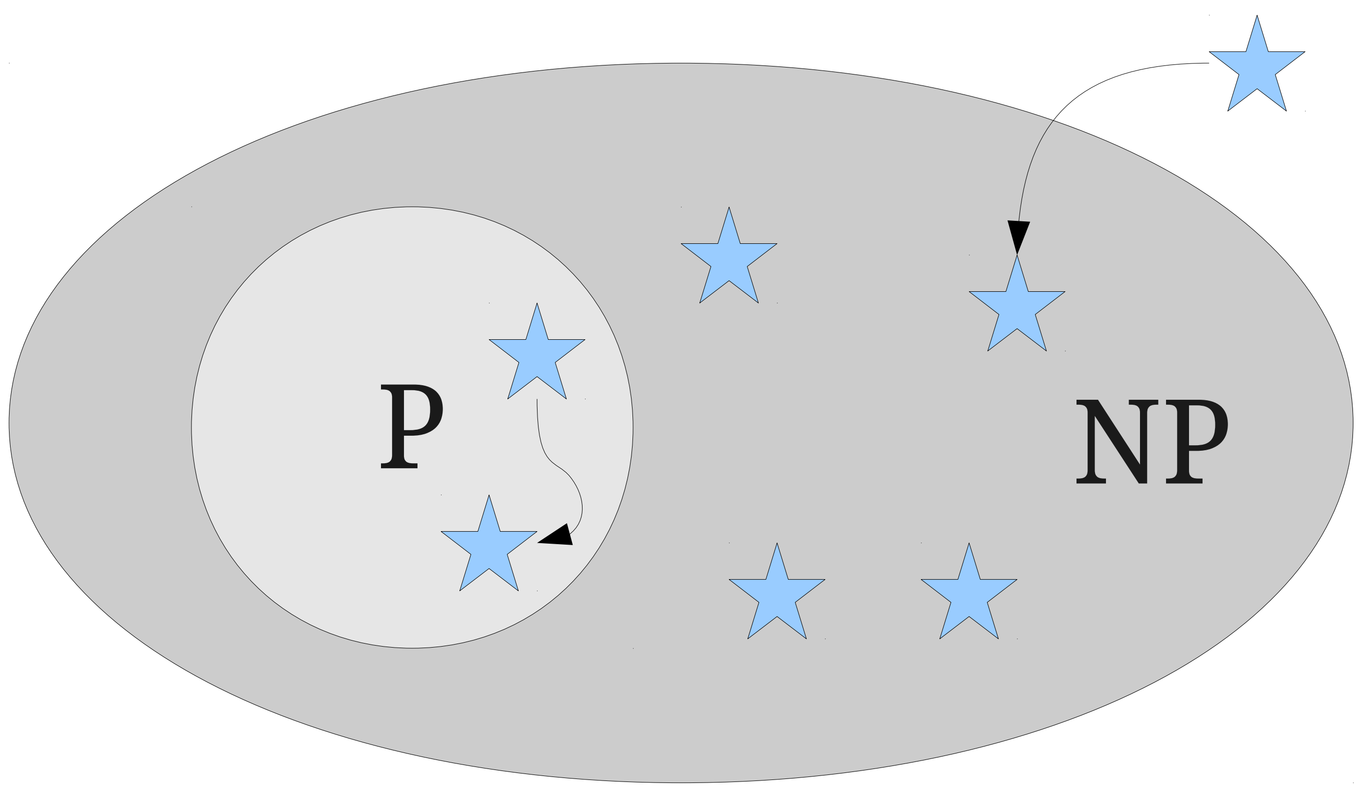
Polynomial-Time Reductions

- If $L_1 \leq_P L_2$ and $L_2 \in \mathbf{P}$, then $L_1 \in \mathbf{P}$.
- If $L_1 \leq_P L_2$ and $L_2 \in \mathbf{NP}$, then $L_1 \in \mathbf{NP}$.



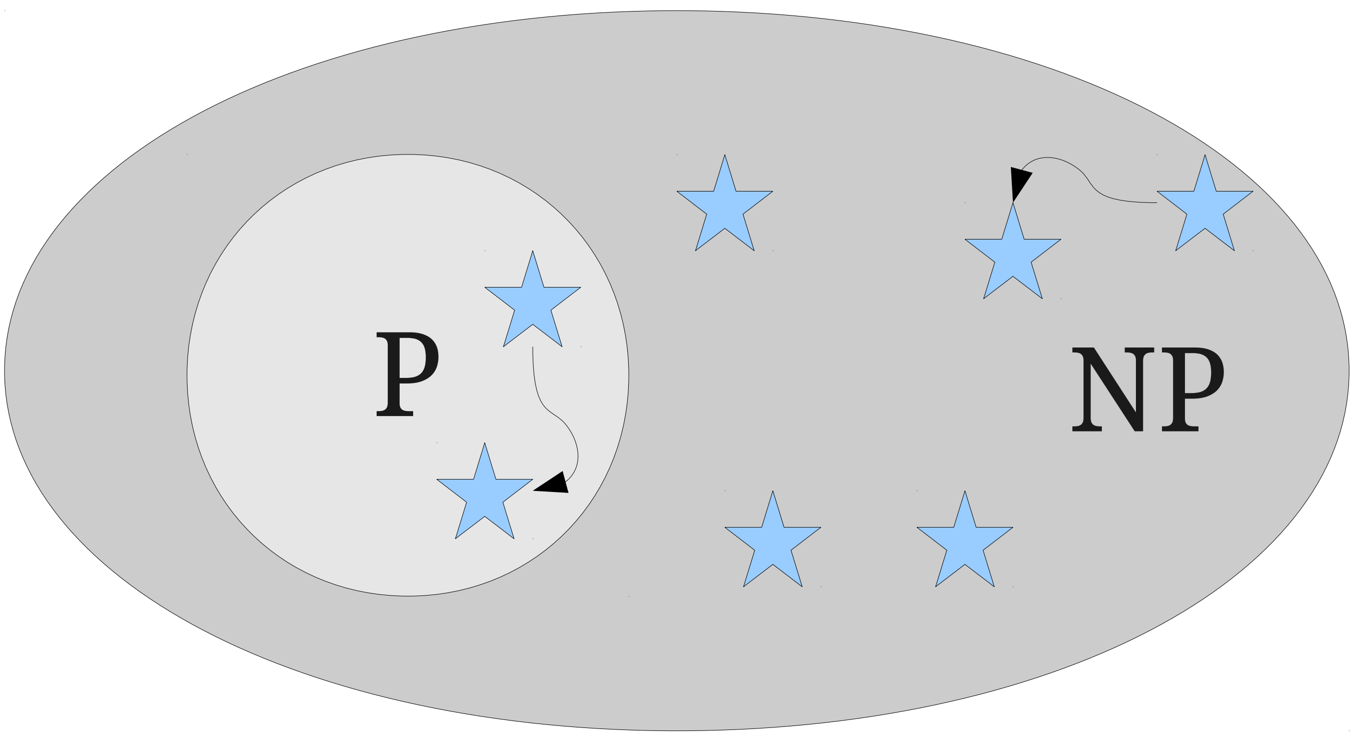
Polynomial-Time Reductions

- If $L_1 \leq_P L_2$ and $L_2 \in \mathbf{P}$, then $L_1 \in \mathbf{P}$.
- If $L_1 \leq_P L_2$ and $L_2 \in \mathbf{NP}$, then $L_1 \in \mathbf{NP}$.



