USING PREDICTIONS TO MAKE BETTER DECISIONS

ZIYAD BENOMAR

Structure of the talk

- Decision-making under uncertainty

- Learning-augmented algorithms

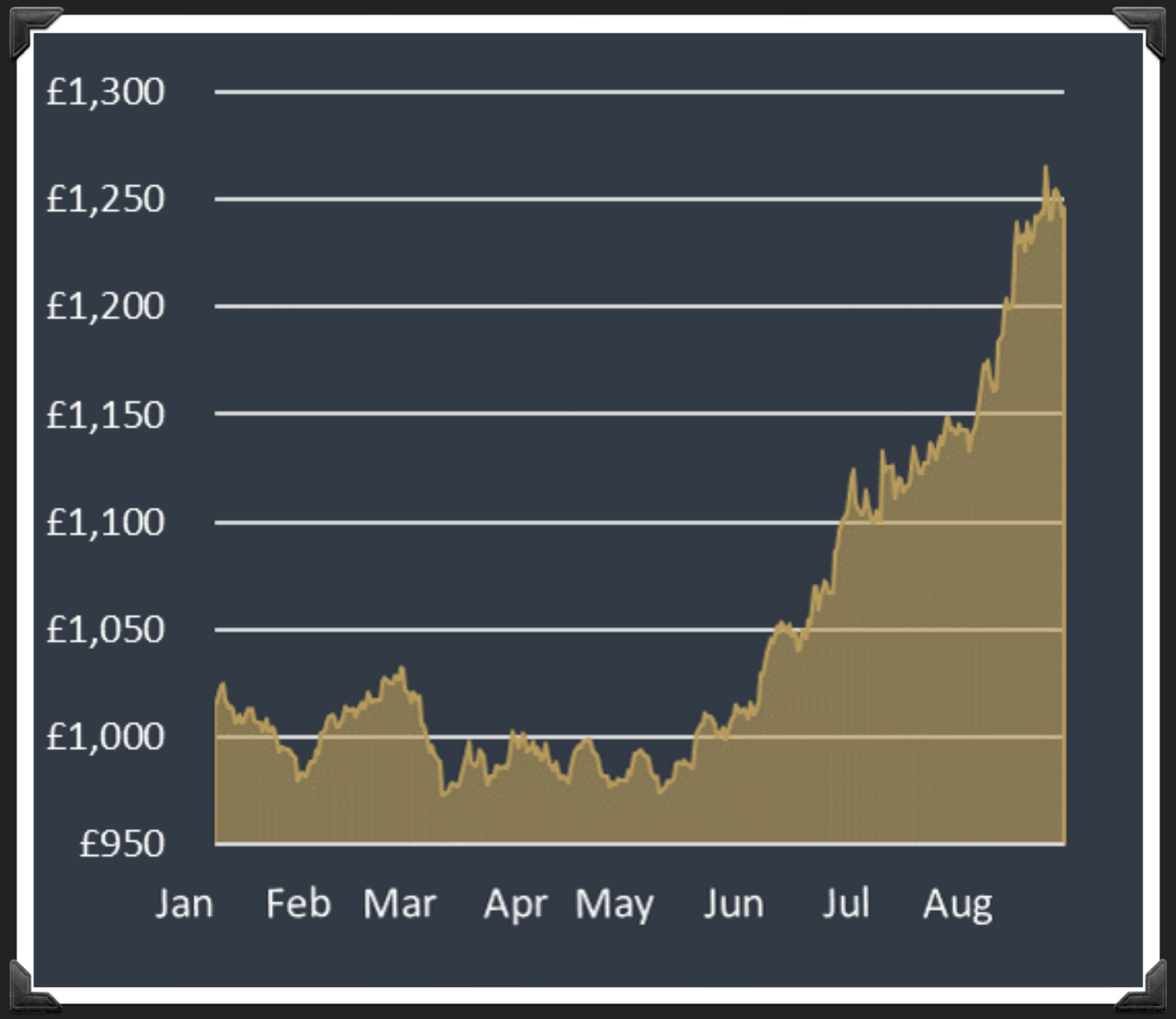
- Some research directions and open questions

Decision-making under uncertainty

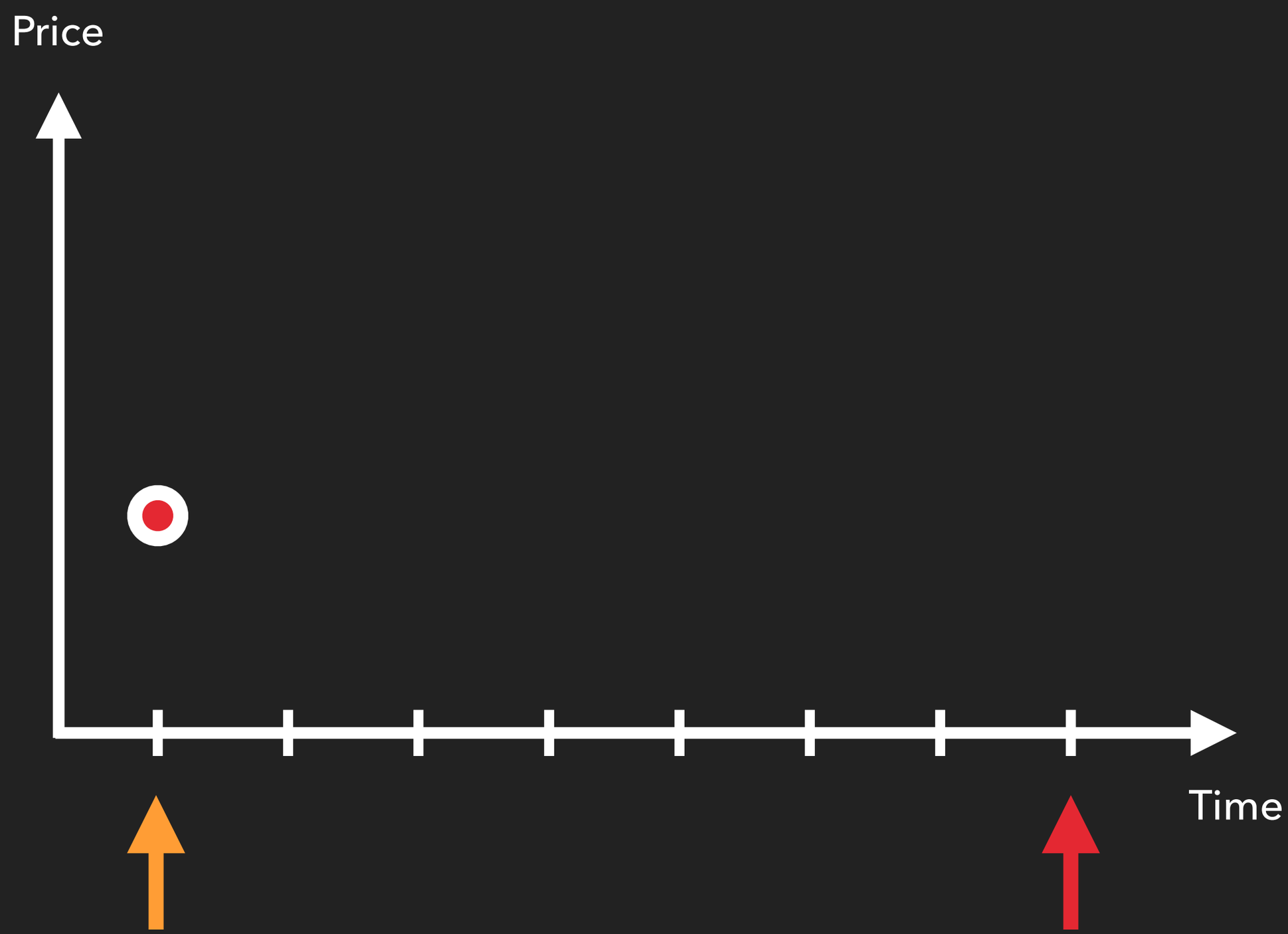
I have gold, and want to sell it

Decision-making under uncertainty

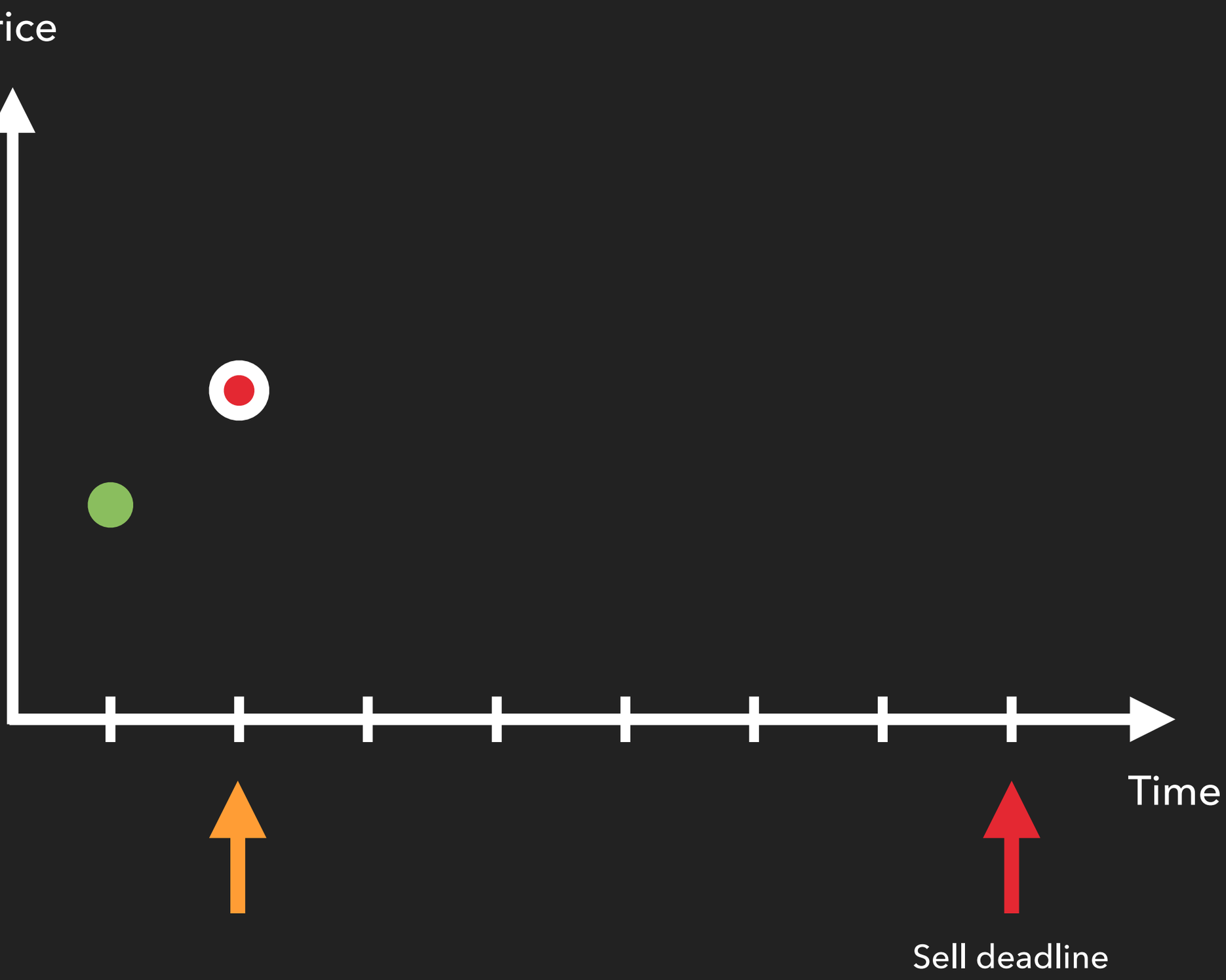
Problem: what is the best time for selling?



