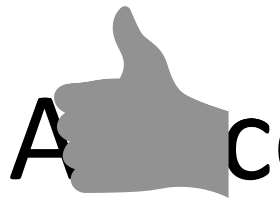
Algorithms for Decision Support

Online Algorithms (2/3)

Online Bidding

Outline

- Recap from the last lecture
- **Online bidding**
    - A  competitive algorithm
- A general technique for designing online algorithms
- “Best” online algorithms

Outline

- Recap from the last lecture
- **Online bidding**
  - A  competitive algorithm
- A general technique for designing online algorithms

- "Best" online algorithms

Outline

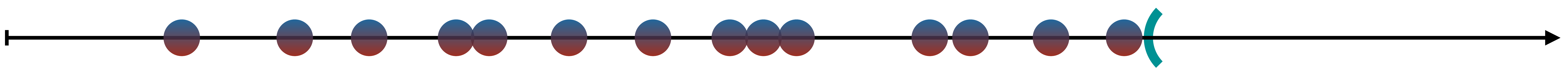
- Recap from the last lecture
- **Online bidding**
  - A competitive algorithm
- A general technique for designing online algorithms
- "Best" online algorithms

Recap

- Online problems, instances, and online algorithms

- Measurement of online algorithms: **competitive ratio**

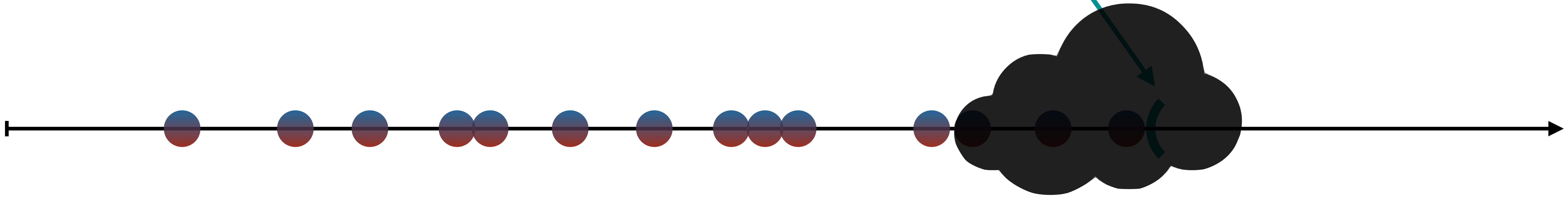
No "ball" (which is $\frac{\text{ALG}(I)}{\text{OPT}(I)}$ for some instance I) is on the right of it



Recap

- Online problems, instances, and online algorithms

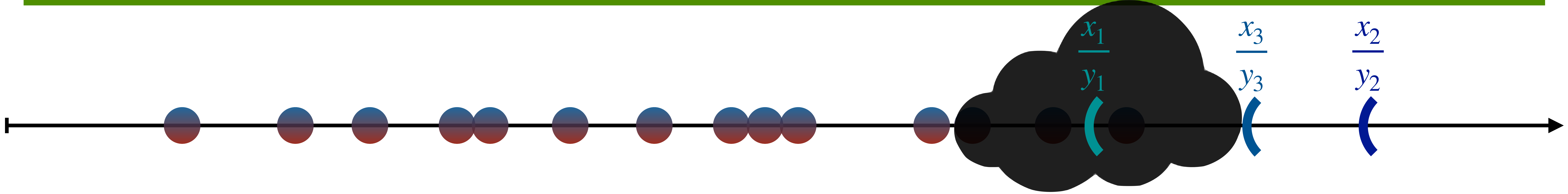
- Measurement of online algorithms: **competitive ratio**



Recap

Method 1: To analyze the competitive ratio of an algorithm, one can argue that: For any instance I, ALG$(I) \leq x$ and OPT$(I) \geq y$. Therefore,

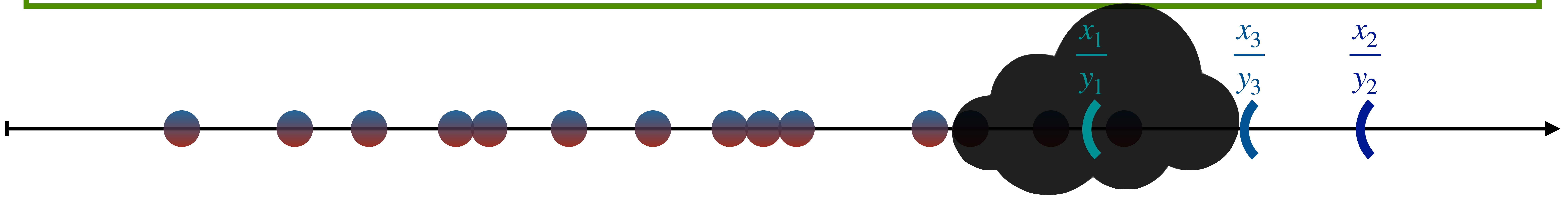
$$\frac{\text{ALG}(I)}{\text{OPT}(I)} \leq \frac{x}{y}$$



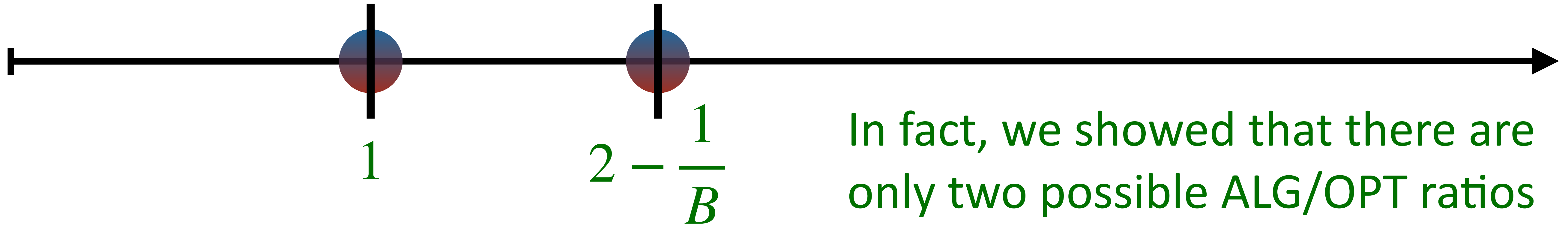
Recap

Method 1: To analyze the competitive ratio of an algorithm, one can argue that: For any instance I, ALG$(I) \leq x$ and OPT$(I) \geq y$. Therefore,

$$\frac{\text{ALG}(I)}{\text{OPT}(I)} \leq \frac{x}{y}$$



For the Ski-Rental problem, we showed that the "Buy on the B-th day" strategy is $(2 - \frac{1}{B})$-competitive

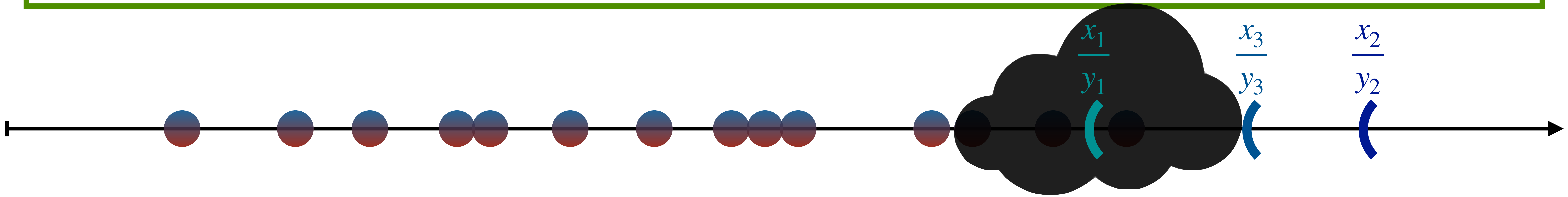


In fact, we showed that there are only two possible ALG/OPT ratios

This Lecture

Method 1: To analyze the competitive ratio of an algorithm, one can argue that: For any instance I, ALG$(I) \leq x$ and OPT$(I) \geq y$. Therefore,

$$\frac{\text{ALG}(I)}{\text{OPT}(I)} \leq \frac{x}{y}$$

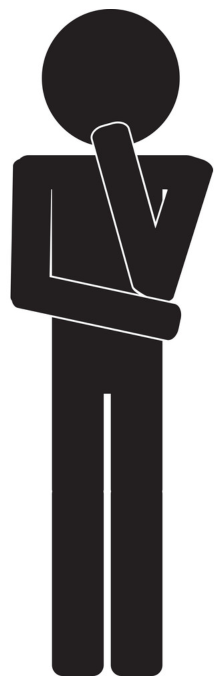


This lecture: how to (upper-) bound the competitive ratio in a more uncertain case?

- **Method 2: Assume we *know* the optimal solution cost, the algorithm cannot cost too much (hopefully…)**

Online Bidding

- The online algorithm (player)



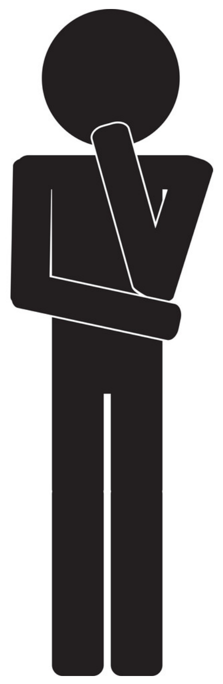
player

Online Bidding

- The online algorithm (player) wants to guess the target number $t \geq 1$, which is unknown to it.

Online Bidding

- The online algorithm (player) wants to guess the target number $t \geq 1$, which is unknown to it. To this end, the player submits a sequence $B = \{b_1, b_2, b_3, \cdots\}$ of *bids*, until one of them with value higher than or equal to t.



player

seller

Online Bidding

- The online algorithm (player) wants to guess the target number $t \geq 1$, which is unknown to it. To this end, the player submits a sequence $B = \{b_1, b_2, b_3, \cdots\}$ of *bids*, until one of them with value higher than or equal to t.

Online Bidding

- The online algorithm (player) wants to guess the target number $t \geq 1$, which is unknown to it. To this end, the player submits a sequence $B = \{b_1, b_2, b_3, \cdots\}$ of *bids*, until one of them with value higher than or equal to t.

Online Bidding

- The online algorithm (player) wants to guess the target number $t \geq 1$, which is unknown to it. To this end, the player submits a sequence $B = \{b_1, b_2, b_3, \cdots\}$ of *bids*, until one of them with value higher than or equal to t.

Online Bidding

- The online algorithm (player) wants to guess the target number $t \geq 1$, which is unknown to it. To this end, the player submits a sequence $B = \{b_1, b_2, b_3, \cdots\}$ of *bids*, until one of them with value higher than or equal to t.

Online Bidding

- The online algorithm (player) wants to guess the target number $t \geq 1$, which is unknown to it. To this end, the player submits a sequence $B = \{b_1, b_2, b_3, \cdots\}$ of *bids*, until one of them with value higher than or equal to t.

Online Bidding

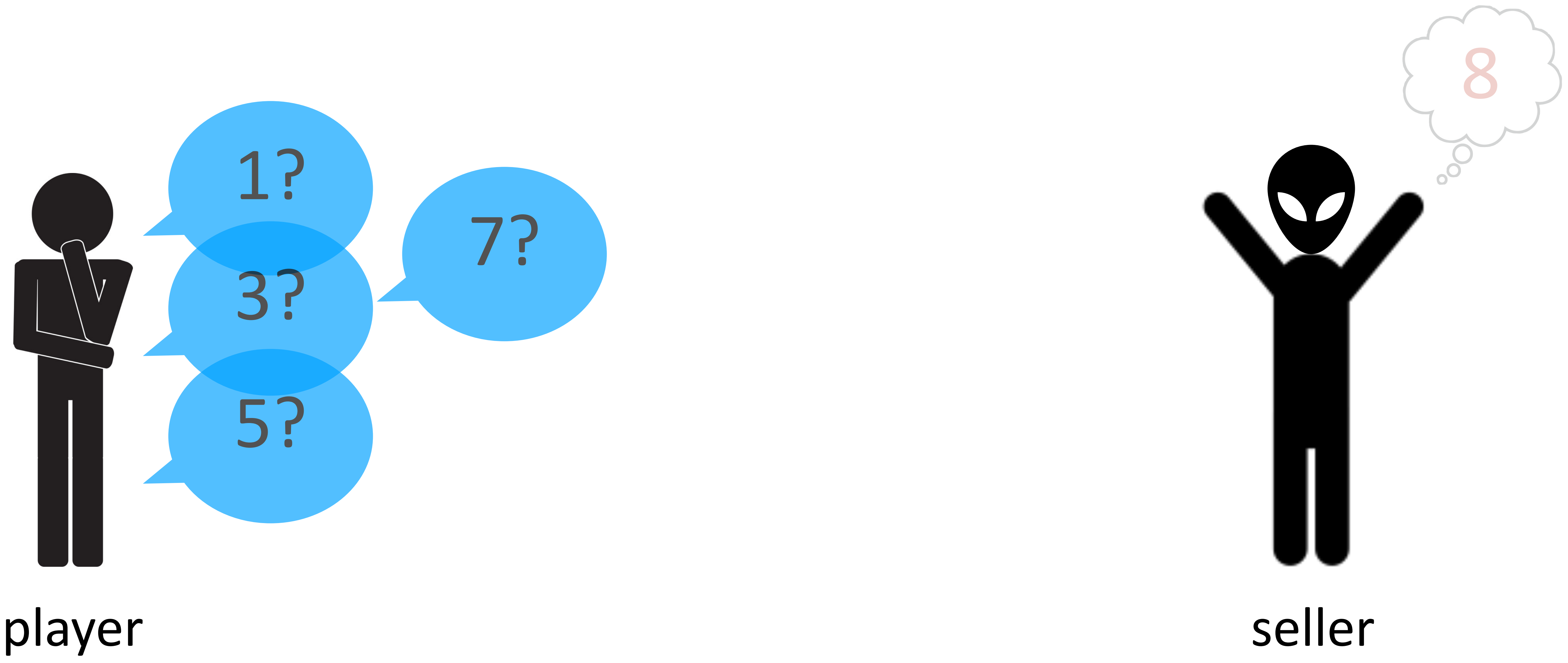
- The online algorithm (player) wants to guess the target number $t \geq 1$, which is unknown to it. To this end, the player submits a sequence $B = \{b_1, b_2, b_3, \cdots\}$ of *bids*, until one of them with value higher than or equal to t.

Online Bidding

- The online algorithm (player) wants to guess the target number $t \geq 1$, which is unknown to it. To this end, the player submits a sequence $B = \{b_1, b_2, b_3, \cdots\}$ of *bids*, until one of them with value higher than or equal to t.

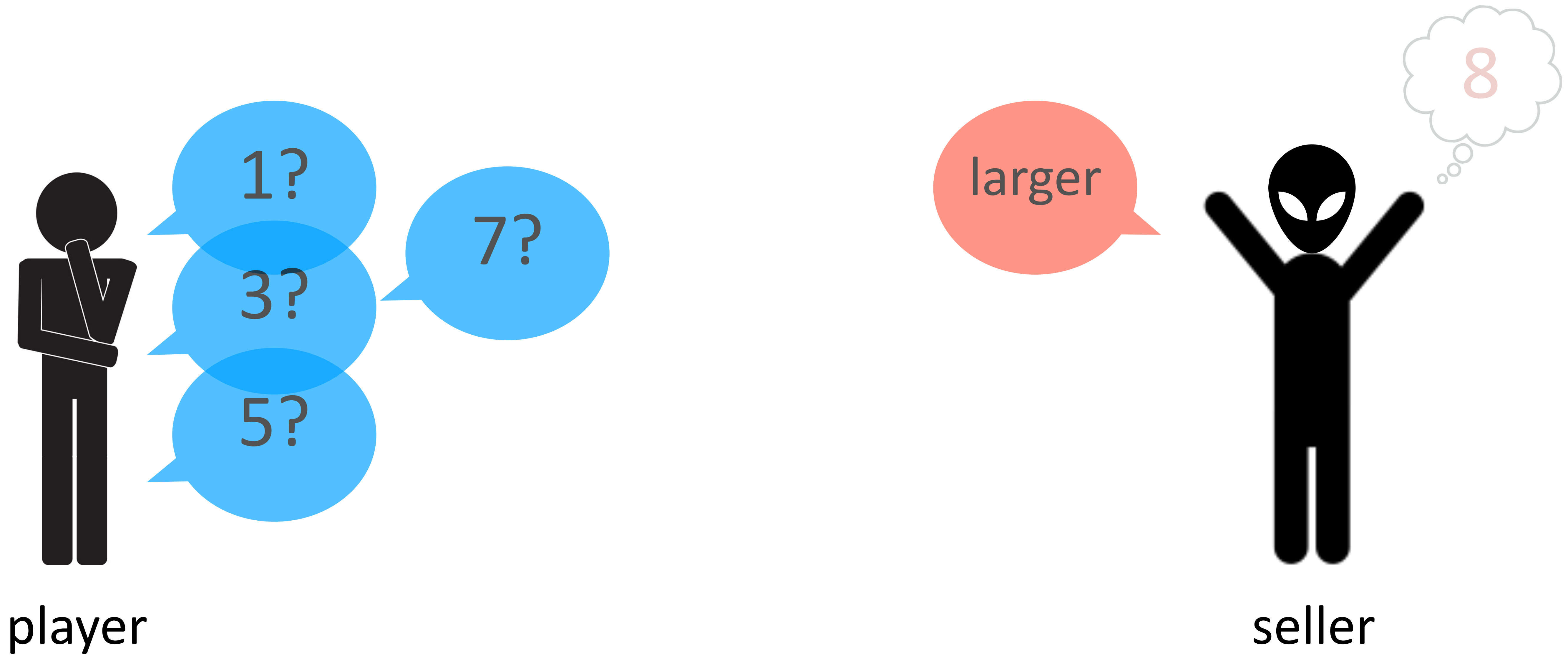
Online Bidding

- The online algorithm (player) wants to guess the target number $t \geq 1$, which is unknown to it. To this end, the player submits a sequence $B = \{b_1, b_2, b_3, \cdots\}$ of *bids*, until one of them with value higher than or equal to t.



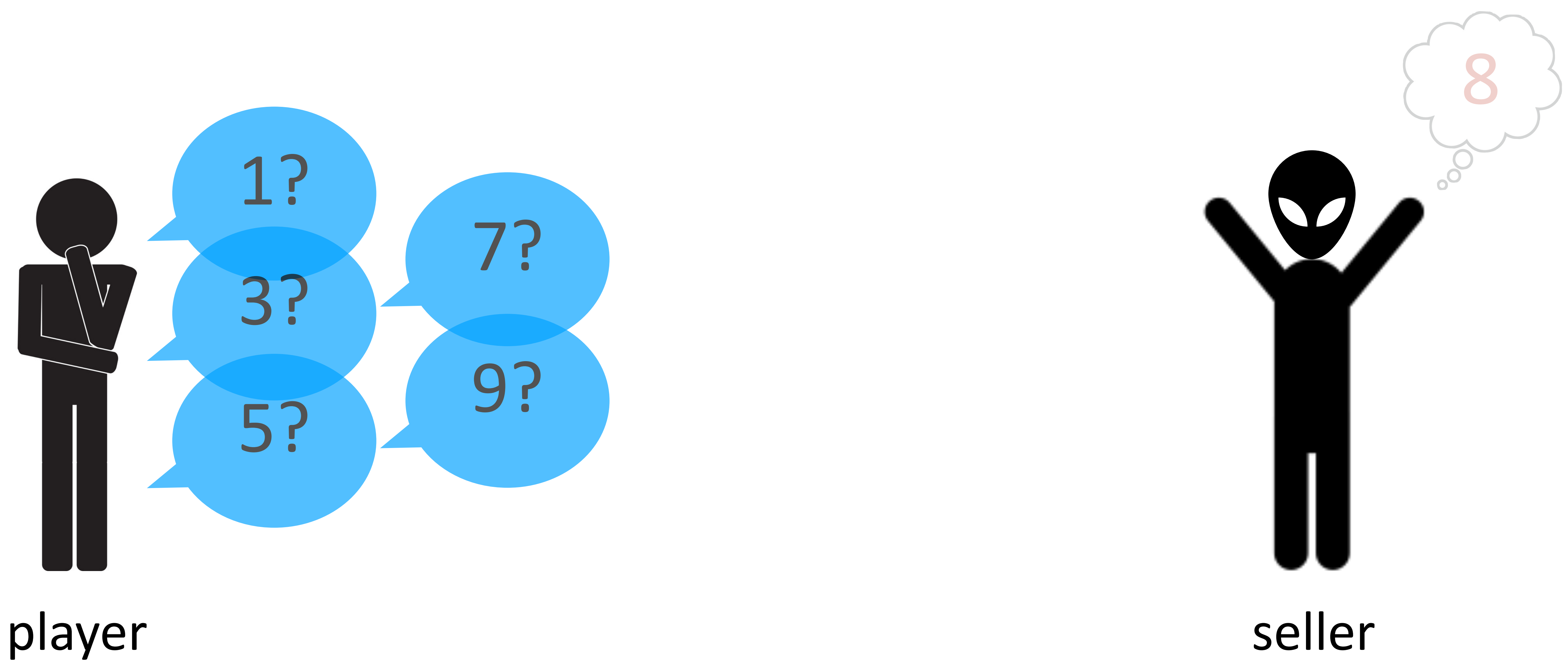
Online Bidding

- The online algorithm (player) wants to guess the target number $t \geq 1$, which is unknown to it. To this end, the player submits a sequence $B = \{b_1, b_2, b_3, \cdots\}$ of *bids*, until one of them with value higher than or equal to t.



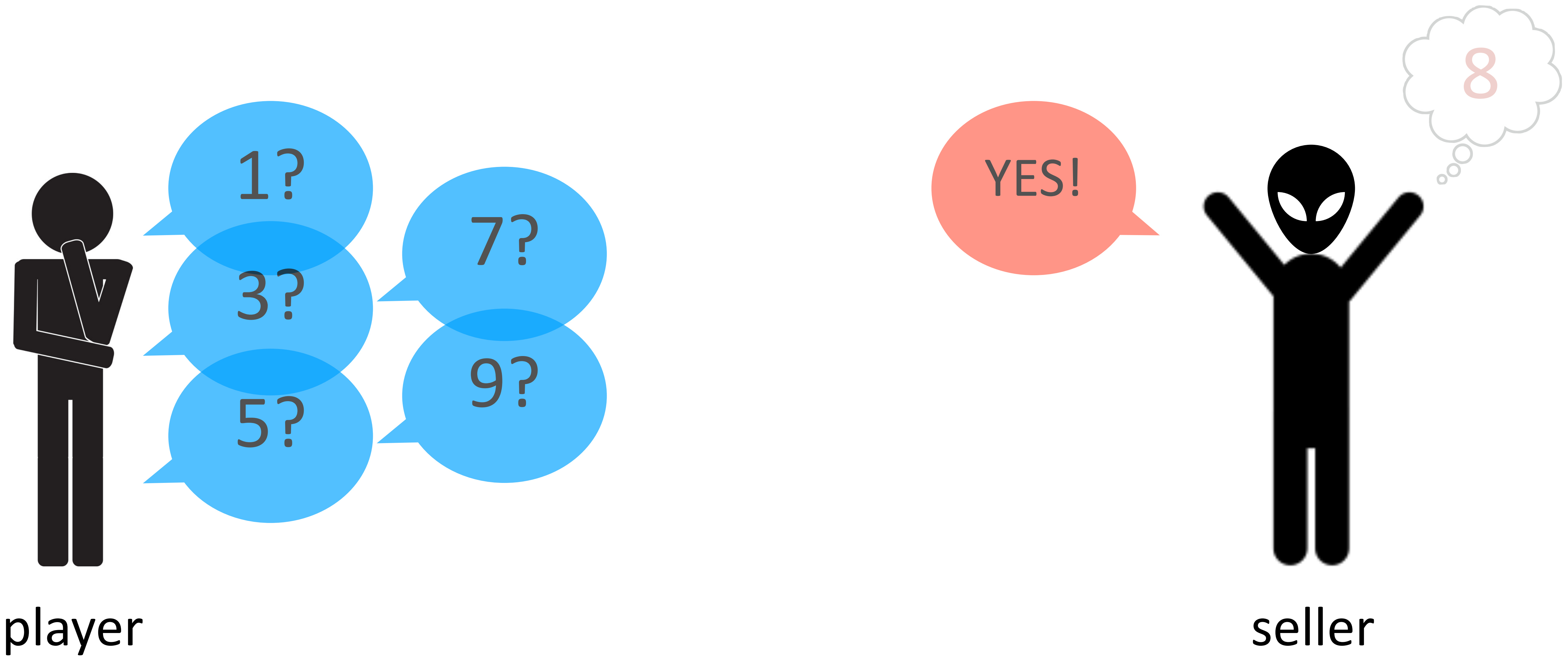
Online Bidding

- The online algorithm (player) wants to guess the target number $t \geq 1$, which is unknown to it. To this end, the player submits a sequence $B = \{b_1, b_2, b_3, \cdots\}$ of *bids*, until one of them with value higher than or equal to t.



Online Bidding

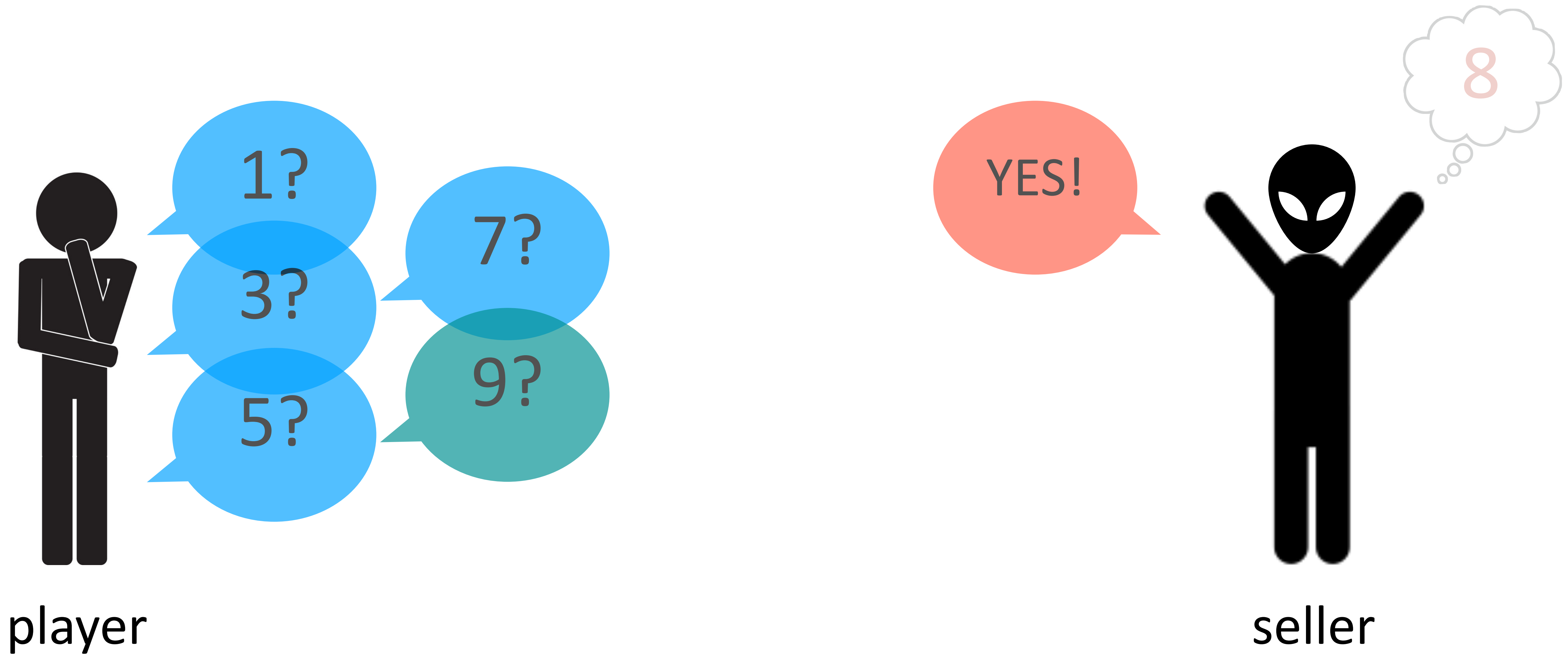
- The online algorithm (player) wants to guess the target number $t \geq 1$, which is unknown to it. To this end, the player submits a sequence $B = \{b_1, b_2, b_3, \cdots\}$ of *bids*, until one of them with value higher than or equal to t.



Online Bidding

- The online algorithm (player) wants to guess the target number $t \geq 1$, which is unknown to it. To this end, the player submits a sequence $B = \{b_1, b_2, b_3, \cdots\}$ of *bids*, until one of them with value higher than or equal to t. The *cost* of the strategy of a player is defined by this sequence of bids, $\Sigma_{i=1}^{k} b_i$, where b_k is the first bid in B such that $b_k \geq t$.

