Introduction to Algorithms

Approximation Algorithm

Approximation Algorithms

How to deal with NP-complete Problem

Many of the important problems in real world are NP-complete.

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

So, we cannot find optimum solutions in polynomial time. We have to enumerate all possible solutions and identify the best one, which sometimes takes n! steps, which is in $O(2^n)$ 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

Polynomial-time Algorithms with a guaranteed approximation ratio.

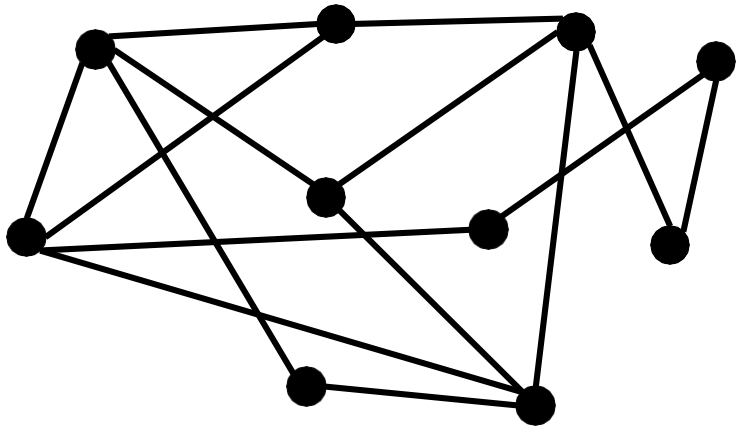
$$\alpha = \frac{\text{Cost of computed solution}}{\text{Cost of the optimum}}$$

worst case over all instances.

Goal: For each NP-hard problem find an 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?

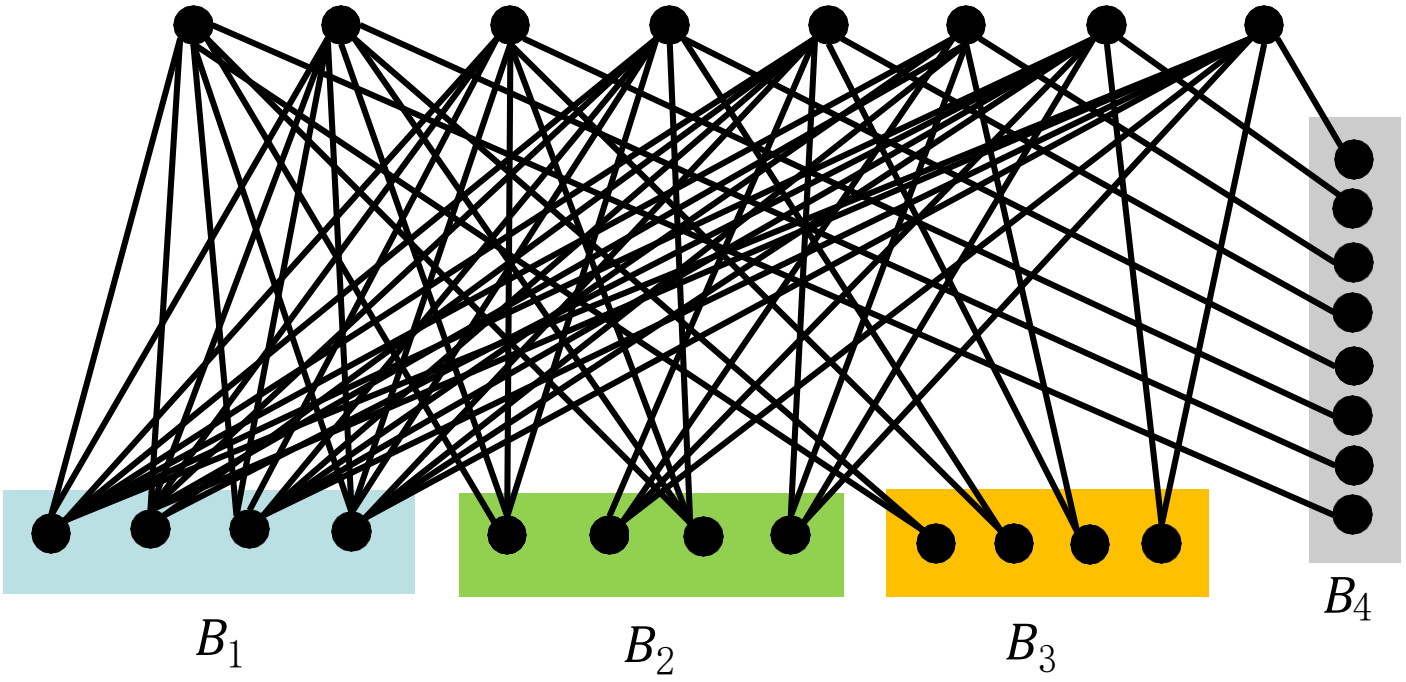
Greedy algorithms are typically used in practice to find a (good) solution to NP-hard problems

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

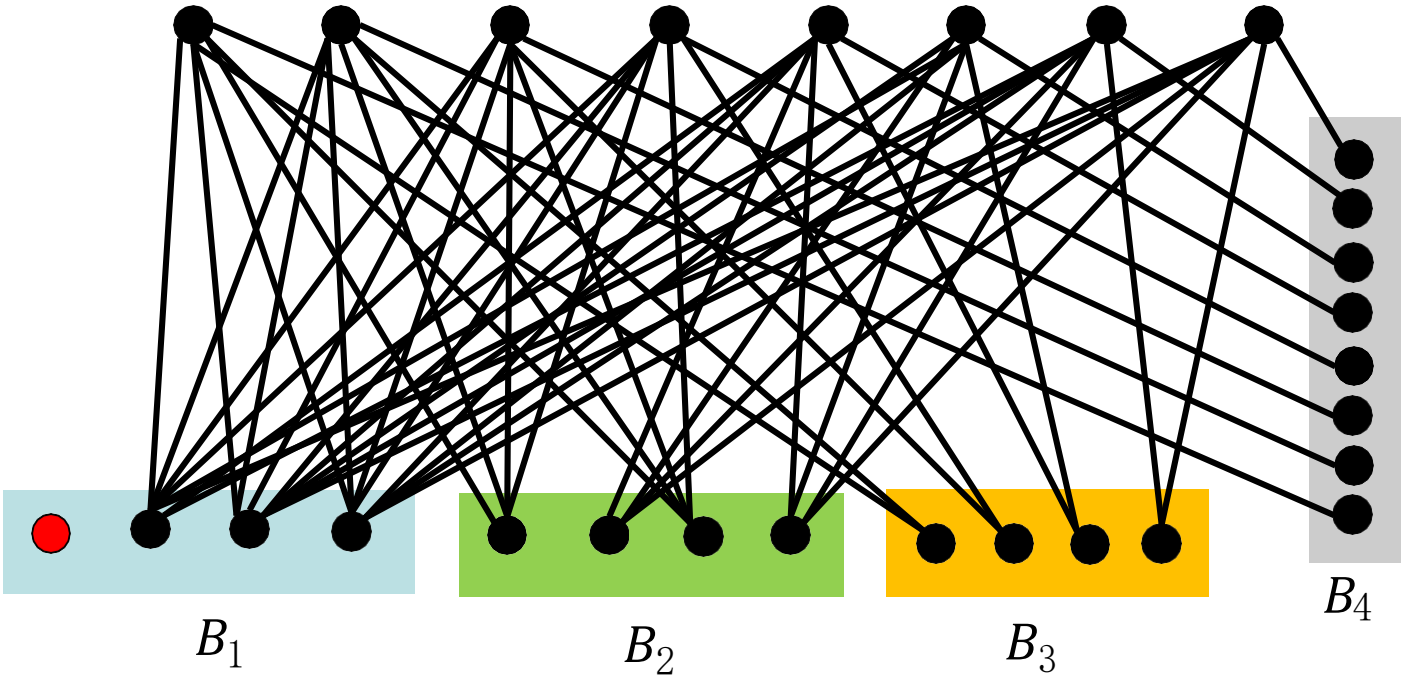
Q:Does this give an optimum solution?

A: No,

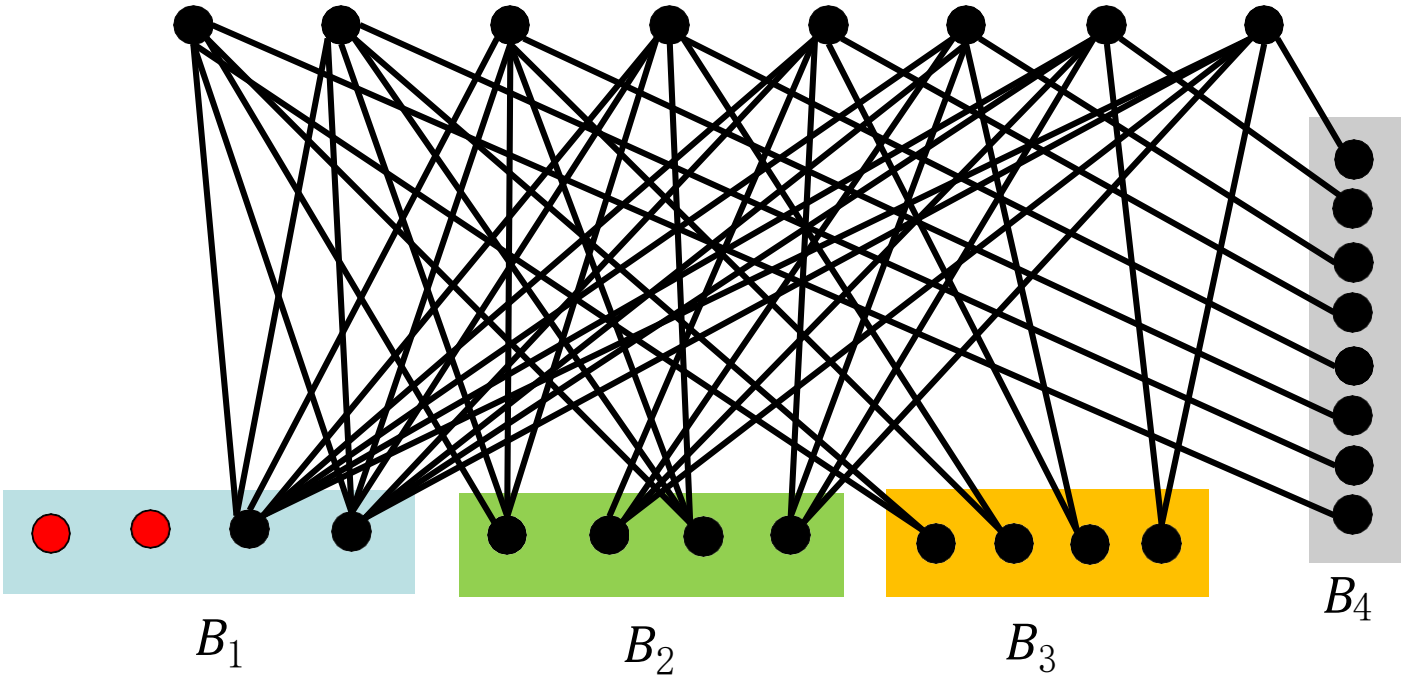
Greedy (1): Pick vertex that covers the most



