Hard problems: P vs. NP

CS16: Introduction to Data Structures & Algorithms

Summer 2021

Garage Sale Optimization

- Imagine you're walking home from work and see a garage sale, selling various items (a TV, a chair, a dresser, …)
- The prices are pretty good! You happen to know that you could resell many of the items and make a profit ($10, $5, $20, …)
- The items are pretty heavy (5 lbs, 10 lbs, 15 lbs, …) so you can't really transport them without a car
- You have a friend with a car that can carry 40 kg without bottoming out, but it's only worth bringing the car if you can make at least $50
- Do you call up your friend?

Garage Sale Optimization

- Imagine you're walking home from work and see a garage sale, selling various items (numbered 1, 2, 3, …)
- The prices are pretty good! You happen to know that you could resell many of the items and make a profit p_1, p_2, p_3
- The items are pretty heavy (w_1, w_2, w_3) so you can't really transport them without a car
- You have a friend with a car that can carry weight W, but it's only worth bringing the car if you can make at least X profit
- Do you call up your friend?

Garage Sale Optimization

- Is there some subset S of items such that

$$\sum_{i \in S} w_i < W$$

and

$$\sum_{i \in S} p_i > X$$

Before we solve the problem…

- How can we check to see if a solution is valid?

Before we solve the problem…

- How can we check to see if a solution is valid?
    - Load the items into the car, see if it bottoms out, sell the items, see how much money you make

Before we solve the problem…

- How can we check to see if a solution is valid?
  - Load the items into the car, see if it bottoms out, sell the items, see how much money you make
  - Or, equivalently: just add up the weights and profits and see if they check out

Before we solve the problem…

- How can we check to see if a solution is valid?
    - Load the items into the car, see if it bottoms out, sell the items, see how much money you make
    - Or, equivalently: just add up the weights and profits and see if they check out
- What's the running time of this check if there are `n` total items?

Before we solve the problem…

- How can we check to see if a solution is valid?
  - Load the items into the car, see if it bottoms out, sell the items, see how much money you make
  - Or, equivalently: just add up the weights and profits and see if they check out
- What's the running time of this check if there are `n` total items?
  - `O(n)`, since we're adding up at most `n` numbers twice

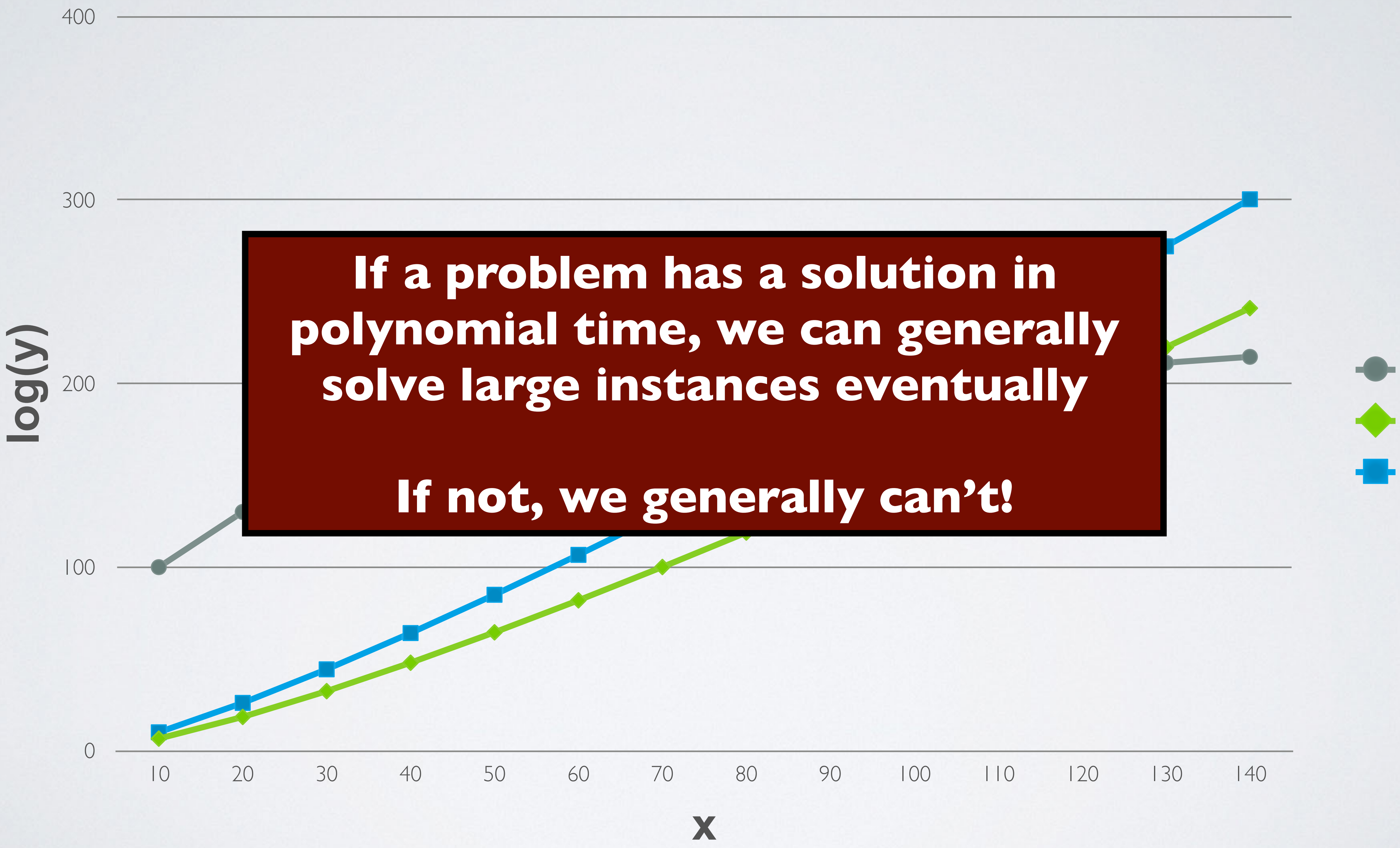
Before we solve the problem…

- How can we check to see if a solution is valid?
  - Load the items into the car, see if it bottoms out, sell the items, see how much money you make
  - Or, equivalently: just add up the weights and profits and see if they check out
- What's the running time of this check if there are `n` tota
  - `O(n)`, si ers twice

"Polynomial time"
O(n^c) for some c

not O(2^n), O(n!), etc.

