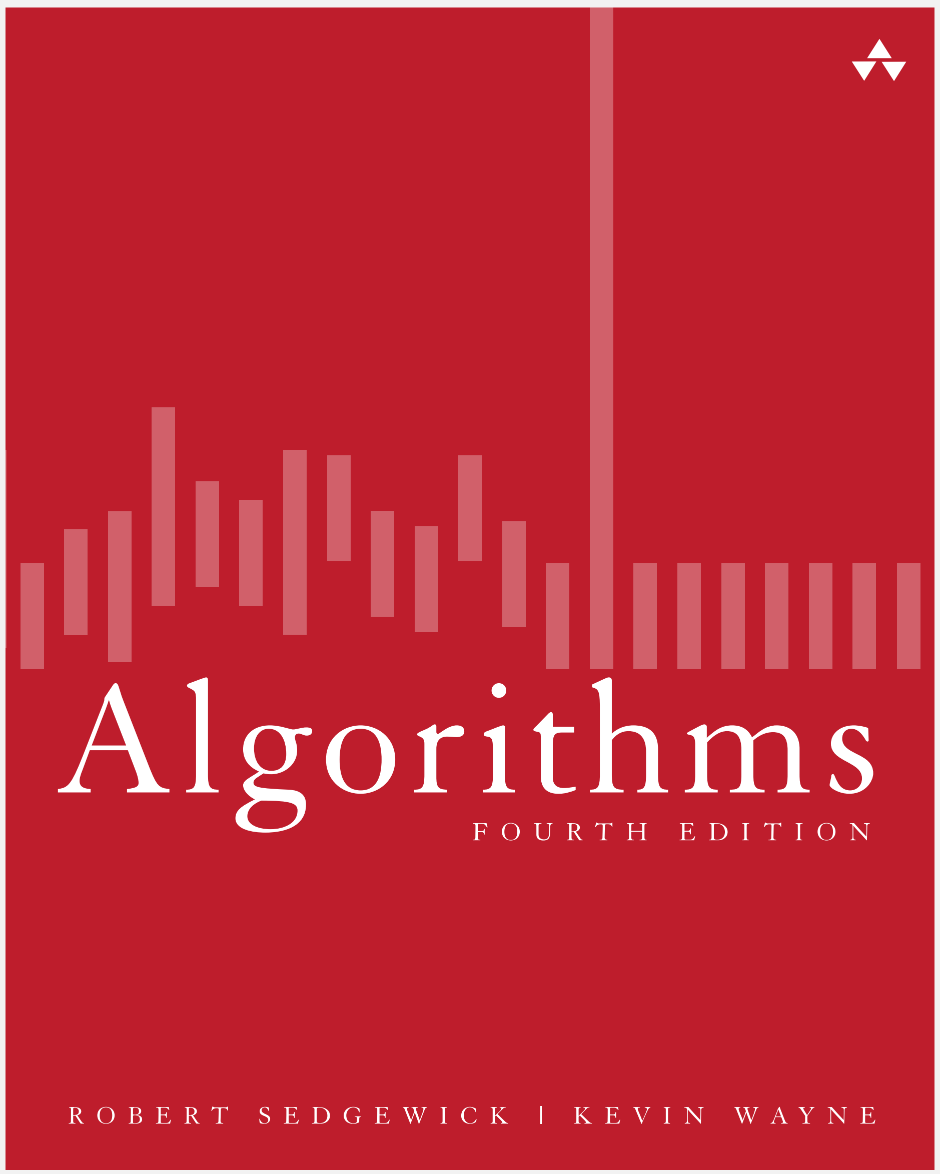
# 7.5 Reductions




- designing algorithms
- establishing lower bounds
- intractability

 ·  · 

**Integer multiplication.** Given two $N$-bit integers, compute their product.

**Brute force.** $N^2$ bit operations.

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| | | | | | | | × | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| | | | | | | | | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| | | | | | | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | | |
| | | | | | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | | | |
| | | | | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | | | | |
| | | | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | | | | | |
| | | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | | | | | | |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | | | | | | |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

**Q.** Is grade-school algorithm optimal?

# Integer arithmetic reductions

**Integer multiplication.** Given two $N$-bit integers, compute their product.

**Brute force.** $N^2$ bit operations.

**Karatsuba-Ofman (1962)** $N^{1.585}$ bit operations.

**Schönhage–Strassen (1971).** $N \log N \log \log N$ bit operations.

**Fürer (2007).** $N \log N \, 2^{\log^* N}$ bit operations.

| problem | arithmetic | order of growth |
| --- | --- | --- |
| integer quotient | a / b | IM(N) |
| integer remainder | a mod b | IM(N) |
| integer square | $a^2$ | IM(N) |

Matrix multiplication. Given two $N$-by-$N$ matrices, compute their product.

Brute force. $N^3$ flops.

$$
\begin{array}{r}
\\ \text{row } i \\ \\ \\
\end{array}
\begin{bmatrix}
0.1 & 0.2 & 0.8 & 0.1 \\
0.5 & 0.3 & 0.9 & 0.6 \\
0.1 & 0 & 0.7 & 0.4 \\
0 & 0.3 & 0.3 & 0.1
\end{bmatrix}
\times
\begin{array}{c}
\begin{array}{cccc} & & \text{column } j & \end{array} \\
\begin{bmatrix}
0.4 & 0.3 & 0.1 & 0.1 \\
0.2 & 0.2 & 0 & 0.6 \\
0 & 0 & 0.4 & 0.5 \\
0.8 & 0.4 & 0.1 & 0.9
\end{bmatrix}
\end{array}
=
\begin{array}{r}
\\ i \\ \\ \\
\end{array}
\begin{array}{c}
\begin{array}{cccc} & & j & \end{array} \\
\begin{bmatrix}
0.16 & 0.11 & 0.34 & 0.62 \\
0.74 & 0.45 & 0.47 & 1.22 \\
0.36 & 0.19 & 0.33 & 0.72 \\
0.14 & 0.1 & 0.13 & 0.42
\end{bmatrix}
\end{array}
$$

Q. Is grade-school algorithm optimal?

**Matrix multiplication.** Given two $N$-by-$N$ matrices, compute their product.

**Brute force.** $N^3$ flops.

**Strassen (1969).** $N^{2.81}$ flops.

**Coppersmith-Winograd (1987).** $N^{2.376}$ flops.

| problem | linear algebra | order of growth |
|---|---|---|
| matrix inversion | $A^{-1}$ | MM(N) |
| system of linear equations | $Ax = b$ | MM(N) |
| LU decomposition | $A = LU$ | MM(N) |
| least squares | $\min \lVert Ax - b \rVert_2$ | MM(N) |

Desiderata. Classify problems according to computational requirements.

| complexity | order of growth | examples |
| --- | --- | --- |
| linear | $N$ | min, max, median, Burrows-Wheeler transform, ... |
| linearithmic | $N \log N$ | sorting, convex hull, closest pair, farthest pair, ... |
| quadratic | $N^2$ | ??? |
| | … | |
| exponential | $c^N$ | ??? |

Frustrating news. Huge number of problems have defied classification.

Desiderata. Classify problems according to computational requirements.

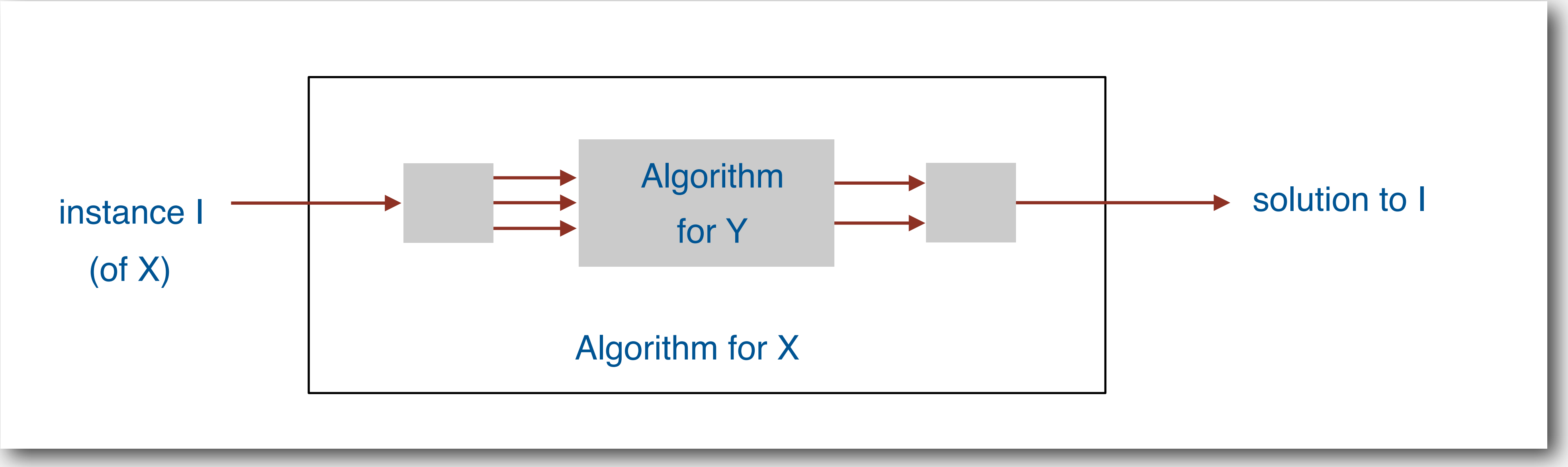
Desiderata'.

Suppose we could (could not) solve problem X efficiently.

What else could (could not) we solve efficiently?

*" Give me a lever long enough and a fulcrum on which to place it, and I shall move the world. "* — *Archimedes*

**Def.** Problem $X$ **reduces to** problem $Y$ if you can use an algorithm that solves $Y$ to help solve $X$.

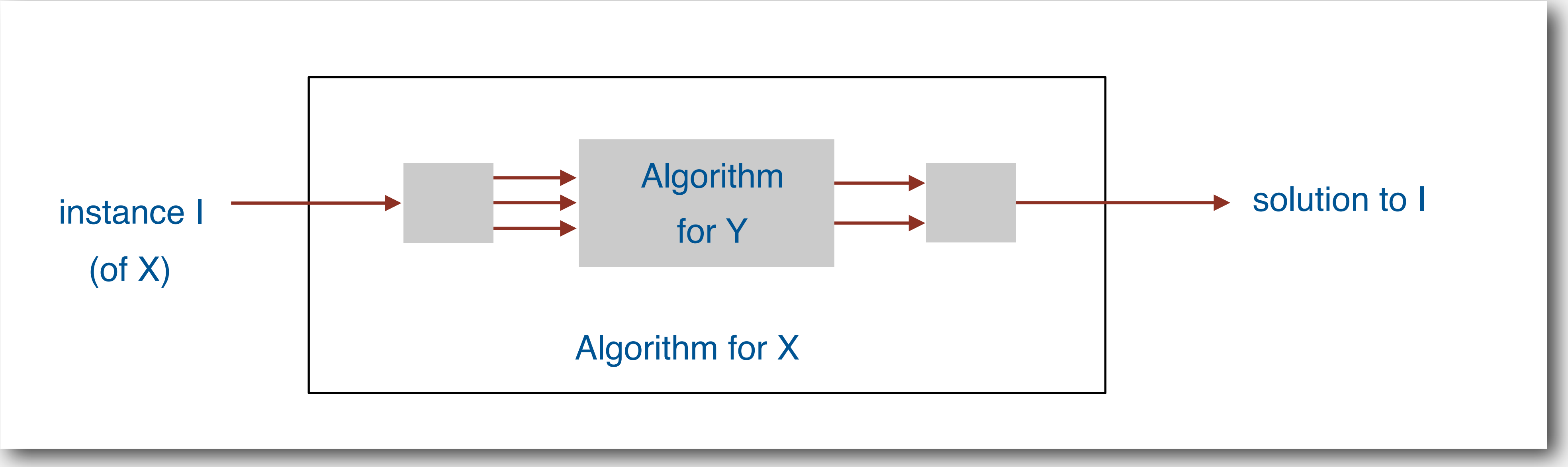



Cost of solving $X$ = total cost of solving $Y$ + cost of reduction.

perhaps many calls to Y on problems of different sizes

preprocessing and postprocessing

**Def.** Problem $X$ reduces to problem $Y$ if you can use an algorithm that solves $Y$ to help solve $X$.




**Ex 1.** [element distinctness reduces to sorting]

To solve element distinctness on $N$ integers:

- Sort $N$ integers.
- Check adjacent pairs for equality.

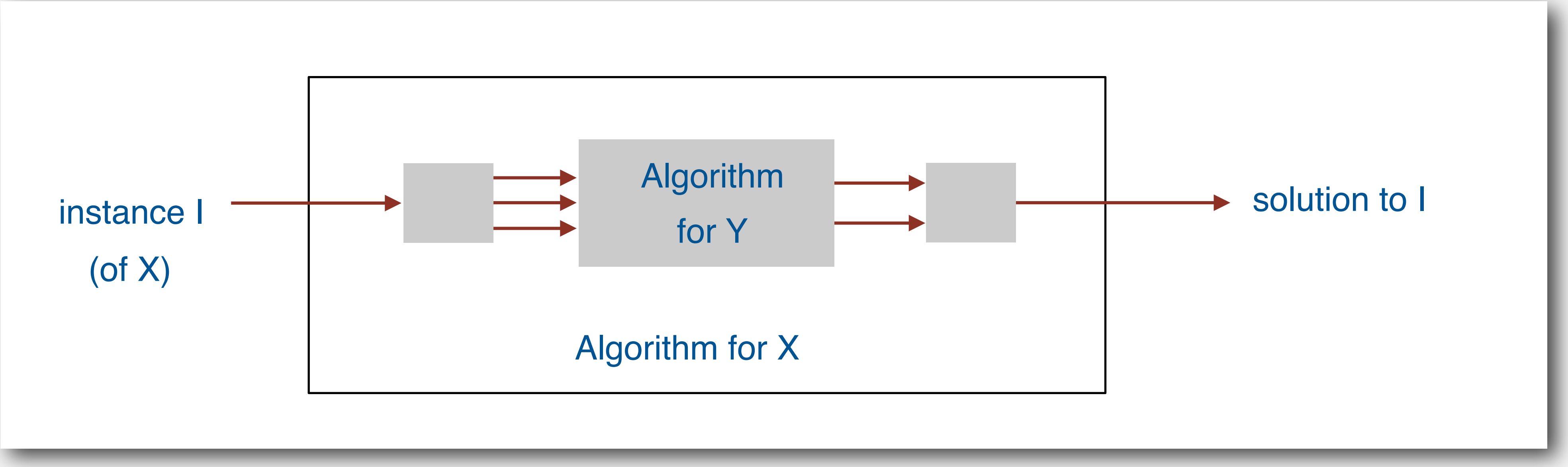
cost of sorting

cost of reduction

**Cost of solving element distinctness.** $N \log N + N$.

# Reduction

Def. Problem $X$ reduces to problem $Y$ if you can use an algorithm that solves $Y$ to help solve $X$.




