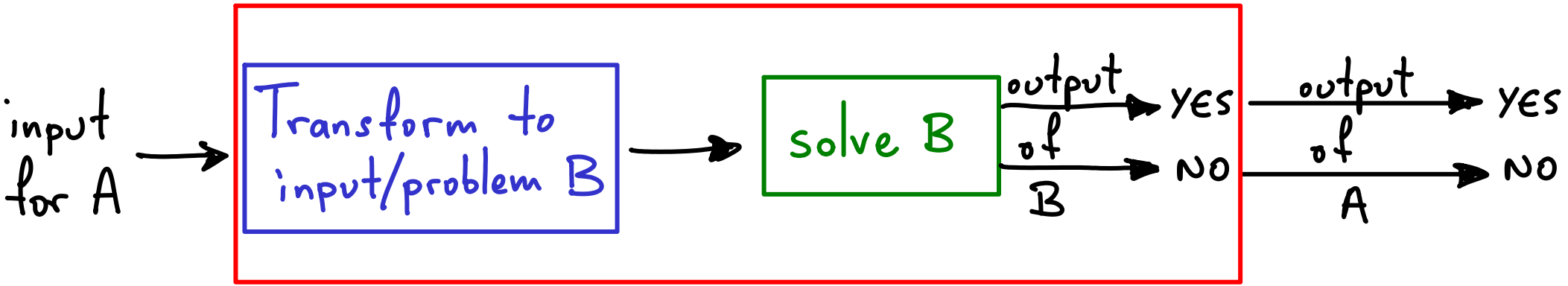
REDUCING ONE (DECISION) PROBLEM TO ANOTHER

A B

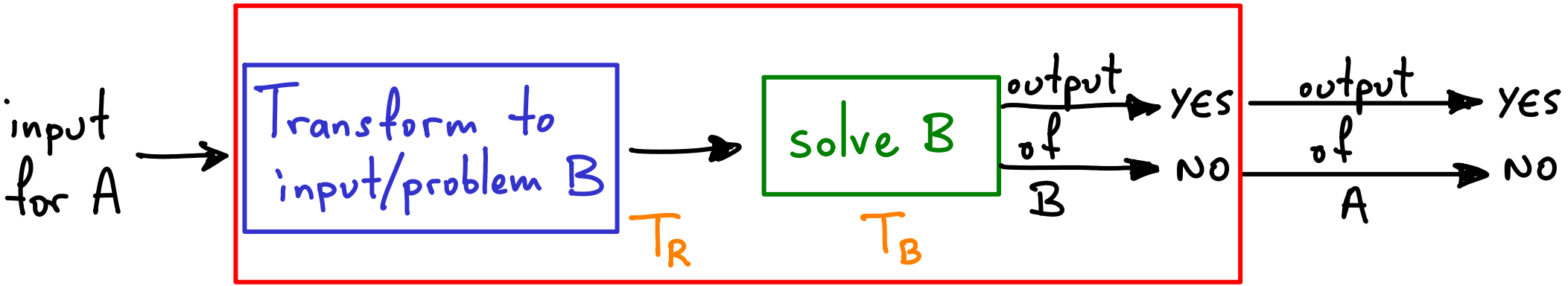
REDUCING ONE (DECISION) PROBLEM TO ANOTHER

A B



REDUCING ONE (DECISION) PROBLEM TO ANOTHER

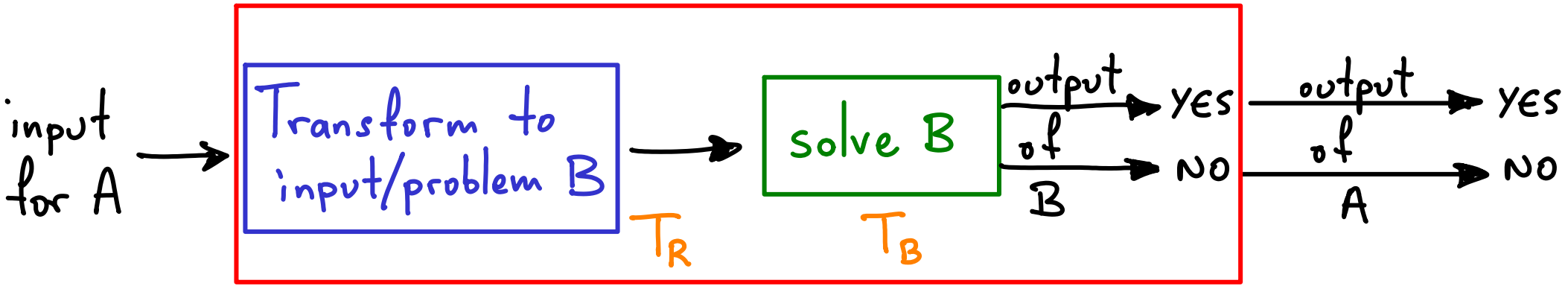
A B



solve A using B : $T_R + T_B$

REDUCING ONE (DECISION) PROBLEM TO ANOTHER

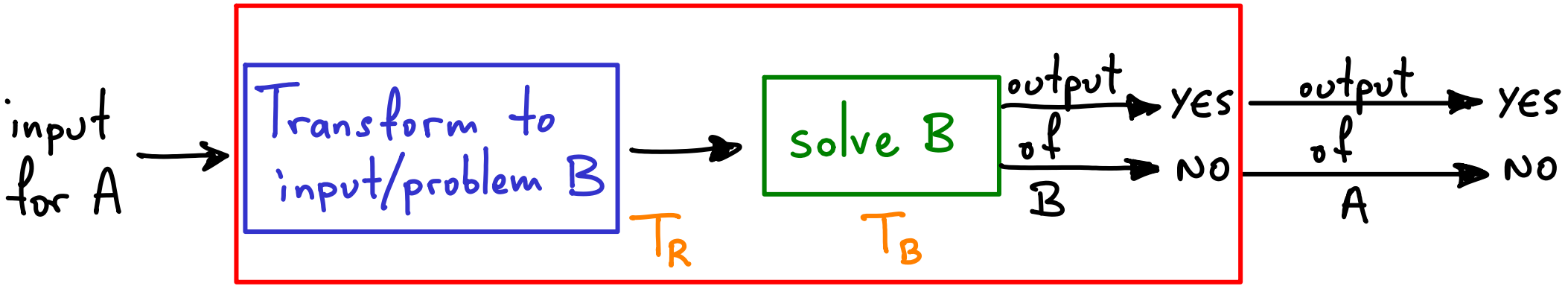
A B



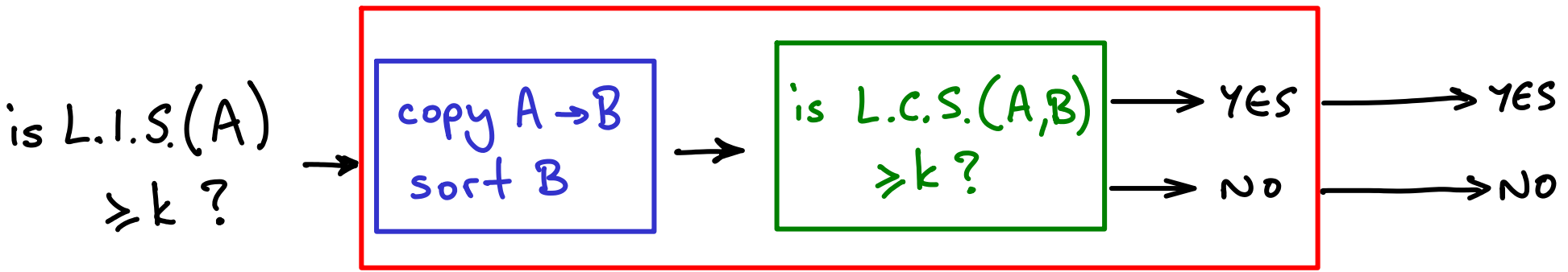
solve A using B : $T_R + T_B$

REDUCING ONE (DECISION) PROBLEM TO ANOTHER

A B



solve A using B : $T_R + T_B$



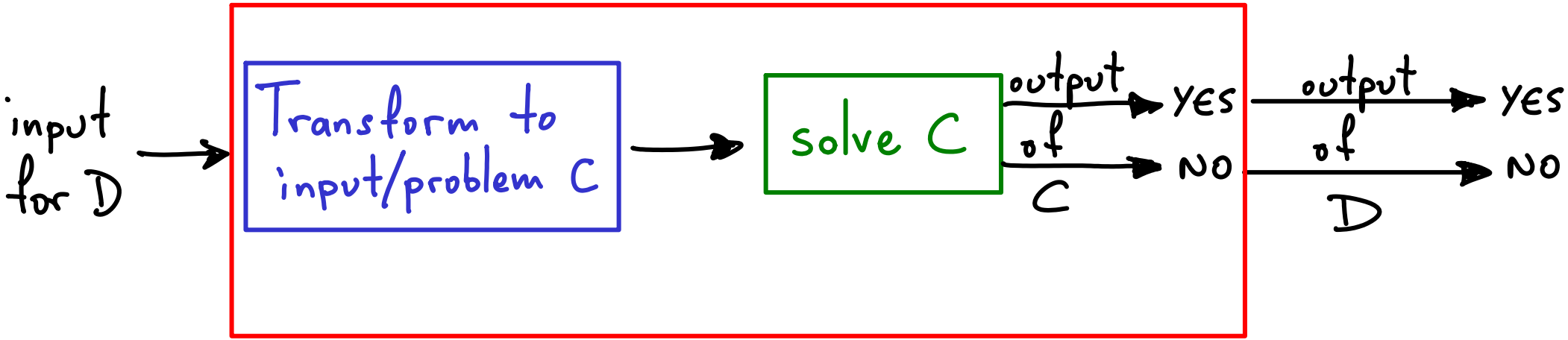
LOWER BOUND FOR ONE (DECISION) PROBLEM VIA ANOTHER

C D

solve D: known $\Omega(f(n))$

LOWER BOUND FOR ONE (DECISION) PROBLEM VIA ANOTHER

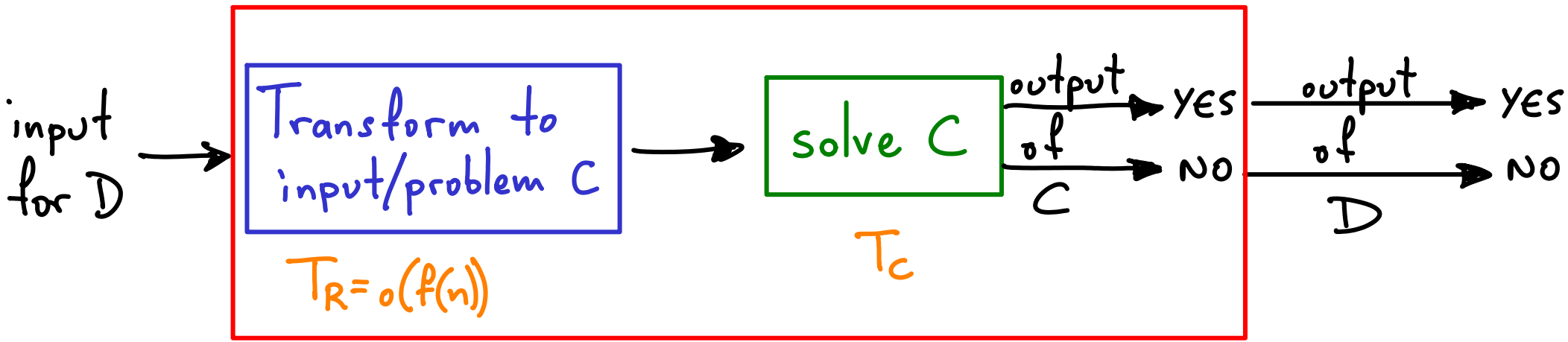
C D



solve D: known $\Omega(f(n))$

LOWER BOUND FOR ONE (DECISION) PROBLEM VIA ANOTHER

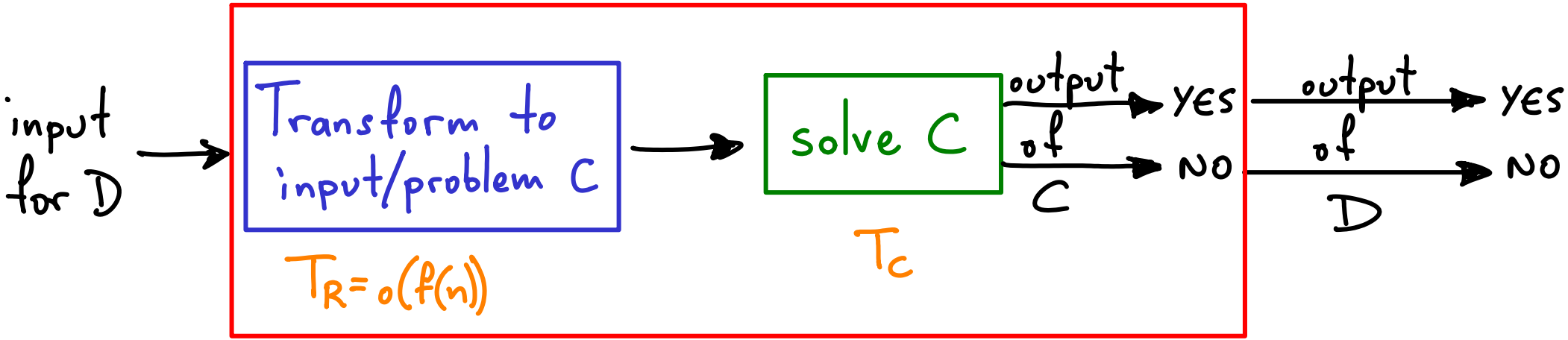
C D



solve D: known $\Omega(f(n))$ = $T_R + T_C$

LOWER BOUND FOR ONE (DECISION) PROBLEM VIA ANOTHER

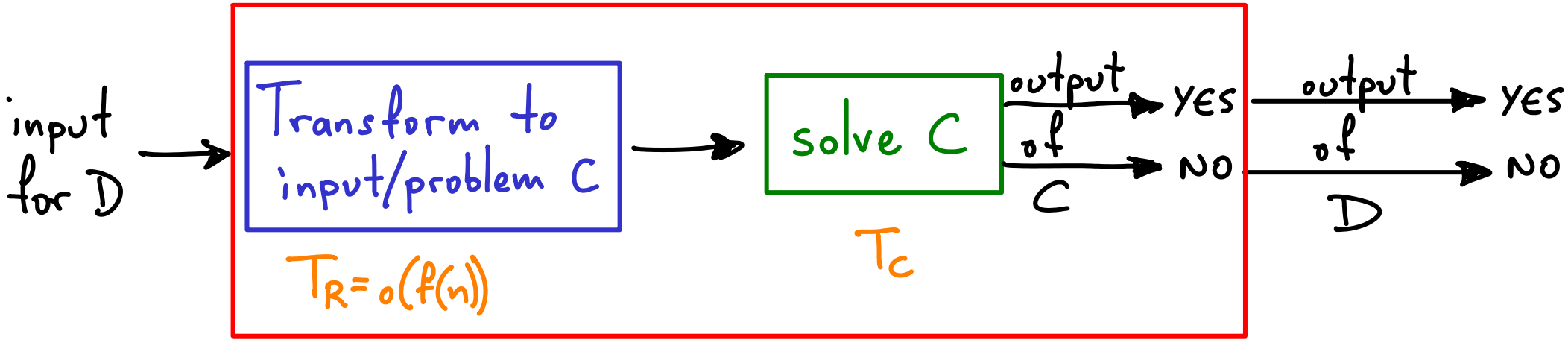
C D



solve D: known $\Omega(f(n)) = T_R + T_C \Rightarrow T_C = \Omega(f(n))$

LOWER BOUND FOR ONE (DECISION) PROBLEM VIA ANOTHER

C D



solve D: known $\Omega(f(n))$ $= T_R + T_C \Rightarrow T_C = \Omega(f(n))$

- quick transformation T_R $\Rightarrow$ C is at least as difficult as D

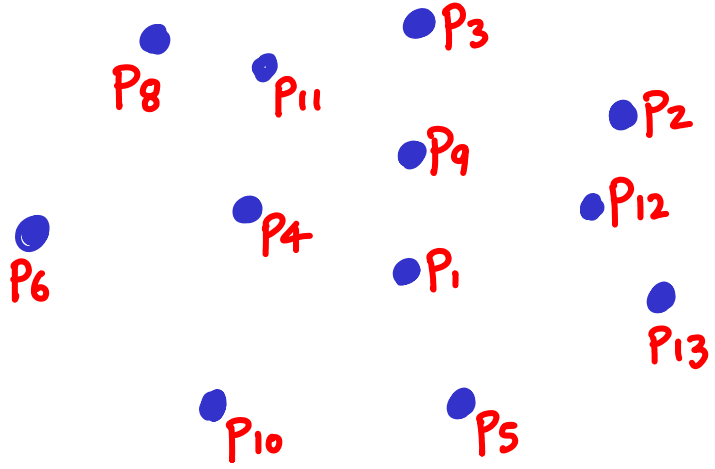
(fast solution for C $\Rightarrow$ fast solution for D)

The CONVEX HULL problem (definition by picture)

C.H. input: list of (2D) points

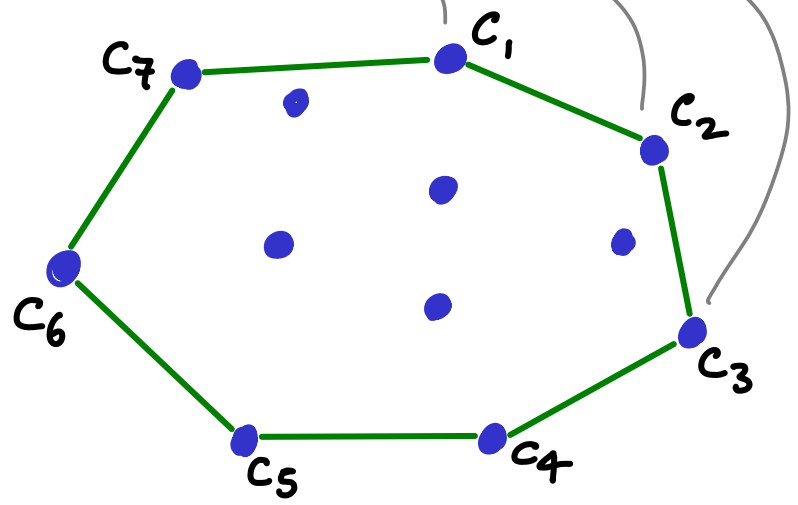
[p_{10} p_5 p_3 p_{12} p_1 ... etc...]