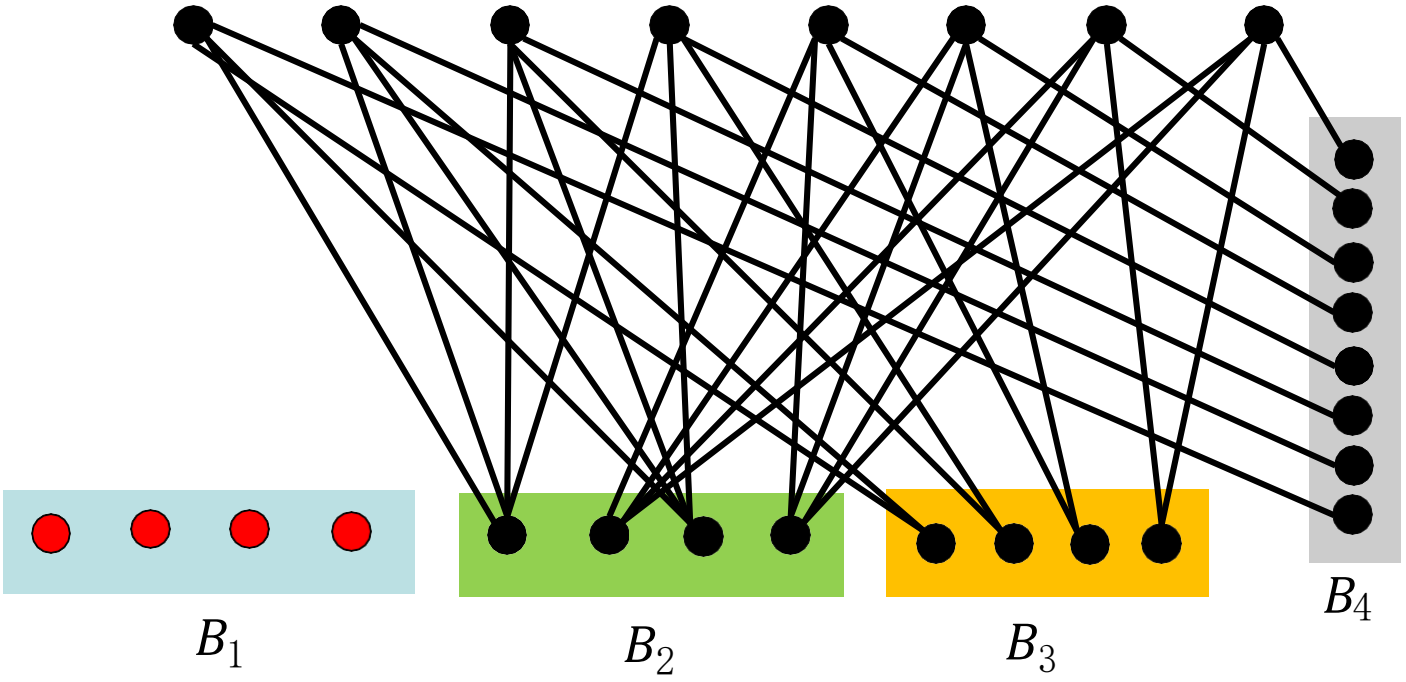
Greedy (1): Pick vertex that covers the most



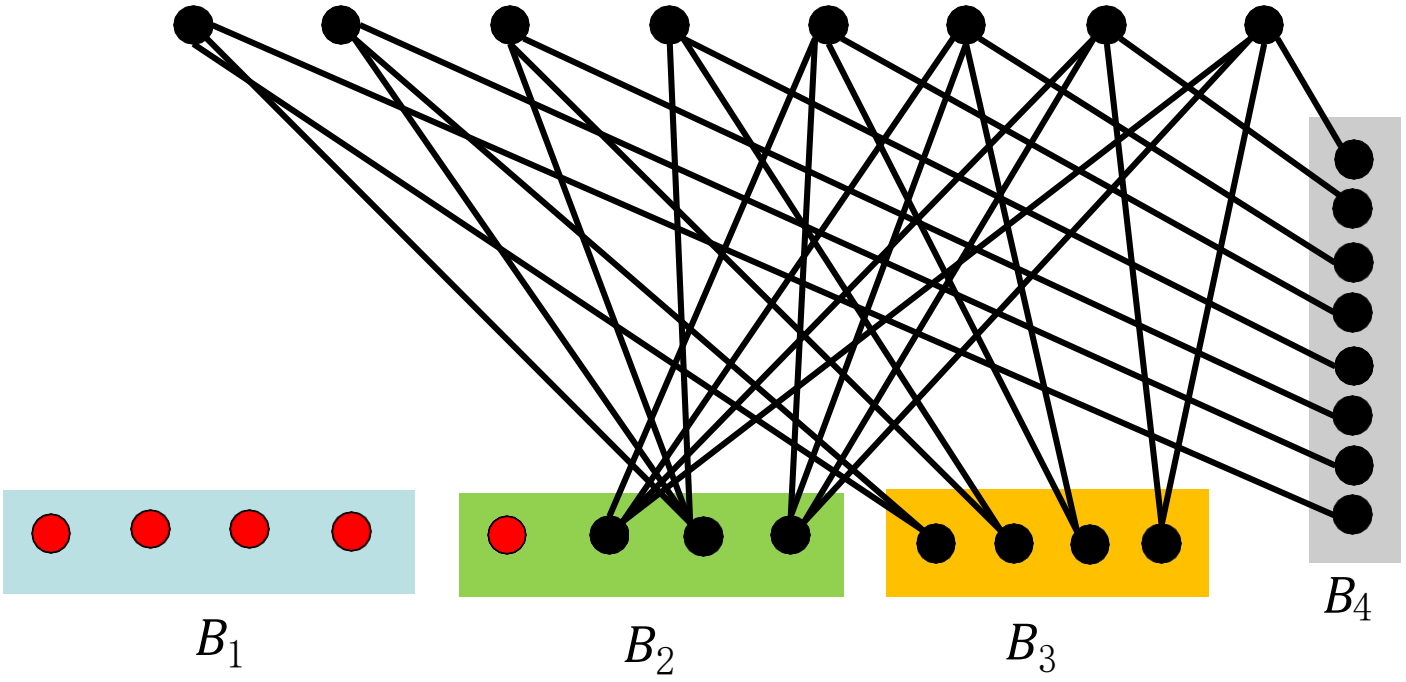
Greedy (1): Pick vertex that covers the most



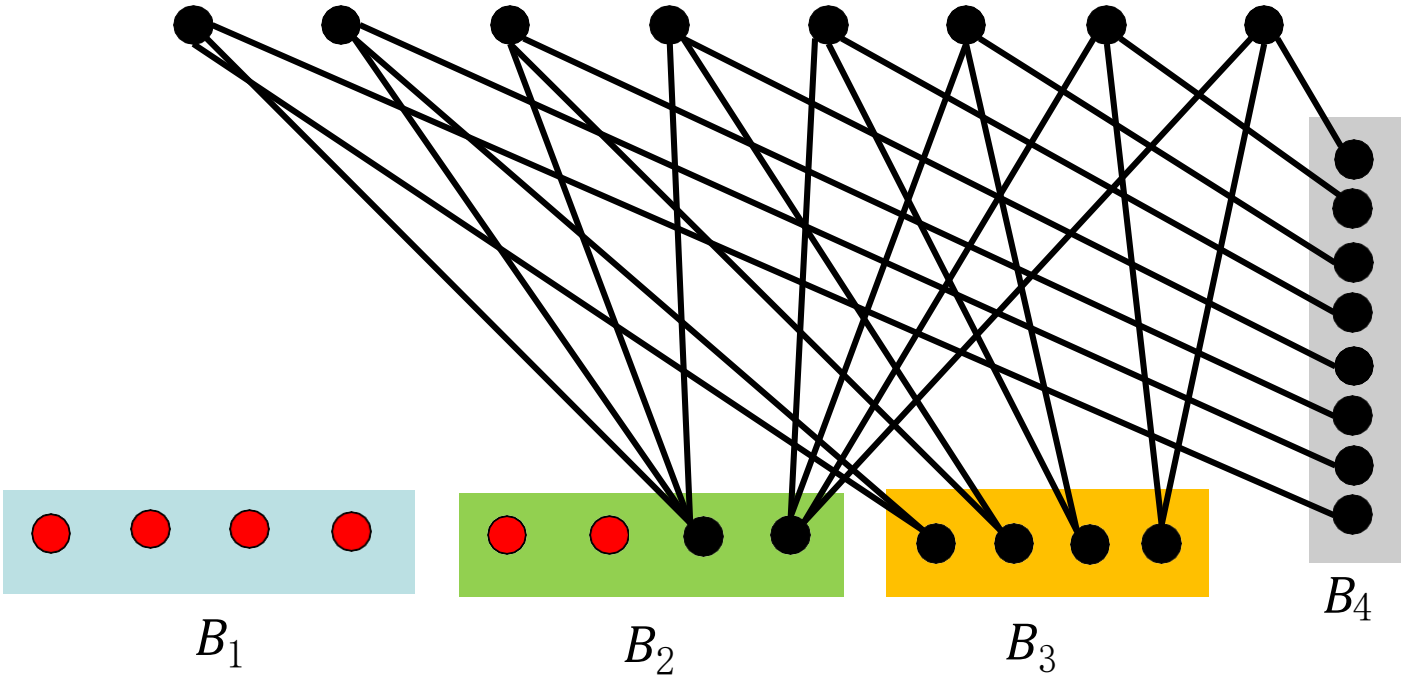
Greedy (1): Pick vertex that covers the most



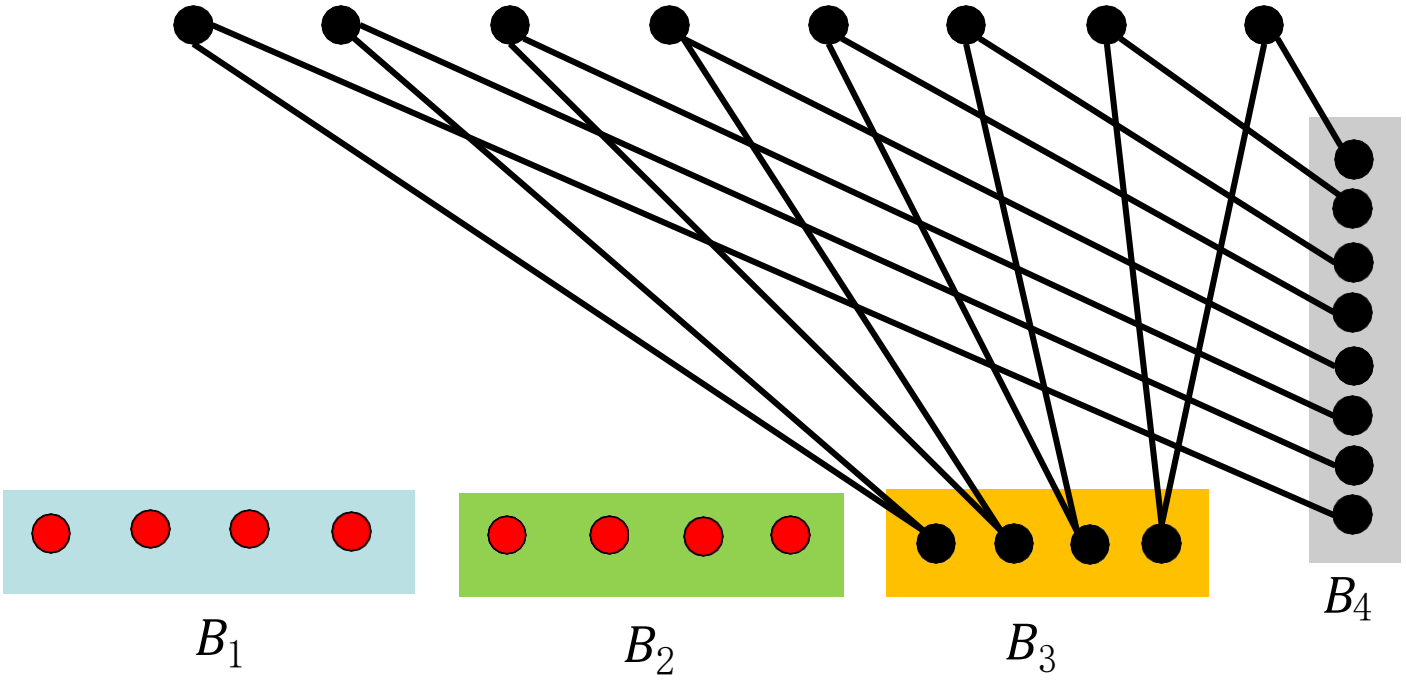
Greedy (1): Pick vertex that covers the most



