CS103, Autumn 2026



Lecture 26:

Complexity Theory

Part 2 of 2

Recap from Last Time

The Complexity Class **P**

- The complexity class **P** (***polynomial time***) contains all problems that can be decided ("solved") in polynomial time.
- Formally:

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

- 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:

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

Adapting Our Techniques

- We already know **R** ≠ **RE**. So does that mean **P** ≠ **NP**?
- To reason about what's in **R** and what's in **RE**, we used two key techniques:
  - ***Universality***: TMs can simulate other TMs.
  - ***Self-Reference***: TMs can get their own source code.
- Why can't we just do that for **P** and **NP**?

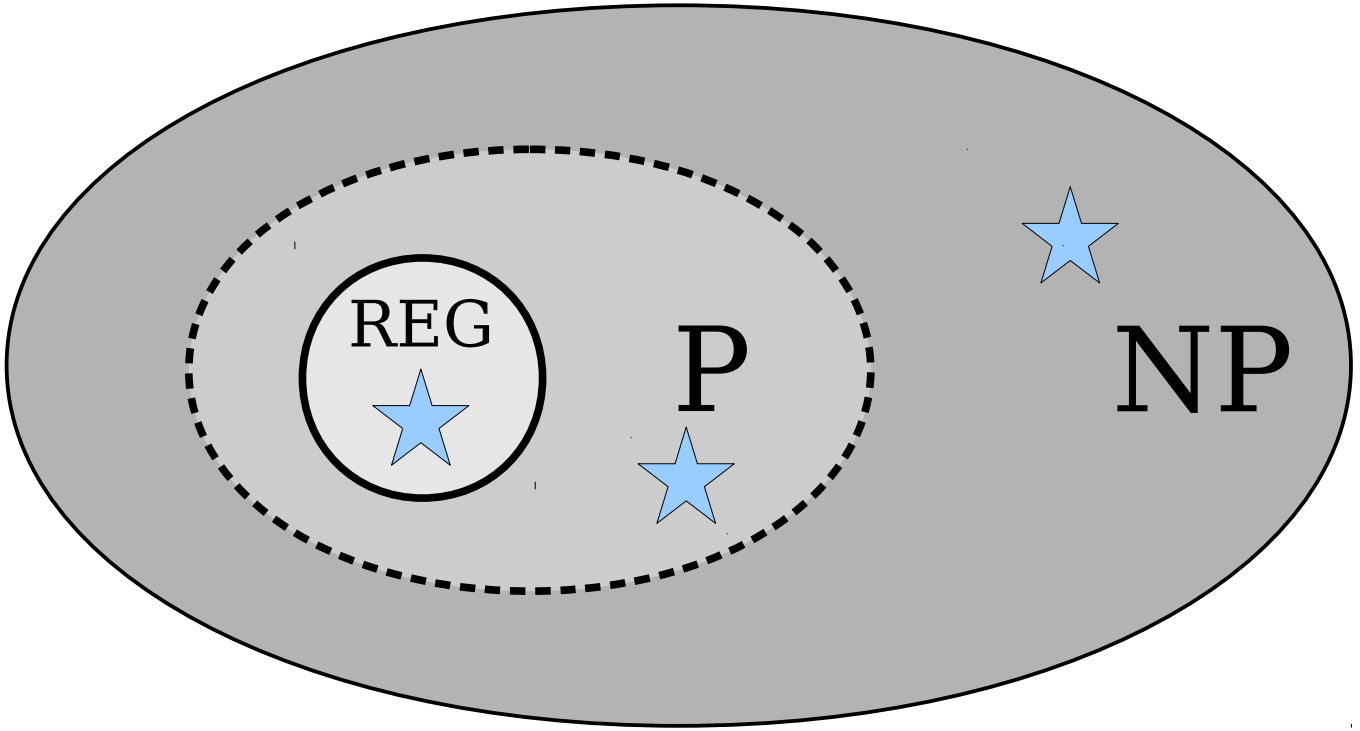
Theorem (Baker-Gill-Solovay): Any proof that purely relies on universality and self-reference cannot resolve $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$.

Proof: Take CS154!

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

New Stuff!

An Initial Observation



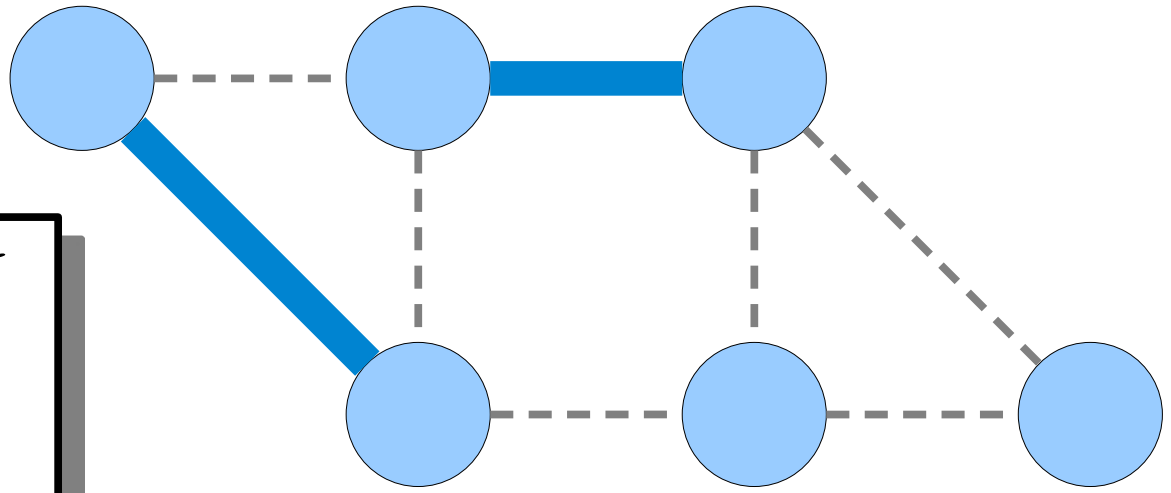
Problems in **NP** vary widely in their difficulty, even if **P** = **NP**.

