Algorithms for Decision Support

Online Algorithms (3/3)

Problem lower bound and optimal online algorithms

Outline

- Problem lower bound and "best" online algorithms
  - Ski-rental
  - Bin packing
  - Paging

- Bounding difference to the optimal solution — potential function
  - List accessing
  - k-server

Outline

- Problem lower bound and “best” online algorithms
  - Ski-rental
  - Bin packing
  - Paging

- Bounding difference to the optimal solution — potential function
  - List accessing
  - k-server

Competitive Ratios

- An algorithm ALG is c-competitive if

$$\text{for all instance } I,\ \frac{\text{ALG}(I)}{\text{OPT}(I)} \leq c \text{ (minimization)}$$

- Show that ALG is at most c-competitive (upper bound):

  Claim that for any I, $\text{ALG}(I) \leq x$ and $\text{OPT}(I) \geq y$, hence, $\frac{\text{ALG}(I)}{\text{OPT}(I)} \leq \frac{x}{y} \leq c$

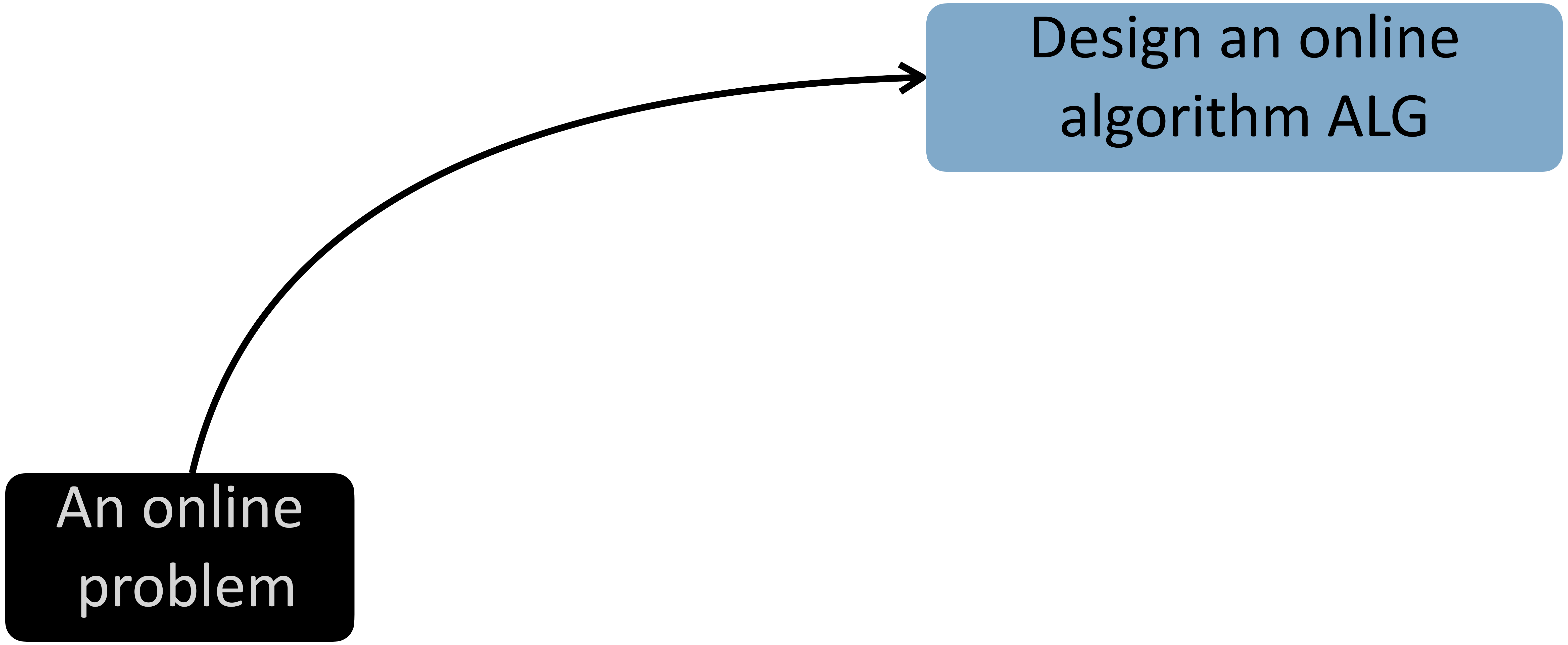
- Show that ALG is at least d-competitive (lower bound):

  Find an instance I' such that $\frac{\text{ALG}(I')}{\text{OPT}(I')} \geq d$

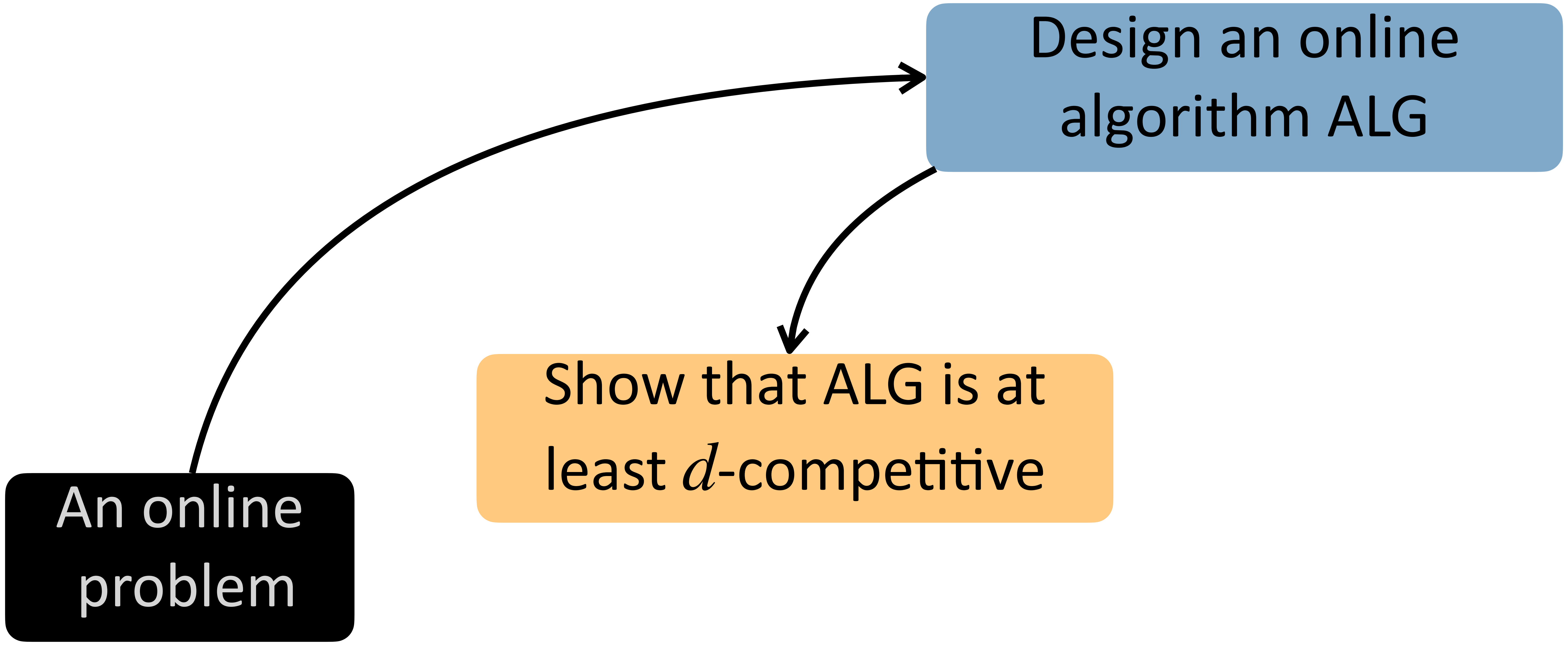
Recap: Online Optimization

An online
problem

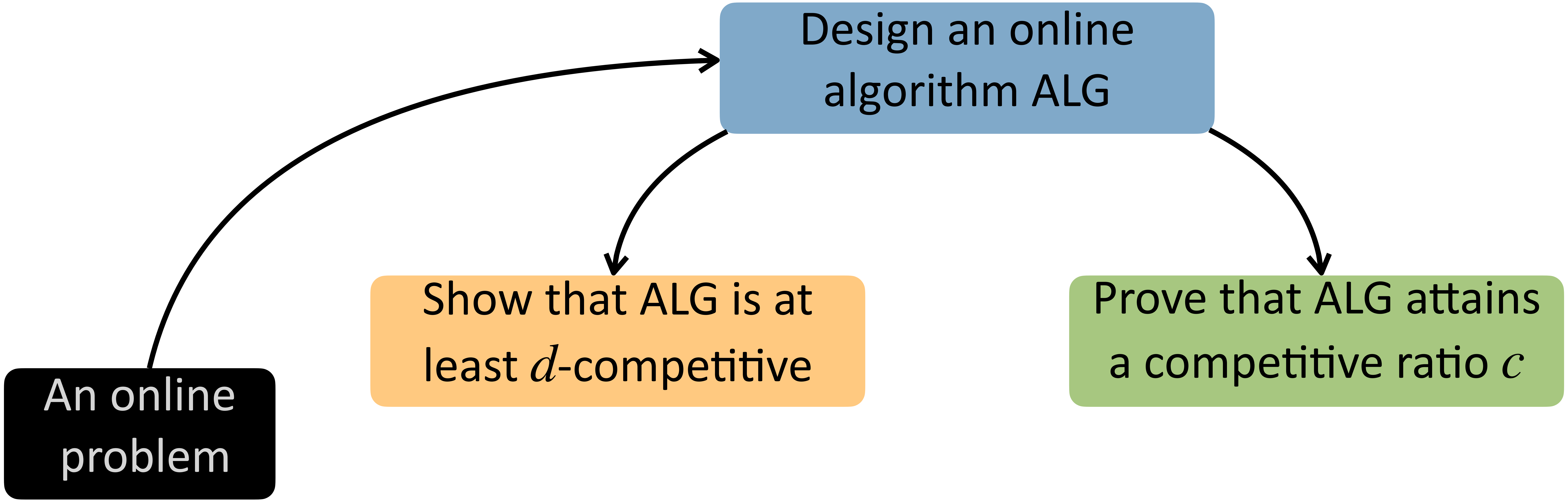
Recap: Online Optimization



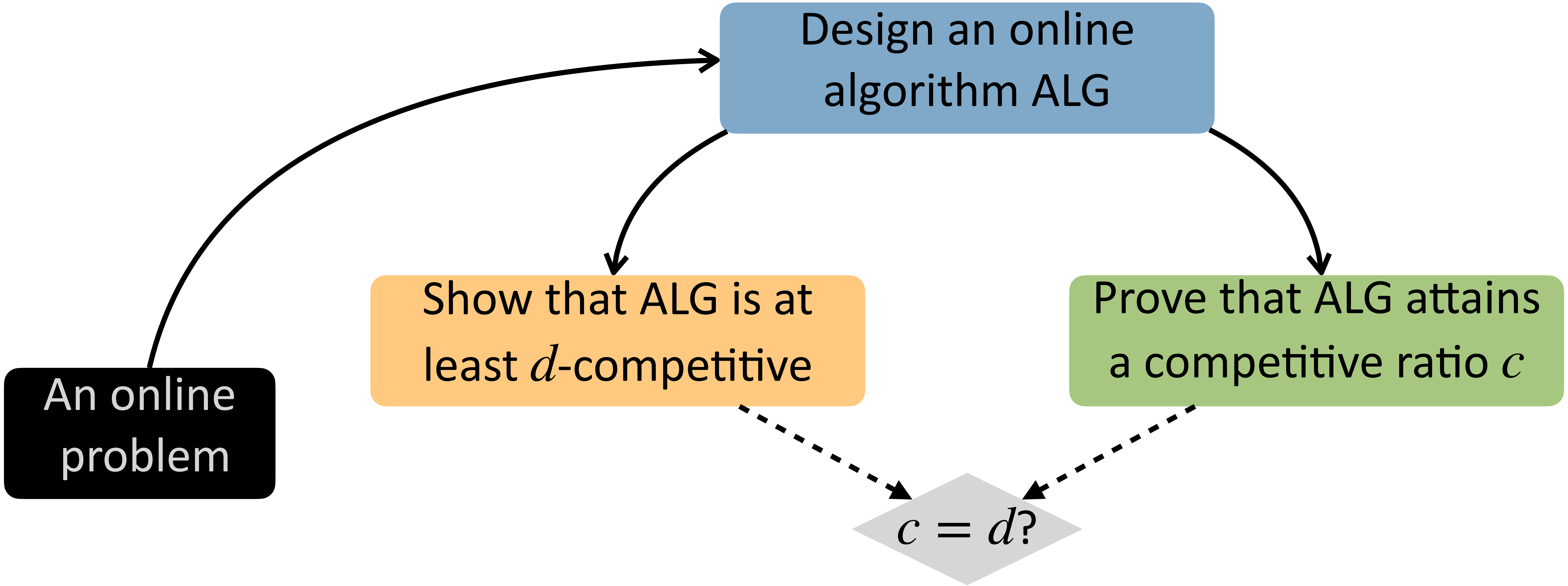
Recap: Online Optimization



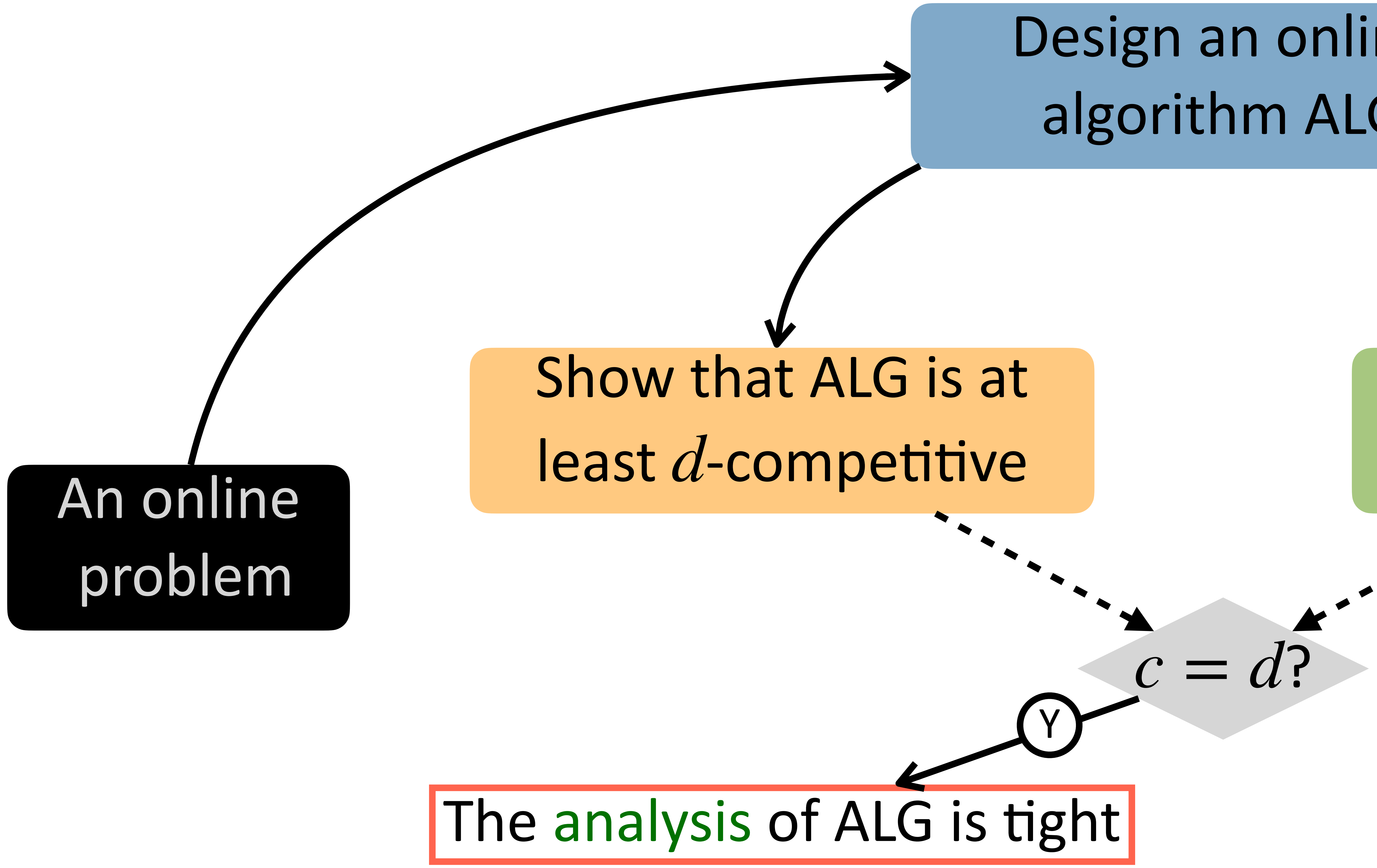
Recap: Online Optimization



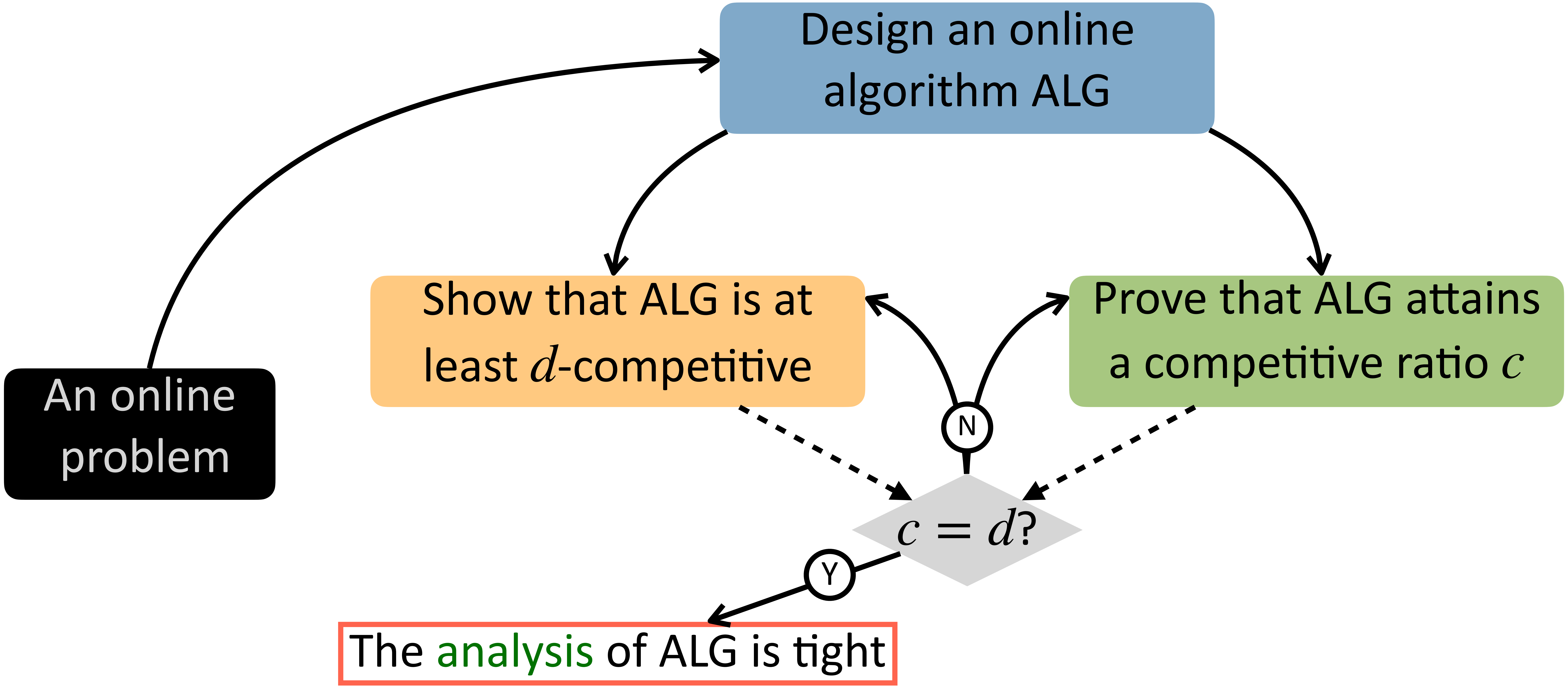
Recap: Online Optimization



Recap: Online Optimization

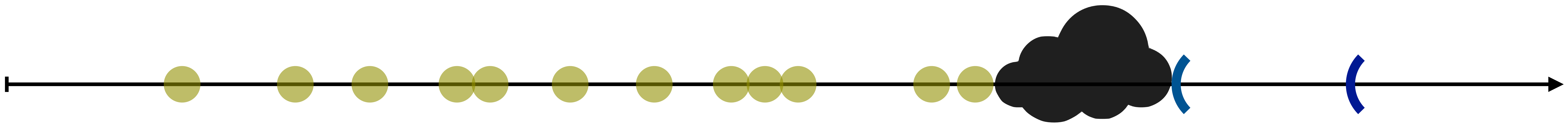


Recap: Online Optimization



Problem Competitive Ratio Lower Bound

- Recall that for any algorithm, we can prove that its competitive ratio has a lower bound (by designing an adversarial input against it)



Problem Competitive Ratio Lower Bound

- Recall that for any algorithm, we can prove that its competitive ratio has a lower bound (by designing an adversarial input against it)

- By designing adversarial instances, one can prove that for a problem, there is a performance lower bound L for all online algorithm. That is, **any (deterministic) online algorithm is at least L-competitive**.

Problem Competitive Ratio Lower Bound

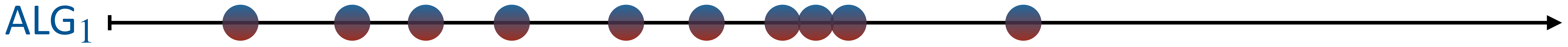
- Formally, we prove that for any online algorithm ALG_i, there exists an instance I_i such that

$$\frac{\mathsf{ALG}_i(I_i)}{\mathsf{OPT}(I_i)} \geq L$$

Problem Competitive Ratio Lower Bound

- Formally, we prove that for any online algorithm ALG$_i$, there exists an instance I_i such that

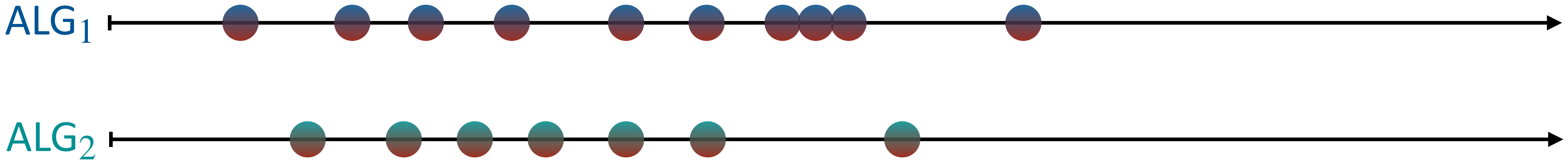
$$\frac{\text{ALG}_i(I_i)}{\text{OPT}(I_i)} \geq L$$



Problem Competitive Ratio Lower Bound

- Formally, we prove that for any online algorithm ALG_i, there exists an instance I_i such that

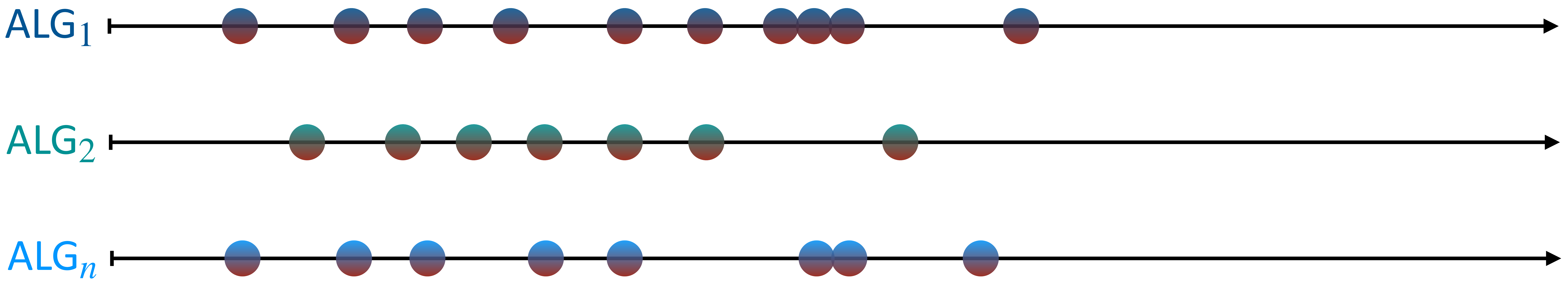
$$\frac{\text{ALG}_i(I_i)}{\text{OPT}(I_i)} \geq L$$



Problem Competitive Ratio Lower Bound

- Formally, we prove that for any online algorithm ALG_i, there exists an instance I_i such that

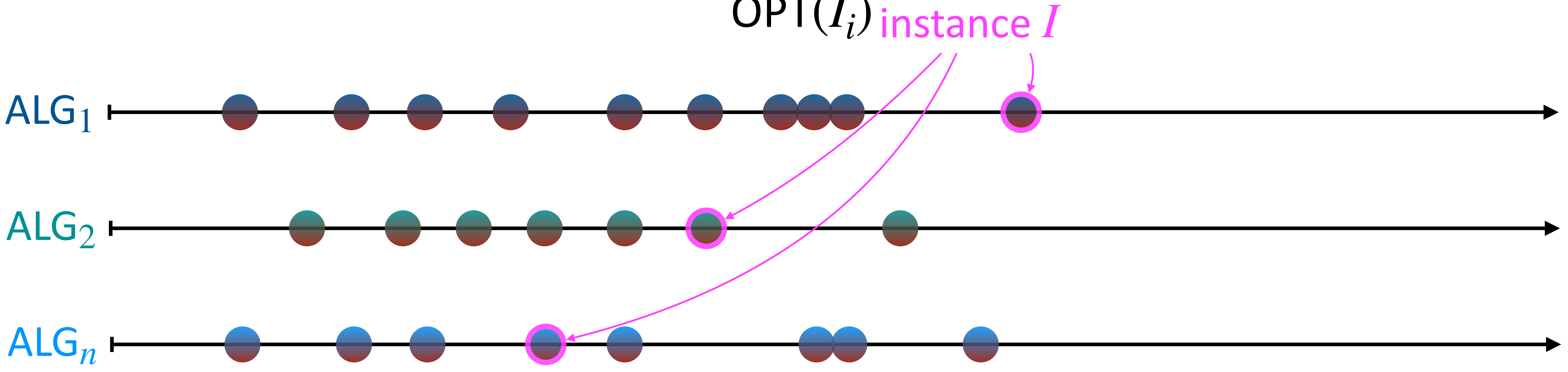
$$\frac{\text{ALG}_i(I_i)}{\text{OPT}(I_i)} \geq L$$



Problem Competitive Ratio Lower Bound

- Formally, we prove that for any online algorithm ALG_i, there exists an instance I_i such that

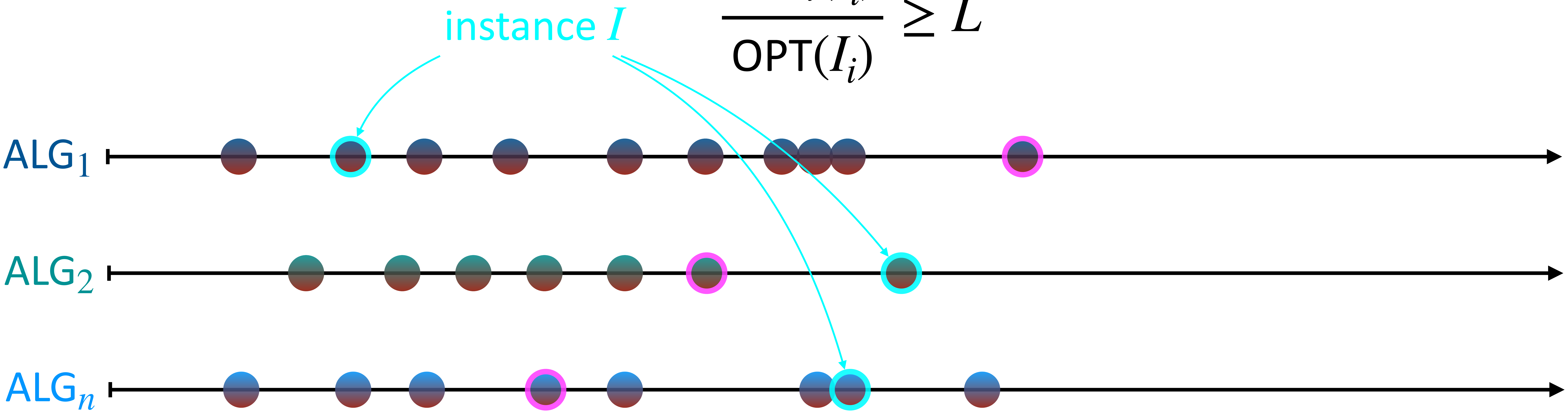
$$\frac{\text{ALG}_i(I_i)}{\text{OPT}(I_i)} \geq L$$



Problem Competitive Ratio Lower Bound

- Formally, we prove that for any online algorithm ALG_i, there exists an instance I_i such that

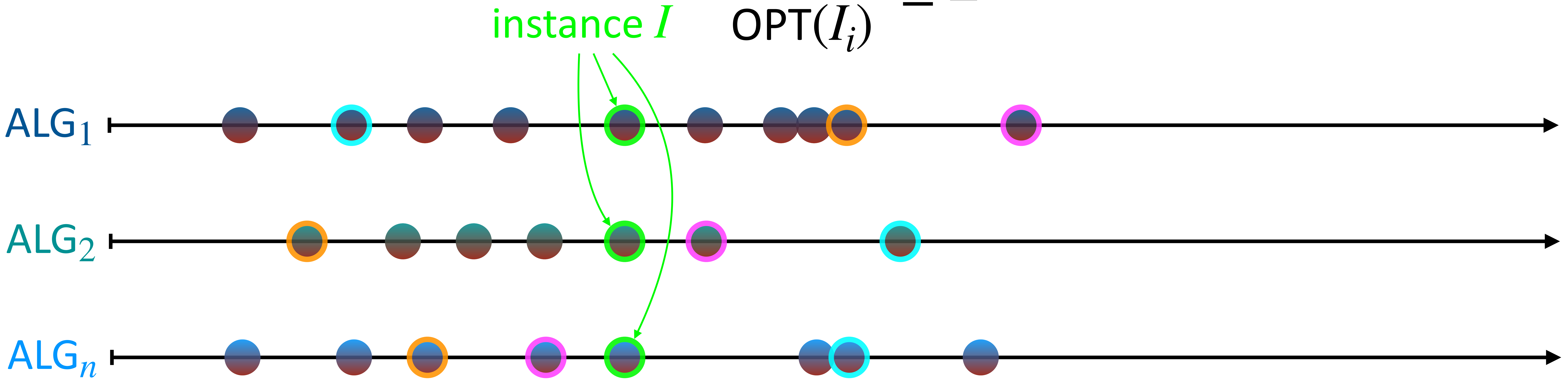
$$\frac{\mathrm{ALG}_i(I_i)}{\mathrm{OPT}(I_i)} \geq L$$



Problem Competitive Ratio Lower Bound

- Formally, we prove that for any online algorithm ALG_i, there exists an instance I_i such that

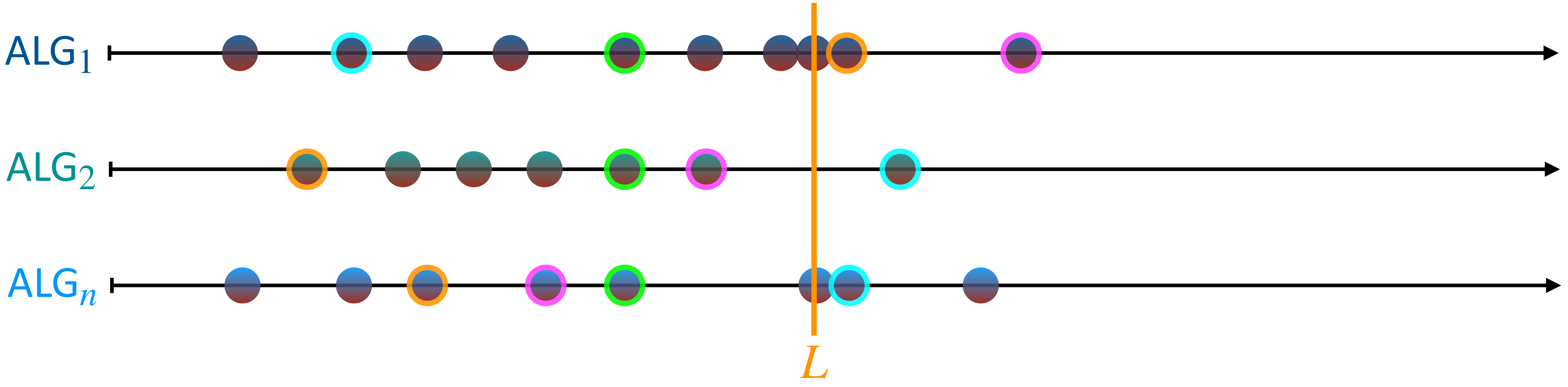
$$\frac{\text{ALG}_i(I_i)}{\text{OPT}(I_i)} \geq L$$



Problem Competitive Ratio Lower Bound

- Formally, we prove that for any online algorithm ALG_i, there exists an instance I_i such that

$$\frac{\text{ALG}_i(I_i)}{\text{OPT}(I_i)} \geq L$$



Problem Competitive Ratio Lower Bound

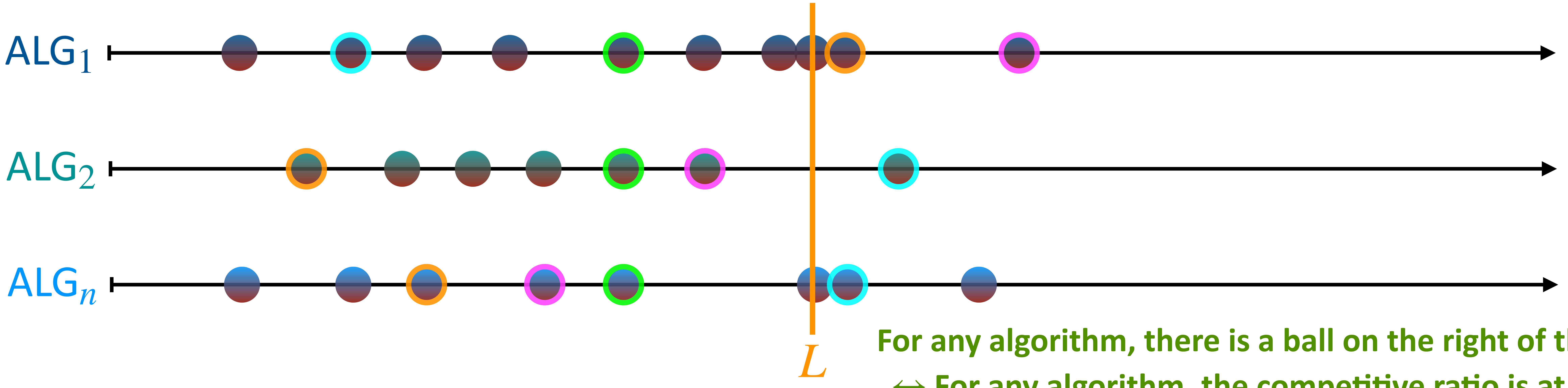
- Formally, we prove that for any online algorithm ALG$_i$, there exists an instance I_i such that

$$\frac{\text{ALG}_i(I_i)}{\text{OPT}(I_i)} \geq L$$

Problem Competitive Ratio Lower Bound

- Formally, we prove that for any online algorithm ALG_i, there exists an instance I_i such that

$$\frac{\text{ALG}_i(I_i)}{\text{OPT}(I_i)} \geq L$$



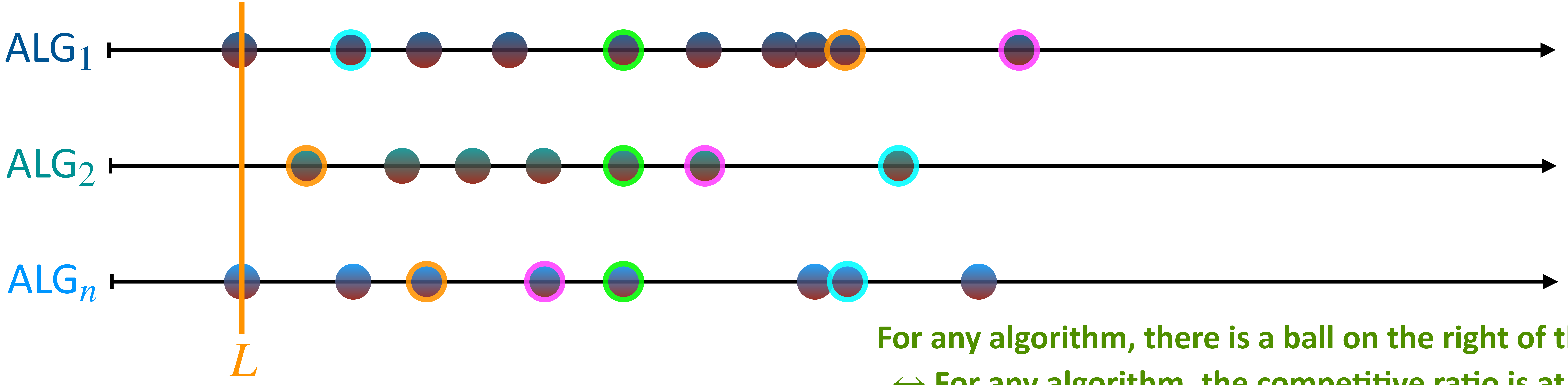
For any algorithm, there is a ball on the right of the bar L

$\leftrightarrow$ For any algorithm, the competitive ratio is at least L

Problem Competitive Ratio Lower Bound

- Formally, we prove that for any online algorithm ALG_i, there exists an instance I_i such that

$$\frac{\text{ALG}_i(I_i)}{\text{OPT}(I_i)} \geq L$$



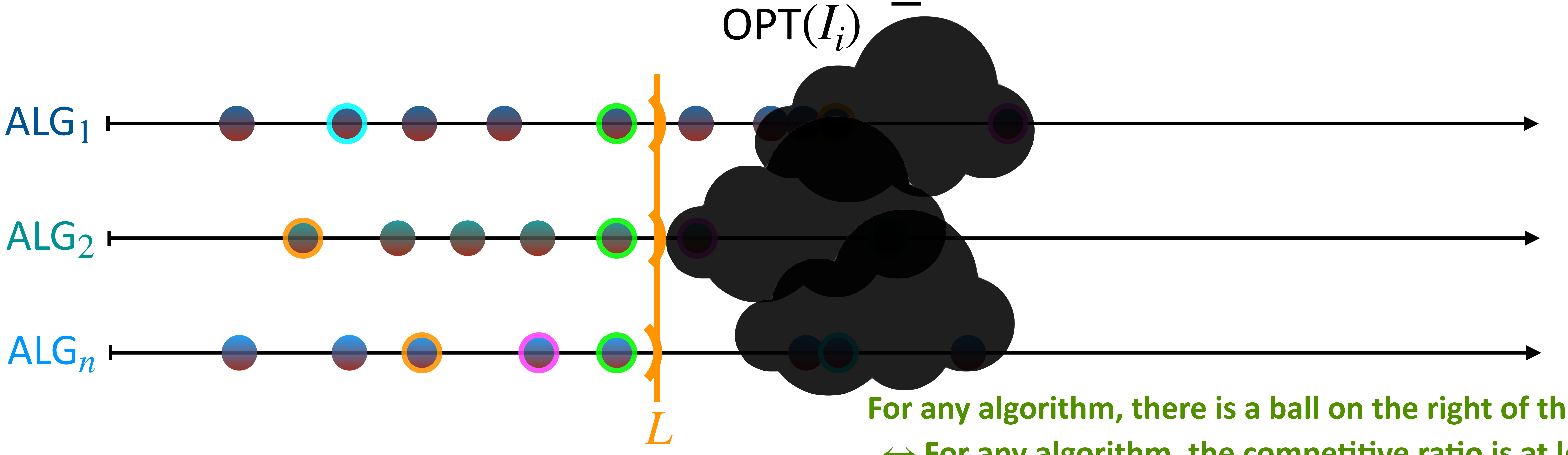
For any algorithm, there is a ball on the right of the bar L