Polynomial time

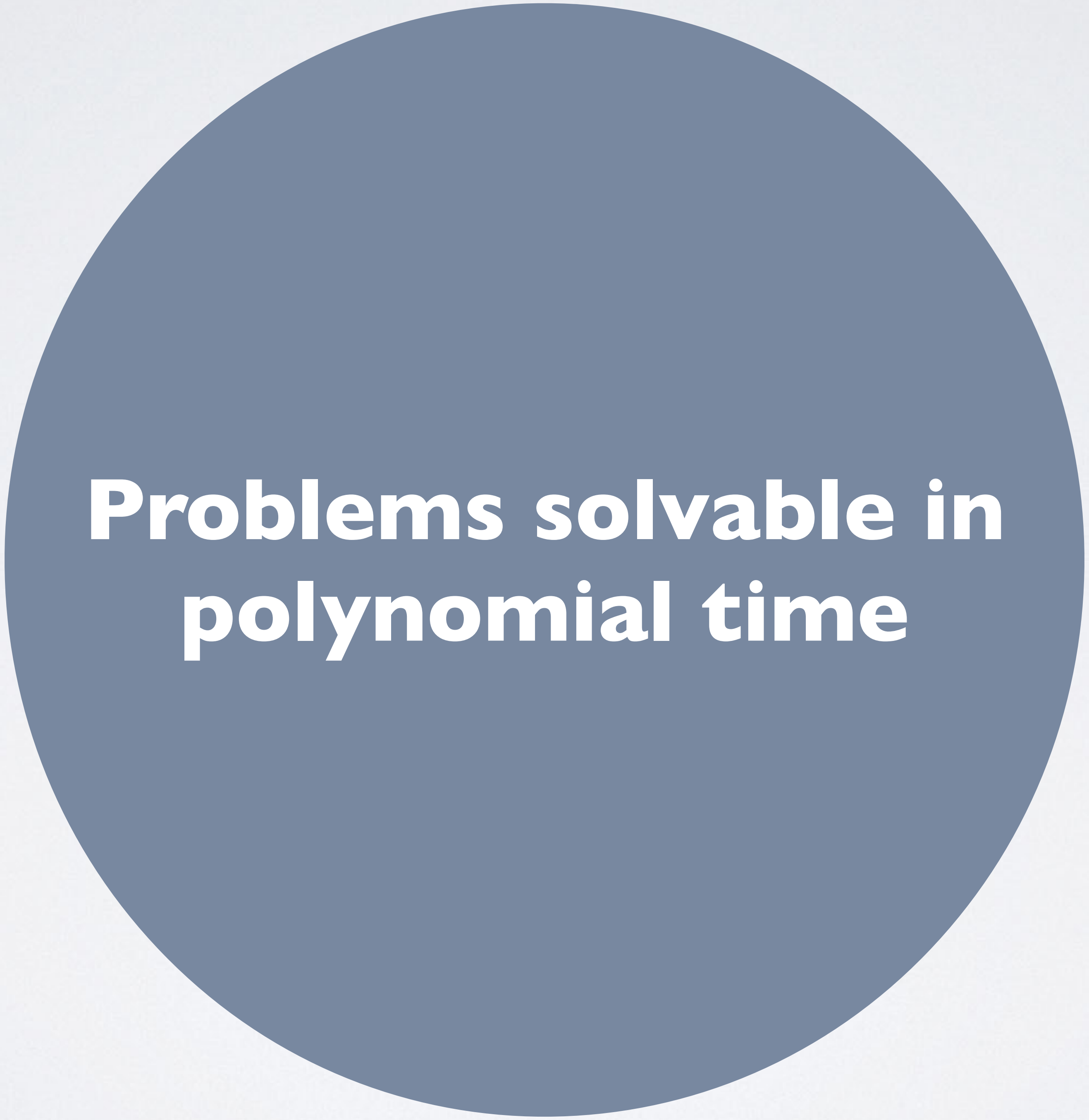


If a problem has a solution in polynomial time, we can generally solve large instances eventually