Total cost:
1+3+5+7+9 = 25

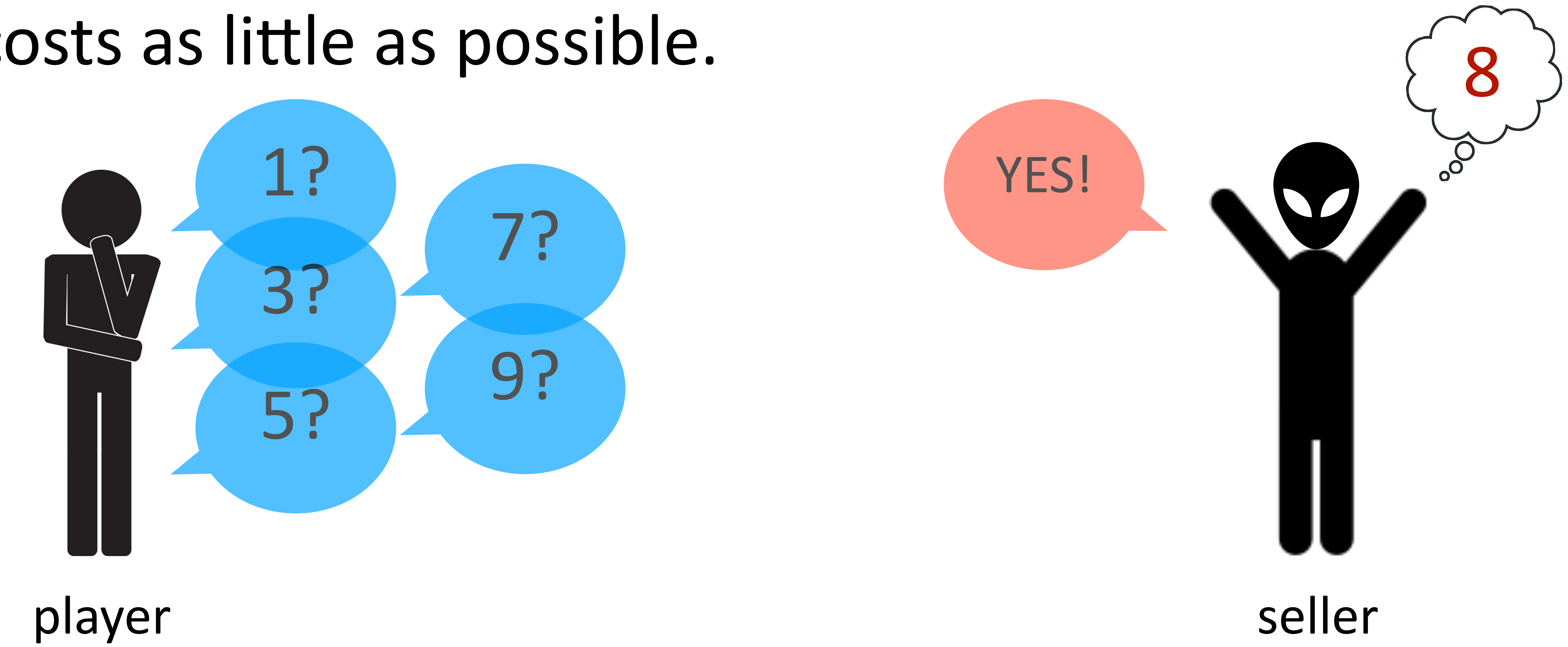


Online Bidding

- The online algorithm (player) wants to guess the target number $t \geq 1$, which is unknown to it. To this end, the player submits a sequence $B = \{b_1, b_2, b_3, \cdots\}$ of *bids*, until one of them with value higher than or equal to t. The *cost* of the strategy of a player is defined by this sequence of bids, $\Sigma_{i=1}^{k} b_i$, where b_k is the first bid in B such that $b_k \geq t$. A player's goal is to guess the target using a strategy with costs as little as possible.

Total cost:
1+3+5+7+9 = 25

Optimal cost:
8



Online Bidding

- The online algorithm (player) wants to guess the target number $t \geq 1$, which is unknown to it. To this end, the player submits a sequence $B = \{b_1, b_2, b_3, \cdots\}$ of *bids*, until one of them with value higher than or equal to t. The *cost* of the strategy of a player is defined by this sequence of bids, $\Sigma_{i=1}^{k} b_i$, where b_k is the first bid in B such that $b_k \geq t$. A player's goal is to guess the target using a strategy with costs as little as possible.
- Example:
  $t = 8$
  $B_1 = \{1,2,3,4,5,6,7,8,9,10,\cdots\}$
  $B_2 = \{1,3,5,7,9,\cdots\}$

Online Bidding

- The online algorithm (player) wants to guess the target number $t \geq 1$, which is unknown to it. To this end, the player submits a sequence $B = \{b_1, b_2, b_3, \cdots\}$ of *bids*, until one of them with value higher than or equal to t. The *cost* of the strategy of a player is defined by this sequence of bids, $\Sigma_{i=1}^{k} b_i$, where b_k is the first bid in B such that $b_k \geq t$. A player's goal is to guess the target using a strategy with costs as little as possible.
- Example:
  $t = 8$
  $B_1 = \{1,2,3,4,5,6,7,8,9,10,\cdots\}$ Cost of $B_1 = 1 + 2 + \cdots + 8 = 36$
  $B_2 = \{1,3,5,7,9,\cdots\}$

Online Bidding

- The online algorithm (player) wants to guess the target number $t \geq 1$, which is unknown to it. To this end, the player submits a sequence $B = \{b_1, b_2, b_3, \cdots\}$ of *bids*, until one of them with value higher than or equal to t. The *cost* of the strategy of a player is defined by this sequence of bids, $\Sigma_{i=1}^{k} b_i$, where b_k is the first bid in B such that $b_k \geq t$. A player's goal is to guess the target using a strategy with costs as little as possible.
- Example:

  $t = 8$

  $B_1 = \{1,2,3,4,5,6,7,8,9,10,\cdots\}$ Cost of $B_1 = 1 + 2 + \cdots + 8 = 36$

  $B_2 = \{1,3,5,7,9,\cdots\}$ Cost of $B_2 = 1 + 3 + 5 + 7 + 9 = 25$

Online Bidding

- The online algorithm (player) wants to guess the target number $t \geq 1$, which is unknown to it. To this end, the player submits a sequence $B = \{b_1, b_2, b_3, \cdots\}$ of *bids*, until one of them with value higher than or equal to t. The *cost* of the strategy of a player is defined by this sequence of bids, $\Sigma_{i=1}^{k} b_i$, where b_k is the first bid in B such that $b_k \geq t$. A player's goal is to guess the target using a strategy with costs as little as possible.
- Example:

$$B_1(8) = 4.5 \cdot \text{OPT}(8)$$

$t = 8$

$B_1 = \{1,2,3,4,5,6,7,8,9,10,\cdots\}$ Cost of $B_1 = 1 + 2 + \cdots + 8 = 36$

$B_2 = \{1,3,5,7,9,\cdots\}$ Cost of $B_2 = 1 + 3 + 5 + 7 + 9 = 25$

Online Bidding

- The online algorithm (player) wants to guess the target number $t \geq 1$, which is unknown to it. To this end, the player submits a sequence $B = \{b_1, b_2, b_3, \cdots\}$ of *bids*, until one of them with value higher than or equal to t. The *cost* of the strategy of a player is defined by this sequence of bids, $\Sigma_{i=1}^{k} b_i$, where b_k is the first bid in B such that $b_k \geq t$. A player's goal is to guess the target using a strategy with costs as little as possible.
- Example:

$B_1(8) = 4.5 \cdot \text{OPT}(8)$

$t = 8$

$B_1 = \{1,2,3,4,5,6,7,8,9,10,\cdots\}$ Cost of $B_1 = 1 + 2 + \cdots + 8 = 36$

$B_2 = \{1,3,5,7,9,\cdots\}$ Cost of $B_2 = 1 + 3 + 5 + 7 + 9 = 25$

$B_2(8) = 3.125 \cdot \text{OPT}(8)$

Online Bidding

- The online algorithm (player) wants to guess the target number $t \geq 1$, which is unknown to it. To this end, the player submits a sequence $B = \{b_1, b_2, b_3, \cdots\}$ of *bids*, until one of them with value higher than or equal to t. The *cost* of the strategy of a player is defined by this sequence of bids, $\Sigma_{i=1}^{k} b_i$, where b_k is the first bid in B such that $b_k \geq t$. A player's goal is to guess the target using a strategy with costs as little as possible.
  - Instance: t

Online Bidding

- The online algorithm (player) wants to guess the target number $t \geq 1$, which is unknown to it. To this end, the player submits a sequence $B = \{b_1, b_2, b_3, \cdots\}$ of *bids*, until one of them with value higher than or equal to t. The *cost* of the strategy of a player is defined by this sequence of bids, $\Sigma_{i=1}^{k} b_i$, where b_k is the first bid in B such that $b_k \geq t$. A player's goal is to guess the target using a strategy with costs as little as possible.
    - Instance: t
    - Uncertain factor: we don't know what t is

Online Bidding

- The online algorithm (player) wants to guess the target number $t \geq 1$, which is unknown to it. To this end, the player submits a sequence $B = \{b_1, b_2, b_3, \cdots\}$ of *bids*, until one of them with value higher than or equal to t. The *cost* of the strategy of a player is defined by this sequence of bids, $\Sigma_{i=1}^{k} b_i$, where b_k is the first bid in B such that $b_k \geq t$. A player's goal is to guess the target using a strategy with costs as little as possible.
  - Instance: t
  - Uncertain factor: we don't know what t is
  - The adversary knows the player's strategy and can choose a worst-case target t

Outline

- Recap from the last lecture
- **Online bidding**
    - A 4-competitive algorithm
- A general technique for designing online algorithms: **doubling**
- Different types of adversaries

Outline

- Recap from the last lecture
- **Online bidding**
  - A 4-competitive algorithm
- A general technique for designing online algorithms: **doubling**
- Different types of adversaries

Double Algorithm

Double: $b_i = 2^{i-1}$

Double Algorithm

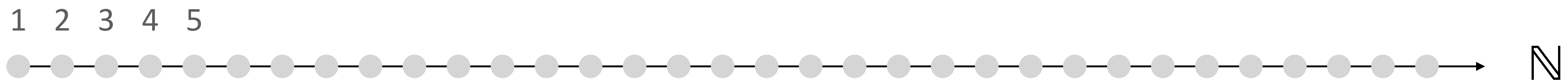
Double: $b_i = 2^{i-1}$

- $B = \{1,2,4,8,16,32,\cdots\}$

Double Algorithm

Double: $b_i = 2^{i-1}$

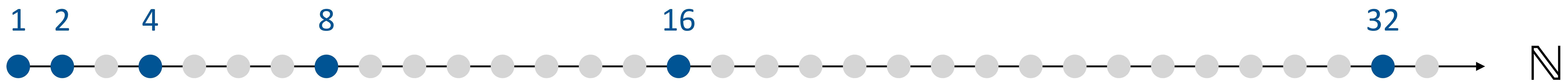
- $B = \{1,2,4,8,16,32,\cdots\}$



Double Algorithm

Double: $b_i = 2^{i-1}$

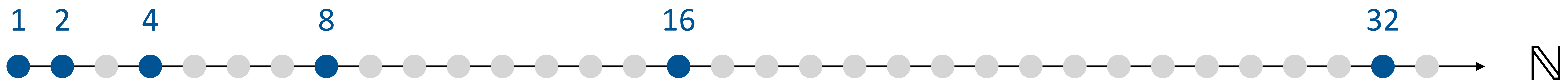
- $B = \{1,2,4,8,16,32,\cdots\}$



Double Algorithm

Double: $b_i = 2^{i-1}$

- How bad is **Double**?



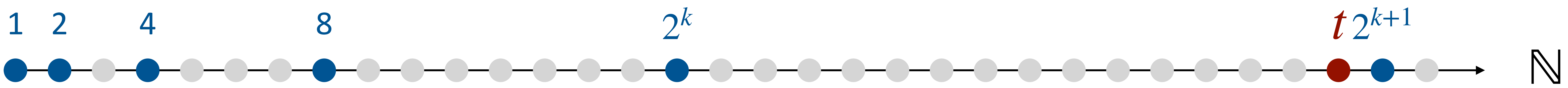
Double Algorithm

Double: $b_i = 2^{i-1}$

- How bad is **Double**?

Consider the adversary where $t = 2^k + x$ for some $k \geq 1$ and $x \in [1, 2^k - 1]$.

OPT $= 2^k + x$, and **Double** $= 1 + 2 + 4 + \cdots + 2^k + 2^{k+1} = 2 \cdot 2^{k+1}$.



$$\frac{\text{ALG}(t = 2^k + 1)}{\text{OPT}(t = 2^k + 1)} = \frac{2 \cdot 2^{k+1}}{2^k + x} = \frac{4 \cdot (2^k + x) - 4x}{2^k + x} = 4 - \frac{4x}{2^k + x} \geq 4 - \frac{4}{2^k + x}$$

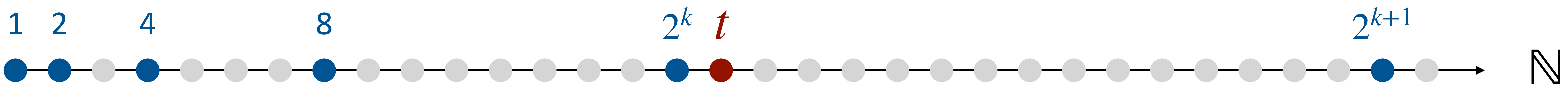
Double Algorithm

Double: $b_i = 2^{i-1}$

- How bad is **Double**?

  Consider the adversary where $t = 2^k + 1$ for some $k \geq 1$.

  OPT $= 2^k + 1$, and **Double** $= 1 + 2 + 4 + \cdots + 2^k + 2^{k+1} = 2 \cdot 2^{k+1}$.



$$\frac{\text{ALG}(t = 2^k + 1)}{\text{OPT}(t = 2^k + 1)} = \frac{2 \cdot 2^{k+1}}{2^k + 1} = \frac{4 \cdot (2^k + 1) - 4}{2^k + 1} = 4 - \frac{4}{2^k + 1}$$

Double Algorithm

Double: $b_i = 2^{i-1}$

- Theorem: The **Double** algorithm attains a competitive ratio 4

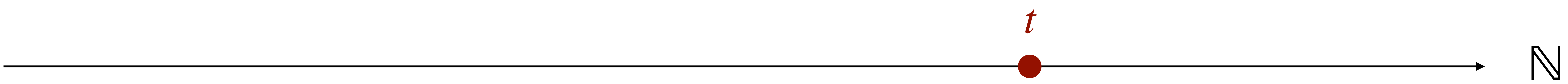
Double Algorithm

Double: $b_i = 2^{i-1}$

- Theorem: The **Double** algorithm attains a competitive ratio 4

<Proof idea>

Assume that the target is t.



Double Algorithm

Double: $b_i = 2^{i-1}$

- Theorem: The **Double** algorithm attains a competitive ratio 4

<Proof idea>

Assume that the target is t. It must be between $2^k + 1$ to 2^{k+1} for some $k \geq 0$.

Double Algorithm

Double: $b_i = 2^{i-1}$

- Theorem: The **Double** algorithm attains a competitive ratio 4

<Proof idea>

$\text{OPT} = t$

Assume that the target is t. It must be between $2^k + 1$ to 2^{k+1} for some $k \geq 0$.

In this case, Double hits the target at bid 2^{k+1} and pays $1 + 2 + 4 + \cdots + 2^k + 2^{k+1}$.

$$\frac{\text{Double}(t)}{\text{OPT}(t)} = \frac{1 + 2 + 4 + \cdots + 2^{k+1}}{t}$$

Double Algorithm

Double: $b_i = 2^{i-1}$

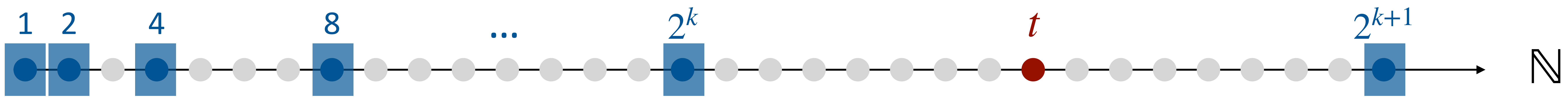
- Theorem: The **Double** algorithm attains a competitive ratio 4

<Proof idea>

$\text{OPT} = t$

Assume that the target is t. It must be between $2^k + 1$ to 2^{k+1} for some $k \geq 0$.

In this case, Double hits the target at bid 2^{k+1} and pays $1 + 2 + 4 + \cdots + 2^k + 2^{k+1}$.



$$\frac{\text{Double}(t)}{\text{OPT}(t)} = \frac{1 + 2 + 4 + \cdots + 2^{k+1}}{t} < \frac{2 \cdot 2^k}{t}$$

2 is a magic number:
$1 + 2^0 + 2^1 + 2^2 + \cdots 2^i = 2 \cdot 2^i$

Double Algorithm

Double: $b_i = 2^{i-1}$

- Theorem: The **Double** algorithm attains a competitive ratio 4

<Proof idea>

$\text{OPT} = t$

Assume that the target is t. It must be between $2^k + 1$ to 2^{k+1} for some $k \geq 0$.

In this case, Double hits the target at bid 2^{k+1} and pays $1 + 2 + 4 + \cdots + 2^k + 2^{k+1}$.

$$\frac{\text{Double}(t)}{\text{OPT}(t)} = \frac{1 + 2 + 4 + \cdots + 2^{k+1}}{t} < \frac{2 \cdot 2^k}{t} \leq \frac{2^{k+2}}{2^k + 1}$$

$t \in (2^k, 2^{k+1}]$

$t \geq 2^k + 1$

Double Algorithm

Double: $b_i = 2^{i-1}$

- Theorem: The **Double** algorithm attains a competitive ratio 4

<Proof idea>

OPT $= t$

Assume that the target is t. It must be between $2^k + 1$ to 2^{k+1} for some $k \geq 0$.

In this case, Double hits the target at bid 2^{k+1} and pays $1 + 2 + 4 + \cdots + 2^k + 2^{k+1}$.

$$\frac{\text{Double}(t)}{\text{OPT}(t)} = \frac{1 + 2 + 4 + \cdots + 2^{k+1}}{t} < \frac{2 \cdot 2^k}{t} \leq \frac{2^{k+2}}{2^k + 1} < \frac{2^{k+2}}{2^k} = 4$$

What Happened

- For the analysis of the **Double** algorithm for the online bidding problem, we assume that the optimal solution is t, and then show that **the algorithm cost cannot be too much than** t by the behavior of the **Double** algorithm:
  - The (assumed) value of t tells us what the last bid is (2^{k+1})

What Happened

- For the analysis of the **Double** algorithm for the online bidding problem, we assume that the optimal solution is t, and then show that **the algorithm cost cannot be too much than** t by the behavior of the **Double** algorithm:
  - The (assumed) value of t tells us what the last bid is (2^{k+1})

What Happened

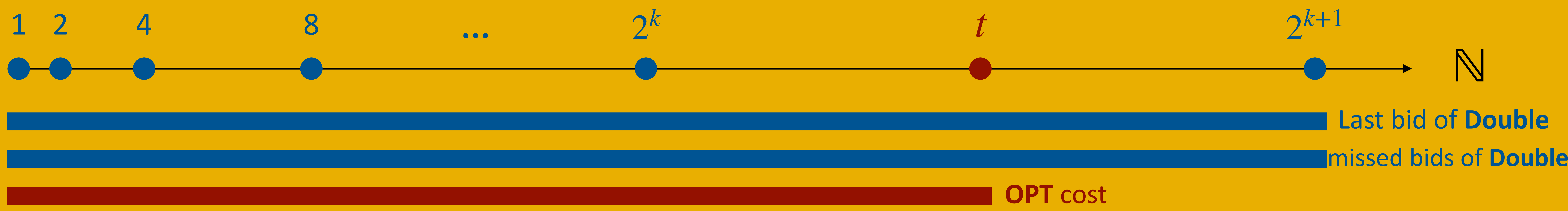
- For the analysis of the **Double** algorithm for the online bidding problem, we assume that the optimal solution is t, and then show that **the algorithm cost cannot be too much than** t by the behavior of the **Double** algorithm:
  - The (assumed) value of t tells us what the last bid is (2^{k+1})
  - At the last bid 2^{k+1} of **Double**, the total cost paid *before* this bid equals to 2^{k+1}

What Happened

- For the analysis of the **Double** algorithm for the online bidding problem, we assume that the optimal solution is t, and then show that **the algorithm cost cannot be too much than** t by the behavior of the **Double** algorithm:
  - The (assumed) value of t tells us what the last bid is (2^{k+1})
  - At the last bid 2^{k+1} of **Double**, the total cost paid *before* this bid equals to 2^{k+1}

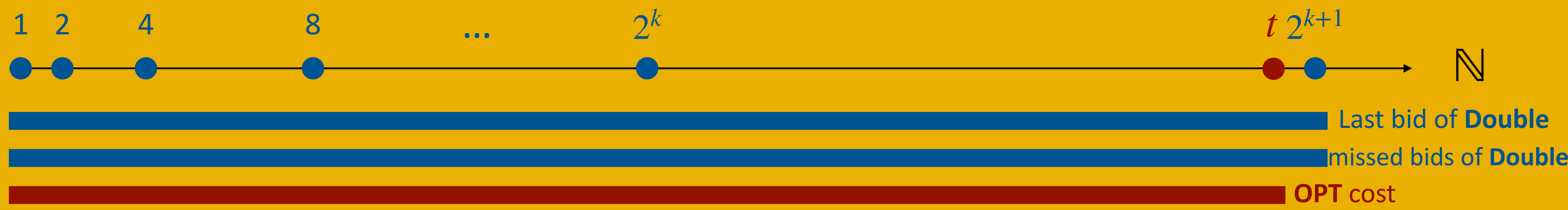
What Happened

- For the analysis of the **Double** algorithm for the online bidding problem, we assume that the optimal solution is t, and then show that **the algorithm cost cannot be too much than** t by the behavior of the **Double** algorithm:
  - The (assumed) value of t tells us what the last bid is (2^{k+1})
  - At the last bid 2^{k+1} of **Double**, the total cost paid *before* this bid equals to 2^{k+1}



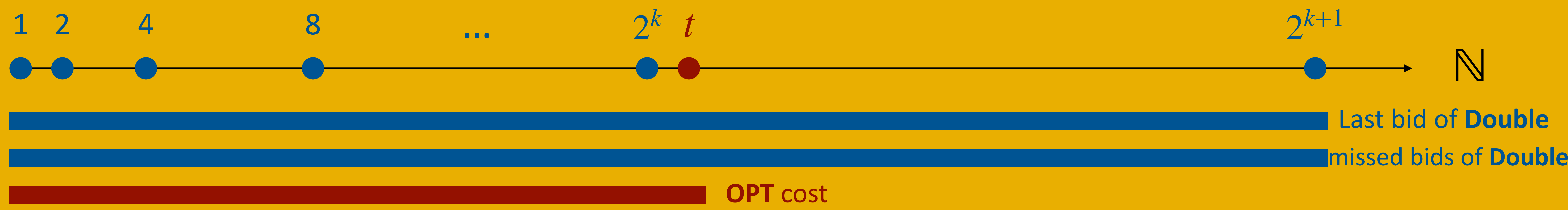
What Happened

- For the analysis of the **Double** algorithm for the online bidding problem, we assume that the optimal solution is t, and then show that **the algorithm cost cannot be too much than** t by the behavior of the **Double** algorithm:
  - The (assumed) value of t tells us what the last bid is (2^{k+1})
  - At the last bid 2^{k+1} of **Double**, the total cost paid *before* this bid equals to 2^{k+1}



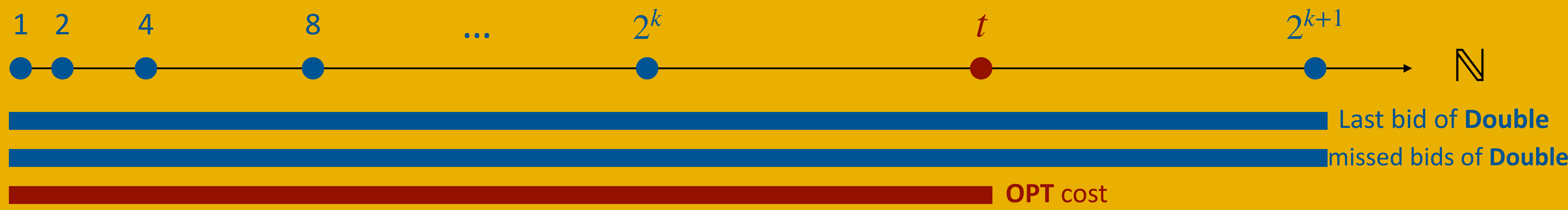
What Happened

- For the analysis of the **Double** algorithm for the online bidding problem, we assume that the optimal solution is t, and then show that **the algorithm cost cannot be too much than** t by the behavior of the **Double** algorithm:
  - The (assumed) value of t tells us what the last bid is (2^{k+1})
  - At the last bid 2^{k+1} of **Double**, the total cost paid *before* this bid equals to 2^{k+1}



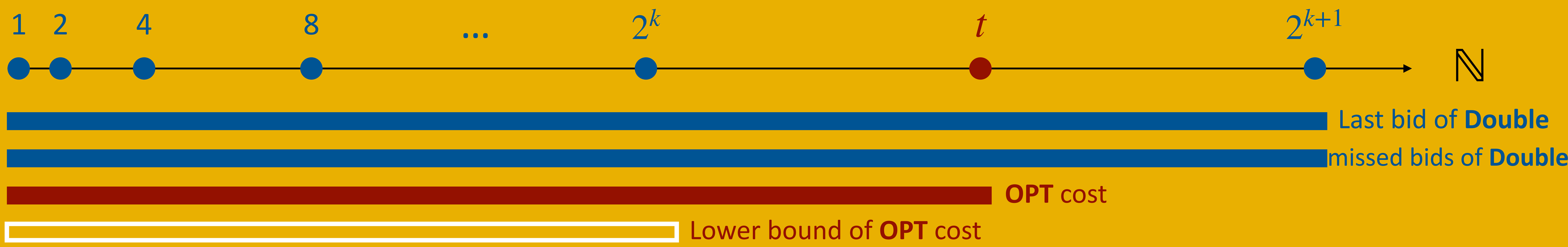
What Happened

- For the analysis of the **Double** algorithm for the online bidding problem, we assume that the optimal solution is t, and then show that **the algorithm cost cannot be too much than** t by the behavior of the **Double** algorithm:
    - The (assumed) value of t tells us what the last bid is (2^{k+1})
    - At the last bid 2^{k+1} of **Double**, the total cost paid *before* this bid equals to 2^{k+1}



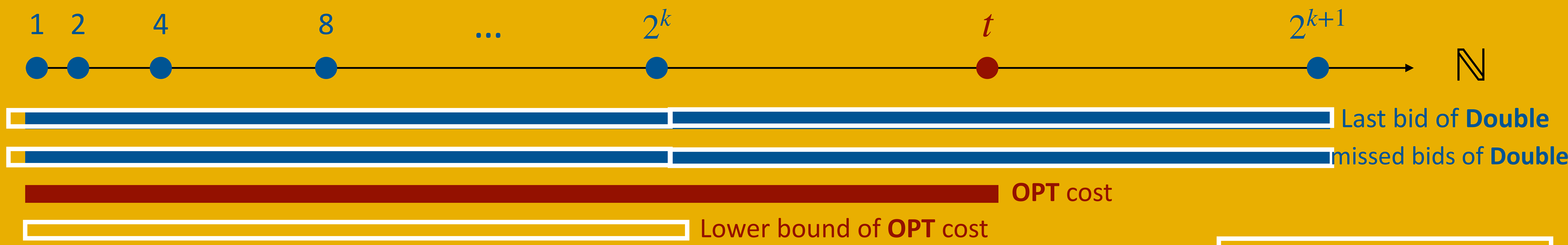
What Happened

- For the analysis of the **Double** algorithm for the online bidding problem, we assume that the optimal solution is t, and then show that **the algorithm cost cannot be too much than** t by the behavior of the **Double** algorithm:
  - The (assumed) value of t tells us what the last bid is (2^{k+1})
  - At the last bid 2^{k+1} of **Double**, the total cost paid *before* this bid equals to 2^{k+1}



What Happened

- For the analysis of the **Double** algorithm for the online bidding problem, we assume that the optimal solution is t, and then show that **the algorithm cost cannot be too much than** t by the behavior of the **Double** algorithm:
  - The (assumed) value of t tells us what the last bid is (2^{k+1})
  - At the last bid 2^{k+1} of **Double**, the total cost paid *before* this bid equals to 2^{k+1}



