Computational Complexity

Harry Buhrman

(buhrman@cwi.nl)

Florian Speelman

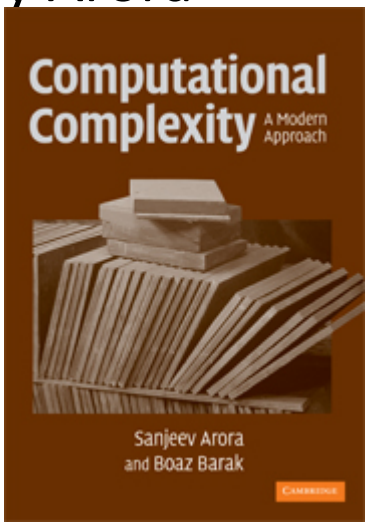
(F.Speelman@cwi.nl)

Algorithms & Complexity group

at CWI and UvA

Course requirements

- Computational Complexity: A Modern Approach by Arora & Barak (http://www.cs.princeton.edu/theory/complexity/)
- lectures (hoorcollege)
  - Monday 17:00-19:00, Wednesday 15:00-17:00
  - Question: Exchange werkcolllege and hoorcollege on Wednesday?
- werkcollege:
  - Wednesday 13:00–15:00 (Thursday 15:00-17:00)
- http://complexity.turing-machine.nl
- Compulsory: hand in exercises every week on Monday
- Final exam

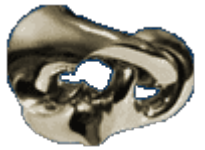


Grade

- Hand in exercises on Monday the week after they were distributed
- Final grade exercises is average of obtained grades. We will drop the lowest grade
- Cooperation is allowed, always write down solutions on your own
- Final grade = average of grade final exam and grade exercises

P versus NP problem

P versus NP



- One of the seven millennium prize problems
- “In the case of the P versus NP problem and the Navier-Stokes problem, the SAB will consider the award of the Millennium Prize for deciding the question in either direction.”

- P not equal NP $\Rightarrow$ 1 million $
- P equal NP $\Rightarrow$ 1 million $

Main characters: Algorithms

Algorithm

- Algorithm is like a cooking recipe

Algorithm

- Algorithm is like a cooking recipe

Recipe
CHOCOLATE CAKE

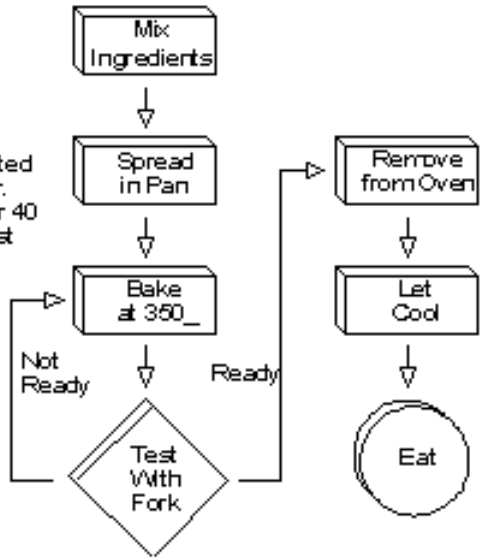
4 oz. chocolate
1 cup butter
2 cups sugar
3 eggs
1 tsp. vanilla
1 cup flour

Melt chocolate and butter. Stir sugar into melted chocolate. Stir in eggs and vanilla. Mix in flour. Spread mix in greased pan. Bake at 350_ for 40 minutes or until inserted fork comes out almost clean. Cool in pan before eating.

Program Code

Declare variables:
chocolate eggs mix
butter vanilla
sugar flour

mix = melted ((4*chocolate) + butter)
mix = stir (mix + (2*sugar))
mix = stir (mix + (3*eggs) + vanilla)
mix = mix + flour
spread (mix)
While not clean (fork)
bake (mix, 350)



Algorithm

- algorithm is like a cooking recipe

- input
- computation
  - steps (1 time unit)
- output

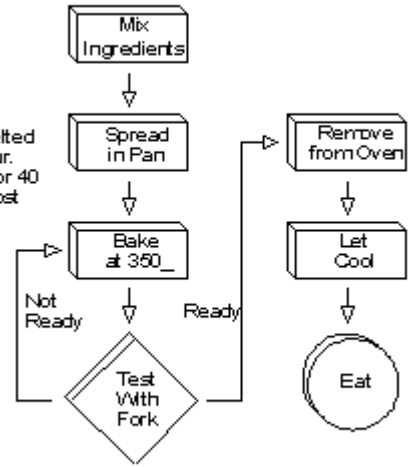
Recipe
CHOCOLATE CAKE

4 oz. chocolate	3 eggs
1 cup butter	1 tsp. vanilla
2 cups sugar	1 cup flour

Melt chocolate and butter. Stir sugar into melted chocolate. Stir in eggs and vanilla. Mix in flour. Spread mix in greased pan. Bake at 350_ for 40 minutes or until inserted fork comes out almost clean. Cool in pan before eating.

Program Code

```
Declare variables:
    chocolate   eggs        mix
    butter      vanilla
    sugar       flour
mix = melted ((4*chocolate) + butter)
mix = stir (mix + (2*sugar))
mix = stir (mix + (3*eggs) + vanilla)
mix = mix + flour
spread (mix)
While not clean (fork)
bake (mix, 350)
```



Example

Greatest Common Divisor (GCD)

Slow Algorithm

a=21 b=13

step 1 i=13 $13 \nmid 21$

step 2 i=12 $12 \nmid 21$

step 3 i=11 $11 \nmid 21$

step 4 i=10 $10 \nmid 21$

⋮

step 7 i=7 $7 \nmid 13$

⋮

step 13 i=1

output 1

```
slow-gcd(a, b)
   i = min(a,b)
      while  i ∤ a  or  i ∤ b
         i := i -1
output  i
```

Analysis of Algorithm

Analysis of alg. is preparation time of recipe

CARL SAGAN'S APPLE PIE

1 universe
1 9" pie shell
6 cups sliced apples
3/4 cup sugar
1/2 cup brown sugar
2 tbsp all-purpose flour
1/2 tsp cinnamon
1/8 tsp nutmeg
1/2 cup all-purpose flour
3 tbsp butter

Preparation time:
12-20 billion years

Servings:
8

Remember -
"If you want to make an apple pie from scratch, you must first create the universe."
–Carl

Preheat oven to 375 F. Make the universe as usual.

Place apples in a large bowl. In a smaller bowl, mix together sugar, 2 tbsp flour, cinnamon, and nutmeg. Sprinkle mixture over apples. Toss until evenly coated. Spoon mixture into pie shell.

In a small bowl mix together 1/2 cup flour and brown sugar. Add butter until mixture is crumbly. Sprinkle mixture over apples. Cover loosely with aluminum foil.

Bake in preheated oven for 25 minutes. Remove foil and bake another 30 minutes, or until golden brown.

Slow Algorithm

a=21 b=13

step 1 i=13 $13 \nmid 21$

step 2 i=12 $12 \nmid 21$

step 3 i=11 $11 \nmid 21$

step 4 i=10 $10 \nmid 21$

⋮

step 7 i=7 $7 \nmid 13$

⋮

step 13 i=1

output 1

```
slow-gcd(a, b)
   i = min(a,b)
      while  i ∤ a  or  i ∤ b
         i := i -1
output  i
```

if gcd(a,b)=1 then

algorithm uses min(a,b) steps

Better Algorithm

Euclidean Algorithm

Greatest Common Divisor (GCD)

step 0 a=21 b=13

step 1 a=13 b= 21 mod 13 = 8

step 2 a=8 b= 13 mod 8 = 5

step 3 a=5 b= 8 mod 5 = 3

step 4 a=3 b= 5 mod 3 = 2

step 5 a=2 b= 3 mod 2 = 1

step 6 a=1 b= 2 mod 1 = 0

output 1

```
function gcd(a, b)
    while b ≠ 0
        t := b
        b := a mod b
        a := t
    output a
```

Analysis GCD-Algorithm

- worst case number of steps?

```
function gcd(a, b)
    while b ≠ 0
        t := b
        b := a mod b
        a := t
    output a
```

Theorem

alg. terminates in

2log (m) +1 steps

m=max(a,b)

Proof:

every second step a is

at least halved

step 0	a=21	b=13
step 1	a=13	b= 21 mod 13 = 8
step 2	a=8	b= 13 mod 8 = 5
step 3	a=5	b= 8 mod 5 = 3
step 4	a=3	b= 5 mod 3 = 2
step 5	a=2	b= 3 mod 2 = 1
step 6	a=1	b= 2 mod 1 = 0

Complexity

- Euclid: 2log (m) +1 m =max(a,b)
- Slow: m' m'= min(a,b)
- length of the input: log(a) + log(b) = n

Euclid: $O(n)$ Slow: $2^{O(n)}$

- Euclid exponentially faster than slow!
- Complexity of computational problem is running time of the best algorithm

Computation & Complexity

- Computational problem:
  - INPUT 

OUTPUT
  - Example: a,b output gcd(a,b)
- Complexity:
  - Number of computation steps needed for "best" algorithm
  - function of the input size

Complexity

- Determine the complexity of a computational problem:
  - Upper bound: construct algorithm
  - Lower bound: **any** algorithm needs this many steps
- Ideally upper bound = lower bound

functions of the input size

Complexity of gcd problem

- Euclid's algorithm runs in O(n) steps
- Can we devise a faster algorithm?
- Not really: any algorithm has to read the whole input: requires n steps
  - Upper Bound: O(n)
  - Lower Bound: n
- Complexity of gcd is linear.

Complexity Class P

Feasible Problems: P

- Feasible or efficient algorithms run in polynomial time: n^c (some c)
- Complexity Class **P :**
  - All the problems that have feasible algorithms
- Example:
  - Linear Programming
  - Network Flow Problems
  - Shortest Path

For these problems upper bound is "close" to lower bound: at most polynomial far off.

