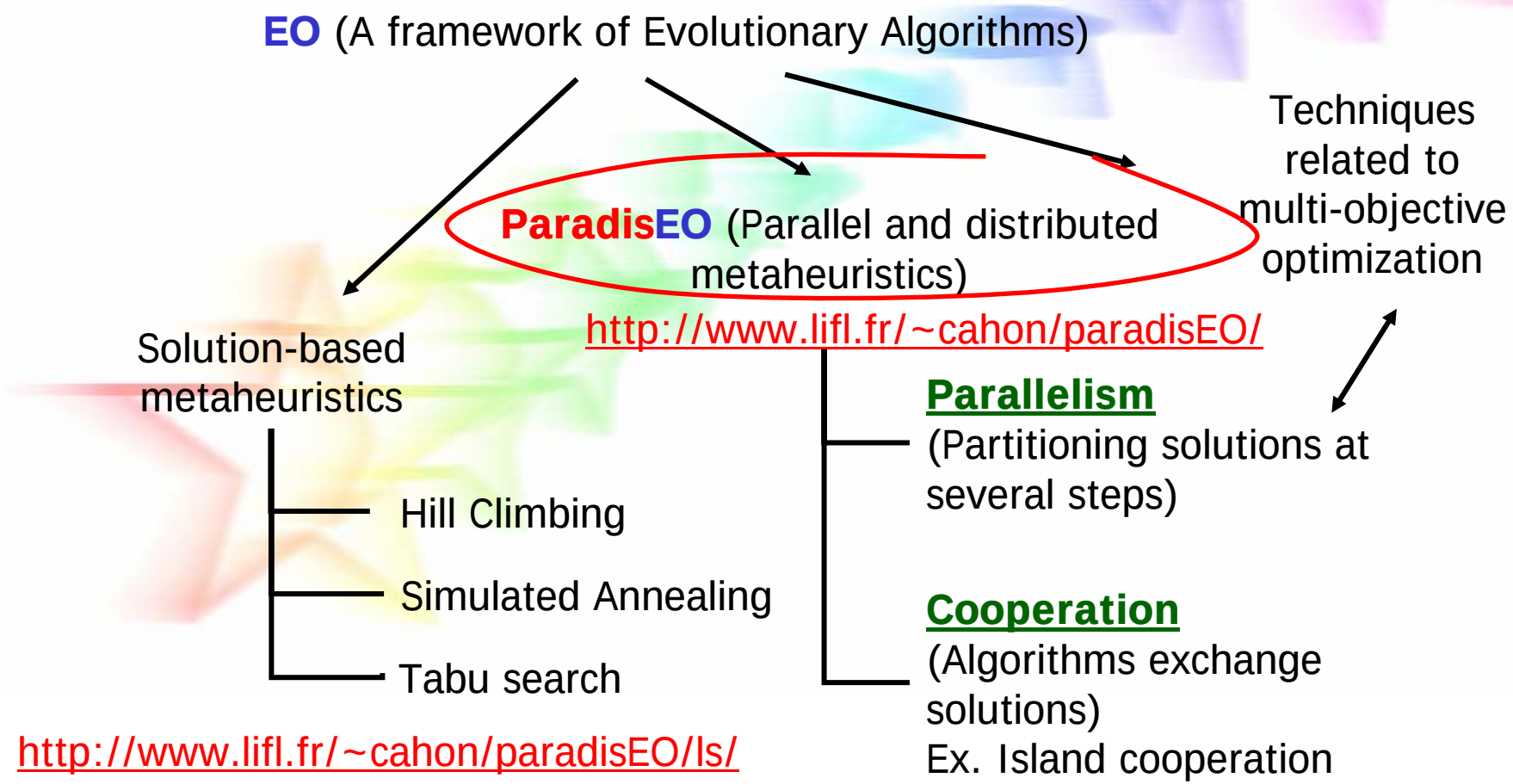
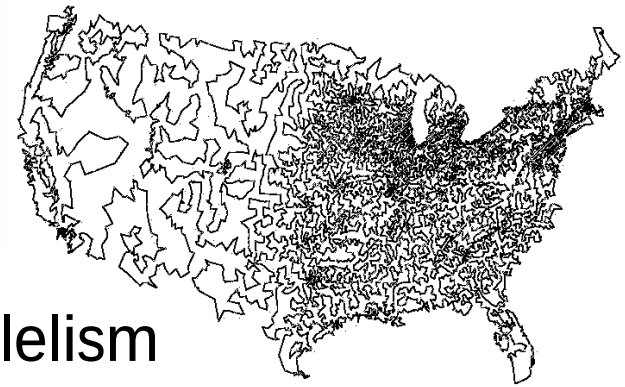
Tutorial on ParadisEO: Hybrid and Parallel Models for the TSP

