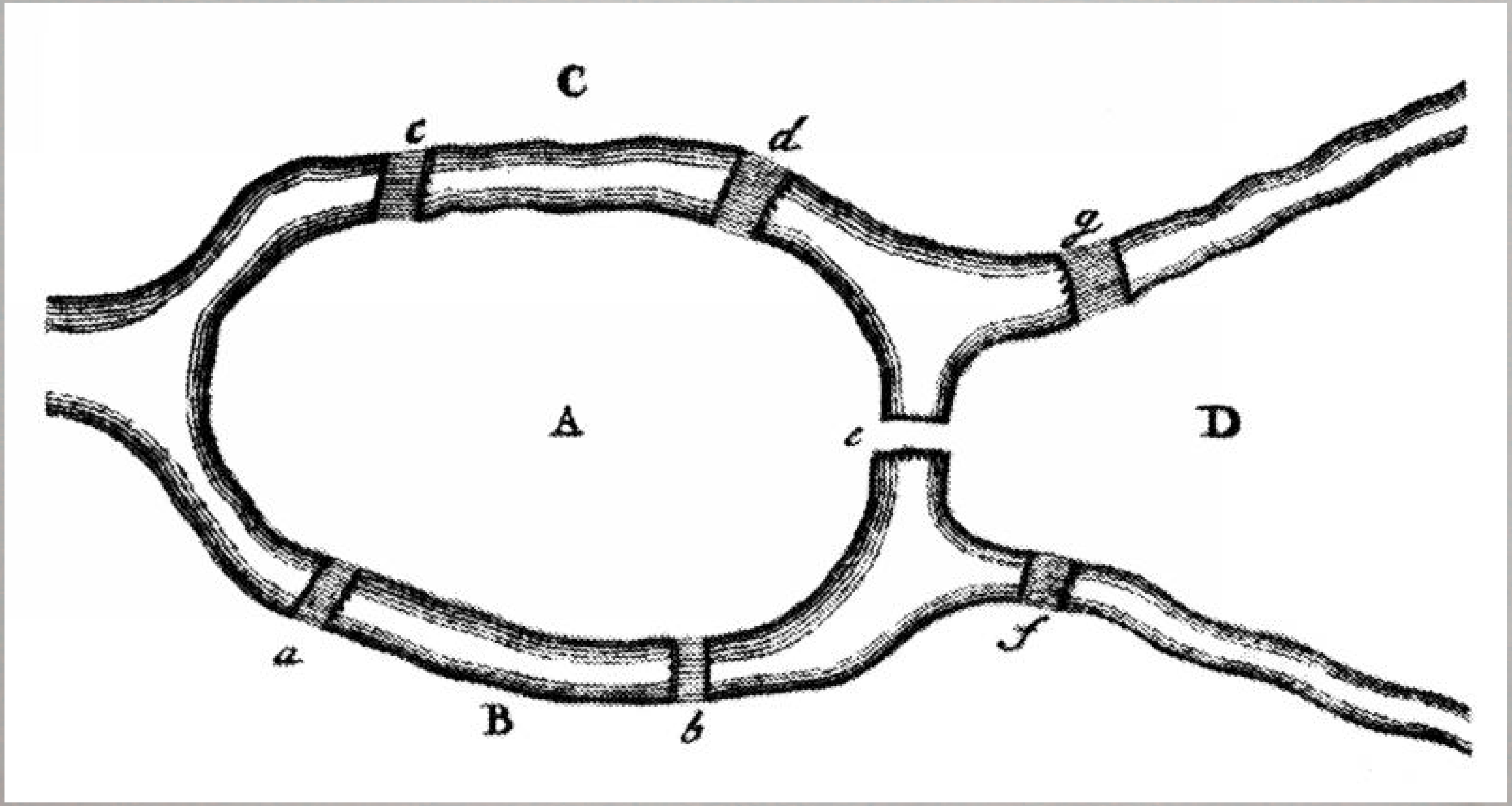
Computational Complexity

Cristopher Moore
University of New Mexico
and the Santa Fe Institute

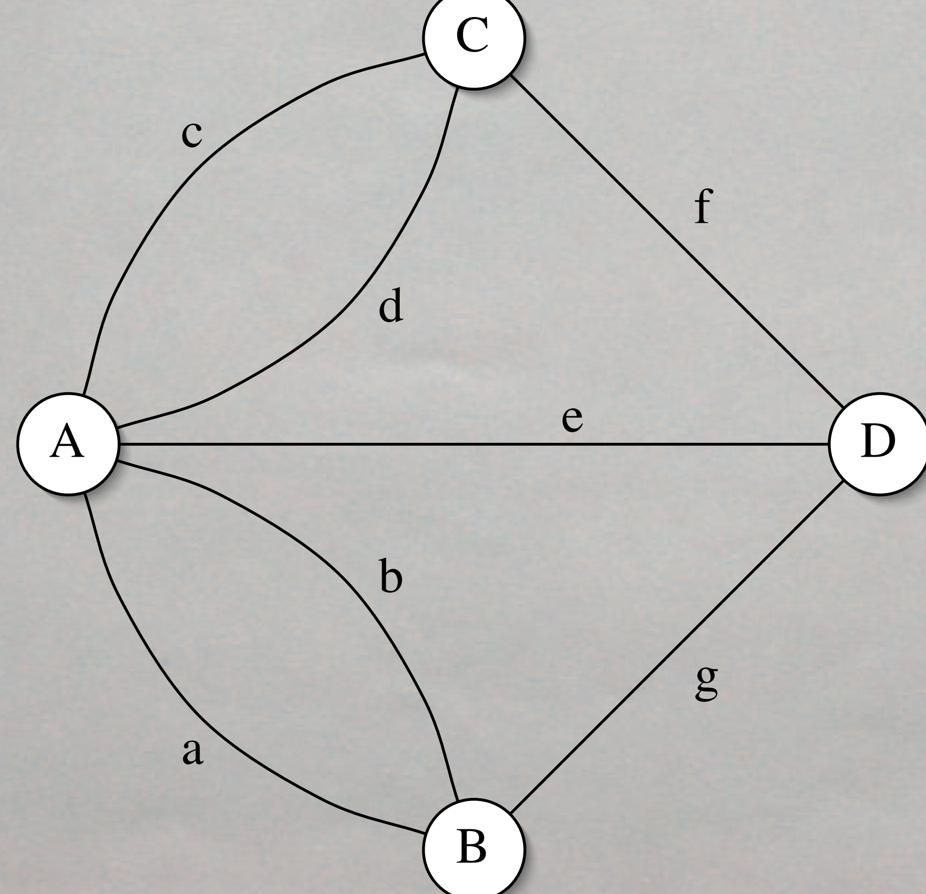
Computational Complexity

- Why are some problems qualitatively harder than others?



