Advanced Algorithmic Techniques (COMP523)

Greedy Algorithms 3

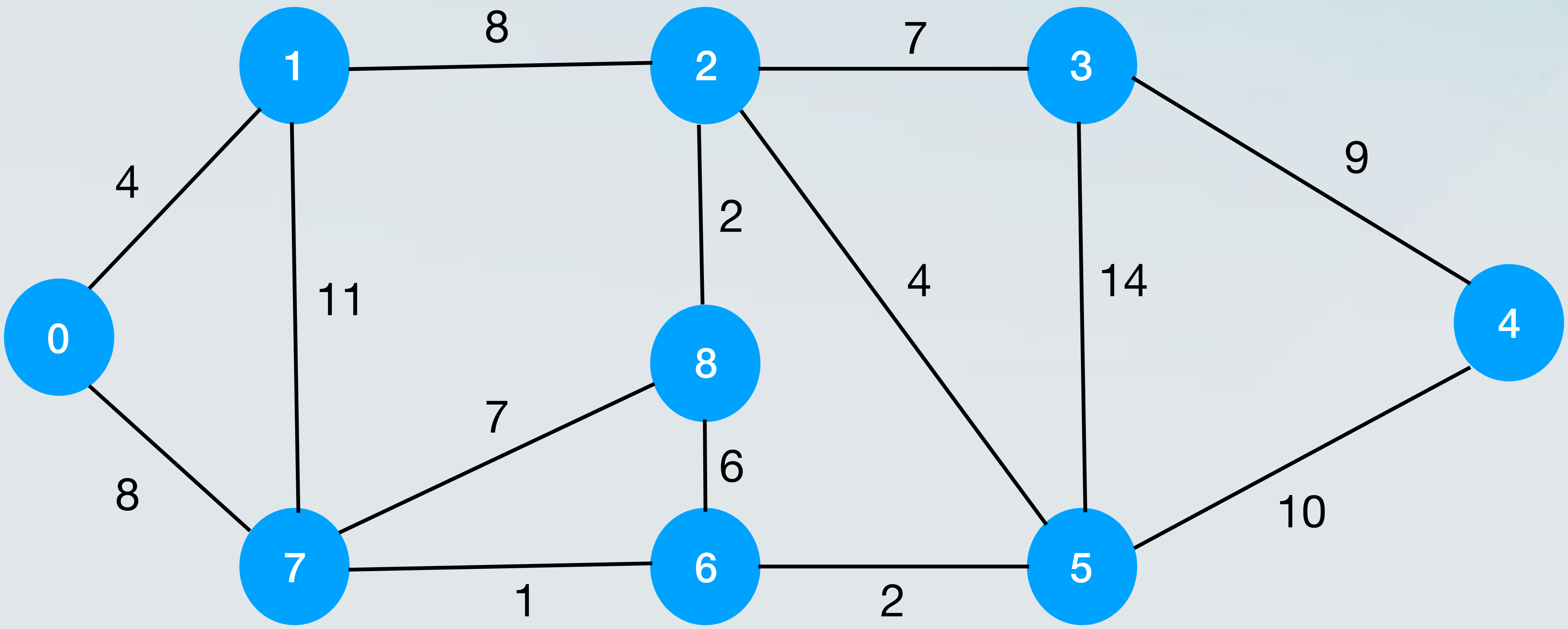
Recap and plan

- **Last lecture:**
  - Minimum Spanning Tree
  - Kruskal’s Algorithm
  - Prim’s Algorithm
- **This lecture:**
  - Prim’s Algorithm (cont.)
  - Clustering

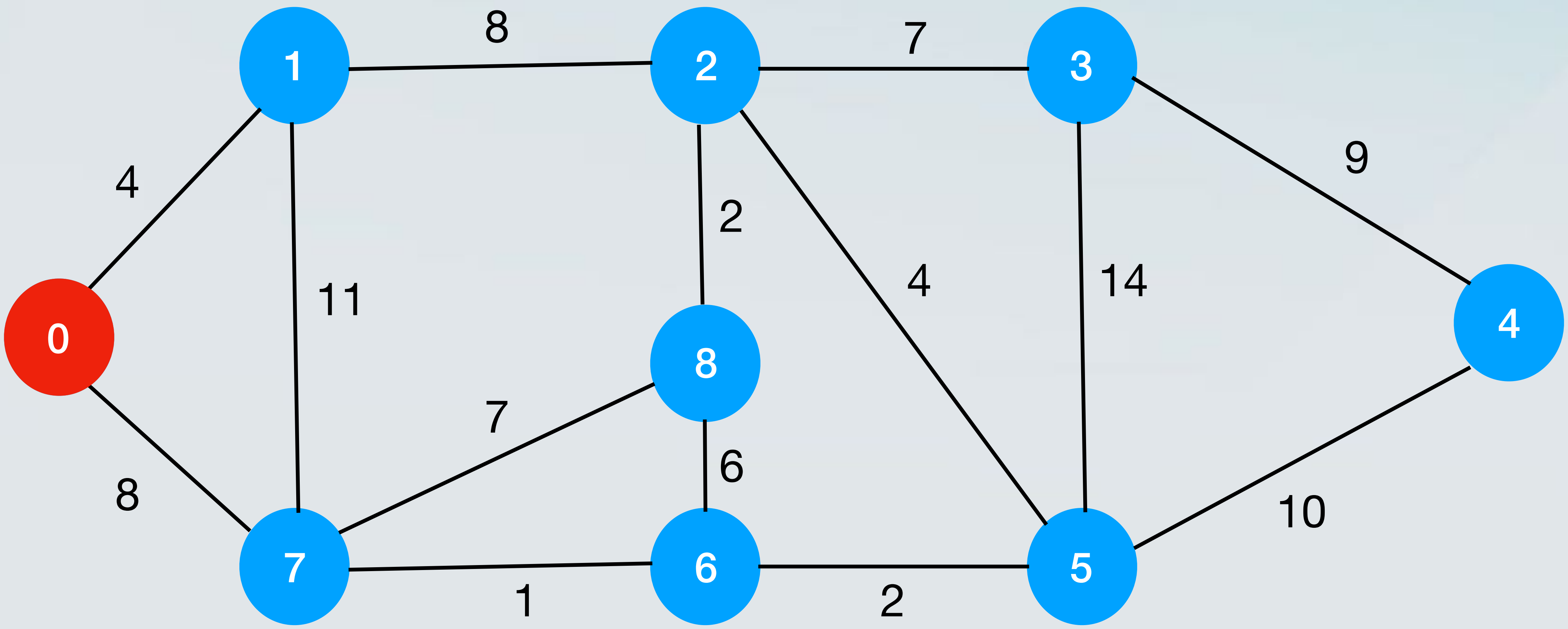
Prim's Algorithm

- Start with an empty set of edges T.
- Start with a node s.
  - Add an edge e=(s,w) to T.
  - Which one?
  - The one with the minimum cost c_e.
- We continue like this.
  - We only consider edges to neighbours that are not in the spanning tree.

