CS 344: Design and Analysis of Computer Algorithms

(Spring 2022 — Sections 5,6,7,8)

Lecture 24:
Introduction to P vs NP

Efficient Algorithms: Polynomial Runtime

NOT Poly-time Algorithms

- ALL of this course has been about designing efficient algorithms:
  - How can we show a problem P can be solved in poly-time?
- What if we instead want to show that a problem P **cannot** be solved in poly-time?
- Why do we want to do that?
  - So we do not waste our time trying to design an efficient algorithm for P
  - So we can use P to design passwords, do cryptography, or create bitcoins,…

Decision Problems

Solving vs Verifying a Decision Problem

Verifier

- A verifier for a decision problem P with input x:
- The verifier specifies what type of a proof y it needs
  - The burden of finding the proof is NOT on the verifier
  - The only requirement is that:
    - If P(x) = YES, a valid proof y should always exist
    - If P(x) = NO, there is no valid proof
- Given x and y, the verifier should output if P(x) = YES or not (alternatively, verify if the "proof" is correct or not)

(Complexity) Classes
P & NP

Classes P and NP

- We use class to refer to a collection of problems

Classes P and NP

- We use class to refer to a collection of problems
- Class P:
  - ALL problems that can be solved in polynomial time
- Class NP:
  - ALL problems that can be verified in polynomial time

Classes P and NP

- We use class to refer to a collection of problems

Problems we can solve efficiently

- Class P:
  - ALL problems that can be solved in polynomial time
- Class NP:
  - ALL problems that can be verified in polynomial time

Problems we “hope” to be able to solve efficiently

Classes P and NP

- Clearly any problem in P is also in NP
  - If we can solve a problem in poly-time, we can definitely verify it in poly-time

Classes P and NP

- Clearly any problem in P is also in NP
  - If we can solve a problem in poly-time, we can definitely verify it in poly-time
- Big open question of Computer Science:

Is P=NP or not?

Classes P and NP

- Most researchers believe that $P \neq NP$ but we are nowhere close proving (even much weaker versions of) this

- Proving $P \neq NP$ is somewhat opposite of what we do in this course:

  - Instead of giving an efficient algorithm for a problem (what we did in this course), we want to show there is NO efficient algorithm for a problem

NP-Hard & NP-Complete problems

Plan

- We have a problem Q in NP
- We want to show that Q is impossible to solve in polynomial time
- We do not know how to do that

Plan

- We have a problem Q in NP
- We want to show that Q is impossible to solve in polynomial time
- We do not know how to do that
- Instead, we go for the next best thing:
  - Show that Q is really hard to solve in polynomial time

Plan

- We have a problem Q in NP
- We want to show that Q is impossible to solve in polynomial time
- We do not know how to do that
- Instead, we go for the next best thing:
  - Show that Q is really hard to solve in polynomial time
- **A simple approach:**
  - Show that if Q can be solved in polynomial time, then P = NP

Plan

- **A simple approach:**
  - Show that if Q can be solved in polynomial time, then P = NP
- How can we show such a thing?

Plan

- **A simple approach:**
  - Show that if Q can be solved in polynomial time, then P = NP
- How can we show such a thing? **Reductions!**
- Show that:
  - if there is a poly-time algorithm A for problem Q, then
  - we can use A in a black-box way to design a poly-time algorithm for every problem in NP

Plan

- **A simple approach:**
  - Show that if Q can be solved in polynomial time, then P = NP
- How can we show such a thing? Reductions!
- Show that:
  - if there is a poly-time algorithm A for problem Q, then
  - we can use A in a black-box way to design a poly-time algorithm for every problem in NP
- **Still NOT an easy plan:** we have to design infinitely many algorithms from A for every possible problem in NP

Plan

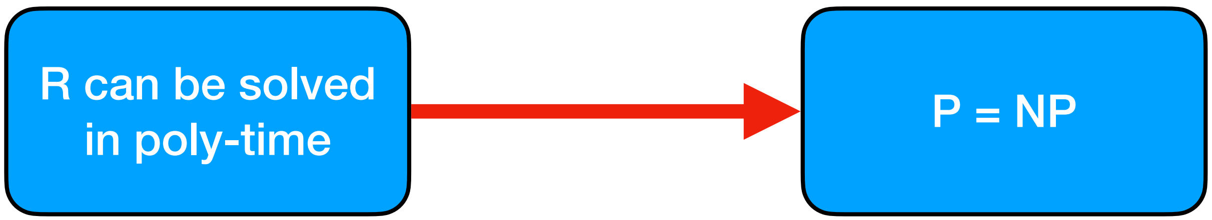
- **Still NOT an easy plan:** we have to design infinitely many algorithms from A for every possible problem in NP

- What if I tell you there is a single problem R such that if R can be solved in poly-time, then P=NP?

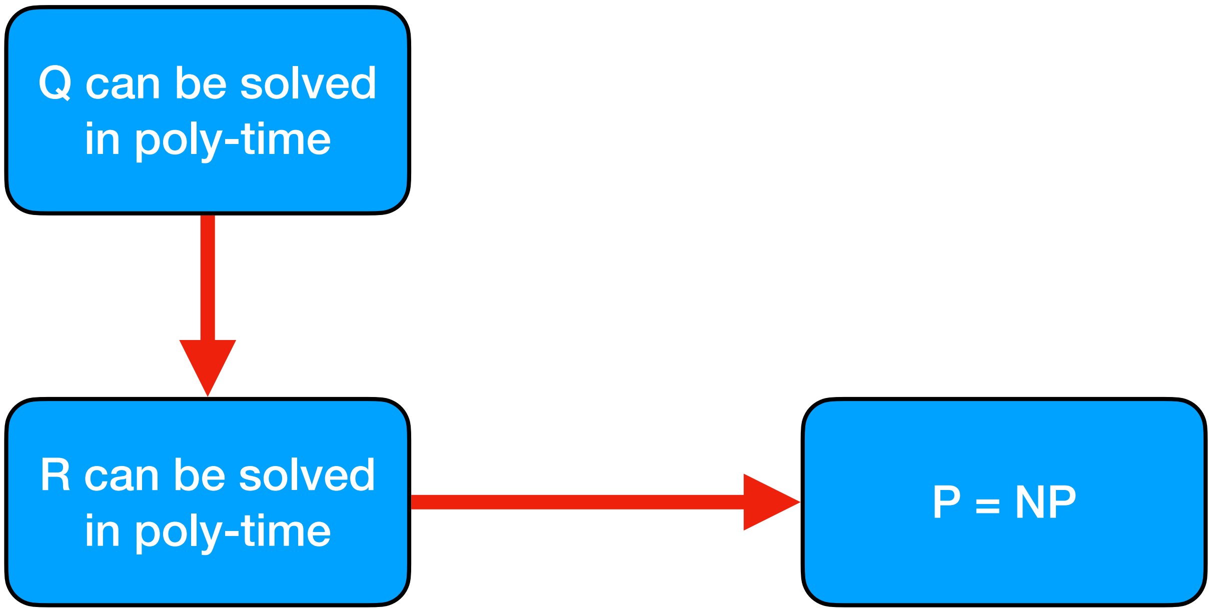
Plan

- **Still NOT an easy plan:** we have to design infinitely many algorithms from A for every possible problem in NP
- What if I tell you there is a single problem R such that if R can be solved in poly-time, then P=NP?
- Then we only need to do a reduction from R:
  - Show that the poly-time algorithm A for Q can be used to design a poly-time algorithm for R