Ex 2. [3-collinear reduces to sorting]

To solve 3-collinear instance on $N$ points in the plane:

- For each point, sort other points by polar angle.
  - check adjacent triples for collinearity

Cost of solving 3-collinear. $\underbrace{N^2 \log N}_{\text{cost of sorting}} + \underbrace{N^2}_{\text{cost of reduction}}$.

- designing algorithms
- establishing lower bounds
- intractability

# Reduction: design algorithms

**Def.** Problem $X$ **reduces to** problem $Y$ if you can use an algorithm that solves $Y$ to help solve $X$.

**Design algorithm.** Given algorithm for $Y$, can also solve $X$.

**Ex.**

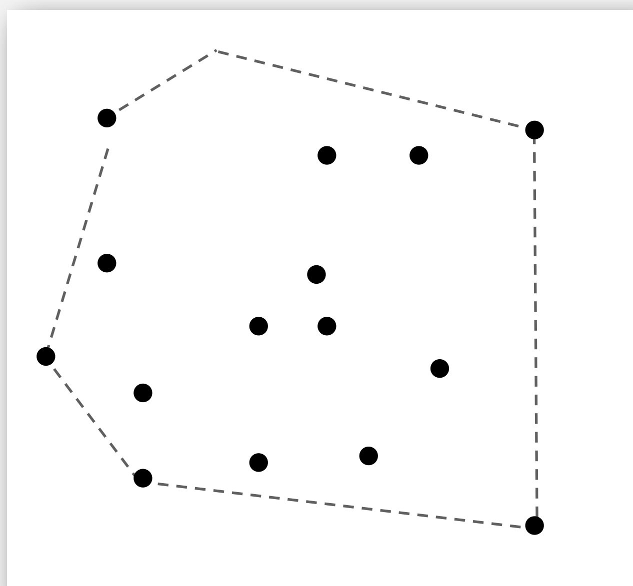
- Element distinctness reduces to sorting.
- 3-collinear reduces to sorting.
- PERT reduces to topological sort. [see digraph lecture]
- h-v line intersection reduces to 1d range searching. [see geometry lecture]
- Burrows-Wheeler transform reduces to suffix sort. [see assignment 8]

**Mentality.** Since I know how to solve $Y$, can I use that algorithm to solve $X$?

programmer's version: I have code for Y. Can I use it for X?

Sorting. Given $N$ distinct integers, rearrange them in ascending order.

Convex hull. Given $N$ points in the plane, identify the extreme points of the convex hull (in counter-clockwise order).



convex hull

```
1251432
2861534
3988818
4190745
13546464
89885444
43434213
34435312
```

sorting

Proposition. Convex hull reduces to sorting.

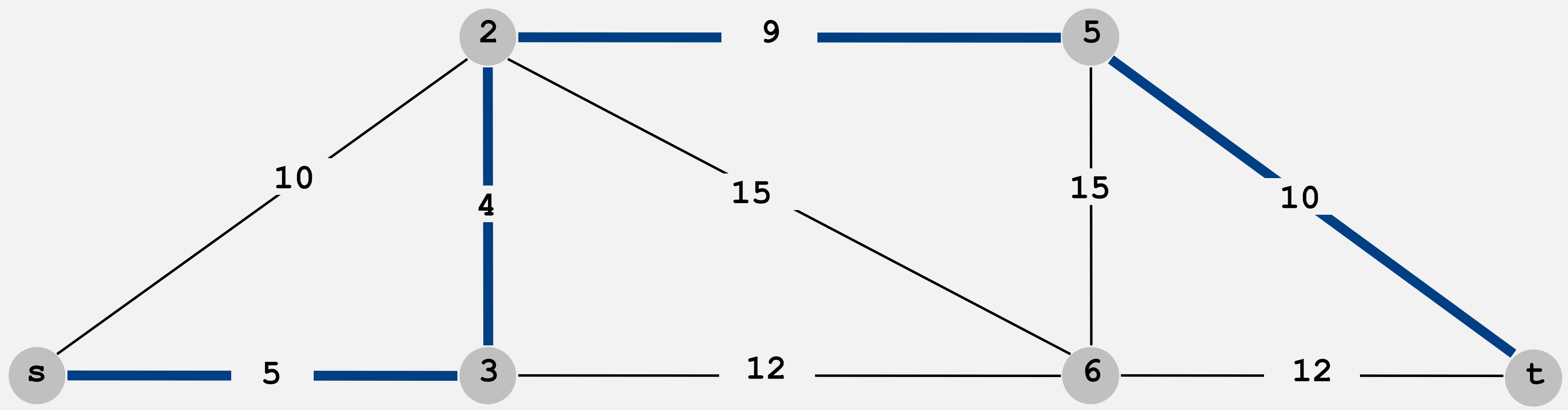
Pf. Graham scan algorithm.

cost of sorting

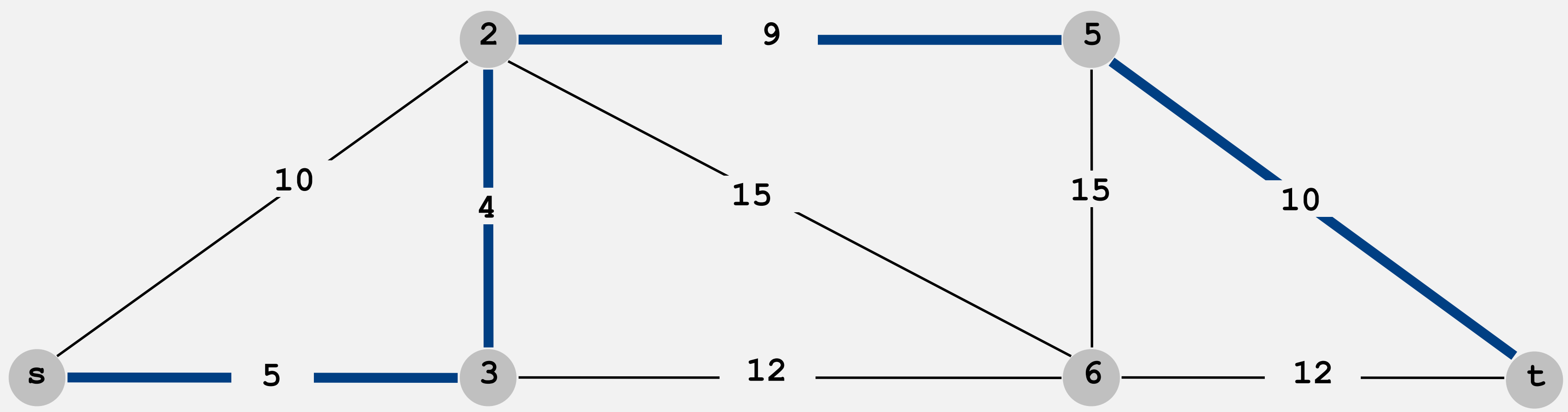
cost of reduction

Cost of convex hull. $N \log \mathrm{N} \; + \; N.$

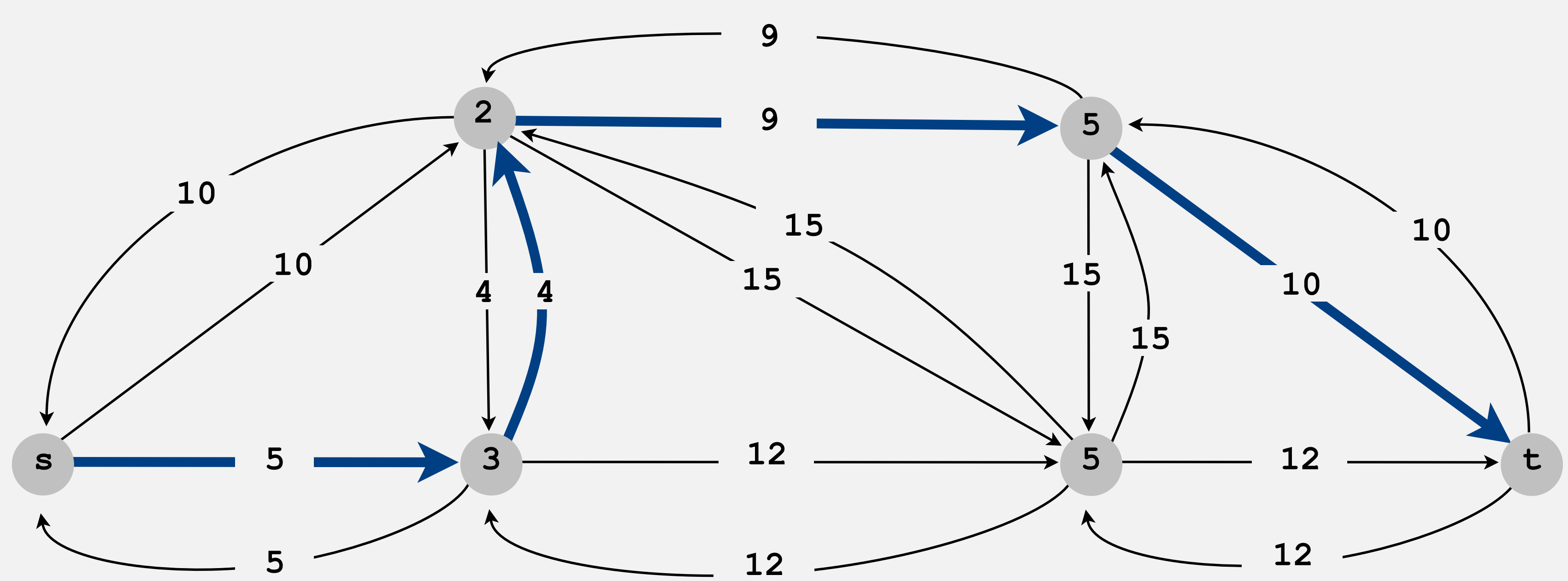
Proposition. Undirected shortest path (with nonnegative weights) reduces to directed shortest path.



Proposition. Undirected shortest path (with nonnegative weights) reduces to directed shortest path.

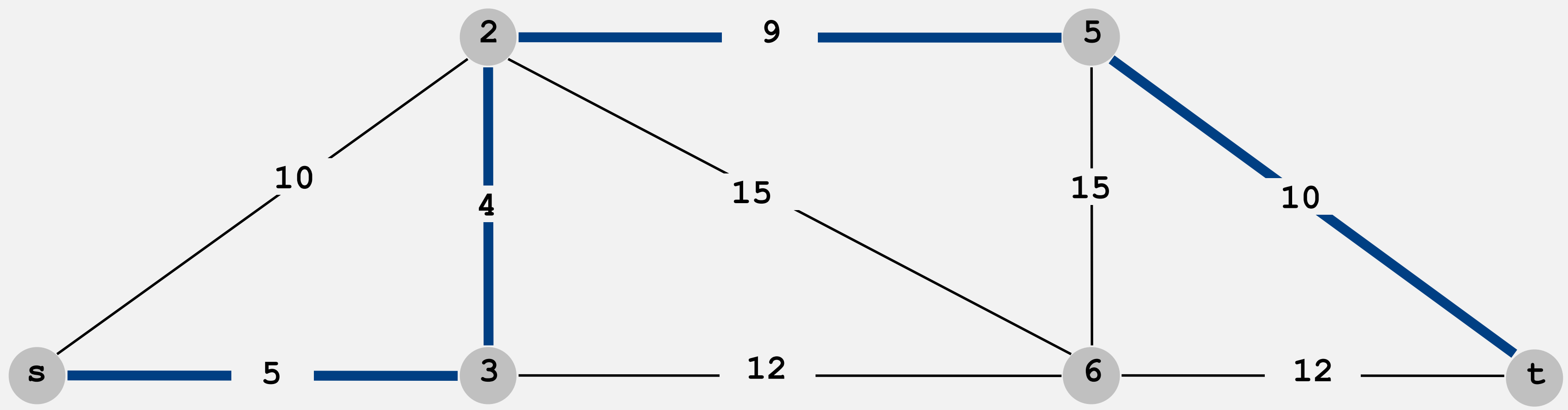



Pf. Replace each undirected edge by two directed edges.