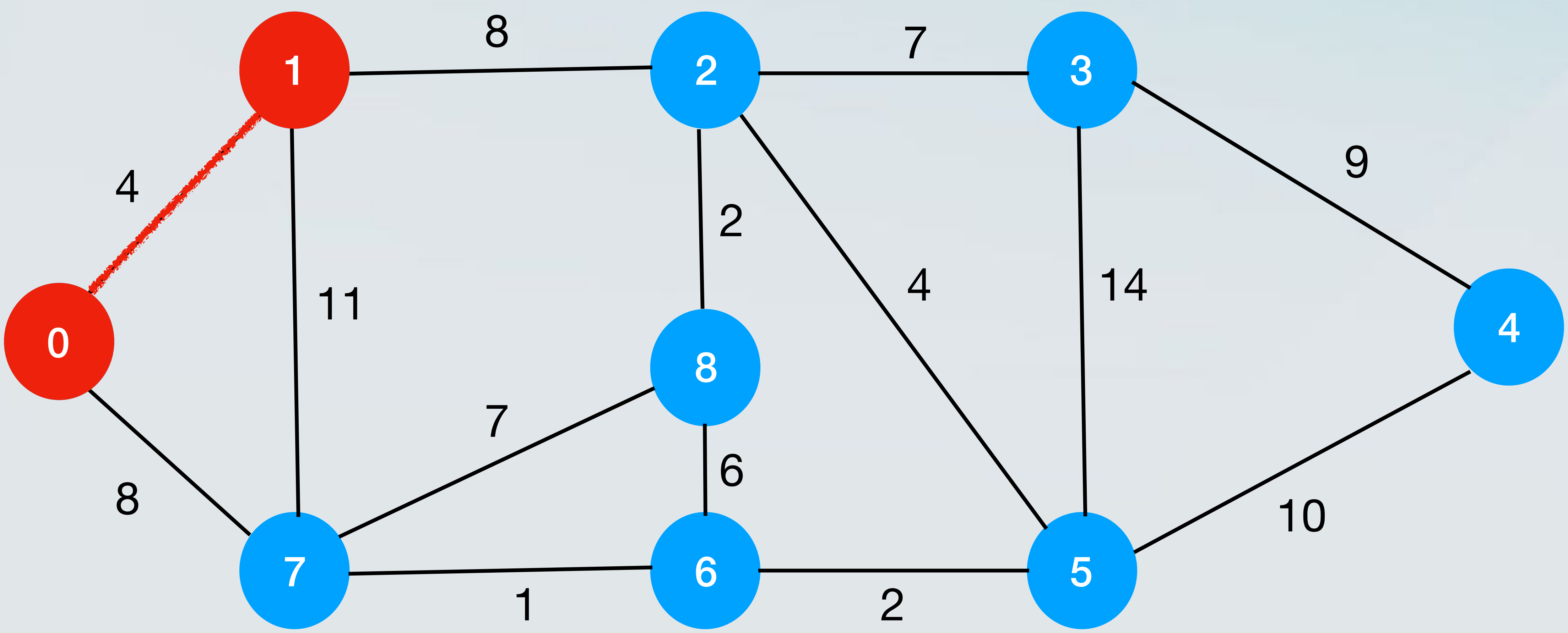
Example



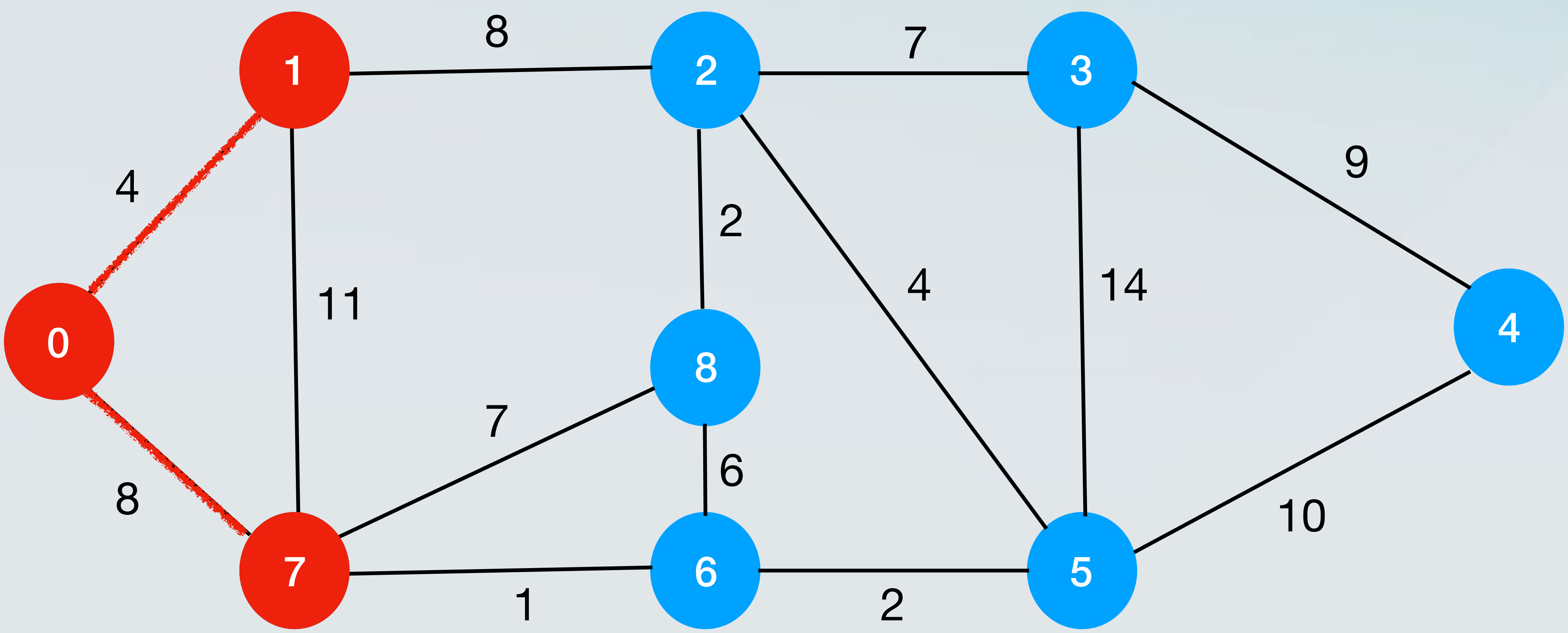
Example



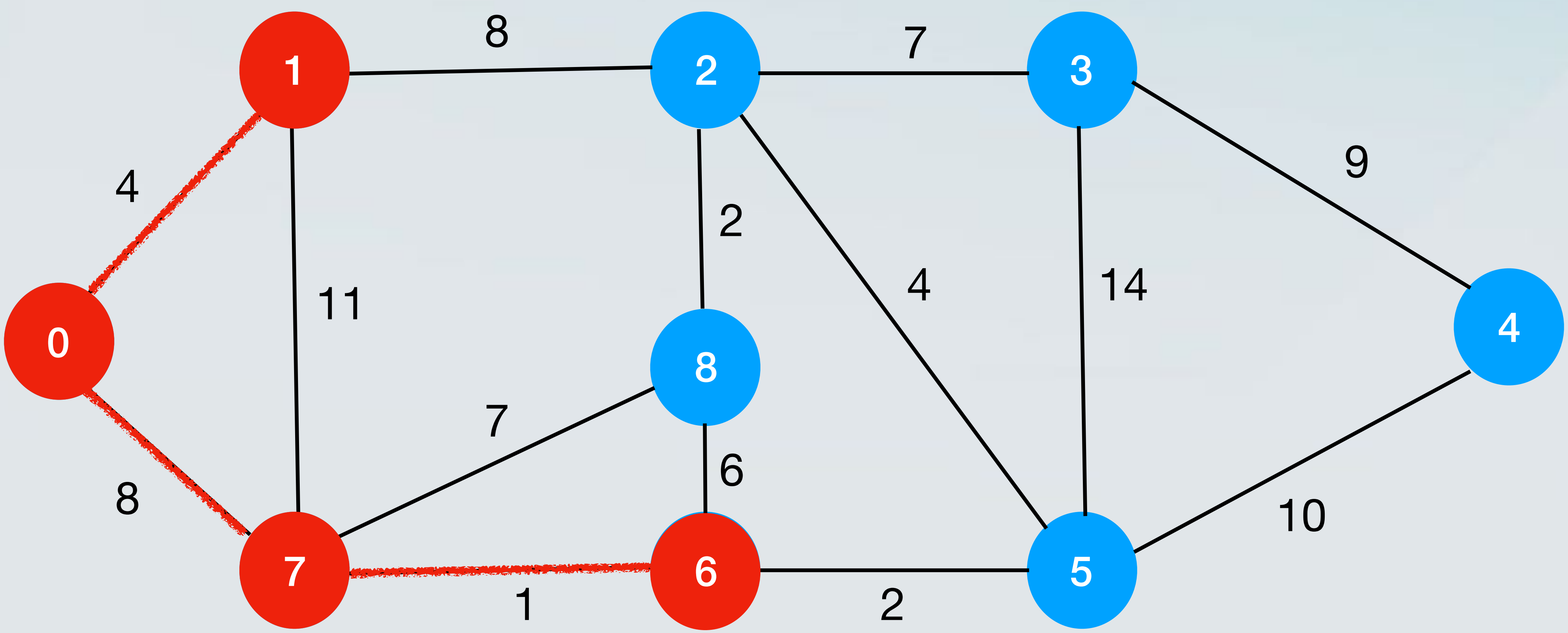
Example



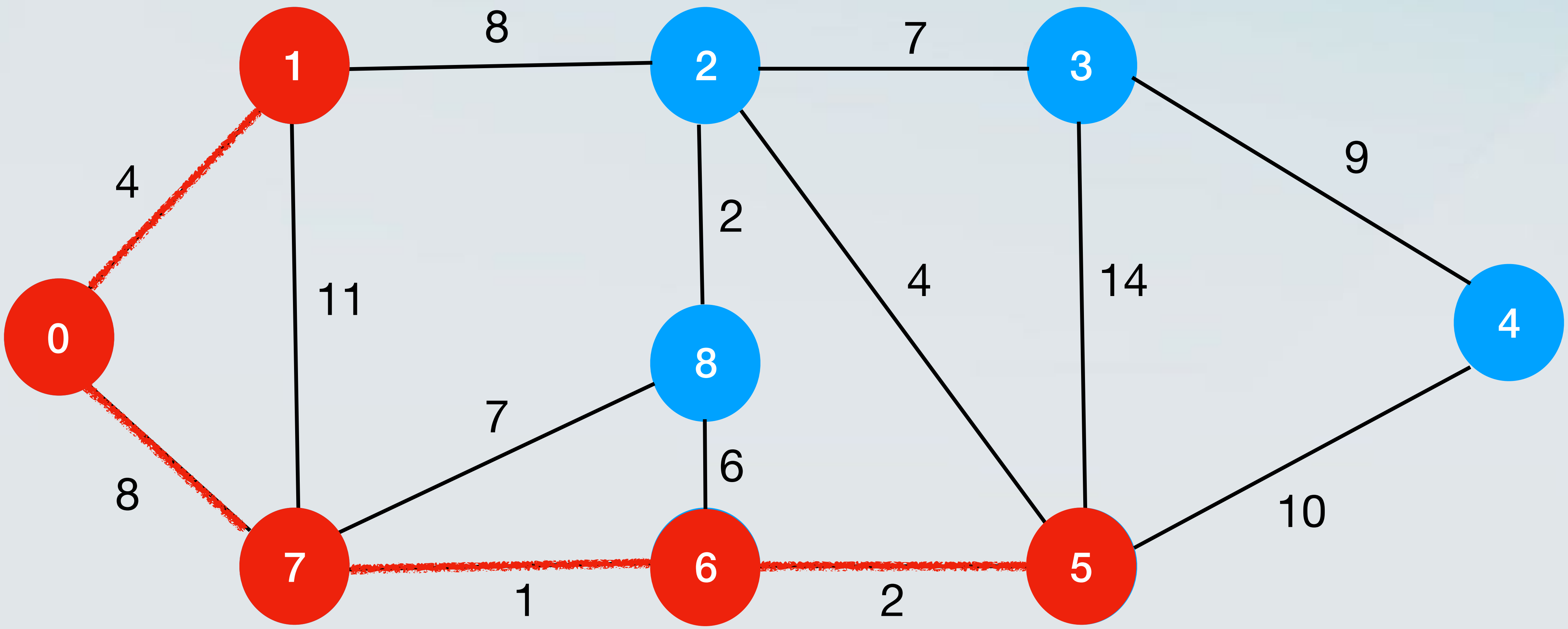
Example



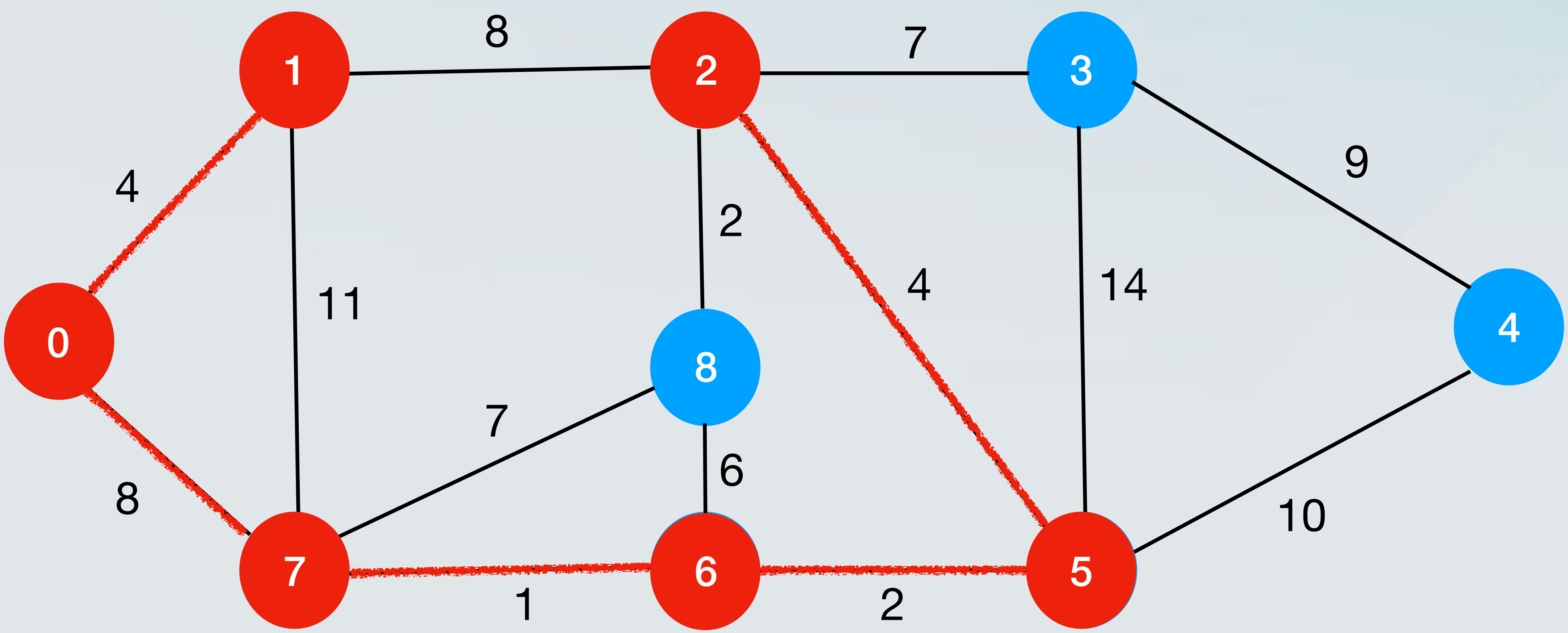
Example



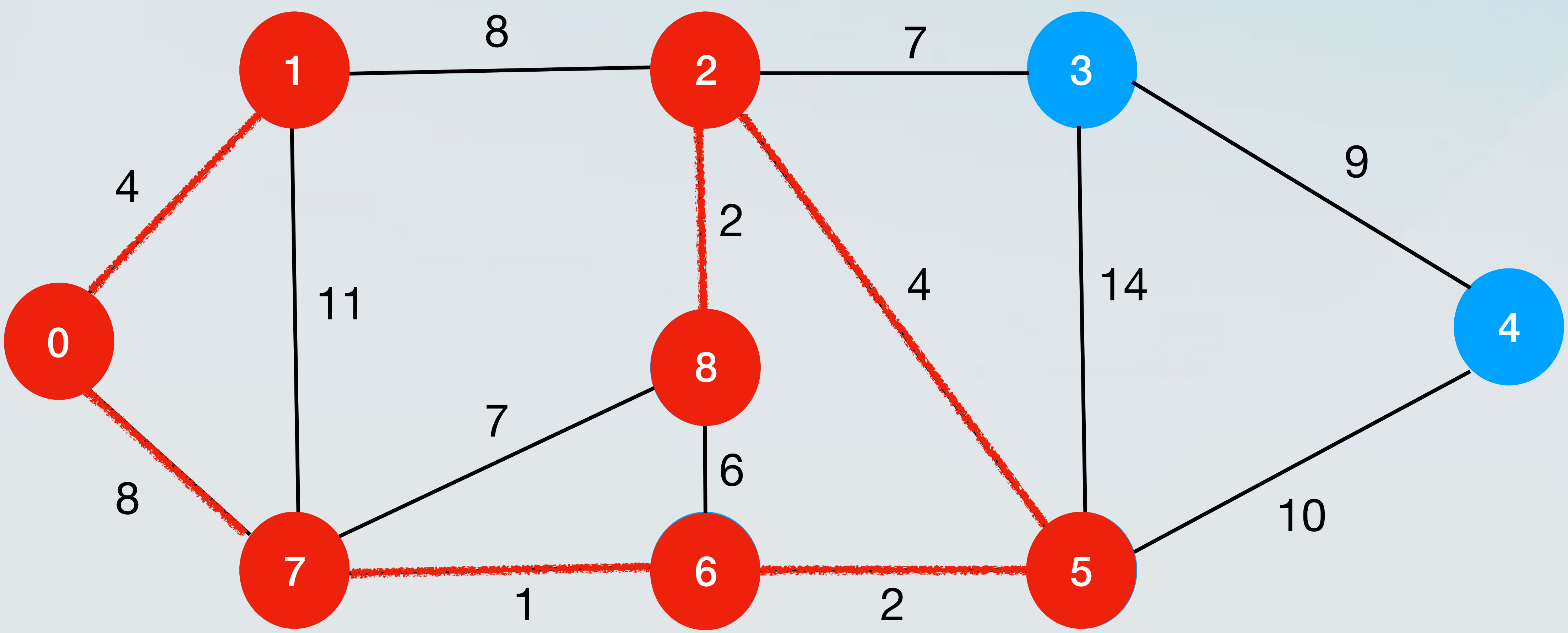
Example



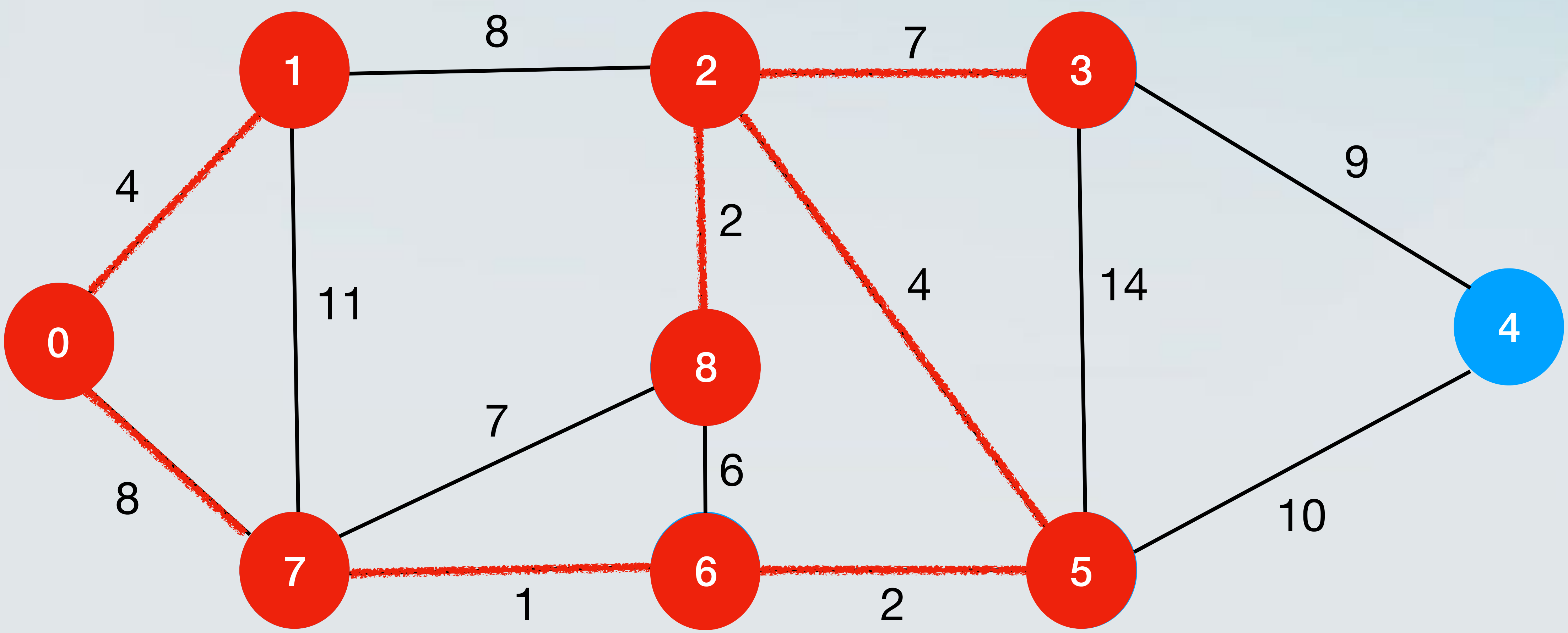
Example



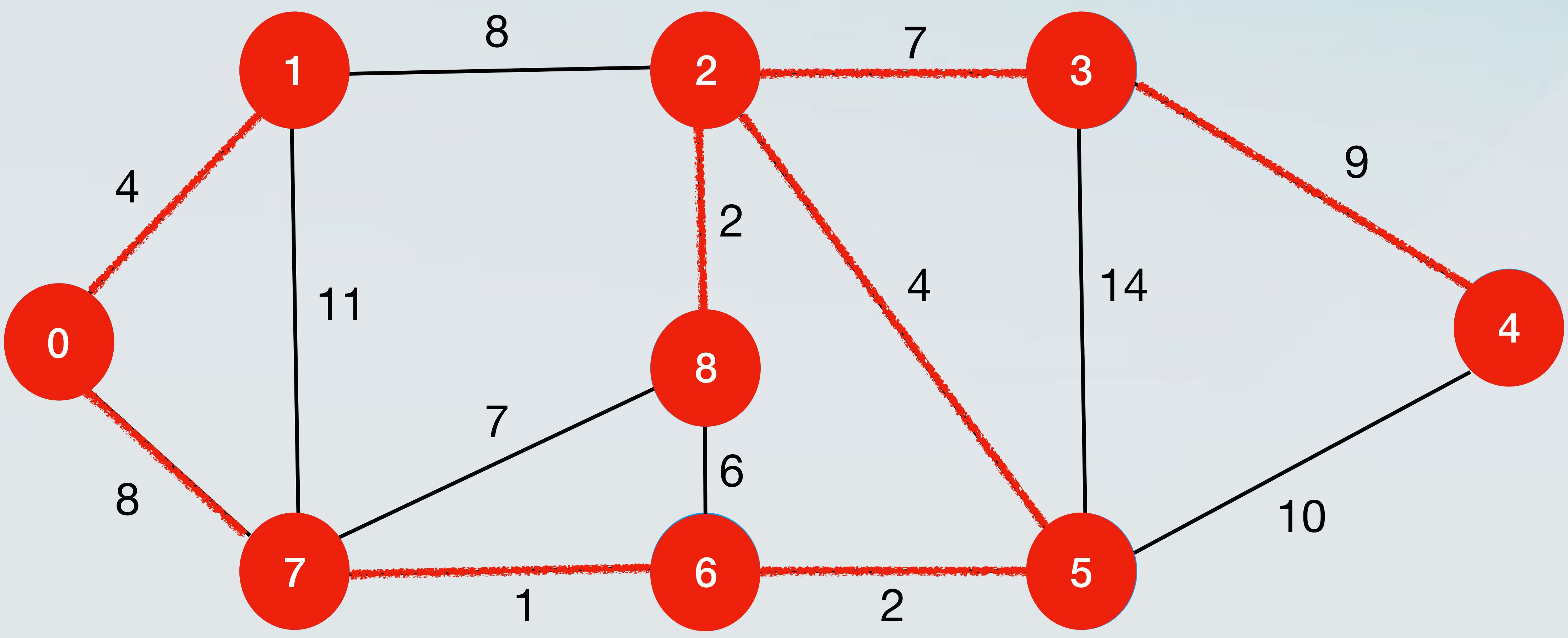
Example



Example



Example



Optimality and running time

- Optimality argued in the last lecture.
- Running time?

Prim’s algorithm running time

Prim’s algorithm running time

- We add nodes to the expanding spanning tree S.

Prim’s algorithm running time

- We add nodes to the expanding spanning tree S.
- We need to figure out which node to add next.

Prim's algorithm running time

- We add nodes to the expanding spanning tree S.
- We need to figure out which node to add next.
- We need to know the attachment cost of each node:

$a(v) = \min_{e=(u,v):u \text{ is in } S} c_e$

Prim's algorithm running time

- We add nodes to the expanding spanning tree S.
- We need to figure out which node to add next.
- We need to know the attachment cost of each node:

  $a(v) = \min_{e=(u,v):u \text{ is in } S} c_e$

- Naive solution: For every step run over all candidates.

Prim's algorithm running time

- We add nodes to the expanding spanning tree S.
- We need to figure out which node to add next.
- We need to know the attachment cost of each node:

  $a(v) = \min_{e=(u,v):u \text{ is in } S} c_e$

- Naive solution: For every step run over all candidates.
  - $O(n^2)$.

Data Structures

Data Structures

- Array

Data Structures

- Array
- Stack

Data Structures

- Array
- Stack
- Priority Queue

Priority Queue

- Maintains
  - A set of elements S.
  - A *key* **key**(v) for each element v in S.
    - The key denotes the priority of v.
- Operations:
  - **Add**(v) - with priority key.
  - **Delete**(v)
  - **Extract_Min**(v)
  - **Change_key**(v)

Priority Queue

Priority Queue

- The Priority Queue is an abstract data type.

Priority Queue

- The Priority Queue is an abstract data type.
- In reality, we have to implement it with known data structures.

Priority Queue

- The Priority Queue is an abstract data type.
- In reality, we have to implement it with known data structures.
- Many implementations exists, the usual one is with heaps.

Priority Queue

- The Priority Queue is an abstract data type.
- In reality, we have to implement it with known data structures.
- Many implementations exists, the usual one is with heaps.
- Will not be covered in the lectures.

Priority Queue

- The Priority Queue is an abstract data type.
- In reality, we have to implement it with known data structures.
- Many implementations exists, the usual one is with heaps.
- Will not be covered in the lectures.
  - Curious readers: Kleinberg and Tardos, Chapter 2.5.

Priority Queue

- The Priority Queue is an abstract data type.
- In reality, we have to implement it with known data structures.
- Many implementations exists, the usual one is with heaps.
- Will not be covered in the lectures.
    - Curious readers: Kleinberg and Tardos, Chapter 2.5.
- For now:

Priority Queue

- The Priority Queue is an abstract data type.
- In reality, we have to implement it with known data structures.
- Many implementations exists, the usual one is with heaps.
- Will not be covered in the lectures.
  - Curious readers: Kleinberg and Tardos, Chapter 2.5.
- For now:
  - PQ operations can be implemented in $O(\log n)$ time.

Prim's algorithm running time

- We add nodes to the expanding spanning tree S.
- We need to figure out which node to add next.
- We need to know the attachment cost of each node:

$$a(v) = \min_{e=(u,v):u \text{ is in } S} c_e$$

Prim's algorithm running time

- We add nodes to the expanding spanning tree S.
- We need to figure out which node to add next.
- We need to know the attachment cost of each node:

  $a(v) = \min_{e=(u,v):u \text{ is in } S} c_e$

- PQ solution: Insert the nodes in a PQ, with minus the attachment cost as the keys.

Prim's algorithm running time

- We add nodes to the expanding spanning tree S.
- We need to figure out which node to add next.
- We need to know the attachment cost of each node:

$a(v) = \min_{e=(u,v):u \text{ is in } S} c_e$

- PQ solution: Insert the nodes in a PQ, with minus the attachment cost as the keys.
    - Run **Extract_Min**(v) to find the next node.

Prim's algorithm running time

- We add nodes to the expanding spanning tree S.
- We need to figure out which node to add next.
- We need to know the attachment cost of each node:

  $a(v) = \min_{e=(u,v):u \text{ is in } S} c_e$

- PQ solution: Insert the nodes in a PQ, with minus the attachment cost as the keys.
  - Run **Extract_Min**(v) to find the next node.
  - Run **Change_key**(v) to update the attachment costs.

Prim’s algorithm running time

- PQ solution: Insert the nodes in a PQ, with minus the attachment cost as the keys.
  - Run **Extract_Min**(v) to find the next node.
    - Run n-1 times.
  - Run **Change_key**(v) to update the attachment costs.
    - Run at most once per edge.

Prim's algorithm running time

- PQ solution: Insert the nodes in a PQ, with minus the attachment cost as the keys.
  - Run **Extract_Min**(v) to find the next node.
    - Run n-1 times.
  - Run **Change_key**(v) to update the attachment costs.
    - Run at most once per edge.
  - Running time O(m log n).

Clustering (abstractly)

Clustering (abstractly)

- We have a collection of objects.

Clustering (abstractly)

- We have a collection of objects.
- They have different degrees of similarity.

Clustering (abstractly)

- We have a collection of objects.
- They have different degrees of similarity.
- We want to organise them into coherent groups.

Clustering (abstractly)

- We have a collection of objects.
- They have different degrees of similarity.
- We want to organise them into coherent groups.
  - Objects in a group exhibit high similarity.

Clustering (abstractly)

- We have a collection of objects.
- They have different degrees of similarity.
- We want to organise them into coherent groups.
  - Objects in a group exhibit high similarity.
- Applications: Many, e.g., machine learning.

Clustering (abstractly)

Clustering (abstractly)

- There is a notion of distance between objects.

Clustering (abstractly)

- There is a notion of distance between objects.
- Could be physical distance (e.g., distance between houses).

Clustering (abstractly)

- There is a notion of distance between objects.
- Could be physical distance (e.g., distance between houses).
- Could be more abstract.

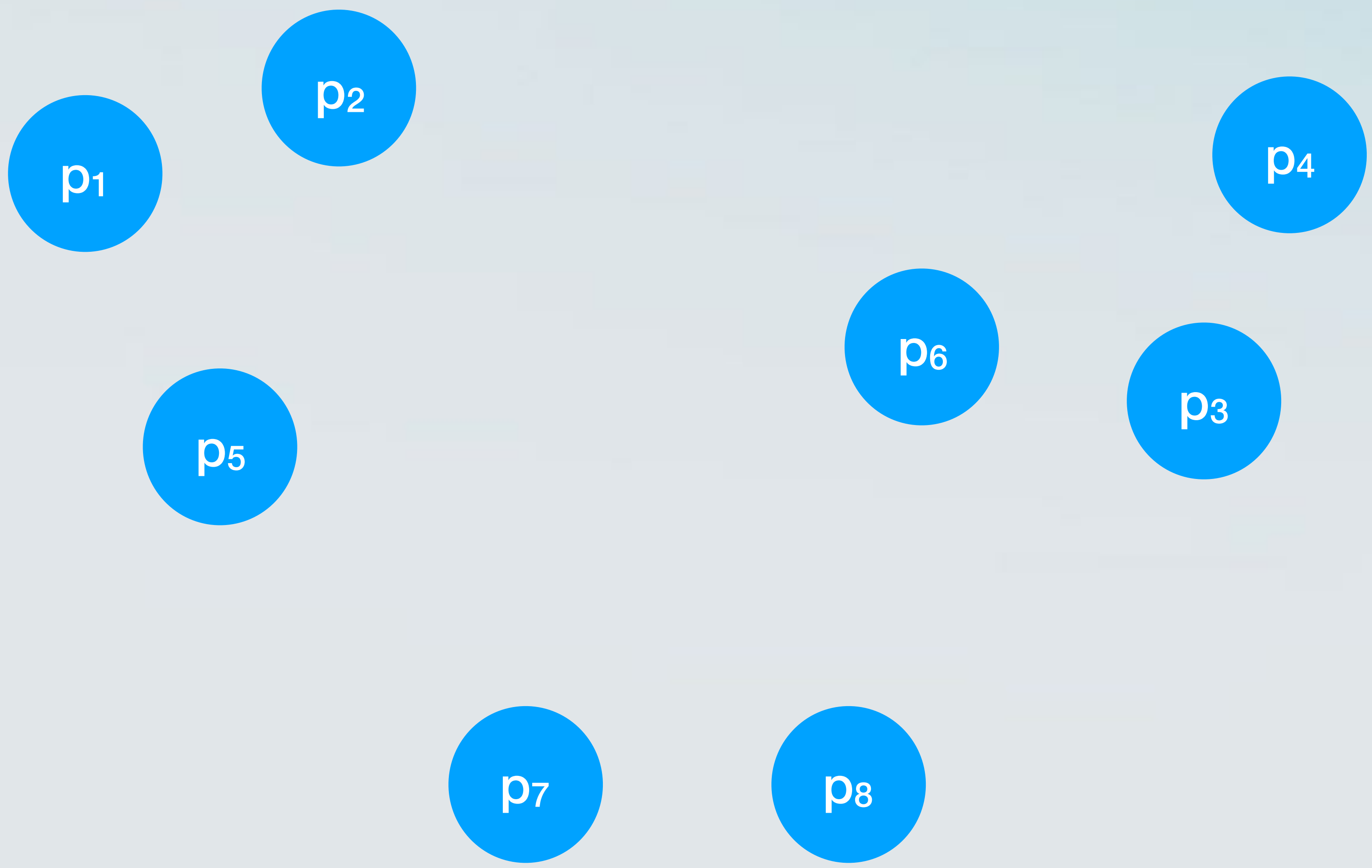
Clustering (abstractly)

- There is a notion of distance between objects.
- Could be physical distance (e.g., distance between houses).
- Could be more abstract.
  - E.g., age, height, nationality.

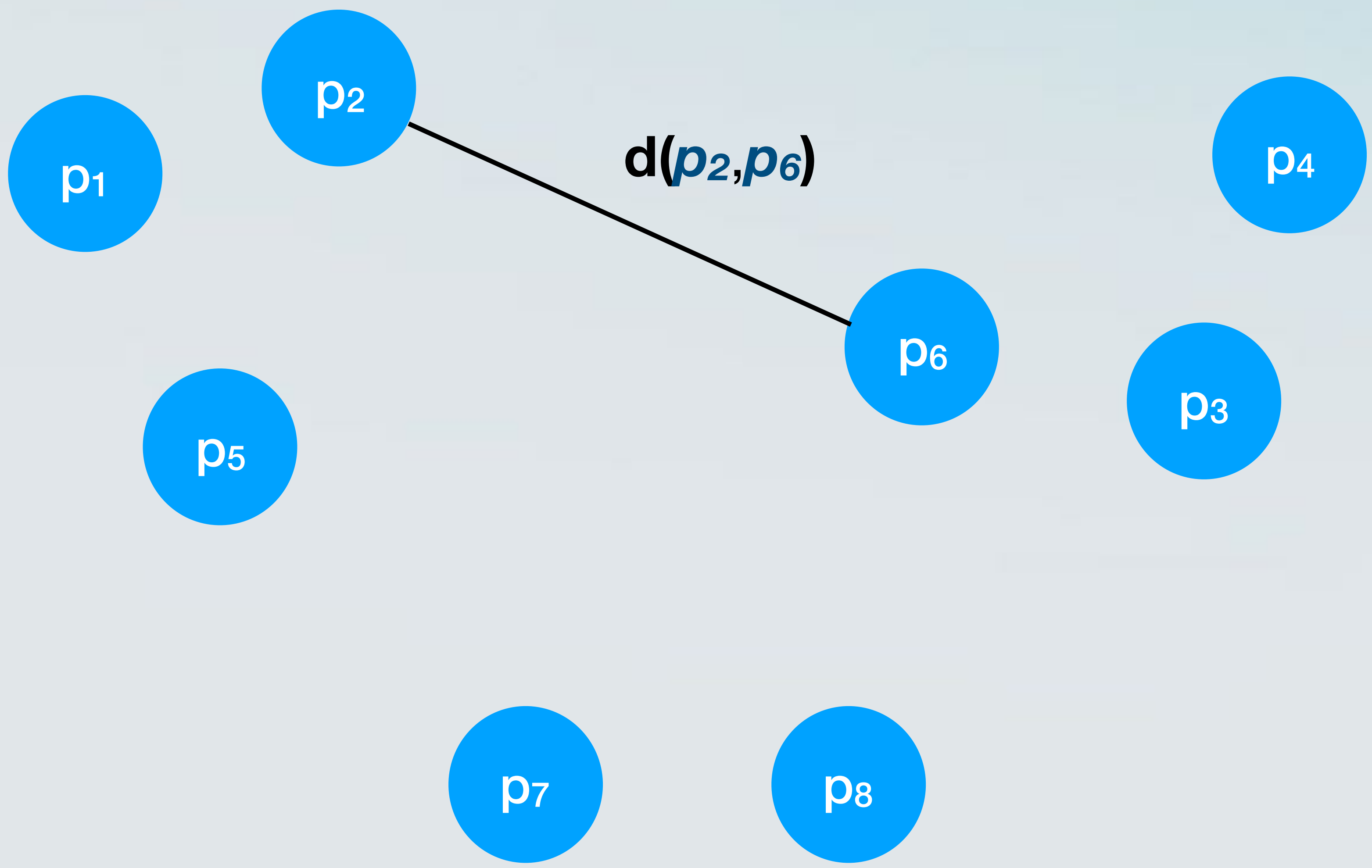
Clustering (abstractly)

- There is a notion of distance between objects.
- Could be physical distance (e.g., distance between houses).
- Could be more abstract.
  - E.g., age, height, nationality.
  - E.g., running time, algorithmic principle.