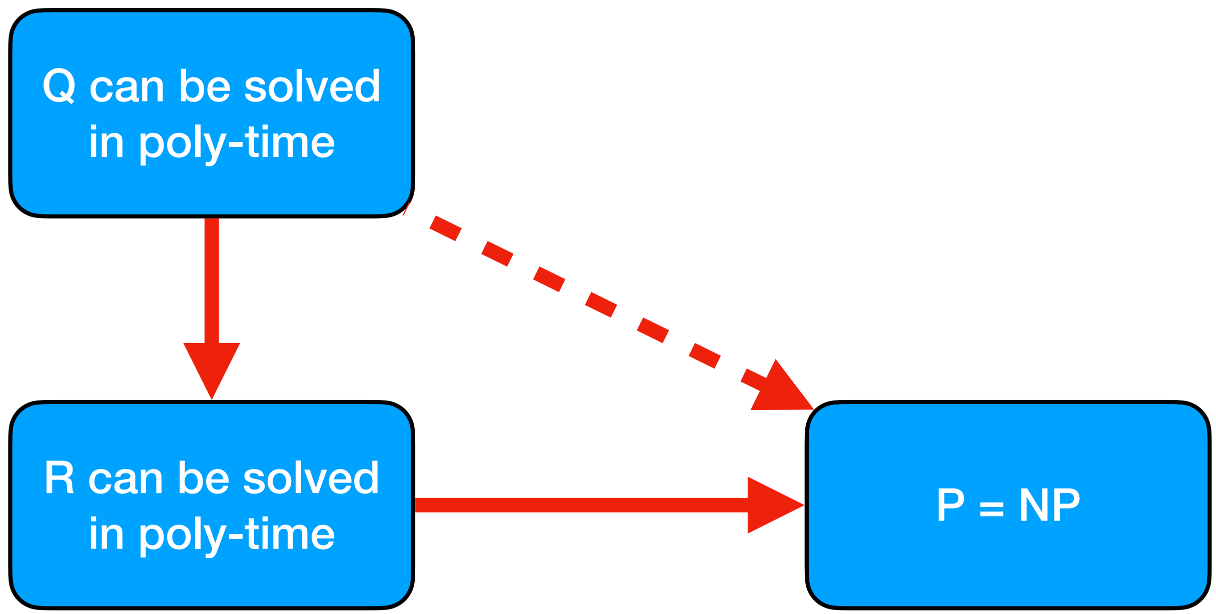
Plan



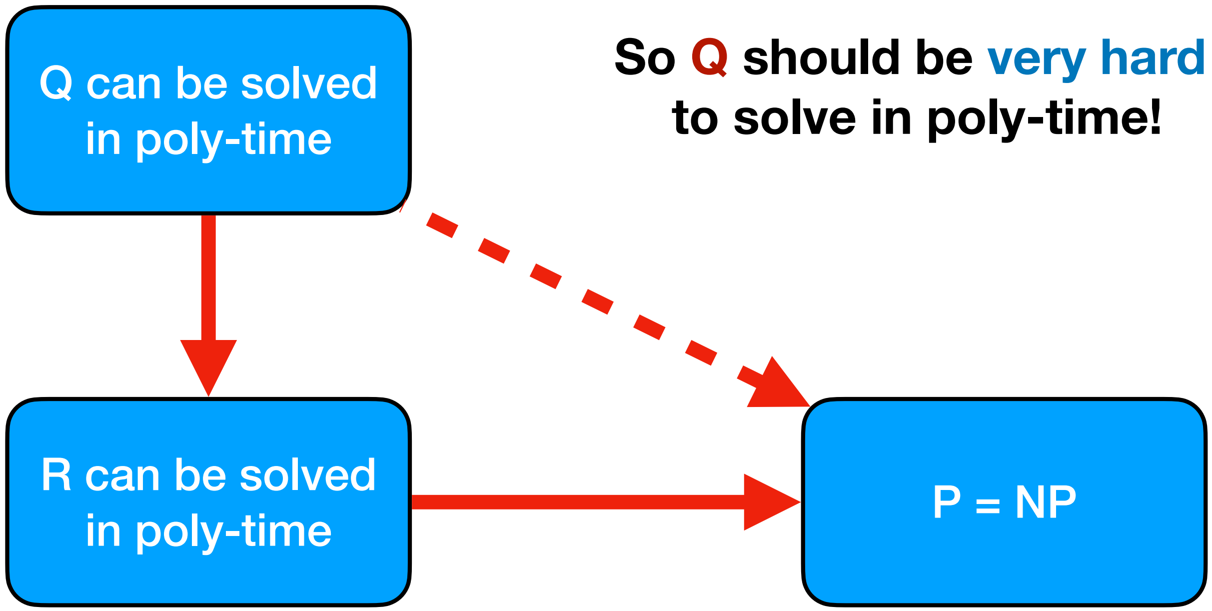
Plan



Plan



Plan



So Q should be very hard to solve in poly-time!

Plan

- **Goal**: Show that Q is very hard to solve in poly-time
- **Approach:**
  - Find any problem R such that if R can be solved in poly-time then P=NP
  - Show that R can be reduced to Q:
    - if Q can be solved in poly-time then R can also be solved in poly-time

NP-Hard Problems

- **NP-hard problems:**
  - We say a problem R is NP-hard if designing a poly-time algorithm for R implies P=NP

NP-Hard Problems

- **NP-hard problems:**
  - We say a problem R is NP-hard if designing a poly-time algorithm for R implies P=NP
- Note that the problem R itself may not be even in NP…
- This is not good since it means R can be too hard to begin with
- If R is too hard, then maybe even if we have a poly-time algorithm A for problem Q, we still cannot solve R with it in poly-time
- In other words, even if P=NP, there is no reason for R to have a poly-time algorithm

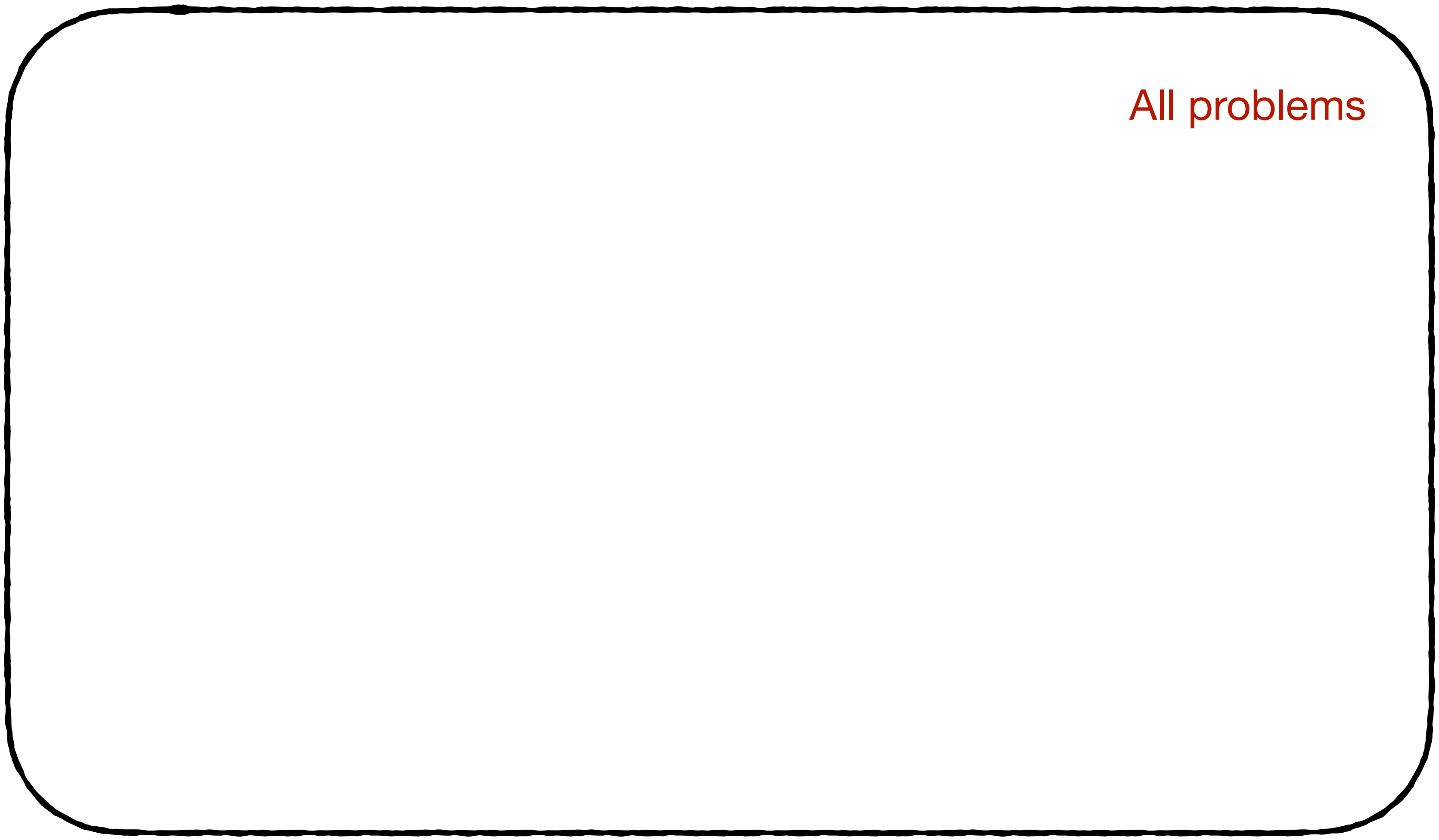
NP-Complete Problems

- **NP-complete problems:**
  - We say a problem R is NP-complete if (1) R is in NP itself, and (2) R is NP-hard