(x_{12}, y_{12})



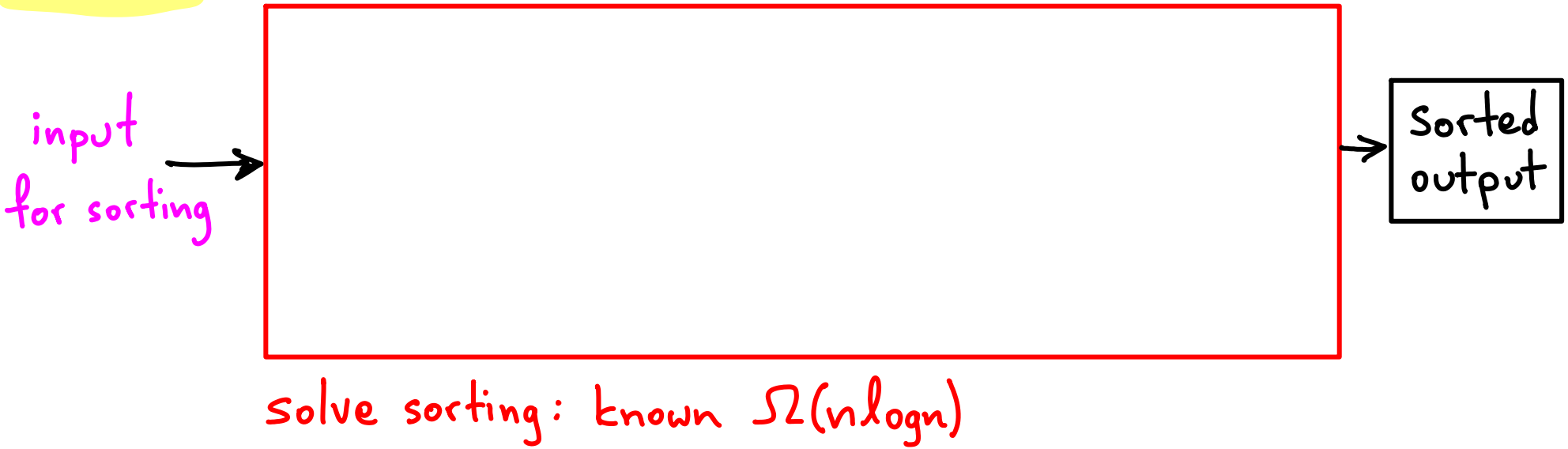
C.H. output: a graph (cycle) containing only the "extreme" points

[p_3 p_2 p_{13} p_5 p_{10} p_6 p_8]



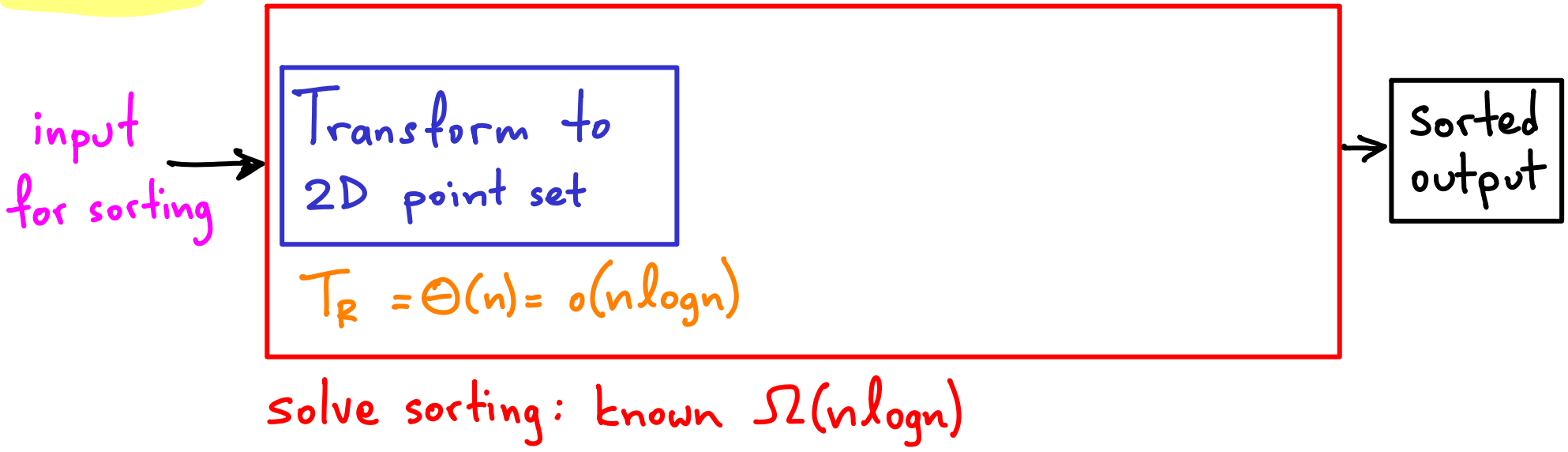
LOWER BOUND FOR ONE ~~(DECISION)~~ PROBLEM VIA ANOTHER

Example

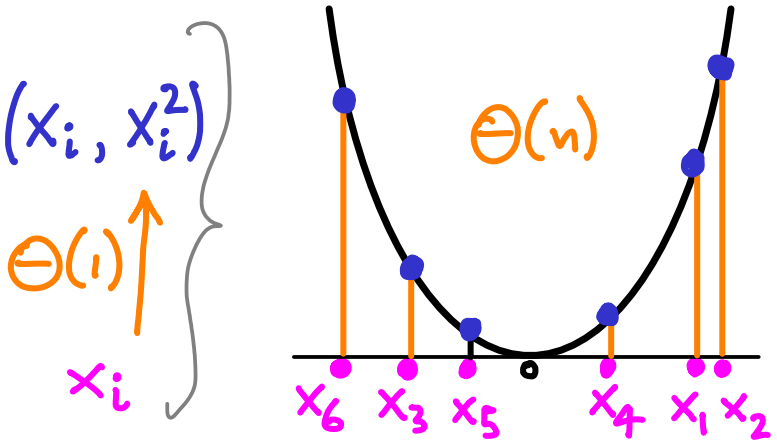


LOWER BOUND FOR ONE ~~(DECISION)~~ PROBLEM VIA ANOTHER

Example

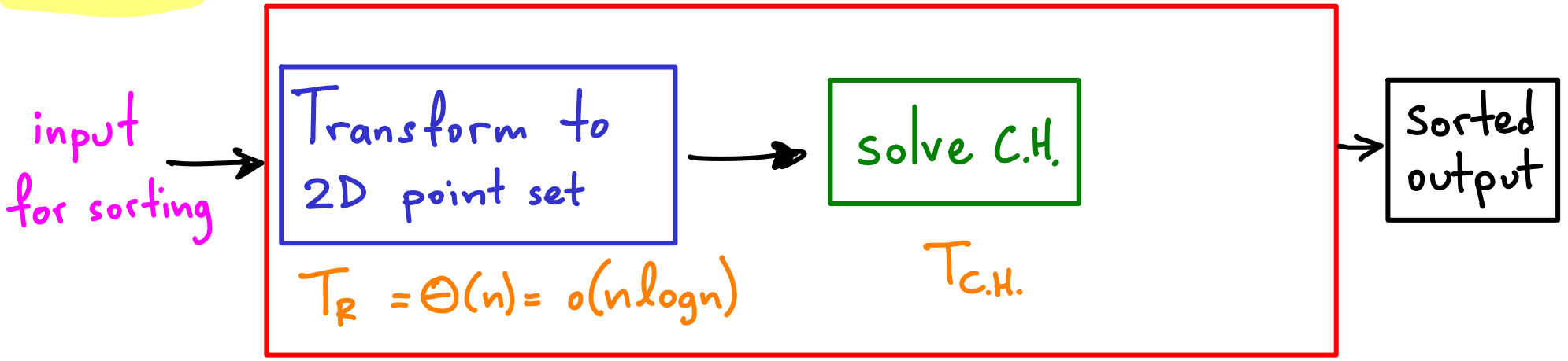


solve sorting: known $\Omega(n \log n)$

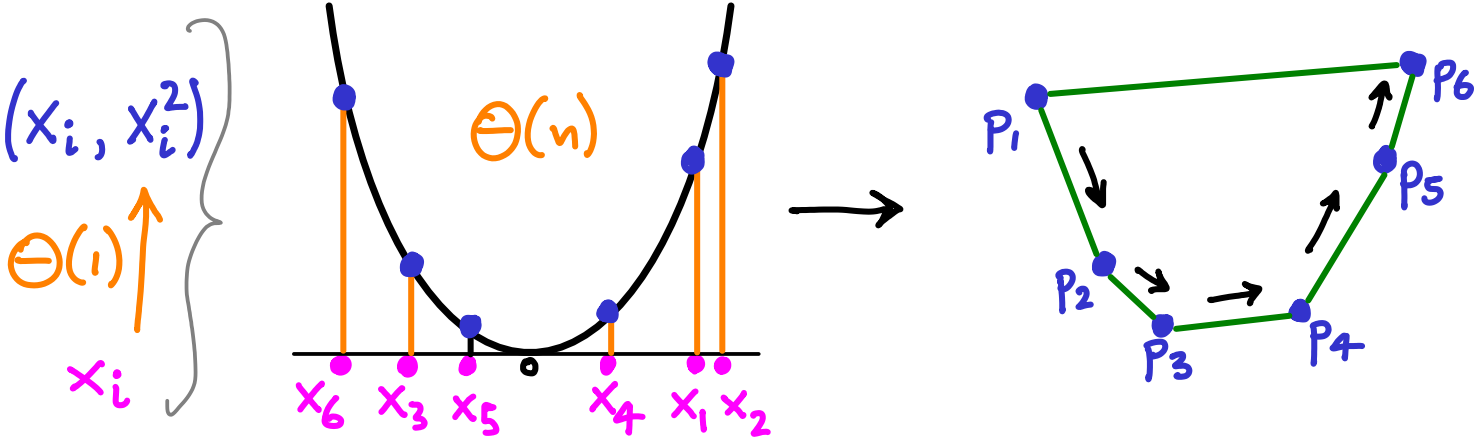


LOWER BOUND FOR ONE ~~(DECISION)~~ PROBLEM VIA ANOTHER

Example

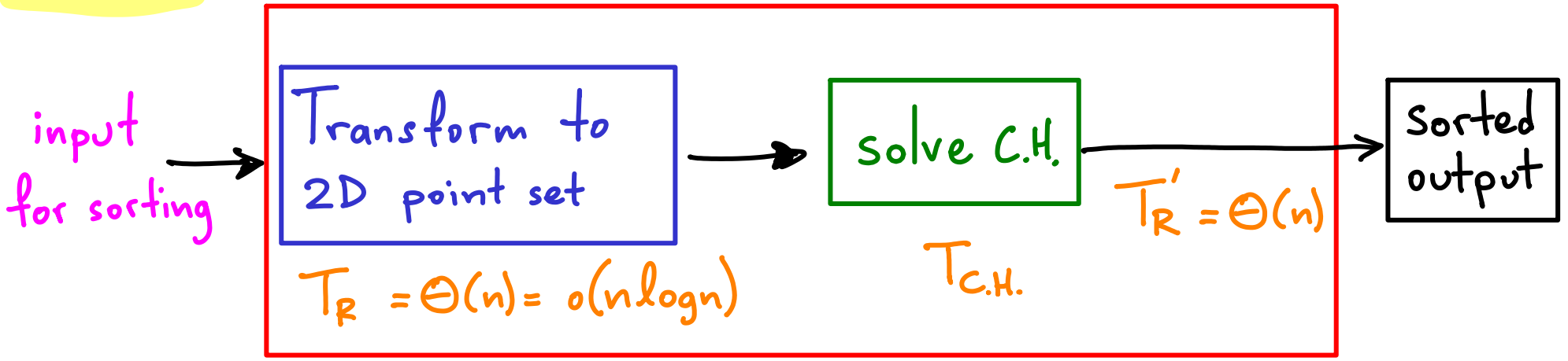


solve sorting: known $\Omega(n \log n)$

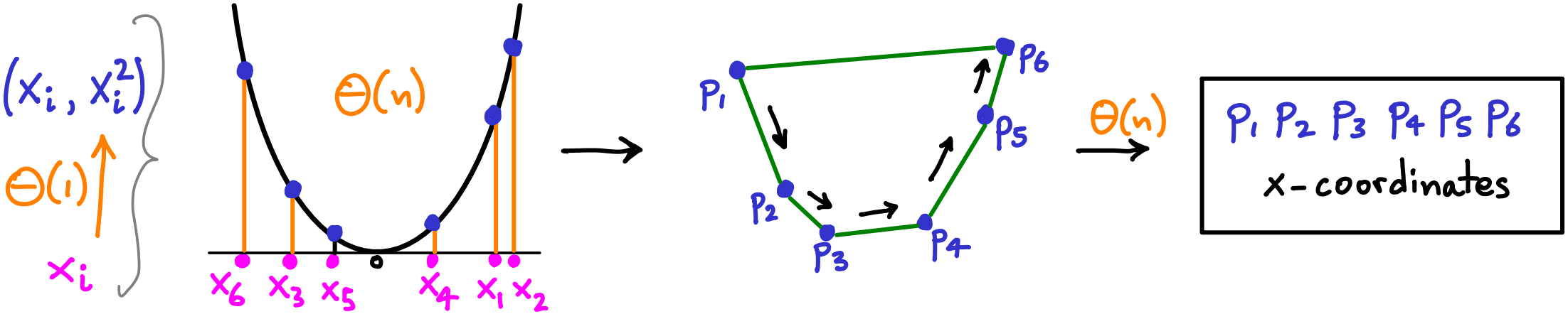


LOWER BOUND FOR ONE ~~(DECISION)~~ PROBLEM VIA ANOTHER

Example

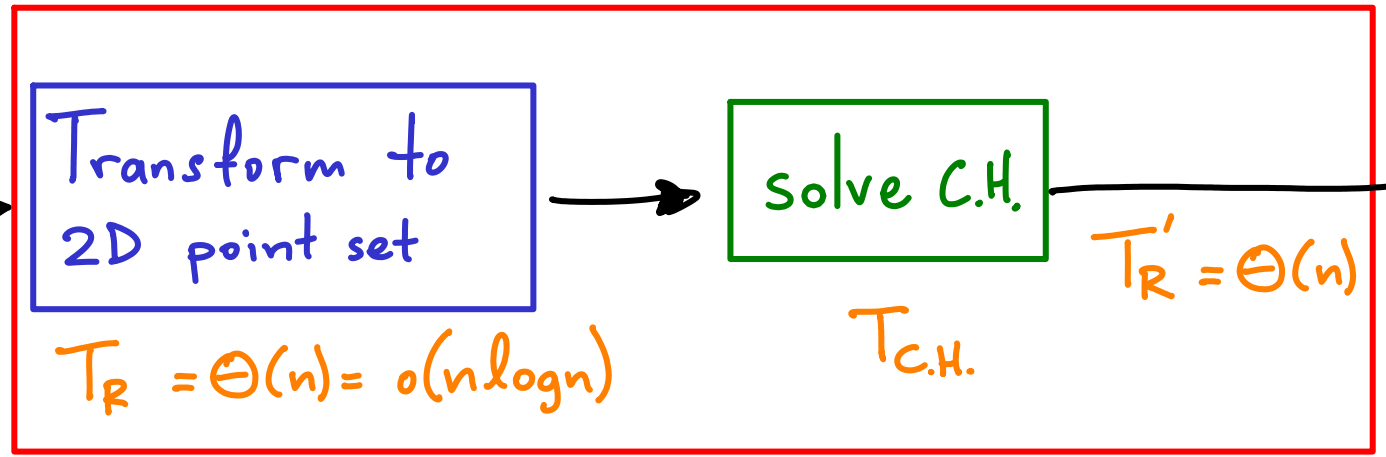


solve sorting: known $\Omega(n \log n)$

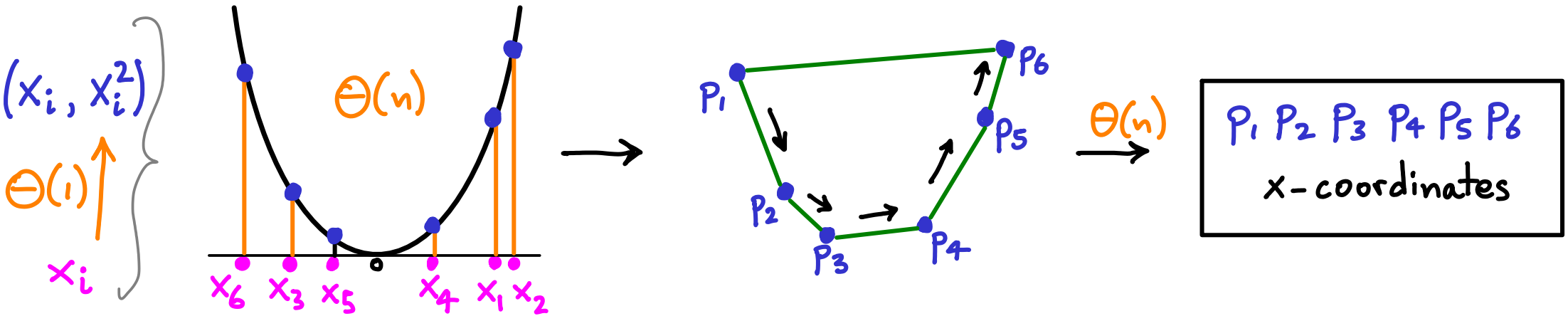


LOWER BOUND FOR ONE ~~(DECISION)~~ PROBLEM VIA ANOTHER

Example



solve sorting: known $\Omega(n \log n) = T_R + T_R' + T_{C.H.} \Rightarrow T_{C.H.} = \Omega(n \log n)$