Clustering (concretely)



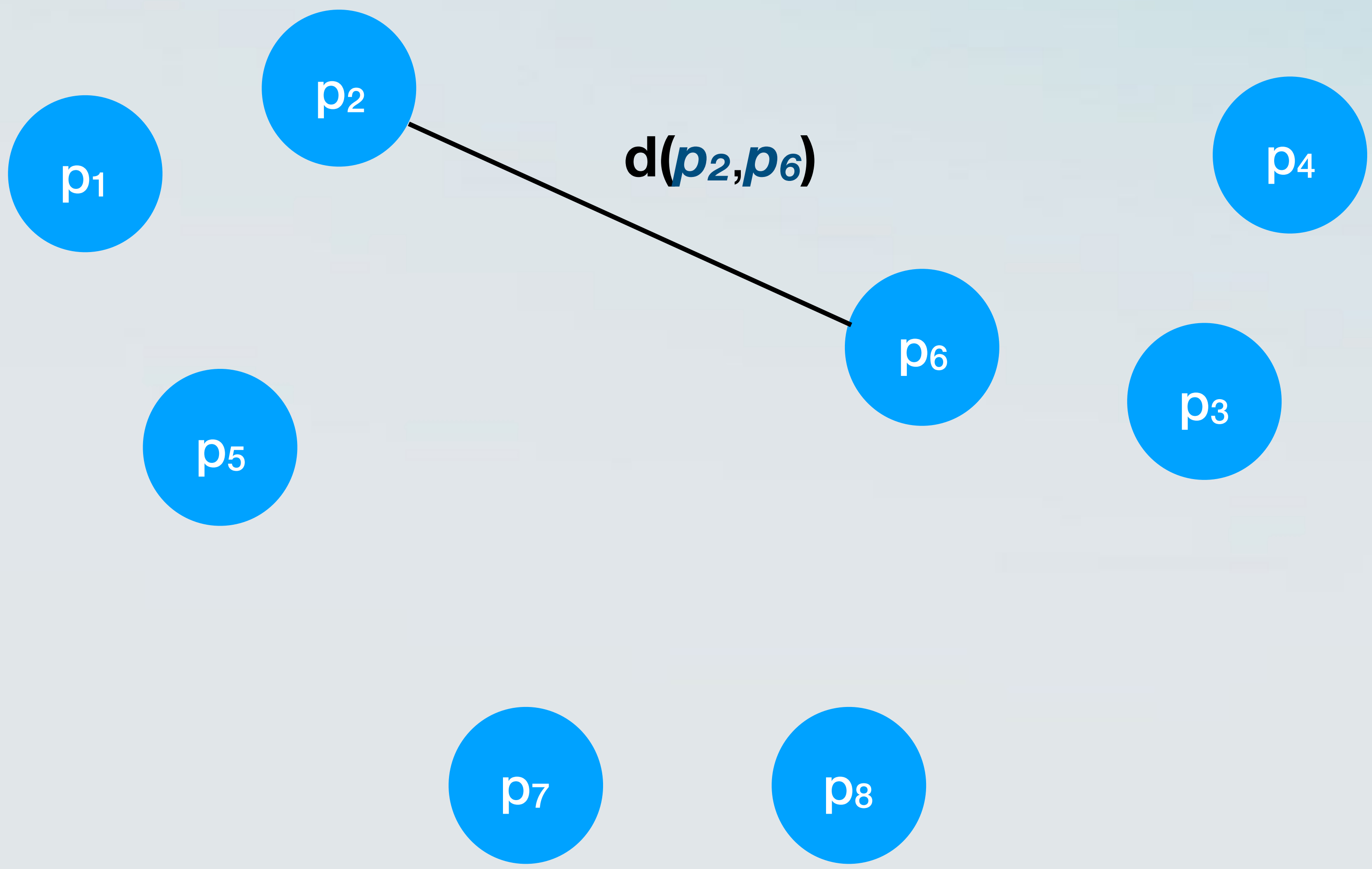
Clustering (concretely)



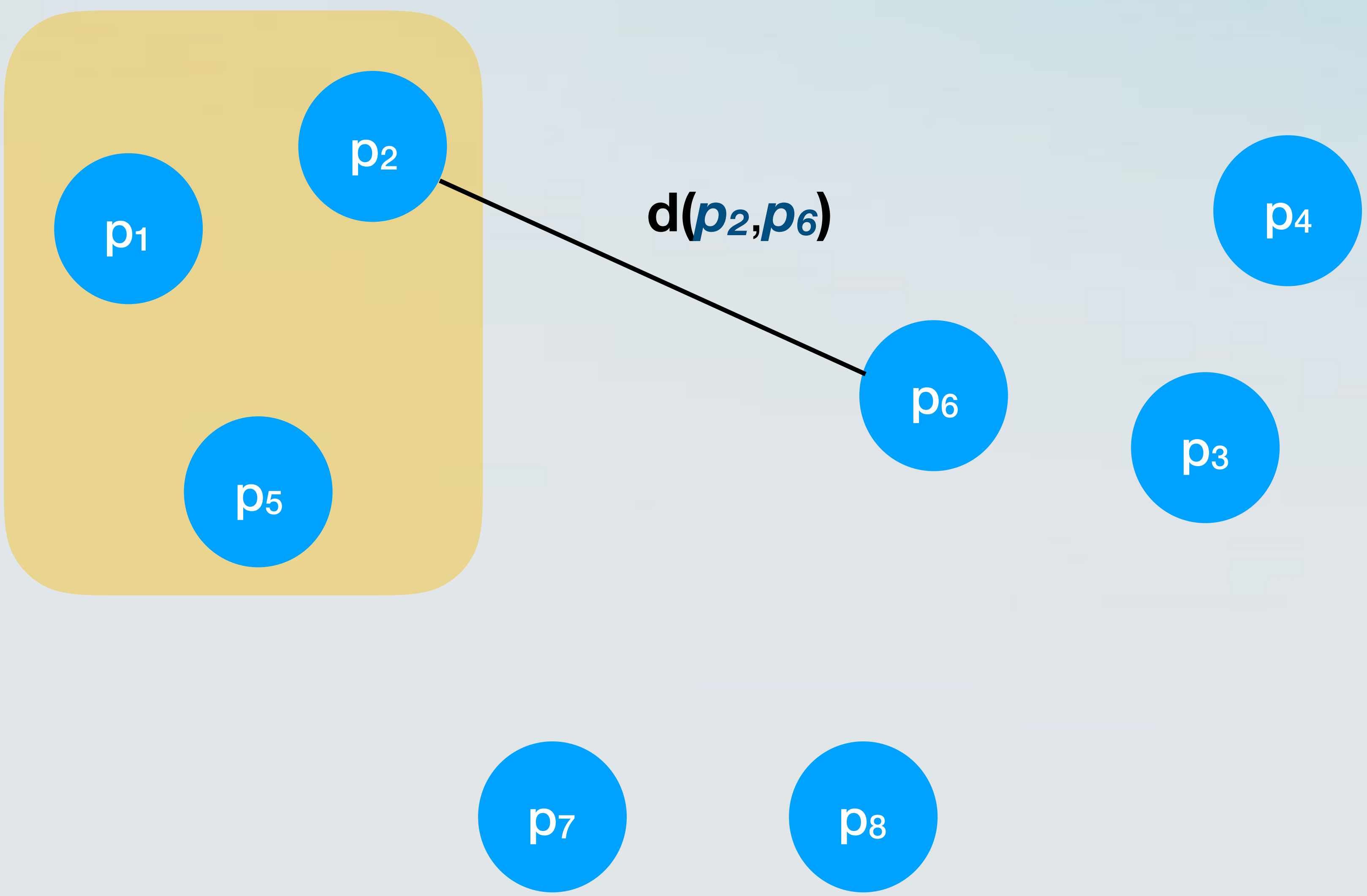
Properties of distance

- $d(p_i,p_i)=0$ for any $i = 1, \ldots , n$
- $d(p_i,p_j)>0$ for any $i \neq j$
- $d(p_i,p_j)=d(p_j,p_i)$ for any i, j

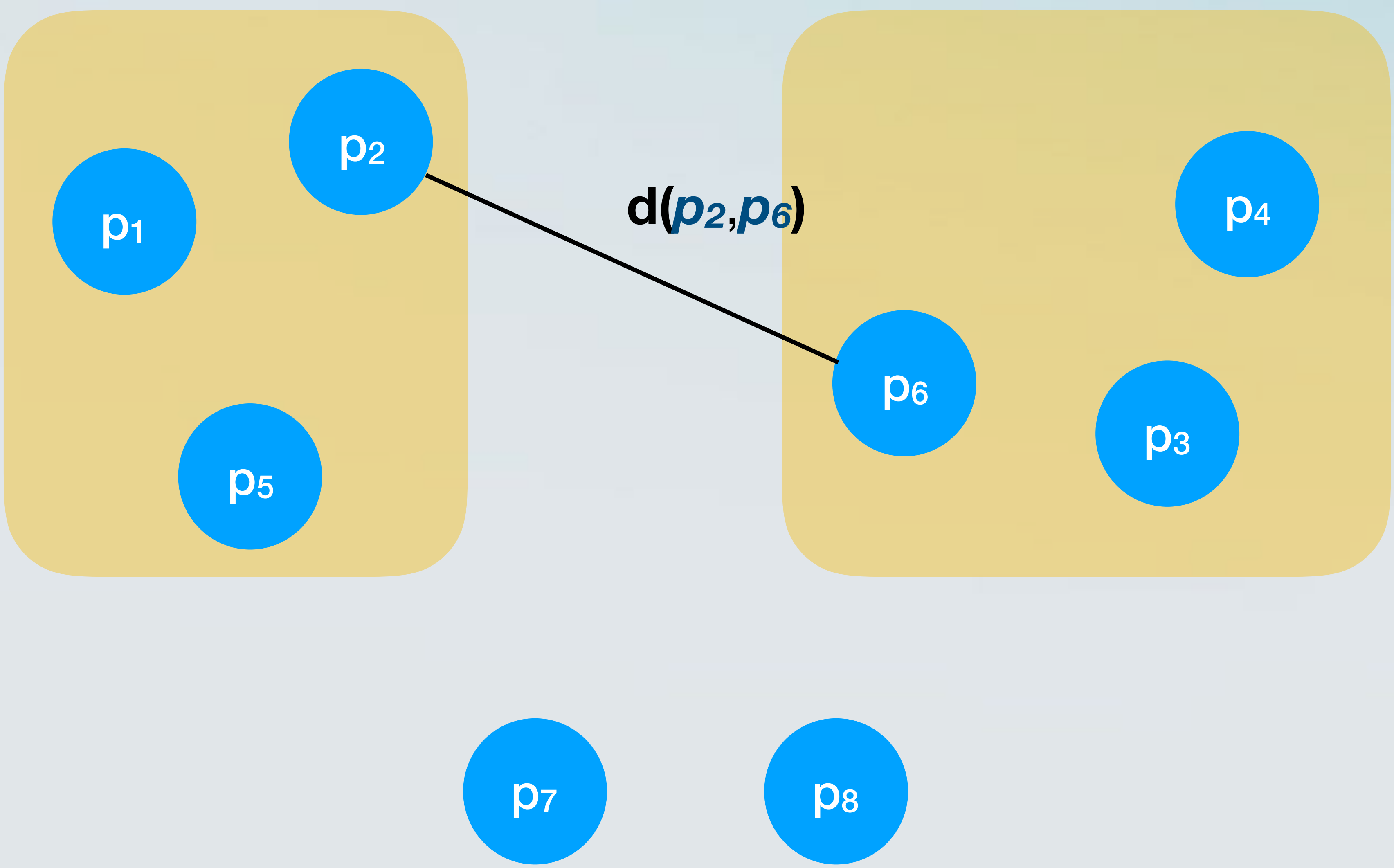
Clustering (concretely)



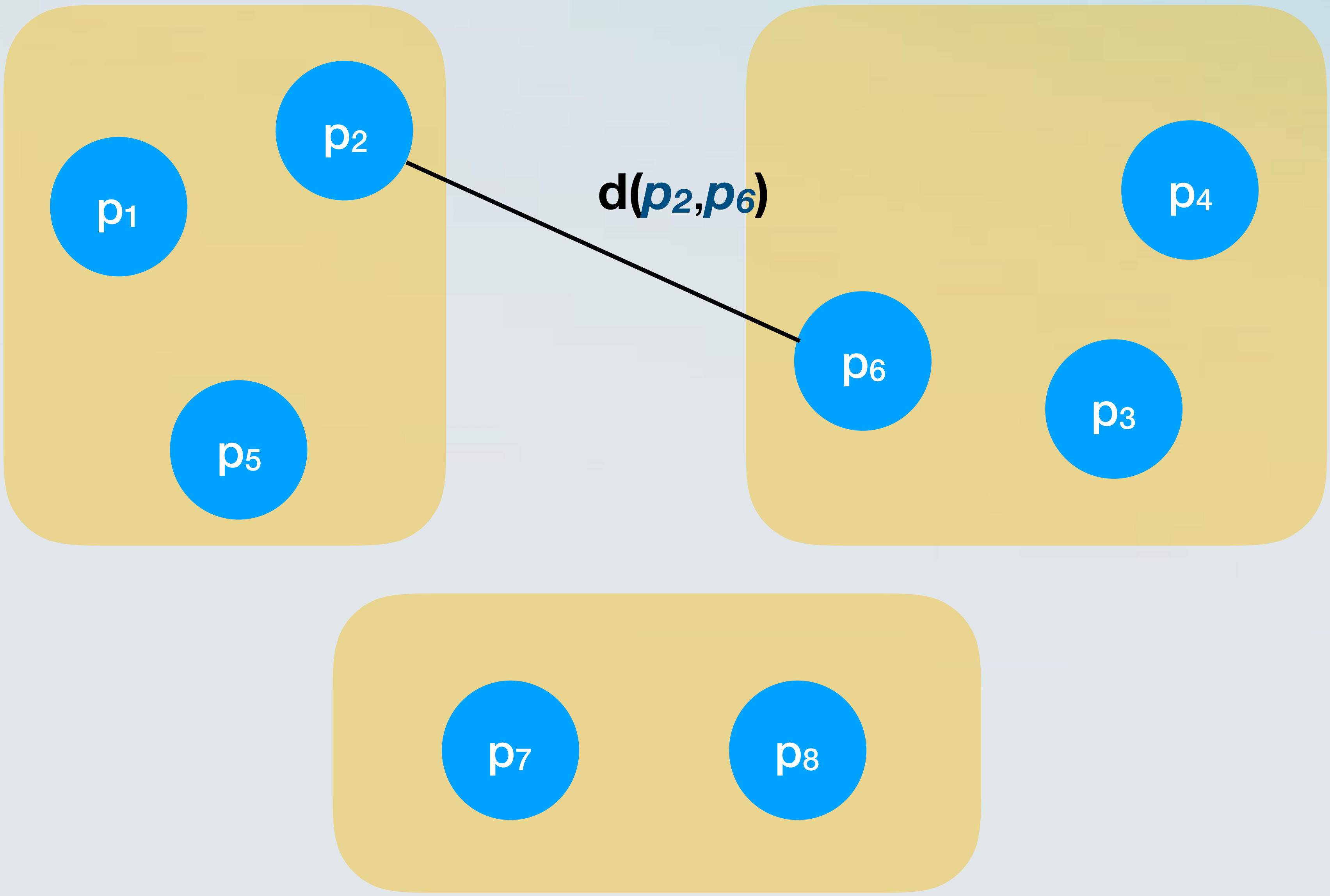
Clustering (concretely)



Clustering (concretely)



Clustering (concretely)



Clustering

Clustering

- Definition: Given a set U of n elements, a *k-clustering* of U is a partition of U into non-empty sets C_1, …, C_k.

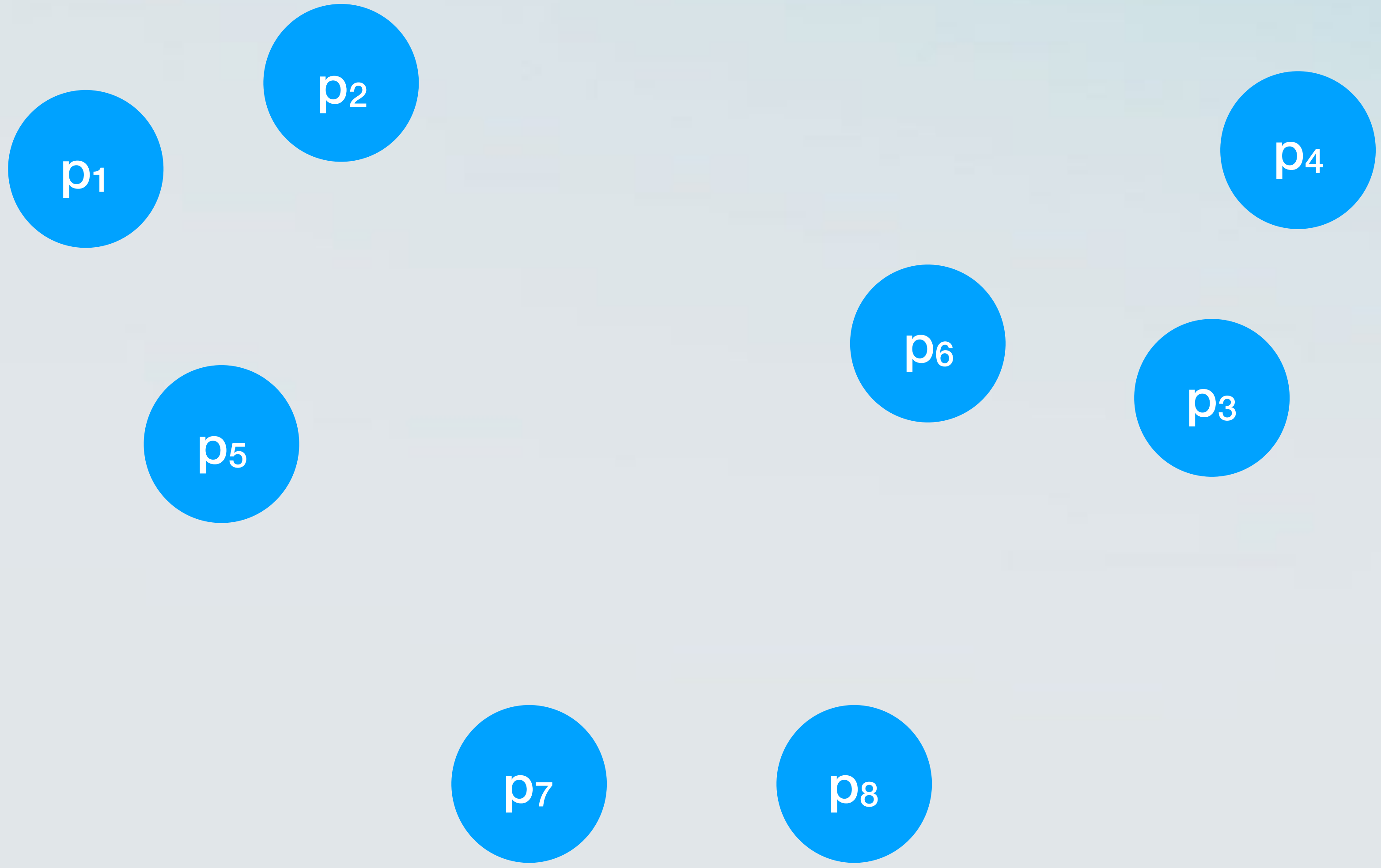
Clustering

- Definition: Given a set U of n elements, a *k-clustering* of U is a partition of U into non-empty sets $C_1, \ldots, C_k$.
- Definition: The *spacing* of a k-clustering is the minimum distance between any pair of points in different clusters.

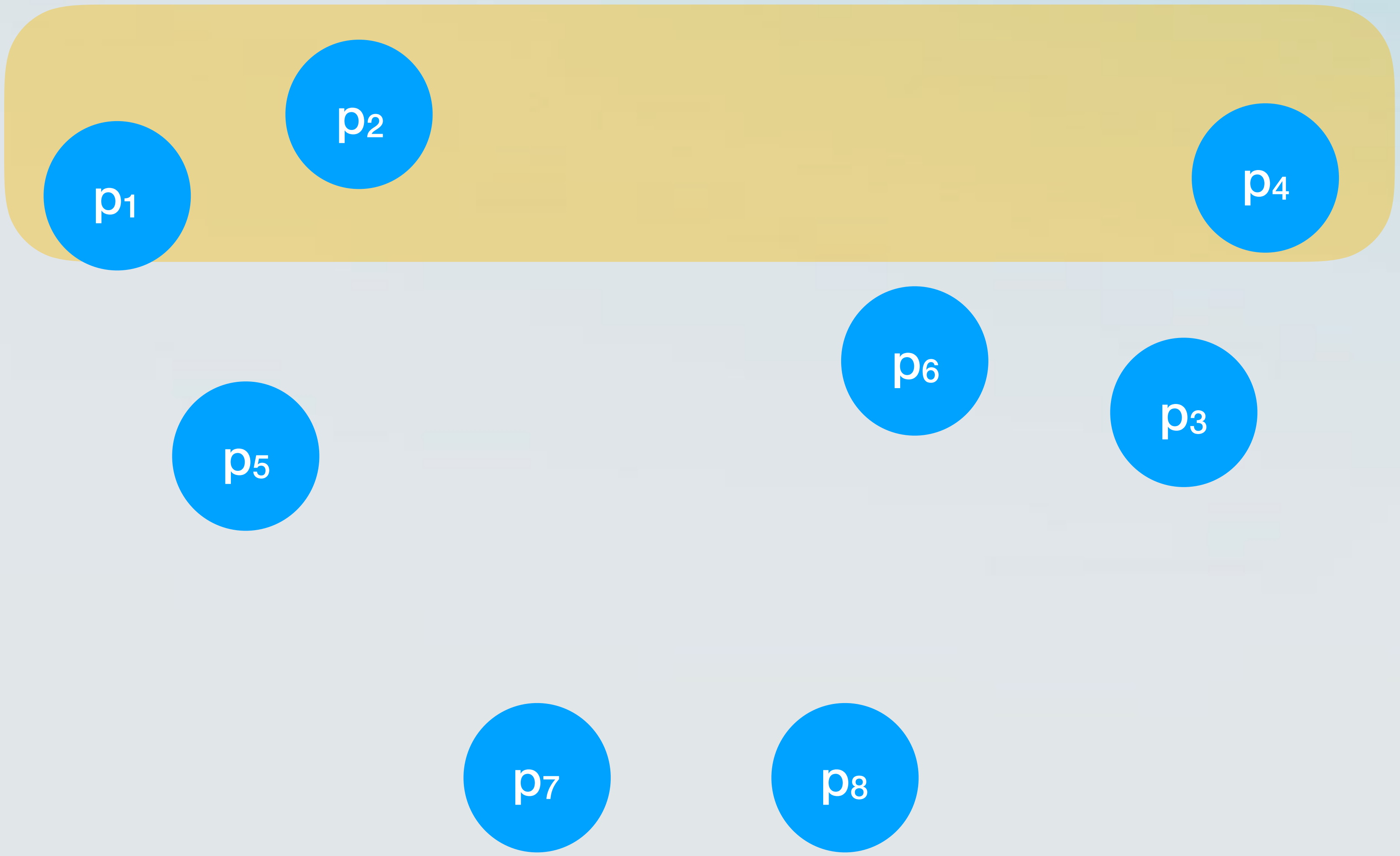
Clustering

- Definition: Given a set U of n elements, a *k-clustering* of U is a partition of U into non-empty sets $C_1, \ldots, C_k$.
- Definition: The *spacing* of a k-clustering is the minimum distance between any pair of points in different clusters.
- Goal: Among all possible k-clusterings, find one with the *maximum possible spacing*.