$\leftrightarrow$ For any algorithm, the competitive ratio is at least L

Problem Competitive Ratio Lower Bound

- Formally, we prove that for any online algorithm ALG_i, there exists an instance I_i such that

$$\frac{\text{ALG}_i(I_i)}{\text{OPT}(I_i)} \geq L$$

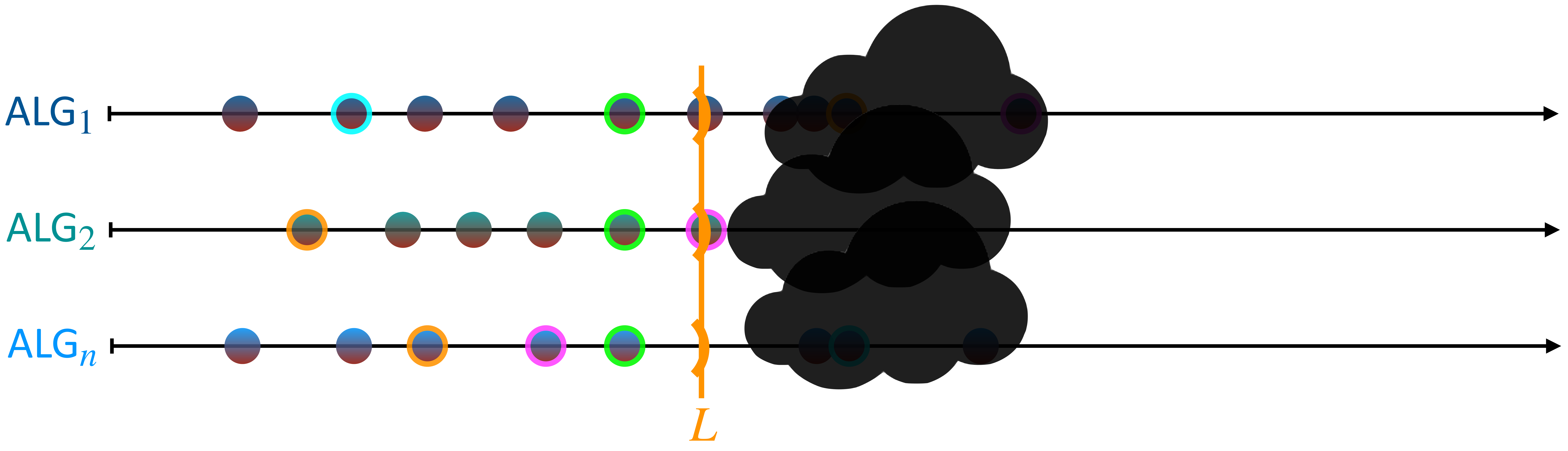


For any algorithm, there is a ball on the right of the bar L

$\leftrightarrow$ For any algorithm, the competitive ratio is at least L

Problem Competitive Ratio Lower Bound

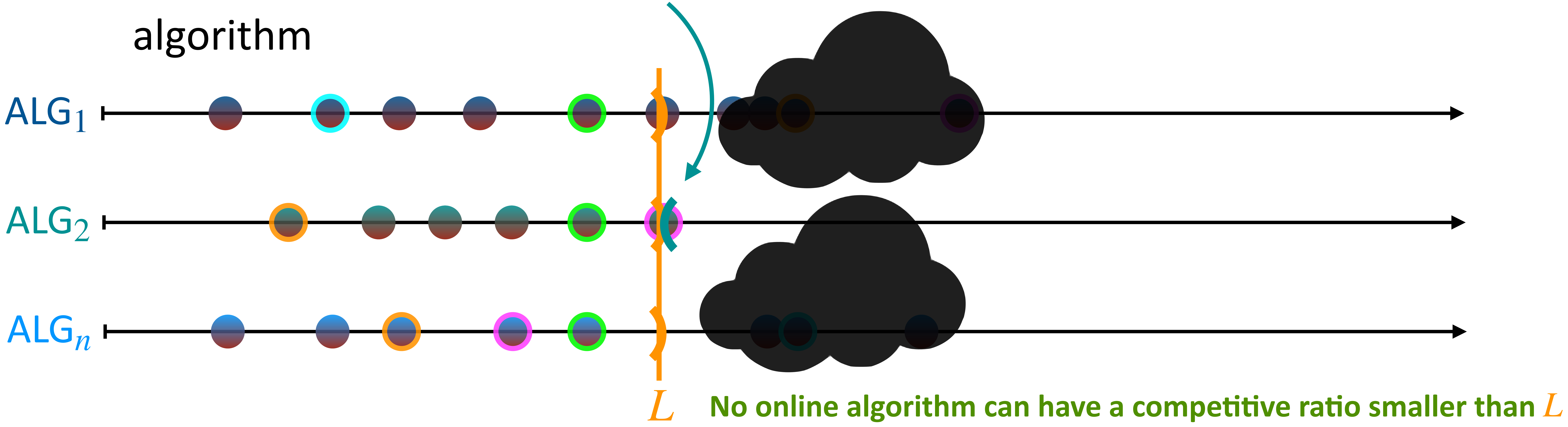
- For any algorithm, there is a ball at or on the right of the bar L
  $\leftrightarrow$ For any algorithm, the competitive ratio is at least L



Problem Competitive Ratio Lower Bound

- For any algorithm, there is a ball at or on the right of the bar L
  $\leftrightarrow$ For any algorithm, the competitive ratio is at least L

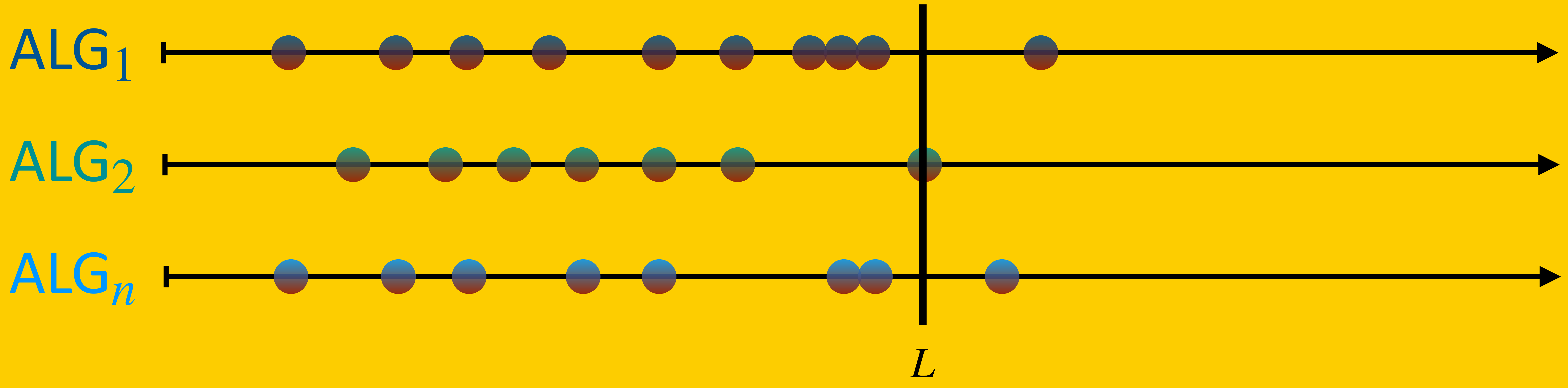
- If there is an algorithm that is L-competitive, it is the best online algorithm



What Happened

- If you find a way to design (a series of) instances such that for any online algorithm, the ratio between its cost and the optimal cost is at least L, you show that no online algorithm can be better than L-competitive

- In this case, if you have an online algorithm which is at most L -competitive, it is the best (optimal) online algorithm for this problem



Competitive Ratios

- An algorithm ALG is c-competitive if

$$\text{for all instance } I, \frac{\text{ALG}(I)}{\text{OPT}(I)} \leq c \text{ (minimization)}$$

  - Show that ALG is at most c-competitive (upper bound):

    Claim that for any I, $\text{ALG}(I) \leq x$ and $\text{OPT}(I) \geq y$, hence, $\frac{\text{ALG}(I)}{\text{OPT}(I)} \leq \frac{x}{y} \leq c$

  - Show that ALG is at least d-competitive (lower bound):

    Find an instance I' such that $\frac{\text{ALG}(I')}{\text{OPT}(I')} \geq d$

- Show that no algorithm can be better than d-competitive:

  Find each possible algorithm ALG_i an instance I_i such that $\frac{\text{ALG}_i(I')}{\text{OPT}(I')} \geq d$

Outline

- Problem lower bound and "best" online algorithms
  - Ski-rental
  - Bin packing
  - Paging

- Bounding difference to the optimal solution — potential function
  - List accessing
  - k-server

Ski-Rental Problem Lower Bound

Ski-Rental Problem Lower Bound

- Theorem: For the Buy-or-Rent problem, there is no deterministic online algorithm better than $(2 - \frac{1}{B})$-competitive.

Ski-Rental Problem Lower Bound

- Theorem: For the Buy-or-Rent problem, there is no deterministic online algorithm better than $(2 - \frac{1}{B})$-competitive.

<Proof Idea>

Any online algorithm must buy the ski on some day.

Ski-Rental Problem Lower Bound

- Theorem: For the Buy-or-Rent problem, there is no deterministic online algorithm better than $(2 - \frac{1}{B})$-competitive.

<Proof Idea>

Any online algorithm must buy the ski on some day.
Assume that algorithm ALG$_k$ buys the ski on the k-th skiing day, we design the adversarial input I_k that there are exactly k skiing days.

Ski-Rental Problem Lower Bound

- Theorem: For the Buy-or-Rent problem, there is no deterministic online algorithm better than $(2-\frac{1}{B})$-competitive.

<Proof Idea>

Any online algorithm must buy the ski on some day.
Assume that algorithm ALG_k buys the ski on the k-th skiing day, we design the adversarial input I_k that there are exactly k skiing days.

As long as we can prove that $\frac{\text{ALG}_k(I_k)}{\text{OPT}(I_k)} \geq 2-\frac{1}{B}$ for all k, the theorem is proven.

Ski-Rental Problem Lower Bound

- Theorem: For the Buy-or-Rent problem, there is no deterministic online algorithm better than $(2 - \frac{1}{B})$-competitive.

<Proof> Consider ALG$_k$ and I_k. Since I_k is the instance with exactly k skiing days. The cost of algorithm ALG$_k$ on instance I_k is $(k-1)+B$, while the optimal cost is $\min\{B, k\}$.

- If $k \geq B$, the optimal cost is B and the ratio

$$\frac{\text{ALG}_k(I_k)}{\text{OPT}_k(I_k)} = \frac{(k-1)+B}{B} \geq \frac{(B-1)+B}{B} = 2 - \frac{1}{B}$$

Ski-Rental Problem Lower Bound

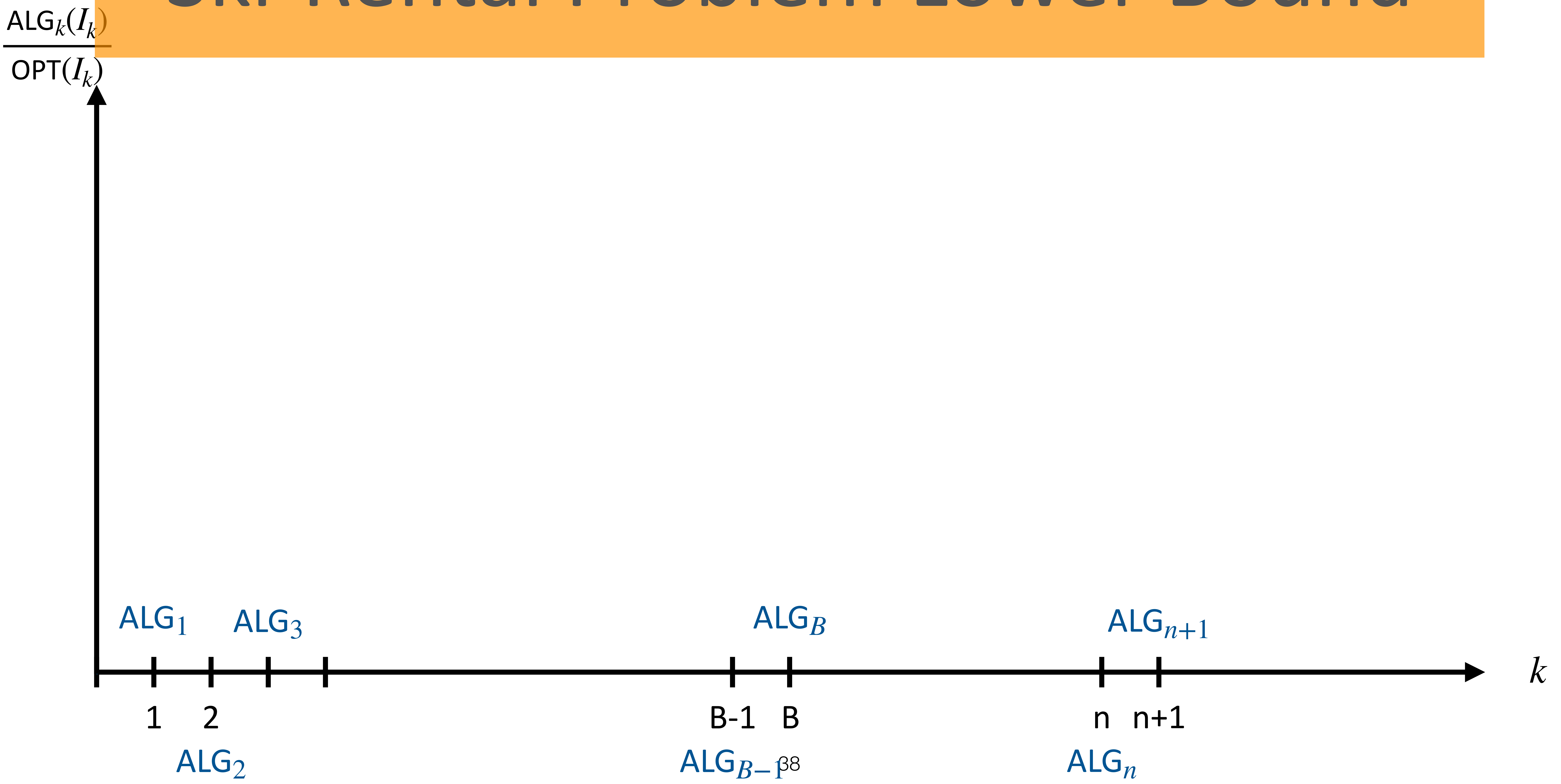
- Theorem: For the Buy-or-Rent problem, there is no deterministic online algorithm better than $(2 - \frac{1}{B})$-competitive.

<Proof> Consider ALG$_k$ and I_k. Since I_k is the instance with exactly k skiing days. The cost of algorithm ALG$_k$ on instance I_k is $(k-1)+B$, while the optimal cost is $\min\{B, k\}$.

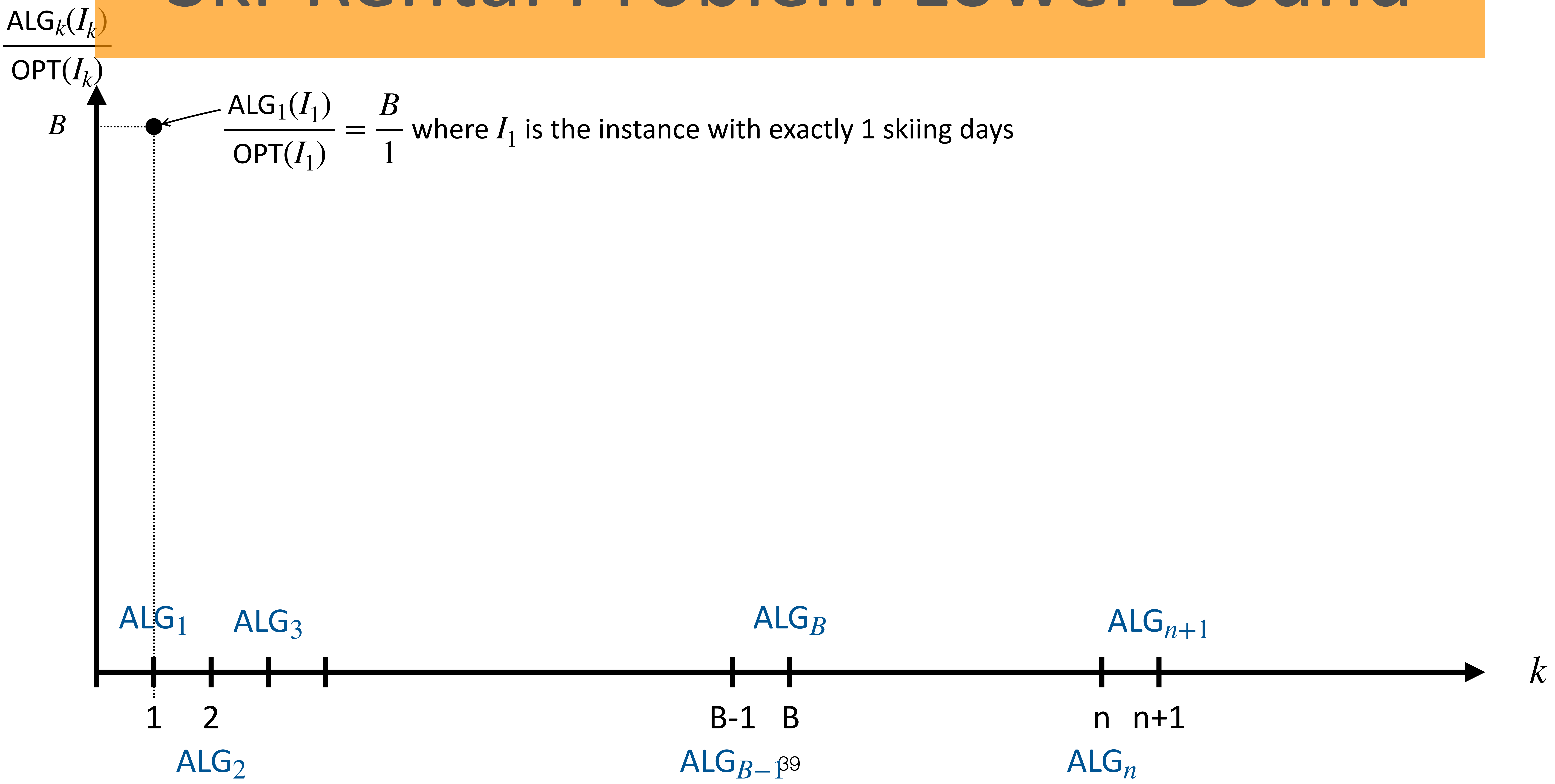
- If $k < B$, the ratio $\frac{\text{ALG}_k(I_k)}{\text{OPT}_k(I_k)} = \frac{(k-1)+B}{k}$. The ratio decreases as k increases.

  Hence, the ratio is lower bounded by $\frac{(B-1)+B}{B}$ since $k < B$

Ski-Rental Problem Lower Bound

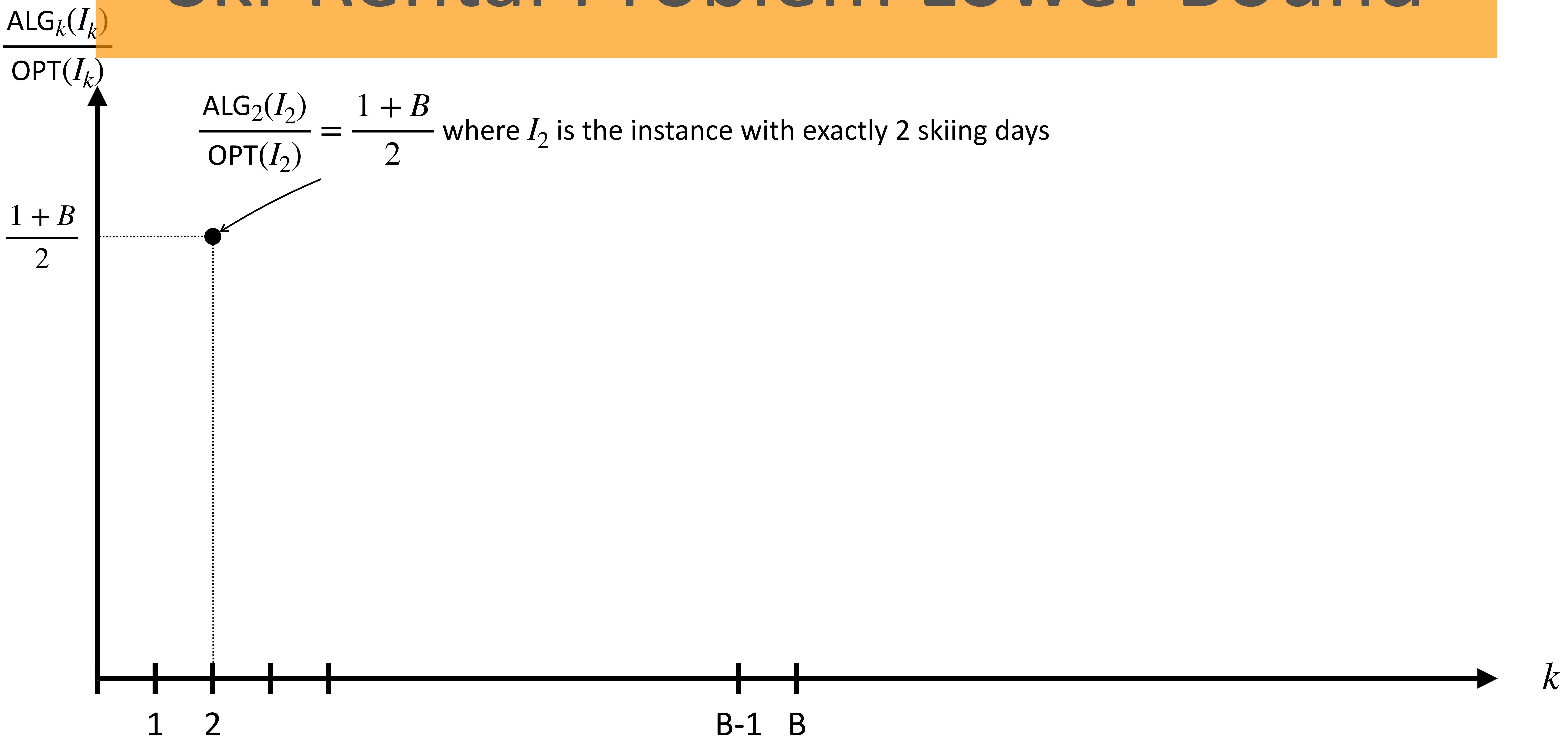


Ski-Rental Problem Lower Bound

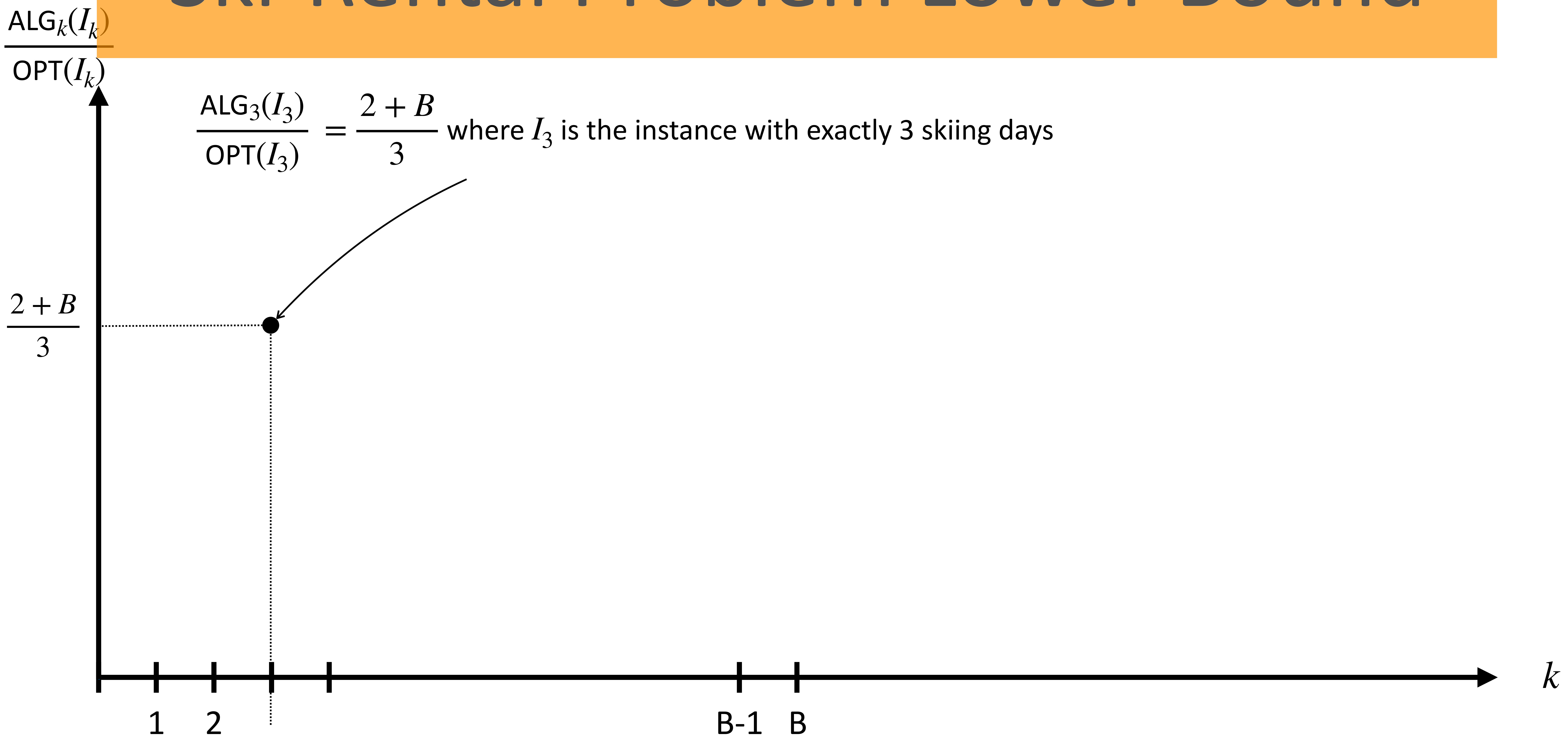


Ski-Rental Problem Lower Bound

$\frac{\text{ALG}_2(I_2)}{\text{OPT}(I_2)} = \frac{1+B}{2}$ where I_2 is the instance with exactly 2 skiing days

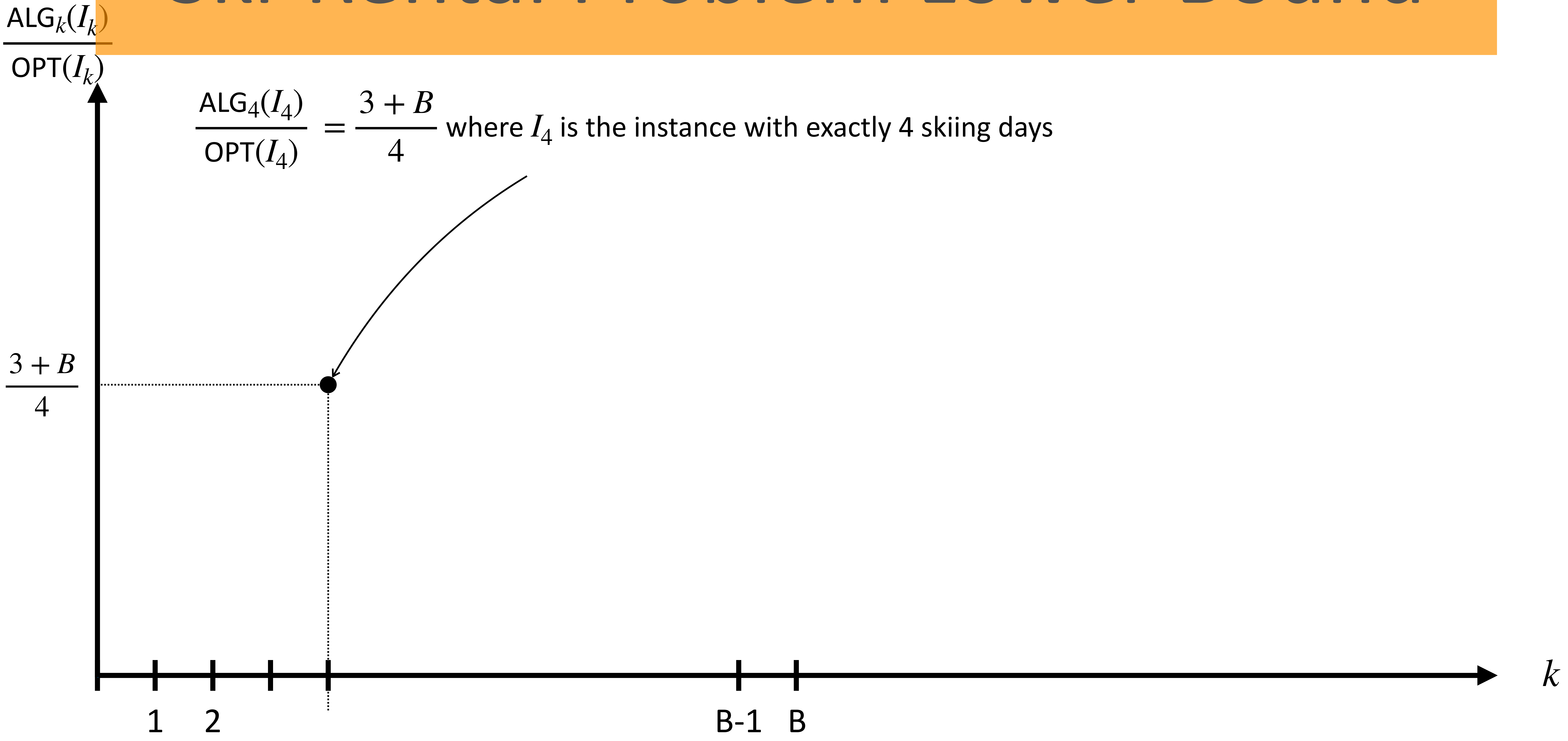


Ski-Rental Problem Lower Bound

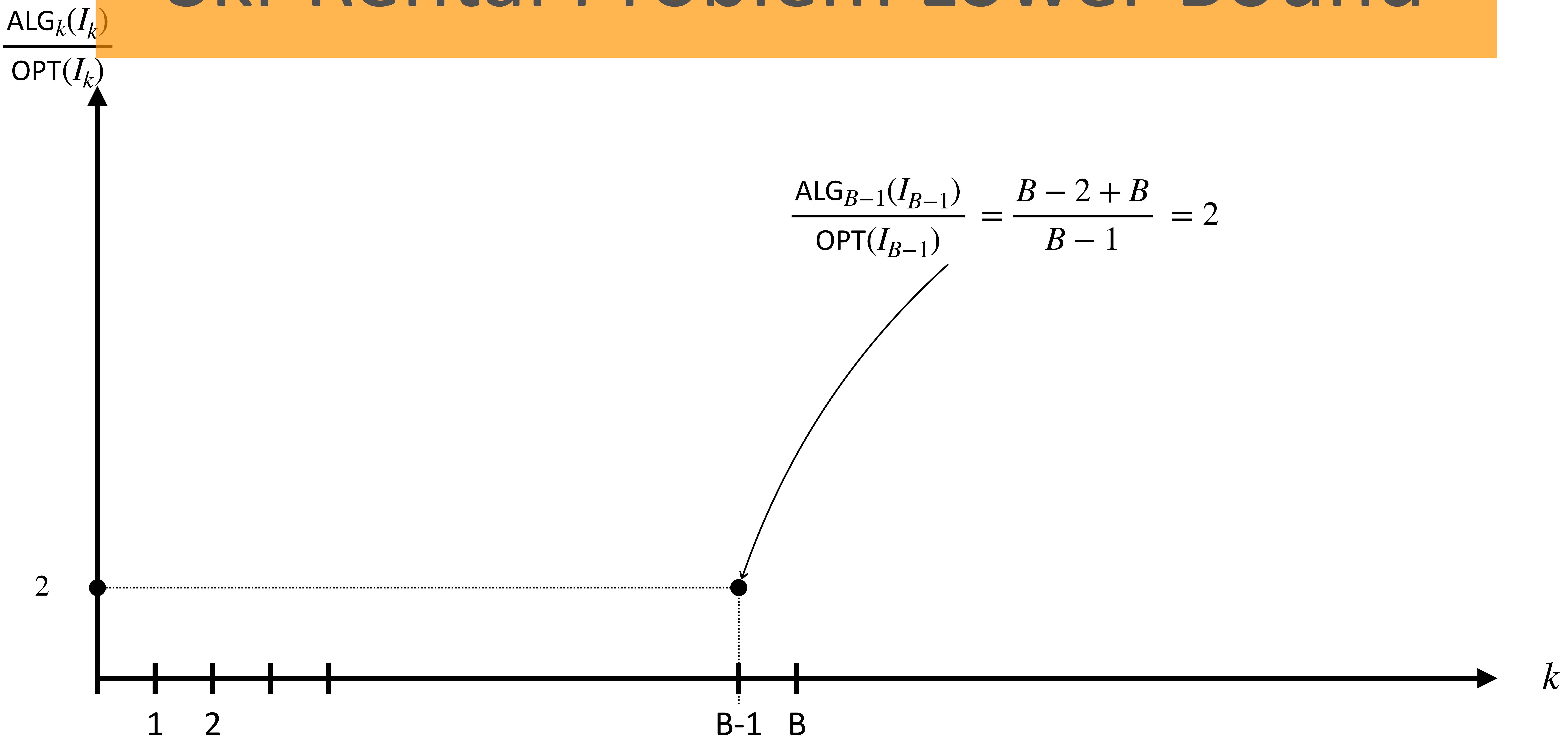


Ski-Rental Problem Lower Bound

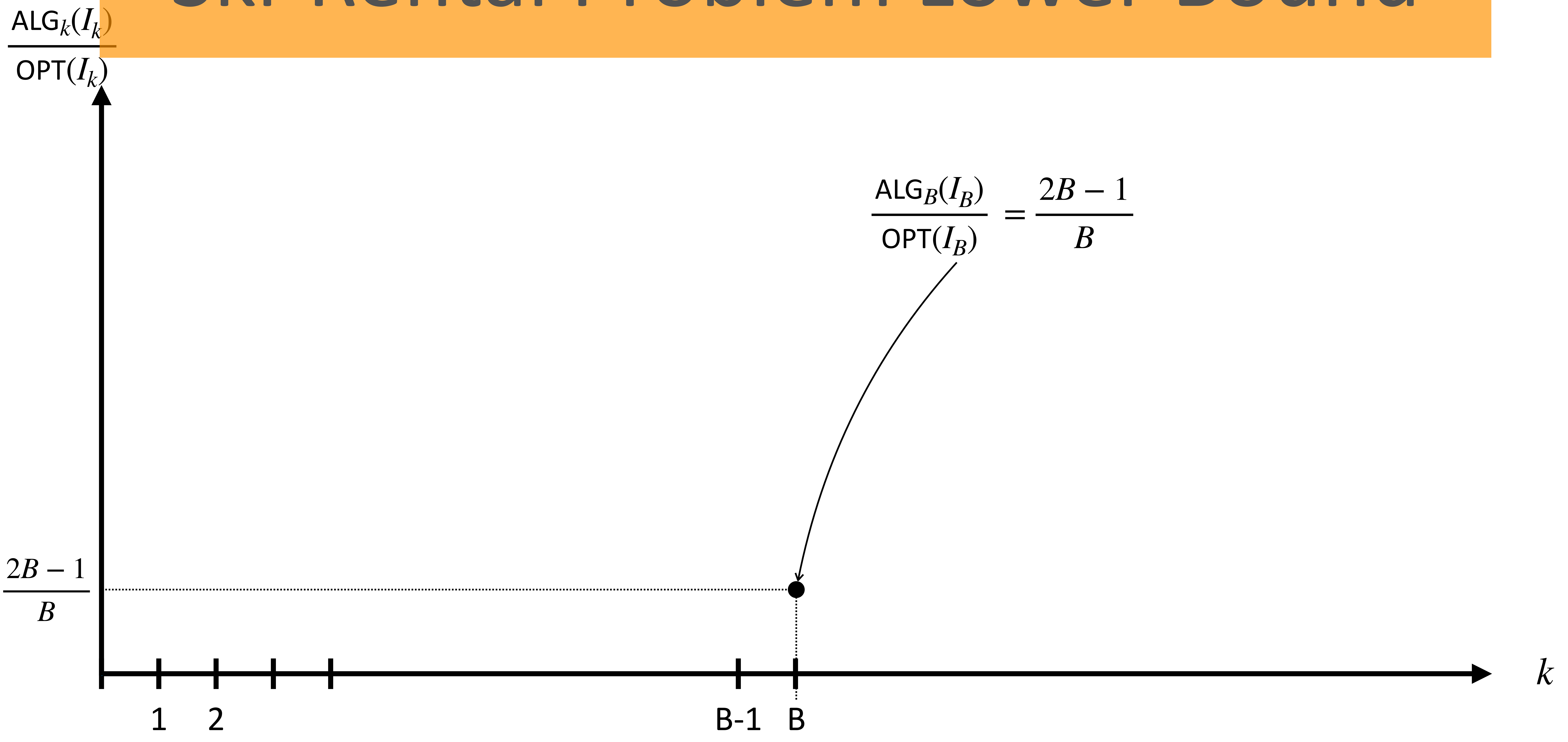
$\frac{\text{ALG}_4(I_4)}{\text{OPT}(I_4)} = \frac{3+B}{4}$ where I_4 is the instance with exactly 4 skiing days



