

CPSC 366. Lecture 19. April 5, 2022.

Lecture 4 on NP:

Nondeterministic Polynomial Time

# Polynomial Time (Cook-Levin) Reductions:

Problem  $X$  is poly time reducible to  $Y$

$$X \leq_p Y$$

if can solve  $X$  using poly computation and a poly number of calls to an algorithm solving  $Y$ .

“Up to poly factors,  $X$  is at least as easy as  $Y$ ”

“Up to poly factors,  $Y$  is harder than  $X$ ”

$Y$  is NP-hard if for all  $X$  in NP,  $X \leq_p Y$

# Karp Reductions (a.k.a. many-to-one reductions)

Problem  $X$  is poly time Karp reducible to  $Y$  if there is a polynomial time algorithm  $A$  such that

$X(s) = \text{yes}$  if and only if  $Y(A(s)) = \text{yes}$

$A$  transforms instances of  $X$  into instances of  $Y$ .

All of our reductions have been (and will be) Karp reductions.

# NP-Hard problems

Circuit-SAT

(3,3)-SAT

Independent Set, Vertex Cover

3 Dimensional Matching, Exact Cover

Subset Sum

## Today

3-coloring

Hamiltonian Cycle

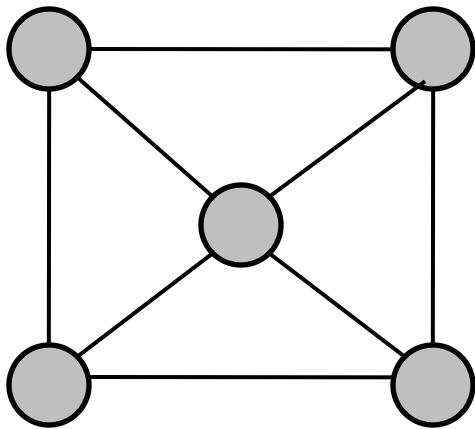
Travelling Salesperson Problem

# k-Coloring

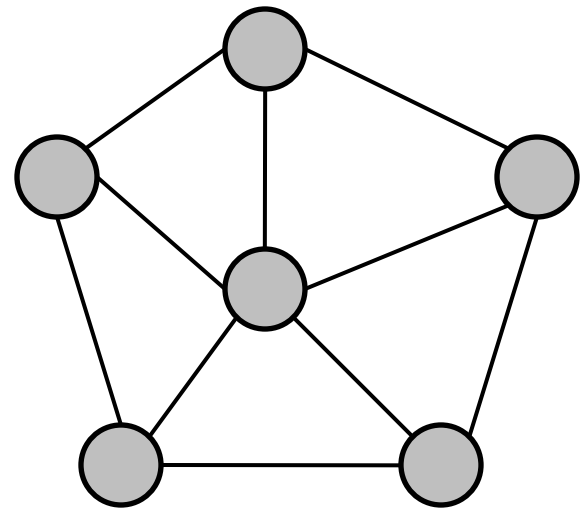
Given a graph  $G = (V, E)$ , does there exist

$f : V \rightarrow \{1, 2, \dots, k\}$  (colors)

So that for all  $(u, v) \in E$   $f(u) \neq f(v)$  ?



3-colorable



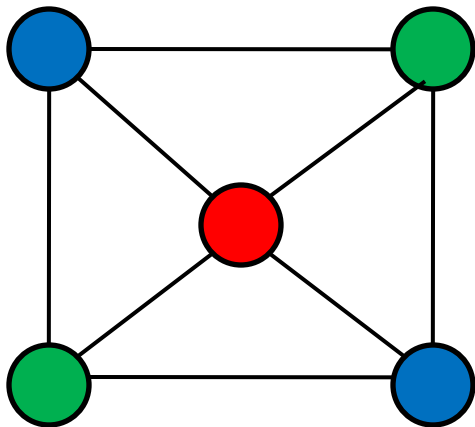
Not 3-colorable

# k-Coloring

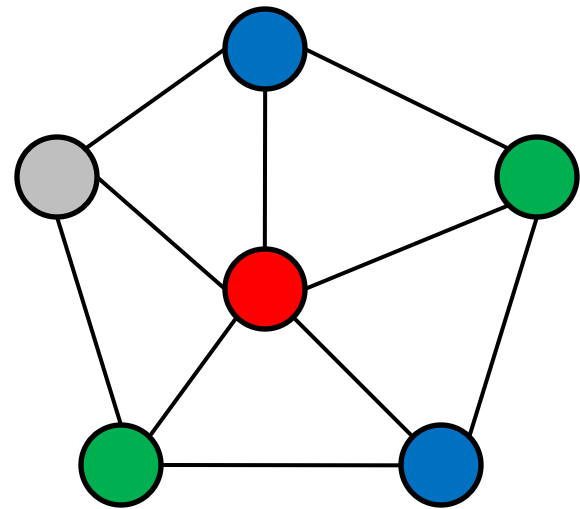
Given a graph  $G = (V, E)$ , does there exist

$f : V \rightarrow \{1, 2, \dots, k\}$  (colors)

So that for all  $(u, v) \in E$   $f(u) \neq f(v)$  ?



3-colorable



Not 3-colorable

# k-Coloring is NP-Complete

Clearly in NP, because can check a proposed coloring

To prove NP-hard, will show  $3\text{-SAT} \leq_p 3\text{-Coloring}$

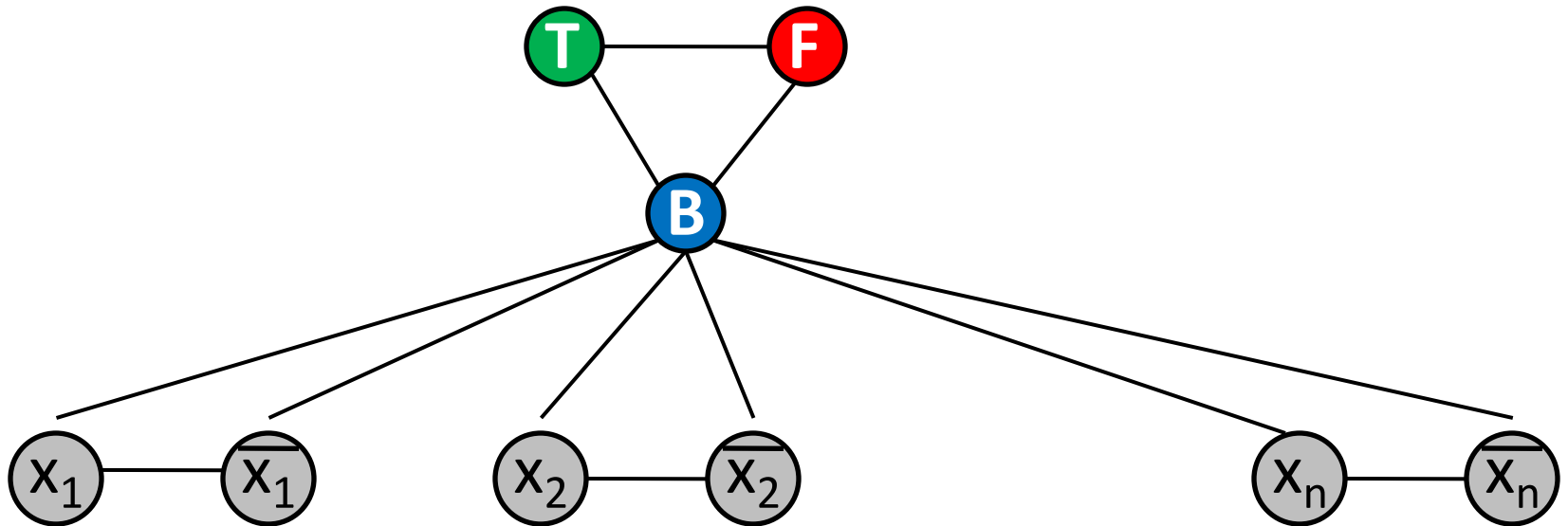
Given a collection of clauses  $C_1, \dots, C_k$ , each with at most 3 terms, on variables  $x_1, \dots, x_n$

produce graph  $G = (V, E)$  that is

3-colorable iff the clauses are satisfiable

# 3-Coloring is NP-Complete – variable gadgets

Create 3 special nodes: T, F, B (base),  
and one node for each term:  $x_i$  and  $\bar{x}_i$

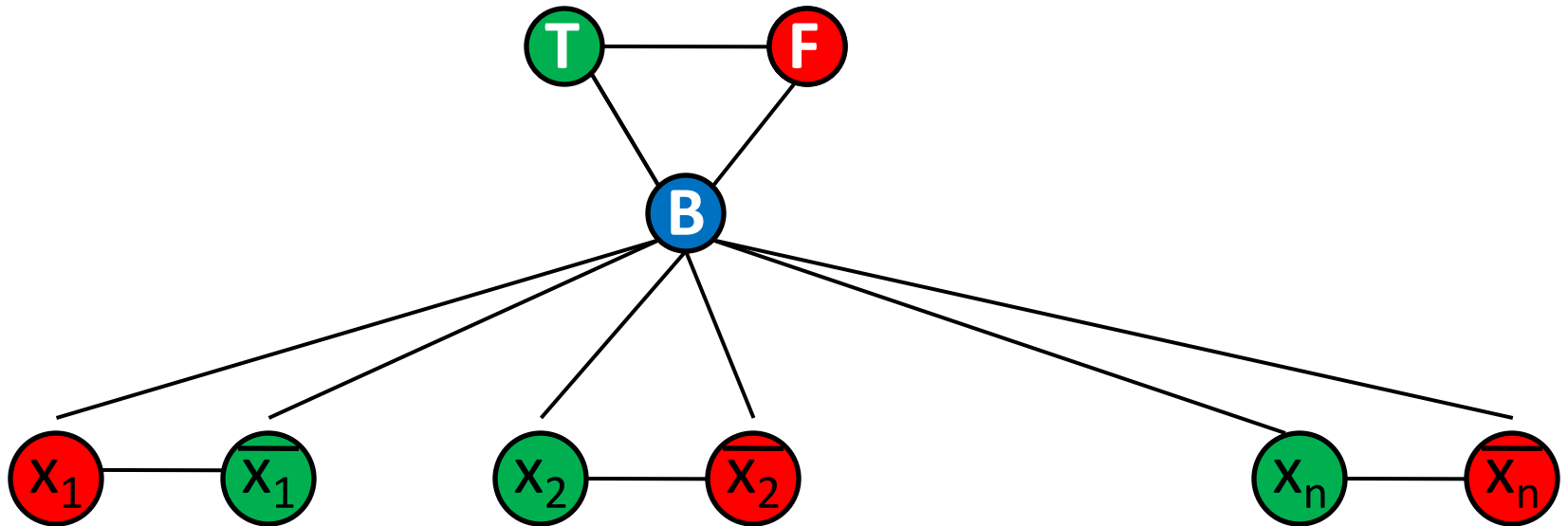


In every 3-coloring, one of  $x_i$  and  $\bar{x}_i$   
is colored T and one is colored F



