CSE 421

NP-Completeness

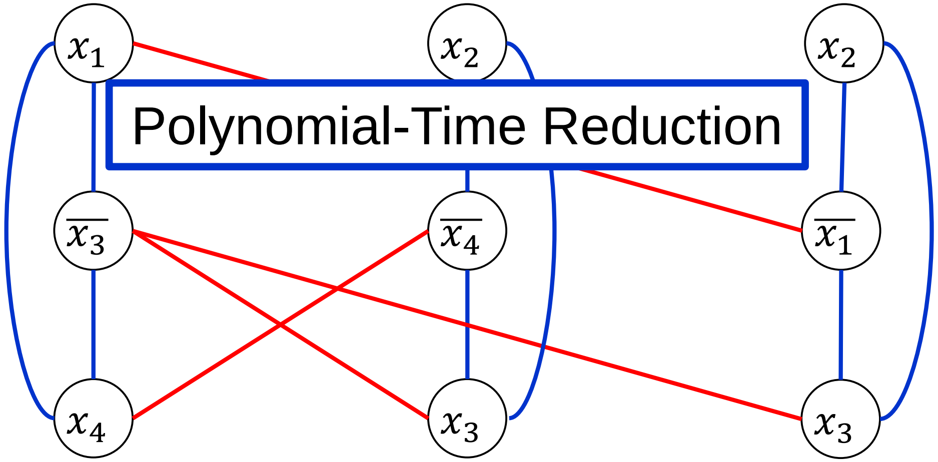
Yin Tat Lee

3-SAT $\leq_p$ Independent Set

Map a 3-CNF to (G,k). Say m is number of clauses

- Create a vertex for each literal
- Joint two literals if
  - They belong to the same clause (blue edges)
  - The literals are negations, e.g., $x_i, \overline{x_i}$ (red edges)
- Set k be the # of clauses.

$$(x_1 \vee \overline{x_3} \vee x_4) \wedge (x_2 \vee \overline{x_4} \vee x_3) \wedge (x_2 \vee \overline{x_1} \vee x_3)$$



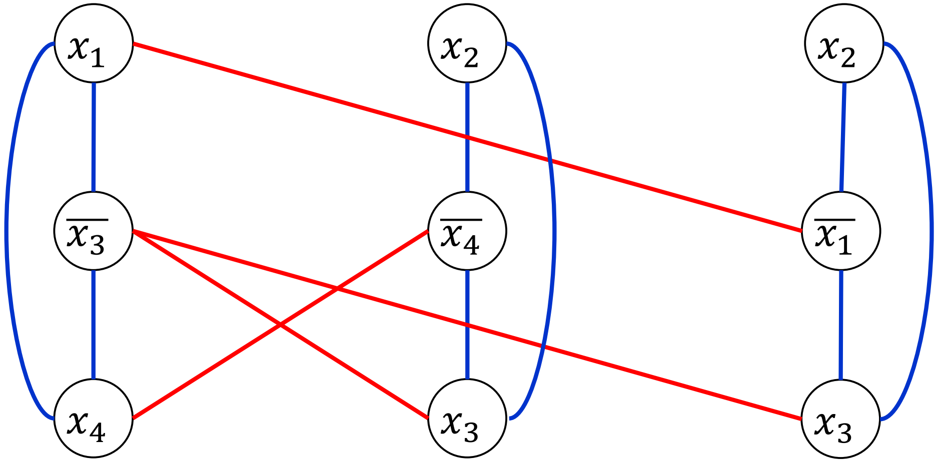
Correctness of 3-SAT $\leq_p$ Indep Set

F satisfiable => An independent of size k

Given a satisfying assignment, Choose one node from each clause where the literal is satisfied

$$(x_1 \vee \overline{x_3} \vee x_4) \wedge (x_2 \vee \overline{x_4} \vee x_3) \wedge (x_2 \vee \overline{x_1} \vee x_3)$$

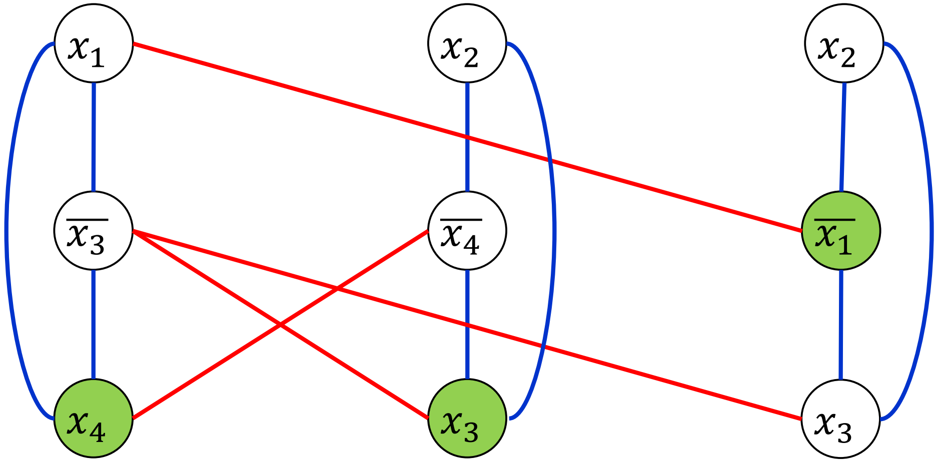
Satisfying assignment: $x_1 = T, x_2 = F, x_3 = T, x_4 = F$



- S has exactly one node per clause => No blue edges between S
- S follows a truth-assignment => No red edges between S
- S has one node per clause => |S|=k

Correctness of 3-SAT $\leq_p$ Indep Set

<u>An independent set of size k => A satisfying assignment</u>
Given an independent set S of size k.
S has exactly one vertex per clause (because of blue edges)
S does not have $x_i, \overline{x_i}$ (because of red edges)
So, S gives a satisfying assignment



Satisfying assignment: $x_1 = F, x_2 =?, x_3 = T, x_4 = T$

$$(x_1 \vee \overline{x_3} \vee x_4) \wedge (x_2 \vee \overline{x_4} \vee x_3) \wedge (x_2 \vee \overline{x_1} \vee x_3)$$

More NP-completeness

- Subset-Sum problem
  (Decision version of Knapsack)
  - Given **n** integers **$w_1,\ldots,w_n$** and integer **W**
  - Is there a subset of the **n** input integers that adds up to exactly **W**?

- **O(nW)** solution from dynamic programming but if **W** and each **w_i** can be **n** bits long then this is exponential time

3-SAT $\leq_P$ Subset-Sum

- Given a 3-CNF formula with **m** clauses and **n** variables
- Will create 2**m**+2**n** numbers that are **m+n** digits long
  - Two numbers for each variable x_i
    - t_i and f_i (corresponding to x_i being true or x_i being false)
  - Two extra numbers for each clause
    - u_j and v_j (filler variables to handle number of false literals in clause C_j)

3-SAT $\le_P$ Subset-Sum

$C_3=(x_1 \vee \neg x_2 \vee x_5)$

				i						j		
	1	2	3	4	…	n	1	2	3	4	…	m
t_1	1	0	0	0	…	0	0	0	1	0	…	1
f_1	1	0	0	0	…	0	1	0	0	1	…	0
t_2	0	1	0	0	…	0	0	1	0	0	…	1
f_2	0	1	0	0	…	0	0	0	1	1	…	0
					…				….			
$u_1=v_1$	0	0	0	0	…	0	1	0	0	0	…	0
$u_2=v_2$	0	0	0	0	…	0	0	1	0	0	…	0
					…				….			
W	1	1	1	1	…	1	3	3	3	3	…	3

Graph Colorability

- Defn: Given a graph $G=(V,E)$, and an integer k, a k-coloring of G is
  - an assignment of up to k different colors to the vertices of G so that the endpoints of each edge have different colors.
- 3-Color: Given a graph $G=(V,E)$, does G have a 3-coloring?
- Claim: 3-Color is NP-complete
- Proof: 3-Color is in NP:
  - Certificate is an assignment of red,green,blue to the vertices of G
  - Easy to check that each edge is colored correctly

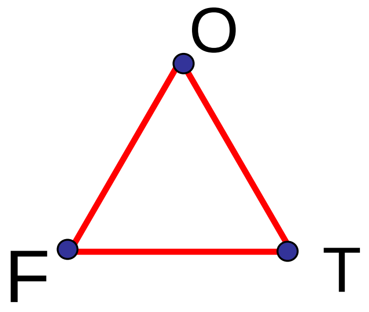
3-SAT $\leq_P$ 3-Color

- Reduction:

  We want to map a 3-CNF formula **F** to a graph **G** so that

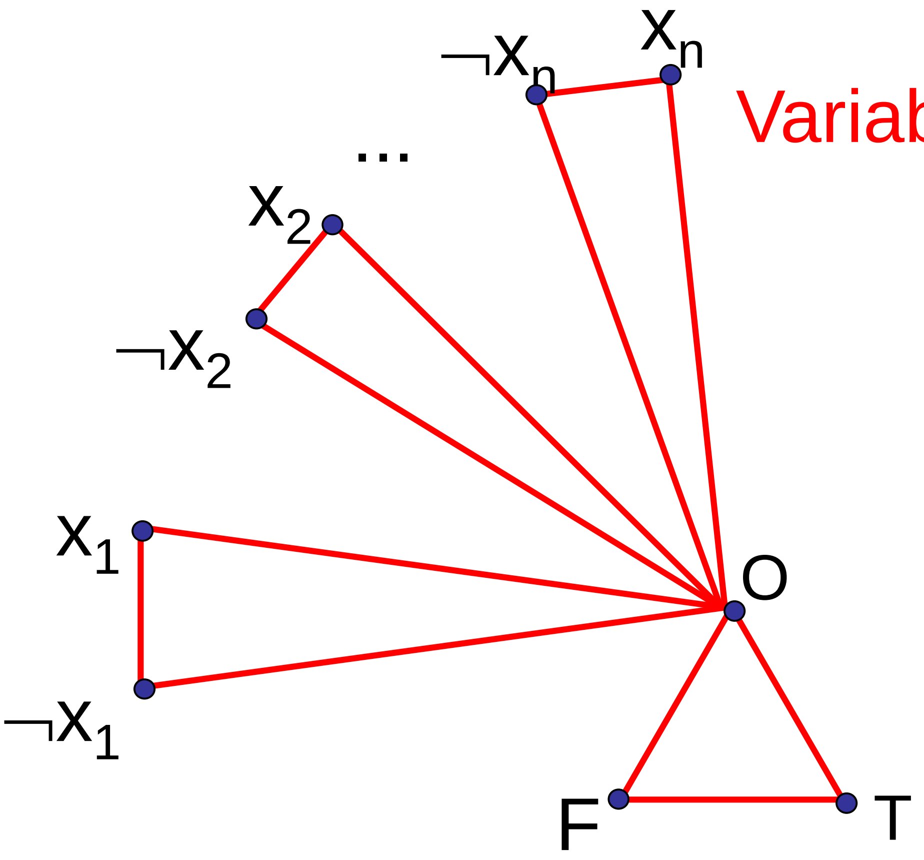
  - **G** is 3-colorable iff **F** is satisfiable

3-SAT $\leq_P$ 3-Color



Base Triangle

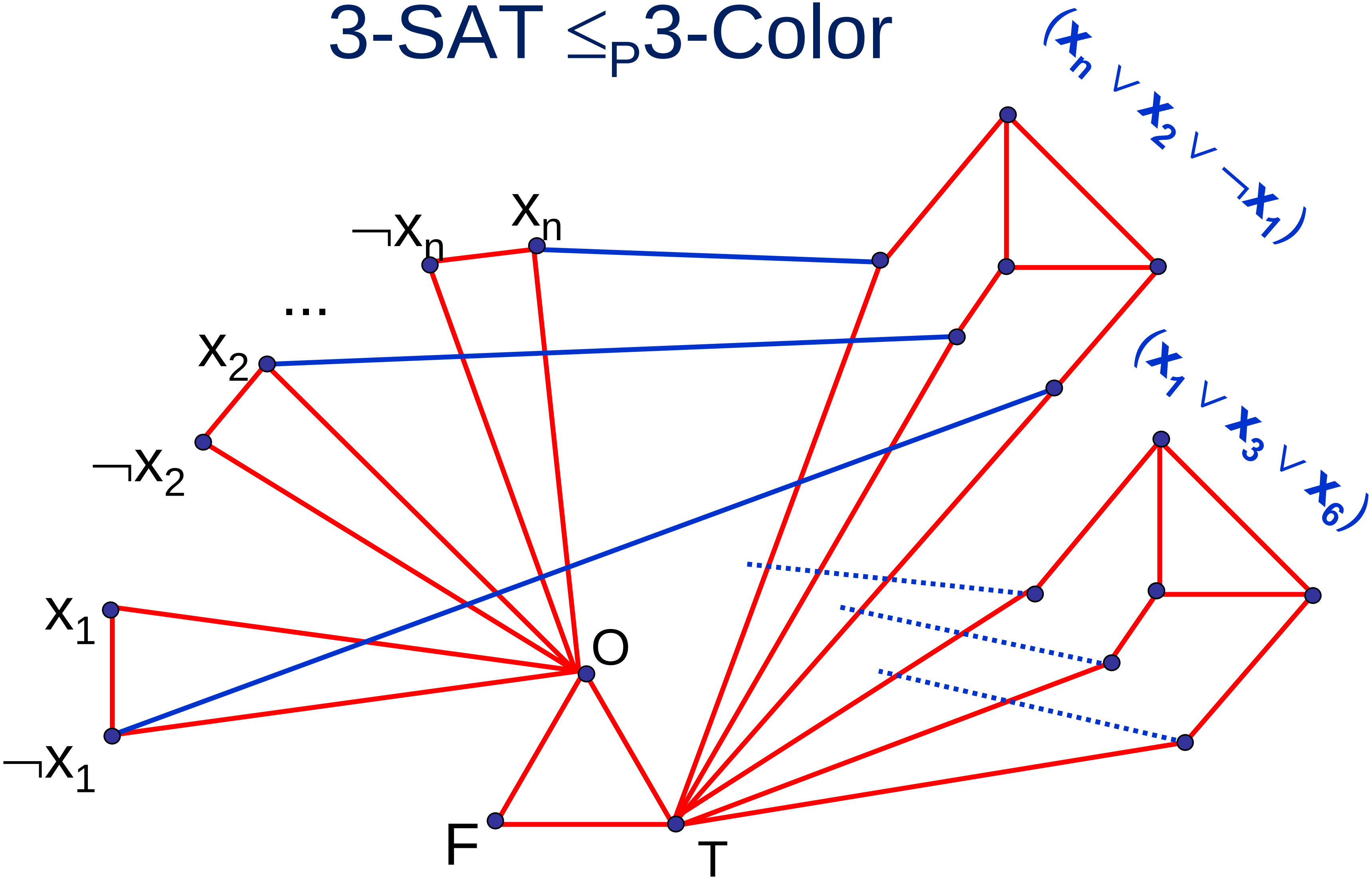
3-SAT $\leq_P$ 3-Color



Variable Part:

in 3-coloring, variable colors correspond to some truth assignment (same color as T or F)