NPC REDUCTION

FROM ONE (DECISION) PROBLEM TO ANOTHER

D

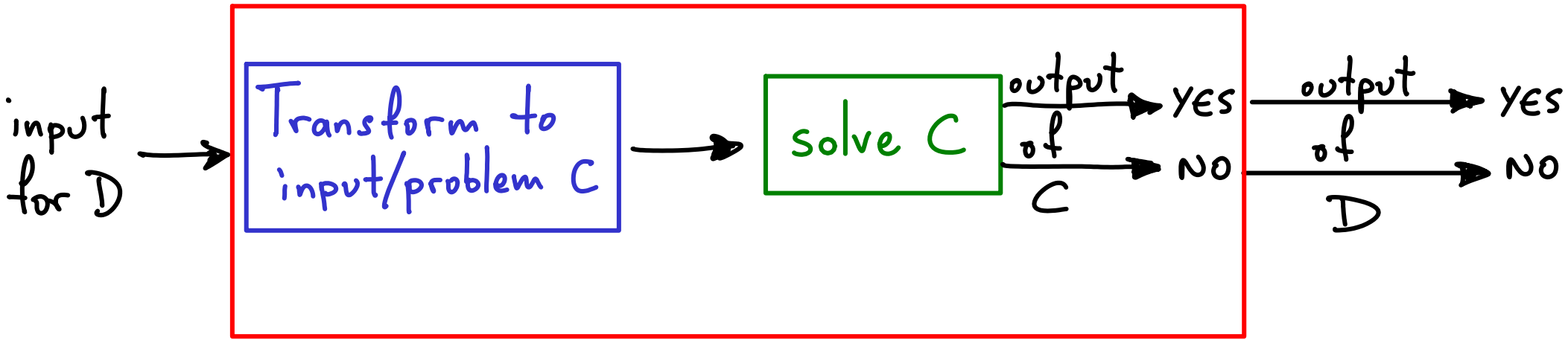
C

NPC REDUCTION FROM ONE (DECISION) PROBLEM TO ANOTHER

D C

NPC REDUCTION FROM ONE (DECISION) PROBLEM TO ANOTHER

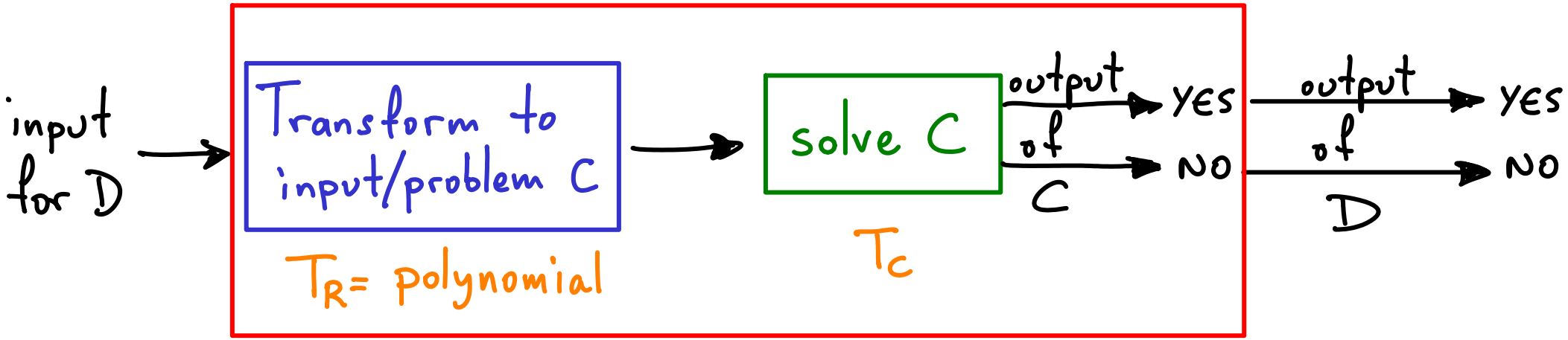
D C



D: known NPC

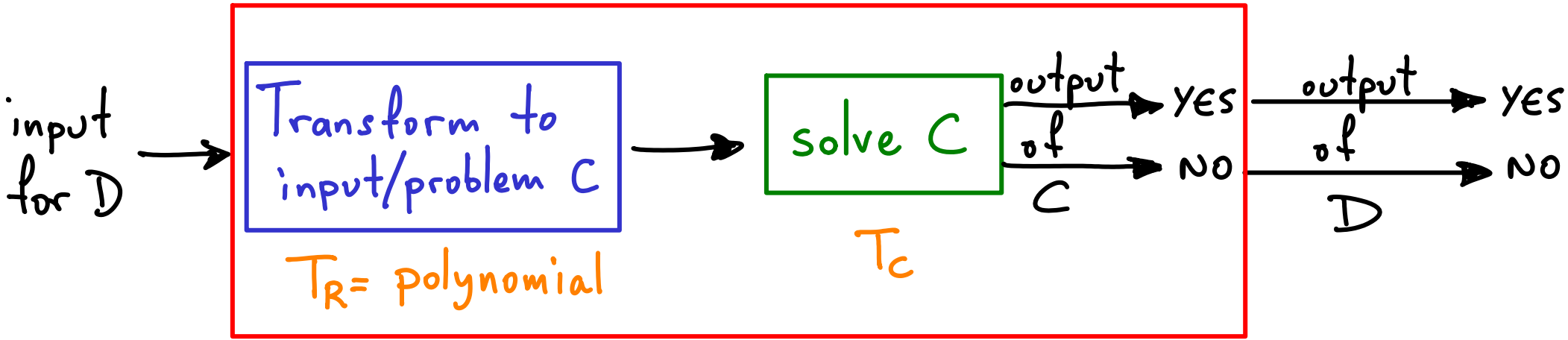
NPC REDUCTION FROM ONE (DECISION) PROBLEM TO ANOTHER

D C



NPC REDUCTION FROM ONE (DECISION) PROBLEM TO ANOTHER

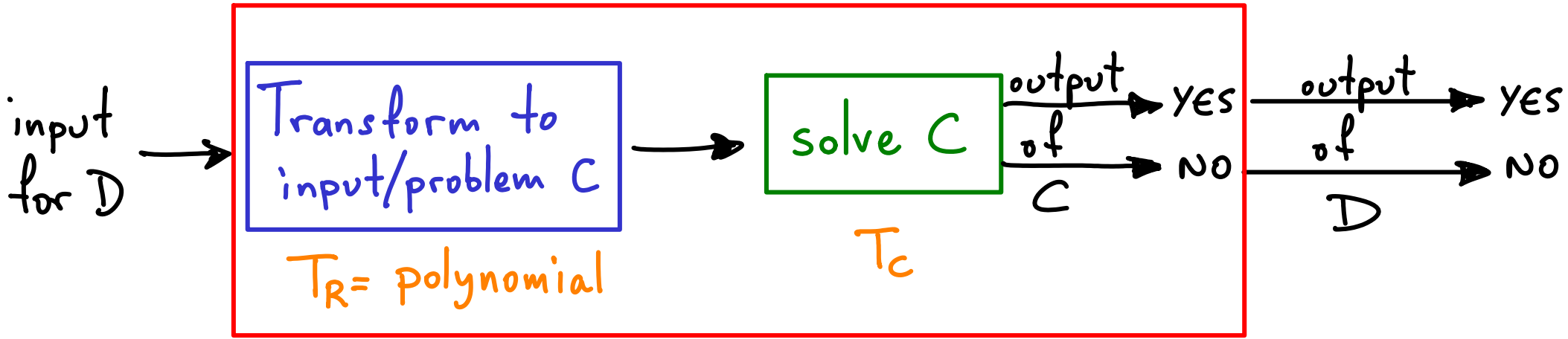
D C



solve D: known NPC $= T_R + T_C =$ poly-time $+ T_C$

NPC REDUCTION FROM ONE (DECISION) PROBLEM TO ANOTHER

D C



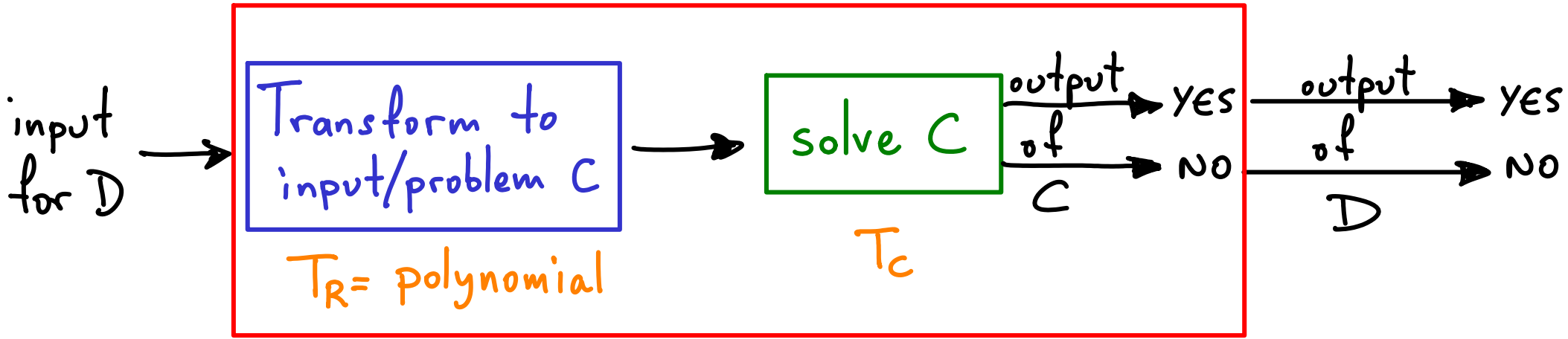
solve D: known NPC $= T_R + T_C =$ poly-time $+ T_C$

$\Rightarrow$ C is NP-hard

(if T_C is polynomial $\Rightarrow$ P = NP)

NPC REDUCTION FROM ONE (DECISION) PROBLEM TO ANOTHER

D C



solve D: known NPC $= T_R + T_C =$ poly-time $+ T_C$

$\Rightarrow$ C is NP-hard

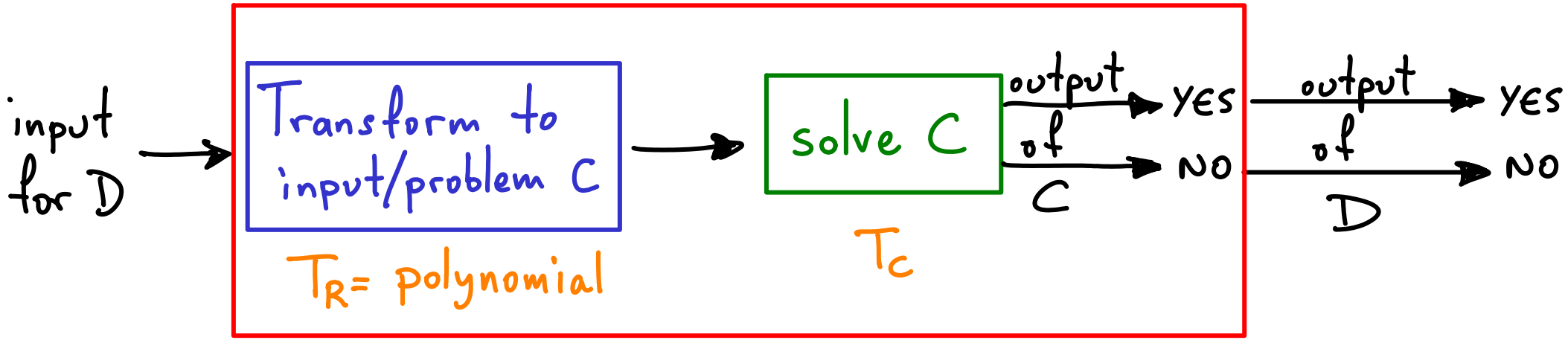
(if T_C is polynomial $\Rightarrow$ P=NP)

& if C is in NP then C is NPC

NPC REDUCTION FROM ONE (DECISION) PROBLEM TO ANOTHER

D C

~LIKE ESTABLISHING AN ALMOST CERTAIN LOWER BOUND FOR C.



solve D: known NPC $= T_R + T_C =$ poly-time $+ T_C$

$\Rightarrow$ C is NP-hard

(if T_C is polynomial $\Rightarrow$ P=NP)

& if C is in NP then C is NPC

$A \leq_p B$: in polynomial time, A can be reduced to B

$A \leq_p B$: in polynomial time, A can be reduced to B

↳ B is at least as hard as A.

Solving B in poly-time ⇒ solving A in poly-time

$A \leq_p B$: in polynomial time, A can be reduced to B

↳ B is at least as hard as A.

Solving B in poly-time $\Rightarrow$ solving A in poly-time

If $\{$every problem in NP$\} \leq_p B$ then B is NP-hard

and if B is in NP, ... ?

$A \leq_p B$: in polynomial time, A can be reduced to B

↳ B is at least as hard as A.

Solving B in poly-time ⇒ solving A in poly-time

If {every problem in NP} $\leq_p$ B then B is NP-hard

↳ or { ... ? } (and if B is in NP, then B is NPC)

$A \leq_p B$: in polynomial time, A can be reduced to B

↳ B is at least as hard as A.

Solving B in poly-time ⇒ solving A in poly-time

If {every problem in NP} $\leq_p$ B

↳ or {any NPC problem}

then B is NP-hard

(and if B is in NP, then B is NPC)

$A \leq_p B$: in polynomial time, A can be reduced to B

↳ B is at least as hard as A.

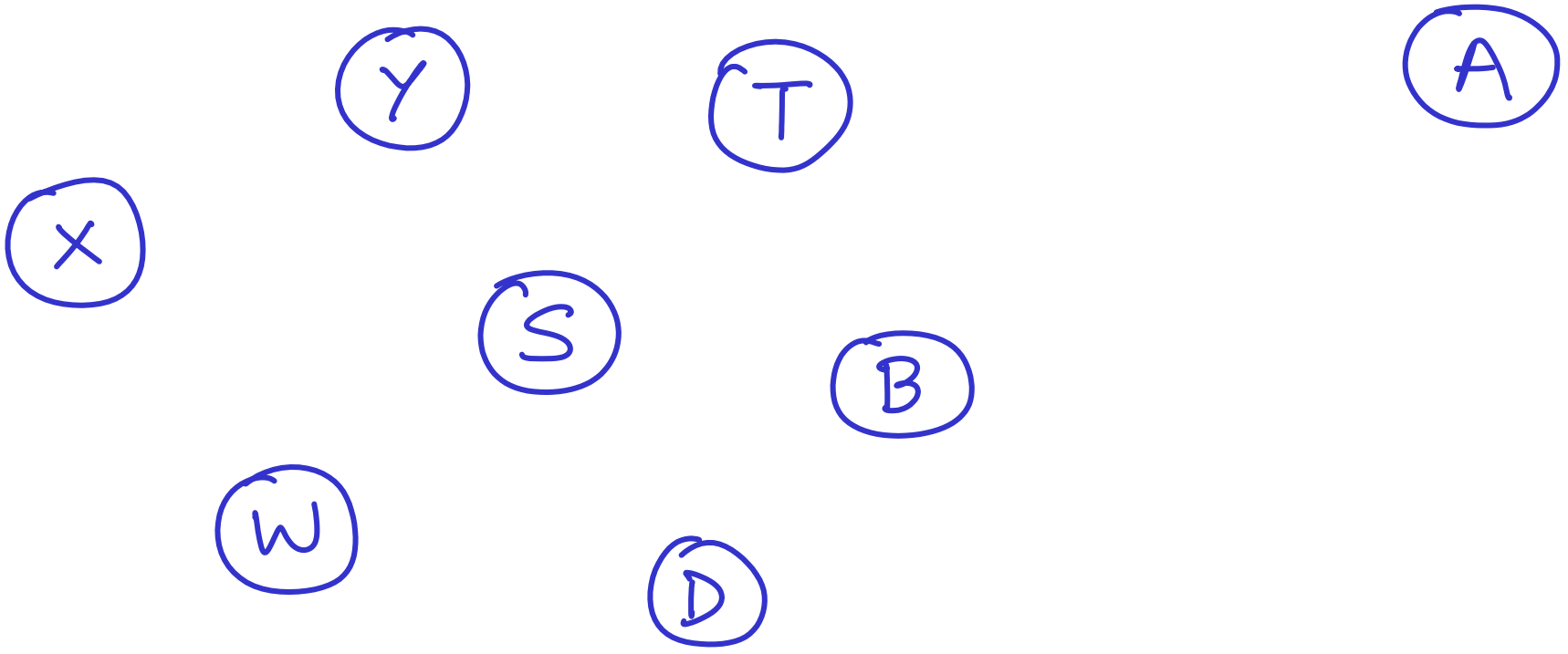
Solving B in poly-time ⇒ solving A in poly-time

If {every problem in NP} $\leq_p$ B then B is NP-hard

↳ or {any NPC problem} (and if B is in NP, then B is NPC)

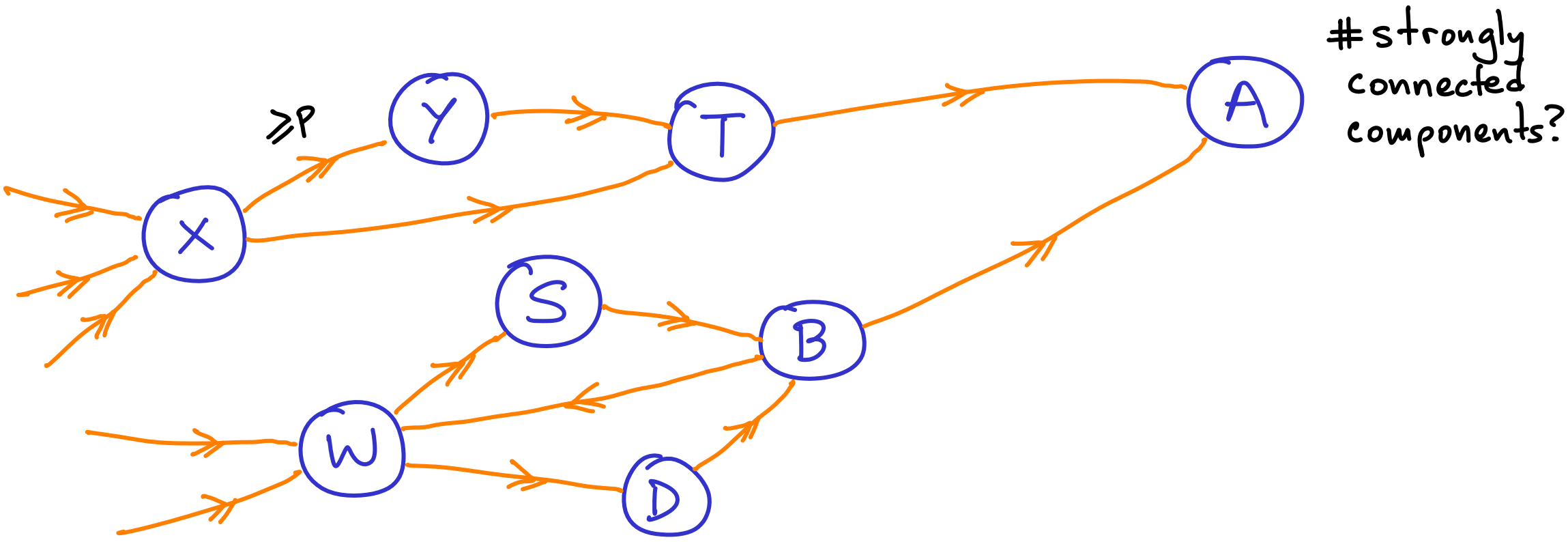
if A & B are NPC then $\underbrace{A \leq_p B \;\&\; B \leq_p A}_{\sim A =_p B}$

THOUSANDS OF NPC PROBLEMS



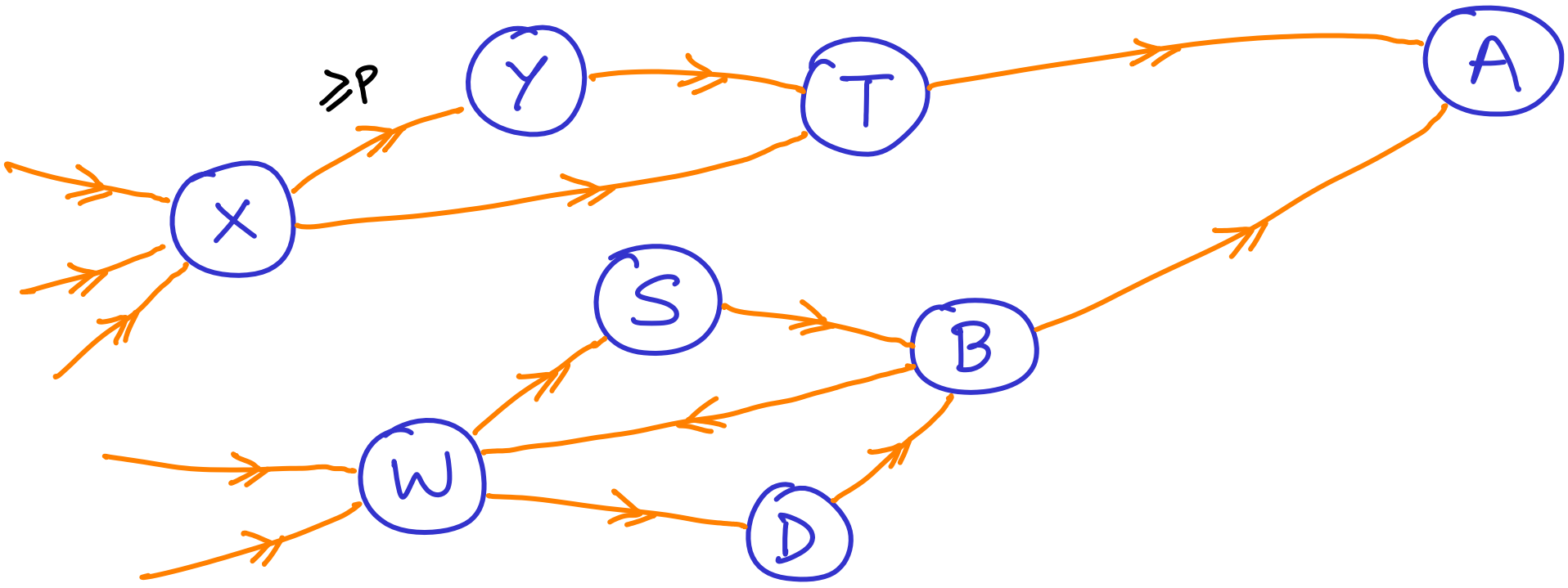
THOUSANDS OF NPC PROBLEMS

Reductions/transformations between them resemble a di-graph.



THOUSANDS OF NPC PROBLEMS

Reductions/transformations between them resemble a di-graph.

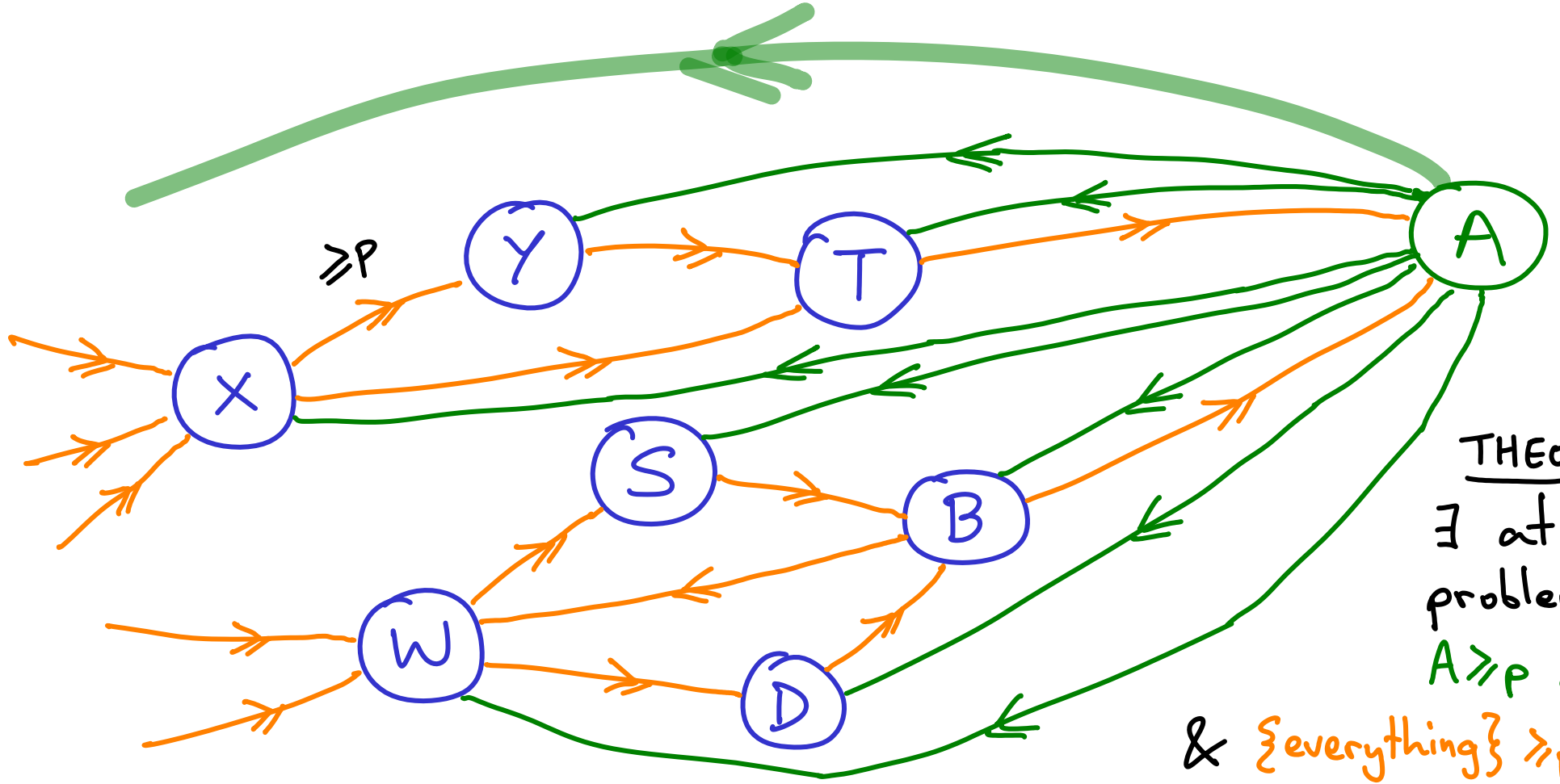


#strongly connected components?

