Artificial Life
Summer 2025

Evolutionary Algorithms 2

Master Computer Science [MA-INF 4201]
Mon 14:15 – 15:45, HSZ, HS-2

Dr. Nils Goerke, Autonomous Intelligent Systems,
Department of Computer Science, University of Bonn

Last Lecture: Mon 19.5.2025

- Evolutionary Computation
- Historic Remarks
- Different Approaches
- Idea of Evolutionary Algorithms (EA)
- EA Steps
  - Individual, Genome, Fitness, Population
  - Parent selection
  - Inheritance
  - Mutation
  - Fitness evaluation
  - External selection

Overview

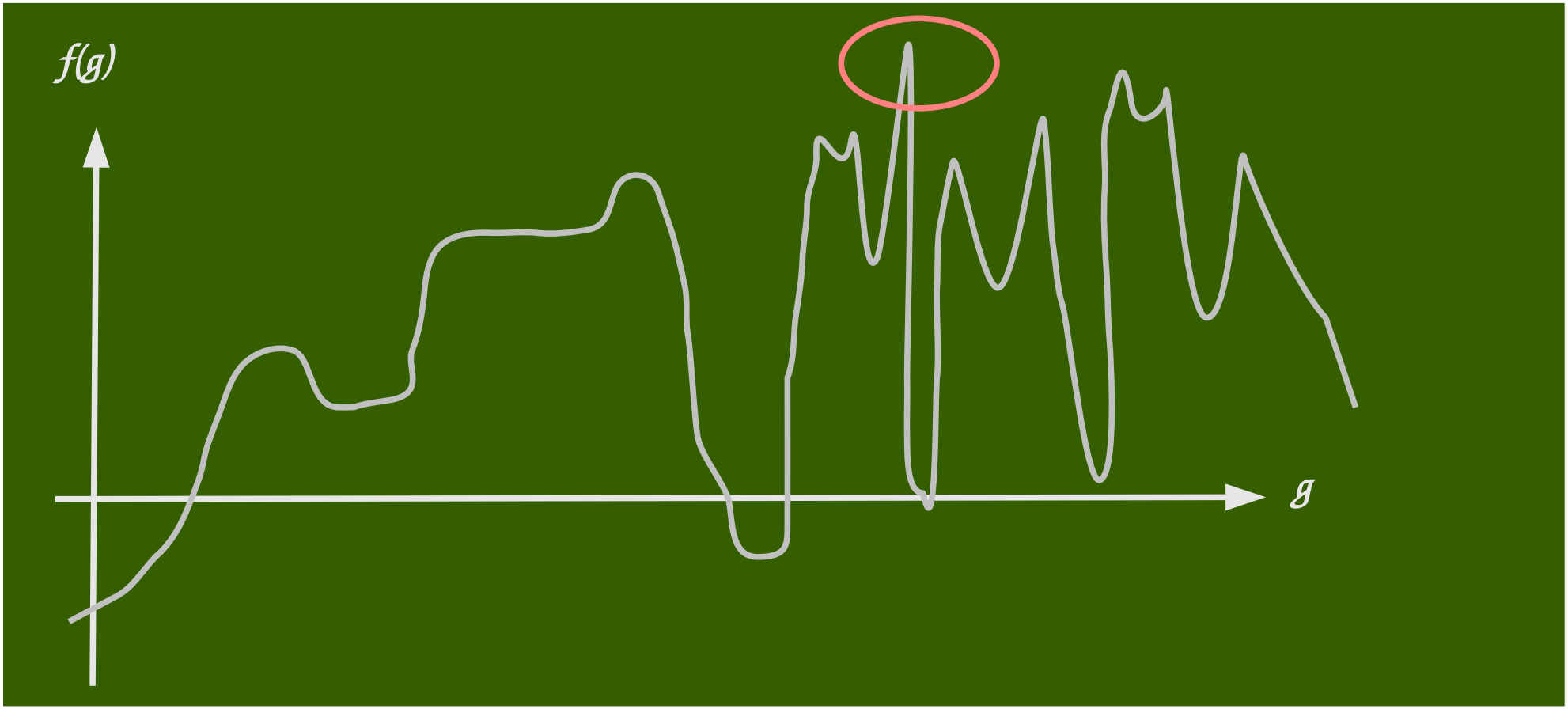
- EA Steps
  - Individual, Genome, Fitness, Population
  - Parent selection
  - Inheritance
  - Mutation
  - Fitness evaluation
  - External selection
  - Finish?
  - Initialization
- Strategy
- Performance Graph
- Genome Structure
- Examples

Overview

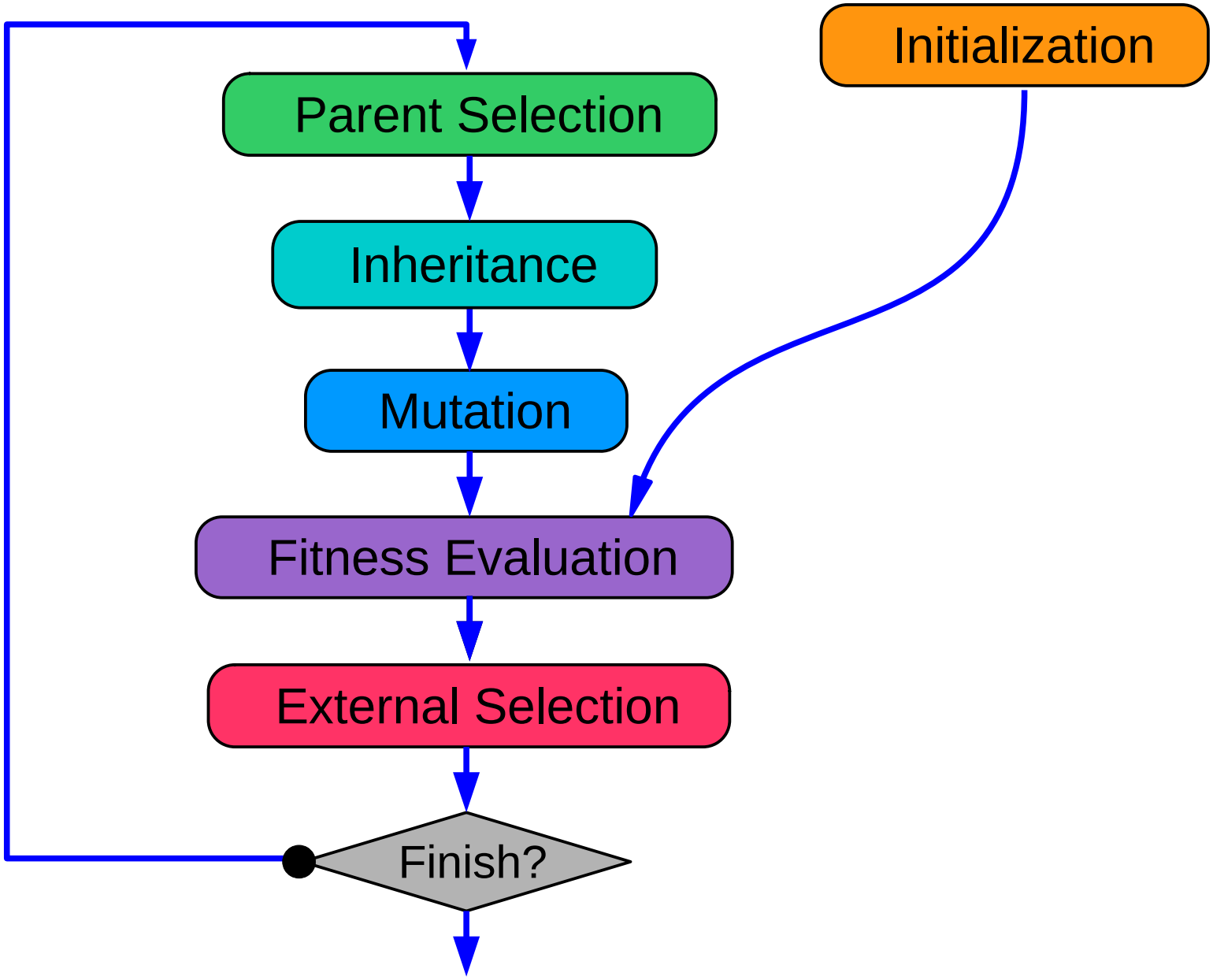
- EA Steps
  - Individual, Genome, Fitness, Population
  - Parent selection
  - Inheritance
  - Mutation
  - Fitness evaluation
  - External selection
  - Finish?
  - Initialization
- Strategy
- Performance Graph
- Genome Structure
- Examples

Optimization:

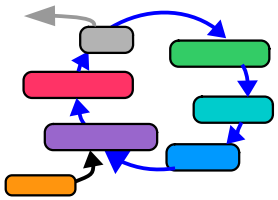
Draw a **nasty fitness function** on the blackboard:

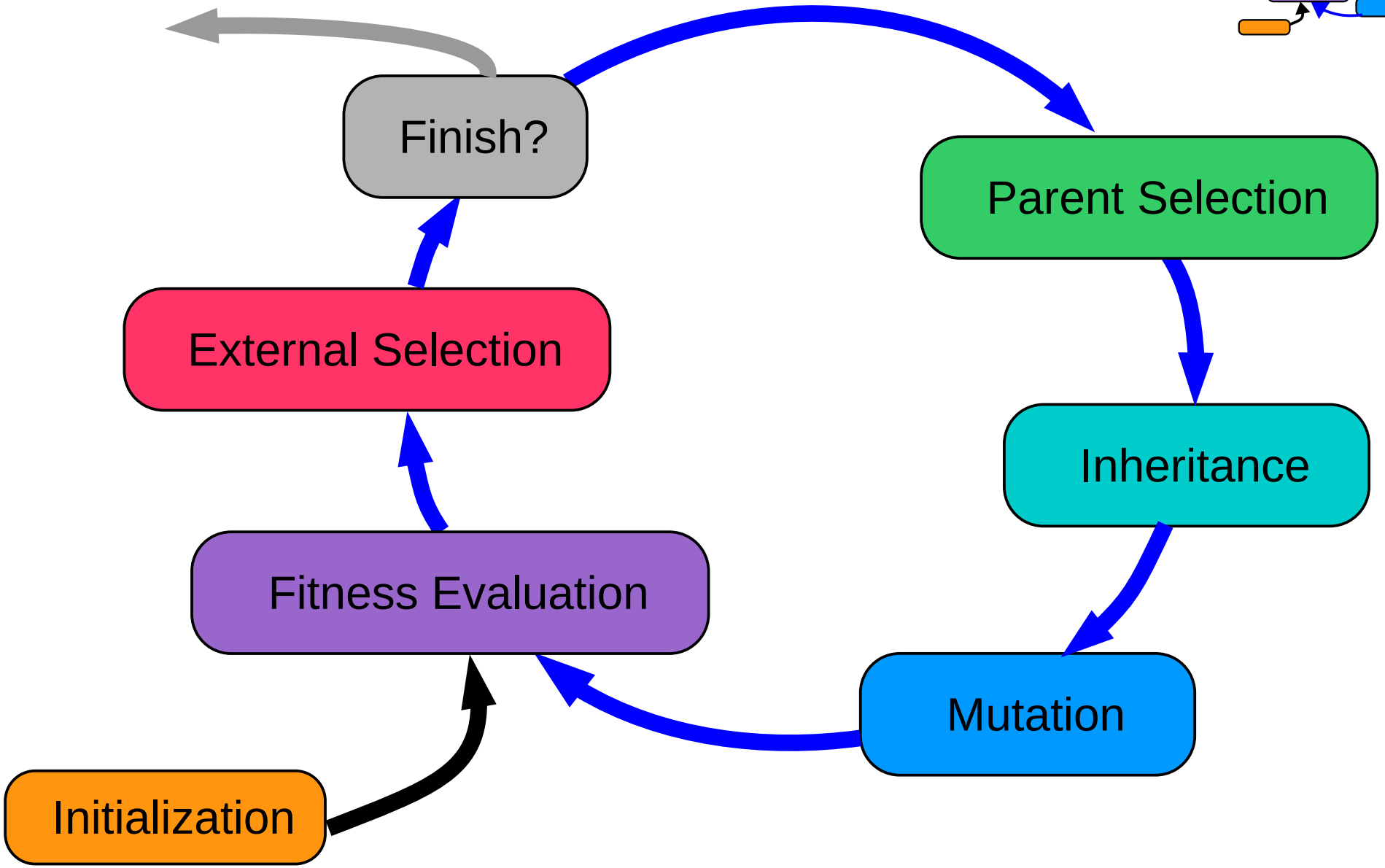


EA steps:


Initialization
Parent Selection
Inheritance
Mutation
Fitness Evaluation
External Selection
Finish?

EA cycle of operations:

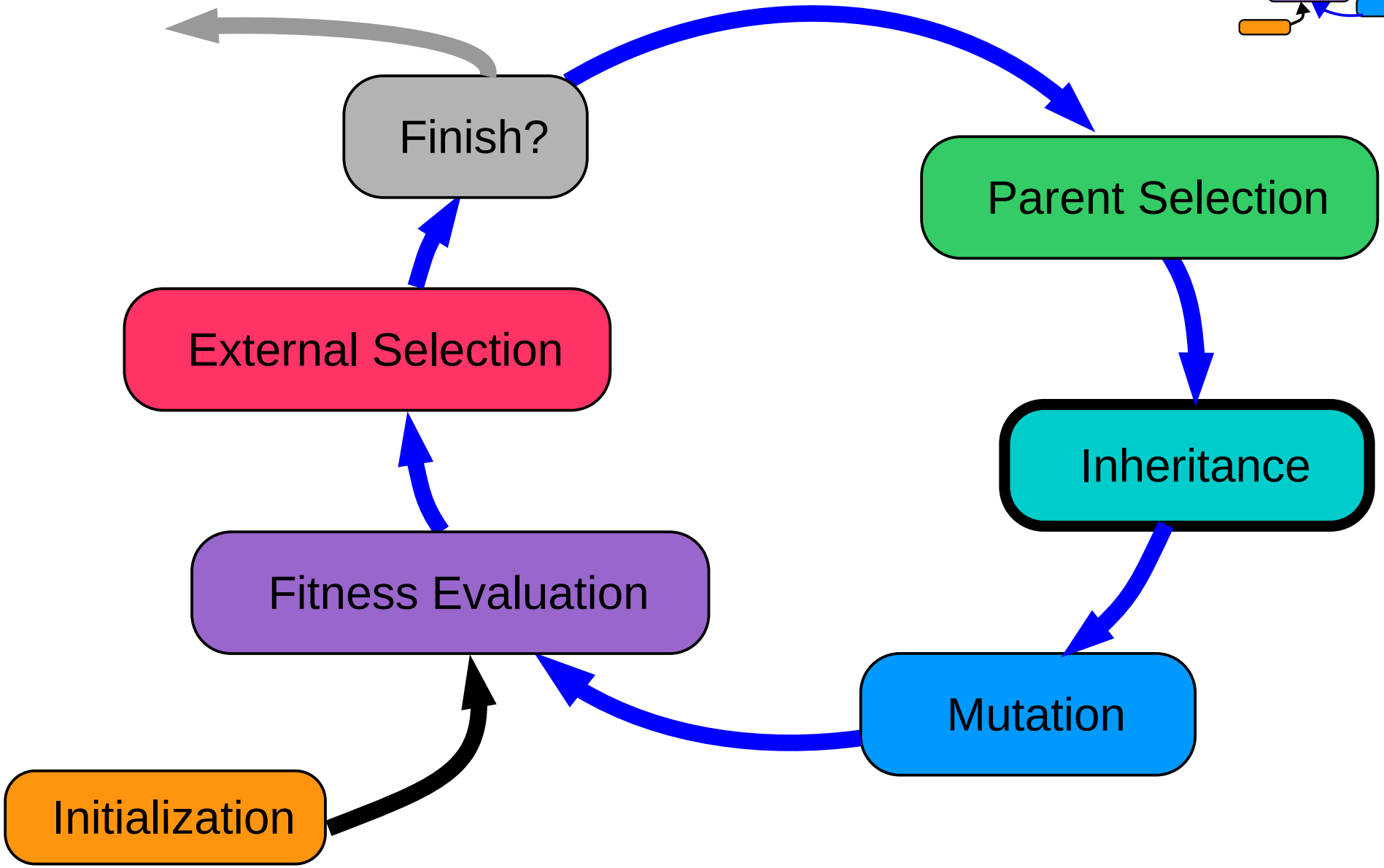




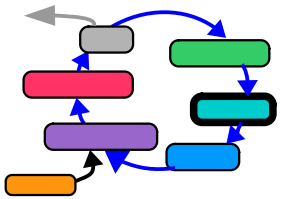
Overview

- EA Steps
  - Individual, Genome, Fitness, Population
  - Parent selection
  - **Inheritance**
  - Mutation
  - Fitness evaluation
  - External selection
  - Finish?
  - Initialization
- Strategy
- Performance Graph
- Genome Structure
- Examples

EA cycle of operations:



EA: Inheritance



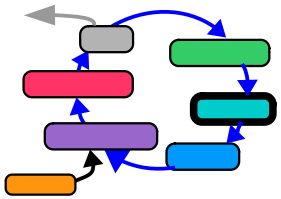
Inheritance is the principle of transporting some of the information from the previous generation (parents), into the next generation, by generating the **offspring** out of the **parents genomes** .

Thus, generating offspring means to create new individuals with new genomes.

Inheritance in the context of Evolutionary Algorithms means to use the genomes of those parents that have been selected.

There is a wide variety of possible ways to generate offspring.

EA: Inheritance



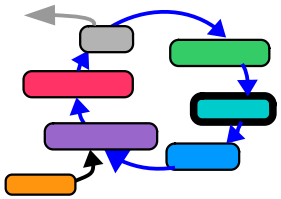
The most popular mechanism to implement the **inheritance** principle is to use k=2 parents and to **combine** their genomes (mostly called recombination).

The most popular mechanism to implement **recombination** of 2 genomes is **cross over:**

Inheritance:

- **recombination**
  - by **cross over**
  - (other recombination operators)
- *(other inheritance mechanism)*
  - (other inheritance operators)
  - ...

EA: Inheritance

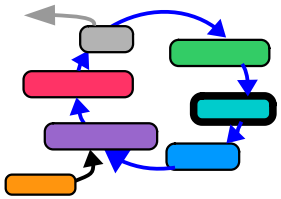


Recombination by **1-point cross over**:

Select two parents, **A** and **B** and **recombine** their genomes by taking one part form parent **A**
and the other part from parent **B**

Choose a **random point C** where to split the genomes, and recombine the four remaining fractions to build new individuals, containing partial information from both parents.

EA: Inheritance



Recombination by **1-point cross over**:

Parent A:

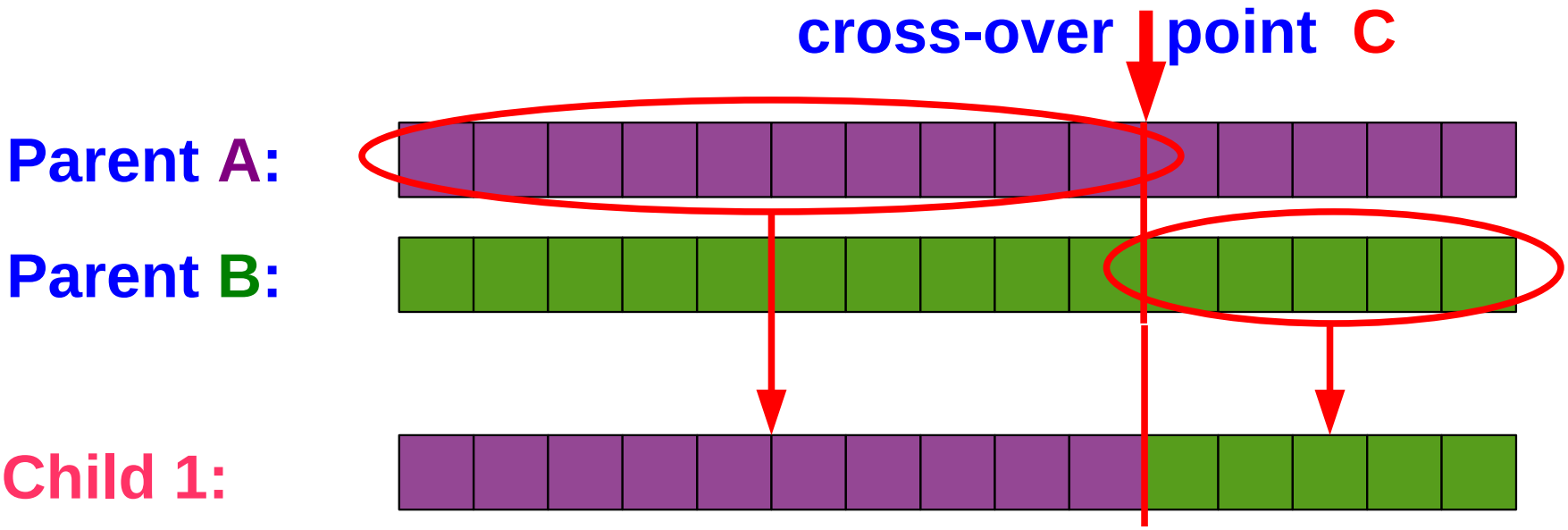
Parent B:

Child 1:

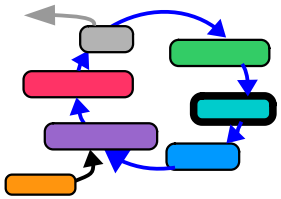
?

EA: Inheritance

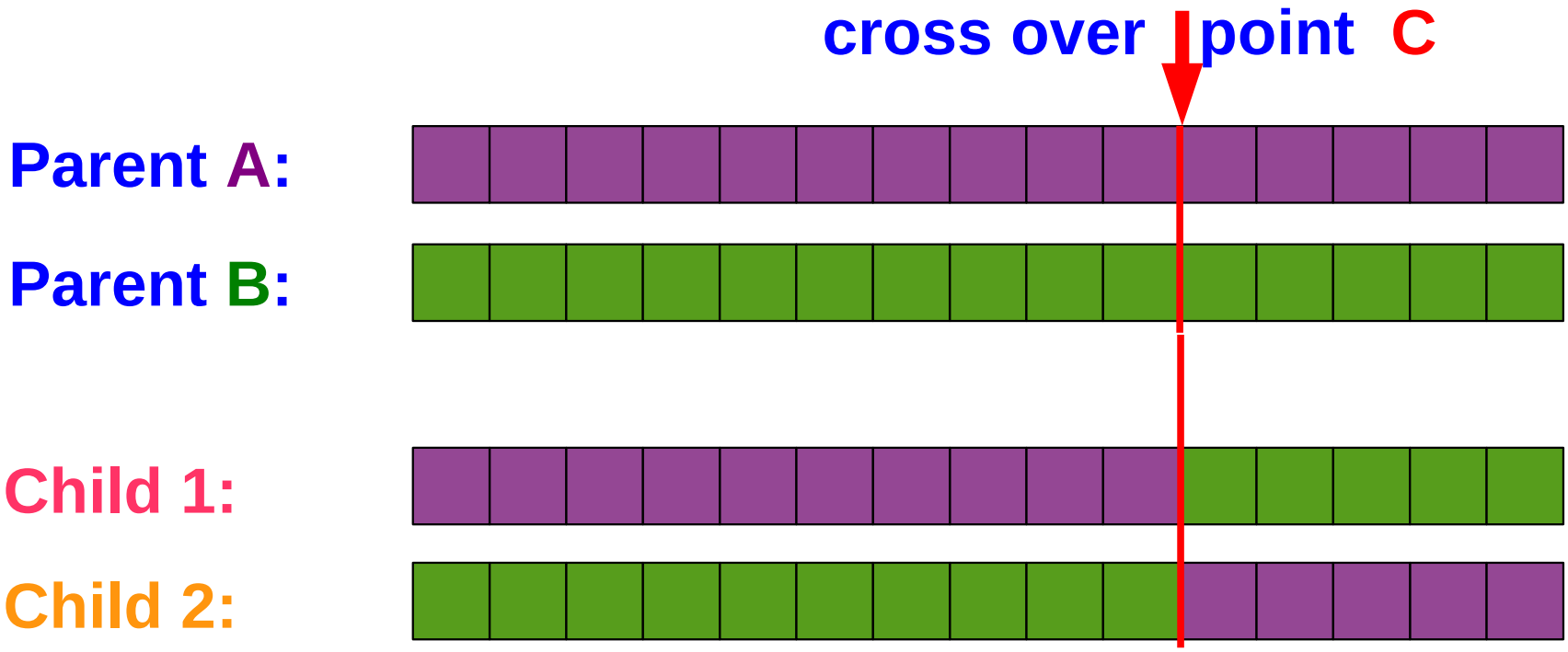
Recombination by **1-point cross-over**:



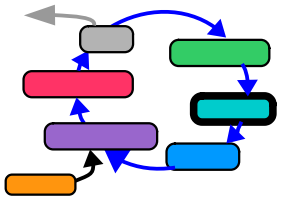
EA: Inheritance



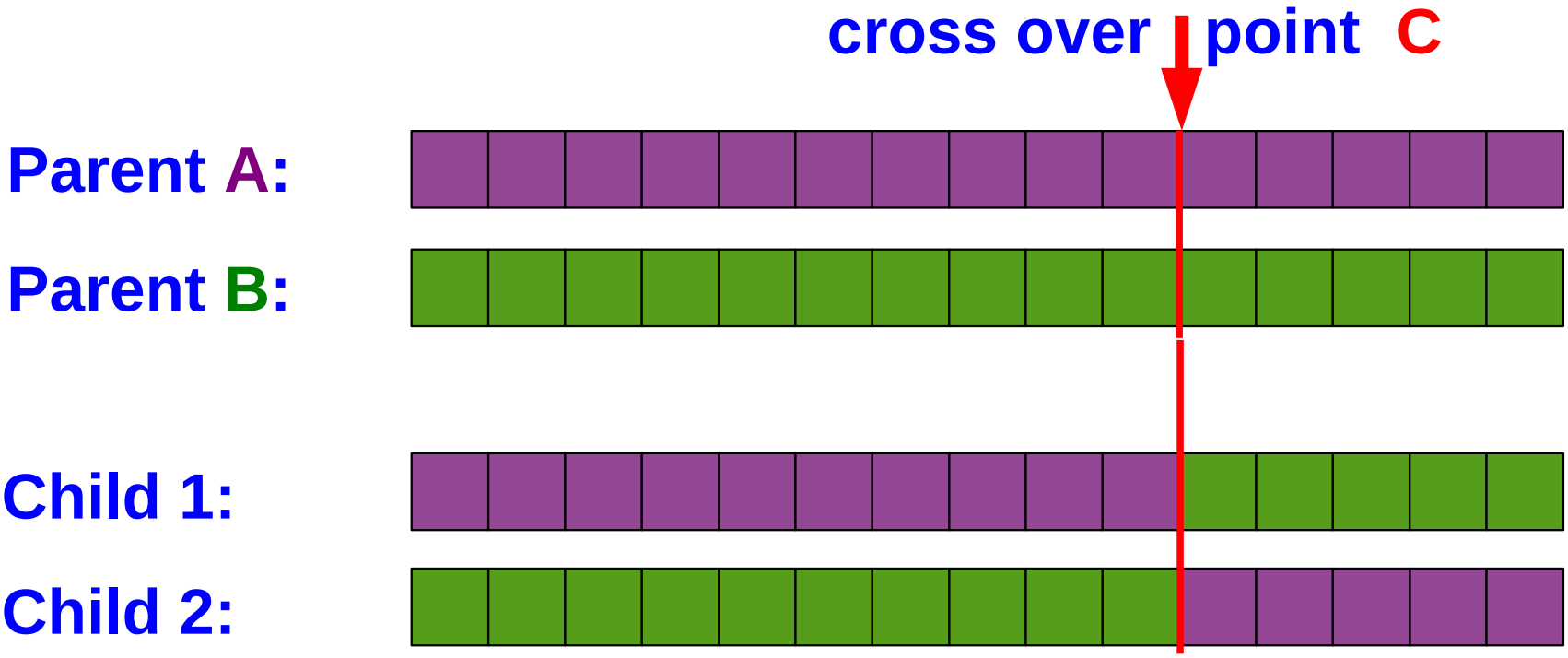
Recombination by **1-point cross over**:



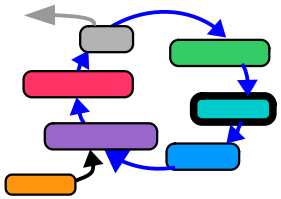
EA: Inheritance



Recombination by **1-point cross over**:



EA: Inheritance



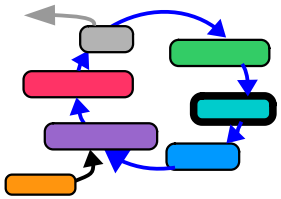
Recombination by 1-point cross over:

Parents **A, B**

A = (x_a , y_a)

B = (x_b , y_b)

EA: Inheritance



Recombination by 1-point cross over:

Parents **A, B**

A = (x_a , y_a)

B = (x_b , y_b)

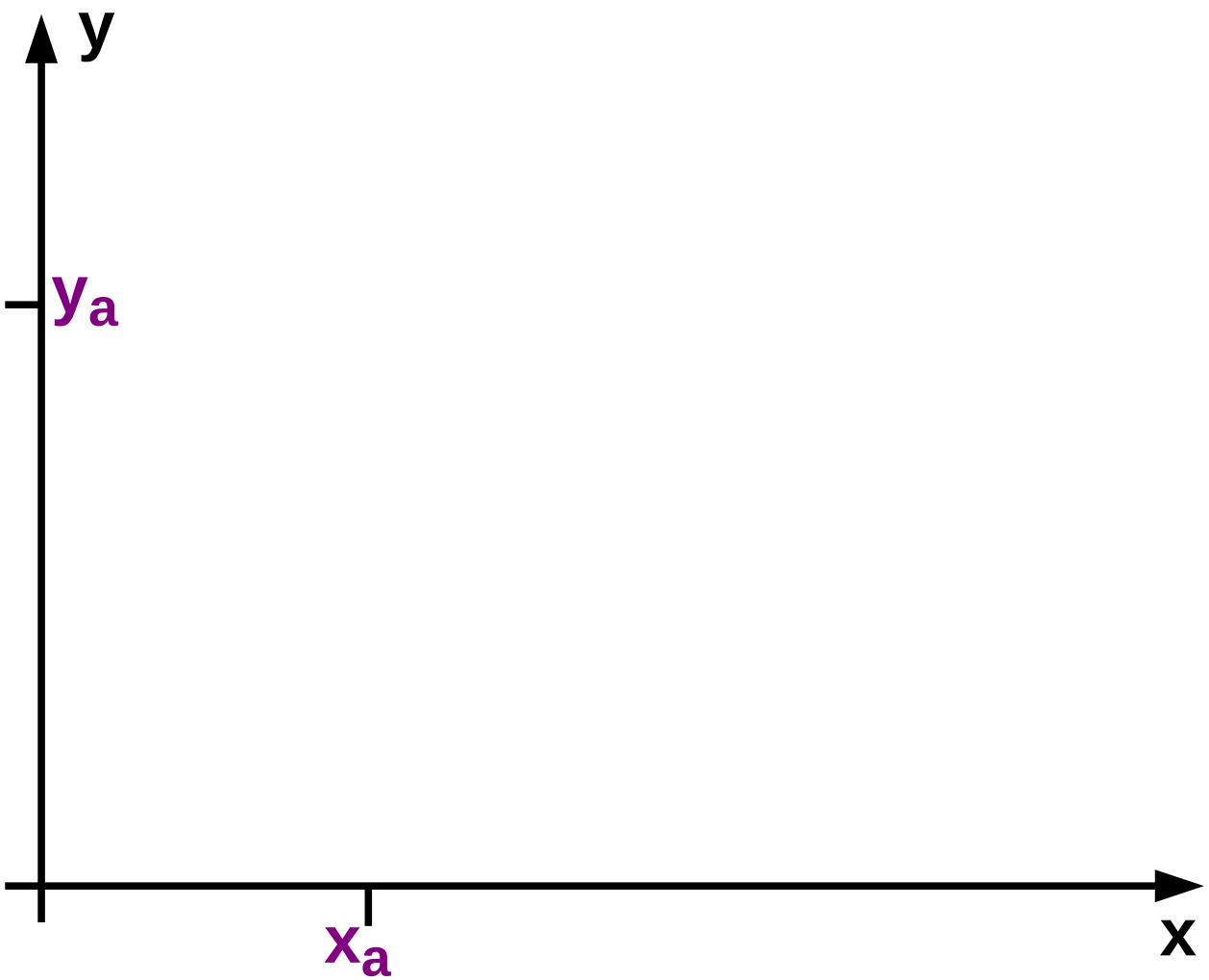
EA: Inheritance

Recombination by 1-point cross over:

Parents **A, B**

A = (x_a , y_a)

B = (x_b , y_b)



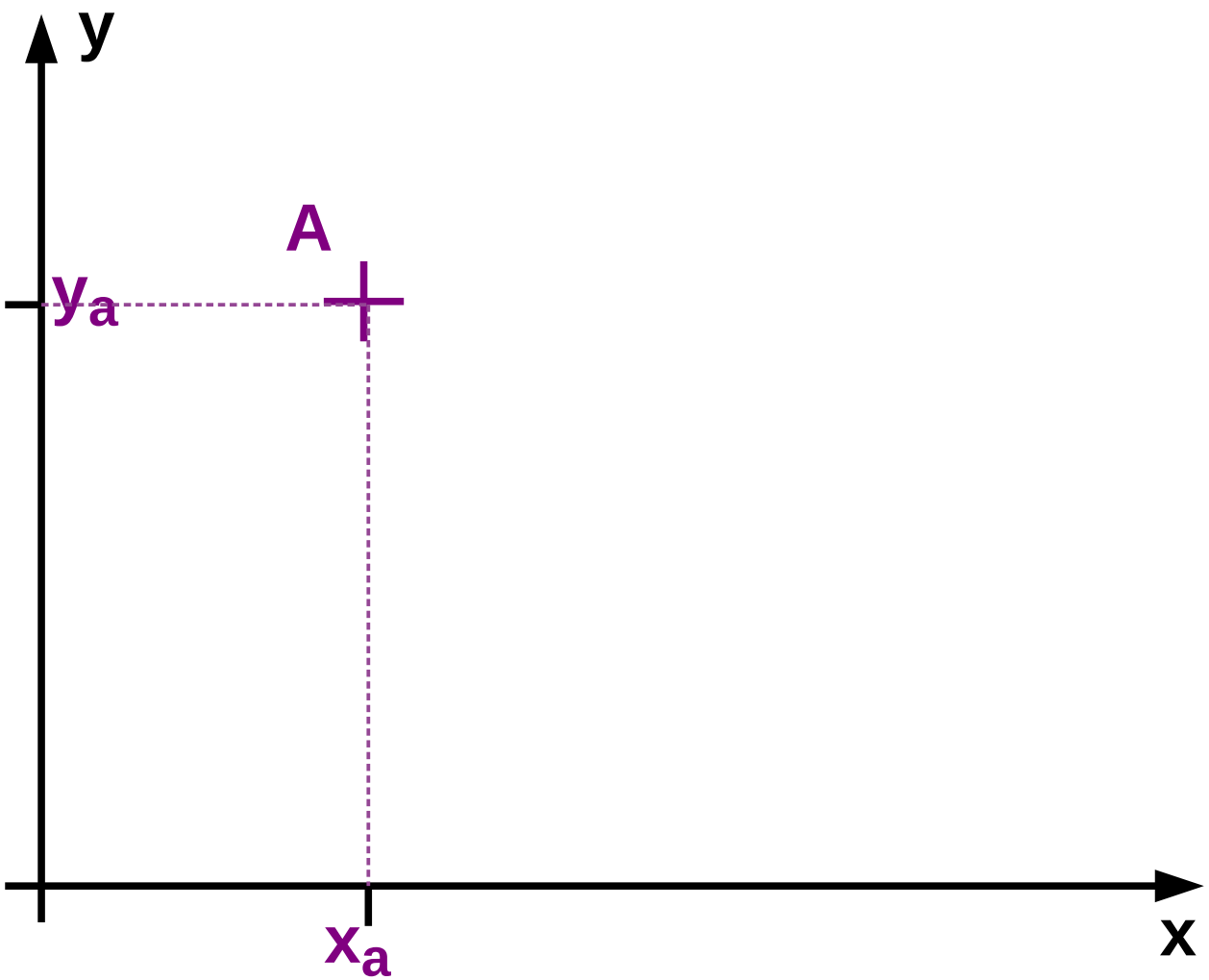
EA: Inheritance

Recombination by 1-point cross over:

Parents **A, B**

A = (x_a , y_a)

B = (x_b , y_b)



