max planck institut
informatik

Towards a Complexity Theory for Randomized Search Heuristics: The Ranking-Based Black-Box Model

Benjamin Doerr / Carola Winzen

CSR, June 14, 2011

Our Motivation

- General-purpose **(randomized) search heuristics** are
  - ...easy to implement,
  - ...very flexible,
  - ...need little a priori knowledge about problem at hand,
  - ...can deal with many constraints and nonlinearities,

Our Motivation

- General-purpose **(randomized) search heuristics** are
  - ...easy to implement,
  - ...very flexible,
  - ...need little a priori knowledge about problem at hand,
  - ...can deal with many constraints and nonlinearities,
- and thus, frequently applied in practice.
- Some theoretical results exist.
- Typical result: runtime analysis for
  - a particular algorithm
  - on a particular problem.

Our Motivation

- General-purpose **(randomized) search heuristics** are
  - ...easy to implement,
  - ...very flexible,
  - ...need little a priori knowledge about problem at hand,
  - ...can deal with many constraints and nonlinearities,
- and thus, frequently applied in practice.
- Some theoretical results exist.
- Typical result: runtime analysis for
  - a particular algorithm
  - on a particular problem.

Comparable to the "early days" of Computer Science

Our Motivation

- In classical theoretical computer science:
  - first results: runtime analysis for
    - a particular algorithm
    - on a particular problem.

Our Motivation

- In classical theoretical computer science:
  - first results: runtime analysis for
    - a particular algorithm
    - on a particular problem.
  - general lower bounds (“tractability of a problem”)
  - complexity theory

Our Motivation

- In classical theoretical computer science:
  - first results: runtime analysis for
    - a particular algorithm
    - on a particular problem.
  - general lower bounds (“tractability of a problem”)
  - complexity theory

How to create a complexity theory for randomized search heuristics?

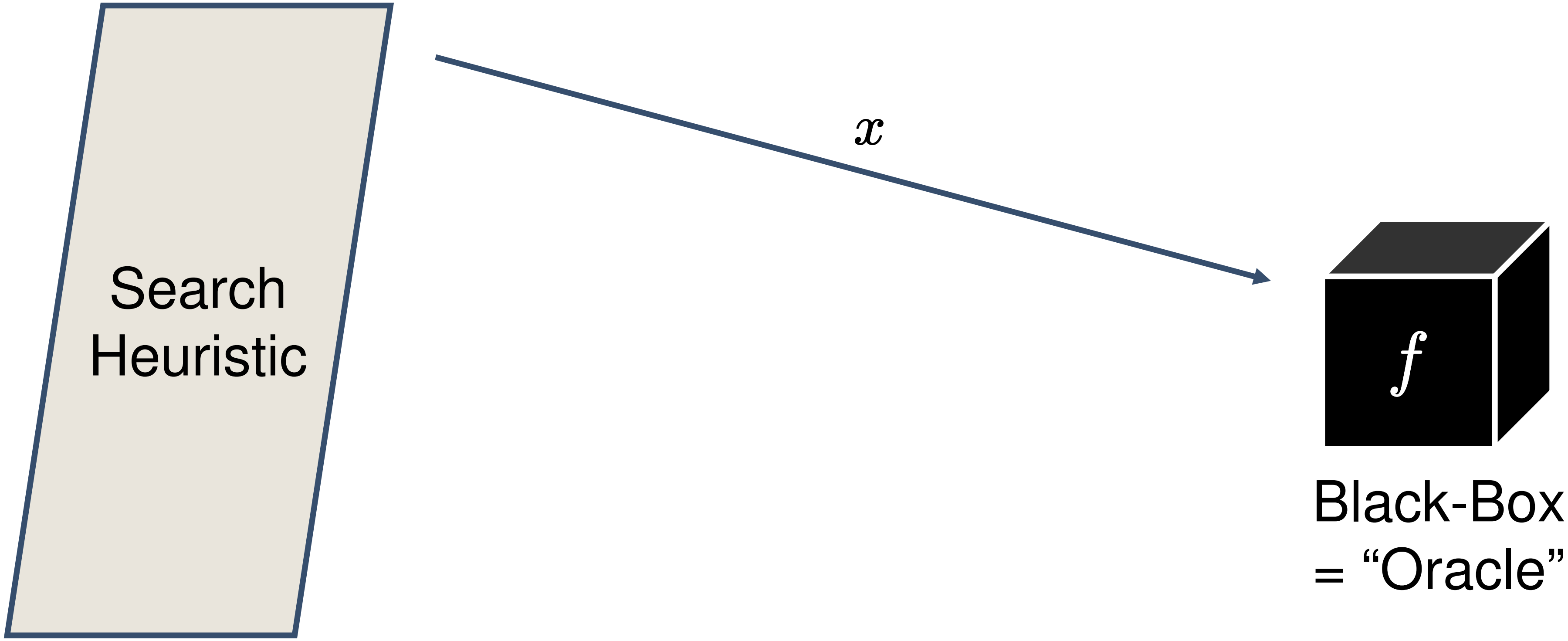
Our Motivation

- In classical theoretical computer science:
  - first results: runtime analysis for
    - a particular algorithm
    - on a particular problem.
  - general lower bounds ("tractability of a problem")
  - complexity theory
- **Our aim:** to understand the **tractability of a problem** for **general-purpose (randomized) search heuristics**

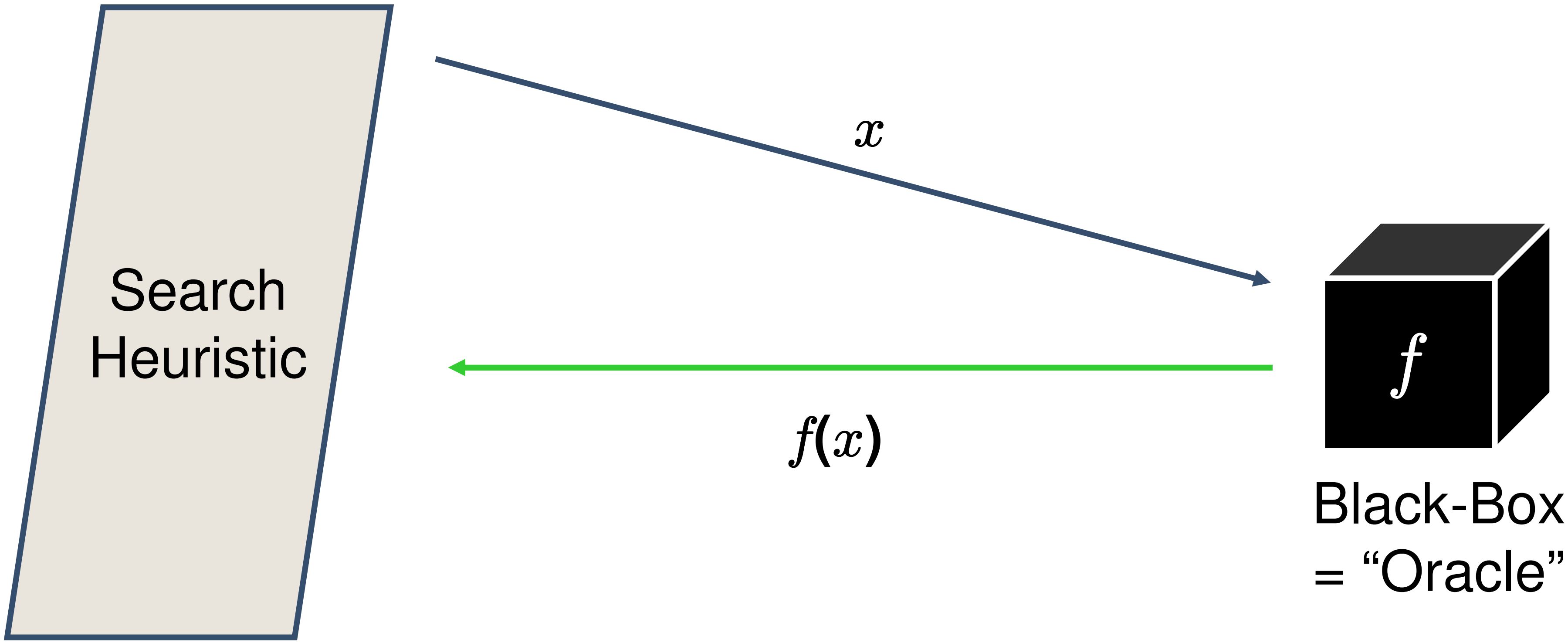
"Towards a Complexity Theory for
Randomized Search Heuristics"