Another problem

Satisfiablity

Satisfiability

- variables $x_1 \ldots x_n$
- Clause $C_1 \ldots C_m$ $\quad C_l = (x_i \vee x_j \vee \overline{x_k})$
- formula $\phi(x_1 \ldots x_n) = C_1 \wedge \ldots \wedge C_m$
- exist $\alpha_1 \ldots \alpha_n$ $\quad \alpha_i \in \{T, F\}$
- such that $\phi(x_1 = \alpha_1 \ldots x_n = \alpha_n) = T$

Example

$$\overset{F}{(\overline{x_1}} \vee \overset{F}{\overline{x_2})} \wedge \overset{T}{(x_1} \vee \overset{T}{x_3)} \wedge \overset{T}{(x_2} \vee \overset{F}{\overline{x_3})} \wedge \overset{F}{(\overline{x_1}} \vee \overset{T}{x_3)}$$

$$\downarrow \qquad \downarrow \qquad \downarrow \qquad \downarrow$$

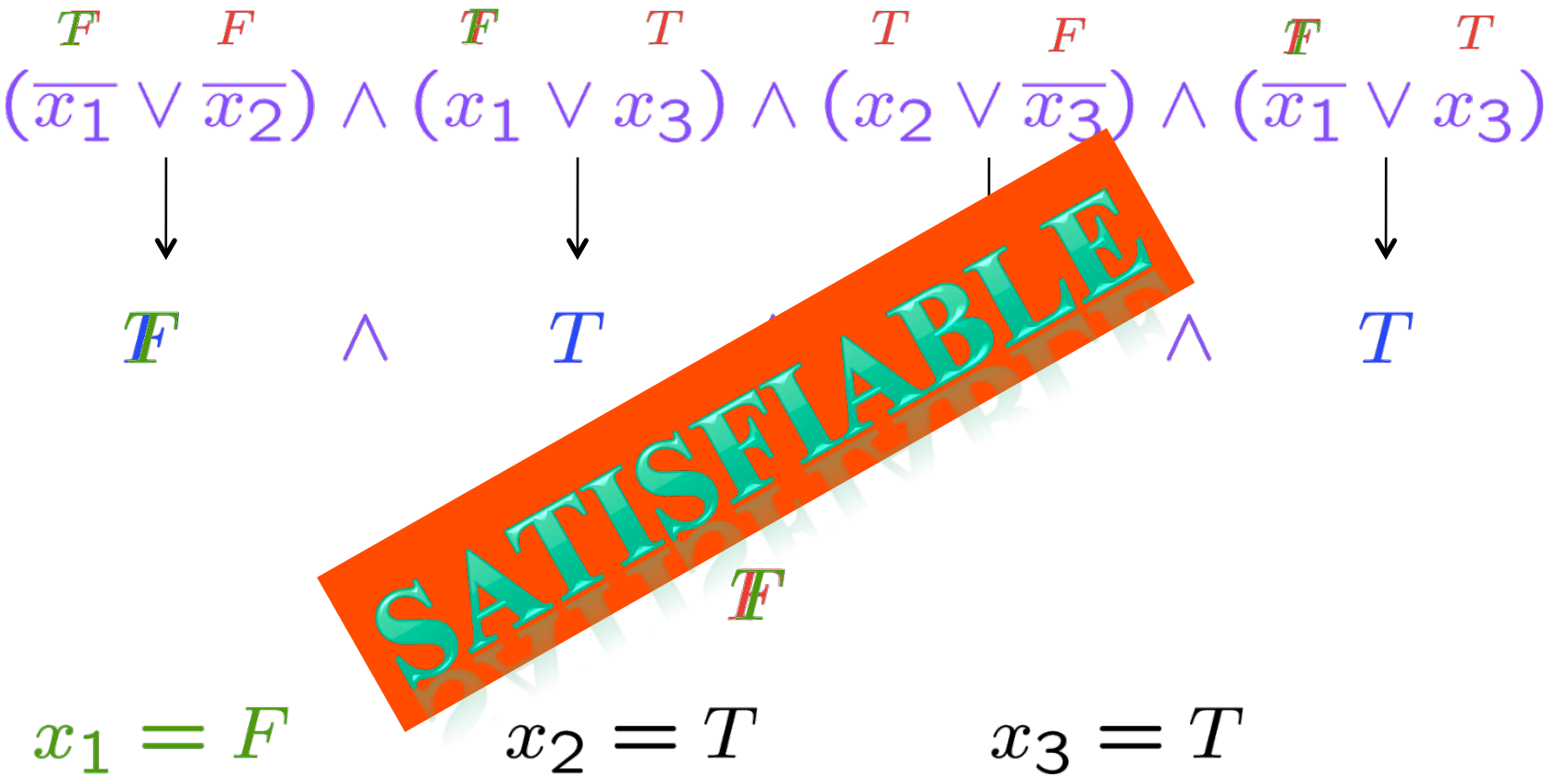
$$F \wedge T \wedge T \wedge T$$

$$\downarrow$$

$$F$$

$$x_1 = T \qquad x_2 = T \qquad x_3 = T$$

Example


F F F T T F F T
$(\overline{x_1} \vee \overline{x_2}) \wedge (x_1 \vee x_3) \wedge (x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_3)$
F ∧ T ∧ T
SATISFIABLE
F
$x_1 = F$
$x_2 = T$
$x_3 = T$

Satisfiability

- variables $x_1 \ldots x_n$
- Clause $C_1 \ldots C_m$ $\quad C_l = (x_i \vee x_j \vee \overline{x_k})$
- formula $\phi(x_1 \ldots x_n) = C_1 \wedge \ldots \wedge C_m$
- exist $\alpha_1 \ldots \alpha_n$ $\quad \alpha_i \in \{T, F\}$
- such that $\phi(x_1 = \alpha_1 \ldots x_n = \alpha_n) = T$

$$SAT = \{\phi \mid \phi \text{ is satisfiable}\}$$

simple algorithm: try all 2^n assignments

Unknown Complexity

- It is hard to determine the complexity of many problems
- Example:
  - Is this formula satisfiable? **SAT**
  - Traveling Salesman Problem. **TSP**
- Lower Bound: n
- Upper Bound: 2^n

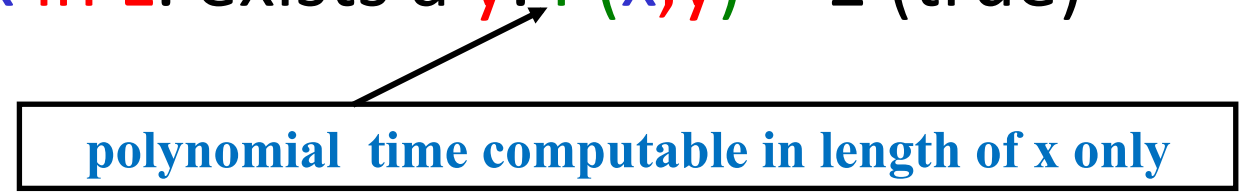
Best Known!

Complexity Class NP

NP

- complexity class NP
  - polynomial time to check solution

- x in L: exists a y: P(x,y) = 1 (true)



SAT in NP

φ is satisfiable

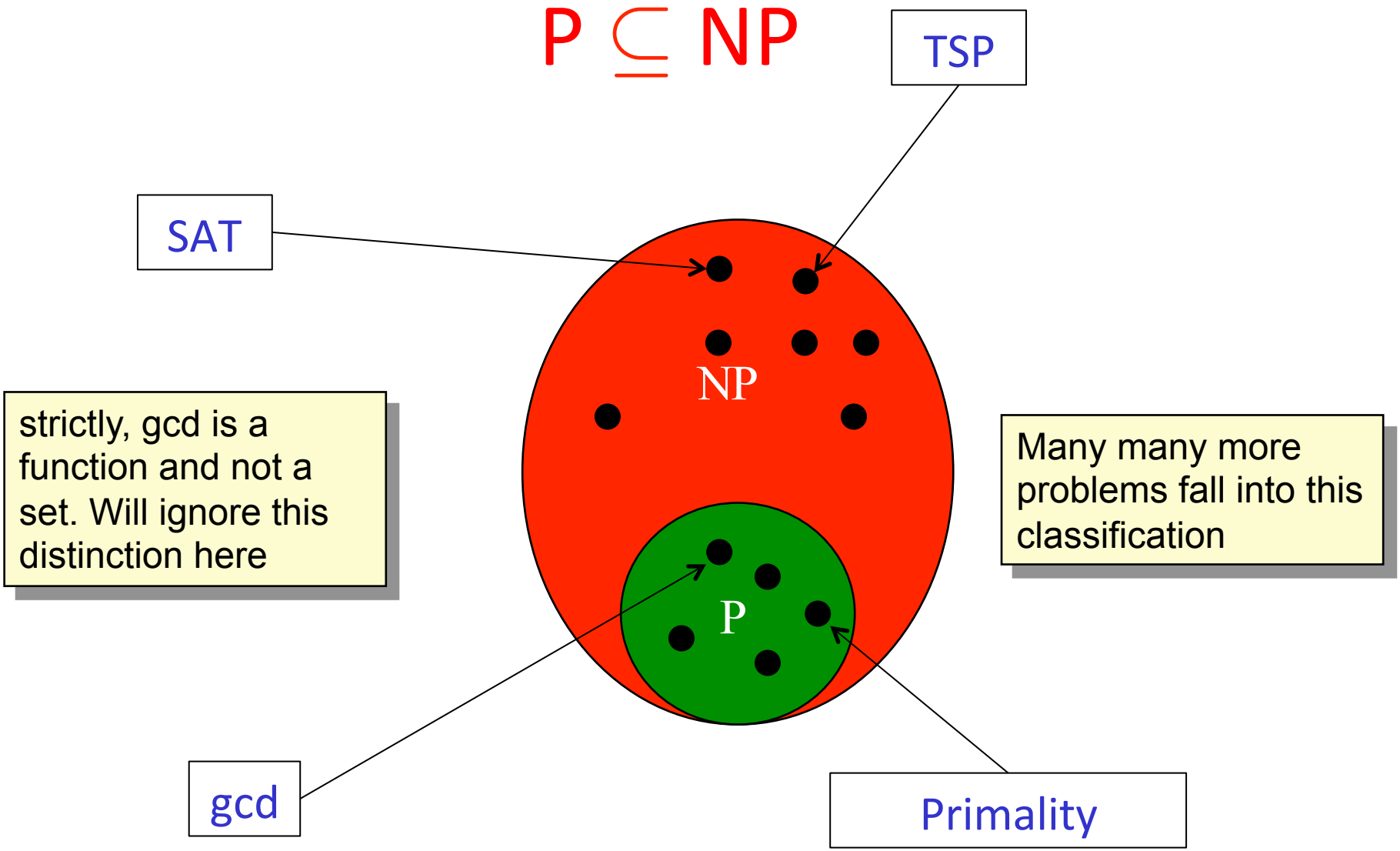
$\exists\, \alpha : \varphi(\alpha) = \text{True}$

P & NP

- complexity class NP
  - easy to check solution
  - polynomial time check
  - easy to check assignment is satisfiable

versus

- complexity class P
  - easy to find solution
  - decide in polynomial time
  - compute in polynomial time gcd(a,b)


P ⊆ NP
TSP
SAT
NP
P
strictly, gcd is a function and not a set. Will ignore this distinction here
Many many more problems fall into this classification
gcd
Primality

Reductions & Completeness

reduction

$$A \leq_T^p B$$

compute A in poly-time with B as free subroutine

"A is computationally not harder than B"

"if B in P then A in P"