Must be 1. precisely.

THOUSANDS OF NPC PROBLEMS

Reductions/transformations between them resemble a di-graph.



strongly connected components?

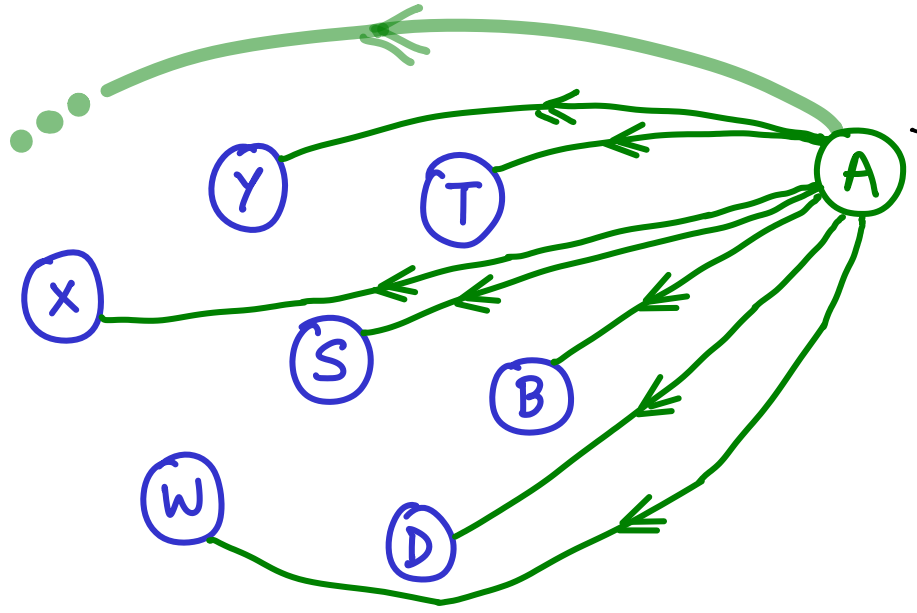
Must be 1. precisely.

THEOREM

∃ at least one problem A s.t.

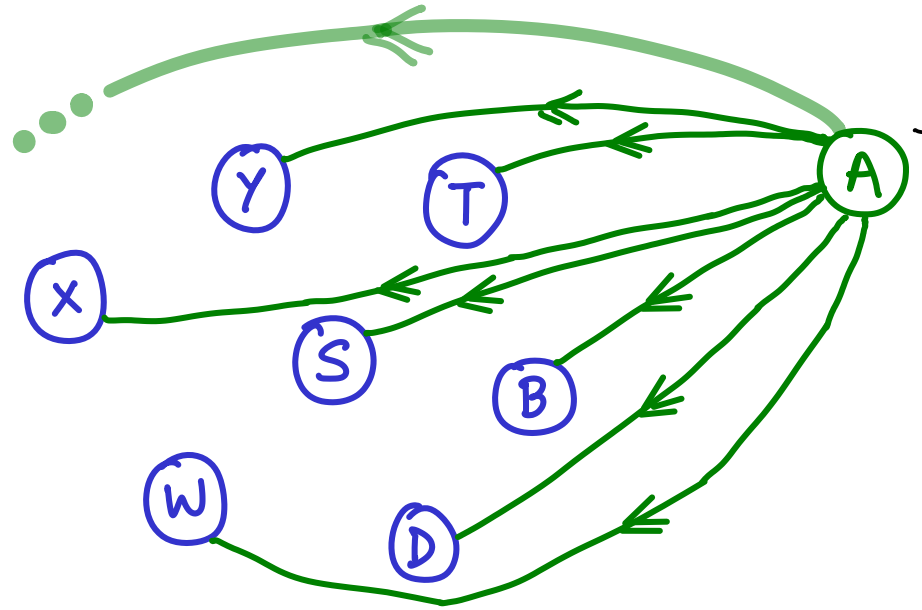
$A \geq_P \{everything\}$

& $\{everything\} \geq_P \ldots \geq_P \ldots \geq_P A$



Circuit SAT (satisfiability)

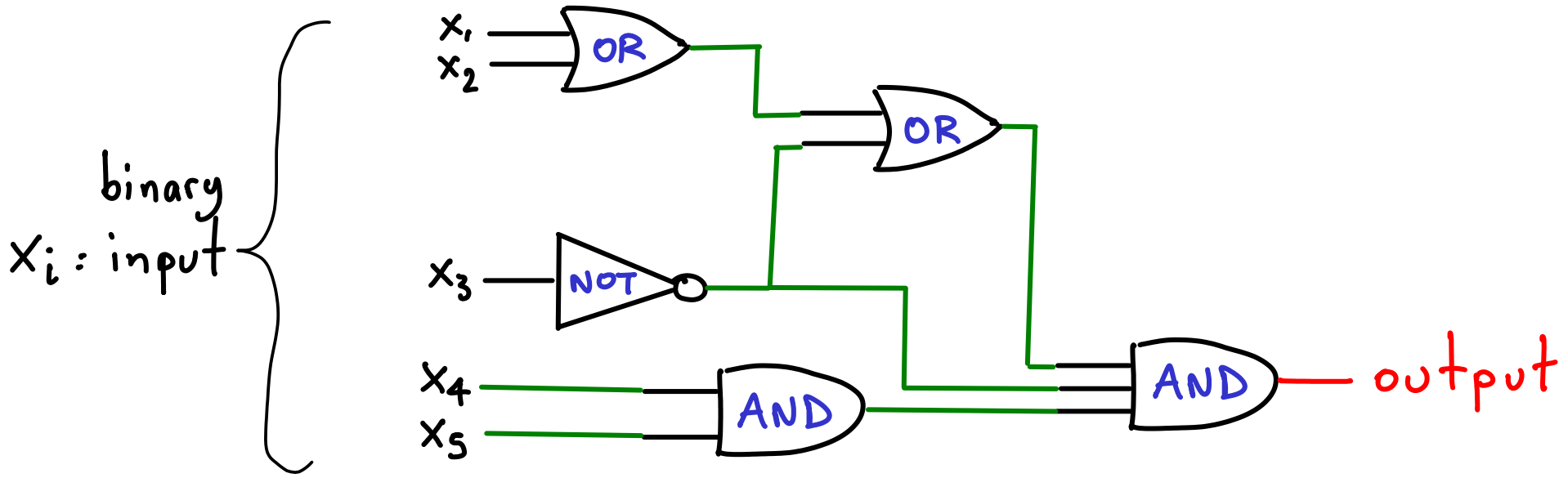
The first NPC problem.

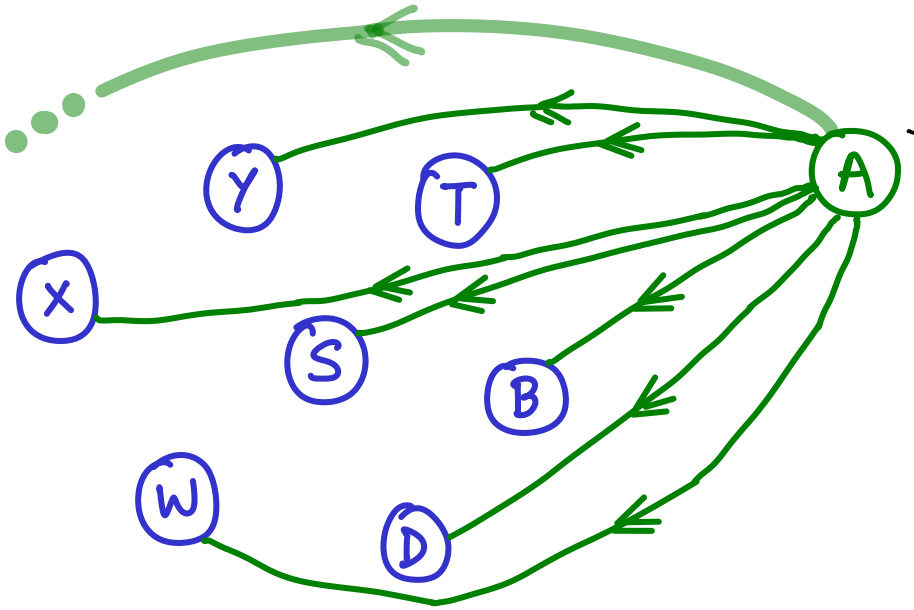


Circuit SAT (satisfiability)

The first NPC problem.

Given a circuit, can the output ever be 1?

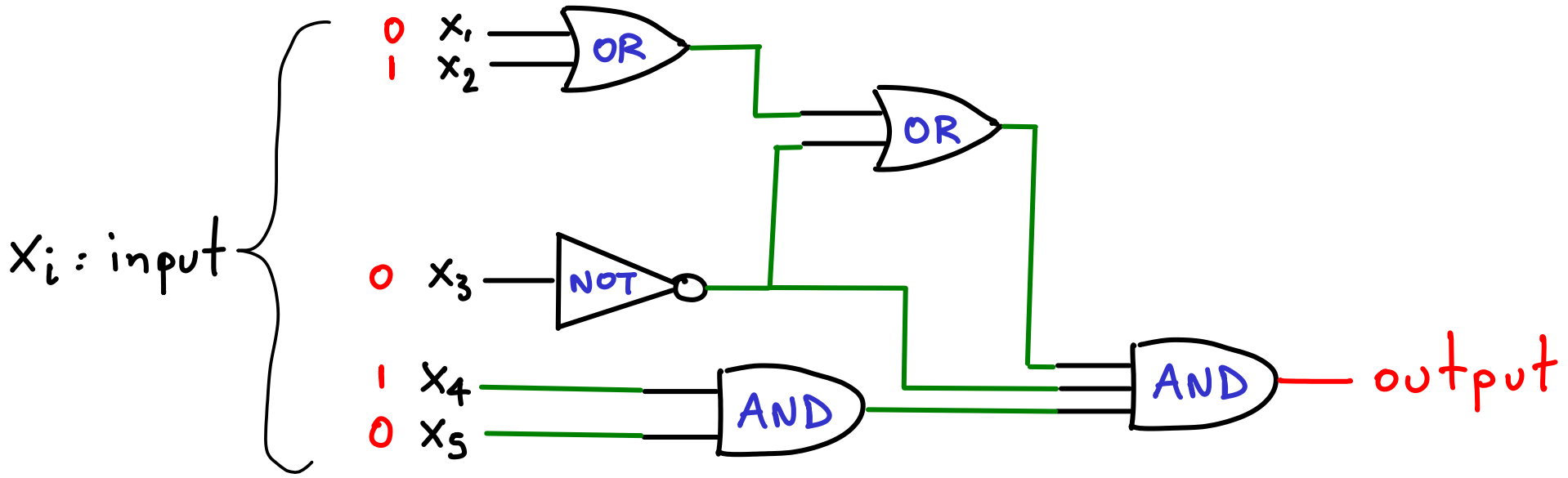


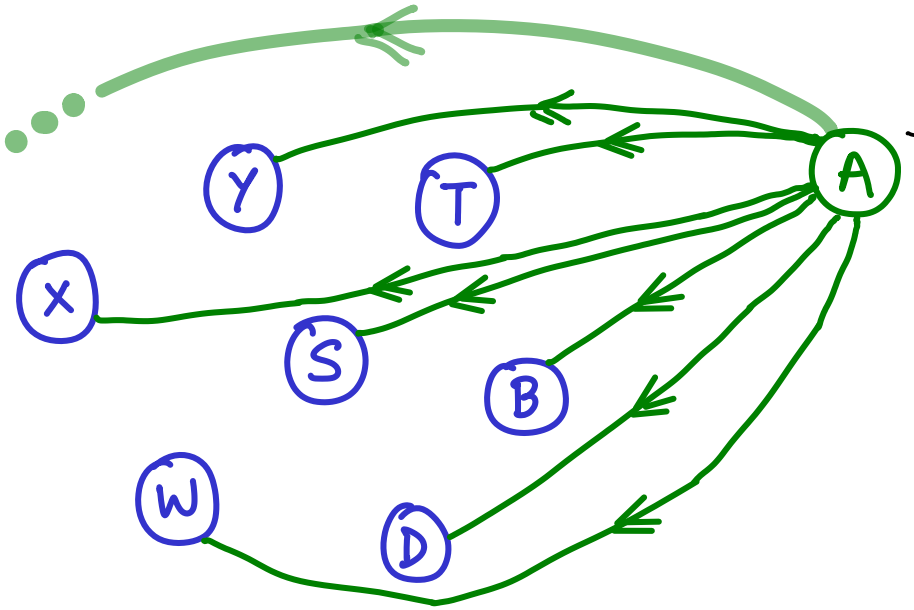


Circuit SAT (satisfiability)

The first NPC problem.

Given a circuit, can the output ever be 1?

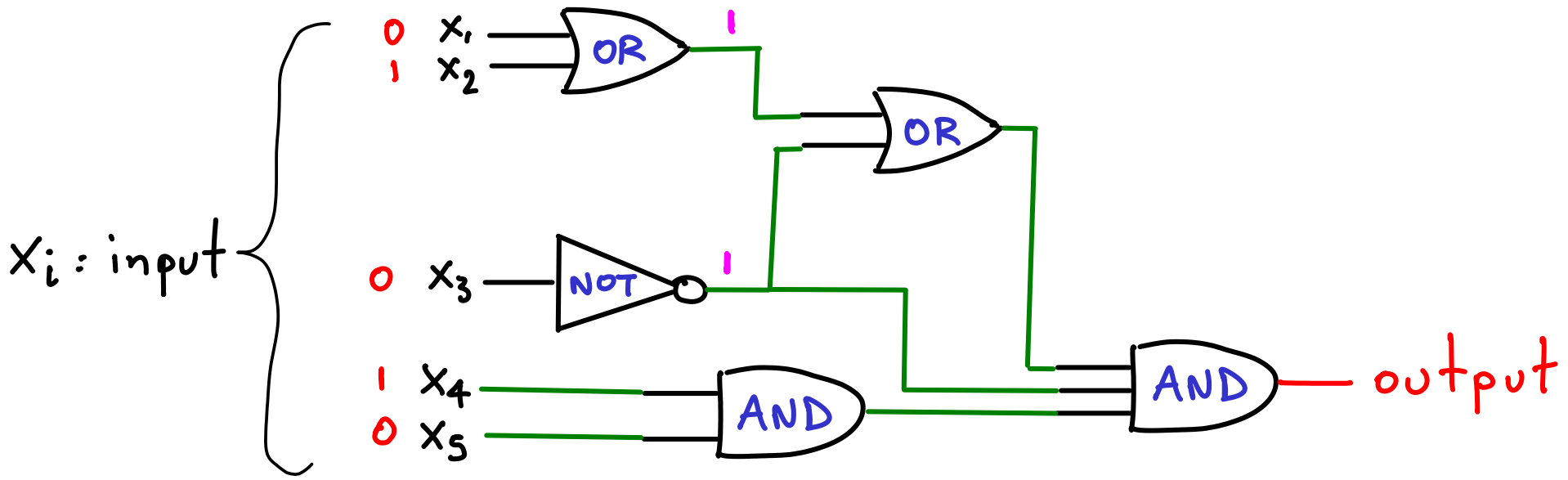


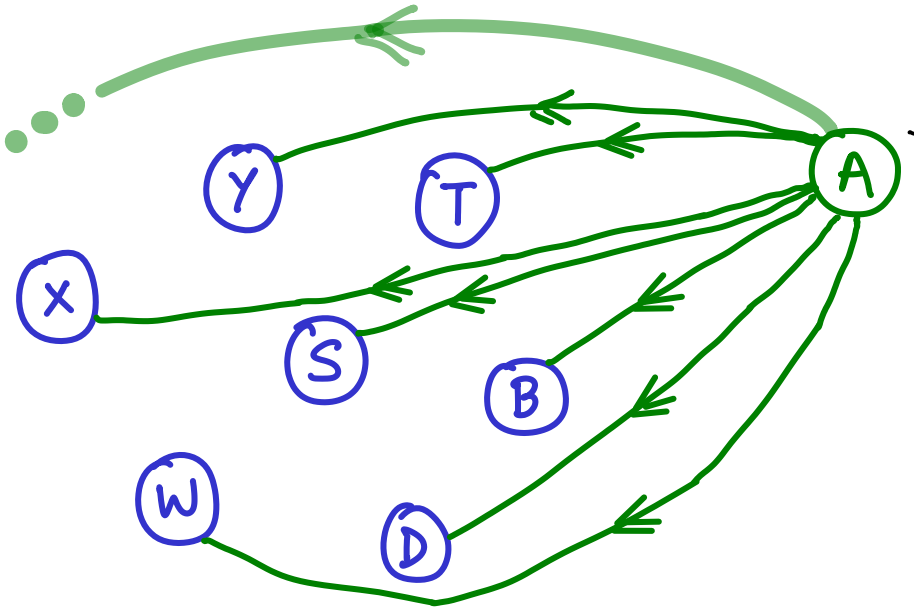


Circuit SAT (satisfiability)

The first NPC problem.

Given a circuit, can the output ever be 1?

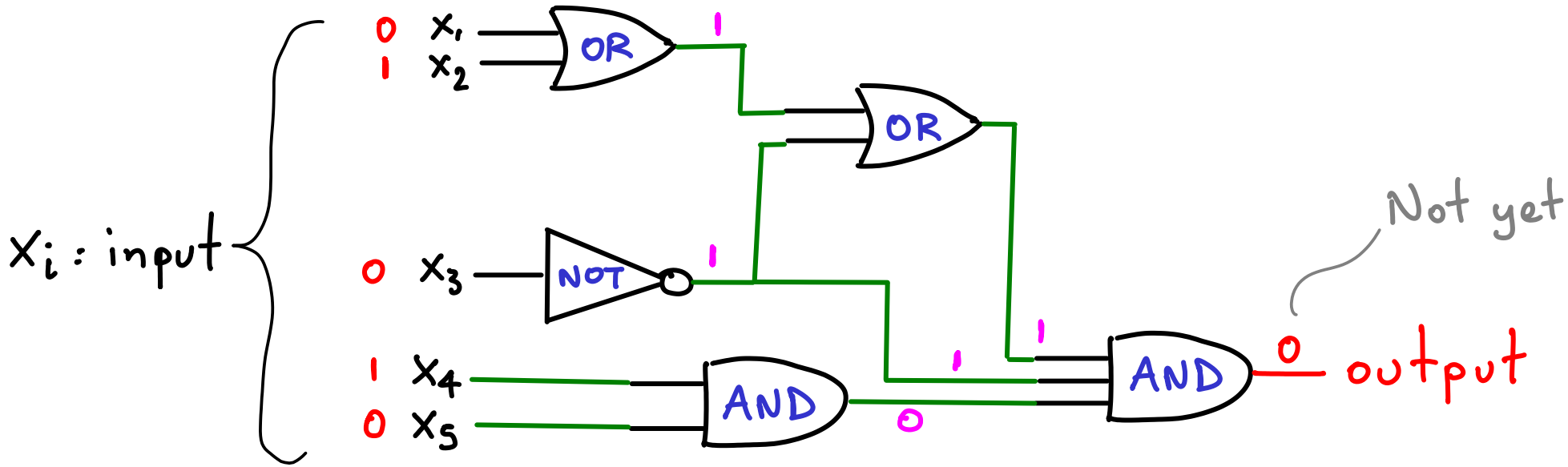


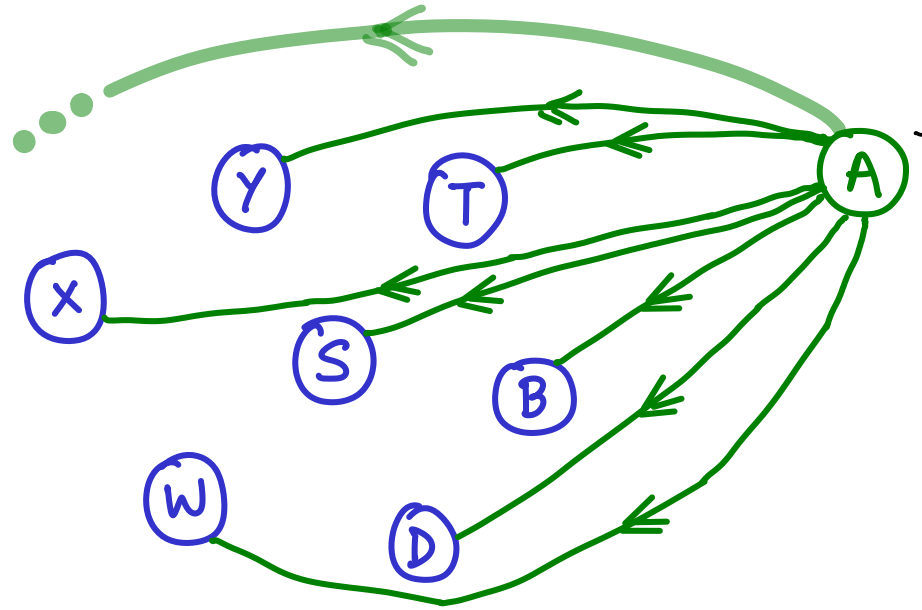


Circuit SAT (satisfiability)

The first NPC problem.