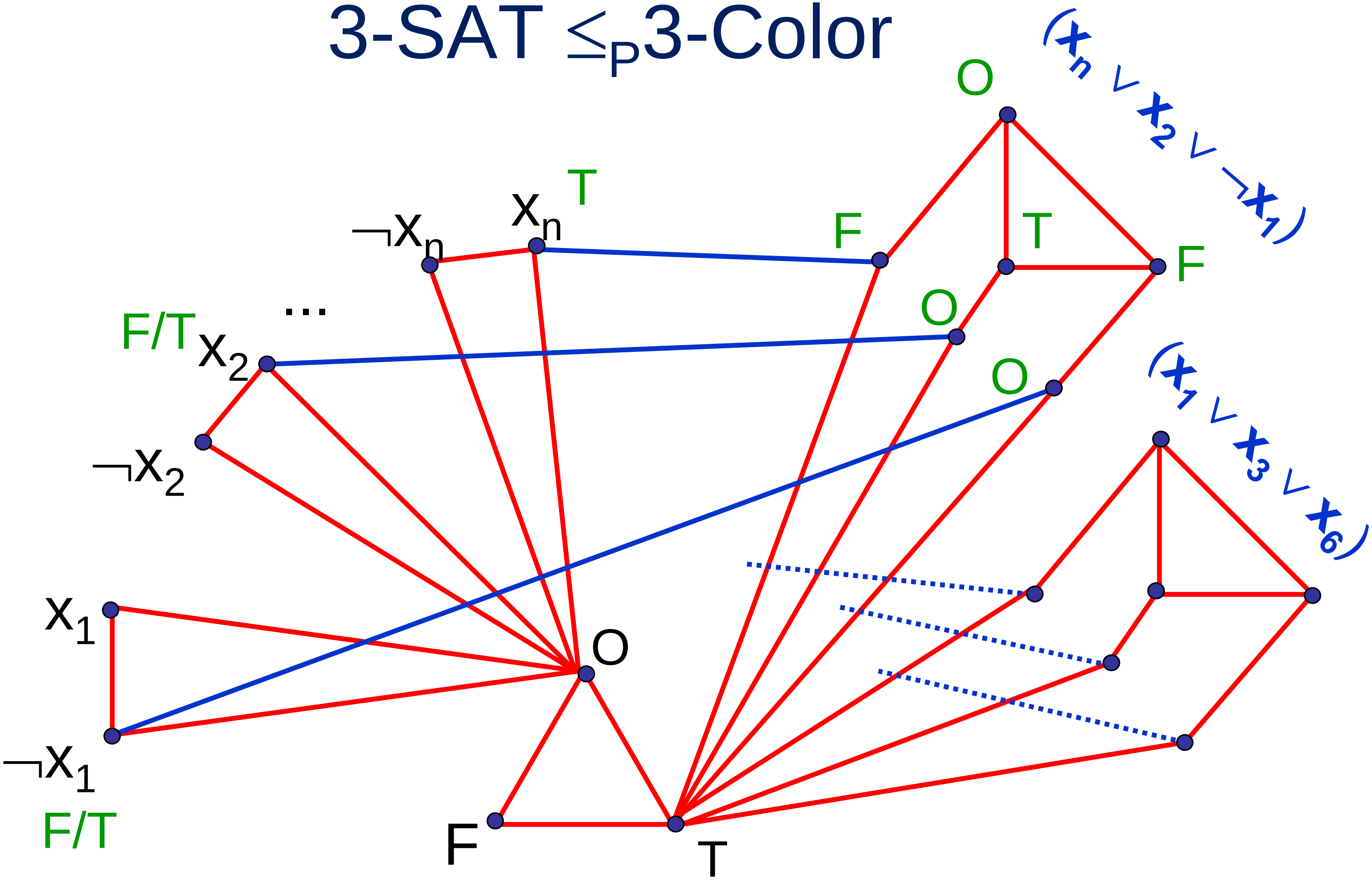
3-SAT $\leq_P$ 3-Color



Clause Part:

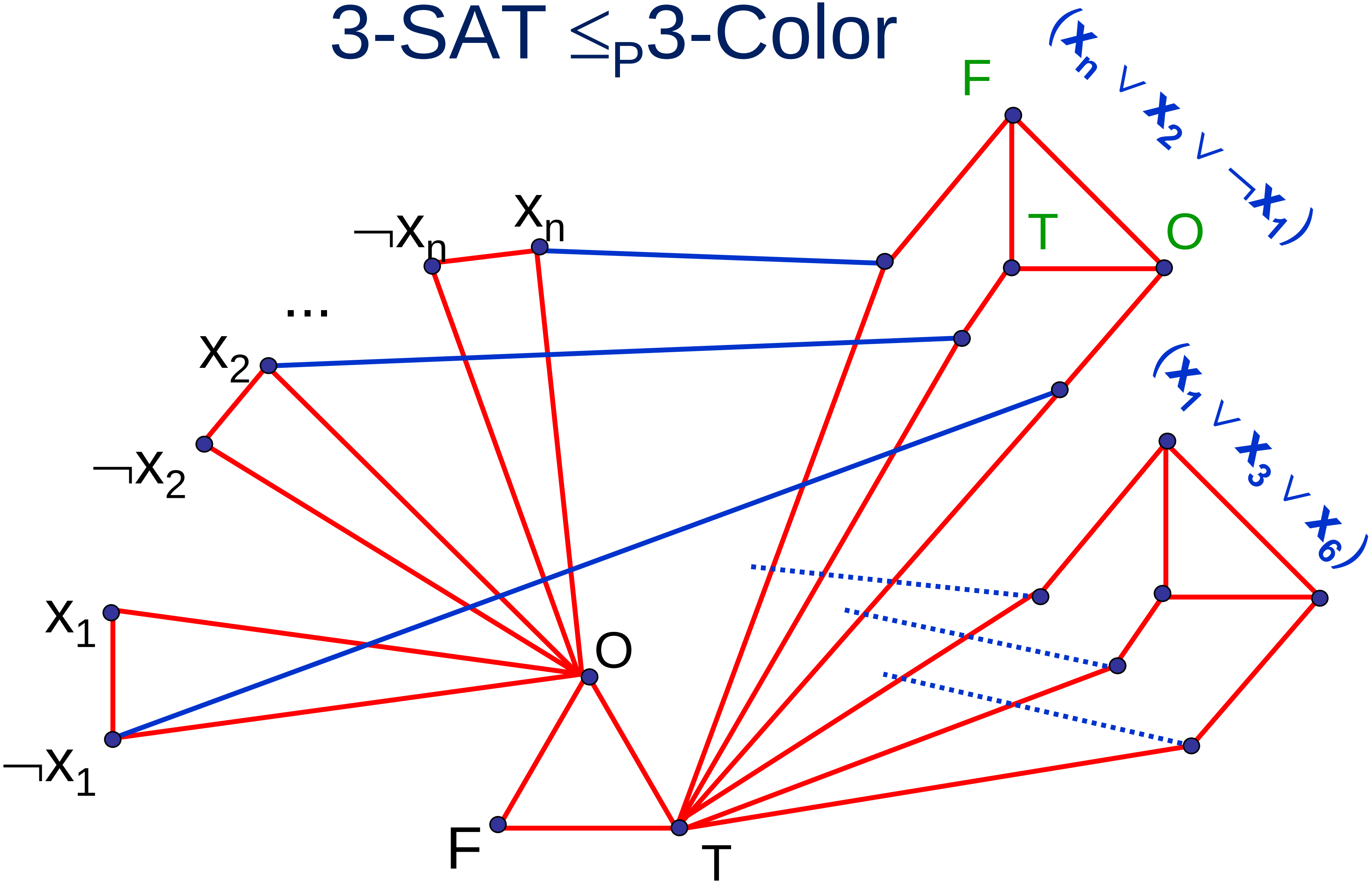
Add one 6 vertex gadget per clause connecting its ‘outer vertices’ to the literals in the clause

3-SAT $\leq_P$ 3-Color



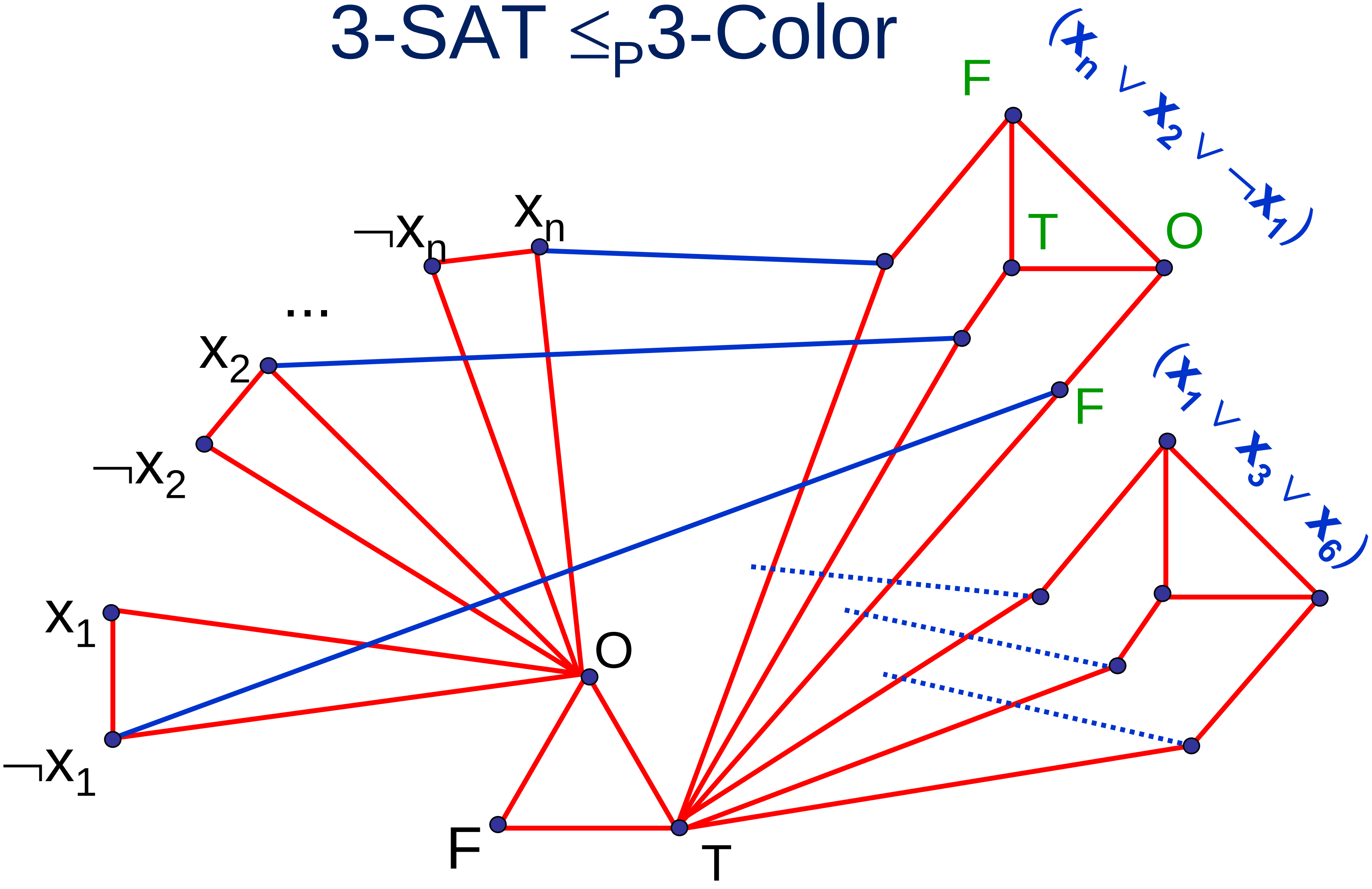
Any truth assignment satisfying the formula can be extended to a 3-coloring of the graph

3-SAT $\leq_P$ 3-Color



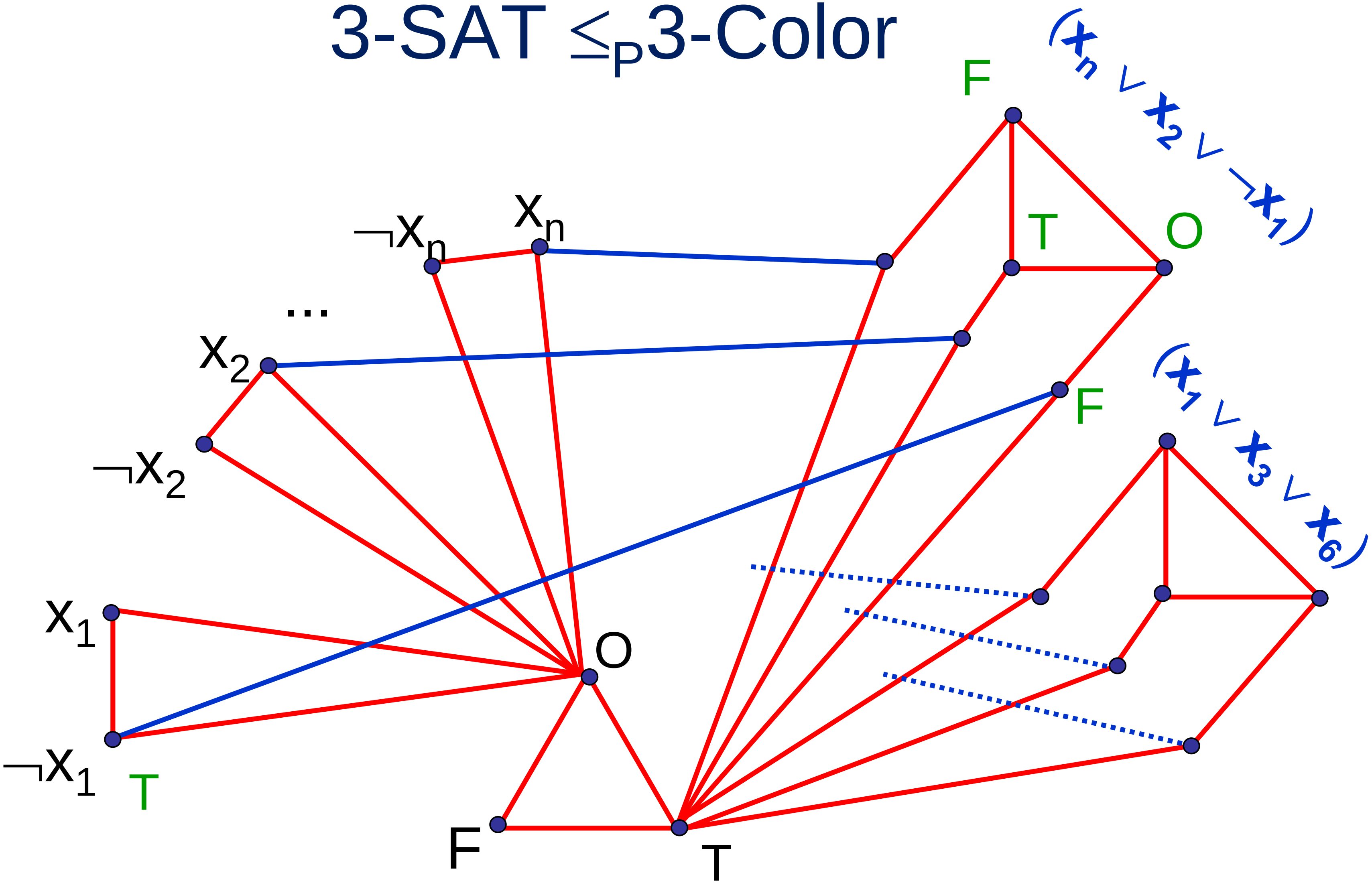
Any 3-coloring of the graph colors each gadget triangle using each color

3-SAT $\le_P$ 3-Color



Any 3-coloring of the graph has an F opposite the O color in the triangle of each gadget

3-SAT $\leq_P$ 3-Color



Any 3-coloring of the graph has T at the other end of the blue edge connected to the F

Summary

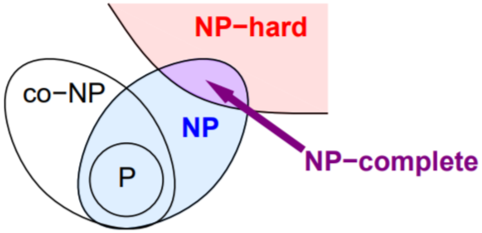
- If a problem is NP-hard it does not mean that all instances are hared, e.g., Vertex-cover has a polynomial-time algorithm in trees

- We learned the crucial idea of polynomial-time reduction. This can be even used in algorithm design, e.g., we know how to solve max-flow so we reduce bipartite matching to max-flow

- NP-Complete problems are the hardest problem in NP

- NP-hard problems may not necessarily belong to NP.

- Polynomial-time reductions are transitive relations



More of what we *think* the world looks like.

CSE 421

Vertex Cover / Set Cover

Yin Tat Lee

Approximation Algorithms

How to deal with NP-complete Problem

Many of the important problems are NP-complete.

