Parallel Cooperative Optimization ***Research Group***

Reusable Design of Metaheuristics for Optimization

http://paradiseo.gforge.inria.fr

Paradiseo

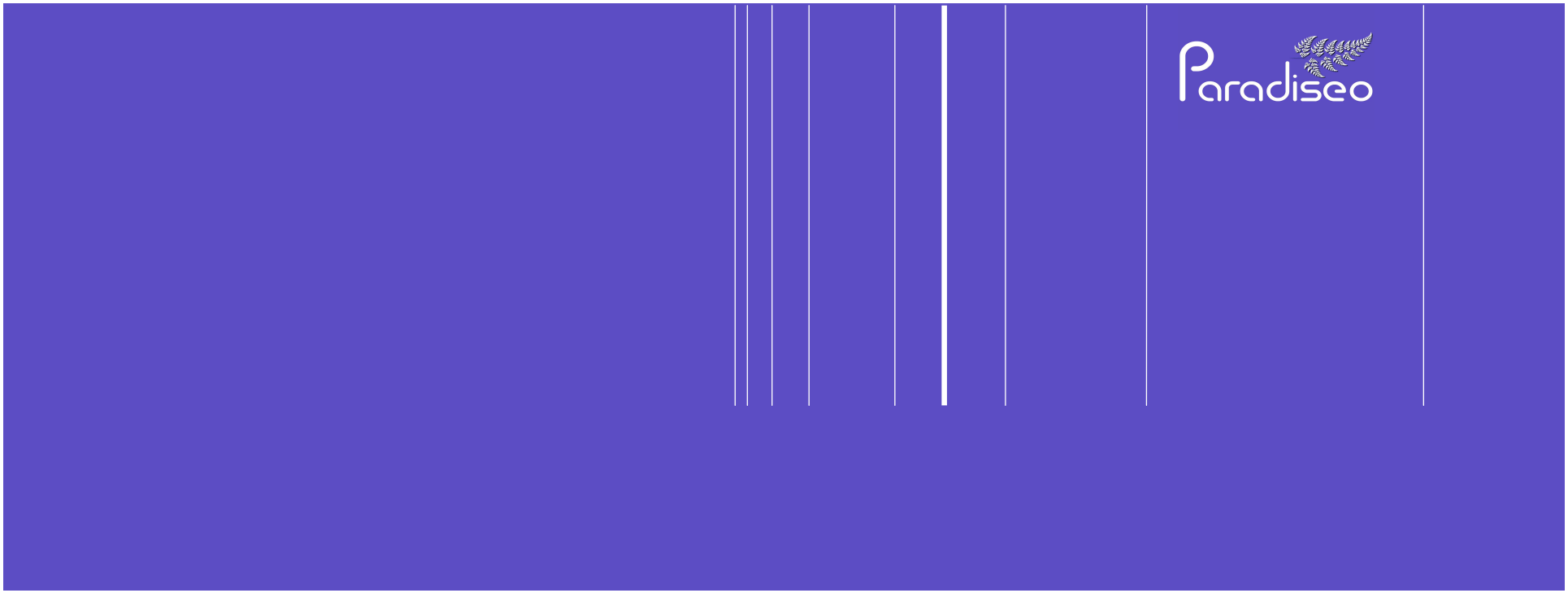
Laboratoire d'Informatique Fondamentale de Lille

INRIA

Features

- Evolutionary Algorithms
- Computational Intelligence : Particle Swarm Optimization
- Applications with 

Evolutionary Algorithms



The main steps to build an Evolutionary Algorithm

1. Design a representation
2. Decide how to initialize a population
3. Design a way of evaluating an individual
5. Design suitable mutation operator(s)
6. Design suitable recombination operator(s)
7. Decide how to manage the population
8. Decide the selection strategy
9. Decide the replacement strategy
10. Decide the continuation criterion

Framework and tutorial application

- Framework dedicated to metaheuristics

  → Paradiseo *Parallel and Distributed Evolving Objects*

- Tutorial application

  → The Traveling Salesman Problem (TSP)

ParadisEO (1)

- A templates-based, ANSI-C++ compliant Metaheuristic Computation Framework
- GForge Project by INRIA Dolphin Team
- Paradigm Free (genetic algorithms, genetic programming, particle swarm optimization, local searches …)
- Hybrid, distributed and cooperative models

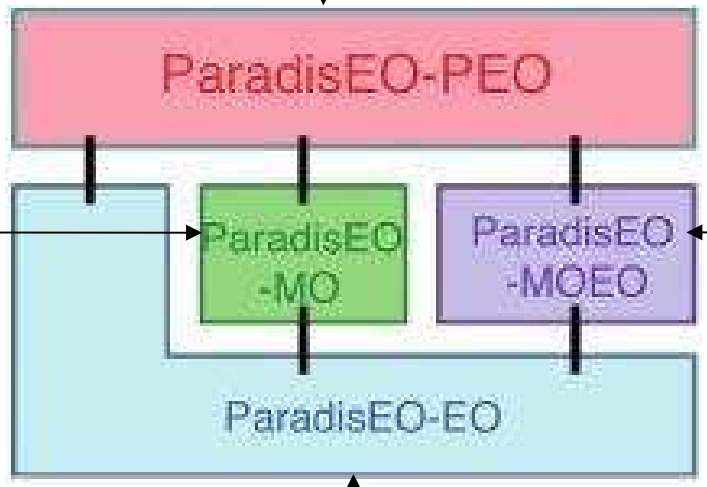
ParadisEO (2)

- Flexible / a considered problem
- Generic components (variation operators, selection, replacement, termination, particle behaviors …)
- Many services (visualization, managing command-line parameters, saving/restarting, …)

ParadisEO : Module-based architecture

Parallel and distributed metaheuristics

Tabu Search, Simulated Annealing, Hill-Climbing: single solution based metaheuristics

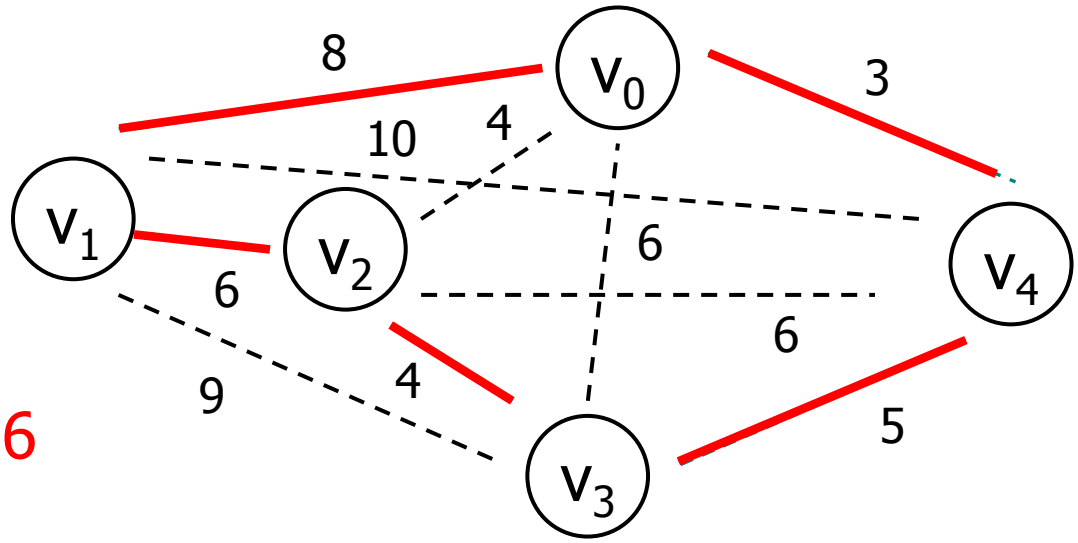


Multi-objective metaheuristics : NSGAII…, metrics…

Evolutionary computation, Swarm intelligence : population-based metaheuristics

Application to the Traveling Salesman Problem (TSP)

- "Given a collection of N cities and the distance between each pair of them, the **TSP** aims at finding the shortest route visiting all of the cities"
- Symmetric TSP: $\frac{(N-1)!}{2}$ candidate solutions
- Example



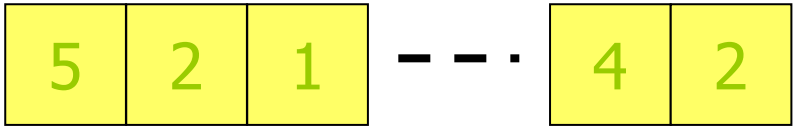
Length: 26

Designing a representation

- Representing an individual as a genotype
- Maybe several ways to do this. The representation must be relevant regards the tackled problem
- When choosing a representation, we have to bear in mind how the genotypes will be evaluated and how the variation operators will be used

Example. Discrete representation

- Representation of an individual can be using discrete values (binary, integer, or any other system with a discrete set of values).



Example. Real-valued representation

- Individuals are represented as a tuple of n real-valued numbers

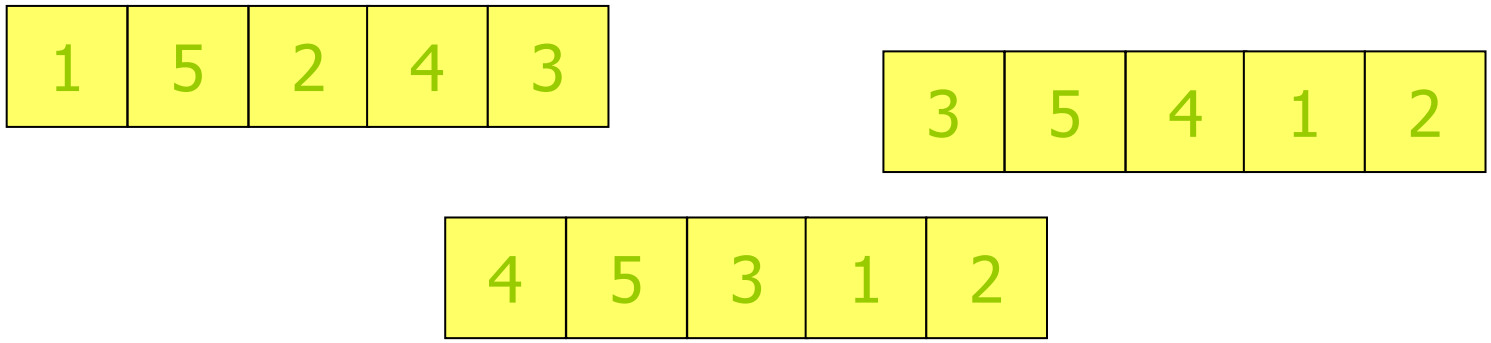
$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, x_i \in R$$

- The fitness function maps tuples of real numbers to a single real number

$$f : R^n \rightarrow R$$

Example. Order-based representation

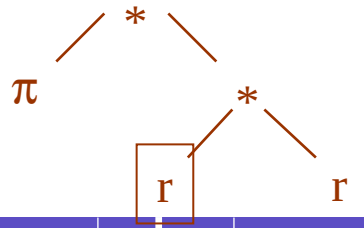
- Individuals are represented as permutations
- Used for ordering/sequencing problems (e.g. Flow-shop)
- Needs special operators to make sure the individuals stay valid permutations



Example. Tree-based representation

- Individuals in the population are trees
- Any S-expression can be drawn as a tree of functions and terminals
- These functions and terminals can be anything
  - Functions: sine, cosine, add, sub, and, If-Then-Else, Turn...
  - Terminals: X, Y, 0.456, true, false, p, Sensor0, …
  - Example: calculating the area of a circle

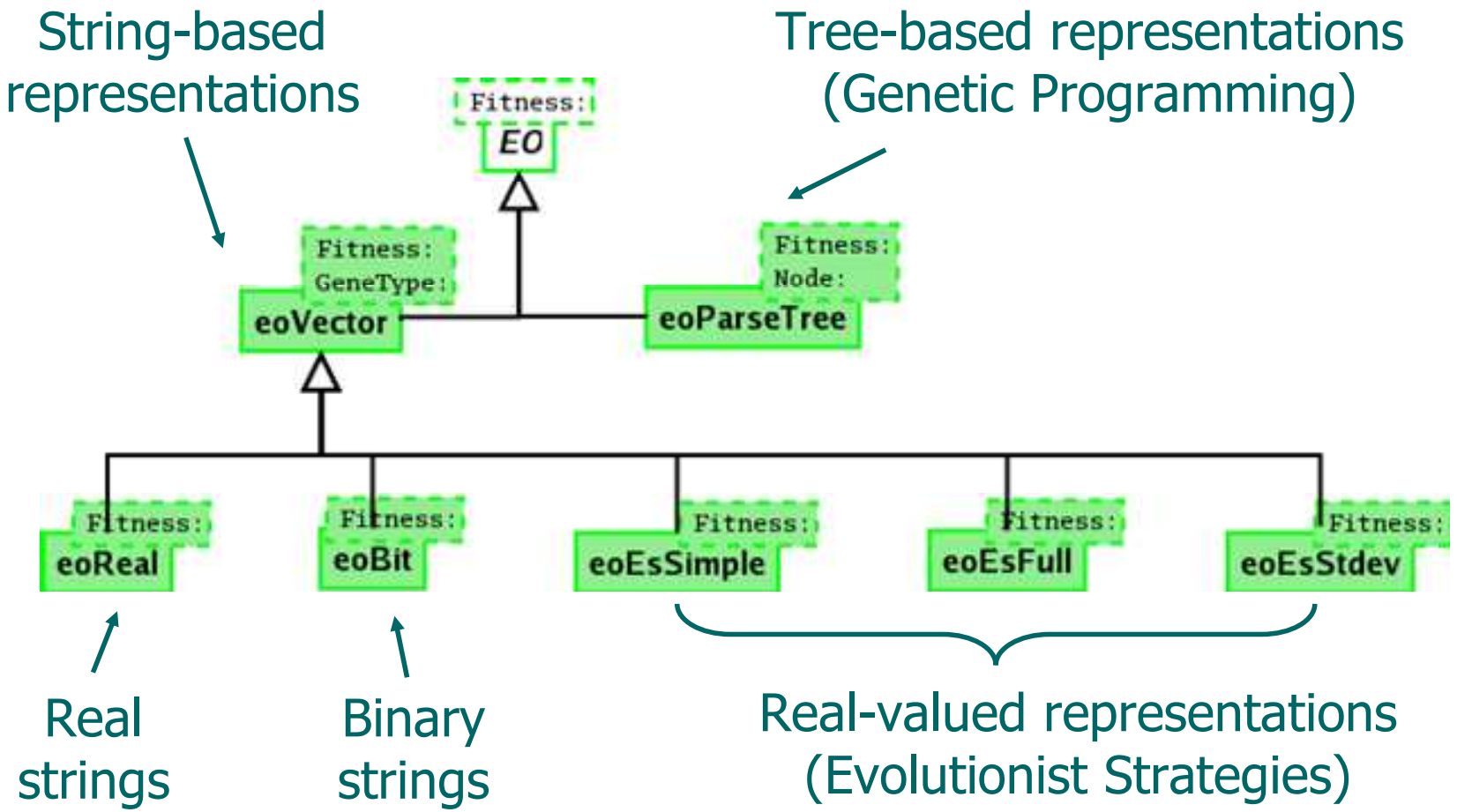
$$\pi * r^2$$



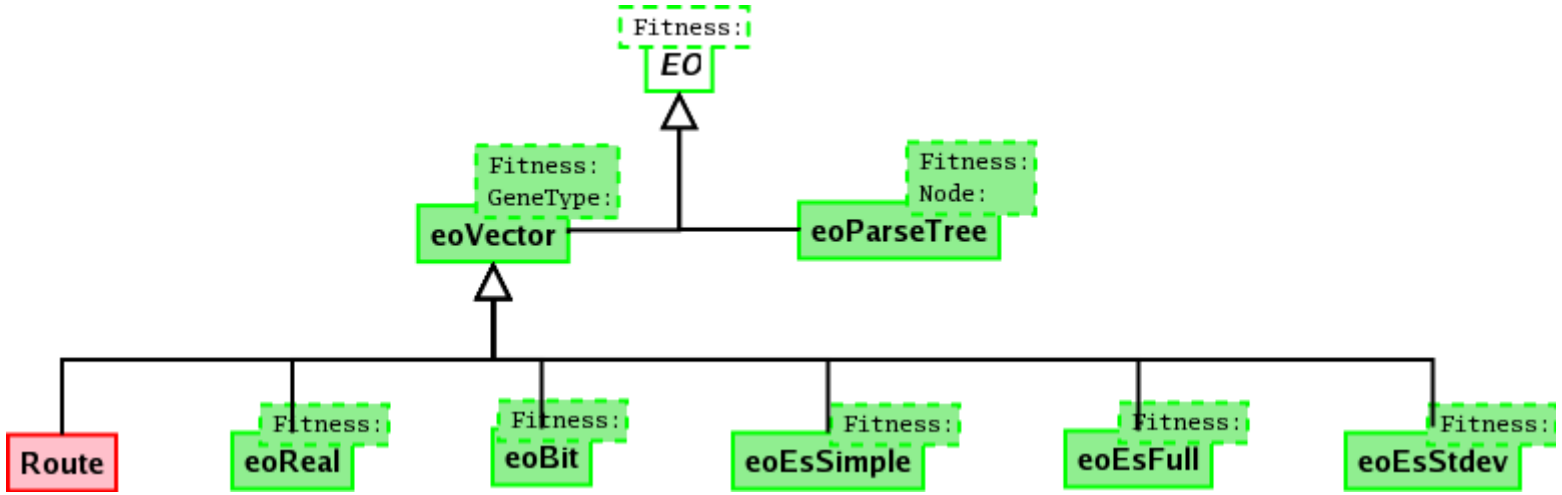
Other representations

- Matrices,
- Graphs,
- Rules (variable-length strings)
- Automata
- Programs, etc

Existent basic representations in ParadisEO.