C is NP-complete

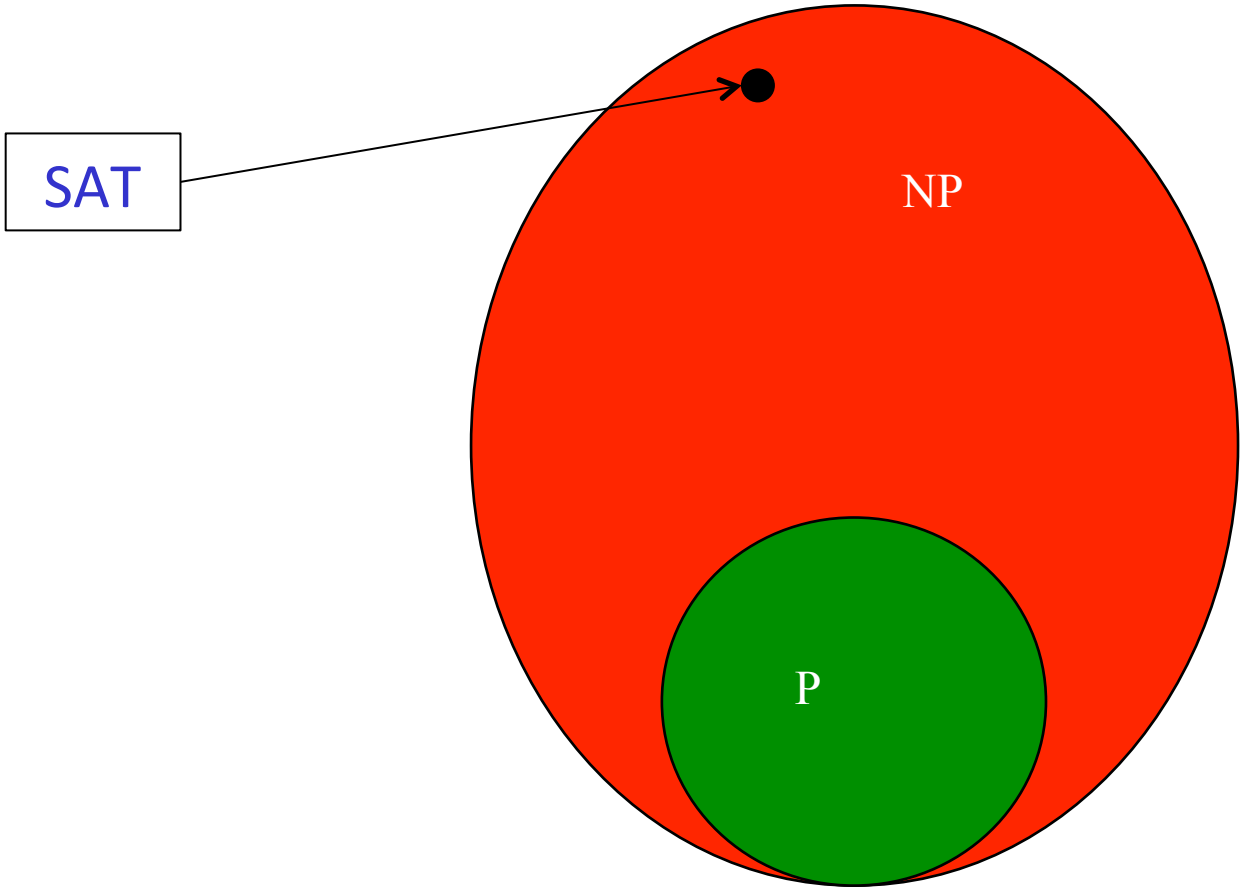
- $C \in NP$
- all $A \in NP$: $A \leq_T^p C$

Theorem [Cook-Levin'71]

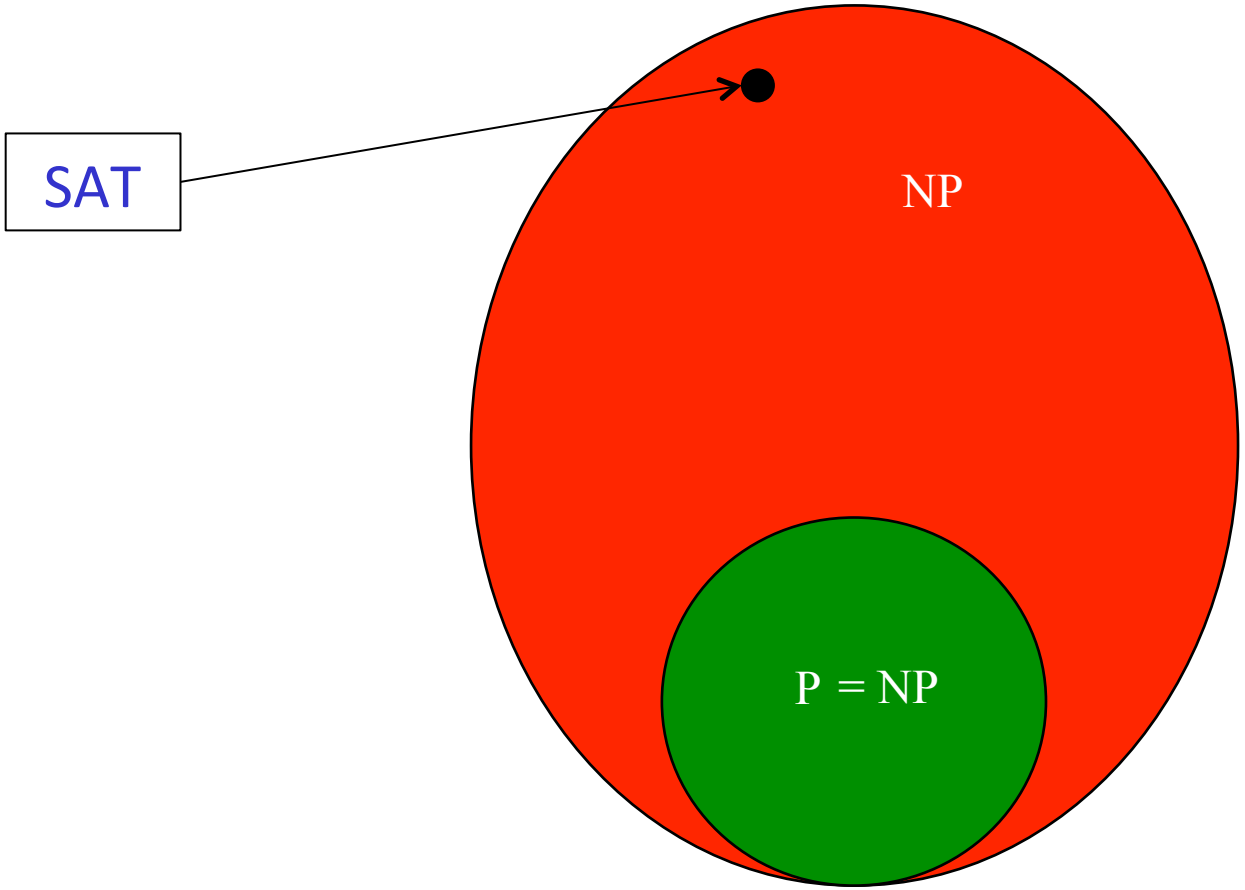
- SAT, TSP, many others NP-complete
- SAT in P $\Leftrightarrow$ P=NP

P versus NP

P ≠ NP



P = NP



P versus NP Question

- P = NP?
- widely believed that $P \neq NP$
- how to show this is true?
  - Prove better lower bounds for existing problems like SAT
  - Construct problem in NP with super polynomial lower bound

Lower Bounds

- Construct D $\in$ NP
- no poly-time algorithm solves D
  - for every poly time algorithm M exists a string x such that:
    - M(x) = 1 & x $\notin$ D or
    - M(x) = 0 & x $\in$ D

$\Rightarrow$ D not in P

D $\leq_T^p$ SAT $\Rightarrow$ SAT not in P

Diagonalization

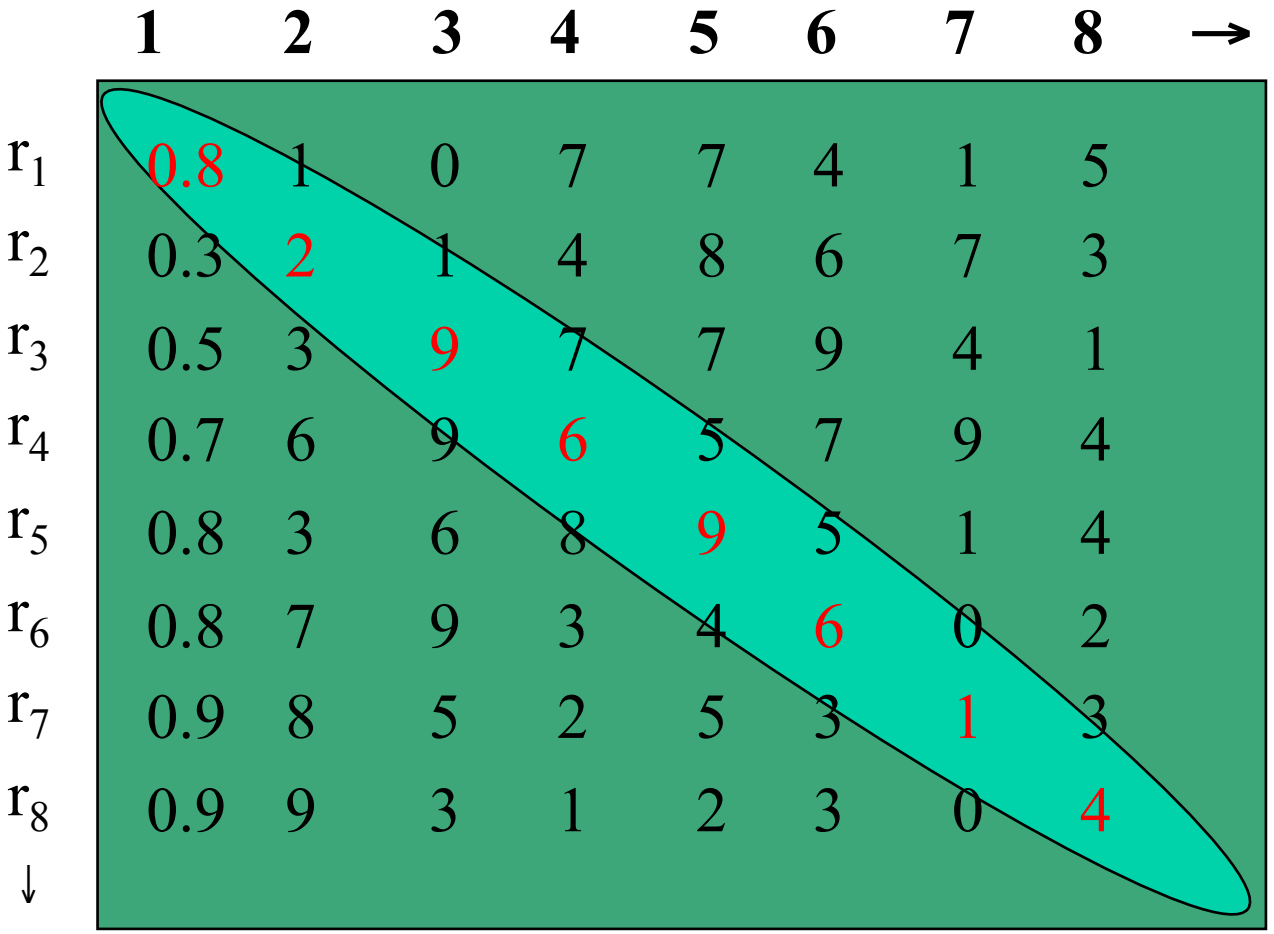
How big are the reals ?

