

# Complexity Theory

## Part Two

Recap from Last Time

# The Complexity Class **P**

- The complexity class **P** (*polynomial time*) is defined as

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

- Intuitively, **P** contains all decision problems that can be solved efficiently.
- This is like class **R**, except with “efficiently” tacked onto the end.

# The Complexity Class **NP**

- The complexity class **NP** (*nondeterministic polynomial time*) contains all problems that can be verified in polynomial time.
- Formally:
$$\mathbf{NP} = \{ L \mid \text{There is a polynomial-time verifier for } L \}$$
- Intuitively, **NP** is the set of problems where “yes” answers can be checked efficiently.
- This is like the class **RE**, but with “efficiently” tacked on to the definition.

The Biggest Unsolved Problem in  
Theoretical Computer Science:

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

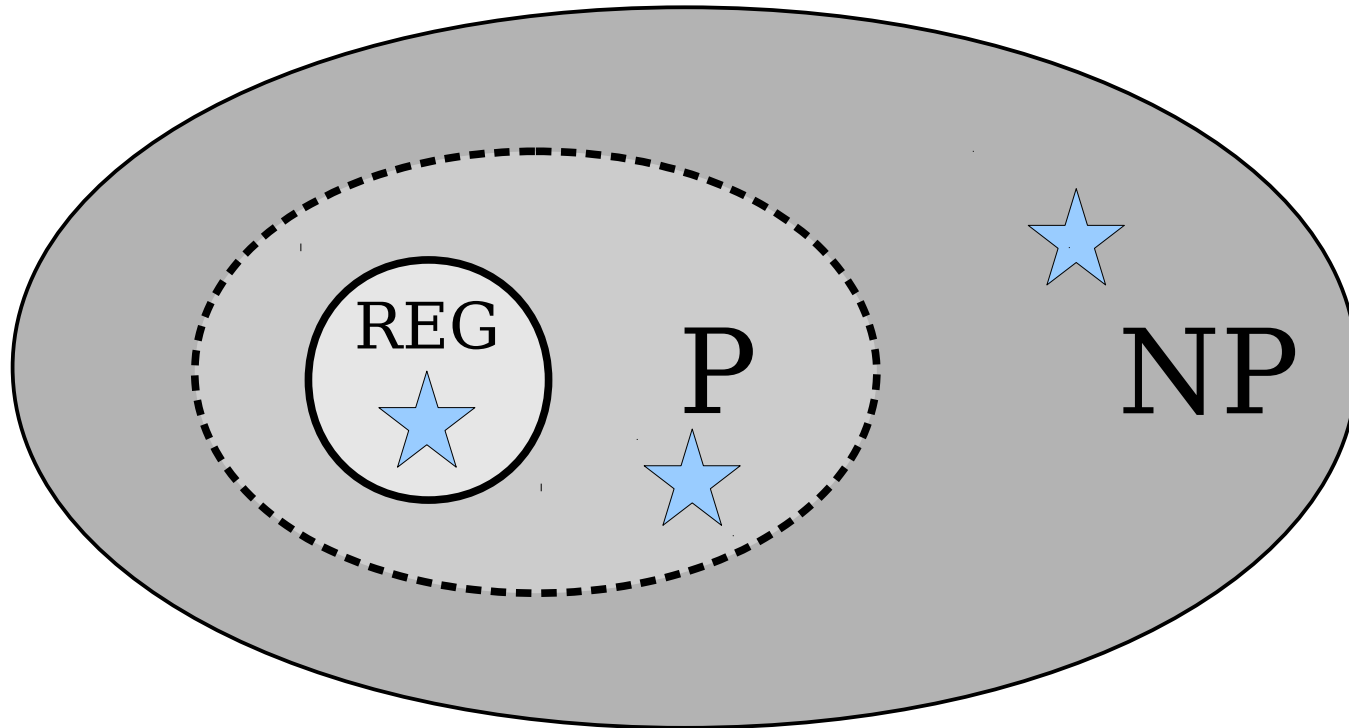
***Theorem (Baker-Gill-Solovay):*** Any proof that purely relies on universality and self-reference cannot resolve  $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ .

***Proof:*** Take CS154!

So how *are* we going to  
reason about **P** and **NP**?

New Stuff!

# A Challenge



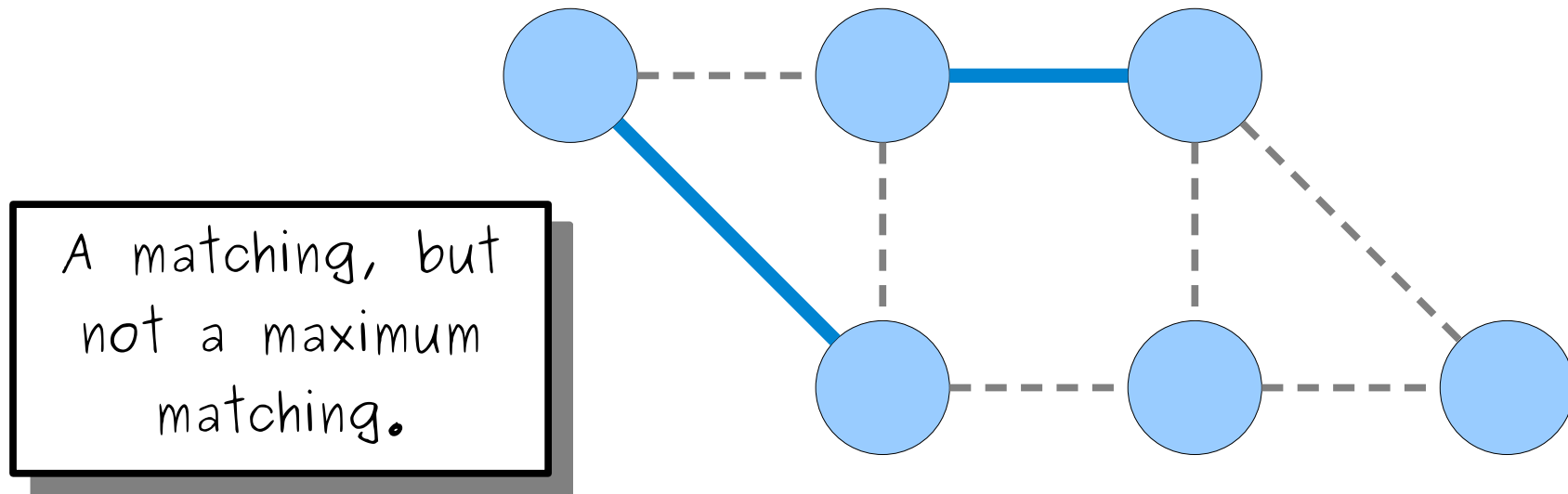
Problems in **NP** vary widely in their difficulty, even if **P** = **NP**.

