CS3102 Theory of Computation

MacGyver's Reduction

Problem known to be "hard"

Opening a door

A

Solution for $\boldsymbol{A}$

Keg cannon battering ram

Aim duct at door, insert keg

Put fire under the Keg

Reduction

Problem of uknown "hardness"

Lighting a fire

B

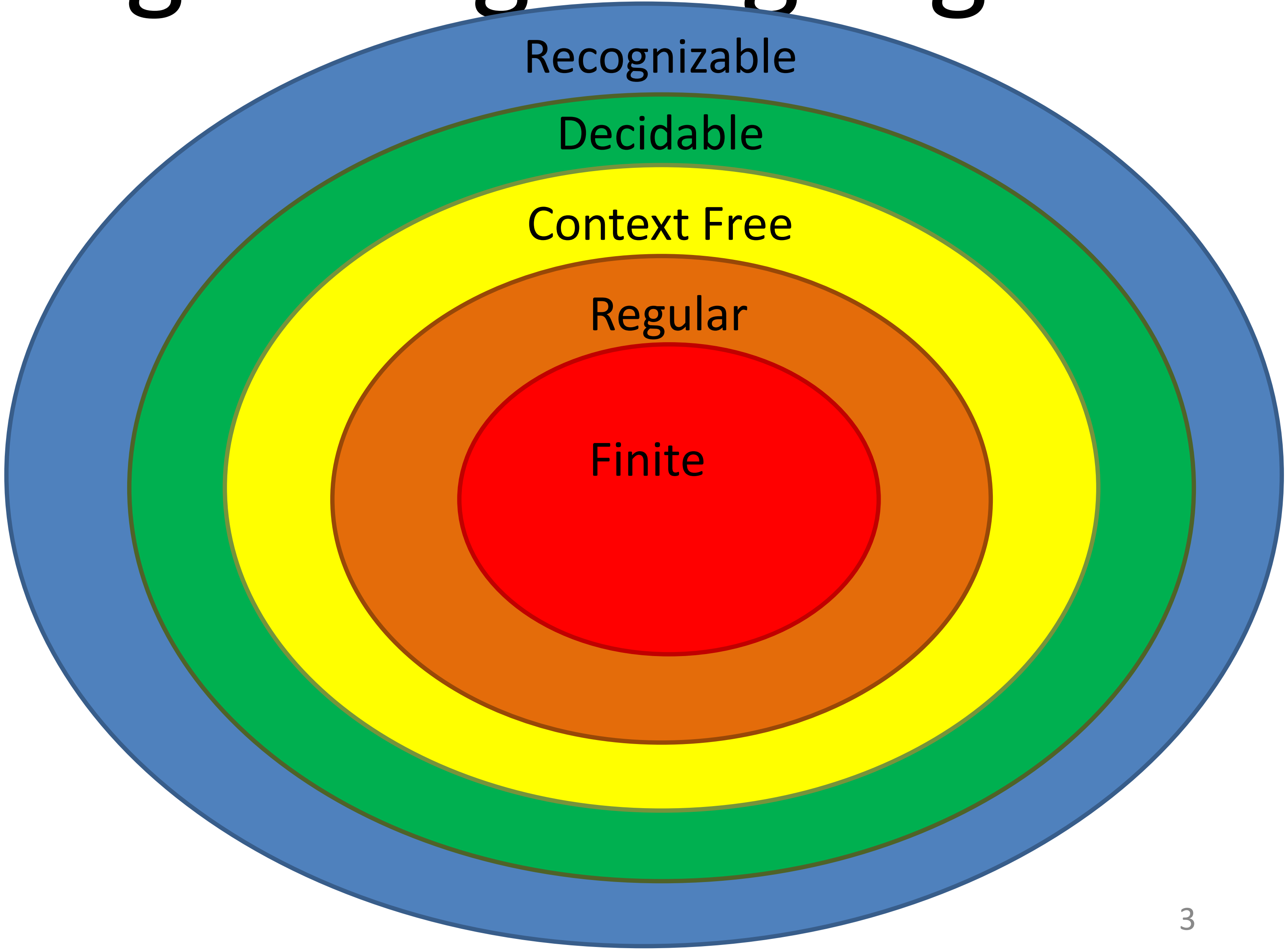
How?