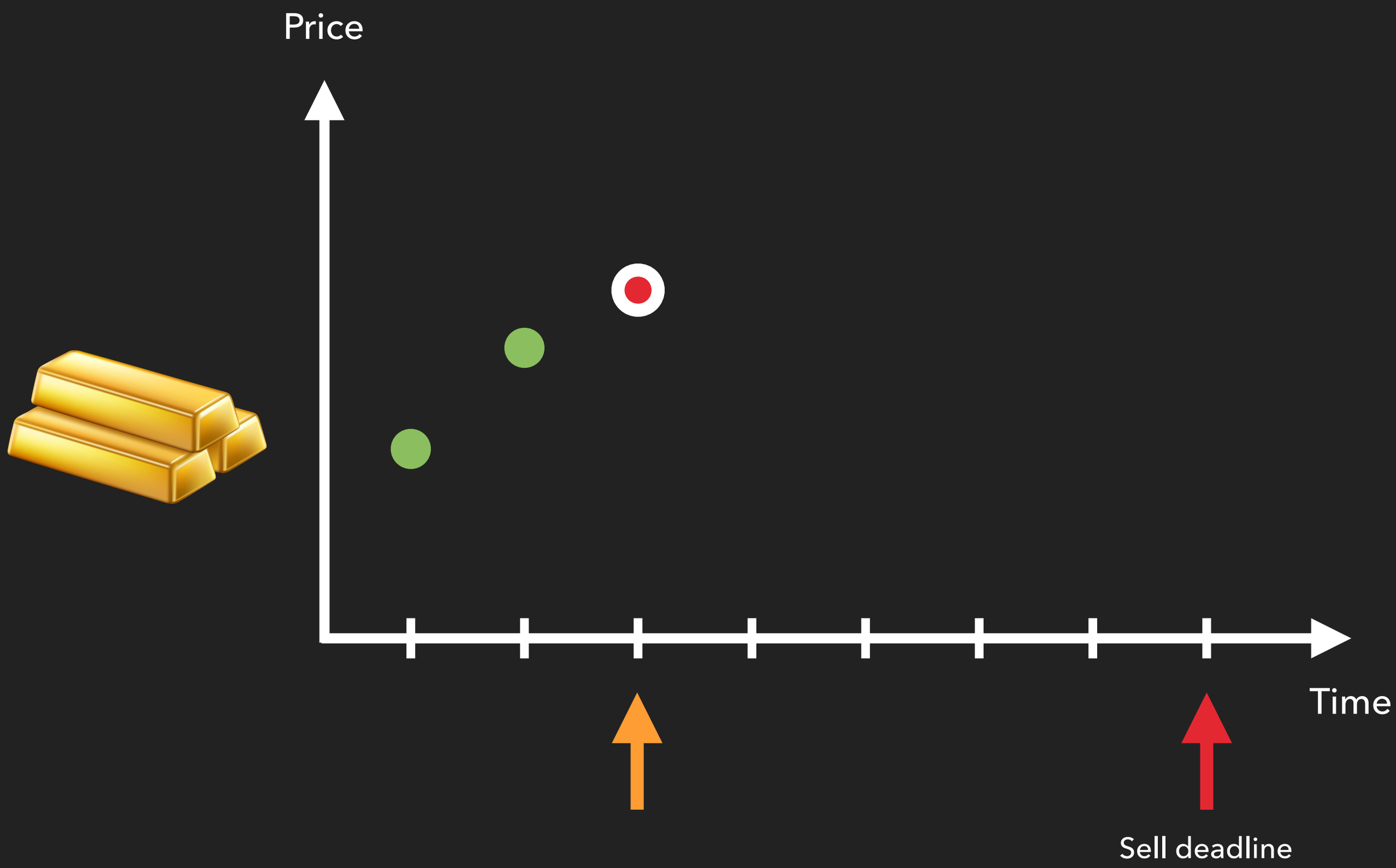
Decision-making under uncertainty



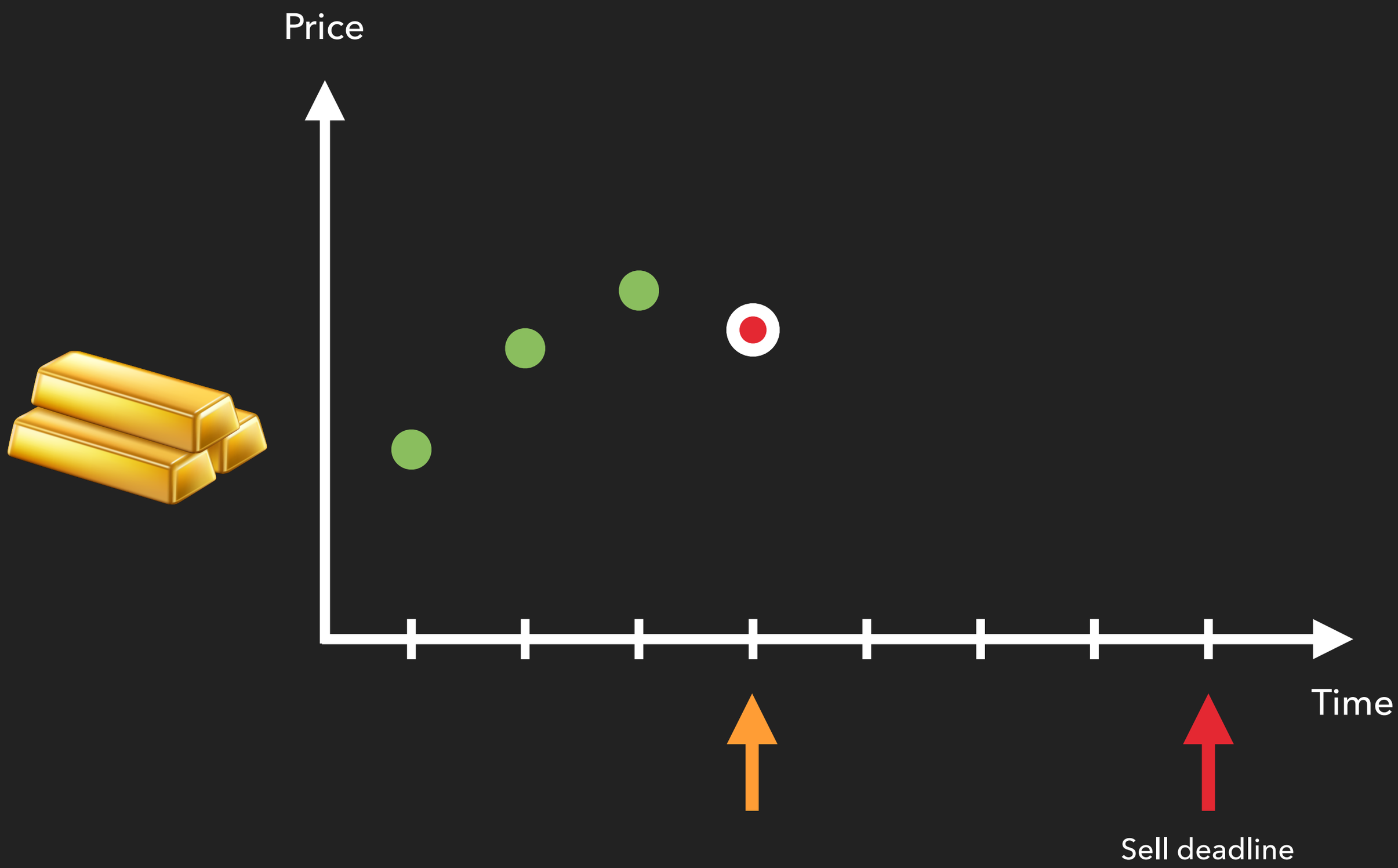
Decision-making under uncertainty



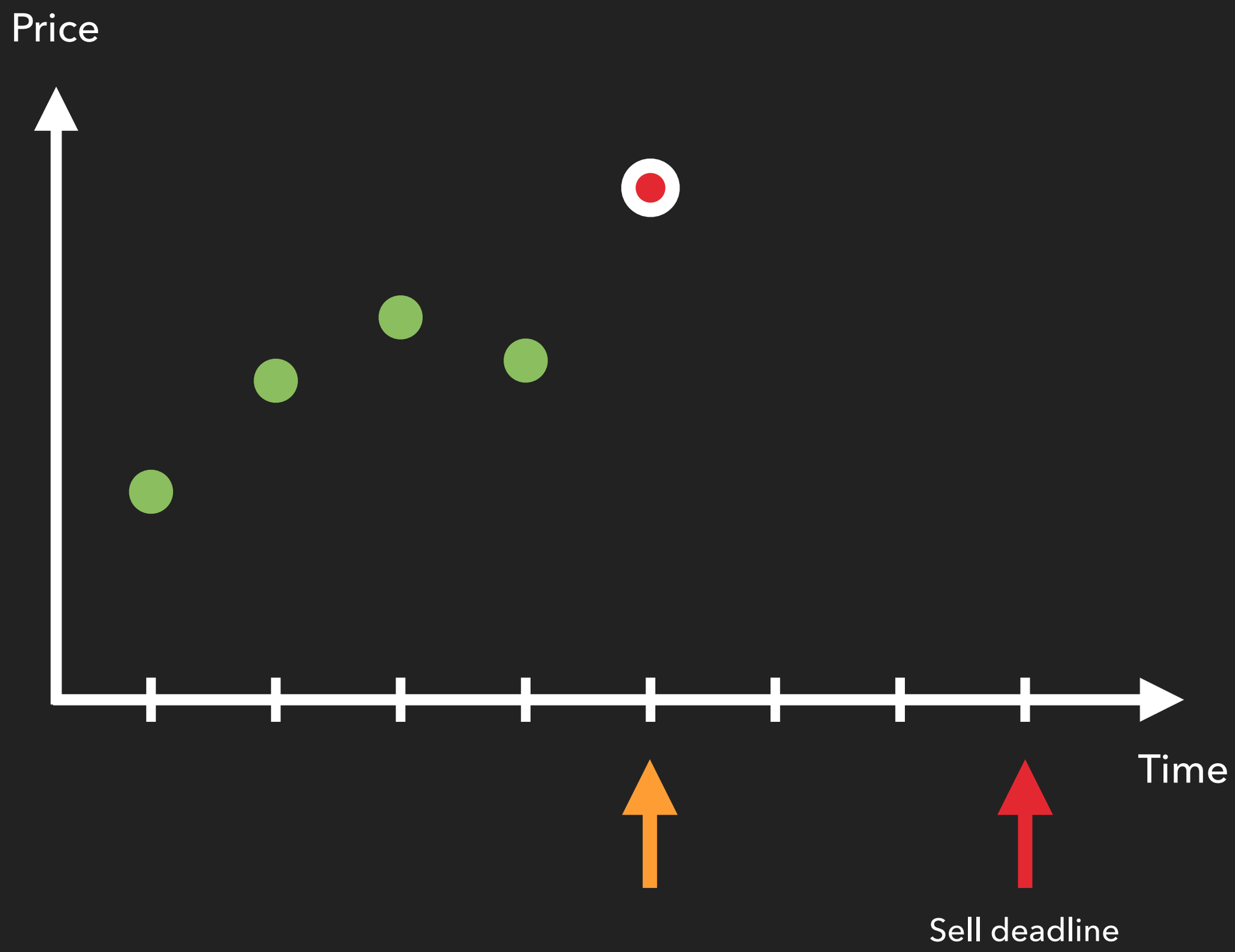
Decision-making under uncertainty



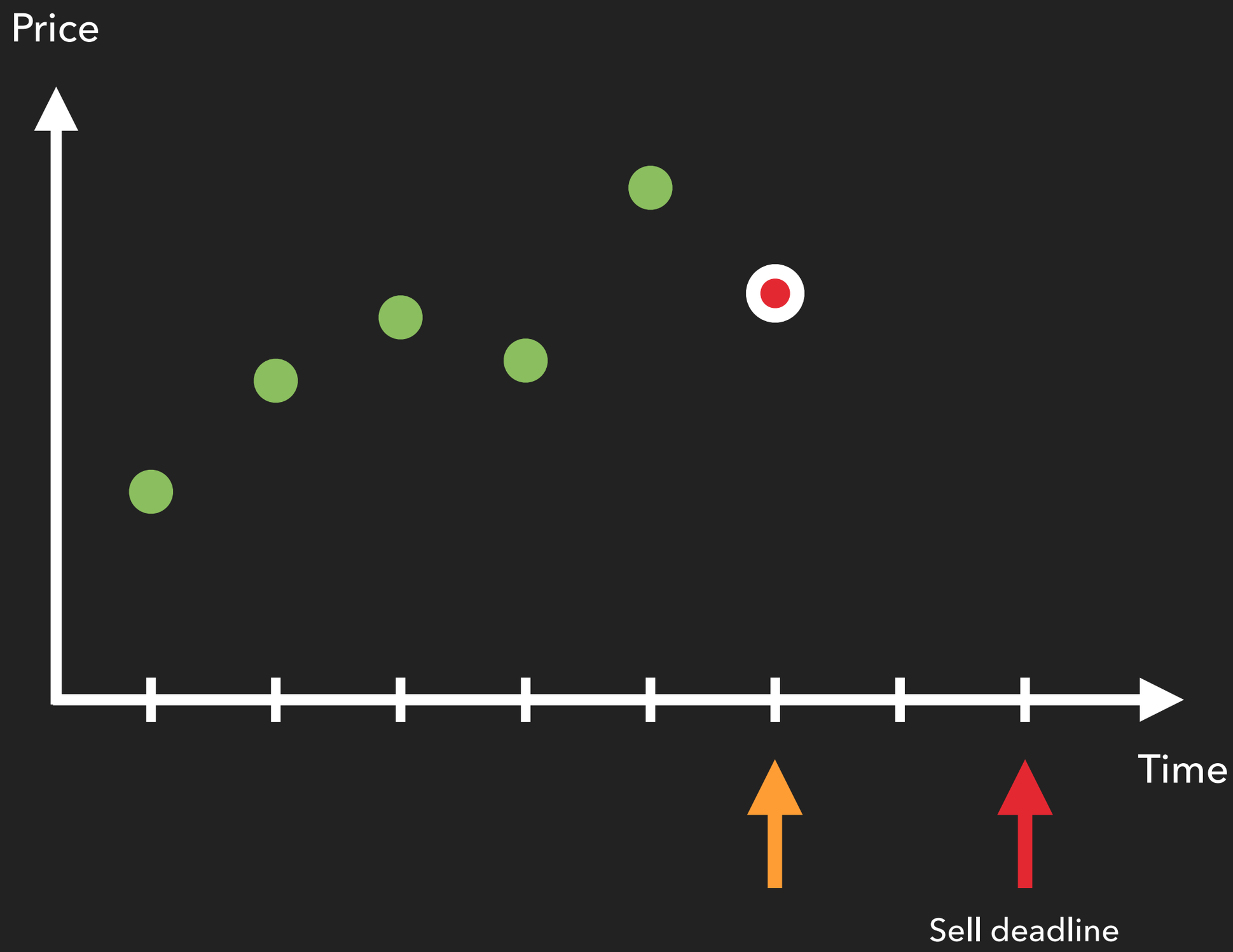
Decision-making under uncertainty



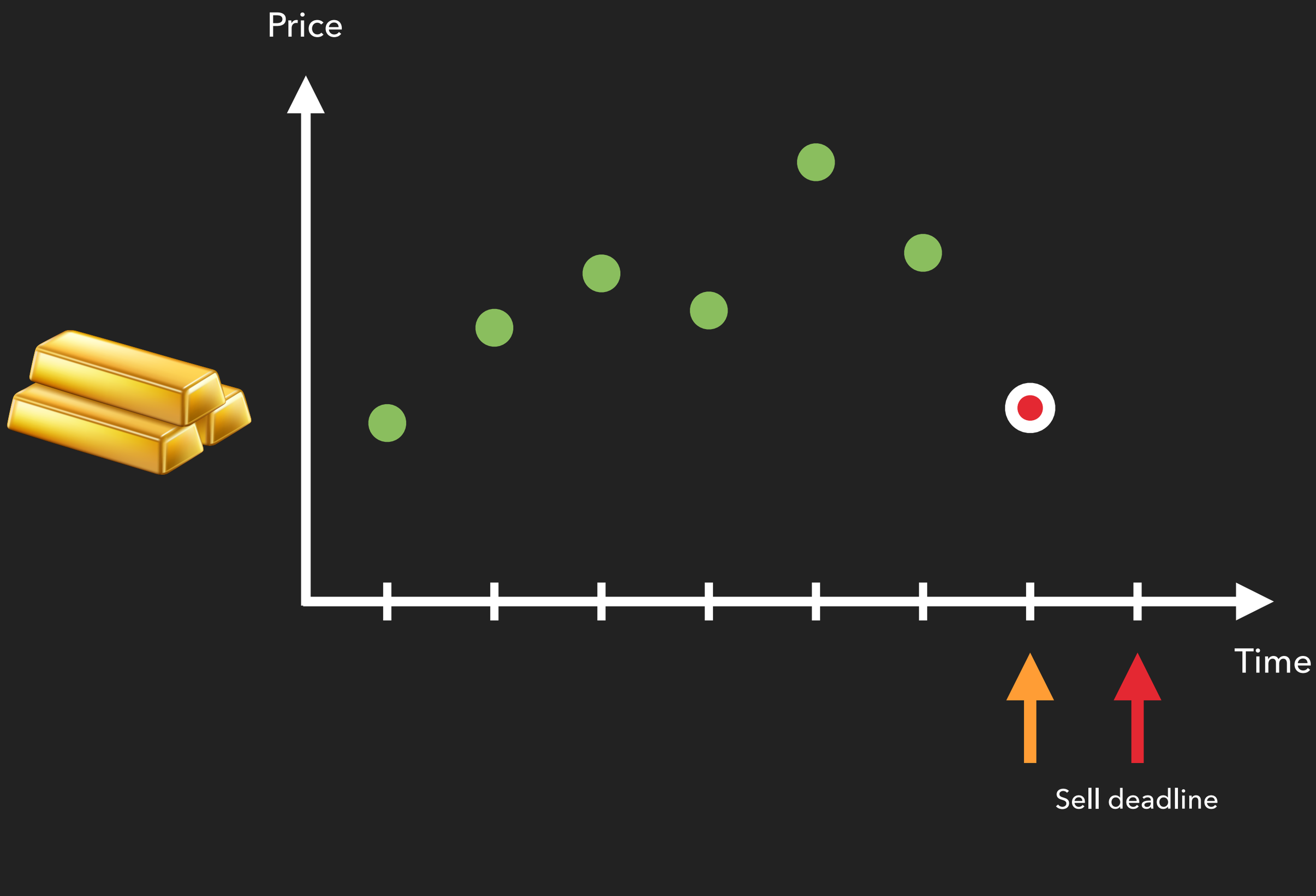