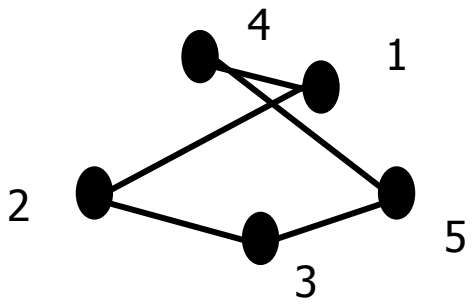
Application to the TSP



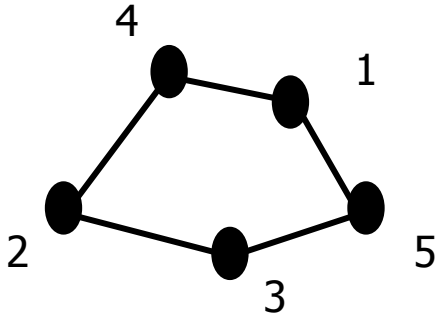
- Path encoding
  - Every node is assigned a number (e.g. from 0 up to *n* - 1) and solutions are represented by the ordered sequence of visited nodes.

Examples of path encoding

- A couple of paths

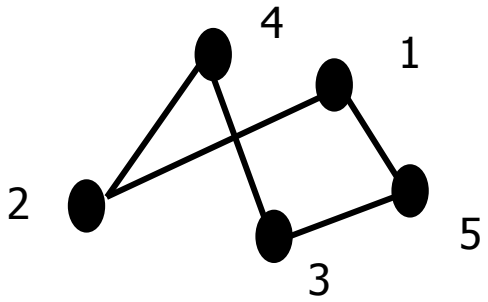


2	1	4	5	3



2	3	5	1	4

- Different solutions may encode a same path !



2	1	5	3	4	⇔	1	5	3	4	2	⇔	4	3	5	1	2

The evaluation of an individual

- This is by far the most costly step for real applications
- It might be a subroutine, a black-box simulator, or any external process (e.g. robot experiment)
- Fitness could be approximated

The evaluation of an individual

- Constraint handling - what if the solution breaks some constraint of the problem
  - penalize the fitness
  - specific evolutionary methods

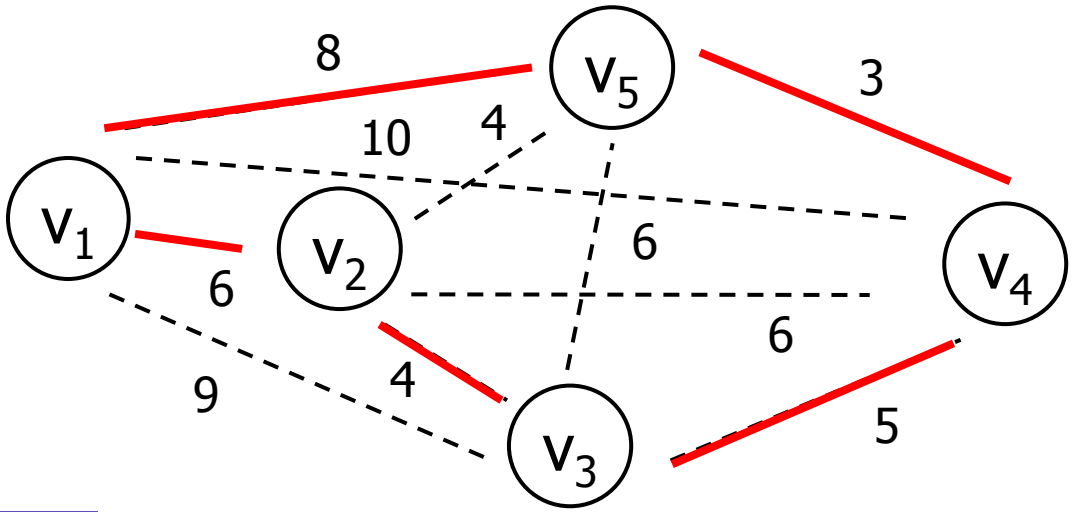
- Multi-objective evolutionary optimization gives a set of compromise solutions

Application to the TSP

- We aim at minimizing the total length of the path

$$\sum_{1 \le i \le N} dist(V_i, V_{(i+1) \bmod N})$$

1	2	3	4	5



	1	2	3	4	5
1	0	6	9	10	8
2	6	0	4	6	4
3	9	4	0	5	6
4	10	6	5	0	3
5	8	4	6	3	0

Variation operators

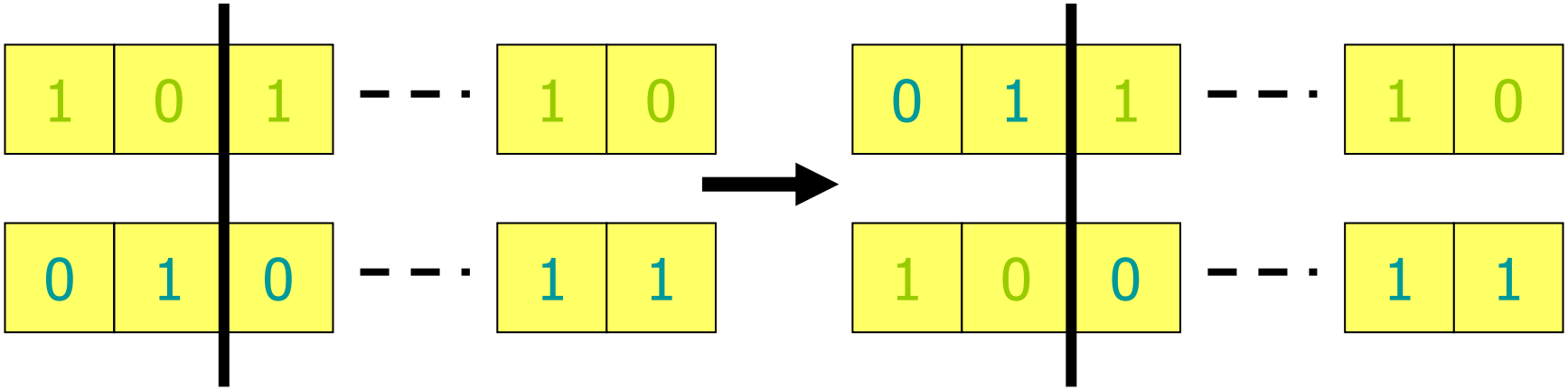
- Crossover operators
- Mutation operators

Recombination operators

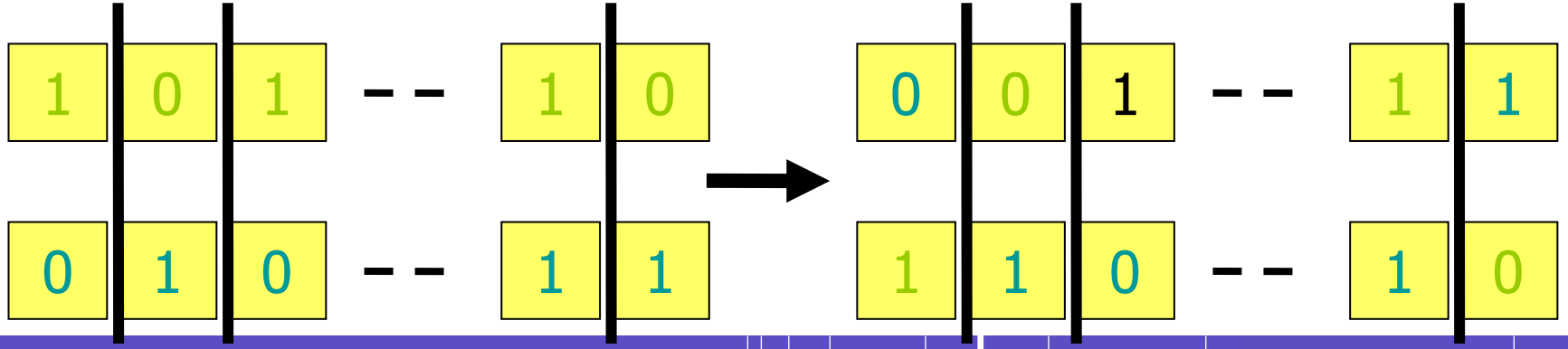
- We might have one or more recombination operators for our representation
- Some important points are
  - The child should inherit something from each parent. If this is not the case then the operator is a copy operator
  - The recombination operator should be designed in conjunction with the representation.
  - Recombination should produce valid chromosomes

Example. Recombination for discrete representation

- N-points crossover (e.g. 1 point)

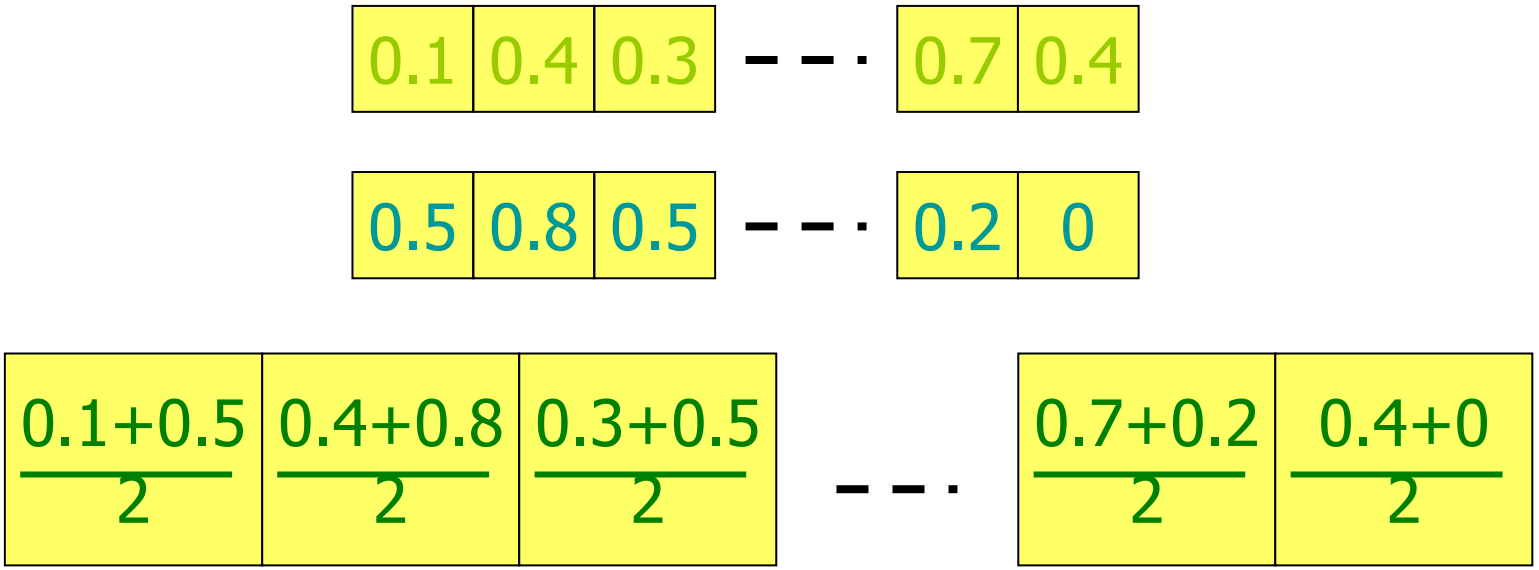


- Uniform crossover



Example. Recombination for real valued representation

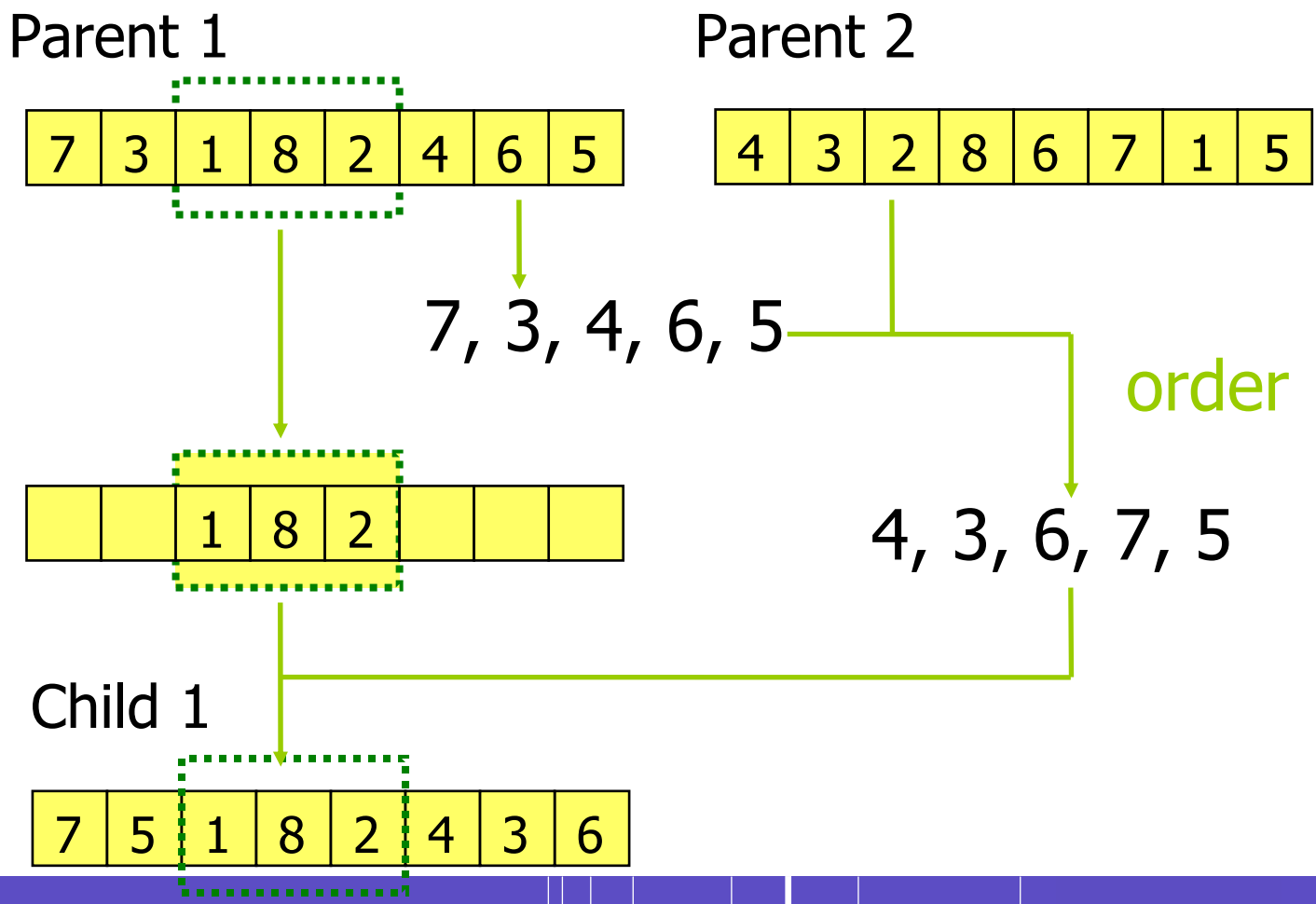
- Intermediate binary recombination (arithmetic crossover). Given two parents one child is created as follows



