Aalto University
School of Science
Department of Computer Science
Combinatorics of Efficient Computations

Approximation Algorithms

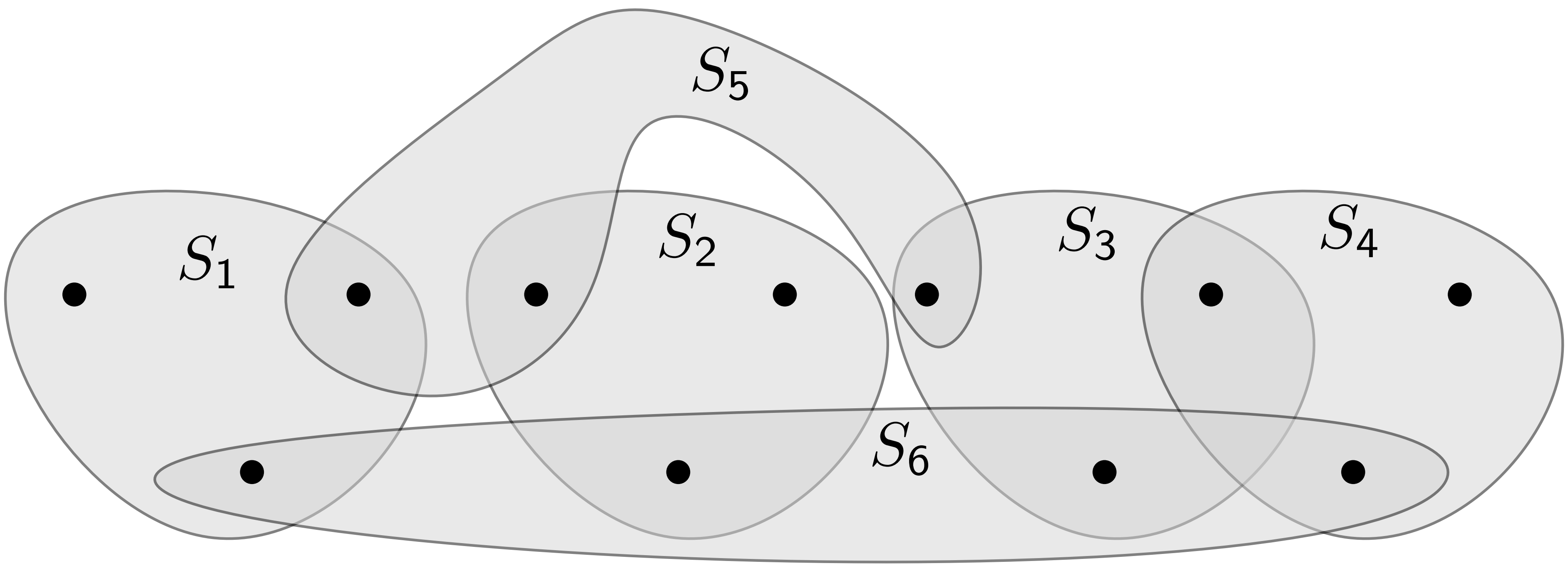
Lecture 2: Set Cover & Shortest SuperString

Joachim Spoerhase

2019

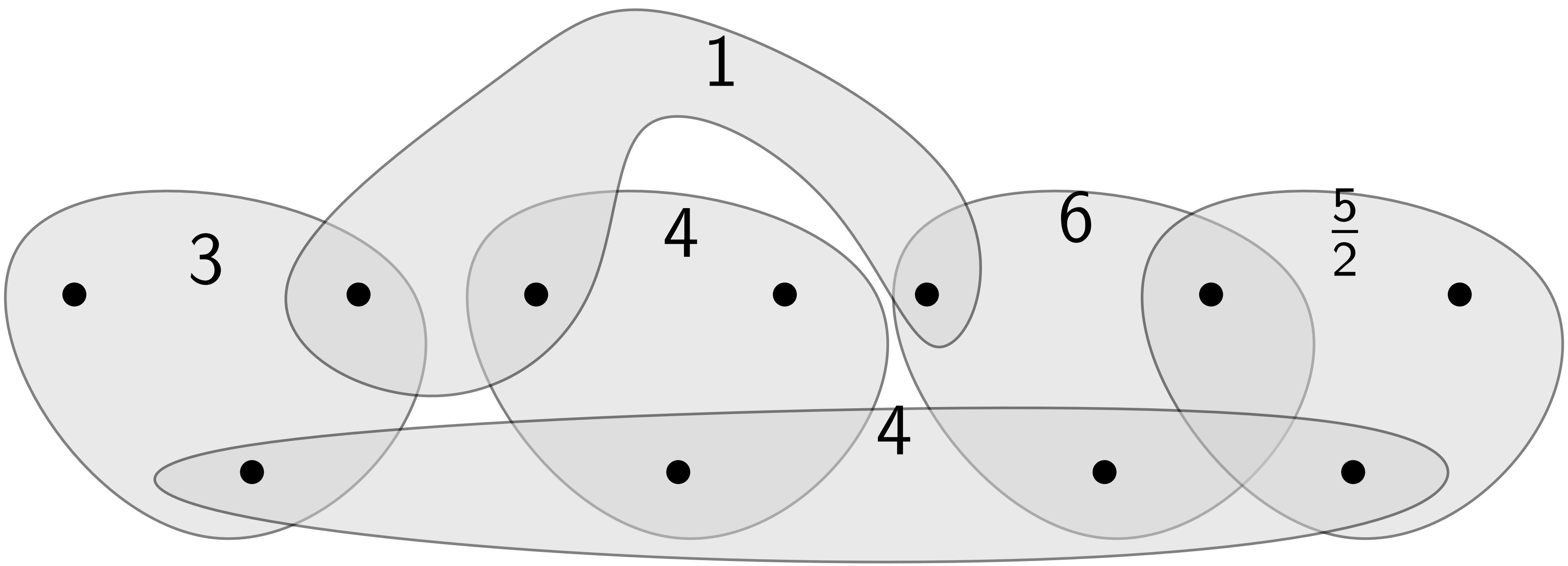
SetCover (card.)

- Given a **ground set** U and a collection $\mathcal{S}$ of **subsets** of U where $\bigcup \mathcal{S} = U$.
- Find a **cover** $\mathcal{S}' \subseteq \mathcal{S}$ of U (i.e. with $\bigcup \mathcal{S}' = U$) of minimum cardinality.



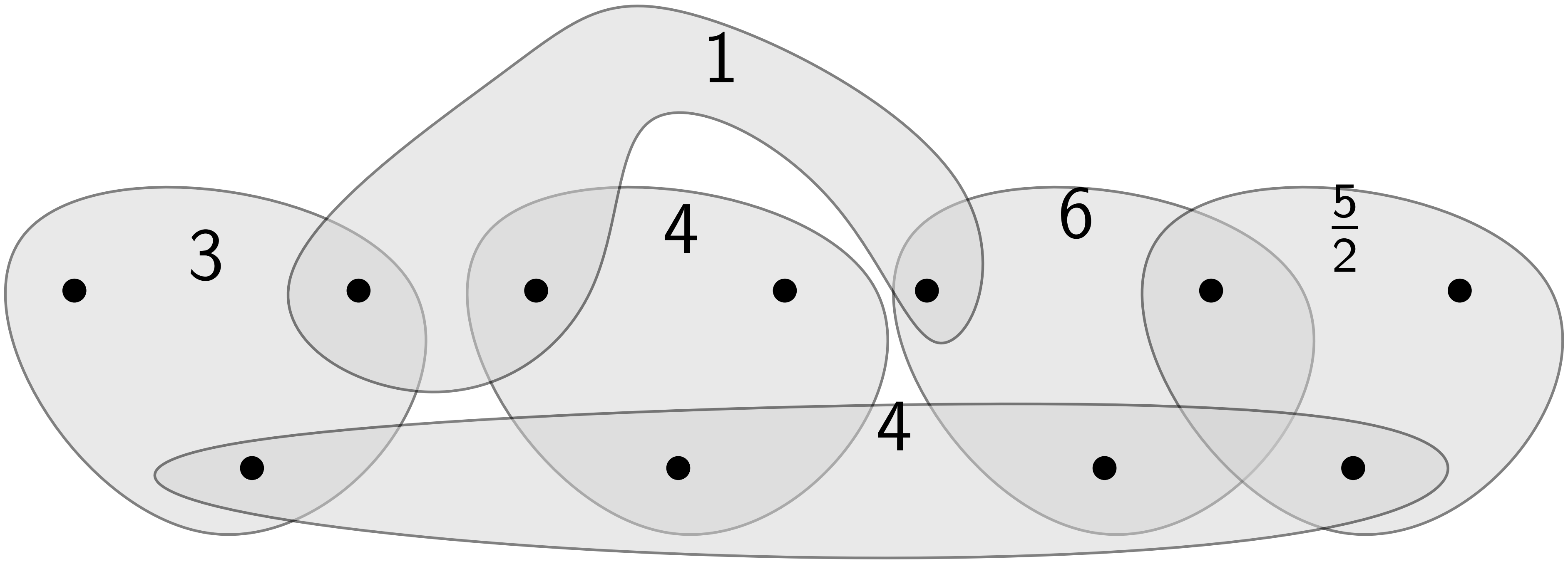
SetCover (general)

- Given a **ground set** U and a collection $\mathcal{S}$ of **subsets** of U where $\bigcup \mathcal{S} = U$. and each $S \in \mathcal{S}$ has a postive cost $c(S)$.
- Find a **cover** $\mathcal{S}' \subseteq \mathcal{S}$ of U (i.e. with $\bigcup \mathcal{S}' = U$) of minimum ~~cardinality.~~ total cost $c(\mathcal{S}') := \sum_{S \in \mathcal{S}'} c(S)$.



Iterative "re-pricing"

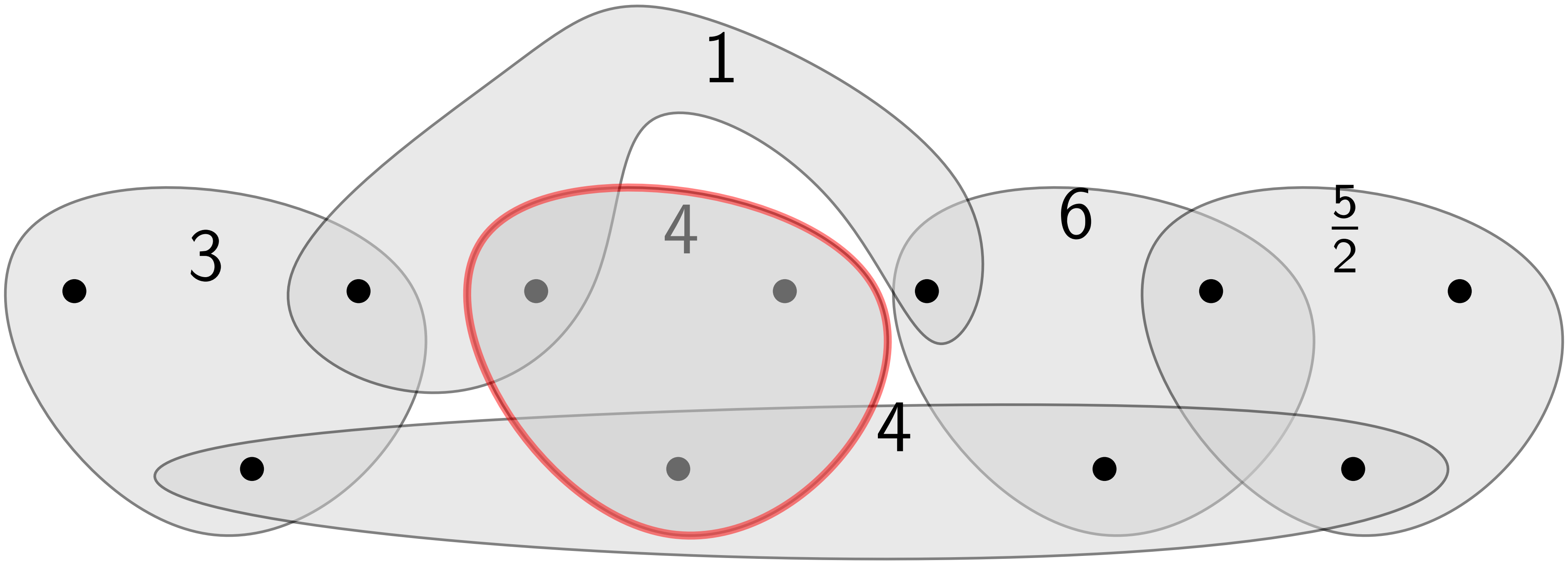
What is the cost of picking a set?



Iterative "re-pricing"

What is the cost of picking a set?