# 3-Coloring is NP-Complete – variable gadgets

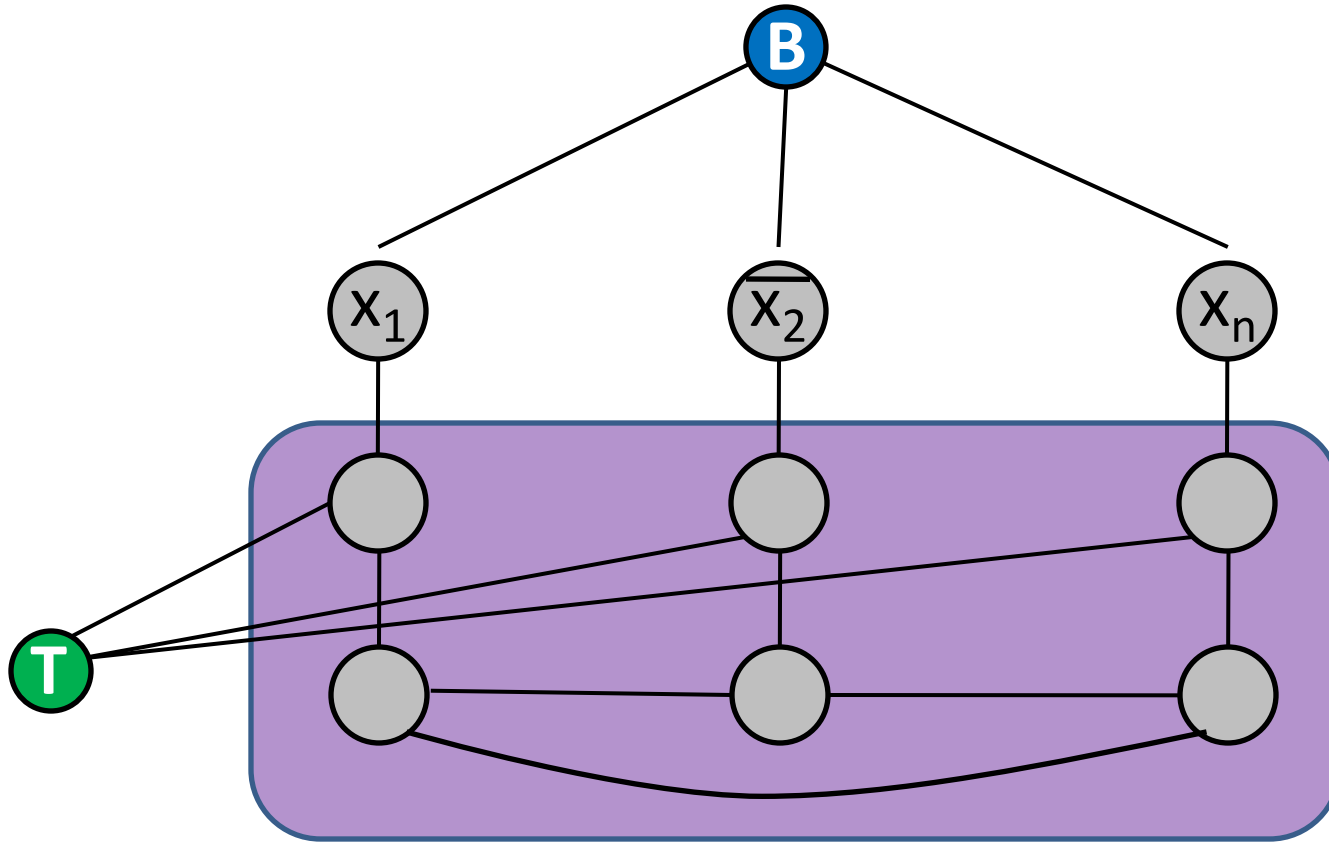
Create 3 special nodes: T, F, B (base),  
and one node for each term:  $x_i$  and  $\overline{x_i}$



In every 3-coloring, one of  $x_i$  and  $\overline{x_i}$   
is colored T and one is colored F

# 3-Coloring is NP-Complete – clause gadgets

Consider clause  $x_1 \vee \overline{x_2} \vee x_n$

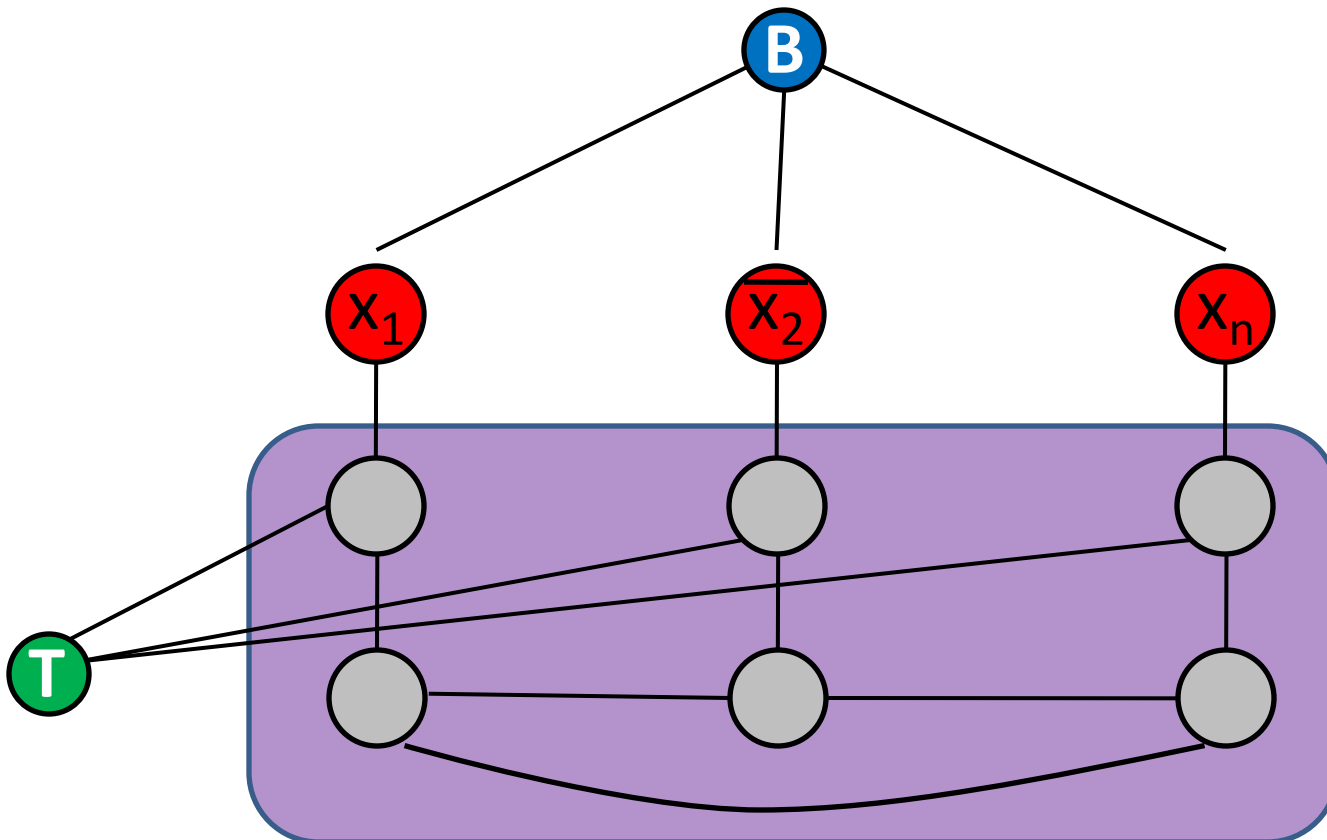


Claim: 3-colorable iff terms colored to satisfy clause

# 3-Coloring is NP-Complete – clause gadgets

Claim: 3-colorable iff terms colored to satisfy clause

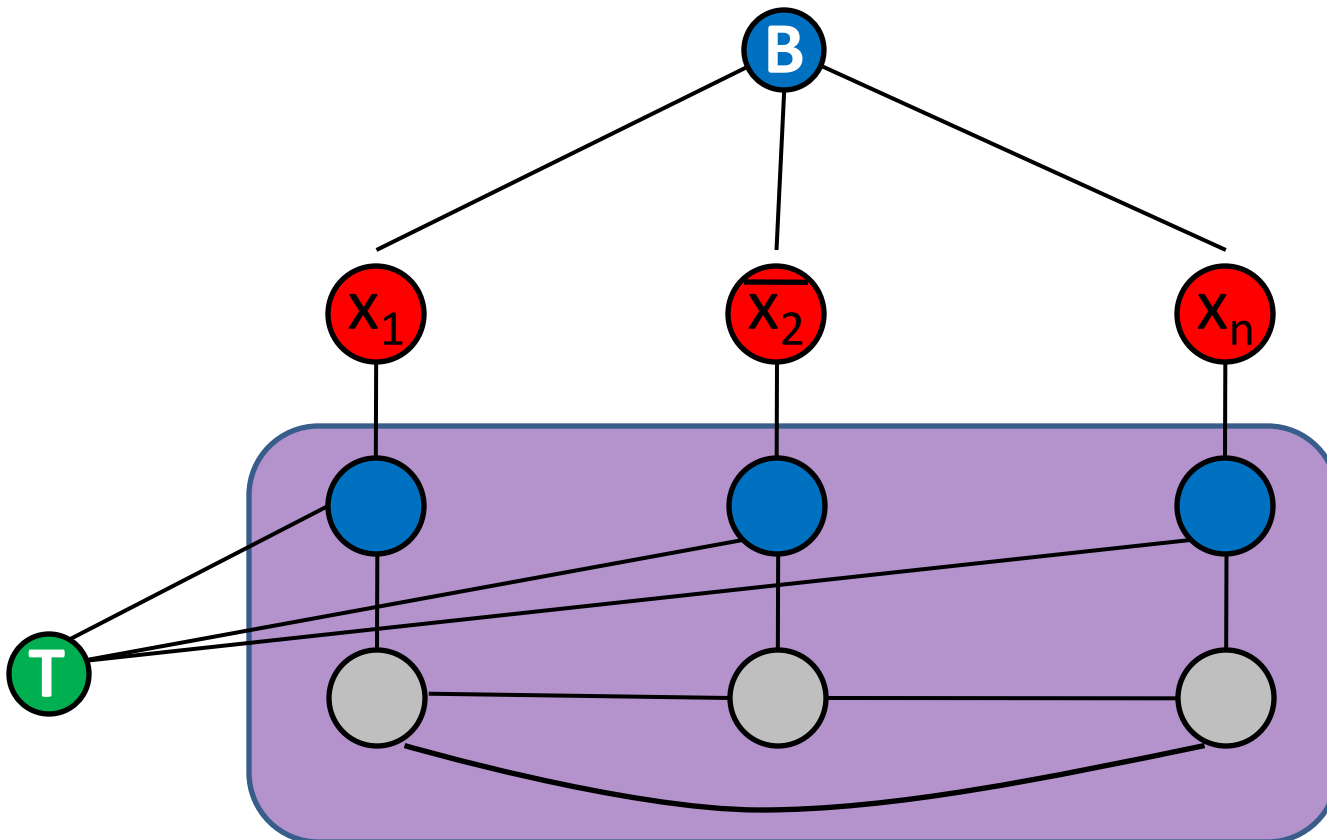
1. If terms all colored F, then cannot 3-color



# 3-Coloring is NP-Complete – clause gadgets

Claim: 3-colorable iff terms colored to satisfy clause

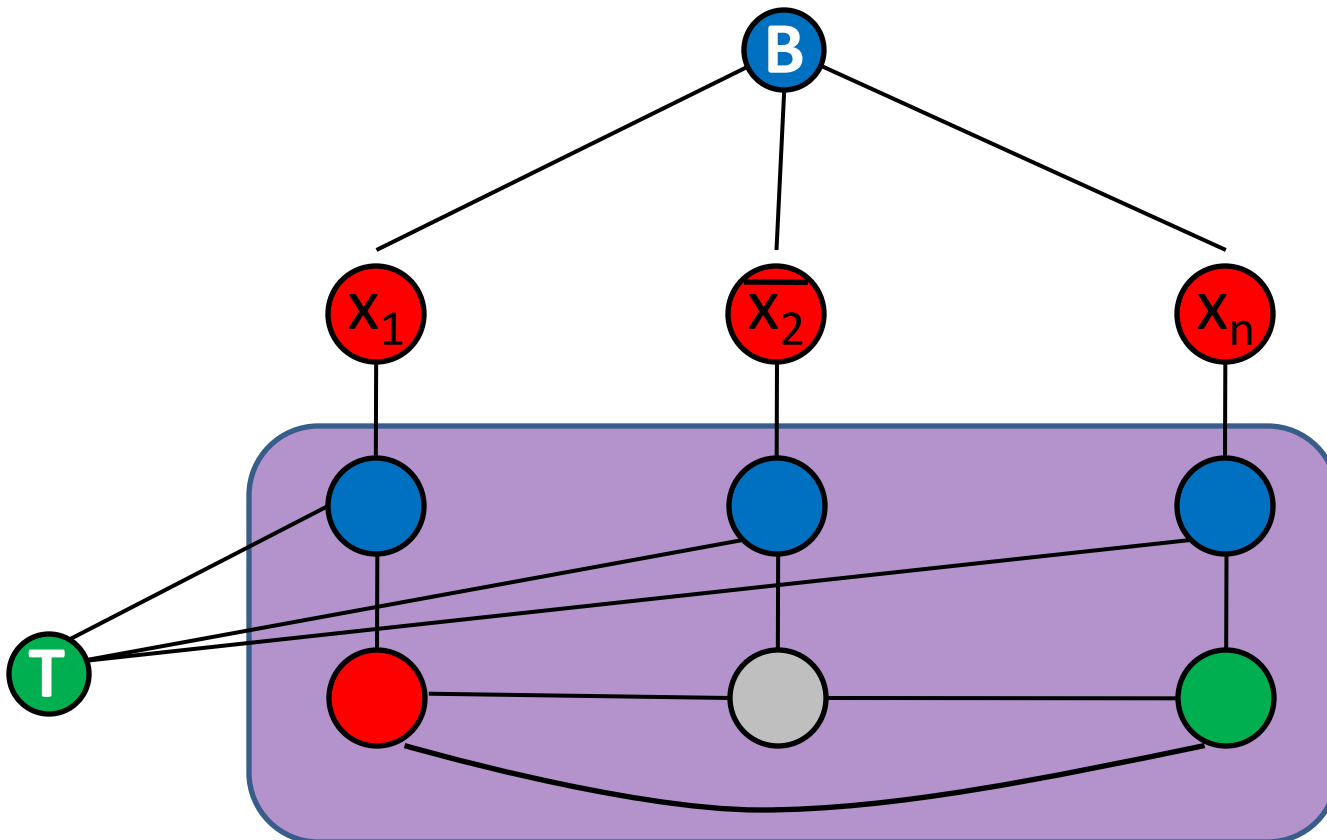
1. If terms all colored F, then cannot 3-color