How can we rank the relative difficulties of problems?

Reducibility

Maximum Matching

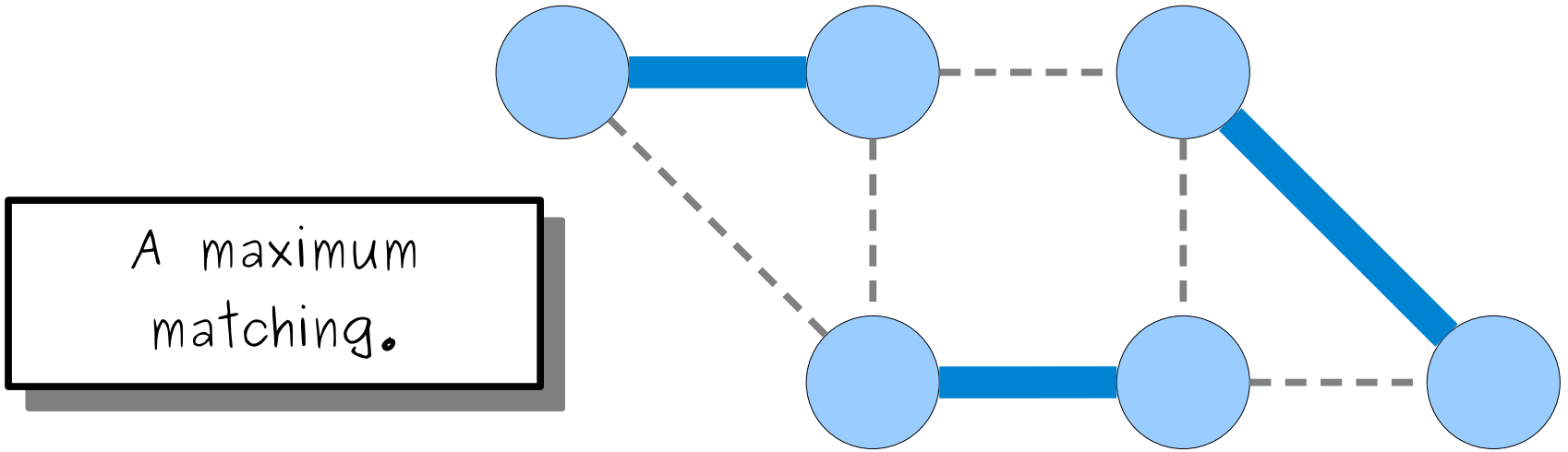
- Given an undirected graph G, a ***matching*** in G is a set of edges such that no two edges share an endpoint.
- A ***maximum matching*** is a matching with the largest number of edges.



A matching, but not a maximum matching.

Maximum Matching

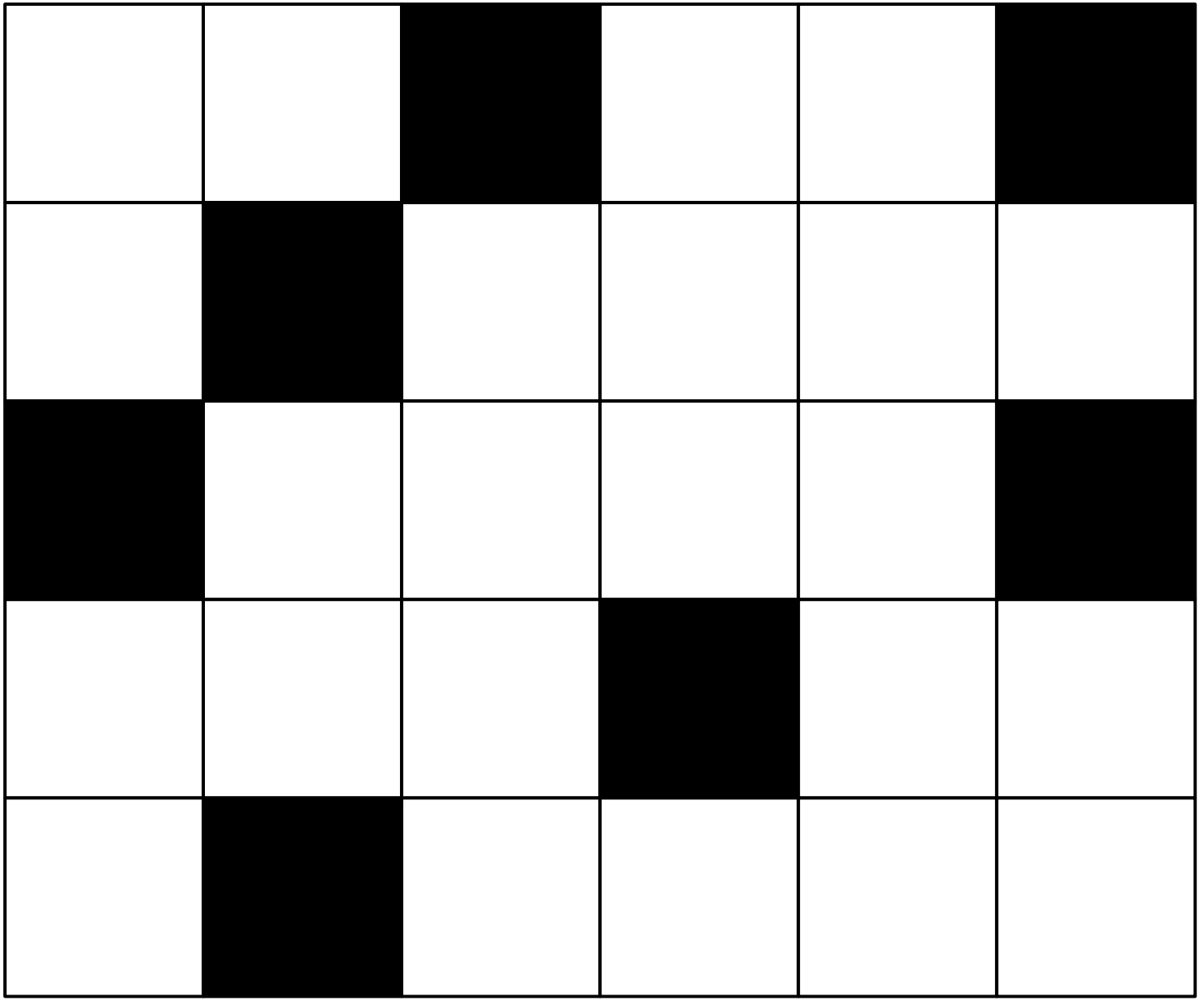
- Given an undirected graph G, a ***matching*** in G is a set of edges such that no two edges share an endpoint.
- A ***maximum matching*** is a matching with the largest number of edges.



