Conclusion
Algorithm Efficiency, Problem Classes & NP-Completeness

AUBURN UNIVERSITY

Hugh Kwon

Slides adapted from Dr. Debswapna Bhattacharya's class

Topics

- Algorithm complexity & classes: Classifying algorithms based on their complexity/efficiency
- Problem complexity & classes: Classifying problems based on the complexity of algorithms to solve them
- Important problem classes: P, NP & NP-Complete
- Ways to be a billionaire…
- Problems computers can’t solve…

Polylogarithmic Algorithms

- If $T(n)=O(\lg^k n)$ for an algorithm, it is a polylogarithmic algorithm.

Polynomial Algorithms

- If $T(n)=O(n^k)$ for an algorithm, $k \geq 1$, it is a polynomial algorithm.
- $k=1$ is a special case: linear algorithm
- $k=2$ is a special case: quadratic algorithm

Polylogarithmic vs. polynomial algorithms

- $\log^a n = o(n^b)$ for any constants $a,b > 0$.

- i.e., any positive polynomial function of n grows faster than any polylogarithmic function of n as n increases.

- So, for large inputs, polylogarithmic algorithms will be more efficient than polynomial algorithms.

Exponential Algorithms

- Exponential functions: a function with a base greater than 1 (e.g. c^n where $c>1$)

- If $T(n)=O(c^n)$ where $c>1$ for an algorithm, it is an exponential algorithm.

Polynomial algorithms v.s. Exponential algorithms

- Any exponential function with a base greater than 1 (e.g. c^n where $c>1$) grows faster than any polynomial function n^b, where b and c are constants.
  - $n^b = o(2^n)$
- So, for large inputs, polynomial algorithms will be more efficient than exponential algorithms.

Factorial Algorithms

- Factorial function: a function of the form n!

- If T(n)=O(n!) for an algorithm, it is an algorithm of factorial complexity.

Exponential algorithms v.s. Factorial algorithms

- $2^n = o(n!)$

- So, for large inputs, exponential algorithms with a base of 2 will be more efficient than factorial algorithms.

- $n! = o(n^n)$

- The function n^n grows even more quickly than the factorial function. Therefore, factorial algorithms will be more efficient than combinatorial algorithms with complexity order $O(n^n)$.

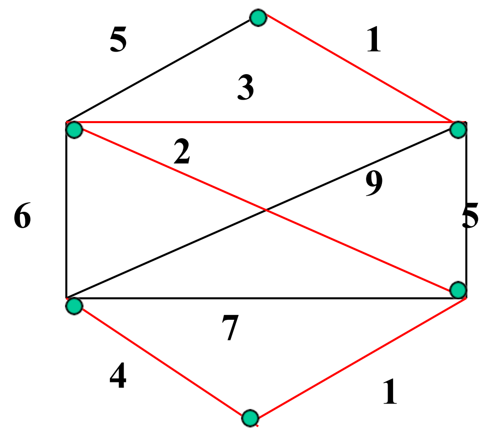
Algorithm Complexity & Classes

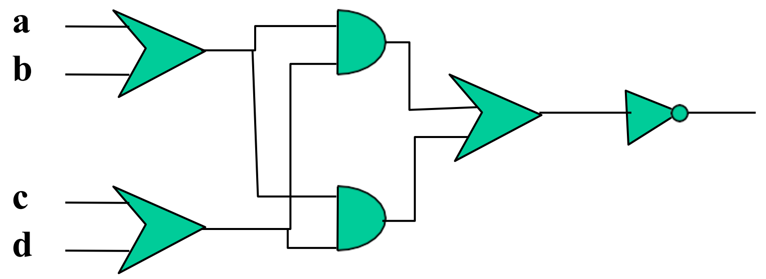
Levels of algorithmic complexity

	if n=8 and k=2	
constant time: $O(1)$	1	
polylogarithmic: $O(\log n)$	3	
linear: $O(n)$	8	
in-between linear and polynomial: $O(n\log n)$	24	**efficient**
polynomial: $O(n^k)$	64	$\uparrow$
exponential: $O(k^n)$	256	$\downarrow$
factorial: $O(n!)$	40,320	**inefficient**
combinatorial: $O(n^n)$	16777216	