Double $< 4 \cdot$ **OPT**

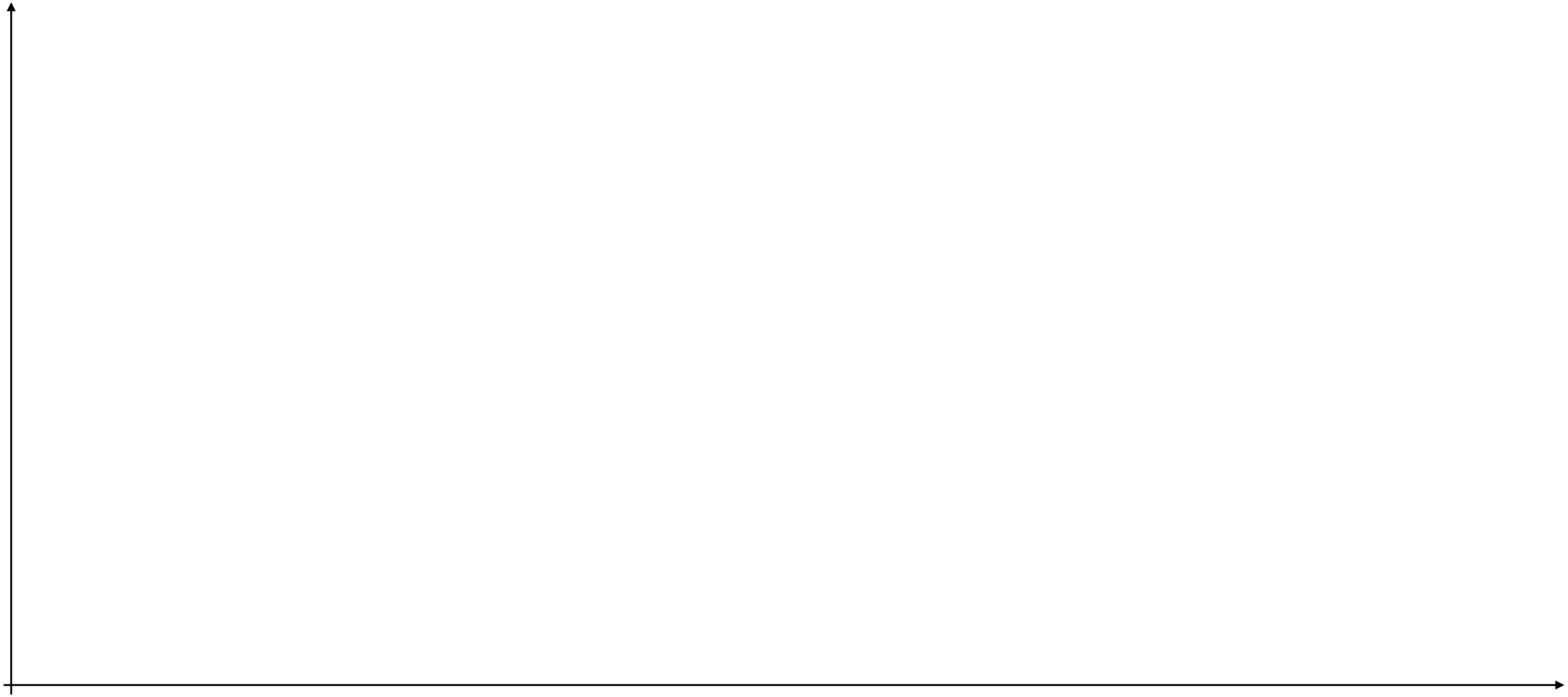
Outline

- Recap from the last lecture
- **Online bidding**
  - A 4-competitive algorithm
- A general technique for designing online algorithms: **doubling**
- “Best” online algorithms

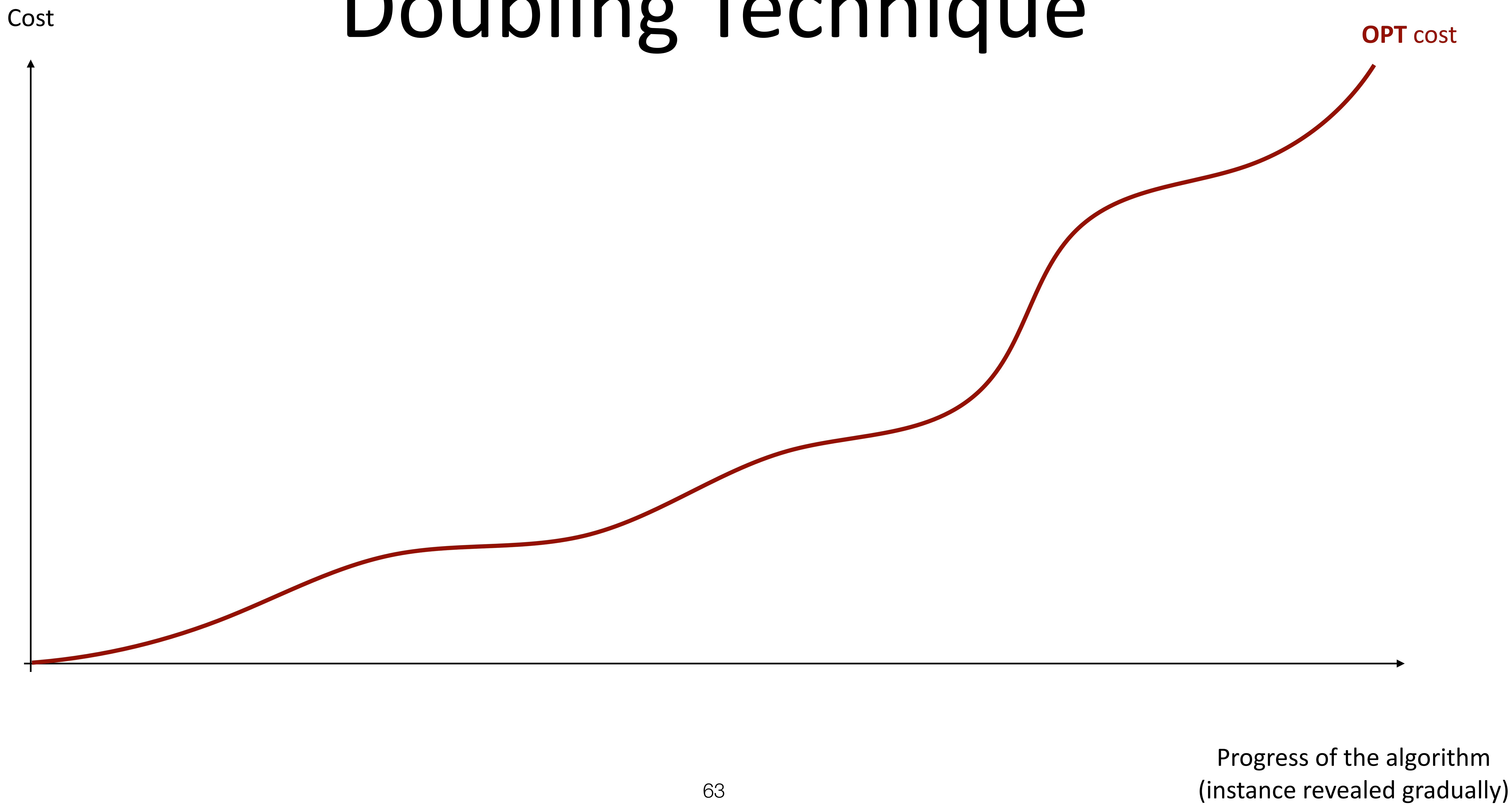
Doubling Technique

- For some problems, the optimal cost increases as the instance is revealed
  - We can design a doubling algorithm **DOUBLE** with a parameter r as follows:
    - We keep track of the value of the optimal cost OPT for the current instance
    - **DOUBLE** works in phases
      - The phases is decided by the value of OPT; the phase i starts at the time when OPT is at least r^i
      - If **DOUBLE** never creates (total) cost that exceeds $c \cdot (r^i - r^{i-1})$ in every phase i, **DOUBLE** is c-competitive