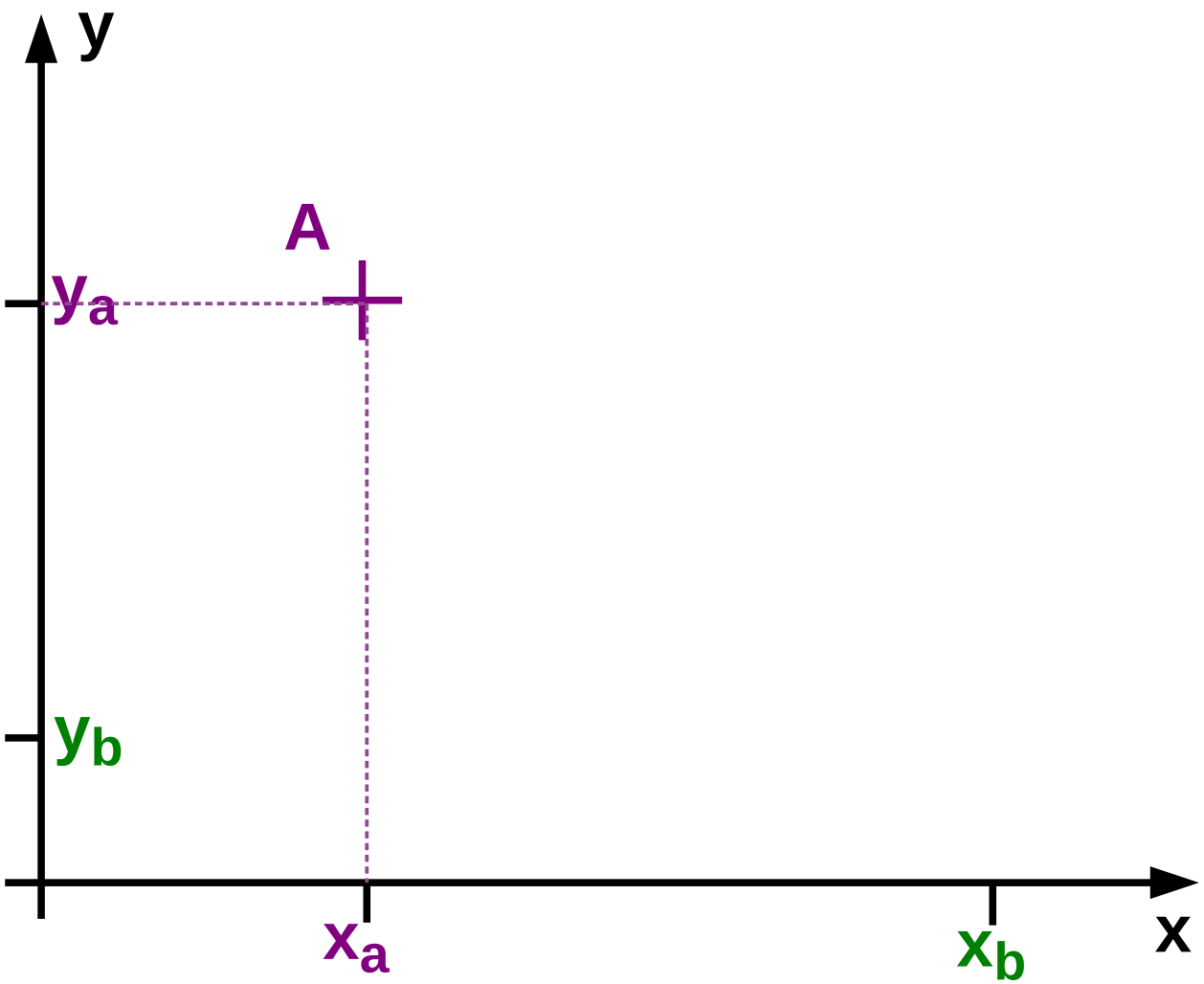
EA: Inheritance

Recombination by 1-point cross over:

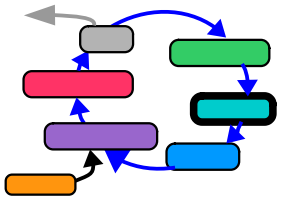
Parents A, B

$A = (x_a , y_a)$

$B = (x_b , y_b)$



EA: Inheritance

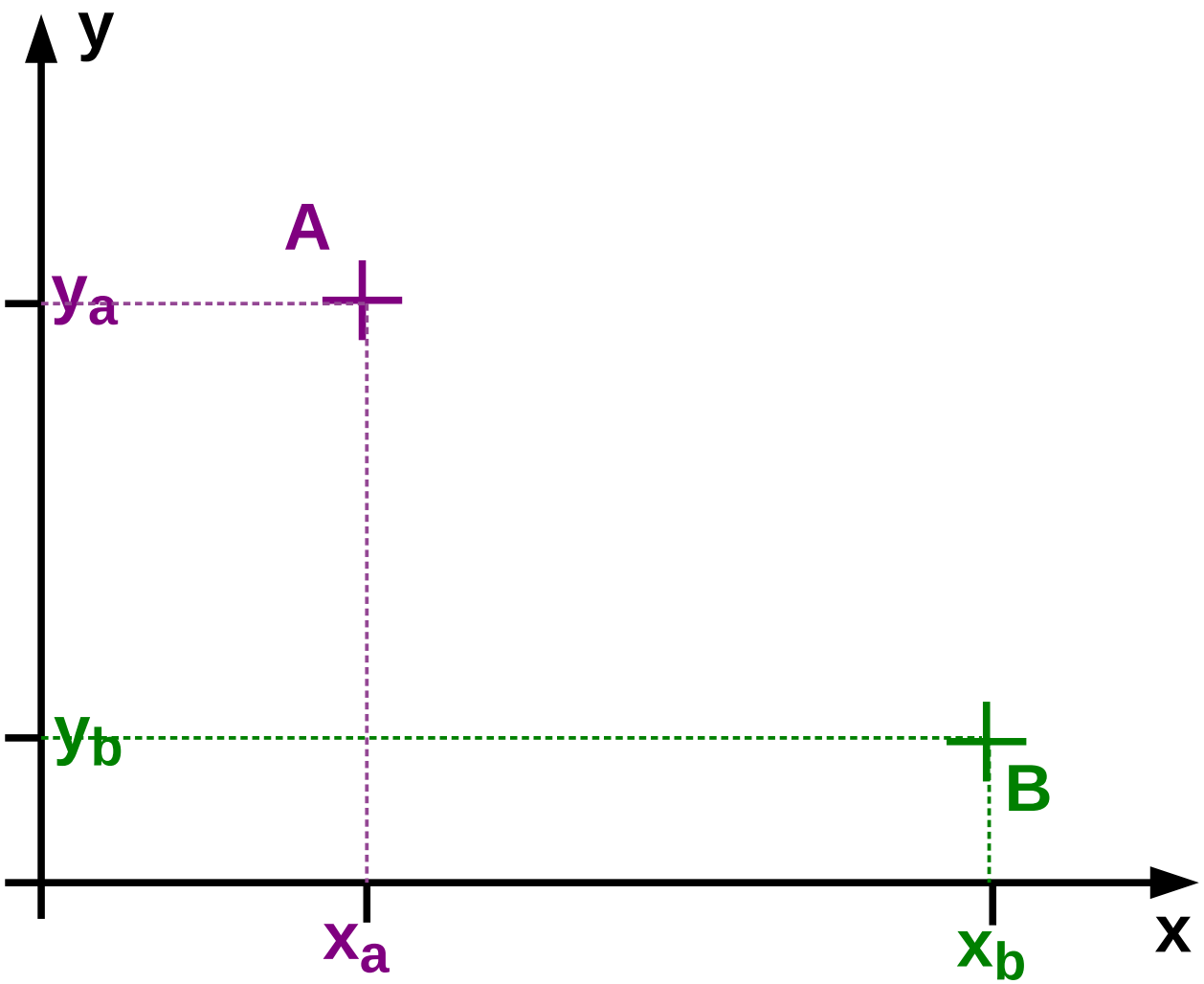


Recombination by 1-point cross over:

Parents **A, B**

A = (x_a , y_a)

B = (x_b , y_b)



EA: Inheritance

Recombination by 1-point cross over:

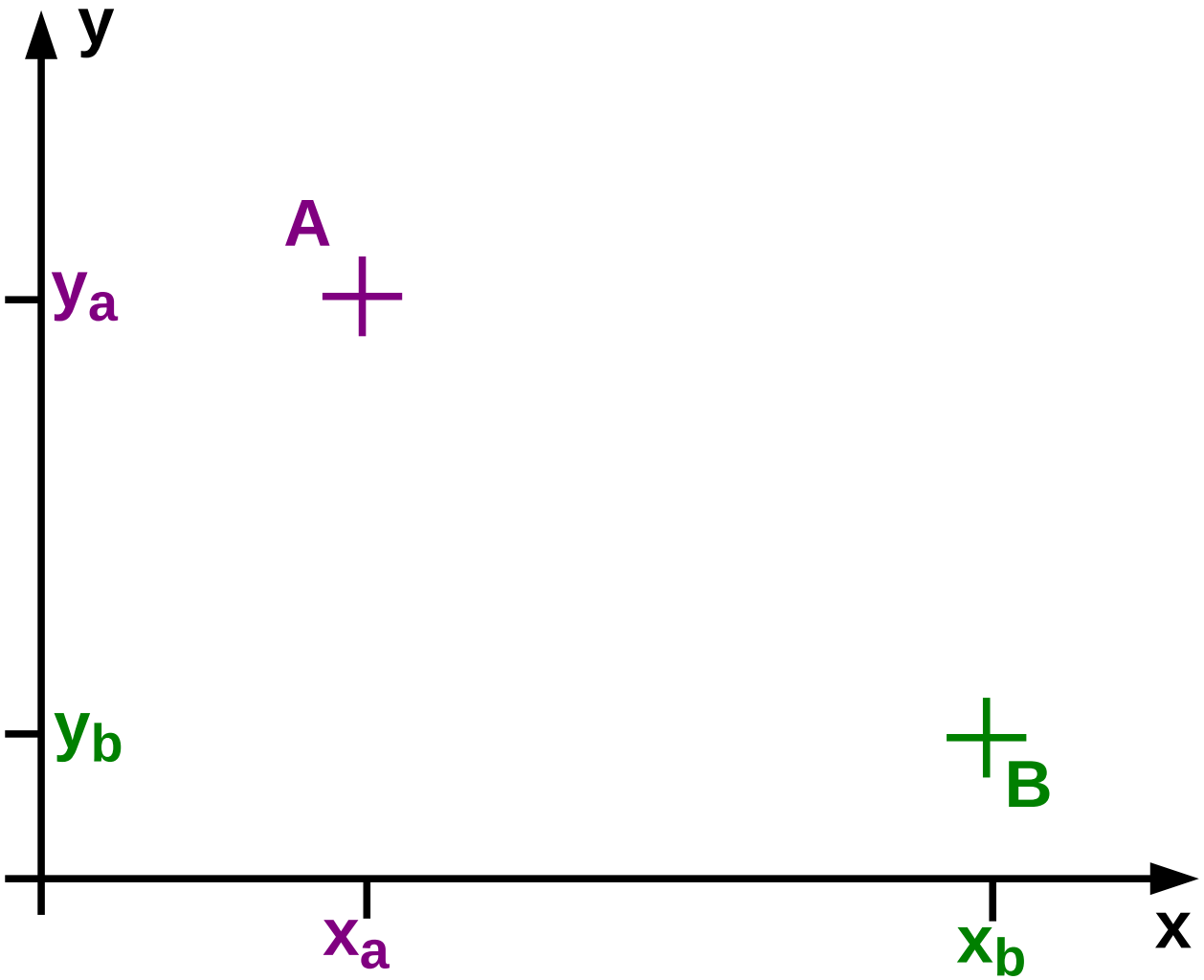
Parents A, B

$A = (x_a , y_a)$

$= (\boxed{x_a \mid y_a})$

$B = (x_b , y_b)$

$= (\boxed{x_b \mid y_b})$



EA: Inheritance

Recombination by 1-point cross over:

Parents A, B

A = (x_a , y_a)

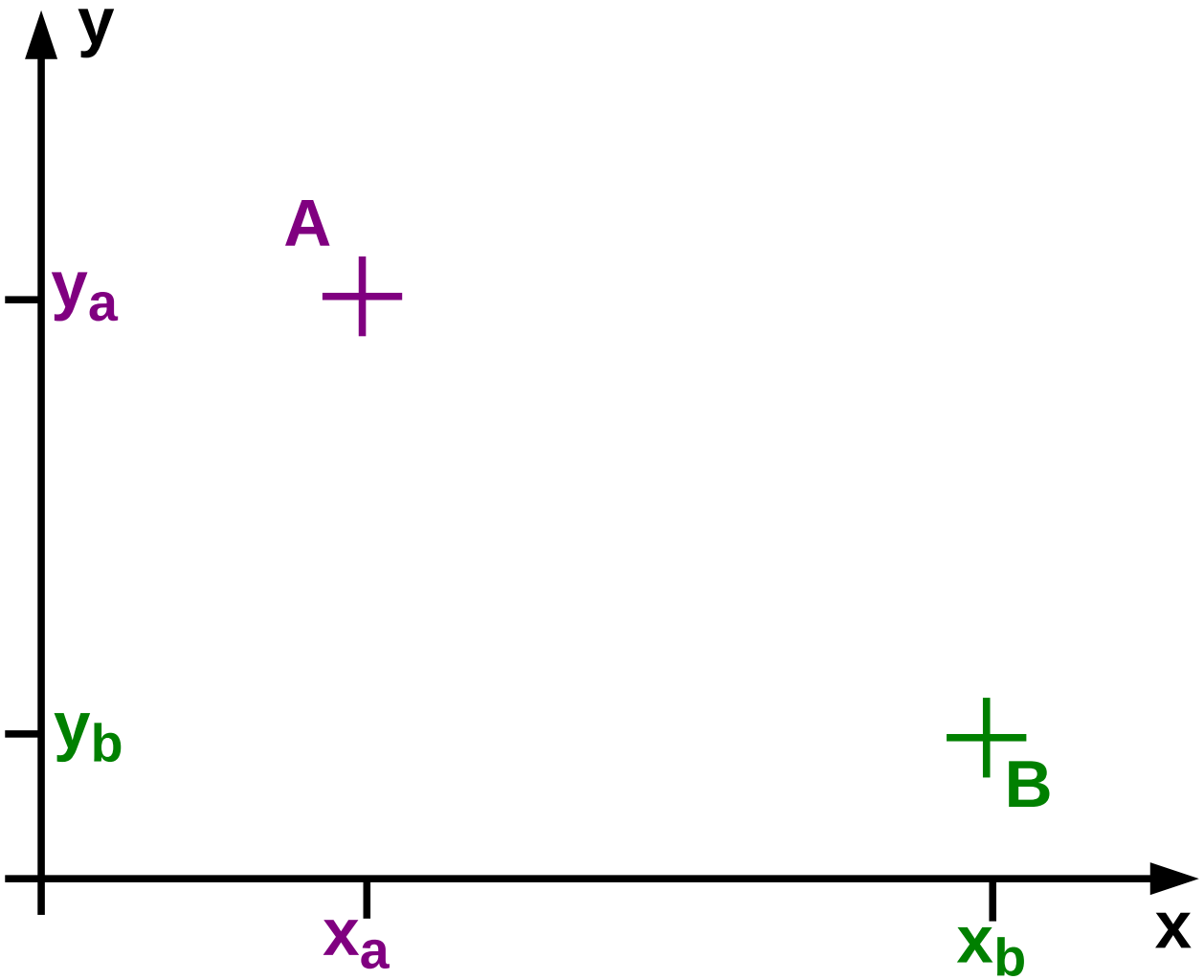
= (x_a | y_a)

B = (x_b , y_b)

= (x_b | y_b)

Child C1

C1 = ( )



EA: Inheritance

Recombination by 1-point cross over:

Parents A, B

A = (x_a , y_a)

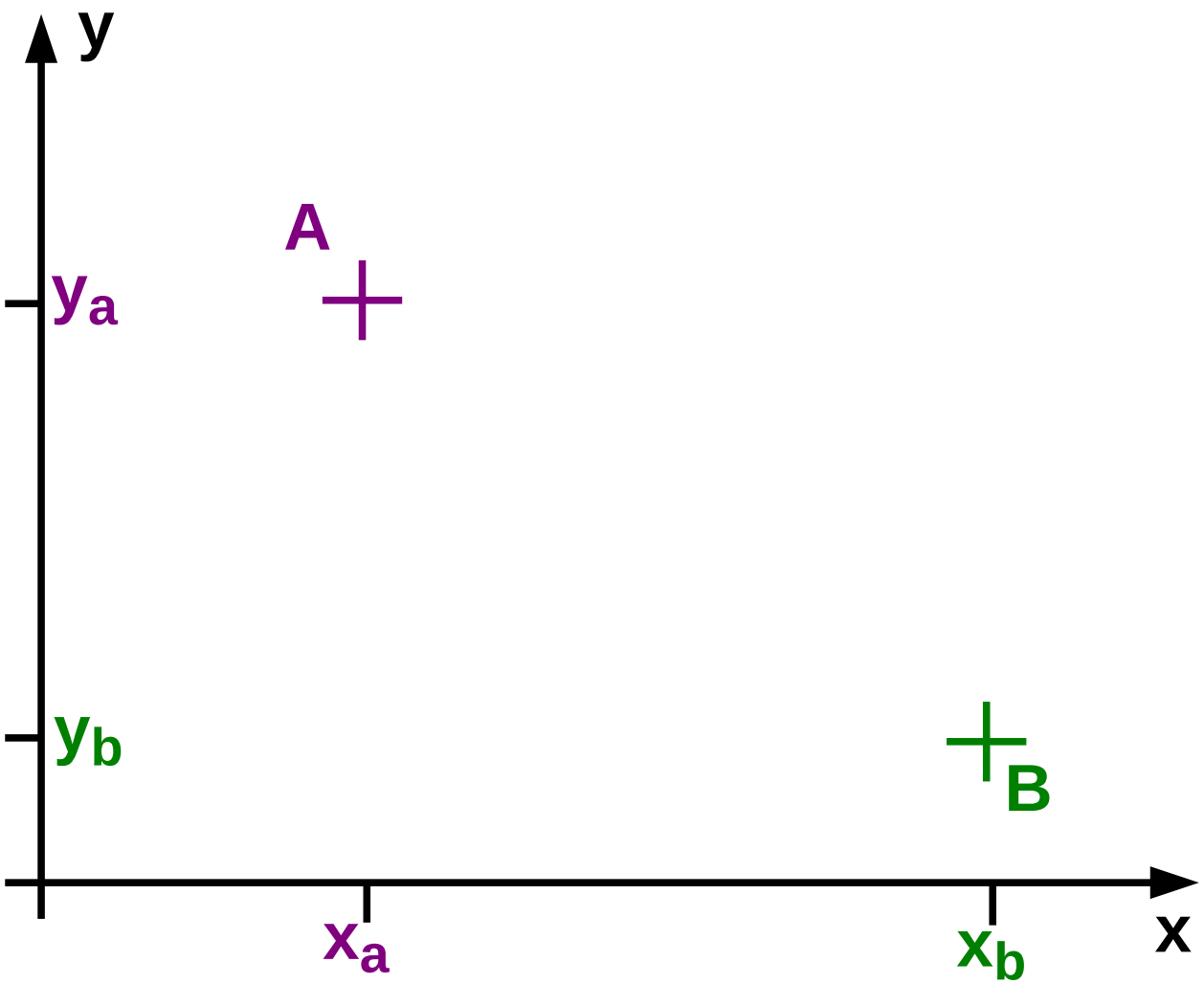
= (x_a | y_a)

B = (x_b , y_b)

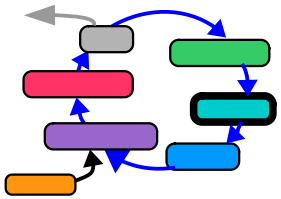
= (x_b | y_b)

Child C1

C1 = (x_a | y_b)



EA: Inheritance



Recombination by 1-point cross over:

Parents A, B

A = (x_a , y_a)

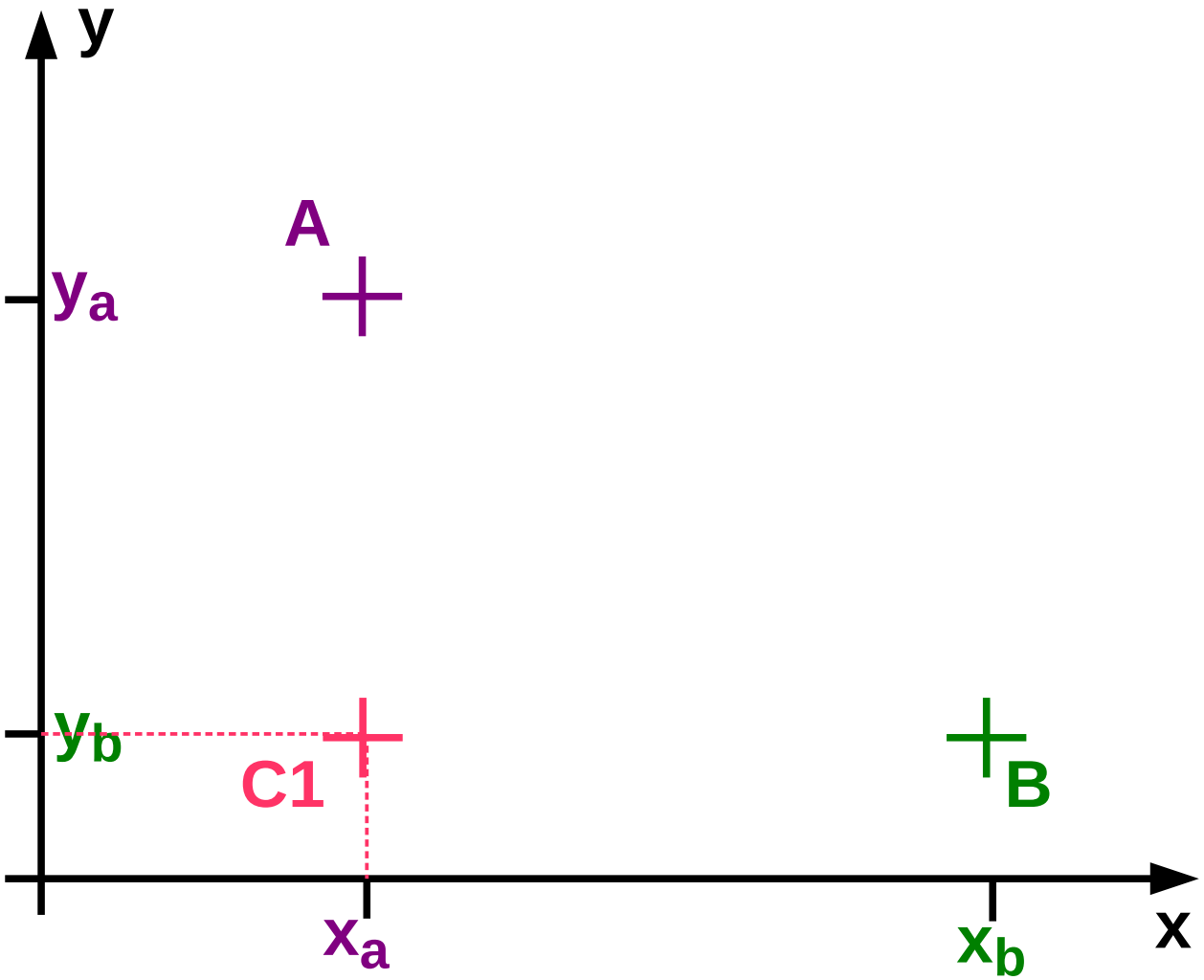
= (x_a | y_a)

B = (x_b , y_b)

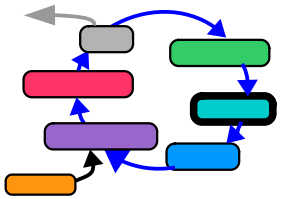
= (x_b | y_b)

Child C1

C1 = (x_a | y_b)



EA: Inheritance



Recombination by 1-point cross over:

Parents A, B

A = (x_a , y_a)

= (x_a | y_a)

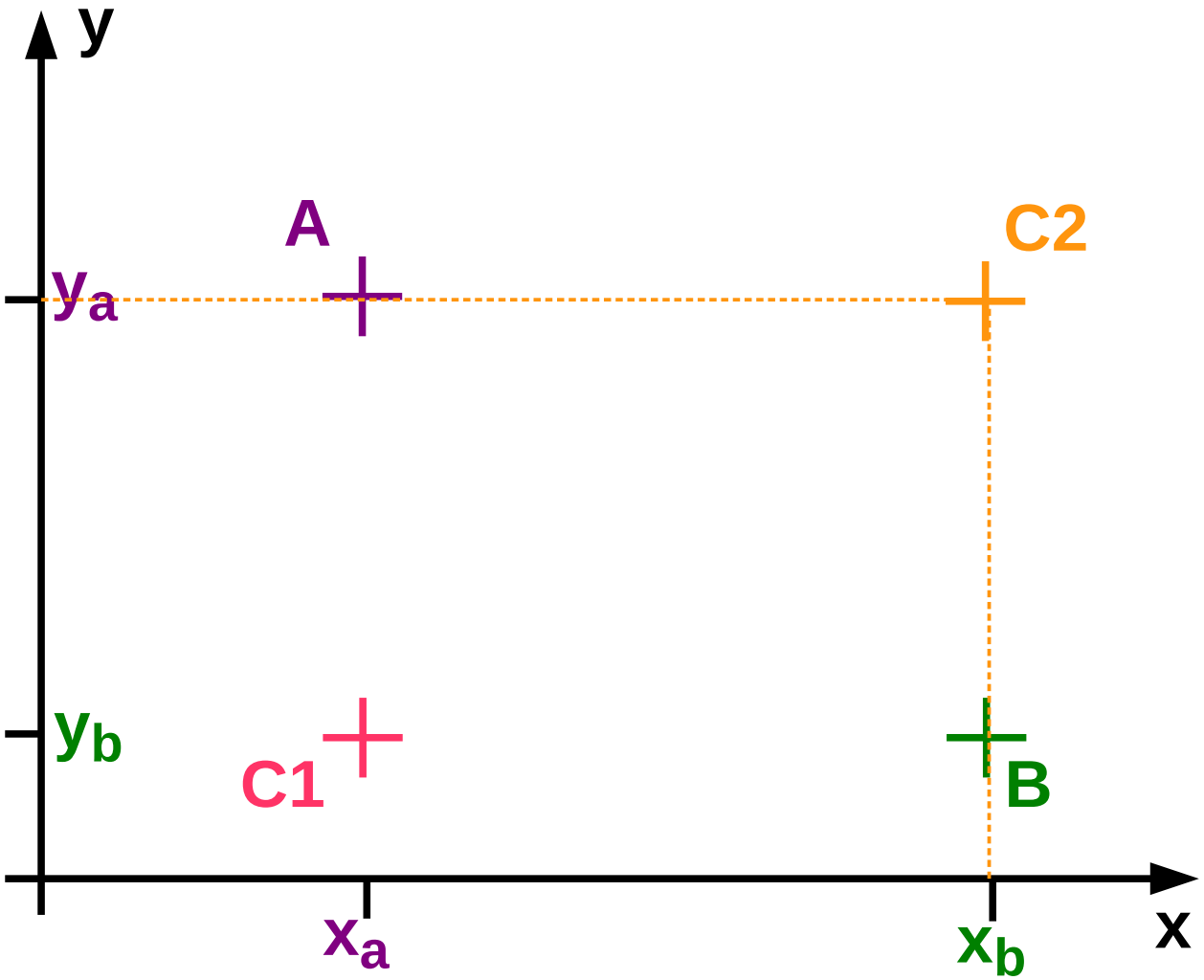
B = (x_b , y_b)

= (x_b | y_b)

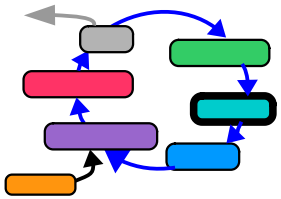
Child C1, C2

C1 = (x_a | y_b)

C2 = (x_b | y_a)



EA: Inheritance



Recombination by 1-point cross over:

Parents A, B

A = (x_a , y_a)

= (x_a | y_a)

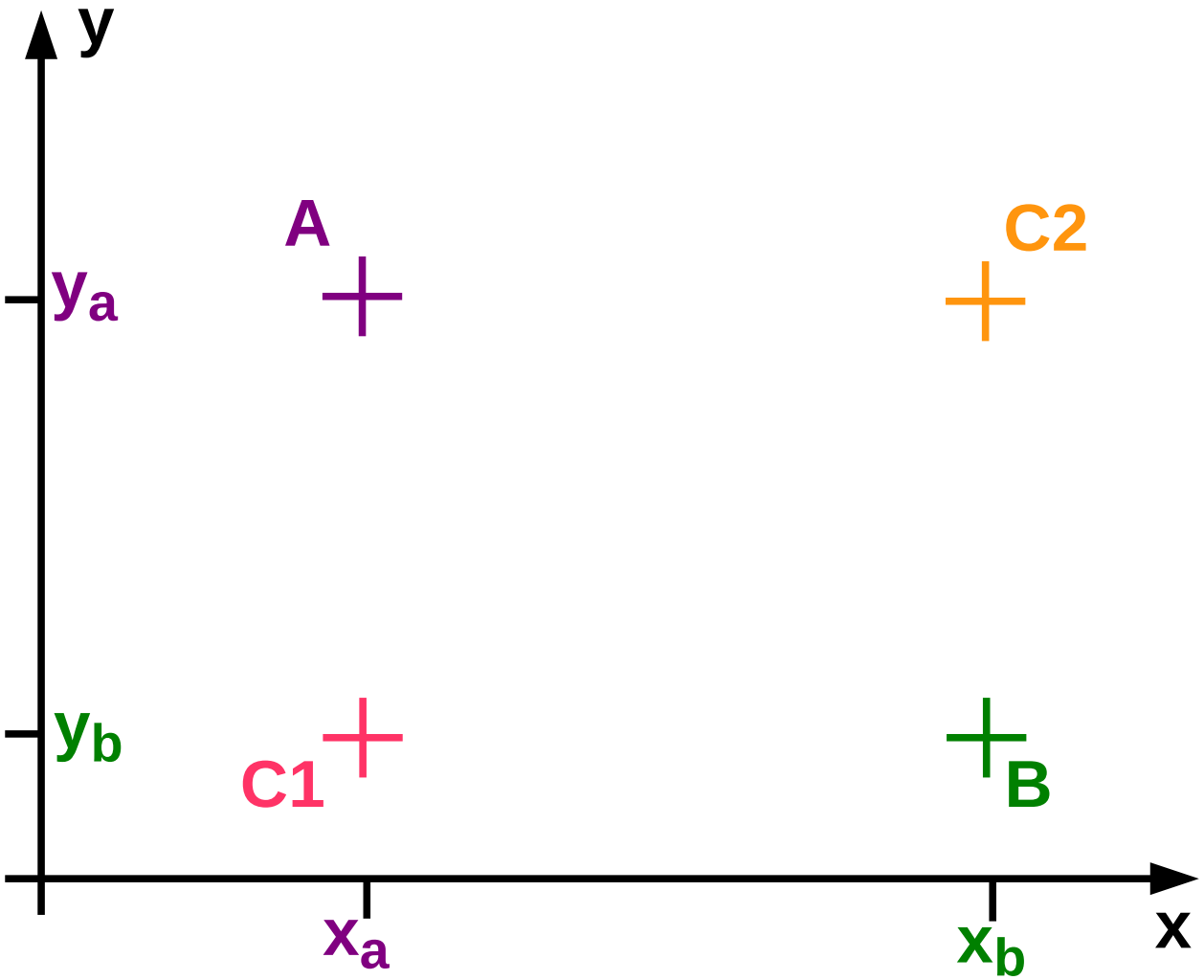
B = (x_b , y_b)

= (x_b | y_b)

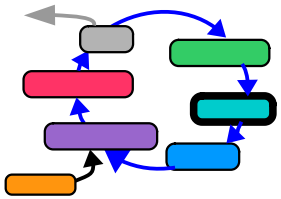
Child C1, C2

C1 = (x_a | y_b)

C2 = (x_b | y_a)



EA: Inheritance



Recombination by 1-point cross over:

Parents A, B

A = (x_a , y_a)

= (x_a | y_a)

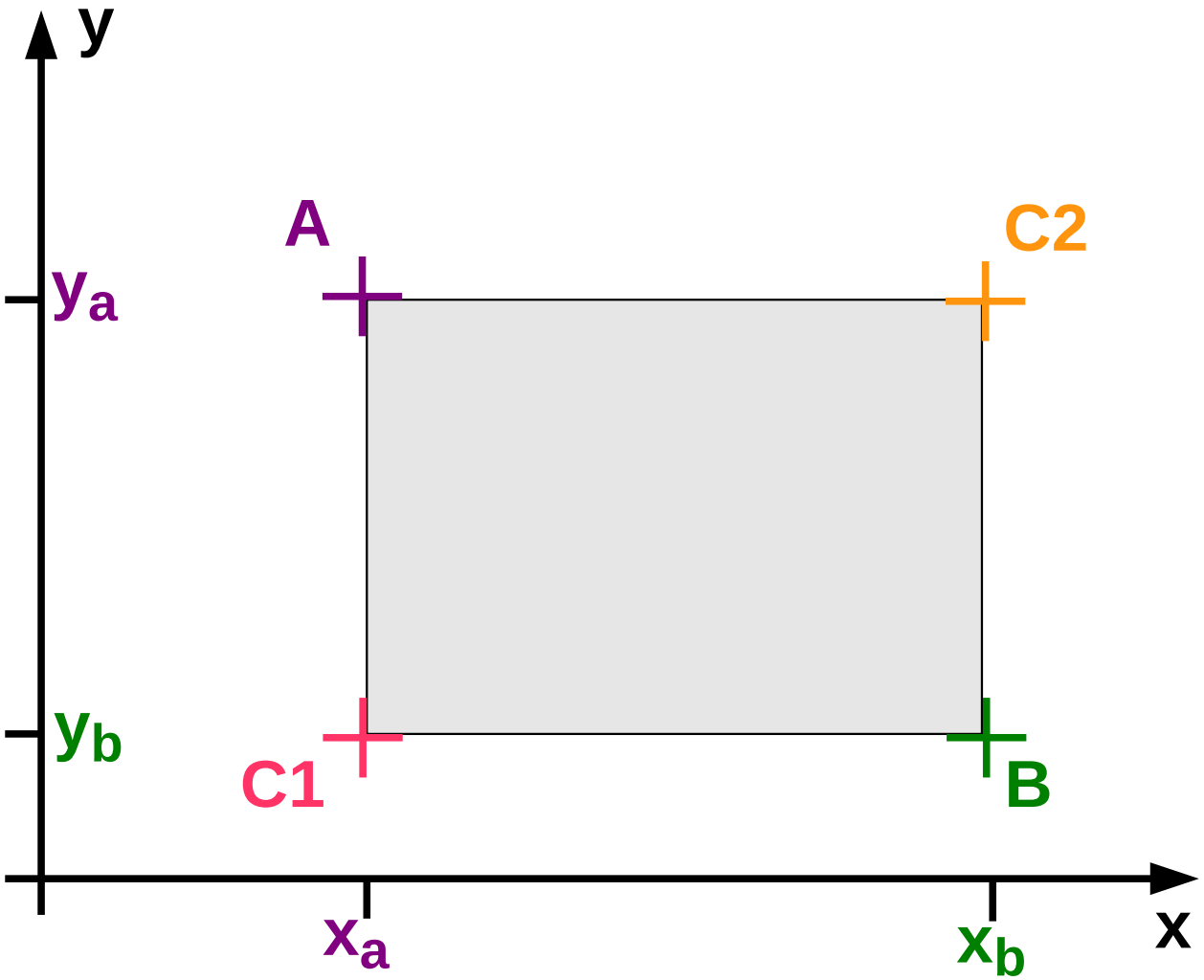
B = (x_b , y_b)

= (x_b | y_b)

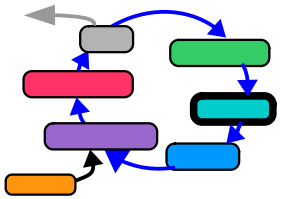
Child C1, C2

C1 = (x_a | y_b)

C2 = (x_b | y_a)



EA: Inheritance

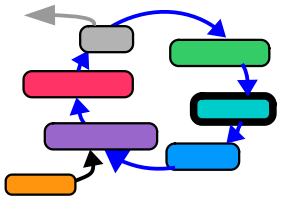


Inheritance can involve **k=**(0),1, **2**, 3, ... or even all μ individuals from the parent generation.

Although, taking k=2 parents is the most popular choice for implementing **inheritance**,
the other possibilities are sensible, and are in use.

- **k=0** parents: no inheritance, generate a novel genome
- **k=1** parent : just copy the genome of the parent
- **k=2** parents: recombination,
  e.g. cross-over, mean value, sum, ...
- **k=3** parents: recombination, e.g. cyclic cross-over
- **k=μ** parents: recombination between all parents

EA: Inheritance



k=0, no inheritance, generate a novel genome.

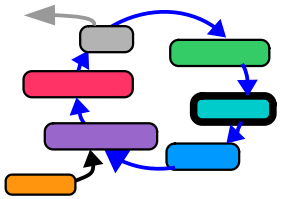
The decision to omit inheritance by generating novel genomes is possible, but not really a good idea for EAs.

Generating new individuals on a random basis, is a very strong implementation of **exploration.**

It can be seen as a special variant of Random Search (RS), (with an extremely large hypersphere).

Typically in EAs, exploration is the role of the subsequent mutation step.

EA: Inheritance



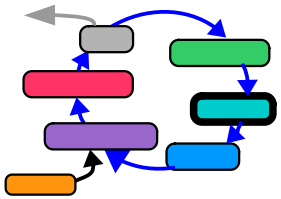
k=1, copy, just copy the genome of the parent.

The decision to make just a **copy** of one of the parents for creating the offspring, is not a bad choice for the inheritance principle.
This is a very strong implementation of **exploitation.**

Just copying, bears the danger of loosing the (necessary) diversity among all genomes within the population.

Only the subsequent EA step (mutation) that will potentially alter the resulting genome, can maintain the diversity to some extent (exploration).

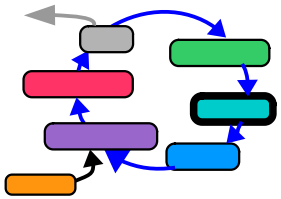
EA: Inheritance



k=2, recombination, combine the genomes of two parents.

The most popular mechanism to implement the **inheritance** principle is to use k=2 parents and to **combine** their genomes (mostly called recombination).

EA: Inheritance



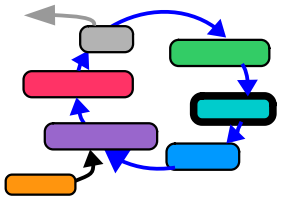
k=2, recombination, combine the genomes of two parents.

The most popular mechanism to implement the **inheritance** principle is to use k=2 parents and to **combine** their genomes (mostly called recombination).

k>2, recombination, combine the genomes of more parents.

Most methods developed and proposed for recombining k=2 parents apply as well for **k>2 parents** with only small alterations.

EA: Inheritance

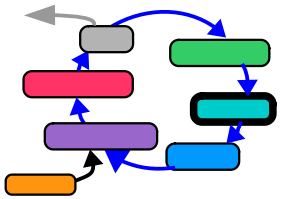


k=2 and k>2, recombination with k parents

Several ways to **combine** (or **re-combine**) the genomes of the k parents have been proposed.
Some of them rely on a special structure of the genome.