# Shortest path on graphs and digraphs

**Proposition.** Undirected shortest path (with nonnegative weights) reduces to directed shortest path.



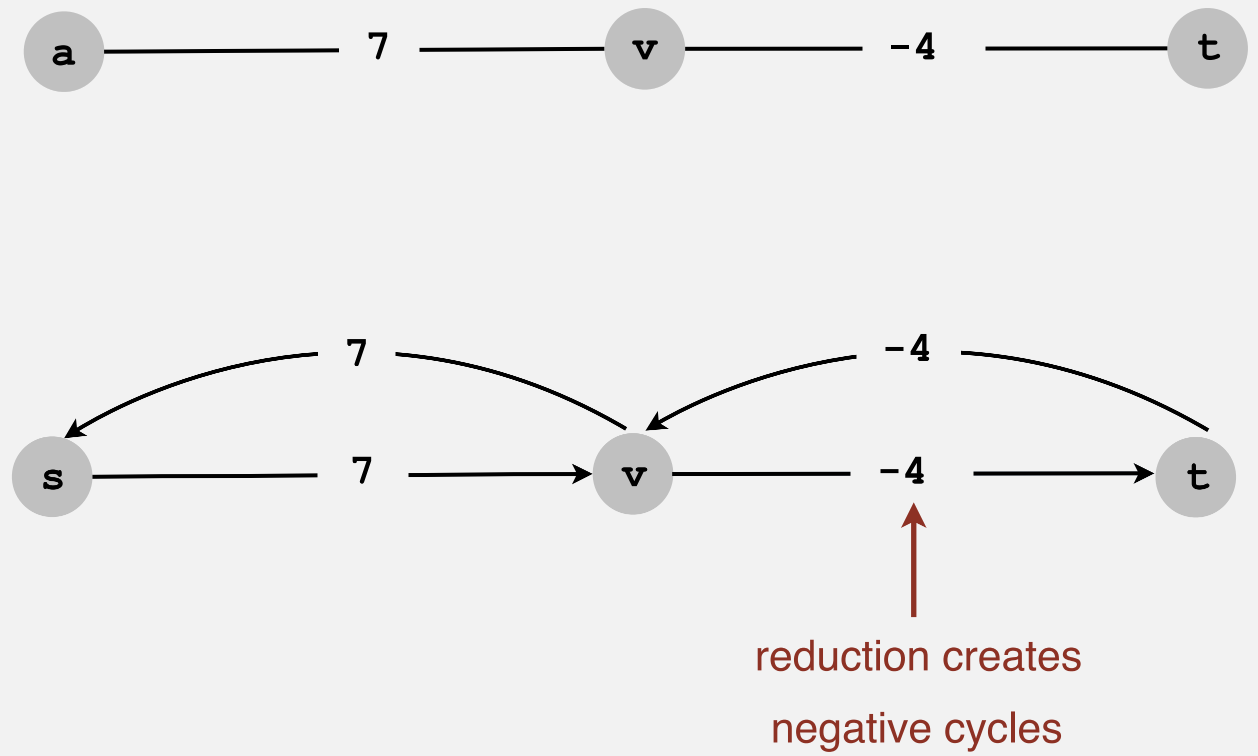

**Cost of undirected shortest path.** $E \log E \; + \; E.$

cost of shortest path in digraph

cost of reduction

# Shortest path with negative weights

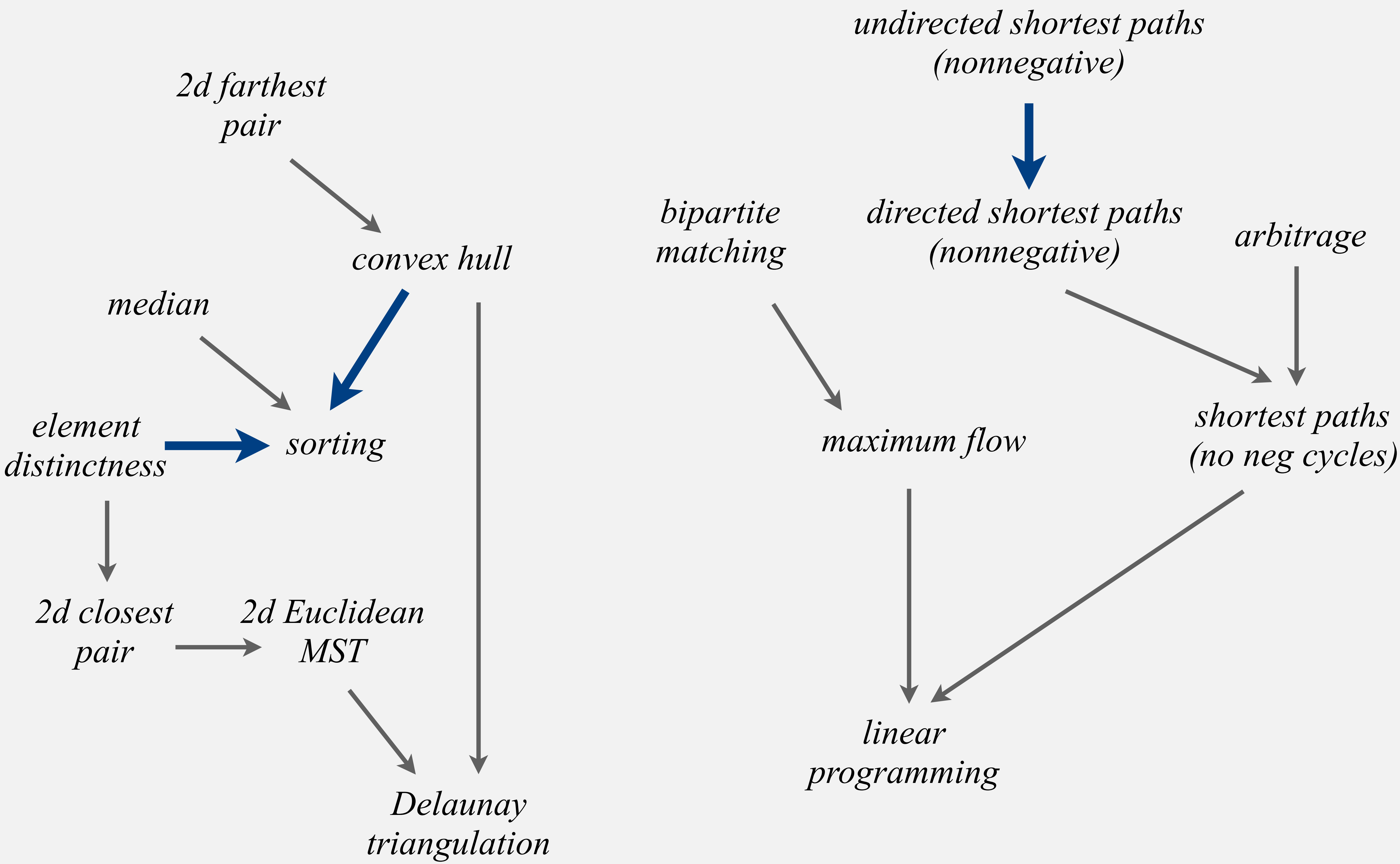
**Caveat.** Reduction is invalid in networks with negative weights (even if no negative cycles).




**Remark.** Can still solve shortest path problem in undirected graphs (if no negative cycles), but need more sophisticated techniques.

reduces to weighted non-bipartite matching (!)

# Some reductions involving familiar problems



- designing algorithms
- **establishing lower bounds**
- intractability

# Bird's-eye view

**Goal.** Prove that a problem requires a certain number of steps.

**Ex.** $\Omega(N \log N)$ lower bound for sorting.

```
1251432
2861534
3988818
4190745
13546464
89885444
43434213
```

argument must apply to all conceivable algorithms

**Bad news.** Very difficult to establish lower bounds from scratch.

**Good news.** Can spread $\Omega(N \log N)$ lower bound to $Y$ by reducing sorting to $Y$.

assuming cost of reduction is not too high

**Def.** Problem $X$ linear-time reduces to problem $Y$ if $X$ can be solved with:

- Linear number of standard computational steps.
- Constant number of calls to $Y$.

**Ex.** Almost all of the reductions we've seen so far. [Which one wasn't?]

**Establish lower bound:**

- If $X$ takes $\Omega(N \log N)$ steps, then so does $Y$.
- If $X$ takes $\Omega(N^2)$ steps, then so does $Y$.

**Mentality.**

- If I could easily solve $Y$, then I could easily solve $X$.
- I can't easily solve $X$.
- Therefore, I can't easily solve $Y$.

# Lower bound for convex hull

**Proposition.** In quadratic decision tree model, any algorithm for sorting $N$ integers requires $\Omega(N \log N)$ steps.

allows quadratic tests of the form:

$x_i < x_j$ or $(x_j - x_i)\ (x_k - x_i) - (x_j\ )\ (x_j - x_i) < 0$

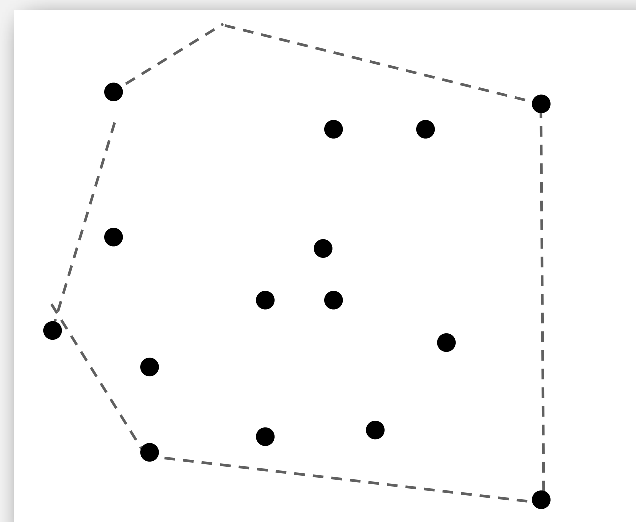
**Proposition.** Sorting linear-time reduces to convex hull.

**Pf.** [see next slide]

lower-bound mentality:
if I can solve convex hull
efficiently, I can sort efficiently

```
1251432
2861534
3988818
4190745
13546464
89885444
43434213
```

*sorting*



*convex hull*

a quadratic test

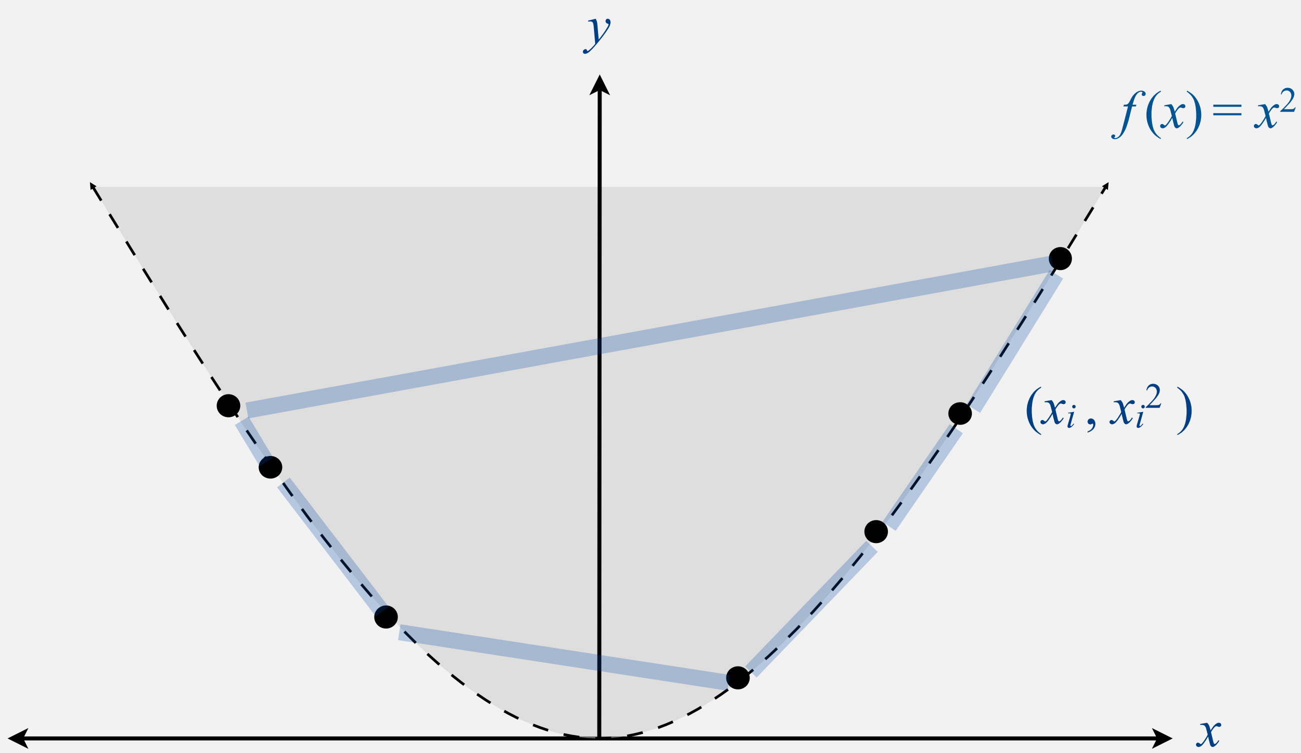
**Implication.** Any ccw-based convex hull algorithm requires $\Omega(N \log N)$ ccw's.

# Sorting linear-time reduces to convex hull

**Proposition.** Sorting linear-time reduces to convex hull.

- Sorting instance: $x_1, x_2, \ldots, x_N$.
- Convex hull instance: $(x_1, x_1^2), (x_2, x_2^2), \ldots, (x_N, x_N^2)$.

lower-bound mentality:
if I can solve convex hull
efficiently, I can sort efficiently




**Pf.**

- Region $\{ x : x^2 \geq x \}$ is convex $\Rightarrow$ all points are on hull.
- Starting at point with most negative $x$, counterclockwise order of hull points yields integers in ascending order.

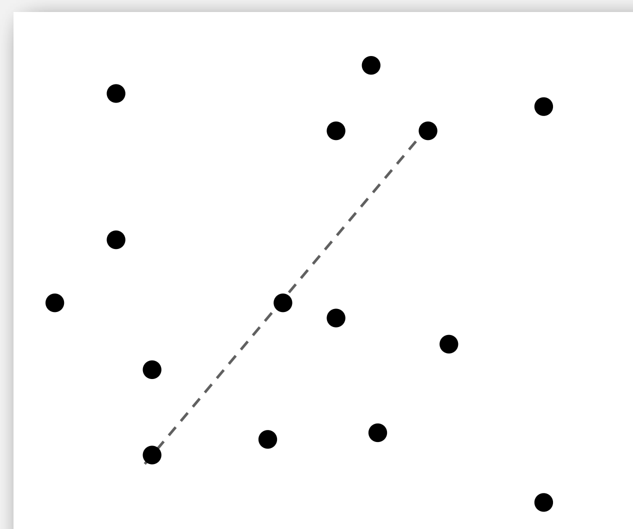
# Lower bound for 3-COLLINEAR

3-SUM. Given $N$ distinct integers, are there three that sum to 0?