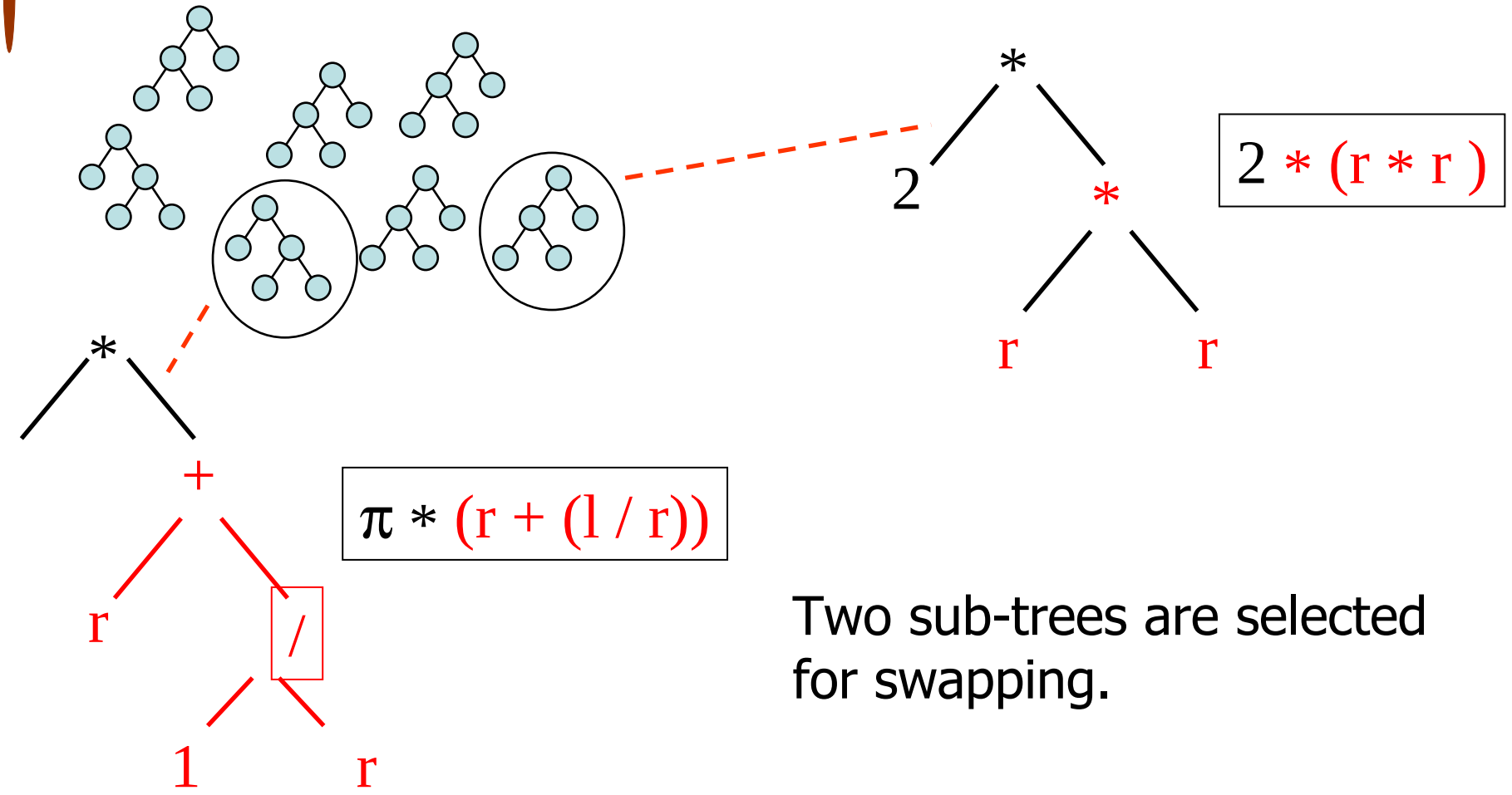
Recombination for order based representation

- Choose an arbitrary part from the first parent and copy this to the first child
- Copy the remaining genes that are not in the copied part to the first child
  - starting right from the cut point of the copied part
  - using the order of genes from the second parent
  - wrapping around at the end of the chromosome
- Repeat this process with the parent roles reversed

Example. Recombination for order based representation (Order 1)

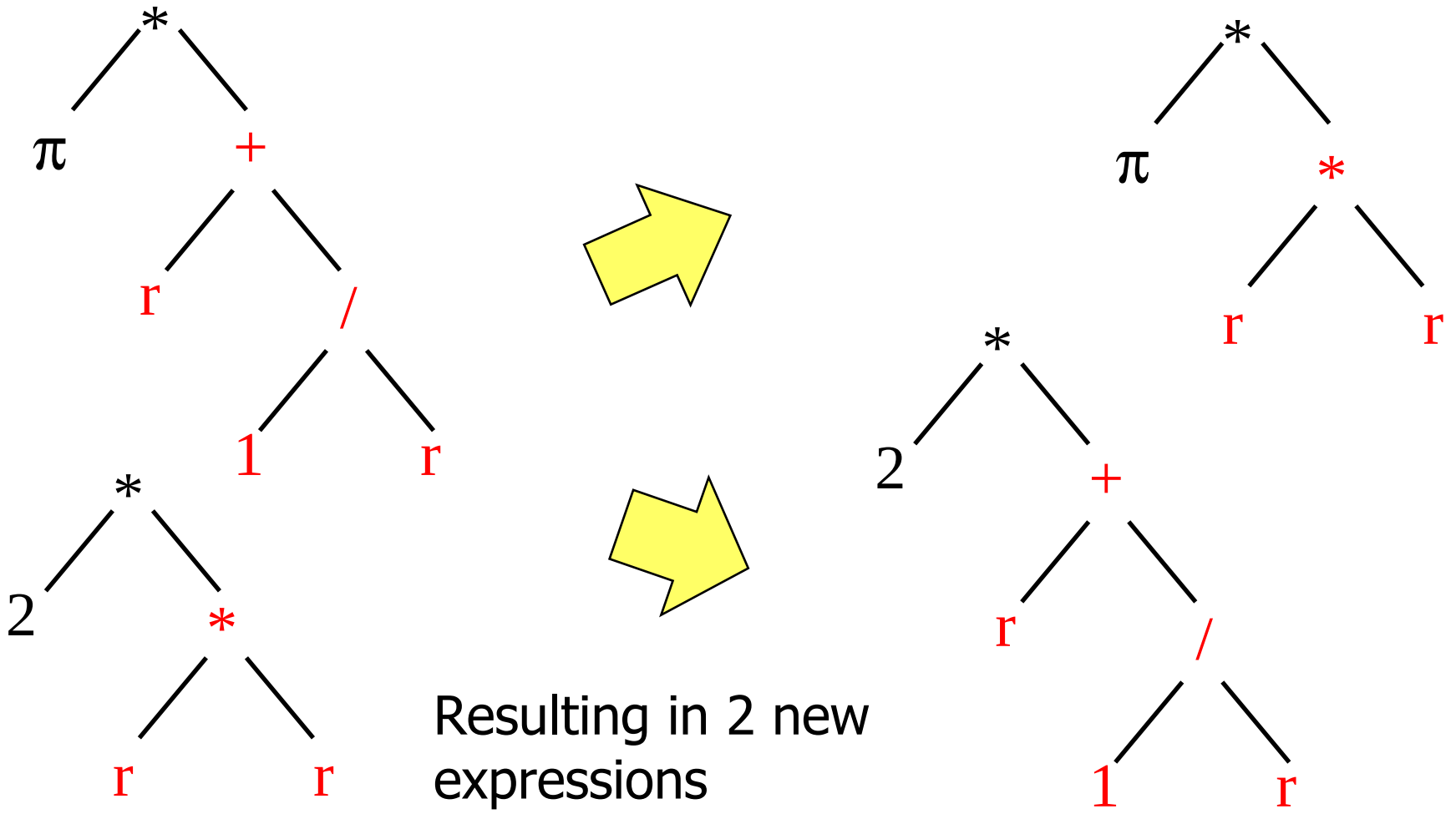


Recombination for tree based representation



Two sub-trees are selected for swapping.

Recombination for tree based representation

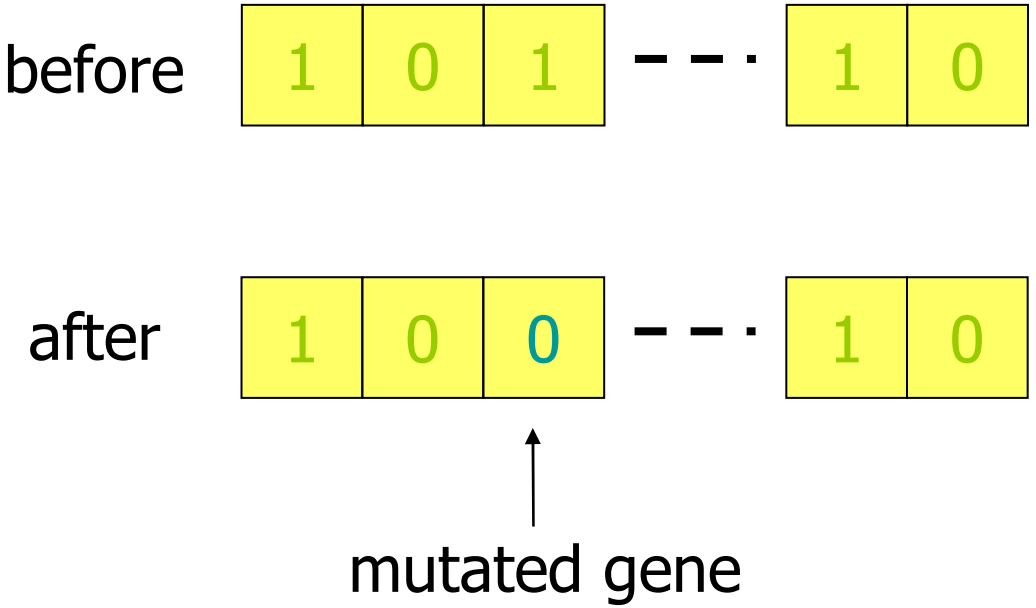


Resulting in 2 new expressions

Mutation operators

- We might have one or more mutation operators for our representation
- Some important points are
  - At least one mutation operator should allow every part of the search space to be reached
  - The size of mutation is important and should be controllable
  - Mutation should produce valid chromosomes

Example. Mutation for discrete representation

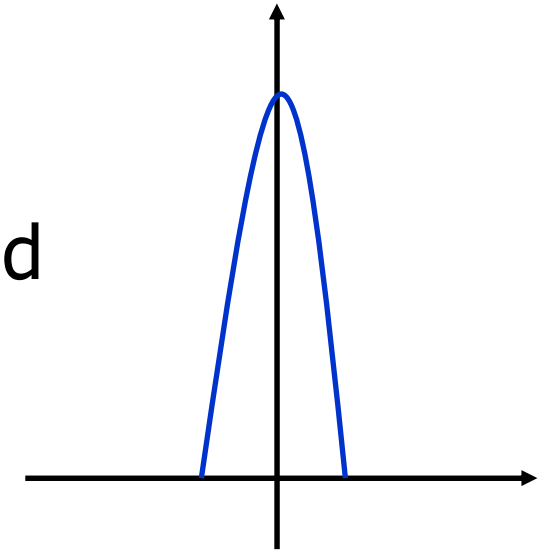


- Mutation usually happens with probability p_m for each gene
- It could affect only one gene too

Example. Mutation for real valued representation

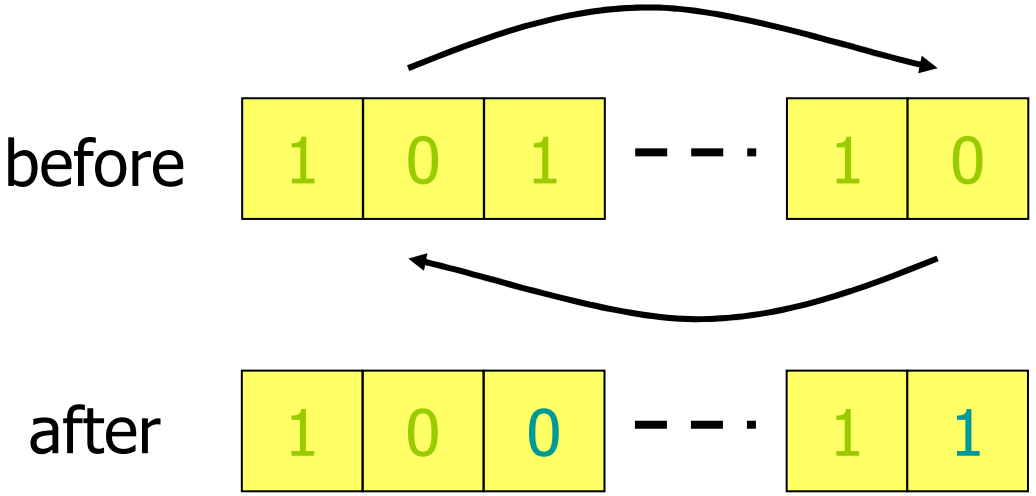
- Perturb values by adding some random noise
- Often, a Gaussian/normal distribution $N(0,\sigma)$ is used, where
  - 0 is the mean value
  - σ is the standard deviation and

$$x'i = xi + N(0,\sigma i)$$



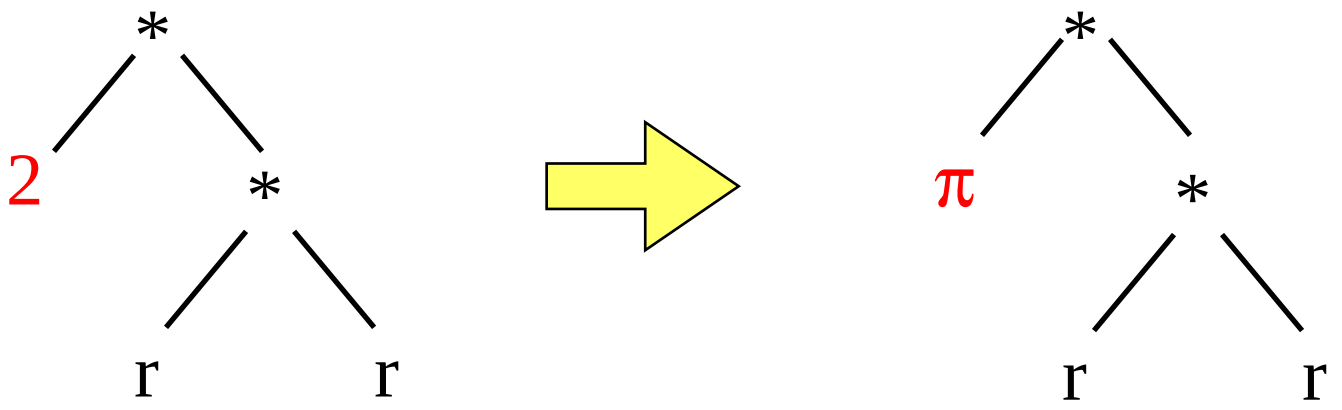
Example. Mutation for order based representation