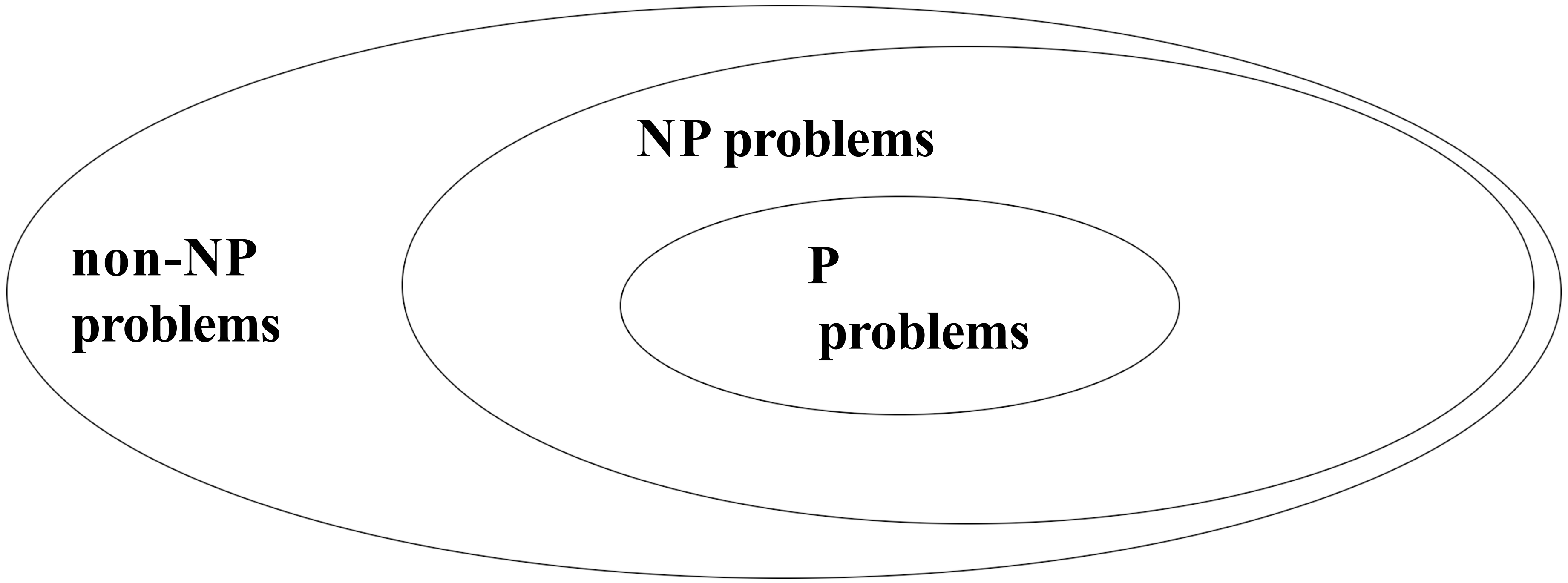
Problem Complexity & Classes

- P (Polynomial)
  - all problems that can be solved by algorithms of polynomial complexity or better
  - all problems we discussed in this course are in P
- NP (Non-deterministic Polynomial)
  - this includes the problem class P
  - also includes problems to solve which polynomial time or faster algorithms are not known (i.e., all known algorithms to solve these problems have worse than polynomial complexity)
  - but not all problems with worse than polynomial complexity are in NP
  - to be in NP problems must be such that their solutions can be verified by algorithms of polynomial complexity or less
  - e.g. Boolean Circuit Satisfiability Problem





Every P problem is also an NP problem



- Any problem in P is also a problem in NP; so P is a subset of NP

Is every NP problem a P problem?

- i.e., if a problem is in NP but it is not in P, is it possible that it can be solved by a polynomial time or faster algorithm?
- If so, i.e., if every problem in NP can be solved by a polynomial time or faster algorithm, then NP will be a subset of P.
- If it was, then the two problem classes will be the same, i.e., P=NP
- Is P=NP? Nobody knows!

P=NP?

- This is the most important open problem in Computer Science
- If P=NP this means that ALL problems in NP can be solved by efficient polynomial algorithms! There are thousands of scientifically and commercially important problems in NP.
- You can be a millionaire if you can solve this. **No kidding!**
- https://www.claymath.org/millennium-problems/p-vs-np-problem

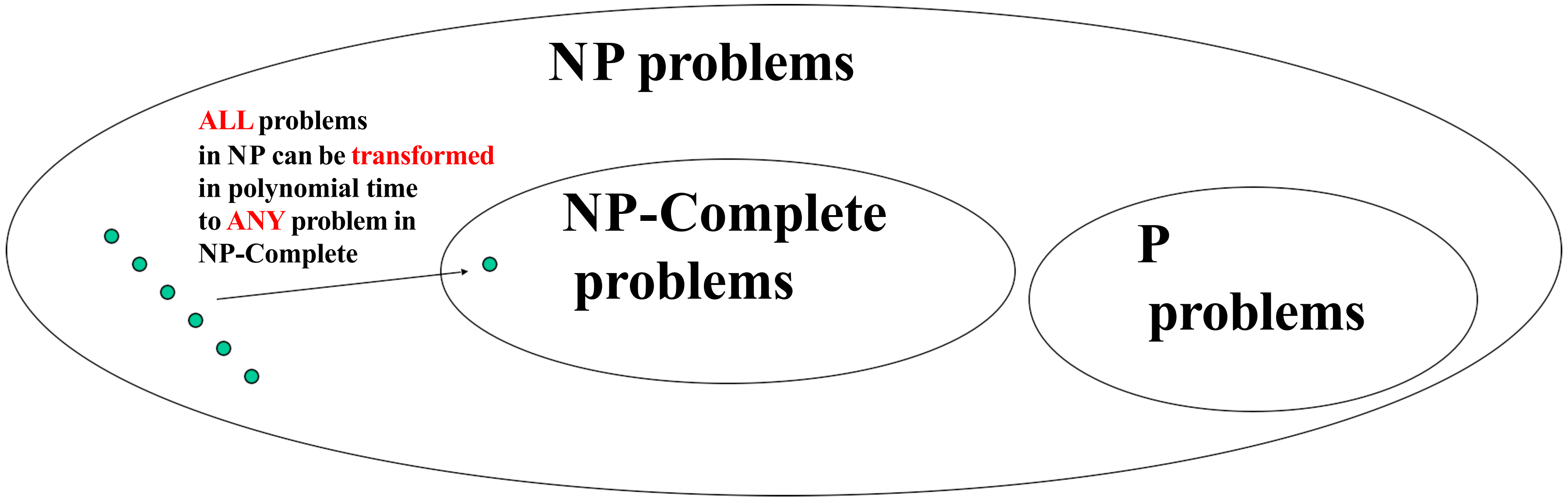
P=NP?

- In the early seventies a UC Berkeley computer scientist called Cook discovered a peculiar property of a problem, the Boolean Circuit Satisfiability Problem, known to be in NP
- He discovered that it was possible to transform any instance of ALL problems in NP to instances of the Circuit Satisfiability Problem (so that you can construct a solution to the original NP problem from the solution to this instance of the Satisfiability problem and vice versa), and that this transformation will only take polynomial time or less for ANY NP problem.
- Why is this significant?

P=NP?

- An NP-Complete problem is one such that ALL problems in NP can be transformed to particular instances of it (so that you can construct a solution to the original NP problem from the solution to this instance of the NP-Complete problem and vice versa), where this transformation can be done in polynomial time or less for any NP problem.
- This means that the invention of an efficient (polynomial time or faster) algorithm to solve any ONE NP-Complete problem will provide the means to solve ALL NP problems efficiently!
- Boolean Circuit Satisfiability Problem was the first NP-Complete problem to be discovered.