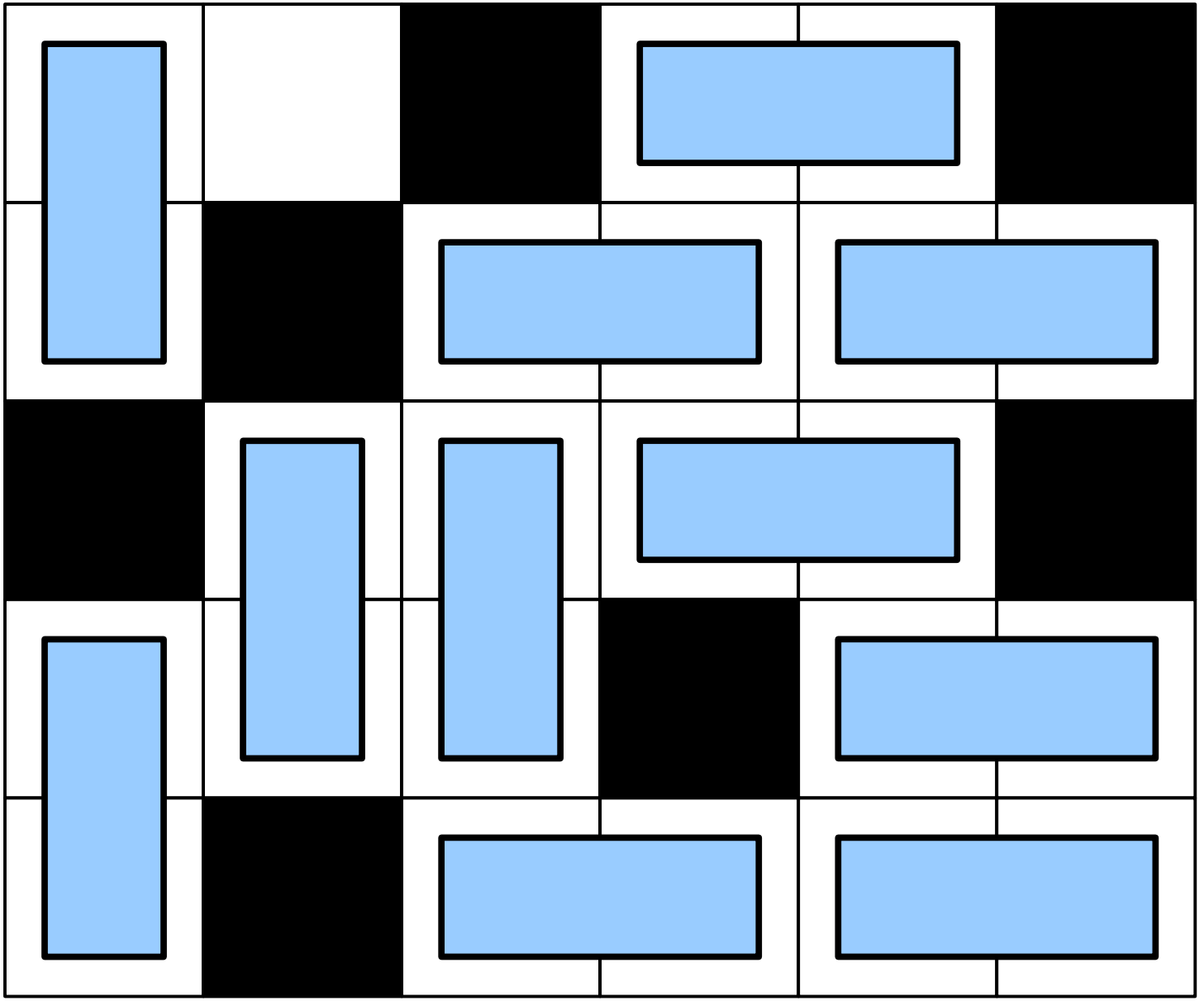
Maximum Matching

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

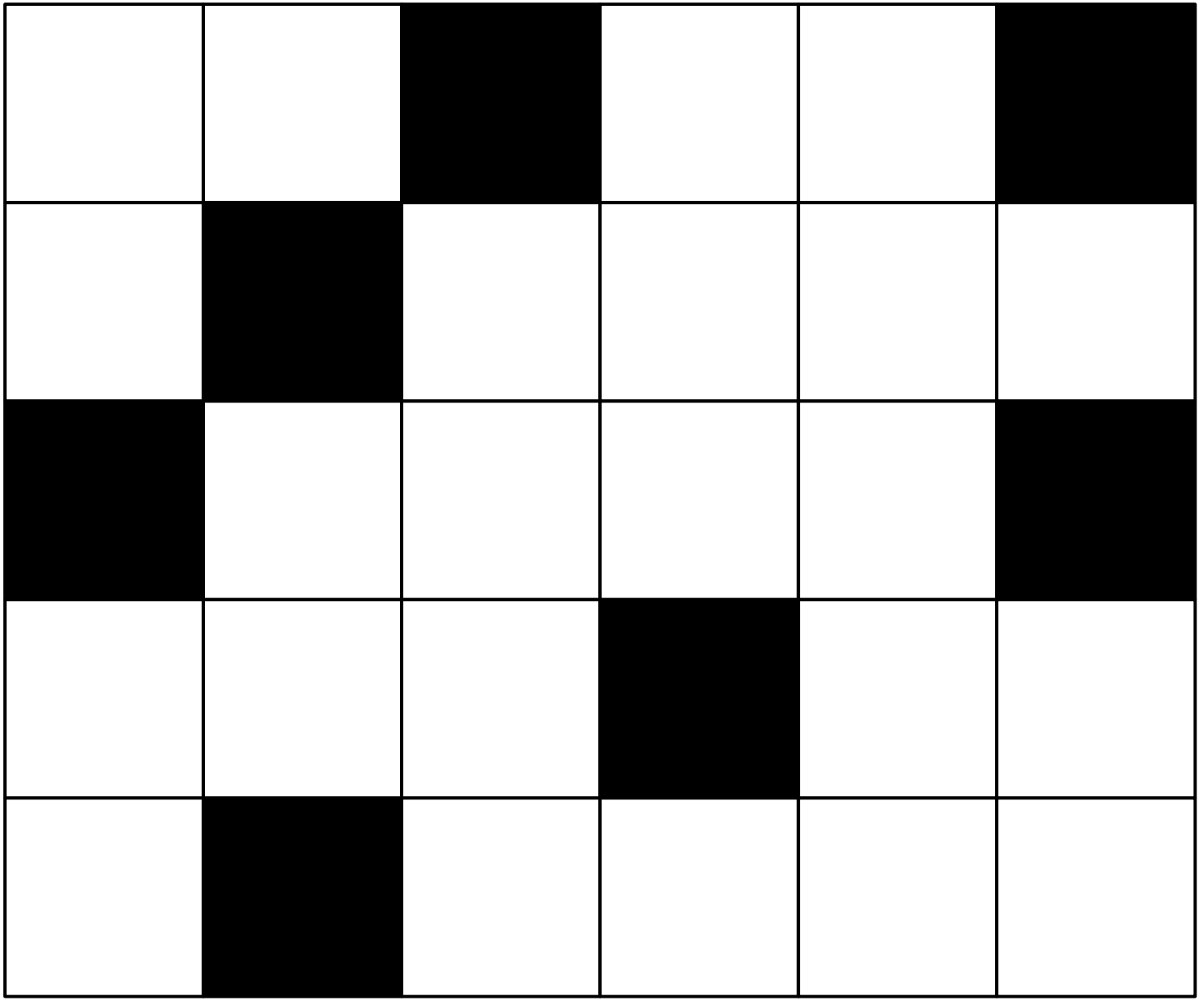
Domino Tiling