Solution for $\boldsymbol{B}$

Alcohol, wood, matches

Theme: Categorizing Languages

- So far:
  - Finite
  - Regular
  - Context Free
  - Decidable
  - Recognizable



Why Categorize Languages?

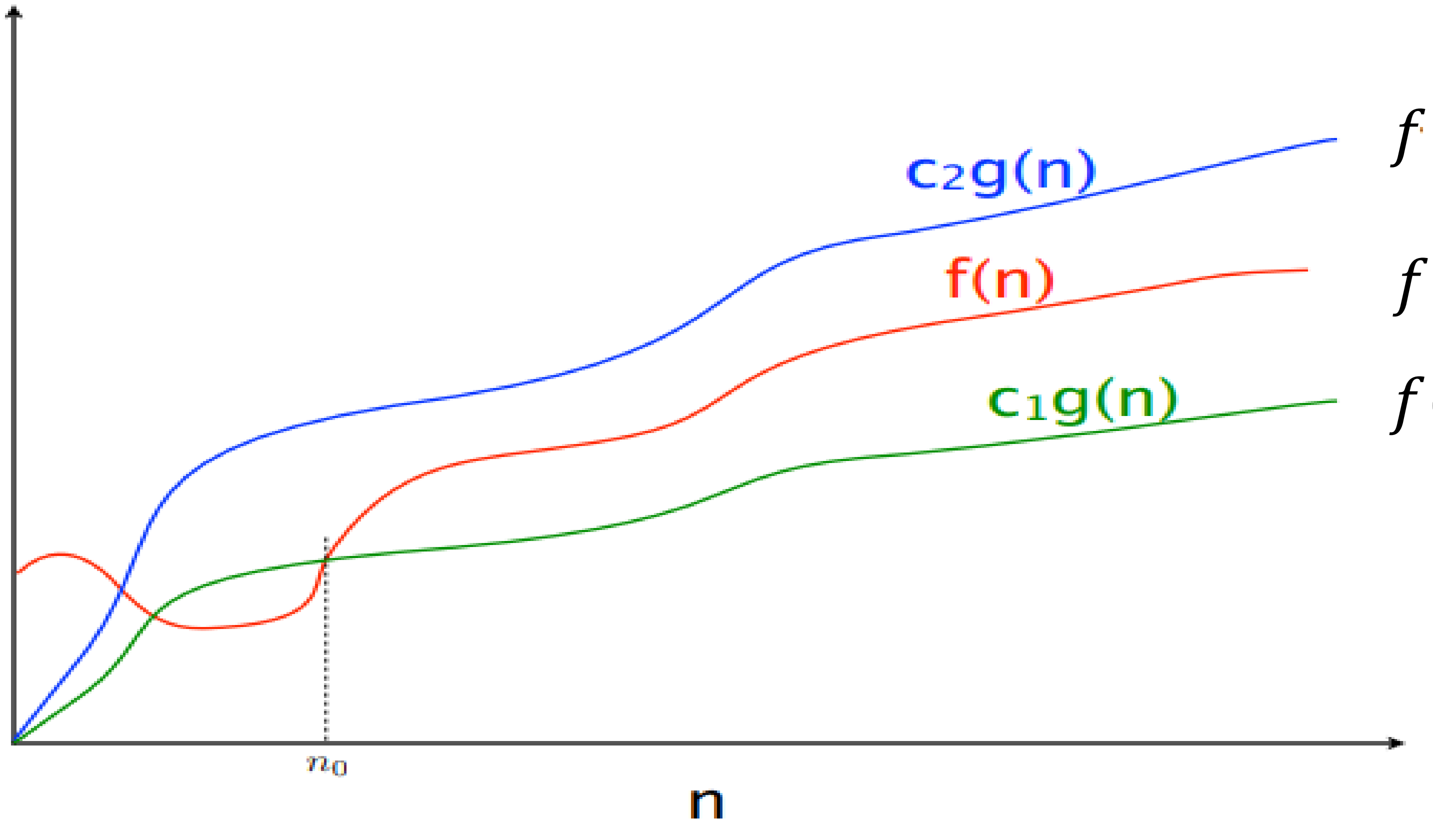
- Indicates limits on resources needed to compute by Turing Machines relative to input size
- Space = number of cells required on the tape (beyond input)
- Time = number of transitions required by the machine
- Regular:
  - Constant space
  - Linear time
- Context Free:
  - Linear space

