# Approximation Algorithms

## Lecture 2:
## SetCover and ShortestSuperString

### Part I:
### SetCover

*Alexander Wolff*

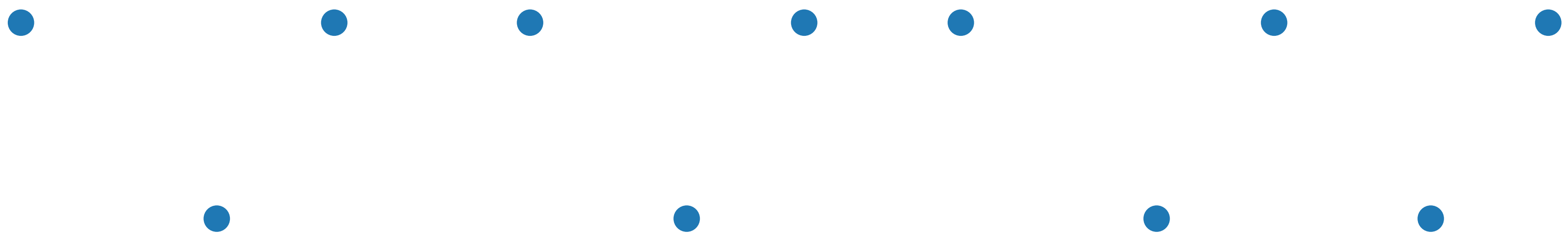
*Winter 2024/25*

# SetCover (card.)

Let $U$ be some **ground set** (universe),

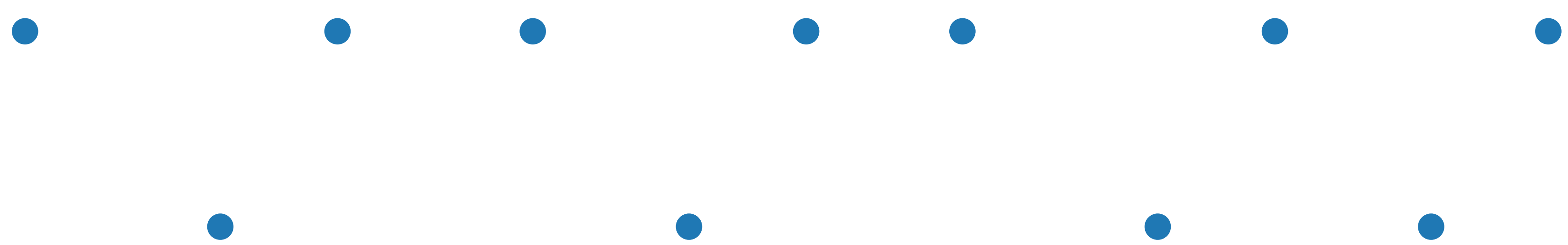
# SetCover (card.)

Let $U$ be some **ground set** (universe),



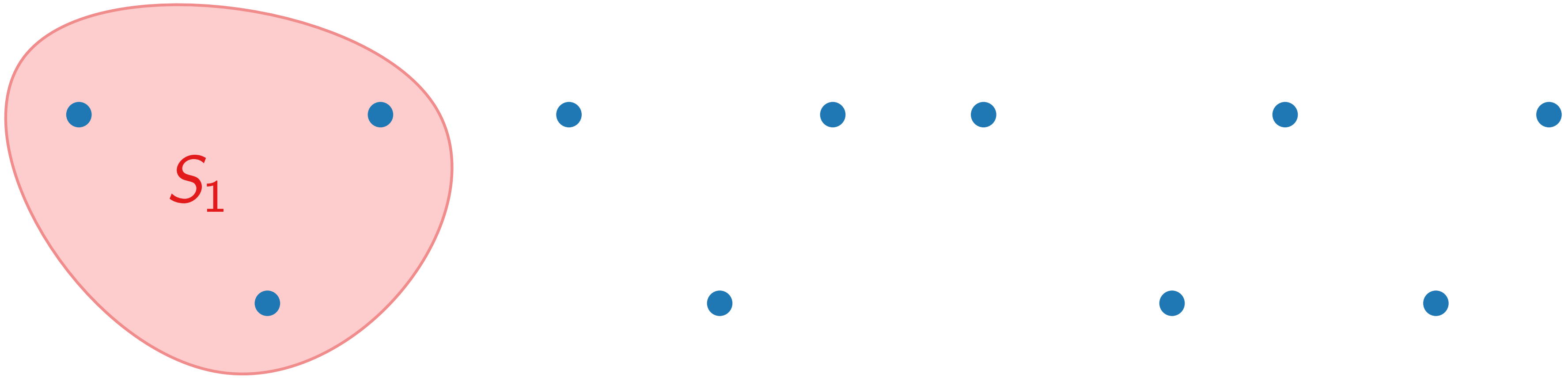
# SetCover (card.)

Let $U$ be some **ground set** (universe),
and let $\mathcal{S}$ be a family of **subsets** of $U$ with $\bigcup \mathcal{S} = U$.



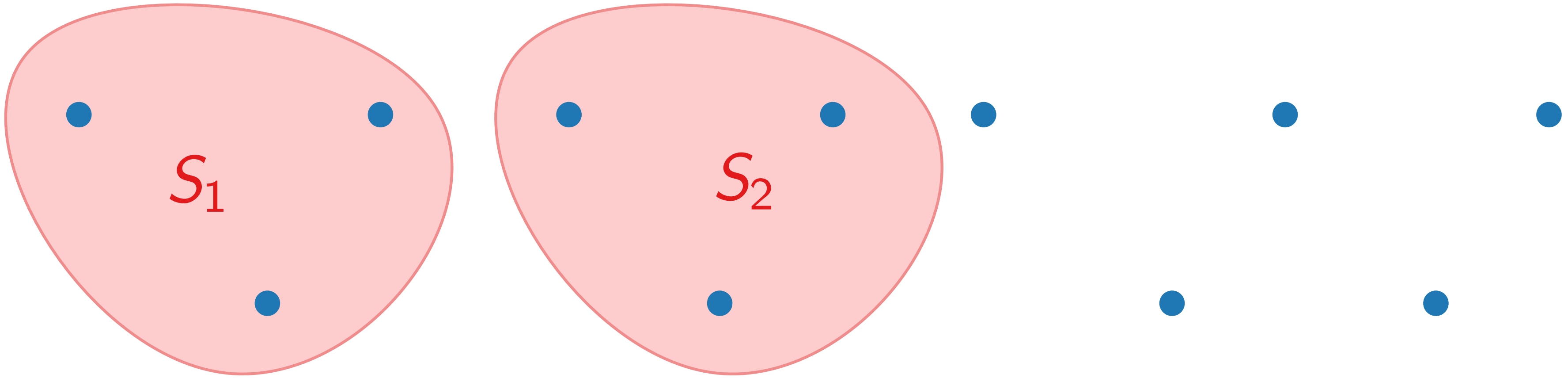
# SetCover (card.)

Let $U$ be some **ground set** (universe),
and let $\mathcal{S}$ be a family of **subsets** of $U$ with $\bigcup \mathcal{S} = U$.



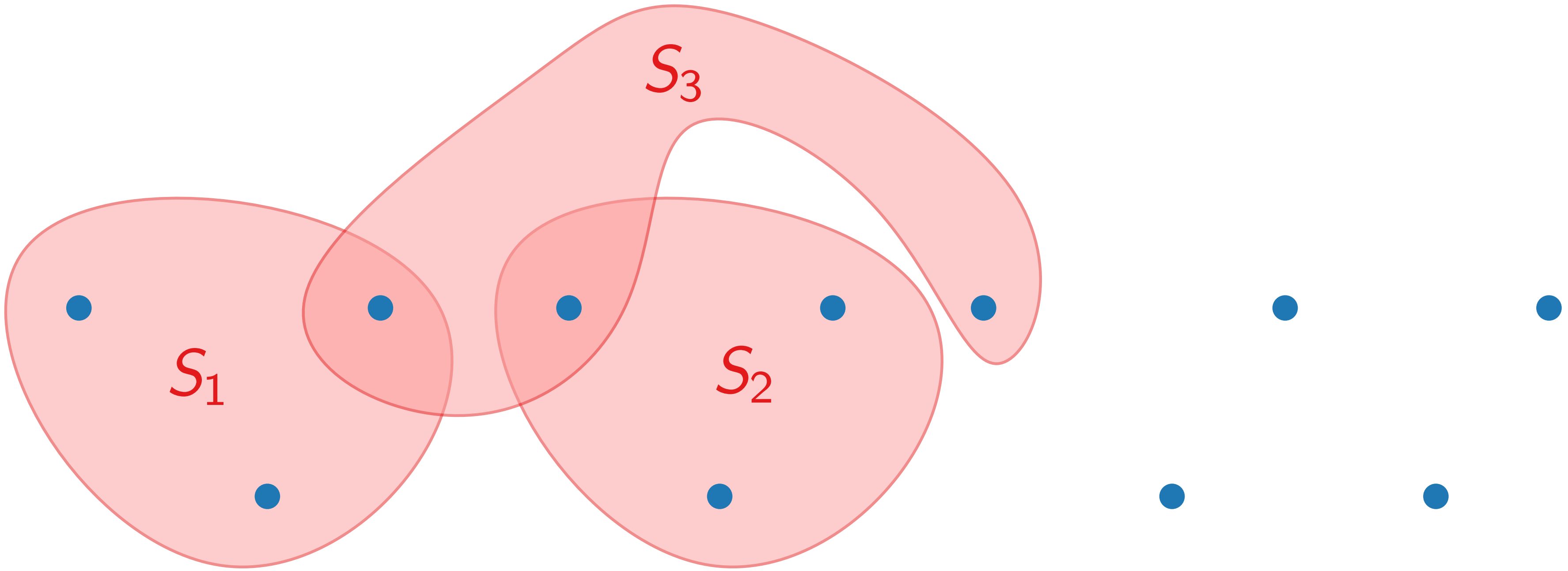
# SetCover (card.)

Let $U$ be some **ground set** (universe),
and let $\mathcal{S}$ be a family of **subsets** of $U$ with $\bigcup \mathcal{S} = U$.



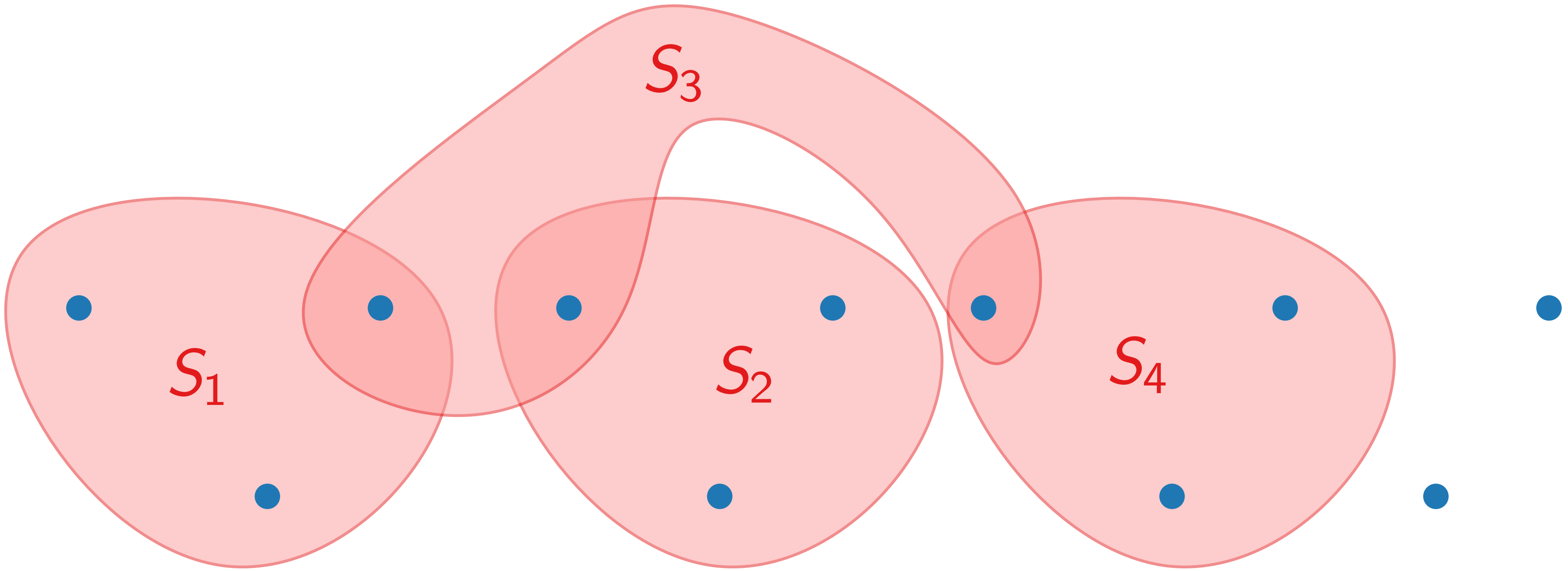
# SetCover (card.)

Let $U$ be some **ground set** (universe),
and let $\mathcal{S}$ be a family of **subsets** of $U$ with $\bigcup \mathcal{S} = U$.



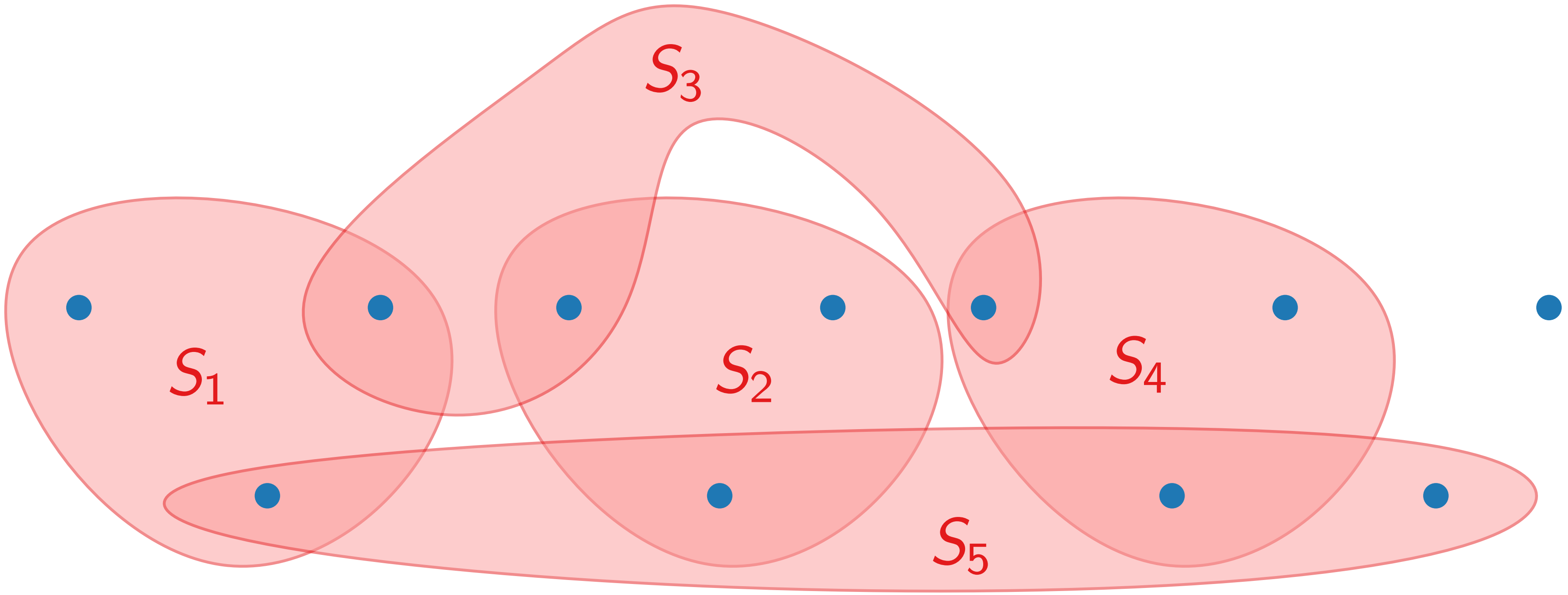
# SetCover (card.)

Let $U$ be some **ground set** (universe),
and let $\mathcal{S}$ be a family of **subsets** of $U$ with $\bigcup \mathcal{S} = U$.



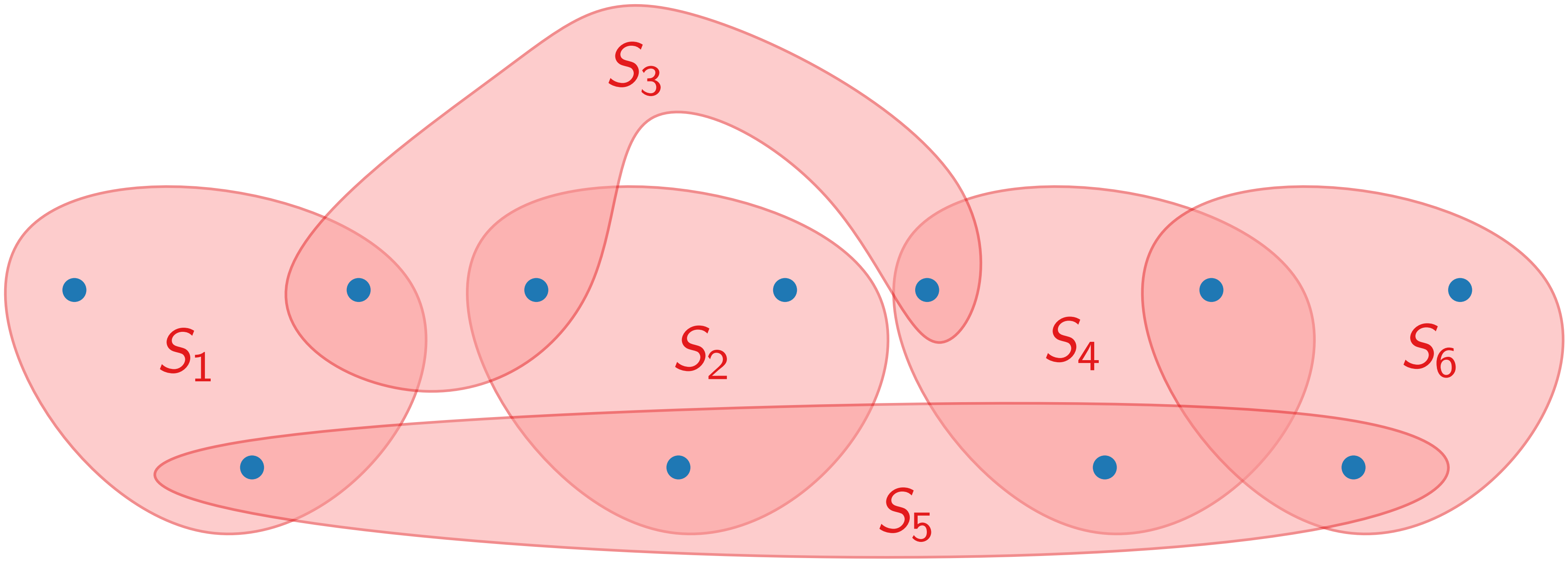
# SetCover (card.)

Let $U$ be some **ground set** (universe),
and let $\mathcal{S}$ be a family of **subsets** of $U$ with $\bigcup \mathcal{S} = U$.



# SetCover (card.)

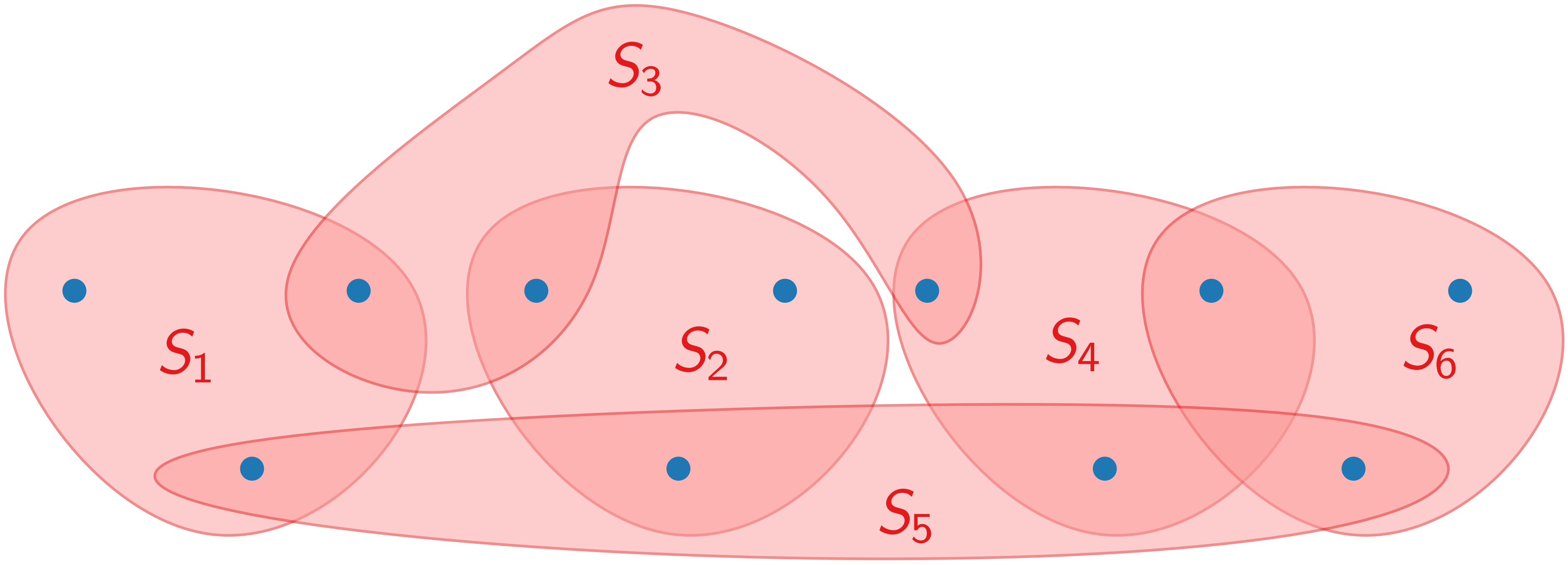
Let $U$ be some **ground set** (universe),
and let $\mathcal{S}$ be a family of **subsets** of $U$ with $\bigcup \mathcal{S} = U$.



# SetCover (card.)

Let $U$ be some **ground set** (universe),
and let $\mathcal{S}$ be a family of **subsets** of $U$ with $\bigcup \mathcal{S} = U$.

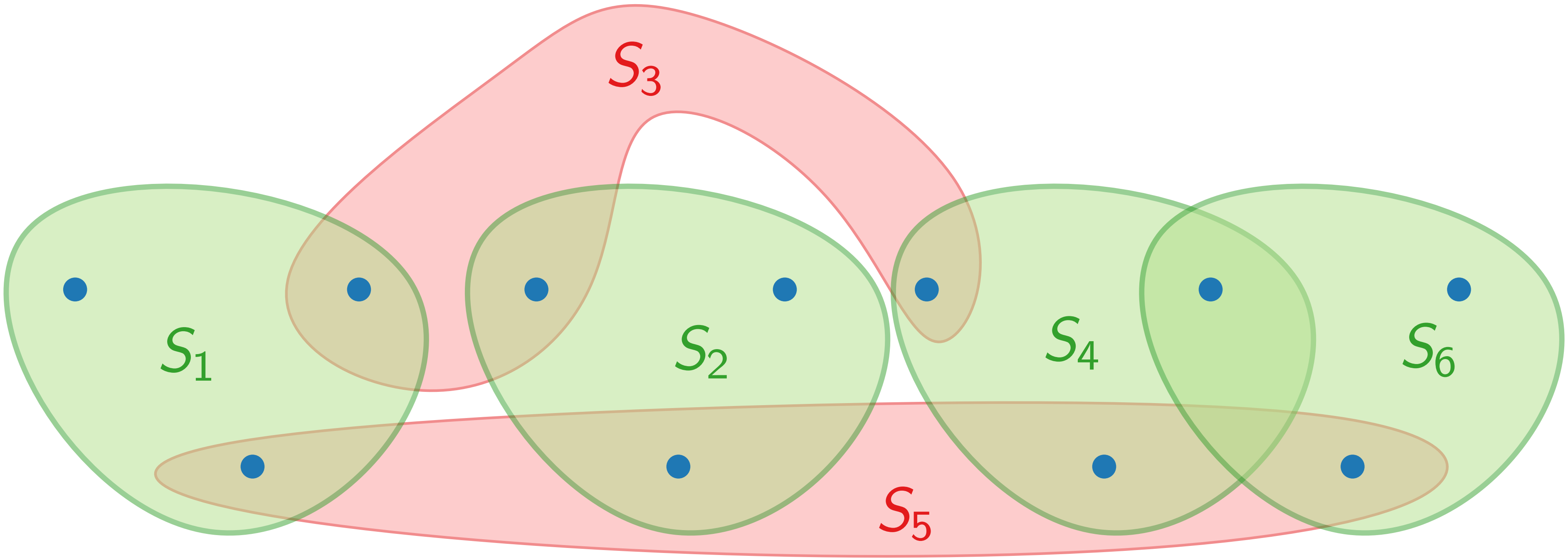
Find a **cover** $\mathcal{S}' \subseteq \mathcal{S}$ of $U$ (i.e., with $\bigcup \mathcal{S}' = U$) of minimum cardinality.



# SetCover (card.)

Let $U$ be some **ground set** (universe),
and let $\mathcal{S}$ be a family of **subsets** of $U$ with $\bigcup \mathcal{S} = U$.

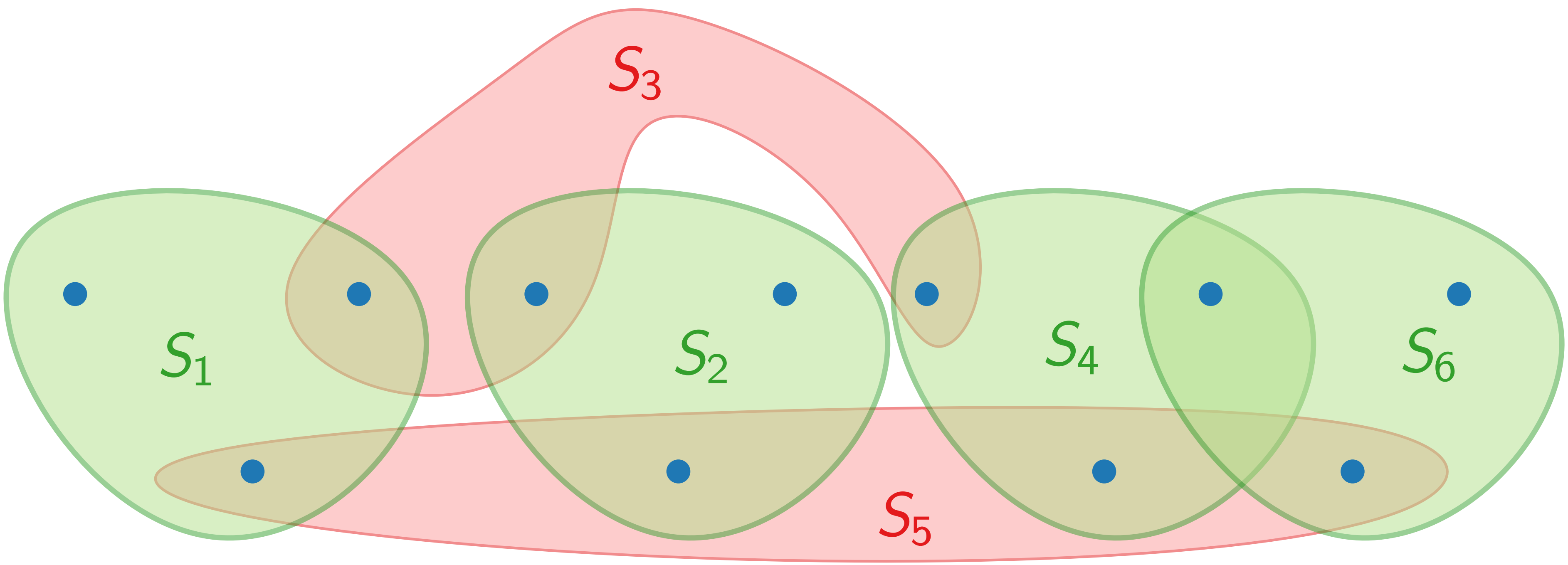
Find a **cover** $\mathcal{S}' \subseteq \mathcal{S}$ of $U$ (i.e., with $\bigcup \mathcal{S}' = U$) of minimum cardinality.



# SetCover (general)

Let $U$ be some **ground set** (universe),
and let $\mathcal{S}$ be a family of **subsets** of $U$ with $\bigcup \mathcal{S} = U$.

Find a **cover** $\mathcal{S}' \subseteq \mathcal{S}$ of $U$ (i.e., with $\bigcup \mathcal{S}' = U$) of minimum cardinality.

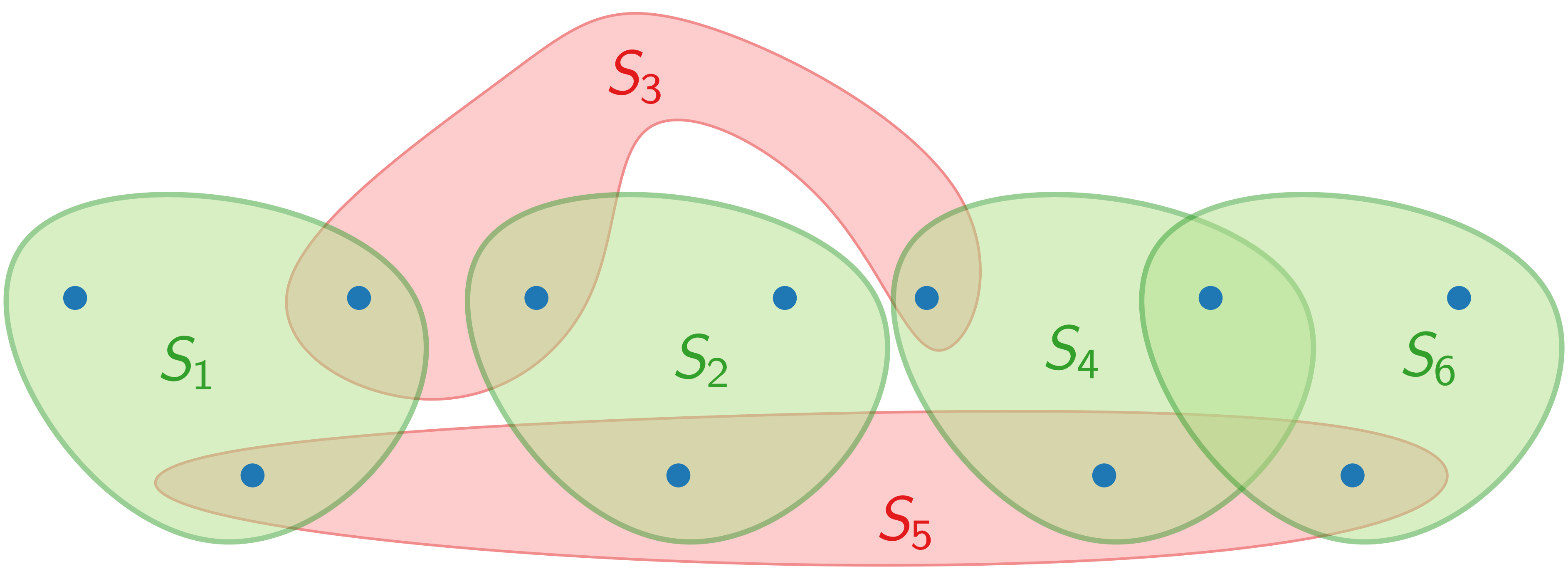


# SetCover (general)

Let $U$ be some **ground set** (universe),
and let $\mathcal{S}$ be a family of **subsets** of $U$ with $\bigcup \mathcal{S} = U$.

Each $S \in \mathcal{S}$ has cost $c(S) > 0$.

Find a **cover** $\mathcal{S}' \subseteq \mathcal{S}$ of $U$ (i.e., with $\bigcup \mathcal{S}' = U$) of minimum cardinality.

