CSC 240

COMPLEXITY 2

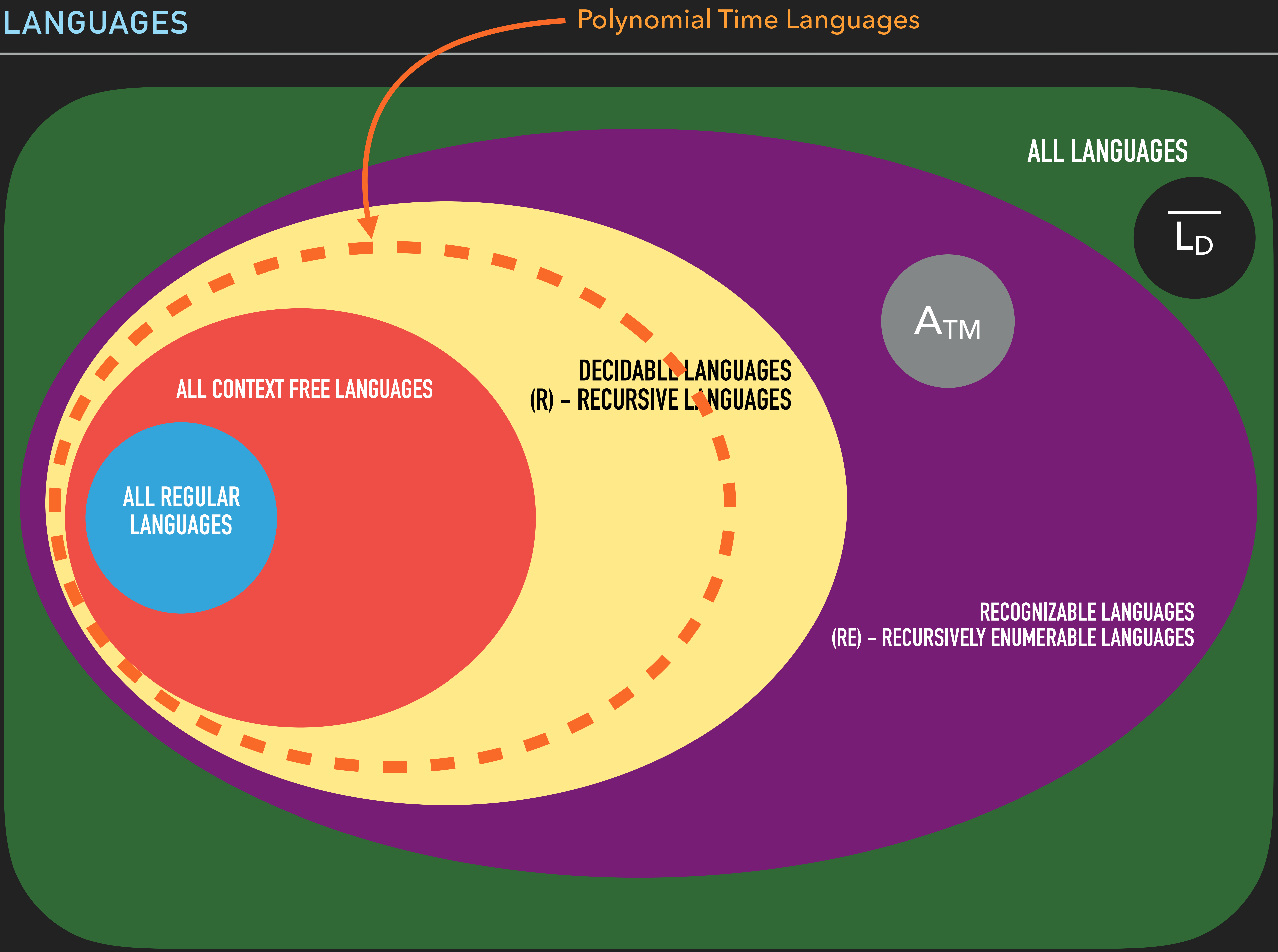
COBHAM-EDMONDS THESIS

“Computational problems can be feasibly computed on some computational device only if they can be computed in polynomial time.”