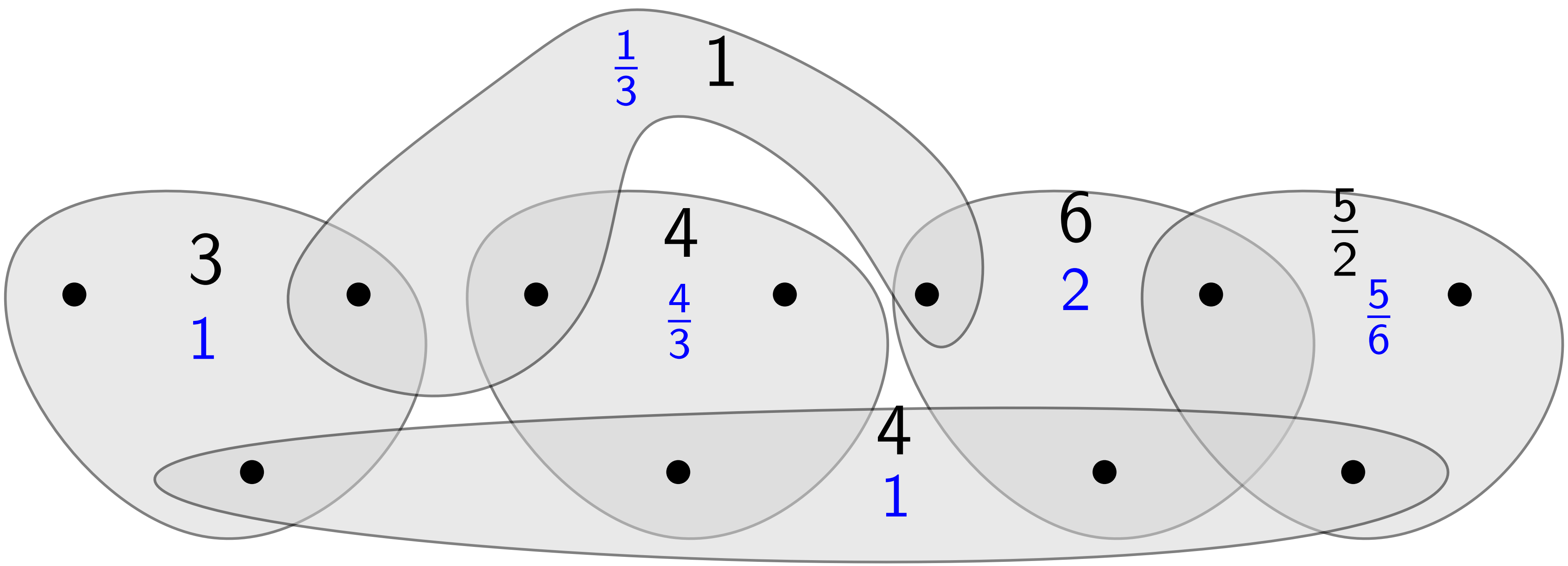
Choosing a 3-element set with cost 4 $\rightsquigarrow$ unit price of $\frac{4}{3}$
i.e., elements of S can be "bought" for a "price" of $\frac{4}{3}$ each.



Iterative "re-pricing"

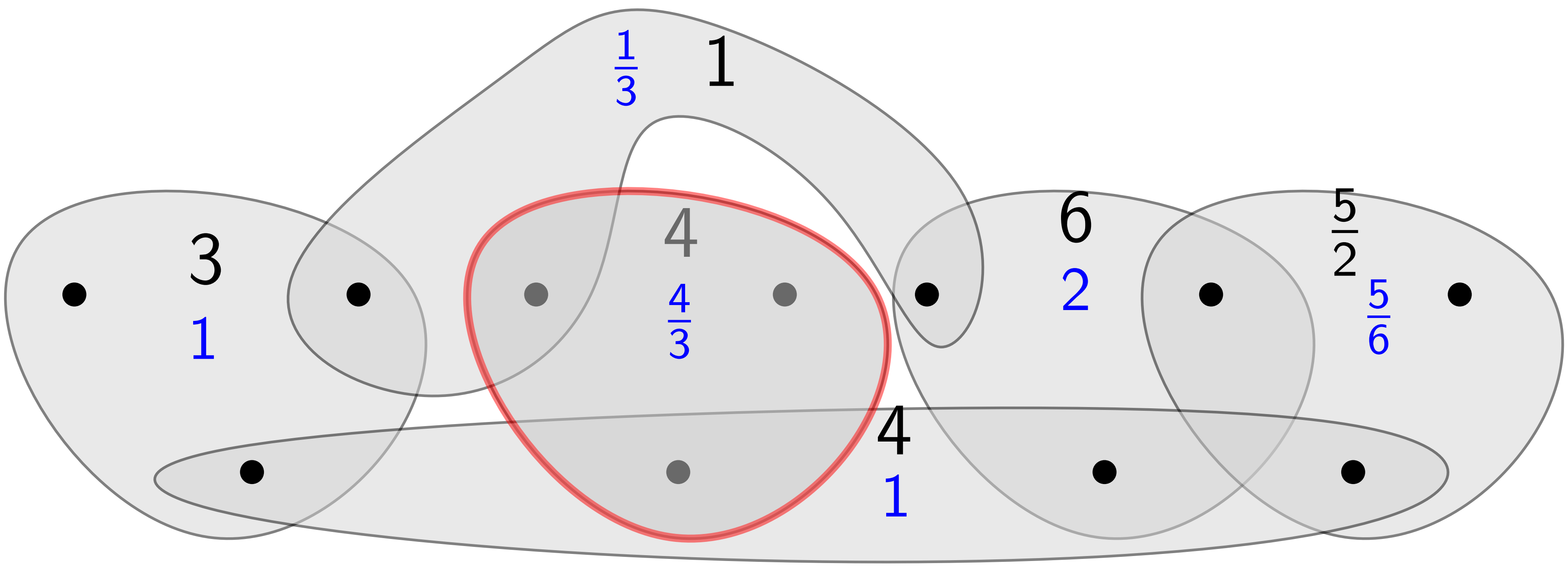
What is the cost of picking a set?

Choosing a 3-element set with cost 4 $\rightsquigarrow$ unit price of $\frac{4}{3}$
i.e., elements of S can be "bought" for a "price" of $\frac{4}{3}$ each.



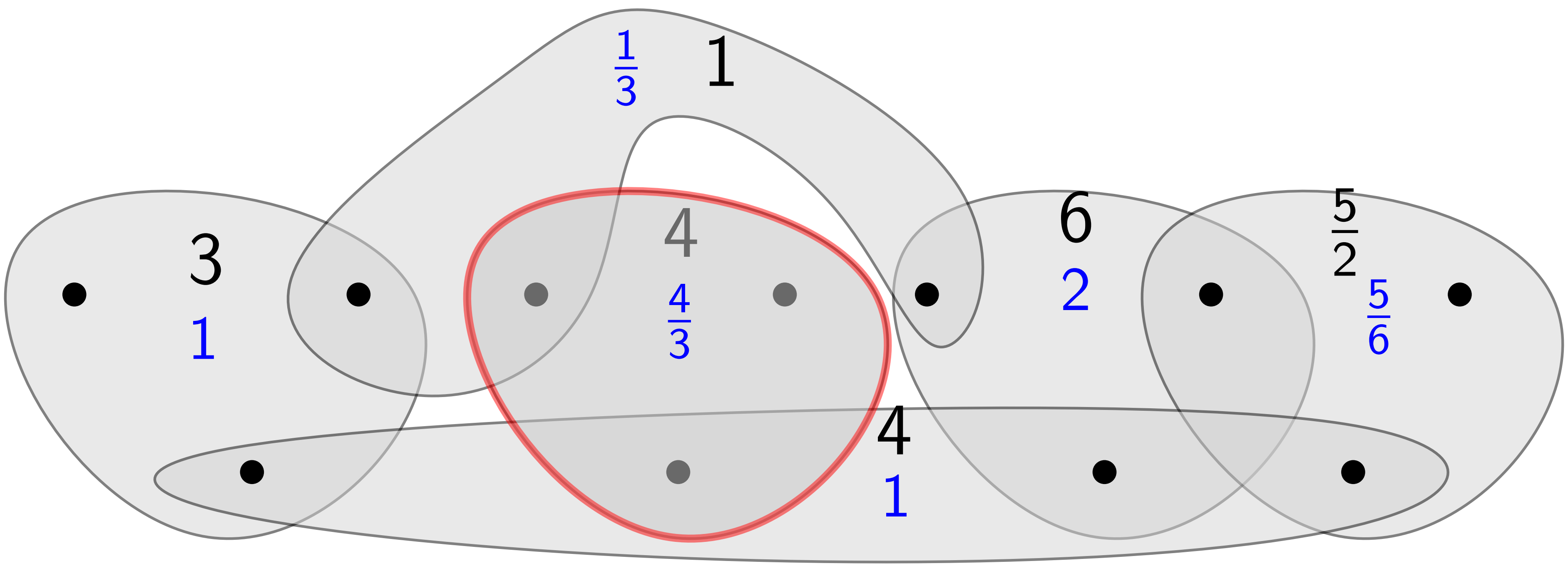
Iterative "re-pricing"

What happens if we "buy" a set?



Iterative "re-pricing"

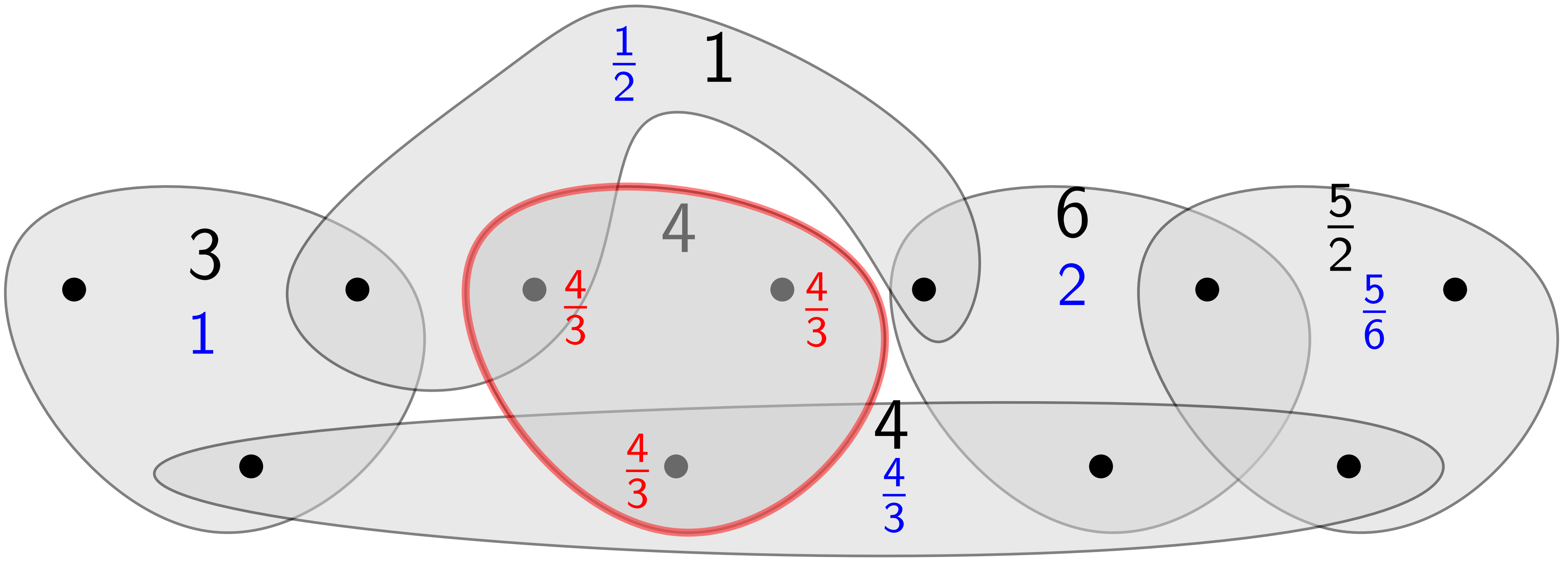
What happens if we "buy" a set?
Fix price of the elements bought and re-evaluate unit prices!



Iterative “re-pricing”

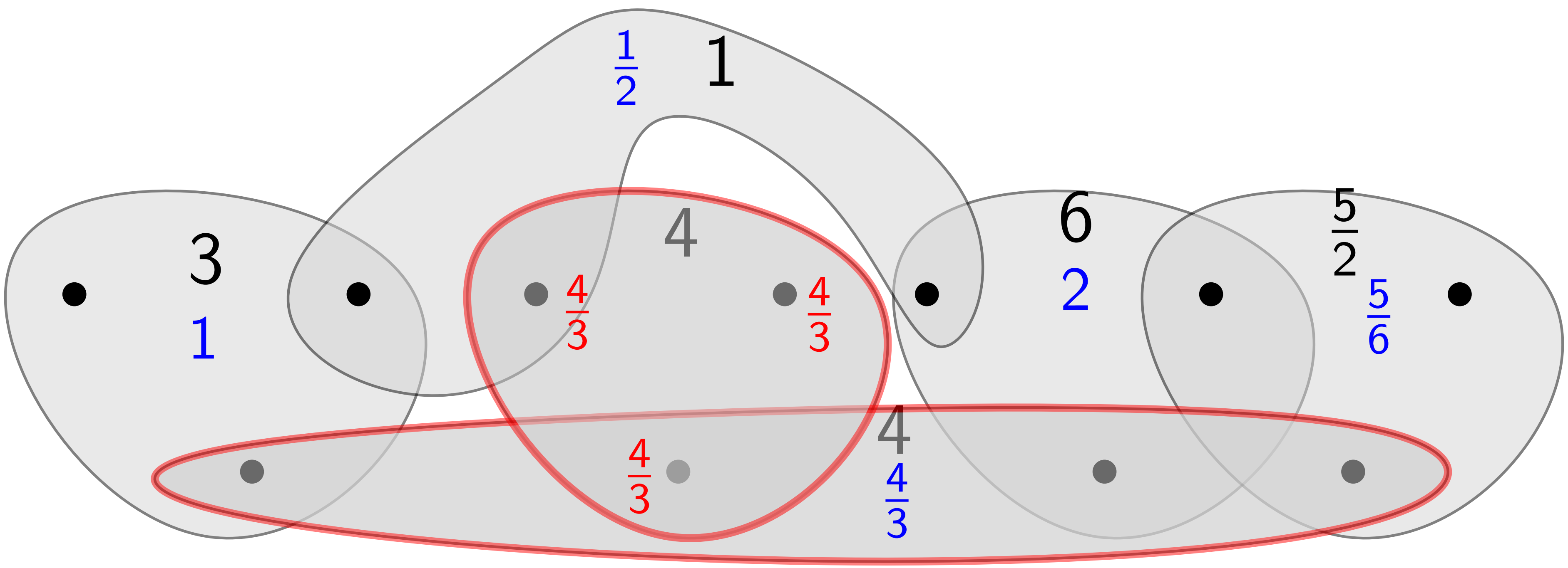
What happens if we “buy” a set?