Categorizing Languages by Complexity

- So far: Categorize by kind of machine needed to express
- Going forward: Categorize by amount of resources a Turing Machine needs (asymptotic, relative to input size)

Asymptotic Notation

- $O(g(n))$
  - At most within constant of g for large n
  - $\{\text{functions } f \mid \exists \text{ constants } c, n_0 > 0 \text{ s.t. } \forall n > n_0, f(n) \leq c \cdot g(n)\}$
- $\Omega(g(n))$
  - At least within constant of g for large n
  - $\{\text{functions } f \mid \exists \text{ constants } c, n_0 > 0\text{s.t. } \forall n > n_0, f(n) \geq c \cdot g(n)\}$
- $\Theta\big(g(n)\big)$
  - "Tightly" within constant of g for large n
  - $\Omega\big(g(n)\big) \cap O(g(n))$



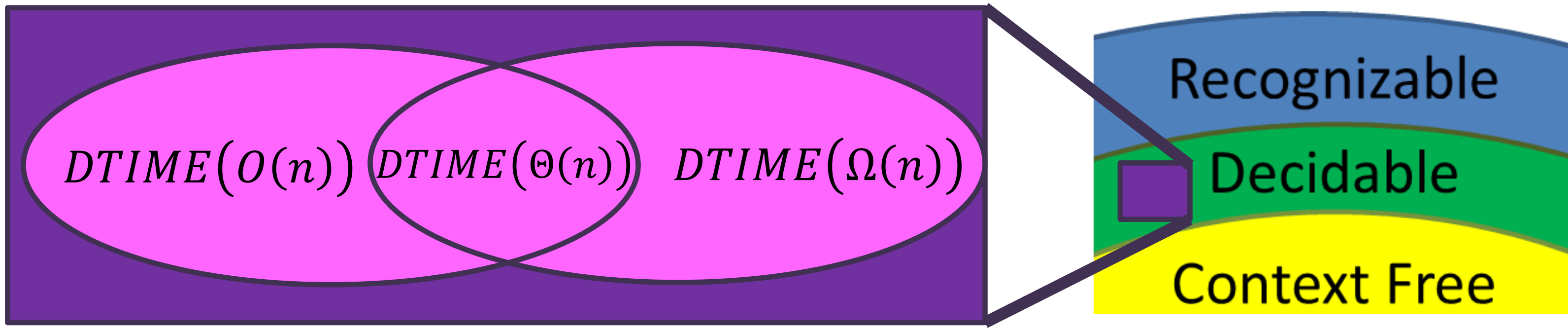
$$f(n) = O(g(n))$$

$$f(n) = \Theta(g(n))$$

$$f(n) = \Omega(g(n))$$

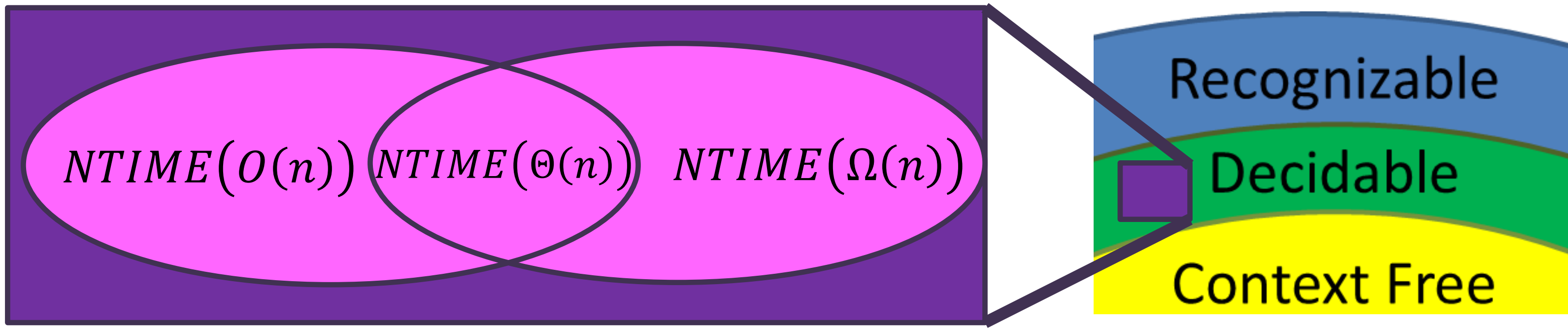
Complexity Classes

- A set of languages grouped by resources required to compute
- $DTIME(O(n)) =$
  $\{L | L \text{ can be decided by an up to linear time deterministic TM}\}$
- $DTIME(\Omega(n)) =$
  $\{L | L \text{ can be decided by an at least linear time deterministic TM}\}$
- $DTIME(\Theta(n)) = DTIME(O(n)) \cap DTIME(\Omega(n))$
- Last class's "enhancements" didn't change what we could possibly compute, but it could change how long the computation took
- Decidable languages only



Non-Determinisitic Complexity Classes

- A set of languages grouped by resources required to compute
- $NTIME(O(n)) =$
  $\{L | L \text{ can be decided by an up to linear time nondeterministic TM}\}$
- $NTIME(\Omega(n)) =$
  $\{L | L \text{ can be decided by an at least linear time nondeterministic TM}\}$