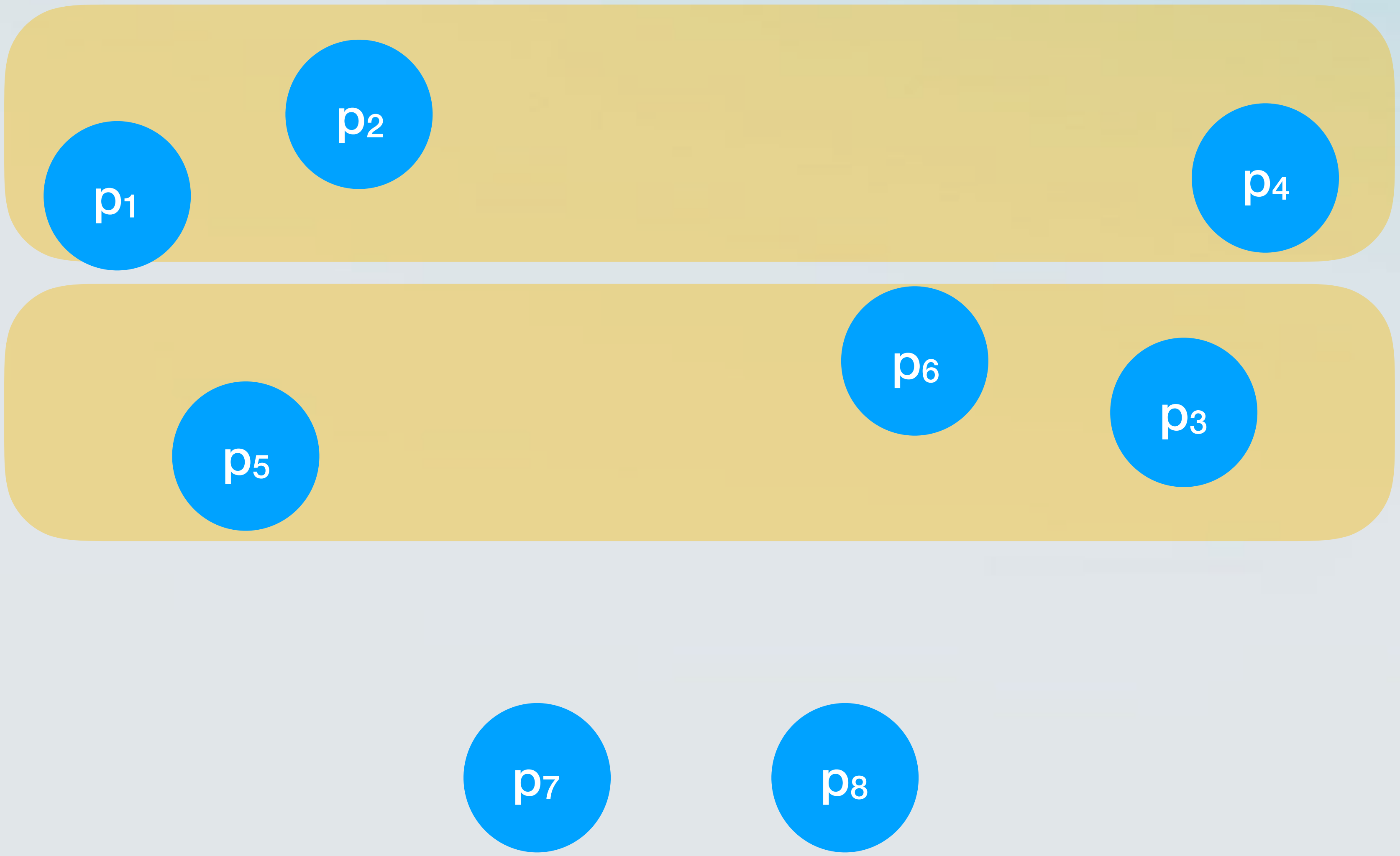
Clustering (concretely)



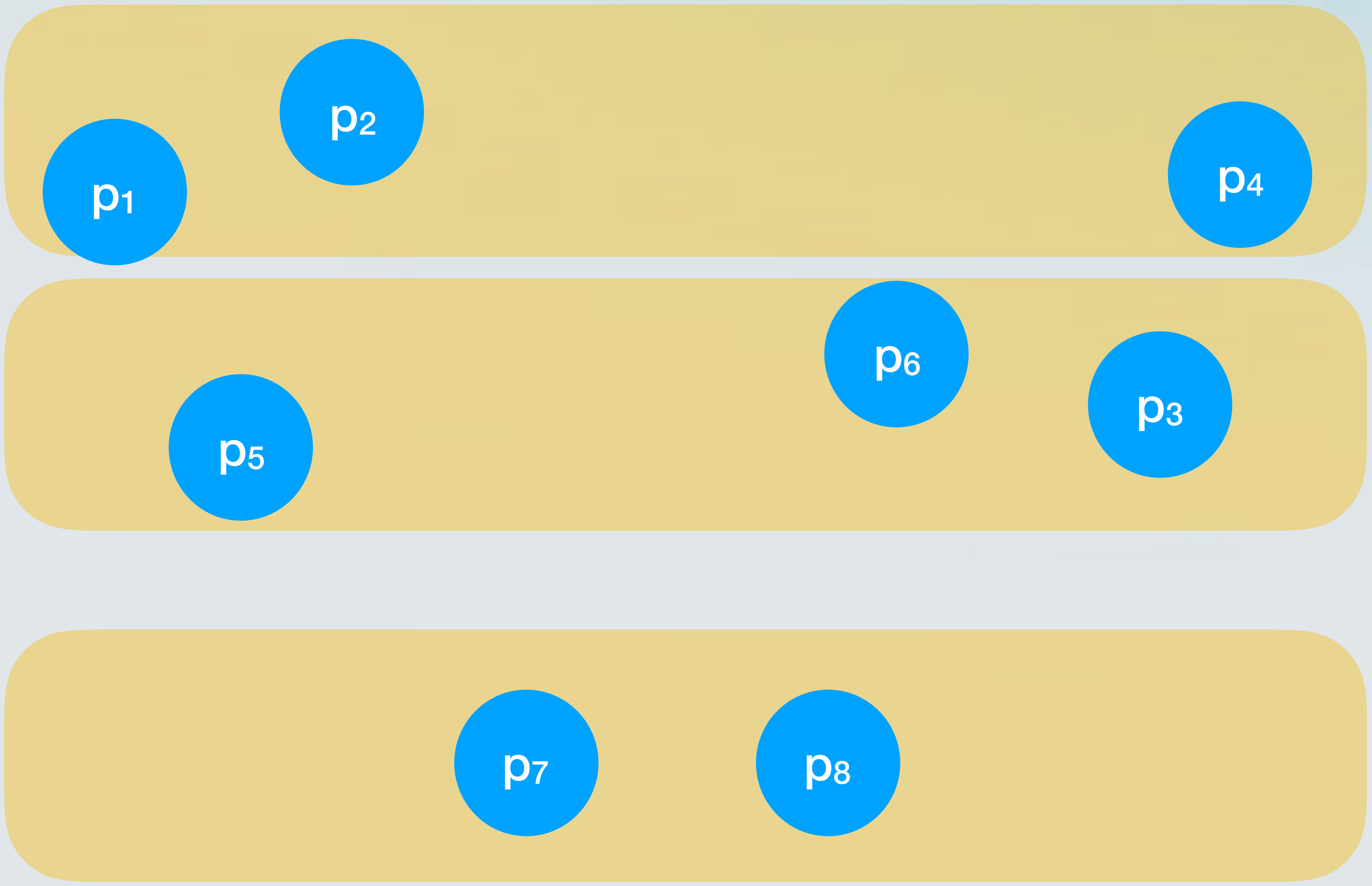
Clustering (concretely)



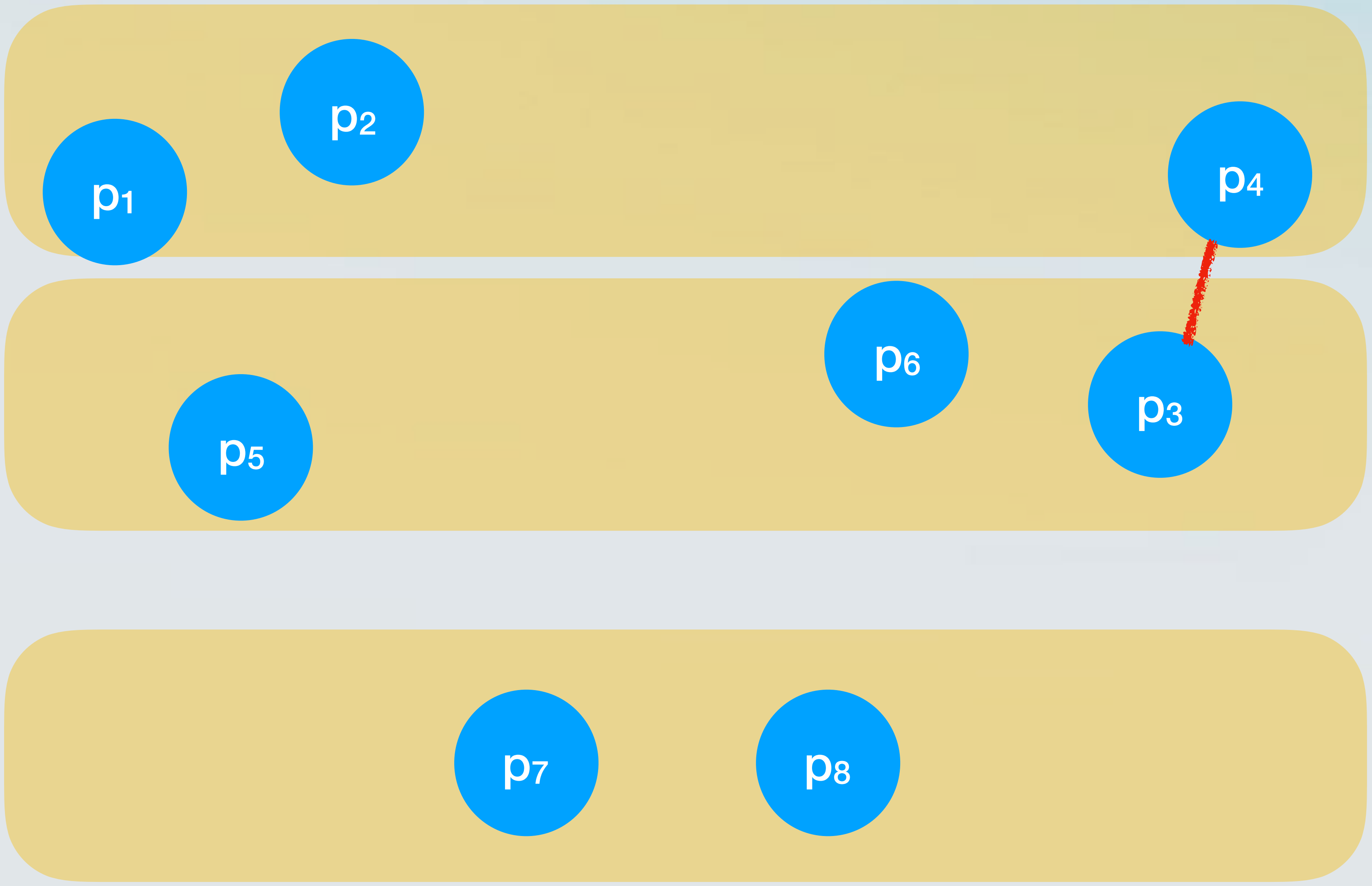
Clustering (concretely)



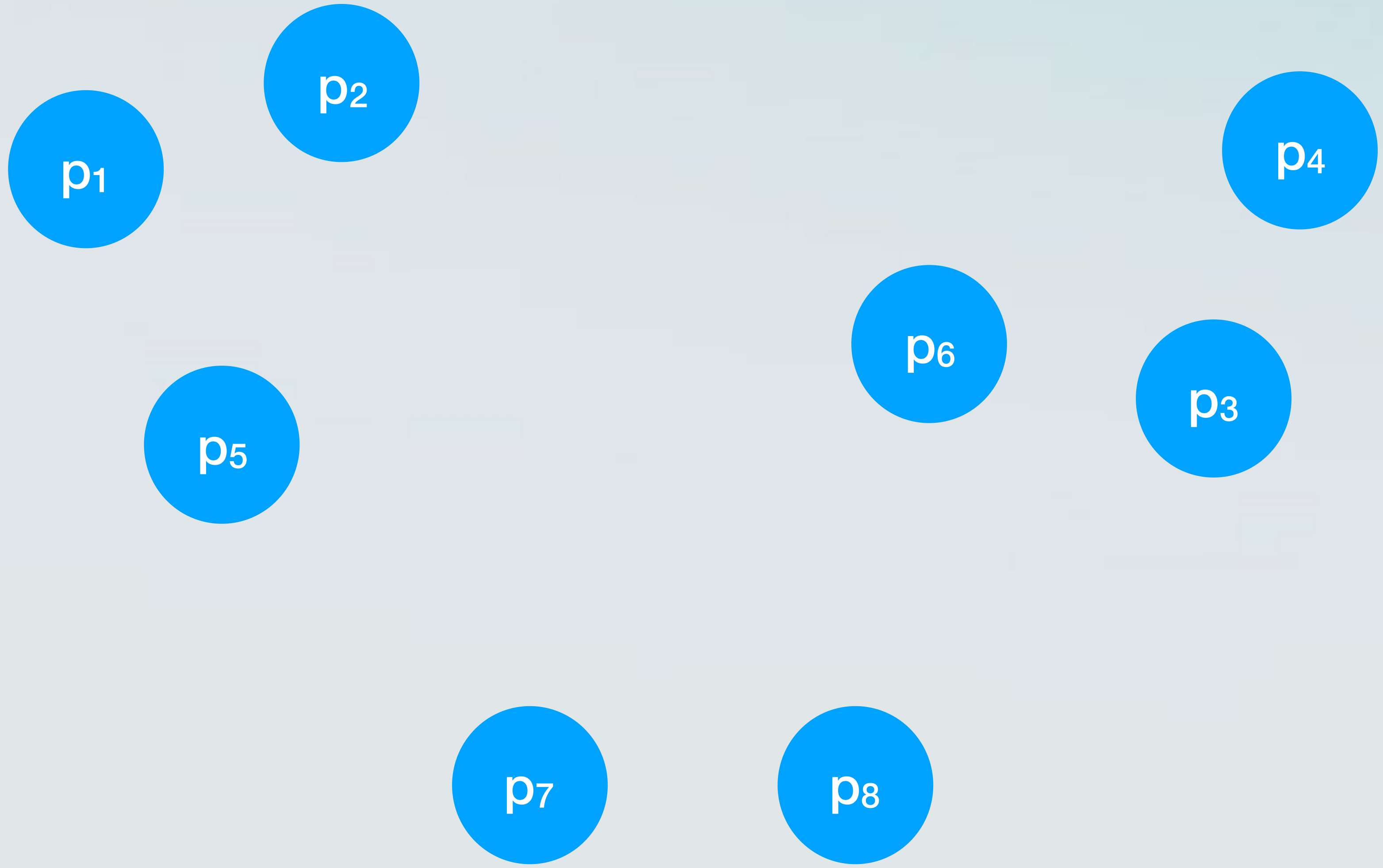
Clustering (concretely)



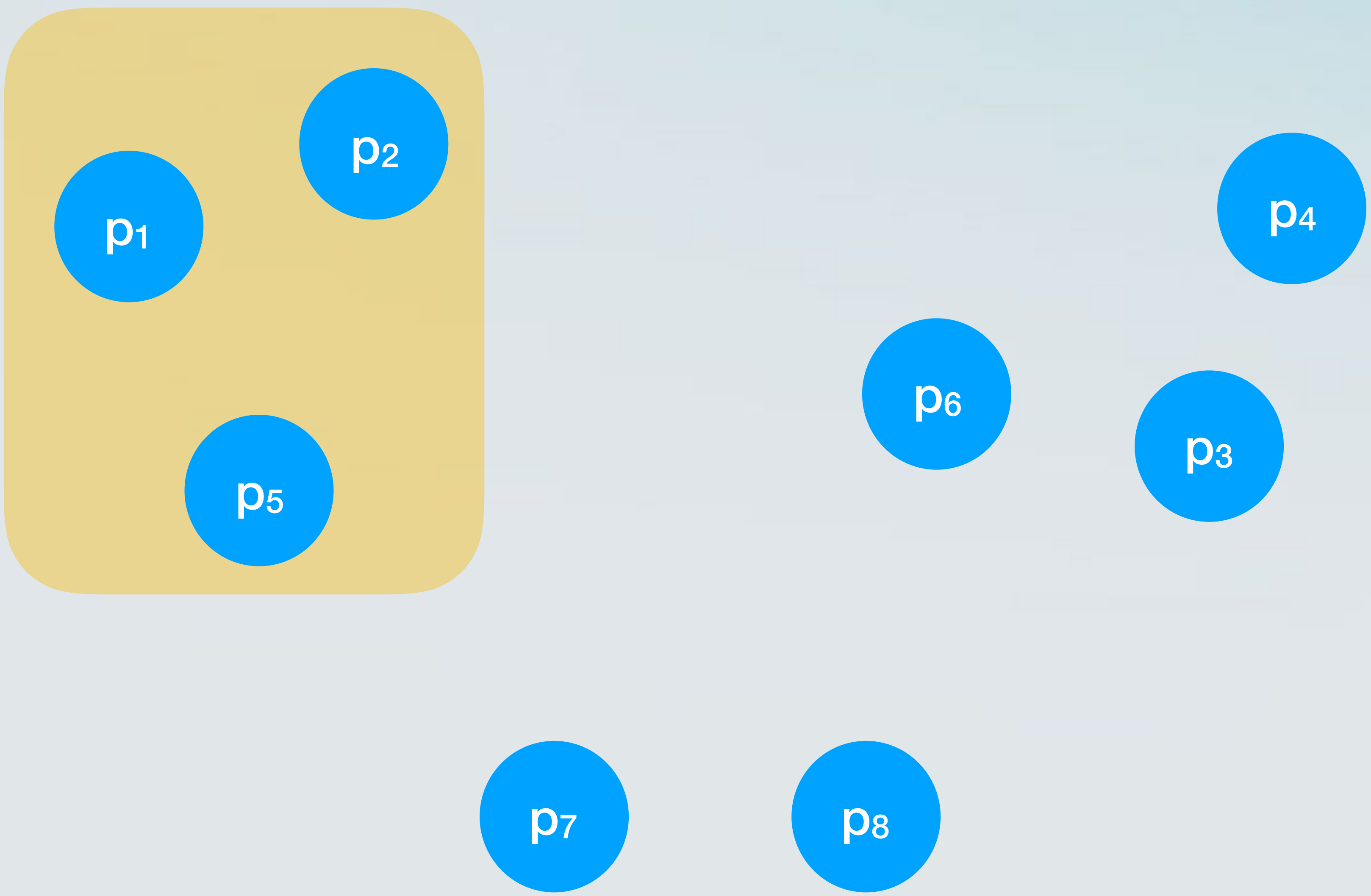
Clustering (concretely)



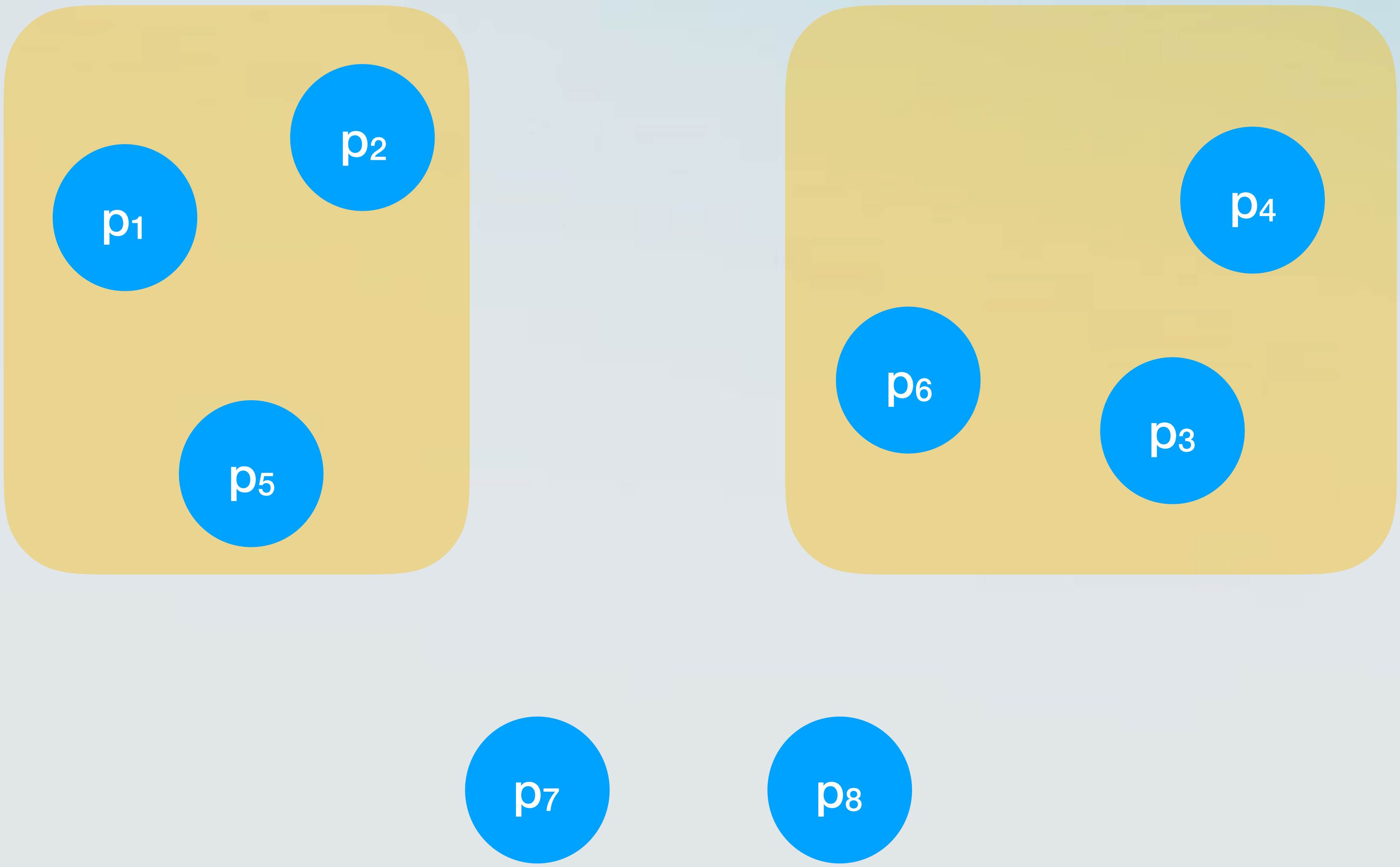
Clustering (concretely)



