

Approximation Algorithms

Lecture 2:

SETCOVER and SHORTESTSUPERSTRING

Part I:

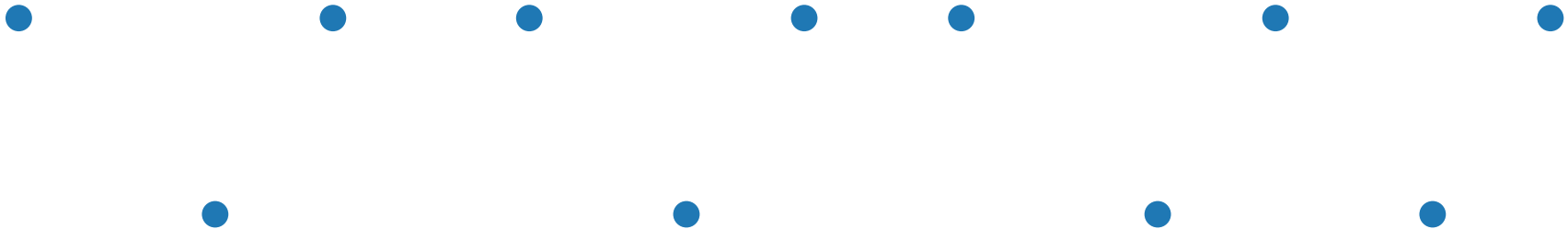
SETCOVER

SETCOVER (card.)

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

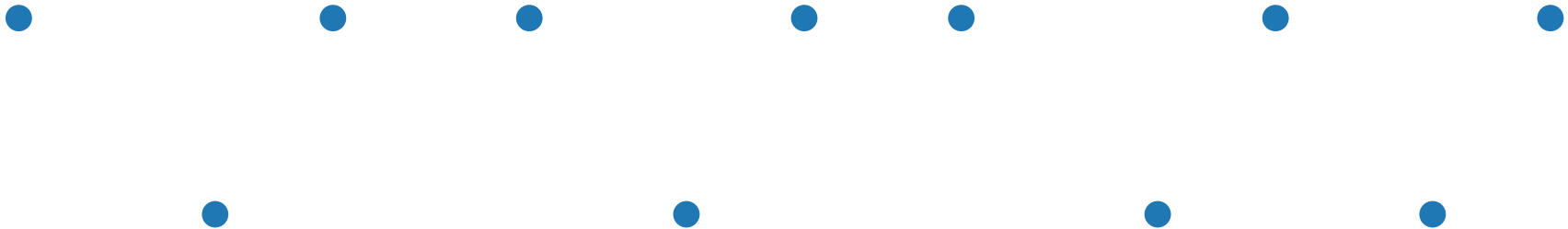
SETCOVER (card.)

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



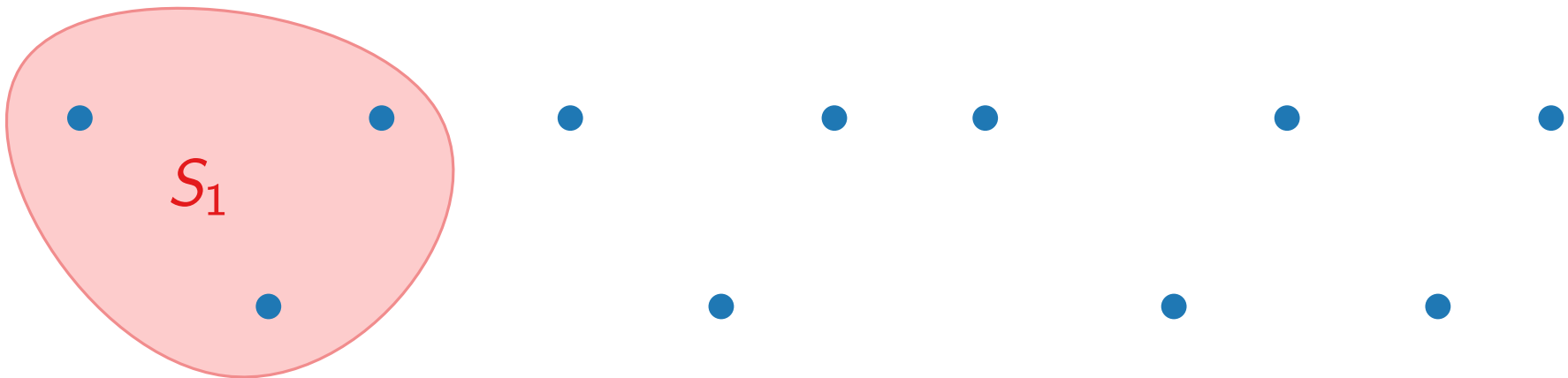
SETCOVER (card.)

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



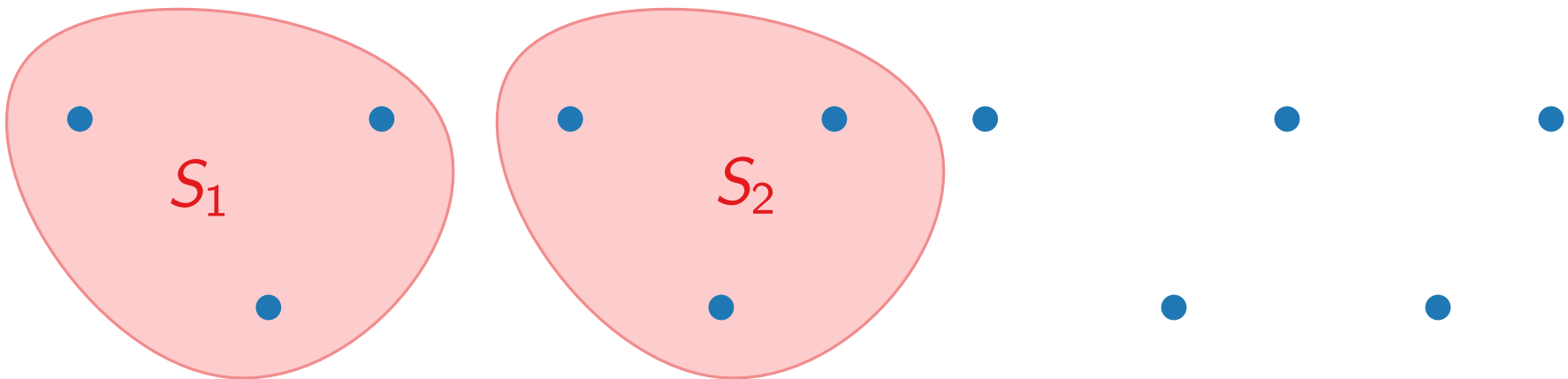
SETCOVER (card.)

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



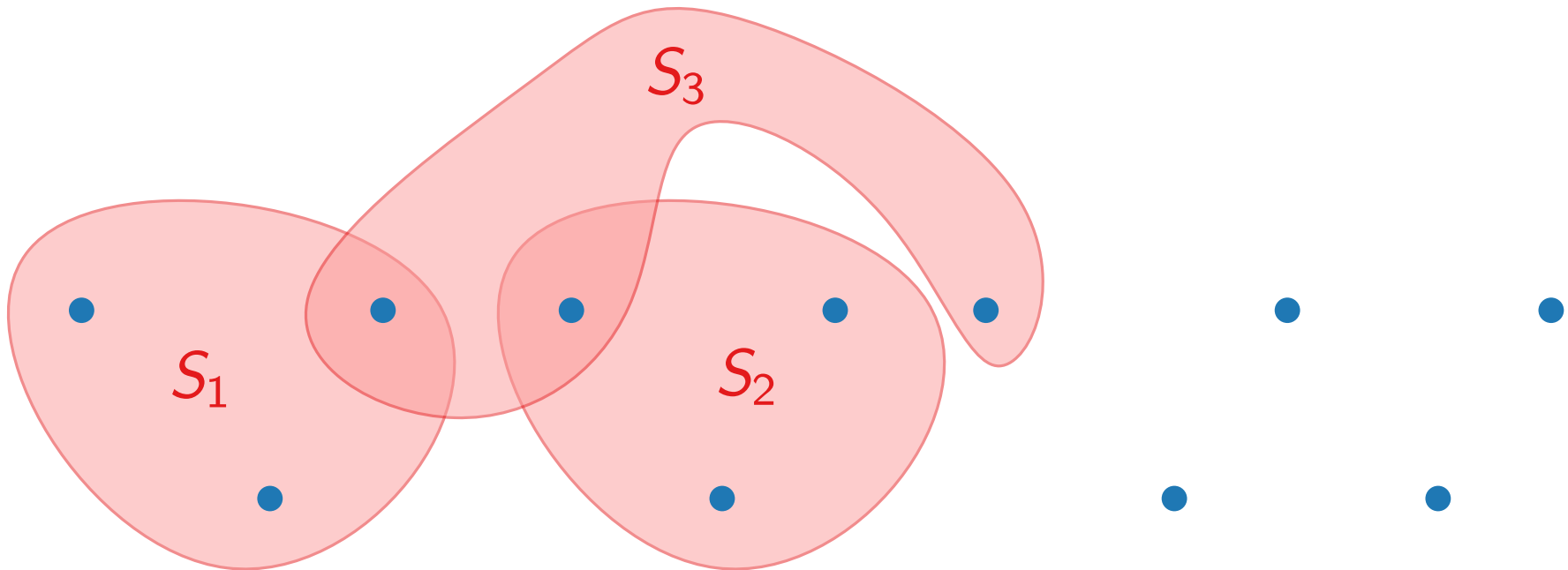
SETCOVER (card.)

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



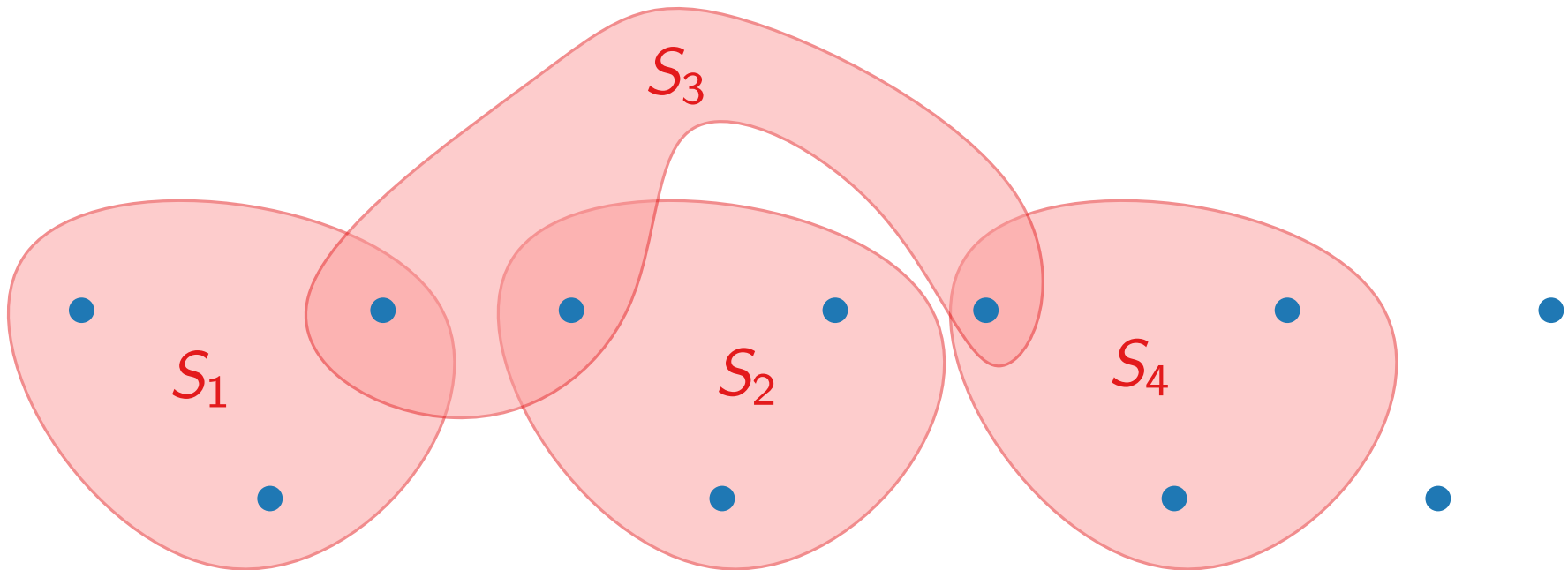
SETCOVER (card.)

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



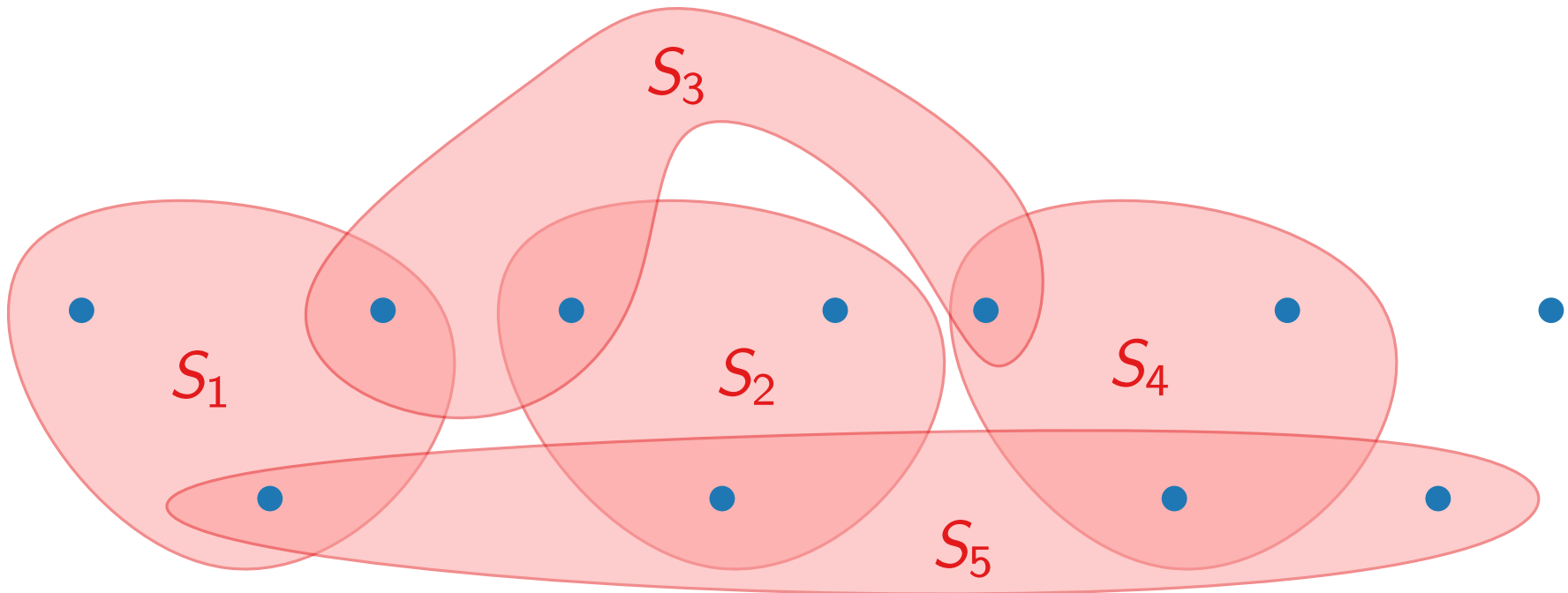
SETCOVER (card.)

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



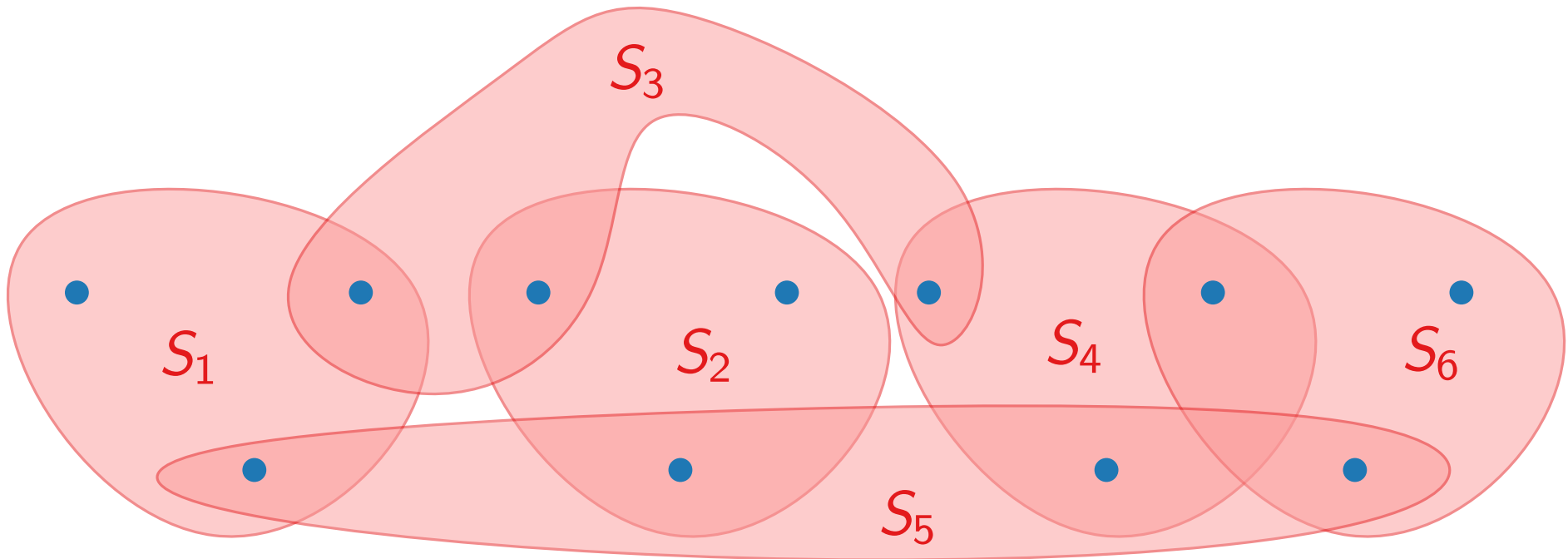
SETCOVER (card.)

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.)

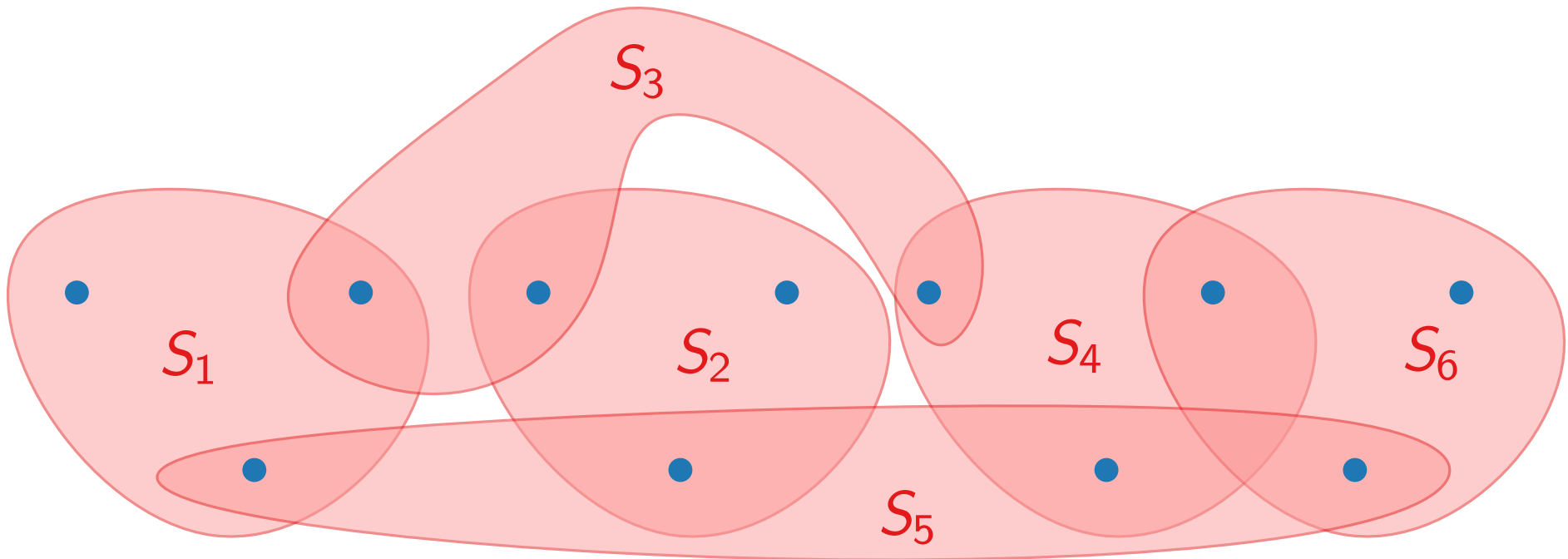
Let U be some **ground set** (universe),
and let \mathcal{S} be a family of **subsets** of U with $\bigcup \mathcal{S} = U$.



SETCOVER (card.)

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

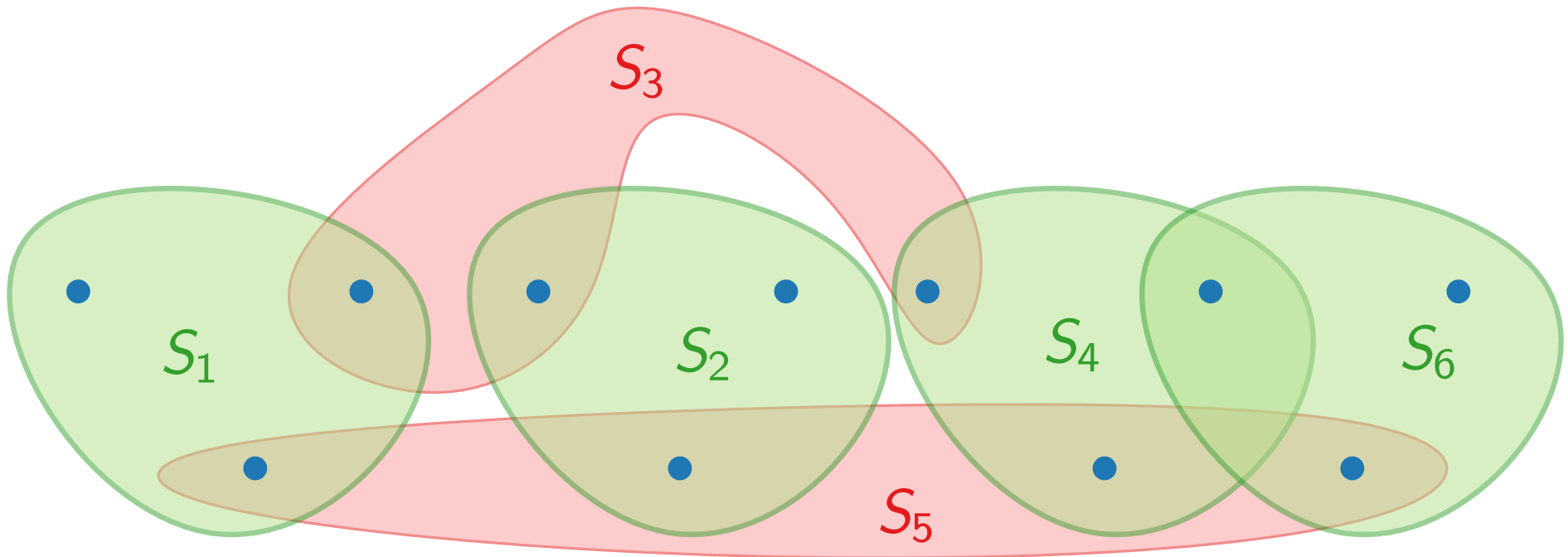
Find a **cover** $\mathcal{S}' \subseteq \mathcal{S}$ of U (i.e., with $\bigcup \mathcal{S}' = U$) of minimum cardinality.



SETCOVER (card.)

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

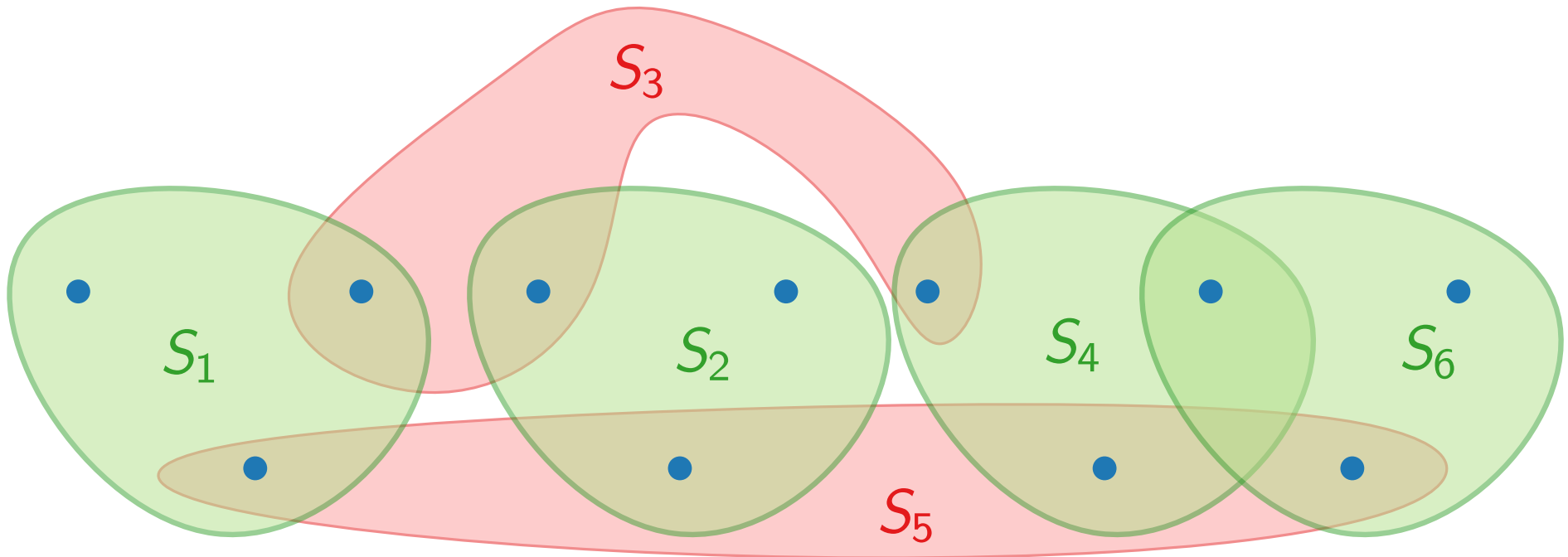
Find a **cover** $\mathcal{S}' \subseteq \mathcal{S}$ of U (i.e., with $\bigcup \mathcal{S}' = U$) of minimum cardinality.



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.

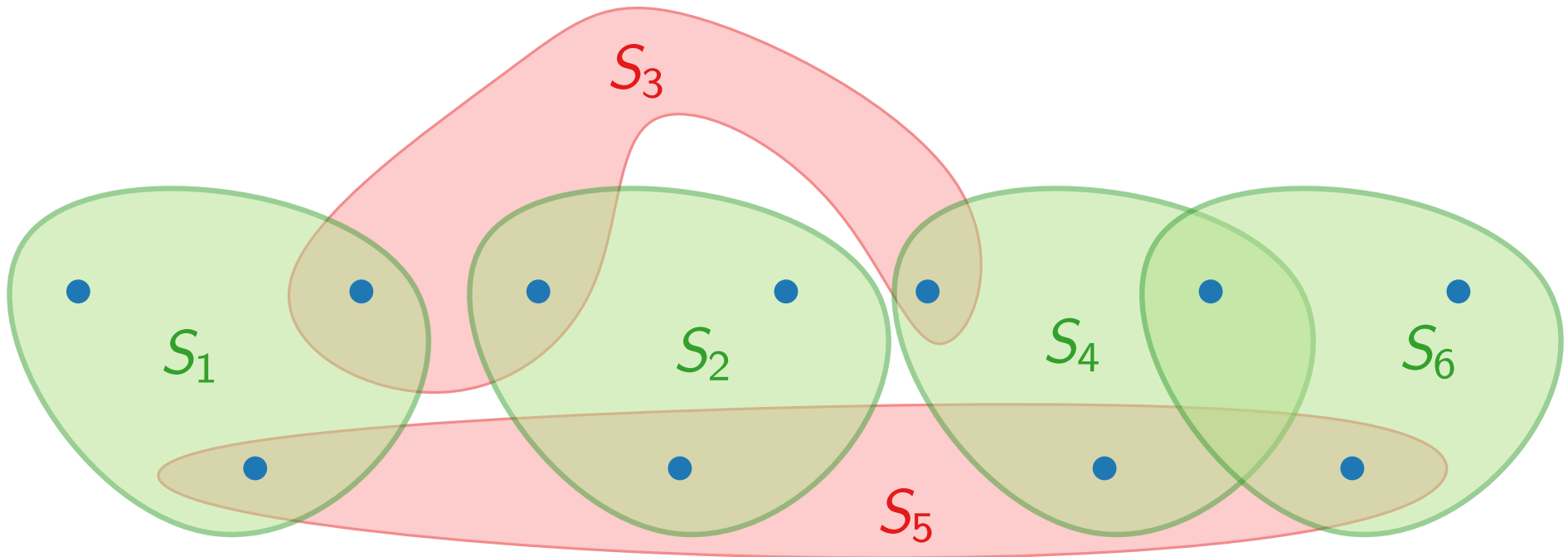


SETCOVER (general)

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.