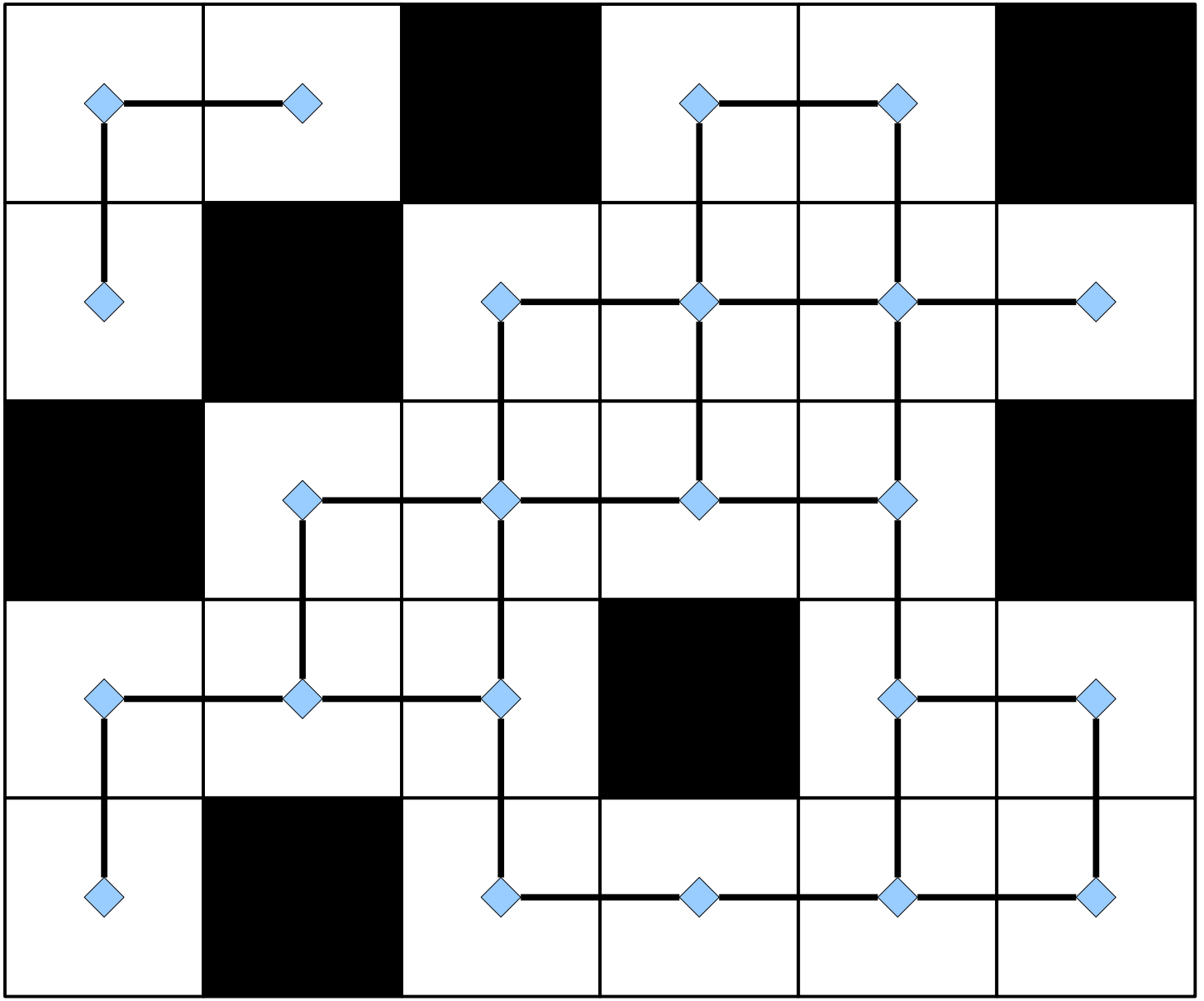
Domino Tiling



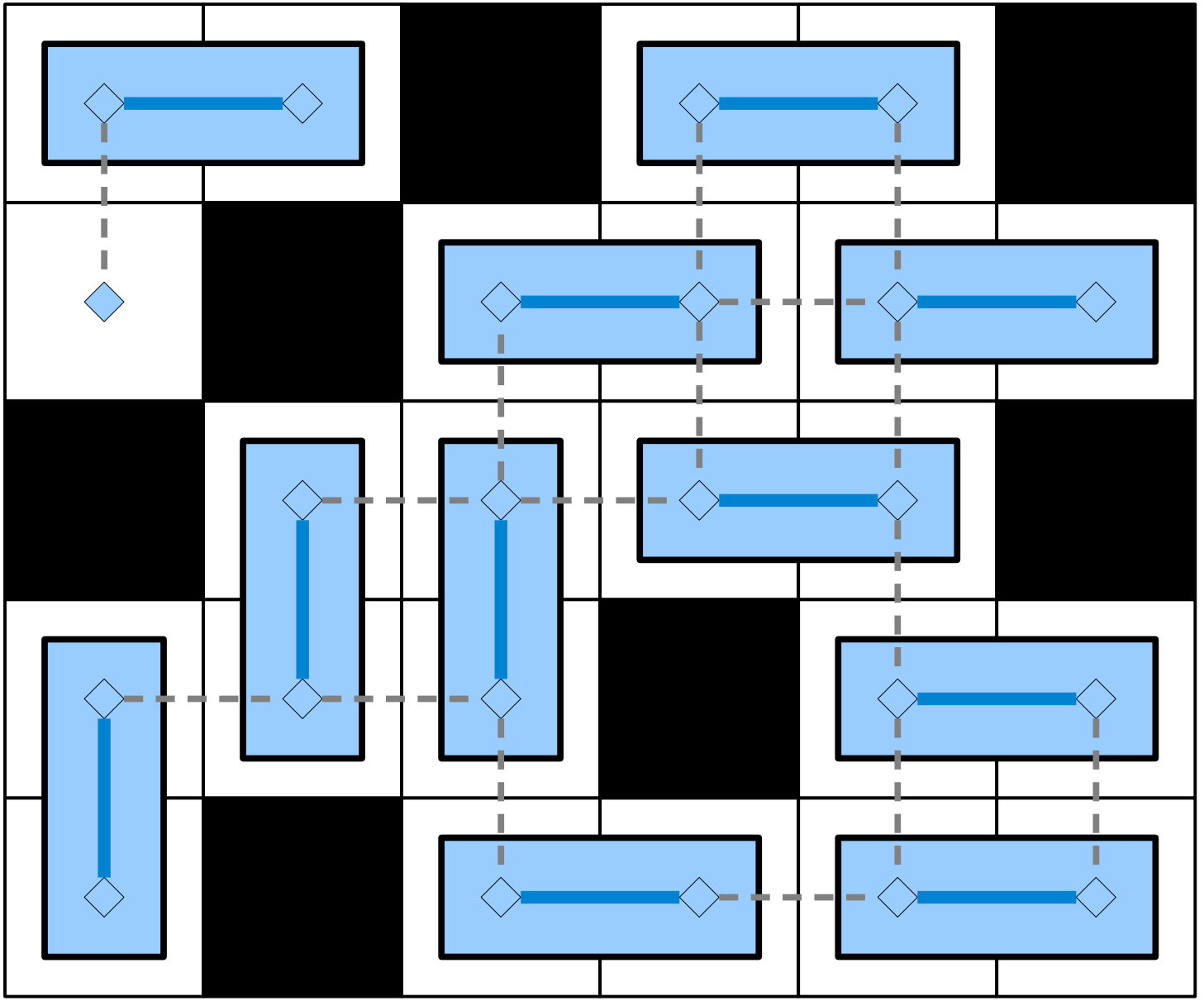
Solving Domino Tiling



Solving Domino Tiling



Solving Domino Tiling



Another Example

Satisfiability

- A propositional logic formula φ 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 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?