- $NTIME(\Theta(n)) = NTIME(O(n)) \cap NTIME(\Omega(n))$
- Last class's "enhancements" didn't change what we could possibly compute, but it could change how long the computation took
- Decidable languages only



Deterministic vs Non-deterministic

- Last time:
  - We can convert any deterministic Turing machine into a non-deterministic Turing machine
  - This conversion was very inefficient
- Open problem:
  - Can we make this efficient?

P vs NP Problem

- Among the most significant open problems
- If a problem is “efficient” on a non-deterministic TM is also “efficient” on a Deterministic one?
- “Efficient” means $O(n^p)$ for some $p \in \mathbb{N}$
- Are the problems solvable in deterministic **p**olynomial time (P), the same as those solvable in **n**on-deterministic **p**olynomial time (NP)?

$P \subseteq NP$

- Why?
- Non-determinism is a “super power”
- Any deterministic Turing machine is already a non-deterministic Turing machine

Why do we care?

- P
  - Problems we can *solve* efficiently
- NP
  - Problems we can *verify* efficiently
  - Verify: Given a potential solution, check if it’s correct
- Equivalent statement
  - If we can verify solutions efficiently, can we find them efficiently as well?

Problem Types

- Decision Problems:
  - Is there a solution?
    - Output is True/False
  - **Can** all these boxes fit in the trunk of my car?
- Search Problems:
  - Find a solution
    - Output is complex
  - Show me how to make these boxes fit in the trunk of my car.
- Verification Problems:
  - Given a potential solution, is it valid?
    - Output is True/False
  - Will the boxes fit in the trunk of your care if you load them like this?

What if P=NP?