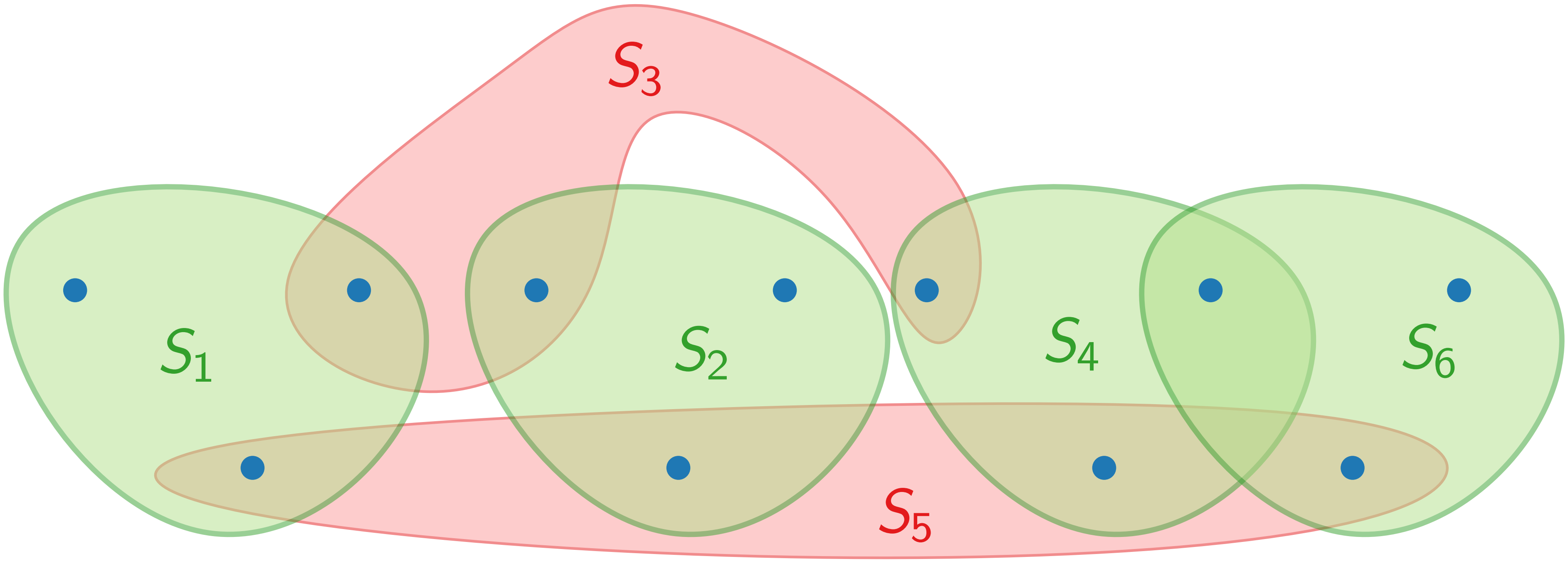


# SetCover (general)

Let $U$ be some **ground set** (universe),
and let $\mathcal{S}$ be a family of **subsets** of $U$ with $\bigcup \mathcal{S} = U$.

Each $S \in \mathcal{S}$ has cost $c(S) > 0$.

Find a **cover** $\mathcal{S}' \subseteq \mathcal{S}$ of $U$ (i.e., with $\bigcup \mathcal{S}' = U$) of minimum ~~cardinality.~~ total cost $c(\mathcal{S}') := \sum_{S \in \mathcal{S}'} c(S)$.

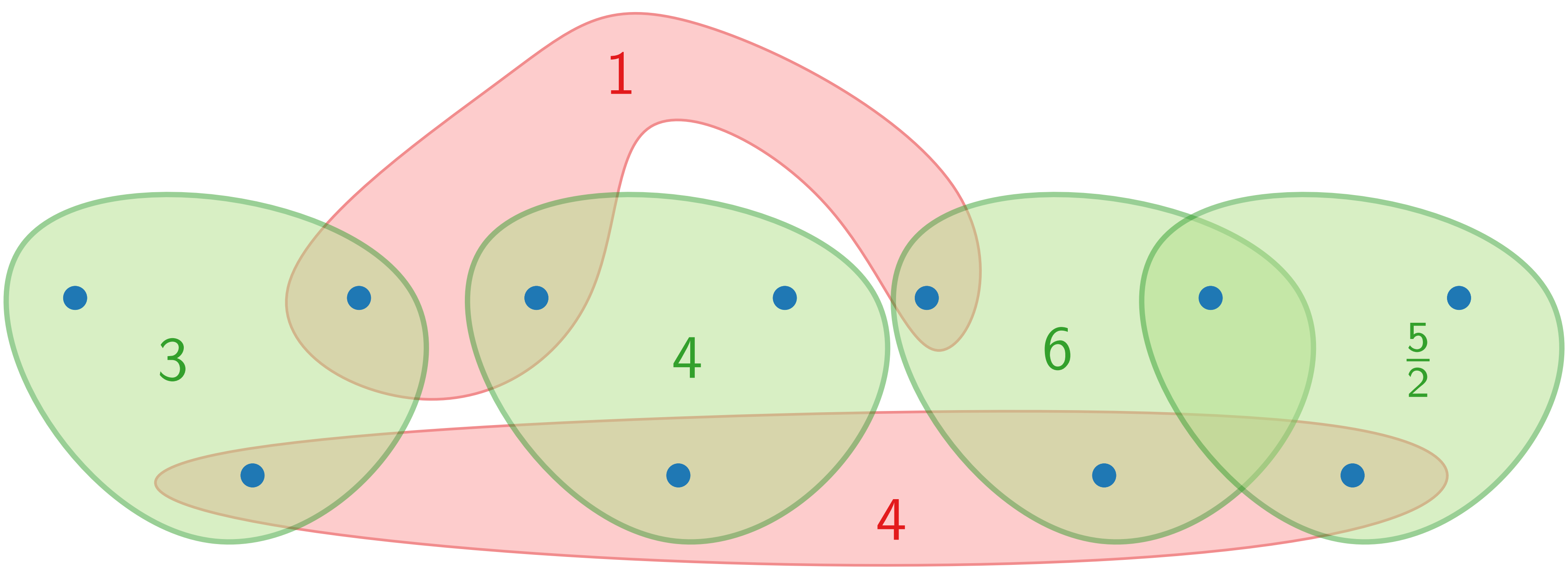


# SetCover (general)

Let $U$ be some **ground set** (universe),
and let $\mathcal{S}$ be a family of **subsets** of $U$ with $\bigcup \mathcal{S} = U$.

Each $S \in \mathcal{S}$ has cost $c(S) > 0$.

Find a **cover** $\mathcal{S}' \subseteq \mathcal{S}$ of $U$ (i.e., with $\bigcup \mathcal{S}' = U$) of minimum ~~cardinality.~~ total cost $c(\mathcal{S}') := \sum_{S \in \mathcal{S}'} c(S)$.

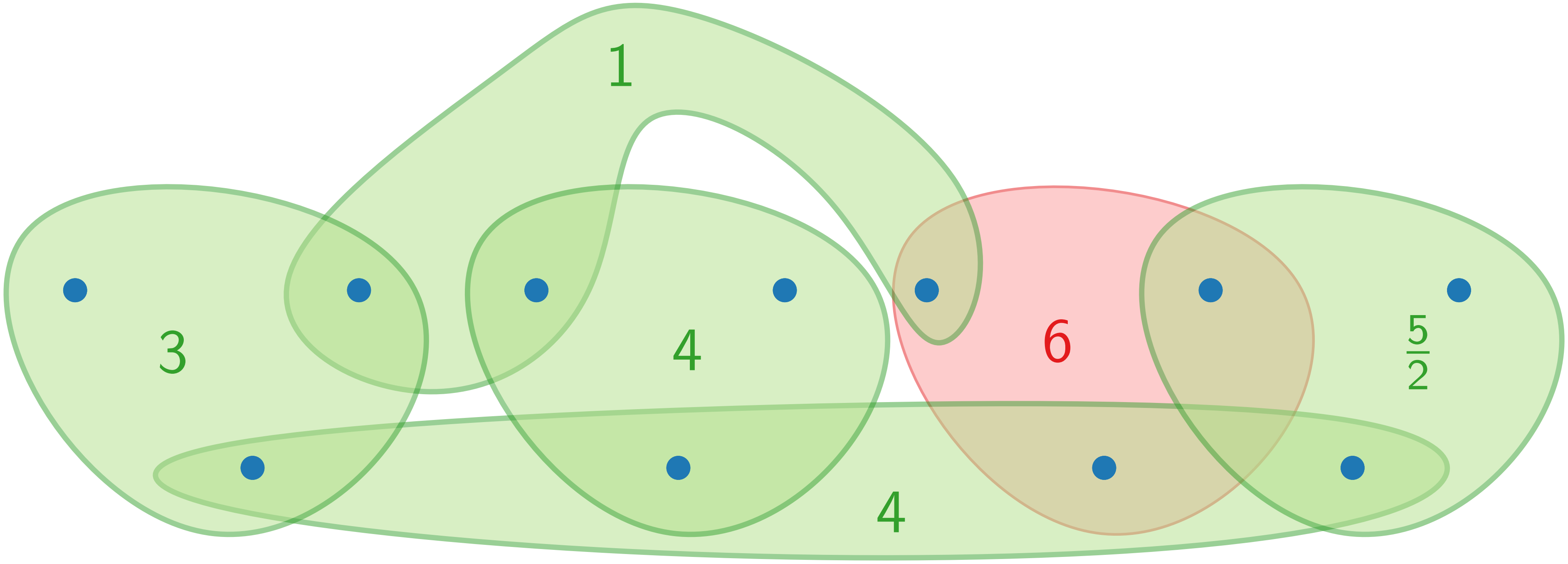


# SetCover (general)

Let $U$ be some **ground set** (universe),
and let $\mathcal{S}$ be a family of **subsets** of $U$ with $\bigcup \mathcal{S} = U$.

Each $S \in \mathcal{S}$ has cost $c(S) > 0$.

Find a **cover** $\mathcal{S}' \subseteq \mathcal{S}$ of $U$ (i.e., with $\bigcup \mathcal{S}' = U$) of minimum ~~cardinality.~~ total cost $c(\mathcal{S}') := \sum_{S \in \mathcal{S}'} c(S)$.



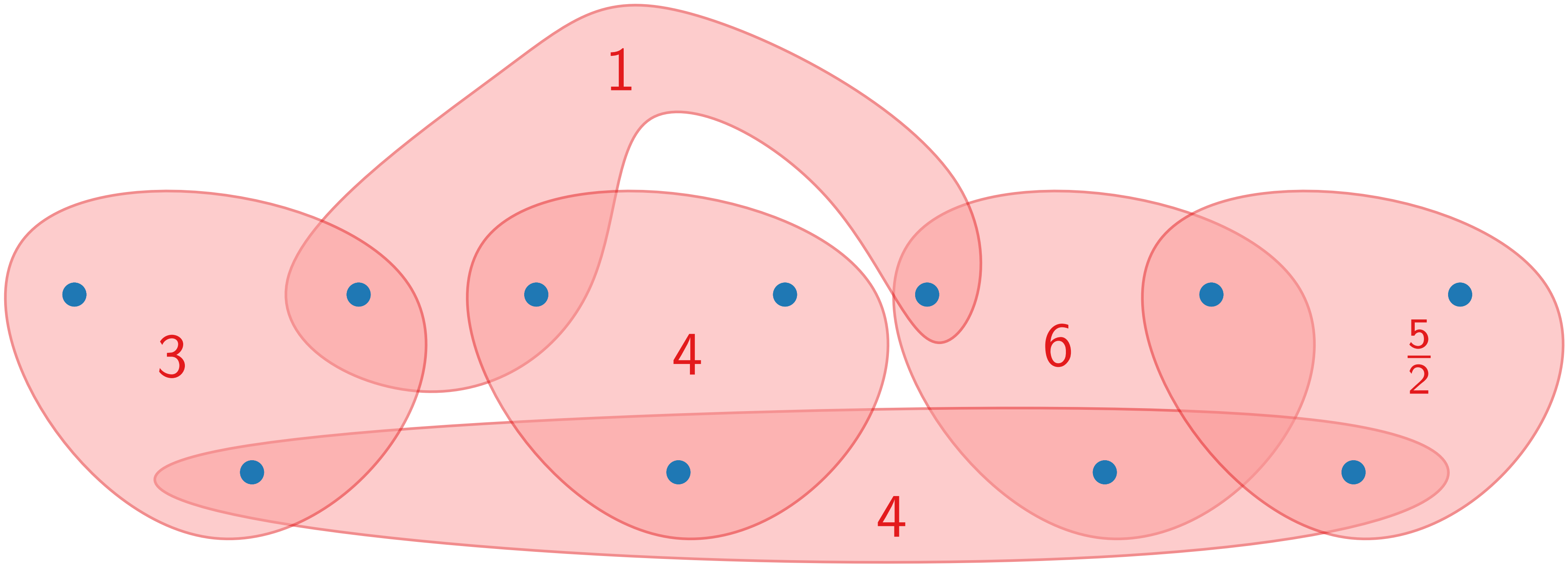
# Approximation Algorithms

## Lecture 2:
## SetCover and ShortestSuperString

### Part II:
### Greedy for SetCover

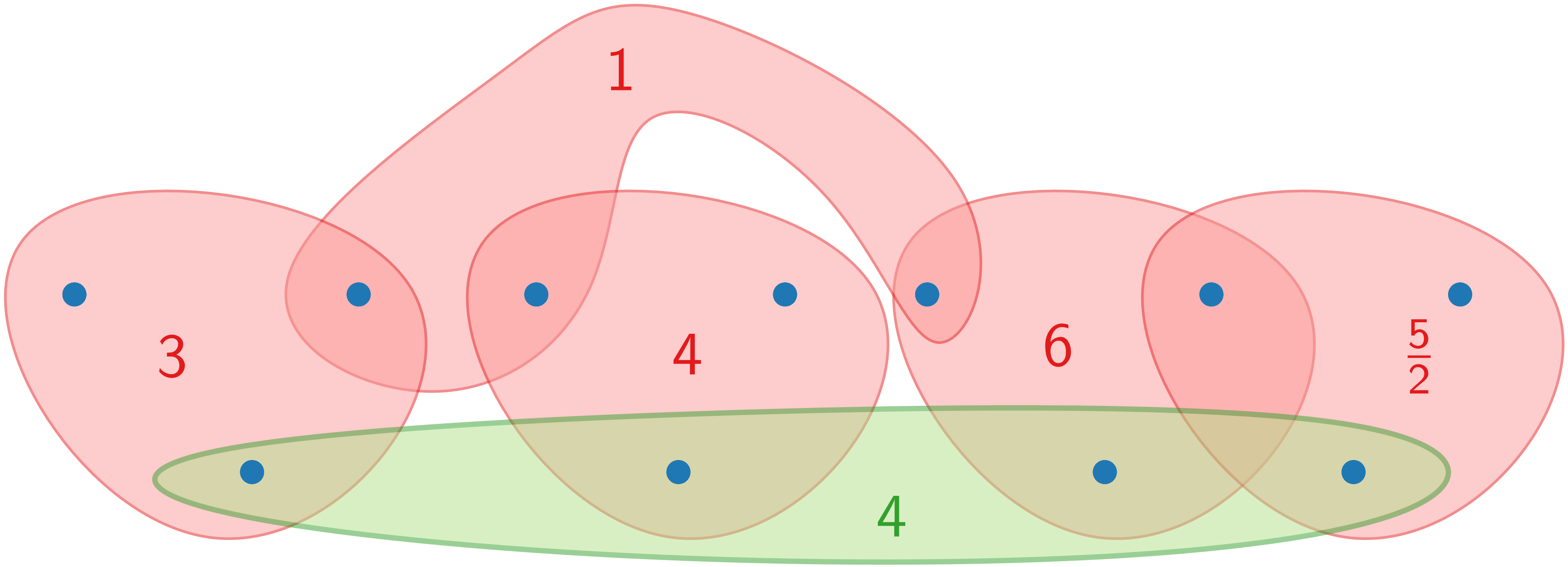
# "Buying" Elements Iteratively

What is the real cost of picking a set?



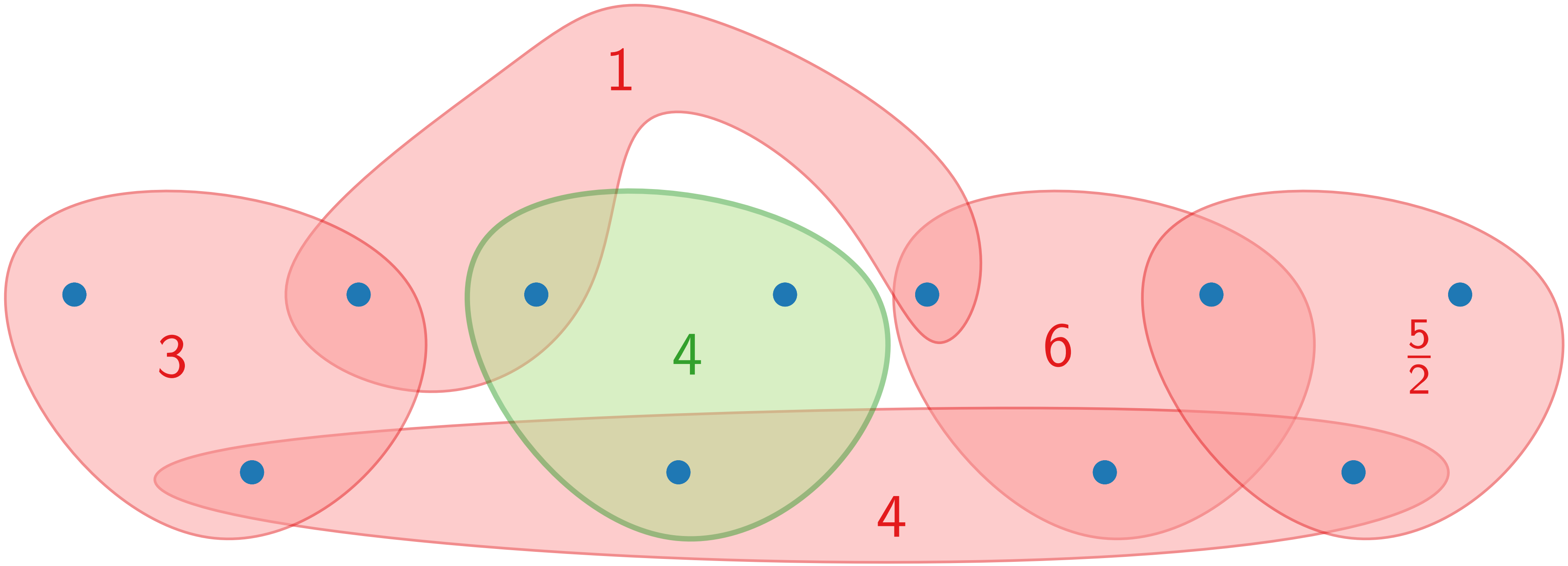
# "Buying" Elements Iteratively

What is the real cost of picking a set?



# "Buying" Elements Iteratively

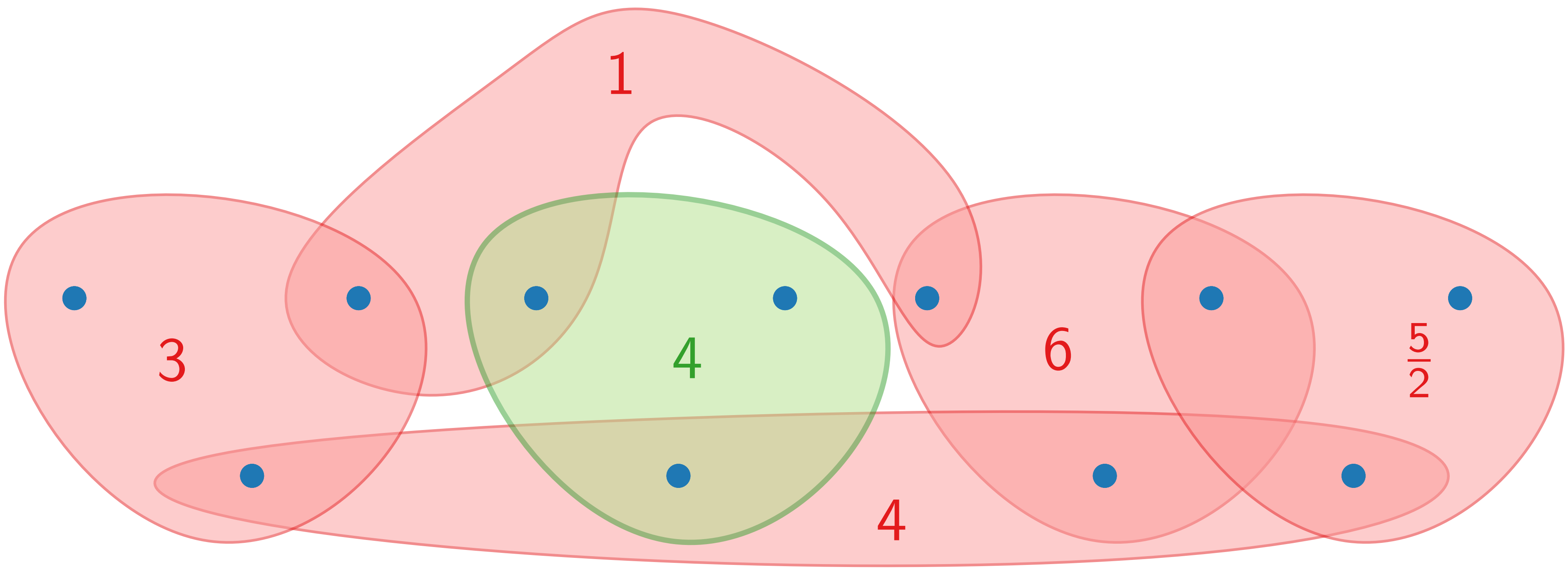
What is the real cost of picking a set?



# "Buying" Elements Iteratively

What is the real cost of picking a set?

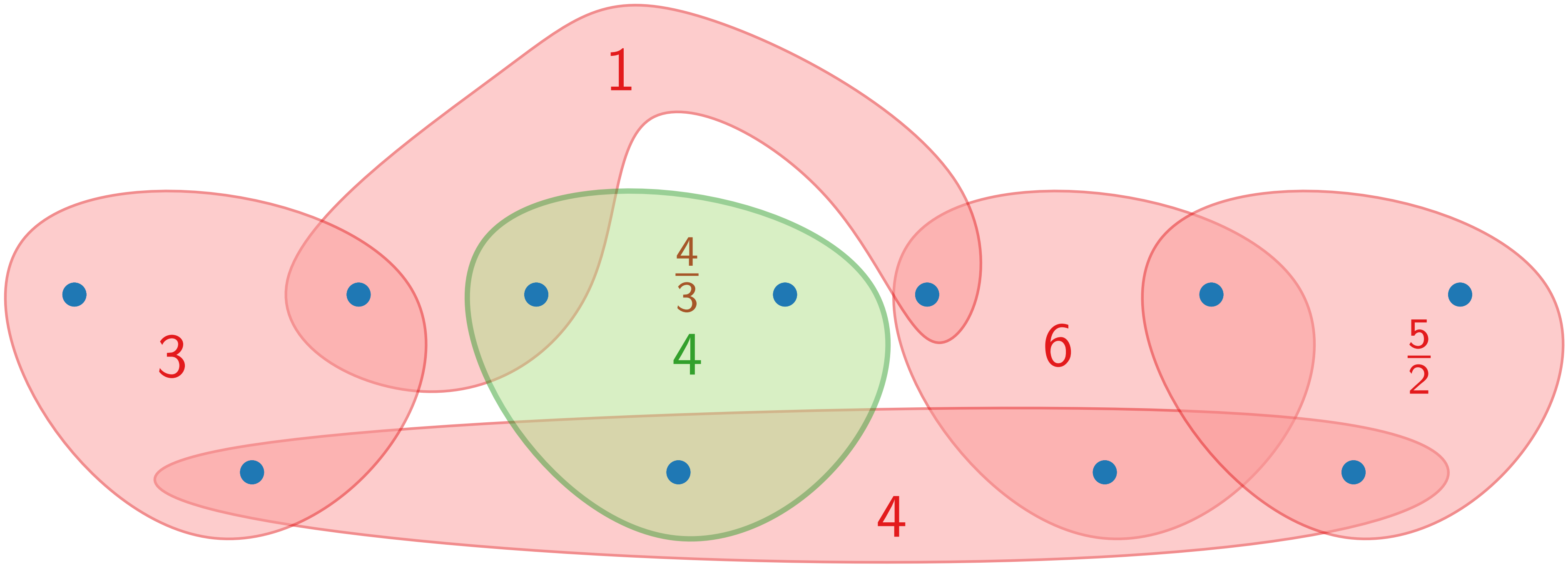
Set with $k$ elements and cost $c$ has per-element cost $c/k$.



# "Buying" Elements Iteratively

What is the real cost of picking a set?

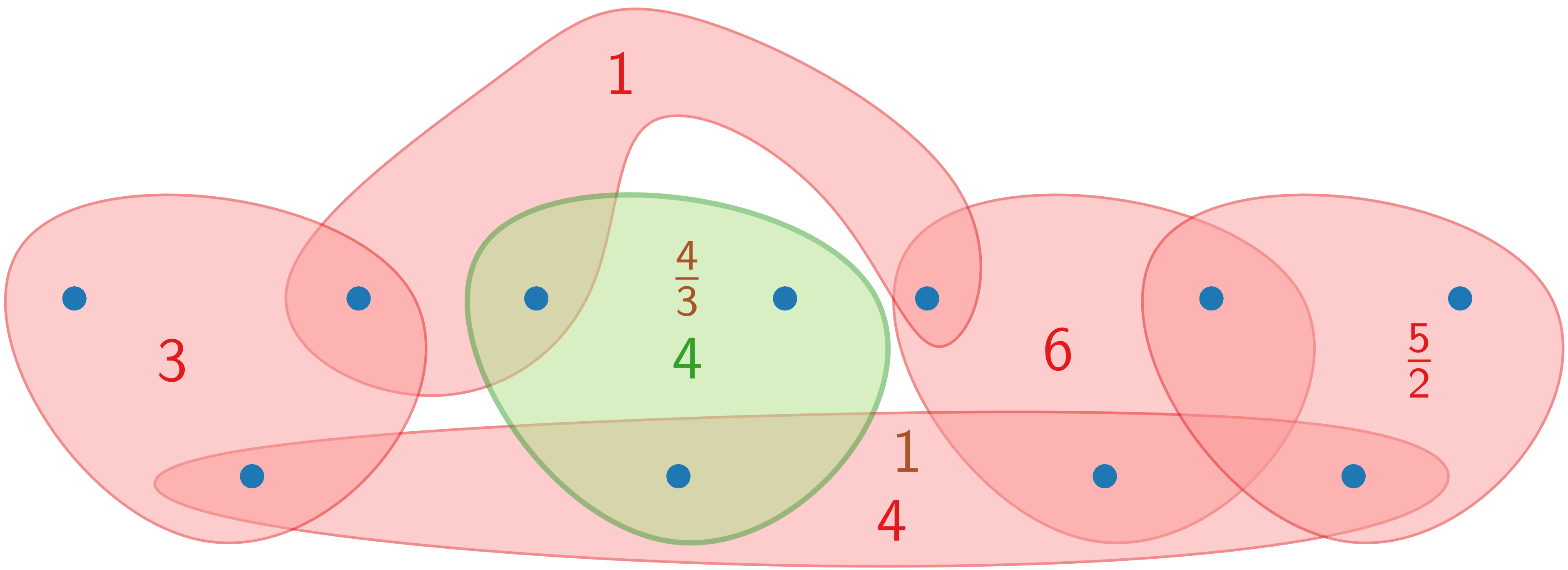
Set with $k$ elements and cost $c$ has per-element cost $c/k$.



# "Buying" Elements Iteratively

What is the real cost of picking a set?

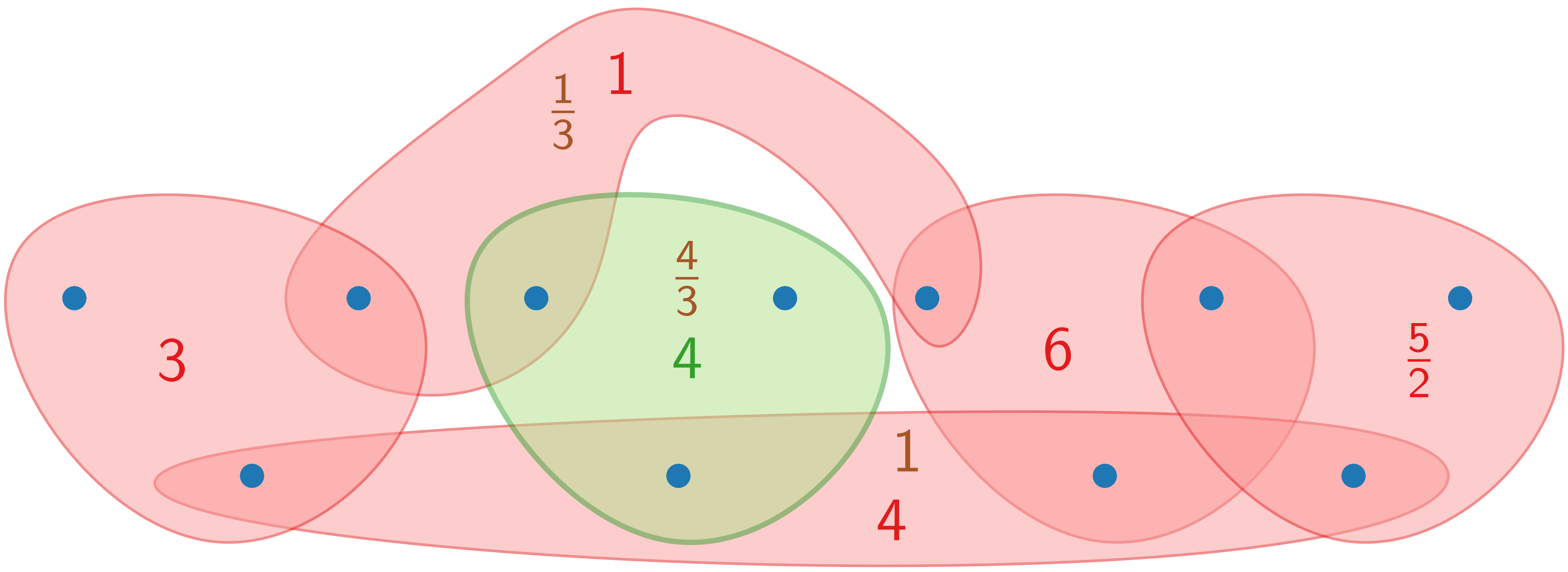
Set with $k$ elements and cost $c$ has per-element cost $c/k$.



# "Buying" Elements Iteratively

What is the real cost of picking a set?

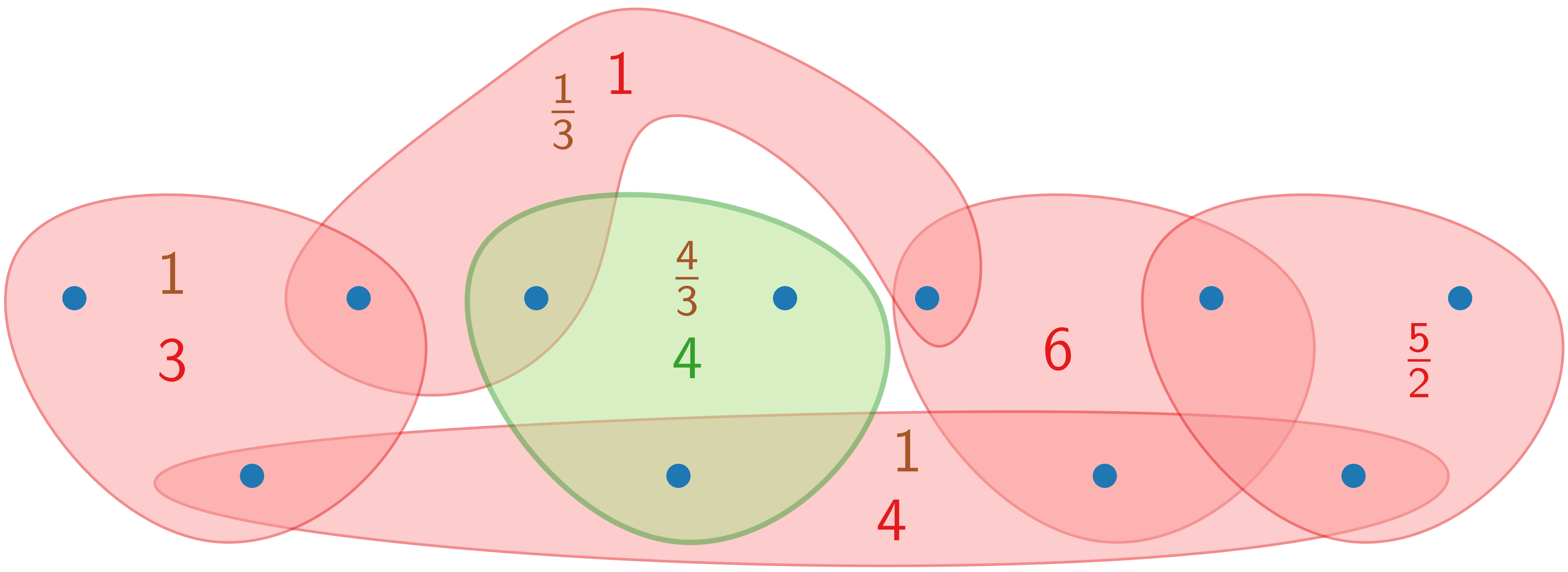
Set with $k$ elements and cost $c$ has per-element cost $c/k$.