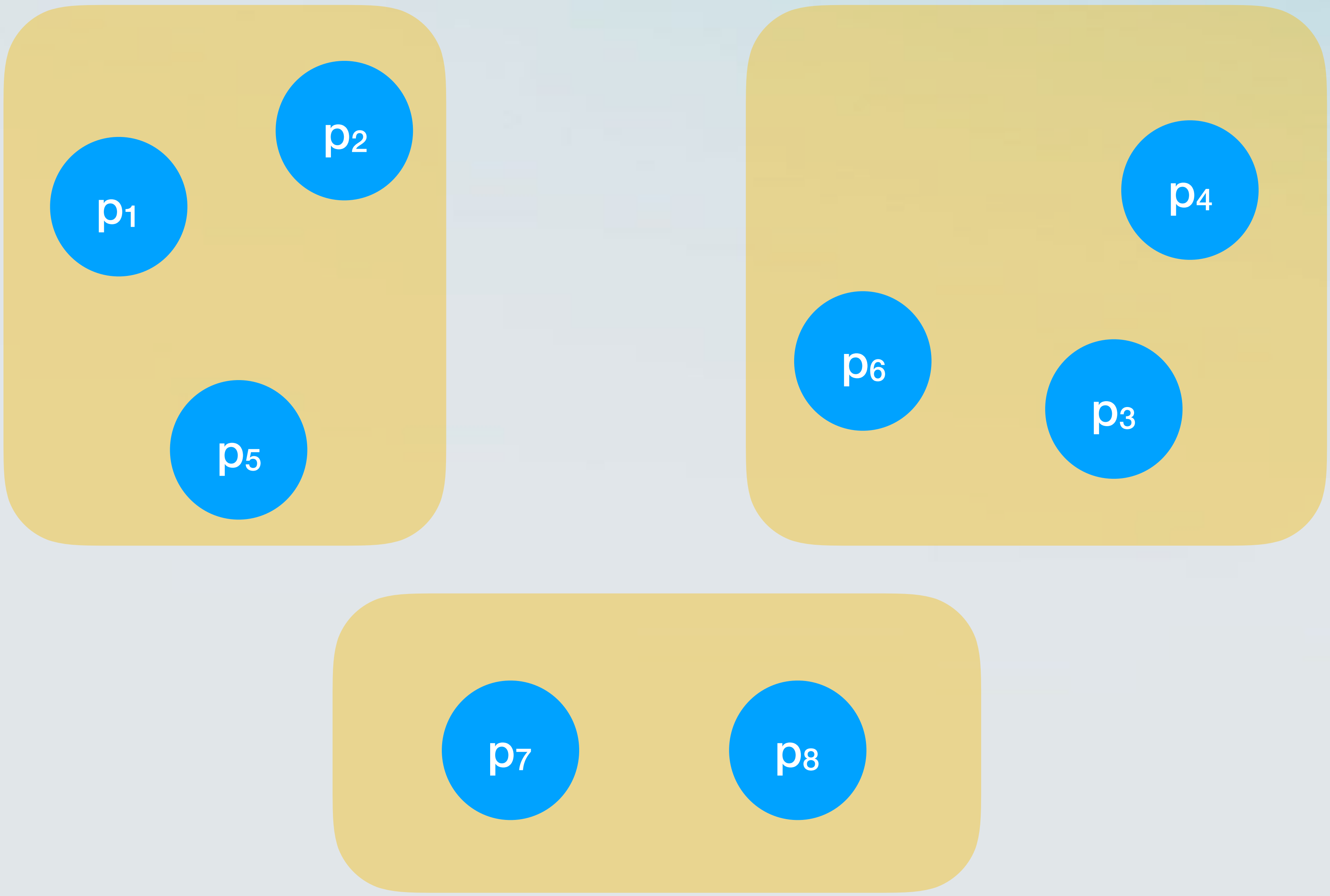
Clustering (concretely)



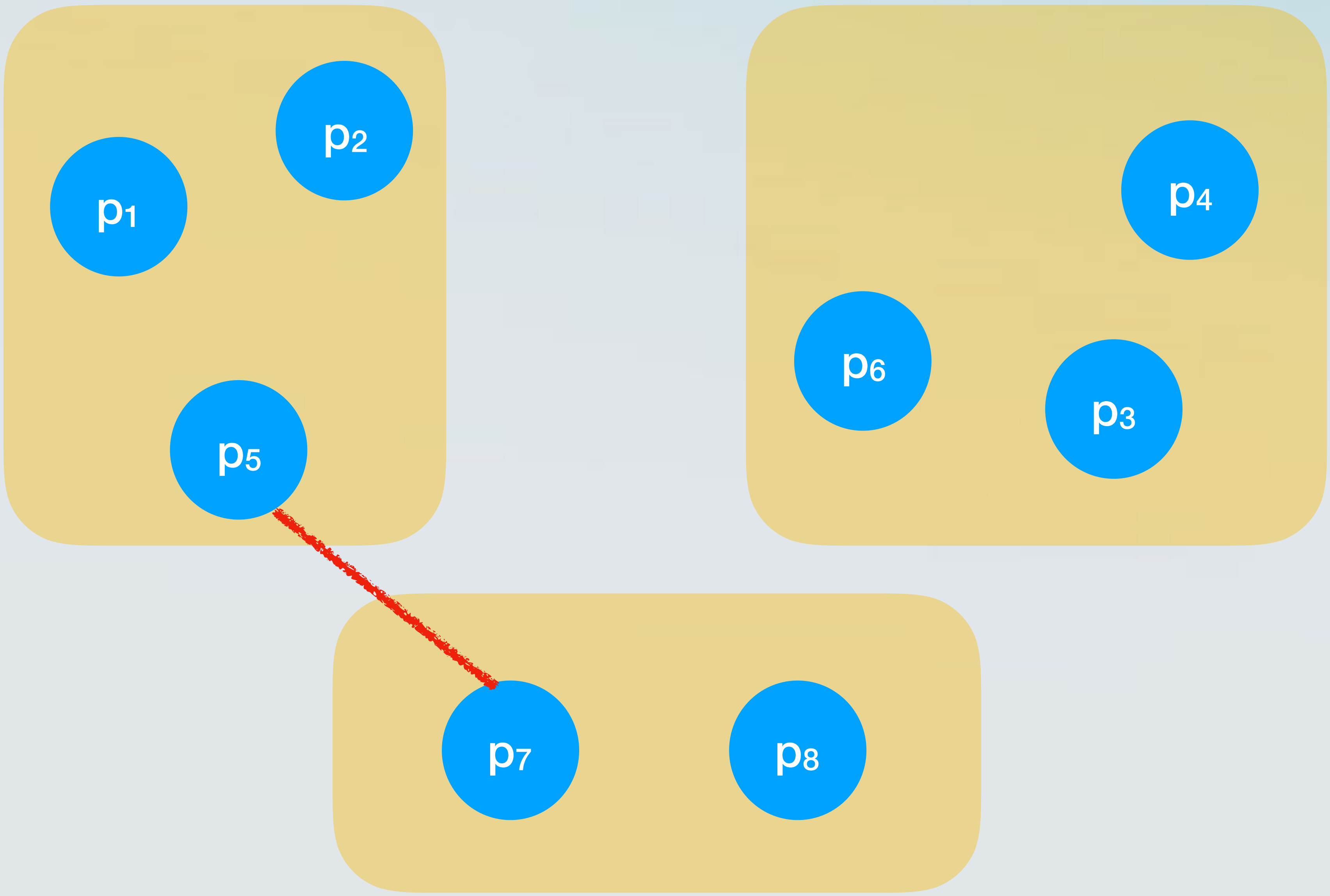
Clustering (concretely)



Clustering (concretely)



Clustering (concretely)



A greedy algorithm

A greedy algorithm

- Pick two objects p_i and p_j with the smallest distance d(p_i,p_j).

A greedy algorithm

- Pick two objects p_i and p_j with the smallest distance d(p_i,p_j).
- Connect them with an edge e=(p_i,p_j).

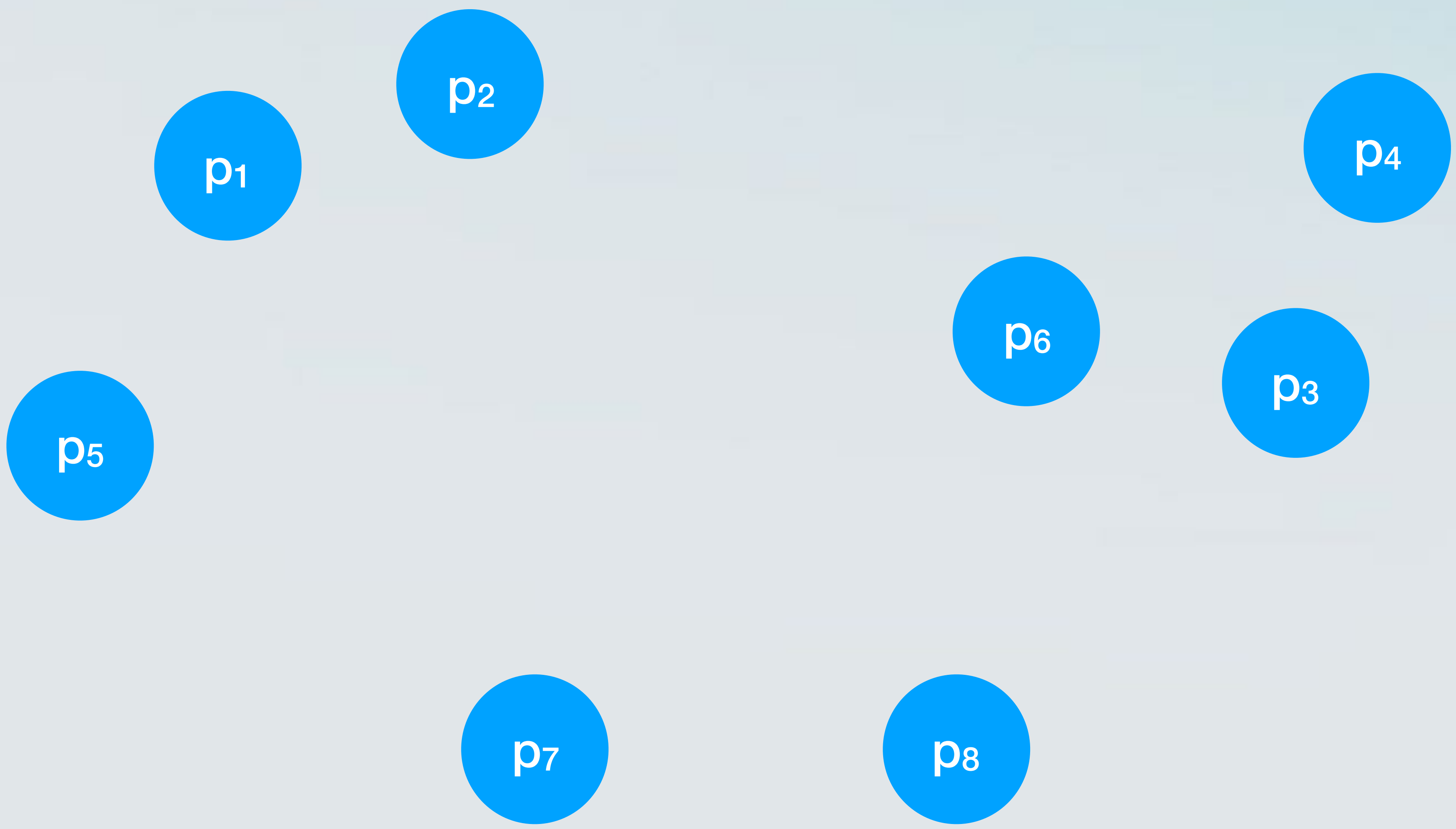
A greedy algorithm

- Pick two objects p_i and p_j with the smallest distance d(p_i,p_j).
- Connect them with an edge $e=(p_i,p_j)$.
- Continue like this until we obtain k clusters.

A greedy algorithm

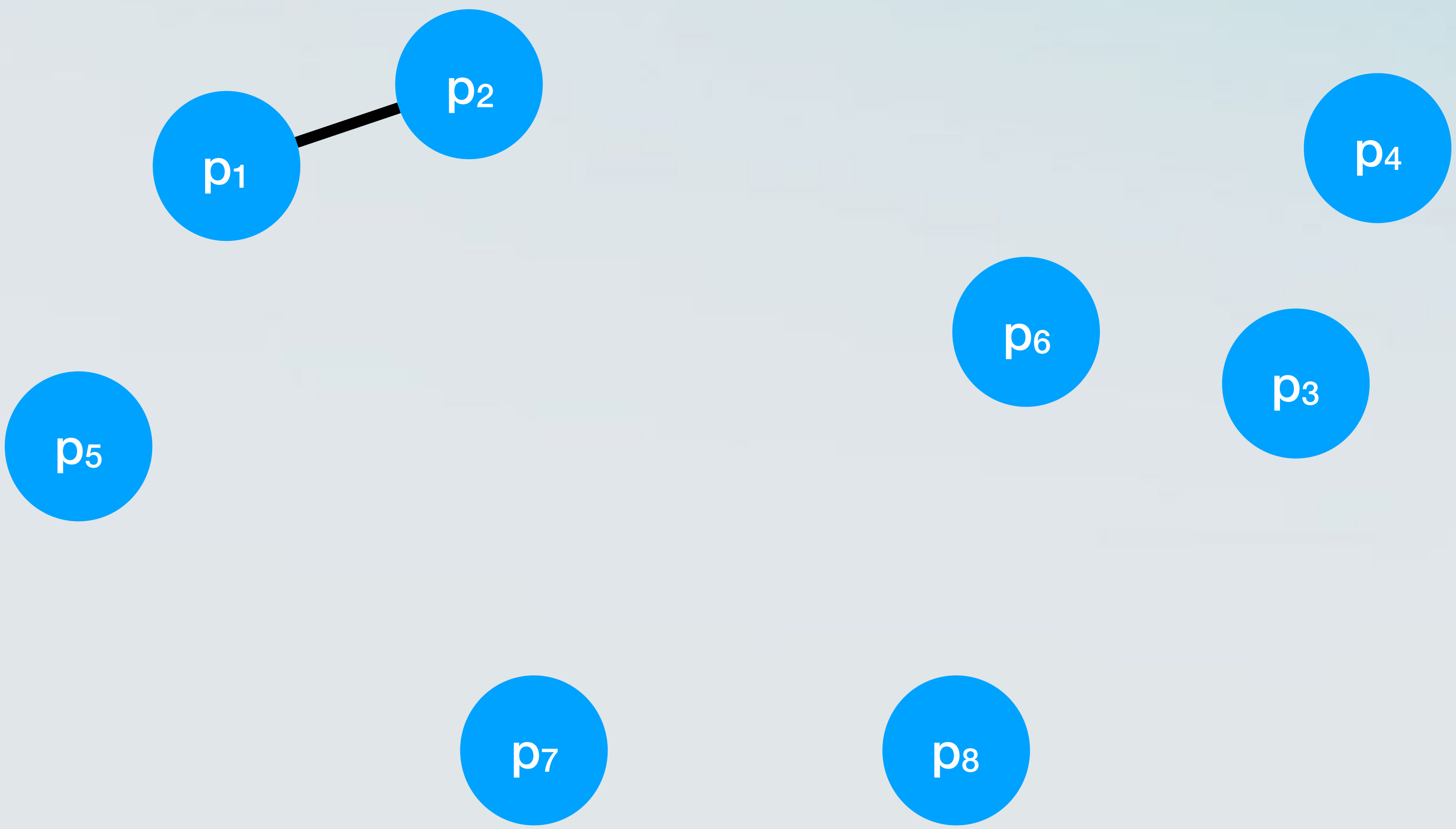
- Pick two objects p_i and p_j with the smallest distance d(p_i,p_j).
- Connect them with an edge e=(p_i,p_j).
- Continue like this until we obtain k clusters.
- If the edge e under consideration connects two objects p_i and p_j already in the same component, skip it.

Clustering (concretely)



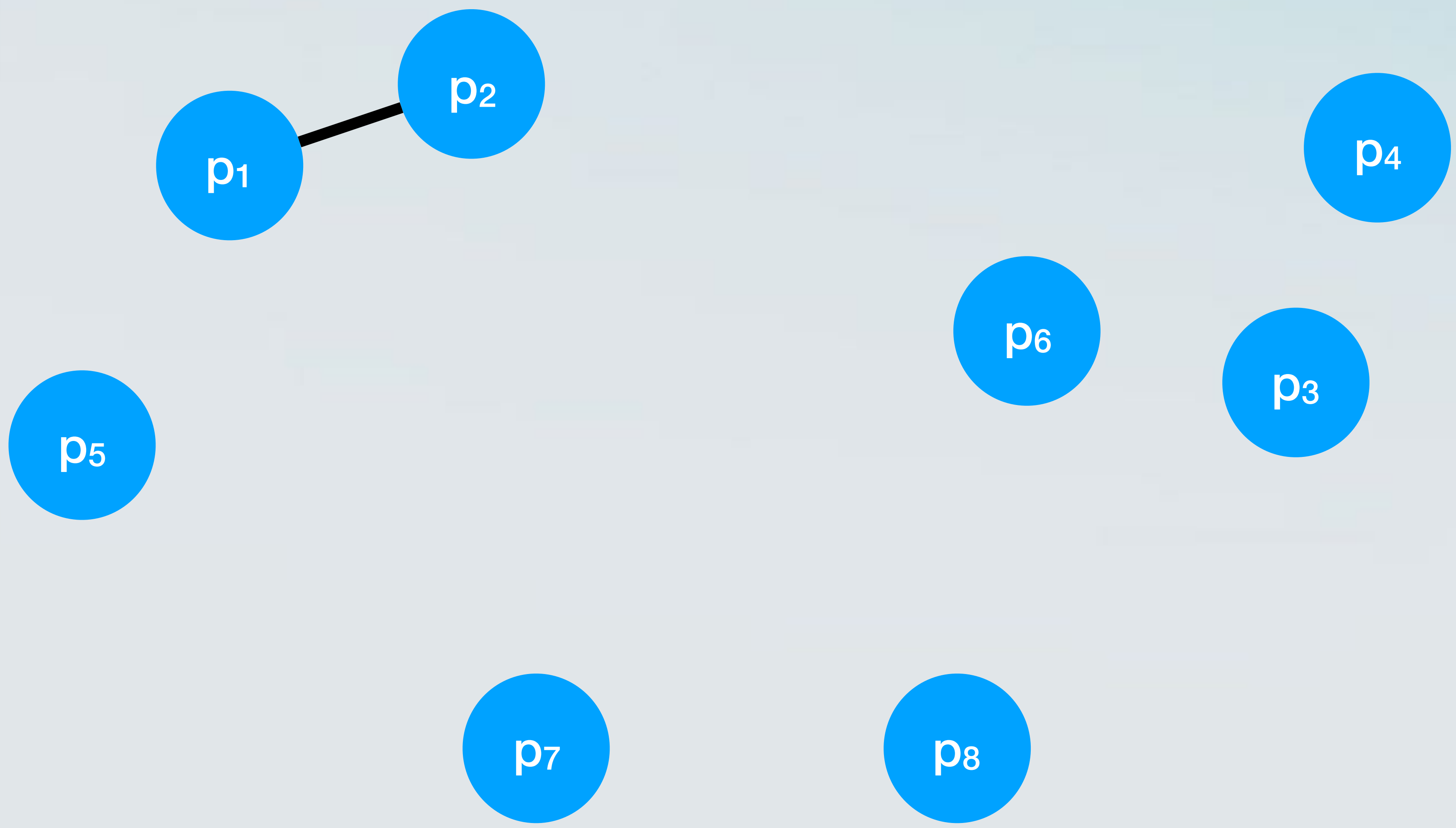
8 components

Clustering (concretely)



8 components

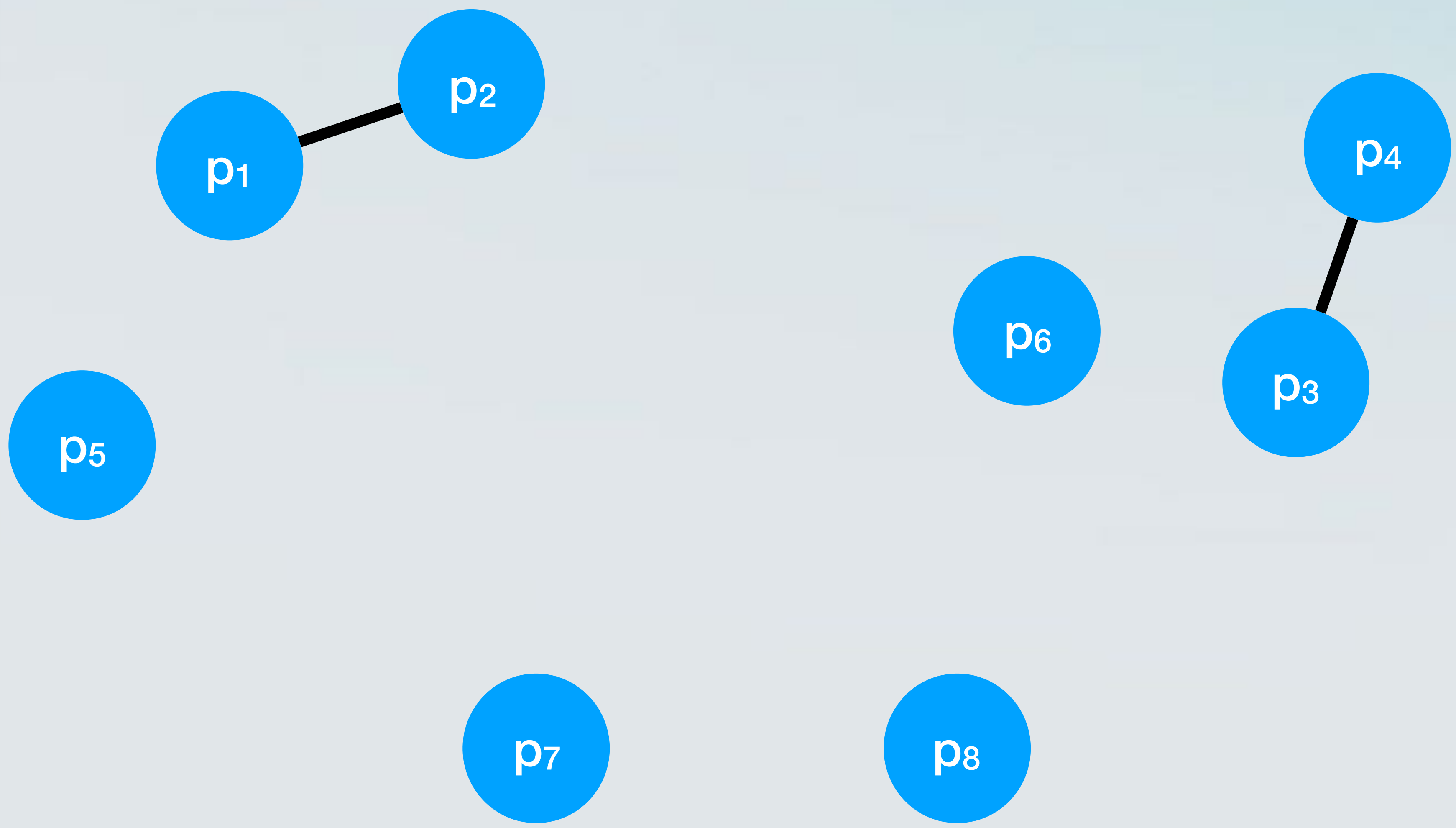
Clustering (concretely)