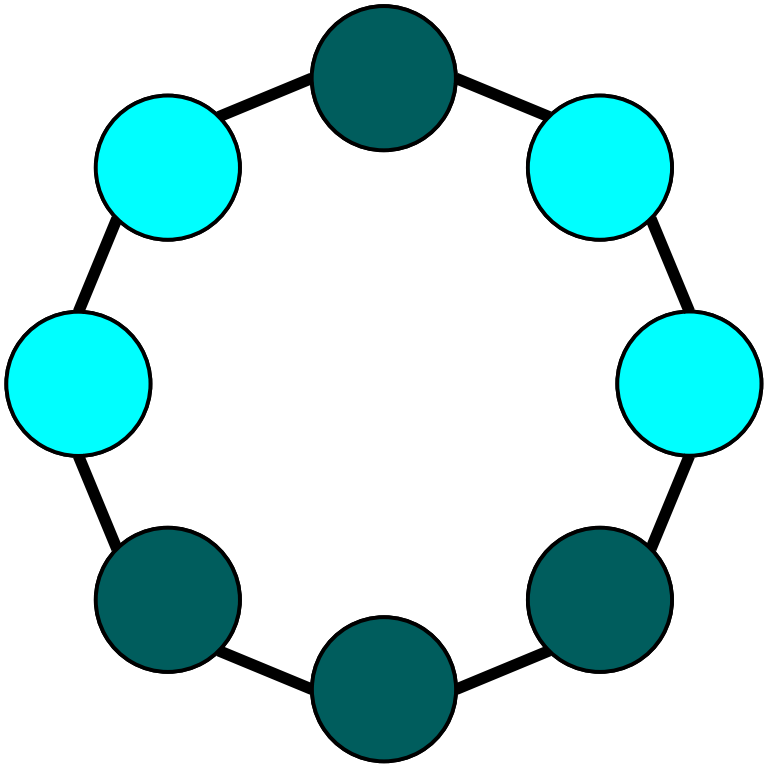
- Formally:

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

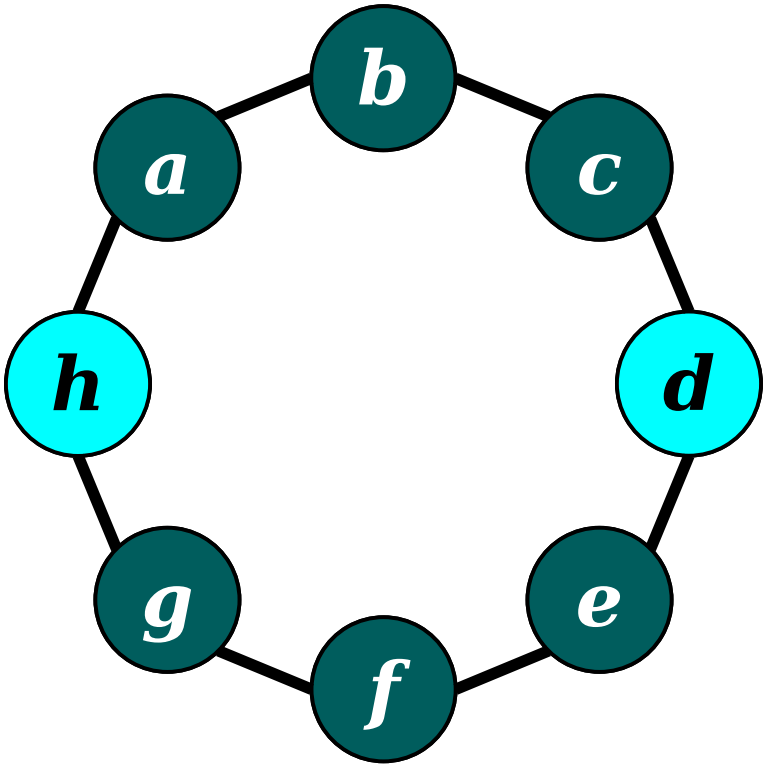
- Finding good algorithms for SAT is an active area of research for reasons we'll discuss later today.
- We have some pretty decent algorithms for solving SAT reasonably quickly most of the time.
- Given this, what other problems can we solve?

Lights Out

- You're given a ring of pushbuttons. Each pushbutton has a light that is either ON or OFF.
- If you push a button, it toggles the state of the two adjacent lights in the ring. (Lights that are ON turn OFF and vice-versa.)
- ***Question:*** Given an initial configuration of lights, can you turn all the lights off?



Solving Lights-Out With a SAT Solver



Observation 1: We never need to press the same button twice.

Observation 2: Button press order doesn't matter.

Idea: Our propositional formula will have one variable per button, indicating whether we press it.

Observation 3: A light that is initially off stays off when an even number of adjacent lights are pressed.

Observation 4: A light that is initially on ends off when an odd number of adjacent lights are pressed.

In Pseudocode

```
bool canTurnLightsOff(LightRing r) {
  return isSatisfiable(ringToFormula(r));
}
```

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

bool canTurnLightsOff(LightRing r) {
   return isSatisfiable(ringToFormula(r));
}
```