Given a circuit, can the output ever be 1?

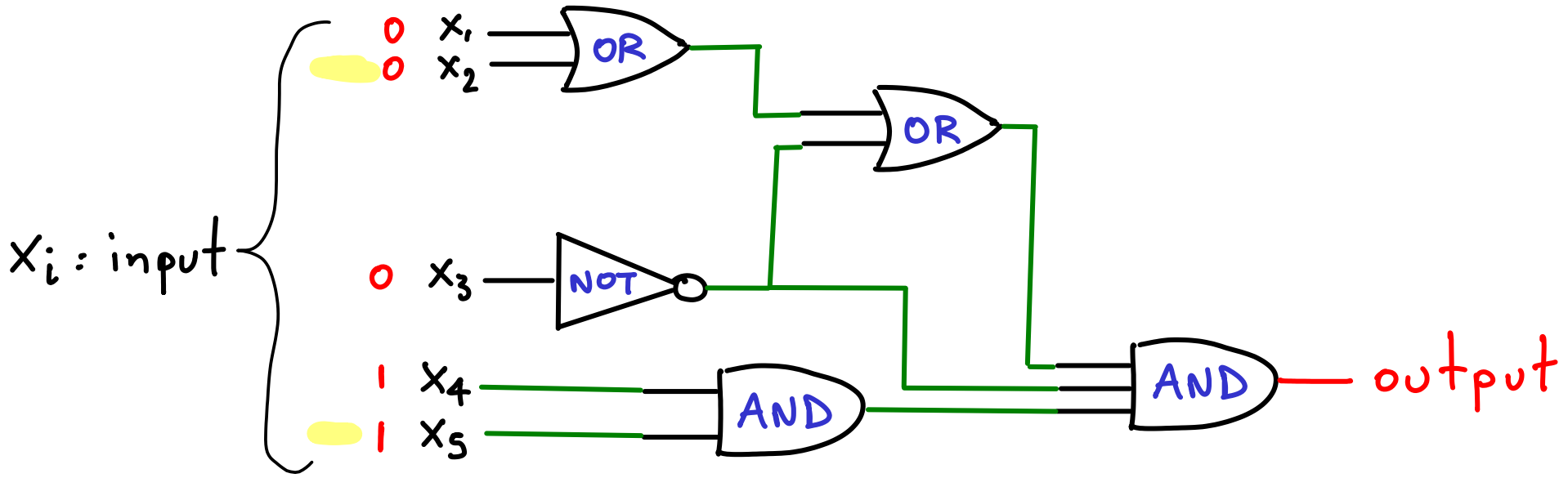


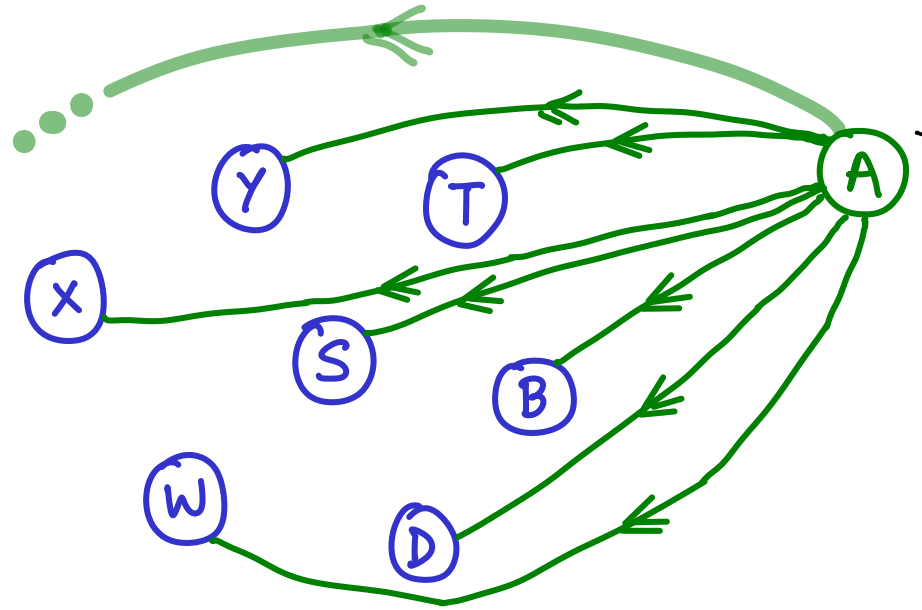


Circuit SAT (satisfiability)

The first NPC problem.

Given a circuit, can the output ever be 1?

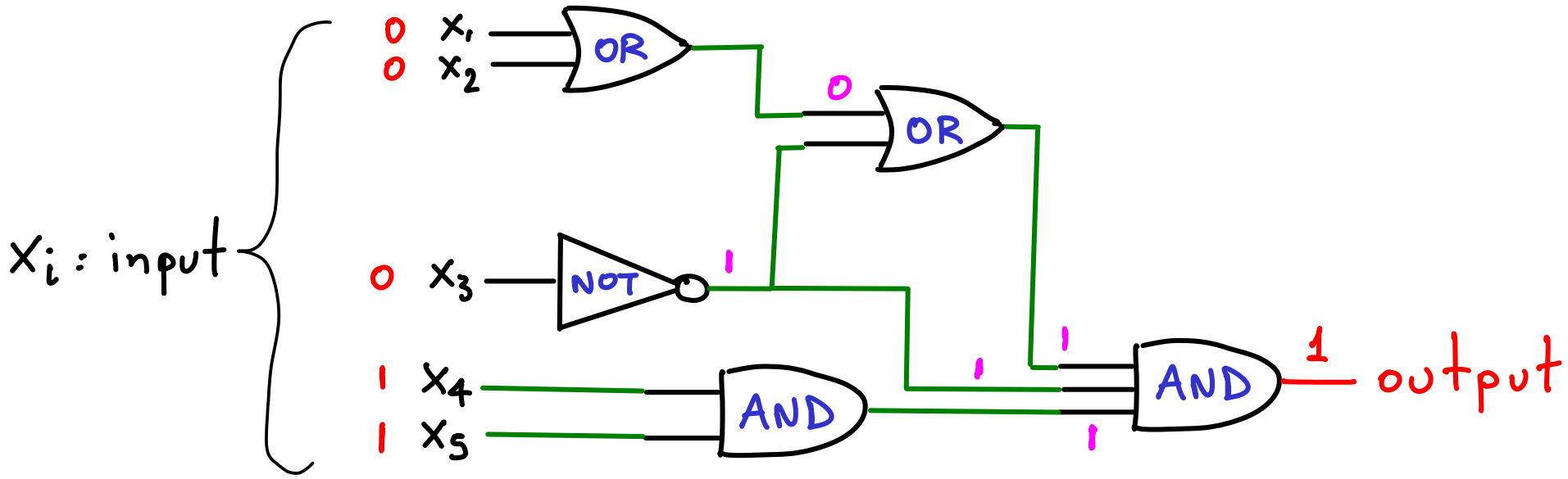


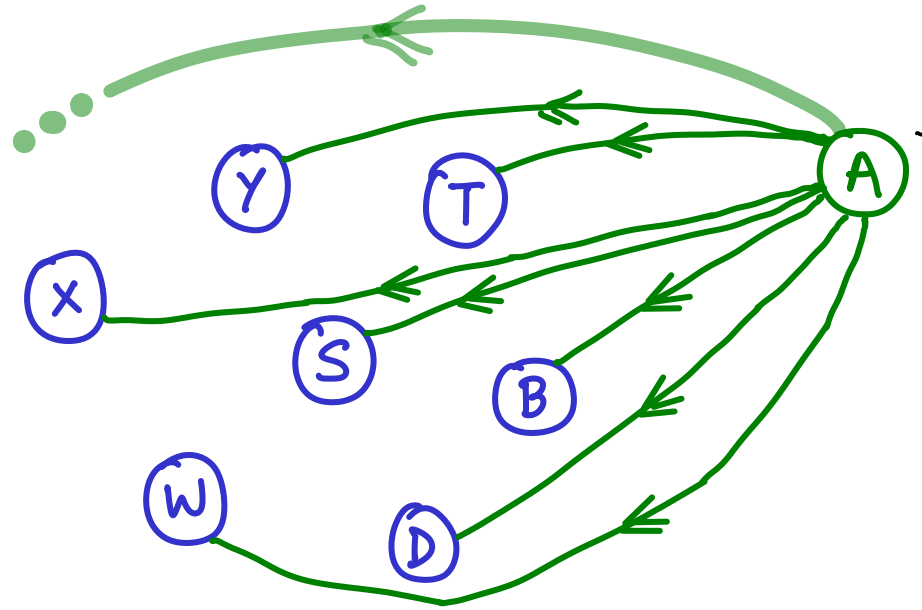


Circuit SAT (satisfiability)

The first NPC problem.

Given a circuit, can the output ever be 1?



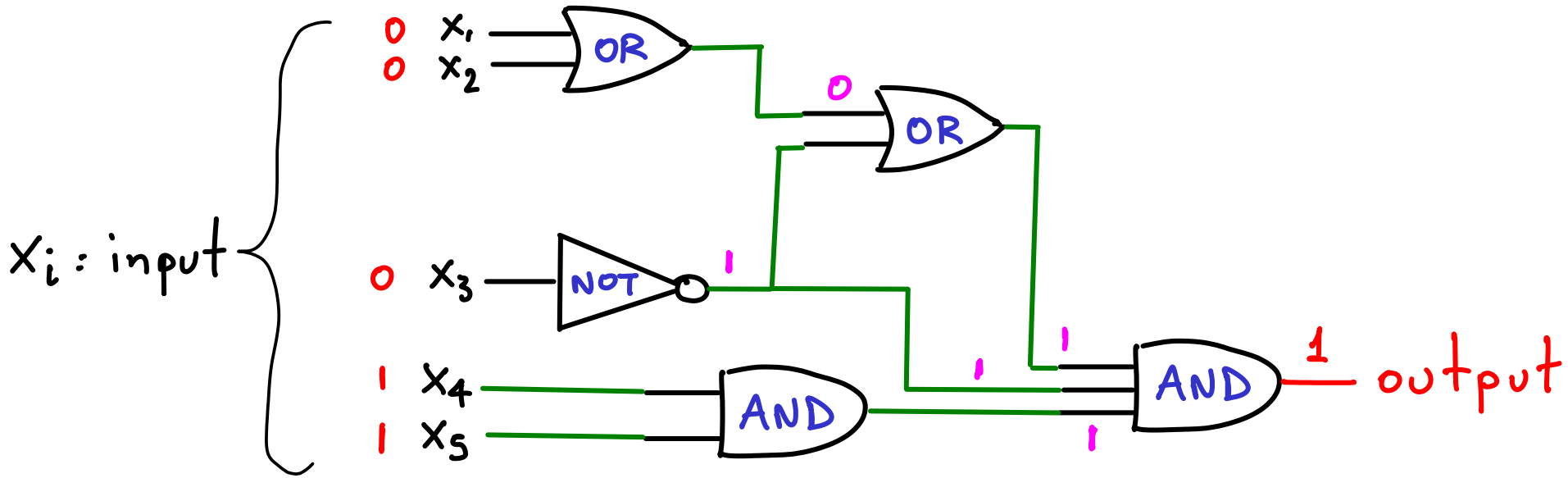


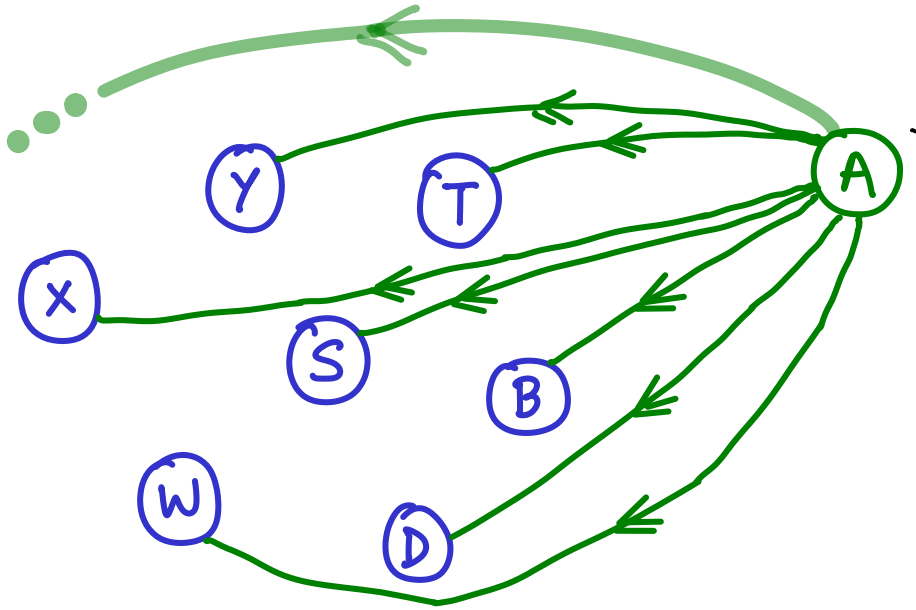
Circuit SAT (satisfiability)

The first NPC problem.

Given a circuit, can the output ever be 1?

1 - in NP ... intuitive





Circuit SAT (satisfiability)

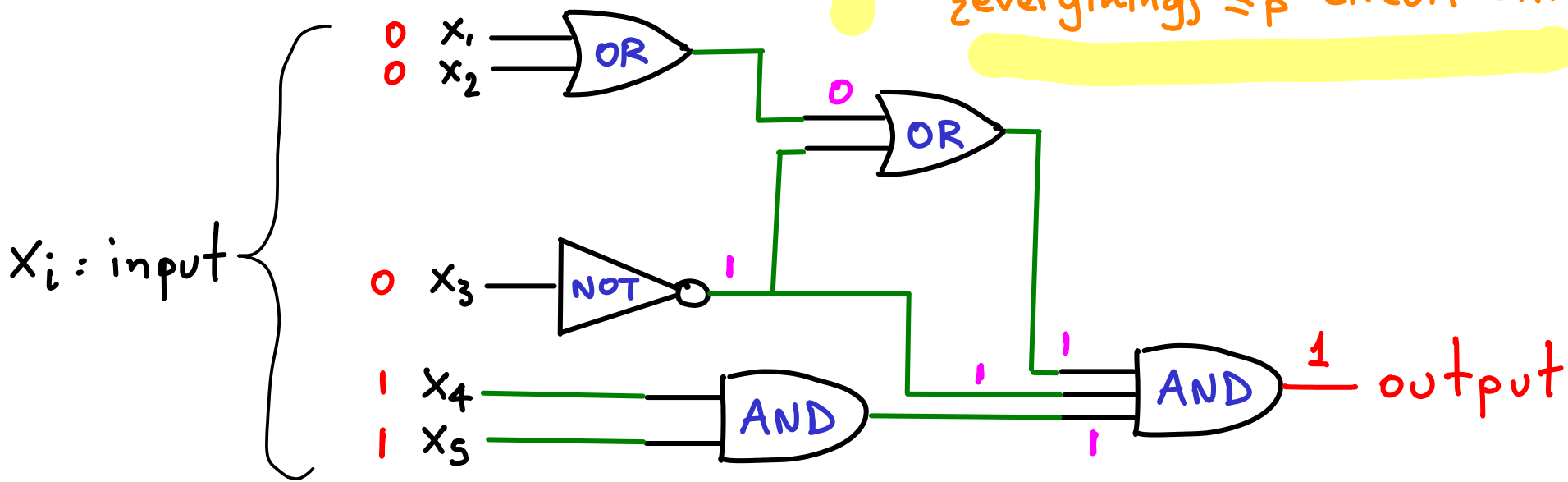
The first NPC problem.

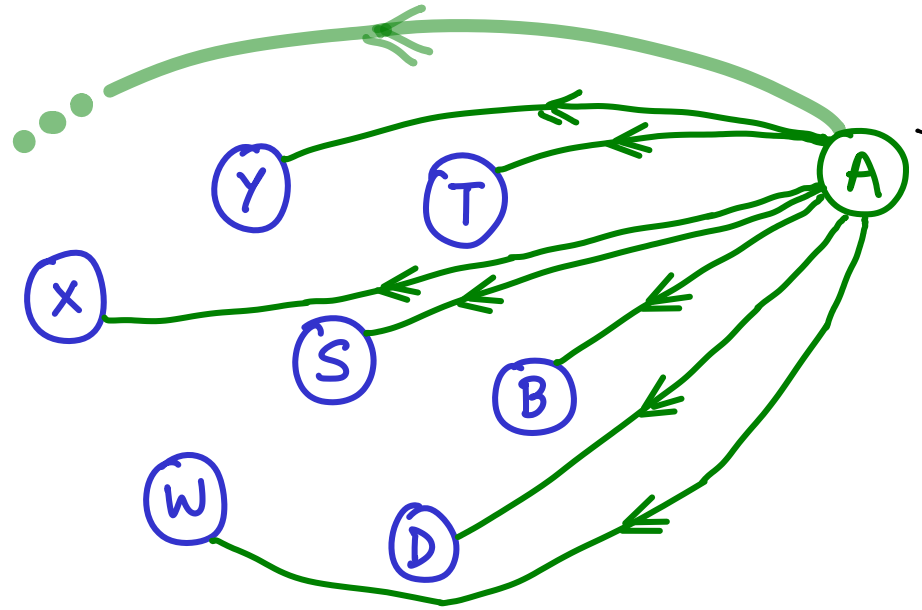
Given a circuit, can the output ever be 1?

1 - in NP ... intuitive

2 ~ every problem can be described as a circuit } technical

{everything} $\leq_P$ circuit-SAT





Circuit SAT (satisfiability)

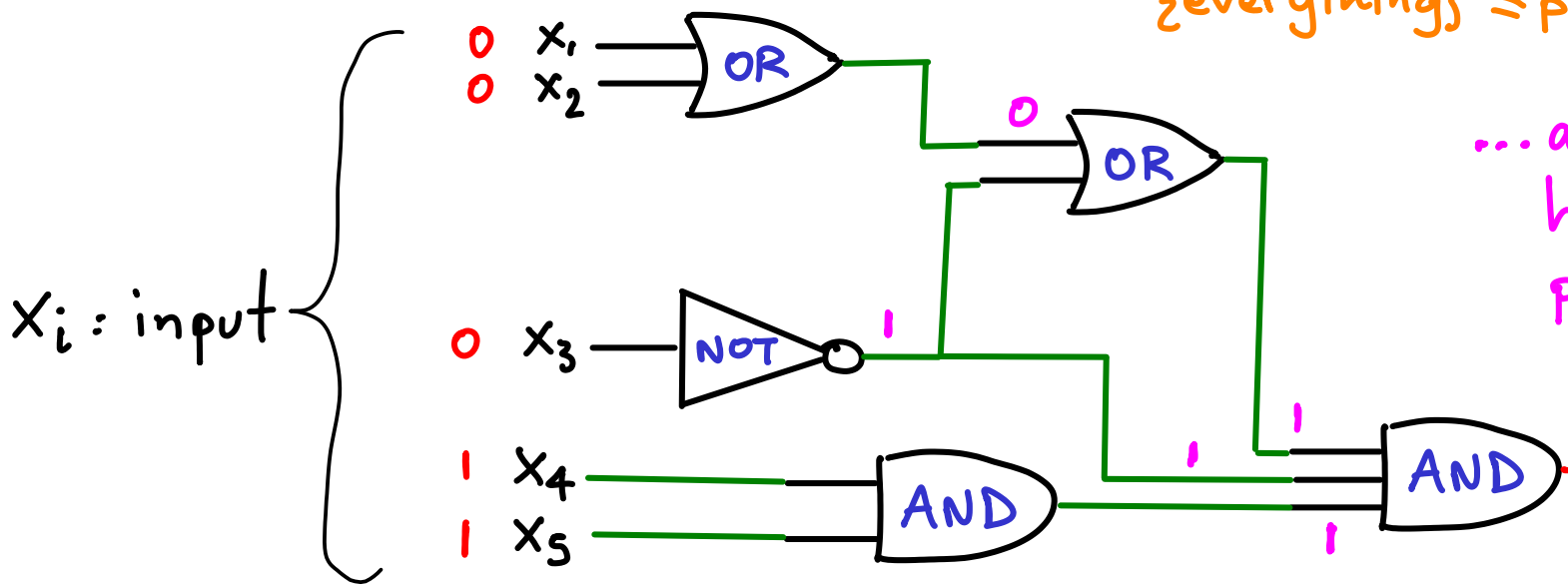
The first NPC problem.