- Randomly select two different genes and swap them

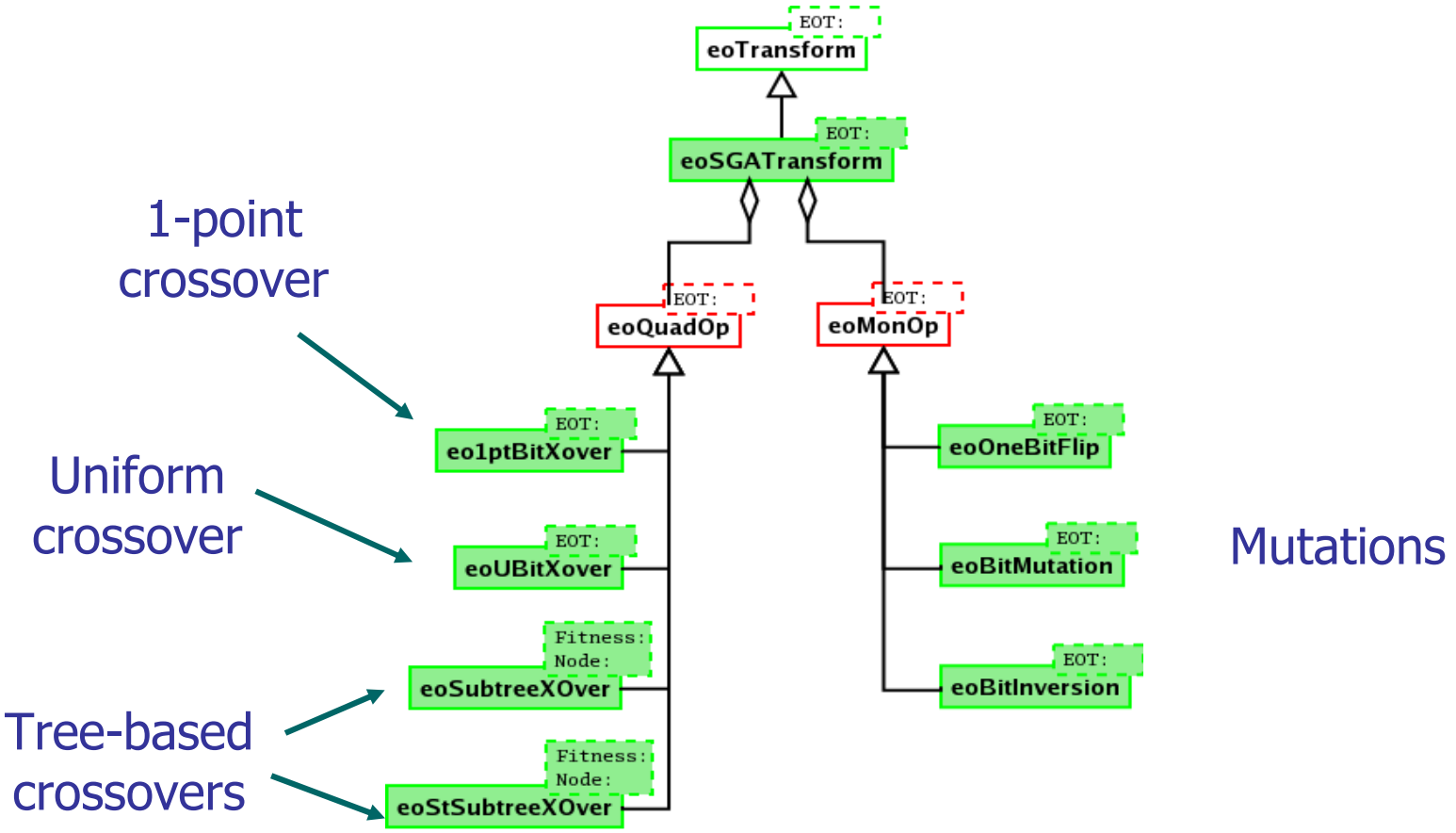


Example. Mutation for tree-based representation

- Single point mutation selects one node and replaces it with a similar one



Core classes

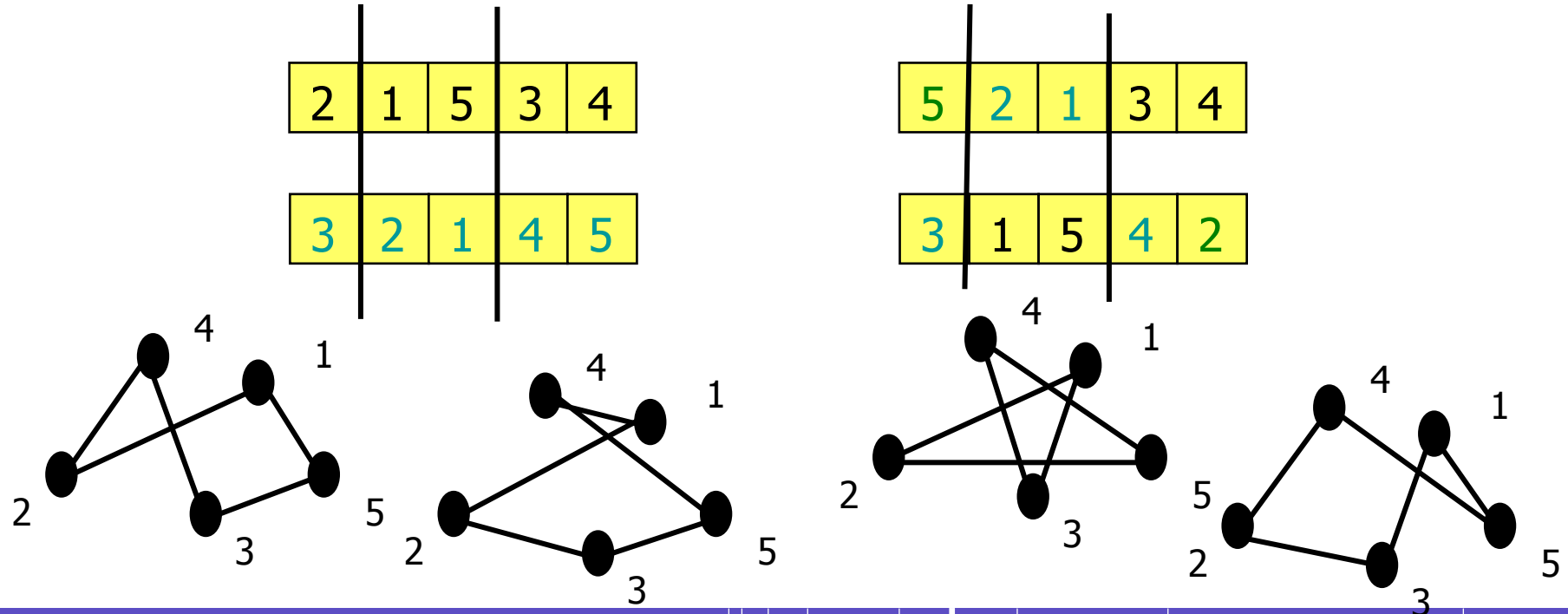


Application to the TSP

- We remind a path is encoded an ordered sequence of vertices
- Some of the most significant operators
  - Recombination
    - Partially Mapped Crossover
    - Order Crossover
    - Edge crossover
  - Mutation
    - Two-Opt mutation
    - City-Swap mutation

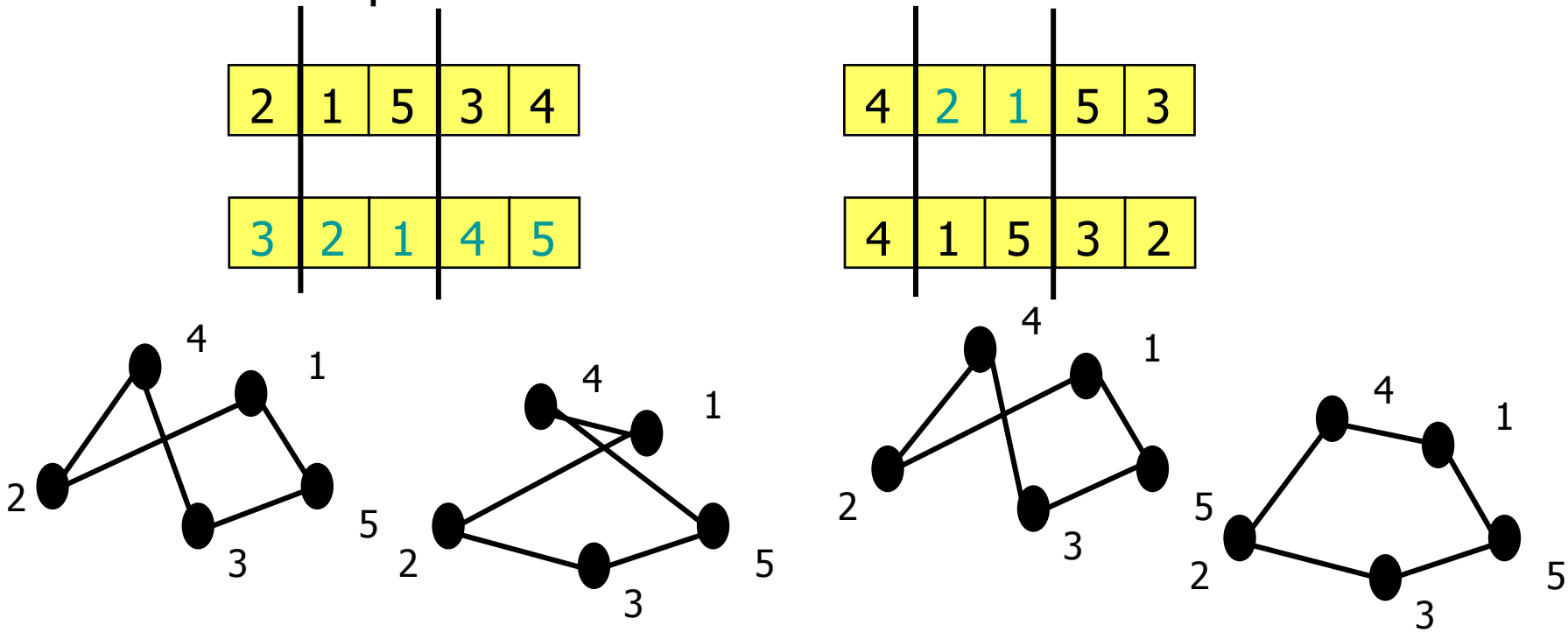
Recombination (1)

- Partially-Mapped Crossover or PMX
  - Designed to preserve many absolute positions from both parents



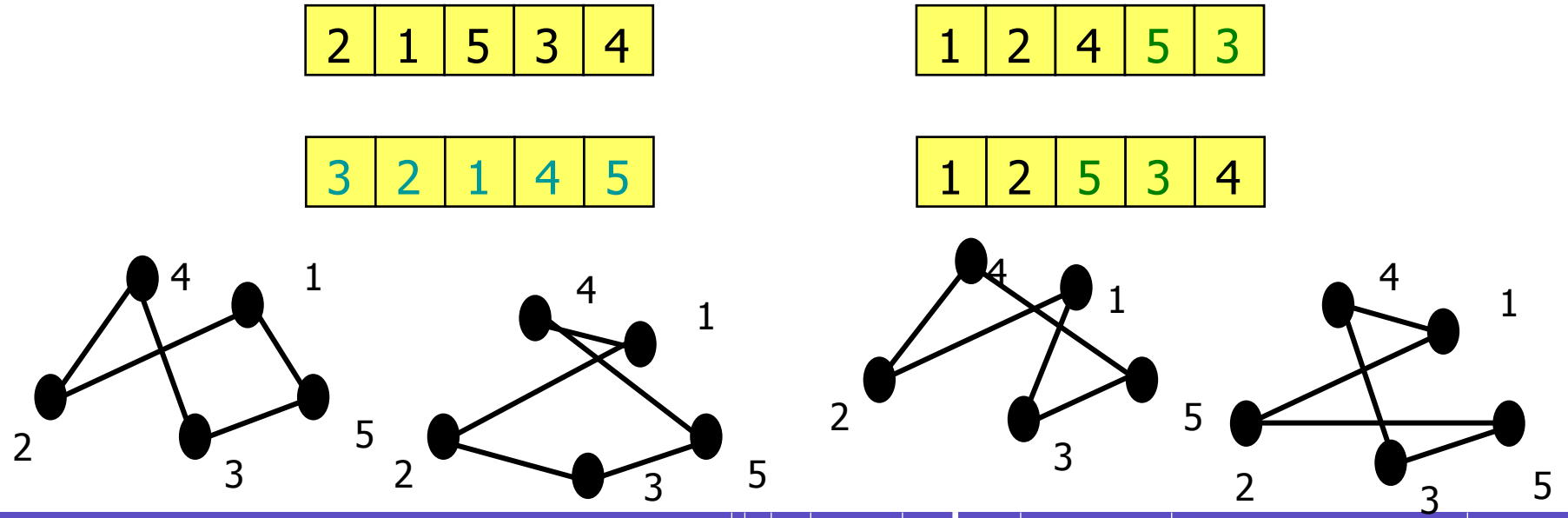
Recombination (2)

- Order Crossover (OX)
  - Designed to preserve the relative ordering from both parents



Recombination (3)

- Edge Crossover (EX)
  - Designed to preserve common edges from both parents (path encoding should be completed with an auxiliary structure called edge-map

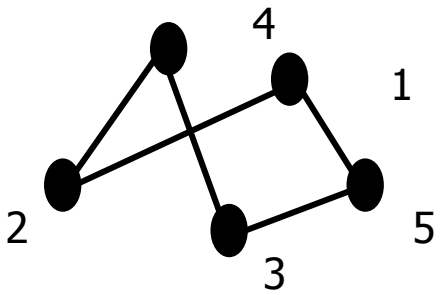


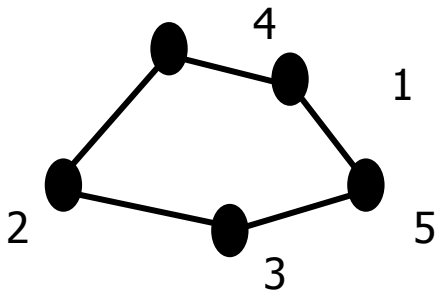
Mutation

- City Swap. The values contained in two random positions are exchanged, thus introducing four new edges in the string

2	1	5	3	4

2	3	5	1	4

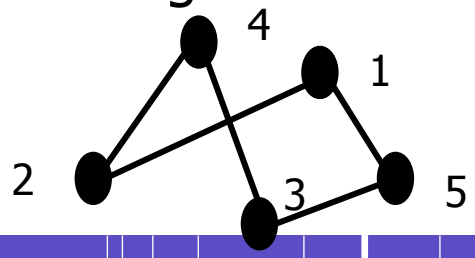


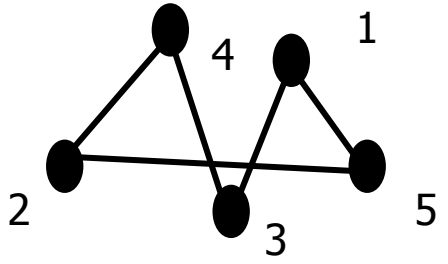


- Two-Opt. Two random points within the string are selected and the segment between them is inverted. This operator put in two new edges in the tour

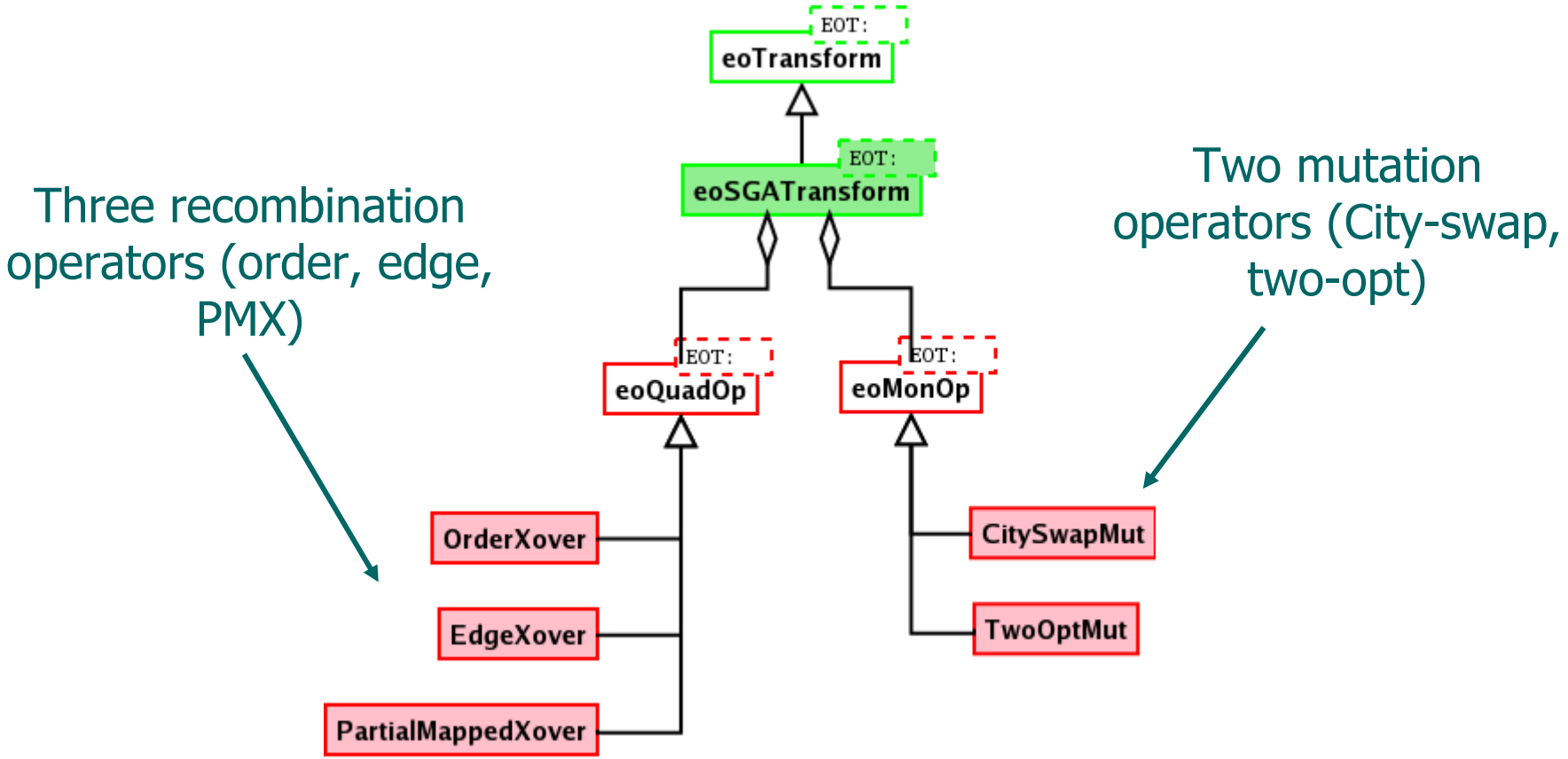
2	1	5	3	4

2	5	1	3	4





Core classes



Implementation of variation operators. A few examples

```
class OrderXover : public eoQuadOp <Route> {

public :

  bool operator () (Route & __route1, Route & __route2) ;
} ;
```

Header class of Order crossover

```
class CitySwap : public eoMonOp <Route> {

public :

  bool operator () (Route & __route) ;
} ;
```

Header class of City-swap mutation

The selection strategy

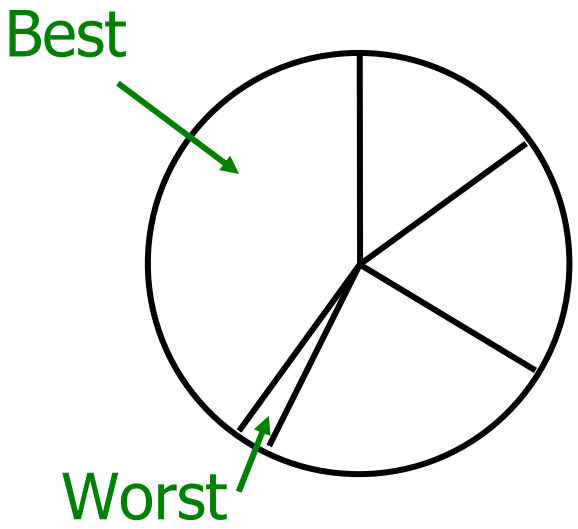
- The main rule: “The better is an individual, the higher is its chance of being parent”.
- Such a selection pressure will drive the population forward
- Worst individuals shouldn’t be discarded but have some chance to be selected. This may lead to useful genetic material