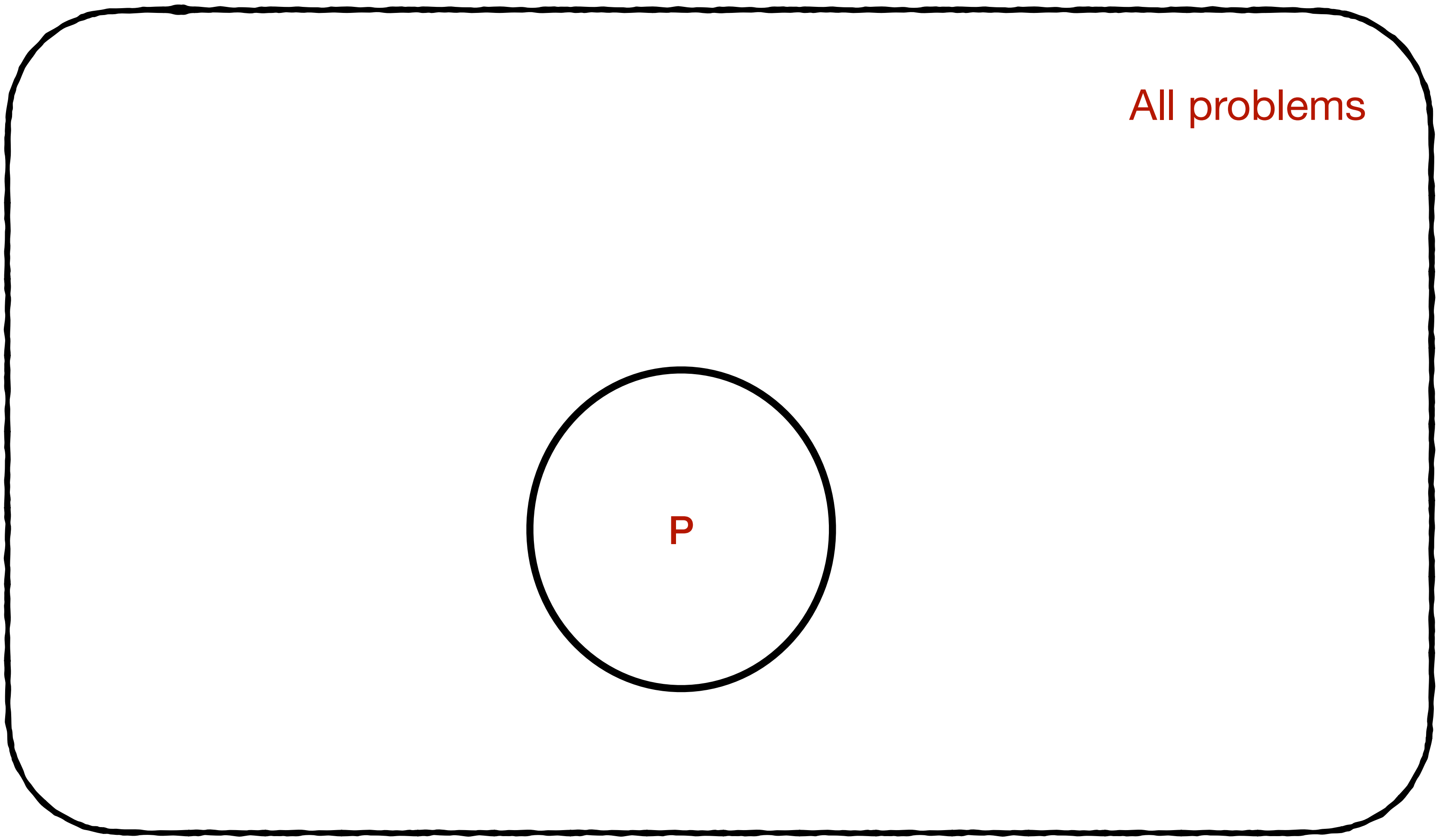
NP-Complete Problems

- **NP-complete problems:**
  - We say a problem R is NP-complete if (1) R is in NP itself, and (2) R is NP-hard
- Any NP-complete problem is in NP so we do not have the previous problem
  - If P=NP, then all NP-complete problems are solved in poly-time
- Any NP-complete problem is also in NP-hard (but the other direction is not necessarily true)