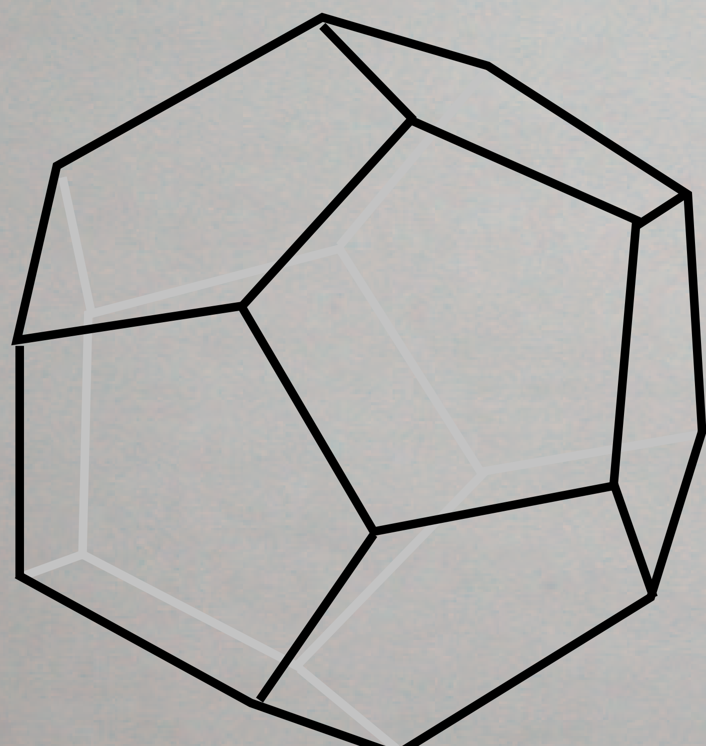
Computational Complexity

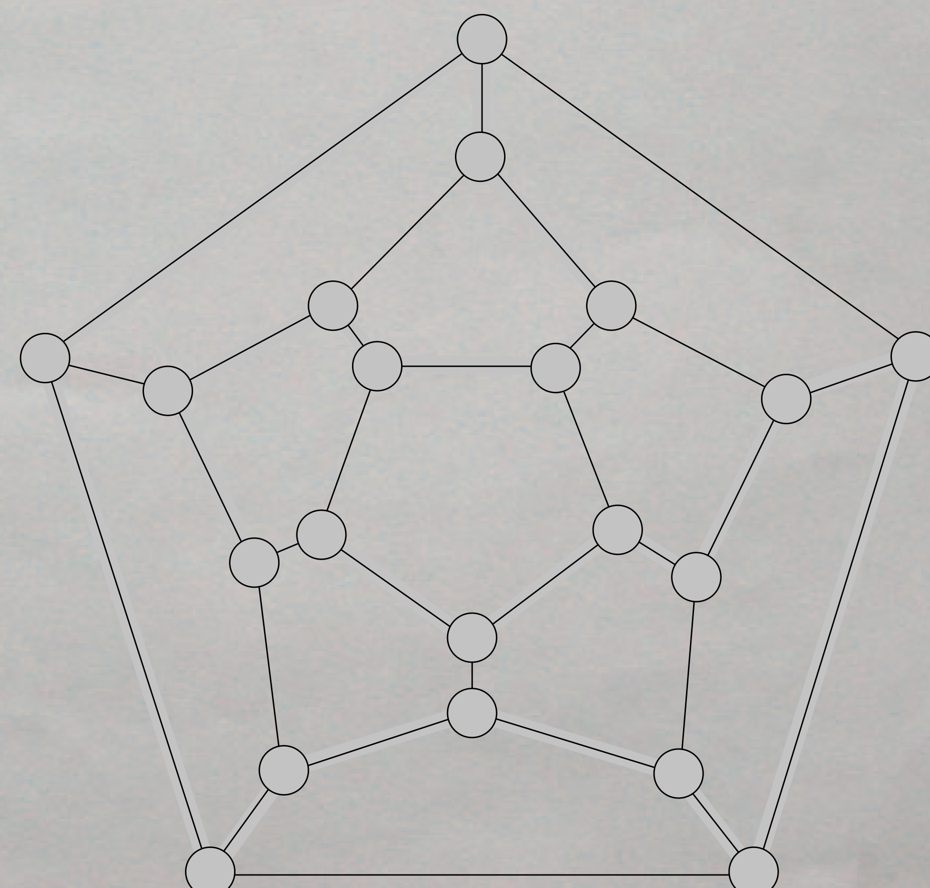
- A simple insight: at most 2 vertices can have odd degree, so no tour is possible!



Computational Complexity

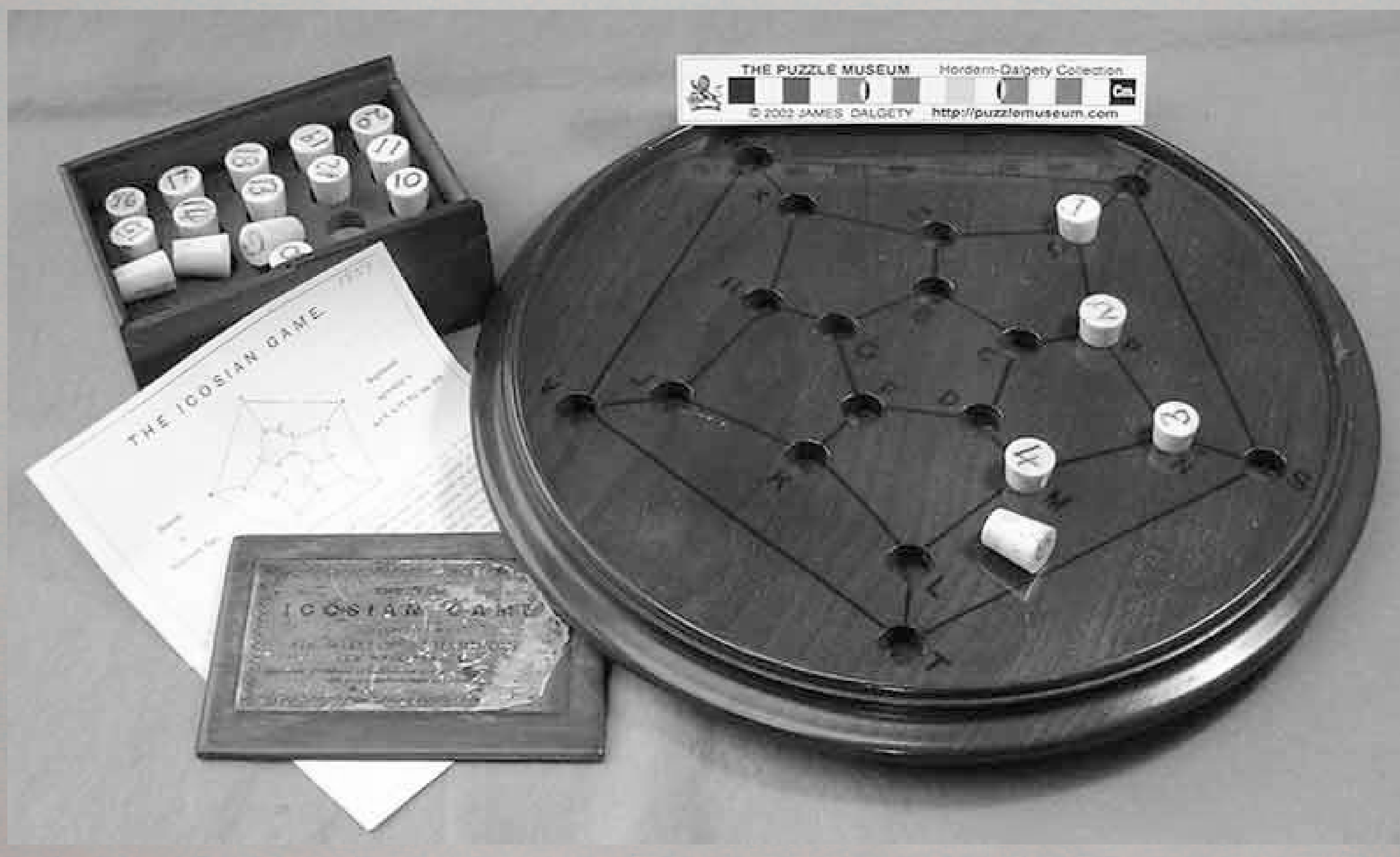
- What if we want to visit every vertex, instead of every edge?