- Cantor showed $\mathbb{R}$ not enumerable

- diagonalization
  - given an enumeration of the reals
  - construct real number d not in the enumeration

Diagonalization

reals in some enumeration

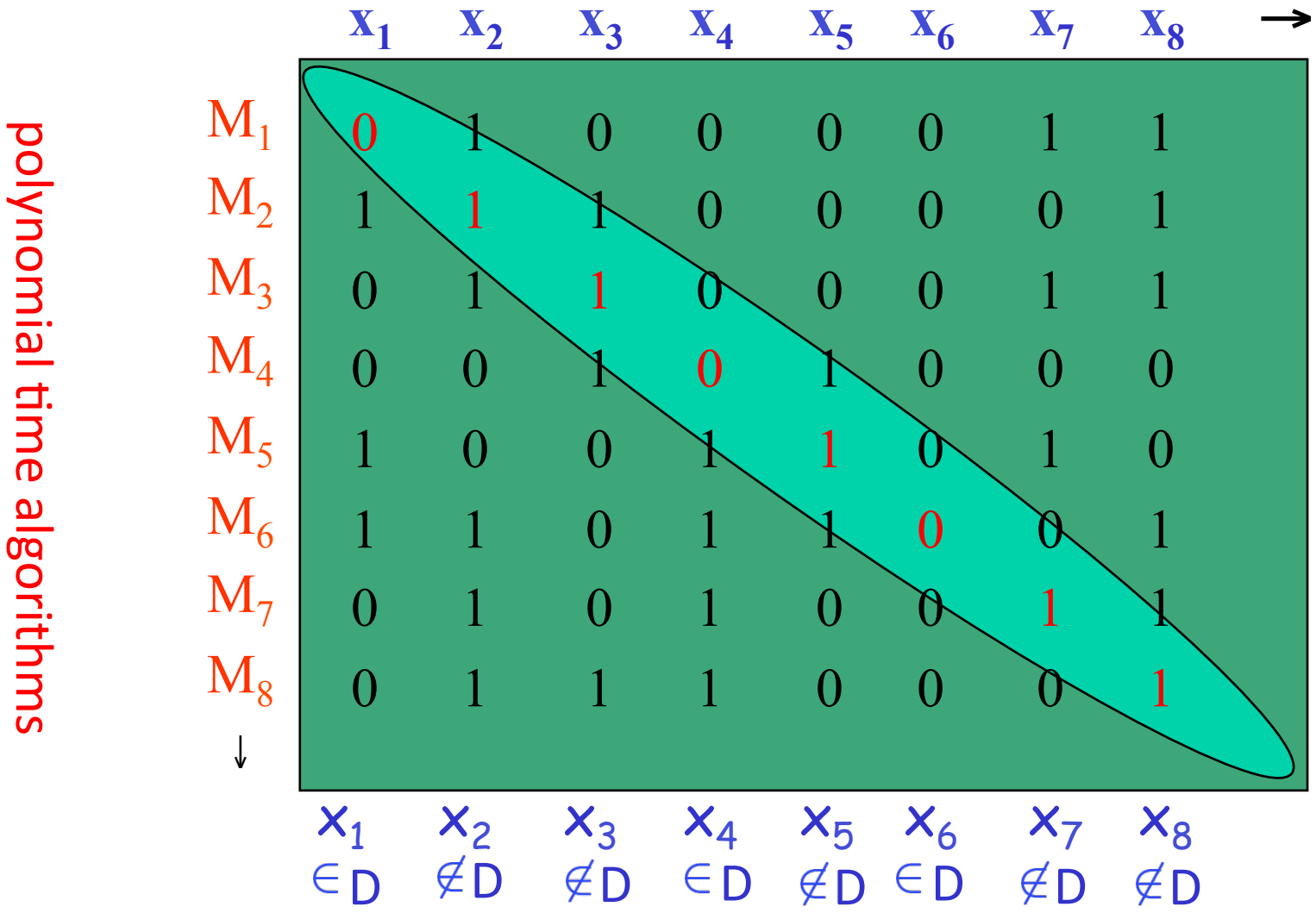


d= 0.9 3 0 7 0 7 2 5 ⋯

i^{th} digit of d is i^{th} entry of diagonal +1 mod 10

Diagonalizing out of P

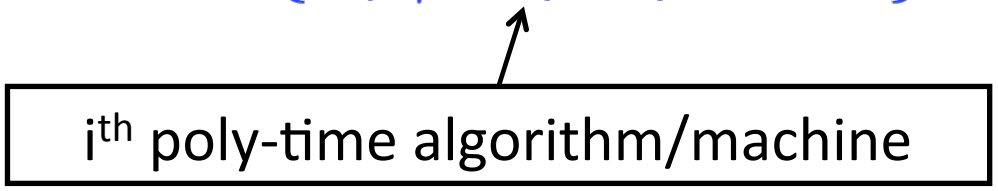
Diagonalization (2)



x_i in D if and only if $M_i(x_i)=0$

Diagonal Language

$$D = \{x_i \mid M_i(x_i) = 0\}$$



D $\notin$ P, every poly-time machine errs on some input

D $\in$ NP ?? probably not, but

D $\in$ time($n^{\log n}$), quasi polynomial time

with more time can compute more

More Bad News

- Relativization (Oracles):
  - Exists oracle A: $\mathbf{P}^A = \mathbf{NP}^A$
  - (Exists oracle B: $\mathbf{P}^B \neq \mathbf{NP}^B$)

Proof technique should not relativize

Diagonalization and most other techniques we know relativize


Space Complexity
to boldly go where no man has gone before

Space Complexity

- Time of a computation not only resource that matters
- Space or memory the computer uses
- L: logarithmic space usage
  - models web applications
- PSPACE: polynomial space usage
  - natural class with natural complete problems

Reuse Space

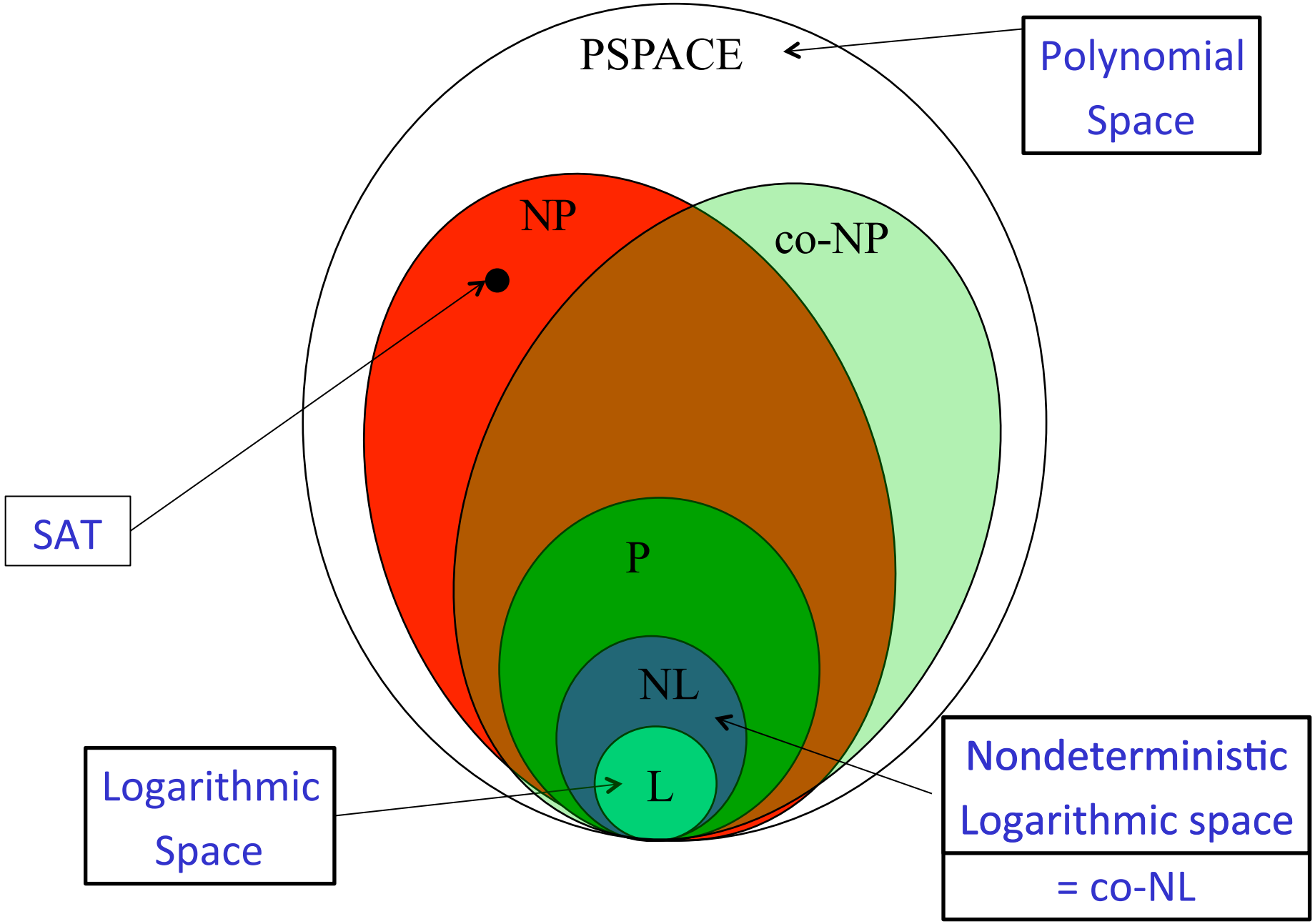
- Space-s(n) computations may run for $2^{s(n)}$ steps
  - if it runs longer it is in a loop and will never stop.
- PSPACE contains P and NP.
- L is contained in P
- NL: non-deterministic LOGSPACE

Some Space Theorems

- NL in P
- NL is closed under complementation

  [Immerman–Szelepcsényi'87]
- NL in DSPACE(log^2 n)

  [Savitch'70]


PSPACE
Polynomial Space
NP
co-NP
SAT
P
NL
L
Logarithmic Space
Nondeterministic Logarithmic space
= co-NL