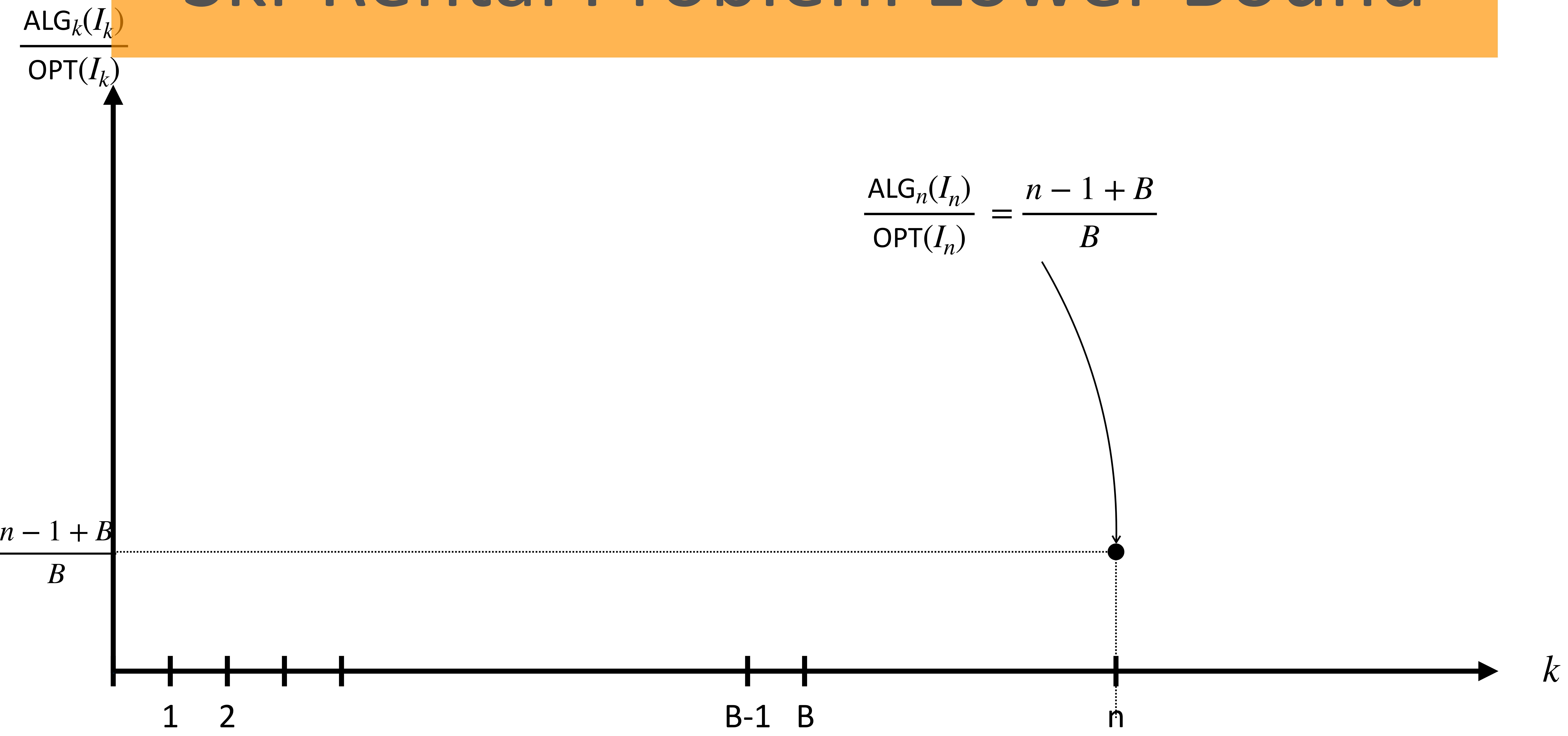
Ski-Rental Problem Lower Bound



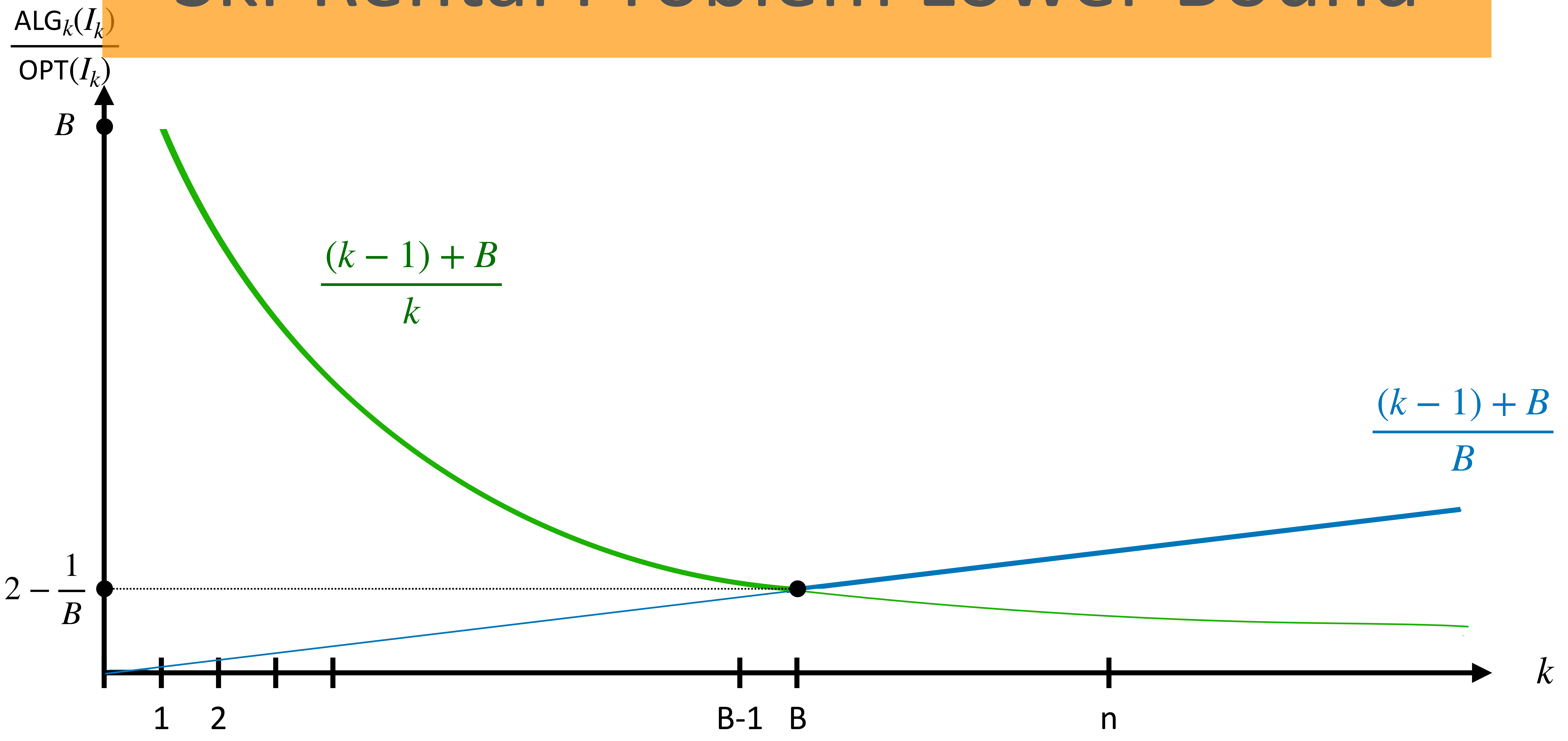
Ski-Rental Problem Lower Bound



Ski-Rental Problem Lower Bound



Ski-Rental Problem Lower Bound



What Happened

- We argue that any deterministic algorithm must buy the ski on some day
  - For any algorithm that buys the ski on the k-th day, we design an corresponding adversary which has exactly k skiing days
  - The case where $k = B - 1$ has the smallest ratio between the algorithm cost and the optimal cost, which gives a ratio of $2 - \frac{1}{B}$
    - That is, for any algorithm, there is an instance making its competitive ratio's lower bound at least $2 - \frac{1}{B}$

Optimal Online Algorithms

ALG: Buy the ski on the B-th skiing day

- Theorem: For the Buy-or-Rent problem, algorithm ALG is $(2-\frac{1}{B})$-competitive.
- Theorem: For the Buy-or-Rent problem, there is no deterministic online algorithm better than $(2-\frac{1}{B})$-competitive.
- Corollary: ALG is an optimal online algorithm
  - If an online algorithm attains the competitive ratio which matches the problem competitive ratio lower bound, the algorithm is an **optimal online algorithm**

Outline

- Problem lower bound and "best" online algorithms
  - Ski-rental
  - Bin packing
  - Paging
- Bounding difference to the optimal solution — potential function
  - List accessing
  - k-server

Bin Packing Problem Lower Bound

Any deterministic online algorithm is at least 1.333-competitive

Any deterministic online algorithm is at least 1.333-competitive

<Proof idea>

Prove by contradiction: design an instance such that any algorithm ALG that is (4/3-ϵ)-competitive for the first half of the instance, it cannot be (4/3-ϵ)-competitive for the whole instance.

Any deterministic online algorithm is at least 1.333-competitive

<Proof idea>

Prove by contradiction: design an instance such that any algorithm ALG that is (4/3-ϵ)-competitive for the first half of the instance, it cannot be (4/3-ϵ)-competitive for the whole instance. Consider the adversarial input:

$$\underbrace{\frac{1}{2}-\epsilon, \frac{1}{2}-\epsilon, \cdots, \frac{1}{2}-\epsilon}_{m}, \underbrace{\frac{1}{2}+\epsilon, \frac{1}{2}+\epsilon, \cdots, \frac{1}{2}+\epsilon}_{m}$$

Any deterministic online algorithm is at least 1.333-competitive

<Proof idea> Assume ALG is (4/3-ϵ)-competitive

Prove by contradiction: design an instance such that any algorithm ALG that is (4/3-ϵ)-competitive for the first half of the instance, it cannot be (4/3-ϵ)-competitive for the whole instance. Consider the adversarial input:

$$\underbrace{\frac{1}{2}-\epsilon, \frac{1}{2}-\epsilon, \cdots, \frac{1}{2}-\epsilon}_{m}$$

OPT(I) = $\dfrac{m}{2}$

Any deterministic online algorithm is at least 1.333-competitive

<Proof idea> Assume ALG is (4/3-ϵ)-competitive

Prove by contradiction: design an instance such that any algorithm ALG that is (4/3-ϵ)-competitive for the first half of the instance, it cannot be (4/3-ϵ)-competitive for the whole instance. Consider the adversarial input:

$$\underbrace{\frac{1}{2}-\epsilon, \frac{1}{2}-\epsilon, \cdots, \frac{1}{2}-\epsilon}_{m}$$

$\text{OPT}(I) = \frac{m}{2}$

$\text{ALG}(I) < \frac{4}{3} \cdot \frac{m}{2}$

Any deterministic online algorithm is at least 1.333-competitive

<Proof idea> Assume ALG is (4/3-ϵ)-competitive

Prove by contradiction: design an instance such that any algorithm ALG that is (4/3-ϵ)-competitive for the first half of the instance, it cannot be (4/3-ϵ)-competitive for the whole instance. Consider the adversarial input:

$$\underbrace{\frac{1}{2}-\epsilon, \frac{1}{2}-\epsilon, \dots, \frac{1}{2}-\epsilon}_{m}$$

$$\text{OPT}(I) = \frac{m}{2}$$

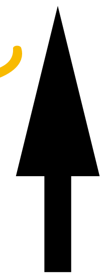
$$\text{ALG}(I) < \frac{4}{3} \cdot \frac{m}{2} = \frac{2}{3} \cdot m$$

Any deterministic online algorithm is at least 1.333-competitive

<Proof idea> Assume ALG is (4/3-ϵ)-competitive

Prove by contradiction: design an instance such that any algorithm ALG that is (4/3-ϵ)-competitive for the first half of the instance, it cannot be (4/3-ϵ)-competitive for the whole instance. Consider the adversarial input:

$$\underbrace{\frac{1}{2}-\epsilon, \frac{1}{2}-\epsilon, \cdots, \frac{1}{2}-\epsilon}_{m}$$



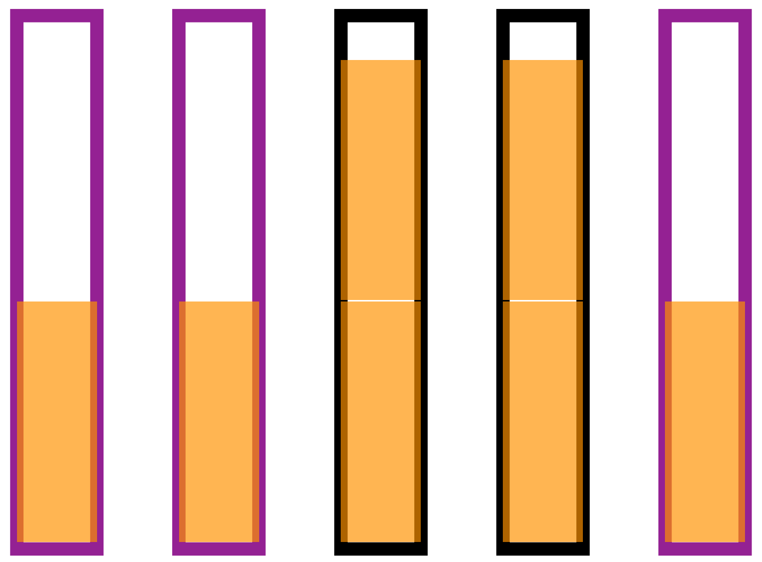
$$\text{OPT}(I) = \frac{m}{2}$$

$$\text{ALG}(I) < \frac{4}{3} \cdot \frac{m}{2} = \frac{2}{3} \cdot m$$

$$= a_1 + a_2$$

a_1: #bins with 1 item in ALG(I)

a_2: #bins with 2 items in ALG(I)



Any deterministic online algorithm is at least 1.333-competitive

<Proof idea> Assume ALG is (4/3-ϵ)-competitive

Prove by contradiction: design an instance such that any algorithm ALG that is (4/3-ϵ)-competitive for the first half of the instance, it cannot be (4/3-ϵ)-competitive for the whole instance. Consider the adversarial input:

$$\underbrace{\frac{1}{2}-\epsilon, \frac{1}{2}-\epsilon, \cdots, \frac{1}{2}-\epsilon}_{m}$$

There are m items

OPT(I) = $\frac{m}{2}$

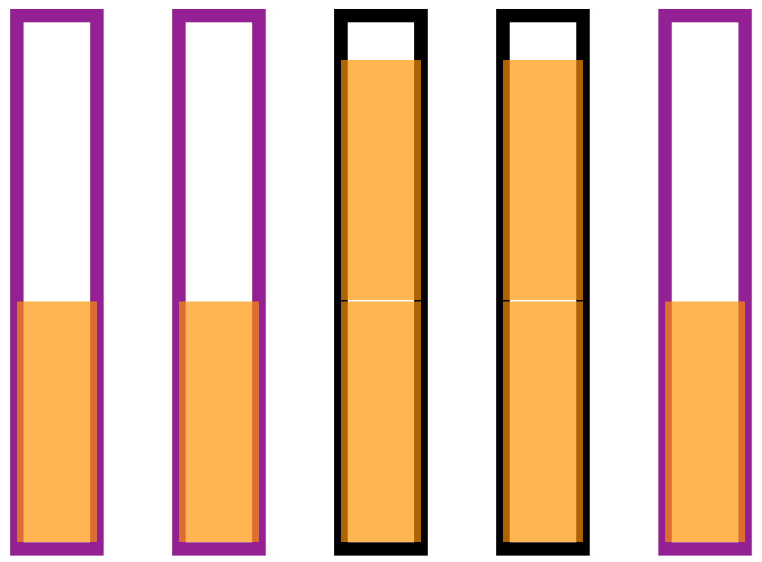
ALG(I) $< \frac{4}{3} \cdot \frac{m}{2}$ = $\frac{2}{3} \cdot m$

ALG(I) = $a_1 + a_2$

a_1: #bins with 1 item in ALG(I)

a_2: #bins with 2 items in ALG(I)

$m = a_1 + 2a_2$



Any deterministic online algorithm is at least 1.333-competitive

<Proof idea> Assume ALG is (4/3-ϵ)-competitive

Prove by contradiction: design an instance such that any algorithm ALG that is (4/3-ϵ)-competitive for the first half of the instance, it cannot be (4/3-ϵ)-competitive for the whole instance. Consider the adversarial input:

$$\underbrace{\frac{1}{2}-\epsilon, \frac{1}{2}-\epsilon, \cdots, \frac{1}{2}-\epsilon}_{m}$$

$\text{OPT}(I) = \frac{m}{2}$

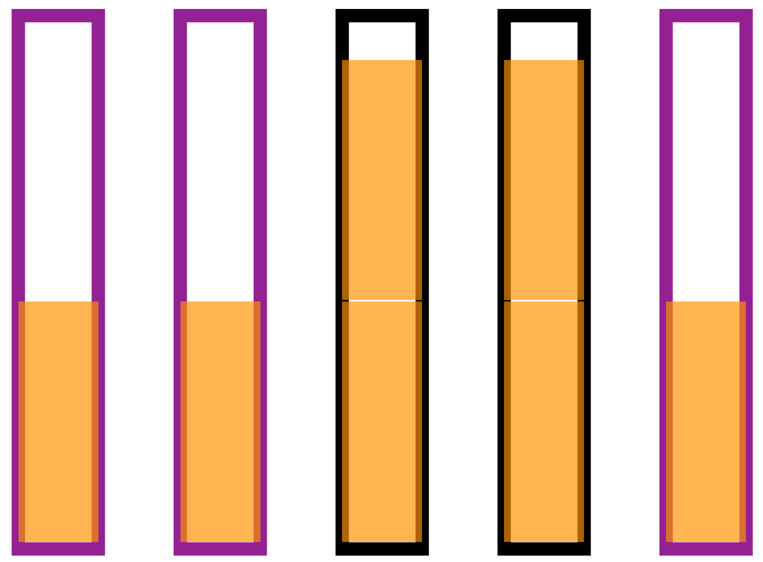
$\text{ALG}(I) < \frac{4}{3} \cdot \frac{m}{2} = \frac{2}{3} \cdot m$

$\text{ALG}(I) = a_1 + a_2 = m - a_2$

a_1: #bins with 1 item in ALG(I)

a_2: #bins with 2 items in ALG(I)

$m = a_1 + 2a_2$



Any deterministic online algorithm is at least 1.333-competitive

<Proof idea> Assume ALG is (4/3-ϵ)-competitive

Prove by contradiction: design an instance such that any algorithm ALG that is (4/3-ϵ)-competitive for the first half of the instance, it cannot be (4/3-ϵ)-competitive for the whole instance. Consider the adversarial input:

$$\underbrace{\frac{1}{2}-\epsilon, \frac{1}{2}-\epsilon, \cdots, \frac{1}{2}-\epsilon}_{m}, \underbrace{\frac{1}{2}+\epsilon, \frac{1}{2}+\epsilon, \cdots, \frac{1}{2}+\epsilon}_{m}$$

$$\text{OPT}(I) = \frac{m}{2}$$

$$\text{ALG}(I) < \frac{4}{3} \cdot \frac{m}{2} = \frac{2}{3} \cdot m$$

$$\text{ALG}(I) = a_1 + a_2 = m - a_2$$

a_1: #bins with 1 item in ALG(I)

a_2: #bins with 2 items in ALG(I)

$m = a_1 + 2a_2$

Any deterministic online algorithm is at least 1.333-competitive

<Proof idea> Assume ALG is (4/3-ϵ)-competitive

Prove by contradiction: design an instance such that any algorithm ALG that is (4/3-ϵ)-competitive for the first half of the instance, it cannot be (4/3-ϵ)-competitive for the whole instance. Consider the adversarial input:

$$\underbrace{\frac{1}{2}-\epsilon, \frac{1}{2}-\epsilon, \cdots, \frac{1}{2}-\epsilon}_{m}, \underbrace{\frac{1}{2}+\epsilon, \frac{1}{2}+\epsilon, \cdots, \frac{1}{2}+\epsilon}_{m}$$

$$\mathrm{OPT}(I) = \frac{m}{2}$$

$$\mathrm{ALG}(I) < \frac{4}{3} \cdot \frac{m}{2} = \frac{2}{3} \cdot m$$

$$\mathrm{ALG}(I) = a_1 + a_2 = m - a_2$$

a_1: #bins with 1 item in ALG(I)

a_2: #bins with 2 items in ALG(I)

$m = a_1 + 2a_2$

Any deterministic online algorithm is at least 1.333-competitive

<Proof idea> Assume ALG is (4/3-ϵ)-competitive

Prove by contradiction: design an instance such that any algorithm ALG that is (4/3-ϵ)-competitive for the first half of the instance, it cannot be (4/3-ϵ)-competitive for the whole instance. Consider the adversarial input:

$$\underbrace{\frac{1}{2}-\epsilon, \frac{1}{2}-\epsilon, \cdots, \frac{1}{2}-\epsilon}_{m}, \underbrace{\frac{1}{2}+\epsilon, \frac{1}{2}+\epsilon, \cdots, \frac{1}{2}+\epsilon}_{m}$$

OPT(I) = $\frac{m}{2}$

OPT(I+I) = m

ALG(I) $< \frac{4}{3} \cdot \frac{m}{2} = \frac{2}{3} \cdot m$

ALG(I) = $a_1 + a_2 = m - a_2$

a_1: #bins with 1 item in ALG(I)

a_2: #bins with 2 items in ALG(I)

$m = a_1 + 2a_2$

Any deterministic online algorithm is at least 1.333-competitive

<Proof idea> Assume ALG is (4/3-ϵ)-competitive

Prove by contradiction: design an instance such that any algorithm ALG that is (4/3-ϵ)-competitive for the first half of the instance, it cannot be (4/3-ϵ)-competitive for the whole instance. Consider the adversarial input:

$$\underbrace{\frac{1}{2}-\epsilon, \frac{1}{2}-\epsilon, \cdots, \frac{1}{2}-\epsilon}_{m}, \underbrace{\frac{1}{2}+\epsilon, \frac{1}{2}+\epsilon, \cdots, \frac{1}{2}+\epsilon}_{m}$$

OPT(I) = $\frac{m}{2}$

OPT(I+I) = m

ALG(I) $< \frac{4}{3} \cdot \frac{m}{2} = \frac{2}{3} \cdot m$

ALG(I+I) = $a_1 + a_2 + x$

ALG(I) = $a_1 + a_2 = m - a_2$

a_1: #bins with 1 item in ALG(I)

a_2: #bins with 2 items in ALG(I)

$m = a_1 + 2a_2$

Any deterministic online algorithm is at least 1.333-competitive

<Proof idea> Assume ALG is (4/3-ϵ)-competitive

Prove by contradiction: design an instance such that any algorithm ALG that is (4/3-ϵ)-competitive for the first half of the instance, it cannot be (4/3-ϵ)-competitive for the whole instance. Consider the adversarial input:

$$\underbrace{\frac{1}{2}-\epsilon, \frac{1}{2}-\epsilon, \cdots, \frac{1}{2}-\epsilon}_{m}, \underbrace{\frac{1}{2}+\epsilon, \frac{1}{2}+\epsilon, \cdots, \frac{1}{2}+\epsilon}_{m}$$

OPT(I) = $\dfrac{m}{2}$

OPT(I+I) = m

ALG(I) $< \dfrac{4}{3} \cdot \dfrac{m}{2} = \dfrac{2}{3} \cdot m$

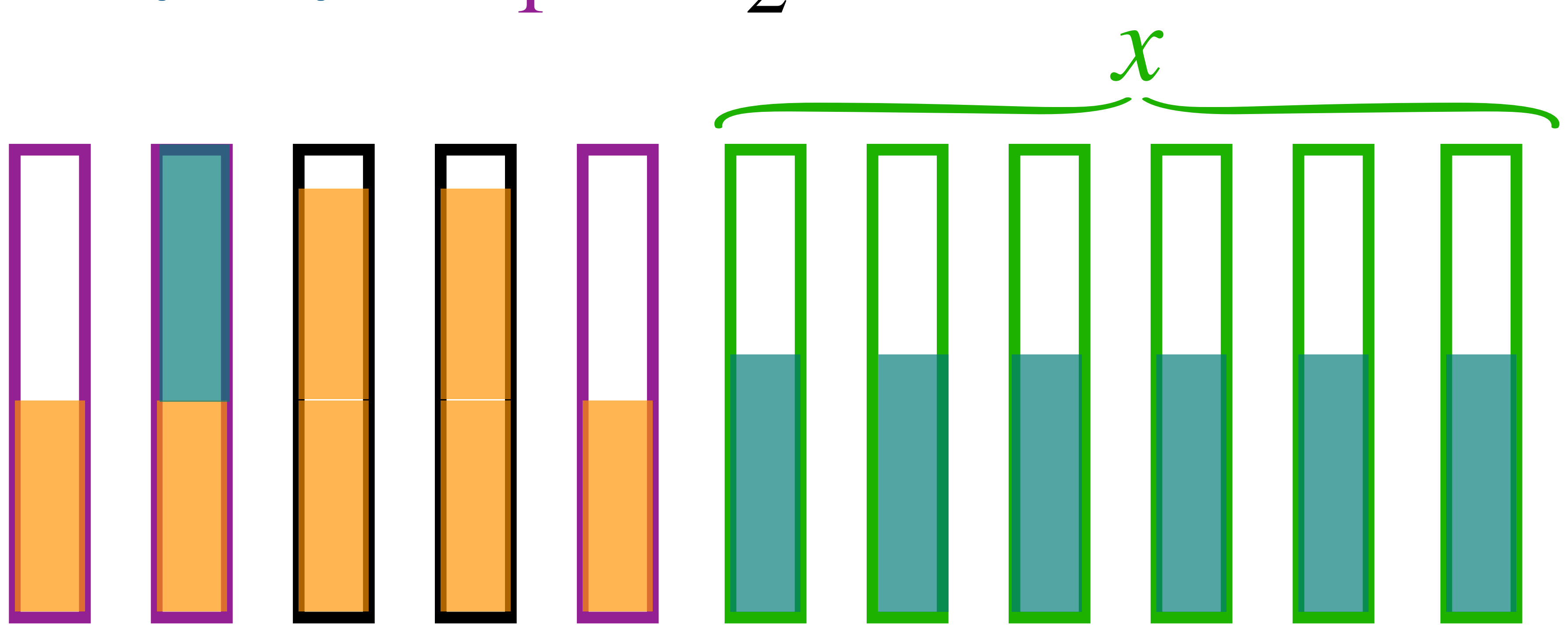
ALG(I) = $a_1 + a_2 = m - a_2$

a_1: #bins with 1 item in ALG(I)

a_2: #bins with 2 items in ALG(I)

$m = a_1 + 2a_2$

ALG(I+I) = $a_1 + a_2 + x$



Any deterministic online algorithm is at least 1.333-competitive

<Proof idea> Assume ALG is (4/3-ϵ)-competitive

Prove by contradiction: design an instance such that any algorithm ALG that is (4/3-ϵ)-competitive for the first half of the instance, it cannot be (4/3-ϵ)-competitive for the whole instance. Consider the adversarial input:

$$\underbrace{\frac{1}{2}-\epsilon, \frac{1}{2}-\epsilon, \cdots, \frac{1}{2}-\epsilon}_{m}, \underbrace{\frac{1}{2}+\epsilon, \frac{1}{2}+\epsilon, \cdots, \frac{1}{2}+\epsilon}_{m}$$

OPT(I) = $\frac{m}{2}$

OPT(I+I) = m

ALG(I) $< \frac{4}{3} \cdot \frac{m}{2} = \frac{2}{3} \cdot m$

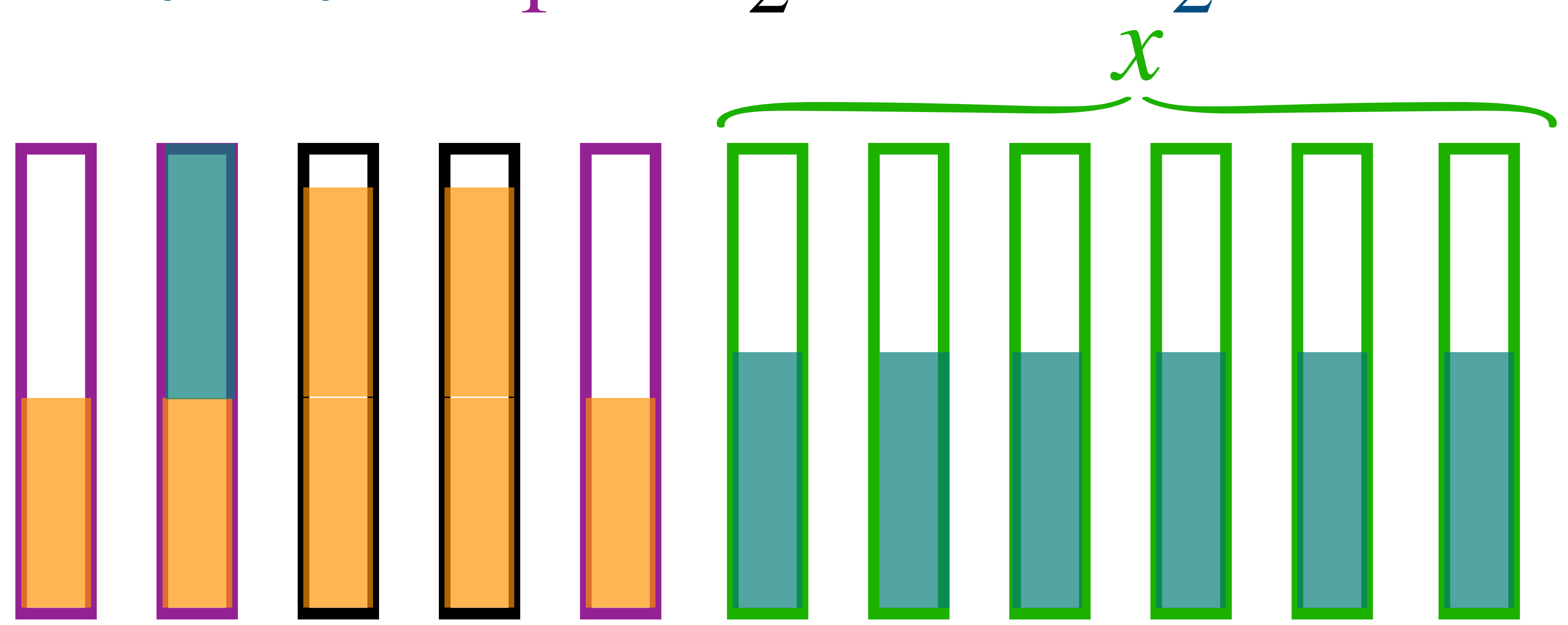
ALG(I) = $a_1 + a_2 = m - a_2$

a_1: #bins with 1 item in ALG(I)

a_2: #bins with 2 items in ALG(I)

$m = a_1 + 2a_2$

ALG(I+I) = $a_1 + a_2 + x \geq a_2 + m$



Any deterministic online algorithm is at least 1.333-competitive

<Proof idea> Assume ALG is (4/3-ϵ)-competitive

Prove by contradiction: design an instance such that any algorithm ALG that is (4/3-ϵ)-competitive for the first half of the instance, it cannot be (4/3-ϵ)-competitive for the whole instance. Consider the adversarial input:

$$\underbrace{\frac{1}{2}-\epsilon, \frac{1}{2}-\epsilon, \cdots, \frac{1}{2}-\epsilon}_{m}, \underbrace{\frac{1}{2}+\epsilon, \frac{1}{2}+\epsilon, \cdots, \frac{1}{2}+\epsilon}_{m}$$

OPT(I) = $\dfrac{m}{2}$

OPT(I+I) = m

ALG(I) $< \dfrac{4}{3} \cdot \dfrac{m}{2}$ = $\dfrac{2}{3} \cdot m$

ALG(I+I) = $a_1 + a_2 + x \geq a_2 + m$

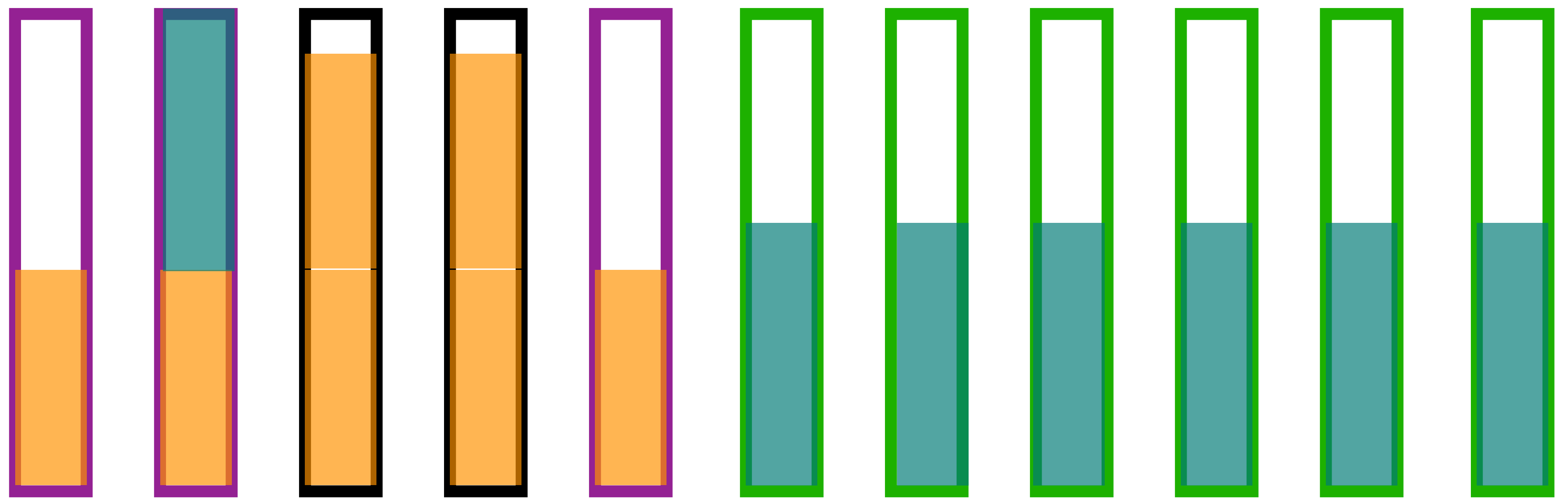
$\underbrace{}_{x}$

ALG(I) = $a_1 + a_2 = m - a_2$

a_1: #bins with 1 item in ALG(I)

a_2: #bins with 2 items in ALG(I)

$m = a_1 + 2a_2$



Any deterministic online algorithm is at least 1.333-competitive

<Proof idea> Assume ALG is (4/3-ϵ)-competitive

Prove by contradiction: design an instance such that any algorithm ALG that is (4/3-ϵ)-competitive for the first half of the instance, it cannot be (4/3-ϵ)-competitive for the whole instance. Consider the adversarial input:

$$\underbrace{\frac{1}{2}-\epsilon, \frac{1}{2}-\epsilon, \cdots, \frac{1}{2}-\epsilon}_{m}, \underbrace{\frac{1}{2}+\epsilon, \frac{1}{2}+\epsilon, \cdots, \frac{1}{2}+\epsilon}_{m}$$

OPT(I) = $\frac{m}{2}$

ALG(I) $< \frac{4}{3} \cdot \frac{m}{2} = \frac{2}{3} \cdot m$

ALG(I) = $a_1 + a_2 = m - a_2$

a_1: #bins with 1 item in ALG(I)

a_2: #bins with 2 items in ALG(I)

$m = a_1 + 2a_2$

OPT(I+I) = m

ALG(I+I) = $a_1 + a_2 + x \geq a_2 + m$

ALG(I+I) $< \frac{4}{3} \cdot$ OPT(I+I)

Any deterministic online algorithm is at least 1.333-competitive

<Proof idea> Assume ALG is (4/3-ϵ)-competitive

Prove by contradiction: design an instance such that any algorithm ALG that is (4/3-ϵ)-competitive for the first half of the instance, it cannot be (4/3-ϵ)-competitive for the whole instance. Consider the adversarial input:

$$\underbrace{\frac{1}{2}-\epsilon, \frac{1}{2}-\epsilon, \cdots, \frac{1}{2}-\epsilon}_{m}, \underbrace{\frac{1}{2}+\epsilon, \frac{1}{2}+\epsilon, \cdots, \frac{1}{2}+\epsilon}_{m}$$

OPT(I) = $\dfrac{m}{2}$

OPT(I+I) = m

ALG(I) $< \dfrac{4}{3} \cdot \dfrac{m}{2} = \dfrac{2}{3} \cdot m$

ALG(I) = $a_1 + a_2 = m - a_2$

a_1: #bins with 1 item in ALG(I)

a_2: #bins with 2 items in ALG(I)

$m = a_1 + 2a_2$

ALG(I+I) = $a_1 + a_2 + x \geq a_2 + m$

ALG(I+I) $< \dfrac{4}{3} \cdot$ OPT(I+I) = $\dfrac{4}{3} \cdot m$

Any deterministic online algorithm is at least 1.333-competitive

<Proof idea> Assume ALG is (4/3-ϵ)-competitive

Prove by contradiction: design an instance such that any algorithm ALG that is (4/3-ϵ)-competitive for the first half of the instance, it cannot be (4/3-ϵ)-competitive for the whole instance. Consider the adversarial input:

$$\underbrace{\frac{1}{2}-\epsilon, \frac{1}{2}-\epsilon, \cdots, \frac{1}{2}-\epsilon}_{m}, \underbrace{\frac{1}{2}+\epsilon, \frac{1}{2}+\epsilon, \cdots, \frac{1}{2}+\epsilon}_{m}$$

OPT(I) = $\frac{m}{2}$

OPT(I+I) = m

ALG(I) $< \frac{4}{3} \cdot \frac{m}{2} = \frac{2}{3} \cdot m$

ALG(I) = $a_1 + a_2 = m - a_2$

a_1: #bins with 1 item in ALG(I)

a_2: #bins with 2 items in ALG(I)

$m = a_1 + 2a_2$

ALG(I+I) = $a_1 + a_2 + x \geq a_2 + m$

ALG(I+I) $< \frac{4}{3} \cdot$ OPT(I+I) $= \frac{4}{3} \cdot m$

Any deterministic online algorithm is at least 1.333-competitive

<Proof idea> Assume ALG is (4/3-ϵ)-competitive

Prove by contradiction: design an instance such that any algorithm ALG that is (4/3-ϵ)-competitive for the first half of the instance, it cannot be (4/3-ϵ)-competitive for the whole instance. Consider the adverisarial input:

$$\underbrace{\frac{1}{2}-\epsilon, \frac{1}{2}-\epsilon, \cdots, \frac{1}{2}-\epsilon}_{m}, \underbrace{\frac{1}{2}+\epsilon, \frac{1}{2}+\epsilon, \cdots, \frac{1}{2}+\epsilon}_{m}$$

OPT(I) = $\frac{m}{2}$

ALG(I) $< \frac{4}{3} \cdot \frac{m}{2} = \frac{2}{3} \cdot m$

ALG(I) = $a_1 + a_2 = m - a_2$

a_1: #bins with 1 item in ALG(I)

a_2: #bins with 2 items in ALG(I)

$m = a_1 + 2a_2$

OPT(I+I) = m

ALG(I+I) = $a_1 + a_2 + x \geq a_2 + m$

ALG(I+I) $< \frac{4}{3} \cdot$ OPT(I+I) $= \frac{4}{3} \cdot m$

$a_2 < \frac{m}{3}$

Any deterministic online algorithm is at least 1.333-competitive

<Proof idea> Assume ALG is (4/3-ϵ)-competitive

Prove by contradiction: design an instance such that any algorithm ALG that is (4/3-ϵ)-competitive for the first half of the instance, it cannot be (4/3-ϵ)-competitive for the whole instance. Consider the adversarial input:

$$\underbrace{\frac{1}{2}-\epsilon, \frac{1}{2}-\epsilon, \cdots, \frac{1}{2}-\epsilon}_{m}, \underbrace{\frac{1}{2}+\epsilon, \frac{1}{2}+\epsilon, \cdots, \frac{1}{2}+\epsilon}_{m}$$

OPT(I) = $\frac{m}{2}$

ALG(I) $< \frac{4}{3} \cdot \frac{m}{2}$ = $\frac{2}{3} \cdot m$

ALG(I) = $a_1 + a_2 = m - a_2$

a_1: #bins with 1 item in ALG(I)

a_2: #bins with 2 items in ALG(I)

$m = a_1 + 2a_2$

OPT(I+I) = m

ALG(I+I) = $a_1 + a_2 + x \geq a_2 + m$

ALG(I+I) $< \frac{4}{3} \cdot$ OPT(I+I) $= \frac{4}{3} \cdot m$

$a_2 < \frac{m}{3}$

ALG(I) = $m - a_2$

Any deterministic online algorithm is at least 1.333-competitive

<Proof idea> Assume ALG is (4/3-ϵ)-competitive

Prove by contradiction: design an instance such that any algorithm ALG that is (4/3-ϵ)-competitive for the first half of the instance, it cannot be (4/3-ϵ)-competitive for the whole instance. Consider the adversarial input:

$$\underbrace{\frac{1}{2}-\epsilon, \frac{1}{2}-\epsilon, \cdots, \frac{1}{2}-\epsilon}_{m}, \underbrace{\frac{1}{2}+\epsilon, \frac{1}{2}+\epsilon, \cdots, \frac{1}{2}+\epsilon}_{m}$$

$\mathrm{OPT}(I) = \frac{m}{2}$

$\mathrm{OPT}(I+I) = m$

$\mathrm{ALG}(I) < \frac{4}{3} \cdot \frac{m}{2} = \frac{2}{3} \cdot m$

$\mathrm{ALG}(I) = a_1 + a_2 = m - a_2$

a_1: #bins with 1 item in ALG(I)

a_2: #bins with 2 items in ALG(I)

$m = a_1 + 2a_2$

$\mathrm{ALG}(I+I) = a_1 + a_2 + x \geq a_2 + m$

$\mathrm{ALG}(I+I) < \frac{4}{3} \cdot \mathrm{OPT}(I+I) = \frac{4}{3} \cdot m$

$a_2 < \frac{m}{3} \iff \mathrm{ALG}(I) = m - a_2 > \frac{2}{3} \cdot m$

Any deterministic online algorithm is at least 1.333-competitive

<Proof idea> Assume ALG is (4/3-ϵ)-competitive

Prove by contradiction: design an instance such that any algorithm ALG that is (4/3-ϵ)-competitive for the first half of the instance, it cannot be (4/3-ϵ)-competitive for the whole instance. Consider the adversarial input:

$$\underbrace{\frac{1}{2}-\epsilon, \frac{1}{2}-\epsilon, \cdots, \frac{1}{2}-\epsilon}_{m}, \underbrace{\frac{1}{2}+\epsilon, \frac{1}{2}+\epsilon, \cdots, \frac{1}{2}+\epsilon}_{m}$$

$\text{OPT}(I) = \frac{m}{2}$

$\text{ALG}(I) < \frac{4}{3} \cdot \frac{m}{2} = \frac{2}{3} \cdot m$

$\text{ALG}(I) = a_1 + a_2 = m - a_2$

a_1: #bins with 1 item in ALG(I)

a_2: #bins with 2 items in ALG(I)

$m = a_1 + 2a_2$

$\text{OPT}(I+I) = m$

$\text{ALG}(I+I) = a_1 + a_2 + x \geq a_2 + m$

$\text{ALG}(I+I) < \frac{4}{3} \cdot \text{OPT}(I+I) = \frac{4}{3} \cdot m$

$a_2 < \frac{m}{3} \Longleftrightarrow \text{ALG}(I) = m - a_2 > \frac{2}{3} \cdot m$ $\square$

Contradiction!