Contributions

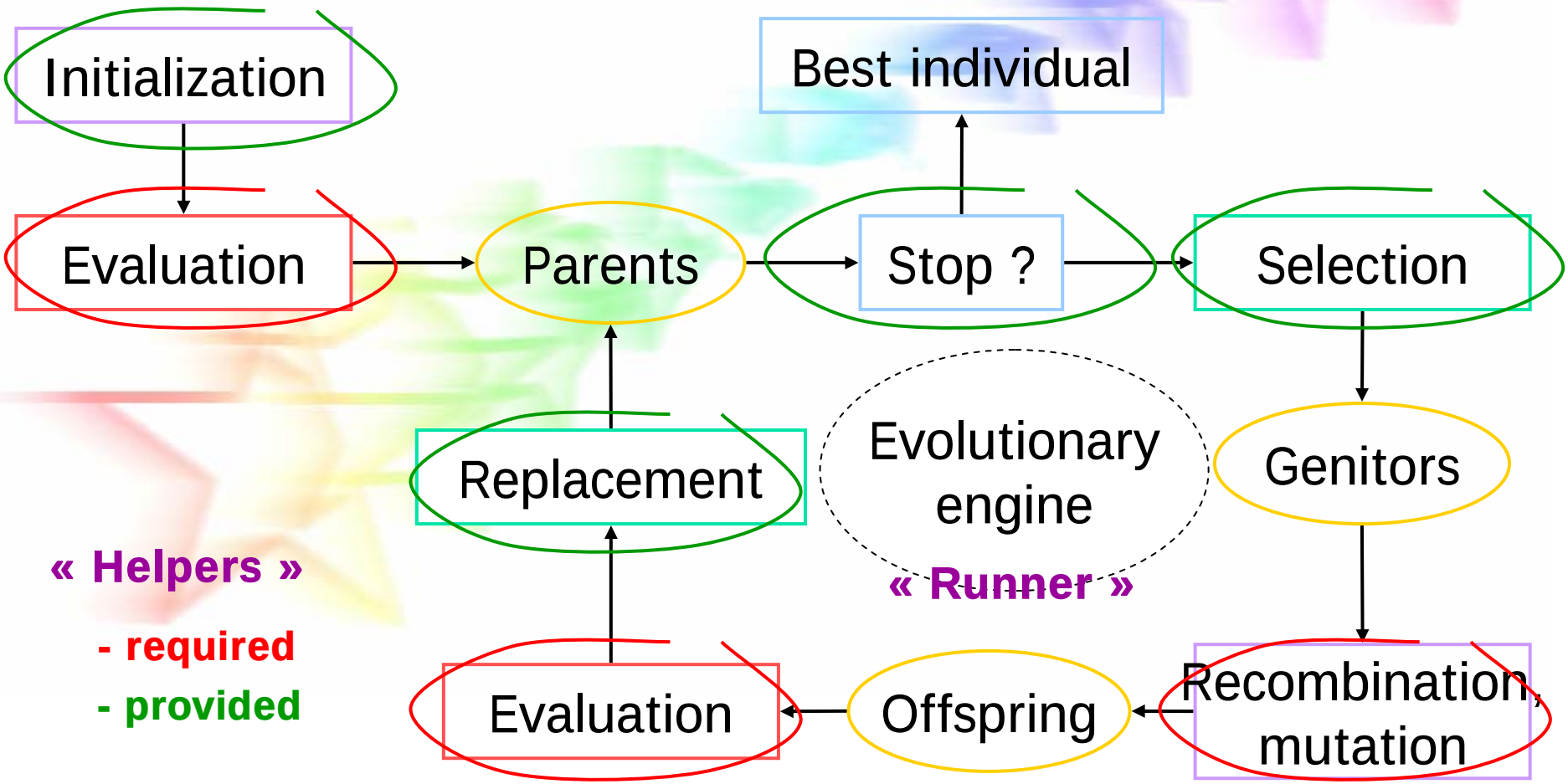


A case study : the «Traveling Salesman Problem»

- From a full graph of vertices
  → find the shortest hamiltonian cycle.
- NP-hard
- EO
  → Designing a G.A.
- ParadisEO
  → Illustrating some forms of parallelism



The sketch of an evolutionary algorithm



TSP – Environment

MPICH2 – MPI Chameleon

```
> which mpicxx
        /home/exterieur/paradise/peo/mpich2-install/bin/mpicxx
> mpdtrace
        lxo9
        lxo10
        ...
> mpdringtest 100
        time for 100 loops = 0.0880811... seconds
> mpiexec –n 8 hostname
```

```
> cd $PEOPATH
        /home/exterieur/paradise/peo/
```

TSP – Environment

Machines + SSH Keys: lxo1-lxo24, lxd1-lxd24

NFS – the account is mounted on all of the machines

```
> ssh lxo10 ls -al
> ssh lxo10 whoami
> listAssignedMachines
```

Still, you may launch the tests ONLY from the machines that were assigned to you!

```
> hostname
```

lxo3

```
> ssh lxo15
> hostname
```

lxo15

```
> exit
```

TSP Case-study – Components

```
> cd $HOME/tsp/src/
> ls –al
```

~/tsp/

src/

main.cpp – main workbench file – GAs, LS etc.

DEFINITION	route.h – chromosome definition route_init.h – random chr. initialization operator
EVALUATION	part_route_eval.h – partial fitness eval. Operator merge_route_eval.h – fitness evaluation aggregation op. route_eval.h – full fitness evaluation operator
OPERATORS	order_xover.h – OrderXover crossover operator edge_xover.h – EdgeXover crossover operator city_swap.h – CitySwap mutation operator
	{two_opt_init, two_opt_next, two_opt_incr_eval}.h

TSP Compiling and Launching

setting the correct directory for compilation:

```
> cd $HOME/tsp/
```

compiling the application

```
> make
```

cleanning the directory (erases the application, obj. files, core etc)

```
> make clean
```

launching four processes – the tsp.param specifies the config. params

```
> mpiexec –n 4 ./tsp @tsp.param
```

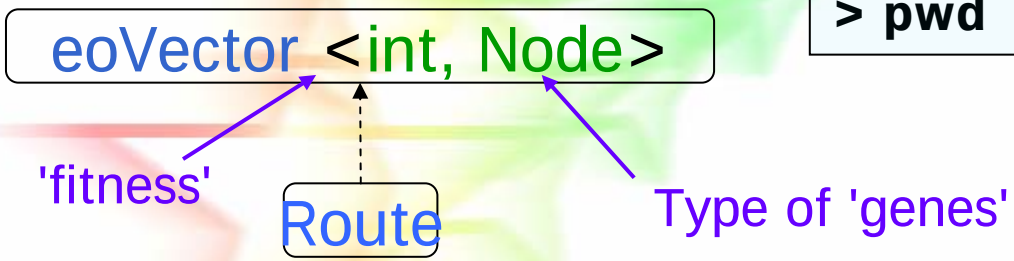
Handled solutions → routes

```
> cd $HOME/tsp/src/
```

```
> cat node.h
> cat route.h
```

```
> pwd
```

/home/exterieur/ecpaN/tsp/src



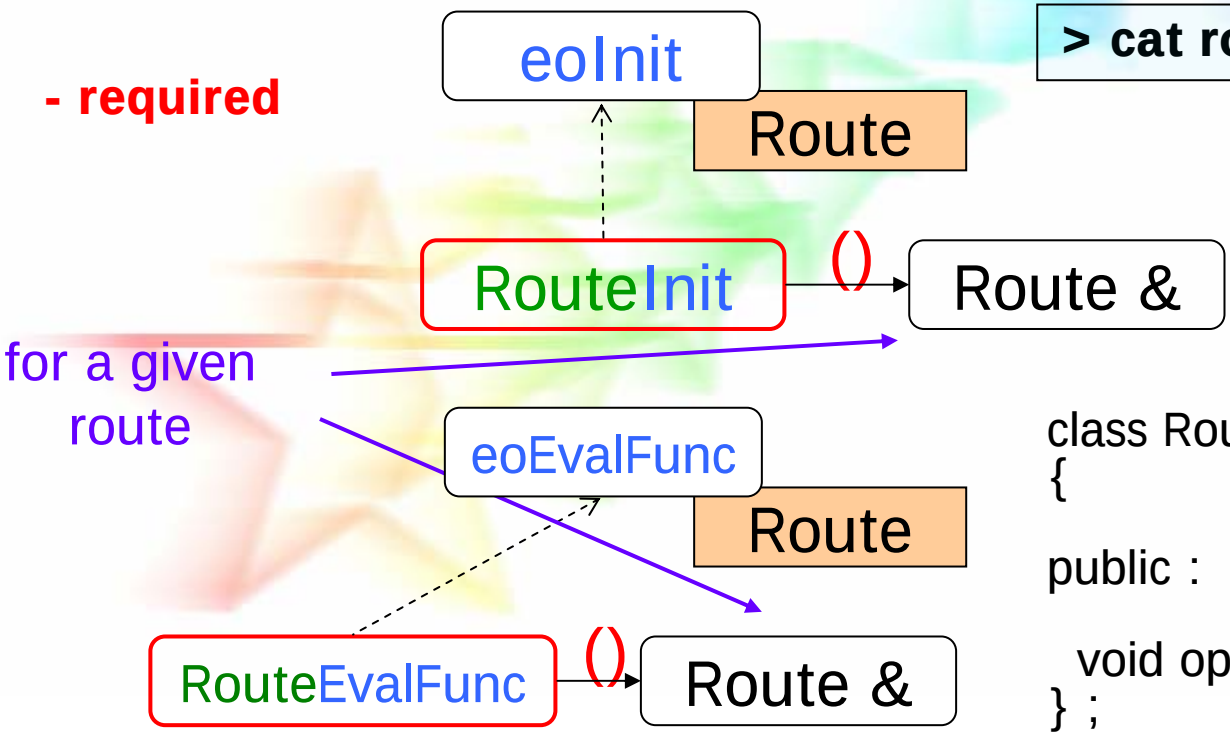

```
typedef unsigned Node;

typedef eoVector <int, Node> Route;
```