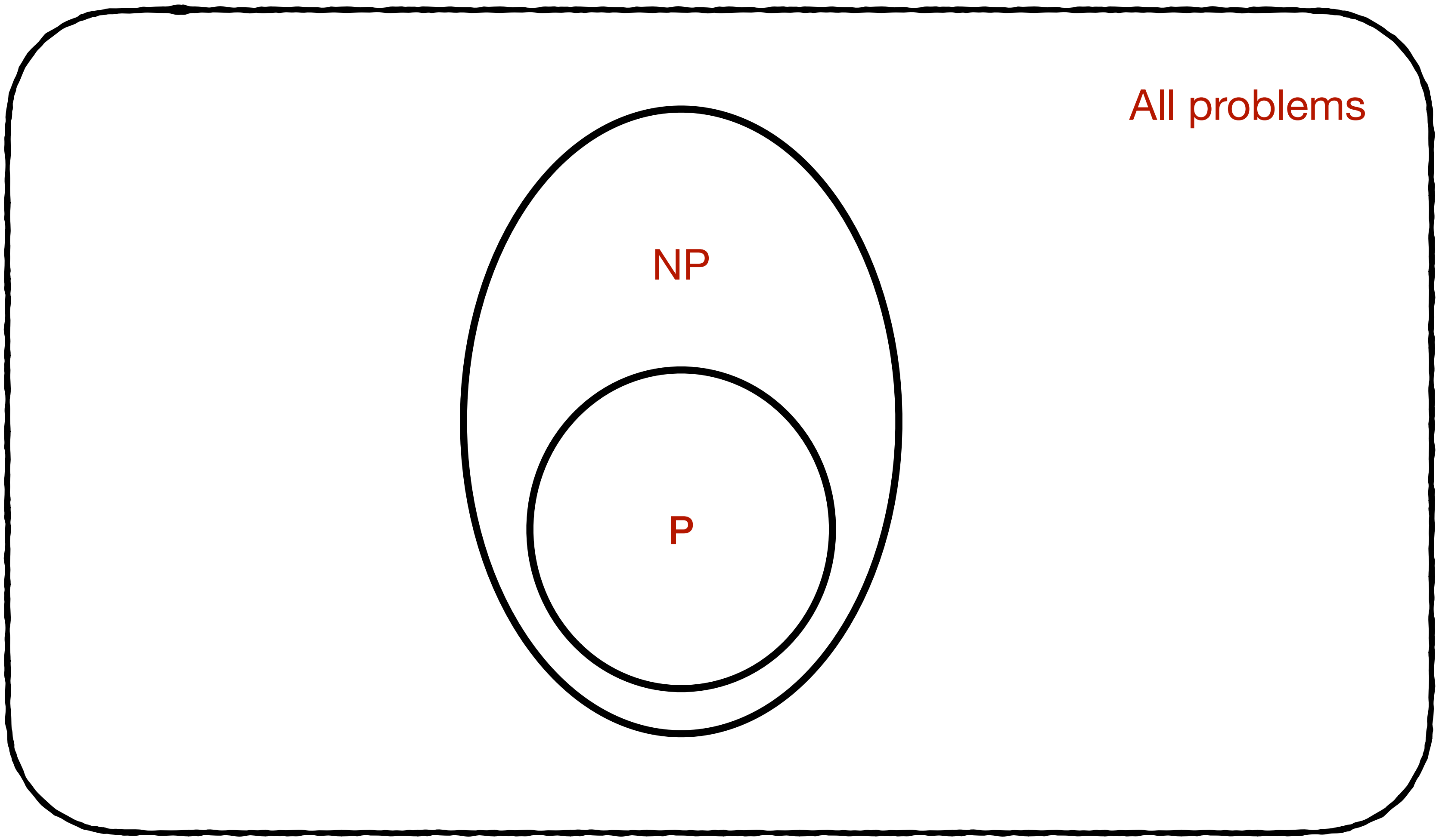
P, NP, NP-complete, NP-hard



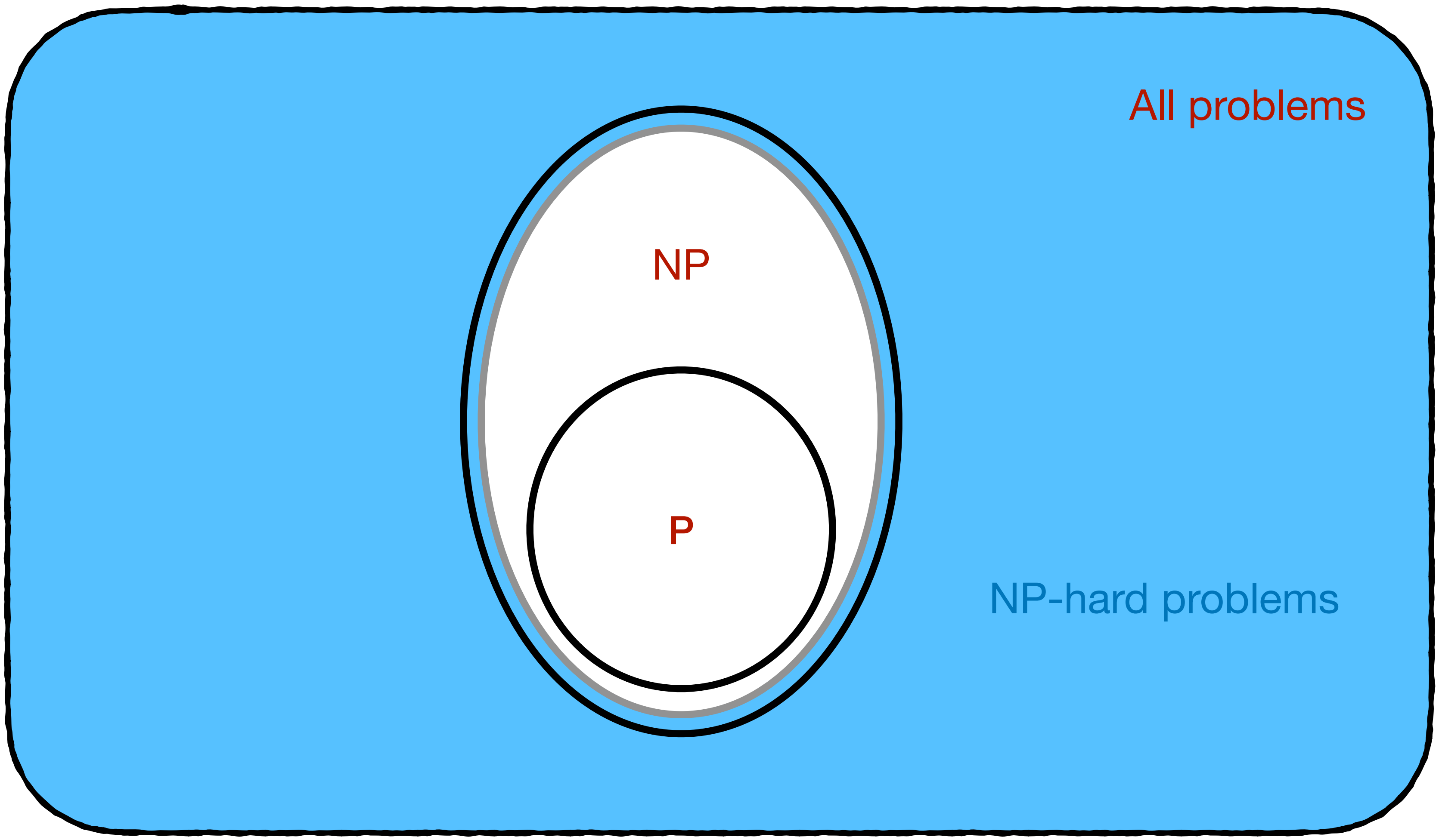
P, NP, NP-complete, NP-hard



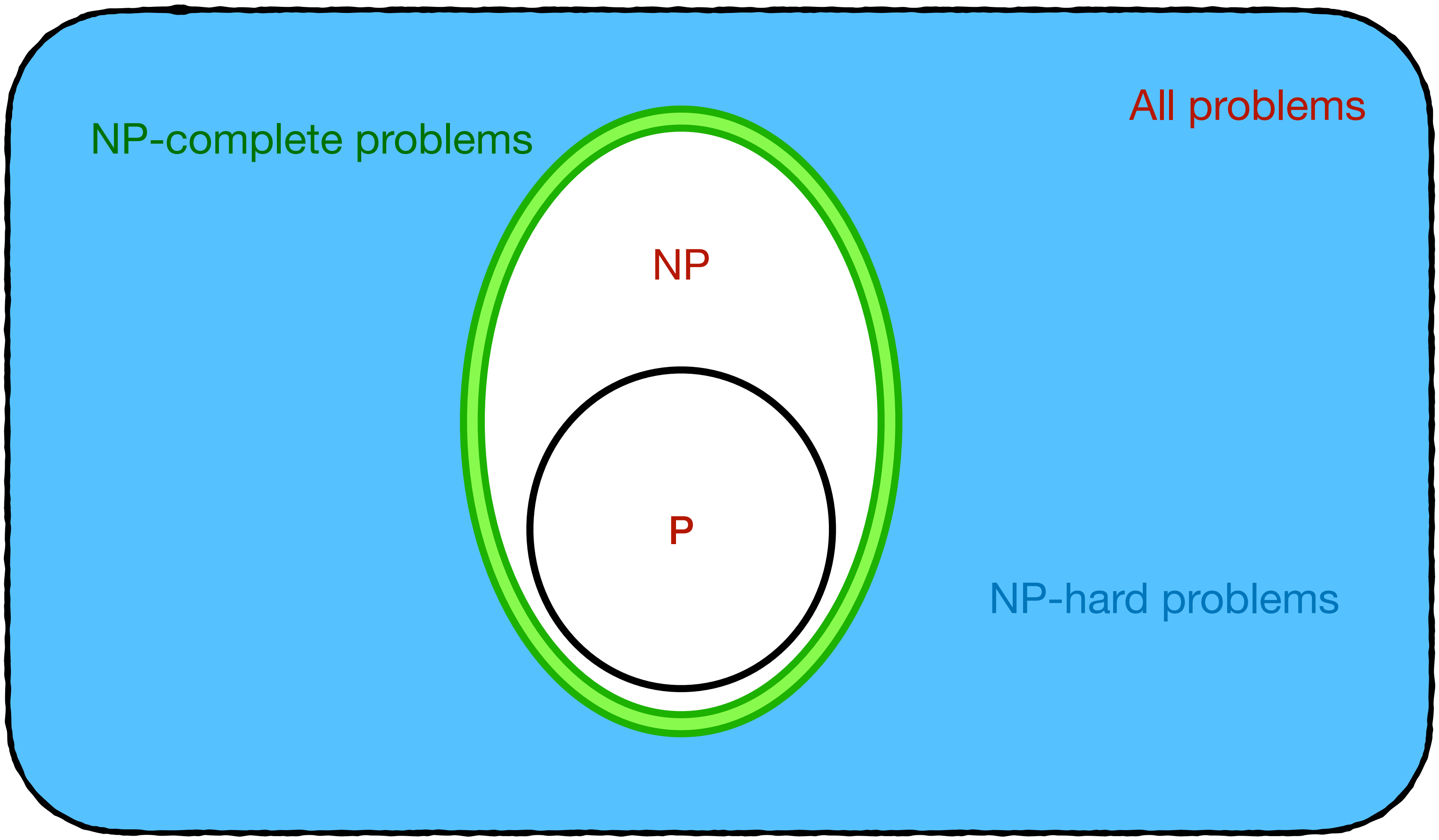
P, NP, NP-complete, NP-hard



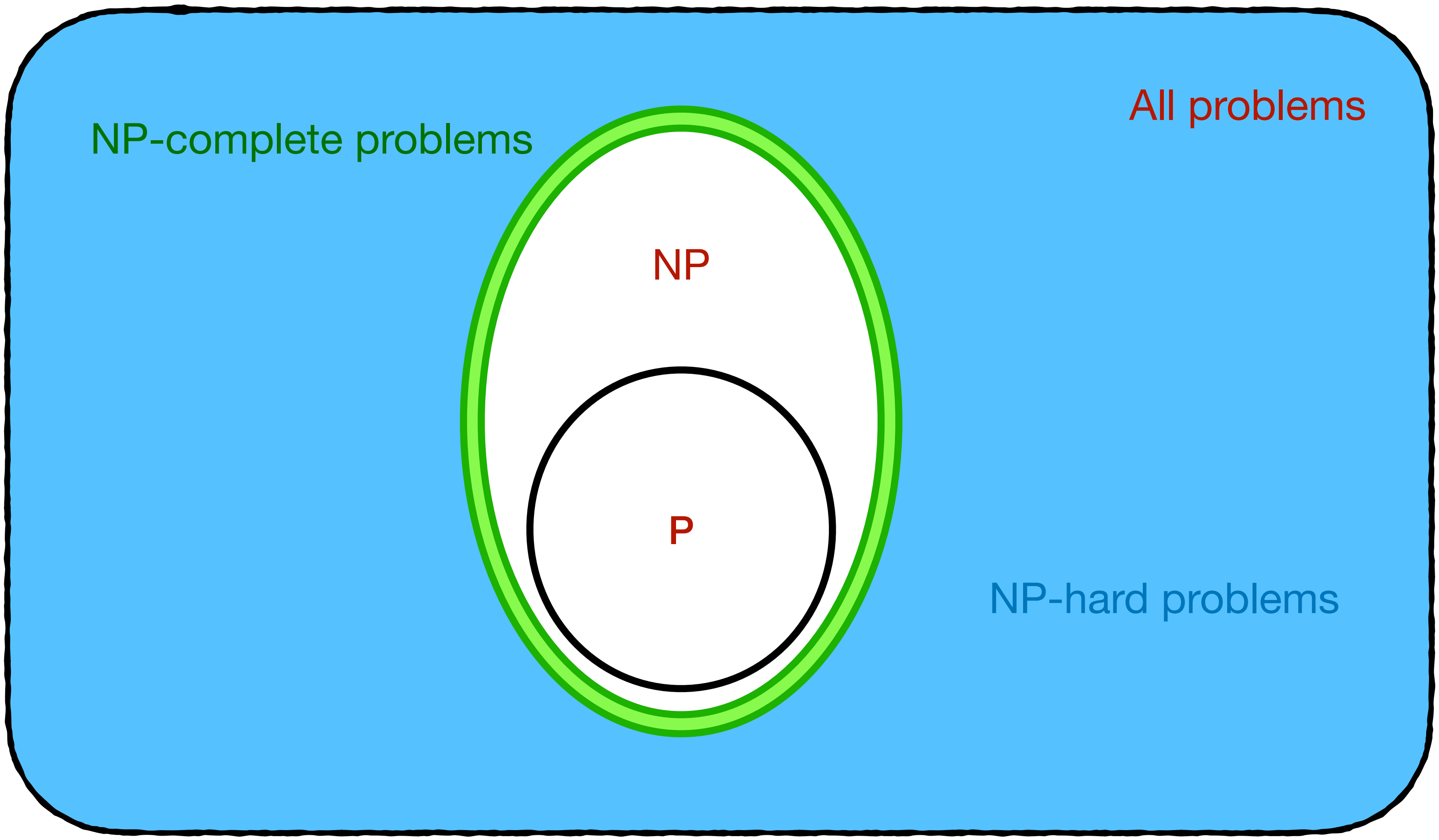