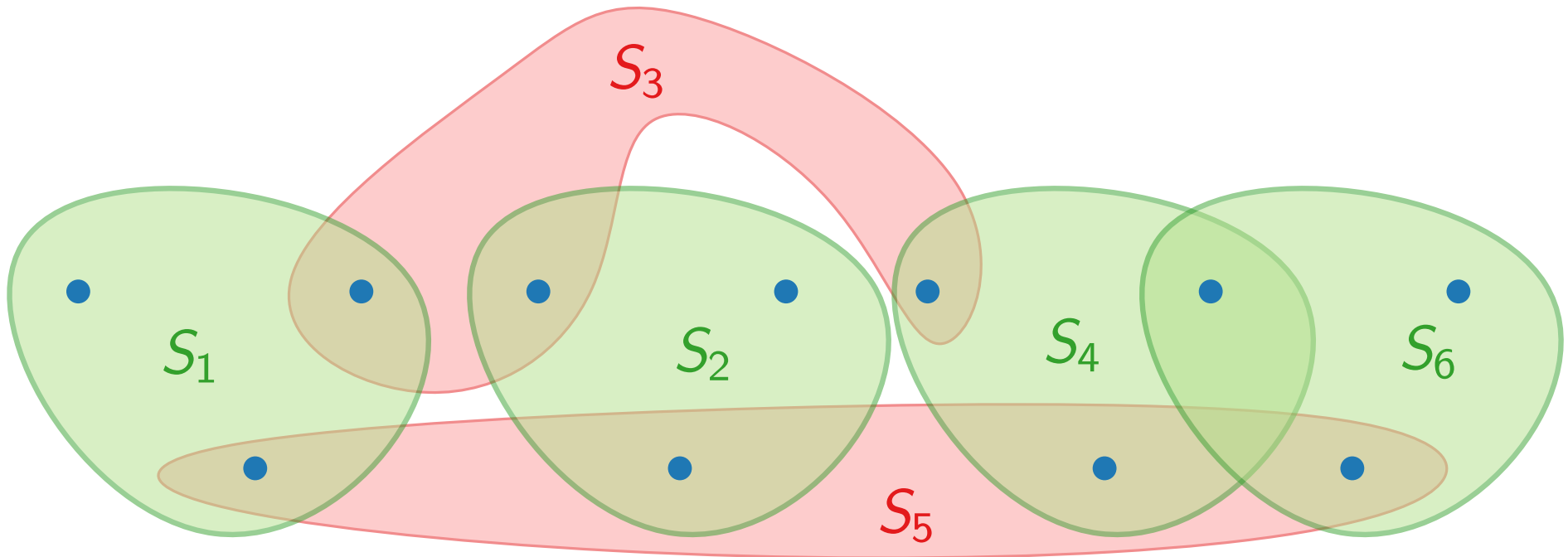


SETCOVER (general)

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 $c(\mathcal{S}') := \sum_{S \in \mathcal{S}'} c(S)$.

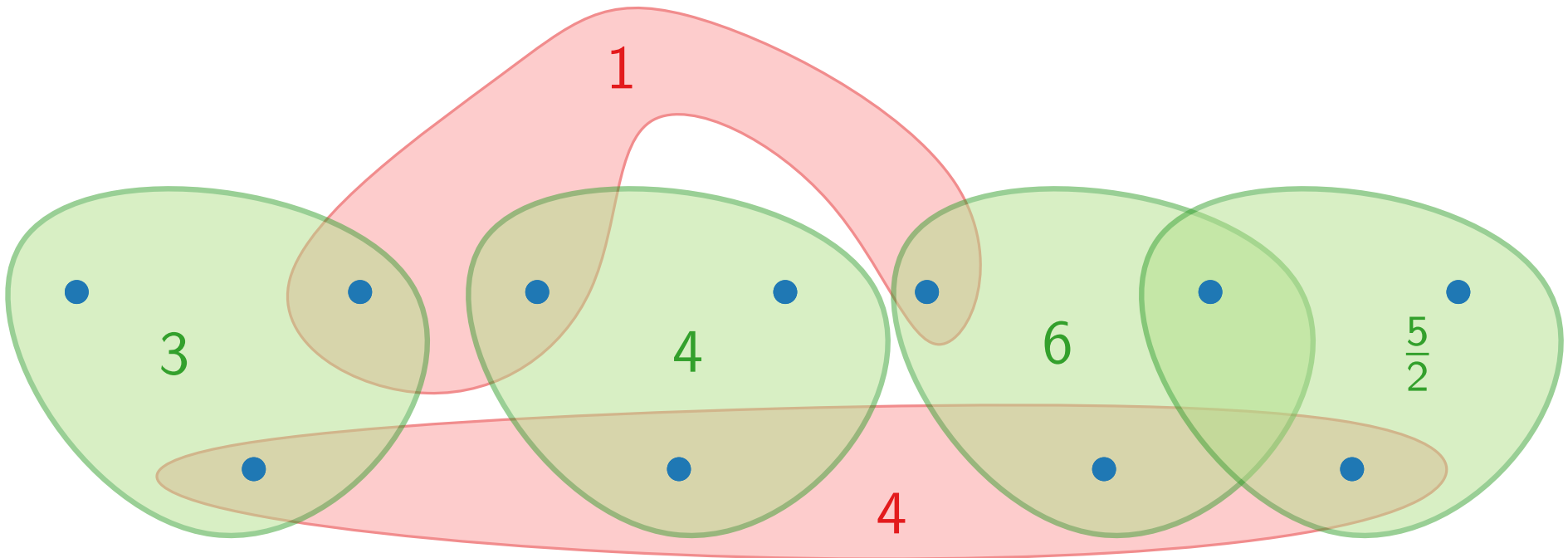


SETCOVER (general)

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 $c(\mathcal{S}') := \sum_{S \in \mathcal{S}'} c(S)$.

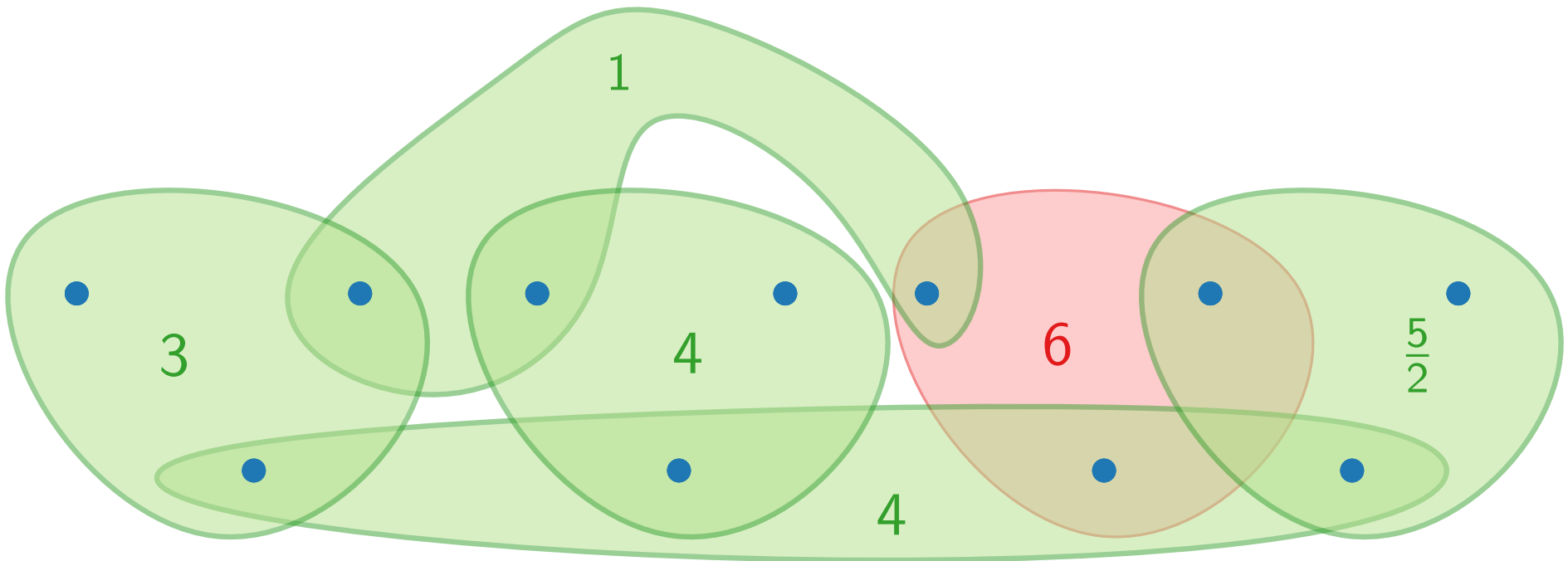


SETCOVER (general)

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:

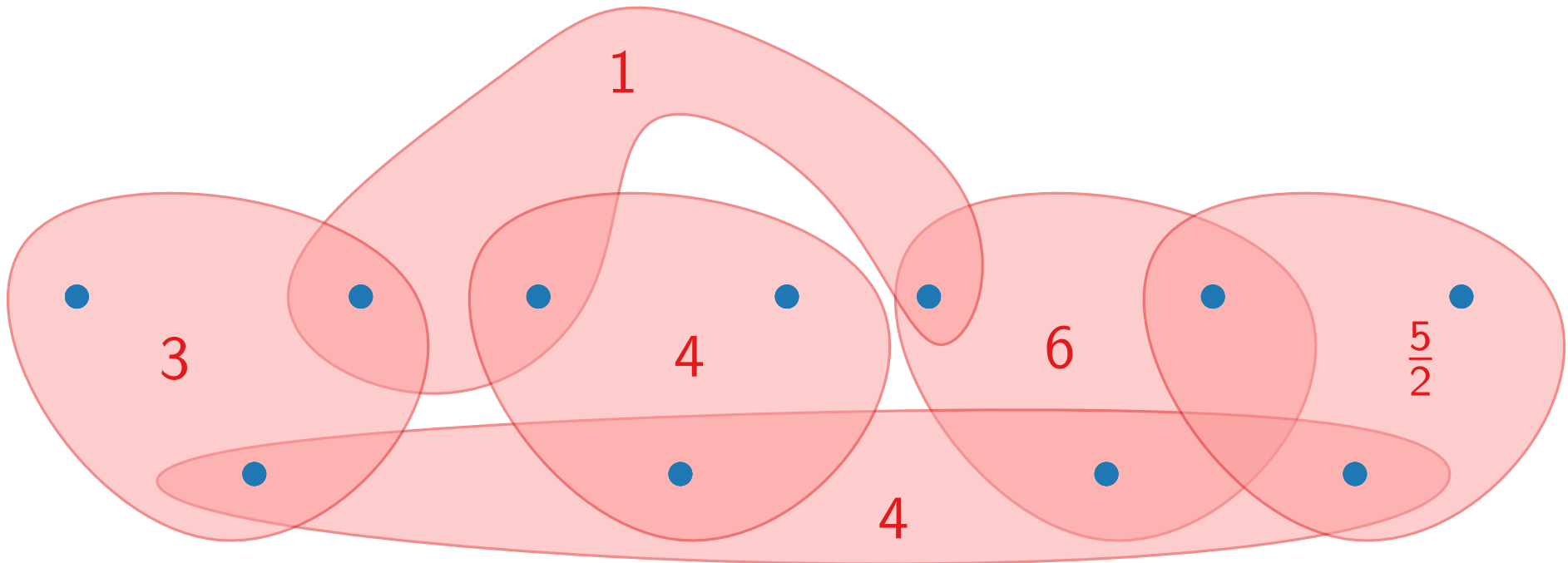
SETCOVER and SHORTESTSUPERSTRING

Part II:

Greedy for SETCOVER

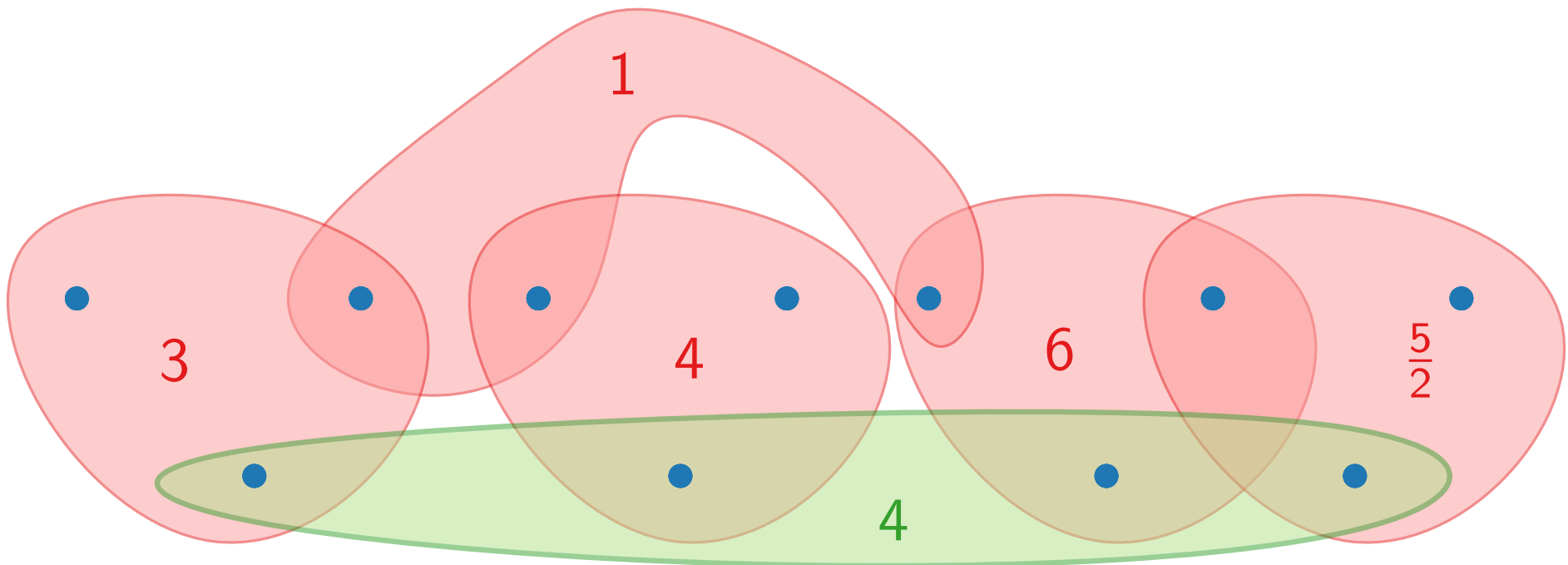
“Buying” Elements Iteratively

What is the real cost of picking a set?



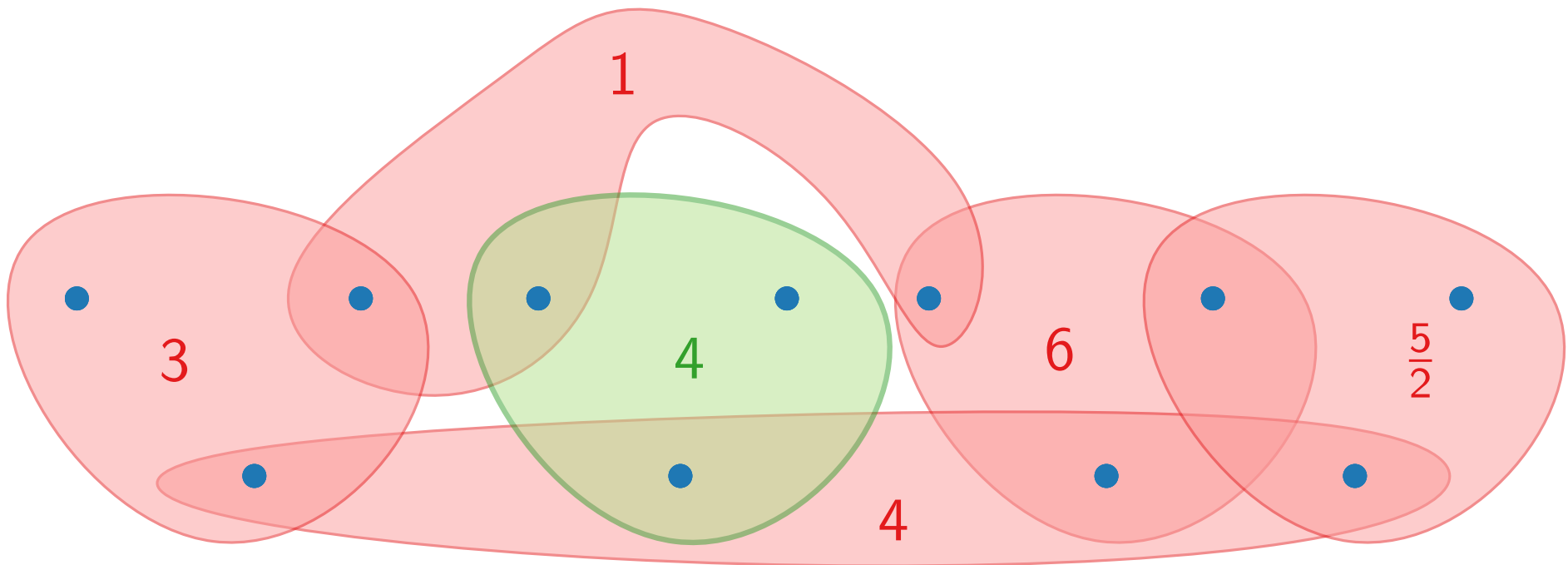
“Buying” Elements Iteratively

What is the real cost of picking a set?



“Buying” Elements Iteratively

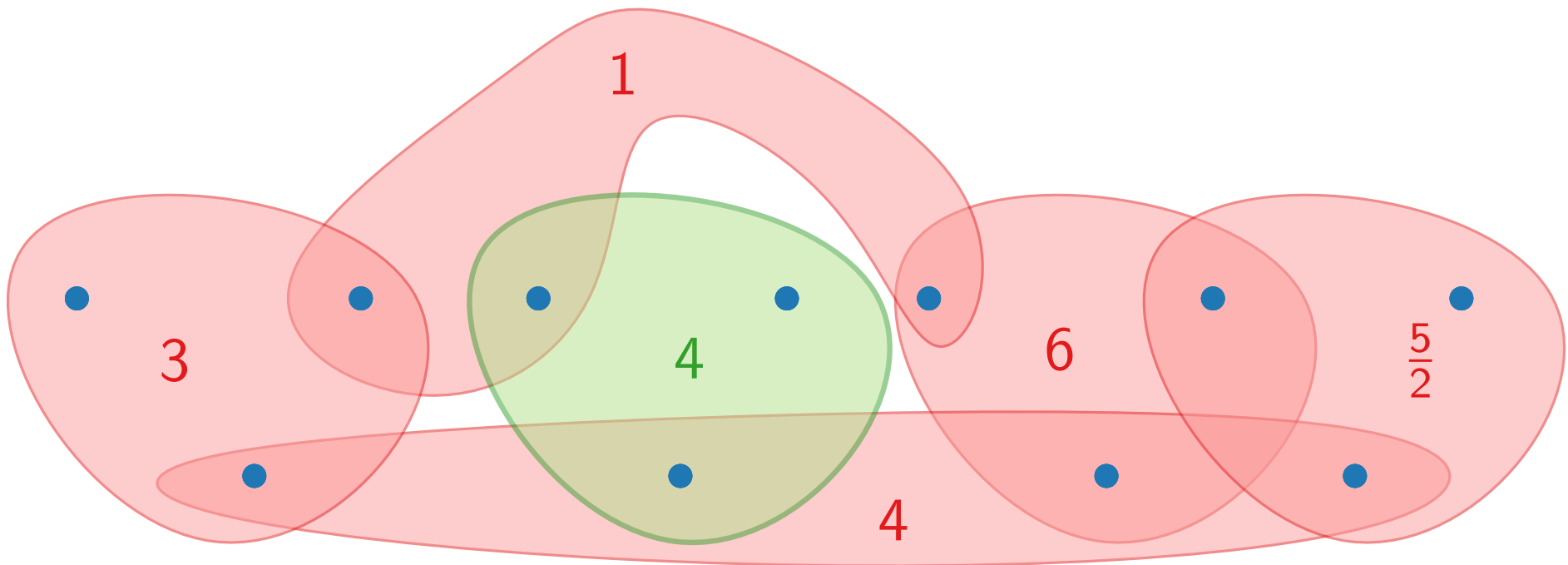
What is the real cost of picking a set?



“Buying” Elements Iteratively

What is the real cost of picking a set?

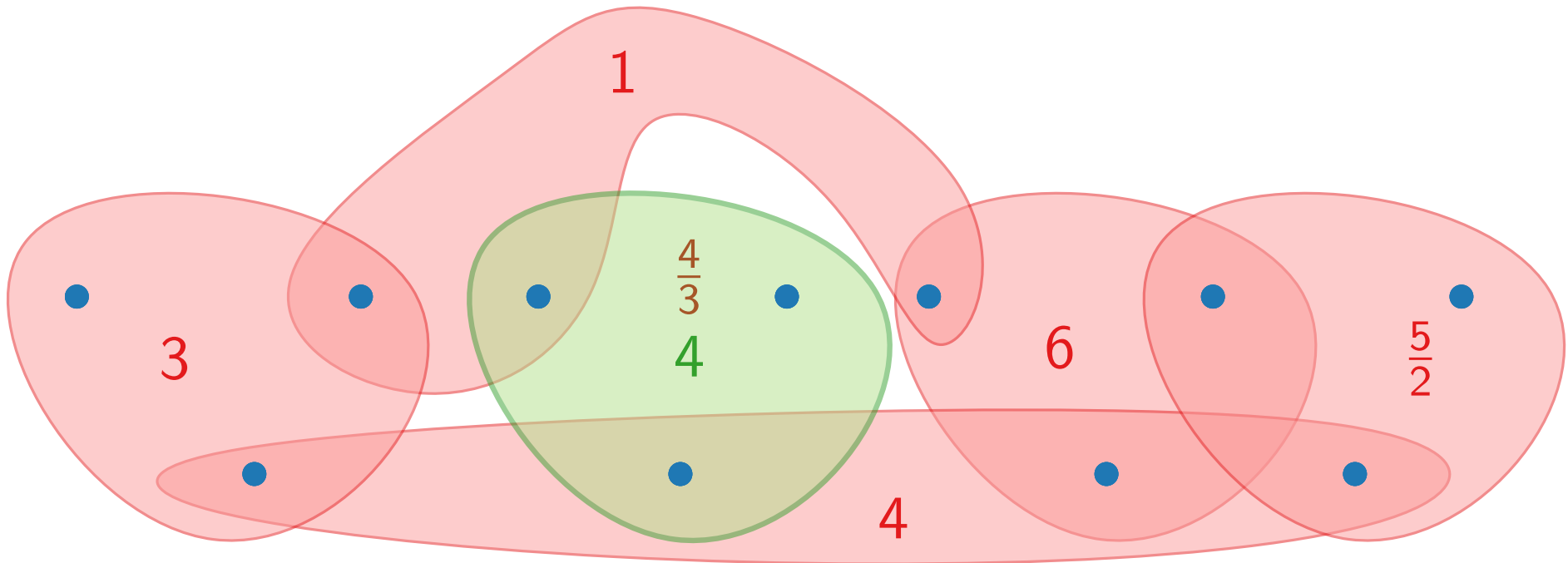
Set with k elements and cost c has per-element cost c/k .



“Buying” Elements Iteratively

What is the real cost of picking a set?

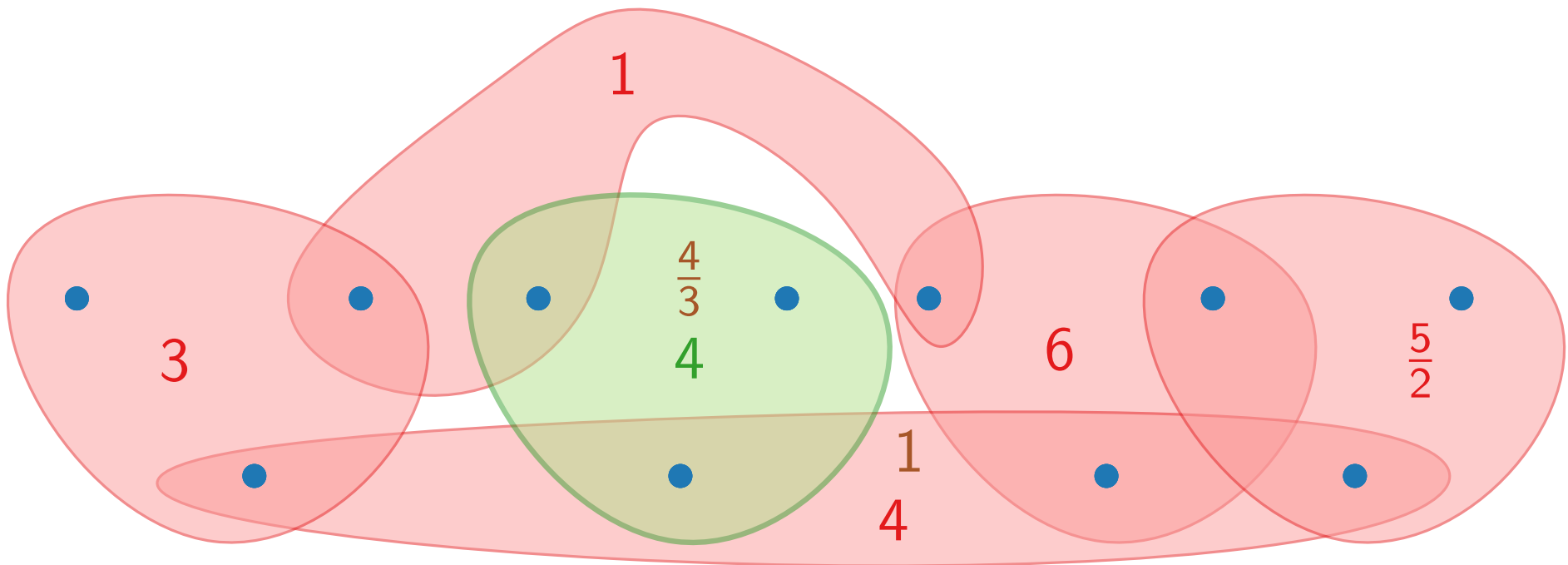
Set with k elements and cost c has per-element cost c/k .



“Buying” Elements Iteratively

What is the real cost of picking a set?