The most common selection strategies

- Proportional selection (roulette wheel)
- Fitness scaling
- Tournament

Fitness proportionate selection

- Let f_i the fitness of the individual p_i in the population P. Its chance to be selected is $\frac{f_i}{\sum_{p_j \in P} f_j}$
- Better (fitter) individuals have more space, more chance to be chosen
- Disadvantages
  - Premature convergence because outstanding individuals take over the entire population very quickly
  - Low selection pressure when fitness values are near each other
  - Behaves differently on transposed versions of the same function

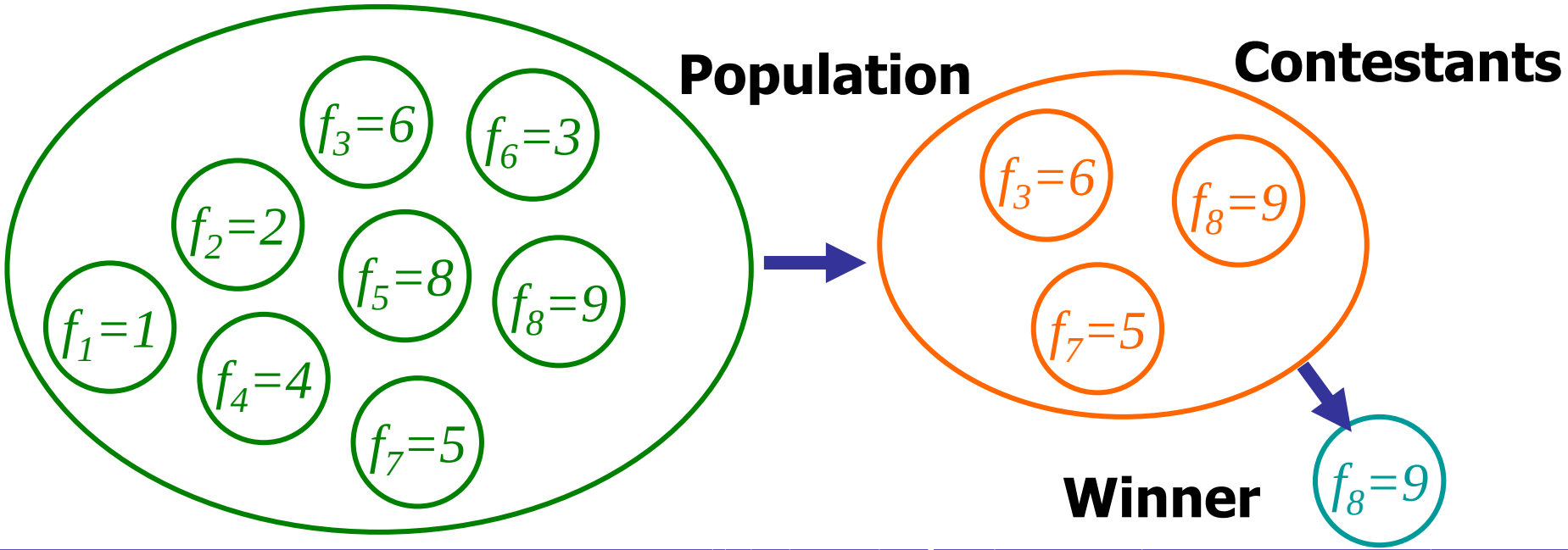


Fitness scaling

- Start with the raw fitness function f_i
- Standardise to ensure
  - Higher fitness is better fitness.
  - Optimal fitness equals to 1.
- Adjust to ensure
  - Fitness ranges from 0 to 1.
- Normalise to ensure
  - The sum of the fitness values equals to 1

The tournament

- Randomly select ***k*** individuals (***k*** is called the size of the tournament)
- Take the best of them



The rank based selection

- Individuals are sorted on their fitness value from best to worse. The place in this sorted list is called rank
- Instead of using the fitness value of an individual, the rank is used by a function to select individuals from this sorted list. The function is biased towards individuals with a high rank (i.e. good fitness)

Example

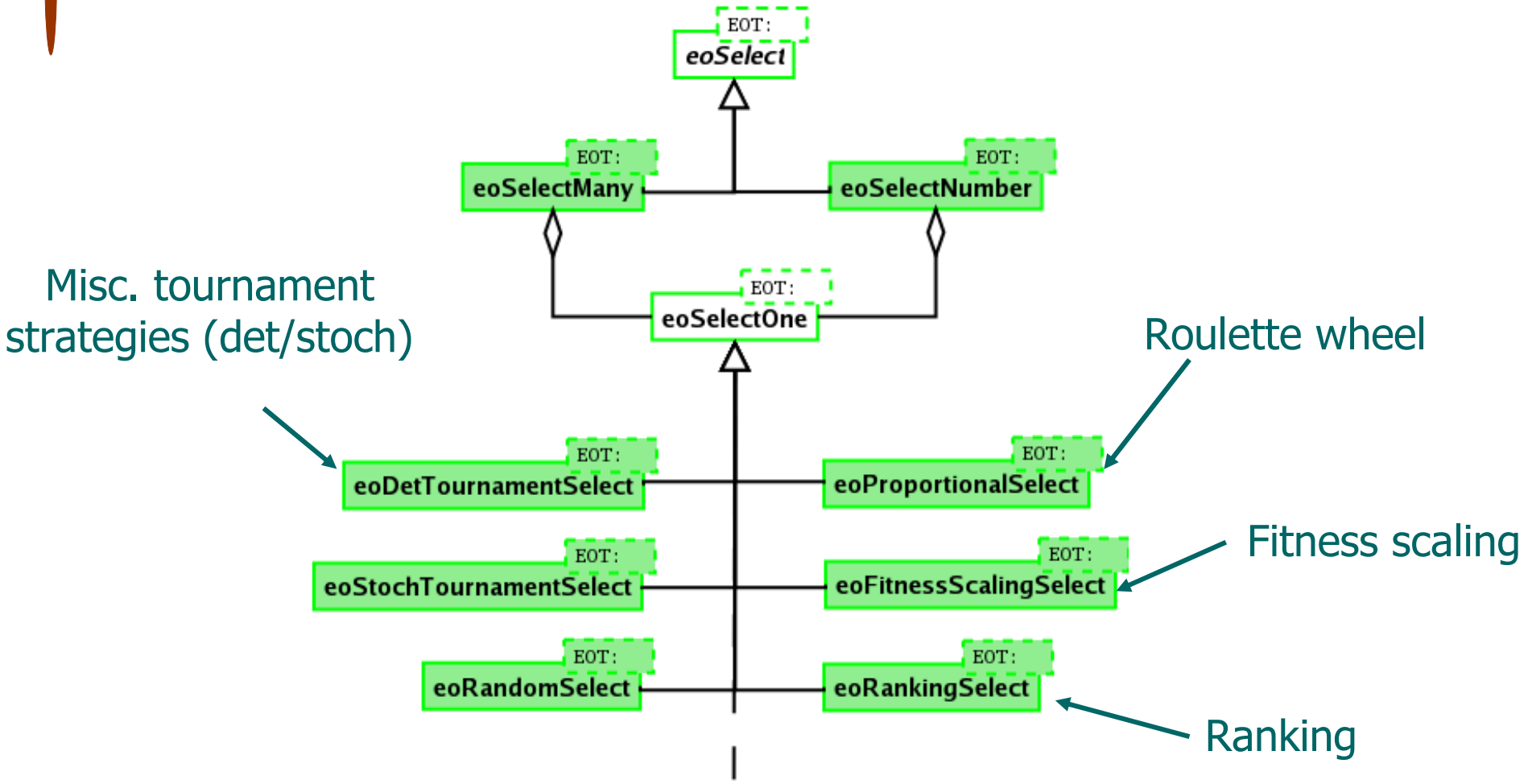
- Fitness. $f_1 = 5$, $f_2 = 2$, $f_3 = 19$
- Associated rank. $r_1=2$, $r_2=3$ $r_3=1$

$$h_i = \min_{P_j \in P}(f_j) + (\max_{P_j \in P}(f_j) - \min_{P_j \in P}(f_j)) \times \frac{(r_i - 1)}{n-1}$$

- Function $h_1 = 3$, $h_2 = 5$, $h_3 = 1$
- Proportion on the roulette wheel

$s_1 = 11.1\%$, $s_2 = 33.3\%$, $s_3 = 55.6\%$

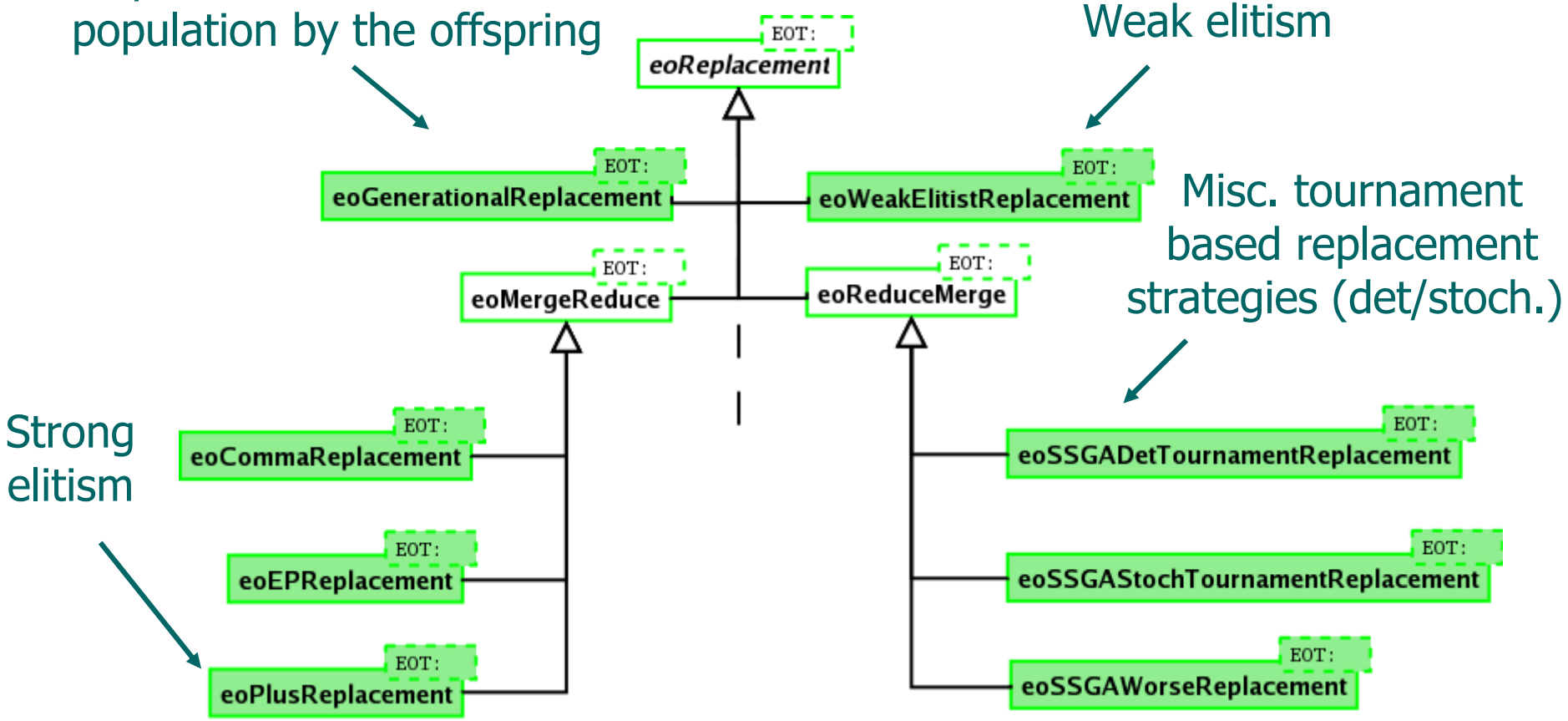
Core classes



The replacement strategy

- Selection pressure is also affected in the replacement step (survivors of both population and offspring)
- Stochastic methods/deterministic strategies
- Elitism (i.e. should fitness ever improve ?)
  $\rightarrow$ Reintroduce in the new generation the best solution found during the search

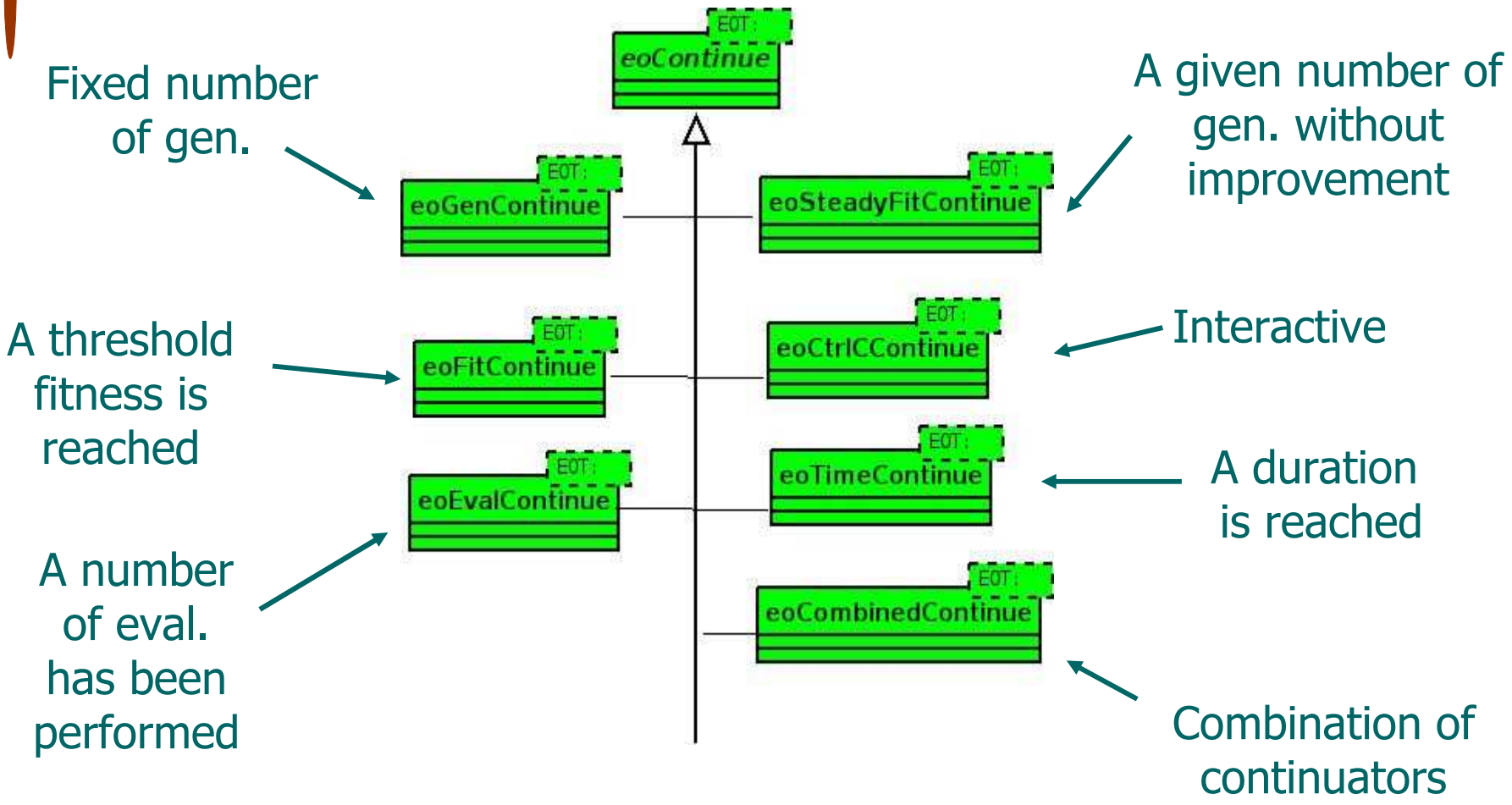
Core classes



The continuation strategy

- The optimum is reached !
- Limit on CPU resources: Maximum number of fitness evaluations
- A given number of generations without improvement, etc ...

