https://algs4.cs.princeton.edu

Intractability

- *introduction*
- *computational problems*
- *poly-time algorithms*
- *P vs. NP*
- *poly-time reductions*
- *coping with intractability*

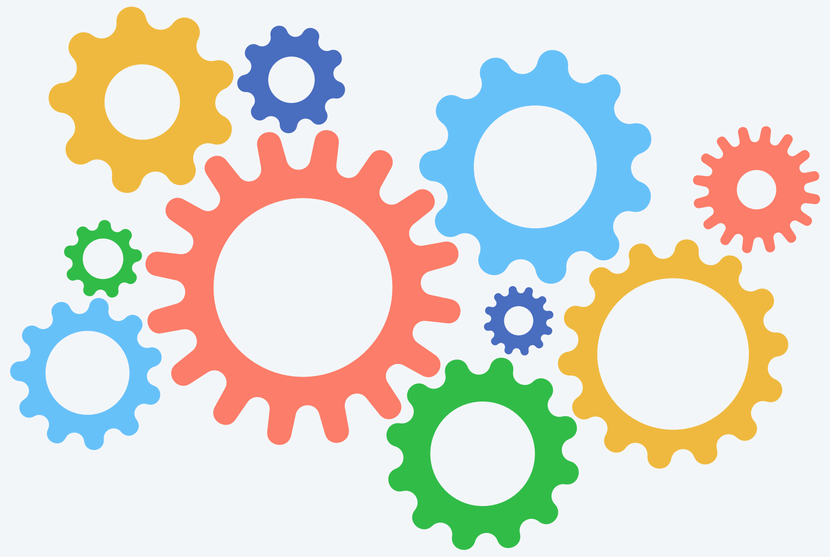
Overview: introduction to advanced topics

Main topics. [final two lectures]

- Intractability: barriers to designing efficient algorithms.
- Algorithm design: general paradigms for solving computational problems.

Shifting gears.

- From individual problems to problem-solving models.
- From linear/quadratic to poly-time/exponential scale.
- From implementation details to conceptual frameworks.



Goals.

- Introduce you to essential ideas.
- Place algorithms and techniques we've studied in a larger context.

https://algs4.cs.princeton.edu

Intractability

- *introduction*
- *computational problems*
- *poly-time algorithms*
- *P vs. NP*
- *poly-time reductions*
- *coping with intractability*

Fundamental questions

Q1. What is an algorithm?

Q2. What is an efficient algorithm?

Q3. Which problems are intractable?

Q4. How can we cope with intractability?

Q5. How can we benefit from intractability?

Integer multiplication

$$37 \cdot 79 = \ ?$$

Integer factorization

$$? \cdot ? = 2881$$

Integer factorization

$$? \cdot ? = \begin{array}{l}12301866845301177551304949583\\84962720772853569595334792197\\32245215172640050726365751874\\52021997864693899564749427740\\63845925192557326303453731548\\26850791702612214291346167042\\92143116022212404792747377940\\80665351419597459856902143413\end{array}$$

$50,000
RSA factoring challenge
2 years, team of mathematicians

Integer multiplication

$$\begin{matrix}3674604366679959042824463379962795263227915816434308764267603228381573966651127923337341714339681027009279873630891 7\end{matrix} \cdot \begin{matrix}33478071698956898786044169848212690817704794983713768568912431388982883793878002287614711652531743087737814467999489\end{matrix} = ?$$

Computed in a split second by a standard laptop!

https://algs4.cs.princeton.edu

Intractability

- introduction
- **computational problems**
- poly-time algorithms
- P vs. NP
- poly-time reductions
- coping with intractability

Integer multiplication: computationally easy

MULTIPLY. Given positive integers x and y, compute $x \cdot y$.

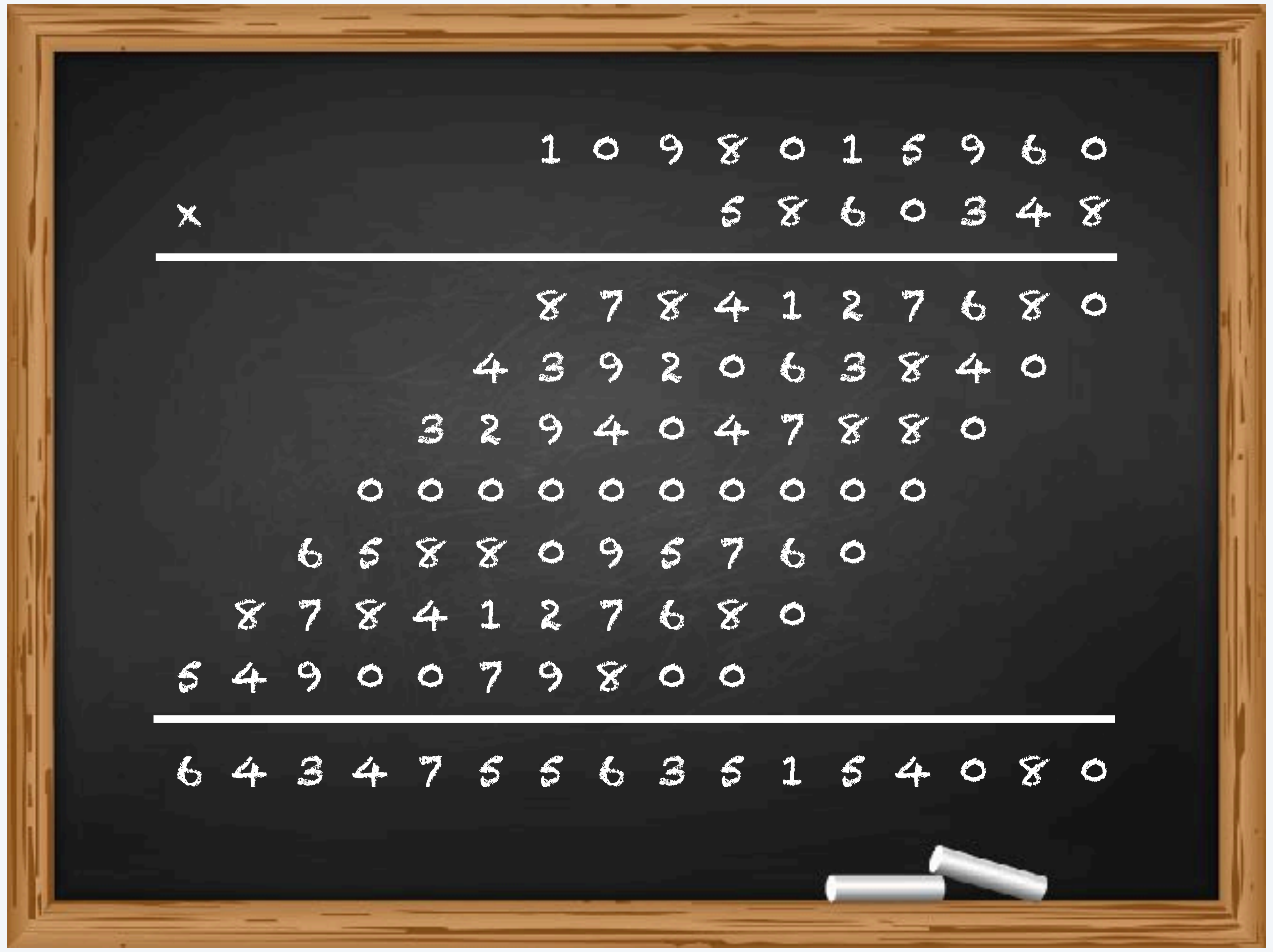
Ex.

$$x = 1098015960$$
$$y = 5860348$$

a MULTIPLY instance

$$xy = 6434755635154080$$

the product



Algorithm. Grade-school multiplication runs in time $\Theta(n^2)$, where n is the number of digits in x and y.

Integer factorization: computationally hard?

FACTOR. Given positive integer x, find a nontrivial factor. ← *or report that no such factor exists*

between 1 and x

Ex. 147573952589676412927 | 193707721

a FACTOR instance | a factor

25195908475657893494027183240048398571429282126204032027777137836043662020707595556264018525880784406918290641249515082189298559149176184502808489120072844992687392807287776735971418347270261896375014971824691165077613379859095700097330459748808428401797429100642458691817195118746121515172654632282216869987549182422433637259085141865462043576798423387184774447920739934236584823824281198163815010674810451660377306056201619676256133844143603833904414952634432190114657544454178424020924616515723350778707749817125772467962926386356373289912154831438167899885040445364023527381951378636564391212010397122822120720357

a very challenging FACTOR instance
(factor to earn an A+ in COS 226)

Brute-force search. Try all possible divisors between 2 and $\sqrt{x}$.

a nontrivial factor > $\sqrt{x}$ implies another < $\sqrt{x}$

Applications. Cryptography. [stay tuned]

How difficult can it be?

Imagine a galactic computer…

- With as many processors as electrons in the universe.
- Each processor having the power of today's supercomputers.
- Each processor working for the lifetime of the universe.

quantity	estimate
electrons in universe	10^{79}
instructions per second	10^{13}
age of universe in seconds	10^{17}

Q. Could galactic computer factor a 300-digit integer using brute-force search?

A. Not even close: $\sqrt{10^{300}} = 10^{150} \gg 10^{79} \cdot 10^{13} \cdot 10^{17} = 10^{109}$.

Lesson. Exponential growth dwarfs technological change.

Boolean satisfiability: computationally hard?

SAT. Given a system of boolean equations, find a satisfying truth assignment. ← *or report that no such assignment is possible*

Ex.

$\neg a$	*or*	$\neg b$	*or*	$\neg c$		=	*true*
a	*or*	b		*or*	d	=	*true*
$\neg a$	*or*	$\neg b$		*or*	$\neg d$	=	*true*
a	*or*	b	*or*	c		=	*true*
a	*or*	$\neg b$				=	*true*

a SAT instance

a	=	*true*
b	=	*true*
c	=	*false*
d	=	*false*

a satisfying truth assignment

Brute-force search. Try all 2^n truth assignments (where n is number of variables).