How can we rank the relative difficulties of problems?

# Reducibility

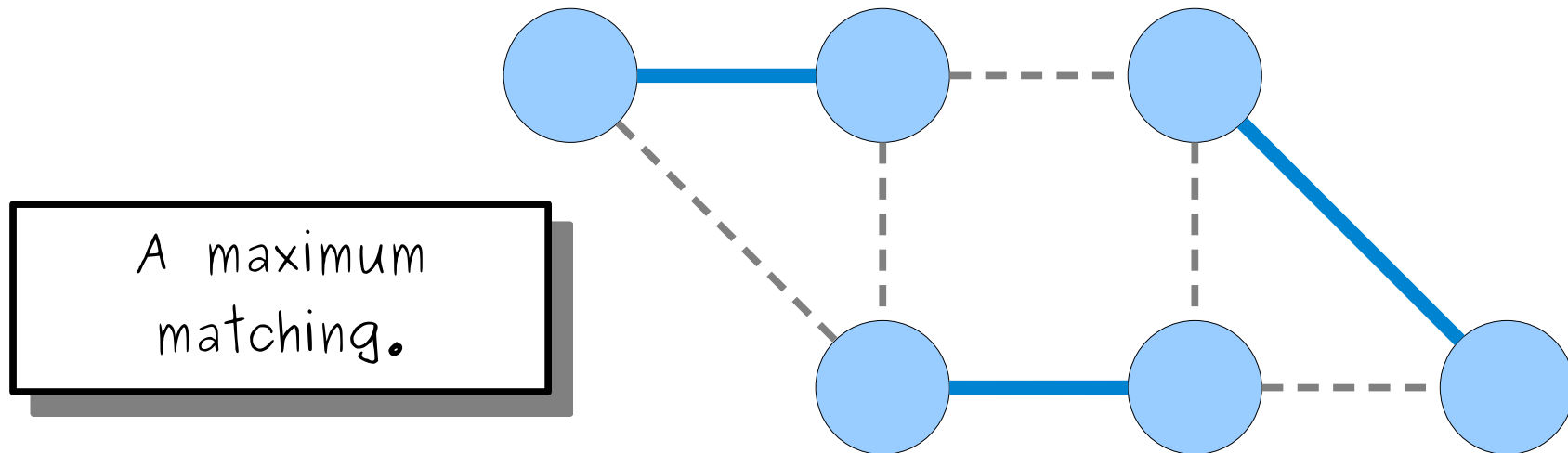
# Maximum Matching

- Given an undirected graph  $G$ , a **matching** in  $G$  is a set of edges such that no two edges share an endpoint.
- A **maximum matching** is a matching with the largest number of edges.



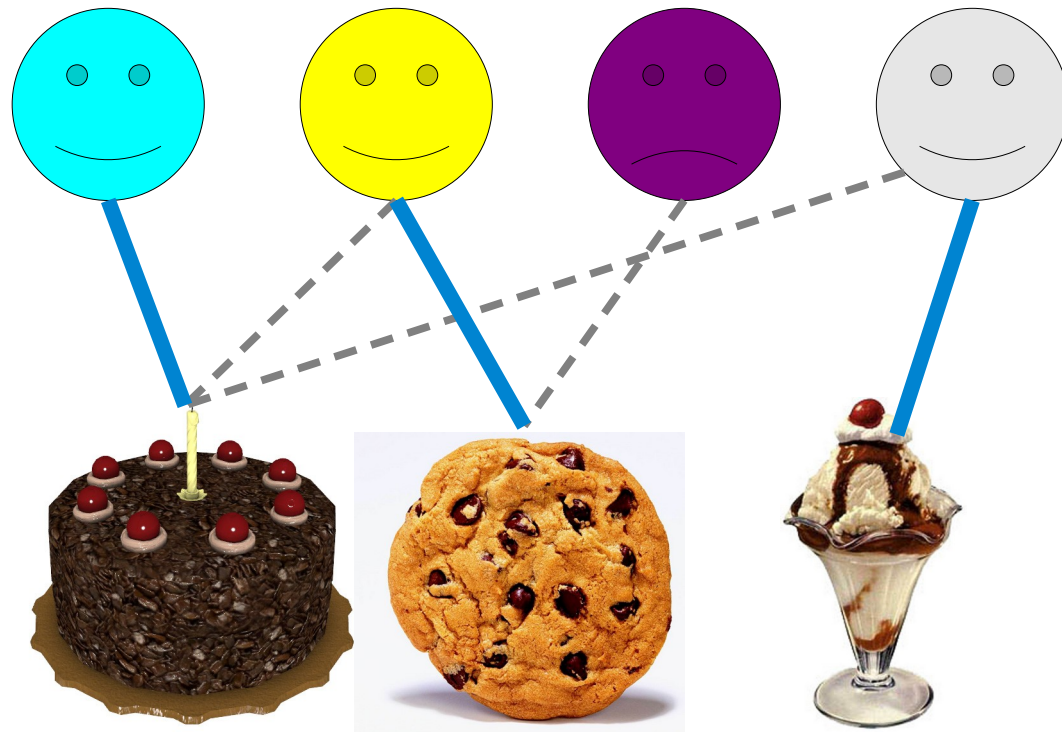
# Maximum Matching

- Given an undirected graph  $G$ , a **matching** in  $G$  is a set of edges such that no two edges share an endpoint.
- A **maximum matching** is a matching with the largest number of edges.