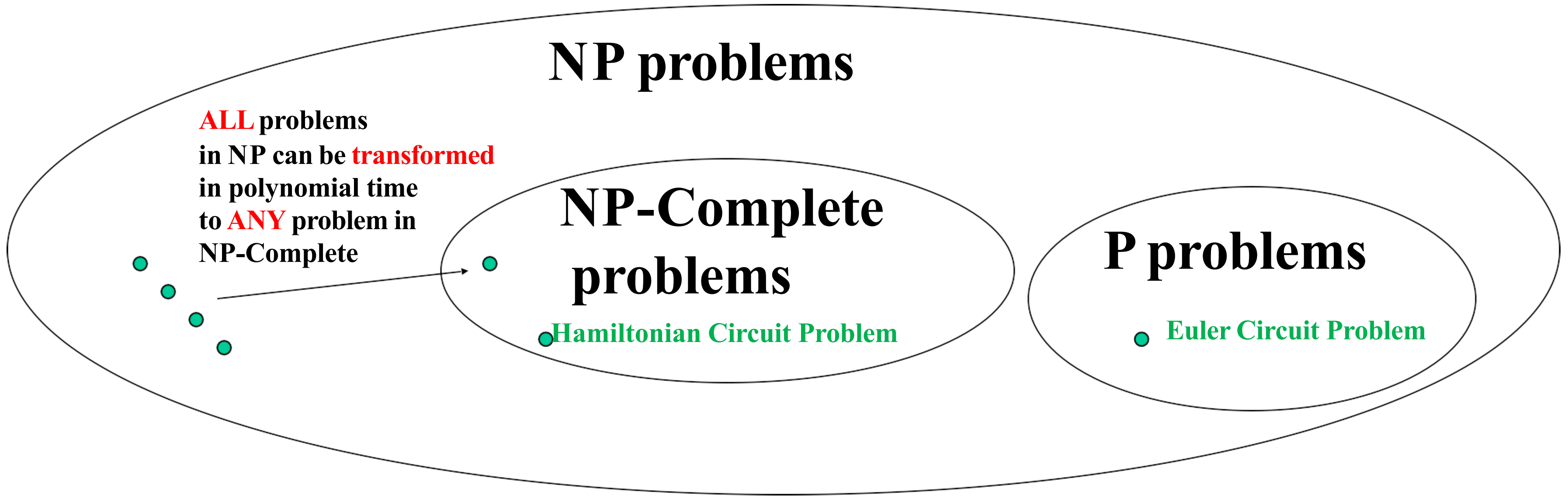
P=NP?


NP problems
ALL problems in NP can be transformed in polynomial time to ANY problem in NP-Complete
NP-Complete problems
P problems

Two seemingly simple problems

- The Hamiltonian Circuit Problem (HCP) is to find a cycle in a graph that visits every node **exactly once**.
- The Euler Circuit Problem (ECP) is to find a cycle in a graph that visits every edge **exactly once**.
- ECP can be solved in **linear** time using an algorithm that employs DFS, but HCP is **NP-Complete** and all known algorithms to solve it are exponential or worse!

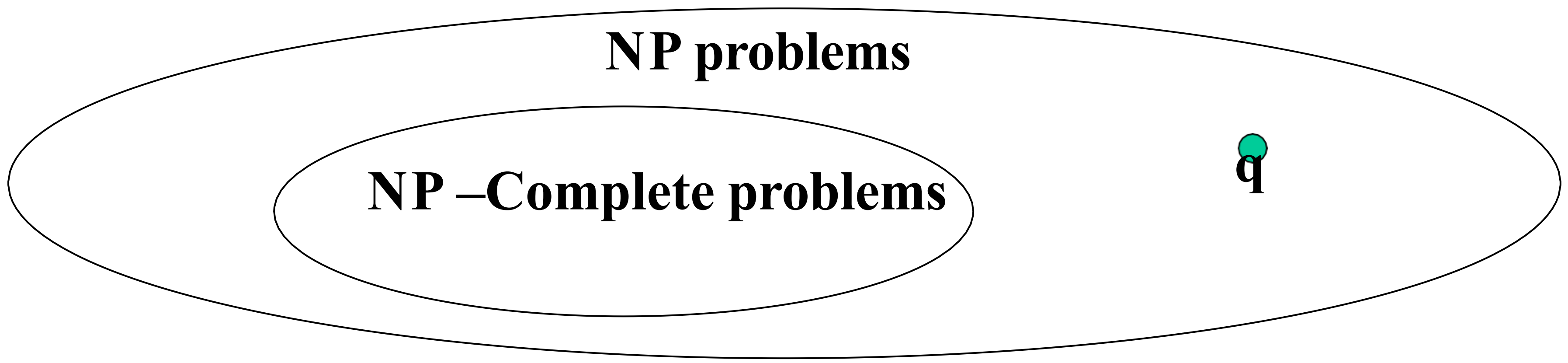
P=NP?


NP problems
ALL problems in NP can be transformed in polynomial time to ANY problem in NP-Complete
NP-Complete problems
Hamiltonian Circuit Problem
P problems
Euler Circuit Problem

New NP-Complete problems

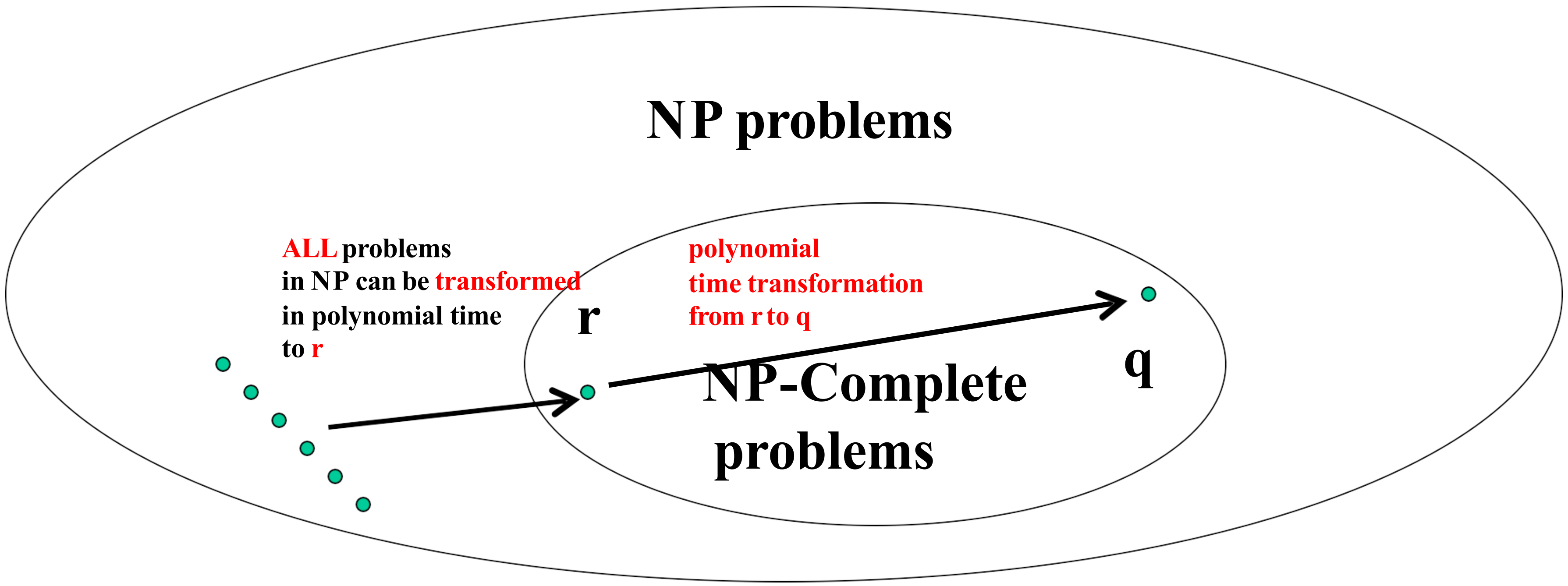
- After Cook discovered the NP-Completeness of the Boolean Circuit Satisfiability Problem, the set of NP-Complete Problems had only one problem in it.
- Now NP-Complete class of problems contain hundreds of problems (see http://en.wikipedia.org/wiki/List_of_NP-complete_problems)
- How did people add to this set? i.e. How do you show a new problem to be NP-Complete?