Applications. Automatic software verification, mean field diluted spin glass model, EDA…

equivalent to **P ≠ NP**

Remark. More "evidence" of hardness than FACTOR.

https://algs4.cs.princeton.edu

Intractability

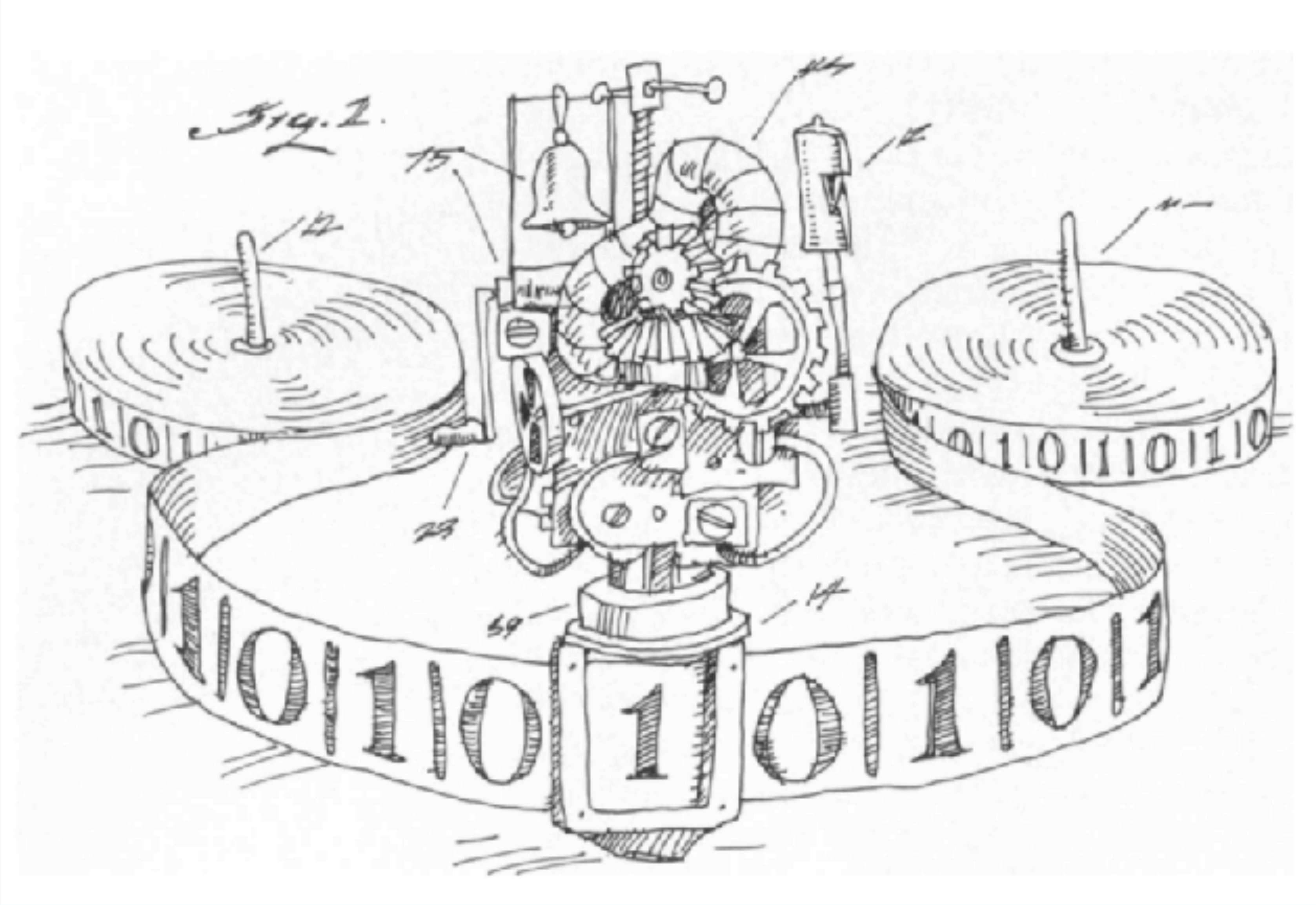
- ‣ *introduction*
- ‣ *computational problems*
- ‣ ***poly-time algorithms***
- ‣ *P vs. NP*
- ‣ *poly-time reductions*
- ‣ *coping with intractability*

Algorithm

Q1. What is an algorithm?

A1. Formally, a Turing machine!

Equivalently, a program in Java, Python, C++, …



A Turing machine

Church–Turing thesis. Any computational problem that can be solved by a physical system can also be solved by a Turing machine.

Efficient algorithm

Q2. What is an efficient algorithm?

A2. One with worst-case running time polynomial in the *size of its input*.

Polynomial time. Number of elementary operations is $\le an^b$ for some constants a, b.

↑ n = # *of bits in input*

Context. We use poly-time as a surrogate for efficient in practice.

- Robust.
- Closed under composition.
- In practice, constants tend to be small.

order	emoji	name	today
$\Theta(1)$	😍	*constant*	🙂
$\Theta(\log n)$	😎	*logarithmic*	🙂
$\Theta(n)$	😁	*linear*	🙂
$\Theta(n \log n)$	😀	*linearithmic*	🙂
$\Theta(n^2)$	😕	*quadratic*	🙂
$\Theta(n^3)$	🙁	*cubic*	🙂
$\Theta(n^{\log n})$	😨	*quasipolynomial*	👿
$\Theta(1.1^n)$	😭	*exponential*	👿
$\Theta(2^n)$	😈	*exponential*	👿
$\Theta(n!)$	👿	*factorial*	👿

Intractability: poll 1

Which of the following are poly-time algorithms?

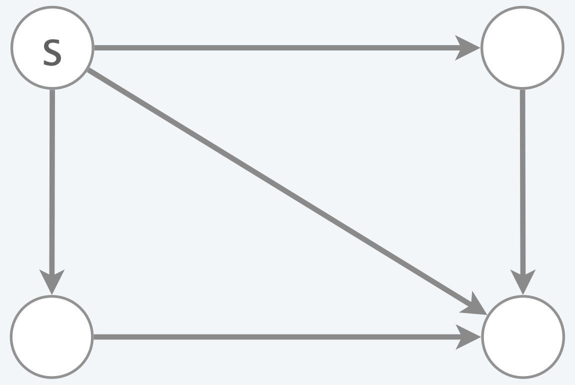
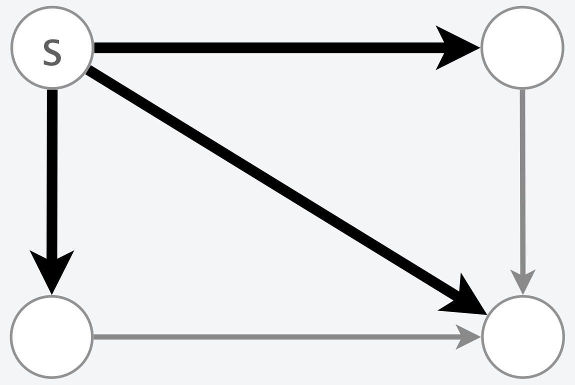
A. Brute-force search for boolean satisfiability.

B. Brute-force search for integer factoring.

C. Both A and B.

D. Neither A nor B.

Some computational problems

problem	description	example instance	a solution	poly-time algorithm
SHORTEST-PATHS (*single-source shortest paths*)	given an unweighted graph, find the shortest paths from source			*BFS*
PRIME (*primality*)	is the given integer prime?	53	yes	*Agrawal-Kayal-Saxena*
JAVA (*Java compilation*)	given a text file, compile into Java byte code	`Percolation.java`	`Percolation.class`	`javac`
FACTOR (*integer factorization*)	given a positive integer, find a nontrivial factor	147573952589676412927	193707721	?
BITCOIN (*bitcoin mining*)	given 76 bytes, find 4 bytes such that concatenation hashes to ≤ target	0020b128b5fe690…7995389f1	995389f1	?
⋮	⋮	⋮	⋮	⋮

Types of computational problems

Search problem. Find a solution.

Decision problem. Does there exist a solution?

Optimization problem. Find the best solution.

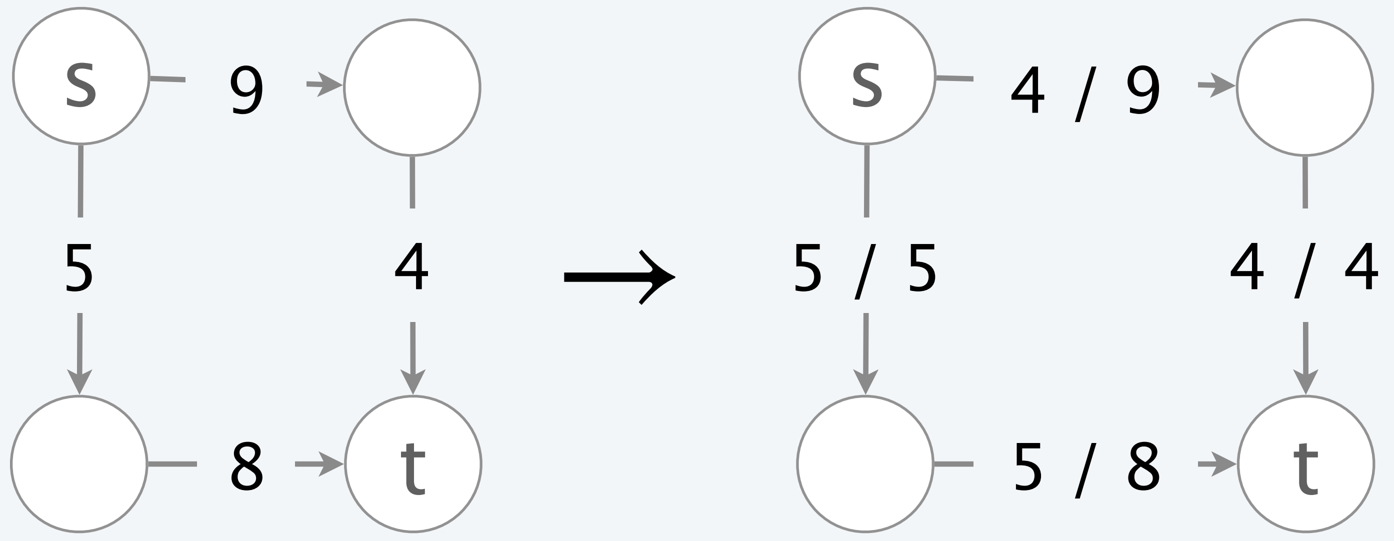
Function problem. Compute the output of a mathematical function.

$2881 \to$ 43 or 67

Factoring: a search problem

$53 \to$ yes

Primality: a decision problem



Maxflow: an optimization problem

$43 \cdot 67 \to 2881$

Multiplication: a function problem

Remarks.

- Problems often naturally formulated in one regime.
- Types are not technically equivalent, but conclusions generalize.
- Definitions of **P** and **NP** are in terms of decision problems.

