# Complexity Theory

Part Two

## Recap from Last Time

## The Complexity Class **P**

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

  **P** = { *L* | There is a polynomial-time decider for *L* }

- Intuitively, **P** contains all decision problems that can be solved efficiently.
- This is like class **P**, 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:

  **NP** = { *L* | 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:

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

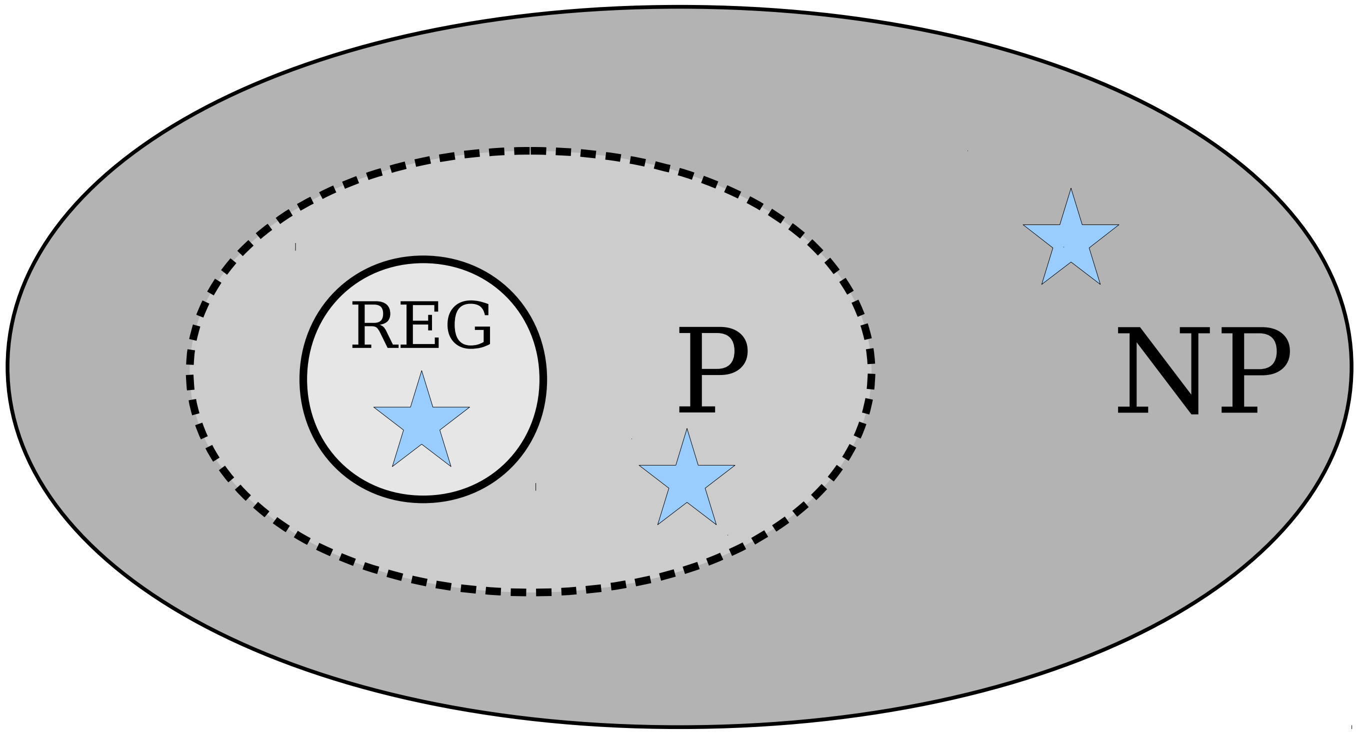
***Theorem (Baker-Gill-Solovay):*** Any proof that purely relies on universality and self-reference cannot resolve $\mathbf{P} \overset{?}{=} \mathbf{NP}$.

***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?

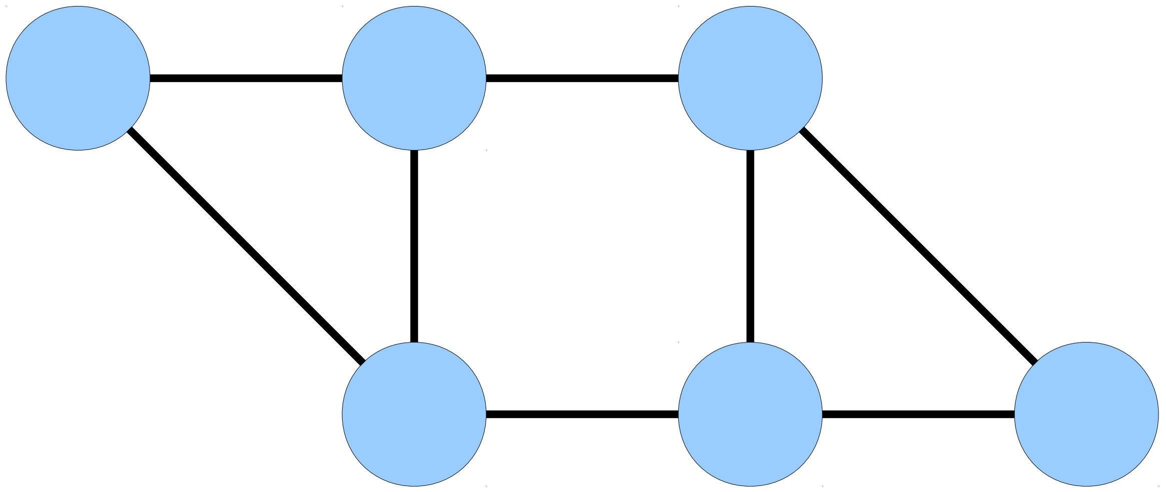
# Reducibility

# 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

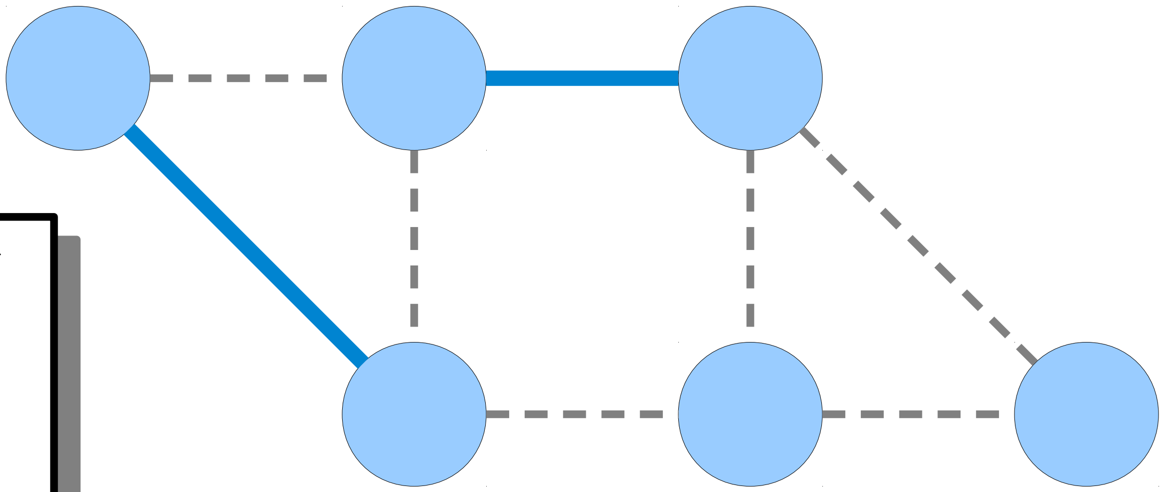
- Given an undirected graph *G*, a ***matching*** in *G* is a set of edges such that no two edges share an endpoint.
- A ***maximum matching*** is a matching with the largest number of edges.



# Maximum Matching

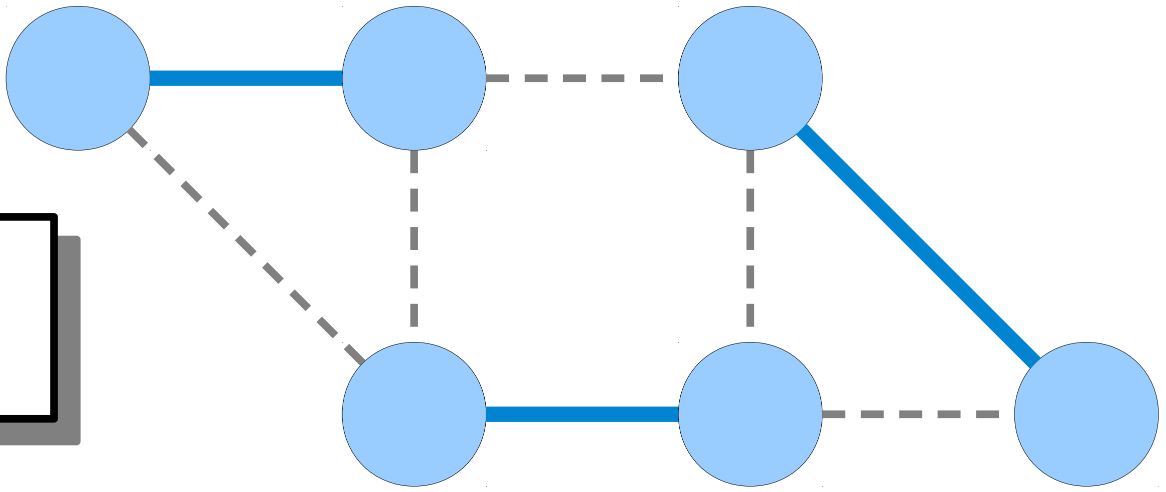
- Given an undirected graph *G*, a ***matching*** in *G* is a set of edges such that no two edges share an endpoint.
- A ***maximum matching*** is a matching with the largest number of edges.

A matching, but not a maximum matching.



## Maximum Matching

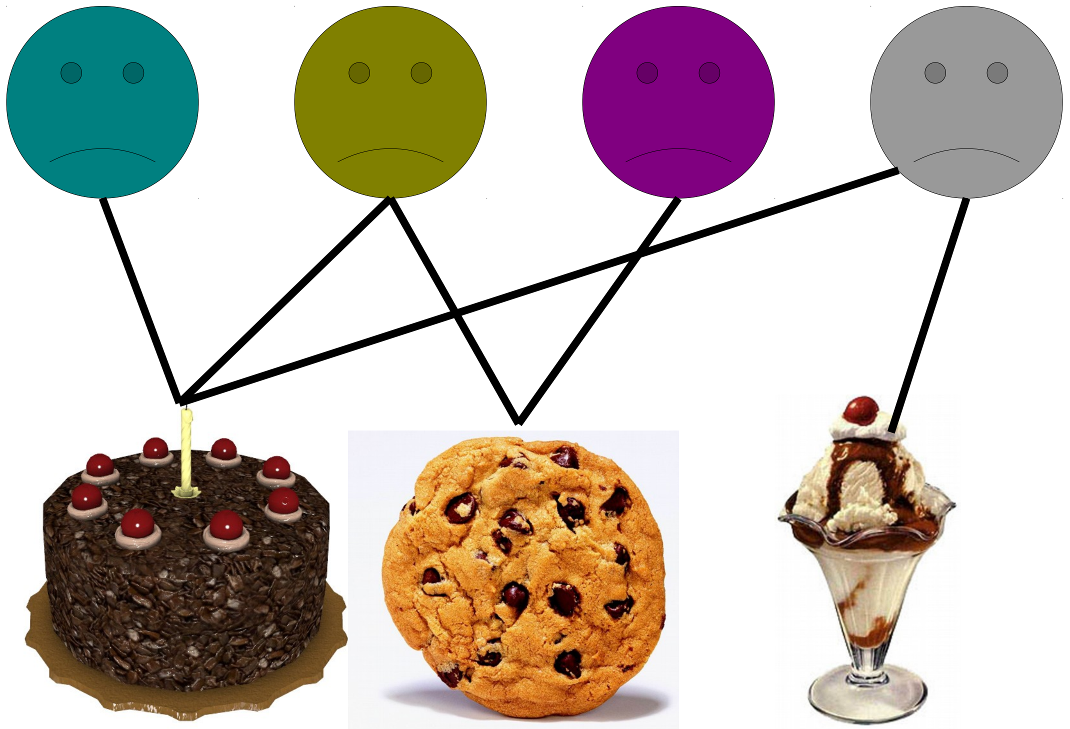
- Given an undirected graph *G*, a ***matching*** in *G* is a set of edges such that no two edges share an endpoint.
- A ***maximum matching*** is a matching with the largest number of edges.



A maximum matching.