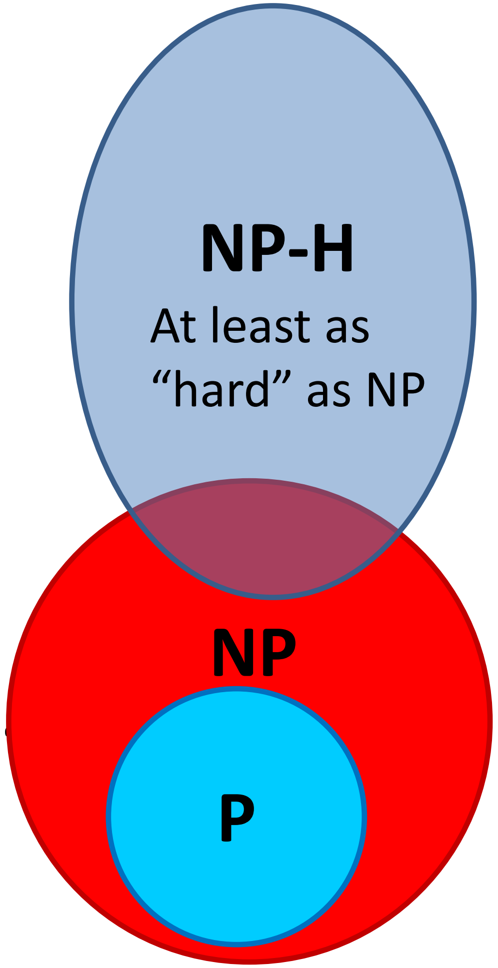
- Any problem we can verify efficiently, we can solve efficiently
- Good things:
  - Optimize packing boxes
  - Predict how proteins will fold
  - Optimally layout computer hardware
- Bad (?) things:
  - No cryptography
  - Stronger AI?

Non-determinism and Verification

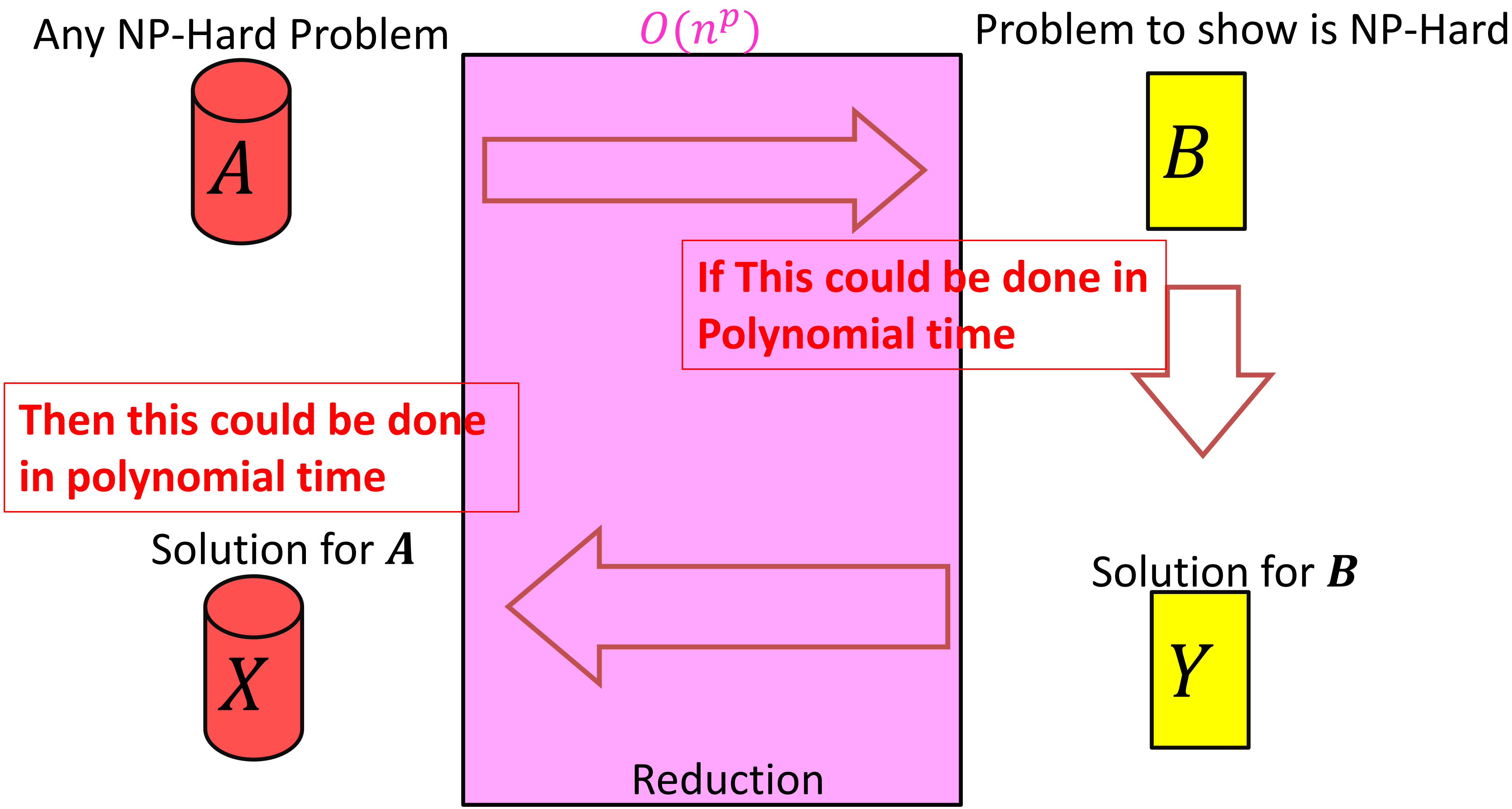
- Non-deterministic TM to Verifier:
  - To accept: Some polynomial-length path in the TM accepts
  - What might we verify: When there is a non-deterministic split, which “fork” to take
- Verifier to Non-determinstic TM:
  - Non-deterministically guess a solution for the verifier
  - Accept if the solution was valid