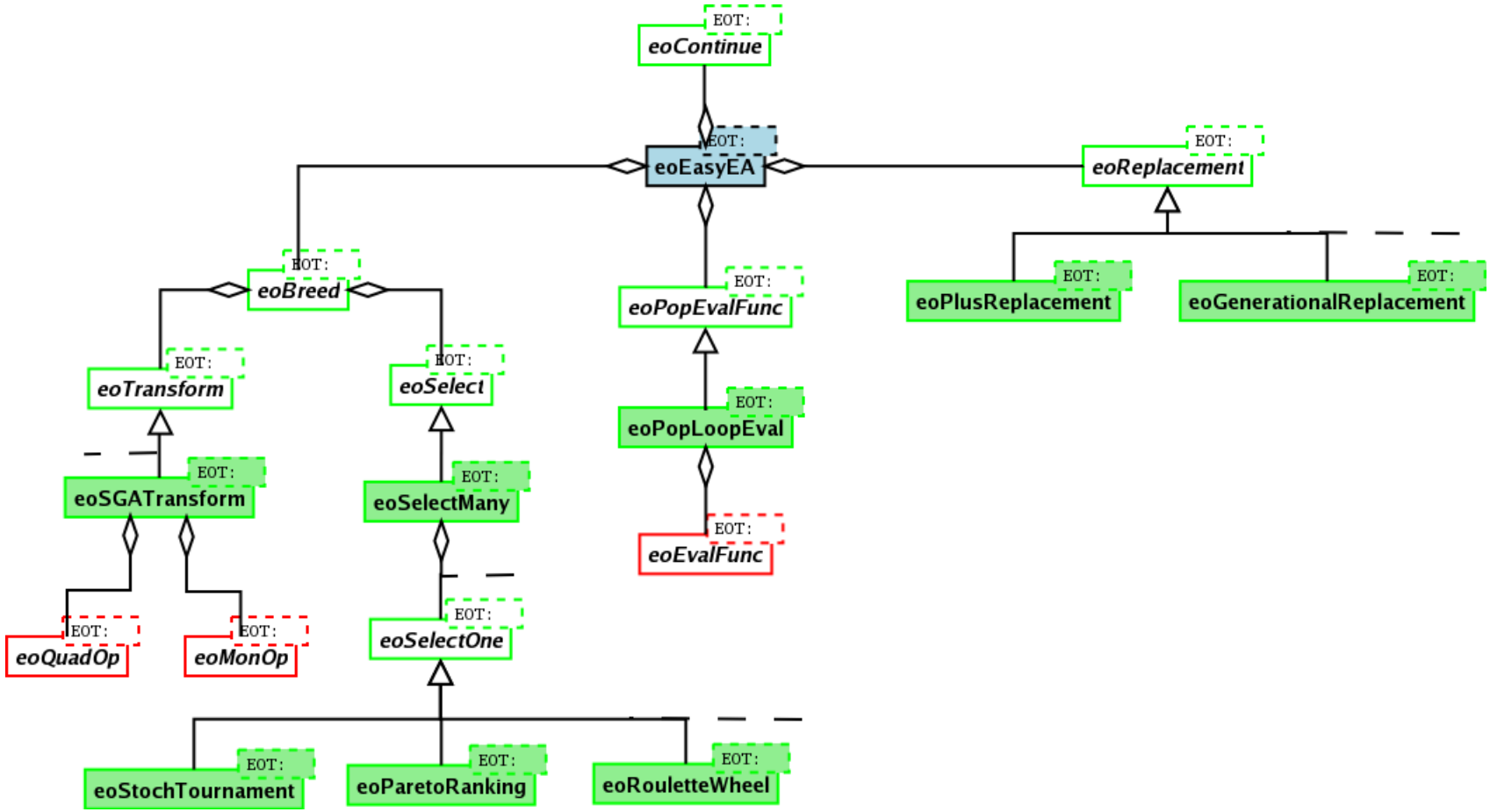
Core classes



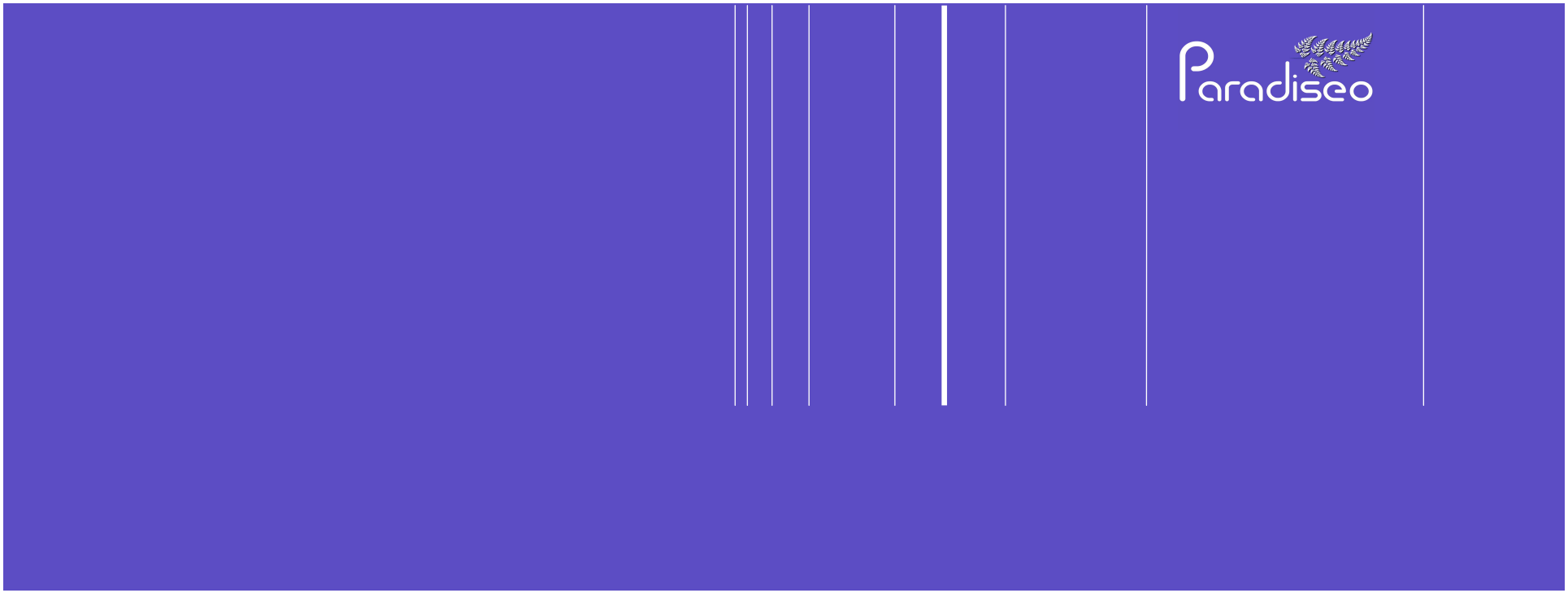
Implementation of a Genetic Algorithm

```
RouteInit route_init; /* Its builds random routes */
RouteEval full_route_eval; /* Full route evaluator */

eoPop <Route> ox_pop (POP_SIZE, route_init);  /* Population */
eoGenContinue <Route> cont (NUM_GEN); /* A fixed number of iterations */

OrderXover order_cross; /* Recombination */
CitySwap city_swap_mut;  /* Mutation */

eoStochTournamentSelect <Route> ox_select_one; /* Stoch. Tournament selection */
eoSelectNumber <Route> ox_select (select_one, POP_SIZE);
/* Standard SGA Transformation */
eoSGATransform <Route> ox_transform (cross, CROSS_RATE, city_swap_mut, MUT_RATE);
eoSelectTransform <Route> ox_breed (select, transform); /* Breeding */
eoEPReplacement <Route> ox_replace (2); /* Tournament repl. */

eoEA <Route> ea (cont, full_route_eval, breed, replace);

ea (pop);   /* Application to the given population */
```

Illustration. Core classes of the Evolutionary Algorithm



Particle Swarm Optimization



The main steps to build a particle swarm optimization algorithm

1. Design a representation
2. Decide how to initialize a population (=swarm)
3. Design a way of evaluating an individual
5. Design suitable velocity operator
6. Decide the flight operator
7. Decide how to manage the population
8. Decide the "best updating" strategy
9. Decide the continuation criterion

Framework and tutorial application

- Framework dedicated to metaheuristics

  →  *Parallel and Distributed Evolving Objects*

- Tutorial application

  → Norm optimization problem (continuous)

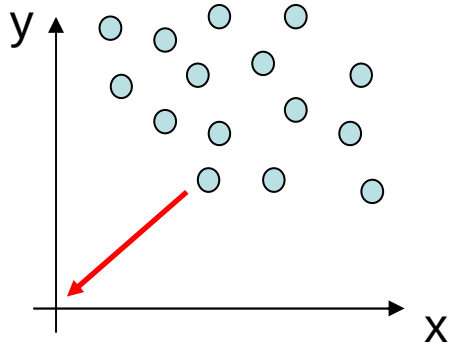
Application to the Norm Optimization Problem (NOP)

- $f : R^n \rightarrow R$ $\qquad X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, x_i \in R$

$$f(x_1, x_2, \ldots, x_n) = \sqrt{\sum_{i=0}^{N} x_i^2}$$

- Example :

$$f(x_1, x_2) = \sqrt{x_1^2 + x_2^2}$$



Designing a representation

- Representing an individual as a position
- Maybe several ways to do this. The representation must be relevant regards the tackled problem
- When choosing a representation, we have to bear in mind how the positions will be evaluated and how the flight operators will be used

Real-valued representation

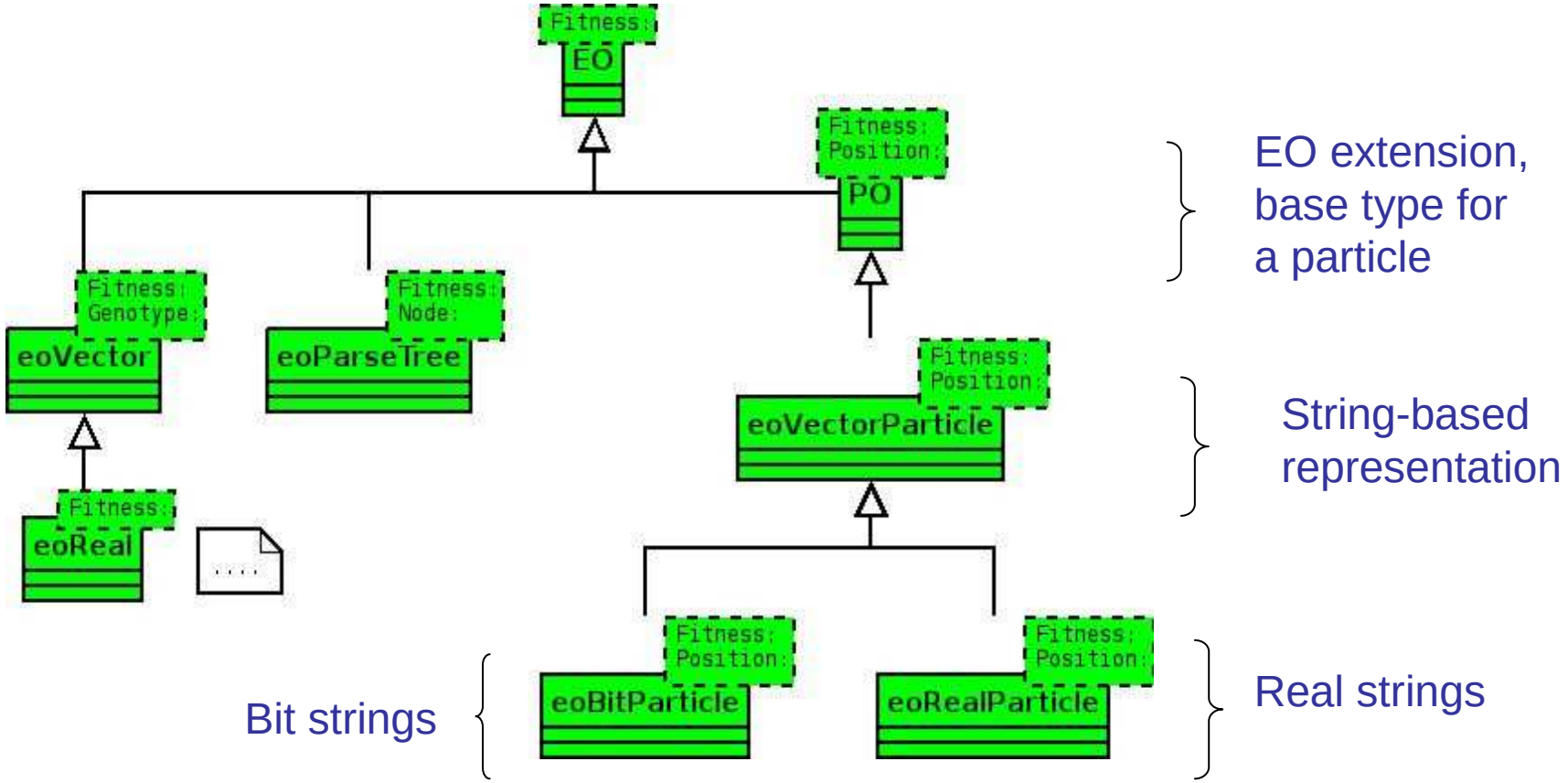
- Individuals are represented as a tuple of n real-valued numbers

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, x_i \in R$$

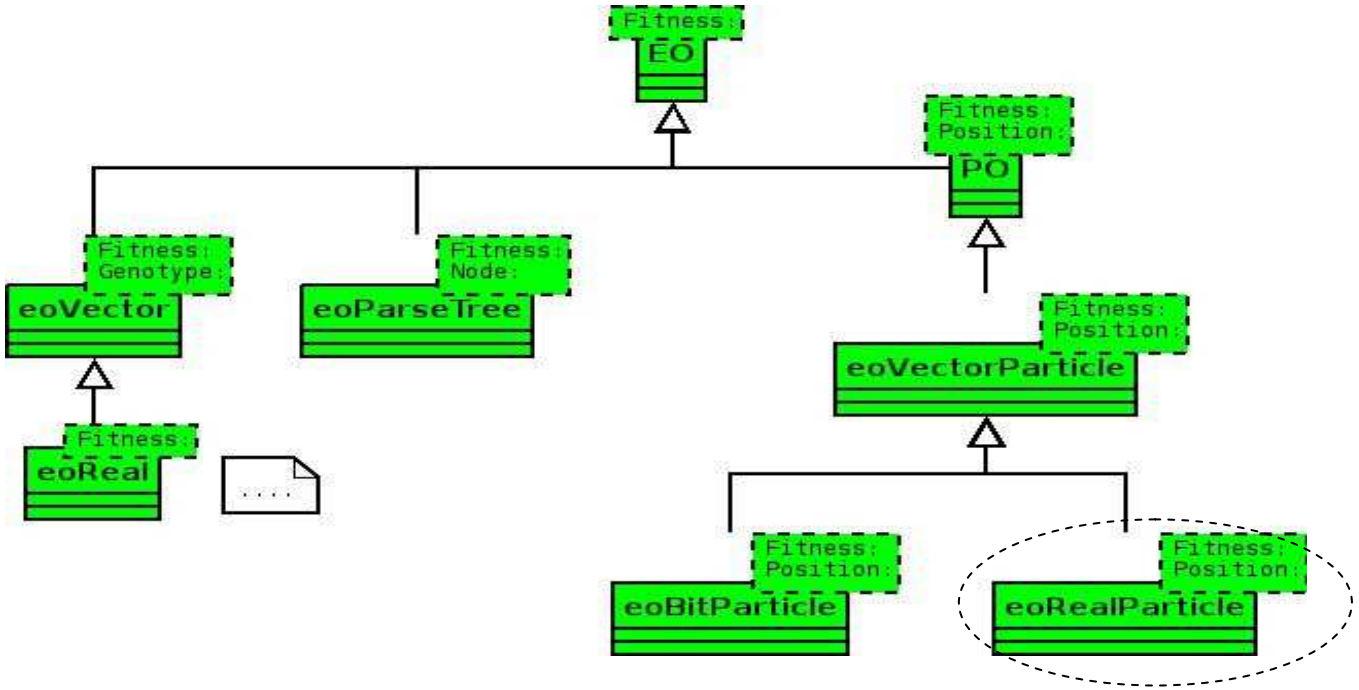
- The fitness function maps tuples of real numbers to a single real number