What Happened

- We first release m jobs, each with a size of $\frac{1}{2} - \epsilon$
  - For any algorithm, if it put these jobs in more than $\frac{2}{3} \cdot m$ bins, the adversary stops, and the algorithm is at least $\frac{4}{3}$-competitive
  - Otherwise, if an algorithm uses at most $\frac{2}{3} \cdot m$ bins for these jobs, we release another m jobs with size of $\frac{1}{2} + \epsilon$
    - This algorithm must uses more than $\frac{4}{3} \cdot m$ bins in total since it uses at most $\frac{2}{3} \cdot m$ bins for the first batch of jobs

Outline

- Problem lower bound and "best" online algorithms
  - Ski-rental
  - Bin packing
  - Paging

- Bounding difference to the optimal solution — potential function
  - List accessing
  - k-server

Paging Problem is at least k-competitive

Paging Problem is at least k-competitive

<Proof idea>

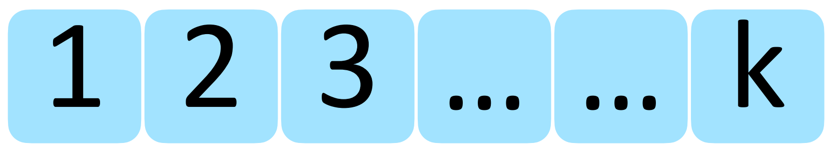
Assume that the cache size is k. Consider any algorithm **ALG** and design the adversary as follows: First request pages $1, 2, 3, \cdots, k$

1	2	3	…	…	k

Paging Problem is at least k-competitive

<Proof idea>

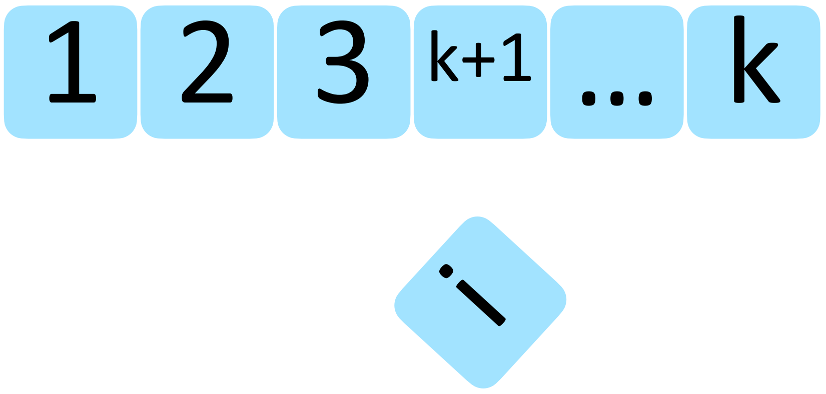
Assume that the cache size is k. Consider any algorithm **ALG** and design the adversary as follows: First request pages $1, 2, 3, \cdots, k, k+1$.



Paging Problem is at least k-competitive

<Proof idea>

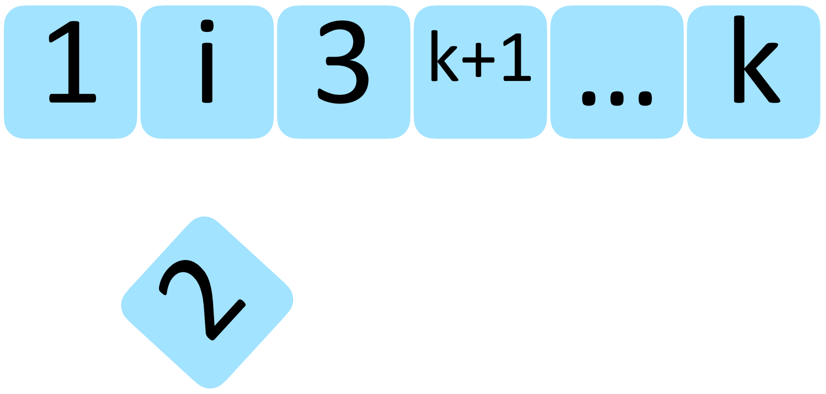
Assume that the cache size is k. Consider any algorithm **ALG** and design the adversary as follows: First request pages $1, 2, 3, \cdots, k, k+1$. At this moment, **ALG** evicts a page $i \in [1,k]$.



Paging Problem is at least k-competitive

<Proof idea>

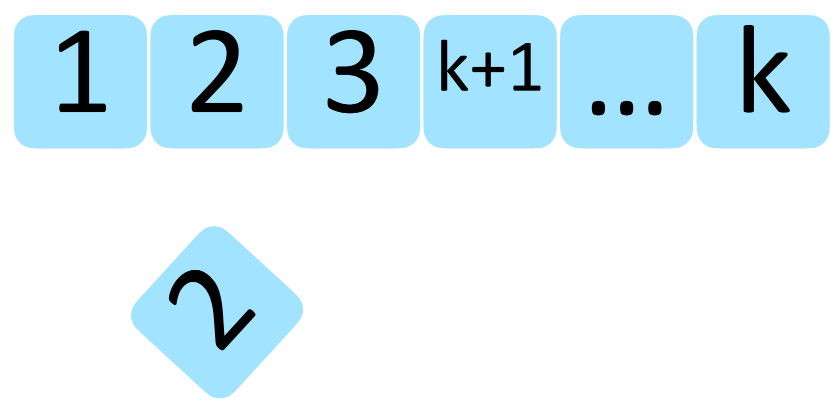
Assume that the cache size is k. Consider any algorithm **ALG** and design the adversary as follows: First request pages $1, 2, 3, \cdots, k, k+1$. At this moment, **ALG** evicts a page $i \in [1,k]$. Then, the adversary requests page i.



Paging Problem is at least k-competitive

<Proof idea>

Assume that the cache size is k. Consider any algorithm **ALG** and design the adversary as follows: First request pages $1, 2, 3, \cdots, k, k+1$. At this moment, **ALG** evicts a page $i \in [1,k]$. Then, the adversary requests page i. The adversary repeatedly requests the page evicted by **ALG** for $n-1$ rounds.



Paging Problem is at least k-competitive

<Proof idea>

Assume that the cache size is k. Consider any algorithm **ALG** and design the adversary as follows: First request pages $1, 2, 3, \cdots, k, k+1$. At this moment, **ALG** evicts a page $i \in [1,k]$. Then, the adversary requests page i. The adversary repeatedly requests the page evicted by **ALG** for $n-1$ rounds.

In this instance, each request incurs a page fault for ALG. Therefore, **ALG** costs $k+n$.

Paging Problem is at least k-competitive

<Proof idea>

Assume that the cache size is k. Consider any algorithm **ALG** and design the adversary as follows: First request pages $1, 2, 3, \cdots, k, k+1$. At this moment, **ALG** evicts a page $i \in [1,k]$. Then, the adversary requests page i. The adversary repeatedly requests the page evicted by **ALG** for $n-1$ rounds.

In this instance, each request incurs a page fault for ALG. Therefore, **ALG** costs $k+n$.

Because there are only $k+1$ pages involved, **OPT** incurs at most 1 page fault per k pages.

Paging Problem is at least k-competitive

<Proof idea>

Assume that the cache size is k. Consider any algorithm **ALG** and design the adversary as follows: First request pages $1, 2, 3, \cdots, k, k+1$. At this moment, **ALG** evicts a page $i \in [1,k]$. Then, the adversary requests page i. The adversary repeatedly requests the page evicted by **ALG** for $n-1$ rounds.

In this instance, each request incurs a page fault for ALG. Therefore, **ALG** costs $k+n$.

Because there are only $k+1$ pages involved, **OPT** incurs at most 1 page fault per k pages.

Therefore, $\dfrac{\text{ALG}(I)}{\text{OPT}(I)} \geq \dfrac{k+n}{k+n/k} \approx \Omega(k)$

Even when every page requests change dramatically, the optimal solution can keep the k pages that will be used in the most recent future and evict the one that will be used later.

Paging Problem is at least k-competitive

<Proof idea>

Assume that the cache size is k. Consider any algorithm **ALG** and design the adversary as follows: First request pages $1, 2, 3, \cdots, k, k+1$. At this moment, **ALG** evicts a page $i \in [1,k]$. Then, the adversary requests page i. The adversary repeatedly requests the page evicted by **ALG** for $n-1$ rounds.

In this instance, each request incurs a page fault for ALG. Therefore, **ALG** costs $k+n$.

Because there are only $k+1$ pages involved, **OPT** incurs at most 1 page fault per k pages.

Therefore, $\frac{\text{ALG}(I)}{\text{OPT}(I)} \geq \frac{k+n}{k+n/k} \approx \Omega(k)$

□

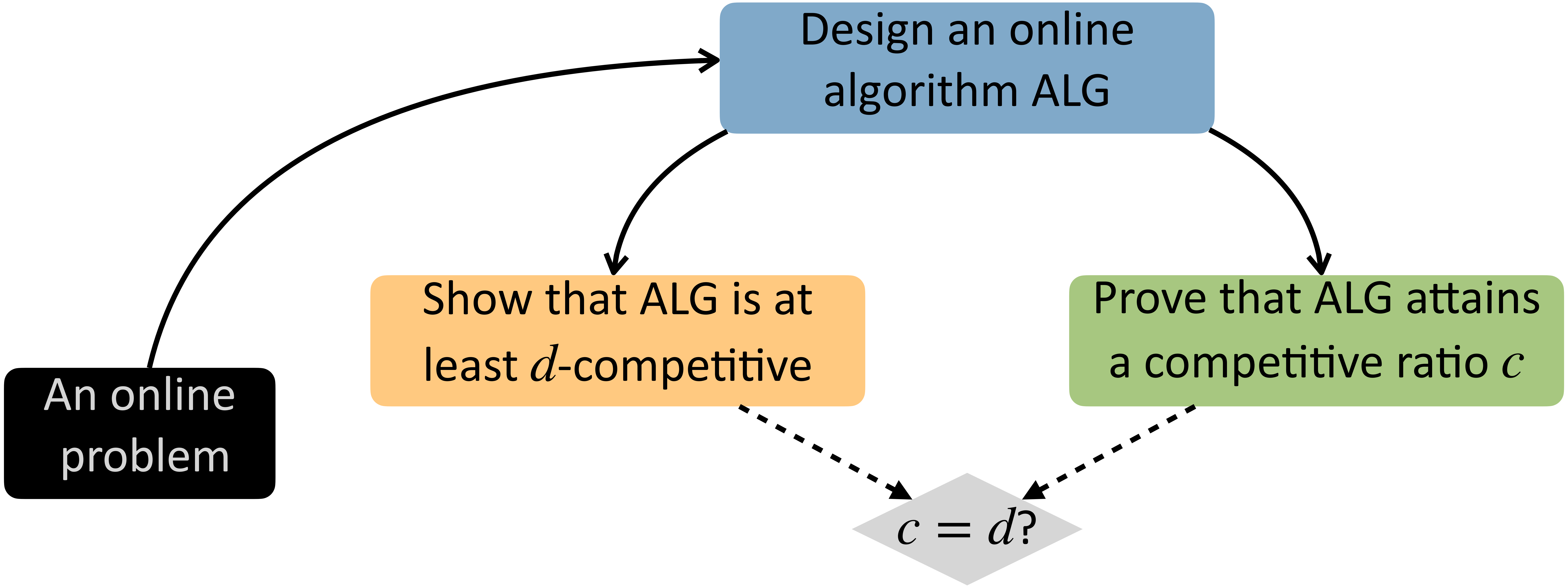
What Happened

- For any paging algorithm, the next page the adversary request is the page that was just evicted by the algorithm
  - The algorithm incurs $k + n$ page faults ($k + n$: number of requests)
  - For any sequence of k distinct requests, the optimal solution can always evict the page that will be used again the latest in the future
    - $\text{OPT} \leq k + \dfrac{n}{k}$

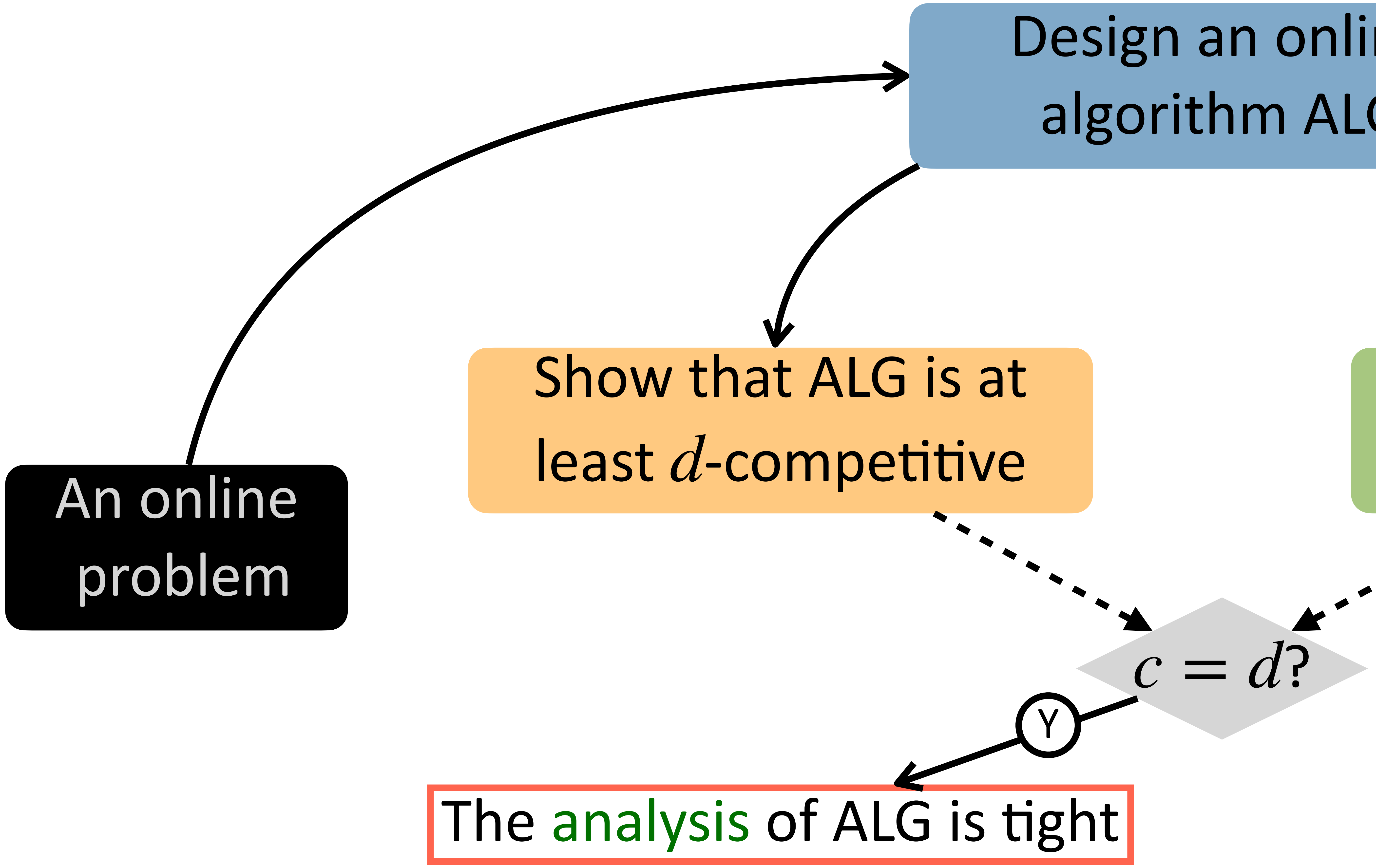
Recap: Online Optimization

An online
problem

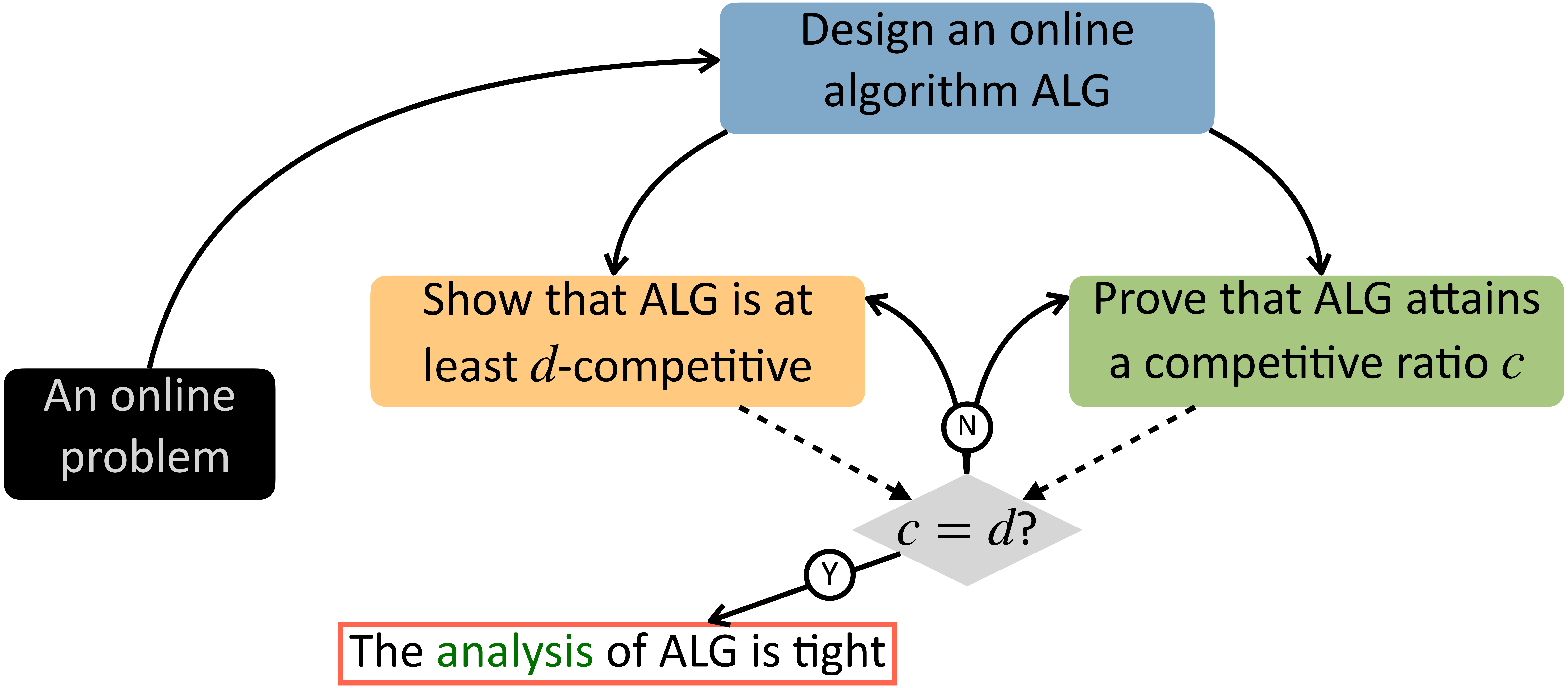
Recap: Online Optimization



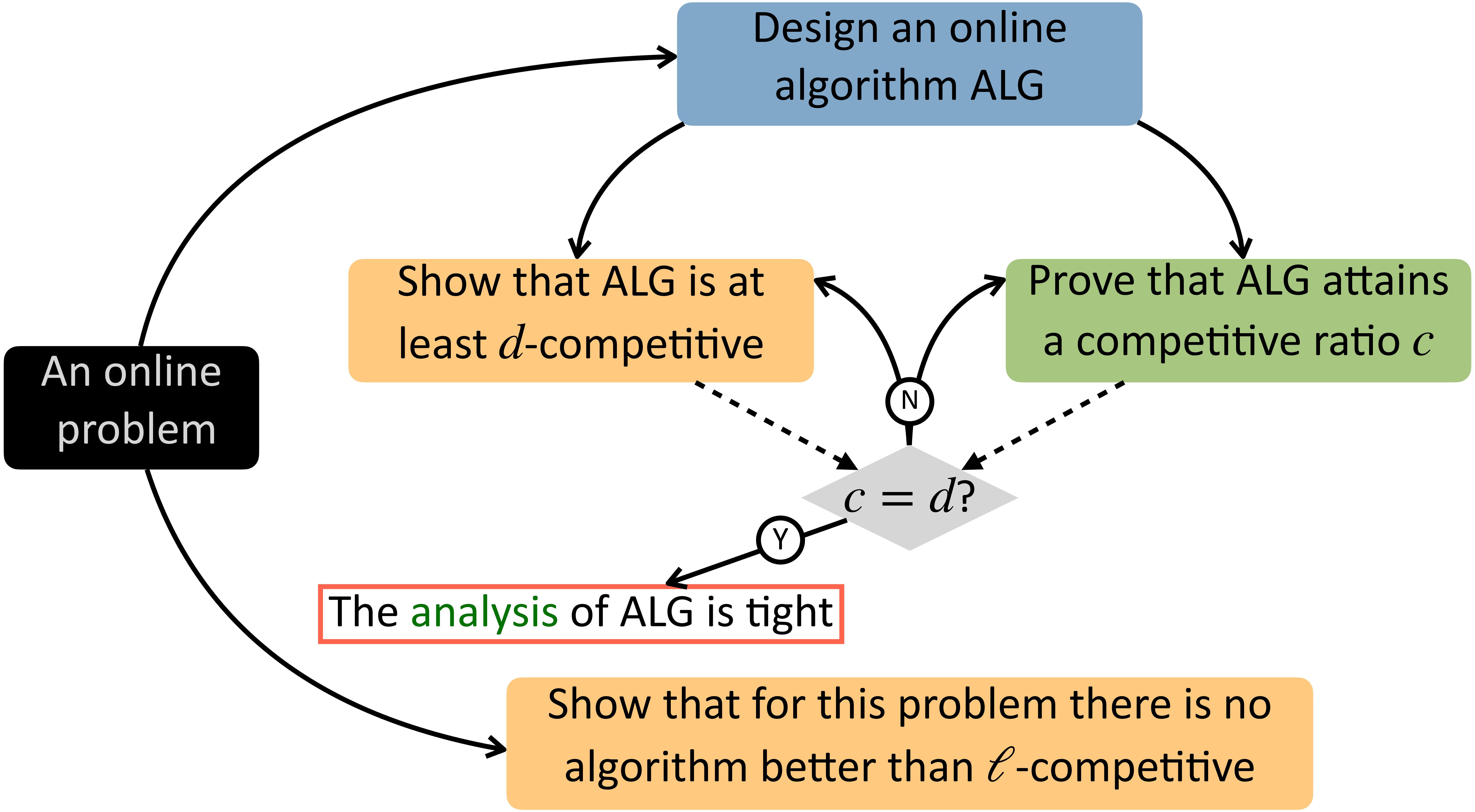
Recap: Online Optimization



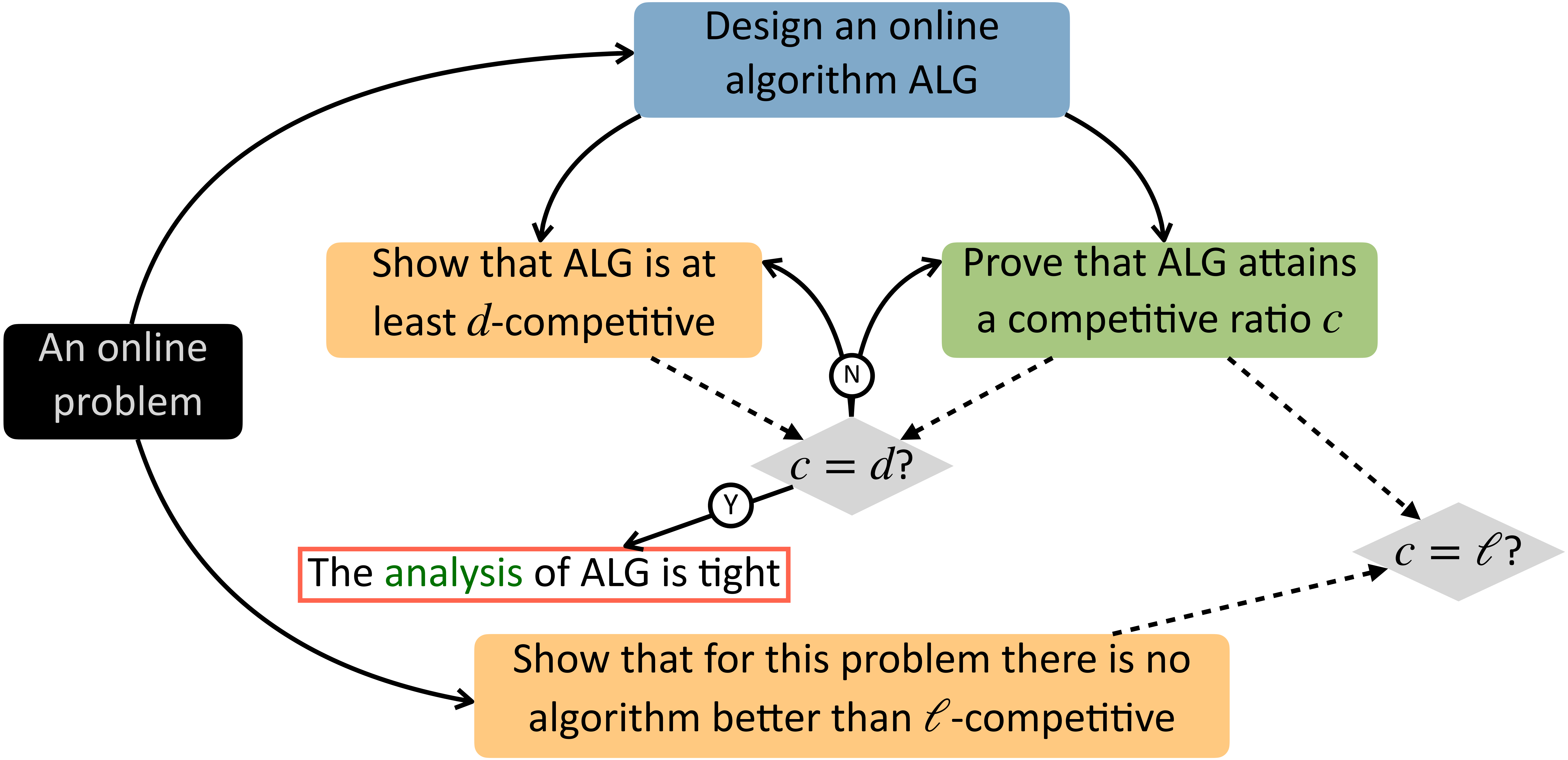
Recap: Online Optimization



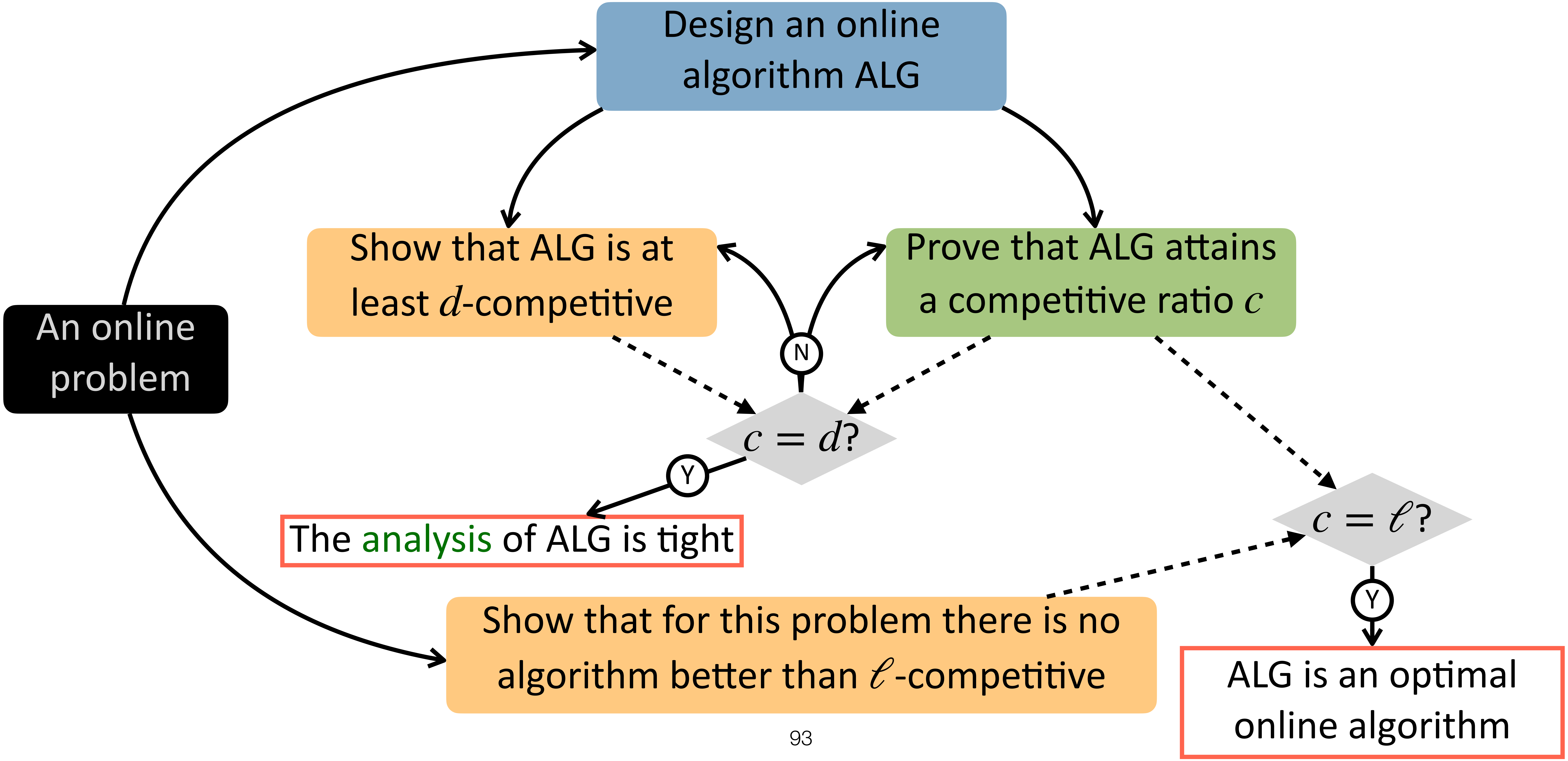
Recap: Online Optimization



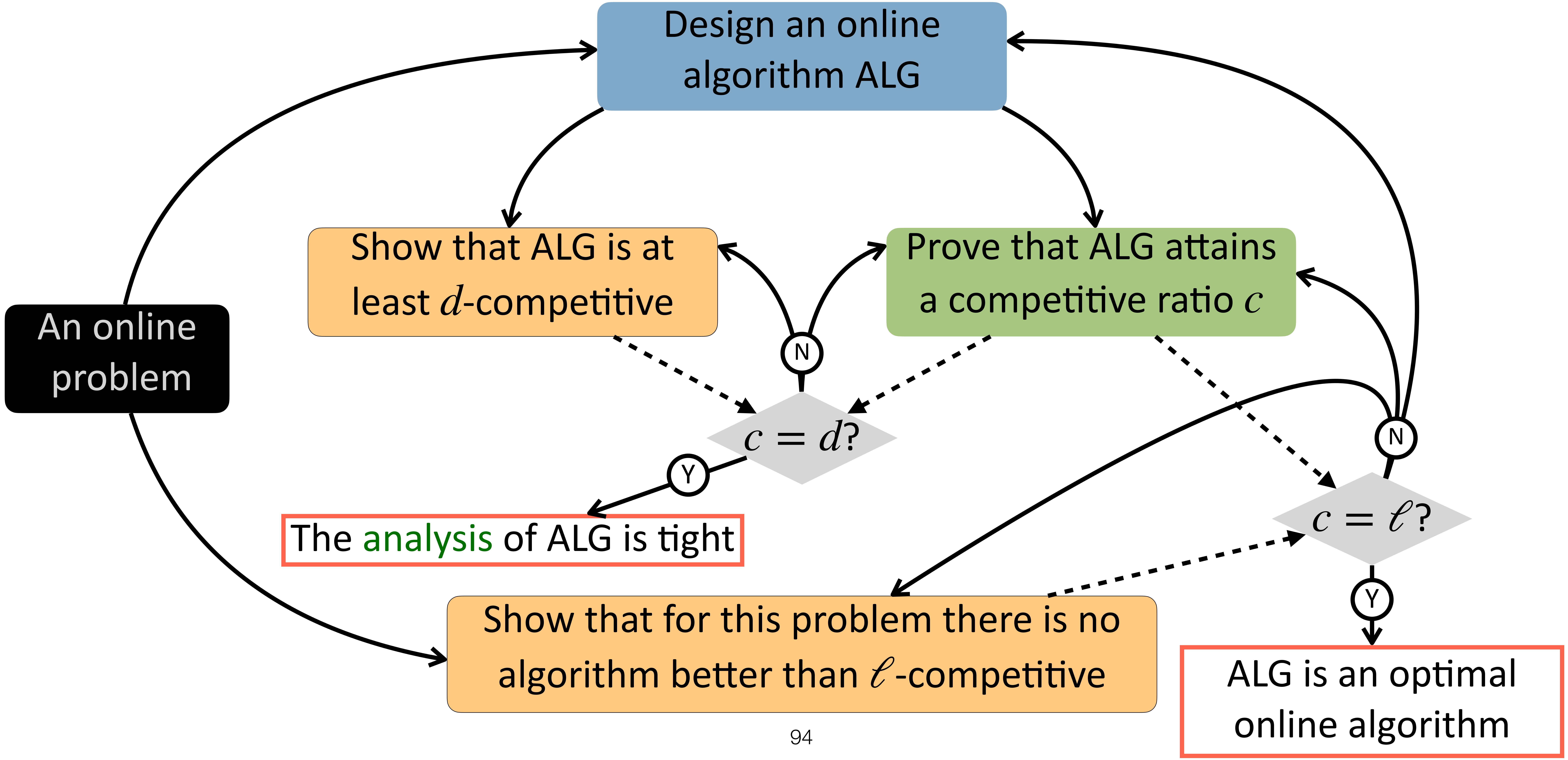
Recap: Online Optimization



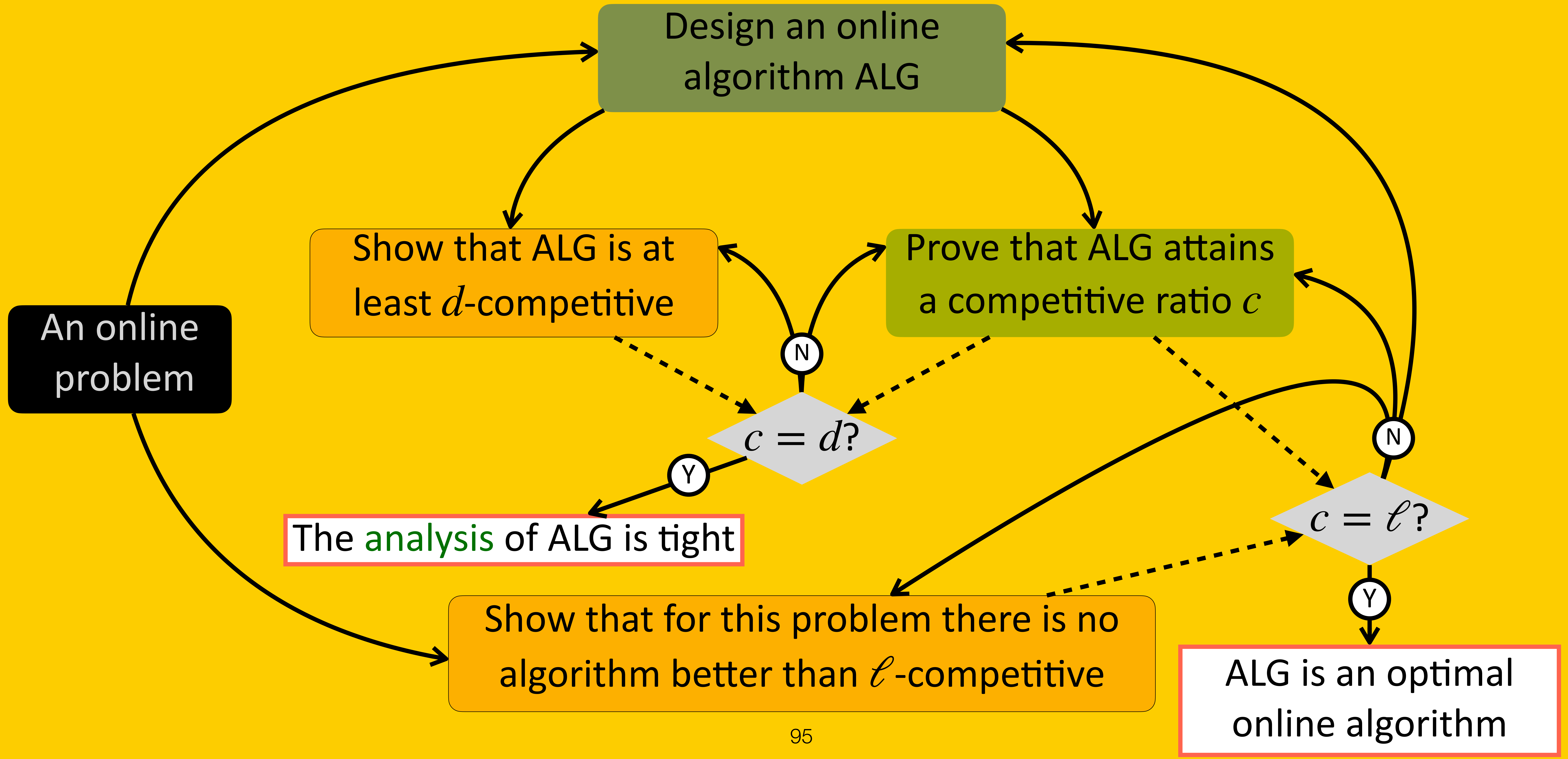
Recap: Online Optimization



Recap: Online Optimization



Recap: Online Optimization

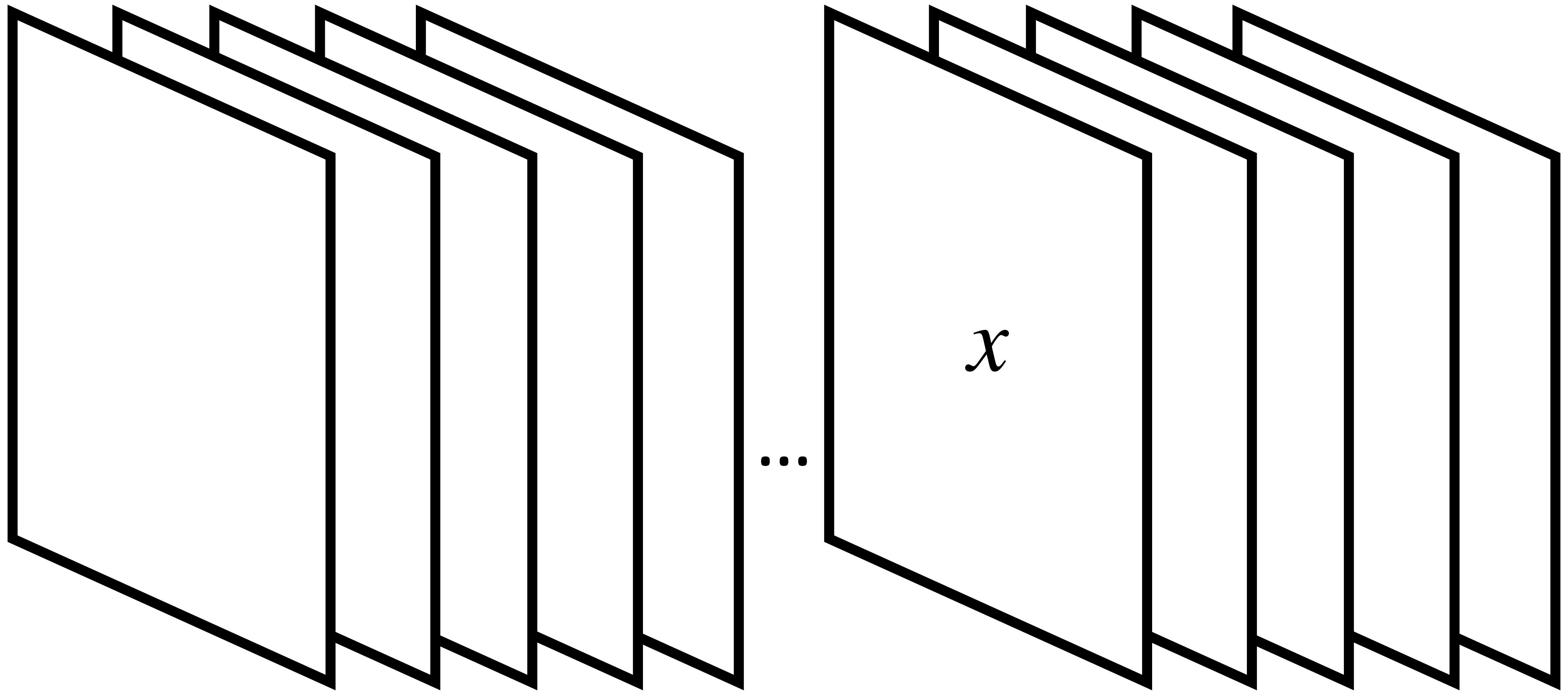


Outline

- Problem lower bound and “best” online algorithms
  - Ski-rental
  - Bin packing
  - Paging

- Bounding difference to the optimal solution — potential function
  - List accessing
  - k-server

List Accessing

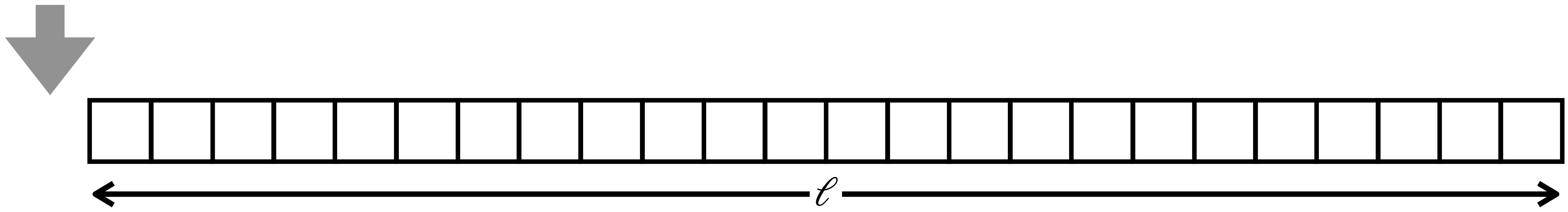


List Accessing

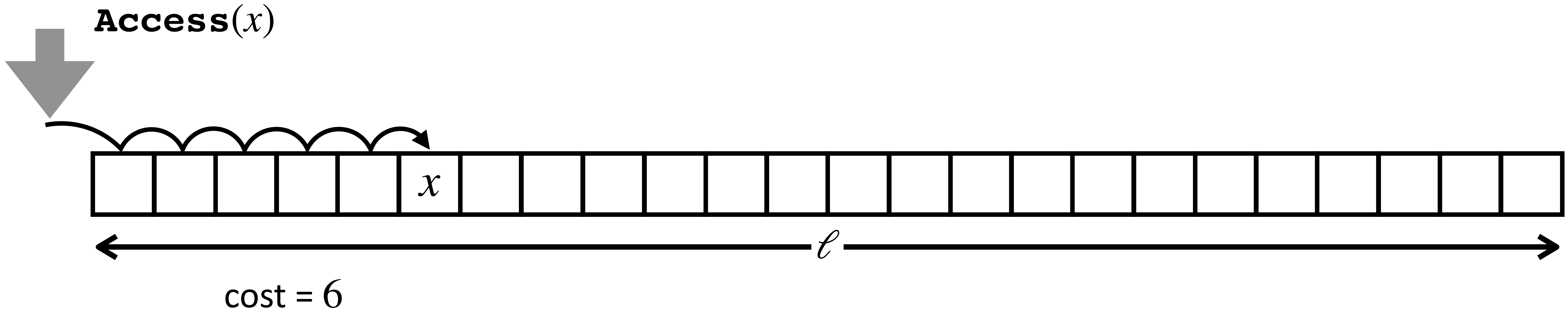
- Given a list of ℓ items
  - There is a pointer always starts from the head of the list
  - An $\texttt{Access}(x)$ request costs p if the item x is at the p-th position in the list
    - After accessing an item x, it is free to move x to any position closer to the front of the list
  - An algorithm can also move an item actively by accessing it and then moving it forward

- How to serve a sequence σ of n $\texttt{Access}$ operations?