3-COLLINEAR. Given $N$ distinct points in the plane, are there 3 that all lie on the same line? ← recall Assignment 3

```
1251432
-2861534
3988818
-4190745
13546464
89885444
-43434213
```

**3-sum**



**3-collinear**

# Lower bound for 3-COLLINEAR

**3-SUM.** Given $N$ distinct integers, are there three that sum to 0?

**3-COLLINEAR.** Given $N$ distinct points in the plane, are there 3 that all lie on the same line?

**Proposition.** 3-SUM linear-time reduces to 3-COLLINEAR.

**Pf.** [see next 2 slide]

**Conjecture.** Any algorithm for 3-SUM requires $\Omega(N^2)$ steps.

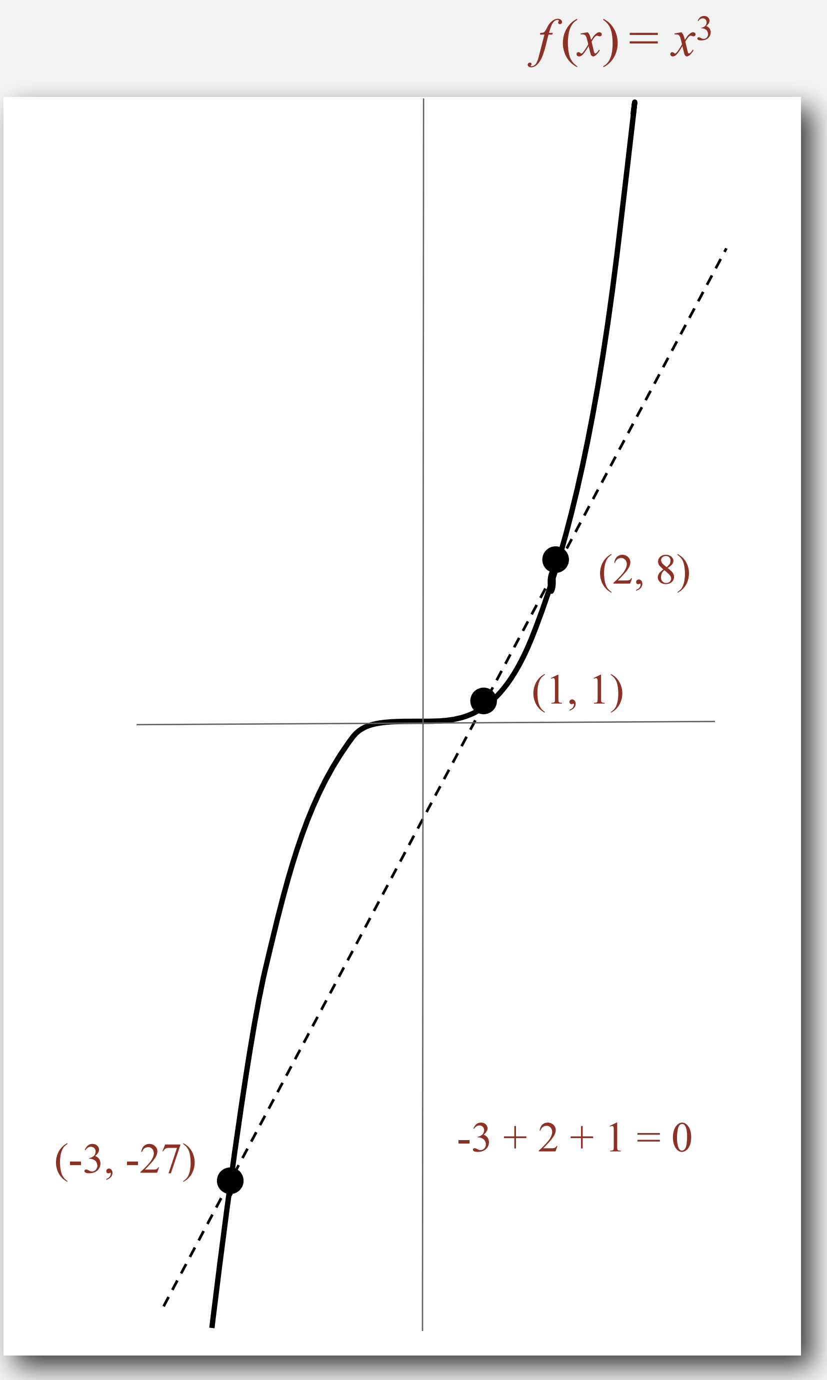
**Implication.** No sub-quadratic algorithm for 3-COLLINEAR likely.

your $N^2 \log N$ algorithm was pretty good

**Proposition.** 3-SUM linear-time reduces to 3-COLLINEAR.

- 3-SUM instance: $x_1, x_2, \ldots, x_N$.
- 3-COLLINEAR instance: $(x_1, x_1^3), (x_2, x_2^3), \ldots, (x_N, x_N^3)$.

**Lemma.** If $a$, $b$, and $c$ are distinct, then $a + b + c = 0$ if and only if $(a, a^3)$, $(b, b^3)$, and $(c, c^3)$ are collinear.



**Proposition.** 3-SUM linear-time reduces to 3-COLLINEAR.

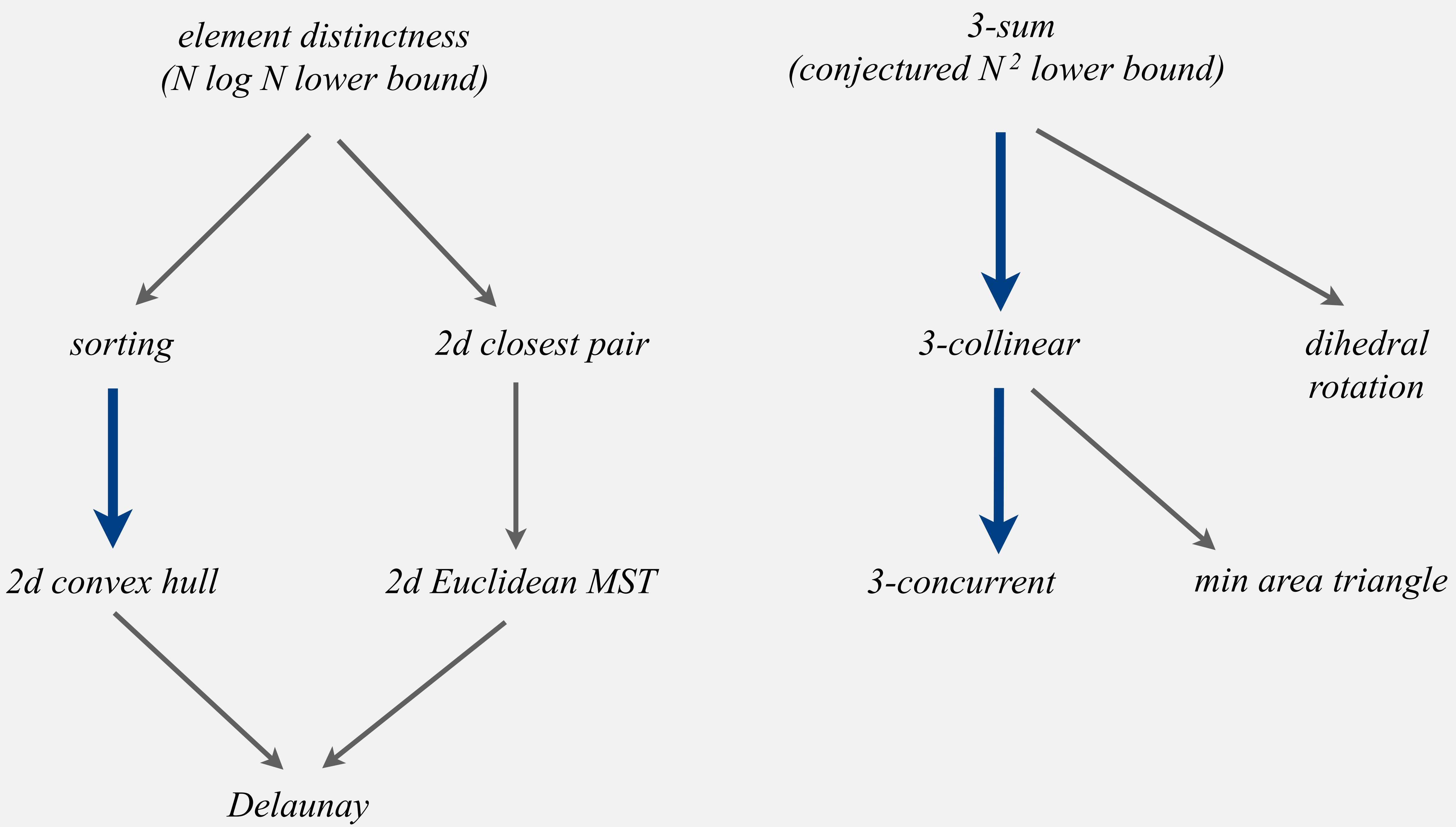
- 3-SUM instance: $x_1, x_2, \ldots, x_N$.
- 3-COLLINEAR instance: $(x_1, x_1^3), (x_2, x_2^3), \ldots, (x_N, x_N^3)$.

**Lemma.** If $a$, $b$, and $c$ are distinct, then $a + b + c = 0$ if and only if $(a, a^3)$, $(b, b^3)$, and $(c, c^3)$ are collinear.

**Pf.** Three distinct points $(a, a^3)$, $(b, b^3)$, and $(c, c^3)$ are collinear iff:

$$
\begin{aligned}
0 &= \begin{vmatrix} a & a^3 & 1 \\ b & b^3 & 1 \\ c & c^3 & 1 \end{vmatrix} \\
&= a(b^3 - c^3) - b(a^3 - c^3) + c(a^3 - b^3) \\
&= (a - b)(b - c)(c - a)(a + b + c)
\end{aligned}
$$

# More linear-time reductions and lower bounds



# Establishing lower bounds: summary

Establishing lower bounds through reduction is an important tool in guiding algorithm design efforts.

Q. How to convince yourself no linear-time convex hull algorithm exists?
A1. [hard way] Long futile search for a linear-time algorithm.
A2. [easy way] Linear-time reduction from sorting.

Q. How to convince yourself no sub-quadratic 3-COLLINEAR algorithm exists.
A1. [hard way] Long futile search for a sub-quadratic algorithm.
A2. [easy way] Linear-time reduction from 3-SUM.

- designing algorithms
- establishing lower bounds
- **intractability**

**Def.** A problem is intractable if it can't be solved in polynomial time.

**Desiderata.** Prove that a problem is intractable.

**Two problems that provably require exponential time.**

input size = c + lg K

- Given a constant-size program, does it halt in at most $K$ steps?
- Given $N$-by-$N$ checkers board position, can the first player force a win?

using forced capture rule

**Frustrating news.** Few successes.

# 3-satisfiability

**Literal.** A boolean variable or its negation. $x_i$ or $\neg x_i$

**Clause.** An *or* of 3 distinct literals. $C_1 = (\neg x_1 \vee x_2 \vee x_3)$

**Conjunctive normal form.** An *and* of clauses. $\Phi = (C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5)$

**3-SAT.** Given a CNF formula $\Phi$ consisting of $k$ clauses over n literals, does it have a satisfying truth assignment?

$$\Phi \;=\; (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_4) \wedge (\neg x_2 \vee x_3 \vee x_4)$$

**Applications.** Circuit design, program correctness, ...

# 3-satisfiability

**Literal.** A boolean variable or its negation. $x_i$ or $\neg x_i$

**Clause.** An *or* of 3 distinct literals. $C_1 = (\neg x_1 \vee x_2 \vee x_3)$

**Conjunctive normal form.** An *and* of clauses. $\Phi = (C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5)$

**3-SAT.** Given a CNF formula $\Phi$ consisting of $k$ clauses over n literals, does it have a satisfying truth assignment?

$$\Phi \;=\; (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_4) \wedge (\neg x_2 \vee x_3 \vee x_4)$$

yes instance

| $x_1$ | $x_2$ | $x_3$ | $x_4$ |
|---|---|---|---|
| T | T | F | T |

$$(\neg T \vee T \vee F) \wedge (T \vee \neg T \vee F) \wedge (\neg T \vee \neg T \vee \neg F) \wedge (\neg T \vee \neg T \vee T) \wedge (\neg T \vee F \vee T)$$

**Applications.** Circuit design, program correctness, ...

# 3-satisfiability is believed intractable

Q. How to solve an instance of 3-SAT with $n$ variables?

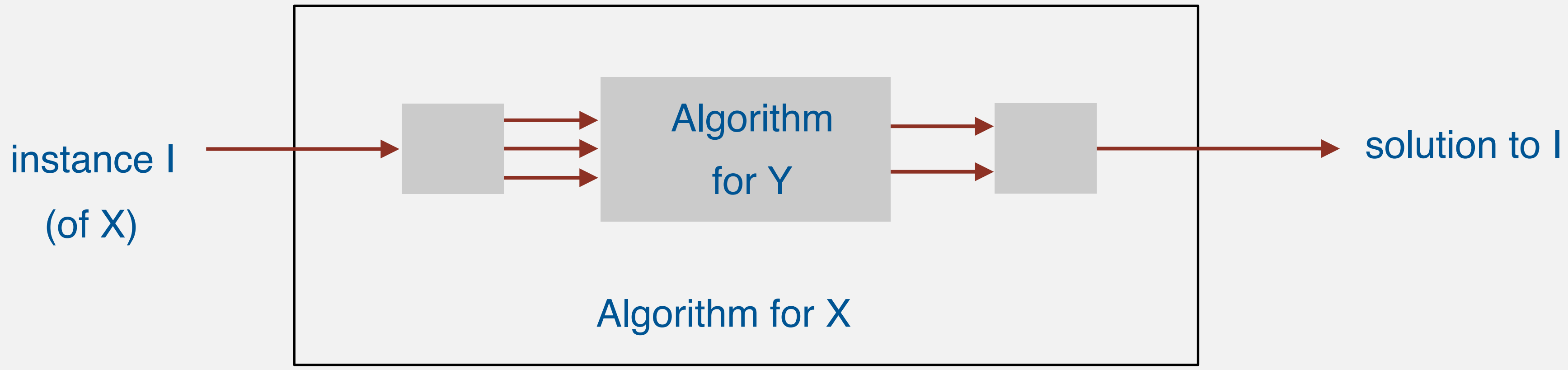
A. Exhaustive search: try all $2^n$ truth assignments.

Q. Can we do anything substantially more clever?


Conjecture (P $\neq$ NP). 3-SAT is intractable (no poly-time algorithm).

**Def.** Problem $X$ poly-time (Cook) reduces to problem $Y$ if $X$ can be solved with:

- Polynomial number of standard computational steps.
- Polynomial number of calls to $Y$.