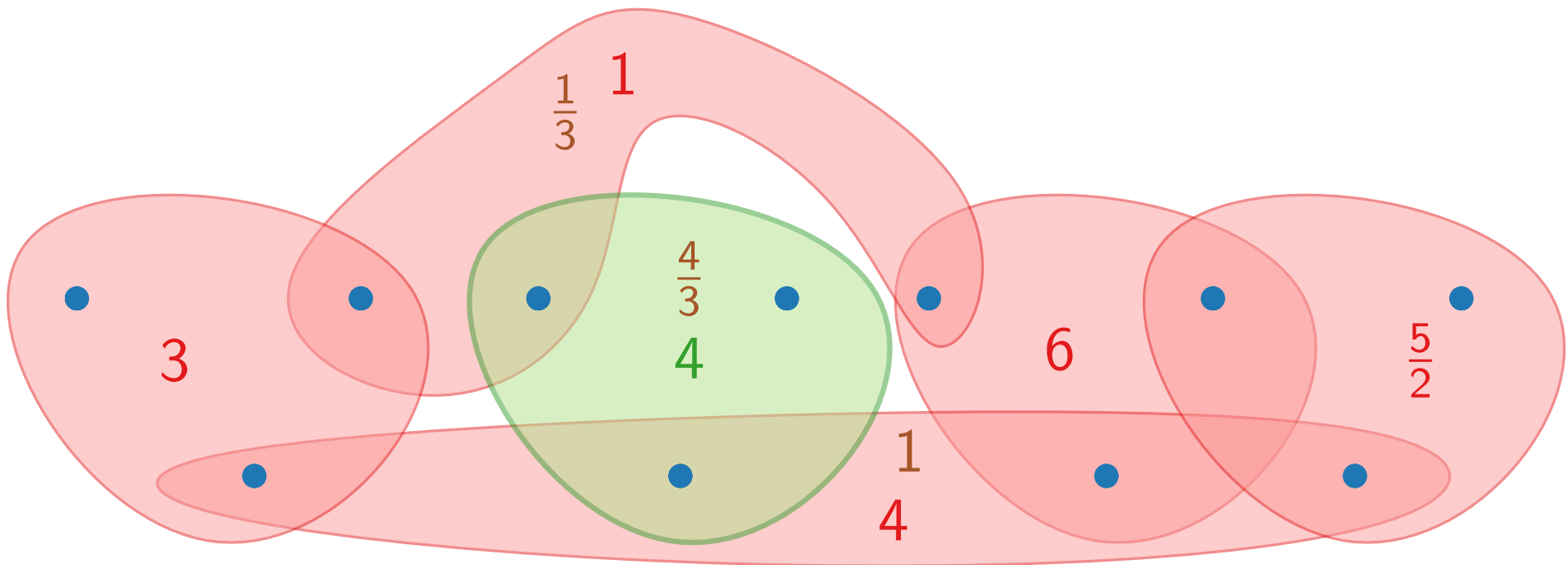
Set with k elements and cost c has per-element cost c/k .



“Buying” Elements Iteratively

What is the real cost of picking a set?

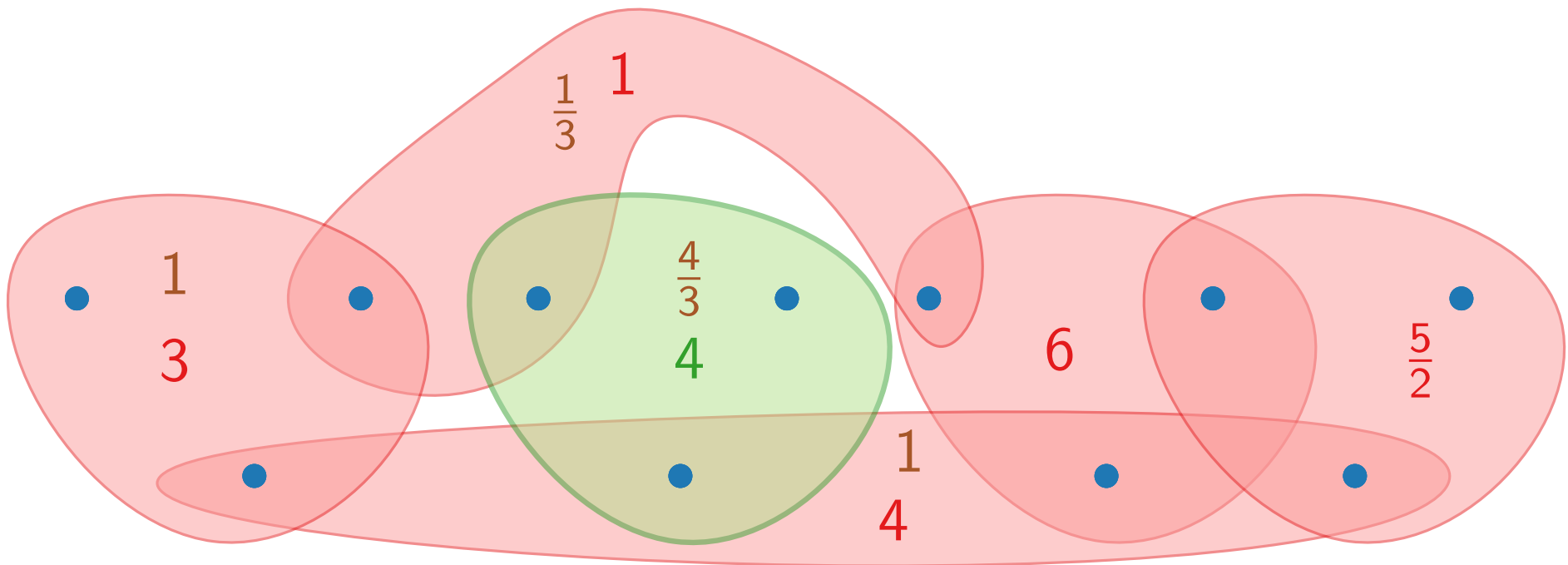
Set with k elements and cost c has per-element cost c/k .



“Buying” Elements Iteratively

What is the real cost of picking a set?

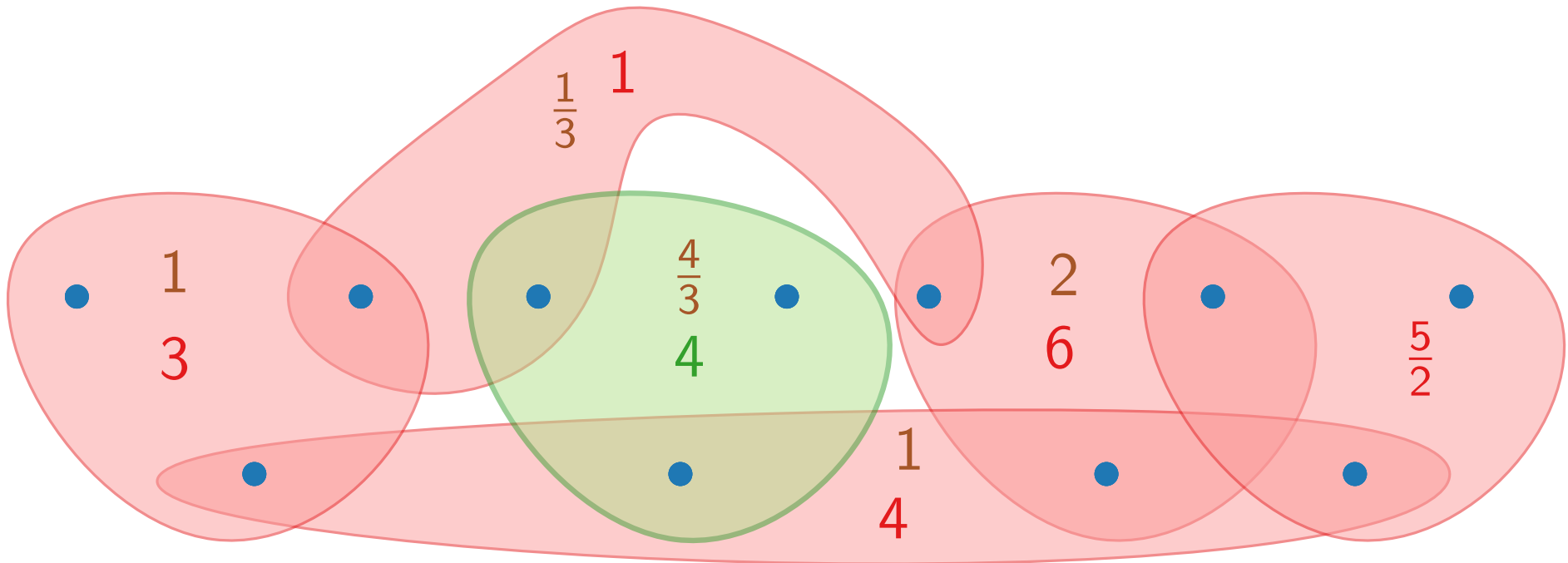
Set with k elements and cost c has per-element cost c/k .



“Buying” Elements Iteratively

What is the real cost of picking a set?

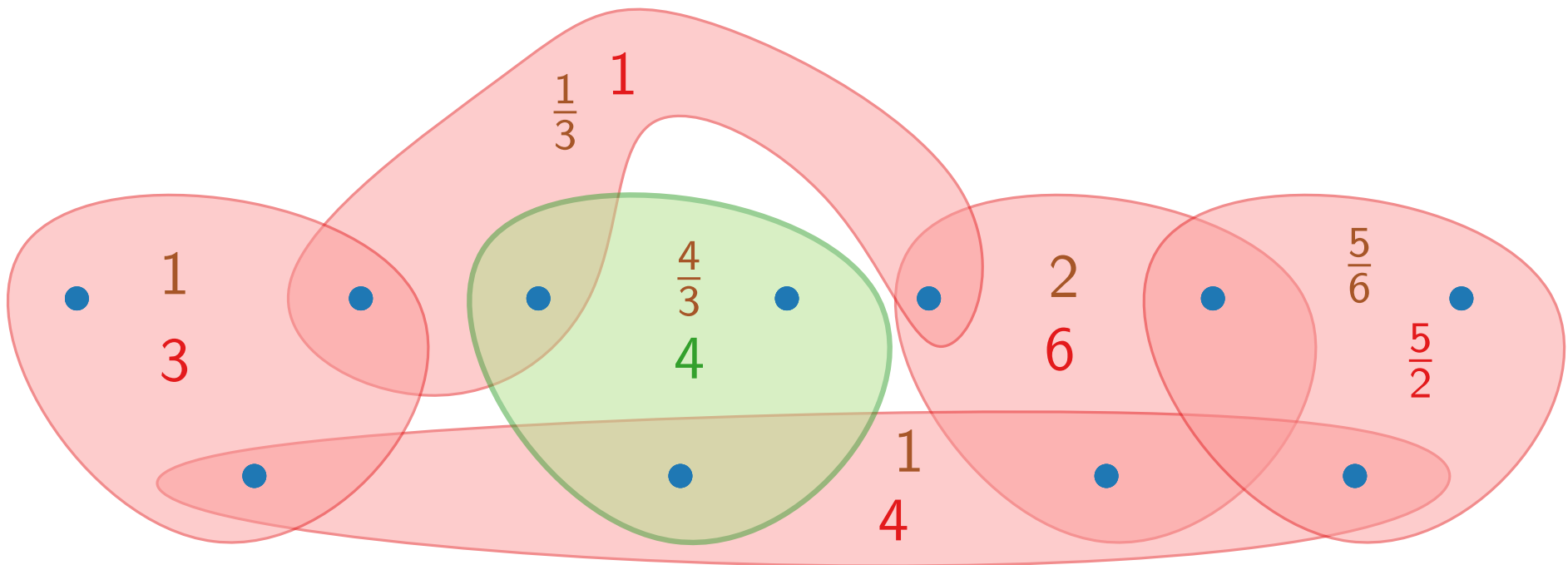
Set with k elements and cost c has per-element cost c/k .



“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 .

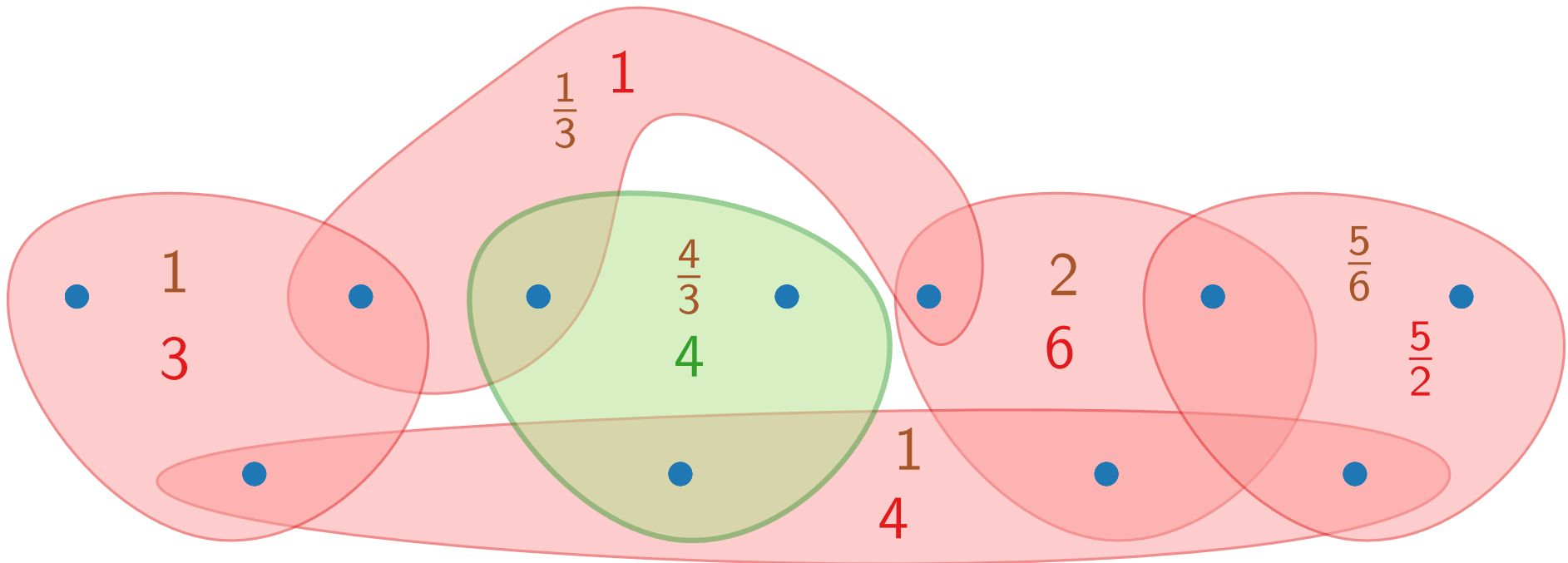


“Buying” Elements Iteratively

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?



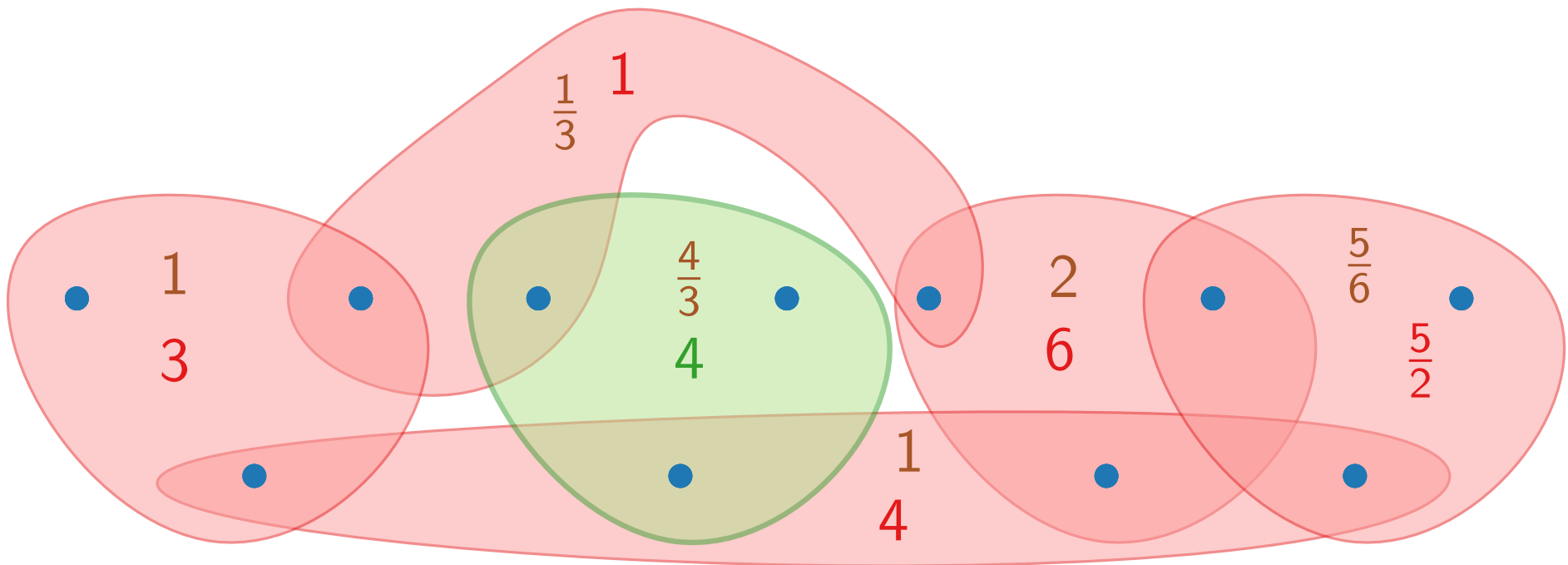
“Buying” Elements Iteratively

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.



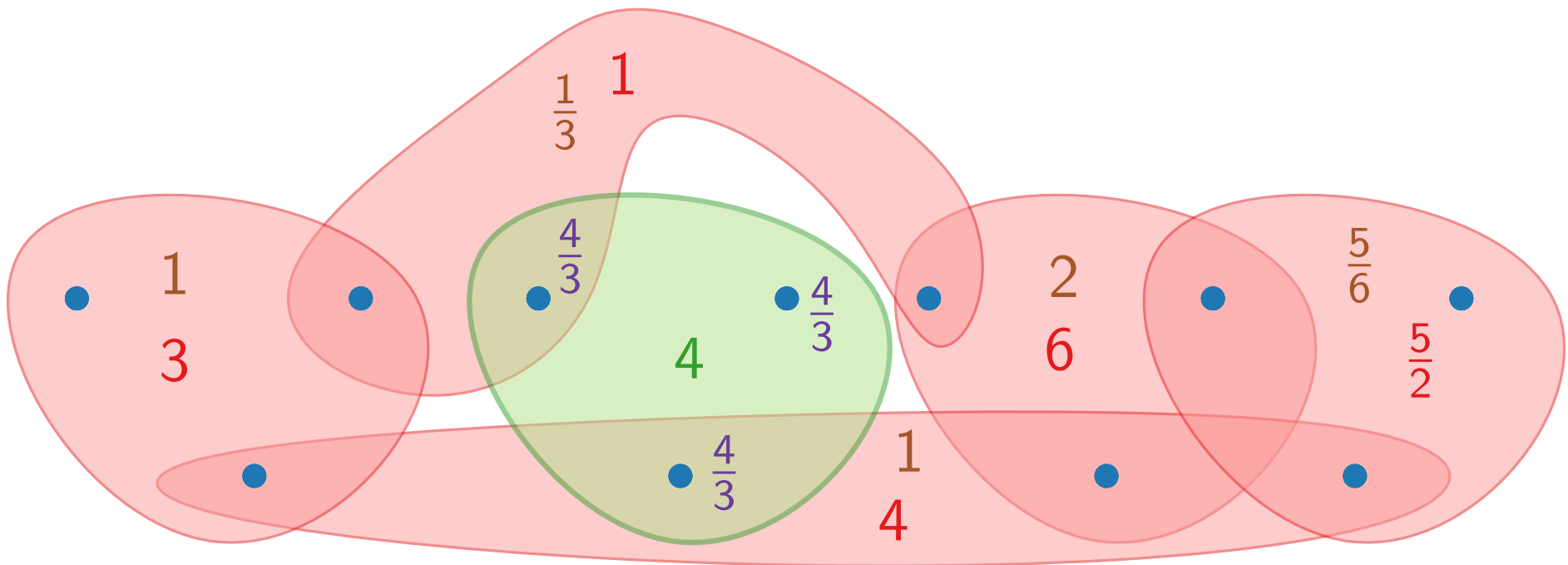
“Buying” Elements Iteratively

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.



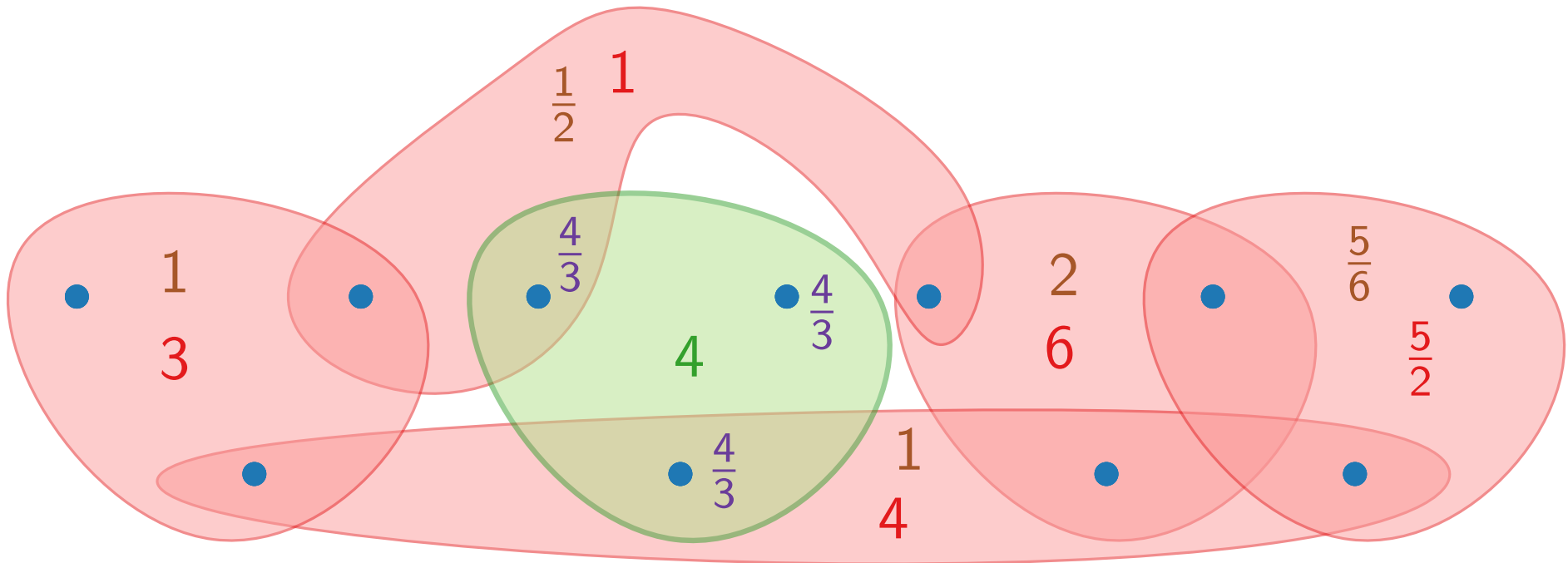
“Buying” Elements Iteratively

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.



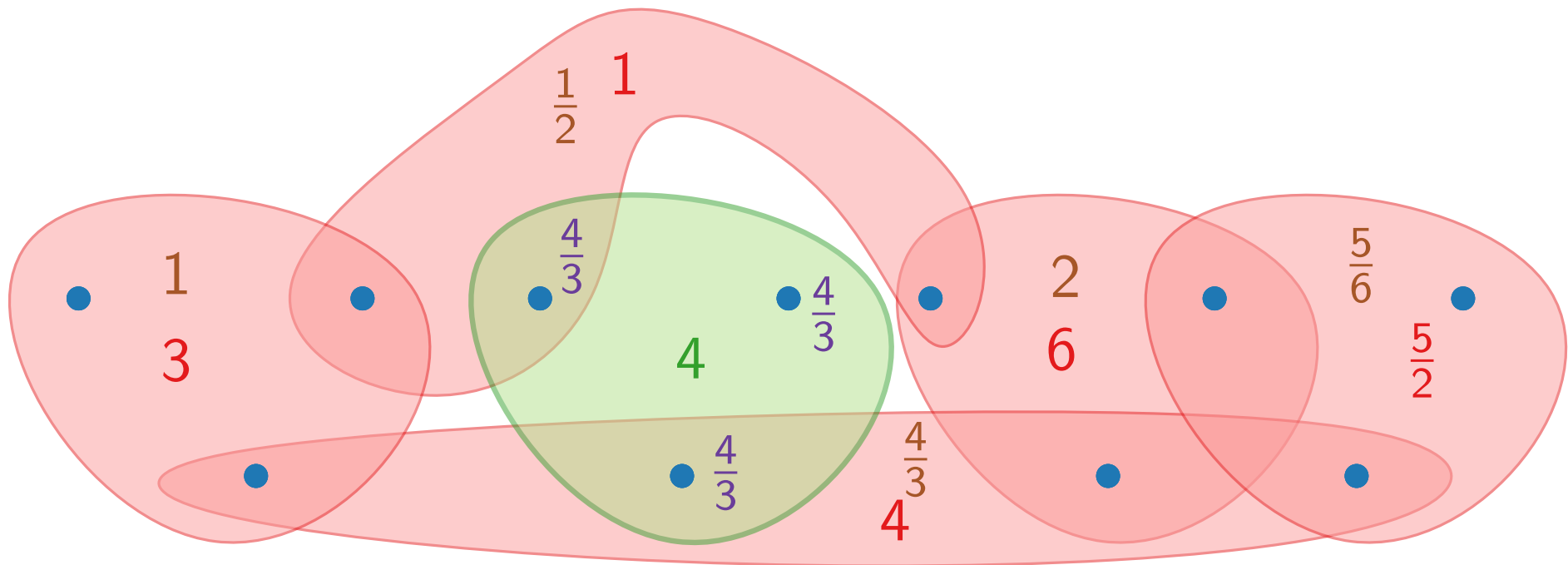
“Buying” Elements Iteratively

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.



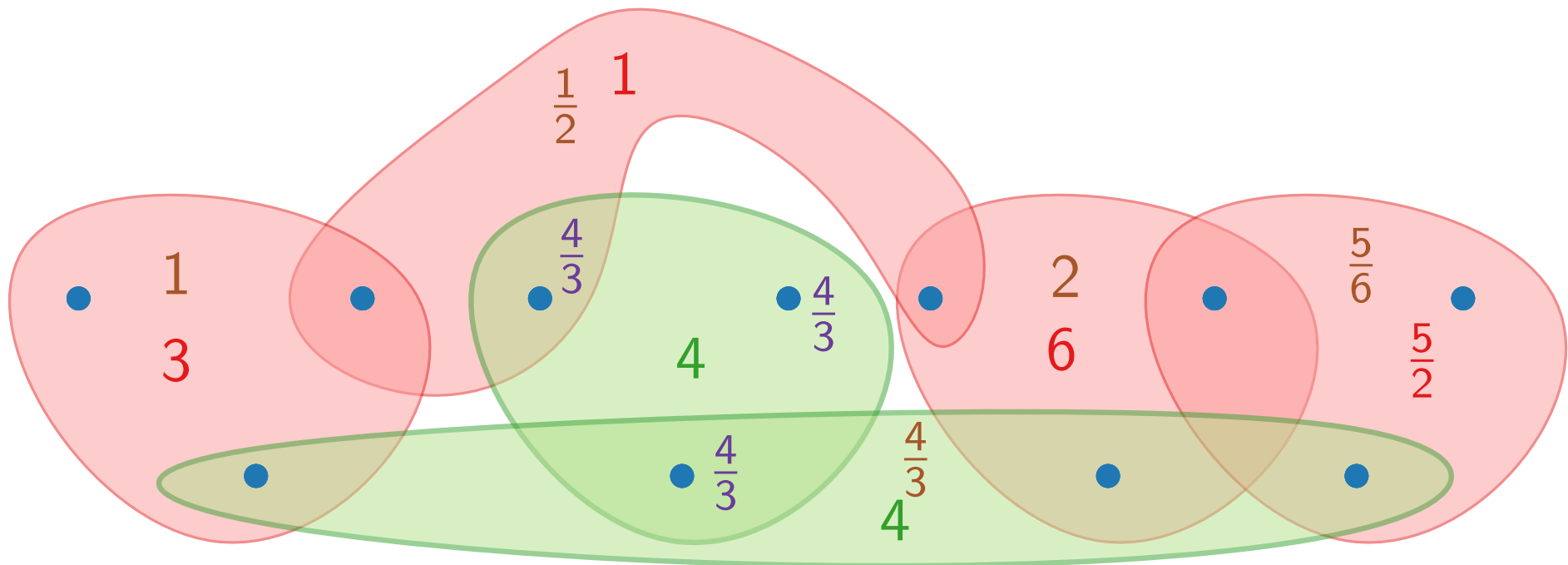
“Buying” Elements Iteratively

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.