Decision-making under uncertainty



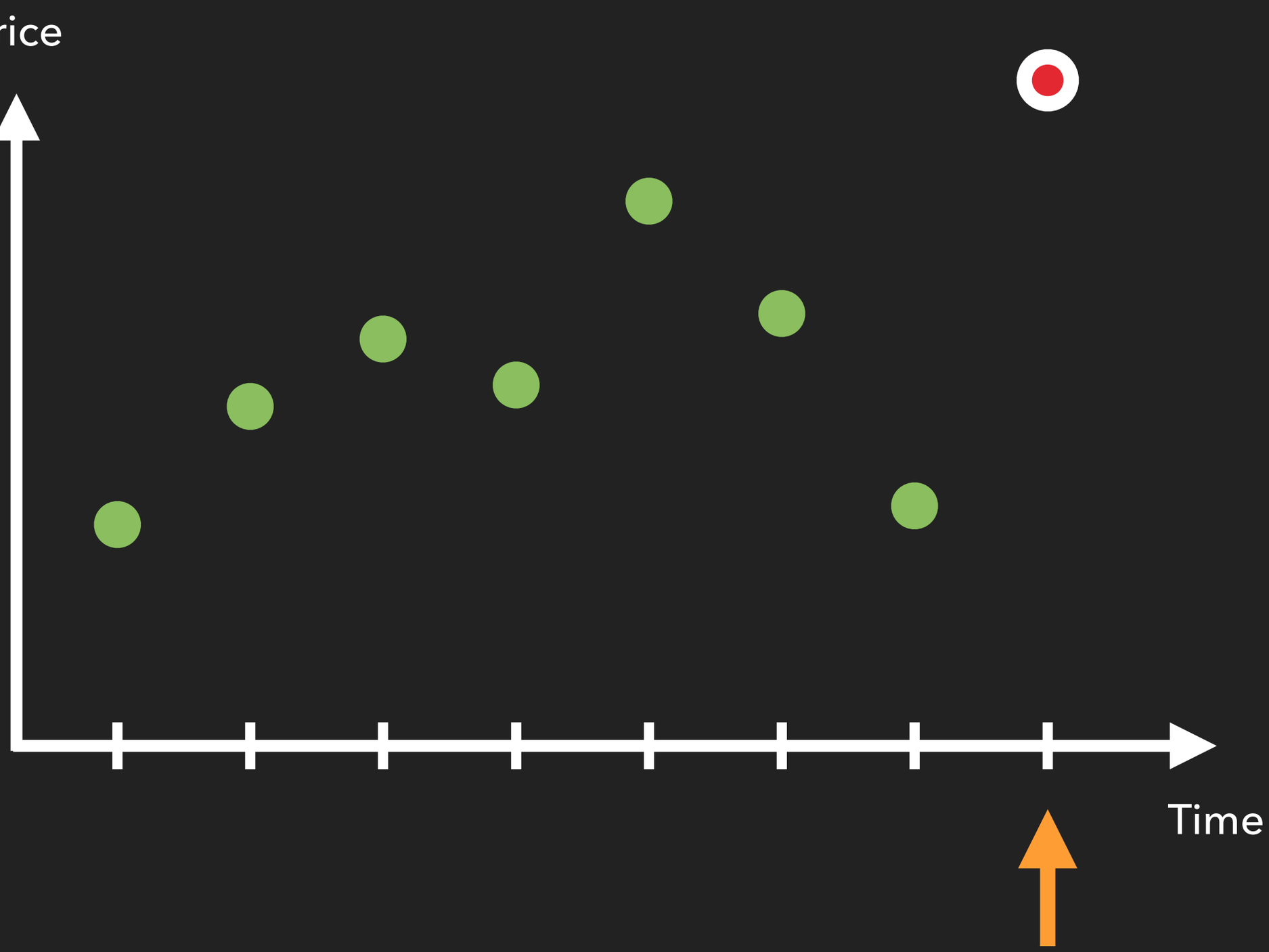
Decision-making under uncertainty



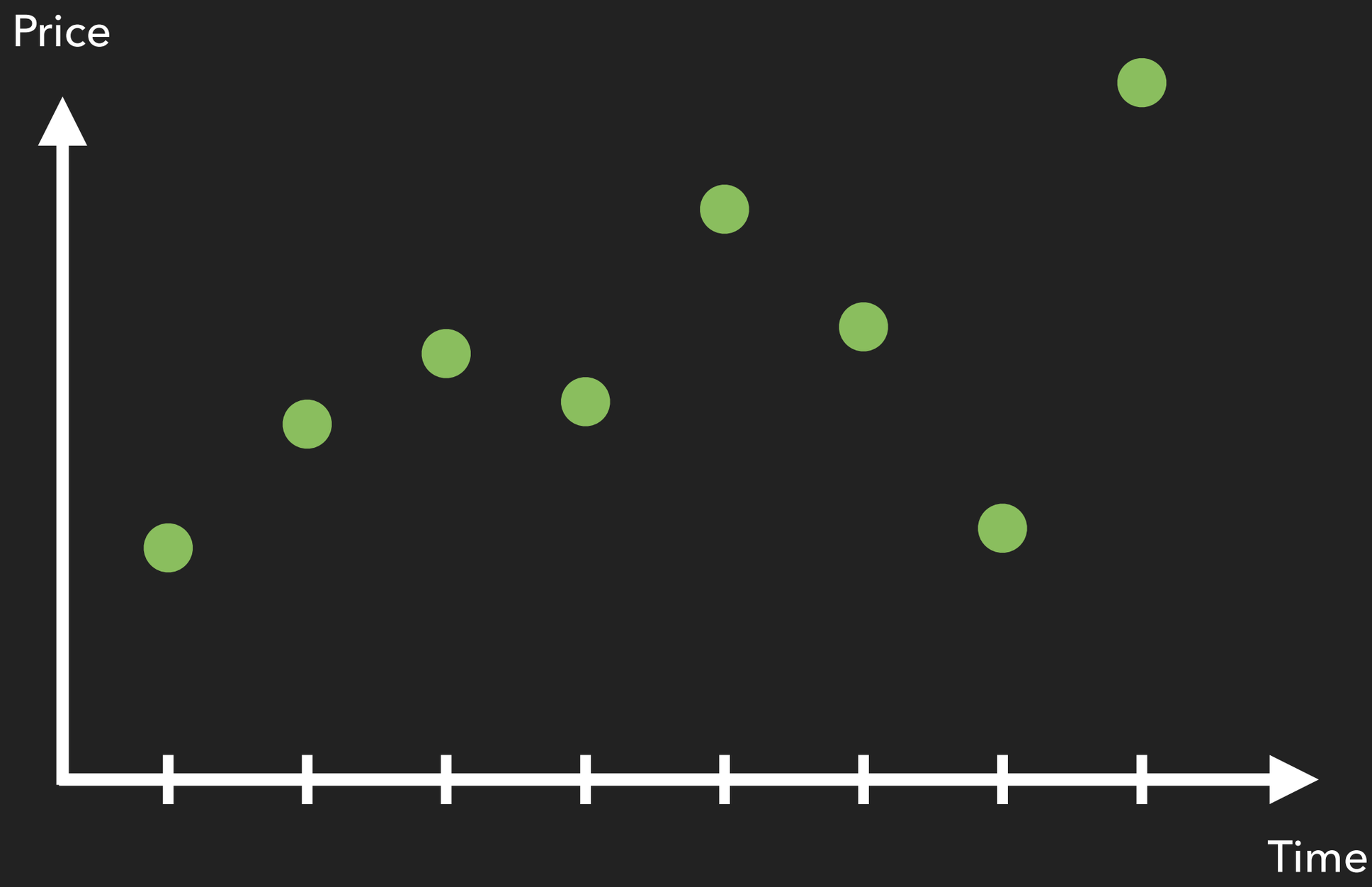
Decision-making under uncertainty



Decision-making under uncertainty



Decision-making under uncertainty



Decision-making under uncertainty

How to obtain guarantees on your strategy?