A General View on Search Heuristics

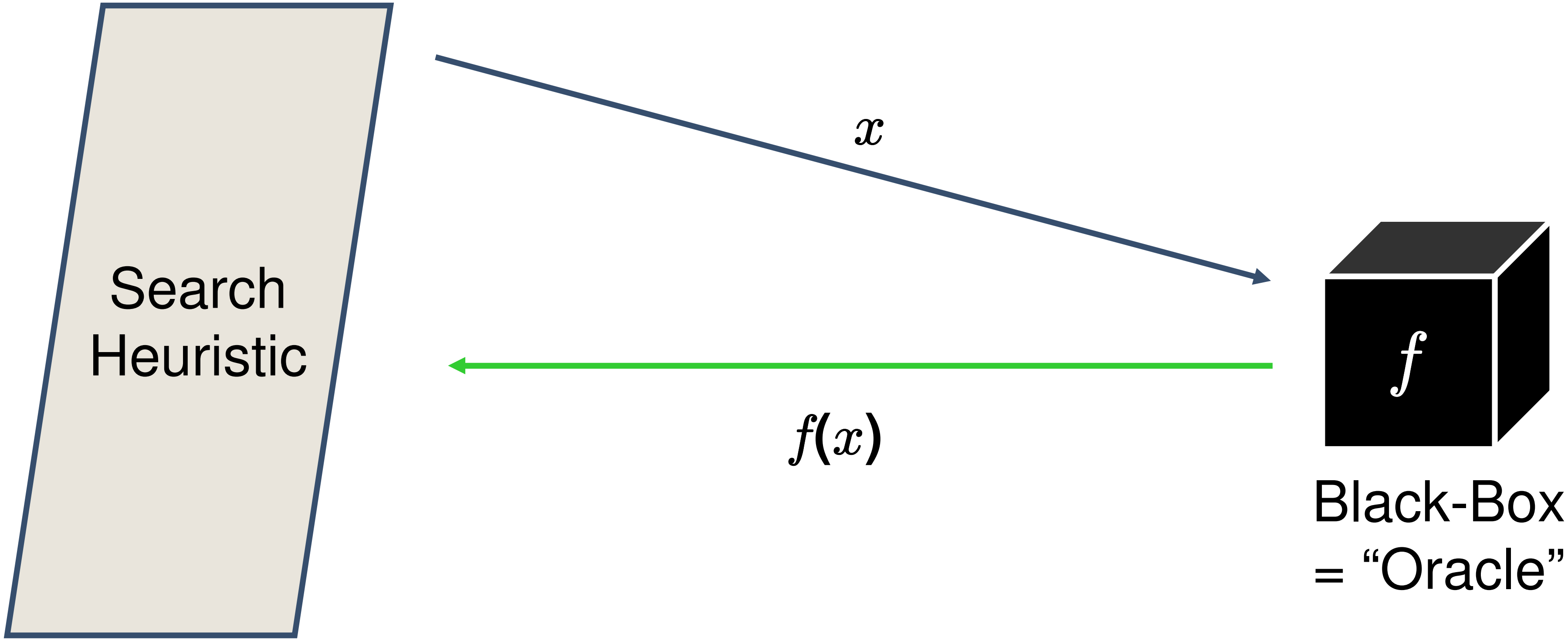
A General View on Search Heuristics



A General View on Search Heuristics

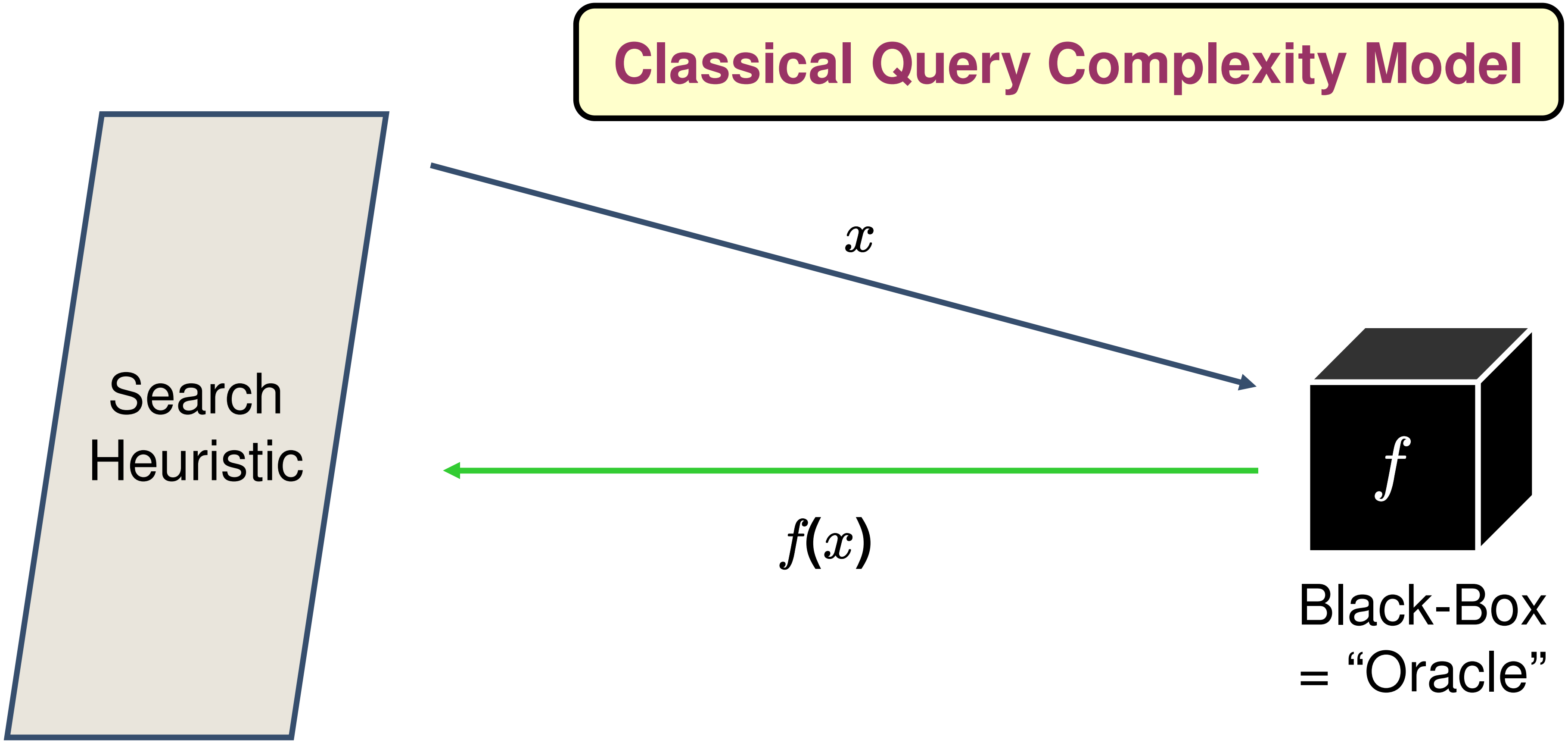


A General View on Search Heuristics



Typical cost measure: **number of function evaluations** until an **optimal solution** is queried for the first time

A General View on Search Heuristics



Typical cost measure: **number of function evaluations** until an **optimal solution** is queried for the first time

A Theory for (Randomized) Search Heuristics

- Part 1: **Classical query complexity model**
  - Game theoretic view
  - Example: Mastermind

- Part 2: Refinement: **ranking-based query complexity**

"Towards a Complexity Theory for
Randomized Search Heuristics:
The Ranking-Based Black-Box Model"

Example: A Mastermind Problem

- Carole (=oracle) chooses a binary string of length n:

1	1	0	1	0	0	1	1

Example: A Mastermind Problem

- Carole (=oracle) chooses a binary string of length n:

1	1	0	1	0	0	1	1

- Paul (=algo.) tries to find it. He may ask any string of length n:

Example: A Mastermind Problem

- Carole (=oracle) chooses a binary string of length n:

1	1	0	1	0	0	1	1

- Paul (=algo.) tries to find it. He may ask any string of length n:

1	0	1	0	1	0	1	0

Example: A Mastermind Problem

- Carole (=oracle) chooses a binary string of length n:

1	1	0	1	0	0	1	1

- Paul (=algo.) tries to find it. He may ask any string of length n:

1	0	1	0	1	0	1	0

- Carole computes in how many positions the strings coincide:

Example: A Mastermind Problem

- Carole (=oracle) chooses a binary string of length n:

1	1	0	1	0	0	1	1

- Paul (=algo.) tries to find it. He may ask any string of length n:

1	0	1	0	1	0	1	0

- Carole computes in how many positions the strings coincide:

1	0	1	0	1	0	1	0

Example: A Mastermind Problem

- Carole (=oracle) chooses a binary string of length n:

1	1	0	1	0	0	1	1

- Paul (=algo.) tries to find it. He may ask any string of length n:

1	0	1	0	1	0	1	0

- Carole computes in how many positions the strings coincide:

1	0	1	0	1	0	1	0

- "Paul, our strings coincide in 3 bits"

Example: A Mastermind Problem

- Carole (=oracle) chooses a binary string of length n:

1	1	0	1	0	0	1	1

- Paul (=algo.) tries to find it. He may ask any string of length n:

1	0	1	0	1	0	1	0

- Carole computes in how many positions the strings coincide:

1	0	1	0	1	0	1	0

- "Paul, our strings coincide in 3 bits"

We say that the **"fitness"** of Paul's string is 3

max planck institut informatik

Example: A Mastermind Problem

- Carole chooses a binary string of length n:

1	1	0	1	0	0	1	1

- Paul tries to find it. He may ask any string of length n:

1	0	1	0	1	0	1	0	**3**

Example: A Mastermind Problem

- Carole chooses a binary string of length n:

1	1	0	1	0	0	1	1

- Paul tries to find it. He may ask any string of length n:

1	0	1	0	1	0	1	0	**3**
1	0	1	1	0	1	0	1	

Example: A Mastermind Problem

- Carole chooses a binary string of length n:

1	1	0	1	0	0	1	1

- Paul tries to find it. He may ask any string of length n:

1	0	1	0	1	0	1	0	**3**
1	0	1	1	0	1	0	1	

Example: A Mastermind Problem

- Carole chooses a binary string of length n:

1	1	0	1	0	0	1	1

- Paul tries to find it. He may ask any string of length n:

1	0	1	0	1	0	1	0	**3**
1	0	1	1	0	1	0	1	**4**

...

Example: A Mastermind Problem

- Carole chooses a binary string of length n:

1	1	0	1	0	0	1	1

- Paul tries to find it. He may ask any string of length n:

1	0	1	0	1	0	1	0	**3**
1	0	1	1	0	1	0	1	**4**

...

How many queries does Paul need, on average, until he has identified Carole's string?

Reminder

- Our aim: To understand **tractability of a problem** for general-purpose (randomized) search heuristics
- Measure: **number of function evaluations** until an optimal solution is queried for the first time
- Our main interest: **good lower bounds**

The Master Mind Problem: What Search Heuristics Do

- Paul tries to find Carole's binary string of length n:

1	1	0	1	0	0	1	1

The Master Mind Problem: What Search Heuristics Do

- Paul tries to find Carole's binary string of length n:

1	1	0	1	0	0	1	1

- First query is arbitrary:

1	0	1	0	1	0	1	0

The Master Mind Problem: What Search Heuristics Do

- Paul tries to find Carole's binary string of length n:

1	1	0	1	0	0	1	1

- First query is arbitrary:

1	0	1	0	1	0	1	0	**3**

The Master Mind Problem: What Search Heuristics Do

- Paul tries to find Carole's binary string of length n:

1	1	0	1	0	0	1	1

- First query is arbitrary:

1	0	1	0	1	0	1	0	**3**

- Then flip *exactly* one bit (chosen u.a.r.):

1	1	1	0	1	0	1	0

The Master Mind Problem: What Search Heuristics Do

- Paul tries to find Carole's binary string of length n:

1	1	0	1	0	0	1	1

- First query is arbitrary:

1	0	1	0	1	0	1	0	**3**

- Then flip *exactly* one bit (chosen u.a.r.):

1	1	1	0	1	0	1	0	**4**

The Master Mind Problem: What Search Heuristics Do

- Paul tries to find Carole's binary string of length n:

1	1	0	1	0	0	1	1

- First query is arbitrary:

1	0	1	0	1	0	1	0	**3**

- Then flip *exactly* one bit (chosen u.a.r.):

1	1	1	0	1	0	1	0	**4**

- And it continues with the better of the two:

1	1	1	0	1	1	1	0

The Master Mind Problem: What Search Heuristics Do

- Paul tries to find Carole's binary string of length n:

1	1	0	1	0	0	1	1

- First query is arbitrary:

1	0	1	0	1	0	1	0	**3**

- Then flip *exactly* one bit (chosen u.a.r.):

1	1	1	0	1	0	1	0	**4**

Random Local Search algorithm:

$\Theta(n \log n)$ Coupon Collector [Folklore]

The Master Mind Problem: What Search Heuristics Do

- Paul tries to find Carole's binary string of length n:

1	1	0	1	0	0	1	1

- First query is arbitrary:

1	0	1	0	1	0	1	0

The Master Mind Problem: What Search Heuristics Do

- Paul tries to find Carole's binary string of length n:

1	1	0	1	0	0	1	1

- First query is arbitrary:

1	0	1	0	1	0	1	0	**3**

- Flip *each bit with probability* $1/n$:

The Master Mind Problem: What Search Heuristics Do

- Paul tries to find Carole's binary string of length n:

1	1	0	1	0	0	1	1

- First query is arbitrary:

1	0	1	0	1	0	1	0	**3**

- Flip *each bit with probability* $1/n$:

0	0	1	0	1	**1**	1	0	**1**

The Master Mind Problem: What Search Heuristics Do

- Paul tries to find Carole's binary string of length n:

1	1	0	1	0	0	1	1

- First query is arbitrary:

1	0	1	0	1	0	1	0	**3**

- Flip *each bit with probability* $1/n$:

0	0	1	0	1	**1**	1	0	**1**

- And it continues with the better of the two

The Master Mind Problem: What Search Heuristics Do

- Paul tries to find Carole's binary string of length n:

1	1	0	1	0	0	1	1

- First query is arbitrary:

1	0	1	0	1	0	1	0	**3**

- Flip *each bit with probability* $1/n$:

0	0	1	0	1	**1**	1	0	**1**

(1+1) Evolutionary Algorithm:

$\Theta(n \log n)$

[Mühlenbein 92]
[Droste/Jansen/Wegener 02]

The Master Mind Problem: Optimal Strategies (1/2)

- Paul tries to find Carole's binary string of length n:

1	0	0	1	0	0	1	1

The Master Mind Problem: Optimal Strategies (1/2)

- Paul tries to find Carole's binary string of length n:

1	0	0	1	0	0	1	1

- He can go through the string bit by bit:

0	0	0	0	0	0	0	0	4

The Master Mind Problem: Optimal Strategies (1/2)

- Paul tries to find Carole's binary string of length n:

1	0	0	1	0	0	1	1

- He can go through the string bit by bit:

0	0	0	0	0	0	0	0	4
1	0	0	0	0	0	0	0	5

The Master Mind Problem: Optimal Strategies (1/2)

- Paul tries to find Carole's binary string of length n:

1	0	0	1	0	0	1	1

- He can go through the string bit by bit:

0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**4**
1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**5**
1	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**4**

The Master Mind Problem: Optimal Strategies (1/2)

- Paul tries to find Carole's binary string of length n:

1	0	0	1	0	0	1	1

- He can go through the string bit by bit:

0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**4**
1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**5**
1	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**4**

...

1	0	0	1	0	0	1	1	**8**

The Master Mind Problem: Optimal Strategies (1/2)

- Paul tries to find Carole's binary string of length n:

1	0	0	1	0	0	1	1

- He can go through the string bit by bit:

0	0	0	0	0	0	0	0	**4**
1	0	0	0	0	0	0	0	**5**
1	1	0	0	0	0	0	0	**4**
...								
1	0	0	1	0	0	1	1	**8**

$\Rightarrow O(n)$

The Master Mind Problem: Optimal Strategies (1/2)

- Paul tries to find Carole's binary string of length n:

1	0	0	1	0	0	1	1

- He can go through the string bit by bit:

0	0	0	0	0	0	0	0	**4**
1	0	0	0	0	0	0	0	**5**
1	1	0	0	0	0	0	0	**4**

...

1	0	0	1	0	0	1

Can we do even better?

$O(n)$

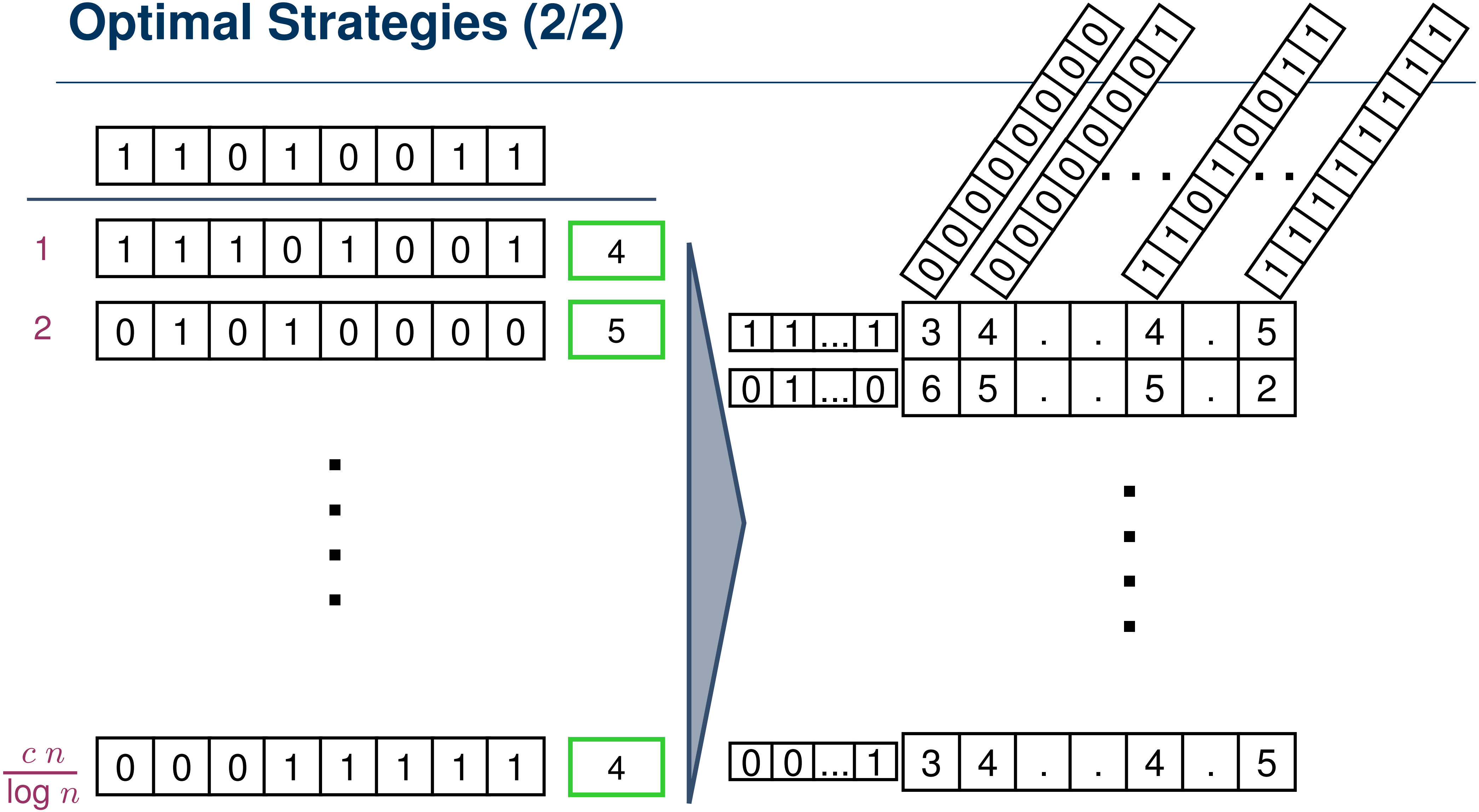
The Master Mind Problem: Optimal Strategies (2/2)

1	1	0	1	0	0	1	1

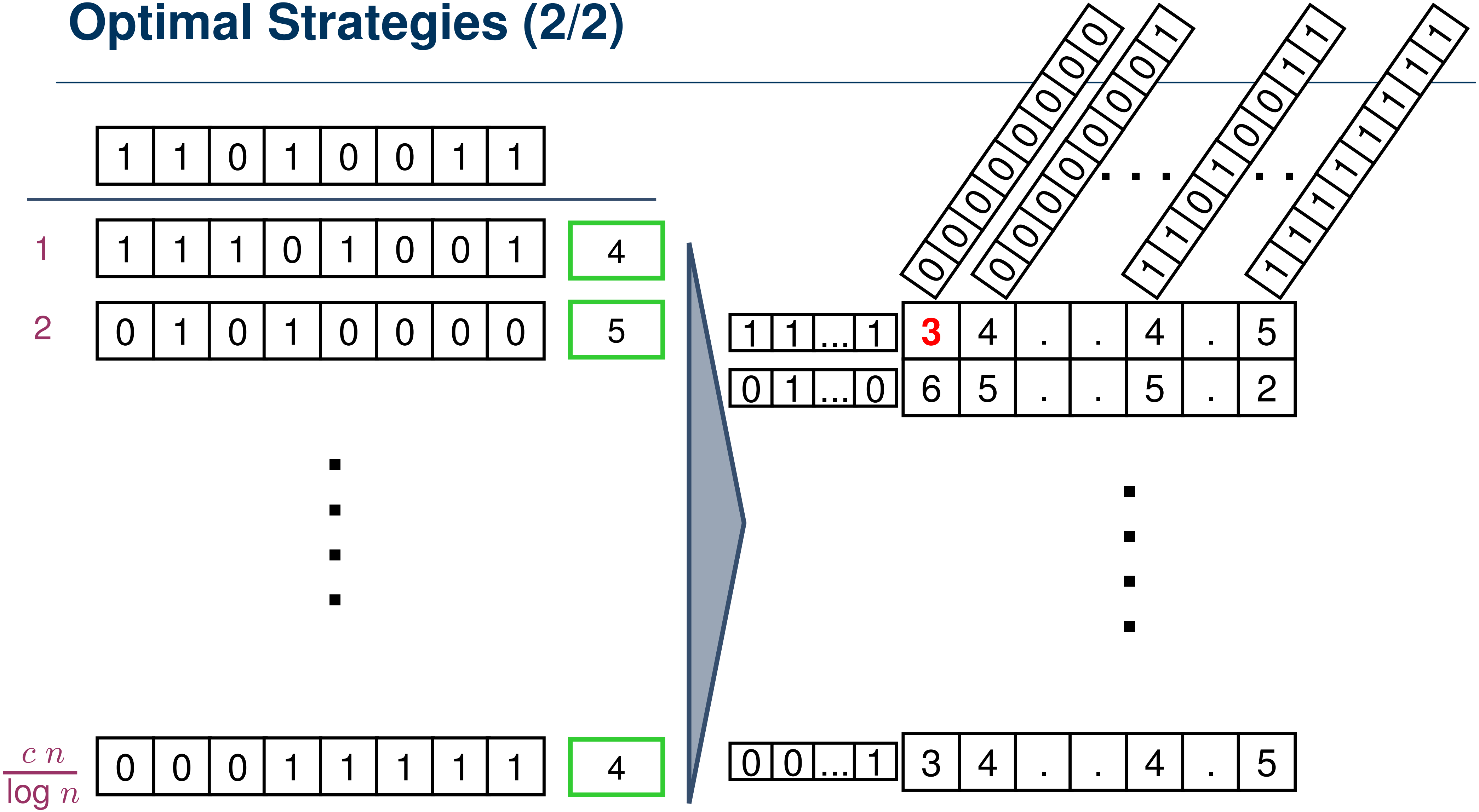
The Master Mind Problem: Optimal Strategies (2/2)