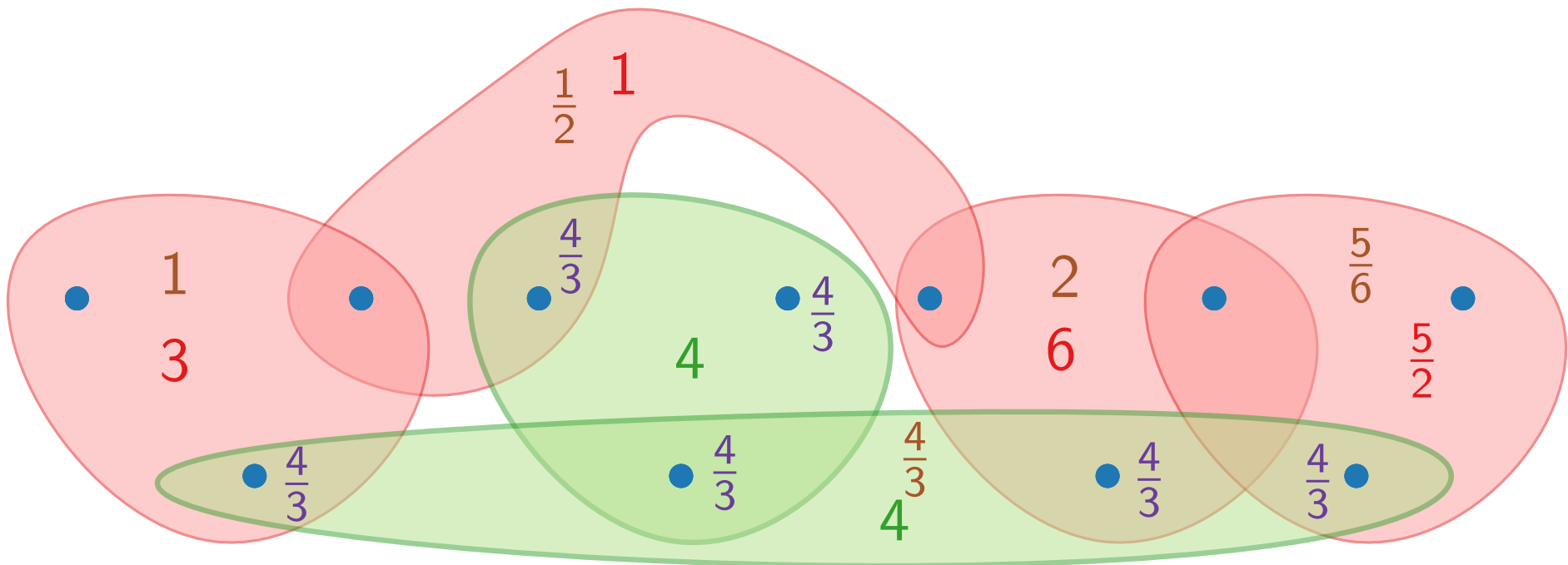
“Buying” Elements Iteratively

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.



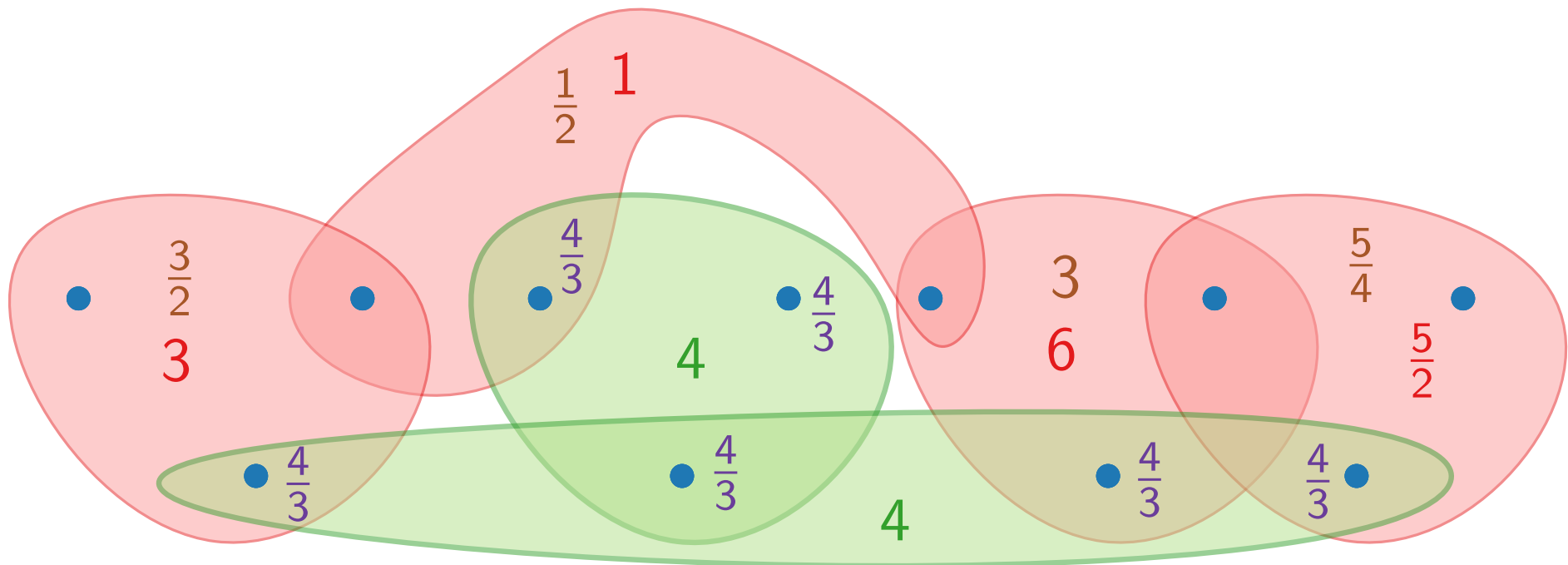
“Buying” Elements Iteratively

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.



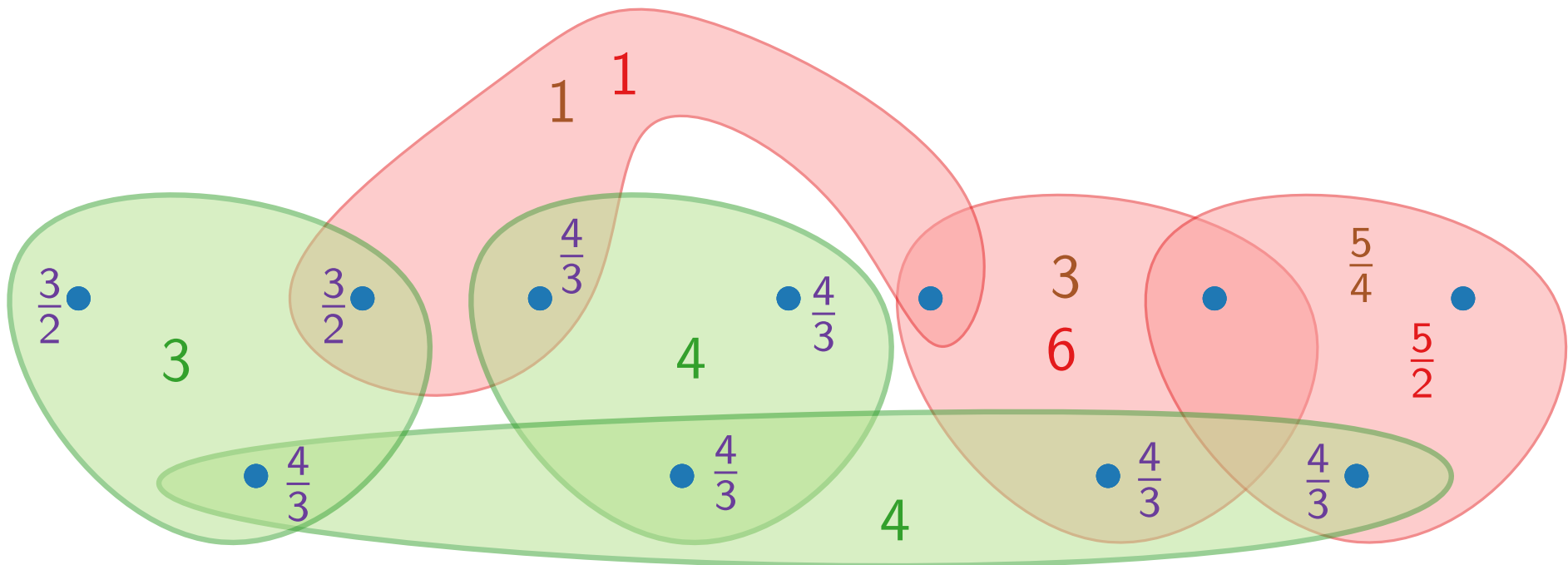
“Buying” Elements Iteratively

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.



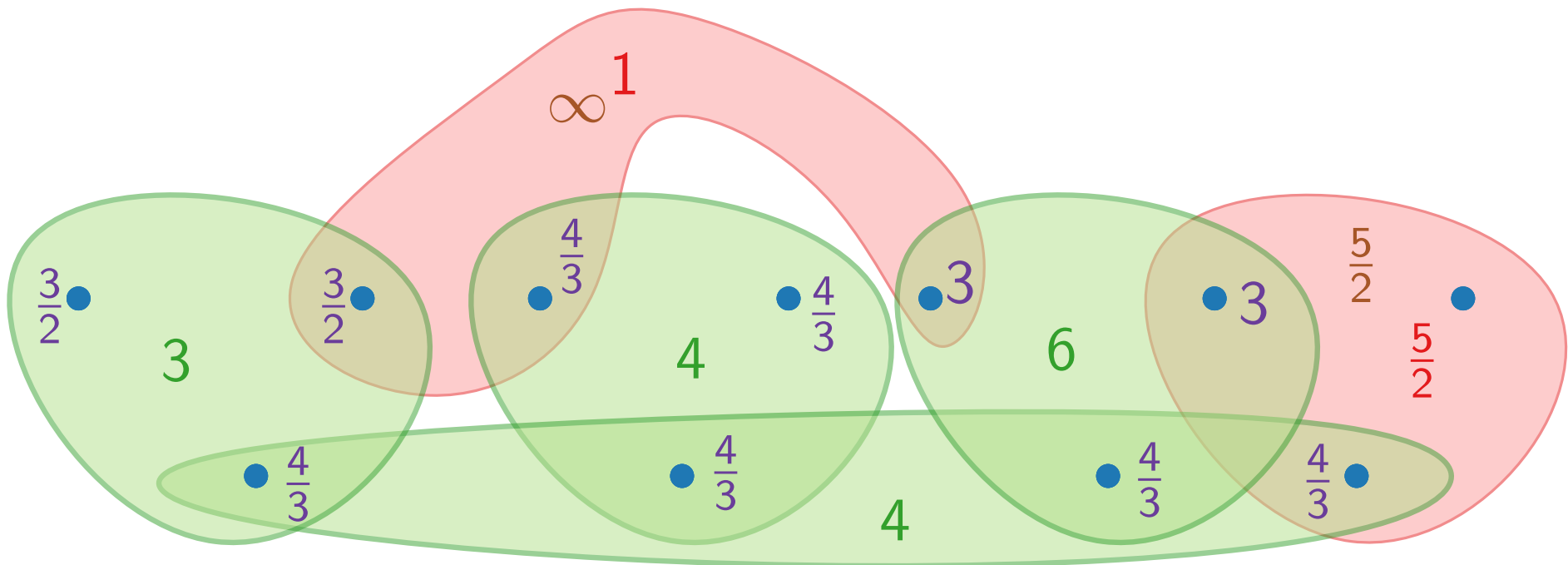
“Buying” Elements Iteratively

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**.



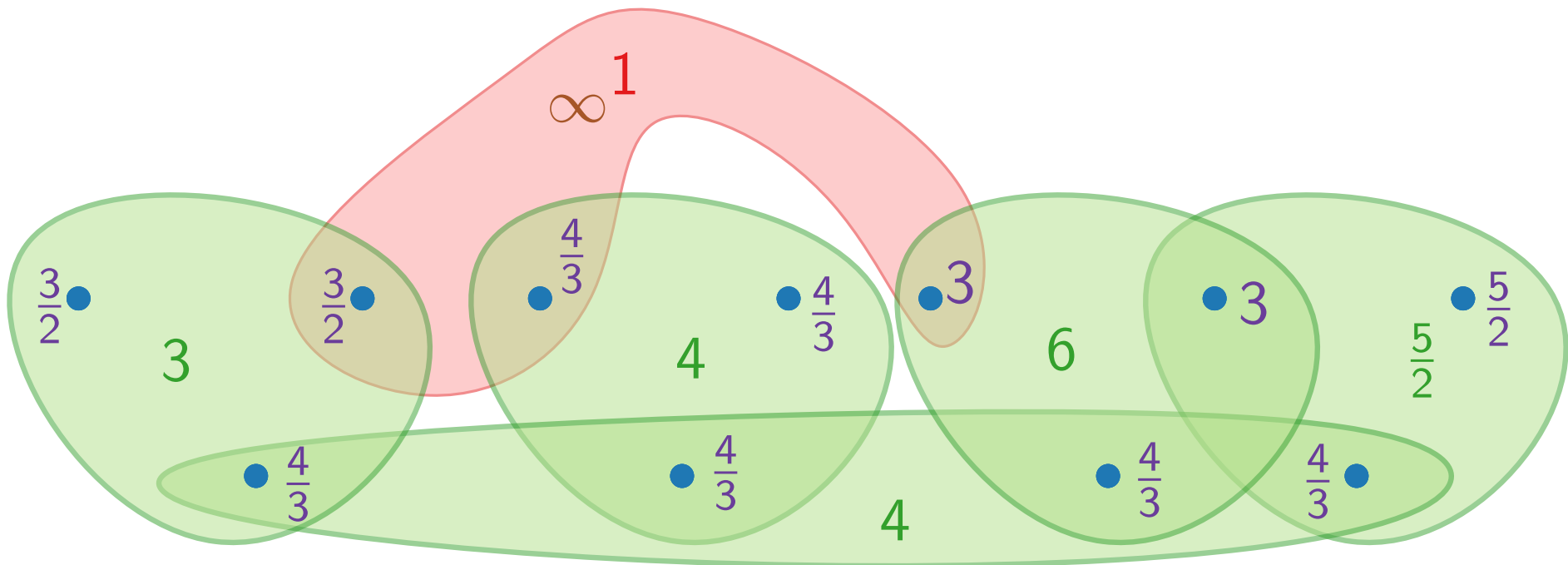
“Buying” Elements Iteratively

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

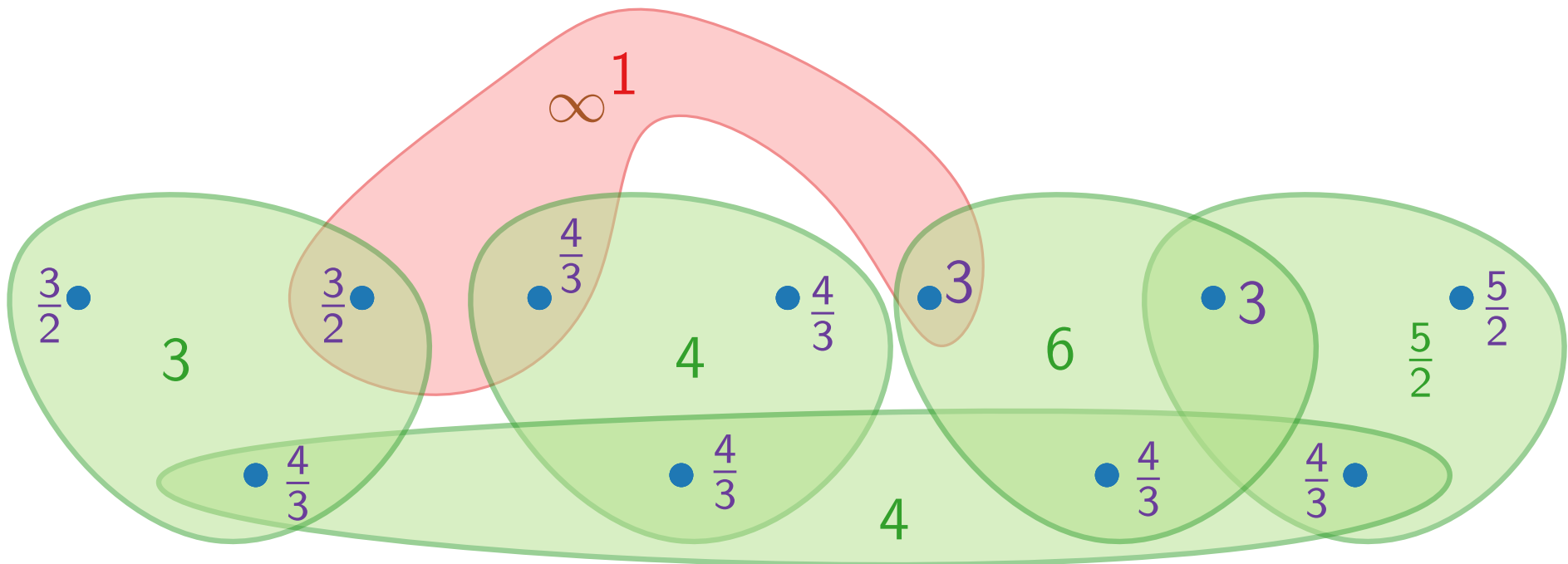
What is the real cost of picking a set?

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?

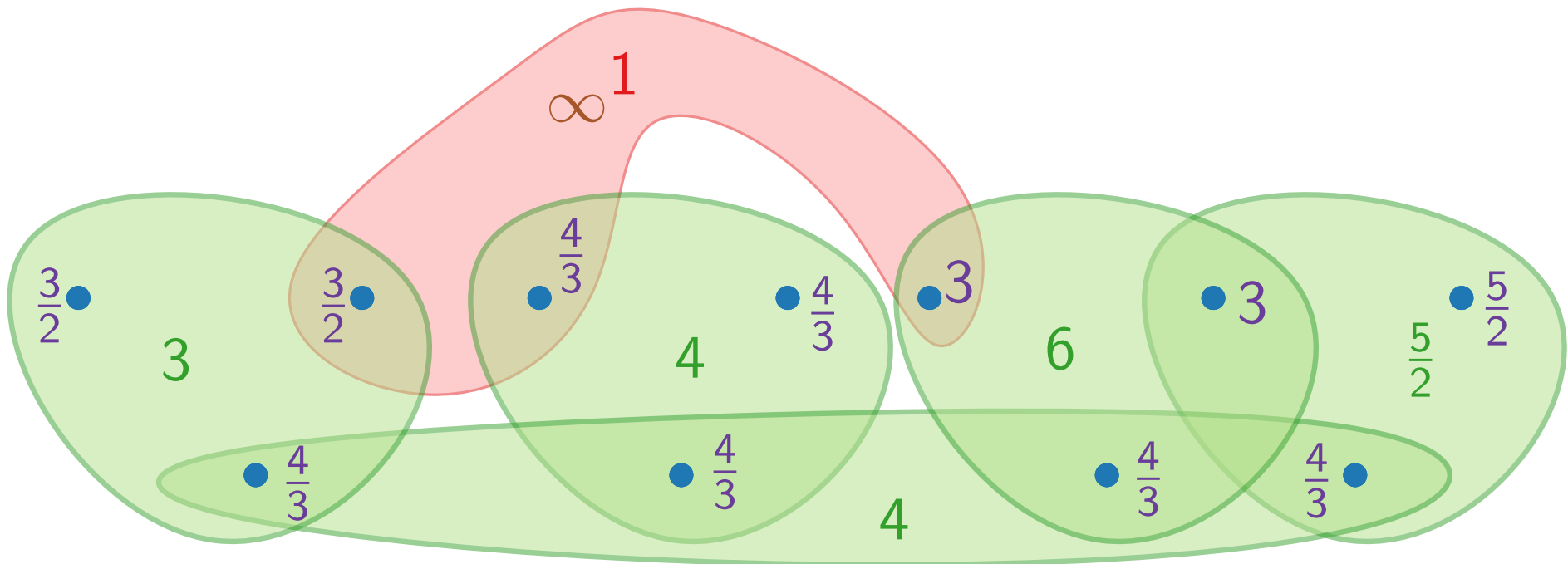
Set with k elements and cost c has **per-element cost** c/k .

What happens if we “buy” a set?

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

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

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



Greedy for SETCOVER

GreedySetCover(U , S , c)

$C \leftarrow \emptyset$

$S' \leftarrow \emptyset$

return S'

// Cover of U

Greedy for SETCOVER

GreedySetCover(U , S , c)

$C \leftarrow \emptyset$

$S' \leftarrow \emptyset$

while $C \neq U$ **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**

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

foreach $u \in \mathcal{S} \setminus C$ **do**

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

return \mathcal{S}'

// Cover of U

Greedy for SETCOVER

GreedySetCover(U , S , c)

$C \leftarrow \emptyset$

$S' \leftarrow \emptyset$

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

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

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

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

$C \leftarrow C \cup S$

return S'

// Cover of U

Greedy for SETCOVER

GreedySetCover(U, S, c)

$C \leftarrow \emptyset$

$S' \leftarrow \emptyset$

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

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

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

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

$C \leftarrow C \cup S$

$S' \leftarrow S' \cup \{S\}$

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 SETCOVER, where k is the cardinality of the largest set in \mathcal{S} and

$$\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \rightarrow 0.5 + \ln k.$$

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 $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \rightarrow 0.5 + \ln k$.

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

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 $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \rightarrow 0.5 + \ln k$.

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

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 $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \rightarrow 0.5 + \ln k$.

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

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 $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \rightarrow 0.5 + \ln k$.

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

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 $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \rightarrow 0.5 + \ln k$.

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

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 $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \rightarrow 0.5 + \ln k$.

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

Proof. Consider the iteration when the algorithm buys u_j :

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

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 $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \rightarrow 0.5 + \ln k$.

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

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.

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 $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \rightarrow 0.5 + \ln k$.

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

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 $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \rightarrow 0.5 + \ln k$.

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

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 $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \rightarrow 0.5 + \ln k$.

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

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 $c(S)/(\ell - j + 1)$
- Price by alg. no larger due to greedy choice.

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 $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \rightarrow 0.5 + \ln k$.

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

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

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 $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \rightarrow 0.5 + \ln k$.

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

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

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 $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \rightarrow 0.5 + \ln k$.

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

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 SETCOVER, where k is the cardinality of the largest set in \mathcal{S} and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \rightarrow 0.5 + \ln k$.

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

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

Proof. Let $\{S_1, \dots, S_m\}$ 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 $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \rightarrow 0.5 + \ln k$.

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

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

Proof. Let $\{S_1, \dots, S_m\}$ 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 $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \rightarrow 0.5 + \ln k$.

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

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

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

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 $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \rightarrow 0.5 + \ln k$.

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

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

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

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 $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \rightarrow 0.5 + \ln k$.

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

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

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

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 $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \rightarrow 0.5 + \ln k$.

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

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

Proof. Let $\{S_1, \dots, S_m\}$ be an opt. sol. $\text{OPT} = \sum_{i=1}^m c(S_i)$.
 $\text{ALG} = \sum_{u \in U} \text{price}(u) \stackrel{!!}{\leq} \sum_{i=1}^m \text{price}(S_i)$ (since $U = \bigcup_i S_i$)

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 $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \rightarrow 0.5 + \ln k$.

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

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

Proof. Let $\{S_1, \dots, S_m\}$ be an opt. sol. $\text{OPT} = \sum_{i=1}^m c(S_i)$.
 $\text{ALG} = \sum_{u \in U} \text{price}(u) \stackrel{!!}{\leq} \sum_{i=1}^m \text{price}(S_i)$ (since $U = \bigcup_i S_i$)
 $\leq \sum_{i=1}^m c(S_i) \cdot \mathcal{H}_{|S_i|} \leq$

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 $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \rightarrow 0.5 + \ln k$.

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

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

Proof. Let $\{S_1, \dots, S_m\}$ be an opt. sol. $\text{OPT} = \sum_{i=1}^m c(S_i)$.
 $\text{ALG} = \sum_{u \in U} \text{price}(u) \stackrel{!!}{\leq} \sum_{i=1}^m \text{price}(S_i)$ (since $U = \bigcup_i S_i$)
 $\leq \sum_{i=1}^m c(S_i) \cdot \mathcal{H}_{|S_i|} \leq \text{OPT} \cdot \mathcal{H}_k$ □

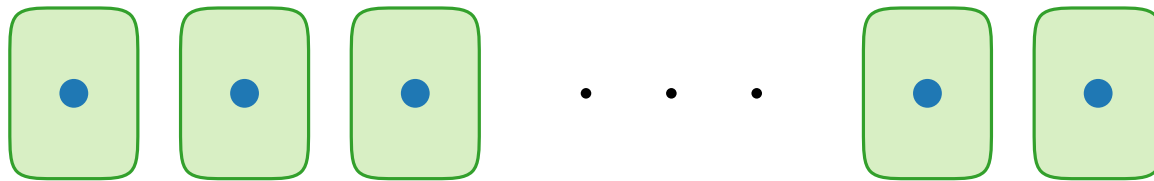
Analysis tight?

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

$$\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \leq 1 + \ln k.$$

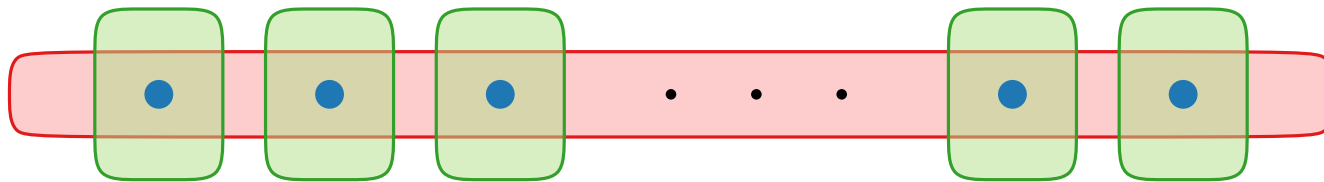
Analysis tight?

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

$$\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \leq 1 + \ln k.$$


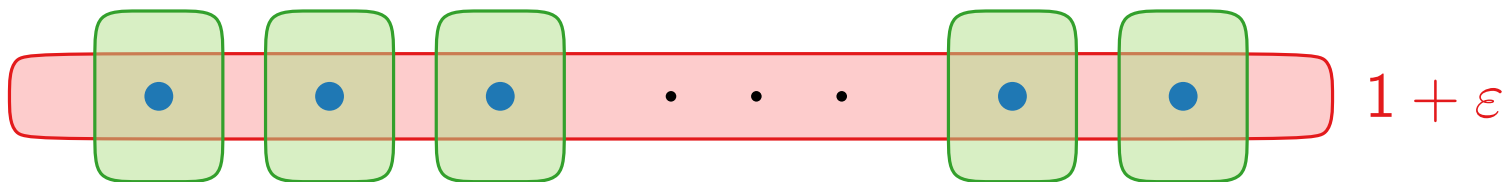
Analysis tight?

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

$$\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \leq 1 + \ln k.$$


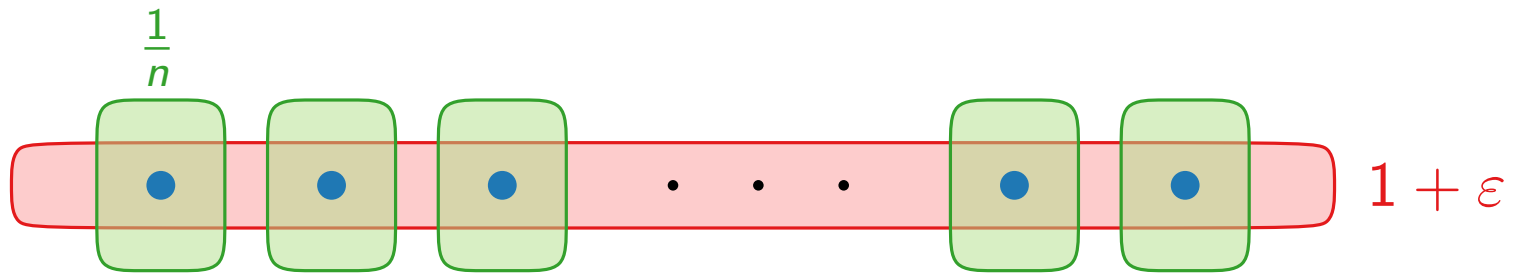
Analysis tight?