Initialization, Evaluation

> cd $HOME/tsp/src/

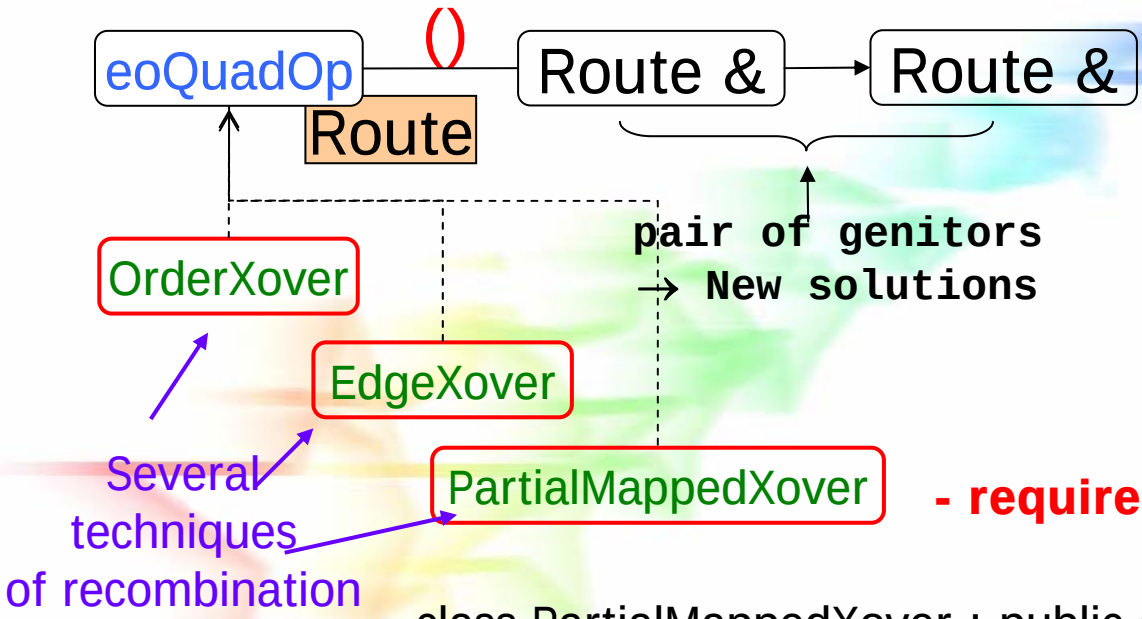
> cat route_init.h

- required




```
class RouteInit : public eoInit <Route>
{

public :

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

Recombination



- required

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

public :

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

Parallel Transform – Initial EA

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

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

eoPop <Route> edge_pop (POP_SIZE, route_init);

eoGenContinue <Route> edge_cont (NUM_GEN);

eoCheckPoint <Route> edge_checkpoint (edge_cont);

peoSeqPopEval <Route> edge_pop_eval (full_eval);
eoRankingSelect <Route> edge_select_one;
eoSelectNumber <Route> edge_select (edge_select_one, POP_SIZE);

peoParaSGATransform <Route> edge_para_transform (edge_cross, CROSS_RATE, city_swap_mut, MUT_RATE);

eoPlusReplacement <Route> edge_replace;

peoEA <Route> edge_ea (edge_checkpoint, edge_pop_eval, edge_select, edge_para_transform, edge_replace);

edge_ea (edge_pop);   /* Application to the given population */
```