SAT, Set Cover, Graph Coloring, TSP, Max IND Set, Vertex Cover, …

So, we cannot find optimum solutions in polynomial time.

What to do instead?

- Find optimum solution of special cases (e.g., random inputs)

- Find near optimum solution in the worst case

Approximation Algorithm

We call an algorithm has approximation ratio $\alpha(n)$ if

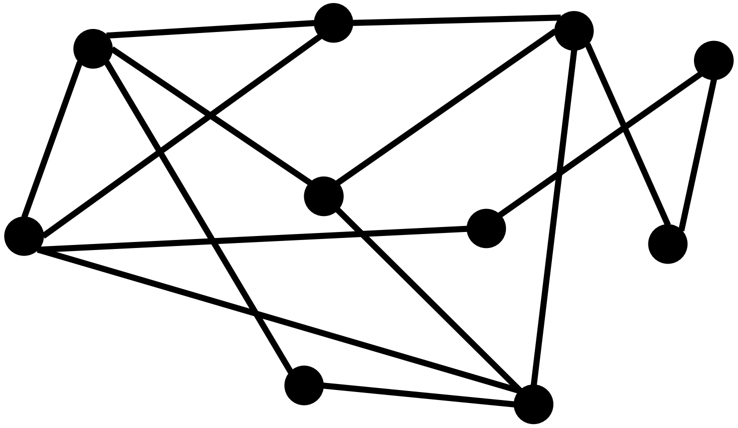
$$\frac{\text{Cost of computed solution}}{\text{Cost of the optimum}} \leq \alpha(n)$$

for any input of length n. (worst case)

Goal: For each NP-hard problem find an poly-time approximation algorithm with the best possible approximation ratio.

Vertex Cover

Given a graph $G = (V, E)$, Find smallest set of vertices touching every edge



Greedy Algorithm?

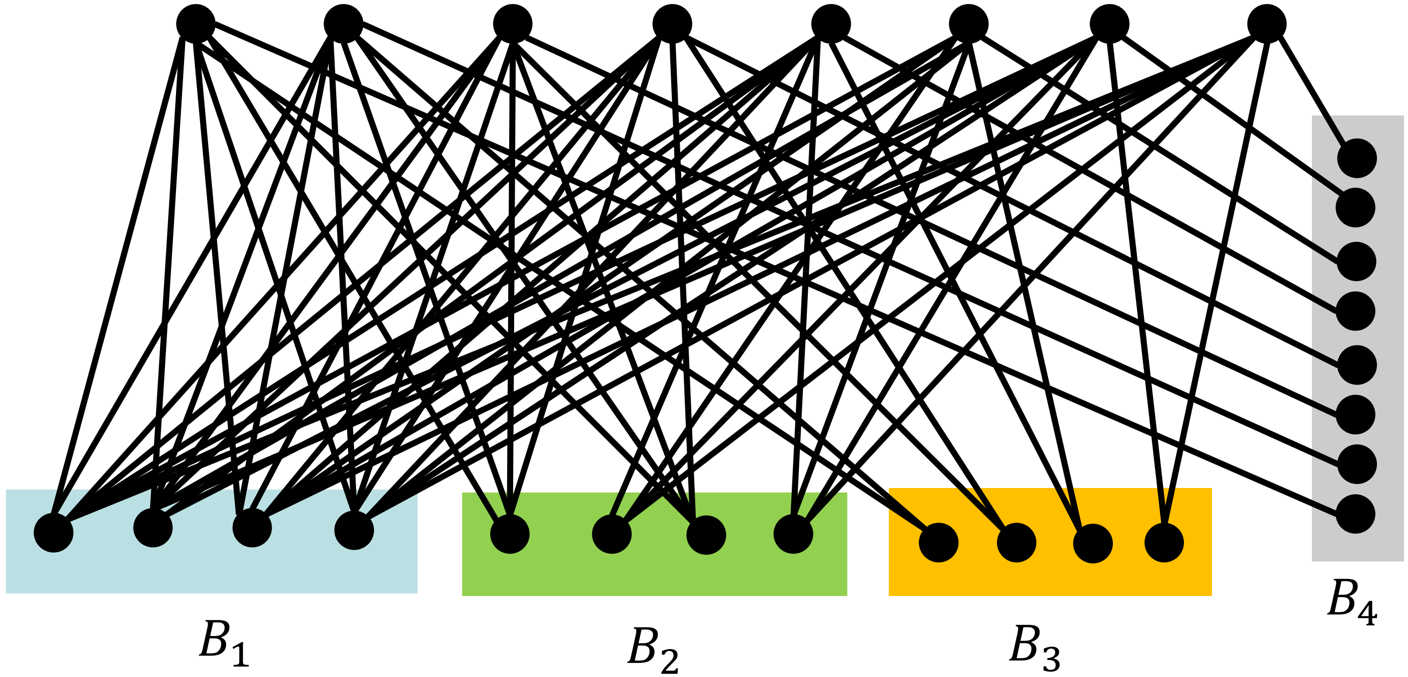
Greedy algorithms are typically used in practice to find a (good) solution to NP-hard problems

Strategy (1): Iteratively, include a vertex that covers most new edges

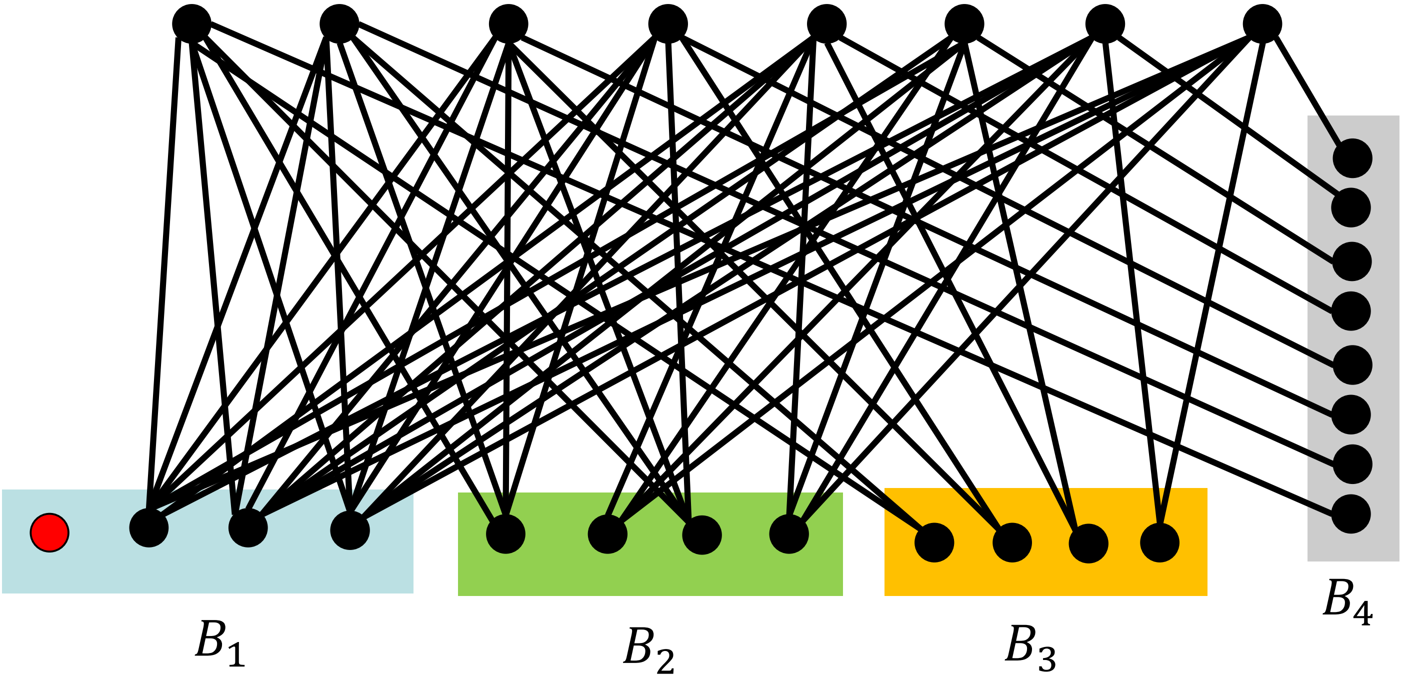
Q:Does this give an optimum solution?

A: No,

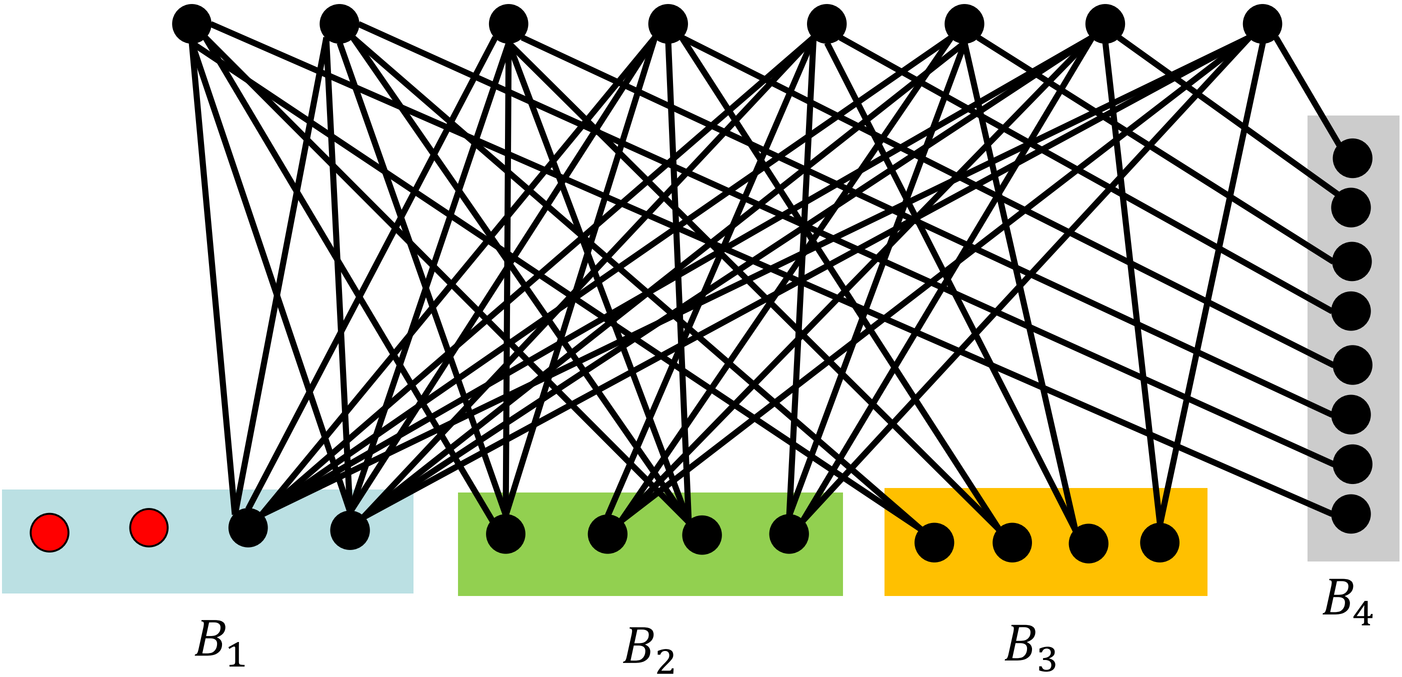
Greedy (1): Pick vertex that covers the most



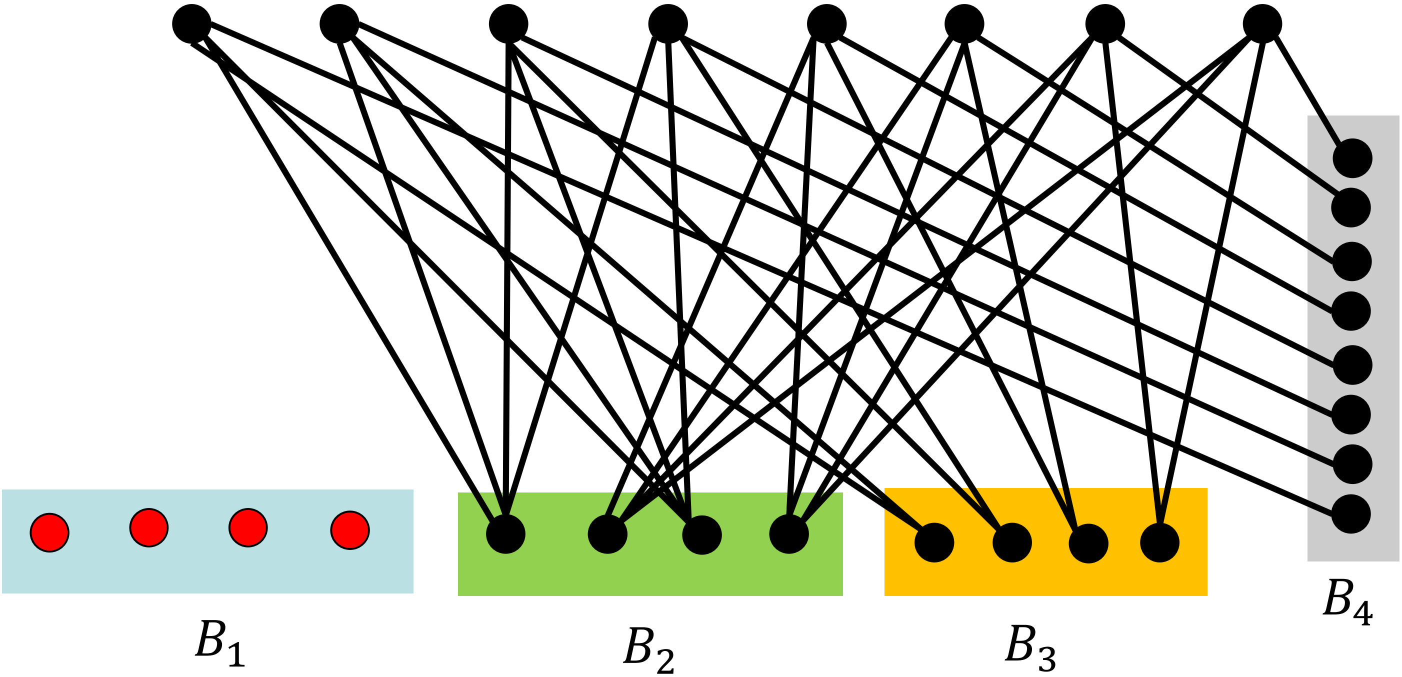
Greedy (1): Pick vertex that covers the most



