Monash University
Faculty of Information Technology

Lectures 22-23
NP-completeness

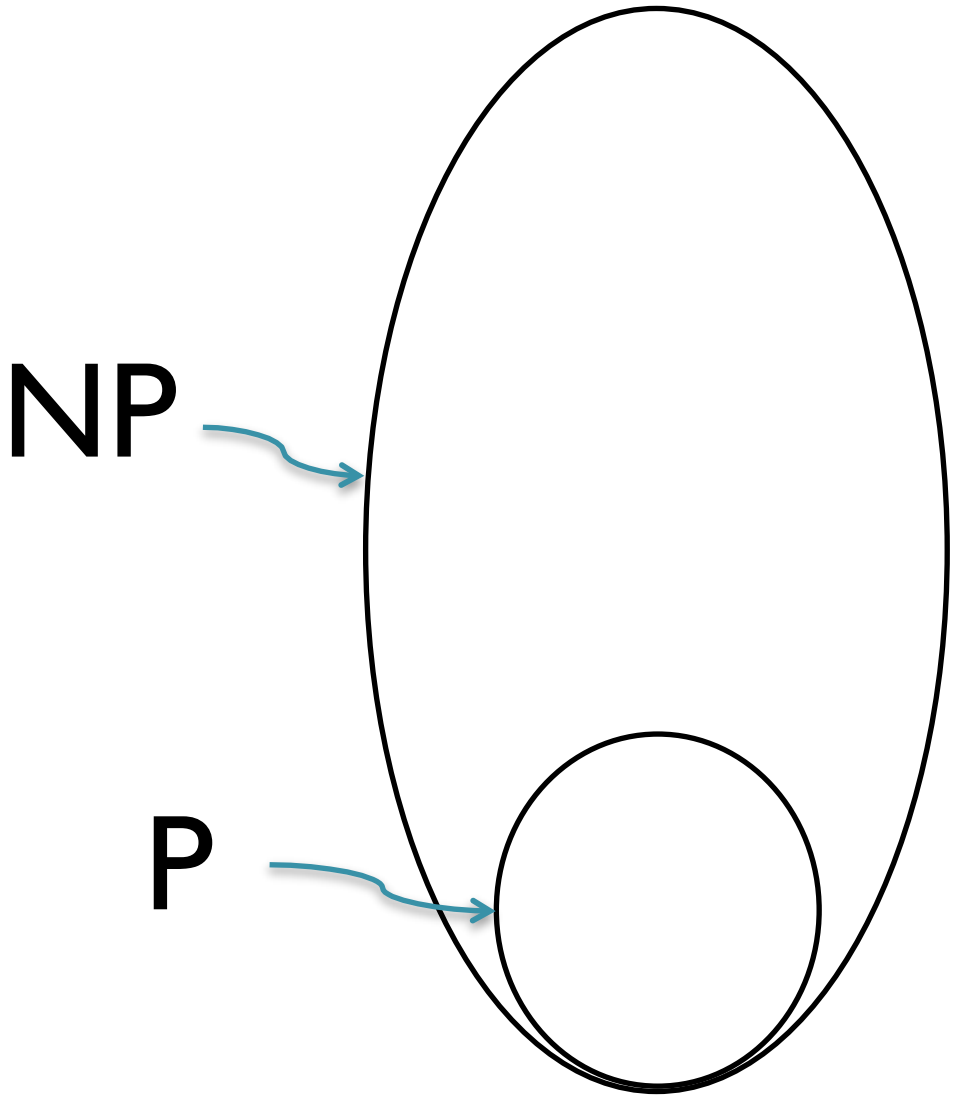
Slides by Graham Farr (2012)

FIT2014 Theory of Computation

Overview

- NP-completeness
- The Cook-Levin Theorem
- Reductions to SATISFIABILITY
- Proving NP-completeness
- Implications of NP-completeness

If $P \neq NP$:



In NP, not known to be in P:

SATISFIABILITY, 3-SAT,
HAMILTONIAN CIRCUIT,
3-COLOURABILITY,
VERTEX COVER,
INDEPENDENT SET, …

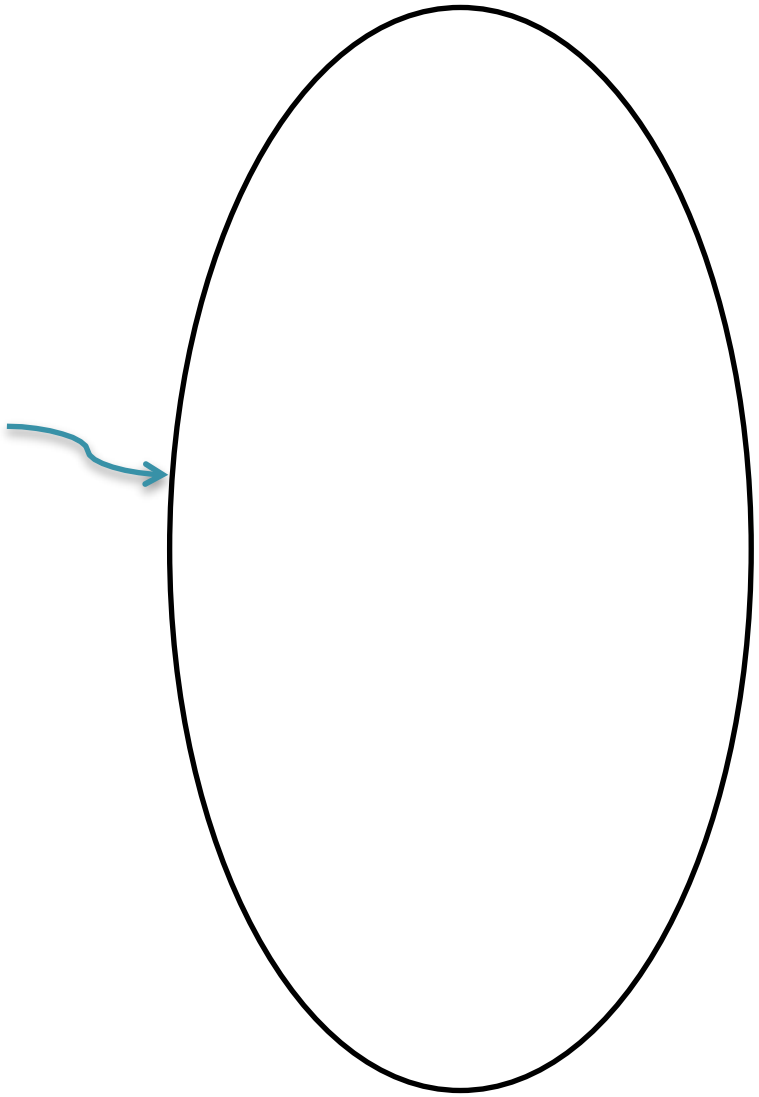
GRAPH ISOMORPHISM,
INTEGER FACTORISATION, …

In P:

2-SAT,
EULERIAN CIRCUIT,
2-COLOURABILITY,
PRIMES,
CONNECTED GRAPHS,
SHORTEST PATH,
Invertible matrices,
Context-free languages,
Regular languages, …

If P = NP:

P = NP



SATISFIABILITY, 3-SAT,
HAMILTONIAN CIRCUIT,
3-COLOURABILITY,
VERTEX COVER,
INDEPENDENT SET, …

GRAPH ISOMORPHISM,
INTEGER FACTORISATION, …

2-SAT,
EULERIAN CIRCUIT,
2-COLOURABILITY,
PRIMES,
CONNECTED GRAPHS,
SHORTEST PATH,
Invertible matrices,
Context-free languages,
Regular languages, …

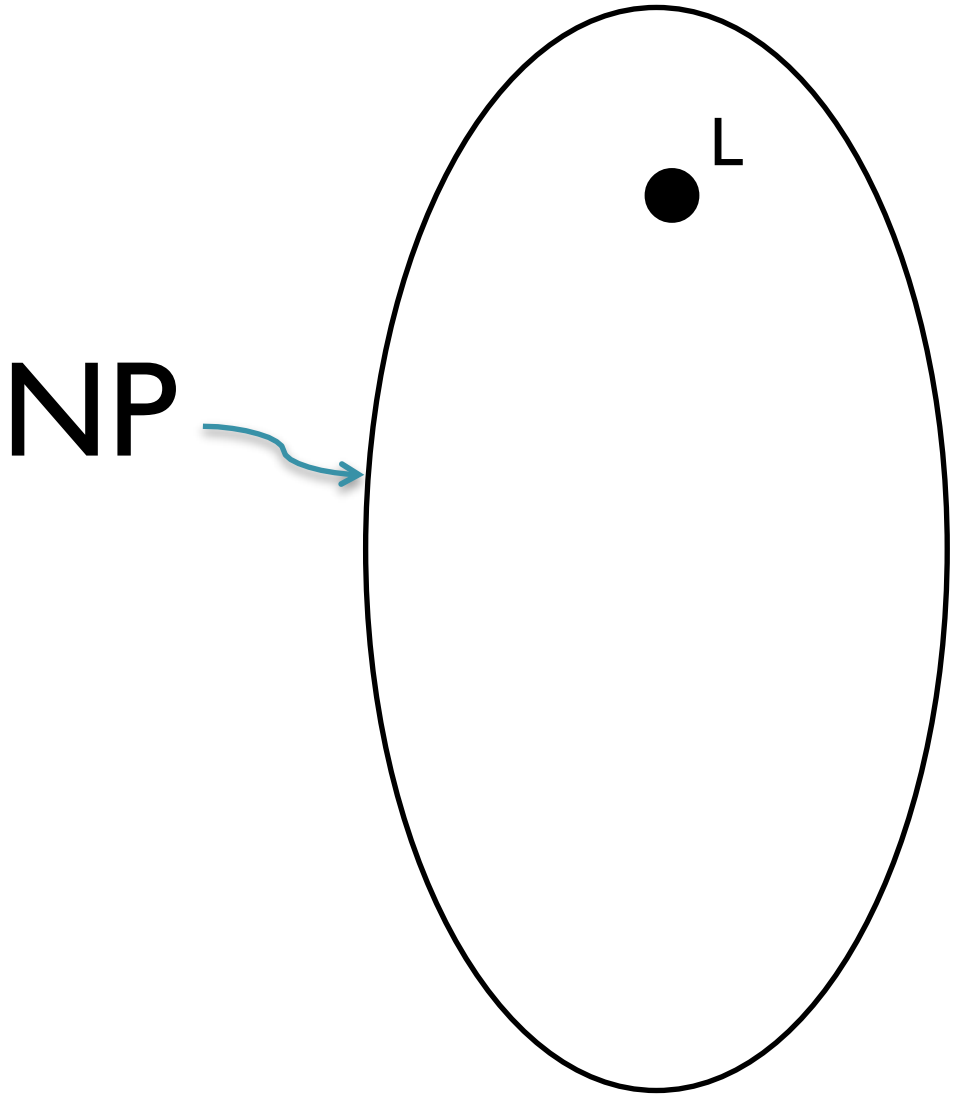
P and NP

- Languages in P are "easy" … or, at least, they have "efficient" (polynomial time) deciders.
- Languages outside P are "hard".
  - They don't have "efficient" (polynomial time) deciders.
- Languages in NP : membership is "easy" (polynomial time) to verify.
- NP contains many languages of great practical importance.
- What are the "hardest" languages in NP?
- Use polynomial-time reduction to compare languages …

NP-completeness

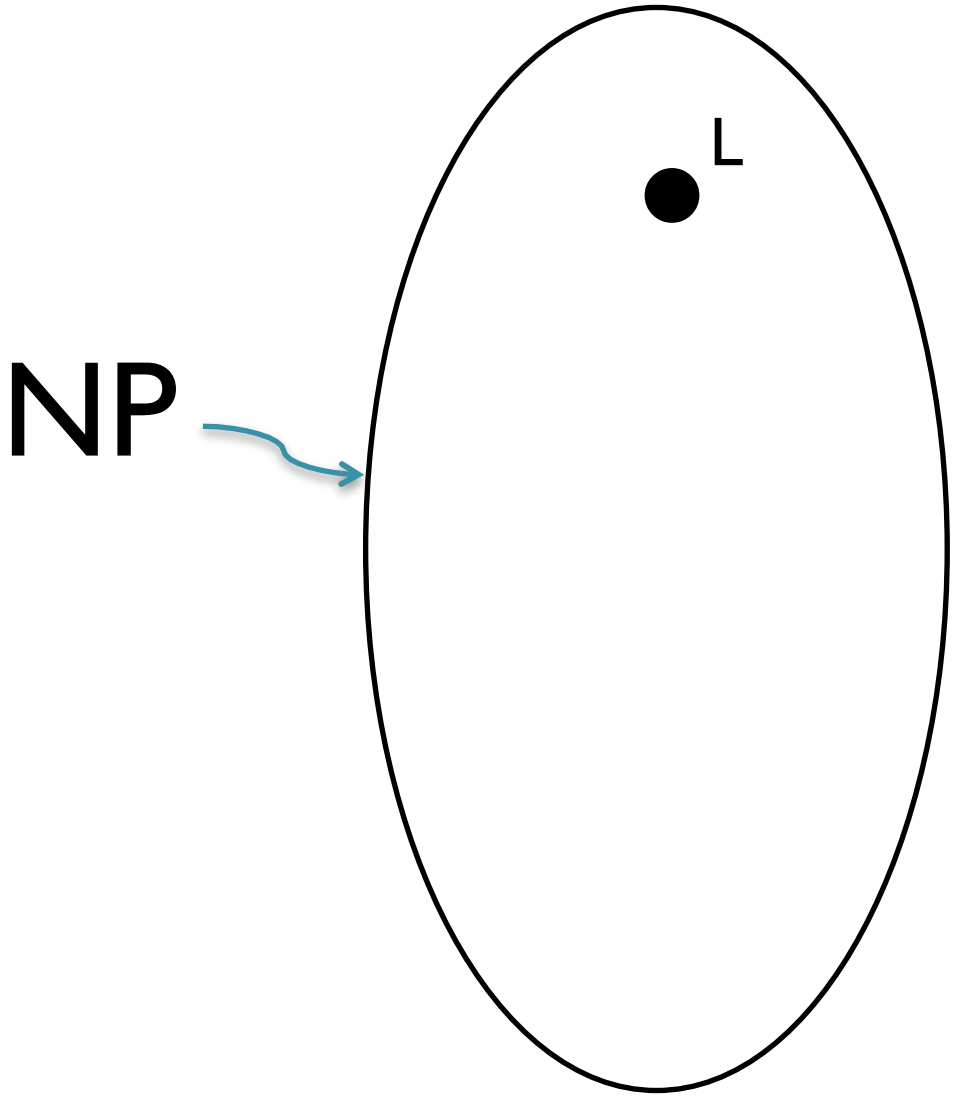
- A language L is ***NP-complete*** if
  - L is in NP, and
  - every language in NP is polynomial-time reducible to L.
    - … i.e.: for all L' in NP : $L' \leq_P L$.