# k-Coloring is NP-Complete – clause gadgets

Claim: 3-colorable iff terms colored to satisfy clause

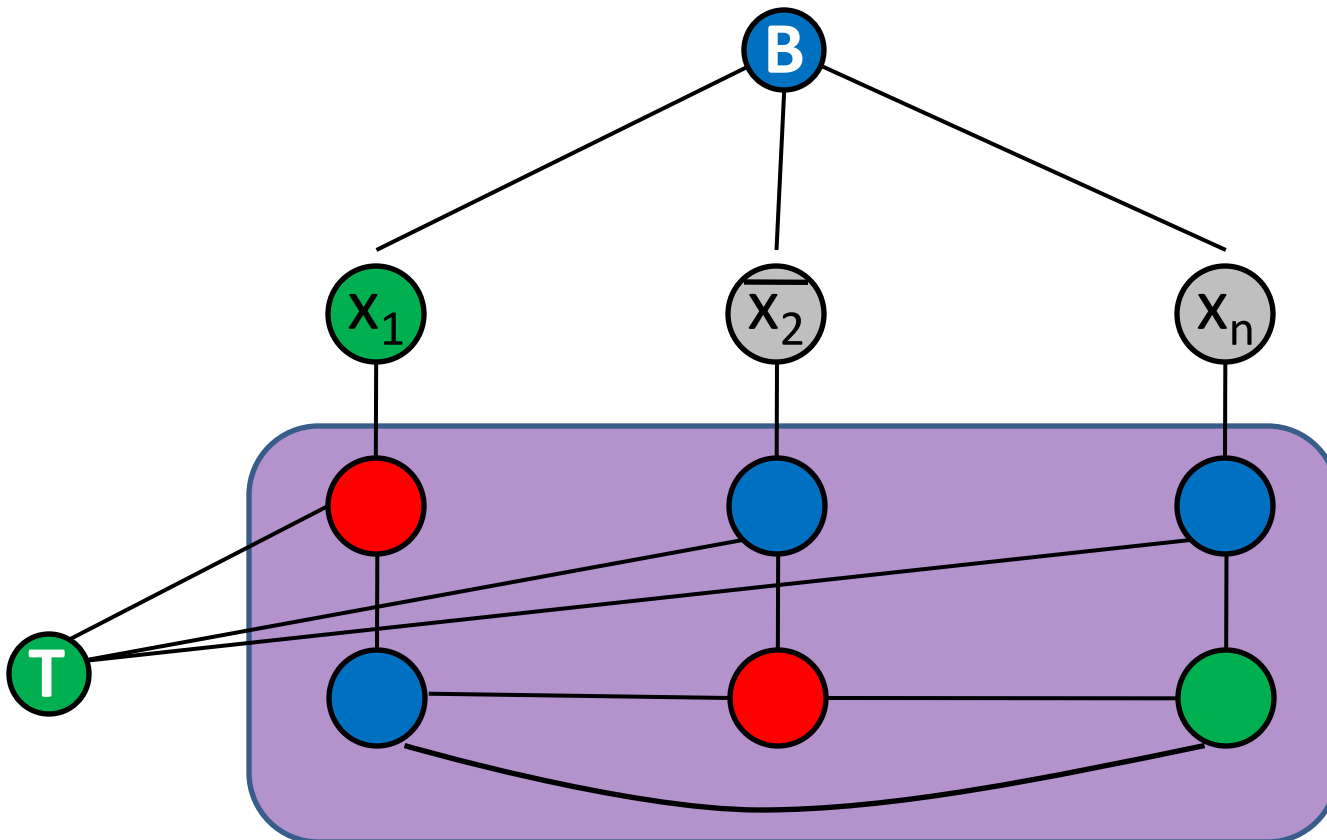
1. If terms all colored F, then cannot 3-color