```
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

$$A \leq_{\mathrm{p}} B.$$

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

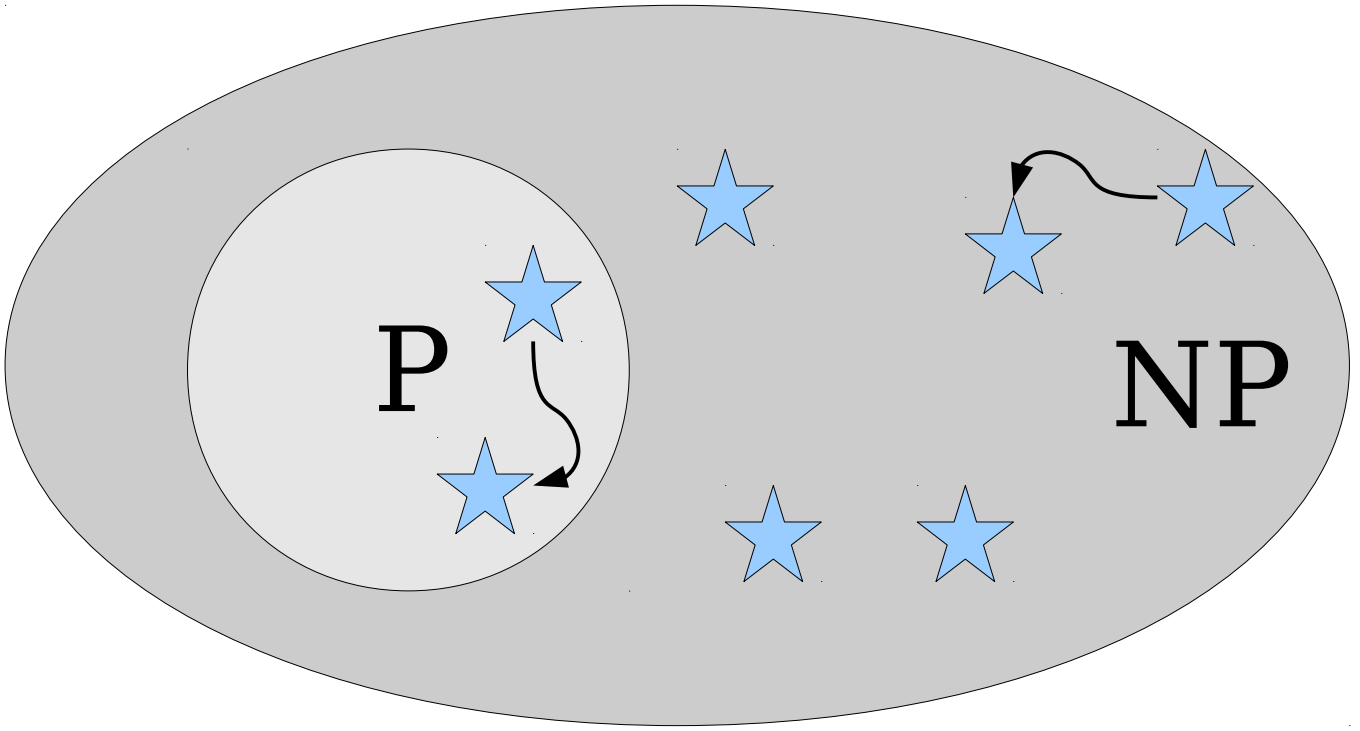
* Assuming that `translate` runs in polynomial time.

```
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**.

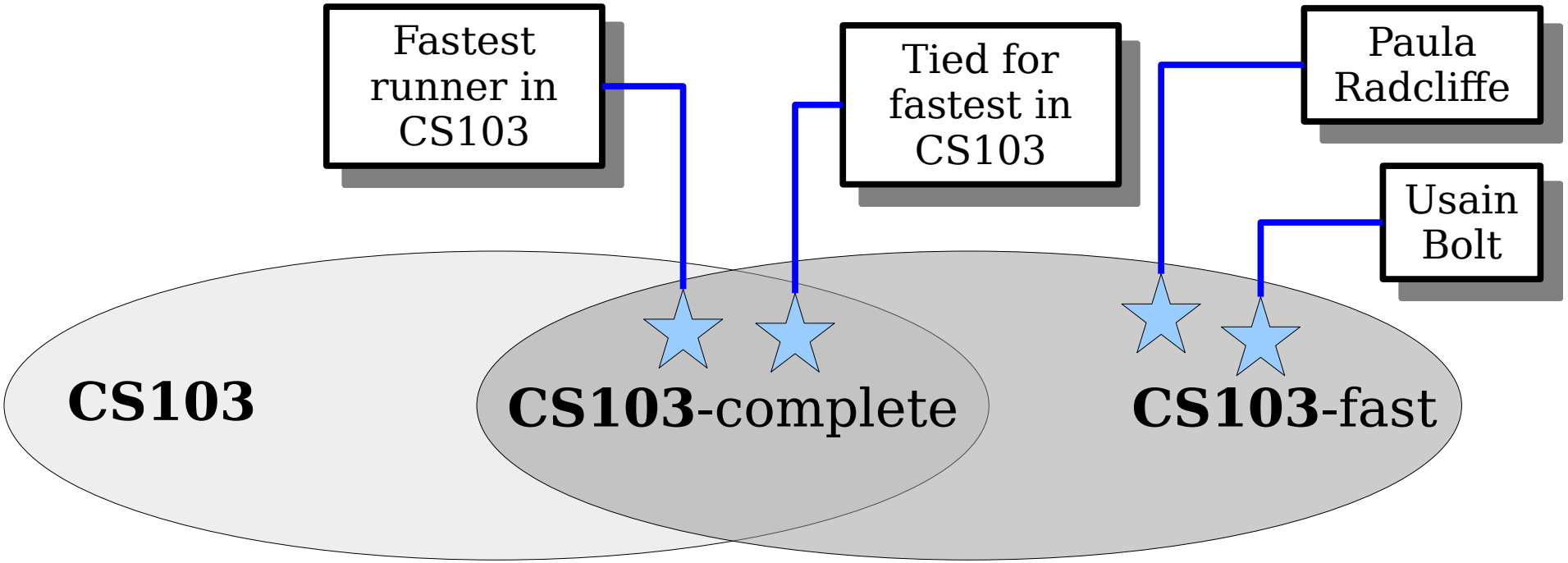
What else can we do with it?


Time out for announcements!
(pending)

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

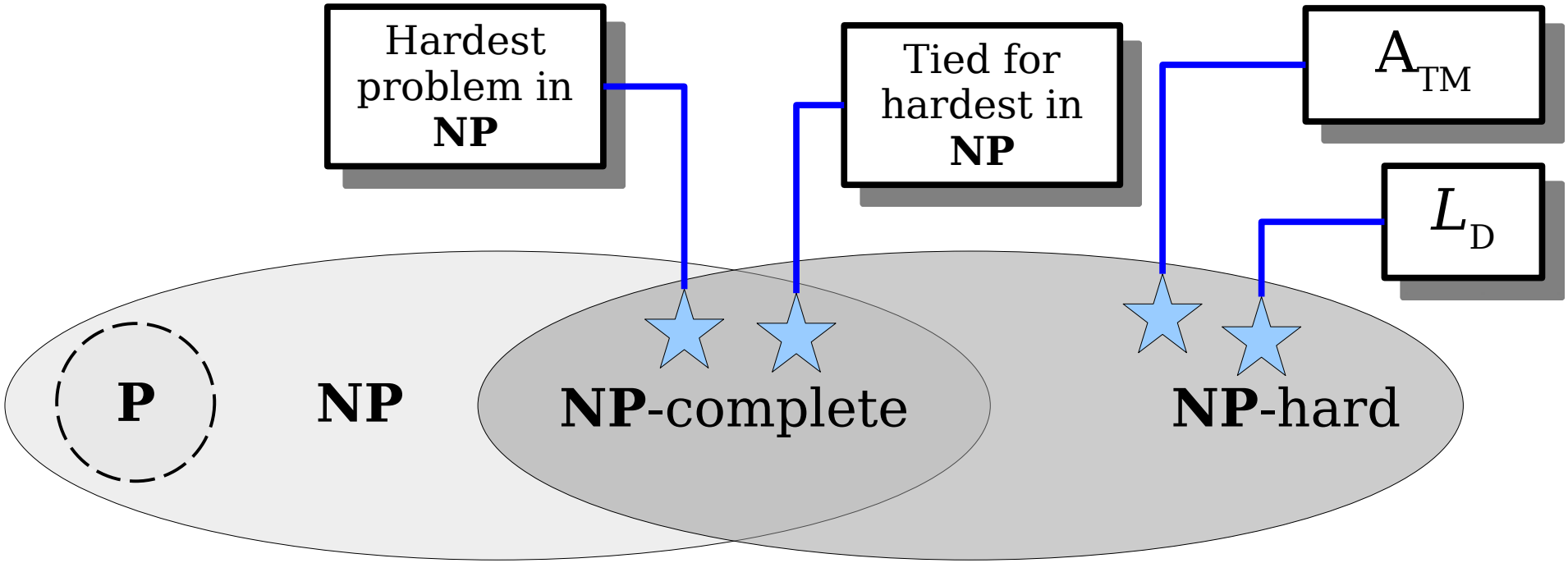
$\forall A \in \mathbf{CS103}.\ A \leq_r P.$

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

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

$P \in \mathbf{CS103}$ and P is **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

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

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

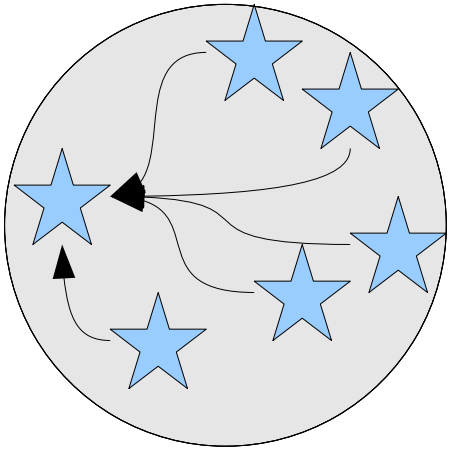
Intuition: The **NP**-complete problems are the hardest problems in **NP**.

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**.