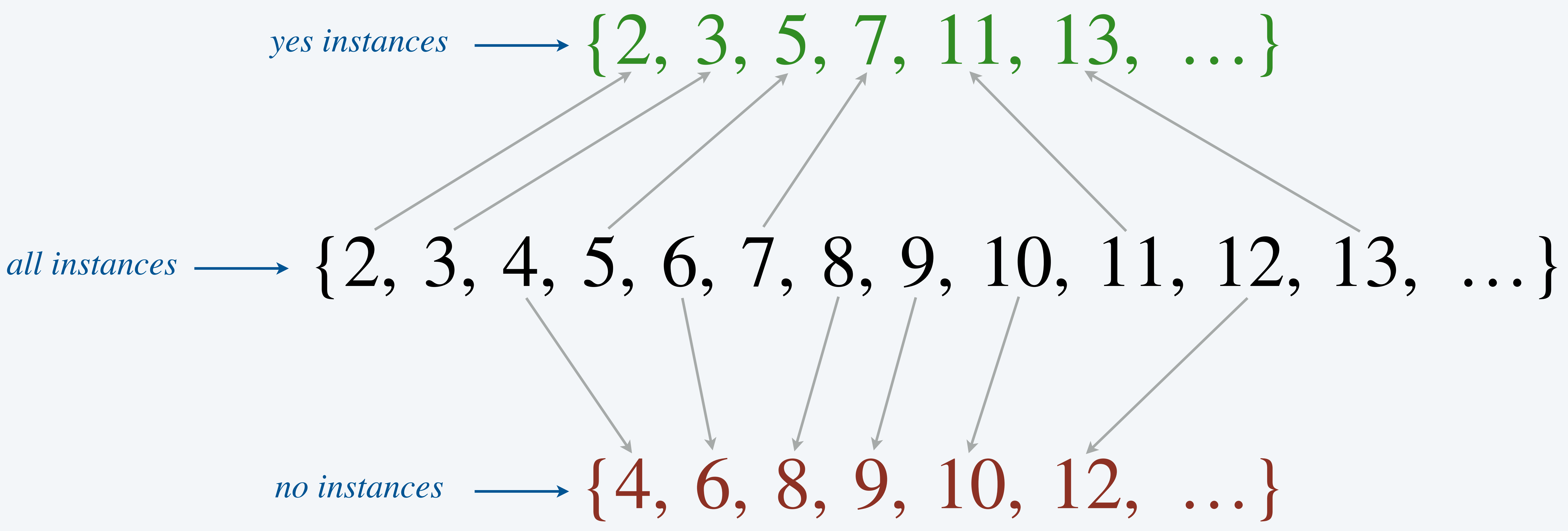
https://algs4.cs.princeton.edu

Intractability

- *introduction*
- *computational problems*
- *poly-time algorithms*
- *P vs. NP*
- *poly-time reductions*
- *coping with intractability*

Decision problems

A decision problem is a partition of input strings into yes-instances and no-instances.



Primality: a decision problem

The P complexity class

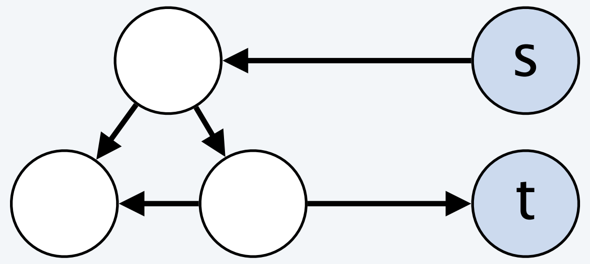
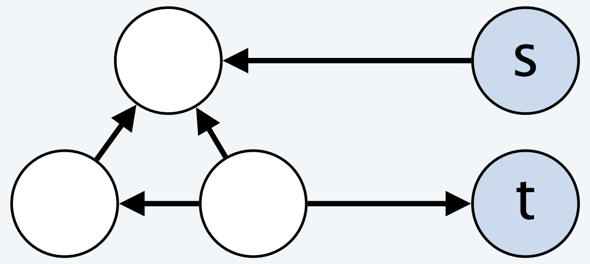
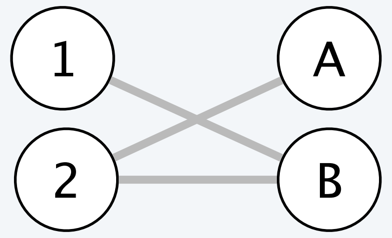
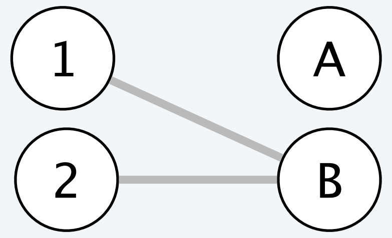
Solving a decision problem. Designing an algorithm that, on every instance x, outputs the correct yes or no classification.

Definition. **P** is the set of all decision problems that are solvable in polynomial time.

Some computational decision problems

problem	description	example yes instance	example no instance	poly-time algorithm	In P?
PRIME	is the integer x prime?	$x = 53$	$x = 10$	*Agrawal-Kayal-Saxena*	✔
COMPOSITE	is the integer x composite (not prime)?	$x = 10$	$x = 53$	*Agrawal-Kayal-Saxena*	✔
COPRIME	are the nontrivial factors of integers x and y disjoint?	$x = 10$ $y = 14$	$x = 10$ $y = 21$	*Euclid's GCD algorithm*	✔
3-SUM	is there a triple in array a that sums to 0?	$a = [-1, 1, 0]$	$a = [-1, 1, 2]$	*check all triples with nested* `for` *loop*	✔
JAVA	is the text file a legal Java program?	any assignment file (that compiles)	`Percolation {}`	`javac`	✔

Some computational decision problems

problem	description	example yes instance	example no instance	poly-time algorithm	In P?
ST-CONN	is there a path from s to t?			*BFS*	✔
BIPARTITE-MATCHING	is there a perfect matching in G?			*Ford-Fulkerson*	✔
SAT	is system Φ of boolean equations satisfiable?	a *and* $(\neg\, a$ *or* $\neg\, b)$	a *and* $\neg\, a$	?	?
FACTOR	does x have nontrivial factor $\leq k$?	$x = 49$ $k = 7$	$x = 49$ $k = 5$	?	?
⋮	⋮	⋮	⋮	⋮	⋮

The NP complexity class

Definition. **NP** is the set of all decision problems that are verifiable in polynomial time. ⟵ *as opposed to "solvable"* (**P**)

Definition'. **NP** is the set of all decision problems for which a yes instance can be verified, provided a witness, in polynomial time.

The NP complexity class

Definition. **NP** is the set of all decision problems for which a yes instance can be verified, provided a witness, in polynomial time.

often, solution of the search problem.
a.k.a. certificate

$\neg a$	*or*	$\neg b$	=	*true*
a			=	*true*

SAT instance

Examples.

- SAT: Is the given system Φ of boolean equations satisfiable?
  - Witness. A boolean assignment that satisfies every equation.
  - Verification algorithm. Output yes if the assignment satisfies Φ (and no otherwise).

a	=	*true*
b	=	*false*

witness

The NP complexity class

Definition. **NP** is the set of all decision problems for which a yes instance can be verified, provided a witness, in polynomial time.

Examples.

- FACTOR: Given integers x and $k < x$, does x have a nontrivial factor $\leq k$?
  - Witness. A nontrivial factor $y \leq k$ of x.
  - Verification algorithm. Output yes if $1 < y \leq k$ and y divides x (and no otherwise).

$x = 2881$ $k = 50$

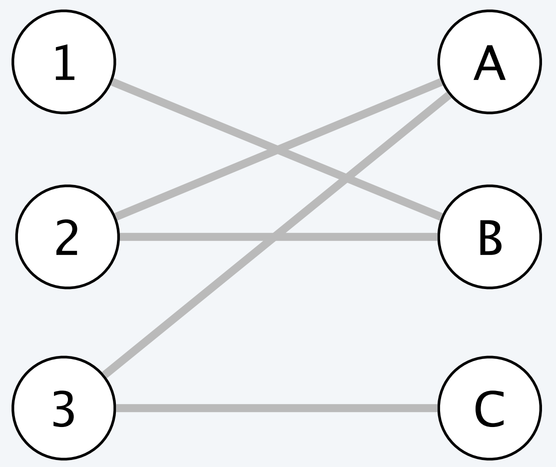
FACTOR instance

$y = 43$

witness

The NP complexity class

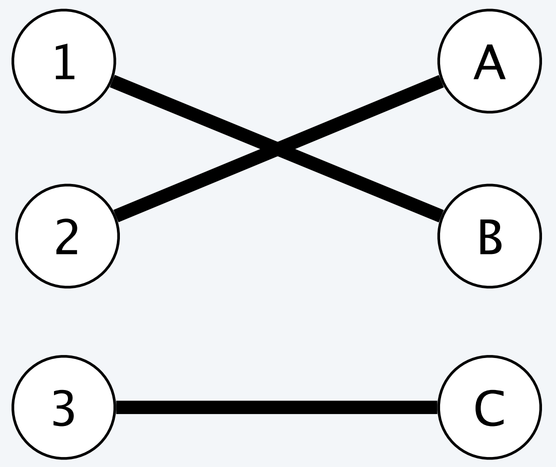
Definition. **NP** is the set of all decision problems for which a yes instance can be verified, provided a witness, in polynomial time.



BIPARTITE-MATCHING instance

Examples.

- BIPARTITE-MATCHING: Does the given a bipartite graph G have a perfect matching?
  - Witness. A subset of edges M that is a perfect matching in G.
  - Verification algorithm. Output yes if every vertex of G is incident to exactly one edge in M (and no otherwise).



witness

Remark 1. An **NP** verifier does *not* find a witness, and must output no if given a no instance (with any purported witness) as input.

Remark 2. **NP** problems can be solved in exponential time by verifying all witnesses.

Some NP problems

problem	instance	description	witness	verification algorithm
Any problem Q in **P**	x	is x a yes instance of Q?	w = *empty string*	algorithm that solves Q
SAT	system Φ of boolean equations	is Φ satisfiable?	w = *true/false assignment for all variables*	plug w into equations of Φ, check that all evaluate to *true*
LONGEST-ST-PATH	weighted digraph G, source s, target t and integer k	is the length of the longest simple st-path $\geq k$?	w = *st-path with* $\geq k$ *edges*	check that w does not repeat vertices and is $\geq k$ edges of G
BITCOIN	positive integers x and t	is there y such that $h(x \circ y) \leq t$?	$w = y$	check that $h(x \circ y) \leq t$
⋮	⋮	⋮	⋮ ⋮	⋮

Which of these problems are (known to be) in NP?