Given a circuit, can the output ever be 1?

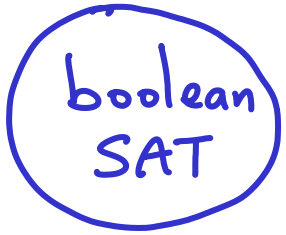
1 - in NP ... intuitive

2 ~ every problem can be described as a circuit } technical

{everything} $\leq_P$ circuit-SAT



...and we don't know how to solve this in polynomial time. (not in P)



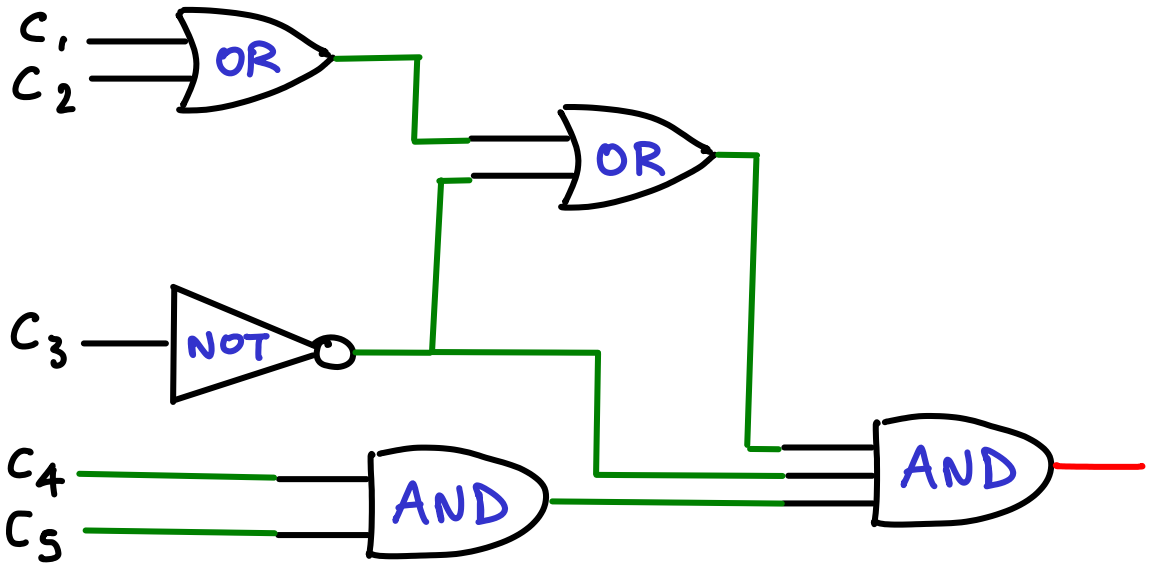
Boolean SAT $(x_1 \vee x_2 \vee \overline{x_3}) \wedge ((x_1 \leftrightarrow x_5) \vee (x_4 \rightarrow \overline{x_3}))$ $\rightarrow ? \rightarrow 1$

boolean SAT

circuit SAT

Boolean SAT $(x_1 \vee x_2 \vee \bar{x}_3) \wedge ((x_1 \leftrightarrow x_5) \vee (x_4 \rightarrow \bar{x}_3))$ $\rightarrow ? \rightarrow$ 1

Given a circuit-SAT instance

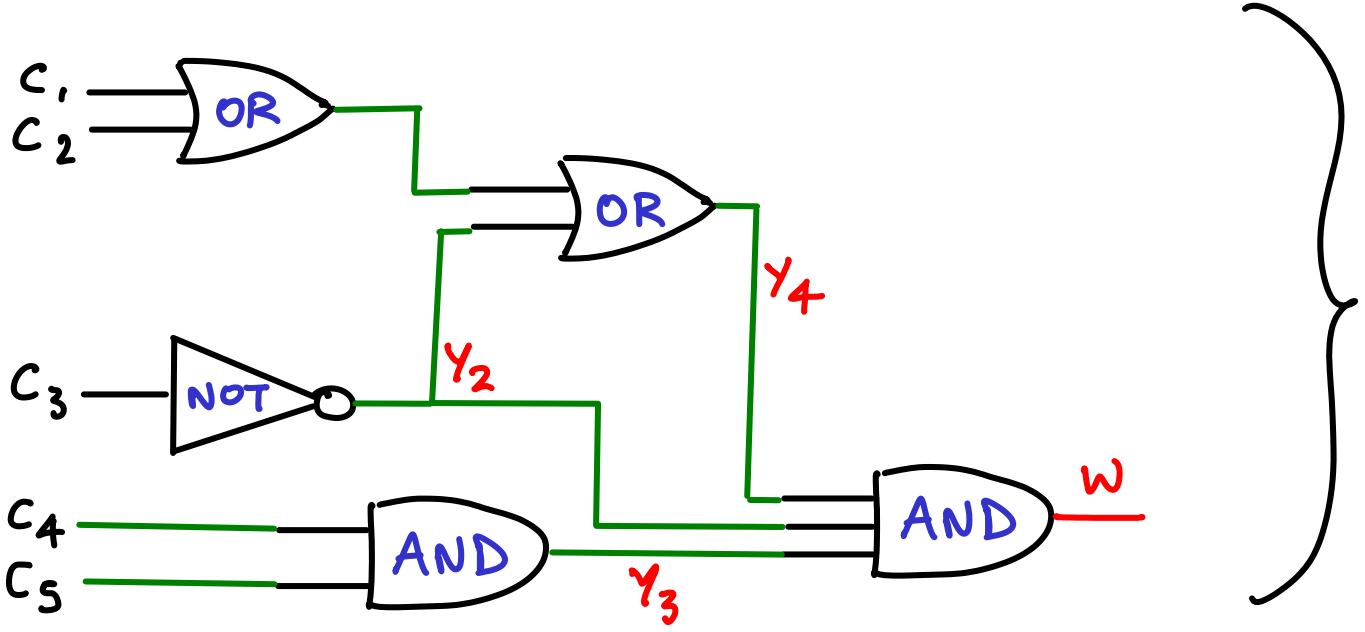


boolean SAT

circuit SAT

Boolean SAT $(x_1 \vee x_2 \vee \bar{x}_3) \wedge ((x_1 \leftrightarrow x_5) \vee (x_4 \rightarrow \bar{x}_3)) \rightarrow ? \rightarrow 1$

Given a circuit-SAT instance transform (quickly) into a Boolean SAT.



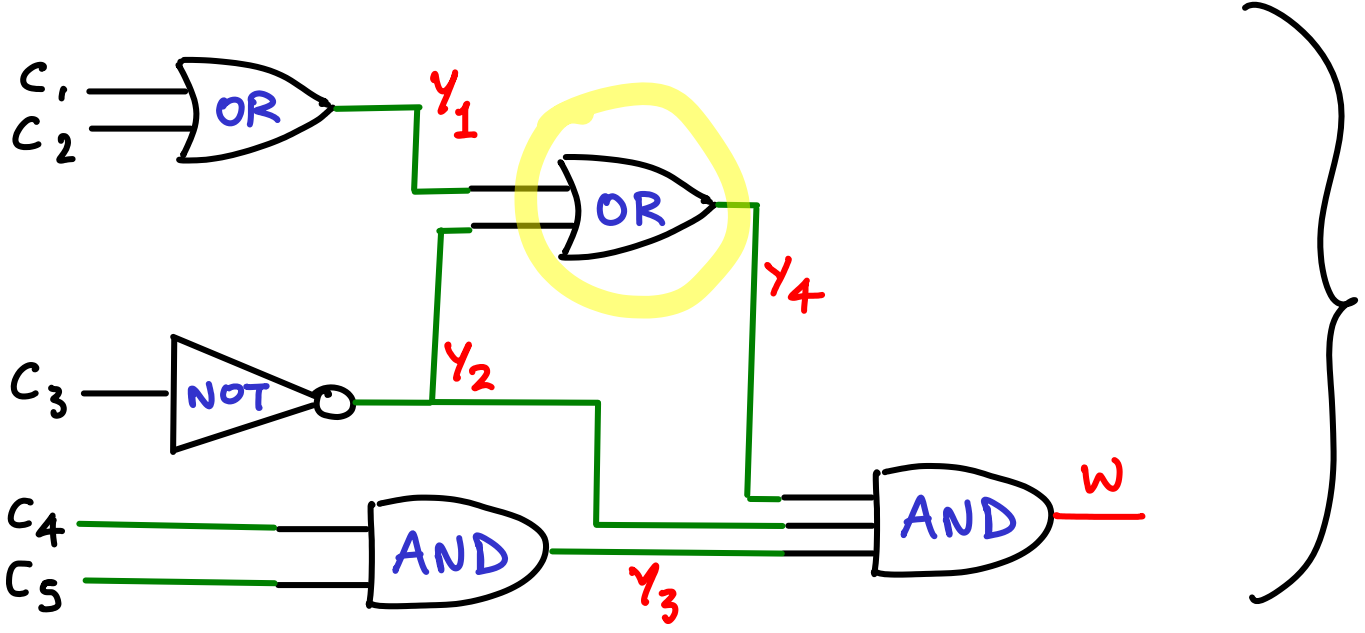
$w \wedge (w \leftrightarrow (y_4 \wedge y_2 \wedge y_3))$

boolean SAT

circuit SAT

Boolean SAT $(x_1 \vee x_2 \vee \bar{x}_3) \wedge ((x_1 \leftrightarrow x_5) \vee (x_4 \rightarrow \bar{x}_3)) \rightarrow ? \rightarrow 1$

Given a circuit-SAT instance transform (quickly) into a Boolean SAT.



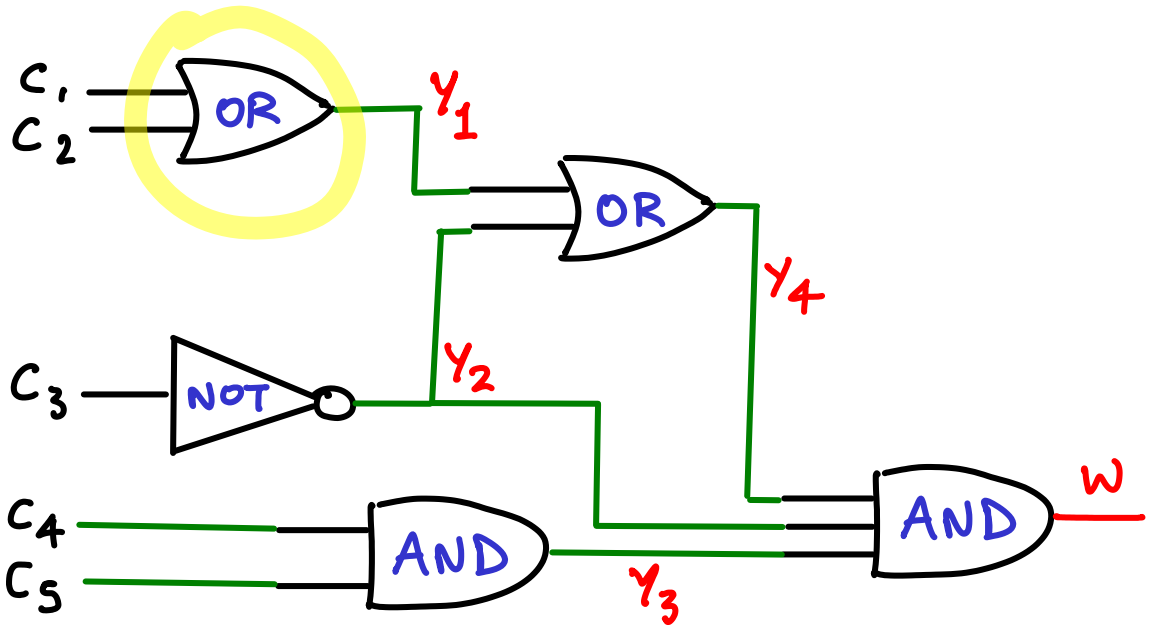
$$w \wedge (w \leftrightarrow (y_4 \wedge y_2 \wedge y_3)) \wedge (y_4 \leftrightarrow (y_1 \vee y_2))$$

boolean SAT

circuit SAT

Boolean SAT $(x_1 \vee x_2 \vee \bar{x}_3) \wedge ((x_1 \leftrightarrow x_5) \vee (x_4 \rightarrow \bar{x}_3)) \rightarrow ? \rightarrow 1$

Given a circuit-SAT instance transform (quickly) into a Boolean SAT.



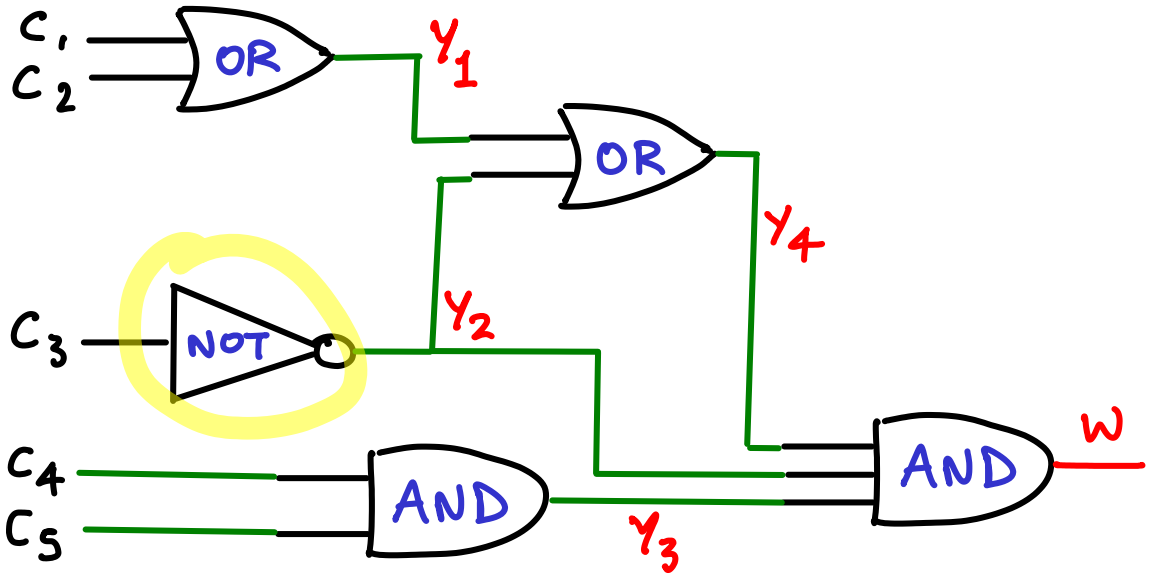
$$w \wedge (w \leftrightarrow (y_4 \wedge y_2 \wedge y_3)) \wedge (y_4 \leftrightarrow (y_1 \vee y_2)) \wedge (y_1 \leftrightarrow (c_1 \vee c_2))$$

boolean SAT

circuit SAT

Boolean SAT $(x_1 \vee x_2 \vee \bar{x}_3) \wedge ((x_1 \leftrightarrow x_5) \vee (x_4 \rightarrow \bar{x}_3)) \rightarrow ? \rightarrow 1$

Given a circuit-SAT instance transform (quickly) into a Boolean SAT.



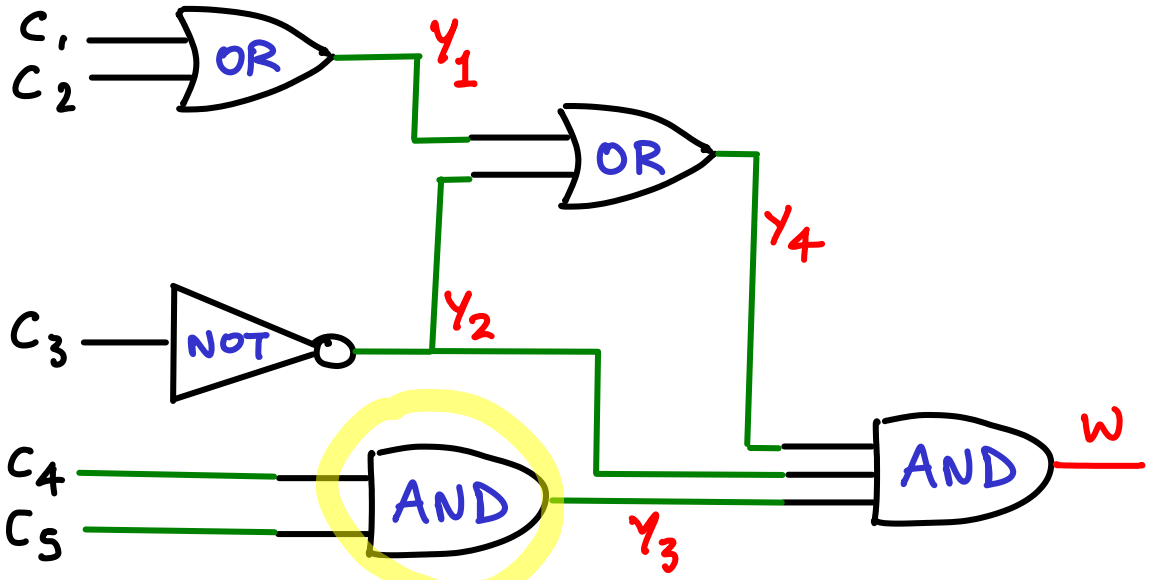
$$w \wedge (w \leftrightarrow (y_4 \wedge y_2 \wedge y_3))$$
$$\wedge (y_4 \leftrightarrow (y_1 \vee y_2))$$
$$\wedge (y_1 \leftrightarrow (c_1 \vee c_2))$$
$$\wedge (y_2 \leftrightarrow \bar{c}_3)$$

boolean SAT

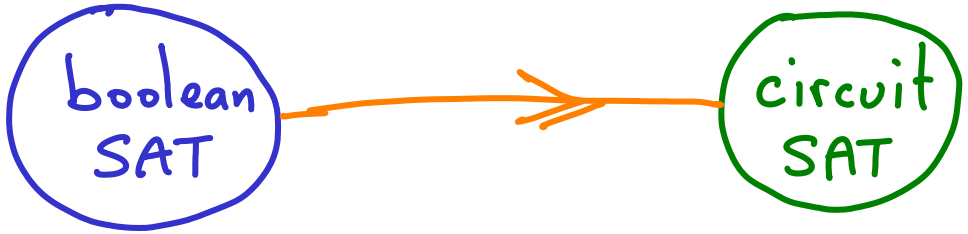
circuit SAT

Boolean SAT $(x_1 \vee x_2 \vee \bar{x}_3) \wedge ((x_1 \leftrightarrow x_5) \vee (x_4 \rightarrow \bar{x}_3)) \rightarrow ? \rightarrow 1$

Given a circuit-SAT instance transform (quickly) into a Boolean SAT.



$$w \wedge (w \leftrightarrow (y_4 \wedge y_2 \wedge y_3))$$
$$\wedge (y_4 \leftrightarrow (y_1 \vee y_2))$$
$$\wedge (y_1 \leftrightarrow (c_1 \vee c_2))$$
$$\wedge (y_2 \leftrightarrow \bar{c}_3)$$
$$\wedge (y_3 \leftrightarrow (c_4 \wedge c_5))$$