**Establish intractability.** If *3-SAT* poly-time reduces to $Y$, then $Y$ is intractable.
(assuming *3-SAT* is intractable)

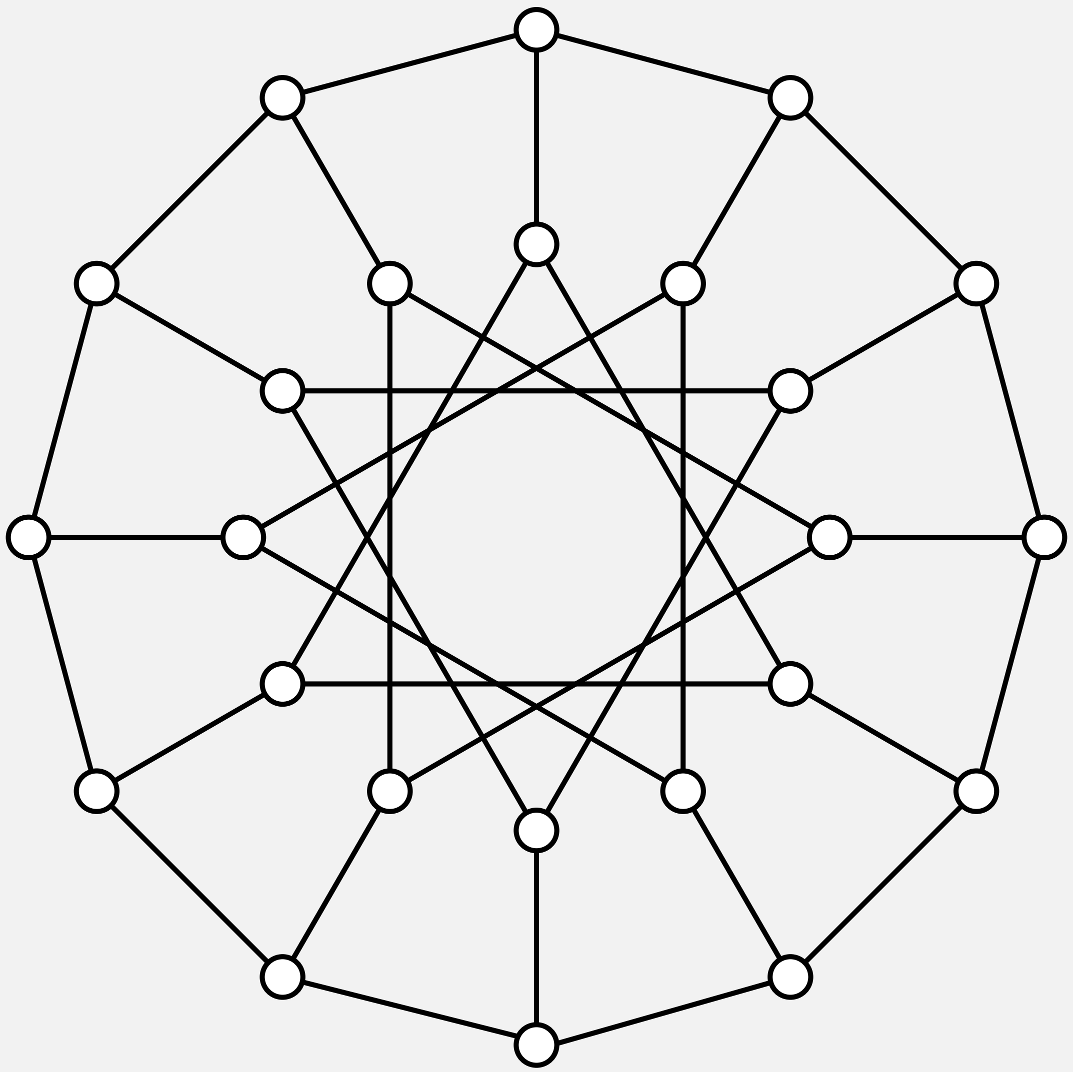
**Mentality.**

- If I could solve $Y$ in poly-time, then I could also solve *3-SAT* in poly-time.
- *3-SAT* is believed to be intractable.
- Therefore, so is $Y$.

Def. An independent set is a set of vertices, no two of which are adjacent.

*IND-SET*. Given a graph $G$ and an integer $k$, find an independent set of size $k$.

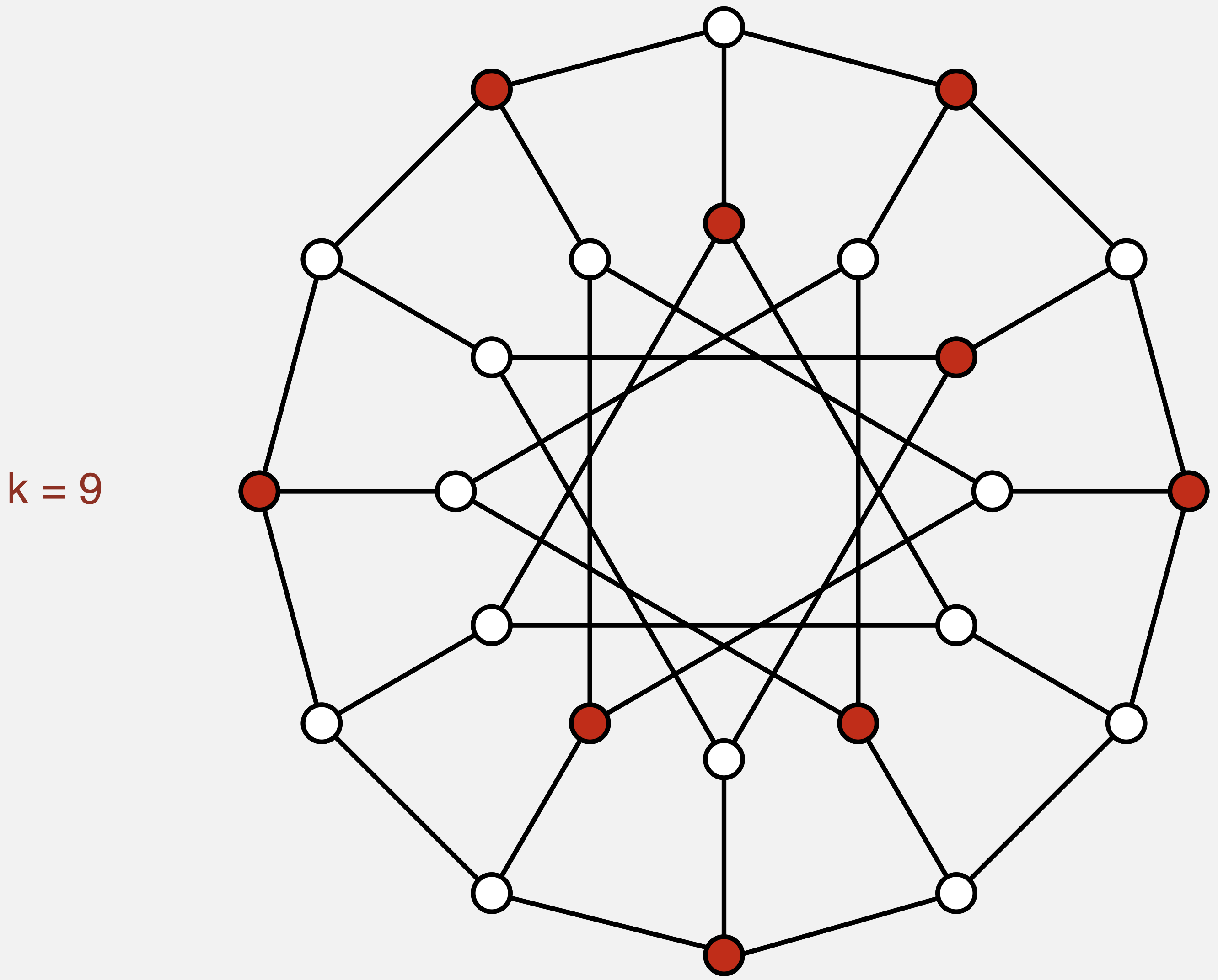
k = 9



Applications. Scheduling, computer vision, clustering, …

**Def.** An independent set is a set of vertices, no two of which are adjacent.

*IND-SET.* Given a graph $G$ and an integer $k$, find an independent set of size $k$.



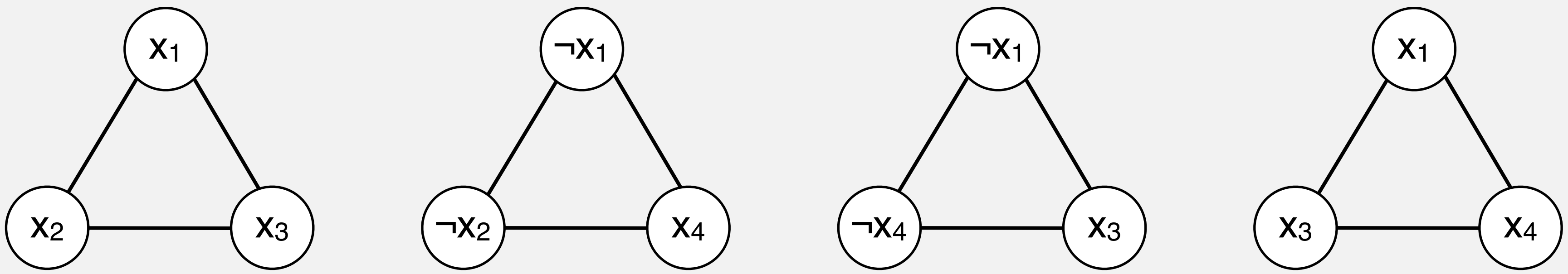

**Applications.** Scheduling, computer vision, clustering, …

Proposition. *3-SAT* poly-time reduces to *IND-SET*.

Pf. Given an instance $\Phi$ of *3-SAT*, create an instance $G$ of *IND-SET*:

- For each clause in $\Phi$, create 3 vertices in a triangle.
- Add an edge between each literal and its negation.

k = 4



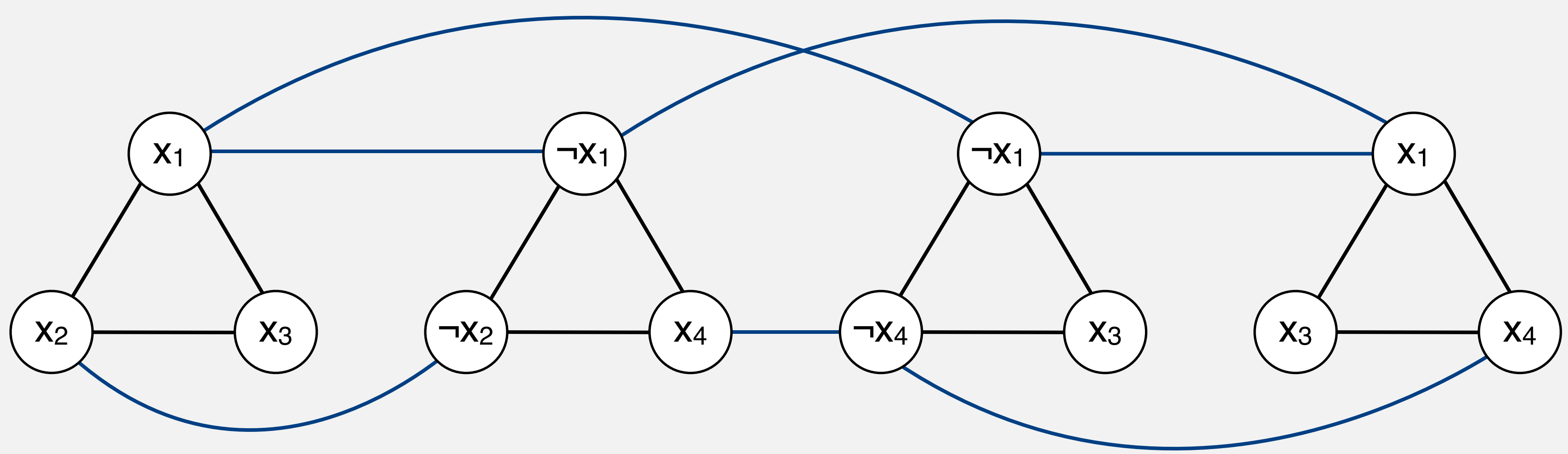

$$\Phi \;=\; (x_1 \vee x_2 \vee x_3) \;\wedge\; (\neg x_1 \vee \neg x_2 \vee x_4) \;\wedge\; (\neg x_1 \vee x_3 \vee \neg x_4) \;\wedge\; (x_1 \vee x_3 \vee x_4)$$

**Proposition.** *3-SAT* poly-time reduces to *IND-SET*.

**Pf.** Given an instance $\Phi$ of *3-SAT*, create an instance $G$ of *IND-SET*:

- For each clause in $\Phi$, create 3 vertices in a triangle.
- Add an edge between each literal and its negation.

k = 4




$$\Phi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_4) \wedge (\neg x_1 \vee x_3 \vee \neg x_4) \wedge (x_1 \vee x_3 \vee x_4)$$

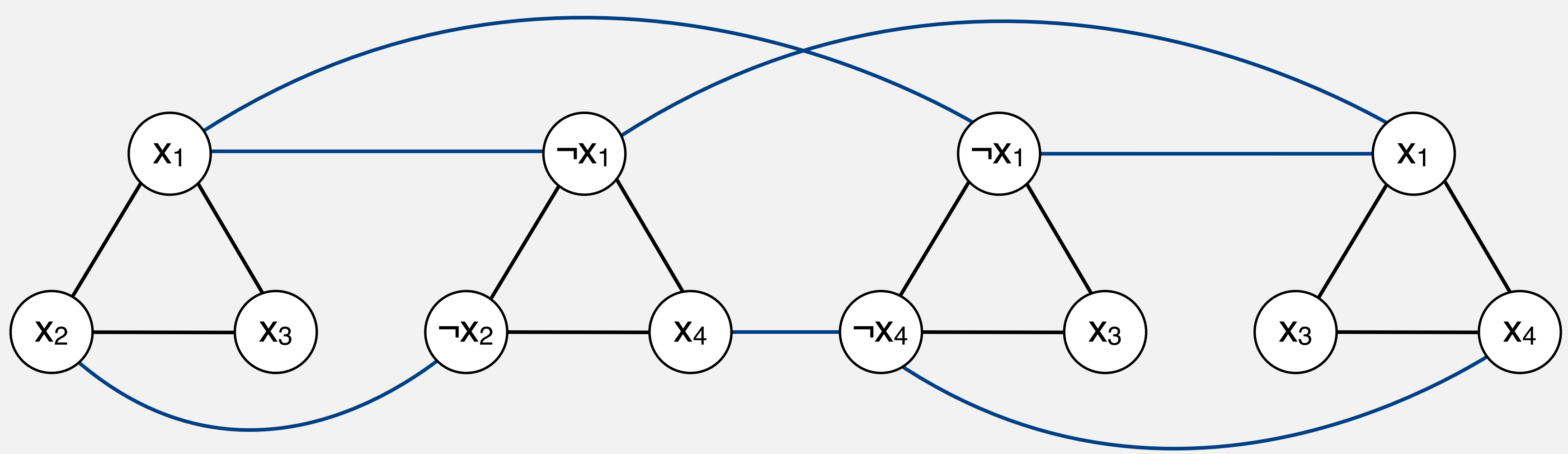
# 3-satisfiability reduces to independent set

Proposition. *3-SAT* poly-time reduces to *IND-SET*.