How to Show A Problem to be NP- Complete?



- To show a new problem q to be NP- Complete:
  - Step 1: show it is in NP
    - i.e. given a potential solution, show it can be checked in polynomial time by an algorithm

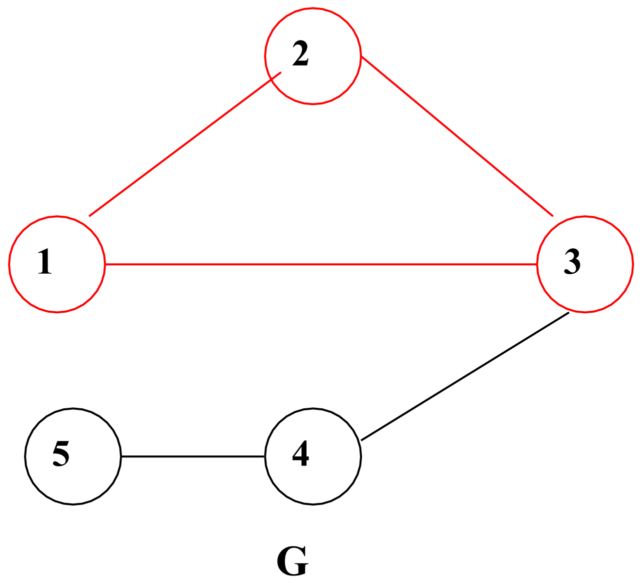
How to Show A Problem to be NP- Complete?



- Step 2: show that a problem r, already known to be NP-Complete, can be transformed in polynomial time or faster to q such that a solution for r gives a solution for q and vice versa.

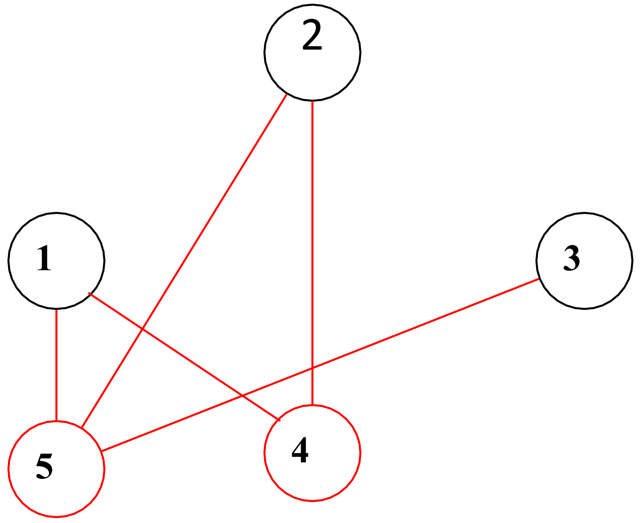
Example

- The Clique Problem (CP) is to find whether a graph G has a clique (a subset of nodes such that there is an edge between every pair of nodes in that subset) of size k . It is a known NP-Complete Problem.



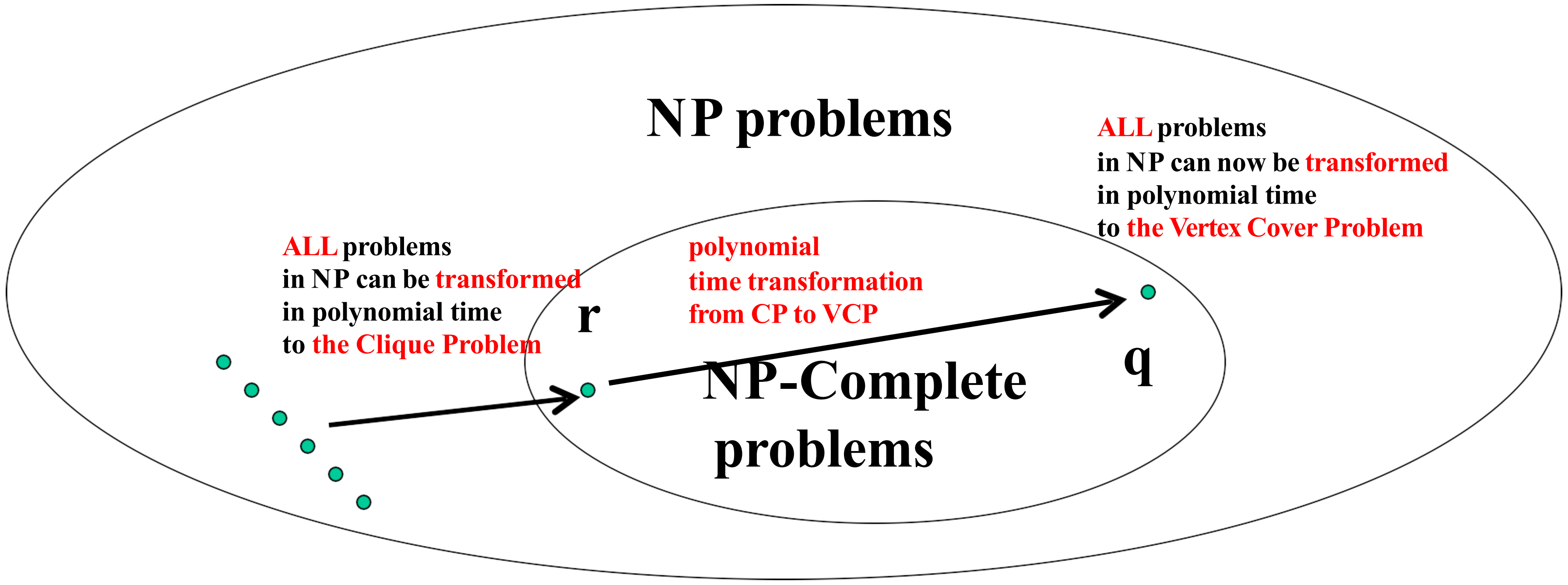
Example

- The Vertex Cover Problem (VCP) is to determine whether a graph has a subset of k nodes such that all its edges are connected to nodes in this subset.