NP-completeness



L is *NP-complete* because …

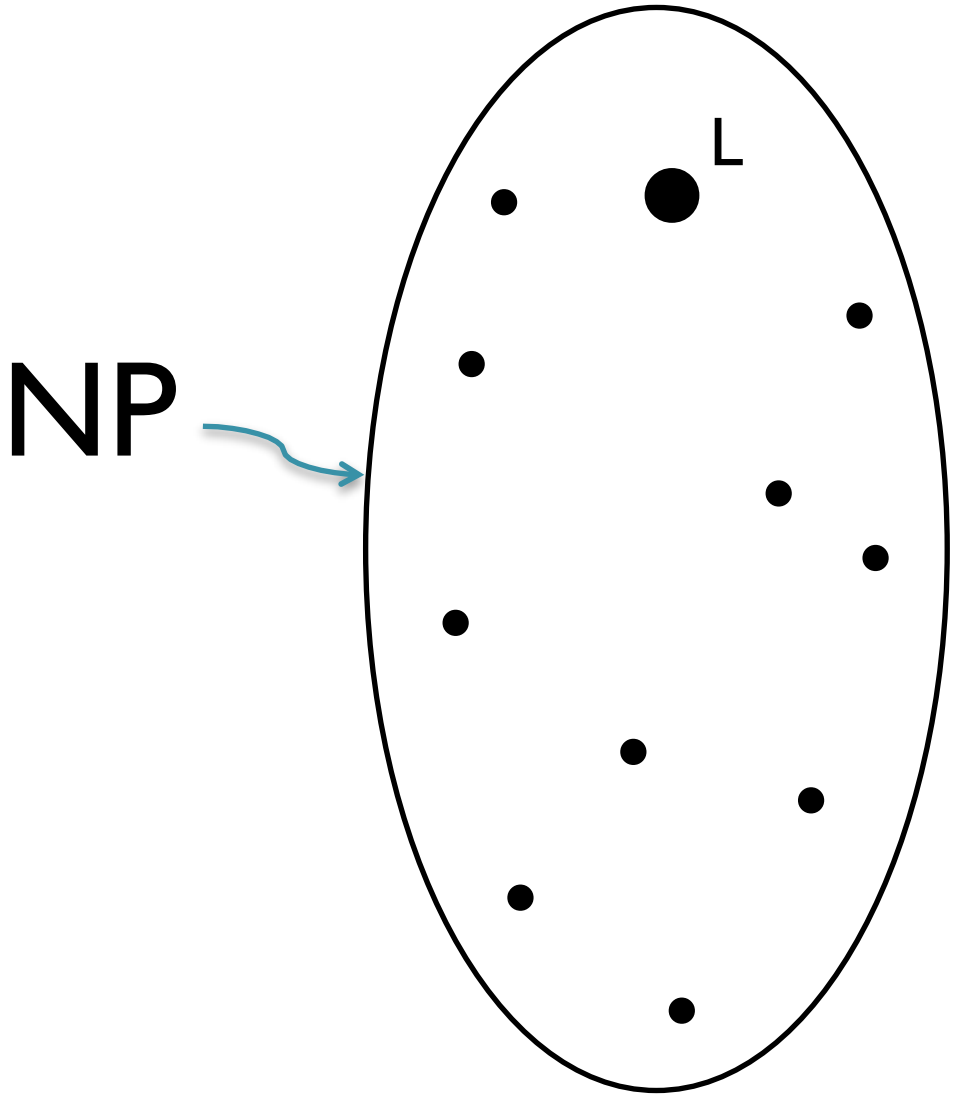
NP-completeness



L is *NP-complete* because …

L is in NP, and …

NP-completeness

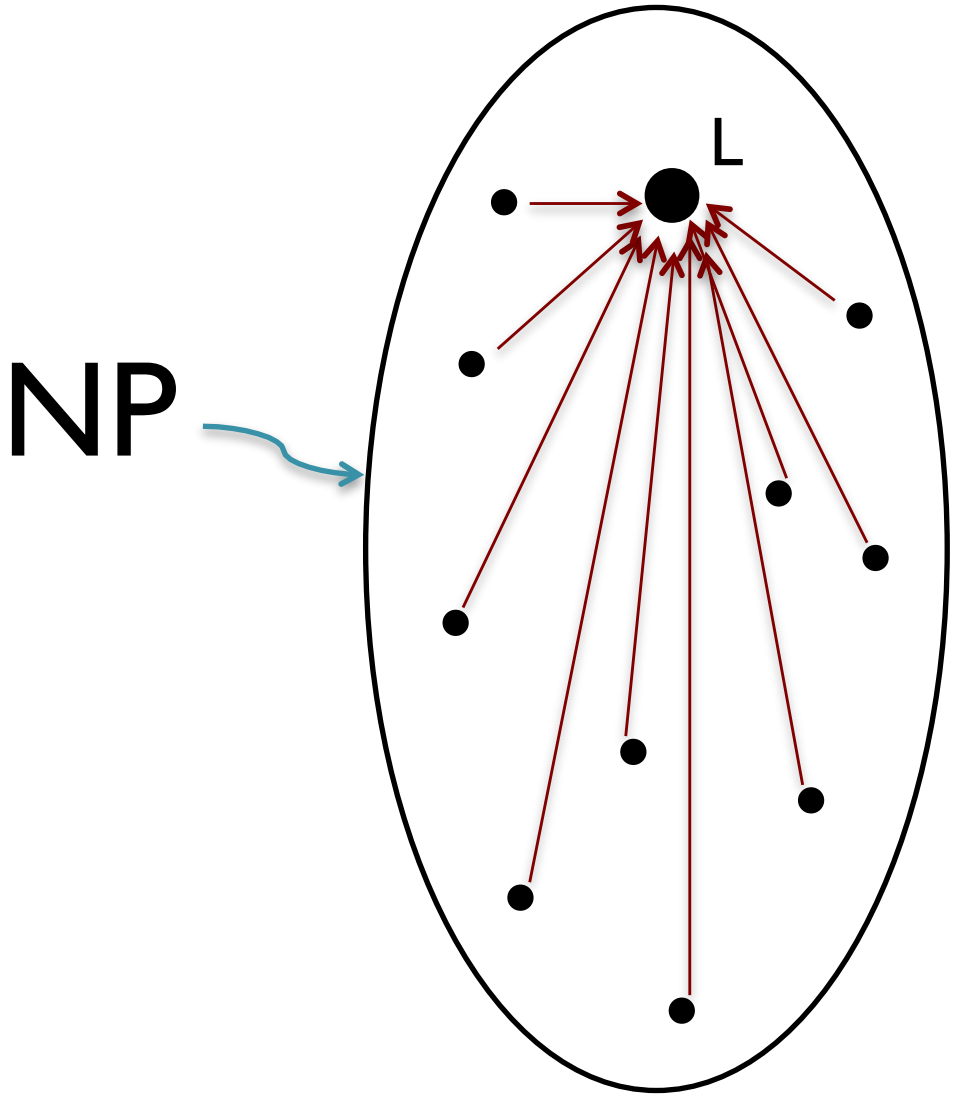


L is *NP-complete* because …

L is in NP, and …

everything in NP …

NP-completeness

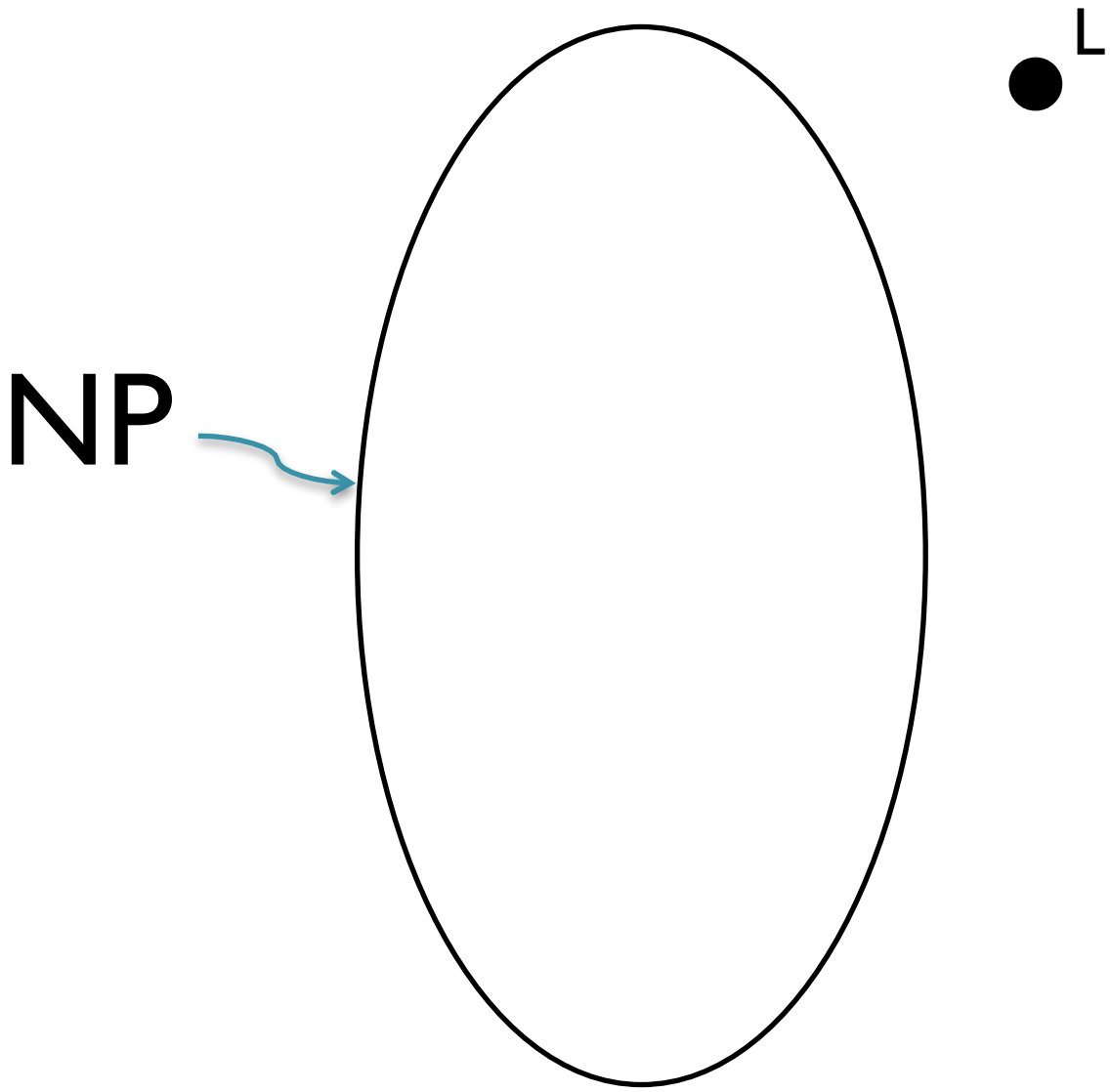


L is *NP-complete* because …

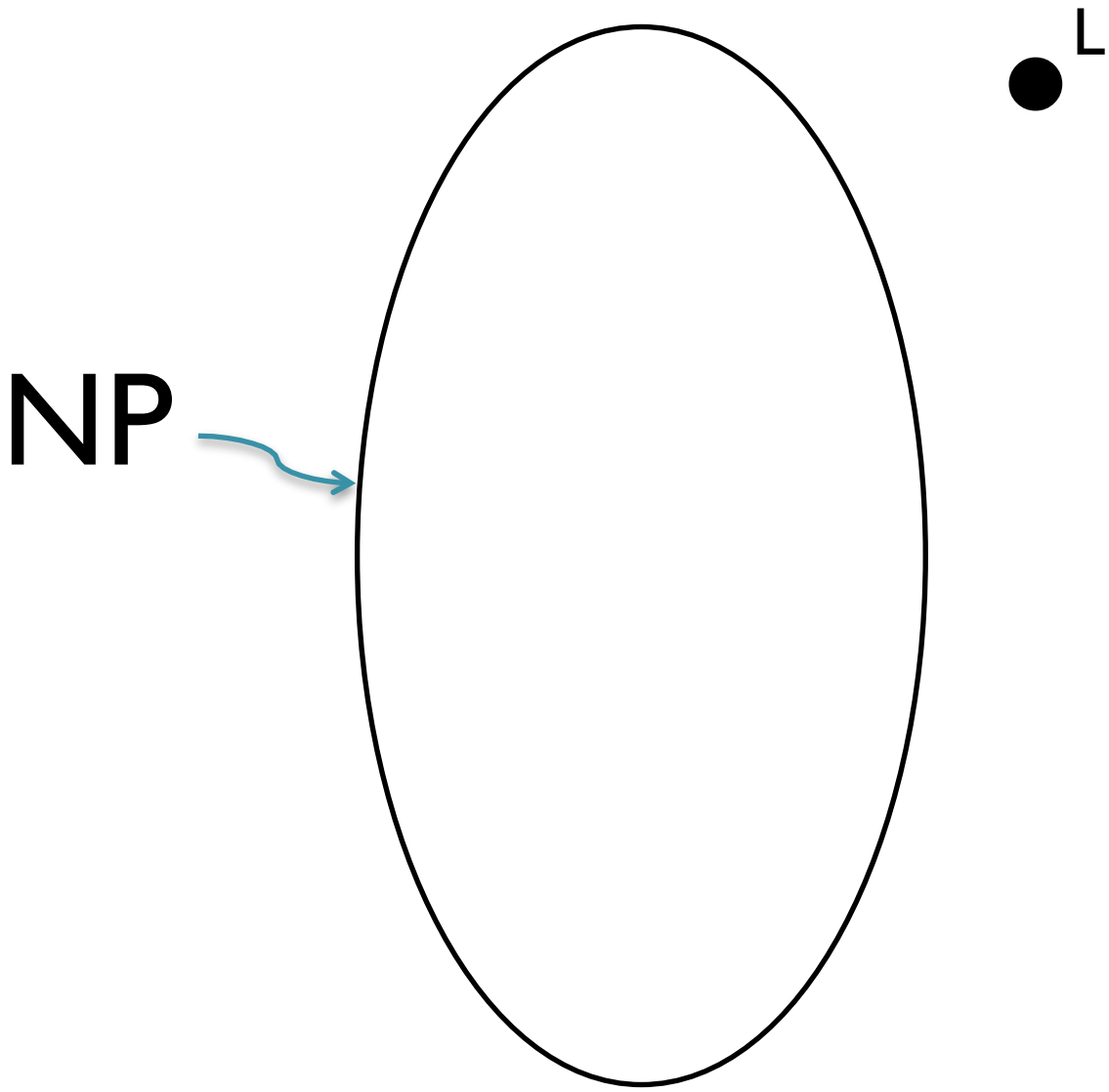