“Languages can be decided efficiently on some Turing machine only if they can be computed in $O(n^k)$.”

P = { L | There is a polynomial time decider for L }

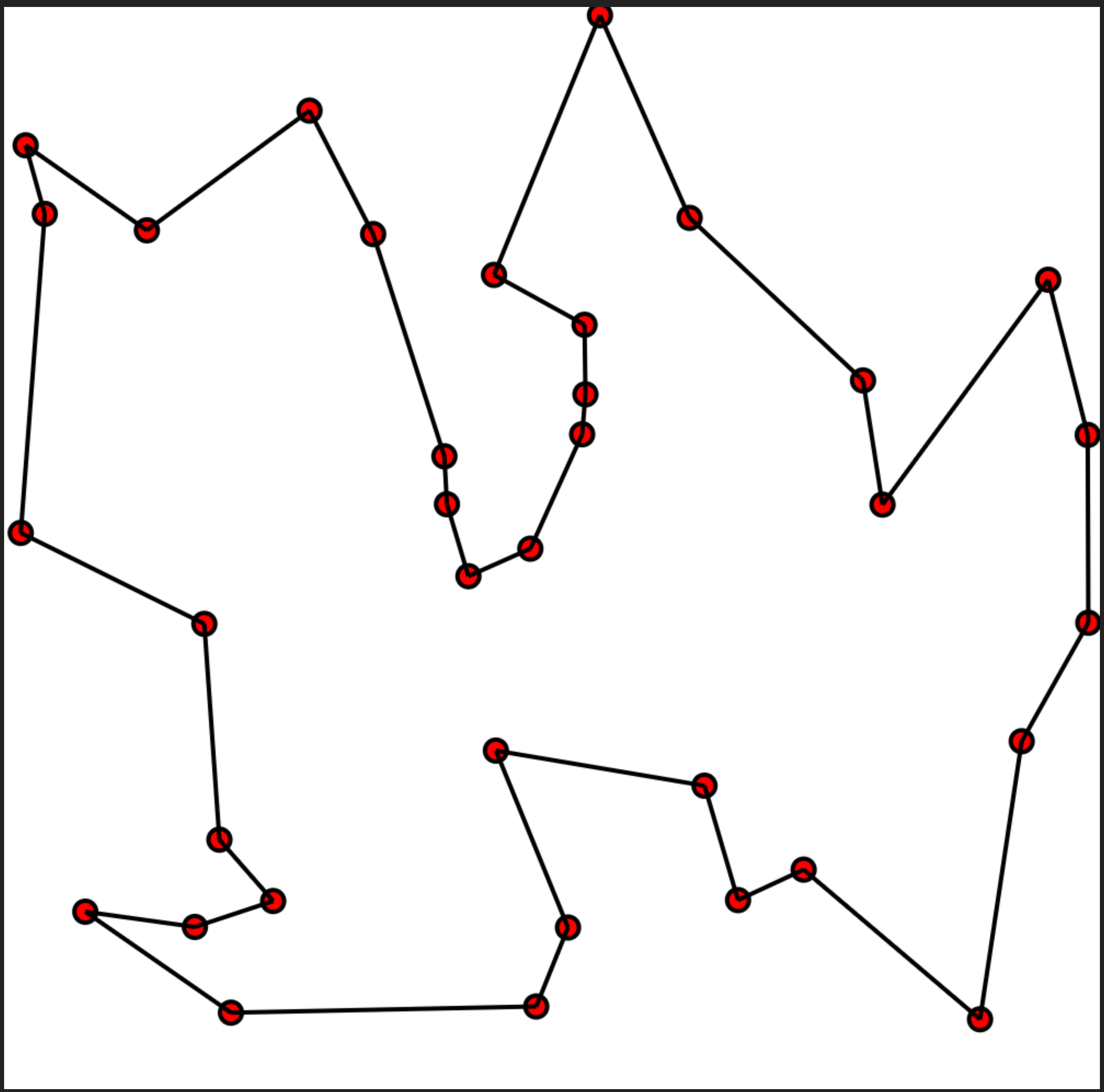
LANGUAGES



Polynomial Time Languages
ALL LANGUAGES
$\overline{L_D}$
A_{TM}
DECIDABLE LANGUAGES
(R) – RECURSIVE LANGUAGES
ALL CONTEXT FREE LANGUAGES
ALL REGULAR LANGUAGES
RECOGNIZABLE LANGUAGES
(RE) – RECURSIVELY ENUMERABLE LANGUAGES