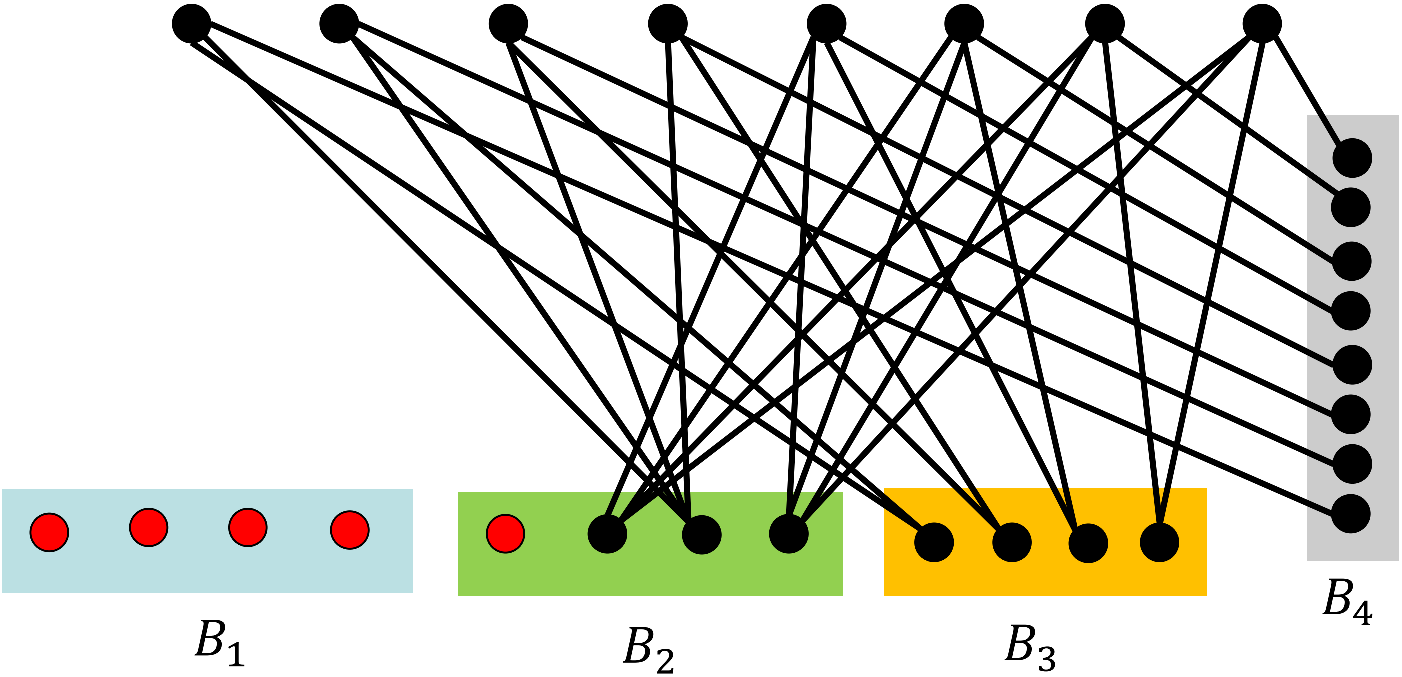
Greedy (1): Pick vertex that covers the most



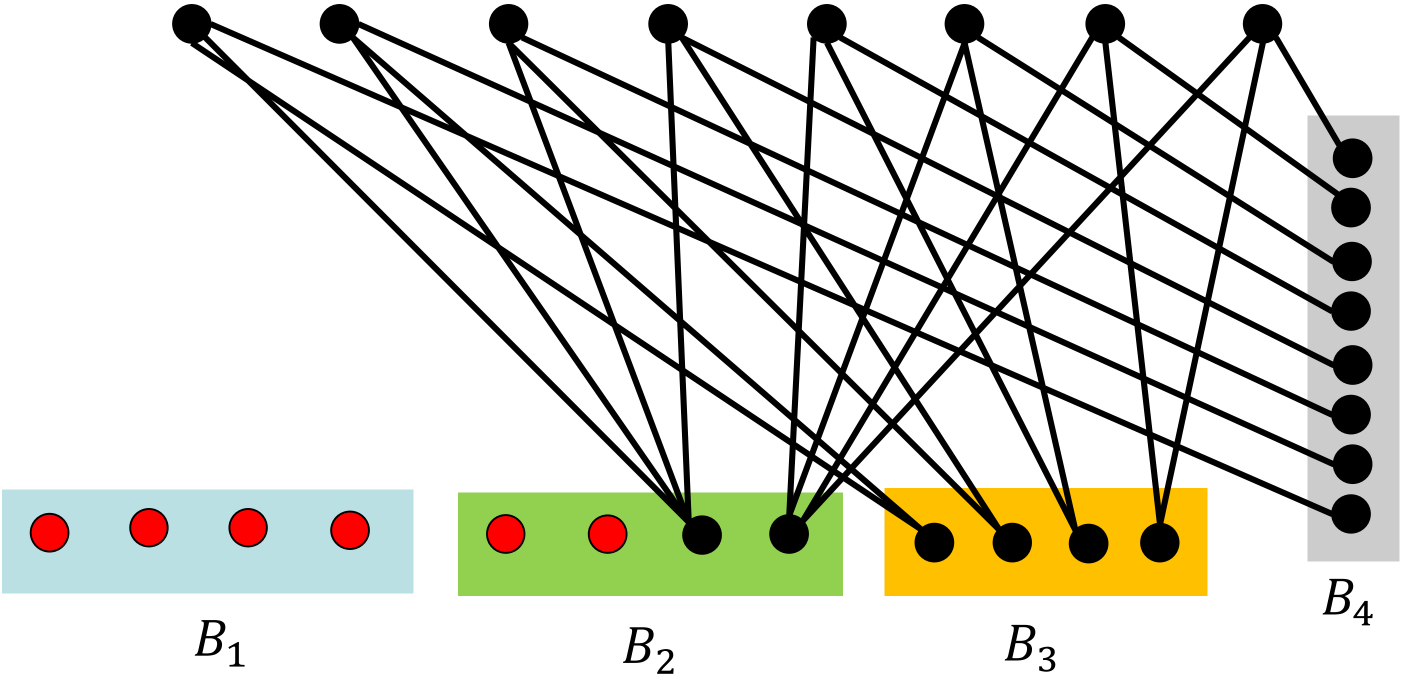
Greedy (1): Pick vertex that covers the most



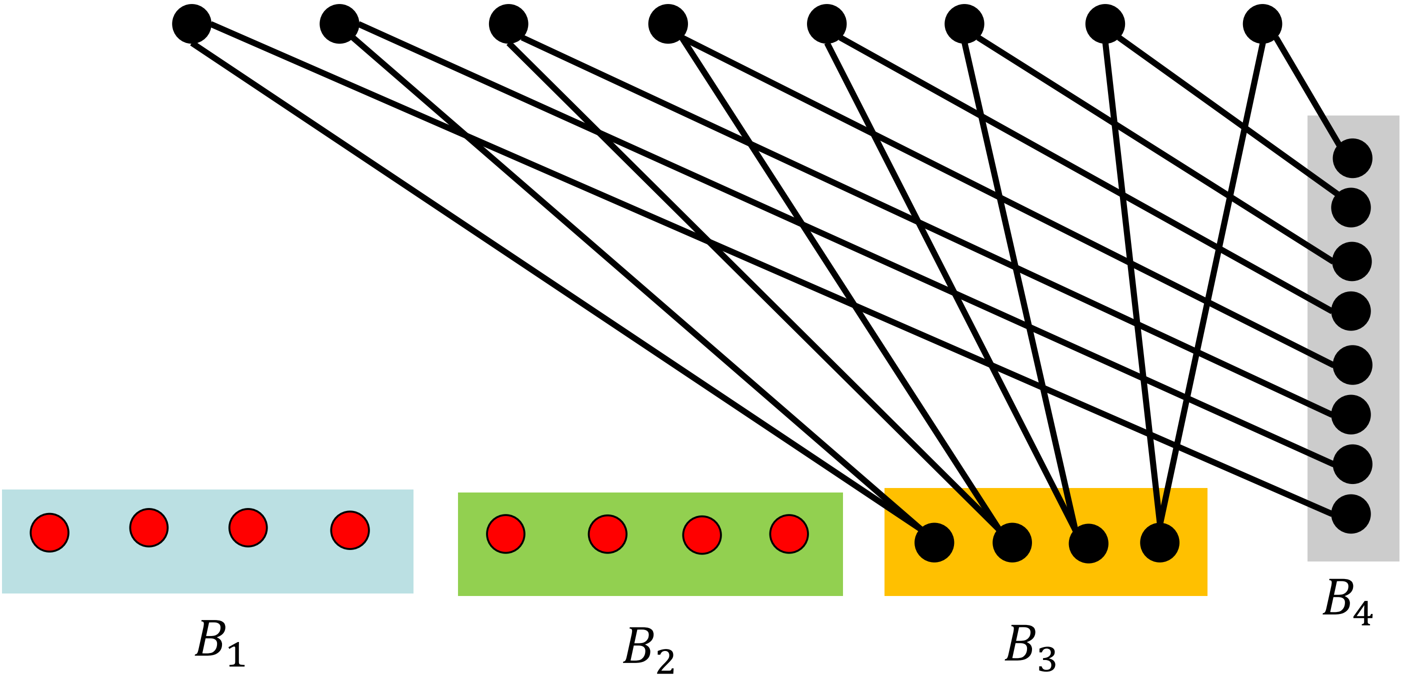
Greedy (1): Pick vertex that covers the most



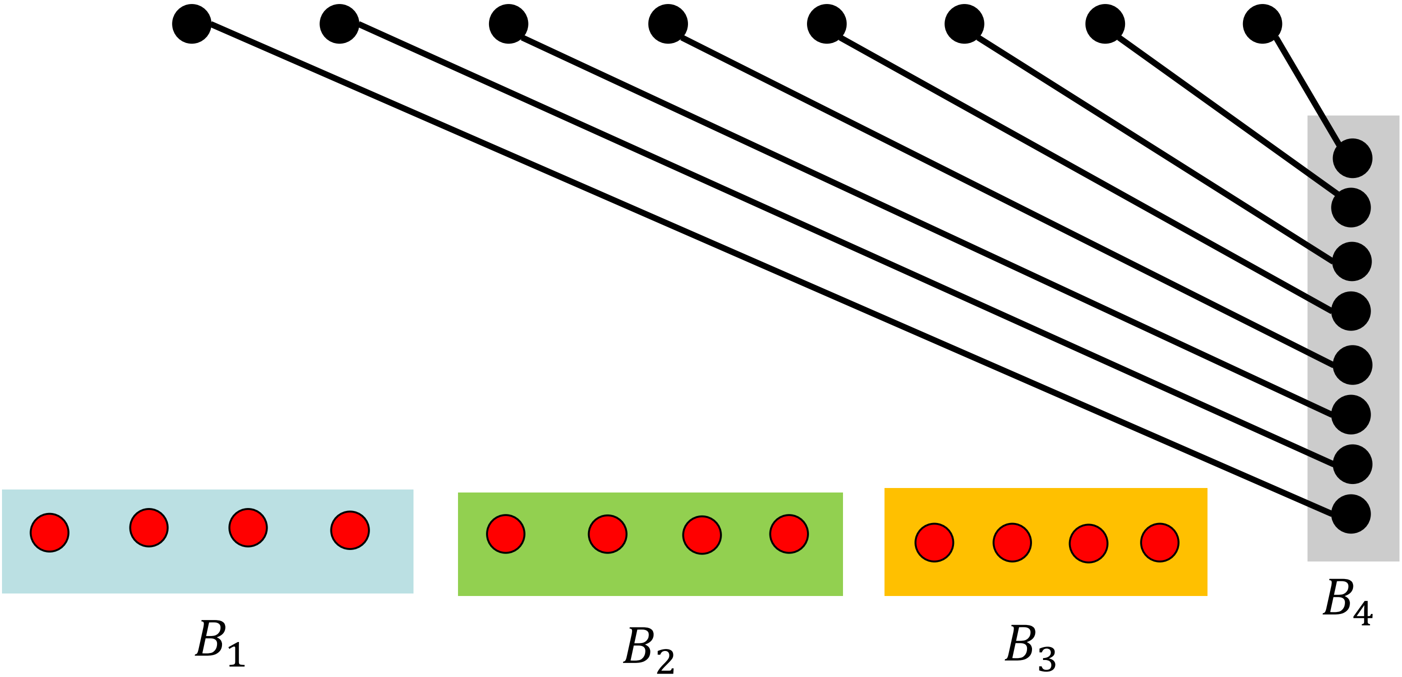
Greedy (1): Pick vertex that covers the most



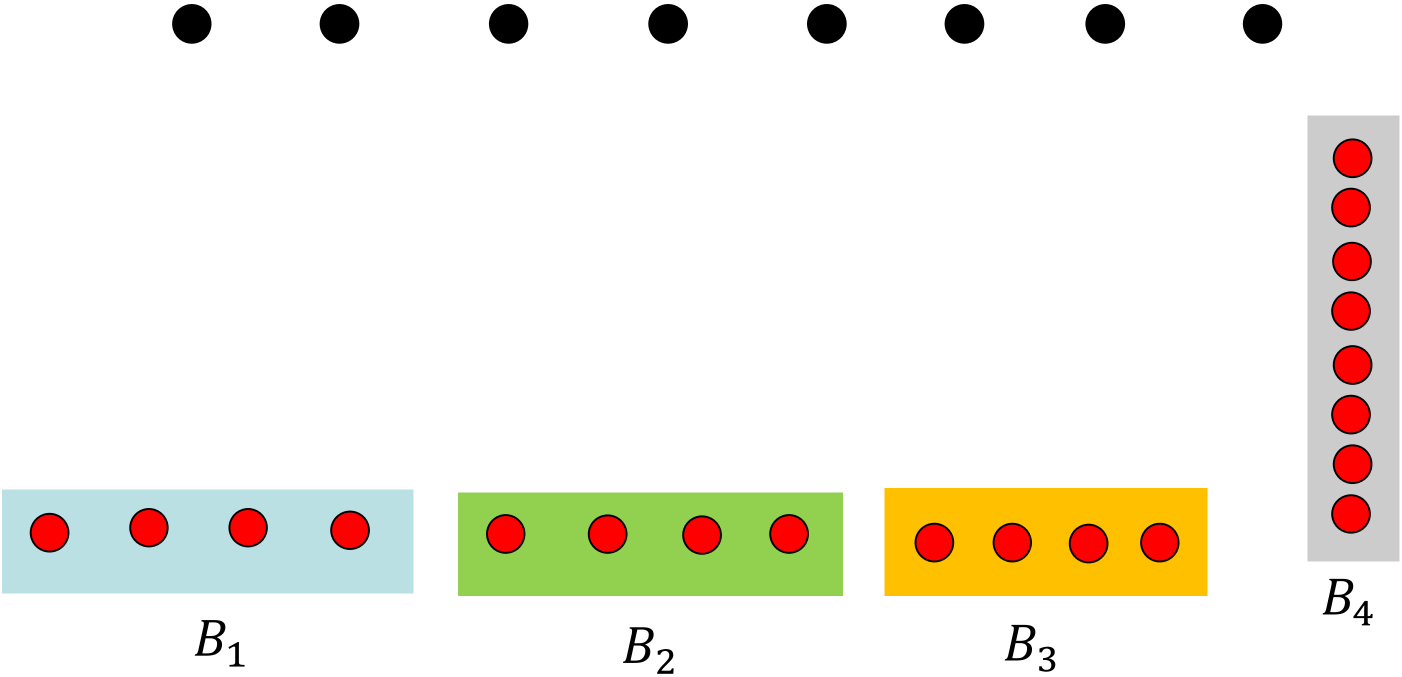
Greedy (1): Pick vertex that covers the most



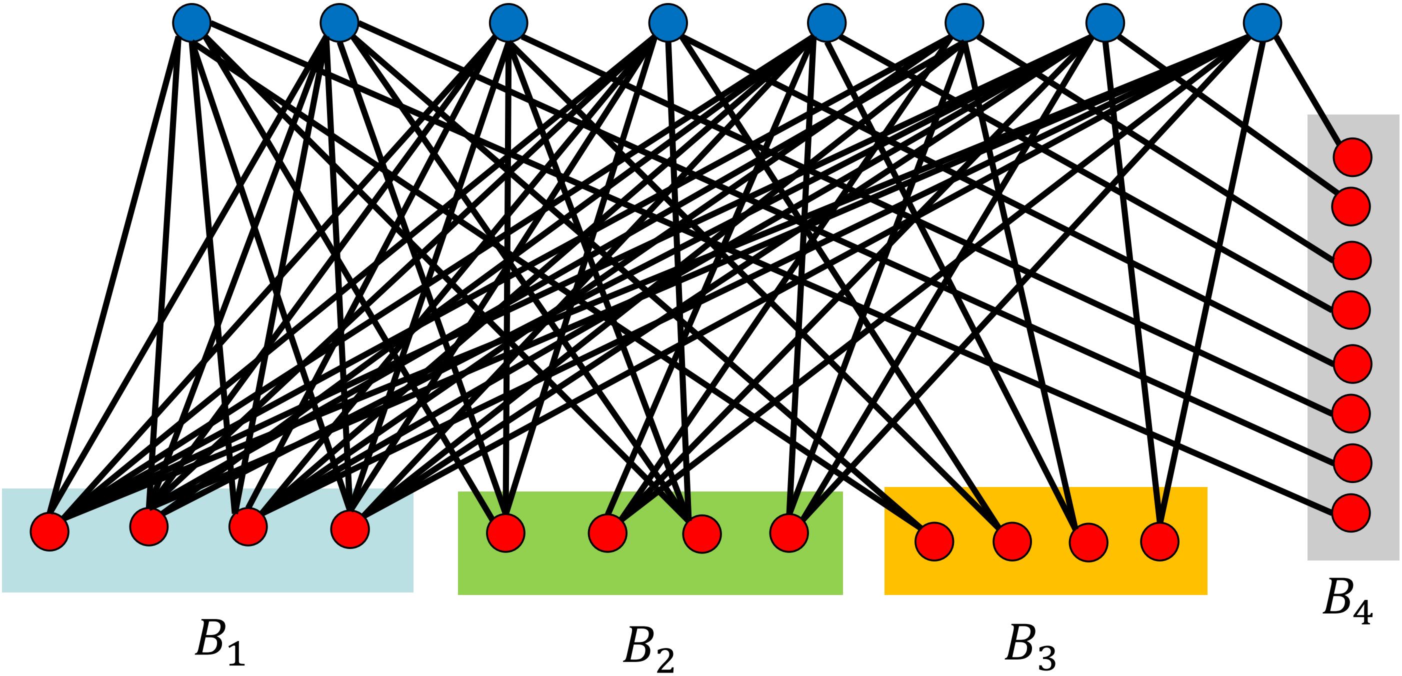
Greedy (1): Pick vertex that covers the most