Computational Complexity

- As far as we know, the only way to solve this problem is (essentially) exhaustive search!



Needles in Haystacks

- **P**: we can find a solution efficiently
- **NP**: we can *check* a solution efficiently

Complexity Classes

NP

Hamiltonian Path

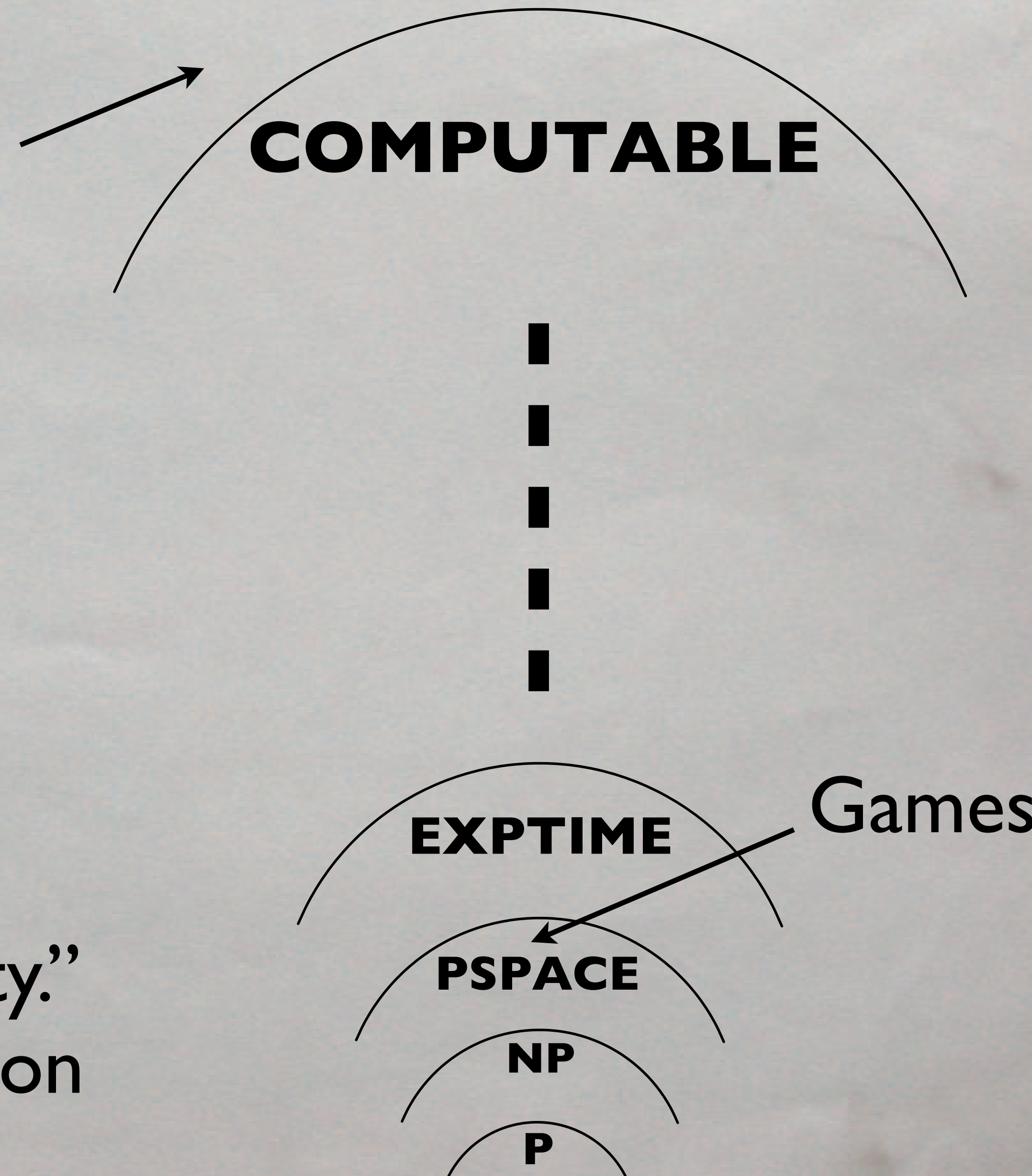
P

Eulerian Path

Multiplication

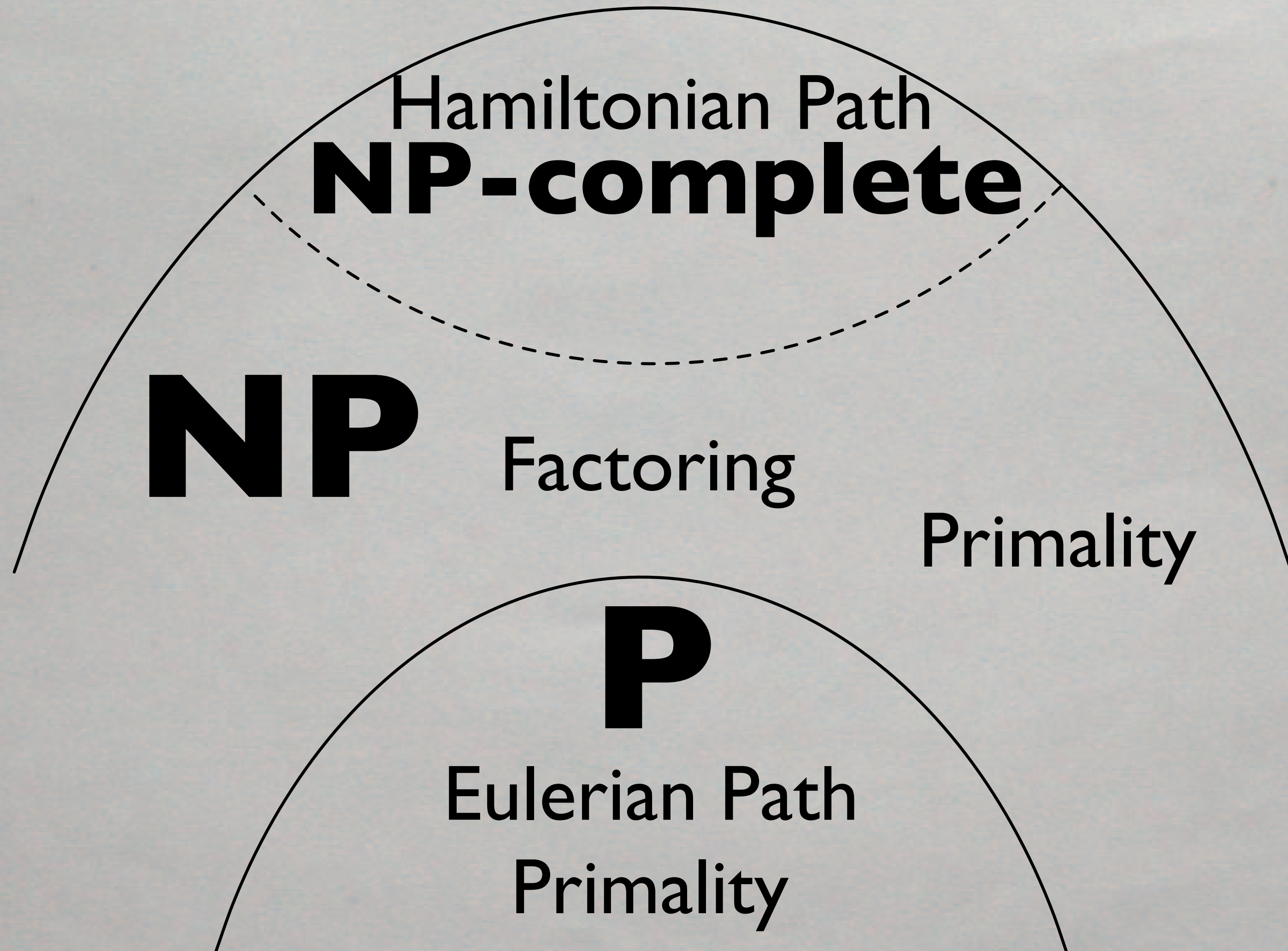
An Infinite Hierarchy

Turing's Halting Problem



"Computers play the same role in complexity that clocks, trains and elevators play in relativity."