G’

To show that the Vertex Cover Problem is NP-Complete


NP problems
ALL problems in NP can now be transformed in polynomial time to the Vertex Cover Problem
ALL problems in NP can be transformed in polynomial time to the Clique Problem
polynomial time transformation from CP to VCP
r
q
NP-Complete problems

- **Step 1**: show that a potential solution to the Vertex Cover Problem can be checked in polynomial time – *this proves that VCP is in NP*
- **Step 2:** show how any instance of the Clique Problem, already known to be NP- Complete, can be transformed in polynomial time or faster to an instance of the Vertex Cover Problem – *this proves that any NP problem can be transformed in polynomial time to VCP* – such that a solution to any instance of the Clique Problem can be turned into a solution to the Vertex Cover problem and vice versa – *this proves that the two problems are equivalent and if you solve one you have solved the other*

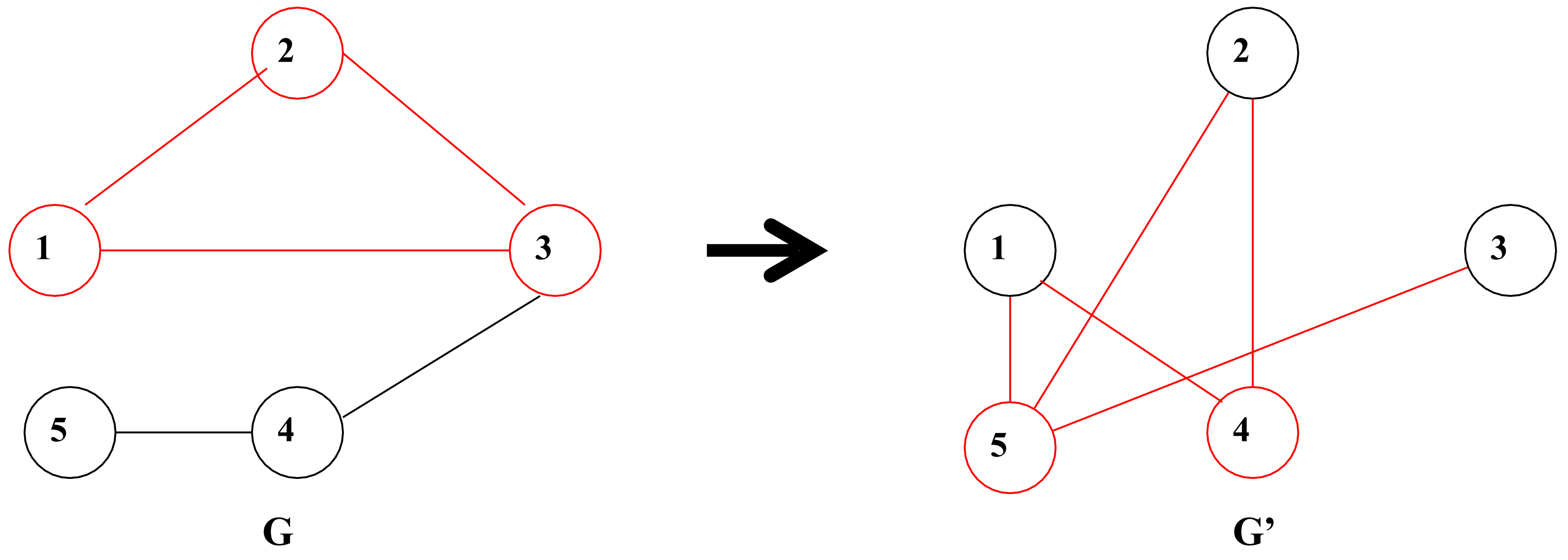
Example

- Step 1: VCP is in NP – why?
- Because we can easily write a polynomial time algorithm to verify a given solution to the Vertex Cover Problem.

	1	**2**	**3**	**4**	**5**
1	0	0	0	1	1
2	0	0	0	1	1
3	0	0	0	0	1
4	1	1	0	0	0
5	1	1	1	0	0