- **N-point cross over** (the most popular)
- **mean-values,** one component, all components
- **max, min, sum, difference** of values
- **union, intersection** of **sets** of items
- **concatenate** (strings)
- **...**

EA: Inheritance



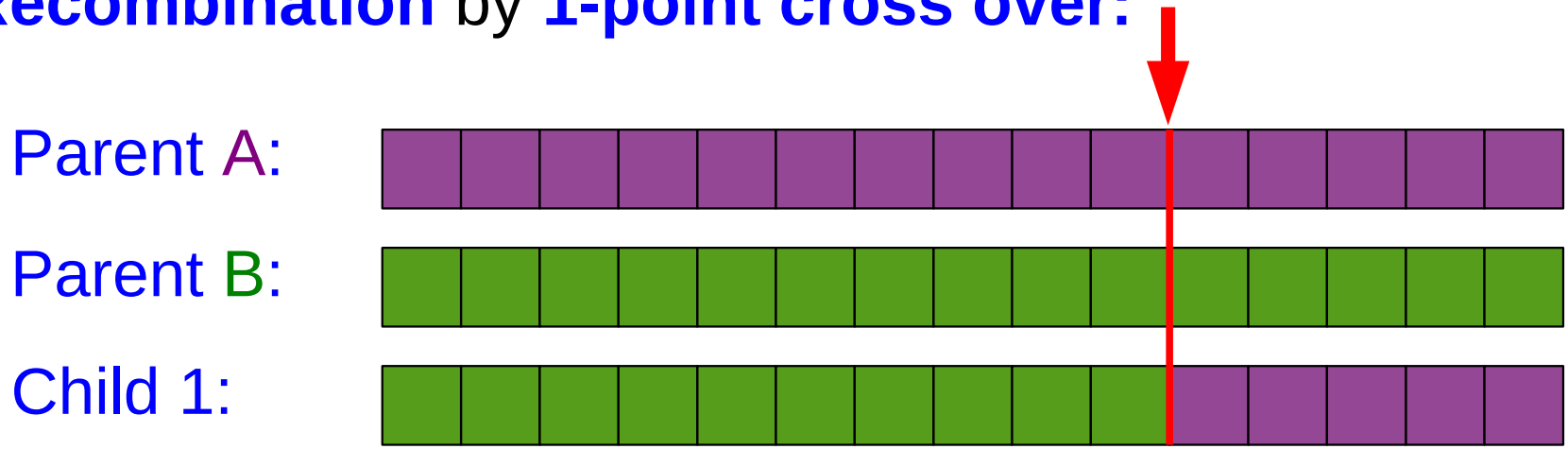
Recombination by **N-point cross over**:

The **N-point cross over** combination operator, produces offspring that lie only on the corners of a hypercube.

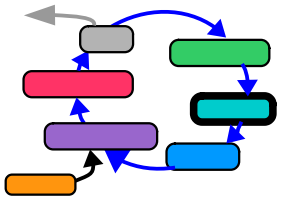
This is a reduced search space to look for solutions.

EA: Inheritance

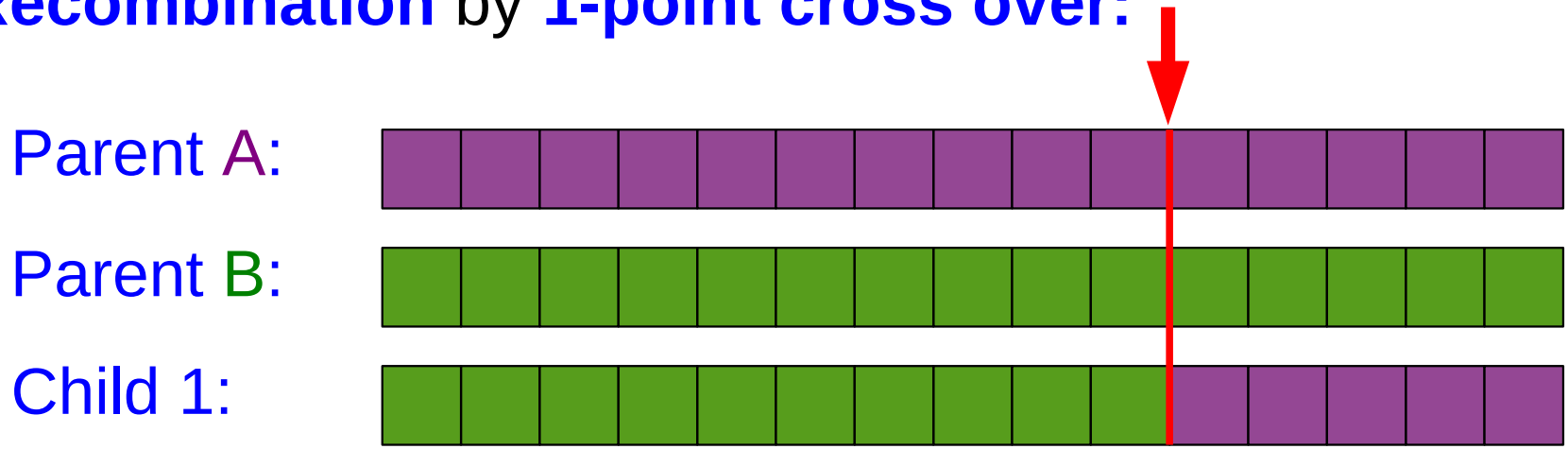
Recombination by 1-point cross over:



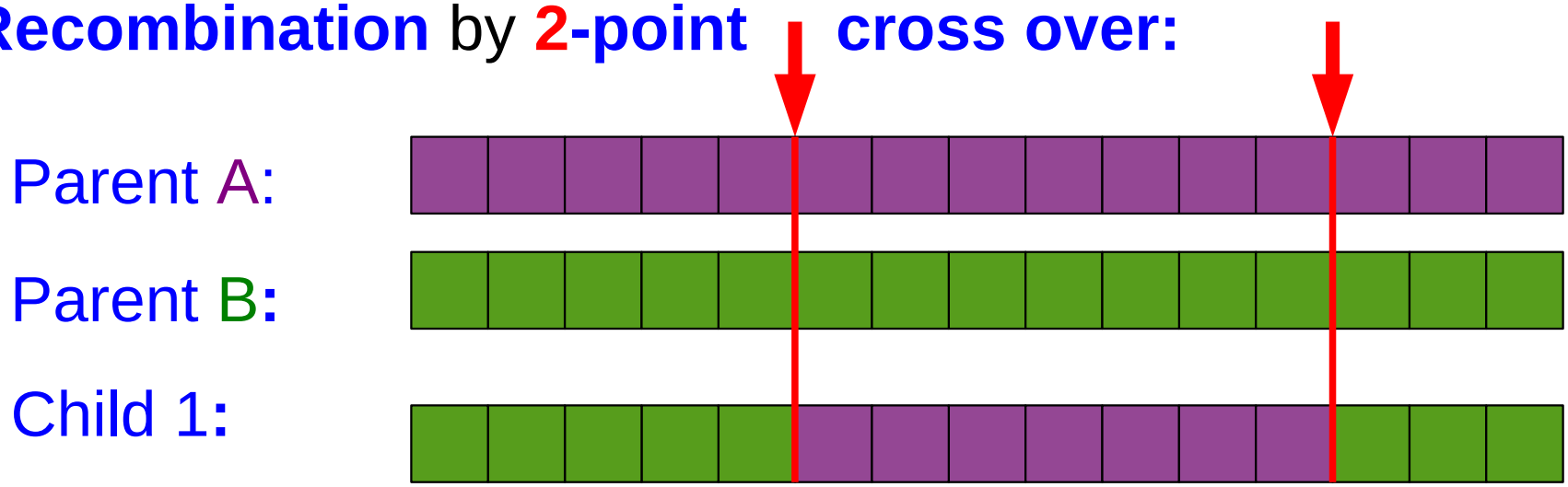
EA: Inheritance



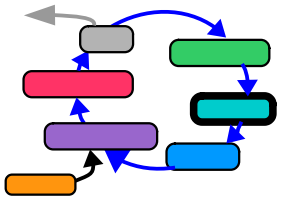
Recombination by 1-point cross over:



Recombination by 2-point cross over:



EA: Inheritance

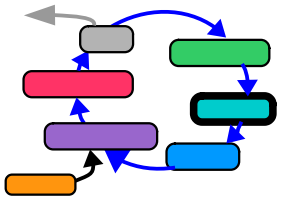


k=2 and k>2, recombination with k parents

Several ways to **combine** (or **re-combine**) the genomes of the k parents have been proposed.
Some of them rely on a special structure of the genome.

- **N-point cross over** (the most popular)
- **mean-values,** one component, all components
- **max, min, sum, difference** of values
- **union, intersection** of **sets** of items
- **concatenate** (strings)
- **...**

EA: Inheritance

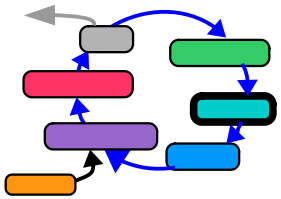


Recombination by point wise **mean values:**

Select two parents, **A** and **B** and a random position **R** within the genome, and **recombine** the genomes by building the **mean value** between **corresponding parts** at position **R** of the genomes.

Pointwise numeric operations, like:
mean value, sum, **difference**, **maximum** or **minimum** are only feasible, if the different parts of the genomes are comparable and numeric.

EA: Inheritance



Recombination by point wise mean values

Parent A:	0.333	144.6	42	92	4.75	5926	10716
Parent B:	0.123	1.40	40	42	1.532	5926	0
Child C1:							

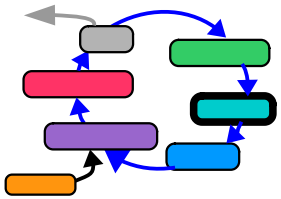
EA: Inheritance

Recombination by point wise mean values

Parent A:	0.333	144.6	42	92	4.75	5926	10716
Parent B:	0.123	1.40	40	42	1.532	5926	0
Child C1:							

R

EA: Inheritance



Recombination by point wise mean values

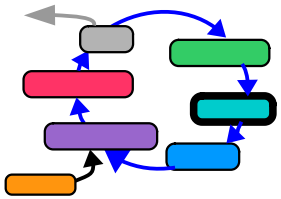
Parent A:	0.333	144.6	42	92	4.75	5926	10716
Parent B:	0.123	1.40	40	42	1.532	5926	0

R

½ (4.75 + 1.532) = ½ (6.282) = 3.141

Child C1:							

EA: Inheritance



Recombination by point wise mean values

Parent A:	0.333	144.6	42	92	4.75	5926	10716
Parent B:	0.123	1.40	40	42	1.532	5926	0

R

½ (4.75 + 1.532) = ½ (6.282) = 3.141

Child C1:					3.141		

EA: Inheritance

Recombination by point wise mean values

Parent A:	0.333	144.6	42	92	4.75	5926	10716
Parent B:	0.123	1.40	40	42	1.532	5926	0
Child C1:					3.141		

R

EA: Inheritance

Recombination by point wise mean values

Parent A:	0.333	144.6	42	92	4.75	5926	10716
Parent B:	0.123	1.40	40	42	1.532	5926	0
Child C1:	0.123	1.40	40	42	3.141	5926	0

R

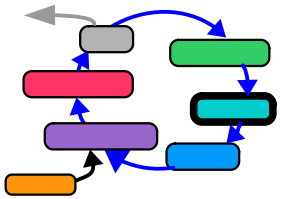
EA: Inheritance

Recombination by point wise mean values

Parent A:	0.333	144.6	42	92	4.75	5926	10716
Parent B:	0.123	1.40	40	42	1.532	5926	0
Child C1:	0.123	1.40	40	42	3.141	5926	0

R

EA: Inheritance



Recombination by point wise mean values

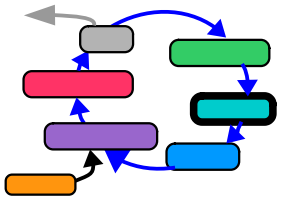
Parent A:	0.333	144.6	42	92	4.75	5926	10716
Parent B:	0.123	1.40	40	42	1.532	5926	0

R

mean value in one component R

Child C1:	0.123	1.40	40	42	3.141	5926	0

EA: Inheritance



Recombination by point wise mean values

Parent A:	0.333	144.6	42	92	4.75	5926	10716
Parent B:	0.123	1.40	40	42	1.532	5926	0

mean value in one component R

Child C1:	0.123	1.40	40	42	3.141	5926	0

mean value in all components

Child C2:							

EA: Inheritance

Recombination by point wise mean values

Parent A:	0.333	144.6	42	92	4.75	5926	10716
Parent B:	0.123	1.40	40	42	1.532	5926	0

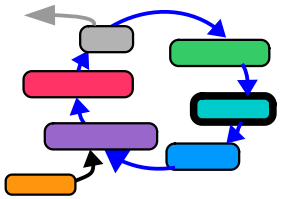
mean value in one component R

Child C1:	0.123	1.40	40	42	3.141	5926	0

mean value in all components

Child C2:	0.228	73	41	67	3.141	5926	5358

EA: Inheritance

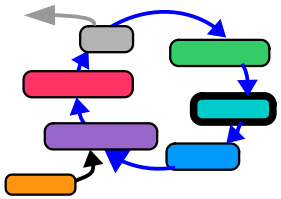


k=2 and k>2, recombination with k parents

Several ways to **combine** (or **re-combine**) the genomes of the k parents have been proposed.
Some of them rely on a special structure of the genome.

- **N-point cross over** (the most popular)
- **mean-values,** one component, all components
- **max, min, sum, difference** of values
- **union, intersection** of **sets** of items
- **concatenate** (strings)
- **...**

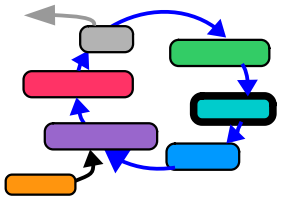
EA: Inheritance



Recombination by union of sets

For genomes that are structured as a set of items, a valid and sensible **recombination** is to build the **union** of those sets.

EA: Inheritance



Recombination by union of sets

For genomes that are structured as a set of items, a valid and sensible **recombination** is to build the **union** of those sets.

Parent A: **{ Soup, Pasta, Fish, White-Wine }**

Parent B: **{ Aperitif, White-Wine, Water, Cheese }**

EA: Inheritance

Recombination by union of sets

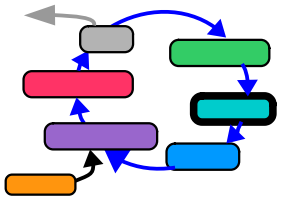
For genomes that are structured as a set of items, a valid and sensible **recombination** is to build the **union** of those sets.

Parent A: { Soup, Pasta, Fish, White-Wine }

Parent B: { Aperitif, White-Wine, Water, Cheese }

Child C4: { Soup, Pasta, Fish, White-Wine, Aperitif, Water, Cheese }

EA: Inheritance

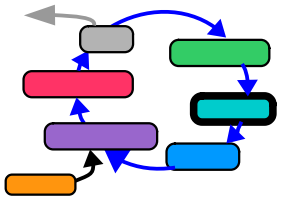


k=2 and k>2, recombination with k parents

Several ways to **combine** (or **re-combine**) the genomes of the k parents have been proposed.
Some of them rely on a special structure of the genome.

- **N-point cross over** (the most popular)
- **mean-values,** one component, all components
- **max, min, sum, difference** of values
- **union, intersection** of **sets** of items
- **concatenate** (strings)
- **...**

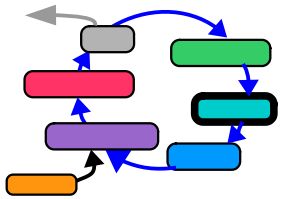
EA: Inheritance



Recombination by concatenation

Genomes that are representing **strings,** can be **recombined** by a simple **concatenation** of their strings, just „**gluing**“ the genomes one after the other.
In this case, the order is important.

EA: Inheritance



Recombination by concatenation

Genomes that are representing **strings,** can be **recombined** by a simple **concatenation** of their strings, just „**gluing**“ the genomes one after the other.
In this case, the order is important.

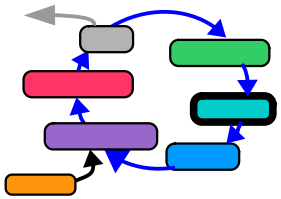
Parent A: „**Throatwobbler** “

Parent B: „**Mangrove**“

Child LY: „**Throatwobbler Mangrove** “

From: Monty Python's Flying Circus: Episode 22, Cosmetic surgery

EA: Inheritance

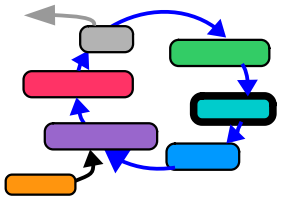


k=2 and k>2, recombination with k parents

Several ways to **combine** (or **re-combine**) the genomes of the k parents have been proposed.
Some of them rely on a special structure of the genome.

- **N-point cross over** (the most popular)
- **mean-values,** one component, all components
- **max, min, sum, difference** of values
- **union, intersection** of **sets** of items
- **concatenate** (strings)
- **...**

EA: Inheritance



Special forms of **genome coding** can require special **recombination operators**.

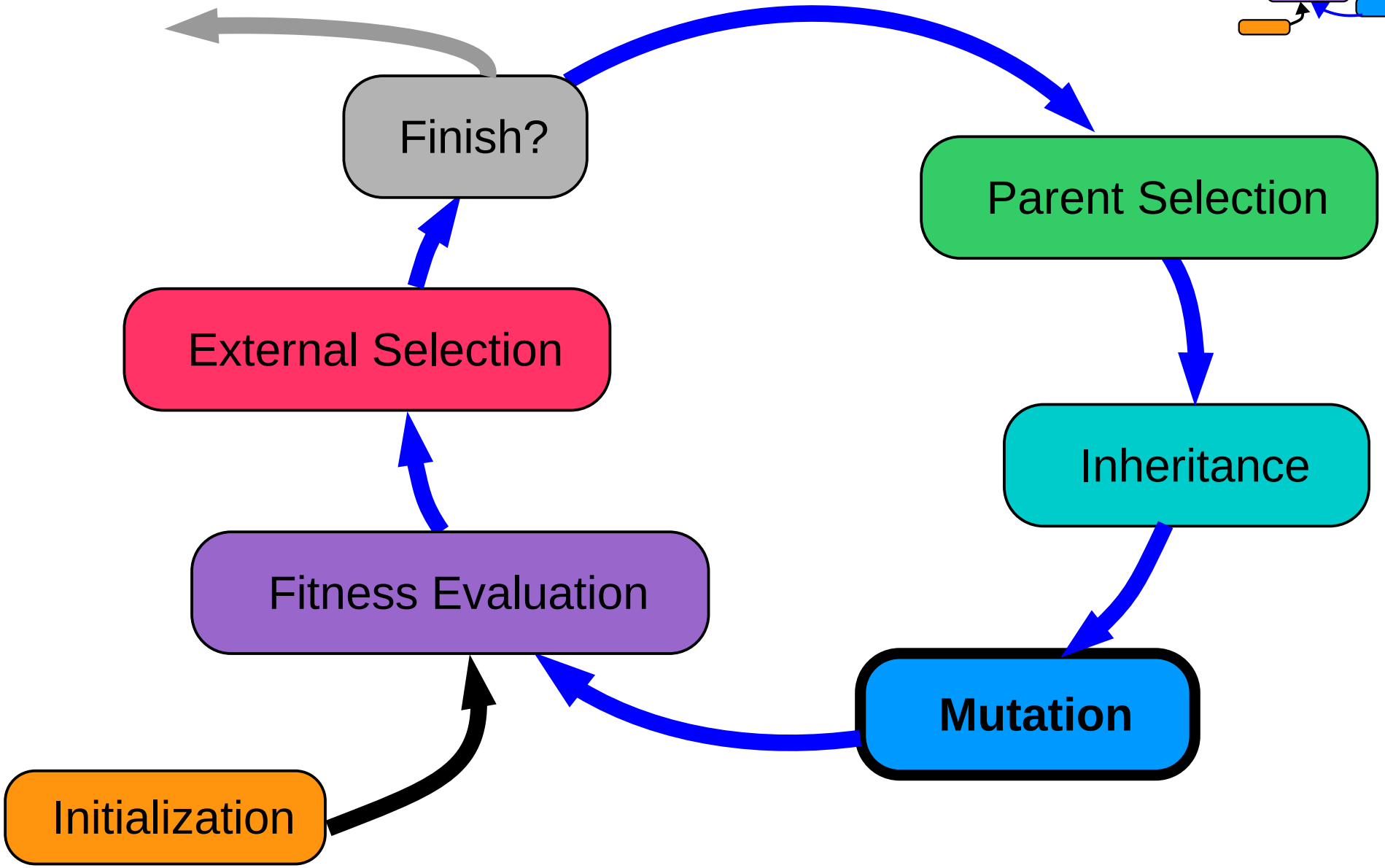
In some cases, the Genome is coded with a special semantics aligned with the application.
The **inheritance**, **recombination operator** **can** pay respect to this fact,
and **can** therefore be structured in this way,
but it **does not have to** .

There are good arguments, to regard the genome as a complete context free binary vector.

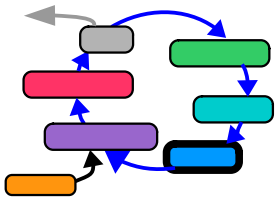
Overview

- EA Steps
  - Individual, Genome, Fitness, Population
  - Parent selection
  - Inheritance
  - **Mutation**
  - Fitness evaluation
  - External selection
  - Finish?
  - Initialization
- Strategy
- Performance Graph
- Genome Structure
- Examples

EA cycle of operations:



EA: Mutation

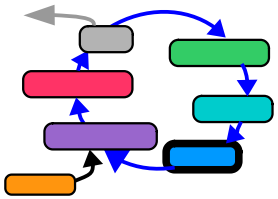


The job of the **mutation** step is to maintain the **diversity** of the genomes within the population:

Different positions in search space sample the fitness function at different points.

Changing the genome by **mutation** is implementing the **exploration** principle of **stochastic optimization.**

EA: Mutation



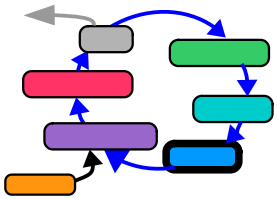
Depending on the different way the information is coded in the genome, the layout of the **mutation** step is to be designed differently.

Some common mutation operators are:

- **Binary genome**: a bit-flip
- **Vector of parameters**: change of value
- **Set of items**: replace a single item
- **String of characters**: replace a single character
- **Sequence**: change order within sequence
- **Other:** other method

Special codings of the genome will require that the **mutation** is paying respect to this coding.

EA: Mutation



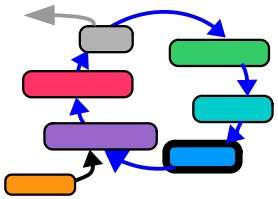
The **mutation** for a binary genome is typically implemented as one or more **bit-flip**s:

Method 1:
Choose a random position **r** in the genome and flip that bit.
This yields a 100% chance that the genome is changed;
the resulting Hamming distance is always exactly 1.

Before:

O	I	O	I	O	O	O	I	O	O	O	I	I	I

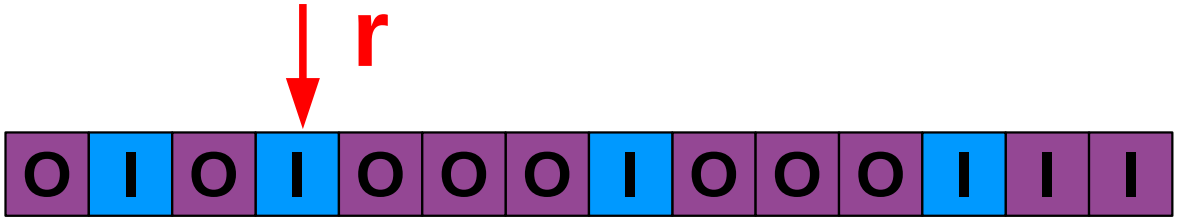
EA: Mutation



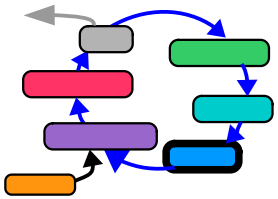
The **mutation** for a binary genome is typically implemented as one or more **bit-flip**s:

Method 1:
Choose a random position **r** in the genome and flip that bit.
This yields a 100% chance that the genome is changed;
the resulting Hamming distance is always exactly 1.

Before:

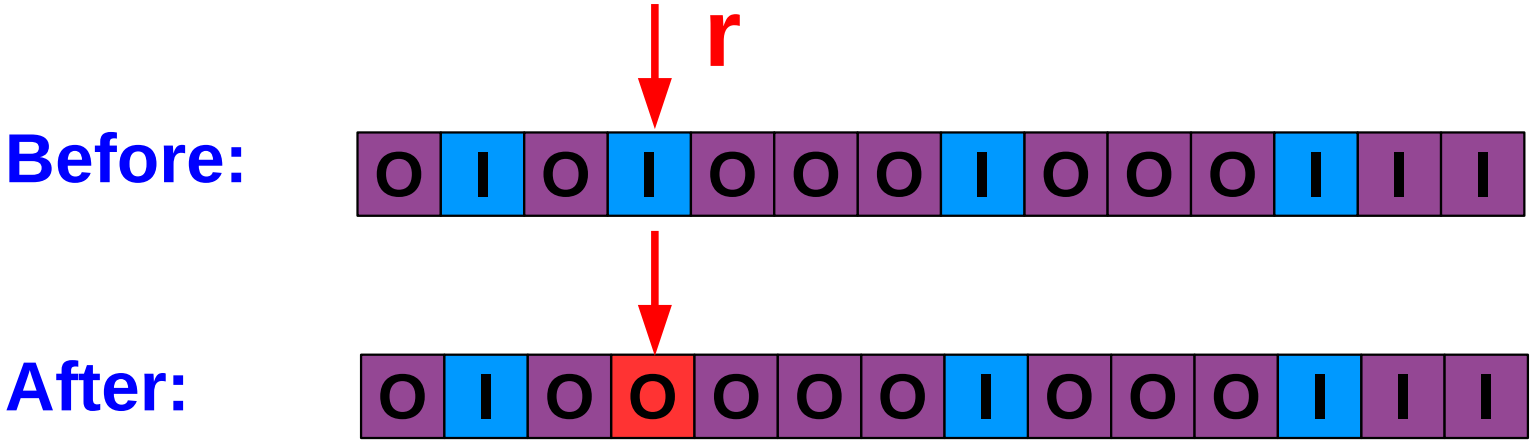


EA: Mutation

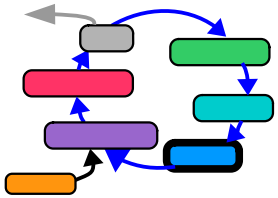


The **mutation** for a binary genome is typically implemented as one or more **bit-flip**s:

Method 1:
Choose a random position **r** in the genome and flip that bit.
This yields a 100% chance that the genome is changed;
the resulting Hamming distance is always exactly 1.



EA: Mutation



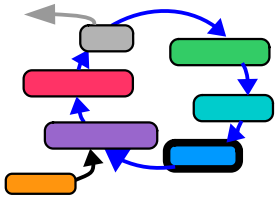
The **mutation** for a binary genome is typically implemented as one or more **bit-flip**s:

Method 2:
Go through all positions in the genome, and **flip** each **bit** with a given **bit-flip**-probability ω .
There is a chance, that all bits are flipped,
and a chance that none is flipped.

Before:

I	I	O	I	I	I	I	O	I	O	O	O	I	O

EA: Mutation

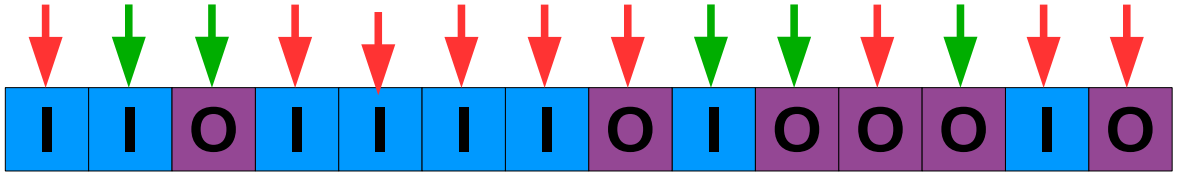


The **mutation** for a binary genome is typically implemented as one or more **bit-flip**s:

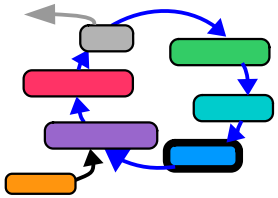
Method 2:
Go through all positions in the genome, and **flip** each **bit** with a given **bit-flip**-probability ω .
There is a chance, that all bits are flipped, and a chance that none is flipped.

$0.0 < \omega < 1.0$

Before:



EA: Mutation

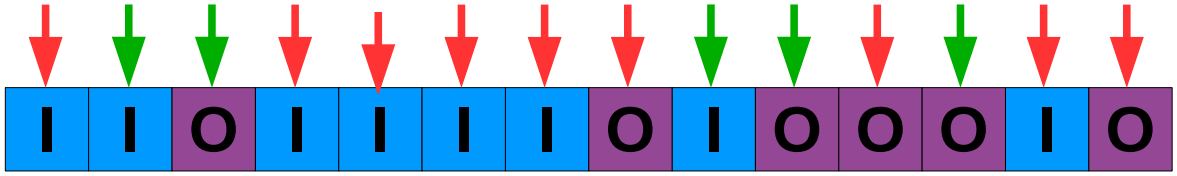


The **mutation** for a binary genome is typically implemented as one or more **bit-flip**s:

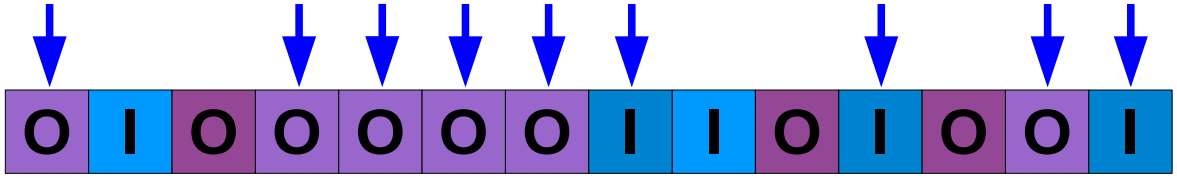
Method 2:
Go through all positions in the genome, and **flip** each **bit** with a given **bit-flip**-probability ω .
There is a chance, that all bits are flipped, and a chance that none is flipped.

$0.0 < \omega < 1.0$

Before:



After:



EA: Mutation & Inheritance

It is possible, that the inheritance and the mutation process yield invalid genomes, which represent hypotheses **s** from outside the allowed search space **S**.

Those illegal genomes would stress the optimization process in an unwanted way.

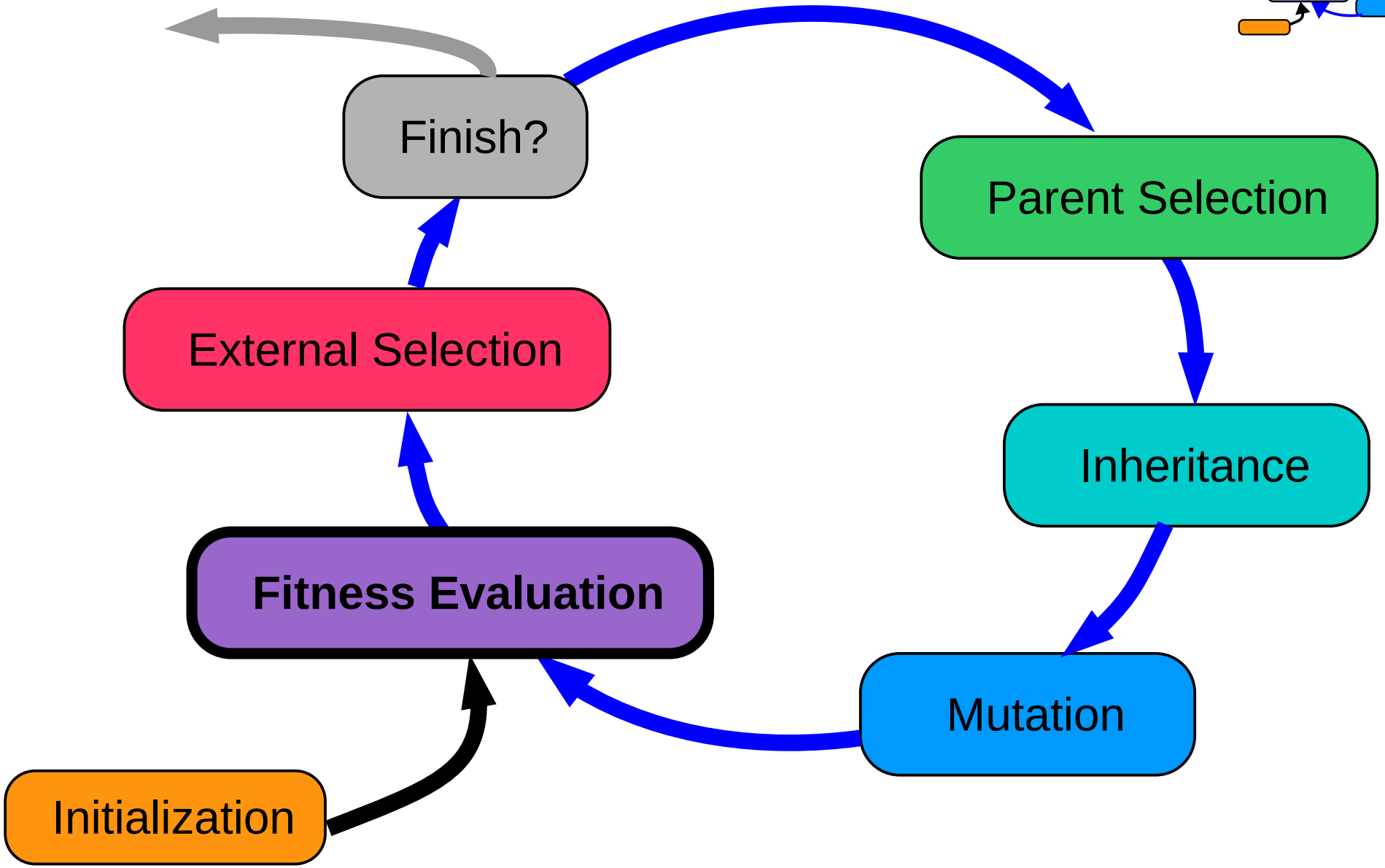
Therefore, it is a good advise, to take precautions against illegal genomes (if possible, and economic):

- shape the structure of the genome,
- shape the structure of the inheritance,
- shape the structure of the mutation.

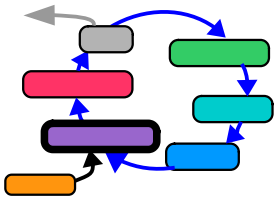
Overview

- EA Steps
  - Individual, Genome, Fitness, Population
  - Parent selection
  - Inheritance
  - Mutation
  - **Fitness evaluation**
  - External selection
  - Finish?
  - Initialization
- Strategy
- Performance Graph
- Genome Structure
- Examples

EA cycle of operations:

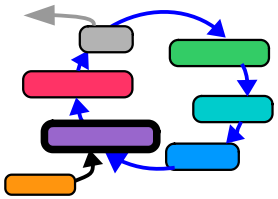


EA: Fitness Evaluation



The **fitness** of each individual is evaluated with respect to the **fitness-function** (most case identical to the objective function). For EAs, it is usual to have a **fitness** function that is to be maximized.

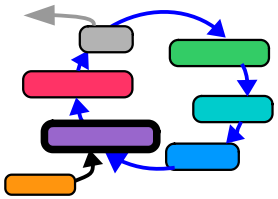
EA: Fitness Evaluation



The **fitness** of each individual is evaluated with respect to the **fitness-function** (most case identical to the objective function). For EAs, it is usual to have a **fitness** function that is to be maximized.

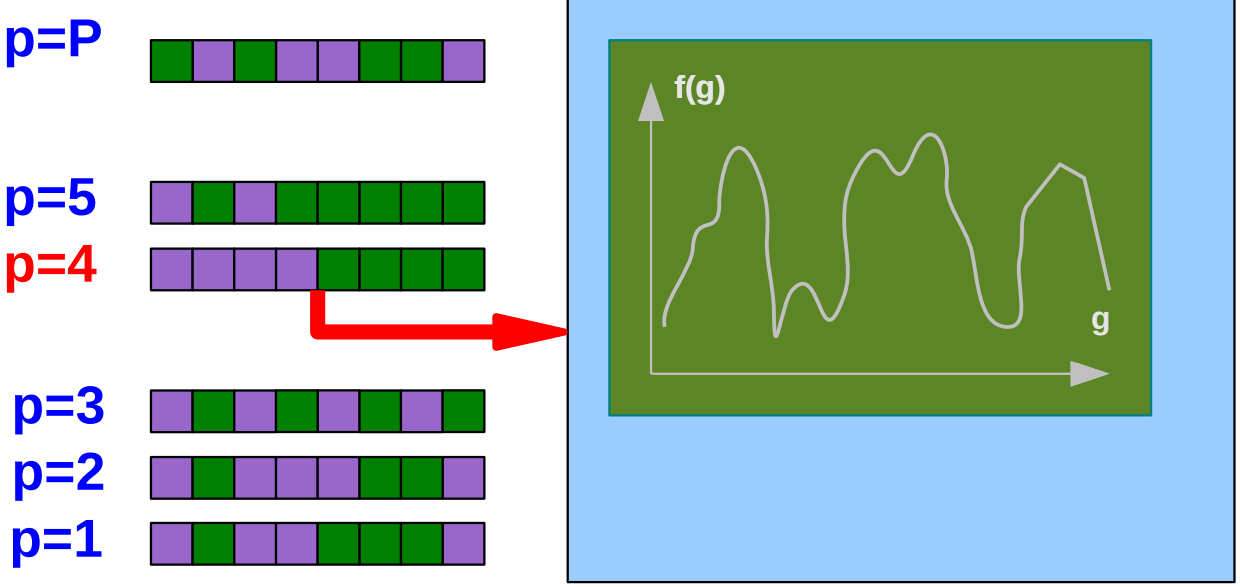
If the **fitness evaluation** is expensive, it is a good idea to process only the new individuals/genomes from the inheritance step, including those individuals/genomes that have changed during mutation.

EA: Fitness Evaluation



The **fitness $f(g_p)$** of each individual **p** is evaluated with respect to the application using the given **fitness-function f(g)**.