Fix price of the elements bought and re-evaluate unit prices!



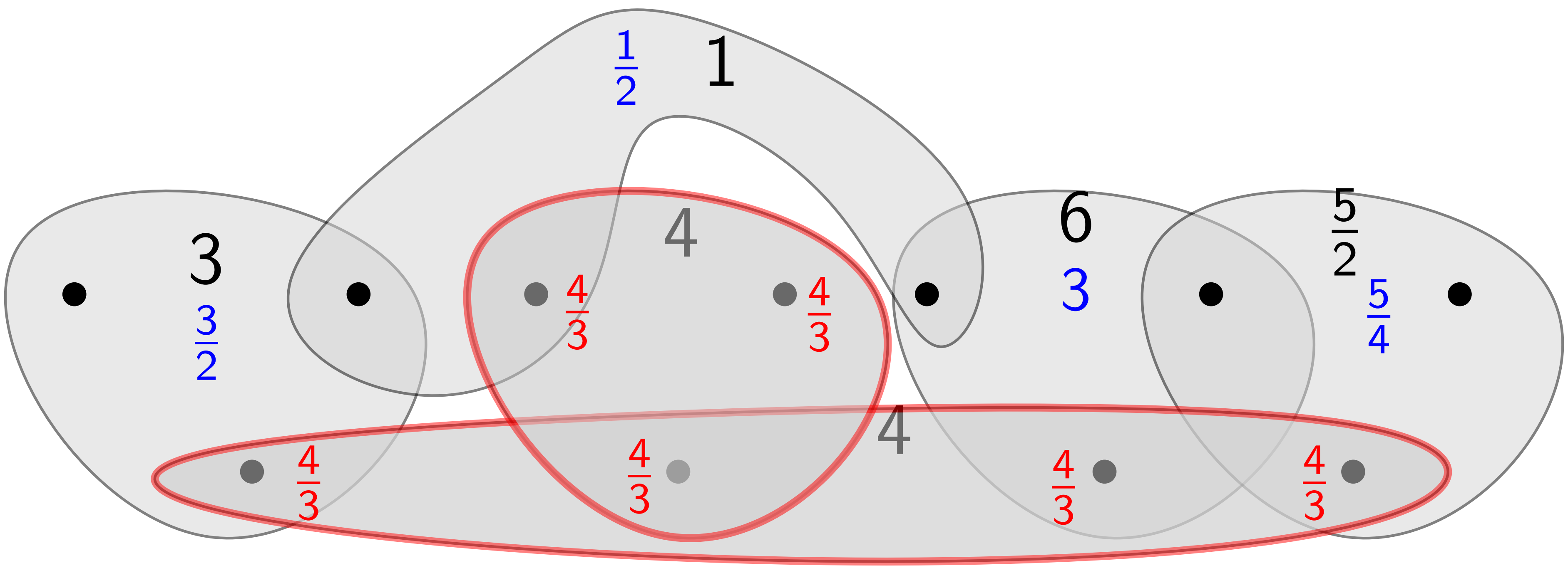
Iterative “re-pricing”

Buy 3 **more** elements for a unit price of $\frac{4}{3}$ and re-price.

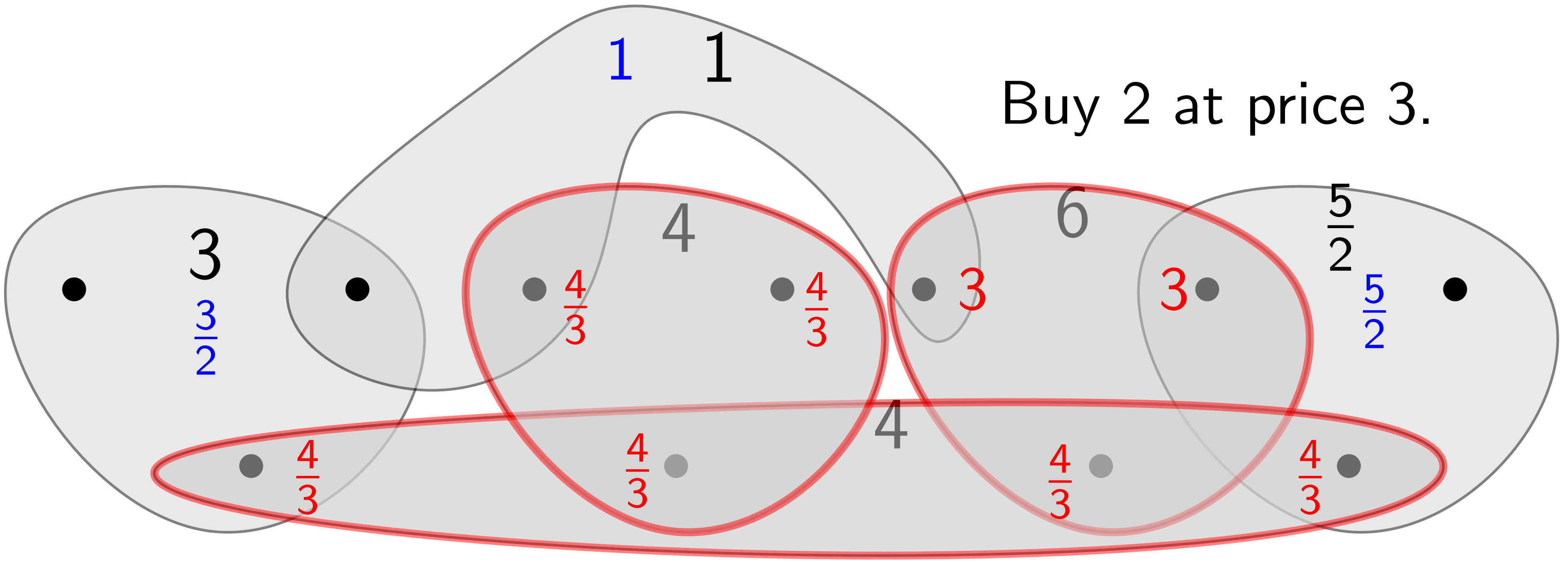


Iterative “re-pricing”

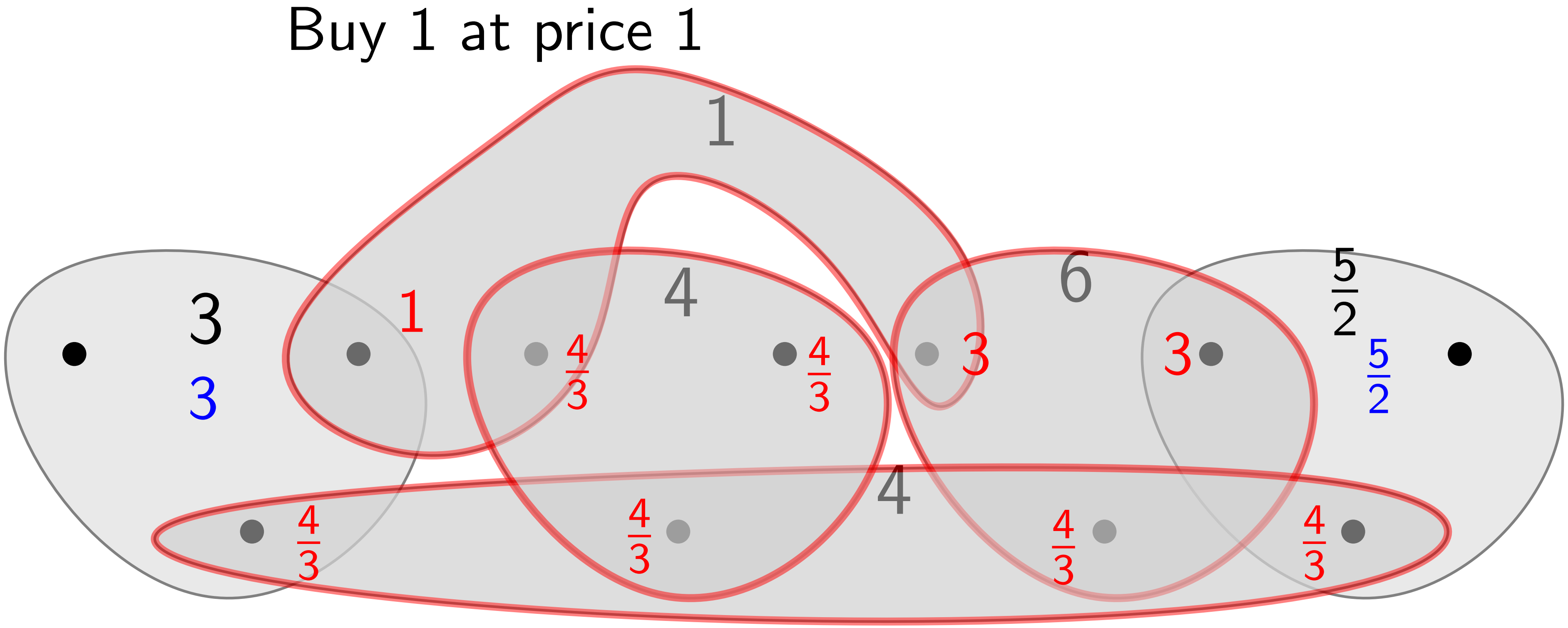
Buy 3 **more** elements for a unit price of $\frac{4}{3}$ and re-price.



Iterative “re-pricing”

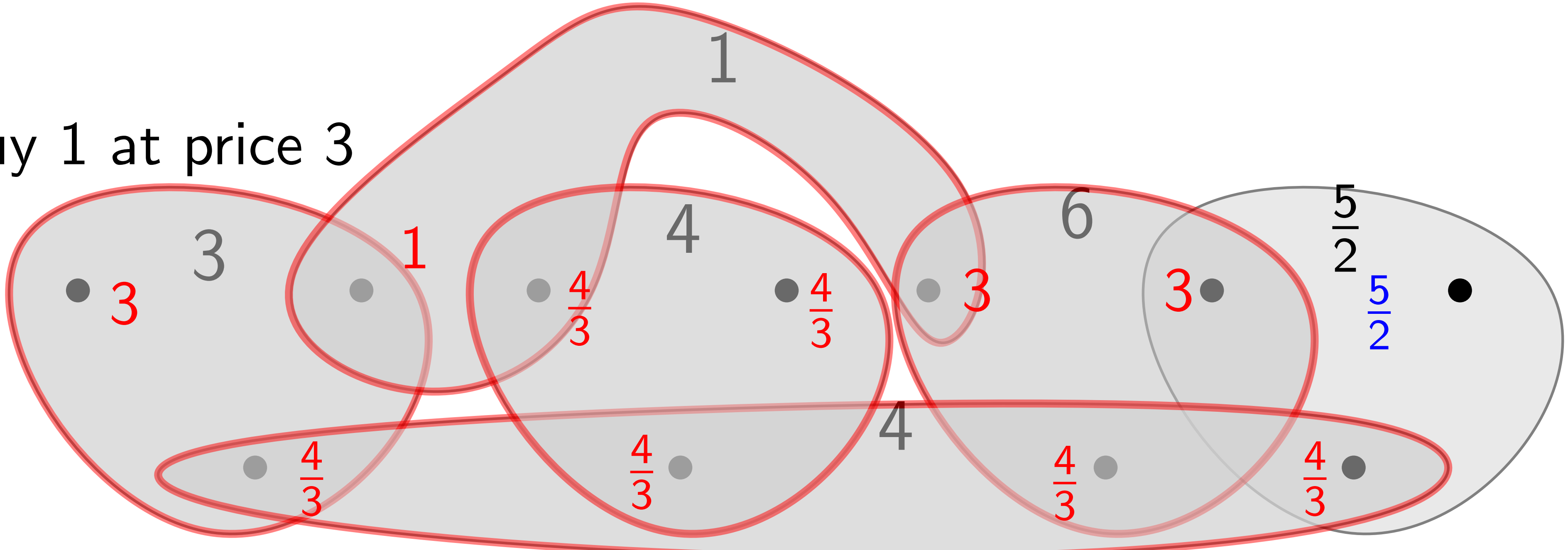


Iterative "re-pricing"