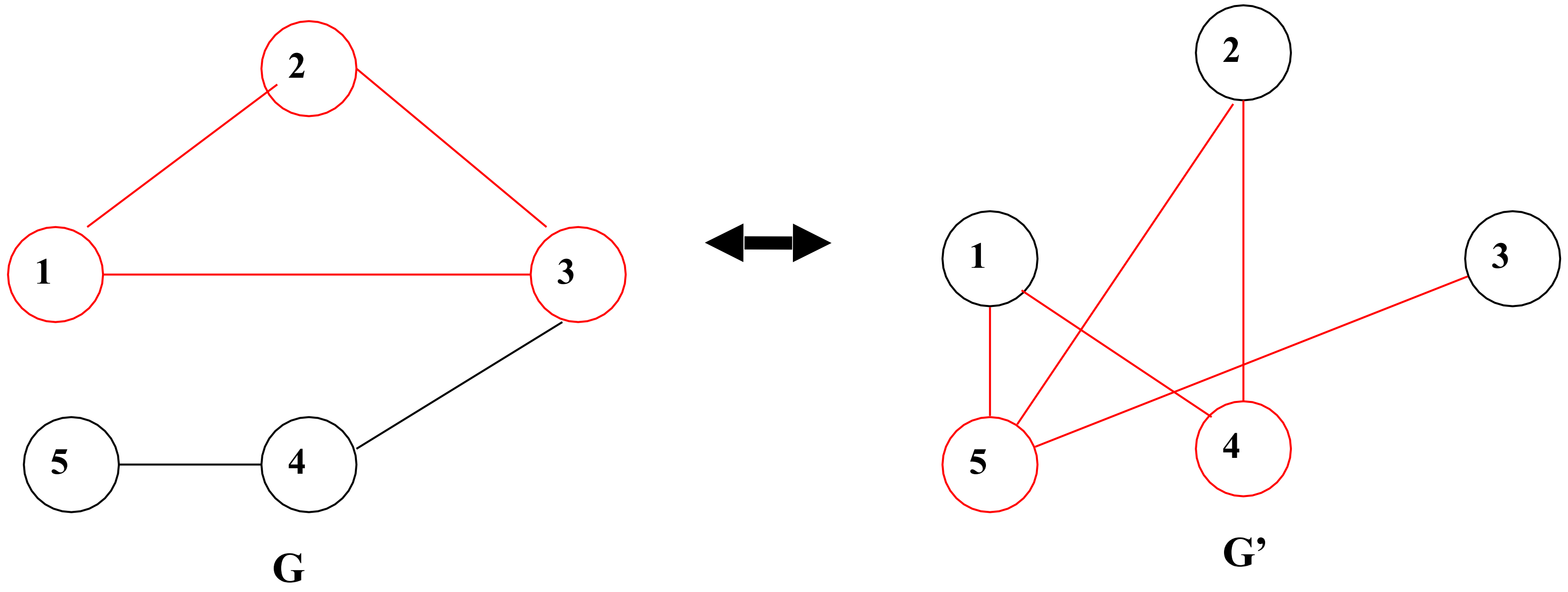
CP-to-VCP Transformation

- Step 2:
- Given an instance of CP, i.e, a particular graph G, construct a new G’ for VCP where G’ contains only the edges that are not in G.
- Clearly, this can be done in polynomial time since there are at most $(n^2-n)/2$ possible edges in a graph with n nodes



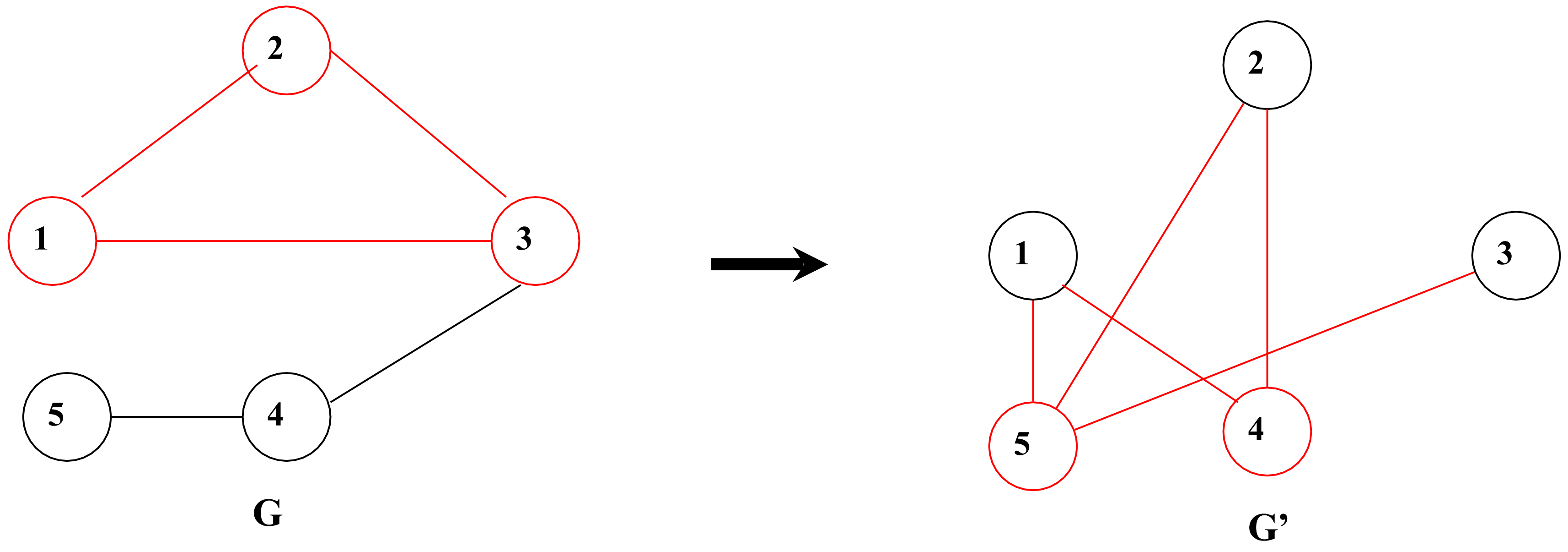
CP-to-VCP Transformation

- We have to prove that under this transformation a solution for CP, i.e., clique in G, provides a solution for VCP, i.e., a vertex cover in G', and vice versa
- The way to derive one solution from the other is: G has a Clique of size k <u>if and only if</u> G' has Vertex Cover of size n-k
- We have to show this, i.e., prove the following two statements:
    - If G has a Clique of size k then G' has Vertex Cover of size n-k, AND
    - If G' has a Vertex Cover of size n-k then G has a Clique of size k



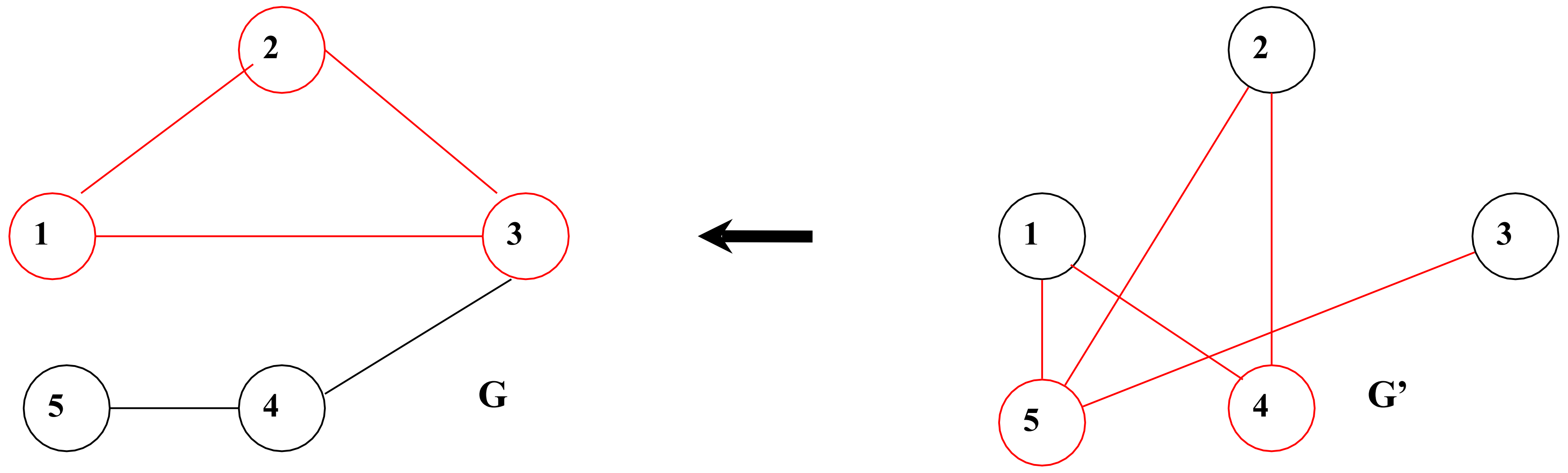
CP Solution → VC Solution

- **If G has a clique of size k then G' has vertex cover of size n-k. Why?**
- Suppose G with n nodes has a Clique C of size k. None of the edges (u',v') in G' can be such that both u and v are in C. Why? <u>If both u and v are in C, a Clique of G, then G will have an edge (u,v) so (u,v) cannot be an edge in G'. So, for every edge (u',v') in G', u', v', or both have to be in the set of nodes V-C</u> where V is the set of all nodes and C is the set of nodes in the Clique of G. The size of V-C is n-k, and so every edge in G' has to have at least one of its end points in the set of nodes V-C. That is, G' has a Vertex Cover of size n-k.
- So, if G has a Clique of size k this means that G' must have a Vertex Cover of size n-k.