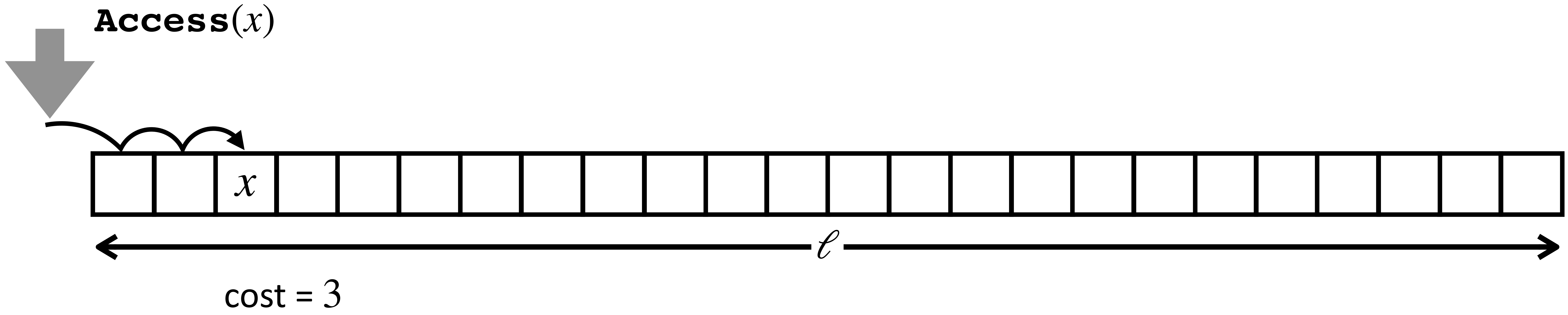
List Accessing



List Accessing

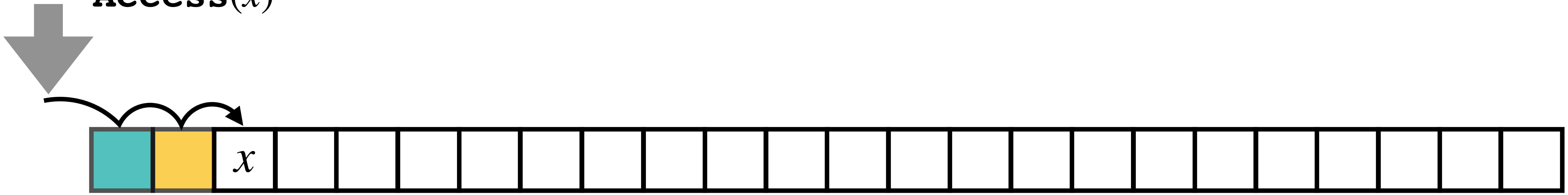


List Accessing

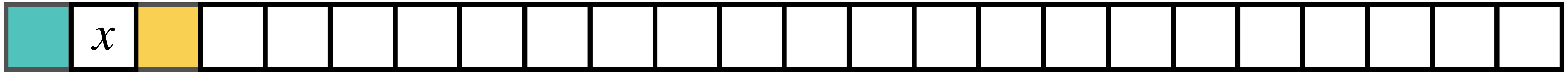


List Accessing

Access(x)



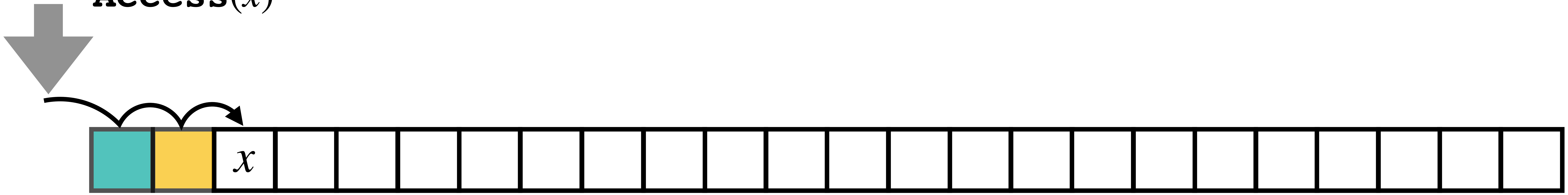
cost = 3



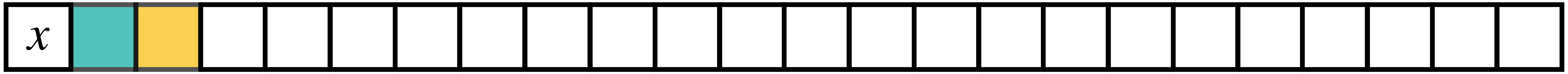
It’s free to move the accessed item closer to the front

List Accessing

Access(x)



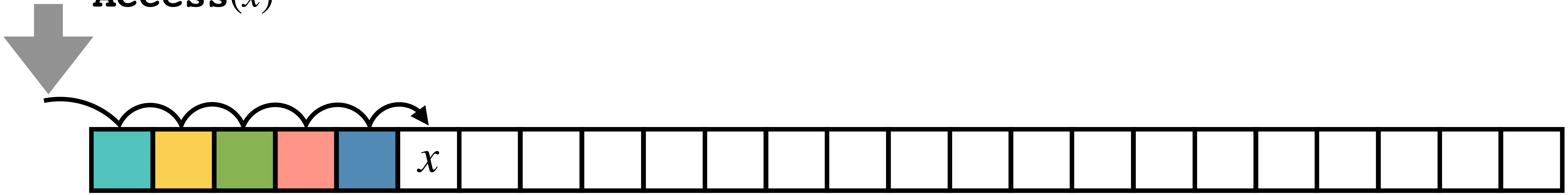
cost = 3



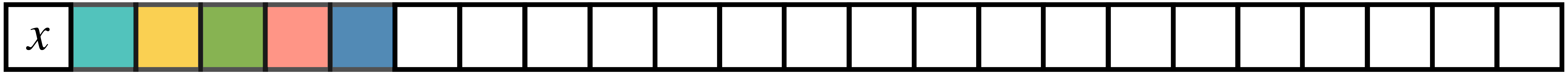
It’s free to move the accessed item closer to the front

List Accessing

Access(x)



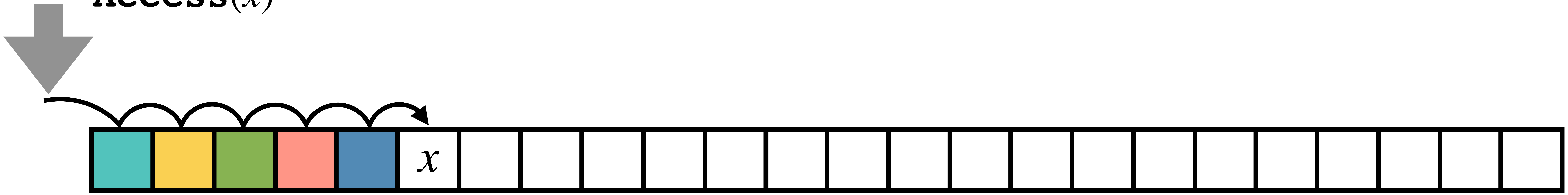
cost = 6



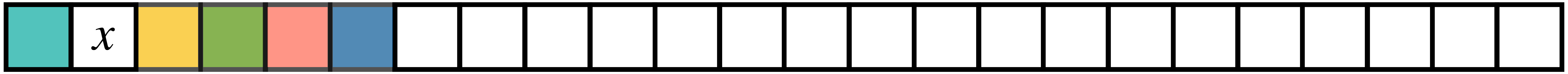
It’s free to move the accessed item closer to the front

List Accessing

Access(x)

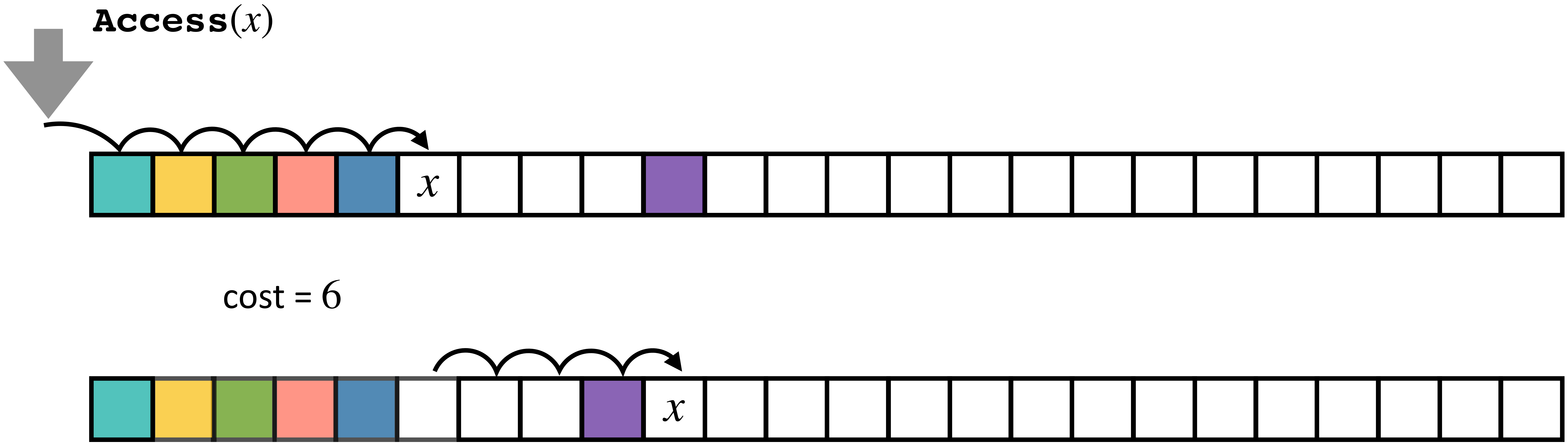


cost = 6



It’s free to move the accessed item closer to the front

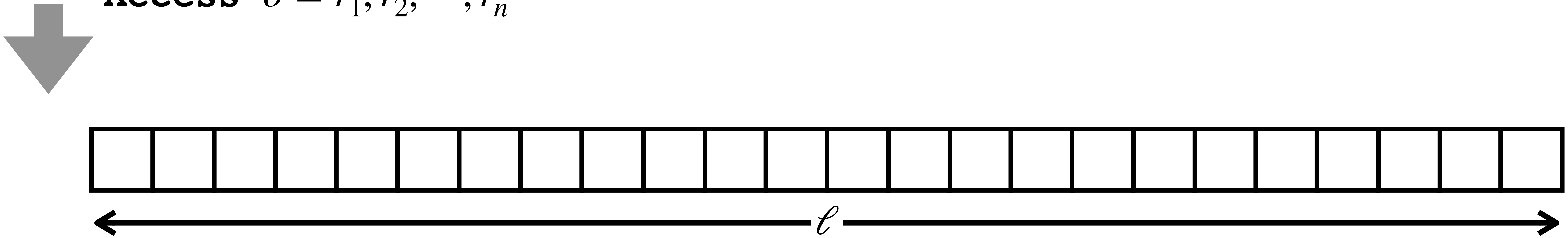
List Accessing



Moving away the accessed item with a farther item with extra cost of 4

List Accessing

`Access` $\sigma = r_1, r_2, \cdots, r_n$



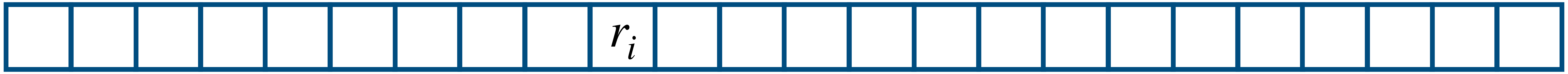
- ALG: decide whether the accessed item should be moved after accessing

Move-to-Front (MTF)

`After accessing an item, move to the front of the list`

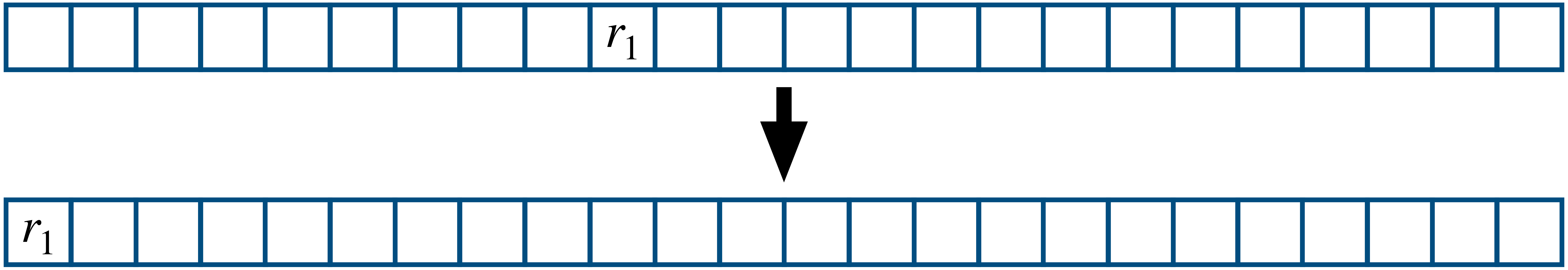
Move-to-Front (MTF)

`After accessing an item, move to the front of the list`



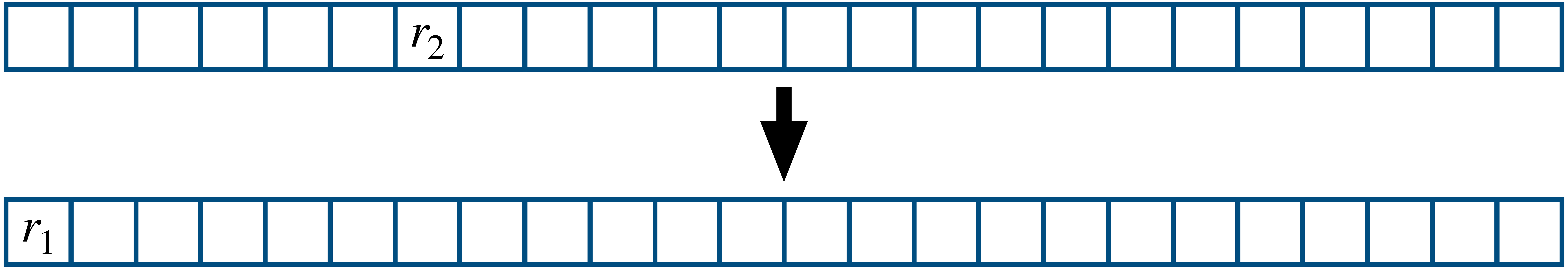
Move-to-Front (MTF)

After accessing an item, move to the front of the list



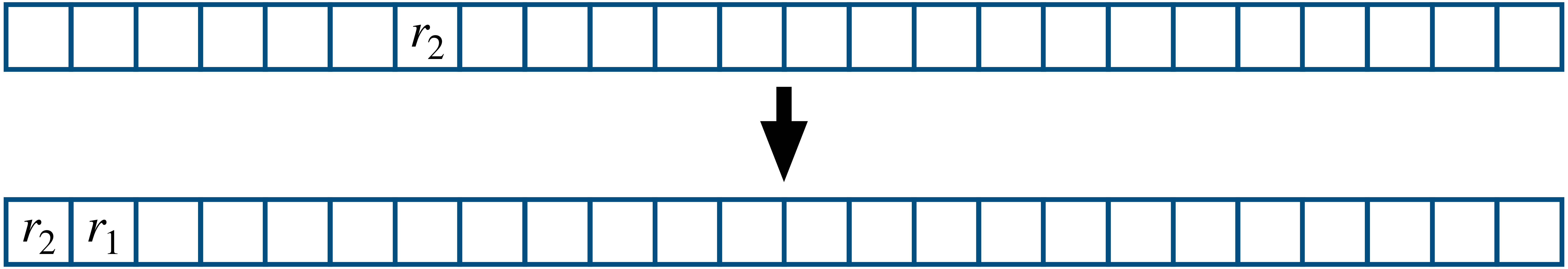
Move-to-Front (MTF)

After accessing an item, move to the front of the list



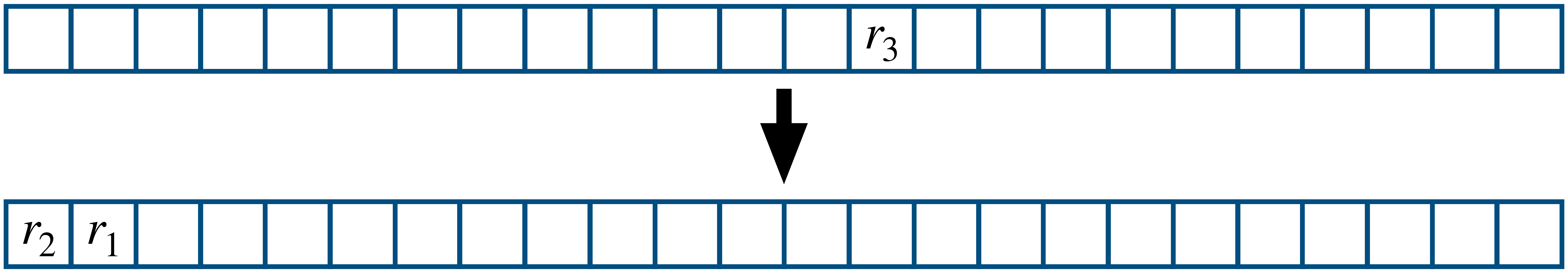
Move-to-Front (MTF)

`After accessing an item, move to the front of the list`



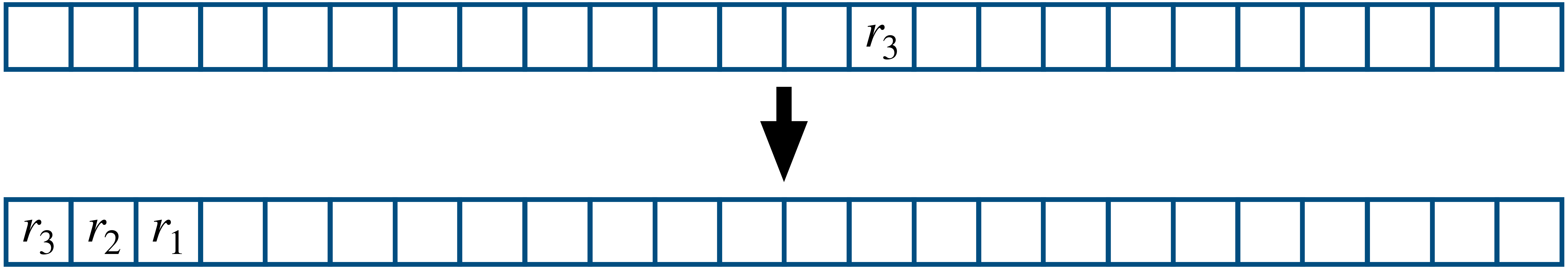
Move-to-Front (MTF)

After accessing an item, move to the front of the list



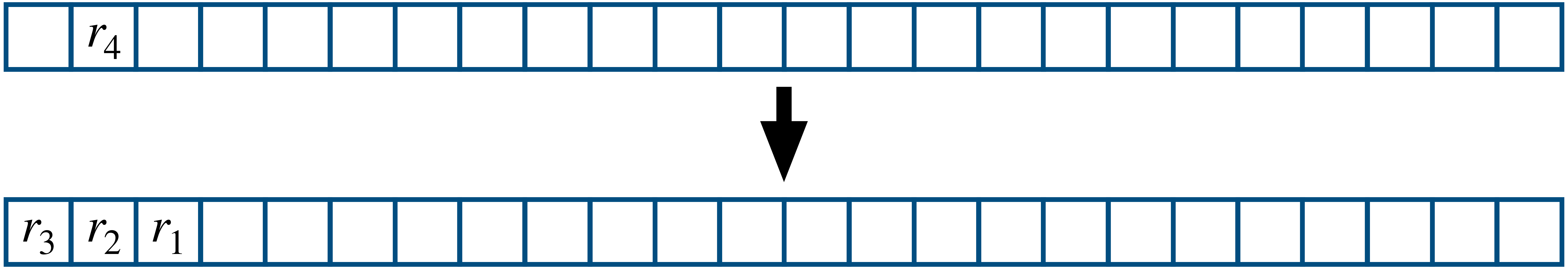
Move-to-Front (MTF)

After accessing an item, move to the front of the list



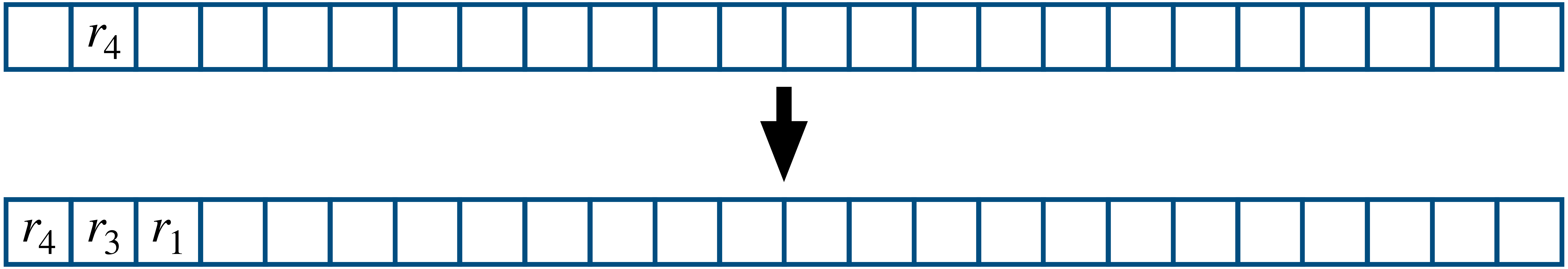
Move-to-Front (MTF)

After accessing an item, move to the front of the list



Move-to-Front (MTF)

After accessing an item, move to the front of the list



MTF is $(2 - \frac{1}{\ell})$-competitive

After accessing an item, move to the front of the list

<Proof Idea>

1. Using amortized cost $a_i = t_i + \Phi_i - \Phi_{i-1}$ to measure the cost MTF incurs for accessing r_i
   - Using a potential function Φ to measure how much different MTF is from OPT
   - $\text{MTF}(\sigma) = \sum_{i=1}^{n} t_i = \Phi_0 - \Phi_n + \sum_{i=1}^{n} a_i$
2. Show that $a_i \leq 2 \cdot \text{OPT}_i - 1$ for all i
3. $\text{MTF}(\sigma) \leq 2 \cdot \text{OPT}(\sigma) - n \leq 2 \cdot \text{OPT}(\sigma) - \frac{\text{OPT}(\sigma)}{\ell} = (2 - \frac{1}{\ell}) \cdot \text{OPT}(\sigma)$

MTF is $(2 - \frac{1}{\ell})$-competitive

After accessing an item, move to the front of the list

<Proof Idea>

1. Using amortized cost $a_i = t_i + \Phi_i - \Phi_{i-1}$ to measure the cost MTF incurs for accessing r_i
   - Using a potential function Φ to measure how much different MTF is from OPT
   - $\text{MTF}(\sigma) = \sum_{i=1}^{n} t_i = \Phi_0 - \Phi_n + \sum_{i=1}^{n} a_i$
2. Show that $a_i \leq 2 \cdot \text{OPT}_i - 1$ for all i

$\text{MTF}(\sigma) \leq \sum_{i=1}^{n} a_i \leq 2 \cdot \sum_{i=1}^{n} \text{OPT}_i - 1$

$\Phi_i \geq 0$ for all i

3. $\text{MTF}(\sigma) \leq 2 \cdot \text{OPT}(\sigma) - n \leq 2 \cdot \text{OPT}(\sigma) - \frac{\text{OPT}(\sigma)}{\ell} = (2 - \frac{1}{\ell}) \cdot \text{OPT}(\sigma)$

$OPT(\sigma) \leq \ell \cdot n$

MTF is $(2 - \frac{1}{\ell})$-competitive

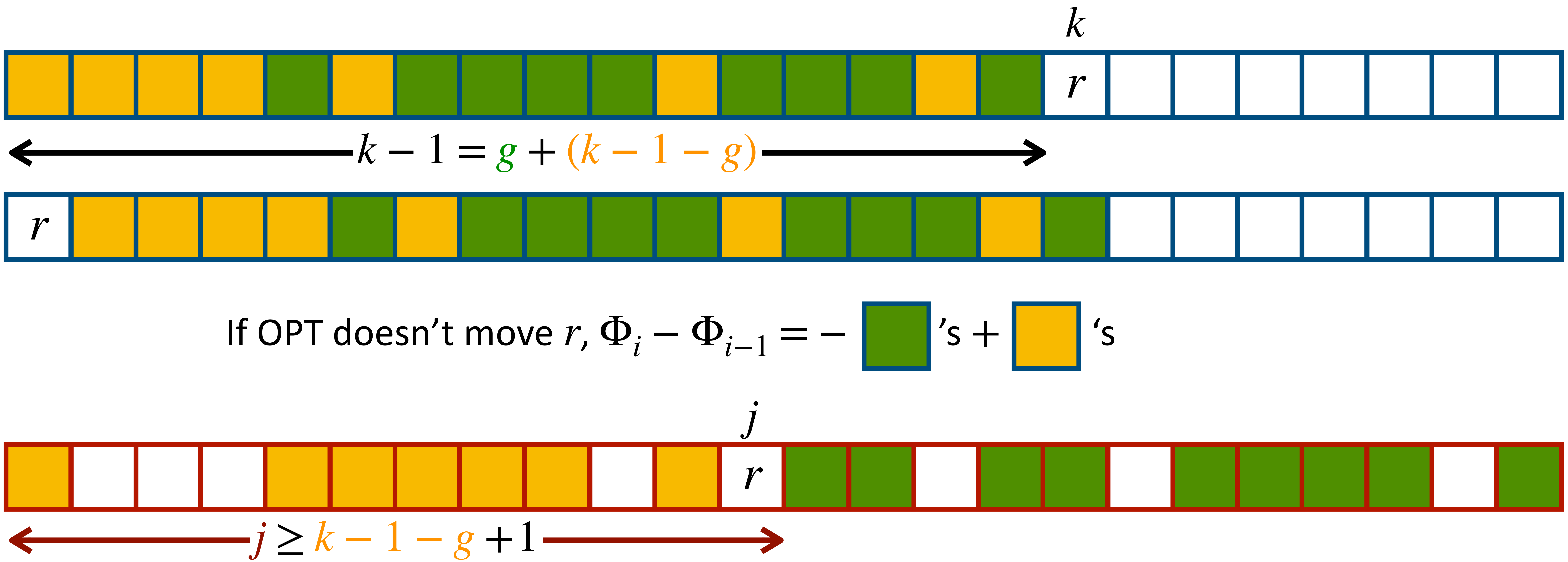
After accessing an item, move to the front of the list

1. Let $a_i = t_i + \Phi_i - \Phi_{i-1}$,
   - t_i is the actual cost that MTF incurs for processing the i-th request
   - Φ_i is a *potential function*, which maps the list configurations of MTF and OPT into a nonnegative real number just after both algorithms have finished processing the i-th request
     - $\Phi_i :=$ number of *inversions* in MTF's list with respect to OPT's list
   - $\text{MTF}(\sigma) = \sum_{i=1}^{n} t_i = \Phi_0 - \Phi_n + \sum_{i=1}^{n} a_i$

MTF is $(2 - \frac{1}{\ell})$-competitive

After accessing an item, move to the front of the list

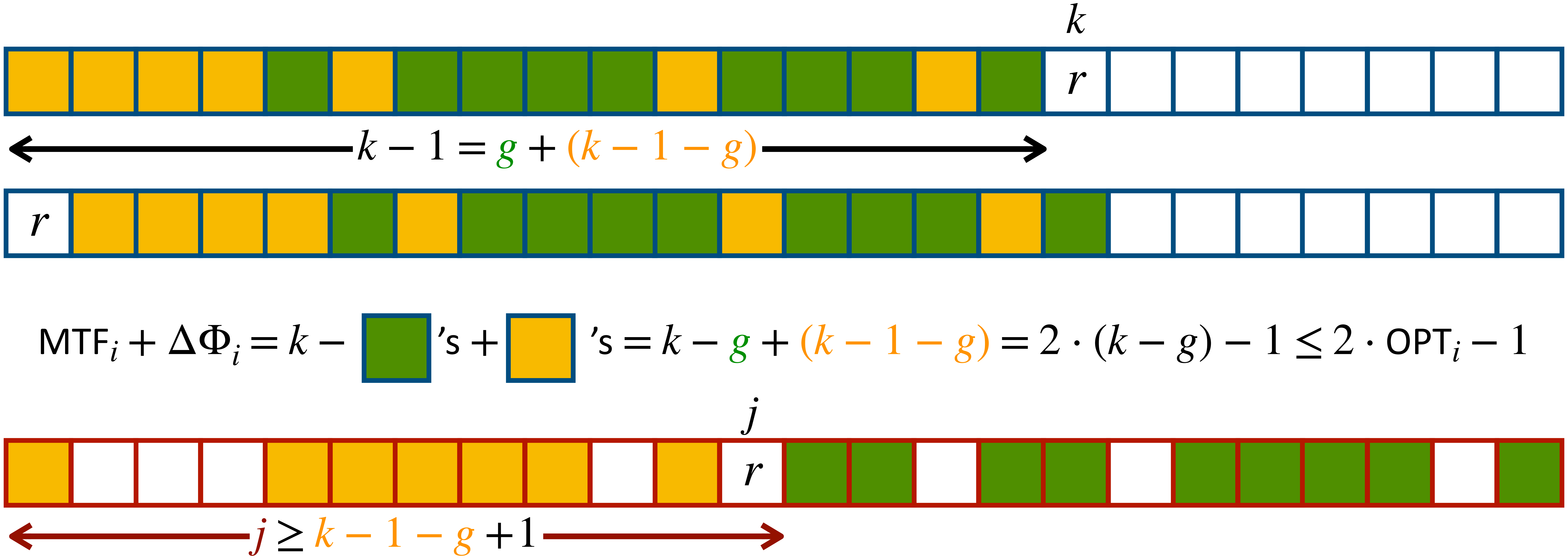
2. Claim: $a_i \leq 2 \cdot \mathrm{OPT}_i - 1$ for all i



MTF is $(2 - \frac{1}{\ell})$-competitive

After accessing an item, move to the front of the list

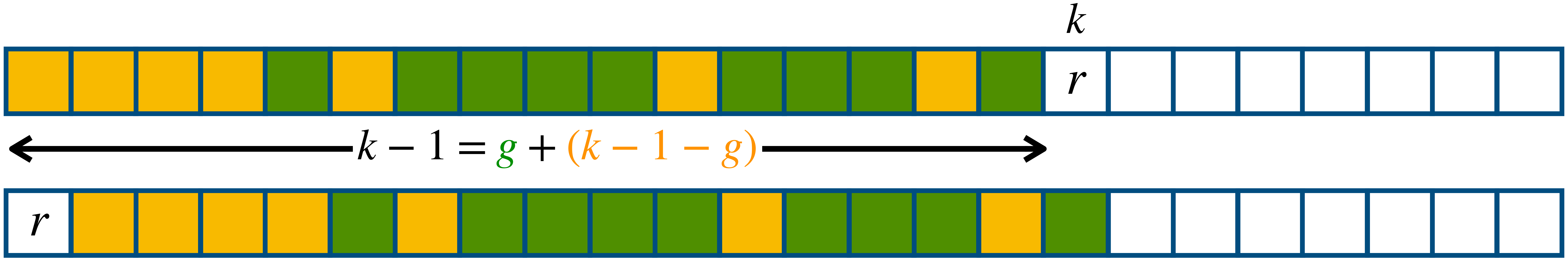
2. Claim: $a_i \leq 2 \cdot \text{OPT}_i - 1$ for all i

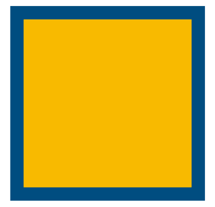


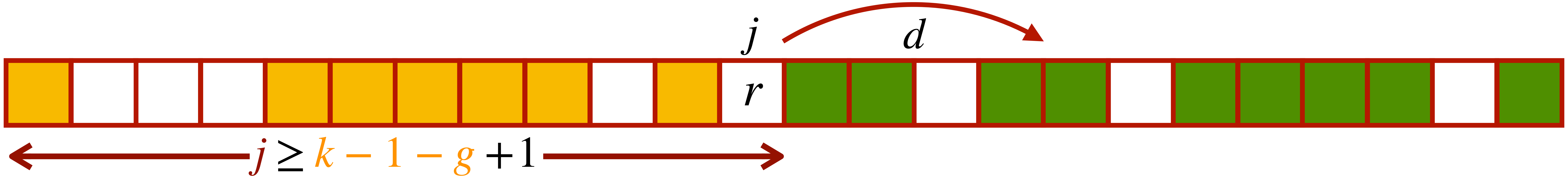
MTF is $(2 - \frac{1}{\ell})$-competitive

`After accessing an item, move to the front of the list`

2. Claim: $a_i \leq 2 \cdot \text{OPT}_i - 1$ for all i



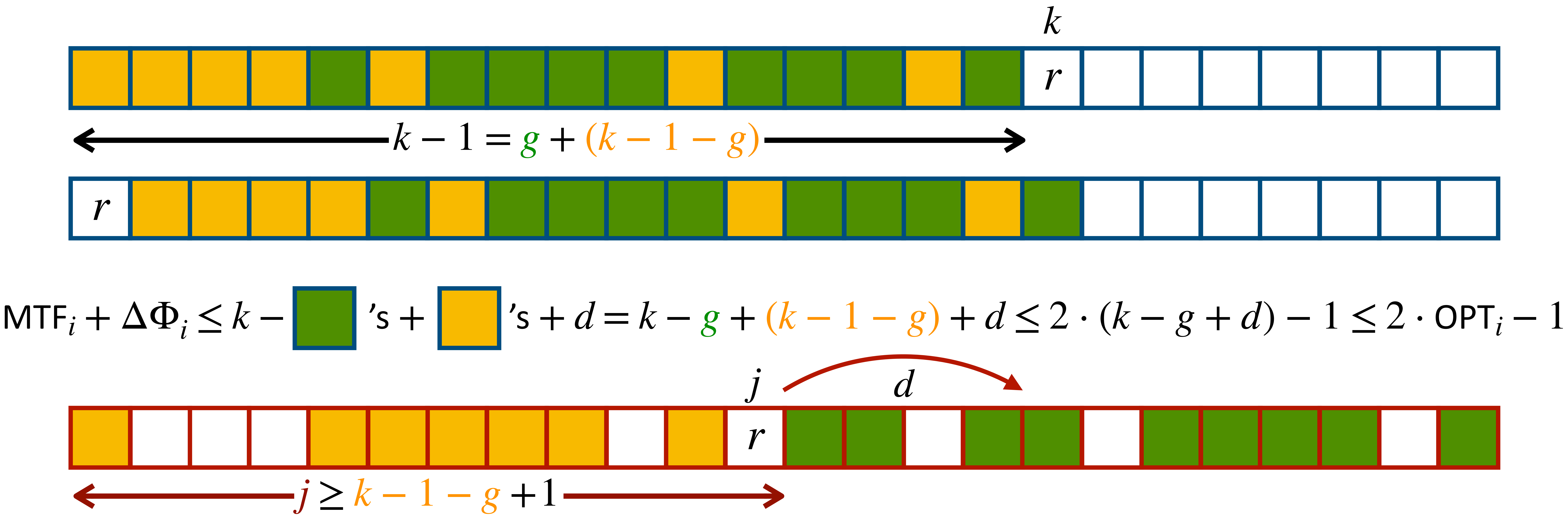
If OPT moves r away for d positions, $\Phi_i - \Phi_{i-1} \leq -$  's +  's $+ d$