# 3-Coloring is NP-Complete – clause gadgets

Claim: 3-colorable iff terms colored to satisfy clause

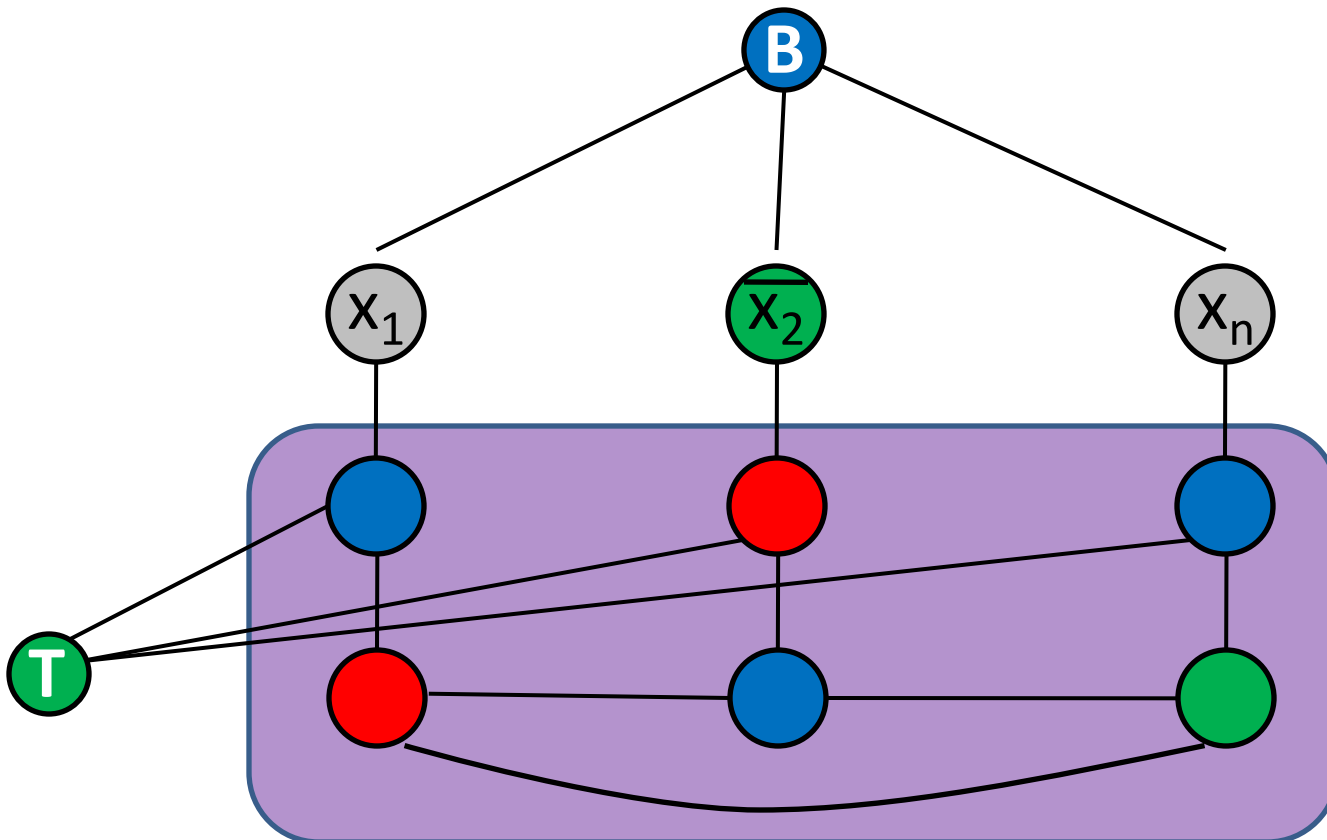
2. If some term true, can 3-color



# 3-Coloring is NP-Complete – clause gadgets

Claim: 3-colorable iff terms colored to satisfy clause

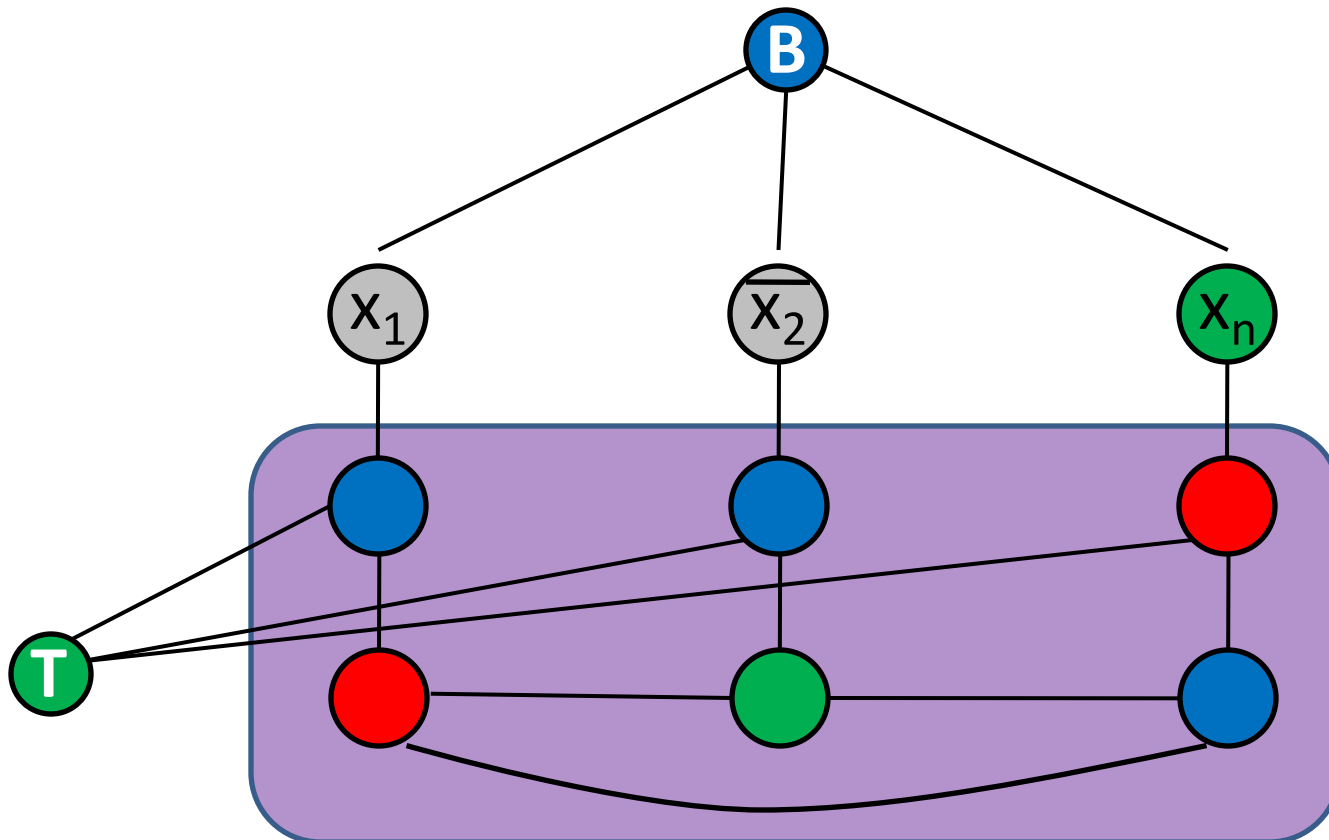
2. If some term true, can 3-color



# 3-Coloring is NP-Complete – clause gadgets

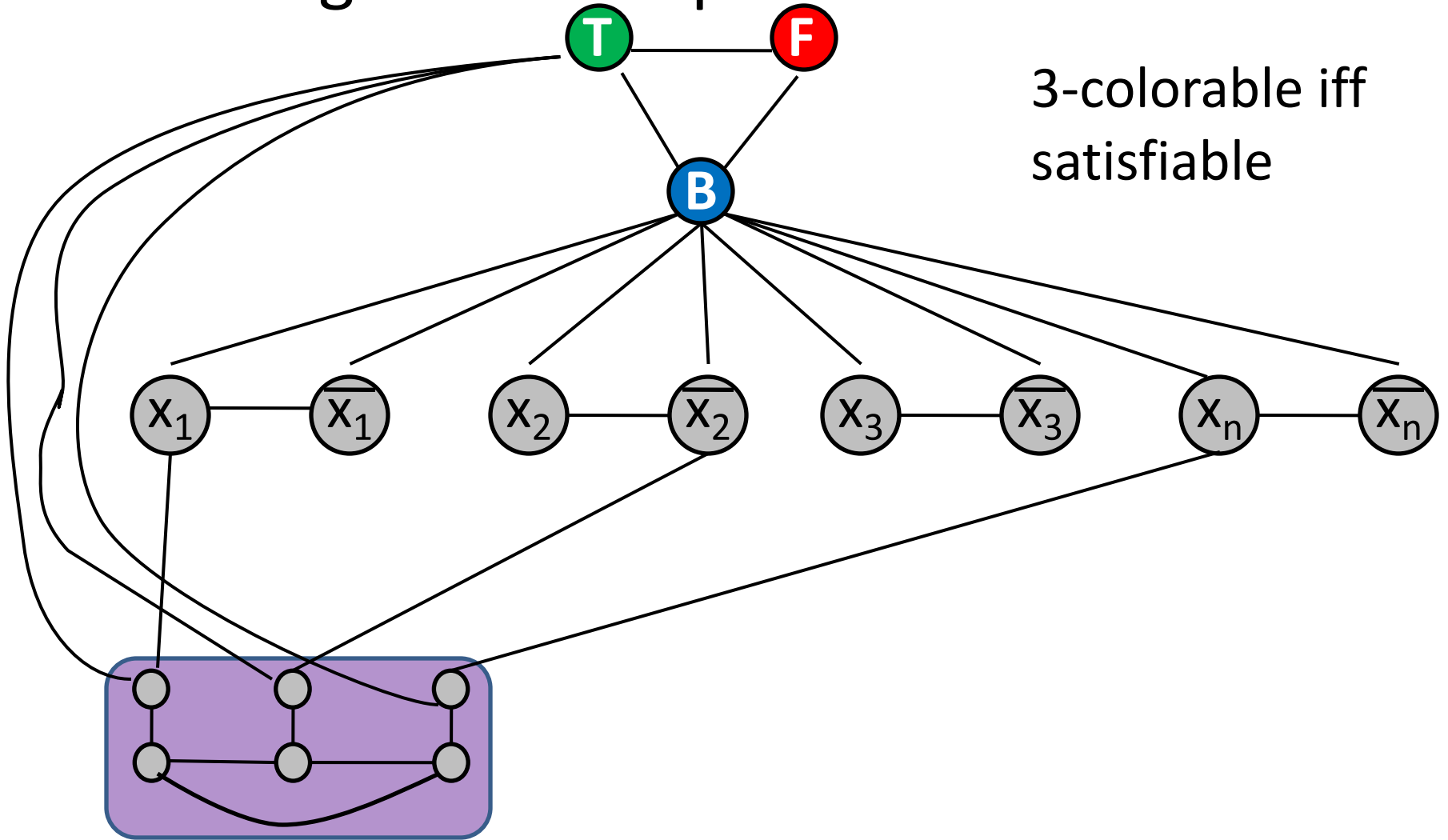
Claim: 3-colorable iff terms colored to satisfy clause

2. If some term true, can 3-color



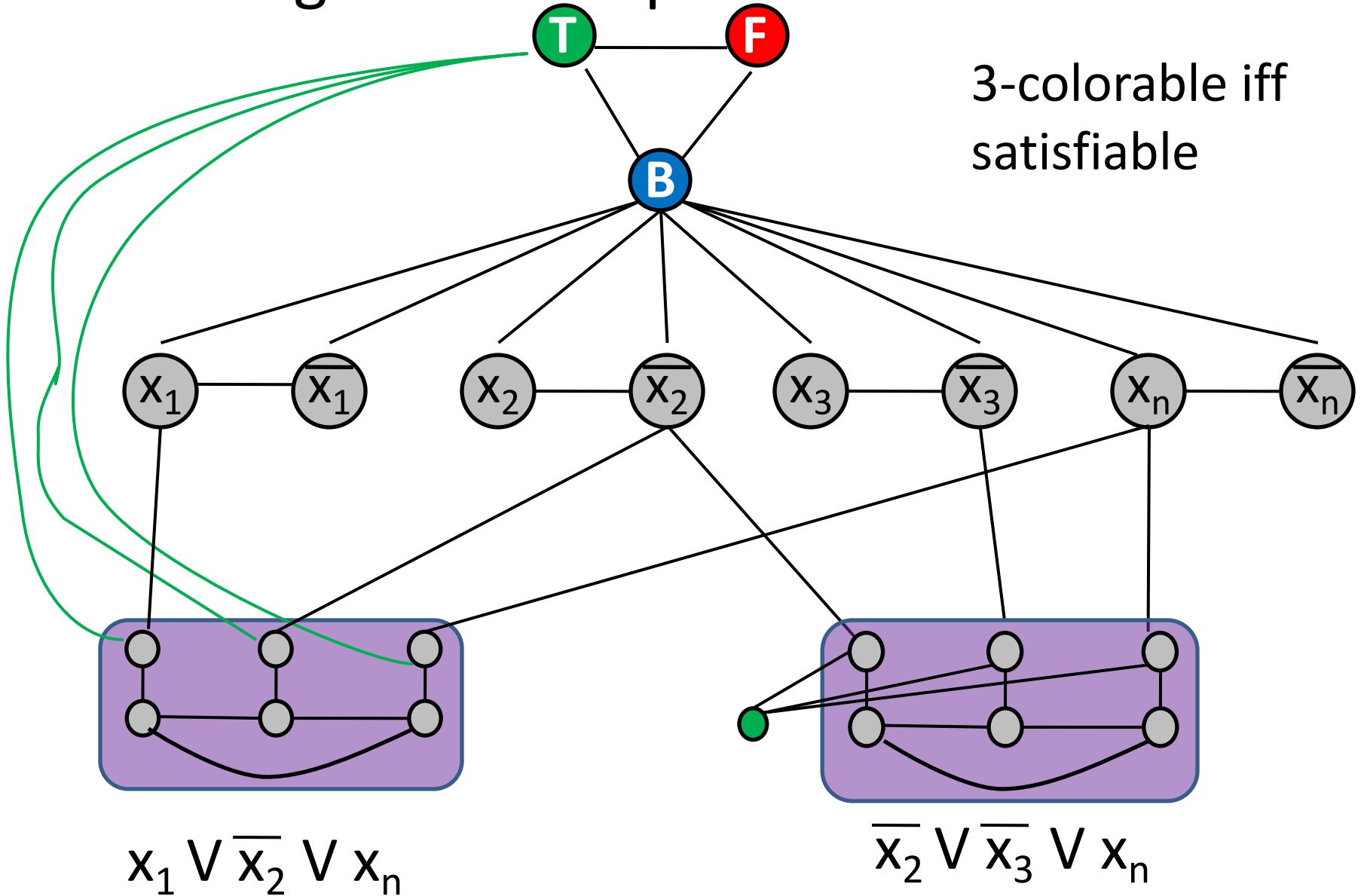


# 3-Coloring is NP-Complete



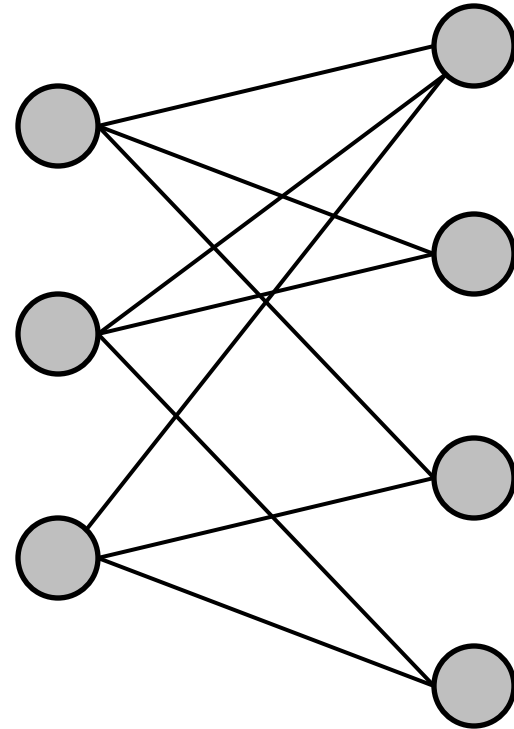
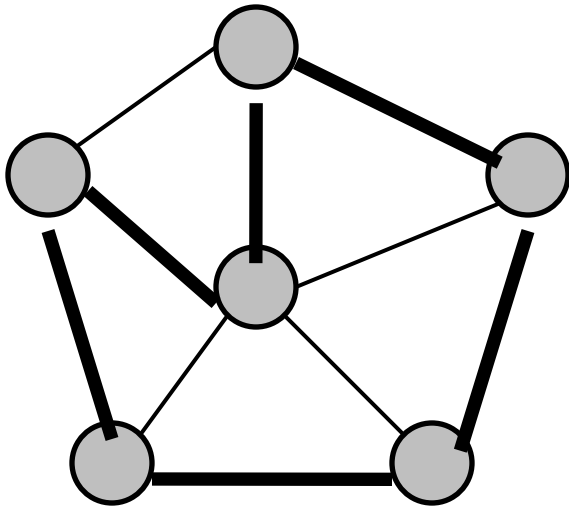
$$x_1 \vee \bar{x}_2 \vee x_n$$

# k-Coloring is NP-Complete



## Hamiltonian Cycle:

A cycle in a graph that hits each vertex once.



## Directed Hamiltonian Cycle:

same, but in a directed graph

# Directed Ham Cycle is NP-Complete

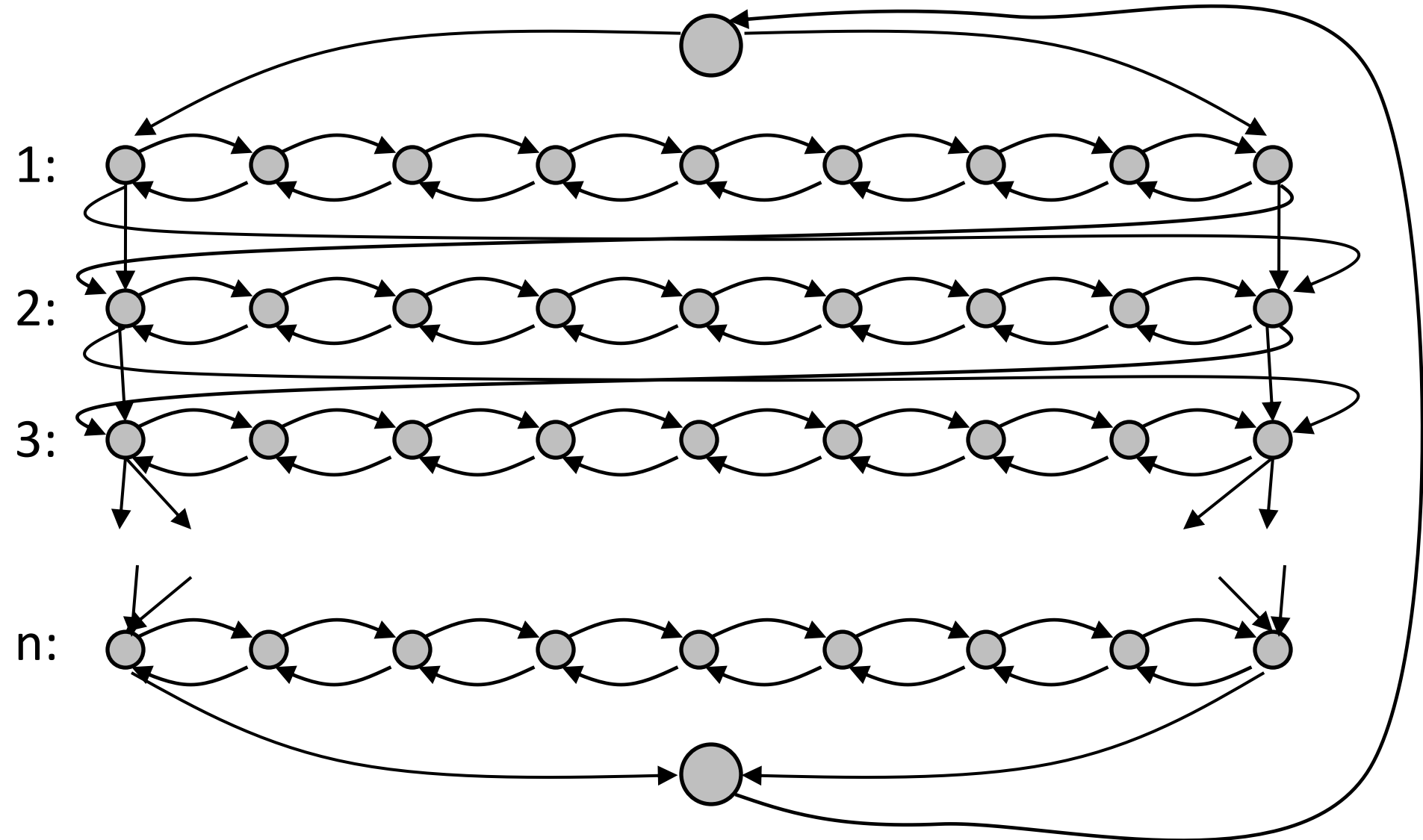
Clearly in NP, because can check if a cycle is Hamiltonian

