Introduction to Intractability

P, NP, NP-completeness

Tue, March 26th

Outline For Today

1. Intractability: P, NP, NP-completeness
2. How to argue a problem is NP-complete

CS 341 Diagram

Fundamental *(& Fast)* Algorithms to Tractable Problems

- MergeSort
- Strassen's MM
- BFS/DFS
- Dijkstra's SSSP
- Kosaraju's SCC
- Kruskal's MST
- Floyd Warshall APSP
- Topological Sort
- …

Common Algorithm Design Paradigms

- Divide-and-Conquer
- Greedy
- Dynamic Programming

Mathematical Tools to Analyze Algorithms

- Big-oh notation
- Recursion Tree
- Master method
- Substitution method
- Exchange Arguments
- Greedy-stays-ahead Arguments

Intractable Problems

- P vs NP
- Poly-time Reductions
- Undecidability

Other (Last Lecture)

- Randomized/Online/Parallel Algorithms

Focus of CS161

- Practical Algorithms to Fundamental Problems
- Almost-linear time super-fast algorithms
  - Sorting: O(nlogn)
  - Dijkstra: O(mlog(n))
  - MST: O(mlog(n))
- Superlinear but still practical
  - Strassen's Algorithm: $O(n^{2.83})$
  - Karatsuba Integer Multiplication: $O(n^{1.58})$
  - Floyd-Warshall: $O(n^3)$

Interesting/Sad/Exciting Fact

Many other important problems seem impossible to solve very efficiently.

Ex: TSP, Knapsack, Independent Set

A critical skill that we all have to acquire is to recognize such problems.

One Technicality: Decision vs Optimization Problems

- From now on we will look at “Decision Problems”
- A decision problem has YES/NO answer.
- An optimization problem maximizes/minimizes a fnc.
- Ex:
  - Knapsack-Optimization: What’s the max value we can put into the knapsack?
  - Knapsack-Decision: Can we put value $\geq k$ into the knapsack?

Optimization and Decision Versions of Problems are Essentially Equivalent (1)

If you can solve the optimization problem (say in time T)

then you can solve the decision version in time T.

E.g.: Knapsack-Decision: Can we put value ≥ k into the knapsack?

```
Alg Knapsack-Decision(values, weights, W, k):
    Let k* = Knapsack-OPT(values, weights, W).
    if k < k* return YES
    else return NO
```

Optimization and Decision Versions of Problems are Essentially Equivalent (1)

If you can solve the decision problem (say in time T) then you can solve the optimization version in time T*log(b), where b is an appropriate bound on the value that the function we are optimizing can take.

```
Alg Knapsack-OPT(values, weights, W):
    let b = sum of the values
    do binary search (i.e., k = b, b/2, b/4, …, 1)
       Knapsack-Decision(values, weights, W, k).
    return maximum k that returns YES
```

- We need decision problems for the formalism of P and NP.
- But the optimization versions of the problems are as hard (or as easy) as the decision versions.

Formalizing Tractability: Class P

- Given a computational (decision) problem C
- **P**: C is $\in$ **P** (polynomial-time solvable) if $\exists$ an algorithm solving C with $O(n^k)$ run-time, for some constant k.
  - where n is the input length in bits
  - k is some constant, say, 1, 2, 5, 1M, etc.
  - Ex poly-times: $O(n)$, $O(n\log(n))$, $O(n^3)$, $O(n^{100000})$
- Ex: Every problem we saw so far (except 0-1 knapsack)

Interpretation of P and Its Caveats

*A rough test for **tractability**.*

Caveat 1: n^{1000} is not tractable in practice

Caveat 2: Some intractable problems can actually be solved in some restricted form efficiently

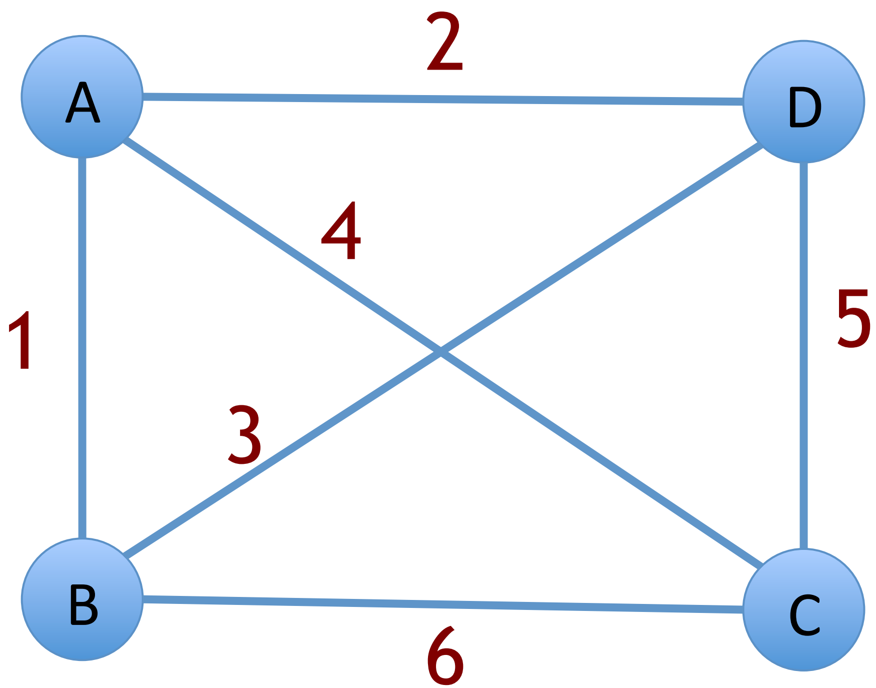
*But generally speaking, has been **extremely successful** in classifying tractable problems.*

Example of Intractable Problem: TSP

TSP: Traveling Salesman Problem

Input: Complete graph G(V, E) with non-negative edge costs

Output: ∃ a tour with cost ≤ 12 [i.e., cycle visiting each v once]?

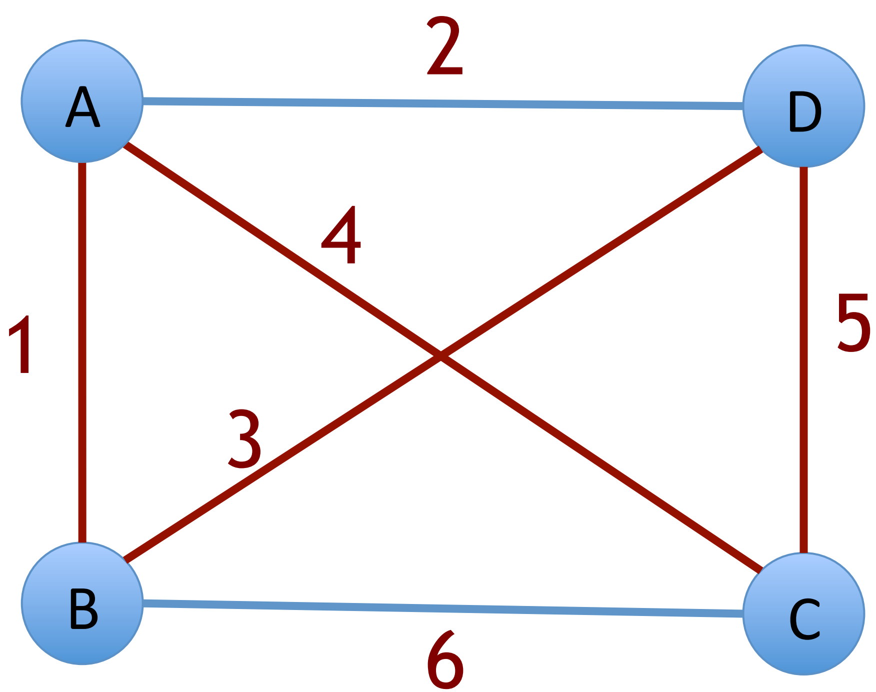


Example of Intractable Problem: TSP

TSP: Traveling Salesman Problem

Input: Complete graph G(V, E) with non-negative edge costs

Output: ∃ a tour with cost ≤ 12 [i.e., cycle visiting each v once]?



Min Tour: A->B->D->C->A
= 13
So the answer is NO.

Conjecture from 1965 (Jack Edmonds)

- We have been looking for a fast algorithm for TSP for > 80 years, and no one has succeeded.
- After 30 years or so in 1965, Jack Edmonds made the following conjecture:

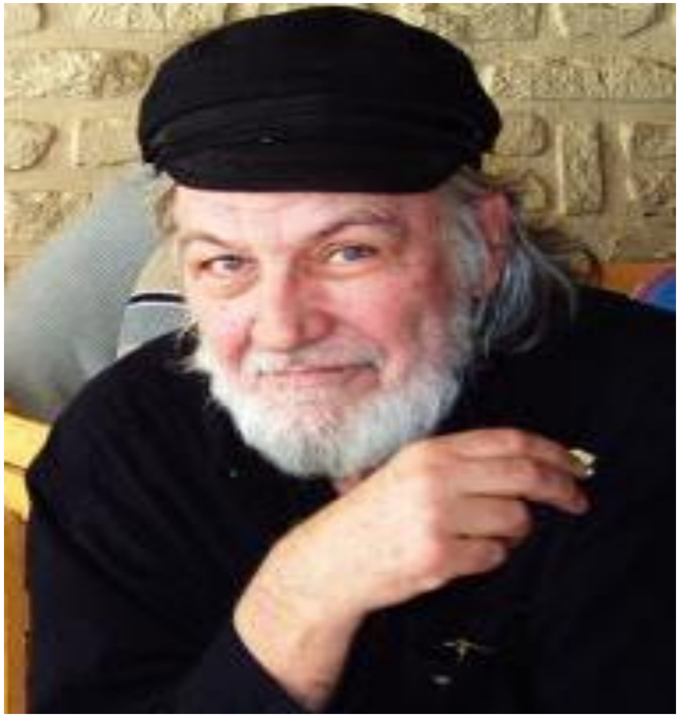
There is no poly-time algorithm for TSP.

(equivalent to conjecture: P != NP)

To this day, no one has been able to prove/ disprove this conjecture.

Jack Edmonds

- UWaterloo Professor from 1969-1999



"The classes of problems which are respectively known and not known to have good algorithms are of great theoretical interest. [...] I conjecture that there is no good algorithm for the TSP. *My reasons are the same as for any mathematical conjecture: (1) It is a legitimate mathematical possibility, and (2) I do not know." (1965)*

Making a Case For TSP's Intractability?