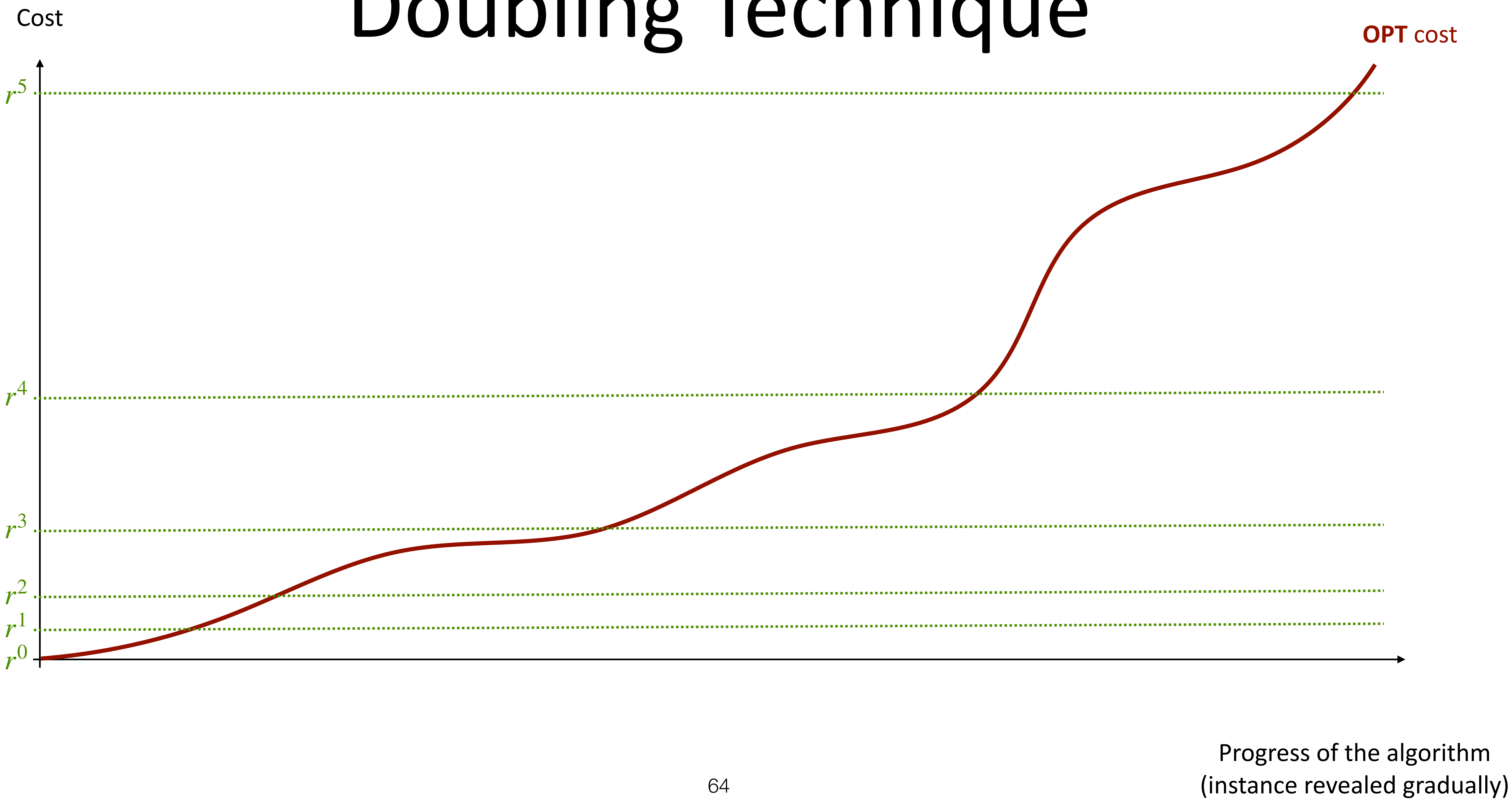
Doubling Technique



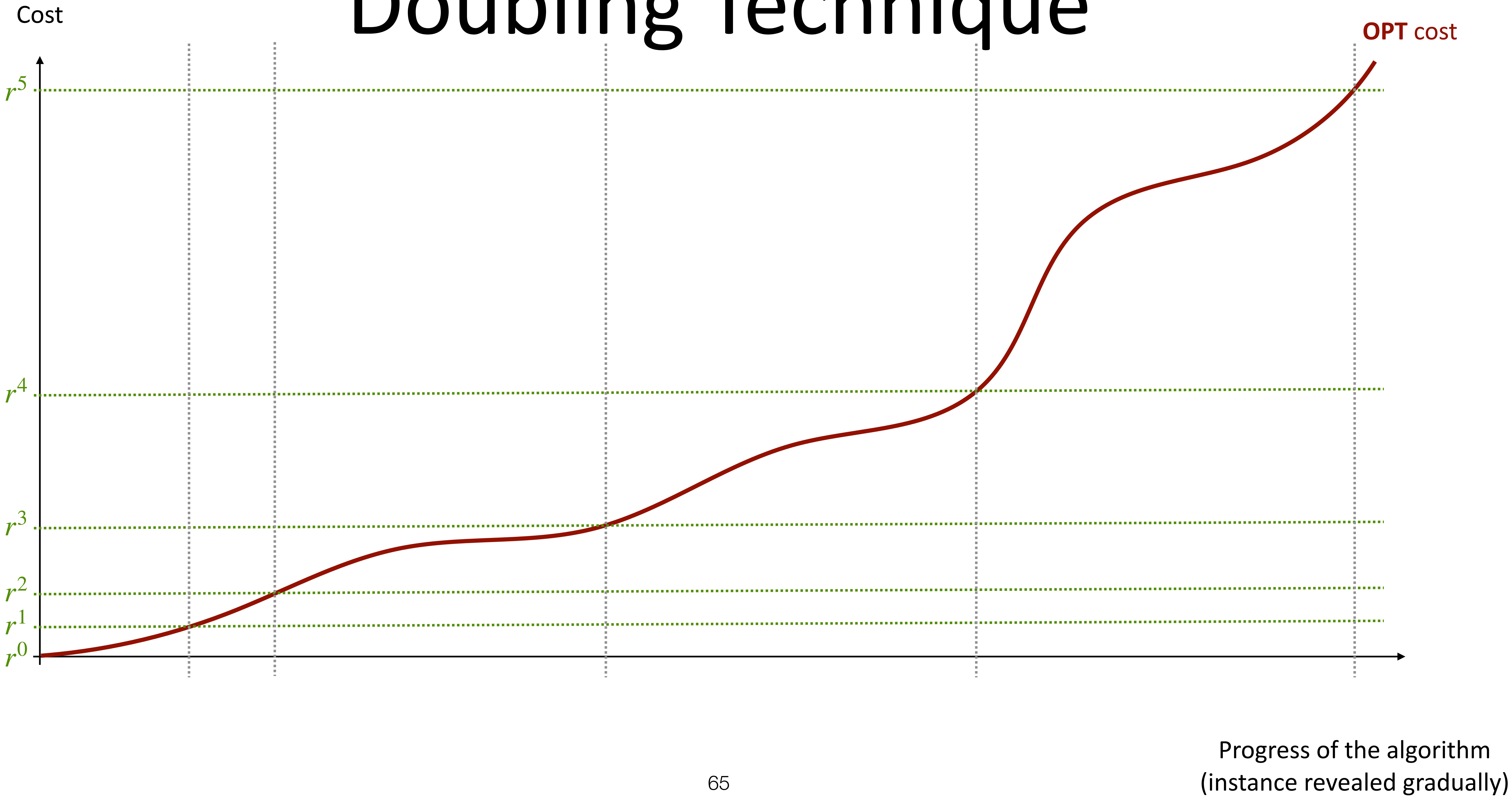
Doubling Technique



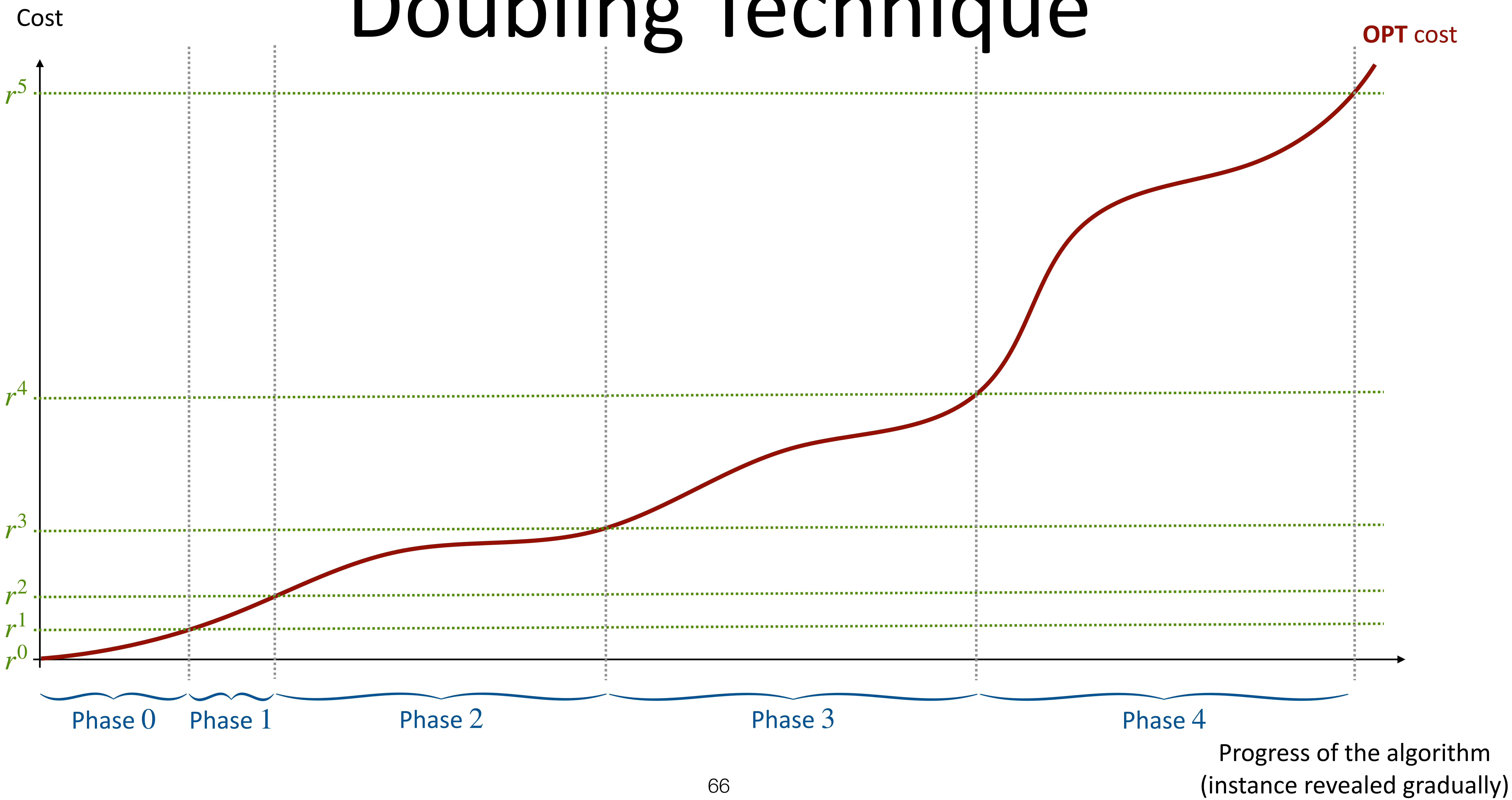
Doubling Technique



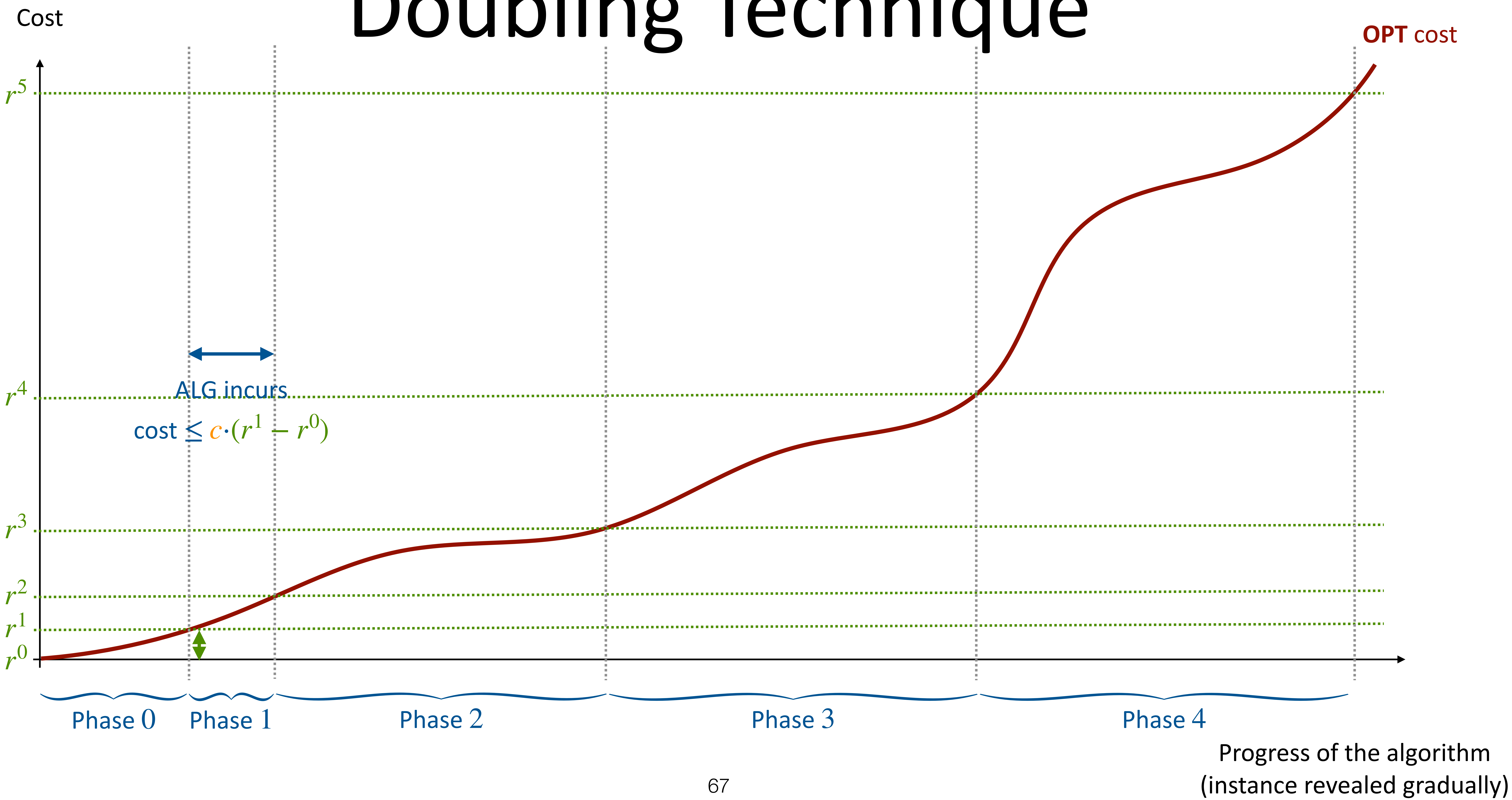
Doubling Technique



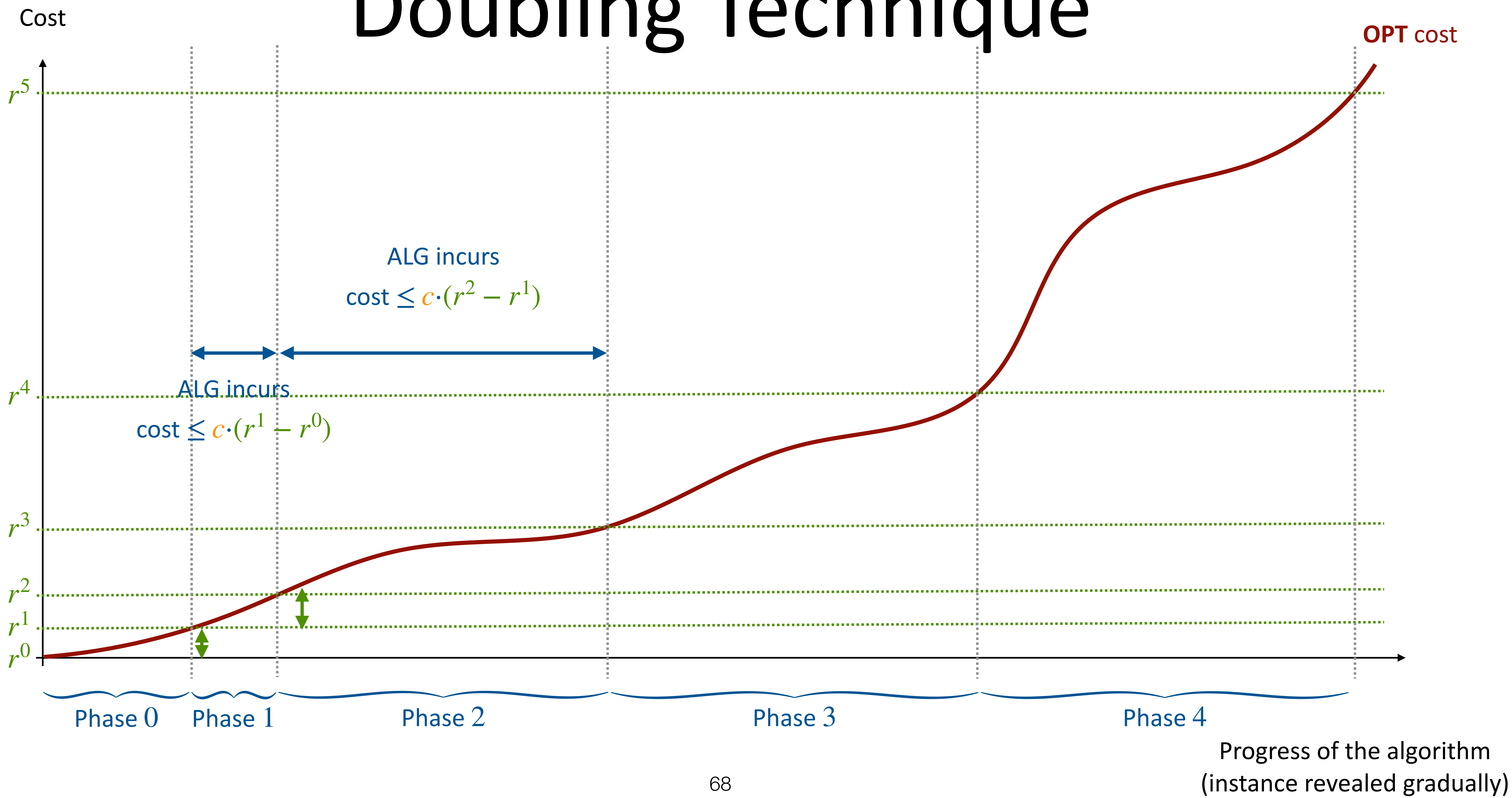
Doubling Technique



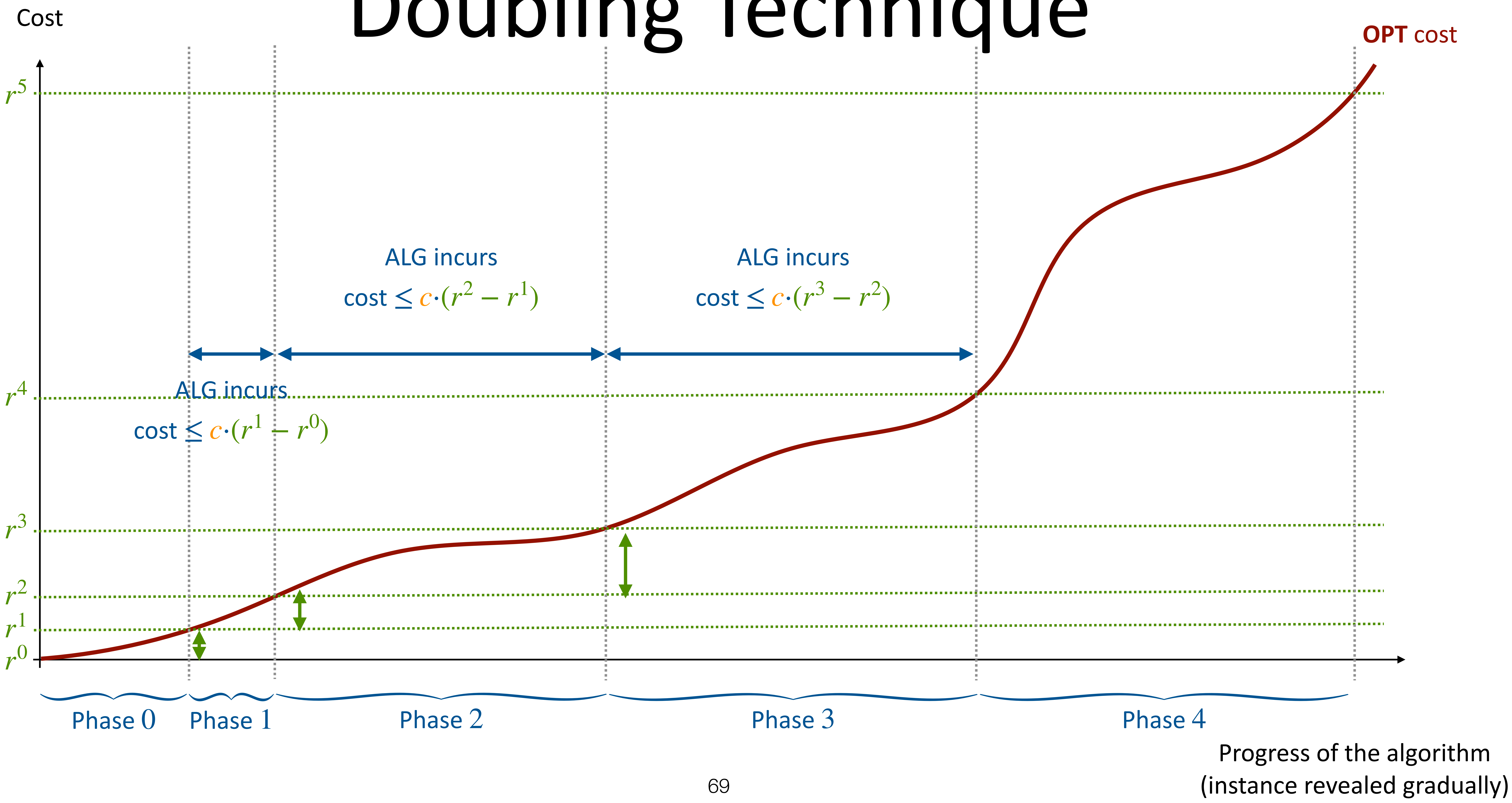
Doubling Technique



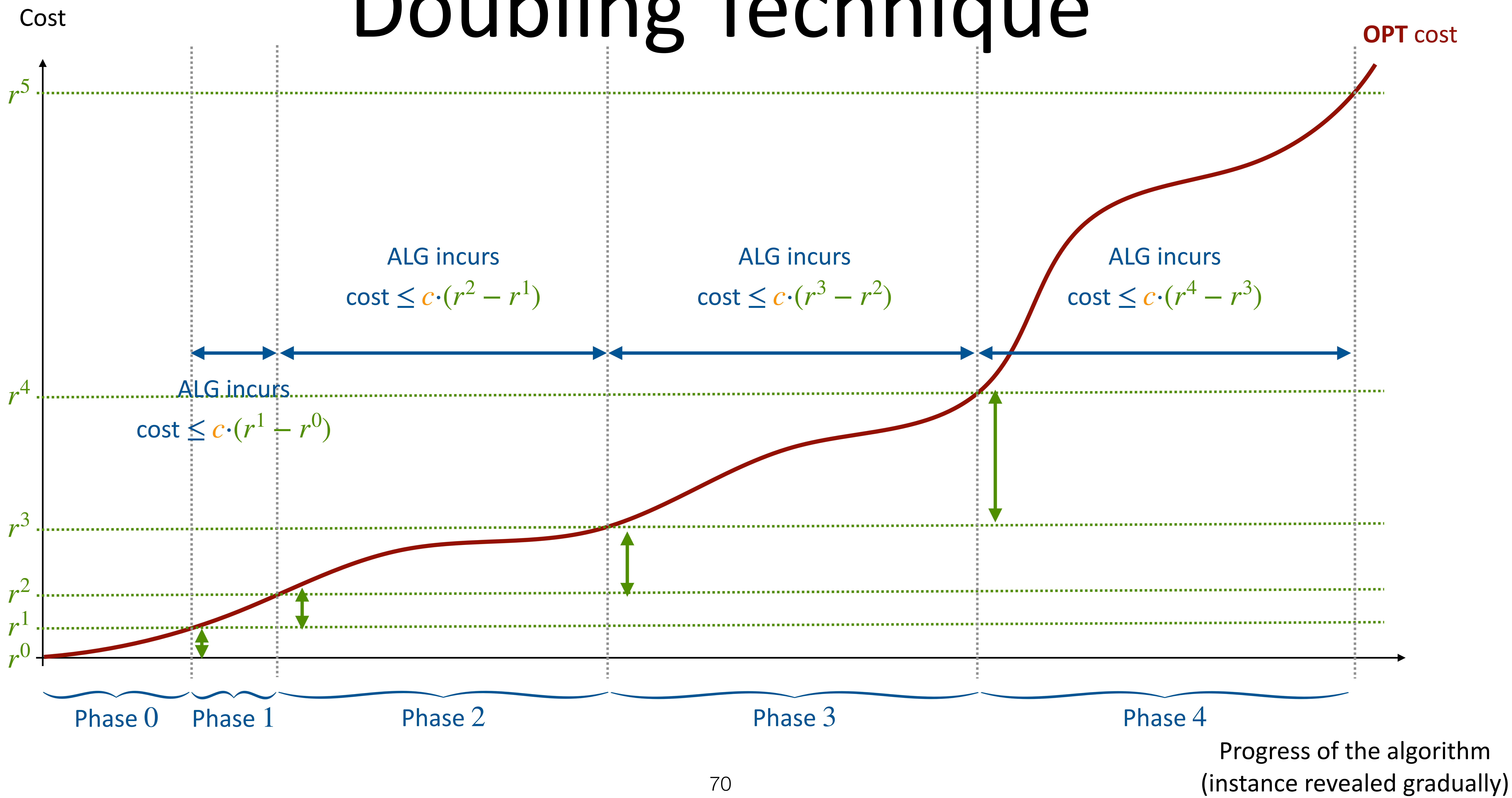
Doubling Technique



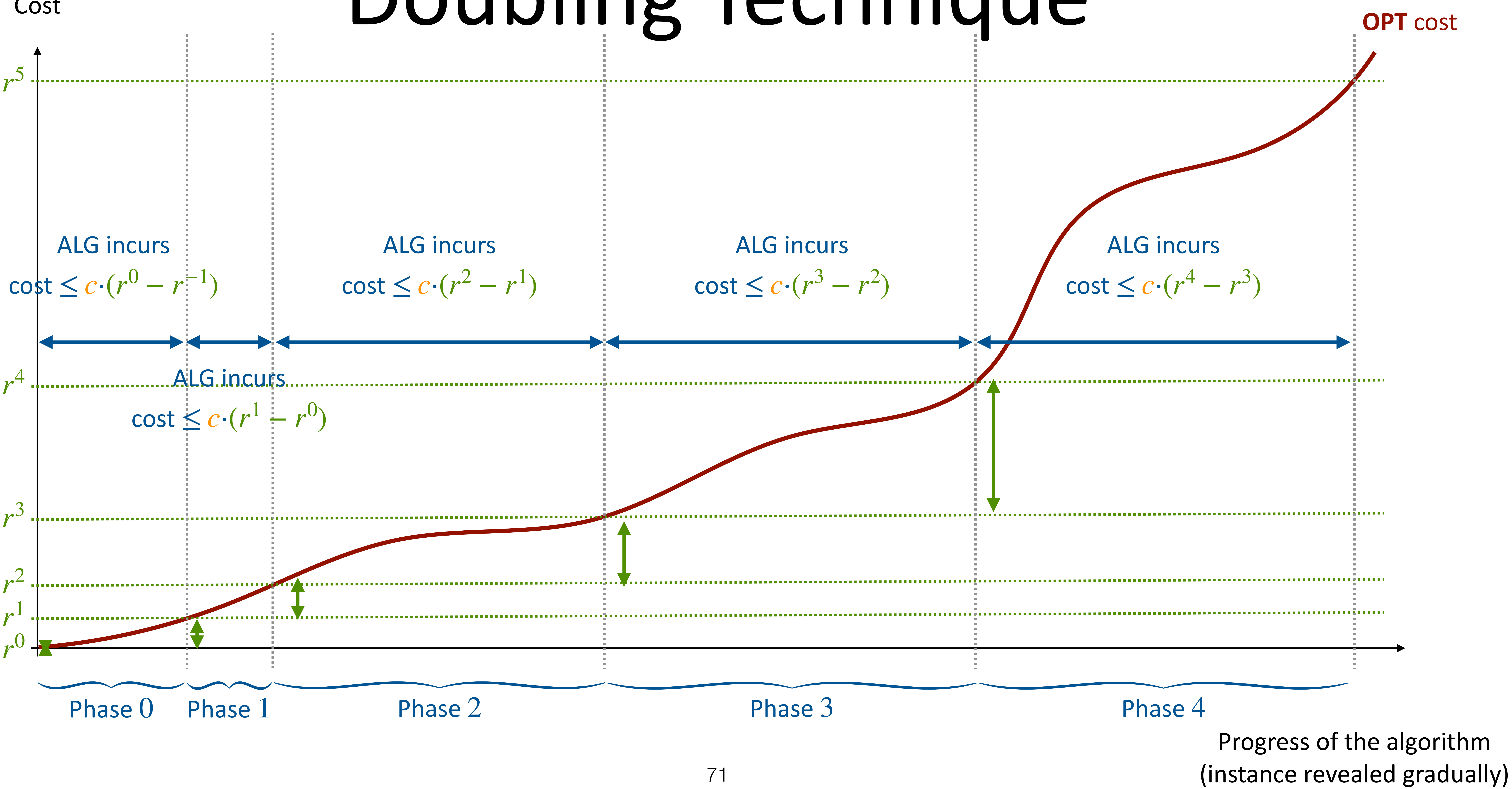
Doubling Technique



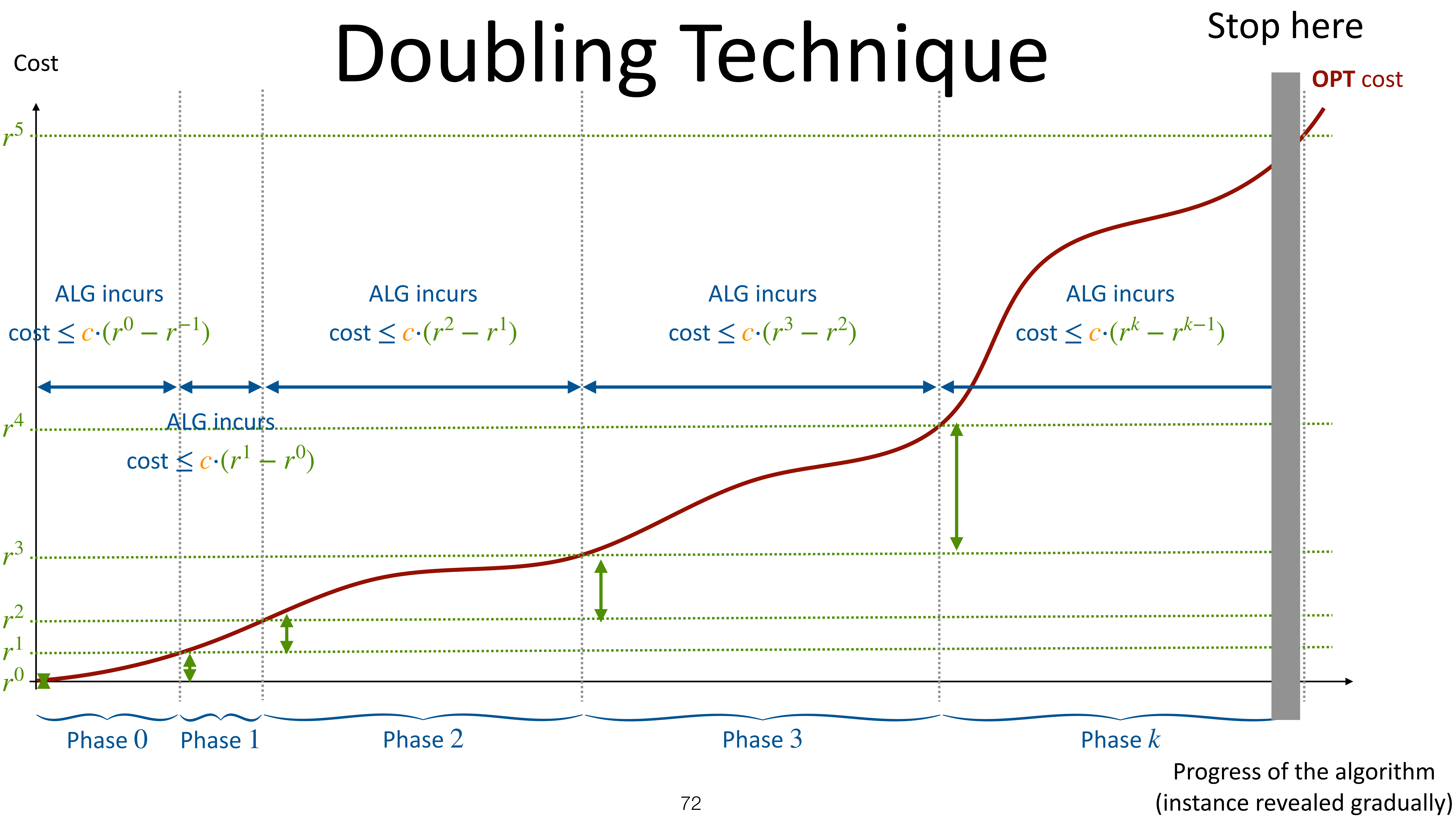
Doubling Technique



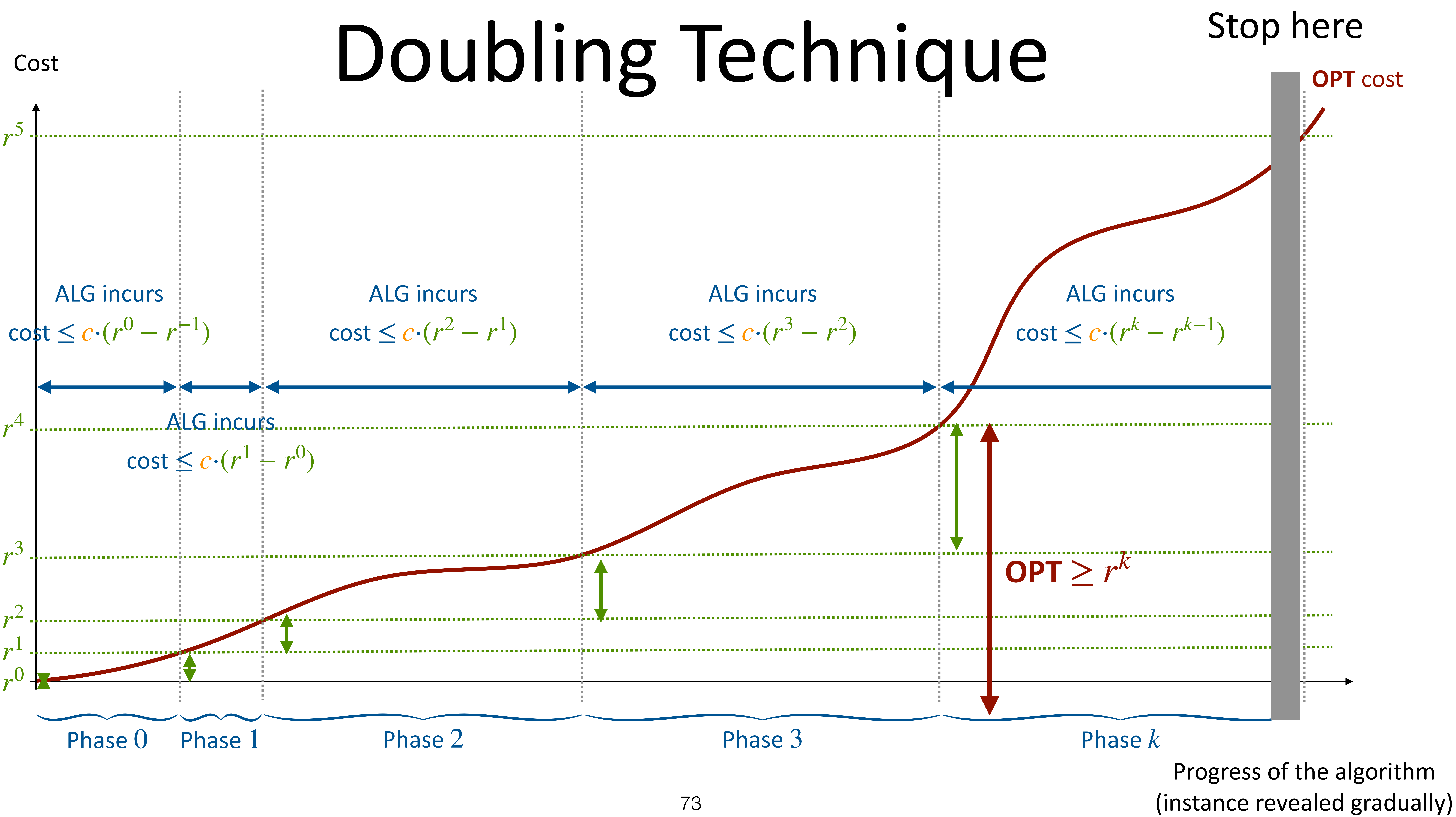
Doubling Technique



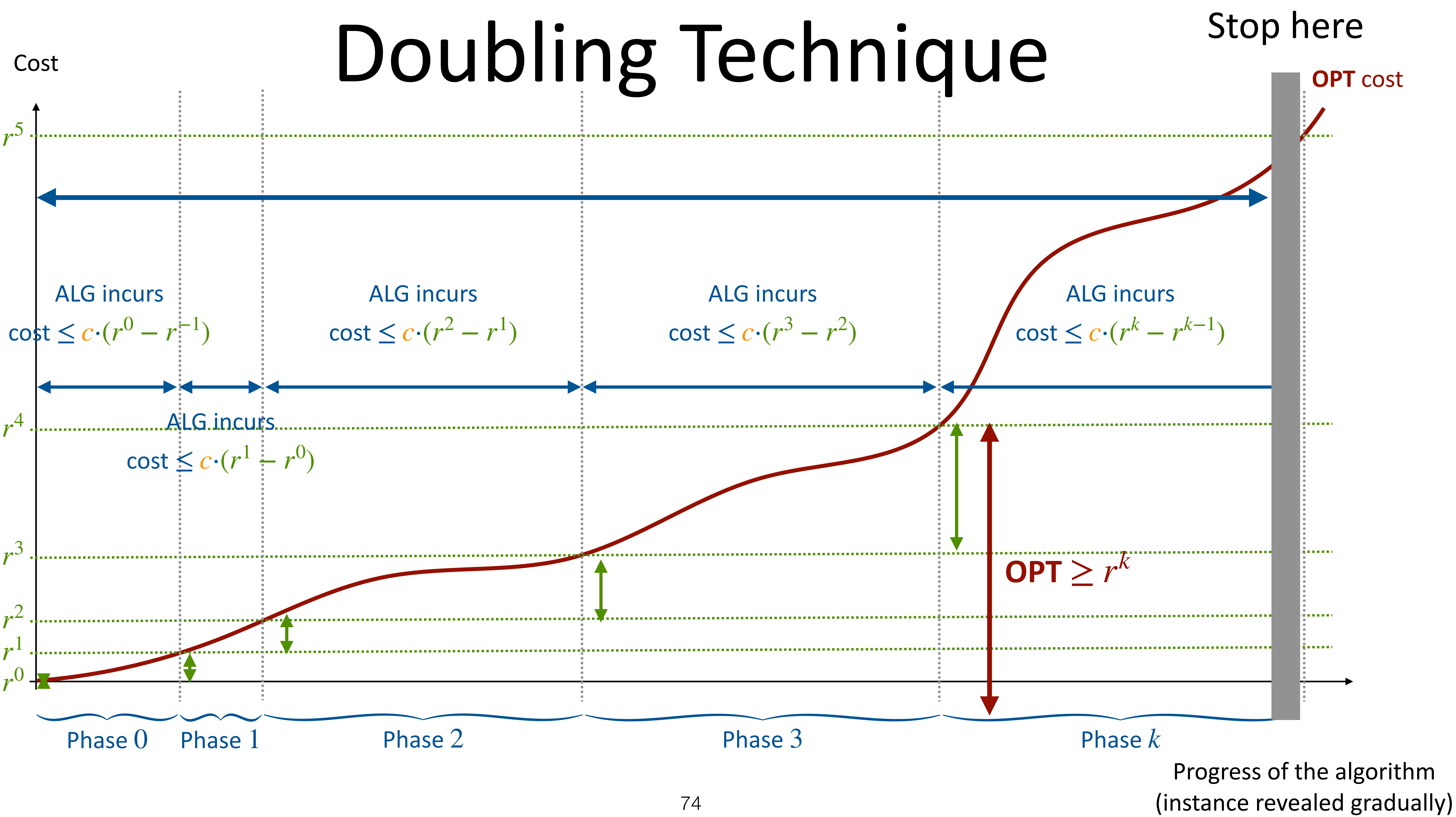
Doubling Technique



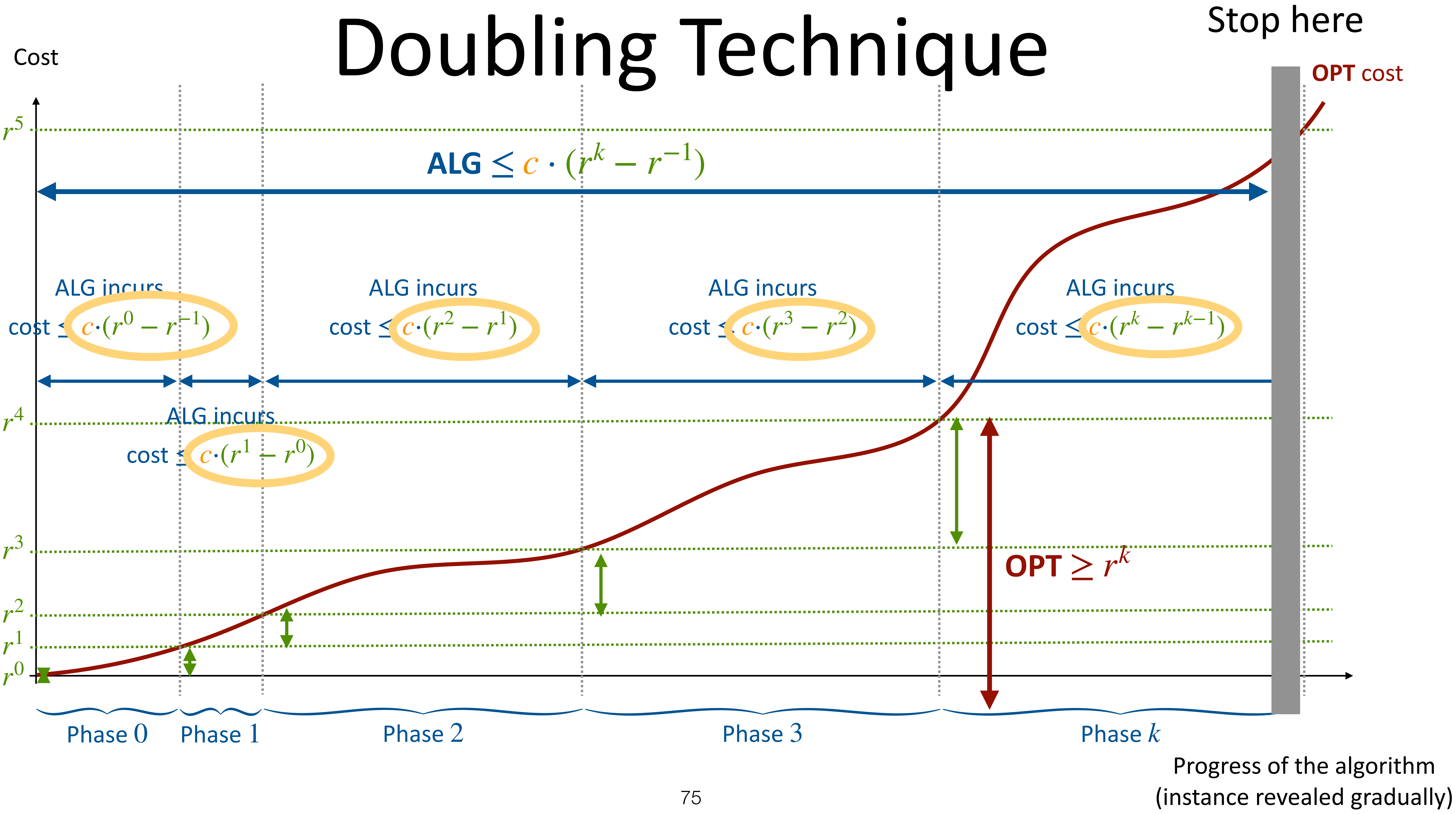
Doubling Technique



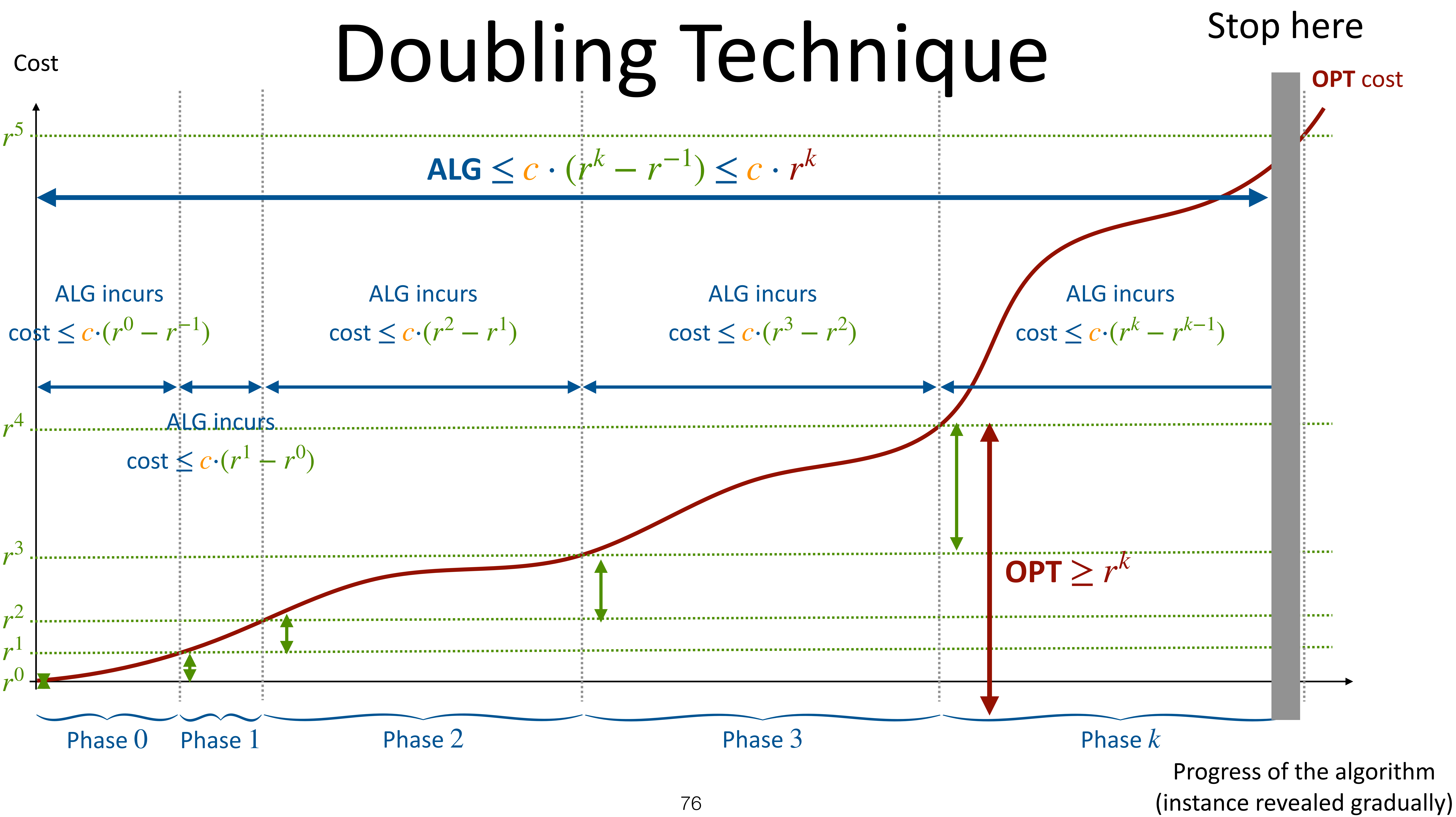
Doubling Technique



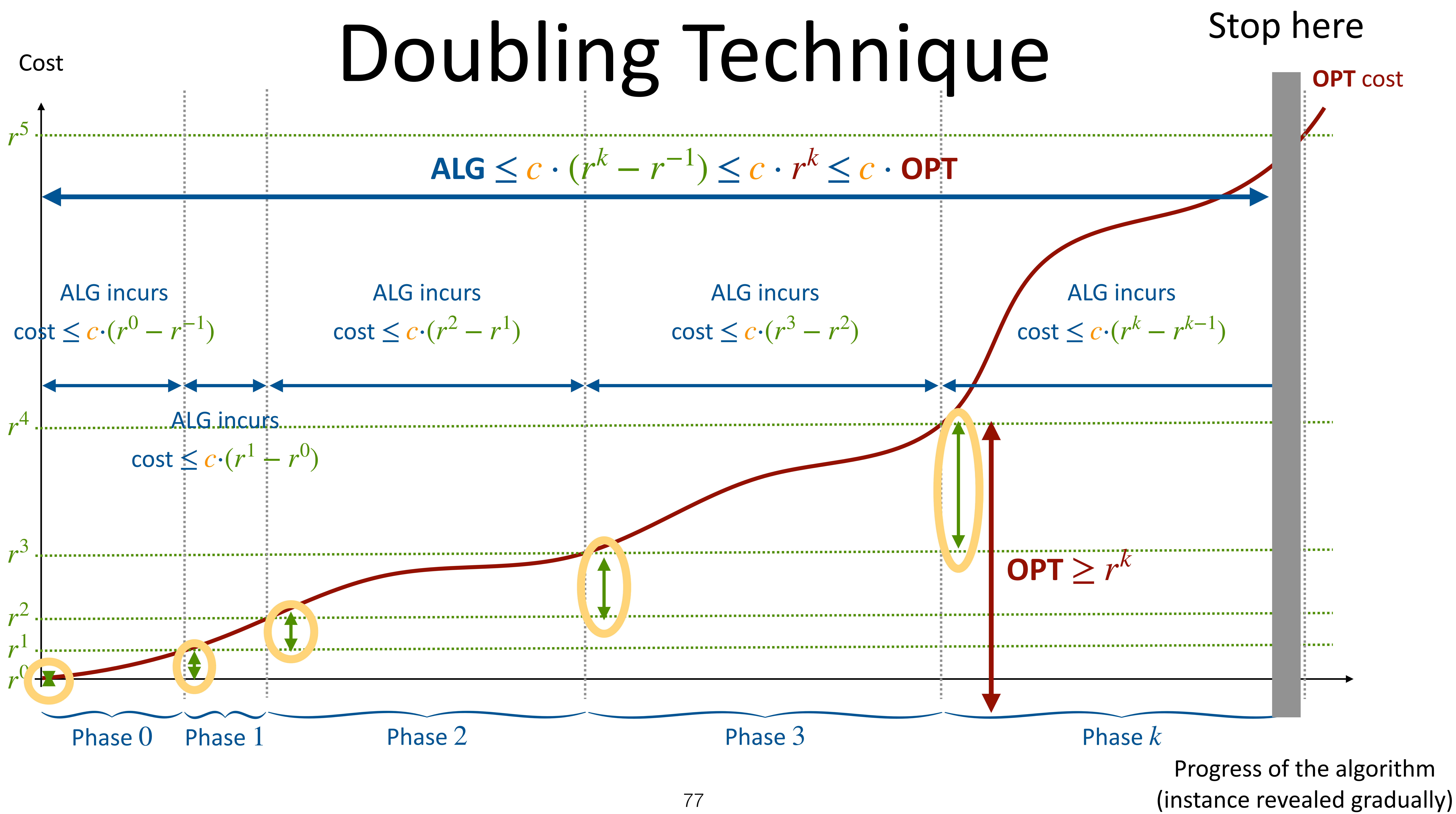
Doubling Technique



Doubling Technique



Doubling Technique



Doubling Technique

- For some problems, the optimal cost increases as the instance is revealed
  - We can design a doubling algorithm **DOUBLE** with a parameter r as follows:
    - We keep track of the value of the optimal cost OPT for the current instance
    - **DOUBLE** works in phases
      - The phases is decided by the value of OPT; the phase i starts at the time when OPT is at least r^i
      - If **DOUBLE** never creates (total) cost that exceeds $c \cdot (r^i - r^{i-1})$ in every phase i, **DOUBLE** is c-competitive

Doubling Technique

- For some problems, the optimal cost increases as the instance is revealed
  - We can design a doubling algorithm **DOUBLE** with a parameter r as follows:
    - We keep track of the value of the optimal cost OPT for the current instance
    - **DOUBLE** works in phases
      - The phases is decided by the value of OPT; the phase i starts at the time when OPT is at least r^i
      - If **DOUBLE** never creates (total) cost that exceeds $c \cdot (r^i - r^{i-1})$ in every phase i, **DOUBLE** is c-competitive

Your main effort when using doubling!