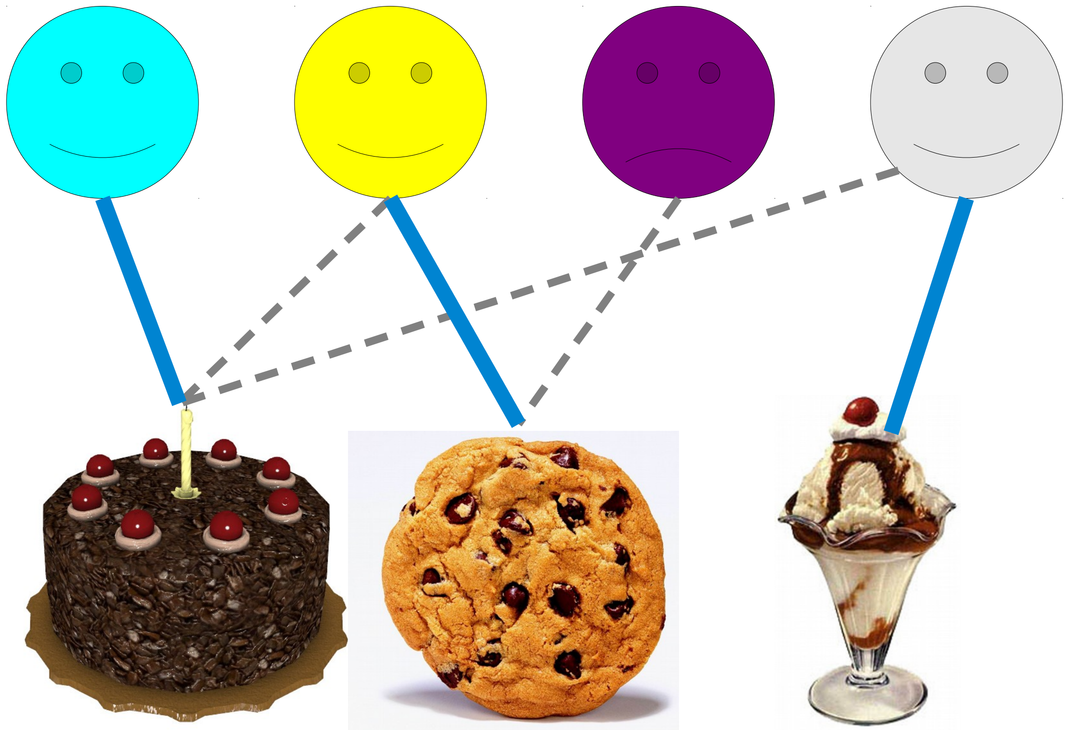
# Maximum Matching

- 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

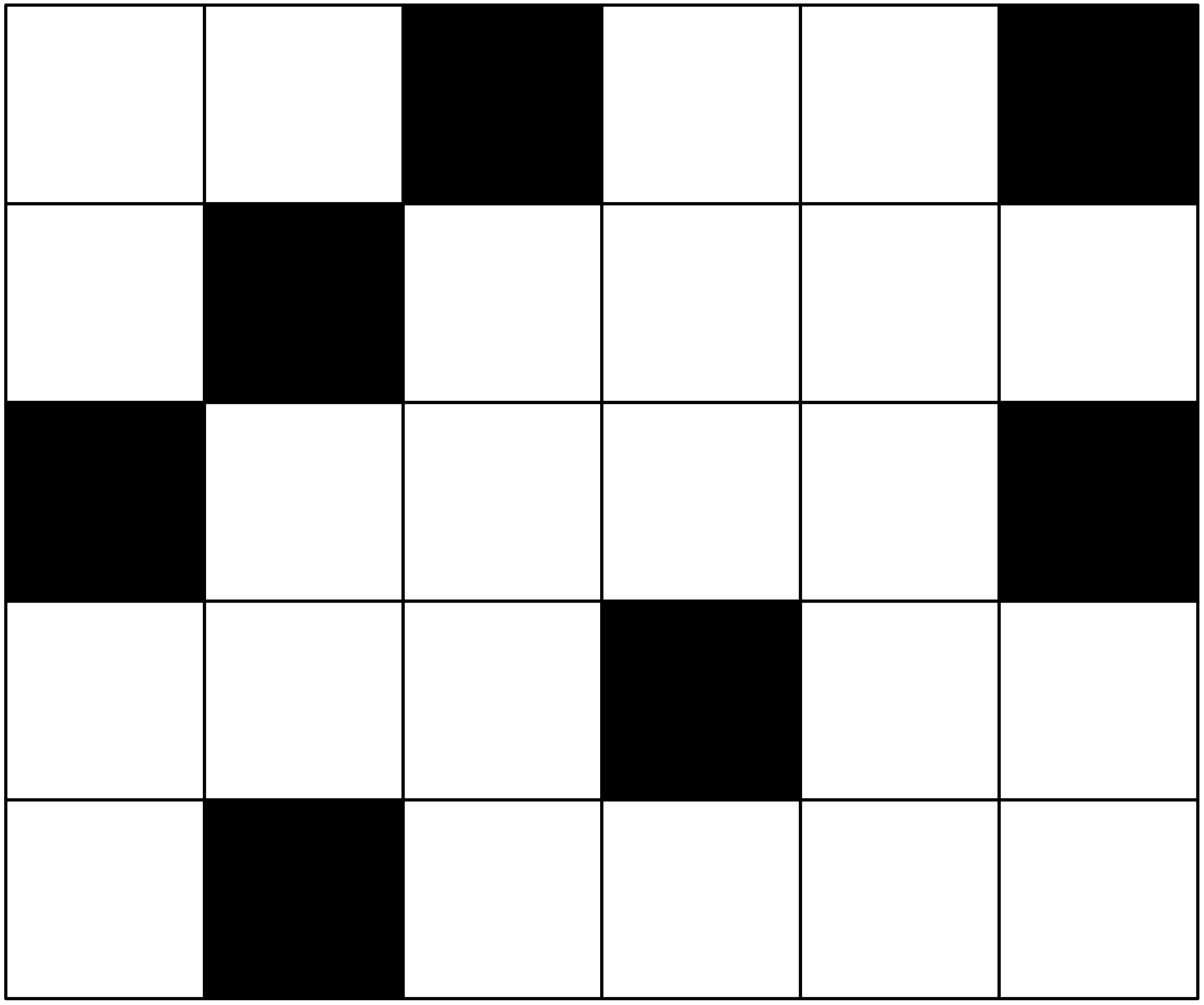
- Given an undirected graph *G*, a ***matching*** in *G* is a set of edges such that no two edges share an endpoint.
- A ***maximum matching*** is a matching with the largest number of edges.



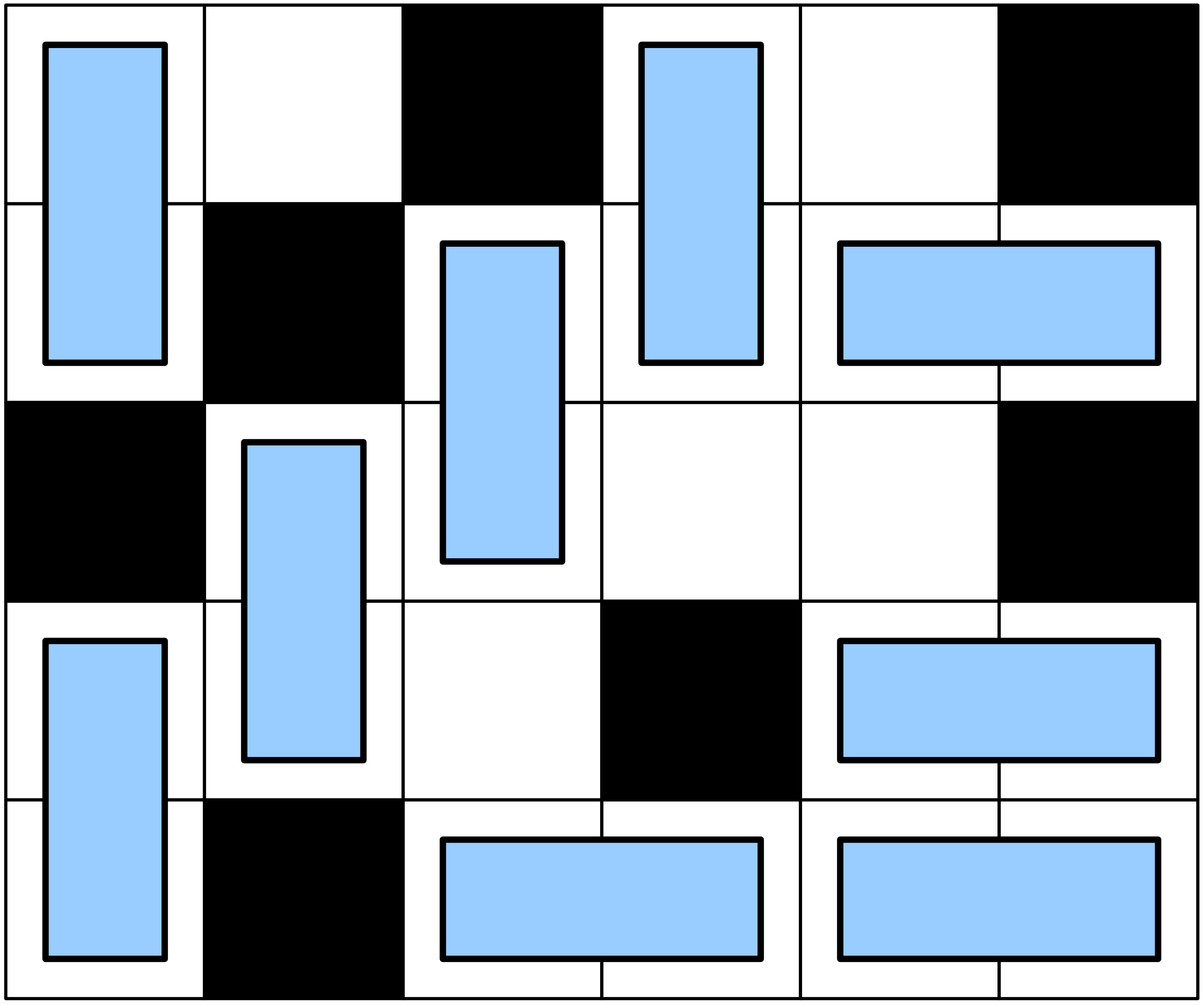
## Maximum Matching

- 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?