$$f : R^n \rightarrow R$$

Existent basic particle representations in ParadisEO



Application to the NOP

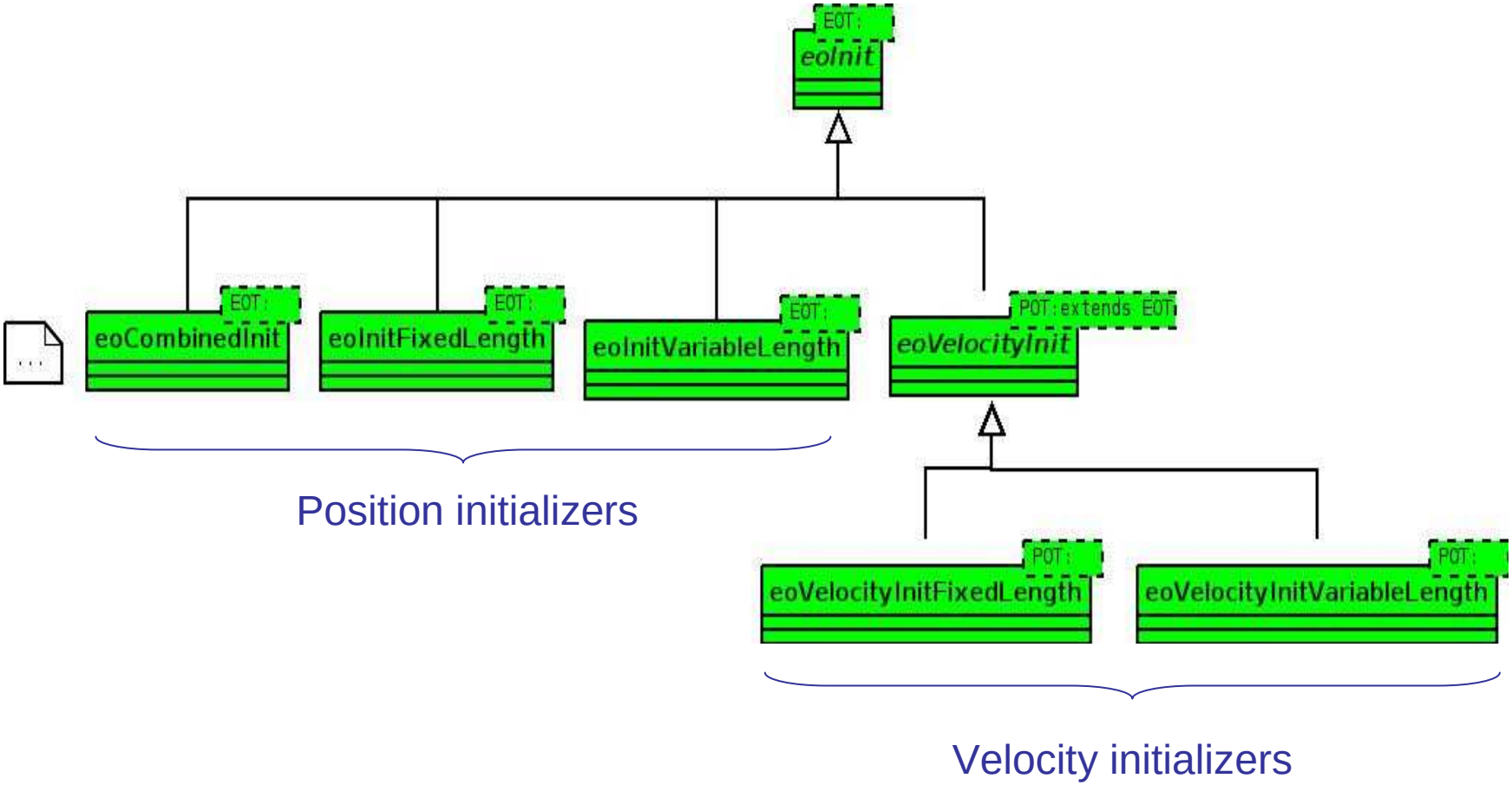


- Particle encoding:
  - Each component of the position is a real
  - Each component of the best position is a real
  - Each component of the velocity is a real

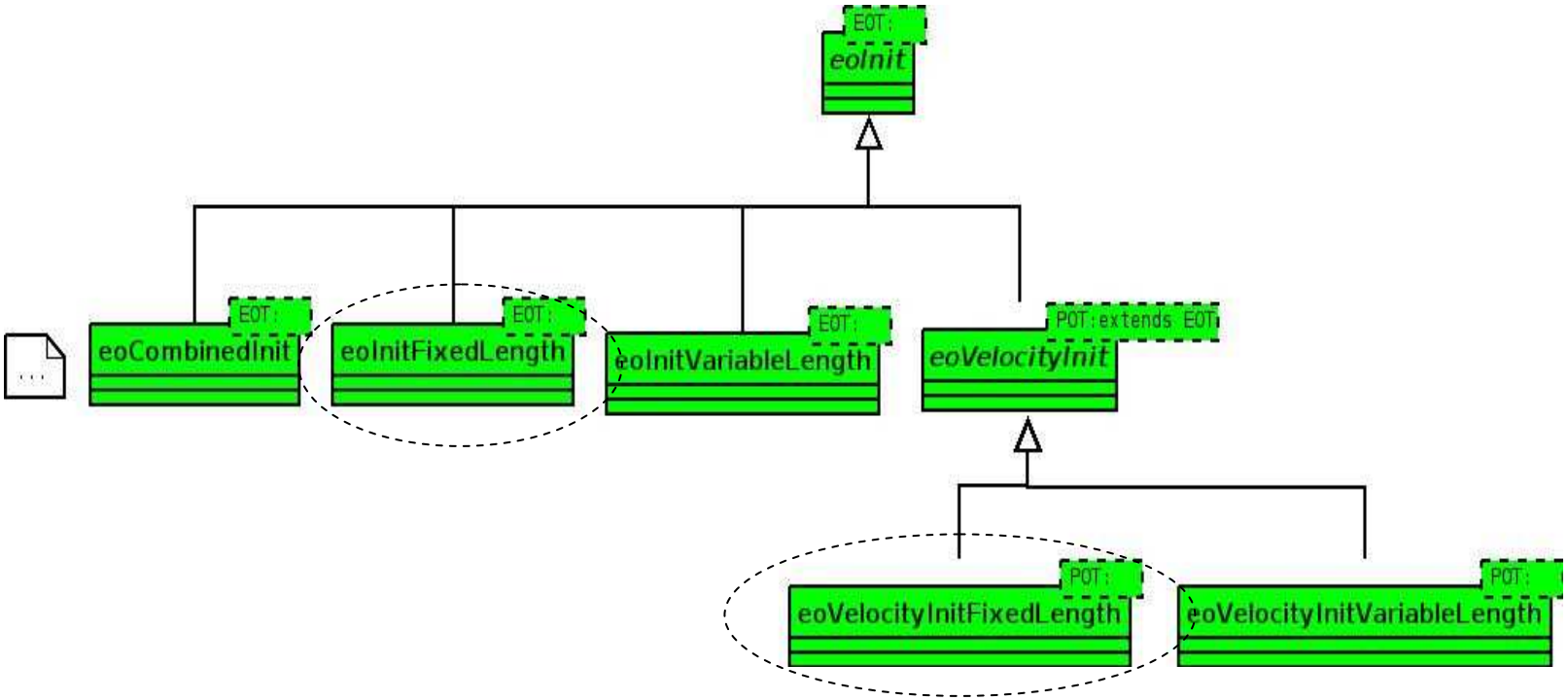
Initialization of the swarm

- Initialize
  - Positions Standard
  - Velocities
  - Best positions of each particle
  - The global best
- Standard strategies
  - Position + Velocities: Random
  - Initial global best = initial best particle

Existent swarm initializers in ParadisEO

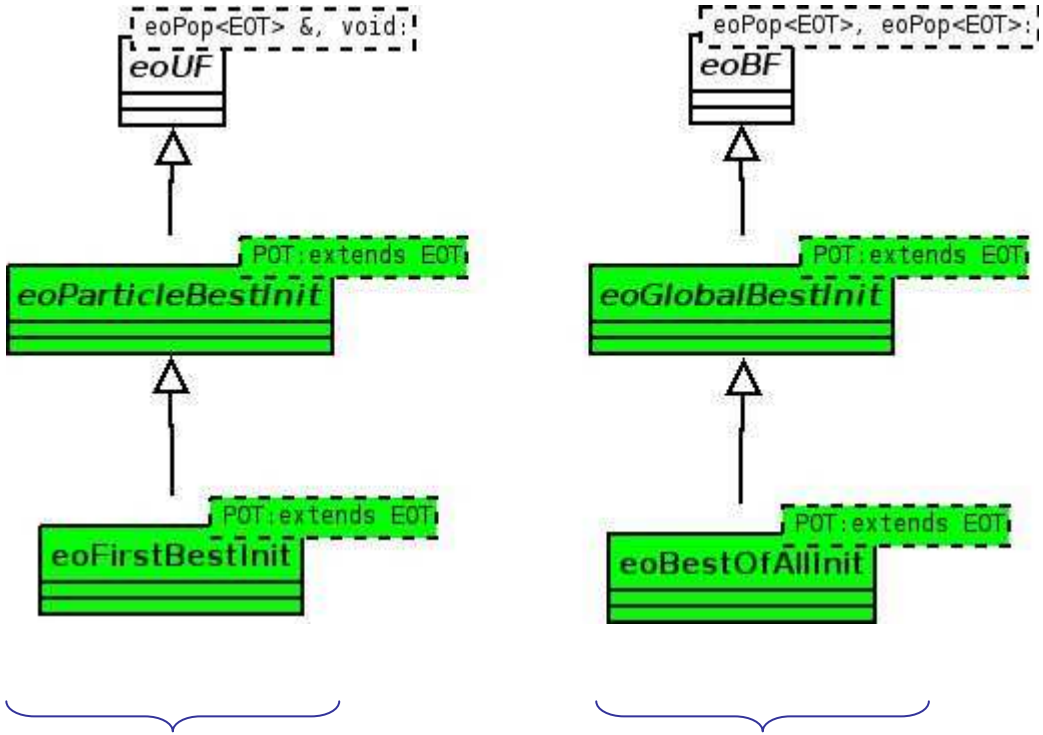


Application to the NOP



- Generate the initial positions and velocities at random between bounds

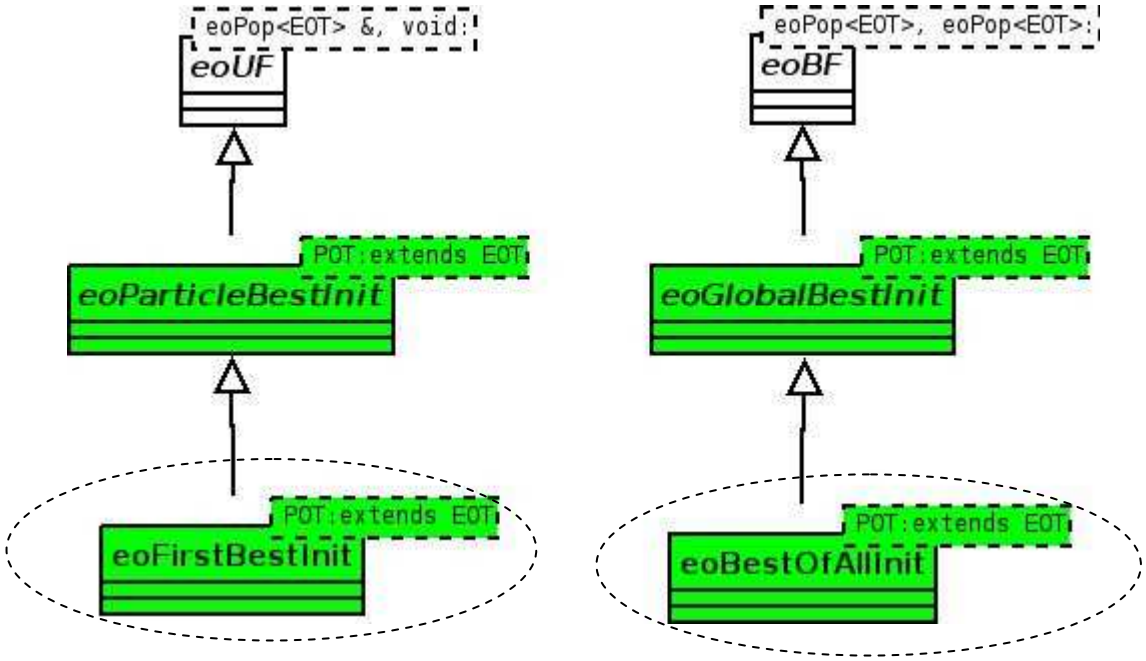
Existent swarm initializers in ParadisEO (2)



Particle best initializers

Global best initializers

Application to the NOP



- The first position is the initial best
- The initial global best is the initial best of the swarm

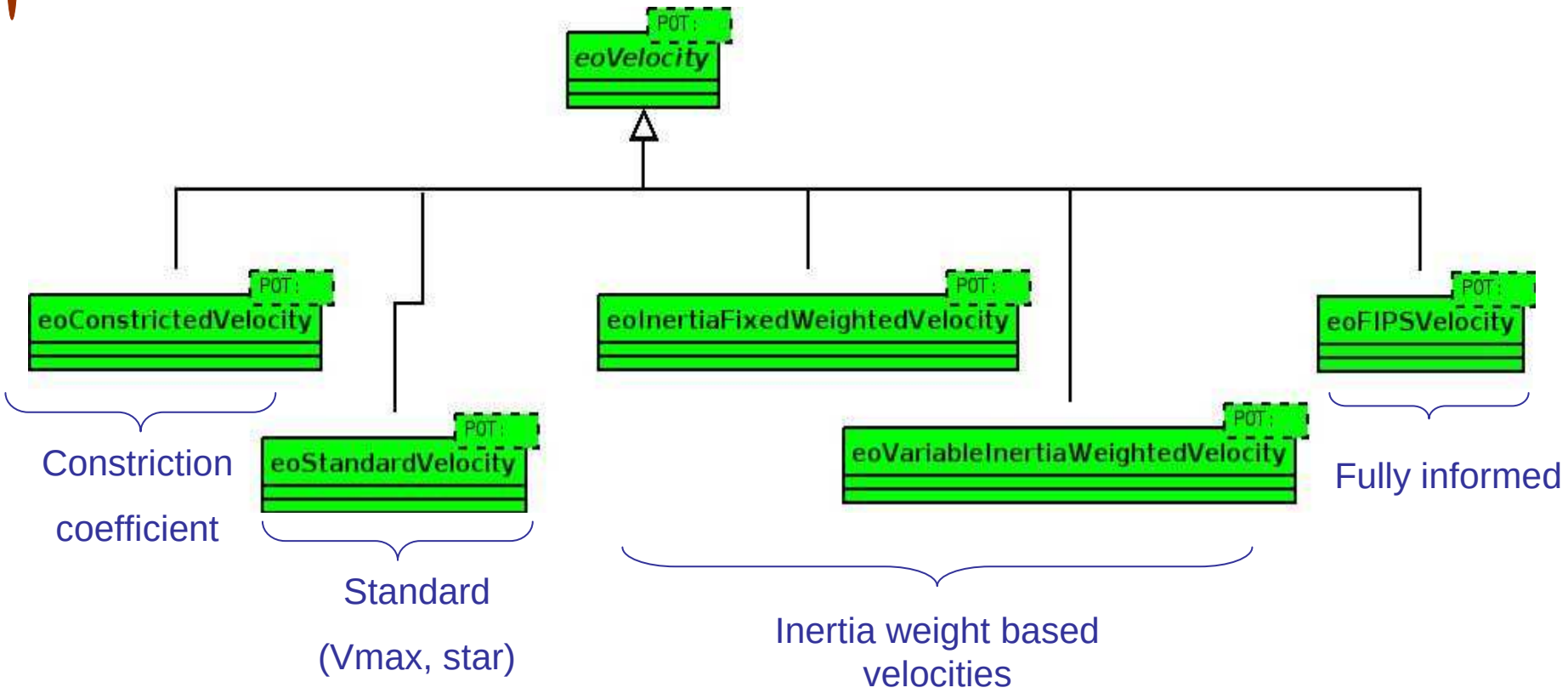
The evaluation of an individual

- This is by far the most costly step for real applications
- It might be a subroutine, a black-box simulator, or any external process (e.g. robot experiment)
- Fitness could be approximated
- Min $f(x_1, x_2) = \sqrt{x_1^2 + x_2^2}$

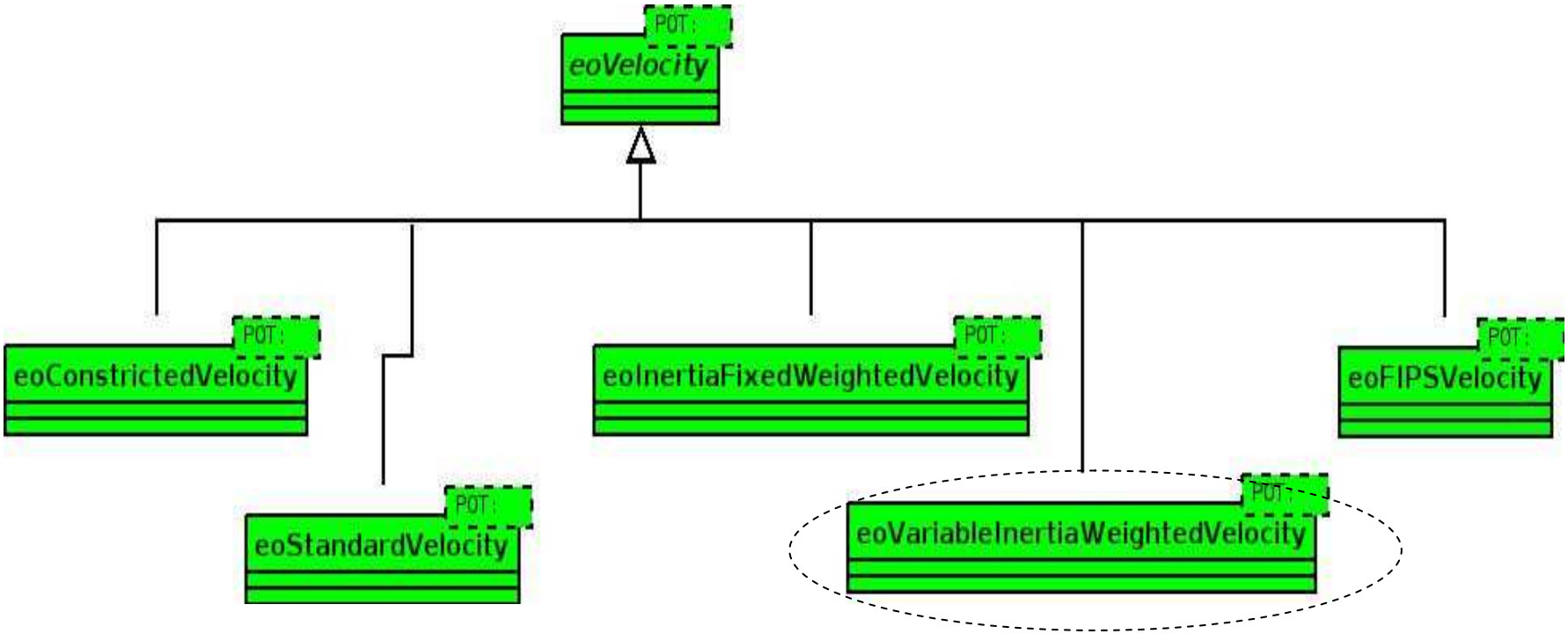
Designing a velocity

- The velocity gives the direction
- Learning factors and memory
  - Follow the global best ? In a neighbourhood ?
  - Follow the particle's best ?
- Global version faster
  - May converge to local optimums
- Local version slower
  - Not easy to be trapped into local optimums