- Searching for some specific type of thing (biggest, smallest, longest, etc…) object in a group usually results in exponentially many options.
- This means that brute-force approaches tend to run in exponential time - $O(2^n)$.
- "Good" algorithms tend to run in polynomial time $O(n)$, $O(n^2)$, $O(n^3)$, etc…
- $O(n^k)$ algorithms tend to scale well as the size of the data (n) increases, while $O(2^n)$ algorithms scale poorly.

COMPLEX PROBLEMS



Can this be *decided* in polynomial time?

Can this be *verified* in polynomial time?

ALGORITHMIC EFFICIENCY

- Complexity class P contains all problems that can be *decided* in polynomial time.
- Complexity class NP contains all problems that can be *verified* in polynomial time.

P = { L | There is a polynomial time decider for L }

NP = { L | There is a polynomial time verifier for L }

The Ultimate Question

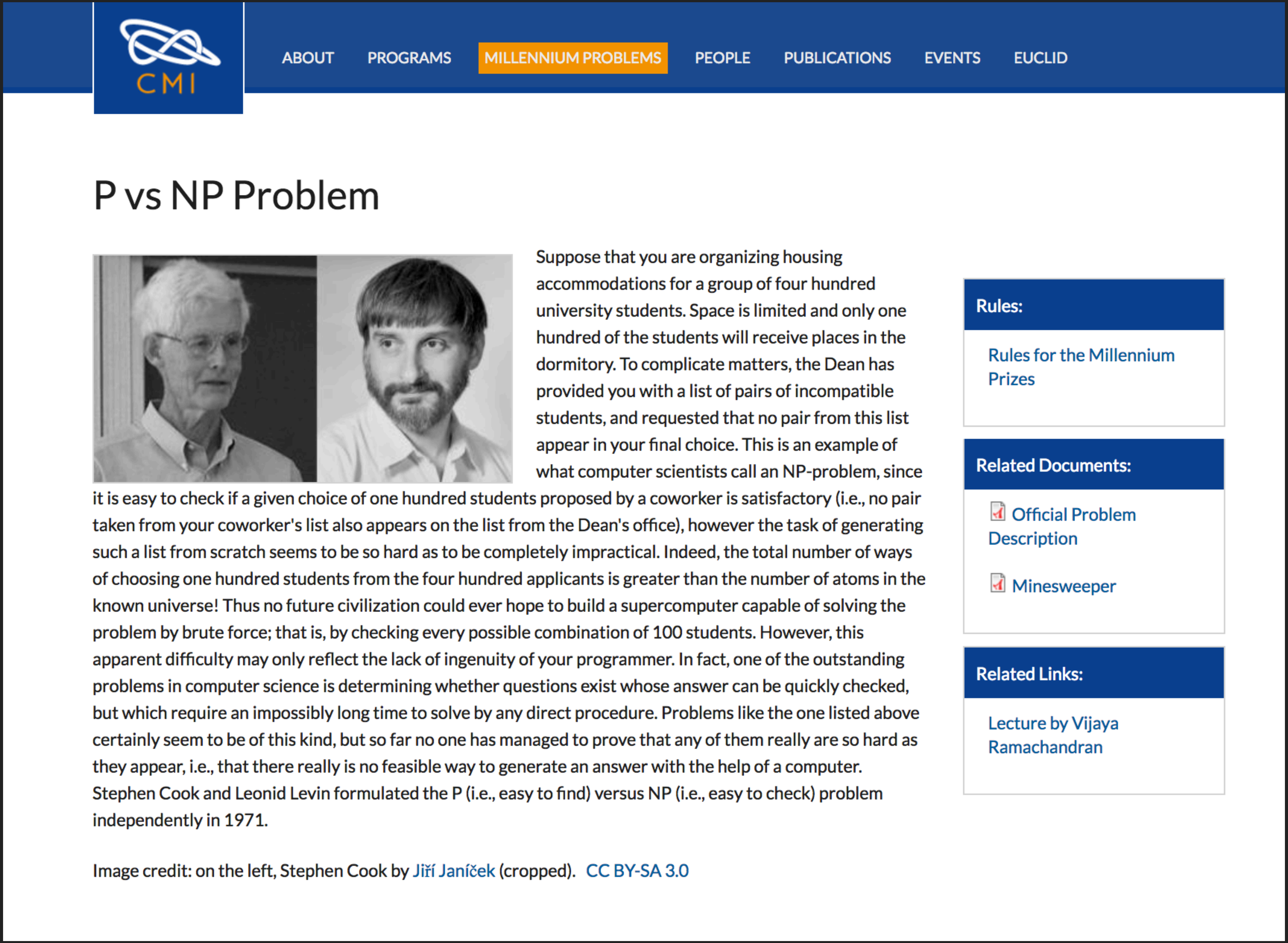
(of Computer Science)