To prove NP-hard, will show

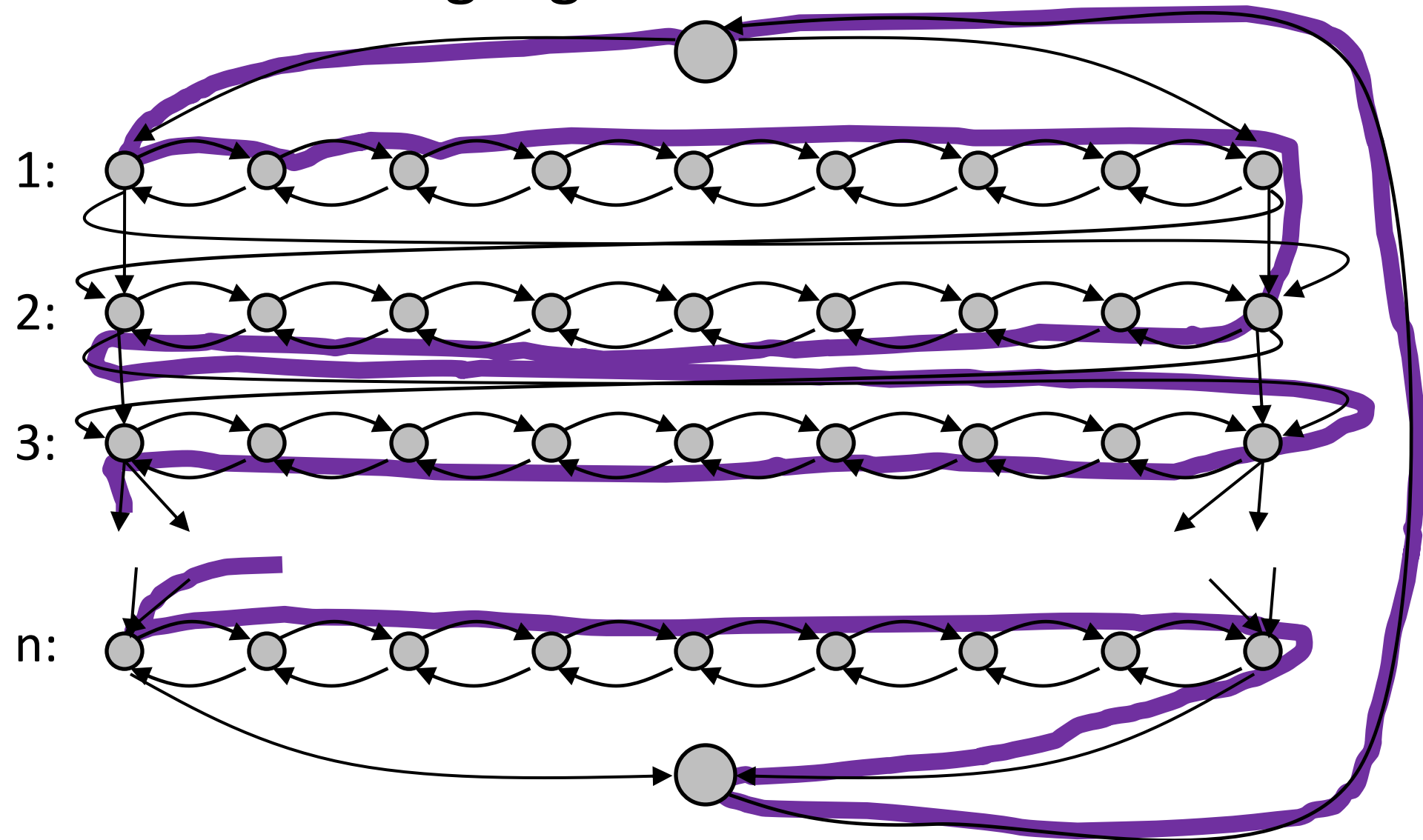
$$3\text{-SAT} \leq_p \text{Directed Ham Cycle}$$

Produce directed graph  $G = (V, E)$  that has Ham Cycle iff the clauses are satisfiable

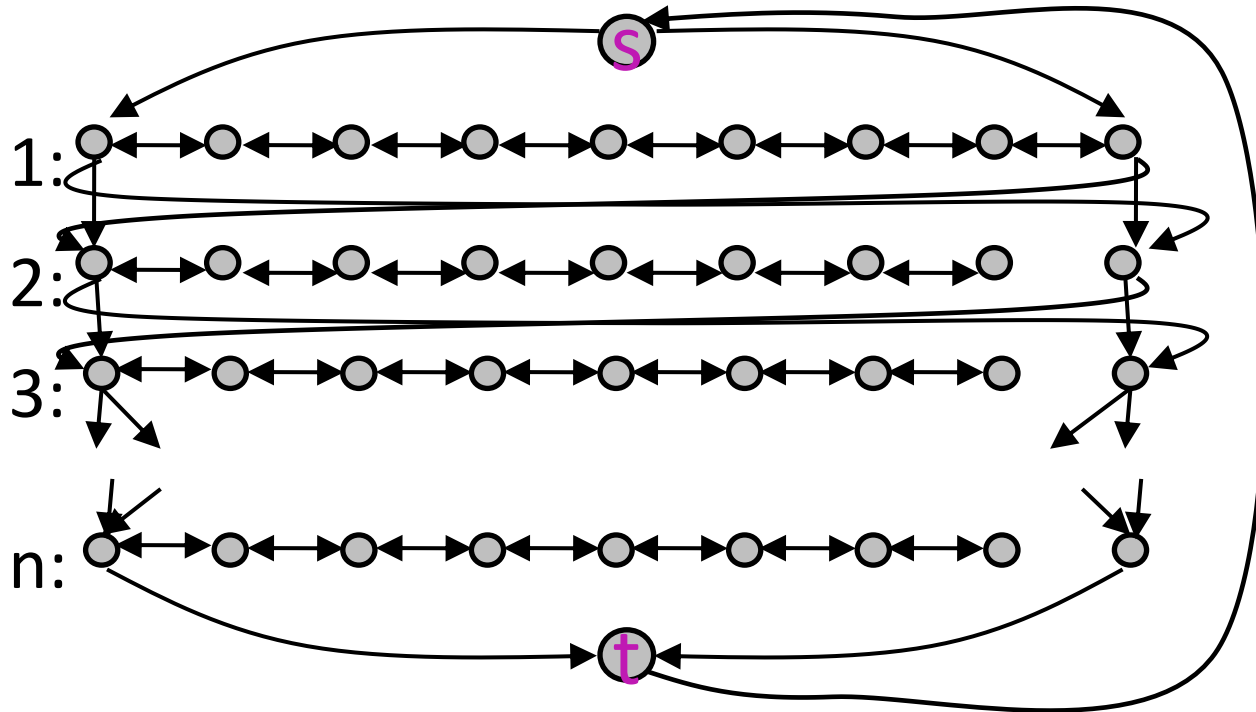
Start: create graph with  $2^n$  Ham Cycles,  
then create gadgets to restrict them



Start: create graph with  $2^n$  Ham Cycles,  
then create gadgets to restrict them

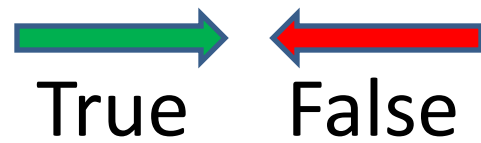


Start: create graph with  $2^n$  Ham Cycles,  
then create gadgets to restrict them

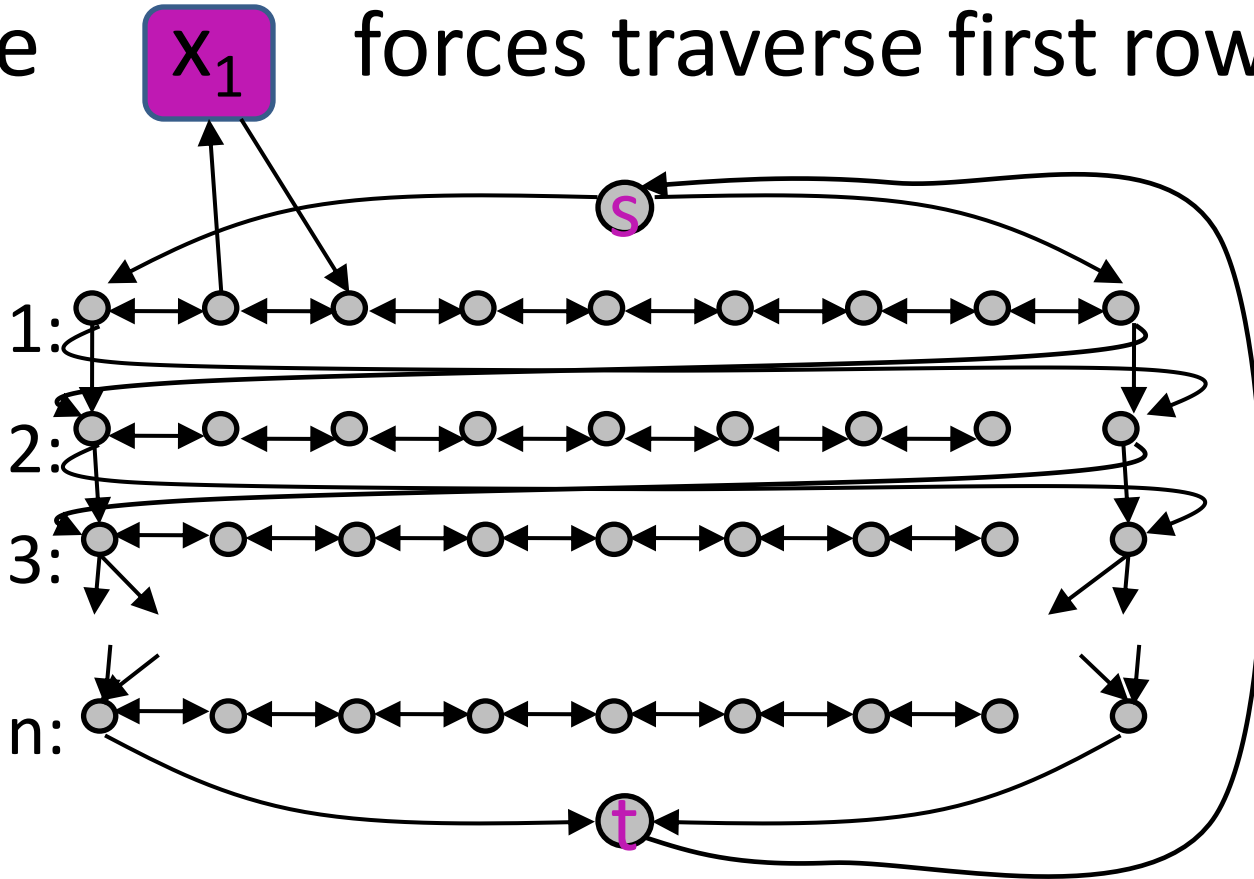


Must go top-to-bottom, and can traverse each row  
left-to-right (True) or right-to-left (False)