CP Solution ←VC Solution

- **If G’ has a vertex cover of size n-k then G has a clique of size k. Why?**
- Suppose G’ has a Vertex Cover of size n-k. That is, all edges in G’ has at least one node from the subset of n-k nodes as its end point. This means that there are k nodes in G’ such that there is not a single edge in G’ between any two of these k nodes. Why? Because G’ has a Vertex Cover of size n-k , every edge in G’ has to have at least one of its end points in the set of n-k nodes, so there can be no edge in G’ between any two of the k nodes. Since there are k nodes in G’ with no edge between them, there must be k nodes in G with all edges between them present, because the absence of an edge in G’ means that the edge is present in G. This means that G has a clique of size k.
- So if G’ has a Vertex Cover of size n-k, G must have a Clique of size k.



NP-Complete Problems

- We have shown
  1. that VCP is in NP,
  2. how to transform, using a polynomial time algorithm, any instance of the Clique Problem to a corresponding instance of a Vertex Cover problem such that if G has a Clique of size k if and only if G' has a Vertex Cover of size n-k when there are n nodes in G and G'.
- Since CP is a known NP-Complete problem, this proves that VCP is an NP-Complete problem too!
- This is the method people have used to show the hundreds of problems (see http://en.wikipedia.org/wiki/List_of_NP-complete_problems) to be NP-Complete so that one polynomial time algorithm to solve one of these problems is sufficient to show that P=NP!

How to be a billionaire?

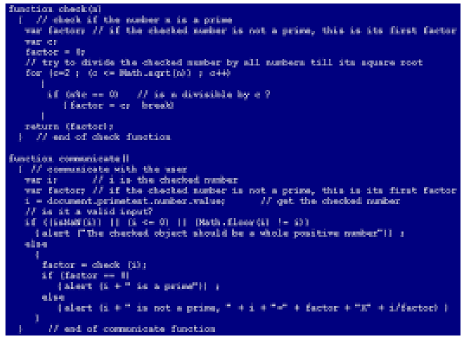
- Invent a polynomial time algorithm to solve any NP-Complete problem (and there are hundreds to choose from)
- Instantly claim the million-dollar prize at http://www.claymath.org/millennium-problems/p-vs-np-problem
- Patent it
- Live on the royalties
- **Even if you are this smart, there are problems for which you won’t be able to develop an algorithm.**

The Halting Problem

```
while (true)
    print “Hello World!”
```

```
print “Hello World!”
```

The Halting Problem



```
function check(n)
  {   // check if the number n is a prime
    var factor; // if the checked number is not a prime, this is its first factor
    var c;
    factor = 0;
    // try to divide the checked number by all numbers till its square root
    for (c=2 ; (c <= Math.sqrt(n)) ; c++)
       {
         if (n%c == 0)    // is n divisible by c ?
            { factor = c;  break}
       }
    return (factor);
  }   // end of check function

function communicate()
  { // communicate with the user
    var i;        // i is the checked number
    var factor; // if the checked number is not a prime, this is its first factor
    i = document.primetest.number.value;       // get the checked number
    // is it a valid input?
    if ((isNaN(i)) || (i <= 0) || (Math.floor(i) != i))
      {alert ("The checked object should be a whole positive number")} ;
    else
      {
       factor = check (i);
       if (factor == 0)
          {alert (i + " is a prime")} ;
       else
          {alert (i + " is not a prime, " + i + "=" + factor + "X" + i/factor) }
      }
  }     // end of communicate function
```

Program

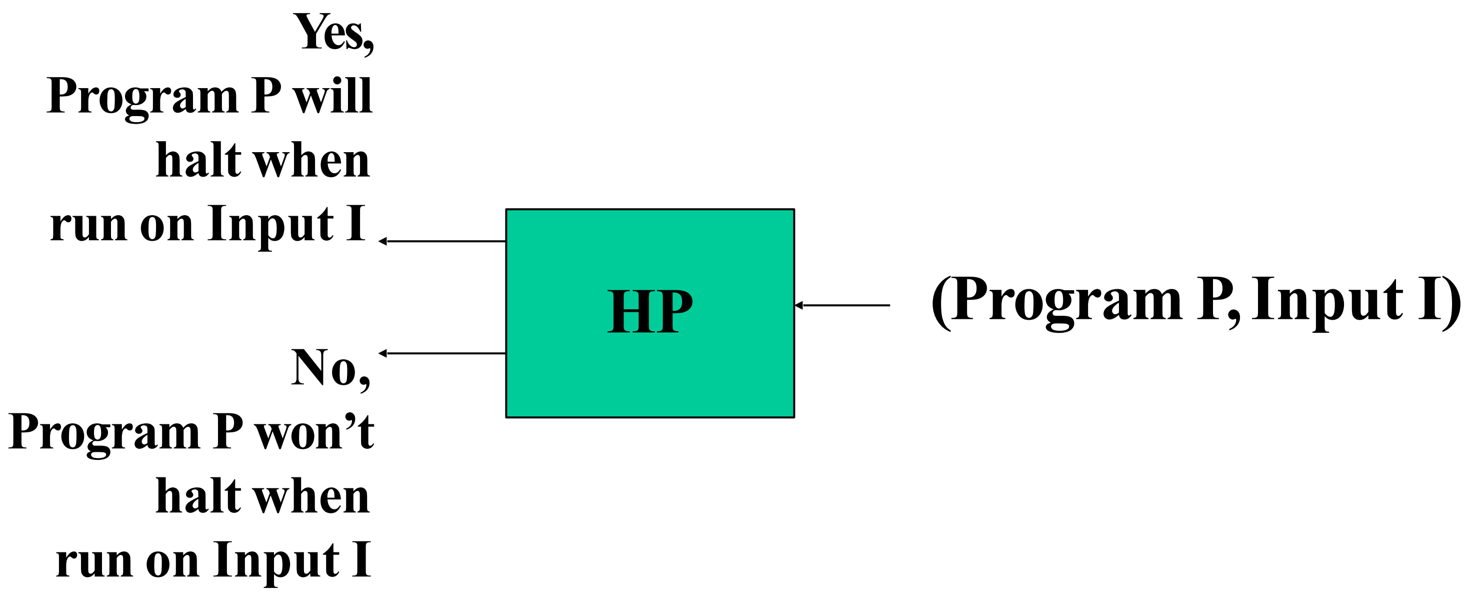
Decider

Input

Decider decides whether the program halts or loops forever?