Genome g_p



Fitness Value $f(g_p)$

$f(g_p) =$

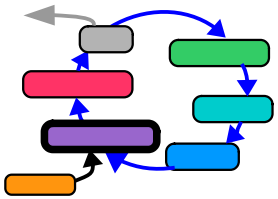
$f(g_4) =$

$f(g_3) = 0.228$

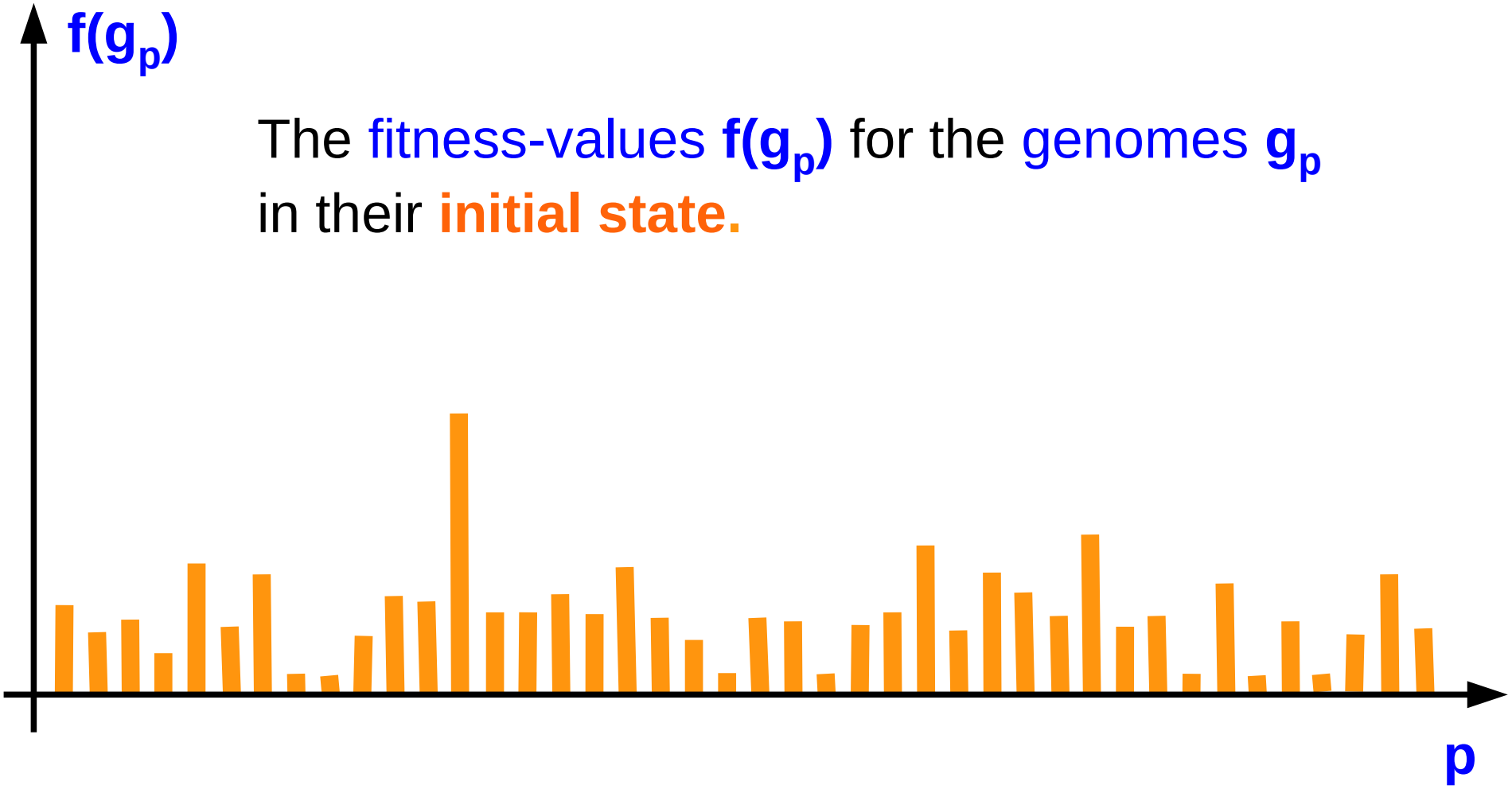
$f(g_2) = -171.0$

$f(g_1) = 0.5$

EA: Fitness Evaluation

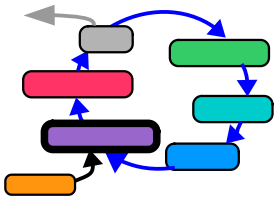


Distribution of **fitness values** over the population.

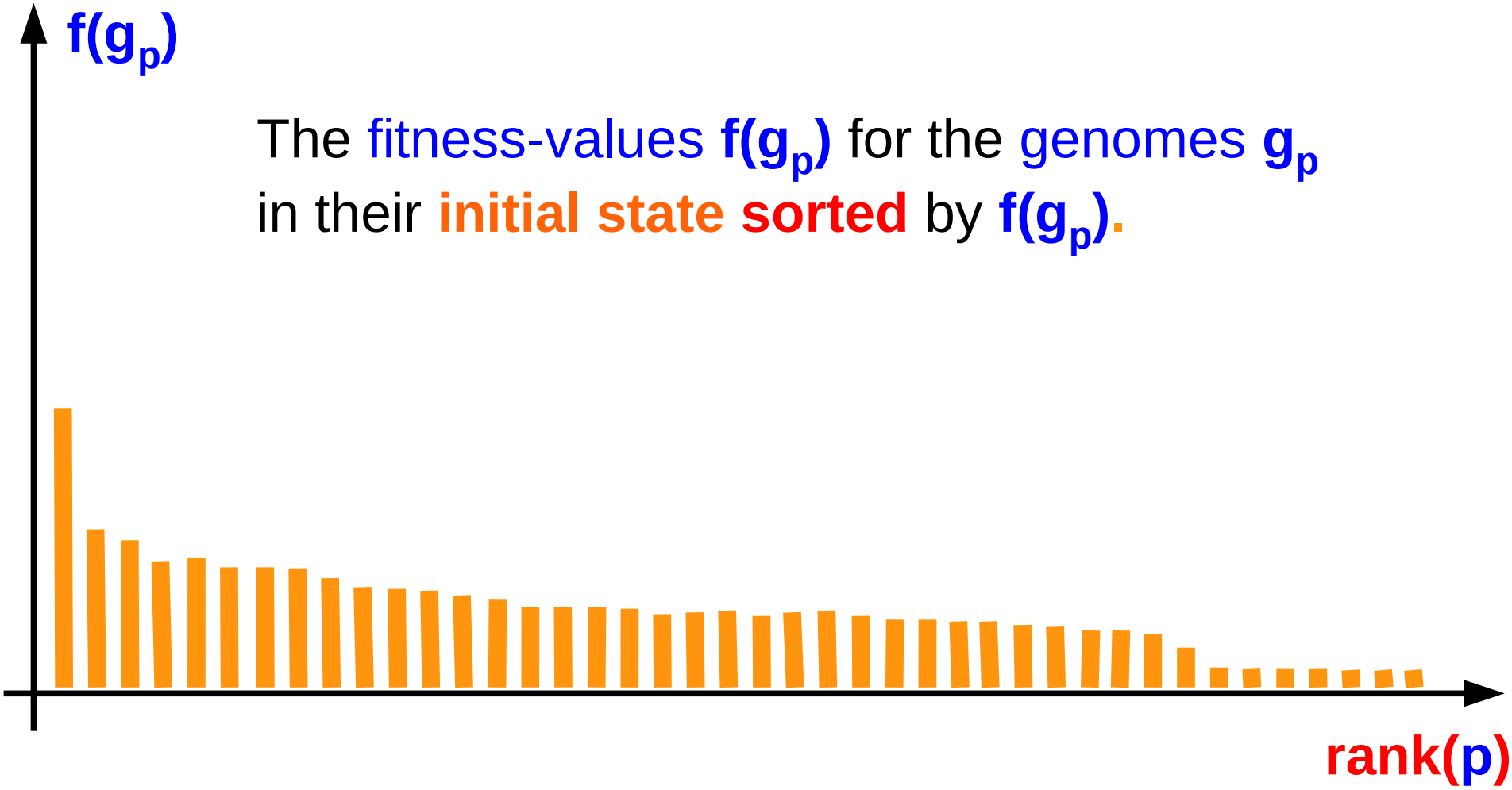


The fitness-values $f(g_p)$ for the genomes g_p in their **initial state.**

EA: Fitness Evaluation

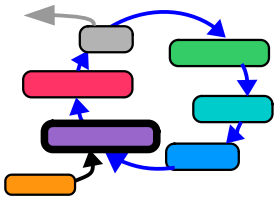


Distribution of **fitness values** over the population.

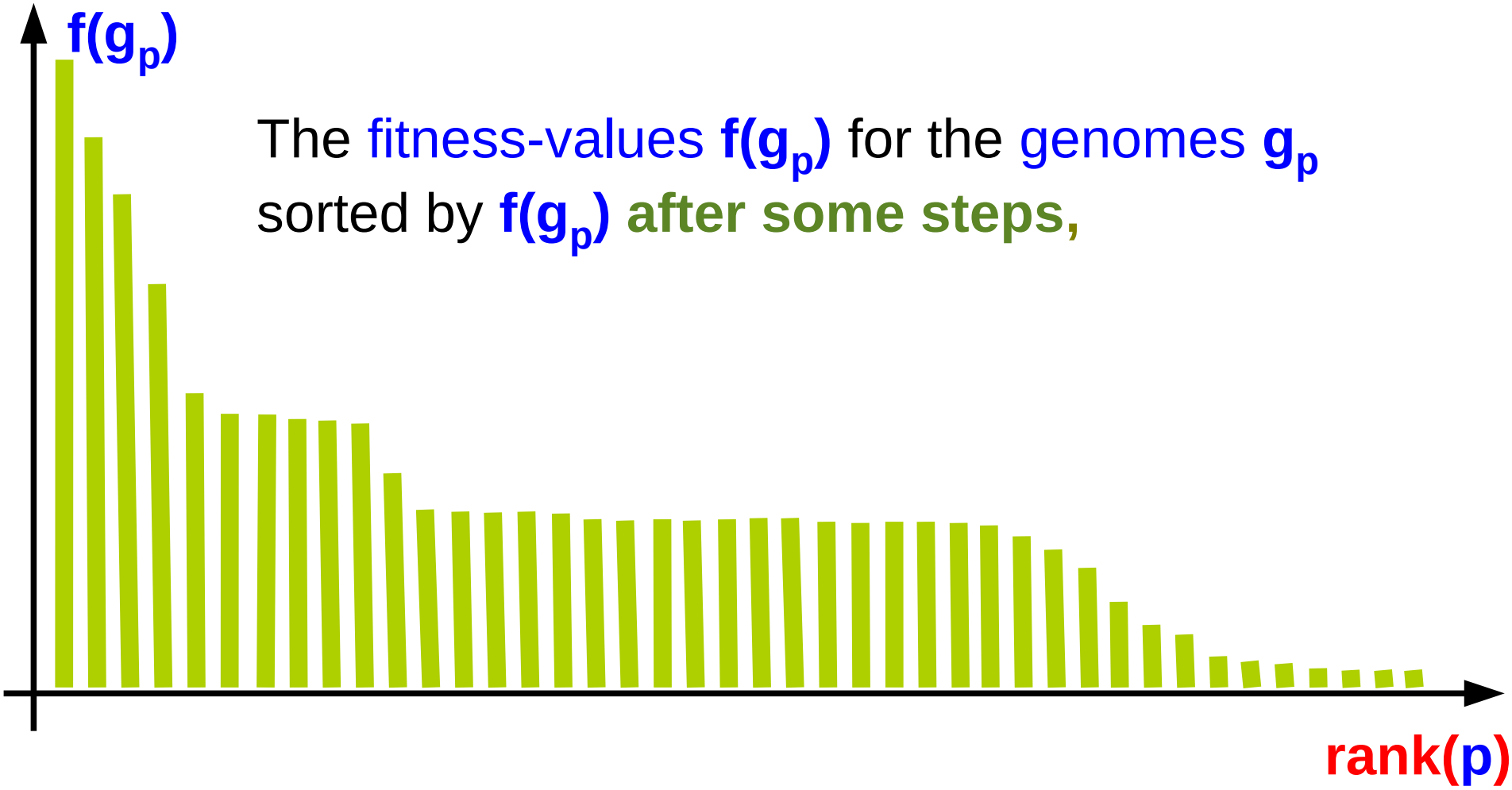


The fitness-values $f(g_p)$ for the genomes g_p in their **initial state sorted** by $f(g_p)$.

EA: Fitness Evaluation

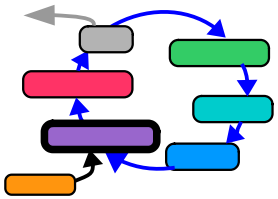


Distribution of **fitness values** over the population.

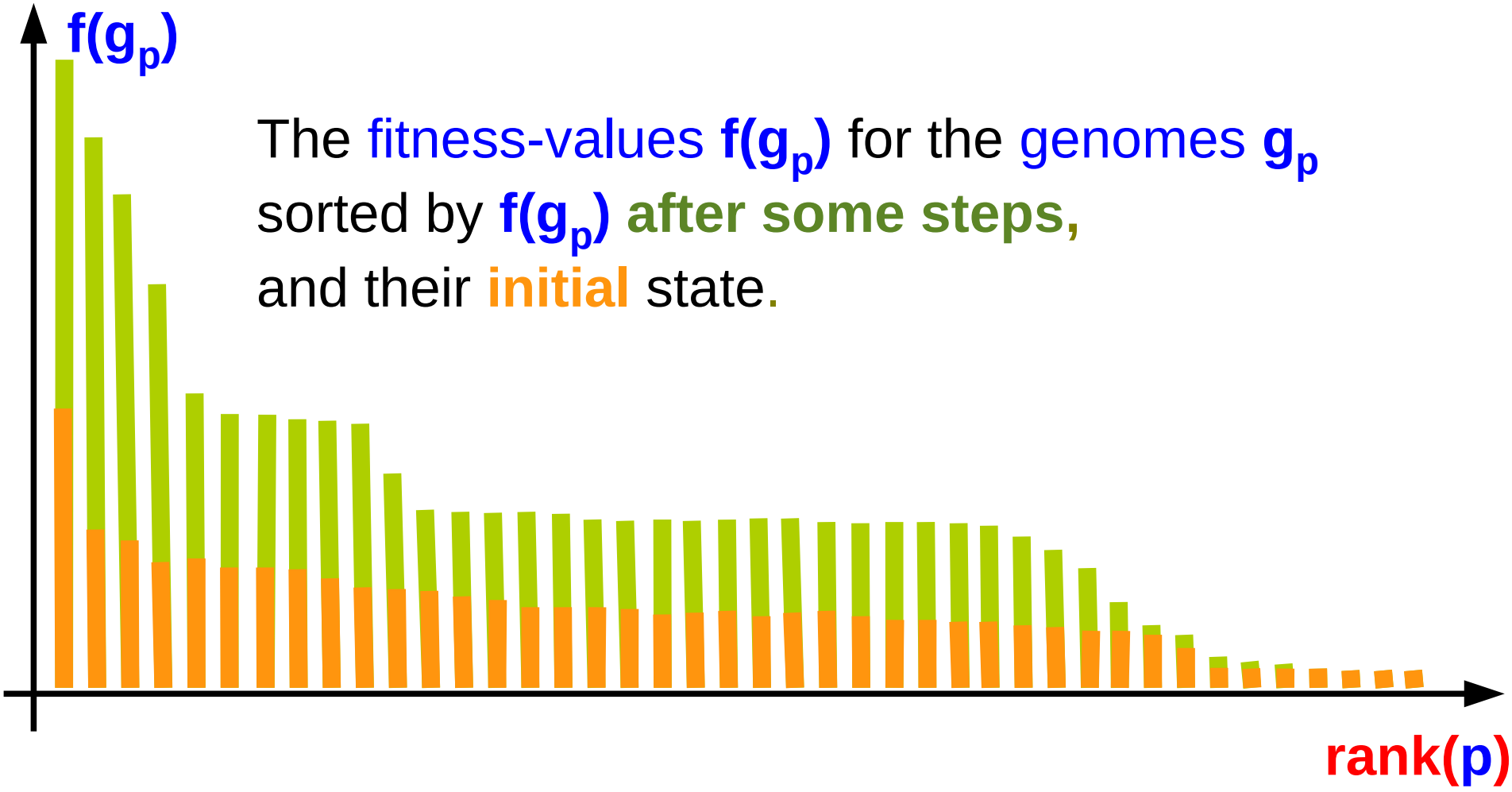


The fitness-values $f(g_p)$ for the genomes g_p sorted by $f(g_p)$ **after some steps,**

EA: Fitness Evaluation



Distribution of **fitness values** over the population.

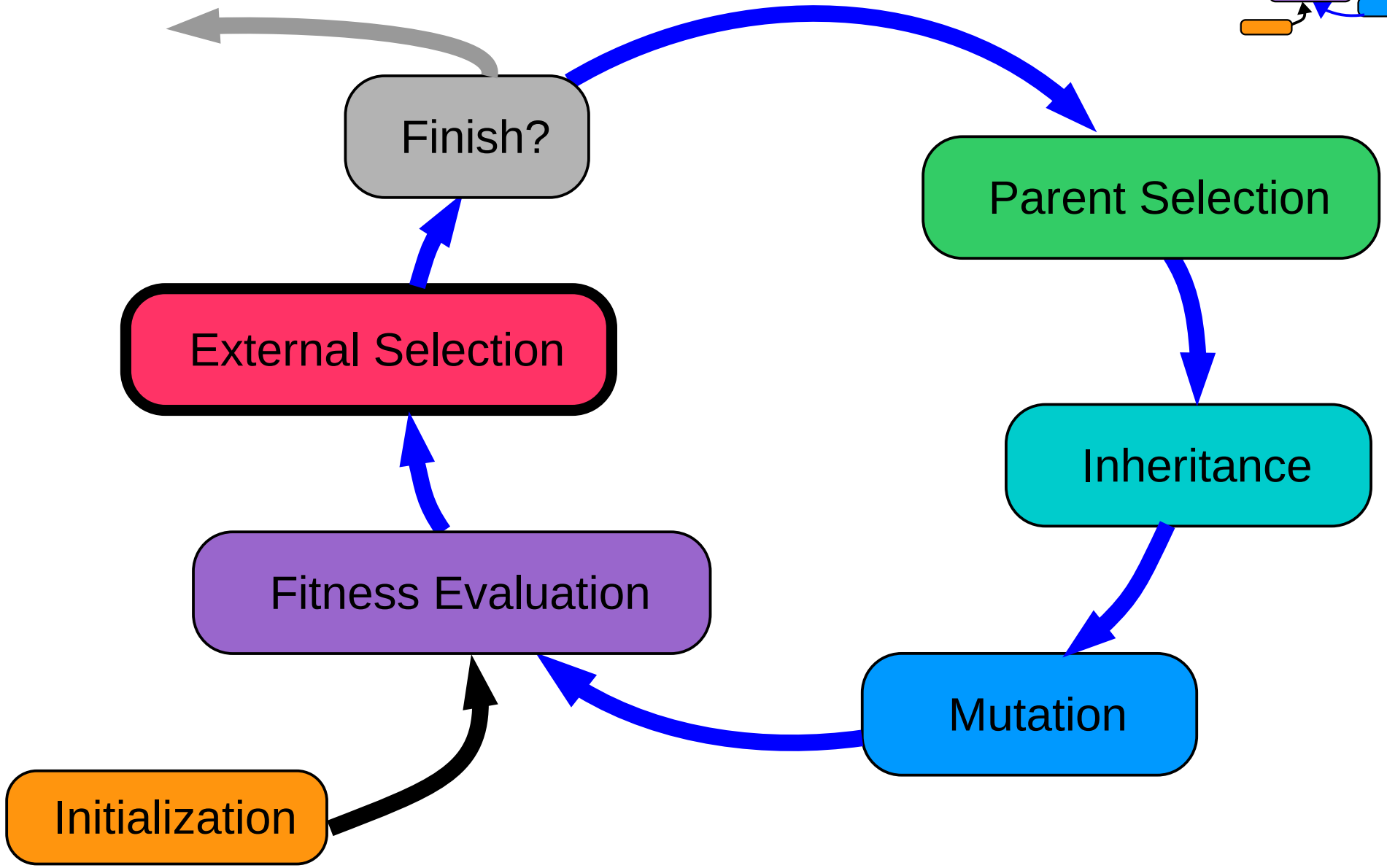


The fitness-values $f(g_p)$ for the genomes g_p sorted by $f(g_p)$ **after some steps,** and their **initial** state.

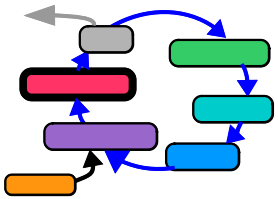
Overview

- EA Steps
  - Individual, Genome, Fitness, Population
  - Parent selection
  - Inheritance
  - Mutation
  - Fitness evaluation
  - **External selection**
  - Finish?
  - Initialization
- Strategy
- Performance Graph
- Genome Structure
- Examples

EA cycle of operations:



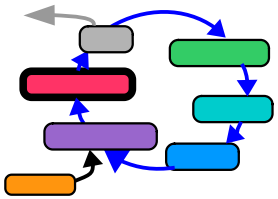
EA: External Selection



The purpose of the **selection process** is to keep the „Best“, and discard the „Losers“.
The **selection** is based on the achieved fitness, which is obtained by the fitness function, quasi from **external** to the EA.

The μ individuals that survive the **external selection** process, will be the pool for the parents of the next generation. Later on, from these μ parents the λ offspring will be generated through the inheritance process.

EA: External Selection



The two questions for the **external selection** are:

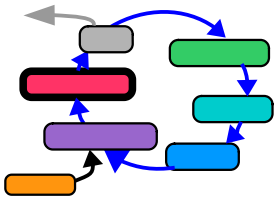
How many (μ) of the individuals shall survive?

and

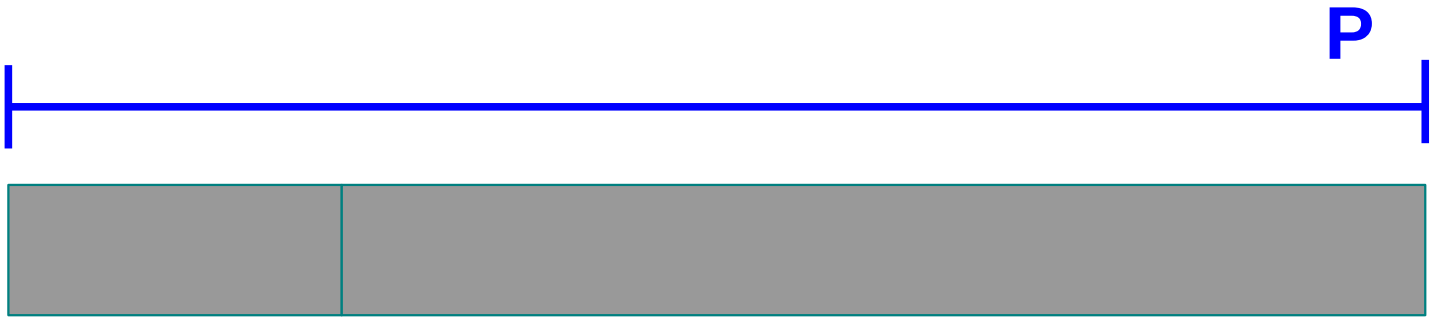
Which are the ones to keep?

A common way is to keep the population constant to **P** individuals.
discard λ by external selection
and keep μ as pool for the parents,
then generate λ offspring for the next generation.

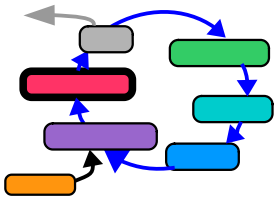
EA: External Selection



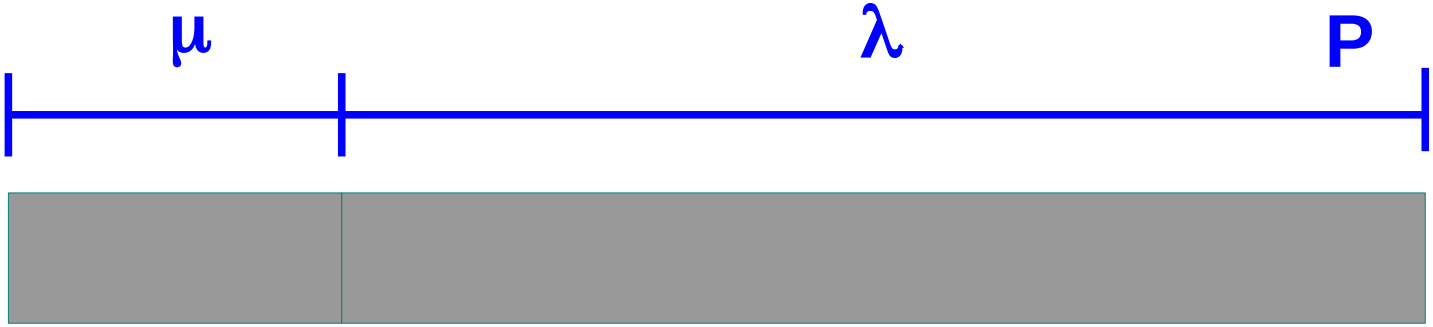
Have **P**, keep μ, generate λ offspring



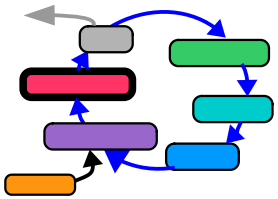
EA: External Selection



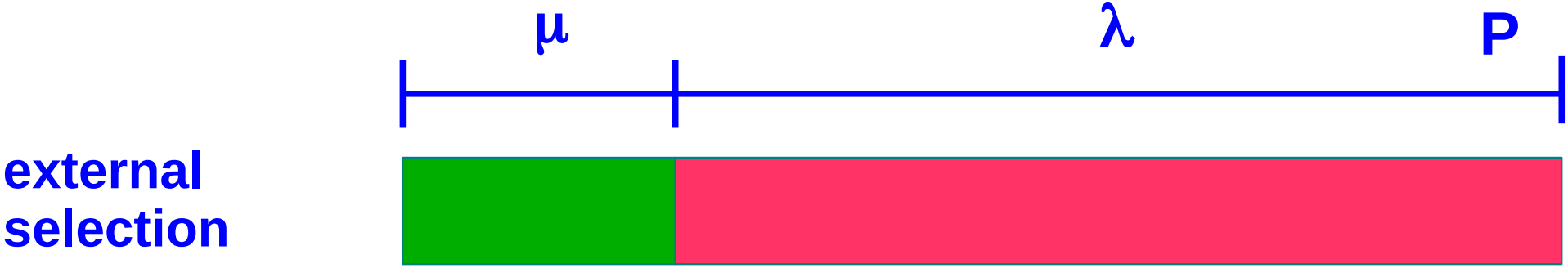
Have **P**, keep **μ**, generate **λ** offspring



EA: External Selection

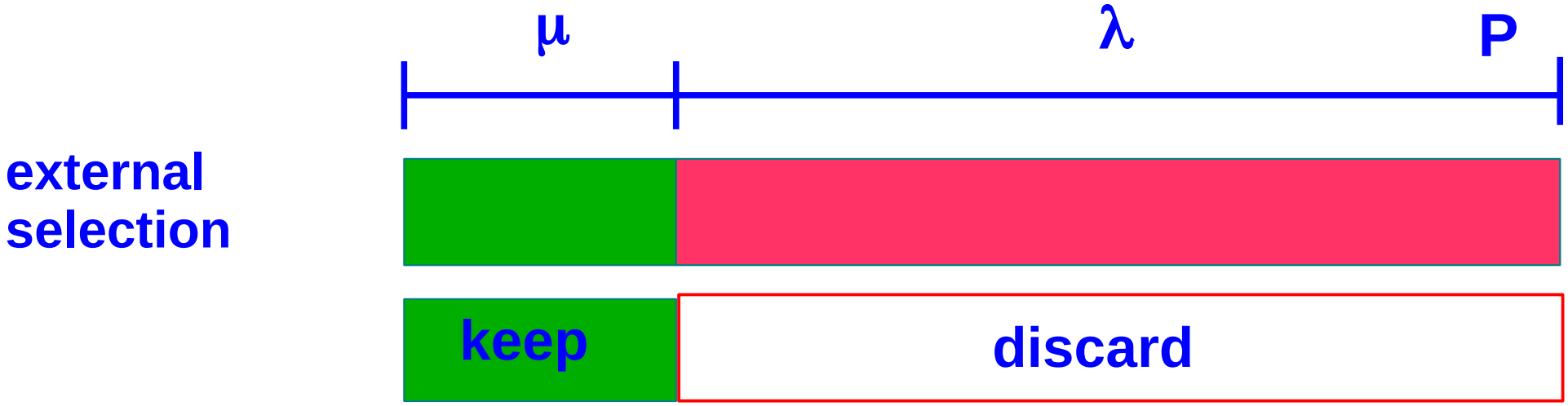


Have **P**, keep **μ**, generate **λ** offspring



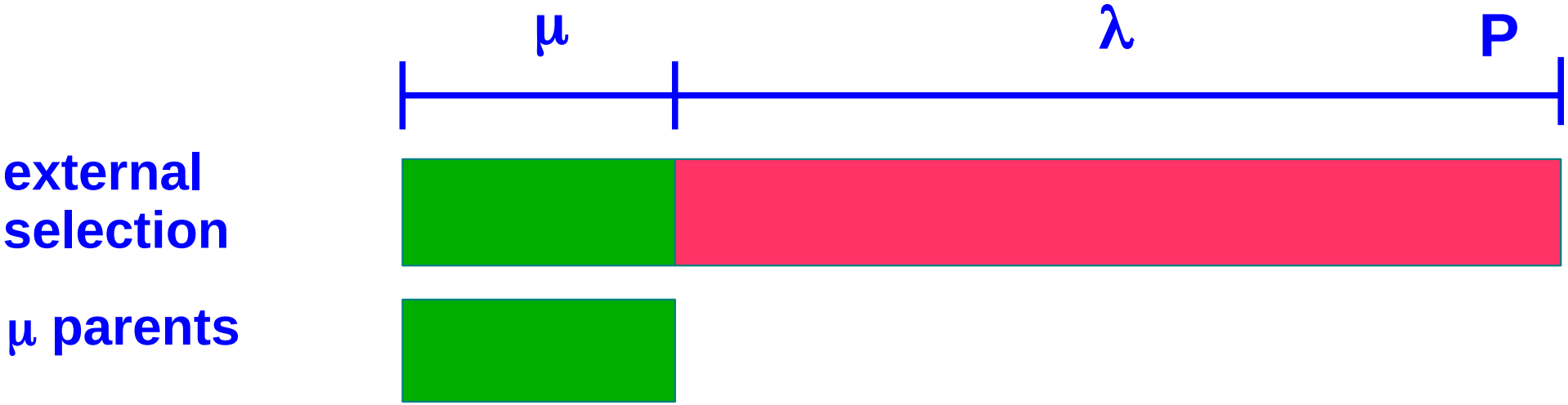
EA: External Selection

Have **P**, keep **μ**, generate **λ** offspring

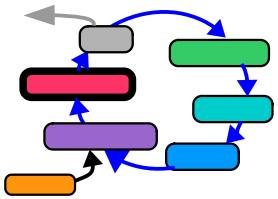


EA: External Selection

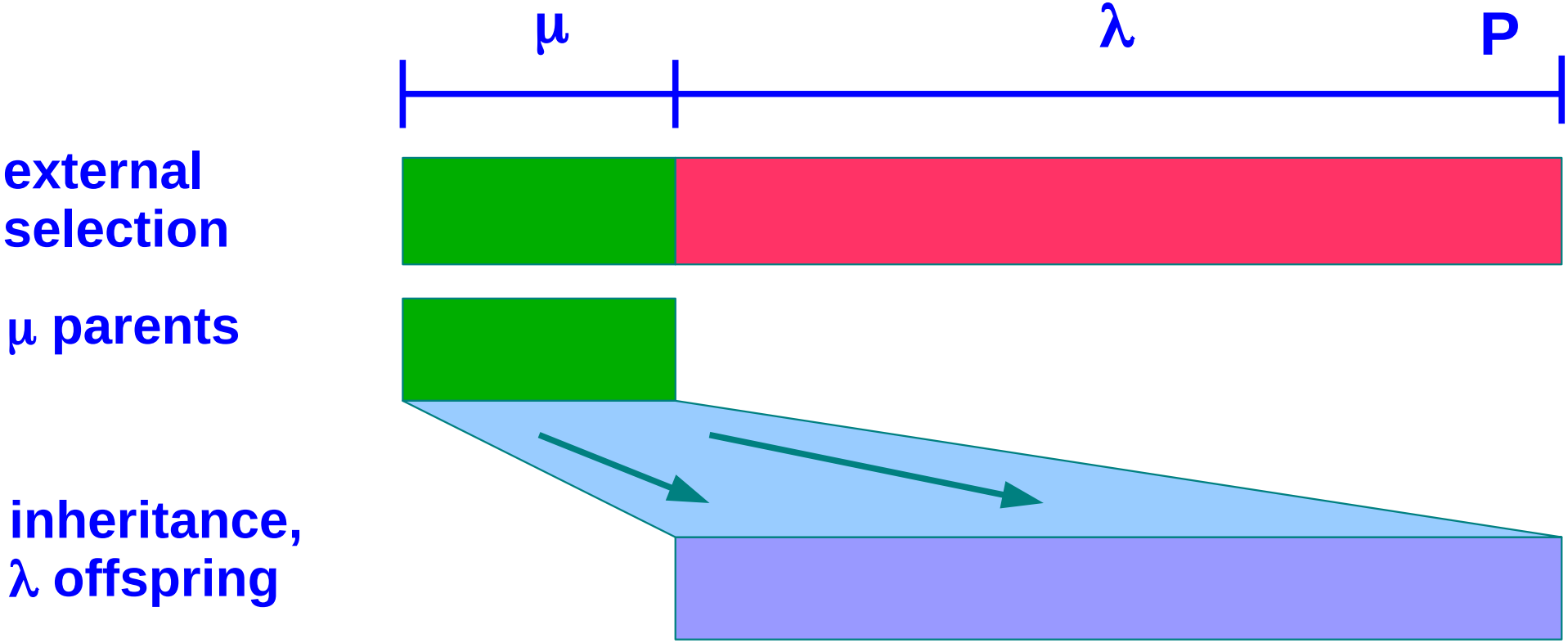
Have **P**, keep **μ**, generate **λ** offspring



EA: External Selection

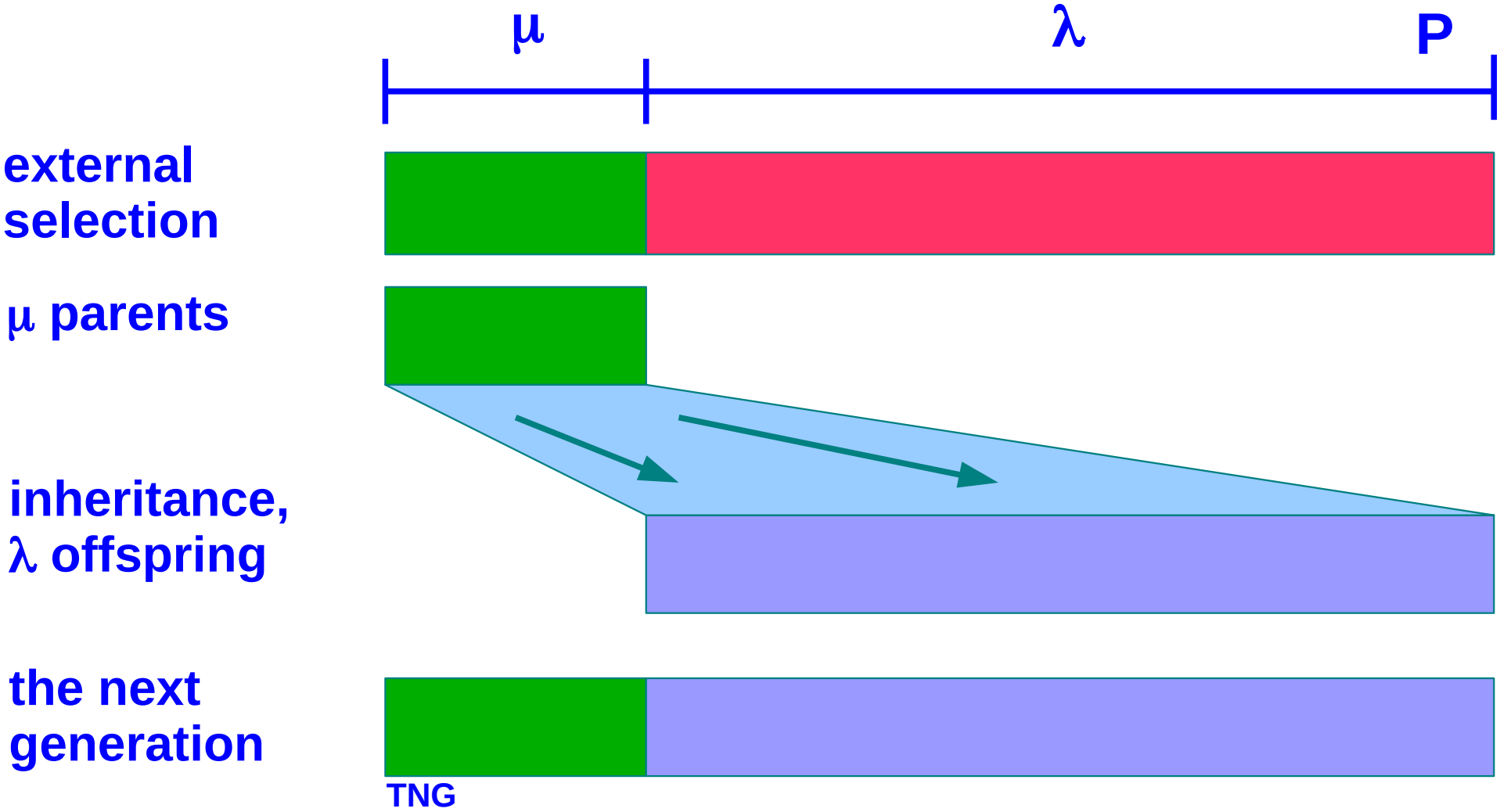


Have **P**, keep **μ**, generate **λ** offspring

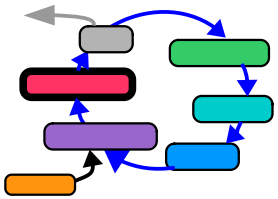


EA: External Selection

Have **P**, keep **μ**, generate **λ** offspring



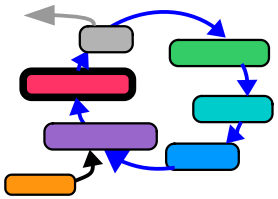
EA: External Selection



Common strategies for the **external selection** are:

- random choice
- schedule based, e.g. round robin
- fitness based elitism:
  - fitness proportional choice
  - rank proportional choice
- fitness based stochastic:
  - fitness proportionate, probabilistic choice
  - rank proportionate, probabilistic choice
- combinations of the above

EA: External Selection



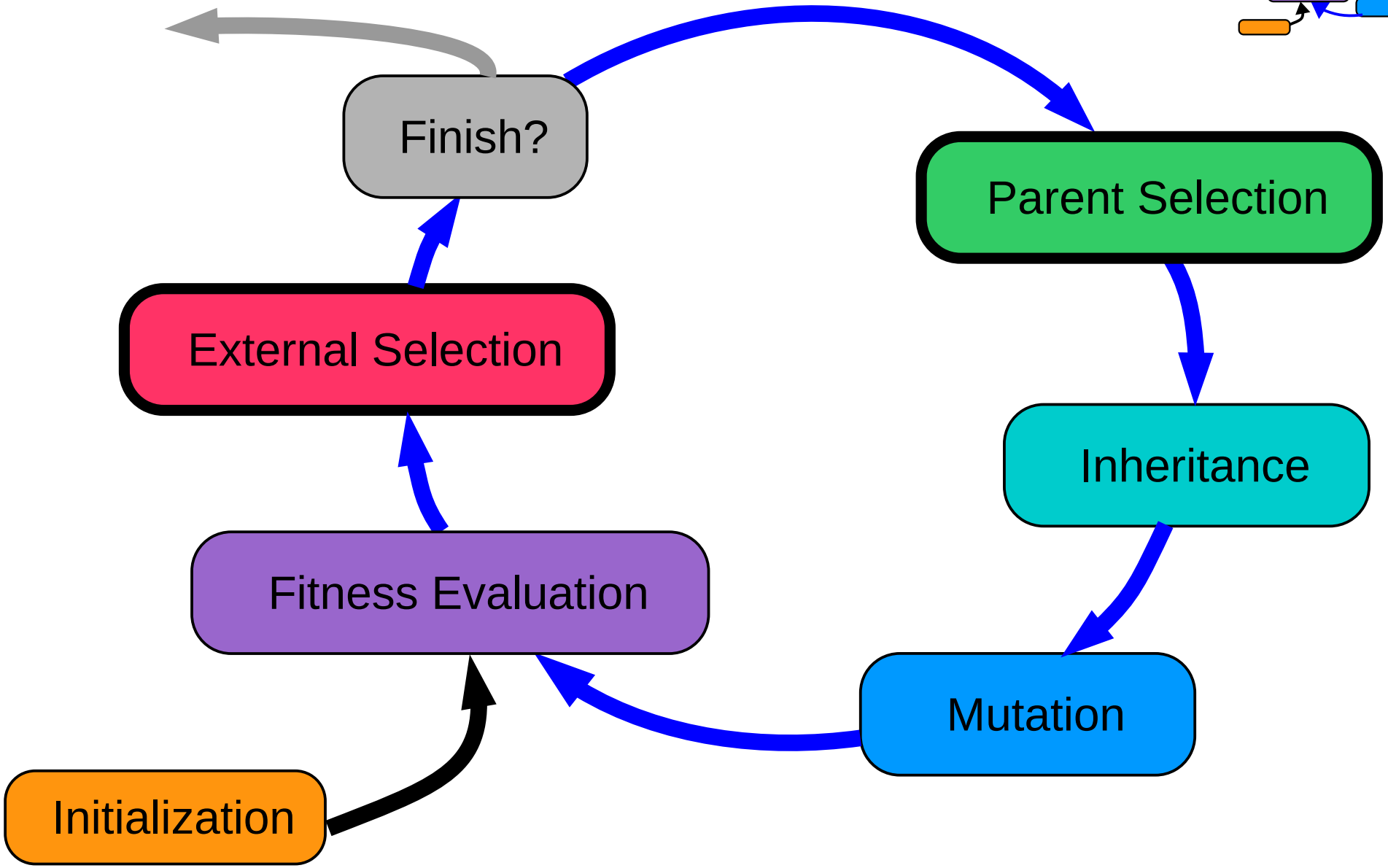
There are different ideas and principles behind the selection of the μ possible parents from the population, with special pros and cons.