# Clause gadgets

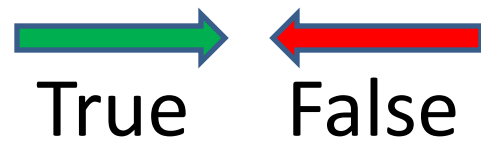


clause  $x_1$  forces traverse first row

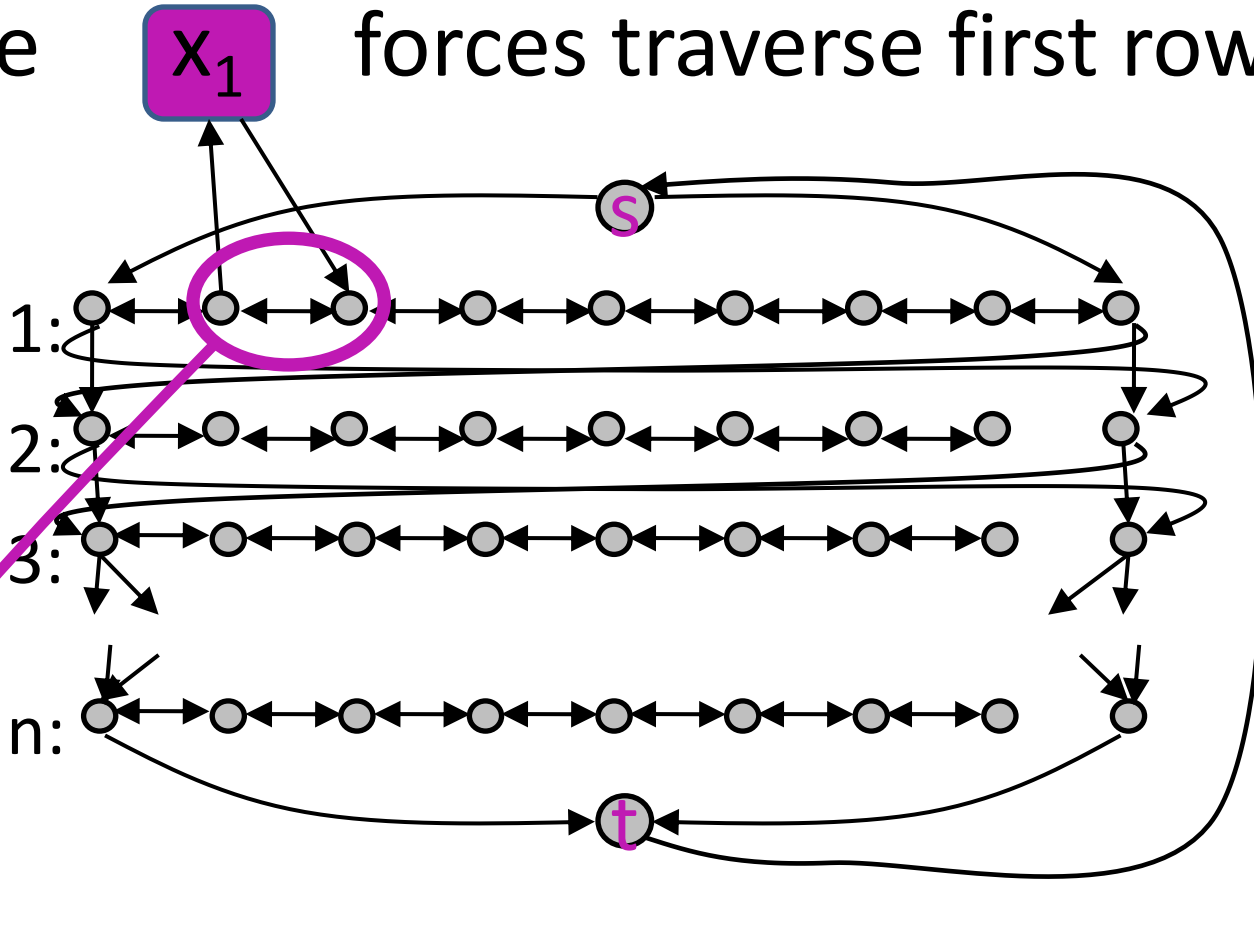




# Clause gadgets

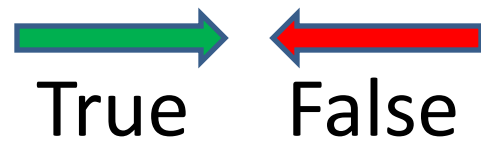


clause  $x_1$  forces traverse first row

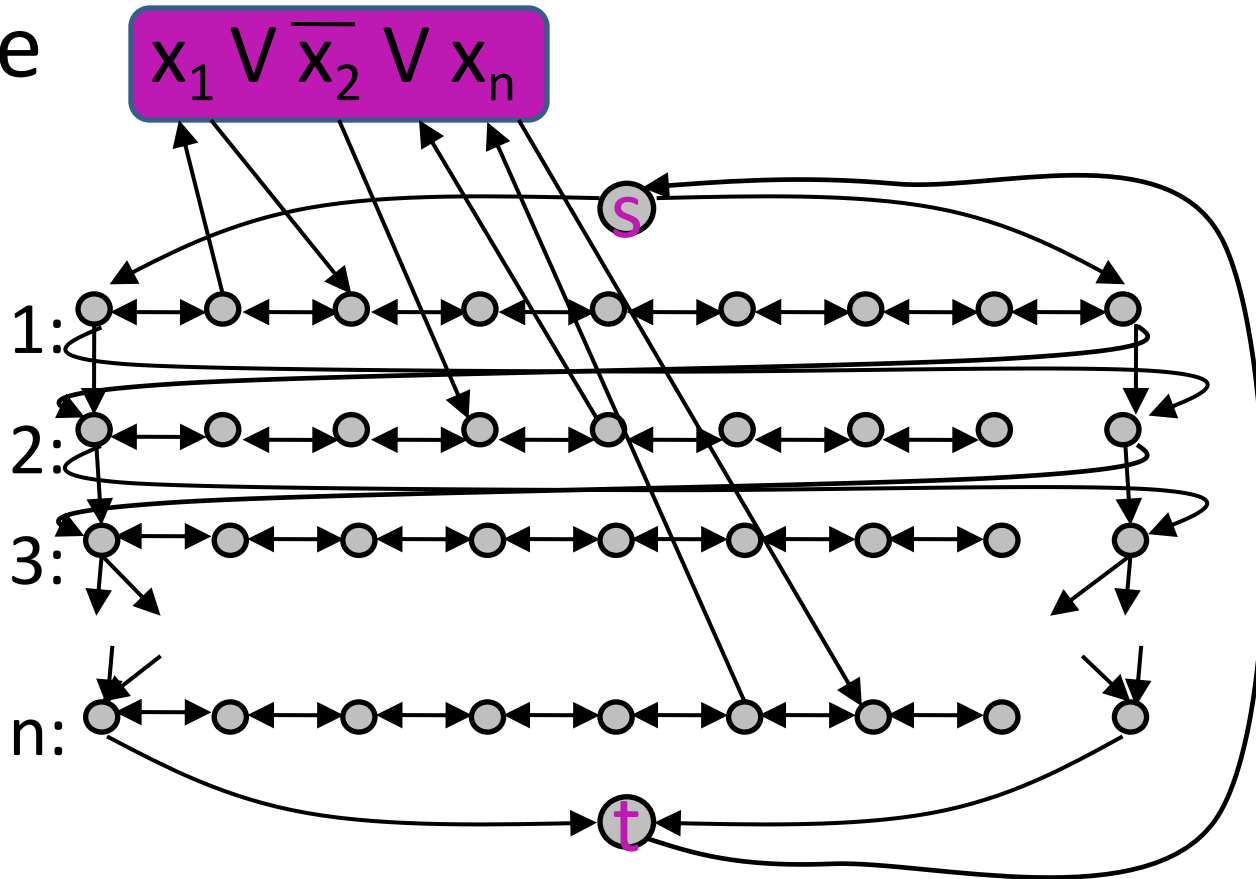


All edges to these vertices are shown.

# Clause gadgets

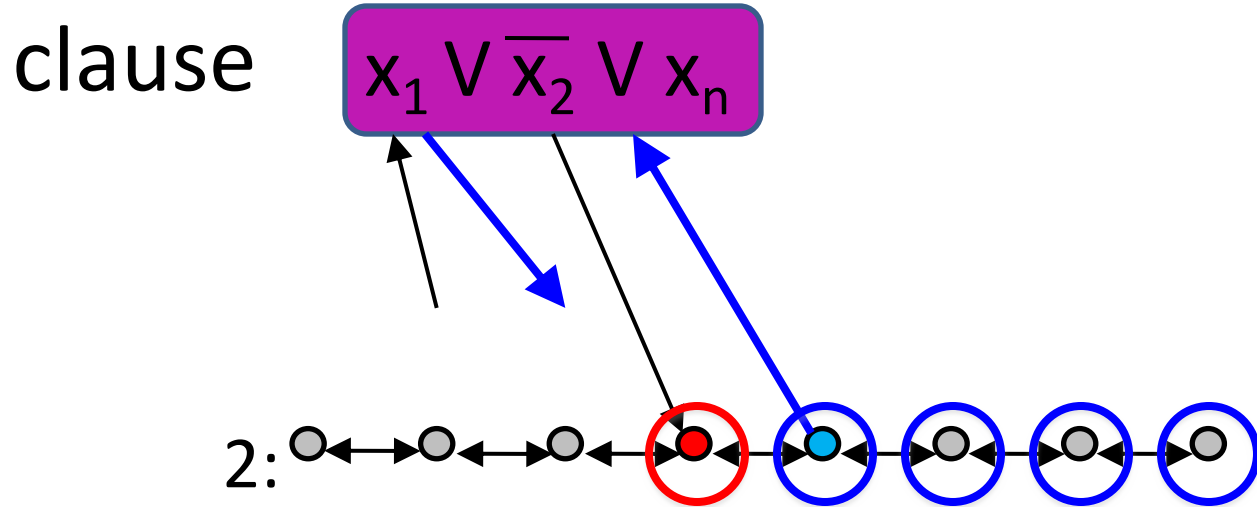
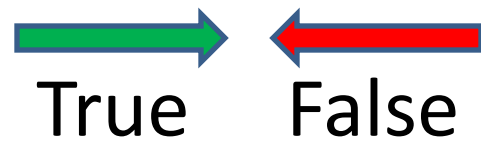


clause



Forces traverse 1  $\rightarrow$ , or 2  $\leftarrow$ , or n  $\rightarrow$

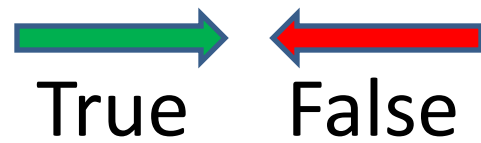
# Clause gadgets



To see must come back to same row,  
note that if do not is no Hamiltonian  
path through unused down-link:

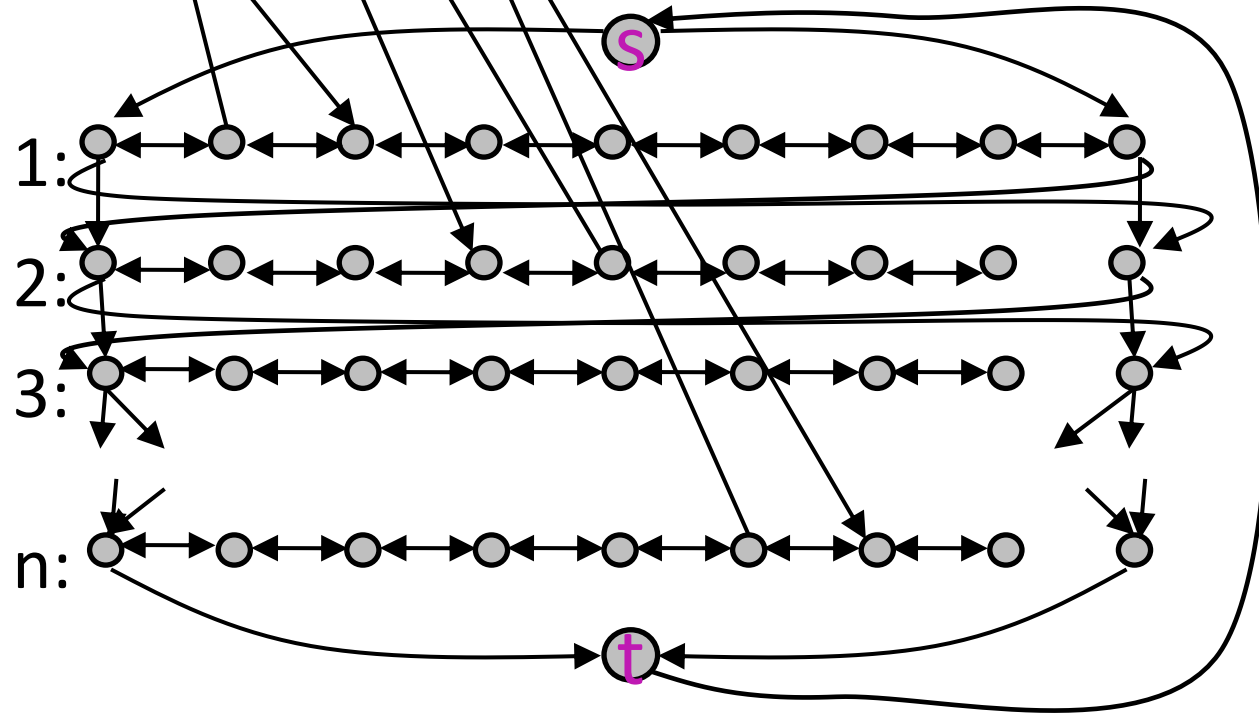
If go through red node from left, is no way out  
if blue node and clause node have been used.