Worst case analysis: guarantees for any possible input

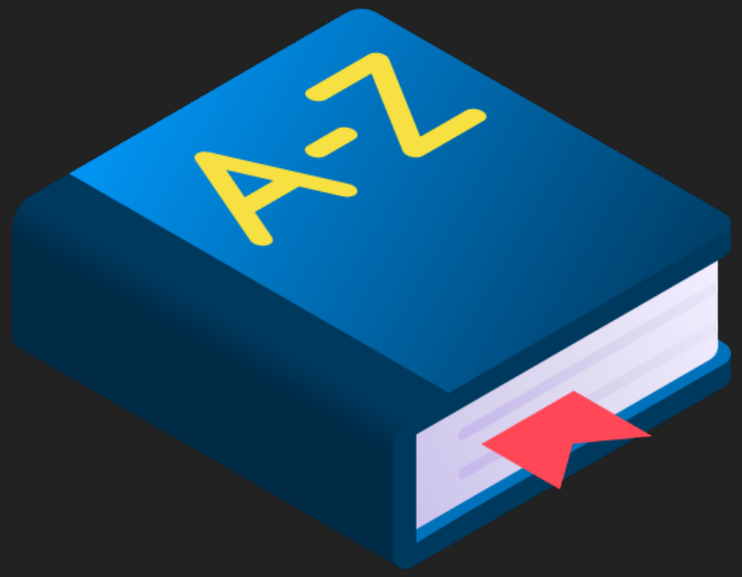
- The guarantees are very limited
- Overly pessimistic

Stochastic assumptions: consider that the input follows some distribution

- Guarantees on the average case
- Difficult in practice to know the distribution of the input

Example: searching in a sorted list

Find the word “wachlabas”

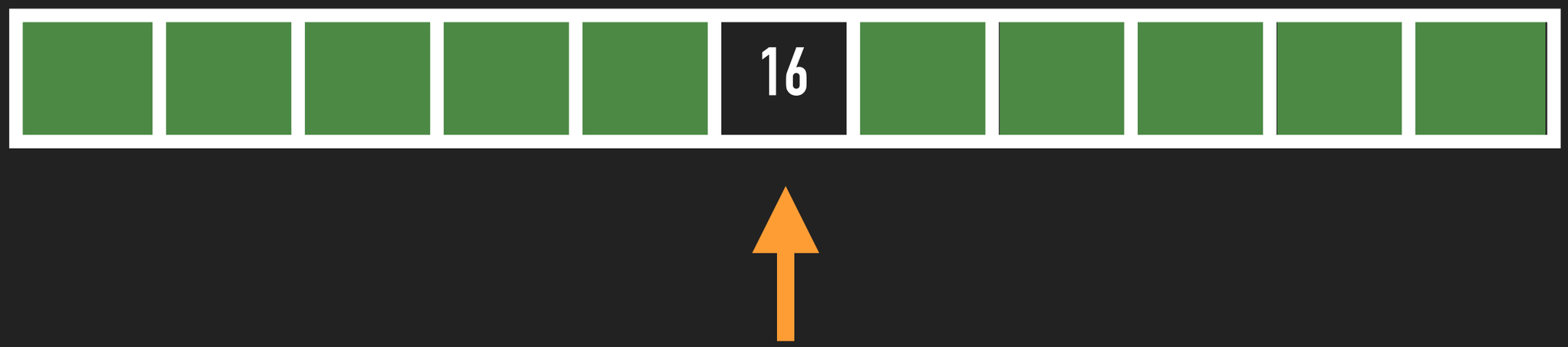


Example: searching in a sorted list

Find the number 27 in a sorted list

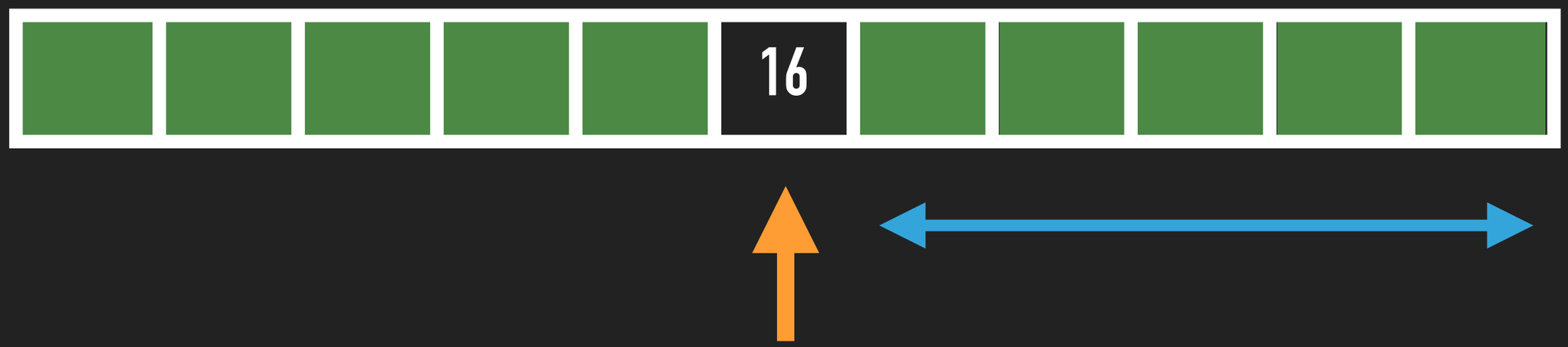
Example: searching in a sorted list

Find the number 27 in a sorted list



Example: searching in a sorted list

Find the number 27 in a sorted list

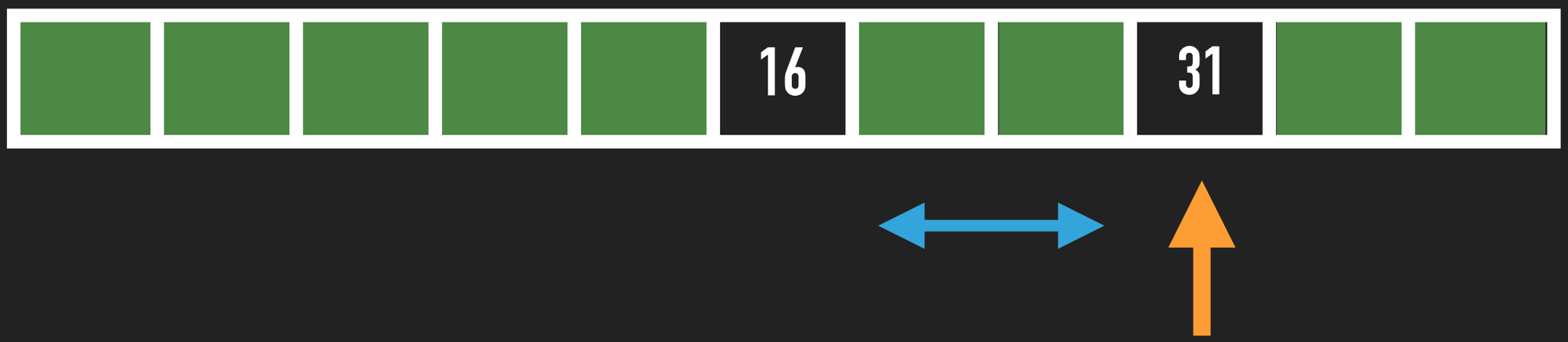


Example: searching in a sorted list

Find the number 27 in a sorted list

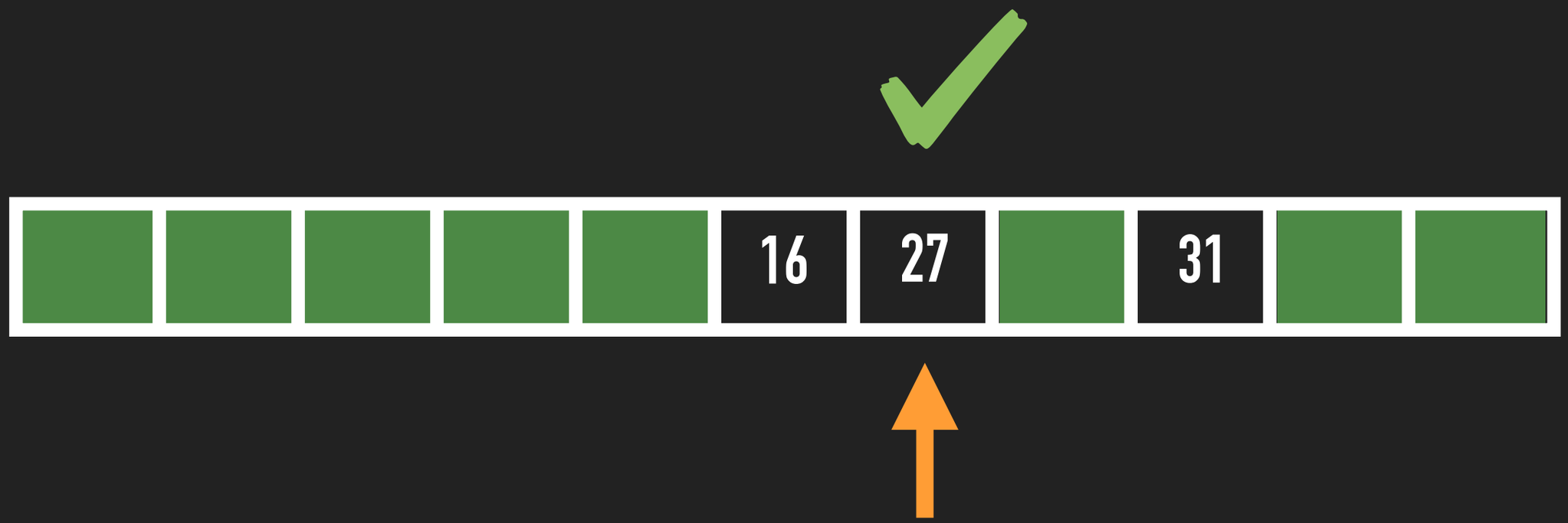
Example: searching in a sorted list

Find the number 27 in a sorted list



Example: searching in a sorted list

Find the number 27 in a sorted list



Example: searching in a sorted list

Find number v in a sorted list L of size n?

Example: searching in a sorted list

Find number v in a sorted list L of size n?

Worst case analysis: "Binary search"

- No assumption
- Complexity: $O(\log n)$ operations

Stochastic assumptions: "Interpolation search"

- The values in the list are i.i.d. samples from a known distribution F, then sorted
- Complexity: $O(\log \log n)$ in expectation

Example: searching in a sorted list

Find number v in a sorted list L of size n?

Worst case analysis: "Binary search"

- No assumption
- Complexity: $O(\log n)$ operations

Stochastic assumptions: "Interpolation search"

- The values in the list are i.i.d. samples from a known distribution F, then sorted
- Complexity: $O(\log \log n)$ in expectation

Learning-augmented algorithm: [Lykouris and Vassilvtiskii, ICML '18]

- The algorithm is given a prediction y of the true position x
- Complexity: $O(\log |x - y|)$

Example: searching in a sorted list

Find number 27 in a sorted list L of size n?

Prediction: “27 is in the position y“ (No guarantees on y)

Example: searching in a sorted list

Find number 27 in a sorted list L of size n?

Prediction: “27 is in the position y“ (No guarantees on y)

Example: searching in a sorted list

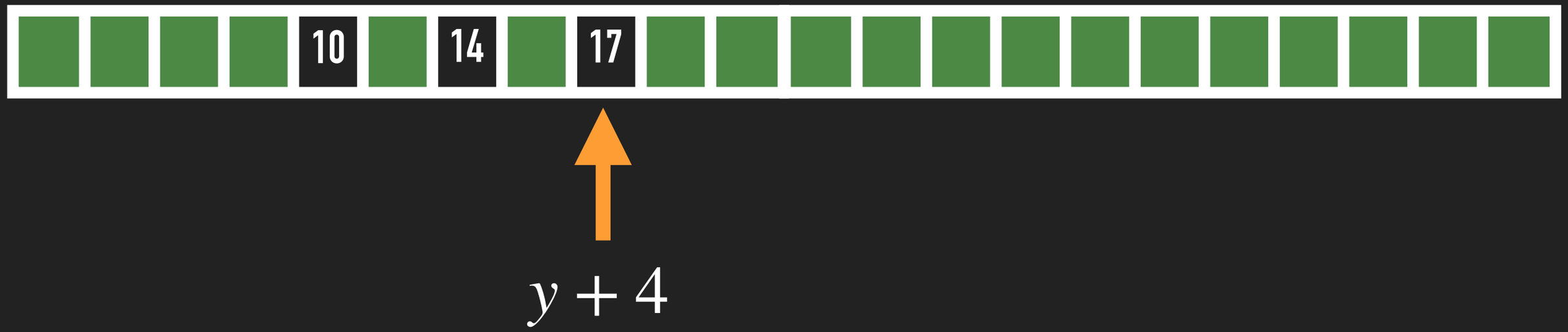
Find number 27 in a sorted list L of size n?

Prediction: “27 is in the position y“ (No guarantees on y)

Example: searching in a sorted list

Find number 27 in a sorted list L of size n?

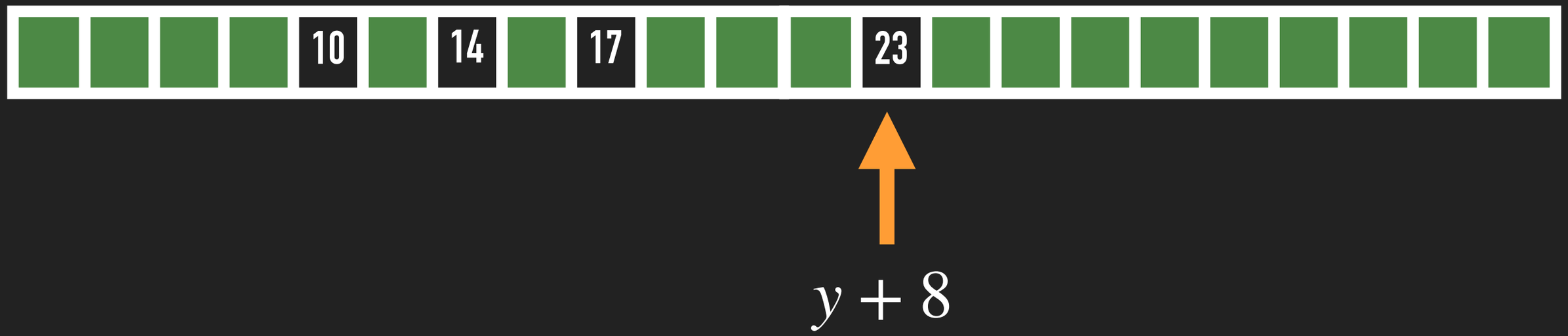
Prediction: “27 is in the position y“ (No guarantees on y)



Example: searching in a sorted list

Find number 27 in a sorted list L of size n?