– Scott Aaronson

The Hardest of Them All



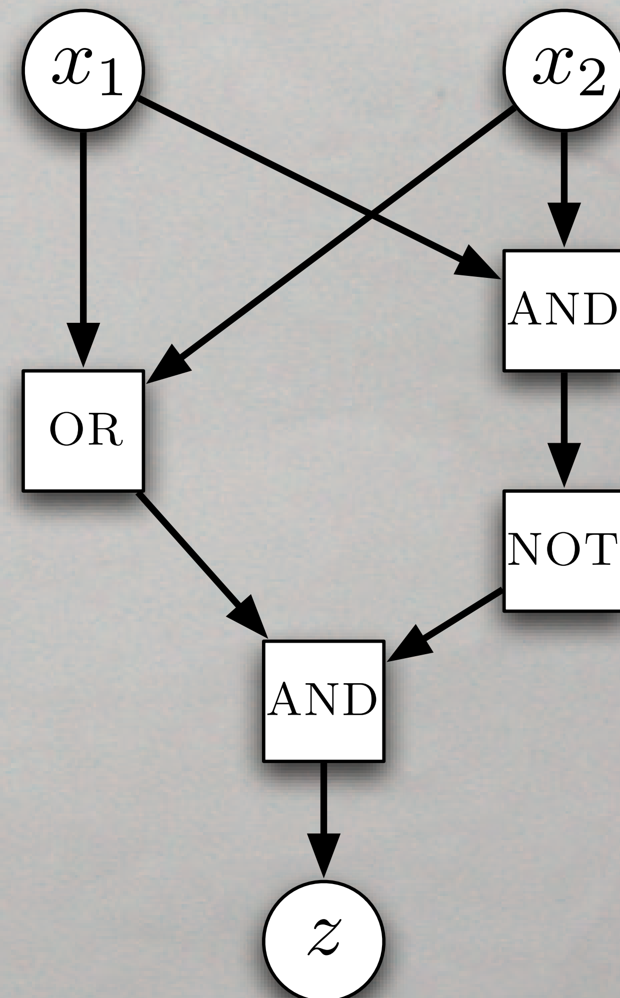
Some problems in **NP** capture the entire class!
If we can solve any of them efficiently, then **P=NP.**

Satisfying a Circuit

Any program that tests solutions (e.g. paths) can be "compiled" into a Boolean circuit

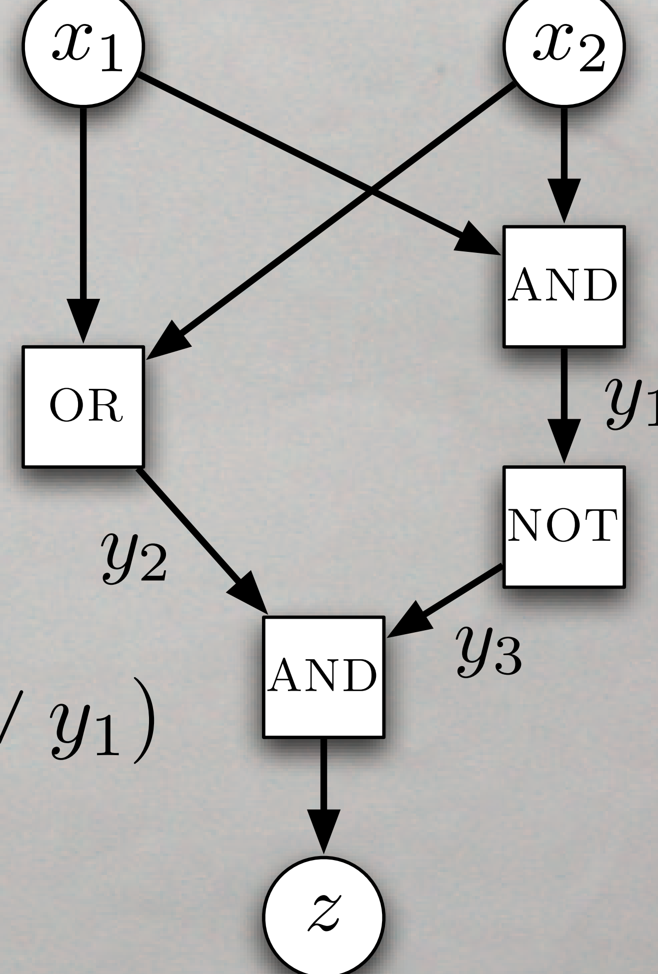
The circuit outputs "true" if an input solution works

Is there a set of values for the inputs that makes the output true?