# "Buying" Elements Iteratively

What is the real cost of picking a set?

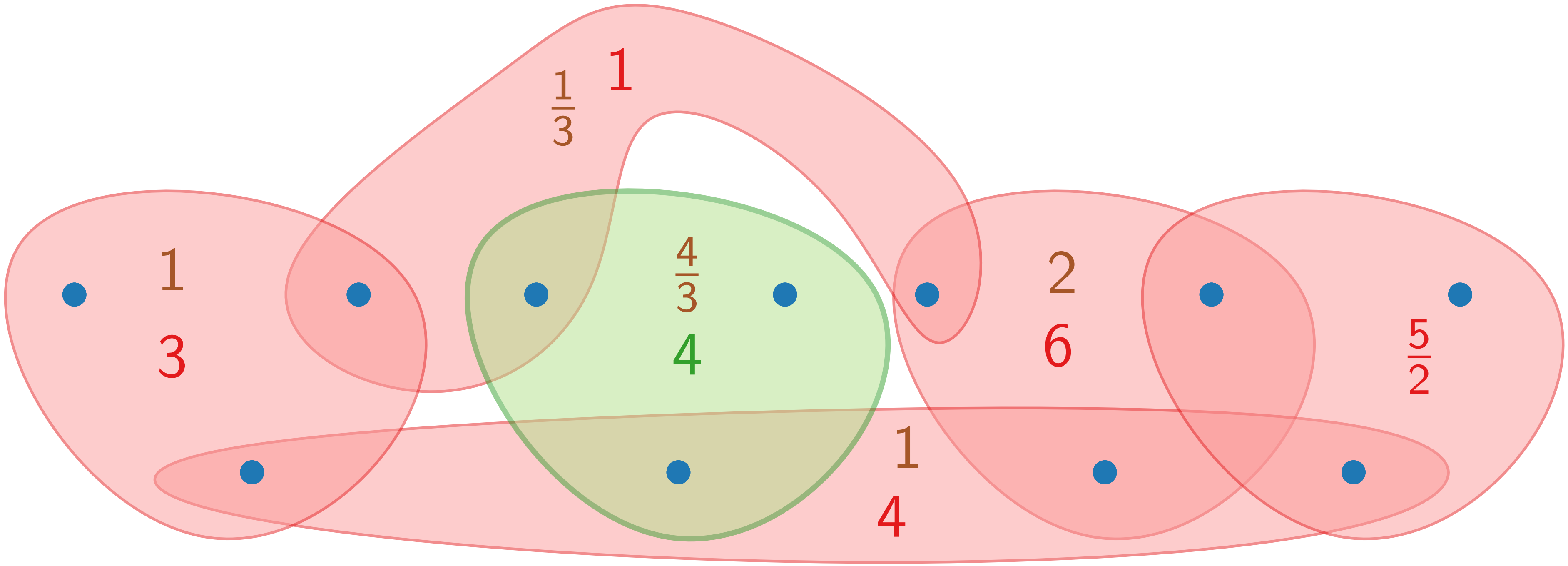
Set with $k$ elements and cost $c$ has per-element cost $c/k$.



# "Buying" Elements Iteratively

What is the real cost of picking a set?

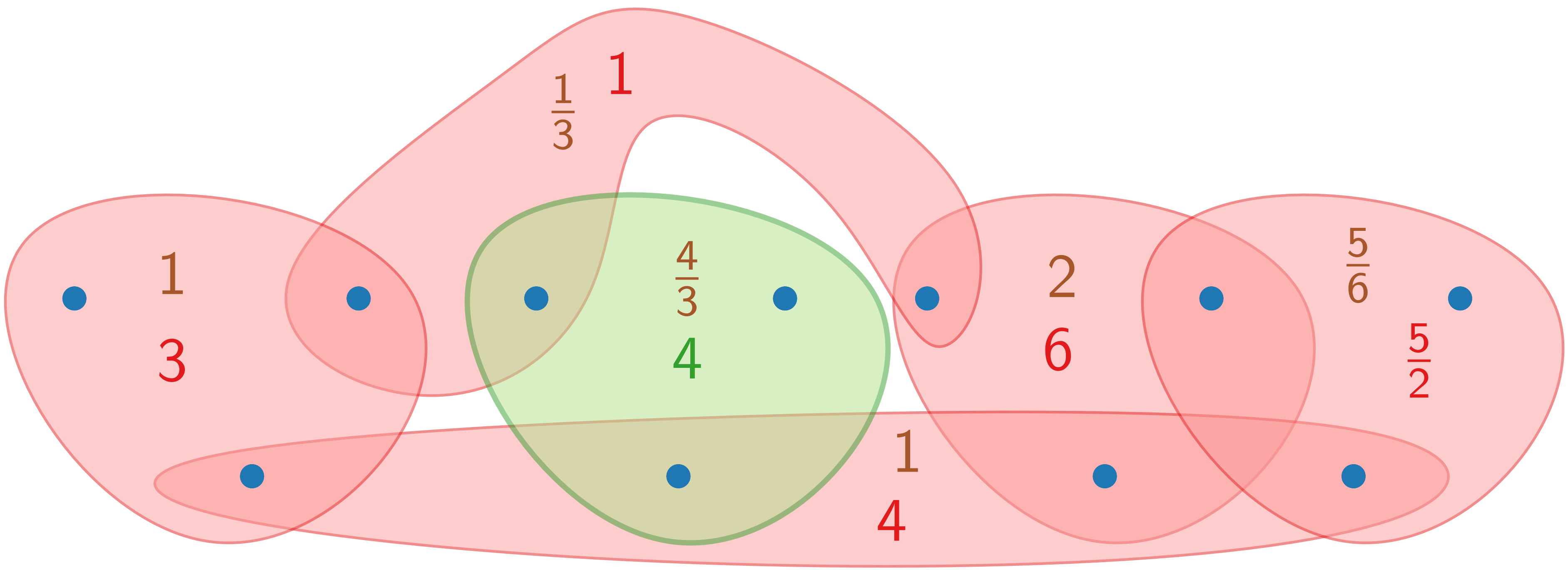
Set with $k$ elements and cost $c$ has per-element cost $c/k$.



# "Buying" Elements Iteratively

What is the real cost of picking a set?

Set with $k$ elements and cost $c$ has per-element cost $c/k$.

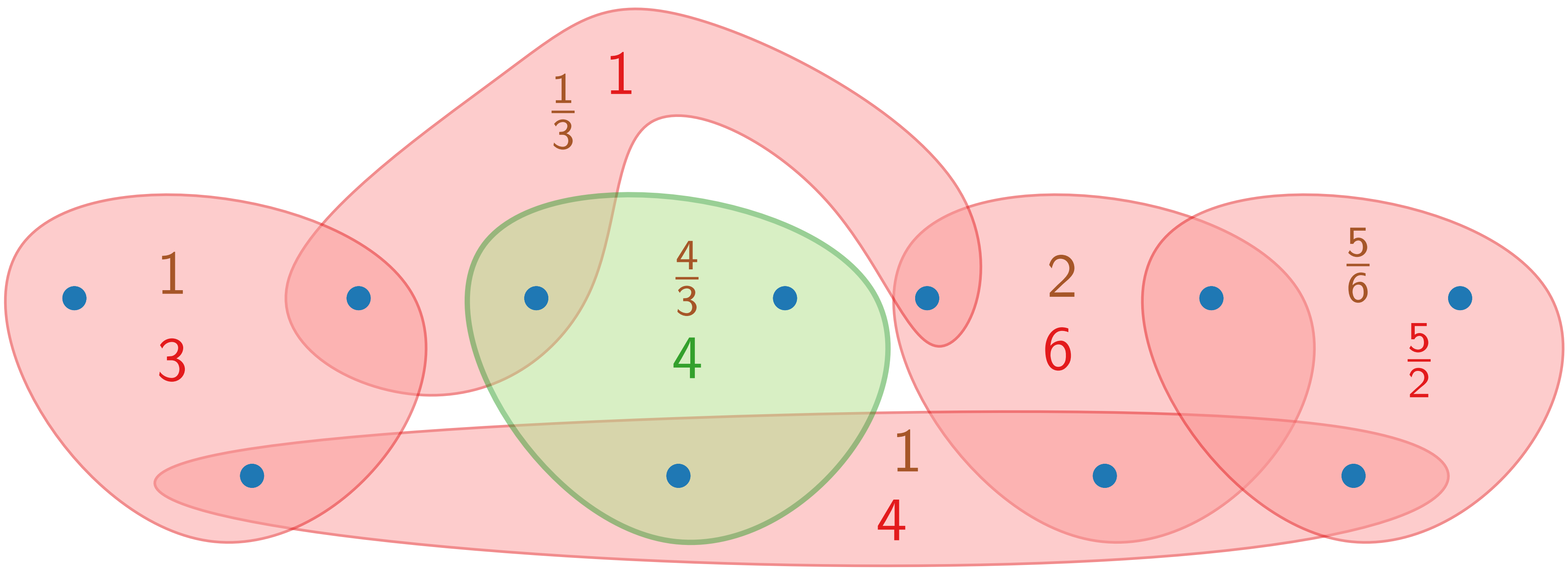


# "Buying" Elements Iteratively

What is the real cost of picking a set?

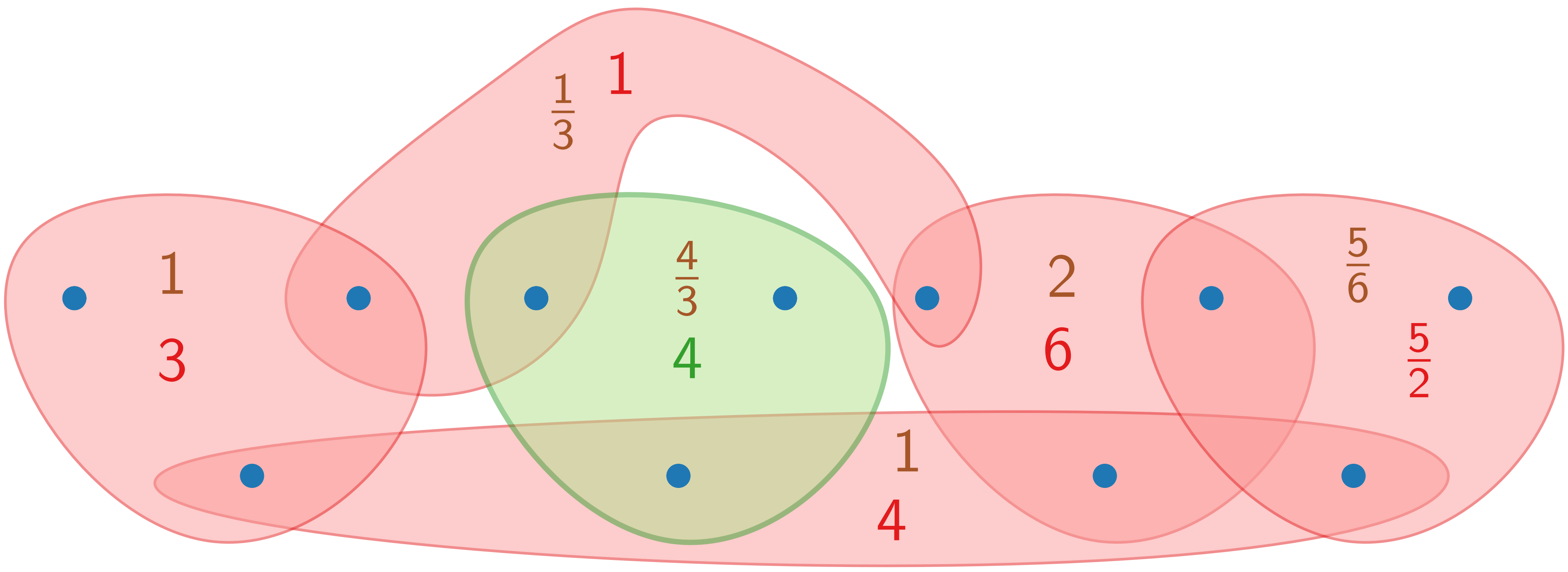
Set with $k$ elements and cost $c$ has per-element cost $c/k$.

What happens if we "buy" a set?



# "Buying" Elements Iteratively

What is the real cost of picking a set?
Set with $k$ elements and cost $c$ has per-element cost $c/k$.
What happens if we "buy" a set?
Fix price of elements bought and recompute per-element cost.



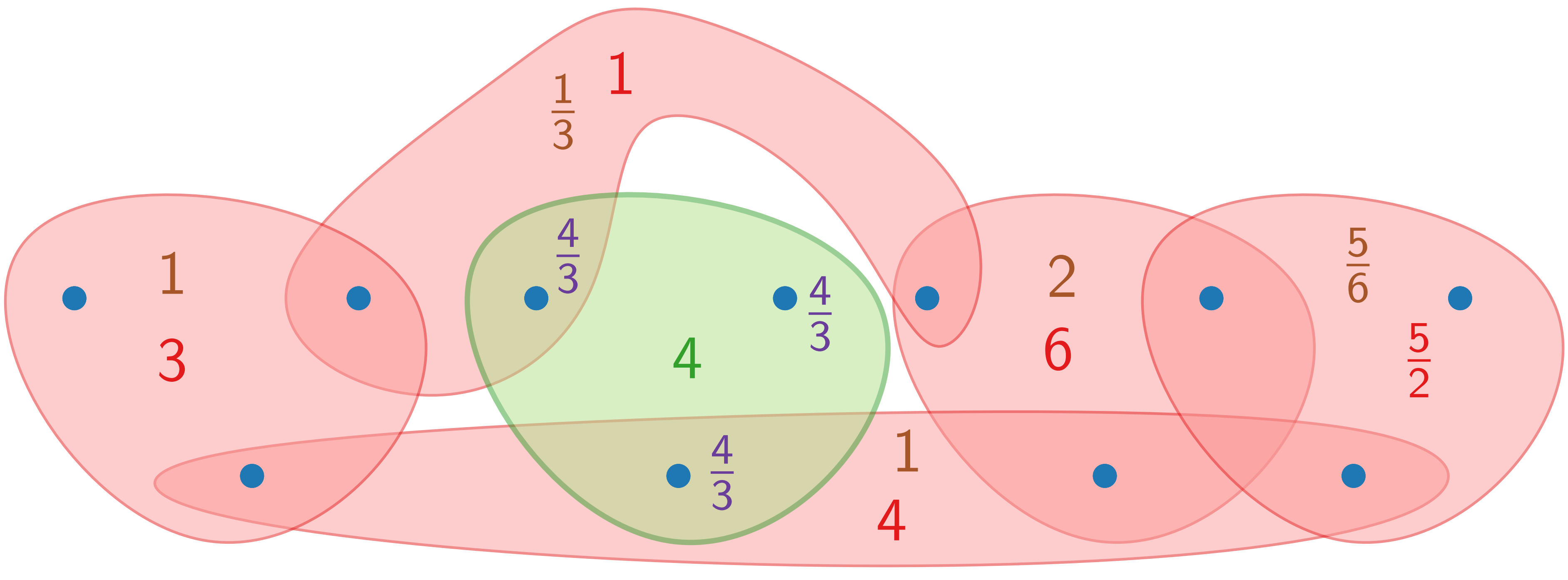
# "Buying" Elements Iteratively

What is the real cost of picking a set?

Set with $k$ elements and cost $c$ has per-element cost $c/k$.

What happens if we "buy" a set?

Fix price of elements bought and recompute per-element cost.



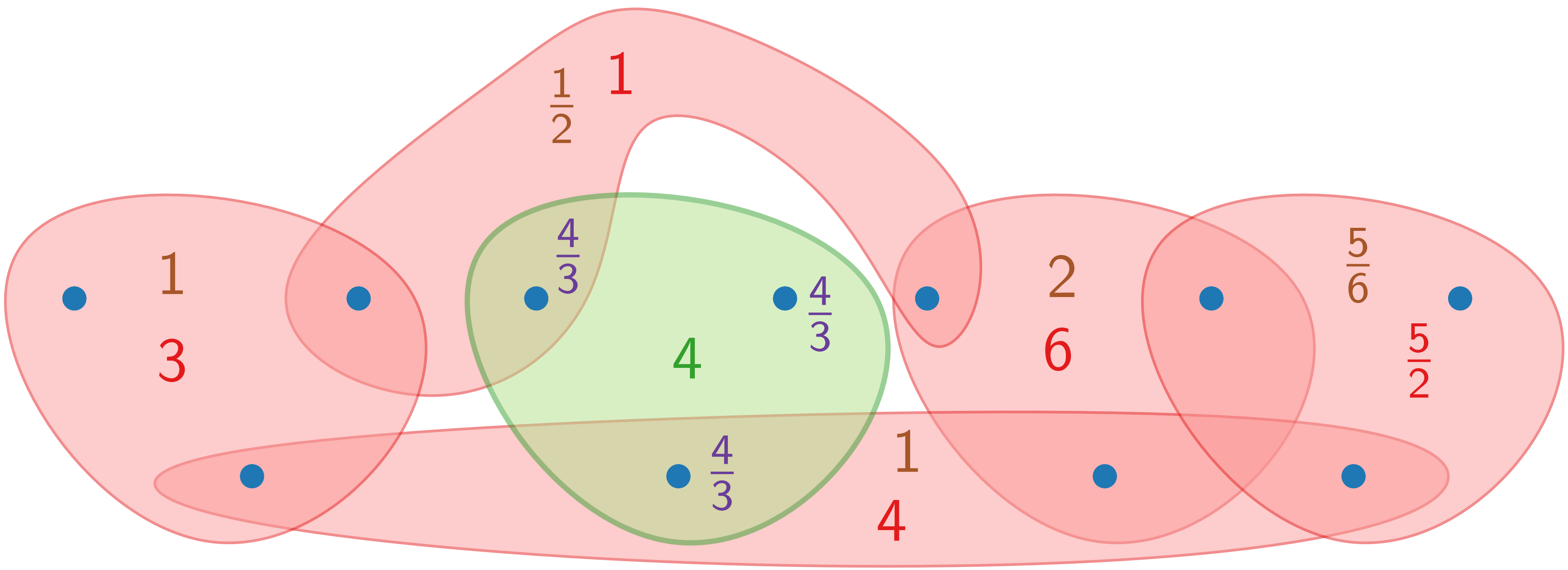
# "Buying" Elements Iteratively

What is the real cost of picking a set?

Set with $k$ elements and cost $c$ has per-element cost $c/k$.

What happens if we "buy" a set?

Fix price of elements bought and recompute per-element cost.



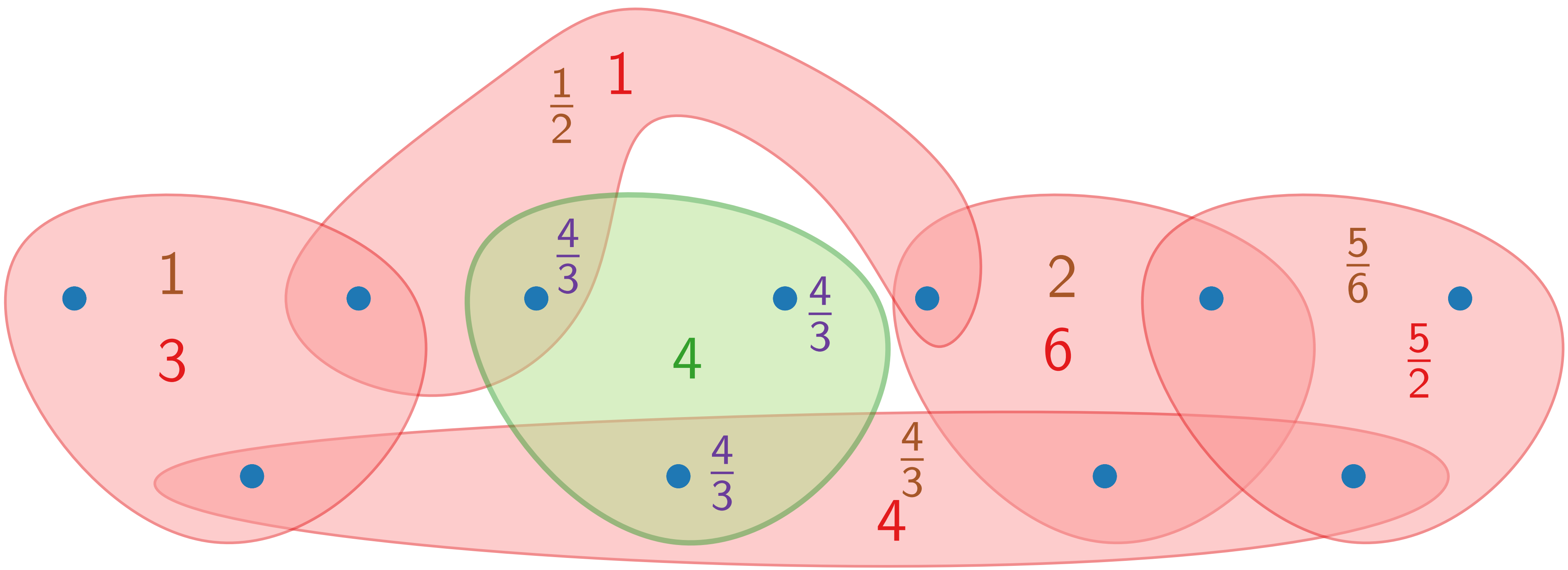
# “Buying” Elements Iteratively

What is the real cost of picking a set?

Set with $k$ elements and cost $c$ has per-element cost $c/k$.

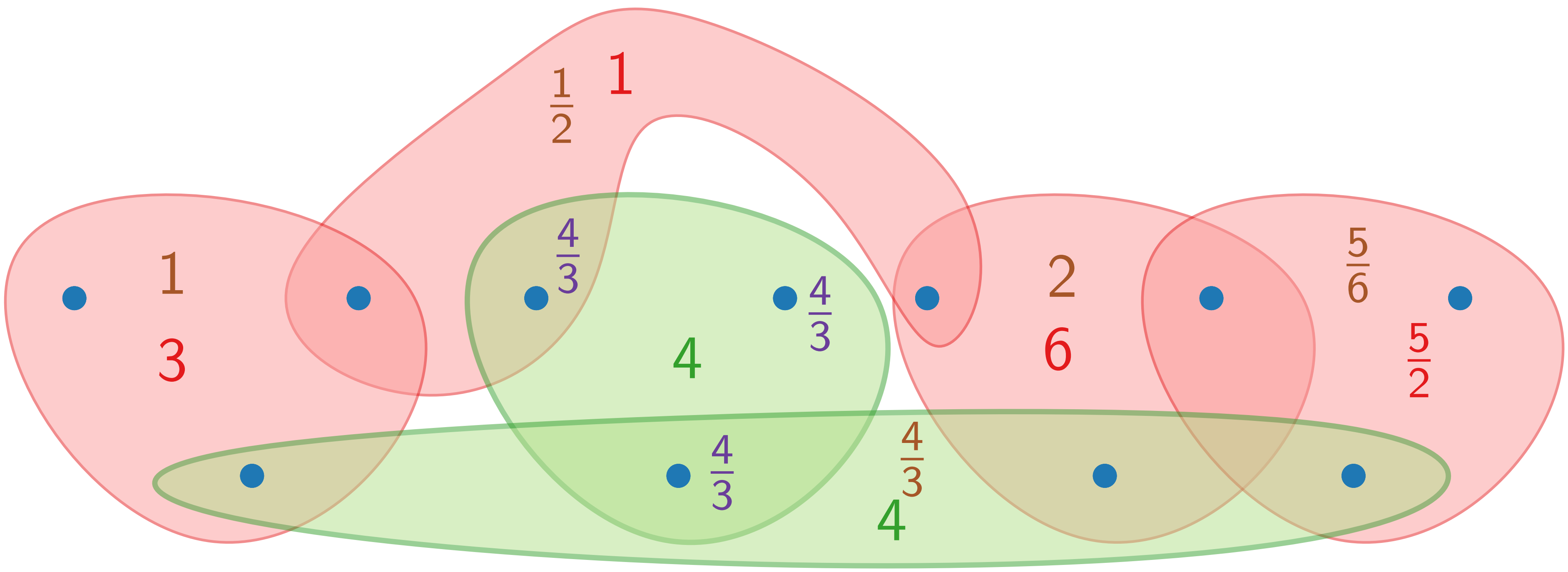
What happens if we “buy” a set?

Fix price of elements bought and recompute per-element cost.



# "Buying" Elements Iteratively

What is the real cost of picking a set?
Set with $k$ elements and cost $c$ has per-element cost $c/k$.
What happens if we "buy" a set?
Fix price of elements bought and recompute per-element cost.



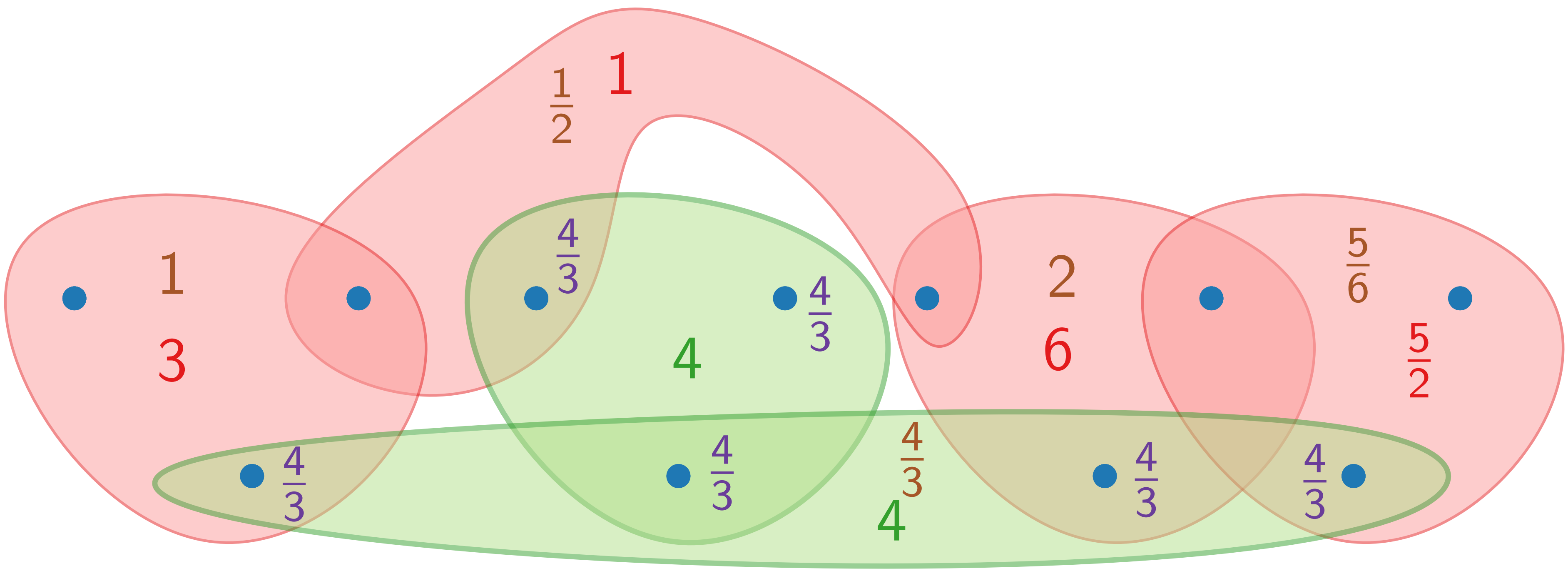
# "Buying" Elements Iteratively

What is the real cost of picking a set?

Set with $k$ elements and cost $c$ has per-element cost $c/k$.

What happens if we "buy" a set?

Fix price of elements bought and recompute per-element cost.



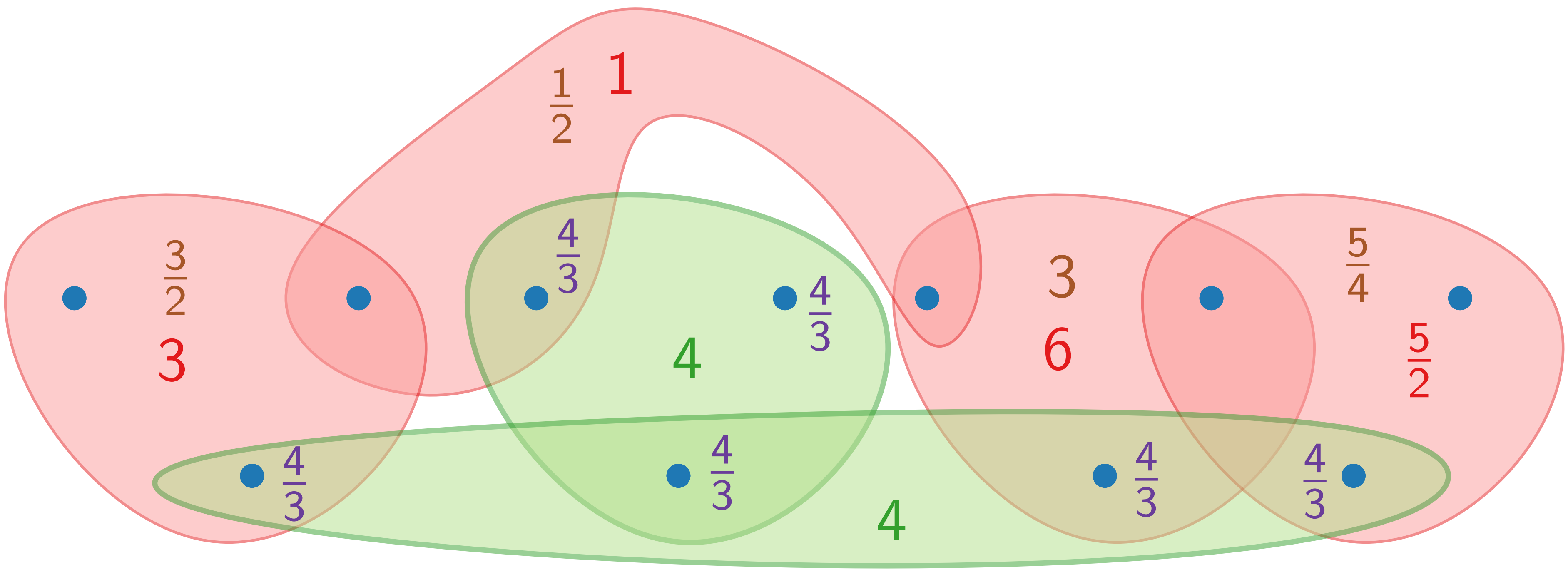
# "Buying" Elements Iteratively

What is the real cost of picking a set?

Set with $k$ elements and cost $c$ has per-element cost $c/k$.

What happens if we "buy" a set?

Fix price of elements bought and recompute per-element cost.