8 components

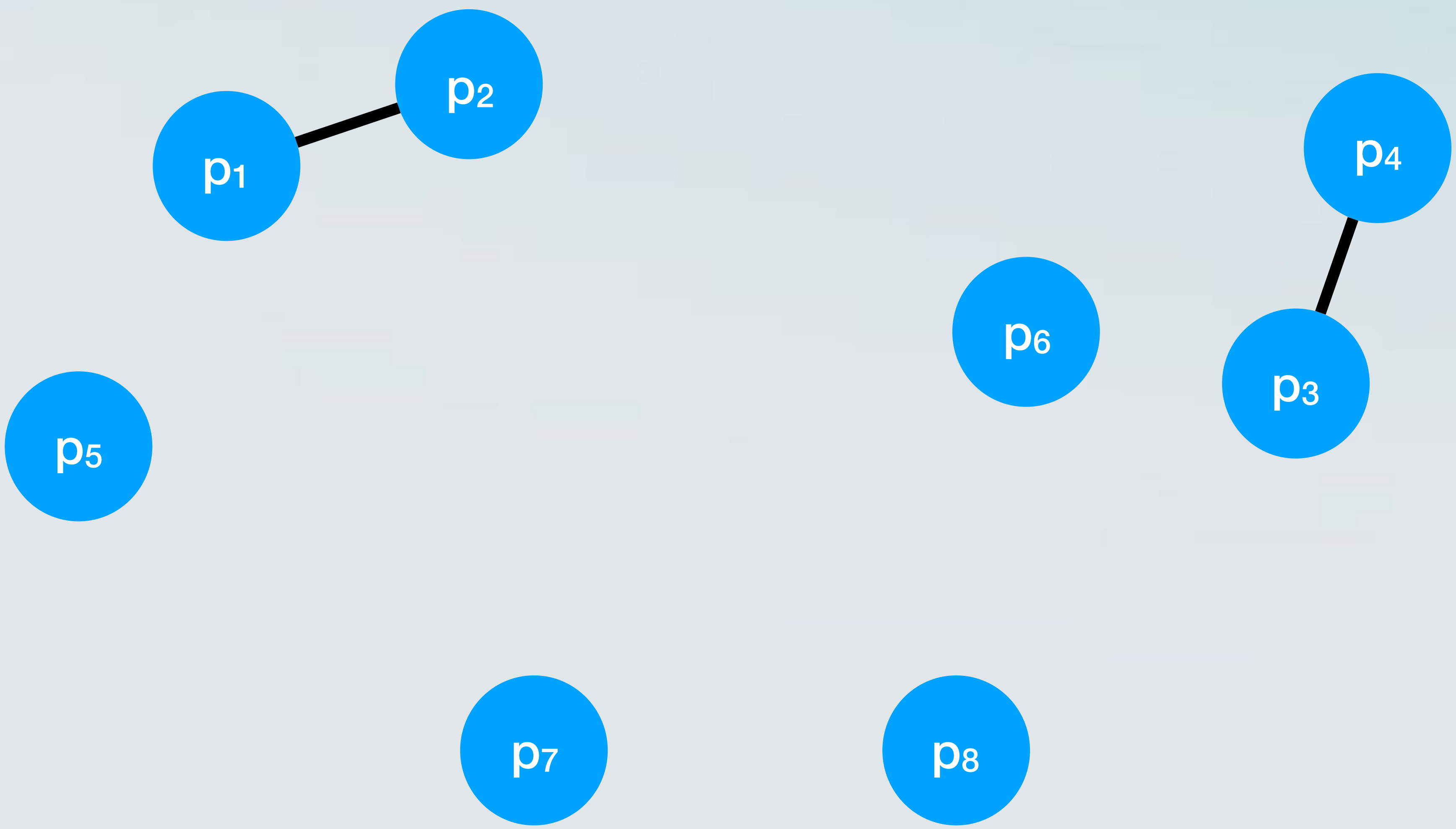
7 components

Clustering (concretely)



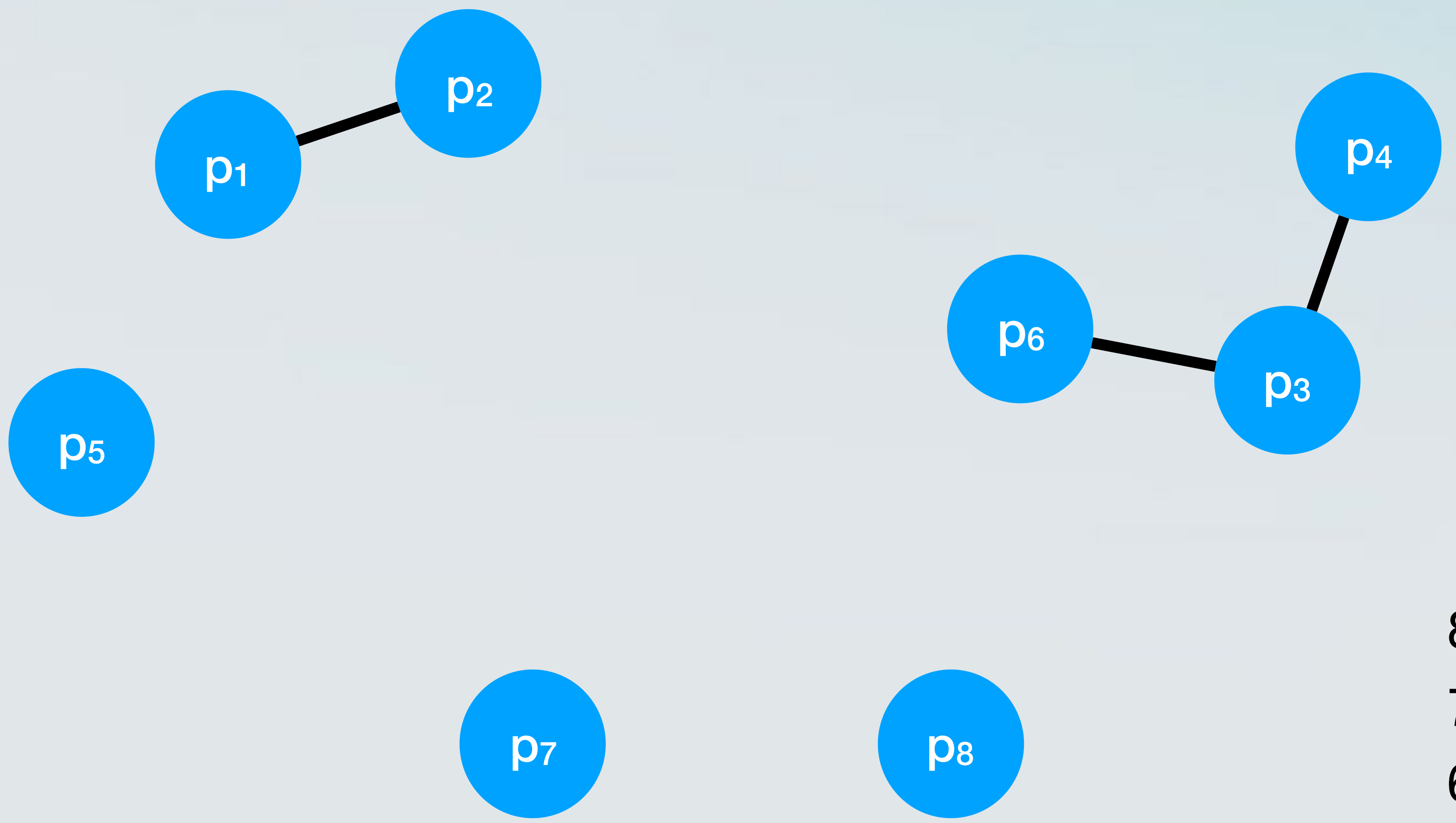
8 components
7 components

Clustering (concretely)



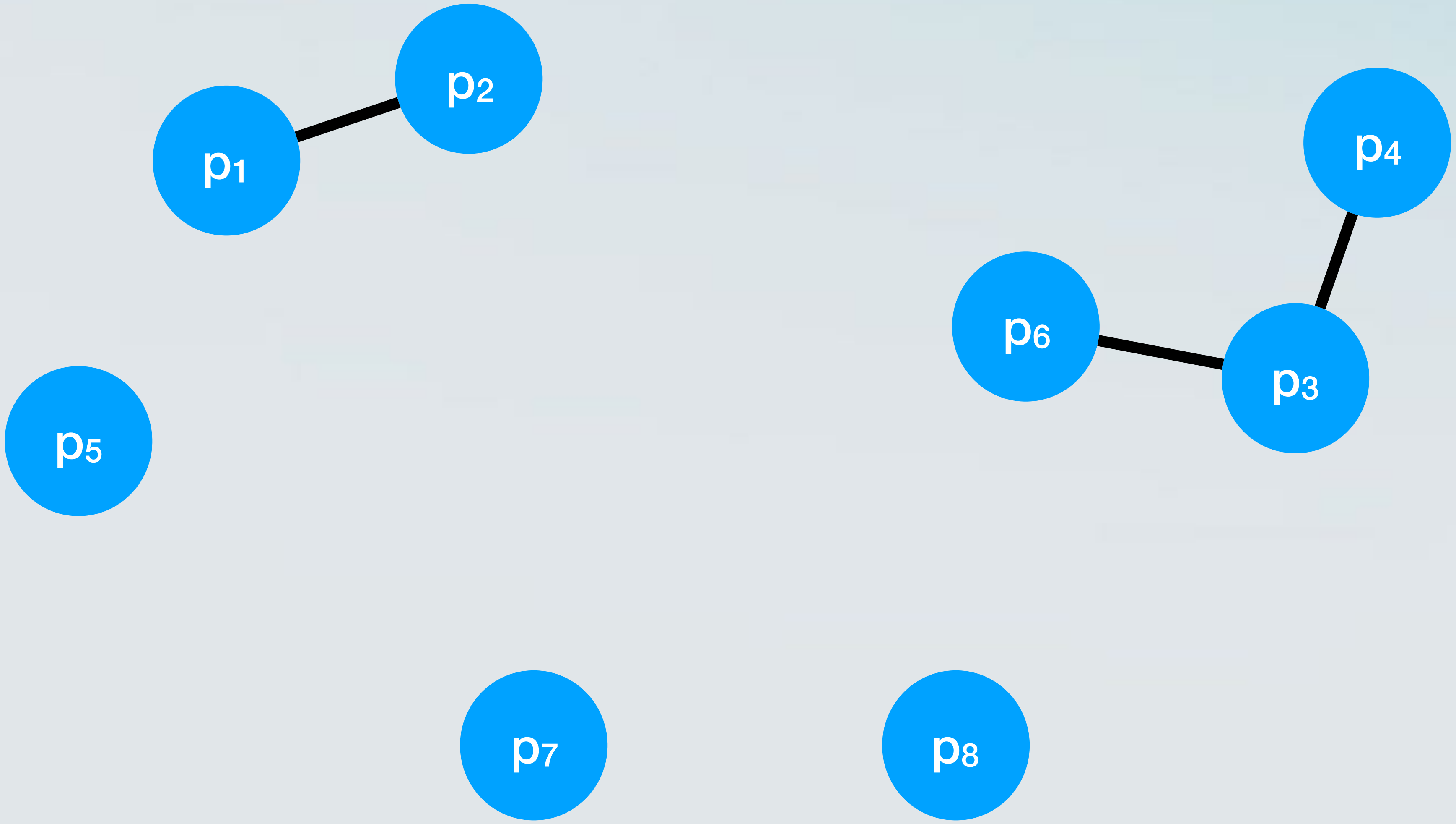
8 components
7 components
6 components

Clustering (concretely)



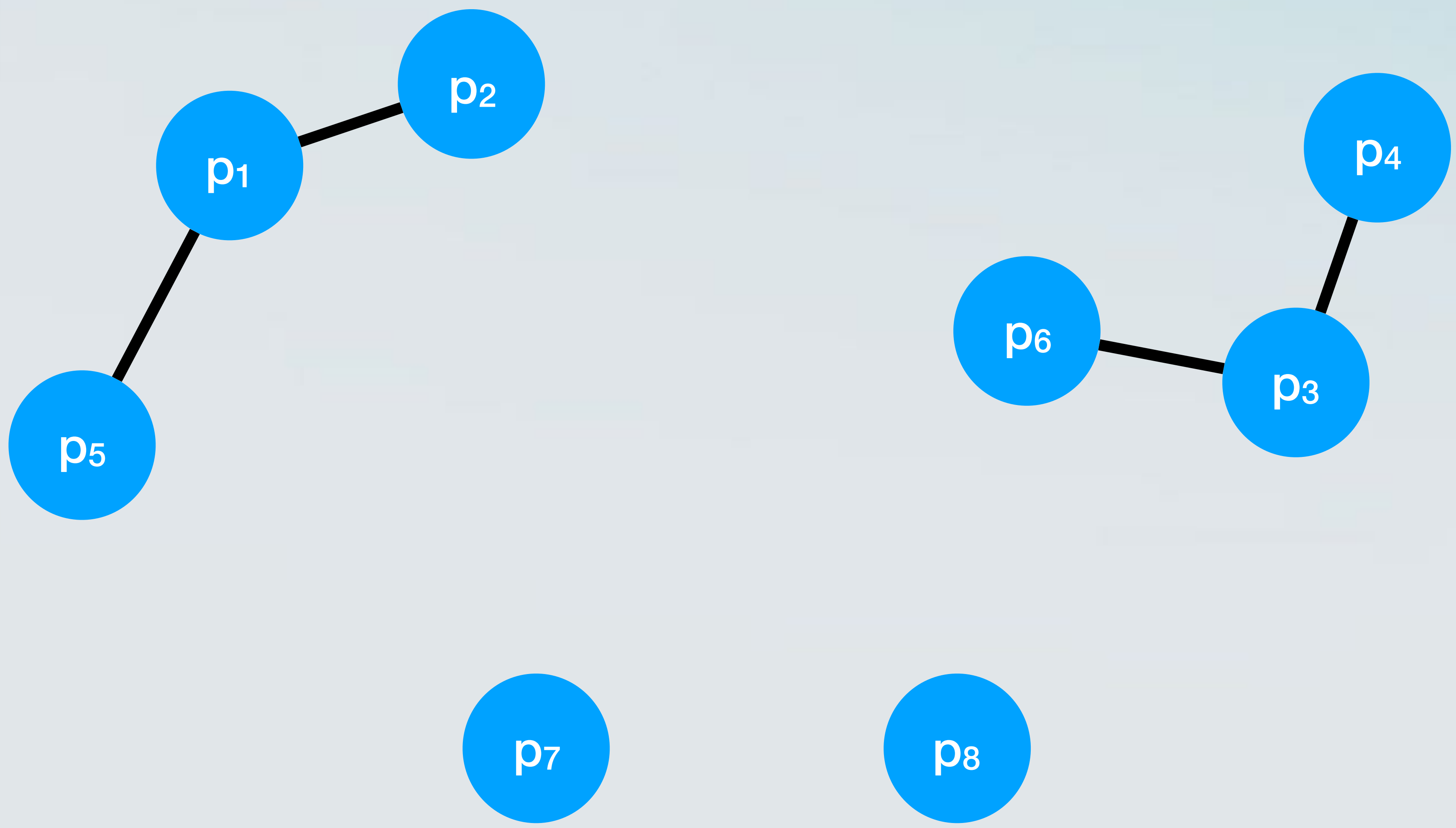
8 components
7 components
6 components

Clustering (concretely)



8 components
7 components
6 components
5 components

Clustering (concretely)



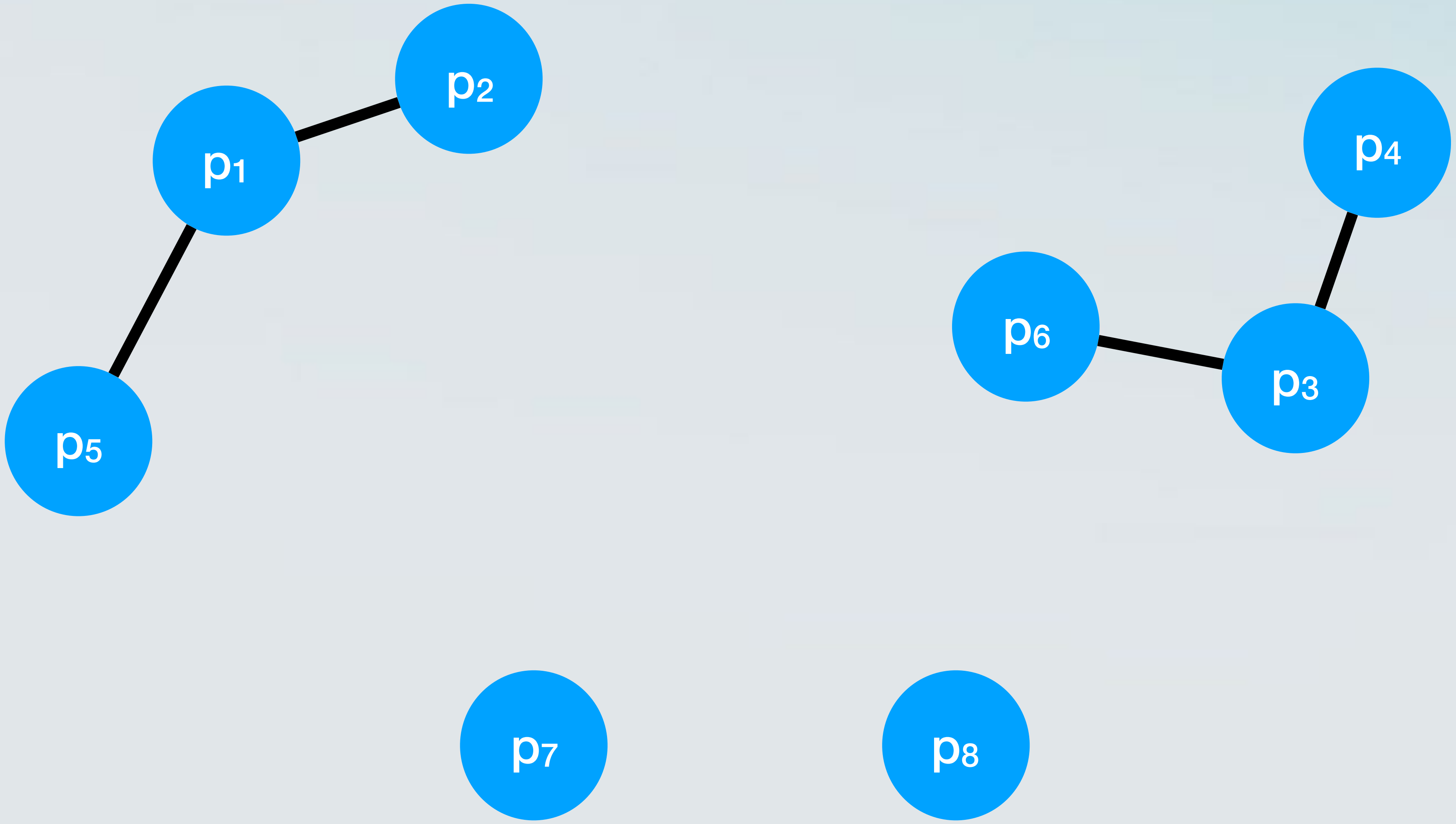
8 components

7 components

6 components

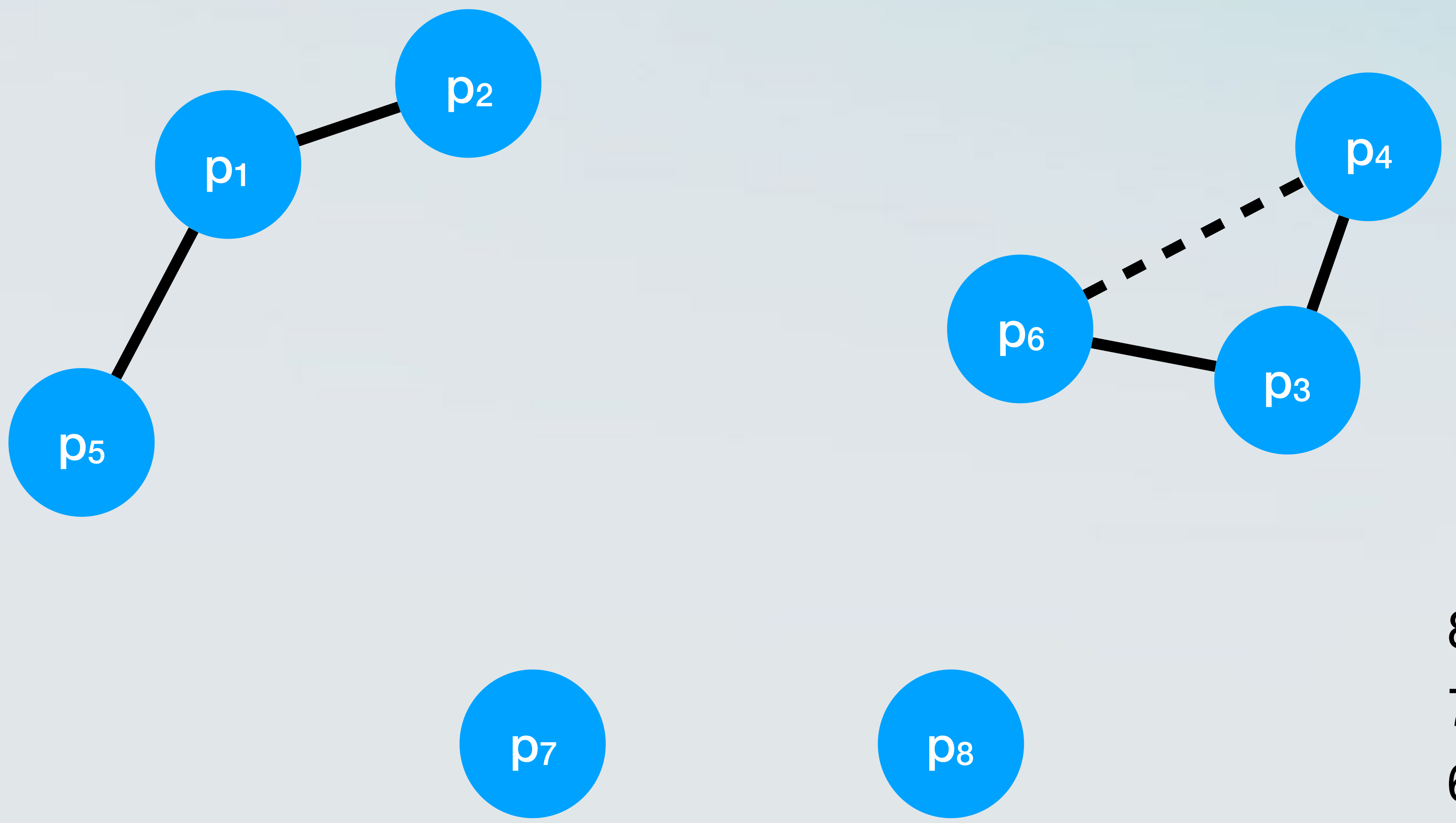
5 components

Clustering (concretely)