Greedy (1): Pick vertex that covers the most



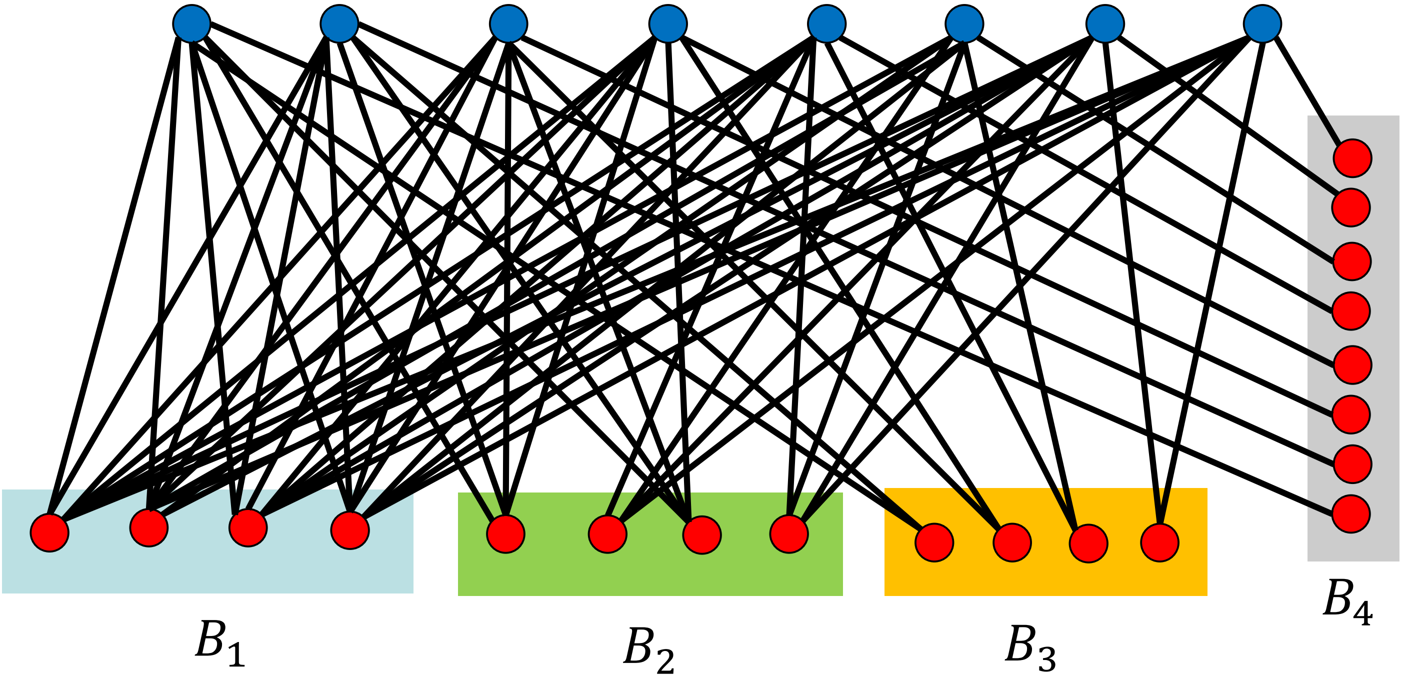
Greedy (1): Pick vertex that covers the most



Greedy Vertex cover = 20

OPT Vertex cover = 8

Greedy (1): Pick vertex that covers the most

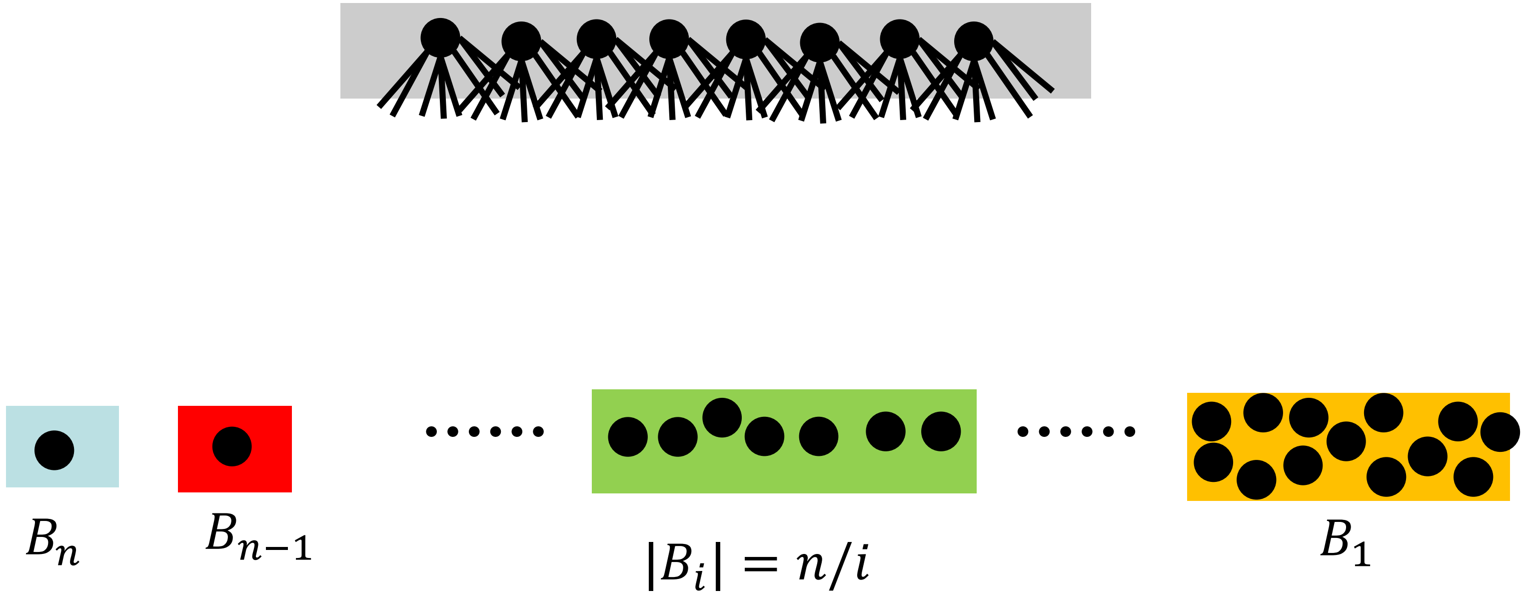


Greedy Vertex cover = 20

OPT Vertex cover = 8

Greedy (1): Pick vertex that covers the most

n vertices. Each vertex has one edge into each B_i



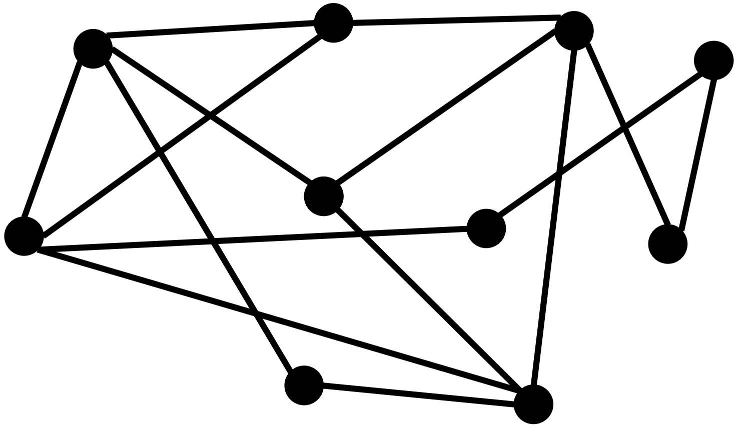
Greedy pick bottom vertices = $n + \frac{n}{2} + \frac{n}{3} + \cdots + 1 \approx n \ln n$

OPT pick top vertices = n

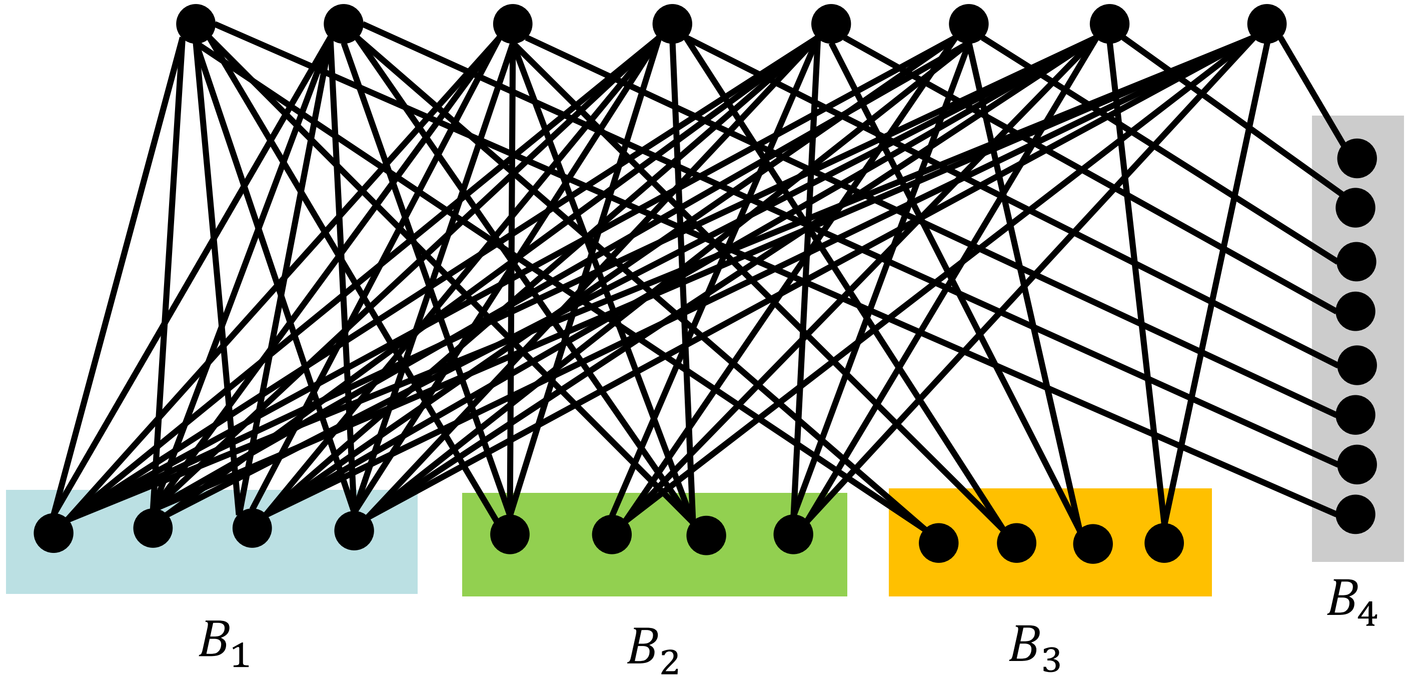
A Different Greedy Rule

Greedy 2: Iteratively, pick both endpoints of an uncovered edge.

Vertex cover = 6



Greedy 2: Pick Both endpoints of an uncovered edge



Greedy vertex cover = 16

OPT vertex cover = 8

Greedy (2) gives 2-approximation

Thm: Size of greedy (2) vertex cover is at most twice as big as size of optimal cover

Pf: Suppose Greedy (2) picks endpoints of edges $e_1, \ldots, e_k$. Since these edges do not touch, every valid cover must pick one vertex from each of these edges!

i.e., $OPT \geq k$.

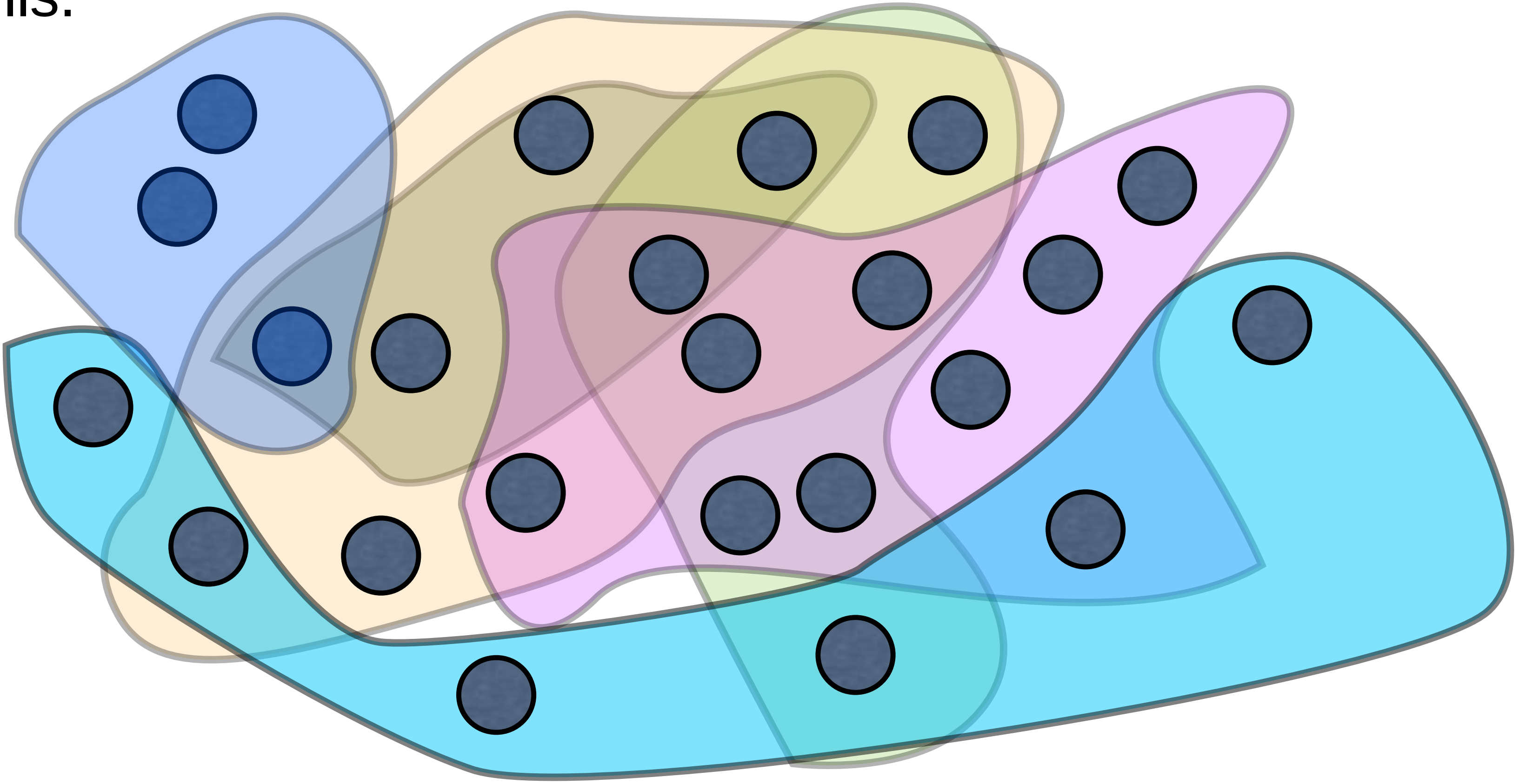
But the size of greedy cover is 2k. So, Greedy is a 2-approximation.