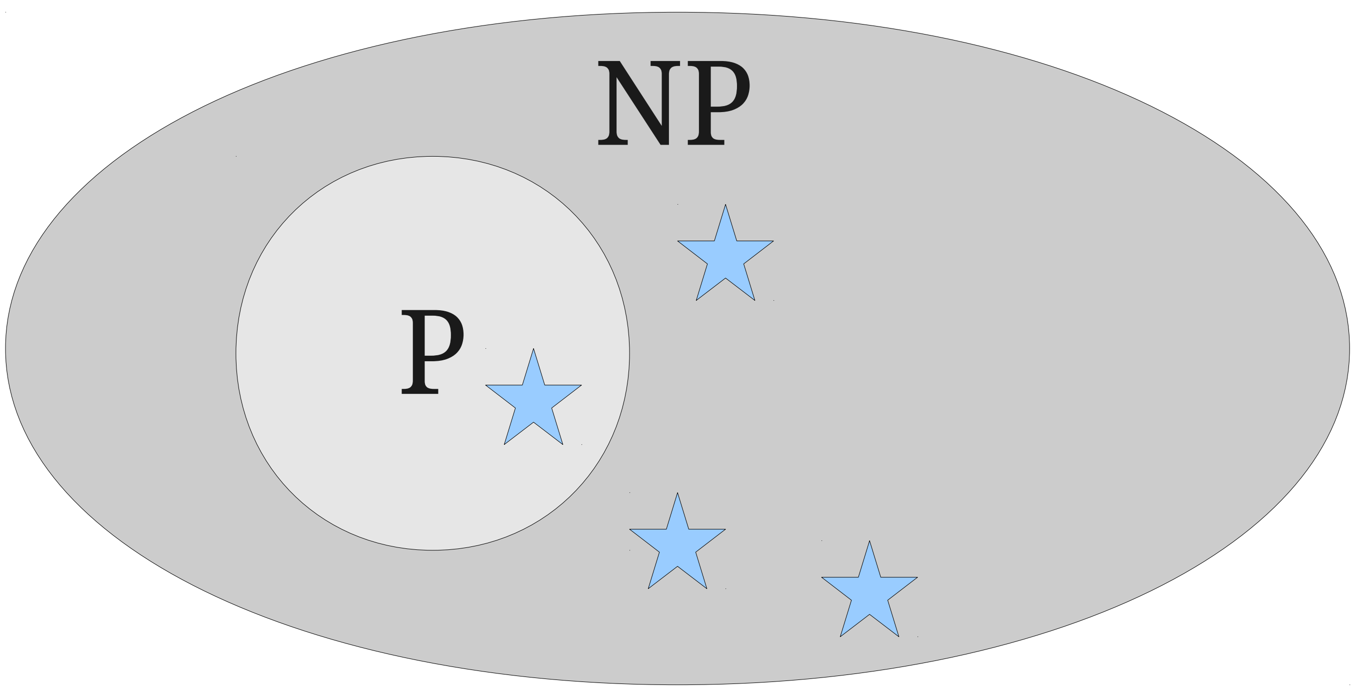
Polynomial-Time Reductions

- If $L_1 \leq_P L_2$ and $L_2 \in \mathbf{P}$, then $L_1 \in \mathbf{P}$.
- If $L_1 \leq_P L_2$ and $L_2 \in \mathbf{NP}$, then $L_1 \in \mathbf{NP}$.



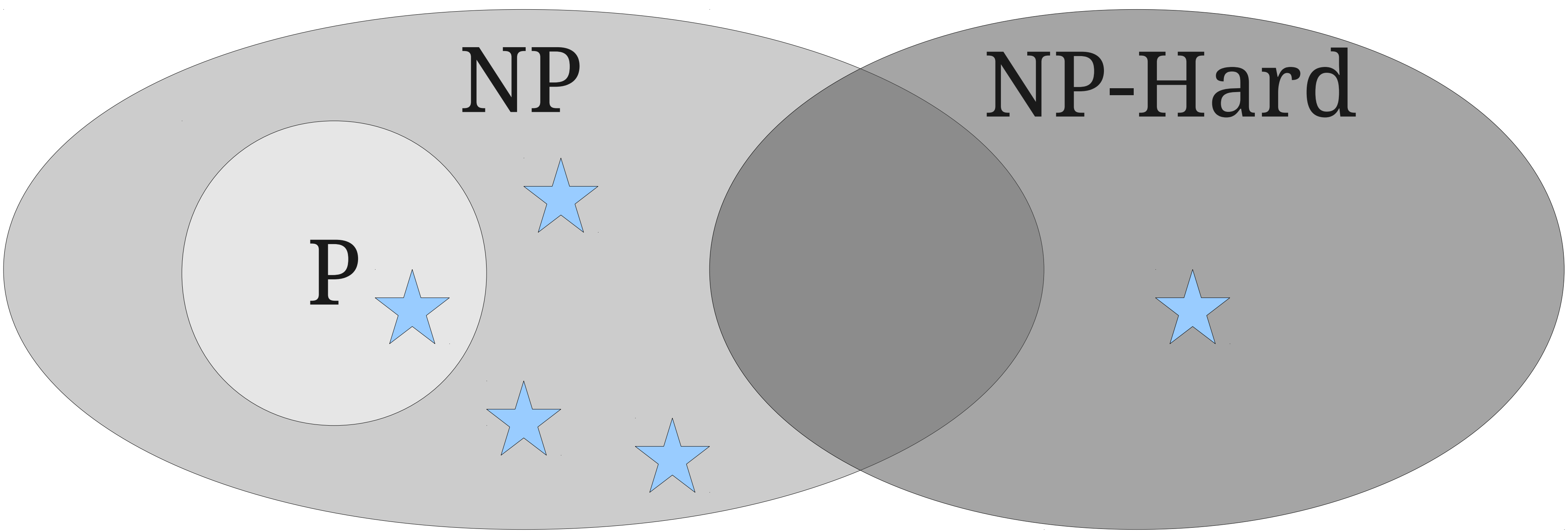
NP-Hardness

- A language L is called **NP-hard** iff for *every* $L' \in$ **NP**, we have $L' \leq_P L$.



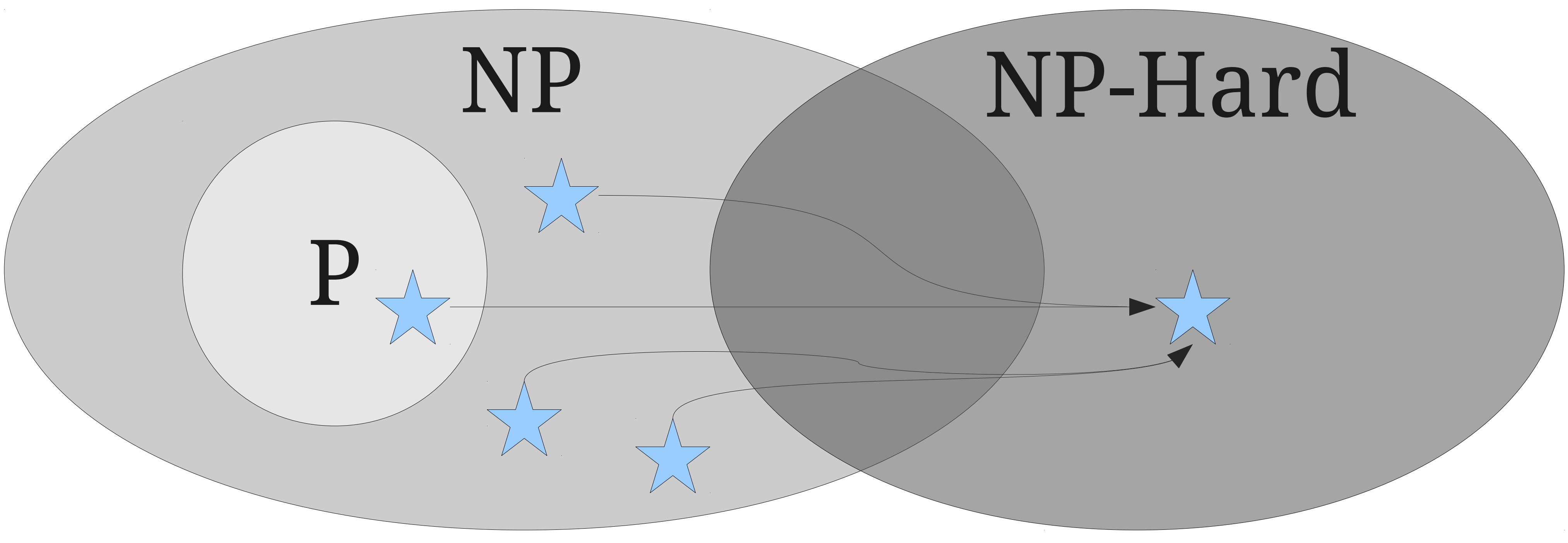
NP-Hardness

- A language L is called **NP-hard** iff for *every* $L' \in$ **NP**, we have $L' \leq_P L$.



NP-Hardness

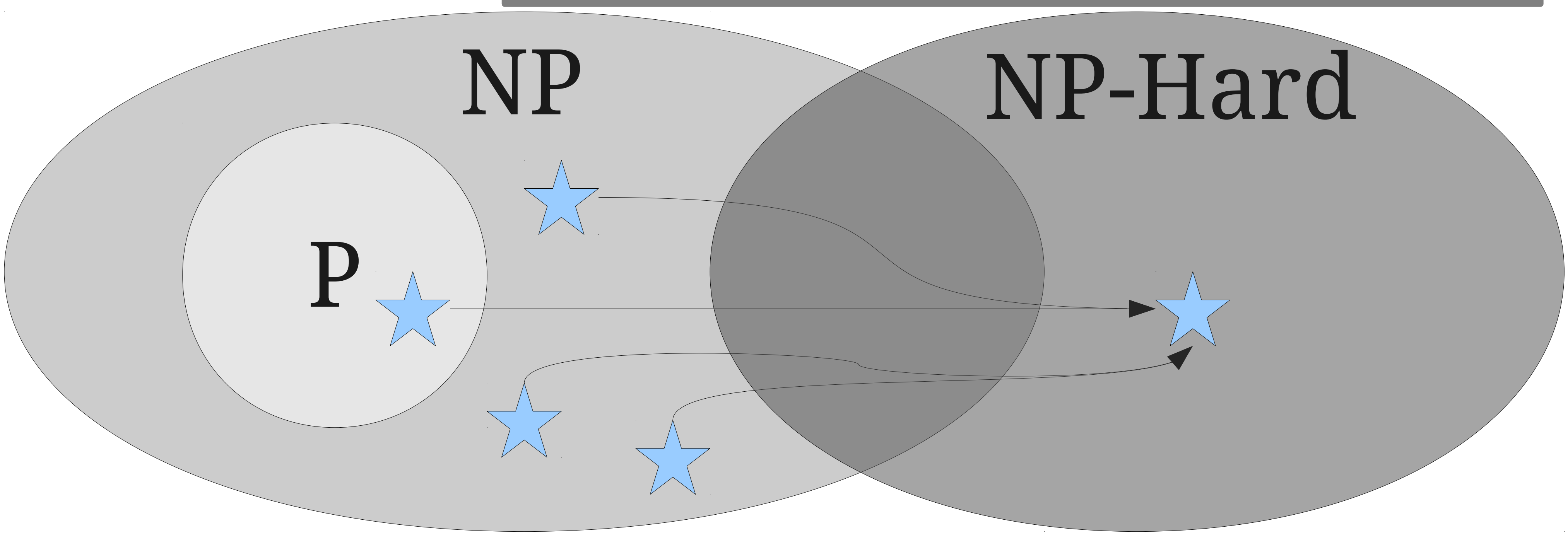
- A language L is called **NP-hard** iff for *every* $L' \in$ **NP**, we have $L' \leq_P L$.