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

$$\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \leq 1 + \ln k.$$


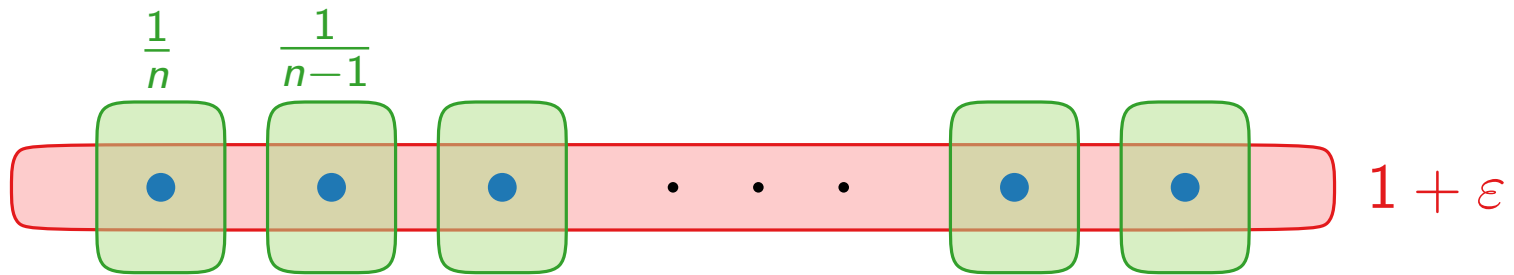
Analysis tight?

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

$$\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \leq 1 + \ln k.$$


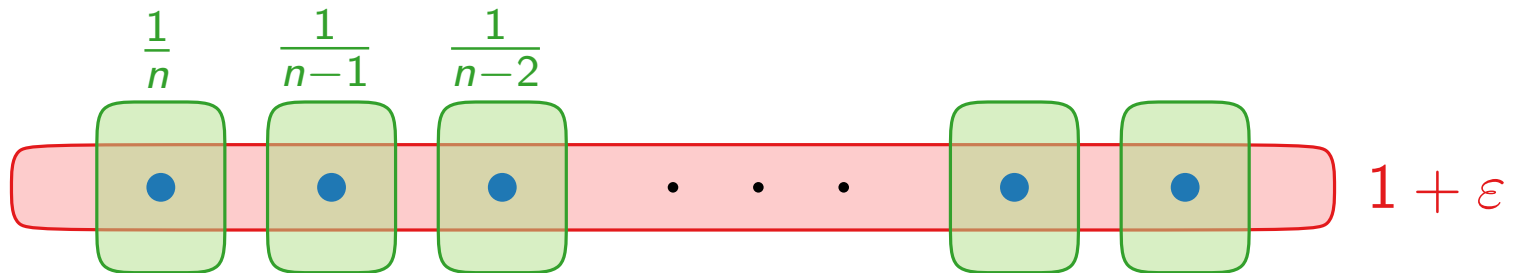
Analysis tight?

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

$$\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \leq 1 + \ln k.$$


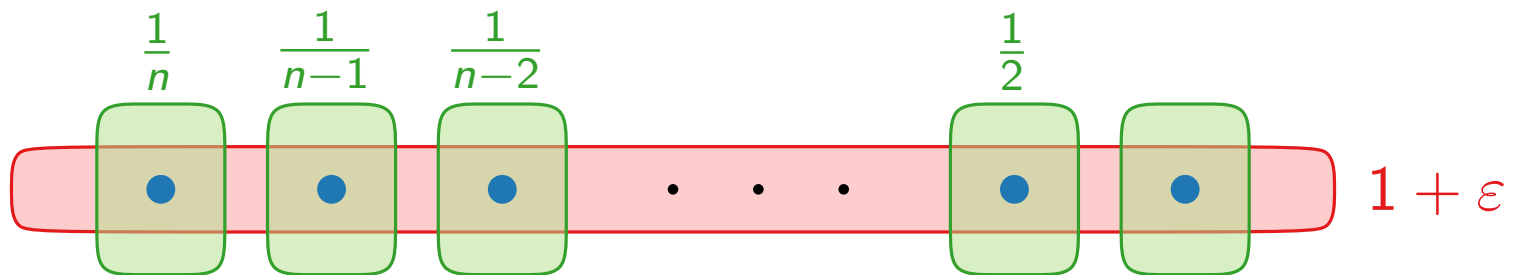
Analysis tight?

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

$$\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \leq 1 + \ln k.$$


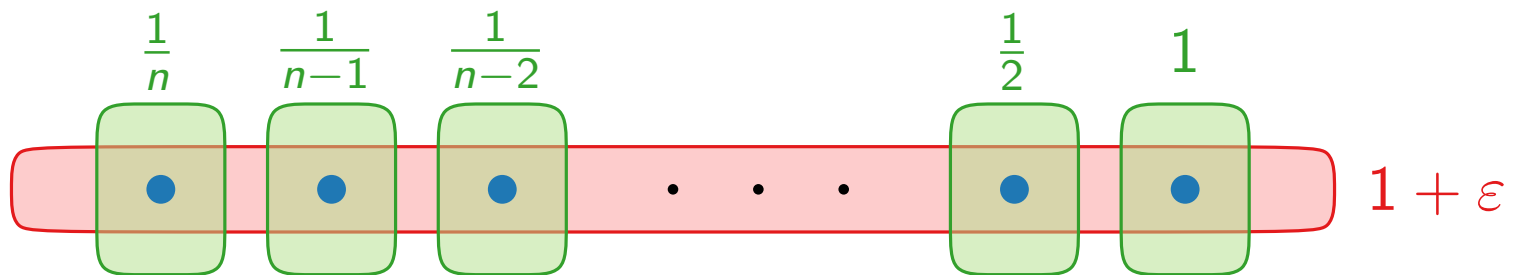
Analysis tight?

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

$$\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \leq 1 + \ln k.$$


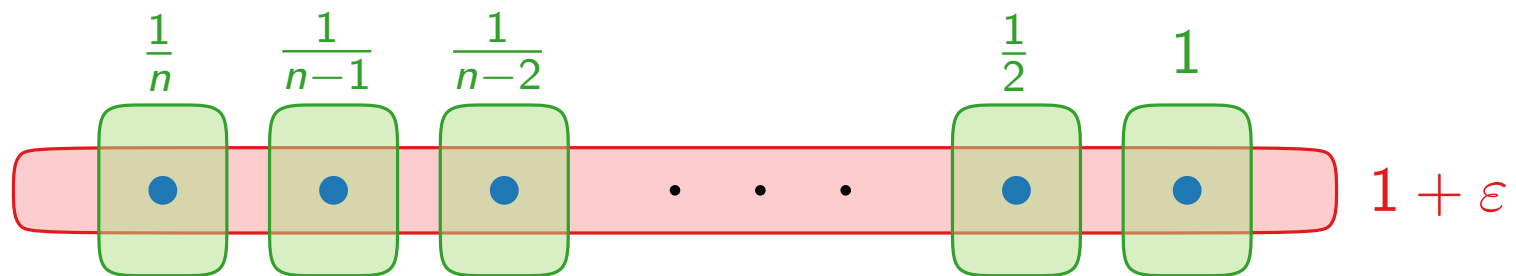
Analysis tight?

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

$$\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \leq 1 + \ln k.$$


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 $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \leq 1 + \ln k$.



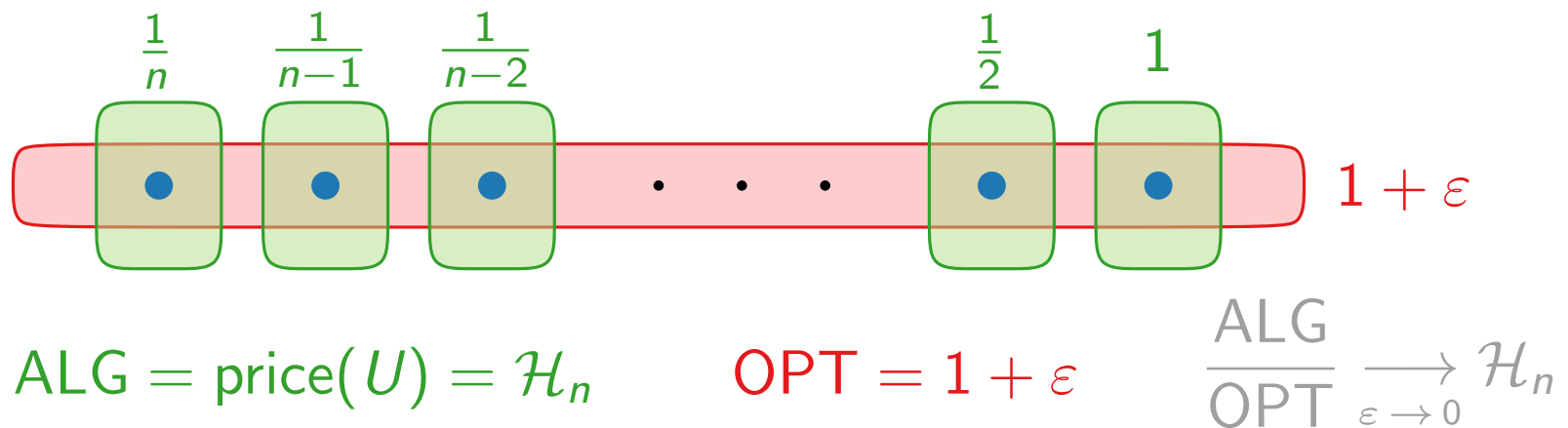
$$\text{ALG} = \text{price}(U) = \mathcal{H}_n$$

$$\text{OPT} = 1 + \varepsilon$$

$$\frac{\text{ALG}}{\text{OPT}} \xrightarrow{\varepsilon \rightarrow 0} \mathcal{H}_n$$

Analysis tight?

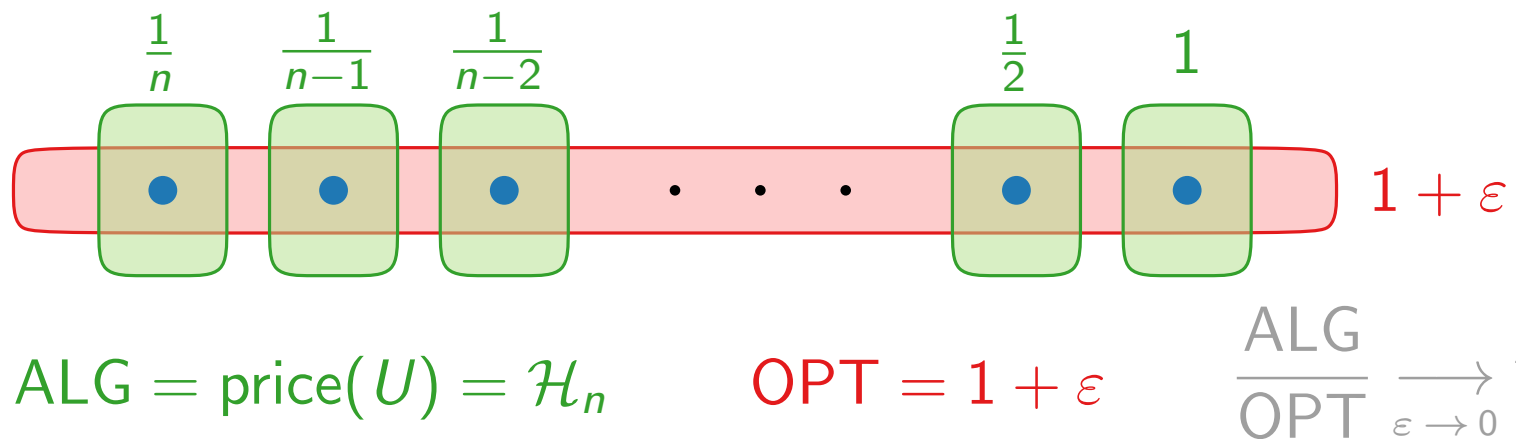
Theorem. GreedySetCover is a factor- \mathcal{H}_k approximation algorithm for SETCOVER, where k is the cardinality of the largest set in \mathcal{S} and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \leq 1 + \ln k$.



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 $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \leq 1 + \ln k$.



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?

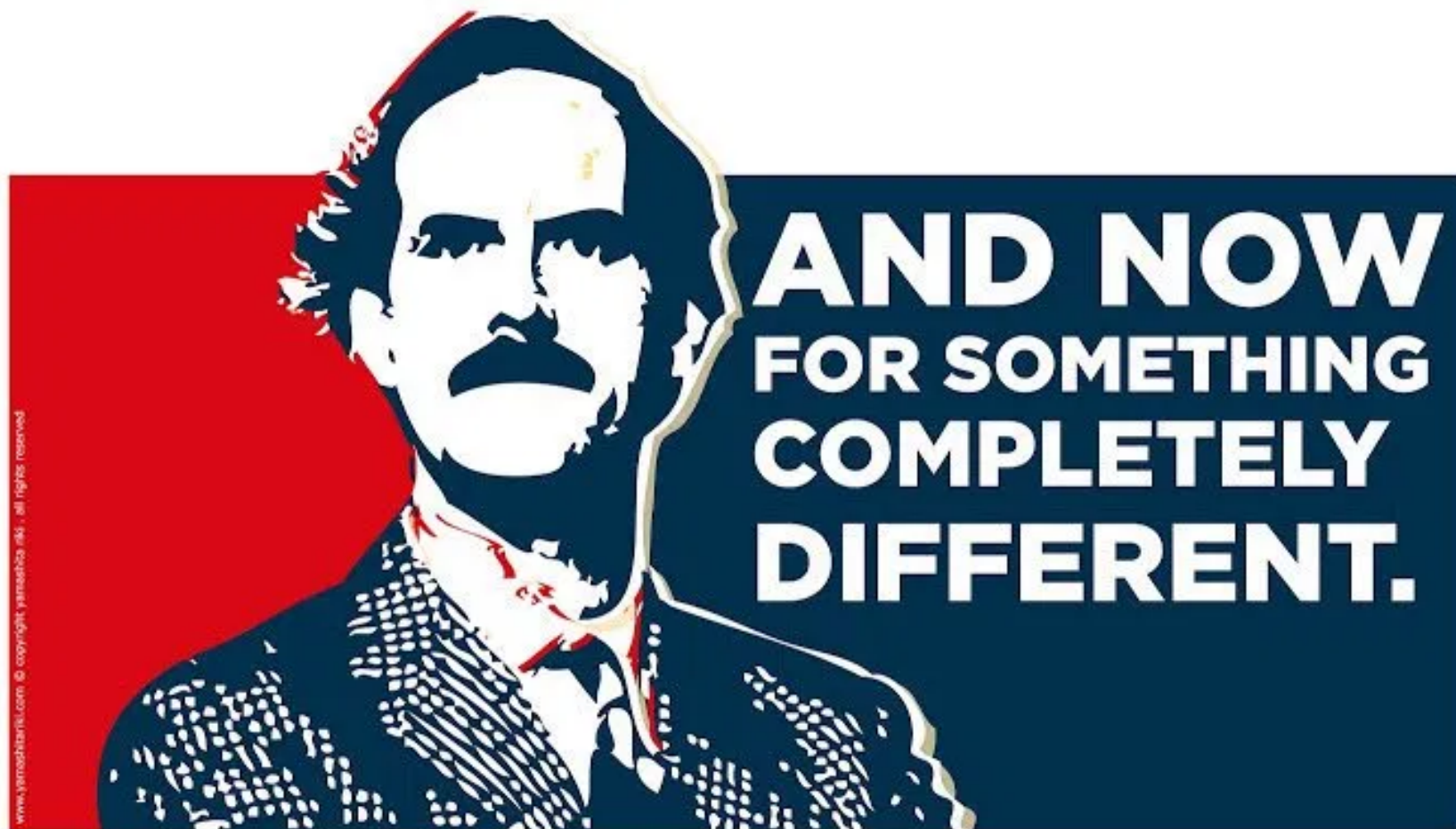
Theorem. GreedySetCover is a factor- \mathcal{H}_k approximation algorithm for SETCOVER, where k is the cardinality of the largest set in \mathcal{S} and

$$\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \leq 1 + \ln k.$$

AL

Can we do better?

No – for any $\epsilon > 0$, there is a set system with factor $(1 + \epsilon)$



Analysis tight?

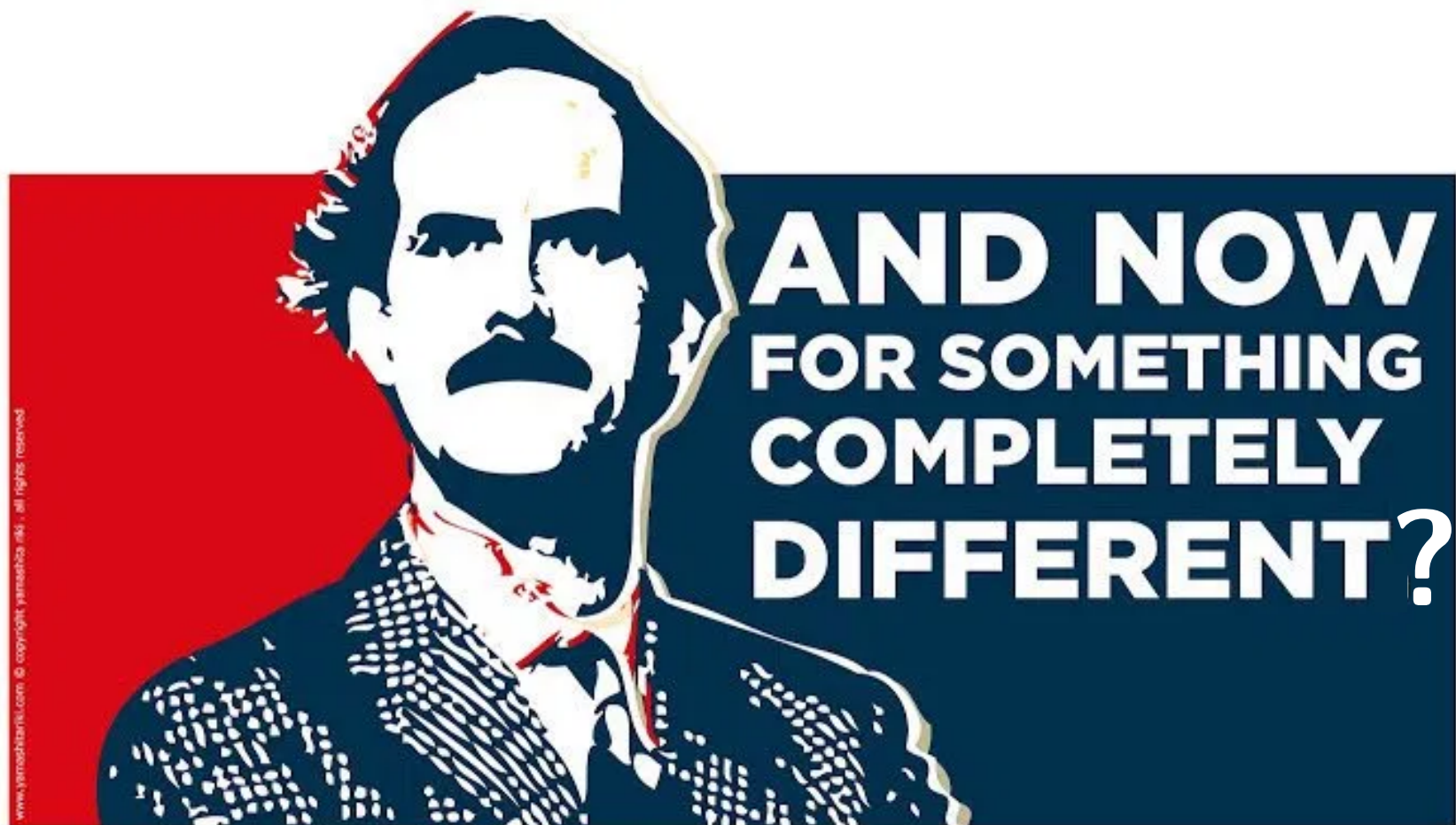
Theorem. GreedySetCover is a factor- \mathcal{H}_k approximation algorithm for SETCOVER, where k is the cardinality of the largest set in \mathcal{S} and

$$\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \leq 1 + \ln k.$$

AL

Can we do better?

No – for any $\epsilon > 0$, there is a set system with factor $(1 + \epsilon)$



Approximation Algorithms

Lecture 2:

SETCOVER and SHORTESTSUPERSTRING

Part IV:

SHORTESTSUPERSTRING

SHORTESTSUPERSTRING (SSS)

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

SHORTESTSUPERSTRING (SSS)

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

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

SHORTESTSUPERSTRING (SSS)

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

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

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

SHORTESTSUPERSTRING (SSS)

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

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

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

SHORTESTSUPERSTRING (SSS)

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

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

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

abc

SHORTESTSUPERSTRING (SSS)

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

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

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

abc
 bcb

SHORTESTSUPERSTRING (SSS)

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

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

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

abc
 bcb
 $cbaa$

SHORTESTSUPERSTRING (SSS)

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

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

Example.

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



$abcbaa$

abc

bcb

$cbaa$

SHORTESTSUPERSTRING (SSS)

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

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

Example.

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



$abcbaa$ “covers” all strings in U

abc

bcb

$cbaa$

SHORTESTSUPERSTRING (SSS)

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

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

Example.

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



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

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 σ_{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 $U := \{s_1, \dots, s_n\}$.

Let be σ_{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)

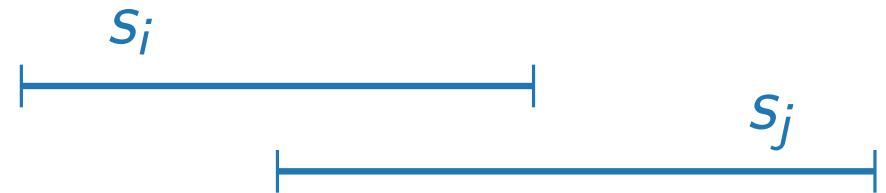


SSS as a SETCOVER Problem

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

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

Let be σ_{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)

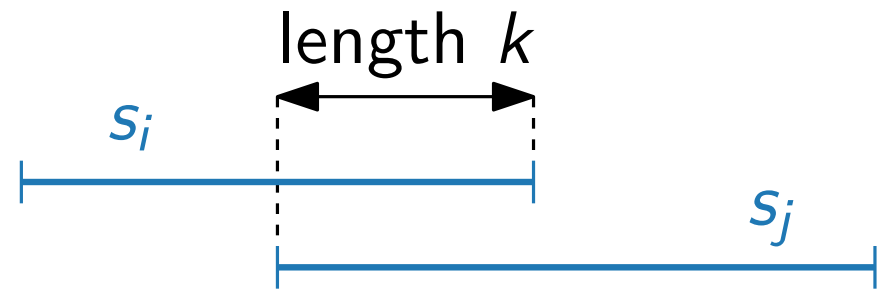


SSS as a SETCOVER Problem

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

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

Let be σ_{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)

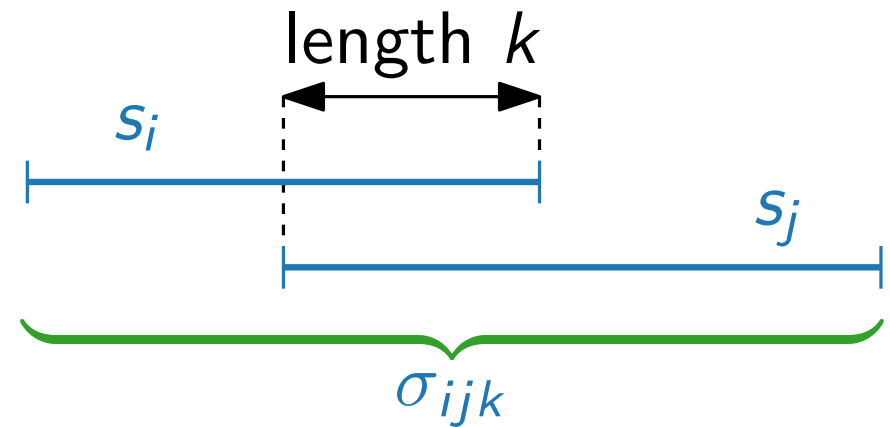


SSS as a SETCOVER Problem

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

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

Let be σ_{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)



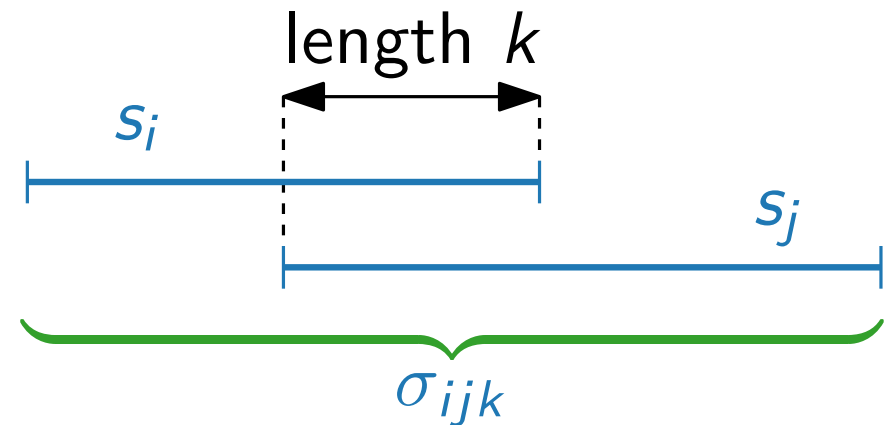
SSS as a SETCOVER Problem

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

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

Let be σ_{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



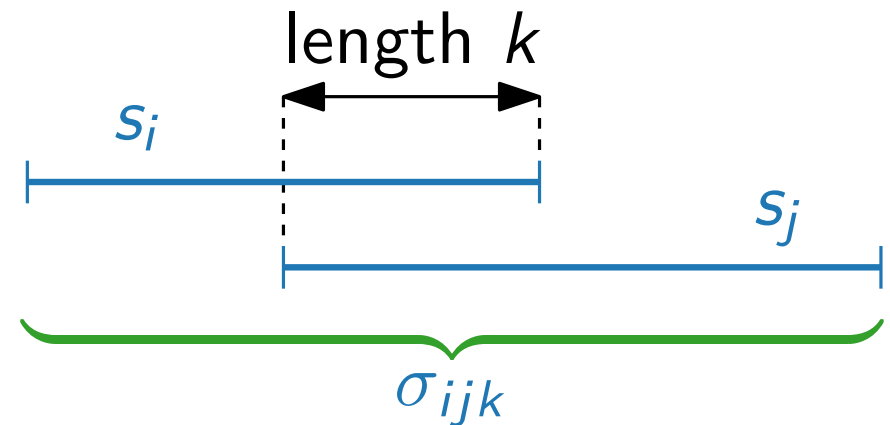
SSS as a SETCOVER Problem

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

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

Let be σ_{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

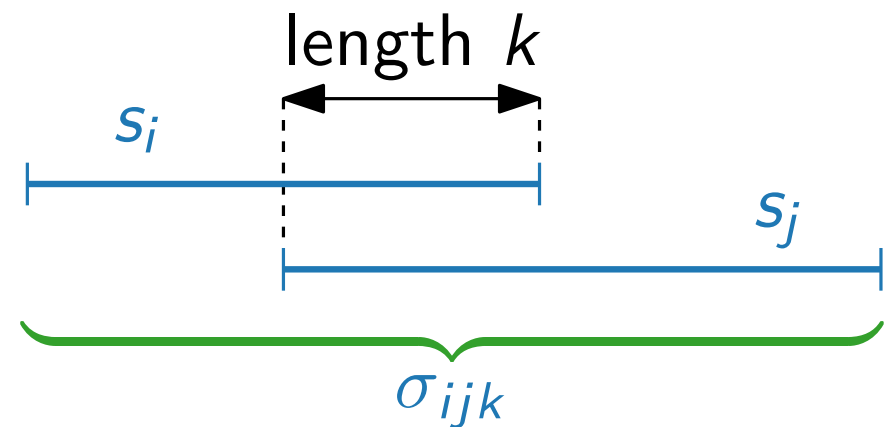
SETCOVER Instance: ground set U , set family \mathcal{S} , costs c .