NP-Hard

- How can we try to figure out if P=NP?
- Identify problems at least as "hard" as NP
  - If any of these "hard" problems can be solved in polynomial time, then all NP problems can be solved in polynomial time.
- Definition: NP-Hard:
  - B is NP-Hard if $\forall A \in NP, A \leq_p B$
  - $A \leq_p B$ means A reduces to B in polynomial time

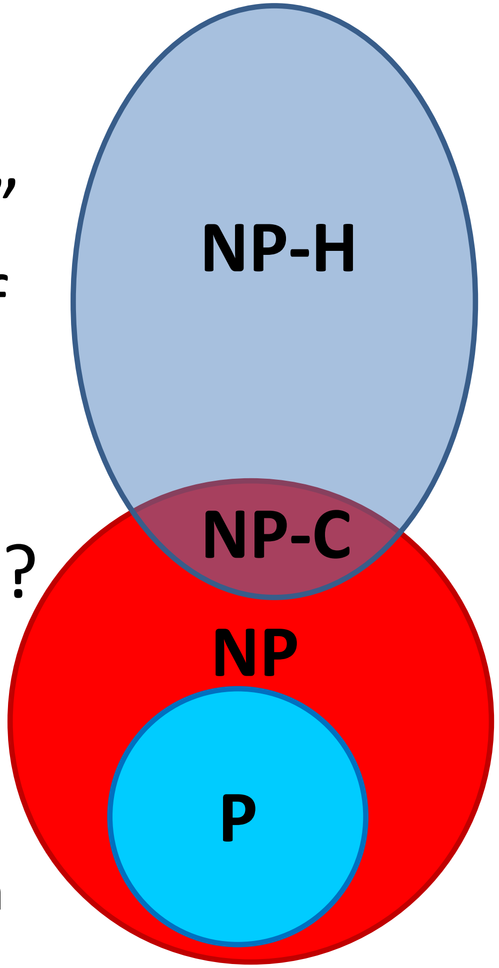


NP-Hardness Reduction



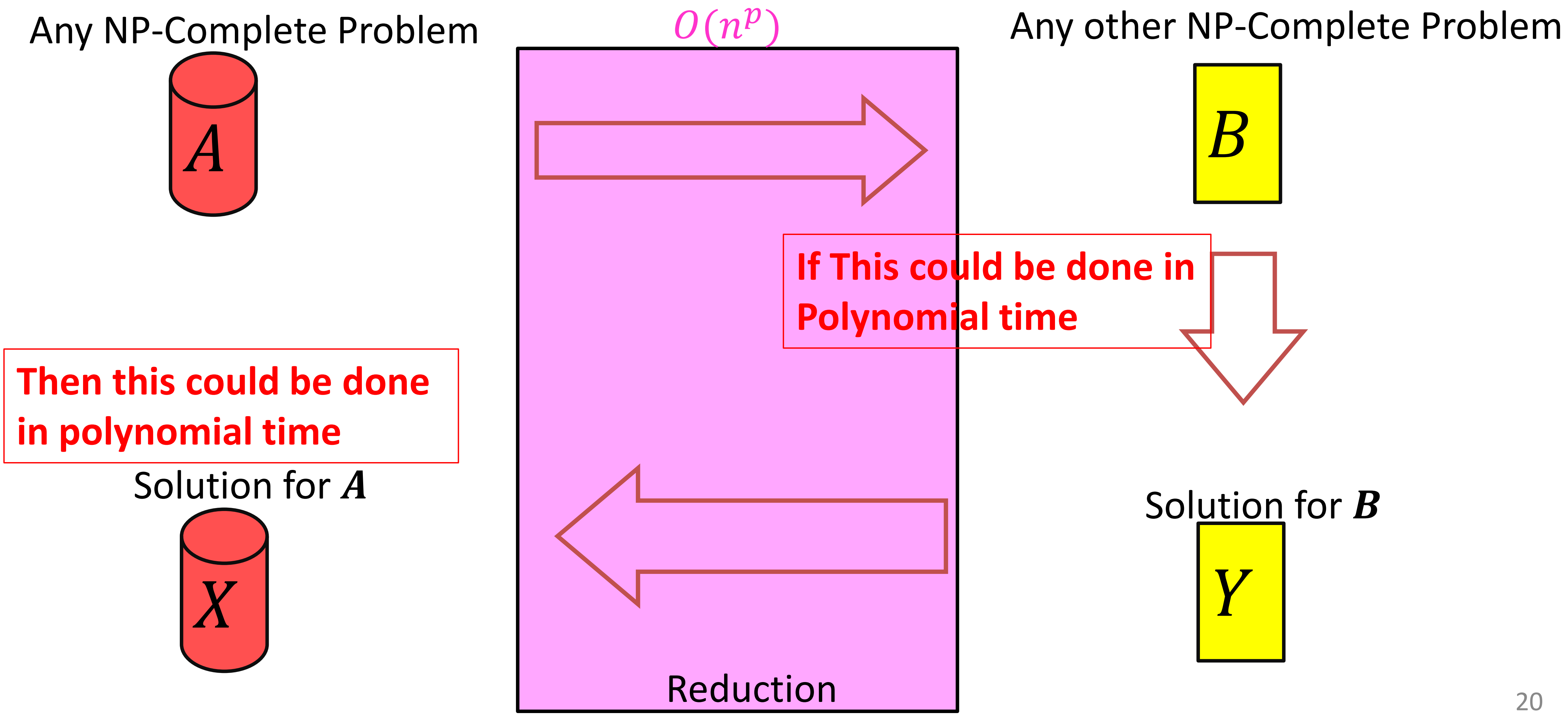
NP-Complete

- "Together they stand, together they fall"
- Problems solvable in polynomial time iff ALL NP problems are
- NP-Complete = NP ∩ NP-Hard
- How to show a problem is NP-Complete?
  - Show it belongs to NP
    - Give a polynomial time verifier
  - Show it is NP-Hard
    - Give a reduction from another NP-H problem