	1	1	0	1	0	0	1	1	
1	1	1	1	0	1	0	0	1	4
2	0	1	0	1	0	0	0	0	5
				⋮					
$\frac{c\,n}{\log n}$	0	0	0	1	1	1	1	1	4

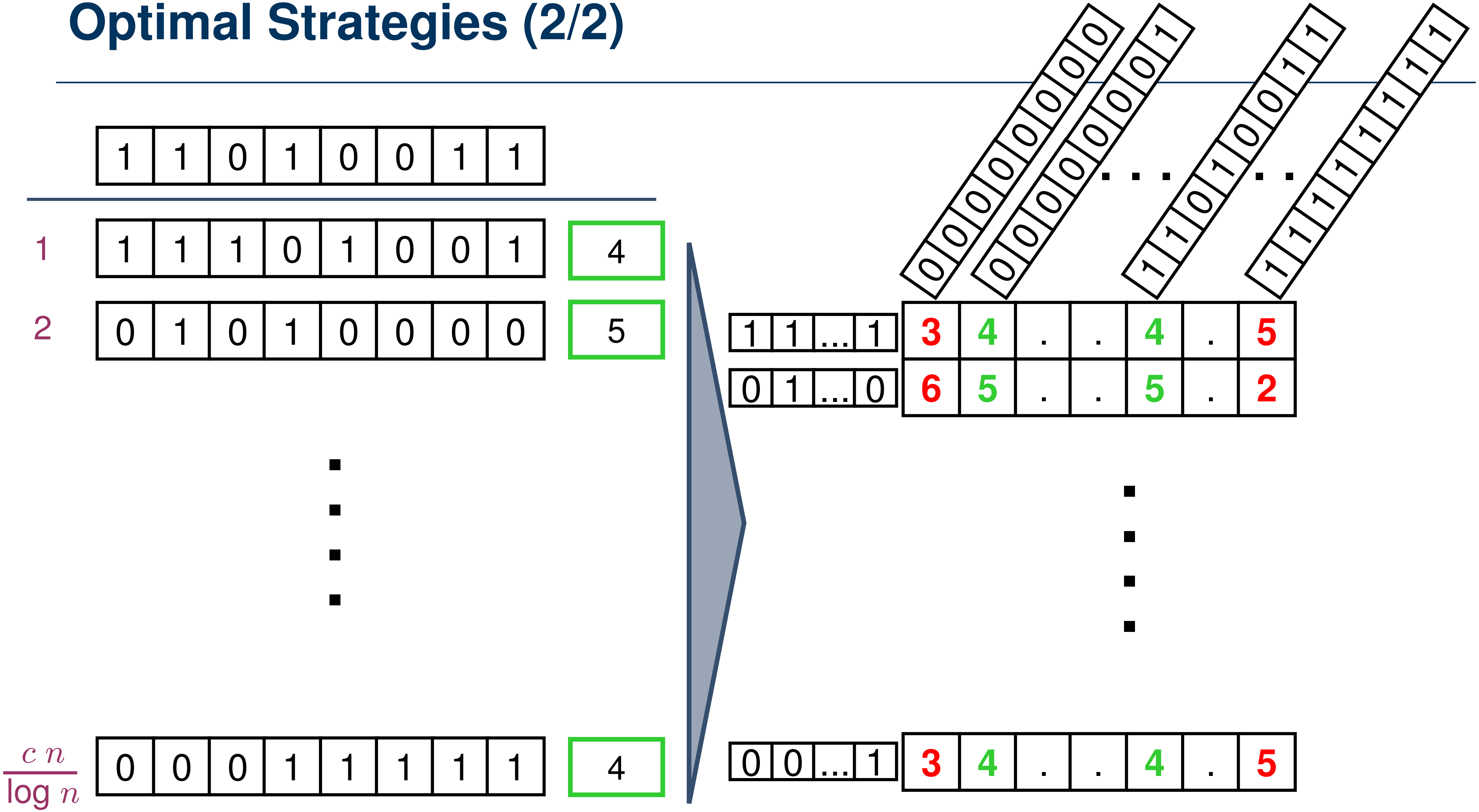
Optimal Strategies (2/2)



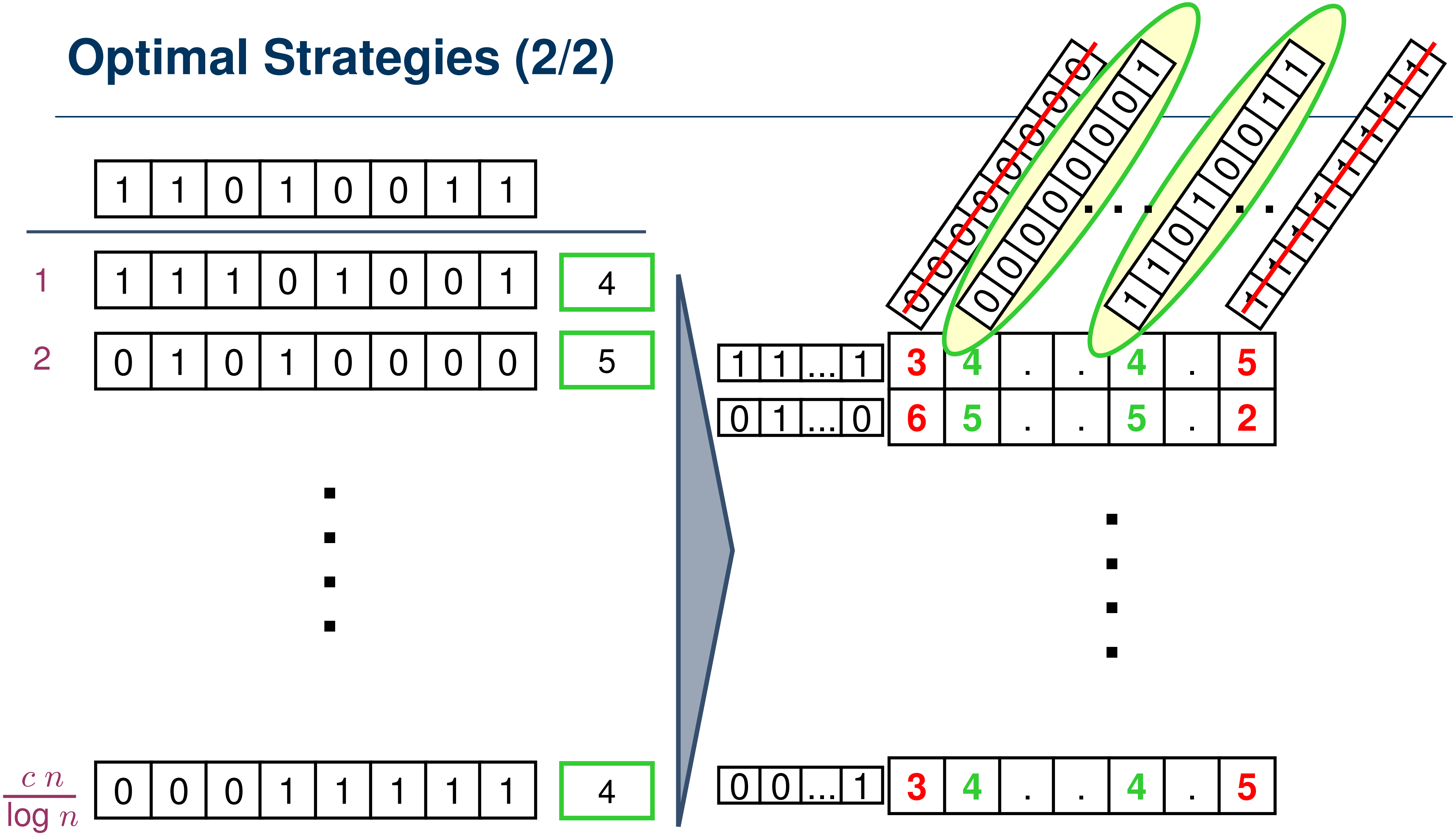
Optimal Strategies (2/2)