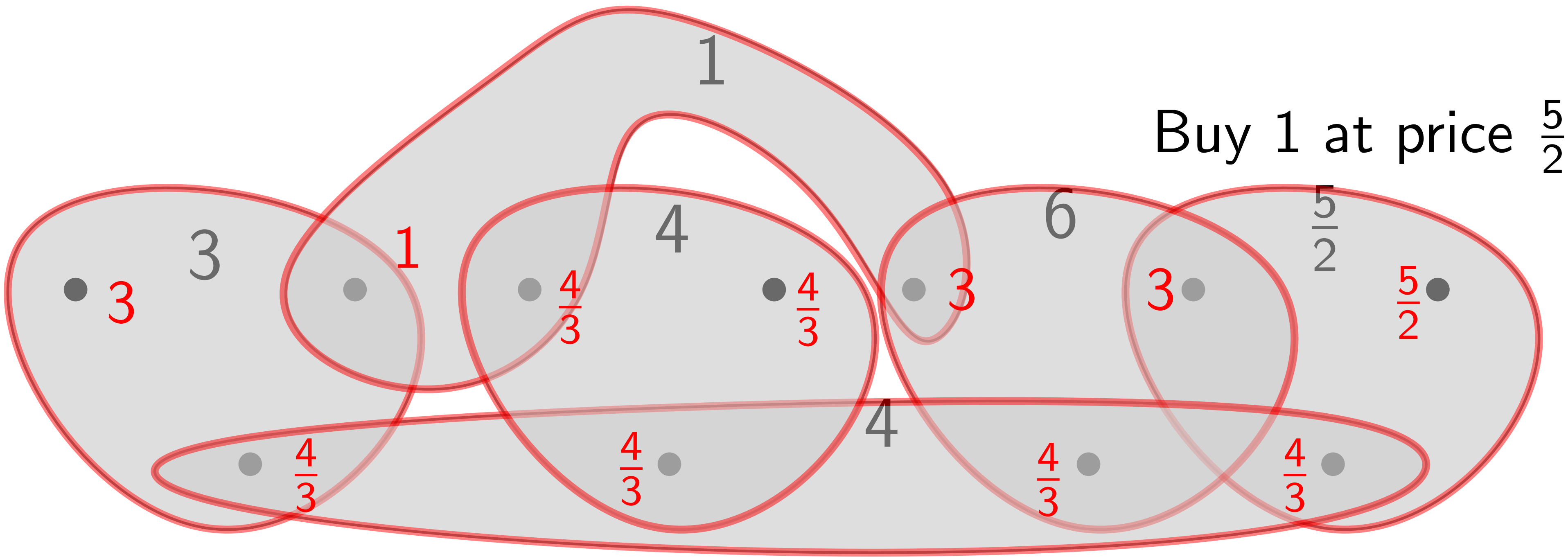


Iterative “re-pricing”

Buy 1 at price 3

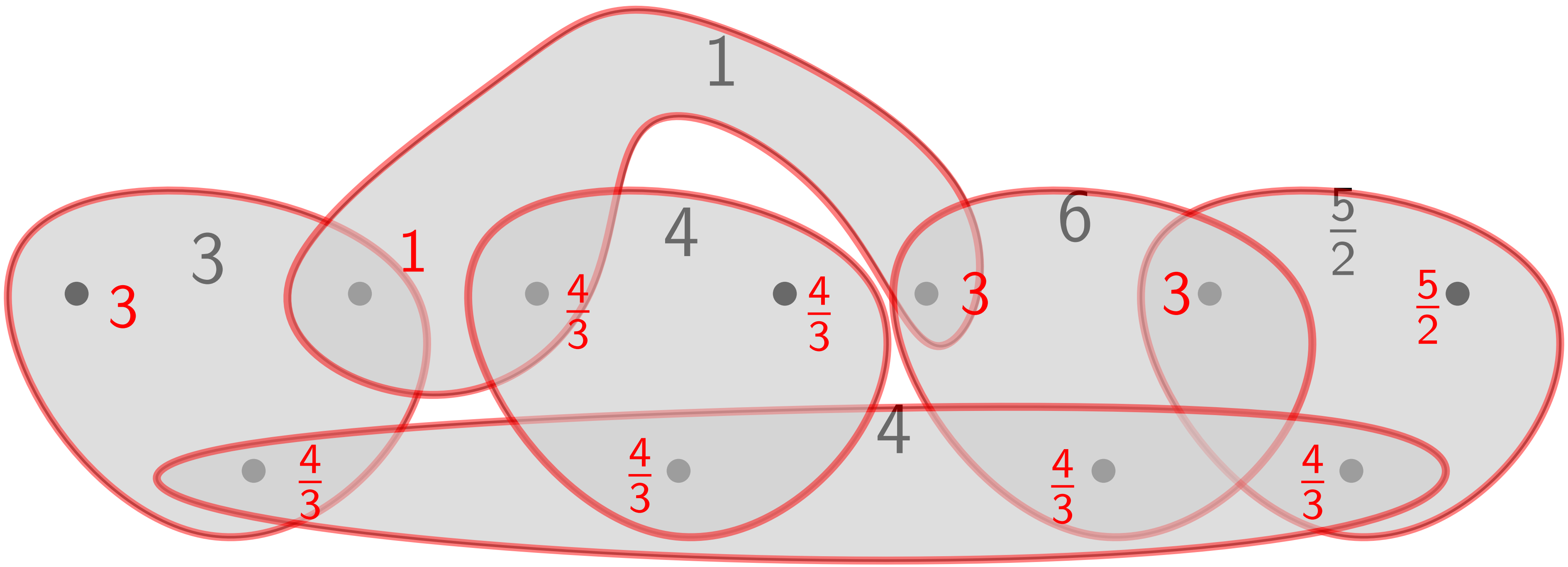


Iterative “re-pricing”



Iterative "re-pricing"

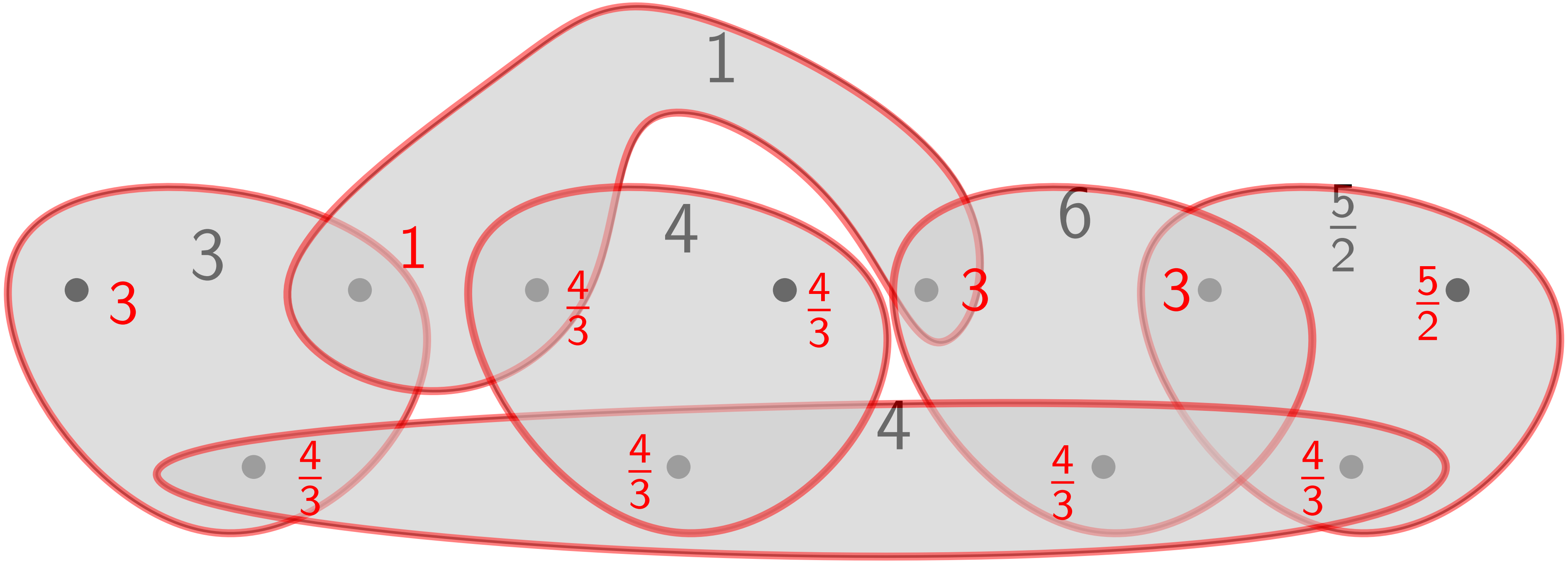
Obs: Total solution cost $= \sum_{e \in U} \text{price}(e)$



Iterative "re-pricing"

Obs: Total solution cost $= \sum_{e \in U} \text{price}(e)$

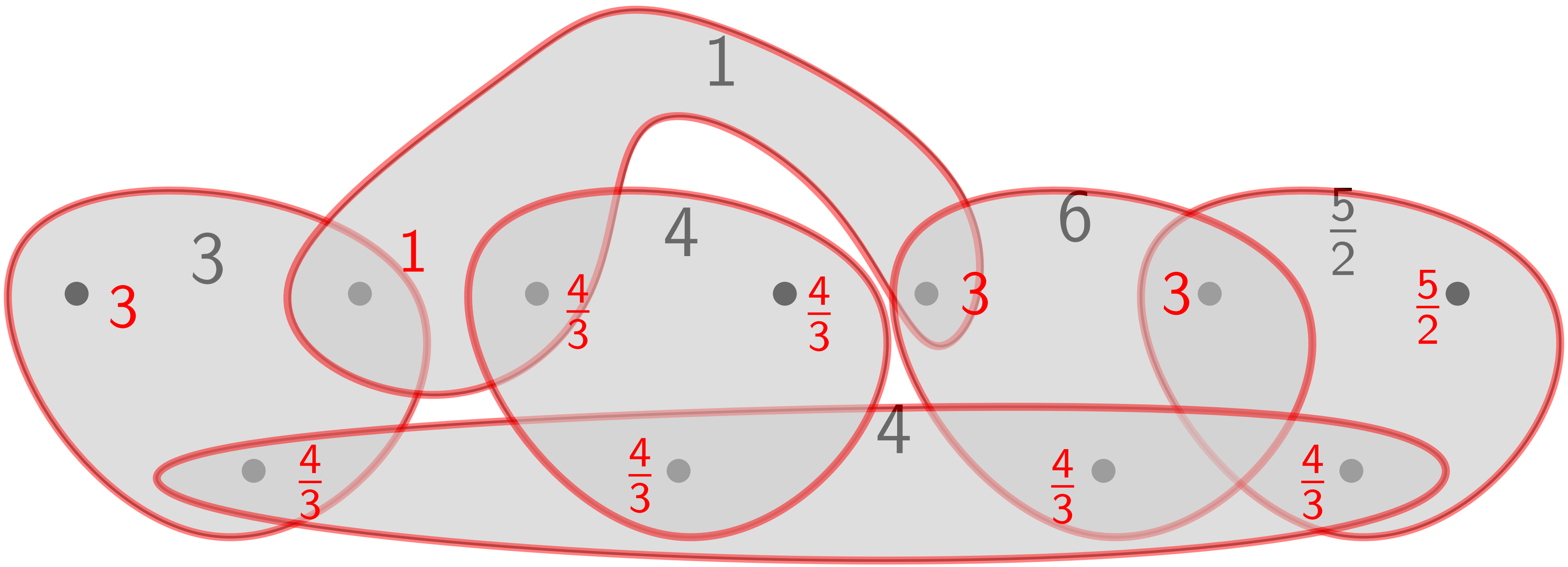
Greedy-Idea: Always choose the set with the cheapest unit price.



Iterative “re-pricing”

Obs: Total solution cost $= \sum_{e \in U} \text{price}(e)$

Greedy-Idea: Always choose the set with the cheapest unit price. Tie-breaking?



Greedy for SetCover

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

```
C ← ∅
S' ← ∅
while C ≠ U do
    S ← Set S from 𝒮, with c(S)/|S\C| minimized
    foreach u ∈ S \ C do
        price(u) ← c(S)/|S\C|
    C ← C ∪ S
    S' ← S' ∪ {S}
return S'                                   // Cover of U
```

$C \leftarrow \emptyset$

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

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

- $S \leftarrow$ Set S from $\mathcal{S}$, with $\frac{c(S)}{|S \setminus C|}$ minimized
- **foreach** $u \in S \setminus C$ **do**
  - price(u) $\leftarrow \frac{c(S)}{|S \setminus C|}$
- $C \leftarrow C \cup S$
- $\mathcal{S}' \leftarrow \mathcal{S}' \cup \{S\}$

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

Analysis

Thm. GreedySetCover is a factor-$\mathcal{H}_k$ approximation alg. where k is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \in O(\log k)$.