P, NP, NP-complete, NP-hard



P, NP, NP-complete, NP-hard

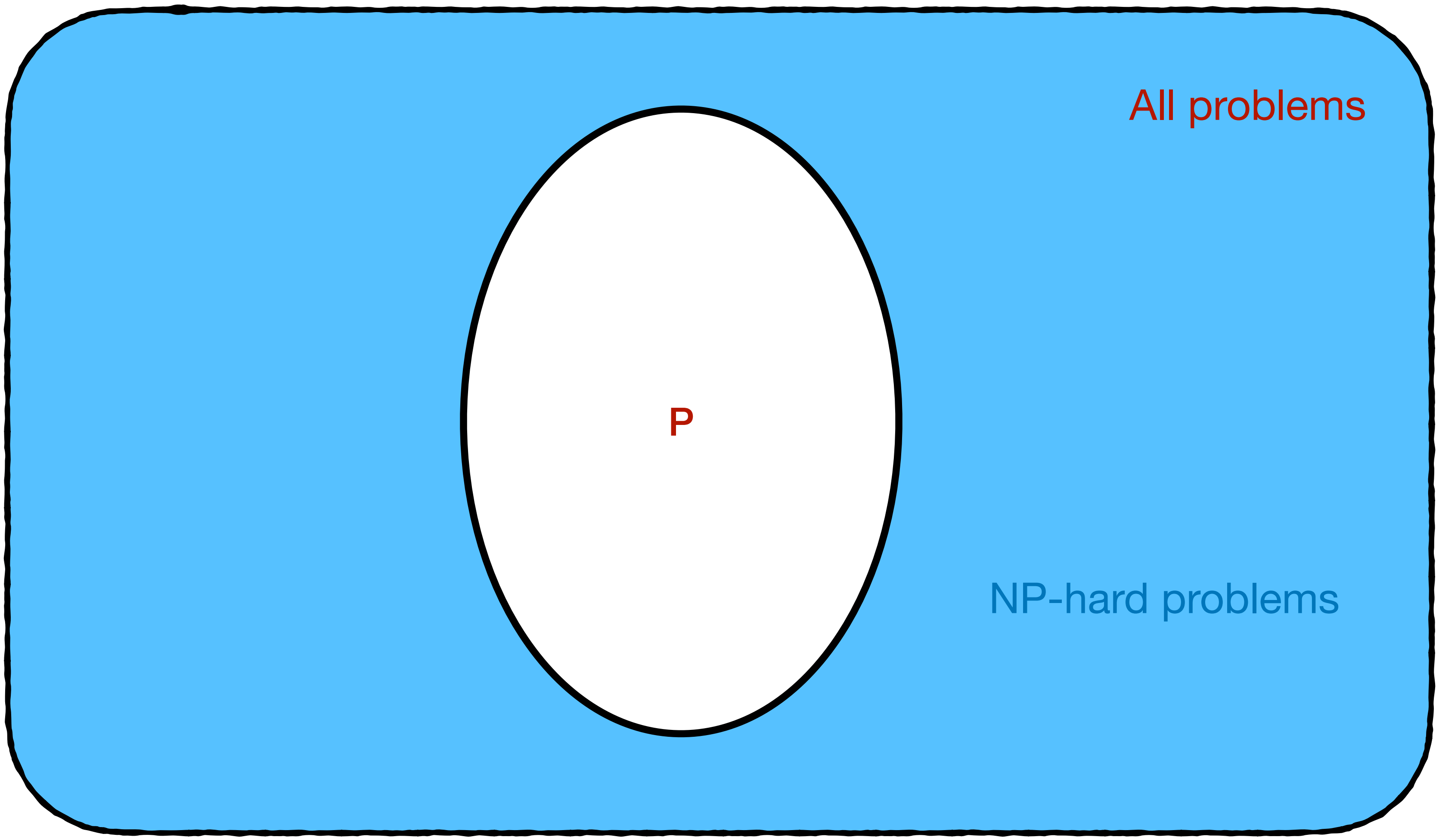


P, NP, NP-complete, NP-hard



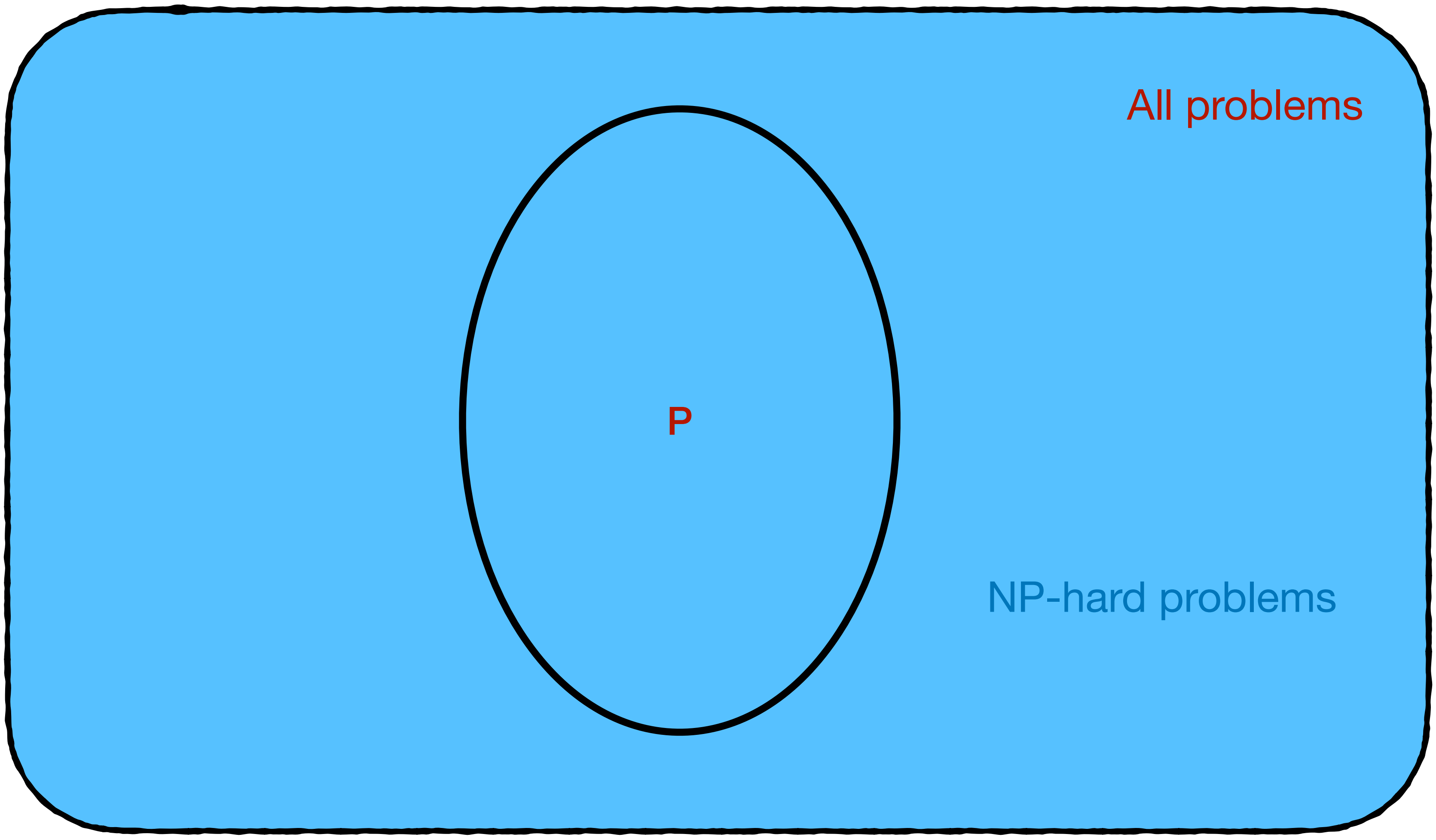
IF P = NP then this picture will become:

P, NP, NP-complete, NP-hard



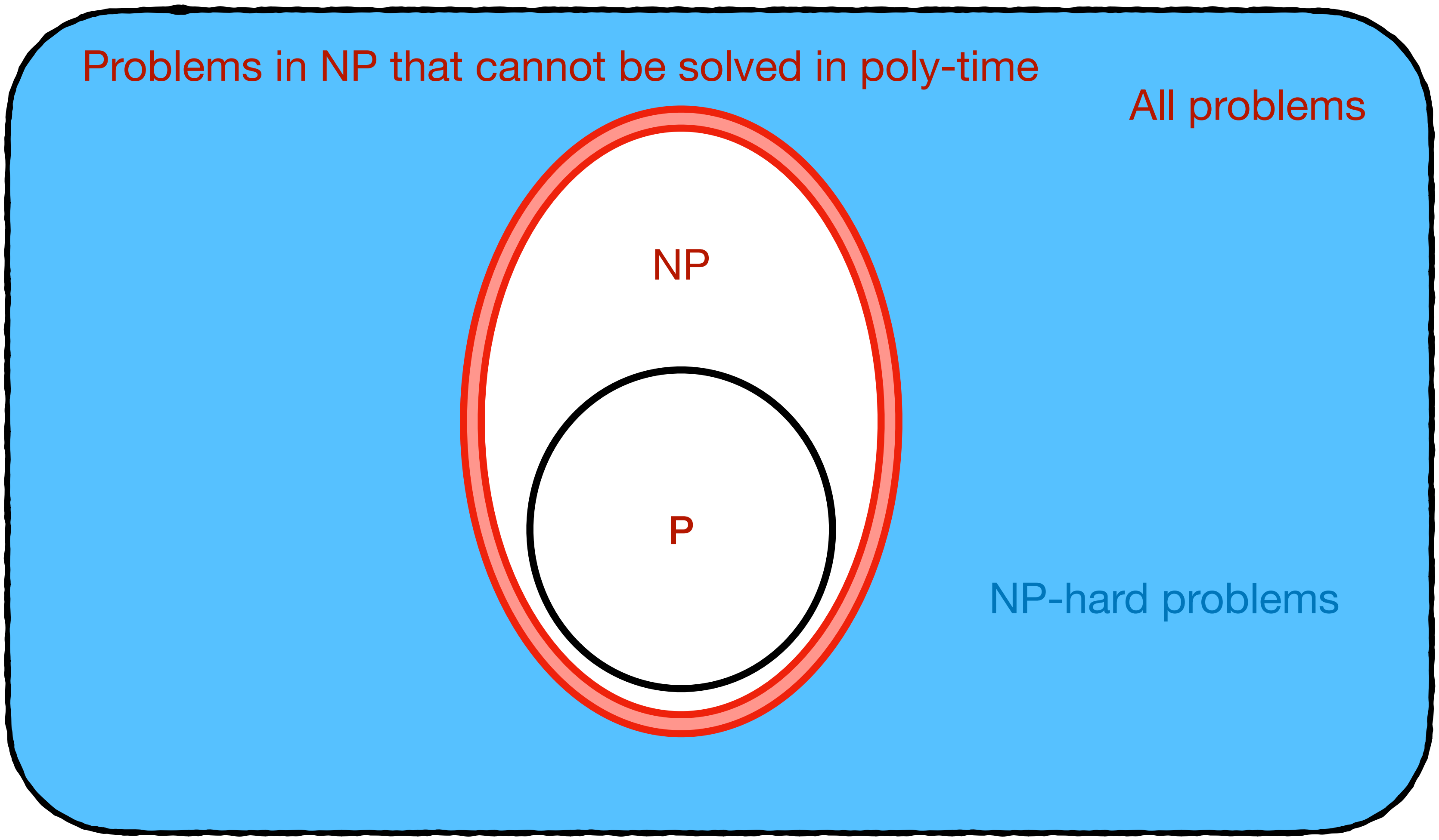
IF P = NP then this picture will become:

P, NP, NP-complete, NP-hard



IF P = NP then this picture will become: every problem will be NP-hard by definition

P, NP, NP-complete, NP-hard



IF P $\neq$ NP then this picture will become:

Plan

- **Goal**: Show that Q is very hard to solve in poly-time
- **Approach:**
  - Find any problem R such that if R can be solved in poly-time then P=NP
  - Show that R can be reduced to Q:
    - if Q can be solved in poly-time then R can also be solved in poly-time

Plan

- **Goal**: Show that Q is ~~very hard to solve in poly-time~~ **is NP-hard**
- **Approach:**
  - Find any problem R ~~such that if R can be solved in poly-time then P=NP~~ **which is already known to be NP-hard**
  - Show that R can be reduced to Q:
    - if Q can be solved in poly-time then R can also be solved in poly-time

Plan

- **Goal**: Show that Q is ~~very hard to solve in poly-time~~
  is NP-hard
- **Approach:**
  - Find any problem R ~~such that if R can be solved in poly-time then P=NP~~ **which is already known to be NP-hard**
  - Show that R can be reduced to Q:
    - if Q can be solved in poly-time then R can also be solved in poly-time
- If you want to prove R is NP-complete:
  - Prove that it is also in NP: give a poly-time verifier for it

Circuit-SAT problem & Cook-Levin Theorem

Plan

- **Goal**: Show that Q is NP-hard
- **Approach:**
  - Find any problem R which is already known to be NP-hard
  - Show that R can be reduced to Q:
    - if Q can be solved in poly-time then R can also be solved in poly-time

Plan

- **Goal**: Show that Q is NP-hard
- **Approach:**
  - Find any problem R which is already known to be NP-hard
  - Show that R can be reduced to Q:
    - if Q can be solved in poly-time then R can also be solved in poly-time
- For this plan to work we should have at least one NP-hard problem to begin with

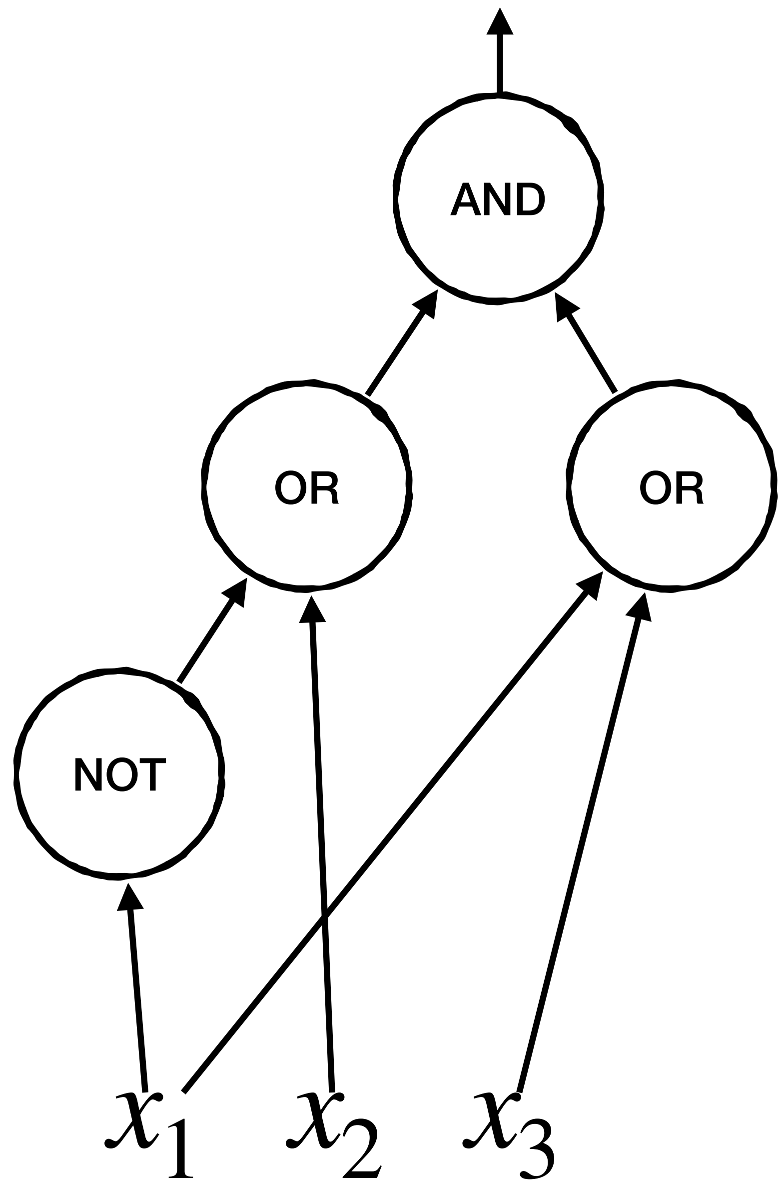
Circuit-SAT Problem

- The very first NP-hard problem is called the circuit-satisfiability problem or **circuit-SAT**