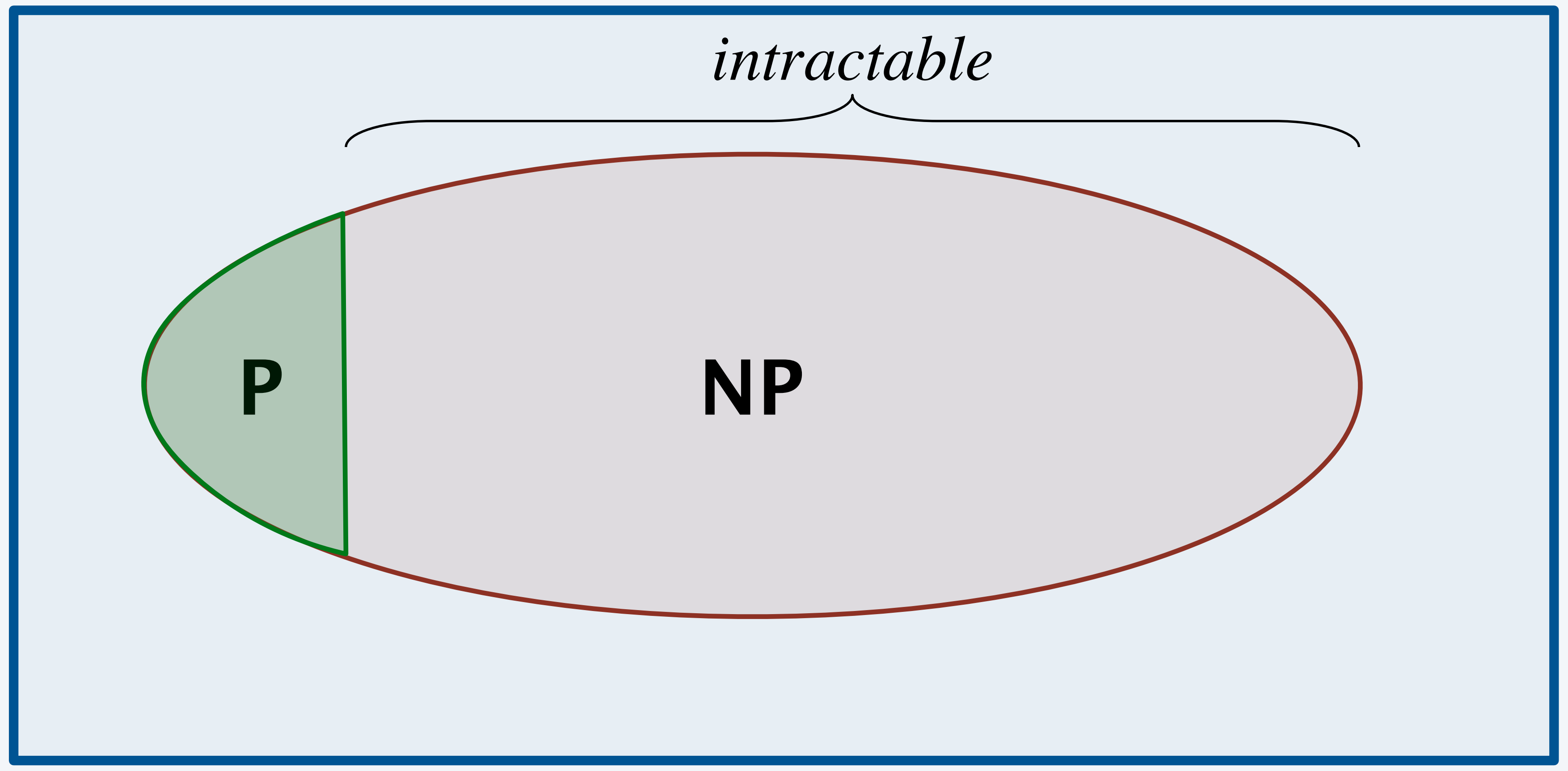
- **A.** Given a graph G, find a simple path with the most edges.
- **B.** Given a graph G and an integer k, is there a simple path with $\geq k$ edges?
- **C.** Both A and B.
- **D.** Neither A nor B.

P vs. NP

The central question. Does **P** = **NP** ?

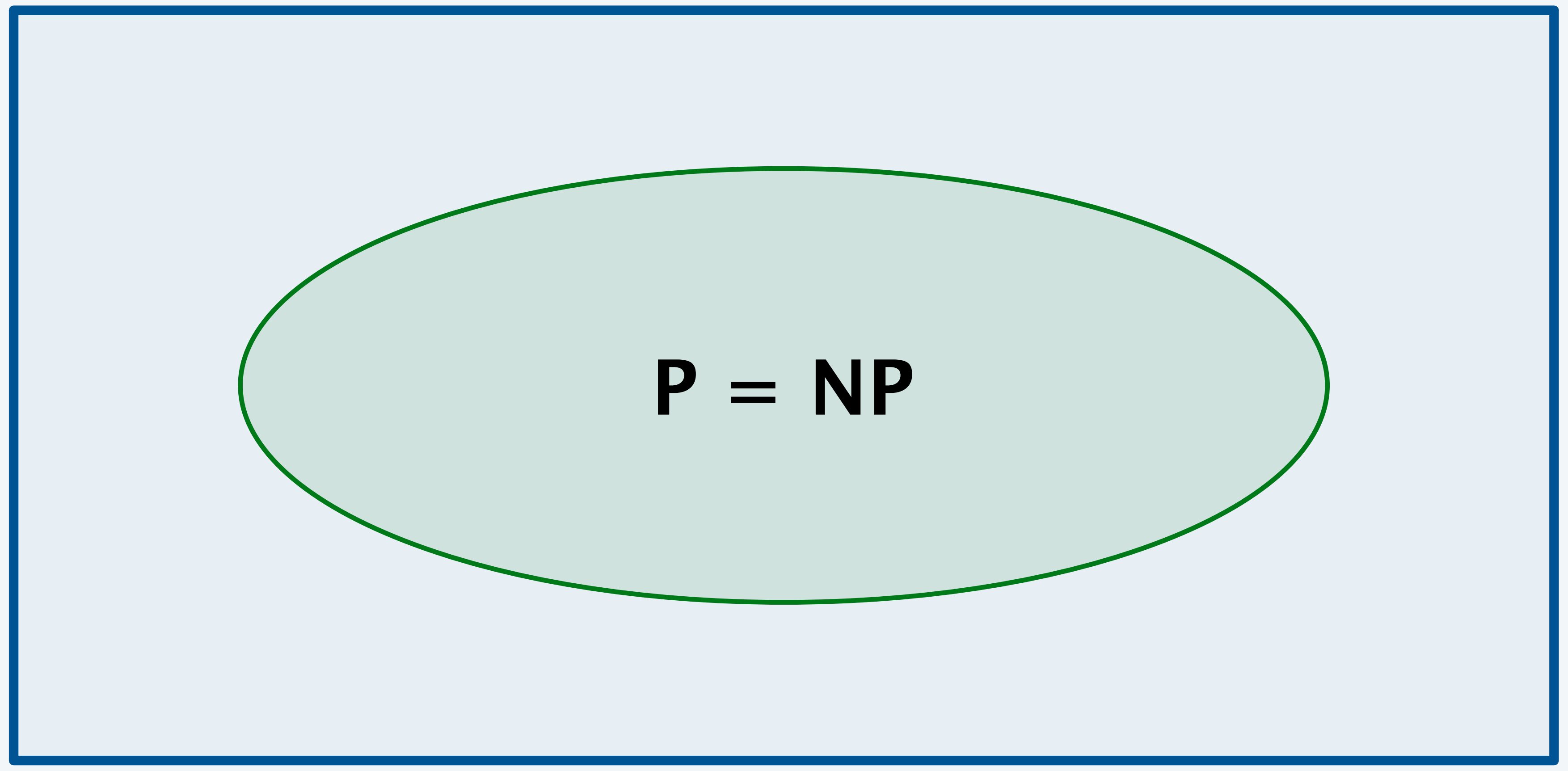
- **P** = set of decision problems solvable in poly-time.
- **NP** = set of decision problems verifiable in poly-time (given witness).

Two possible worlds. Since **NP** contains **P**, ← *empty string serves as witness*



P ≠ NP

brute-force search may be the best we can do



P = NP

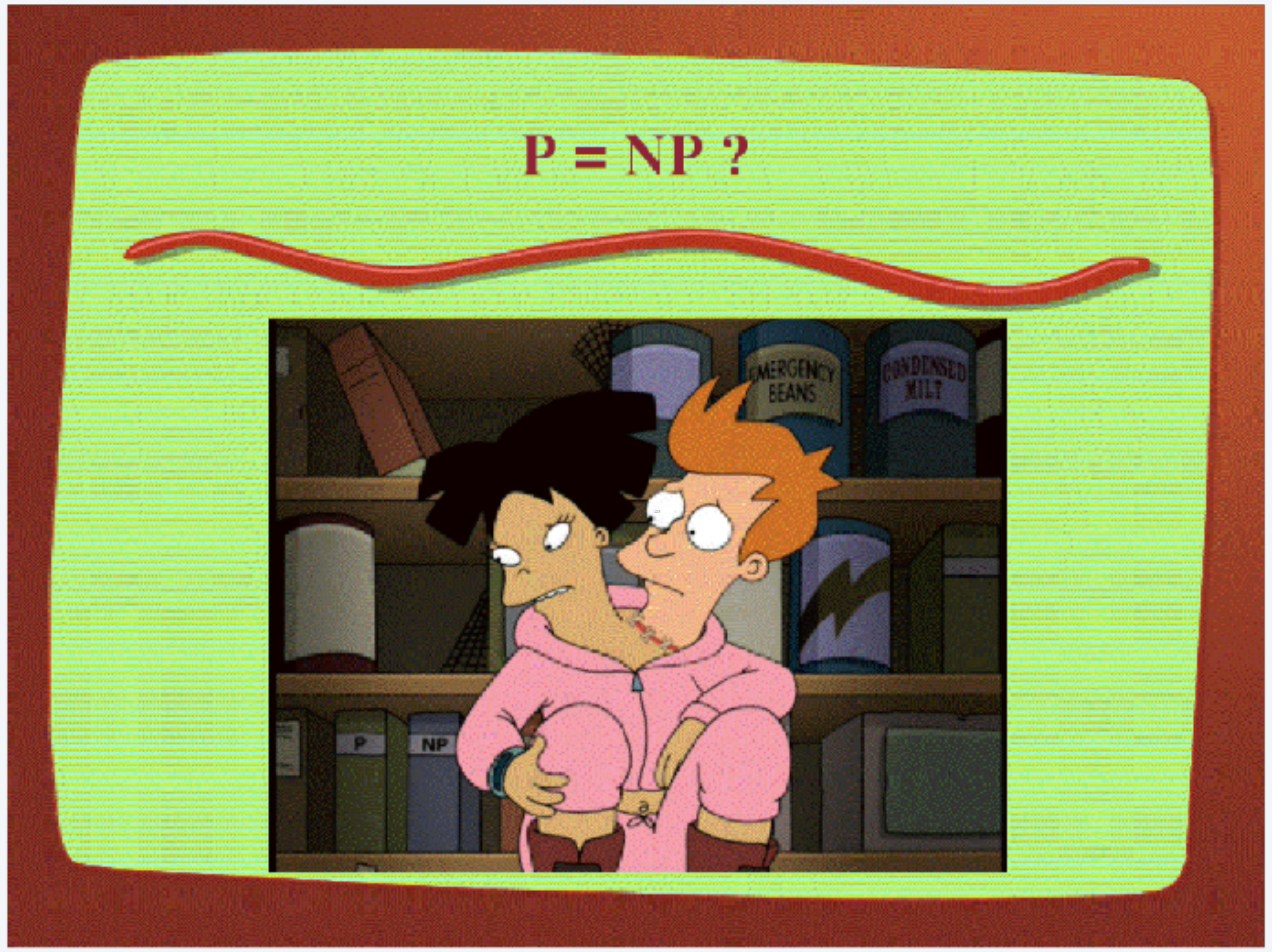
poly-time algorithms for FACTOR, SAT, LONGEST-ST-PATH, …