Lemma. Let $S \in \mathcal{S}$ and $u_1, \ldots, u_l$ be the elements of S in the order they are covered ("bought") by GreedySetCover. Then

$$\text{price}(u_j) \le \frac{c(S)}{l-j+1}.$$

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

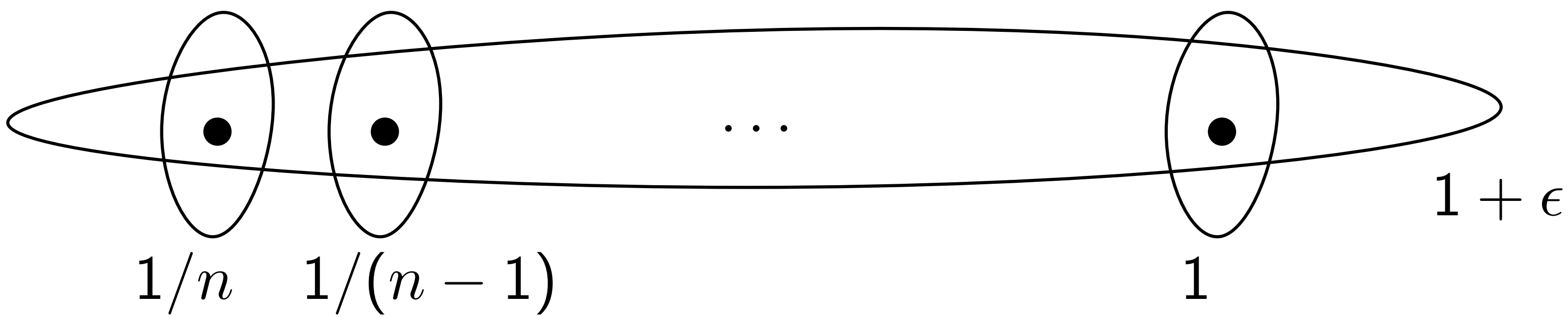
Is the Analysis of Greedy Tight?

Question: Does there exist a set cover instance where our greedy algorithm performs as bad (asymptotically) as our performance guarantee?

Is the Analysis of Greedy Tight?

Question: Does there exist a set cover instance where our greedy algorithm performs as bad (asymptotically) as our performance guarantee?

Yes :-(



ShortestSuperstring (SSS)

Given a collection $\mathcal{S} = \{s_1, \ldots, s_n\}$ of strings over a finite alphabet Σ (i.e., $\mathcal{S} \subseteq \Sigma^+$).
Find a *shortest string* s such that each s_i is a *substring* of s.

ShortestSuperstring (SSS)

Given a collection $\mathcal{S} = \{s_1, \ldots, s_n\}$ of strings over a finite alphabet Σ (i.e., $\mathcal{S} \subseteq \Sigma^+$).
Find a *shortest string* s such that each s_i is a *substring* of s.

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

ShortestSuperstring (SSS)

Given a collection $\mathcal{S} = \{s_1, \ldots, s_n\}$ of strings over a finite alphabet Σ (i.e., $\mathcal{S} \subseteq \Sigma^+$).
Find a *shortest string* s such that each s_i is a *substring* of s.

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

$$s = abcbaa$$

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

ShortestSuperstring (SSS)

Given a collection $\mathcal{S} = \{s_1, \ldots, s_n\}$ of strings over a finite alphabet Σ (i.e., $\mathcal{S} \subseteq \Sigma^+$).
Find a *shortest string* s such that each s_i is a *substring* of s.

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

note: s "covers" the strings in U

$$\begin{array}{l} s = abcbaa \\ abc \\ bcb \\ cbaa \end{array}$$

ShortestSuperstring (SSS)

Given a collection $\mathcal{S} = \{s_1, \ldots, s_n\}$ of strings over a finite alphabet Σ (i.e., $\mathcal{S} \subseteq \Sigma^+$).
Find a *shortest string* s such that each s_i is a *substring* of s.

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

WLOG: No string s_i is a substring of any other string s_j.

note: s "covers" the strings in U

$$\begin{array}{l} s = abcbaa \\ \phantom{s = {}}abc \\ \phantom{s = {}a}bcb \\ \phantom{s = {}ab}cbaa \end{array}$$

SSS as a SetCover problem

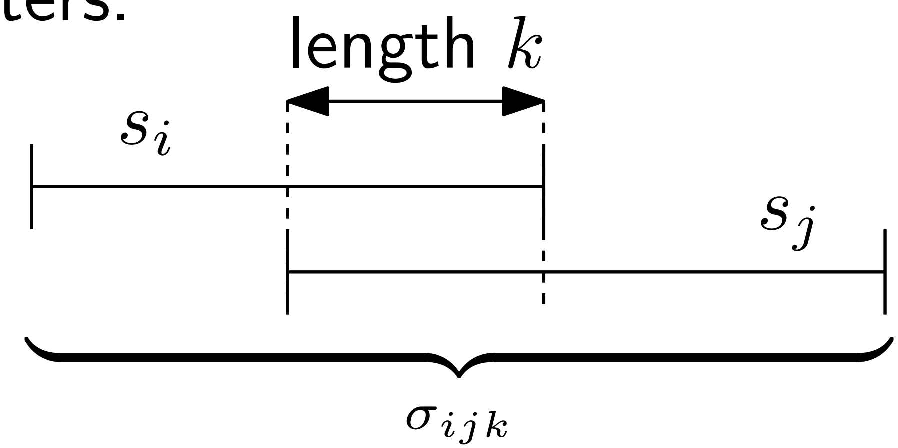
- SetCover Instance: ground set U, set family $\mathcal{S}$, costs c.

SSS as a SetCover problem

- SetCover Instance: ground set U, set family $\mathcal{S}$, costs c.
- ground set $U := \{s_1, \ldots, s_n\}$

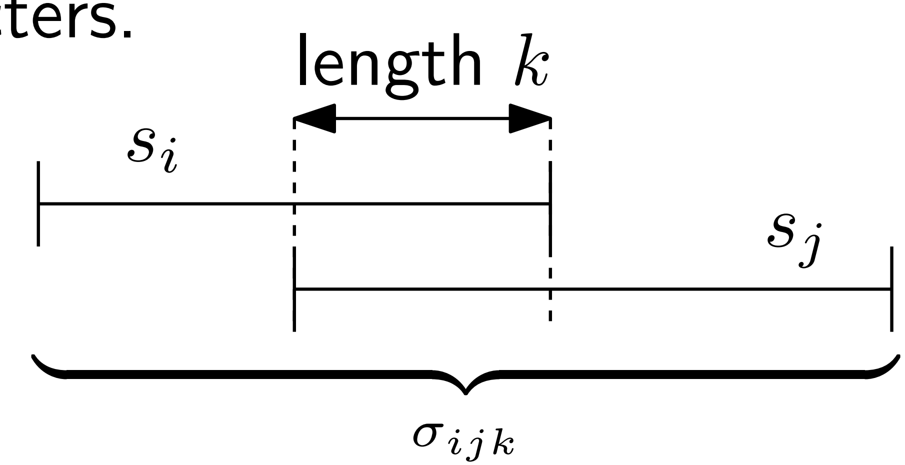
SSS as a SetCover problem

- SetCover Instance: ground set U, set family $\mathcal{S}$, costs c.
- ground set $U := \{s_1, \ldots, s_n\}$
- a string σ_{ijk} has prefix s_i, suffix s_j where s_i and s_j *overlap* on k characters.



SSS as a $\textsc{SetCover}$ problem

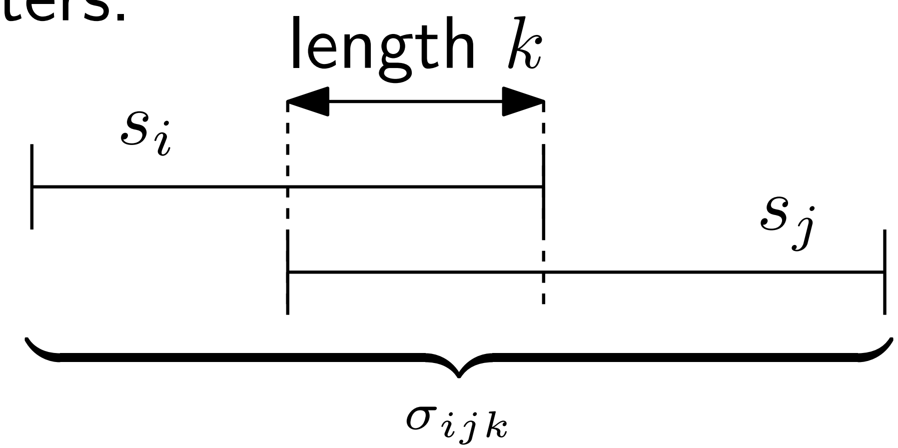
- $\textsc{SetCover}$ Instance: ground set U, set family $\mathcal{S}$, costs c.
- ground set $U := \{s_1, \ldots, s_n\}$
- a string σ_{ijk} has prefix s_i, suffix s_j where s_i and s_j *overlap* on k characters.



- for each σ_{ijk} let $\text{set}(\sigma_{ijk}) = \{s \in U \mid s \text{ substring of } \sigma_{ijk}\}$, i.e., the ground elements covered by σ_{ijk}.

SSS as a SetCover problem

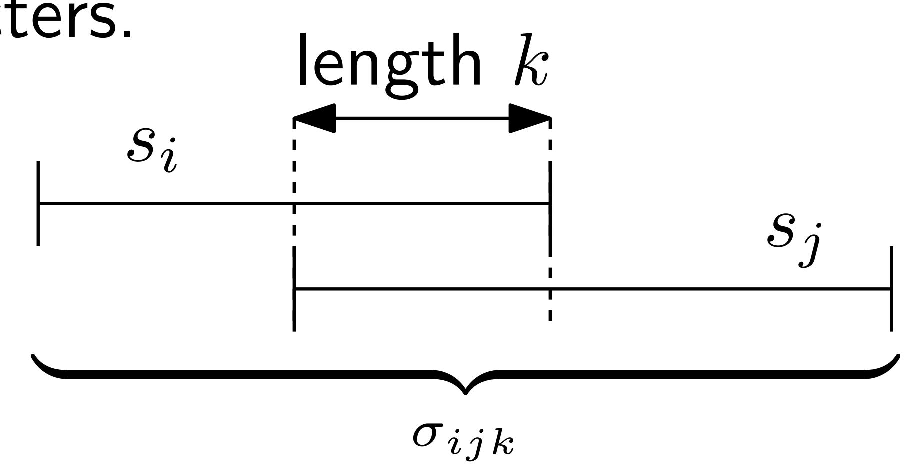
- SetCover Instance: ground set U, set family $\mathcal{S}$, costs c.
- ground set $U := \{s_1, \ldots, s_n\}$
- a string σ_{ijk} has prefix s_i, suffix s_j where s_i and s_j *overlap* on k characters.



- for each σ_{ijk} let $\text{set}(\sigma_{ijk}) = \{s \in U \mid s \text{ substring of } \sigma_{ijk}\}$, i.e., the ground elements covered by σ_{ijk}.
- cost of σ_{ijk} is $|\sigma_{ijk}|$

SSS as a SetCover problem

- SetCover Instance: ground set U, set family $\mathcal{S}$, costs c.
- ground set $U := \{s_1, \ldots, s_n\}$
- a string σ_{ijk} has prefix s_i, suffix s_j where s_i and s_j *overlap* on k characters.



- for each σ_{ijk} let set$(\sigma_{ijk}) = \{s \in U \mid s$ substring of $\sigma_{ijk}\}$, i.e., the ground elements covered by σ_{ijk}.
- cost of σ_{ijk} is $|\sigma_{ijk}|$
- set family $\mathcal{S} = \{$set$(\sigma_{ijk}) \mid$ for valid choices of i, j (possibly $i = j$) and $k > 0\}$.

Relating SSS and SetCover

Lemma. Let OPT be the length of a shortest superstring of U and $\mathrm{OPT}_{\mathrm{SC}}$ be the minimum cost of the corresponding SetCover-Instance. Then:

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

Relating SSS and $\textsc{SetCover}$

Lemma. Let OPT be the length of a shortest superstring of U and $\mathrm{OPT}_{\mathrm{SC}}$ be the minimum cost of the corresponding $\textsc{SetCover}$-Instance. Then:

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

Proof.

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

Relating SSS and SetCover

Lemma. Let OPT be the length of a shortest superstring of U and $\mathrm{OPT}_{\mathsf{SC}}$ be the minimum cost of the corresponding SetCover-Instance. Then:

$$\mathrm{OPT} \leq \mathrm{OPT}_{\mathsf{SC}}$$

Proof.

- Consider an optimal set cover $\mathrm{set}(\pi_1), \ldots, \mathrm{set}(\pi_k)$ of U.
- $s := \pi_1 \circ \cdots \circ \pi_k$ is a superstring of U whose length is $\mathrm{OPT}_{\mathsf{SC}} = \sum_{i=1}^{k} |\pi_i|$

Relating SSS and SetCover

Lemma. Let OPT be the length of a shortest superstring of U and $\mathrm{OPT}_{\mathsf{SC}}$ be the minimum cost of the corresponding SetCover-Instance. Then:

$$\mathrm{OPT} \leq \mathrm{OPT}_{\mathsf{SC}}$$

Proof.

- Consider an optimal set cover $\mathrm{set}(\pi_1), \ldots, \mathrm{set}(\pi_k)$ of U.
- $s := \pi_1 \circ \cdots \circ \pi_k$ is a superstring of U whose length is $\mathrm{OPT}_{\mathsf{SC}} = \sum_{i=1}^{k} |\pi_i|$
- Thus, $\mathrm{OPT} \leq |s| = \mathrm{OPT}_{\mathsf{SC}}$

Relating SSS and SetCover

Lemma. Let OPT be the length of a shortest superstring of U and $\mathrm{OPT}_{\mathrm{SC}}$ be the minimum cost of the corresponding SetCover-Instance. Then:

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

Proof.