- **deterministic** or **stochastic** selection
- **fitness dependent** selection
- **fitness proportional** selection
- **rank-based** selection
- **tournament** selection
- **life-time based** selection
- **combinations** of the above + more

Overview

- EA Steps
  - Individual, Genome, Fitness, Population
  - **Parent selection**
  - Inheritance
  - Mutation
  - Fitness evaluation
  - **External selection**
  - Finish?
  - Initialization
- **Strategy**
- Performance Graph
- Genome Structure
- Examples

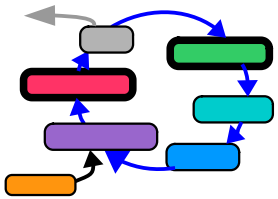
EA cycle of operations:



EA:

External Selection

Parent Selection



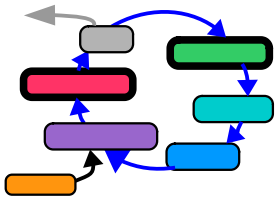
Selection in **E**volutionary **S**trategy **ES**-Systems:
Following the initial idea of **E**volutionary **S**trategies there are two strategies for the selection process:

$$(\mu + \lambda) \quad \text{and} \quad (\mu , \lambda)$$

EA:

External Selection

Parent Selection



Selection in **E**volutionary **S**trategy **ES**-Systems:
Following the initial idea of **E**volutionary **S**trategies there are two strategies for the selection process:

$$(\mu + \lambda) \quad \text{and} \quad (\mu , \lambda)$$

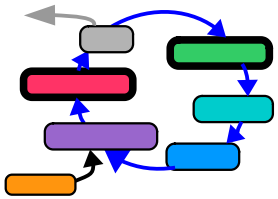
+ (plus) strategy**:** next generation consists of μ parents **+** λ offspring, parents survive.

, (comma) strategy**:** next generation consists of only the λ offspring, parents are discarded.

EA:

External Selection

Parent Selection

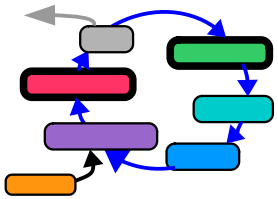


(1 + 1): $\mu=1$ one parent, $\lambda=1$ one child,
inheritance by copying, only mutation
rank based, deterministic external selection

EA:

External Selection

Parent Selection



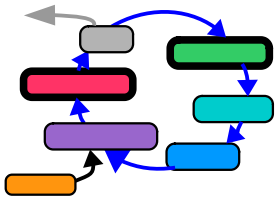
(1 + 1): $\mu=1$ one parent, $\lambda=1$ one child,
inheritance by copying, only mutation
rank based, deterministic external selection

(1 + λ): $\mu=1$ one parent, λ children, offspring,
inheritance by copying, only mutation
rank based, deterministic external selection

EA:

External Selection

Parent Selection



(1 + 1): $\mu=1$ one parent, $\lambda=1$ one child,
inheritance by copying, only mutation
rank based, deterministic external selection

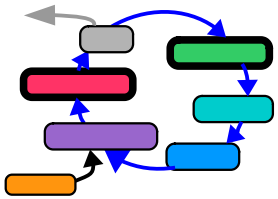
(1 + λ): $\mu=1$ one parent, λ children, offspring,
inheritance by copying, only mutation
rank based, deterministic external selection

(μ + λ): μ parents, λ offspring, parents survive
recombination, mutation, external selection.

EA:

External Selection

Parent Selection



(1 + 1): $\mu=1$ one parent, $\lambda=1$ one child,
inheritance by copying, only mutation
rank based, deterministic external selection

(1 + λ): $\mu=1$ one parent, λ children, offspring,
inheritance by copying, only mutation
rank based, deterministic external selection

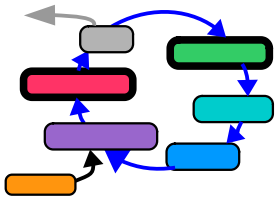
(μ + λ): μ parents, λ offspring, parents survive
recombination, mutation, external selection.

(μ , λ): μ parents, λ offspring,
recombination, mutation, external selection,
parents are discarded.

EA:

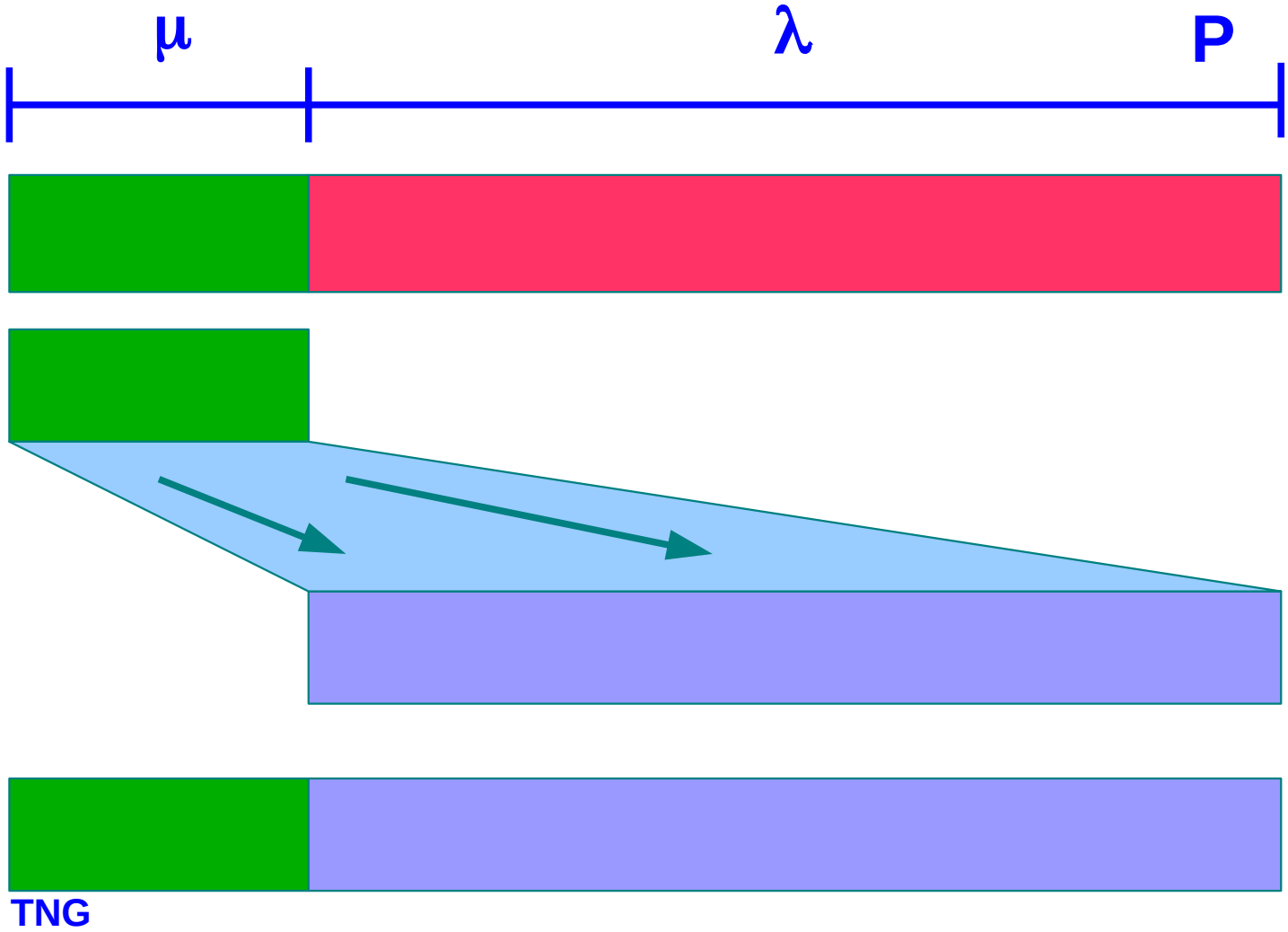
External Selection

Parent Selection



Have **P**, keep **μ**, generate **λ** offspring

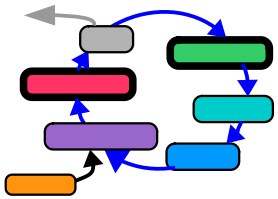
(μ + λ)



EA:

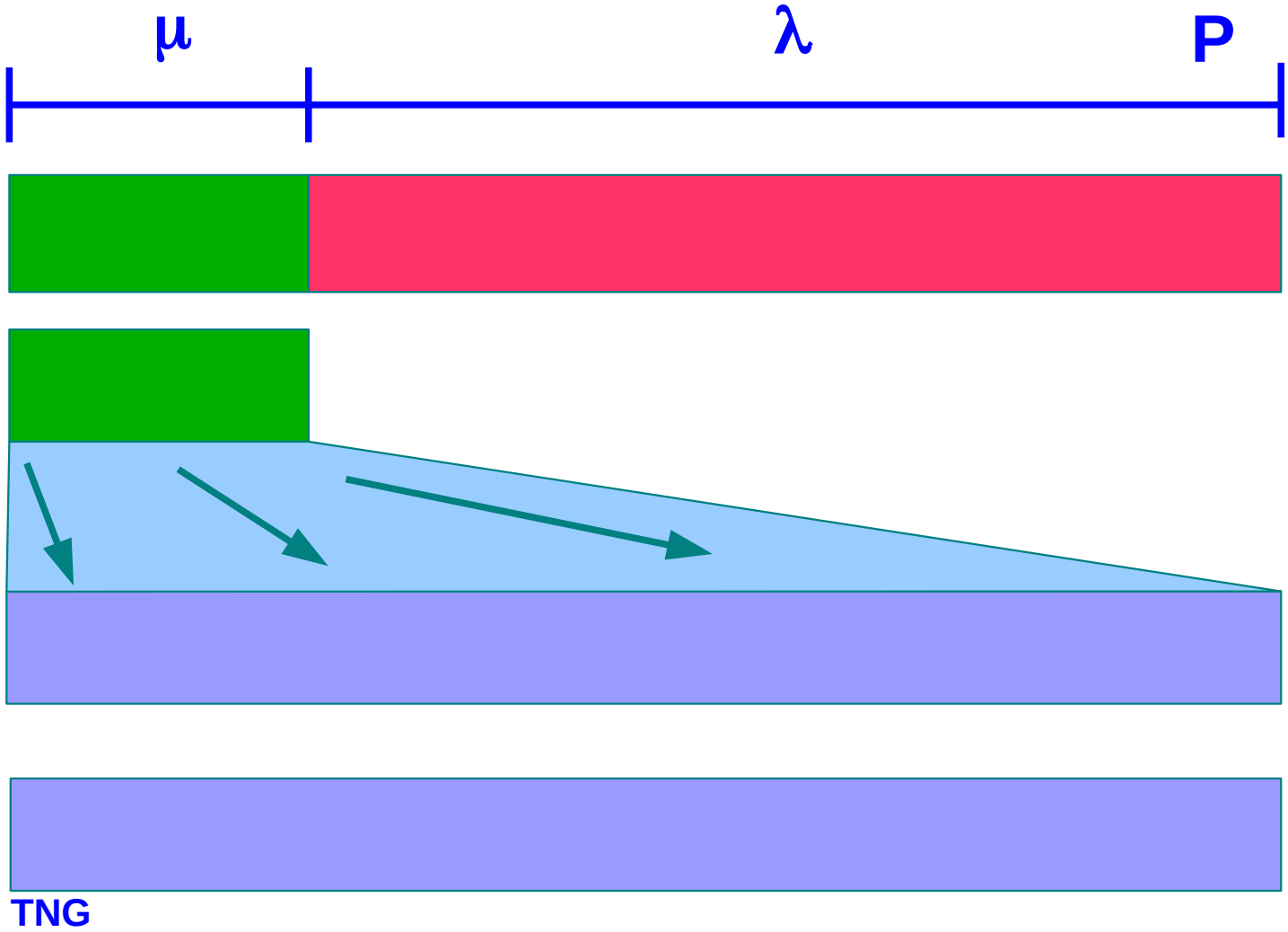
External Selection

Parent Selection



Have **P**, keep **μ**, generate **λ** offspring

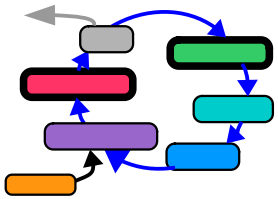
(μ , λ)



EA:

External Selection

Parent Selection



Selection in **E**volutionary **S**trategy **ES**-Systems:
Following the initial idea of **E**volutionary **S**trategies there are two strategies for the selection process:

$$(\mu + \lambda) \quad \text{and} \quad (\mu , \lambda)$$

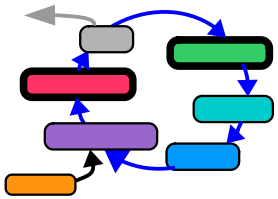
+ (plus) strategy**:** next generation consists of μ parents **+** λ offspring, parents survive.

, (comma) strategy**:** next generation consists of only the λ offspring, parents are discarded.

EA:

External Selection

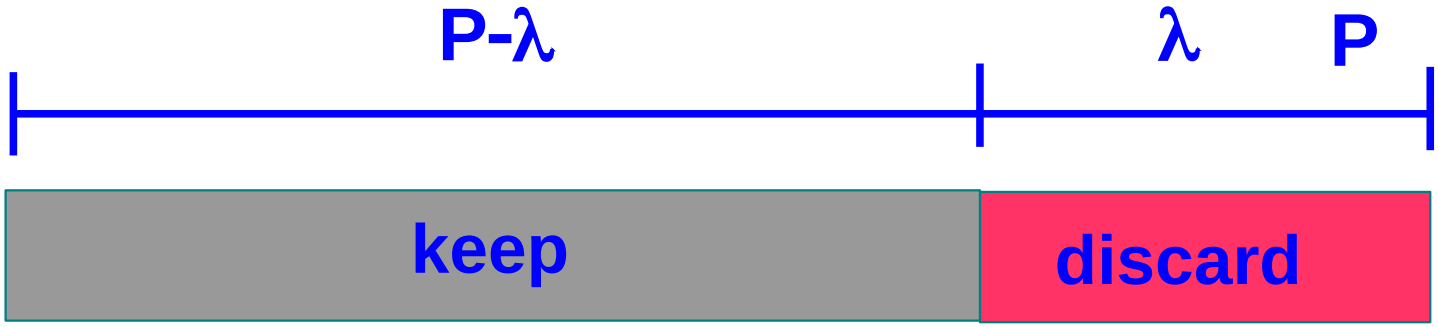
Parent Selection



Have **P**, discard **λ**, select **μ** parents, generate **λ** offspring

(μ +* λ)

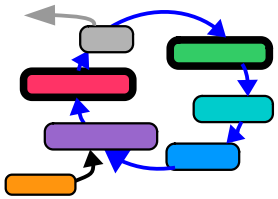
external selection



EA:

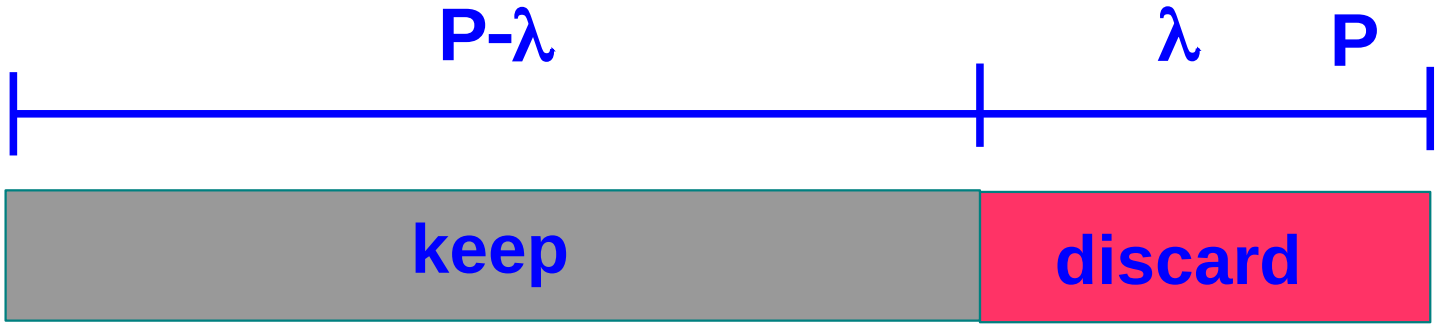
External Selection

Parent Selection



Have **P**, discard **λ**, select **μ** parents, generate **λ** offspring

(μ +* λ)



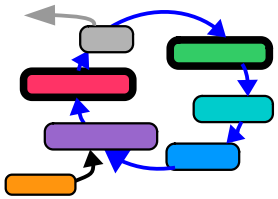
external selection

select μ parents

EA:

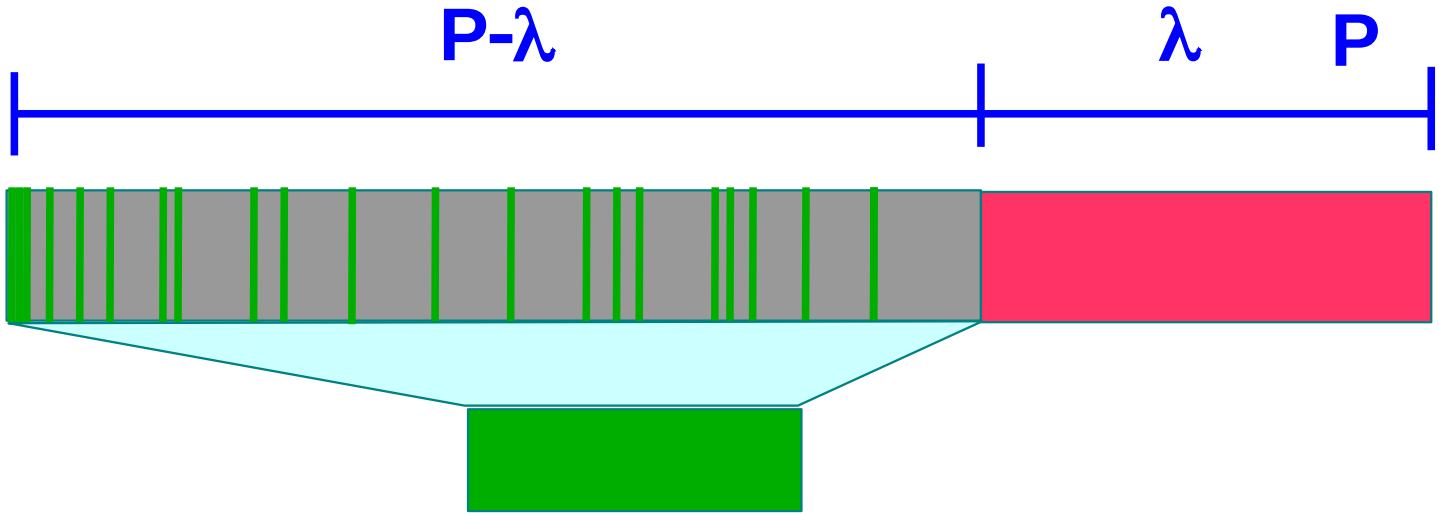
External Selection

Parent Selection



Have **P**, discard **λ**, select **μ** parents, generate **λ** offspring

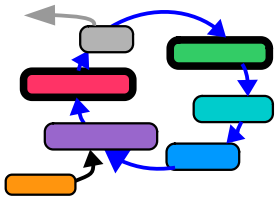
(μ +* λ)



EA:

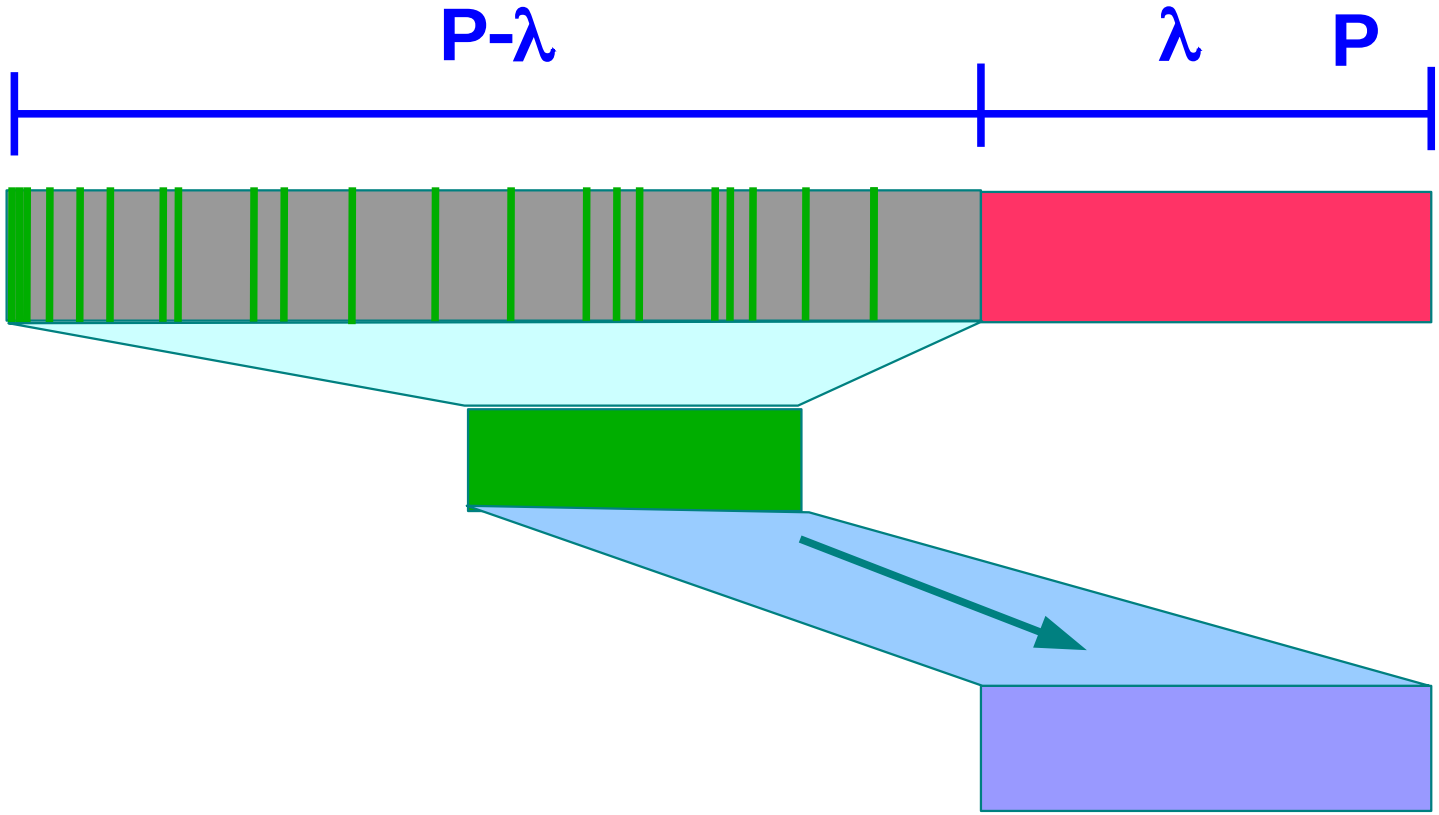
External Selection

Parent Selection



Have **P**, discard **λ**, select **μ** parents, generate **λ** offspring

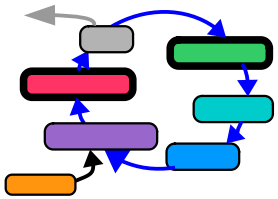
(μ +* λ)



EA:

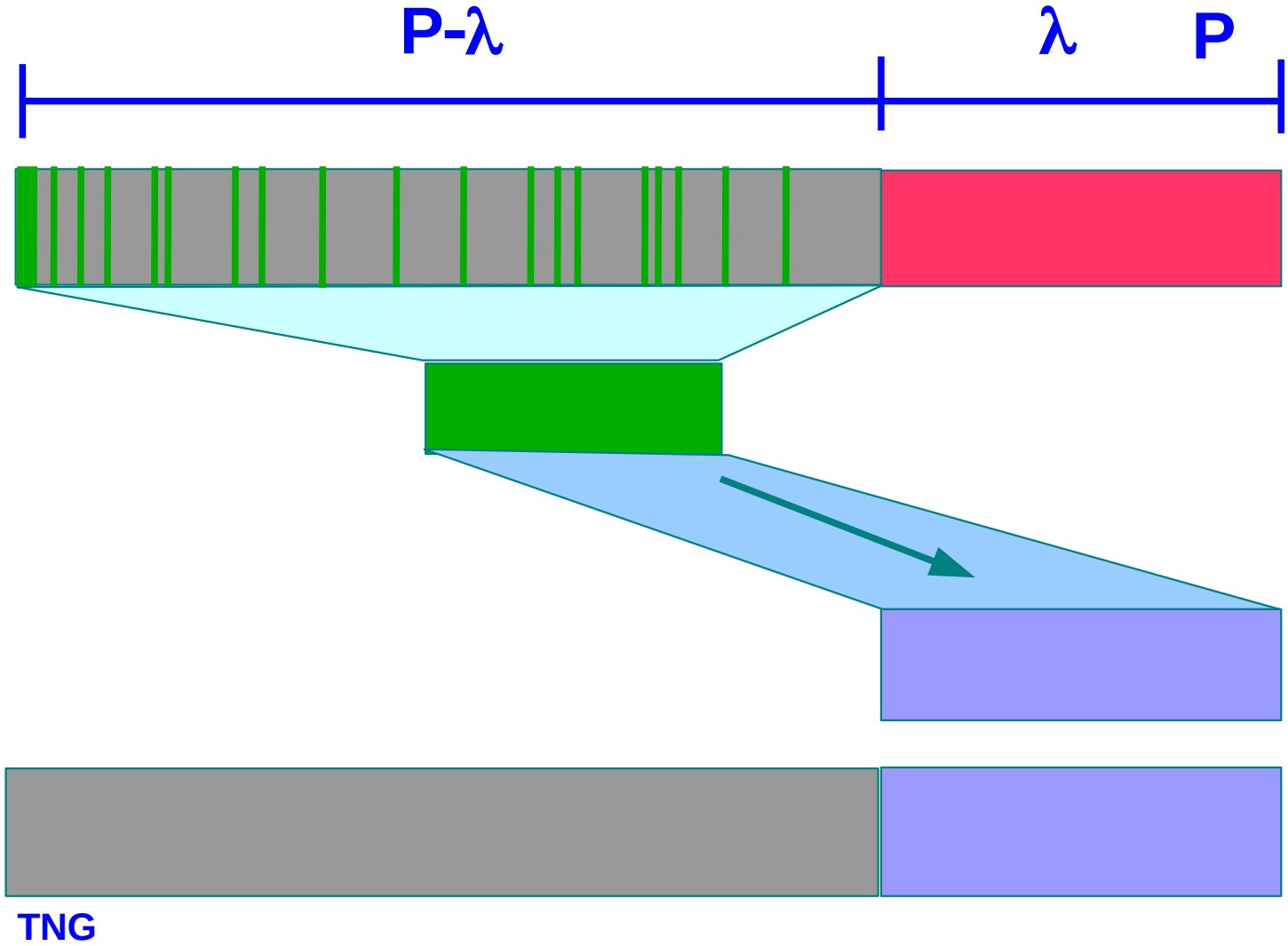
External Selection

Parent Selection



Have **P**, discard **λ**, select **μ** parents, generate **λ** offspring

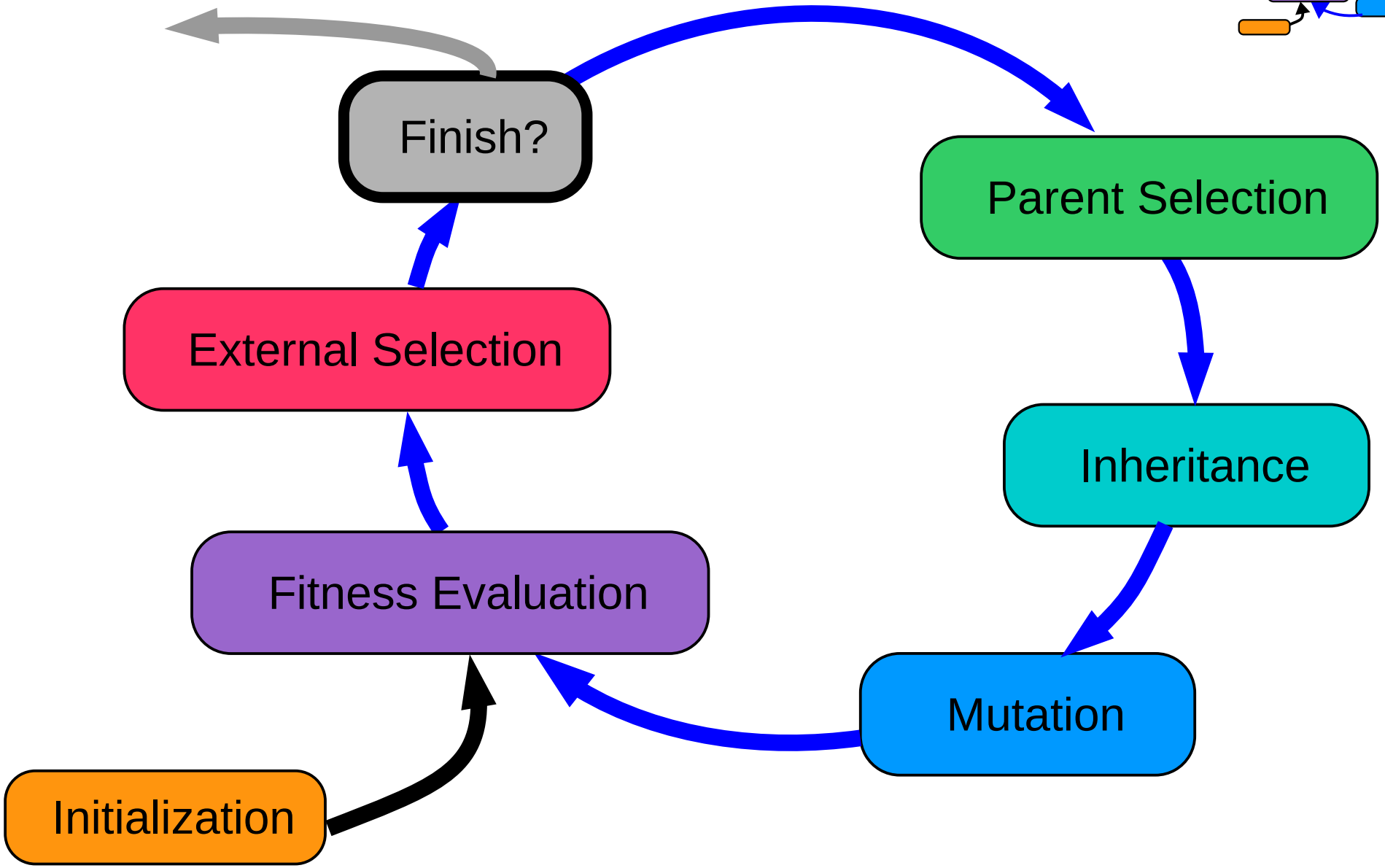
(μ +* λ)



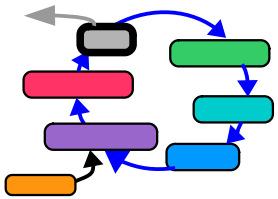
Overview

- EA Steps
  - Individual, Genome, Fitness, Population
  - Parent selection
  - Inheritance
  - Mutation
  - Fitness evaluation
  - External selection
  - **Finish?**
  - Initialization
- Strategy
- Performance Graph
- Genome Structure
- Examples

EA cycle of operations:

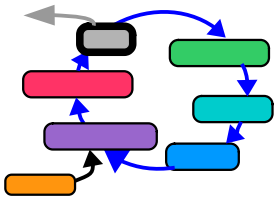


EA: Finish?



There are several criteria to determine the **finishing** of the **EA** process:

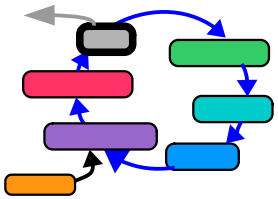
EA: Finish?



There are several criteria to determine the **finishing** of the **EA** process:

- **by performance** of best individual
- **by performance** of sub-population
- **by stagnation**/**development** of fitness improvement
- **by time**
- **by number** of generations
- ...

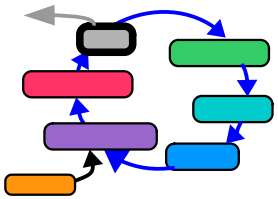
EA: Finish?



There are several criteria to determine the **finishing** of the **EA** process:

- **by performance** of best individual
- **by performance** of sub-population
- **by stagnation/development** of fitness improvement
- **by time**
- **by number** of generations
- ...
- **by choice** of human operator

EA: Finish?