Set Cover

Given a number of sets on a ground set of elements,

Goal: choose minimum number of sets that cover all.

e.g., a company wants to hire employees with certain skills.

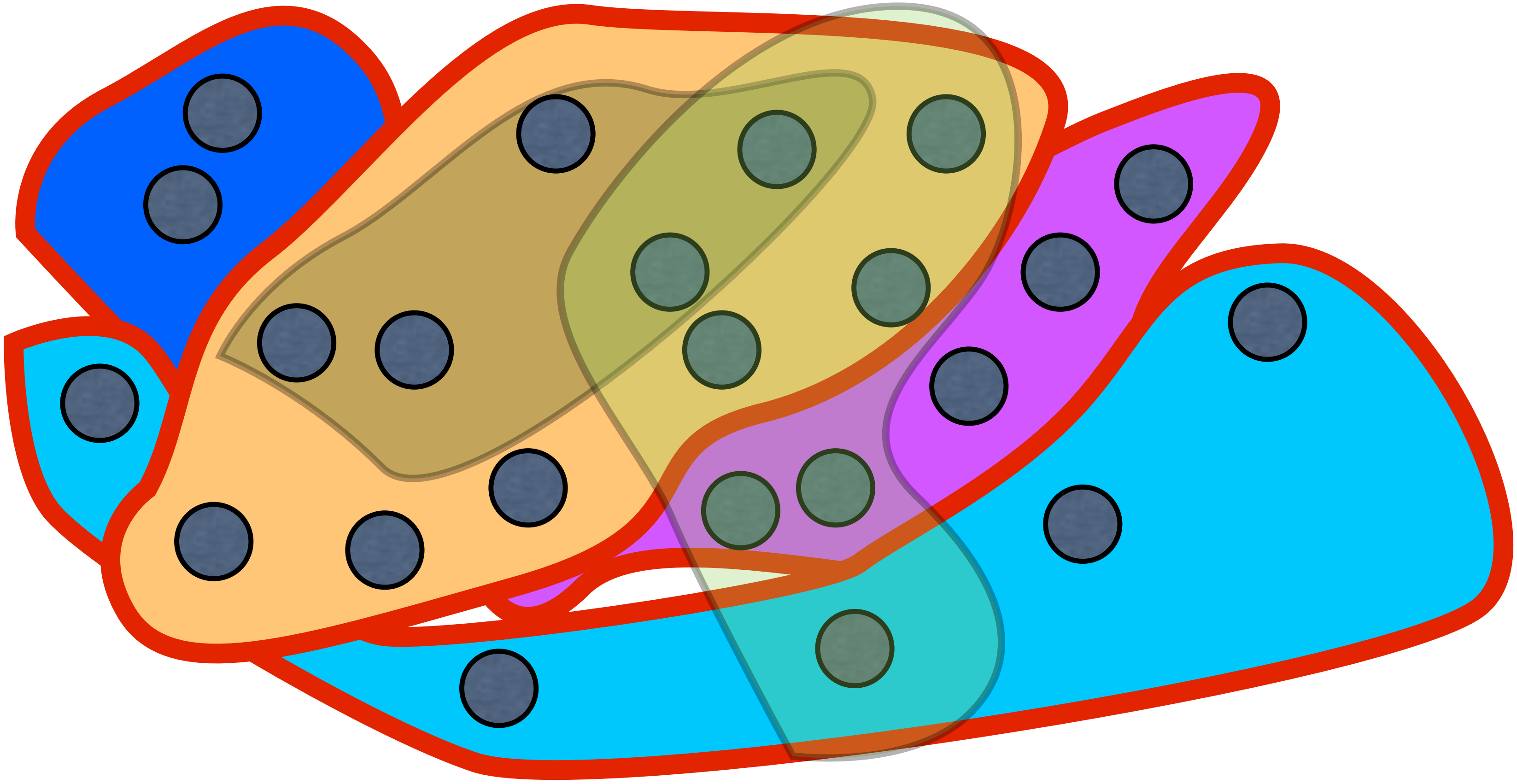


Set Cover

Given a number of sets on a ground set of elements,

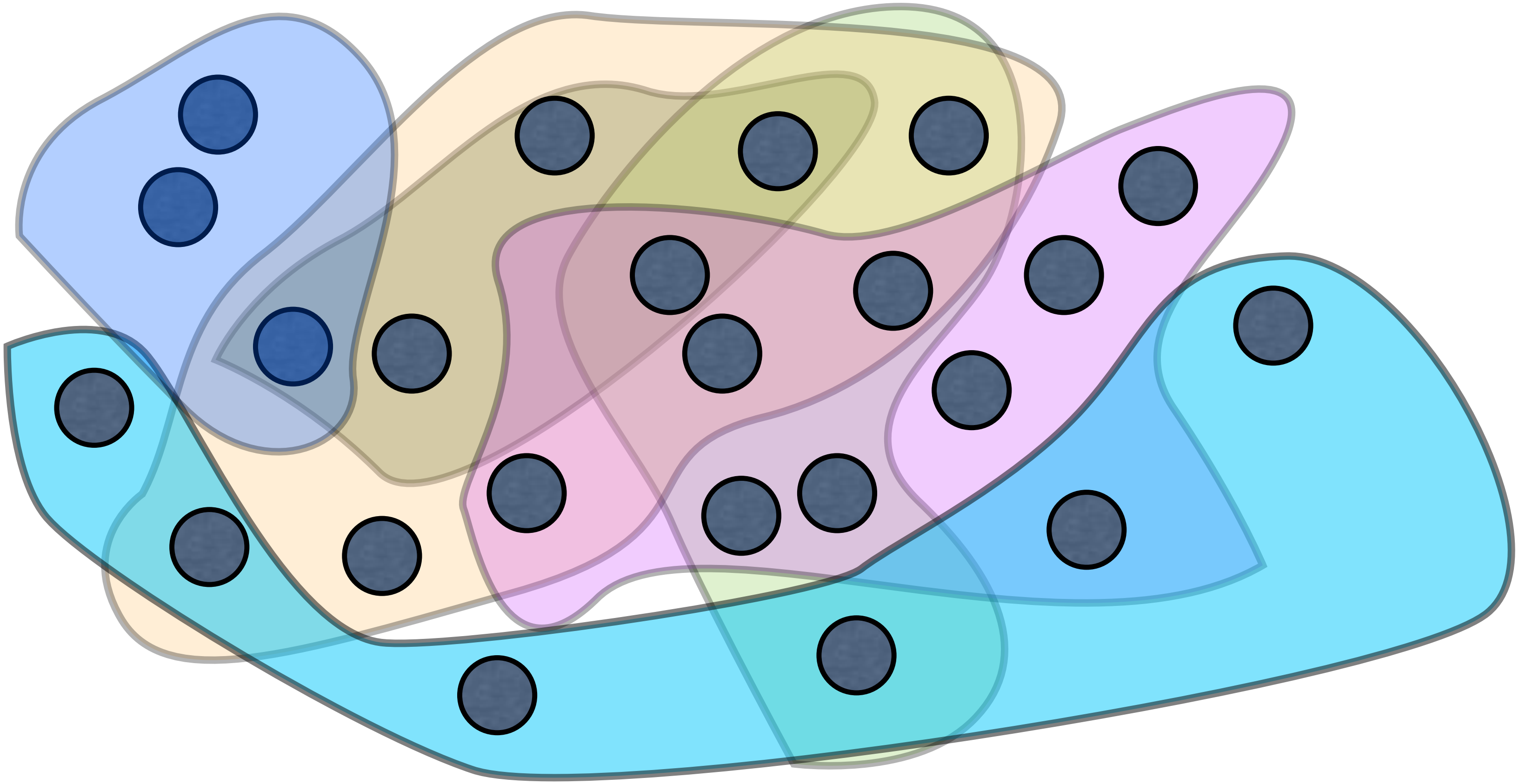
Goal: choose minimum number of sets that cover all.