- (So far) We have not been able to argue for TSP's intractability in an absolute sense.
- Instead accumulate evidence of intractability

Idea From Early 1970s:

Instead show "relative" intractability

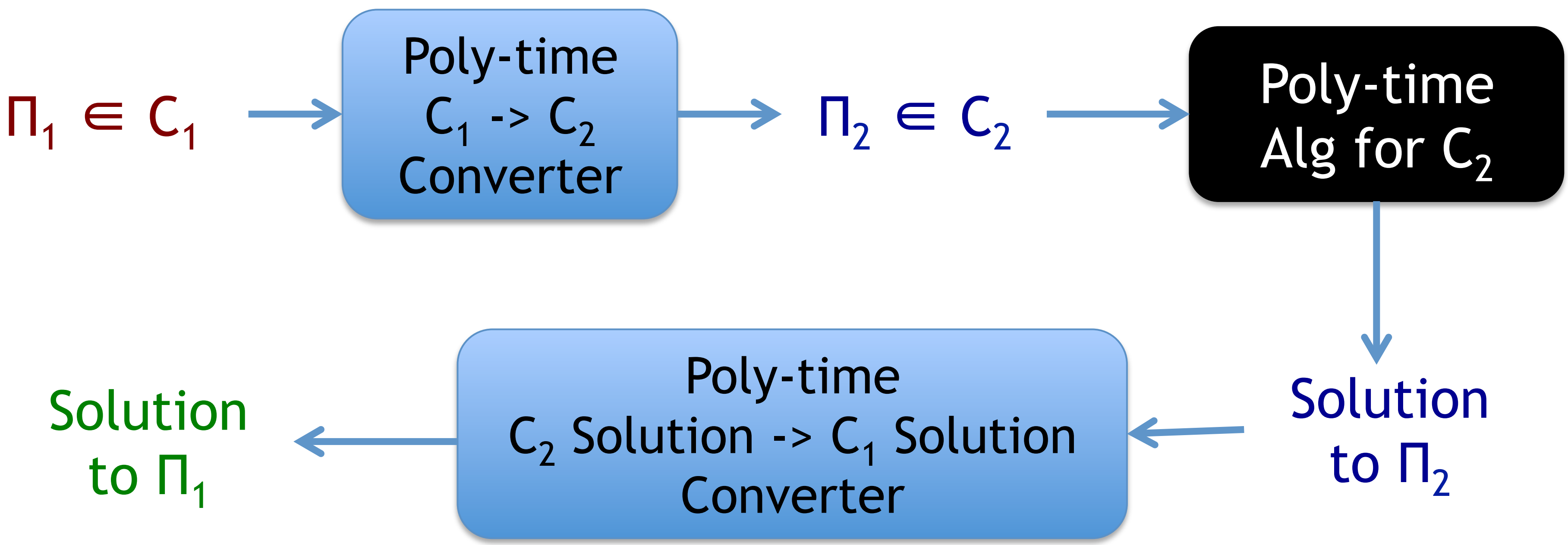
(i.e. show TSP is as hard as bunch of other problems.)

Reductions

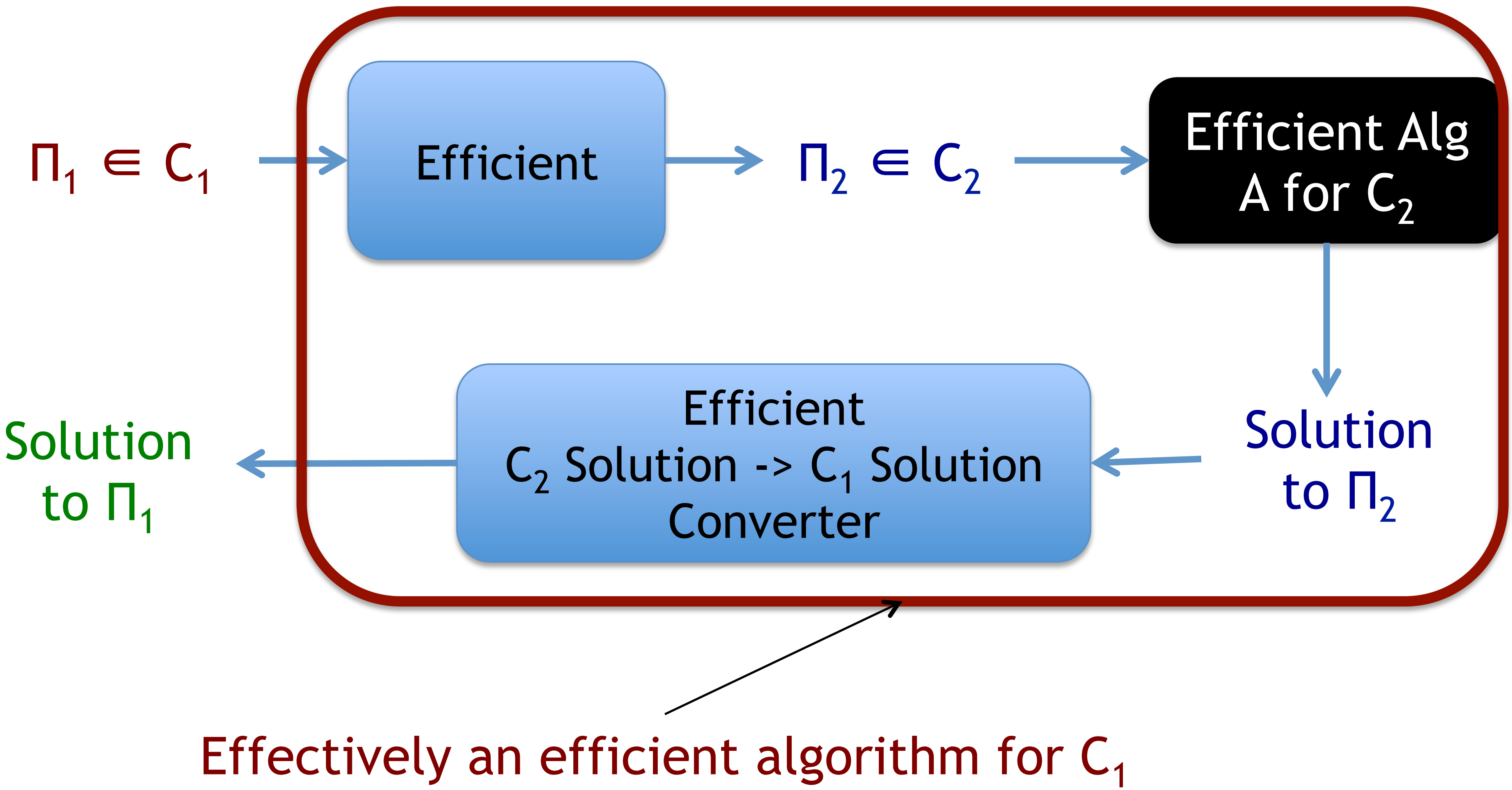
What does it mean for problem C_2 to be as hard as C_1?

Definition: C_1 *reduces* to C_2, ($C_1 \leq_p C_2$), if given a poly-time algorithm for C_2, we can solve C_1 in poly-time.

If C_1 reduces to C_2 => C_2 is "as hard as" C_1



High-level Idea of Reduction



If we can solve C_2 efficiently, we can also solve C_1 efficiently

$\Rightarrow$ *C_2 as hard as C_1*

Reductions are Transitive

Claim: If $C_1 \leq_p C_{2,}$ and $C_2 \leq_p C_3$, then $C_1 \leq_p C_3$

Proof: We have to argue if we have a poly-time algorithm for C_3, we can solve C_1 efficiently. Or C_3 is as hard as C_1.

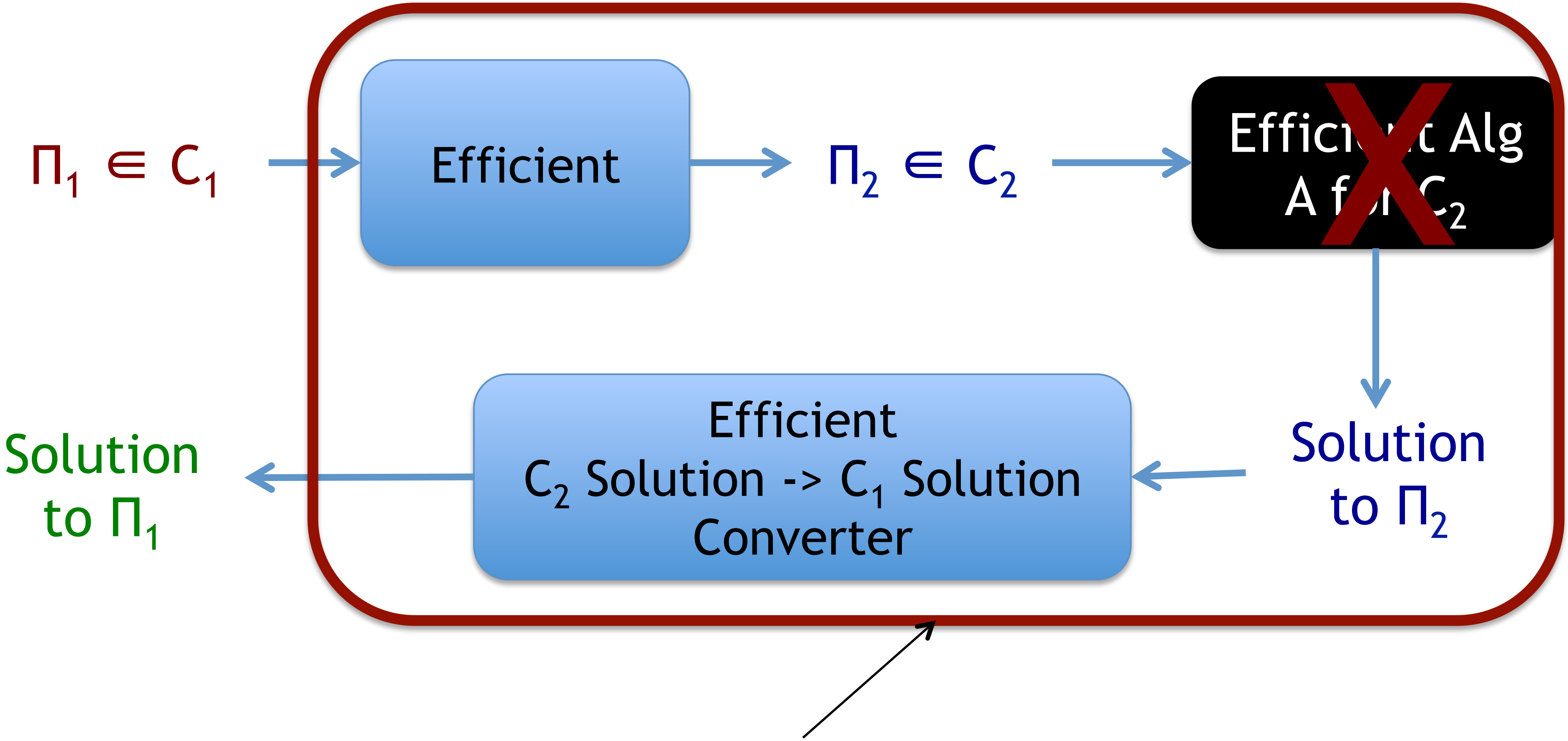
If we had a poly-time algorithm for C_3

$\Rightarrow$ we could solve C_2 in poly-time (since $C_2 \leq C_3$)

$\Rightarrow$ we could solve C_1 in poly-time (since $C_1 \leq C_2$)

Q.E.D.