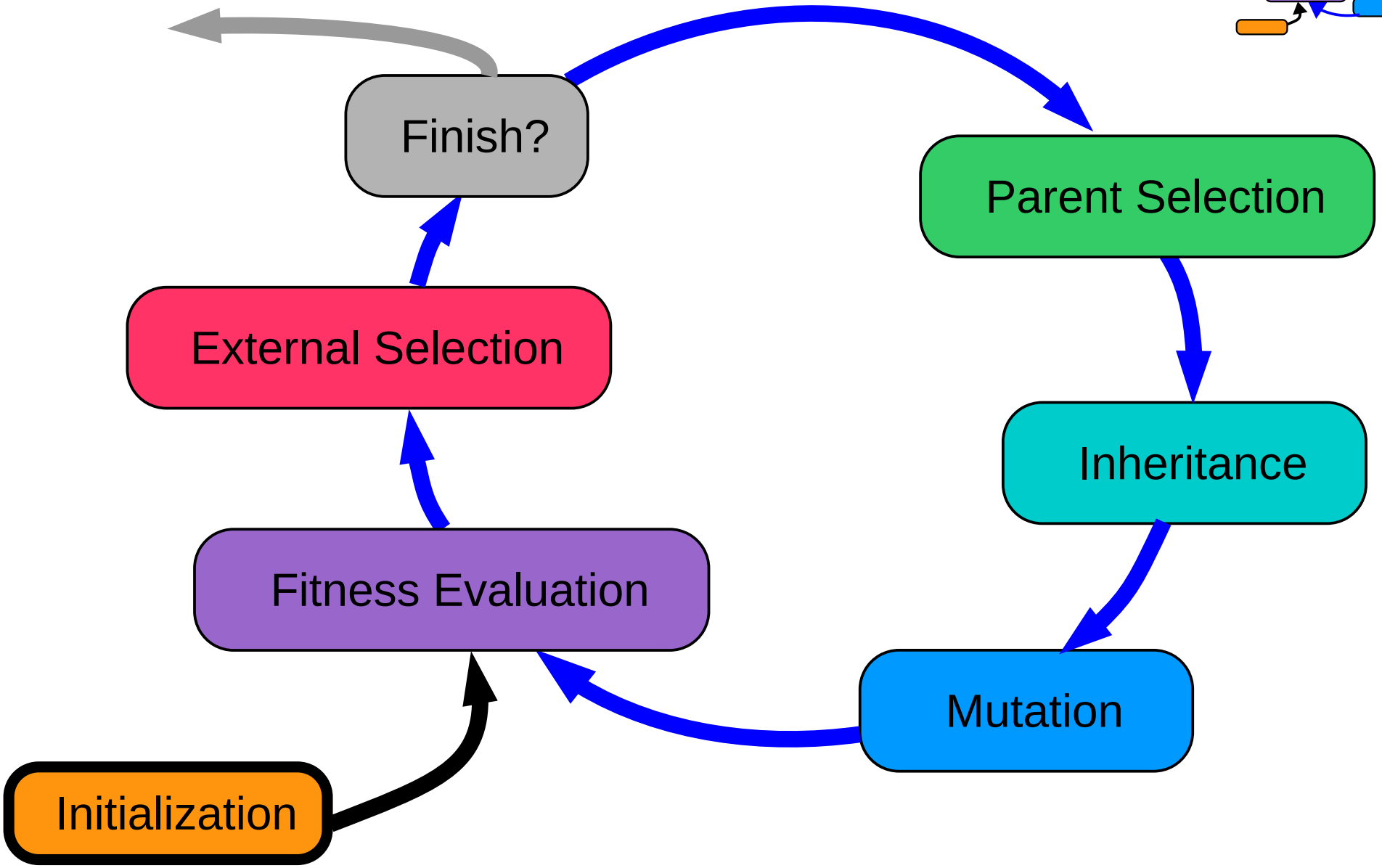
There are several criteria to determine the **finishing** of the **EA** process:

- **by performance** of best individual
- **by performance** of sub-population
- **by stagnation/development** of fitness improvement
- **by time**
- **by number** of generations
- ...
- **by choice** of human operator; **whatever that means**.

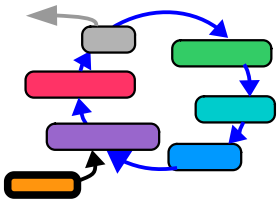
Overview

- EA Steps
  - Individual, Genome, Fitness, Population
  - Parent selection
  - Inheritance
  - Mutation
  - Fitness evaluation
  - External selection
  - Finish?
  - **Initialization**
- Strategy
- Performance Graph
- Genome Structure
- Examples

EA cycle of operations:

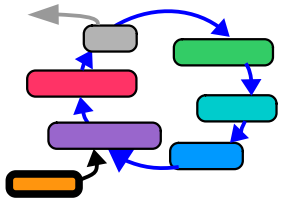


EA: Initialization



The main principles driving the **initialization** of the genomes of the first population are:

EA: Initialization

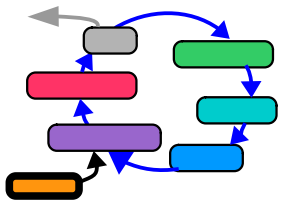


The main principles driving the **initialization** of the genomes of the first population are:

Start as good as possible:

Enough richness, enough diversity:

EA: Initialization



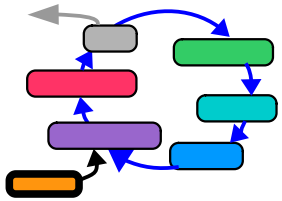
The main principles driving the **initialization** of the genomes of the first population are:

Start as good as possible:

Try to use all a priori knowledge available.
Try to avoid illegal genomes (if possible).

Enough richness, enough diversity:

EA: Initialization



The main principles driving the **initialization** of the genomes of the first population are:

Start as good as possible:

Try to use all a priori knowledge available.
Try to avoid illegal genomes (if possible).

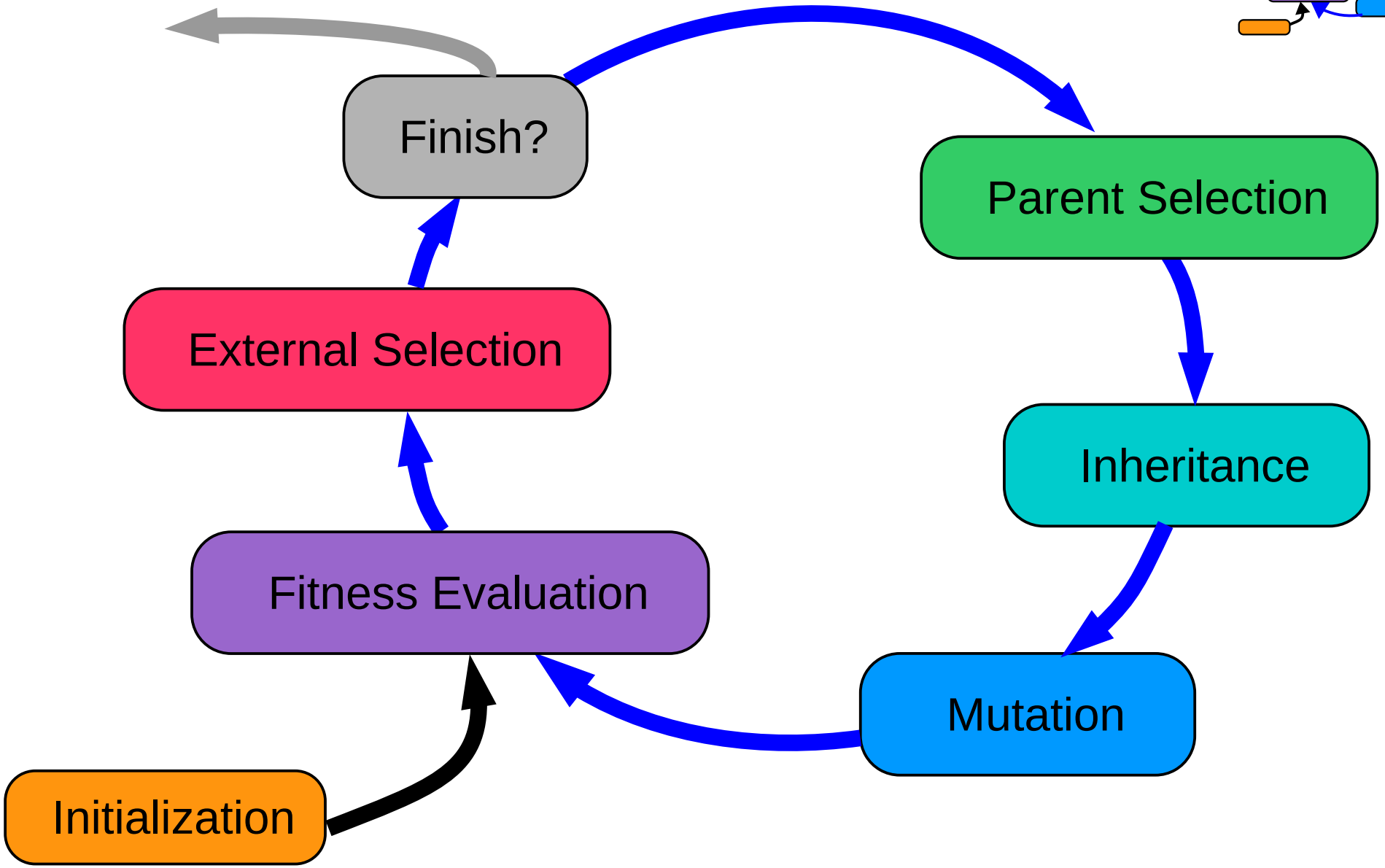
Enough richness, enough diversity:

Sample as much from the fitness landscape as possible.
Try to cover the complete search space.

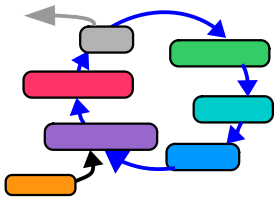
Overview

- EA Steps
  - Individual, Genome, Fitness, Population
  - Parent selection
  - Inheritance
  - Mutation
  - Fitness evaluation
  - External selection
  - Finish?
  - Initialization
- Strategy
- **Performance Graph**
- Genome Structure
- Examples

EA cycle of operations:



EA: Performance Graph



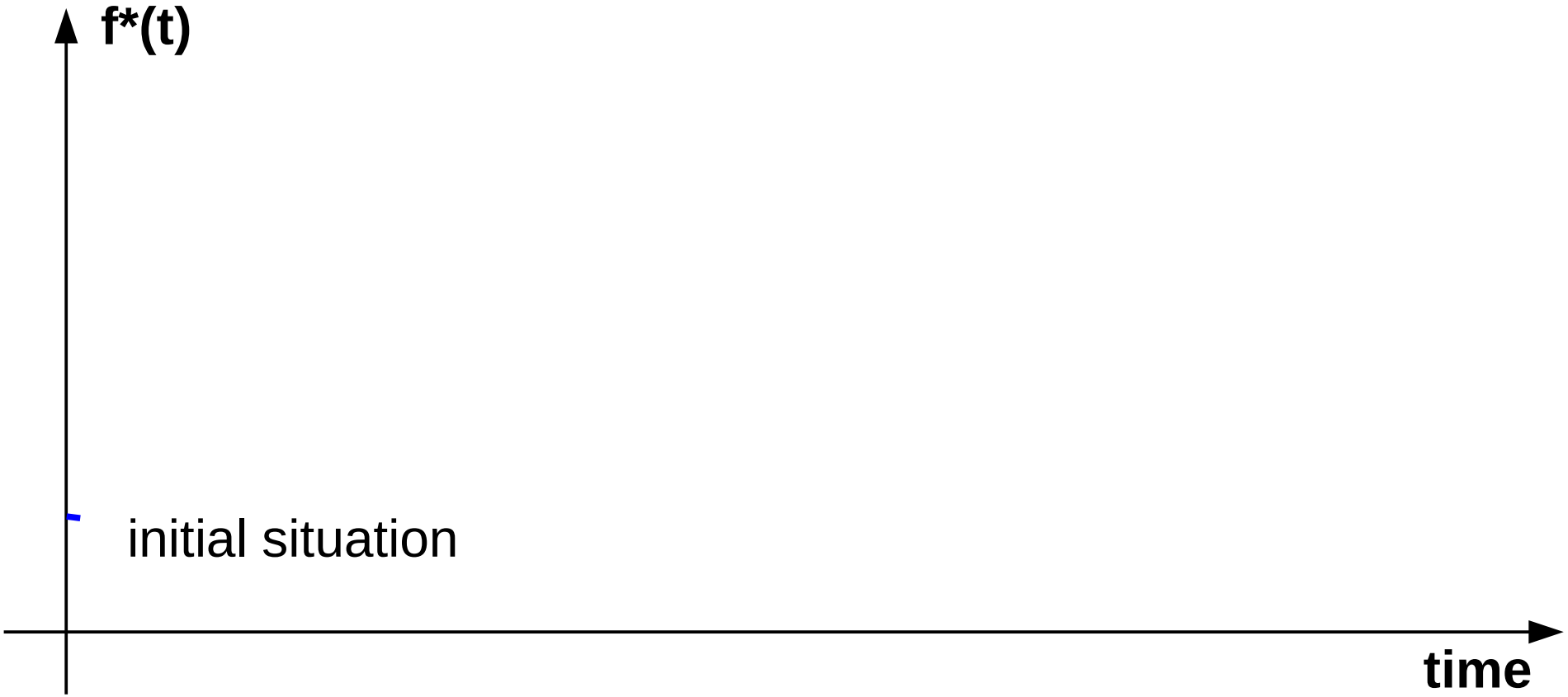
The goal of the EA is to maximize the fitness.
So, if the EA is working properly, somehow the fitness value within the complete population should rise with time.

How is the fitness of the best individual in the population changing over time?

The performance graph is showing the development of the fitness f*(t) of the best individual in each generation with respect to time.

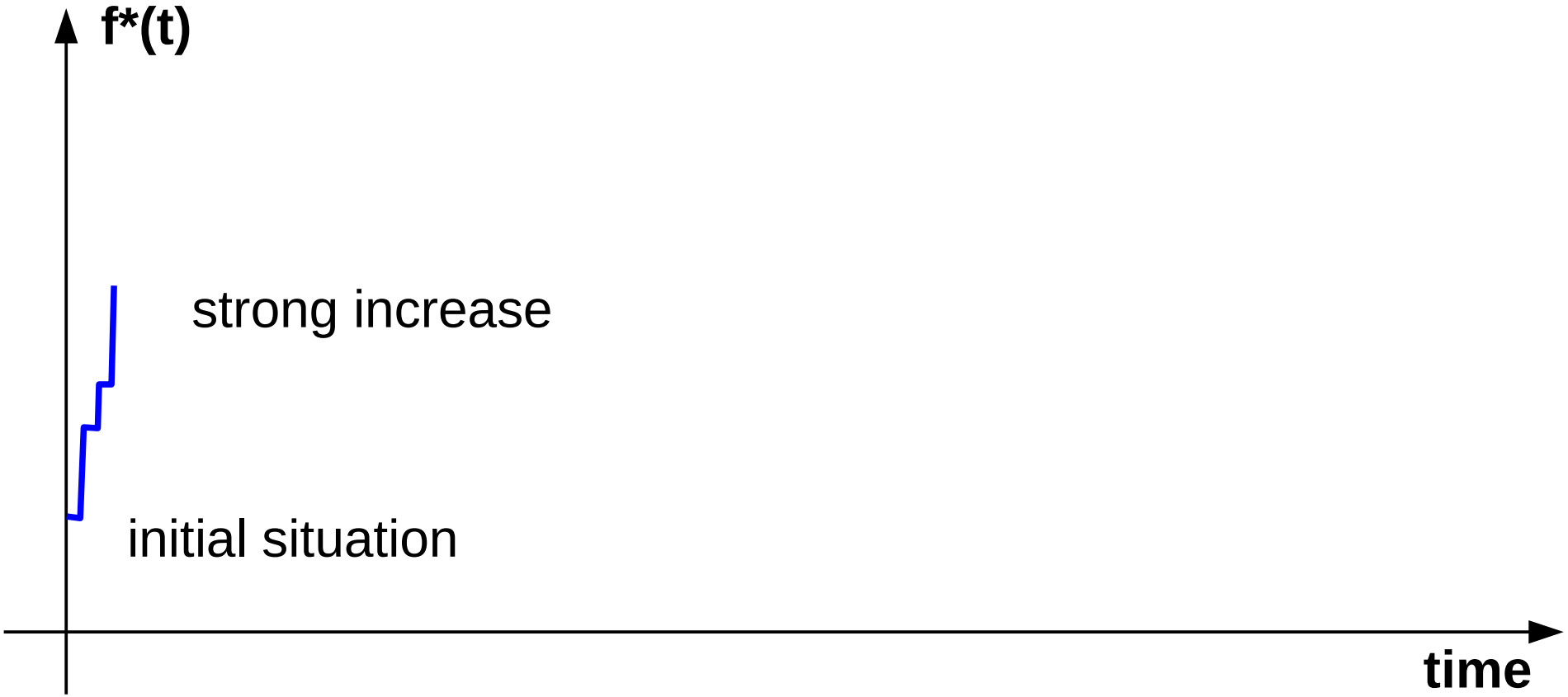
EA: Performance Graph

The performance graph is showing the development of the fitness f*(t) of the best individual in each generation with respect to time.



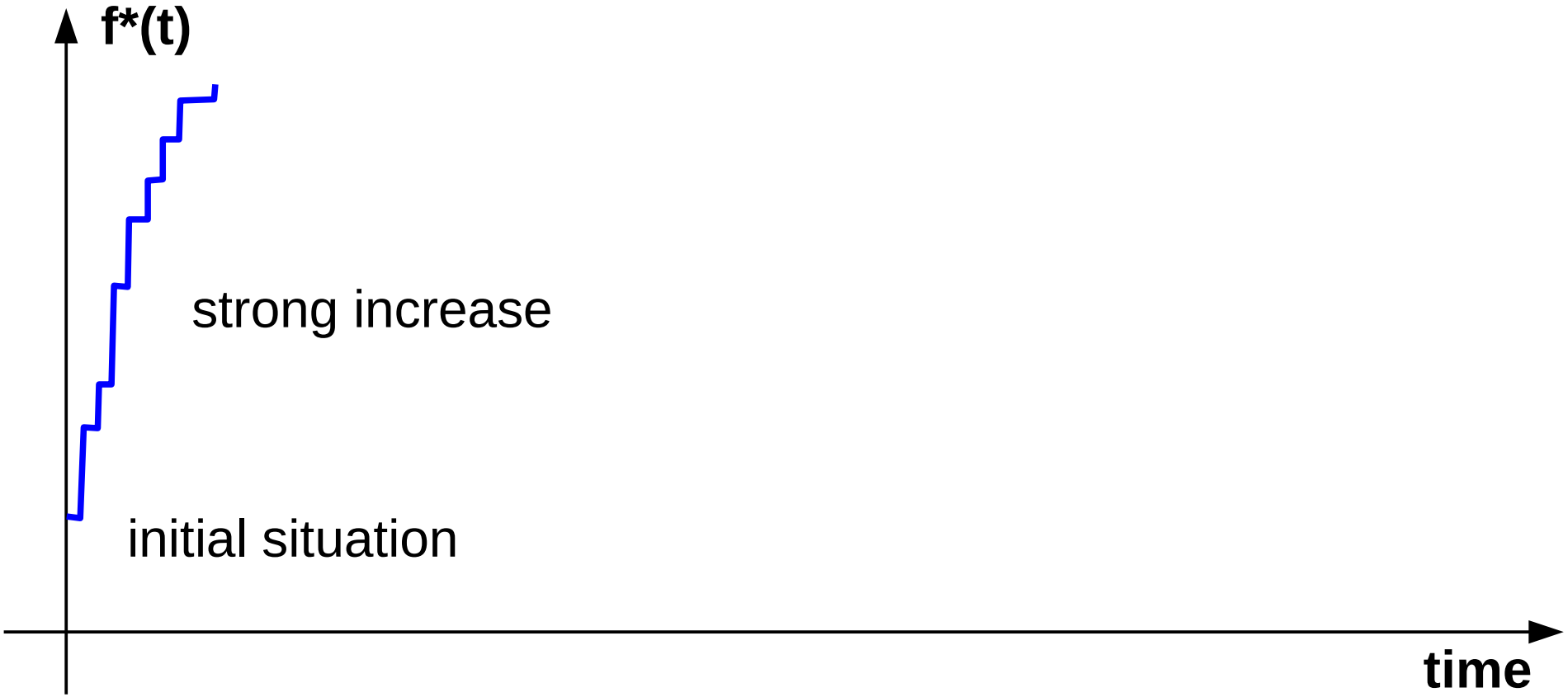
EA: Performance Graph

The performance graph is showing the development of the fitness f*(t) of the best individual in each generation with respect to time.



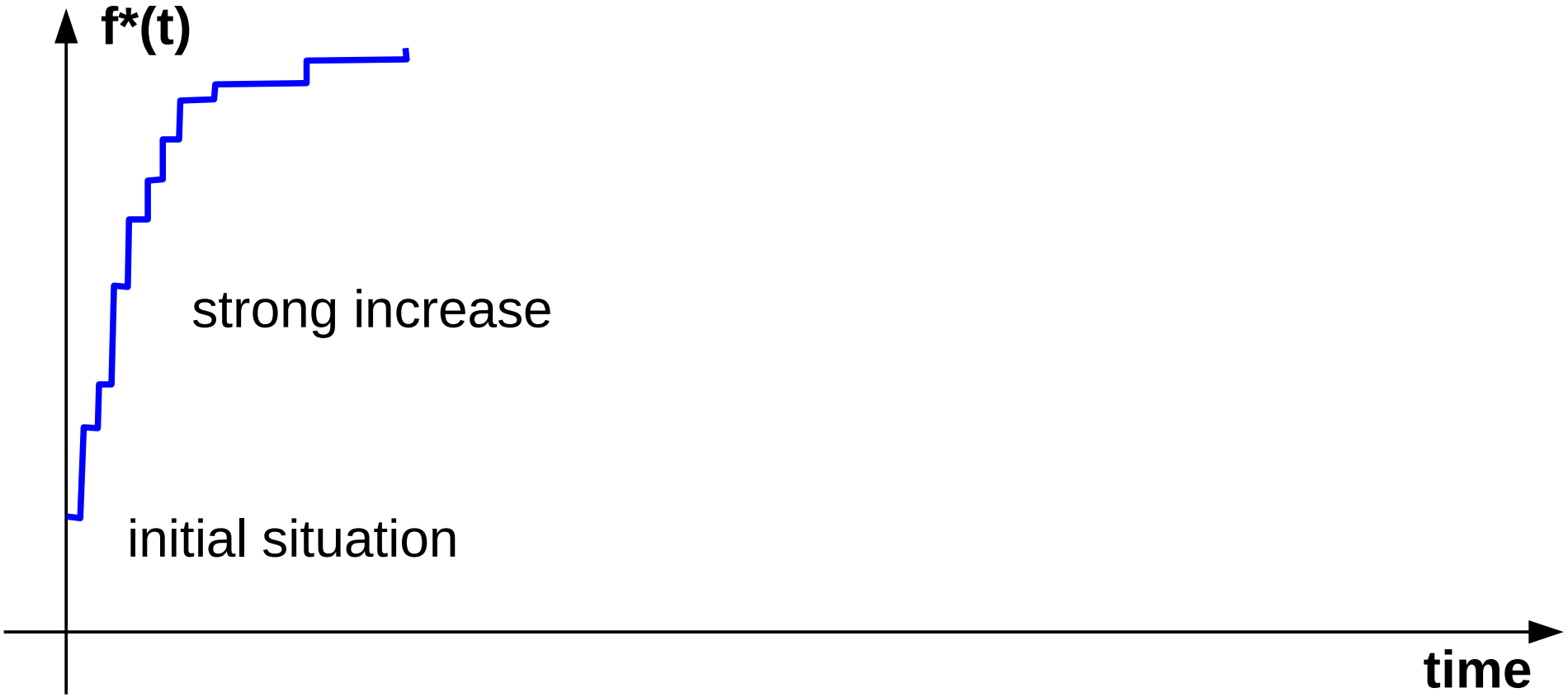
EA: Performance Graph

The performance graph is showing the development of the fitness f*(t) of the best individual in each generation with respect to time.



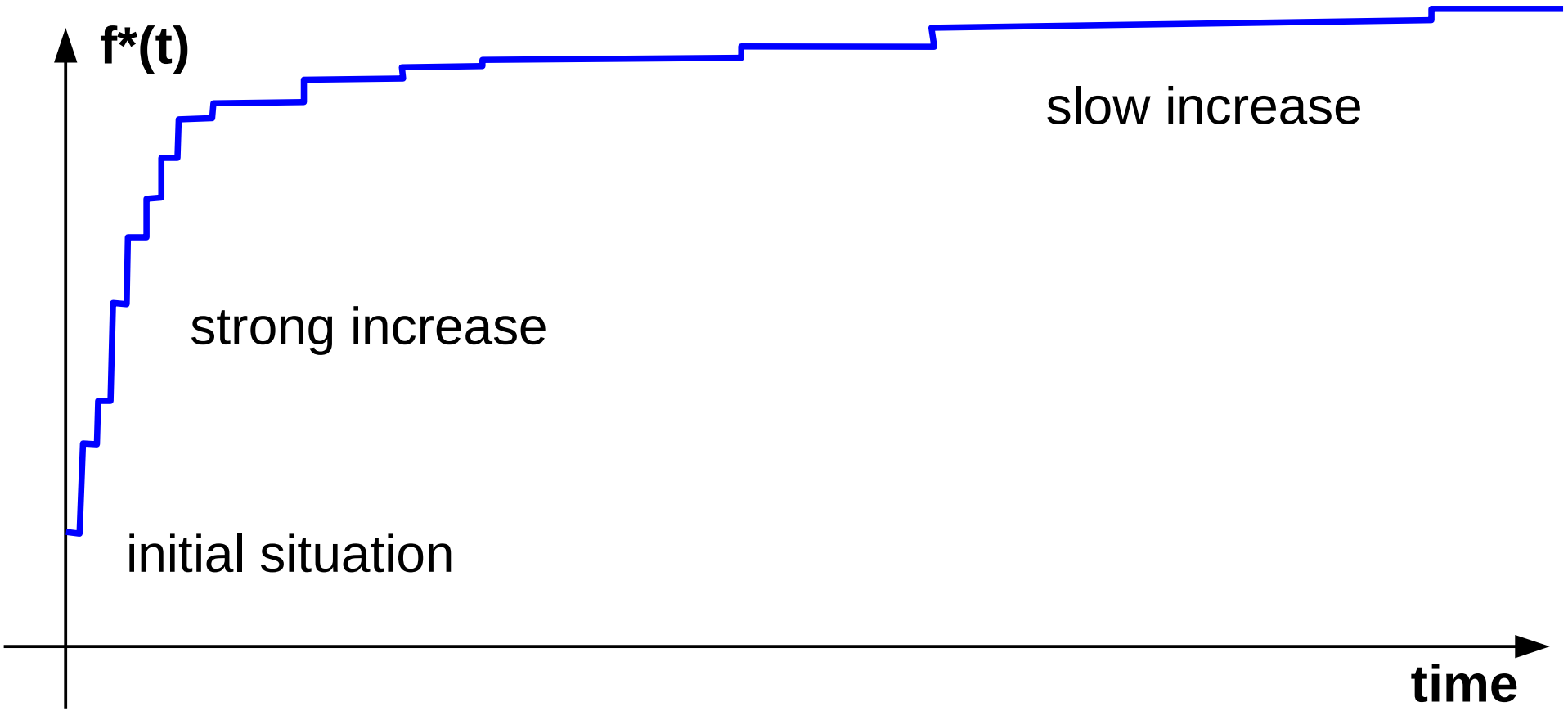
EA: Performance Graph

The performance graph is showing the development of the fitness f*(t) of the best individual in each generation with respect to time.

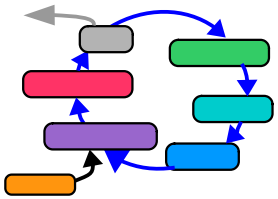


EA: Performance Graph

The performance graph is showing the development of the fitness f*(t) of the best individual in each generation with respect to time.



EA: Performance Graph



The performance graph is the most important tool to monitor the optimization process of a working evolutionary algorithm.

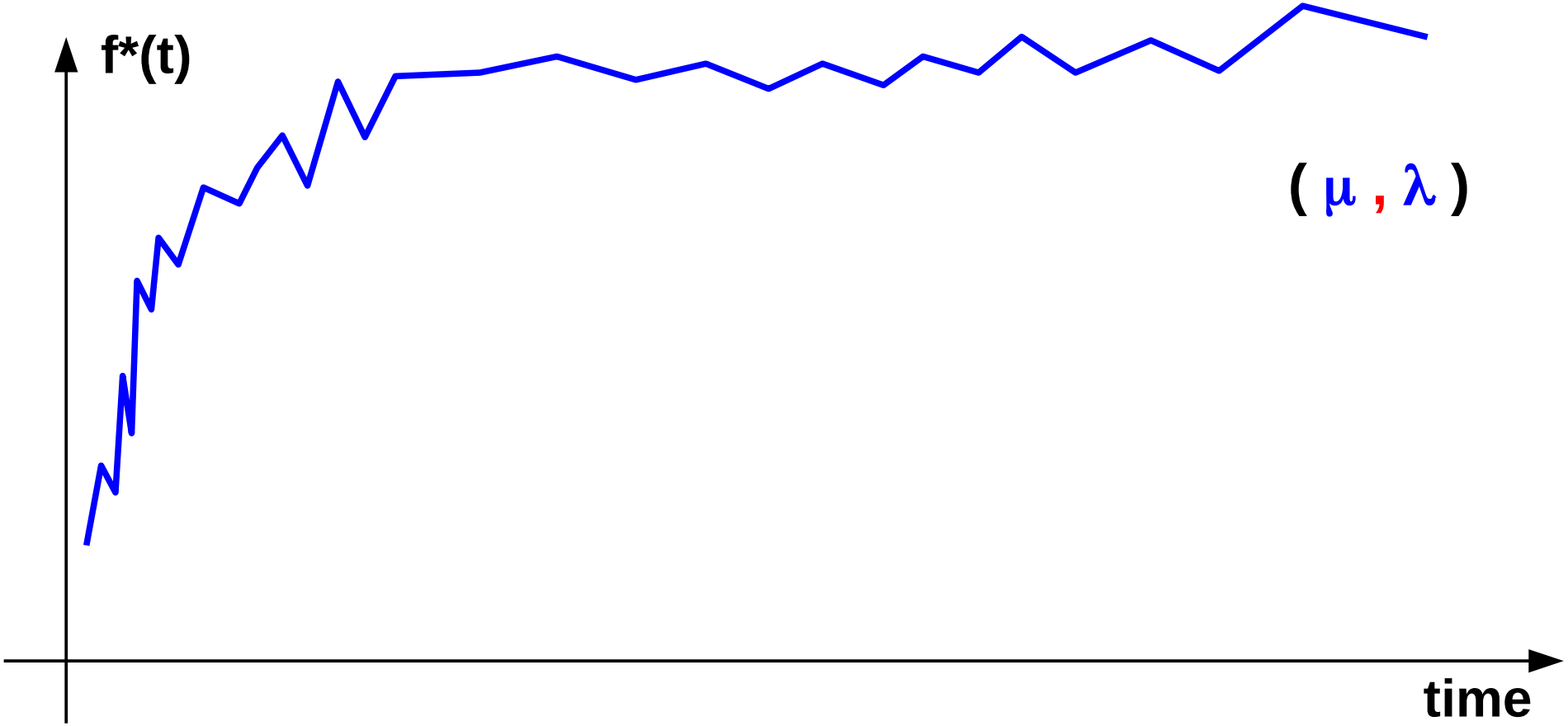
Depending on the chosen strategy, the performance graph shows different properties:

For a deterministic, rank dependent elitism strategy **(** $\mu + \lambda$ **)**, the performance graph will increase monotonically.

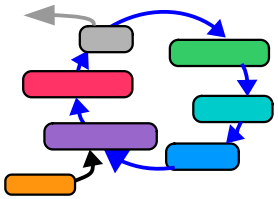
For a probabilistic, non-elitism strategy, the performance graph can decrease, but should show an increase on the long run.

EA: Performance Graph

The performance graph is showing the development of the fitness f*(t) of the best individual in each generation with respect to time.



EA: Performance Graph



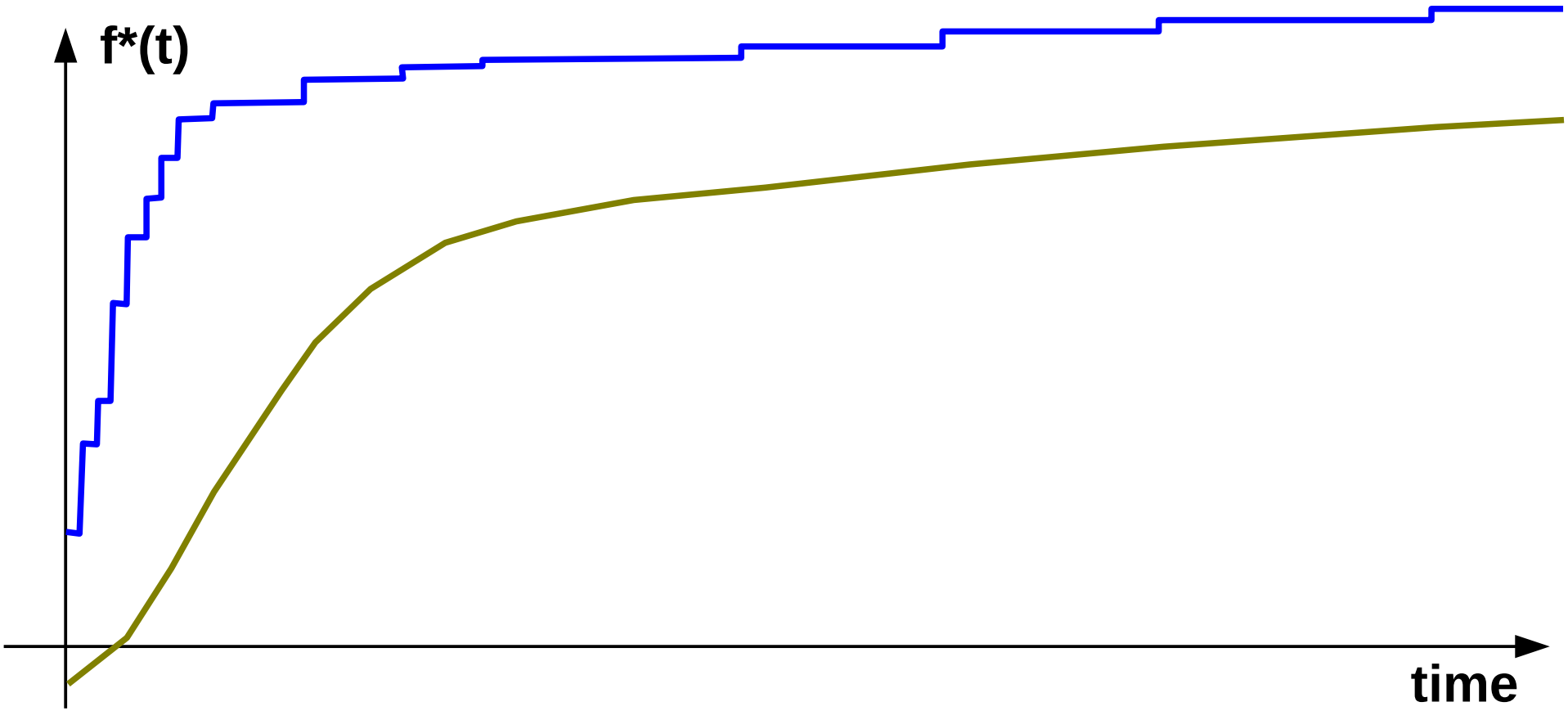
The performance graph is the most important tool to monitor the optimization process of a working evolutionary algorithm.

The performance graph can depict different aspects:

- the fitness $f^*(t)$ of the best individual, (most important)
- the average fitness $f_{av}(t)$ of the complete population
- the average and variance over the complete population
- the average fitness $f_{avp}(t)$ of the parents
- all fitness values $f_p(t)$ from all parents

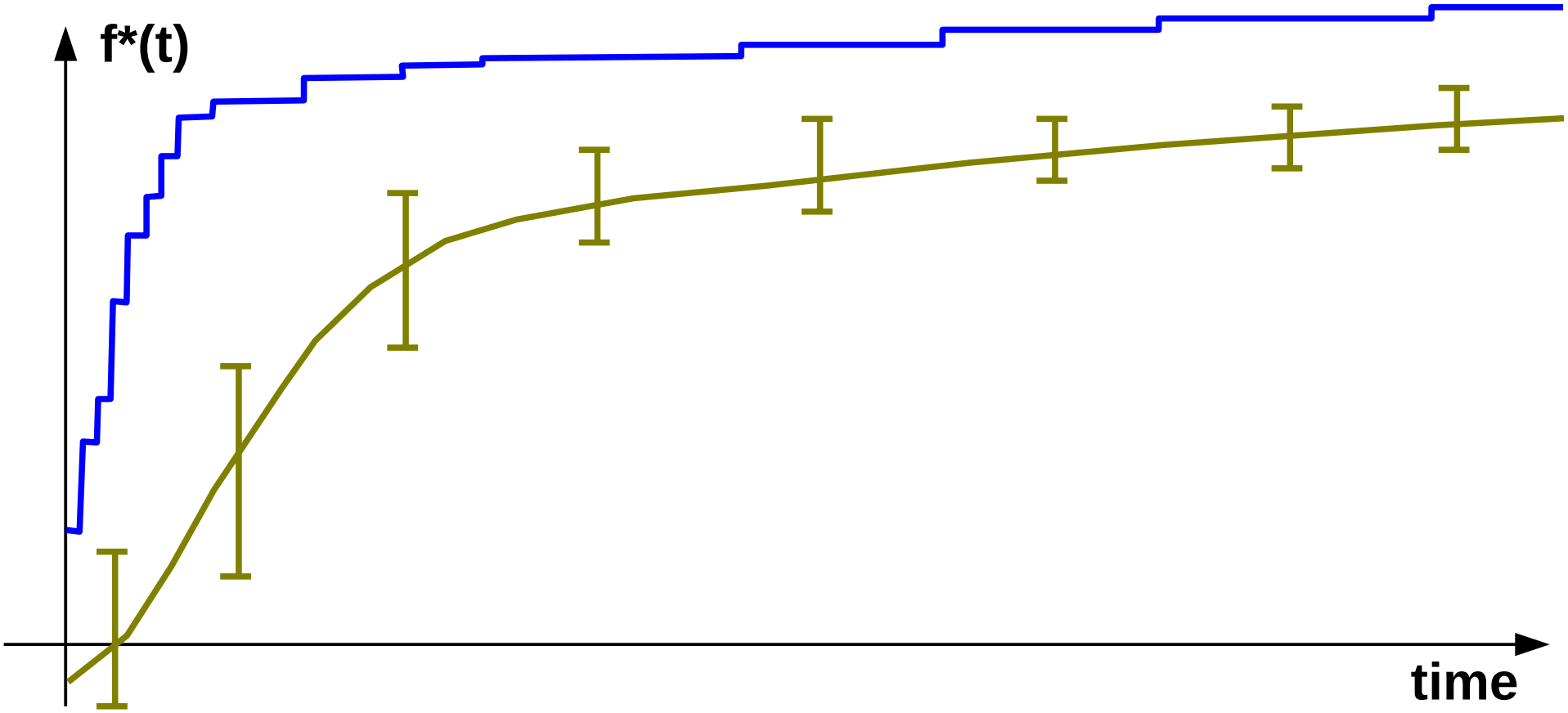
EA: Performance Graph

The performance graph is showing the development of the fitness f*(t) of the best individual in each generation with respect to time.