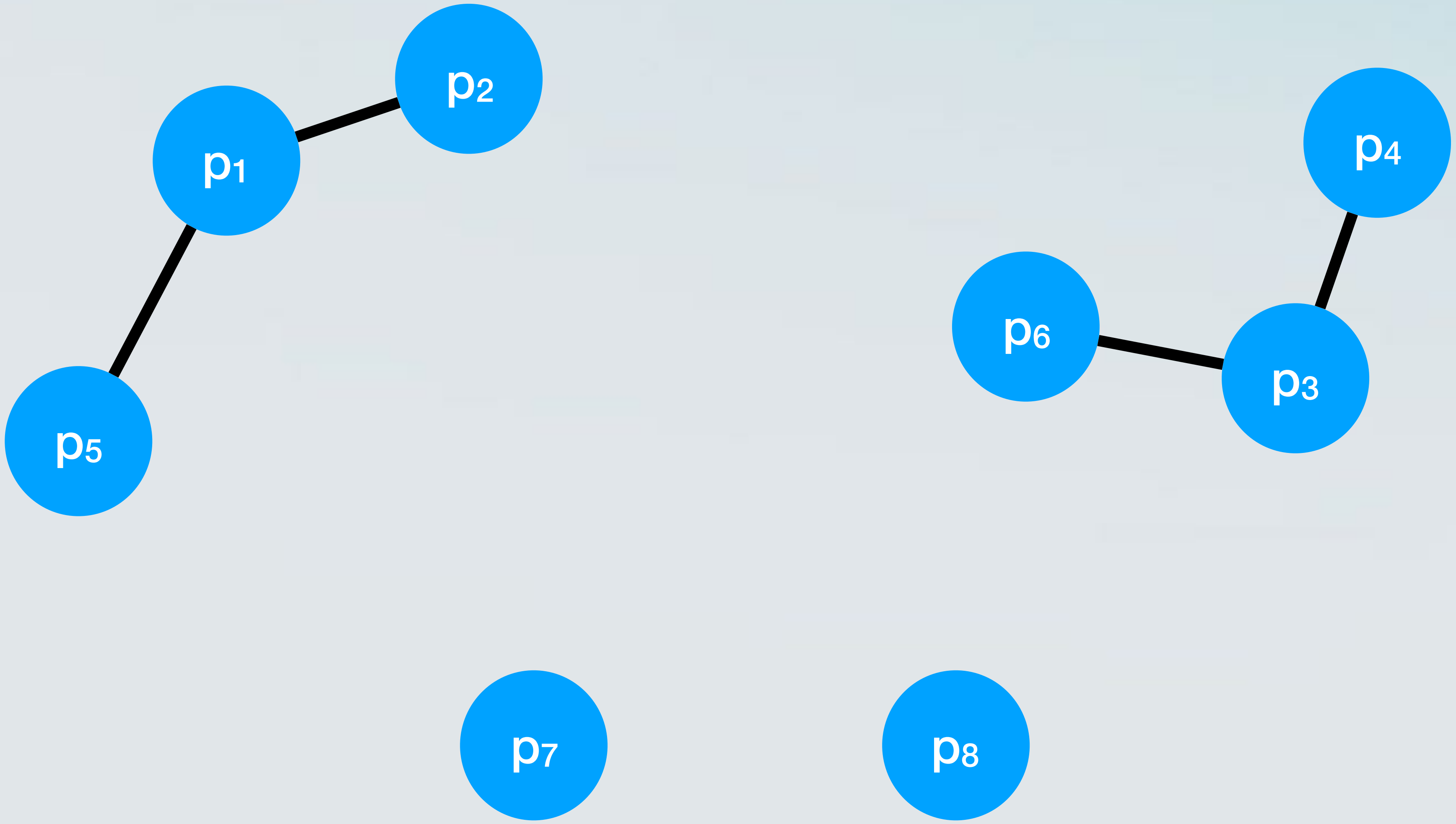
8 components
7 components
6 components
5 components
4 components

Clustering (concretely)



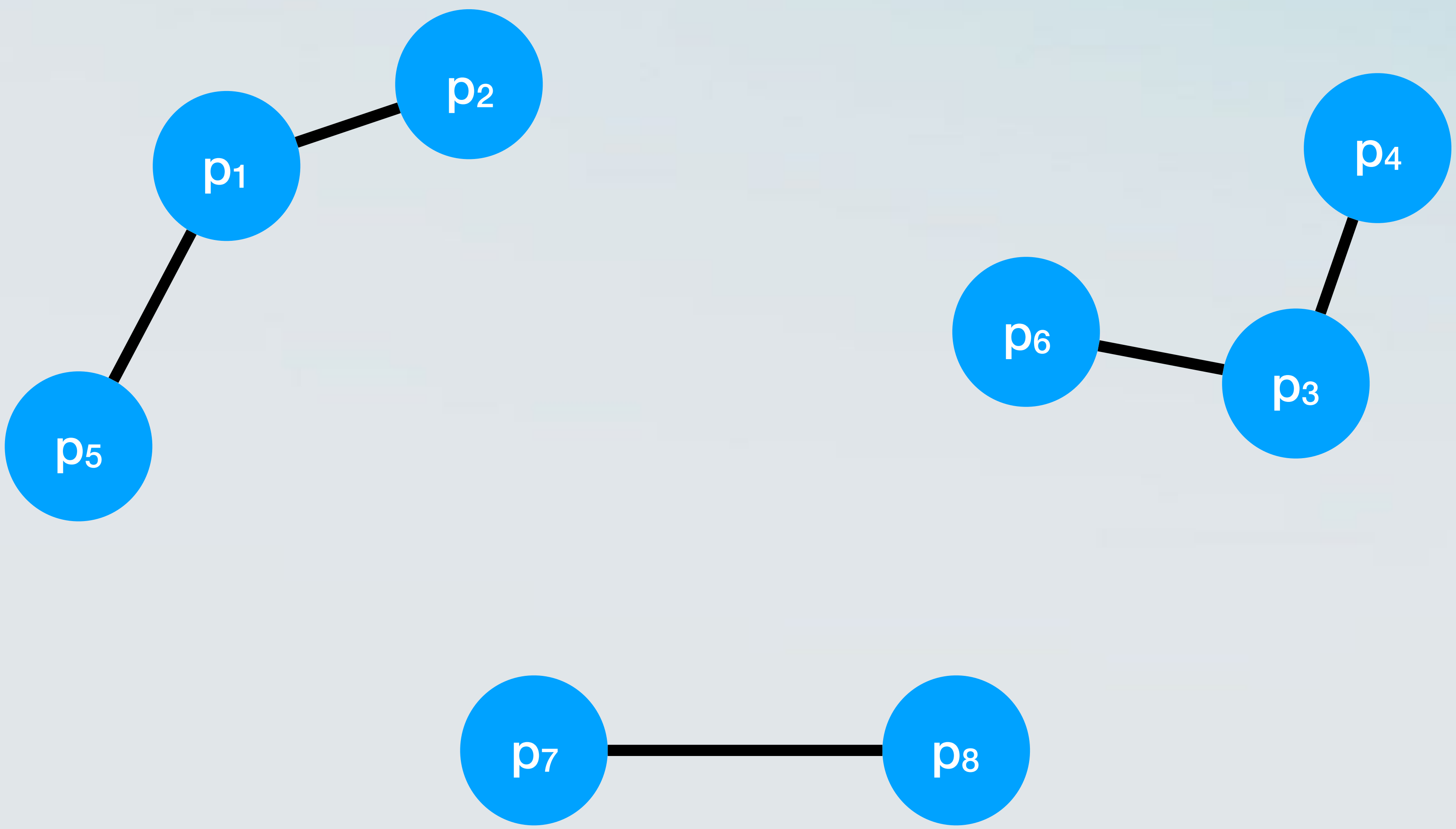
8 components
7 components
6 components
5 components
4 components

Clustering (concretely)



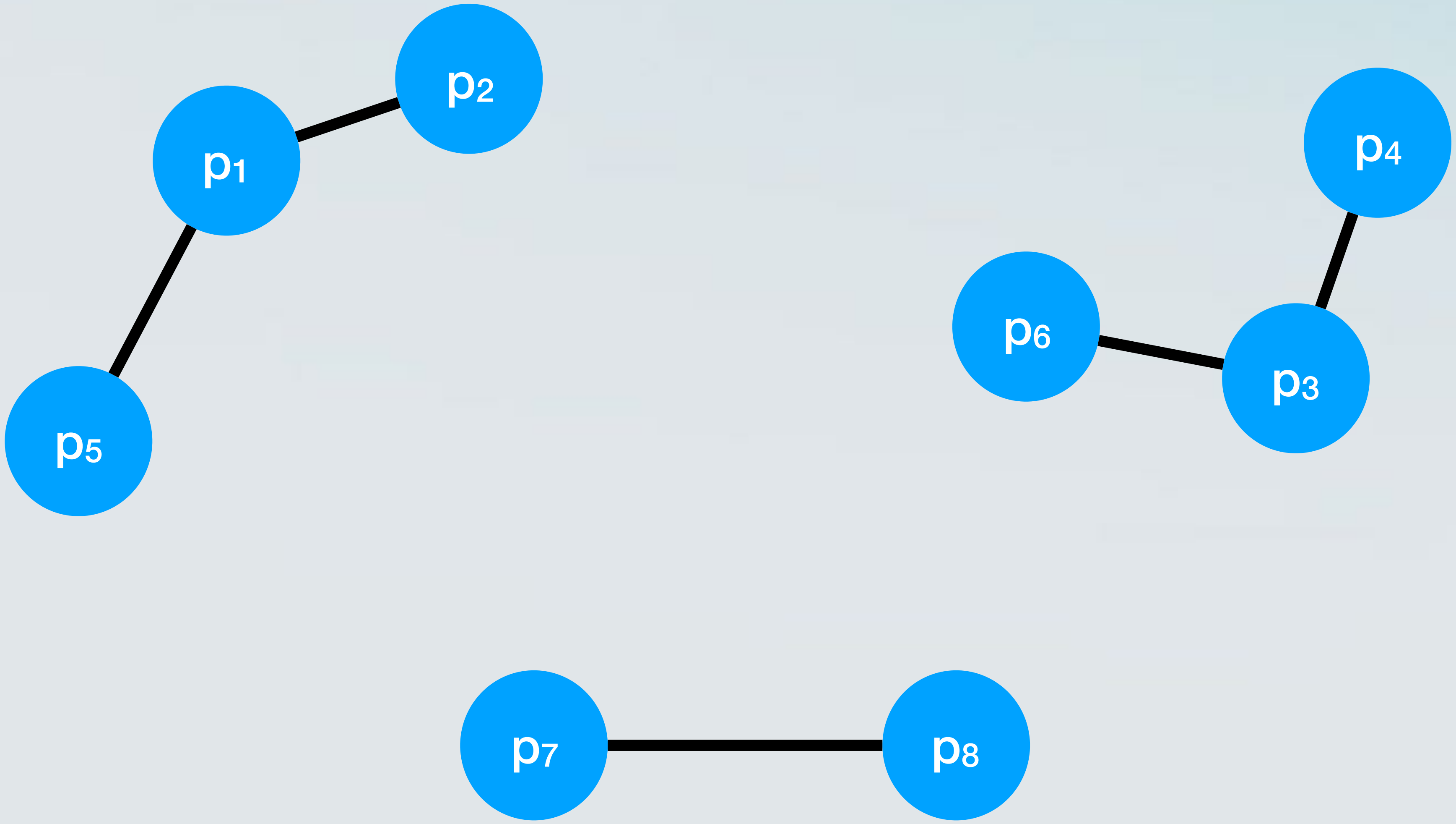
8 components
7 components
6 components
5 components
4 components

Clustering (concretely)



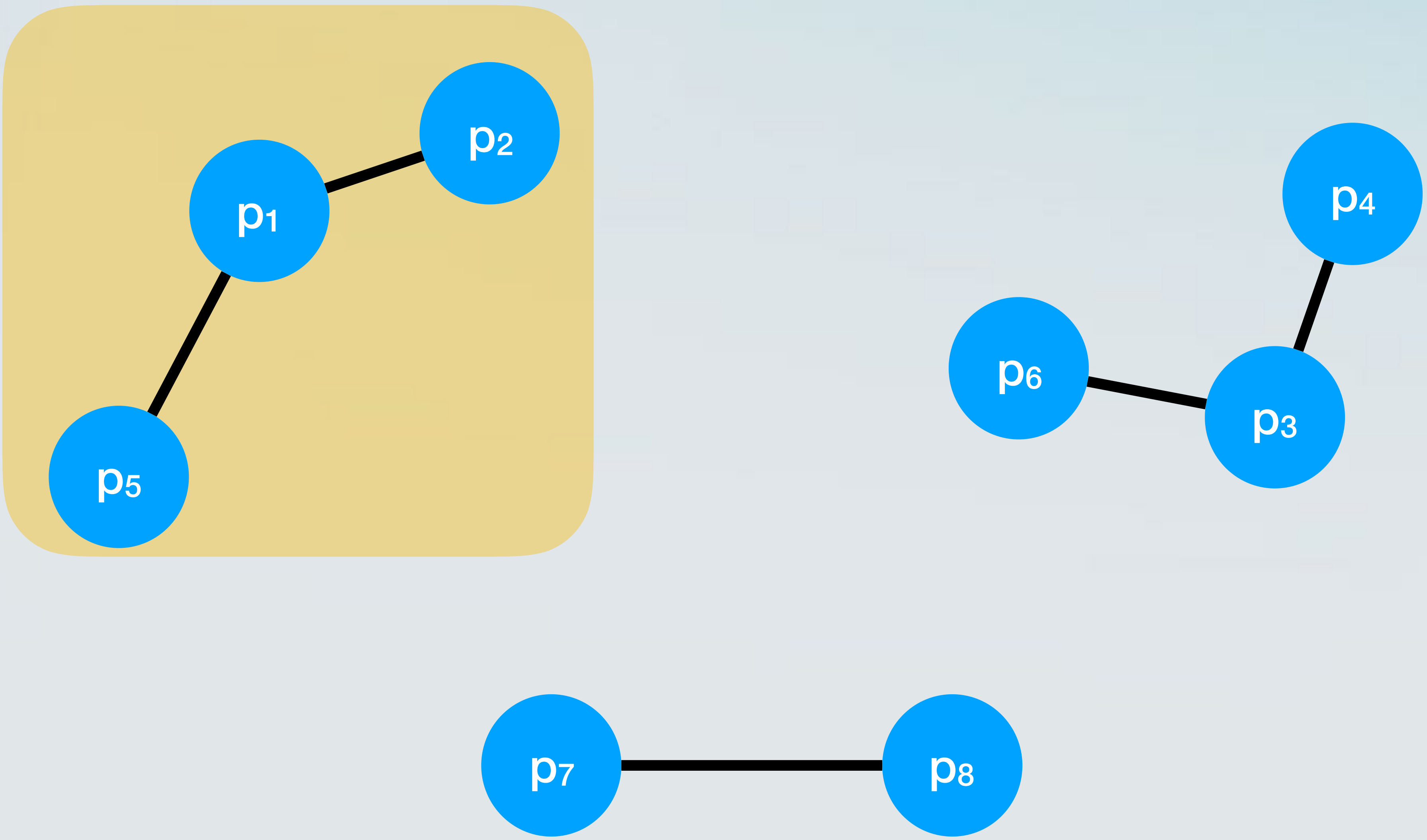
8 components
7 components
6 components
5 components
4 components

Clustering (concretely)



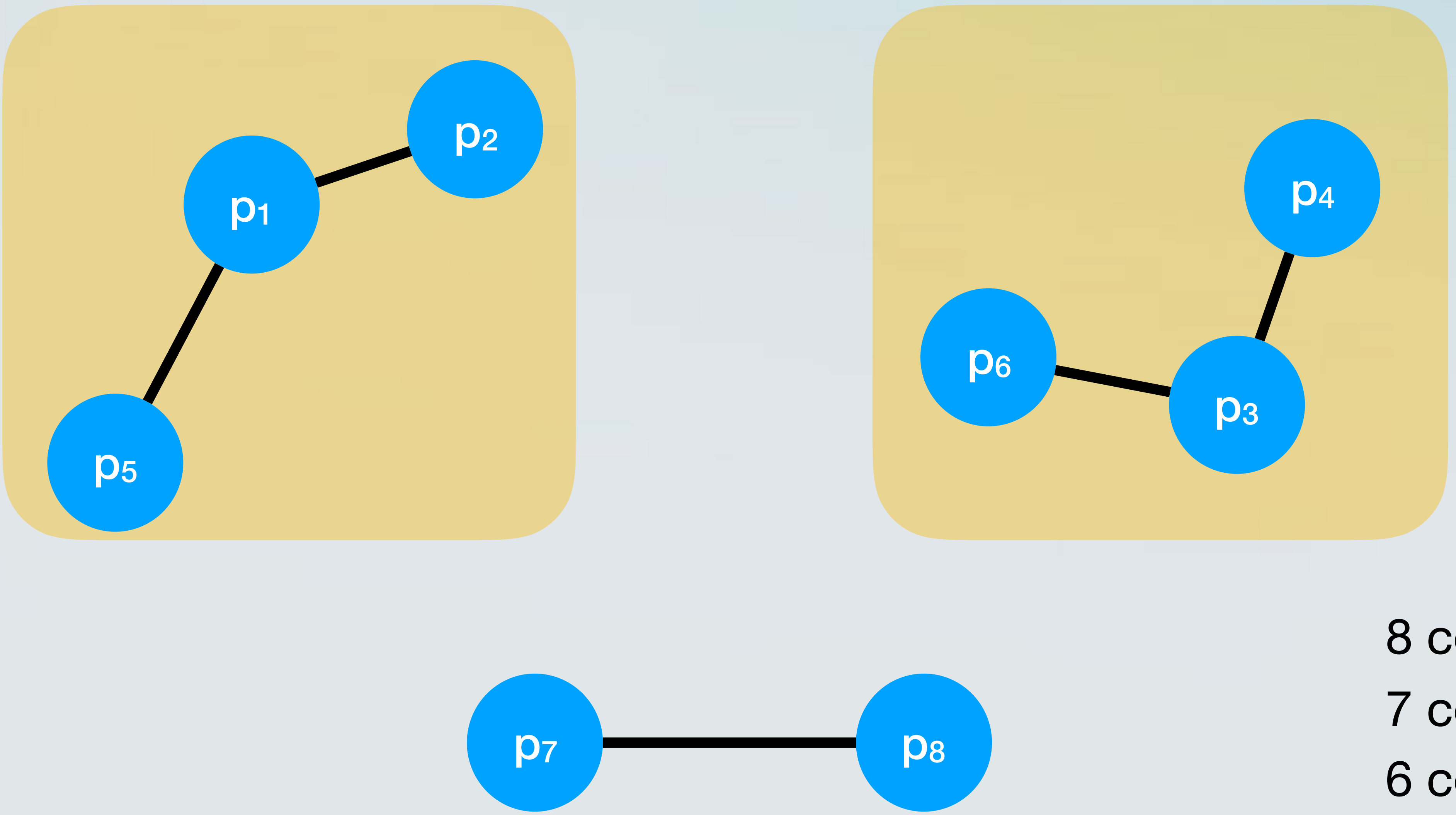
8 components
7 components
6 components
5 components
4 components
3 components

Clustering (concretely)



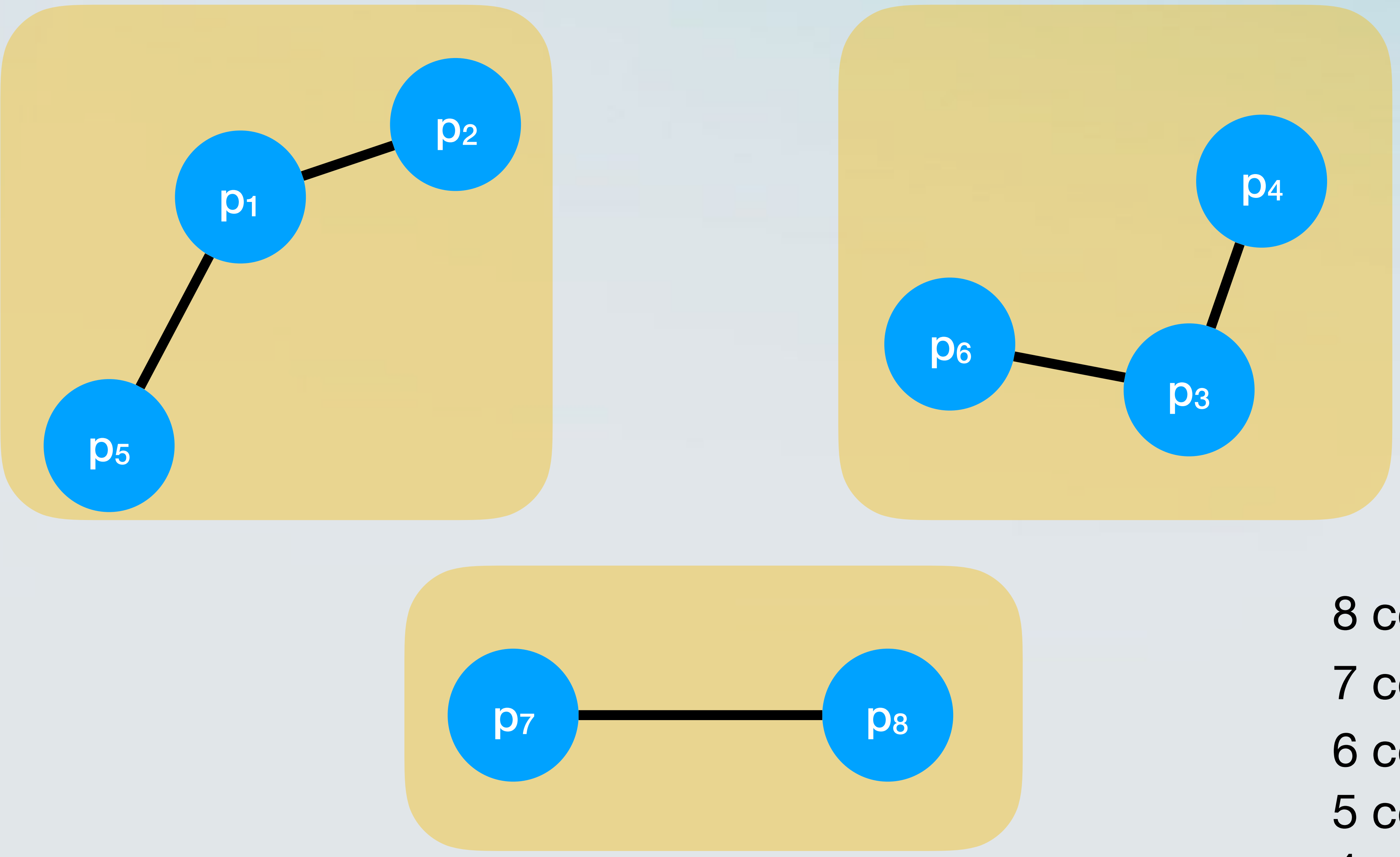
8 components
7 components
6 components
5 components
4 components
3 components

Clustering (concretely)



8 components
7 components
6 components
5 components
4 components
3 components

Clustering (concretely)



8 components
7 components
6 components
5 components
4 components
3 components

Sound familiar?