Doubling Technique

- For some problems, the optimal cost increases as the instance is revealed
  - We can design a doubling algorithm **DOUBLE** with a parameter r as follows:
    - We keep track of the value of the optimal cost OPT for the current instance
    - **DOUBLE** works in phases
      - The phases is decided by the value of OPT; the phase i starts at the time when OPT is at least r^i
      - If **DOUBLE** never creates (total) cost that exceeds $c \cdot (r^i - r^{i-1})$ in every phase i, **DOUBLE** is c-competitive
  - Using geometrically increasing estimation on the optimal solution, we can produce fragments of the algorithm's solution.

Doubling Technique

- For some problems, the optimal cost increases as the instance is revealed
  - We can design a doubling algorithm **DOUBLE** with a parameter r as follows:
    - We keep track of the value of the optimal cost OPT for the current instance
    - **DOUBLE** works in phases
      - The phases is decided by the value of OPT; the phase i starts at the time when OPT is at least r^i
      - If **DOUBLE** never creates (total) cost that exceeds $c \cdot (r^i - r^{i-1})$ in every phase i, **DOUBLE** is c-competitive
  - Using geometrically increasing estimation on the optimal solution, we can produce fragments of the algorithm's solution.
  - The term "doubling" is a bit misleading; sometimes we also use factors other than 2.

Doubling Technique — Bidding

- For some problems, the optimal cost increases as the instance is revealed
  - We can design a doubling algorithm **DOUBLE** with a parameter r as follows:
    - We keep track of the value of the optimal cost OPT for the current instance

    - **DOUBLE** works in phases
      - The phases is decided by the value of OPT; the phase i starts at the time when OPT is at least r^i

$r = 2$

      - If **DOUBLE** never creates (total) cost that exceeds $c \cdot (r^i - r^{i-1})$ in every phase i, **DOUBLE** is c-competitive

Doubling Technique — Bidding

- For some problems, the optimal cost increases as the instance is revealed
  - We can design a doubling algorithm **DOUBLE** with a parameter r as follows:
    - We keep track of the value of the optimal cost OPT for the current instance

OPT > 2

- **DOUBLE** works in phases
  - The phases is decided by the value of OPT; the phase i starts at the time when OPT is at least r^i

$r = 2$

  - If **DOUBLE** never creates (total) cost that exceeds $c \cdot (r^i - r^{i-1})$ in every phase i, **DOUBLE** is c-competitive

Doubling Technique — Bidding

- For some problems, the optimal cost increases as the instance is revealed
  - We can design a doubling algorithm **DOUBLE** with a parameter r as follows:
    - We keep track of the value of the optimal cost OPT for the current instance

-
  -
    - **DOUBLE** works in phases
      - The phases is decided by the value of OPT; the phase i starts at the time when OPT is at least r^i

$r = 2$

-
  -
    -
      - If **DOUBLE** never creates (total) cost that exceeds $c \cdot (r^i - r^{i-1})$ in every phase i, **DOUBLE** is c-competitive

Doubling Technique — Bidding

- For some problems, the optimal cost increases as the instance is revealed
  - We can design a doubling algorithm **DOUBLE** with a parameter r as follows:
    - We keep track of the value of the optimal cost OPT for the current instance

$\text{OPT} > 4$

- **DOUBLE** works in phases
  - The phases is decided by the value of OPT; the phase i starts at the time when OPT is at least r^i

$r = 2$

  - If **DOUBLE** never creates (total) cost that exceeds $c \cdot (r^i - r^{i-1})$ in every phase i, **DOUBLE** is c-competitive

Doubling Technique — Bidding

- For some problems, the optimal cost increases as the instance is revealed
  - We can design a doubling algorithm **DOUBLE** with a parameter r as follows:
    - We keep track of the value of the optimal cost OPT for the current instance

- **DOUBLE** works in phases
  - The phases is decided by the value of OPT; the phase i starts at the time when OPT is at least r^i

$r = 2$

  - If **DOUBLE** never creates (total) cost that exceeds $c \cdot (r^i - r^{i-1})$ in every phase i, **DOUBLE** is c-competitive

Doubling Technique — Bidding

- For some problems, the optimal cost increases as the instance is revealed
  - We can design a doubling algorithm **DOUBLE** with a parameter r as follows:
    - We keep track of the value of the optimal cost OPT for the current instance

OPT > 8

- **DOUBLE** works in phases
  - The phases is decided by the value of OPT; the phase i starts at the time when OPT is at least r^i

$r = 2$

  - If **DOUBLE** never creates (total) cost that exceeds $c \cdot (r^i - r^{i-1})$ in every phase i, **DOUBLE** is c-competitive

Doubling Technique — Bidding

- For some problems, the optimal cost increases as the instance is revealed
  - We can design a doubling algorithm **DOUBLE** with a parameter r as follows:
    - We keep track of the value of the optimal cost OPT for the current instance

    - **DOUBLE** works in phases
      - The phases is decided by the value of OPT; the phase i starts at the time when OPT is at least r^i

$r = 2$

      - If **DOUBLE** never creates (total) cost that exceeds $c \cdot (r^i - r^{i-1})$ in every phase i, **DOUBLE** is c-competitive

Doubling Technique — Bidding

- For some problems, the optimal cost increases as the instance is revealed
  - We can design a doubling algorithm **DOUBLE** with a parameter r as follows:
    - We keep track of the value of the optimal cost OPT for the current instance

$\text{OPT} > 16$

- **DOUBLE** works in phases
  - The phases is decided by the value of OPT; the phase i starts at the time when OPT is at least r^i

$r = 2$

  - If **DOUBLE** never creates (total) cost that exceeds $c \cdot (r^i - r^{i-1})$ in every phase i, **DOUBLE** is c-competitive

Doubling Technique — Bidding

- For some problems, the optimal cost increases as the instance is revealed
  - We can design a doubling algorithm **DOUBLE** with a parameter r as follows:
    - We keep track of the value of the optimal cost OPT for the current instance

Phase i: (**OPT** $\geq 2^i$) submit bid 2^{i+1} (if fail, **OPT** $\geq 2^{i+1}$ and phase $i+1$ starts)

    - **DOUBLE** works in phases
      - The phases is decided by the value of OPT; the phase i starts at the time when OPT is at least r^i

$r = 2$

      - If **DOUBLE** never creates (total) cost that exceeds $c \cdot (r^i - r^{i-1})$ in every phase i, **DOUBLE** is c-competitive

Doubling Technique — Bidding

- For some problems, the optimal cost increases as the instance is revealed
  - We can design a doubling algorithm **DOUBLE** with a parameter r as follows:
    - We keep track of the value of the optimal cost OPT for the current instance

Phase i: (**OPT** $\geq 2^i$) submit bid 2^{i+1} (if fail, **OPT** $\geq 2^{i+1}$ and phase $i+1$ starts)

- **DOUBLE** works in phases
  - The phases is decided by the value of OPT; the phase i starts at the time when OPT is at least r^i
    - **For the Doubling online bidding algorithm, each phase i consists of one bid, which costs 2^{i+1}**

$r = 2$

- If **DOUBLE** never creates (total) cost that exceeds $c \cdot (r^i - r^{i-1})$ in every phase i, **DOUBLE** is c-competitive

Doubling Technique — Bidding

- For some problems, the optimal cost increases as the instance is revealed
  - We can design a doubling algorithm **DOUBLE** with a parameter r as follows:
    - We keep track of the value of the optimal cost OPT for the current instance

Phase i: (**OPT** $\geq 2^i$) submit bid 2^{i+1} (if fail, **OPT** $\geq 2^{i+1}$ and phase $i+1$ starts)

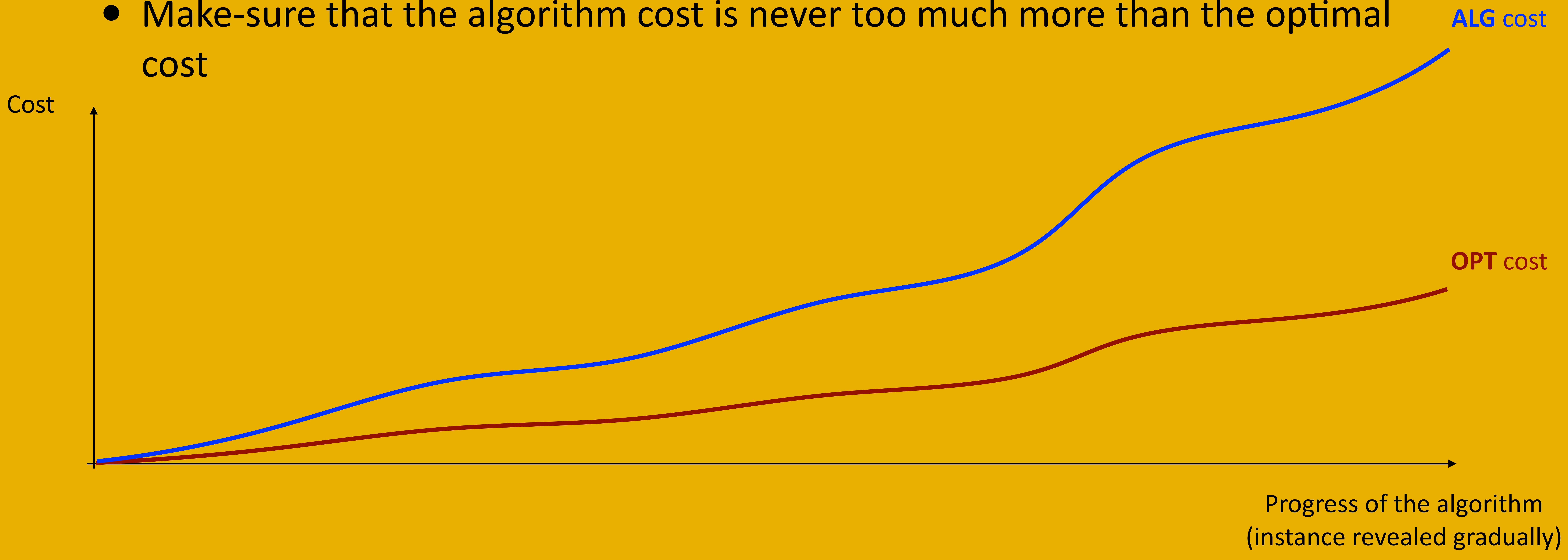
- **DOUBLE** works in phases
  - The phases is decided by the value of OPT; the phase i starts at the time when OPT is at least r^i
    - **For the Doubling online bidding algorithm, each phase i consists of one bid, which costs 2^{i+1}**

$r = 2$

  - If **DOUBLE** never creates (total) cost that exceeds $c \cdot (r^i - r^{i-1})$ in every phase i, **DOUBLE** is c-competitive
    - **The DOUBLE algorithm incurs cost 2^{i+1} in phase i, and $2^{i+1} \leq 4 \cdot (2^i - 2^{i-1})$**

What Happened

- The idea of **doubling technique** for designing online algorithms:
  - Keep track of the optimal solution throughout the process
  - Make-sure that the algorithm cost is never too much more than the optimal cost

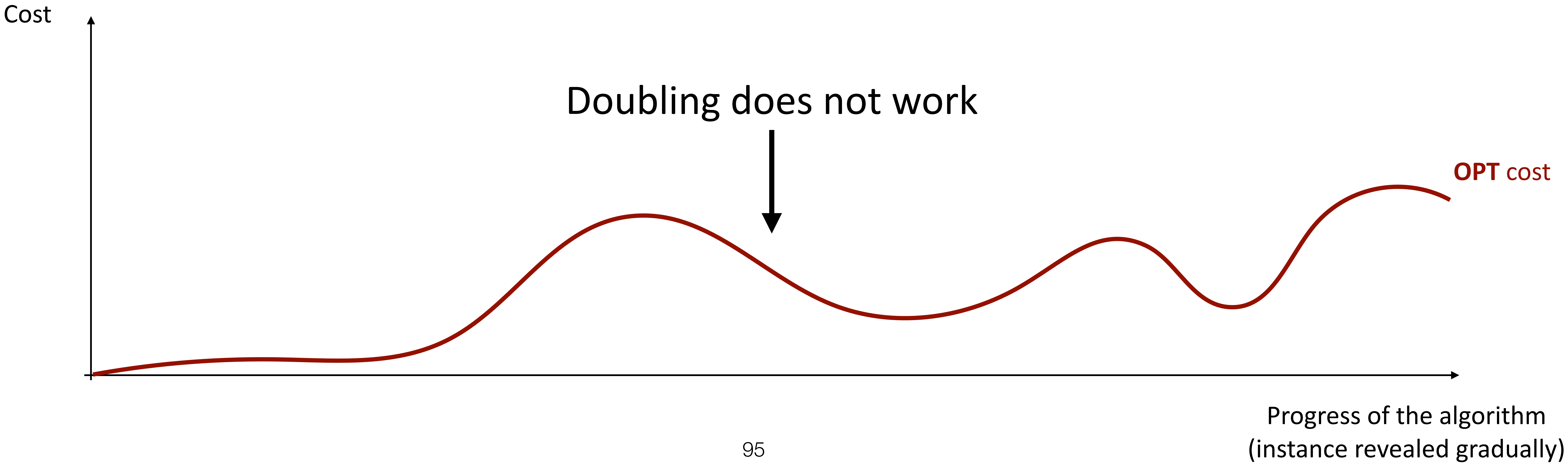


Problems that Doubling Does not Help

- Doubling algorithm helps only when the optimal solution cost is increasing with the releasing of events

Problems that Doubling Does not Help

- Doubling algorithm helps only when the optimal solution cost is increasing with the releasing of events



Problems that Doubling Does not Help

- Doubling algorithm helps only when the optimal solution cost is increasing with the releasing of events
  - Online minimum dominating set

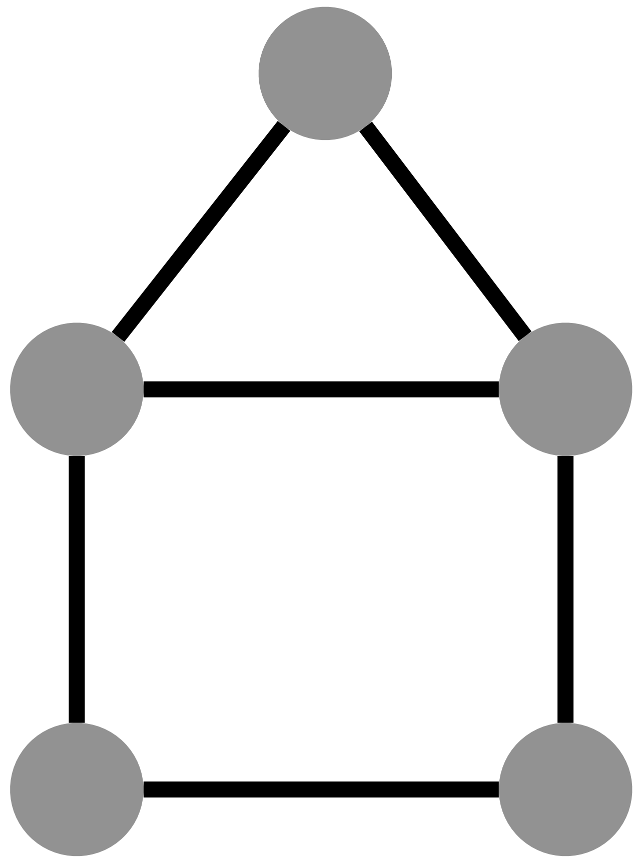
Problems that Doubling Does not Help

- Doubling algorithm helps only when the optimal solution cost is increasing with the releasing of events
  - Online minimum dominating set

Dominating set:
A subset D of vertices such that
Every vertex which is not in D has at least one neighbor in D

Problems that Doubling Does not Help

- Doubling algorithm helps only when the optimal solution cost is increasing with the releasing of events
    - Online minimum dominating set



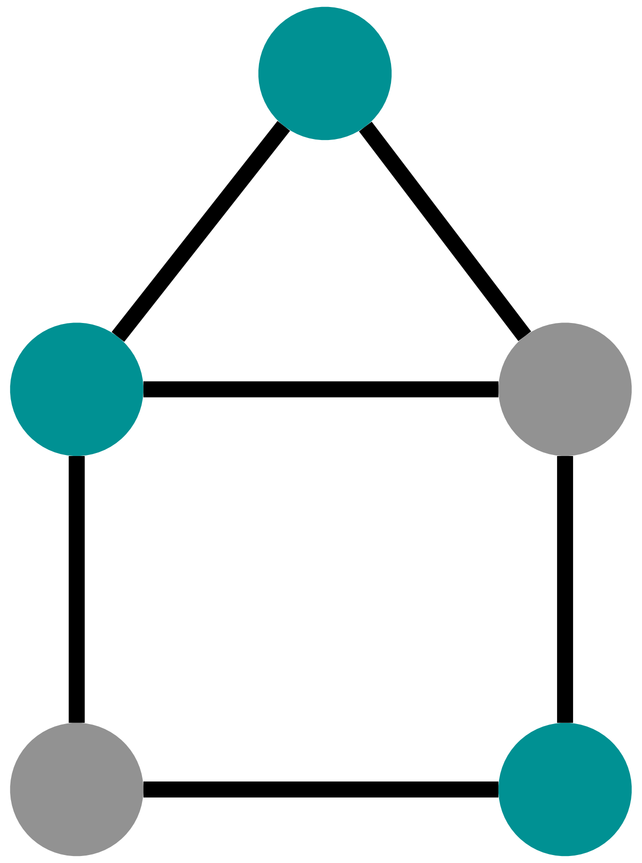
Dominating set:

A subset D of vertices such that

Every vertex which is not in D has at least one neighbor in D

Problems that Doubling Does not Help

- Doubling algorithm helps only when the optimal solution cost is increasing with the releasing of events
  - Online minimum dominating set



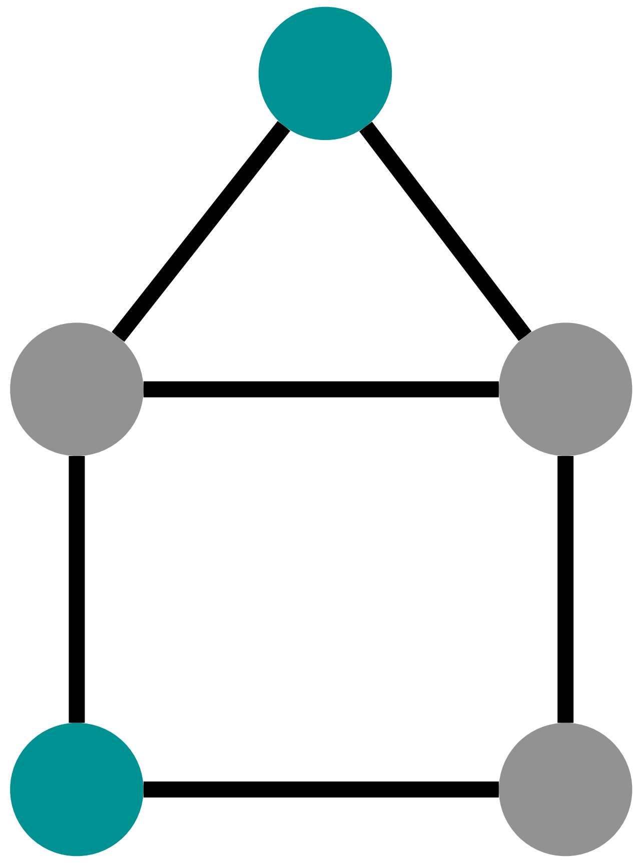
Dominating set:

A subset D of vertices such that

Every vertex which is not in D has at least one neighbor in D

Problems that Doubling Does not Help

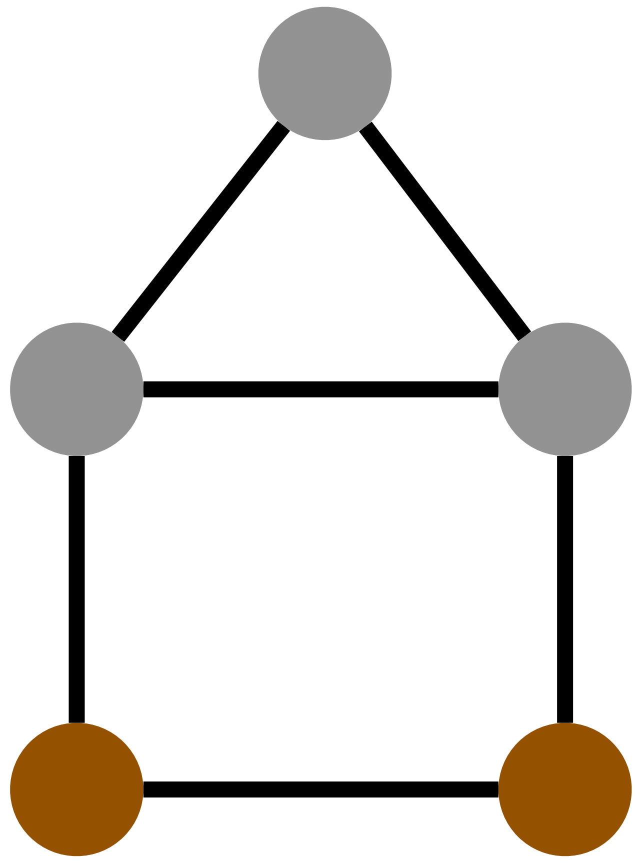
- Doubling algorithm helps only when the optimal solution cost is increasing with the releasing of events
  - Online minimum dominating set



Dominating set:
A subset D of vertices such that
Every vertex which is not in D has at least one neighbor in D

Problems that Doubling Does not Help

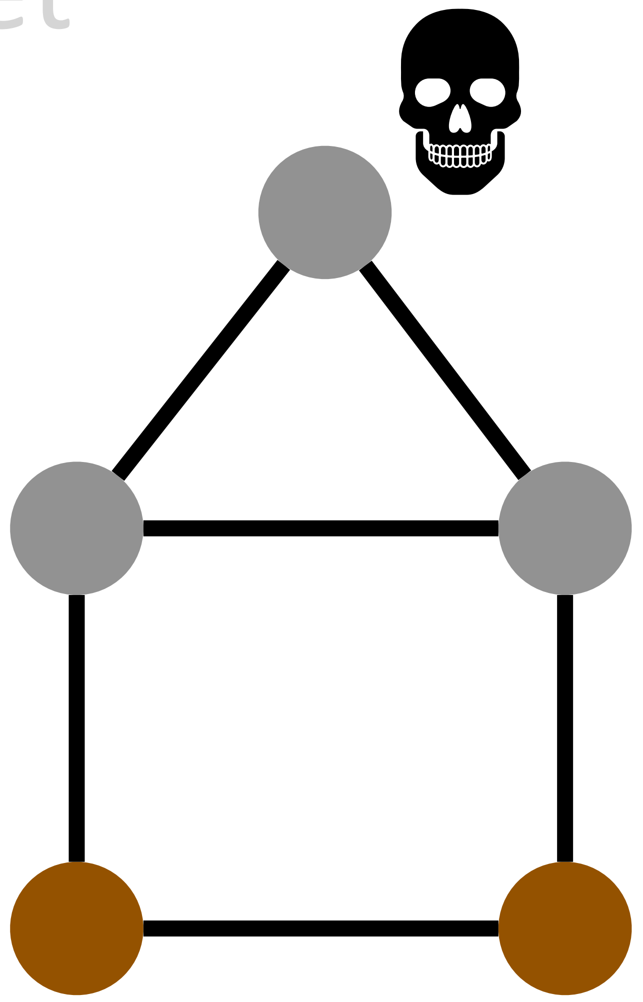
- Doubling algorithm helps only when the optimal solution cost is increasing with the releasing of events
    - Online minimum dominating set



Dominating set:
A subset D of vertices such that
Every vertex which is not in D has at least one neighbor in D

Problems that Doubling Does not Help

- Doubling algorithm helps only when the optimal solution cost is increasing with the releasing of events
    - Online minimum dominating set



Dominating set:

A subset D of vertices such that

Every vertex which is not in D has at least one neighbor in D

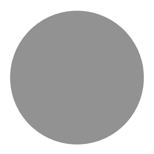
Problems that Doubling Does not Help

- Doubling algorithm helps only when the optimal solution cost is increasing with the releasing of events
  - Online minimum dominating set: The vertex arrives one-by-one. When a vertex arrives, all edges between this vertex and the released vertices are also known. A vertex can only be chosen to D upon arrival.

Dominating set:
A subset D of vertices such that
Every vertex which is not in D has at least one neighbor in D

Problems that Doubling Does not Help

- Doubling algorithm helps only when the optimal solution cost is increasing with the releasing of events
  - Online minimum dominating set: The vertex arrives one-by-one. When a vertex arrives, all edges between this vertex and the released vertices are also known. A vertex can only be chosen to D upon arrival.



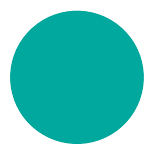
Dominating set:

A subset D of vertices such that

Every vertex which is not in D has at least one neighbor in D

Problems that Doubling Does not Help

- Doubling algorithm helps only when the optimal solution cost is increasing with the releasing of events
  - Online minimum dominating set: The vertex arrives one-by-one. When a vertex arrives, all edges between this vertex and the released vertices are also known. A vertex can only be chosen to D upon arrival.



OPT = 1

Dominating set:
A subset D of vertices such that
Every vertex which is not in D has at least one neighbor in D

Problems that Doubling Does not Help

- Doubling algorithm helps only when the optimal solution cost is increasing with the releasing of events
  - Online minimum dominating set: The vertex arrives one-by-one. When a vertex arrives, all edges between this vertex and the released vertices are also known. A vertex can only be chosen to D upon arrival.

Dominating set:
A subset D of vertices such that
Every vertex which is not in D has at least one neighbor in D

Problems that Doubling Does not Help

- Doubling algorithm helps only when the optimal solution cost is increasing with the releasing of events
  - Online minimum dominating set: The vertex arrives one-by-one. When a vertex arrives, all edges between this vertex and the released vertices are also known. A vertex can only be chosen to D upon arrival.

OPT = 1

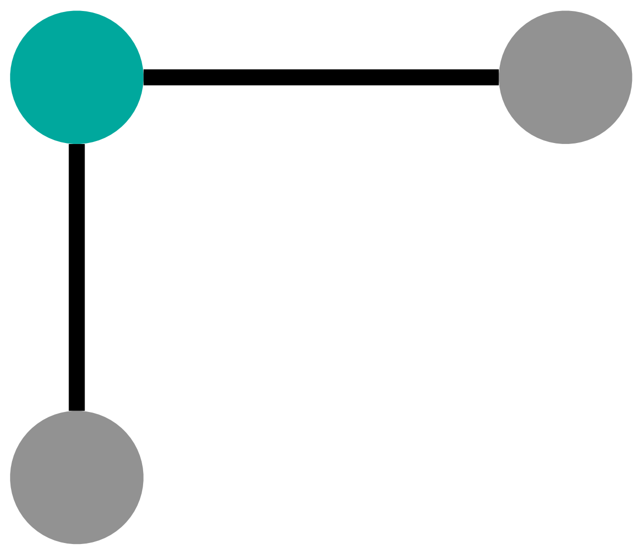
Dominating set:

A subset D of vertices such that

Every vertex which is not in D has at least one neighbor in D

Problems that Doubling Does not Help

- Doubling algorithm helps only when the optimal solution cost is increasing with the releasing of events
  - Online minimum dominating set: The vertex arrives one-by-one. When a vertex arrives, all edges between this vertex and the released vertices are also known. A vertex can only be chosen to D upon arrival.