L is in NP, and …

everything in NP is polynomial-time reducible to L.

NP-completeness

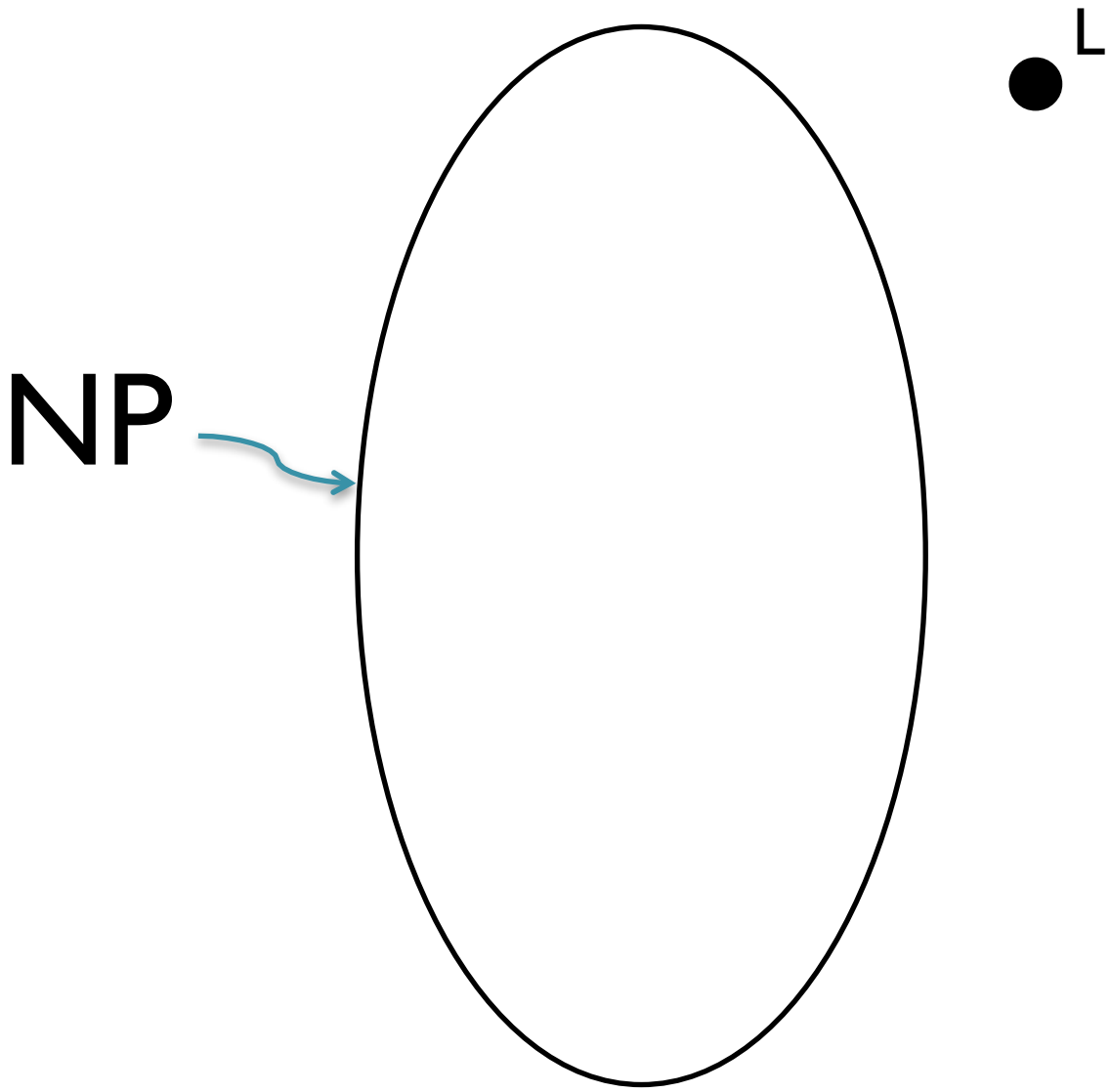


NP-completeness



L is ***not*** *NP-complete* because …

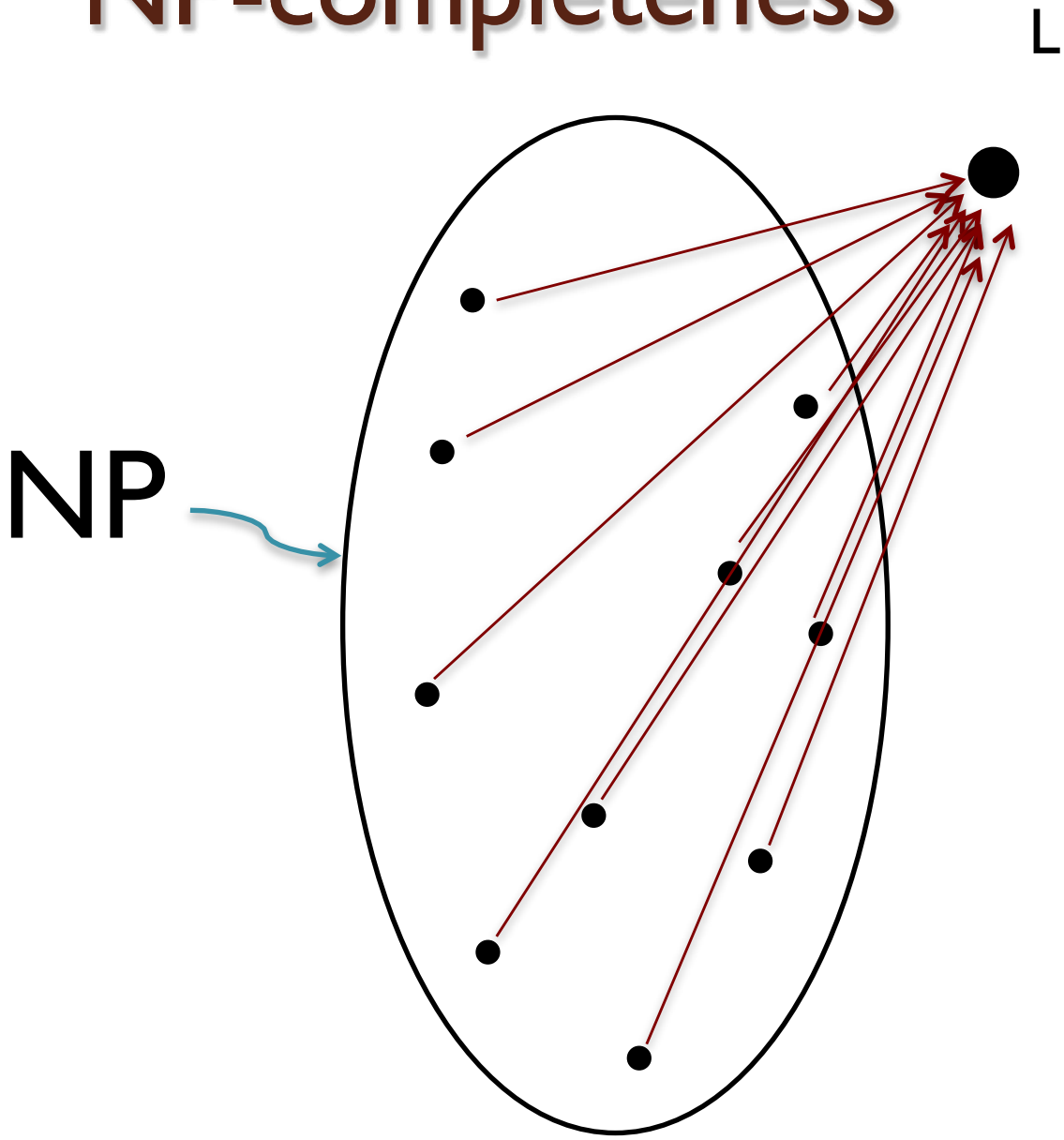
NP-completeness



L is ***not*** *NP-complete* because …

L is ***not*** in NP.

NP-completeness

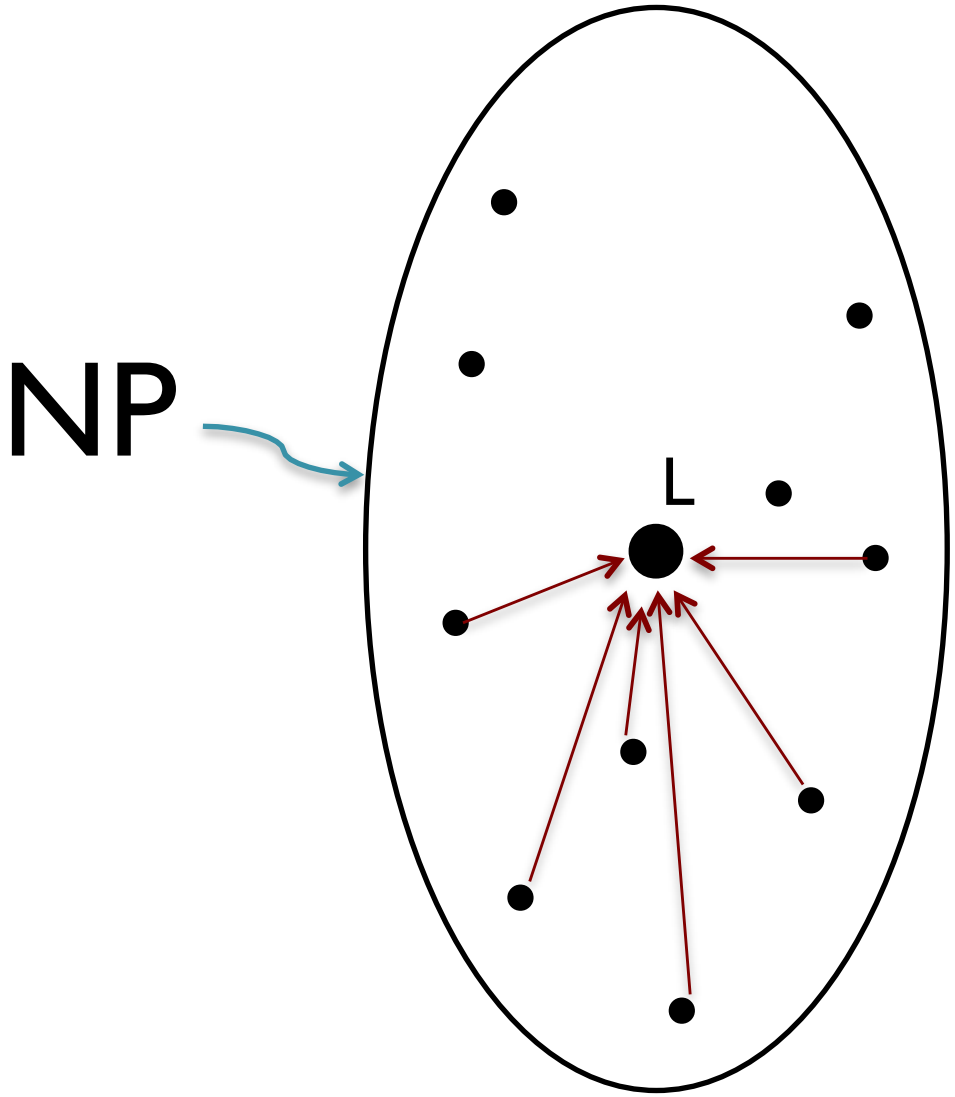


L is ***not*** *NP-complete* because …

L is ***not*** in NP.