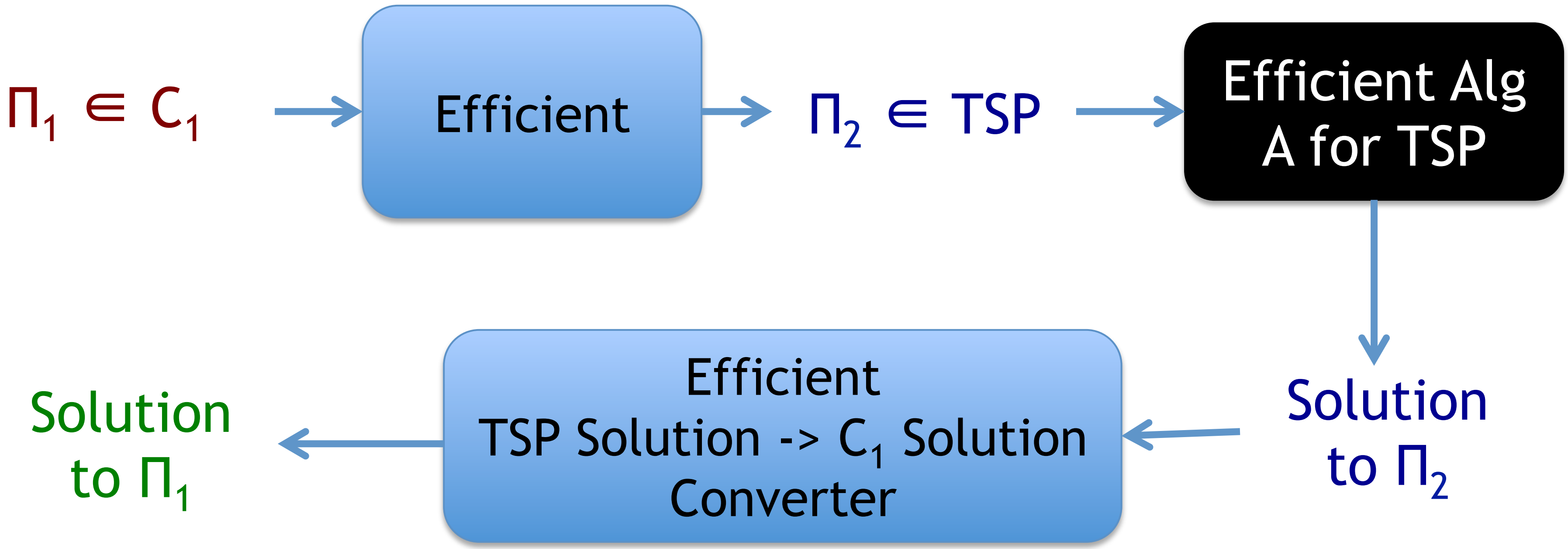
Contrapositive



Assume we knew NO efficient algorithm for C_1 exists

Then if $C_1 \leq_p C_2$, *there cannot be an efficient algorithm A for* C_2

Making a Case for TSP's Intractability



If we can solve TSP efficiently, we can also solve C_1 efficiently

$\Rightarrow$ *TSP is as hard as C_1*

Goal: Argue TSP is as hard as a large set $\mathcal{C}$ of problems

Definition: Completeness

Let $\mathcal{C}$ be a set of problems.

$C_i \in \mathcal{C}$

If $\forall\ C_k \in \mathcal{C}$, $C_k \leq_p C_i$, then C_i is $\mathcal{C}$-complete.

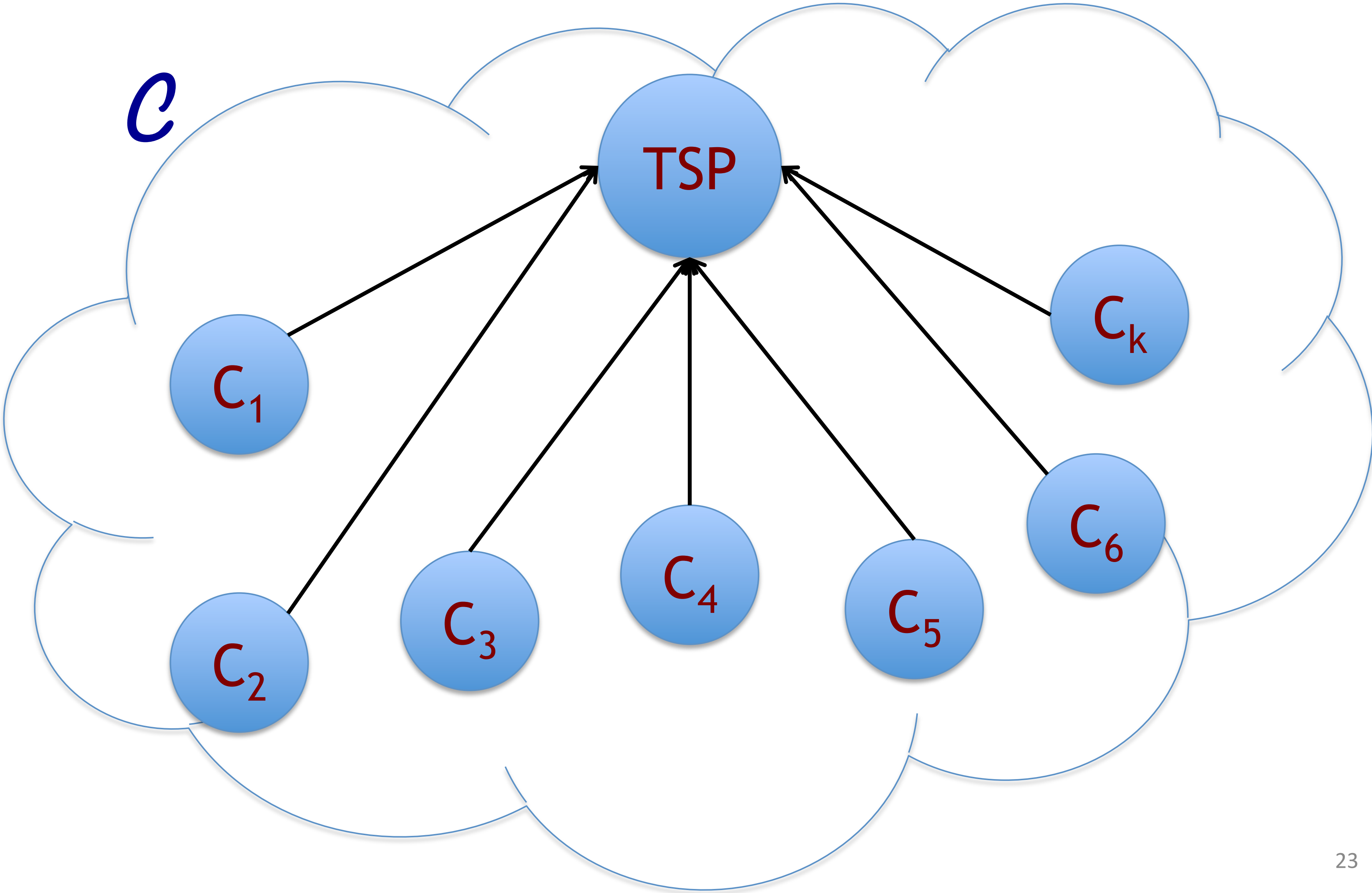
(i.e. C_i is as hard as every other problem in $\mathcal{C}$)

(i.e. C_i is the hardest problem in $\mathcal{C}$)

Goal: Argue TSP is C-complete for as large a set $\mathcal{C}$ as possible.

The larger $\mathcal{C}$, the more evidence we accumulate for TSP's intractability.

Arguing TSP is $\mathcal{C}$-complete for a large $\mathcal{C}$



Picking an Appropriate $\mathcal{C}$

Option 1: Let $\mathcal{C}$ be any problem

Q: Could we hope to prove that TSP is as hard as any other computational problem?

A: No, there are problems, such as the Halting Problem, which we know are simply not solvable! (i. e, do not have any algorithms solving them, let alone an efficient one).

But TSP is solvable: if nothing, by brute-force search. Exponential-time but solvable.

Option 2: $\mathcal{C}$ = NP: Brute-force Solvable Problems

Option 2: Let $\mathcal{C}$ be all brute-force solvable problems

Definition (NP or brute-force solvable problems):

A problem C $\in$ NP if:

1. Correct solutions have polynomial length.
2. Problem instances that have YES answer are verifiable given a claimed solution in poly-time.

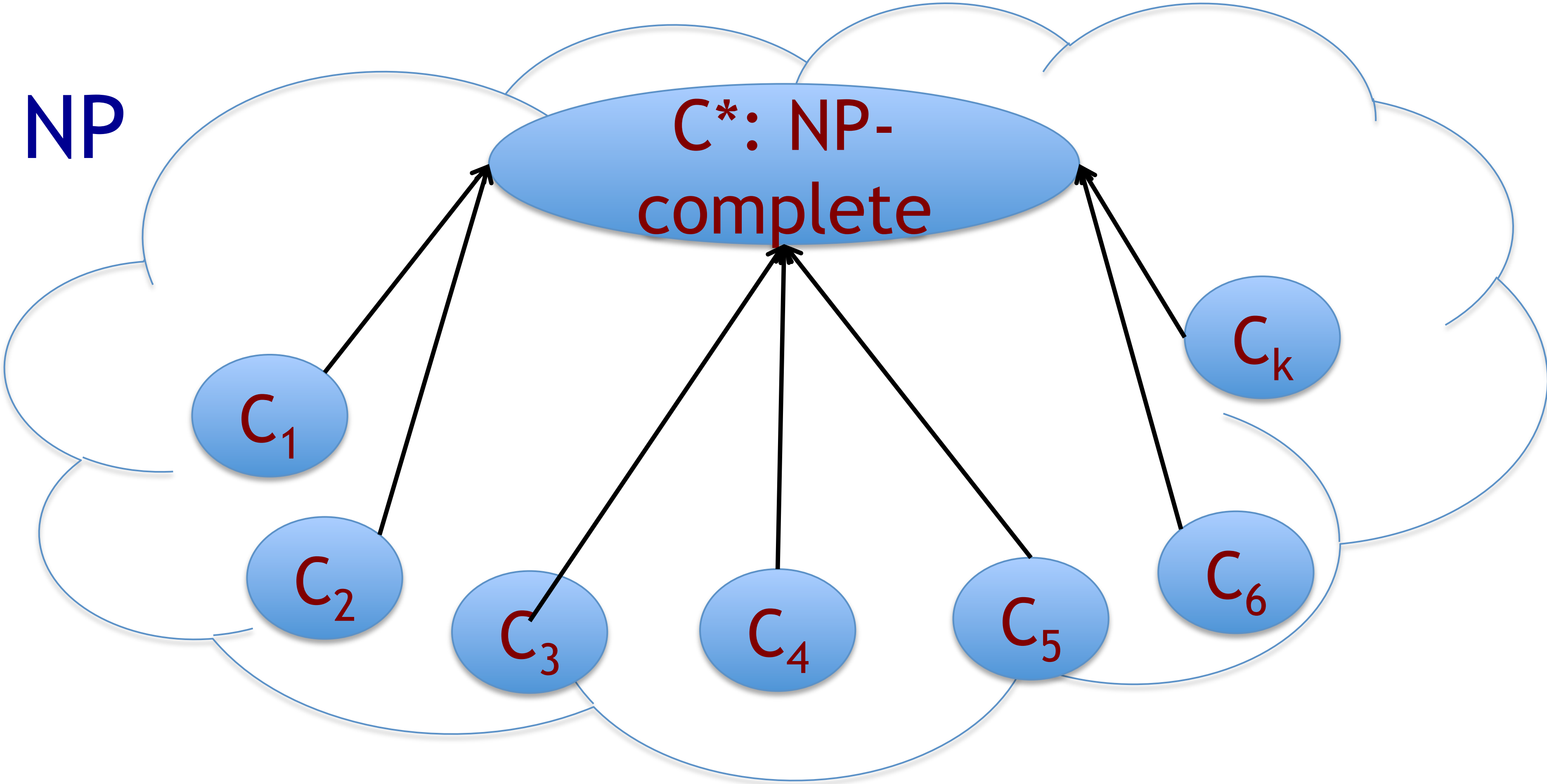
****All that is required to be in NP is that we can verify the solvable problems efficiently given a claimed claimed solution.****