From Circuits to Formulas

The condition that each AND or OR gate works, and the output is "true," can be written as a Boolean formula:



$$(x_1 \vee \overline{y}_1) \wedge (x_2 \vee \overline{y}_1) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee y_1)$$

$$\wedge \cdots \wedge z \quad .$$

3-SAT

- Our first NP-complete problem!
- Given a set of *clauses* with 3 variables each,

$$(x_1 \vee \overline{x}_2 \vee x_3) \wedge (x_2 \vee x_{17} \vee \overline{x}_{293}) \wedge \cdots$$

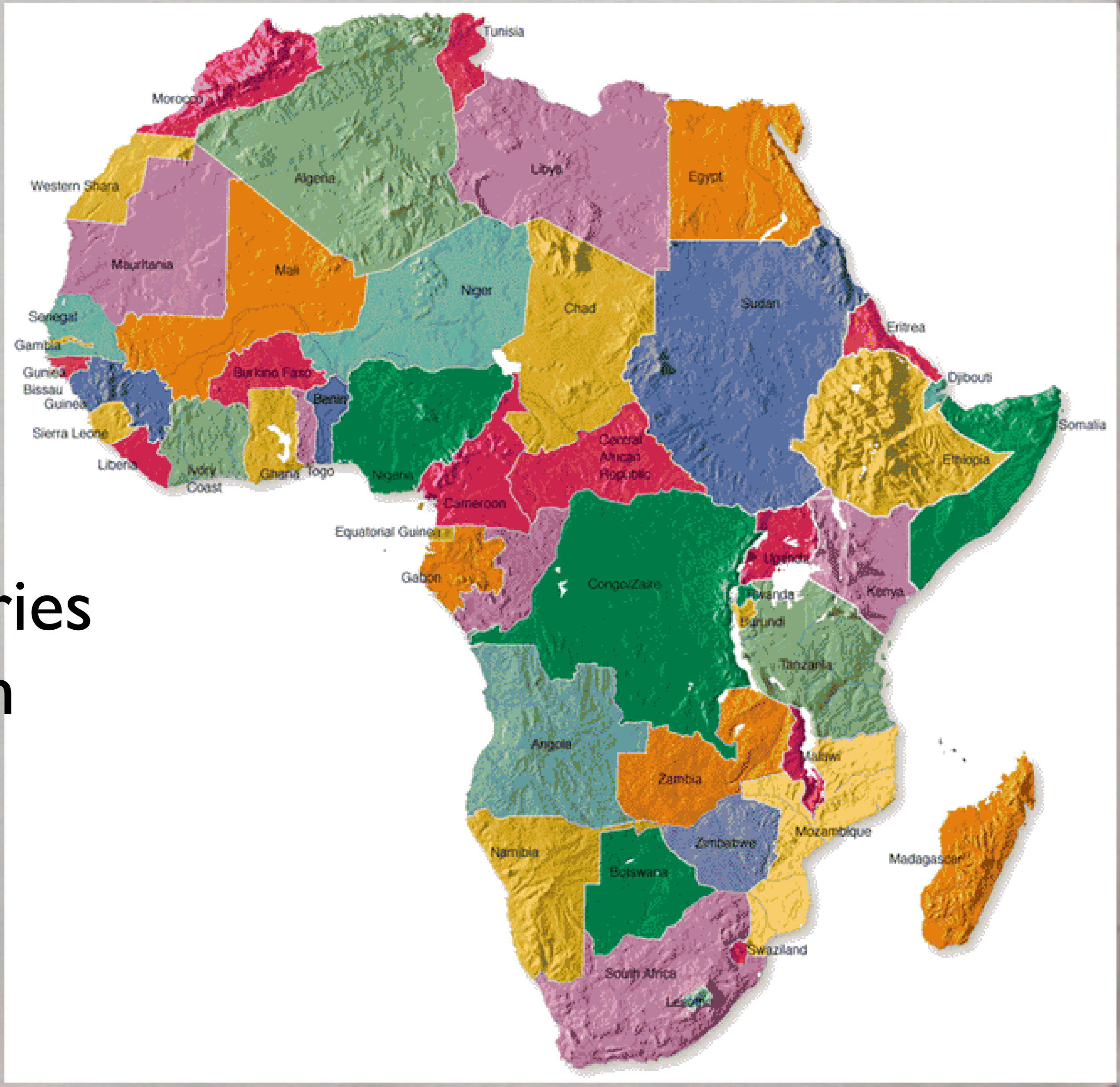
does a set of truth values for the x_i exist such that all the clauses are satisfied?

- k-SAT (k variables per clause) is NP-complete for $k \geq 3$.

If 3-SAT Were Easy...

- ...we could convert any problem in NP to a circuit that tests solutions
- ...and convert that circuit to a 3-SAT formula which is satisfiable if a solution exists
- ...and use our efficient algorithm for 3-SAT to solve it!
- So, if 3-SAT is in **P**, then all of **NP** is too, and **P=NP!**

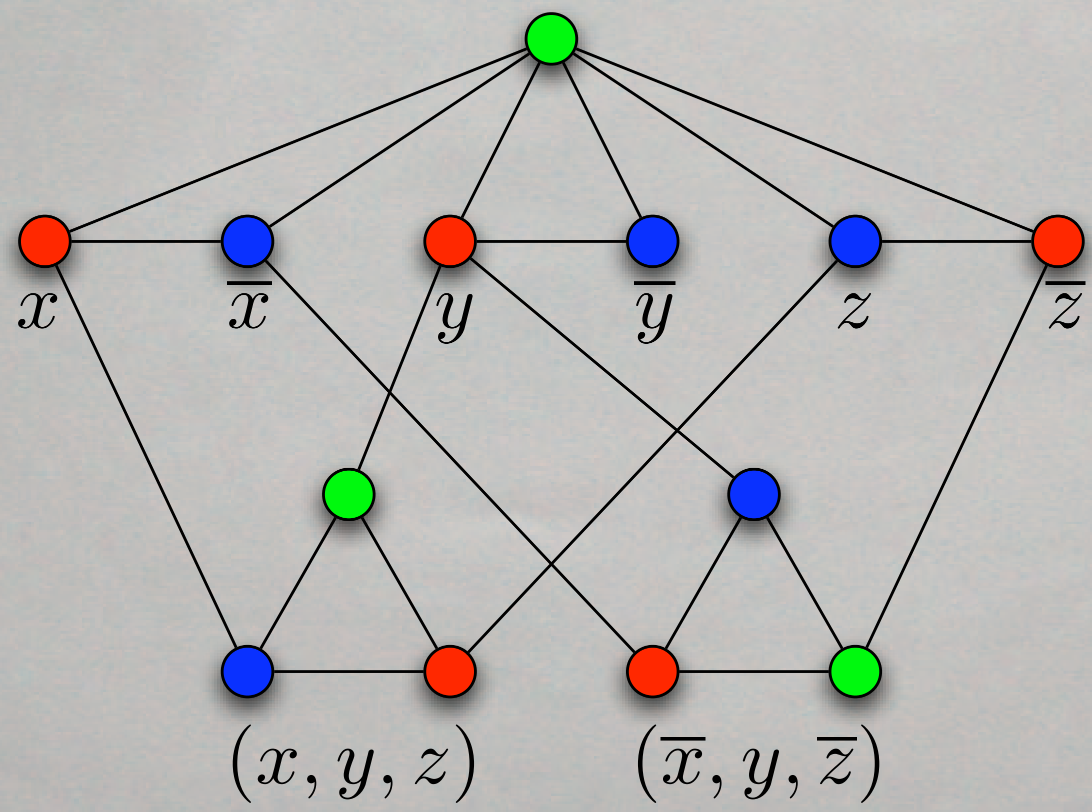
Graph Coloring



Given a set of countries and borders between them, what is the smallest number of colors we need?