Undecidable Problems

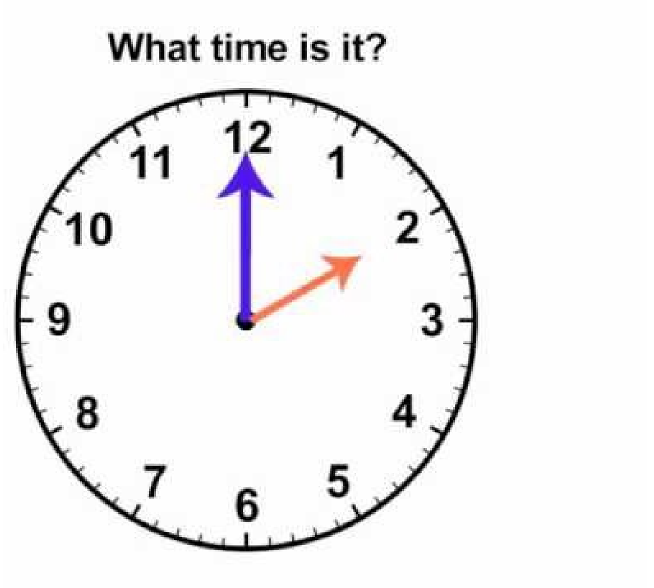
- Example: The Halting Problem



- **Determining whether a program loops or halts on an input**

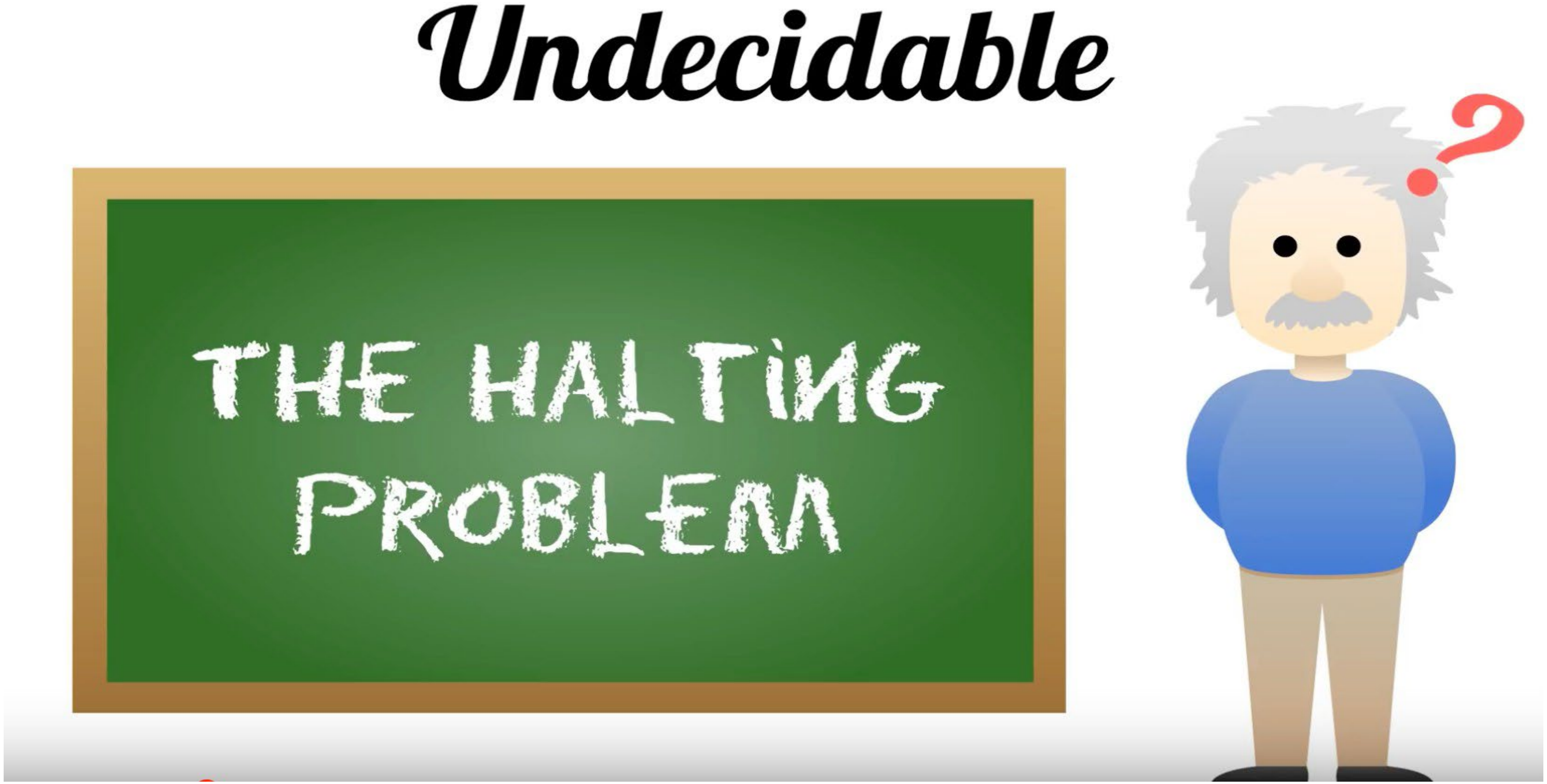
The Halting Problem

Halts



Loops

- One of them has to be correct, but the other will also be wrong
- Can we make a program that wont give a wrong answer?
- Use time to decide? 1 sec? 1 mint? 1 hour? 1 day?…


Undecidable
THE HALTING PROBLEM

The End

- This is an appropriate place to end a course on algorithms…

Optional Reading Assignment

If you want to know more about NP- Completeness, read Chapter 34

AUBURN UNIVERSITY