NP-Hardness

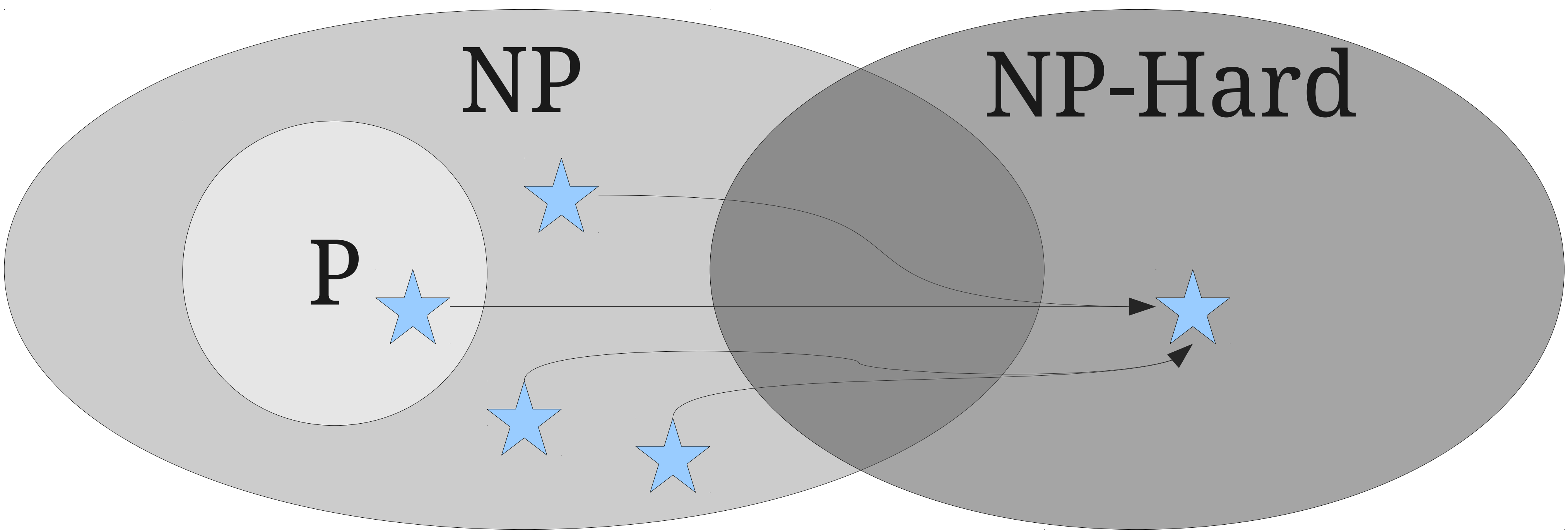
- A language L is called **NP-hard** iff for *every* $L' \in \mathbf{NP}$, we have $L' \leq_P L$.

Intuitively: L has to be at least as hard as every problem in **NP**, since an algorithm for L can be used to decide all problems in **NP**.



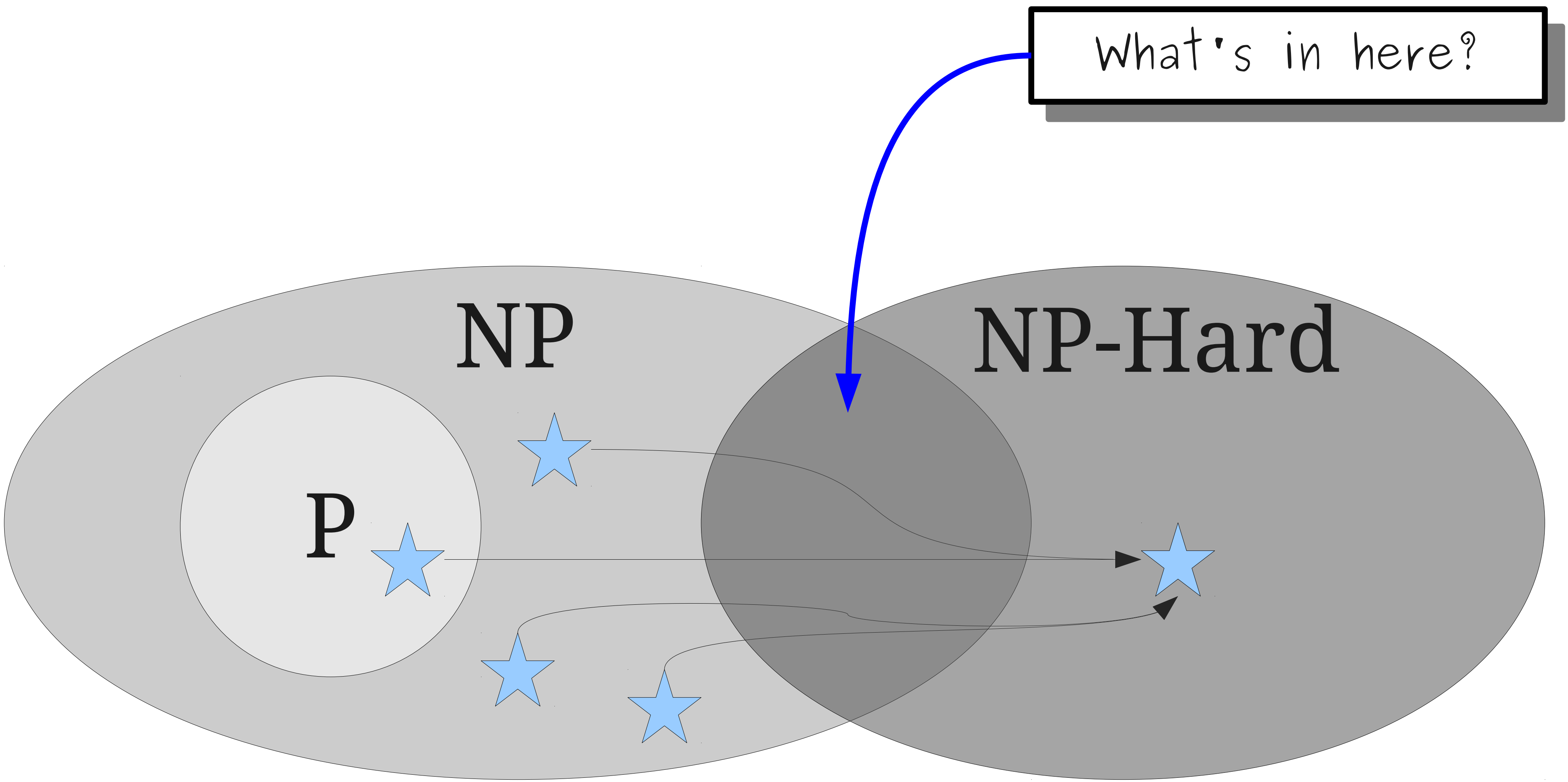
NP-Hardness

- A language L is called **NP-hard** iff for *every* $L' \in \mathbf{NP}$, we have $L' \leq_P L$.