Open Questions

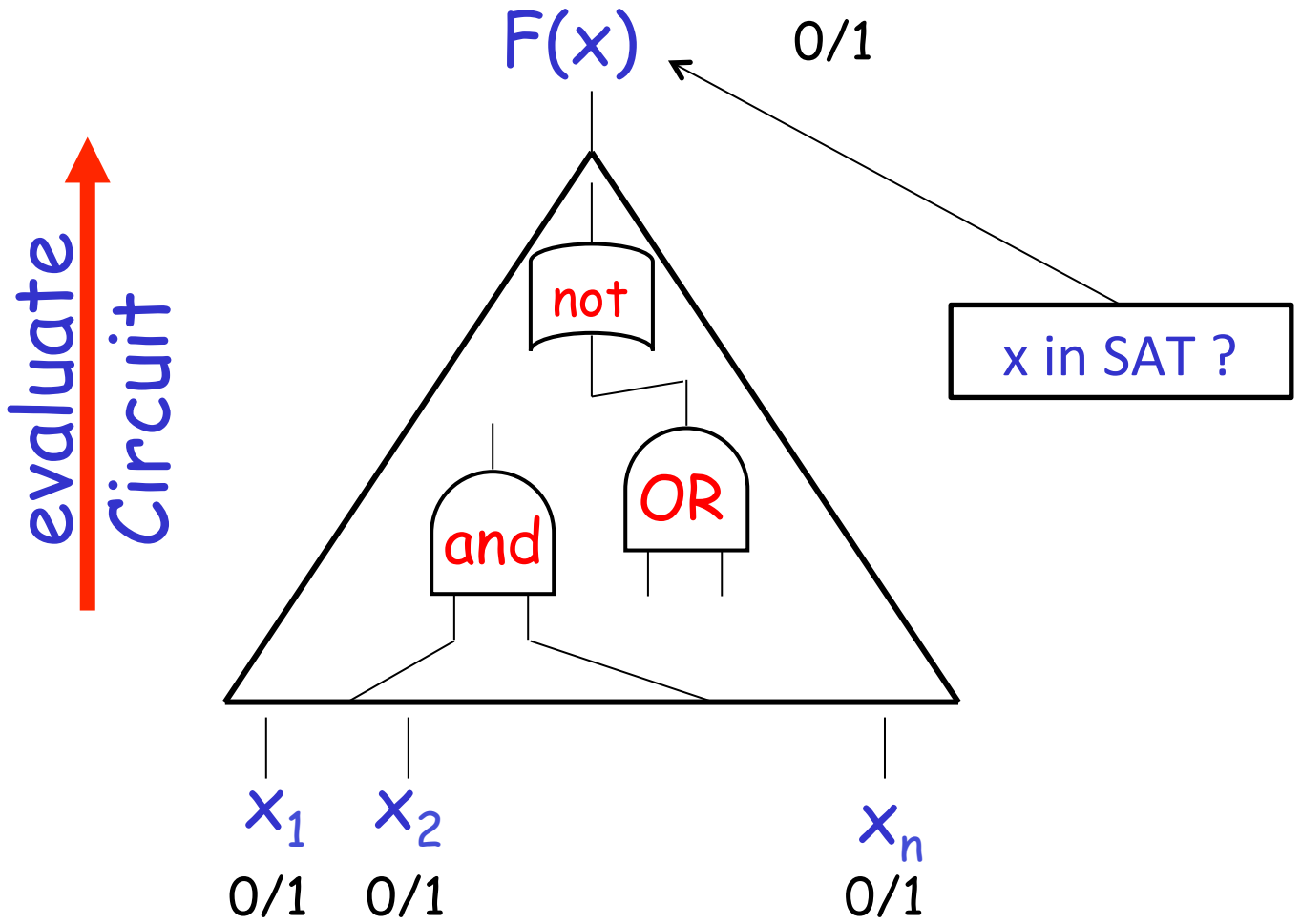
- Besides P versus NP
- L versus NL
- L versus P
- L versus NP
- P versus PSPACE
- More refinements and open embarrassing open problems

Try something easier

- Study weaker models of computation and develop new lower bound techniques
  - Circuits with small depth
  - Monotone circuits
  - Decision Trees
  - Branching Programs
- The weaker the model the better the lower bounds!

Simple model:
Circuits

Circuit Model of Computation



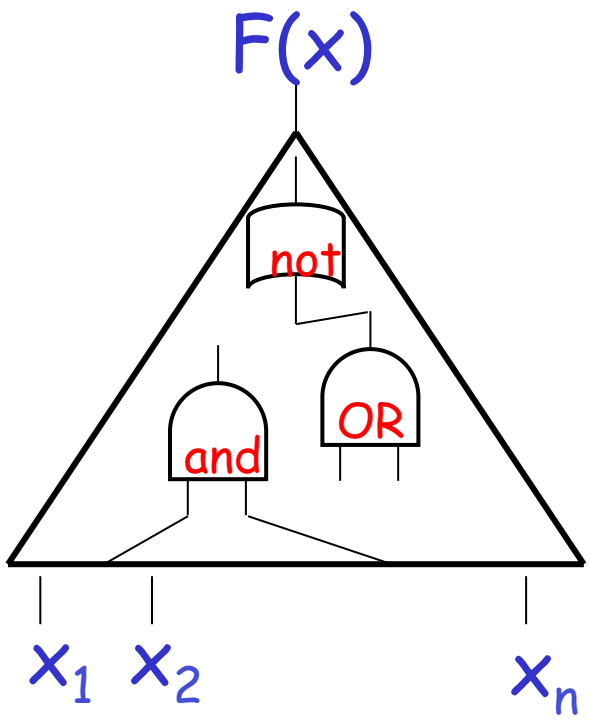
Size of the Circuit

1. most important:

   number of gates

2. Depth of the circuit

   Parallel time of computation



Constant Depth

depth is constant

size is polynomial

AC^0

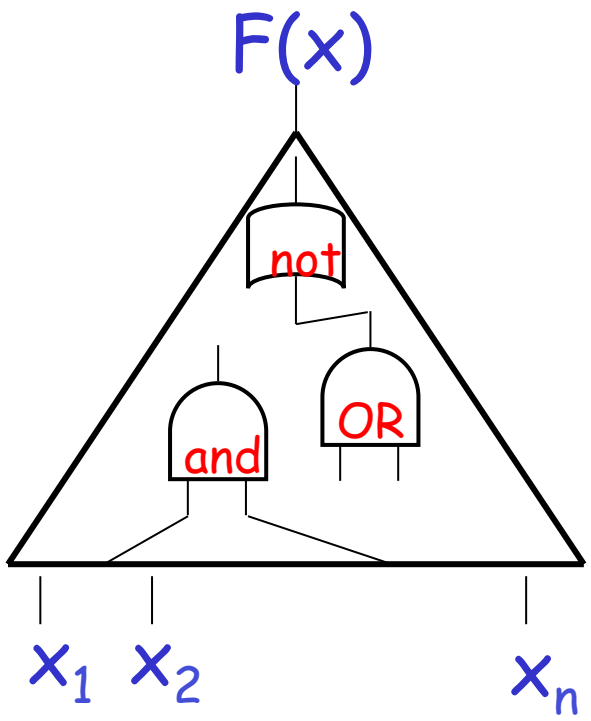
compute parity:

$F(x) = x_1 + x_2 + \cdots + x_n \bmod 2$

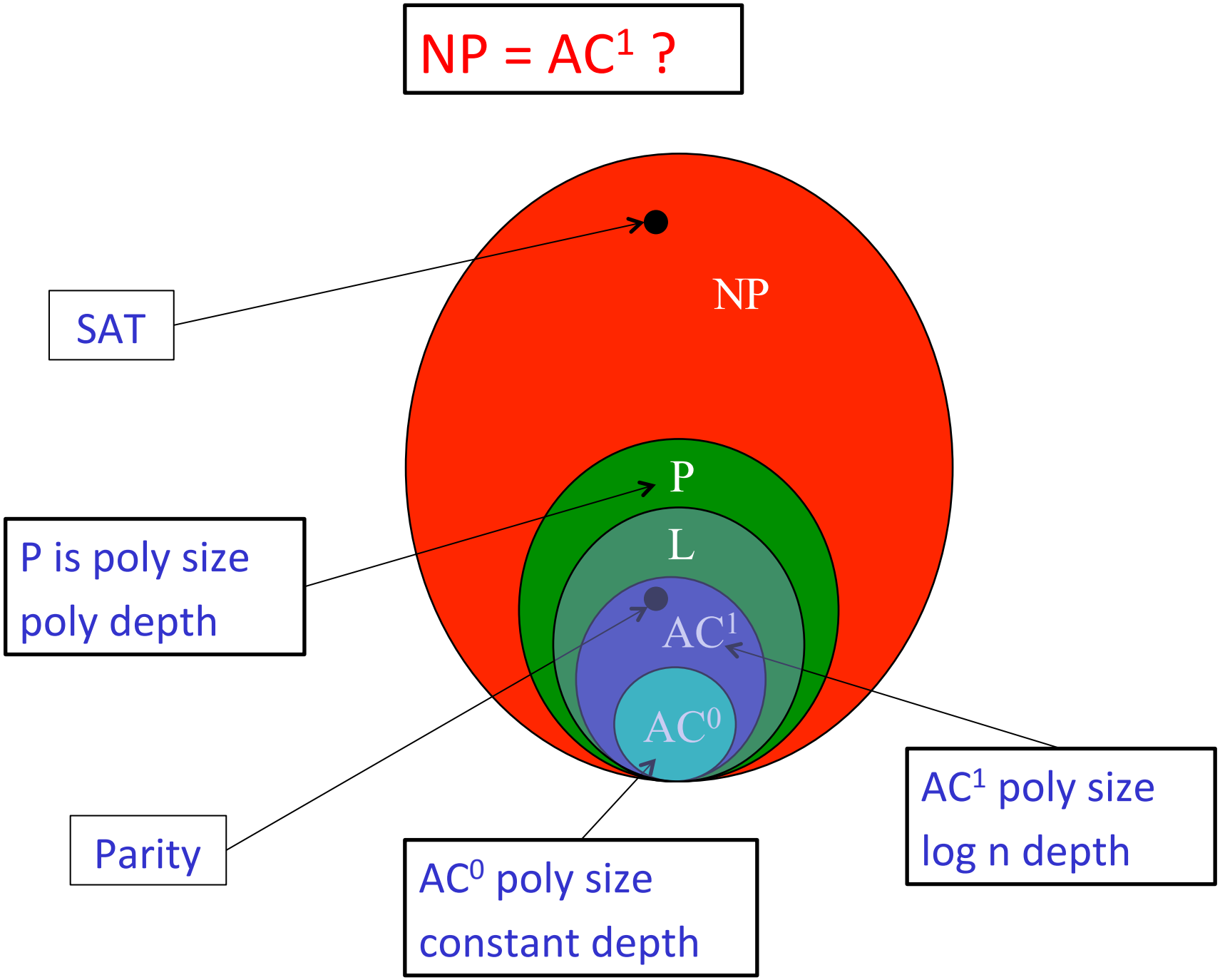
Theorem

parity requires $2^{n^{1/d}}$ size circuits

of depth d



Note: d = log n bound is meaningless


NP = AC1 ?
NP
P
L
AC1
AC0
SAT
P is poly size
poly depth
Parity
AC0 poly size
constant depth
AC1 poly size
log n depth

natural proofs another hurdle?

- proof technique that shows parity not in AC^0 likely won't work to separate P from NP
- these proofs fit in a framework called natural proofs

Theorem

if one-way functions exist then

natural proofs can't separate P and NP

natural proofs another hurdle?