MTF is $(2 - \frac{1}{\ell})$-competitive

After accessing an item, move to the front of the list

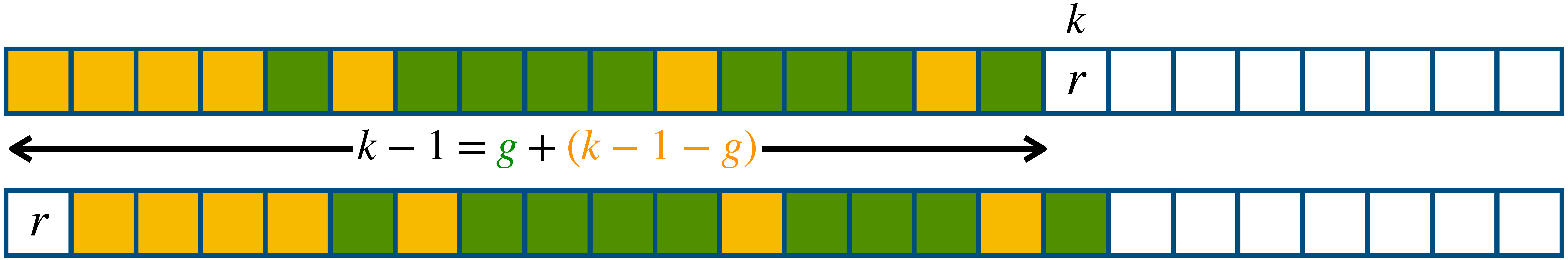
2. Claim: $a_i \le 2 \cdot \mathrm{OPT}_i - 1$ for all i



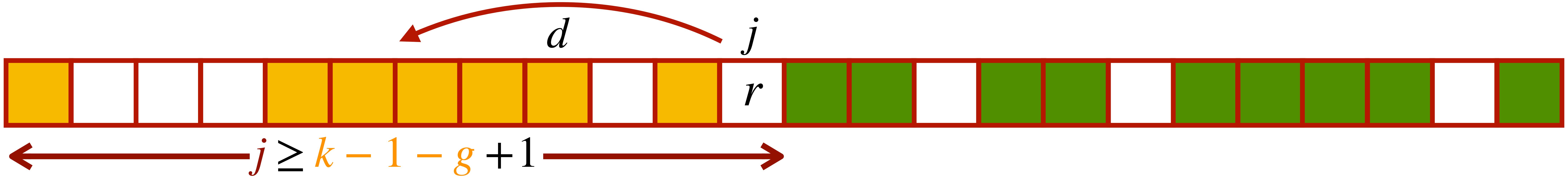
MTF is $(2 - \frac{1}{\ell})$-competitive

After accessing an item, move to the front of the list

2. Claim: $a_i \leq 2 \cdot \text{OPT}_i - 1$ for all i



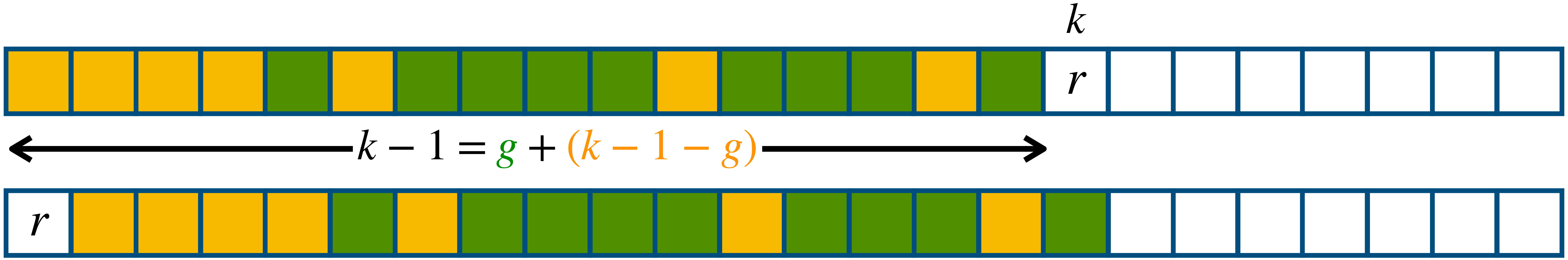
If OPT moves r forward for d positions, $\Phi_i - \Phi_{i-1} \leq -\,\square$'s $+\,\square$'s



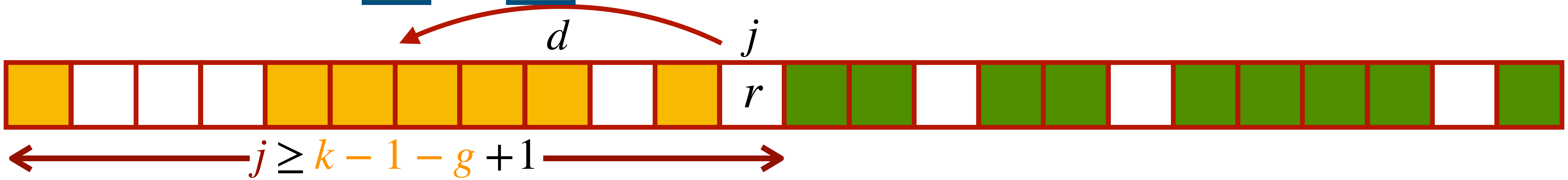
MTF is $(2 - \frac{1}{\ell})$-competitive

After accessing an item, move to the front of the list

2. Claim: $a_i \leq 2 \cdot \text{OPT}_i - 1$ for all i



$\text{MTF}_i + \Delta\Phi_i \leq k -$ [green box]'s + [yellow box]'s $= k - g + (k - 1 - g) = 2 \cdot (k - g) - 1 \leq 2 \cdot \text{OPT}_i - 1$



Potential function method

List Accessing is at least $(2 - \frac{1}{\ell + 1})$-competitive

- Adversary σ: given any ALG, always access the last item in its list
  - Let $n = |\sigma|$, ALG$(\sigma) = \ell \cdot n$

List Accessing is at least $(2 - \frac{1}{\ell + 1})$-competitive

$\ell\,!$ static algorithms: first get one of the $\ell\,!$ possible permutations of the items using $O(\ell^2)$ paid movings

When access x, there are $(\ell - 1)!$ static algorithms that costs i

Total cost of n requests on all static algorithms $= n \cdot \sum_{i=1}^{\ell} i \cdot (\ell - 1)! \Rightarrow$ on average, OPT $\leq \frac{n \cdot \sum_{i=1}^{\ell} i \cdot (\ell - 1)!}{\ell\,!}$

$$\text{OPT}(\sigma) \leq \frac{n(\ell + 1)}{2} + \ell^2$$

List Accessing is at least $(2 - \frac{1}{\ell + 1})$-competitive

$\ell\,!$ static algorithms: first get one of the $\ell\,!$ possible permutations of the items using $O(\ell^2)$ paid movings

$$\frac{\text{ALG}(\sigma)}{\text{OPT}(\sigma)} \geq \frac{2\ell}{\ell + 1}$$

$$\text{ALG}(\sigma) = \ell \cdot n \qquad \text{OPT}(\sigma) \leq \frac{n(\ell + 1)}{2} + \ell^2$$

List Accessing is at least $(2 - \frac{1}{\ell + 1})$-competitive

- Consider $\ell\,!$ static algorithms that never change the order of the list, each starts at one of the $\ell\,!$ permutation of ℓ elements (which can be formed within at most $O(\ell^2)$ swaps)
  - In total, each `Access`(r_i) costs $\sum_{i=1}^{\ell} i \cdot (\ell - 1)!$ in all the static algorithm, and the total cost of n accessing $= n \cdot \sum_{i=1}^{\ell} i \cdot (\ell - 1)!$
    - On average, the cost of n accessing on one static algorithm is $\frac{n \cdot \sum_{i=1}^{\ell} i \cdot (\ell - 1)!}{\ell\,!}$
      - There is at least one static algorithm with total cost $\leq \frac{n \cdot \sum_{i=1}^{\ell} i \cdot (\ell - 1)!}{\ell\,!}$
      - OPT cannot be worst than that static algorithm and has cost $\leq \frac{n \cdot \sum_{i=1}^{\ell} i \cdot (\ell - 1)!}{\ell\,!} + \ell^2$
- $\frac{\text{ALG}(\sigma)}{\text{OPT}(\sigma)} \geq \frac{\ell \cdot n}{\frac{n \cdot \sum_{i=1}^{\ell} i \cdot (\ell - 1)!}{\ell\,!} + \ell^2}$, when $n \to \infty$, $\frac{\text{ALG}(\sigma)}{\text{OPT}(\sigma)} \geq \frac{2\ell^2 n}{(\ell^2 + \ell)n} = 2 - \frac{2}{\ell + 1}$

Bound by average

- A useful technique to get the lower bound of the optimal strategy on the instance is to set a set of (offline) algorithms
  - Calculate the total cost incurred by these algorithms
  - The optimal algorithm must be as good as the average cost

Outline

- Problem lower bound and “best” online algorithms
  - Ski-rental
  - Bin packing
  - Paging

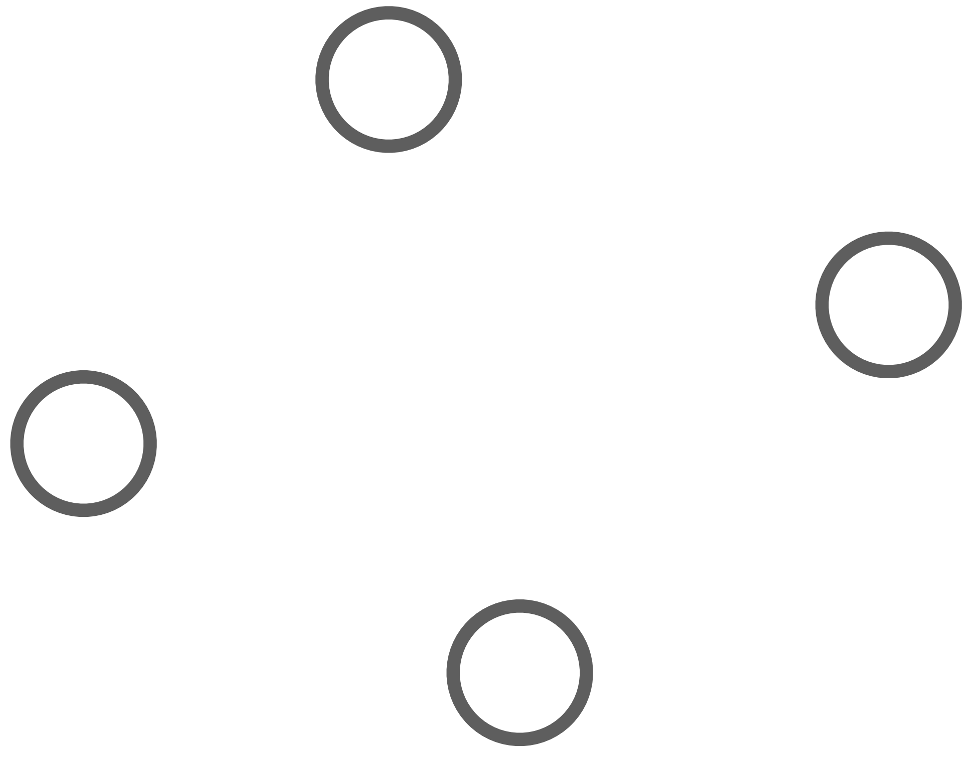
- Bounding difference to the optimal solution — potential function
  - List accessing
  - k-server

k-Server

- On a metric space $(\mathscr{M}, d)$

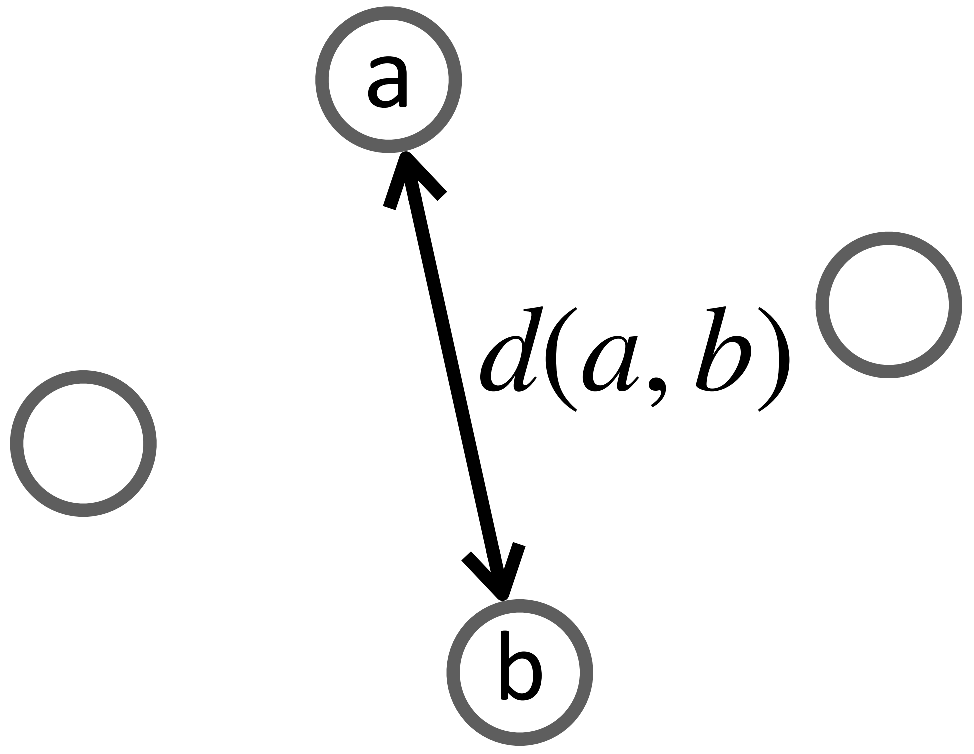
k-Server

- On a metric space $(\mathscr{M}, d)$



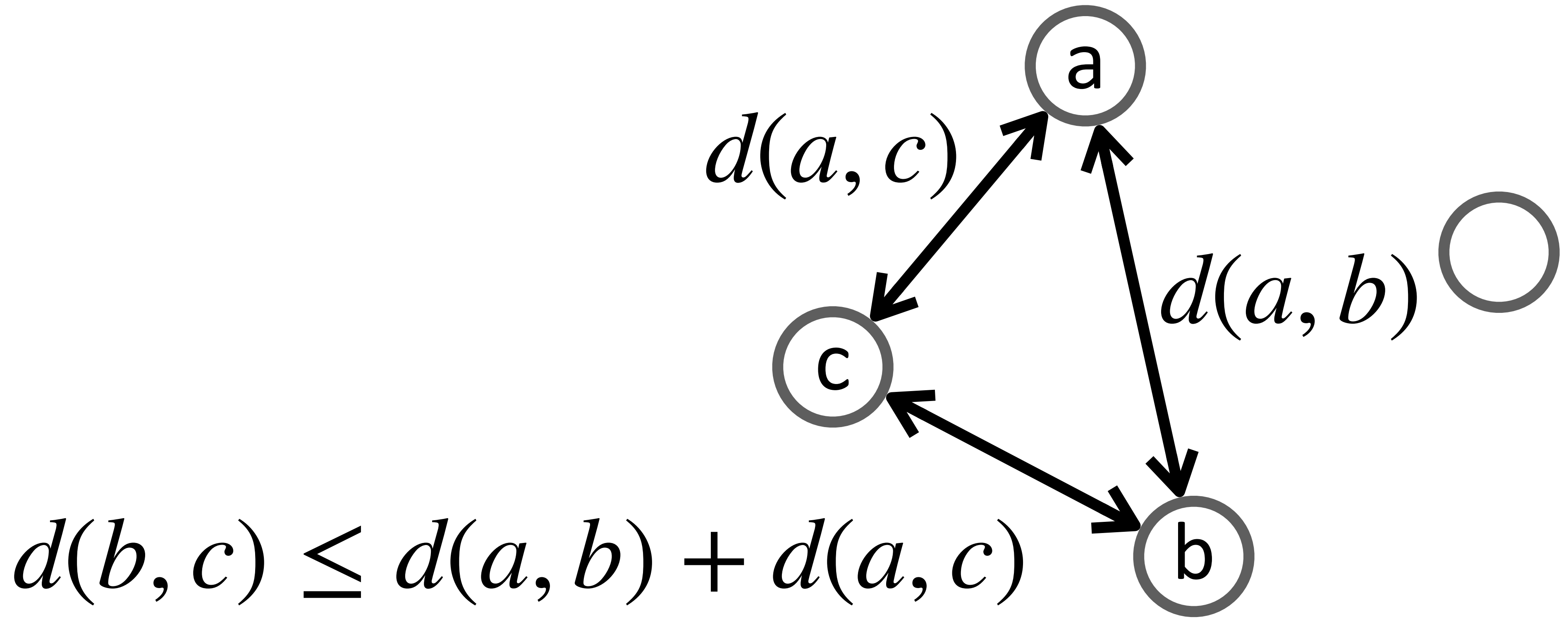
k-Server

- On a metric space $(\mathscr{M}, d)$



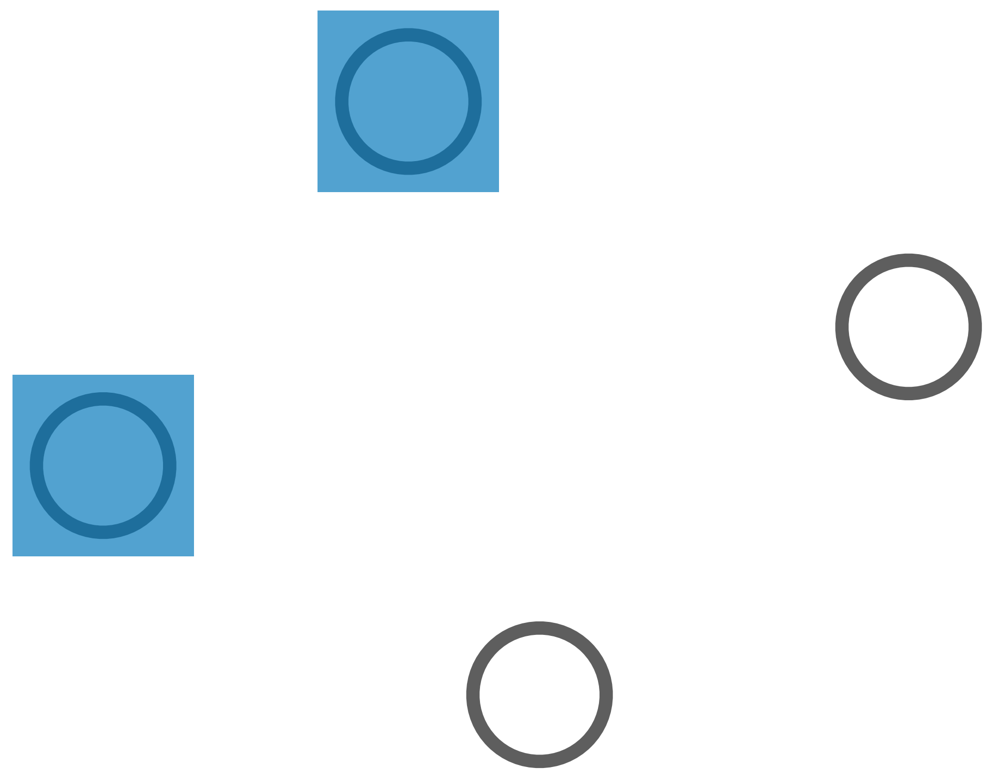
k-Server

- On a metric space $(\mathscr{M}, d)$



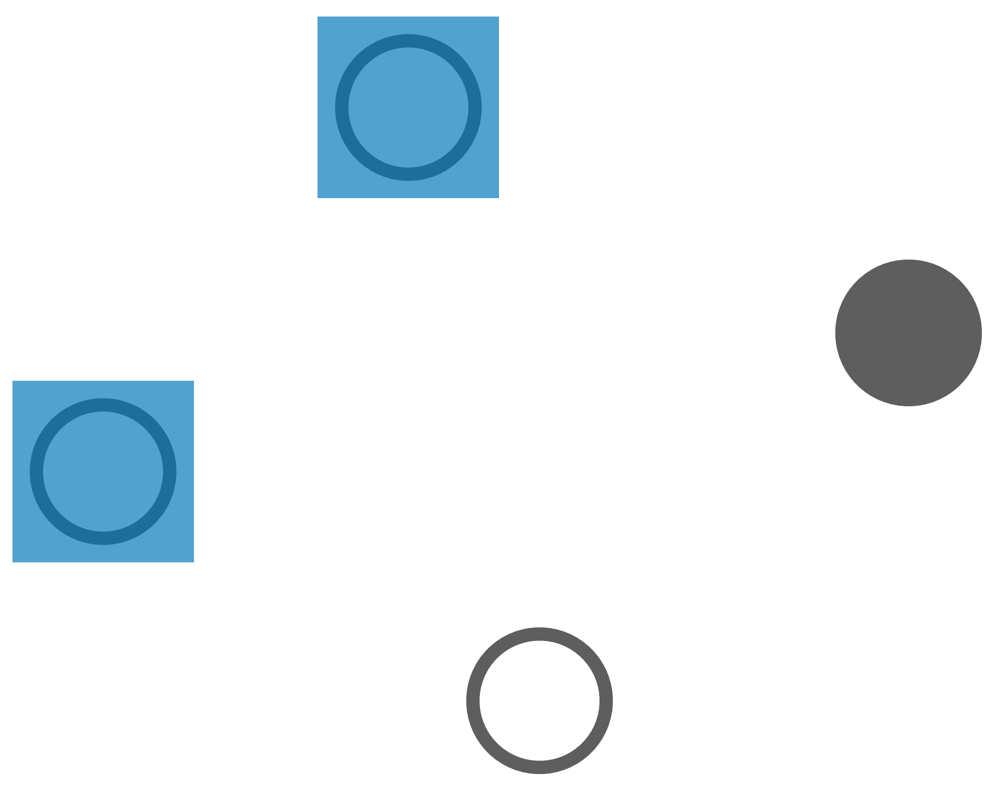
k-Server

- On a metric space $(\mathscr{M}, d)$, there are k servers sitting at some points in $\mathscr{M}$



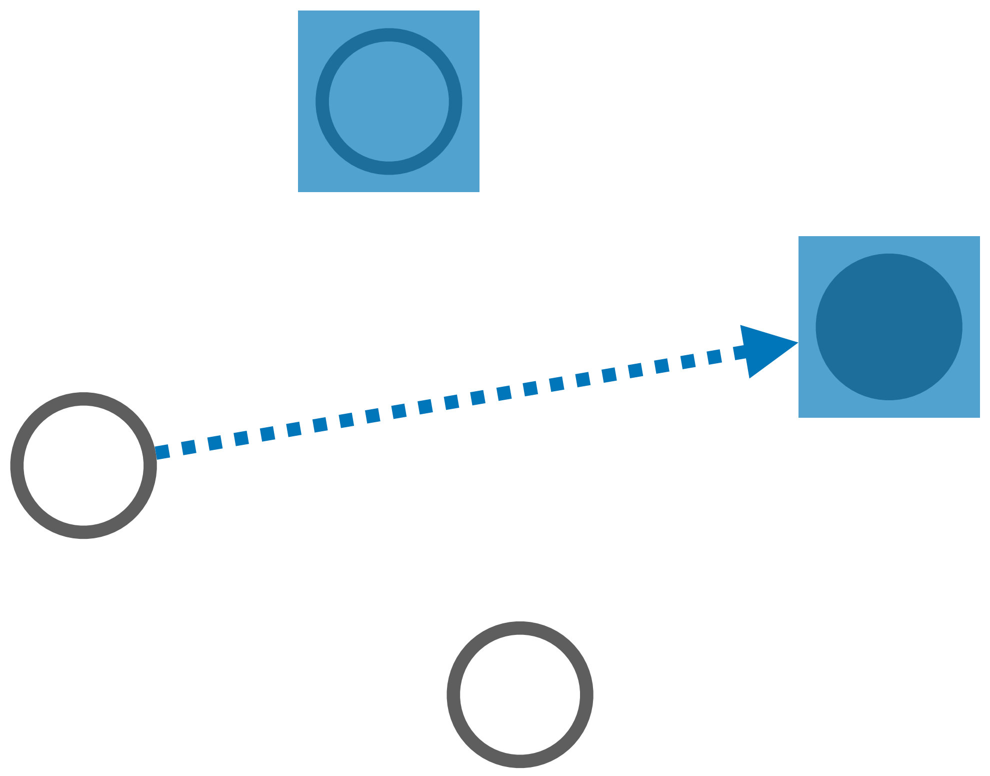
k-Server

- On a metric space $(\mathscr{M}, d)$, there are k servers sitting at some points in $\mathscr{M}$
  - A sequence σ of requests, each is a point in $\mathscr{M}$



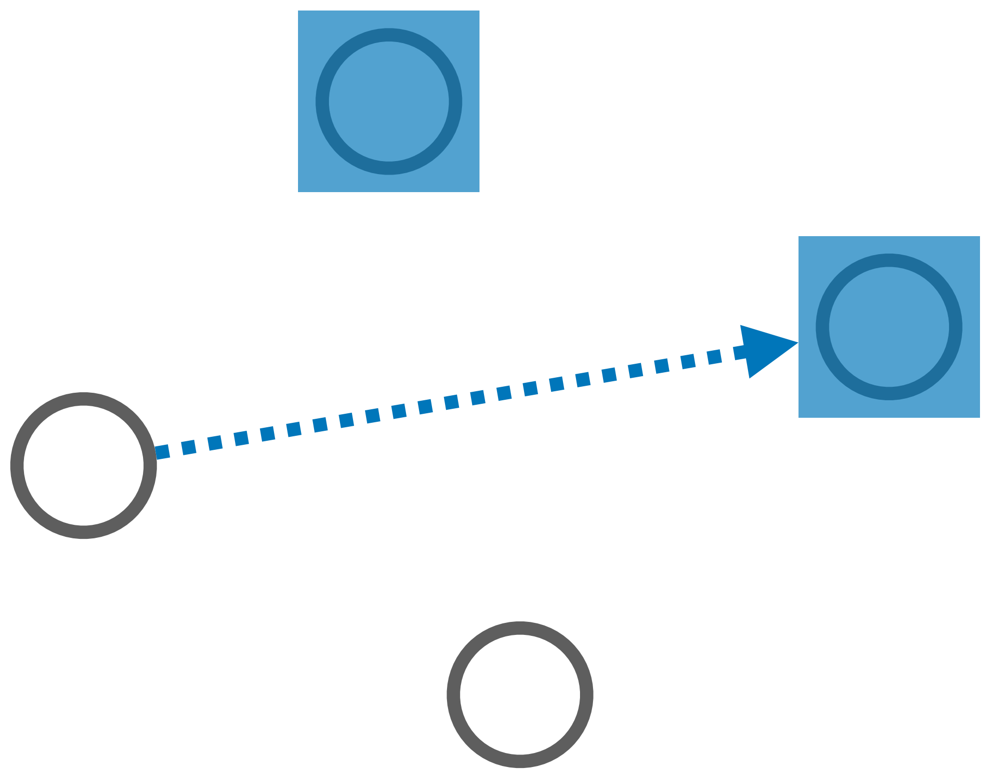
k-Server

- On a metric space $(\mathscr{M}, d)$, there are k servers sitting at some points in $\mathscr{M}$
  - A sequence σ of requests, each is a point in $\mathscr{M}$
  - Once a request arises at a point $p \in \mathscr{M}$, the algorithm has to send at least one server to p and serve the request



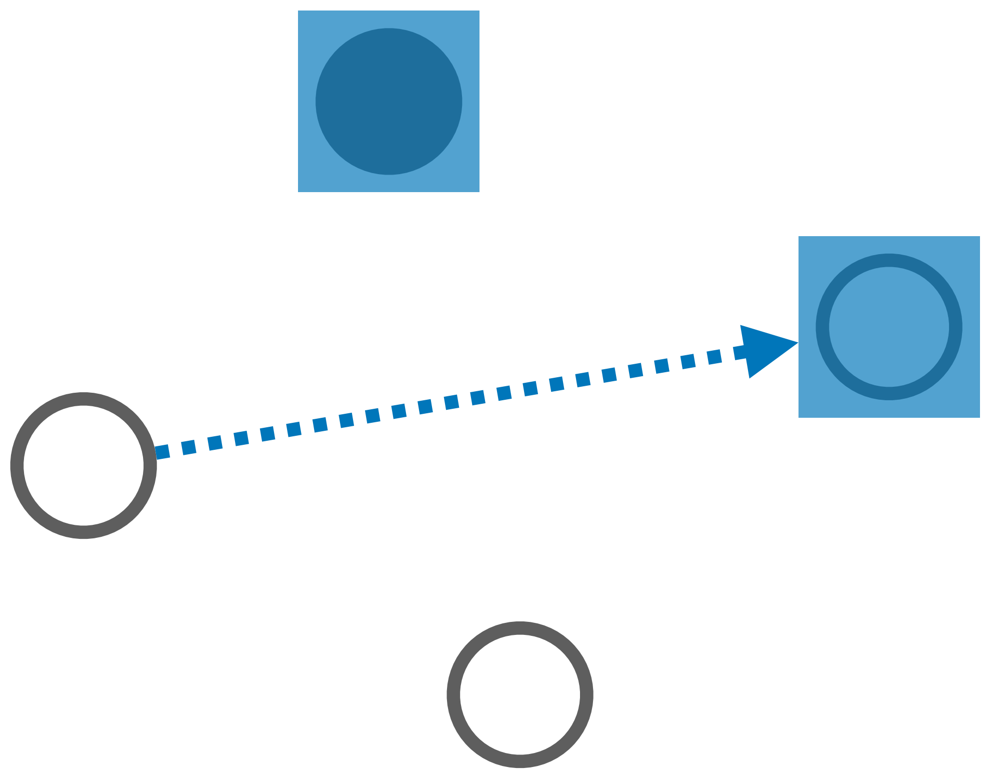
k-Server

- On a metric space $(\mathscr{M}, d)$, there are k servers sitting at some points in $\mathscr{M}$
  - A sequence σ of requests, each is a point in $\mathscr{M}$
  - Once a request arises at a point $p \in \mathscr{M}$, the algorithm has to send at least one server to p and serve the request



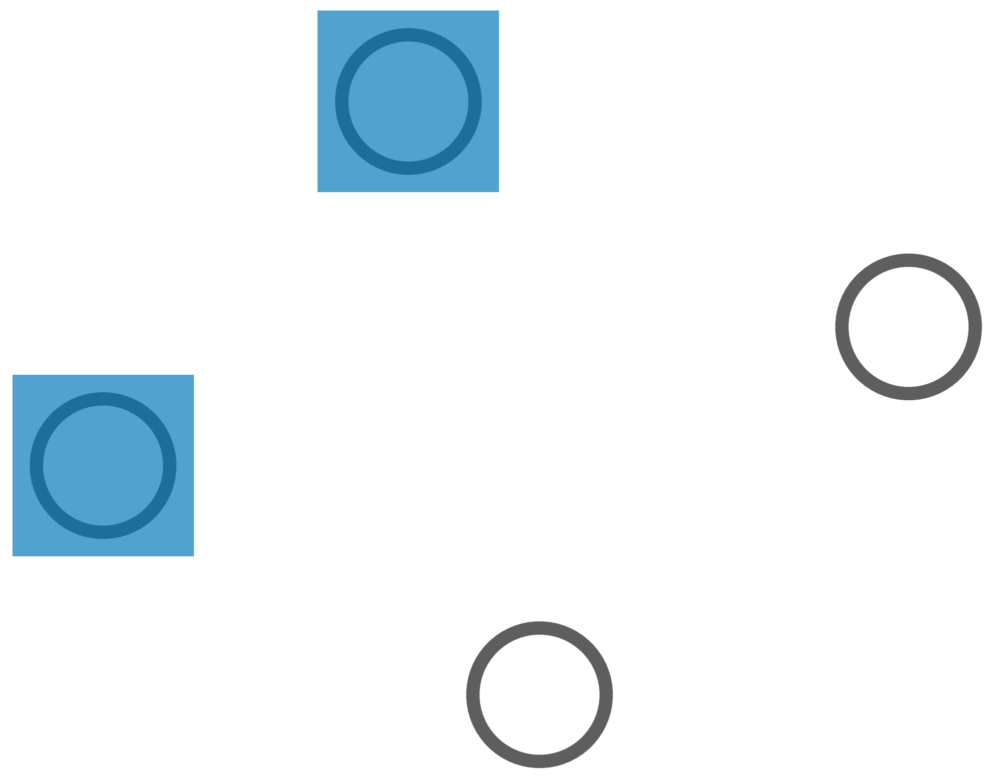
k-Server

- On a metric space $(\mathscr{M}, d)$, there are k servers sitting at some points in $\mathscr{M}$
  - A sequence σ of requests, each is a point in $\mathscr{M}$
  - Once a request arises at a point $p \in \mathscr{M}$, the algorithm has to send at least one server to p and serve the request



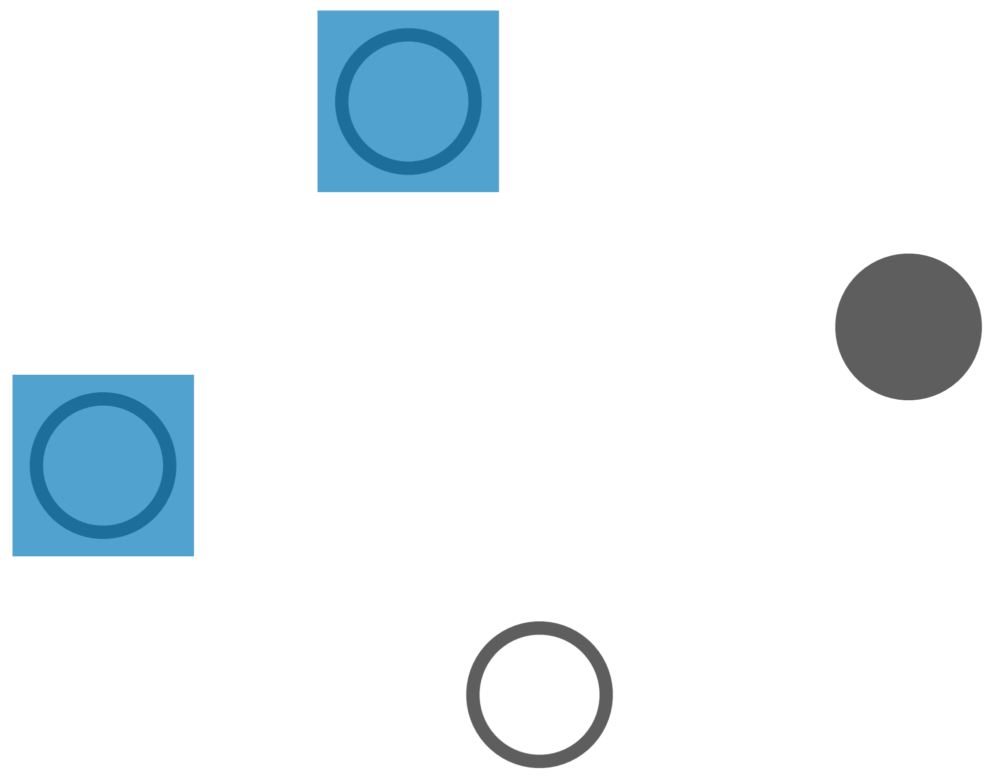
k-Server

- On a metric space $(\mathscr{M}, d)$, there are k servers sitting at some points in $\mathscr{M}$
  - A sequence σ of requests, each is a point in $\mathscr{M}$
  - Once a request arises at a point $p \in \mathscr{M}$, the algorithm has to send at least one server to p and serve the request
  - The goal is to minimize the total traveling distance of all servers



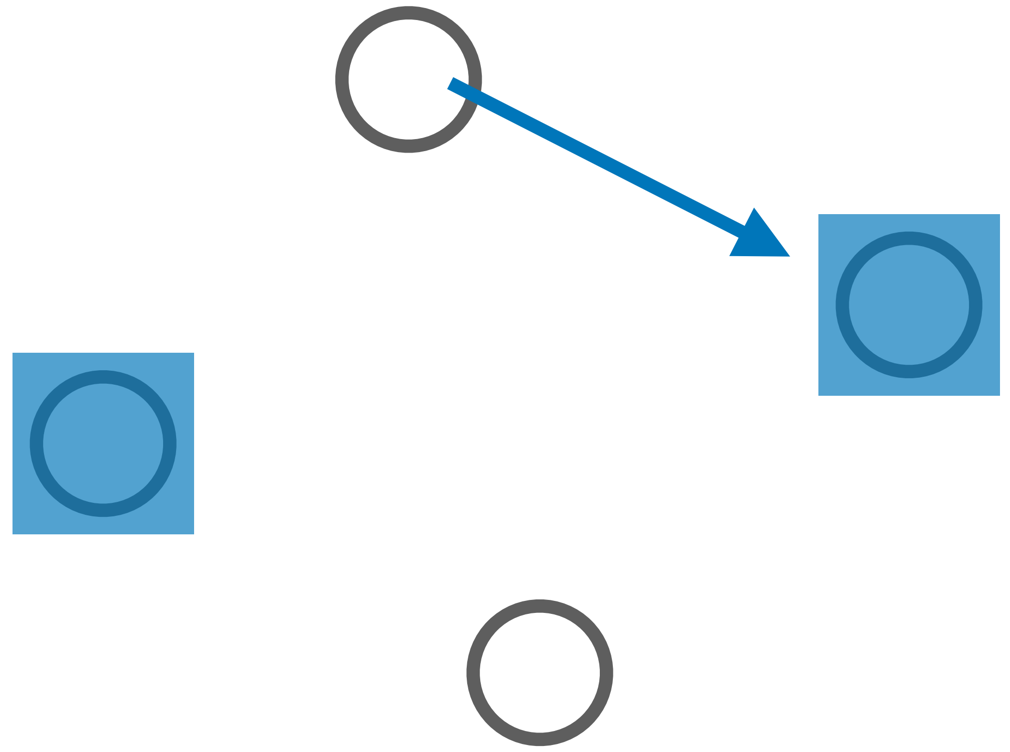
k-Server

- On a metric space $(\mathscr{M}, d)$, there are k servers sitting at some points in $\mathscr{M}$
  - A sequence σ of requests, each is a point in $\mathscr{M}$
  - Once a request arises at a point $p \in \mathscr{M}$, the algorithm has to send at least one server to p and serve the request
  - The goal is to minimize the total traveling distance of all servers