Pf. Given an instance $\Phi$ of *3-SAT*, create an instance $G$ of *IND-SET*:

- For each clause in $\Phi$, create 3 vertices in a triangle.
- Add an edge between each literal and its negation.

k = 4




$$\Phi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_4) \wedge (\neg x_1 \vee x_3 \vee \neg x_4) \wedge (x_1 \vee x_3 \vee x_4)$$

- $G$ has independent set of size $k \Rightarrow \Phi$ satisfiable.

set literals corresponding to vertices in independent to true;
set remaining literals in consistent manner

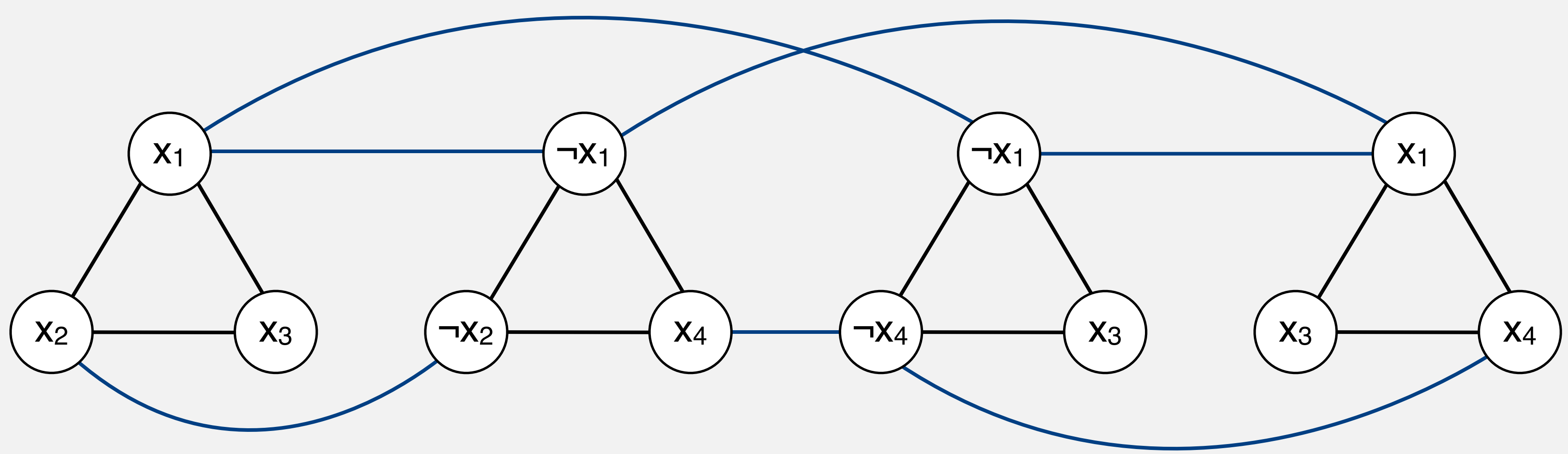
# 3-satisfiability reduces to independent set

Proposition. *3-SAT* poly-time reduces to *IND-SET*.

Pf. Given an instance $\Phi$ of *3-SAT*, create an instance $G$ of *IND-SET*:

- For each clause in $\Phi$, create 3 vertices in a triangle.
- Add an edge between each literal and its negation.

k = 4




$$\Phi \ = \ (x_1 \vee x_2 \vee x_3) \ \wedge \ (\neg x_1 \vee \neg x_2 \vee x_4) \ \wedge \ (\neg x_1 \vee x_3 \vee \neg x_4) \ \wedge \ (x_1 \vee x_3 \vee x_4)$$

- $G$ has independent set of size $k \Rightarrow \Phi$ satisfiable.
- $\Phi$ satisfiable $\Rightarrow$ $G$ has independent set of size $k$.

for each clause, take vertex corresponding to one true literal

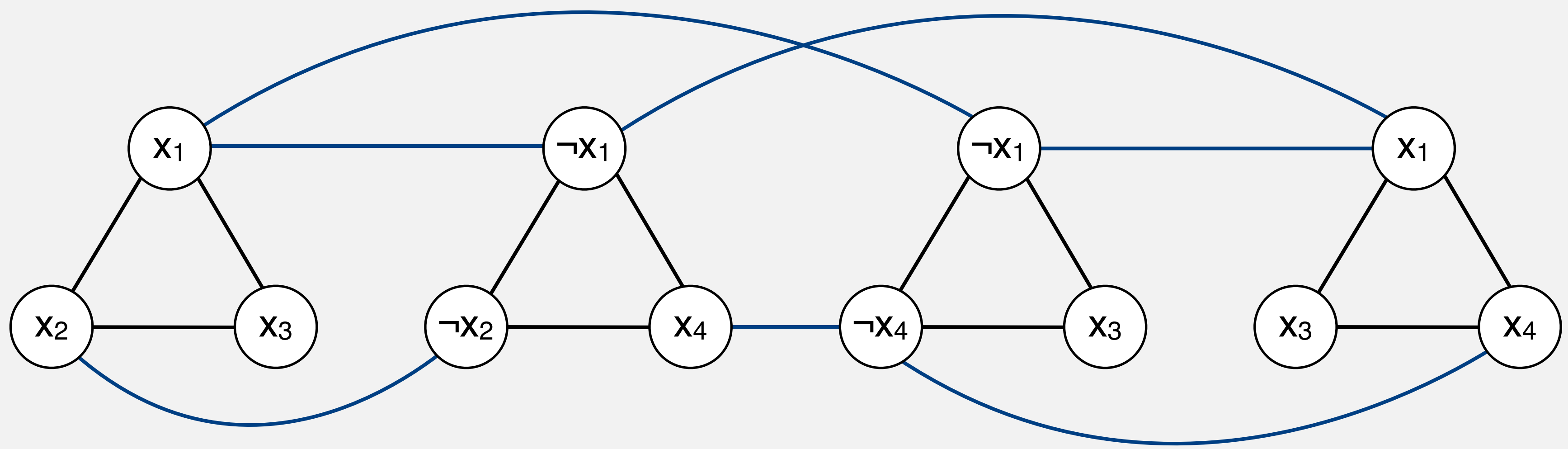
# 3-satisfiability reduces to independent set

**Proposition.** *3-SAT* poly-time reduces to *IND-SET*.

lower-bound mentality:
if I could solve IND-SET efficiently,
I could solve 3-SAT efficiently

**Implication.** Assuming *3-SAT* is intractable, so is *IND-SET*.

k = 4




$$\Phi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_4) \wedge (\neg x_1 \vee x_3 \vee \neg x_4) \wedge (x_1 \vee x_3 \vee x_4)$$

# Integer linear programming

ILP. Given a system of linear inequalities, find an integral solution.

| |
|---|
| $3x_1 + 5x_2 + 2x_3 + x_4 + 4x_5 \ \geq\ 10$ |
| $5x_1 + 2x_2 + 4x_4 + 1x_5 \ \leq\ 7$ |
| $x_1 + x_3 + 2x_4 \ \leq\ 2$ |
| $3x_1 + 4x_3 + 7x_4 \ \leq\ 7$ |
| $x_1 + x_4 \ \leq\ 1$ |
| $x_1 + x_3 + x_5 \ \leq\ 1$ |
| all $x_i \ = \ \{ 0, 1 \}$ |

linear inequalities

integer variables

Context. Cornerstone problem in operations research.

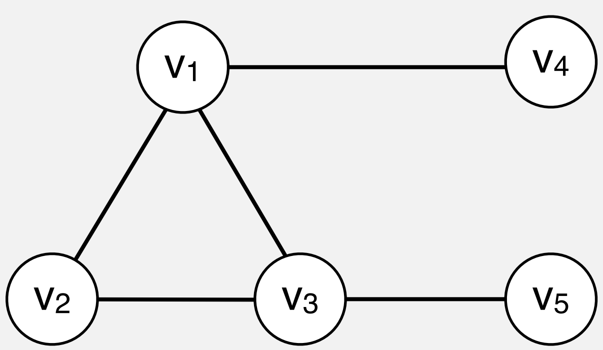
Remark. Finding a real-valued solution is tractable (linear programming).

# Independent set reduces to integer linear programming

**Proposition.** *IND-SET* poly-time reduces to *ILP*.

**Pf.** Given an instance $G$, $k$ of *IND-SET*, create an instance of *ILP* as follows:

**Intuition.** $x_i = 1$ if and only if vertex $v_i$ is in independent set.




**is there an independent set of size 3 ?**

$$x_1 + x_2 + x_3 + x_4 + x_5 \ = \ 3$$

number of vertices selected

$$x_1 + x_2 \ \leq \ 1$$

$$x_2 + x_3 \ \leq \ 1$$

$$x_1 + x_3 \ \leq \ 1$$

$$x_1 + x_4 \ \leq \ 1$$

$$x_3 + x_5 \ \leq \ 1$$

at most one vertex selected from each edge

$$\text{all } x_i \ = \ \{\ 0, 1\ \}$$

binary variables

**is there a feasible solution?**

# 3-satisfiability reduces to integer linear programming

Proposition. *3-SAT* poly-time reduces to *IND-SET*.

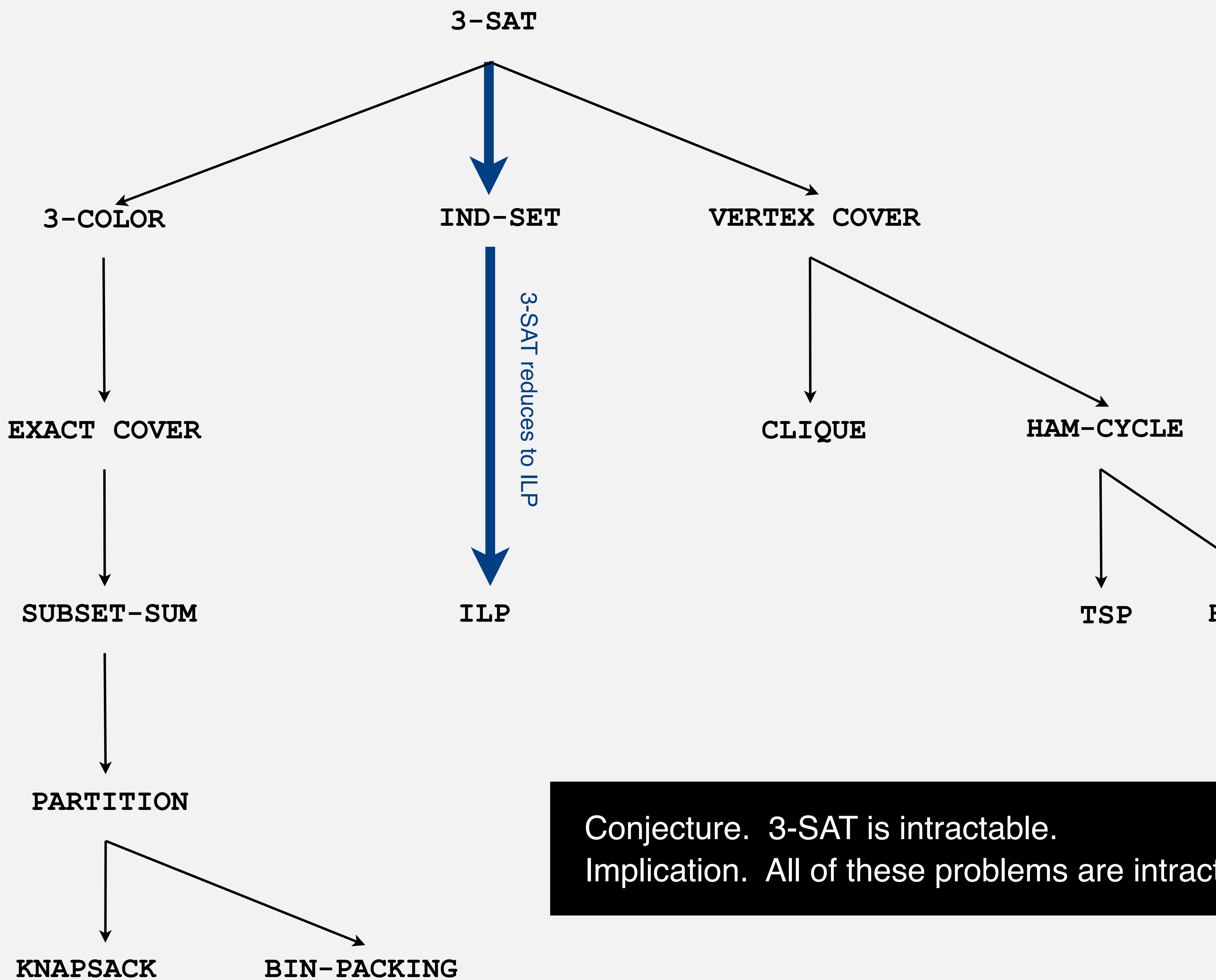
Proposition. *IND-SET* poly-time reduces to *ILP*.

Transitivity. If $X$ poly-time reduces to $Y$ and $Y$ poly-time reduces to $Z$, then $X$ poly-time reduces to $Z$.

Implication. Assuming *3-SAT* is intractable, so is *ILP*.

lower-bound mentality:
if I could solve ILP efficiently,
I could solve 3-SAT efficiently

# More poly-time reductions from 3-satisfiability




Dick Karp

'85 Turing award

Conjecture. 3-SAT is intractable.
Implication. All of these problems are intractable.

# Implications of poly-time reductions from 3-satisfiability

Establishing intractability through poly-time reduction is an important tool in guiding algorithm design efforts.

Q. How to convince yourself that a new problem is (probably) intractable?

A1. [hard way] Long futile search for an efficient algorithm (as for *3-SAT*).

A2. [easy way] Reduction from *3-SAT*.

Caveat. Intricate reductions are common.

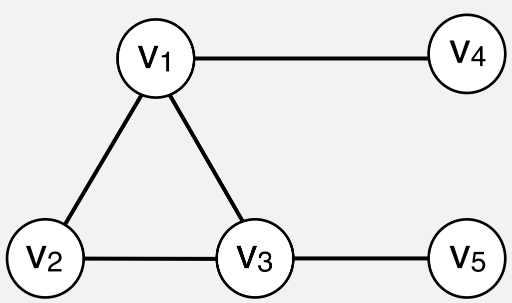
Search problem. Problem where you can check a solution in poly-time.