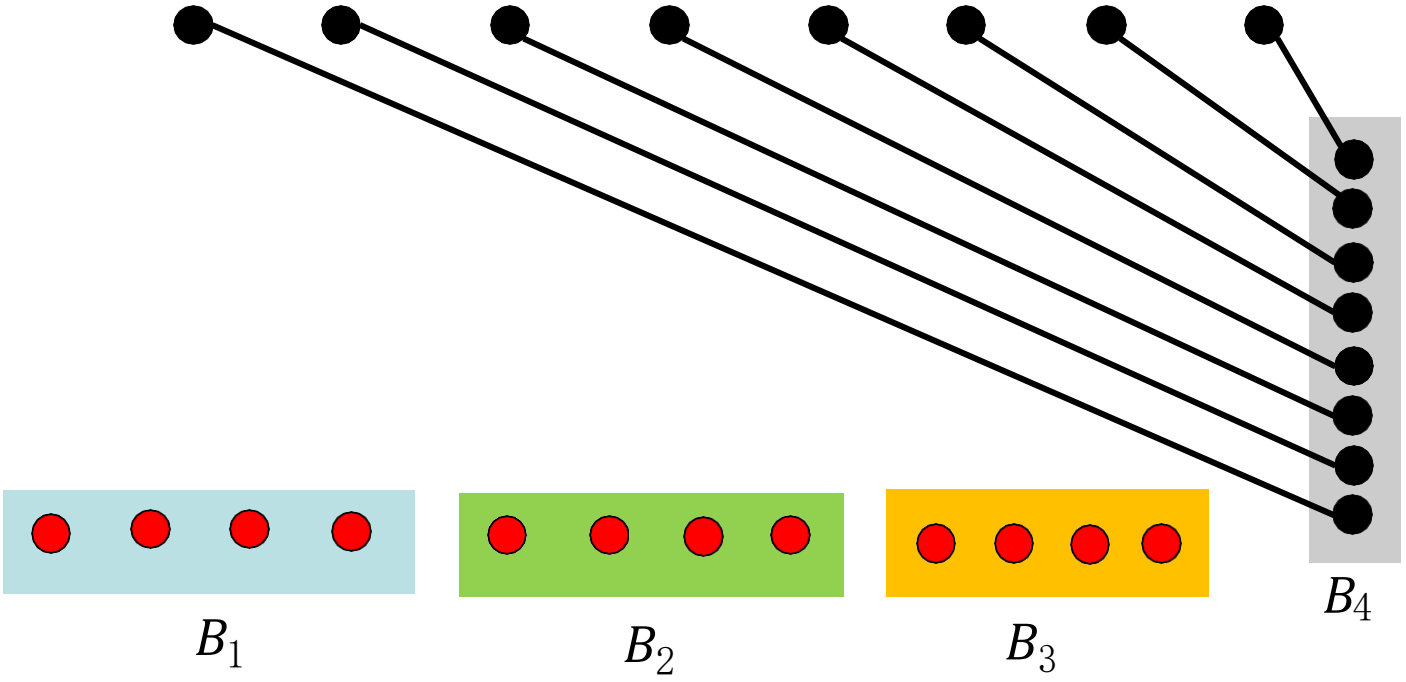
Greedy (1): Pick vertex that covers the most



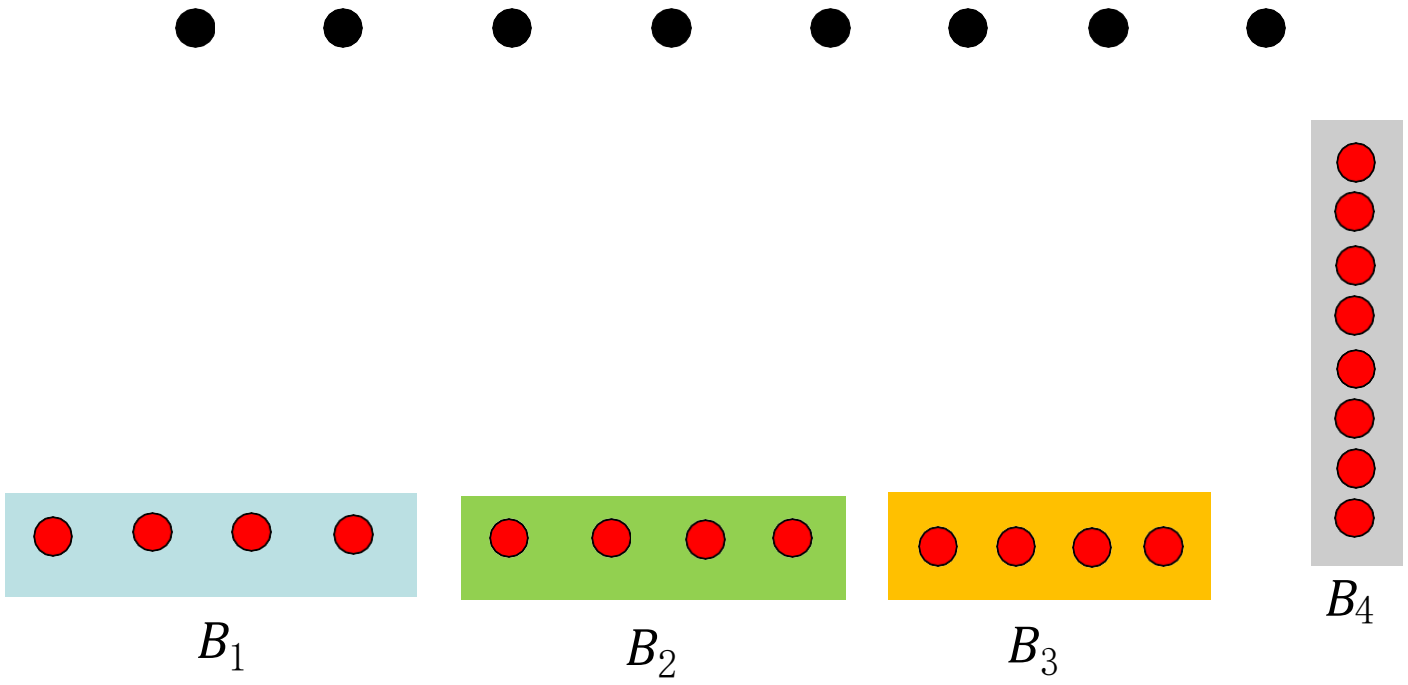
Greedy (1): Pick vertex that covers the most



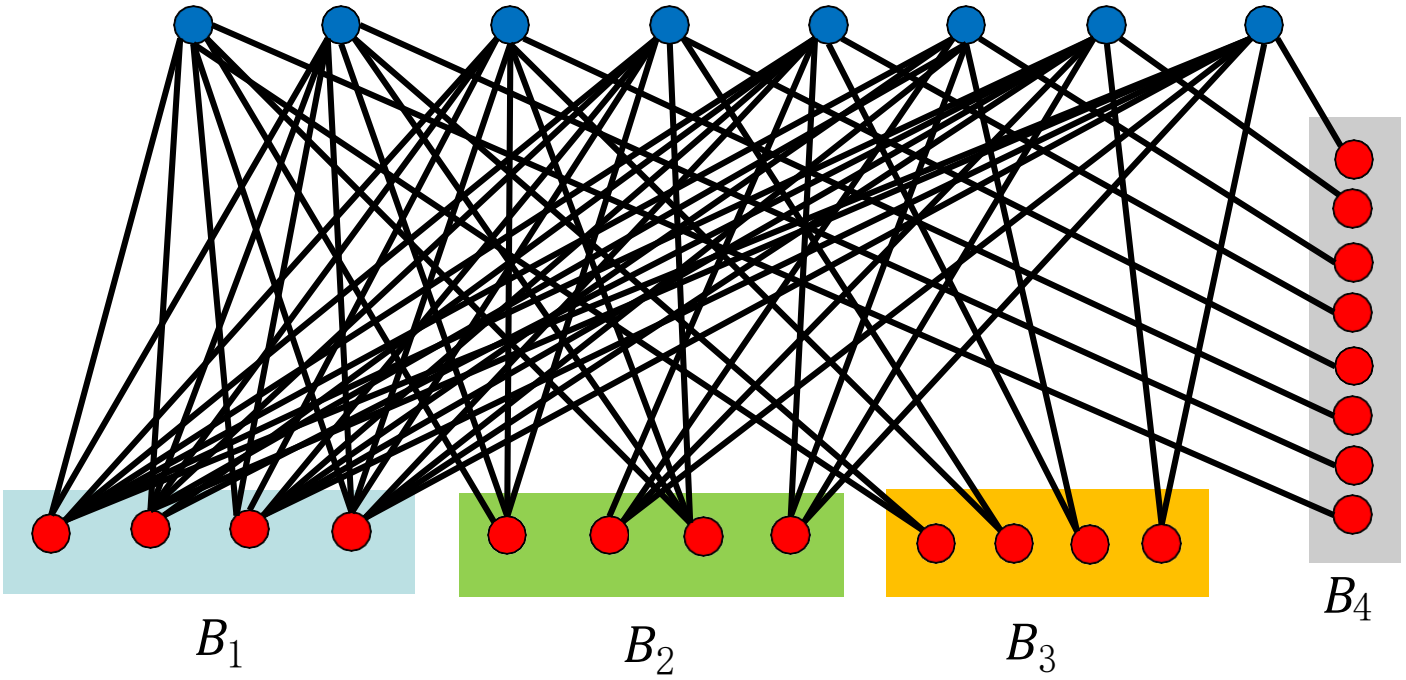
Greedy (1): Pick vertex that covers the most