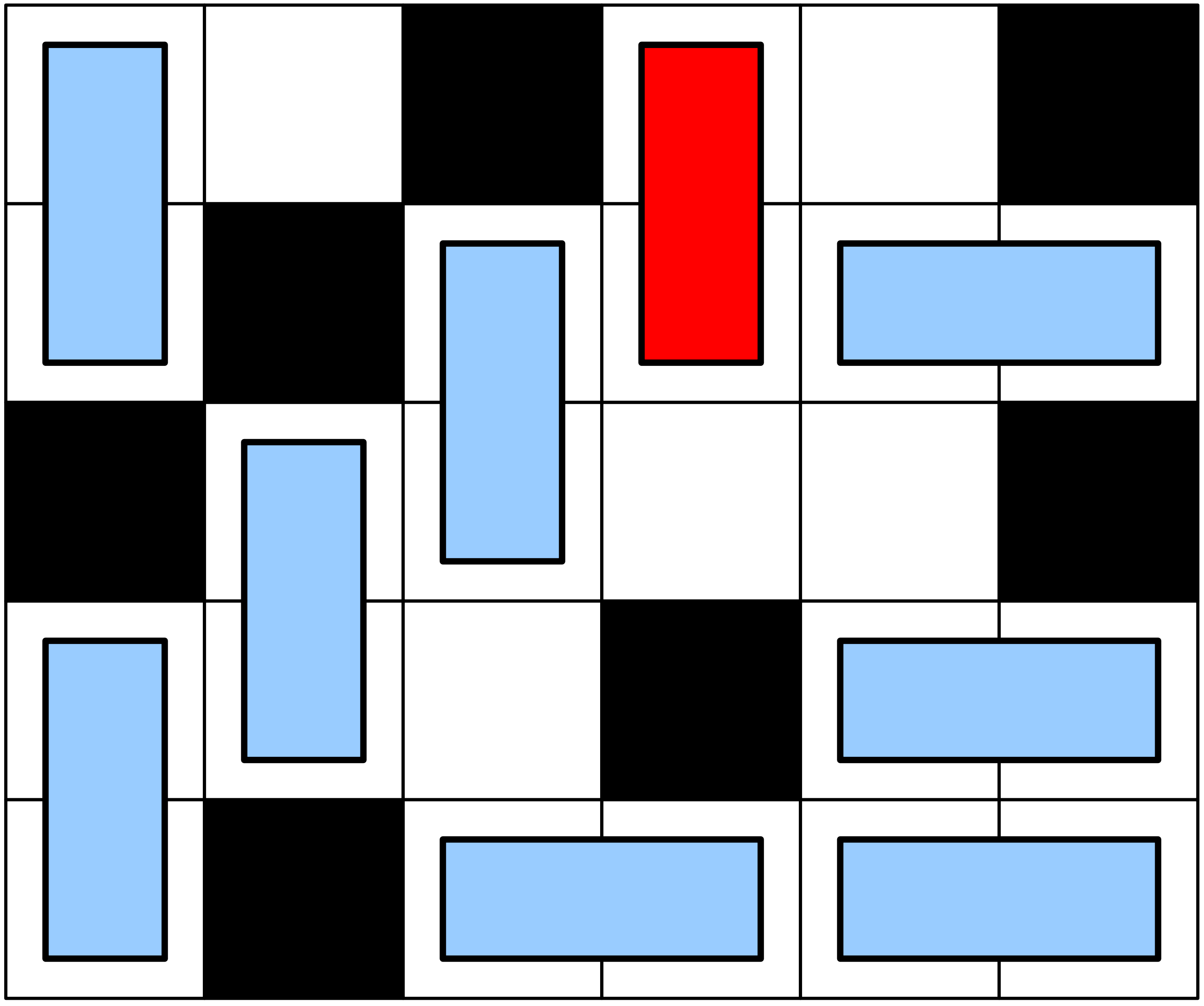
# Domino Tiling



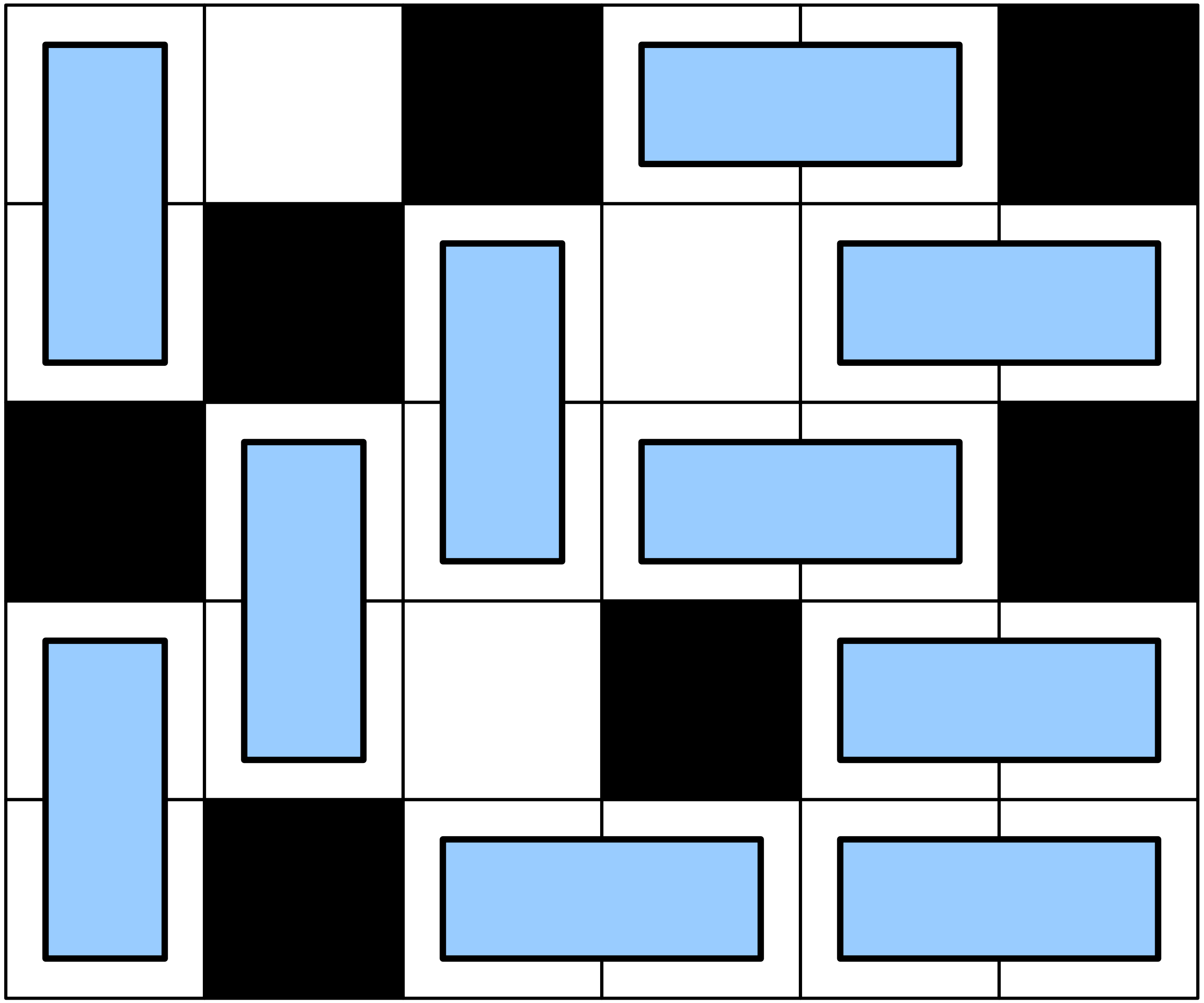
# Domino Tiling



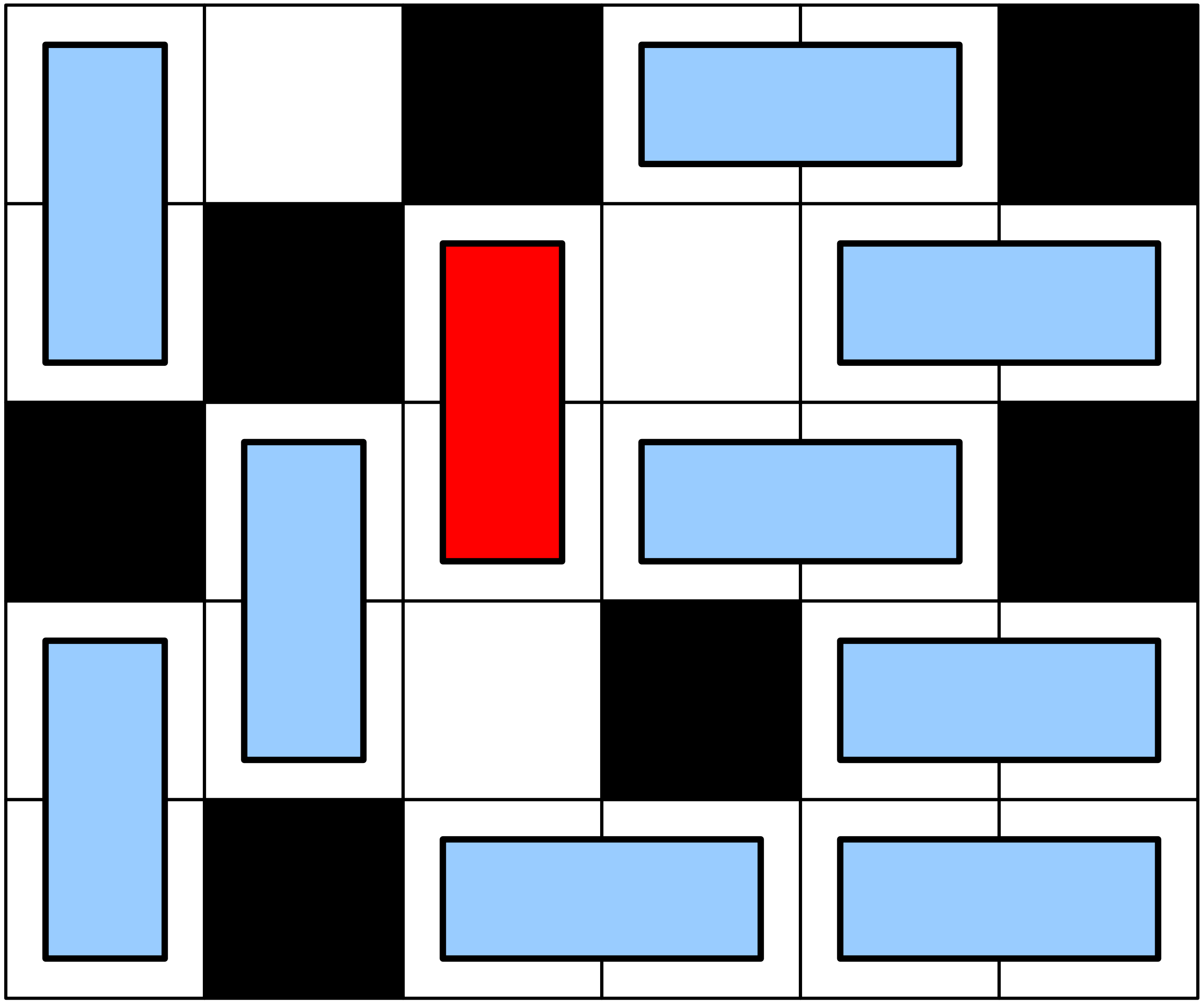
# Domino Tiling



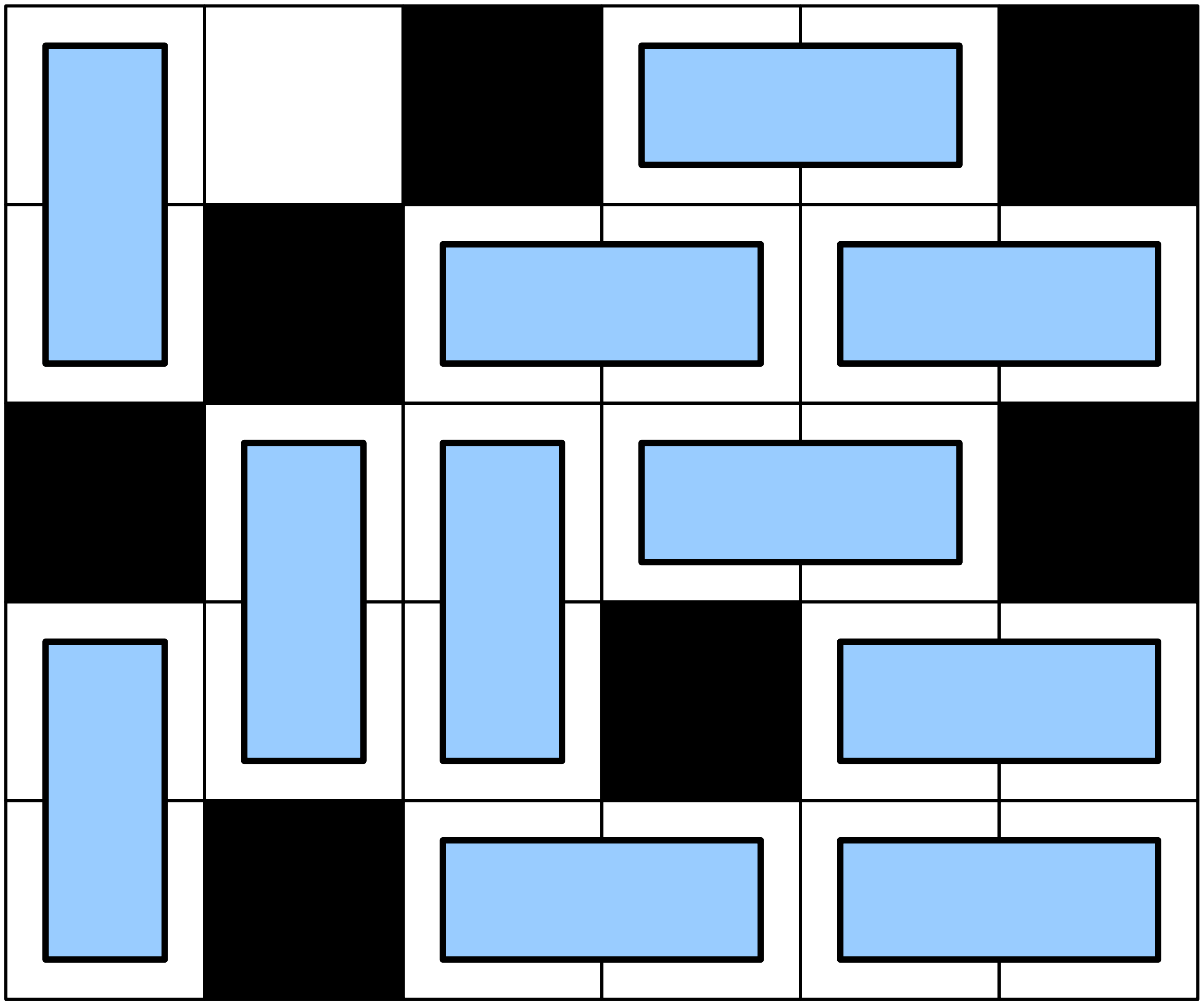
## Domino Tiling



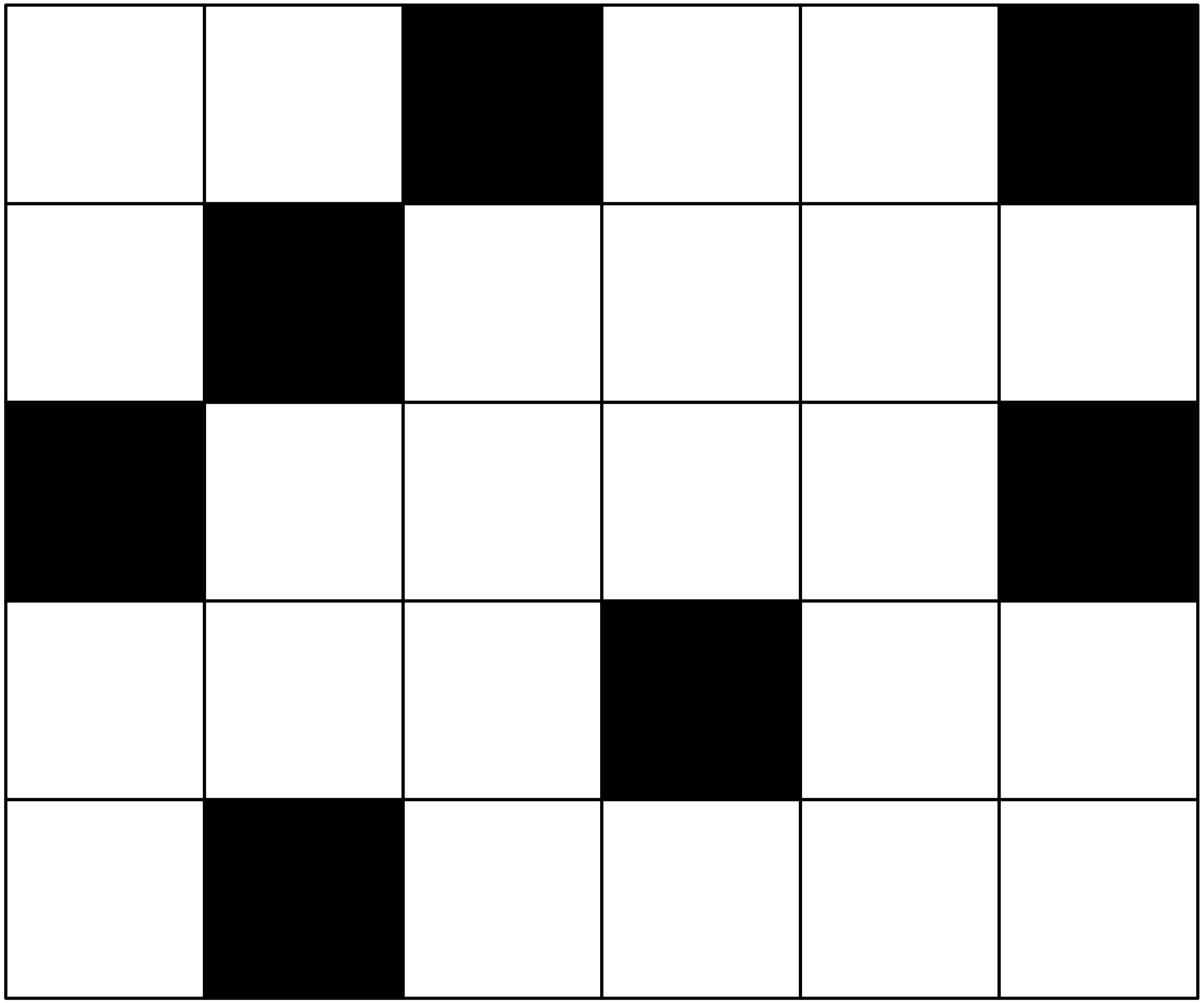
# Domino Tiling



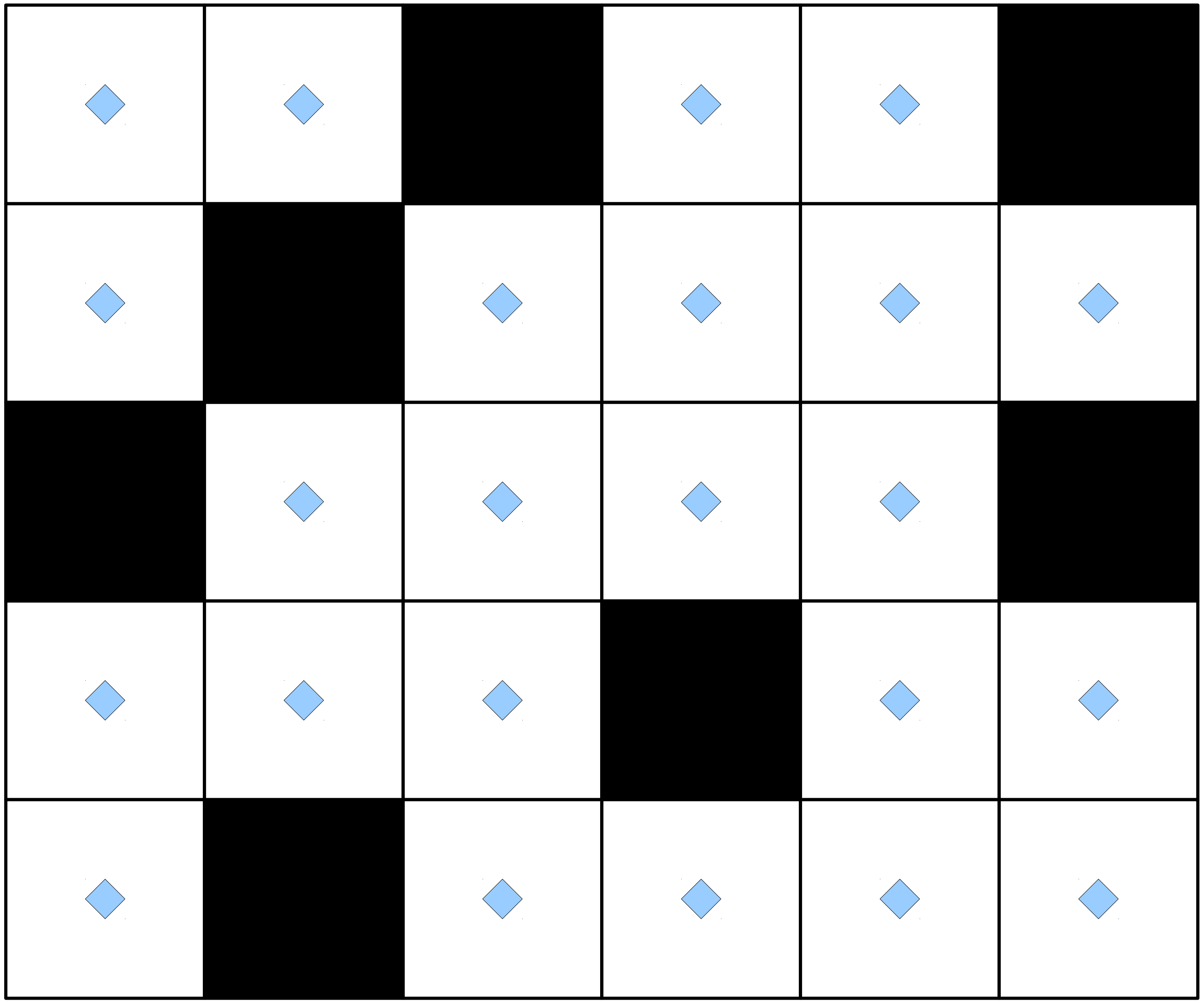
# Domino Tiling



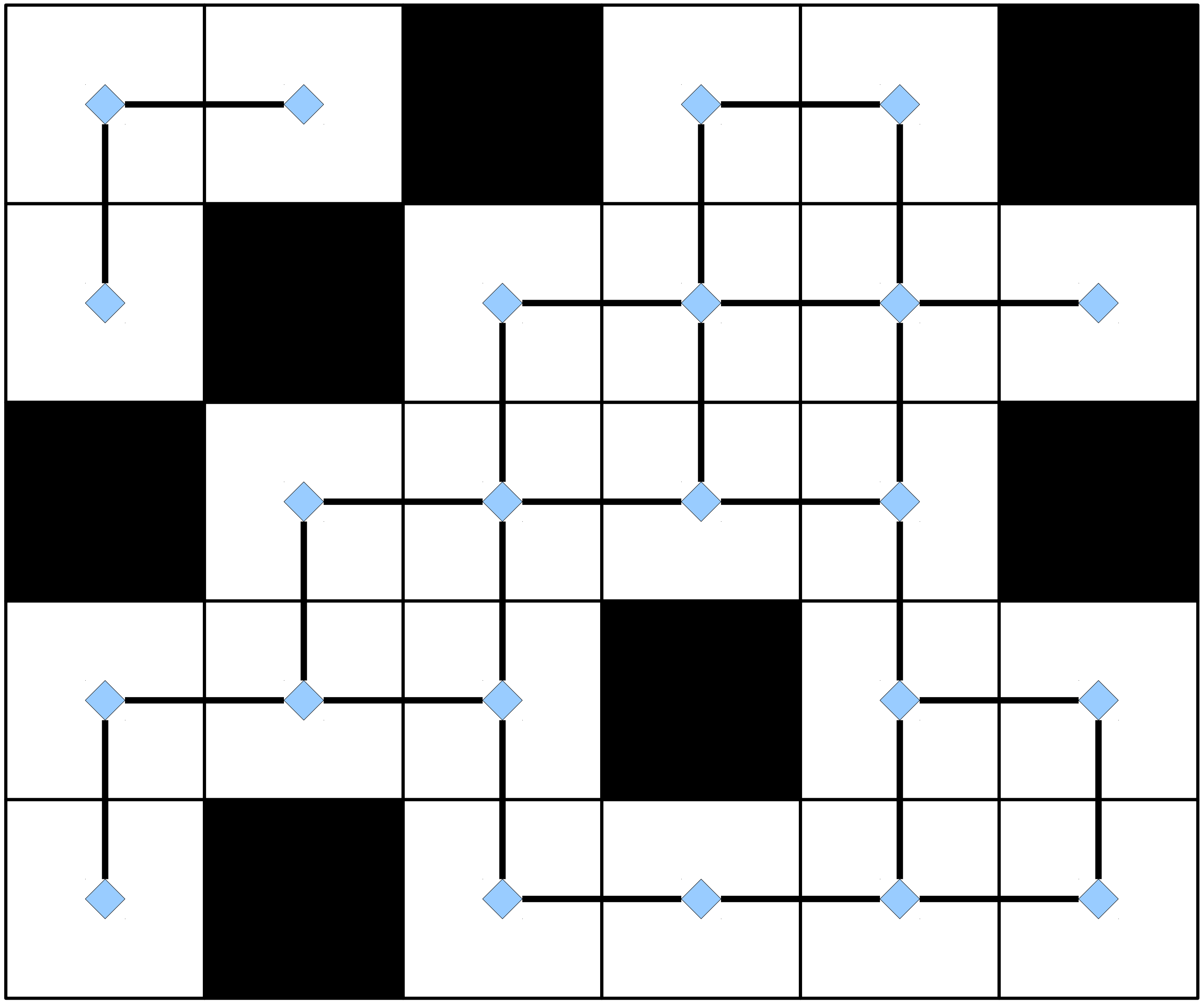
Solving Domino Tiling



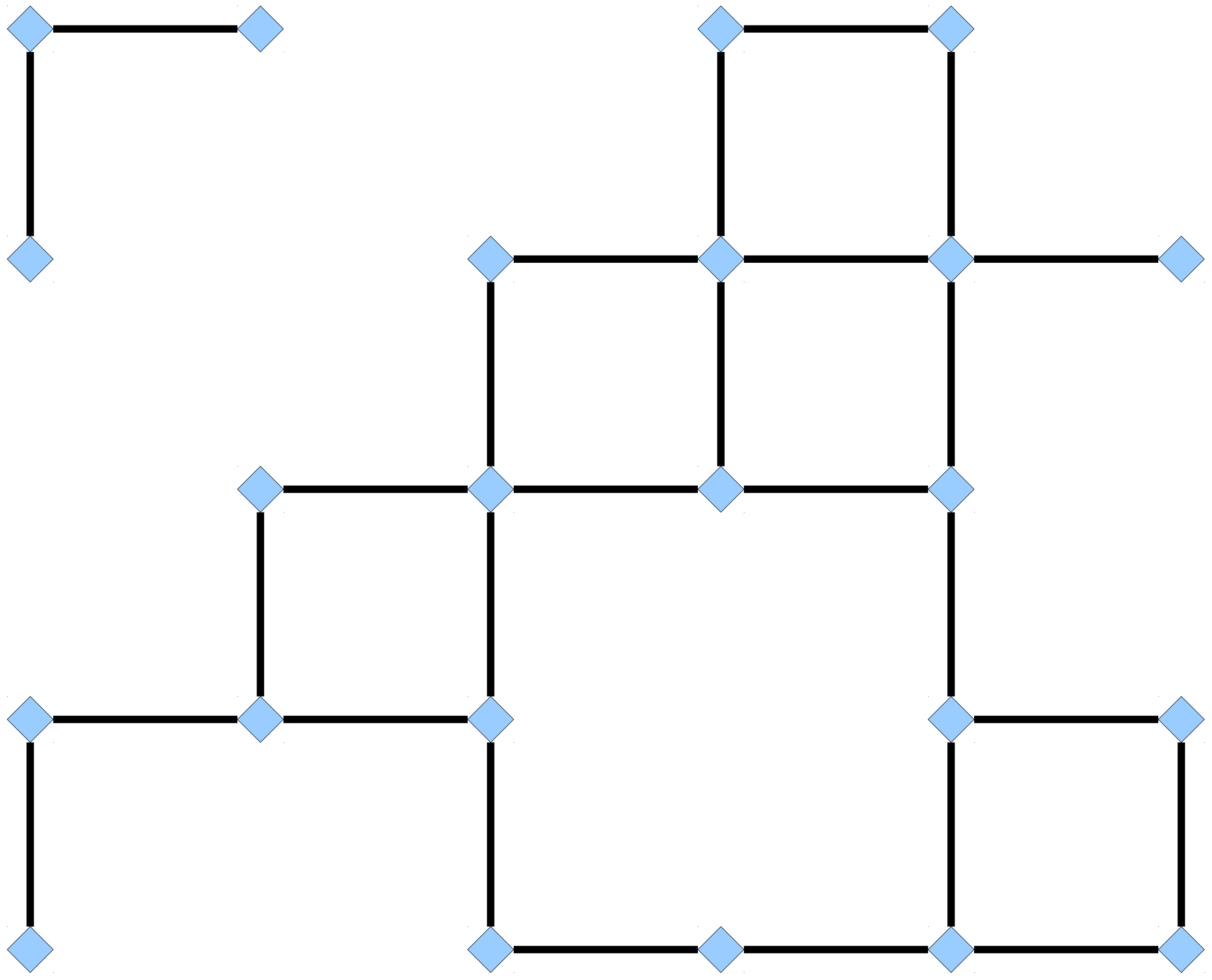