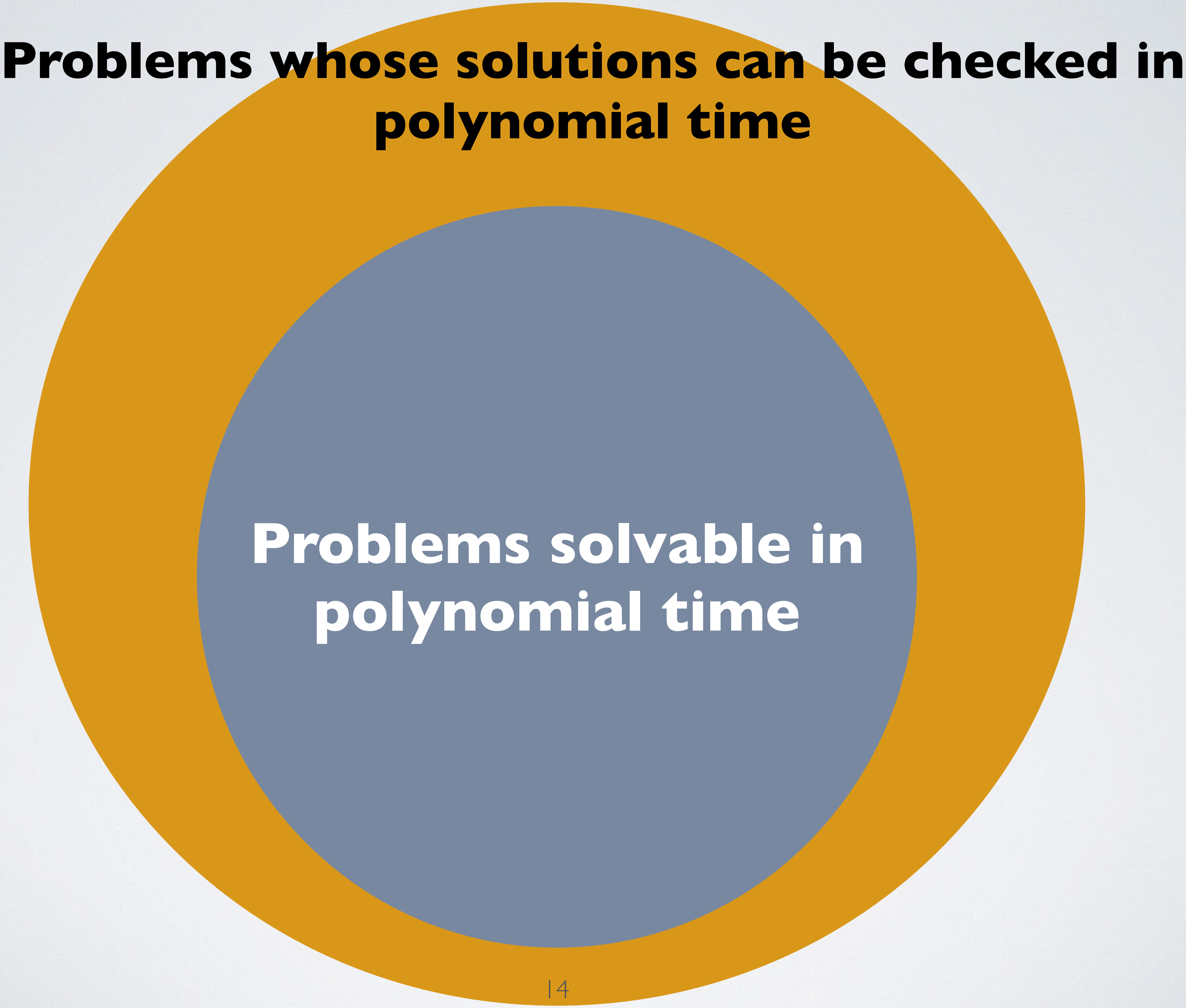
If not, we generally can’t!

Before we solve the problem…

- So, we can check solutions in polynomial time
- Does this mean we can *find* a solution in polynomial time?
- …i.e., is there guaranteed to be an O(n^c) algorithm to solve the Garage Sale Profit Problem?