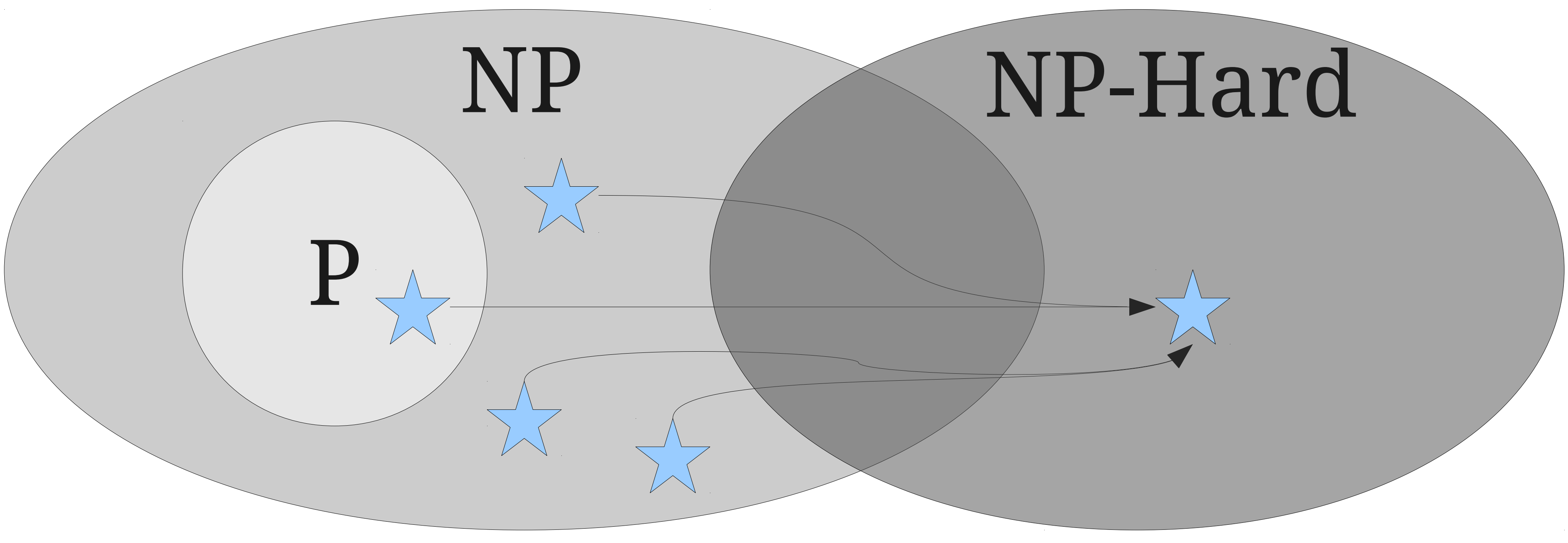
NP-Hardness

- A language L is called **NP-hard** iff for *every* $L' \in$ **NP**, we have $L' \leq_P L$.



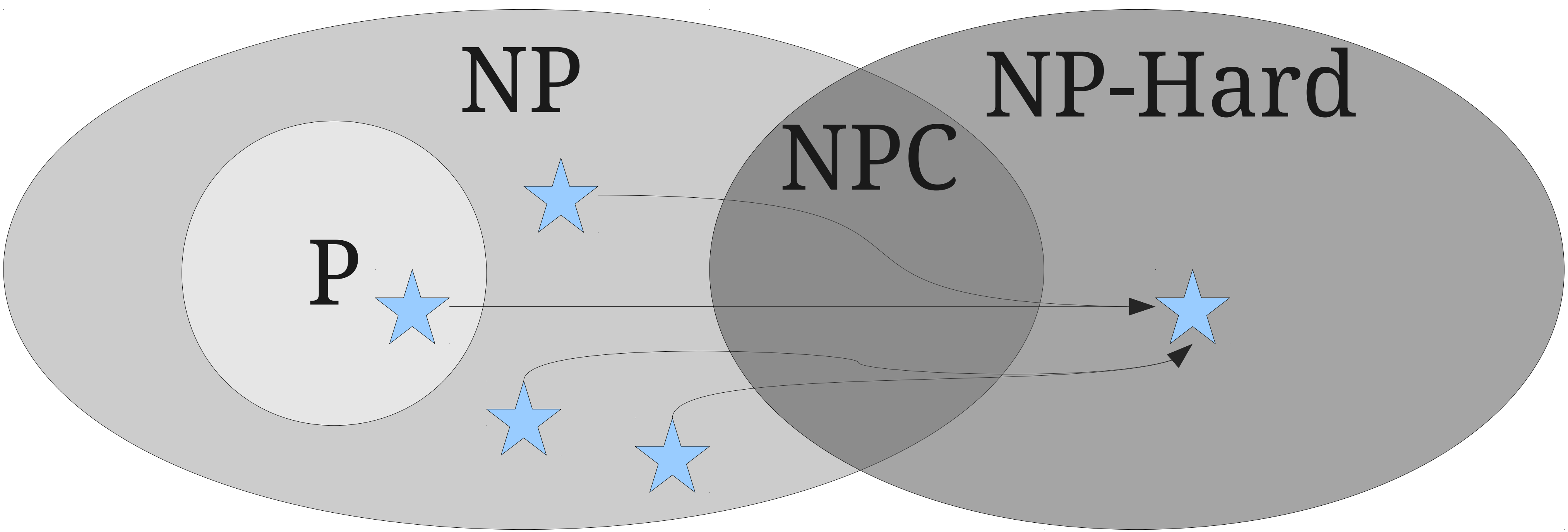
NP-Hardness

- A language L is called **NP-hard** iff for *every* $L' \in$ **NP**, we have $L' \leq_P L$.
- A language in L is called **NP-complete** iff L is **NP**-hard and $L \in$ **NP**.
- The class **NPC** is the set of **NP**-complete problems.



NP-Hardness

- A language L is called **NP-hard** iff for *every* $L' \in \mathbf{NP}$, we have $L' \leq_P L$.
- A language in L is called **NP-complete** iff L is **NP**-hard and $L \in \mathbf{NP}$.
- The class **NPC** is the set of **NP**-complete problems.



The Tantalizing Truth

Theorem: If ***any*** **NP**-complete language is in **P**, then **P** = **NP**.

The Tantalizing Truth

Theorem: If ***any*** **NP**-complete language is in **P**, then **P** = **NP**.

Proof: If $L \in$ **NPC** and $L \in$ **P**, we know for any $L' \in$ **NP** that $L' \leq_P L$, because L is **NP**-complete.

The Tantalizing Truth

Theorem: If ***any*** **NP**-complete language is in **P**, then **P** = **NP**.

Proof: If $L \in \mathbf{NPC}$ and $L \in \mathbf{P}$, we know for any $L' \in \mathbf{NP}$ that $L' \leq_P L$, because L is **NP**-complete. Since $L' \leq_P L$ and $L \in \mathbf{P}$, this means that $L' \in \mathbf{P}$ as well.

The Tantalizing Truth

Theorem: If ***any*** **NP**-complete language is in **P**, then **P** = **NP**.

Proof: If $L \in$ **NPC** and $L \in$ **P**, we know for any $L' \in$ **NP** that $L' \leq_P L$, because L is **NP**-complete. Since $L' \leq_P L$ and $L \in$ **P**, this means that $L' \in$ **P** as well. Since our choice of L' was arbitrary, any language $L' \in$ **NP** satisfies $L' \in$ **P**, so **NP** ⊆ **P**.

The Tantalizing Truth

Theorem: If ***any*** **NP**-complete language is in **P**, then **P** = **NP**.

Proof: If L ∈ **NPC** and L ∈ **P**, we know for any L' ∈ **NP** that $L' \leq_P L$, because L is **NP**-complete. Since $L' \leq_P L$ and L ∈ **P**, this means that L' ∈ **P** as well. Since our choice of L' was arbitrary, any language L' ∈ **NP** satisfies L' ∈ **P**, so **NP** ⊆ **P**. Since **P** ⊆ **NP**, this means **P** = **NP**.

The Tantalizing Truth

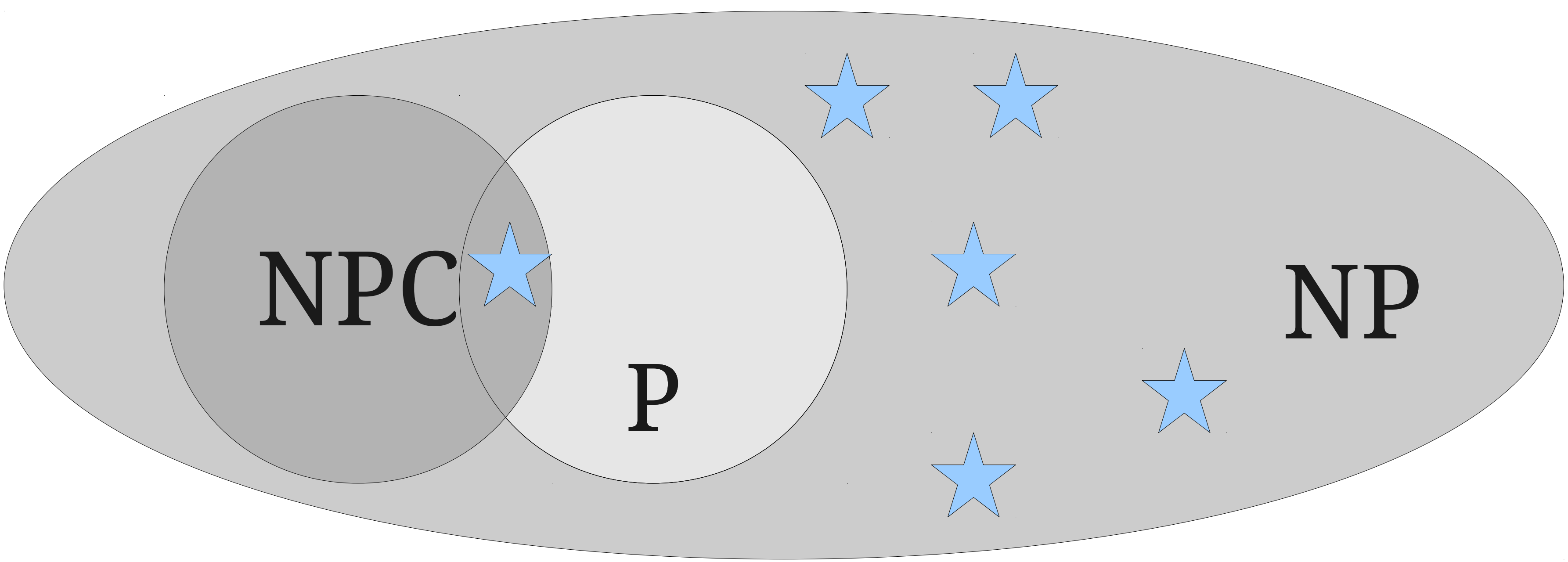
Theorem: If ***any*** **NP**-complete language is in **P**, then **P** = **NP**.