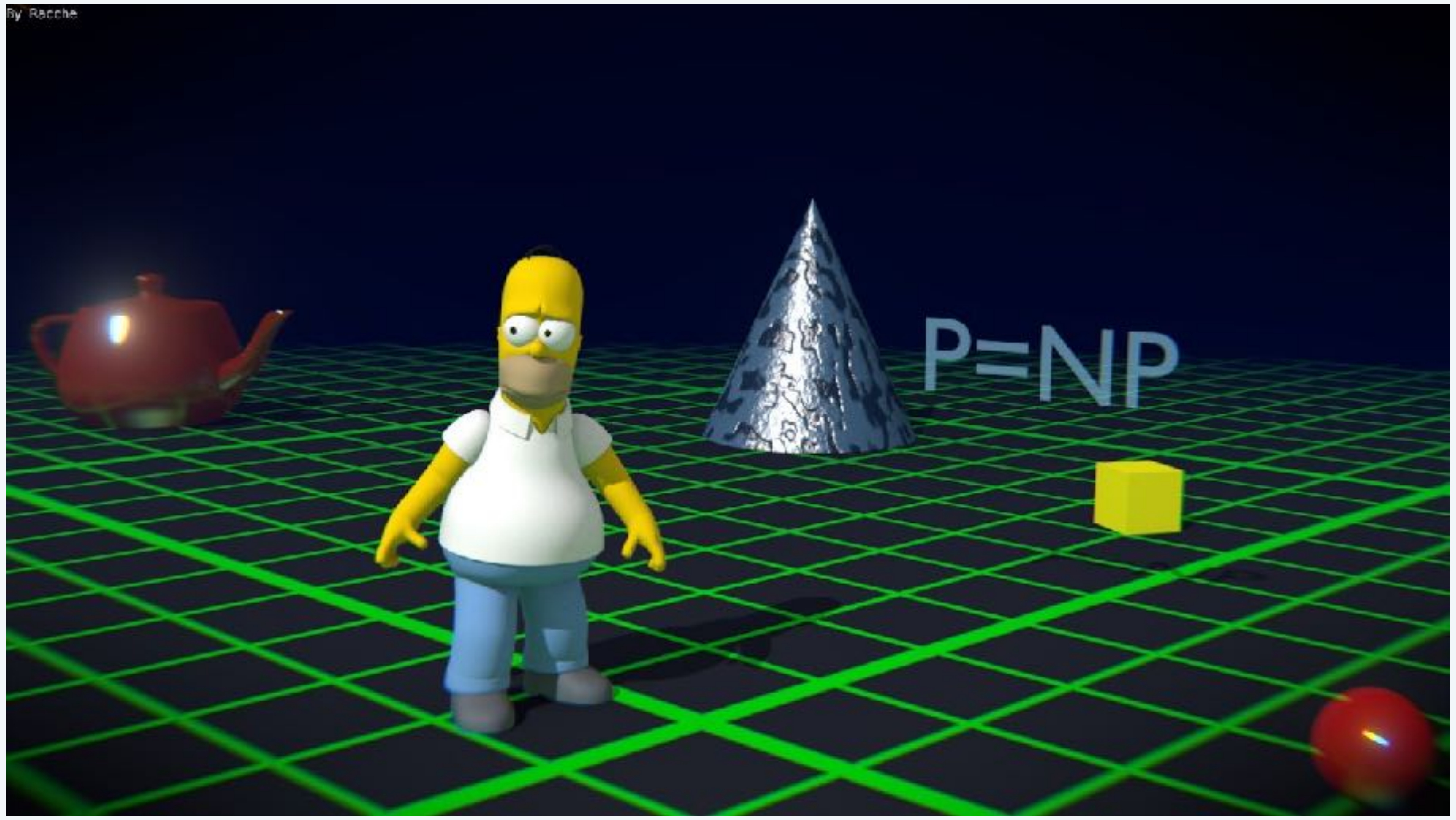
Consensus opinion. **P ≠ NP.** ← *but nobody has been able to prove or disprove (!!!)*

P vs. NP

The central question. Does **P** = **NP** ?

- **P** = set of decision problems solvable in poly-time.
- **NP** = set of decision problems verifiable in poly-time (given witness).





Why P vs. NP is so central?

Analogy. Creative genius vs. ordinary appreciation of creativity.

domain	problem	witness/certificate
mathematics	find a proof of a conjecture	mathematical proof
engineering	given constraints (size, weight, energy), find a design (bridge, medicine, computer)	blueprint
science	given data on a phenomenon, find a theory explaining it	scientific theory
the arts	write a beautiful poem/novel/pop song; draw a beautiful painting	poem, novel, pop song, painting
programming	write a program to solve a problem	program

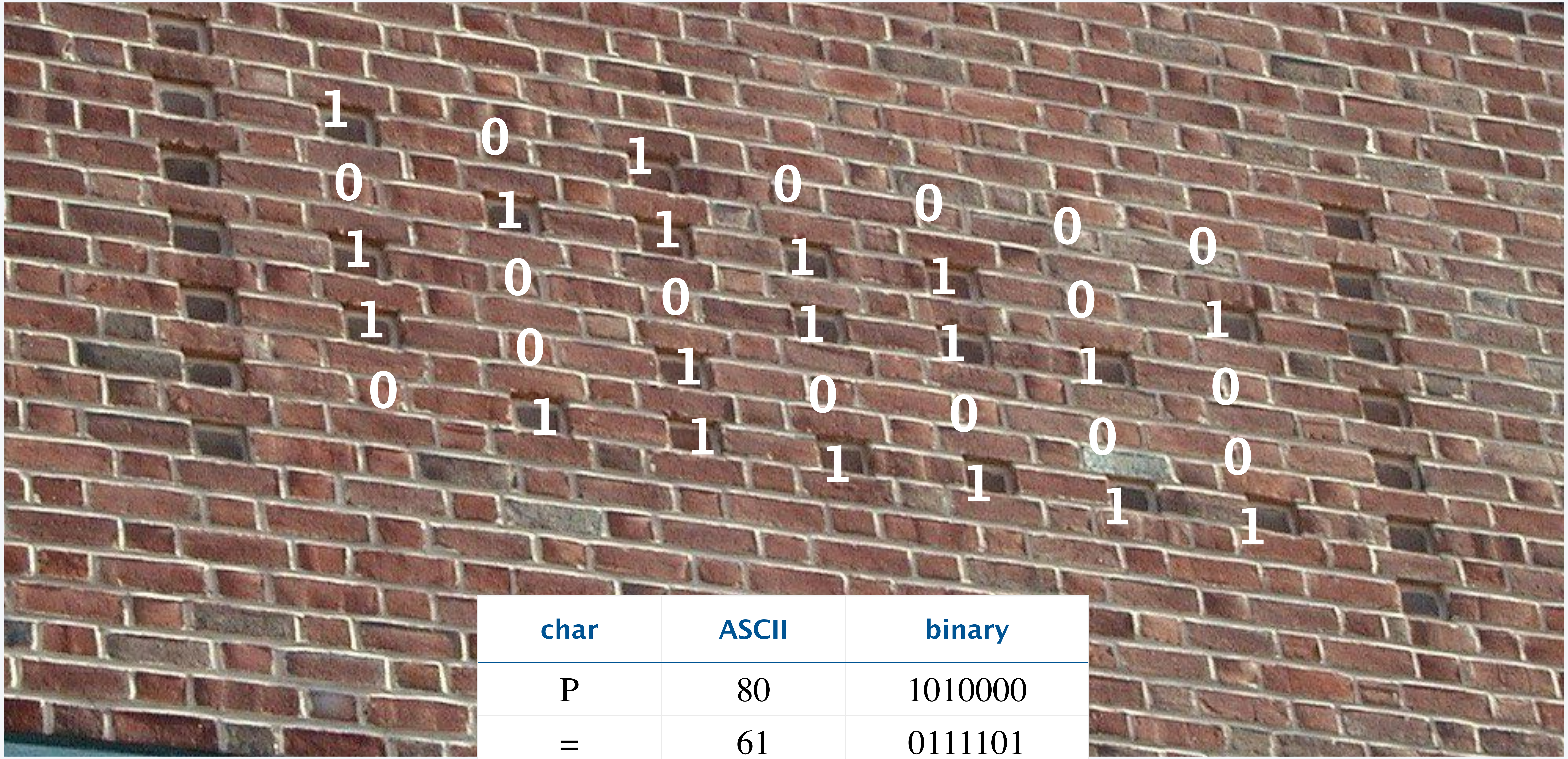
creative genius (NP)

ordinary appreciation (P)

Intuition. Verifying a solution should be way easier than finding one.

Princeton computer science building

Princeton computer science building (closeup)



char	ASCII	binary
P	80	1010000
=	61	0111101
N	78	1001110
P	80	1010000
?	63	0111111

https://algs4.cs.princeton.edu

Intractability

Bird's-eye view

Design strategy. Suppose we can solve problem X efficiently.

Which other problems can we solve efficiently?

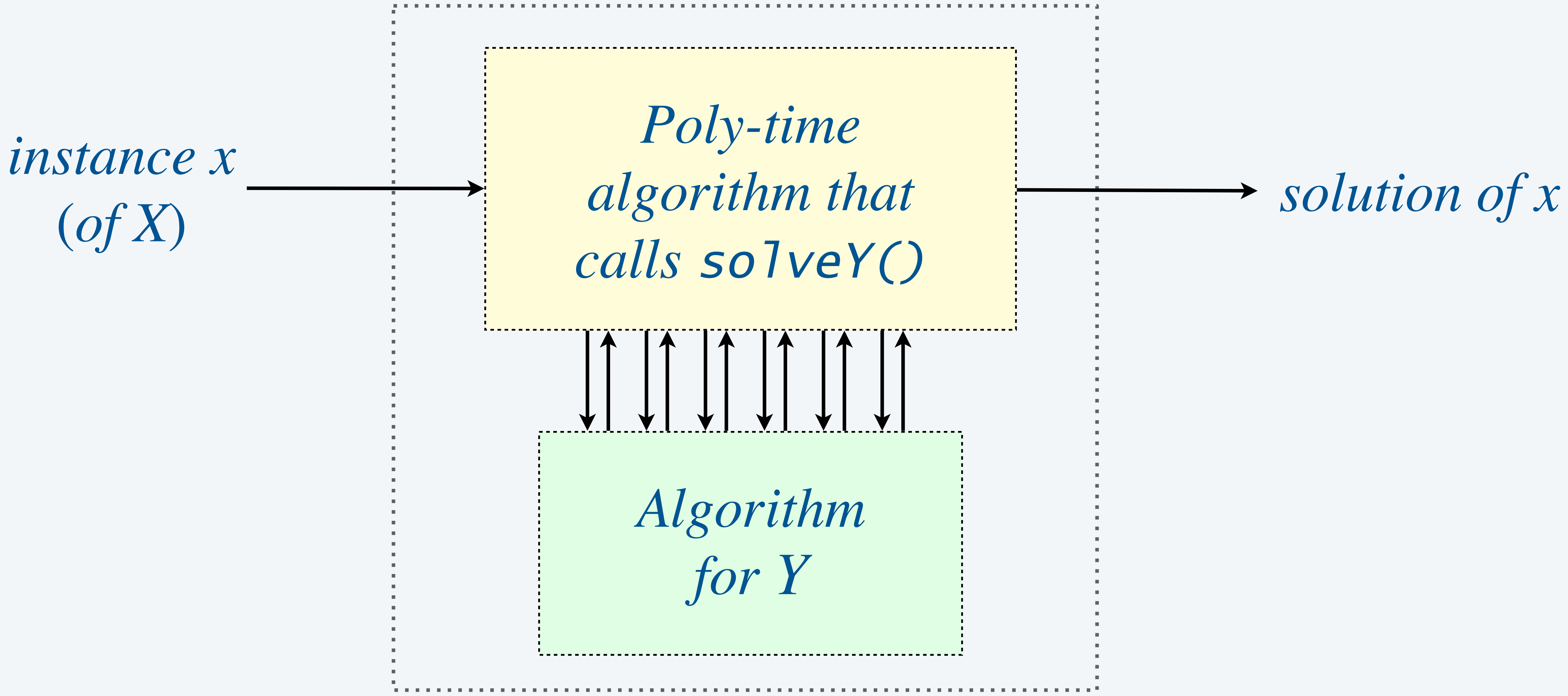
> *" Give me a lever long enough and a fulcrum on which to place it, and I shall move the world."* — *Archimedes*