Existent velocities in ParadisEO



Application to the NOP

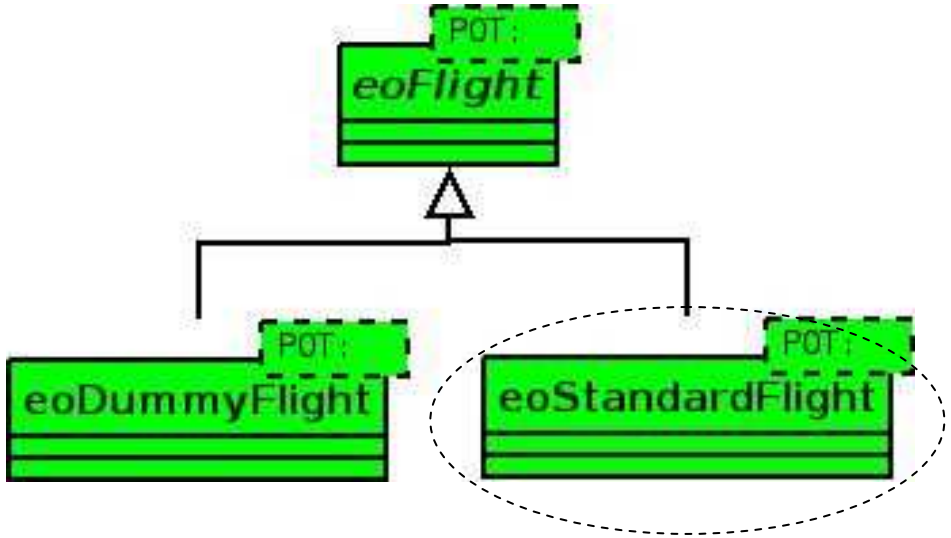


- Inertia weight factor which decreases with the number of generations

Vi = K * [Vi + c1 x (Pi best – Pi) + c2 x (Pgbest – Pi)]

Flight operator

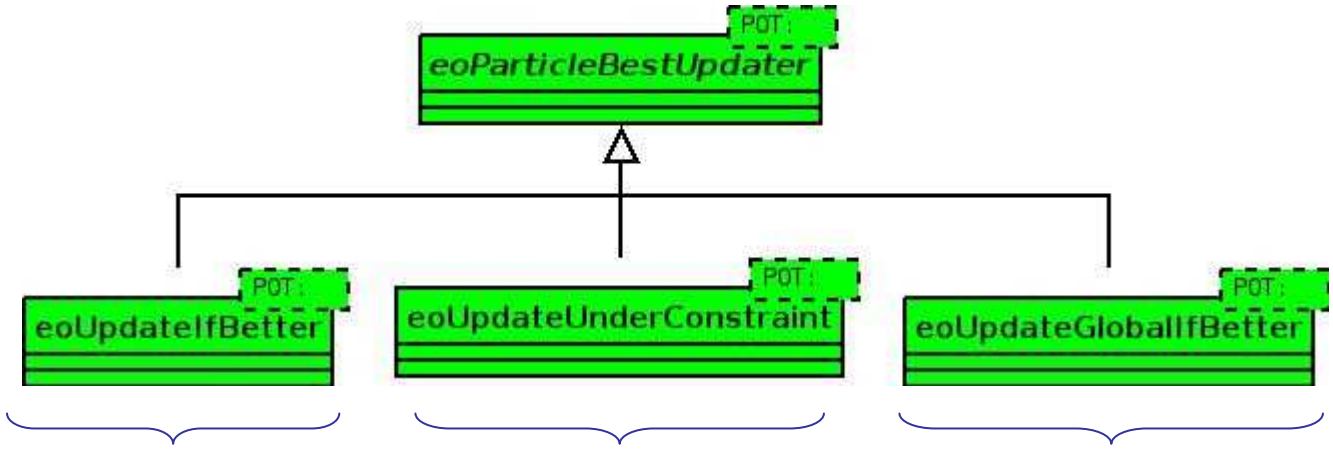
- The flight consists in adding the velocity to the current position (standard)
- ParadisEO provides this standard approach but is opened



Designing a "best updating" strategy

- The global/local directions depends on the updating strategy
- Standard approach: update the best position of a particle if the new fitness value is better
- Constraint handling - what if the solution breaks some constraint of the problem
  - penalize the fitness
  - specific methods
- Multi-objective optimization gives a set of compromise solutions

Existent updating strategies in ParadisEO

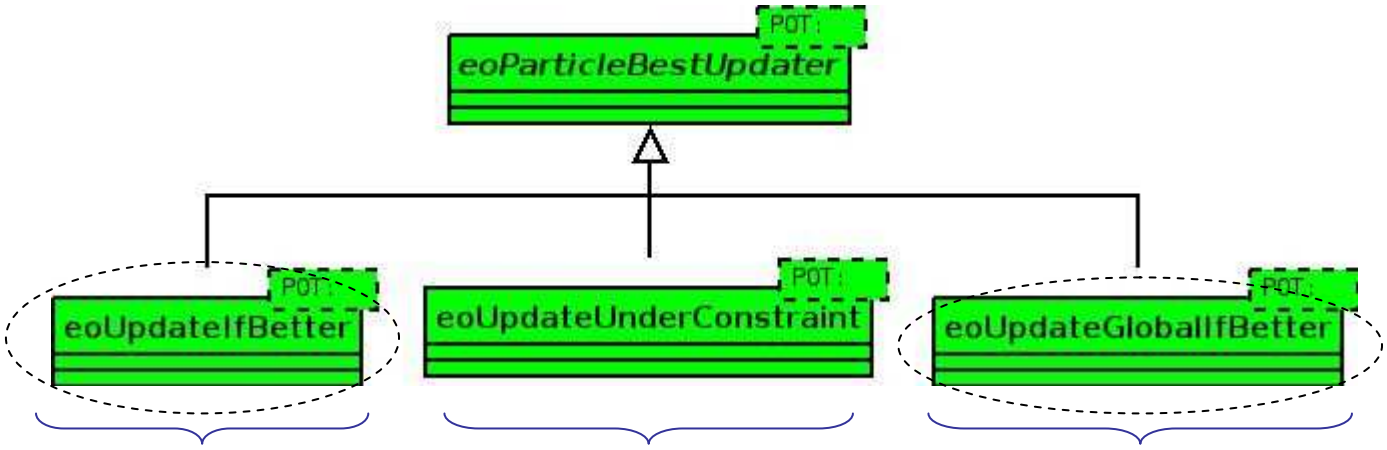


Standard particle's best update

Update the fitness if it's ε-better

Standard global best update

Application to the LSP



Standard particle's best update

Update the fitness if it's ε-better

Standard global best update

The continuation strategy

- The optimum is reached !
- Limit on CPU resources: Maximum number of fitness evaluations
- A given number of generations without improvement: for the swarm or for the global best

Core classes

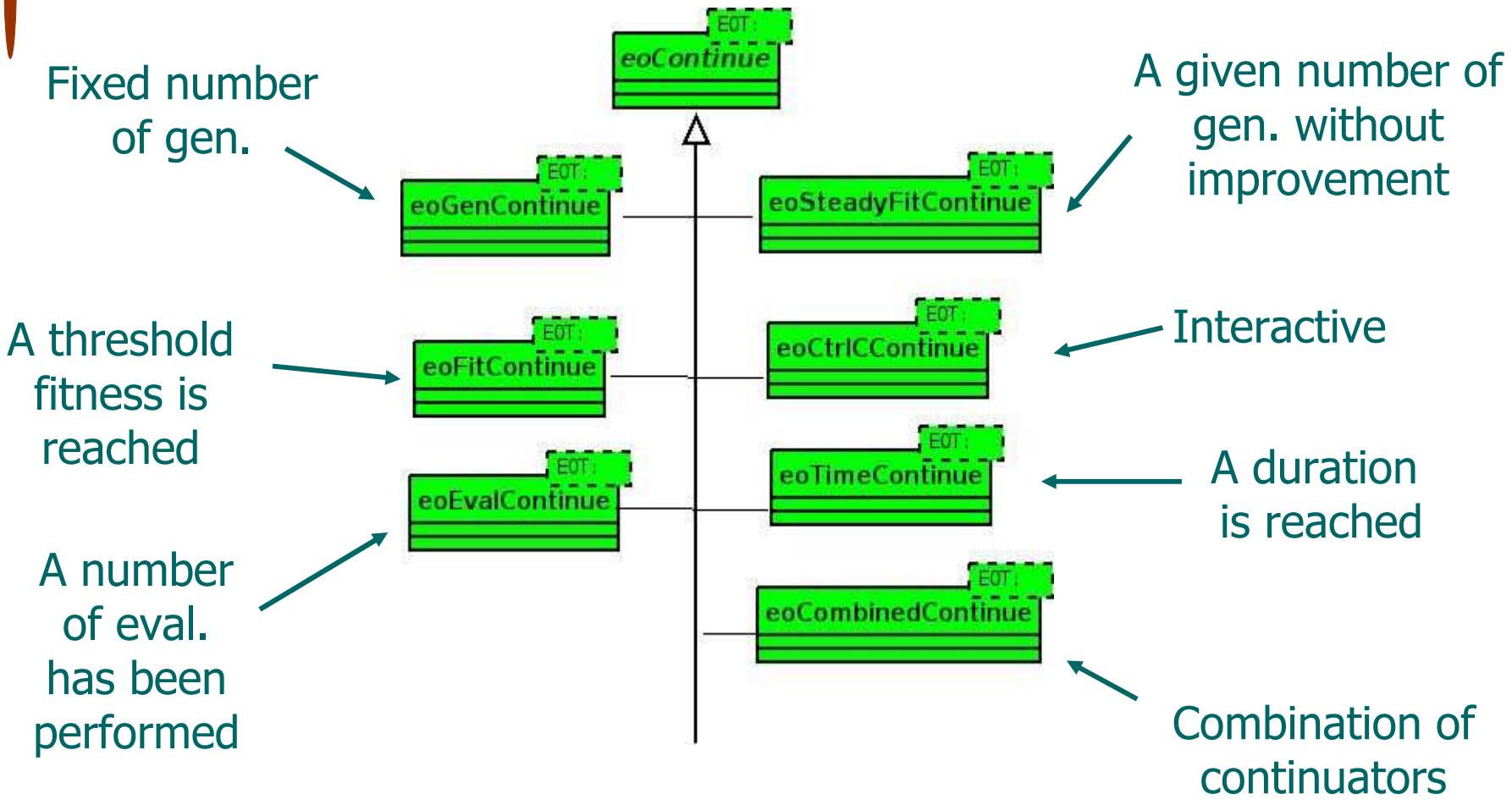
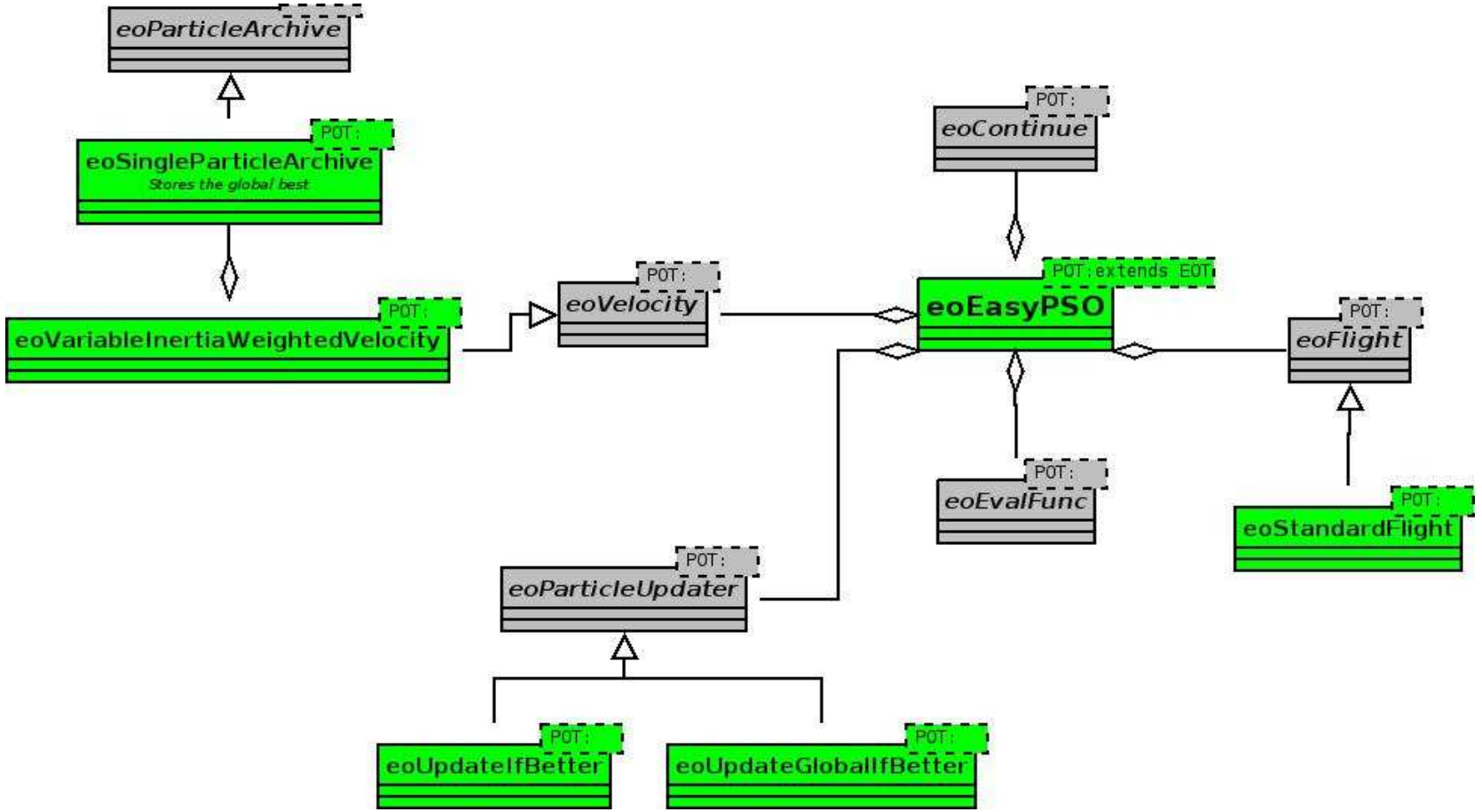


Illustration. Core classes of the Particle Swarm Algorithm



Implementation

- eoPop < Indi > pop; /* population */
- eoEvalFuncPtr<Indi, double, const Indi& > eval(real_value); /* evaluator */

- eoUniformGenerator < double >uGen (-1.0, 1.0);
- eoInitFixedLength < Indi > random (VEC_SIZE, uGen);
- eoVelocityInitFixedLength < Indi > speedRandom (VEC_SIZE, uGen);
- eoFirstIsBestInit < Indi > localInit;
- eoSingleParticleArchive < Indi > archive;
- eoBestOfAllInit < Indi > globalInit (archive); /* initializers */

- eoVariableInertiaWeightedVelocity < Indi > velocity (archive, generationCounter,MAX_GEN, C1, C2);
- eoStandardFlight < Indi > flight; /* main operators */

- eoUpdateIfBetter < Indi > updater;
- eoUpdateGlobalIfBetter < Indi > globalUpdater (archive); /* updaters */

- eoGenContinue < Indi > genCont (MAX_GEN);
- **eoEasyPSO < Indi > psa (genCont , eval, velocity, flight, updater, globalUpdater); /* PSO */**
- **psa (pop); /* run it */**

Execution and visualization

- Advanced runtime control
  - Termination condition on the swarm
  - Termination condition on the global best
- Advanced statistics and utilities
  - Visualize or store :
    - the best fitness found
    - the global best fitness
    - the average fitness
    - the standard deviation
    - the state of the population
    - ...