# Ham cycle iff satisfiable

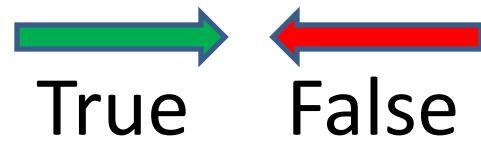


$$x_1 \vee \overline{x_2} \vee x_n$$

Pf. If satisfiable,  
traverse in order  
indicated by vars,  
picking up each  
clause once using  
some true term.

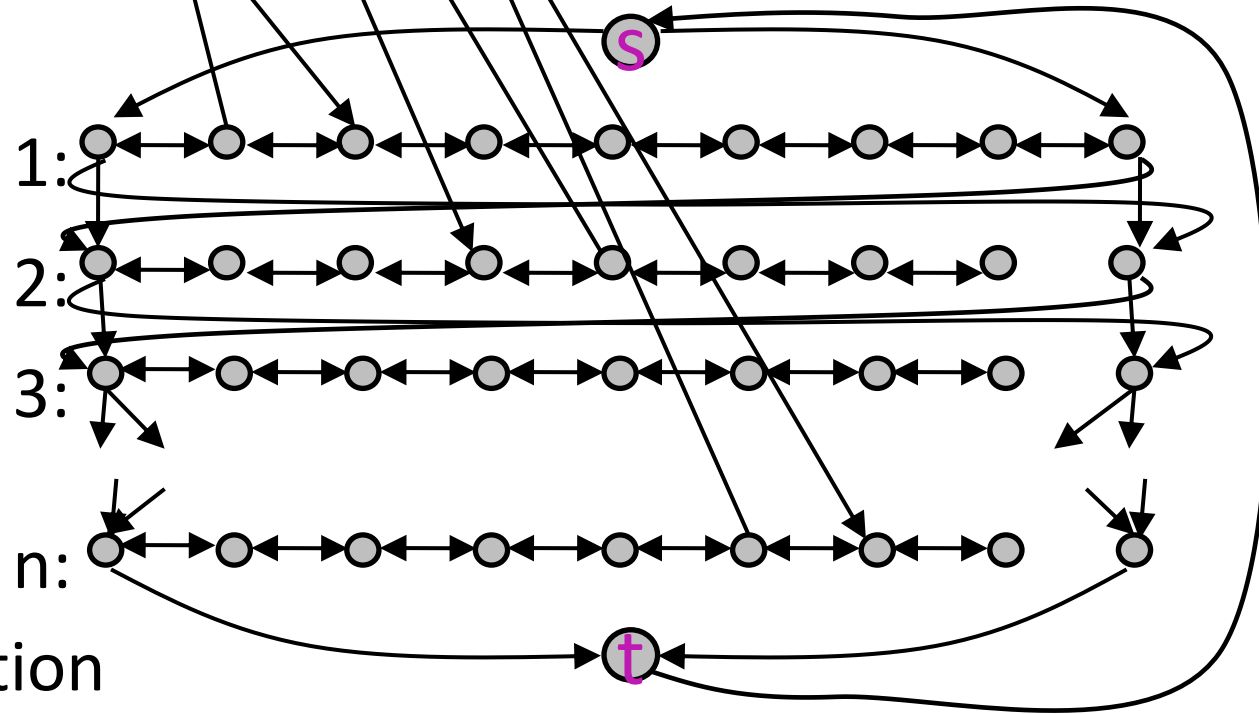


# Ham cycle iff satisfiable



$x_1 \vee \bar{x}_2 \vee x_n$

Pf. If Ham Cycle,  
must go top to bot



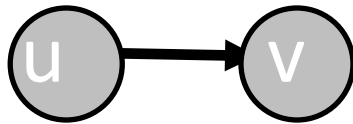
assign vars by direction

if visit each clause node, then is made true by term on row from which make the visit.

# Directed Ham Cycle $\leq_p$ Ham Cycle

1. In directed problem,  
answer same if reverse all arrows.
2. To transform to undirected,  
replace each vertex  $v$  with three vertices:

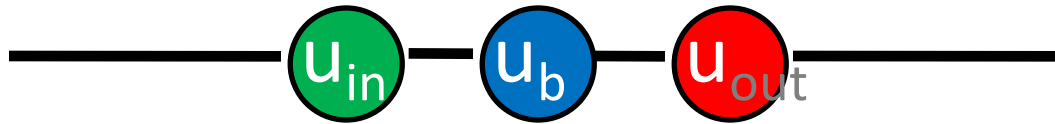
$v_{in}, v_{base}, v_{out}$



Replace directed  $(u,v)$  edge with  $(u_{out}, v_{in})$



# Directed Ham Cycle $\leq_p$ Ham Cycle

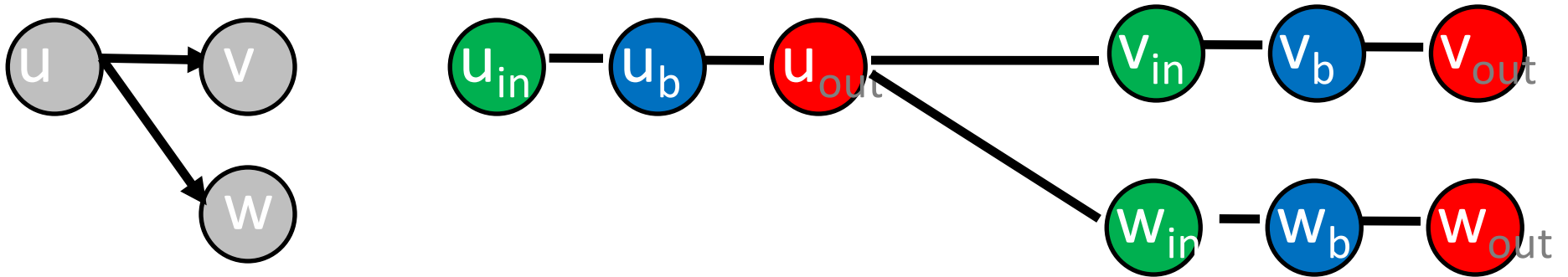


Claim: If these are only edges to  $u_b$ ,  
then in every Hamiltonian cycle  
 $u_b$  must be adjacent to  $u_{in}$

Proof: if it is not, then once enter  $u_b$   
can not get out

# Directed Ham Cycle $\leq_p$ Ham Cycle

Replace directed  $(u,v)$  edge with  $(u_{out}, v_{in})$

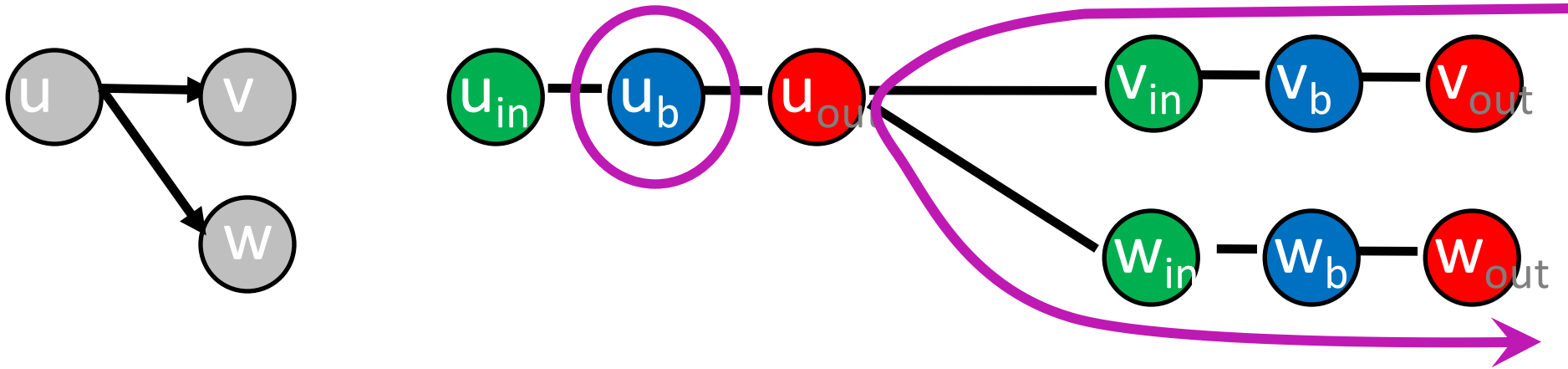


Directed Ham Cycle in original  $\rightarrow$  Ham Cycle



# Directed Ham Cycle $\leq_p$ Ham Cycle

Replace directed  $(u,v)$  edge with  $(u_{out}, v_{in})$

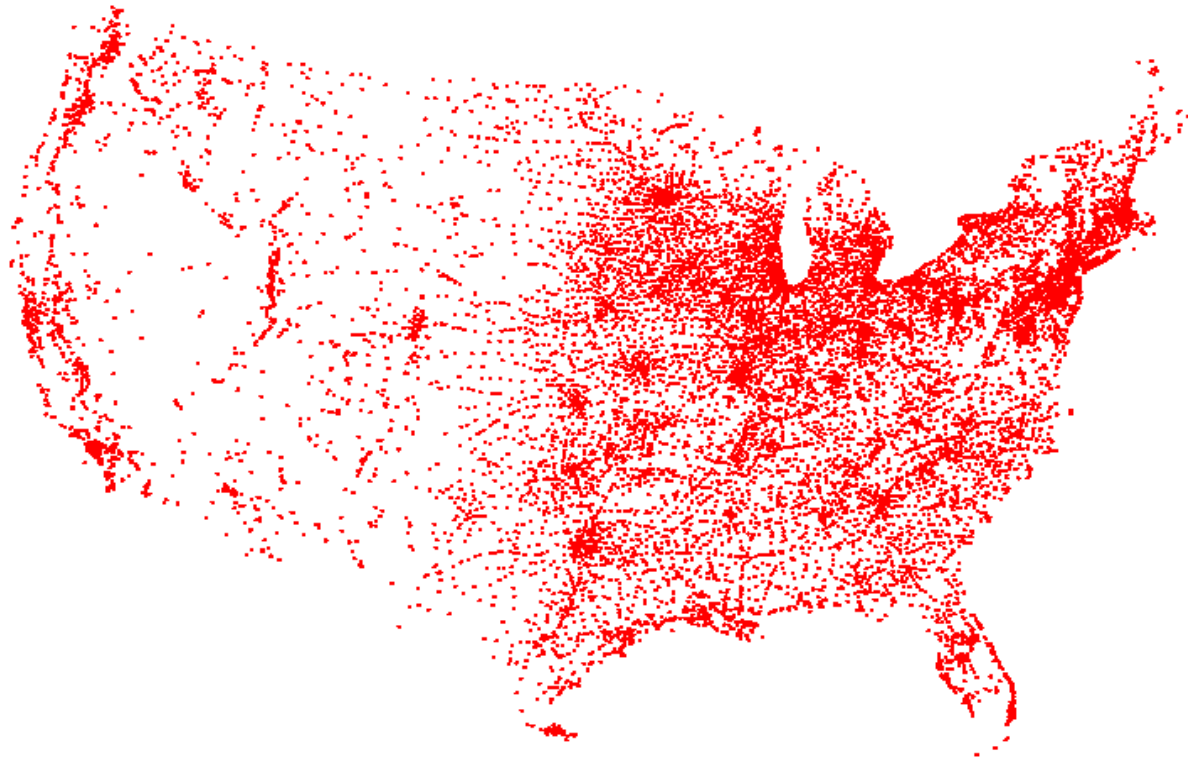


Lemma:

Every Ham Cycle in the undirected graph must go **in, base, out, in, base, out, in, base, out**, etc, must correspond to a Ham Cyc in directed graph