Poly-time reduction

Definition. Problem X poly-time reduces to problem Y ($X \preceq Y$) if ← *Cook reduction*

X can be solved with:

- Polynomial number of elementary operations.
- Polynomial number of calls to algorithm for Y.



algorithm for problem X

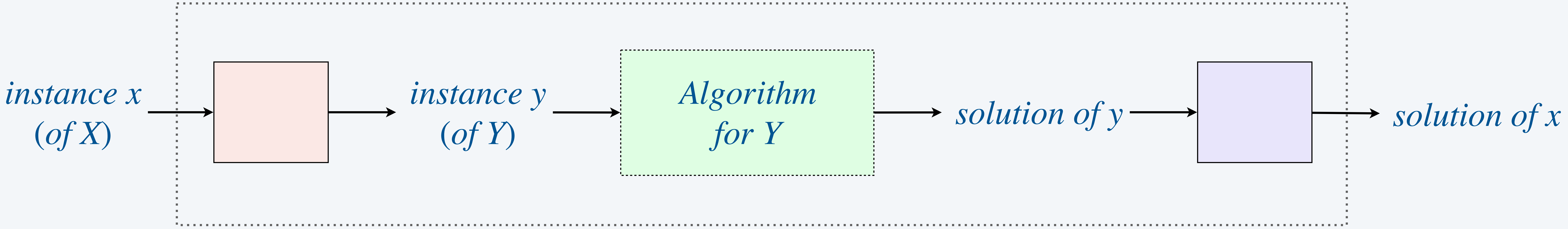
Remark. We can reduce to/from search, decision or optimization problems.

Poly-time reduction

Important special case. Problem X poly-time reduces to problem Y if ← *Levin reduction (Cook with one call to* `solveY`*)*

X can be solved by:

1. Mapping instance of X into instance of Y (in poly-time).
2. Running algorithm for Y on new instance;
3. Mapping solution of Y to solution of X (in poly-time).

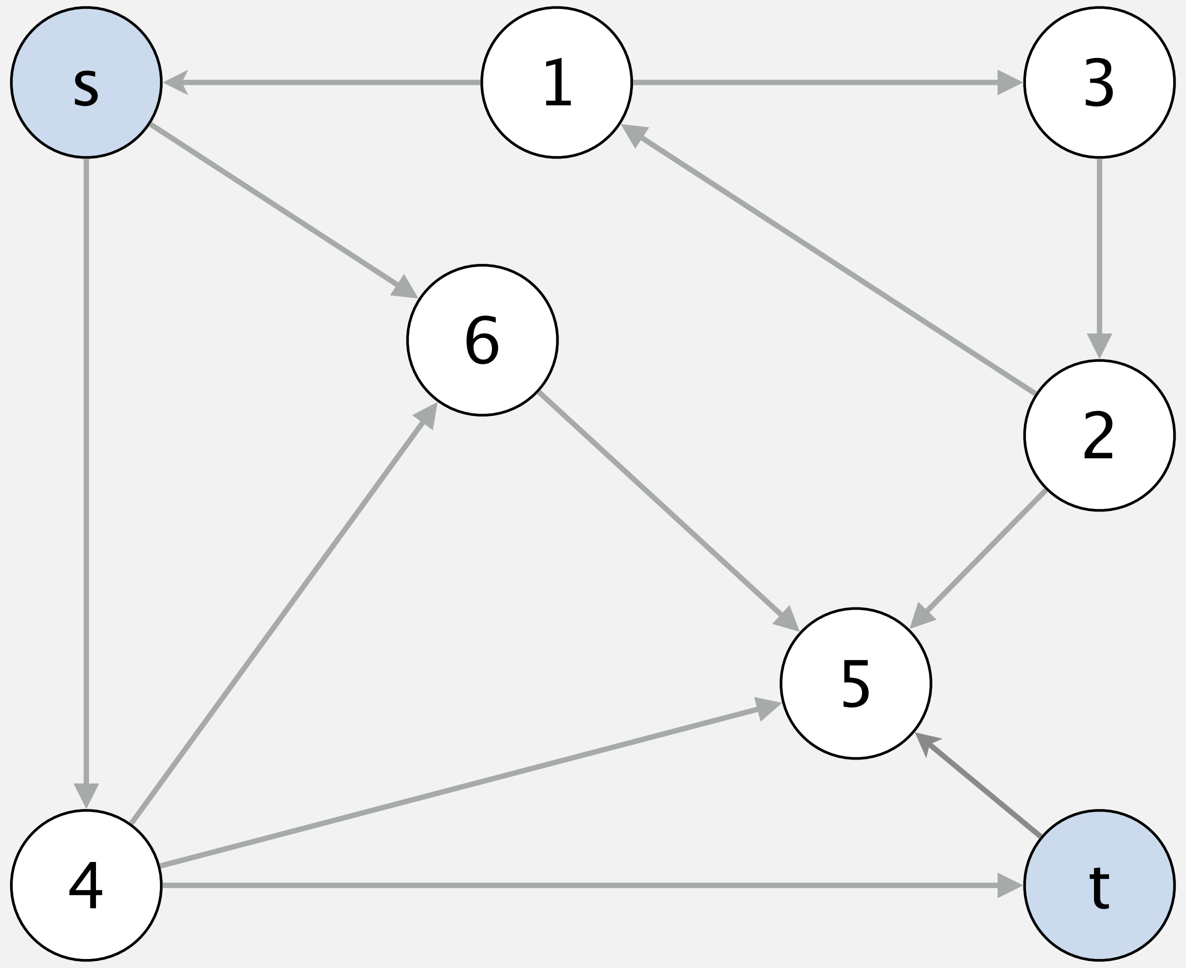


algorithm for problem X

Algorithm design. Efficient algorithm for Y yields efficient algorithm for X.

Poly-time reduction: ST-CONN to SHORTEST-PATHS

Example 1. ST-CONN poly-time reduces to SHORTEST-PATHS. ← *decision reduces to search*



instance of ST-CONN

Poly-time reduction: ST-CONN to SHORTEST-PATHS

Example 1. ST-CONN poly-time reduces to SHORTEST-PATHS. ← *decision reduces to search*

v	s	1	2	3	4	5	6	t
marked[]	✔	✘	✘	✘	✔	✔	✔	✔

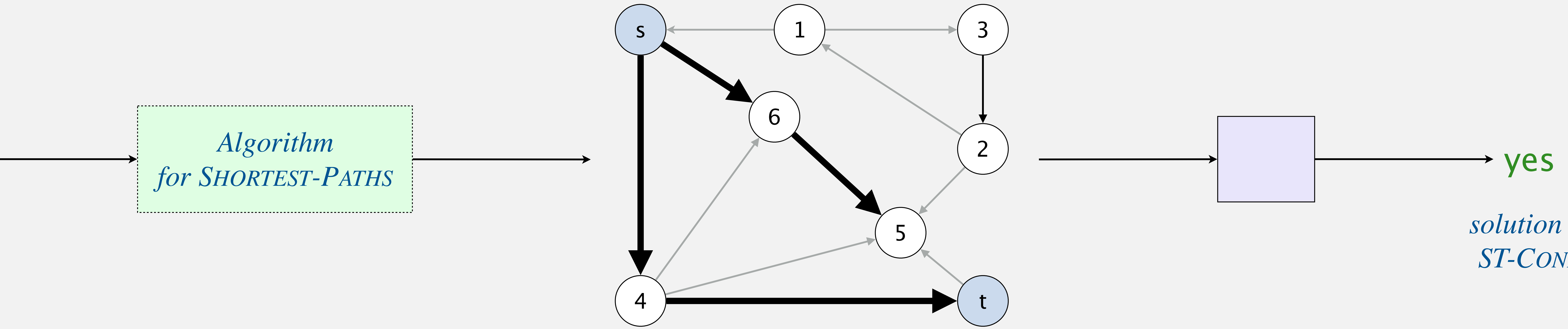
instance of SHORTEST-PATHS

solution of SHORTEST-PATHS

Poly-time reduction: ST-CONN to SHORTEST-PATHS

Example 1. ST-CONN poly–time reduces to SHORTEST-PATHS. ⟵ *decision reduces to search*

v	s	1	2	3	4	5	6	t
marked[]	✔	✘	✘	✘	✔	✔	✔	✔

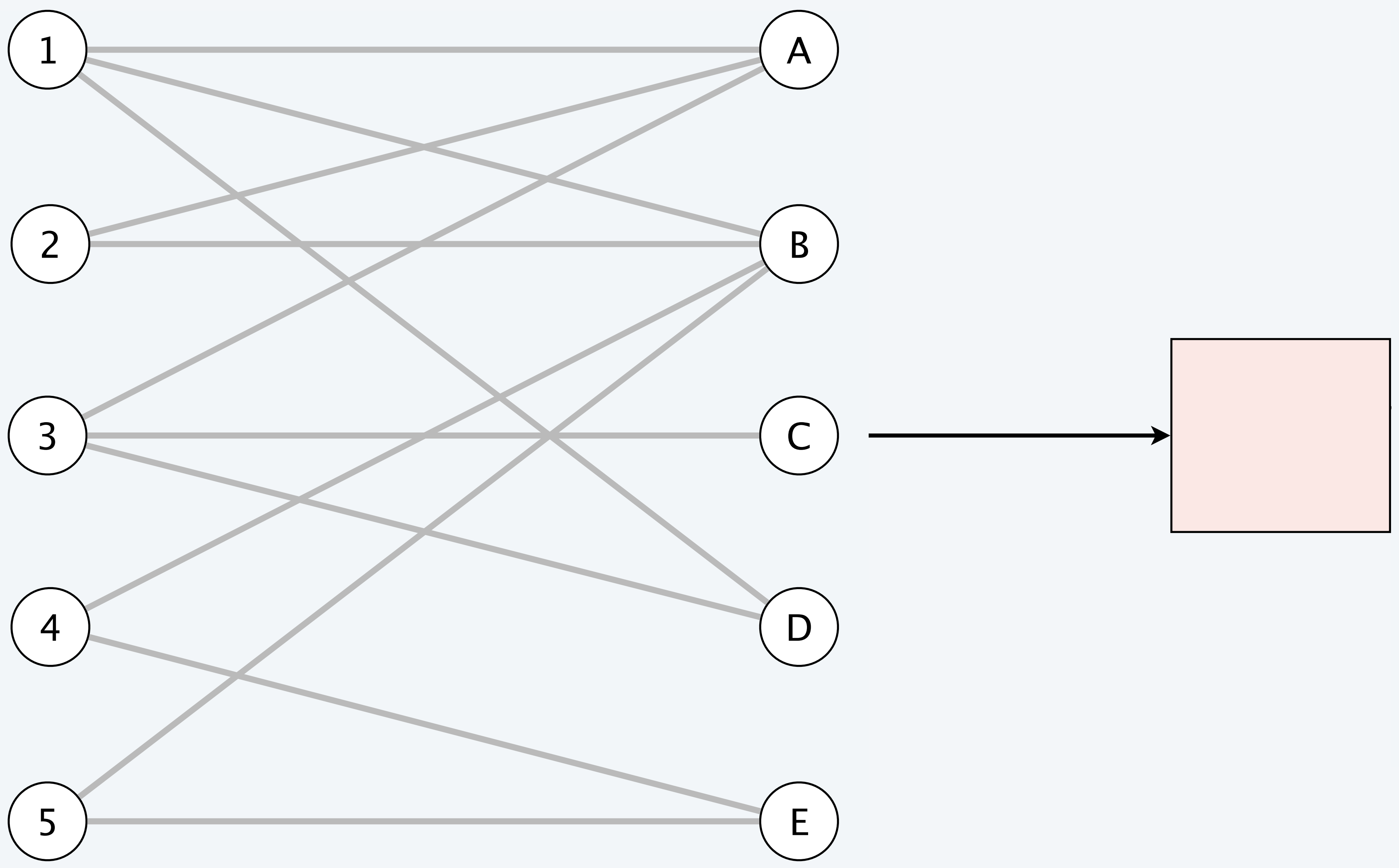


solution of SHORTEST-PATHS

solution of ST-CONN

Poly-time reduction: BIPARTITE-MATCHING to MAXFLOW

Example 2. Bipartite matching poly-time reduces to maxflow.