Proof: If L ∈ **NPC** and L ∈ **P**, we know for any L' ∈ **NP** that $L' \leq_P L$, because L is **NP**-complete. Since $L' \leq_P L$ and L ∈ **P**, this means that L' ∈ **P** as well. Since our choice of L' was arbitrary, any language L' ∈ **NP** satisfies L' ∈ **P**, so **NP** ⊆ **P**. Since **P** ⊆ **NP**, this means **P** = **NP**. ■

The Tantalizing Truth

Theorem: If ***any*** **NP**-complete language is in **P**, then **P** = **NP**.

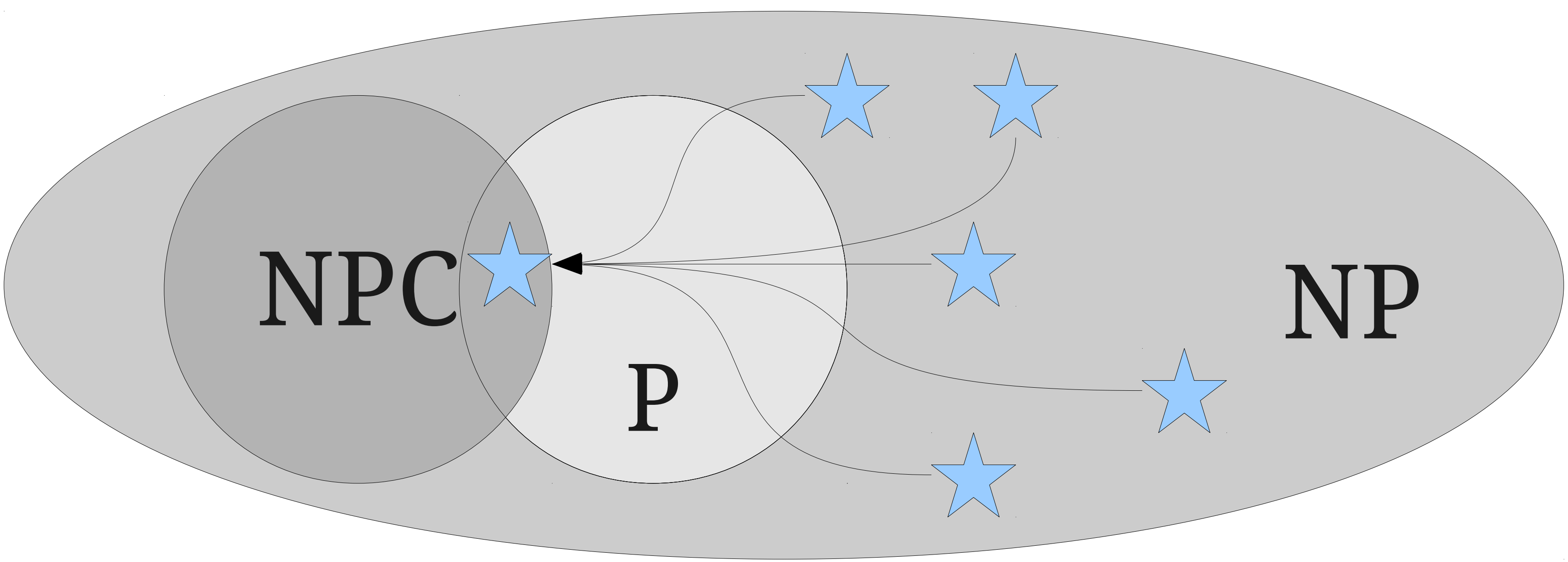
Proof: If $L \in$ **NPC** and $L \in$ **P**, we know for any $L' \in$ **NP** that $L' \leq_P L$, because L is **NP**-complete. Since $L' \leq_P L$ and $L \in$ **P**, this means that $L' \in$ **P** as well. Since our choice of L' was arbitrary, any language $L' \in$ **NP** satisfies $L' \in$ **P**, so **NP** $\subseteq$ **P**. Since **P** $\subseteq$ **NP**, this means **P** = **NP**. ■



The Tantalizing Truth

Theorem: If ***any*** **NP**-complete language is in **P**, then **P** = **NP**.

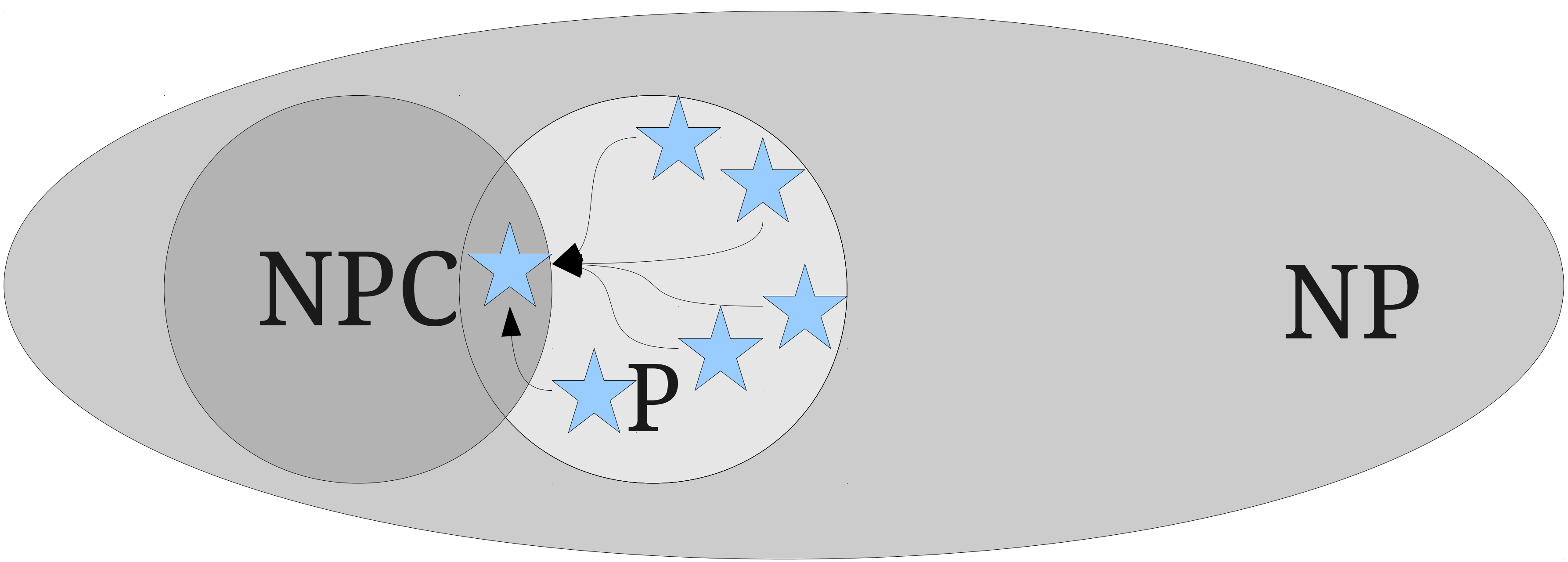
Proof: If $L \in \mathbf{NPC}$ and $L \in \mathbf{P}$, we know for any $L' \in \mathbf{NP}$ that $L' \leq_P L$, because L is **NP**-complete. Since $L' \leq_P L$ and $L \in \mathbf{P}$, this means that $L' \in \mathbf{P}$ as well. Since our choice of L' was arbitrary, any language $L' \in \mathbf{NP}$ satisfies $L' \in \mathbf{P}$, so $\mathbf{NP} \subseteq \mathbf{P}$. Since $\mathbf{P} \subseteq \mathbf{NP}$, this means $\mathbf{P} = \mathbf{NP}$. ■



The Tantalizing Truth

Theorem: If *any* **NP**-complete language is in **P**, then **P** = **NP**.