Sound familiar?

- Pick two objects p_i and p_j with the smallest distance d(p_i,p_j).

Sound familiar?

- Pick two objects p_i and p_j with the smallest distance d(p_i,p_j).
- Pick an edge (p_i, p_j) with the smallest cost d(p_i,p_j).

Sound familiar?

- Pick two objects p_i and p_j with the smallest distance $d(p_i,p_j)$.
- Connect them with an edge $e=(p_i,p_j)$.

- Pick an edge (p_i, p_j) with the smallest cost $d(p_i,p_j)$.

Sound familiar?

- Pick two objects p_i and p_j with the smallest distance d(p_i,p_j).
- Connect them with an edge e=(p_i,p_j).

- Pick an edge (p_i, p_j) with the smallest cost d(p_i,p_j).
- Include the edge in the output.

Sound familiar?

- Pick two objects p_i and p_j with the smallest distance $d(p_i,p_j)$.
- Connect them with an edge $e=(p_i,p_j)$.
- Continue like this until we obtain k clusters.

- Pick an edge (p_i, p_j) with the smallest cost $d(p_i,p_j)$.
- Include the edge in the output.

Sound familiar?

- Pick two objects p_i and p_j with the smallest distance $d(p_i,p_j)$.
- Connect them with an edge $e=(p_i,p_j)$.
- Continue like this until we obtain k clusters.

- Pick an edge (p_i, p_j) with the smallest cost $d(p_i,p_j)$.
- Include the edge in the output.
- Continue like this until obtain k connected components.

Sound familiar?

- Pick two objects p_i and p_j with the smallest distance d(p_i,p_j).
- Connect them with an edge e=(p_i,p_j).
- Continue like this until we obtain k clusters.
- If the edge e under consideration connects two objects p_i and p_j already in the same component, skip it.

- Pick an edge (p_i, p_j) with the smallest cost d(p_i,p_j).
- Include the edge in the output.
- Continue like this until obtain k connected components.

Sound familiar?

- Pick two objects p_i and p_j with the smallest distance d(p_i,p_j).
- Connect them with an edge e=(p_i,p_j).
- Continue like this until we obtain k clusters.
- If the edge e under consideration connects two objects p_i and p_j already in the same component, skip it.

- Pick an edge (p_i, p_j) with the smallest cost d(p_i,p_j).
- Include the edge in the output.
- Continue like this until obtain k connected components.
- If the edge e under consideration introduces a cycle, then skip it.

Kruskal's algorithm

- Pick two objects p_i and p_j with the smallest distance d(p_i,p_j).
- Connect them with an edge e=(p_i,p_j).
- Continue like this until we obtain k clusters.
- If the edge e under consideration connects two objects p_i and p_j already in the same component, skip it.

- Pick an edge (p_i, p_j) with the smallest cost d(p_i,p_j).
- Include the edge in the output.
- Continue like this until we connect all nodes.
- If the edge e under consideration introduces a cycle, then skip it.

Kruskal's algorithm

- Pick an edge (p_i, p_j) with the smallest cost d(p_i,p_j).
- Include the edge in the output.

Kruskal's algorithm

- Pick an edge (p_i, p_j) with the smallest cost d(p_i,p_j).
- Include the edge in the output.
- Stop before including the last k-1 edges.

Kruskal’s algorithm

- Pick an edge (p_i, p_j) with the smallest cost d(p_i,p_j).
- Include the edge in the output.
- Stop before including the last k-1 edges.
    - i.e., in the end, remove the k-1 most expensive edges.

Kruskal’s algorithm

- Pick an edge (p_i, p_j) with the smallest cost d(p_i,p_j).
- Include the edge in the output.
- Stop before including the last k-1 edges.
  - i.e., in the end, remove the k-1 most expensive edges.
- If the edge e under consideration introduces a cycle, then skip it.

Correctness

- Lemma: Let $C_1, C_2, \ldots, C_k$ be the k connected components formed by deleting the k-1 most expensive edges from a minimum spanning tree T.

  These are a k-clustering of maximum spacing.

Proof of the Lemma

Proof of the Lemma

- Let $C = \{C_1, C_2, \ldots, C_k\}$.

Proof of the Lemma

- Let $C = \{C_1, C_2, \ldots , C_k\}$.
  - C is obviously a clustering (feasiblity).

Proof of the Lemma

- Let $C = \{C_1, C_2, \ldots, C_k\}$.
    - C is obviously a clustering (feasiblity).
- Order the k-1 most expensive edges of the minimum spanning tree in non-increasing order:

Proof of the Lemma

- Let $C = \{C_1, C_2, \ldots, C_k\}$.
  - C is obviously a clustering (feasiblity).
- Order the k-1 most expensive edges of the minimum spanning tree in non-increasing order:
  - $e_{k-1}, e_{k-2}, \ldots, e_1$

Proof of the Lemma

- Let $C = \{C_1, C_2, \ldots, C_k\}$.
  - C is obviously a clustering (feasiblity).
- Order the k-1 most expensive edges of the minimum spanning tree in non-increasing order:
  - $e_{k-1}, e_{k-2}, \ldots, e_1$
- What is the *spacing* of C?

Proof of the Lemma

- Let $C = \{C_1, C_2, \ldots, C_k\}$.
  - C is obviously a clustering (feasiblity).
- Order the k-1 most expensive edges of the minimum spanning tree in non-increasing order:
  - $e_{k-1}, e_{k-2}, \ldots, e_1$
- What is the *spacing* of C?
  - It is the cost of e_1.

Proof of the Lemma

Proof of the Lemma

- Let C' = $\{C'_1, C'_2, \ldots, C'_k\}$ be *any other* k-clustering.

Proof of the Lemma

- Let C' = $\{C'_1, C'_2, \ldots, C'_k\}$ be *any other* k-clustering.
- By *other*, there exists a cluster C_r of C which is not contained in any cluster C'_s of C'.

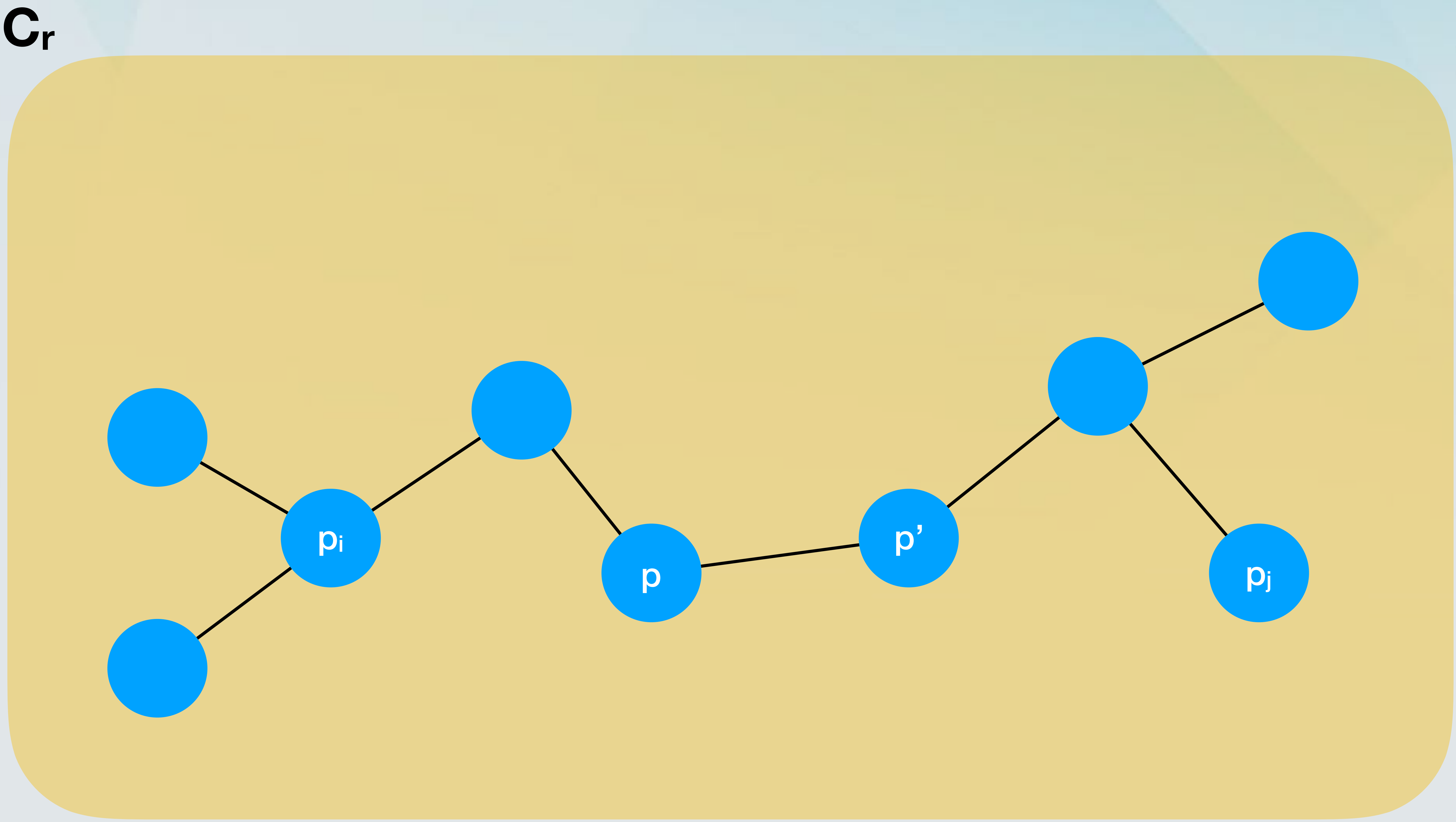
Proof of the Lemma

- Let C' = {C'_1, C'_2, … , C'_k} be *any other* k-clustering.
- By *other*, there exists a cluster C_r of C which is not contained in any cluster C'_s of C'.
- This means that there exist points p_i, p_j in C_r that belong to different clusters in C'.

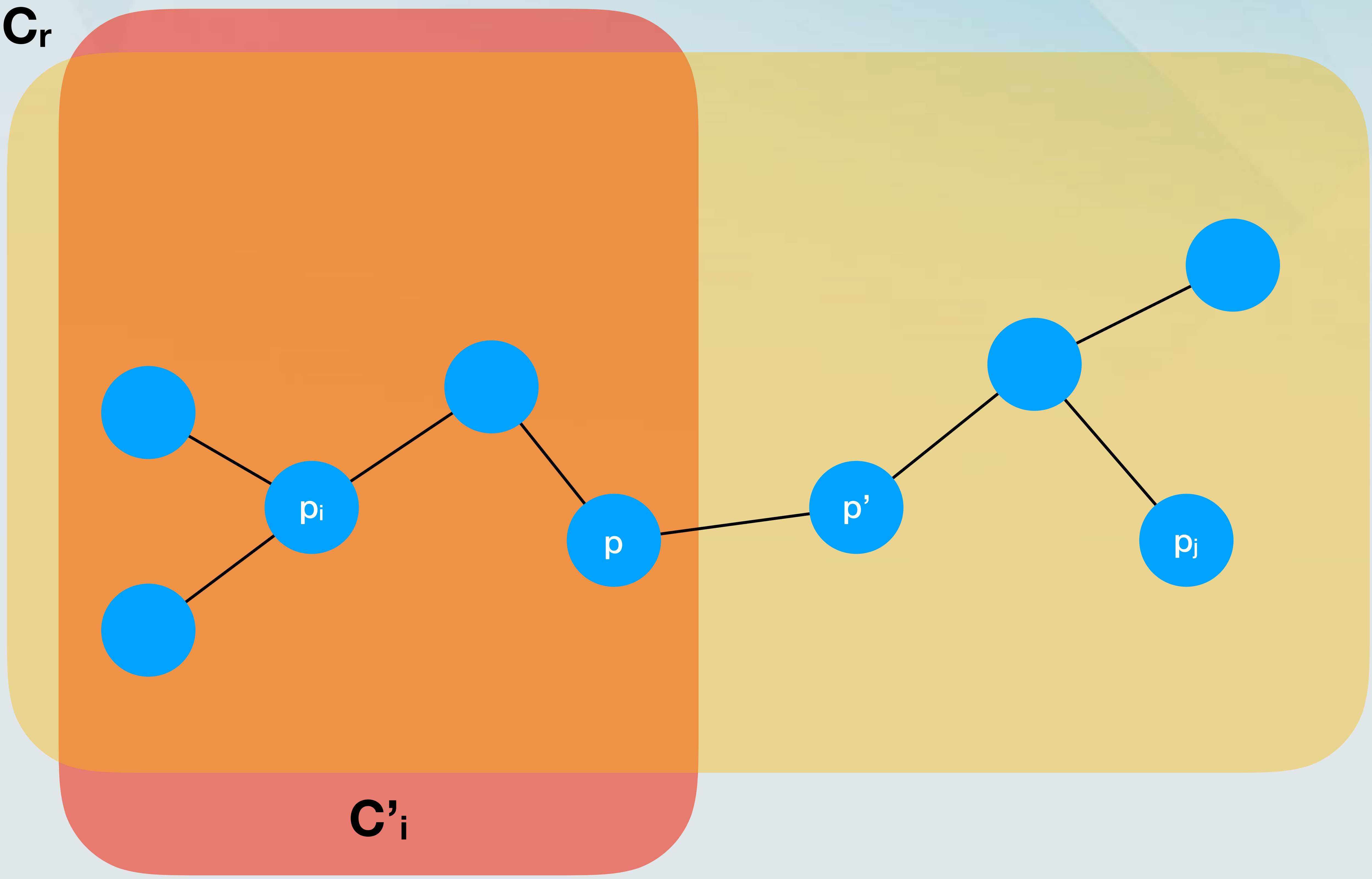
Proof of the Lemma

- Let C' = $\{C'_1, C'_2, \ldots, C'_k\}$ be *any other* k-clustering.
- By *other*, there exists a cluster C_r of C which is not contained in any cluster C'_s of C'.
- This means that there exist points p_i, p_j in C_r that belong to different clusters in C'.
  - Let C'_i and C'_j denote these clusters respectively.

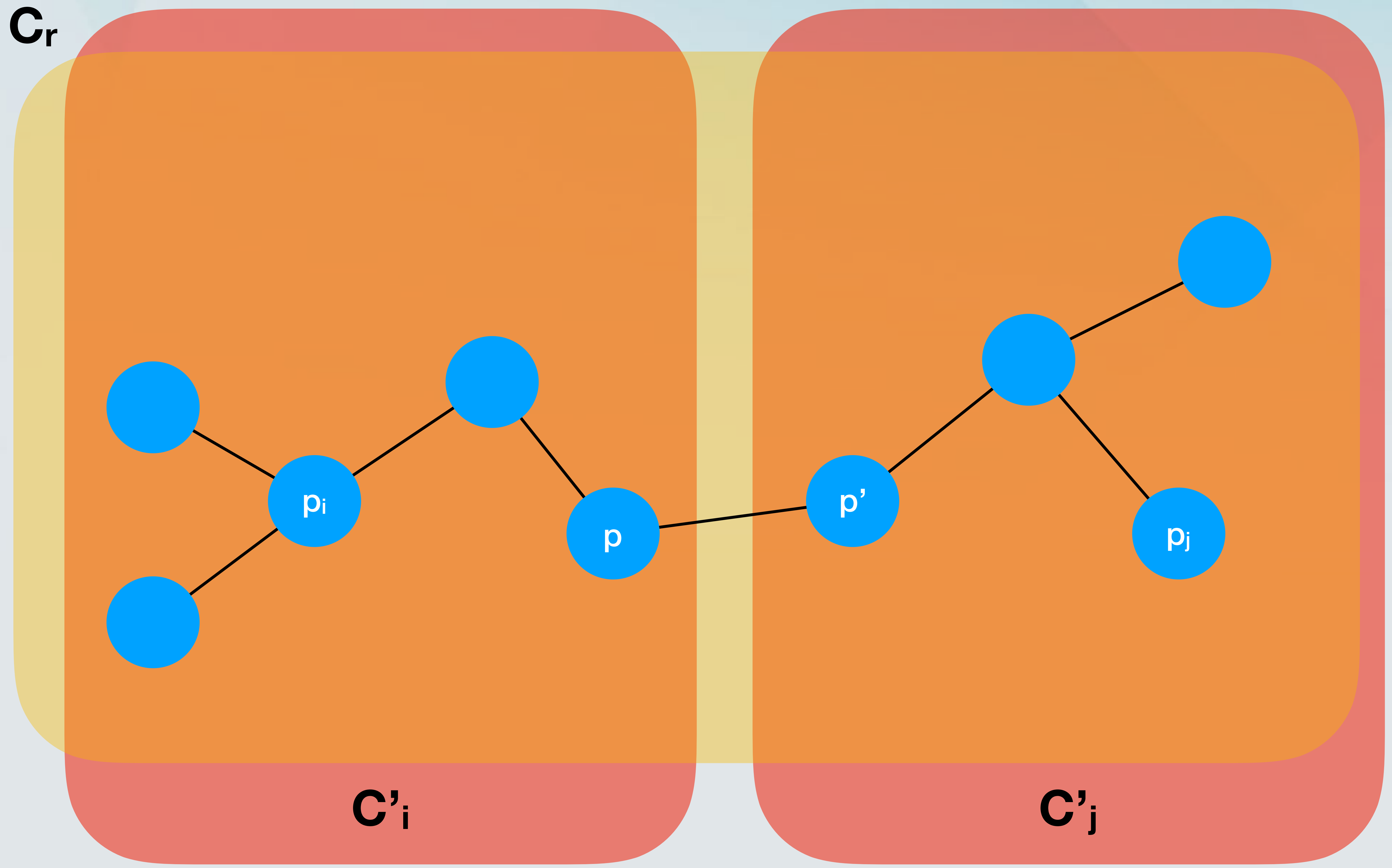
Proof of the Lemma



Proof of the Lemma



Proof of the Lemma



Proof of the Lemma

Proof of the Lemma

- This means that there exist points p_i, p_j in C_r that belong to different clusters in C'.

Proof of the Lemma

- This means that there exist points p_i, p_j in C_r that belong to different clusters in C’.

- This implies that Kruskal’s algorithm added all the edges of path from p_i to p_j and *none of these edges was deleted.*

Proof of the Lemma

- This means that there exist points p_i, p_j in C_r that belong to different clusters in C'.
- This implies that Kruskal's algorithm added all the edges of path from p_i to p_j and *none of these edges was deleted.*
- What is the maximum cost of any of these edges then?

Proof of the Lemma

- This means that there exist points p_i, p_j in C_r that belong to different clusters in C'.
- This implies that Kruskal's algorithm added all the edges of path from p_i to p_j and *none of these edges was deleted.*
- What is the maximum cost of any of these edges then?
  - The cost of e_1.

Proof of the Lemma

Proof of the Lemma

- This means that there exist points p_i, p_j in C_r that belong to different clusters in C'.

Proof of the Lemma

- This means that there exist points p_i, p_j in C_r that belong to different clusters in C'.

  - Let C'_i and C'_j denote these clusters respectively.

Proof of the Lemma

- This means that there exist points p_i, p_j in C_r that belong to different clusters in C'.
  - Let C'_i and C'_j denote these clusters respectively.
- On the path from p_i to p_j let p be the last node of C'_i and p' be the first node of C'_j.

Proof of the Lemma

- This means that there exist points p_i, p_j in C_r that belong to different clusters in C'.
  - Let C'_i and C'_j denote these clusters respectively.
- On the path from p_i to p_j let p be the last node of C'_i and p' be the first node of C'_j.
- What is the cost of (p,p')?

Proof of the Lemma

- This means that there exist points p_i, p_j in C_r that belong to different clusters in C'.
  - Let C'_i and C'_j denote these clusters respectively.
- On the path from p_i to p_j let p be the last node of C'_i and p' be the first node of C'_j.
- What is the cost of (p,p')?
  - At most the cost of e_1.

Proof of the Lemma

Proof of the Lemma

- What is the cost of (p,p')?

Proof of the Lemma

- What is the cost of (p,p')?
  - At most the cost of e_1.

Proof of the Lemma

- What is the cost of (p,p')?
  - At most the cost of e_1.
- What is the edge (p,p') with respect to C'?

Proof of the Lemma

- What is the cost of (p,p')?
  - At most the cost of e_1.
- What is the edge (p,p') with respect to C'?
  - It's an edge "crossing" clusters.

Proof of the Lemma

- What is the cost of (p,p')?
    - At most the cost of e_1.
- What is the edge (p,p') with respect to C'?
    - It's an edge "crossing" clusters.
    - The distance is at least **d(p,p').**

Proof of the Lemma

- What is the cost of (p,p')?
  - At most the cost of e_1.
- What is the edge (p,p') with respect to C'?
  - It's an edge "crossing" clusters.
  - The distance is at least **d(p,p').**
  - The spacing of C' is not smaller.

Proof of the Lemma