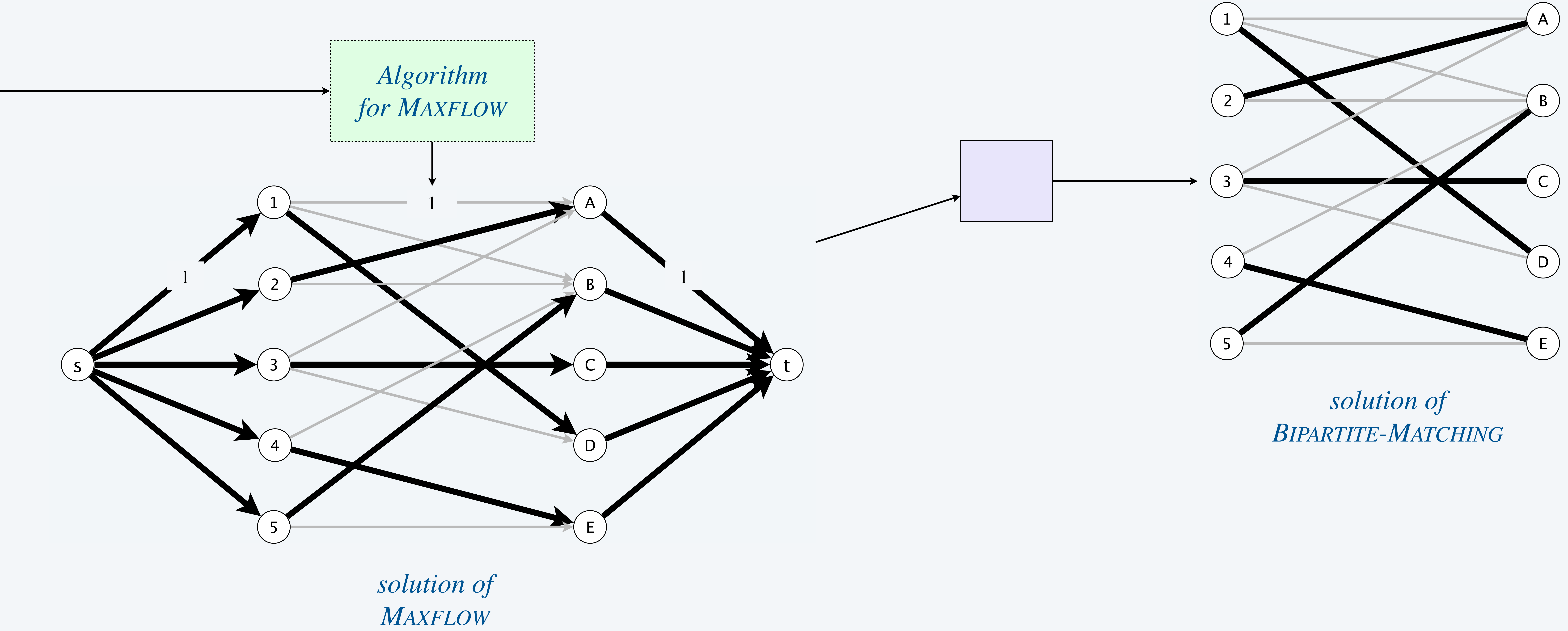
instance of
BIPARTITE-MATCHING

Poly-time reduction: BIPARTITE-MATCHING to MAXFLOW

Example 2. Bipartite matching poly-time reduces to maxflow.

Poly-time reduction: BIPARTITE-MATCHING to MAXFLOW

Example 2. Bipartite matching poly-time reduces to maxflow.



How many vertices and edges are there in the flow network obtained from a V–vertex, E–edge graph via the reduction?

A. $\Theta(V)$ vertices, $\Theta(E)$ edges

B. $\Theta(V)$ vertices, $\Theta(V+E)$ edges

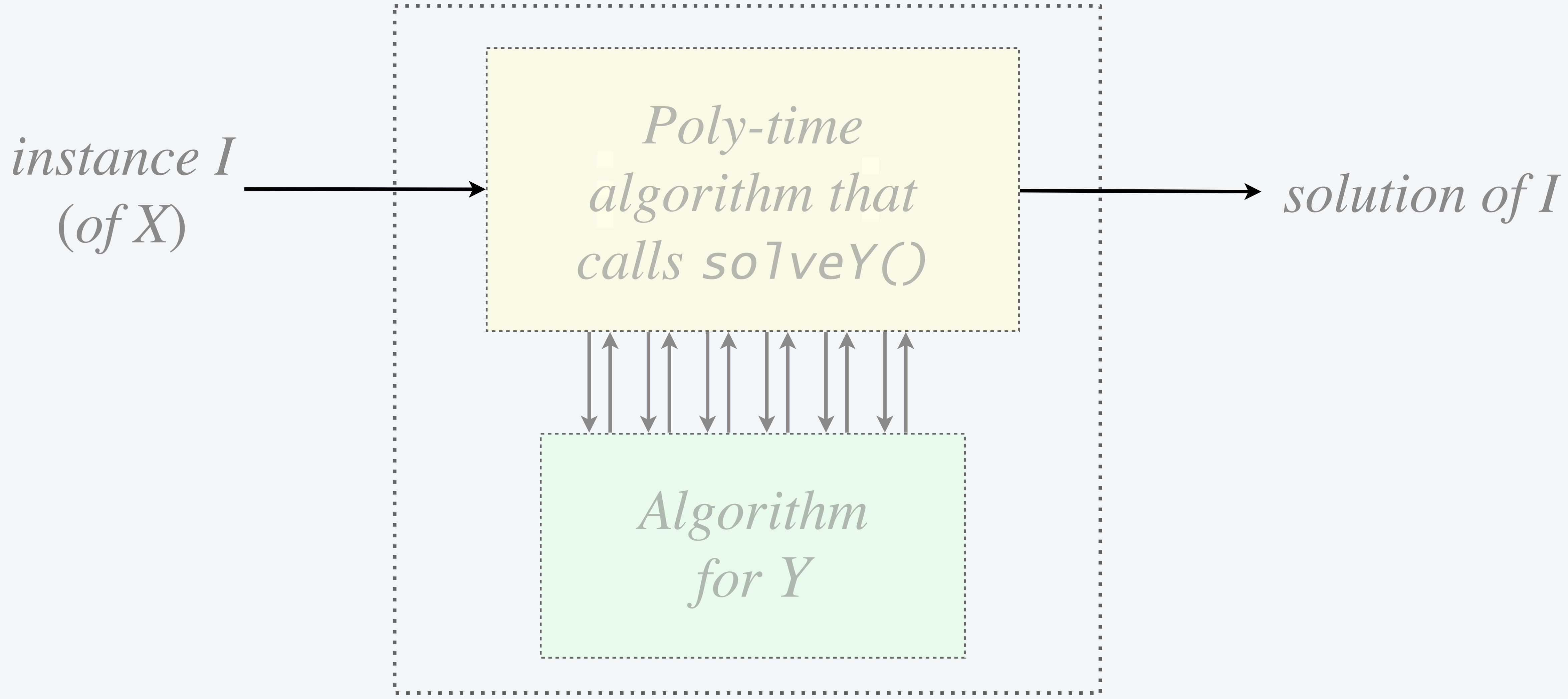
C. $\Theta(V^2)$ vertices, $\Theta(V+E)$ edges

D. $\Theta(V^2)$ vertices, $\Theta(E^2)$ edges

Poly-time reduction (review)

Definition. Problem X poly-time reduces to problem Y if X can be solved with:

- Polynomial number of elementary operations.
- Polynomial number of calls to algorithm for Y.



algorithm for problem X

Common mistake. Confusing *X poly-time reduces to Y* with *Y poly-time reduces to X*.

Intractability: poll 4

Suppose that Problem X poly-time reduces to Problem Y.

Which of the following can we infer?

A. If X can be solved in poly-time, then so can Y.

B. If X cannot be solved in $\Theta(n^3)$ time, Y cannot be solved in poly-time.

C. If Y can be solved in $\Theta(n^3)$ time, then X can be solved in poly-time.

D. If Y cannot be solved in poly-time, then neither can X.

Intractable problems

Q3. Which problems are intractable?

A3. Those with no poly-time algorithm.

Bird's-eye view (counterpoint)

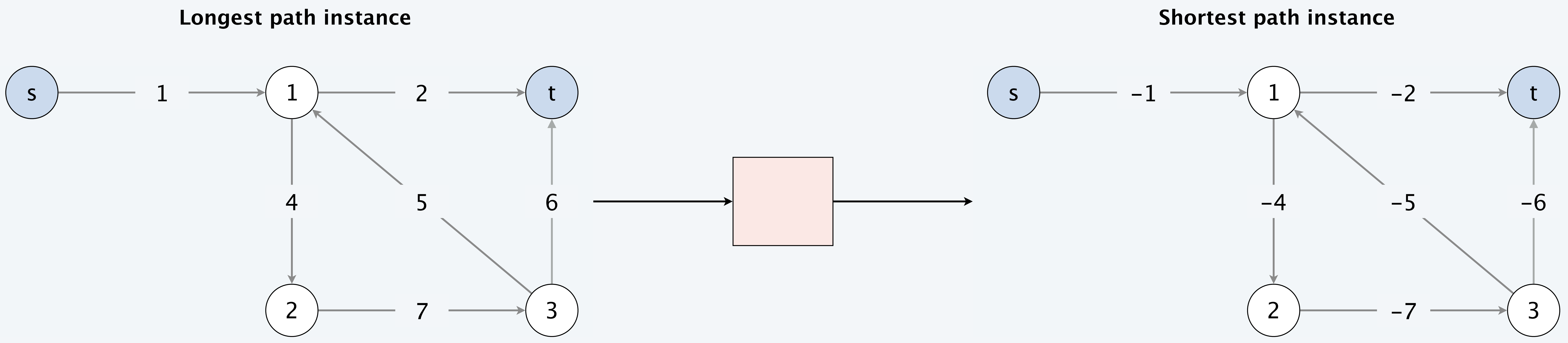
Design strategy. Suppose we can solve problem X efficiently.
Which other problems can we solve efficiently?