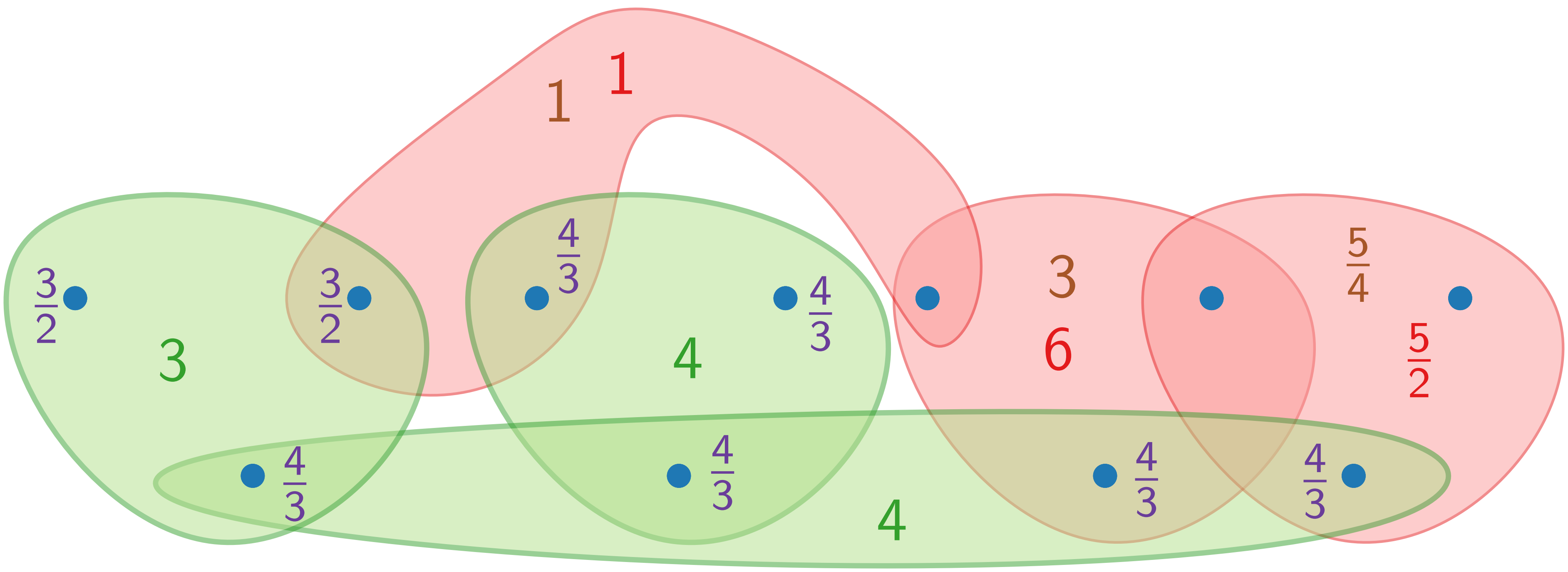
# “Buying” Elements Iteratively

What is the real cost of picking a set?

Set with $k$ elements and cost $c$ has per-element cost $c/k$.

What happens if we “buy” a set?

Fix price of elements bought and recompute per-element cost.



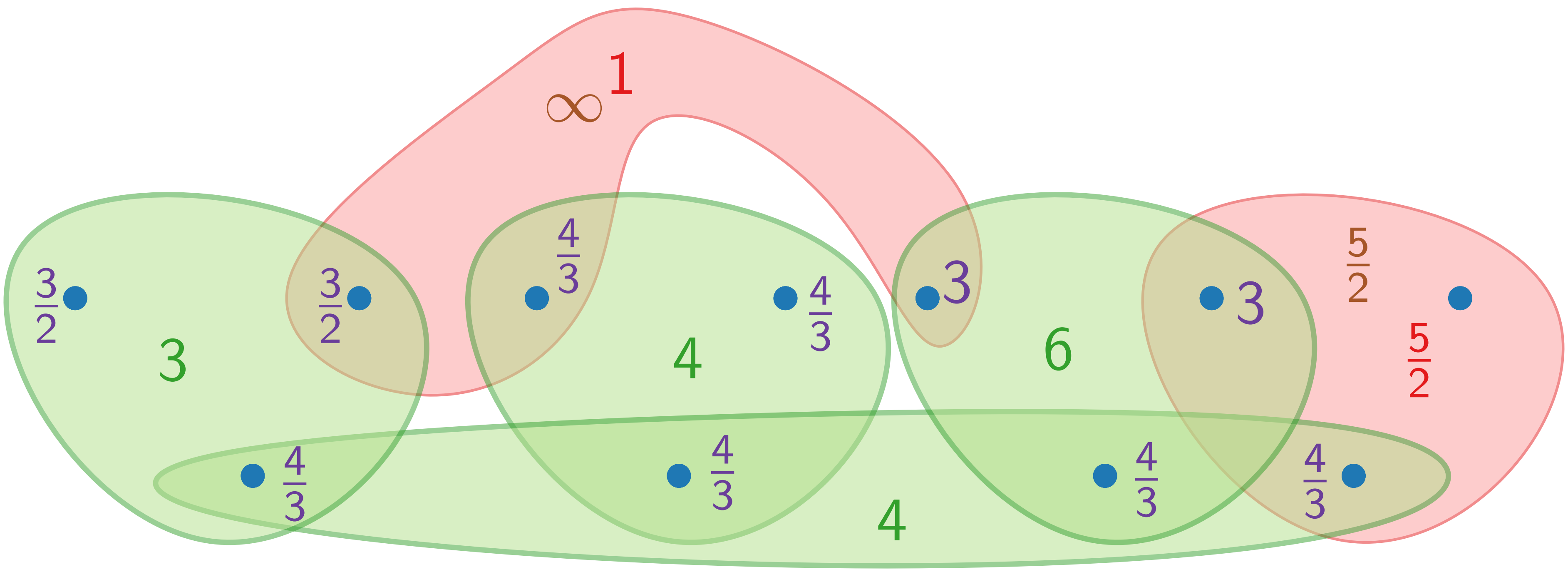
# "Buying" Elements Iteratively

What is the real cost of picking a set?

Set with $k$ elements and cost $c$ has per-element cost $c/k$.

What happens if we "buy" a set?

Fix price of elements bought and recompute per-element cost.



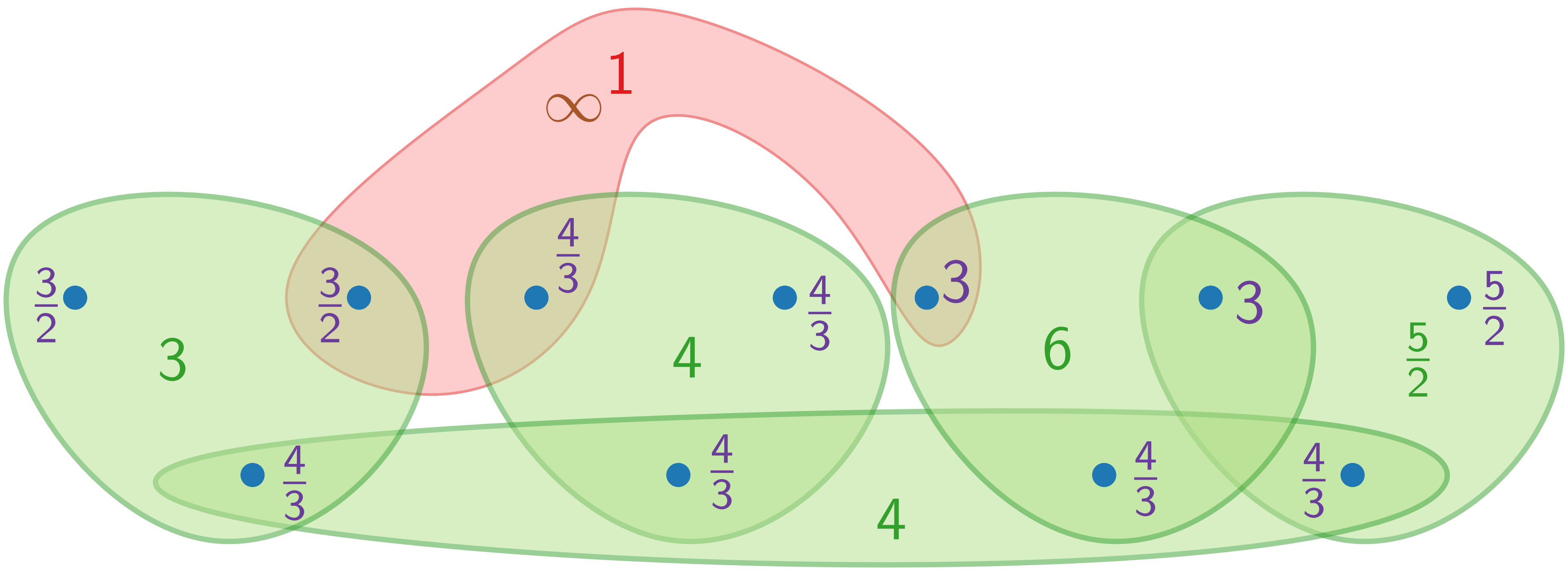
# "Buying" Elements Iteratively

What is the real cost of picking a set?

Set with $k$ elements and cost $c$ has per-element cost $c/k$.

What happens if we "buy" a set?

Fix price of elements bought and recompute per-element cost.



# "Buying" Elements Iteratively

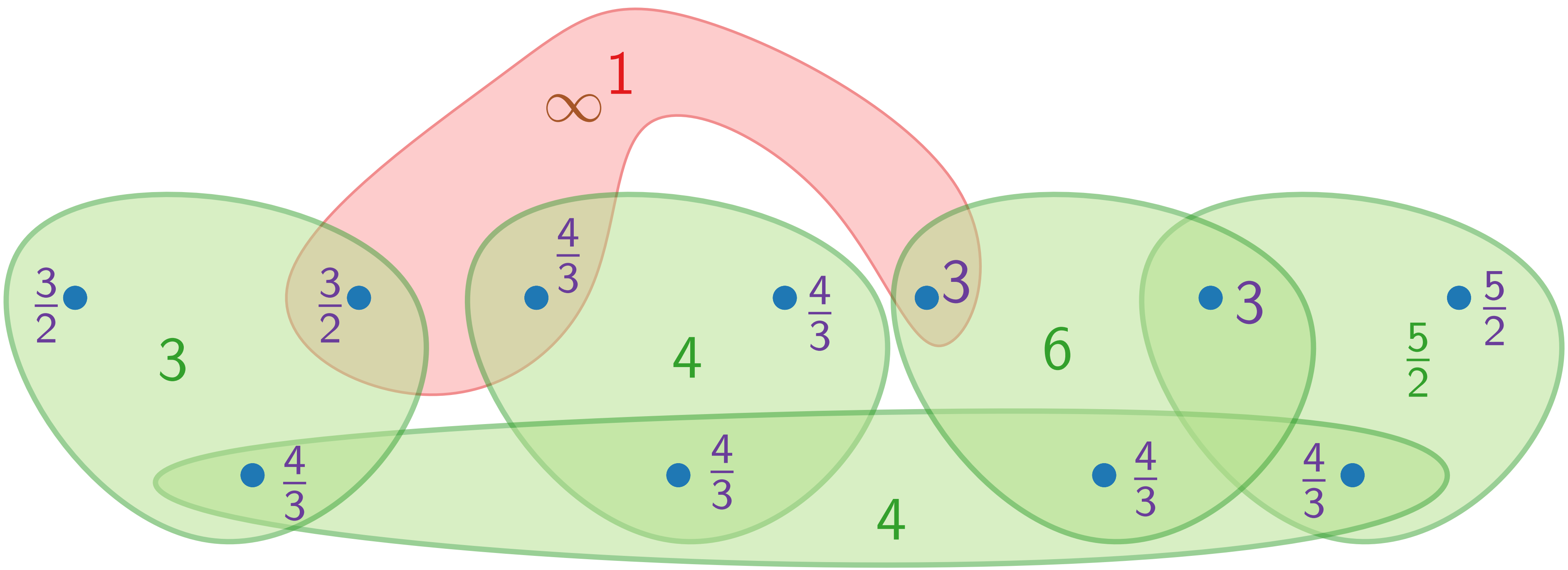
What is the real cost of picking a set?

Set with $k$ elements and cost $c$ has per-element cost $c/k$.

What happens if we "buy" a set?

Fix price of elements bought and recompute per-element cost.

total cost: $\sum_{u \in U} \text{price}(u)$



# "Buying" Elements Iteratively

What is the real cost of picking a set?

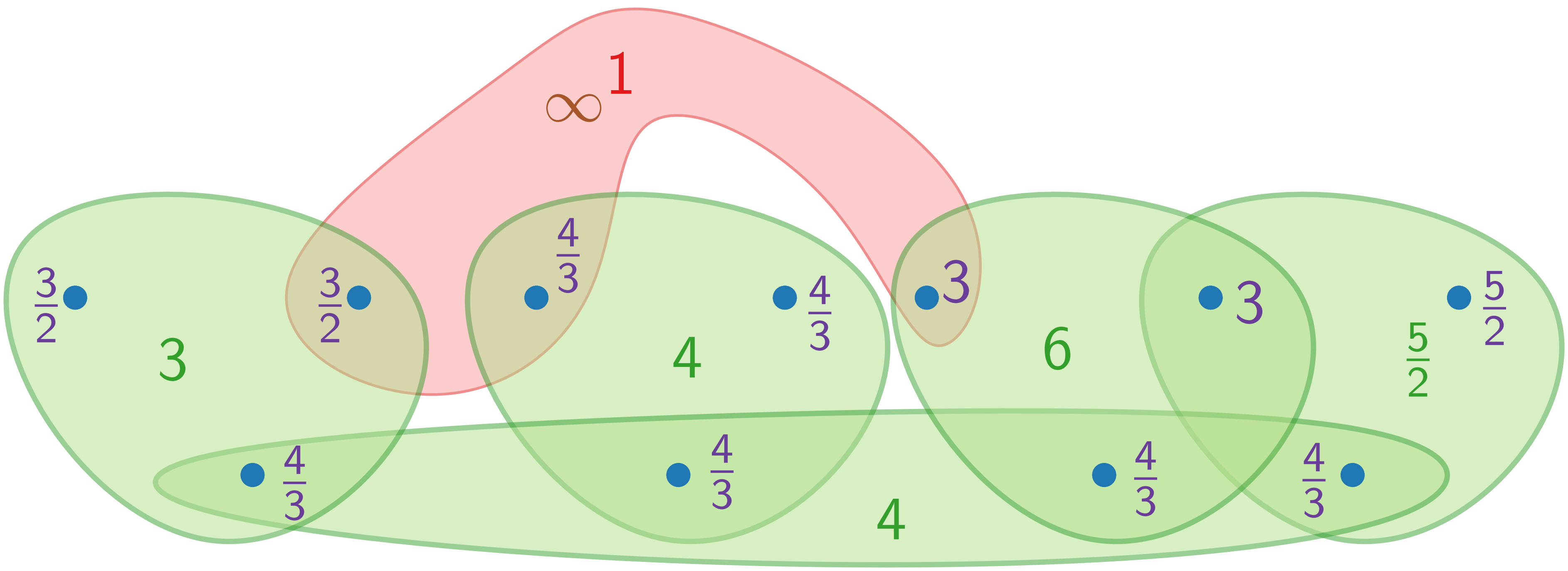
Set with $k$ elements and cost $c$ has per-element cost $c/k$.

What happens if we "buy" a set?

Fix price of elements bought and recompute per-element cost.

total cost: $\sum_{u \in U} \text{price}(u)$

Greedy: Always choose the set with minimum per-element cost.



# Greedy for SetCover

GreedySetCover($U, \mathcal{S}, c$)

$C \leftarrow \emptyset$

$\mathcal{S}' \leftarrow \emptyset$

**return** $\mathcal{S}'$ `// Cover of` $U$

# Greedy for SetCover

```
GreedySetCover(U, 𝒮, c)
  C ← ∅
  𝒮' ← ∅
  while C ≠ U do

  return 𝒮'                                   // Cover of U
```

# Greedy for SetCover

GreedySetCover($U, \mathcal{S}, c$)

$C \leftarrow \emptyset$

$\mathcal{S}' \leftarrow \emptyset$

**while** $C \neq U$ **do**

  $S \leftarrow$ set in $\mathcal{S}$ that minimizes $\frac{c(S)}{|S \setminus C|}$

**return** $\mathcal{S}'$ // Cover of $U$

# Greedy for SetCover

GreedySetCover($U, \mathcal{S}, c$)

$C \leftarrow \emptyset$

$\mathcal{S}' \leftarrow \emptyset$

**while** $C \neq U$ **do**

  $S \leftarrow$ set in $\mathcal{S}$ that minimizes $\frac{c(S)}{|S \setminus C|}$

  **foreach** $u \in S \setminus C$ **do**

**return** $\mathcal{S}'$ // Cover of $U$

# Greedy for SetCover

GreedySetCover($U, \mathcal{S}, c$)

$C \leftarrow \emptyset$

$\mathcal{S}' \leftarrow \emptyset$

**while** $C \neq U$ **do**

  $S \leftarrow$ set in $\mathcal{S}$ that minimizes $\frac{c(S)}{|S \setminus C|}$

  **foreach** $u \in S \setminus C$ **do**

    price($u$) $\leftarrow \frac{c(S)}{|S \setminus C|}$

**return** $\mathcal{S}'$ `// Cover of` $U$

# Greedy for SetCover

```
GreedySetCover(U, 𝒮, c)
  C ← ∅
  𝒮' ← ∅
  while C ≠ U do
      S ← set in 𝒮 that minimizes c(S)/|S \ C|
      foreach u ∈ S \ C do
          price(u) ← c(S)/|S \ C|
      C ← C ∪ S
  return 𝒮'                                  // Cover of U
```

# Greedy for SetCover

```
GreedySetCover(U, 𝒮, c)
  C ← ∅
  𝒮' ← ∅
  while C ≠ U do
      S ← set in 𝒮 that minimizes c(S)/|S \ C|
      foreach u ∈ S \ C do
          price(u) ← c(S)/|S \ C|
      C ← C ∪ S
      𝒮' ← 𝒮' ∪ {S}
  return 𝒮'                                  // Cover of U
```

# Approximation Algorithms

## Lecture 2:
## SetCover and ShortestSuperString

### Part III:
### Analysis

# Analysis

**Theorem.** GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for $\textsc{SetCover}$, where $k$ is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \to 0.5 + \ln k$.

# Analysis

**Theorem.** GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for $\textsc{SetCover}$, where $k$ is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \to 0.5 + \ln k$.

**Lemma.** Let $S \in \mathcal{S}$, and let $u_1, \ldots, u_\ell$ be the elements of $S$ in the order in which they are covered ("bought") by GreedySetCover. Then, for every $j \in \{1, \ldots, \ell\}$: $\text{price}(u_j) \leq$

# Analysis

**Theorem.** GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for $\textsc{SetCover}$, where $k$ is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \to 0.5 + \ln k$.

**Lemma.** Let $S \in \mathcal{S}$, and let $u_1, \ldots, u_\ell$ be the elements of $S$ in the order in which they are covered ("bought") by GreedySetCover. Then, for every $j \in \{1, \ldots, \ell\}$: $\text{price}(u_j) \leq c(S)/$

# Analysis

**Theorem.** GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for $\textsc{SetCover}$, where $k$ is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \to 0.5 + \ln k$.

**Lemma.** Let $S \in \mathcal{S}$, and let $u_1, \ldots, u_\ell$ be the elements of $S$ in the order in which they are covered ("bought") by GreedySetCover. Then, for every $j \in \{1, \ldots, \ell\}$: $\mathrm{price}(u_j) \le c(S)/(\ell - j + 1)$.

# Analysis

**Theorem.** GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for $\textsc{SetCover}$, where $k$ is the cardinality of the largest set in $\mathcal{S}$ and
$\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \to 0.5 + \ln k$.

**Lemma.** Let $S \in \mathcal{S}$, and let $u_1, \ldots, u_\ell$ be the elements of $S$ in the order in which they are covered ("bought") by GreedySetCover. Then, for every $j \in \{1, \ldots, \ell\}$:
$\mathrm{price}(u_j) \leq c(S)/(\ell - j + 1)$.

**Proof.**

# Analysis

**Theorem.** GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where $k$ is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \to 0.5 + \ln k$.

**Lemma.** Let $S \in \mathcal{S}$, and let $u_1, \ldots, u_\ell$ be the elements of $S$ in the order in which they are covered ("bought") by GreedySetCover. Then, for every $j \in \{1, \ldots, \ell\}$: $\text{price}(u_j) \le c(S)/(\ell - j + 1)$.

**Proof.** Consider the iteration when the algorithm buys $u_j$:

# Analysis

**Theorem.** GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where $k$ is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \to 0.5 + \ln k$.

**Lemma.** Let $S \in \mathcal{S}$, and let $u_1, \ldots, u_\ell$ be the elements of $S$ in the order in which they are covered ("bought") by GreedySetCover. Then, for every $j \in \{1, \ldots, \ell\}$: $\text{price}(u_j) \le c(S)/(\ell - j + 1)$.

**Proof.** Consider the iteration when the algorithm buys $u_j$:

- At most $j - 1$ elements of $S$ already bought.

# Analysis

**Theorem.** GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for $\textsc{SetCover}$, where $k$ is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \to 0.5 + \ln k$.

**Lemma.** Let $S \in \mathcal{S}$, and let $u_1, \ldots, u_\ell$ be the elements of $S$ in the order in which they are covered ("bought") by GreedySetCover. Then, for every $j \in \{1, \ldots, \ell\}$: $\text{price}(u_j) \le c(S)/(\ell - j + 1)$.

**Proof.** Consider the iteration when the algorithm buys $u_j$:

- At most $j - 1$ elements of $S$ already bought.
- At least $\ell - j + 1$ elements of $S$ not yet bought.

# Analysis

**Theorem.** GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for $\textsc{SetCover}$, where $k$ is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \to 0.5 + \ln k$.

**Lemma.** Let $S \in \mathcal{S}$, and let $u_1, \ldots, u_\ell$ be the elements of $S$ in the order in which they are covered ("bought") by GreedySetCover. Then, for every $j \in \{1, \ldots, \ell\}$: $\text{price}(u_j) \leq c(S)/(\ell - j + 1)$.

**Proof.** Consider the iteration when the algorithm buys $u_j$:

- At most $j - 1$ elements of $S$ already bought.
- At least $\ell - j + 1$ elements of $S$ not yet bought.
- Per-element cost for $S$: at most

# Analysis

**Theorem.** GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where $k$ is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \to 0.5 + \ln k$.

**Lemma.** Let $S \in \mathcal{S}$, and let $u_1, \ldots, u_\ell$ be the elements of $S$ in the order in which they are covered ("bought") by GreedySetCover. Then, for every $j \in \{1, \ldots, \ell\}$: $\text{price}(u_j) \leq c(S)/(\ell - j + 1)$.

**Proof.** Consider the iteration when the algorithm buys $u_j$:

- At most $j - 1$ elements of $S$ already bought.
- At least $\ell - j + 1$ elements of $S$ not yet bought.
- Per-element cost for $S$: at most $c(S)/(\ell - j + 1)$

# Analysis

**Theorem.** GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where $k$ is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \to 0.5 + \ln k$.

**Lemma.** Let $S \in \mathcal{S}$, and let $u_1, \ldots, u_\ell$ be the elements of $S$ in the order in which they are covered ("bought") by GreedySetCover. Then, for every $j \in \{1, \ldots, \ell\}$: $\text{price}(u_j) \le c(S)/(\ell - j + 1)$.

**Proof.** Consider the iteration when the algorithm buys $u_j$:

- At most $j - 1$ elements of $S$ already bought.
- At least $\ell - j + 1$ elements of $S$ not yet bought.
- Per-element cost for $S$: at most $c(S)/(\ell - j + 1)$
- Price by alg. no larger due to greedy choice.

# Analysis

**Theorem.** GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for $\textsc{SetCover}$, where $k$ is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \to 0.5 + \ln k$.

**Lemma.** Let $S \in \mathcal{S}$, and let $u_1, \ldots, u_\ell$ be the elements of $S$ in the order in which they are covered ("bought") by GreedySetCover. Then, for every $j \in \{1, \ldots, \ell\}$: $\text{price}(u_j) \le c(S)/(\ell - j + 1)$.

**Lemma.** $\text{price}(S) := \sum_{i=1}^{\ell} \text{price}(u_i) \le$

# Analysis

**Theorem.** GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for $\textsc{SetCover}$, where $k$ is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \to 0.5 + \ln k$.

**Lemma.** Let $S \in \mathcal{S}$, and let $u_1, \ldots, u_\ell$ be the elements of $S$ in the order in which they are covered ("bought") by GreedySetCover. Then, for every $j \in \{1, \ldots, \ell\}$: $\mathrm{price}(u_j) \leq c(S)/(\ell - j + 1)$.

**Lemma.** $\mathrm{price}(S) := \sum_{i=1}^{\ell} \mathrm{price}(u_i) \leq c(S) \cdot \mathcal{H}_\ell$.

# Analysis

**Theorem.** GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where $k$ is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \to 0.5 + \ln k$.

**Lemma.** Let $S \in \mathcal{S}$, and let $u_1, \ldots, u_\ell$ be the elements of $S$ in the order in which they are covered ("bought") by GreedySetCover. Then, for every $j \in \{1, \ldots, \ell\}$: $\text{price}(u_j) \leq c(S)/(\ell - j + 1)$.

**Lemma.** $\text{price}(S) := \sum_{i=1}^{\ell} \text{price}(u_i) \leq c(S) \cdot \mathcal{H}_\ell$.

**Proof.**

# Analysis

**Theorem.** GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for $\textsc{SetCover}$, where $k$ is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \to 0.5 + \ln k$.

**Lemma.** Let $S \in \mathcal{S}$, and let $u_1, \ldots, u_\ell$ be the elements of $S$ in the order in which they are covered ("bought") by GreedySetCover. Then, for every $j \in \{1, \ldots, \ell\}$: $\mathrm{price}(u_j) \le c(S)/(\ell - j + 1)$.

**Lemma.** $\mathrm{price}(S) := \sum_{i=1}^{\ell} \mathrm{price}(u_i) \le c(S) \cdot \mathcal{H}_\ell$.

**Proof.** Let $\{S_1, \ldots, S_m\}$ be an opt. sol.

# Analysis

**Theorem.** GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where $k$ is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \to 0.5 + \ln k$.

**Lemma.** Let $S \in \mathcal{S}$, and let $u_1, \ldots, u_\ell$ be the elements of $S$ in the order in which they are covered ("bought") by GreedySetCover. Then, for every $j \in \{1, \ldots, \ell\}$: $\text{price}(u_j) \le c(S)/(\ell - j + 1)$.

**Lemma.** $\text{price}(S) := \sum_{i=1}^{\ell} \text{price}(u_i) \le c(S) \cdot \mathcal{H}_\ell$.

**Proof.** Let $\{S_1, \ldots, S_m\}$ be an opt. sol. $\text{OPT} = \sum_{i=1}^{m} c(S_i)$.

# Analysis

**Theorem.** GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where $k$ is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \to 0.5 + \ln k$.

**Lemma.** Let $S \in \mathcal{S}$, and let $u_1, \ldots, u_\ell$ be the elements of $S$ in the order in which they are covered ("bought") by GreedySetCover. Then, for every $j \in \{1, \ldots, \ell\}$: $\mathrm{price}(u_j) \le c(S)/(\ell - j + 1)$.

**Lemma.** $\mathrm{price}(S) := \sum_{i=1}^{\ell} \mathrm{price}(u_i) \le c(S) \cdot \mathcal{H}_\ell$.

**Proof.** Let $\{S_1, \ldots, S_m\}$ be an opt. sol. $\mathrm{OPT} = \sum_{i=1}^{m} c(S_i)$.

$\mathrm{ALG} \le$

# Analysis

**Theorem.** GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where $k$ is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \to 0.5 + \ln k$.

**Lemma.** Let $S \in \mathcal{S}$, and let $u_1, \ldots, u_\ell$ be the elements of $S$ in the order in which they are covered ("bought") by GreedySetCover. Then, for every $j \in \{1, \ldots, \ell\}$: $\text{price}(u_j) \le c(S)/(\ell - j + 1)$.

**Lemma.** $\text{price}(S) := \sum_{i=1}^{\ell} \text{price}(u_i) \le c(S) \cdot \mathcal{H}_\ell$.

**Proof.** Let $\{S_1, \ldots, S_m\}$ be an opt. sol. $\text{OPT} = \sum_{i=1}^{m} c(S_i)$.

$\text{ALG} \le \text{price}(U) =$

# Analysis

**Theorem.** GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for $\textsc{SetCover}$, where $k$ is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \to 0.5 + \ln k$.

**Lemma.** Let $S \in \mathcal{S}$, and let $u_1, \ldots, u_\ell$ be the elements of $S$ in the order in which they are covered ("bought") by GreedySetCover. Then, for every $j \in \{1, \ldots, \ell\}$: $\mathrm{price}(u_j) \le c(S)/(\ell - j + 1)$.

**Lemma.** $\mathrm{price}(S) := \sum_{i=1}^{\ell} \mathrm{price}(u_i) \le c(S) \cdot \mathcal{H}_\ell$.

**Proof.** Let $\{S_1, \ldots, S_m\}$ be an opt. sol. $\mathrm{OPT} = \sum_{i=1}^{m} c(S_i)$.

$\mathrm{ALG} \le \mathrm{price}(U) = \sum_{u \in U} \mathrm{price}(u) \le$

# Analysis

**Theorem.** GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for $\textsc{SetCover}$, where $k$ is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \to 0.5 + \ln k$.

**Lemma.** Let $S \in \mathcal{S}$, and let $u_1, \ldots, u_\ell$ be the elements of $S$ in the order in which they are covered ("bought") by GreedySetCover. Then, for every $j \in \{1, \ldots, \ell\}$: $\text{price}(u_j) \le c(S)/(\ell - j + 1)$.

**Lemma.** $\text{price}(S) := \sum_{i=1}^{\ell} \text{price}(u_i) \le c(S) \cdot \mathcal{H}_\ell$.

**Proof.** Let $\{S_1, \ldots, S_m\}$ be an opt. sol. $\text{OPT} = \sum_{i=1}^{m} c(S_i)$.
$\text{ALG} \le \text{price}(U) = \sum_{u \in U} \text{price}(u) \le \sum_{i=1}^{m} \text{price}(S_i)$

# Analysis

**Theorem.** GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where $k$ is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \to 0.5 + \ln k$.

**Lemma.** Let $S \in \mathcal{S}$, and let $u_1, \ldots, u_\ell$ be the elements of $S$ in the order in which they are covered ("bought") by GreedySetCover. Then, for every $j \in \{1, \ldots, \ell\}$: $\text{price}(u_j) \le c(S)/(\ell - j + 1)$.

**Lemma.** $\text{price}(S) := \sum_{i=1}^{\ell} \text{price}(u_i) \le c(S) \cdot \mathcal{H}_\ell$.

**Proof.** Let $\{S_1, \ldots, S_m\}$ be an opt. sol. $\text{OPT} = \sum_{i=1}^{m} c(S_i)$.

$$\text{ALG} \le \text{price}(U) = \sum_{u \in U} \text{price}(u) \le \sum_{i=1}^{m} \text{price}(S_i)$$
$$\le \sum_{i=1}^{m} c(S_i) \cdot \mathcal{H}_k =$$

# Analysis

**Theorem.** GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for $\textsc{SetCover}$, where $k$ is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \to 0.5 + \ln k$.

**Lemma.** Let $S \in \mathcal{S}$, and let $u_1, \ldots, u_\ell$ be the elements of $S$ in the order in which they are covered ("bought") by GreedySetCover. Then, for every $j \in \{1, \ldots, \ell\}$: $\mathrm{price}(u_j) \le c(S)/(\ell - j + 1)$.

**Lemma.** $\mathrm{price}(S) := \sum_{i=1}^{\ell} \mathrm{price}(u_i) \le c(S) \cdot \mathcal{H}_\ell$.

**Proof.** Let $\{S_1, \ldots, S_m\}$ be an opt. sol. $\mathrm{OPT} = \sum_{i=1}^{m} c(S_i)$.

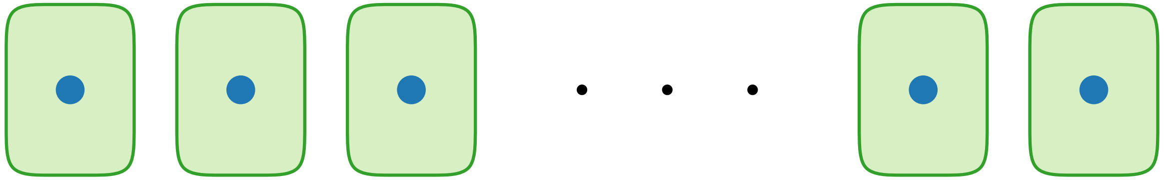
$$\mathrm{ALG} \le \mathrm{price}(U) = \sum_{u \in U} \mathrm{price}(u) \le \sum_{i=1}^{m} \mathrm{price}(S_i)$$
$$\le \sum_{i=1}^{m} c(S_i) \cdot \mathcal{H}_k = \mathrm{OPT} \cdot \mathcal{H}_k \qquad \square$$

# Analysis tight?

**Theorem.** GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for $\textsc{SetCover}$, where $k$ is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \leq 1 + \ln k$.