# Maximum Matching

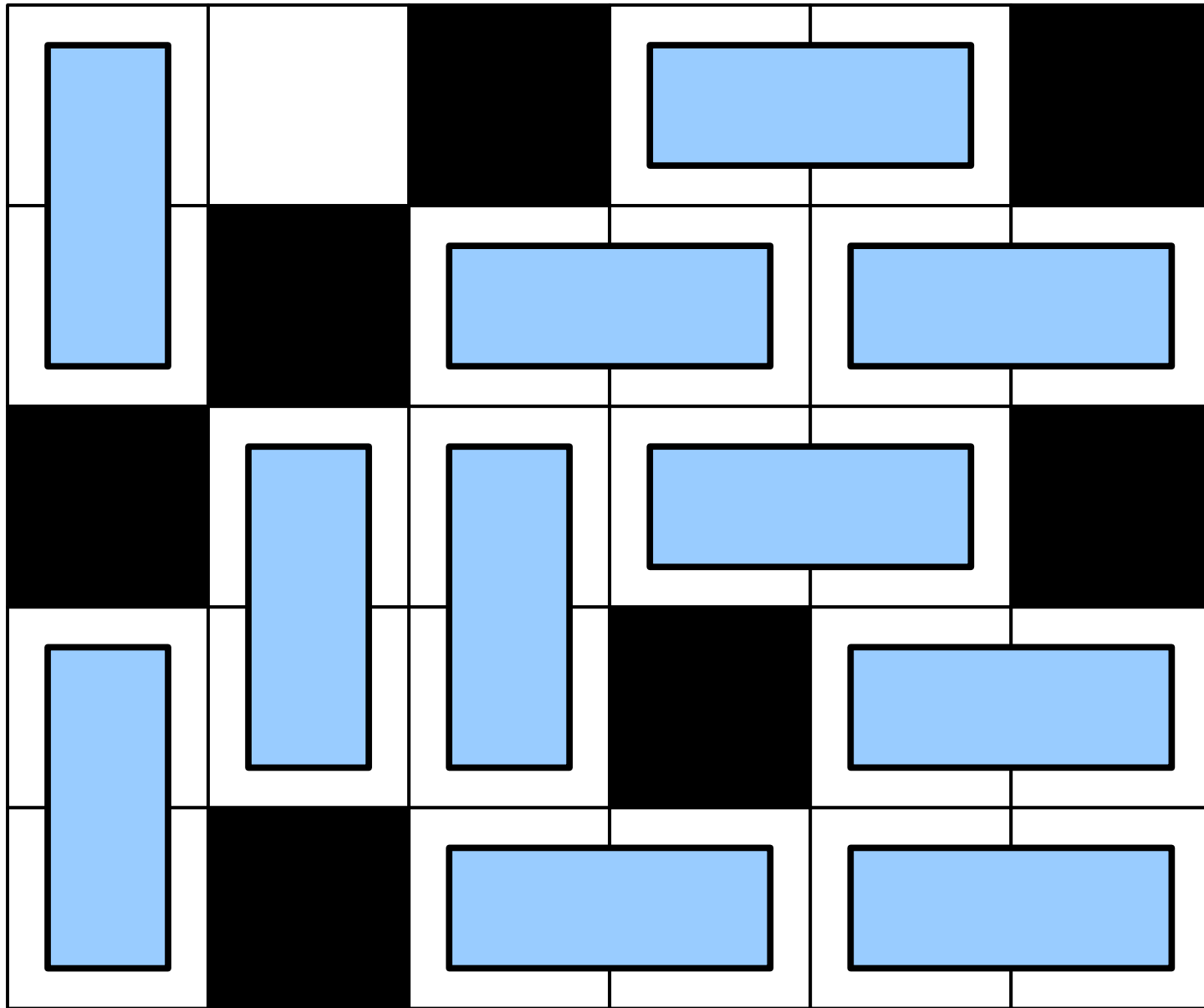
- Given an undirected graph  $G$ , a **matching** in  $G$  is a set of edges such that no two edges share an endpoint.
- A **maximum matching** is a matching with the largest number of edges.