# Solving Domino Tiling



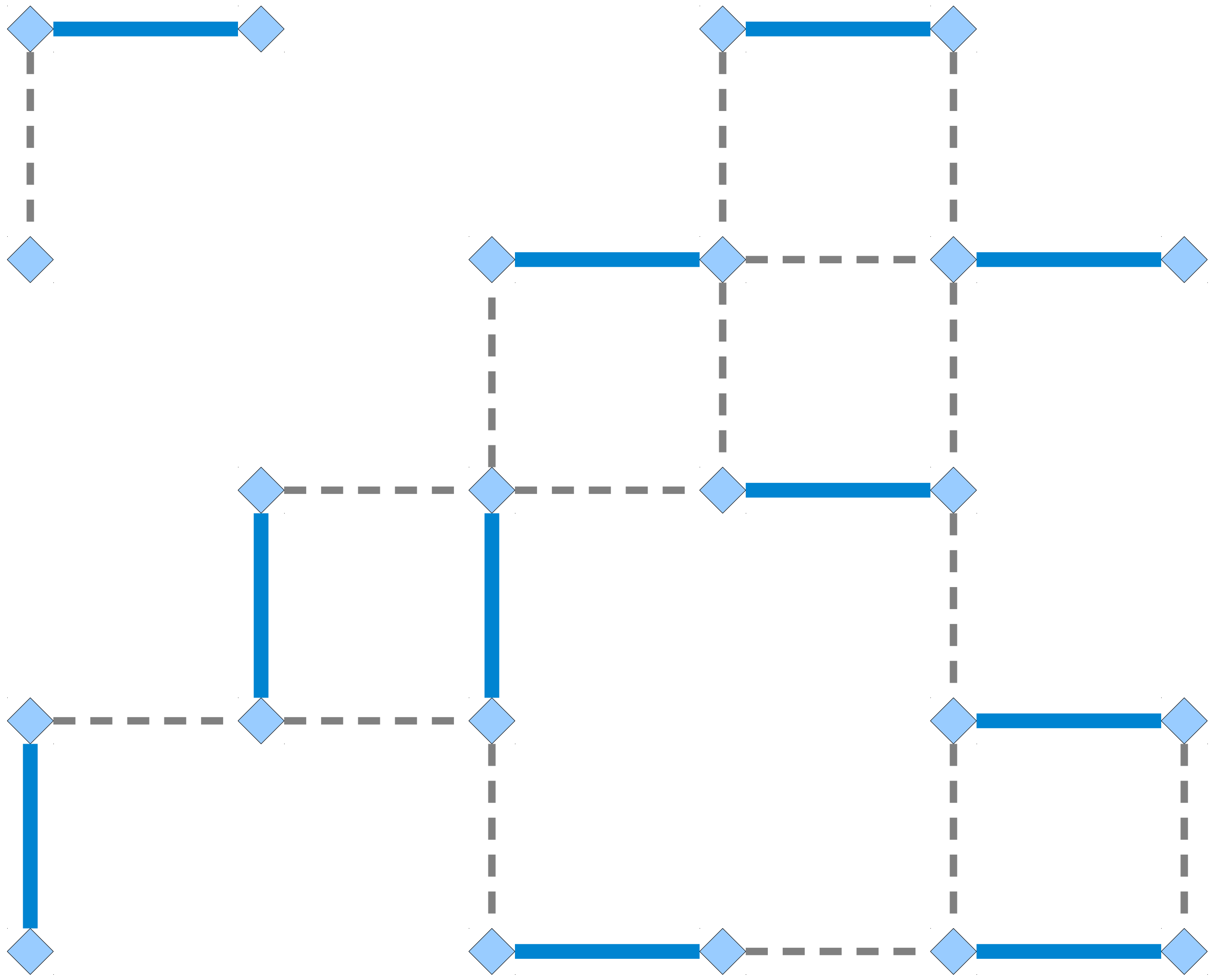
# Solving Domino Tiling



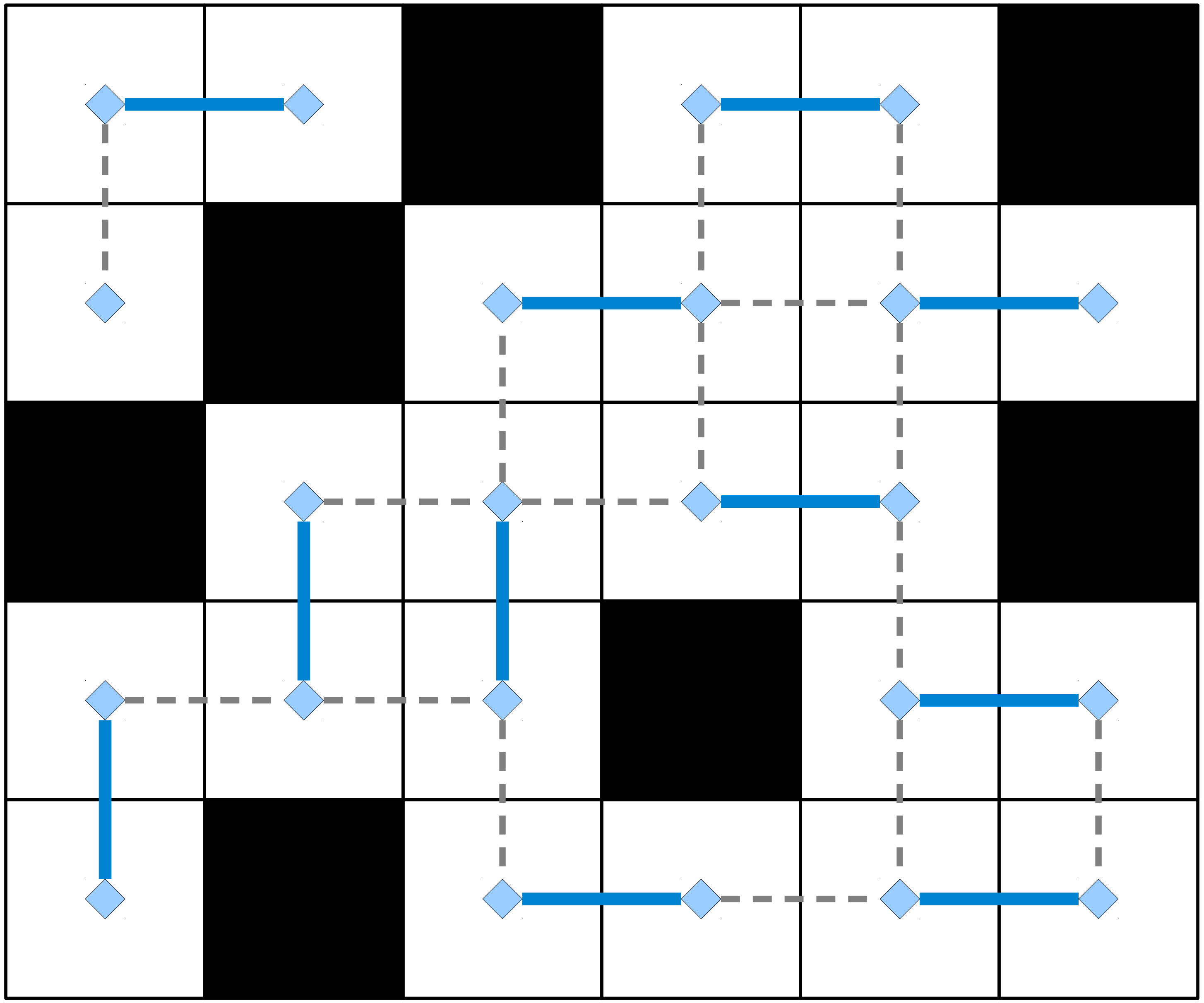
# Solving Domino Tiling



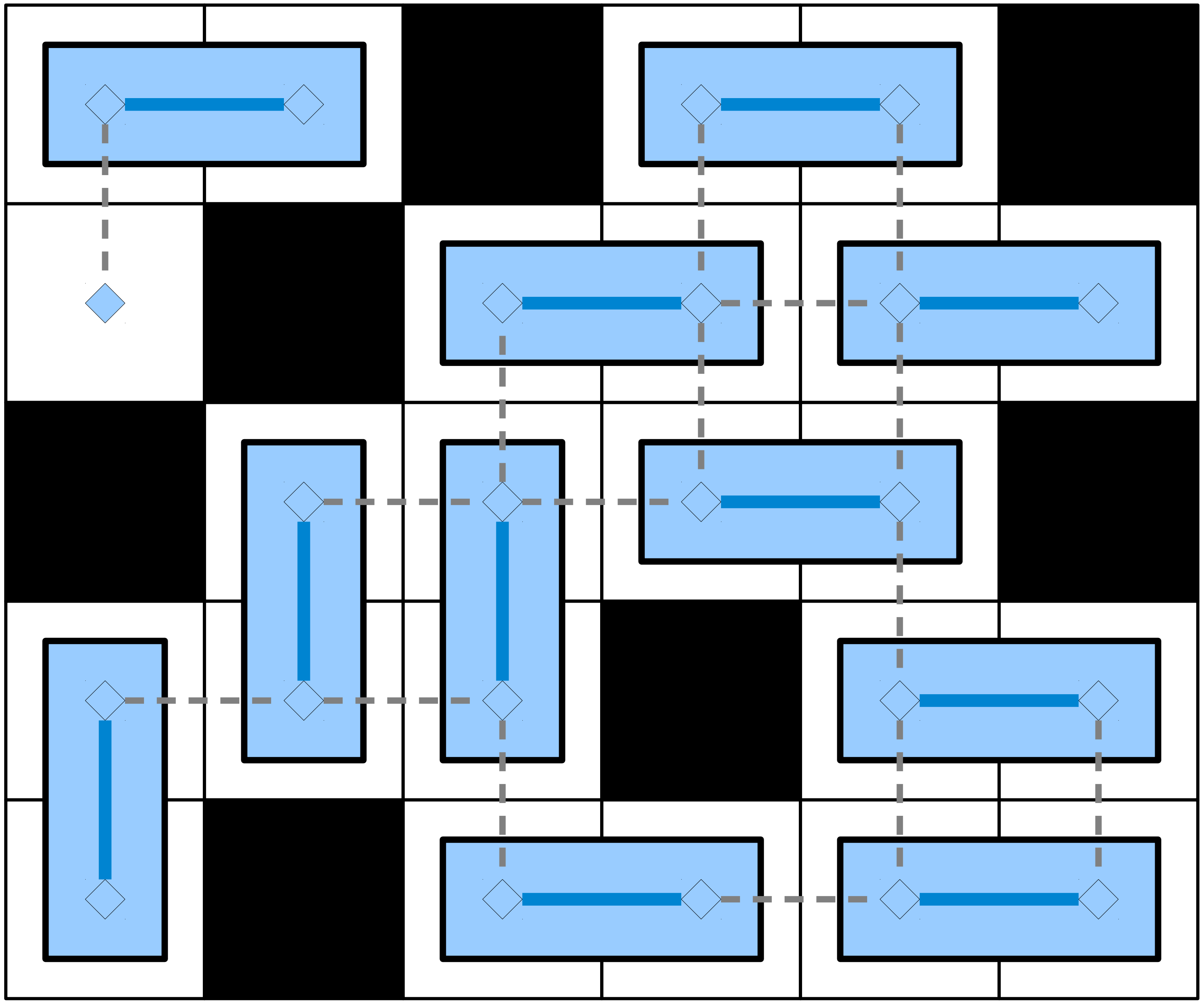
# Solving Domino Tiling



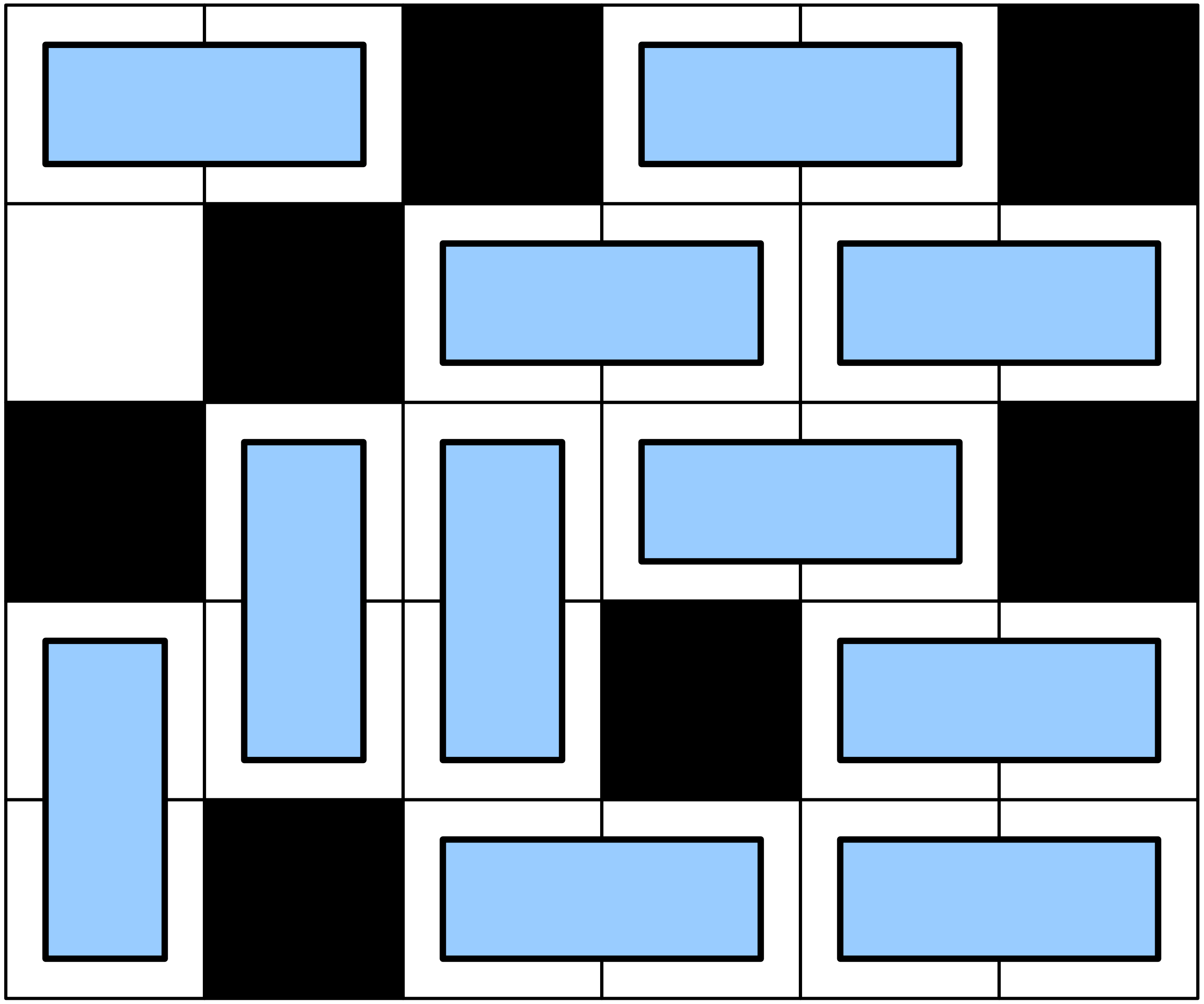
# Solving Domino Tiling



## Solving Domino Tiling



# Solving Domino Tiling



## In Pseudocode

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