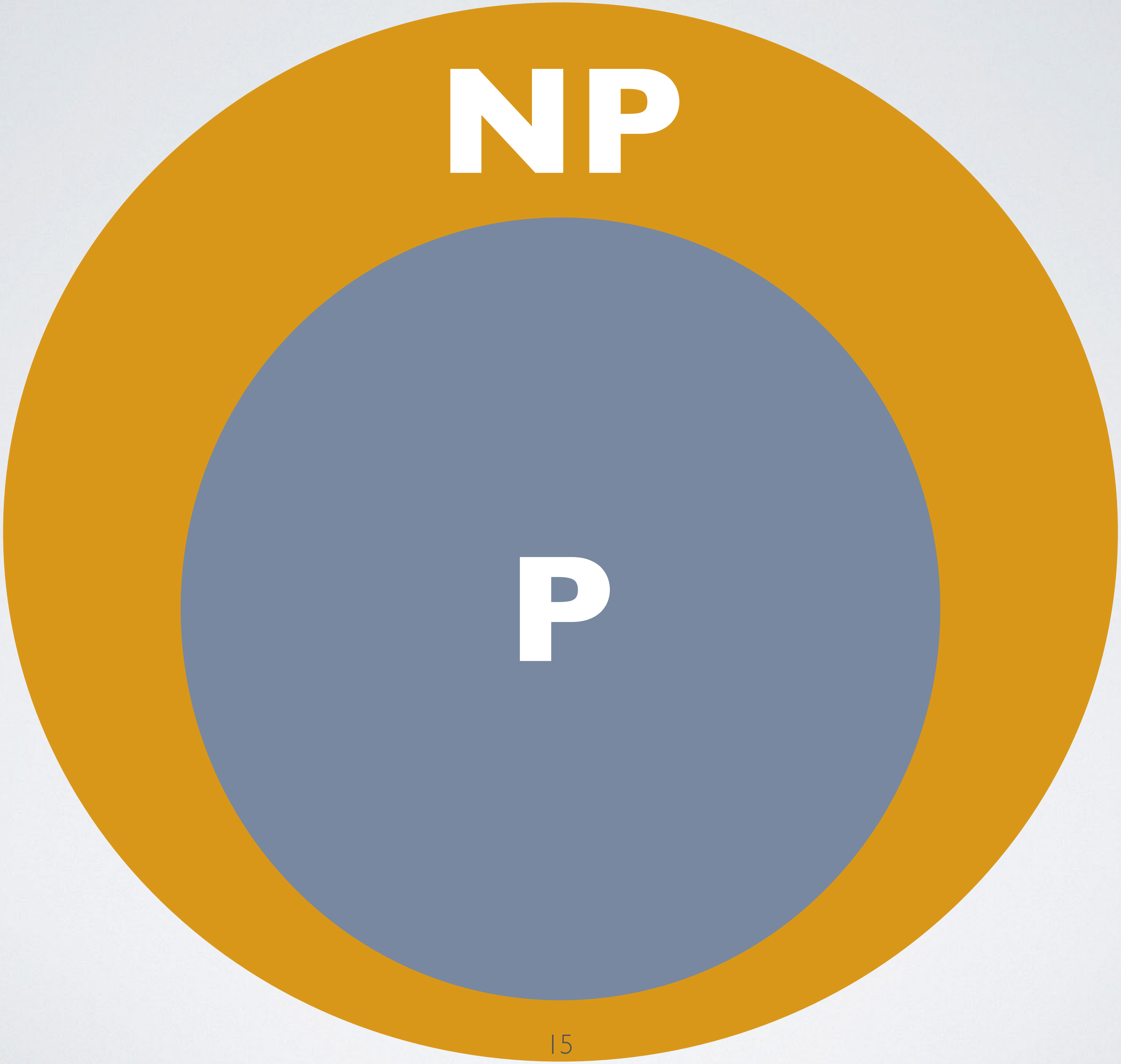


Problems solvable in polynomial time



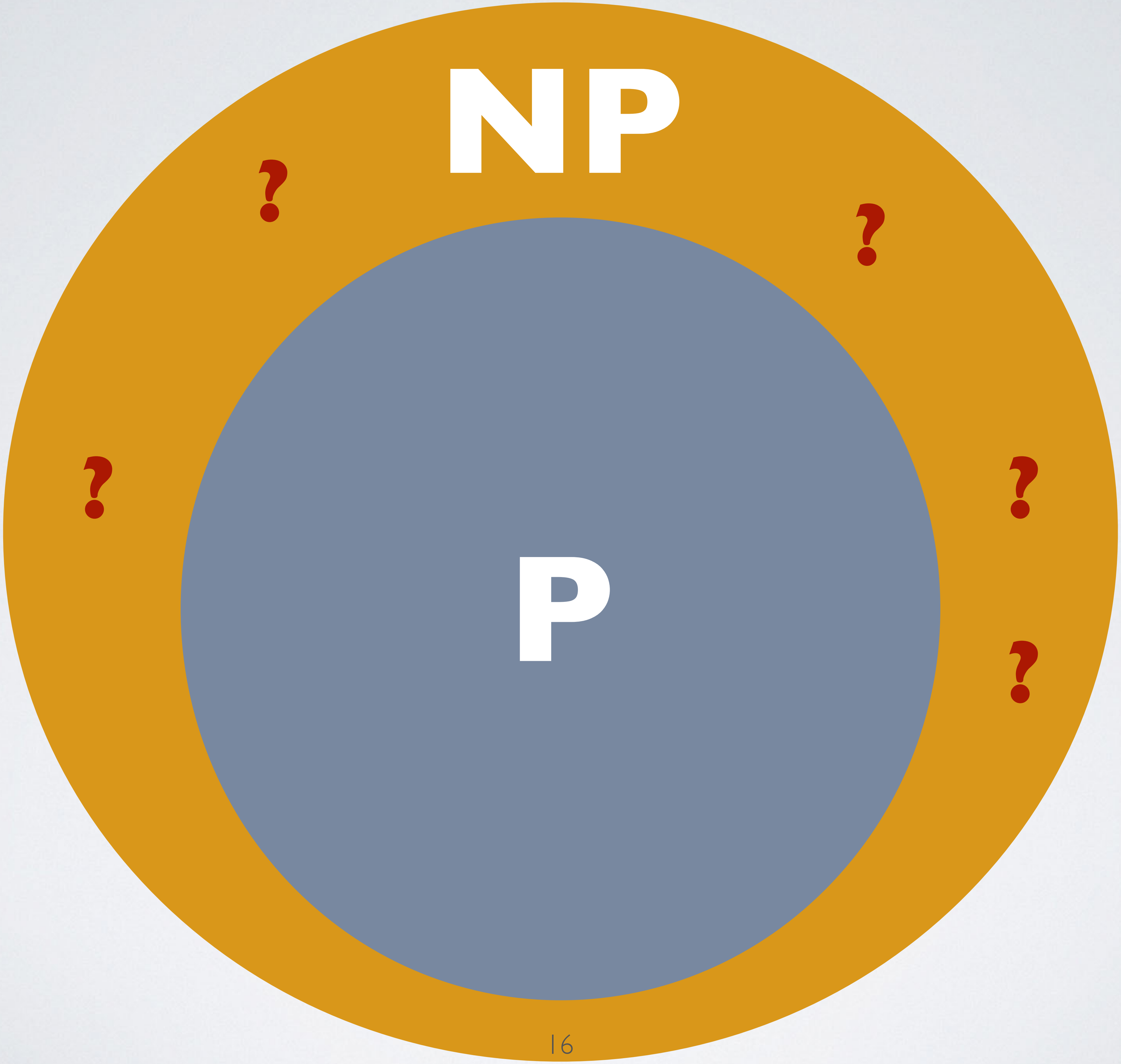
Problems whose solutions can be checked in polynomial time

Problems solvable in polynomial time



NP

P


NP
?
?
?
?
?
P

P vs. NP

- Does P = NP?
  - *No one knows!*
  - Most famous unsolved problem in computer science
  - People have been trying to prove that either P = NP or P != NP for 50 years
- Most computer scientists believe that P != NP
  - Meaning that being able to efficiently **check a solution** *doesn't* mean you can efficiently **solve the problem**

P vs. NP implications

- One problem we know is in NP: factoring large numbers
  - Every number uniquely expressible as a product of primes
  - Factoring: find those primes
- Modern cryptography generally based on prime factorization
- If P = NP, online banking doesn't work any more!