- Consider an optimal set cover $\mathrm{set}(\pi_1), \ldots, \mathrm{set}(\pi_k)$ of U.
- $s := \pi_1 \circ \cdots \circ \pi_k$ is a superstring of U whose length is $\mathrm{OPT}_{\mathrm{SC}} = \sum_{i=1}^{k} |\pi_i|$
- Thus, $\mathrm{OPT} \leq |s| = \mathrm{OPT}_{\mathrm{SC}}$

$\square$

Relating SSS and SetCover

Lemma.

$$\mathsf{OPT}_{\mathsf{SC}} \leq 2 \cdot \mathsf{OPT}$$

Relating SSS and SetCover

Lemma.

$$\mathsf{OPT}_{\mathsf{SC}} \leq 2 \cdot \mathsf{OPT}$$

Proof.

Let s be an optimal superstring. Construct set cover with:

$$\text{cost} \leq 2|s| = 2 \cdot \mathsf{OPT}.$$

Relating SSS and SetCover

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

Proof.
Let s be an optimal superstring. Construct set cover with:

$$\text{cost} \leq 2|s| = 2 \cdot \mathrm{OPT}.$$

s

s_{b_1}

leftmost occurence of some string $s_{b_1} \in U$

Relating SSS and SetCover

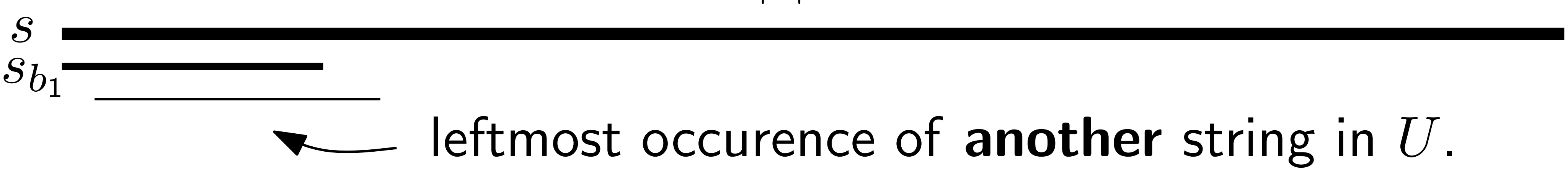
Lemma.

$$\mathsf{OPT}_{\mathsf{SC}} \leq 2 \cdot \mathsf{OPT}$$

Proof.

Let s be an optimal superstring. Construct set cover with:

$$\text{cost} \leq 2|s| = 2 \cdot \mathsf{OPT}.$$



Relating SSS and SetCover

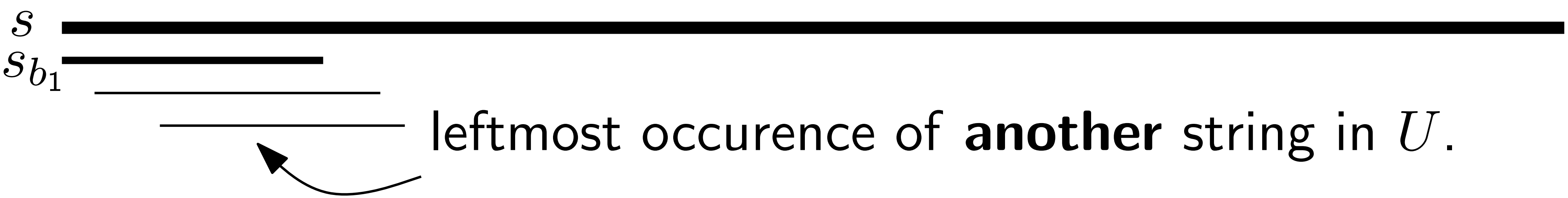
Lemma.

$$\mathsf{OPT}_{\mathsf{SC}} \leq 2 \cdot \mathsf{OPT}$$

Proof.

Let s be an optimal superstring. Construct set cover with:

$$\text{cost} \leq 2|s| = 2 \cdot \mathsf{OPT}.$$



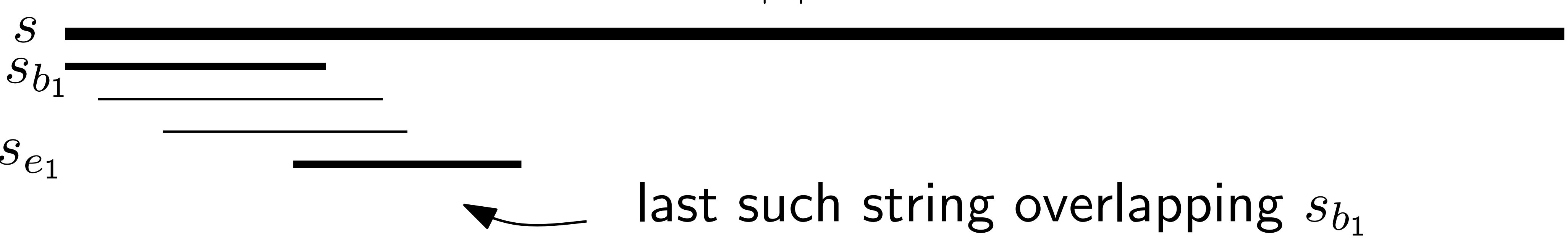
Relating SSS and SetCover

Lemma. $\mathsf{OPT}_{\mathsf{SC}} \le 2 \cdot \mathsf{OPT}$

Proof.

Let s be an optimal superstring. Construct set cover with:

$$\text{cost} \le 2|s| = 2 \cdot \mathsf{OPT}.$$



Relating SSS and SetCover

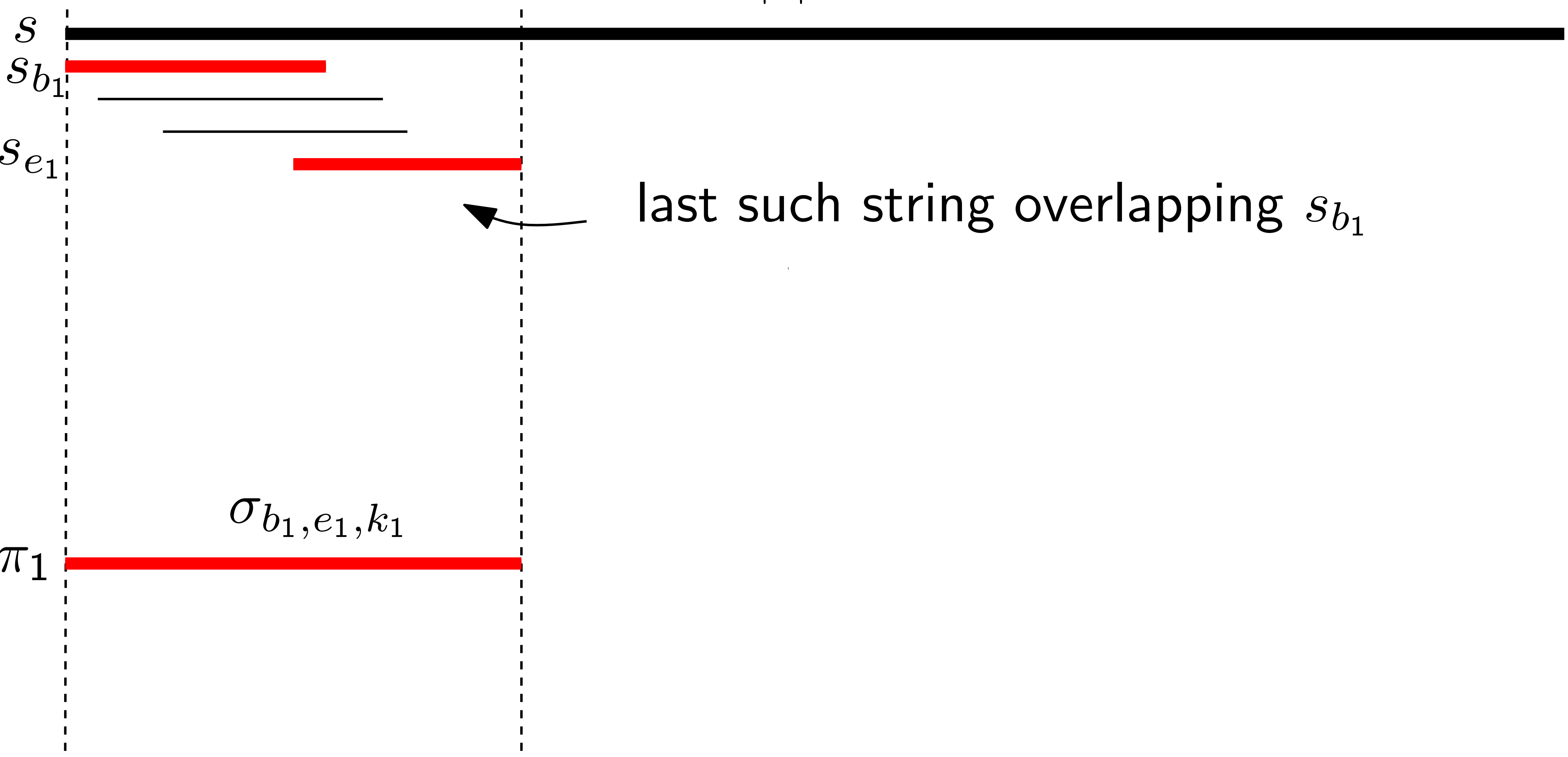
Lemma.

$$\mathsf{OPT}_{\mathsf{SC}} \leq 2 \cdot \mathsf{OPT}$$

Proof.

Let s be an optimal superstring. Construct set cover with:

$$\text{cost} \leq 2|s| = 2 \cdot \mathsf{OPT}.$$

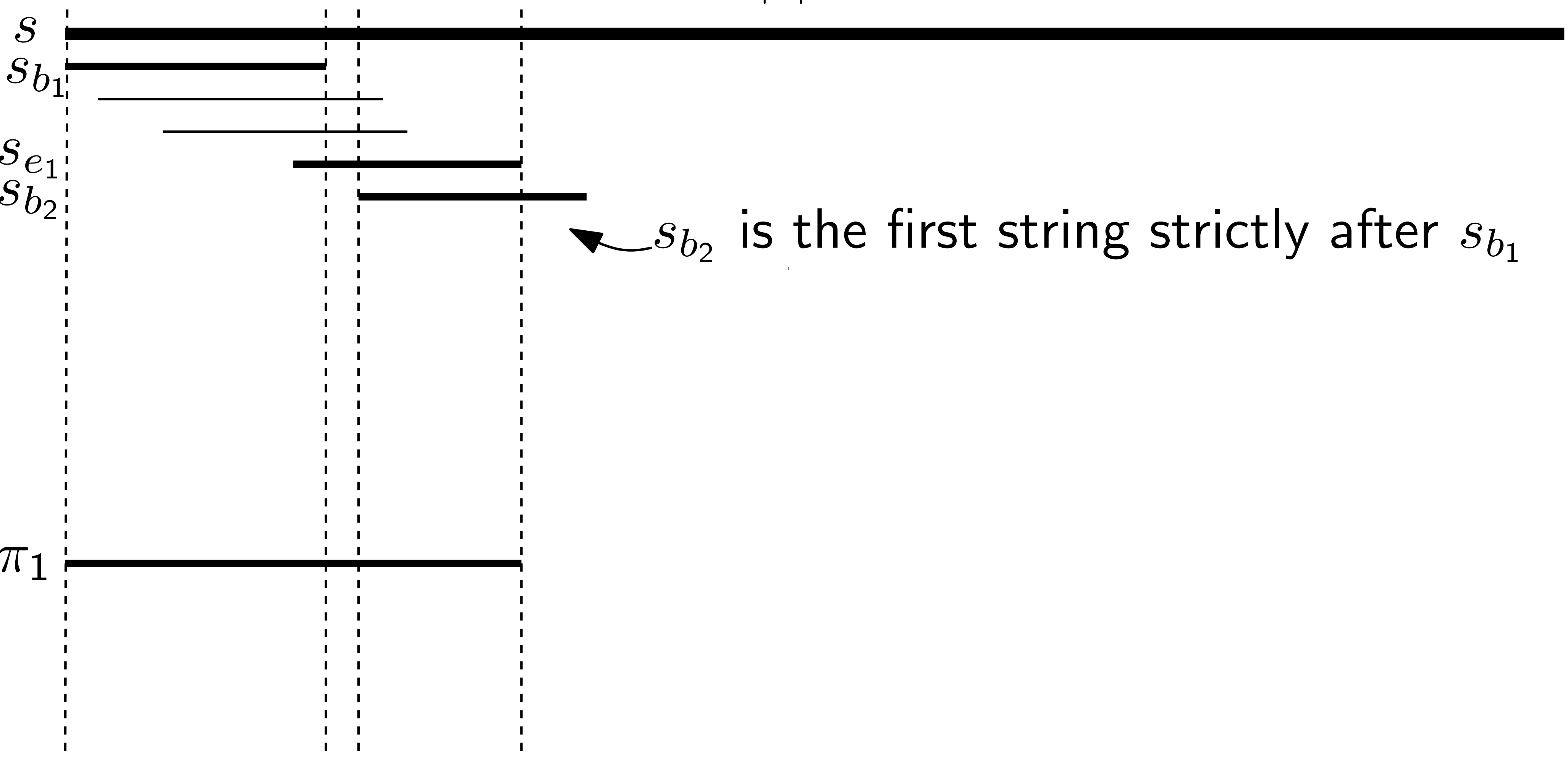


Relating SSS and SetCover

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

Proof.
Let s be an optimal superstring. Construct set cover with:

$$\text{cost} \leq 2|s| = 2 \cdot \mathsf{OPT}.$$



Relating SSS and SetCover

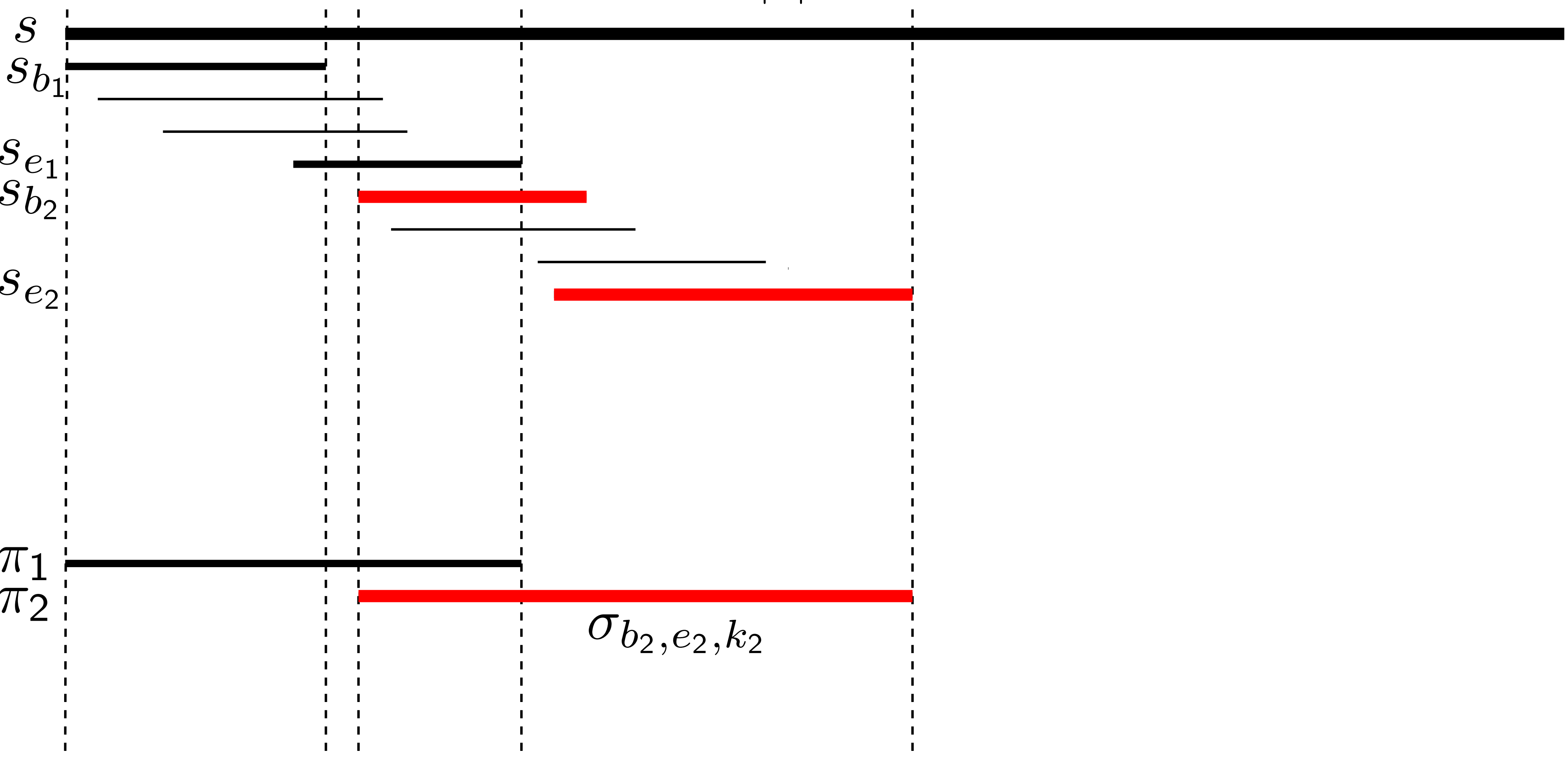
Lemma.

$$\mathsf{OPT}_{\mathsf{SC}} \leq 2 \cdot \mathsf{OPT}$$

Proof.

Let s be an optimal superstring. Construct set cover with:

$$\text{cost} \leq 2|s| = 2 \cdot \mathsf{OPT}.$$



Relating SSS and SetCover

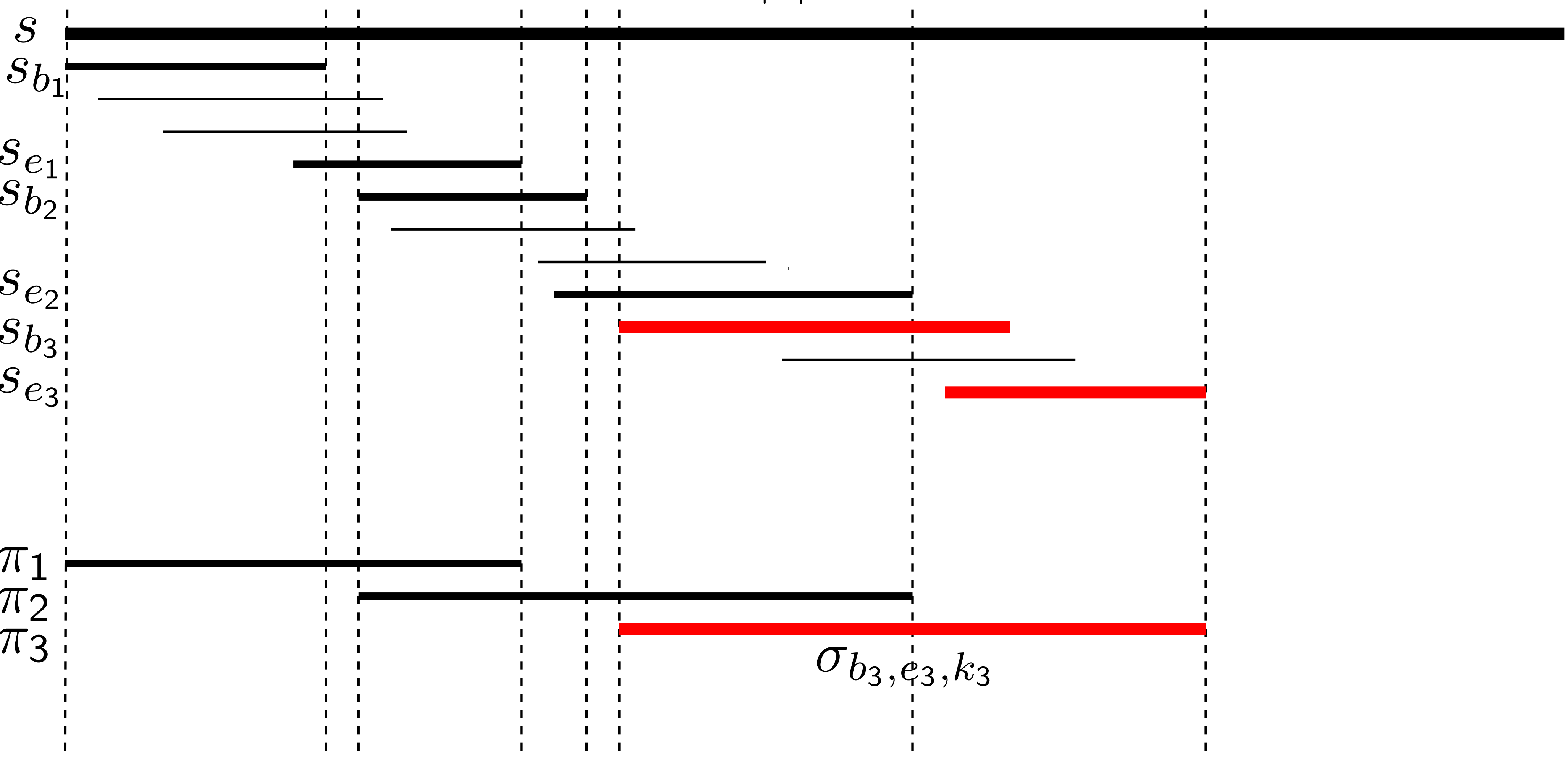
Lemma.

$$\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}$$

Proof.

Let s be an optimal superstring. Construct set cover with:

$$\text{cost} \leq 2|s| = 2 \cdot \mathrm{OPT}.$$



Relating SSS and SetCover

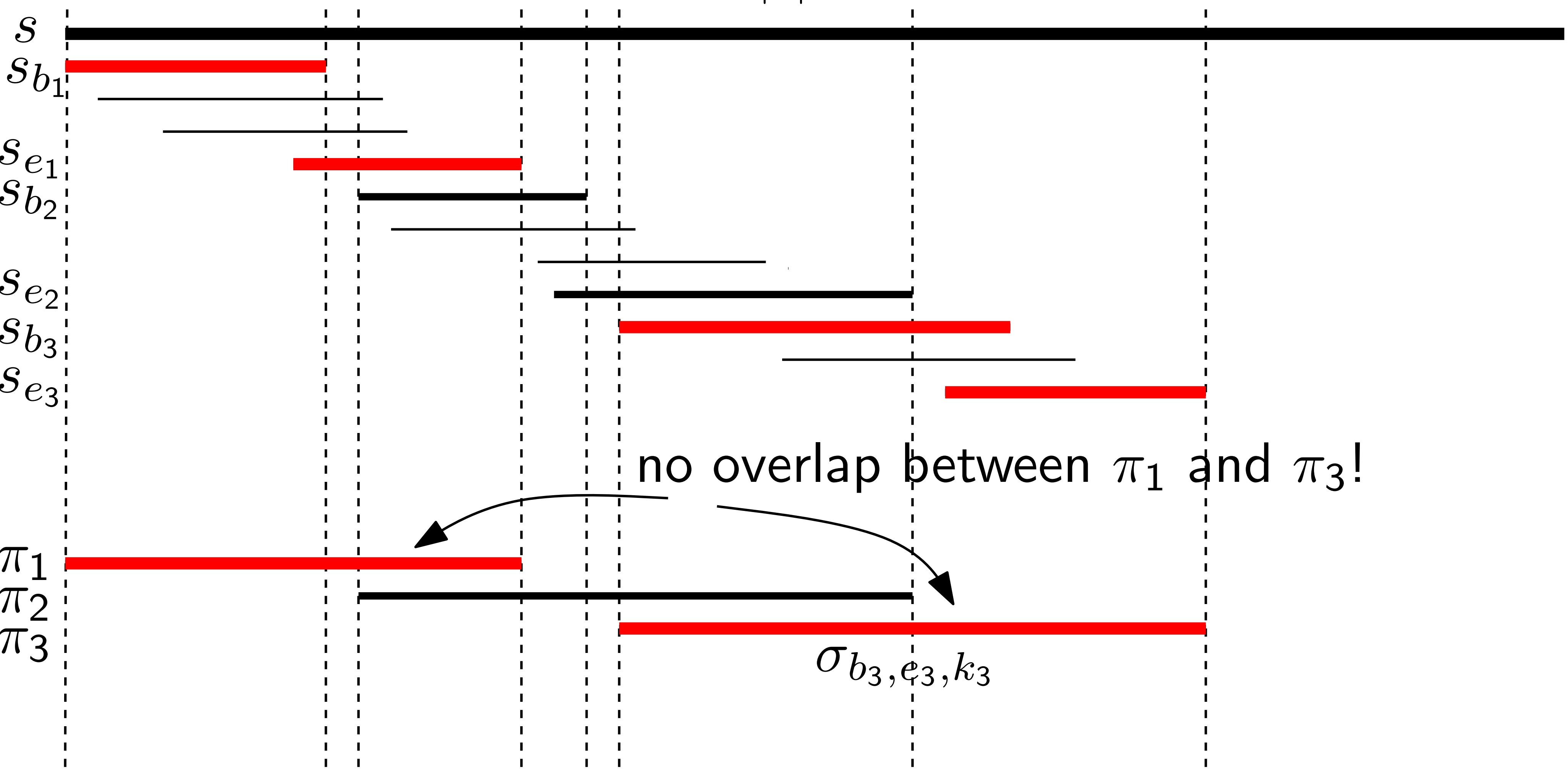
Lemma.

$$\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}$$

Proof.

Let s be an optimal superstring. Construct set cover with:

cost $\leq 2|s| = 2 \cdot \mathrm{OPT}$.