Prediction: “27 is in the position y“ (No guarantees on y)



Example: searching in a sorted list

Find number 27 in a sorted list L of size n?

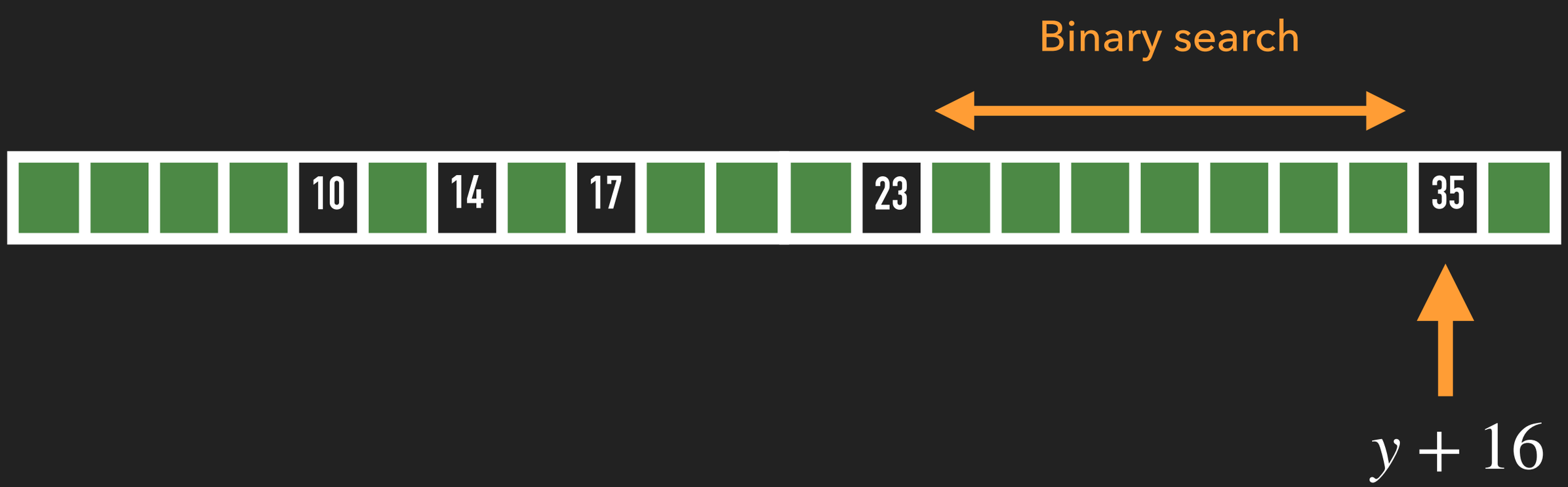
Prediction: “27 is in the position y“ (No guarantees on y)

$y + 16$

Example: searching in a sorted list

Find number 27 in a sorted list L of size n?

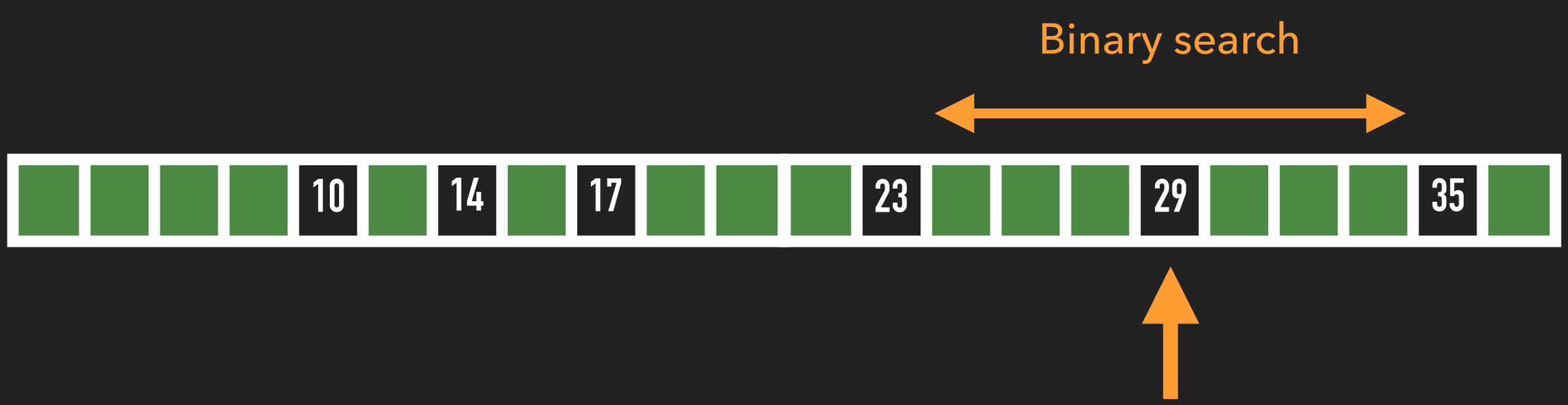
Prediction: "27 is in the position y" (No guarantees on y)



Example: searching in a sorted list

Find number 27 in a sorted list L of size n?

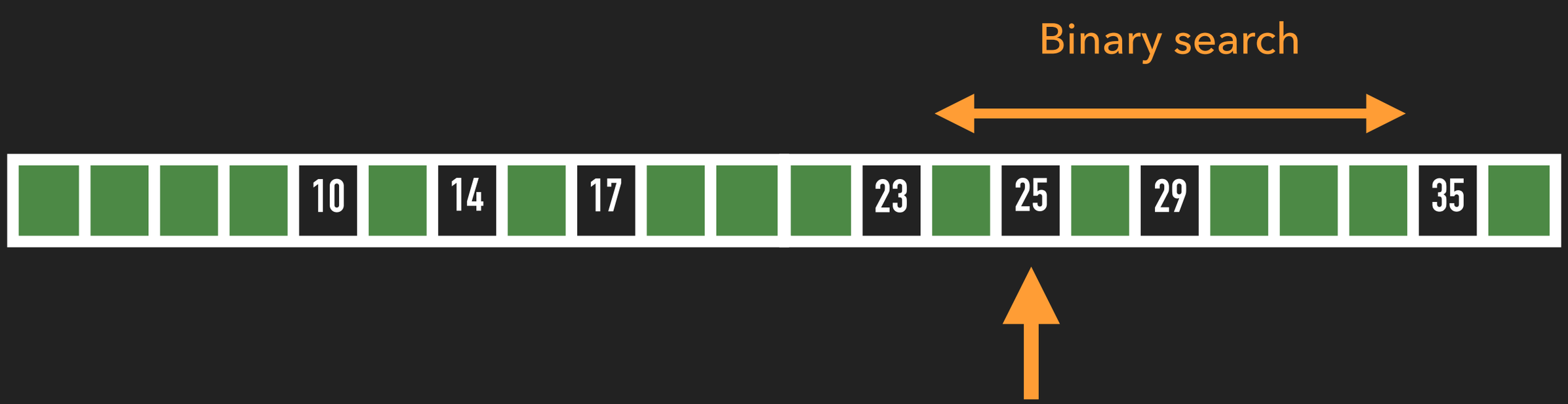
Prediction: "27 is in the position y" (No guarantees on y)



Example: searching in a sorted list

Find number 27 in a sorted list L of size n?

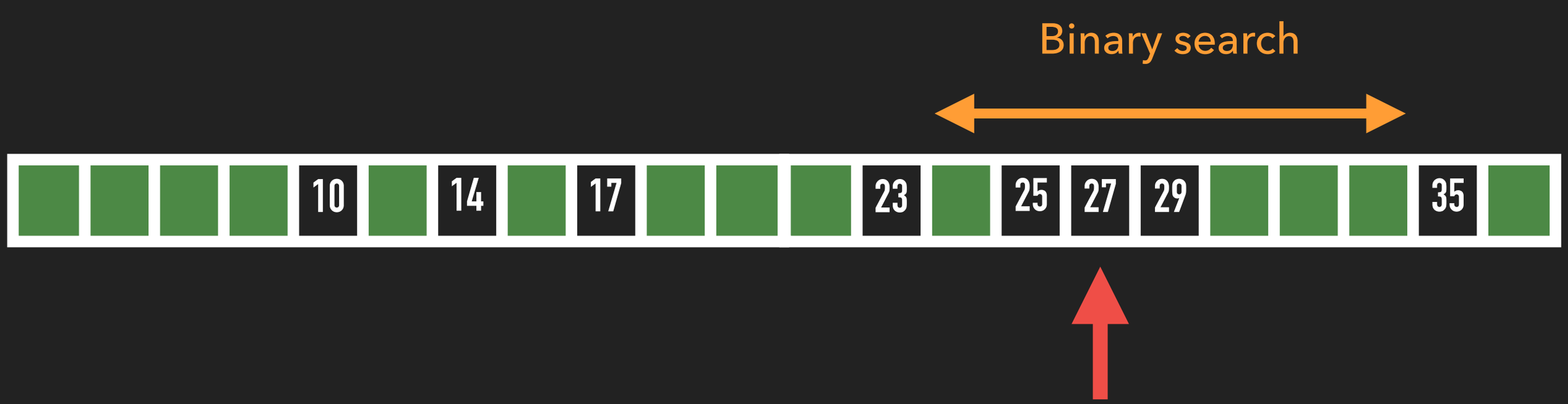
Prediction: "27 is in the position y" (No guarantees on y)



Example: searching in a sorted list

Find number 27 in a sorted list L of size n?

Prediction: "27 is in the position y" (No guarantees on y)



Example: searching in a sorted list

Find number 27 in a sorted list L of size n?

Prediction: "27 is in the position y" (No guarantees on y)

Complexity: $O(\log|x-y|) =$

$O(\log n)$ if the prediction is bad

$O(1)$ if the prediction is good

Example 2: Rent or buy

Example 2: Rent or buy

Example 2: Rent or buy

Example 2: Rent or buy

Example 2: Rent or buy

Example 2: Rent or buy

Price of bus ticket/subscription:

1 = cost of a single bus ticket

b = cost of subscription for one year

Example 2: Rent or buy

Price of bus ticket/subscription:

1 = cost of a single bus ticket

b = cost of subscription for one year

Decision-making problem: keep “renting“ or “buy”?

Optimal solution: If the number of raining x days is known

Total renting and buying cost: $\min(x, b)$

Example 2: Rent or buy

Price of bus ticket/subscription:

1 = cost of a single bus ticket

b = cost of subscription for one year

Decision-making problem: keep “renting“ or “buy”?

Difficulty: the number of raining x days is unknown

Example 2: Rent or buy

- Given b, but ignoring x, decide when to buy

Example 2: Rent or buy

- Given b, but ignoring x, decide when to buy
- ALG(b,x) = total cost of renting and buying
- Objective: Pay a cost as close as possible to $\min(x, b)$

Example 2: Rent or buy

- Given b, but ignoring x, decide when to buy
- ALG(b,x) = total cost of renting and buying
- Objective: Pay a cost as close as possible to $\min(x, b)$

Theorem: for any algorithm, there exists $b, x > 0$ such that

$$\text{ALG}(b, x) \geq 2 \min(b, x)$$

Example 2: Rent or buy

- Given b, but ignoring x, decide when to buy
- ALG(b,x) = total cost of renting and buying
- Objective: Pay a cost as close as possible to $\min(x, b)$

Theorem: for any algorithm, there exists $b, x > 0$ such that

$$\text{ALG}(b, x) \geq 2\min(b, x)$$

Algorithm: The algorithm that "rents for b days then buys" satisfies

$$\forall x, b: \qquad \text{ALG}(b, x) \leq 2\min(b, x)$$

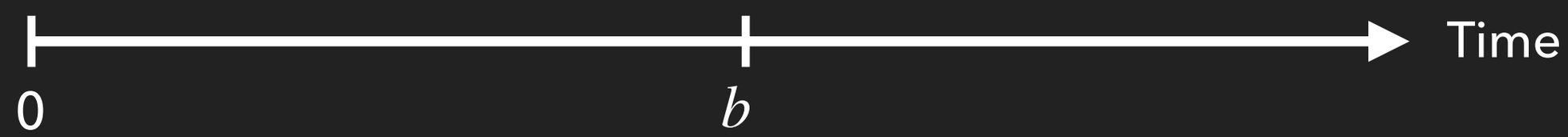
Example 2: Rent or buy

- Given b, but ignoring x, decide when to buy
- Additional input: prediction y of x (with no guarantees)

Example 2: Rent or buy

- Given b, but ignoring x, decide when to buy
- Additional input: prediction y of x (with no guarantees)