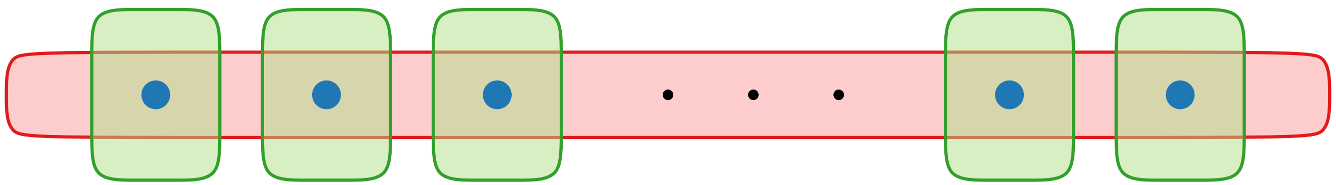
# Analysis tight?

**Theorem.** GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for $\textsc{SetCover}$, where $k$ is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \leq 1 + \ln k$.



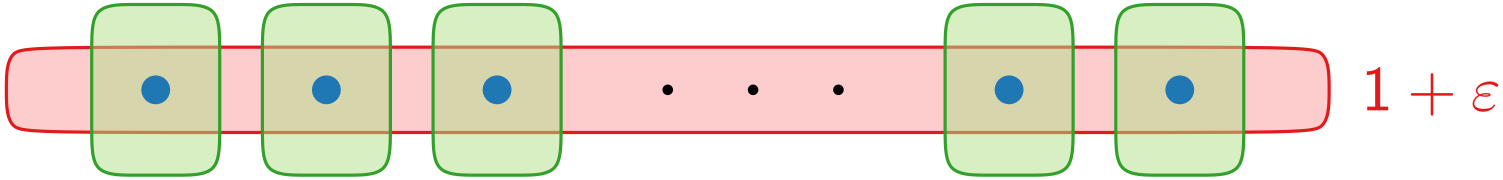
# Analysis tight?

**Theorem.** GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where $k$ is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \leq 1 + \ln k$.



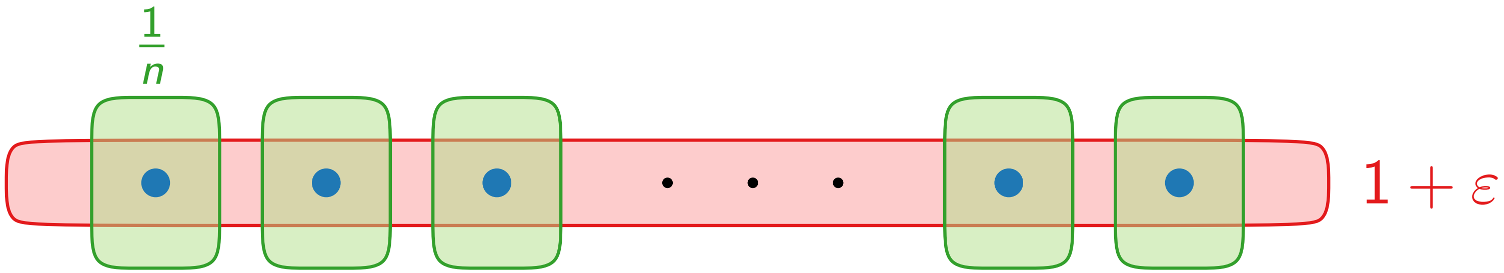
# Analysis tight?

**Theorem.** GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where $k$ is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \leq 1 + \ln k$.



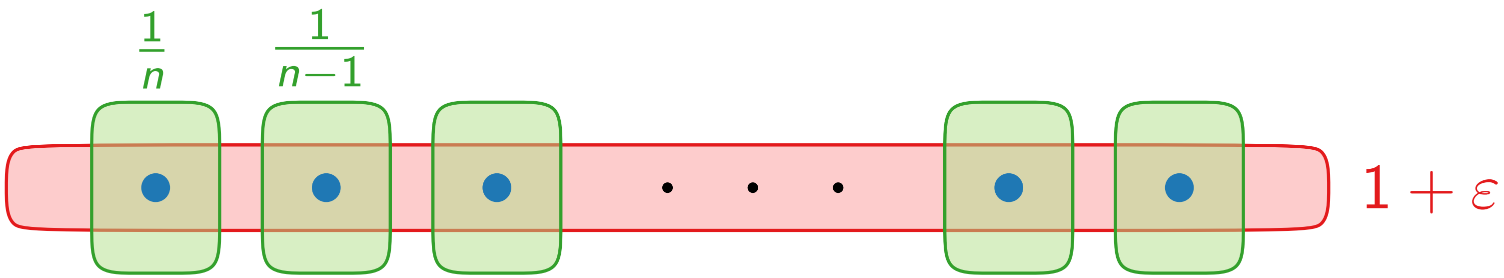
# Analysis tight?

**Theorem.** GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for $\textsc{SetCover}$, where $k$ is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \le 1 + \ln k$.



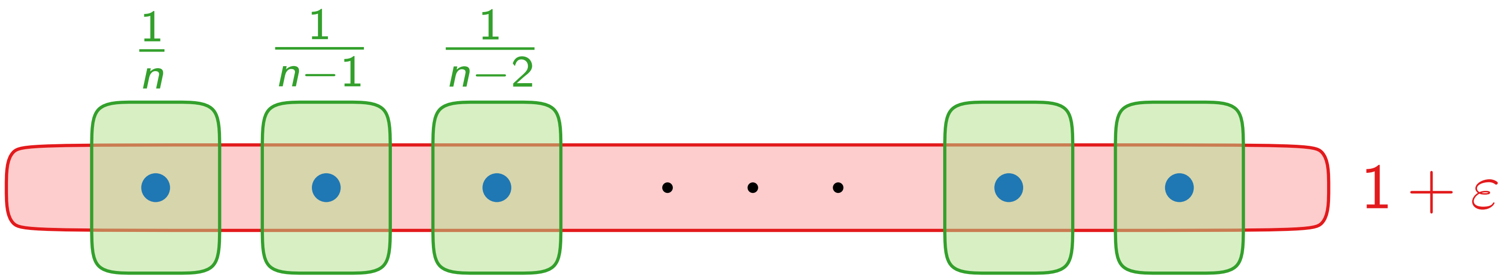
# Analysis tight?

**Theorem.** GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where $k$ is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \leq 1 + \ln k$.



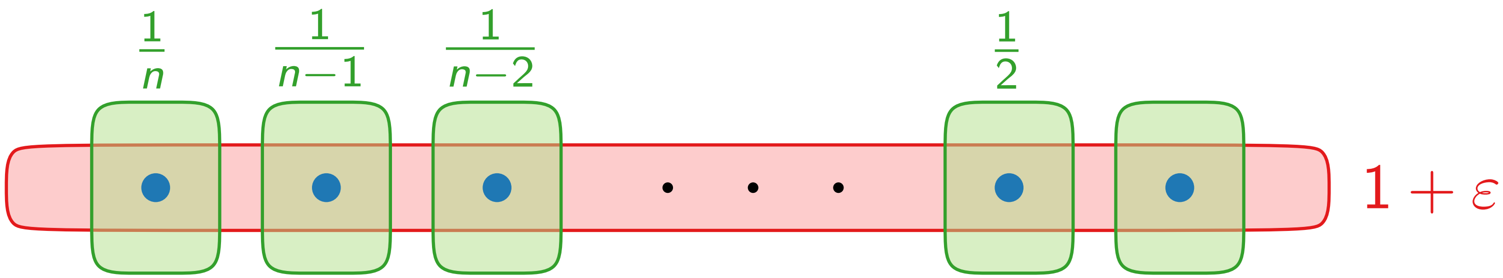
# Analysis tight?

**Theorem.** GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where $k$ is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \leq 1 + \ln k$.



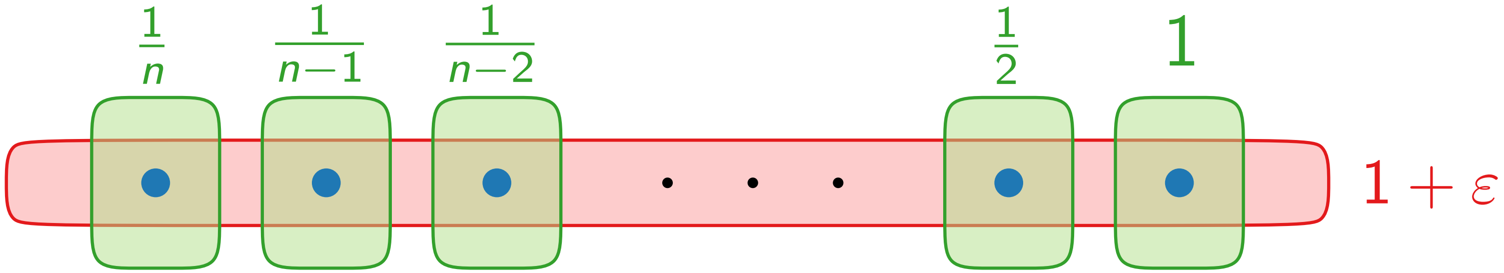
# Analysis tight?

**Theorem.** GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where $k$ is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \leq 1 + \ln k$.



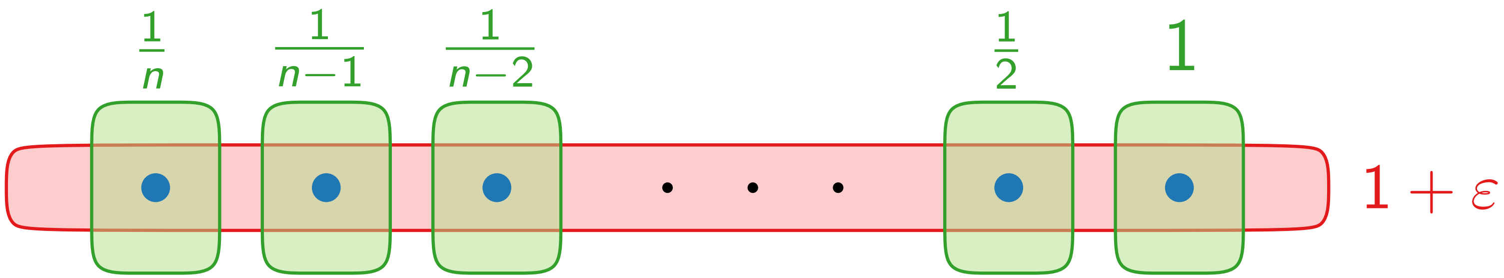
# Analysis tight?

**Theorem.** GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for $\textsc{SetCover}$, where $k$ is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \leq 1 + \ln k$.



# Analysis tight?

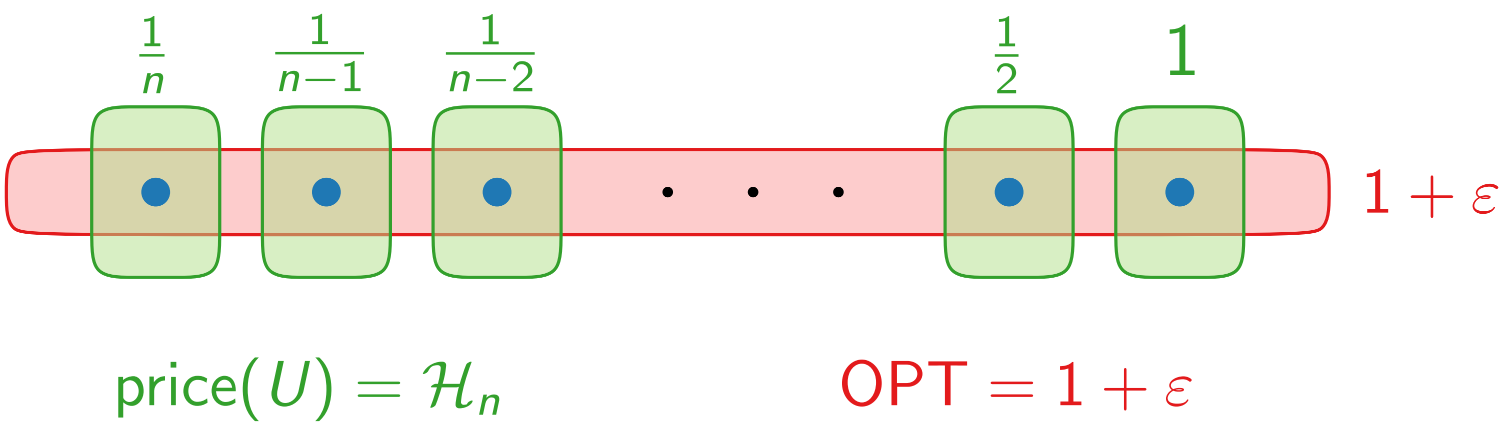
**Theorem.** GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for $\textsc{SetCover}$, where $k$ is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \leq 1 + \ln k$.




$\text{price}(U) = \mathcal{H}_n$ $\qquad$ $\text{OPT} = 1 + \varepsilon$

# Analysis tight?

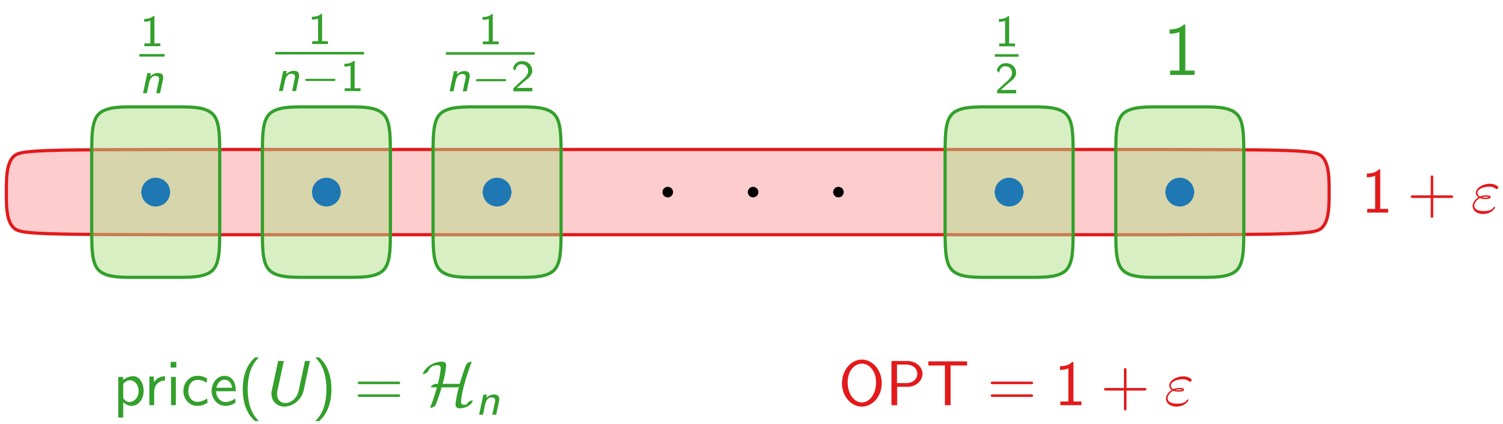
**Theorem.** GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where $k$ is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \leq 1 + \ln k$.




**Can we do better?**

# Analysis tight?

**Theorem.** GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where $k$ is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \leq 1 + \ln k$.

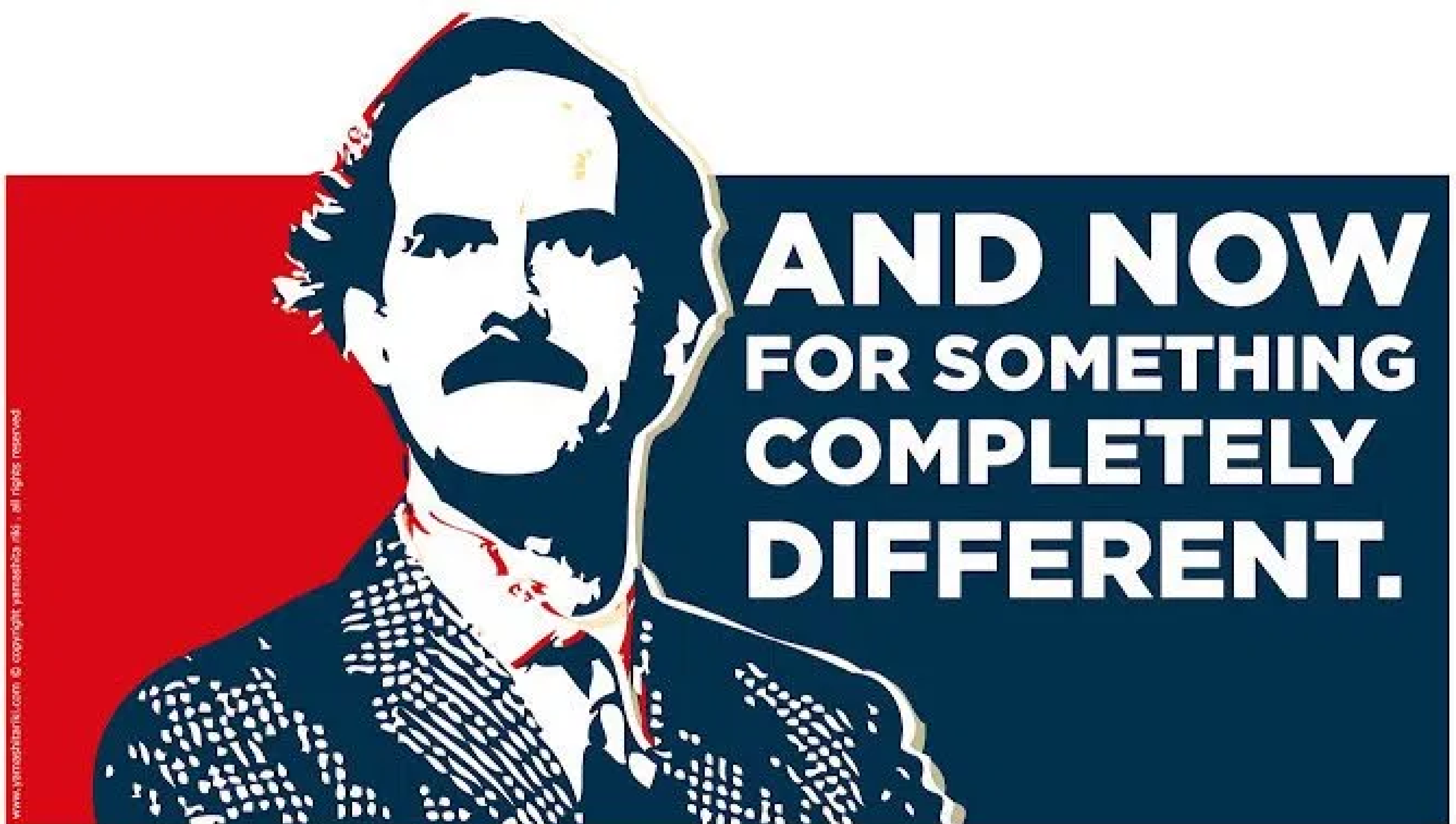



**Can we do better?**

No – for any $\varepsilon > 0$, it is NP-hard to approximate SetCover with factor $(1 - \varepsilon) \cdot \ln n$ [Feige, JACM 1998] [Dinur, Steurer, STOC 2014]

# Analysis tight?

**Theorem.** GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for $\textsc{SetCover}$, where $k$ is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \leq 1 + \ln k$.

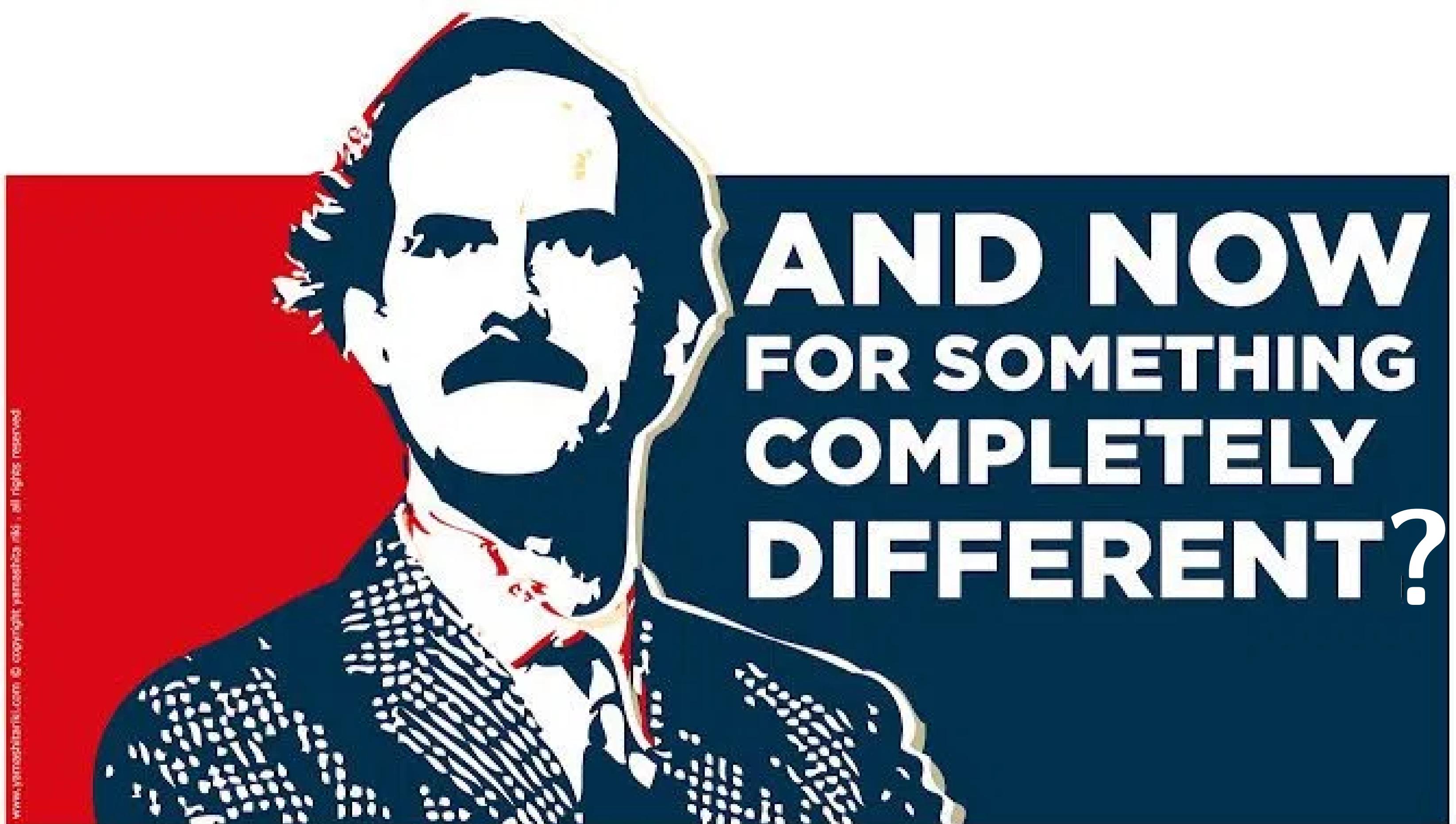



**Can we do**

No – for any
with factor (

# Analysis tight?

**Theorem.** GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SETCOVER, where $k$ is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \leq 1 + \ln k$.




**Can we do**

No – for any
with factor (

# Approximation Algorithms

## Lecture 2:
## SetCover and ShortestSuperString

### Part IV:
### ShortestSuperString

# ShortestSuperString (SSS)

Given a set $\{s_1, \ldots, s_n\} \subseteq \Sigma^+$ of strings over a finite alphabet $\Sigma$.

# ShortestSuperString (SSS)

Given a set $\{s_1, \ldots, s_n\} \subseteq \Sigma^+$ of strings over a finite alphabet $\Sigma$.

Find a **shortest string** $s$ (*superstring*) such that,
for each $i \in \{1, \ldots, n\}$, the string $s_i$ is a *substring* of $s$.

# ShortestSuperString (SSS)

Given a set $\{s_1, \ldots, s_n\} \subseteq \Sigma^+$ of strings over a finite alphabet $\Sigma$.

Find a **shortest string** $s$ (*superstring*) such that,
for each $i \in \{1, \ldots, n\}$, the string $s_i$ is a *substring* of $s$.

**Example.** $U := \{cbaa, abc, bcb\}$

# ShortestSuperString (SSS)

Given a set $\{s_1, \ldots, s_n\} \subseteq \Sigma^+$ of strings over a finite alphabet $\Sigma$.

Find a **shortest string** $s$ (*superstring*) such that,
for each $i \in \{1, \ldots, n\}$, the string $s_i$ is a *substring* of $s$.

**Example.** $U := \{cbaa, abc, bcb\} \quad \rightarrow cbaabcb$ ?

# ShortestSuperString (SSS)

Given a set $\{s_1, \ldots, s_n\} \subseteq \Sigma^+$ of strings over a finite alphabet $\Sigma$.

Find a **shortest string** $s$ (*superstring*) such that,
for each $i \in \{1, \ldots, n\}$, the string $s_i$ is a *substring* of $s$.

**Example.** $U := \{cbaa, abc, bcb\} \quad \rightarrow cbaabcb$ ?

$abc$

# ShortestSuperString (SSS)

Given a set $\{s_1, \ldots, s_n\} \subseteq \Sigma^+$ of strings over a finite alphabet $\Sigma$.

Find a **shortest string** $s$ (*superstring*) such that,
for each $i \in \{1, \ldots, n\}$, the string $s_i$ is a *substring* of $s$.

**Example.** $U := \{cbaa, abc, bcb\}$ $\rightarrow$ $cbaabcb$ ?

$abc$
$\;\;bcb$

# ShortestSuperString (SSS)

Given a set $\{s_1, \ldots, s_n\} \subseteq \Sigma^+$ of strings over a finite alphabet $\Sigma$.

Find a **shortest string** $s$ (*superstring*) such that, for each $i \in \{1, \ldots, n\}$, the string $s_i$ is a *substring* of $s$.

**Example.** $U := \{cbaa, abc, bcb\} \quad \to cbaabcb$ ?

$$\begin{array}{l} abc \\ \;\;bcb \\ \;\;\;\;cbaa \end{array}$$

# ShortestSuperString (SSS)

Given a set $\{s_1, \ldots, s_n\} \subseteq \Sigma^+$ of strings over a finite alphabet $\Sigma$.

Find a **shortest string** $s$ (*superstring*) such that,
for each $i \in \{1, \ldots, n\}$, the string $s_i$ is a *substring* of $s$.

**Example.** $U := \{cbaa, abc, bcb\} \quad \rightarrow cbaabcb\ ?$

$abcbaa$

$abc$
$\ \ bcb$
$\ \ \ \ cbaa$

# ShortestSuperString (SSS)

Given a set $\{s_1, \ldots, s_n\} \subseteq \Sigma^+$ of strings over a finite alphabet $\Sigma$.

Find a **shortest string** $s$ (*superstring*) such that,
for each $i \in \{1, \ldots, n\}$, the string $s_i$ is a *substring* of $s$.

**Example.** $U := \{cbaa, abc, bcb\}$ $\to cbaabcb$ ?

$abcbaa$ "covers" all strings in $U$

$abc$
$\ \ bcb$
$\ \ \ \ cbaa$

# ShortestSuperString (SSS)

Given a set $\{s_1, \ldots, s_n\} \subseteq \Sigma^+$ of strings over a finite alphabet $\Sigma$.

Find a **shortest string** $s$ (*superstring*) such that, for each $i \in \{1, \ldots, n\}$, the string $s_i$ is a *substring* of $s$.

**Example.** $U := \{cbaa, abc, bcb\}$ $\to$ $cbaabcb$ ?

W.l.o.g.: No string $s_i$ is a substring of any other string $s_j$.

$abcbaa$ "covers" all strings in $U$

$abc$

$\;\;bcb$

$\;\;\;\;cbaa$

# SSS as a SetCover Problem

SetCover Instance: ground set $U$, set family $\mathcal{S}$, costs $c$.

# SSS as a SetCover Problem

SetCover Instance: ground set $U$, set family $\mathcal{S}$, costs $c$.

Ground set $U := \{s_1, \ldots, s_n\}$.

# SSS as a SetCover Problem

SetCover Instance: ground set $U$, set family $\mathcal{S}$, costs $c$.

Ground set $U := \{s_1, \ldots, s_n\}$.

Let be $\sigma_{ijk}$ be the unique string with prefix $s_i$ and suffix $s_j$ where $s_i$ and $s_j$ *overlap* on $k$ characters (for suitable $i, j, k$)

# SSS as a SetCover Problem

SetCover Instance: ground set $U$, set family $\mathcal{S}$, costs $c$.

Ground set $U := \{s_1, \ldots, s_n\}$.

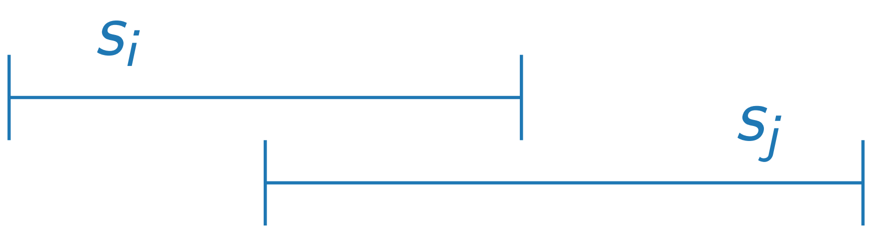
Let be $\sigma_{ijk}$ be the unique string with prefix $s_i$ and suffix $s_j$ where $s_i$ and $s_j$ *overlap* on $k$ characters (for suitable $i, j, k$)

# SSS as a SetCover Problem

SetCover Instance: ground set $U$, set family $\mathcal{S}$, costs $c$.

Ground set $U := \{s_1, \ldots, s_n\}$.

Let be $\sigma_{ijk}$ be the unique string with prefix $s_i$ and suffix $s_j$ where $s_i$ and $s_j$ *overlap* on $k$ characters (for suitable $i, j, k$)

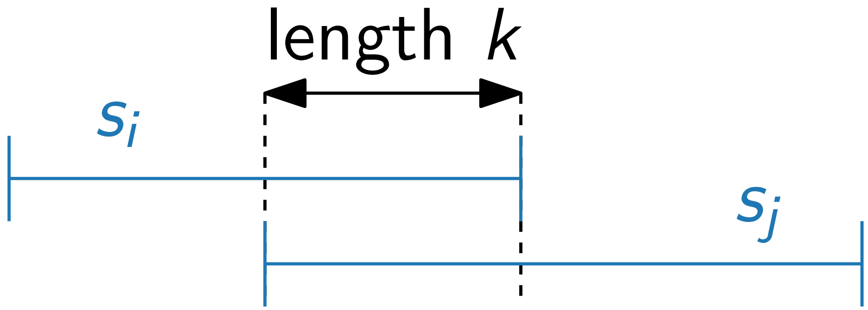


# SSS as a SetCover Problem

SetCover Instance: ground set $U$, set family $\mathcal{S}$, costs $c$.

Ground set $U := \{s_1, \ldots, s_n\}$.

Let be $\sigma_{ijk}$ be the unique string with prefix $s_i$ and suffix $s_j$ where $s_i$ and $s_j$ *overlap* on $k$ characters (for suitable $i, j, k$)

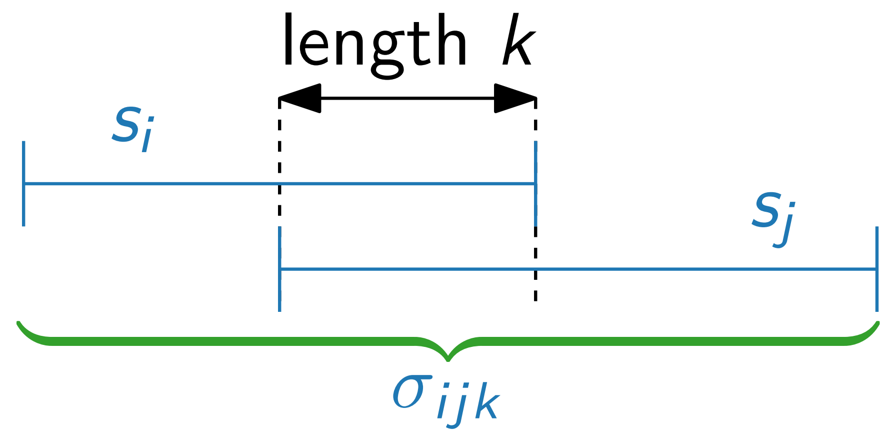


# SSS as a SetCover Problem

SetCover Instance: ground set $U$, set family $\mathcal{S}$, costs $c$.