Back to the Garage Sale

- How would we solve the Garage Sale Profit Problem?

Back to the Garage Sale

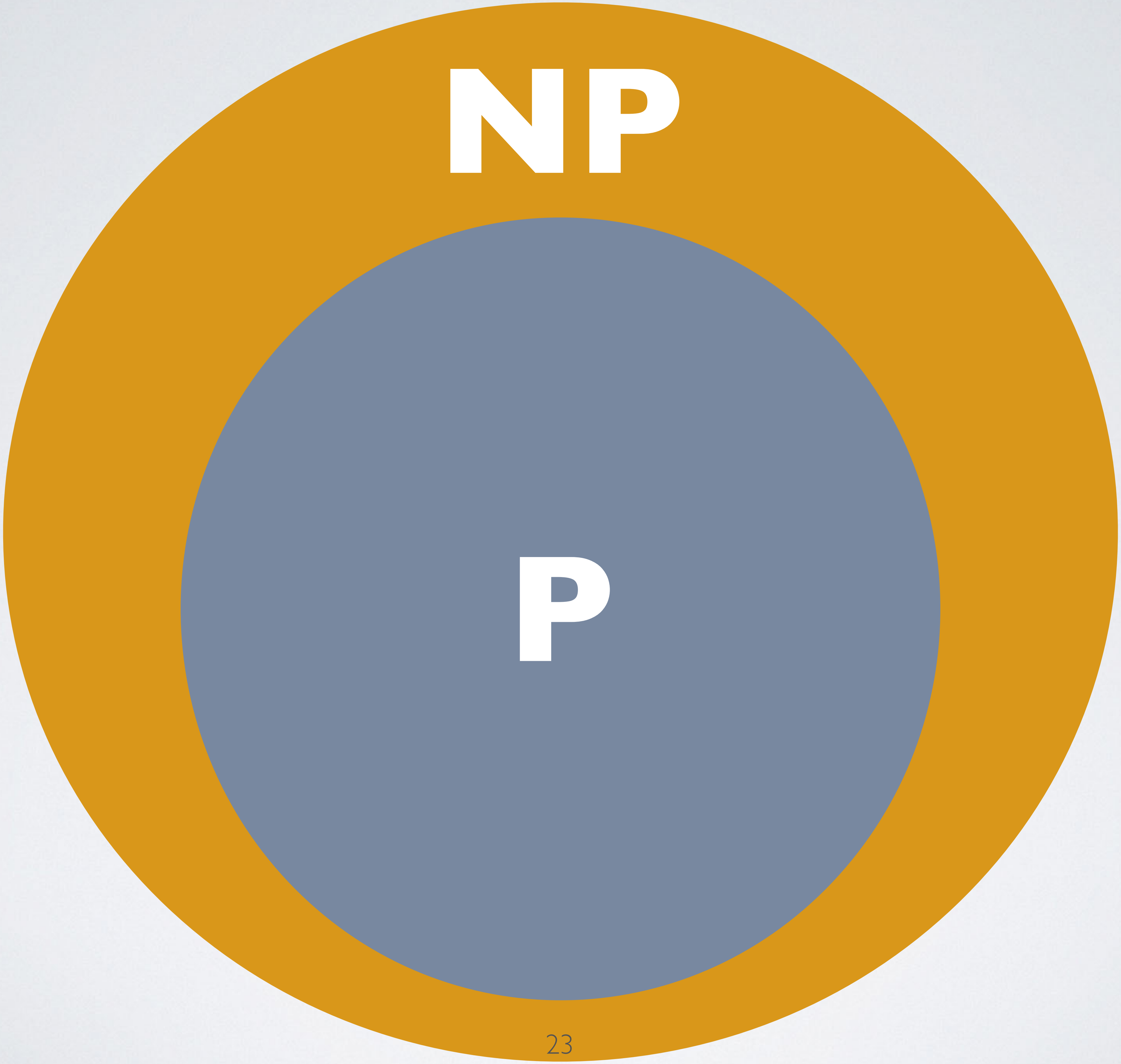
- How would we solve the Garage Sale Profit Problem?
- First try: brute force search
  - Try all combinations of items
  - Do any of them fit in the car and generate enough profit?

Can we do better?

