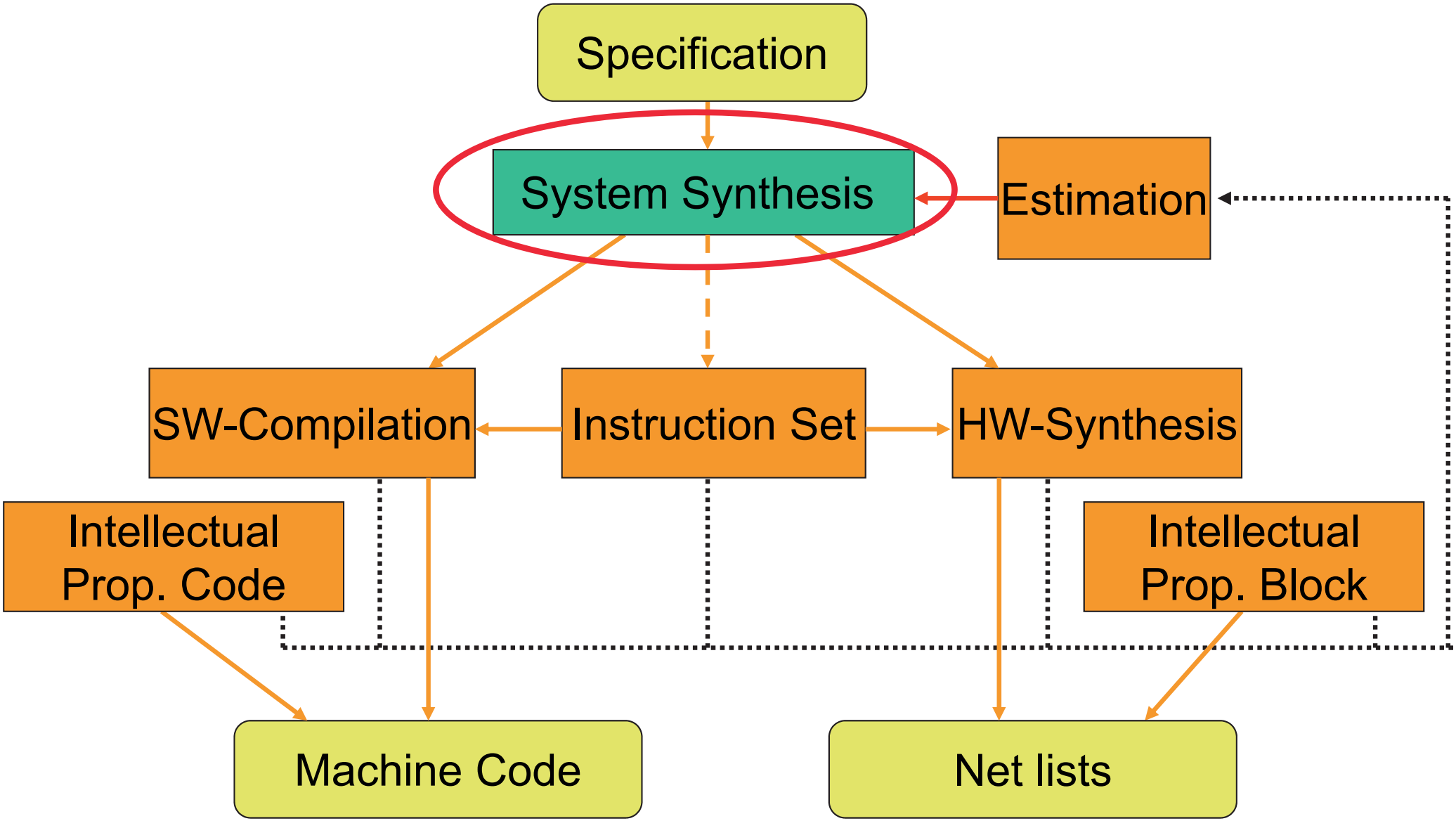
Hardware-Software Codesign

5. Multi-Criteria Optimization

Lothar Thiele

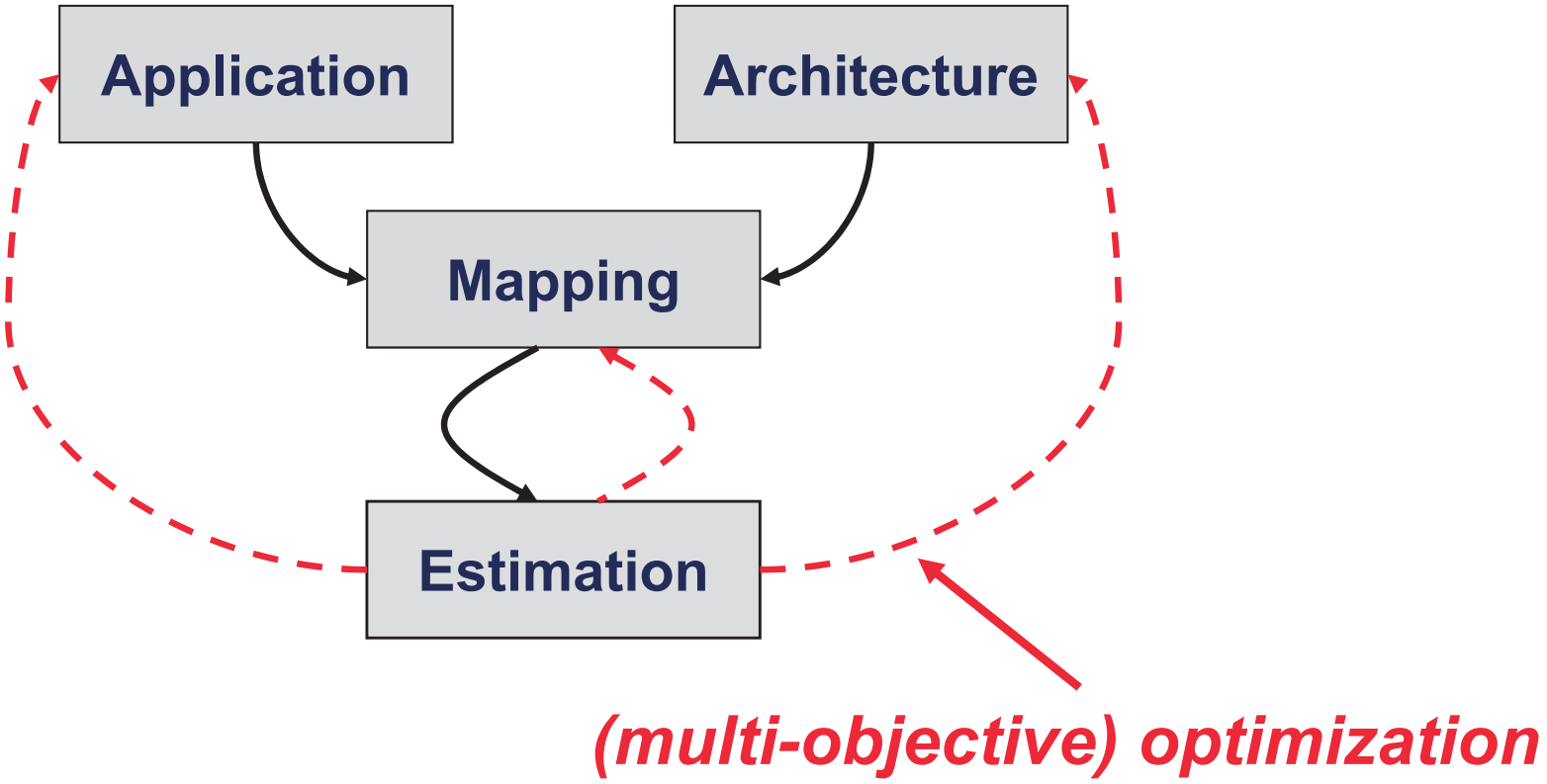
System Design