Example NP problems

- Every problem we have seen in CS 341.
- Every problem in P.
- (Most-likely) Every problem you will ever see in practice.
- Ex: Decision version of TSP: $\exists$ T of size $\leq$ k?

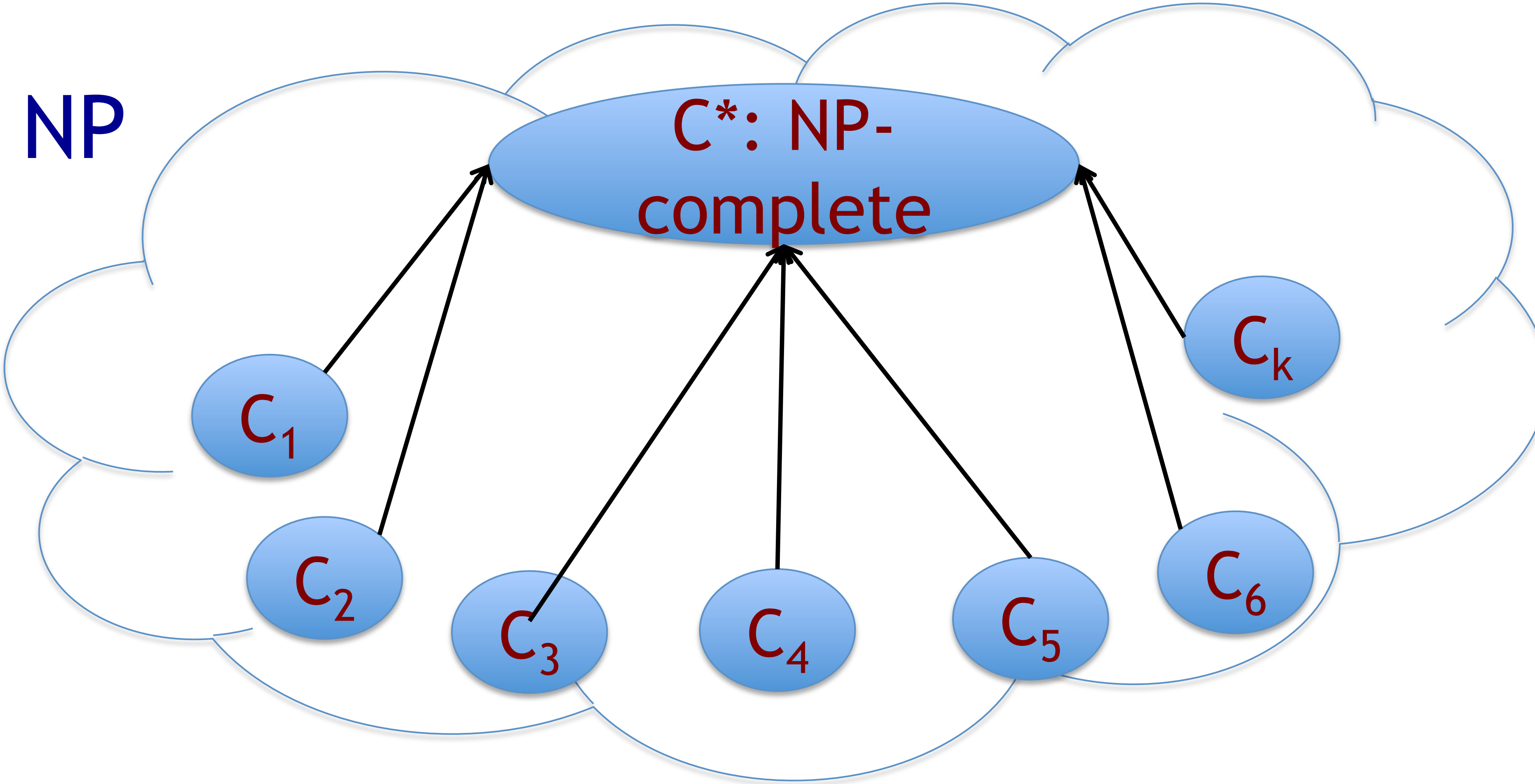
1. Each tour T is a cycle of size n, so poly-length.
2. We can verify length of T in poly-time: just sum each edge in T.

What is an NP-complete Problem?



C is as hard as any NP problem!*

Solving an NP-complete Problem



If we can solve C efficiently, we can solve every single NP problem (basically all problems in practice)!!!*

Do NP-complete Problems Exist?

YES! They're Everywhere!

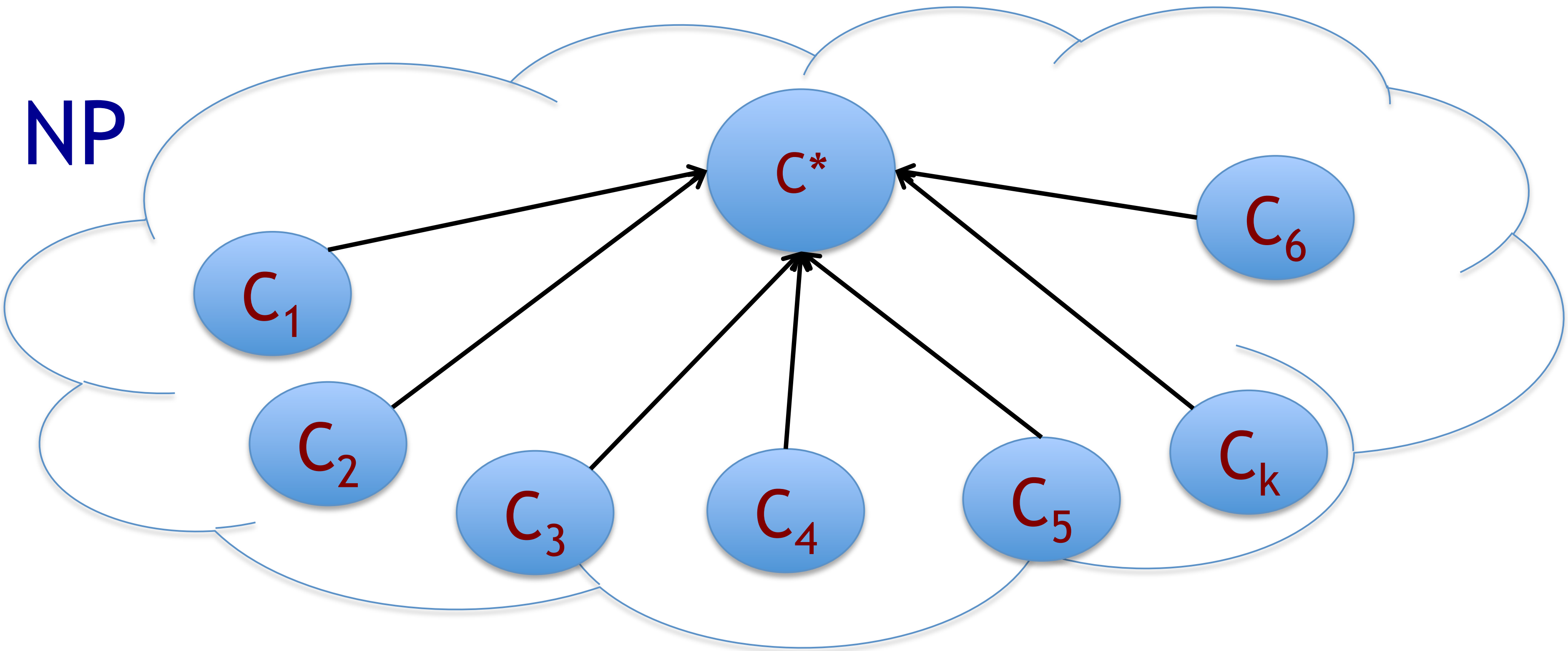
(There are thousands of them)

Outline For Today

1. Intractability: P, NP, NP-completeness
2. How to argue a problem is NP-complete

Two Ways to Argue C* is NP-Complete (1)