- proof technique that shows parity not in AC^0 likely won't work to separate P from NP
- these proofs fit in a framework called natural proofs

Theorem

if one-way functions exist then

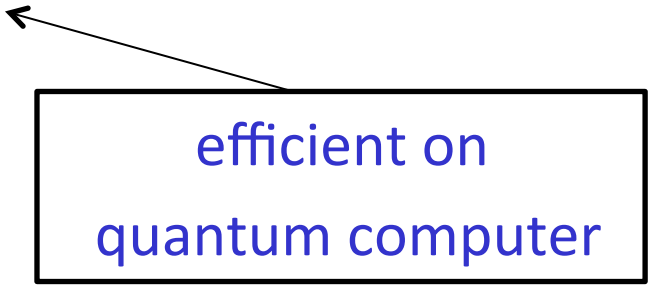
natural proofs can't separate P and NP

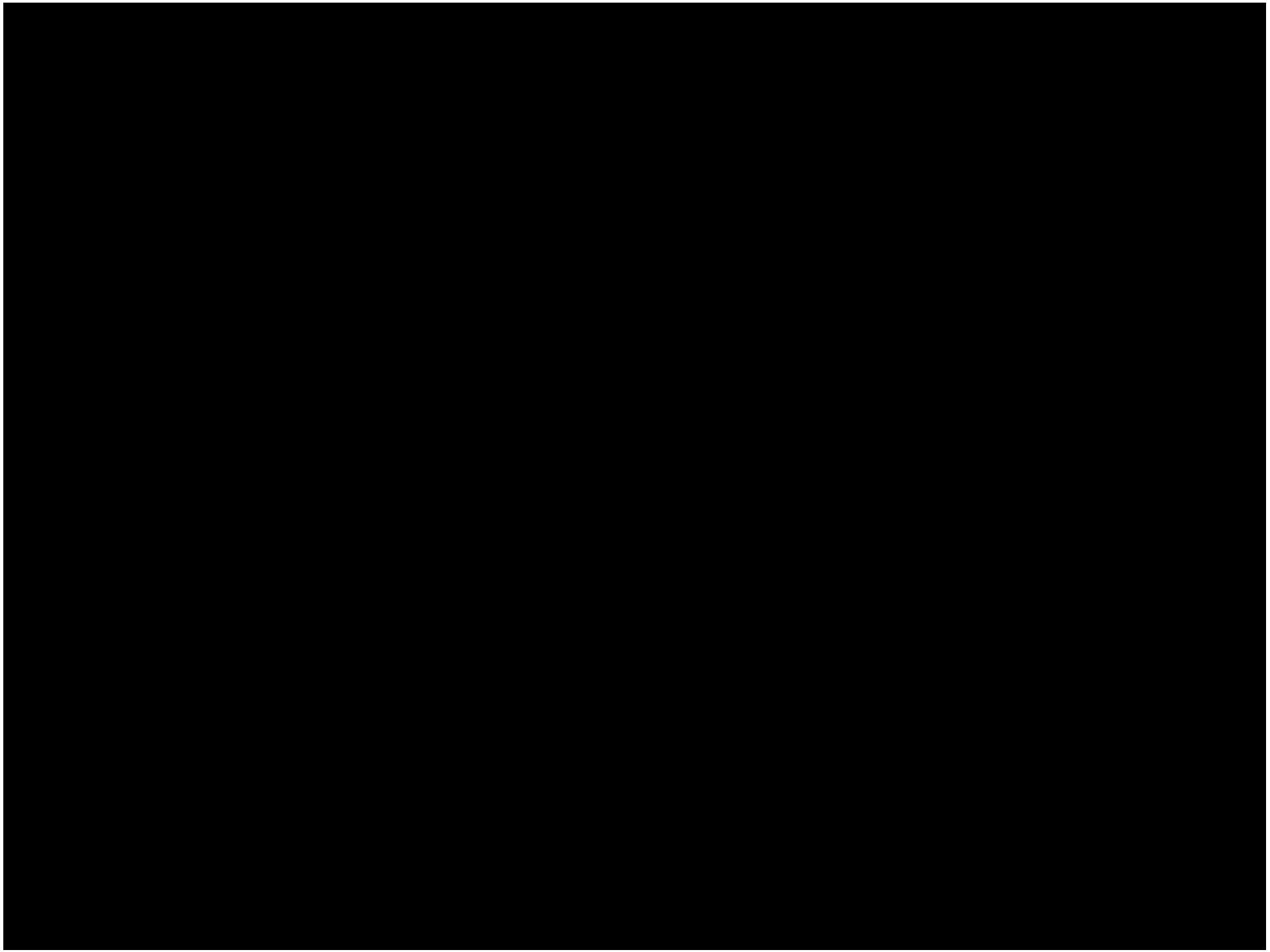
Approaches

- Structural approach using eg. autoreducibility
- Combinatorial approach
- Algebraic, degrees of multivariate polynomials
- Geometric Complexity
  - algebraic geometry
  - representation theory
- Communication complexity

P vs NP & Cryptography

- computational hardness guarantees security of cryptographic protocols
  - factoring, discrete logarithm
  - lattice problems
  - learning problems
- one-way functions
  - compute f(x) quickly
  - hard to invert
- if P=NP then no cryptography





Quantum Computing
&
Complexity Theory

Physics and Computing

Computing is physical

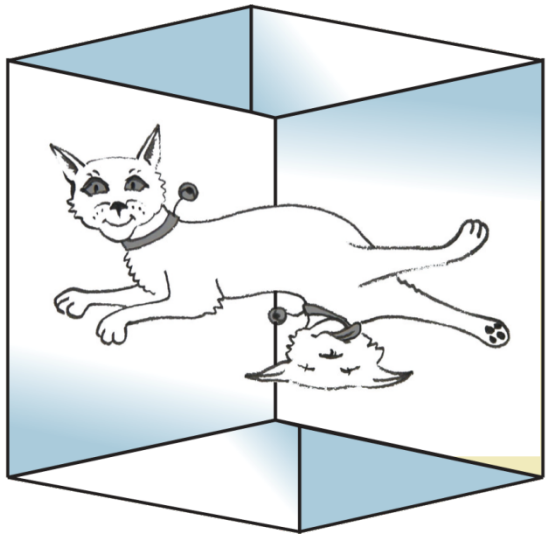
Miniaturization → quantum effects

→Quantum Computers

1) Enables continuing miniaturization
2) Fundamentally faster algorithms
3) New computing paradigm

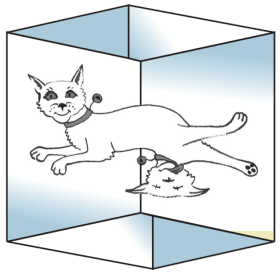
Superposition

- object in *more* states at *same* time
- Schrödinger's cat: dead *and* alive



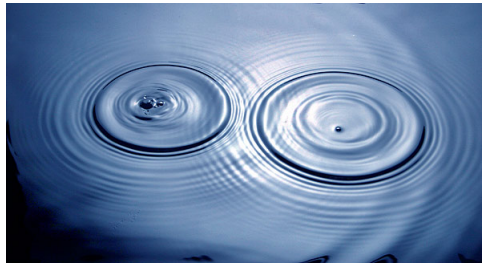
- Experimentally verified:
  - small systems, e.g. photons
  - larger systems, molecules

Quantum Mechanics

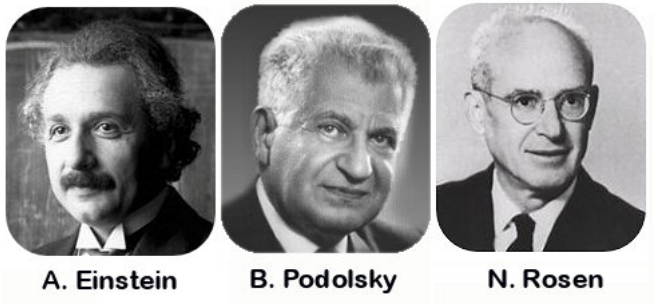


- Most complete description of Nature to date
- Superposition principle:
  - "particle can be at two positions at the same time"

- Interference:
  - particle in superposition can interfere with itself



- Entanglement:
  - Non-locality
  - EPR paradox



Quantum Information Processing

- qubit, superposition of bits

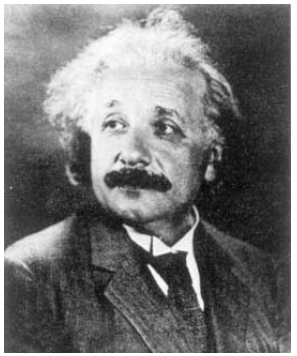
$$\alpha|0\rangle + \beta|1\rangle \qquad \begin{array}{c}\alpha, \beta \in \mathbb{C} \\ |\alpha|^2 + |\beta|^2 = 1\end{array}$$

- no cloning: qubit cannot be copied

- entanglement

$$\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$$

EPR-paradox
non-locality



Measurement & Evolution

- Measuring qubit: $\alpha|0\rangle + \beta|1\rangle$ $\quad \alpha, \beta \in \mathbb{C}$ $\quad |\alpha|^2 + |\beta|^2 = 1$
- Outcome: prob. distribution
  - observe 0 with prob. $|\alpha|^2$
  - observe 1 with prob. $|\beta|^2$

Example

$$|\psi\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

Example

$$|\psi\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

Measuring ψ: Prob [0] = 1/2

Prob [1] = 1/2

Example

$$|\psi\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

Measuring ψ: Prob [0] = 1/2

Prob [1] = 1/2

After measurement:

with prob 1/2 $|\psi\rangle = |0\rangle$

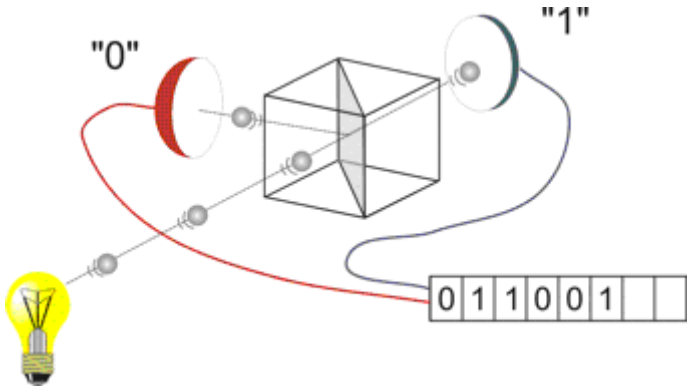
with prob 1/2 $|\psi\rangle = |1\rangle$

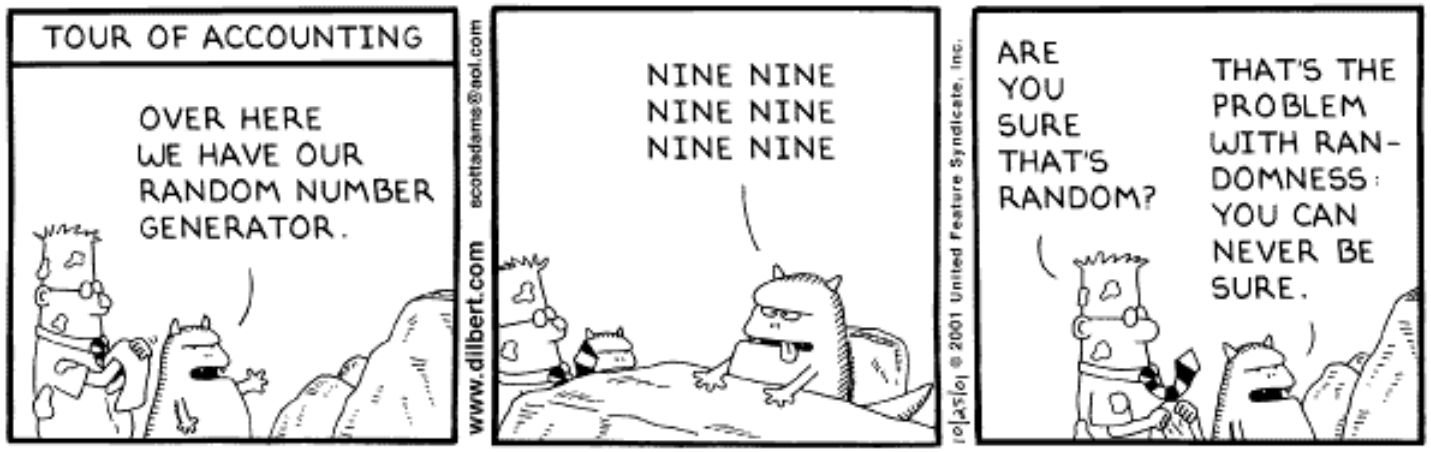
id Quantique
A quantum leap for cryptography

Quantis – QUANTUM RANDOM NUMBER GENERATOR

Although random numbers are required in many applications, their generation is often overlooked. Being deterministic, computers are not capable of producing random numbers. A physical source of randomness is necessary. Quantum physics being intrinsically random, it is natural to exploit a quantum process for such a source. Quantum random number generators have the advantage over conventional randomness sources of being invulnerable to environmental perturbations and of allowing live status verification.