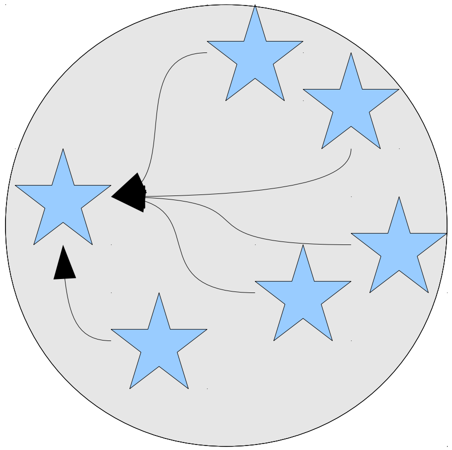
Proof: If $L \in$ **NPC** and $L \in$ **P**, we know for any $L' \in$ **NP** that $L' \leq_P L$, because L is **NP**-complete. Since $L' \leq_P L$ and $L \in$ **P**, this means that $L' \in$ **P** as well. Since our choice of L' was arbitrary, any language $L' \in$ **NP** satisfies $L' \in$ **P**, so **NP** ⊆ **P**. Since **P** ⊆ **NP**, this means **P** = **NP**. ■



The Tantalizing Truth

Theorem: If ***any*** **NP**-complete language is in **P**, then **P** = **NP**.

Proof: If $L \in$ **NPC** and $L \in$ **P**, we know for any $L' \in$ **NP** that $L' \leq_P L$, because L is **NP**-complete. Since $L' \leq_P L$ and $L \in$ **P**, this means that $L' \in$ **P** as well. Since our choice of L' was arbitrary, any language $L' \in$ **NP** satisfies $L' \in$ **P**, so **NP** $\subseteq$ **P**. Since **P** $\subseteq$ **NP**, this means **P** = **NP**. ■

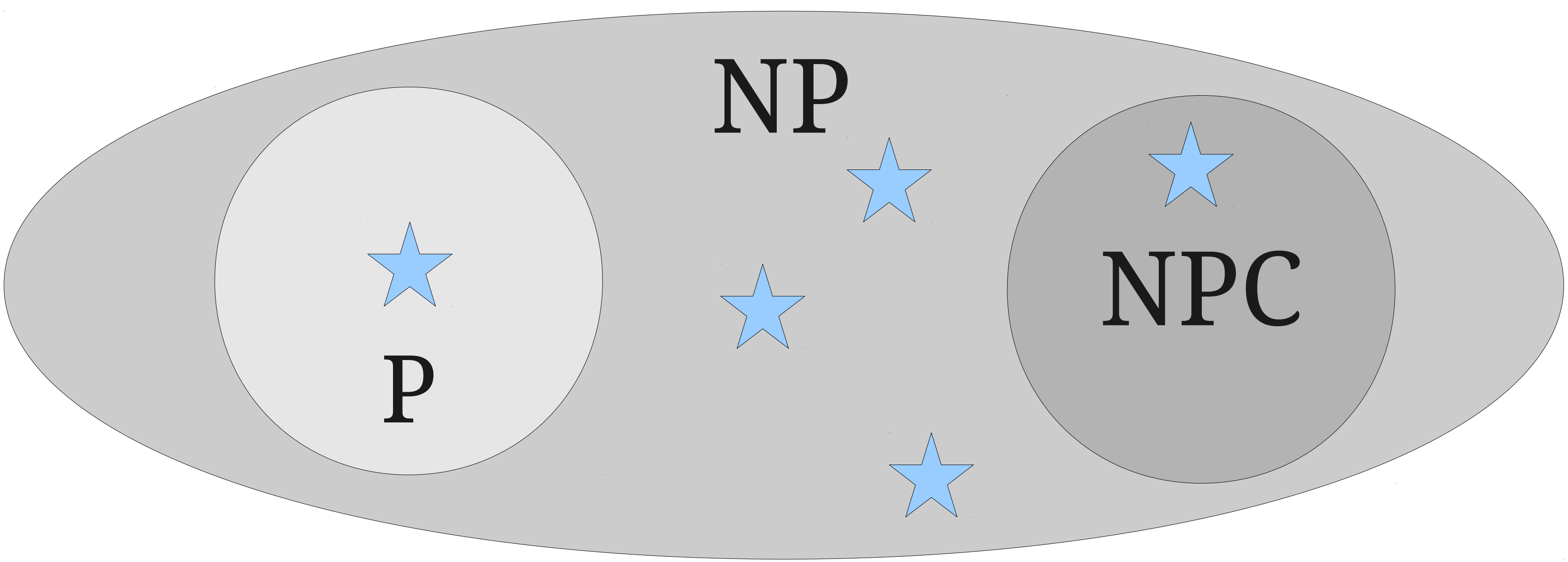


P = NP

The Tantalizing Truth

Theorem: If *any* **NP**-complete language is not in **P**, then **P ≠ NP**.

Proof: If $L \in$ **NPC**, then $L \in$ **NP**. Thus if $L \notin$ **P**, then $L \in$ **NP** – **P**.
This means that **NP** – **P** ≠ Ø, so **P** ≠ **NP**. ■



A Feel for **NP**-Completeness

- If a problem is **NP**-complete, then under the assumption that **P ≠ NP**, there cannot be an efficient algorithm for it.
- In a sense, **NP**-complete problems are the hardest problems in **NP**.
- All known **NP**-complete problems are enormously hard to solve:
  - All known algorithms for **NP**-complete problems run in worst-case exponential time.
  - Most algorithms for **NP**-complete problems are infeasible for reasonably-sized inputs.

How do we even know NP-complete problems exist in the first place?

Satisfiability

- A propositional logic formula φ is called **satisfiable** if there is some assignment to its variables that makes it evaluate to true.
  - $p \land q$ is satisfiable.
  - $p \land \neg p$ is unsatisfiable.
  - $p \to (q \land \neg q)$ is satisfiable.
- An assignment of true and false to the variables of φ that makes it evaluate to true is called a **satisfying assignment**.

SAT

- The **boolean satisfiability problem** (**SAT**) is the following:

Given a propositional logic formula φ, is φ satisfiable?

- Formally:

$$\textbf{SAT} = \{ \langle \varphi \rangle \mid \varphi \textbf{ is a satisfiable PL formula} \}$$

Theorem (Cook-Levin): SAT is **NP**-complete.

A Simpler **NP**-Complete Problem

Literals and Clauses

- A **literal** in propositional logic is a variable or its negation:
    - x
    - $\neg y$
    - But not $x \wedge y$.
- A **clause** is a many-way OR (*disjunction*) of literals.
    - $\neg x \vee y \vee \neg z$
    - x
    - But not $x \vee \neg(y \vee z)$

Conjunctive Normal Form

- A propositional logic formula φ is in **conjunctive normal form** (**CNF**) if it is the many-way AND (*conjunction*) of clauses.
  - $(x \vee y \vee z) \wedge (\neg x \vee \neg y) \wedge (x \vee y \vee z \vee \neg w)$
  - $x \vee z$
  - But not $(x \vee (y \wedge z)) \vee (x \vee y)$
- Only legal operators are ¬, ∨, ∧.
- No nesting allowed.

The Structure of CNF

$$(x \lor y \lor \neg z) \land (\neg x \lor \neg y \lor z) \land (\neg x \lor y \lor \neg z)$$

Each clause must have
at least one
true literal in it.

The Structure of CNF

$$(x \lor y \lor \neg z) \land (\neg x \lor \neg y \lor z) \land (\neg x \lor y \lor \neg z)$$

We should pick at least one true literal from each clause

The Structure of CNF

$$(x \lor y \lor \neg z) \land (\neg x \lor \neg y \lor z) \land (\neg x \lor y \lor \neg z)$$

… subject to the constraint that we never choose a literal and its negation

3-CNF

- A propositional formula is in **3-CNF** if
  - It is in CNF, and
  - Every clause has *exactly* three literals.
- For example:
  - $(x \lor y \lor z) \land (\neg x \lor \neg y \lor z)$
  - $(x \lor x \lor x) \land (y \lor \neg y \lor \neg x) \land (x \lor y \lor \neg y)$
  - But not $(x \lor y \lor z \lor w) \land (x \lor y)$
- The language **3SAT** is defined as follows:

3SAT = { ⟨φ⟩ | φ is a satisfiable 3-CNF formula }

Theorem: 3SAT is **NP**-Complete

Using the Cook-Levin Theorem

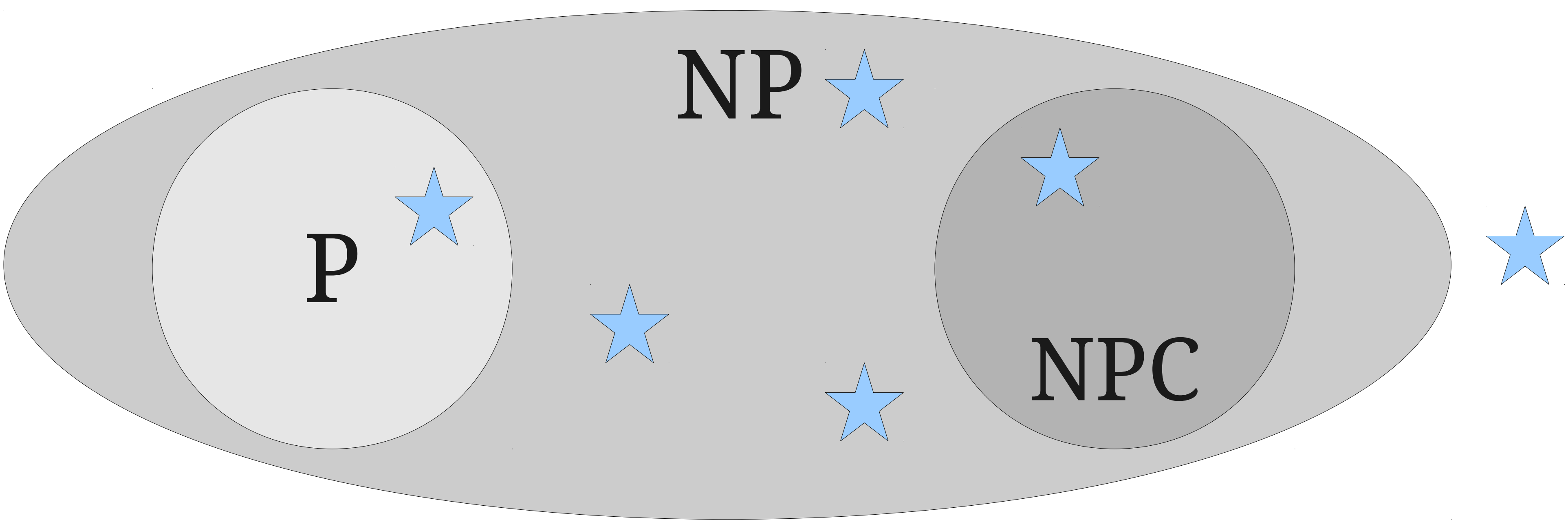
- When discussing decidability, we used the fact that $A_{TM} \notin \mathbf{R}$ as a starting point for finding other undecidable languages.
  - **Idea:** Reduce A_{TM} to some other language.
- When discussing **NP**-completeness, we will use the fact that 3SAT ∈ **NPC** as a starting point for finding other **NPC** languages.
  - **Idea**: Reduce 3SAT to some other language.

NP-Completeness

Theorem: Let L_1 and L_2 be languages. If $L_1 \leq_P L_2$ and L_1 is **NP**-hard, then L_2 is **NP**-hard.

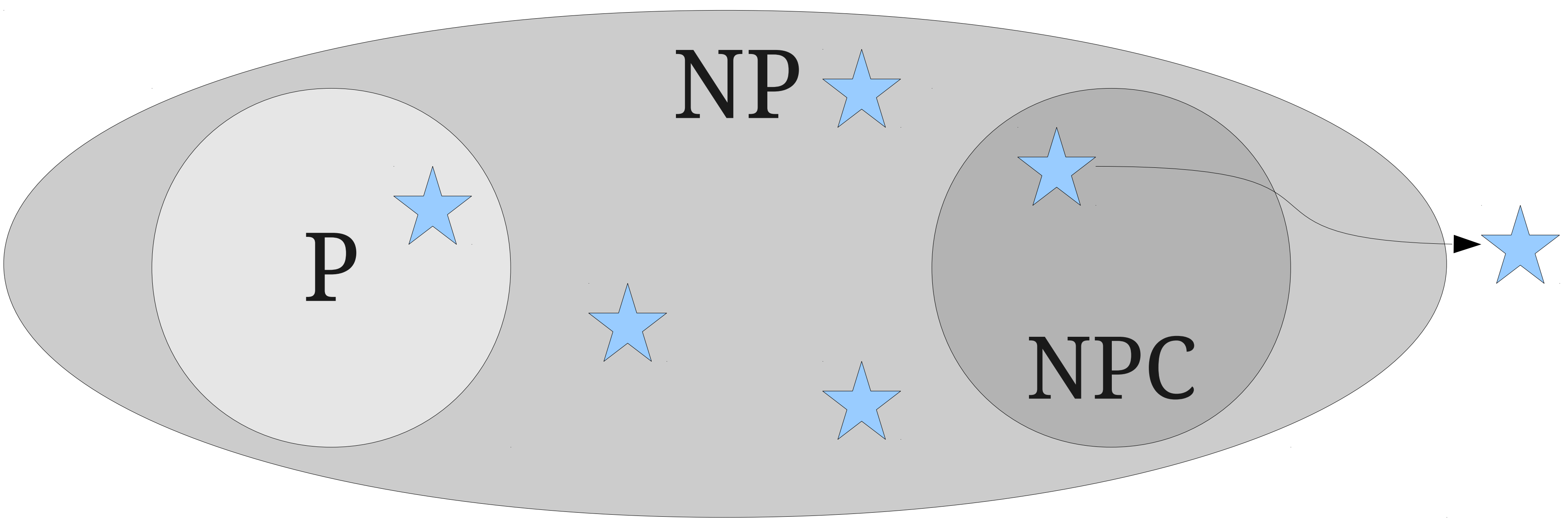
NP-Completeness

Theorem: Let L_1 and L_2 be languages. If $L_1 \leq_P L_2$ and L_1 is **NP**-hard, then L_2 is **NP**-hard.



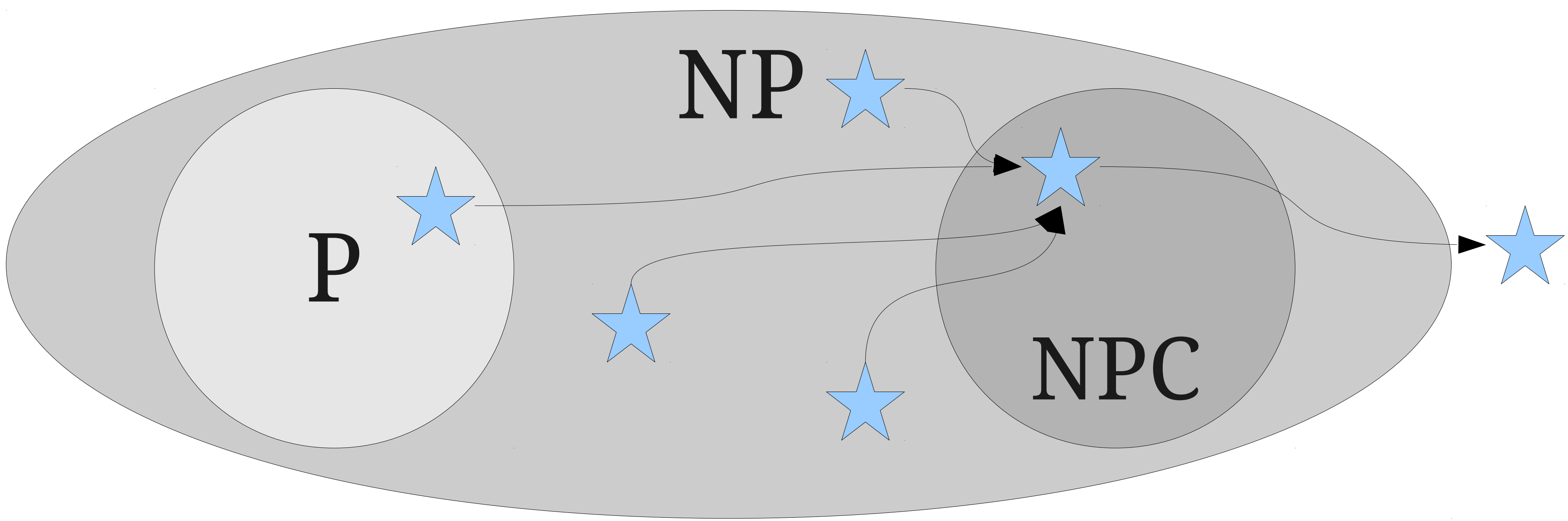
NP-Completeness

Theorem: Let L_1 and L_2 be languages. If $L_1 \leq_P L_2$ and L_1 is **NP**-hard, then L_2 is **NP**-hard.