Ground set $U := \{s_1, \ldots, s_n\}$.

Let be $\sigma_{ijk}$ be the unique string with prefix $s_i$ and suffix $s_j$ where $s_i$ and $s_j$ *overlap* on $k$ characters (for suitable $i, j, k$)



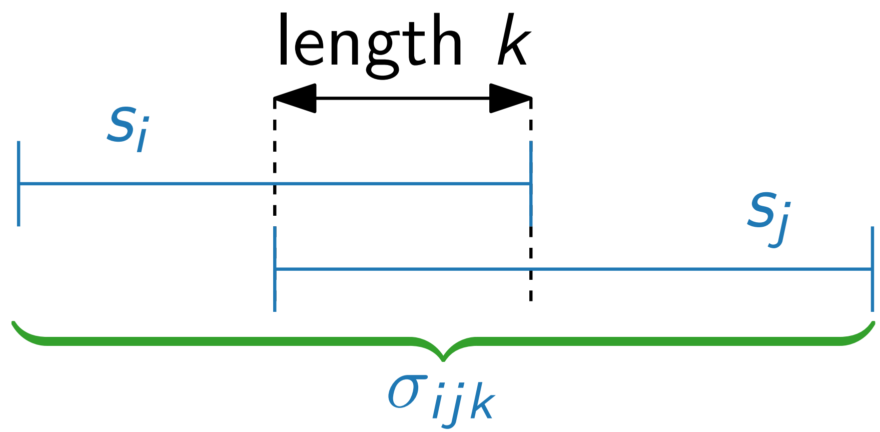
# SSS as a SetCover Problem

SetCover Instance: ground set $U$, set family $\mathcal{S}$, costs $c$.

Ground set $U := \{s_1, \ldots, s_n\}$.

Let be $\sigma_{ijk}$ be the unique string with prefix $s_i$ and suffix $s_j$ where $s_i$ and $s_j$ *overlap* on $k$ characters (for suitable $i, j, k$)

$s_i$: cabab   $s_j$: ababc



# SSS as a SetCover Problem

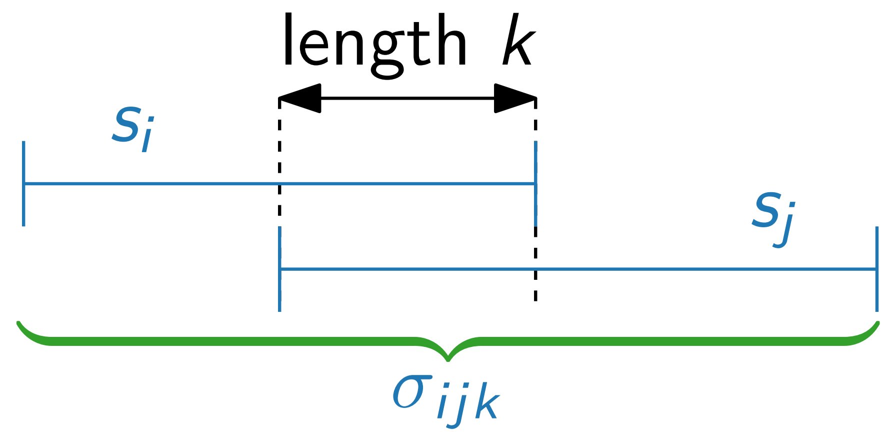
SetCover Instance: ground set $U$, set family $\mathcal{S}$, costs $c$.

Ground set $U := \{s_1, \ldots, s_n\}$.

Let be $\sigma_{ijk}$ be the unique string with prefix $s_i$ and suffix $s_j$ where $s_i$ and $s_j$ *overlap* on $k$ characters (for suitable $i, j, k$)

$s_i$: cabab $s_j$: ababc

```
cabab
  ababc
```



# SSS as a SetCover Problem

SetCover Instance: ground set $U$, set family $\mathcal{S}$, costs $c$.

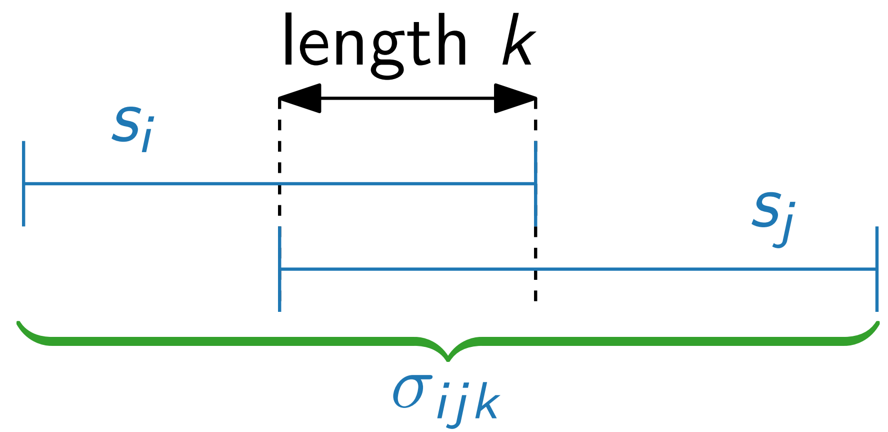
Ground set $U := \{s_1, \ldots, s_n\}$.

Let be $\sigma_{ijk}$ be the unique string with prefix $s_i$ and suffix $s_j$ where $s_i$ and $s_j$ *overlap* on $k$ characters (for suitable $i, j, k$)

$s_i$: cabab   $s_j$: ababc

cabab

  ababc



# SSS as a SetCover Problem

SetCover Instance: ground set $U$, set family $\mathcal{S}$, costs $c$.

Ground set $U := \{s_1, \ldots, s_n\}$.

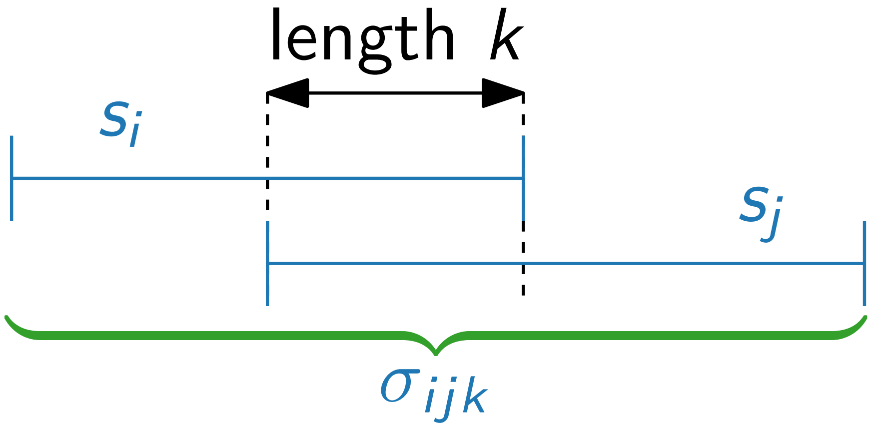
Let be $\sigma_{ijk}$ be the unique string with prefix $s_i$ and suffix $s_j$ where $s_i$ and $s_j$ *overlap* on $k$ characters (for suitable $i, j, k$)

$s_i$: cabab $s_j$: ababc

cabab

ababc

$\sigma_{ij2}$: cababac



# SSS as a SetCover Problem

SetCover Instance: ground set $U$, set family $\mathcal{S}$, costs $c$.

Ground set $U := \{s_1, \ldots, s_n\}$.

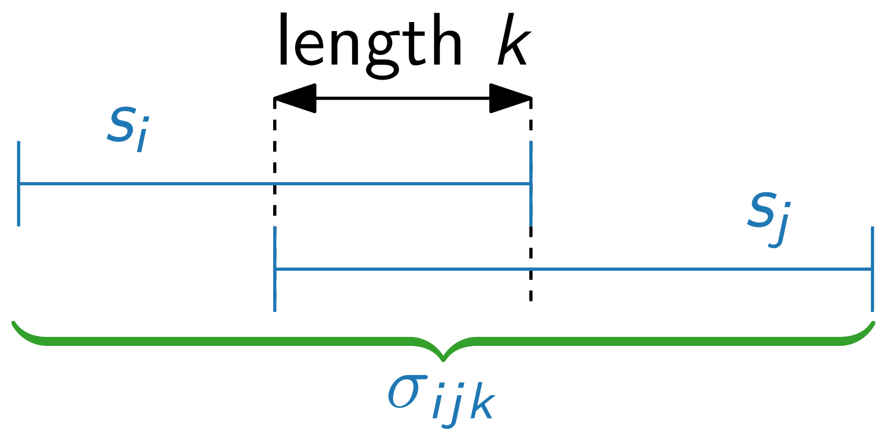
Let be $\sigma_{ijk}$ be the unique string with prefix $s_i$ and suffix $s_j$ where $s_i$ and $s_j$ *overlap* on $k$ characters (for suitable $i, j, k$)

$s_i$: cabab   $s_j$: ababc

cabab   cabab

  ababc   ababc

$\sigma_{ij2}$: cabababc



# SSS as a SetCover Problem

SetCover Instance: ground set $U$, set family $\mathcal{S}$, costs $c$.

Ground set $U := \{s_1, \ldots, s_n\}$.

Let be $\sigma_{ijk}$ be the unique string with prefix $s_i$ and suffix $s_j$ where $s_i$ and $s_j$ *overlap* on $k$ characters (for suitable $i, j, k$)

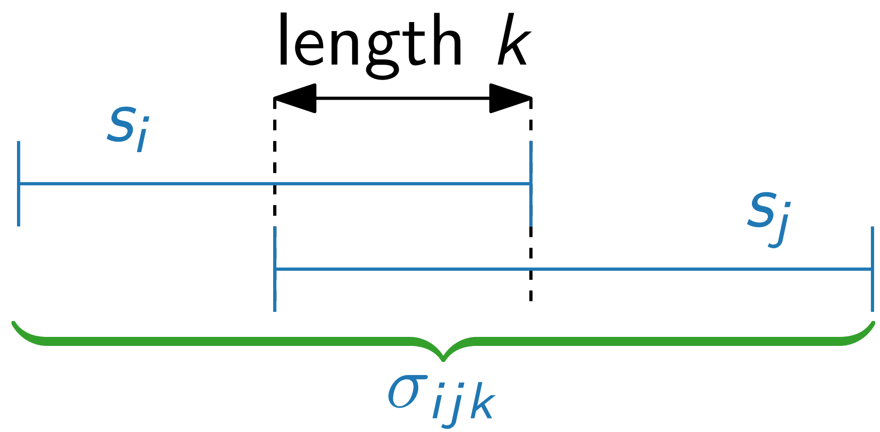
$s_i$: cabab $s_j$: ababc

cabab
ababc

$\sigma_{ij2}$: cabababc

cabab
ababc

$\sigma_{ij4}$: cababc



# SSS as a SetCover Problem

SetCover Instance: ground set $U$, set family $\mathcal{S}$, costs $c$.

Ground set $U := \{s_1, \dots, s_n\}$.

Let be $\sigma_{ijk}$ be the unique string with prefix $s_i$ and suffix $s_j$ where $s_i$ and $s_j$ *overlap* on $k$ characters (for suitable $i, j, k$)

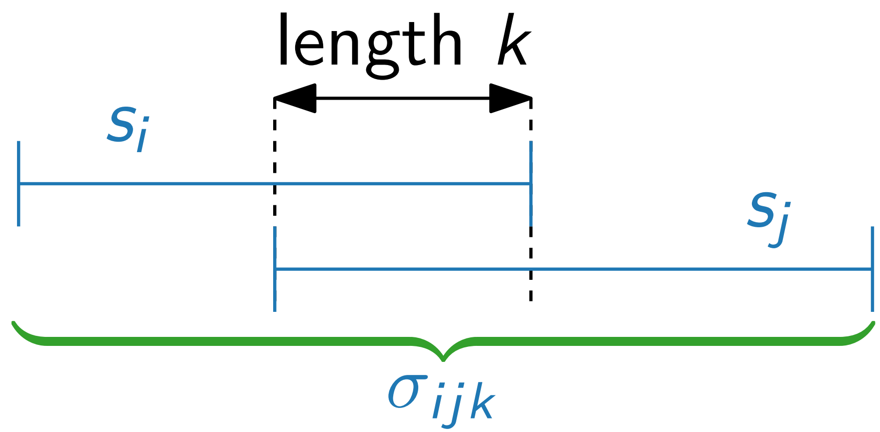
$s_i$: cabab $s_j$: ababc

cabab
ababc

$\sigma_{ij2}$: cababab

cabab
ababc

$\sigma_{ij4}$: cababc




$S(\sigma_{ijk}) =$

$c\,(S(\sigma_{ijk})) =$

$\mathcal{S} =$

# SSS as a SetCover Problem

SetCover Instance: ground set $U$, set family $\mathcal{S}$, costs $c$.

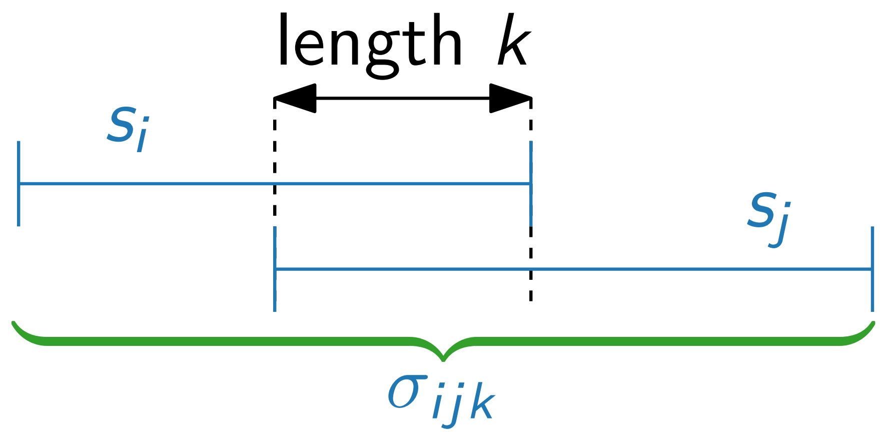
Ground set $U := \{s_1, \ldots, s_n\}$.

Let be $\sigma_{ijk}$ be the unique string with prefix $s_i$ and suffix $s_j$ where $s_i$ and $s_j$ *overlap* on $k$ characters (for suitable $i, j, k$)

$s_i$: cabab $s_j$: ababc

cabab / ababc (overlap of 2) cabab / ababc (overlap of 4)

$\sigma_{ij2}$: cabababc $\sigma_{ij4}$: cababc




$S(\sigma_{ijk}) = \{s \in U \mid s$ substring of $\sigma_{ijk}\}$ – contains the elements of the ground set covered by $\sigma_{ijk}$.

$c\,(S(\sigma_{ijk})) =$

$\mathcal{S} =$

# SSS as a SetCover Problem

SetCover Instance: ground set $U$, set family $\mathcal{S}$, costs $c$.

Ground set $U := \{s_1, \ldots, s_n\}$.

Let be $\sigma_{ijk}$ be the unique string with prefix $s_i$ and suffix $s_j$ where $s_i$ and $s_j$ *overlap* on $k$ characters (for suitable $i, j, k$)

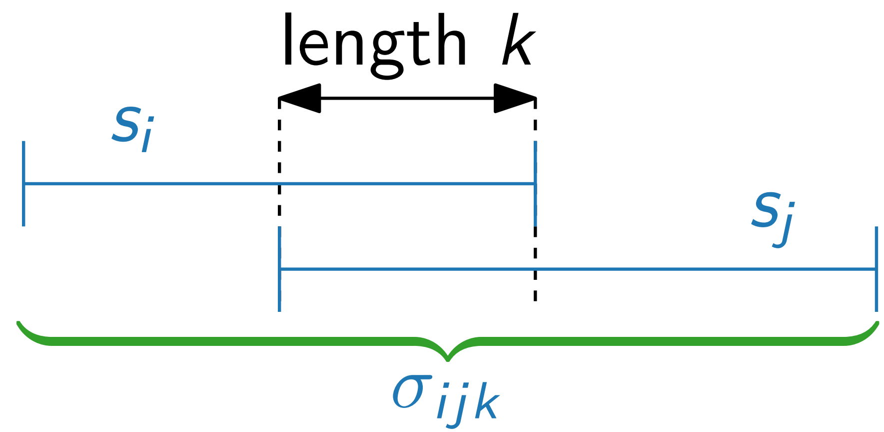
$s_i$: cabab $s_j$: ababc

cabab
ababc

$\sigma_{ij2}$: cababab c

cabab
ababc

$\sigma_{ij4}$: cababc




$S(\sigma_{ijk}) = \{s \in U \mid s$ substring of $\sigma_{ijk}\}$ – contains the elements of the ground set covered by $\sigma_{ijk}$.

$c\,(S(\sigma_{ijk})) = |\sigma_{ijk}|$ (number of characters in $\sigma_{ijk}$)

$\mathcal{S} =$

# SSS as a SetCover Problem

SetCover Instance: ground set $U$, set family $\mathcal{S}$, costs $c$.

Ground set $U := \{s_1, \dots, s_n\}$.

Let be $\sigma_{ijk}$ be the unique string with prefix $s_i$ and suffix $s_j$ where $s_i$ and $s_j$ *overlap* on $k$ characters (for suitable $i, j, k$)

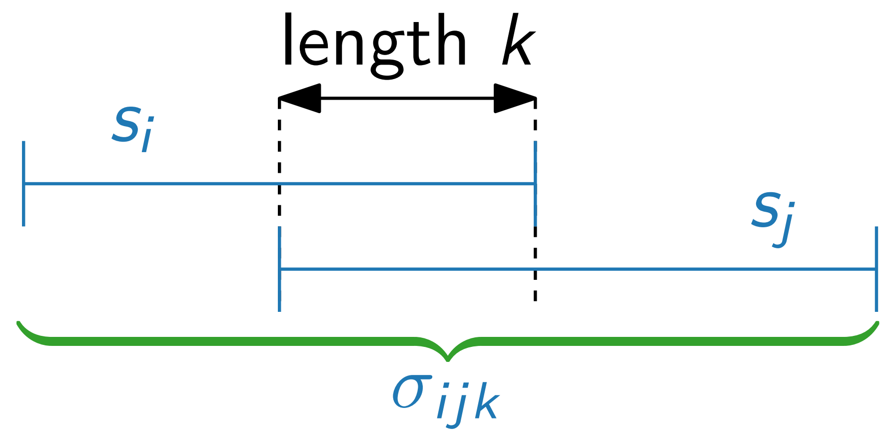
$s_i$: cabab $s_j$: ababc

cabab
ababc

$\sigma_{ij2}$: cababab c

cabab
ababc

$\sigma_{ij4}$: cababc




$S(\sigma_{ijk}) = \{s \in U \mid s \text{ substring of } \sigma_{ijk}\}$ – contains the elements of the ground set covered by $\sigma_{ijk}$.

$c\,(S(\sigma_{ijk})) = |\sigma_{ijk}|$ (number of characters in $\sigma_{ijk}$)

$\mathcal{S} = \{S(\sigma_{ijk}) \mid 1 \leq i, j \leq n, \text{suitable } k \geq 0\}$

# Approximation Algorithms

## Lecture 2: SetCover and ShortestSuperString

### Part V: Solving ShortestSuperString via SetCover

# Relating SSS and SetCover

**Lemma.** Let $\mathsf{OPT}_{\mathsf{SSS}}$ be the length of a shortest superstring of $U$, and let $\mathsf{OPT}_{\mathsf{SC}}$ be the minimum cost of the corresponding SetCover instance. Then

$$\mathsf{OPT}_{\mathsf{SSS}} \leq \mathsf{OPT}_{\mathsf{SC}}.$$

# Relating SSS and SetCover

**Lemma.** Let $\mathrm{OPT}_{\mathrm{SSS}}$ be the length of a shortest superstring of $U$, and let $\mathrm{OPT}_{\mathrm{SC}}$ be the minimum cost of the corresponding SetCover instance. Then

$$\mathrm{OPT}_{\mathrm{SSS}} \leq \mathrm{OPT}_{\mathrm{SC}}.$$

**Proof.**

Consider an optimal set cover $\{S(\pi_1), \ldots, S(\pi_k)\}$ of $U$.

# Relating SSS and SetCover

**Lemma.** Let $\mathrm{OPT}_{\mathrm{SSS}}$ be the length of a shortest superstring of $U$, and let $\mathrm{OPT}_{\mathrm{SC}}$ be the minimum cost of the corresponding SetCover instance. Then

$$\mathrm{OPT}_{\mathrm{SSS}} \leq \mathrm{OPT}_{\mathrm{SC}}.$$

**Proof.**

Consider an optimal set cover $\{S(\pi_1), \ldots, S(\pi_k)\}$ of $U$.

Then $s := \pi_1 \circ \cdots \circ \pi_k$ is a superstring of $U$ of length

# Relating SSS and SetCover

**Lemma.** Let $\mathrm{OPT}_{\mathrm{SSS}}$ be the length of a shortest superstring of $U$, and let $\mathrm{OPT}_{\mathrm{SC}}$ be the minimum cost of the corresponding SetCover instance. Then

$$\mathrm{OPT}_{\mathrm{SSS}} \leq \mathrm{OPT}_{\mathrm{SC}}.$$

**Proof.**

Consider an optimal set cover $\{S(\pi_1), \ldots, S(\pi_k)\}$ of $U$.

Then $s := \pi_1 \circ \cdots \circ \pi_k$ is a superstring of $U$ of length $\sum_{i=1}^{k} |\pi_i| = \sum_{i=1}^{k} c(S(\pi_i)) = \mathrm{OPT}_{\mathrm{SC}}$.

# Relating SSS and SetCover

**Lemma.** Let $\mathrm{OPT}_{\mathrm{SSS}}$ be the length of a shortest superstring of $U$, and let $\mathrm{OPT}_{\mathrm{SC}}$ be the minimum cost of the corresponding SetCover instance. Then

$$\mathrm{OPT}_{\mathrm{SSS}} \leq \mathrm{OPT}_{\mathrm{SC}}.$$

**Proof.**

Consider an optimal set cover $\{S(\pi_1), \ldots, S(\pi_k)\}$ of $U$.

Then $s := \pi_1 \circ \cdots \circ \pi_k$ is a superstring of $U$ of length $\sum_{i=1}^{k} |\pi_i| = \sum_{i=1}^{k} c(S(\pi_i)) = \mathrm{OPT}_{\mathrm{SC}}$.

Thus, $\mathrm{OPT}_{\mathrm{SSS}} \leq |s| = \mathrm{OPT}_{\mathrm{SC}}$.

# Relating SSS and SetCover

**Lemma.** $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

# Relating SSS and SetCover

**Lemma.** $\mathsf{OPT}_{\mathsf{SC}} \leq 2 \cdot \mathsf{OPT}_{\mathsf{SSS}}$.

**Proof.** Consider an optimal superstring $s$.

# Relating SSS and SetCover

**Lemma.** $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

**Proof.** Consider an optimal superstring $s$.

# Relating SSS and SetCover

**Lemma.** $\text{OPT}_{\text{SC}} \leq 2 \cdot \text{OPT}_{\text{SSS}}$.

**Proof.** Consider an optimal superstring $s$.
Construct a set cover of cost $\leq 2|s| = 2 \cdot \text{OPT}_{\text{SSS}}$.

# Relating SSS and SetCover

**Lemma.** $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

**Proof.** Consider an optimal superstring $s$.
Construct a set cover of cost $\leq 2|s| = 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

# Relating SSS and SetCover

**Lemma.** $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

**Proof.** Consider an optimal superstring $s$.
Construct a set cover of cost $\leq 2|s| = 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

# Relating SSS and SetCover

**Lemma.** $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

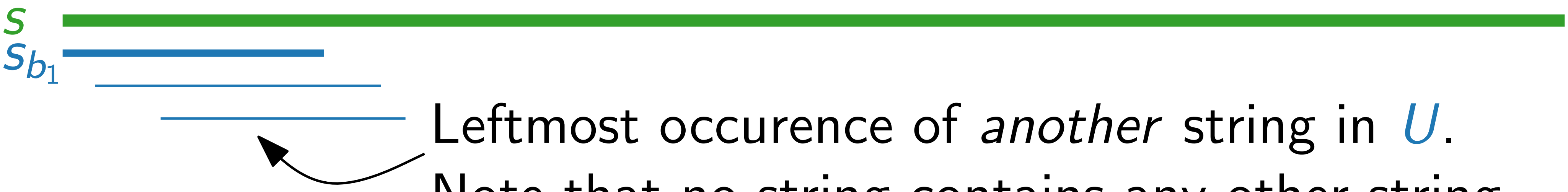
**Proof.** Consider an optimal superstring $s$.
Construct a set cover of cost $\leq 2|s| = 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.


Leftmost occurence of *another* string in $U$.
Note that no string contains any other string.

# Relating SSS and SetCover

**Lemma.** $\text{OPT}_{\text{SC}} \leq 2 \cdot \text{OPT}_{\text{SSS}}$.

**Proof.** Consider an optimal superstring $s$.
Construct a set cover of cost $\leq 2|s| = 2 \cdot \text{OPT}_{\text{SSS}}$.

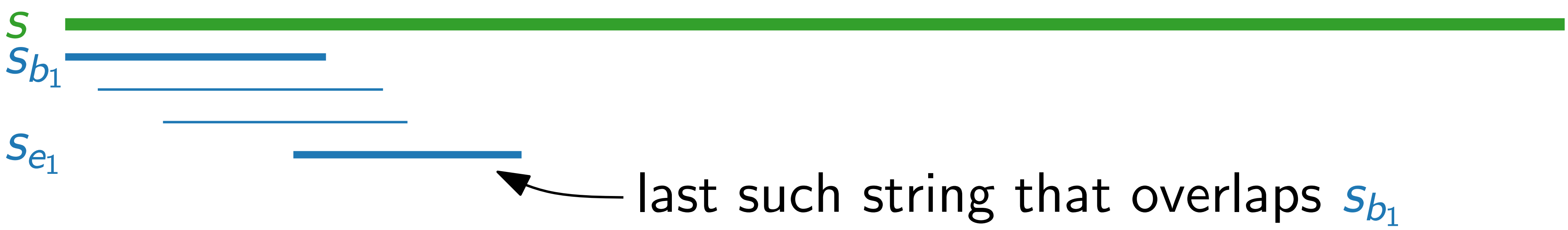



Leftmost occurence of *another* string in $U$.
Note that no string contains any other string.
$\Rightarrow$ Right endpoints are ordered like left endpoints.

# Relating SSS and SetCover

**Lemma.** $\text{OPT}_{\text{SC}} \leq 2 \cdot \text{OPT}_{\text{SSS}}$.

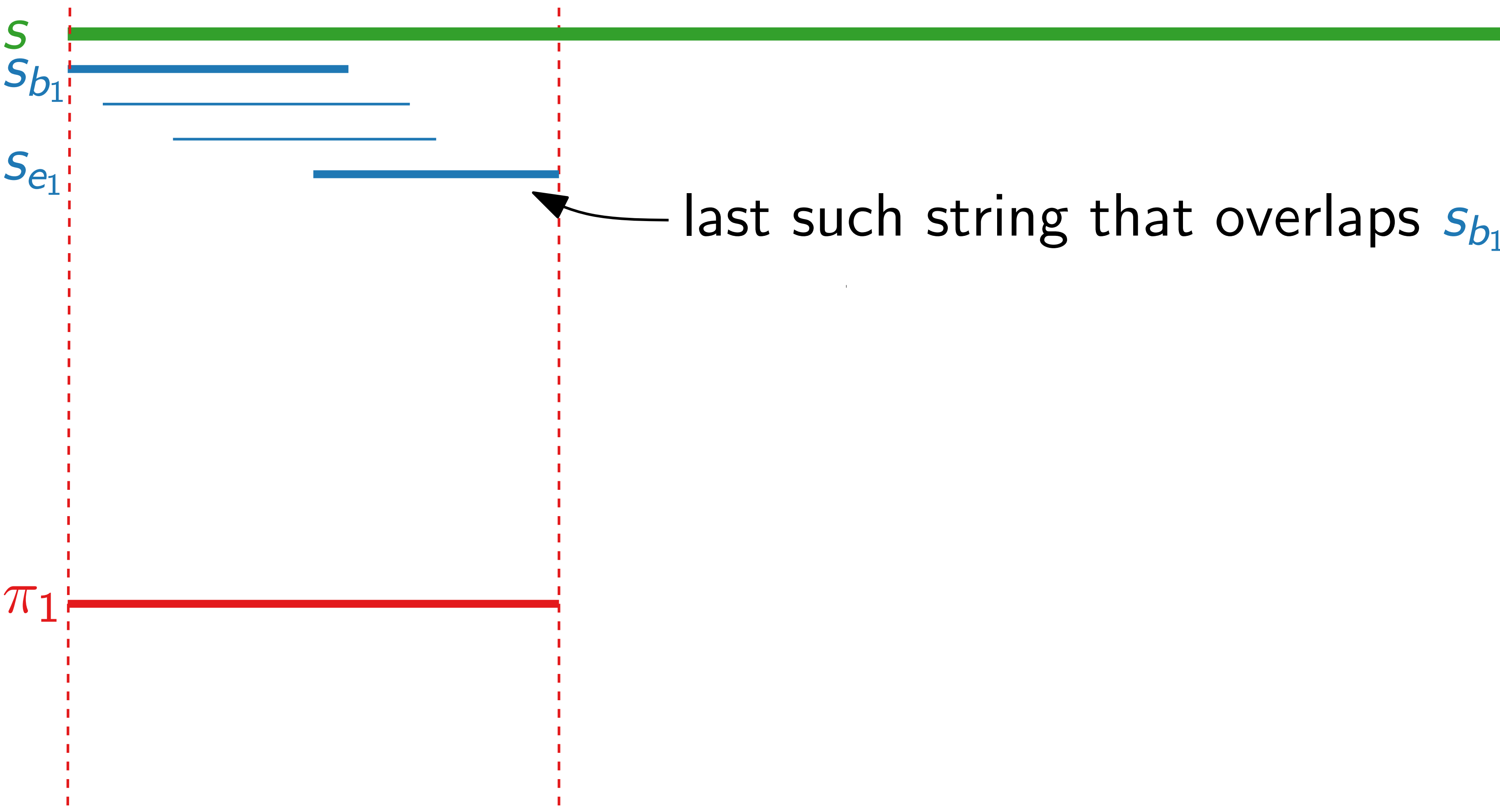
**Proof.** Consider an optimal superstring $s$.
Construct a set cover of cost $\leq 2|s| = 2 \cdot \text{OPT}_{\text{SSS}}$.