Greedy (1): Pick vertex that covers the most



Greedy (1): Pick vertex that covers the most

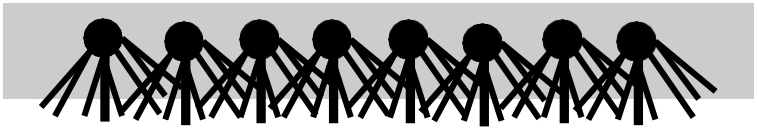


Greedy Vertex cover = 20

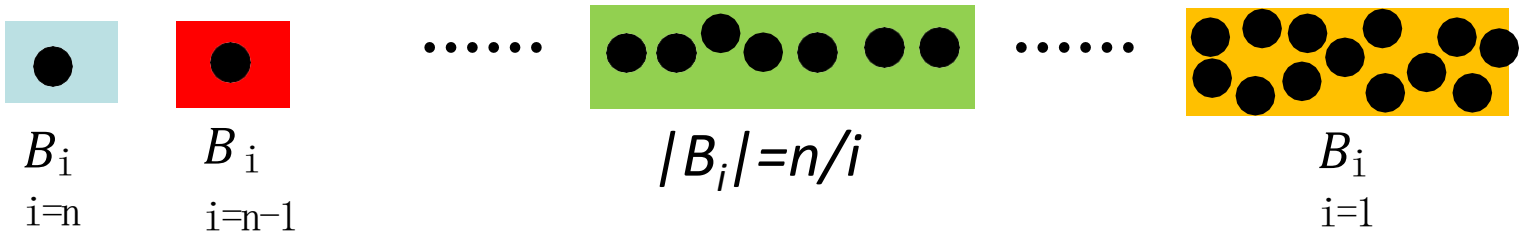
OPT Vertex cover = 8

Greedy (1): Pick vertex that covers the most

n vertices. Each vertex has i edges into each node in B_i



n/i nodes



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

OPT pick top vertices = n

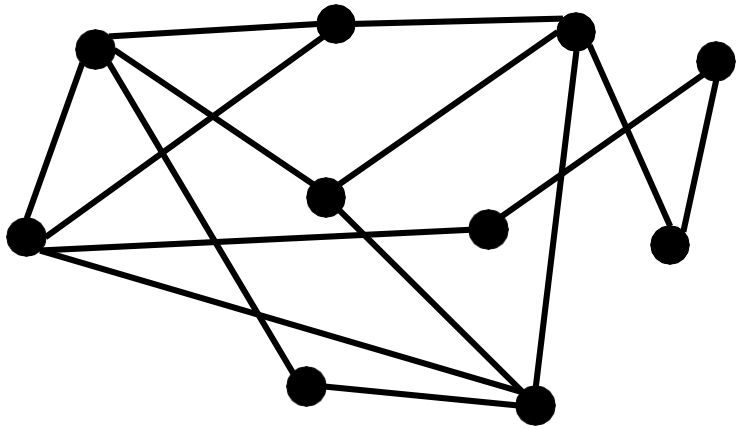
对应于B_1

调和级数

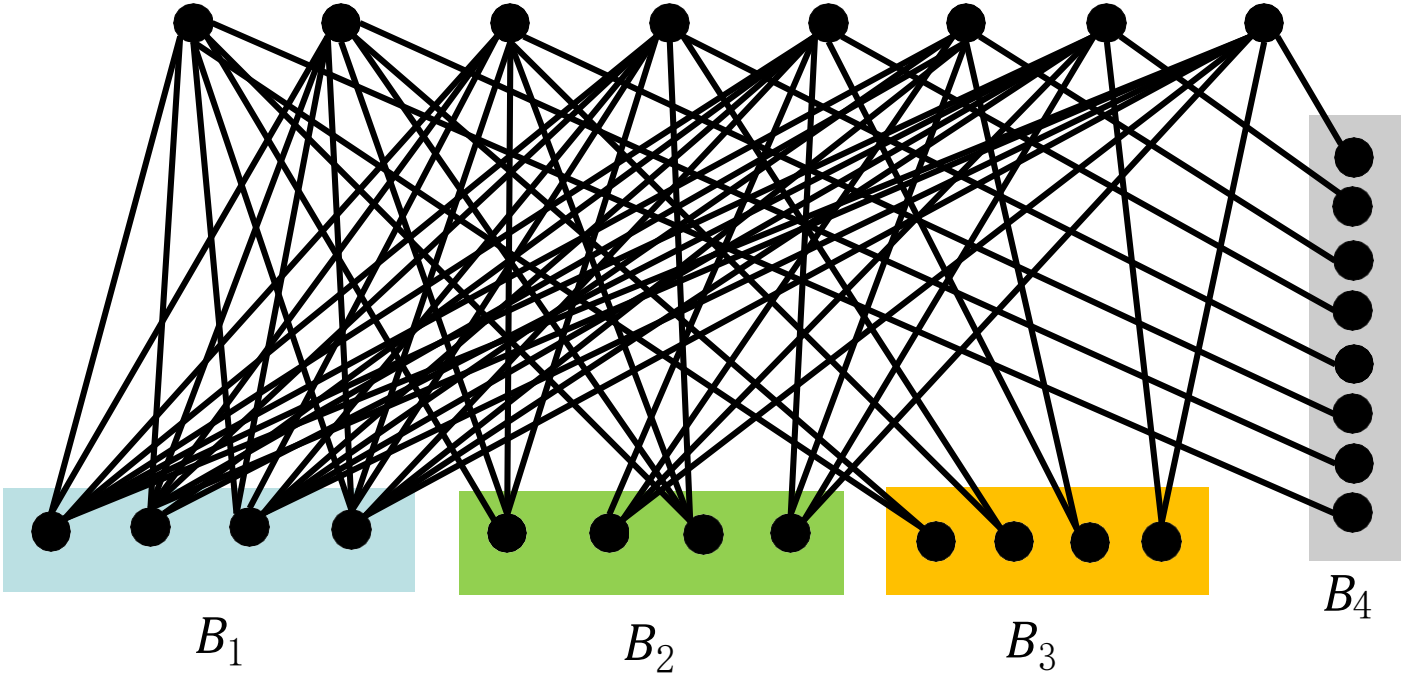
A Different Greedy Rule

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 $e_1, \cdots, e_k$.

Since these edges do not touch, every valid cover must pick one vertex from each of these edges!

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

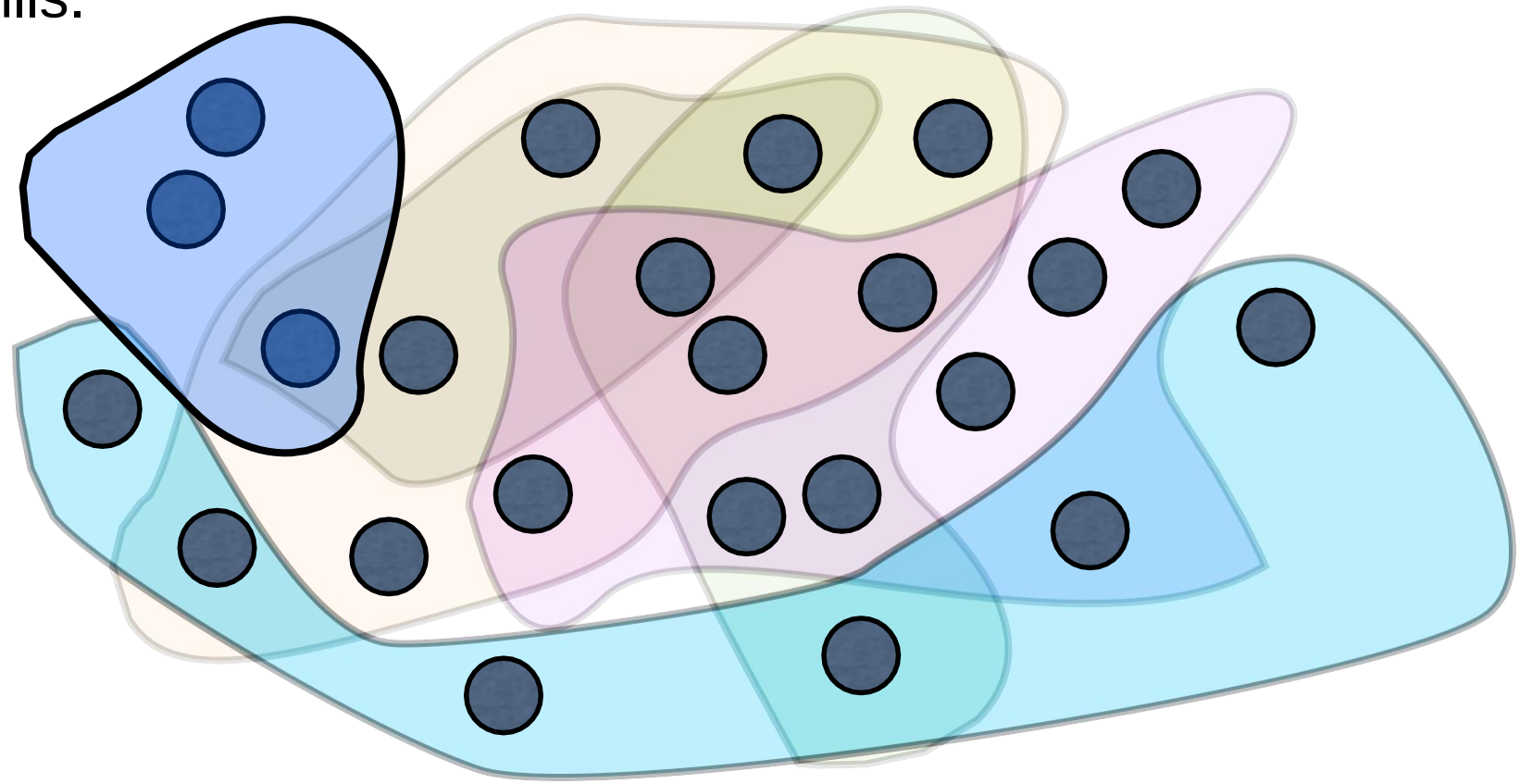
But the size of greedy cover is 2k. So, Greedy is a 2-approximation.

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.