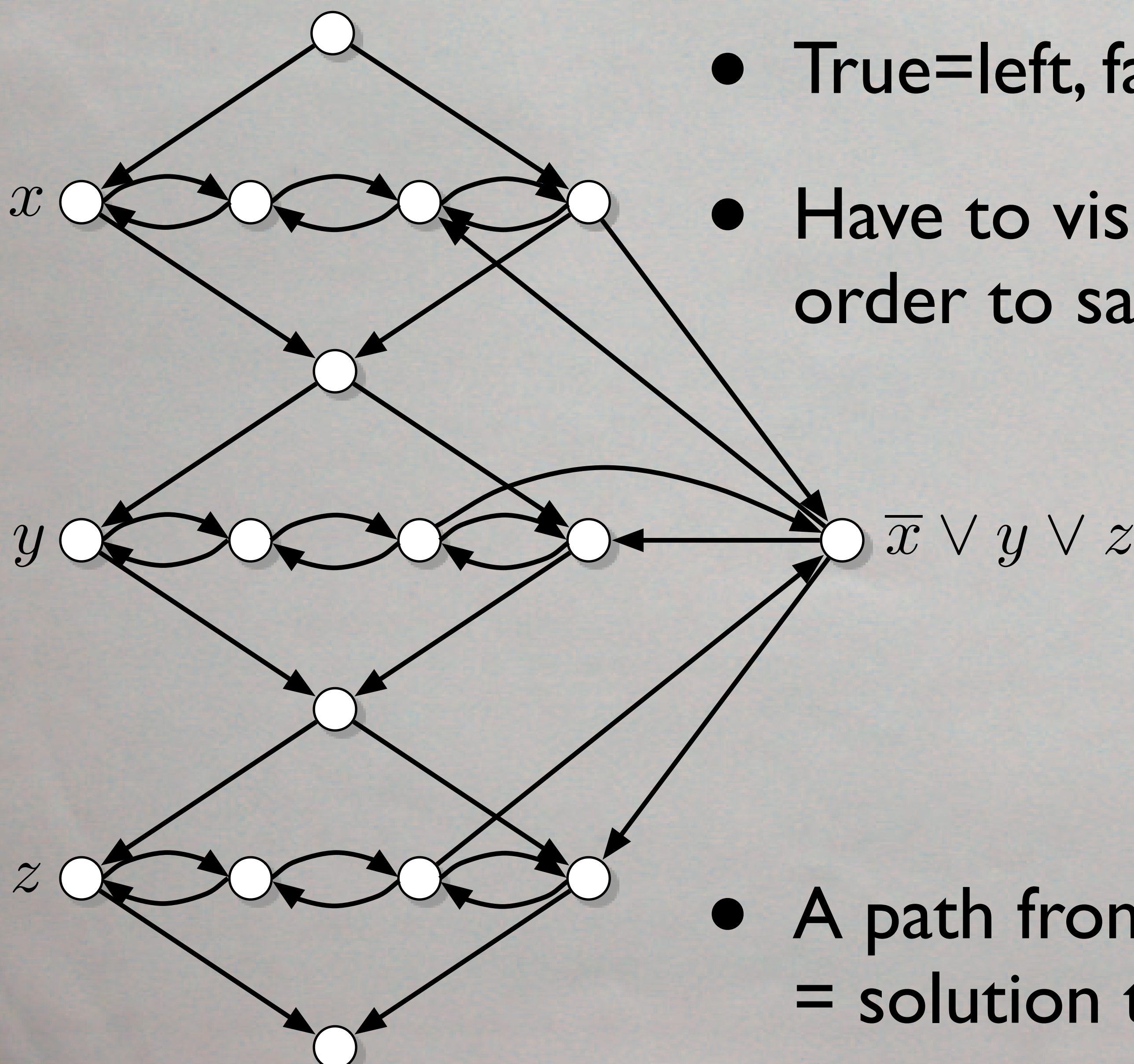
From SAT to Coloring

- “Gadgets” enforce constraints:



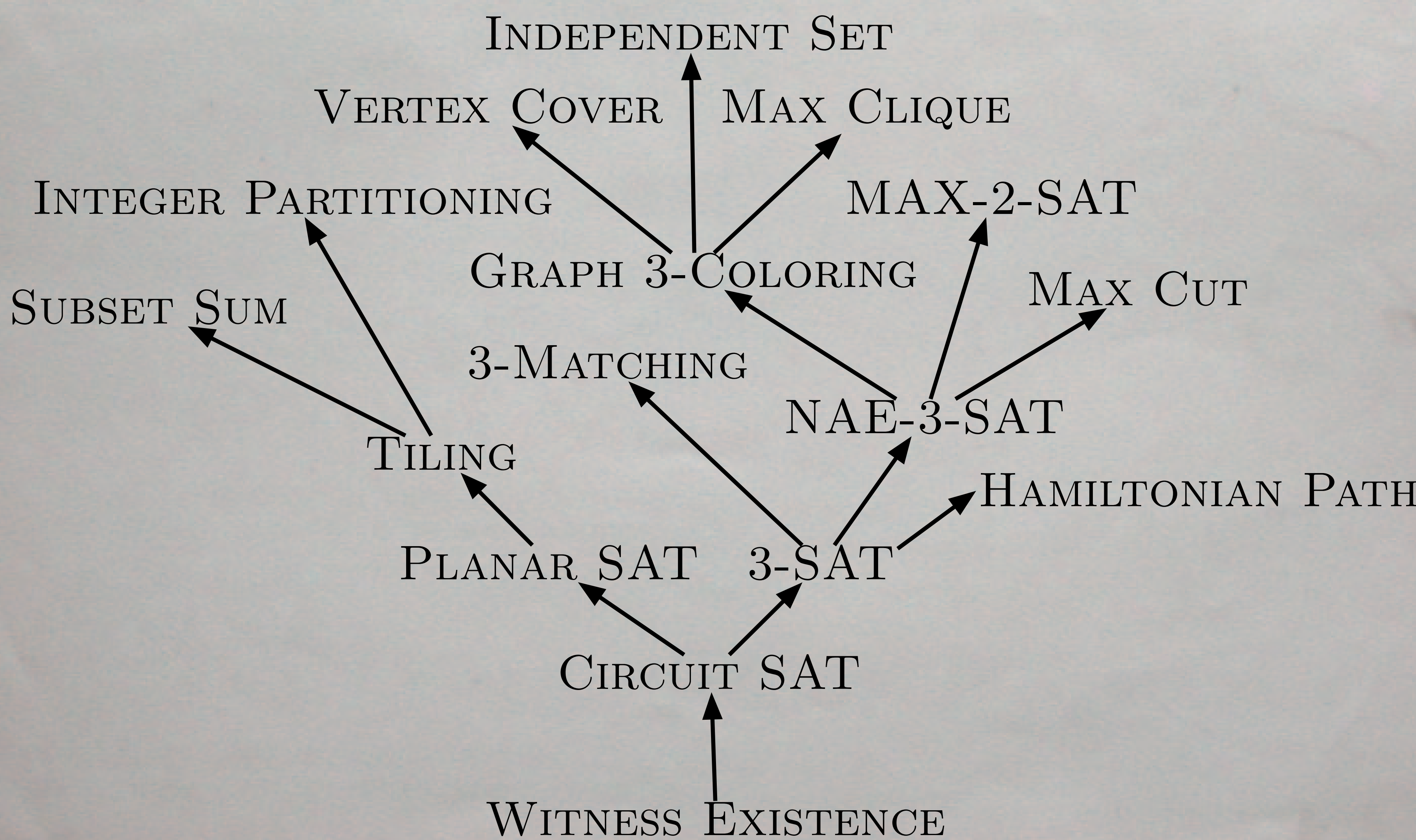
- Graph 3-Coloring is NP-complete
- Graph 2-Coloring is in **P** (why?)