Relating SSS and SetCover

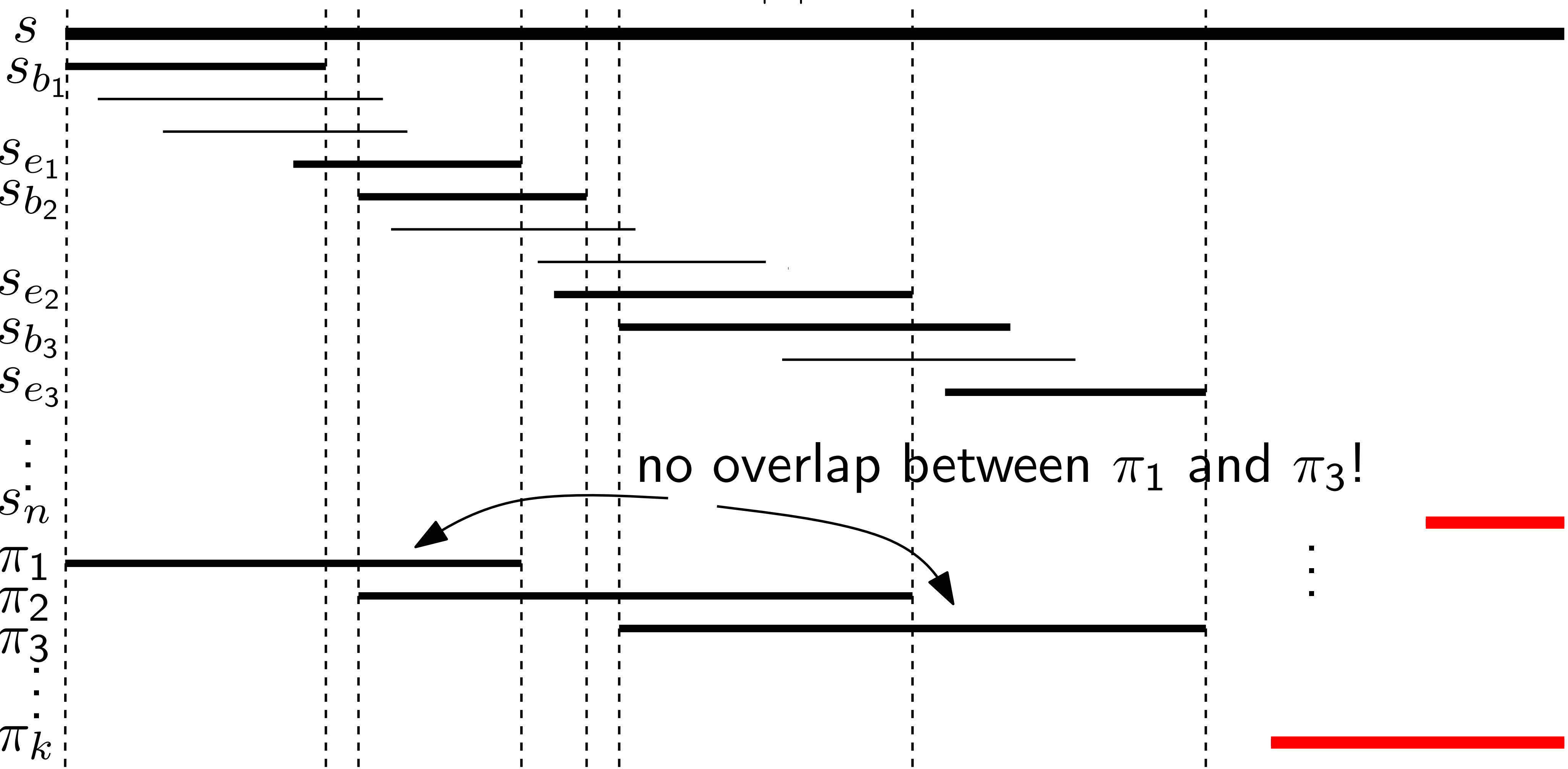
Lemma.

$$\text{OPT}_{\text{SC}} \leq 2 \cdot \text{OPT}$$

Proof.

Let s be an optimal superstring. Construct set cover with:

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



Relating SSS and SetCover

Lemma.

$$\mathsf{OPT}_{\mathsf{SC}} \leq 2 \cdot \mathsf{OPT}$$

Proof.

- each string $s_i \in U$ is a substring of some π_j

Relating SSS and $\textsc{SetCover}$

Lemma.

$$\mathsf{OPT}_{\mathsf{SC}} \leq 2 \cdot \mathsf{OPT}$$

Proof.

- each string $s_i \in U$ is a substring of some π_j
- $\mathsf{set}(\pi_1), \ldots, \mathsf{set}(\pi_k)$ is a solution for the $\textsc{SetCover}$ instance with cost $\sum_i |\pi_i|$

Relating SSS and SetCover

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

Proof.

- each string $s_i \in U$ is a substring of some π_j
- set$(\pi_1), \ldots,$ set(π_k) is a solution for the SetCover instance with cost $\sum_i |\pi_i|$
- the substrings π_j, π_{j+2} of s do not overlap

Relating SSS and SetCover

Lemma.

$$\text{OPT}_{\text{SC}} \leq 2 \cdot \text{OPT}$$

Proof.

- each string $s_i \in U$ is a substring of some π_j
- $\text{set}(\pi_1), \ldots, \text{set}(\pi_k)$ is a solution for the SetCover instance with cost $\sum_i |\pi_i|$
- the substrings π_j, π_{j+2} of s do not overlap
- each character of s lies in at most two substrings (i.e., π_j and π_{j+1})

Relating SSS and $\textsc{SetCover}$

Lemma.

$$\mathsf{OPT}_{\mathsf{SC}} \leq 2 \cdot \mathsf{OPT}$$

Proof.

- each string $s_i \in U$ is a substring of some π_j
- $\mathsf{set}(\pi_1), \ldots, \mathsf{set}(\pi_k)$ is a solution for the $\textsc{SetCover}$ instance with cost $\sum_i |\pi_i|$
- the substrings π_j, π_{j+2} of s do not overlap
- each character of s lies in at most two substrings (i.e., π_j and π_{j+1})
- $\sum_i |\pi_i| \leq 2|s| = 2 \cdot \mathsf{OPT}$

Relating SSS and SetCover

Lemma.

$$\mathsf{OPT}_{\mathsf{SC}} \leq 2 \cdot \mathsf{OPT}$$

Proof.

- each string $s_i \in U$ is a substring of some π_j
- $\mathrm{set}(\pi_1), \ldots, \mathrm{set}(\pi_k)$ is a solution for the SetCover instance with cost $\sum_i |\pi_i|$
- the substrings π_j, π_{j+2} of s do not overlap
- each character of s lies in at most two substrings (i.e., π_j and π_{j+1})
- $\sum_i |\pi_i| \leq 2|s| = 2 \cdot \mathsf{OPT}$

$\square$

Algorithm for SSS

- construct the $\textsc{SetCover}$ instance U, $\mathcal{S}$, c
- Let set$(\pi_1), \ldots,$ set(π_k) be the solution provided by the GreedySetCover algorithm.
- return $\pi_1 \circ \cdots \circ \pi_k$ as the superstring.

Algorithm for SSS

- construct the $\textsc{SetCover}$ instance U, $\mathcal{S}$, c
- Let $\mathsf{set}(\pi_1), \ldots, \mathsf{set}(\pi_k)$ be the solution provided by the GreedySetCover algorithm.
- return $\pi_1 \circ \cdots \circ \pi_k$ as the superstring.

Thm. This algorithm for $\textsc{ShortestSuperstring}$ has approximation factor $2\mathcal{H}_n$.

Algorithm for SSS

- construct the $\textsc{SetCover}$ instance U, $\mathcal{S}$, c
- Let set$(\pi_1), \ldots,$ set(π_k) be the solution provided by the GreedySetCover algorithm.
- return $\pi_1 \circ \cdots \circ \pi_k$ as the superstring.

Thm. This algorithm for $\textsc{ShortestSuperstring}$ has approximation factor $2\mathcal{H}_n$.

Proof.

The cost of the solution is bounded as follows:

$$\mathcal{H}_n \cdot \mathrm{OPT}_{\mathsf{SC}} \leq 2 \cdot \mathcal{H}_n \cdot \mathrm{OPT}$$

Algorithm for SSS

- construct the $\textsc{SetCover}$ instance U, $\mathcal{S}$, c
- Let set$(\pi_1), \ldots,$ set(π_k) be the solution provided by the GreedySetCover algorithm.
- return $\pi_1 \circ \cdots \circ \pi_k$ as the superstring.

Thm. This algorithm for $\textsc{ShortestSuperstring}$ has approximation factor $2\mathcal{H}_n$.

Proof.

The cost of the solution is bounded as follows:

$$\mathcal{H}_n \cdot \mathrm{OPT}_{\mathsf{SC}} \leq 2 \cdot \mathcal{H}_n \cdot \mathrm{OPT}$$

$\square$

Algorithm for SSS

- construct the $\textsc{SetCover}$ instance U, $\mathcal{S}$, c
- Let set$(\pi_1), \ldots,$ set(π_k) be the solution provided by the GreedySetCover algorithm.
- return $\pi_1 \circ \cdots \circ \pi_k$ as the superstring.

Thm. This algorithm for $\textsc{ShortestSuperstring}$ has approximation factor $2\mathcal{H}_n$.

Proof.

The cost of the solution is bounded as follows:

$$\mathcal{H}_n \cdot \mathsf{OPT}_{\mathsf{SC}} \leq 2 \cdot \mathcal{H}_n \cdot \mathsf{OPT}$$

$\square$

Note: SSS has a factor-3 approximation alg. see [V. §7].

Algorithm for SSS

- construct the SETCOVER instance U, $\mathcal{S}$, c
- Let set$(\pi_1), \ldots,$ set(π_k) be the solution provided by the GreedySetCover algorithm.
- return $\pi_1 \circ \cdots \circ \pi_k$ as the superstring.

Thm. This algorithm for SHORTESTSUPERSTRING has approximation factor $2\mathcal{H}_n$.

Proof.

The cost of the solution is bounded as follows:

$$\mathcal{H}_n \cdot \mathsf{OPT}_{\mathsf{SC}} \leq 2 \cdot \mathcal{H}_n \cdot \mathsf{OPT}$$

$\square$

Note: SSS has a factor-3 approximation alg. see [V. §7].

Next Week: Steiner Trees & Multiway-Cuts

Approximation Preserving Reduction

Let Π_1, Π_2 be minimization problems. An **approximation preserving reduction** from Π_1 to Π_2 is a pair (f, g) of poly-time computable functions with the following properties.

(i) for each instance I_1 of Π_1, $I_2 := f(I_1)$ is an instance of Π_2 where $\mathsf{OPT}_{\Pi_2}(I_2) \leq \mathsf{OPT}_{\Pi_1}(I_1)$

(ii) for each feasible solution t of I_2, $s := g(I_1, t)$ is a feasible solution of I_1 where $\mathsf{obj}_{\Pi_1}(I_1, s) \leq \mathsf{obj}_{\Pi_2}(I_2, t)$

$$
\begin{tikzcd}
 & \Pi_1 & \Pi_2 \\
\text{instances} & I_1 \arrow[r, "f"] \arrow[d, dashed] & I_2 \arrow[d] \\
\text{solutions} & s & t \arrow[l, "g"']
\end{tikzcd}
$$

Approximation Preserving Reduction

Thm. Let Π_1, Π_2 be minimization problems where there is an approximation preserving reduction from Π_1 to Π_2. Then, for each factor-α approximation algorithm of Π_2, there is a factor-α approximation algorithm of Π_1.

$$
\begin{tikzcd}
 & \Pi_1 & & \Pi_2 \\
\text{instances} & I_1 \arrow[rr, "f"] \arrow[d, dashed, "\alpha\text{-Approximation}"'] & & I_2 \arrow[d, "\alpha\text{-Approximation}"] \\
\text{solutions} & s & & t \arrow[ll, "g"']
\end{tikzcd}
$$

Approximation Preserving Reduction

Thm. Let Π_1, Π_2 be minimization problems where there is an approximation preserving reduction from Π_1 to Π_2. Then, for each factor-α approximation algorithm of Π_2, there is a factor-α approximation algorithm of Π_1.

Proof.

- Consider a factor-α approx. alg. A of Π_2 and an instance I_1 of Π_1.
- Let $I_2 := f(I_1)$, $t := A(I_2)$ and $s := g(I_1, t)$
- $\mathsf{obj}_{\Pi_1}(I_1, s) \le \mathsf{obj}_{\Pi_2}(I_2, t) \le \alpha \cdot \mathsf{OPT}_{\Pi_2}(I_2) \le \alpha \cdot \mathsf{OPT}_{\Pi_1}(I_1)$

Approximation Preserving Reduction

Thm. Let Π_1, Π_2 be minimization problems where there is an approximation preserving reduction from Π_1 to Π_2. Then, for each factor-α approximation algorithm of Π_2, there is a factor-α approximation algorithm of Π_1.

Proof.

- Consider a factor-α approx. alg. A of Π_2 and an instance I_1 of Π_1.
- Let $I_2 := f(I_1)$, $t := A(I_2)$ and $s := g(I_1, t)$
- $\mathrm{obj}_{\Pi_1}(I_1, s) \leq \mathrm{obj}_{\Pi_2}(I_2, t) \leq \alpha \cdot \mathrm{OPT}_{\Pi_2}(I_2) \leq \alpha \cdot \mathrm{OPT}_{\Pi_1}(I_1)$

$\square$

Approximation Preserving Reduction

Thm. Let Π_1, Π_2 be minimization problems where there is an approximation preserving reduction from Π_1 to Π_2. Then, for each factor-α approximation algorithm of Π_2, there is a factor-α approximation algorithm of Π_1.

Proof.

- Consider a factor-α approx. alg. A of Π_2 and an instance I_1 of Π_1.
- Let $I_2 := f(I_1)$, $t := A(I_2)$ and $s := g(I_1, t)$
- $\mathrm{obj}_{\Pi_1}(I_1, s) \leq \mathrm{obj}_{\Pi_2}(I_2, t) \leq \alpha \cdot \mathrm{OPT}_{\Pi_2}(I_2) \leq \alpha \cdot \mathrm{OPT}_{\Pi_1}(I_1)$

$\square$