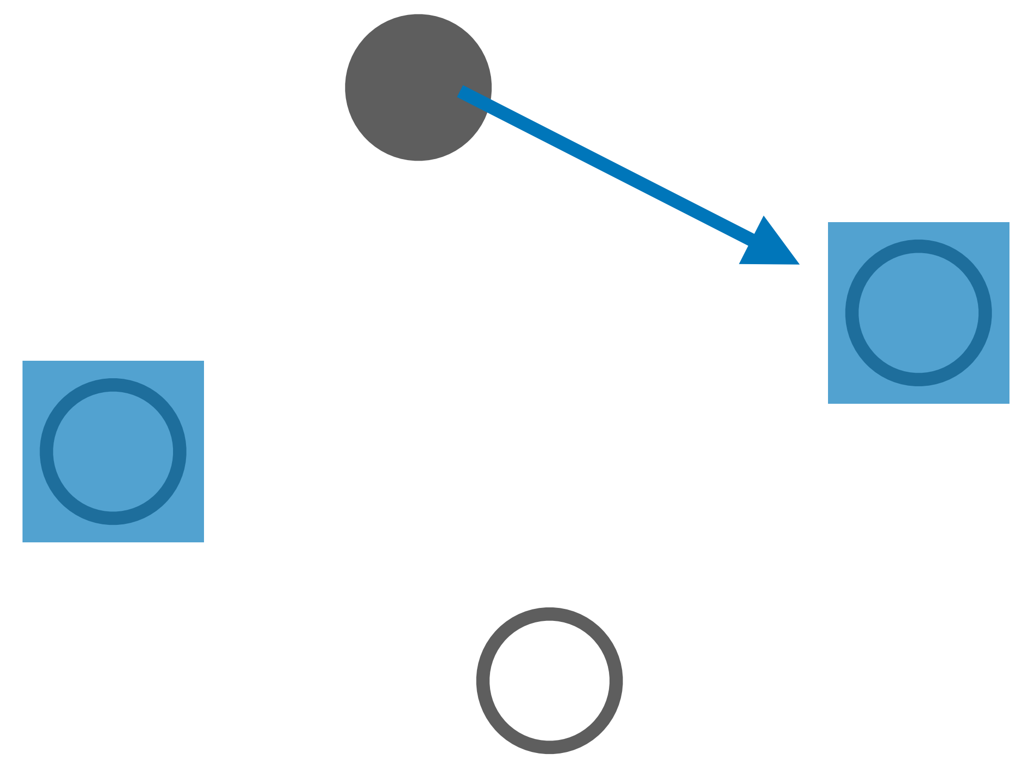
k-Server

- On a metric space $(\mathscr{M}, d)$, there are k servers sitting at some points in $\mathscr{M}$
  - A sequence σ of requests, each is a point in $\mathscr{M}$
  - Once a request arises at a point $p \in \mathscr{M}$, the algorithm has to send at least one server to p and serve the request
  - The goal is to minimize the total traveling distance of all servers



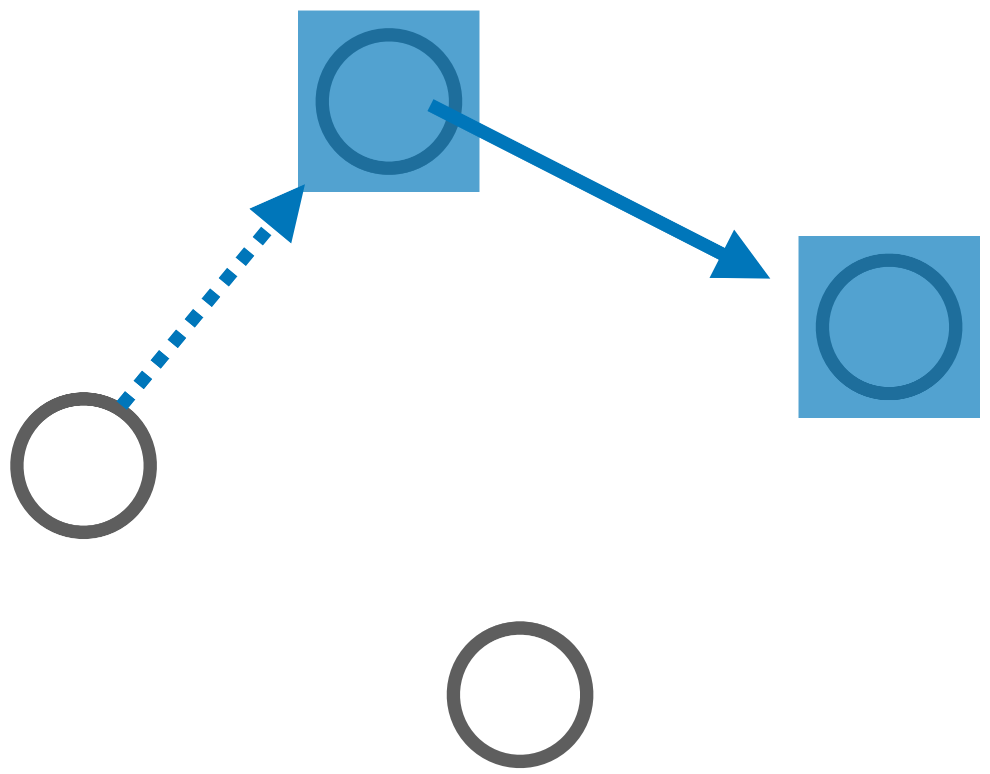
k-Server

- On a metric space $(\mathscr{M}, d)$, there are k servers sitting at some points in $\mathscr{M}$
  - A sequence σ of requests, each is a point in $\mathscr{M}$
  - Once a request arises at a point $p \in \mathscr{M}$, the algorithm has to send at least one server to p and serve the request
  - The goal is to minimize the total traveling distance of all servers



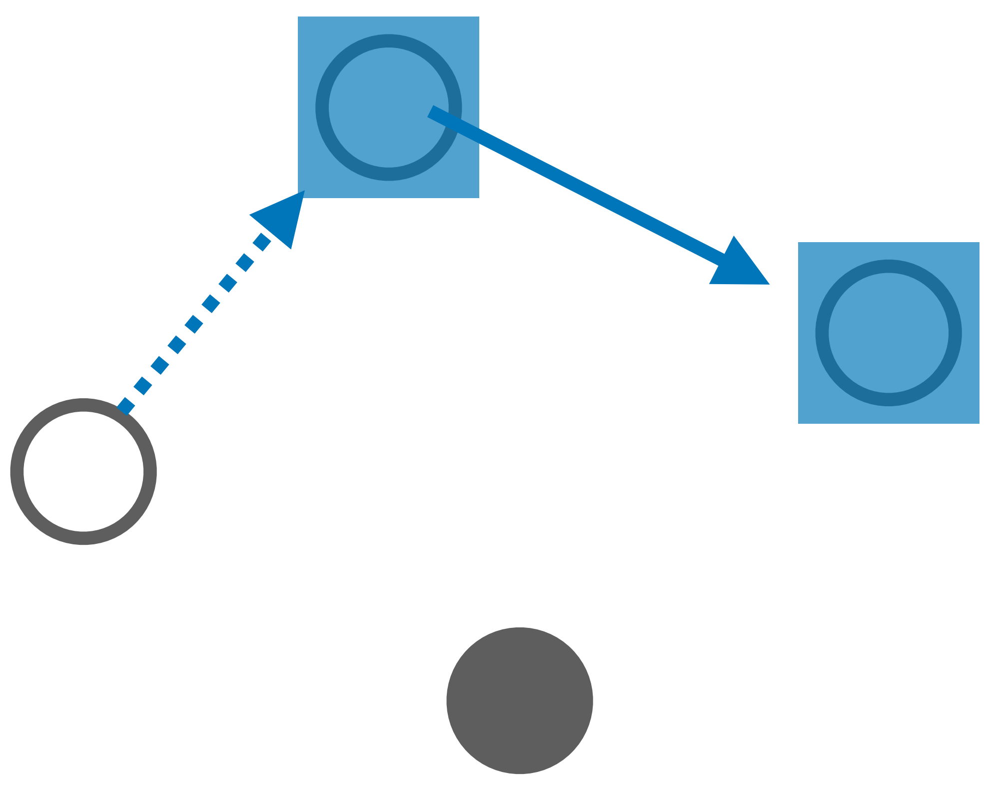
k-Server

- On a metric space $(\mathscr{M}, d)$, there are k servers sitting at some points in $\mathscr{M}$
  - A sequence σ of requests, each is a point in $\mathscr{M}$
  - Once a request arises at a point $p \in \mathscr{M}$, the algorithm has to send at least one server to p and serve the request
  - The goal is to minimize the total traveling distance of all servers



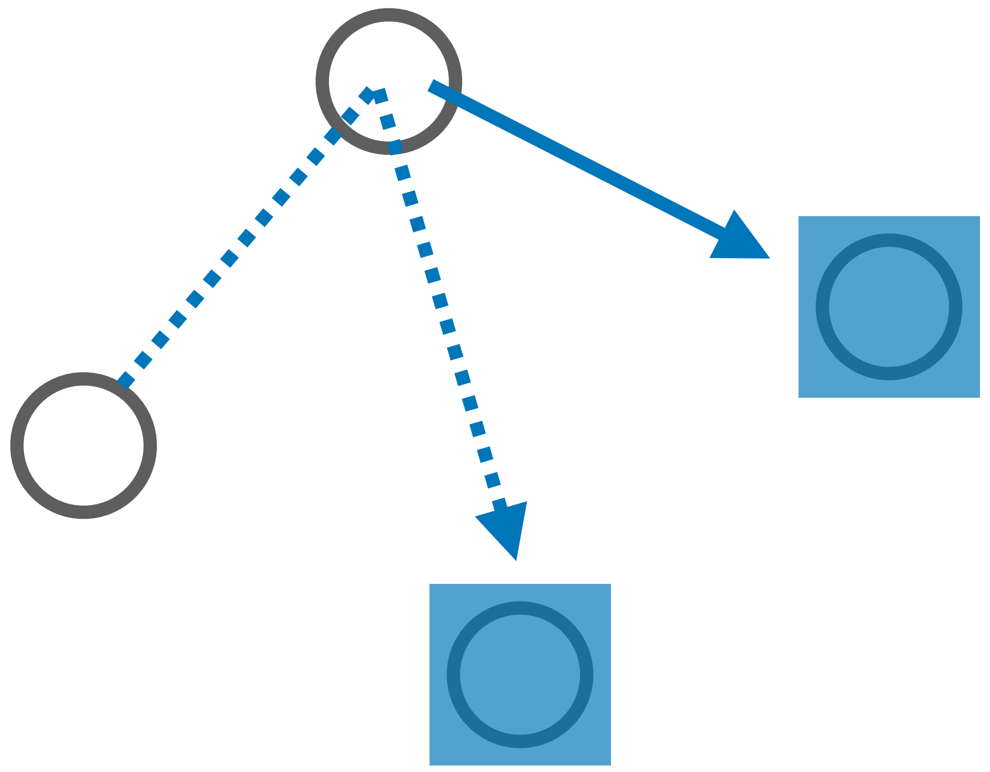
k-Server

- On a metric space $(\mathscr{M}, d)$, there are k servers sitting at some points in $\mathscr{M}$
  - A sequence σ of requests, each is a point in $\mathscr{M}$
  - Once a request arises at a point $p \in \mathscr{M}$, the algorithm has to send at least one server to p and serve the request
  - The goal is to minimize the total traveling distance of all servers



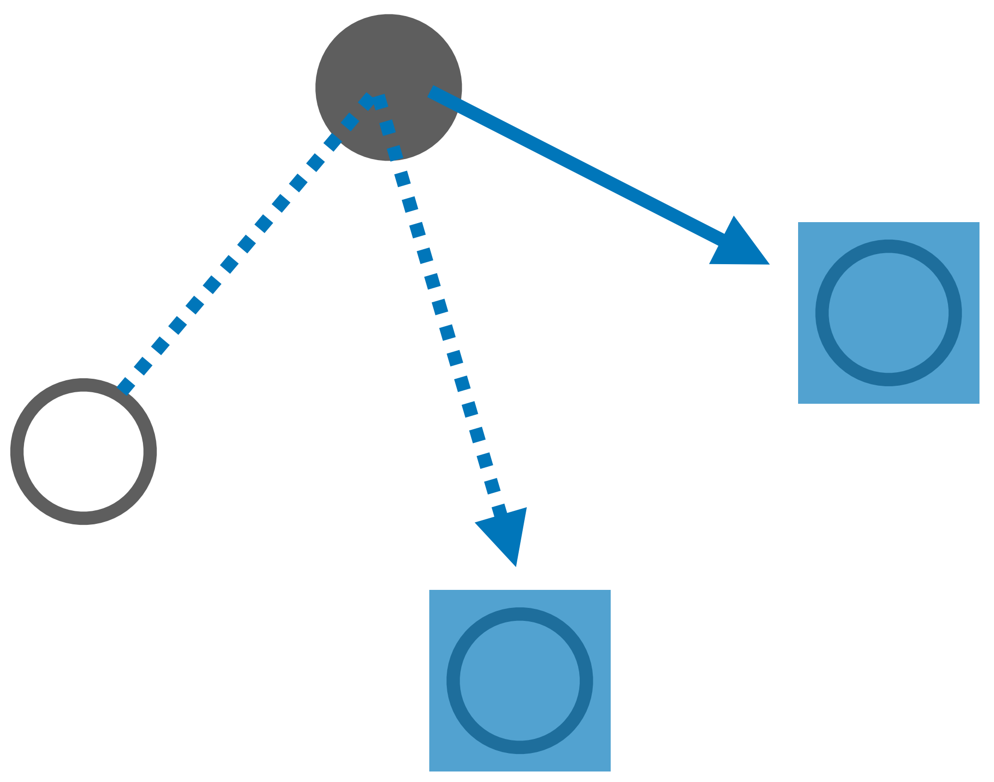
k-Server

- On a metric space $(\mathscr{M}, d)$, there are k servers sitting at some points in $\mathscr{M}$
  - A sequence σ of requests, each is a point in $\mathscr{M}$
  - Once a request arises at a point $p \in \mathscr{M}$, the algorithm has to send at least one server to p and serve the request
  - The goal is to minimize the total traveling distance of all servers



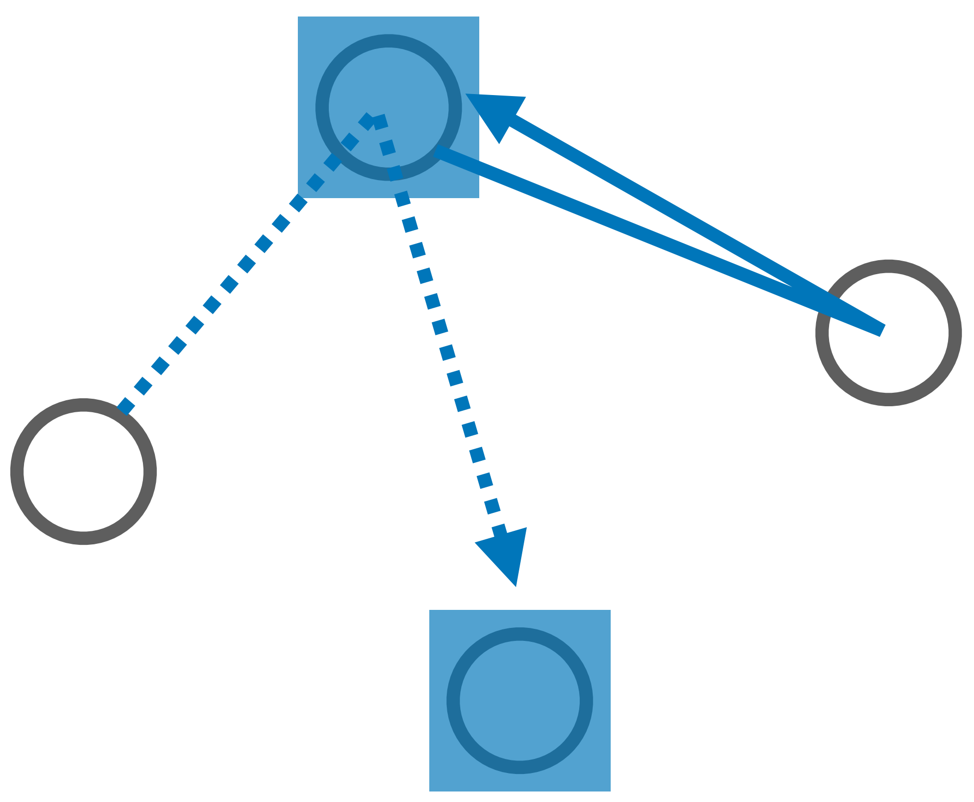
k-Server

- On a metric space $(\mathscr{M}, d)$, there are k servers sitting at some points in $\mathscr{M}$
  - A sequence σ of requests, each is a point in $\mathscr{M}$
  - Once a request arises at a point $p \in \mathscr{M}$, the algorithm has to send at least one server to p and serve the request
  - The goal is to minimize the total traveling distance of all servers



k-Server

- On a metric space $(\mathscr{M}, d)$, there are k servers sitting at some points in $\mathscr{M}$
    - A sequence σ of requests, each is a point in $\mathscr{M}$
    - Once a request arises at a point $p \in \mathscr{M}$, the algorithm has to send at least one server to p and serve the request
    - The goal is to minimize the total traveling distance of all servers



k-Server

- On a metric space $(\mathscr{M}, d)$, there are k servers sitting at some points in $\mathscr{M}$
  - A sequence σ of requests, each is a point in $\mathscr{M}$
  - Once a request arises at a point $p \in \mathscr{M}$, the algorithm has to send at least one server to p and serve the request
  - The goal is to minimize the total traveling distance of all servers

k-Server

- On a metric space $(\mathscr{M}, d)$, there are k servers sitting at some points in $\mathscr{M}$
  - A sequence σ of requests, each is a point in $\mathscr{M}$
  - Once a request arises at a point $p \in \mathscr{M}$, the algorithm has to send at least one server to p and serve the request
  - The goal is to minimize the total traveling distance of all servers

k-Server

- On a metric space $(\mathscr{M}, d)$, there are k servers sitting at some points in $\mathscr{M}$
  - A sequence σ of requests, each is a point in $\mathscr{M}$
  - Once a request arises at a point $p \in \mathscr{M}$, the algorithm has to send at least one server to p and serve the request
  - The goal is to minimize the total traveling distance of all servers

k-Server

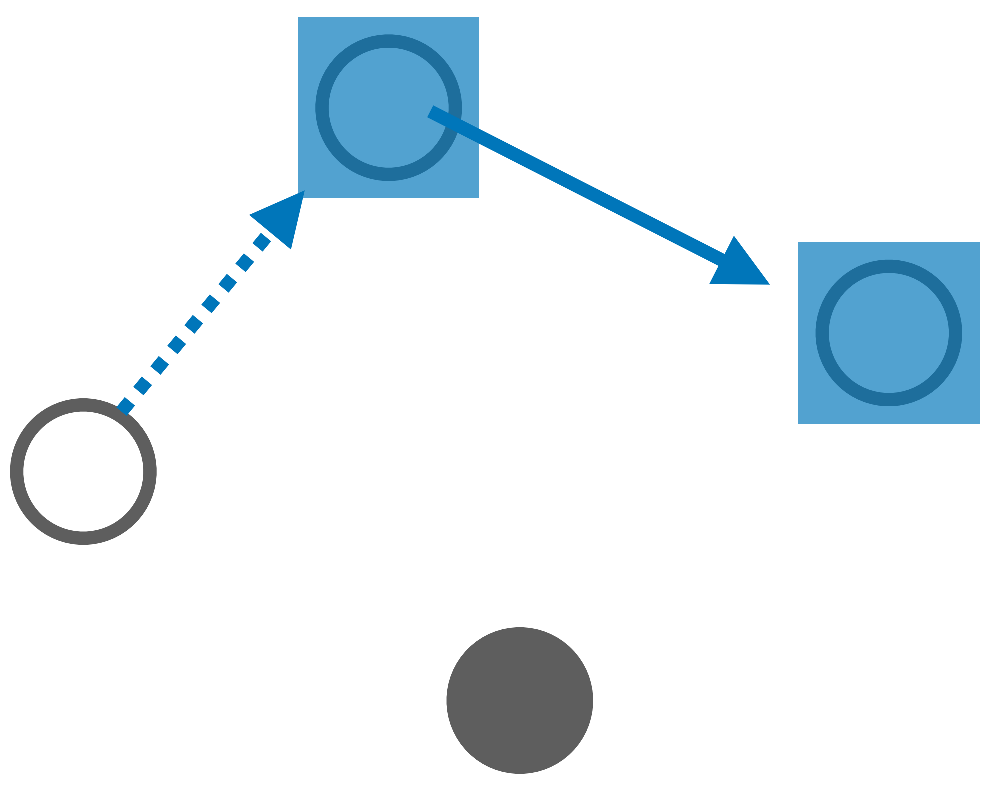
- On a metric space $(\mathscr{M}, d)$, there are k servers sitting at some points in $\mathscr{M}$
  - A sequence σ of requests, each is a point in $\mathscr{M}$
  - Once a request arises at a point $p \in \mathscr{M}$, the algorithm has to send at least one server to p and serve the request
  - The goal is to minimize the total traveling distance of all servers

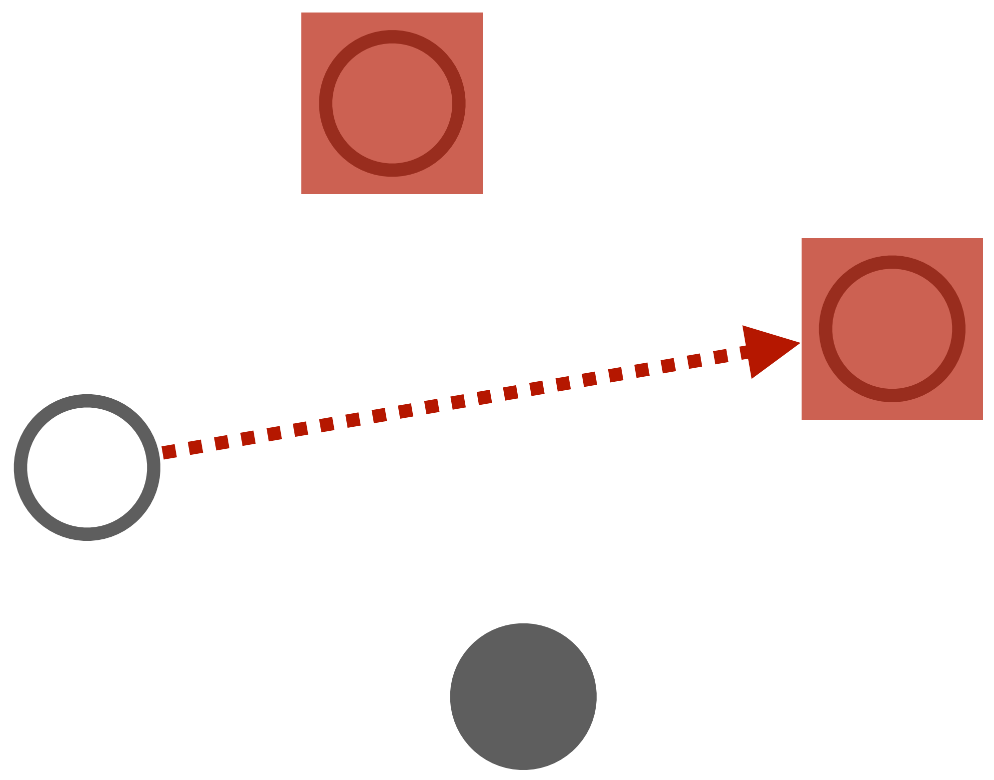
k-Server

- On a metric space $(\mathscr{M}, d)$, there are k servers sitting at some points in $\mathscr{M}$
  - A sequence σ of requests, each is a point in $\mathscr{M}$
  - Once a request arises at a point $p \in \mathscr{M}$, the algorithm has to send at least one server to p and serve the request
  - The goal is to minimize the total traveling distance of all servers

k-Server

- On a metric space $(\mathscr{M}, d)$, there are k servers sitting at some points in $\mathscr{M}$
  - A sequence σ of requests, each is a point in $\mathscr{M}$
  - Once a request arises at a point $p \in \mathscr{M}$, the algorithm has to send at least one server to p and serve the request
  - The goal is to minimize the total traveling distance of all servers





k-Server

- On a metric space $(\mathscr{M}, d)$, there are k servers sitting at some points in $\mathscr{M}$
  - A sequence σ of requests, each is a point in $\mathscr{M}$
  - Once a request arises at a point $p \in \mathscr{M}$, the algorithm has to send at least one server to p and serve the request
  - The goal is to minimize the total traveling distance of all servers

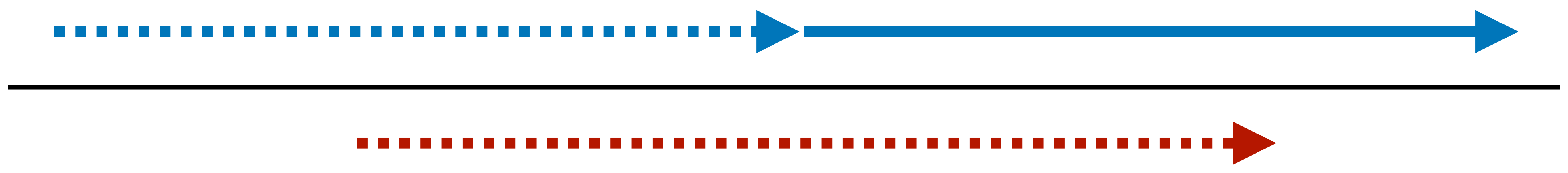
k-Server

- On a metric space $(\mathscr{M}, d)$, there are k servers sitting at some points in $\mathscr{M}$
  - A sequence σ of requests, each is a point in $\mathscr{M}$
  - Once a request arises at a point $p \in \mathscr{M}$, the algorithm has to send at least one server to p and serve the request
  - The goal is to minimize the total traveling distance of all servers

k-Server

- On a metric space $(\mathscr{M}, d)$, there are k servers sitting at some points in $\mathscr{M}$
  - A sequence σ of requests, each is a point in $\mathscr{M}$
  - Once a request arises at a point $p \in \mathscr{M}$, the algorithm has to send at least one server to p and serve the request
  - The goal is to minimize the total traveling distance of all servers

k-Server

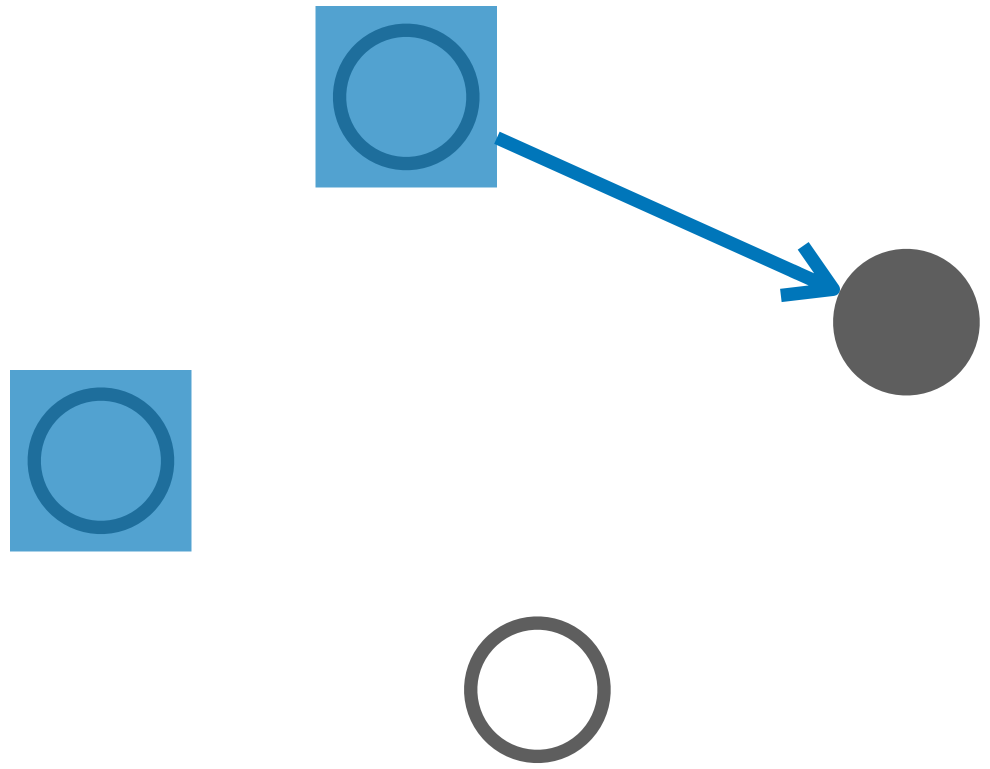
ratio = 

Greedy algorithm

`Always send the server that is the closest to the request`

Greedy algorithm

`Always send the server that is the closest to the request`



Greedy algorithm is unbounded

`Always send the server that is the closest to the request`

Greedy algorithm is unbounded

`Always send the server that is the closest to the request`

Greedy algorithm is unbounded

`Always send the server that is the closest to the request`

Greedy algorithm is unbounded

Always send the server that is the closest to the request

Greedy algorithm is unbounded

`Always send the server that is the closest to the request`

Greedy algorithm is unbounded

`Always send the server that is the closest to the request`

Greedy algorithm is unbounded

`Always send the server that is the closest to the request`

Greedy algorithm is unbounded

`Always send the server that is the closest to the request`

Greedy algorithm is unbounded

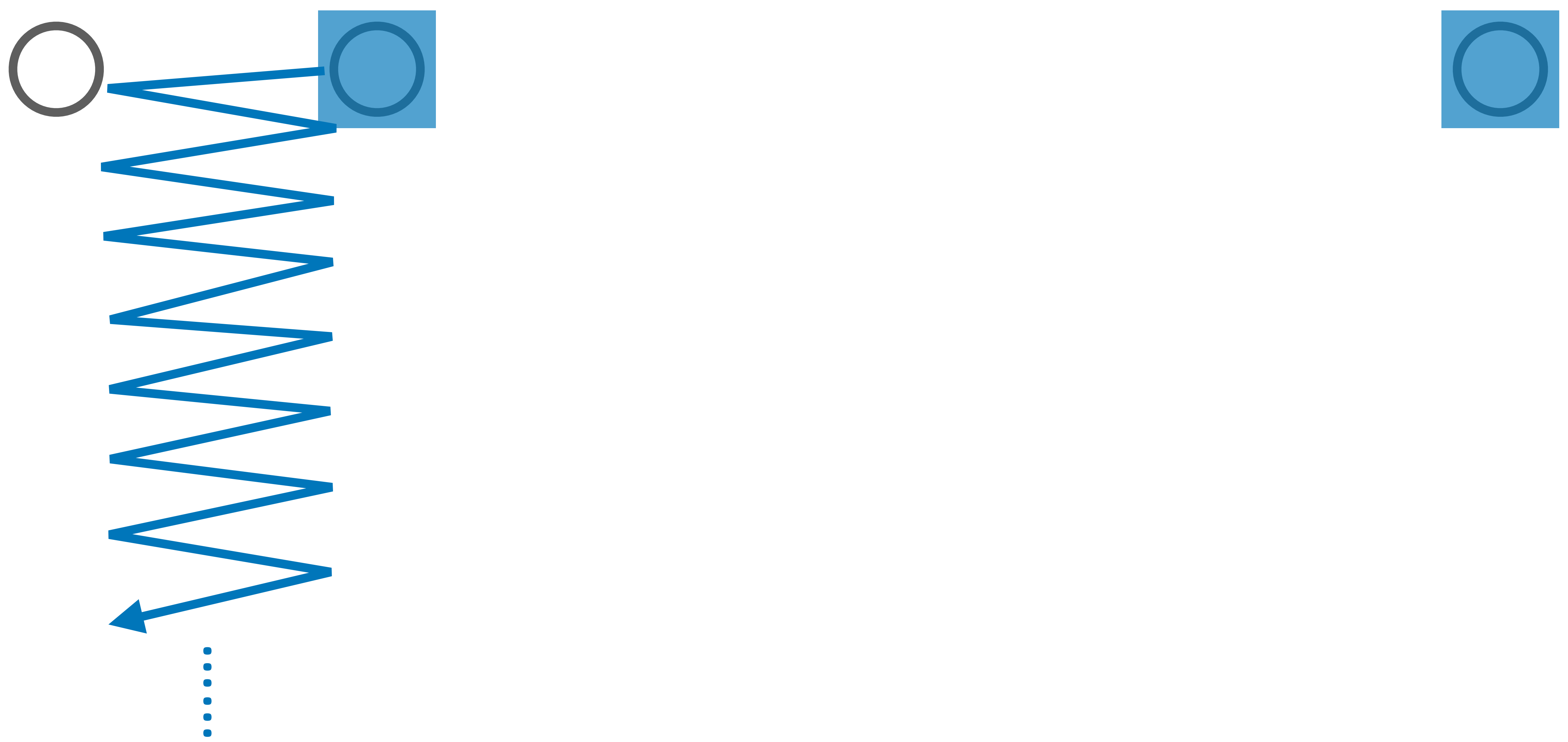
`Always send the server that is the closest to the request`

Greedy algorithm is unbounded

Always send the server that is the closest to the request

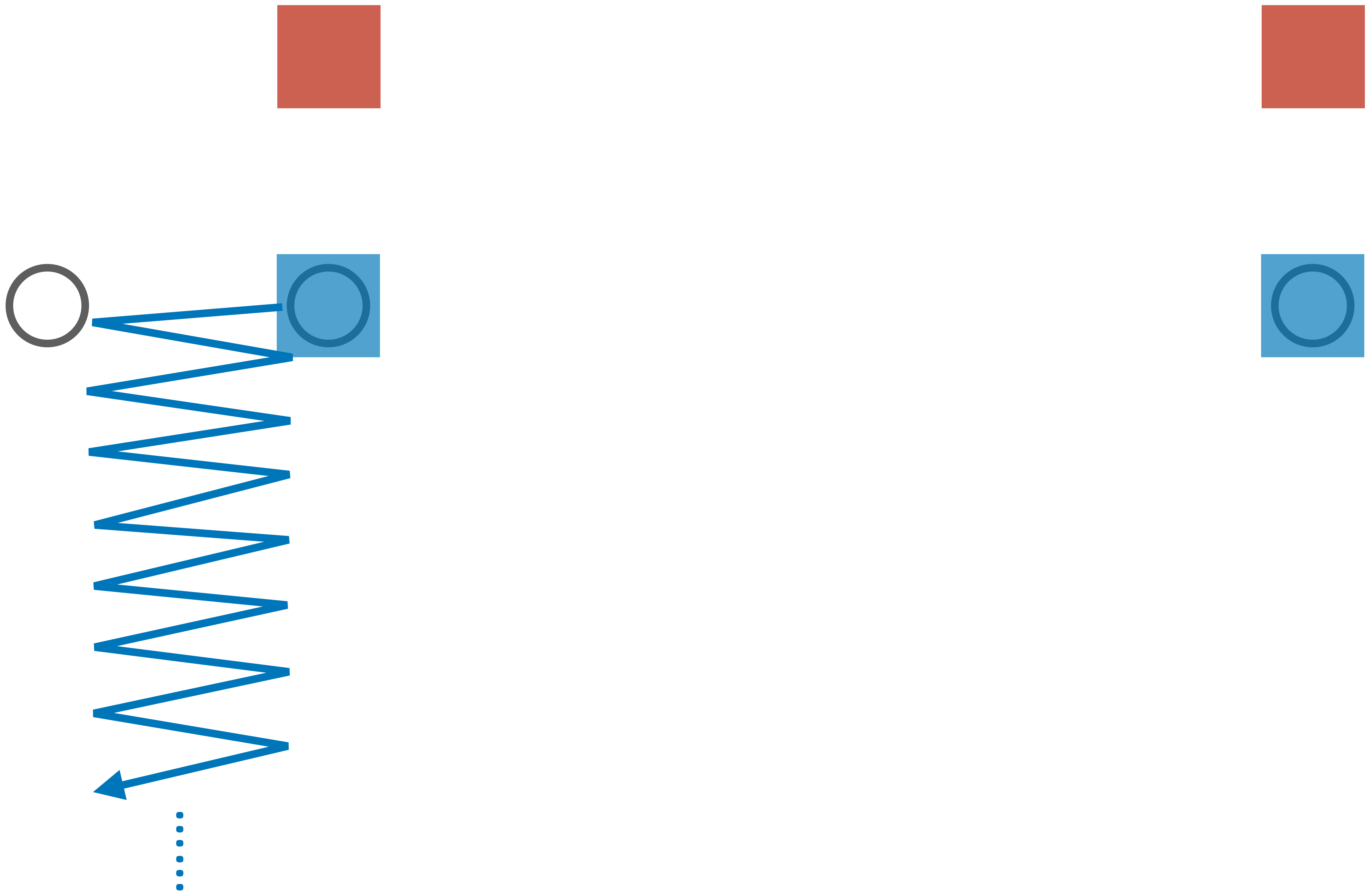
Greedy algorithm is unbounded

`Always send the server that is the closest to the request`



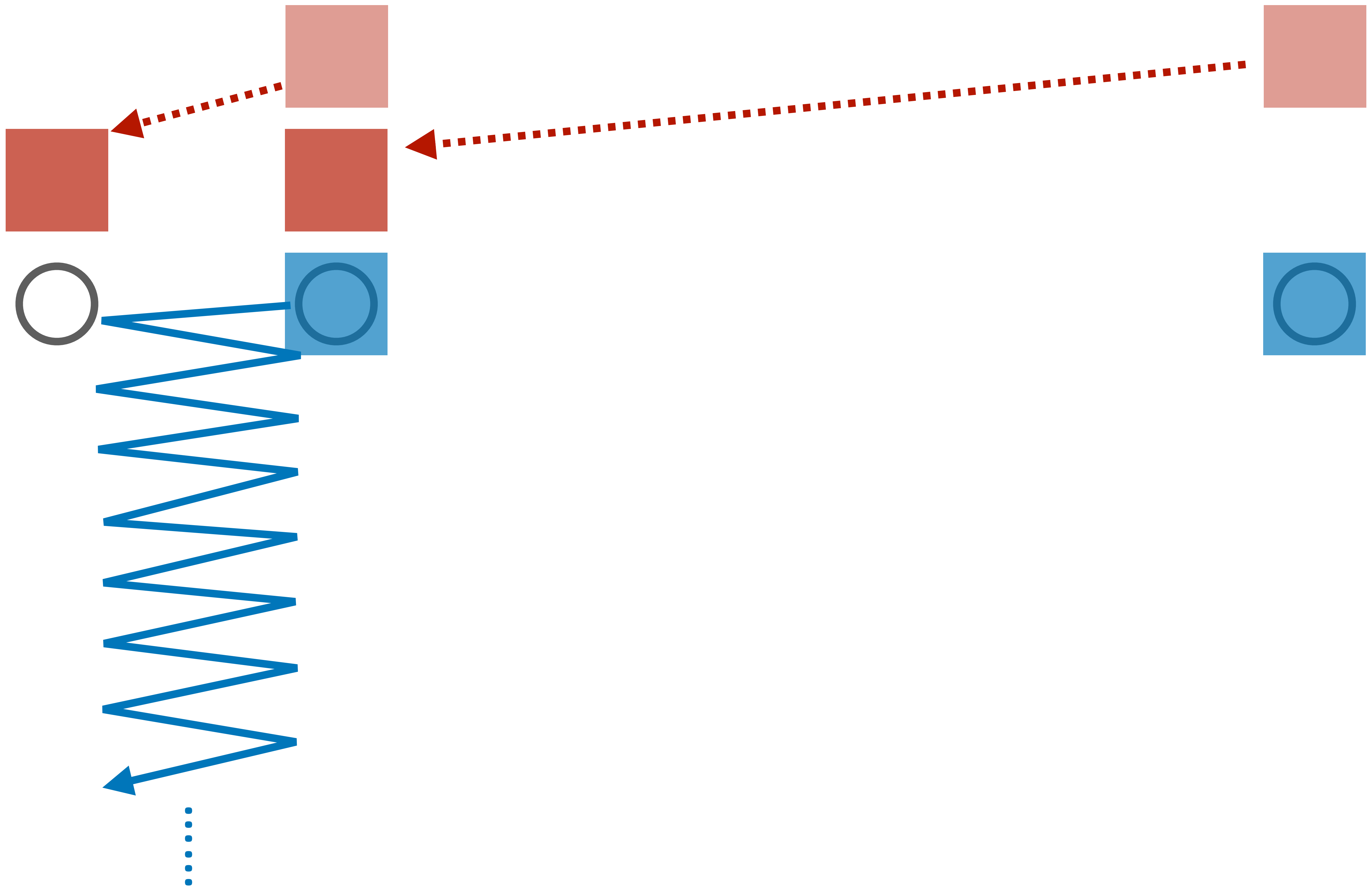
Greedy algorithm is unbounded

```
Always send the server that is the closest to the request
```