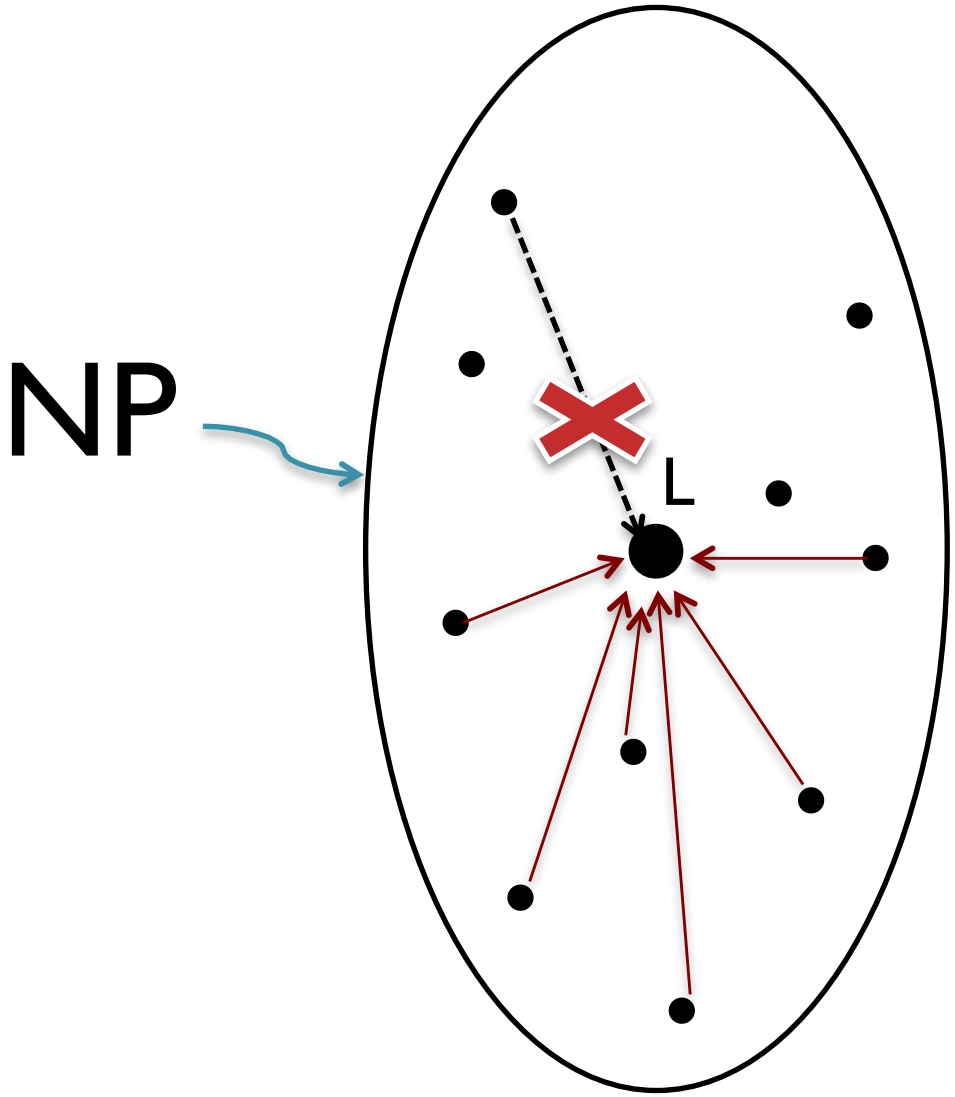
It doesn't matter if everything in NP is polynomial-time reducible to L.

NP-completeness



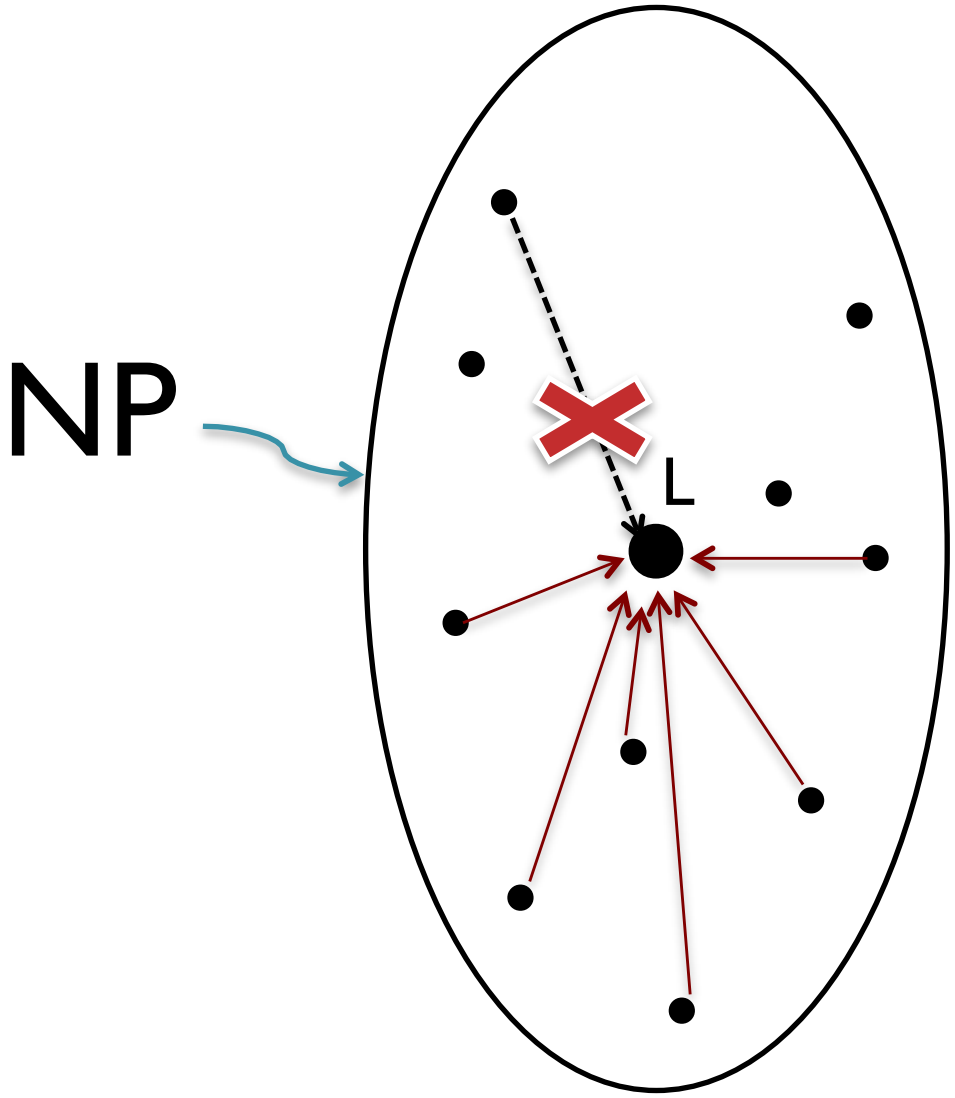
L is ***not*** NP-complete because …

NP-completeness



L is ***not*** NP-complete because …

NP-completeness



L is ***not*** NP-complete because …

not everything in NP is polynomial-time reducible to L.

NP-completeness

- If we think …
  "polynomial-time reducible to" = "easier than"
  … then an NP-complete language can be thought of as a hardest language in NP.
- But take care with this interpretation.
- "polynomial-time reducible to" = "easier, or not too much harder than"
- Polynomial-time reducibility does not give a precise comparison of time complexities.

NP-completeness

Theorem

If L is NP-complete, and there is a polynomial-time decider for L, then P = NP.

Proof.

Suppose L is as given in the statement of the Theorem.

We know $P \subseteq NP$.

It remains to show that, under our assumptions,

$NP \subseteq P$.

Then we'll know that P = NP.

NP-completeness

Let L' be any language in NP. We will show it is also in P.

Since L is NP-complete, any language in NP is polynomial-time reducible to L.

Therefore $L' \leq_P L$.

But we know that, if $L' \leq_P L$ and L is in P,

then L' is in P too. (Theorem near end of Lecture 21.)

So L' is in P.

We have shown that NP $\subseteq$ P, which completes the proof.

End of proof.

Cook-Levin Theorem

Our first NP-complete language:

SATISFIABILITY:

the set of satisfiable Boolean expressions in CNF

Cook-Levin Theorem

SATISFIABILITY is NP-complete.

History: S. Cook (1971), L. Levin (1972)

Cook-Levin Theorem

To prove the Cook-Levin Theorem, we must show:

- SATISFIABILITY is in NP
  - the easy part
  - Given Boolean expression ϕ in CNF:
  - Certificate = truth assignnment to variables of ϕ.
  - Verification: check that each clause is satisfied …
  - Prove verification works and takes polynomial time.
- For every L in NP,

  $$L \leq_P \text{SATISFIABILITY}.$$

  - much harder.

Reducing to SATISFIABILITY

To understand how any language in NP can be polynomial-time reduced to SAT, we will look at some specific languages.

Reducing to SATISFIABILITY

Suppose we have a language which we want to reduce to SAT.

Approach:

1. Introduce Boolean variables to describe the parts of the certificate.
2. (Possibly introduce other variables to help describe the conditions under which a certificate is valid.)
3. Translate the **rules of the language** into CNF.
4. Put it all together as an algorithm.

Reducing to SATISFIABILITY

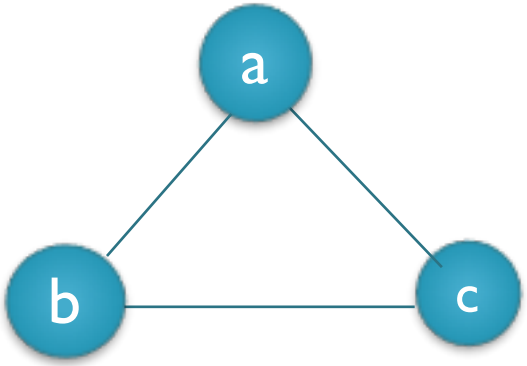
Let's do it for the following language.

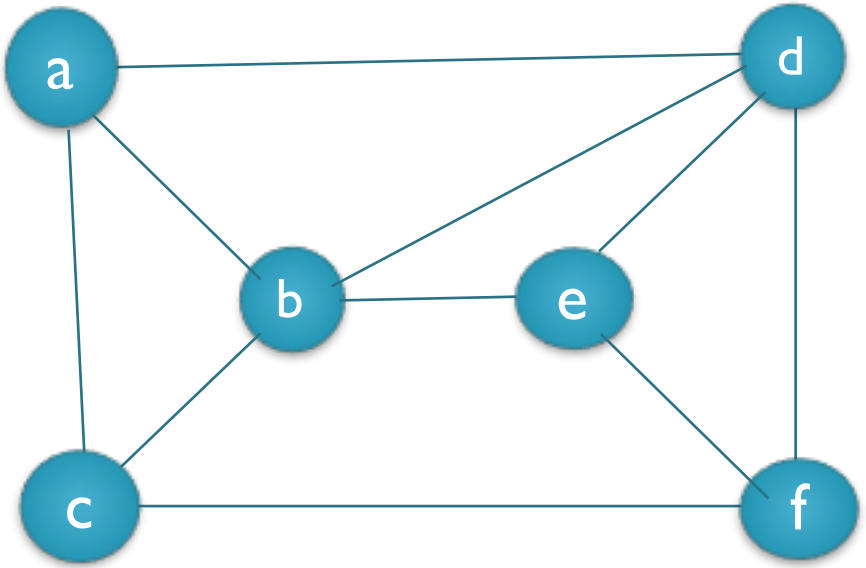
PARTITION INTO TRIANGLES:

the set of graphs G such that the vertex set of G can be partitioned into 3-sets (i.e., sets of size 3) such that each of these 3-sets induces a triangle in G.