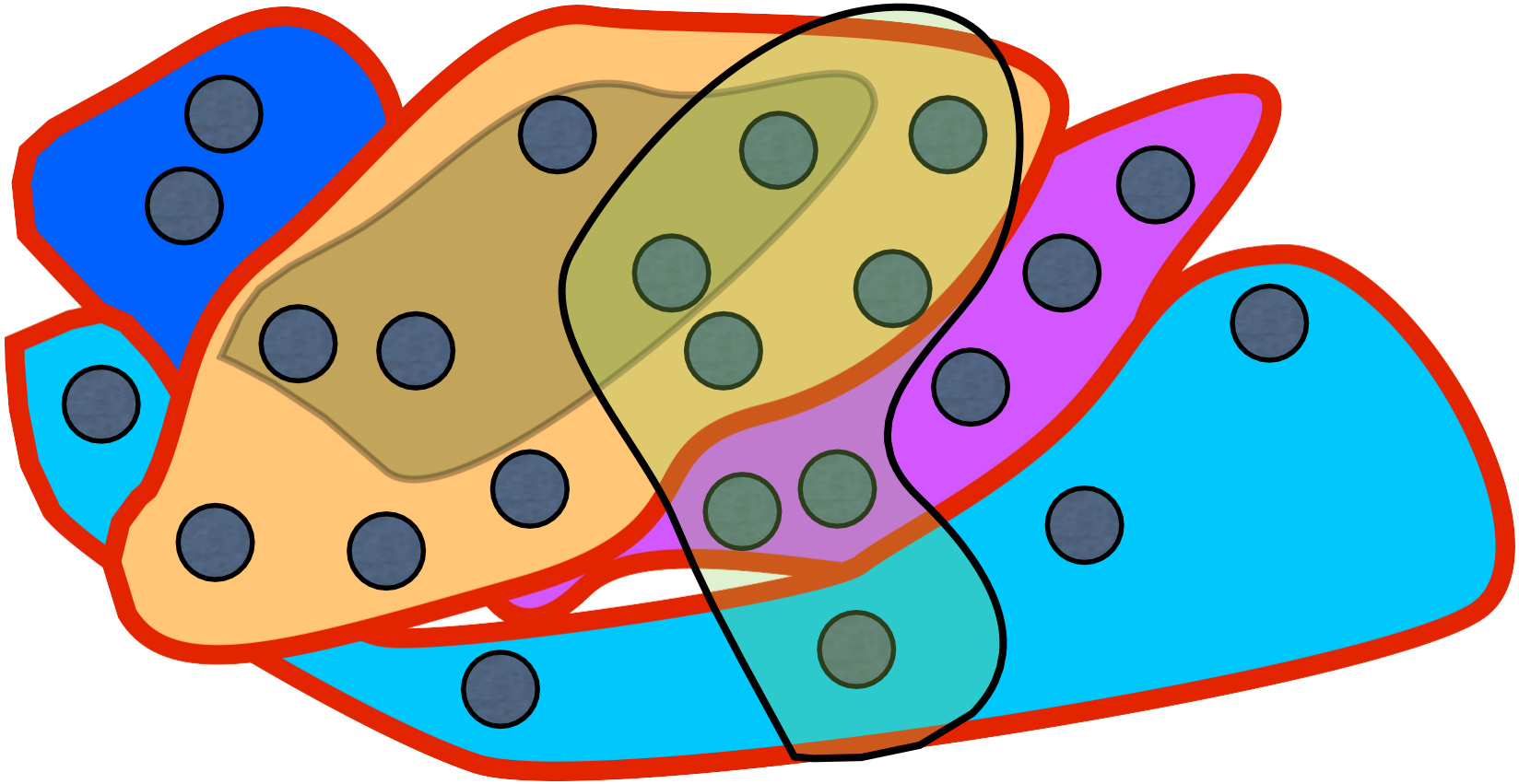


Set Cover

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

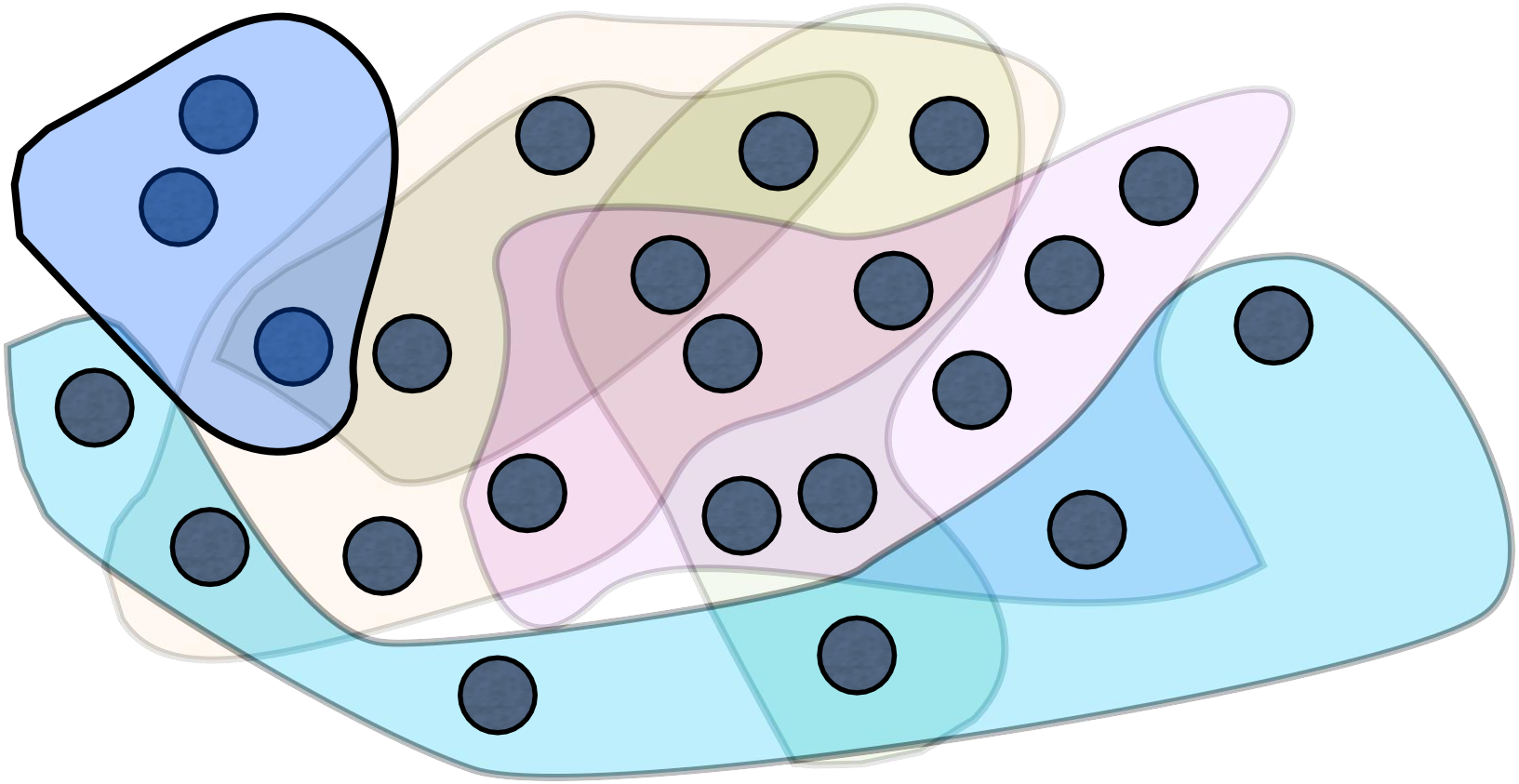
Goal: choose minimum number of sets that cover all.