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

Let be σ_{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



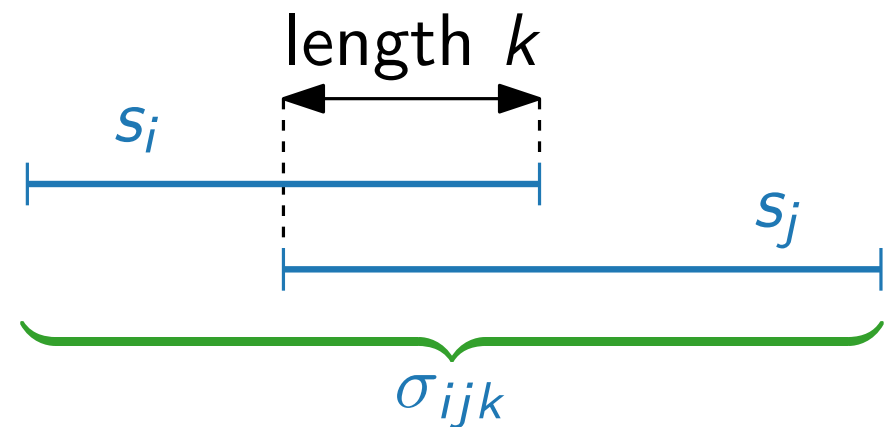
SSS as a SETCOVER Problem

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

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

Let be σ_{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
 σ_{ij2} : cabababc

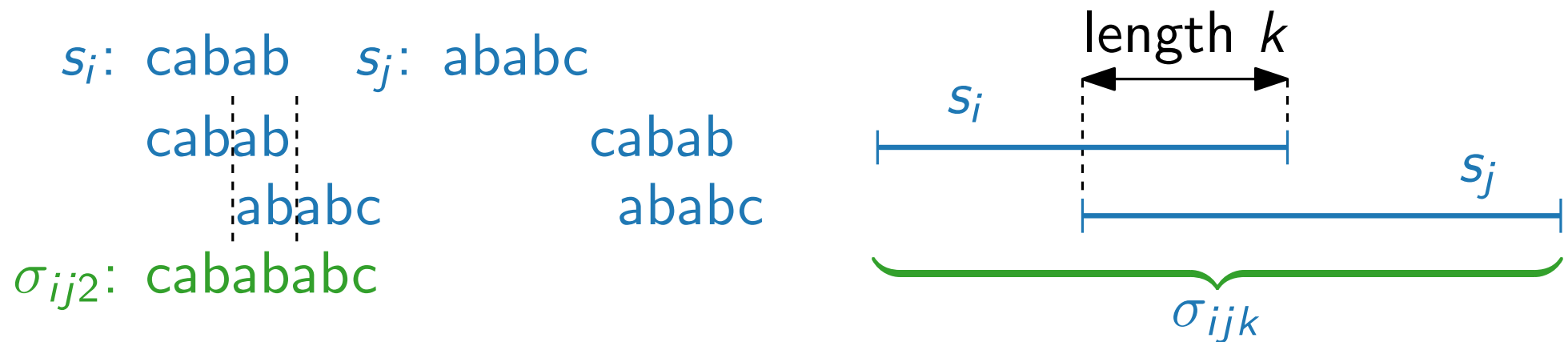


SSS as a SETCOVER Problem

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

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

Let be σ_{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)



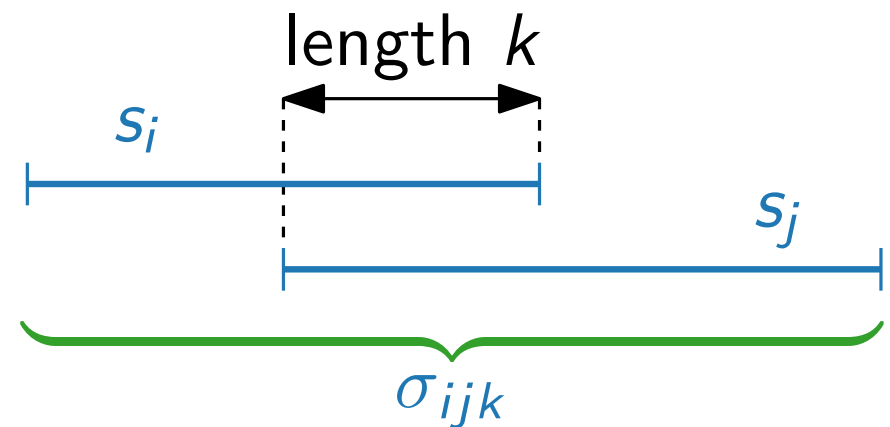
SSS as a SETCOVER Problem

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

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

Let be σ_{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
σ_{ij2} : cabababc	σ_{ij4} : cababc



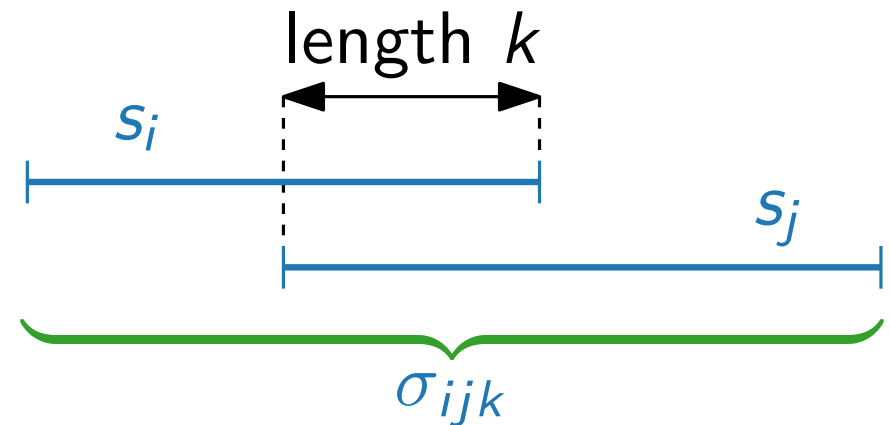
SSS as a SETCOVER Problem

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

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

Let be σ_{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
σ_{ij2} : cabababc	σ_{ij4} : cababc



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

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

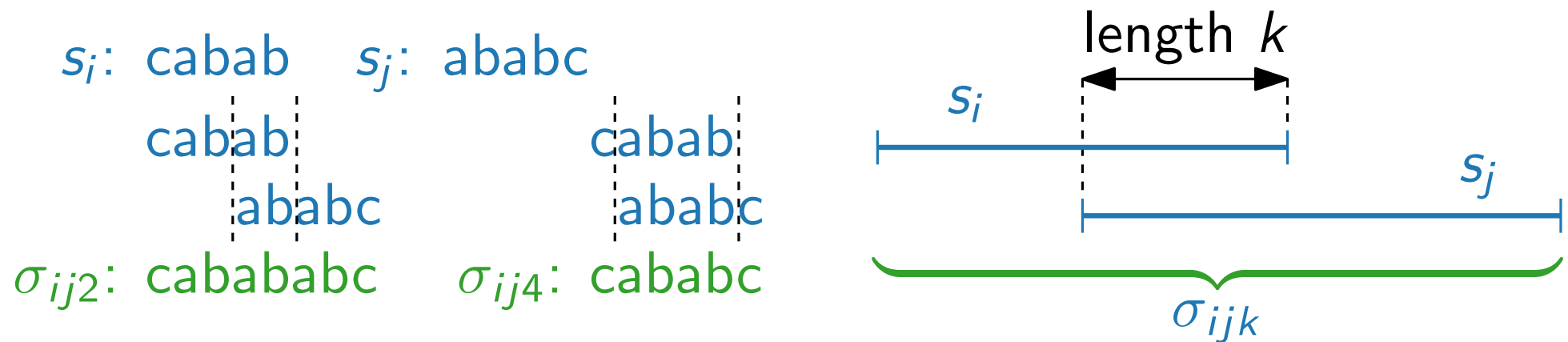
$$\mathcal{S} =$$

SSS as a SETCOVER Problem

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

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

Let be σ_{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)



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

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

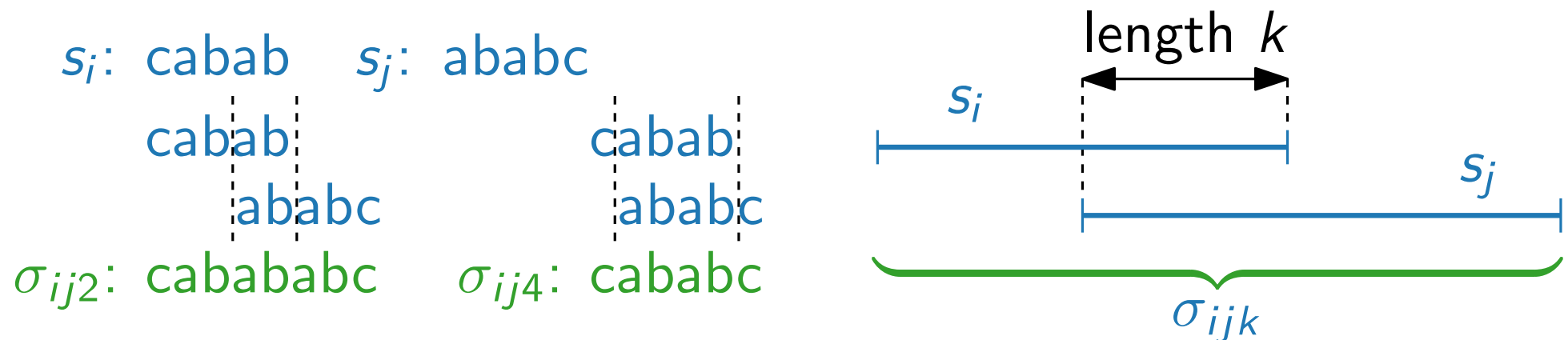
$\mathcal{S} =$

SSS as a SETCOVER Problem

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

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

Let be σ_{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)



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

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

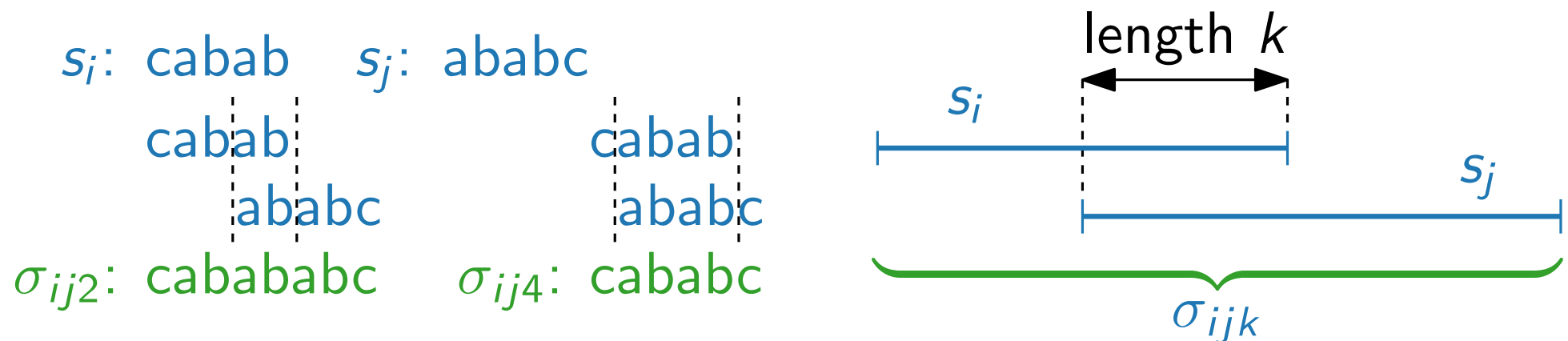
$\mathcal{S} =$

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 σ_{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)



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

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

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

Approximation Algorithms

Lecture 2:

SETCOVER and SHORTESTSUPERSTRING

Part V:

Solving SHORTESTSUPERSTRING via SETCOVER

Relating SSS and SETCOVER

Lemma. Let OPT_{SSS} be the length of a shortest superstring of U , and let OPT_{SC} be the minimum cost of the corresponding SETCOVER instance. Then

$$OPT_{SSS} \leq OPT_{SC}.$$

Relating SSS and SETCOVER

Lemma. Let OPT_{SSS} be the length of a shortest superstring of U , and let OPT_{SC} be the minimum cost of the corresponding SETCOVER instance. Then

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

Proof.

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

Relating SSS and SETCOVER

Lemma. Let OPT_{SSS} be the length of a shortest superstring of U , and let OPT_{SC} be the minimum cost of the corresponding SETCOVER instance. Then

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

Proof.

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

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

Relating SSS and SETCOVER

Lemma. Let OPT_{SSS} be the length of a shortest superstring of U , and let OPT_{SC} be the minimum cost of the corresponding SETCOVER instance. Then

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

Proof.

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

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

$$\sum_{i=1}^k |\pi_i| = \sum_{i=1}^k c(S(\pi_i)) = \text{OPT}_{\text{SC}}.$$

Relating SSS and SETCOVER

Lemma. Let OPT_{SSS} be the length of a shortest superstring of U , and let OPT_{SC} be the minimum cost of the corresponding SETCOVER instance. Then

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

Proof.

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

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

$$\sum_{i=1}^k |\pi_i| = \sum_{i=1}^k c(S(\pi_i)) = \text{OPT}_{\text{SC}}.$$

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

Relating SSS and SETCOVER

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

Relating SSS and SETCOVER

Lemma. $OPT_{sc} \leq 2 \cdot OPT_{SSS}$.

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 .

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 $\leq 2|s| = 2 \cdot \text{OPT}_{\text{SSS}}$.

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. $\text{OPT}_{\text{sc}} \leq 2 \cdot \text{OPT}_{\text{SSS}}$.

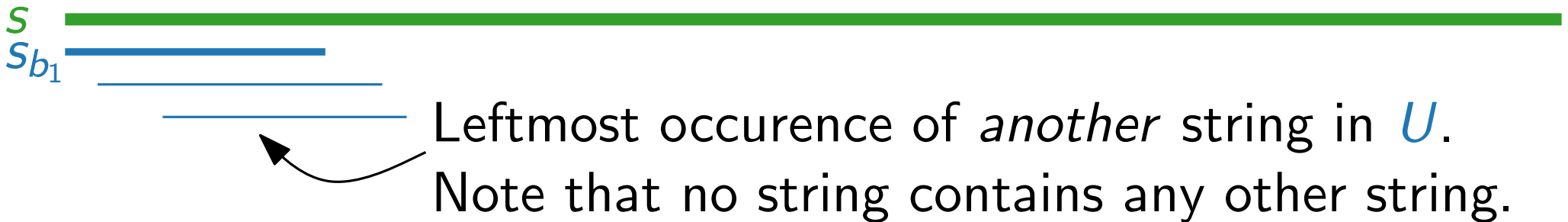
Proof. Consider an optimal superstring s .
Construct a set cover of cost $\leq 2|s| = 2 \cdot \text{OPT}_{\text{SSS}}$.



Relating SSS and SETCOVER

Lemma. $OPT_{sc} \leq 2 \cdot OPT_{SSS}$.

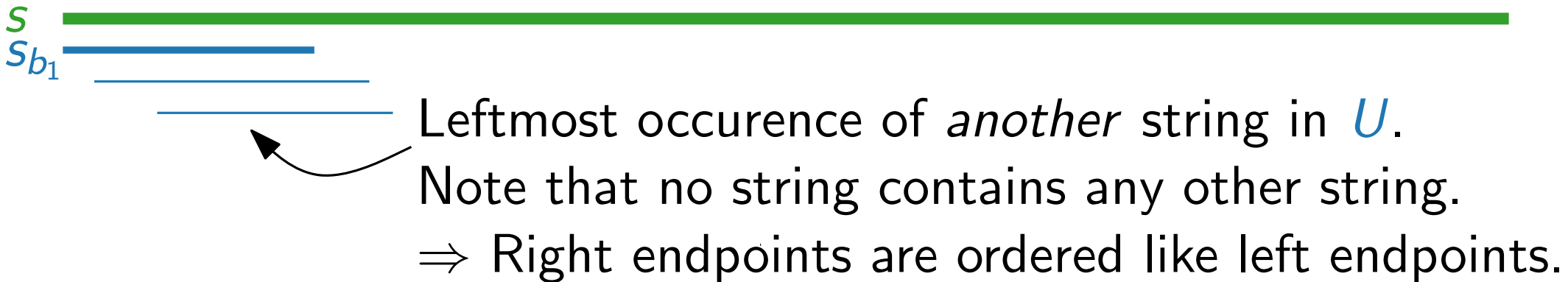
Proof. Consider an optimal superstring s .
Construct a set cover of cost $\leq 2|s| = 2 \cdot OPT_{SSS}$.



Relating SSS and SETCOVER

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

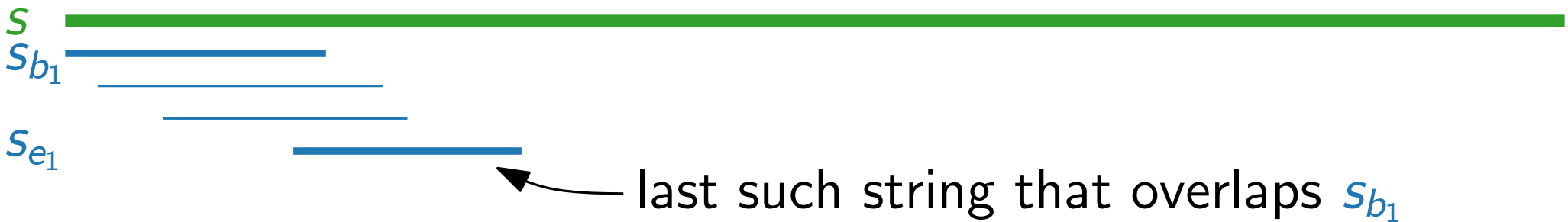
Proof. Consider an optimal superstring s .
Construct a set cover of cost $\leq 2|s| = 2 \cdot \text{OPT}_{\text{SSS}}$.



Relating SSS and SETCOVER

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

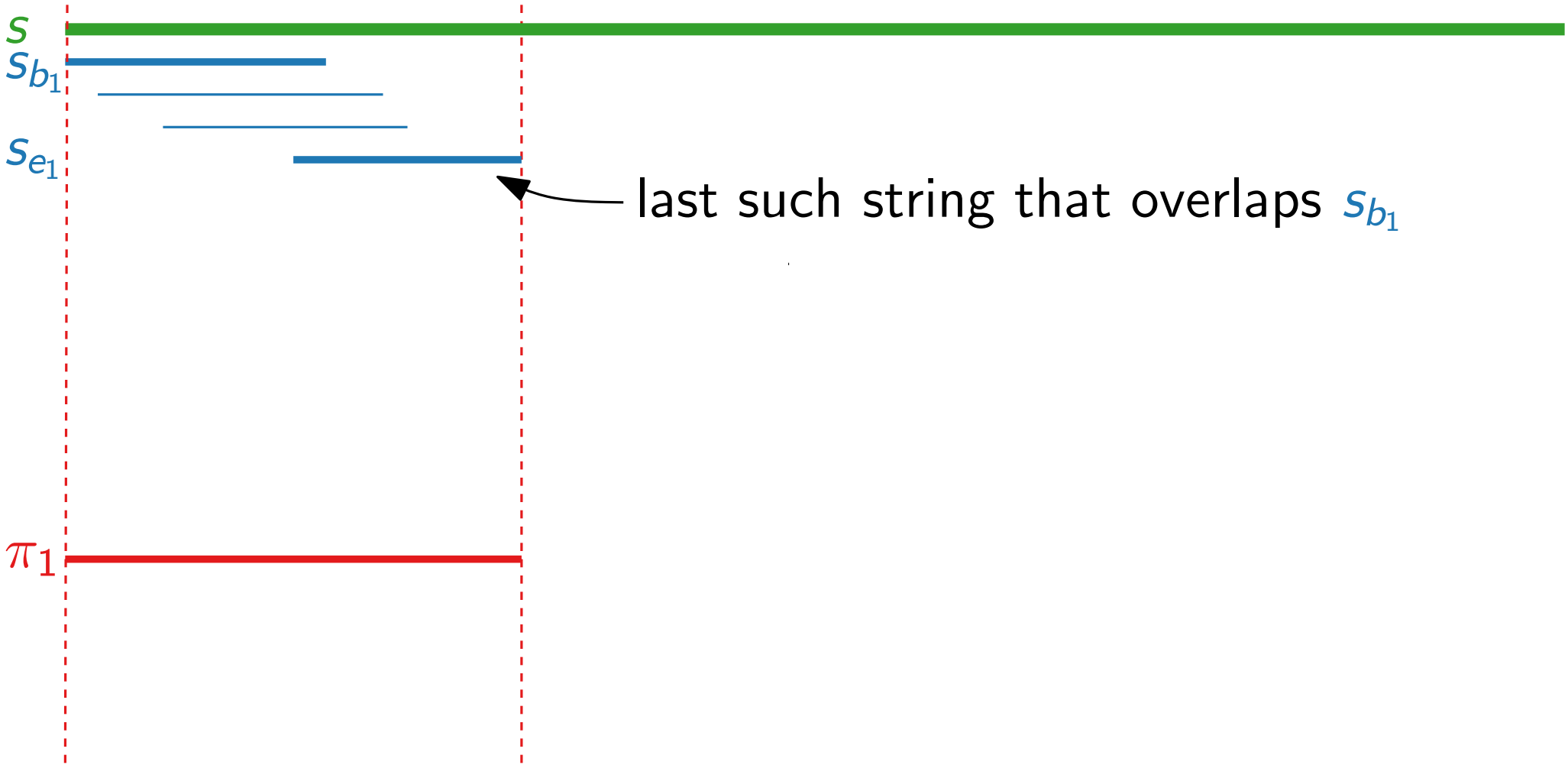
Proof. Consider an optimal superstring s .
Construct a set cover of cost $\leq 2|s| = 2 \cdot \text{OPT}_{\text{SSS}}$.



Relating SSS and SETCOVER

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

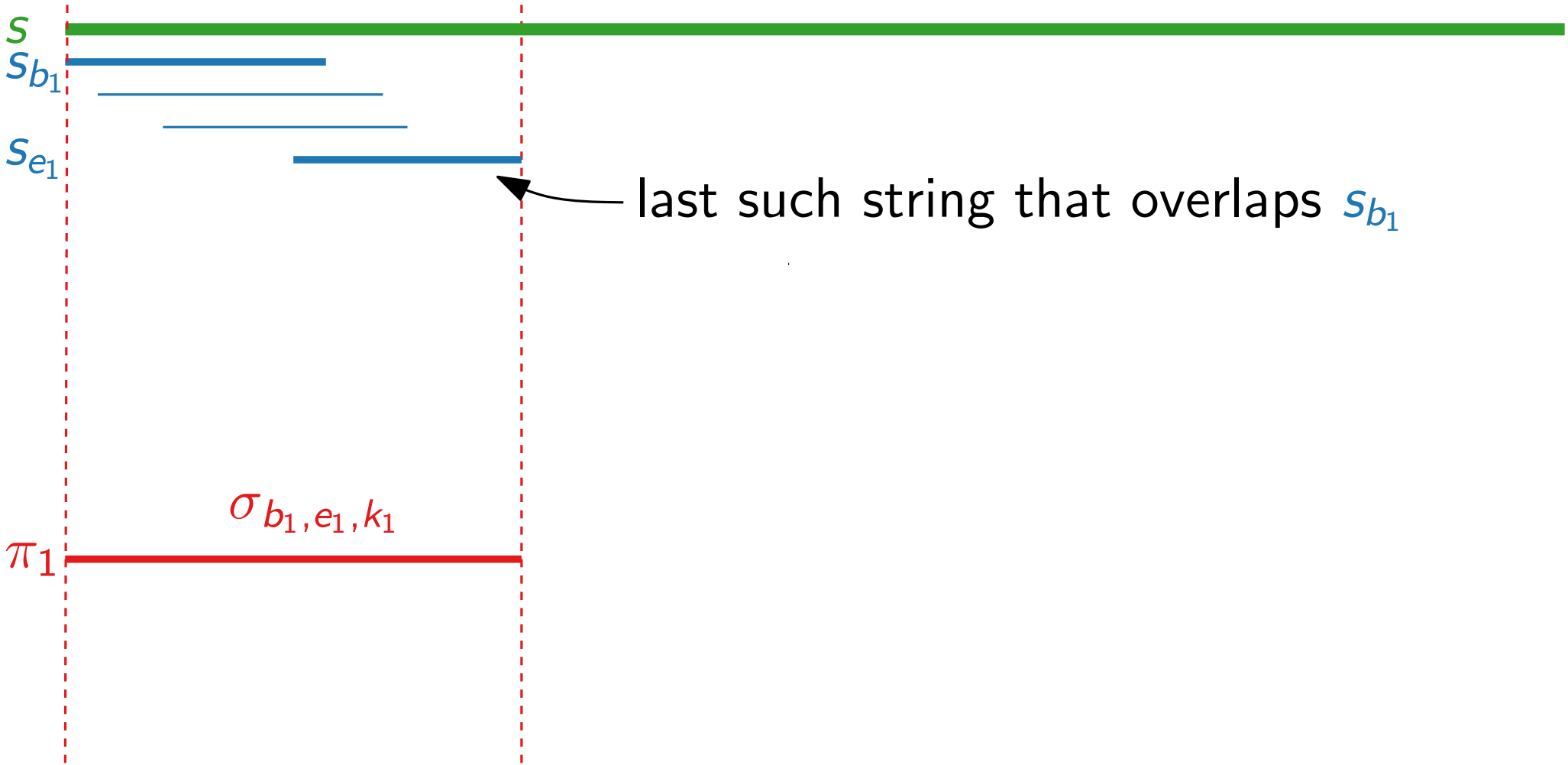
Proof. Consider an optimal superstring s .
Construct a set cover of cost $\leq 2|s| = 2 \cdot \text{OPT}_{\text{SSS}}$.



Relating SSS and SETCOVER

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

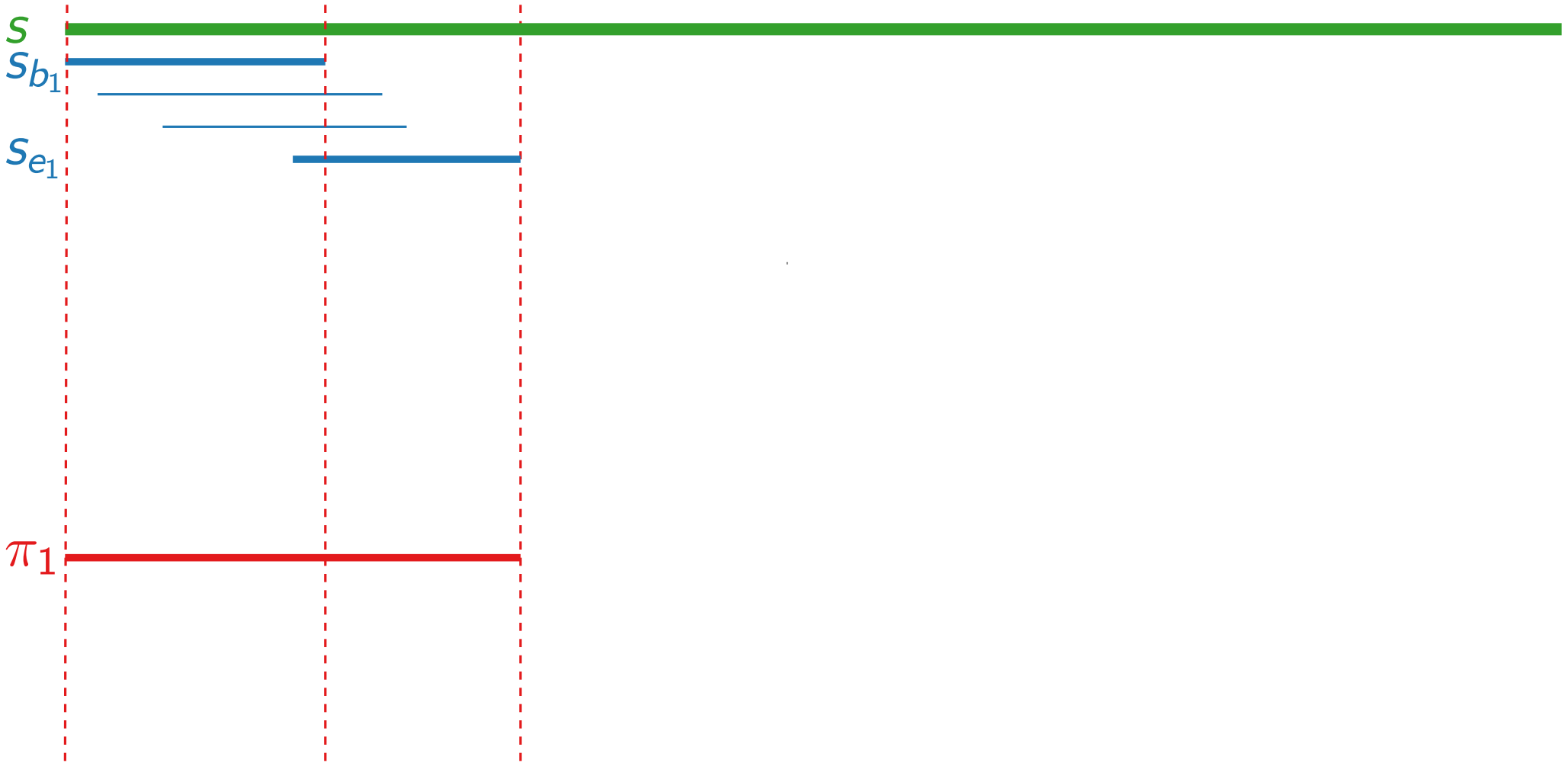
Proof. Consider an optimal superstring s .
Construct a set cover of cost $\leq 2|s| = 2 \cdot \text{OPT}_{\text{SSS}}$.