Reducing to SATISFIABILITY

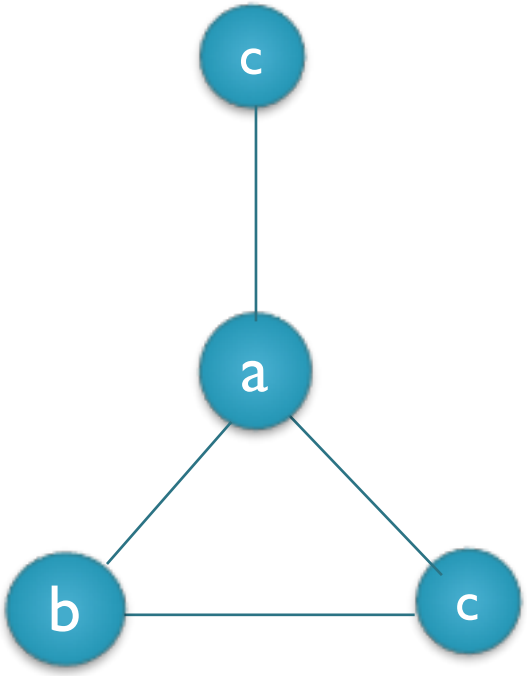
Some members of PARTITION INTO TRIANGLES:

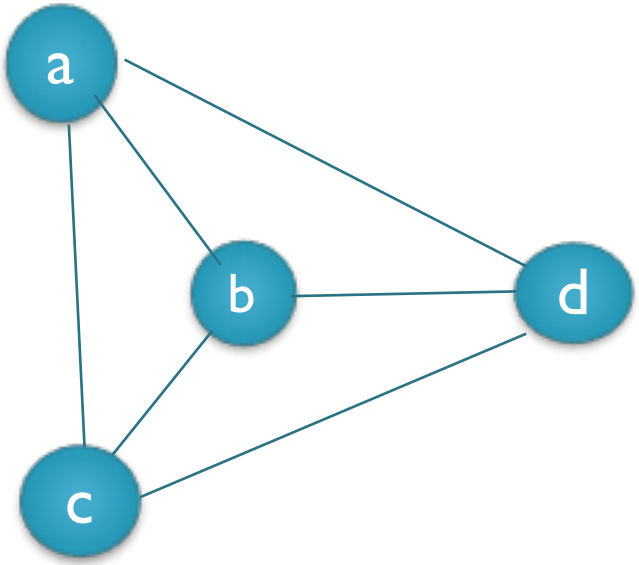




Reducing to SATISFIABILITY

Some **non-members** of PARTITION INTO TRIANGLES:





Reducing to SATISFIABILITY

PARTITION INTO TRIANGLES

Certificate:

- state which triangles, in the graph, are in the partition

Using Boolean variables:

- for each triangle T,

$$x_T = \begin{cases} \text{True}, & \text{if } T \text{ is in the partition;} \\ \text{False}, & \text{otherwise.} \end{cases}$$

Reducing to SATISFIABILITY

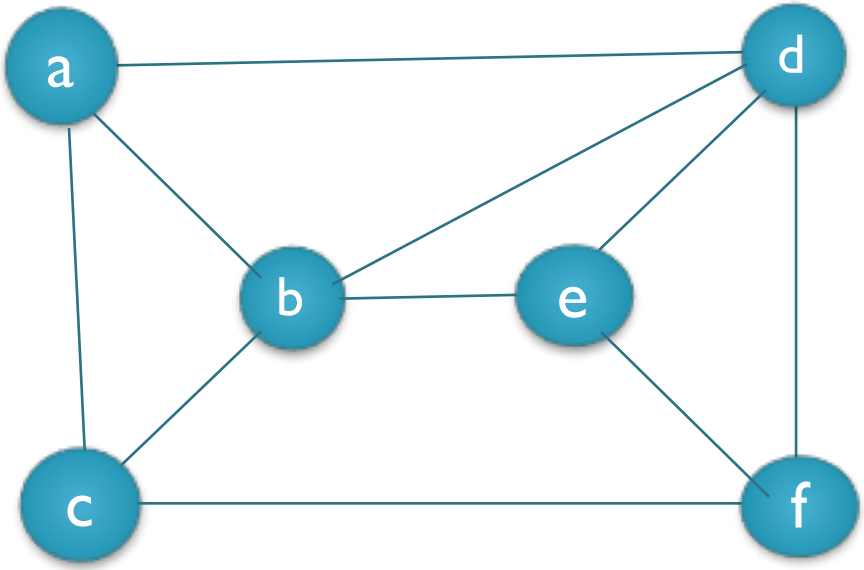
Variables for this graph:

x_{abc}

x_{abd}

x_{bde}

x_{def}



In general,

triangles = $O(n^3)$

Reducing to SATISFIABILITY

PARTITION INTO TRIANGLES

Rules of the language:

The set of triangles must form a partition of $V(G)$.

I.e.:

- every vertex belongs to at least one triangle, and
- no vertex belongs to more than one triangle.

Let's look at each of these in turn.

Reducing to SATISFIABILITY

"Every vertex belongs to at least one triangle."

Express this in terms of our variables.

For each vertex: at least one of the triangles at that vertex must be included in the partition

For each vertex, use a clause

$$x_{T1} \vee x_{T2} \vee \ldots \vee x_{Tk}$$

where T1, T2, …, Tk are the triangles at the vertex.

# of these clauses = n (i.e., # vertices of graph)

Reducing to SATISFIABILITY

Vertex a:

$x_{abc} \vee x_{abd}$

Vertex b:

$x_{abc} \vee x_{abd} \vee x_{bde}$

Vertex c:

x_{abc}

Vertex d:

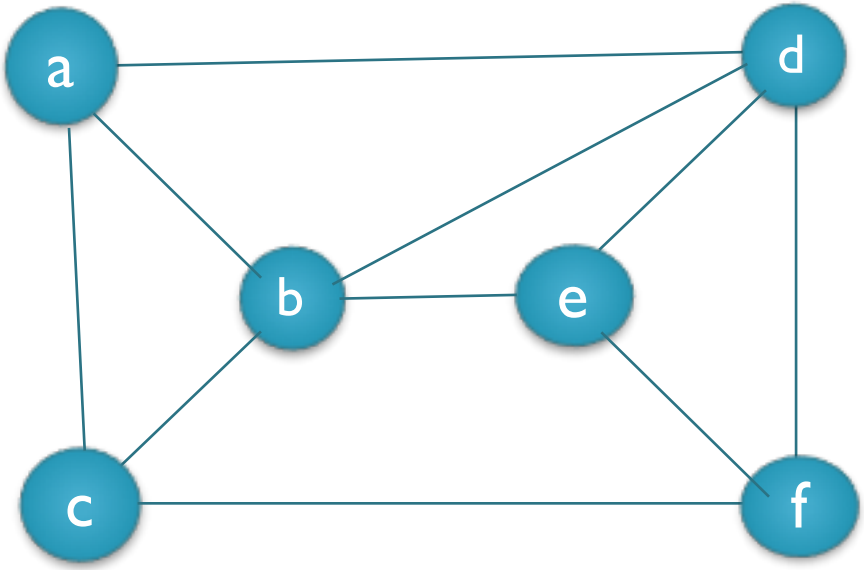
$x_{abd} \vee x_{bde} \vee x_{def}$

Vertex e:

$x_{bde} \vee x_{def}$

Vertex f:

x_{def}



Reducing to SATISFIABILITY

"No vertex belongs to more than one triangle."

Express this in terms of our variables.

For each vertex, and each pair of triangles at that vertex: at least one of the triangles in this pair must **not** be included in the partition

For each vertex, and each pair of triangles T_i, T_j at the vertex, use a clause

$$\neg x_{Ti} \vee \neg x_{Tj}$$

$$\#\text{ of these clauses} \leq n \cdot \left(d(d-1)/2 \right)^2 \quad \text{(where d = degree)}$$

$$= O(n^5)$$

Reducing to SATISFIABILITY

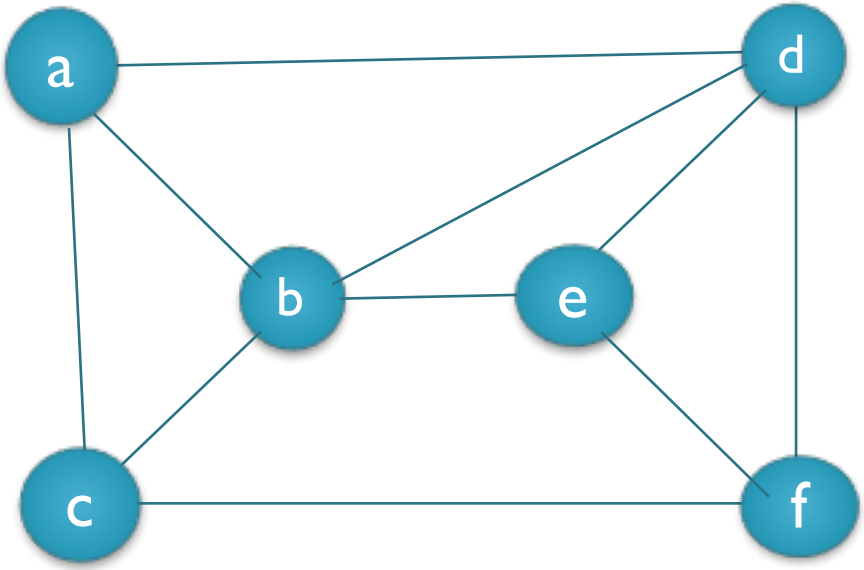
$\neg x_{abc} \vee \neg x_{abd}$

$\neg x_{abc} \vee \neg x_{bde}$

$\neg x_{abd} \vee \neg x_{bde}$

$\neg x_{abd} \vee \neg x_{def}$

$\neg x_{bde} \vee \neg x_{def}$



Reducing to SATISFIABILITY

We now have clauses for:

- every vertex belongs to at least one triangle, and
- no vertex belongs to more than one triangle.

Take the conjunction of all clauses.

This gives a Boolean formula ϕ which is satisfiable if and only if the original graph has a partition into triangles.

Specify the mapping from G to ϕ as an algorithm, and show it's a polynomial-time reduction.

Reducing to SATISFIABILITY

ϕ =

$(x_{abc} \vee x_{abd})$

$\wedge (x_{abc} \vee x_{abd} \vee x_{bde})$

$\wedge (x_{abc})$

$\wedge (x_{abd} \vee x_{bde} \vee x_{def})$

$\wedge (x_{bde} \vee x_{def})$

$\wedge (x_{def})$

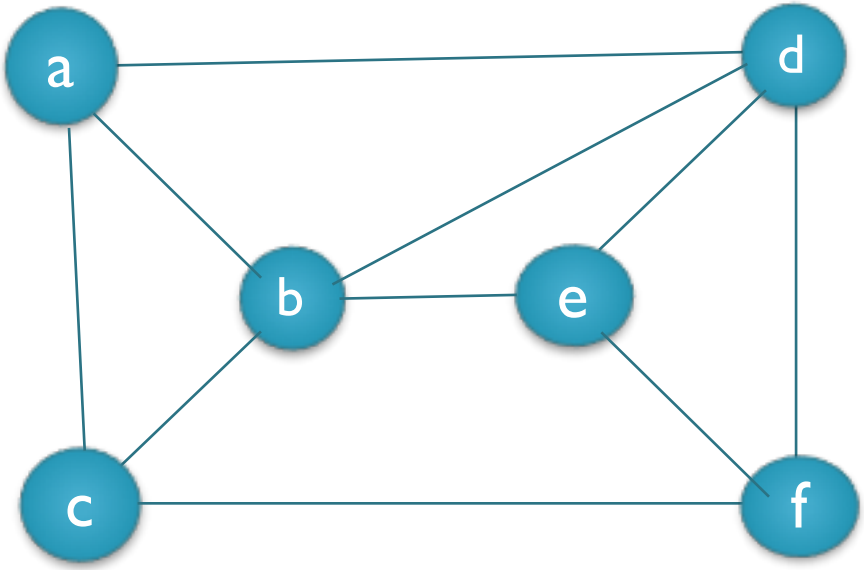
$\wedge (\neg x_{abc} \vee \neg x_{abd})$

$\wedge (\neg x_{abc} \vee \neg x_{bde})$

$\wedge (\neg x_{abd} \vee \neg x_{bde})$

$\wedge (\neg x_{abd} \vee \neg x_{def})$

$\wedge (\neg x_{bde} \vee \neg x_{def})$



Reducing to SATISFIABILITY

Describe the reduction as an algorithm.

Input: Graph G

For each triangle T of G: create new variable x_T

For each vertex v of G: make the clause

$$x_{T1} \vee x_{T2} \vee \ldots \vee x_{Tk}$$

where $T1, T2, \ldots, Tk$ are the triangles at v.

For each pair of triangles Ti, Tj which share a vertex,

make a clause $\neg x_{Ti} \vee \neg x_{Tj}$

ϕ := conjunction of all these clauses.

Output ϕ

Reducing to SATISFIABILITY

Time complexity?

Main factor: # pairs of triangles that share a vertex

= $O(n^5)$

For each such pair, a small amount of work needs doing. Looks like time $O(1)$ or $O(n)$ …

So the time complexity is polynomial.

Reducing to SATISFIABILITY

Other languages can be polynomial-time reduced to SAT by a similar approach.

3-COLOURABILITY

Try:

CUBIC SUBGRAPH:

the set of graphs with a subgraph consisting entirely of vertices of degree 3

HAMILTONIAN CIRCUIT

Reducing to SATISFIABILITY

Experience with reducing to SAT *suggests* that any language in NP can be polynomial-time reduced to SAT. (This is the Cook-Levin Theorem.)

But how to *prove* it?

Need a generic way of reducing from any specific language in NP to SAT.