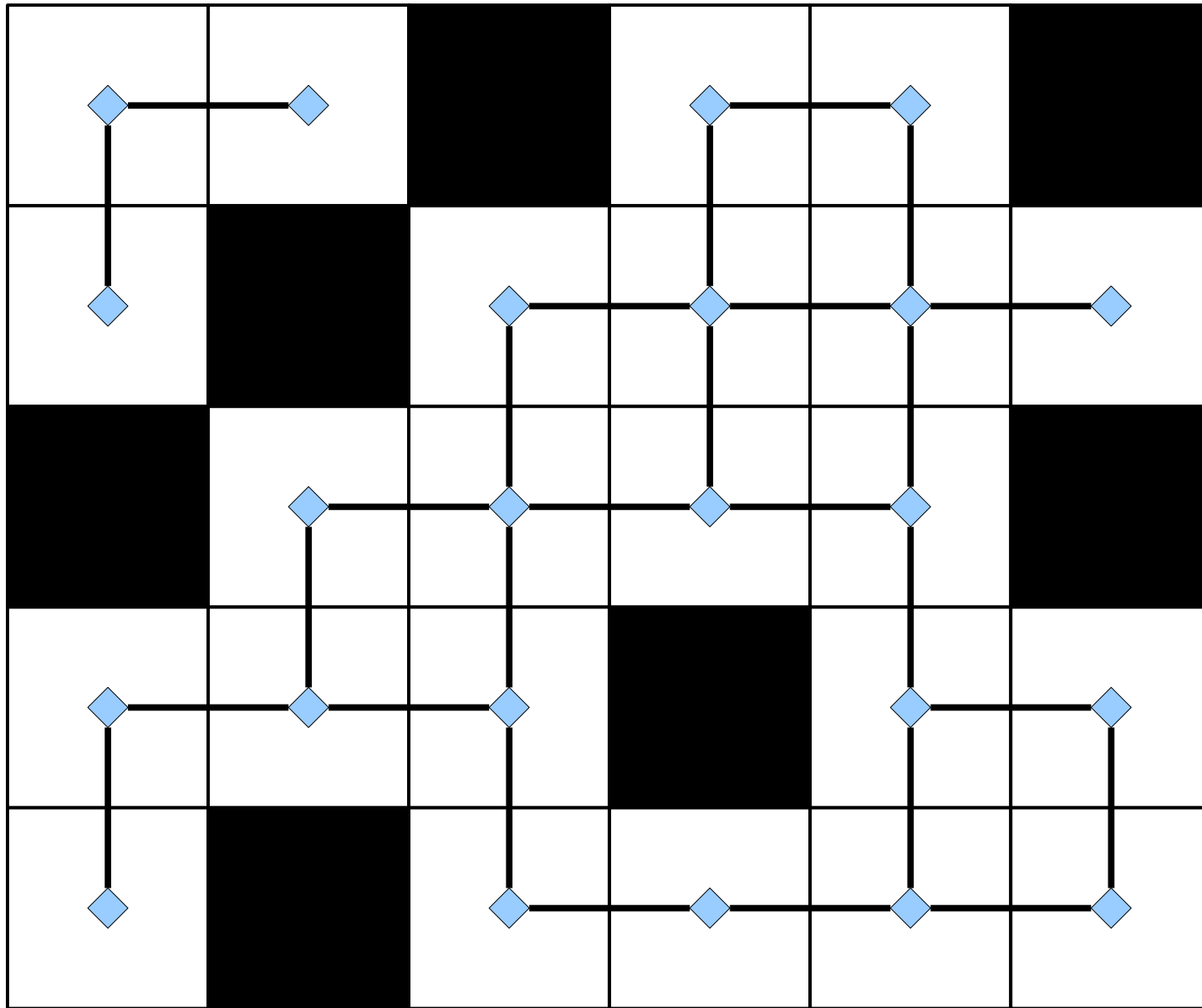
# Maximum Matching

- Jack Edmonds' paper “Paths, Trees, and Flowers” gives a polynomial-time algorithm for finding maximum matchings.
  - He's the guy from last time with the quote about “better than decidable.”
- Using this fact, what other problems can we solve?

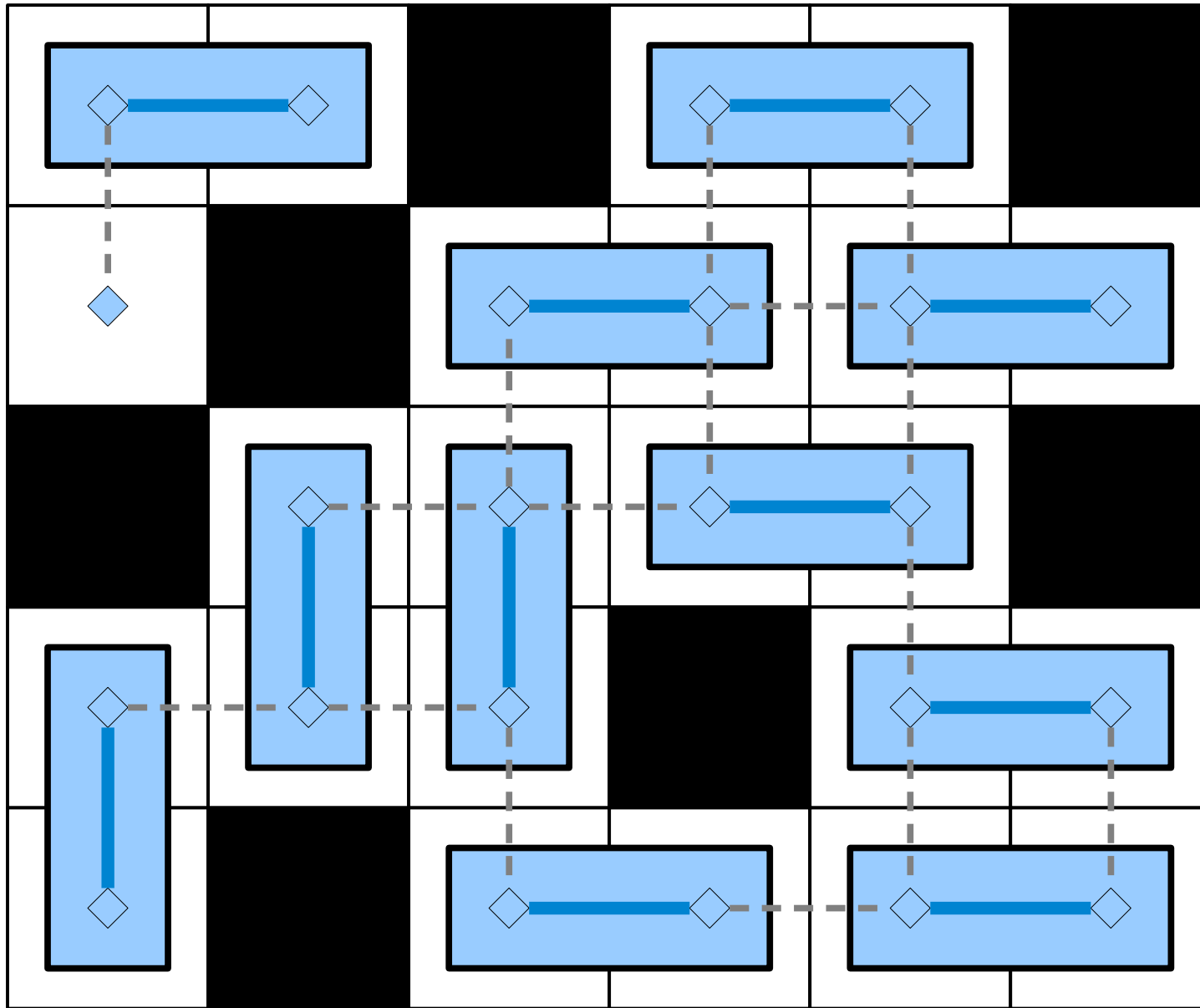
# Domino Tiling



# Solving Domino Tiling



# Solving Domino Tiling



```
bool canPlaceDominoes(Grid G, int k) {  
    return hasMatching(gridToGraph(G), k);  
}
```

Which of the following is the most reasonable conclusion to draw, given the existence of the above function?