Quantis is a physical random number generator exploiting an elementary quantum optics process. Photons - light particles - are sent one by one onto a semi-transparent mirror and detected. The exclusive events (reflection - transmission) are associated to "0" - "1" bit values.





Measurement & Evolution

- Measuring qubit: $\alpha|0\rangle + \beta|1\rangle$ $\quad \alpha, \beta \in \mathbb{C}$, $|\alpha|^2 + |\beta|^2 = 1$
- Outcome: prob. distribution
  - observe 0 with prob. $|\alpha|^2$
  - observe 1 with prob. $|\beta|^2$
- Evolution of system (quantum program)
  - Unitary operation
  - $U \cdot U^* = I$ (U^* : complex conjugate, transpose)

Major results of QIP

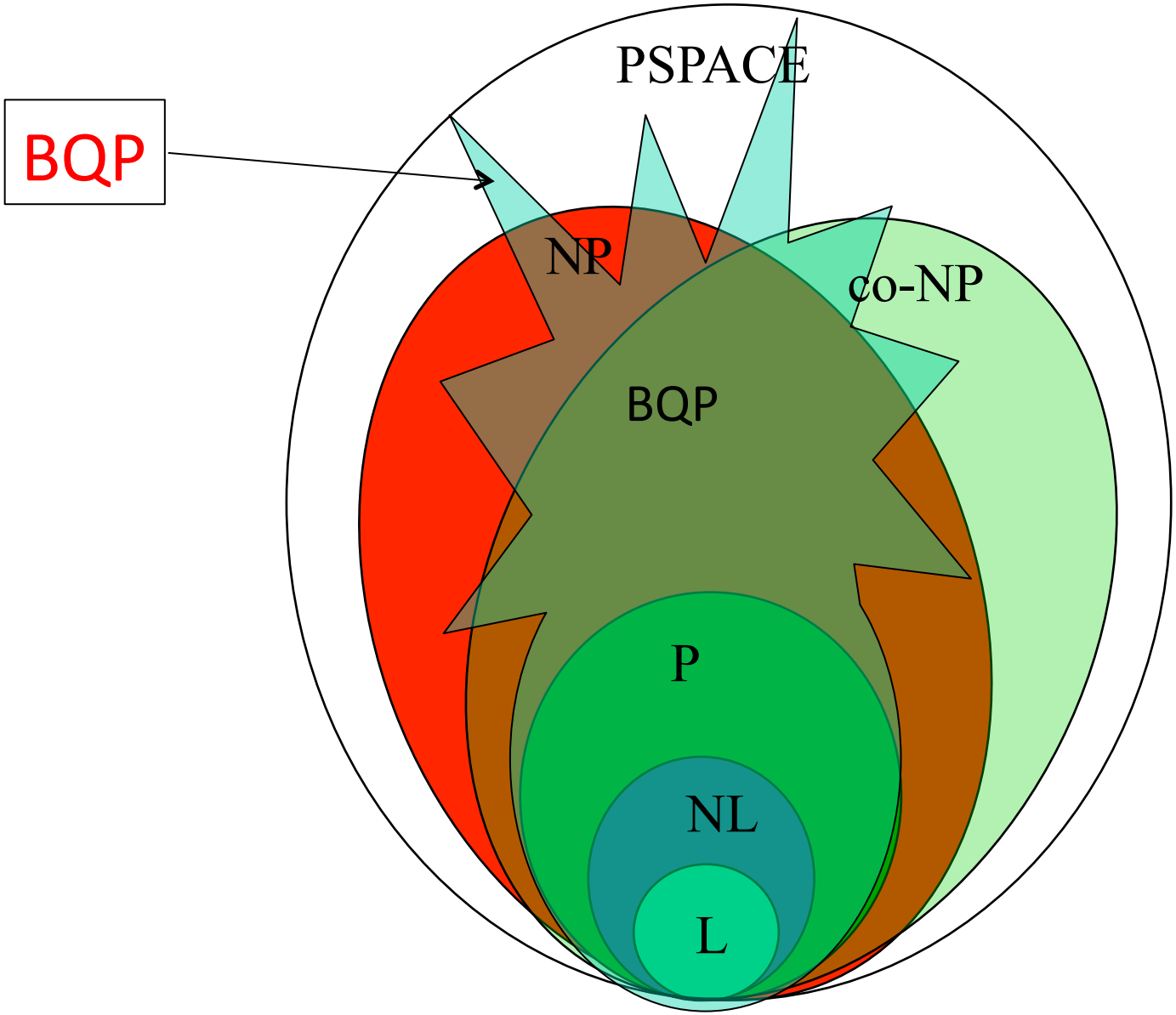
- Efficient quantum algorithm for factoring
  - breaks public key cryptography (RSA) [Shor'94]
- Fast quantum search algorithm [Grover'96]
  - quadratic speedup, widely applicable
- Quantum communication complexity
  - exponential savings in communication
- Quantum Cryptography [Bennett-Brassard'84]
  - Quantum key exchange

Quantum Polynomial Time

- New Complexity Class
- Problems that can be efficiently computed on a quantum computer