EA: Performance Graph

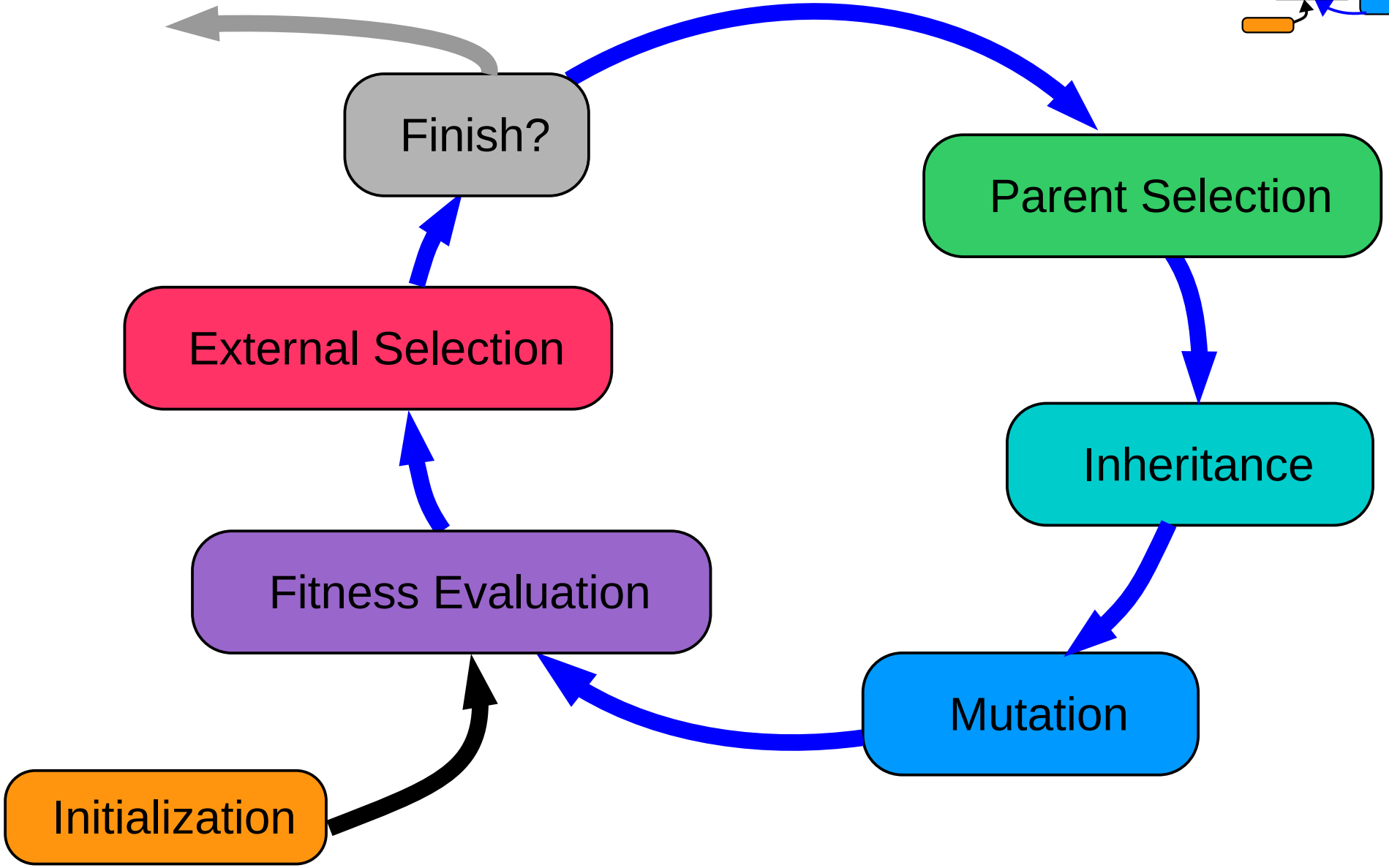
The performance graph is showing the development of the fitness f*(t) of the best individual in each generation with respect to time.



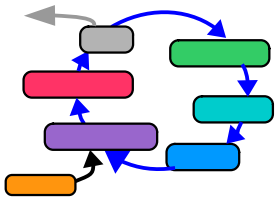
Overview

- EA Steps
  - Individual, Genome, Fitness, Population
  - Parent selection
  - Inheritance
  - Mutation
  - Fitness evaluation
  - External selection
  - Finish?
  - Initialization
- Strategy
- Performance Graph
- **Genome Structure**
- Examples

EA cycle of operations:



EA: Genome Structure

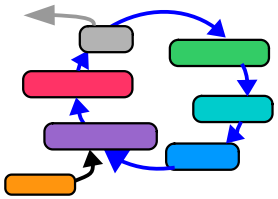


Naive structuring of the genome:

“Normal” structuring of the genome:

Sophisticated structuring of the genome:

EA: Genome Structure

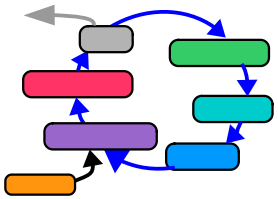


Naive structuring of the genome:
Pro: easy to implement, only few knowledge necessary.
Con: large search space, a lot of local maxima possible, may be hard or impossible to find a good solution.

“Normal” structuring of the genome:

Sophisticated structuring of the genome:

EA: Genome Structure

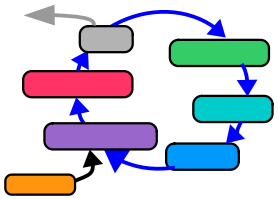


Naive structuring of the genome:
Pro: easy to implement, only few knowledge necessary.
Con: large search space, a lot of local maxima possible, may be hard or impossible to find a good solution.

“Normal” structuring of the genome:
Pro: still easy to implement, some knowledge necessary, wide variety to implement inheritance and mutation.
Con: still a lot of bad or illegal genomes possible.

Sophisticated structuring of the genome:

EA: Genome Structure



Naive structuring of the genome:
Pro: easy to implement, only few knowledge necessary.
Con: large search space, a lot of local maxima possible, may be hard or impossible to find a good solution.

“Normal” structuring of the genome:
Pro: still easy to implement, some knowledge necessary, wide variety to implement inheritance and mutation.
Con: still a lot of bad or illegal genomes possible.

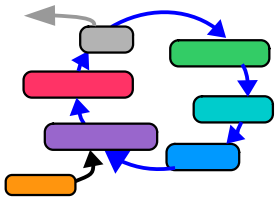
Sophisticated structuring of the genome:
Pro: only legal, or good genomes are to be investigated.
Con: profound knowledge about the process and of the kind of possible solutions is required, can become computational expensive to implement inheritance and mutation.

Overview

- EA Steps
  - Individual, Genome, Fitness, Population
  - Parent selection
  - Inheritance
  - Mutation
  - Fitness evaluation
  - External selection
  - Finish?
  - Initialization
- Strategy
- Performance Graph
- Genome Structure
- **Examples**

EA:

Example:



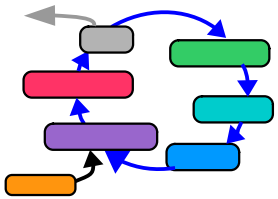
Implementing a task using an Evolutionary Algorithm,
a series of structuring decision will be necessary.

The sequence below is not common theory, but just my personal choice of doing it.

Objective:	specify the objective
Genome:	structure the genome
Fitness Function:	define an appropriate fitness function
Inheritance:	layout the inheritance process
Mutation:	layout the mutation process
Selection Strategy:	specify the selection strategy

EA:

Example: Fkt Maximum



Objective:

Genome:

Fitness Function:

EA:

Example: Fkt Maximum

Objective:
find the position **X,** where an objective function o**(X)** has its maximal value (or minimal value).

Genome:

Fitness Function:

EA:

Example: Fkt Maximum

Objective:
find the position **X,** where an objective function o**(X)** has its maximal value (or minimal value).

Genome:
the N-dimensional vector **X**
g = X = $\{ x_1, x_2, ..., x_i, \dots , x_N \}$
with N components x_1 to x_N

Fitness Function:

EA:

Example: Fkt Maximum

Objective:
find the position **X,** where an objective function o**(X)** has its maximal value (or minimal value).

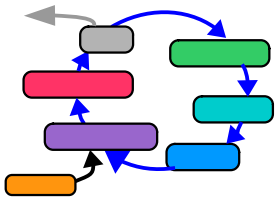
Genome:
the N-dimensional vector **X**
g = X = $\{ x_1, x_2, ..., x_i, \dots , x_N \}$
with N components x_1 to x_N

Fitness Function:
fitness function **f(g)** is identical to the objective function o**(X)**

f(g) = o**(X)**

EA:

Example: Fkt Maximum



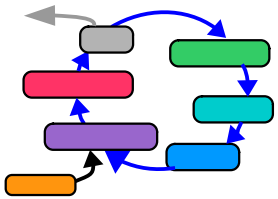
Inheritance:

Mutation:

Selection Strategy:

EA:

Example: Fkt Maximum



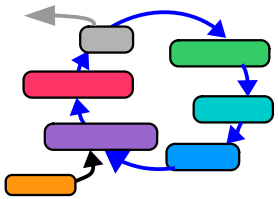
Inheritance:
recombination with k=2 parents, 1-point cross over

Mutation:

Selection Strategy:

EA:

Example: Fkt Maximum



Inheritance:
recombination with k=2 parents, 1-point cross over

Mutation:
chose a random component x_i, and change that value to a complete new value.

Selection Strategy:

EA:

Example: Fkt Maximum

Inheritance:
recombination with k=2 parents, 1-point cross over

Mutation 1:
chose a random component x_i, and change that value to a complete new value.

Mutation 2:

Selection Strategy:

EA:

Example: Fkt Maximum

Inheritance:
recombination with k=2 parents, 1-point cross over

Mutation 1:
chose a random component x_i, and change that value to a complete new value.

Mutation 2:
chose one or two random components x_i, x_j and add a small normally distributed value to them.

Selection Strategy:

EA:

Example: Fkt Maximum

Inheritance:
recombination with k=2 parents, 1-point cross over

Mutation 1:
chose a random component x_i, and change that value to a complete new value.

Mutation 2:
chose one or two random components x_i, x_j and add a small normally distributed value to them.

Selection Strategy:

EA:

Example: Fkt Maximum

Inheritance:
recombination with k=2 parents, 1-point cross over

Mutation 1:
chose a random component x_i, and change that value to a complete new value.

Mutation 2:
chose one or two random components x_i, x_j and add a small normally distributed value to them.

Selection Strategy:
elitism, μ parents, parents survive, no mutation for the parents
deterministic, rank dependent, take the μ best individuals
(μ + λ), with P=500, μ=100, λ=400.

EA:

Example: Fkt Maximum

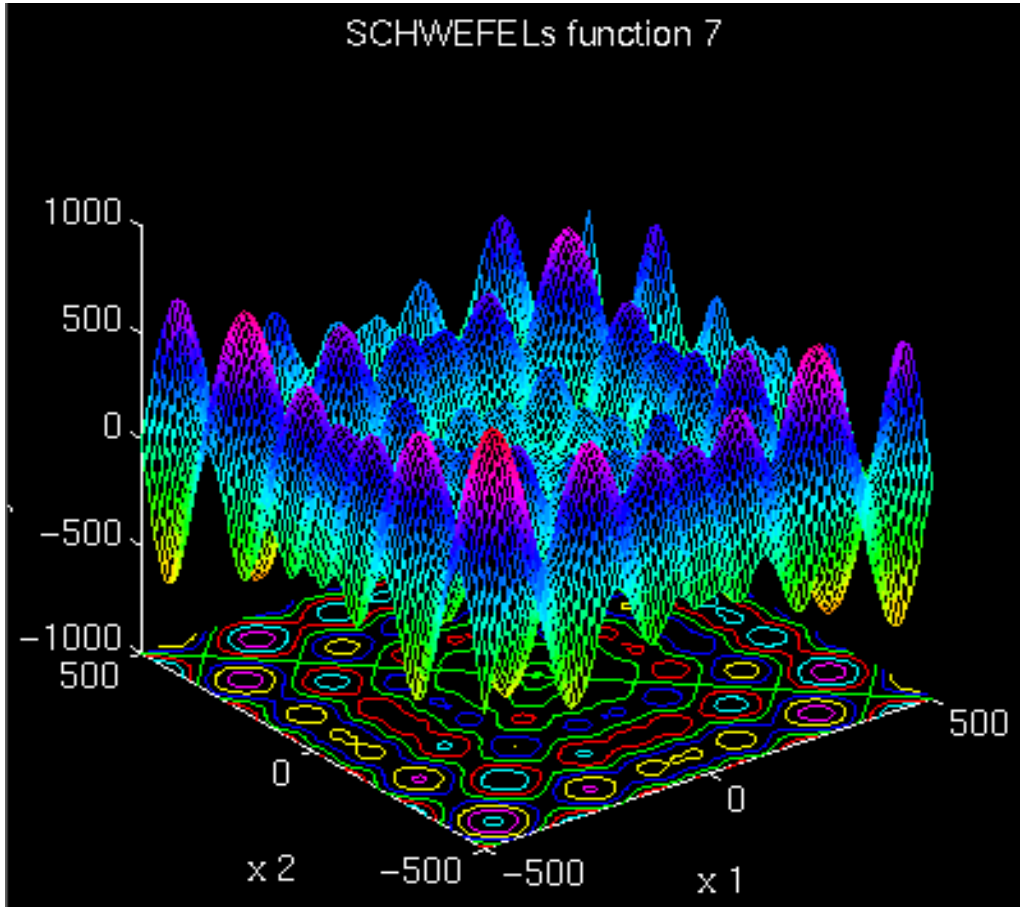
A typical **playground** to investigate the capabilities of optimzation methods are **test functions** to minimize.

Schwefel's Function (1981)

$f(\mathbf{x}) = \sum -x_j \, sin(\, sqrt(\, abs(\, x_j \,) \,) \,)$

$j = 1 \ldots d$

$-500 < x_j < 500$

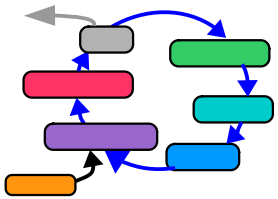


Schwefel, H.-P.:
Numerical optimization of computer models.
Chichester: Wiley & Sons, 1981.

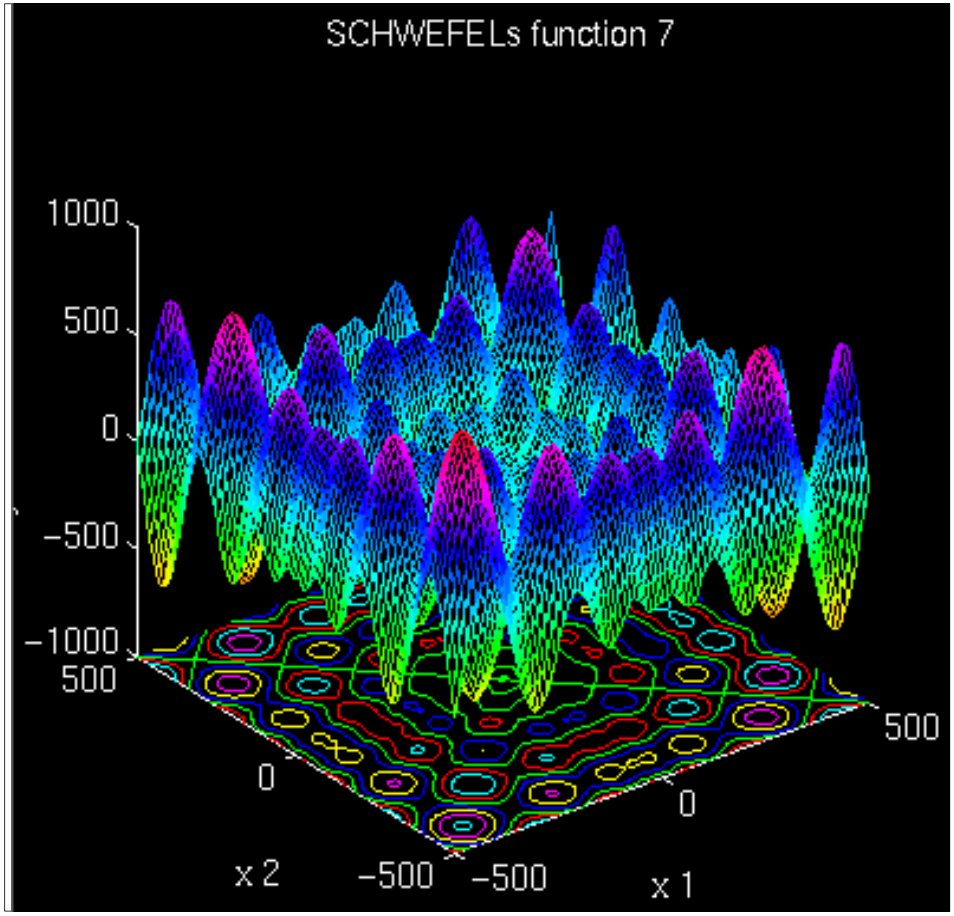
http://www.tu-ilmenau.de/fakia/fileadmin/template/startIA/oek_systeme/Interess_adressen/pohlheim/GA_Toolbox/fcnindex.html

EA:

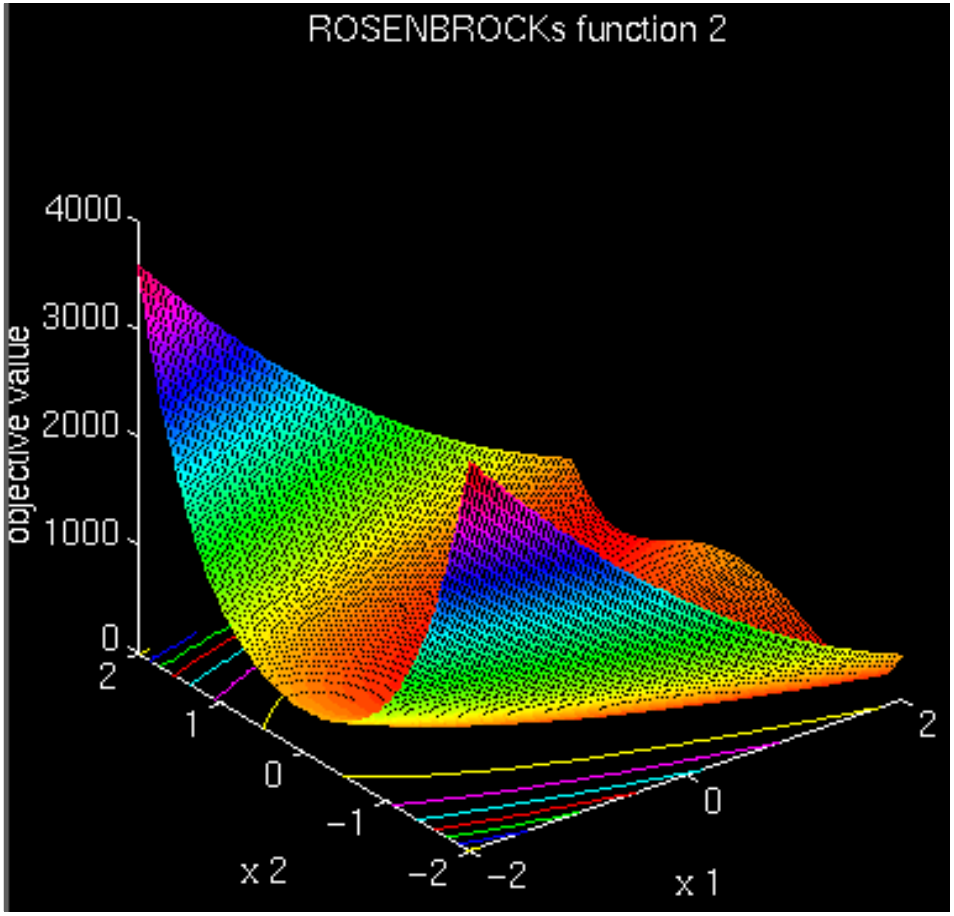
Example: Fkt Maximum



f(x)=sum(-x(i)·sin(sqrt(abs(x(i))))),
i=1:n; -500<=x(i)<=500.



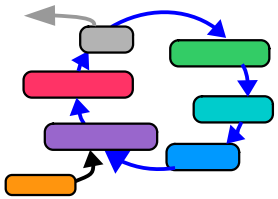
f(x)=sum(100·(x(i+1)-x(i)^2)^2+(1-x(i))^2),
i=1:n-1; -2.048<=x(i)<=2.048.



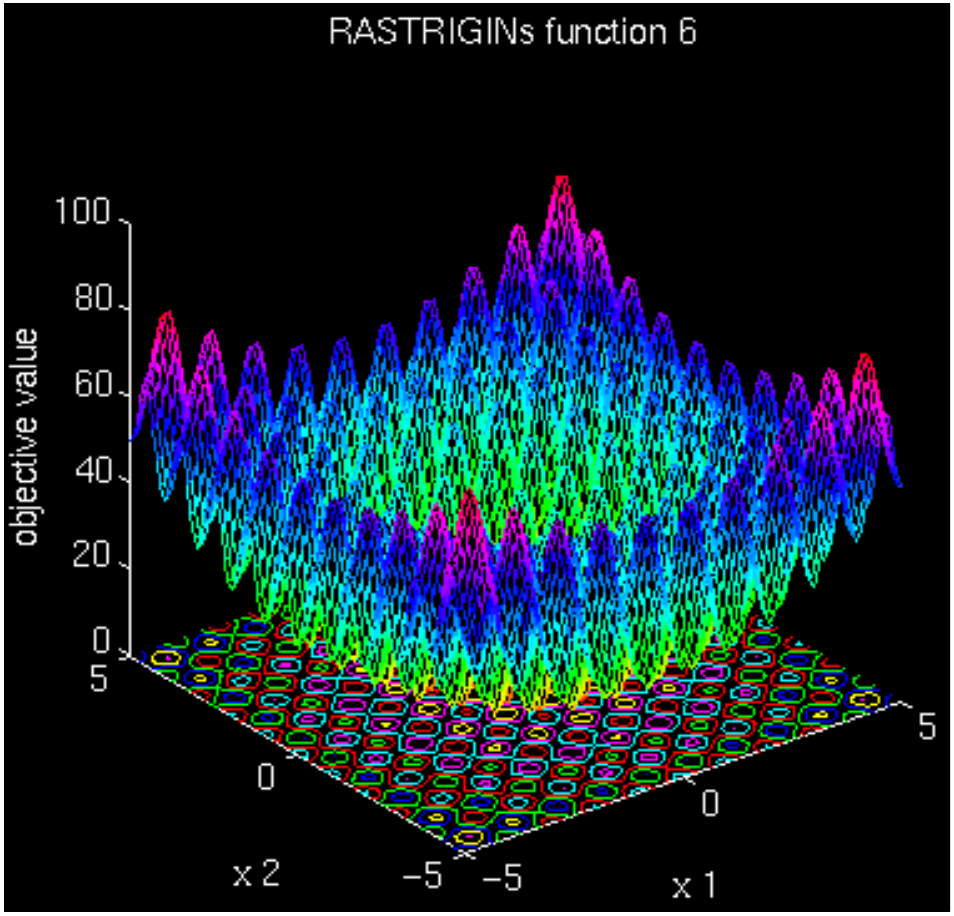
http://www.tu-ilmenau.de/fakia/fileadmin/template/startIA/oek_systeme/Interess_adressen/pohlheim/GA_Toolbox/fcnindex.html

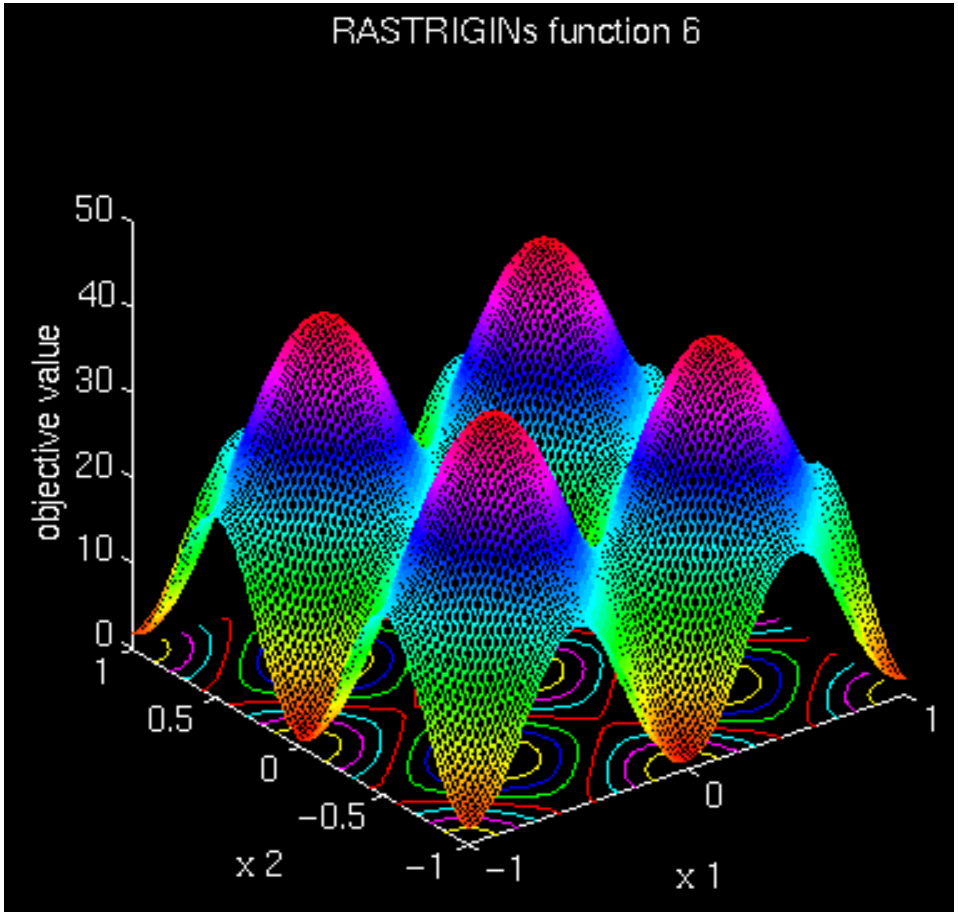
EA:

Example: Fkt Maximum



f(x)=10·n+sum(x(i)^2-10·cos(2·pi·x(i))),
i=1:n; -5.12<=x(i)<=5.12.



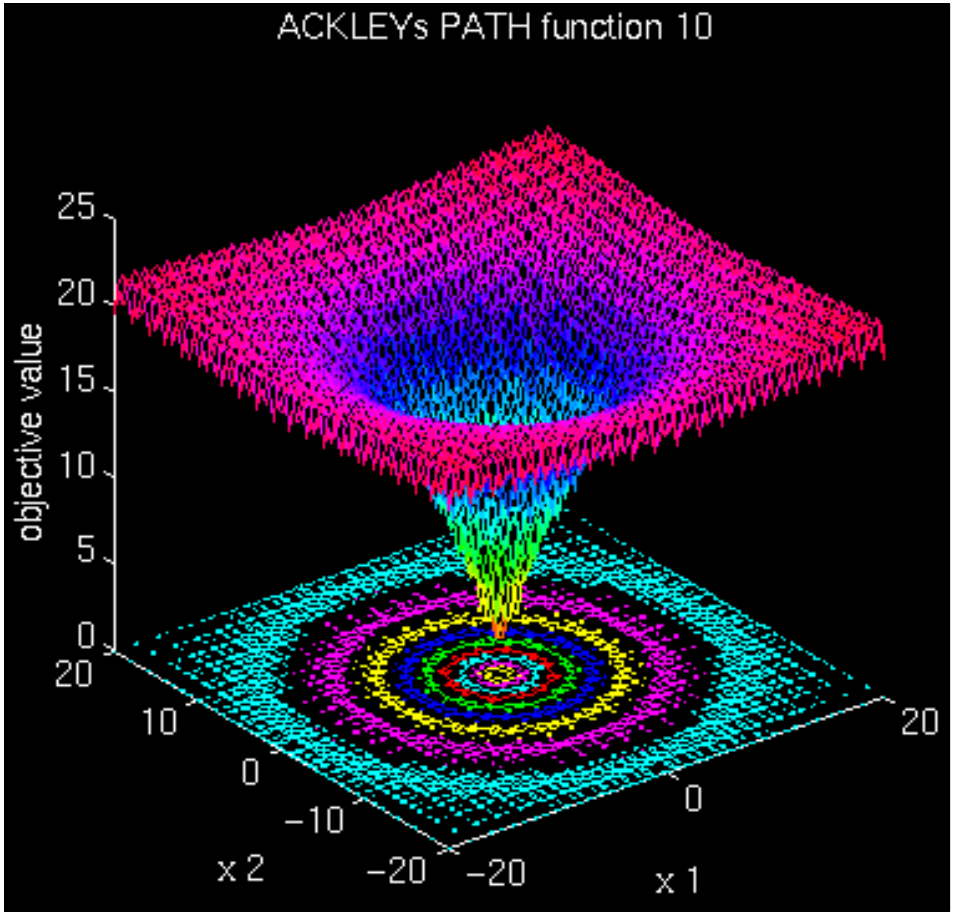


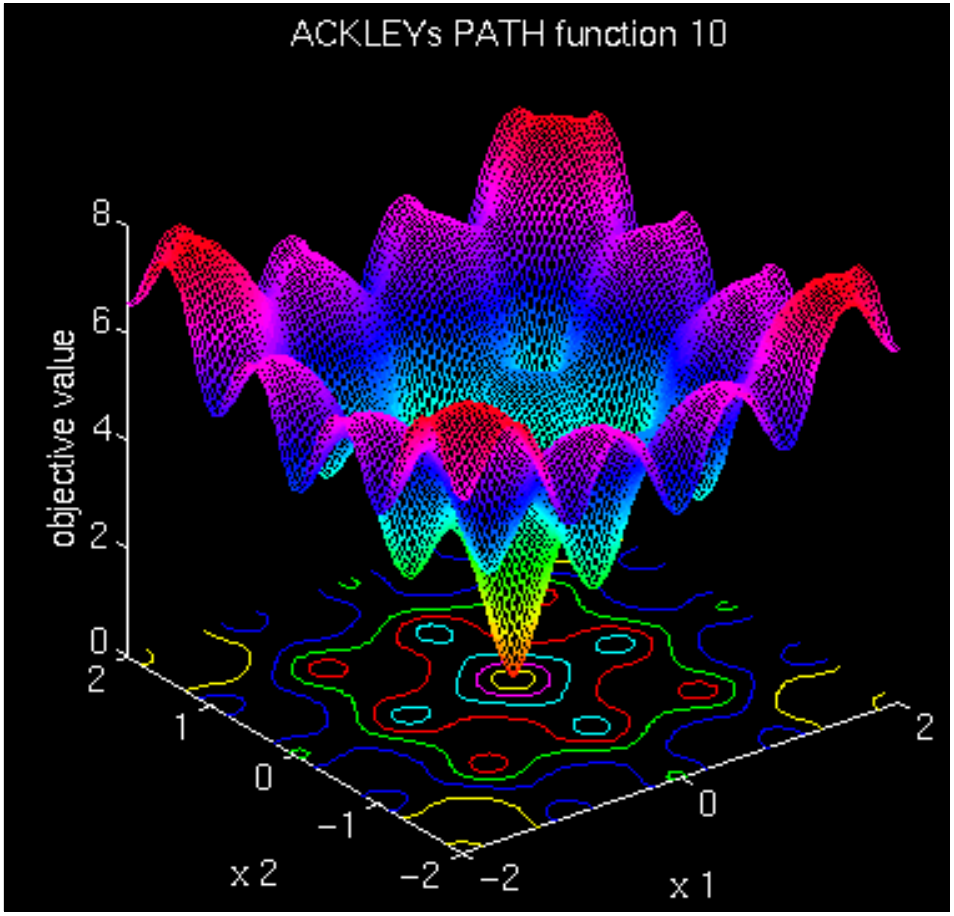
http://www.tu-ilmenau.de/fakia/fileadmin/template/startIA/oek_systeme/Interess_adressen/pohlheim/GA_Toolbox/fcnindex.html

EA:

Example: Fkt Maximum

f(x)=-a·exp(-b·sqrt(1/n·sum(x(i)^2)))-exp(1/n·sum(cos(c·x(i))))+a+exp(1);
a=20; b=0.2; c=2·pi; i=1:n;



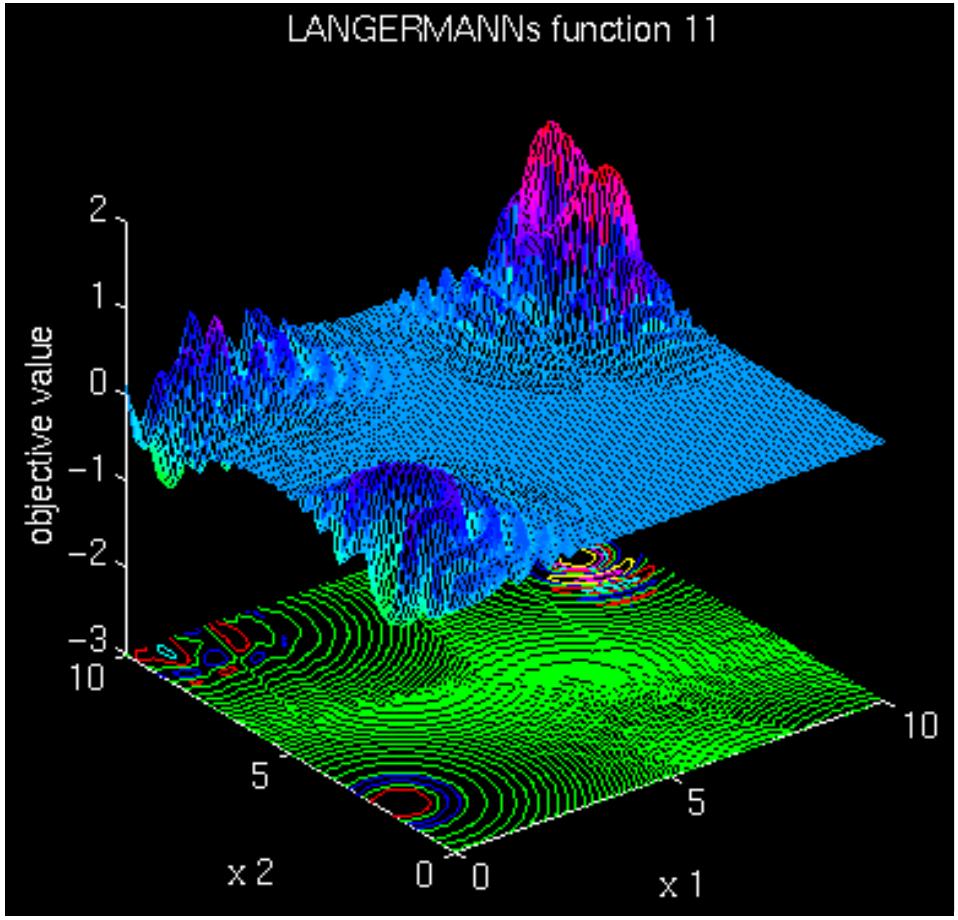


http://www.tu-ilmenau.de/fakia/fileadmin/template/startIA/oek_systeme/Interess_adressen/pohlheim/GA_Toolbox/fcnindex.html

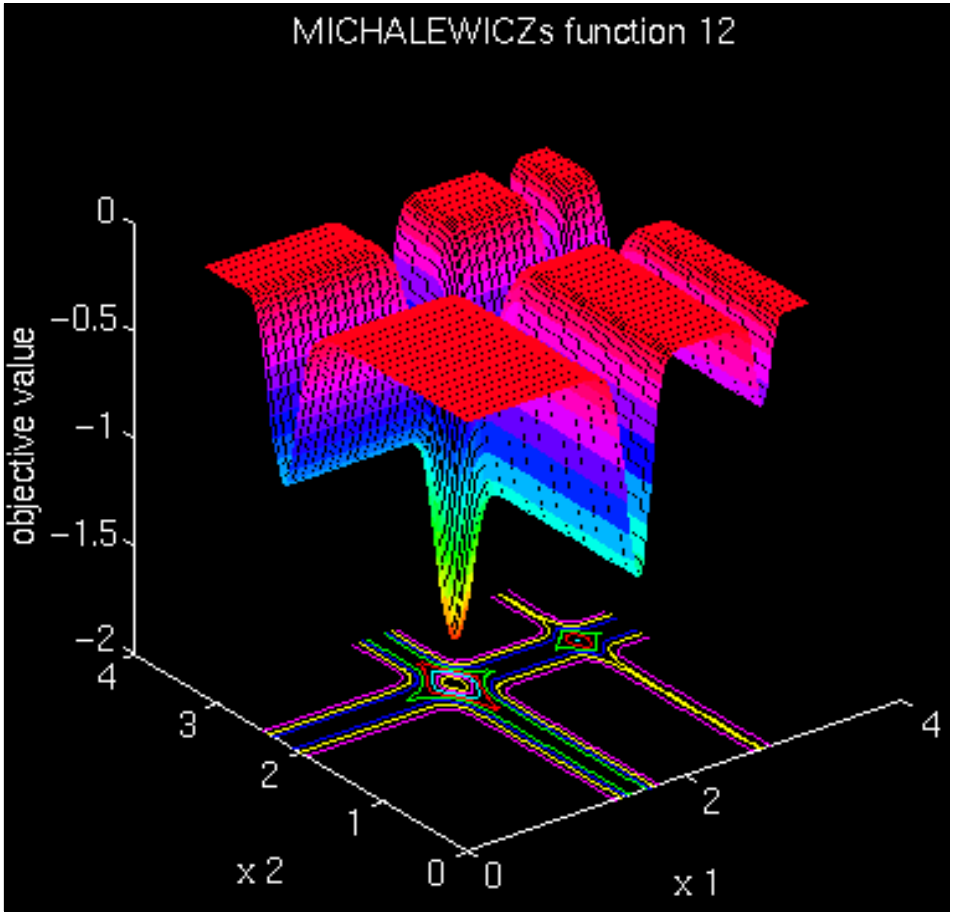
EA:

Example: Fkt Maximum

f(x)=-sum(c(i)·(exp(-1/pi·sum((x-A(i))^2))·cos(pi·sum((x-A(i))^2)))),
i=1:m, m=5



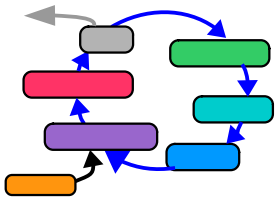
f(x)=-sum(sin(x(i))·(sin(i·x(i)^2/pi))^(2·m)),
i=1:n, m=10;



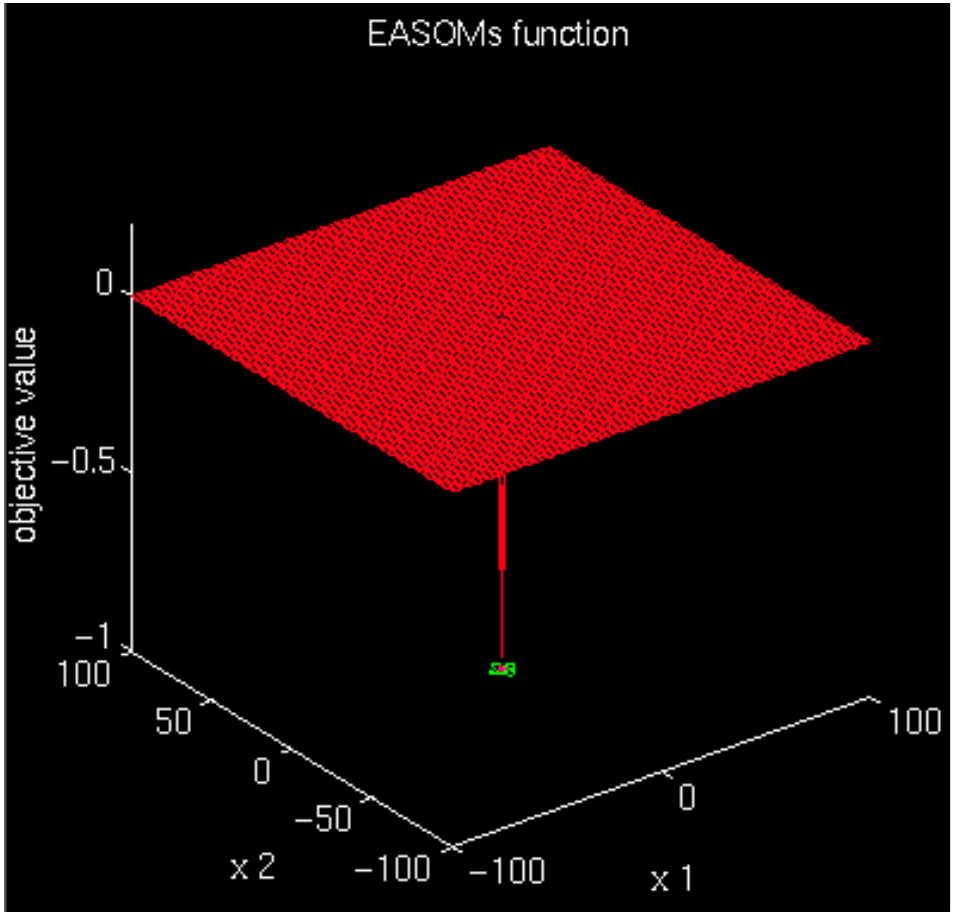
http://www.tu-ilmenau.de/fakia/fileadmin/template/startIA/oek_systeme/Interess_adressen/pohlheim/GA_Toolbox/fcnindex.html

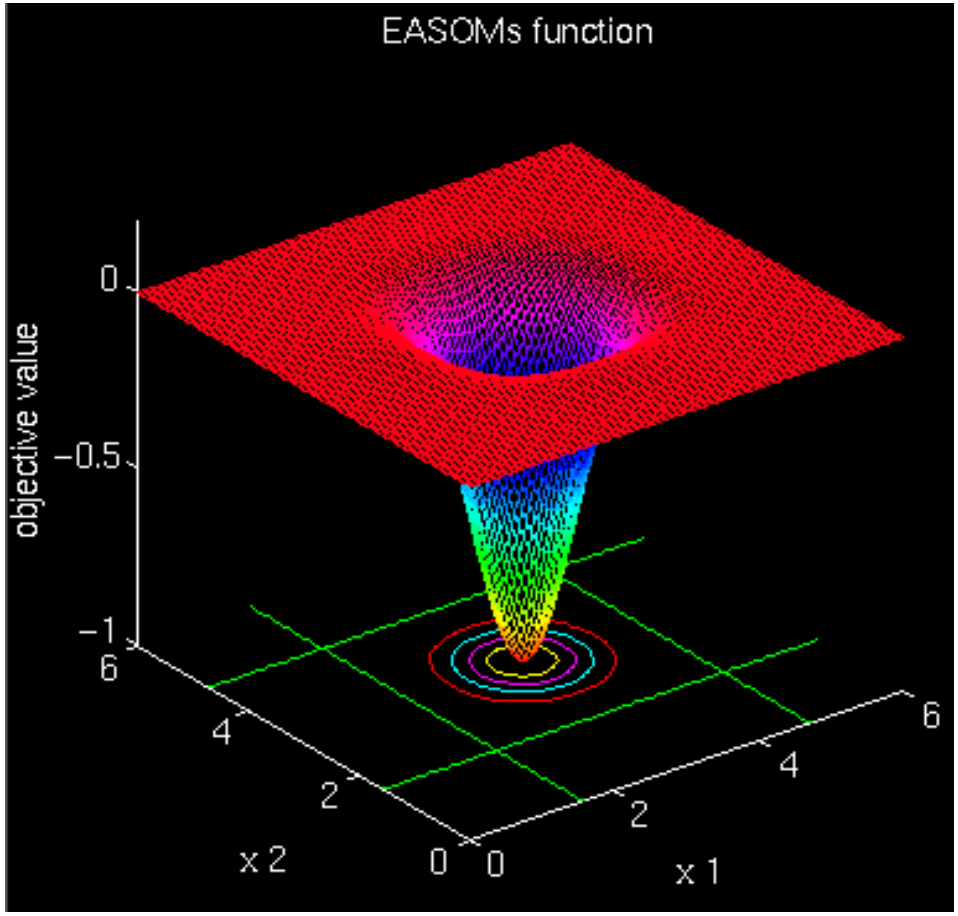
EA:

Example: Fkt Maximum



f(x1,x2) = -cos(x1)·cos(x2)·exp(-((x1-pi)^2+(x2-pi)^2));
-100<=x(i)<=100, i=1:2.





http://www.tu-ilmenau.de/fakia/fileadmin/template/startIA/oek_systeme/Interess_adressen/pohlheim/GA_Toolbox/fcnindex.html

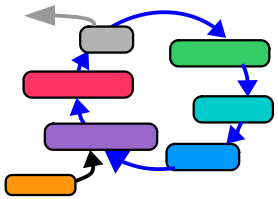
EA: Example: Fkt Maximum

Below is a list of common **test functions (to minimize)** that are used to investigate the capabilities of optimization methods:

- Schwefel's Function
- Generalized Rosenbrock's Function
- Rastrigin's Function
- Ackley's (path) Function
- Langermann's Function
- Michalewicz's Function
- Easoms Function
- Griewangk's Function
- Bohachevsky's Function
- Watson's Function
- Colville's Function
- ...

EA:

Example: Sorting

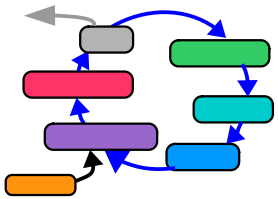


Objective:

Genome:

(easy) Fitness Function:

EA: Example: Sorting



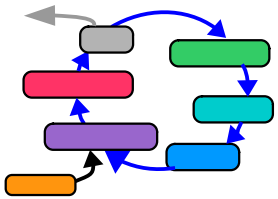
Objective:
sort a set of numerical values x_i with respect to the square x_i^2 of their value.

Genome:
an ordered set of values (x_i), $\mathbf{g} = \{ x_1, x_2, \ldots, x_i, \ldots, x_{42}, \ldots \}$

(easy) Fitness Function:
f(g) is the accumulation of all comparisons of subsequent values
add a **+1** if the order is correct, and a **-1** if the order is wrong.

$$f(g) = \sum sign(x_i^2 - x_{i+1}^2)$$

EA: Example: Sorting



Easy Inheritance:

since a classical cross over will generate illegal genomes
the k=1, copy operator with no recombination is fine.

Mutation:

exchange 2 values within the sequence (genome),
or even a complete subsequence,
or invert the order of a subsequence within the genome.

Selection Strategy:

elitism, μ parents, parents survive, no mutation for the parents
deterministic, rank dependent, take the μ best individuals
(μ + λ), with P=500, μ=100, λ=400.

EA:

Example: 42

Objective:
find the L-bit binary representation of the square root of 42; unfortunately, the math. function *sqrt* is not available.

Genome:

Fitness function:

EA:

Example: 42

Objective:
find the L-bit binary representation of the square root of 42; unfortunately, the math. function *sqrt* is not available.

Genome:
an L-bit binary vector **g,** in fix-point representation

Fitness function:

EA:

Example: 42

Objective:
find the L-bit binary representation of the square root of 42;
unfortunately, the math. function *sqrt* is not available.

Genome:
an L-bit binary vector **g,** in fix-point representation

Fitness function:
take the genome **g** as a binary, fix point number **x(g)**
calculate the square of x(g),
compare the result to 42.0, the square of the objective number

$$f(g) = | 42.0 - x(g)^2 |$$

EA:

Example: 42

Objective:
find the L-bit binary representation of the square root of 42;
unfortunately, the math. function *sqrt* is not available.

Genome:
an L-bit binary vector **g,** in fix-point representation

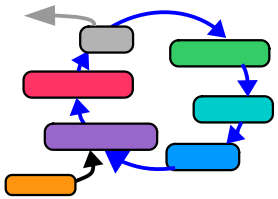
Fitness function:
take the genome **g** as a binary, fix point number **x(g)**
calculate the square of x(g),
compare the result to 42.0, the square of the objective number

$$f(g) = | 42.0 - x(g)^2 |$$

In fact, this is a *cost function,* that is to be minimized.

EA:

Example: 42



Population:
P = 100, population size kept constant.

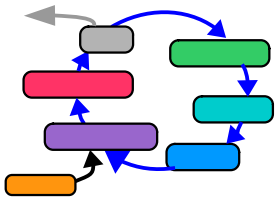
Inheritance:
recombination, k=2 parents, 1-point cross over.

Mutation:
bit-flip, flip each bit of the genome with a probability ω **= 0.01**
Caution: be aware of Hamming-cliffs.

Selection Strategy:
elitism, μ parents, parents survive, no mutation for the parents
deterministic, rank dependent, take the μ best individuals
(μ + λ), with μ=20, λ=80.

EA:

Example: TSP



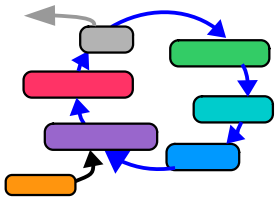
Objective:
find the shortest route visiting each point from a set of given points (cities) exactly once (Traveling Salesman Problem).

Genome:

Fitness Function:

EA:

Example: TSP



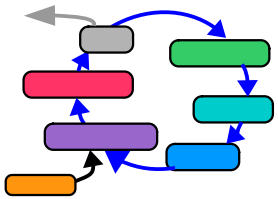
Objective:
find the shortest route visiting each point from a set of given points (cities) exactly once (Traveling Salesman Problem).

Genome:
an ordered set of points (cities, C_i), sequence of cities to be visited, containing each city exactly once.
g = { C_3, C_{17}, C_i, C_{42}, ...}

Fitness Function:

EA:

Example: TSP



Objective:
find the shortest route visiting each point from a set of given points (cities) exactly once (Traveling Salesman Problem).

Genome:
an ordered set of points (cities, C_i), sequence of cities to be visited, containing each city exactly once.
g = { C_3, C_{17}, C_i, C_{42}, ...}

Fitness Function:
f(g) is the sum of all distances **$d(C_i,C_{i+1})$** between subsequent cities in the list (genome). Of course all distances **$d(C_i,C_j)$** between the cities i and j is required; either by a distance matrix or by a calculation from the coordinates.

EA:

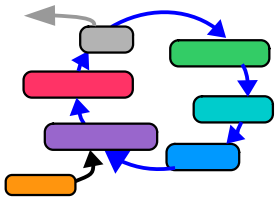
Example: TSP

Easy Inheritance:

Mutation:

EA:

Example: TSP



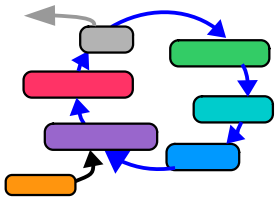
Easy Inheritance:

Since a classical cross over will generate illegal genomes the k=1, copy operator, omitting recombination is O.K.

Mutation:

EA:

Example: TSP

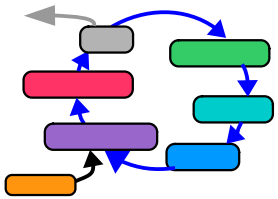


Easy Inheritance:
Since a classical cross over will generate illegal genomes the k=1, copy operator, omitting recombination is O.K.

Mutation:
exchange 2 cities within the sequence (genome),
or even a complete subsequence,
or invert the order of a subsequence within the genome.

EA:

Example: TSP



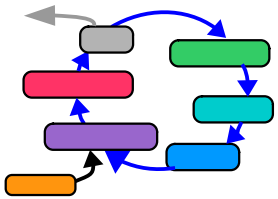
Easy Inheritance:
Since a classical cross over will generate illegal genomes
the k=1, copy operator, omitting recombination is O.K.

Mutation:
exchange 2 cities within the sequence (genome),
or even a complete subsequence,
or invert the order of a subsequence within the genome.

g = { … , London, Rio, Moscow, New York, Paris, Tokyo, LA, ... }

EA:

Example: TSP



Easy Inheritance:
Since a classical cross over will generate illegal genomes the k=1, copy operator, omitting recombination is O.K.

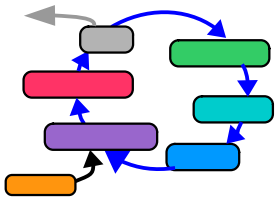
Mutation:
exchange 2 cities within the sequence (genome),
or even a complete subsequence,
or invert the order of a subsequence within the genome.

g = { … , London, Rio, Moscow, New York, Paris, Tokyo, LA, ... }

g' = { … , London, Paris, Moscow, New York, Rio, Tokyo, LA, ... }

EA:

Example: TSP



Easy Inheritance:
Since a classical cross over will generate illegal genomes the k=1, copy operator, omitting recombination is O.K.

Mutation:
exchange 2 cities within the sequence (genome),
or even a complete subsequence,
or invert the order of a subsequence within the genome.

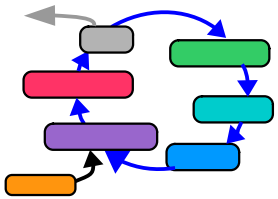
g = { … , London, Rio, Moscow, New York, Paris, Tokyo, LA, ... }

g' = { … , London, Paris, Moscow, New York, Rio, Tokyo, LA, ... }

g'' = { … , London, Tokyo, Rio, Moscow, New York, Paris, LA, ... }

EA:

Example: TSP



Easy Inheritance:
Since a classical cross over will generate illegal genomes the k=1, copy operator, omitting recombination is O.K.

Mutation:
exchange 2 cities within the sequence (genome),
or even a complete subsequence,
or invert the order of a subsequence within the genome.

g = { … , London, Rio, Moscow, New York, Paris, Tokyo, LA, ... }

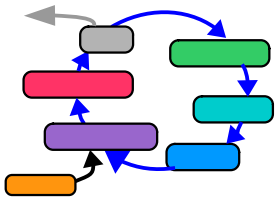
g' = { … , London, Paris, Moscow, New York, Rio, Tokyo, LA, ... }

g'' = { … , London, Tokyo, Rio, Moscow, New York, Paris, LA, ... }

g''' = { … , London, Paris, New York, Moscow, Rio, Tokyo, LA, ... }

EA:

Example: TSP



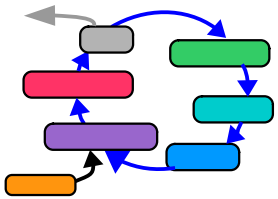
Sophisticated Inheritance:

A more sophisticated inheritance operator would have to guarantee that the offspring still represent legal genomes,
- routes that visit each city exactly once - .
This may cause high computational effort,
which might be larger than the potential benefit.

Sophisticated Mutation:

EA:

Example: TSP



Sophisticated Inheritance:

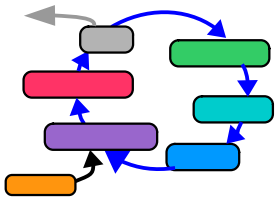
A more sophisticated inheritance operator would have to guarantee that the offspring still represent legal genomes,
- routes that visit each city exactly once - .
This may cause high computational effort,
which might be larger than the potential benefit.

Sophisticated Mutation:

a more sophisticated mutation operator might take the fitness value f(g) of the genome g into account, to make only those changes to the genome that have a high probability for a fitness increase.

EA:

Example: TSP



Sophisticated Inheritance:

A more sophisticated inheritance operator would have to guarantee that the offspring still represent legal genomes,
- routes that visit each city exactly once - .
This may cause high computational effort,
which might be larger than the potential benefit.

Sophisticated Mutation:

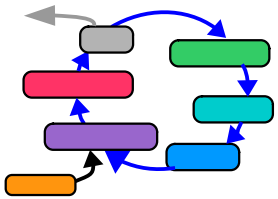
a more sophisticated mutation operator might take the fitness value f(g) of the genome g into account, to make only those changes to the genome that have a high probability for a fitness increase.
How do you know?
Either by theory (if available), or by a heuristics.

EA:

Example: 8 Queens

EA:

Example: 8 Queens



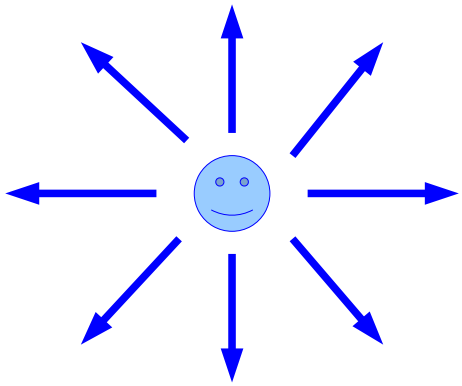
The 8 queens puzzle with an evolutionary algorithm:

The task of the 8 queens puzzle is to place 8 queens on a chess board (8x8) so that they can not reach each other.

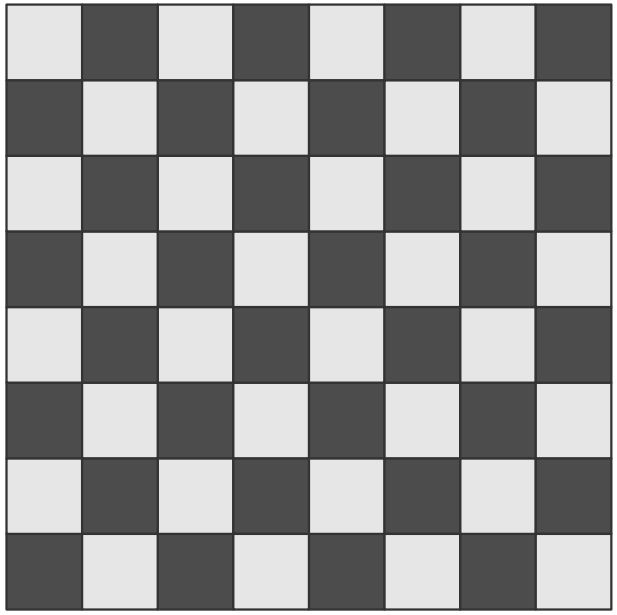
EA: Example: 8 Queens

The 8 queens puzzle with an evolutionary algorithm:

The task of the 8 queens puzzle is to place 8 queens on a chess board (8x8) so that they can not reach each other.



allowed moves for a queen

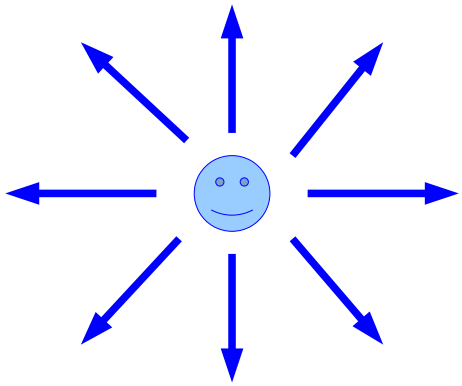


chess board with 8x8 = 64 positions

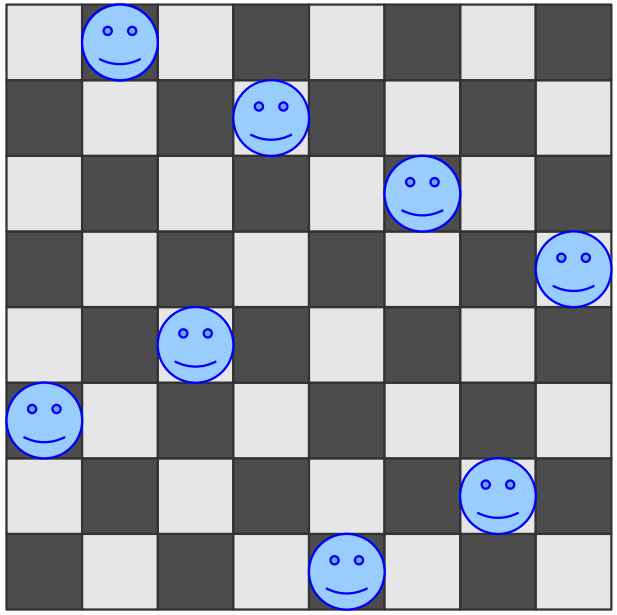
EA: Example: 8 Queens

The 8 queens puzzle with an evolutionary algorithm:

The task of the 8 queens puzzle is to place 8 queens on a chess board (8x8) so that they can not reach each other.



allowed moves for a queen



One of the **92** solutions

chess board with 8x8 = 64 positions

EA:

Example: 8 Queens

Very naive implementation of the genome:

a 64 bit binary vector; queen is 1, no queen is 0;
more than 8 queens are possible.

This very naive implementation is generating an extremely large search space with 2^64 possible genomes. This is beyond any computing power to be investigated in total.

{  }

Semi naive implementation of the genome:

a 64 bit binary vector; queen is 1, no queen is 0;
exactly 8 queens == 8 bits set are possible.

Still the resulting search space is very large.

EA:

Example: 8 Queens

Normal/Sophisticated implementation of the genome:

8 rows of 8 bit binary vectors; queen is 1, no queen is 0; each row contains exactly one queen.

This is reducing the search space to $8^8 = 16777216$ possibilities, (which can be managed by brute-force).

EA:

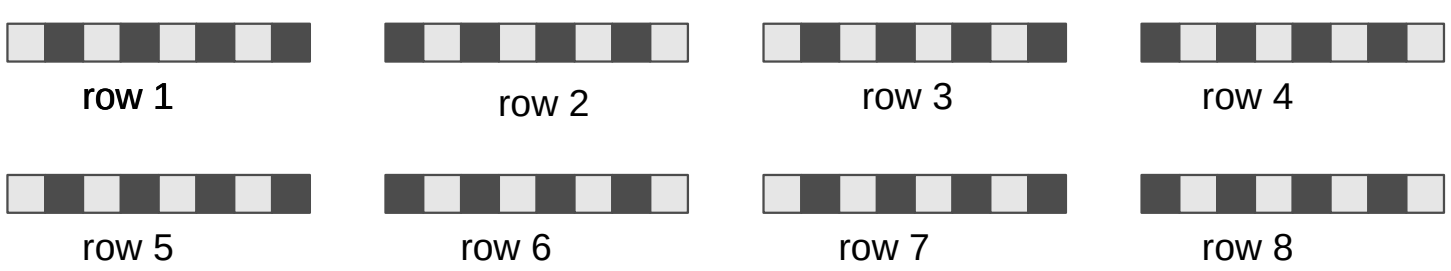
Example: 8 Queens

Normal/Sophisticated implementation of the genome:

8 rows of 8 bit binary vectors; queen is 1, no queen is 0; each row contains exactly one queen.

This is reducing the search space to $8^8 = 16777216$ possibilities, (which can be managed by brute-force).

Genome =

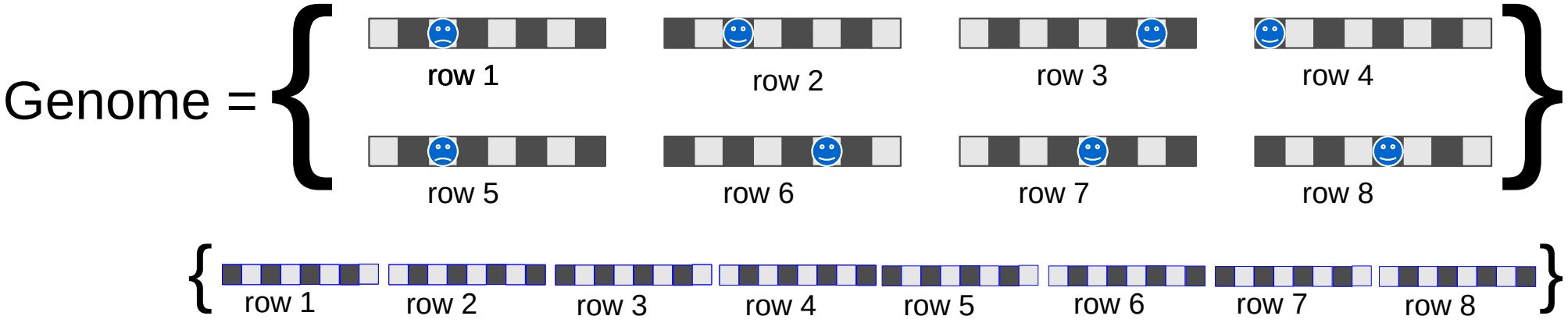


EA: Example: 8 Queens

Normal/Sophisticated implementation of the genome:

8 rows of 8 bit binary vectors; queen is 1, no queen is 0; each row contains exactly one queen.

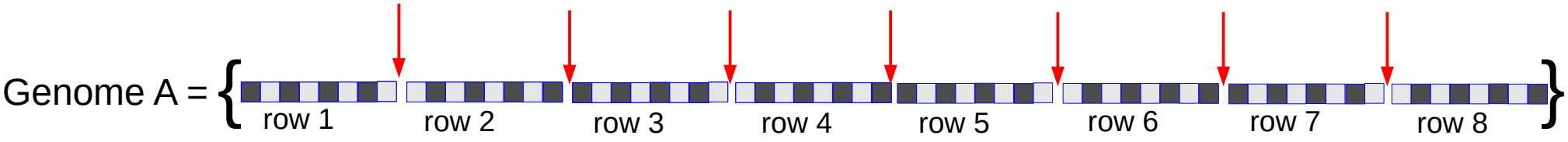
This is reducing the search space to $8^8 = 16777216$ possibilities, (which can be managed by brute-force).



EA:

Example: 8 Queens

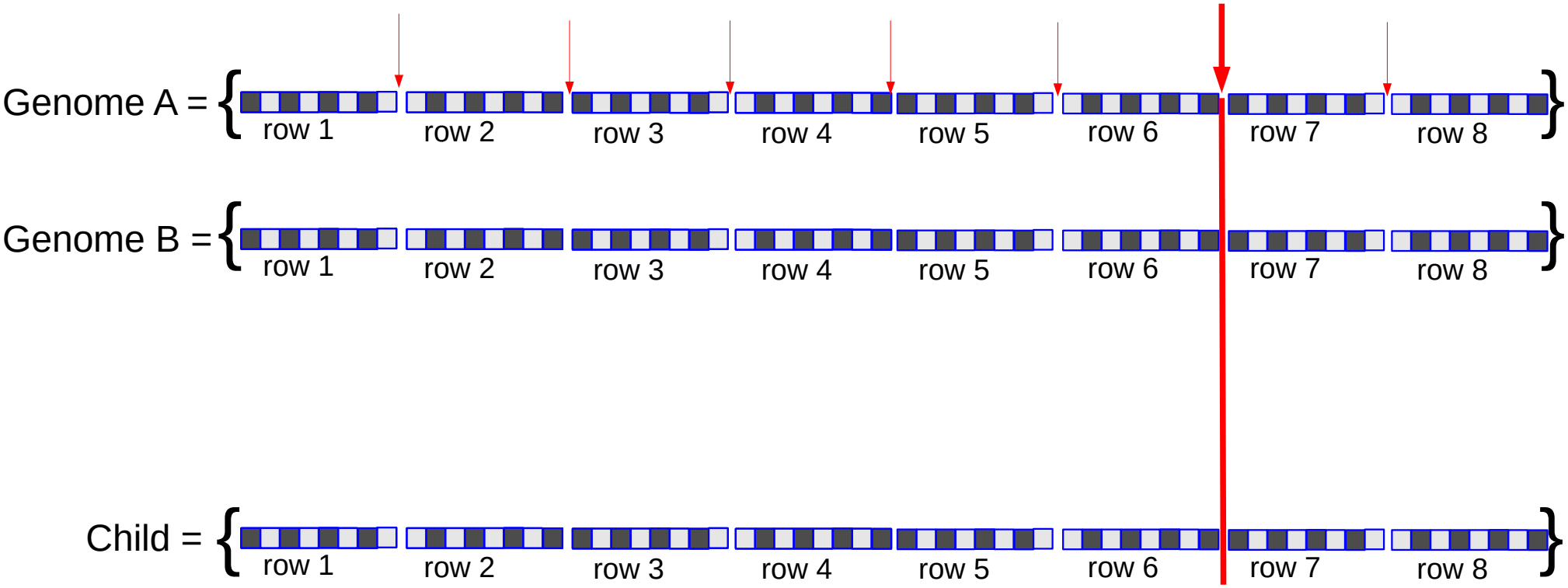
Inheritance, recombination, 1 point cross over
cross-over points only between the rows



EA:

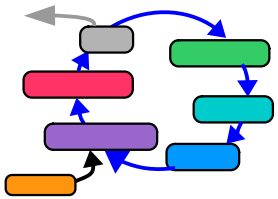
Example: 8 Queens

Inheritance, recombination, 1 point cross over
cross-over points only between the rows

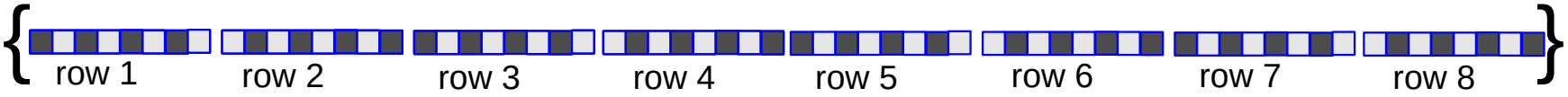


EA:

Example: 8 Queens



Mutation only the position of the queen within the row is altered

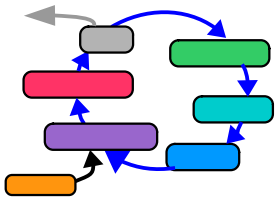
Genome A = 

Chose a random row **r** (1, …, 8):
and pick a new random position (1, …, 8) for the queen

before mutation row **3**

after mutation row **3**

EA: Example: 8 Queens



A more sophisticated implementation of the genome:

If you have an even more sophisticated idea for structuring the genome, don't forget to shape the inheritance and mutation operators and the fitness function accordingly.

Go ahead,

feel free to do experiments

EA: Example: 8 Queens

The objective, is to find a placement of the 8 queens so that they can't reach each other.

The fitness function for the EA should reflect this:

a large value of **f** if the placement is o.k. **I**
a small value of **f** if the placement is not o.k. **O**

A fitness function that yields a binary value (**O**,**I**) is not a good idea for an evolutionary algorithm.

The resulting fitness surface is flat (**O**), with only a few isolated peeks (**I**).

The value of this kind of fitness function is not reflecting, that a genome can be close to a possible solution.

EA:

Example: 8 Queens

The objective, is to find a placement of the 8 queens so that they can't reach each other.

An appropriate fitness function for an EA must be shaped accordingly;

In addition, the fitness function **f** value should implement, that genomes **g,** close to an optimal solution should yield a larger fitness value **f(g)** than those far away.

The optimal would be, to have a fitness function **f(g)** that is proportional to the distance between the genome **g** to an optimal genome **g***.

Unfortunately this is not possible in general.

EA:

Example: 8 Queens

The objective, is to find a placement of the 8 queens so that they can't reach each other.

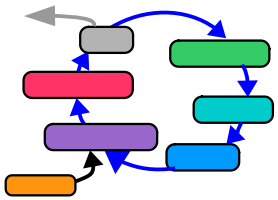
A proposal for a fitness function for the 8 queens problem:

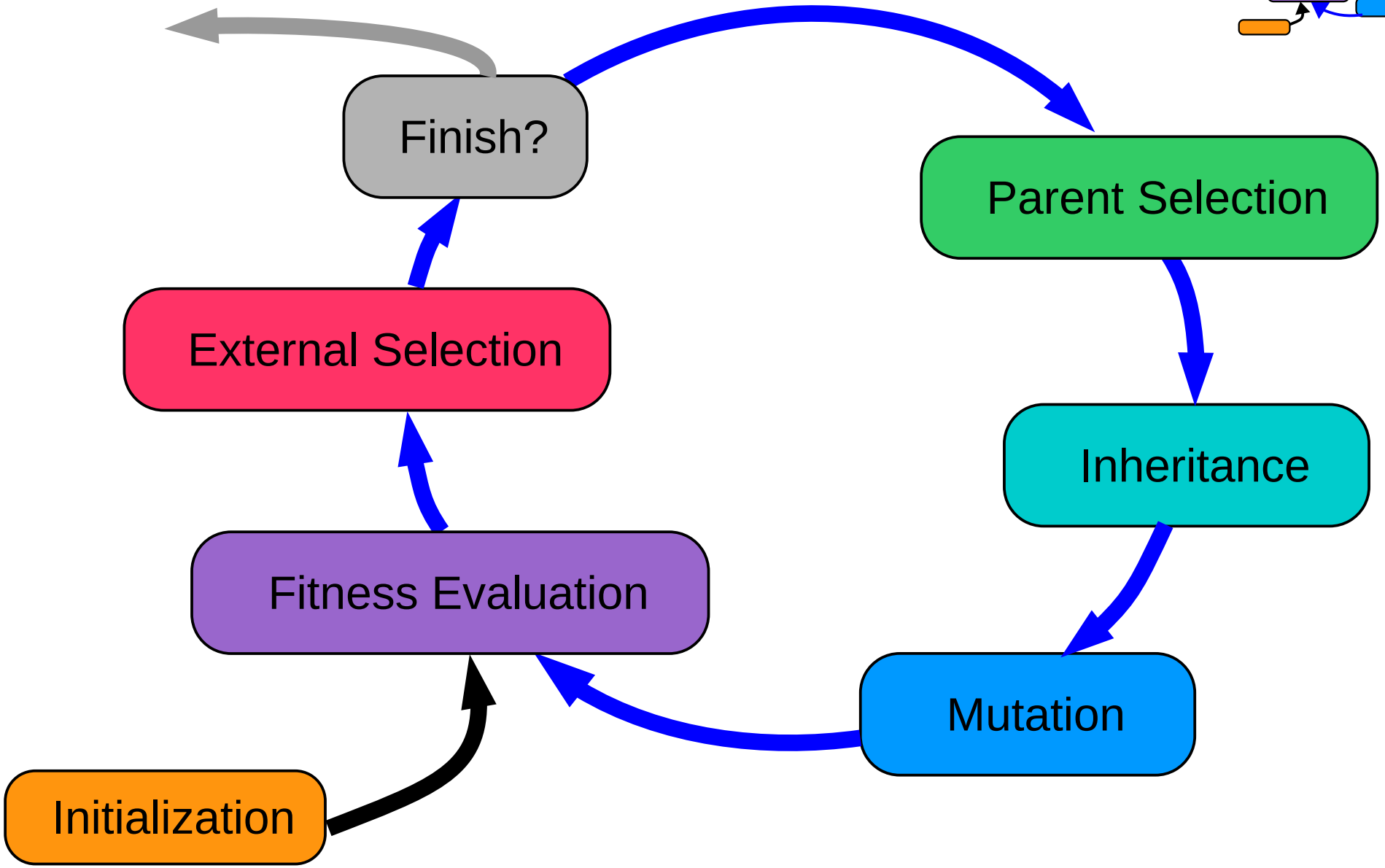
Each possibilty that one queen can attack another queen is counted as **-1**
The fitness value **f(g)** is the sum of all attack possibilities.

This yields a graded response as required,
with a maximal value **f(g*) = 0.0** when no attack is possible.

Caution:
This fitness function is only appropriate if the number of queens is fixed to 8 *(0 queens => 0 attack possibilities)*

EA cycle of operations:





Overview

- EA Steps
  - Individual, Genome, Fitness, Population
  - Parent selection
  - Inheritance
  - Mutation
  - Fitness evaluation
  - External selection
  - Finish?
  - Initialization
- Strategy
- Performance Graph
- Genome Structure
- Examples

Some important dates

Wed 14.5.25: **Dies Academicus** , special talks, no regular teaching

Thu 29.5.25 : Ascension Day , no lectures, no exercises, ...

Sun 8.6. - 9.6.25 : **Pentecost, Whitsun, Pfingsten**, Public holiday
Tue 10.6.25 – Fri 13.6.25 : Excursion week, no lectures, exercises, …

Thu 19.6.25 : **Feast of Corpus Christi**, no lectures, exercises, …

Some important dates

Thu 29.5.25 : Ascension Day (Chr. Himmelfahrt),

University is closed (building is closed, no exercises, lectures ...)

shops are closed, restaurants are typically open.

Since Ascension Day is also known as Father's Day (Vatertag) or Men's Day (Männertag, Herrentag) in some parts of Germany,

groups of male friends or male relatives spend a day together.

They often take part in an outdoor activity,

such as a walk in the country or a horse-and-cart ride.

There might be alcohol involved.

Artificial Life
Summer 2025

Evolutionary Algorithms 2

Master Computer Science [MA-INF 4201]
Mon 14:15 – 15:45, HSZ, HS-2

Dr. Nils Goerke, Autonomous Intelligent Systems,
Department of Computer Science, University of Bonn

Artificial Life
Summer 2025

Evolutionary Algorithms 2

Thank you for listening

Dr. Nils Goerke, Autonomous Intelligent Systems,
Department of Computer Science, University of Bonn