- A. Solving domino tiling can't be "harder" than solving maximum matching.
- B. Solving maximum matching can't be "harder" than solving domino tiling.
- C. Both A and B.

Answer at

<https://pollev.com/cs103aut23>

## ***Intuition:***

Tiling a grid with dominoes can't be “harder” than solving maximum matching, because if we can solve maximum matching efficiently, we can solve domino tiling efficiently.

Another Example

# Reachability

- Consider the following problem:  
**Given an directed graph  $G$  and nodes  $s$  and  $t$  in  $G$ , is there a path from  $s$  to  $t$ ?**
- This problem can be solved in polynomial time (use DFS or BFS).

# Converter Conundrums

- Suppose that you want to plug your laptop into a projector.
- Your laptop only has a VGA output, but the projector needs HDMI input.
- You have a box of connectors that convert various types of input into various types of output (for example, VGA to DVI, DVI to DisplayPort, etc.)
- **Question:** Can you plug your laptop into the projector?

# Converter Conundrums

## Connectors

RGB to USB

VGA to DisplayPort

DB13W3 to CATV

DisplayPort to RGB

DB13W3 to HDMI

DVI to DB13W3

S-Video to DVI

FireWire to SDI

VGA to RGB

DVI to DisplayPort

USB to S-Video

SDI to HDMI

Can I plug my VGA output laptop into an HDMI input projector using these connectors?

Answer at

<https://pollev.com/cs103aut23>



# In Pseudocode

```
bool canPlugIn(vector<Plug> plugs) {  
    return isReachable(plugsToGraph(plugs),  
                        VGA, HDMI);  
}
```

## ***Intuition:***

Finding a way to plug a computer into a projector can't be “harder” than determining reachability in a graph, since if we can determine reachability in a graph, we can find a way to plug a computer into a projector.

```
bool solveProblemA(string input) {  
    return solveProblemB(translate(input));  
}
```

### ***Intuition:***

Problem *A* can't be “harder” than problem *B*, because solving problem *B* lets us solve problem *A*.

```
bool solveProblemA(string input) {  
    return solveProblemB(translate(input));  
}
```

- If  $A$  and  $B$  are problems where it's possible to solve problem  $A$  using the strategy shown above\*, we write

$$A \leq_p B.$$

- We say that ***A is polynomial-time reducible to B.***

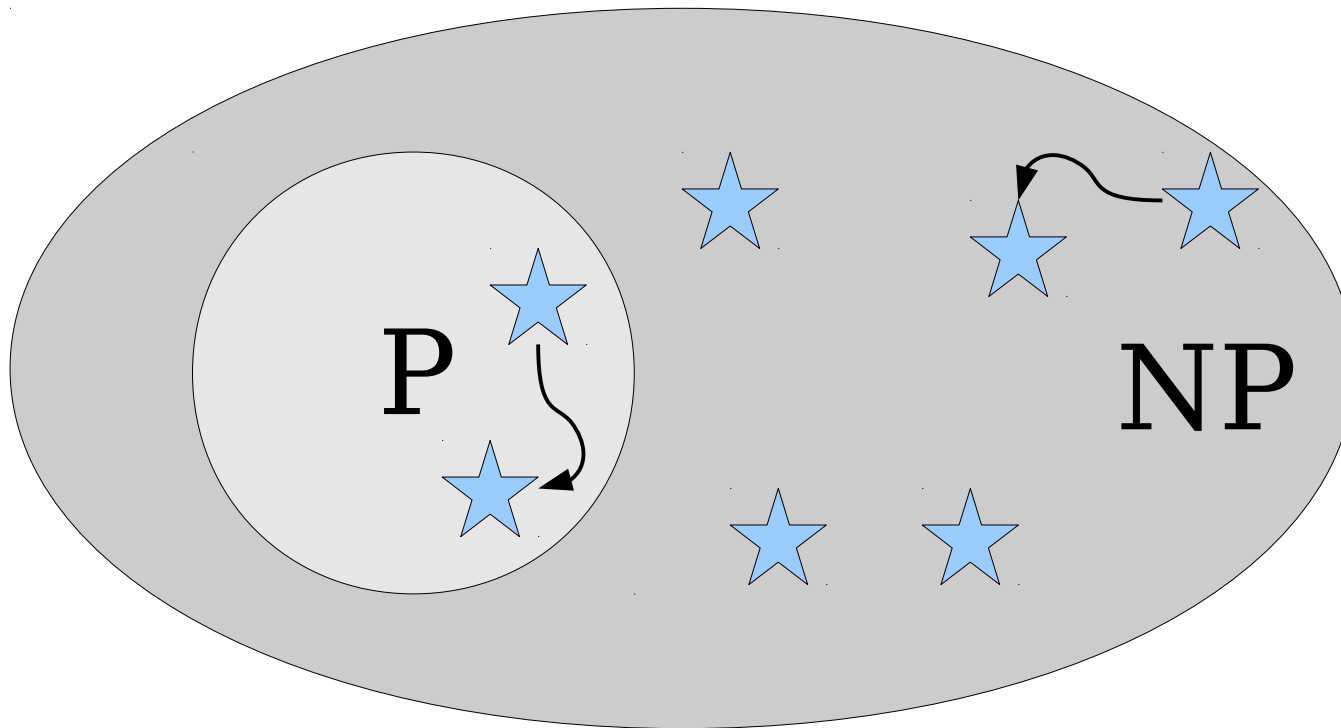
\* Assuming that translate runs in polynomial time.

```
bool solveProblemA(string input) {  
    return solveProblemB(translate(input));  
}
```

- This is a powerful general problem-solving technique. You'll see it a lot in CS161.