Greedy algorithm is unbounded

`Always send the server that is the closest to the request`



Double-Coverage on a line

```
If the request falls outside the convex hull of the
servers, serve it with the nearest server

Otherwise, move the two closest servers towards the request
at equal speeds until at least one server reaches it
```

Double-Coverage on a line

If the request falls outside the convex hull of the servers, serve it with the nearest server

Otherwise, move the two closest servers towards the request at equal speeds until at least one server reaches it

Double-Coverage on a line

If the request falls outside the convex hull of the servers, serve it with the nearest server

Otherwise, move the two closest servers towards the request at equal speeds until at least one server reaches it

Double-Coverage on a line

If the request falls outside the convex hull of the servers, serve it with the nearest server

Otherwise, move the two closest servers towards the request at equal speeds until at least one server reaches it

Double-Coverage on a line

If the request falls outside the convex hull of the servers, serve it with the nearest server

Otherwise, move the two closest servers towards the request at equal speeds until at least one server reaches it

Double-Coverage on a line

If the request falls outside the convex hull of the servers, serve it with the nearest server

Otherwise, move the two closest servers towards the request at equal speeds until at least one server reaches it

Double-Coverage on a line

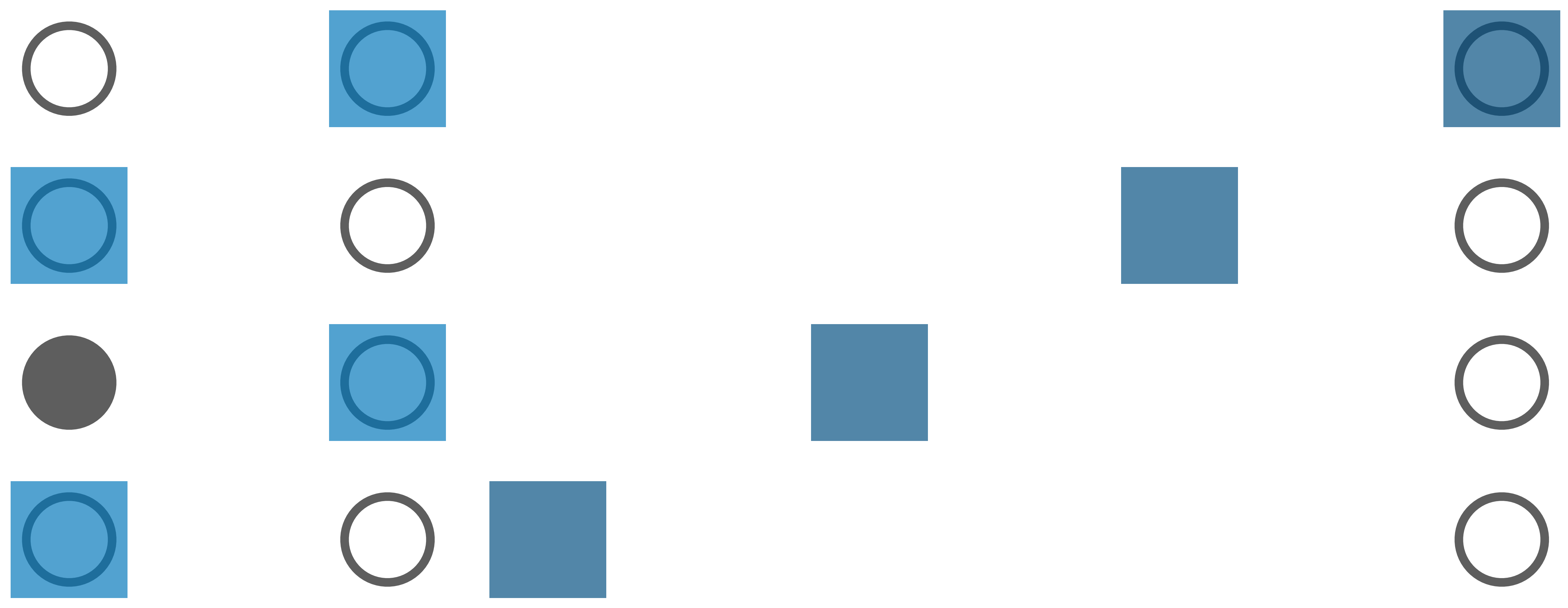
If the request falls outside the convex hull of the servers, serve it with the nearest server

Otherwise, move the two closest servers towards the request at equal speeds until at least one server reaches it

Double-Coverage on a line

If the request falls outside the convex hull of the servers, serve it with the nearest server

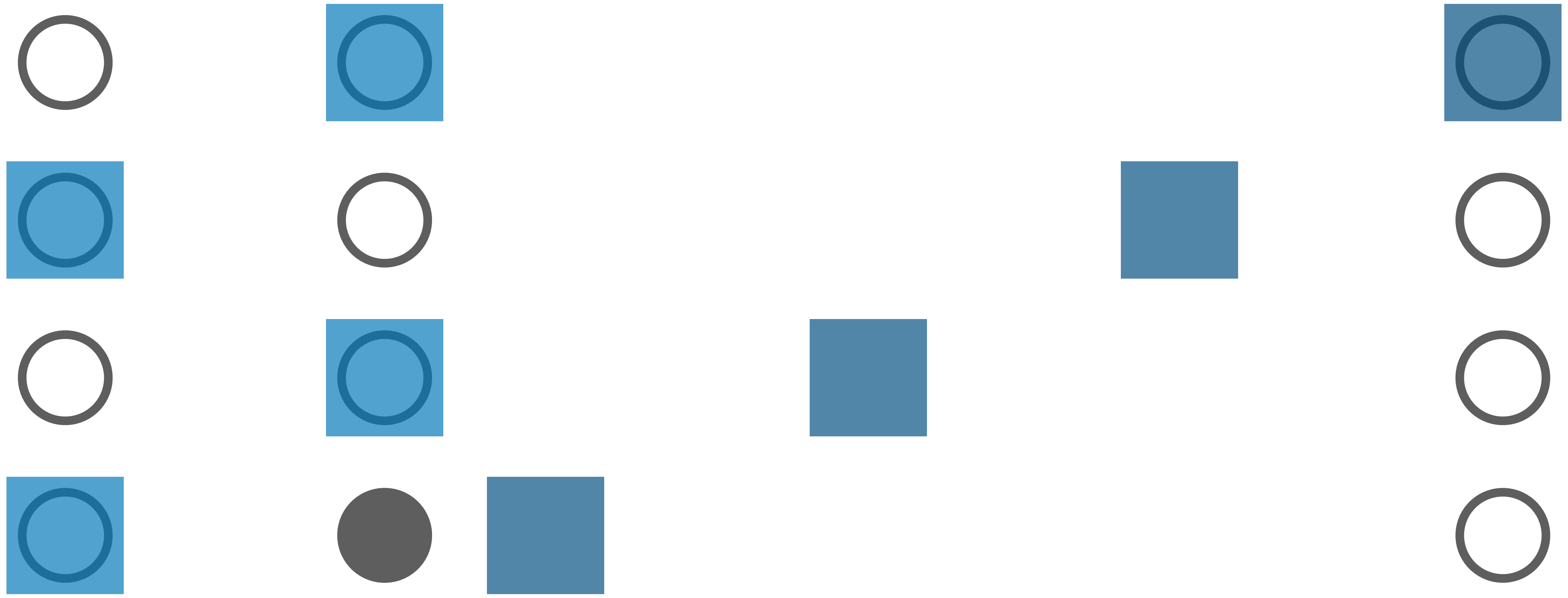
Otherwise, move the two closest servers towards the request at equal speeds until at least one server reaches it



Double-Coverage on a line

If the request falls outside the convex hull of the servers, serve it with the nearest server

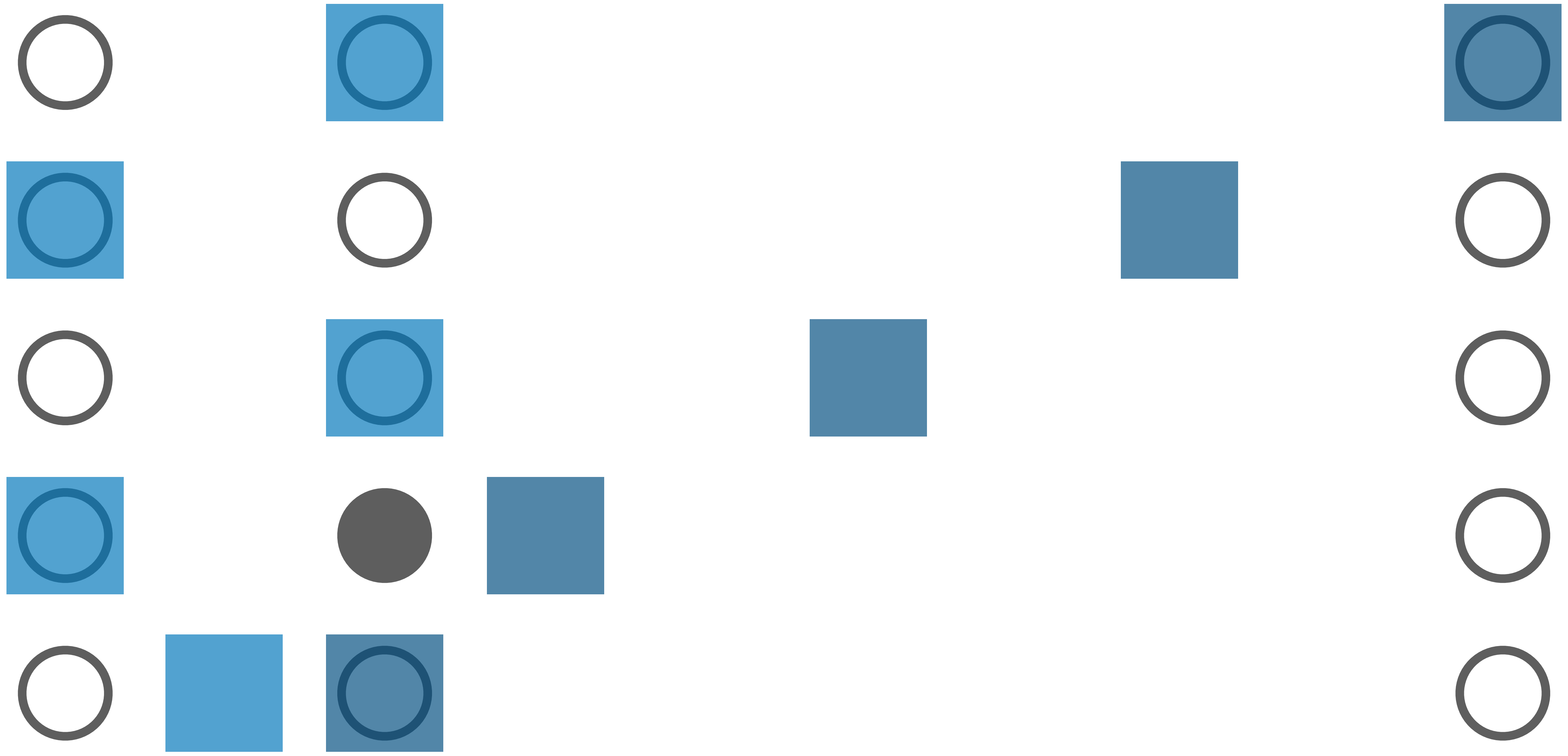
Otherwise, move the two closest servers towards the request at equal speeds until at least one server reaches it



Double-Coverage on a line

If the request falls outside the convex hull of the servers, serve it with the nearest server

Otherwise, move the two closest servers towards the request at equal speeds until at least one server reaches it



DC is k-competitive on a line

If the request falls outside the convex hull of the servers, serve it with the nearest server

Otherwise, move the two closest servers towards the request at equal speeds until at least one server reaches it

DC is k-competitive on a line

```
If the request falls outside the convex hull of the
servers, serve it with the nearest server

Otherwise, move the two closest servers towards the request
at equal speeds until at least one server reaches it
```

<Proof Idea>

1. Set a potential function $\Phi = k \cdot M_{\min} + \Sigma_{DC}$
   - $M_{\min}$: cost of the minimum matching between DC servers to OPT servers
   - Σ_{DC}: sum of pairwise distance between DC servers
2. Assume that once a request arrives, OPT moves first, and then DC moves. Show that:

   (1) When OPT moves d, $\Delta\Phi_i \leq k \cdot d$

   (2) When DC moves d, $\Delta\Phi_i \leq -d$

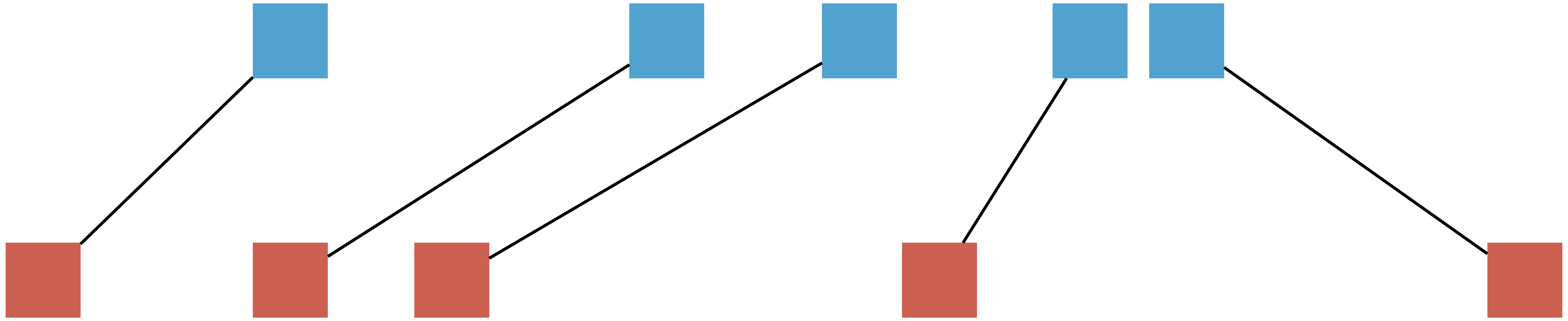
DC is k-competitive on a line

```
If the request falls outside the convex hull of the
servers, serve it with the nearest server

Otherwise, move the two closest servers towards the request
at equal speeds until at least one server reaches it
```

<Proof Idea>

1. Set a potential function $\Phi = k \cdot M_{\min} + \Sigma_{DC}$
   - $M_{\min}$: cost of the minimum matching between DC servers to OPT servers
   - Σ_{DC}: sum of pairwise distance between DC servers



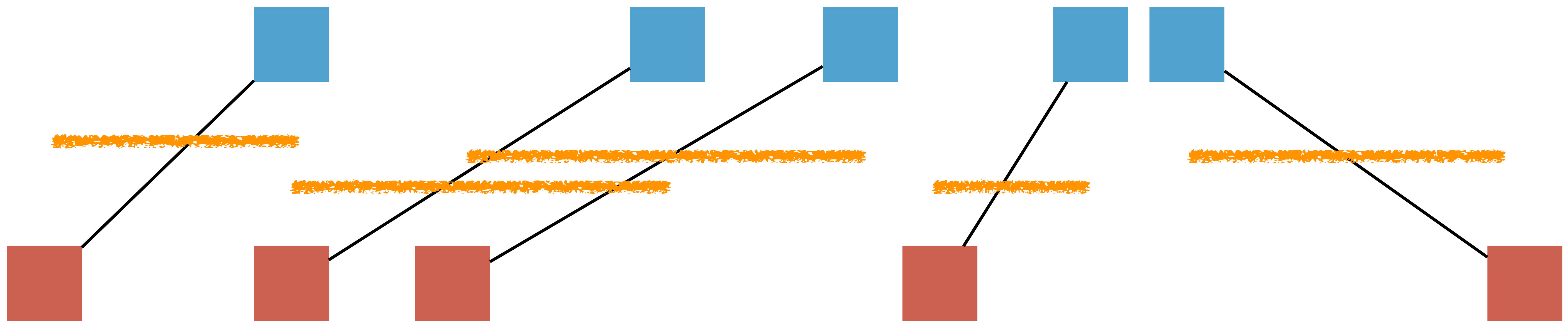
DC is k-competitive on a line

```
If the request falls outside the convex hull of the
servers, serve it with the nearest server

Otherwise, move the two closest servers towards the request
at equal speeds until at least one server reaches it
```

<Proof Idea>

1. Set a potential function $\Phi = k \cdot M_{\min} + \Sigma_{DC}$
   - $M_{\min}$: cost of the minimum matching between DC servers to OPT servers
   - Σ_{DC}: sum of pairwise distance between DC servers



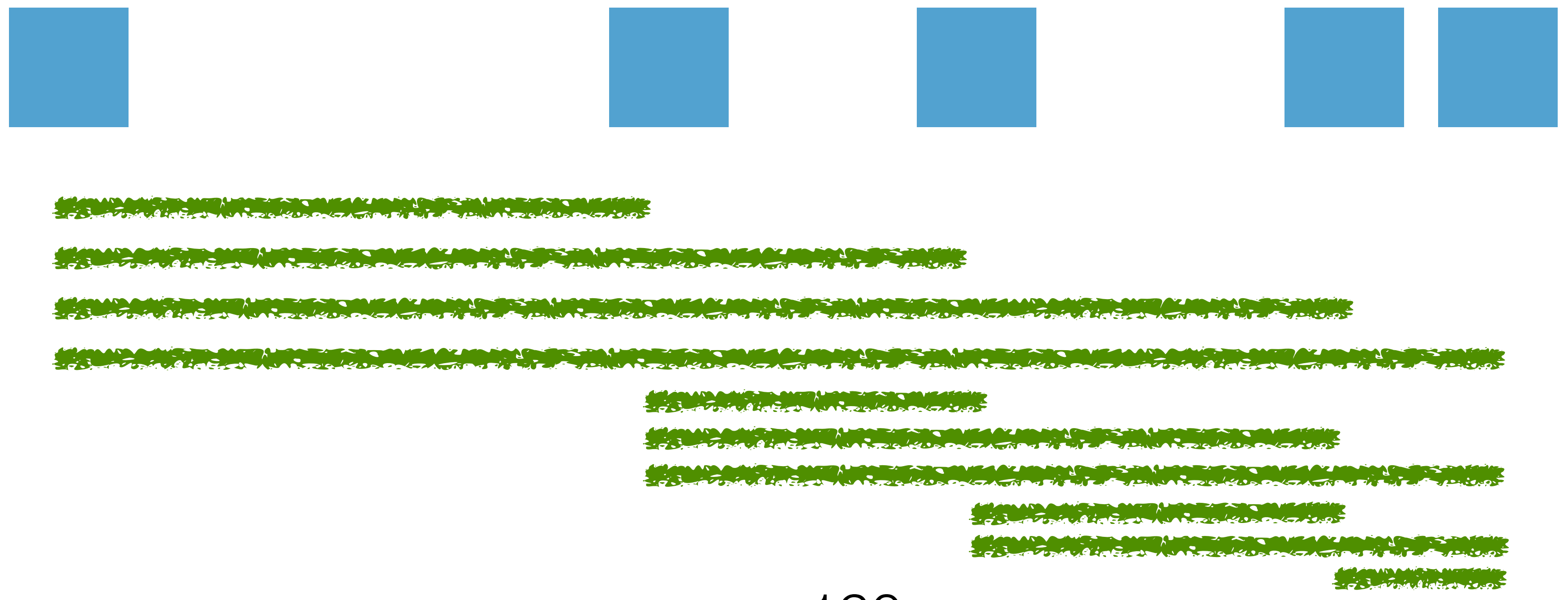
DC is k-competitive on a line

```
If the request falls outside the convex hull of the
servers, serve it with the nearest server

Otherwise, move the two closest servers towards the request
at equal speeds until at least one server reaches it
```

<Proof Idea>

1. Set a potential function $\Phi = k \cdot M_{\min} + \Sigma_{DC}$
    - $M_{\min}$: cost of the minimum matching between DC servers to OPT servers
    - Σ_{DC}: sum of pairwise distance between DC servers



DC is k-competitive on a line

If the request falls outside the convex hull of the servers, serve it with the nearest server

Otherwise, move the two closest servers towards the request at equal speeds until at least one server reaches it

<Proof Idea>

1. Set a potential function $\Phi = k \cdot M_{\min} + \Sigma_{DC}$
   - $M_{\min}$: cost of the minimum matching between DC servers to OPT servers
   - Σ_{DC}: sum of pairwise distance between DC servers
2. Assume that once a request arrives, OPT moves first, and then DC moves. Show that:

   (1) When OPT moves d, $\Delta\Phi_i \leq k \cdot d$ — $\text{DC}_i + \Phi_i \leq 0 + k \cdot d = k \cdot \text{OPT}_i$

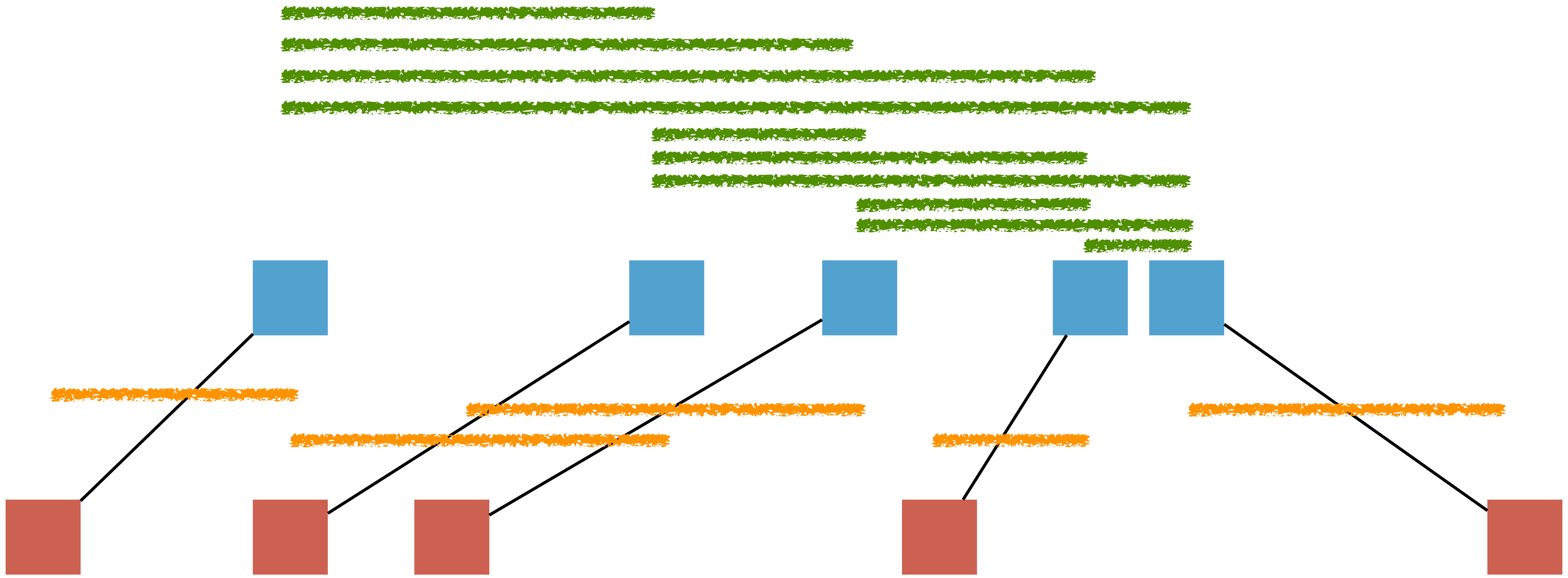
   (2) When DC moves d, $\Delta\Phi_i \leq -d$ — $\text{DC}_i + \Phi_i \leq d - d = 0 = k \cdot \text{OPT}_i$

DC is k-competitive on a line

If the request falls outside the convex hull of the servers, serve it with the nearest server

Otherwise, move the two closest servers towards the request at equal speeds until at least one server reaches it

(1) When OPT moves d, $\Delta\Phi_i \leq k \cdot d$



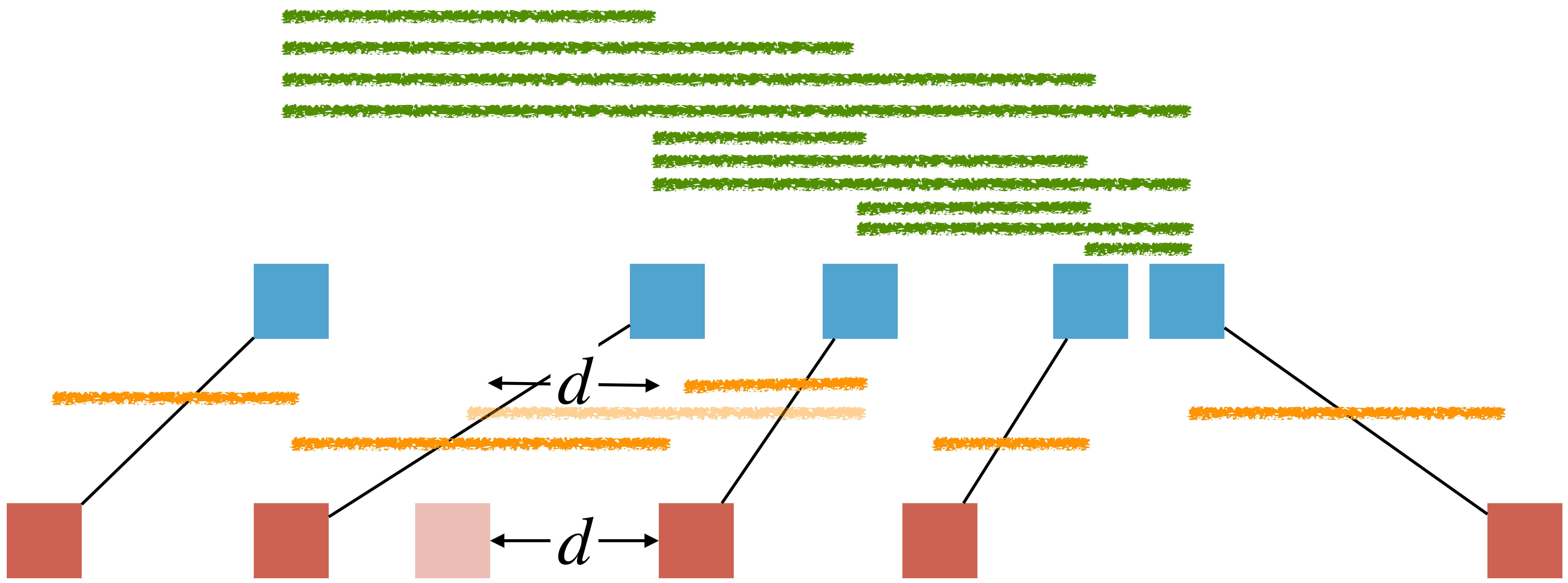
DC is k-competitive on a line

```
If the request falls outside the convex hull of the
servers, serve it with the nearest server

Otherwise, move the two closest servers towards the request
at equal speeds until at least one server reaches it
```

(1) When OPT moves d, $\Delta\Phi_i \leq k \cdot d$

$$\Phi = k \cdot M_{\min} + \Sigma_{DC}$$

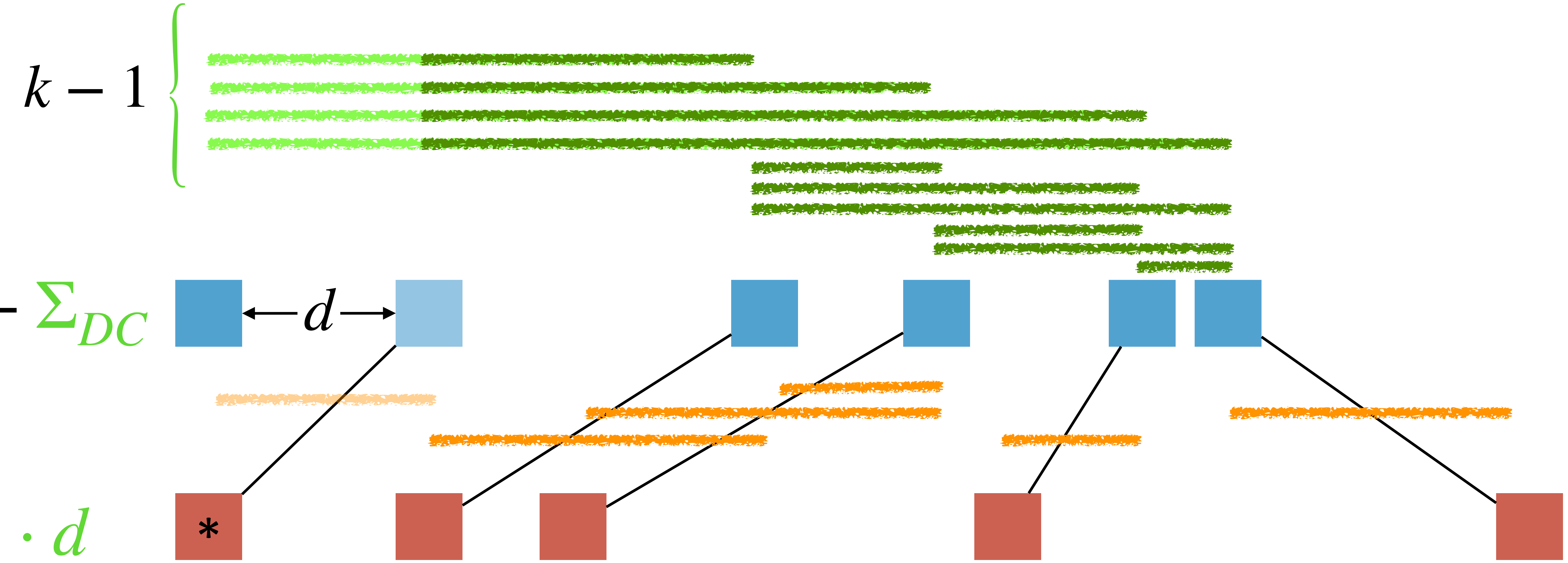


DC is k-competitive on a line

If the request falls outside the convex hull of the servers, serve it with the nearest server

Otherwise, move the two closest servers towards the request at equal speeds until at least one server reaches it

(2) When DC moves d, $\Delta\Phi_i \leq -d$



$$\Phi = k \cdot M_{\min} + \Sigma_{DC}$$
$$= k \cdot (-d)$$
$$+ (k-1) \cdot d$$

DC is k-competitive on a line

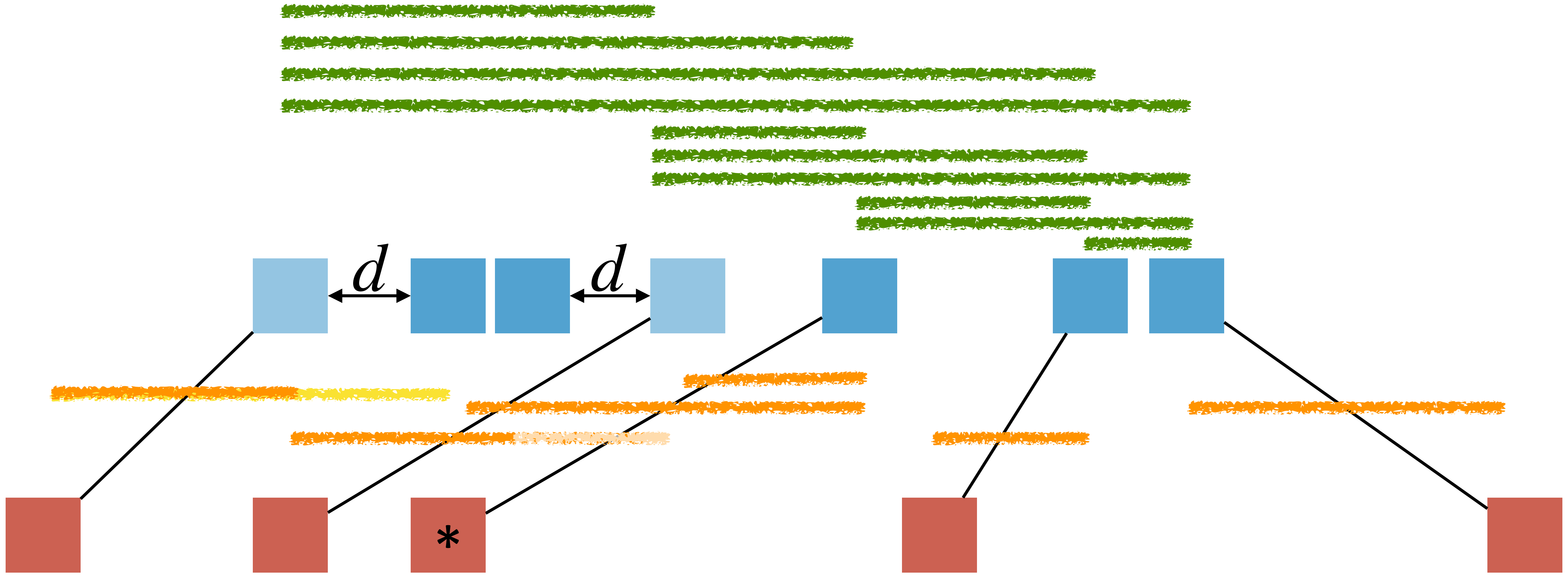
If the request falls outside the convex hull of the servers, serve it with the nearest server

Otherwise, move the two closest servers towards the request at equal speeds until at least one server reaches it

(2) When DC moves $2d$, $\Delta\Phi_i \leq -2d$

$$\Phi = k \cdot M_{\min} + \Sigma_{DC}$$

$$= k \cdot 0$$



DC is k-competitive on a line

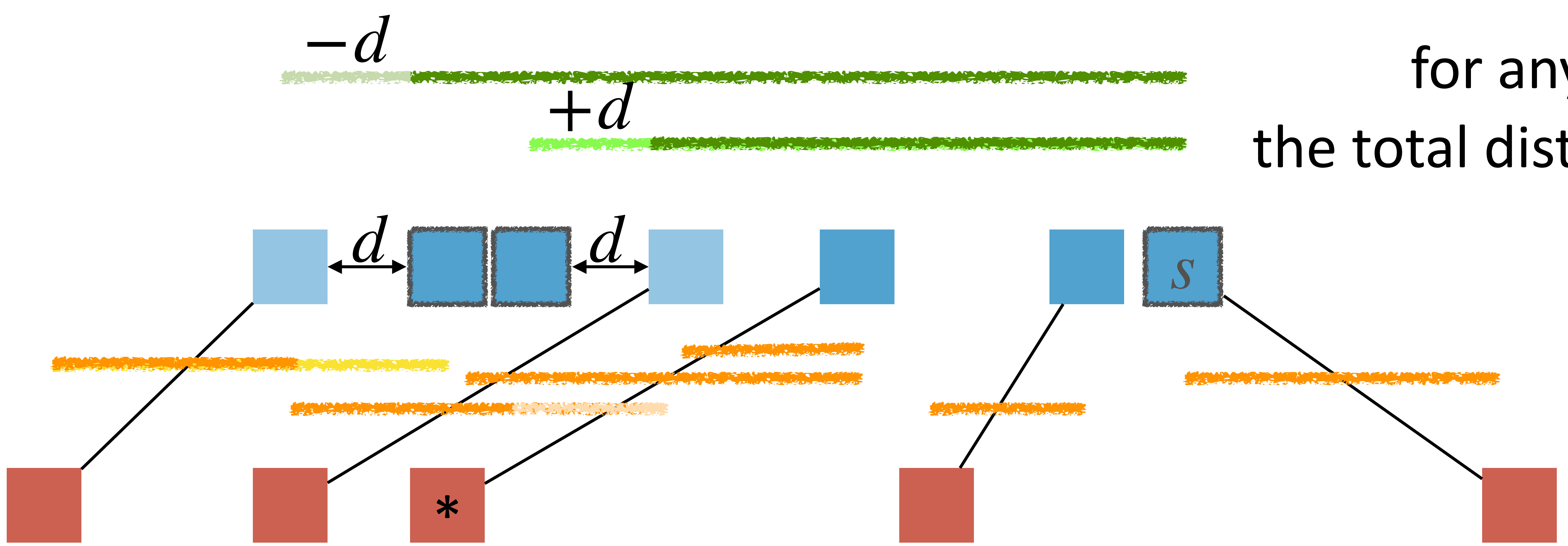
```
If the request falls outside the convex hull of the
servers, serve it with the nearest server

Otherwise, move the two closest servers towards the request
at equal speeds until at least one server reaches it
```

(2) When DC moves $2d$, $\Delta\Phi_i \leq -2d$

$$\Phi = k \cdot M_{\min} + \Sigma_{DC}$$

$$= k \cdot 0$$



for any other server s, the total distance does not change

DC is k-competitive on a line

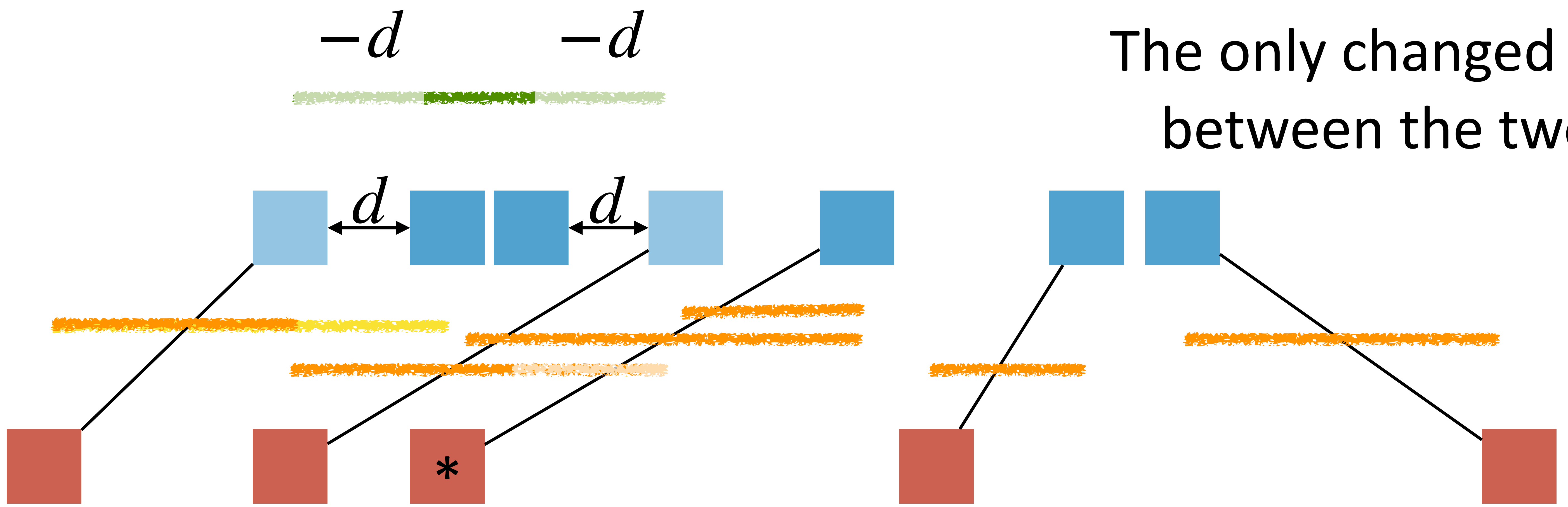
```
If the request falls outside the convex hull of the
servers, serve it with the nearest server

Otherwise, move the two closest servers towards the request
at equal speeds until at least one server reaches it
```

(2) When DC moves $2d$, $\Delta\Phi_i \le -2d$

$$\Phi = k \cdot M_{\min} + \Sigma_{DC}$$

$$= k \cdot 0 + 2d$$



The only changed distance is the one between the two moving servers

k-Server Lower Bound