## ***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

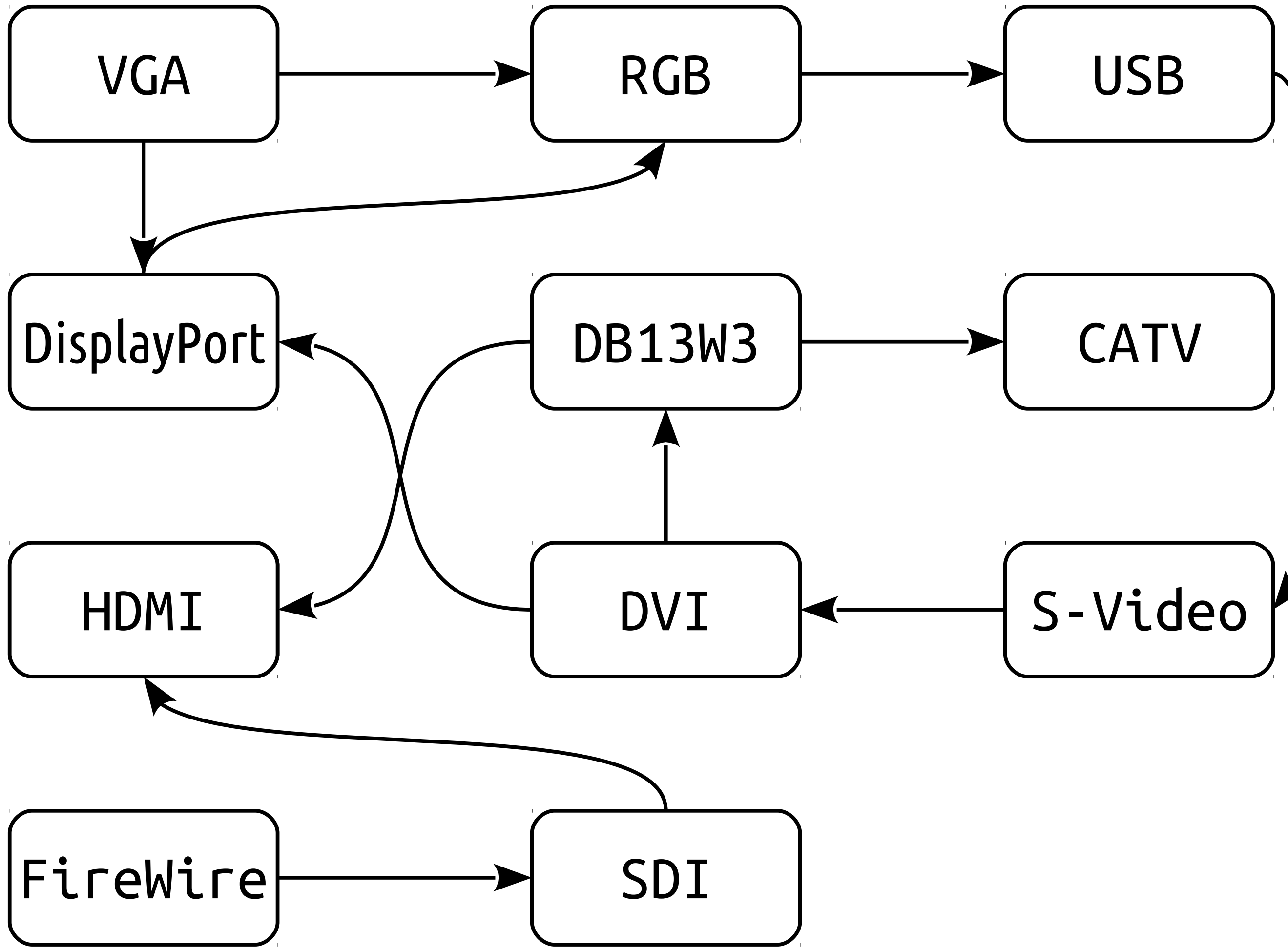
**Connectors**

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

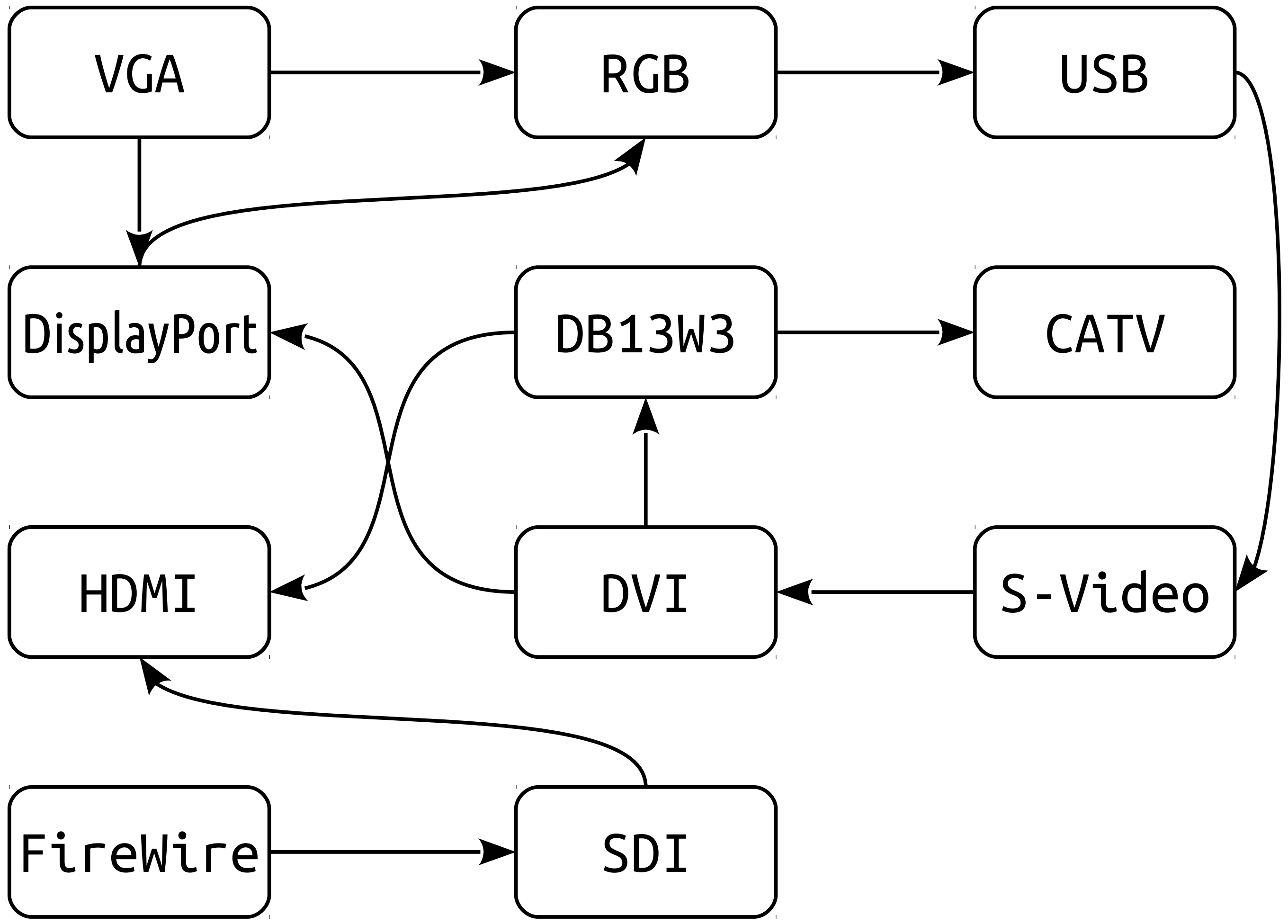
# Converter Conundrums

**Connectors**

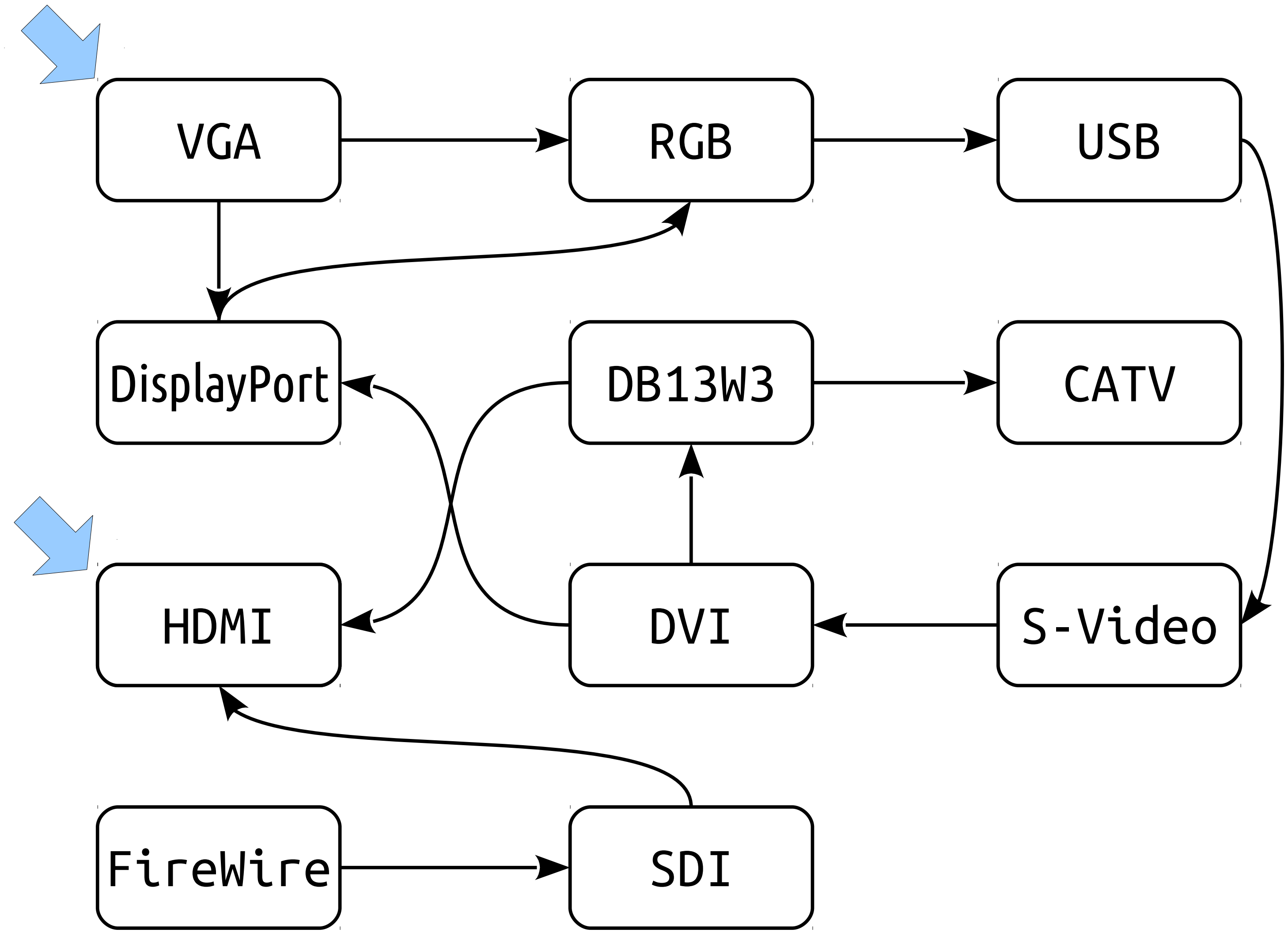
RGB to USB
VGA to DisplayPort
DB13W3 to CATV
DisplayPort to RGB
DB13W3 to HDMI
DVI to DB13W3
S-Video to DVI
FireWire to SDI
VGA to RGB
DVI to DisplayPort
USB to S-Video
SDI to HDMI