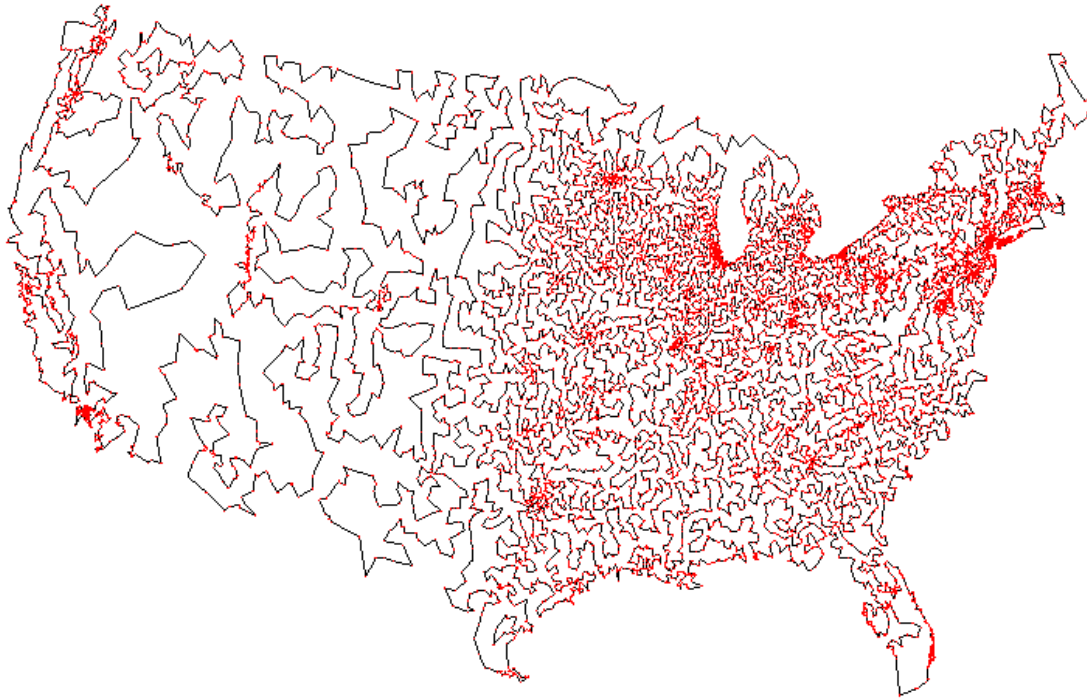
# TSP (Travelling Salesperson Problem)

Given  $n$  locations, a distance function  $d(u,v)$  and a total distance  $D$ , does there exist a tour through all locations of total distance at most  $L$ ?



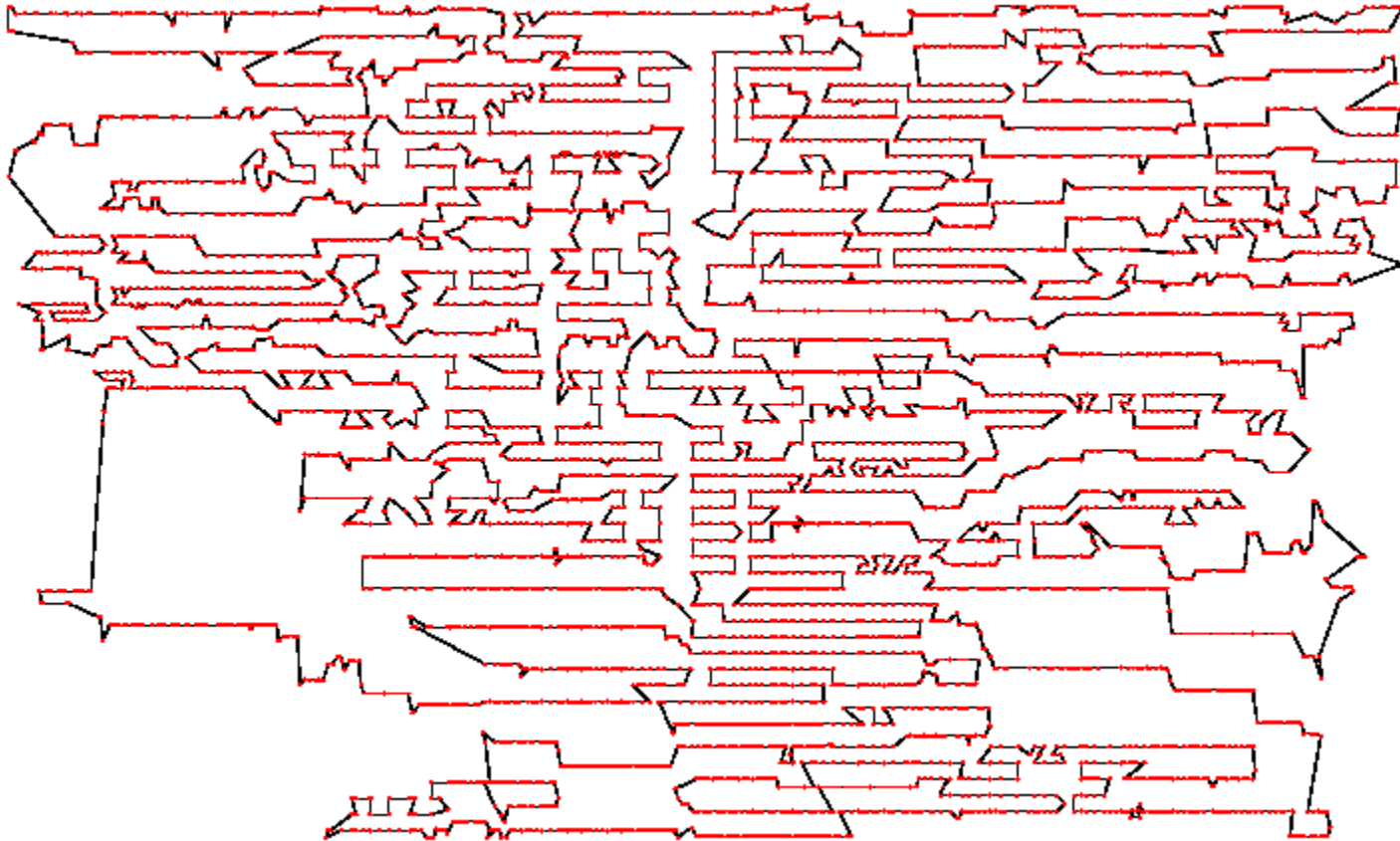
# TSP (Travelling Salesperson Problem)

Given  $n$  locations, a distance function  $d(u,v)$  and a total distance  $D$ , does there exist a tour through all locations of total distance at most  $L$ ?



# RL5915 optimal solution

An optimal solution for [RL5915](#) is given by the following tour, which has length 565530.



[http://www.tsp.gatech.edu/rl5915/rl5915\\_sol.html](http://www.tsp.gatech.edu/rl5915/rl5915_sol.html)

# TSP is NP-complete

Ham Cycle  $\leq_p$  TSP

Given graph  $G = (V, E)$ ,  
create one location for each vertex,

$d(u, v) = 1$  if  $(u, v) \in E$   
2 otherwise

Target distance =  $|V|$

A tour of all locations that returns to start and has total length  $|V|$  must use exactly  $n$  edges of  $G$

# TSP is NP-complete

Ham Cycle  $\leq_p$  TSP

Given graph  $G = (V, E)$ ,  
create one location for each vertex,

$d(u, v) = 1$  if  $(u, v) \in E$   
2 otherwise

This is an abstract distance function.

# TSP is NP-complete

Ham Cycle  $\leq_p$  TSP

Given graph  $G = (V, E)$ ,  
create one location for each vertex,

$d(u, v) = 1$  if  $(u, v) \in E$   
2 otherwise

This is an abstract distance function.

Remains NP-hard for integer points in plane.

# Issue with Planar TSP

If input is locations of points, instead of distances

The problem is not known to be in NP, because do not know if can compare distances in polynomial time.

For integers  $x_1, \dots, x_n$  integer  $t$ , do not have poly time algorithm to test if

$$\sum_i \sqrt{x_i} \leq t$$



# Classic Nintendo Games are (NP-)Hard

Greg Aloupis\*

Erik D. Demaine<sup>†</sup>

Alan Guo<sup>†‡</sup>

March 9, 2012

## Abstract

We prove NP-hardness results for five of Nintendo's largest video game franchises: Mario, Donkey Kong, Legend of Zelda, Metroid, and Pokémon. Our results apply to Super Mario Bros. 1, 3, Lost Levels, and Super Mario World; Donkey Kong Country 1–3; all Legend of Zelda games except Zelda II: The Adventure of Link; all Metroid games; and all Pokémon role-playing games. For Mario and Donkey Kong, we show NP-completeness. In addition, we observe that several games in the Zelda series are PSPACE-complete.

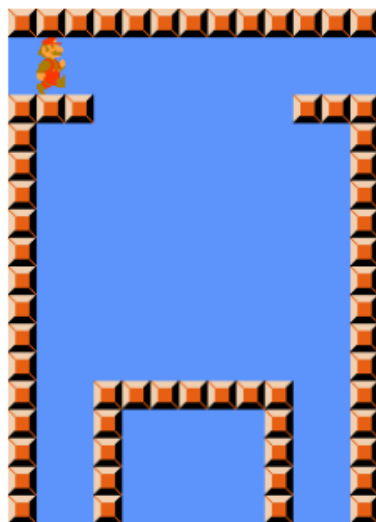


Figure 4: Variable gadget for Mario

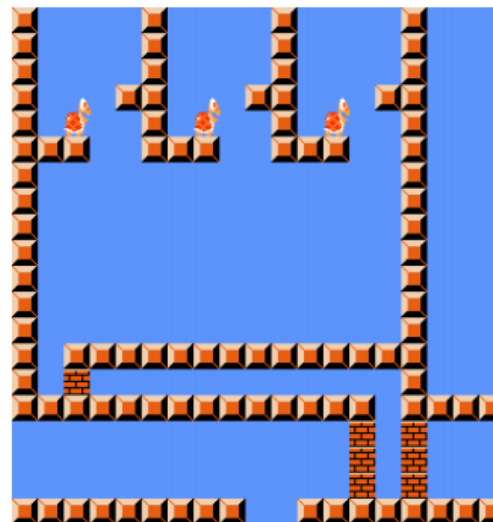


Figure 5: Clause gadget for Mario

# Candy Crush is NP-hard

Toby Walsh

*NICTA and University of NSW, Sydney, Australia*

---

## **Abstract**

We prove that playing Candy Crush to achieve a given score in a fixed number of swaps is NP-hard.

*Keywords:* computational complexity, NP-completeness, Candy Crush.

---

# The Legend of Zelda: The Complexity of Mechanics

Jeffrey Bosboom\*

Josh Brunner\*

Michael Coulombe\*

Erik D. Demaine\*

Dylan H. Hendrickson\*

Jayson Lynch<sup>†</sup>

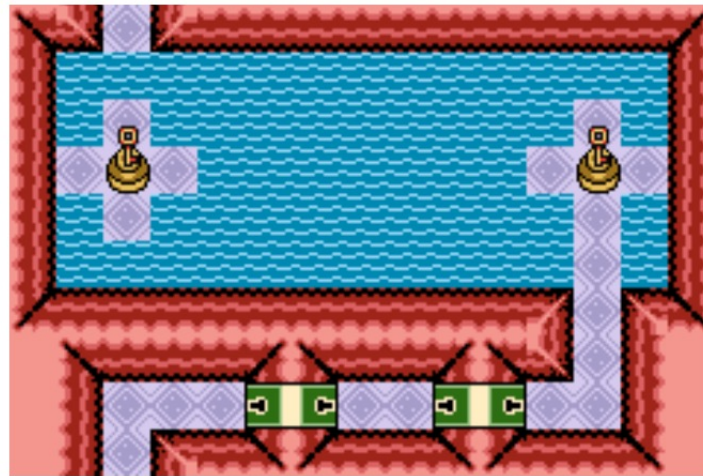
Elle Najt<sup>‡</sup>

## Abstract

We analyze some of the many game mechanics available to Link in the classic Legend of Zelda series of video games. In each case, we prove that the generalized game with that mechanic is polynomial, NP-complete, NP-hard and in PSPACE, or PSPACE-complete. In the process we give an overview of many of the hardness proof techniques developed for video games over the past decade: the motion-planning-through-gadgets framework, the planar doors framework, the doors-and-buttons framework, the “Nintendo” platform game / SAT framework, and the collectible tokens and toll roads / Hamiltonicity framework.

## 1 Introduction

“It’s dangerous to go alone! Take this.”



**Figure 3:** Platforms and locked door finale gadgets in the construction of Theorem [4.1](#) from a two-vertex