Set cover = 4



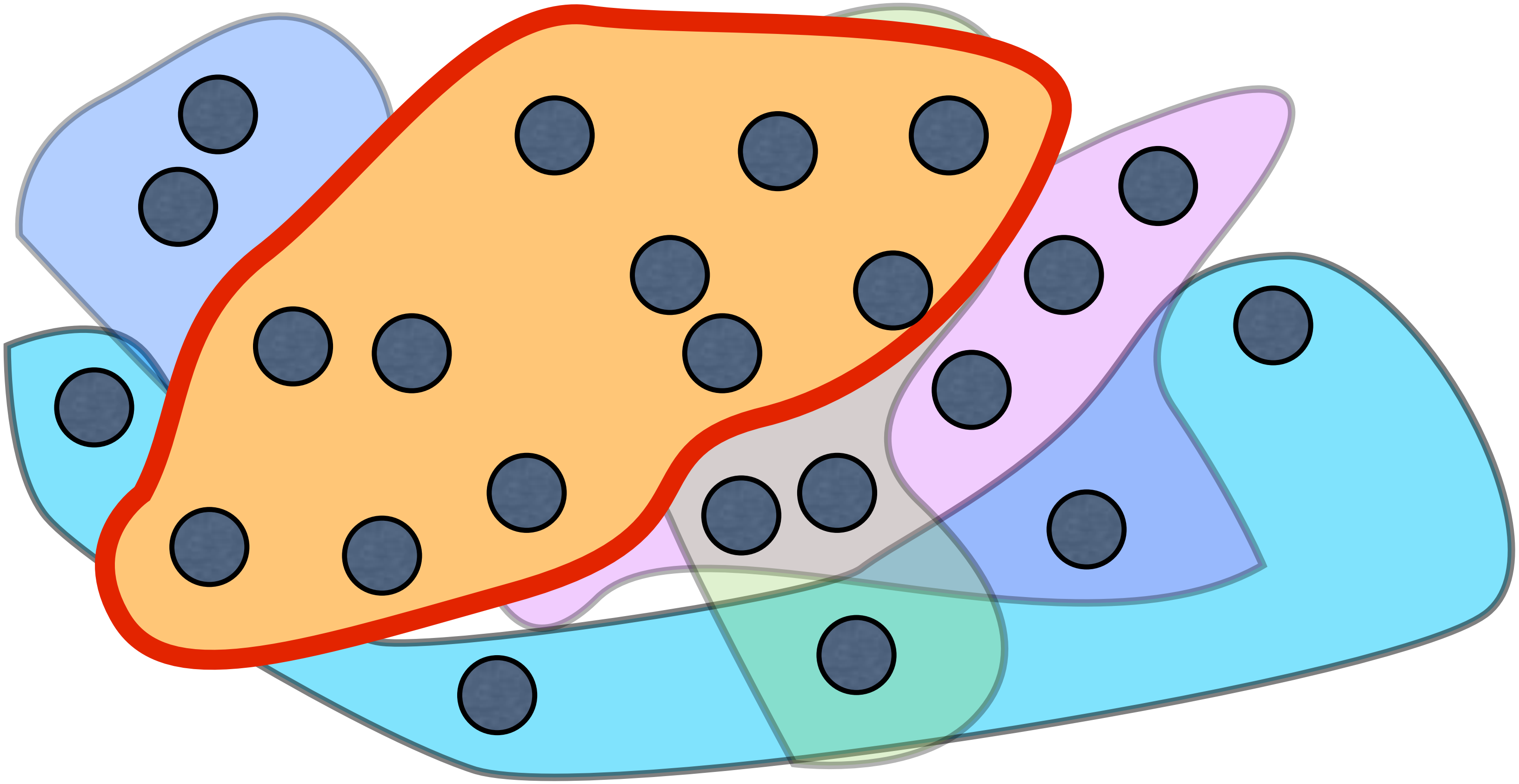
A Greedy Algorithm

Strategy: Pick the set that maximizes # new elements covered



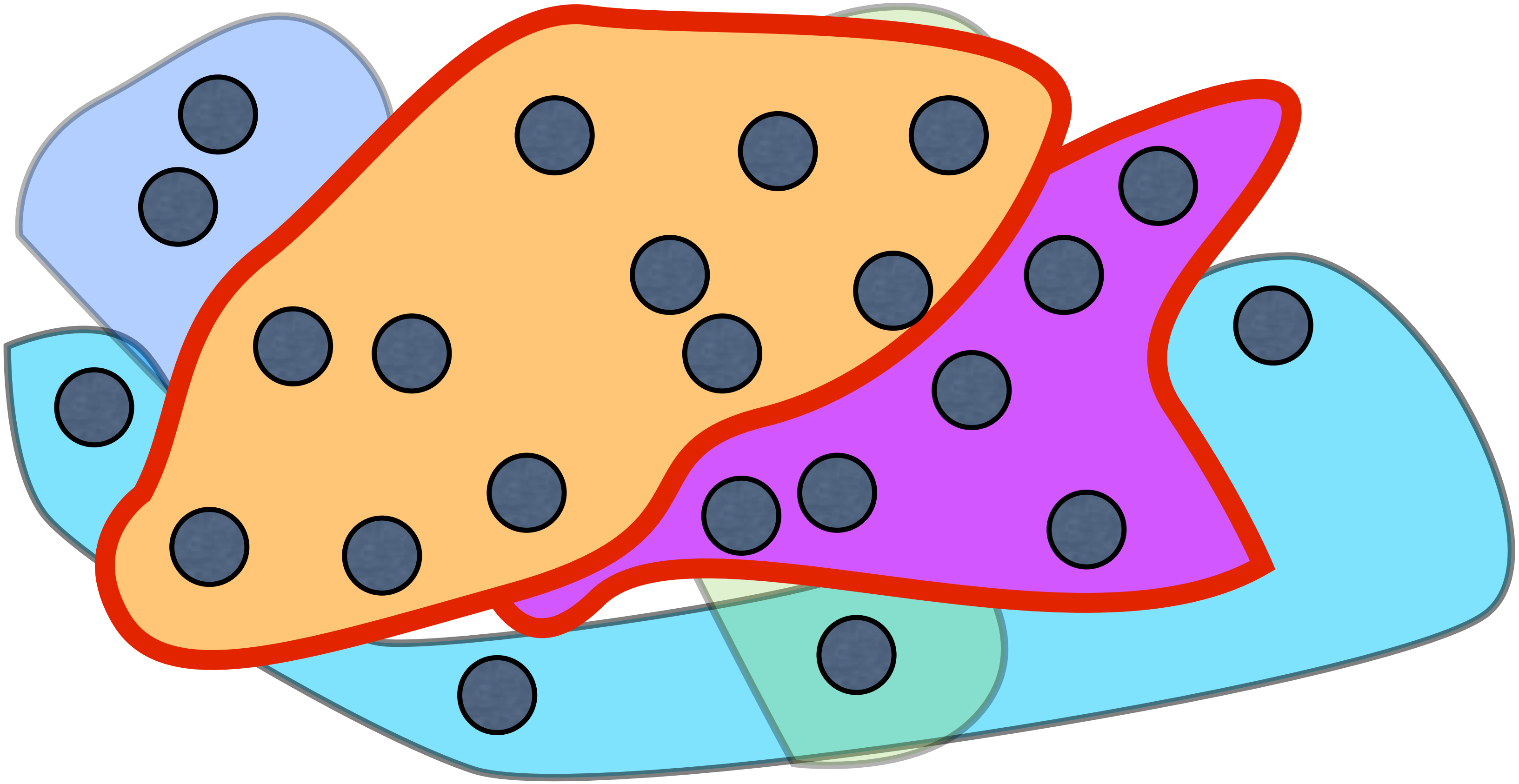
A Greedy Algorithm

Strategy: Pick the set that maximizes # new elements covered



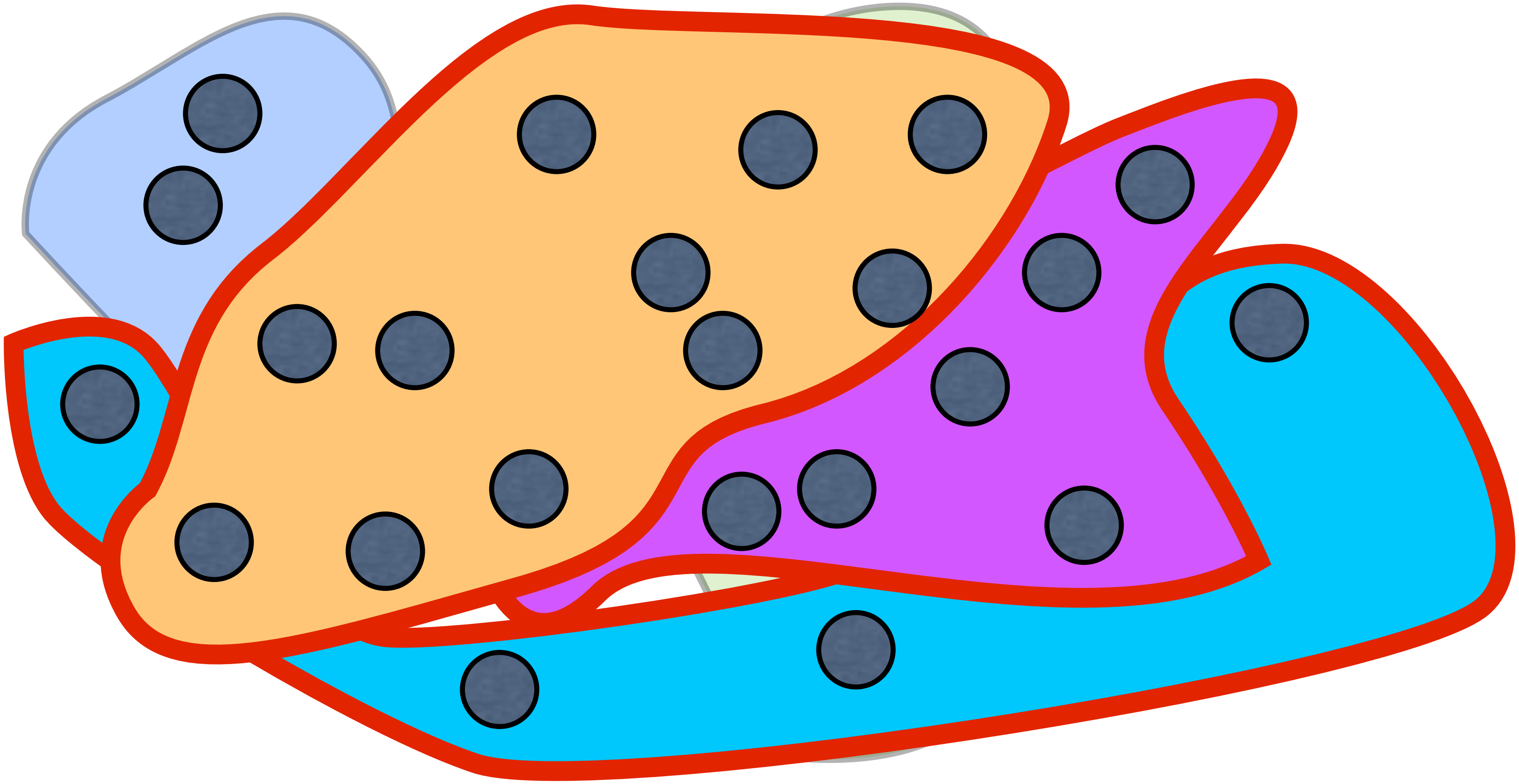
A Greedy Algorithm

Strategy: Pick the set that maximizes # new elements covered



A Greedy Algorithm

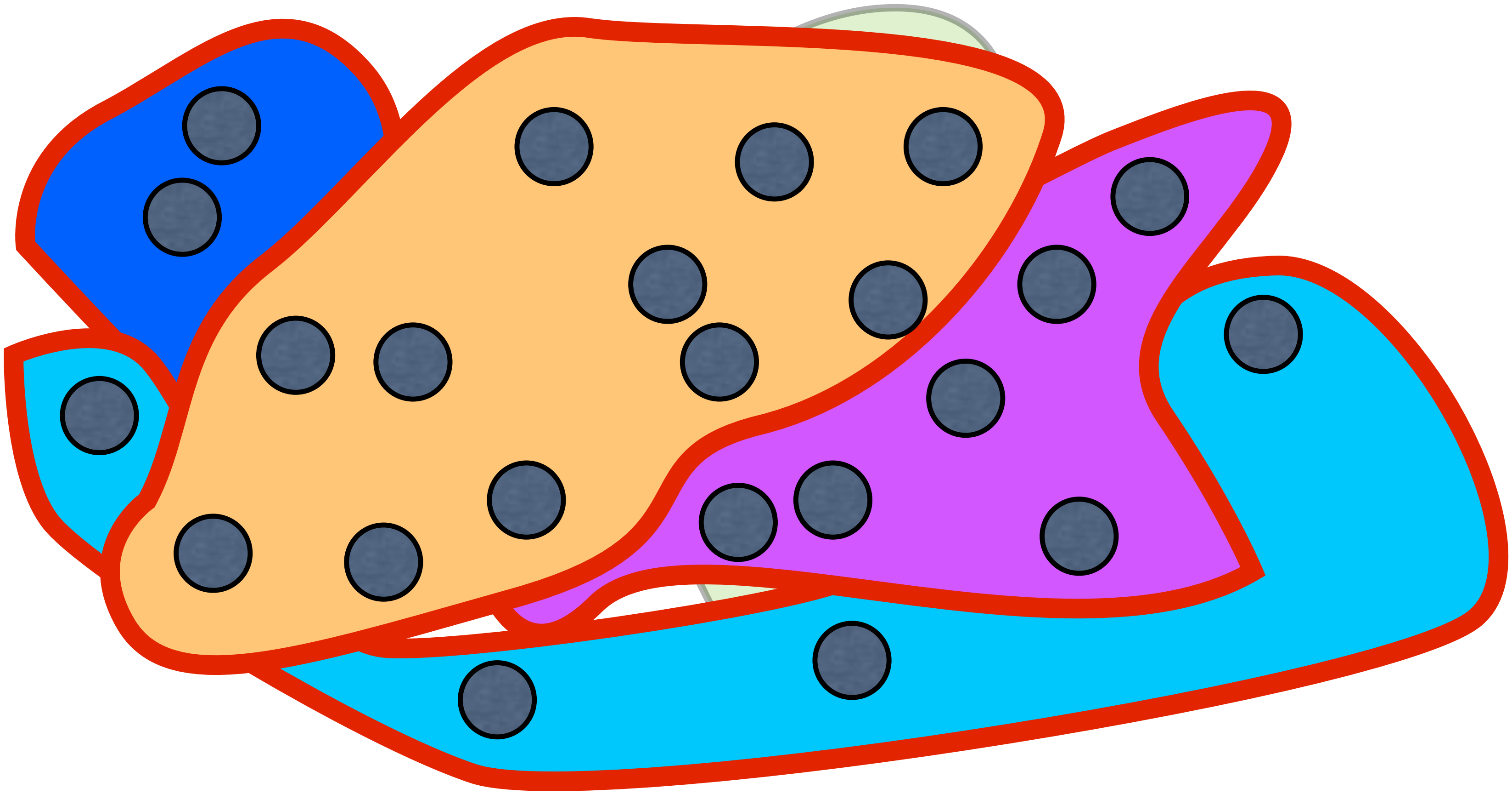
Strategy: Pick the set that maximizes # new elements covered



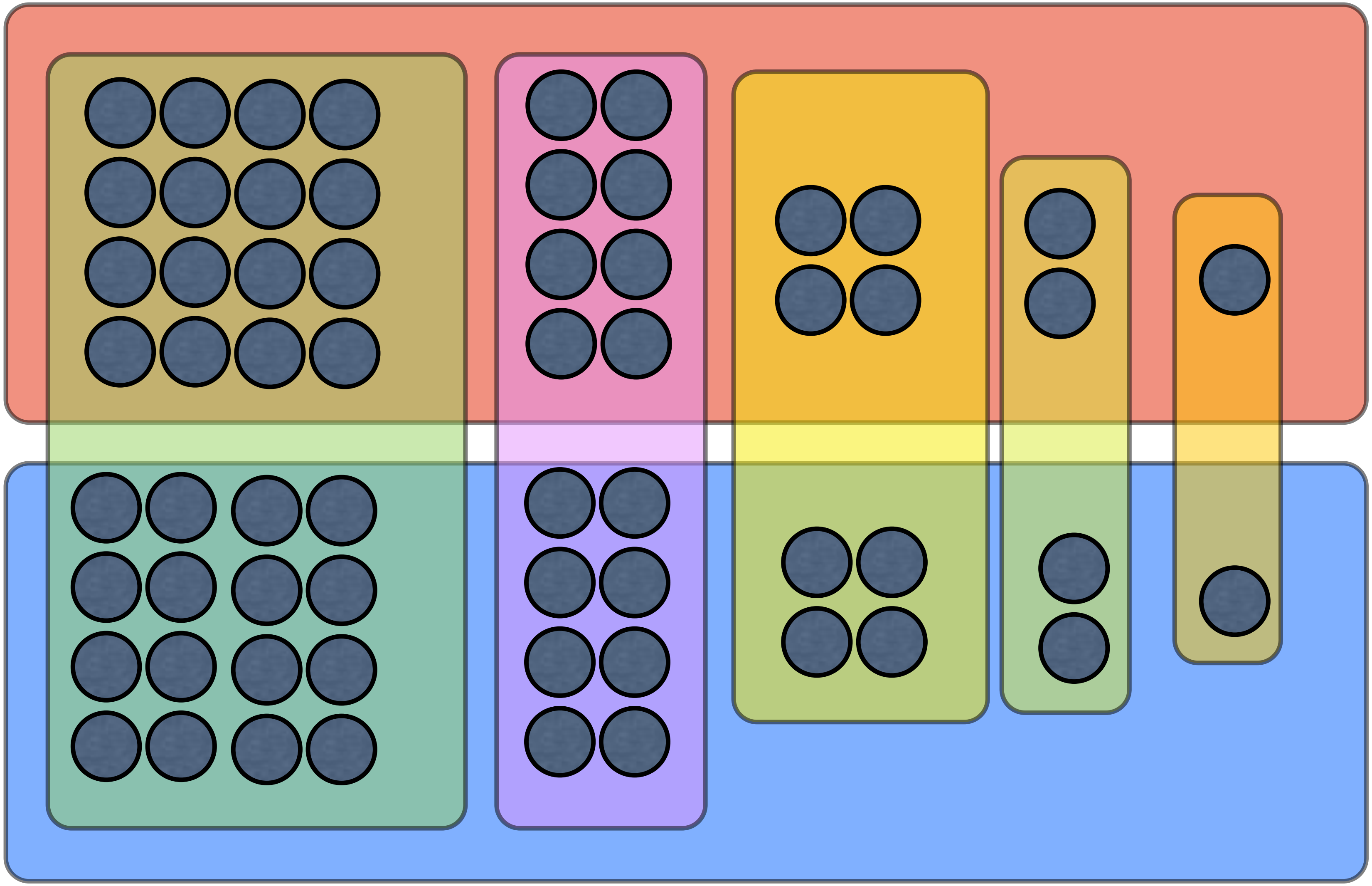
A Greedy Algorithm

Strategy: Pick the set that maximizes # new elements covered

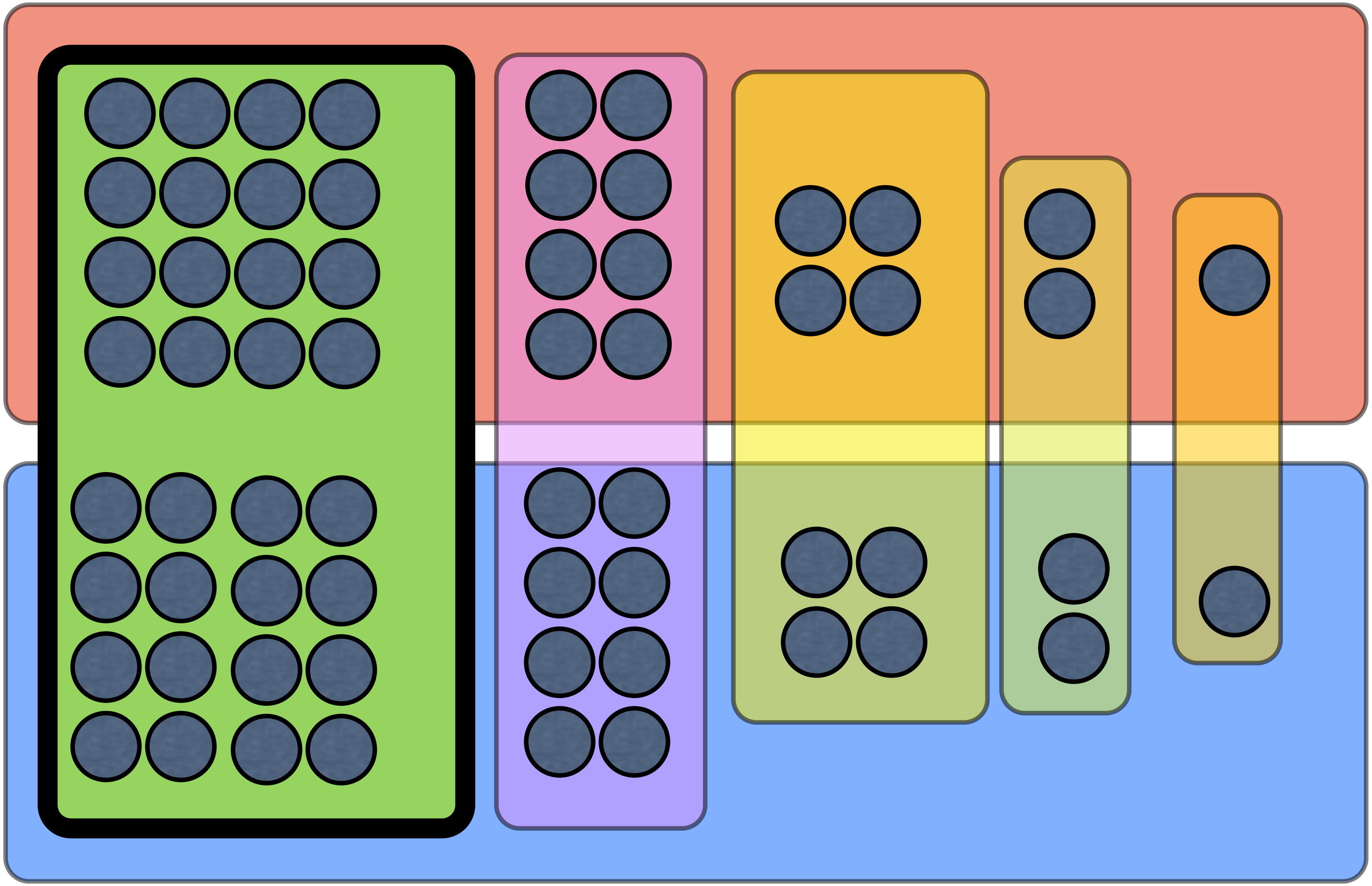
Thm: Greedy has ln n approximation ratio