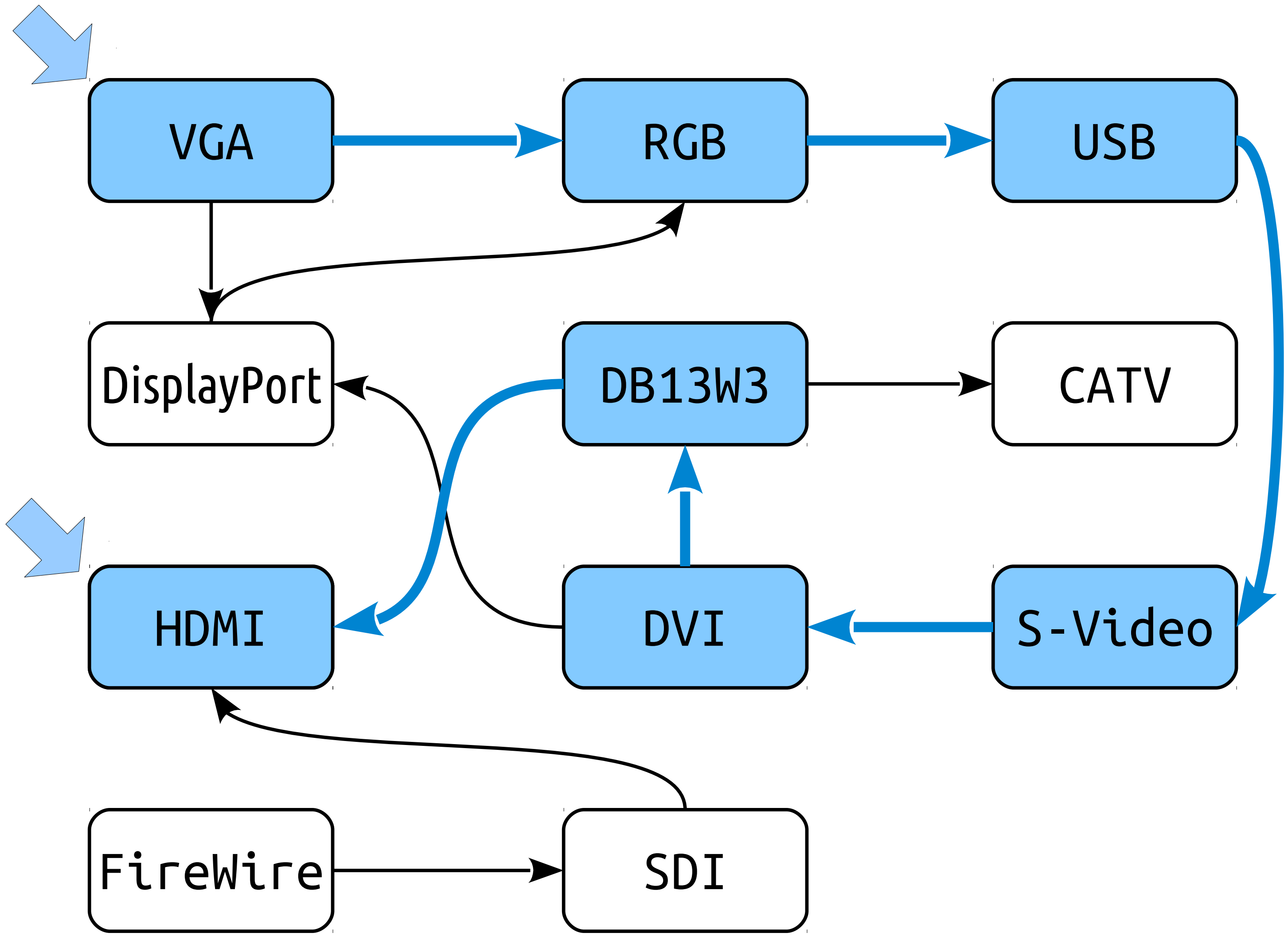
# Converter Conundrums



# Converter Conundrums

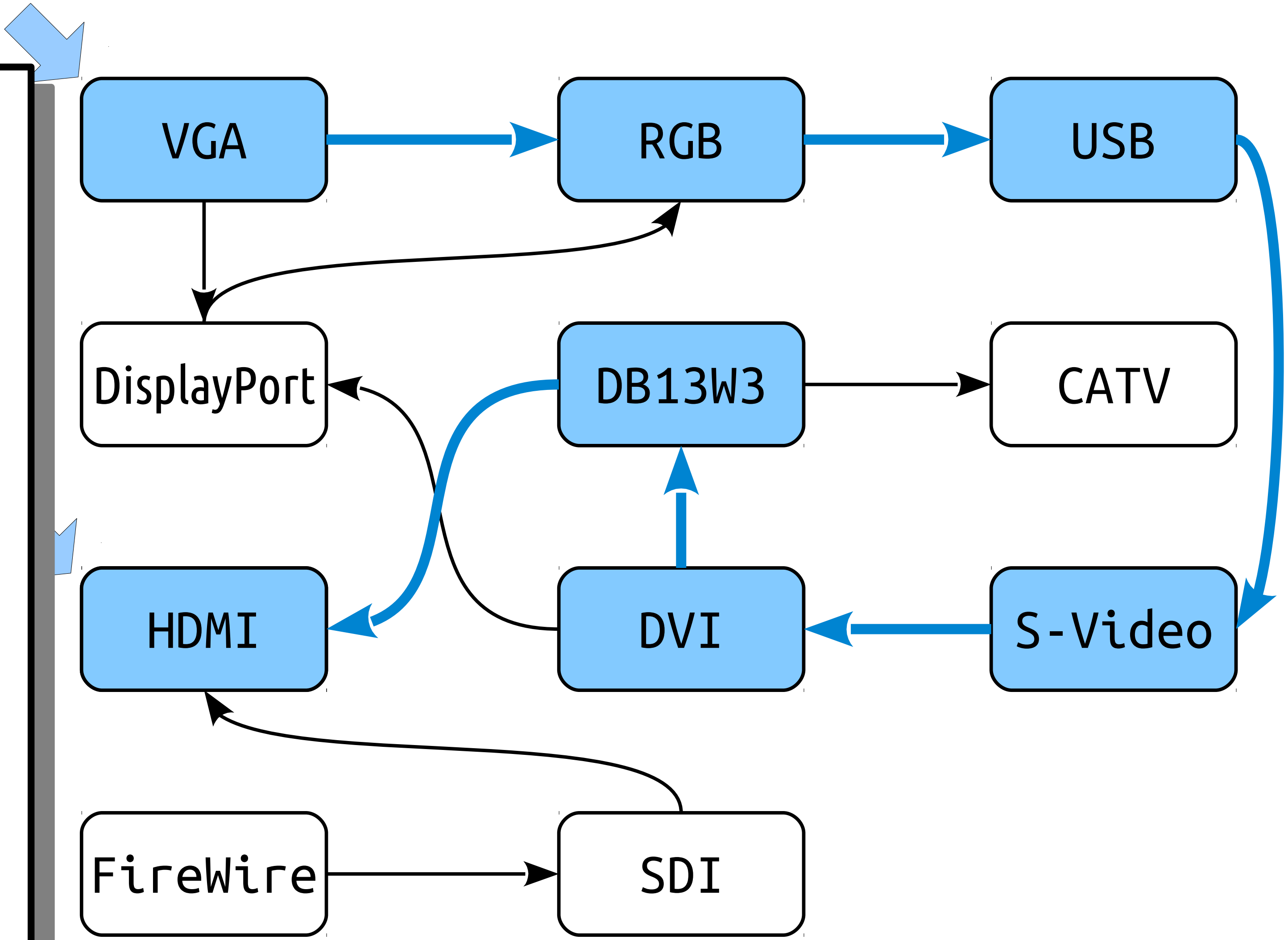


# Converter Conundrums



# 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



## In Pseudocode

```
boolean canPlugIn(List<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(transform(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(transform(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***.

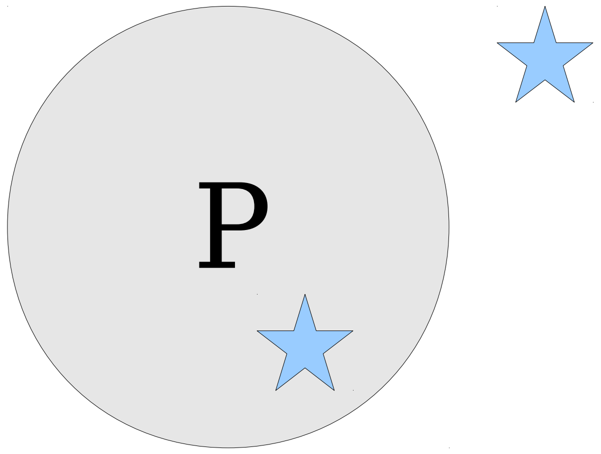
* Assuming that `transform` runs in polynomial time.

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

- This is a powerful general problem-solving technique. You'll see it a lot in CS161.
- Hypothetically speaking, maybe something like this would be useful to you on PS9?

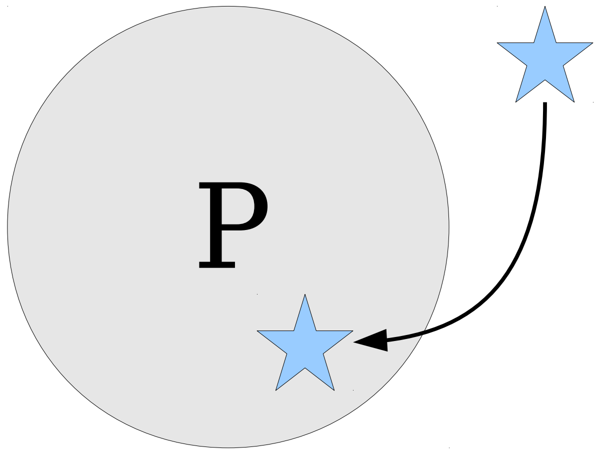
# Polynomial-Time Reductions

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



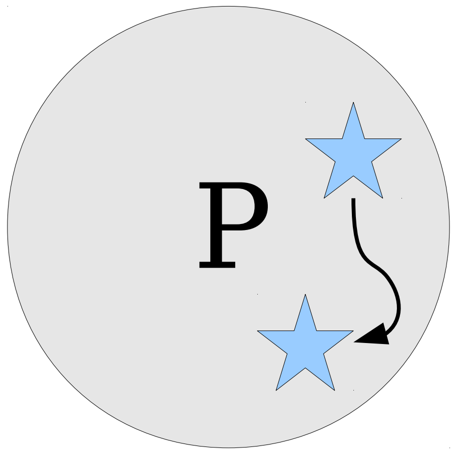
# Polynomial-Time Reductions

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



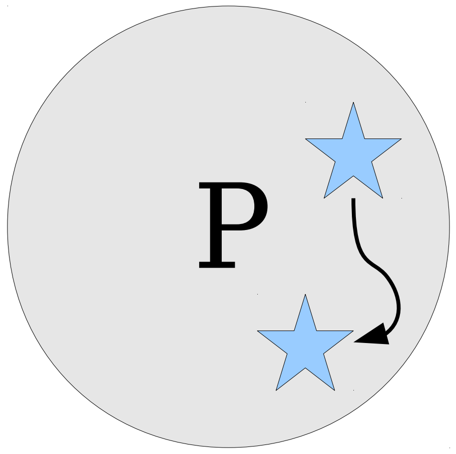
# Polynomial-Time Reductions

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



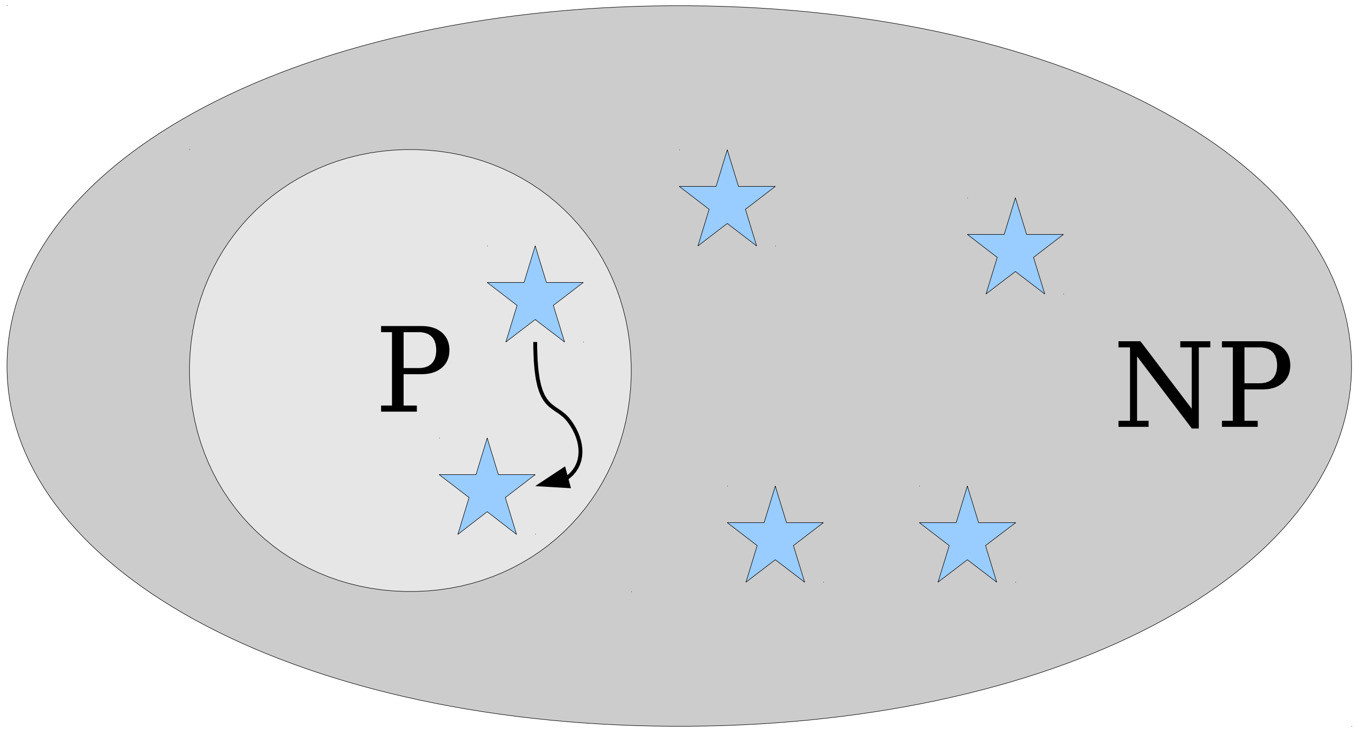
# Polynomial-Time Reductions

- 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 $A \in \mathbf{NP}$.



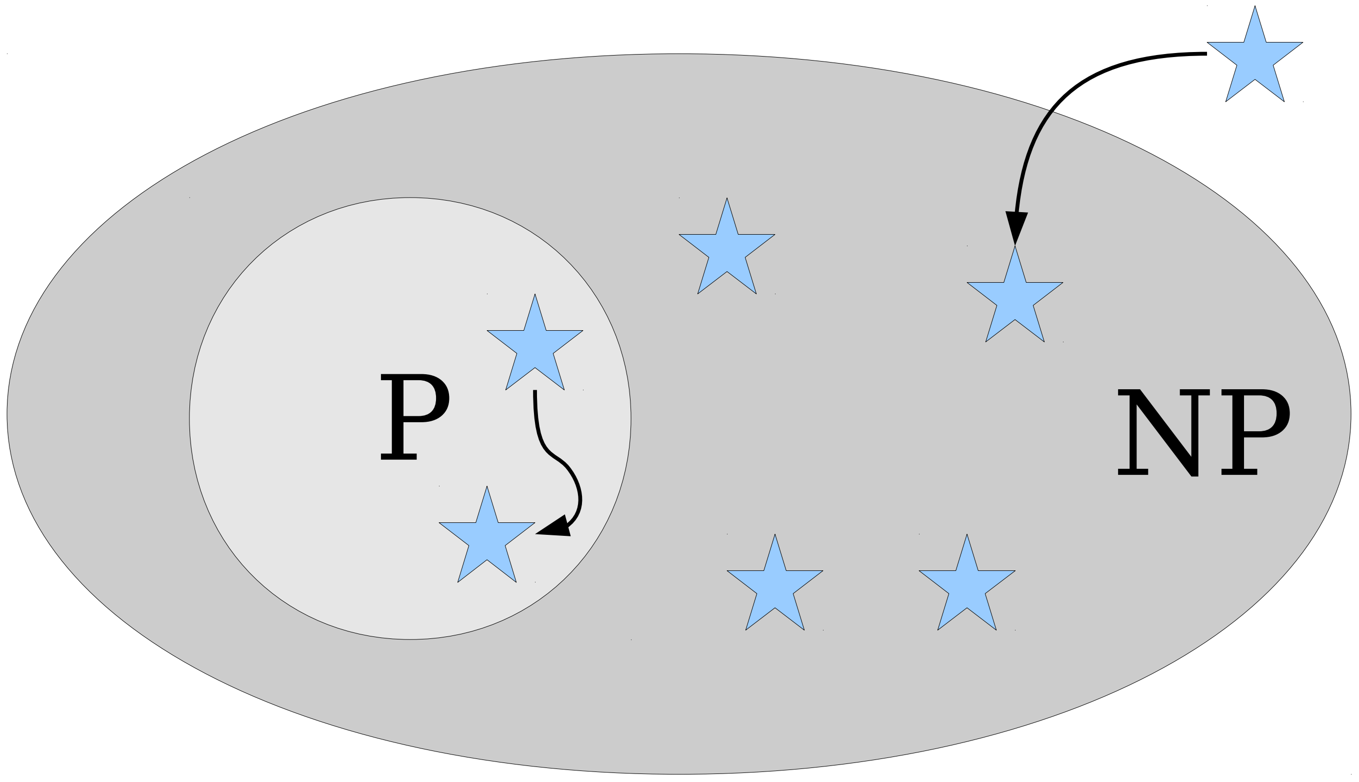
# Polynomial-Time Reductions

- 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 $A \in \mathbf{NP}$.



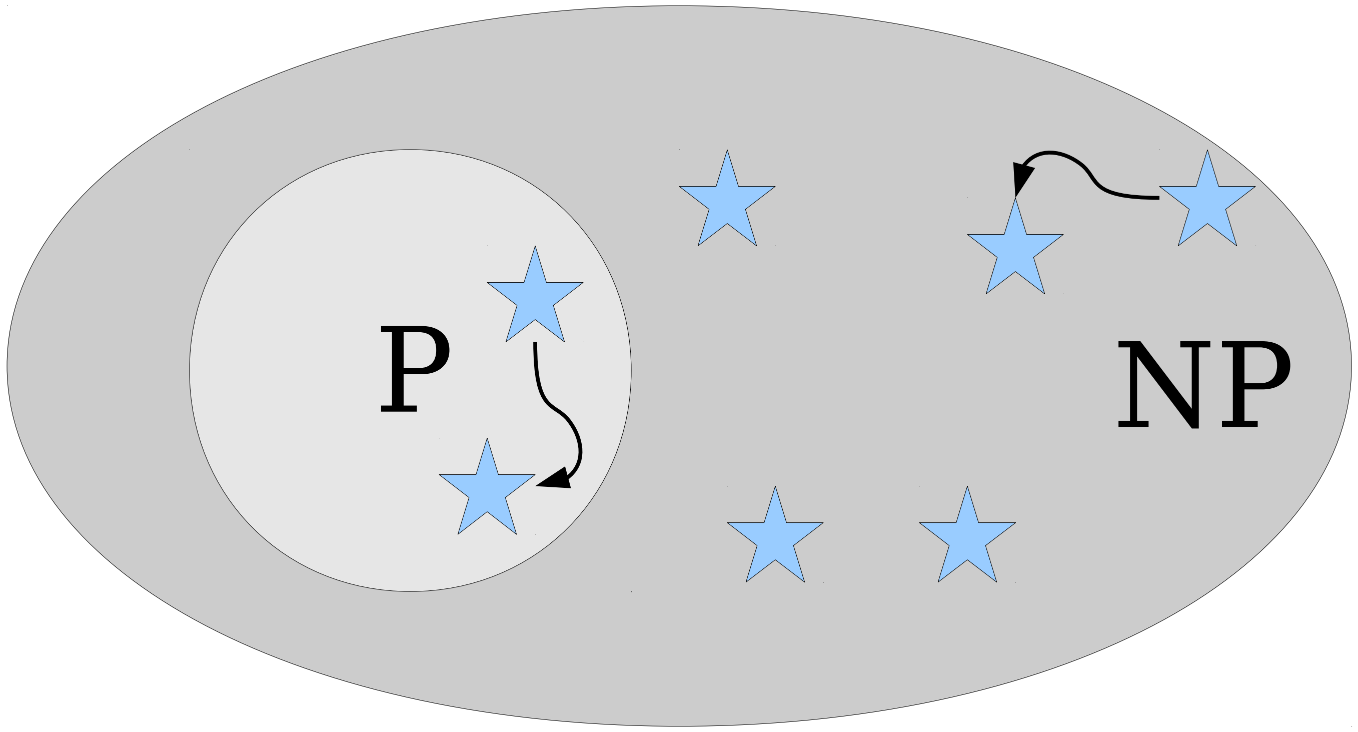
# Polynomial-Time Reductions

- 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 $A \in \mathbf{NP}$.



# 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!

# Camp Kesem

Camp Kesem
A Week of Magic
Apply by December 6th, 11:59 PM at
www.campkesemstanford.org/apply
Camp Kesem is a week-long summer camp for kids affected by a parent's cancer.
Check out our site for more info.

- Camp Kesem is a week-long summer camp and year-round community for kids who have a parent currently in treatment for cancer, in remission from cancer, or who has passed away from cancer. It's free for all families, and the week of camp is an opportunity for these kids to just be kids (in other words, Camp is *fun*!).
- New counselor applications for Camp Kesem 2019 are due ***tomorrow (Thursday)*** at midnight, and you can find more information at campkesemstanford.org/apply. Feel free to email your head TA Josh if you have any questions!

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

Your feedback makes a difference.

# Problem Set Nine

- Problem Set Nine is due this Friday at 2:30PM.
  - ***No late submissions can be accepted***. This is university policy – sorry!
- As always, if you have questions, stop by office hours or ask on Piazza.

## Final Exam Logistics

- Our final exam is Monday, December 10th from 3:30PM – 6:30PM. Locations are divvied up by last (family) name:
  - `A-L`: Go to ***Nvidia Auditorium***.