Dominating set:

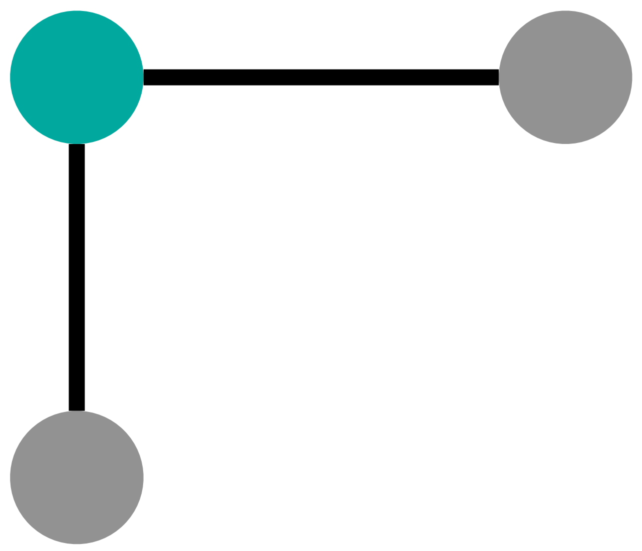
A subset D of vertices such that

Every vertex which is not in D has at least one neighbor in D

Problems that Doubling Does not Help

- Doubling algorithm helps only when the optimal solution cost is increasing with the releasing of events
  - Online minimum dominating set: The vertex arrives one-by-one. When a vertex arrives, all edges between this vertex and the released vertices are also known. A vertex can only be chosen to D upon arrival.

OPT = 1



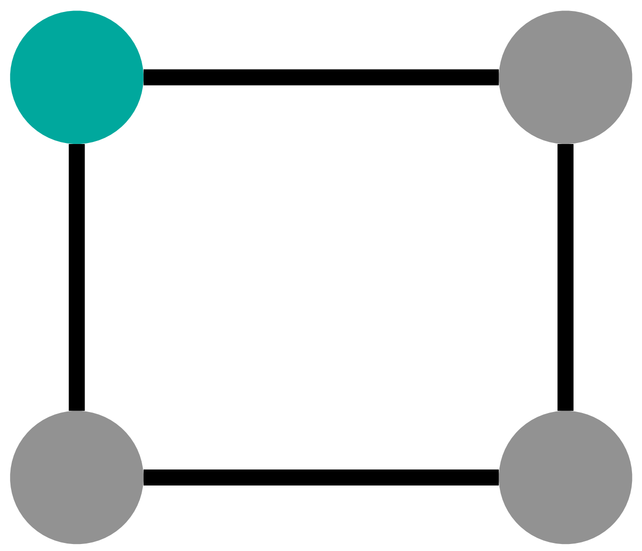
Dominating set:

A subset D of vertices such that

Every vertex which is not in D has at least one neighbor in D

Problems that Doubling Does not Help

- Doubling algorithm helps only when the optimal solution cost is increasing with the releasing of events
  - Online minimum dominating set: The vertex arrives one-by-one. When a vertex arrives, all edges between this vertex and the released vertices are also known. A vertex can only be chosen to D upon arrival.

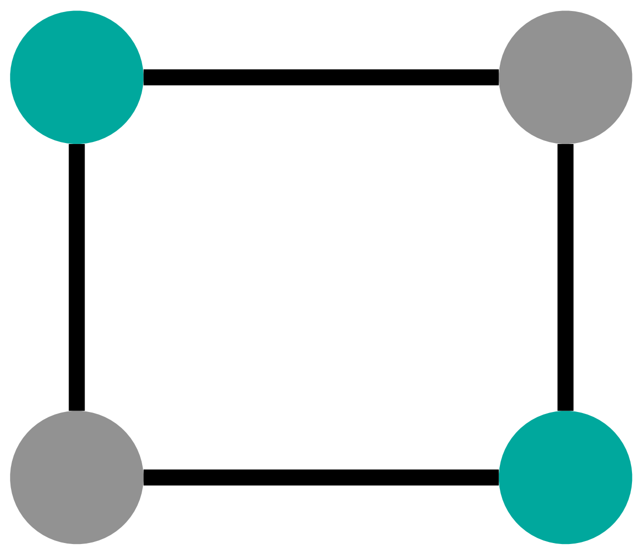


Dominating set:
A subset D of vertices such that
Every vertex which is not in D has at least one neighbor in D

Problems that Doubling Does not Help

- Doubling algorithm helps only when the optimal solution cost is increasing with the releasing of events
  - Online minimum dominating set: The vertex arrives one-by-one. When a vertex arrives, all edges between this vertex and the released vertices are also known. A vertex can only be chosen to D upon arrival.

OPT = 2



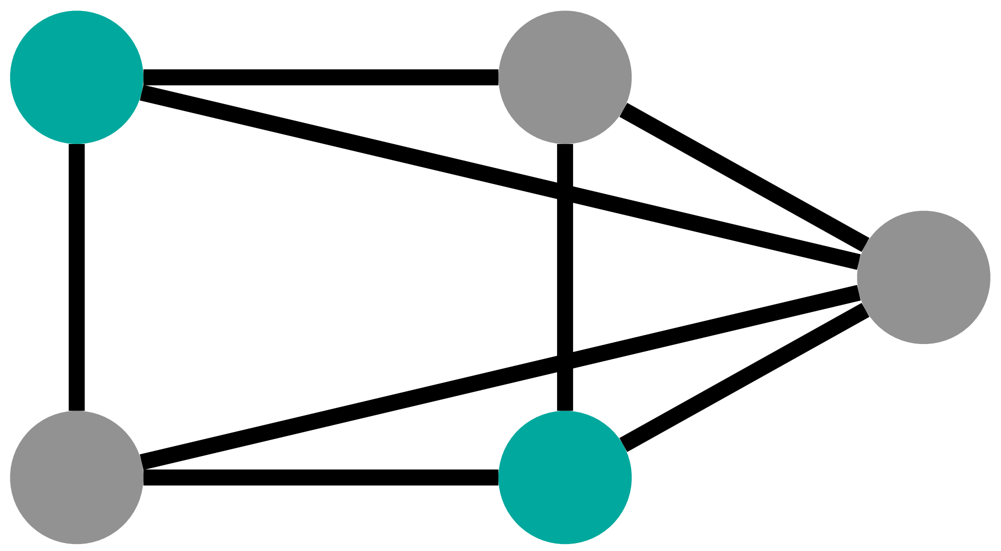
Dominating set:

A subset D of vertices such that

Every vertex which is not in D has at least one neighbor in D

Problems that Doubling Does not Help

- Doubling algorithm helps only when the optimal solution cost is increasing with the releasing of events
  - Online minimum dominating set: The vertex arrives one-by-one. When a vertex arrives, all edges between this vertex and the released vertices are also known. A vertex can only be chosen to D upon arrival.

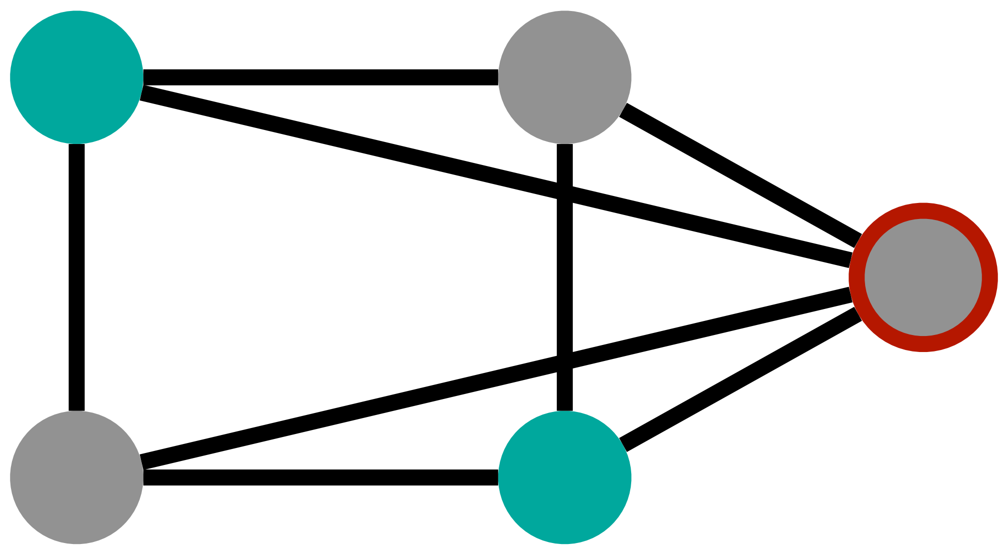


Dominating set:
A subset D of vertices such that
Every vertex which is not in D has at least one neighbor in D

Problems that Doubling Does not Help

- Doubling algorithm helps only when the optimal solution cost is increasing with the releasing of events
  - Online minimum dominating set: The vertex arrives one-by-one. When a vertex arrives, all edges between this vertex and the released vertices are also known. A vertex can only be chosen to D upon arrival.

OPT = 1



Dominating set:

A subset D of vertices such that

Every vertex which is not in D has at least one neighbor in D

What Happened

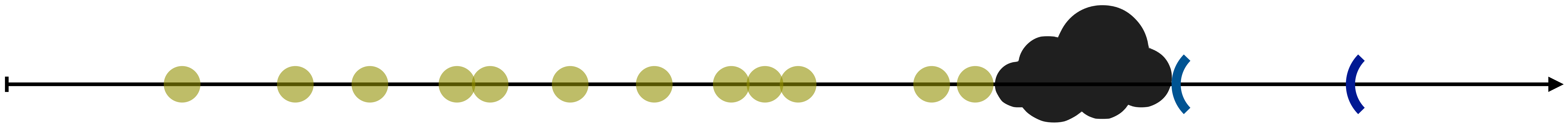
- Doubling helps only when the optimal cost is non-decreasing when more inputs are revealed

Outline

- Recap from the last lecture
- **Online bidding**
  - A 4-competitive algorithm
- A general technique for designing online algorithms: **doubling**
- "Best" online algorithms

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

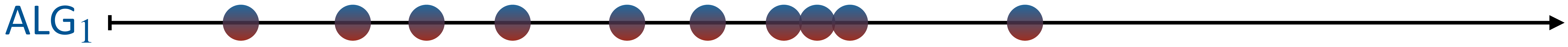
- Formally, we prove that for any online algorithm ALG_i, there exists an instance I_i such that

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

Problem Competitive Ratio Lower Bound

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

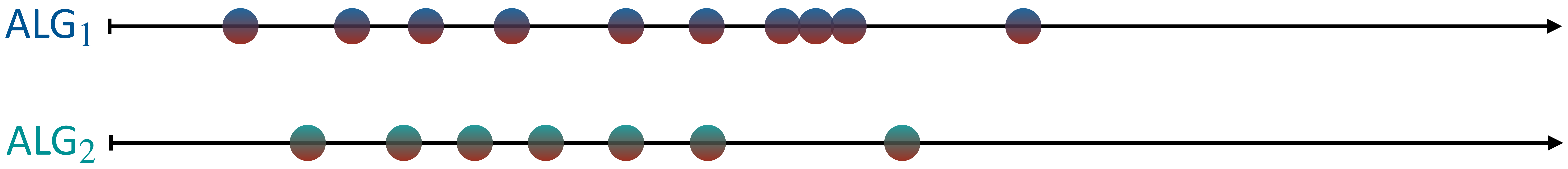
$$\frac{\text{ALG}_i(I_i)}{\text{OPT}(I_i)} \geq L$$



Problem Competitive Ratio Lower Bound

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

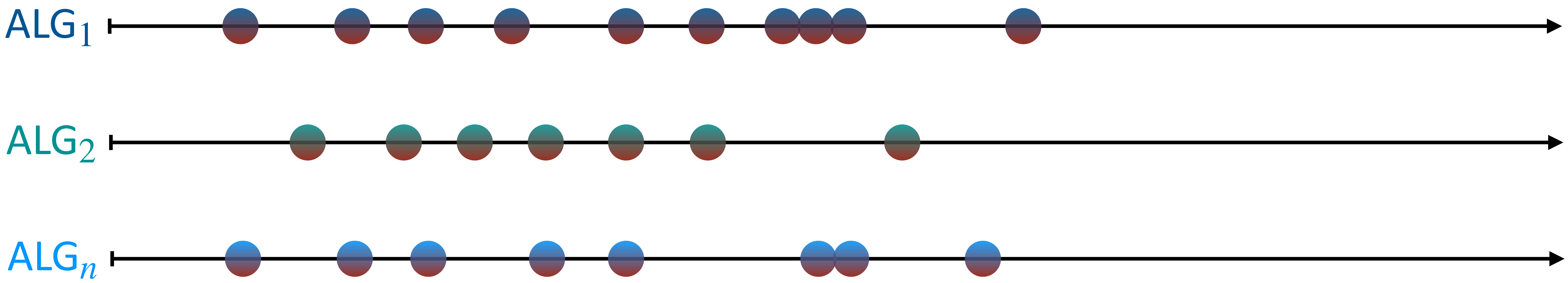
$$\frac{\text{ALG}_i(I_i)}{\text{OPT}(I_i)} \geq L$$



Problem Competitive Ratio Lower Bound

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

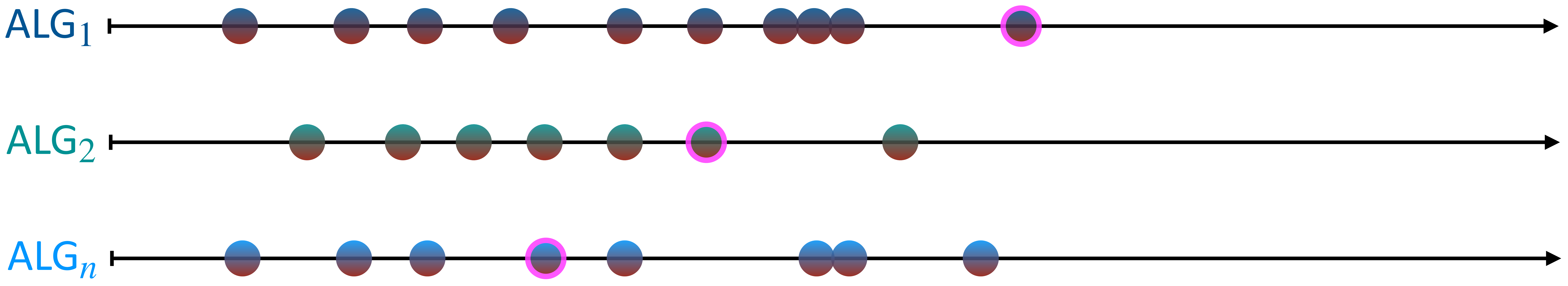
$$\frac{\text{ALG}_i(I_i)}{\text{OPT}(I_i)} \geq L$$



Problem Competitive Ratio Lower Bound

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

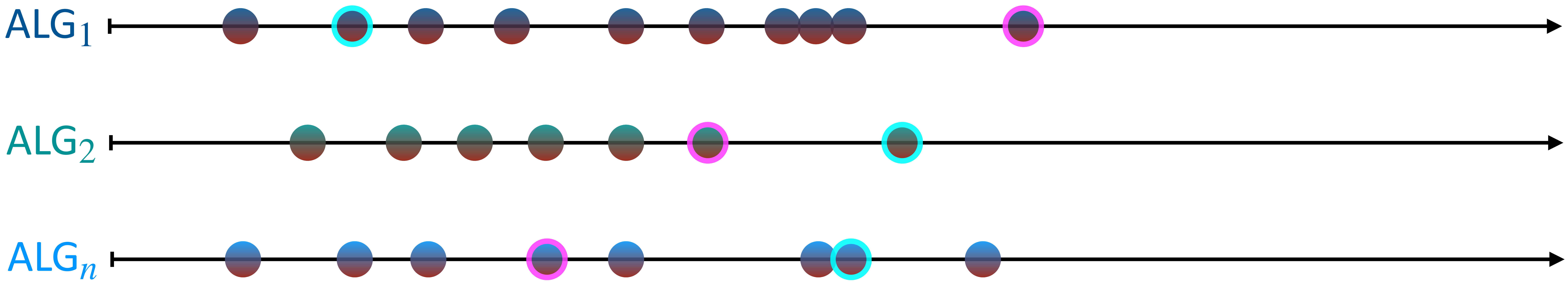
$$\frac{\mathsf{ALG}_i(I_i)}{\mathsf{OPT}(I_i)} \geq L$$



Problem Competitive Ratio Lower Bound

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

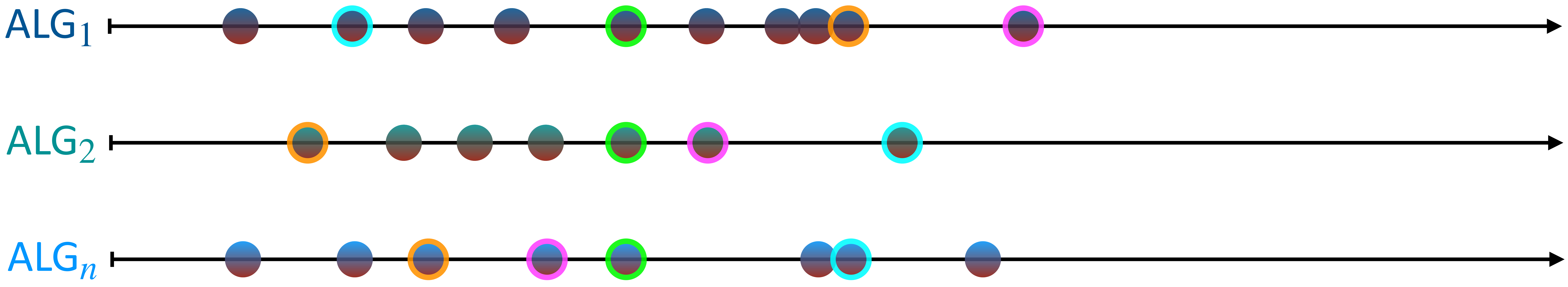
$$\frac{\text{ALG}_i(I_i)}{\text{OPT}(I_i)} \geq L$$



Problem Competitive Ratio Lower Bound

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

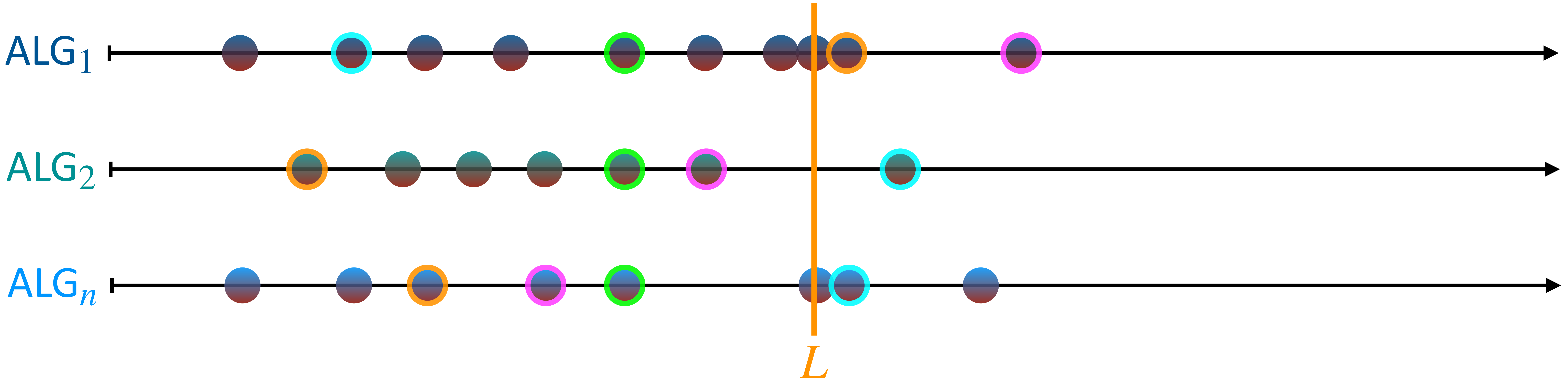
$$\frac{\text{ALG}_i(I_i)}{\text{OPT}(I_i)} \geq L$$



Problem Competitive Ratio Lower Bound

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

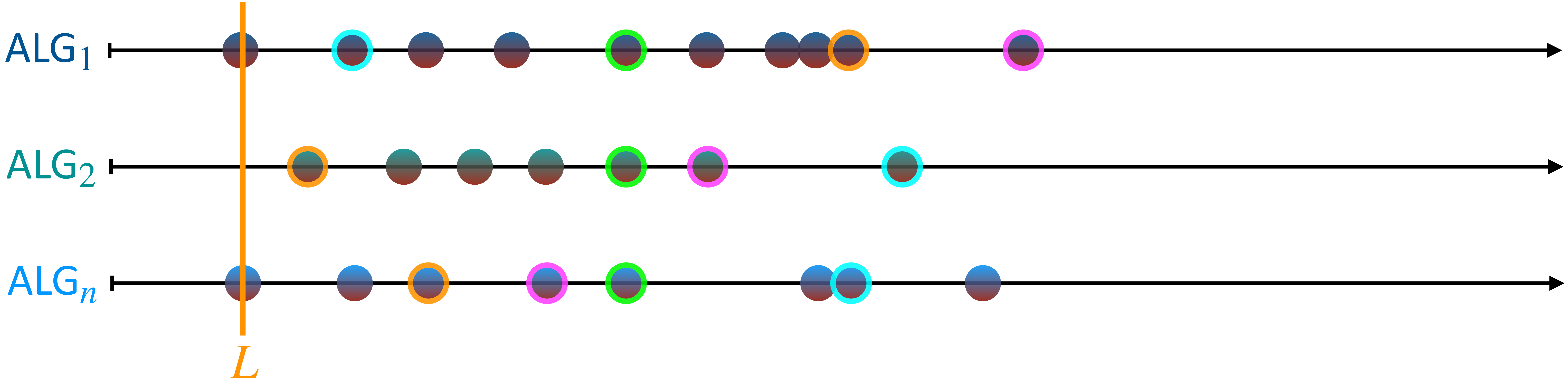
$$\frac{\text{ALG}_i(I_i)}{\text{OPT}(I_i)} \geq L$$



Problem Competitive Ratio Lower Bound

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

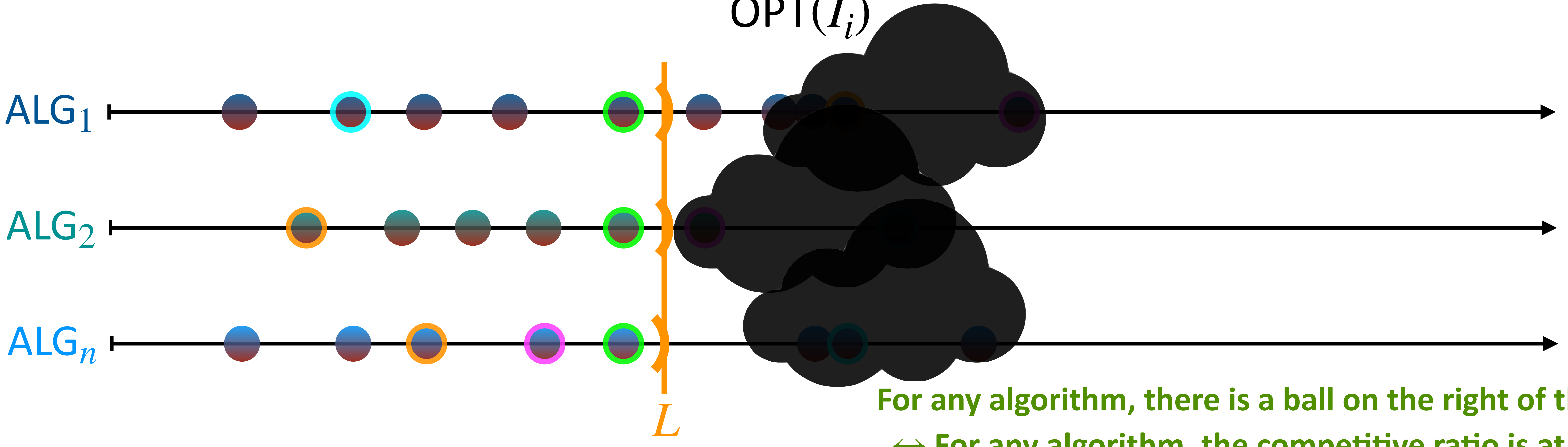
$$\frac{\mathsf{ALG}_i(I_i)}{\mathsf{OPT}(I_i)} \geq L$$



Problem Competitive Ratio Lower Bound

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

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

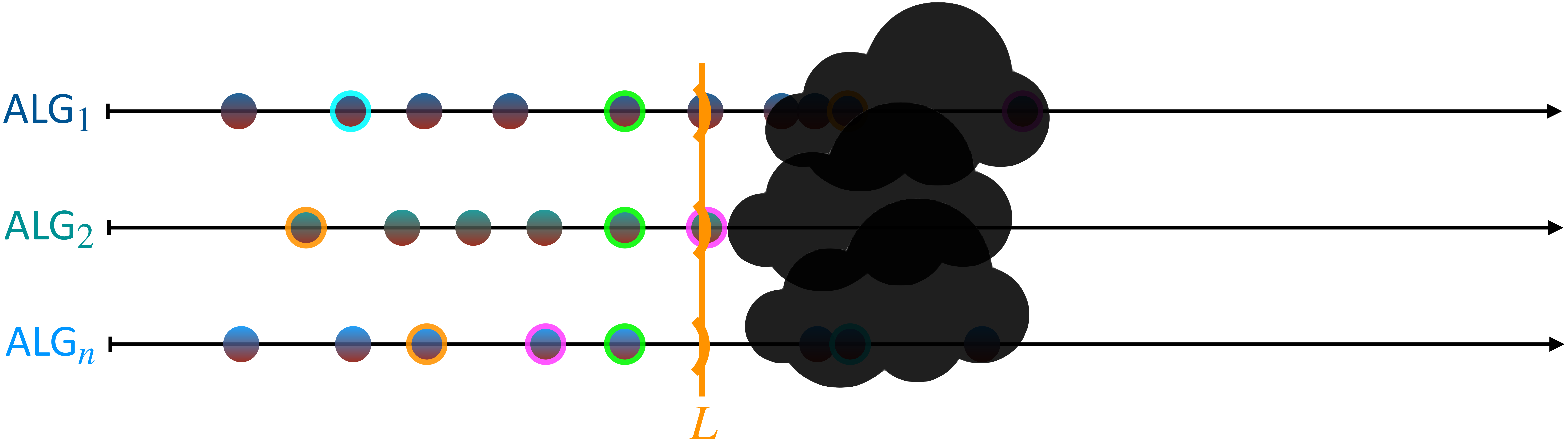


For any algorithm, there is a ball on the right of the bar L

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

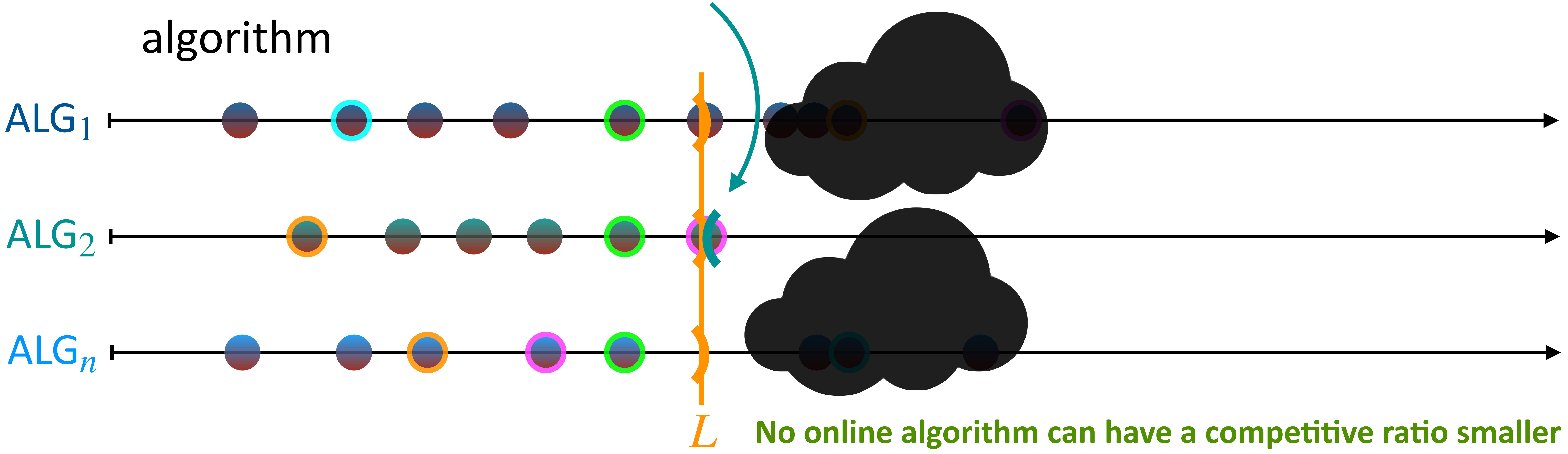
Problem Competitive Ratio Lower Bound

- 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

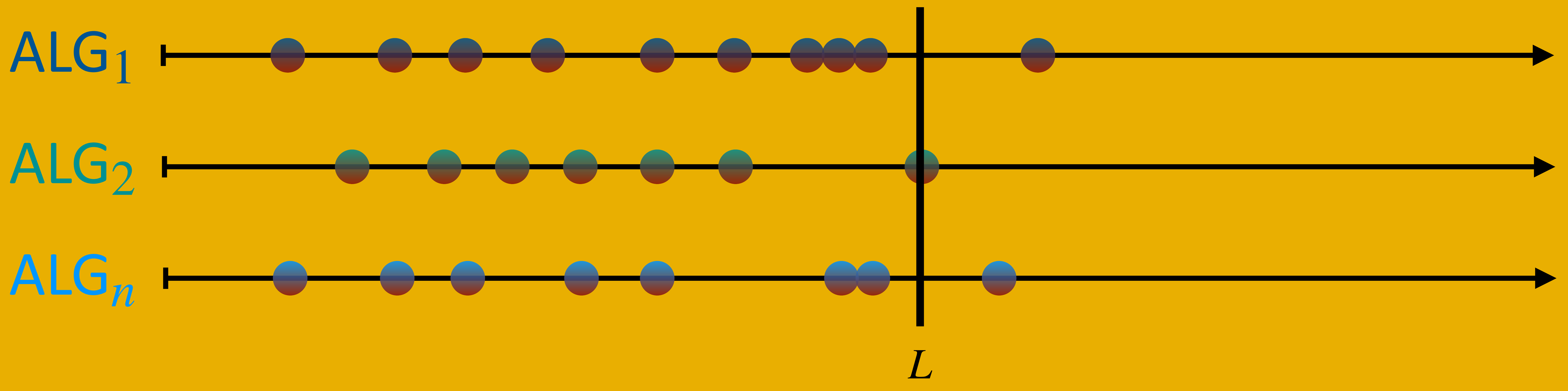


No online algorithm can have a competitive ratio smaller than L

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



Problem Competitive Ratio Lower Bound

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

Problem Competitive Ratio 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.