Relating SSS and SETCOVER

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

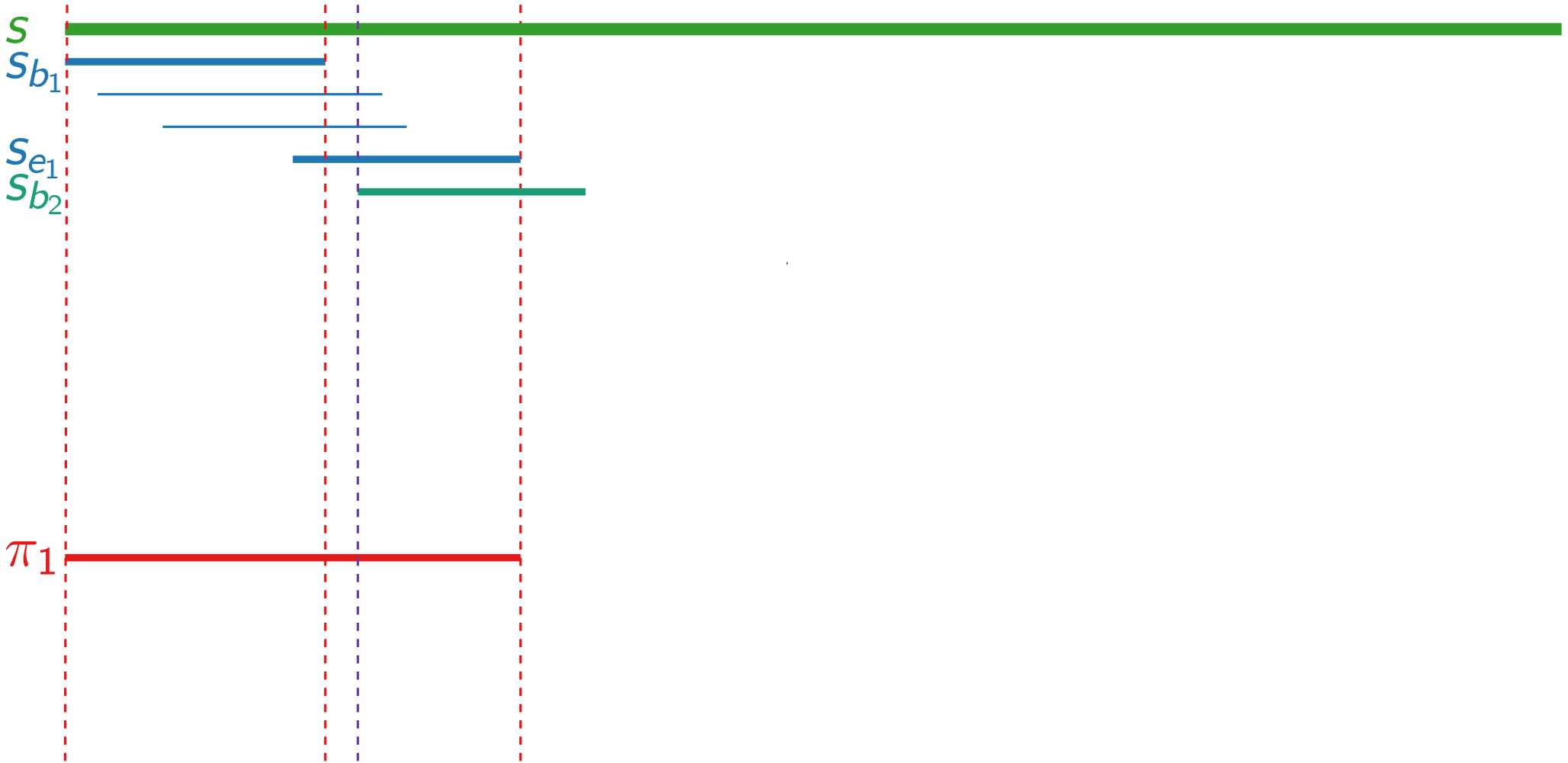
Proof. Consider an optimal superstring s .
Construct a set cover of cost $\leq 2|s| = 2 \cdot \text{OPT}_{\text{SSS}}$.



Relating SSS and SETCOVER

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

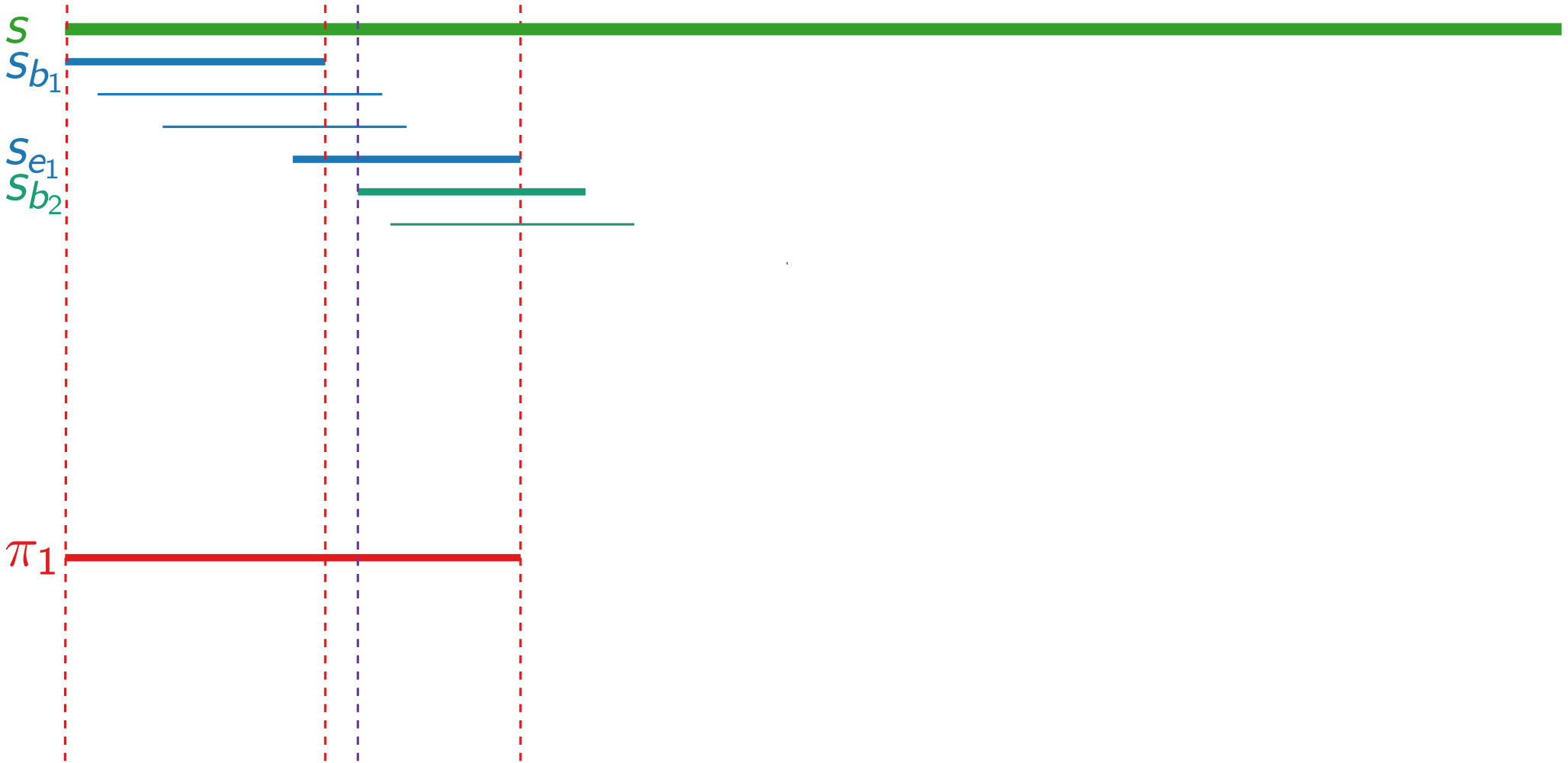
Proof. Consider an optimal superstring s .
Construct a set cover of cost $\leq 2|s| = 2 \cdot \text{OPT}_{\text{SSS}}$.



Relating SSS and SETCOVER

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

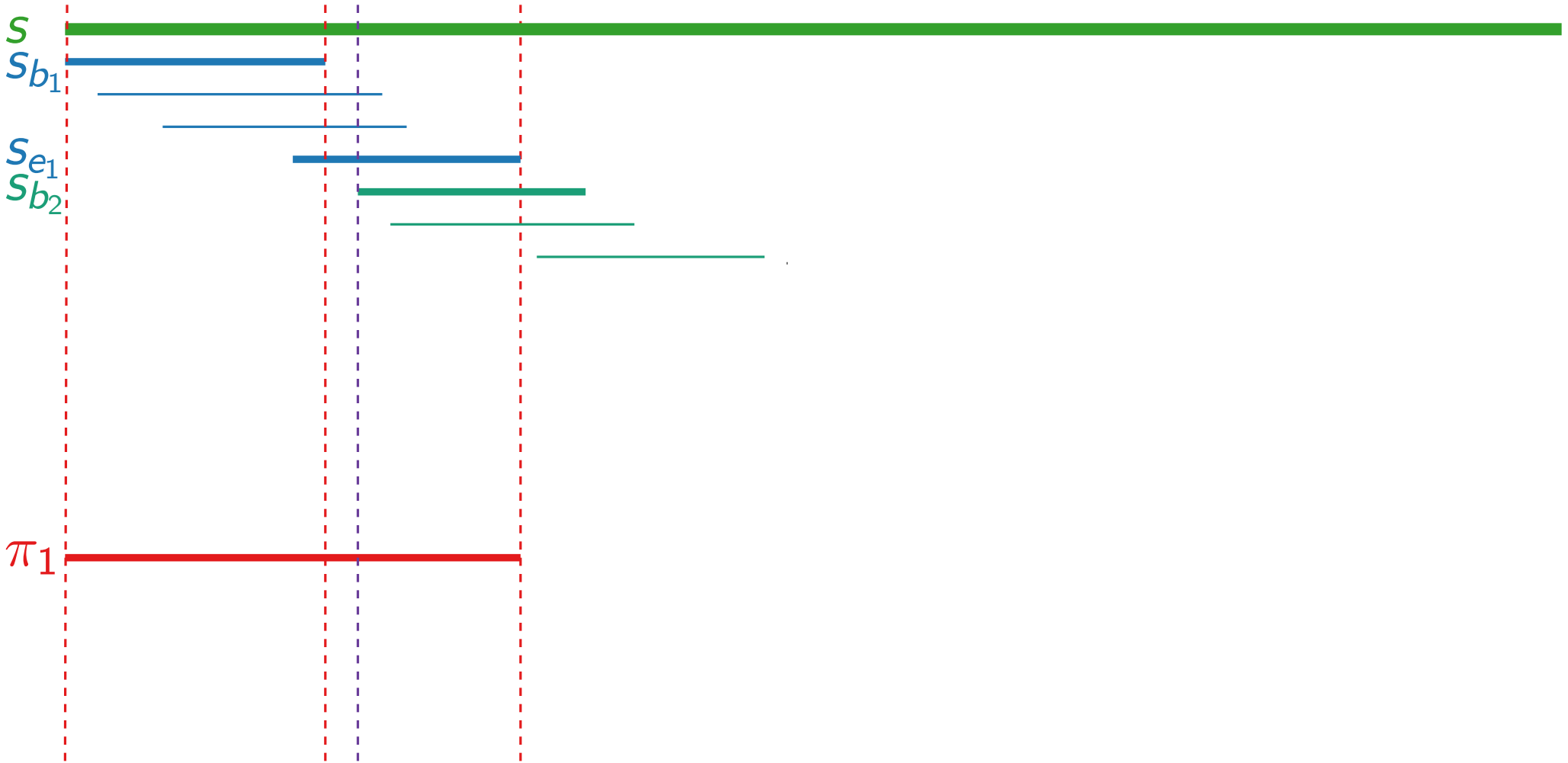
Proof. Consider an optimal superstring s .
Construct a set cover of cost $\leq 2|s| = 2 \cdot \text{OPT}_{\text{SSS}}$.



Relating SSS and SETCOVER

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

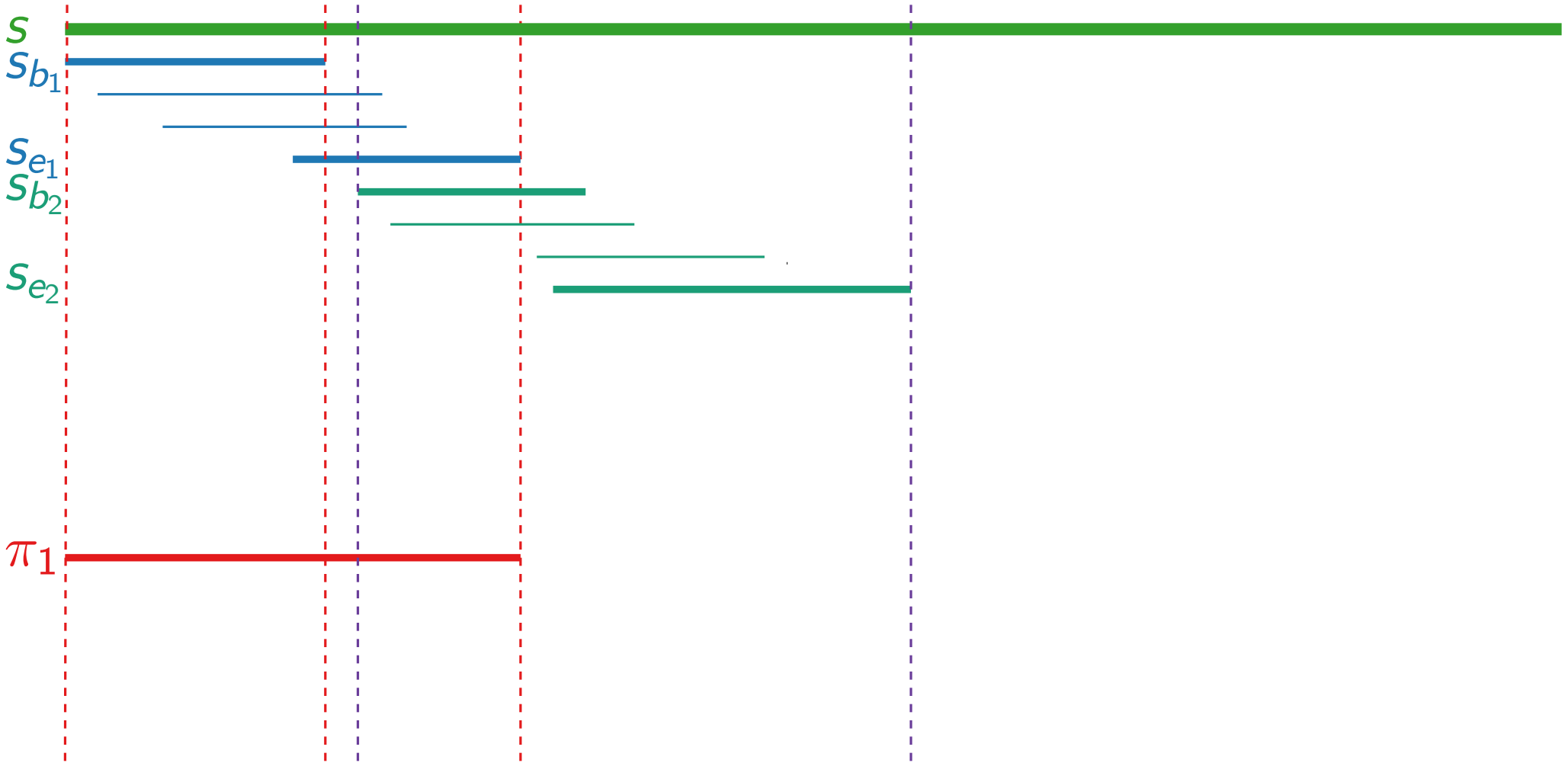
Proof. Consider an optimal superstring s .
Construct a set cover of cost $\leq 2|s| = 2 \cdot \text{OPT}_{\text{SSS}}$.



Relating SSS and SETCOVER

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

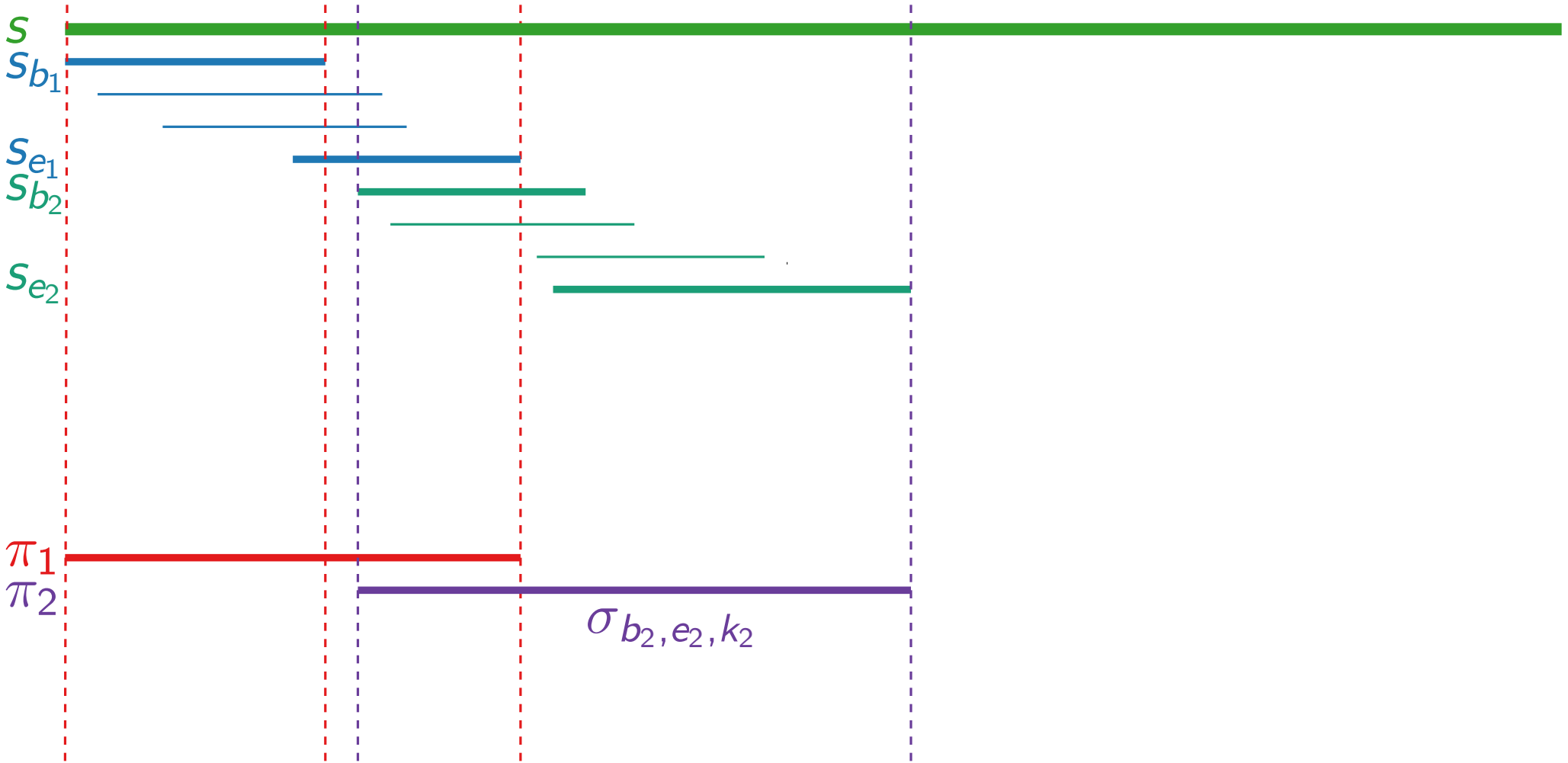
Proof. Consider an optimal superstring s .
Construct a set cover of cost $\leq 2|s| = 2 \cdot \text{OPT}_{\text{SSS}}$.



Relating SSS and SETCOVER

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

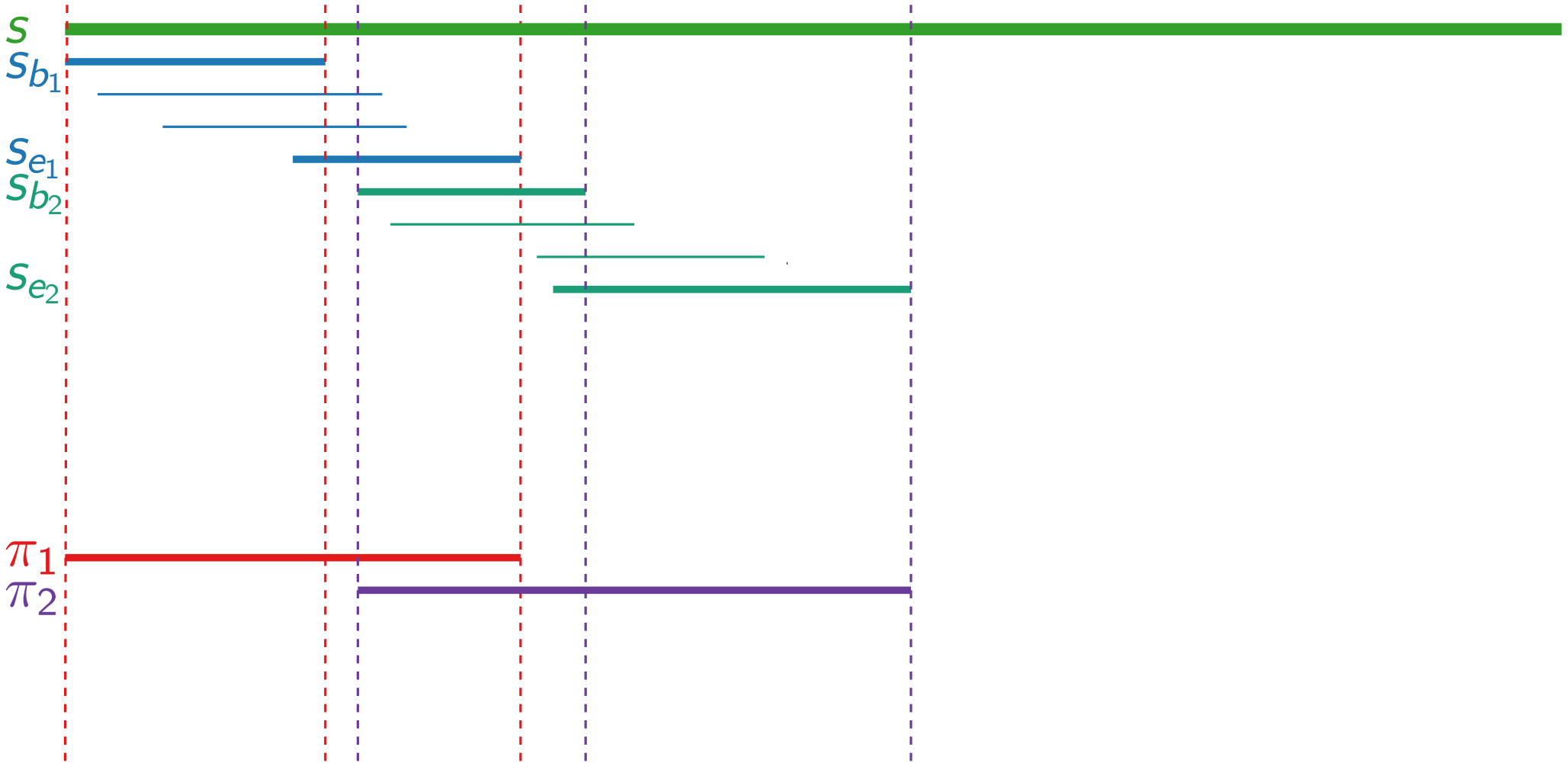
Proof. Consider an optimal superstring s .
Construct a set cover of cost $\leq 2|s| = 2 \cdot \text{OPT}_{\text{SSS}}$.



Relating SSS and SETCOVER

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

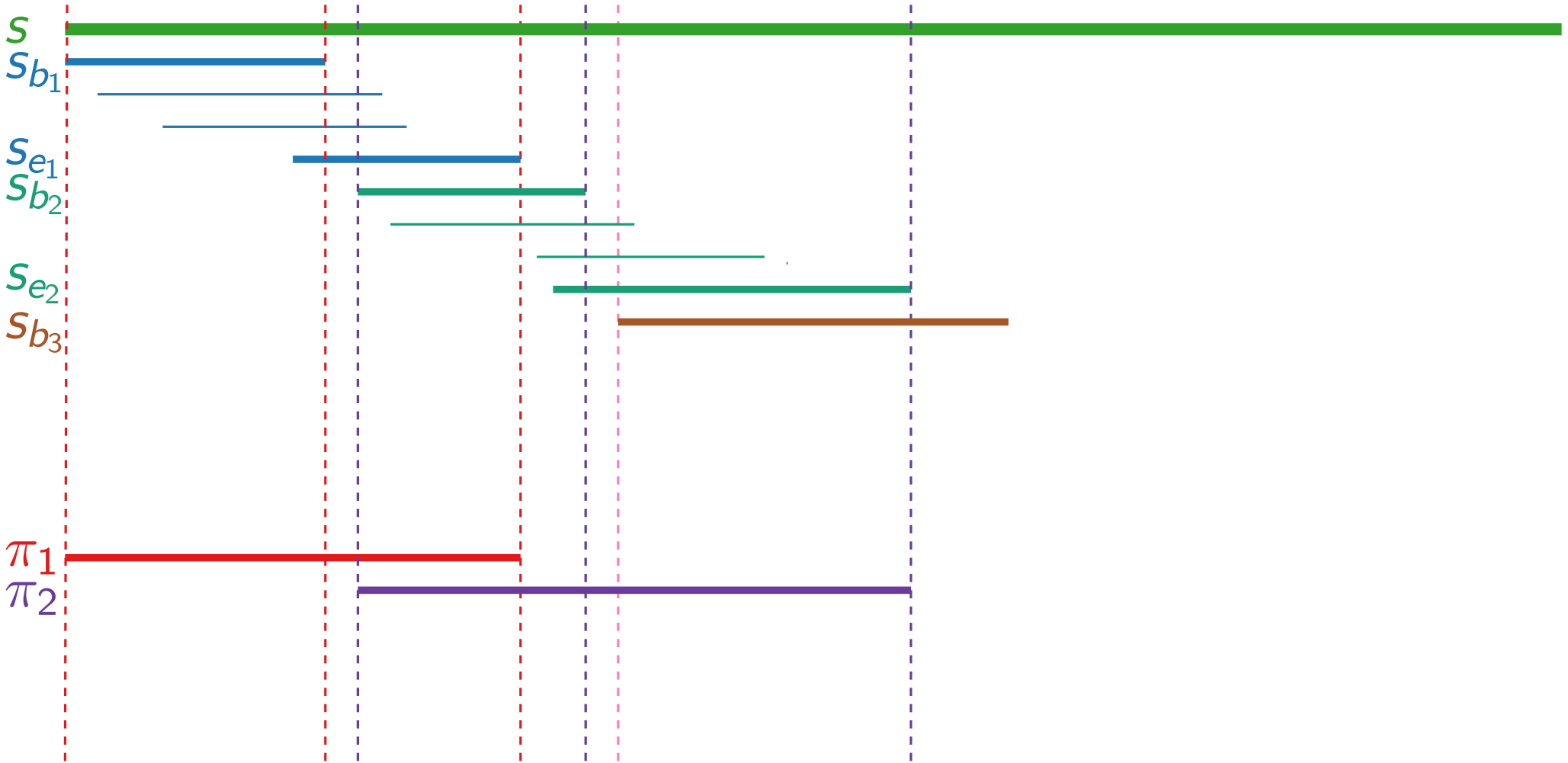
Proof. Consider an optimal superstring s .
Construct a set cover of cost $\leq 2|s| = 2 \cdot \text{OPT}_{\text{SSS}}$.