Ex 1. *3-SAT*.

$$\Phi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_4) \wedge (\neg x_1 \vee x_3 \vee \neg x_4) \wedge (x_1 \vee x_3 \vee x_4)$$

$x_1 = true,\ x_2 = true,\ x_3 = true,\ x_4 = true$

Ex 2. *IND-SET*.




$k = 3$

$\{ v_2 ,\ x_4,\ v_5 \}$

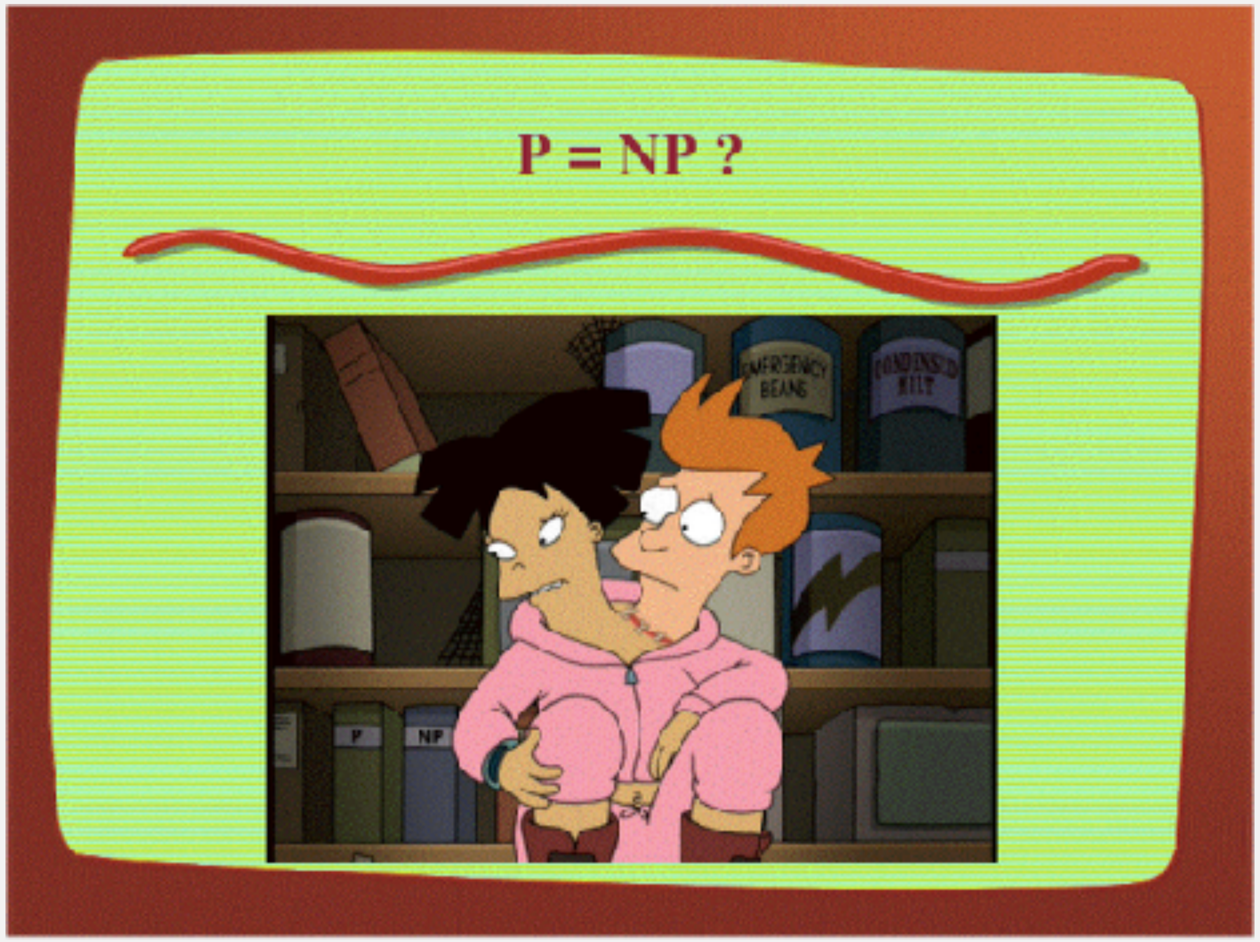
P. Set of search problems solvable in poly-time.

Importance. What scientists and engineers can compute feasibly.

NP. Set of search problems.

Importance. What scientists and engineers aspire to compute feasibly.

Fundamental question.



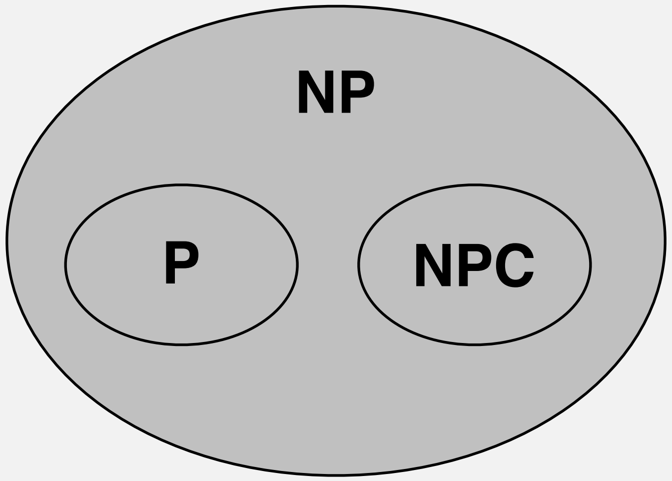

Consensus opinion. No.

**Def.** An NP problem is NP-complete if all problems in NP poly-time to reduce to it.

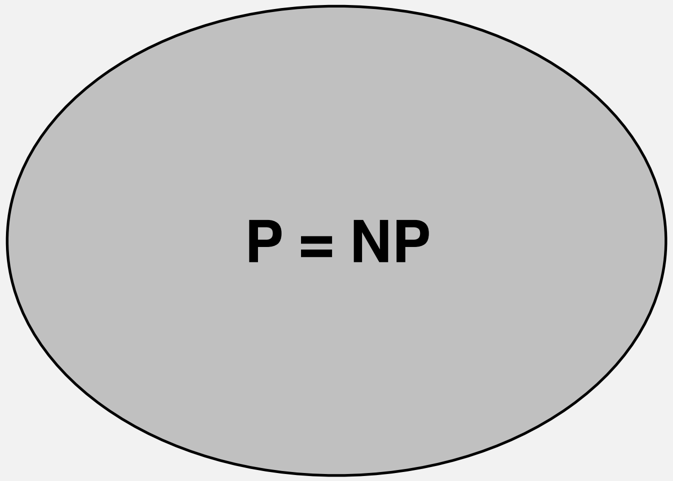
**Cook's theorem.** *3-SAT* is NP-complete.

**Corollary.** *3-SAT* is tractable if and only if P = NP.

**Two worlds.**

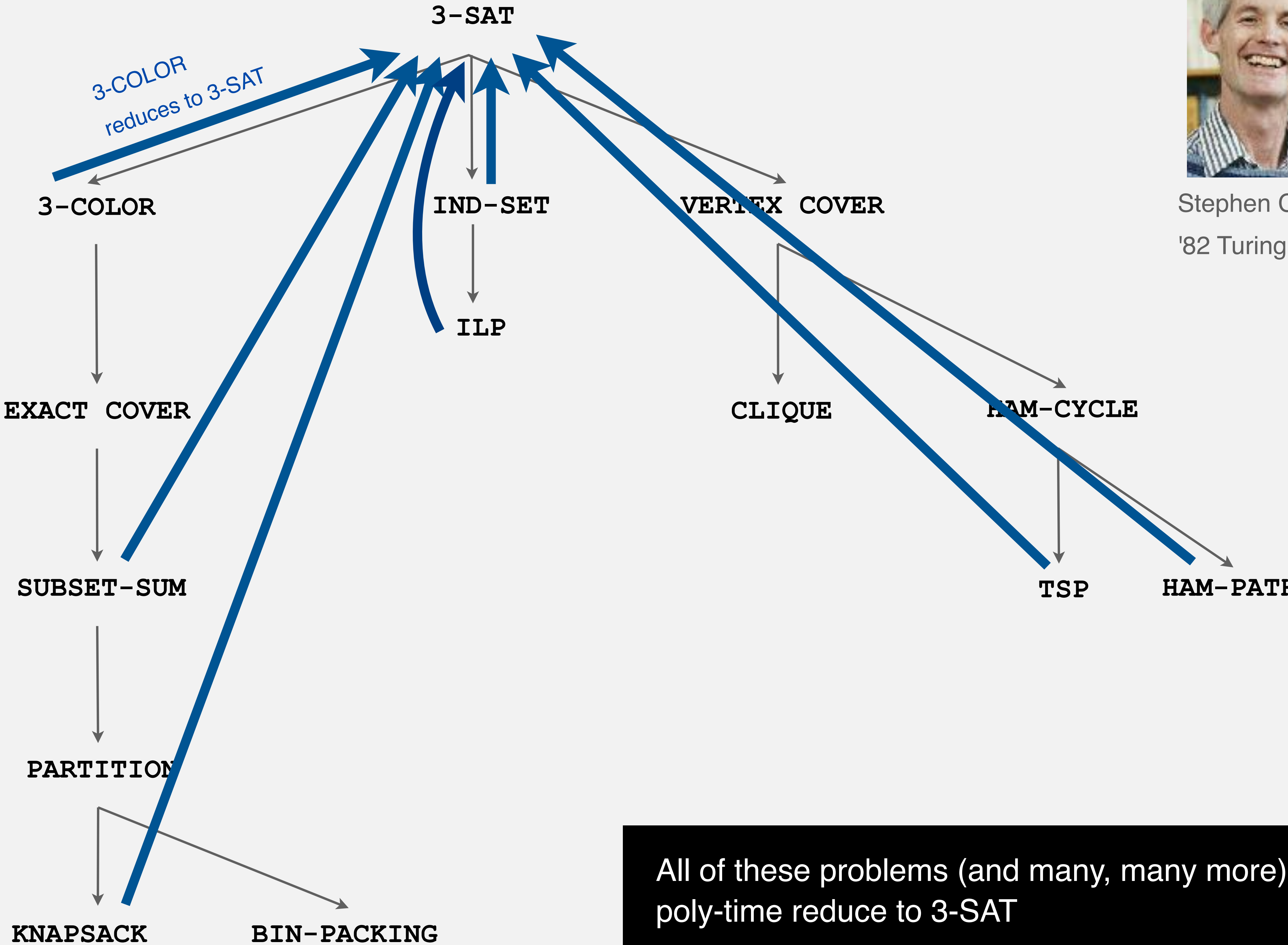



P ≠ NP




P = NP

# Implications of Cook's theorem

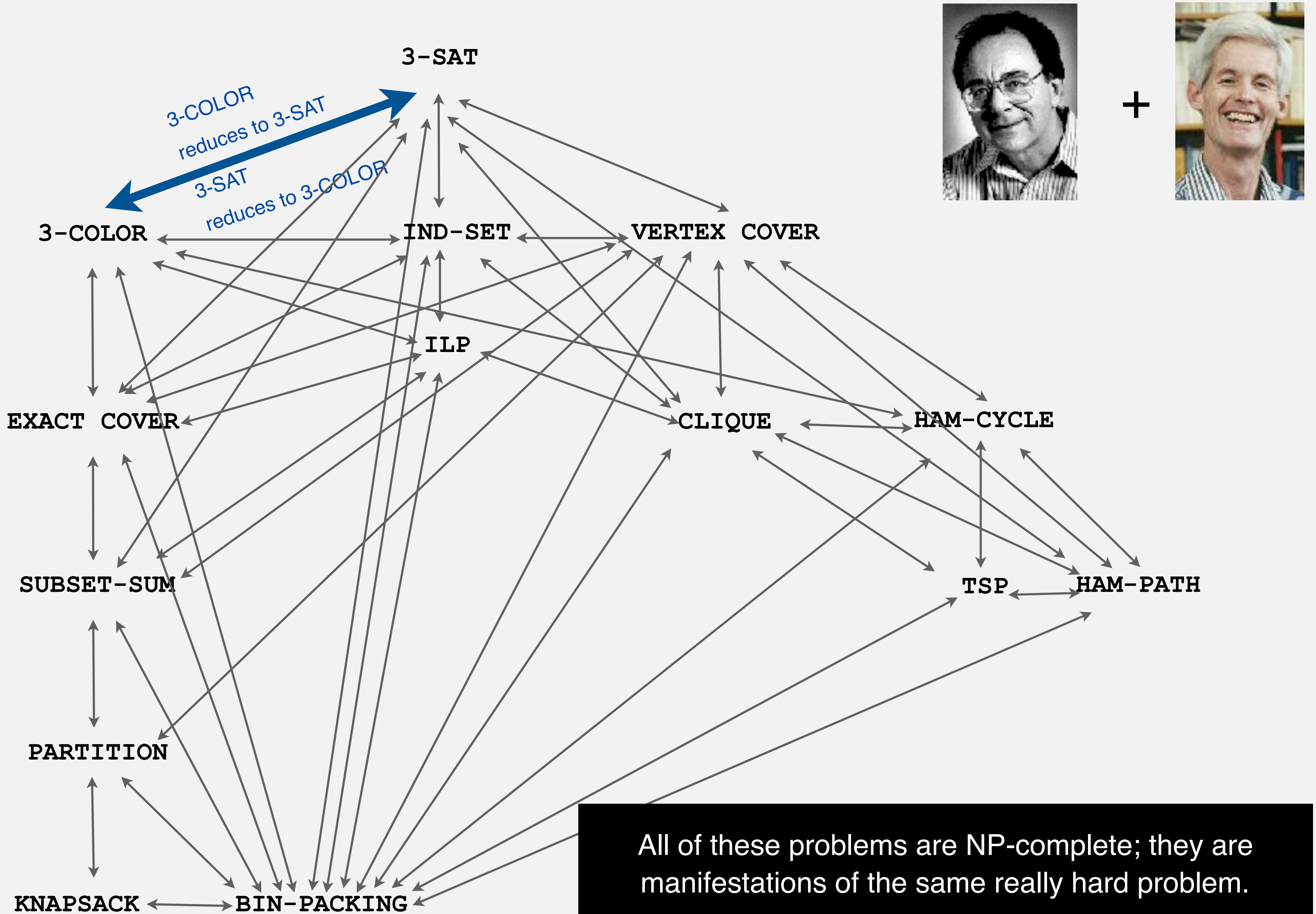



Stephen Cook

'82 Turing award

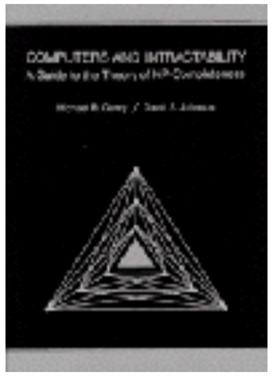
All of these problems (and many, many more) poly-time reduce to 3-SAT