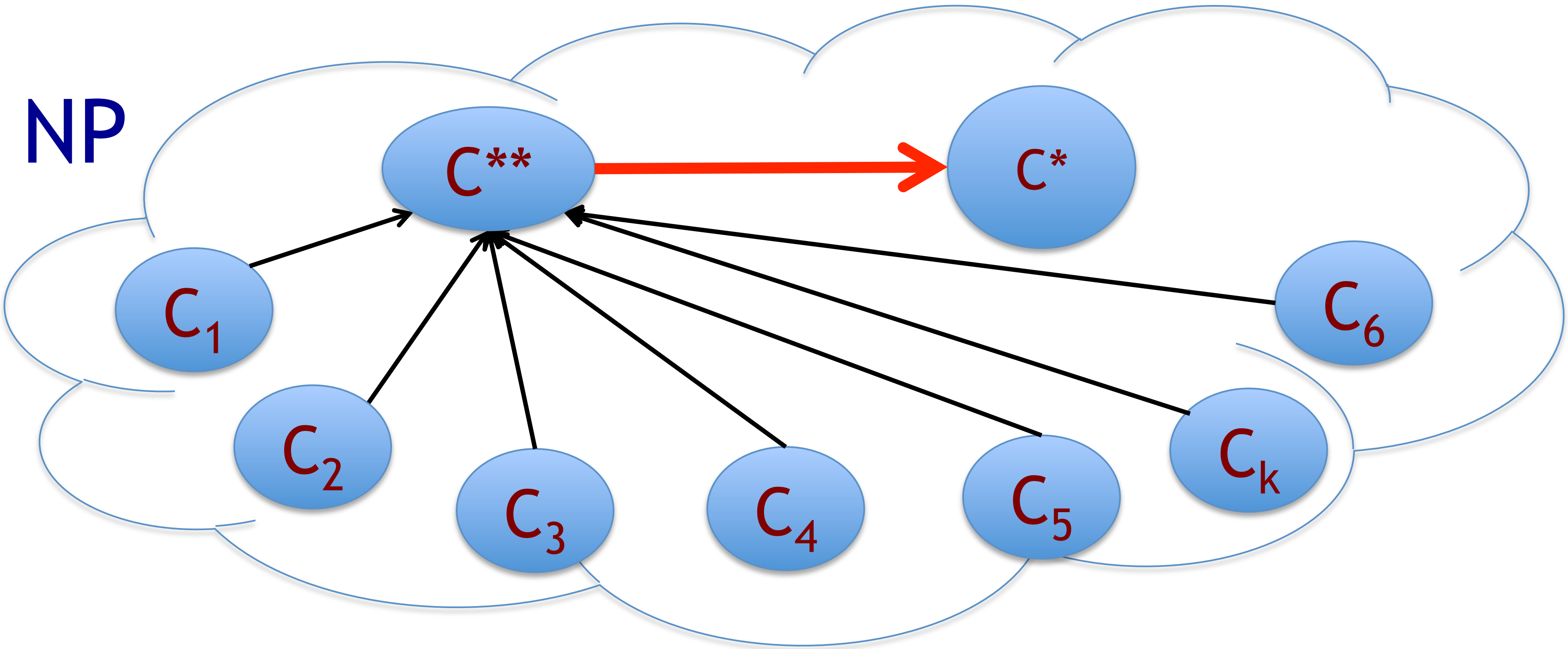
1. Directly argue every NP problem reduces to C*



Done for SAT (& CIRCUIT-SAT) in 1970 by Cook & Levin

Two Ways to Argue C* is NP-Complete (2)

2. Argue an NP-complete C** reduces to C*



B/c reductions are transitive => All NP problems reduce to C*

Done since 1971 for 1000s of problems

History of NP-completeness

- *< 1970: lots of unsolved problems*
- *1971: Cook-Levin Thm: SAT is NP-complete (just from the definition of NP)*
- *1972: Karp: By showing SAT reduces to 21 other problems, showed the existence of 21 other NP-complete problems. (why?)*
- *Since 1972: 1000s of problems are NP-complete.*