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 $\mathbf{P} = \mathbf{NP}$. ■

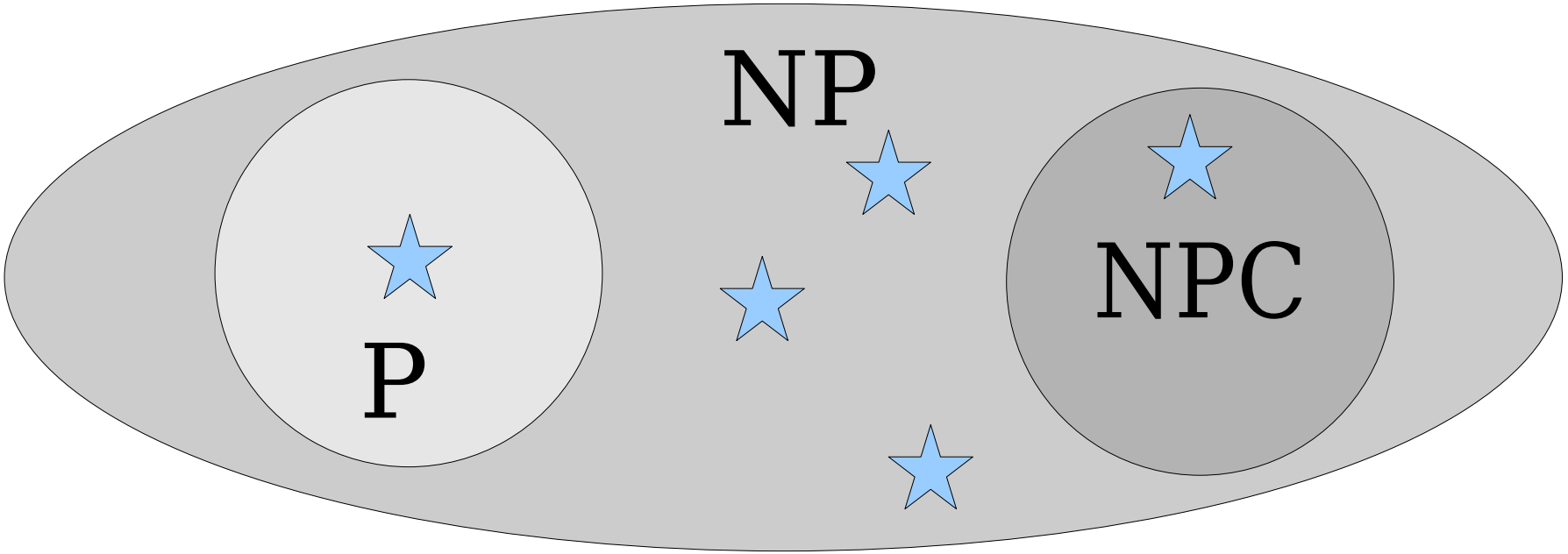


P = NP

The Tantalizing Truth

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

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



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

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

Proof Idea: To see that SAT ∈ **NP**, show how to make a polynomial-time verifier for it. Key idea: the certificate is a candidate satisfying assignment.

To show that SAT is **NP**-hard, given a polymomial-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's 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.

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

- 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!

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)*

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.

Next Time

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