> cd $HOME/tsp/src/

> emacs main.cpp &

FIXED No. OF GENERATIONS

EVALUATION

SELECTION

TRANSFORMATION OPs

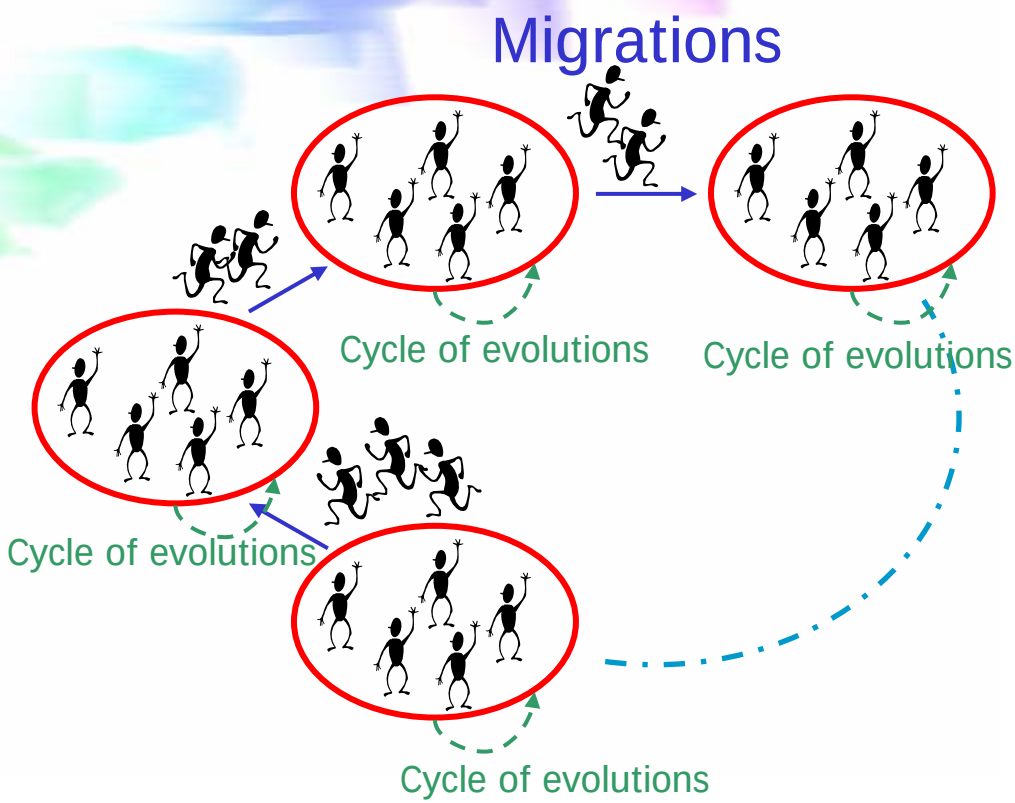
REPLACEMENT

An unifying view of three hierarchical levels

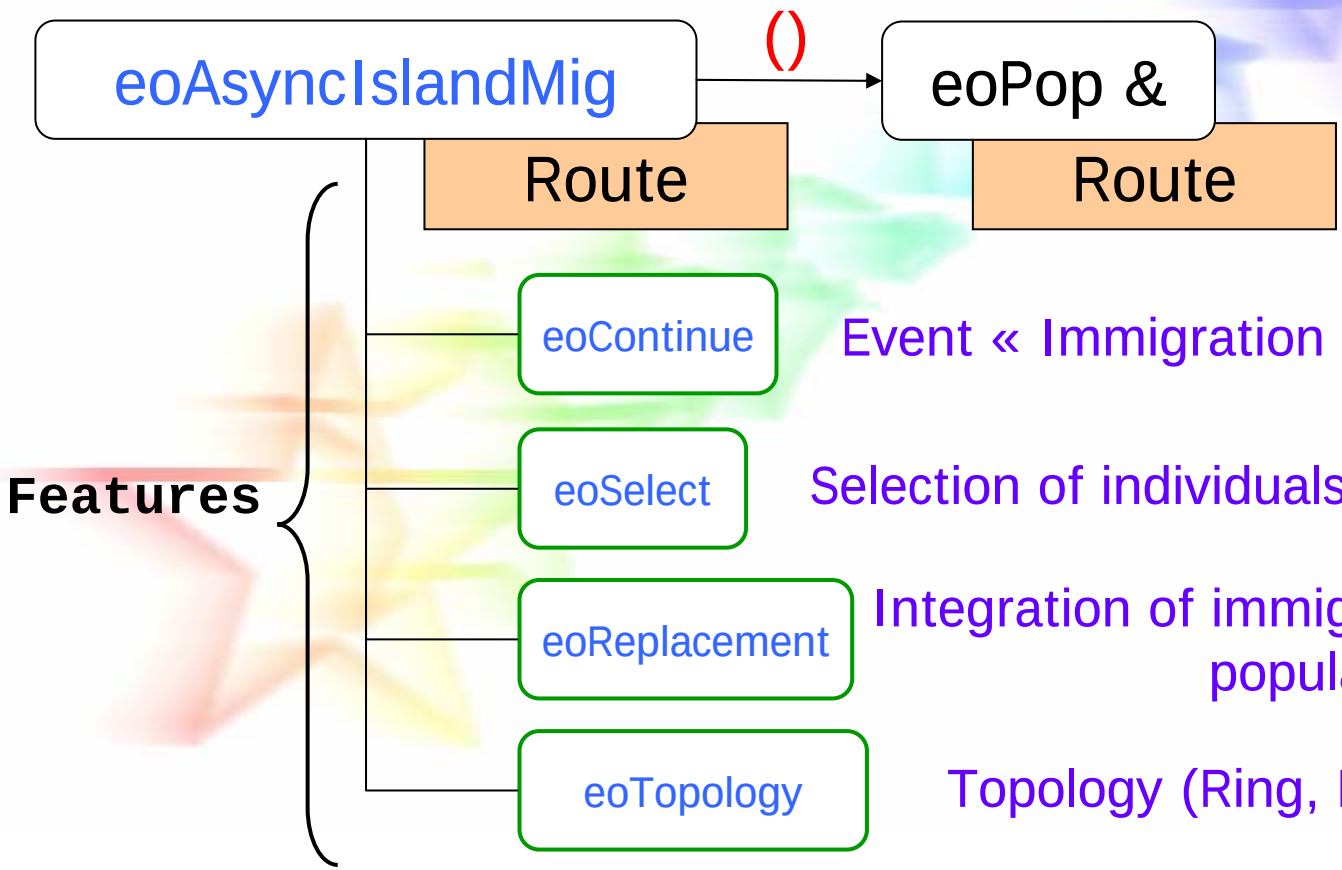
- For both the population-based and solution-based metaheuristics,
  - The deployment of concurrent independent/cooperative metaheuristics
  - The parallelization of a single step of the metaheuristic (based on distribution of the handled solutions)
  - The parallelization of the processing of a single solution

The cooperative island model

- Several asynchronous GAs (coarse grain)
  - Different sub-populations
- Asynchronous migration of individuals
  - Different stopping criteria
    - Convergence
    - Frequency / generations
- Miscellaneous
  - Heterogeneous parameters, operators, algorithms, ...



Manager of asynchronous migrations



- provided

Asynchronous island model - A

```
#define CROSS_RATE 1.0
#define MUT_RATE 0.01
#define NUM_PART_EVALS 2

#define MIG_FREQ 10
#define MIG_SIZE 10

int main (int __argc, char * * __argv)

  peo :: init (__argc, __argv);

  loadParameters (__argc, __argv);
```

1. Define the Migration Frequency and the Number of Individuals per Migration

```
#define MIG_FREQ    10
#define MIG_SIZE    10
```

constants definition area
at the beginning of the file

Asynchronous island model - B

2. Island Topology Definition

RingTopology topo;

at the beginning of the main function,
to be used later in the code for other algorithms

```
eoSelectNumber <Route> edge_select (edge_select_one, POP_SIZE);
peoParaSGATransform <Route> edge_para_transform (edge_cross, CROSS_RATE, city_swap_mut, MU
eoPlusReplacement <Route> edge_replace;

/* The migration policy */
eoPeriodicContinue <Route> edge_mig_cont (MIG_FREQ); /* Migration occurs periodically */
eoRandomSelect <Route> edge_mig_select_one; /* Emigrants are randomly selected */
eoSelectNumber <Route> edge_mig_select (edge_mig_select_one, MIG_SIZE);
eoPlusReplacement <Route> edge_mig_replace; /* Immigrants replace the worse individuals */
peoAsyncIslandMig <Route> edge_mig (edge_mig_cont, edge_mig_select,
                                   edge_mig_replace, topo, edge_pop, edge_pop);

//peoSyncIslandMig <Route> edge_mig (MIG_FREQ, edge_mig_select,
//                                   edge_mig_replace, topo, edge_pop, edge_pop);
edge_checkpoint.add (edge_mig);

peoEA <Route> edge_ea (edge_checkpoint, edge_pop_eval, edge_select,
                       edge_para_transform, edge_replace);

edge_mig.setOwner (edge_ea);
```

Asynchronous island model - C

2. Migration occurs peridiocally:

```
eoPeriodicContinue<Route> edge_mig_cont(MIG_FREQ);
```

3. Random selection of emigrants:

```
eoRandomSelect<Route> edge_mig_select_one;
```

4. Number of emigrants to be selected:

```
eoSelectNumber<Route> edge_mig_select(edge_mig_select_one, MIG_SIZE);
```

5. Replacement strategy:

```
eoPlusReplacement<Route> edge_mig_replace;
```

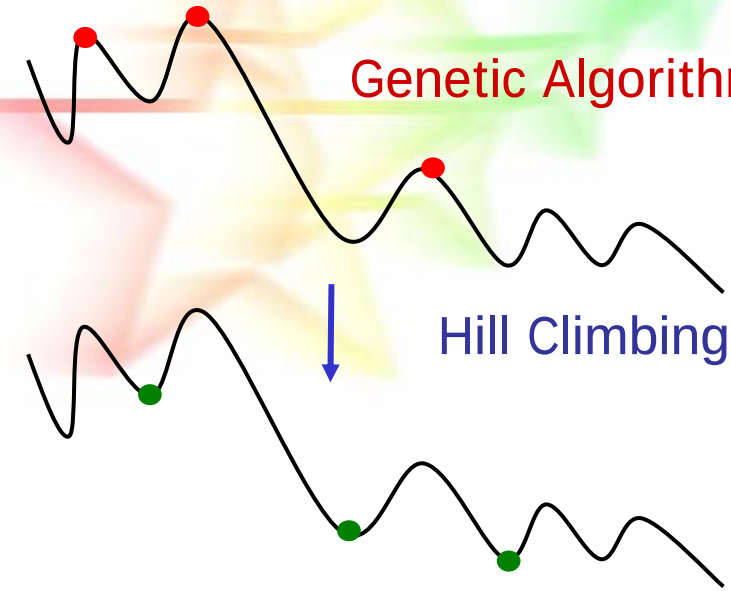
6. Asynchronous Island Model:

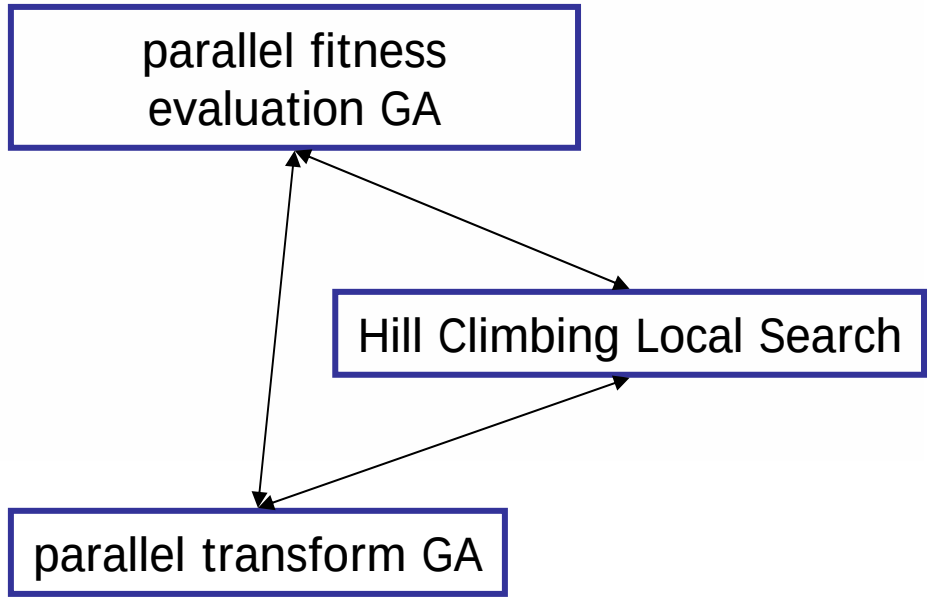
```
peoAsyncIslandMig<Route> edge_mig( edge_mig_cont,
        edge_mig_select, edge_mig_replace, topo,
        edge_pop, edge_pop);
```

```
7. edge_checkpoint.add( edge_mig );
8. edge_mig.setOwner( edge_ea );
```

Hybridization

- Power of intensification and diversification
- To improve both quality of computed solutions and the robustness of solvers





Defining the EA

```
#define HYBRID_SIZE 3

PartialMappedXover pm_cross;

eoPop <Route> pmx_pop (POP_SIZE, route_init);

eoGenContinue <Route> pmx_cont (NUM_GEN);

peoSeqPopEval <Route> pmx_pop_eval (full_eval);

eoRankingSelect <Route> pmx_select_one;
eoSelectNumber <Route> pmx_select (pmx_select_one, POP_SIZE);
eoSGATransform <Route> pmx_transform (pm_cross, CROSS_RATE, city_swap_mut,
    MUT_RATE);

peoSeqTransform <Route> pmx_para_transform (pmx_transform);

eoPlusReplacement <Route> pmx_replace;
eoCheckPoint <Route> pmx_checkpoint (pmx_cont);
```

Defining the EA - Migrations

```
/* Migration occurs periodically */
eoPeriodicContinue <Route> pmx_mig_cont (MIG_FREQ);

/* Emigrants are randomly selected */
eoRandomSelect <Route> pmx_mig_select_one;
eoSelectNumber <Route> pmx_mig_select (pmx_mig_select_one, MIG_SIZE);

/* Immigrants replace the worse individuals */
eoPlusReplacement <Route> pmx_mig_replace;

peoAsyncIslandMig <Route> pmx_mig (pmx_mig_cont, pmx_mig_select,
    pmx_mig_replace, topo, pmx_pop, pmx_pop);

//peoSyncIslandMig <Route> pmx_mig (MIG_FREQ, pmx_mig_select, pmx_mig_replace,
    topo, pmx_pop, pmx_pop);

pmx_checkpoint.add (pmx_mig);
```

TSP - Hill Climbing

```
TwoOptInit pmx_two_opt_init;
TwoOptNext pmx_two_opt_next;
TwoOptIncrEval pmx_two_opt_incr_eval;

moBestImprSelect <TwoOpt> pmx_two_opt_move_select;

moHC <TwoOpt> hc (pmx_two_opt_init, pmx_two_opt_next,
    pmx_two_opt_incr_eval, pmx_two_opt_move_select, full_eval);

eoPeriodicContinue <Route> pmx_ls_cont (MIG_FREQ);
eoRandomSelect <Route> pmx_ls_select_one;
eoSelectNumber <Route> pmx_ls_select (pmx_ls_select_one, HYBRID_SIZE);
eoPlusReplacement <Route> pmx_ls_replace;

peoSyncMultiStart <Route> pmx_ls (pmx_ls_cont, pmx_ls_select, pmx_ls_replace, hc,
    pmx_pop);
pmx_checkpoint.add (pmx_ls);
```

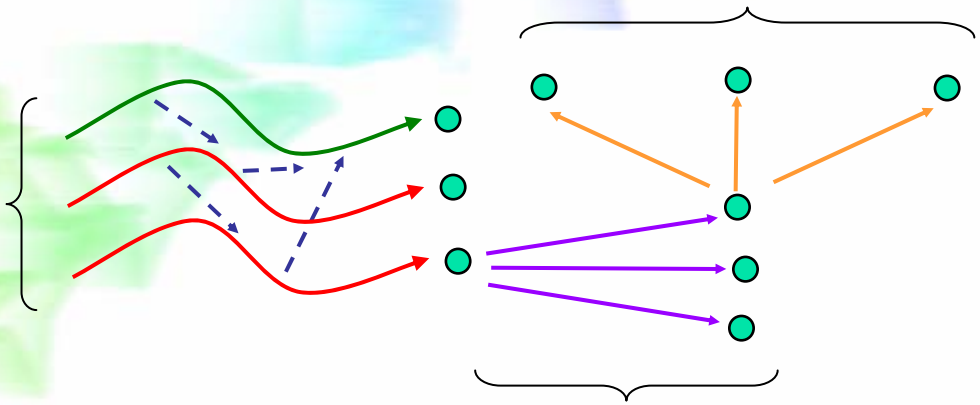
EA + HC Hybridization

```
peoEA <Route> pmx_ea (pmx_checkpoint, pmx_pop_eval,
        pmx_select, pmx_para_transform, pmx_replace);

pmx_mig.setOwner (pmx_ea);
pmx_ls.setOwner (pmx_ea);

pmx_ea (pmx_pop);
```

Design of several levels of parallelization/hybridization

Processing of a single solution
(Objective / Data partitioning)

Independent walks,
Multi-start model,
Hybridization/Cooperation
of metaheuristics