# Relating SSS and SetCover

**Lemma.** $\text{OPT}_{\text{SC}} \leq 2 \cdot \text{OPT}_{\text{SSS}}$.

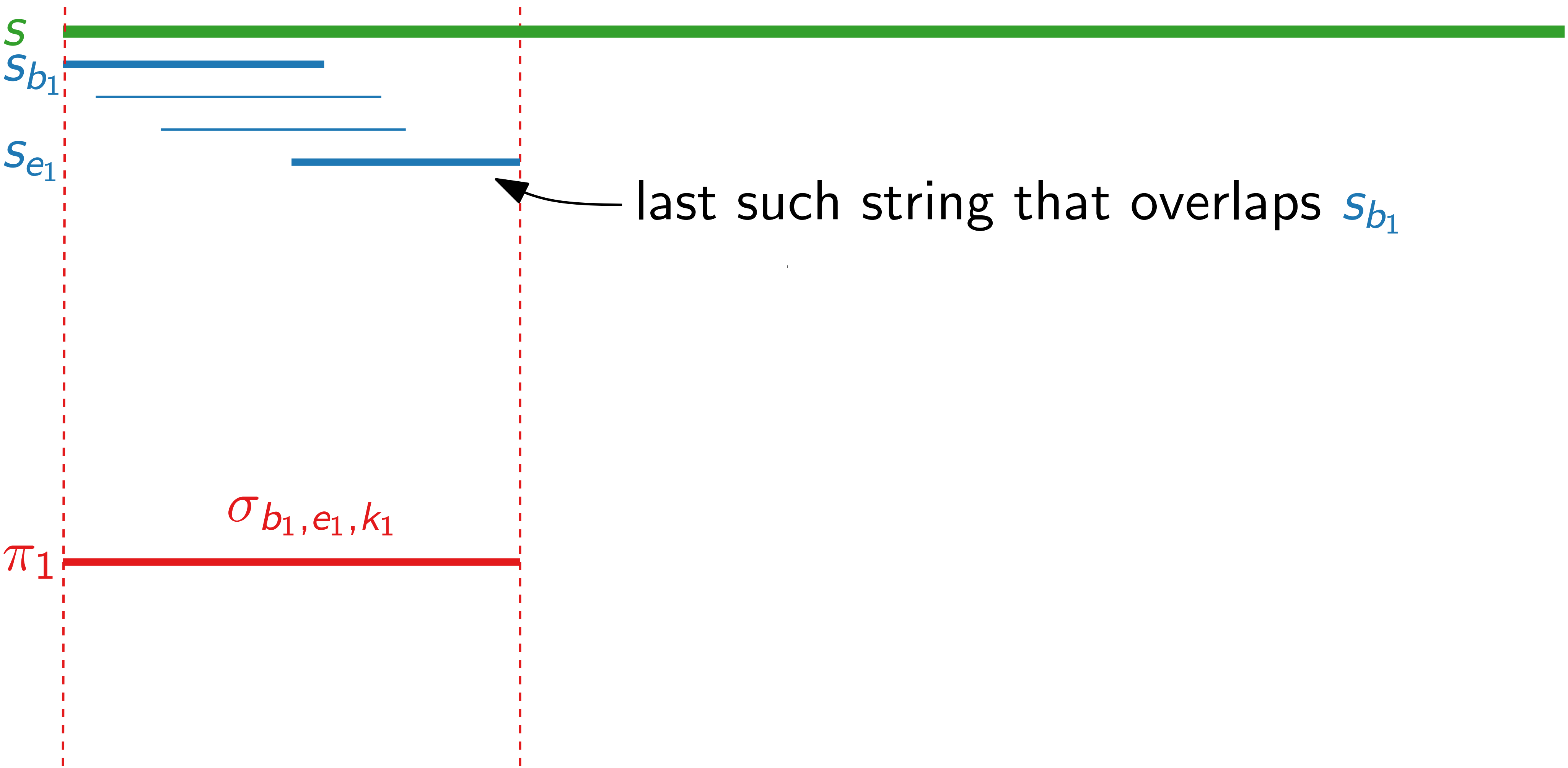
**Proof.** Consider an optimal superstring $s$.
Construct a set cover of cost $\leq 2|s| = 2 \cdot \text{OPT}_{\text{SSS}}$.



# Relating SSS and SetCover

**Lemma.** $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

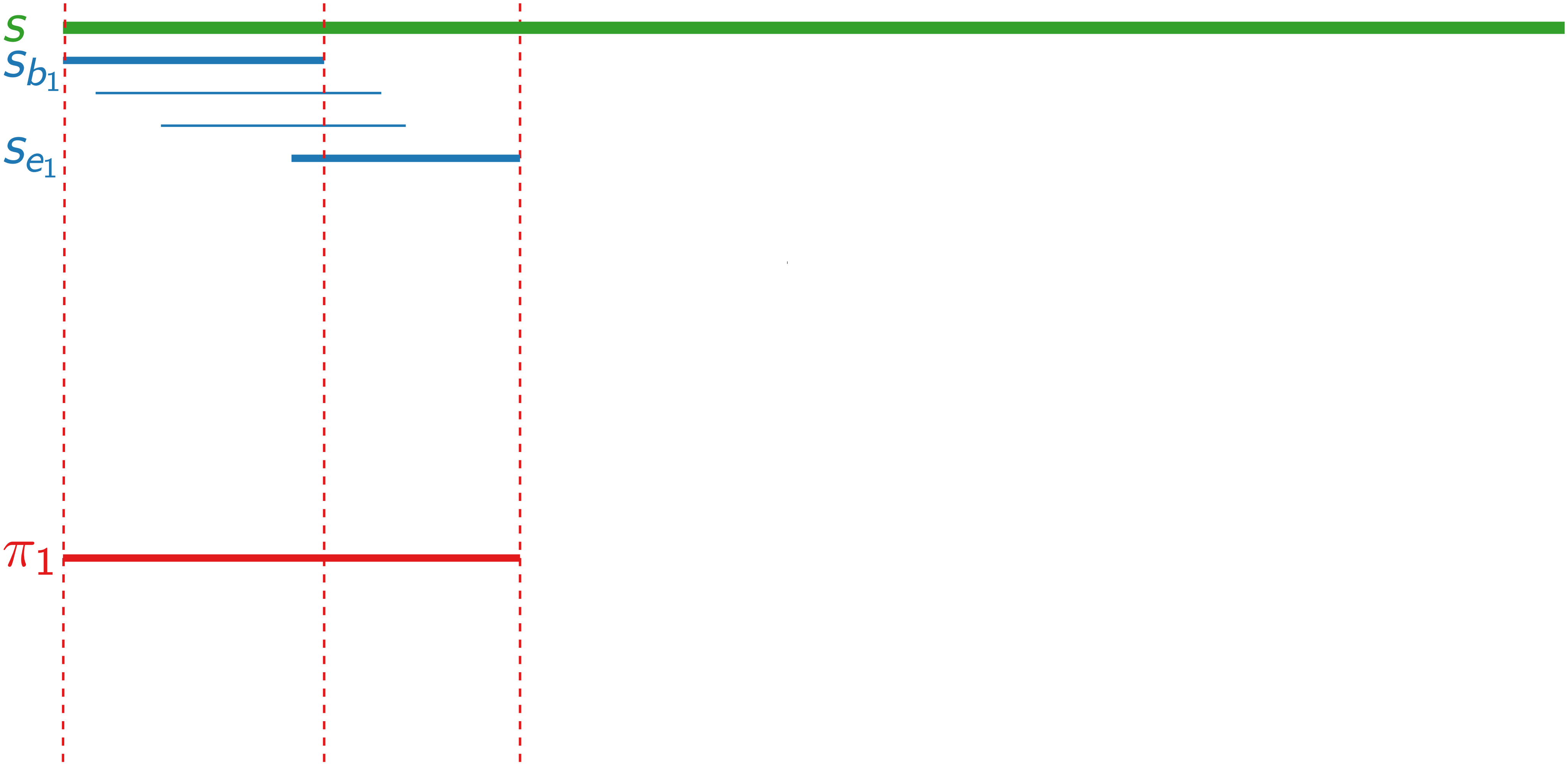
**Proof.** Consider an optimal superstring $s$.
Construct a set cover of cost $\leq 2|s| = 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.



# Relating SSS and SetCover

**Lemma.** $\text{OPT}_{\text{SC}} \leq 2 \cdot \text{OPT}_{\text{SSS}}$.

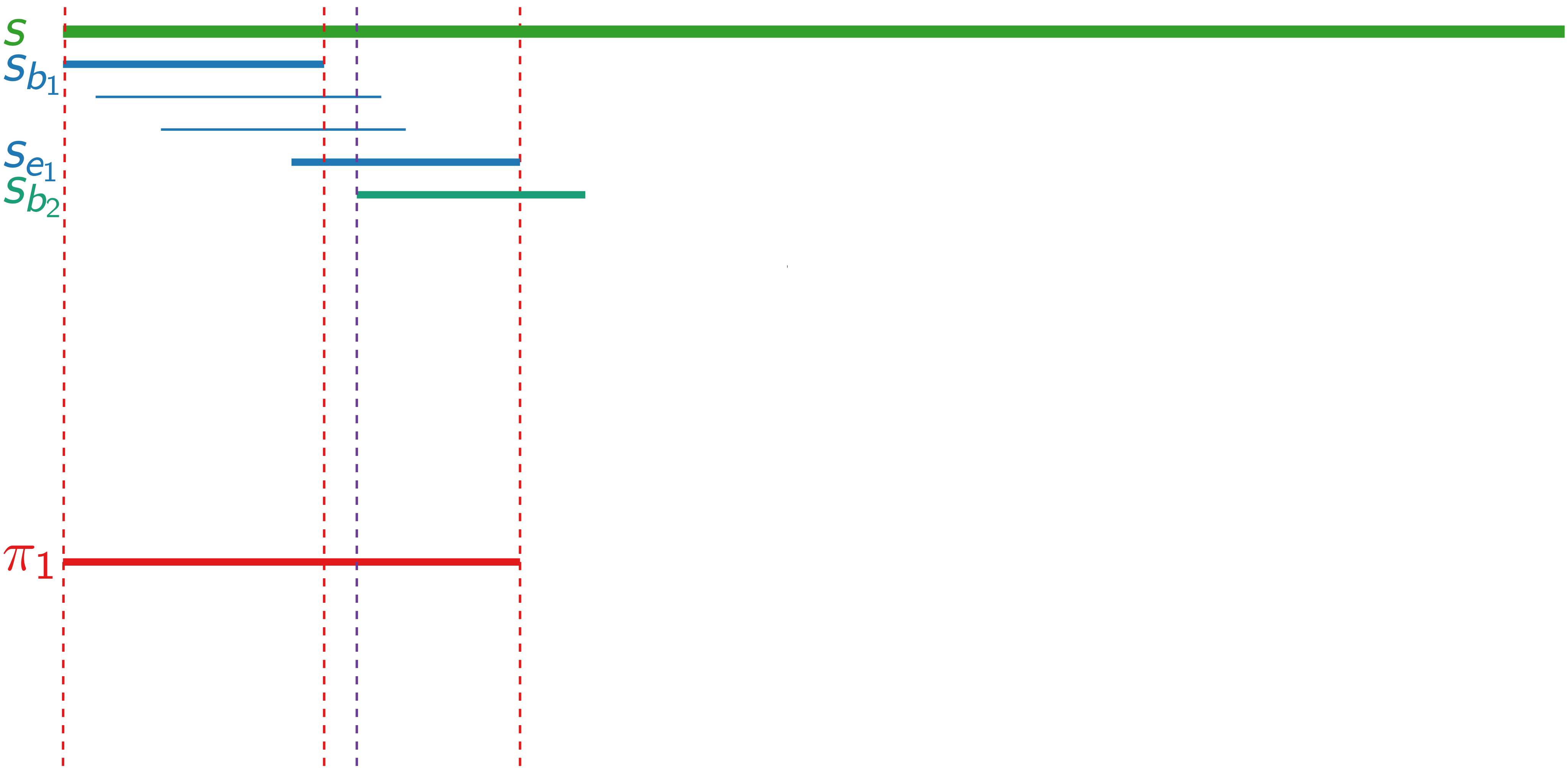
**Proof.** Consider an optimal superstring $s$.
Construct a set cover of cost $\leq 2|s| = 2 \cdot \text{OPT}_{\text{SSS}}$.



# Relating SSS and SetCover

**Lemma.** $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

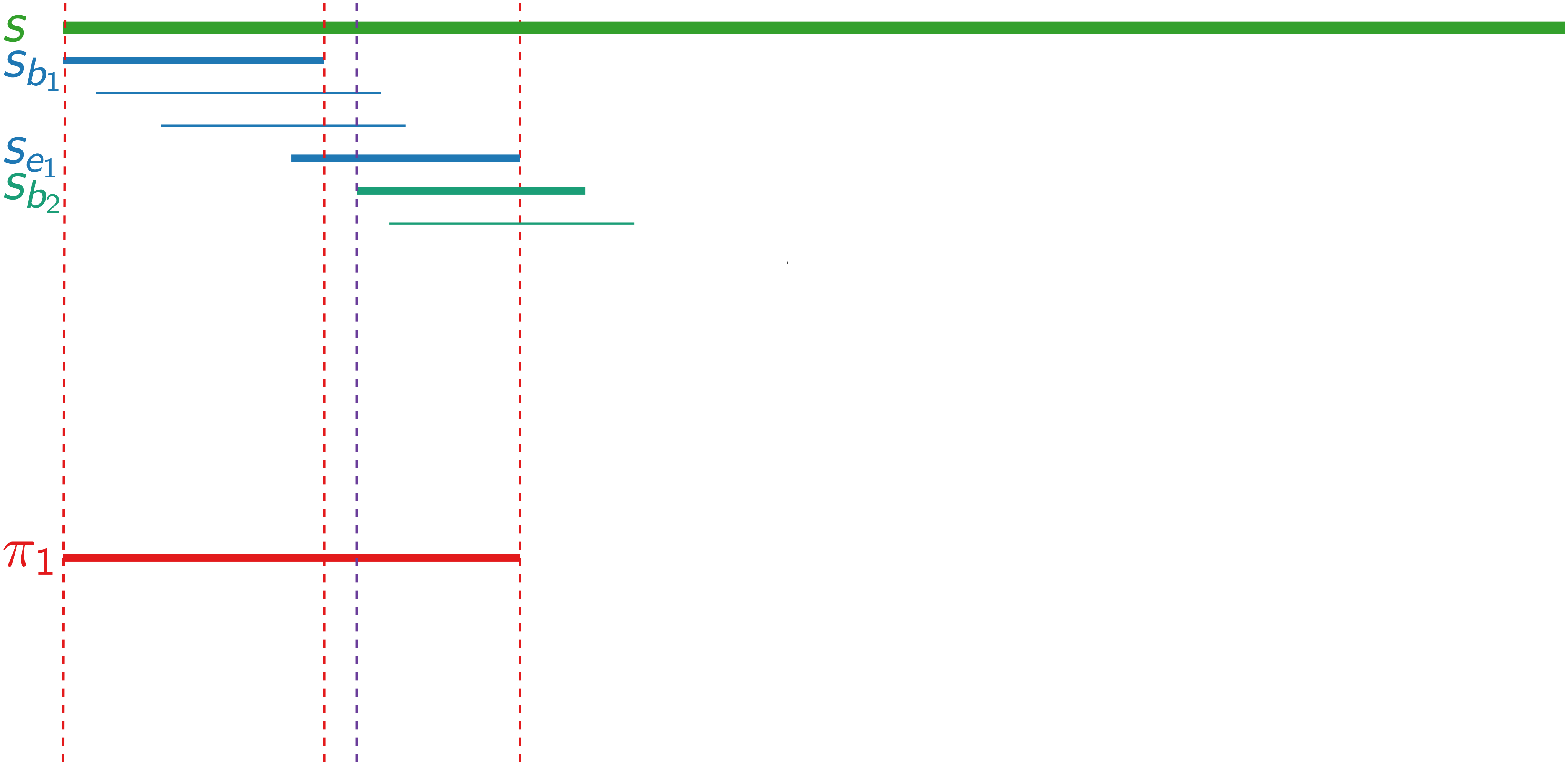
**Proof.** Consider an optimal superstring $s$.
Construct a set cover of cost $\leq 2|s| = 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.



# Relating SSS and SETCOVER

**Lemma.** $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

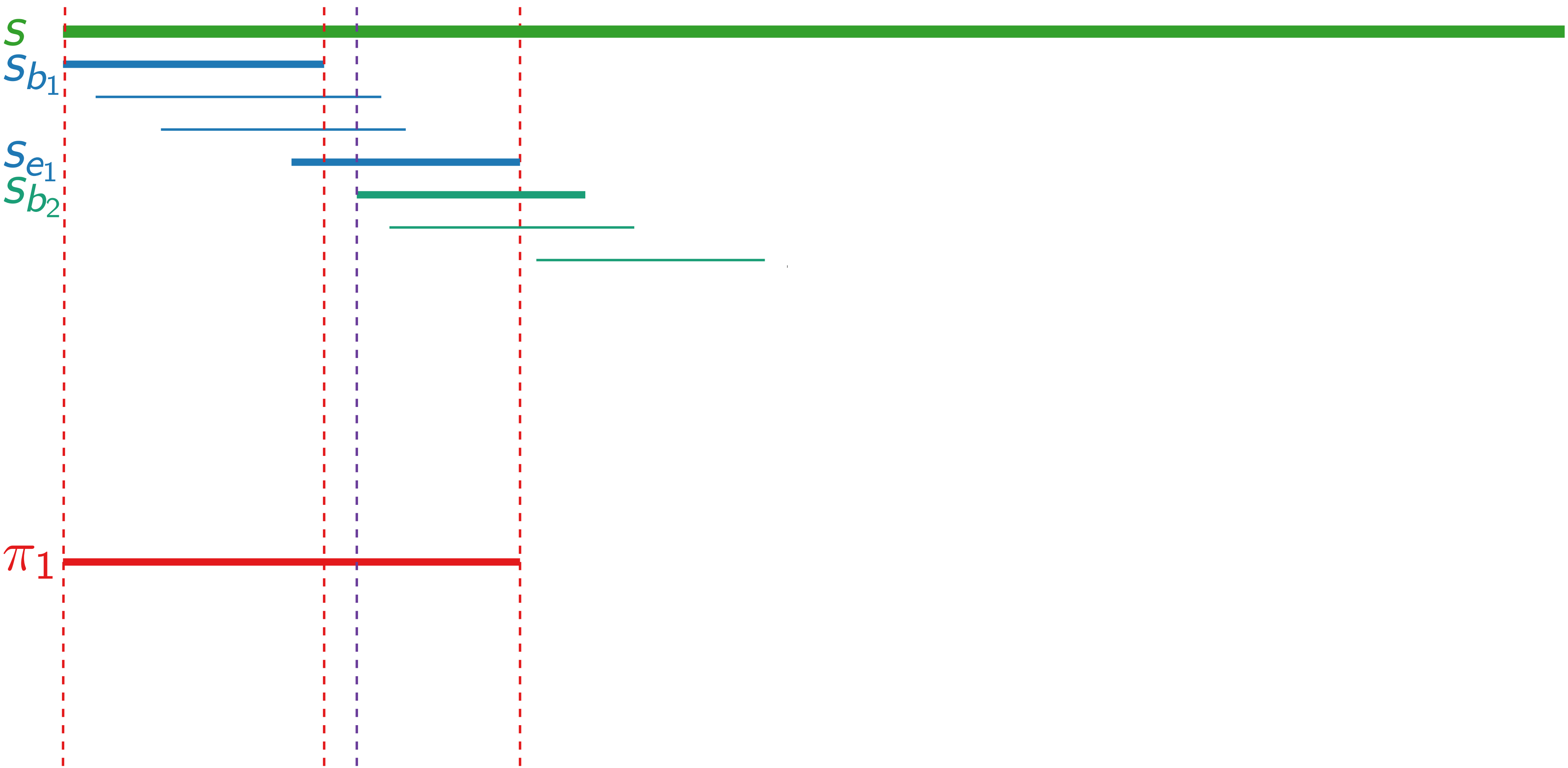
**Proof.** Consider an optimal superstring $s$.
Construct a set cover of cost $\leq 2|s| = 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.



# Relating SSS and SetCover

**Lemma.** $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

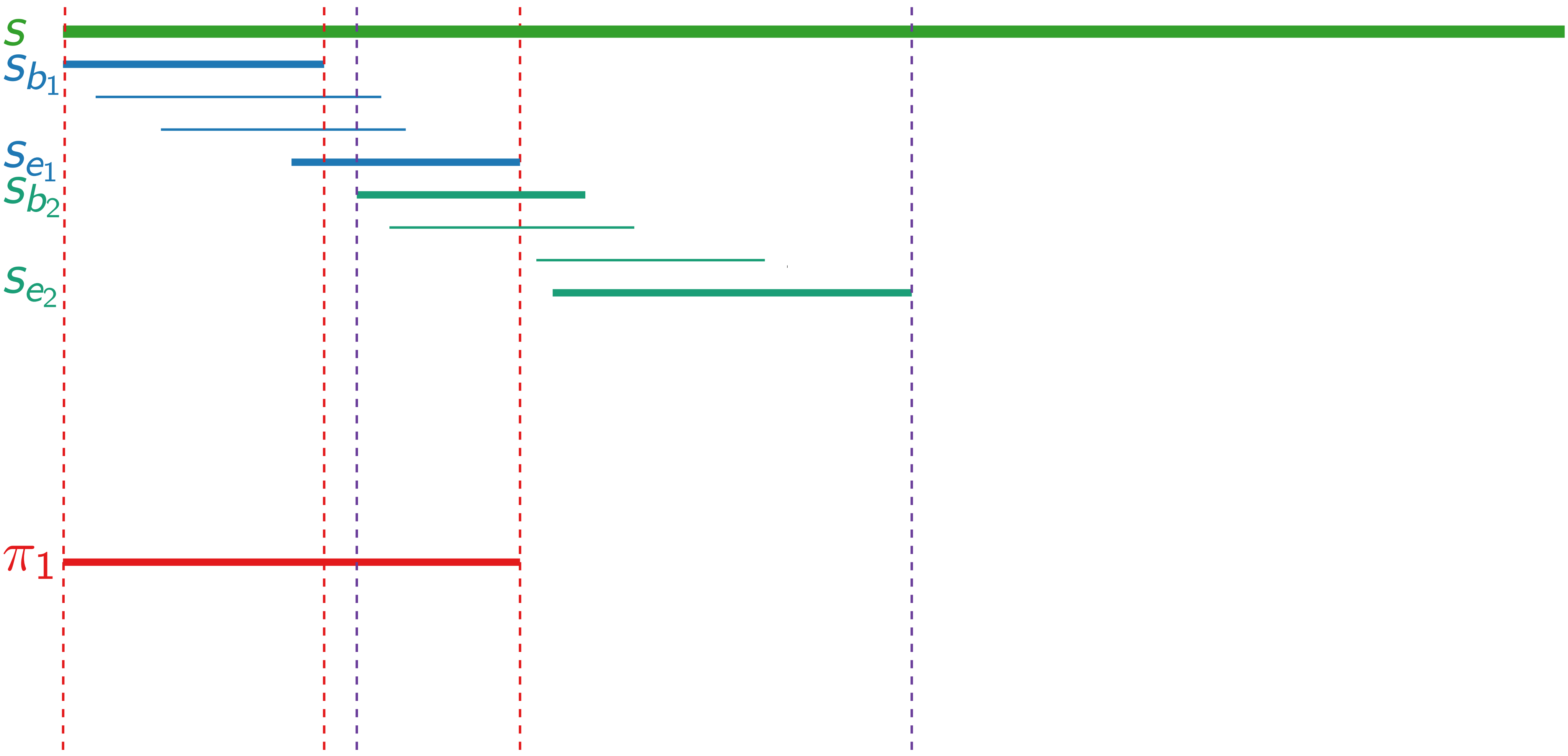
**Proof.** Consider an optimal superstring $s$.
Construct a set cover of cost $\leq 2|s| = 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.



# Relating SSS and SetCover

**Lemma.** $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

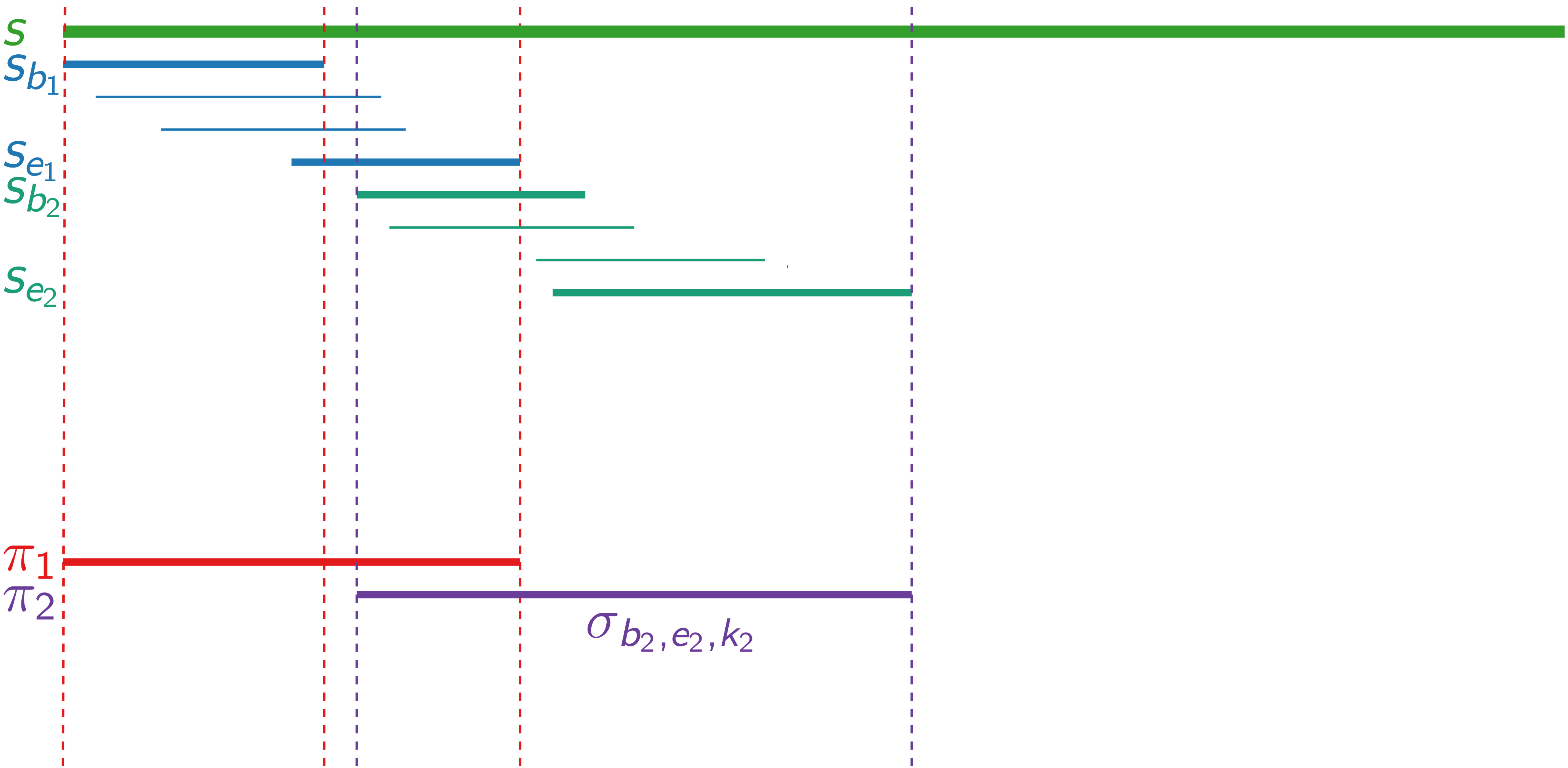
**Proof.** Consider an optimal superstring $s$.
Construct a set cover of cost $\leq 2|s| = 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.



# Relating SSS and SetCover

**Lemma.** $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

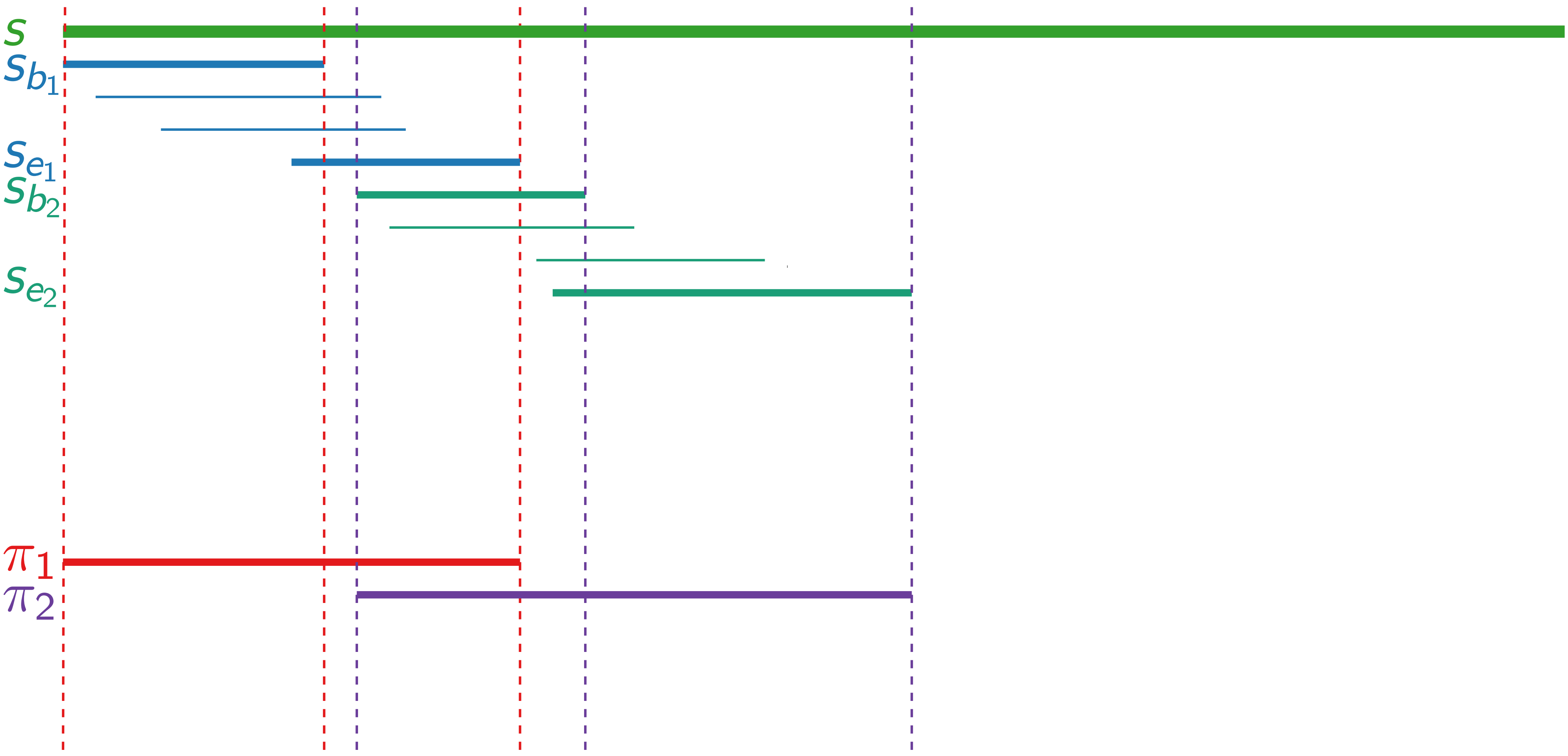
**Proof.** Consider an optimal superstring $s$.
Construct a set cover of cost $\leq 2|s| = 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.



# Relating SSS and SetCover

**Lemma.** $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

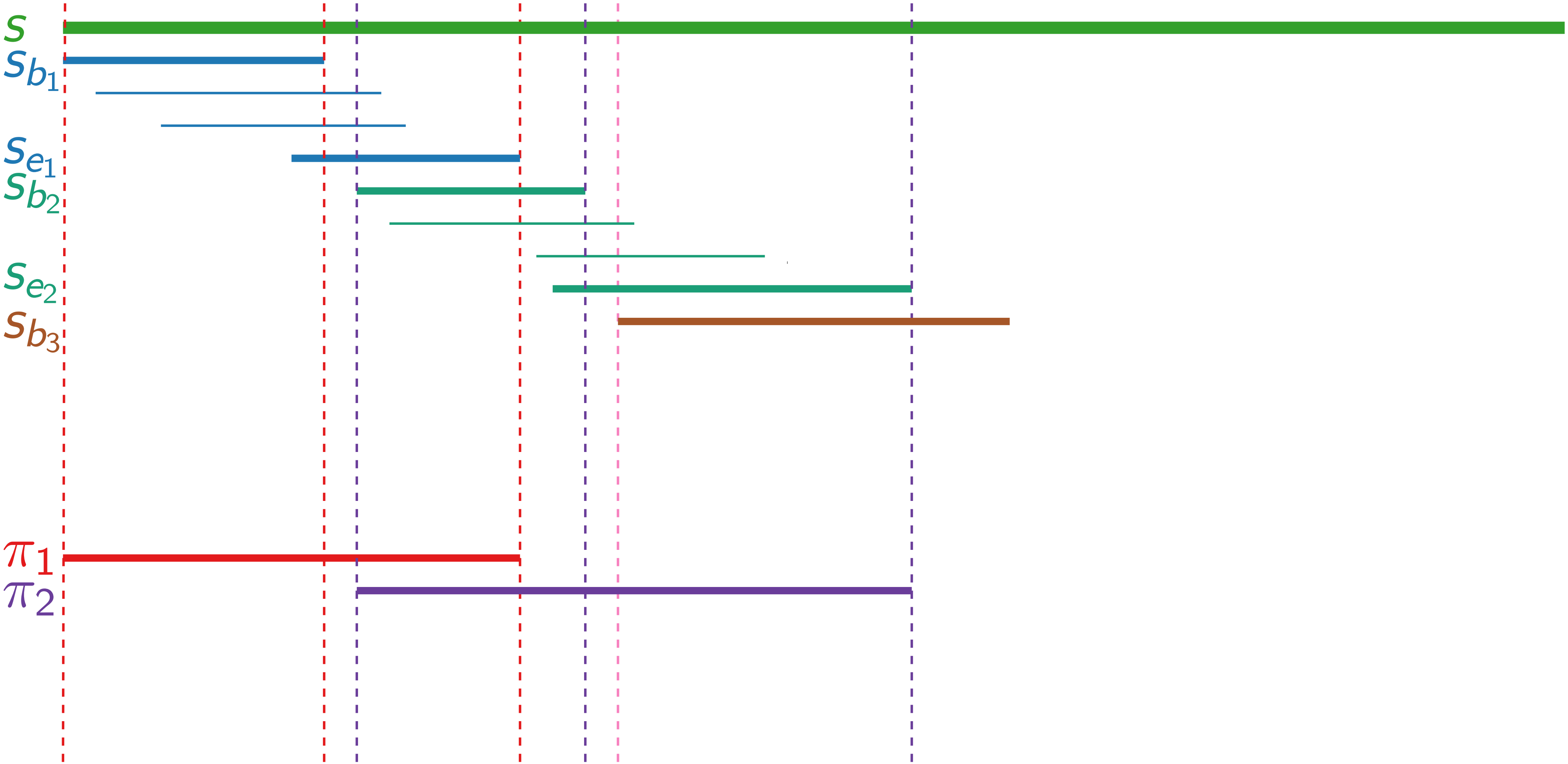
**Proof.** Consider an optimal superstring $s$.
Construct a set cover of cost $\leq 2|s| = 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.