Problem Competitive Ratio 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.

Problem Competitive Ratio 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.

Problem Competitive Ratio 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

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

Problem Competitive Ratio Lower Bound

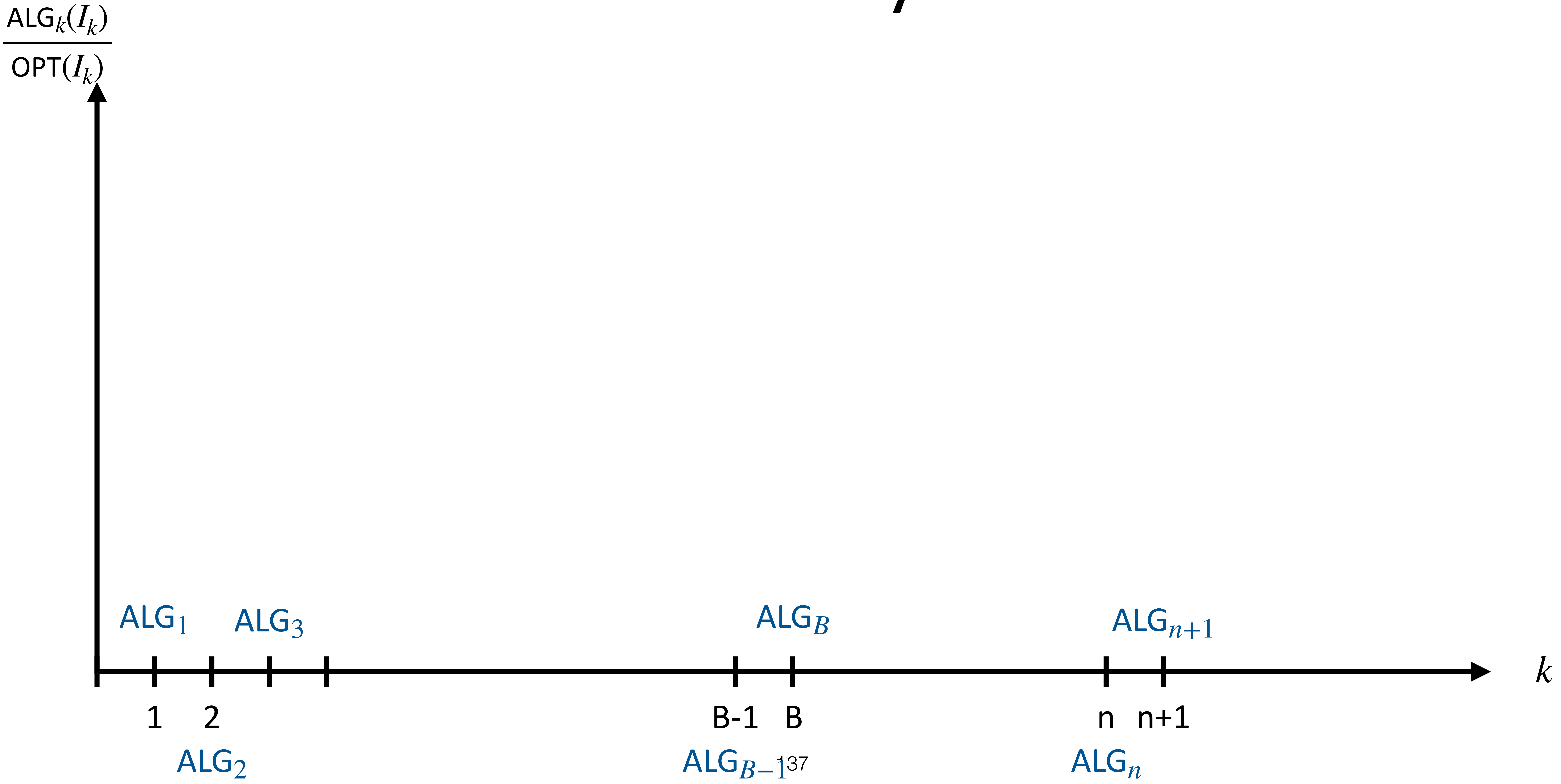
- Theorem: For the Buy-or-Rent problem, there is no deterministic online algorithm better than $(2 - \frac{1}{B})$-competitive.

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

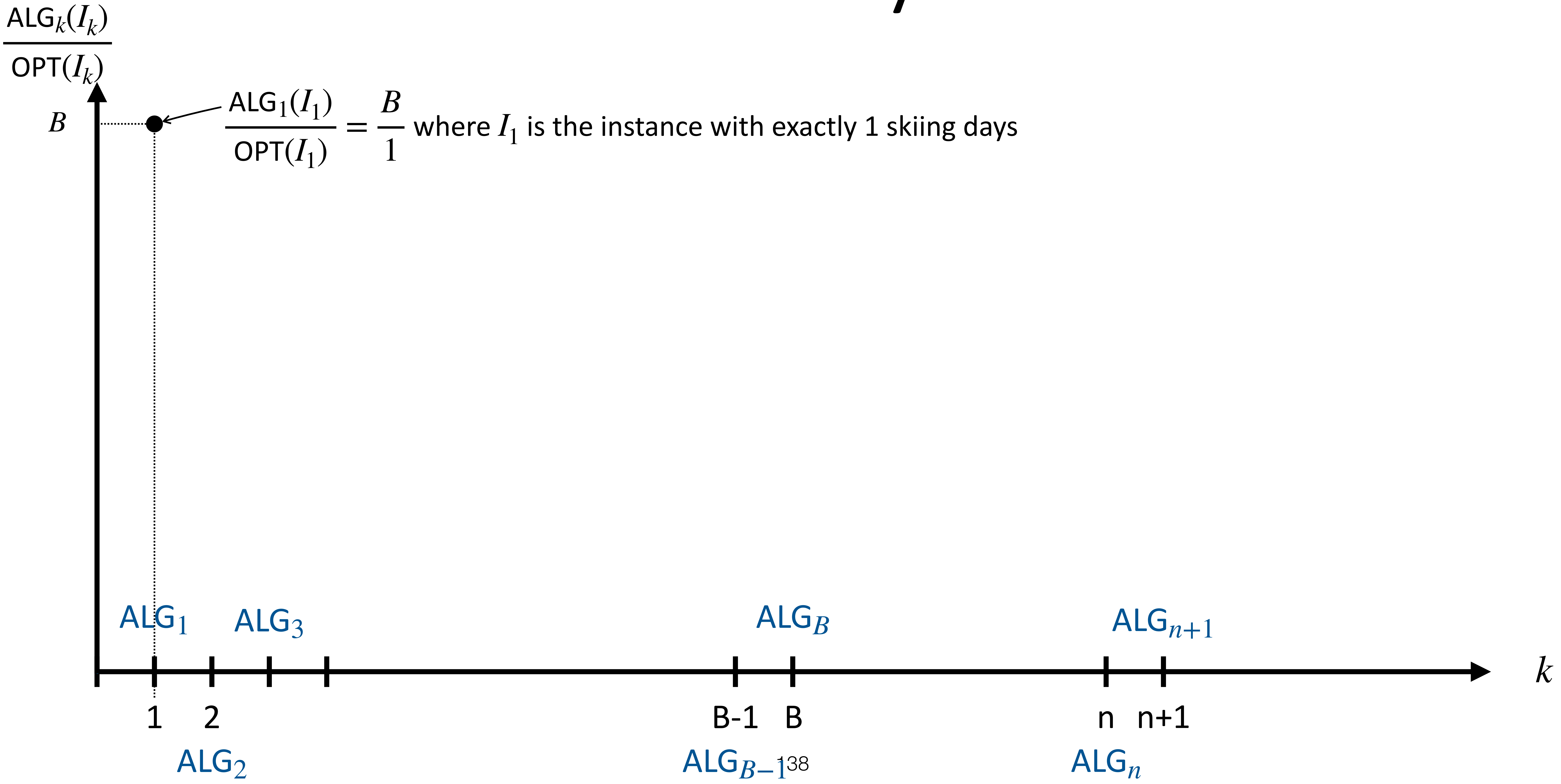
  Hence, the ratio is lower bounded by $\dfrac{(B-1)+B}{B}$ since $k < B$

Lower Bound for the Buy-or-Rent Problem

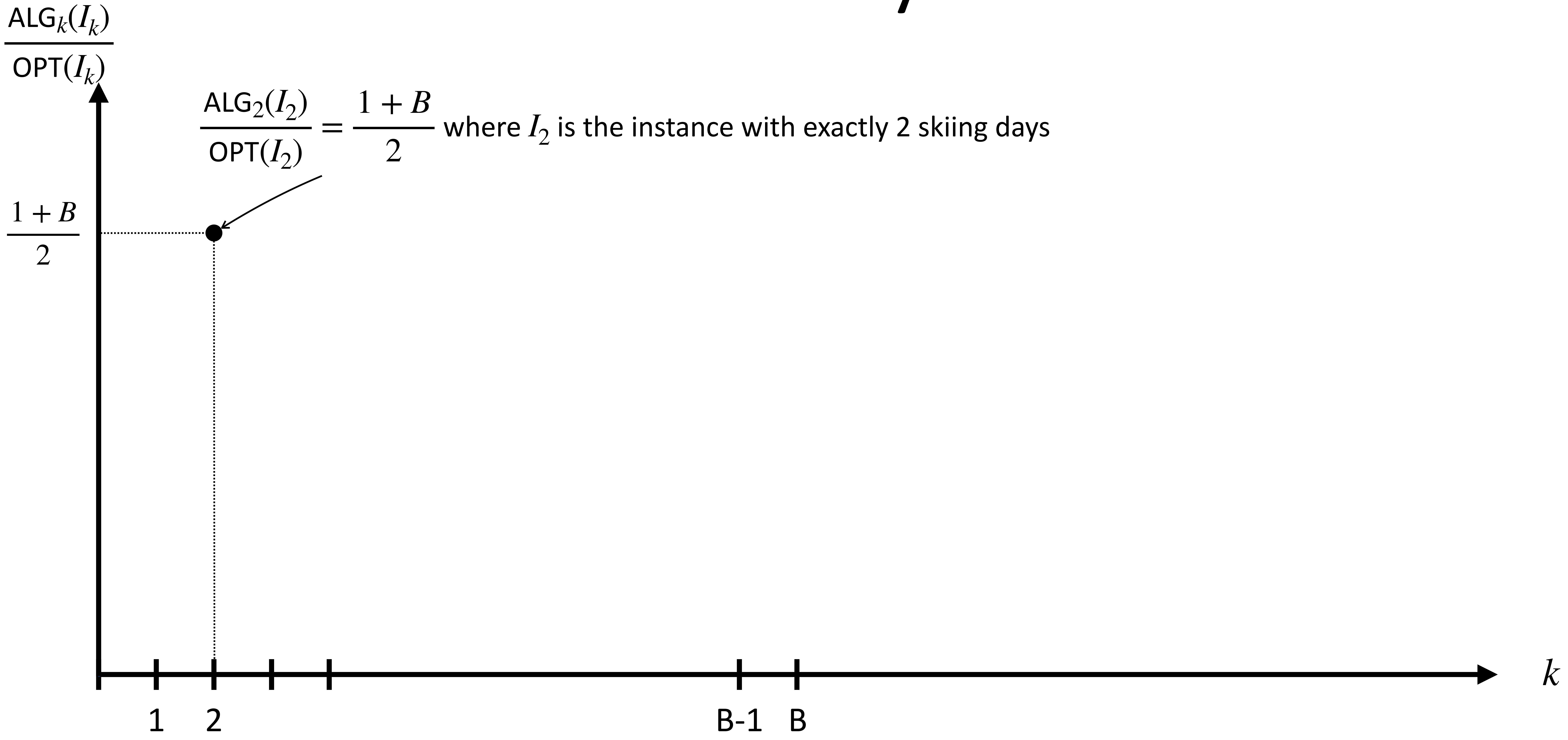


Lower Bound for the Buy-or-Rent Problem

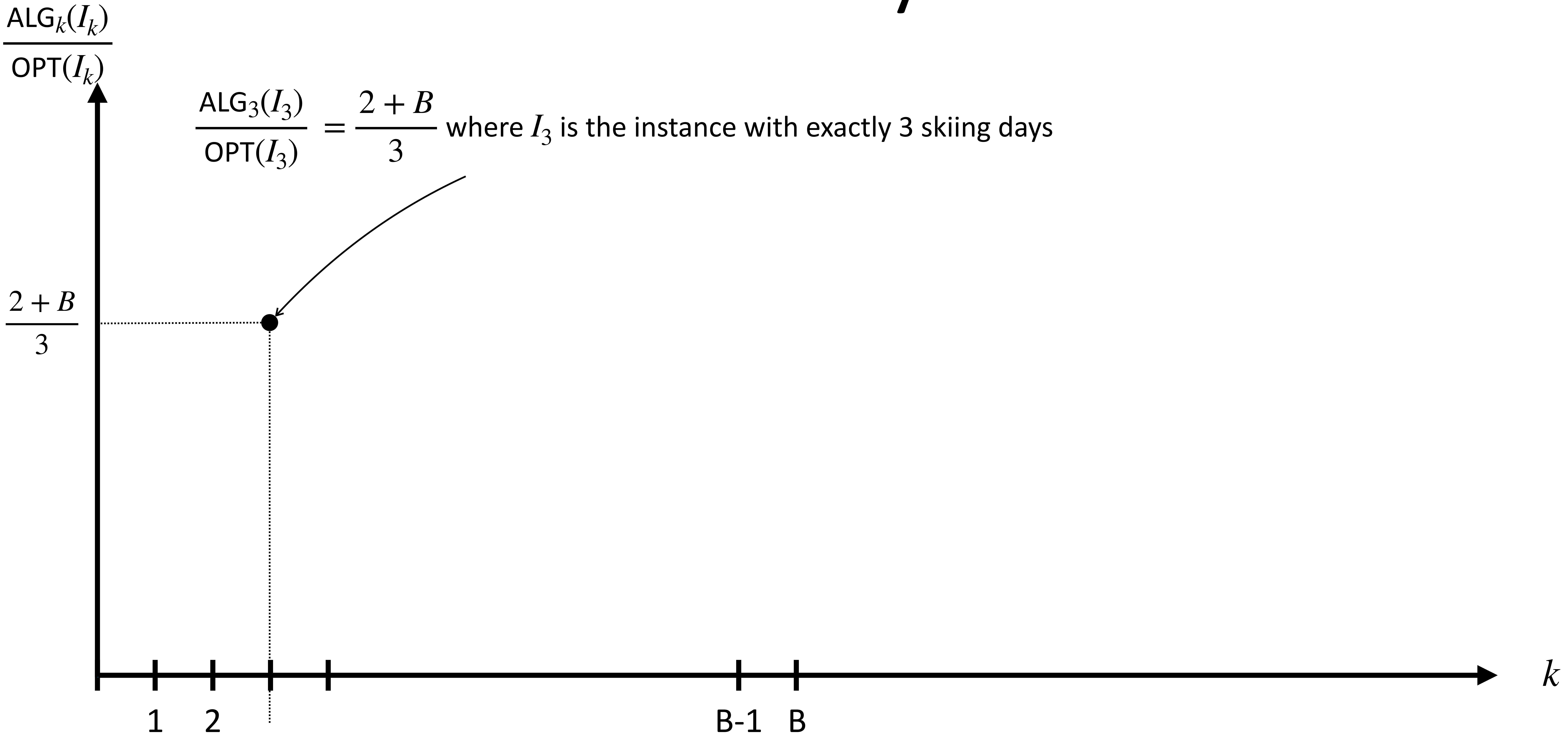
$$\frac{\text{ALG}_1(I_1)}{\text{OPT}(I_1)} = \frac{B}{1}$$ where I_1 is the instance with exactly 1 skiing days



Lower Bound for the Buy-or-Rent Problem

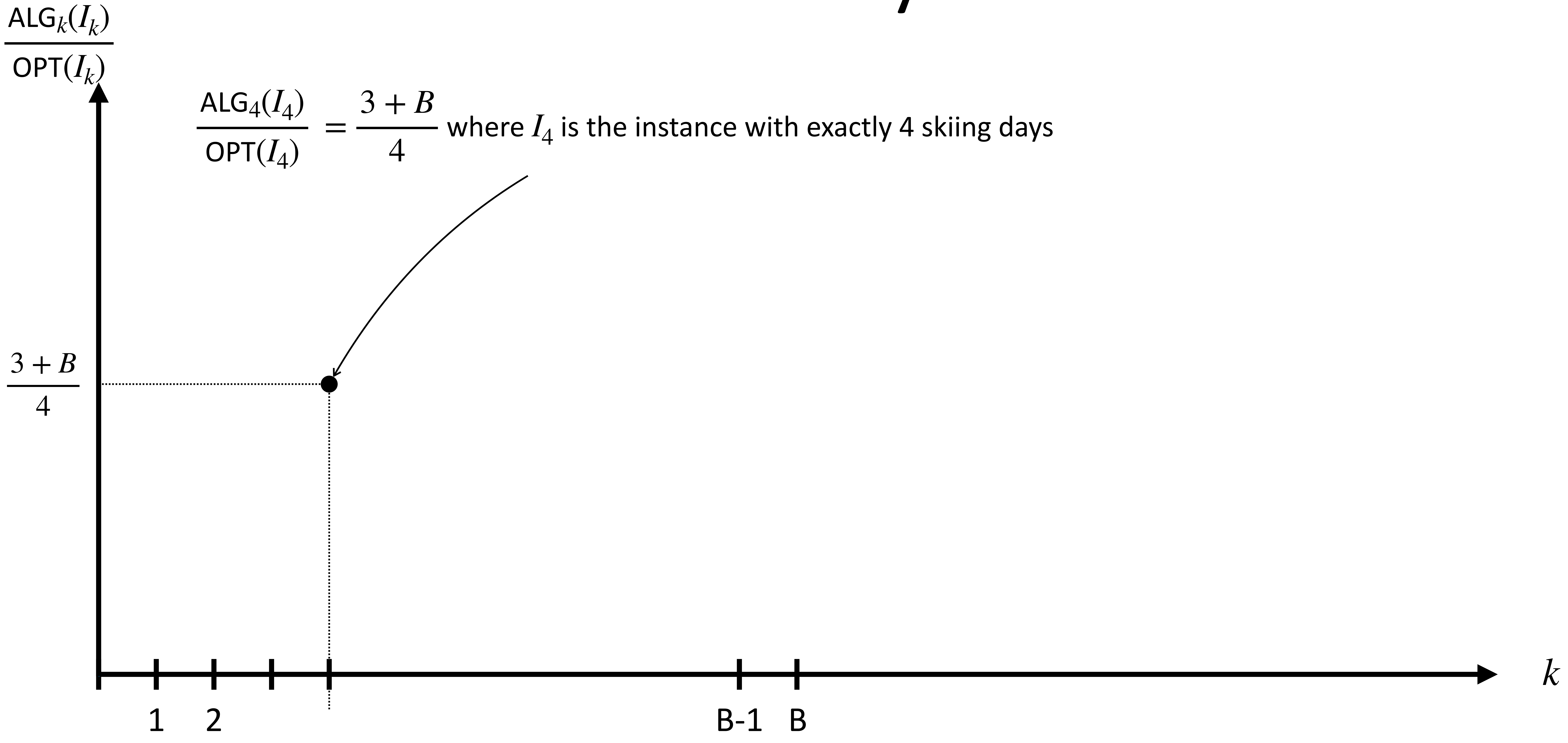


Lower Bound for the Buy-or-Rent Problem

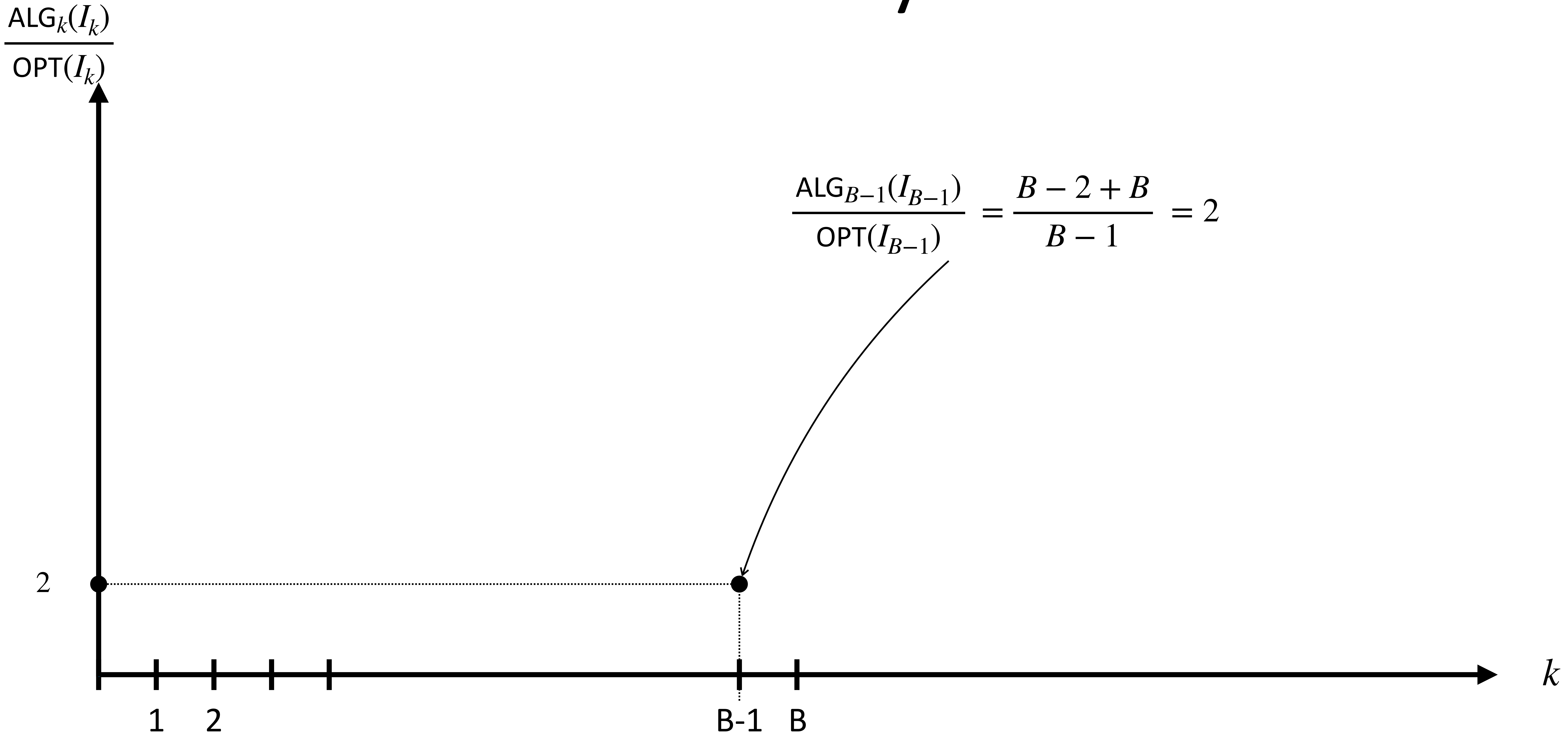


Lower Bound for the Buy-or-Rent Problem

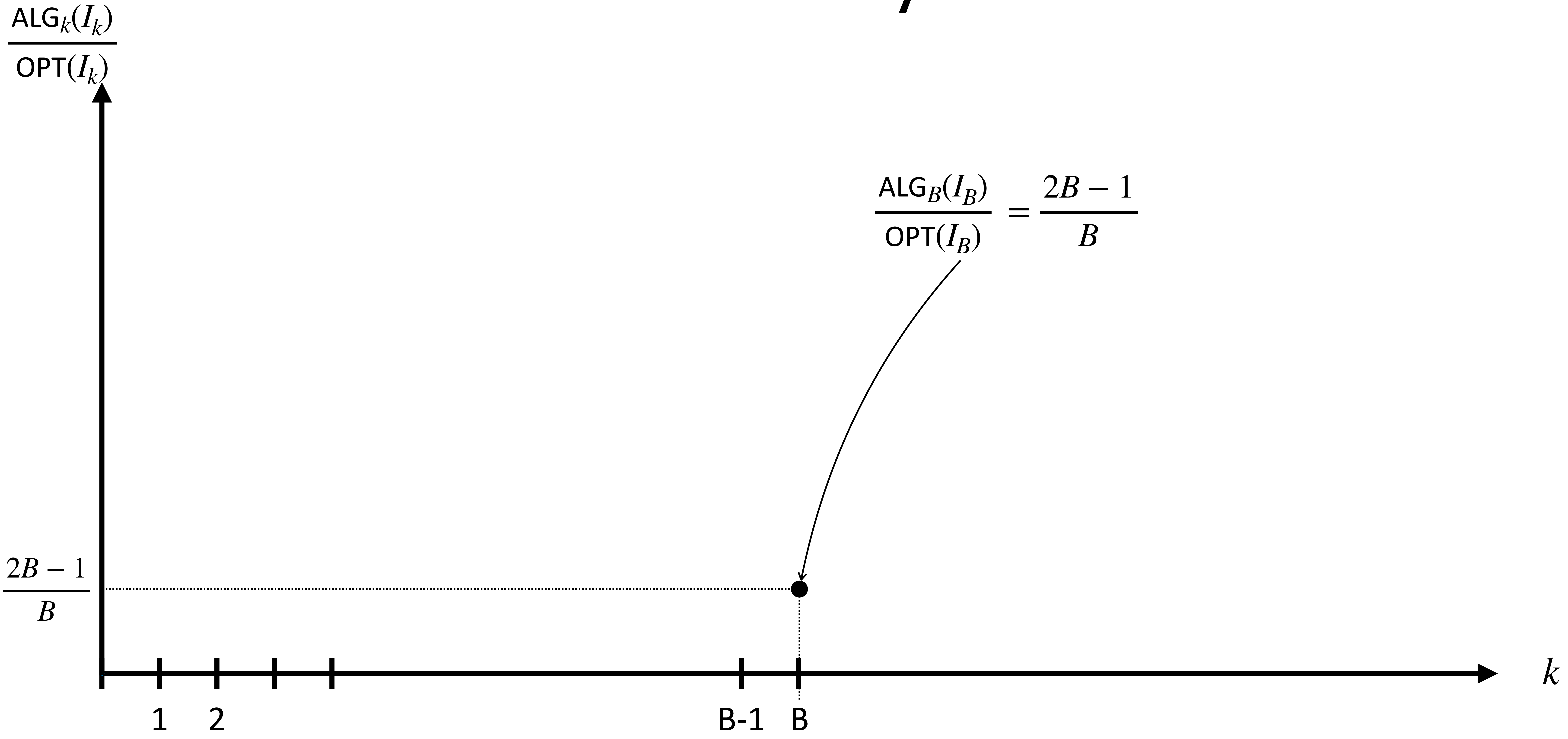
$\frac{\text{ALG}_4(I_4)}{\text{OPT}(I_4)} = \frac{3+B}{4}$ where I_4 is the instance with exactly 4 skiing days