Algorithm: Follow the prediction… with precaution

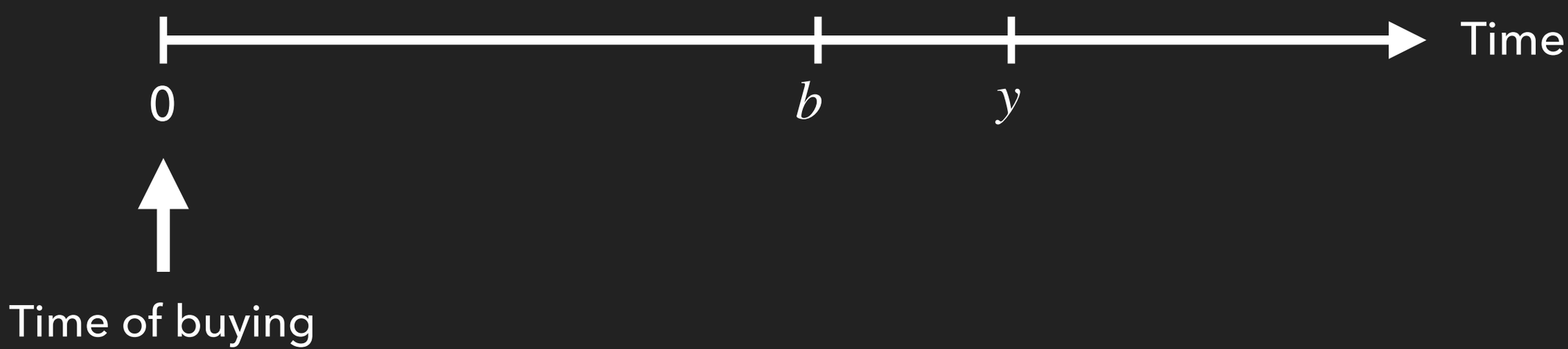


Example 2: Rent or buy

- Given b, but ignoring x, decide when to buy
- Additional input: prediction y of x (with no guarantees)

Algorithm: Follow the prediction…

If $y \geq b$

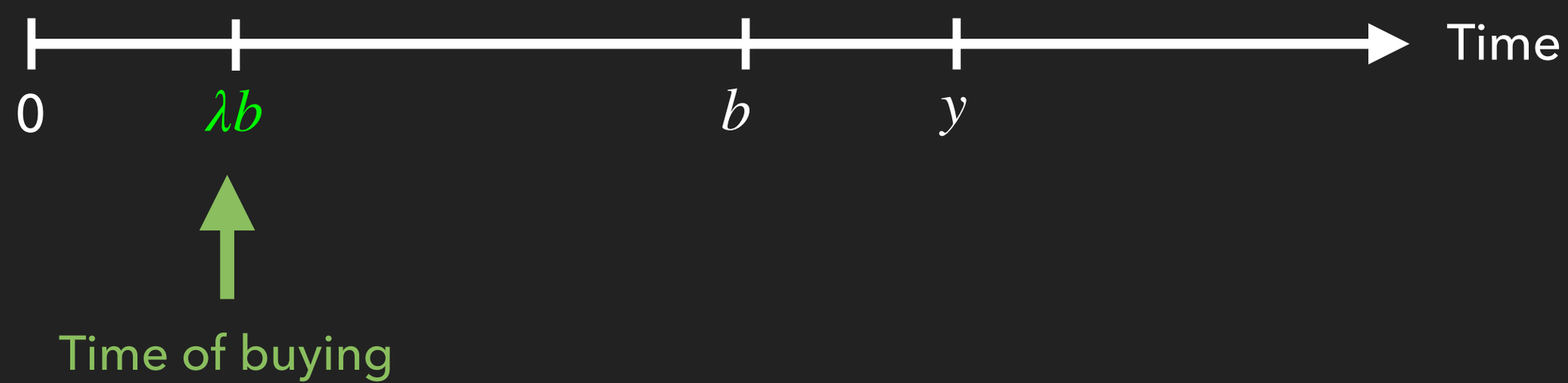


Example 2: Rent or buy

- Given b, but ignoring x, decide when to buy
- Additional input: prediction y of x (with no guarantees)

Algorithm: Follow the prediction… with precaution $\lambda \in [0,1]$

If $y \geq b$

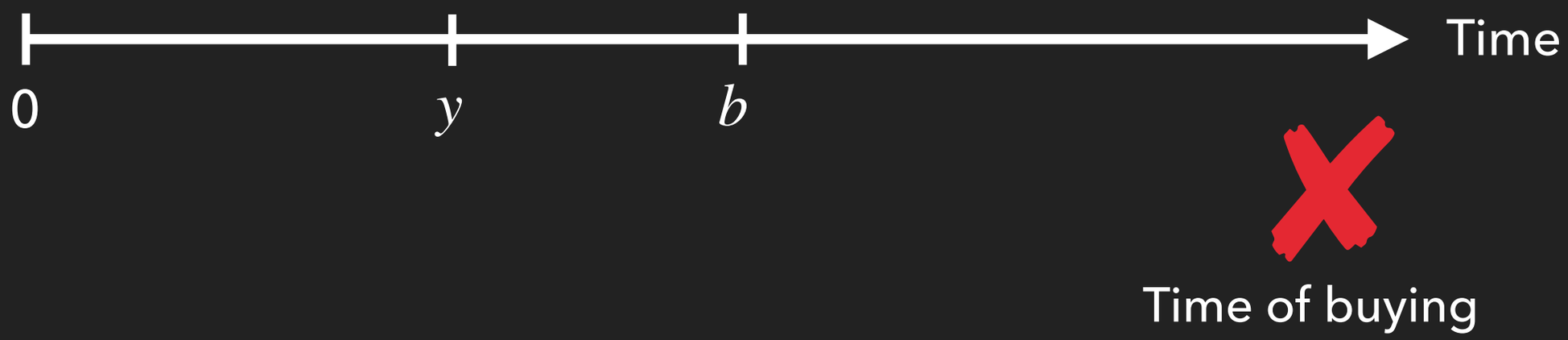


Example 2: Rent or buy

- Given b, but ignoring x, decide when to buy
- Additional input: prediction y of x (with no guarantees)

Algorithm: Follow the prediction…

If $y < b$

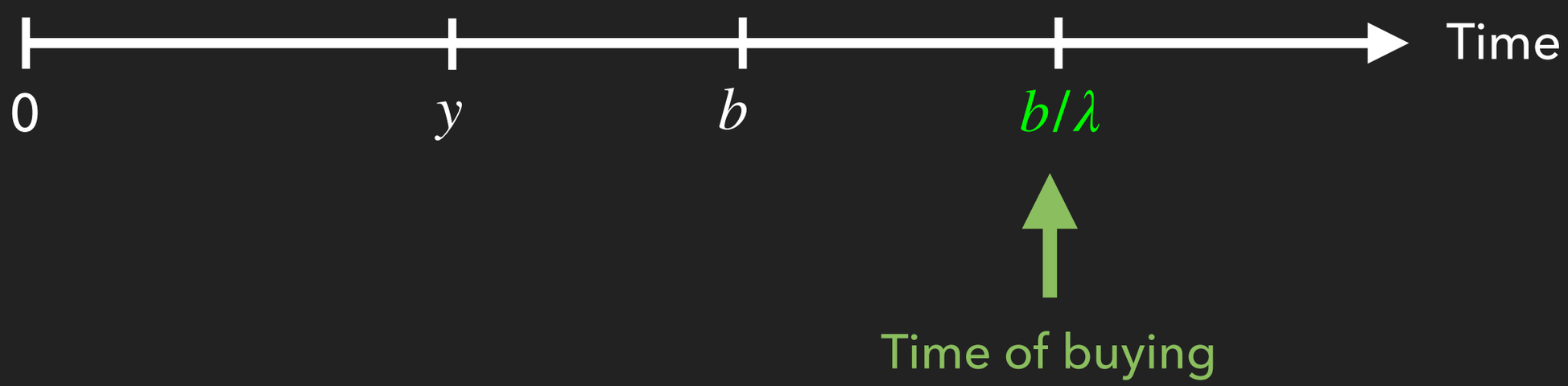


Example 2: Rent or buy

- Given b, but ignoring x, decide when to buy
- Additional input: prediction y of x (with no guarantees)

Algorithm: Follow the prediction… with precaution $\lambda \in [0,1]$

If $y < b$



Example 2: Rent or buy

- Given b, but ignoring x, decide when to buy
- Additional input: prediction y of x (with no guarantees)

Algorithm: Follow the prediction… with precaution $\lambda \in [0,1]$

Theorem: [Kumar, Purohit, Svitkina, NeurIPS ’18]

$$\forall b, x, y: \qquad \mathsf{ALG}(b, x, y) \leq \left(1 + \frac{1}{\lambda}\right) \min(x, b)$$

Robustness

Example 2: Rent or buy

- Given b, but ignoring x, decide when to buy
- Additional input: prediction y of x (with no guarantees)

Algorithm: Follow the prediction… with precaution $\lambda \in [0,1]$

Theorem: [Kumar, Purohit, Svitkina, NeurIPS '18]

$$\forall b, x, y: \quad \mathsf{ALG}(b, x, y) \leq \left(1 + \frac{1}{\lambda}\right) \min(x, b)$$

$$\mathsf{ALG}(b, x, y) \leq (1 + \lambda) \min(x, b) + \frac{|x - y|}{1 - \lambda}$$

Robustness Consistency

Example 2: Rent or buy

- Given b, but ignoring x, decide when to buy
- Additional input: prediction y of x (with no guarantees)

Algorithm: Follow the prediction… with precaution $\lambda \in [0,1]$

Theorem: [Kumar, Purohit, Svitkina, NeurIPS '18]

$$\forall b, x, y: \quad \mathsf{ALG}(b, x, y) \leq \left(1 + \frac{1}{\lambda}\right) \min(x, b)$$

$$\mathsf{ALG}(b, x, y) \leq (1 + \lambda) \min(x, b) + \frac{|x - y|}{1 - \lambda}$$

Robustness Consistency Smoothness

Example 2: Rent or buy

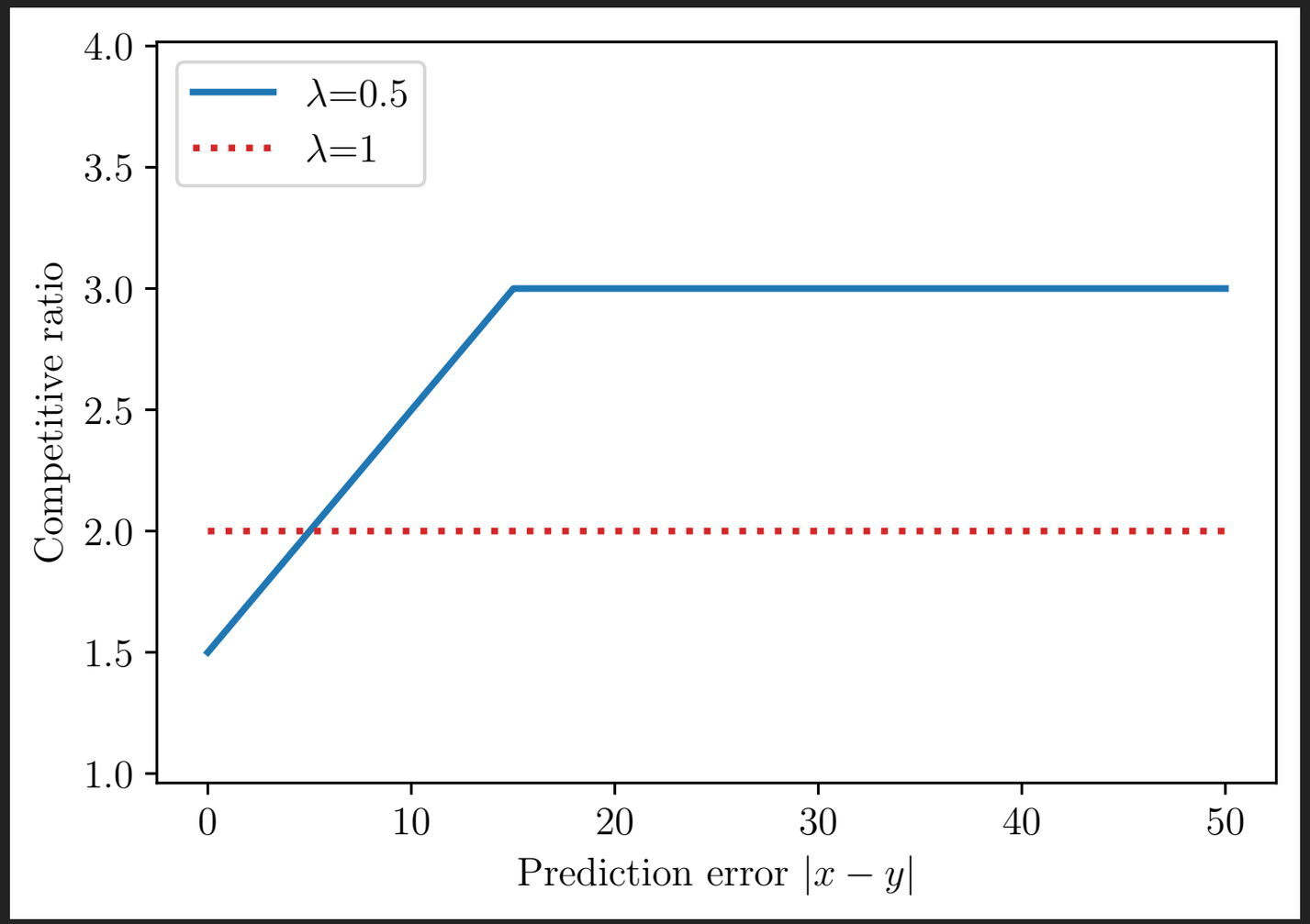
$$\forall b, x, y: \quad \mathsf{ALG}(b, x, y) \leq \left(1 + \frac{1}{\lambda}\right) \min(x, b)$$

$$\mathsf{ALG}(b, x, y) \leq (1 + \lambda) \min(x, b) + \frac{|x - y|}{1 - \lambda}$$