# Polynomial-Time Reductions

- If  $A \leq_p B$  and  $B \in \mathbf{P}$ , then  $A \in \mathbf{P}$ .
- If  $A \leq_p B$  and  $B \in \mathbf{NP}$ , then  $A \in \mathbf{NP}$ .



This  $\leq_p$  relation lets us rank the relative difficulties of problems in **P** and **NP**.

What else can we do with it?

**Time-Out for Announcements!**

***Please evaluate this course on Axess.***

Your feedback makes a difference.

# Final Exam Logistics

- Our final exam is on ***Wednesday, December 13<sup>th</sup>*** from ***3:30PM - 6:30PM*** in ***Hewlett 200***.
- The final exam is cumulative and covers topics from PS1 - PS9 and L00 - L27. The format is similar to that of the midterm, with a mix of short-answer questions and formal written proofs.
- Like the midterms, it's closed-book, closed-computer, and limited-note. You can bring one double-sided 8.5" × 11" notes sheet with you.
- Students with OAE accommodations: contact us ASAP if you haven't heard from us with alternate exam logistics.

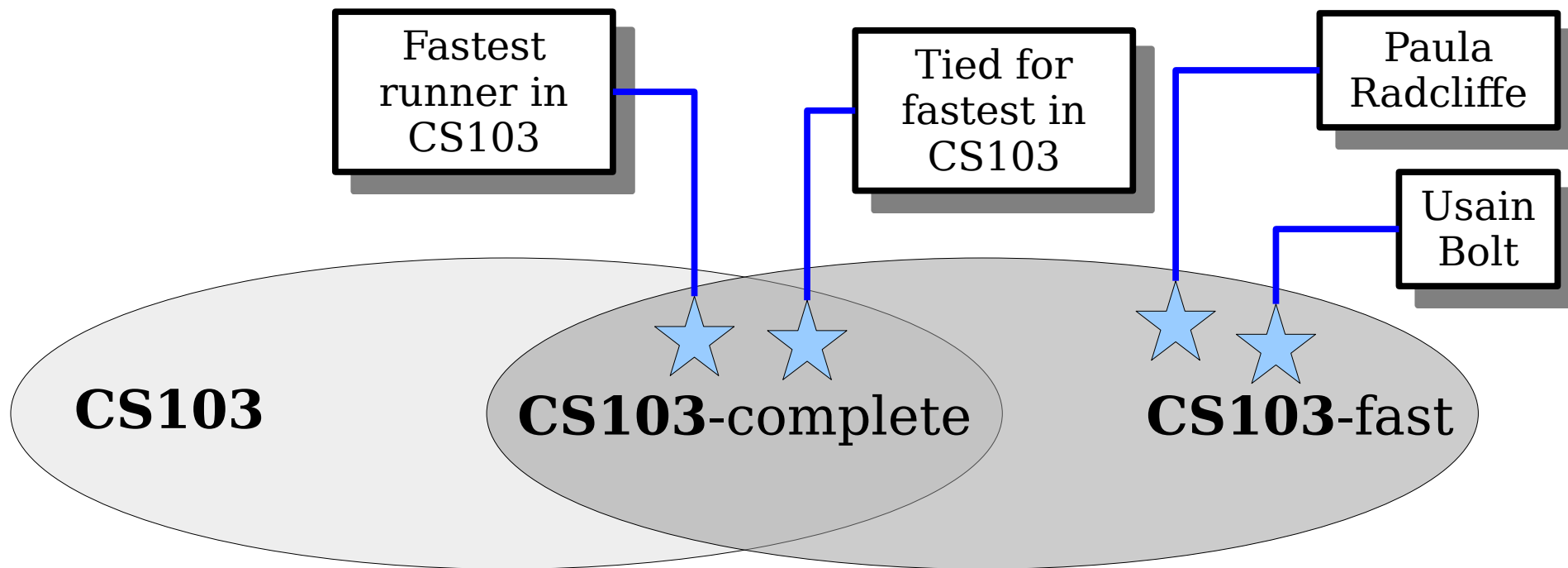
# Preparing for the Exam

- Benson will be holding a final exam review session this ***Friday, December 8<sup>th</sup>*** from ***11:30AM - 12:20PM*** in ***Thornton 102***.
  - It will be recorded and the video made available on Canvas.
- We also have a giant compendium of practice problems on the course website you can use to brush up on specific topics.
- As always, ***keep the TAs in the loop when studying!*** That's what we're here for.

Back to CS103!

# **NP**-Hardness and **NP**-Completeness

***An Analogy:*** Running Really Fast



For people  $A$  and  $B$ , we say  $A \leq_r B$  if  $A$ 's top running speed is at most  $B$ 's top speed.  
*(Intuitively:  $B$  can run at least as fast as  $A$ .)*