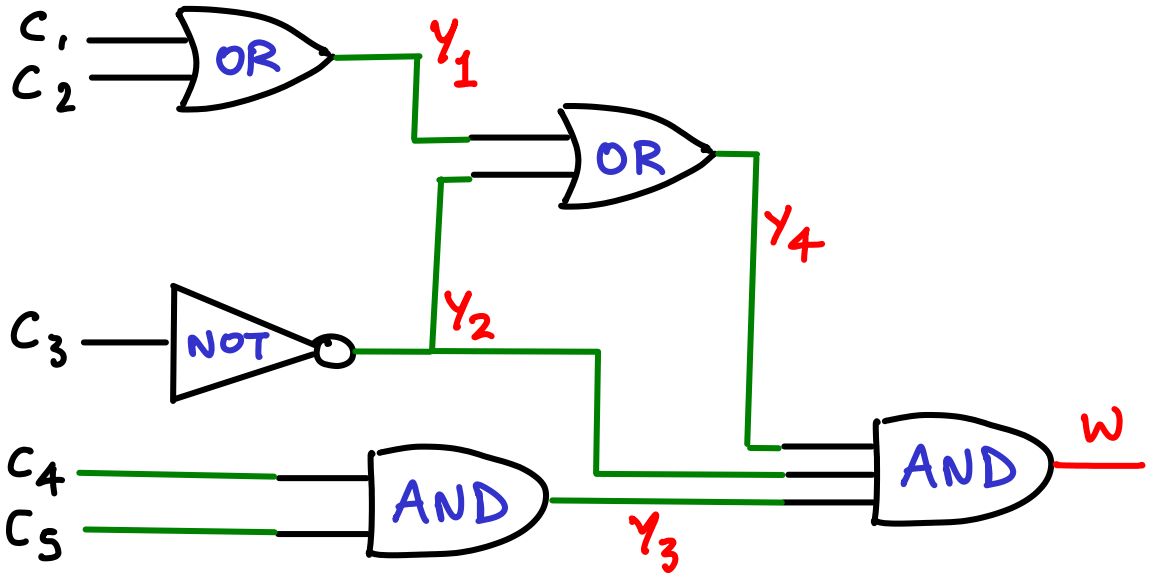


circuit-SAT $\leq_p$ Boolean SAT

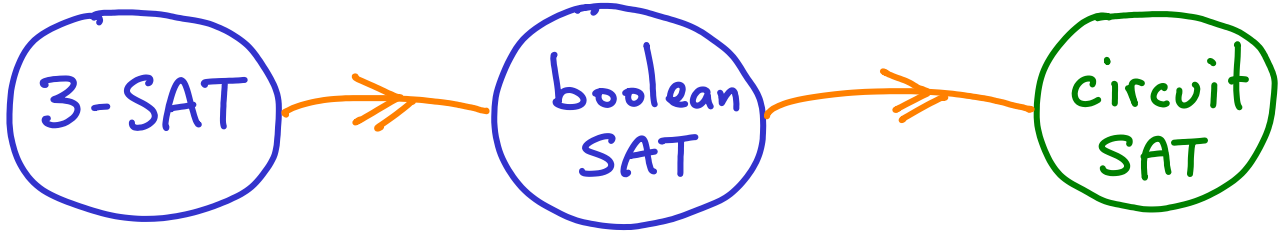
(actually also prototypical)

Boolean SAT $(x_1 \vee x_2 \vee \overline{x}_3) \wedge ((x_1 \leftrightarrow x_5) \vee (x_4 \rightarrow \overline{x}_3)) \rightarrow ? \rightarrow 1$

Given a circuit-SAT instance transform (quickly) into a Boolean SAT.



$$w \wedge (w \leftrightarrow (y_4 \wedge y_2 \wedge y_3))$$
$$\wedge (y_4 \leftrightarrow (y_1 \vee y_2))$$
$$\wedge (y_1 \leftrightarrow (c_1 \vee c_2))$$
$$\wedge (y_2 \leftrightarrow \overline{c}_3)$$
$$\wedge (y_3 \leftrightarrow (c_4 \wedge c_5))$$



3-SAT

$$(x_1 \vee x_2 \vee x_3)$$
$$\wedge (x_1 \vee x_4 \vee x_5)$$
$$\wedge (x_2 \vee \overline{x}_4 \vee x_{13})$$
$$\wedge (x_2 \vee \overline{x}_3 \vee x_3)$$
$$\vdots$$

k clauses
each w/ 3 literals

$(a \vee b \vee c)$

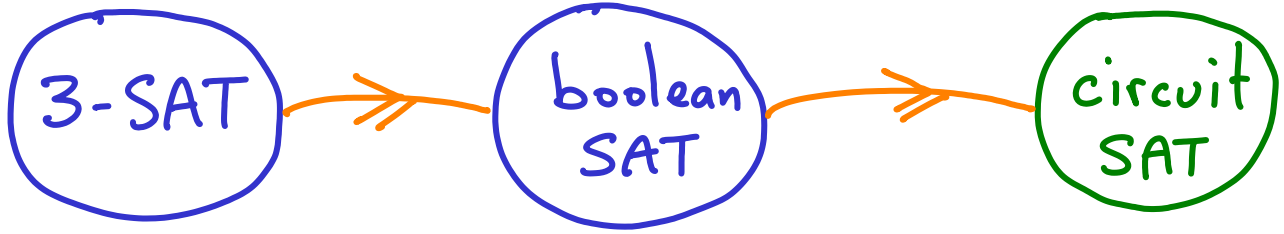
$\wedge (b \vee \bar{c} \vee \bar{d})$

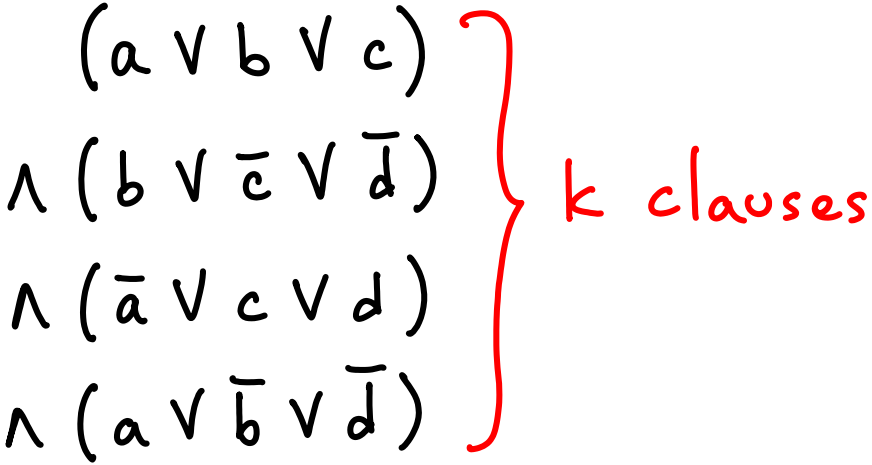
$\wedge (\bar{a} \vee c \vee d)$

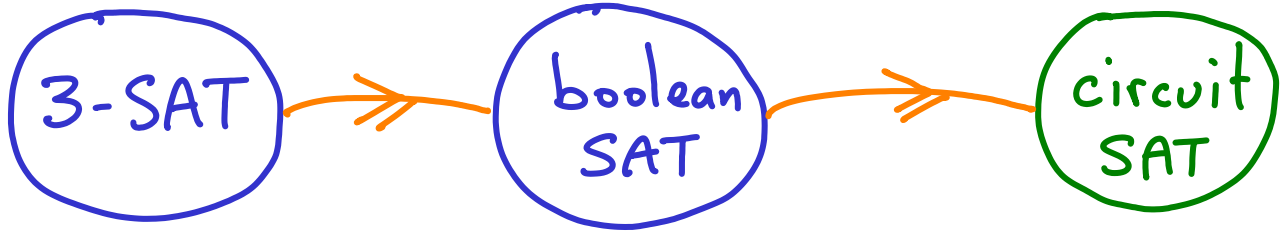
$\wedge (a \vee \bar{b} \vee \bar{d})$

} k clauses

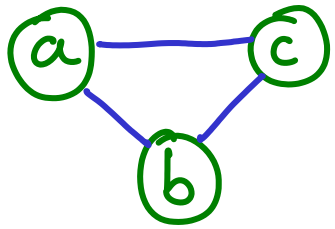
construct graph (quickly)

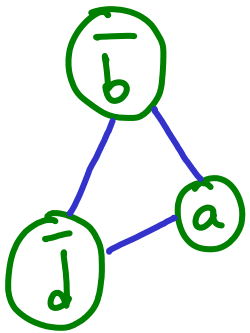


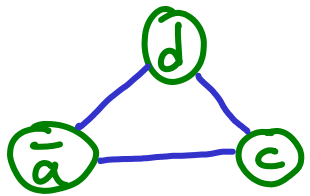


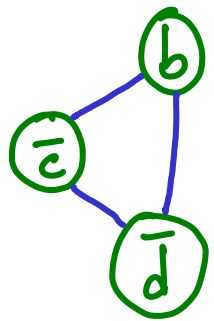


construct graph (quickly)









(a ∨ b ∨ c)

(a ∨ b̄ ∨ d̄)

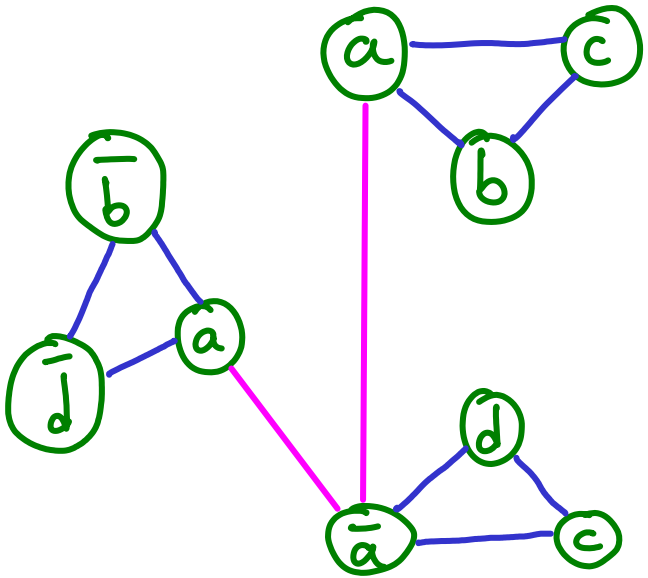
(b ∨ c̄ ∨ d̄)

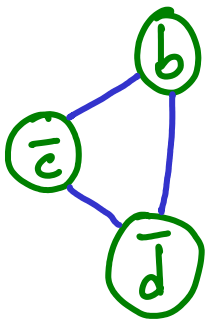
(ā ∨ c ∨ d)

$(a \vee b \vee c)$
$\wedge (b \vee \bar{c} \vee \bar{d})$
$\wedge (\bar{a} \vee c \vee d)$
$\wedge (a \vee \bar{b} \vee \bar{d})$

} k clauses

construct graph (quickly)



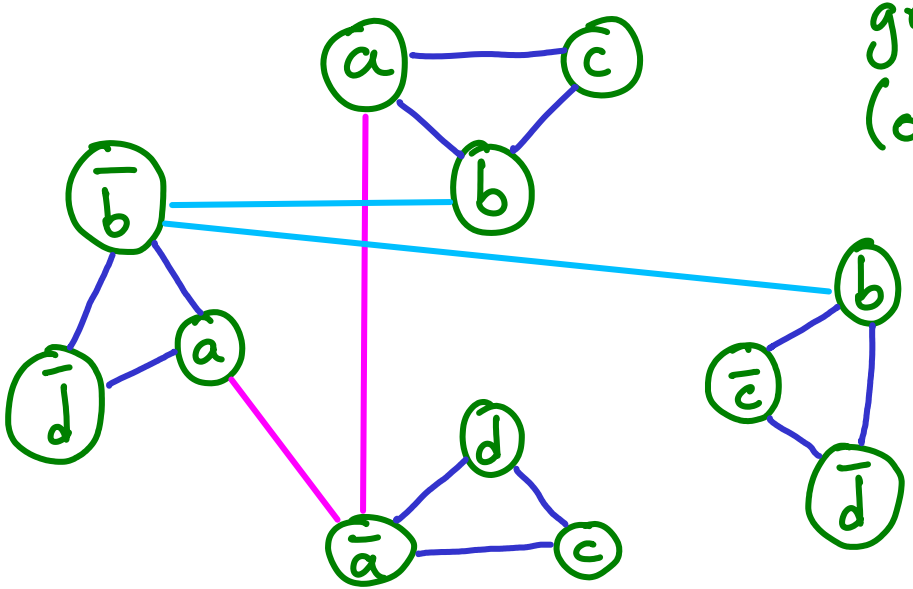


3-SAT → boolean SAT → circuit SAT

$(a \vee b \vee c)$
$\wedge (b \vee \bar{c} \vee \bar{d})$
$\wedge (\bar{a} \vee c \vee d)$
$\wedge (a \vee \bar{b} \vee \bar{d})$

k clauses

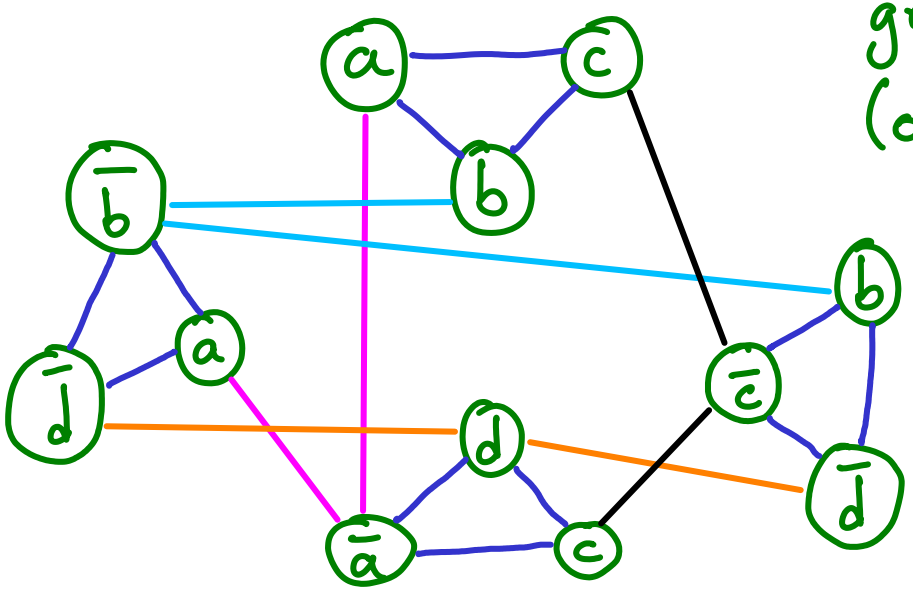
construct graph (quickly)



$(a \vee b \vee c)$
$\wedge (b \vee \bar{c} \vee \bar{d})$
$\wedge (\bar{a} \vee c \vee d)$
$\wedge (a \vee \bar{b} \vee \bar{d})$

k clauses

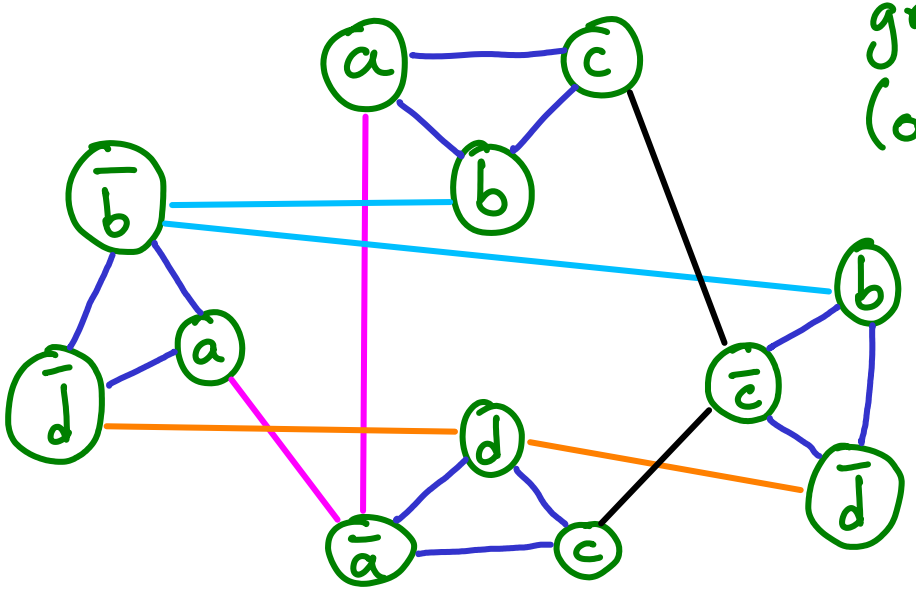
construct graph (quickly)



$(a \vee b \vee c)$
$\wedge (b \vee \bar{c} \vee \bar{d})$
$\wedge (\bar{a} \vee c \vee d)$
$\wedge (a \vee \bar{b} \vee \bar{d})$

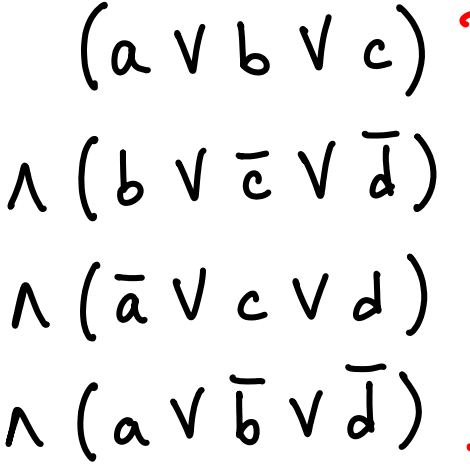
} k clauses

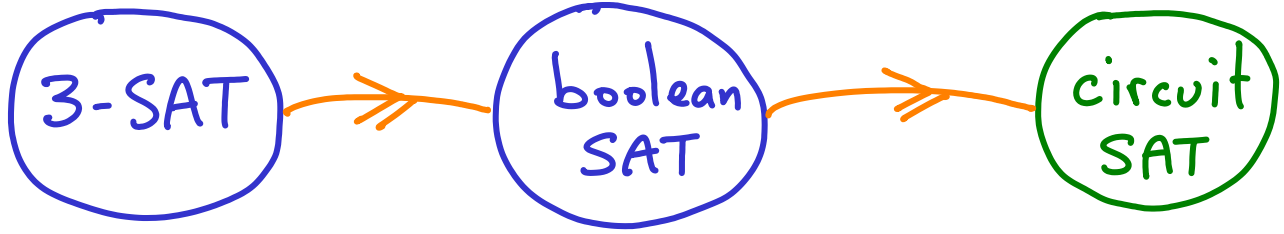
construct graph (quickly)



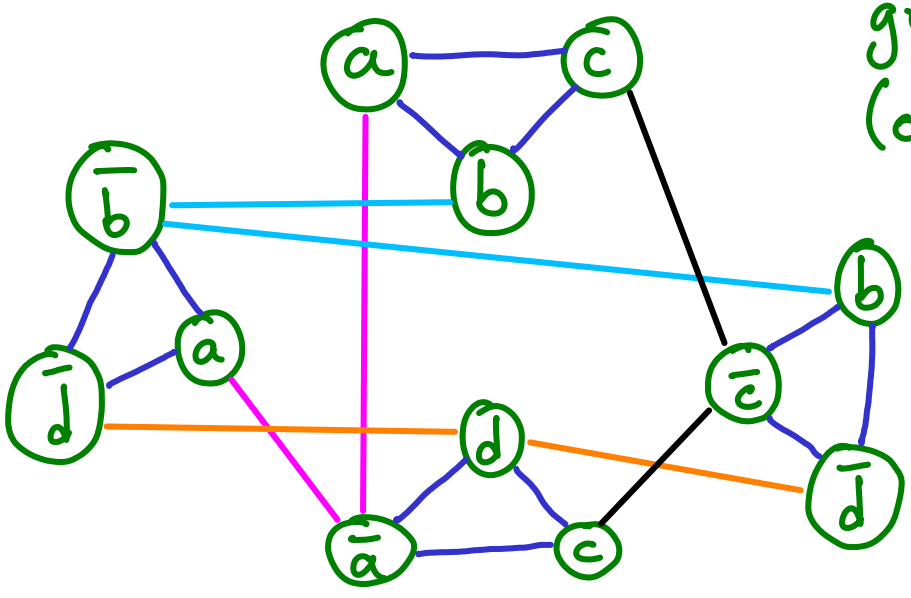
3-SAT → boolean SAT → circuit SAT

Ask: does this graph have k independent vertices?