NP-Completeness

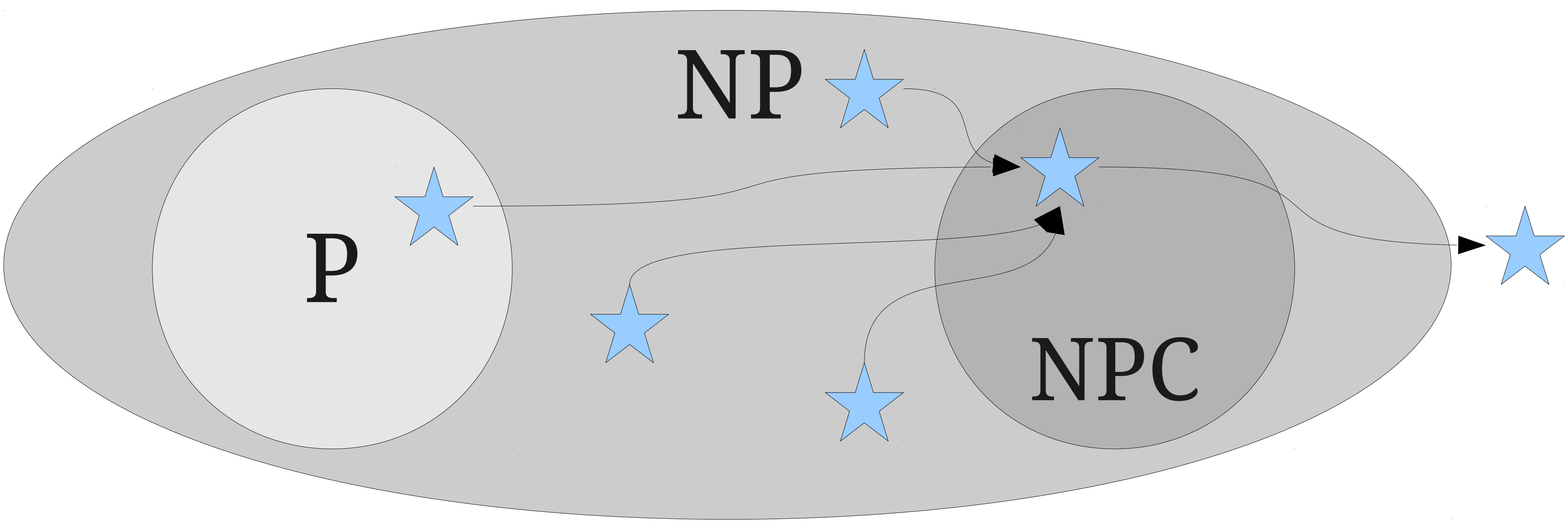
Theorem: Let L_1 and L_2 be languages. If $L_1 \leq_P L_2$ and L_1 is **NP**-hard, then L_2 is **NP**-hard.



NP-Completeness

Theorem: Let L_1 and L_2 be languages. If $L_1 \leq_P L_2$ and L_1 is **NP**-hard, then L_2 is **NP**-hard.

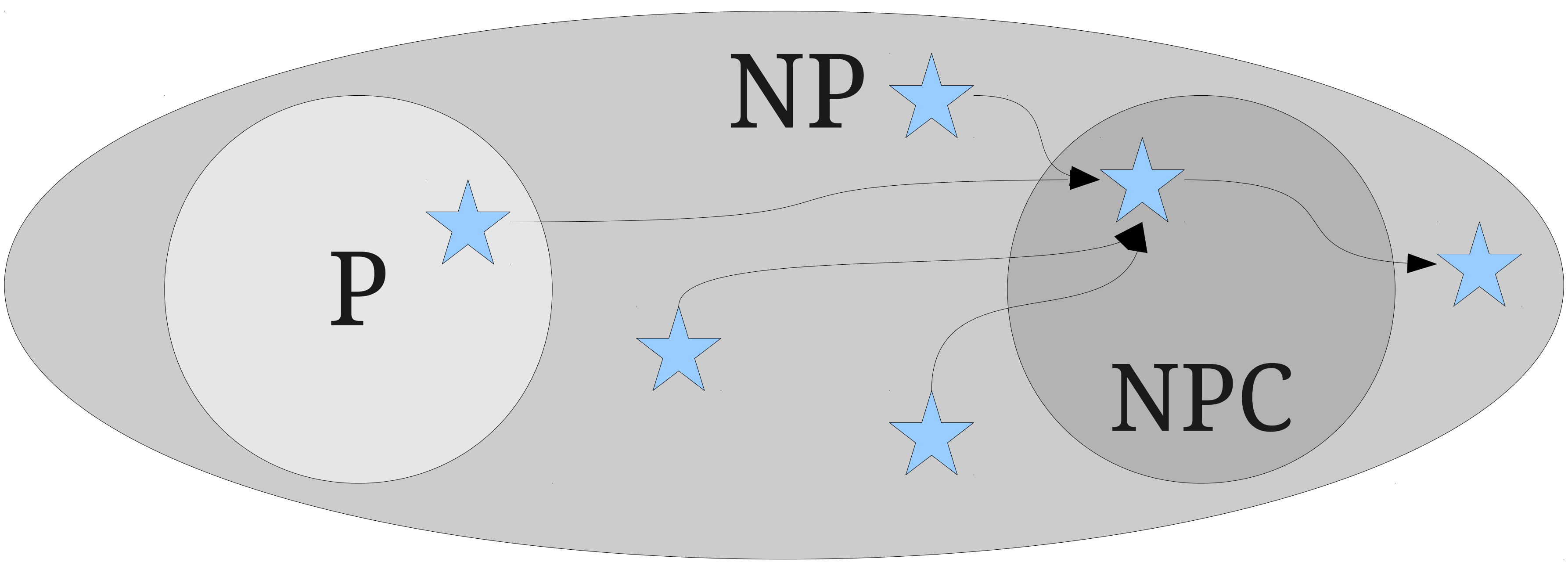
Theorem: Let L_1 and L_2 be languages where $L_1 \in$ **NPC** and $L_2 \in$ **NP**. If $L_1 \leq_P L_2$, then $L_2 \in$ **NPC**.



NP-Completeness

Theorem: Let L_1 and L_2 be languages. If $L_1 \leq_P L_2$ and L_1 is **NP**-hard, then L_2 is **NP**-hard.

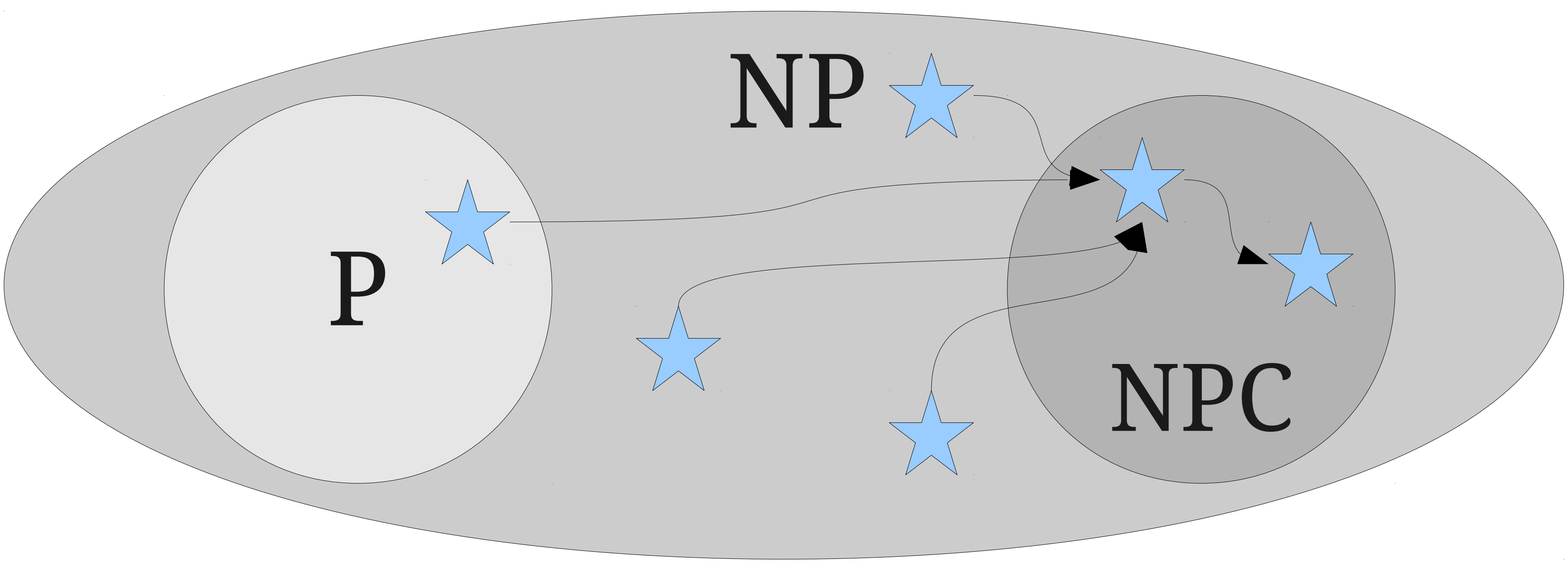
Theorem: Let L_1 and L_2 be languages where $L_1 \in$ **NPC** and $L_2 \in$ **NP**. If $L_1 \leq_P L_2$, then $L_2 \in$ **NPC**.



NP-Completeness

Theorem: Let L_1 and L_2 be languages. If $L_1 \leq_P L_2$ and L_1 is **NP**-hard, then L_2 is **NP**-hard.

Theorem: Let L_1 and L_2 be languages where $L_1 \in$ **NPC** and $L_2 \in$ **NP**. If $L_1 \leq_P L_2$, then $L_2 \in$ **NPC**.



Next Time

- **More NP-Complete Problems**
  - Independent Sets
  - Graph Coloring
- **Applied Complexity Theory (ITA)**
  - Why does all of this matter?