Traveling Salespeople



- True=left, false=right
- Have to visit clause vertices in order to satisfy them
- A path from top to bottom = solution to 3-SAT problem!

And so on...


Independent Set
Vertex Cover
Max Clique
Integer Partitioning
MAX-2-SAT
Graph 3-Coloring
Max Cut
Subset Sum
3-Matching
NAE-3-SAT
Tiling
Hamiltonian Path
Planar SAT
3-SAT
Circuit SAT
Witness Existence

Deep Questions

- Is finding solutions harder than checking them? When can we avoid exhaustive search?
- What happens if we only need good answers, instead of the best ones? Are there problems where even finding good answers is hard?
- How much does it help to do many things at once (parallelism)?
- How much does randomness help? Can we foil the adversary by being unpredictable?
- How much does quantum physics help?

Clay Mathematics Institute

Dedicated to increasing and disseminating mathematical knowledge

How to make $1,000,000 (and maybe $7,000,000)

Millennium Problems

- **P=NP**?
- Poincaré Conjecture
- Riemann Hypothesis
- Yang-Mills Theory
- Navier-Stokes Equations
- Birch and Swinnerton-Dyer Conjecture
- Hodge Conjecture

Millennium Problems

- **P=NP**?

Millennium Problems

- **P=NP**?
- Is it harder to find solutions than to check them?
- Question about the *nature of mathematical truth*
- ...and whether finding it requires as much creativity as we think.

What if P=NP?

- Better Traveling Salesmen, can pack luggage
- No Cryptography
- The entire *polynomial hierarchy* collapses, NEXP=EXP, etc.
- We can find (up to any reasonable length)
    - Proofs
    - Theories
    - *Anything* we can recognize.

Gödel to Von Neumann

Let $\varphi(n)$ be the time it takes to decide whether a proof of length n exists. Gödel writes:

The question is, how fast does $\varphi(n)$ grow for an optimal machine. If there actually were a machine with, say, $\varphi(n) \sim n^2$, this would have consequences of the greatest magnitude. That is to say, it would clearly indicate that, despite the unsolvability of the *Entscheidungsproblem*, **the mental effort of the mathematician in the case of yes-or-no questions could be completely replaced by machines**. One would simply have to select an n large enough that, if the machine yields no result, there would then be no reason to think further about the problem.

P≠NP if understanding matters.

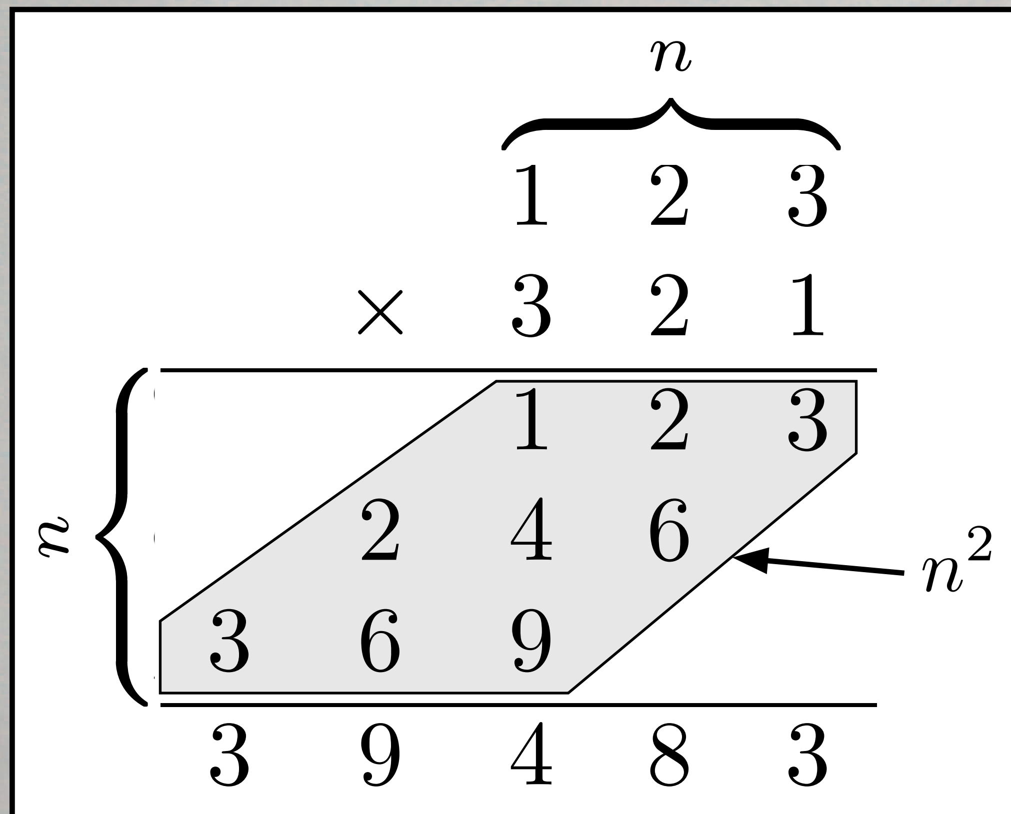
P≠NP iff we can't just "turn the crank"

Upper Bounds are Easy; Lower Bounds are Hard

- Why is the P vs. NP question so hard?
- Algorithms are *upper bounds* on complexity...
- ...but how do you know if you have the best algorithm?

Algorithmic Surprises

- Grade school multiplication takes $O(n^2)$ time:



- Can we do better?

Algorithmic Surprises

- Divide and conquer:

$$x = 2^{n/2}a + b,\ y = 2^{n/2}c + d$$

$$xy = 2^n ac + 2^{n/2}(ad + bc) + bd$$

Looks like we need ac, ad, bc, bd. But

$$(a + b)(c + d) - ac - bd = ad + bc$$

so we only need three products. Running time:

$$T(n) \approx 3T(n/2)$$

so $T(n) \sim n^{\log_2 3} = n^{1.58}$. How low can we go?