Lecture Synopsis

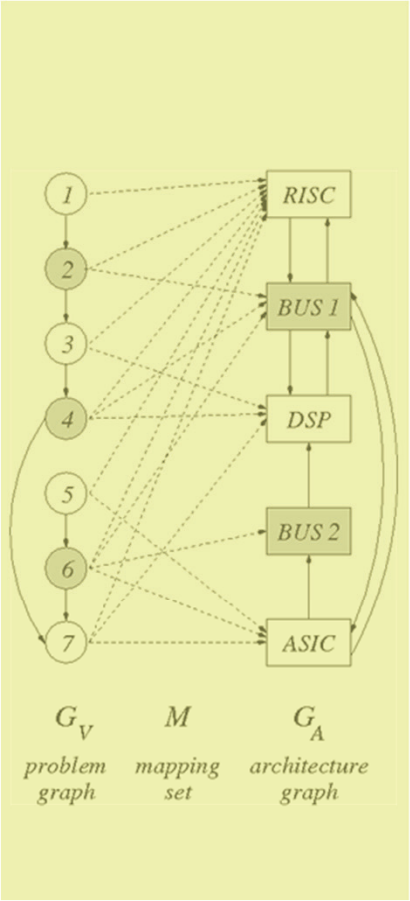
- ***Introduction***
- Multiobjective Optimization
- Multiobjective Evolutionary Algorithms
- Implementation Aspects