$P \stackrel{?}{=} NP$

If a solution to a problem can be *verified* efficiently, does that mean there is *some* algorithm to *decide* that problem efficiently?

Proving this statement true or false will change the face of computer science.

THIS IS LITERALLY A MILLION DOLLAR QUESTION



P vs NP Problem

Suppose that you are organizing housing accommodations for a group of four hundred university students. Space is limited and only one hundred of the students will receive places in the dormitory. To complicate matters, the Dean has provided you with a list of pairs of incompatible students, and requested that no pair from this list appear in your final choice. This is an example of what computer scientists call an NP-problem, since it is easy to check if a given choice of one hundred students proposed by a coworker is satisfactory (i.e., no pair taken from your coworker's list also appears on the list from the Dean's office), however the task of generating such a list from scratch seems to be so hard as to be completely impractical. Indeed, the total number of ways of choosing one hundred students from the four hundred applicants is greater than the number of atoms in the known universe! Thus no future civilization could ever hope to build a supercomputer capable of solving the problem by brute force; that is, by checking every possible combination of 100 students. However, this apparent difficulty may only reflect the lack of ingenuity of your programmer. In fact, one of the outstanding problems in computer science is determining whether questions exist whose answer can be quickly checked, but which require an impossibly long time to solve by any direct procedure. Problems like the one listed above certainly seem to be of this kind, but so far no one has managed to prove that any of them really are so hard as they appear, i.e., that there really is no feasible way to generate an answer with the help of a computer. Stephen Cook and Leonid Levin formulated the P (i.e., easy to find) versus NP (i.e., easy to check) problem independently in 1971.

Image credit: on the left, Stephen Cook by Jiří Janíček (cropped). CC BY-SA 3.0

Rules:

Rules for the Millennium Prizes

Related Documents:

Official Problem Description

Minesweeper

Related Links:

Lecture by Vijaya Ramachandran

IF P = NP

A huge number of problems that were previously thought to be verifiable, but unsolvable in polynomial time, can in fact be solved in polynomial time if the right algorithm is found.