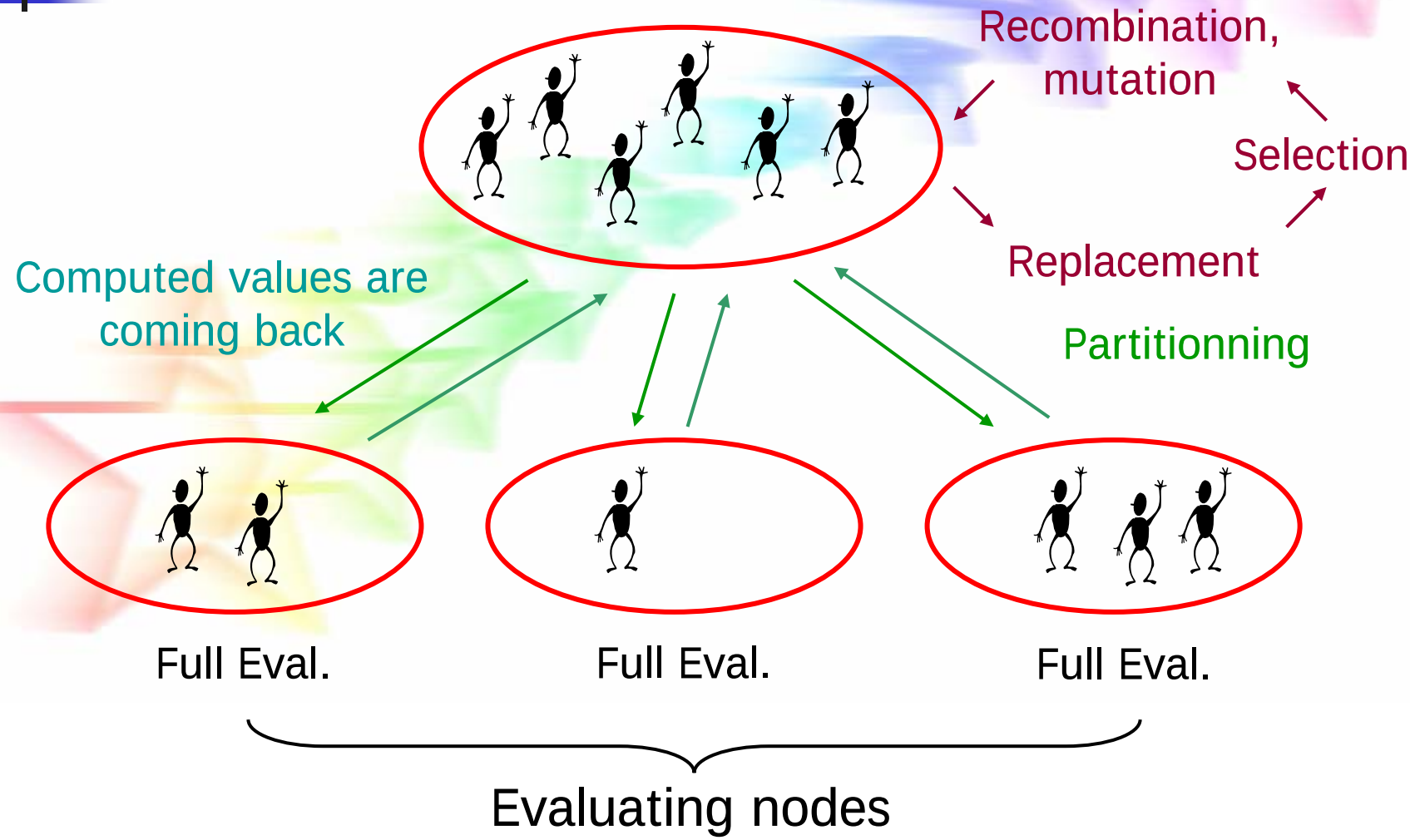
Parallel evaluation of
the neighborhood/population