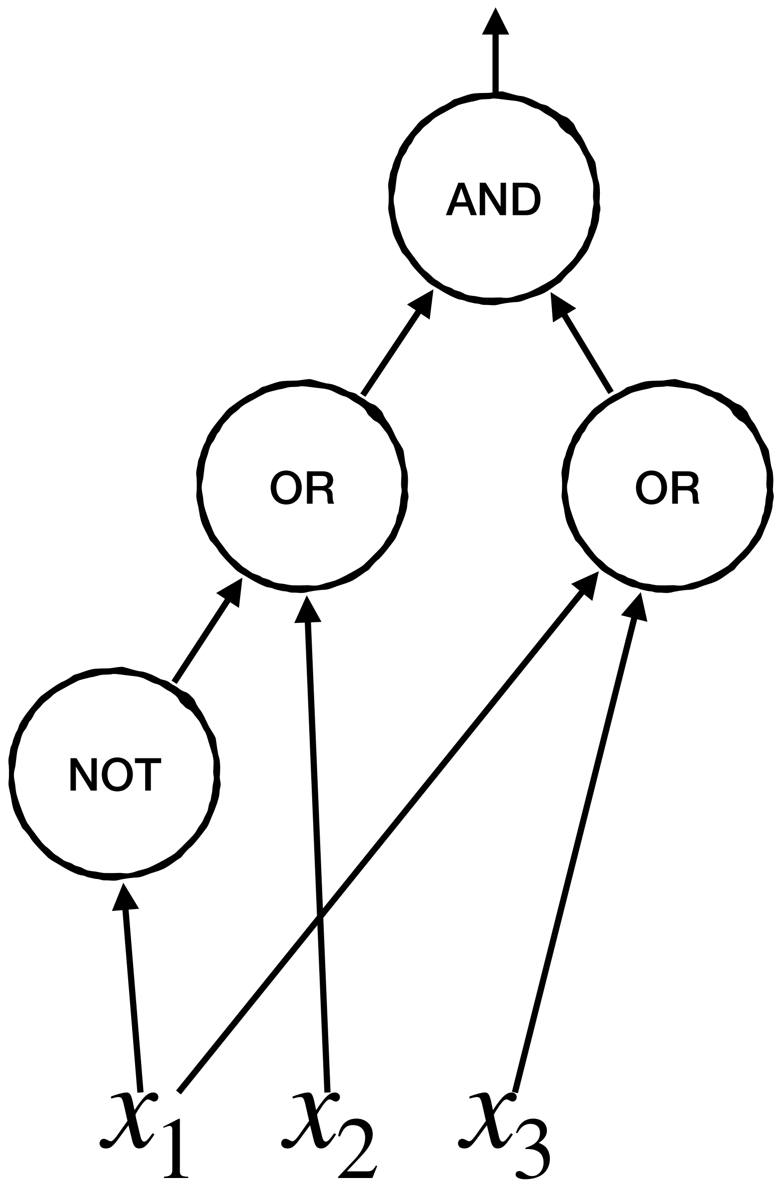
Circuit-SAT Problem

- The very first NP-hard problem is called the circuit-satisfiability problem or **circuit-SAT**
- **Input:**
  - A circuit C with binary AND and OR gates and unary NEGATE gates with n inputs in total
  - For any $x \in \{0,1\}^n$ we use C(x) to denote the value of circuit on the input x
- **Output:**
  - Is there any x such that C(x) = True?