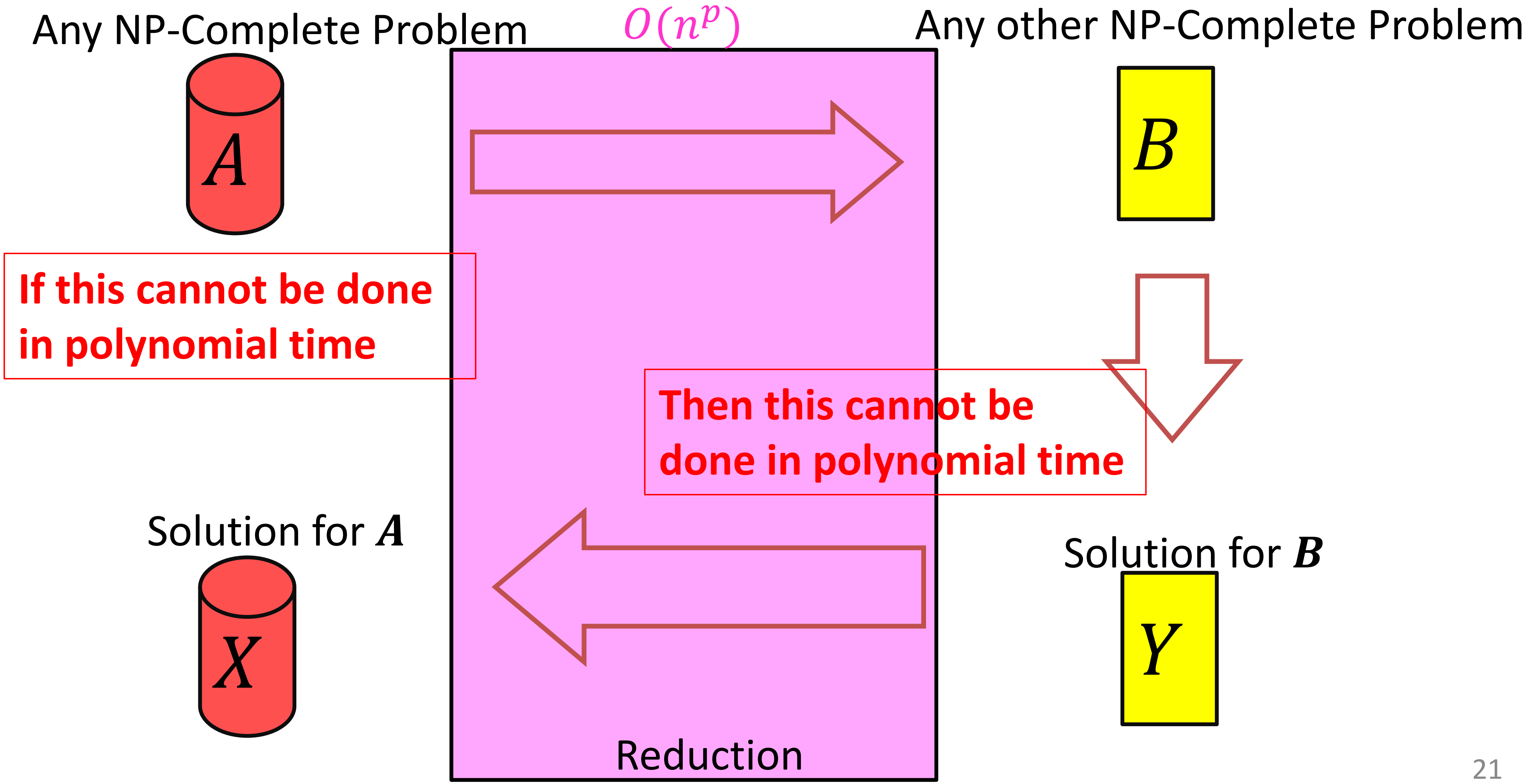


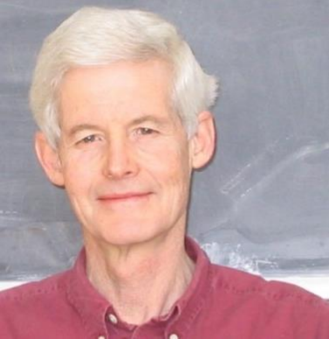
We now just need a FIRST NP-Hard problem

NP-Completeness



NP-Completeness





Stephen Cook

Leonid Levin

3-SAT

- Shown to be NP-Hard by Cook and Levin (independently)
- Given a 3-CNF formula (logical AND of clauses, each an OR of 3 variables), Is there an assignment of true/false to each variable to make the formula true?

$$\underbrace{(x \vee y \vee z)}_{\text{Clause}} \wedge (x \vee \bar{y} \vee y) \wedge (u \vee y \vee \bar{z}) \wedge (z \vee \bar{x} \vee u) \wedge (\bar{x} \vee \bar{y} \vee \bar{z})$$

Variables

$$\boldsymbol{x = true}$$
$$\boldsymbol{y = false}$$
$$\boldsymbol{z = false}$$
$$\boldsymbol{u = true}$$