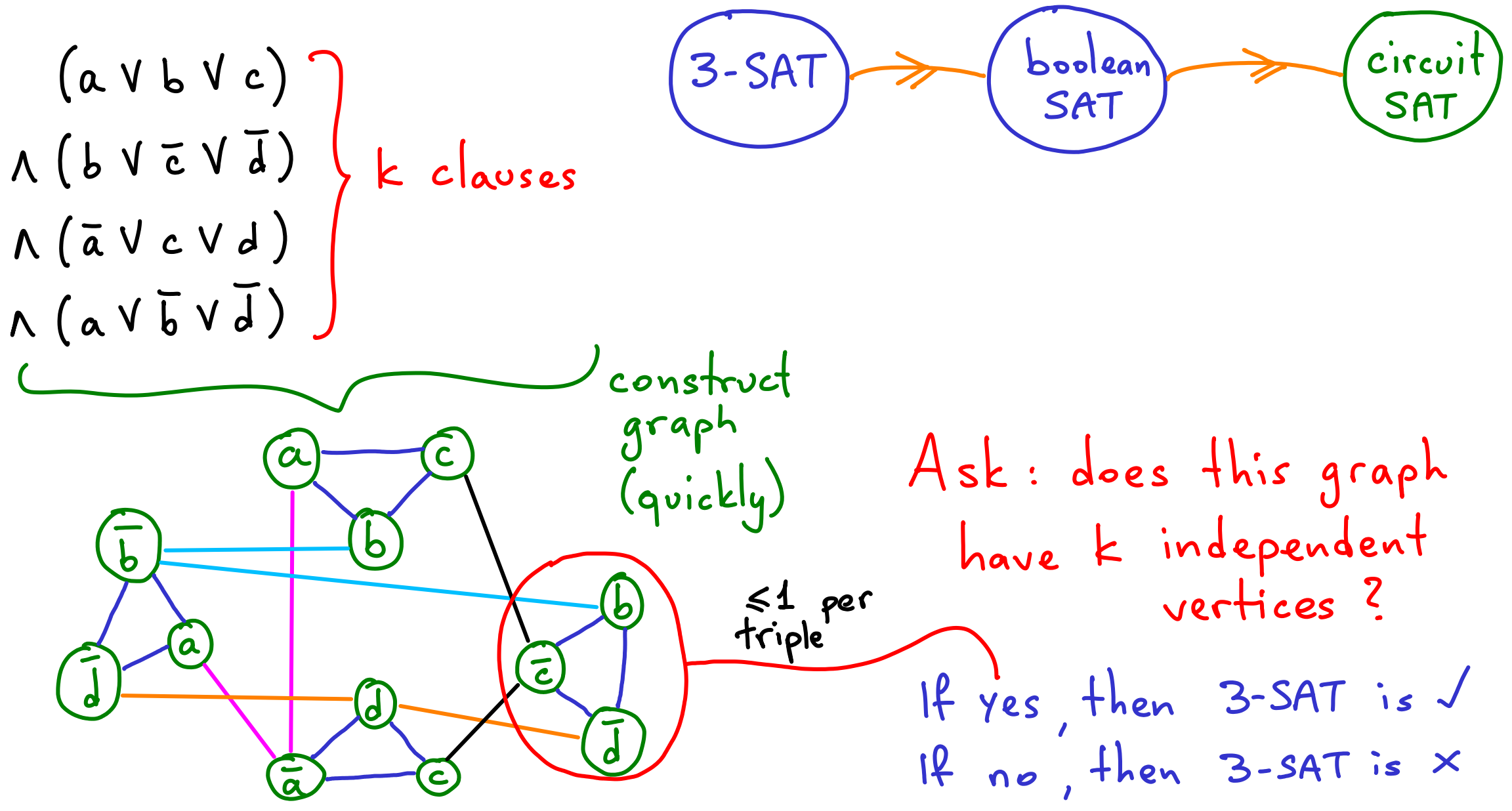
construct graph (quickly)



Ask: does this graph have k independent vertices?

Would need 1 vertex per triple
(can't have 2 in any triple)

Answer maps back to 3-SAT


3-SAT
boolean SAT
circuit SAT
(a ∨ b ∨ c)
∧ (b ∨ c̄ ∨ d̄)
∧ (ā ∨ c ∨ d)
∧ (a ∨ b̄ ∨ d̄)
k clauses
construct graph (quickly)
≤1 per triple
Ask: does this graph have k independent vertices?
If yes, then 3-SAT is ✓
If no, then 3-SAT is ×

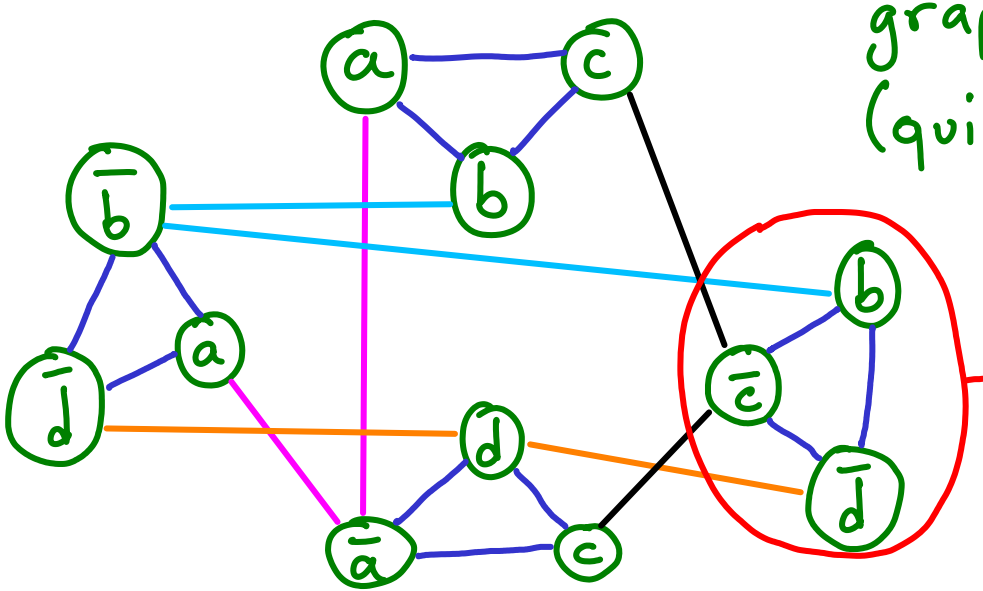
$(a \vee b \vee c)$
$\wedge (b \vee \bar{c} \vee \bar{d})$
$\wedge (\bar{a} \vee c \vee d)$
$\wedge (a \vee \bar{b} \vee \bar{d})$

} k clauses

3-SAT → boolean SAT → circuit SAT

3-SAT → independent set

construct graph (quickly)



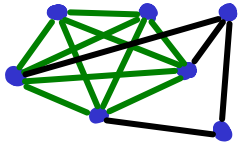
≤1 per triple

Ask: does this graph have k independent vertices?

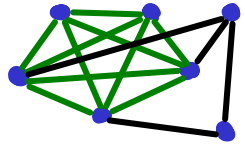
If yes, then 3-SAT is ✓

If no, then 3-SAT is ×

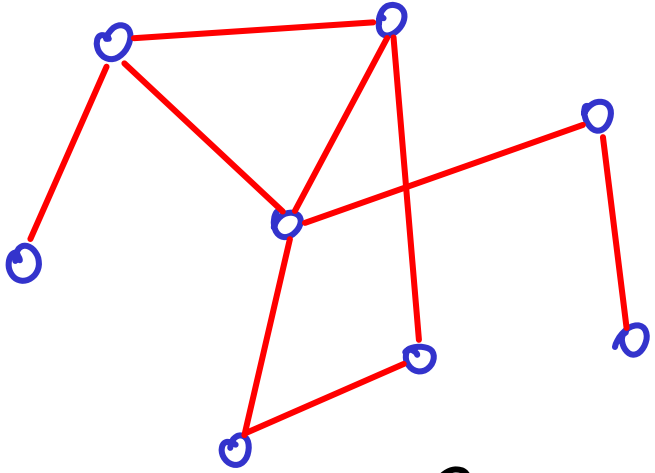
CLIQUE in a graph: subset of V s.t. all pairs of vertices share edges.



CLIQUE in a graph: subset of V s.t. all pairs of vertices share edges.

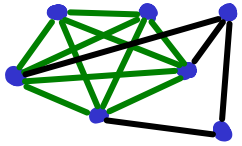


size k independent set in G?

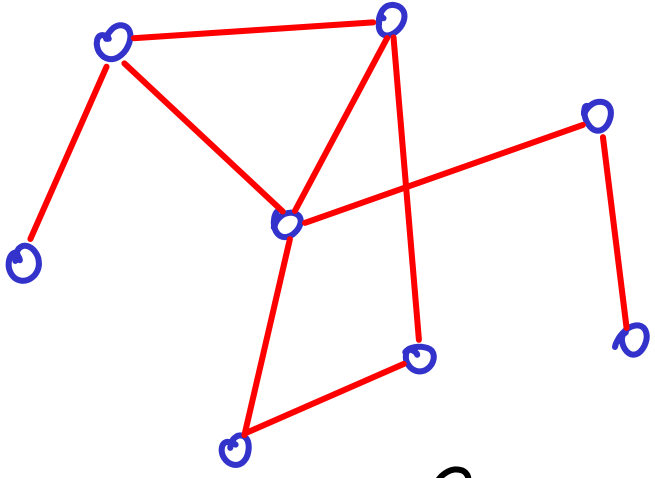


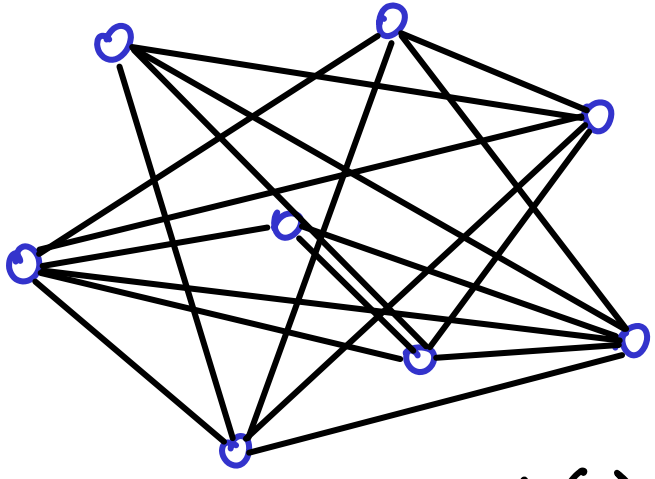
G

CLIQUE in a graph : subset of V s.t. all pairs of vertices share edges.



size k independent set in G?



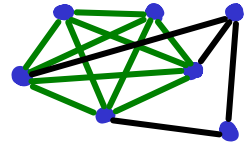


G

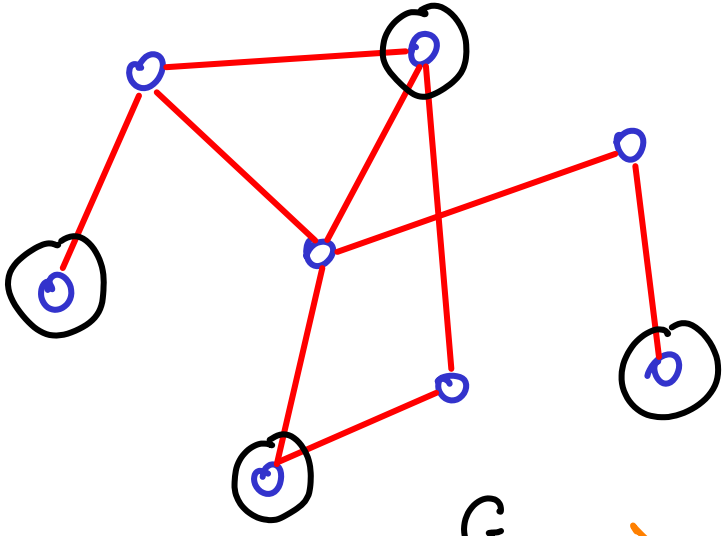
transform

complement (G) = G^c

CLIQUE in a graph : subset of V s.t. all pairs of vertices share edges.

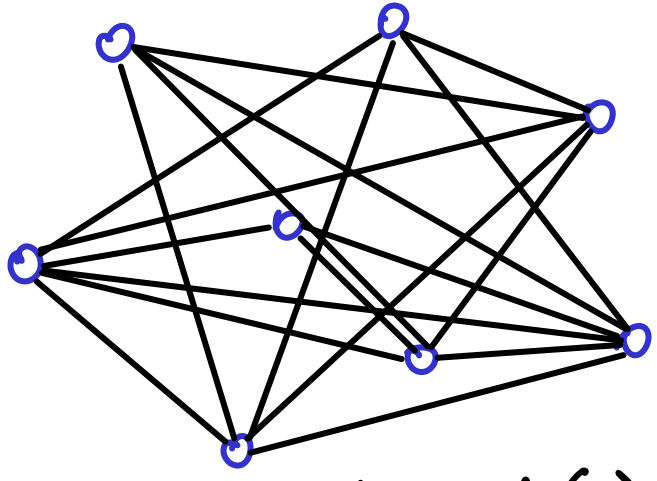


size k independent set in G? $\Longleftrightarrow$ size k clique in G^c?

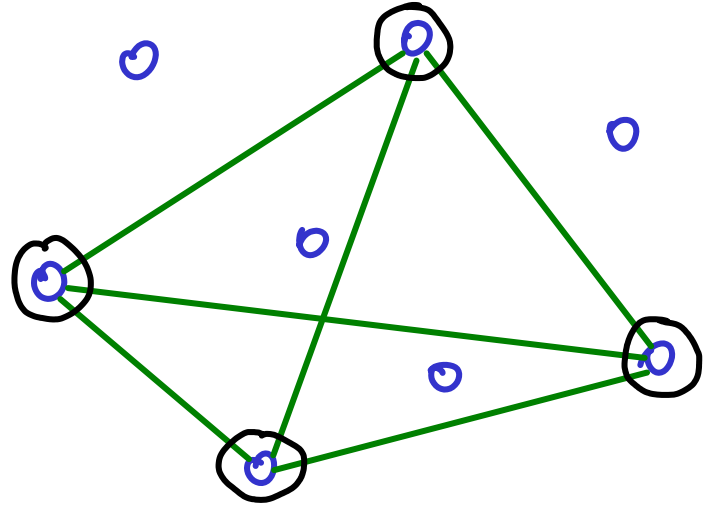


G

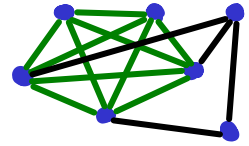
transform



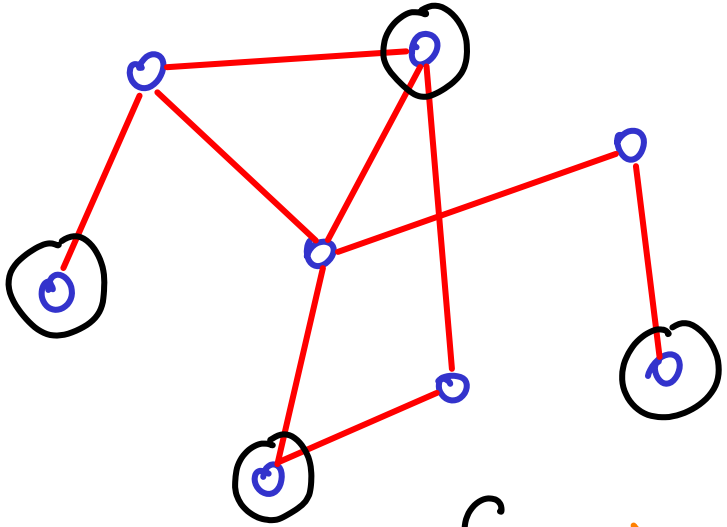
complement (G) = G^c



CLIQUE in a graph : subset of V s.t. all pairs of vertices share edges.

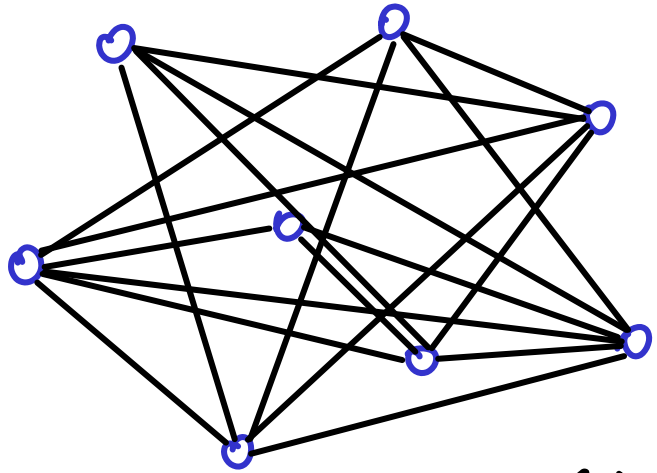


size k independent set in G? $\iff$ size k clique in G^c?

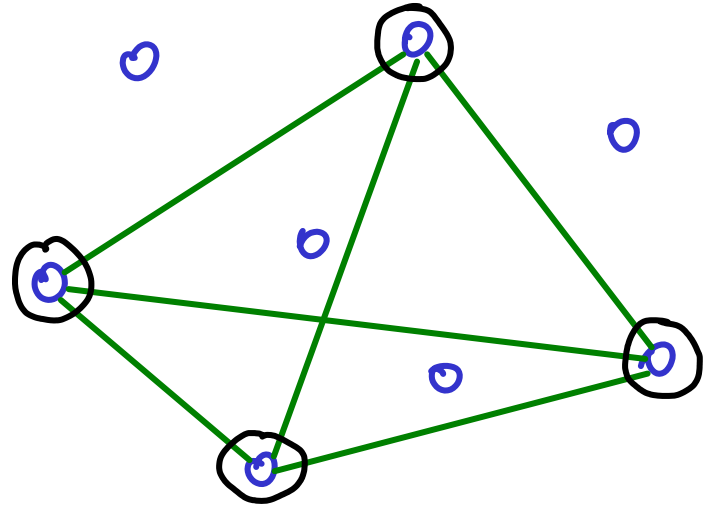


G

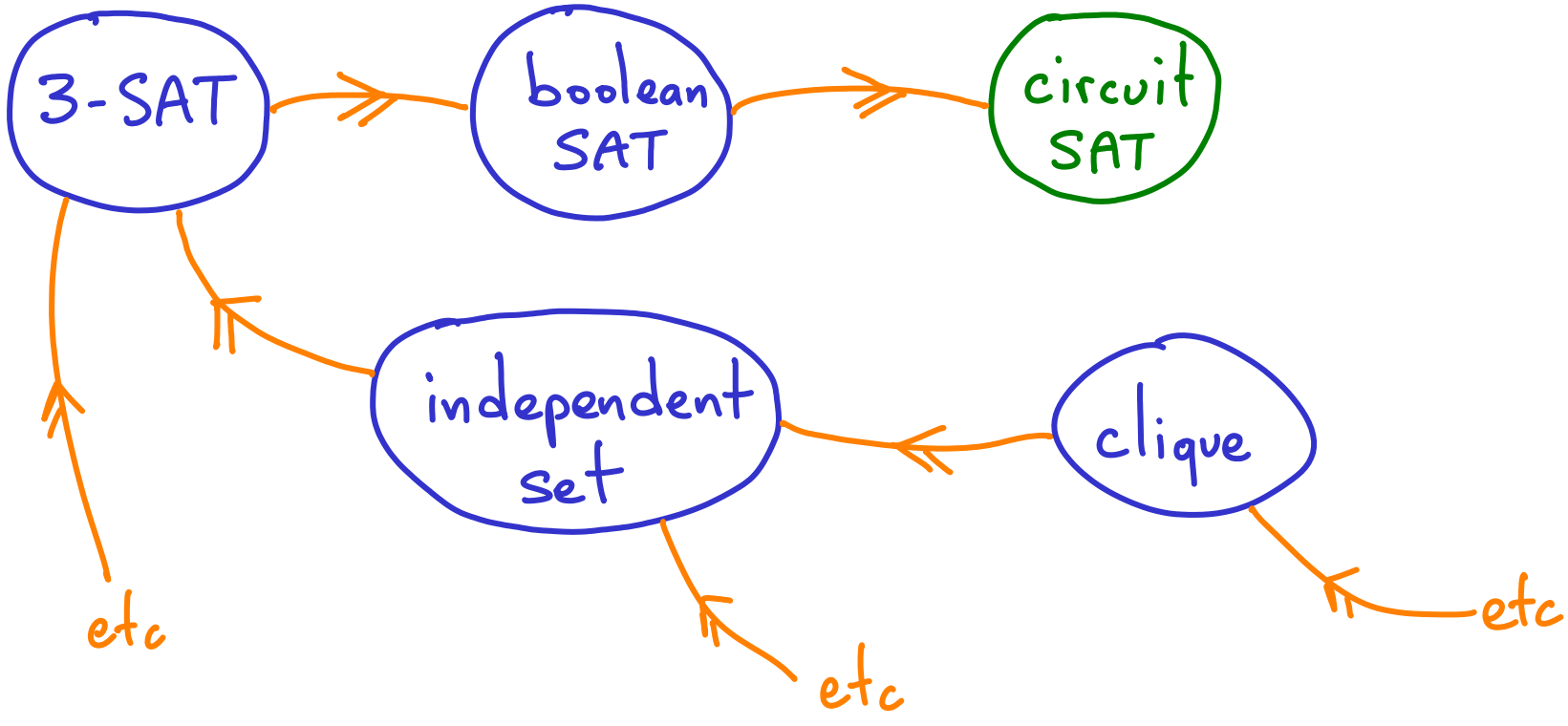
transform



complement (G) = G^c



Asking if a graph has a clique of size k : NPC


3-SAT
boolean SAT
circuit SAT
independent set
clique
etc
etc
etc

OTHER HARD PROBLEMS

- knapsack: given item types, w/ size & value, fill a bag w/ max value (multiples ok)
- subset sum: does a subset of given integers sum to t?
- partition: split set of integers in 2 groups with equal sums
- planar SAT: SAT without wire crossings
- set cover: given some sets, select min# sets s.t all elements are present
- hitting set: given sets, select min# elements s.t. all sets are represented
- longest path: visiting each vertex once.
- Steiner tree: ~MST on selected subset of a graph
- traveling salesman: min-cost simple cycle on weighted complete graph

OTHER HARD PROBLEMS

... & some are even harder

Tetris

Minesweeper

Lemmings

Mario Bros

Pac-man

Prince of Persia

Portal

Doom

etc