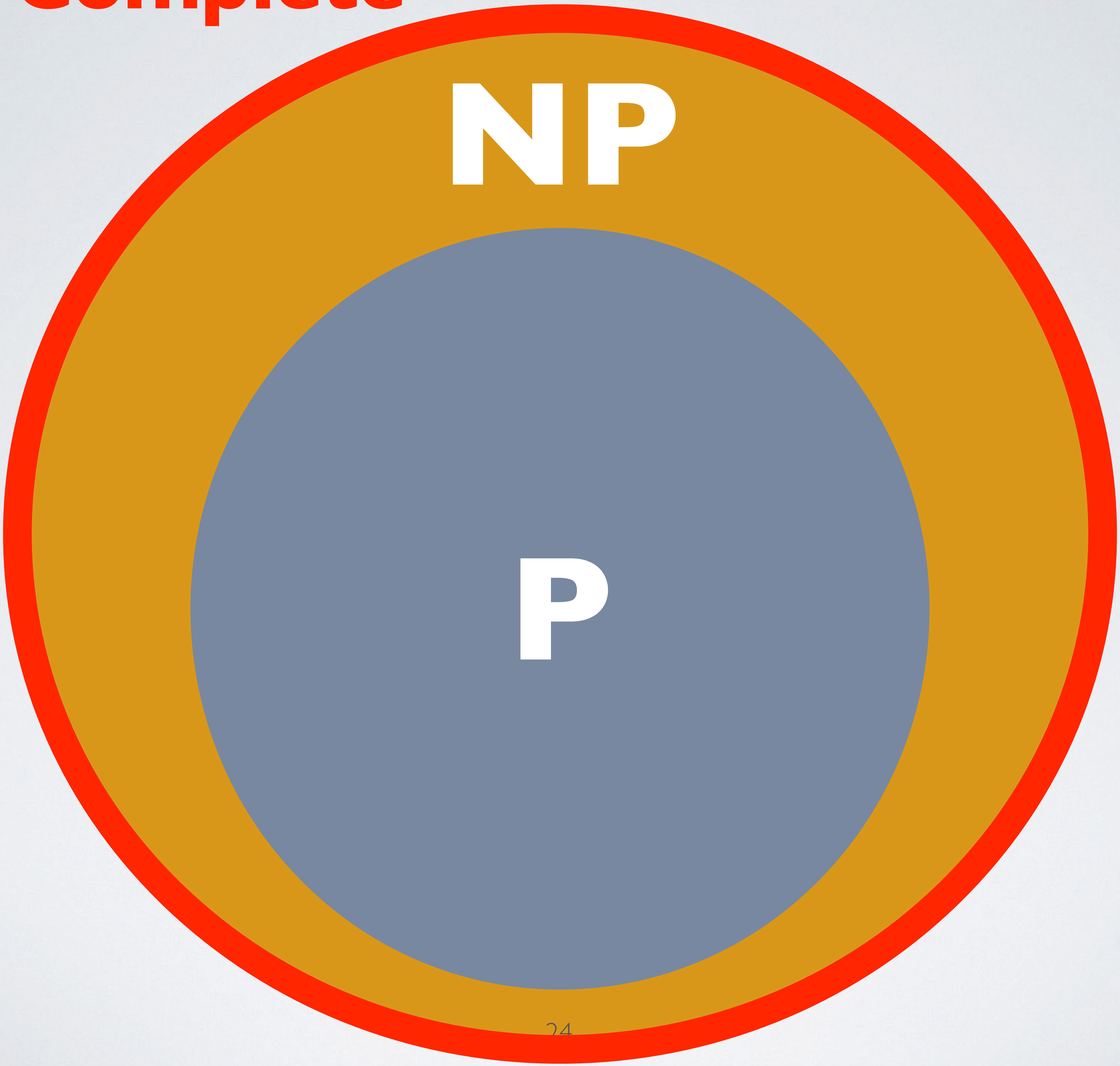
- Maybe!
- But….probably not
- *If* there is a polynomial-time solution to the Garage Sale Profit Problem (called the *knapsack problem* in the CS literature), **then P = NP**

Why????

- We can prove that, for **every** problem in NP
  - (problems whose solutions can be checked in polynomial time)
- Every instance of that problem can be *reduced* to an equivalent instance of the knapsack problem in polynomial time
- So if we can solve knapsack in polynomial time, we can also solve all problems in NP in polynomial time