  - `M-Z`: Go to ***Cubberley Auditorium***.
- The exam is cumulative. You're responsible for topics from PS0 – PS9 and all of the lectures.
- As with the midterms, the exam is closed-book, closed-computer, and limited-note. You can bring one double-sided sheet of 8.5" × 11" notes with you to the exam, decorated any way you'd like.
- Students with OAE accommodations: Josh should have reached out to you at this point to set up alternate exams. Contact us ASAP if you didn't hear from us.

# Preparing for the Final

- On the course website you'll find
  - ***seven*** practice final exams, which are all real exams with minor modifications, with solutions, and
  - a giant set of practice problems (EPP3), with solutions.
- Our recommendation: Look back over the exams and problem sets and redo any problems that you didn't really get the first time around.
- Keep the TAs in the loop: stop by office hours to have them review your answers and offer feedback.

## Exam Review Sessions

- Your amazing TAs will be holding two review sessions this weekend:

  **Saturday, 1:00PM – 3:00PM in 200-034**

  **Saturday, 4:00PM – 6:00PM in 200-034**

- SCPD students: Julian will be holding an online review session on Saturday from 3:00PM – 5:00PM.

## Your Questions

“What do you think about the relative importance of talent vs hard work?”

Hard work is so much more valuable than talent. The most successful people I know are that way because they've worked hard and strategically to get where they are.

I'd qualify this by saying that hard work is less important than strategic work. You need to make sure to focus your efforts in places and areas where they matter.

# “If you could teach a non-CS, non-math class, what would it be?”

Perhaps this doesn't fully answer the question, but I'd love to co-teach a class called "Truth" with folks from philosophy, law, religious studies, sociology, etc. where we investigate what the concept of "truth" means. I'd talk about mathematical truth and how it works, and someone else would talk about what legal truth is, and we'd get voices about religious truths, how people decide what they consider true, how societies arrive at conclusions of what's true, etc.

Lest you think this is a political statement, I've had this idea since at least 2014. ☺

## “have you always been such a confident teacher/public speaker? what have you done to improve?”

No, not at all. I used to be absolutely terrified of messing things up in front of people, so I used to memorize almost everything I was going to say. Then I realized that when I did mess up nothing bad actually happened and if anything it helped lead the discussion in a productive way.

Pro tip: in many cases, public speaking is about advocating on behalf of something. Focus on doing the best advocacy you can rather than worrying about how you yourself are coming across.

"Some say that learning CS theory like the contents of CS103 is an inefficient use of time for those pursuing non-research careers. What's your opinion?"