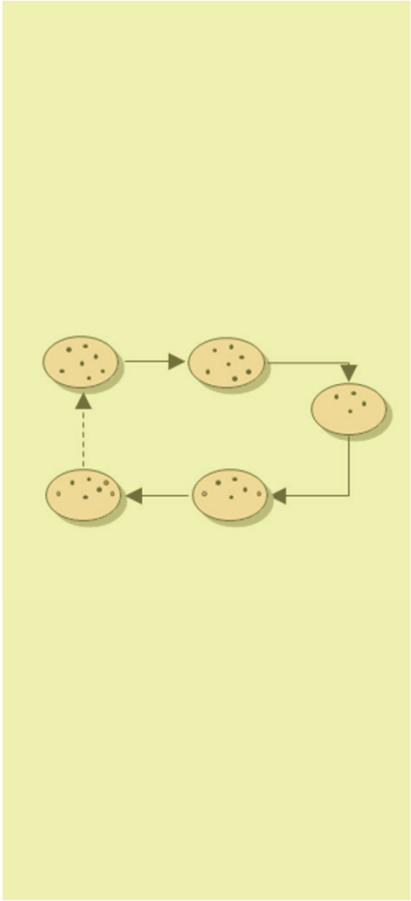
Design Space Exploration

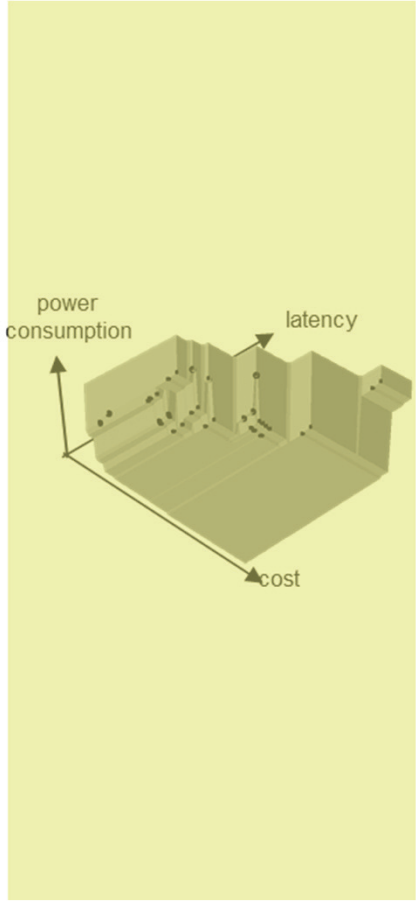


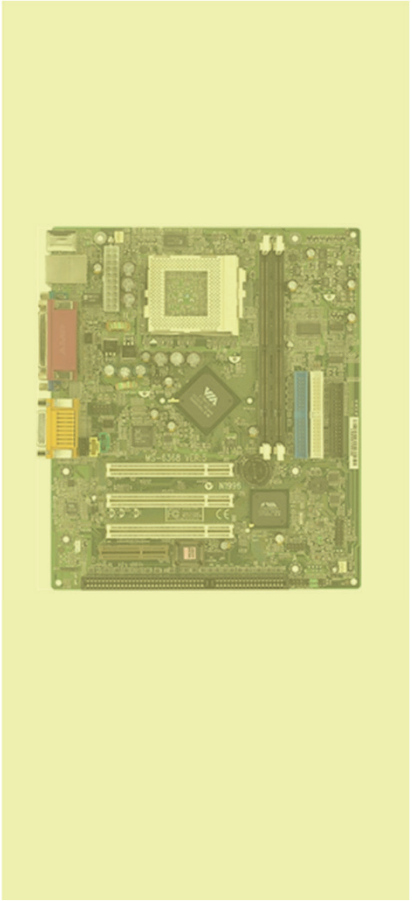
Design Space Exploration

Specification → Optimization → Evaluation → Implementation

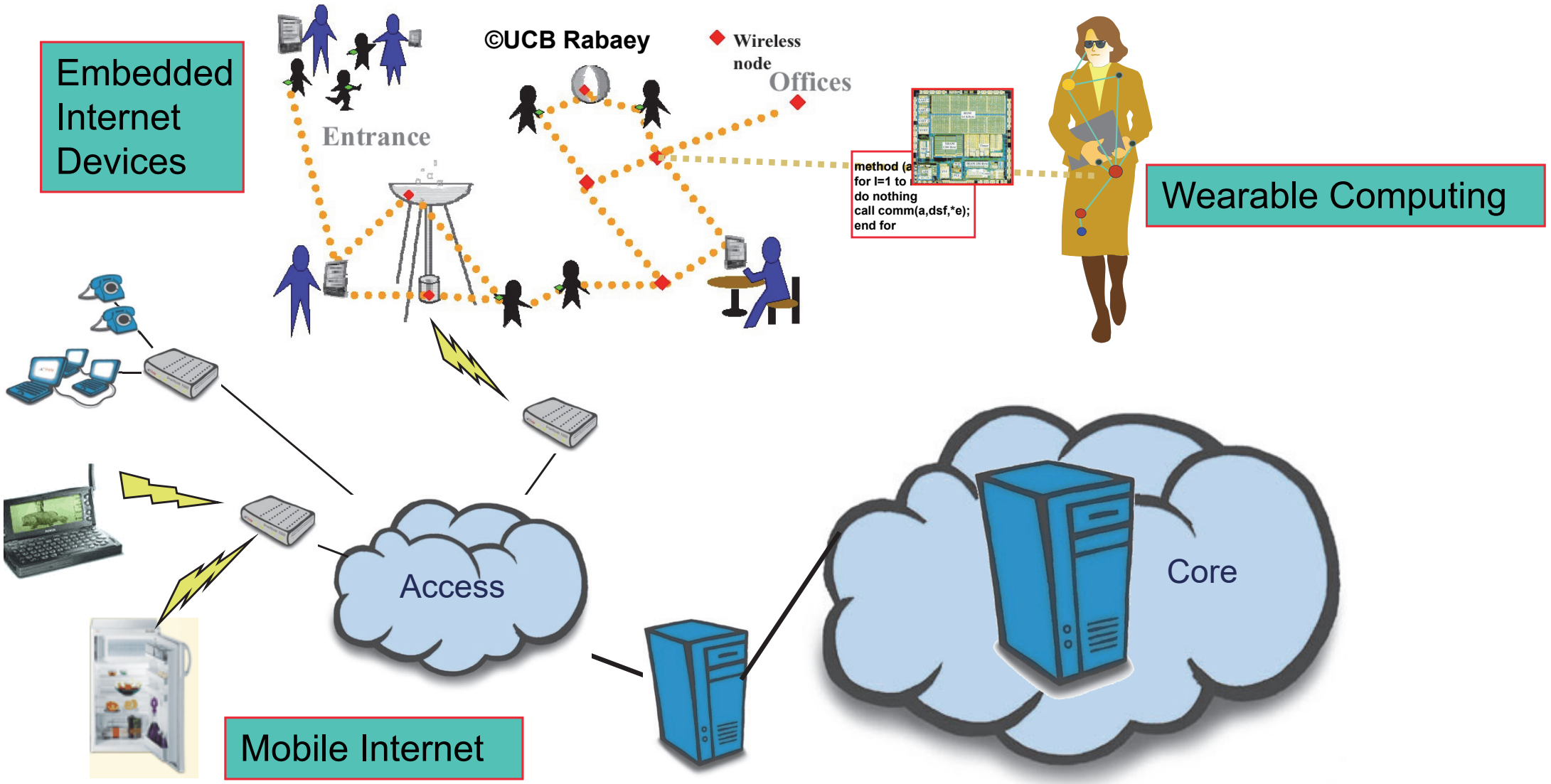