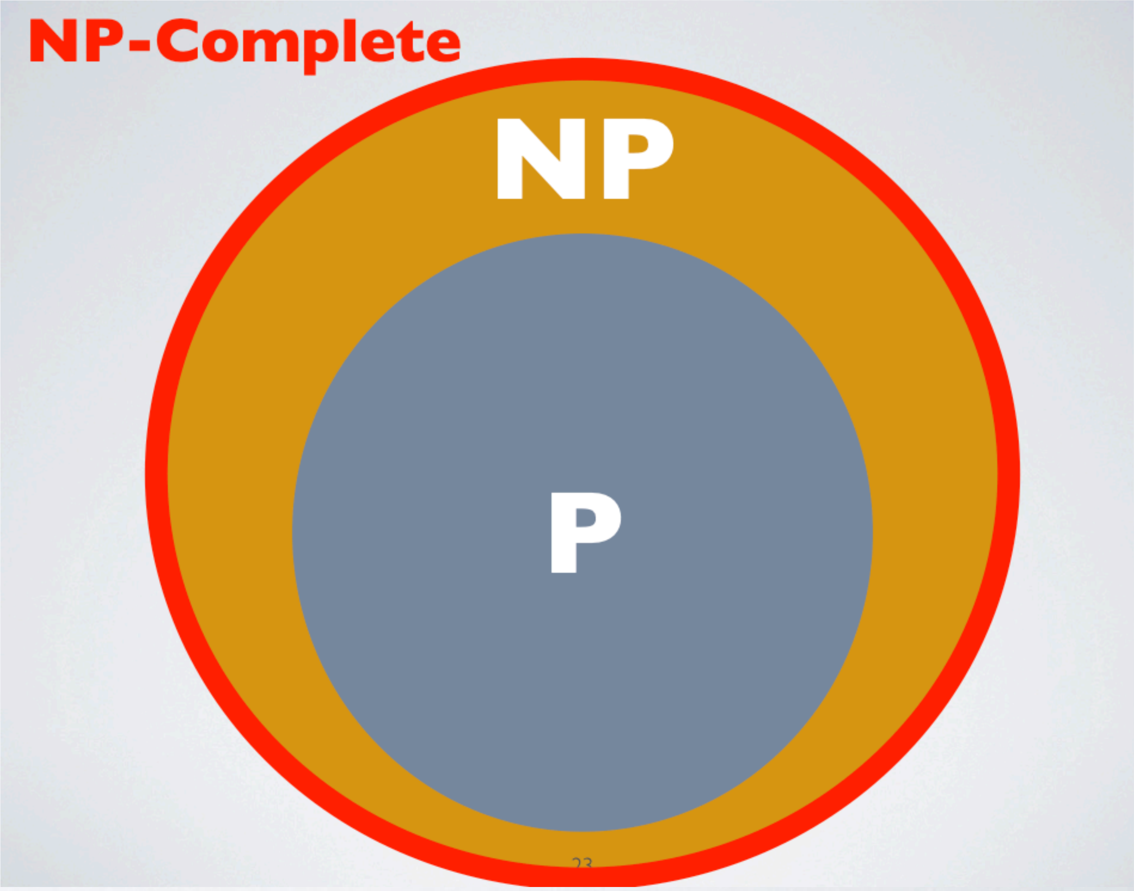

NP
P

NP-Complete



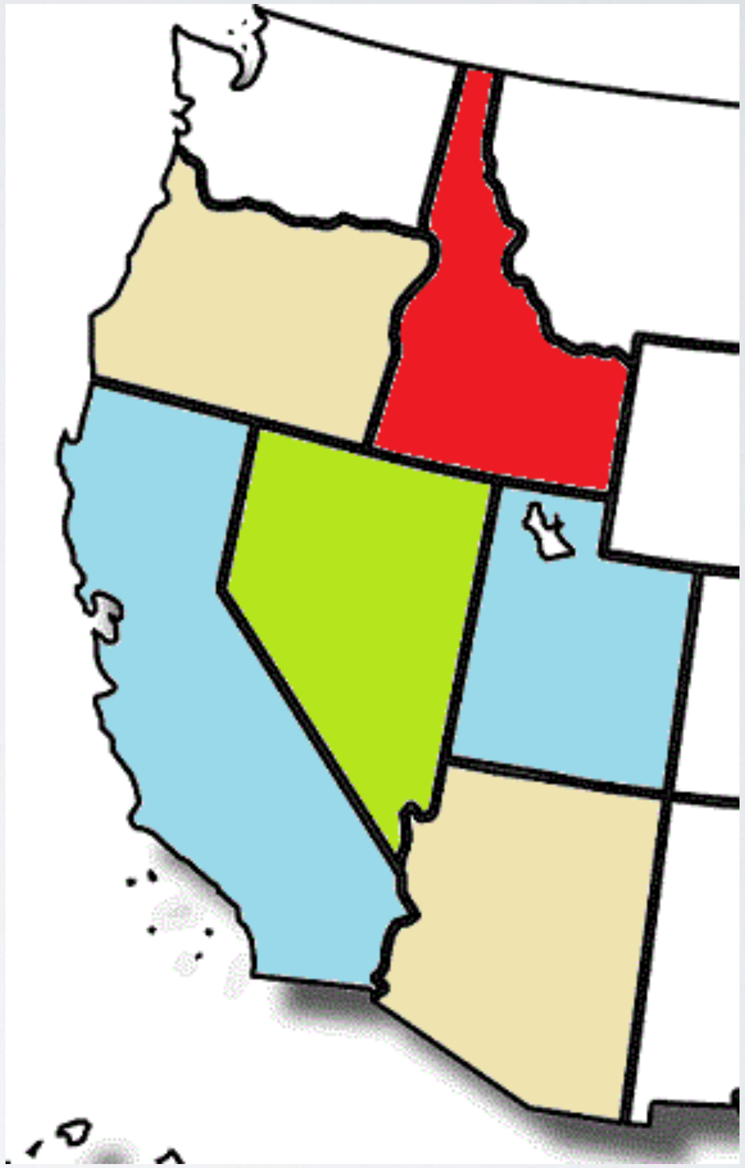
NP-complete problems

- In NP, and at least as hard as any other problem in NP
- *Many* NP-complete problems!