Set cover = 4



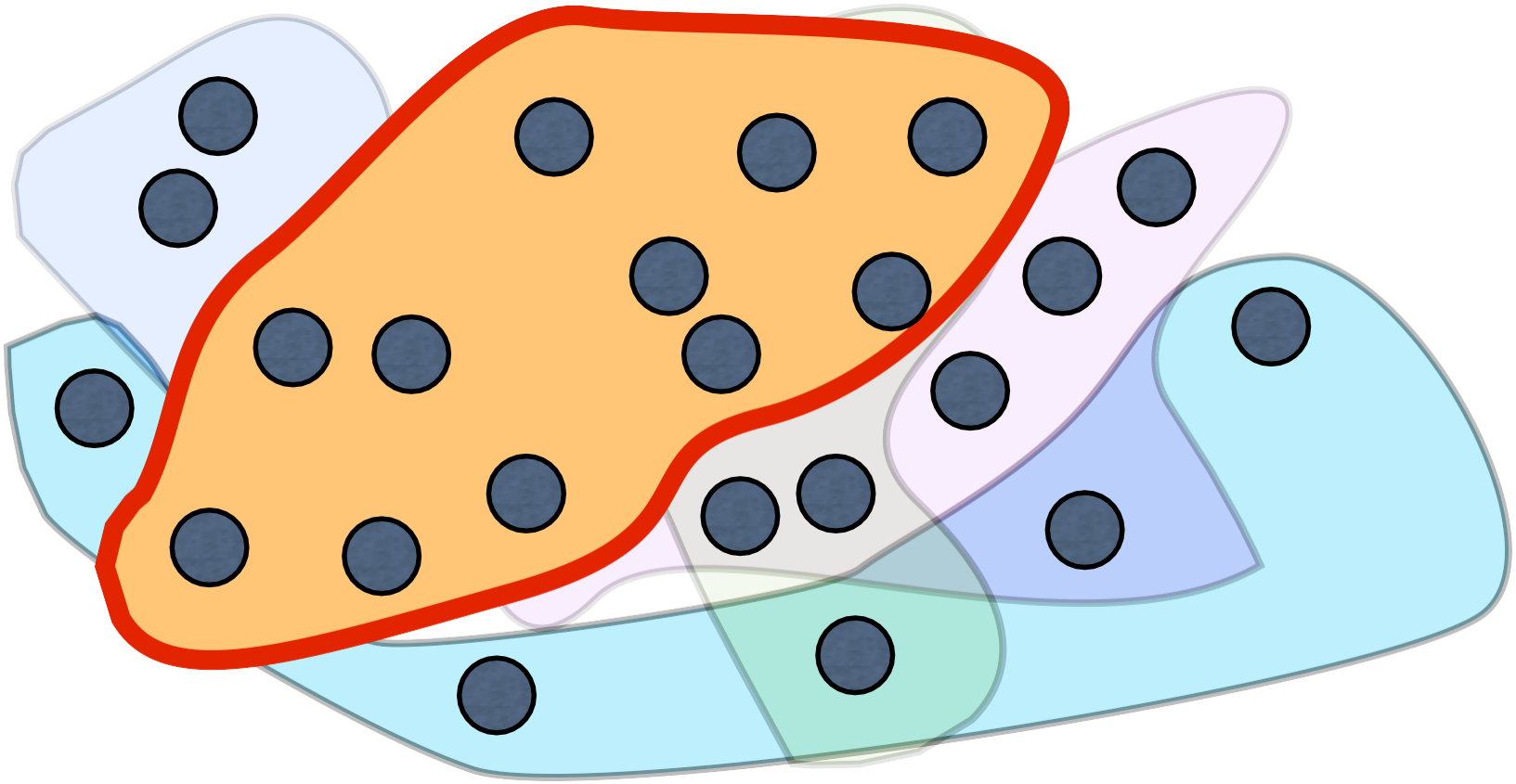
A Greedy Algorithm

Strategy: Pick the set that maximizes # new elements covered



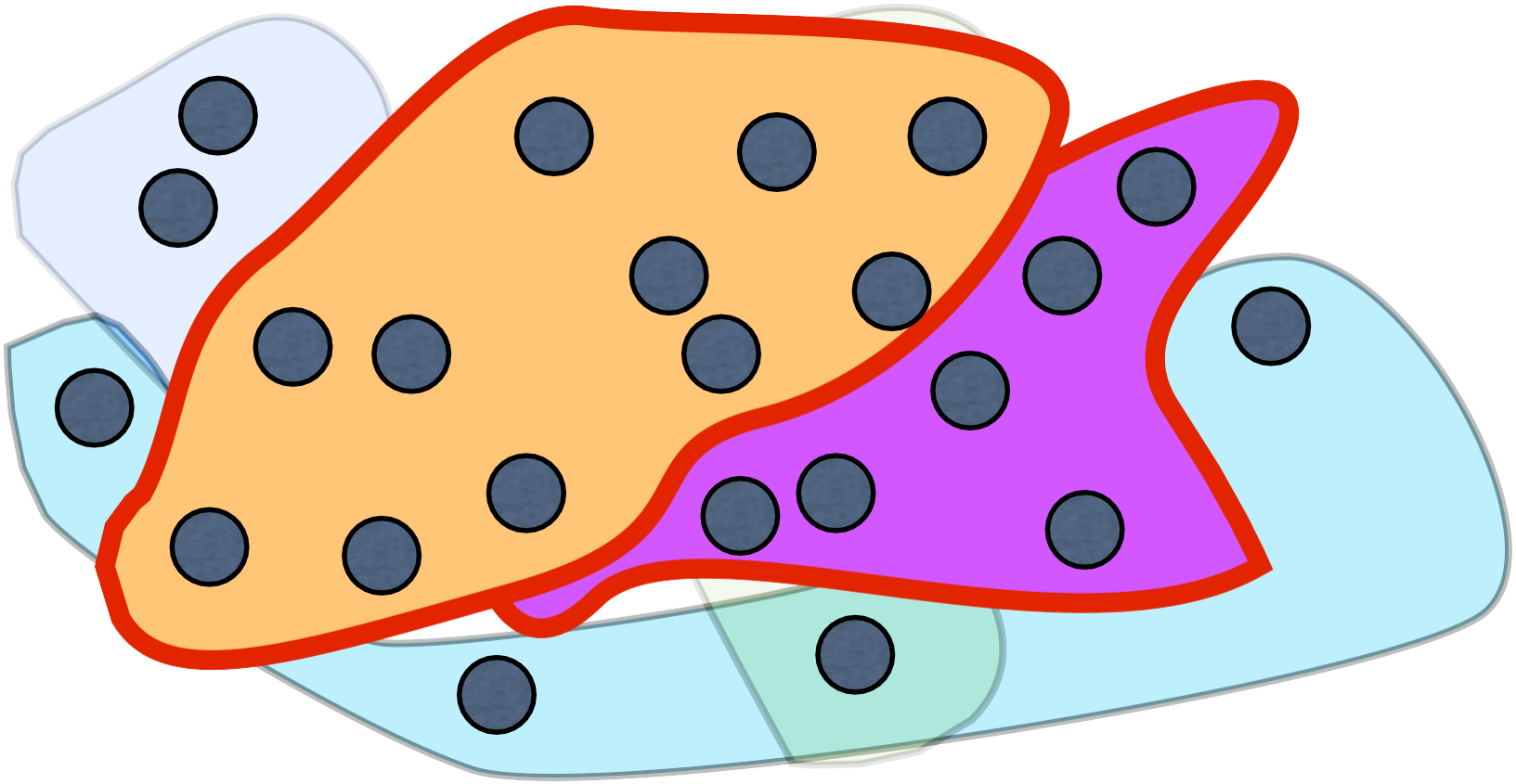
A Greedy Algorithm

Strategy: Pick the set that maximizes # new elements covered



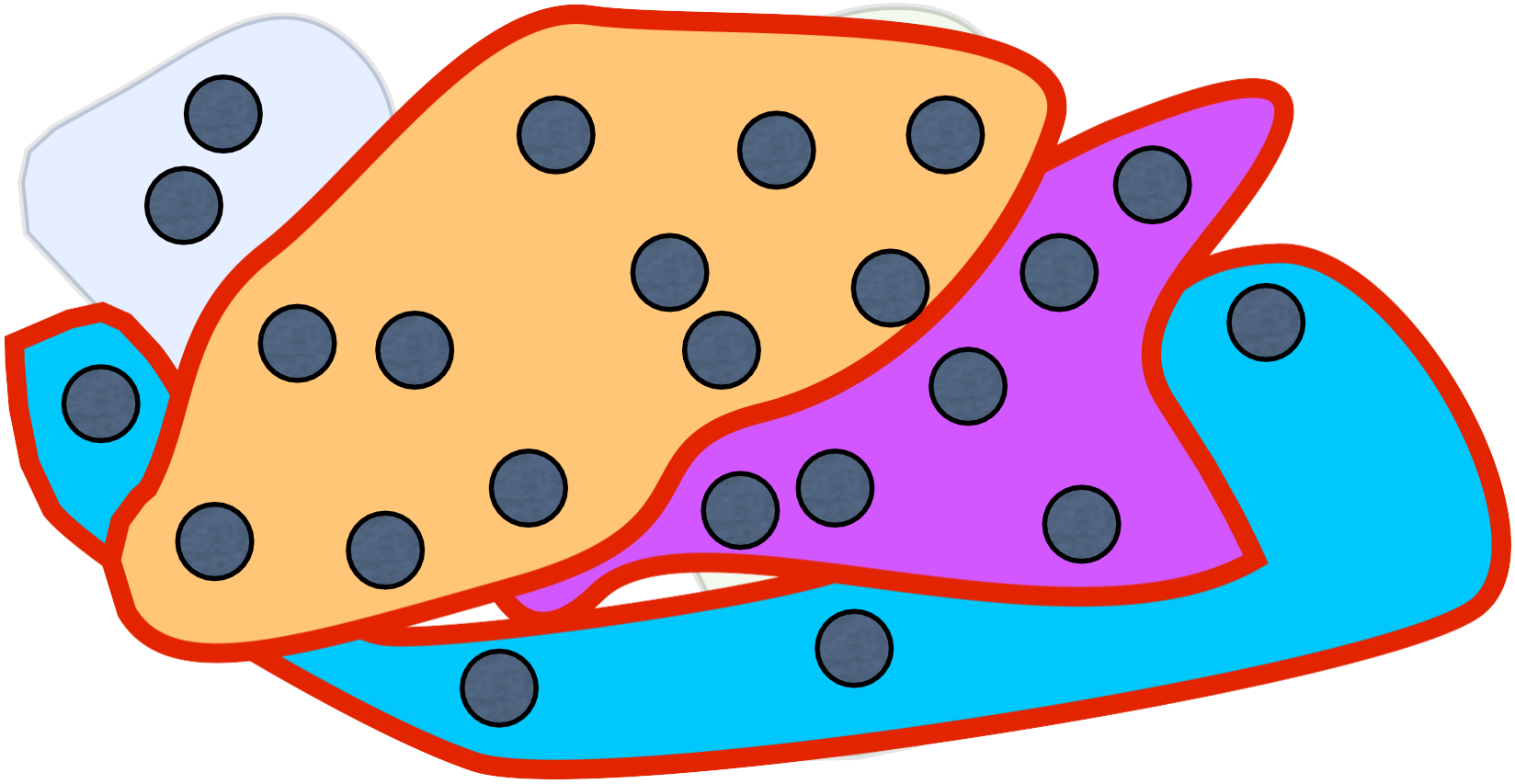
A Greedy Algorithm

Strategy: Pick the set that maximizes # new elements covered



A Greedy Algorithm

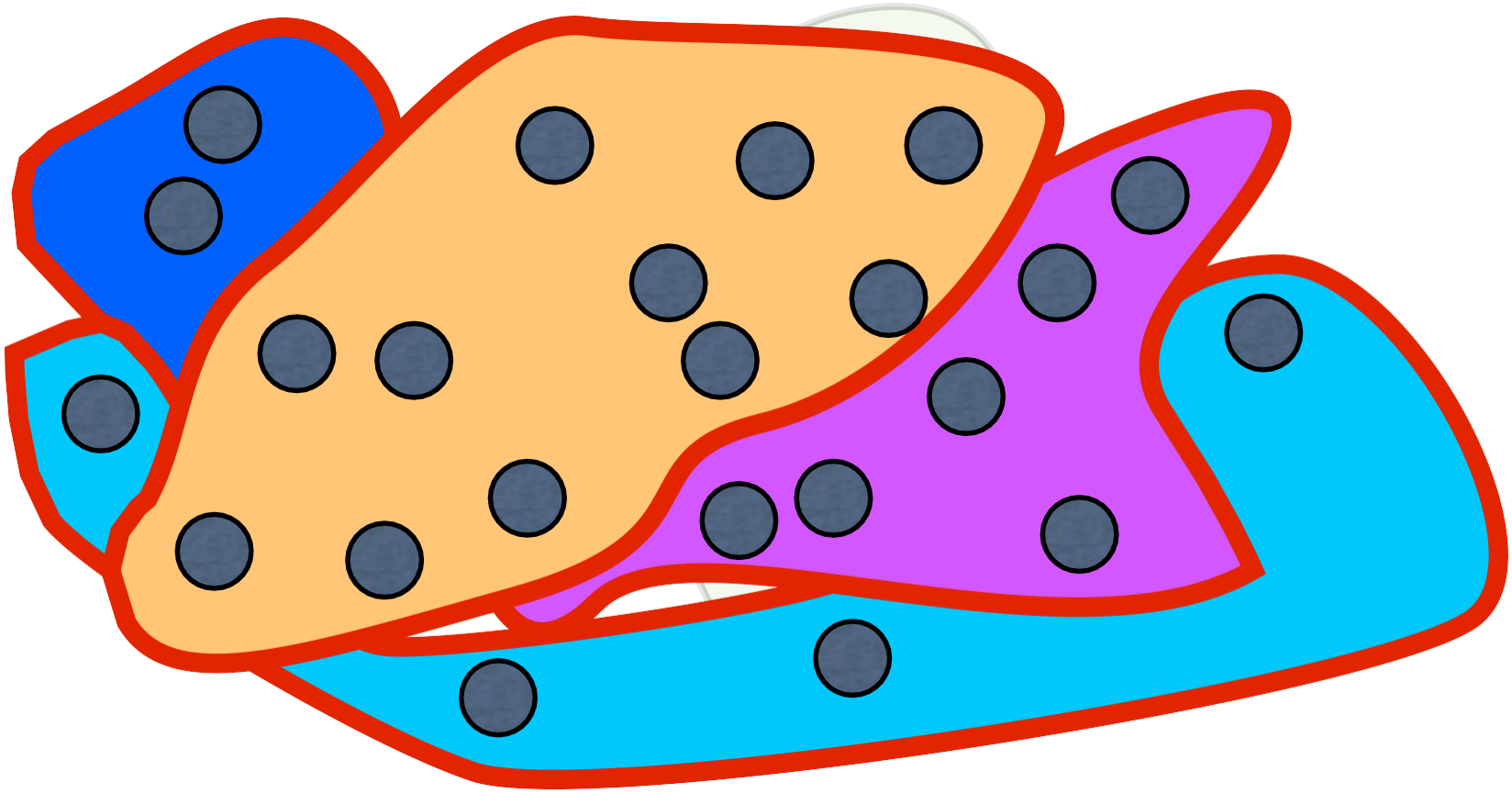
Strategy: Pick the set that maximizes # new elements covered