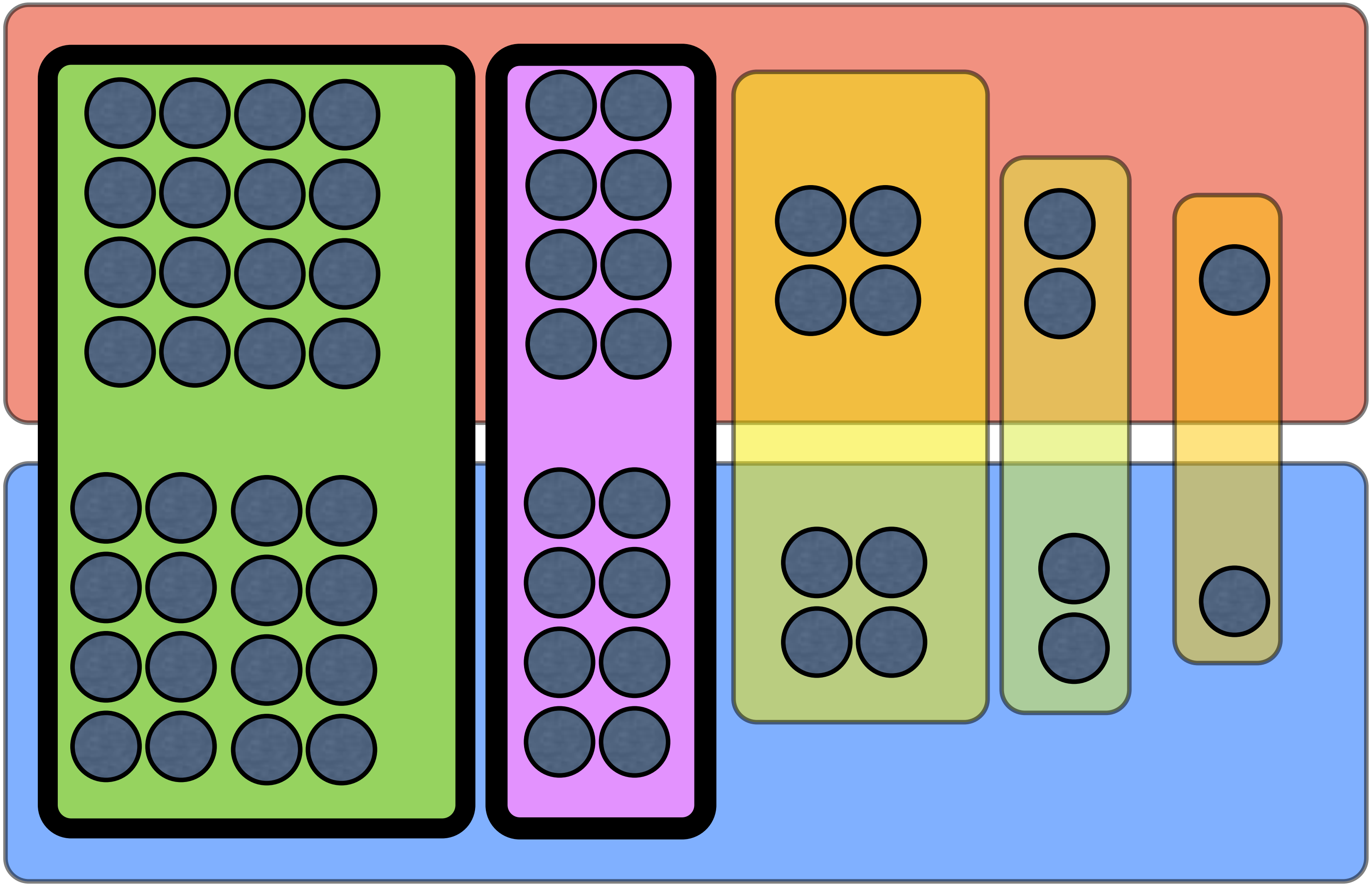
A Tight Example for Greedy



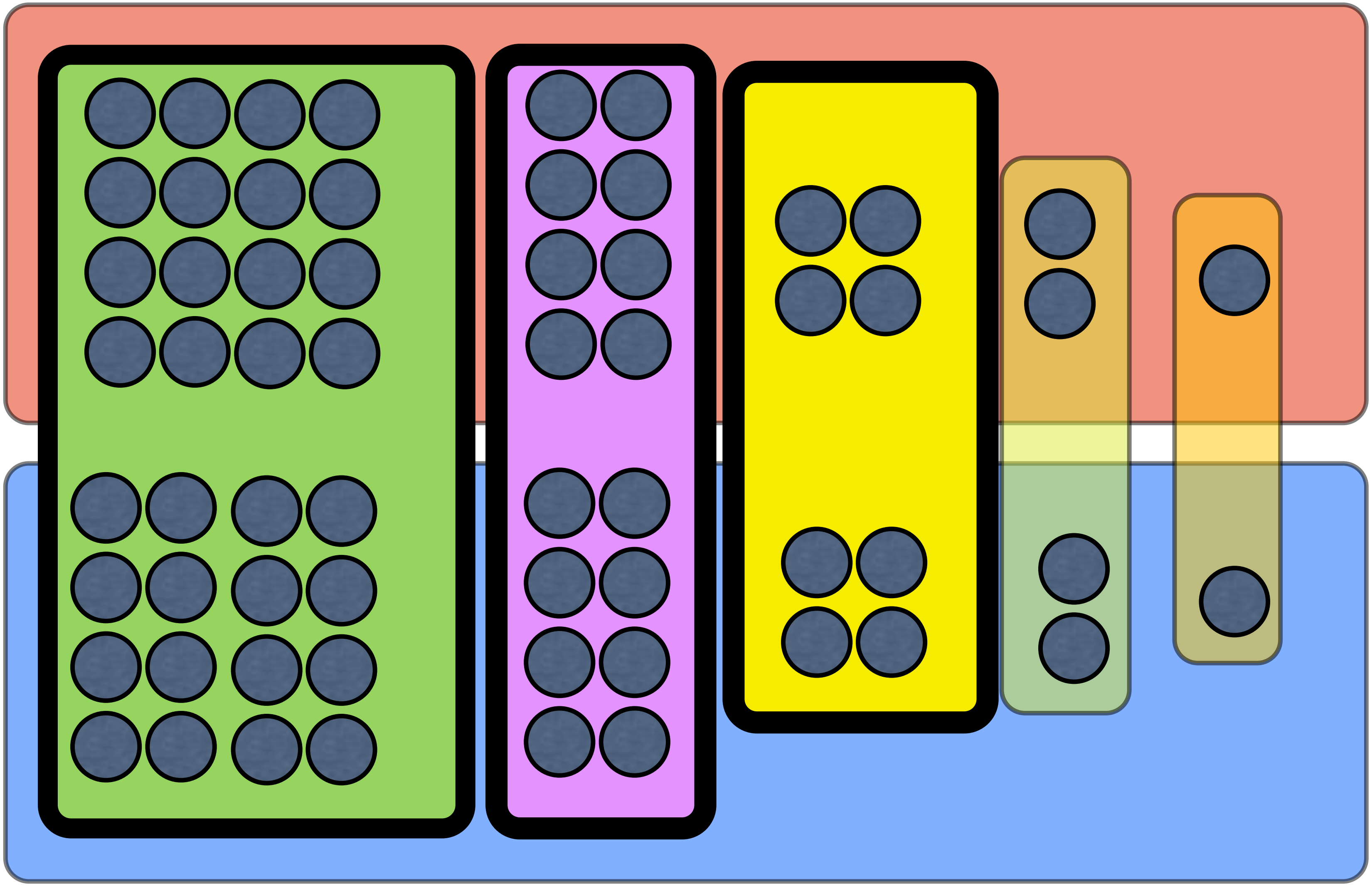
A Tight Example for Greedy



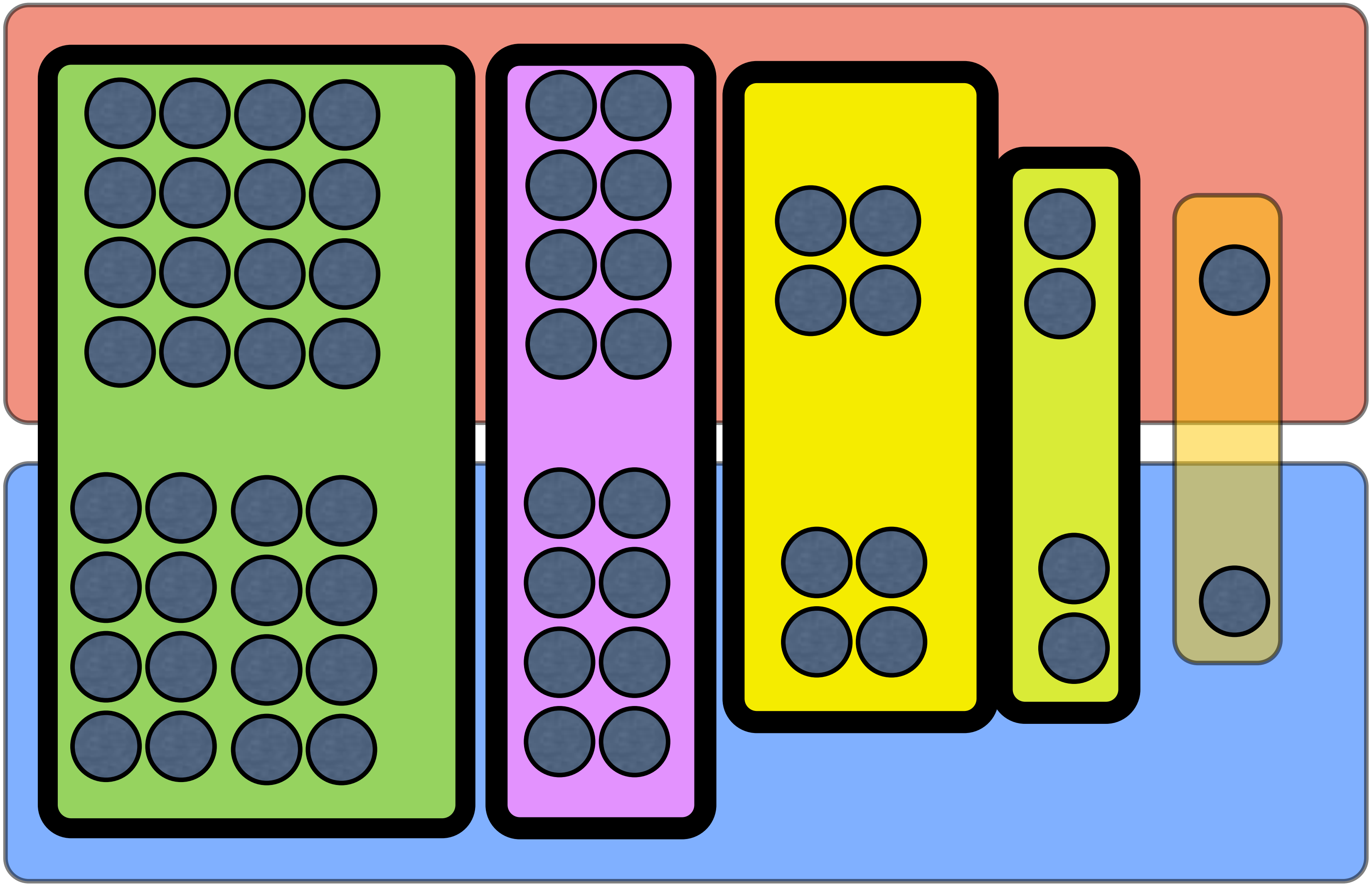
A Tight Example for Greedy



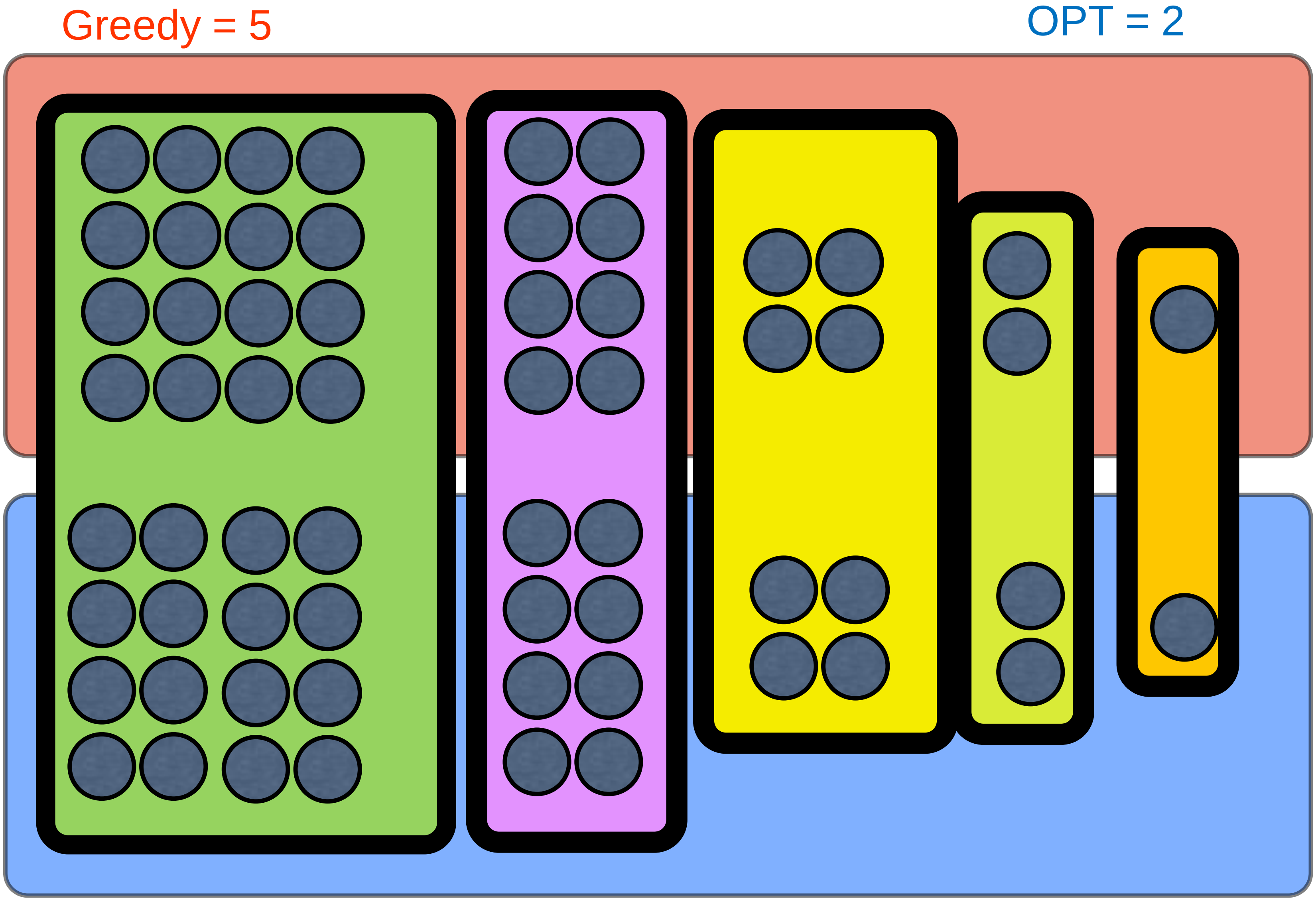
A Tight Example for Greedy



A Tight Example for Greedy



A Tight Example for Greedy



Greedy Gives O(log(n)) approximation

Thm: If the best solution has k sets, greedy finds at most k ln(n) sets.

Pf: Suppose OPT=k

There is set that covers 1/k fraction of remaining elements, since there are k sets that cover all remaining elements.

So in each step, algorithm will cover 1/k fraction of remaining elements.

#elements uncovered after t steps

$$\leq n\left(1-\frac{1}{k}\right)^{t} \leq n e^{-\frac{t}{k}}$$

So after $t = k \ln n$ steps, # uncovered elements < 1.

Approximation Algorithm Summary

- The best known approximation algorithm for set cover is the greedy.
  - It is NP-Complete to obtain better than ln(n) approximation ratio for set cover.
- The best known approximation algorithm for vertex cover is the greedy.
  - It has been open for 40 years to obtain a polynomial time algorithm with approximation ratio better than 2
- There is a long list of questions we do not know the best approximation algorithm.
- https://en.wikipedia.org/wiki/Unique_games_conjecture