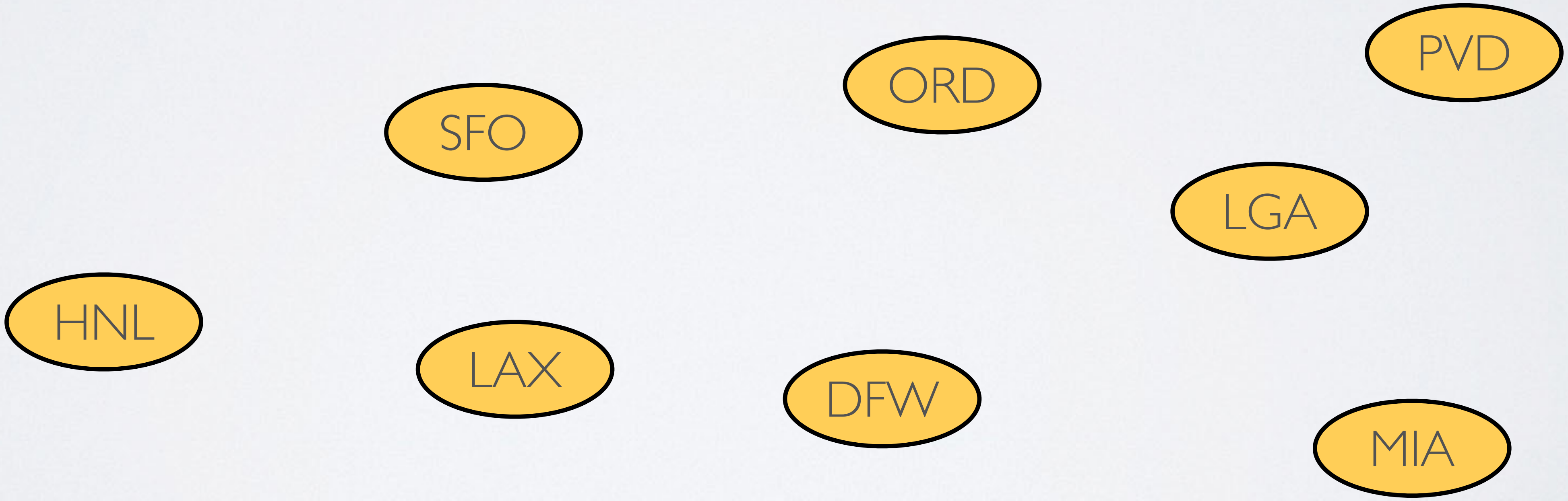
Map coloring

- Given a map of territories (e.g., states, countries, watersheds, etc.)
- Can we color each territory with only **three** colors?
- Neighboring territories can't have same color



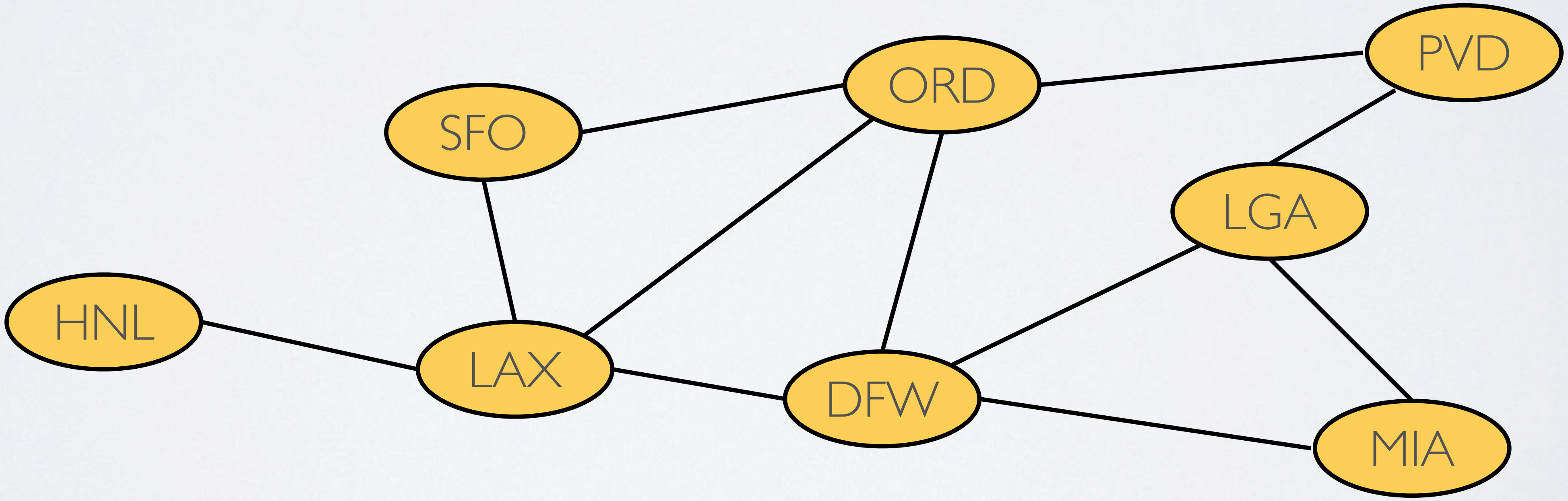
Traveling salesperson

- Given a list of cities, distances between each pair of cities, and a distance **`D`**
- Can I visit every city, traveling at most **`D`** in total?



Vertex cover

- Given a graph **G** and a number **k**
- Is there a set of at most **k** *vertices* of **G** such that each *edge* in the graph has an endpoint in the set?



Satisfiability

- Given a boolean formula like:
    - `(x || !y || z) && (!x || w || v)`
    - `x && (y || z) && !y && (!z || !x)`
- Is there some assignment of **`true`** and **`false`** to the variables such that the formula is true?

What do these problems have in common?

Let’s assume P != NP

- Is there anything we can do?
- Exponential time is SLOW
- Can we approximate solutions?

Approximating

- Some NP-complete problems amenable to approximation
    - Find an OK solution
    - Find a solution under certain conditions
    - etc.

Approximating at the garage sale

Approximating at the garage sale

- Sort items by $/lb
- Grab items until you can’t fit any more

Approximating at the garage sale

- Sort items by $/lb
- Grab items until you can't fit any more
- A version of this algorithm is guaranteed to find a solution provided that there's *some* way of combining items such that you make at least W * 2 profit
  - Not optimal!
  - But not terrible

Satisfiability

- No good approximations exist
- But many SAT problems that arise in practice can be solved quickly by modern solvers
  - Good heuristics
  - Very thorny problems don’t seem to come up very often

Satisfiability

- At the core of some automatic verification tools
- More on which next time!

If this topic seems cool…

- CSCI 1010
  - Lots of proofs about NP-completeness
  - Another very cool topic: undecidability
  - P vs. NP: are there problems such that we can’t write an *efficient* algorithm that solves them?
  - Undecidability: are there problems such that we can’t write *any* algorithm that solves them?

Summary

- There are interesting problems whose solutions can be *checked* quickly
    - We don't know whether the problems can be *solved* quickly
- If one can be solved quickly, all can be solved quickly
- Just because problems are hard doesn't mean they go away!
    - Some problems have good approximate or heuristic algorithms