We say that person  $P$  is **CS103-fast** if

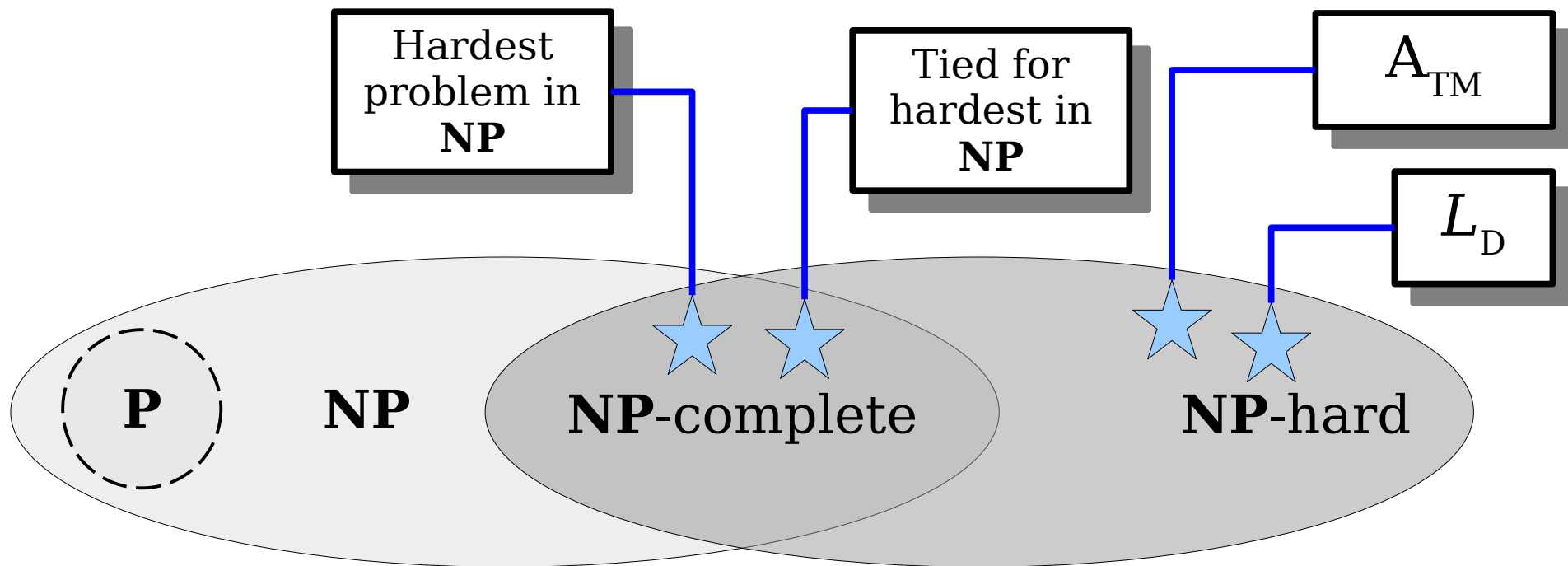
$$\forall A \in \mathbf{CS103}. A \leq_r P.$$

*(How fast are you if you're CS103-fast?)*

We say that person  $P$  is **CS103-complete** if

$$P \in \mathbf{CS103} \text{ and } P \text{ is } \mathbf{CS103-fast}.$$

*(How fast are you if you're CS103-complete?)*



For languages  $A$  and  $B$ , we say  $A \leq_p B$  if  $A$  reduces to  $B$  in polynomial time.

*(Intuitively:  $B$  is at least as hard as  $A$ .)*

We say that a language  $L$  is ***NP-hard*** if

$$\forall A \in \mathbf{NP}. A \leq_p L.$$

*(How hard is a problem that's NP-hard?)*

We say that a language  $L$  is ***NP-complete*** if

$$L \in \mathbf{NP} \text{ and } L \text{ is } \mathbf{NP}\text{-hard}.$$

*(How hard is a problem that's NP-complete?)*

***Intuition:*** The **NP**-complete problems are the hardest problems in **NP**.

If we can determine how hard those problems are, it would tell us a lot about the **P**  $\stackrel{?}{=}$  **NP** question.

# The Tantalizing Truth

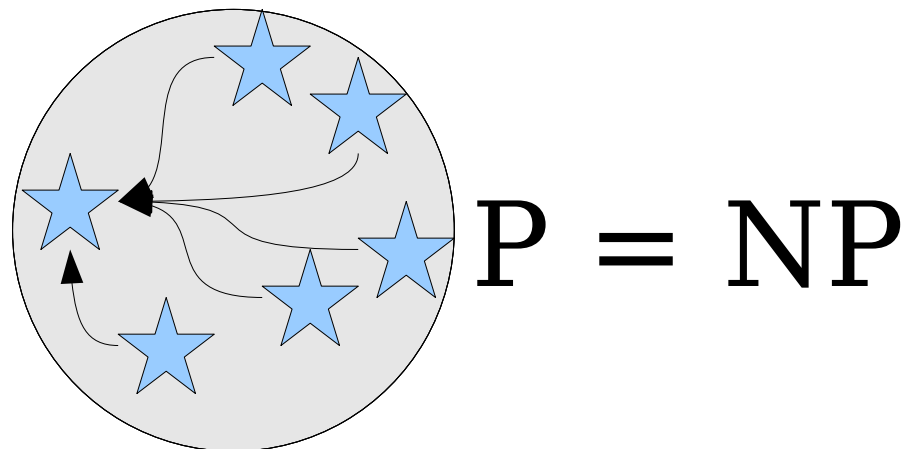
**Theorem:** If *any* **NP**-complete language is in **P**, then **P** = **NP**.

**Intuition:** This means the hardest problems in **NP** aren't actually that hard. We can solve them in polynomial time. So that means we can solve all problems in **NP** in polynomial time.

# The Tantalizing Truth

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

**Proof:** Suppose that  $L$  is **NP**-complete and  $L \in \mathbf{P}$ . Now consider any arbitrary **NP** problem  $A$ . Since  $L$  is **NP**-complete, we know that  $A \leq_p L$ . Since  $L \in \mathbf{P}$  and  $A \leq_p L$ , we see that  $A \in \mathbf{P}$ . Since our choice of  $A$  was arbitrary, this means that  $\mathbf{NP} \subseteq \mathbf{P}$ , so **P** = **NP**. ■