Scalability

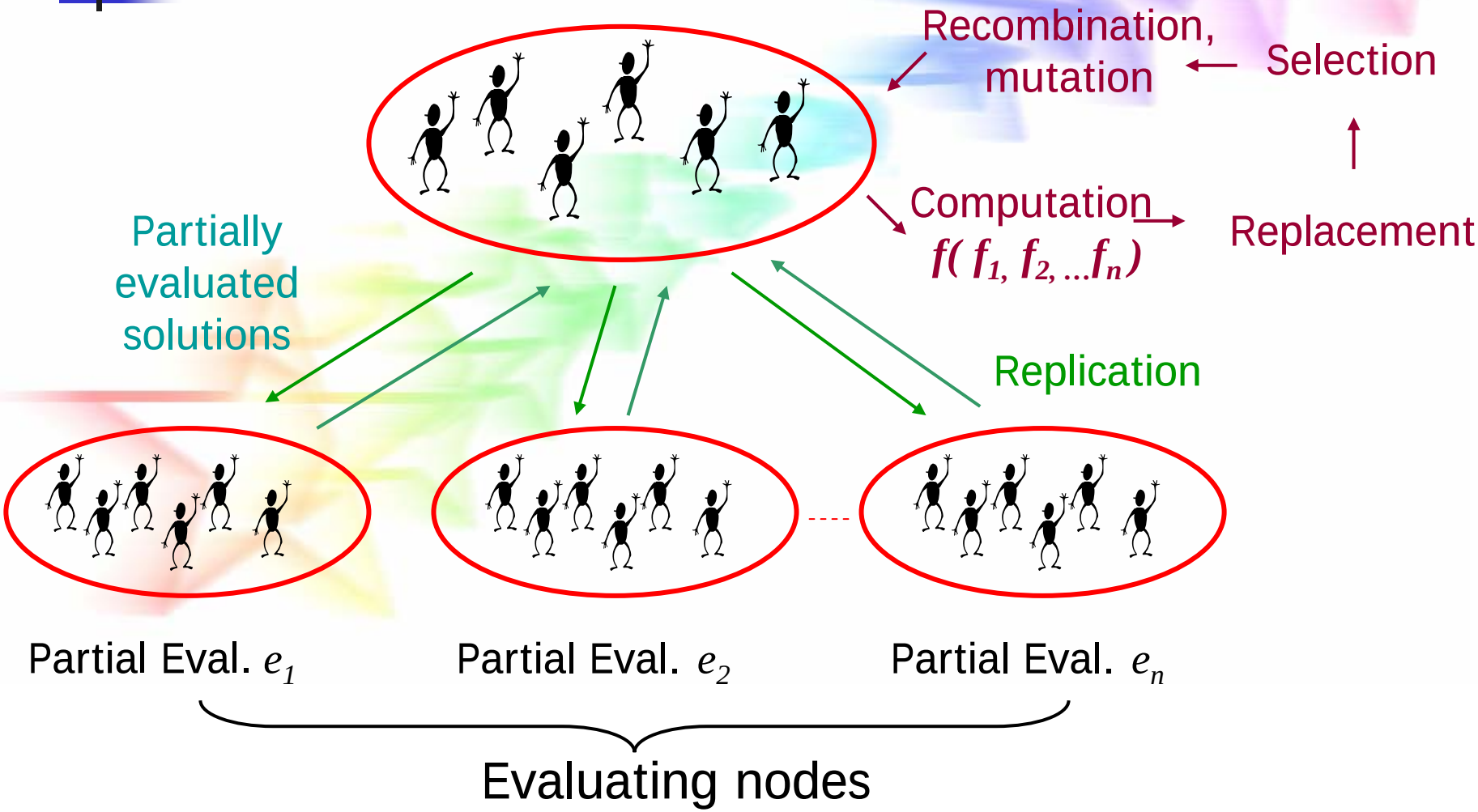
$$|H| \times |P| \times |S|$$

Heuristic — Population / Neighborhood — Solution

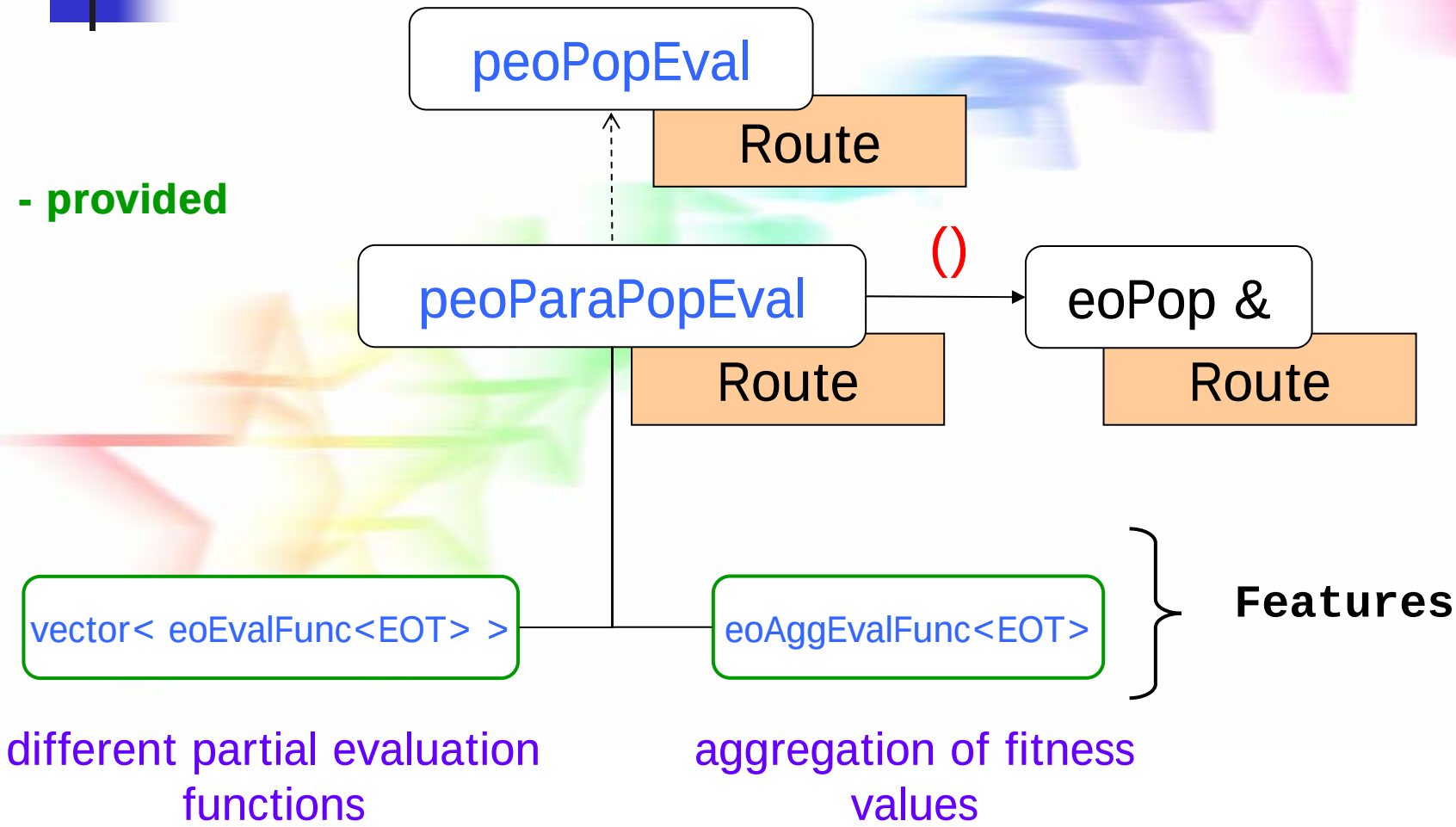
The parallel evaluation of the population



The parallel evaluation of a single solution



The distributed evaluation step



Parallel Transform GA

```
eoPop <Route> edge_pop (POP_SIZE, route_init);  /* Population */

eoGenContinue <Route> edge_cont (NUM_GEN); /* A fixed number of iterations */
eoCheckPoint <Route> edge_checkpoint (edge_cont); /* Checkpoint */
peoSeqPopEval <Route> edge_pop_eval (full_eval);
eoRankingSelect <Route> edge_select_one;
eoSelectNumber <Route> edge_select (edge_select_one, POP_SIZE);

peoParaSGATransform <Route> edge_para_transform (edge_cross, CROSS_RATE,
    city_swap_mut, MUT_RATE);

eoPlusReplacement <Route> edge_replace;
```

Parallel Evaluation

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

MergeRouteEval merge_eval;

std :: vector <eoEvalFunc <Route> *> part_eval;

for (unsigned i = 1 ; i <= NUM_PART_EVALS ; i ++)
   part_eval.push_back (new PartRouteEval ((float) (i - 1) / NUM_PART_EVALS, (float) i /
     NUM_PART_EVALS));

RingTopology topo;

eoPop <Route> ox_pop (POP_SIZE, route_init);  /* Population */

eoGenContinue <Route> ox_cont (NUM_GEN); /* A fixed number of iterations */

peoParaPopEval <Route> ox_pop_eval (part_eval, merge_eval);

eoStochTournamentSelect <Route> ox_select_one;
eoSelectNumber <Route> ox_select (ox_select_one, POP_SIZE);
```

Asynchronous Island Model

eoPeriodicContinue <Route> ox_mig_cont (**MIG_FREQ**); **/* Migration occurs periodically */**
eoRandomSelect <Route> ox_mig_select_one**; /* Emigrants are randomly selected */**
eoSelectNumber <Route> ox_mig_select (ox_mig_select_one, **MIG_SIZE**);
eoPlusReplacement <Route> ox_mig_replace**; /* Immigrants replace the worse individuals */**

```
peoAsyncIslandMig <Route> ox_mig (ox_mig_cont, ox_mig_select, ox_mig_replace, topo,
    ox_pop, ox_pop);
//peoSyncIslandMig <Route> ox_mig (MIG_FREQ, ox_mig_select, ox_mig_replace, topo,
    ox_pop, ox_pop);
```

ox_checkpoint.add (ox_mig);

peoEA <Route> ox_ea (ox_checkpoint, ox_pop_eval, ox_select, ox_para_transform, ox_replace);
ox_mig.setOwner (ox_ea);

ox_ea (ox_pop); /* Application to the given population */

Local Search Hybridization

```
TwoOptInit pmx_two_opt_init;
TwoOptNext pmx_two_opt_next;
TwoOptIncrEval pmx_two_opt_incr_eval;
moBestImprSelect <TwoOpt> pmx_two_opt_move_select;