Circuit-SAT Problem: Example

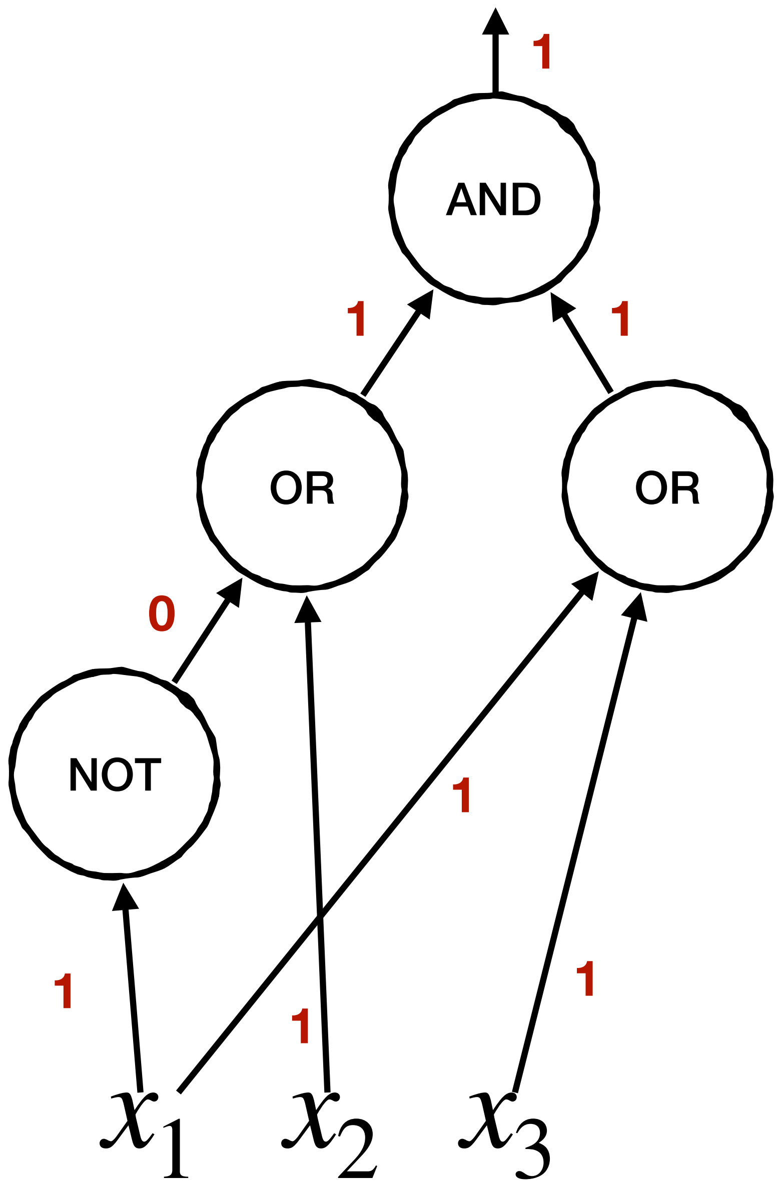


Circuit-SAT Problem: Example



Answer is YES

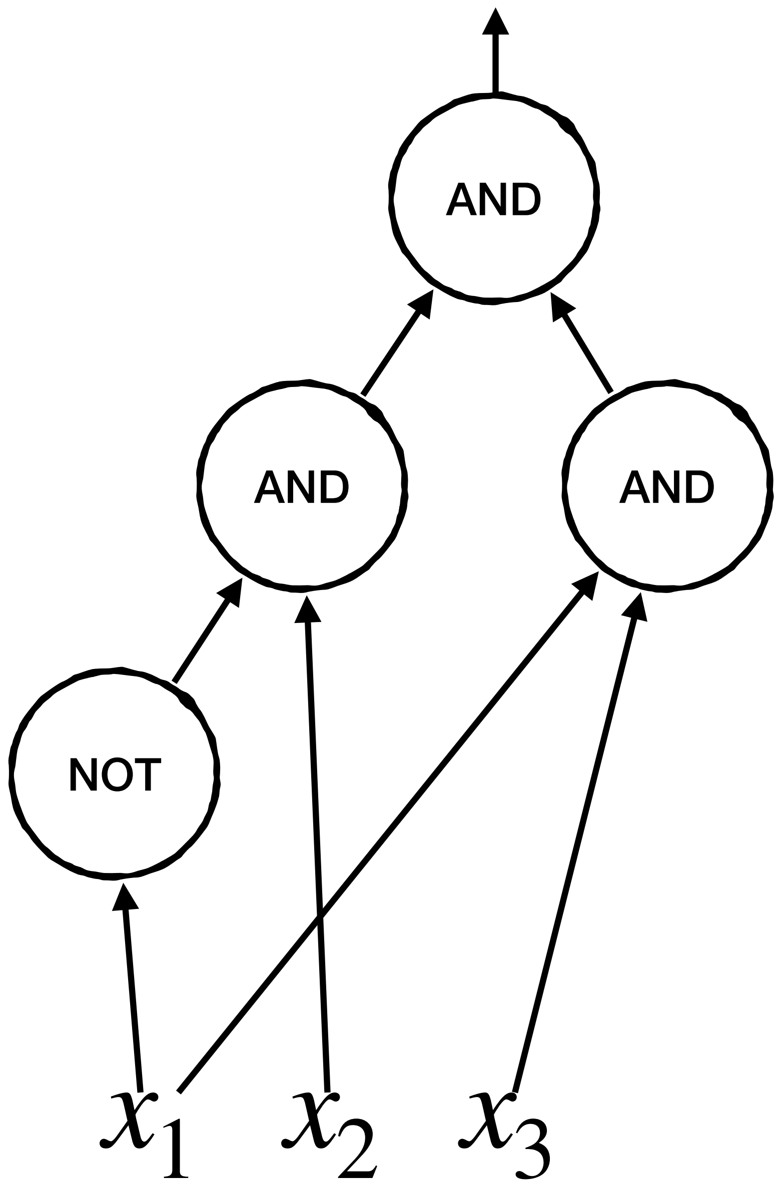
Circuit-SAT Problem: Example



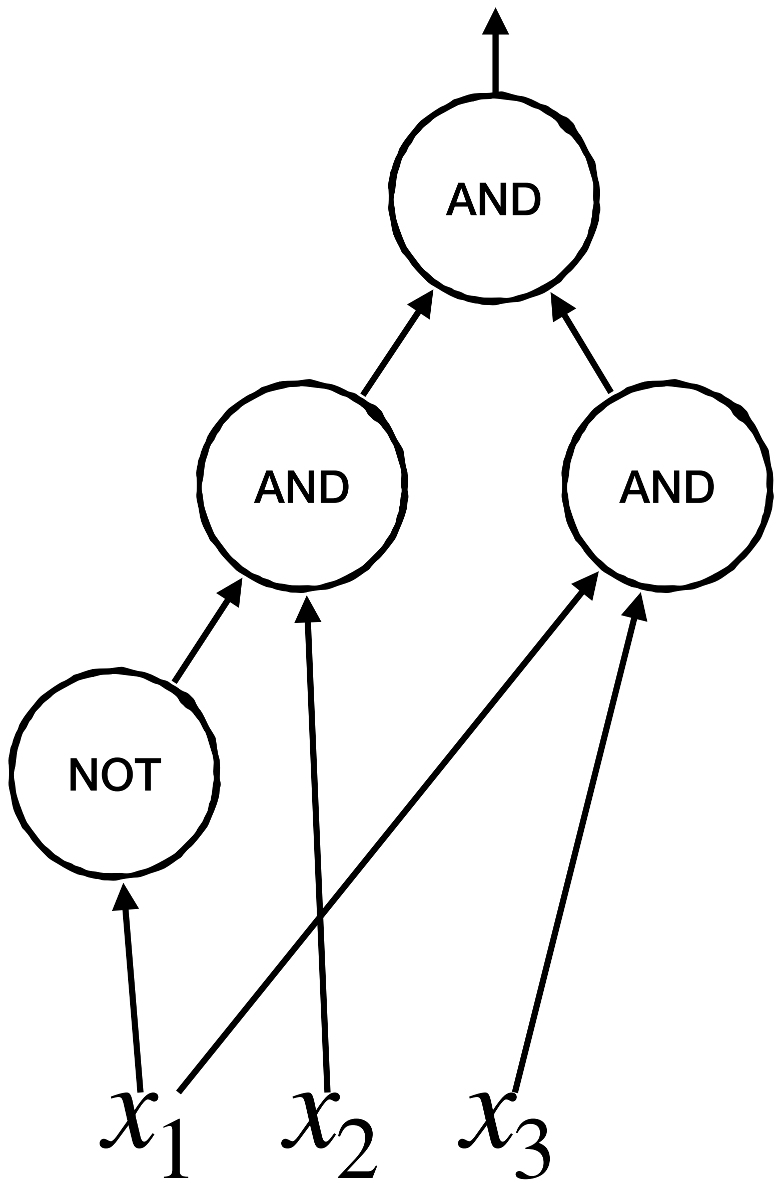
Answer is YES

For instance (1,1,1)

Circuit-SAT Problem: Example

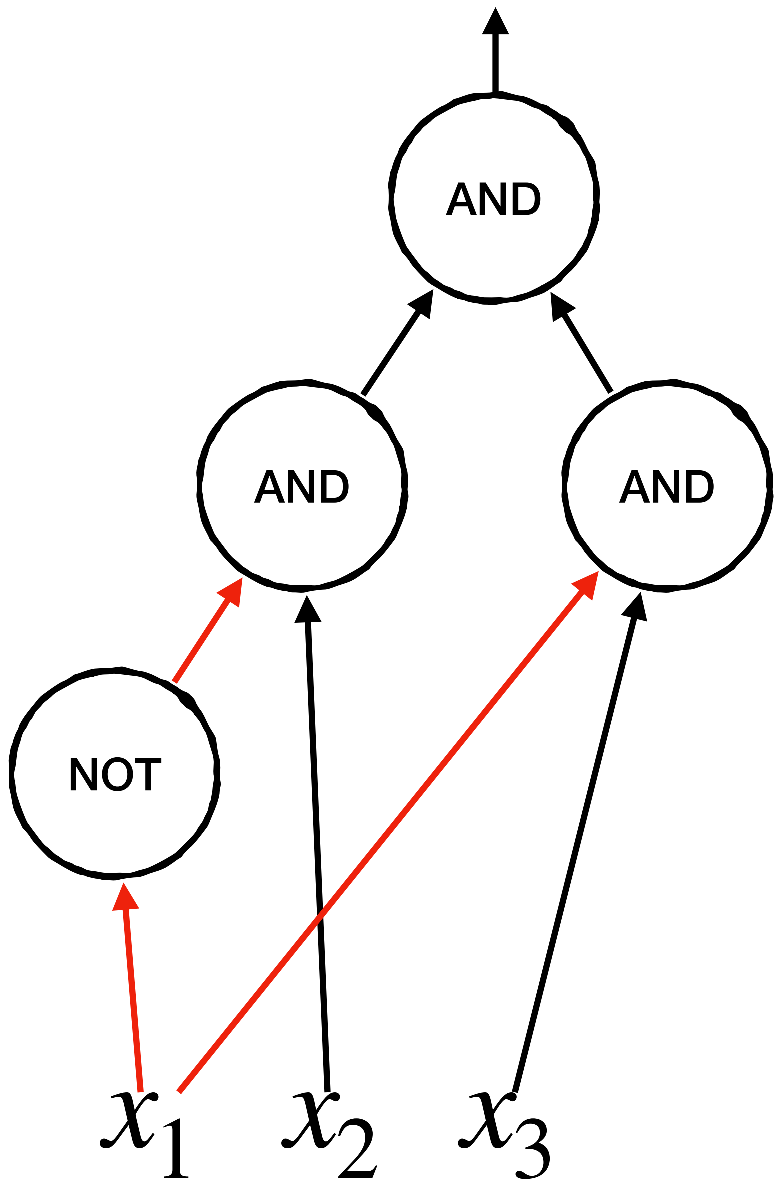


Circuit-SAT Problem: Example



Answer is NO

Circuit-SAT Problem: Example



Answer is NO

Circuit-SAT

- Circuit-SAT is in NP

Circuit-SAT

- Circuit-SAT is in NP
- A poly-time verifier:
  - The proof is an assignment x such that $C(x) = 1$
  - We can evaluate x in C to compute the answer — the evaluation is done bottom-up by computing value of each gate

Circuit-SAT

- Circuit-SAT is in NP
- A poly-time verifier:
  - The proof is an assignment x such that $C(x) = 1$
  - We can evaluate x in C to compute the answer — the evaluation is done bottom-up by computing value of each gate
- **Cook-Levin Theorem:** Circuit-SAT is NP-complete

Reductions

- We now know that circuit-SAT is NP-complete (and so NP-hard)
- We can use circuit-SAT in reductions to prove other problems are also NP-hard

Reductions

- We now know that circuit-SAT is NP-complete (and so NP-hard)
- We can use circuit-SAT in reductions to prove other problems are also NP-hard
- This is the topic of the next lecture