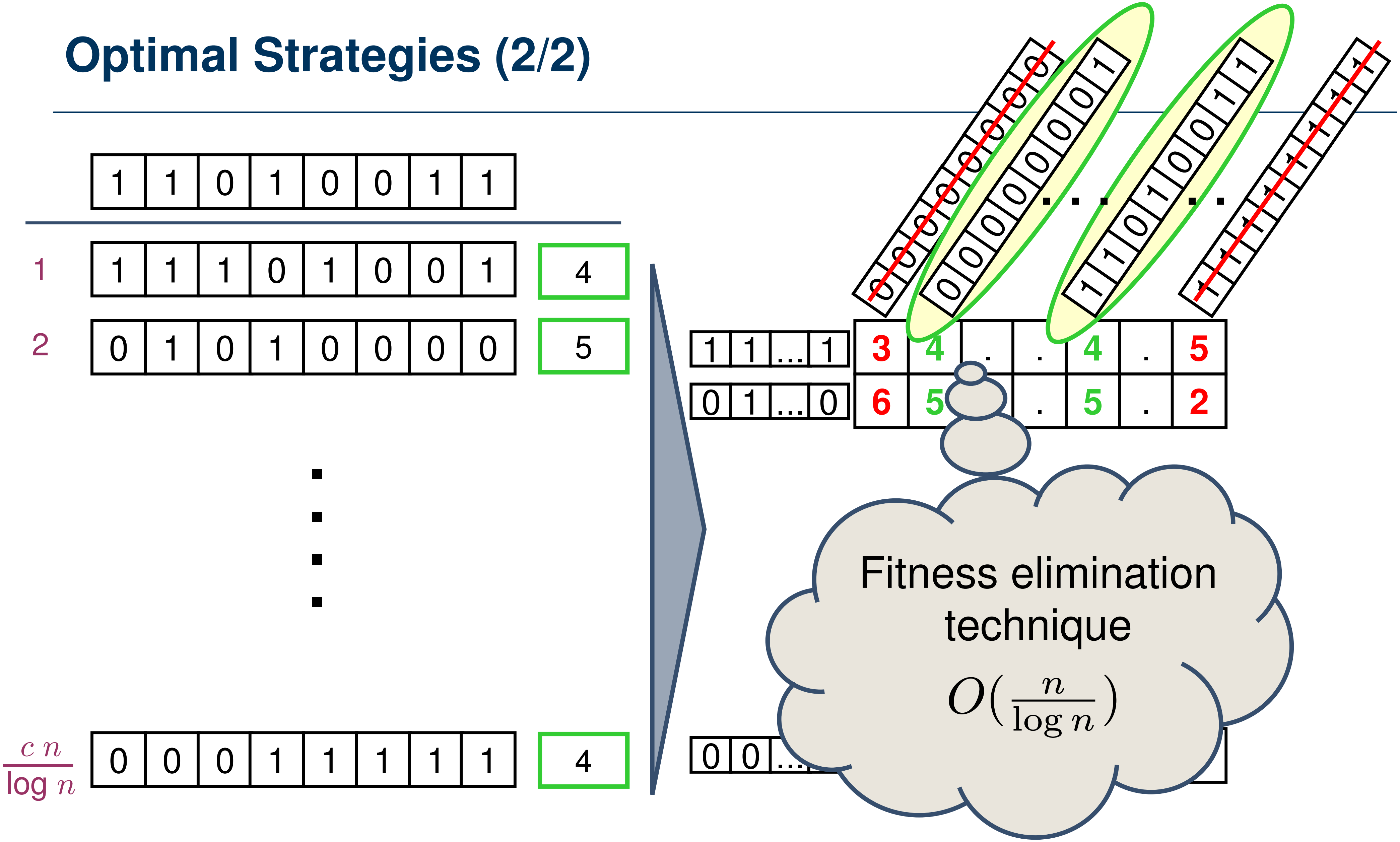
Optimal Strategies (2/2)



Optimal Strategies (2/2)

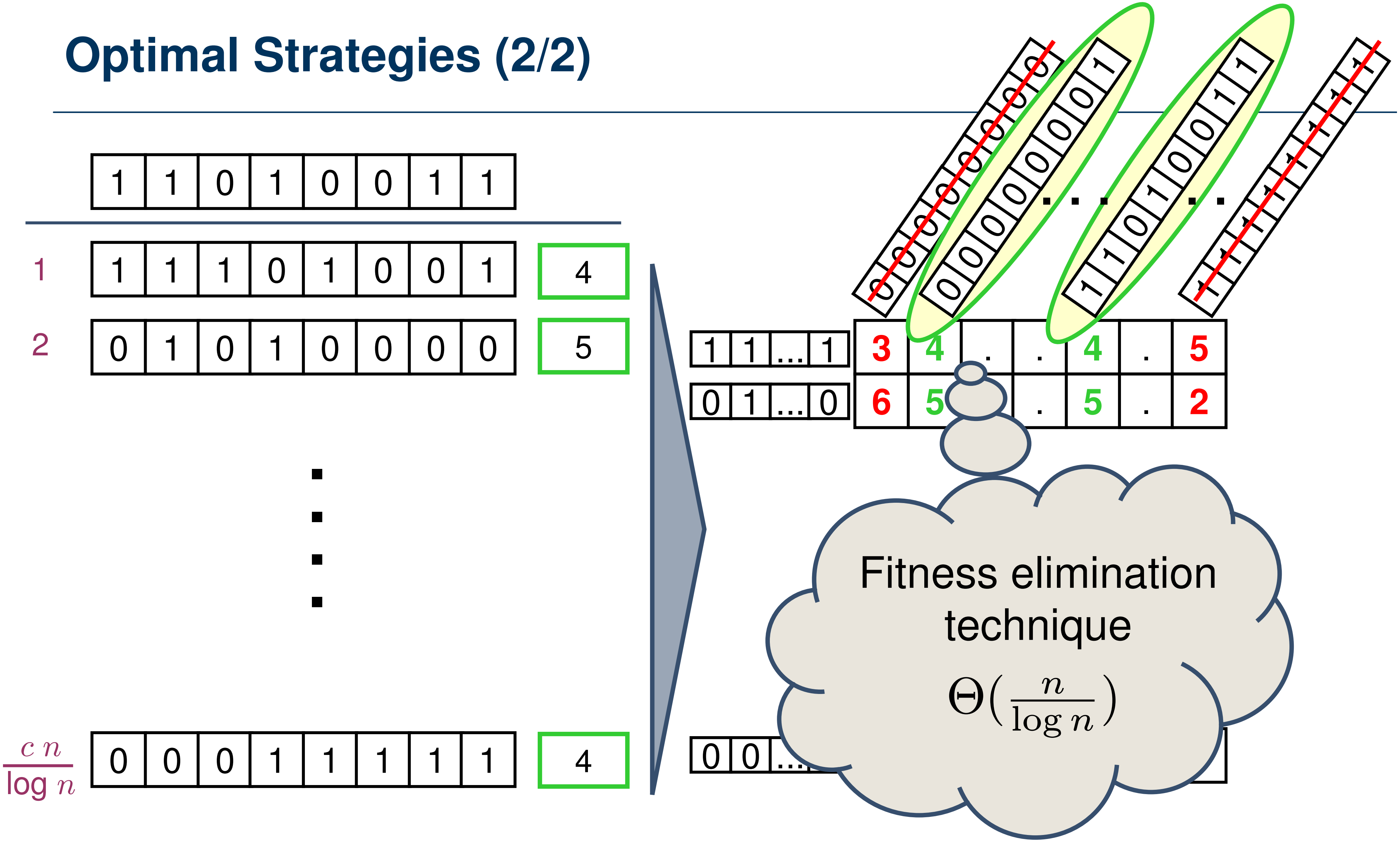


Optimal Strategies (2/2)



[Anil/Wiegand 09], see also [D./Johannsen/Kötzing/Lehre/Wagner/W. 11]

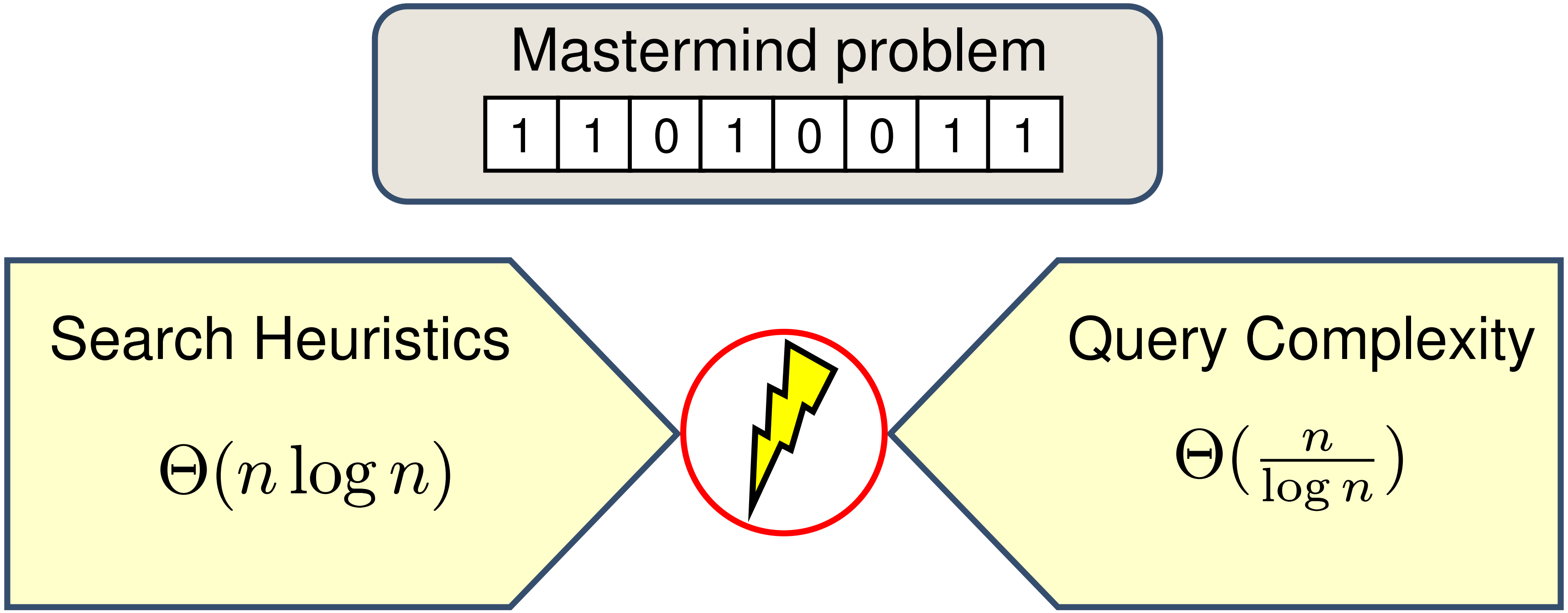
Optimal Strategies (2/2)



[Anil/Wiegand 09], see also [D./Johannsen/Kötzing/Lehre/Wagner/W. 11]

Intermediate Summary

- Want to understand **tractability of a problem** for general-purpose (randomized) search heuristics
- **Query complexity** as such is not a sufficient measure:

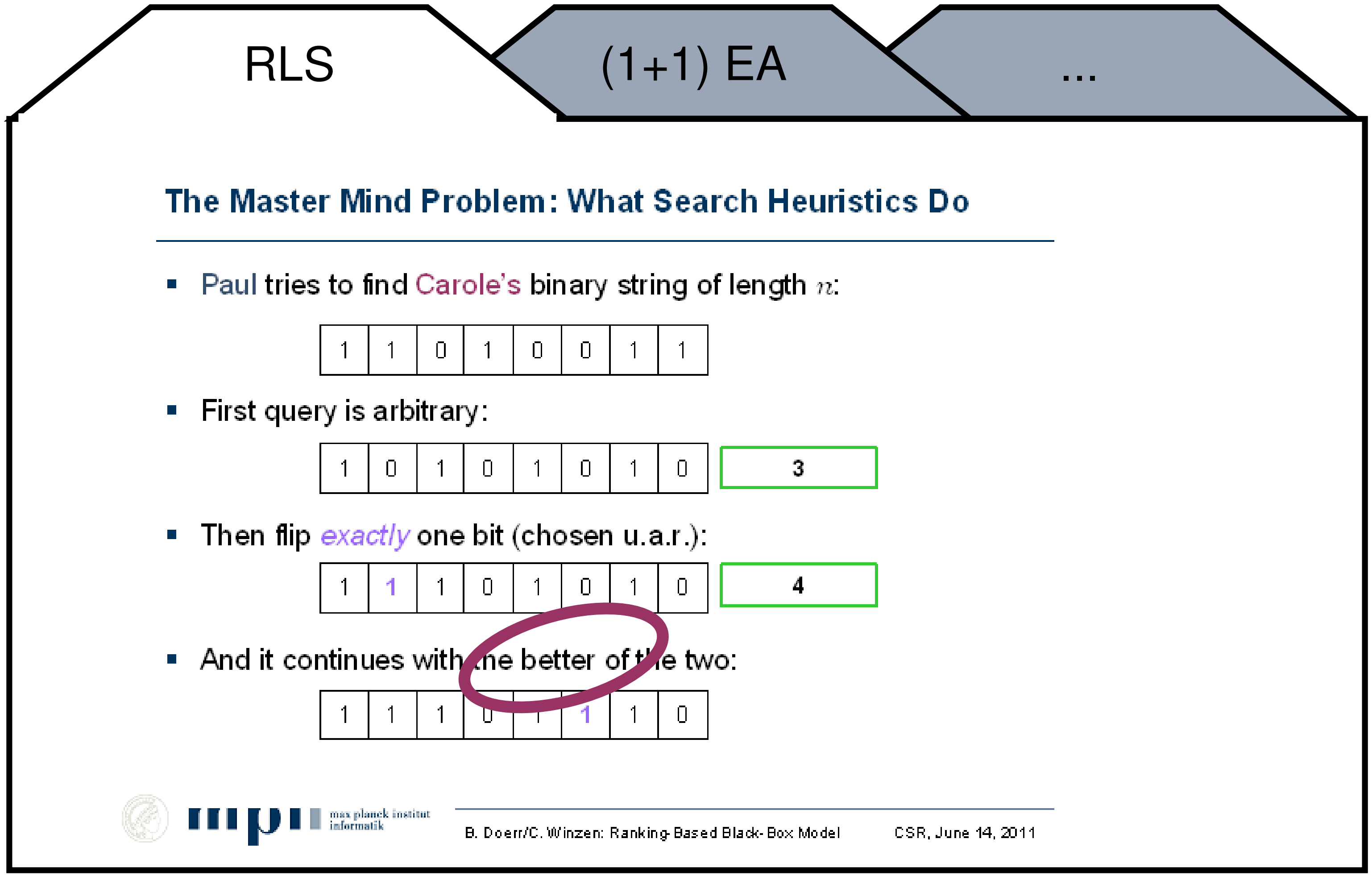


The Ranking-Based Black-Box Model

- Observation: many randomized search heuristics use fitness values only to compare

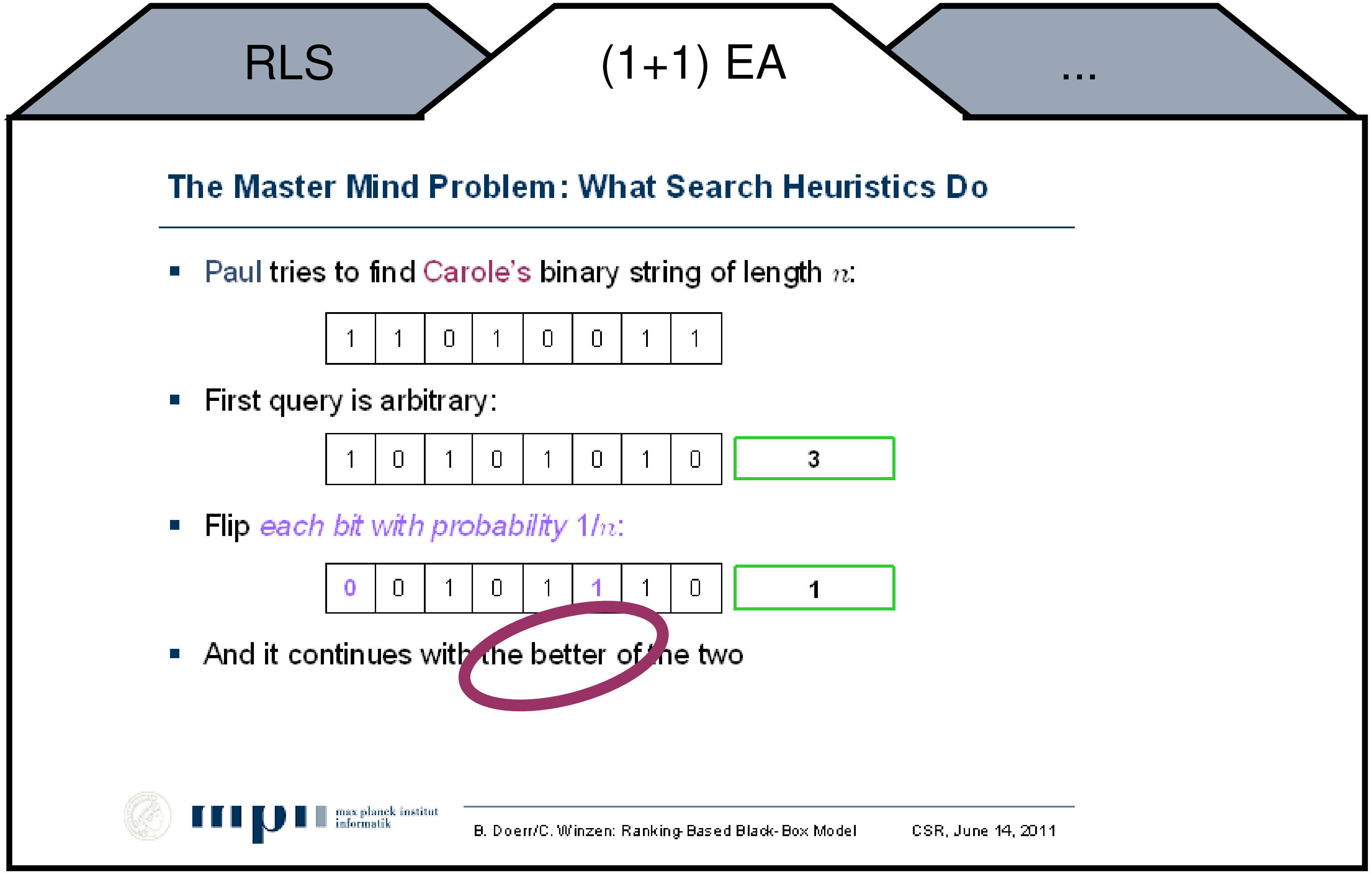
The Ranking-Based Black-Box Model

- Observation: many randomized search heuristics use fitness values only to compare



The Ranking-Based Black-Box Model

- Observation: many randomized search heuristics use fitness values only to compare

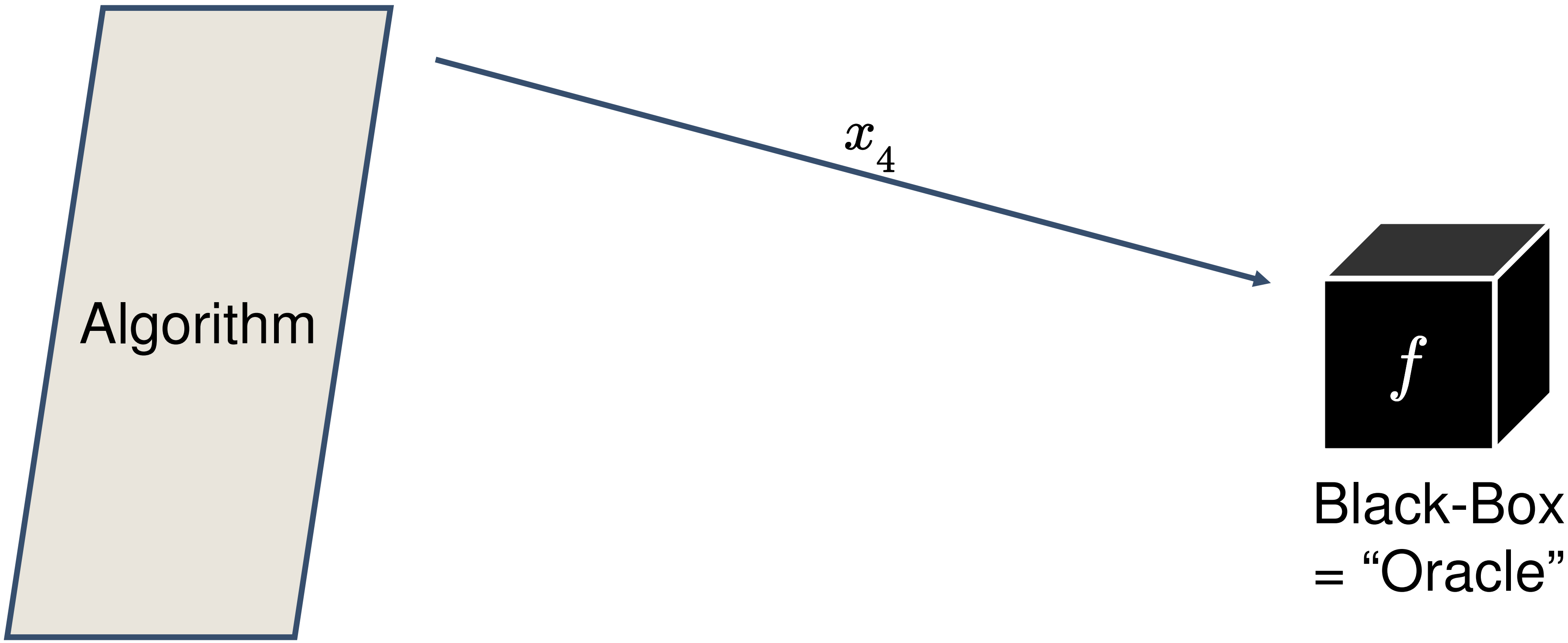


The Ranking-Based Black-Box Model

Does not reveal absolute fitness values:

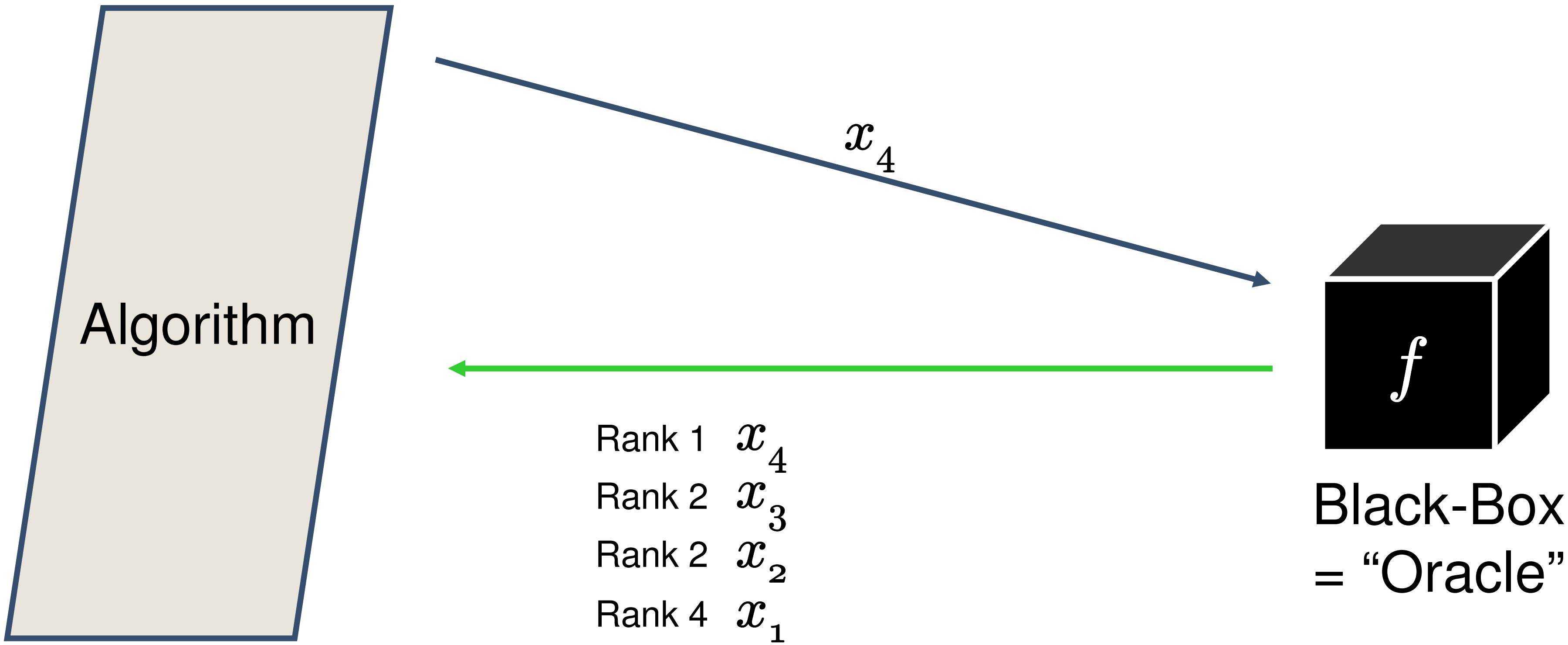
The Ranking-Based Black-Box Model

Does not reveal absolute fitness values:



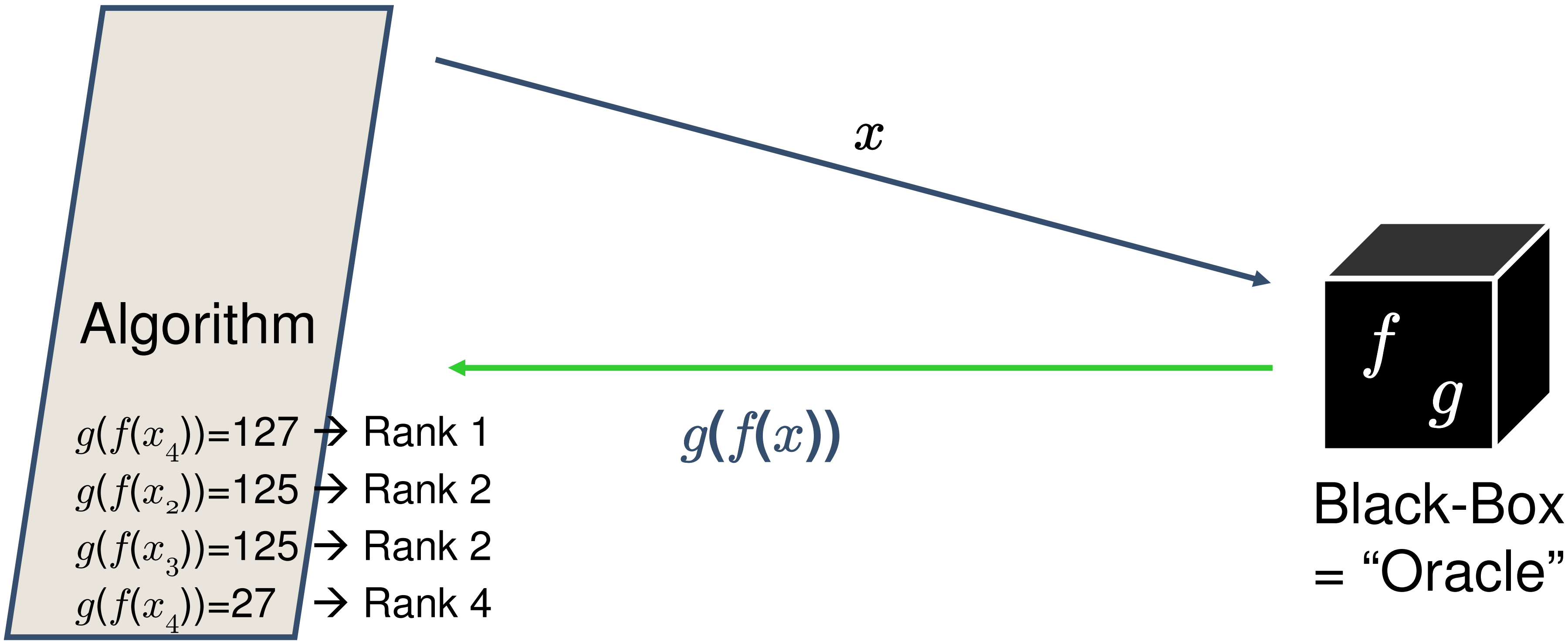
The Ranking-Based Black-Box Model

Does not reveal absolute fitness values:



The Ranking-Based Black-Box Model

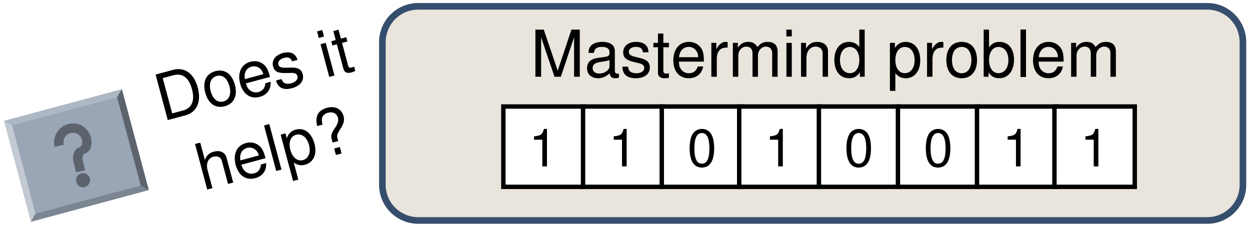
Equivalent formulation: Let $g : \mathbb{R} \to \mathbb{R}$ be a strictly monotone function



Intermediate Summary

- Want to understand **tractability of a problem** for general-purpose (randomized) search heuristics
- **Query complexity** as such is not a sufficient measure
- (Many) Randomized search heuristics do selection based on **relative fitness values** only, **not** on absolute values:

  Ranking-Based Black-Box Model



The Ranking-Based BBC of Mastermind is $\Theta(n$ / log $n)$

Mastermind problem

1	1	0	1	0	0	1	1

Search Heuristics

$\Theta(n \log n)$

Classical Query Complexity

$\Theta(\frac{n}{\log n})$

Ranking-Based Query Complexity

$\Theta(\frac{n}{\log n})$

Example: BinaryValue //Weighted Mastermind

- Carole chooses a binary string of length n and a permutation σ

1	1	0	1	0	0	1	1
2^4	2^2	2^1	2^3	2^5	2^8	2^6	2^7

σ=(4 2 1 3 5 8 6 7)

- Paul wants to find it. He may ask any string of length n:

1	0	1	0	1	0	1	0

- Carole computes the **weighted fitness value:**

1	0	1	0	1	**0**	**1**	0

- "Paul, your string has a score of 336 (=2^4+2^8+2^6)"

The Query Complexity of BinaryValue is O(log n)

Paul can do a binary search (parallel for each $i \leq n$):

1	1	0	1	0	0	1	1
2^4	2^2	2^1	2^3	2^5	2^8	2^6	2^7

1	1	1	1	1	1	1	1	$2^4+2^2+2^3+2^6+2^7$
1	1	1	1	0	0	0	0	$2^4+2^2+2^3+2^5+2^8$
1	1	0	0	1	1	0	0	$2^4+2^2+2^1$
1	0	1	0	1	0	1	0	$2^4+2^8+2^6$

1	1	0	1	0	0	1	1

Binary Search *not* Possible in Ranking-Based Model

Paul can do a binary search (parallel for each $i \leq n$):