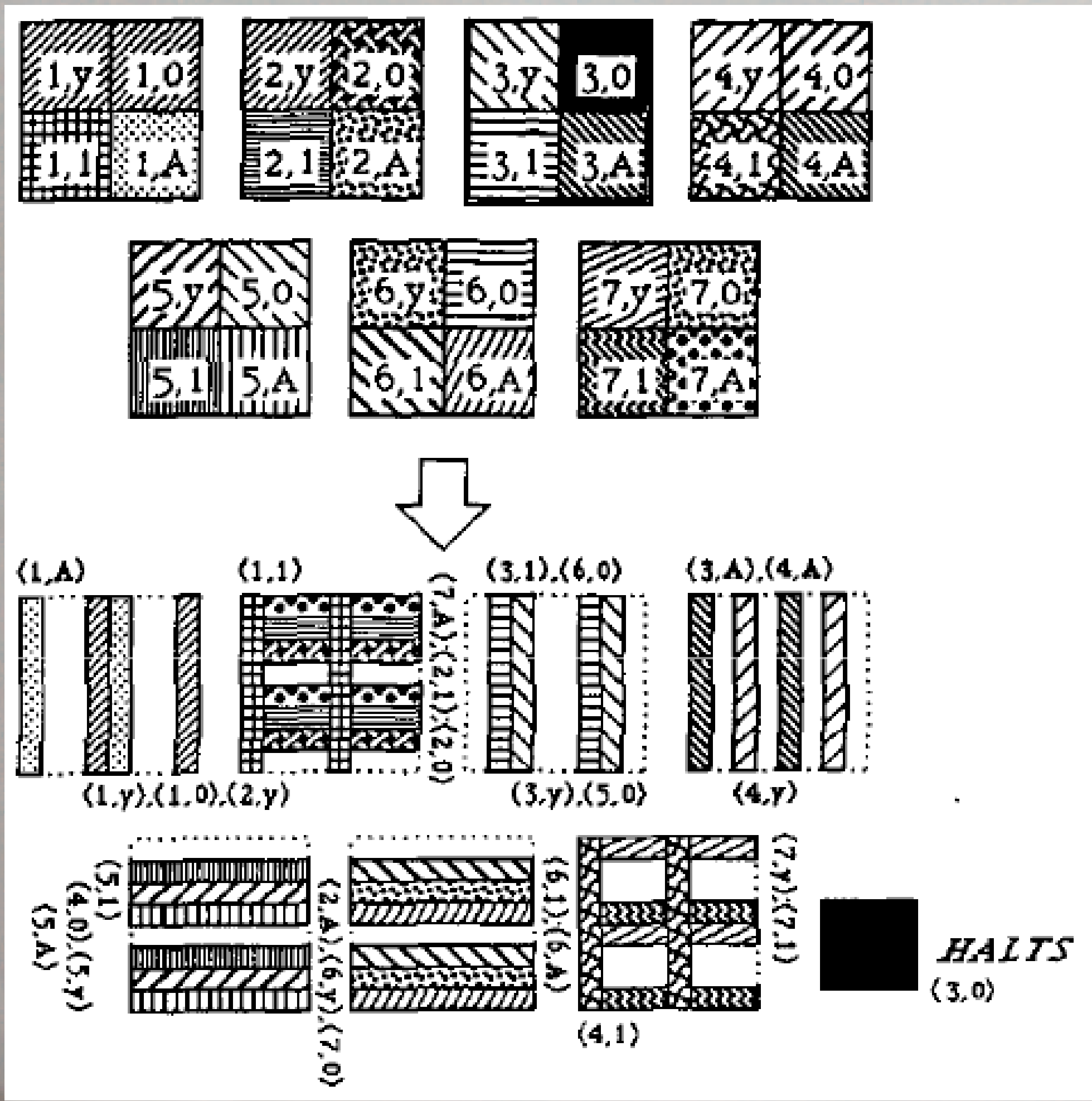
The Undecidable

- Suppose we could tell whether a program *p*, given an input *x*, will ever halt.

```
trouble(p):
  if halt(p,p) loop forever
  else halt
```

- Will `trouble(trouble)` halt or not?
- Undecidable problems ⇒ unprovable truths!

Worse Than Chaos



- Will an initial point *x* ever halt?
- Is *x* periodic?
- Undecidable, even if initial conditions are known exactly!

Parallelization

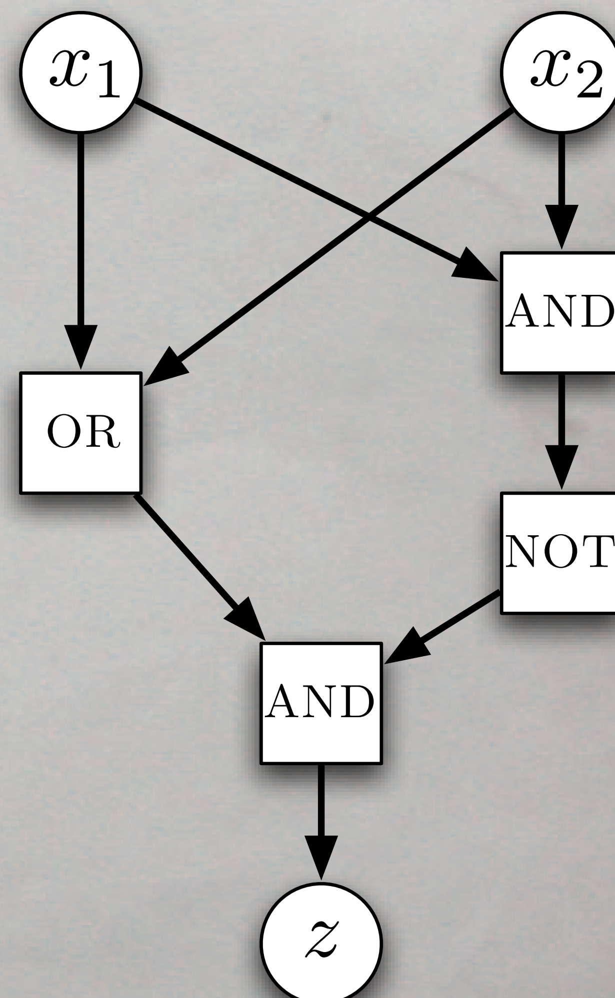
- What if we can do many things at once?
- **NC** is the subset of **P** consisting of problems that can be *efficiently parallelized:*
- With poly(n) processors, we can solve problems in *polylogarithmic* time: $(\log n)^k$ for some constant k
- Example: matrix multiplication

Deep vs. Shallow

- Parallel computers are like circuits: time = depth, #processors = width

- The **P** vs. **NC** problem: can every circuit be compressed to depth $(\log n)^k$ and poly width?

- We believe that **P-complete** problems are "inherently sequential," so that NC $\neq$ P



Randomness

- *Randomized* algorithms: use random numbers, and guarantee the right answer with probability 2/3

- **BPP** (**B**ounded **P**robability **P**olynomial time) is the class of problems that can be solved this way

- The **P** vs. **BPP** problem: are there deterministic pseudorandom number generators that are "good enough"?

- We think so, but this is hard to prove...

Beyond NP

- Does Black have a winning strategy? If so, how could we prove it?

- Black has a move such that, no matter what White does, Black has a move such that...
- Deep logical structure!

  $\forall x{:}\exists y{:}\forall z{:}\exists w{:}\ \ldots$

- But even $P \neq PSPACE$ isn't known.

Shameless Plug