"If we get a B- in this class, would you recommend repeating the class or studying on our own to fill the gaps?"

"What are your top 3 book recommendations?"

"Why is the Collatz Conjecture difficult to prove? It seems like a mathematician definitely should have cracked this by now."

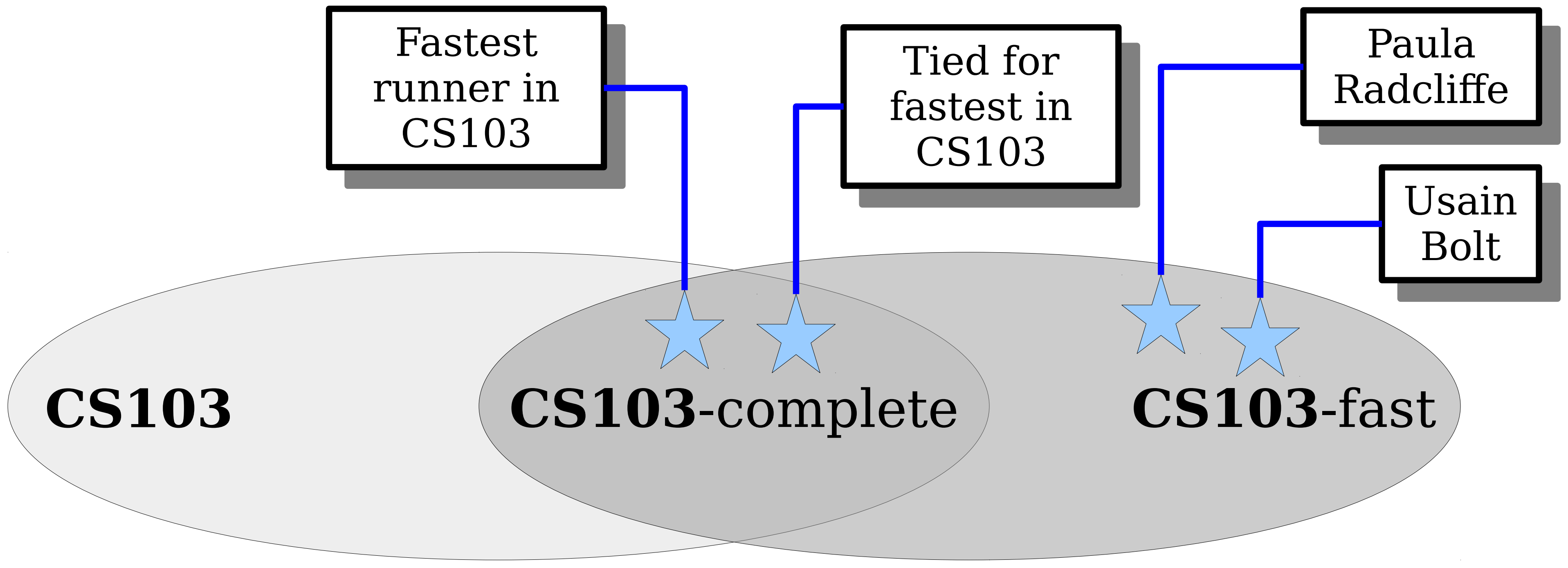
"What advice do you have for leading a happy and fulfilling life?"

"…"

These are great questions. Ask me next time!

Back to CS103!

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



For people $A$ and $B$, we say ***A* ≤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

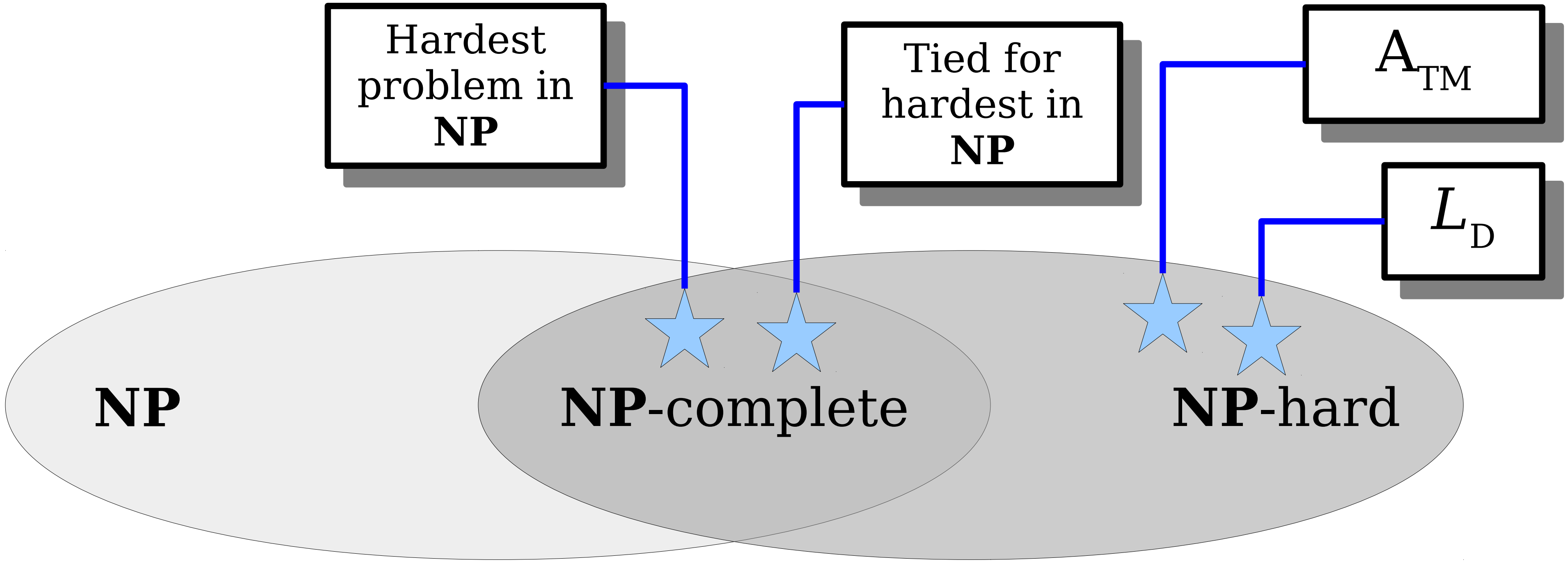
$\forall A \in$ **CS103**. $A \leq_r P$.

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

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

$P \in$ **CS103** and $P$ is **CS103**-fast.

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



For languages $A$ and $B$, we say $\boldsymbol{A \leq_p B}$ 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 $\mathbf{P} \overset{?}{=} \mathbf{NP}$ question.

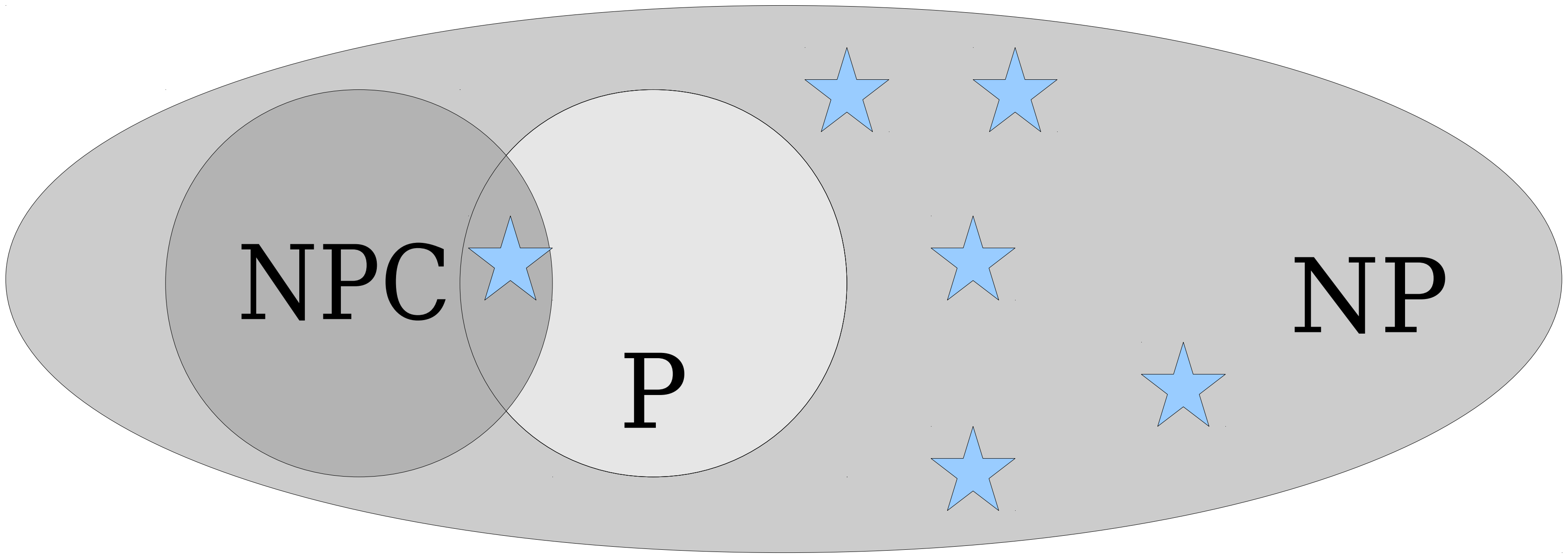
## The Tantalizing Truth

***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.

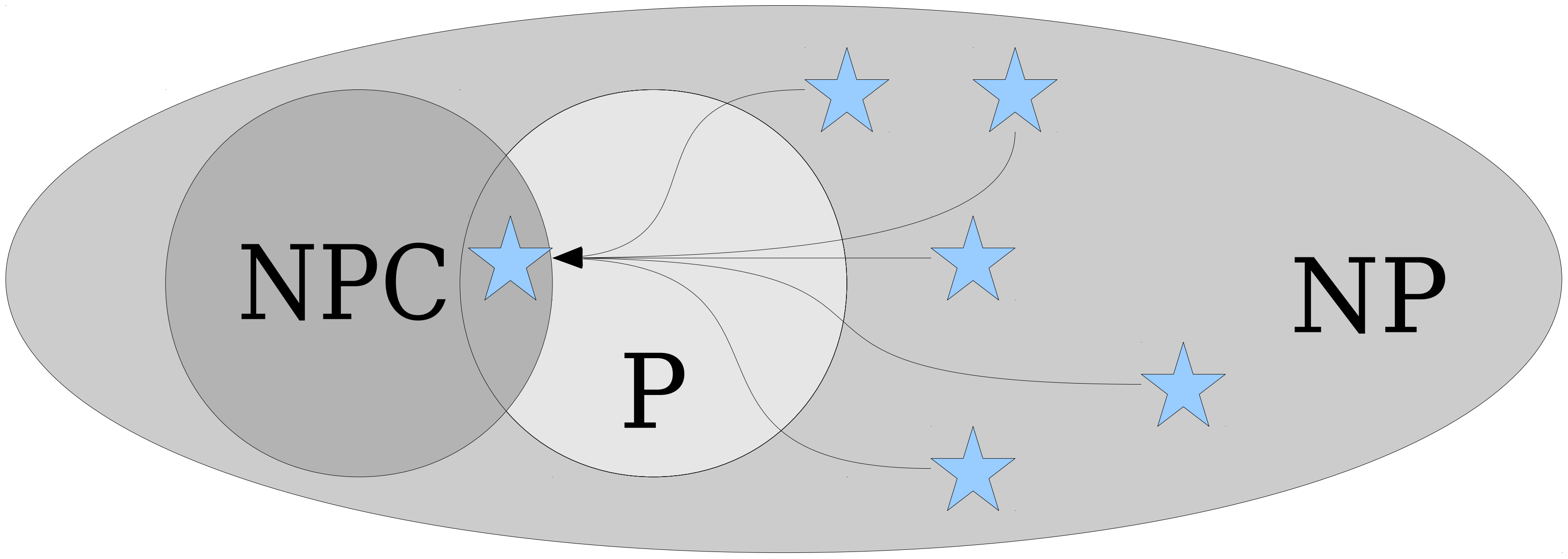
## The Tantalizing Truth

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



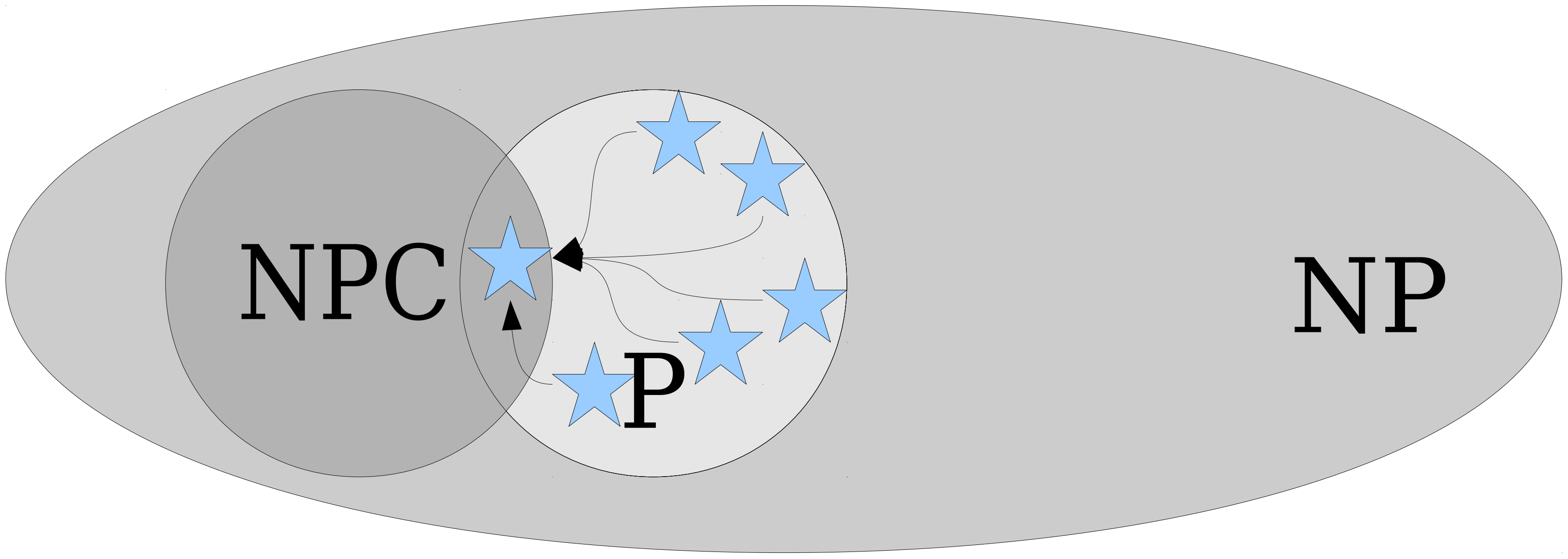
# The Tantalizing Truth

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



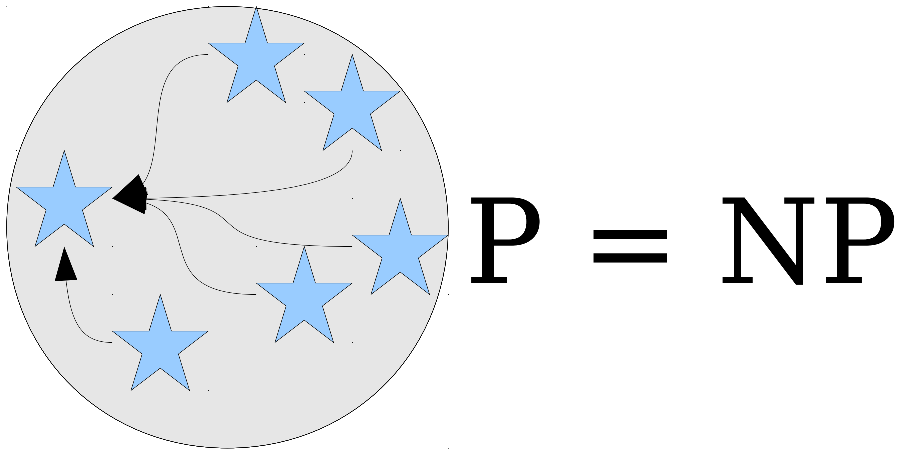
# The Tantalizing Truth

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



# The Tantalizing Truth

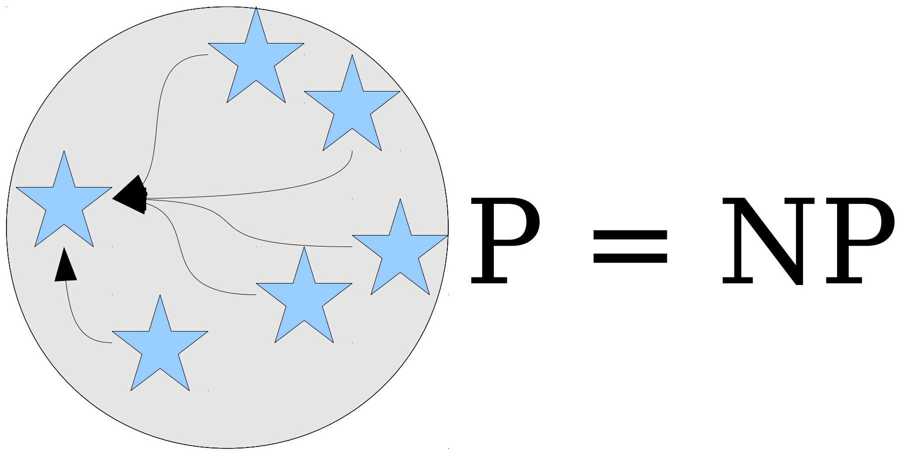
***Theorem:*** If *any* **NP**-complete language is in **P**, then **P** = **NP**.



# The Tantalizing Truth

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

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



# The Tantalizing Truth

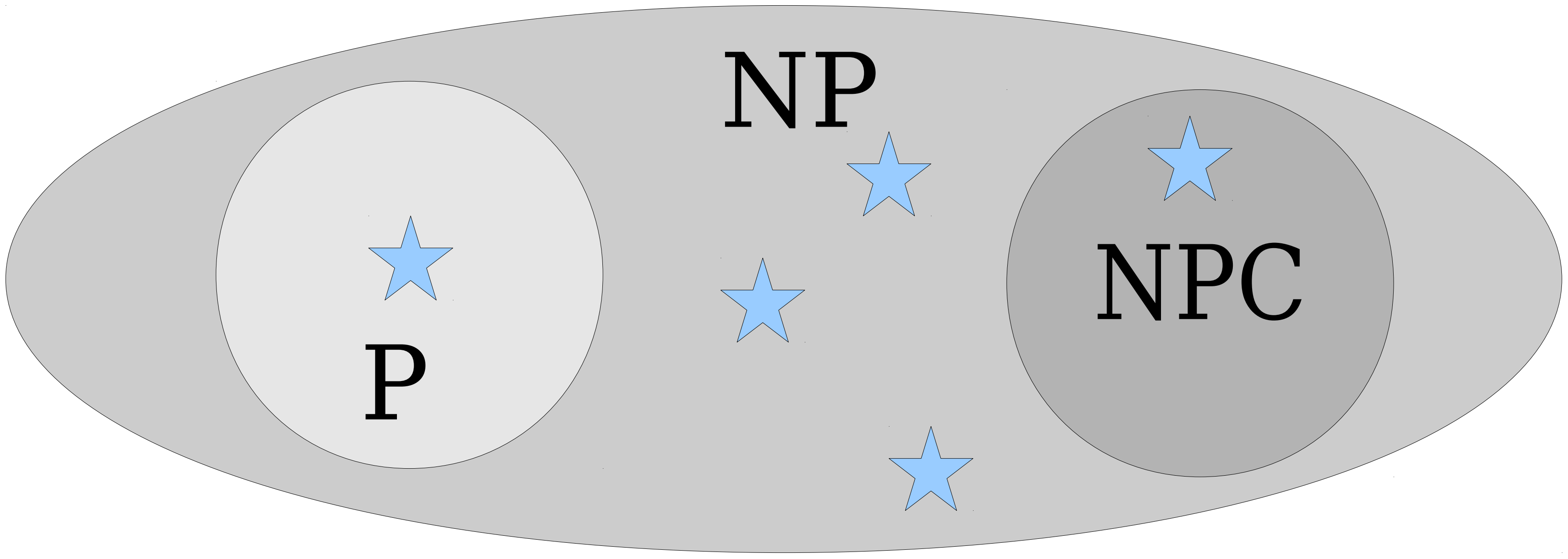
***Theorem:*** If *any* **NP**-complete language is not in **P**, then **P** ≠ **NP**.

***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** ≠ **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?*

# 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:

  ***SAT*** **= { ⟨φ⟩ | φ is a satisfiable PL formula }**

***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: have the certificate be a 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 φ($w$) 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 $\mathbf{P} \overset{?}{=} \mathbf{NP}$ is equivalent to just figuring out how hard SAT is.

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

- 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 survive. *(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 $\mathbf{P} \overset{?}{=} \mathbf{NP}$ is resolved?

## Next Time

- ***The Big Picture***
- ***Where to Go from Here***
- ***A Final "Your Questions"***
- ***Parting Words!***