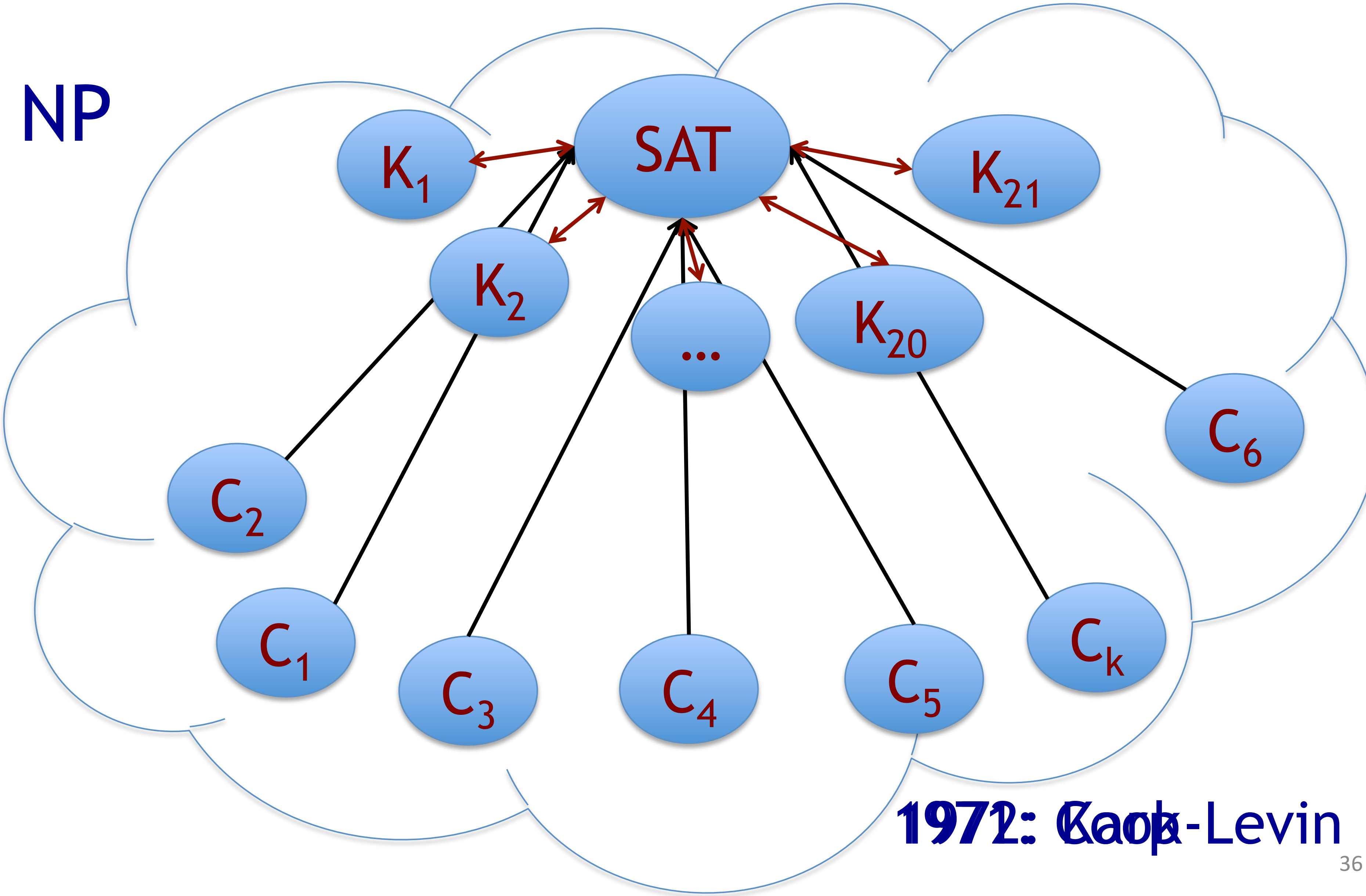
Stephen Arthur Cook

- Currently University of Toronto Professor

Leonid Levin

- Soviet-American computer scientist
- Currently at Boston University

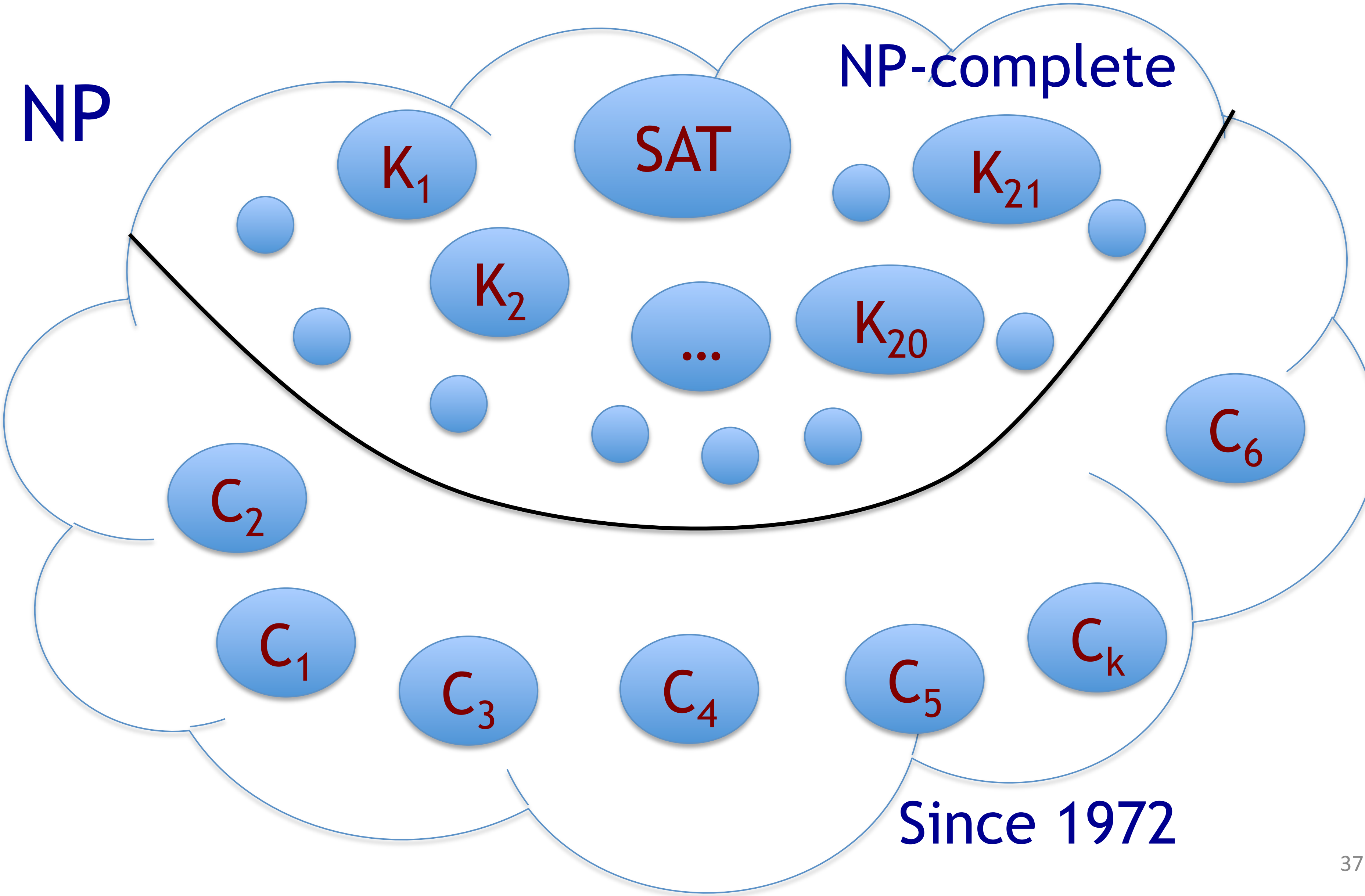
History of NP-completeness



1971: Cook-Levin

1972: Karp

History of NP-completeness

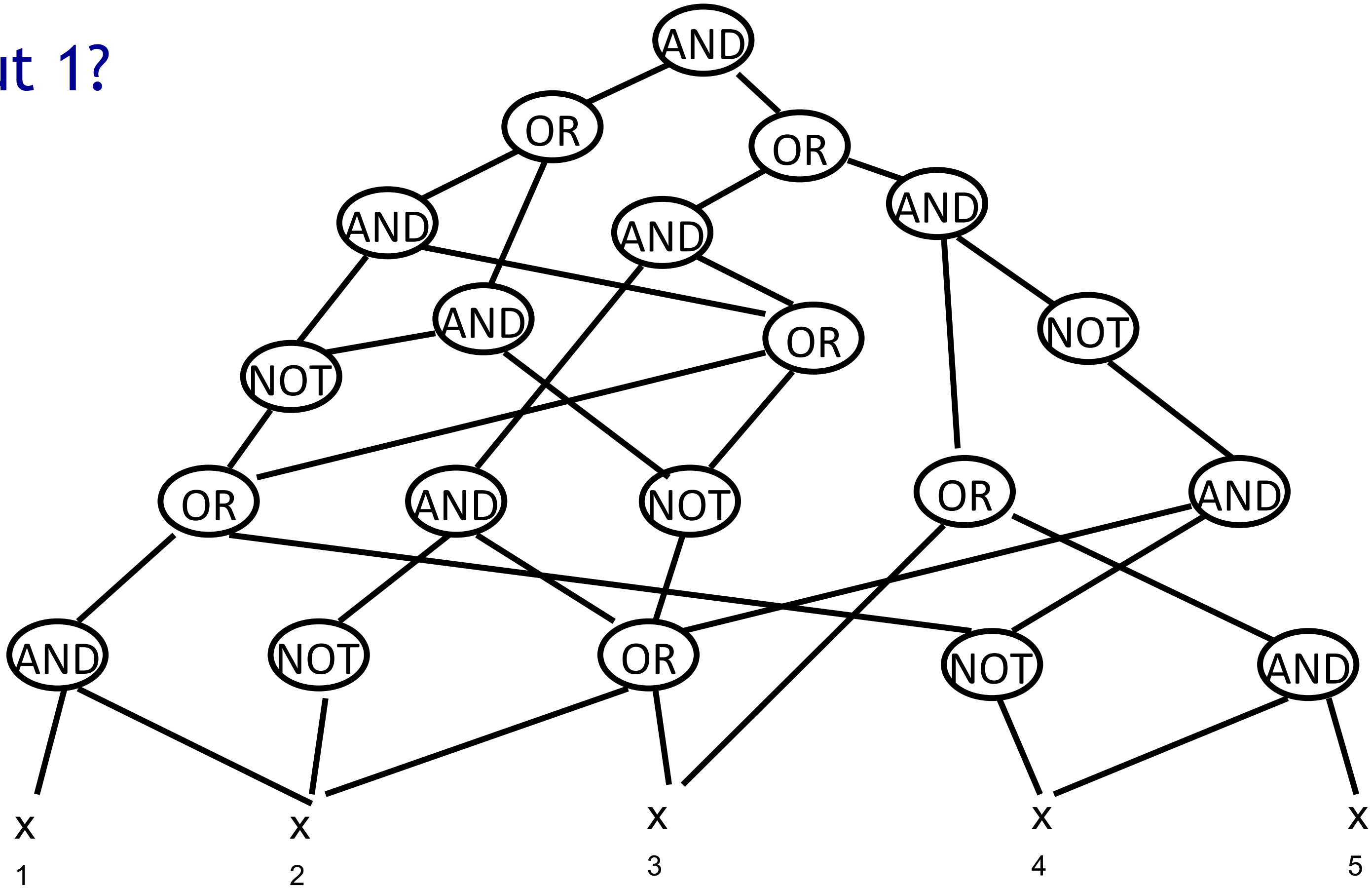


Method 1: Direct Argument: Ex: CIRCUIT-SAT

Input: A circuit C of AND, OR, and NOT gates

n inputs $x_1, x_2, \ldots, x_n$

Output: Is C satisfiable, are there 0/1 values to x_i that make C output 1?



Method 1: Direct Argument: Ex: CIRCUIT-SAT

Idea for proving every NP problem C* reduces to CIRCUIT-SAT:

Every NP problem by dfn has a poly-time verifier algorithm A.

A takes poly-size inputs and runs poly-time.

Represent the state of the machine at each time-step of A by a sub-circuit.

Merge the sub-circuits into a circuit Z. (All poly-time operations)

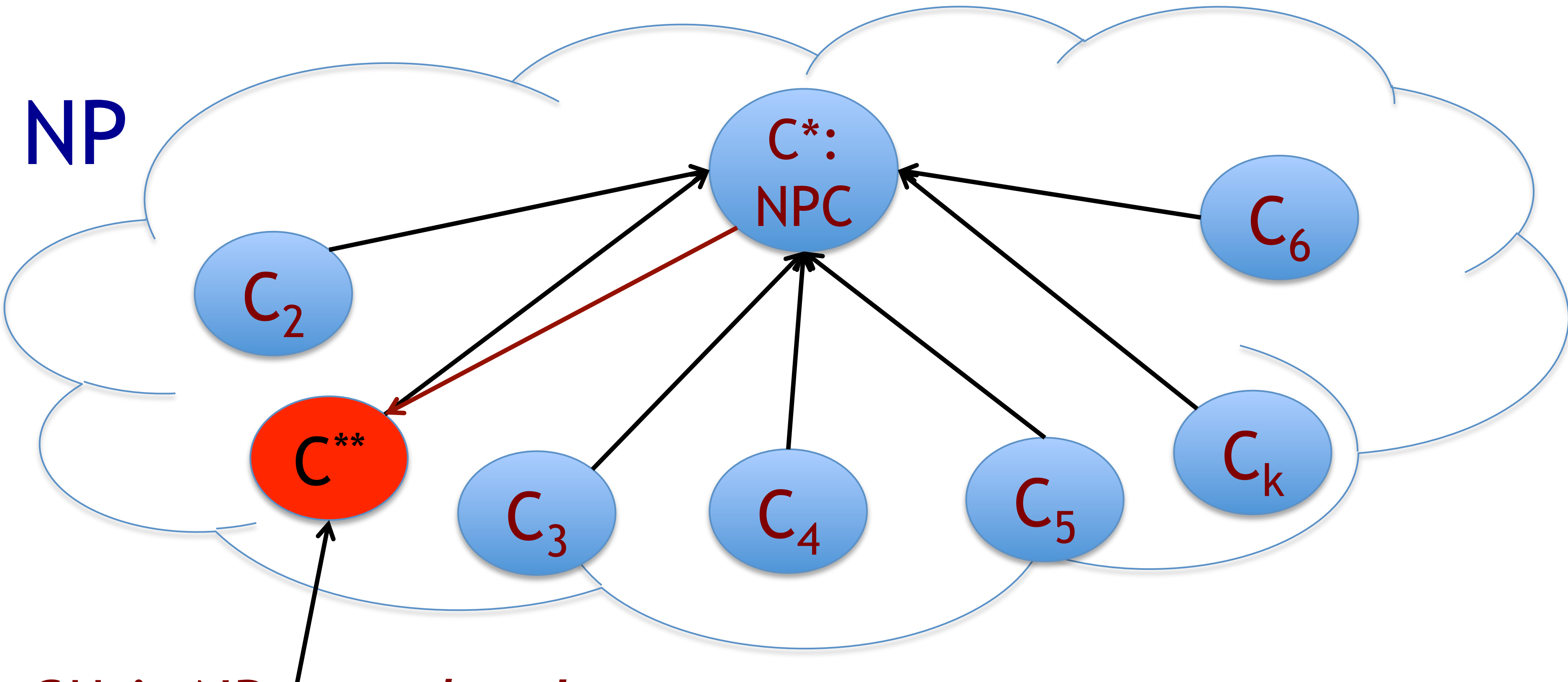
And argue that C* returns YES iff circuit Z has a satisfiable input.

****Method 2: Reductions***
Ex: TSP

Show that TSP is NP-complete => i.e. TSP is as hard as any other NP problem

By showing that another known NP-complete problem reduces to it.

Reducing C* to C** => C** is NP-complete



*C** is NP-complete!*

*If we can solve C** efficiently => we solve C* efficiently*

=> therefore we solve all NP problems efficiently

Why is TSP NP-complete?

HAM-CYCLE Problem:

Input: Undirected Graph G(V, E)

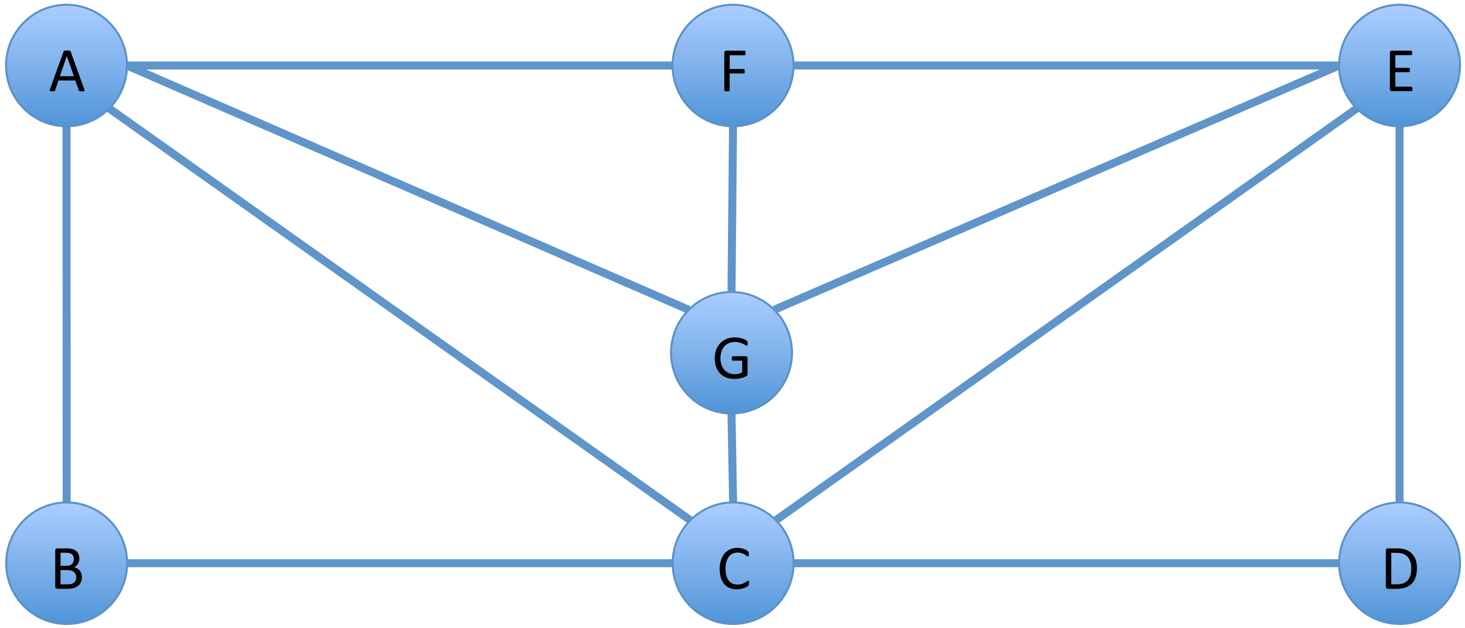
Output: YES if G contains a Hamiltonian cycle/NO o.w

Dfn: A Ham. Cycle is a simple cycle that contain each vertex of the graph once.