Reducing to SATISFIABILITY

Very sketchy overview of proof of Cook-Levin Theorem:

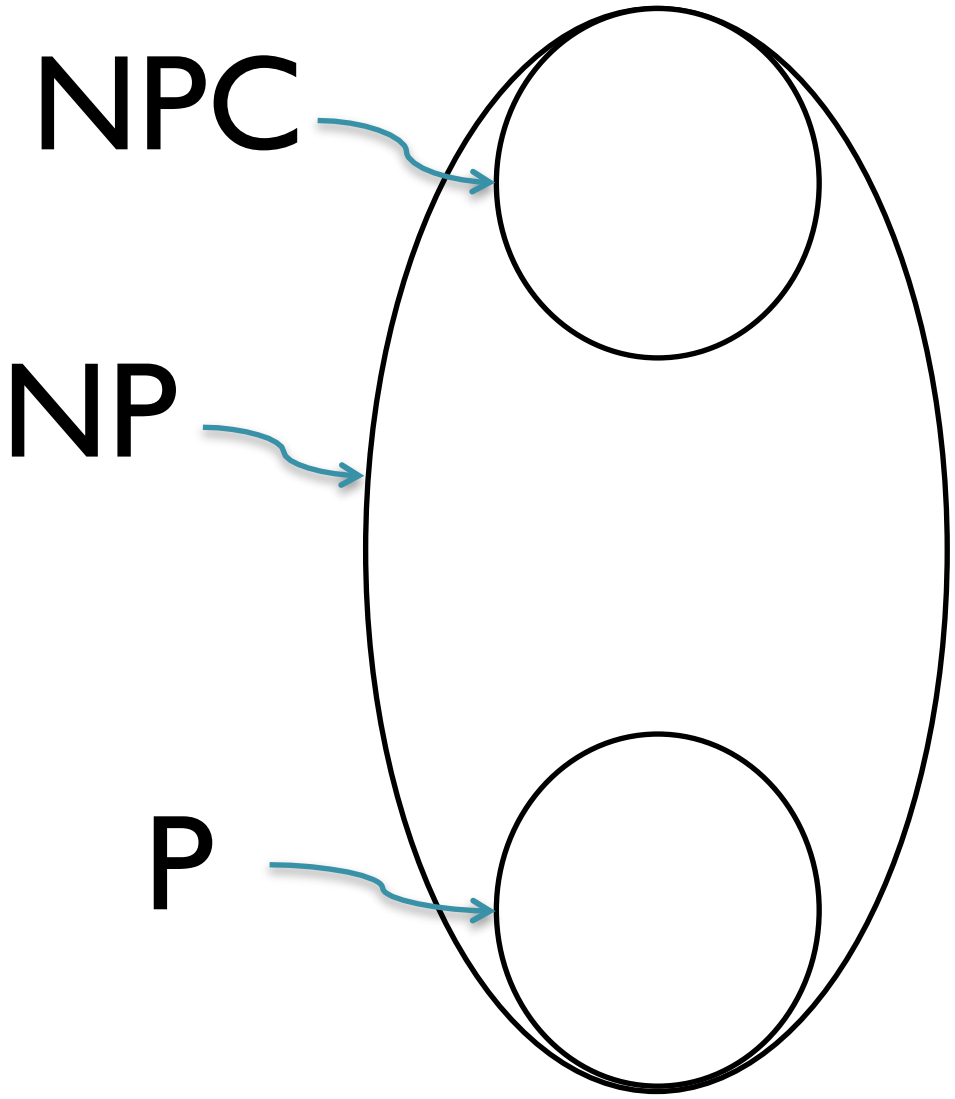
Given L in NP, let M be a polynomial-time verifier for L.

Create variables for the certificate (considered as a binary string), and more variables to describe all the components of M at all possible timesteps.

Make clauses to capture the working of the verifier. These express the allowed transitions of the machine, and forbid any others. (See Tute 5, Q10.)

Show that the construction is indeed the desired polynomial-time reduction.

If $P \neq NP$:



In NP, not known to be in P:

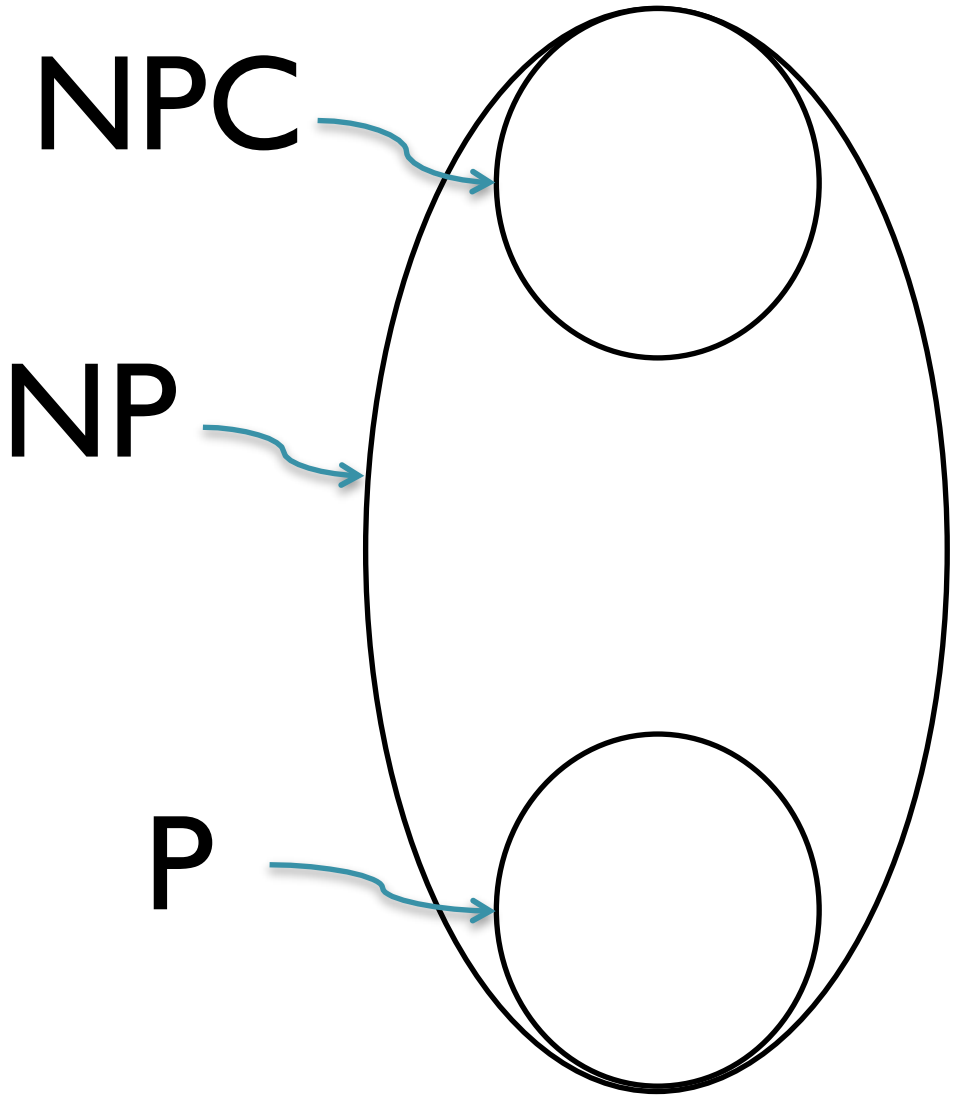
SATISFIABILITY, 3-SAT,
HAMILTONIAN CIRCUIT,
3-COLOURABILITY,
VERTEX COVER,
INDEPENDENT SET, …

GRAPH ISOMORPHISM,
INTEGER FACTORISATION, …

In P:

2-SAT,
EULERIAN CIRCUIT,
2-COLOURABILITY,
PRIMES,
CONNECTED GRAPHS,
SHORTEST PATH,
Invertible matrices,
Context-free languages,
Regular languages, …

If $P \neq NP$:



In NP, not known to be in P:

NPC SATISFIABILITY, 3-SAT,
HAMILTONIAN CIRCUIT,
3-COLOURABILITY,
VERTEX COVER,
INDEPENDENT SET, …

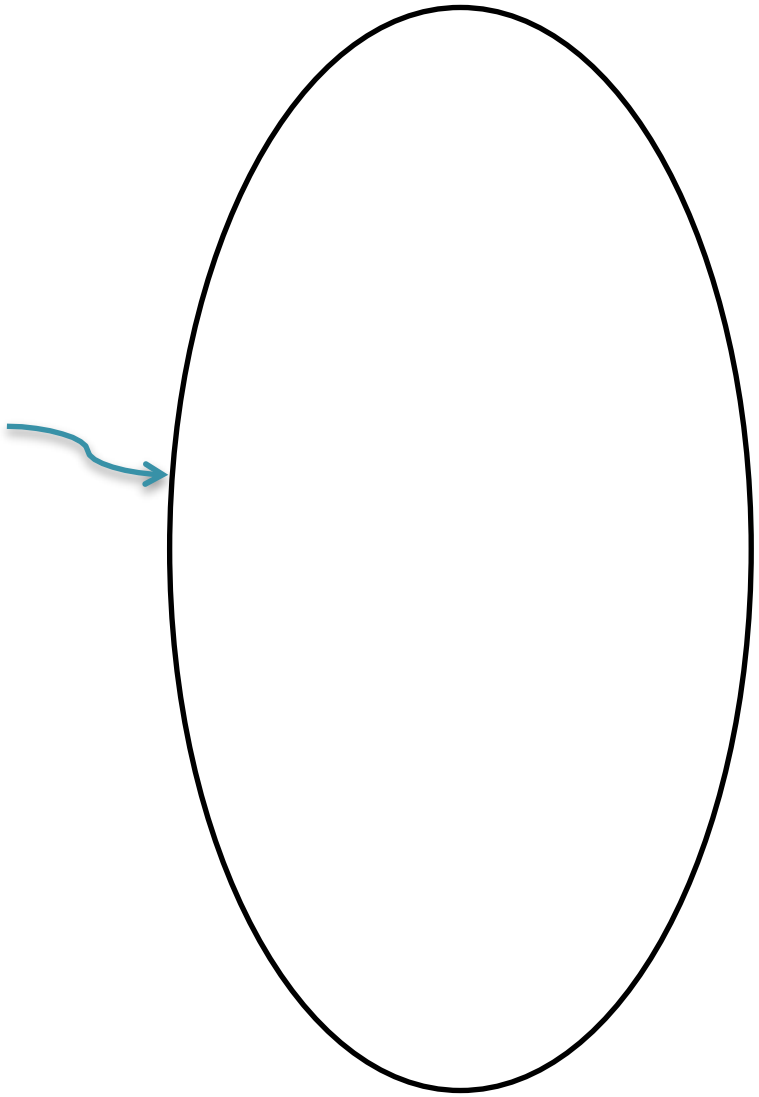
GRAPH ISOMORPHISM,
INTEGER FACTORISATION, …

In P:

2-SAT,
EULERIAN CIRCUIT,
2-COLOURABILITY,
PRIMES,
CONNECTED GRAPHS,
SHORTEST PATH,
Invertible matrices,
Context-free languages,
Regular languages, …

If P = NP:

P = NP
= NPC



SATISFIABILITY, 3-SAT,
HAMILTONIAN CIRCUIT,
3-COLOURABILITY,
VERTEX COVER,
INDEPENDENT SET, …

GRAPH ISOMORPHISM,
INTEGER FACTORISATION, …

2-SAT,
EULERIAN CIRCUIT,
2-COLOURABILITY,
PRIMES,
CONNECTED GRAPHS,
SHORTEST PATH,
Invertible matrices,
Context-free languages,
Regular languages, …

Proving NP-completeness

Now that we have one NP-complete language, it is much easier to prove NP-completeness for many other languages.

Theorem

If L_1 is NP-complete, L_2 is in NP, and $L_1 \leq_P L_2$, then L_2 is NP-complete.

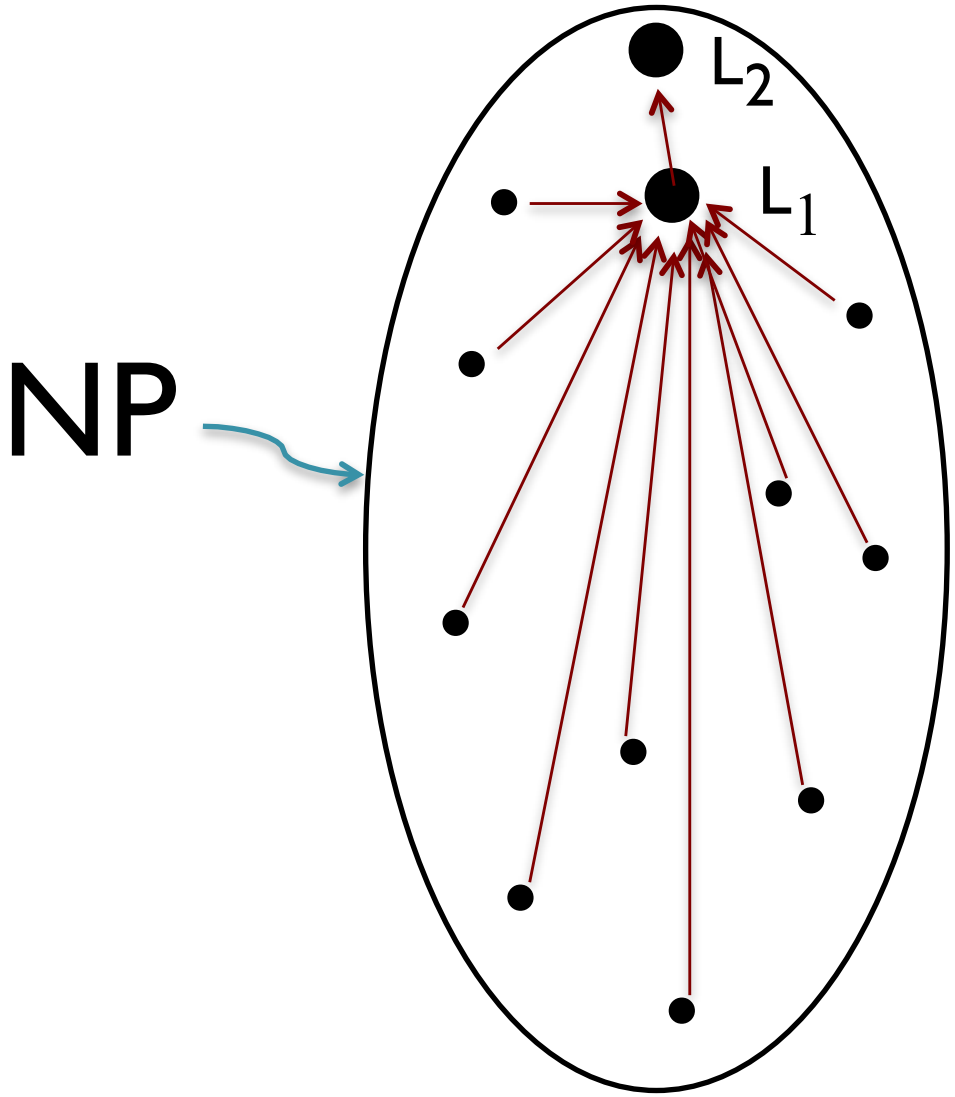
Proof

We are given that L_2 is in NP.