# Relating SSS and SetCover

**Lemma.** $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

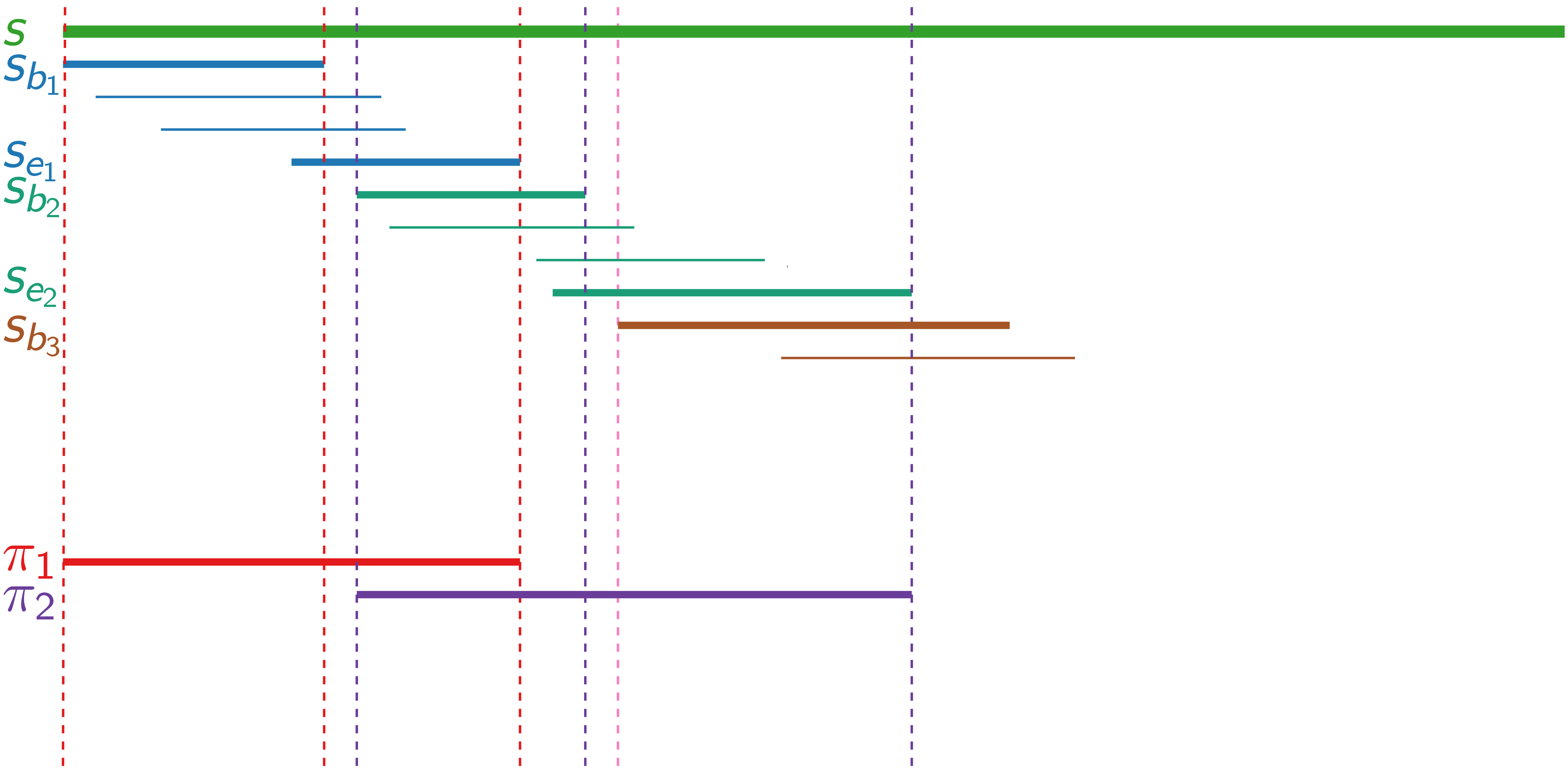
**Proof.** Consider an optimal superstring $s$.
Construct a set cover of cost $\leq 2|s| = 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.



# Relating SSS and SetCover

**Lemma.** $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

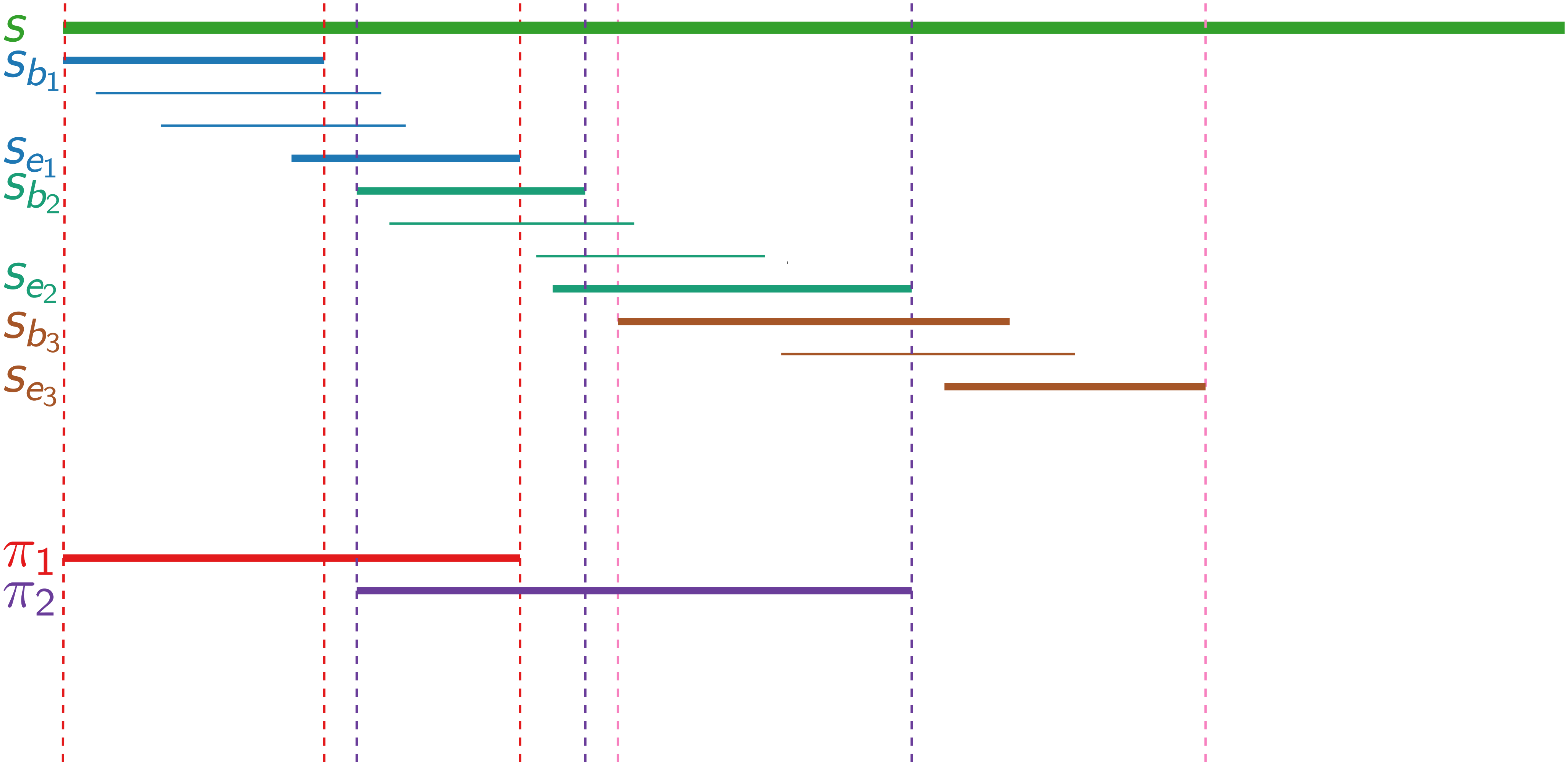
**Proof.** Consider an optimal superstring $s$.
Construct a set cover of cost $\leq 2|s| = 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.



# Relating SSS and SetCover

**Lemma.** $\text{OPT}_{\text{SC}} \leq 2 \cdot \text{OPT}_{\text{SSS}}$.

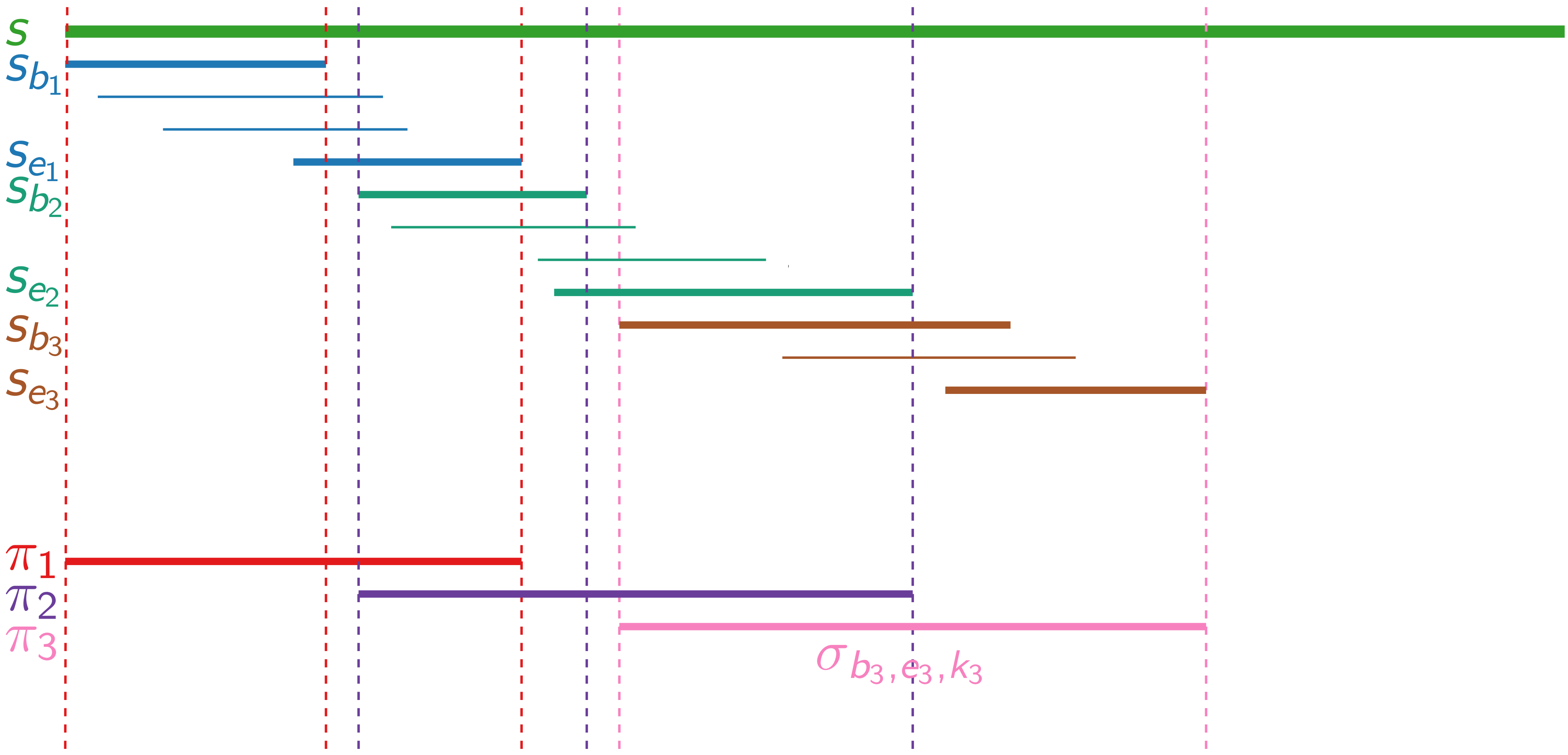
**Proof.** Consider an optimal superstring $s$.
Construct a set cover of cost $\leq 2|s| = 2 \cdot \text{OPT}_{\text{SSS}}$.



# Relating SSS and SetCover

**Lemma.** $\mathsf{OPT}_{\mathsf{SC}} \leq 2 \cdot \mathsf{OPT}_{\mathsf{SSS}}$.

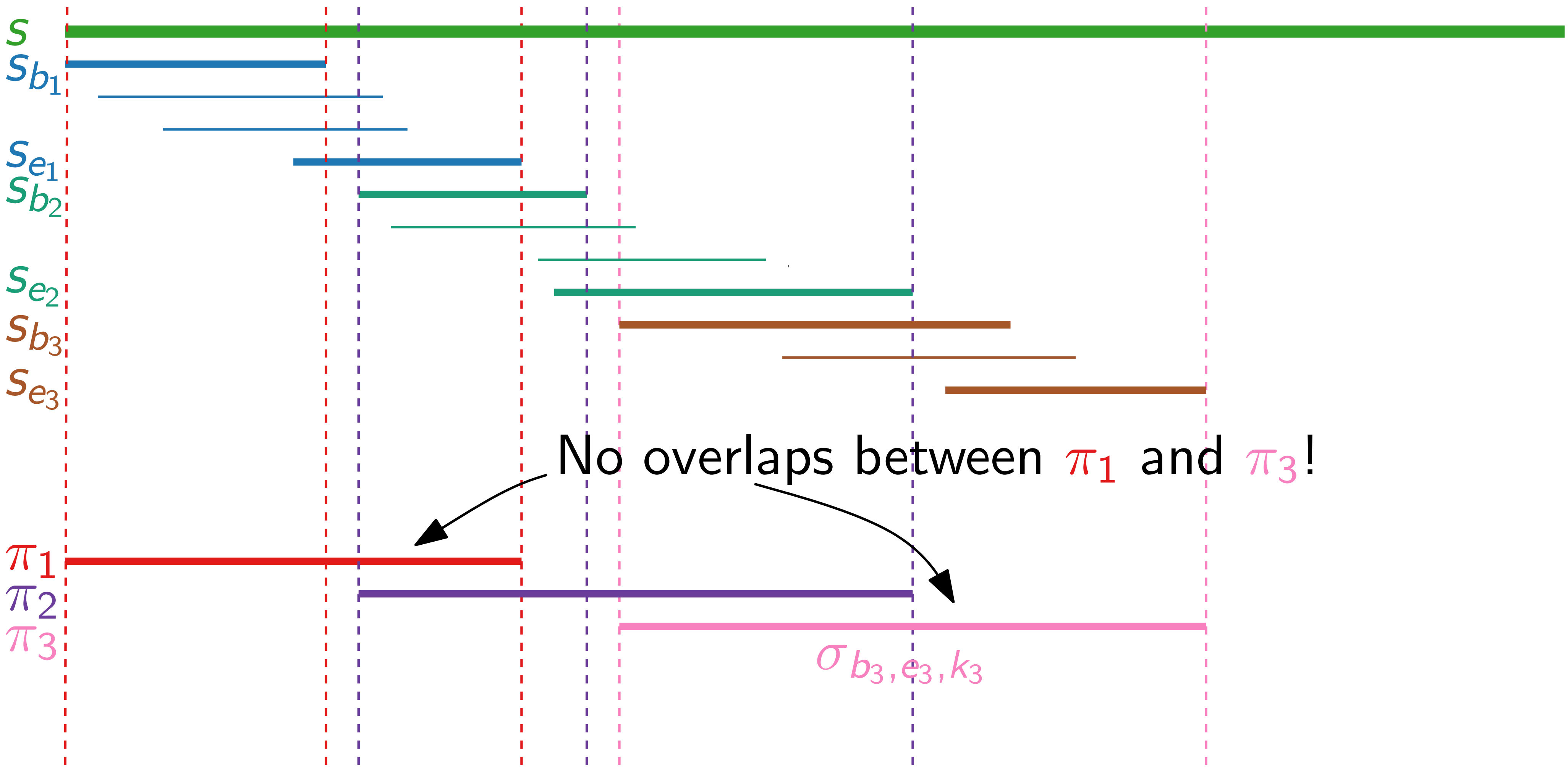
**Proof.** Consider an optimal superstring $s$.
Construct a set cover of cost $\leq 2|s| = 2 \cdot \mathsf{OPT}_{\mathsf{SSS}}$.



# Relating SSS and SetCover

**Lemma.** $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

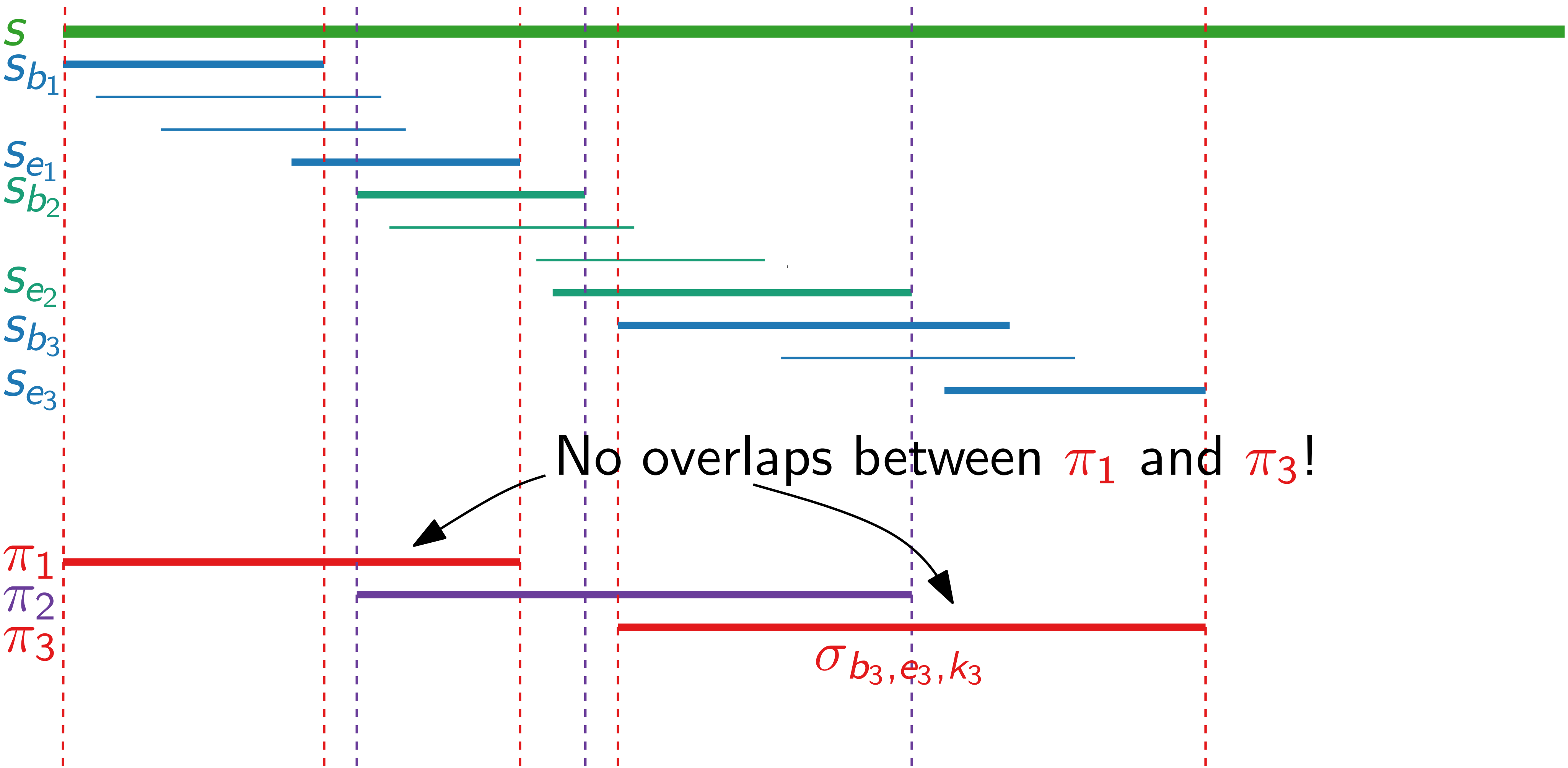
**Proof.** Consider an optimal superstring $s$.
Construct a set cover of cost $\leq 2|s| = 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.



# Relating SSS and SetCover

**Lemma.** $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

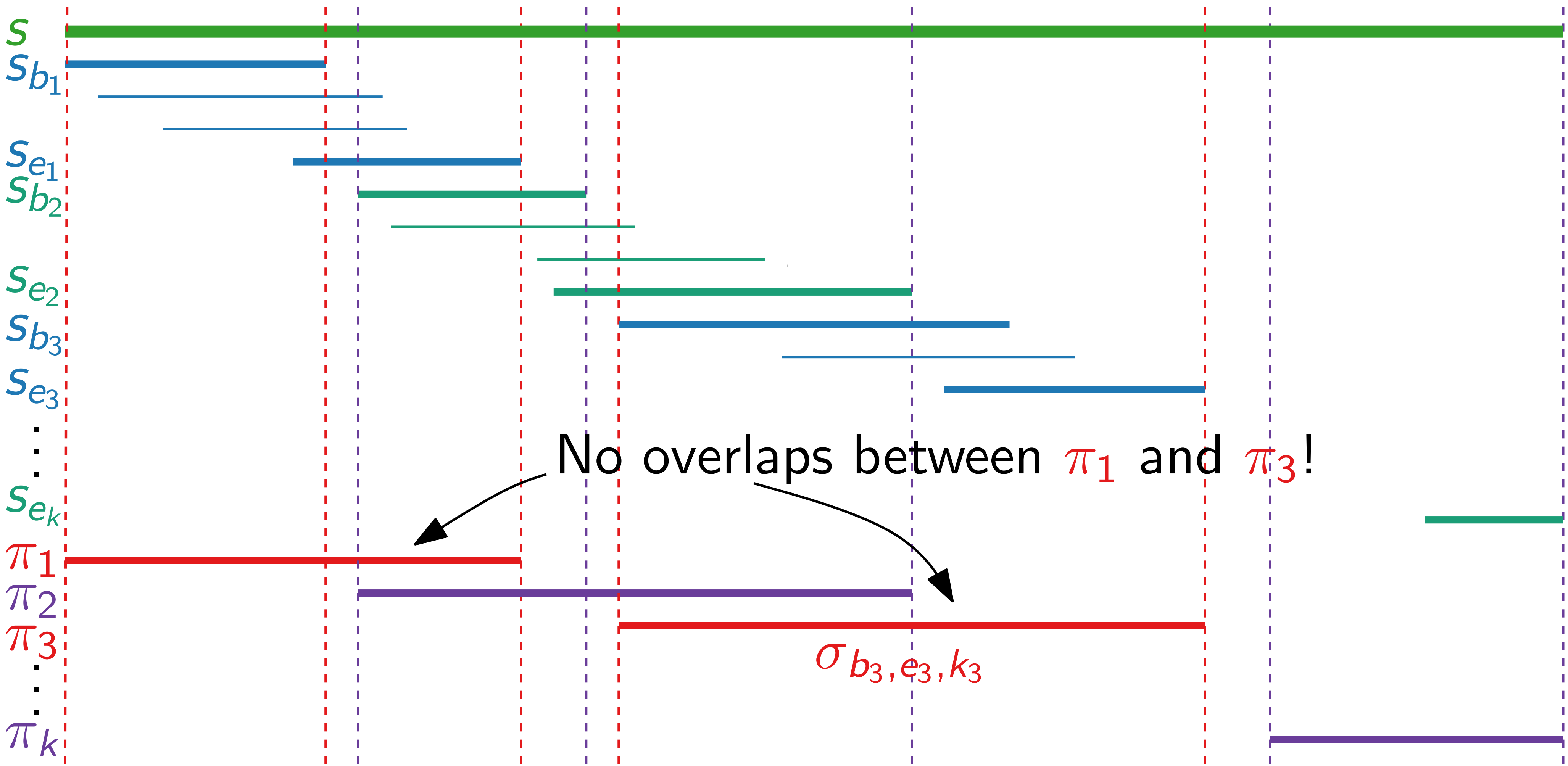
**Proof.** Consider an optimal superstring $s$.
Construct a set cover of cost $\leq 2|s| = 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.



# Relating SSS and SetCover

**Lemma.** $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

**Proof.** Consider an optimal superstring $s$.
Construct a set cover of cost $\leq 2|s| = 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.



# Relating SSS and SetCover

**Lemma.** $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

**Proof.**

Each string $s_i \in U$ is a substring of some $\pi_j$.

# Relating SSS and SetCover

**Lemma.** $\text{OPT}_{\text{SC}} \leq 2 \cdot \text{OPT}_{\text{SSS}}$.

**Proof.**

Each string $s_i \in U$ is a substring of some $\pi_j$.

$\{S(\pi_1), \ldots, S(\pi_k)\}$ is **a** solution for the SetCover instance with cost $\sum_i |\pi_i|$.

# Relating SSS and SetCover

**Lemma.** $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

**Proof.**

Each string $s_i \in U$ is a substring of some $\pi_j$.

$\{S(\pi_1), \ldots, S(\pi_k)\}$ is **a** solution for the SetCover instance with cost $\sum_i |\pi_i|$.

For $j \in \{1, \ldots, k-2\}$, substrings $\pi_j, \pi_{j+2}$ do **not** overlap.

# Relating SSS and SetCover

**Lemma.** $\text{OPT}_{\text{SC}} \leq 2 \cdot \text{OPT}_{\text{SSS}}$.

**Proof.**

Each string $s_i \in U$ is a substring of some $\pi_j$.

$\{S(\pi_1), \ldots, S(\pi_k)\}$ is **a** solution for the SetCover instance with cost $\sum_i |\pi_i|$.

For $j \in \{1, \ldots, k-2\}$, substrings $\pi_j, \pi_{j+2}$ do **not** overlap.

Each character of the optimal superstring $s$ lies in at most **two** (subsequent) substrings, say, $\pi_j$ and $\pi_{j+1}$.

# Relating SSS and SetCover

**Lemma.** $\text{OPT}_{\text{SC}} \leq 2 \cdot \text{OPT}_{\text{SSS}}$.

**Proof.**

Each string $s_i \in U$ is a substring of some $\pi_j$.

$\{S(\pi_1), \ldots, S(\pi_k)\}$ is **a** solution for the SetCover instance with cost $\sum_i |\pi_i|$.

For $j \in \{1, \ldots, k-2\}$, substrings $\pi_j, \pi_{j+2}$ do **not** overlap.

Each character of the optimal superstring $s$ lies in at most **two** (subsequent) substrings, say, $\pi_j$ and $\pi_{j+1}$.

$\text{OPT}_{\text{SC}} \leq$

# Relating SSS and SetCover

**Lemma.** $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

**Proof.**

Each string $s_i \in U$ is a substring of some $\pi_j$.

$\{S(\pi_1), \ldots, S(\pi_k)\}$ is **a** solution for the SetCover instance with cost $\sum_i |\pi_i|$.

For $j \in \{1, \ldots, k-2\}$, substrings $\pi_j, \pi_{j+2}$ do **not** overlap.

Each character of the optimal superstring $s$ lies in at most **two** (subsequent) substrings, say, $\pi_j$ and $\pi_{j+1}$.

$\mathrm{OPT}_{\mathrm{SC}} \leq \sum_i |\pi_i| \leq$

# Relating SSS and SetCover

**Lemma.** $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

**Proof.**

Each string $s_i \in U$ is a substring of some $\pi_j$.

$\{S(\pi_1), \ldots, S(\pi_k)\}$ is **a** solution for the SetCover instance with cost $\sum_i |\pi_i|$.

For $j \in \{1, \ldots, k-2\}$, substrings $\pi_j, \pi_{j+2}$ do **not** overlap.

Each character of the optimal superstring $s$ lies in at most **two** (subsequent) substrings, say, $\pi_j$ and $\pi_{j+1}$.

$\mathrm{OPT}_{\mathrm{SC}} \leq \sum_i |\pi_i| \leq 2|s| =$

# Relating SSS and SetCover

**Lemma.** $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

**Proof.**

Each string $s_i \in U$ is a substring of some $\pi_j$.

$\{S(\pi_1), \ldots, S(\pi_k)\}$ is **a** solution for the SetCover instance with cost $\sum_i |\pi_i|$.

For $j \in \{1, \ldots, k-2\}$, substrings $\pi_j, \pi_{j+2}$ do **not** overlap.

Each character of the optimal superstring $s$ lies in at most **two** (subsequent) substrings, say, $\pi_j$ and $\pi_{j+1}$.

$\mathrm{OPT}_{\mathrm{SC}} \leq \sum_i |\pi_i| \leq 2|s| = 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$ □

# Algorithm for SSS

1. Construct SetCover instance $\langle U, \mathcal{S}, c \rangle$.

# Algorithm for SSS

1. Construct SETCOVER instance $\langle U, \mathcal{S}, c \rangle$.
2. Compute a set cover $\{S(\pi_1), \ldots, S(\pi_k)\}$ with the algorithm GreedySetCover.

# Algorithm for SSS

1. Construct SetCover instance $\langle U, \mathcal{S}, c \rangle$.
2. Compute a set cover $\{S(\pi_1), \ldots, S(\pi_k)\}$ with the algorithm GreedySetCover.
3. Return $\pi_1 \circ \cdots \circ \pi_k$ as the superstring.

# Algorithm for SSS

1. Construct SetCover instance $\langle U, \mathcal{S}, c \rangle$.
2. Compute a set cover $\{S(\pi_1), \ldots, S(\pi_k)\}$ with the algorithm GreedySetCover.
3. Return $\pi_1 \circ \cdots \circ \pi_k$ as the superstring.

**Theorem.** This algorithm is a factor-$2\mathcal{H}_n$ approximation algorithm for ShortestSuperString.

# Algorithm for SSS

1. Construct SetCover instance $\langle U, \mathcal{S}, c \rangle$.
2. Compute a set cover $\{S(\pi_1), \ldots, S(\pi_k)\}$ with the algorithm GreedySetCover.
3. Return $\pi_1 \circ \cdots \circ \pi_k$ as the superstring.

**Theorem.** This algorithm is a factor-$2\mathcal{H}_n$ approximation algorithm for ShortestSuperString.

**Lemma.** $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

# Algorithm for SSS

1. Construct SetCover instance $\langle U, \mathcal{S}, c \rangle$.
2. Compute a set cover $\{S(\pi_1), \dots, S(\pi_k)\}$ with the algorithm GreedySetCover.
3. Return $\pi_1 \circ \dots \circ \pi_k$ as the superstring.

**Theorem.** This algorithm is a factor-$2\mathcal{H}_n$ approximation algorithm for ShortestSuperString.

**Lemma.** $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

**Theorem.** GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where $k$ is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \leq 1 + \ln k$.

# Can we do better?

# Can we do better?

- The best known approximation factor for ShortestSuperString is $(\sqrt{67} + 14)/9 \approx 2.466$.
  [Englert, Matsakis, Veselý: STOC 2022, ISAAC 2023]

# Can we do better?

- The best known approximation factor for ShortestSuperString is $(\sqrt{67}+14)/9 \approx 2.466$.
  [Englert, Matsakis, Veselý: STOC 2022, ISAAC 2023]

- ShortestSuperString cannot be approximated within factor $\frac{333}{332} \approx 1.003$ (unless $\mathsf{P} = \mathsf{NP}$).
  [Karpinski & Schmied: CATS 2013]