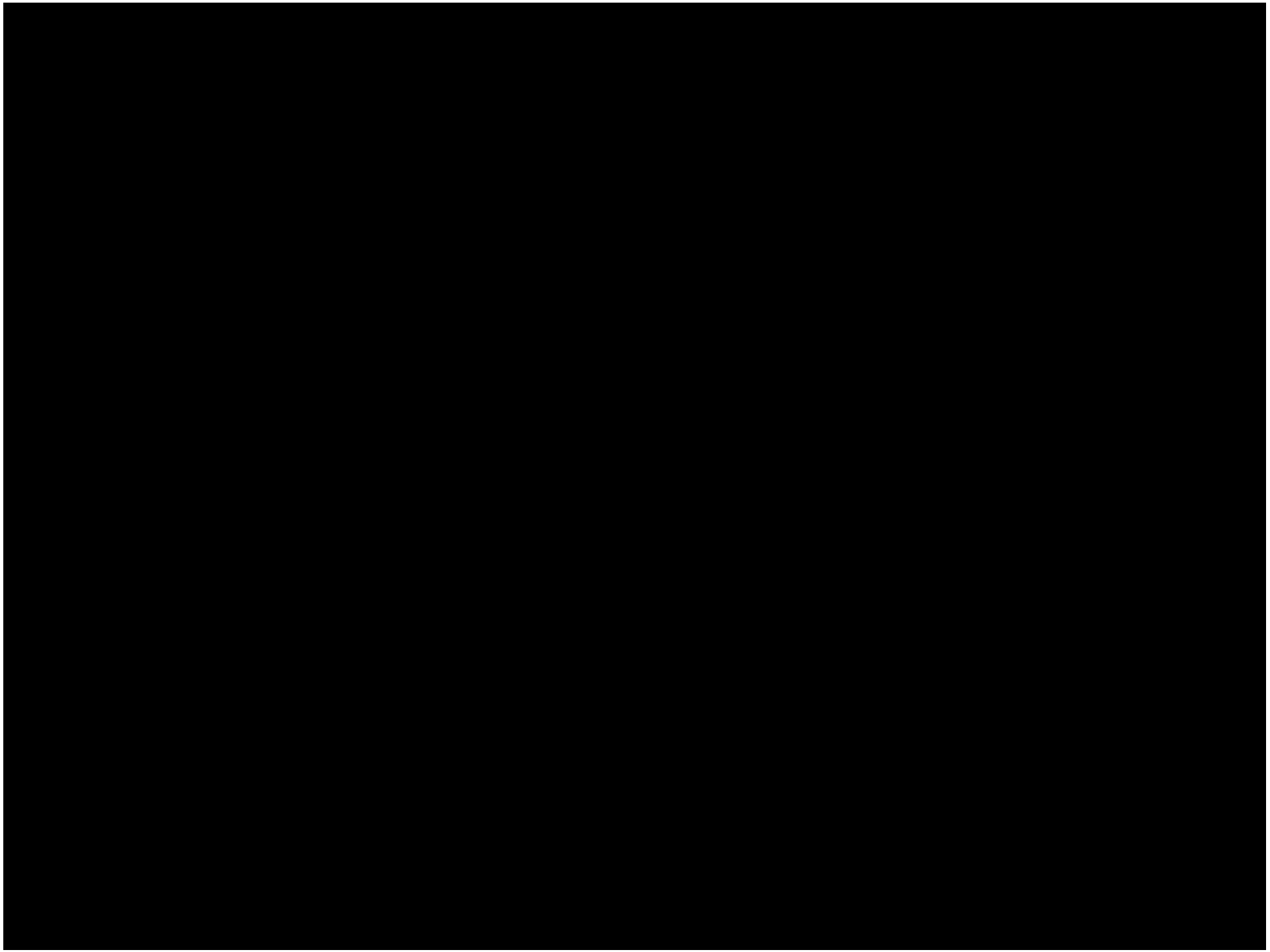
BQP

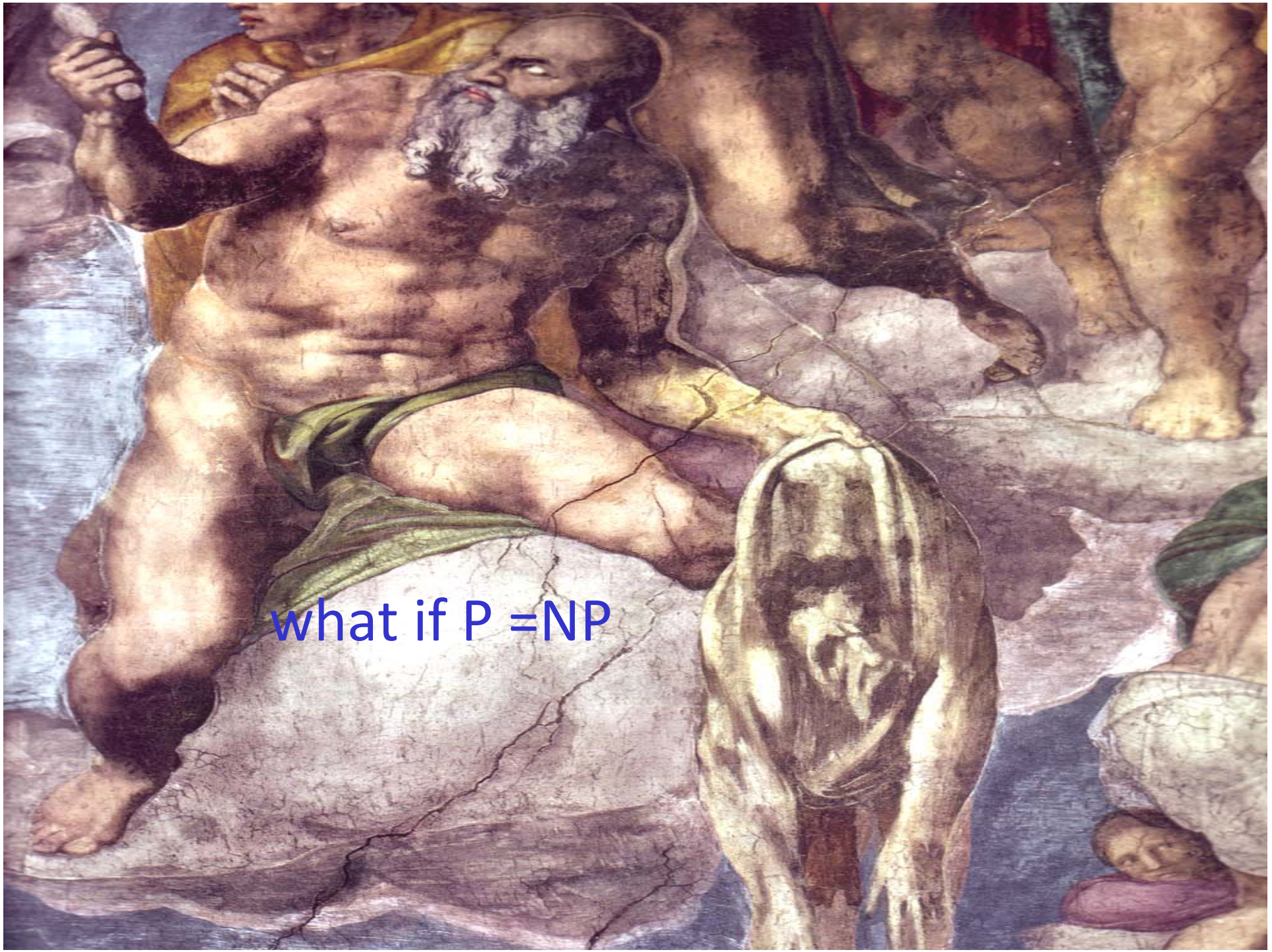
- Where does BQP sit in the complexity landscape?


BQP
PSPACE
NP
co-NP
BQP
P
NL
L

BQP and Complexity

- BQP contains P (and BPP)
- BQP is in PSPACE
- Is not believed to contain NP
  - open: show this would imply unlikely classical consequences.
- Is not known to be in the Polynomial Hierarchy (PH)
  - open: oracle such that BQP not in PH




what if P =NP

P=NP

- P=NP, but the proof does not give us an algorithm
- P=NP, but algorithm for SAT runs in time $n^{1000000}$
- P=NP, but algorithm for SAT runs in time $2^{100}n$
- P=NP, and algorithm for SAT runs in time n^2

n^2 algorithm for SAT

- Wonderful!!!
  - computing ground states of Hamiltonians
  - protein folding problem solved
  - artificial Intelligence takes really off
  - optimal scheduling
  - computational learning theory
  - weather prediction improves

n^2 algorithm for SAT

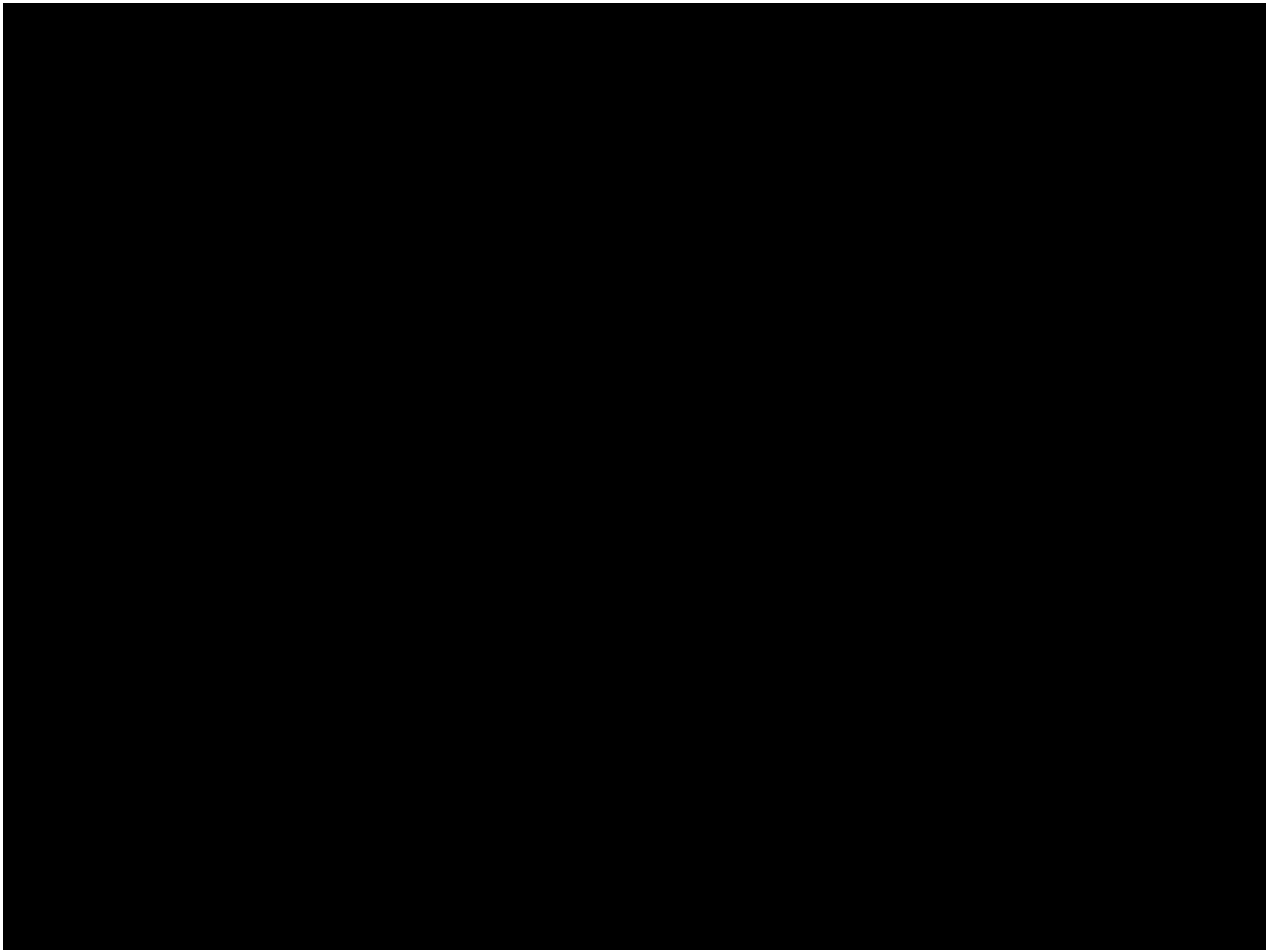
- for mathematics
  - can find proofs to theorems, provided they have short proofs
  - can simply ask computer whether theorem/ conjecture is true/false
  - mathematics will change dramatically
  - quickly solve the other 5 remaining Clay problems

Summary

- P versus NP central, not just in mathematics and computer science but also in physics, biology, chemistry, cryptography etc.
- not clear how to attack it, several obstacles: relativization, natural proofs, algebraization
- much simpler questions are still way out of reach

Schedule

2) P, NP, reductions, co-NP
3) Cook-Levin Thm:3-SAT is NP-complete, Decision vs Search
4) Diagonalization, time hierarchies
5) Relativization
6) Space complexity, PSPACE, L, NL
7) The polynomial hierarchy
8) Circuit complexity, the Karp-Lipton Theorem
9) Parity not on AC^0
10) Probabilistic algorithms
11) BPP, circuits and polynomial hierarchy
12) Interactive proofs, Graph-Isomorphism problem
13) IP = PSPACE
14) Derandomization



Het is opgelost: het grootste en mooiste probleem uit de computerwetenschap. Dat zegt een onderzoeker. In zijn bewijs zitten nog veel gaten, zeggen anderen.

tekst Margriet van der Heijden

De oplossing van het onoplosbare

Monday, August 9th, 2010

Putting my money where my mouth isn't

A few days ago, Vinay Deolalikar of HP Labs started circulating a claimed proof of P≠NP. As anyone could predict, the alleged proof has already been Slashdotted (see also Lipton's blog and Bacon's blog), and my own inbox has been filling up faster than the Gulf of Mexico.

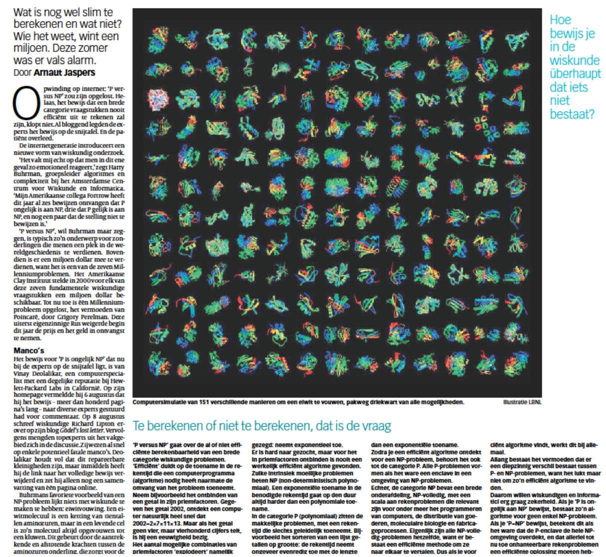
The New York Times

Step 1: Post Elusive Proof. Step 2: Watch Fireworks.

By John Markoff

Published: August 16, 2010

The potential of Internet-based collaboration was vividly demonstrated this month when complexity theorists used blogs and wikis to pounce on a claimed proof for one of the most profound and difficult problems facing mathematicians and computer scientists.



Bloggers slopen droombewijs

Wat is nog wel slim te berekenen en wat niet? Wie het weet, wint een miljoen. Deze zomer was er vals alarm.

Door Arnaut Jaspers

Hoe bewijs je in de wiskunde überhaupt dat iets niet bestaat?

Te berekenen of niet te berekenen, dat is de vraag