Let L be any language in NP. Then $L \leq_P L_1$ (by NP-completeness of L_1), and $L_1 \leq_P L_2$ (given), so by transitivity of $\leq_P$, we conclude $L \leq_P L_2$.

End of proof.

NP-completeness



Proving NP-completeness

So, to prove a language L is NP-complete, it is sufficient to

- show L is in NP, and
- show that another language known to be NP-complete polynomial-time-reduces to L.
  - just one reduction!

This approach didn't help us show SAT is NP-complete, since at that stage, we didn't know of any other NP-complete languages.

But now we can use SAT to show that other languages are NP-complete.

This approach first taken by R Karp (1972).

Proving NP-completeness

3-SAT

- belongs to NP
- SAT $\leq_P$ 3-SAT
- So it's NP-complete.

We now show that VERTEX COVER is NP-complete.

Easy to show it's in NP.

To prove completeness, we show that

3-SAT $\leq_P$ VERTEX COVER.

Proving NP-completeness

Given a Boolean expression Φ in CNF with exactly 3 literals in each clause, we must show how to construct a graph G_Φ and positive integer k_Φ such that

Φ is in 3-SAT if and only if

(G_Φ, k_Φ) is in VERTEX COVER.

Suppose Φ has

- variables $x_1, \ldots, x_n$
- clauses $C_1, \ldots, C_m$

Proving NP-completeness

Construction of G_Φ :

Vertices:

one for each literal: $x_1, \neg x_1, \ldots, x_n, \neg x_n$.

Three for each clause:

$C_{11}, C_{12}, C_{13}, \quad C_{21}, C_{22}, C_{23}, \quad \ldots, \quad C_{m1}, C_{m2}, C_{m3}$

Edges:

Join each literal to its "partner":

$(x_1, \neg x_1), \ldots, \quad (x_n, \neg x_n)$

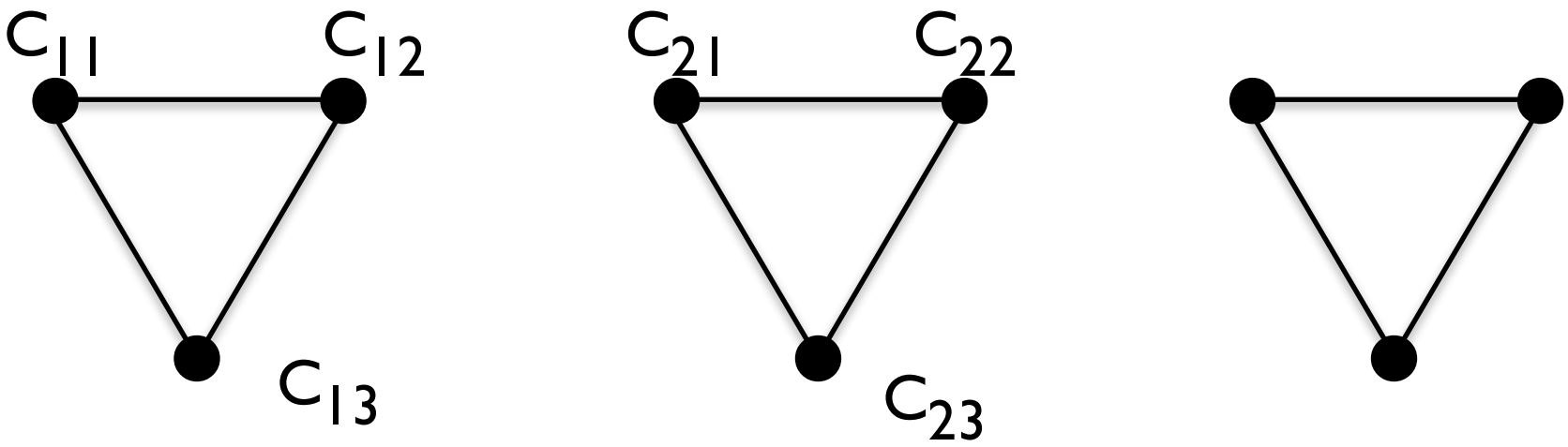
Join up all vertices corresponding to a clause:

$(C_{11}, C_{12}), \quad (C_{11} C_{13}), \quad (C_{12}, C_{13}), \ldots\ldots$

… so each clause is a separate triangle.

Proving NP-completeness

The story so far:



Looking at the graph so far: how large must a vertex cover be?

Proving NP-completeness

Observations:

Each "variable-edge" (for a pair of literals) must be covered, by at least one vertex of the VC.

Each "clause-triangle" must be covered, by at least two vertices of the VC.

All variable-edges and clause-triangles are disjoint from each other.

So all vertex covers must have size $\geq 2m+n$.

Set $k_\Phi = 2m+n$. This forces the vertex cover to have exactly one vertex from each variable-edge and exactly two vertices from each clause-triangle.

Proving NP-completeness

For each position in each clause:

add an edge from the clause position's vertex to the vertex representing the corresponding literal.

So, if the literal in the j-th position of the i-th clause is x_k, then add the edge (C_{ij}, x_k).

If the literal is $\neg x_k$, then instead add edge $(C_{ij}, \neg x_k)$.

This completes construction of G_Φ .

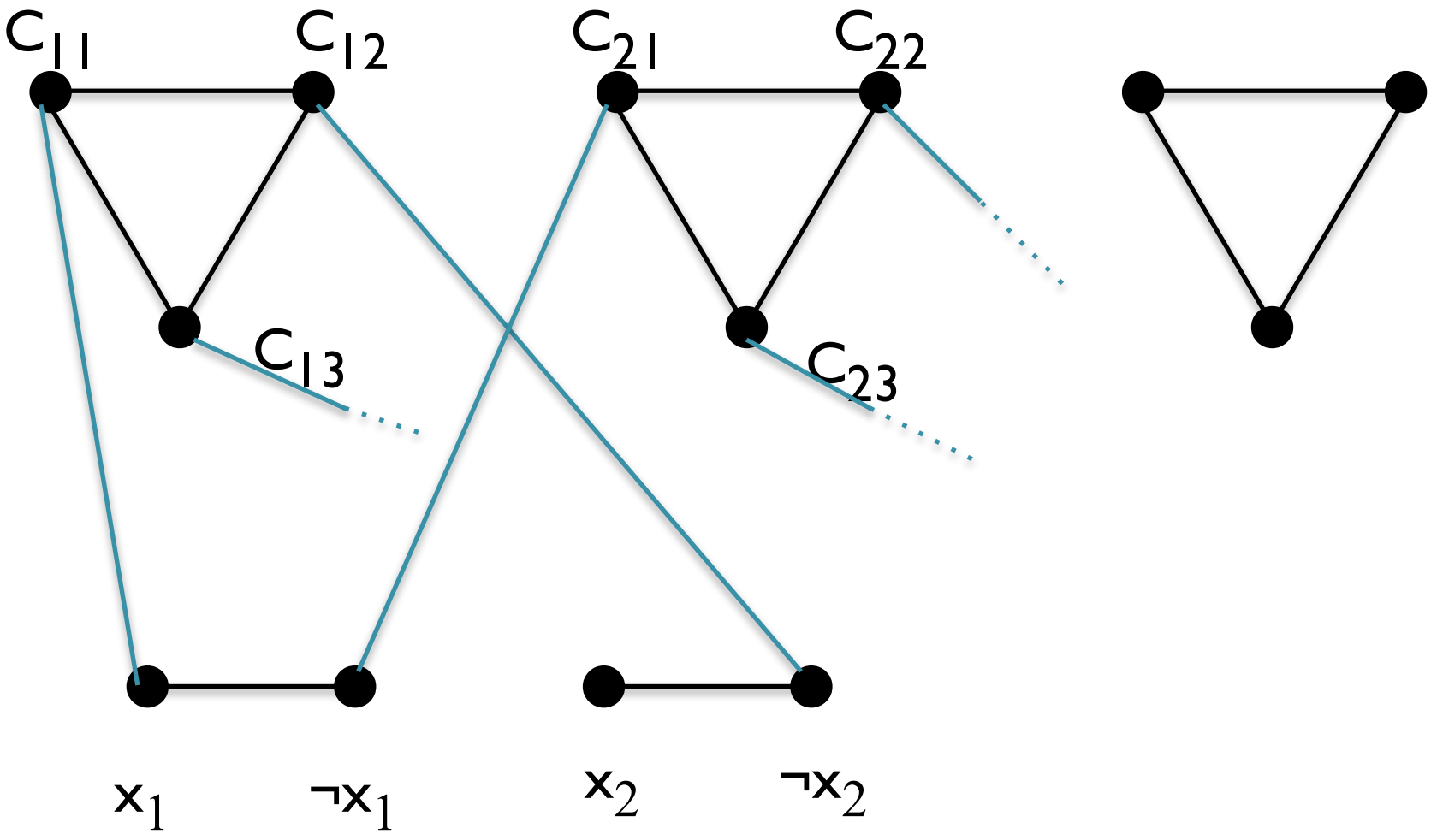
For example: if Φ =

$(x_1 \vee \neg x_2 \vee x_7) \wedge (\neg x_1 \vee x_3 \vee x_4) \wedge \ldots$

then …

Proving NP-completeness

$(x_1 \vee \neg x_2 \vee x_7) \wedge (\neg x_1 \vee x_3 \vee x_4) \wedge \ldots$



Proving NP-completeness

For each variable-edge, the choice of which endpoint is in the vertex cover corresponds to assigning a truth value to that variable. (The *chosen* vertex is the *true* literal.)

The chosen literal covers all edges going from it up to the clauses. So, if literal x_k is True and it appears in position j in clause C_i, then the edge (x_k, C_{ij}) is covered by x_k and does not need to be covered again in the clause-triangle for C_i. So the vertex C_{ij} does not need to be in the vertex cover; the clause-triangle for C_i can be covered by its other two vertices.

Proving NP-completeness

So every clause containing a true literal is easily covered by two vertices.

Conversely, if a clause-triangle only has two vertices from the vertex cover, then the other vertex (not in the VC) gives the position of a literal which *must* be covered.

The current truth assignment is satisfying if and only if every clause has a true literal, which in turn holds if and only if the vertex cover only meets each clause-triangle twice, which in turn holds if and only if the vertex cover has size $\leq k_\Phi$.

Proving NP-completeness

So we have:

Φ is satisfiable if and only if

G_Φ has a vertex cover of size $\leq k_\Phi$.

It remains to show that the reduction is polynomial time. This is fairly straightforward.

Proving NP-completeness

So we now have three NP-complete problems:

SAT, 3-SAT, VERTEX COVER.

We saw in Lecture 22 (polynomial-time reductions) that

VERTEX COVER $\leq_P$ INDEPENDENT SET

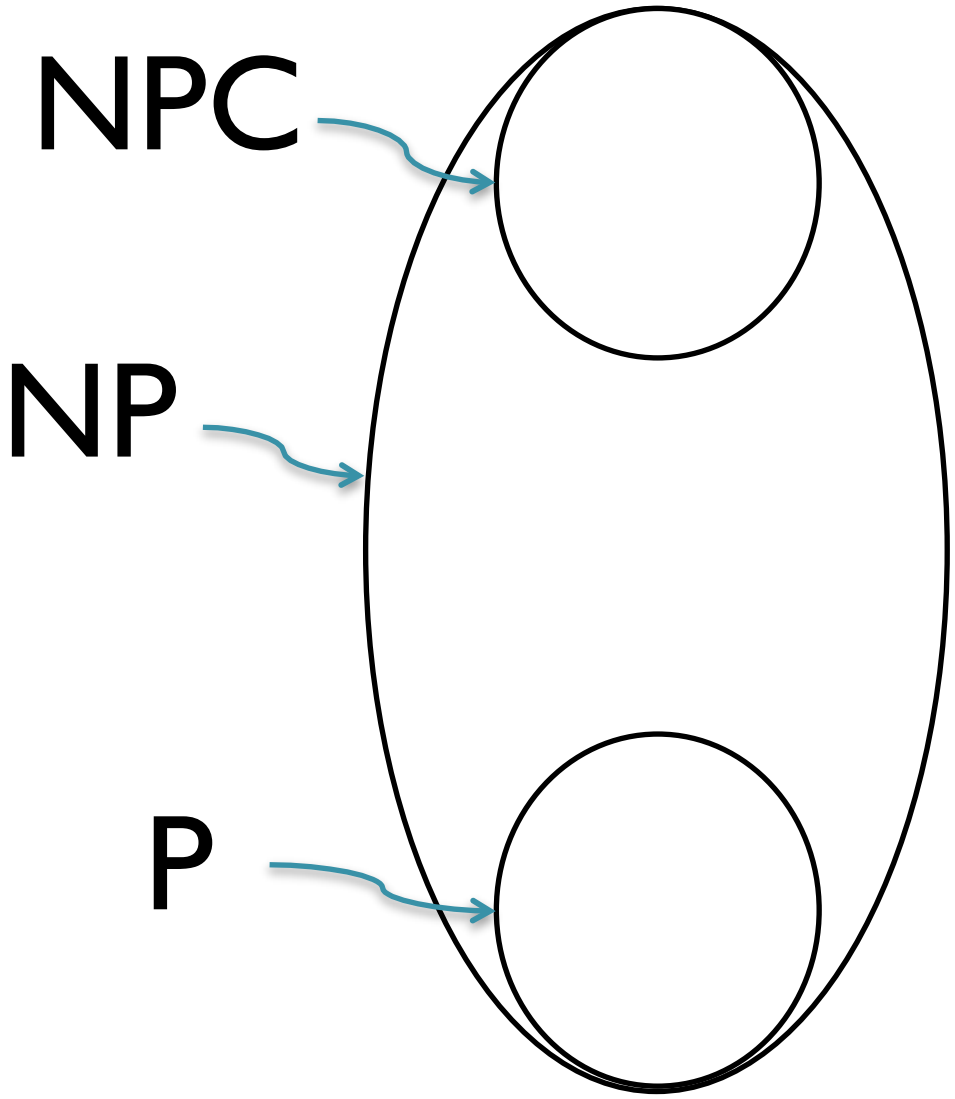
$\leq_P$ CLIQUE

so these two languages are NP-complete too.