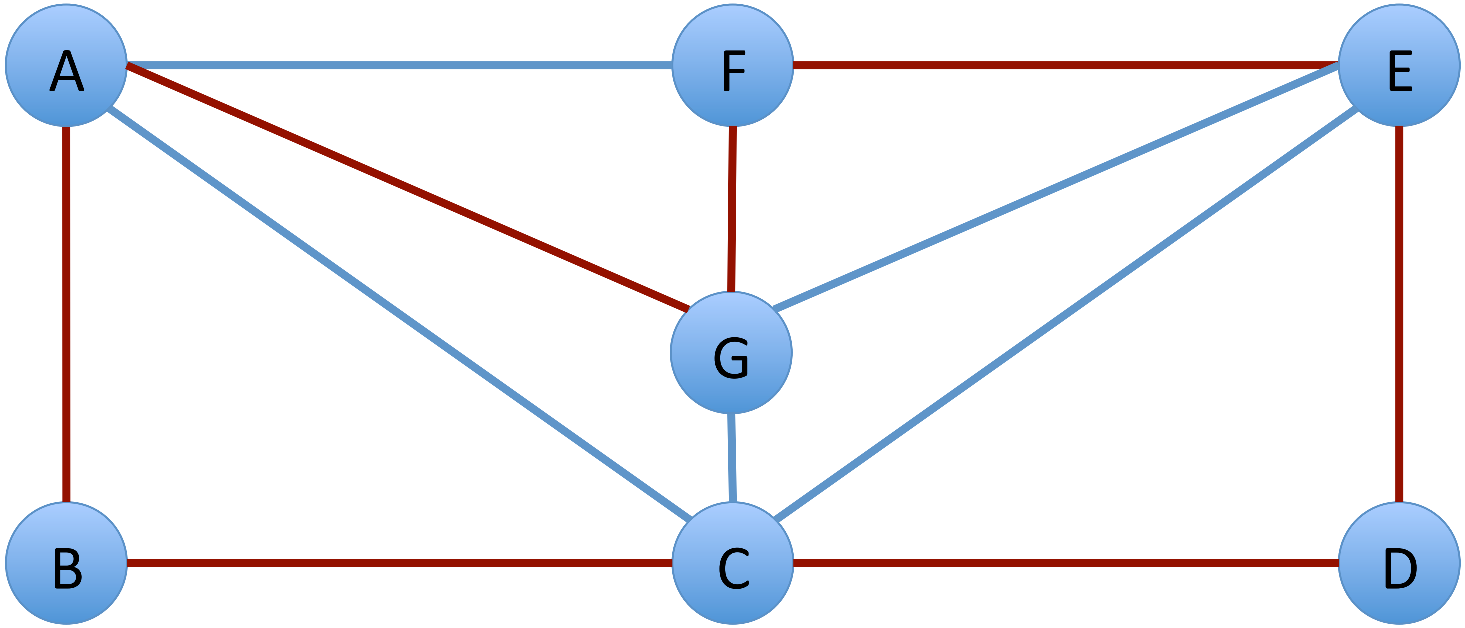
Fact: Hamiltonian Cycle is NP-complete.

$\text{SAT} \le_p \text{3-SAT} \le_p \text{CLIQUE} \le_p \text{VERTEX-COVER} \le_p \text{HAM-CYCLE}$

Hamiltonian Cycle Example (1)

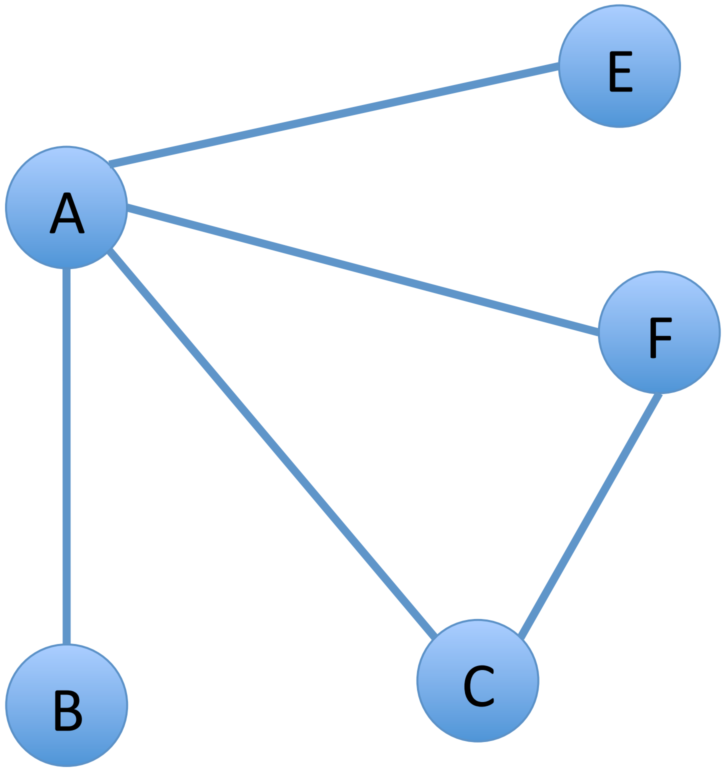


Hamiltonian Cycle Example

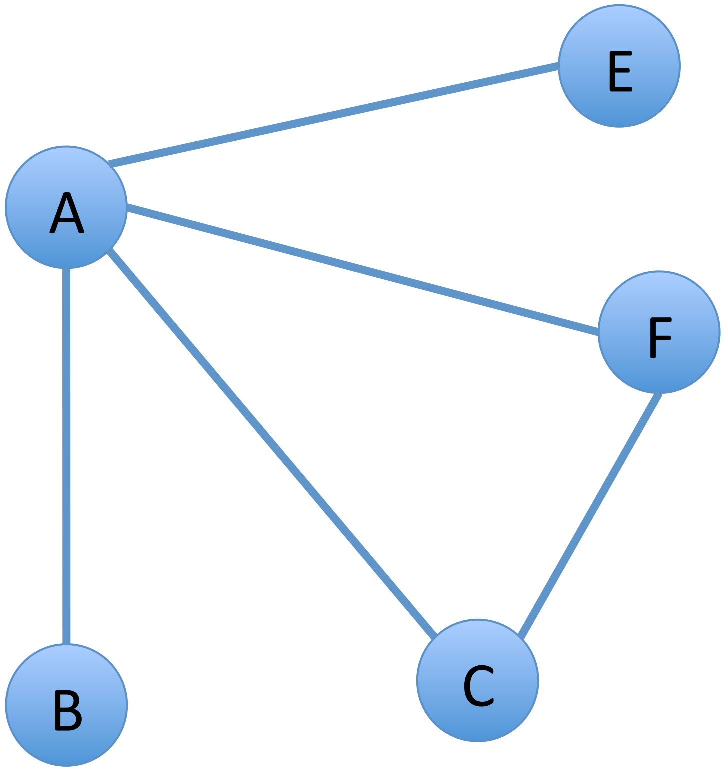


Answer: YES.