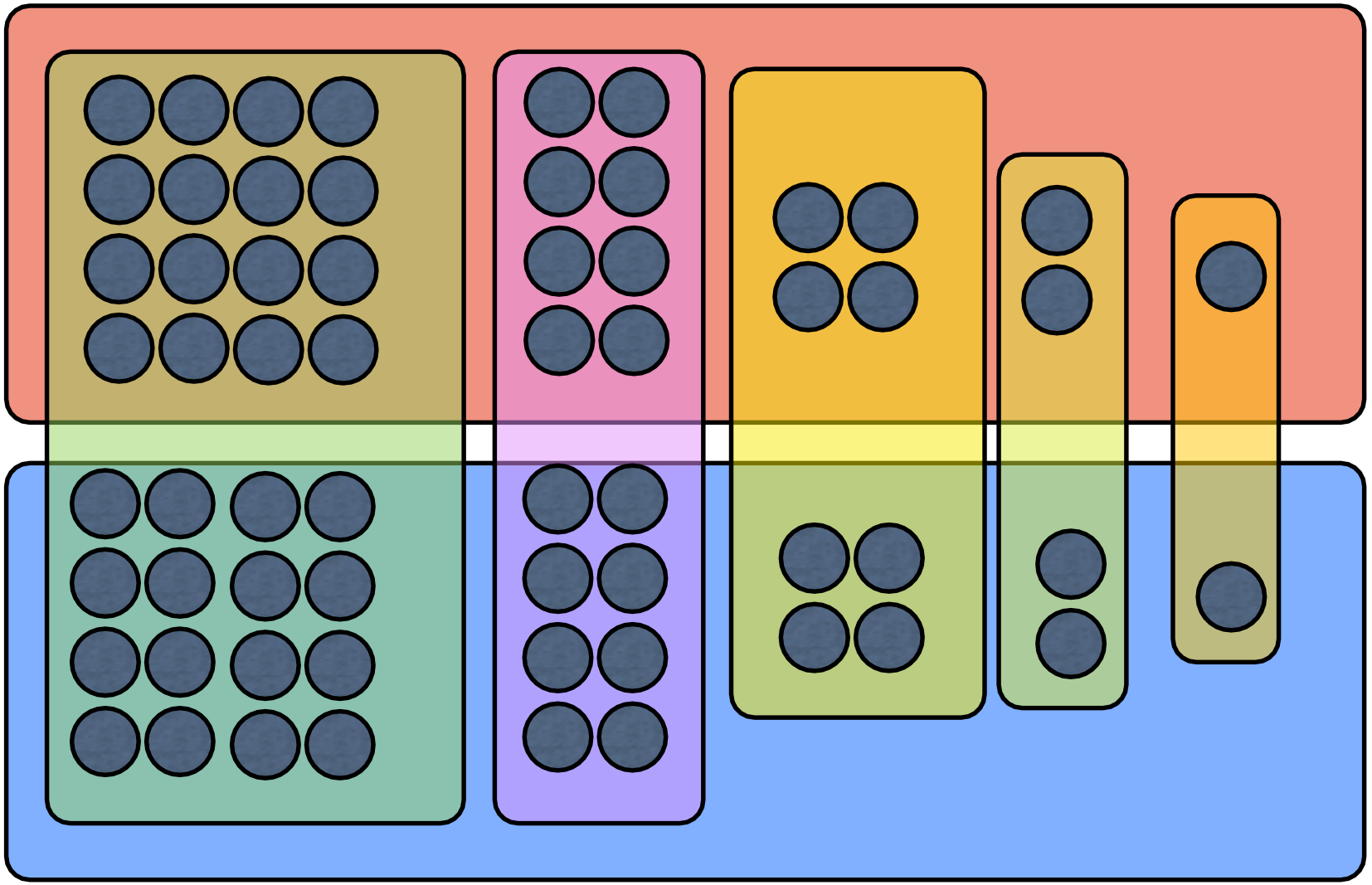
A Greedy Algorithm

Strategy: Pick the set that maximizes # new elements covered

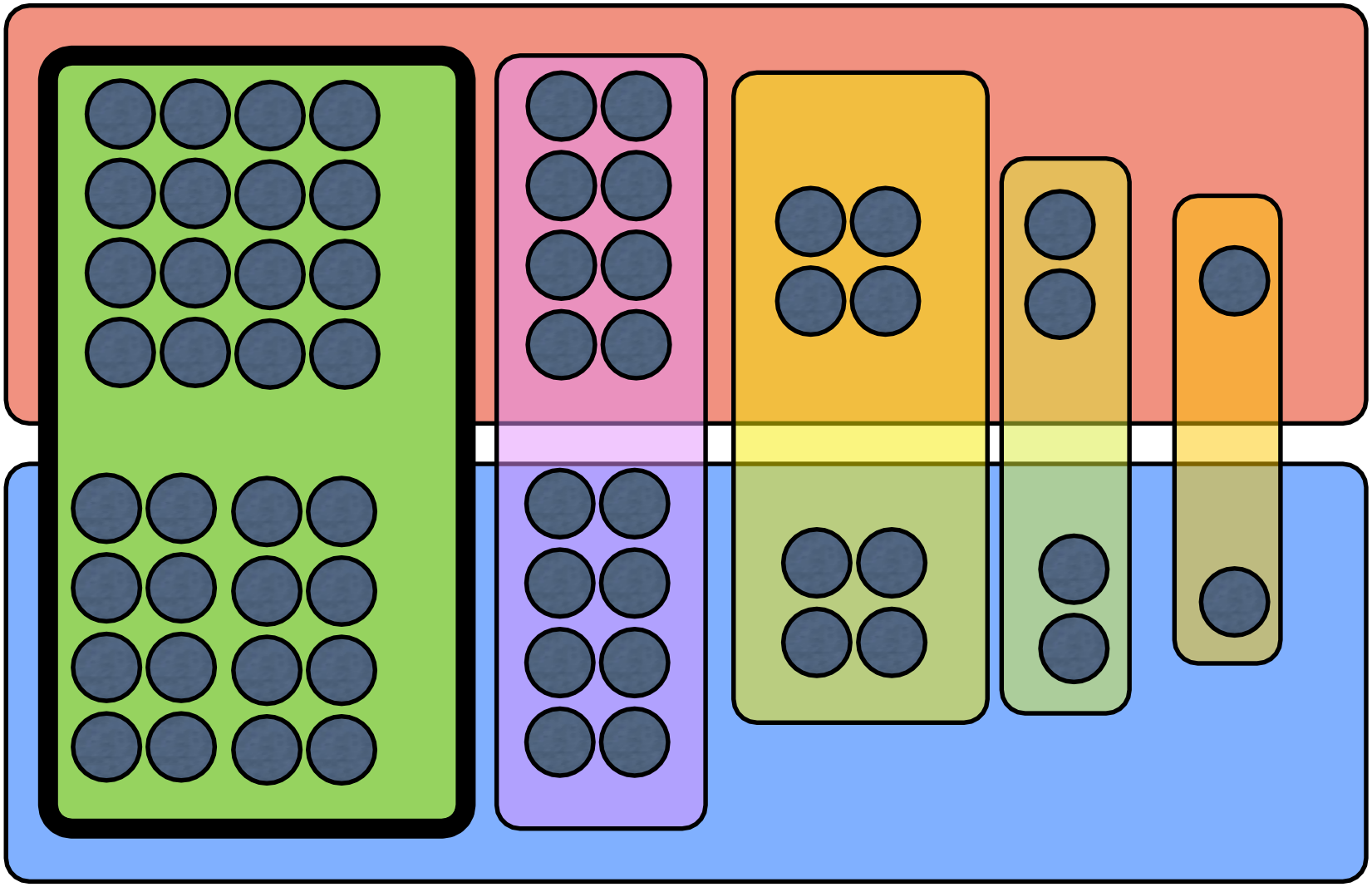
Thm: Greedy has ln n approximation ratio