Good exercise:

Prove that 3-COLOURABILITY is NP-complete, by reduction from INDEPENDENT SET.

If $P \neq NP$:



In NP, not known to be in P:

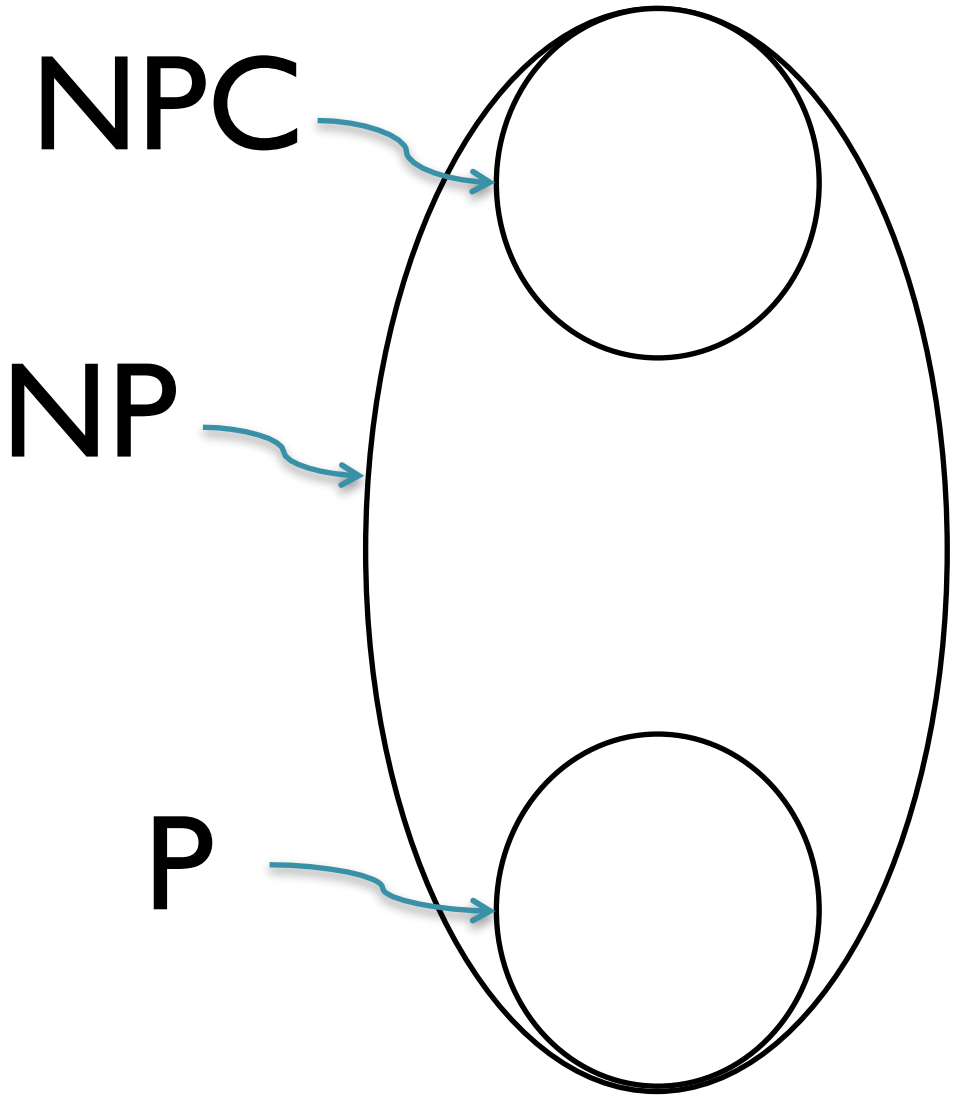
SATISFIABILITY, 3-SAT,
HAMILTONIAN CIRCUIT,
3-COLOURABILITY,
VERTEX COVER,
INDEPENDENT SET, …

GRAPH ISOMORPHISM,
INTEGER FACTORISATION, …

In P:

2-SAT,
EULERIAN CIRCUIT,
2-COLOURABILITY,
PRIMES,
CONNECTED GRAPHS,
SHORTEST PATH,
Invertible matrices,
Context-free languages,
Regular languages, …

If $P \neq NP$:



NP-complete:

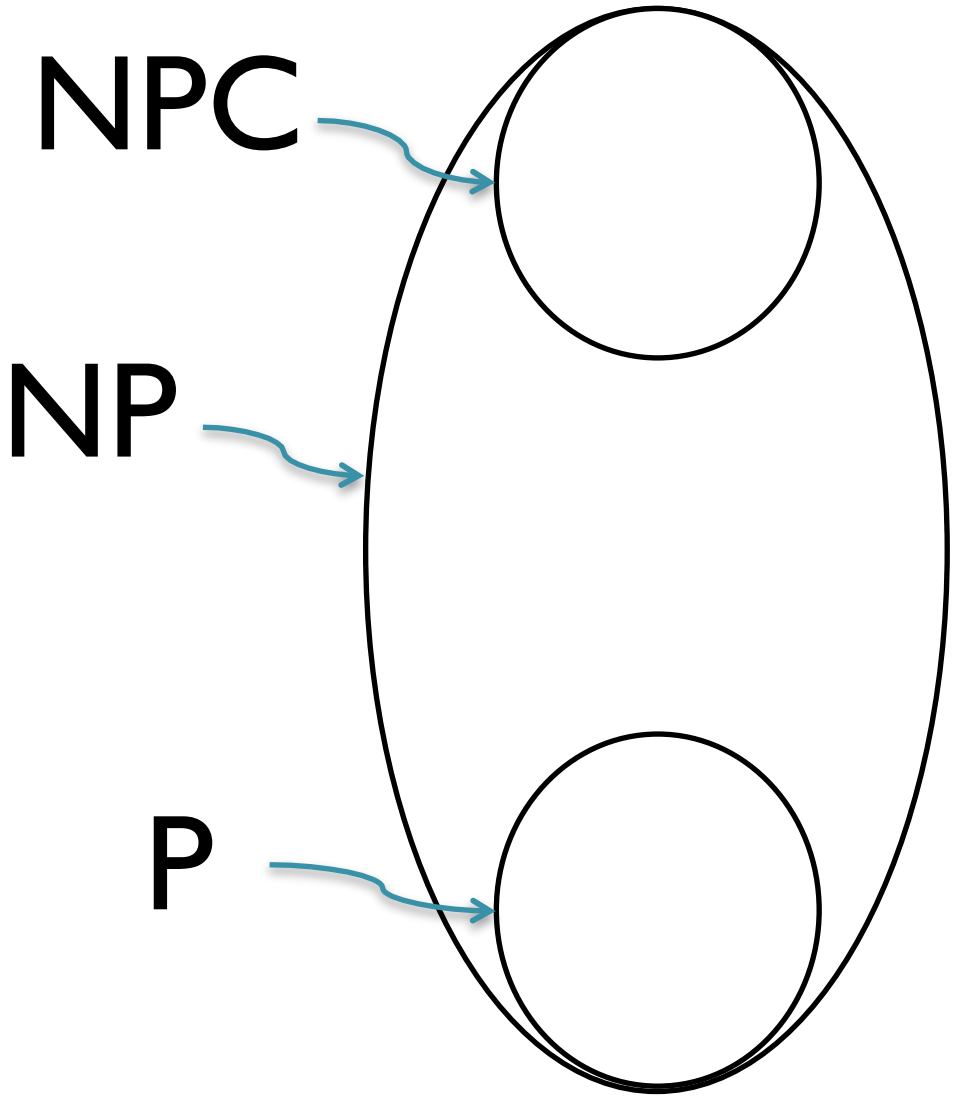
SATISFIABILITY, 3-SAT,
HAMILTONIAN CIRCUIT,
3-COLOURABILITY,
VERTEX COVER,
INDEPENDENT SET, …

GRAPH ISOMORPHISM,
INTEGER FACTORISATION, …

In P:

2-SAT,
EULERIAN CIRCUIT,
2-COLOURABILITY,
PRIMES,
CONNECTED GRAPHS,
SHORTEST PATH,
Invertible matrices,
Context-free languages,
Regular languages, …

If $P \neq NP$:



NP-complete:

SATISFIABILITY, 3-SAT,
HAMILTONIAN CIRCUIT,
3-COLOURABILITY,
VERTEX COVER,
INDEPENDENT SET, …

In NP, not known to be in P **or NPC**:

GRAPH ISOMORPHISM,
INTEGER FACTORISATION, …

In P:

2-SAT,
EULERIAN CIRCUIT,
2-COLOURABILITY,
PRIMES,
CONNECTED GRAPHS,
SHORTEST PATH,
Invertible matrices,
Context-free languages,
Regular languages, …

Implications of NP-completeness

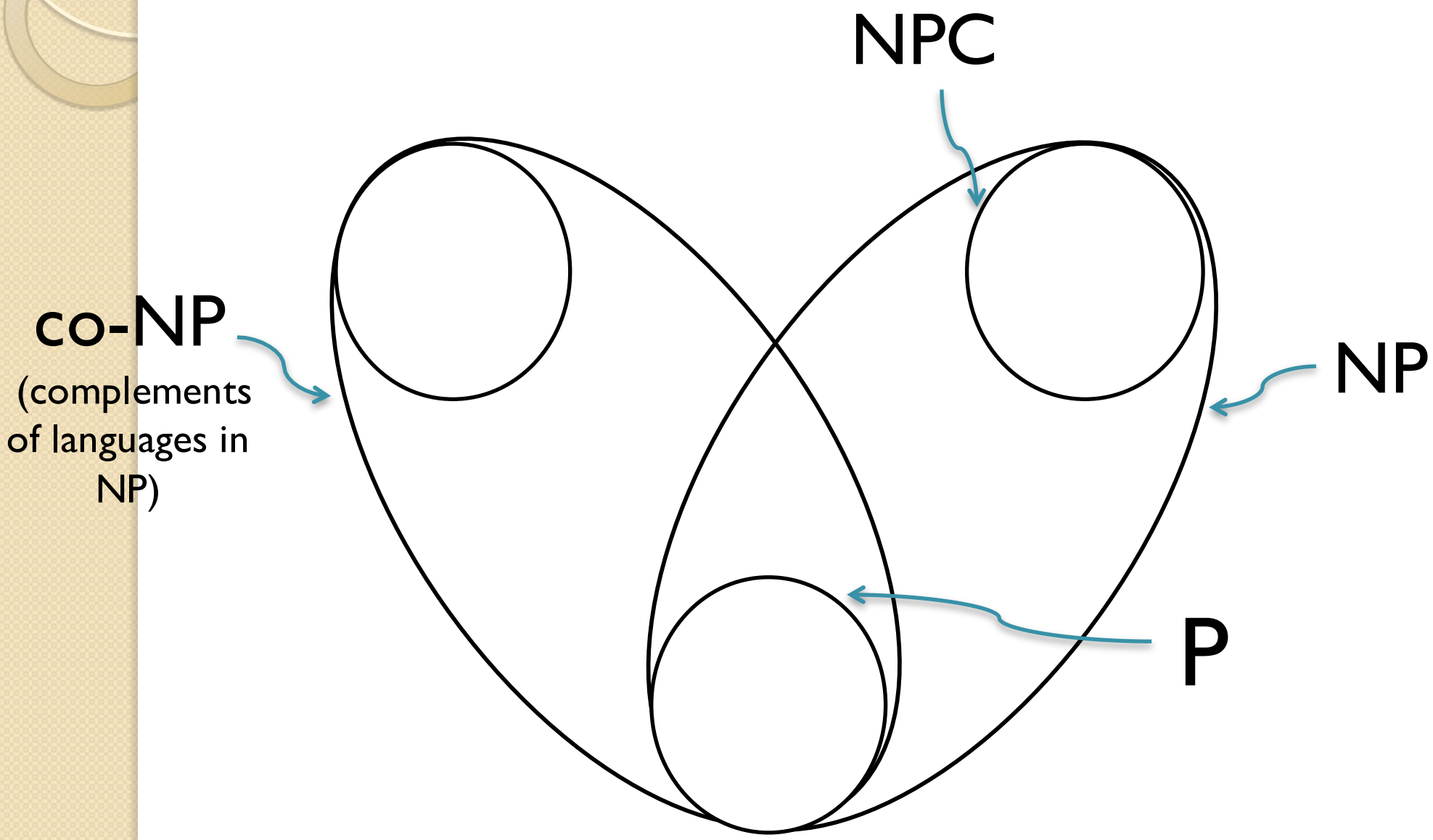
Showing that a language is NP-complete does not make it go away! You may still need to find an algorithm for it.

NP-completeness is evidence that you won't find an *efficient, deterministic* algorithm that works *in all cases* and solves the problem *exactly*.

So you have several options:

- a slow algorithm (exponential time)
- an algorithm for special cases
- approximation algorithm
- randomised algorithm
- maybe in future: quantum computer?

If P ≠ NP and NP ≠ co-NP and …



Revision

- Sipser, section 7.4, 7.5.