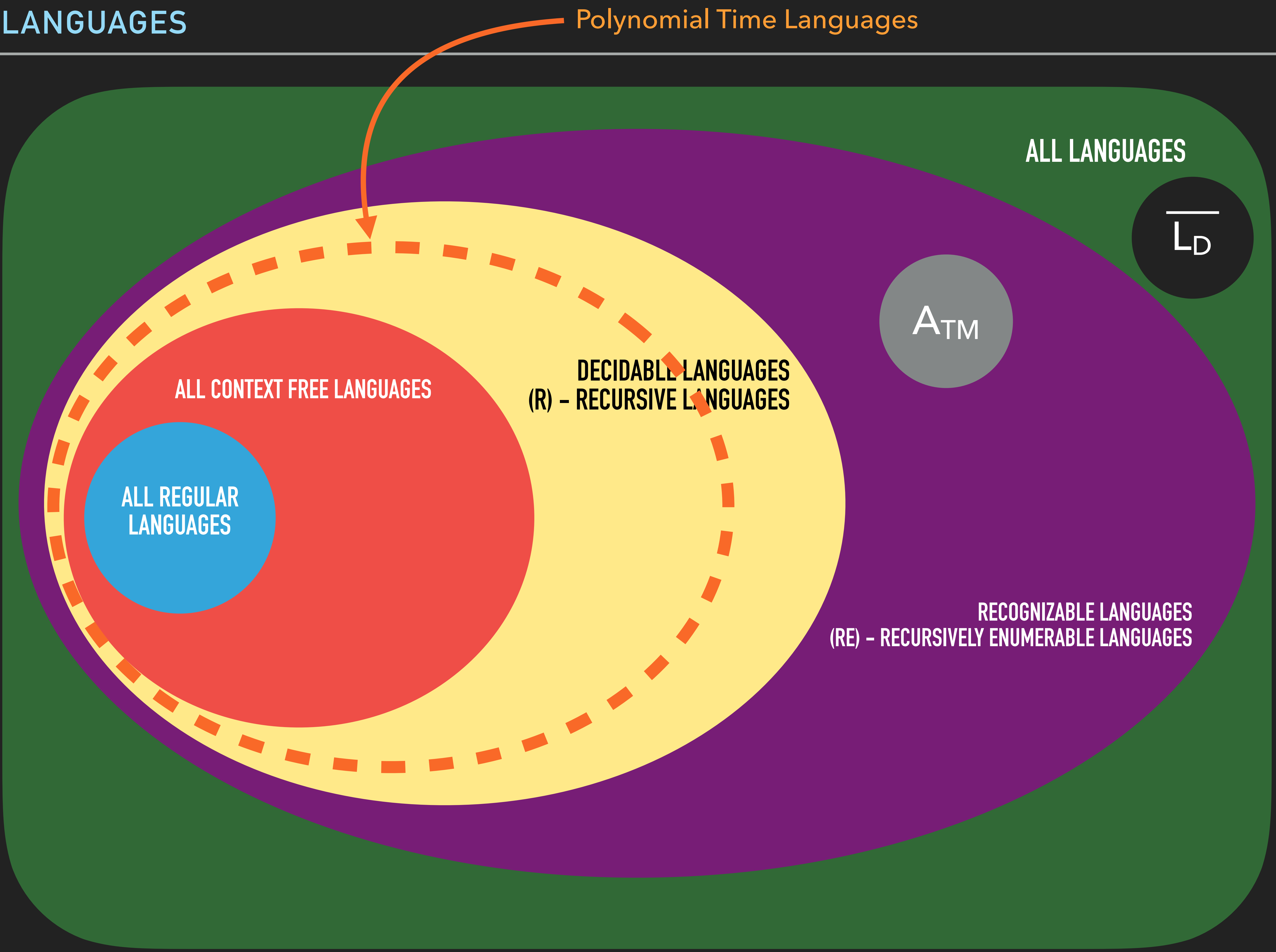
IF P ≠ NP

Entire groups of seemingly simple problems may not in fact be solvable by computers.

47 Years of attempts have been inadequate to prove this either way.

What do leading scientists think? Introduction to Complexity Theory Column 36

LANGUAGES


Polynomial Time Languages
ALL LANGUAGES
$\overline{L_D}$
A_{TM}
DECIDABLE LANGUAGES
(R) – RECURSIVE LANGUAGES
ALL CONTEXT FREE LANGUAGES
ALL REGULAR LANGUAGES
RECOGNIZABLE LANGUAGES
(RE) – RECURSIVELY ENUMERABLE LANGUAGES