Hamiltonian Cycle Example



Hamiltonian Cycle Example

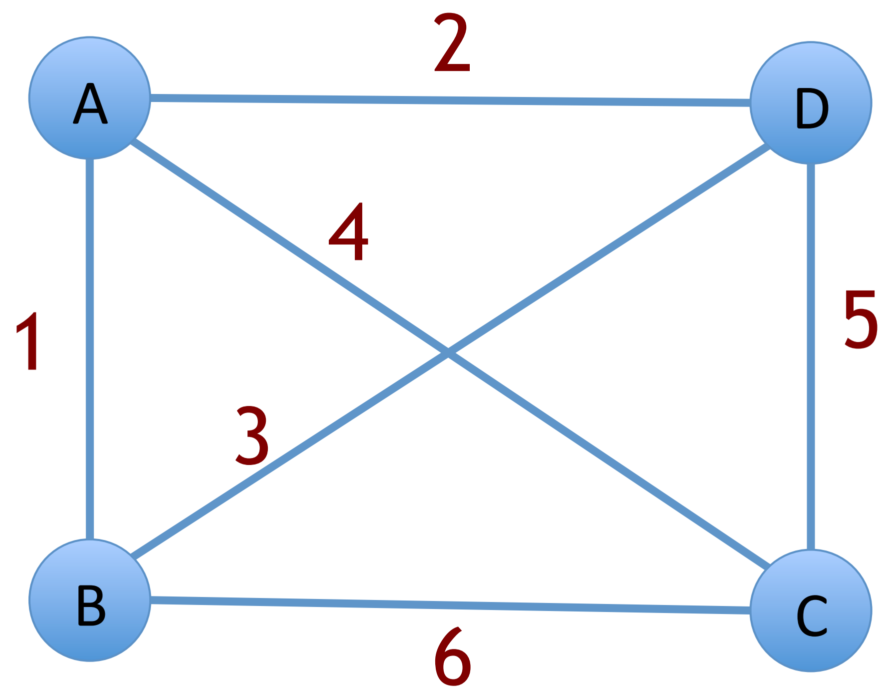


Answer: NO. Any complete cycle has to visit A twice

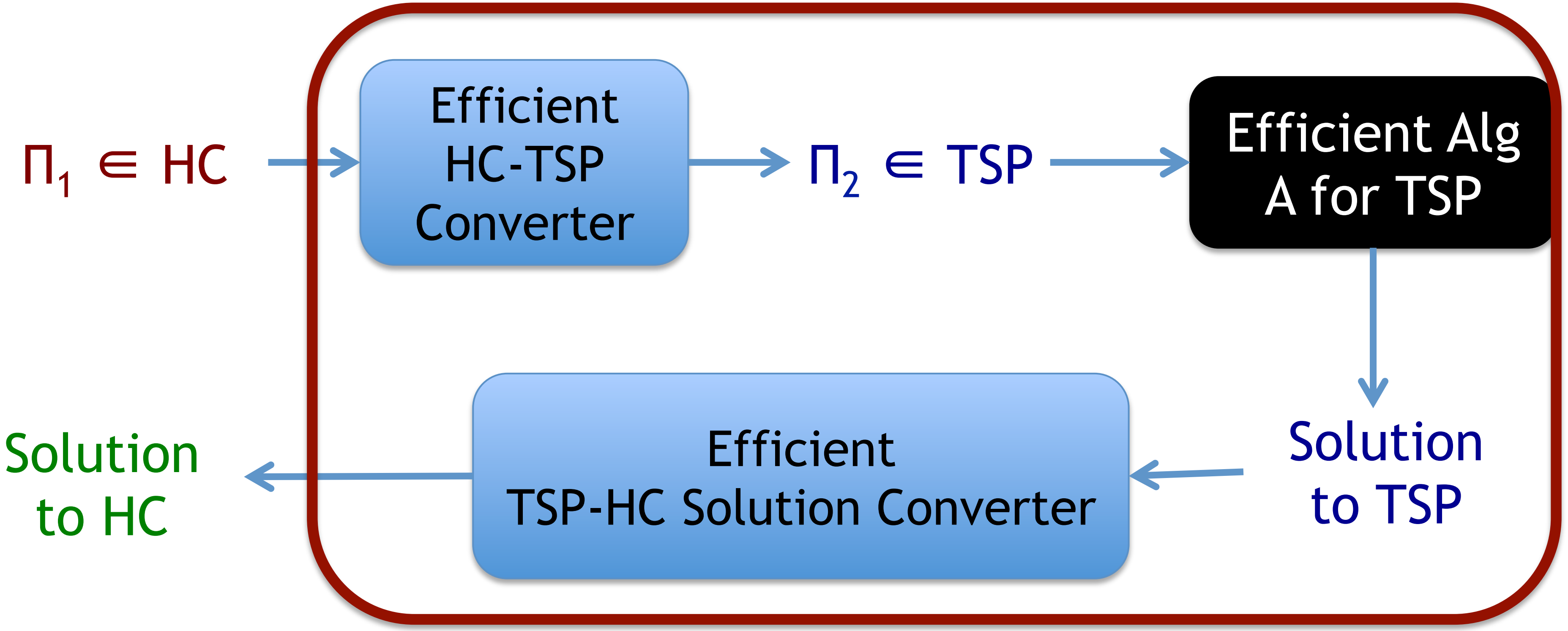
Observation: TSP vs Hamiltonian Cycle

TSP is simply asking for the minimum weight HC.

(or TSP-Decision is asking if a HC with weight < k exists?



HAM-CYCLE $\leq_p$ TSP



Need to show the two converters

HAM-CYCLE $\leq_p$ TSP

Let $G(V, E)$ be the input to HAM-CYCLE

HC-TSP Converter: $\longleftarrow$ Runtime: $O(n^2)$

Let $G^*(V, E^*)$ be a complete graph with edge weights:

$w((u, v)) = 0$ if $(u, v) \in E$

$w((u,v)) = 1$ if $(u, v) \notin E$

TSP-HC Solution Converter: $\longleftarrow$ Runtime: $O(1)$

$\exists$ a Hamiltonian Cycle in G $\Leftrightarrow$ $\exists$ a TSP Tour with weight 0

Proof of Claim

=> If ∃ a Hamiltonian Cycle C in G, then since each edge of C has weight 0 in E*, then C is a tour in G* with weight 0

<= If ∃ a tour T with weight 0 in G*, then all of its edges must be of weight 0, and hence from E, so T is a hamiltonian cycle in G

Q.E.D

Therefore, if we can solve TSP efficiently, we can solve HAM-CYCLE efficiently.

Completing TSP's NP-completeness Proof

If we can solve TSP efficiently, we can solve HAM-CYCLE efficiently

(by just transforming the input G to G* in poly-time & transforming the solution of TSP to HC in poly-time)

since HAM-CYCLE is NP-complete

$\Rightarrow$ we can solve every single NP problem efficiently

$\Rightarrow$ TSP is NP-complete

Q.E.D

Summary:
Overall "Intractability" Argument for TSP

Can't prove (to this day) that TSP is hard in an absolute sense.

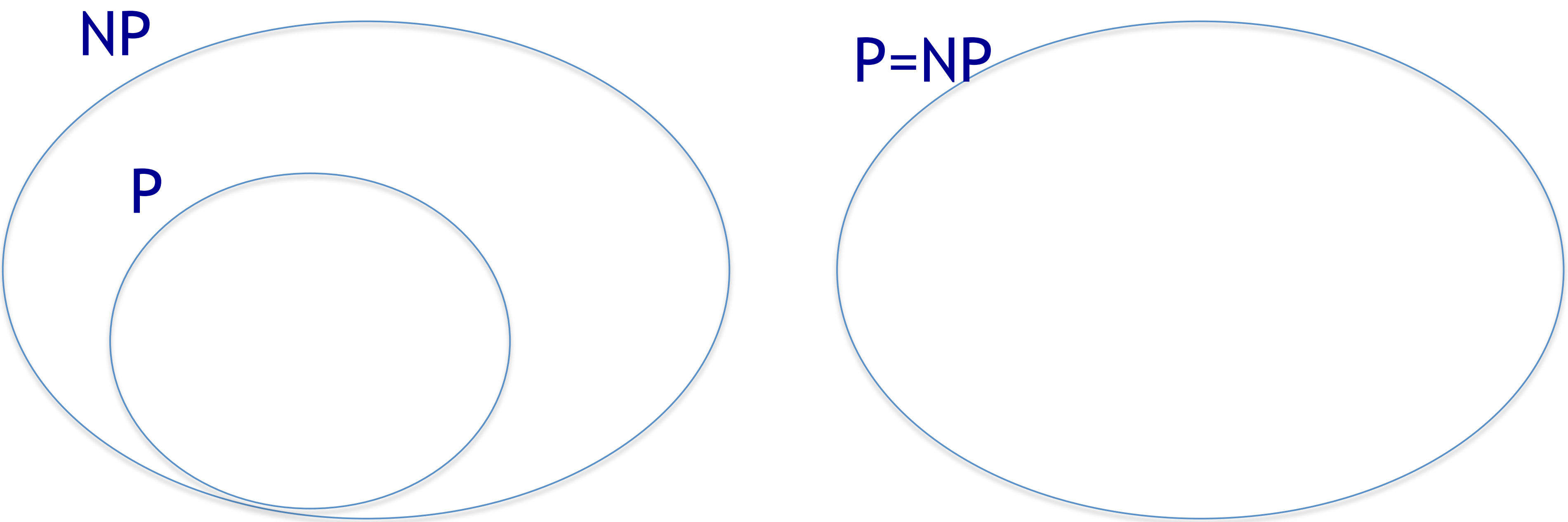
But we argued that TSP is "hard" in a relative sense.

That is we showed TSP is NP-complete => i.e. TSP is as hard as any other NP problem

This is our evidence that TSP is intractable.

Does P = NP?

We know P $\subseteq$ NP. Two possibilities:



Is every problem, whose solutions are efficiently verifiable, is also efficiently solvable?

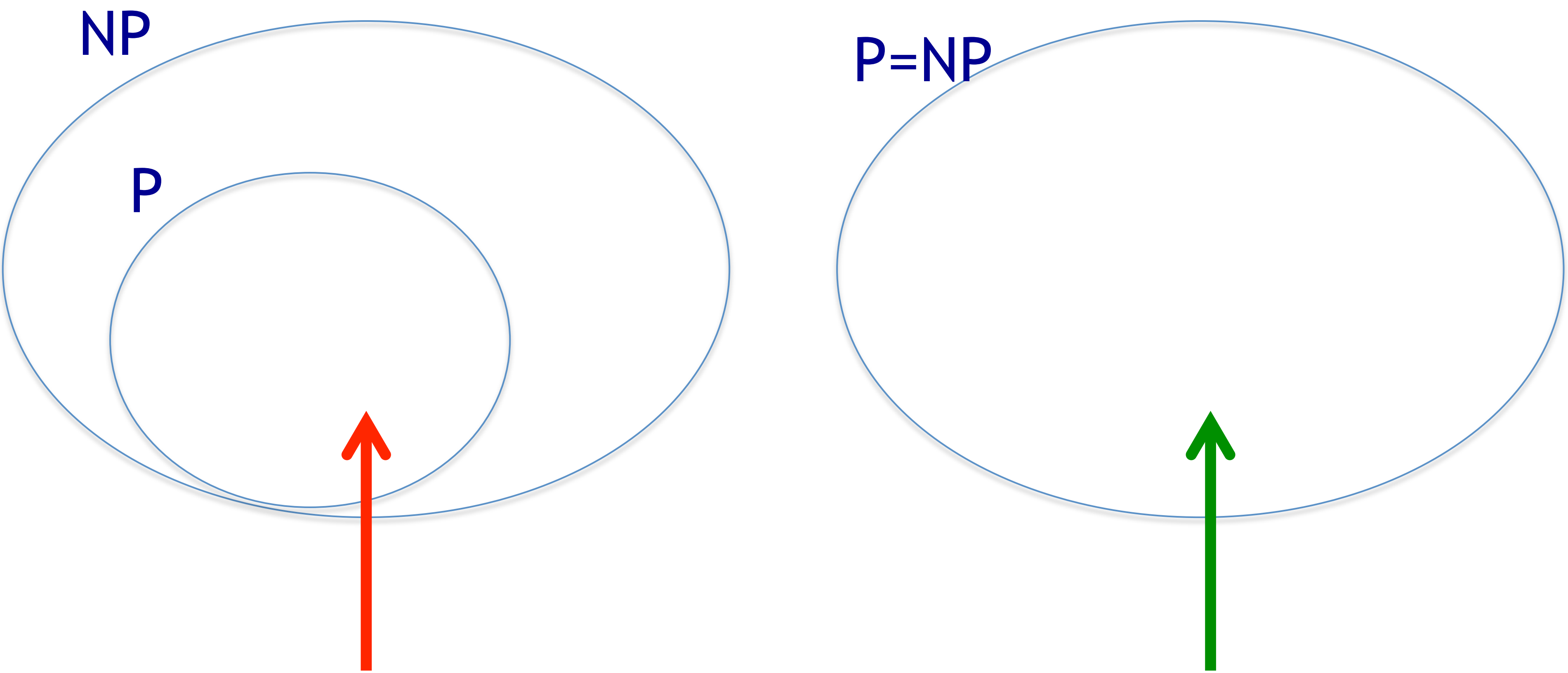
We don't know. But most people believe P $\neq$ NP.

Why Do People Believe P ≠ NP?

1. There are 1000s of NP-complete problems and for none of them is there a poly-time algorithm.
2. Counter to General Human Experience
3. Also some weird mathematical consequences, such as polynomial hierarchy collapsing, and others.

But We Simply Don't Know (Yet)!

How could we resolve P vs NP?



Prove an NP-complete problem is NOT solvable in poly time.

Prove an NP-complete problem is solvable in poly time.

We don’t know which world we live in.

Your Problem is NP-complete. Now What?

- Option 1: Focus to special-case inputs.
  - Ex: Independent Set is NP-complete.
  - Focusing on line graphs, had a O(n) DP alg.
- Option 2: Find an approximate answer.
  - Will show a very simple algorithm for 0-1 Knapsack.
- Option 3: Be exponential time but better than brute-force search.
  - 0-1 Knapscak O(nW) runtime DP algorihm.
- Option 4: Heuristics: fast algorithms that are not always correct (or even approximate)
- Option 5: Mix some of these options