Example 2: Rent or buy

$$\forall b, x, y: \quad \mathsf{ALG}(b, x, y) \leq \left(1 + \frac{1}{\lambda}\right) \min(x, b)$$

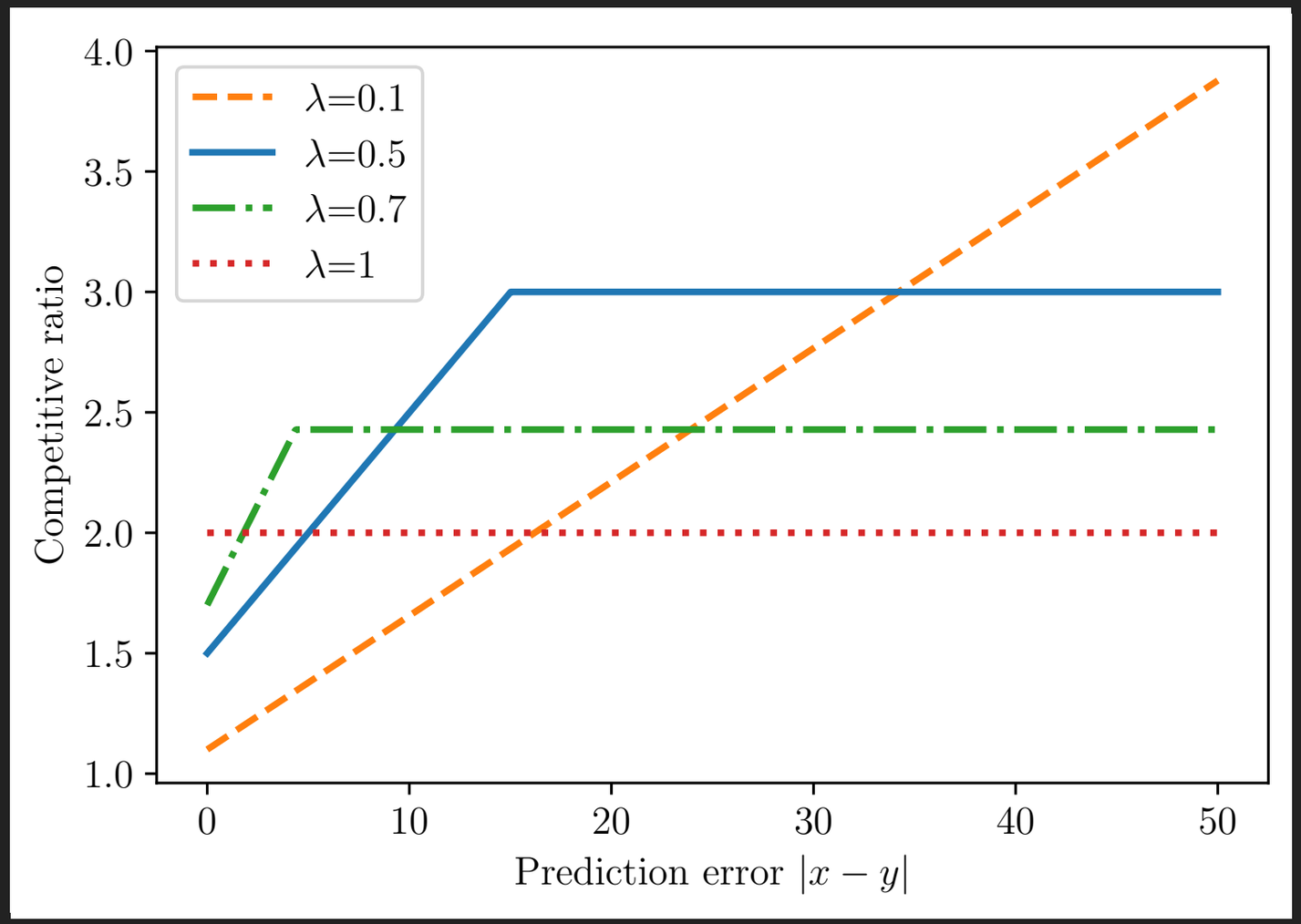
$$\mathsf{ALG}(b, x, y) \leq (1 + \lambda) \min(x, b) + \frac{|x - y|}{1 - \lambda}$$



Example 2: Rent or buy

$$\forall b, x, y: \quad \mathsf{ALG}(b, x, y) \leq \left(1 + \frac{1}{\lambda}\right) \min(x, b)$$

$$\mathsf{ALG}(b, x, y) \leq (1 + \lambda) \min(x, b) + \frac{|x - y|}{1 - \lambda}$$



Optimality of a learning-augmented algorithm?

$$\forall b, x, y: \qquad \mathsf{ALG}(b, x, y) \leq \left(1 + \frac{1}{\lambda}\right) \min(x, b)$$

$$\mathsf{ALG}(b, x, y) \leq (1 + \lambda) \min(x, b) + \frac{|x - y|}{1 - \lambda}$$

Can we find an algorithm having, at the same time, better robustness and better consistency?

Optimality of a learning-augmented algorithm?

$$\forall b, x, y: \qquad \mathsf{ALG}(b, x, y) \leq \left(1 + \frac{1}{\lambda}\right) \min(x, b)$$

$$\mathsf{ALG}(b, x, y) \leq (1 + \lambda) \min(x, b) + \frac{|x - y|}{1 - \lambda}$$

Can we find an algorithm having, at the same time, better robustness and better consistency?

For example: can we find an algorithm satisfying

$$\text{Robustness} = 1 + \frac{1}{\lambda} \qquad \text{Consistency} = 1 + \frac{\lambda}{2} \quad ?$$

Optimality of a learning-augmented algorithm?

$$\forall b, x, y: \qquad \mathsf{ALG}(b, x, y) \leq \left(1 + \frac{1}{\lambda}\right) \min(x, b)$$

$$\mathsf{ALG}(b, x, y) \leq (1 + \lambda) \min(x, b) + \frac{|x - y|}{1 - \lambda}$$

Can we find an algorithm having, at the same time, better robustness and better consistency?

For example: can we find an algorithm satisfying

$$\text{Robustness} = 1 + \frac{1}{\lambda} \qquad \text{Consistency} = 1 + \frac{\lambda}{2} \quad ?$$

Answer: No! The robustness of $1 + \frac{1}{\lambda}$ and consistency $1 + \lambda$ are optimal

Problems with multiple unknown parameters

Some problems involve multiple unknown variables $x_1, \ldots, x_n$

Problems with multiple unknown parameters

Some problems involve multiple unknown variables $x_1, \ldots, x_n$

$\Longrightarrow$ We need multiple predictions $y_1, \ldots, y_n$

Problems with multiple unknown parameters

Some problems involve multiple unknown variables $x_1, \ldots, x_n$

$\Longrightarrow$ We need multiple predictions $y_1, \ldots, y_n$

In many real-world scenarios

- We only have access to some predictions $y_1, \ldots, y_B$ with $B < n$
- We can decide when to query predictions

The scheduling problem

Schedule n tasks on a machine, minimizing the sum of the completion times.

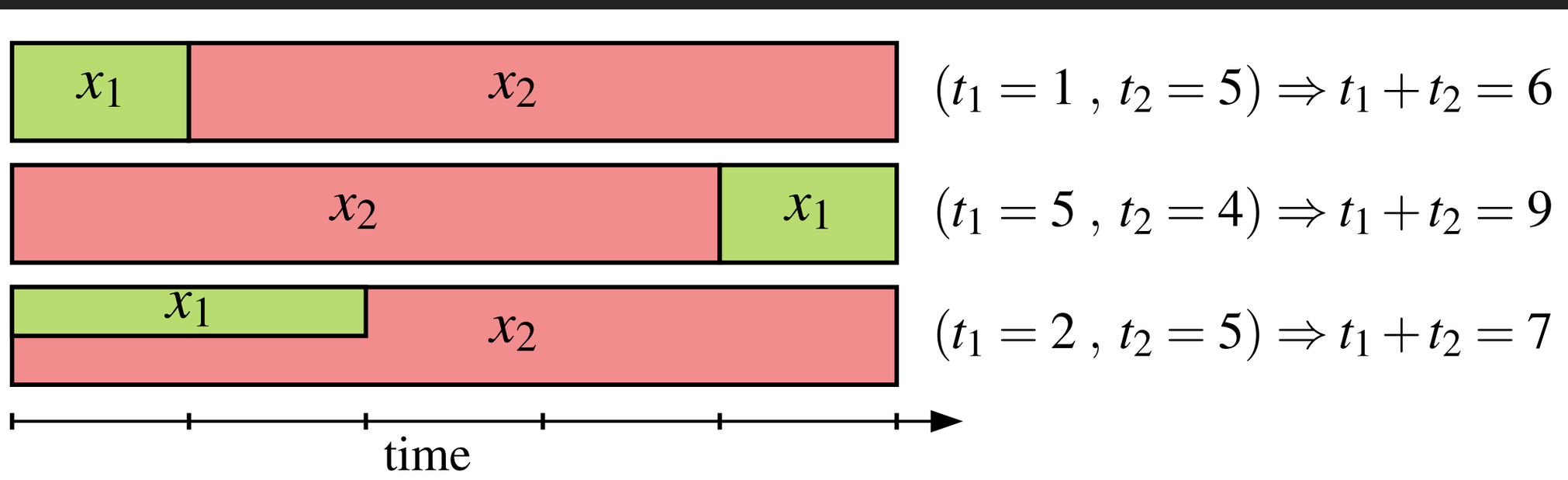
$\forall i \in \{1,\ldots,n\}:$ x_i = time for executing task i (unknown)

The scheduling problem

Schedule n tasks on a machine, minimizing the sum of the completion times.

$\forall i \in \{1,\ldots,n\}$: x_i = time for executing task i (unknown)

Example: $x_1 = 1$, $x_2 = 5$



The scheduling problem

Schedule n tasks on a machine, minimizing the sum of the completion times.

$\forall i \in \{1,\ldots,n\}$: x_i = time for executing task i (unknown)

If $x_1, \ldots, x_n$ are known, then the optimal algorithm is

OPT: is to run the shortest tasks first

The scheduling problem

Schedule n tasks on a machine, minimizing the sum of the completion times.

$\forall i \in \{1,\ldots,n\}$: x_i = time for executing task i (unknown)

If $x_1, \ldots, x_n$ are known, then the optimal algorithm is

OPT: is to run the shortest tasks first

If $x_1, \ldots, x_n$ are unknown, then the optimal algorithm is “round robin”

RR: is to run the shortest tasks first

The scheduling problem

Schedule n tasks on a machine, minimizing the sum of the completion times.