# The Tantalizing Truth

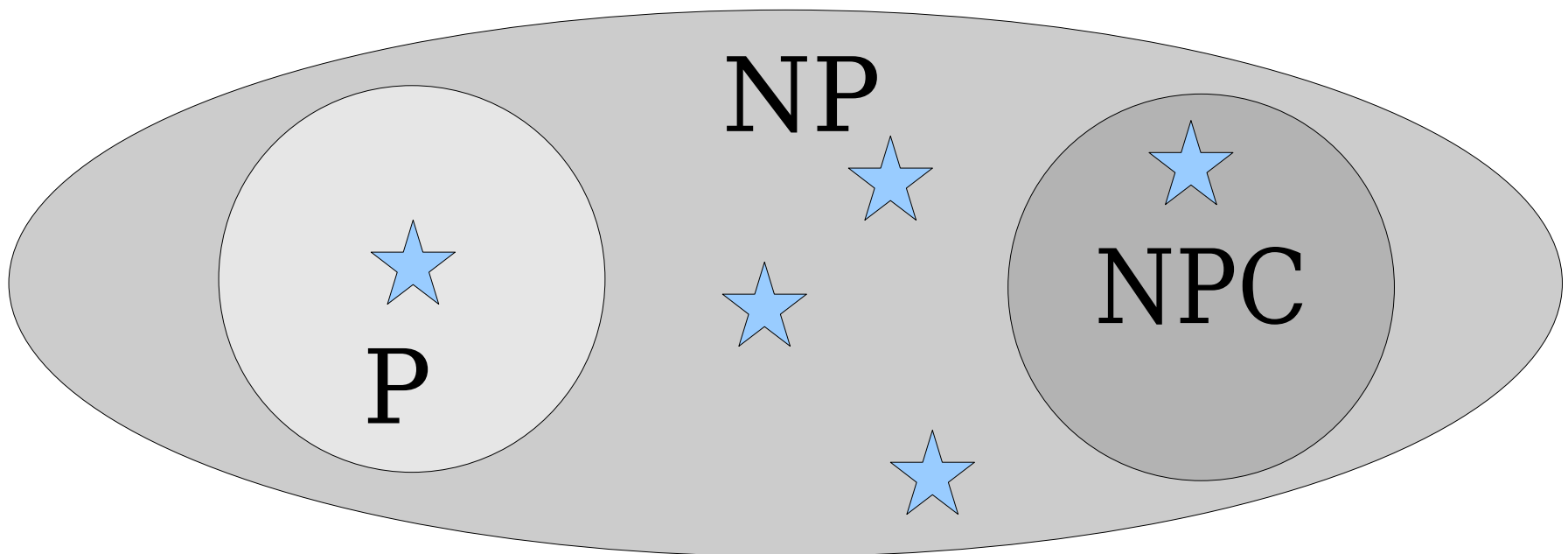
**Theorem:** If *any* **NP**-complete language is not in **P**, then **P**  $\neq$  **NP**.

**Intuition:** This means the hardest problems in **NP** are so hard that they can't be solved in polynomial time. So the hardest problems in **NP** aren't in **P**, meaning **P**  $\neq$  **NP**.

# The Tantalizing Truth

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

**Proof:** Suppose that  $L$  is an **NP**-complete language not in **P**. Since  $L$  is **NP**-complete, we know that  $L \in \mathbf{NP}$ . Therefore, we know that  $L \in \mathbf{NP}$  and  $L \notin \mathbf{P}$ , so **P**  $\neq$  **NP**. ■



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

# Satisfiability

- A propositional logic formula  $\varphi$  is called **satisfiable** if there is some assignment to its variables that makes it evaluate to true.
- Which of the following formulas are satisfiable?

$$p \wedge q$$

$$p \wedge \neg p$$

$$p \rightarrow (q \wedge \neg q)$$

Answer at

<https://pollev.com/cs103aut23>

# Satisfiability

- A propositional logic formula  $\varphi$  is called **satisfiable** if there is some assignment to its variables that makes it evaluate to true.
- Which of the following formulas are satisfiable?

$$p \wedge q$$

$$p \wedge \neg p$$

$$p \rightarrow (q \wedge \neg q)$$

- An assignment of true and false to the variables of  $\varphi$  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  $\varphi$ , is  $\varphi$  satisfiable?**

- Formally:

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

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

***Proof Idea:*** To see that **SAT**  $\in$  **NP**, show how to make a polynomial-time verifier for it. Key idea: have the certificate be a satisfying assignment.

To show that **SAT** is **NP**-hard, given a polynomial-time verifier  $V$  for an arbitrary **NP** language  $L$ , for any string  $w$  you can construct a polynomially-sized formula  $\varphi(w)$  that says “there is a certificate  $c$  where  $V$  accepts  $\langle w, c \rangle$ .” This formula is satisfiable if and only if  $w \in L$ , so deciding whether the formula is satisfiable decides whether  $w$  is in  $L$ . ■-ish

***Proof:*** Take CS154!

# Why All This Matters

- Resolving  $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$  is equivalent to just figuring out how hard SAT is.

$$\mathbf{SAT} \in \mathbf{P} \quad \leftrightarrow \quad \mathbf{P} = \mathbf{NP}$$

- We've turned a huge, abstract, theoretical problem about solving problems versus checking solutions into the concrete task of seeing how hard one problem is.
- You can get a sense for how little we know about algorithms and computation given that we can't yet answer this question!

# Why All This Matters

- You will almost certainly encounter **NP**-hard problems in practice – they're everywhere!
- If a problem is **NP**-hard, then there is no known algorithm for that problem that
  - is efficient on all inputs,
  - always gives back the right answer, and
  - runs deterministically.
- ***Useful intuition:*** If you need to solve an **NP**-hard problem, you will either need to settle for an approximate answer, an answer that's likely but not necessarily right, or have to work on really small inputs.

# Sample NP-Hard Problems

- **Computational biology:** Given a set of genomes, what is the most probable evolutionary tree that would give rise to those genomes? (*Maximum parsimony problem*)
- **Game theory:** Given an arbitrary perfect-information, finite, two-player game, who wins? (*Generalized geography problem*)
- **Operations research:** Given a set of jobs and workers who can perform those tasks in parallel, can you complete all the jobs within some time bound? (*Job scheduling problem*)
- **Machine learning:** Given a set of data, find the simplest way of modeling the statistical patterns in that data (*Bayesian network inference problem*)
- **Medicine:** Given a group of people who need kidneys and a group of kidney donors, find the maximum number of people who can receive transplants. (*Cycle cover problem*)
- **Systems:** Given a set of processes and a number of processors, find the optimal way to assign those tasks so that they complete as soon as possible (*Processor scheduling problem*)

***Coda:*** What if  $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$  is resolved?

# Next Time

- ***Why All This Matters***
- ***Where to Go from Here***
- ***A Final “Your Questions”***
- ***Parting Words!***