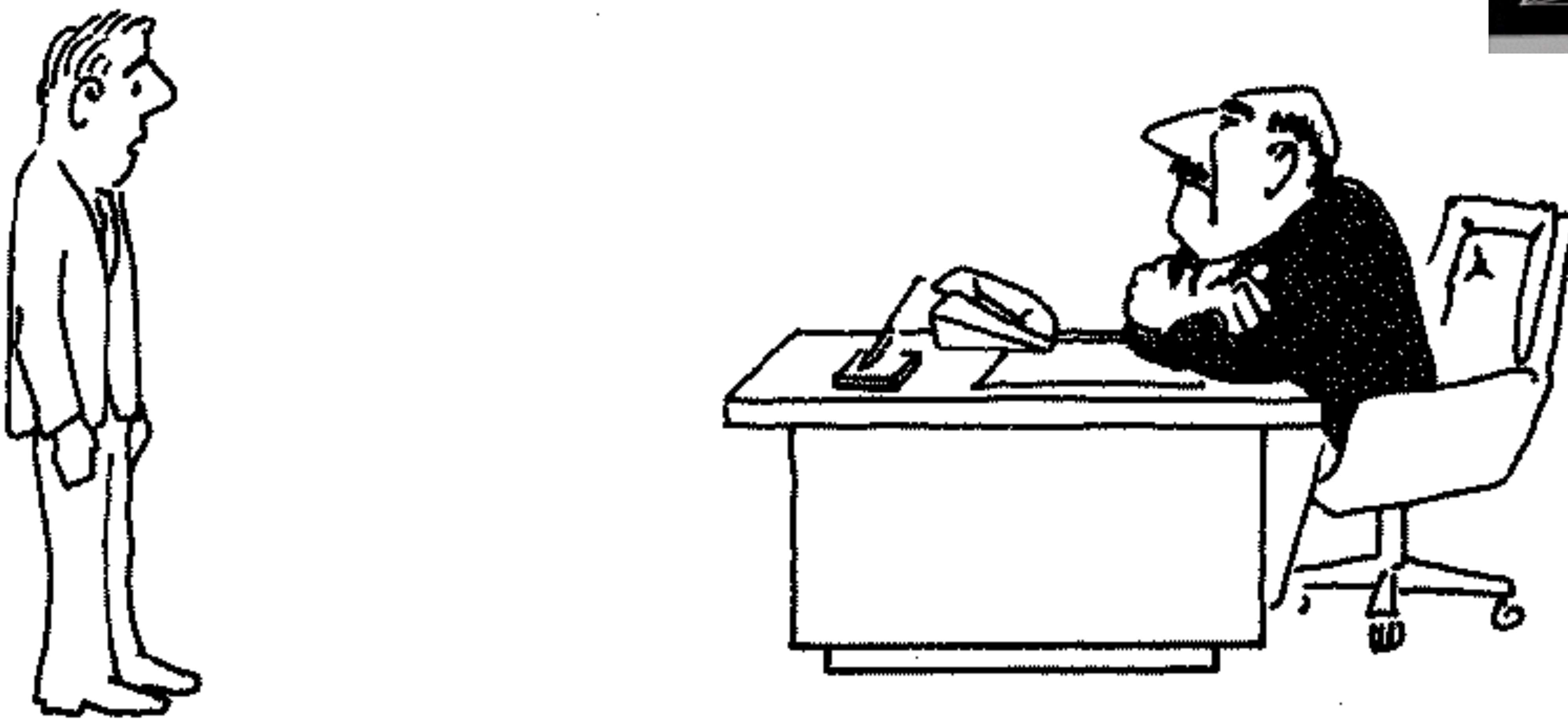
# Implications of Karp + Cook




All of these problems are NP-complete; they are manifestations of the same really hard problem.

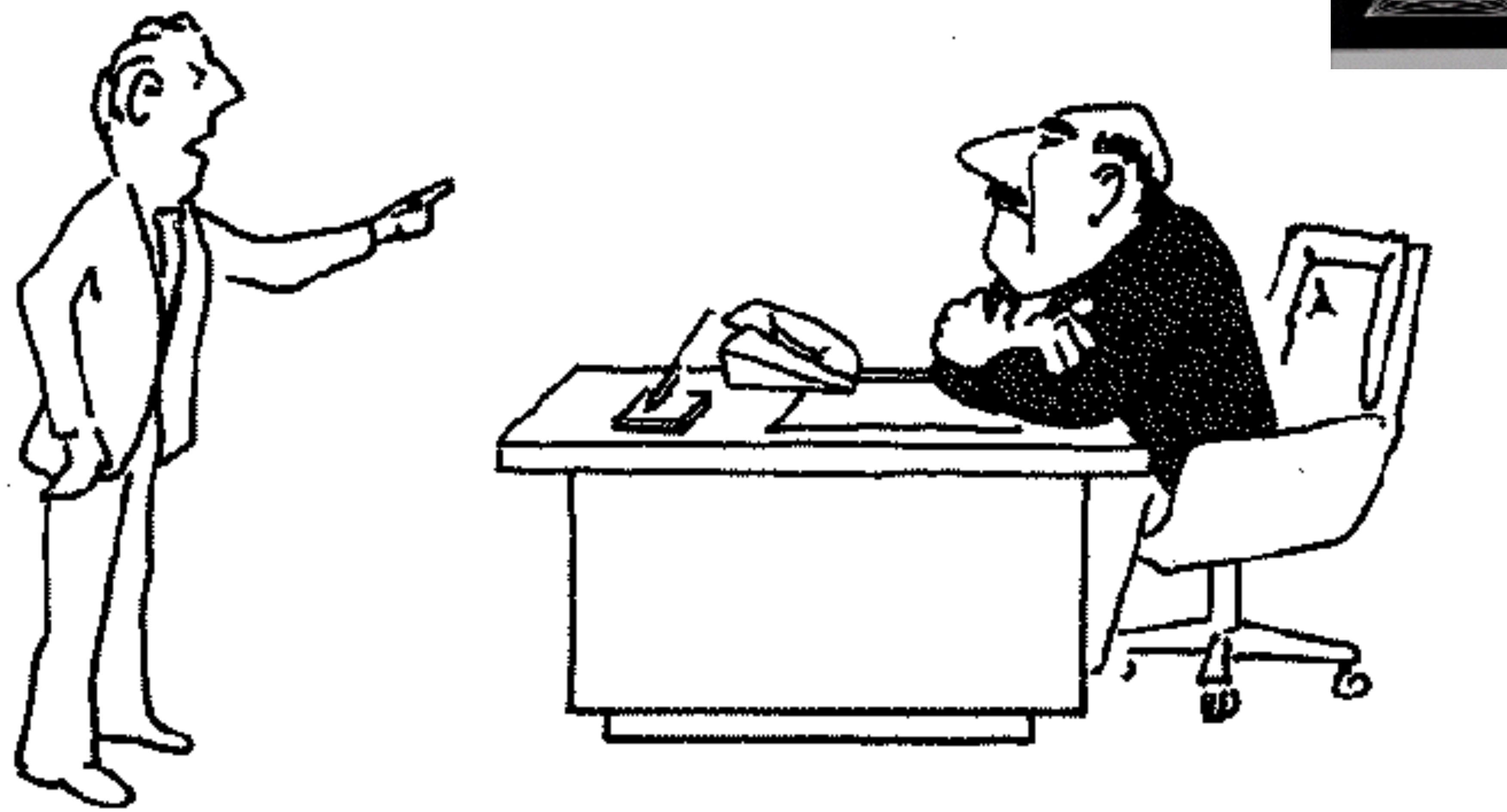
# Implications of NP-completeness

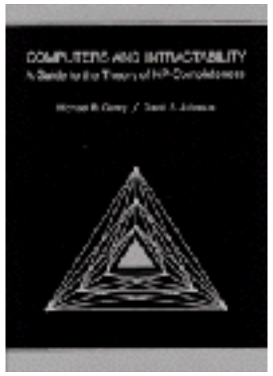




"I can't find an efficient algorithm, I guess I'm just too dumb."

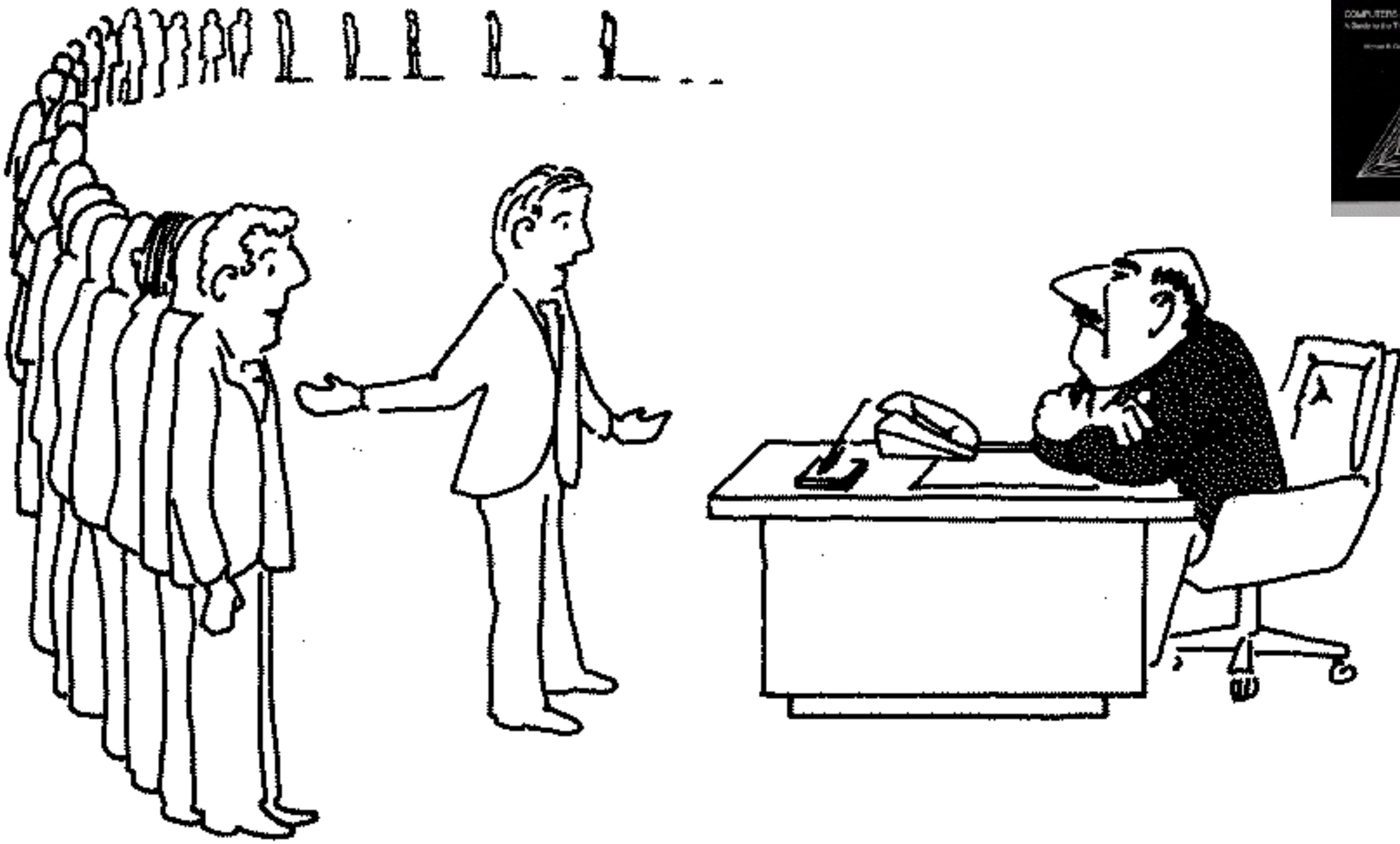
# Implications of NP-completeness

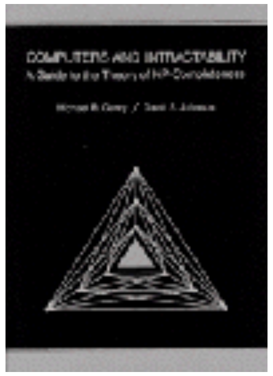




"I can't find an efficient algorithm, because no such algorithm is possible!"

# Implications of NP-completeness





"I can't find an efficient algorithm, but neither can all these famous people."

## Birds-eye view: review

Desiderata. Classify problems according to computational requirements.

| complexity | order of growth | examples |
|---|---|---|
| linear | $N$ | min, max, median, Burrows-Wheeler transform, ... |
| linearithmic | $N \log N$ | sorting, convex hull. closest pair, farthest pair, ... |
| quadratic | $N^2$ | ??? |
| | … | |
| exponential | $c^N$ | ??? |

Frustrating news. Huge number of problems have defied classification.

# Birds-eye view: revised

**Desiderata.** Classify problems according to computational requirements.

| complexity | order of growth | examples |
|---|---|---|
| linear | N | min, max, median,<br>Burrows-Wheeler transform, ... |
| linearithmic | N log N | sorting, convex hull.<br>closest pair, farthest pair, ... |
| 3-SUM complete | probably $N^2$ | 3-SUM, 3-COLLINEAR,<br>3-CONCURRENT, ... |
| | … | |
| NP-complete | probably $c^{\varepsilon N}$ | 3-SAT, IND-SET, ILP, ... |

**Good news.** Can put problems in equivalence classes.

# Summary

Reductions are important in theory to:

- Establish tractability.
- Establish intractability.
- Classify problems according to their computational requirements.

Reductions are important in practice to:

- Design algorithms.
- Design reusable software modules.
  - stack, queue, priority queue, symbol table, set, graph
  - sorting, regular expression, Delaunay triangulation
  - minimum spanning tree, shortest path, maximum flow, linear programming
- Determine difficulty of your problem and choose the right tool.
  - use exact algorithm for tractable problems
  - use heuristics for intractable problems