$\forall i \in \{1,\ldots,n\}$: x_i = time for executing task i (unknown)

If $x_1, \ldots, x_n$ are known, then the optimal algorithm is

OPT: is to run the shortest tasks first

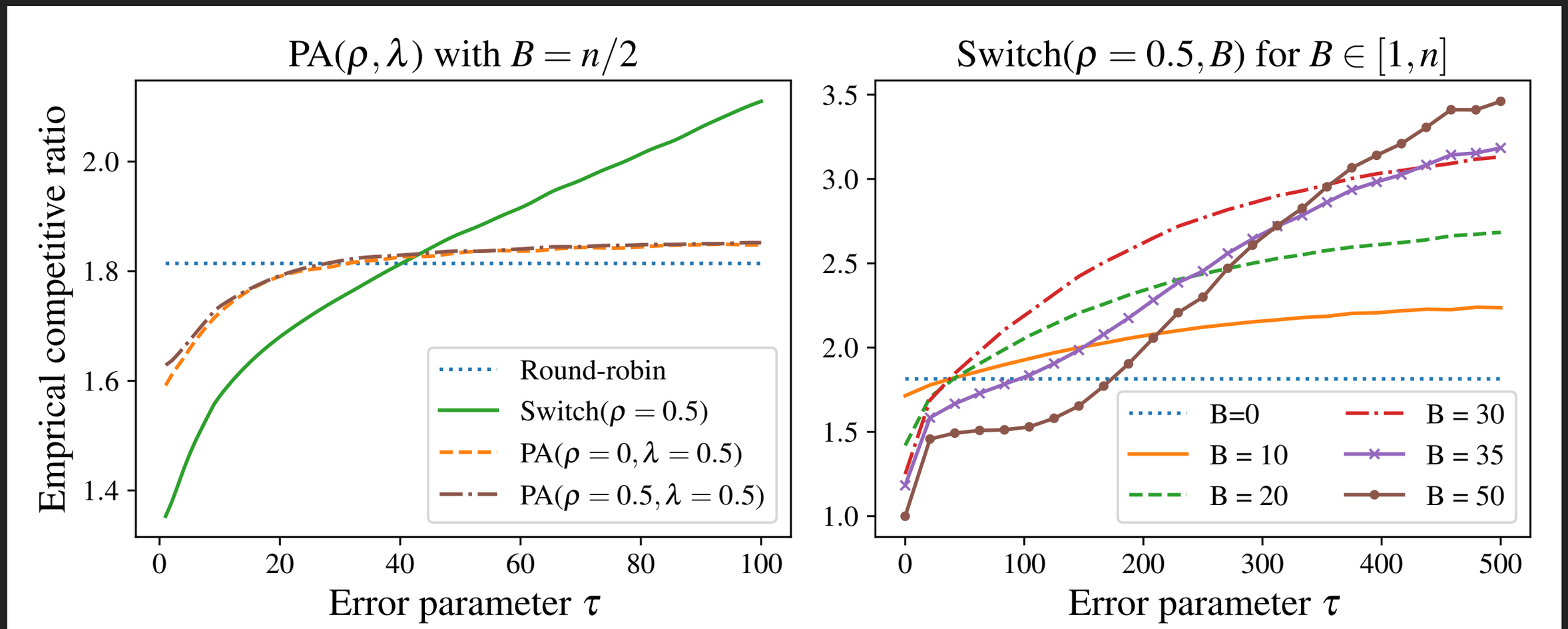
If $x_1, \ldots, x_n$ are unknown, then the optimal algorithm is "round robin"

RR: is to run the shortest tasks first

Theorem: $\mathrm{RR}(x_1, \ldots, x_n) \leq 2\mathrm{OPT}(x_1, \ldots, x_n)$

Learning-augmented scheduling

- All the predictions are available [Kumar, Purohit, Svitkina, NeurIPS '18]
- Only B predictions are available [Benomar and Perchet, ICML '24]



Learning-augmented Priority queues

[Benomar and Coester, NeurIPS ‘24]

- Insert(item, key)

- ExtractMin()

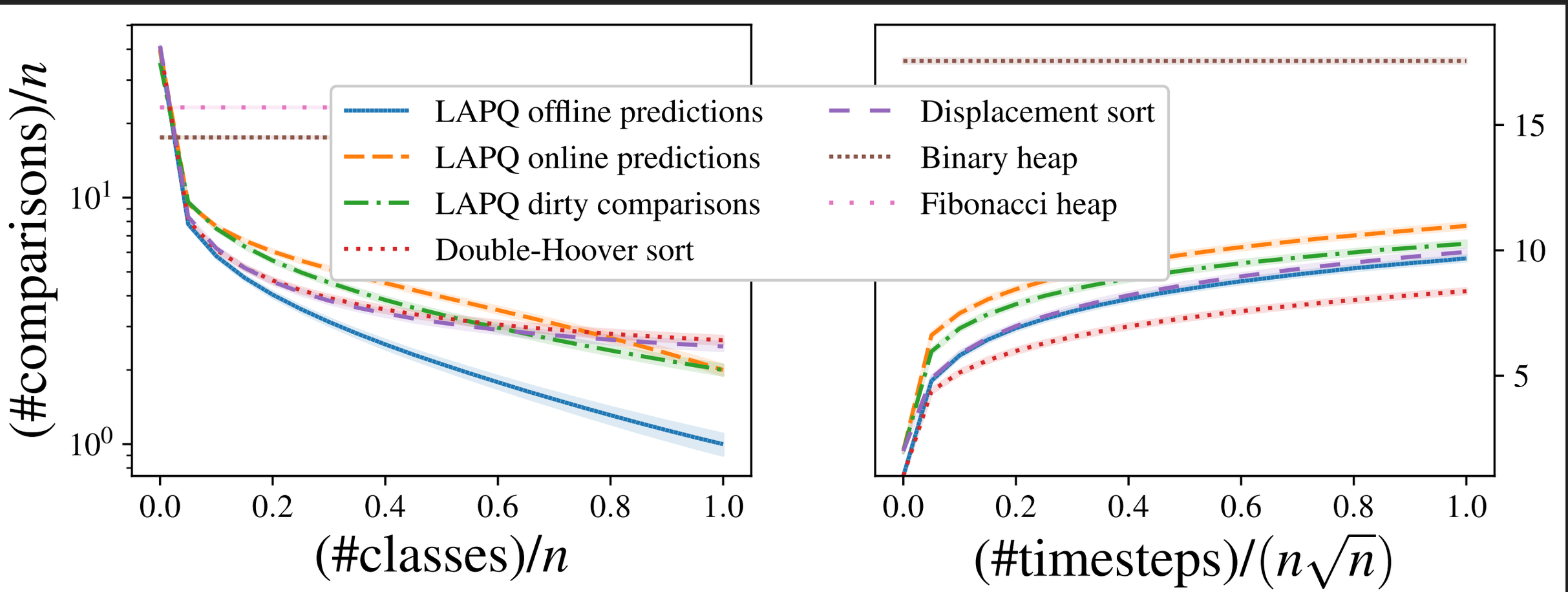
- FindMin()

- DecreaseKey(item, key)

Applications: Sorting, shortest path (Dijkstra), minimum spanning trees, scheduling/load balancing, networks, hierarchical clustering, …

Learning-augmented Priority queues

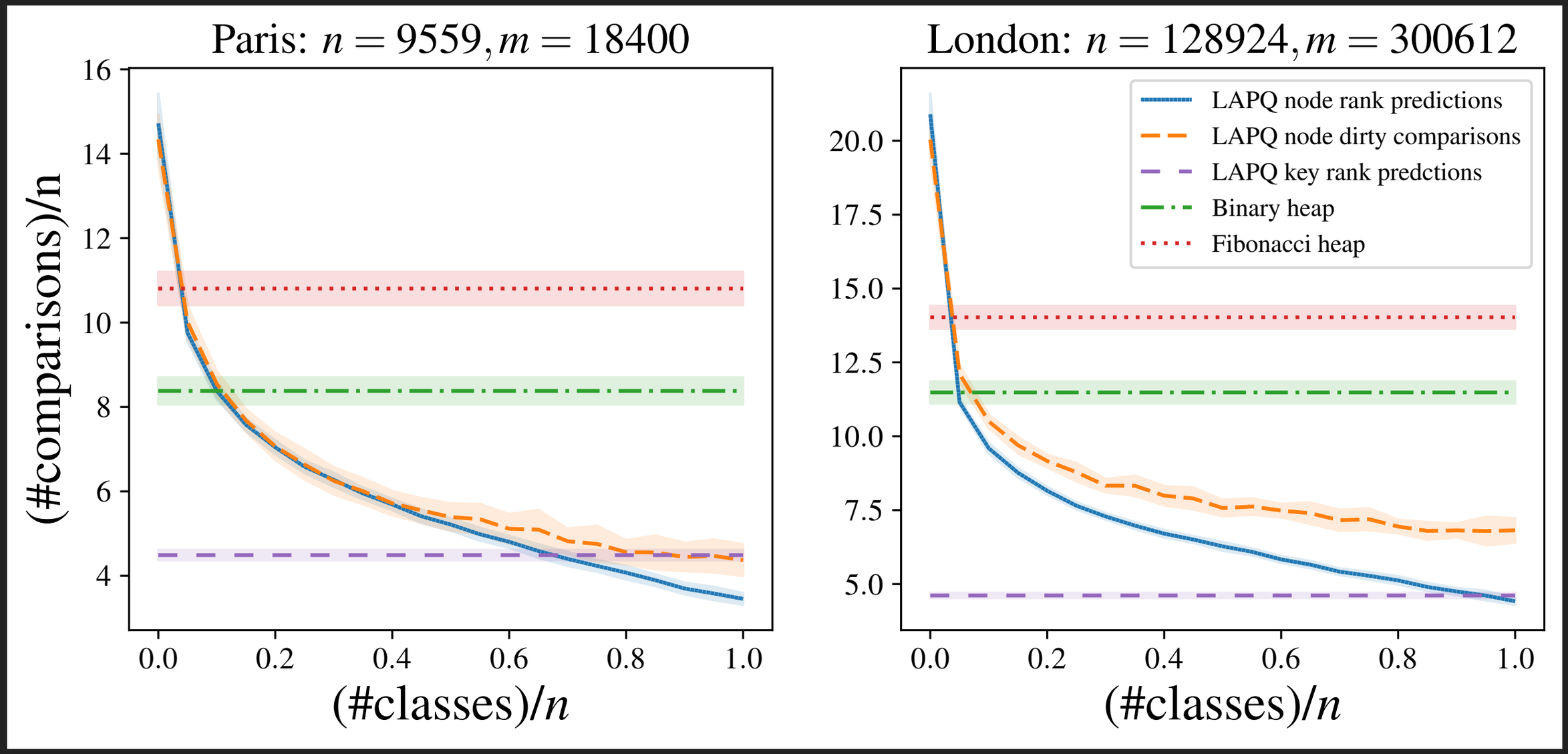
Application in sorting



n=100,000

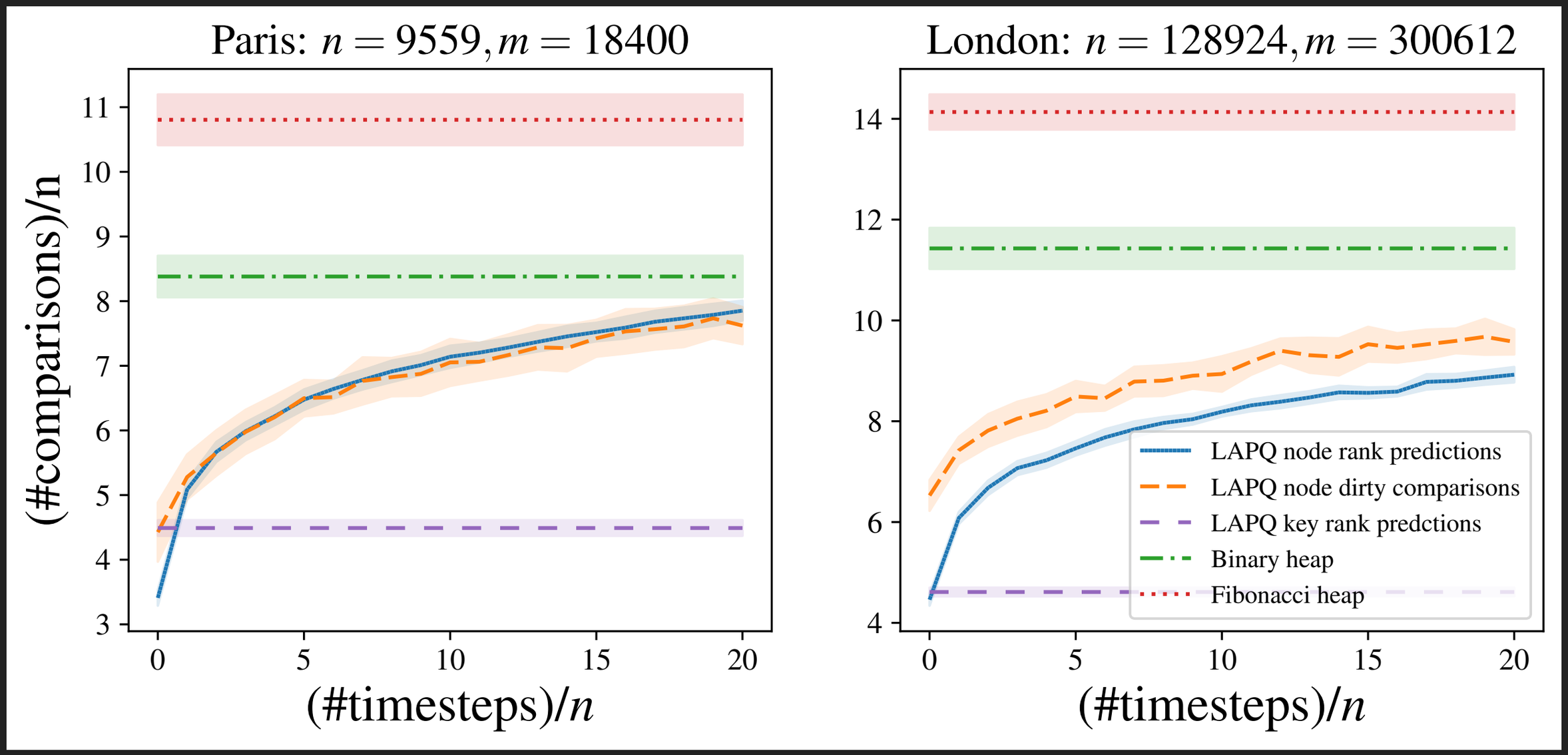
Learning-augmented Priority queues

Application in Dijkstra's algorithm (shortest path in a graph)



Learning-augmented Priority queues

Application in Dijkstra's algorithm (shortest path in a graph)



Final remarks

- The setting of “learning-augmented algorithms” is very generic and can be applied to multiple problems
- Brings together ML and TCS researchers
- Real implementations are still very limited
- List of papers: https://algorithms-with-predictions.github.io/