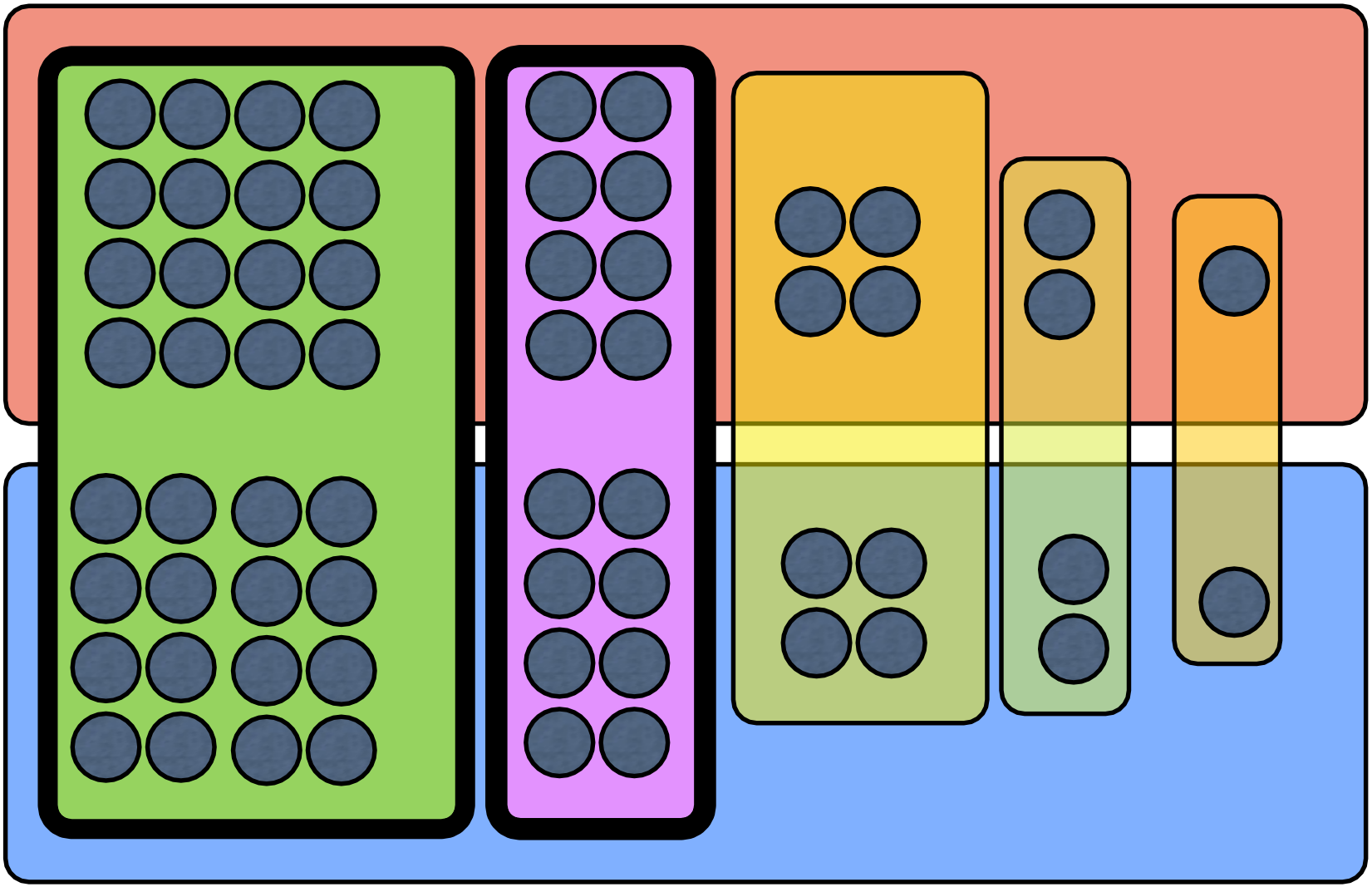
A Tight Example for Greedy



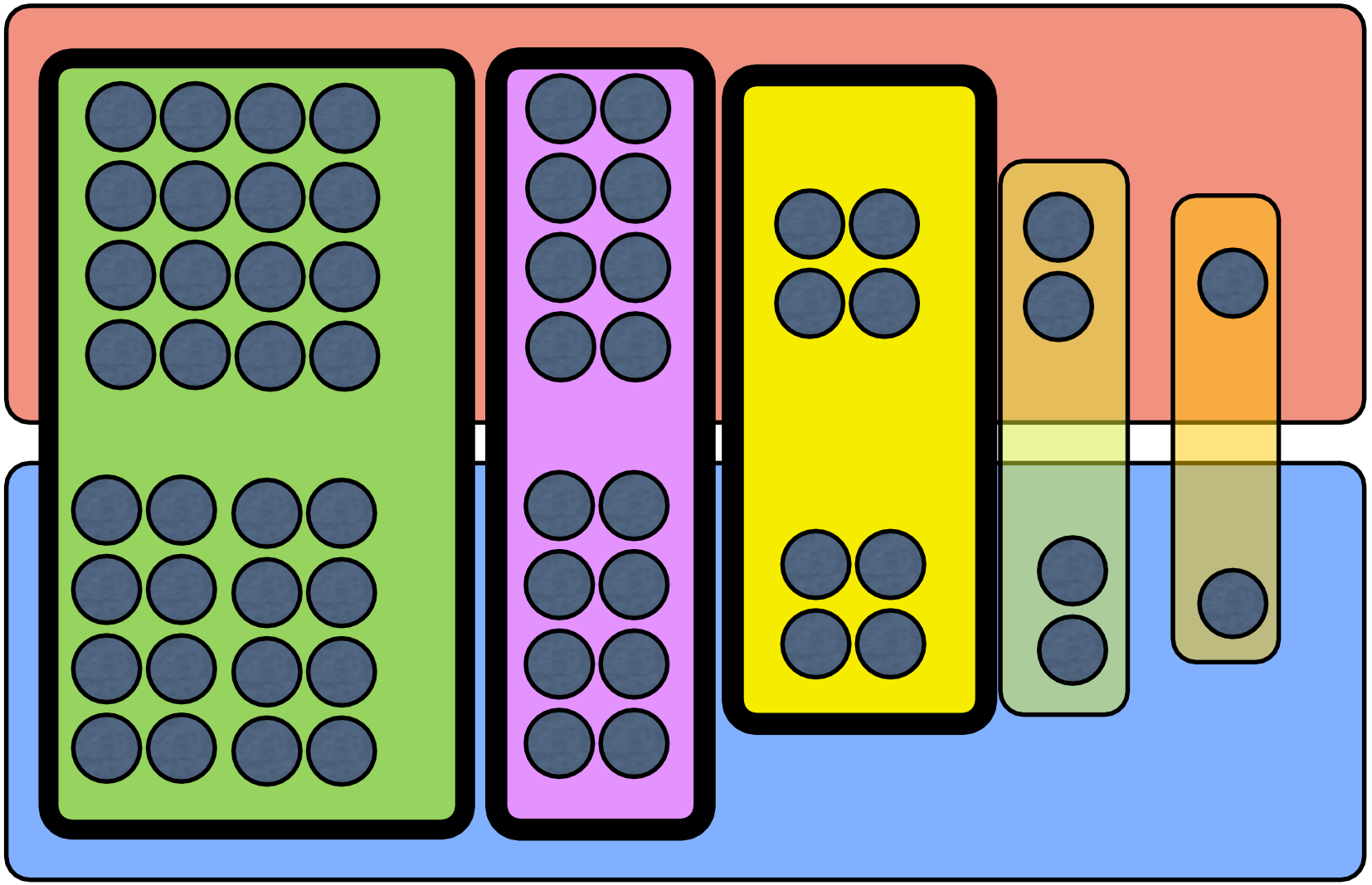
A Tight Example for Greedy



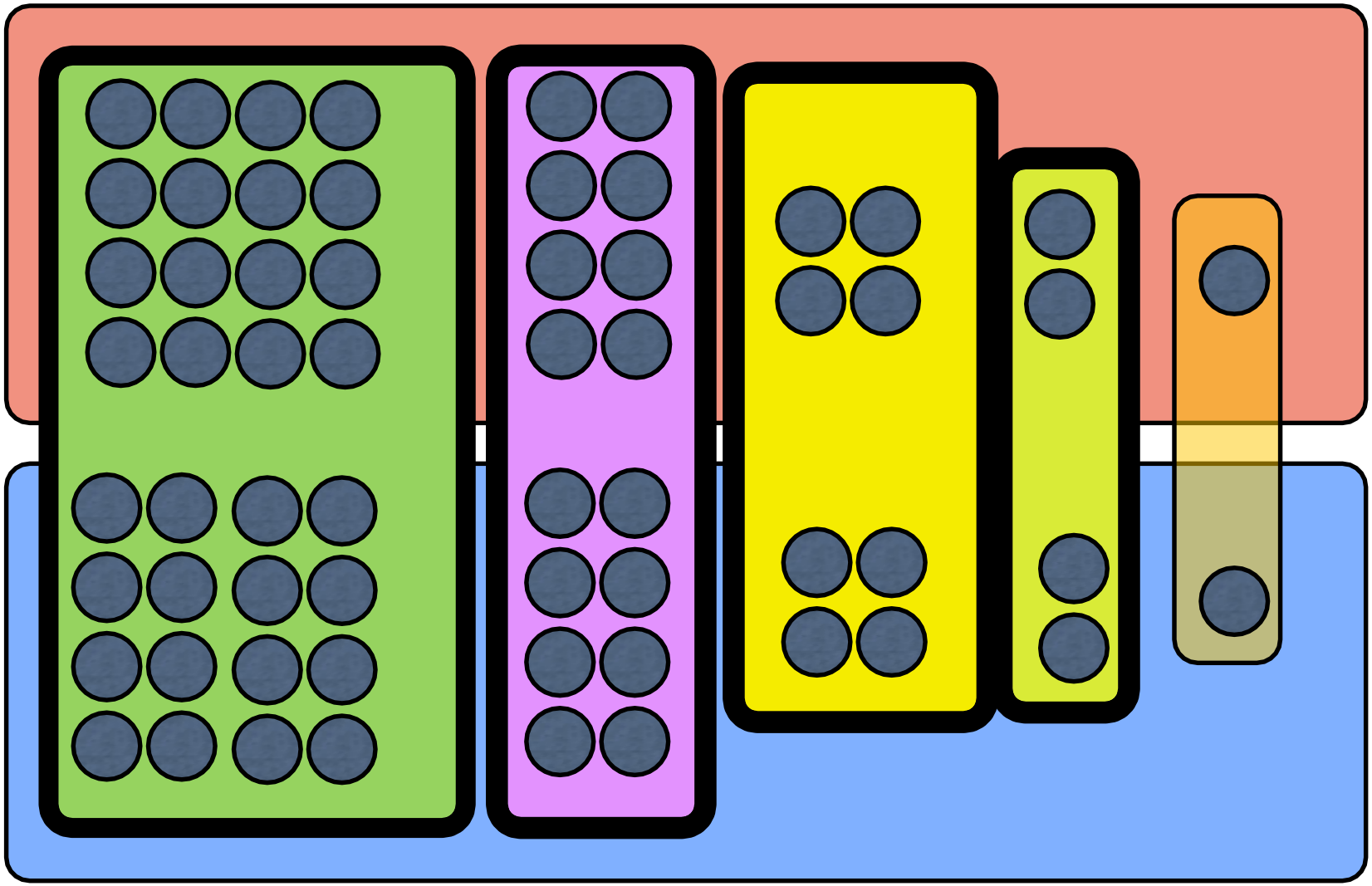
A Tight Example for Greedy



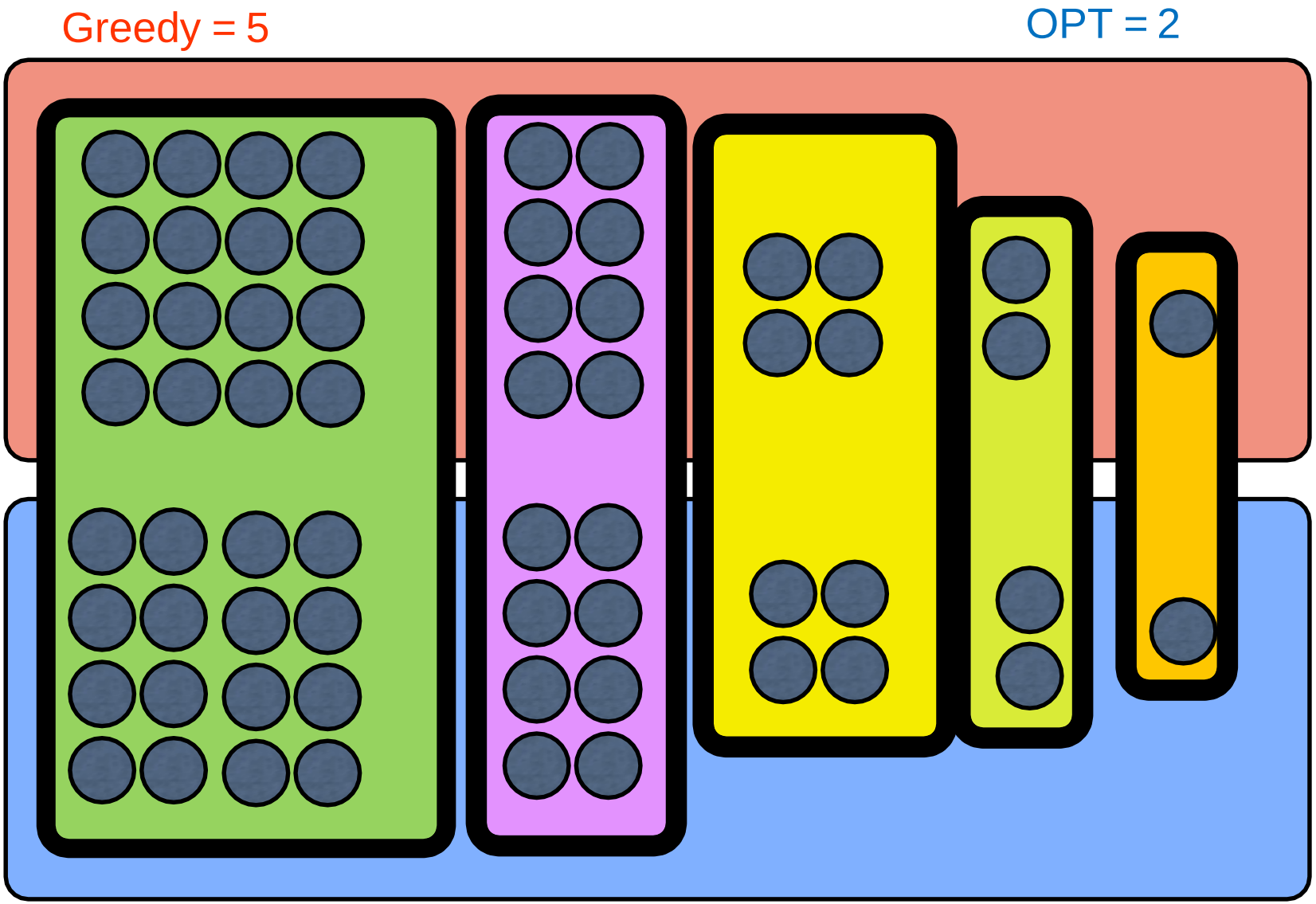
A Tight Example for Greedy



A Tight Example for Greedy



A Tight Example for Greedy



An Algorithm

Input: A universe *U* of n elements, a collection of subsets of *U*, T = {S_1 ,...,S_k}, and a cost function c : S -> Q.

Output: A subset T' of T s.t. *U*=$\bigcup_{S_i \in T}$, S_i

1. $C \leftarrow \emptyset$
2. While $C \neq U$ do
   Find the most cost-effective set in the current iteration, say S.
   Let $\alpha = \frac{\text{cost}(S)}{|S-C|}$, i.e., the cost-effectiveness of S.
   Pick S, and for each $e \in S - C$, set $\text{price}(e) = \alpha$.
   $C \leftarrow C \cup S$.
3. Output the picked sets.

Lemma: For each k ∈ {1,...,n}, price(e_k) ≤ OPT/(n − k + 1).

Proof: In any iteration, the leftover sets of the optimal solution can cover the remaining elements at a cost of at most OPT. Therefore, among these sets, there must be one having cost-effectiveness of at most OPT/|$\bar{C}$|(Otherwise, every set has cost-effectivensss greater than OPT/|$\bar{C}$|, then total cost will exceed OPT). In the iteration in which element e_k was covered, $\bar{C}$ contained at least n − k + 1 elements(Current S_i contains at least one element, hence C contains no more than k-1 elements). Since e_k was covered by the most cost-effective set in this iteration, it follows that

$$\text{price}(e_k) \leq \frac{\text{OPT}}{|\overline{C}|} \leq \frac{\text{OPT}}{n-k+1}.$$

Approximation Ratio

Theorem: The greedy algorithm is an H_n factor approximation algorithm for the minimum set cover problem, where

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$$

Proof: Since the cost of each set picked is distributed among the new elements covered, the total cost of the set cover picked is equal to $\sum_{k=1}^{n} price(e_k)$.
By Lemma, this is at most $(1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n})$ OPT.

$$1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \approx \ln(n)$$

$$\text{price}(e_k) \leq \frac{\text{OPT}}{|\overline{C}|} \leq \frac{\text{OPT}}{n-k+1}.$$