Relating SSS and SETCOVER

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

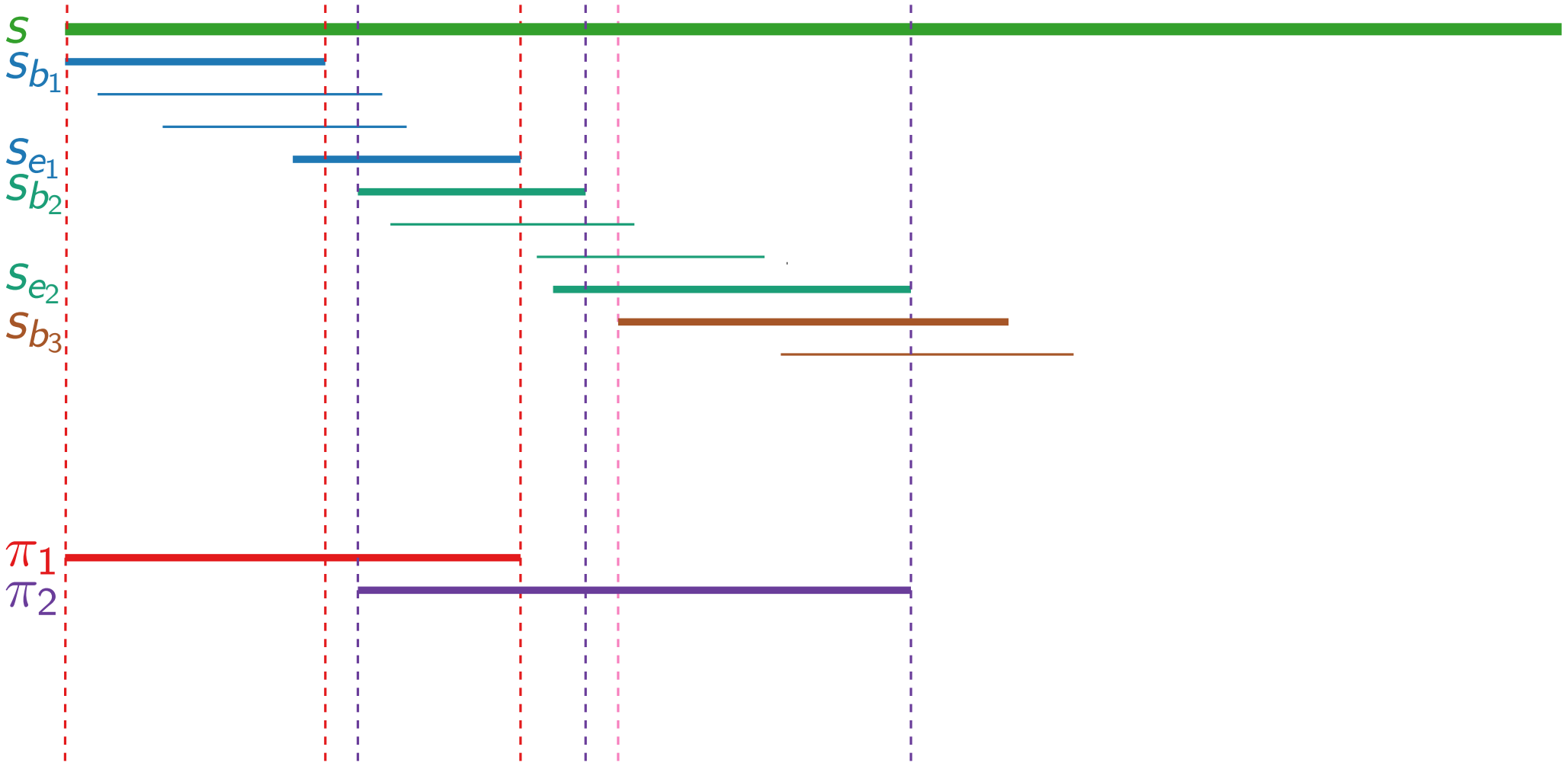
Proof. Consider an optimal superstring s .
Construct a set cover of cost $\leq 2|s| = 2 \cdot \text{OPT}_{\text{SSS}}$.



Relating SSS and SETCOVER

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

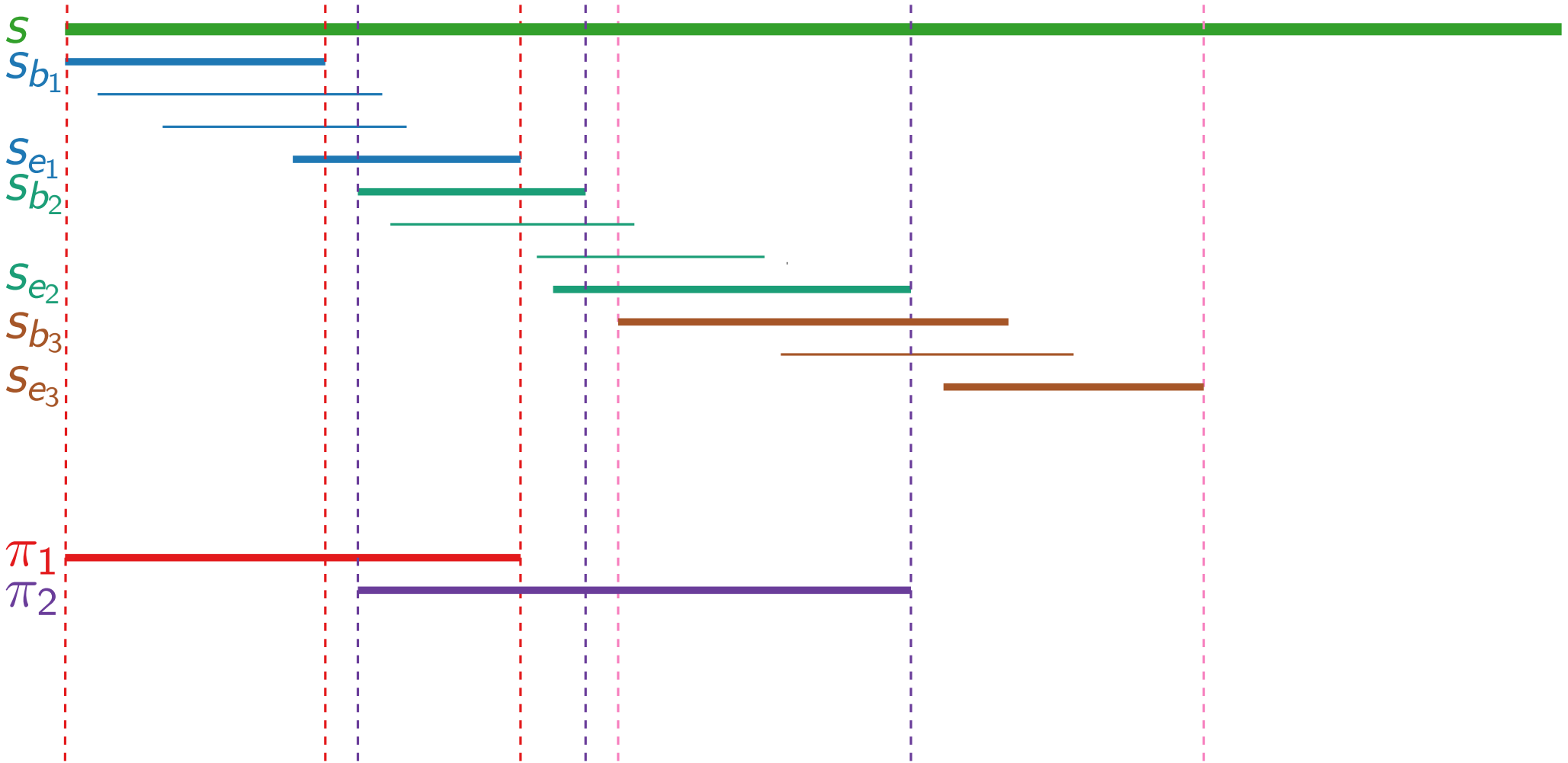
Proof. Consider an optimal superstring s .
Construct a set cover of cost $\leq 2|s| = 2 \cdot \text{OPT}_{\text{SSS}}$.



Relating SSS and SETCOVER

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

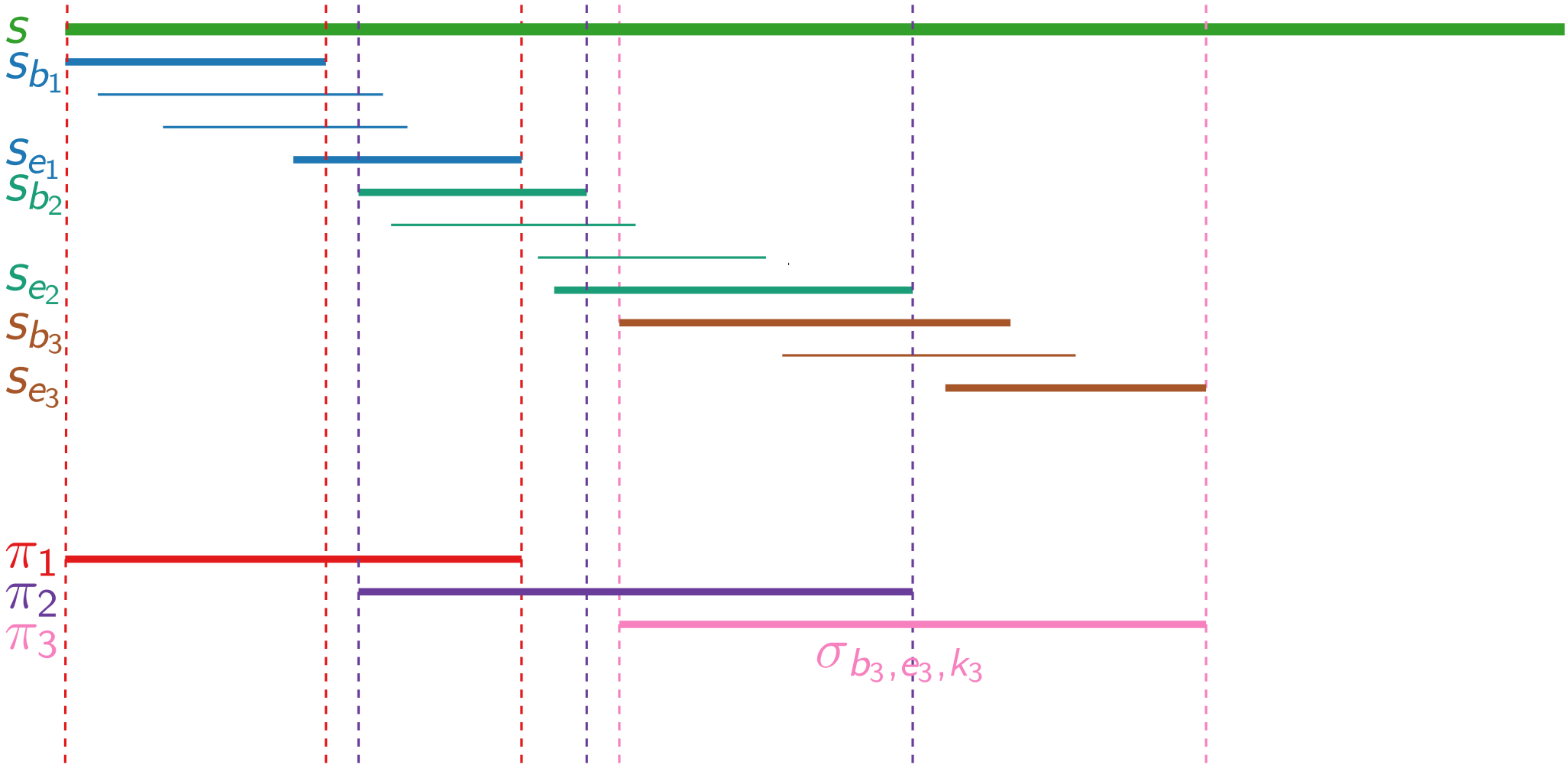
Proof. Consider an optimal superstring s .
Construct a set cover of cost $\leq 2|s| = 2 \cdot \text{OPT}_{\text{SSS}}$.



Relating SSS and SETCOVER

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

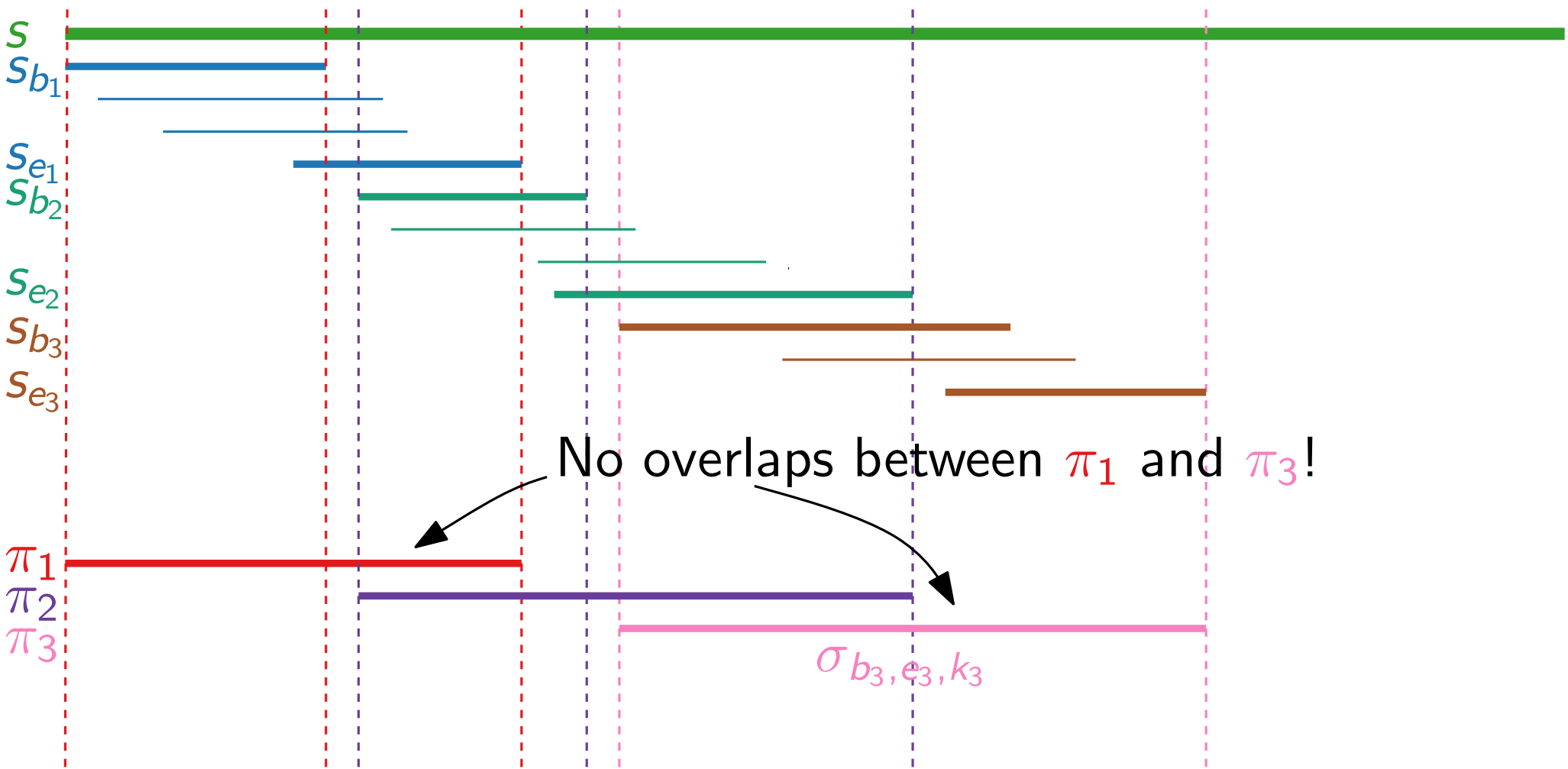
Proof. Consider an optimal superstring s .
Construct a set cover of cost $\leq 2|s| = 2 \cdot \text{OPT}_{\text{SSS}}$.



Relating SSS and SETCOVER

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

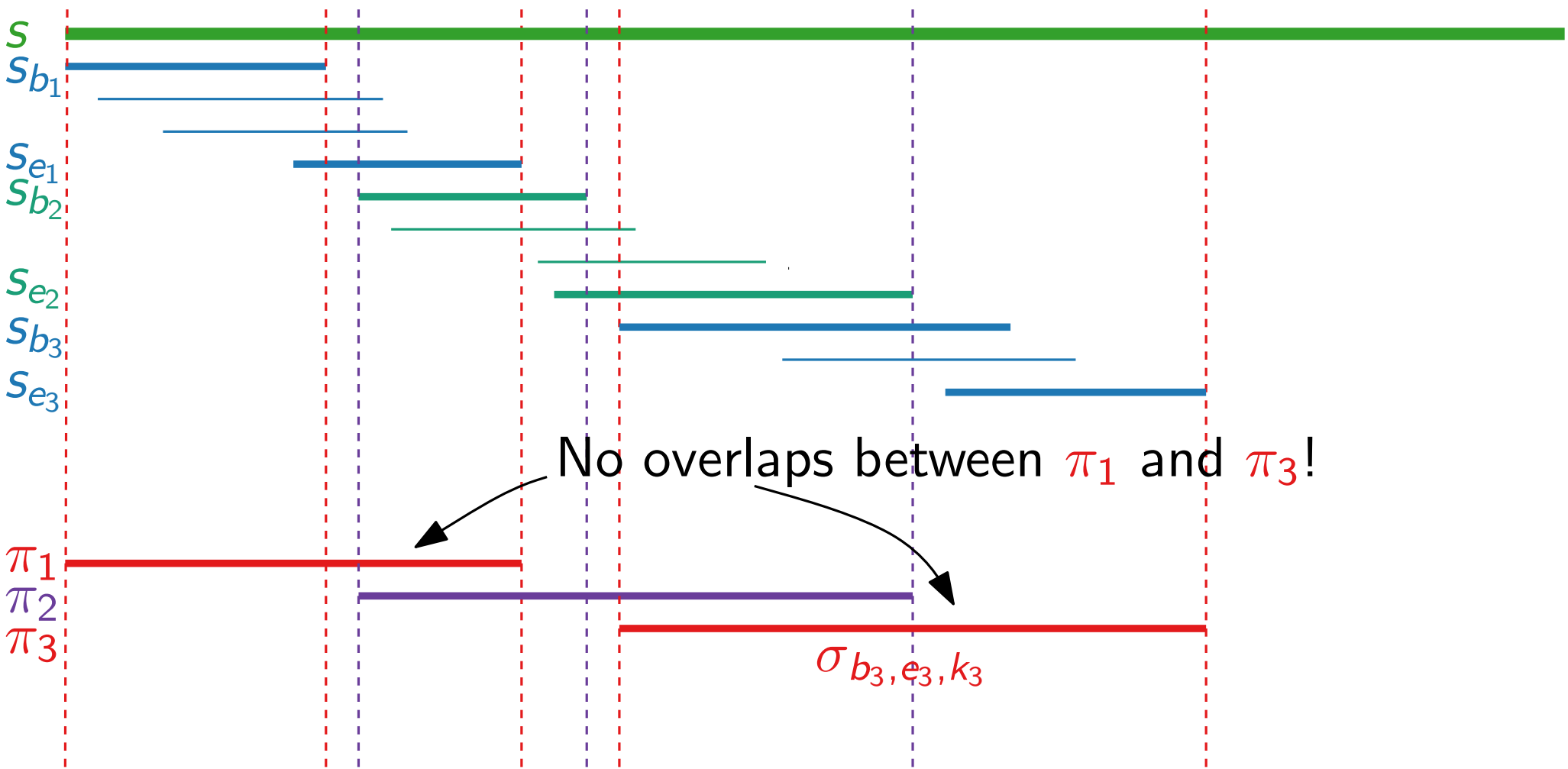
Proof. Consider an optimal superstring s .
Construct a set cover of cost $\leq 2|s| = 2 \cdot \text{OPT}_{\text{SSS}}$.



Relating SSS and SETCOVER

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

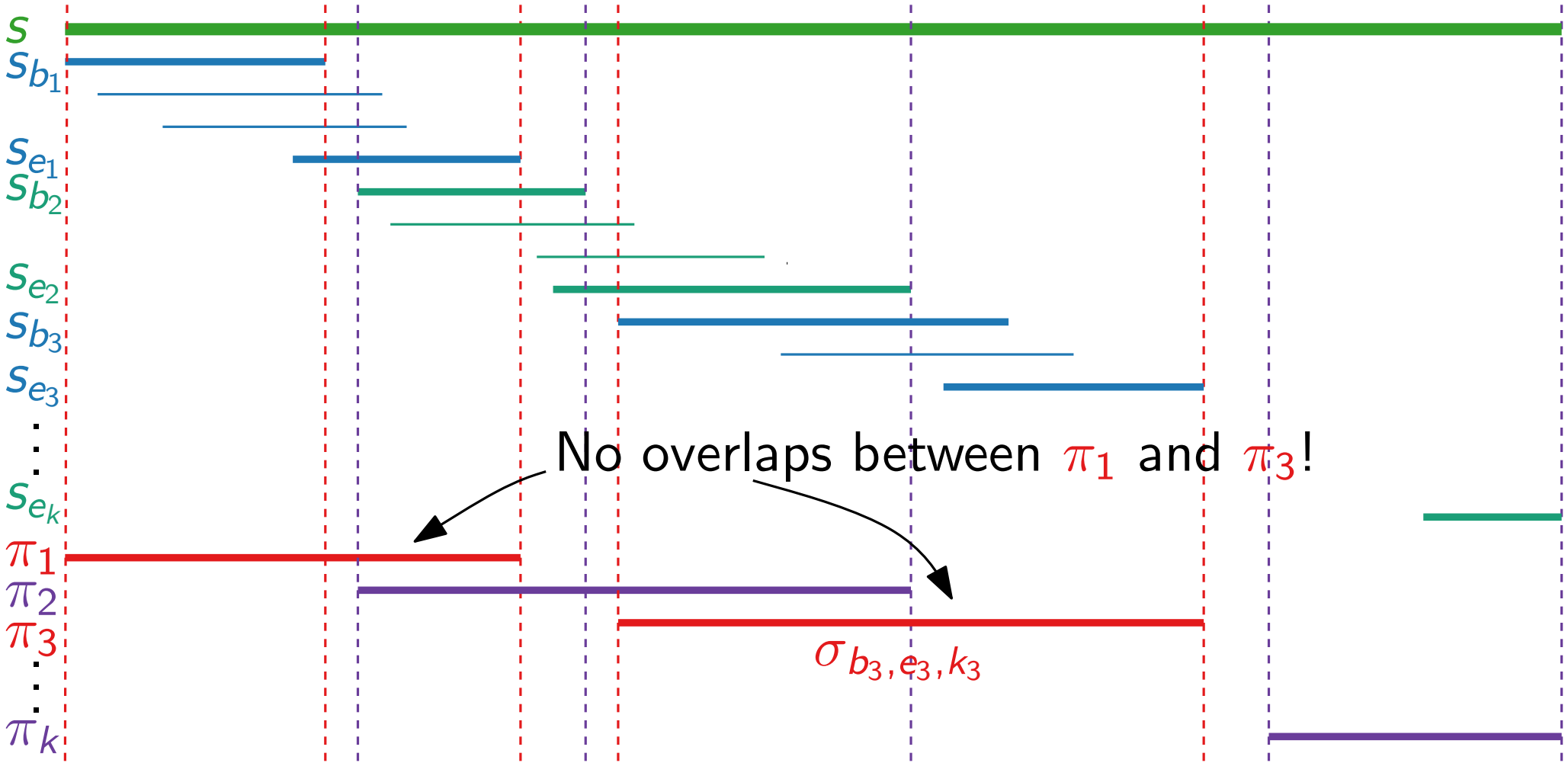
Proof. Consider an optimal superstring s .
Construct a set cover of cost $\leq 2|s| = 2 \cdot \text{OPT}_{\text{SSS}}$.



Relating SSS and SETCOVER

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

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.

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

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

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 π_j .

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

For $j \in \{1, \dots, k-2\}$, substrings π_j, π_{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 π_j .

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

For $j \in \{1, \dots, k-2\}$, substrings π_j, π_{j+2} do **not** overlap.

Each character of the optimal superstring s lies in at most **two** (subsequent) substrings, say, π_j and π_{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 π_j .

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

For $j \in \{1, \dots, k-2\}$, substrings π_j, π_{j+2} do **not** overlap.

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

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

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 π_j .

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

For $j \in \{1, \dots, k-2\}$, substrings π_j, π_{j+2} do **not** overlap.

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

$$\text{OPT}_{\text{sc}} \leq \sum_i |\pi_i| \leq$$

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 π_j .

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

For $j \in \{1, \dots, k-2\}$, substrings π_j, π_{j+2} do **not** overlap.

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

$$\text{OPT}_{\text{sc}} \leq \sum_i |\pi_i| \leq 2|s| =$$

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 π_j .

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

For $j \in \{1, \dots, k-2\}$, substrings π_j, π_{j+2} do **not** overlap.

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

$$\text{OPT}_{\text{sc}} \leq \sum_i |\pi_i| \leq 2|s| = 2 \cdot \text{OPT}_{\text{SSS}}$$

□

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 $\{S(\pi_1), \dots, S(\pi_k)\}$ with the algorithm GreedySetCover.

Algorithm for SSS

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

Algorithm for SSS

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

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. $\text{OPT}_{\text{sc}} \leq 2 \cdot \text{OPT}_{\text{SSS}}$.

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. $\text{OPT}_{\text{sc}} \leq 2 \cdot \text{OPT}_{\text{SSS}}$.

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

$$\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \leq 1 + \ln k.$$

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.

What is n ?

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

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

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

$$\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \leq 1 + \ln k.$$

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.

What is n ?

$$n = \max_{S \in \mathcal{S}} |S|$$

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

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

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

$$\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \leq 1 + \ln k.$$

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.

What is n ?

$$\begin{aligned} n &= \max_{S \in \mathcal{S}} |S| \\ &= \max |S(\sigma_{ijk})| \end{aligned}$$

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

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

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

$$\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \leq 1 + \ln k.$$

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.

What is n ?

$$n = \max_{S \in \mathcal{S}} |S| \\ = \max |S(\sigma_{ijk})|$$

s_i  s_j

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

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

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

$$\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \leq 1 + \ln k.$$

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.

What is n ?

$$n = \max_{S \in \mathcal{S}} |S| \\ = \max |S(\sigma_{ijk})|$$

s_i  s_j

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

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

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

$$\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \leq 1 + \ln k.$$

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.

What is n ?

$$n = \max_{S \in \mathcal{S}} |S| \\ = \max |S(\sigma_{ijk})|$$



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

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

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

$$\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \leq 1 + \ln k.$$

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.

What is n ?

$$\begin{aligned} n &= \max_{S \in \mathcal{S}} |S| \\ &= \max |S(\sigma_{ijk})| \end{aligned}$$



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

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

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

$$\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \leq 1 + \ln k.$$

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.

What is n ?

$$\begin{aligned} n &= \max_{S \in \mathcal{S}} |S| \\ &= \max |S(\sigma_{ijk})| \\ &= \max_{u \in U} |u| \end{aligned}$$



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

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

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

$$\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \leq 1 + \ln k.$$

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 $P = NP$).
[Karpinski & Schmied: CATS 2013]