Establishing intractability. Suppose problem X is intractable.
Which other problems are also intractable?

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

Poly-time reduction: LONGEST-ST-PATH to SHORTEST-ST-PATH

Example. Longest simple path poly-time reduces to shortest simple path with negative weights.



Conjecture (equivalent to **P ≠ NP**). LONGEST-ST-PATH is intractable.

Conditional conclusion. SHORTEST-ST-PATH with negative weights is intractable.

https://algs4.cs.princeton.edu

INTRACTABILITY

- introduction
- computational problems
- poly-time algorithms
- P vs. NP
- poly-time reductions
- coping with intractability

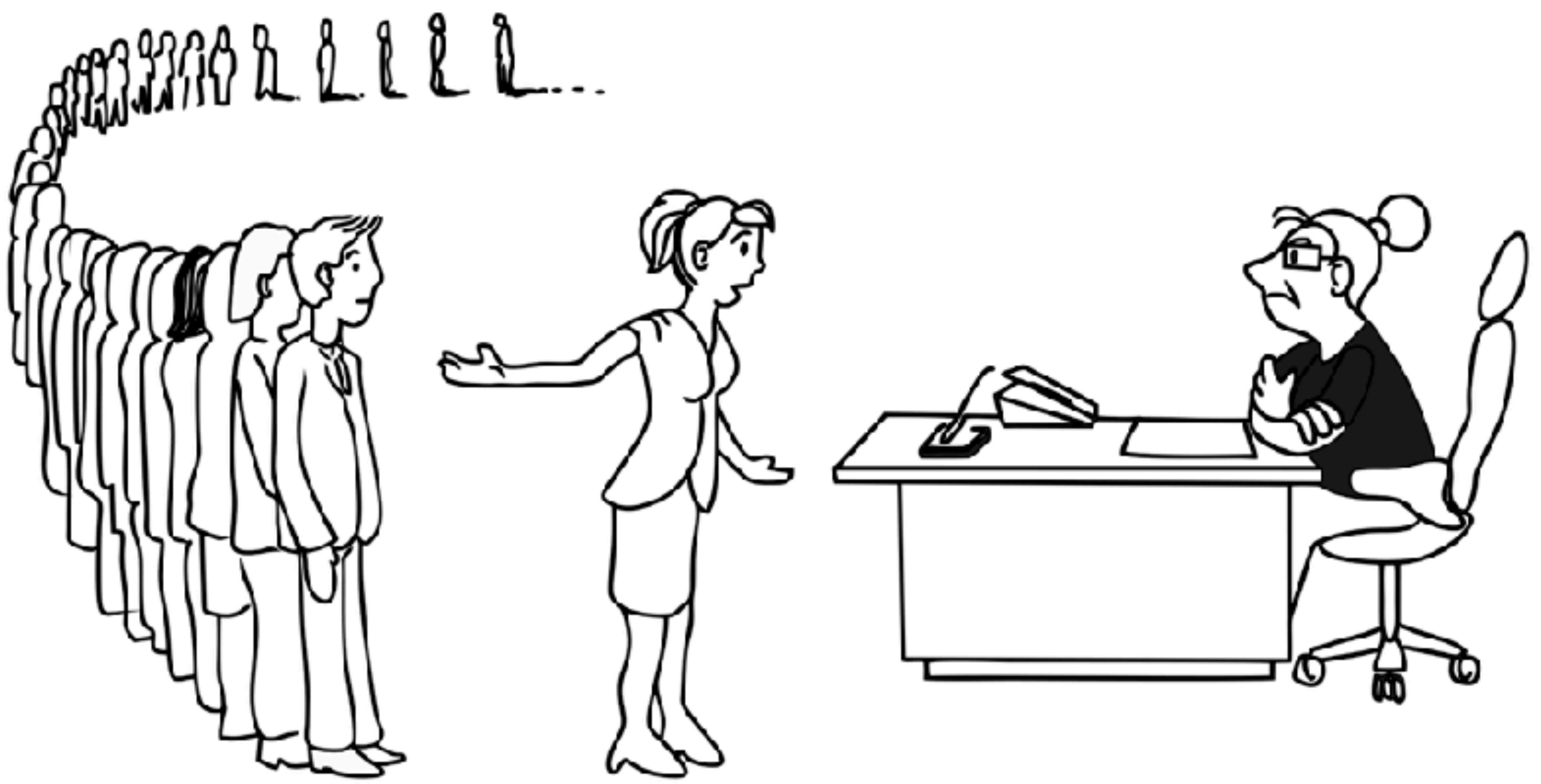
Identifying intractable problems

Step 1. Start with an **NP** problem believed to be intractable (e.g., LONGEST-ST-PATH).

Step 2. Find a poly-time reduction from it to your problem.

"I can't find a poly-time algorithm."

does not know reduction from LONGEST-ST-PATH



"I can't find a poly-time algorithm, but neither can all these famous people."

knows reduction from LONGEST-ST-PATH

Approaches to dealing with intractability

Q. What to do when you find a poly-time reduction from (conjectured) hard problem?

A. Safe to assume intractable: no (worst-case) poly-time algorithm.

Q1. Must your algorithm *always* run fast?

Solve real-world instances. Backtracking, SAT.

Q2. Do you need the *optimal* solution or a *good* solution?

Approximation algorithms. Find slightly suboptimal solutions.

Q3. Can you use the problem's hardness in your favor?

Leverage intractability. Cryptography.

Intractability: poll 5

A program with which of these running times is most likely to be useful in practice?

- **A.** $10^{226} n$
- **B.** n^{226}
- **C.** 1.000000001^{n}
- **D.** $(n!)!$

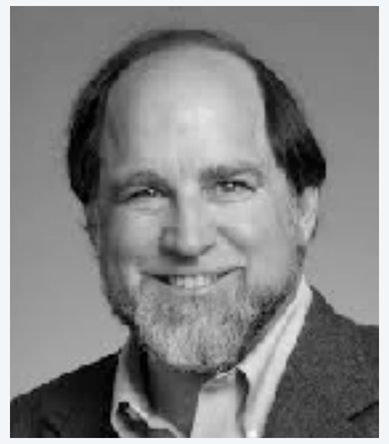
Leveraging intractability: RSA cryptosystem

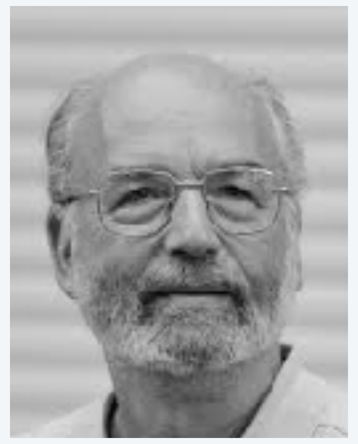
Modern cryptography applications.

- Secure a secret communication.
- Append a digital signature.
- Credit card transactions.
- …

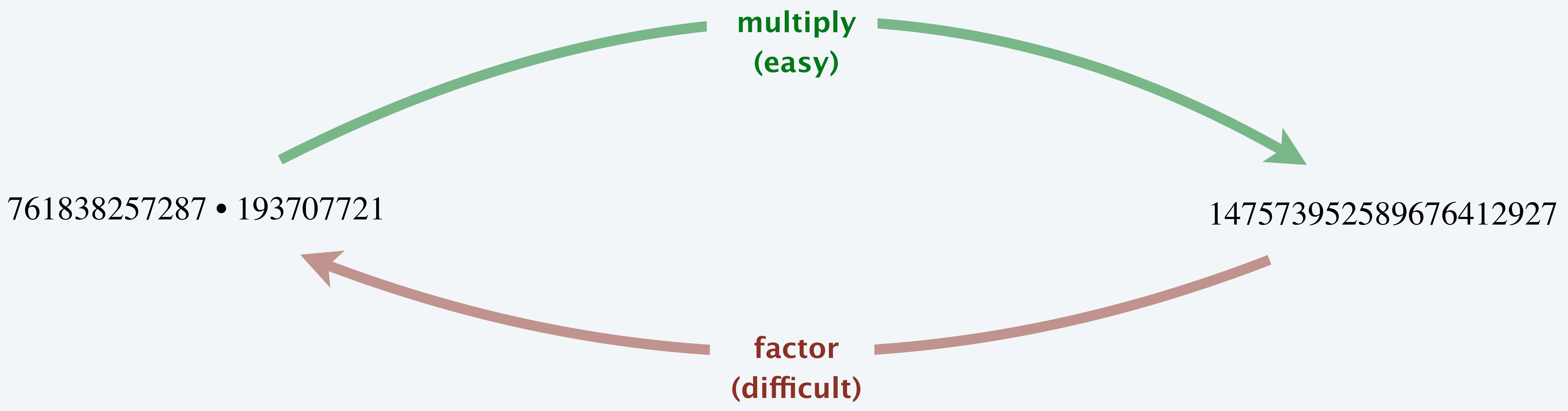
RSA cryptosystem exploits intractability.

- To use: multiply/divide two n–digit integers (easy).
- To break: factor a $2n$–digit integer (intractable?).


Ron Rivest


Adi Shamir


Len Adelman



Summary

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

NP. Set of decision problems verifiable in poly-time (given witness).

Poly-time reduction.

- Algorithm for problem X via
  - Reduction from X to Y, plus
  - Algorithm for Y.
- Intractability of X established via
  - Reduction from intractable Y to X.

Use theory as a guide.

- You will confront (conjectured) intractable problems in your career.
- It is safe to assume that **P** ≠ **NP** and that such problems are intractable.
- Identify these situations and proceed accordingly.

Credits

image	source	license
Gears	Adobe Stock	Education License
Finding a Needle in a Haystack	Basic Vision	
Galactic Computer	Adobe Stock	Education License
Taylor Swift Caricature	Cory Jensen	CC BY-NC-ND
Fans in a Stadium	Adobe Stock	Education License
P and NP cookbooks	Futurama S2E10	
Homer Simpson and P = NP	Simpsons	
Archimedes, Lever, and Fulcrum	unknown	
COS Building, Western Wall	Kevin Wayne	
Garey–Johnson Cartoon Updated	Stefan Szeider	CC BY 4.0
Cartoon of Turing Machine	Tom Dunne	
Warning sign	Adobe Stock	Education License
Glass with water	Adobe Stock	Education License
John Nash	Wikimedia	CC BY-SA 3.0

A final thought

“ *Now my general conjecture is as follows: for almost all sufficiently complex types of enciphering, […] the mean key computation length increases exponentially with the length of the key […].*

The nature of this conjecture is such that I cannot prove it […].
Nor do I expect it to be proven. ”

— John Nash