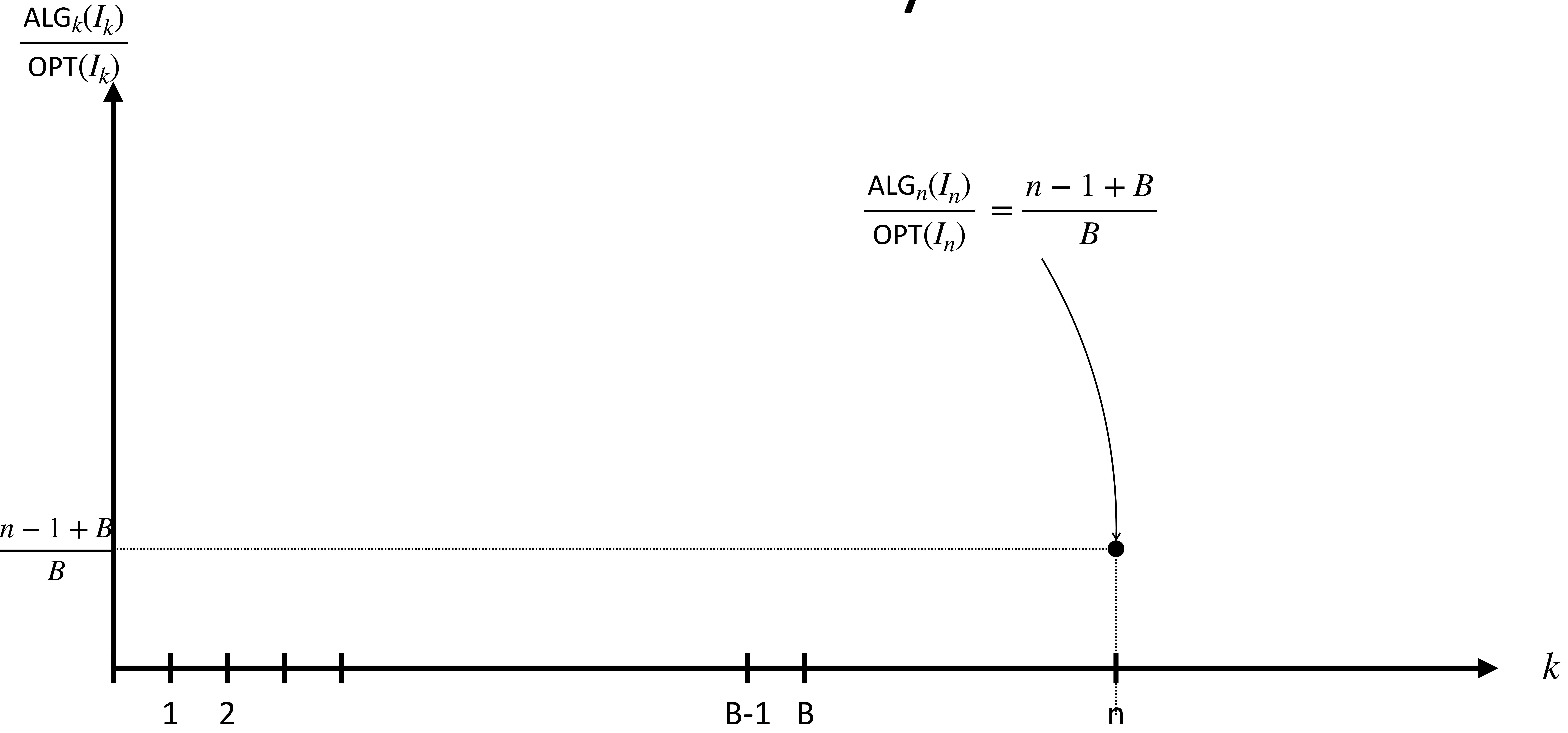
Lower Bound for the Buy-or-Rent Problem



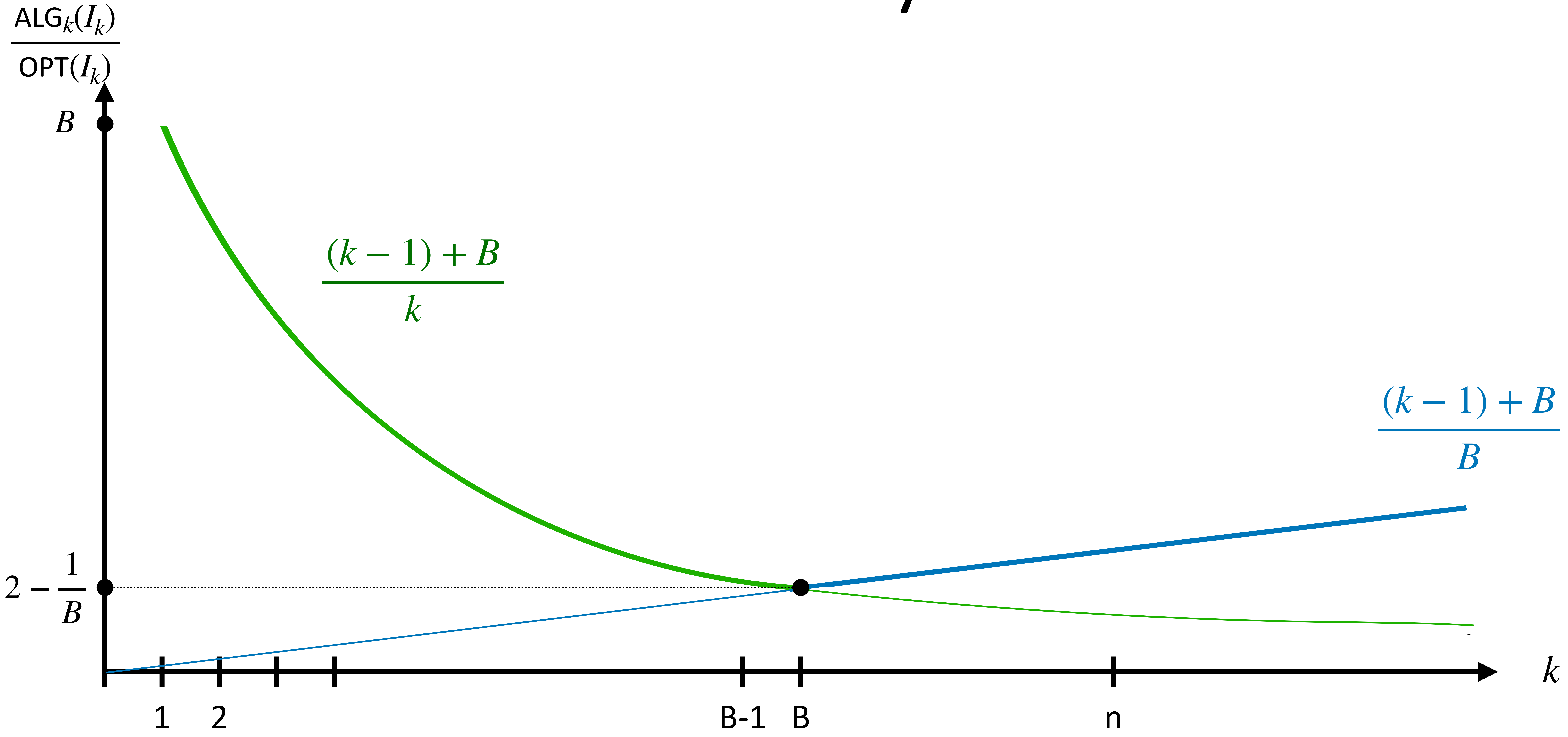
Lower Bound for the Buy-or-Rent Problem



Lower Bound for the Buy-or-Rent Problem



Lower Bound for the Buy-or-Rent Problem



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**

Recap: Online Optimization

An online problem

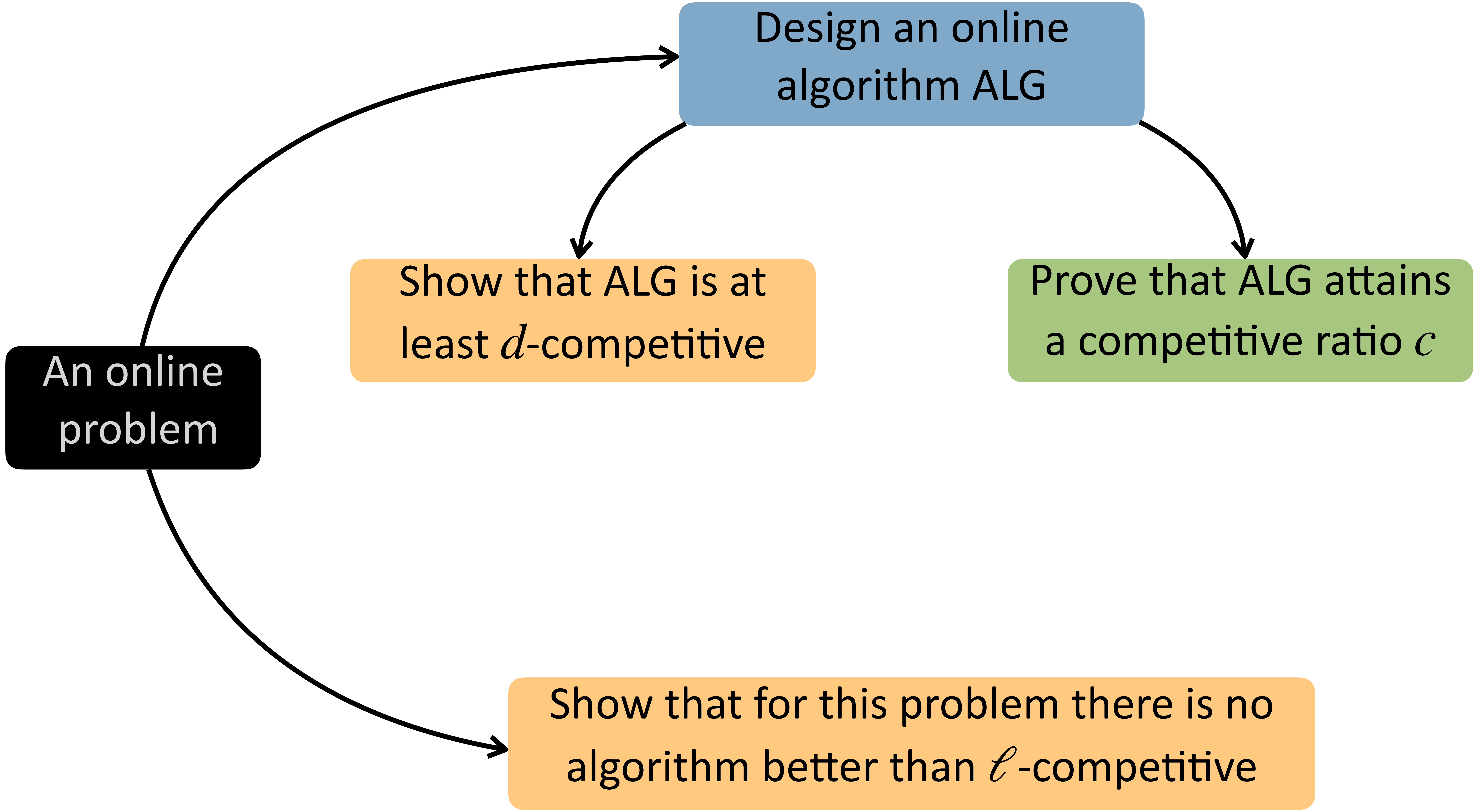
Recap: Online Optimization

Design an online algorithm ALG

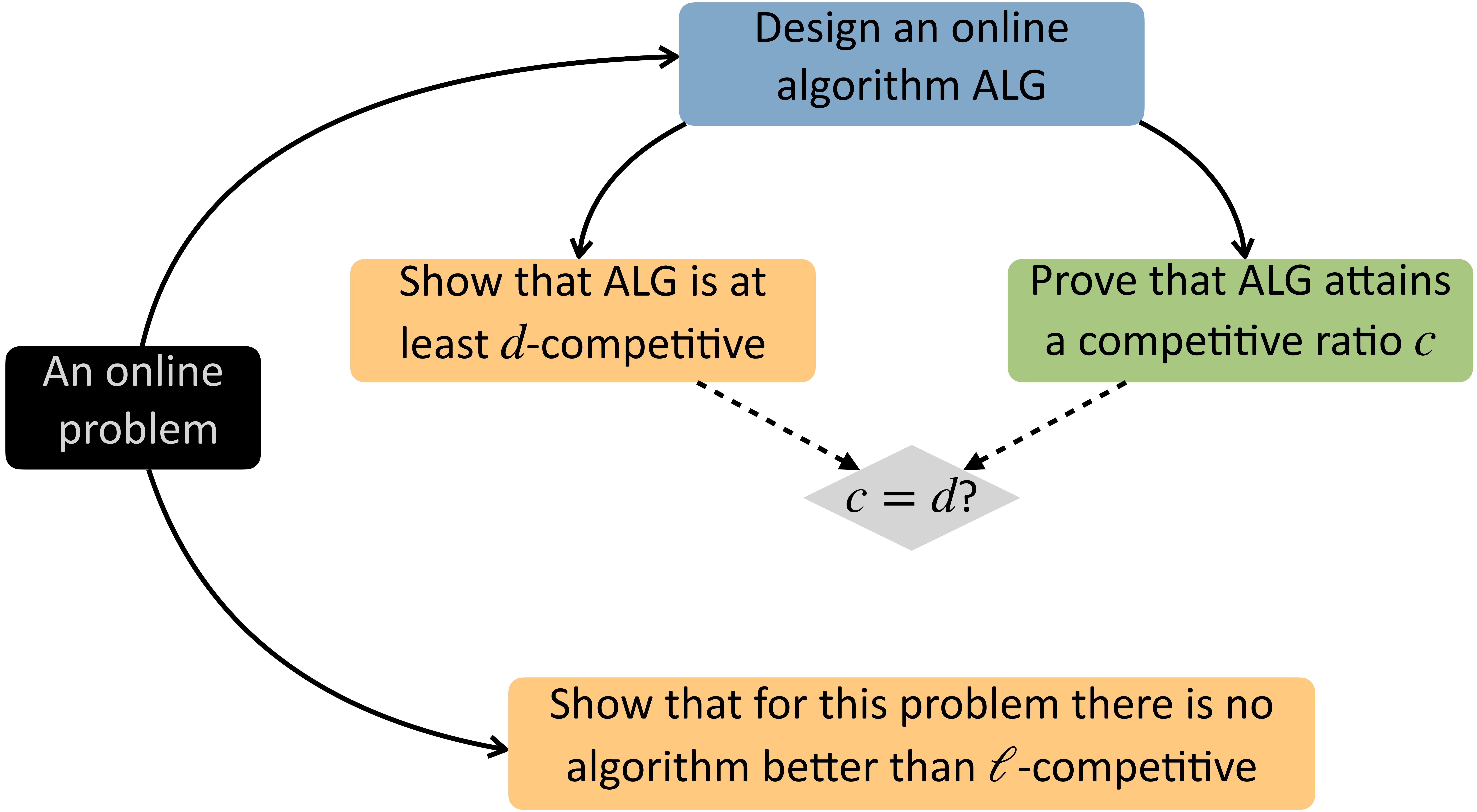
An online problem

Show that for this problem there is no algorithm better than ℓ-competitive

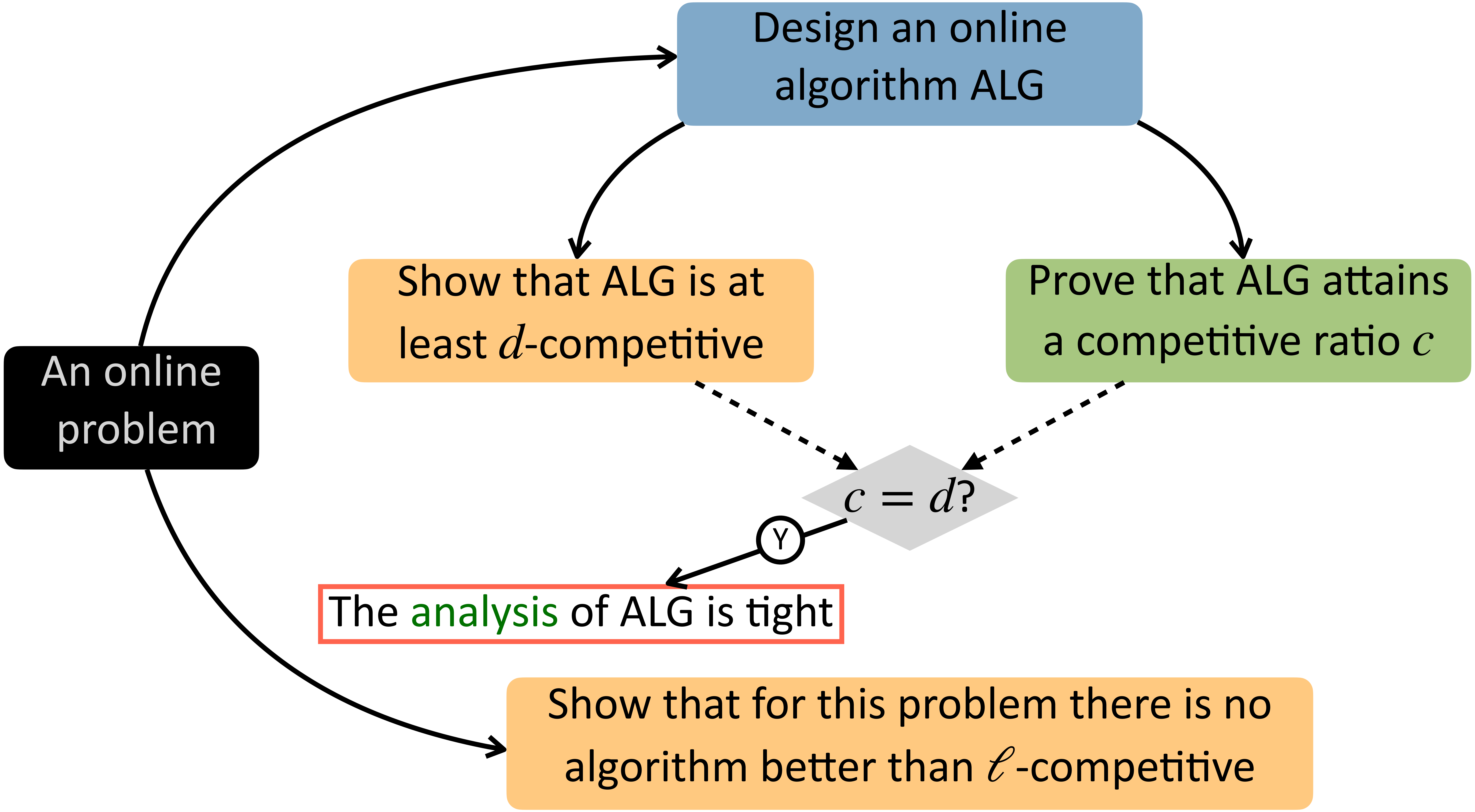
Recap: Online Optimization



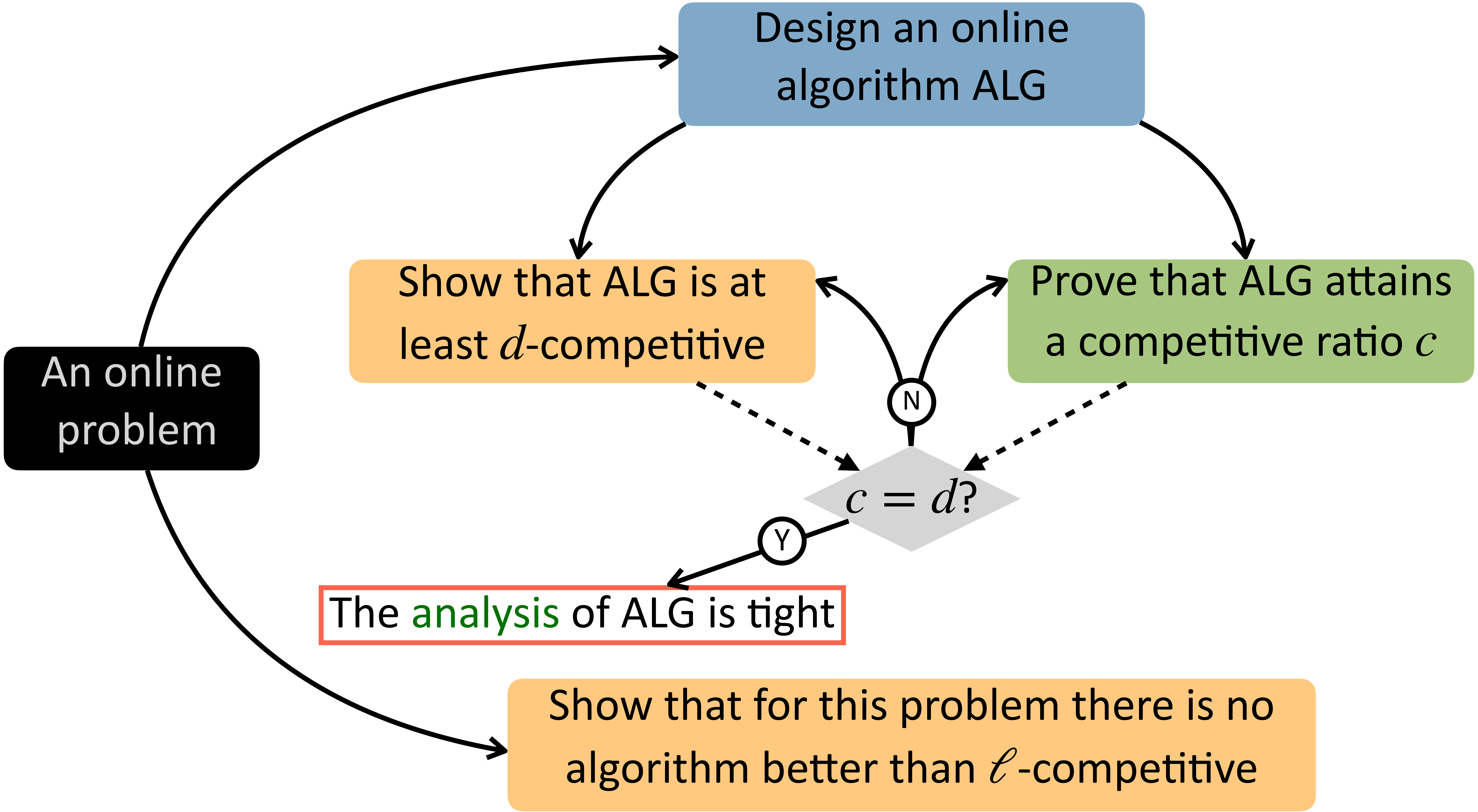
Recap: Online Optimization



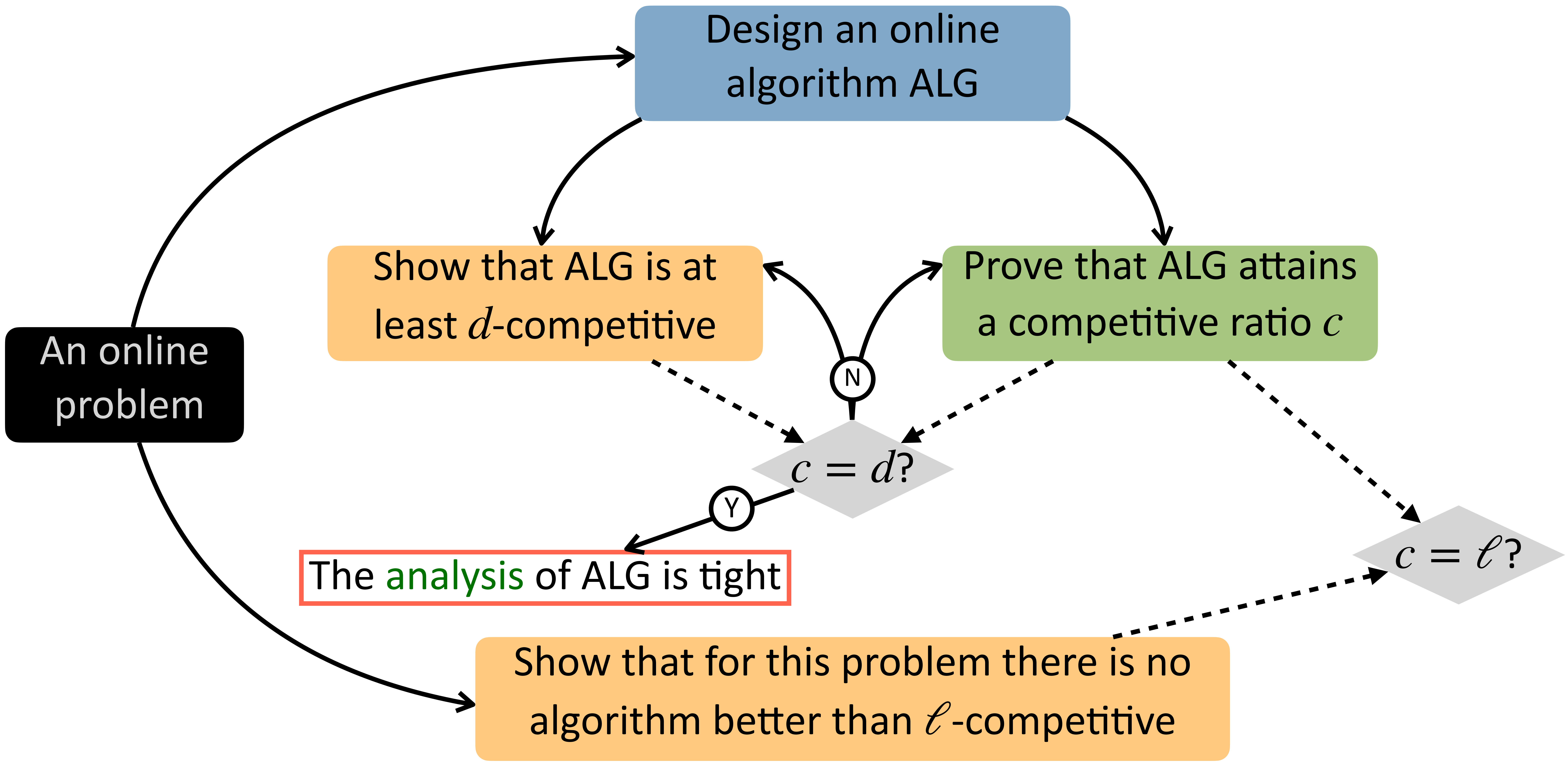
Recap: Online Optimization



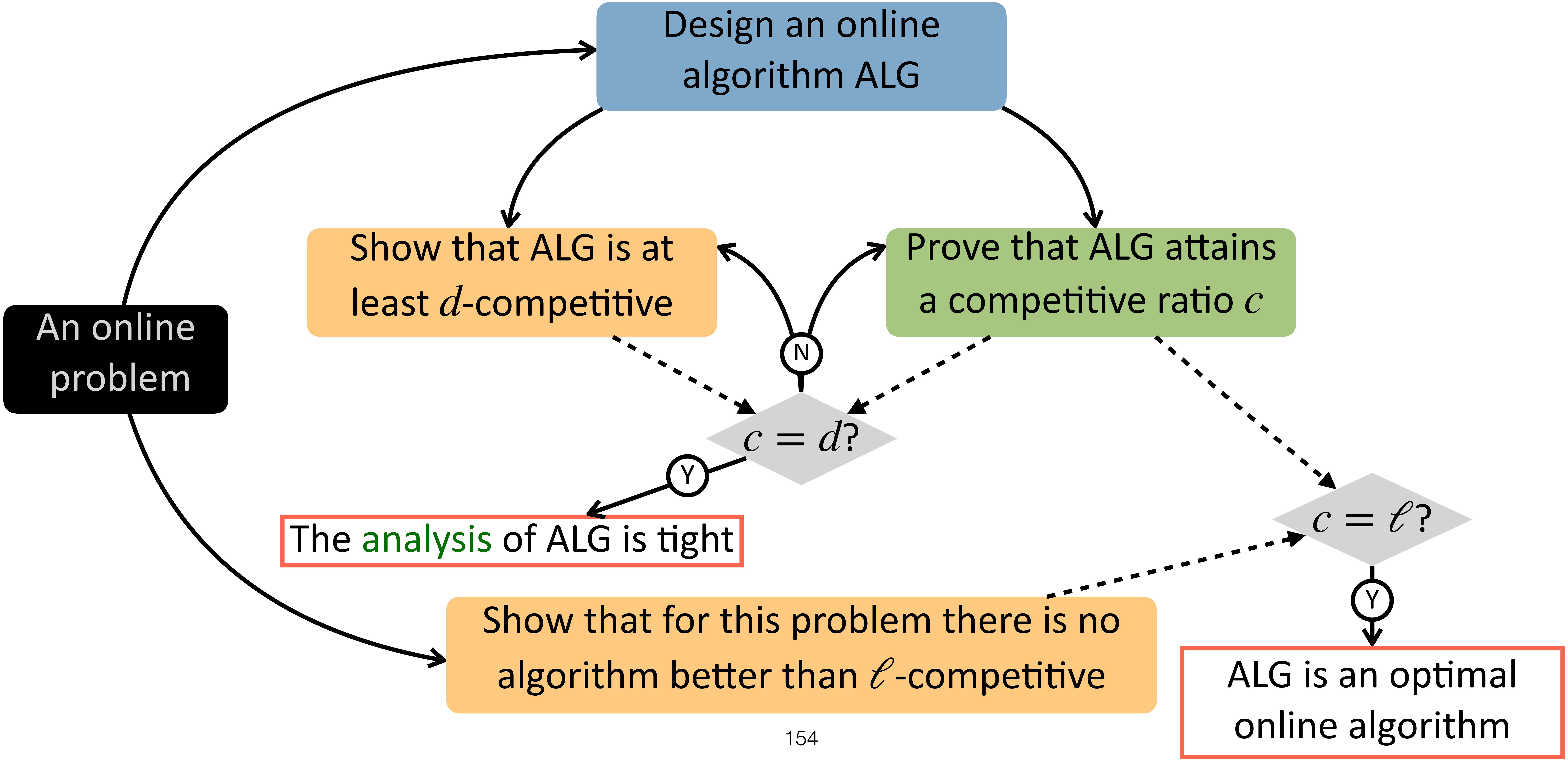
Recap: Online Optimization



Recap: Online Optimization



Recap: Online Optimization



Recap: Online Optimization