1	1	0	1	0	0	1	1

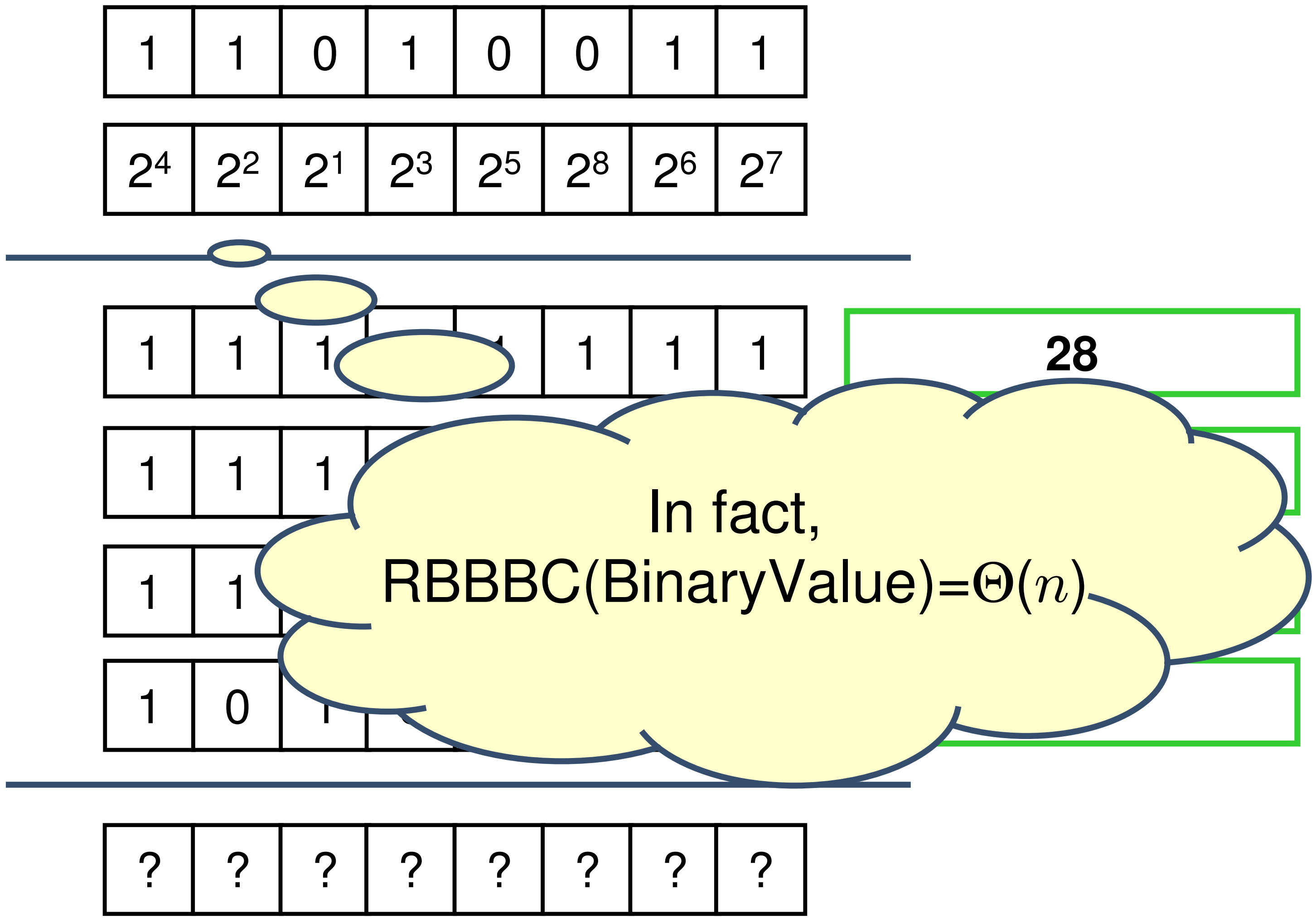
2^4	2^2	2^1	2^3	2^5	2^8	2^6	2^7

1	1	1	1	1	1	1	1	**28**
1	1	1	1	0	0	0	0	**317**
1	1	0	0	1	1	0	0	**-29**
1	0	1	0	1	0	1	0	**29**

?	?	?	?	?	?	?	?

Binary Search *not* Possible in Ranking-Based Model

Paul can do a binary search (parallel for each $i \leq n$):



The Ranking-Based Black-Box Complexity of BinaryValue is $\Theta(n)$

Limited Learning
t=2

If Algorithm queries two strings x and y, it can learn at most 1 bit of the target string z.

Example:

x=**100000**

y=**000000**

$$g(\mathrm{BV}_{z,\sigma}(x)) > g(BV_{z,\sigma}(y))$$
$$\Leftrightarrow$$
$$\mathrm{z}_1 = x_1 = 1$$

The Ranking-Based Black-Box Complexity of BinaryValue is $\Theta(n)$

Limited Learning t=2	**If Algorithm queries two strings x and y, it can learn at most 1 bit of the target string z.**
Limited Learning general t	**If Algorithm queries t strings $x_1,\dots,x_t$, it can learn at most t-1 bit of the target string z.**

The Ranking-Based Black-Box Complexity of BinaryValue is $\Theta(n)$

Limited Learning t=2	**If Algorithm queries two strings x and y, it can learn at most 1 bit of the target string z.**
Limited Learning general t	**If Algorithm queries t strings $x_1,\ldots,x_t$, it can learn at most t-1 bit of the target string z.**

***Ω(n)* deterministic lower bound**	**There is no deterministic algorithm which optimizes BinaryValue in sublinear time.**

The Ranking-Based Black-Box Complexity of BinaryValue is $\Theta(n)$

Limited Learning t=2	**If Algorithm queries two strings x and y, it can learn at most 1 bit of the target string z.**
Limited Learning general t	**If Algorithm queries t strings $x_1,\ldots,x_t$, it can learn at most t-1 bit of the target string z.**
***Ω(n)* deterministic lower bound**	**There is no deterministic algorithm which optimizes BinaryValue in sublinear time.**
***Ω(n)* randomized lower bound**	**Follows from deterministic lower bound and Yao's minimax principle**

Summary

- Want to understand tractability of different problems for (randomized) search heuristics
- Measure: number of function evaluations

Summary

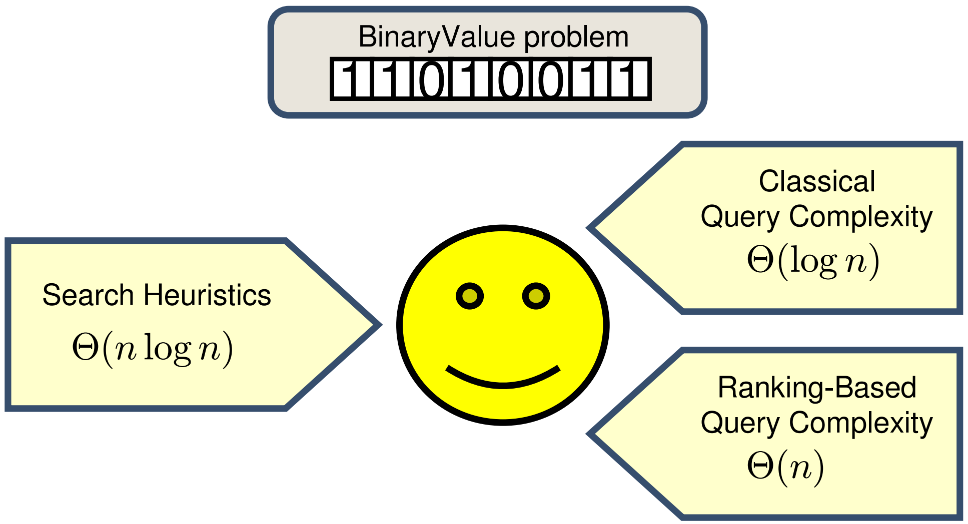
- Want to understand tractability of different problems for (randomized) search heuristics
- Measure: number of function evaluations
- Our main interest are **good lower bounds**
- Classical query complexity: often too weak lower bounds

Summary

- Want to understand tractability of different problems for (randomized) search heuristics
- Measure: number of function evaluations
- Our main interest are **good lower bounds**
- Classical query complexity: often too weak lower bounds
- **Ranking-Based Black-Box Model:**
  - query complexity model
  - only relative, not absolute fitness values are given

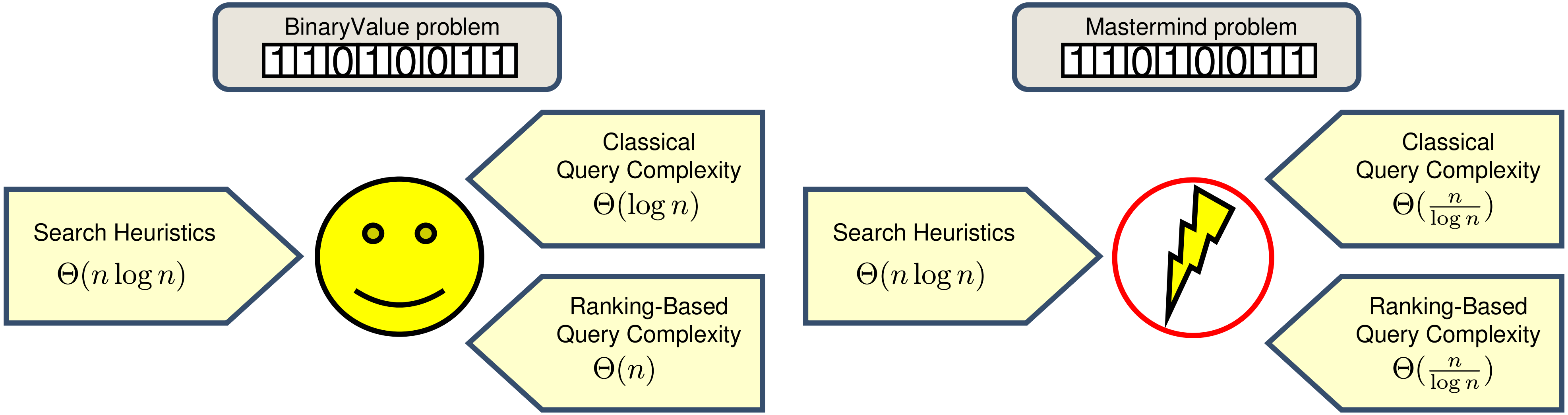
Summary

- Want to understand tractability of different problems for (randomized) search heuristics
- Measure: number of function evaluations
- Our main interest are **good lower bounds**
- Classical query complexity: often too weak lower bounds
- **Ranking-Based Black-Box Model:**
  - only relative, not absolute fitness values are given



Summary

- Want to understand tractability of different problems for (randomized) search heuristics
- Measure: number of function evaluations
- Our main interest are **good lower bounds**
- Classical query complexity: often too weak lower bounds
- **Ranking-Based Black-Box Model:**
  - only relative, not absolute fitness values are given



Future Work

Plenty!

Future Work

Plenty!

- Other black-box models:
  - restricted memory models
  - unbiased sampling strategies
  - combinations thereof
  - ...

Future Work

Plenty!

- Other black-box models:
  - restricted memory models
  - unbiased sampling strategies
  - combinations thereof
  - ...
- Combinatorial problems:
  - MST, SSSP problems
  - partition problem
  - ...
- ...

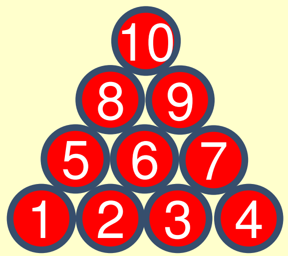
Future Work

Plenty!

- Other black-box models:
  - restricted memory models
  - unbiased sampling strategies
  - combinations thereof
  - ...
- Combinatorial problems:
  - MST, SSSP problems
  - partition problem
  - ...
- ...

<u>Almost equivalent problem:</u>

n distinguishable balls of unknown weight



How often do you need to use the balance to find a perfect partition of the balls?