moHC <TwoOpt> hc (pmx_two_opt_init, pmx_two_opt_next, pmx_two_opt_incr_eval,
    pmx_two_opt_move_select, full_eval);
```

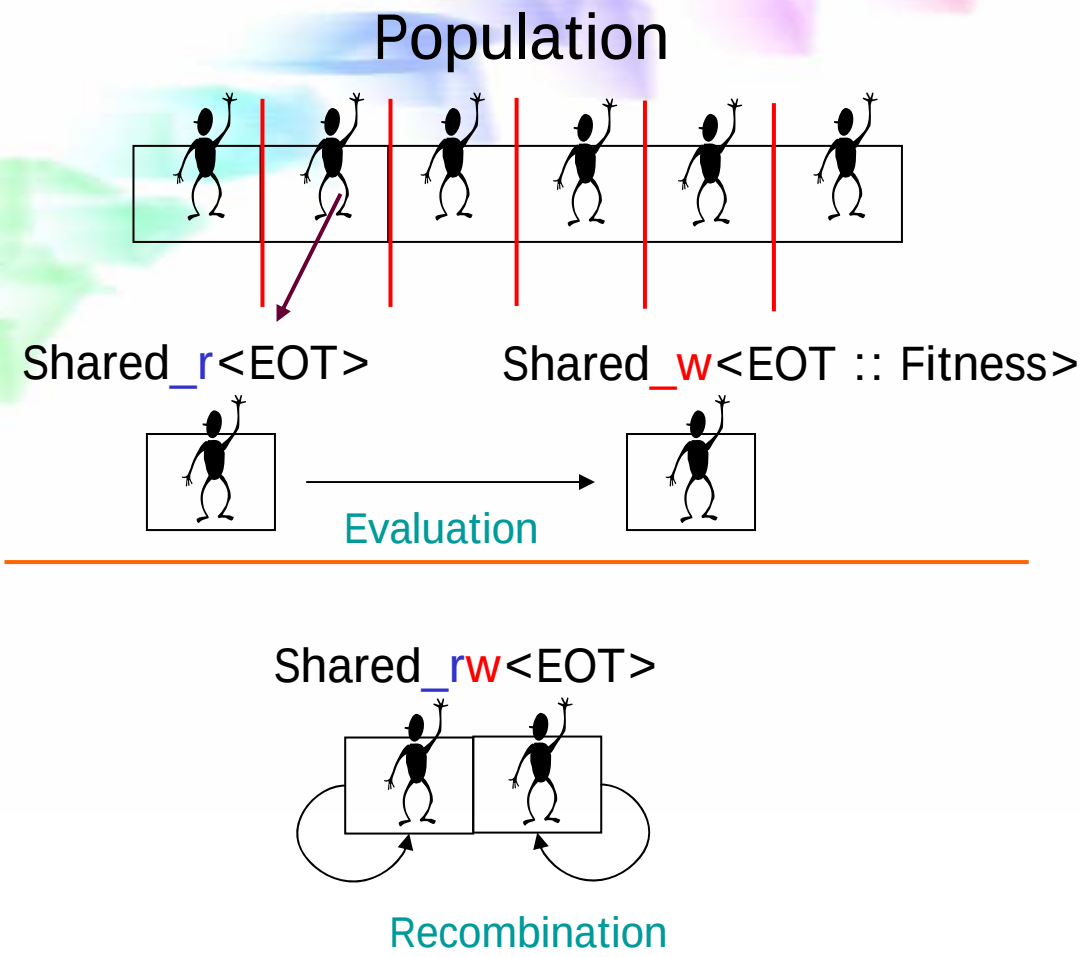
```
eoPeriodicContinue <Route> pmx_ls_cont (MIG_FREQ); /* Hybridization occurs periodically */
eoRandomSelect <Route> pmx_ls_select_one;
eoSelectNumber <Route> pmx_ls_select (pmx_ls_select_one, HYBRID_SIZE);
eoPlusReplacement <Route> pmx_ls_replace;
```

```
peoSyncMultiStart <Route> pmx_ls (pmx_ls_cont, pmx_ls_select, pmx_ls_replace, hc, pmx_pop);
pmx_checkpoint.add (pmx_ls);
```

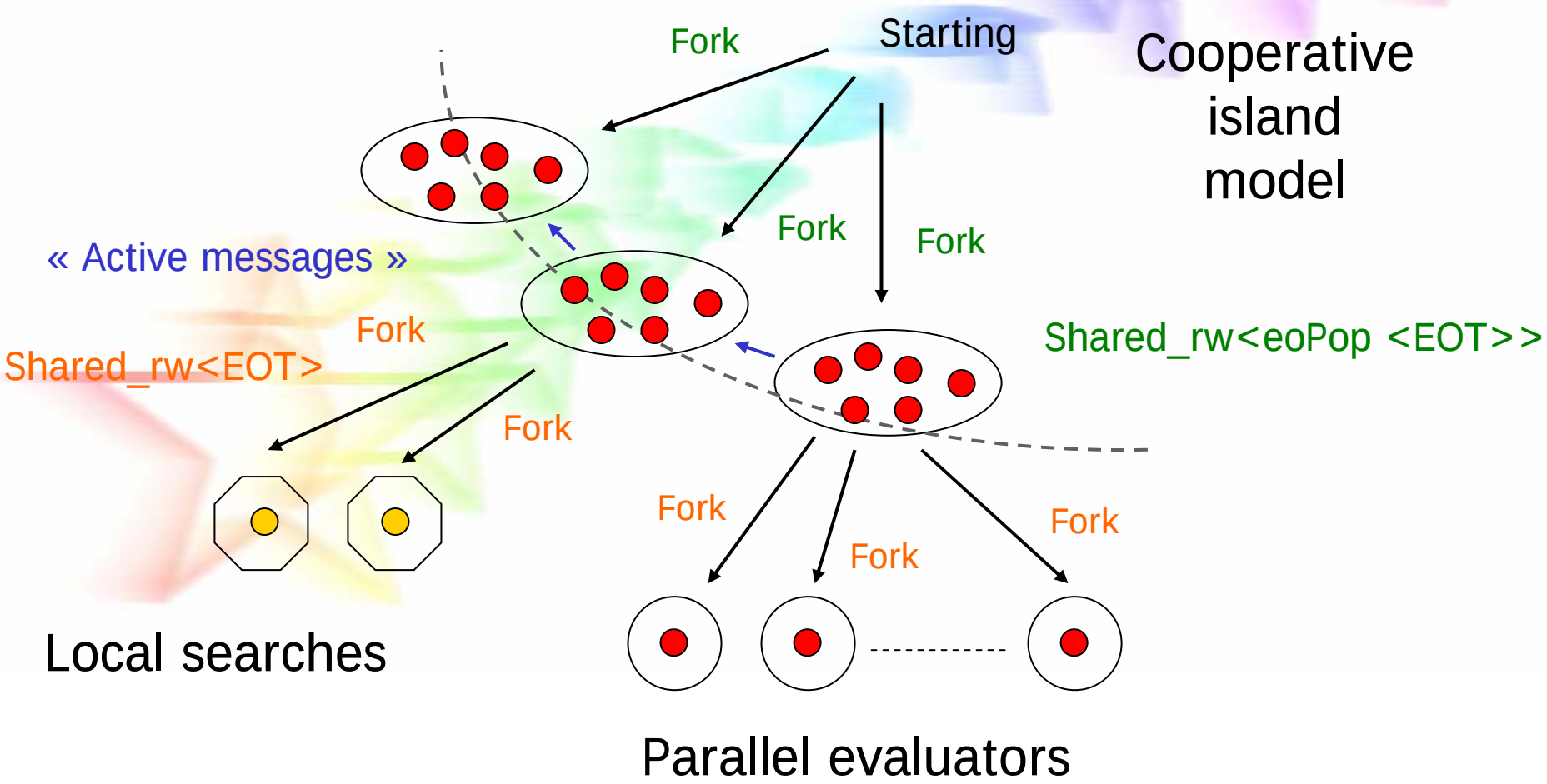
```
peoEA <Route> pmx_ea (pmx_checkpoint, pmx_pop_eval, pmx_select, pmx_para_transform, pmx_replace);
pmx_mig.setOwner (pmx_ea);
pmx_ls.setOwner (pmx_ea);

pmx_ea (pmx_pop);   /* Application to the given population */
```

An example of decomposition in tasks (Low level)

- Internal functions embedded in E.As
  → Partitioning the « data » for the steps of
  - Selection
  - Crossover
  - Evaluation
  - Replacement



An example of decomposition in tasks (High level)



Real-world applications developed with ParadisEO

- Data Mining in Near-Infrared Spectroscopy
- Radio Network design
- Genomics

Conclusion

- ParadisEO is a white-box OO framework
  - Clear conceptual separation of solution methods and problems
  - Maximum design and code reuse
  - High flexibility (fine-grained EO objects)
- ParadisEO provides a broad range of features
  - Evolutionary algorithms, local searches and natural hybridization mechanisms
    - Invariant part provided
  - Various transparent and portable parallel/distributed models
- Experimental evaluation on academic problems and industrial applications
  - High reuse capabilities, efficiency of the parallel/distributed models