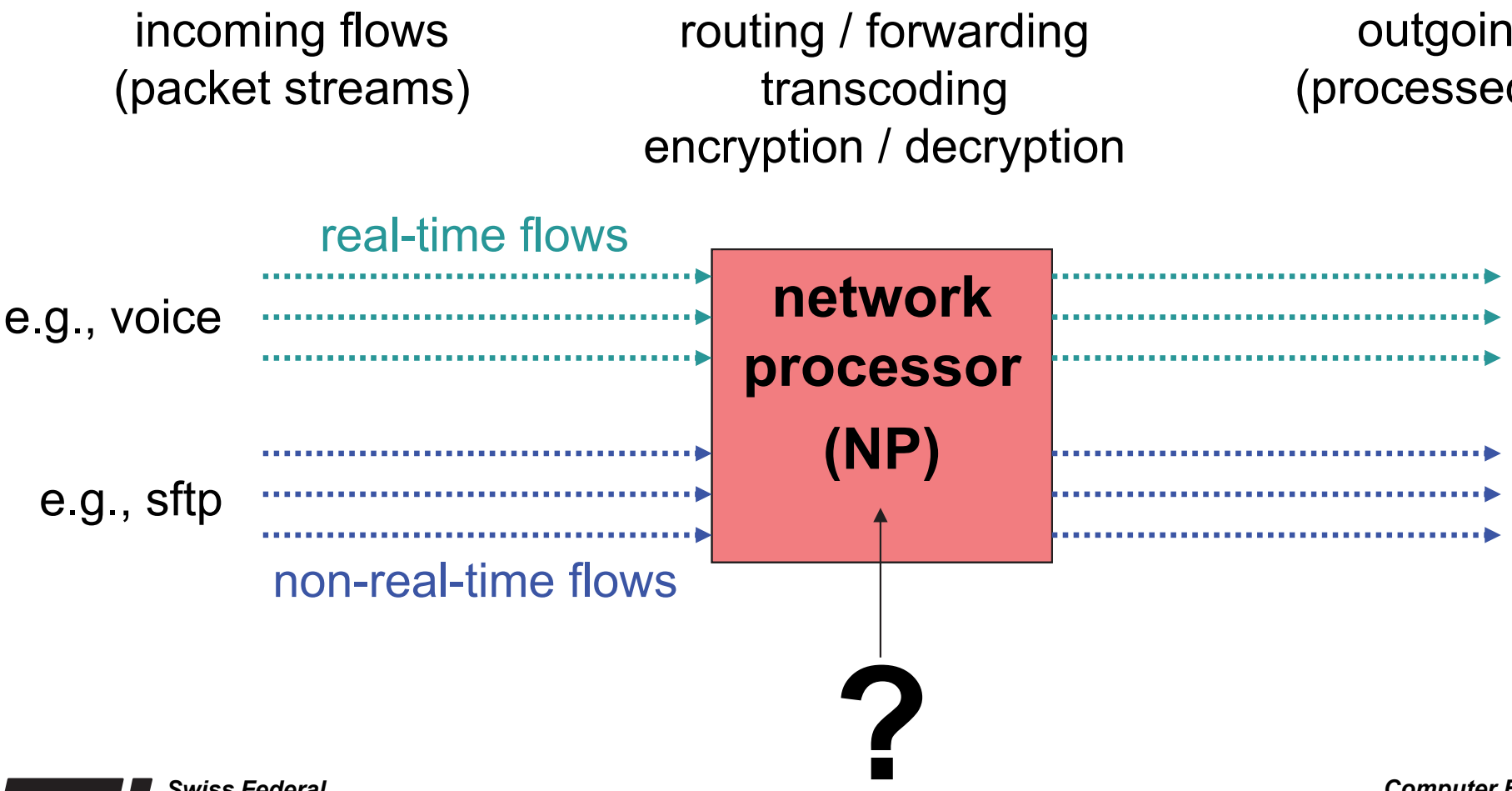


Example: Packet Processing in Networks



Network Processors

Network processor = high-performance, programmable device designed to efficiently execute communication workloads



Optimization Scenario: Overview

Given:

❶ specification of the task structure
(**task model**) = tasks to be executed for each flow

❷ different usage scenarios
(**flow model**) = sets of flows

Sought: network processor implementation =
architecture + task mapping + scheduling

Objectives:

❶ maximize performance

❷ minimize cost

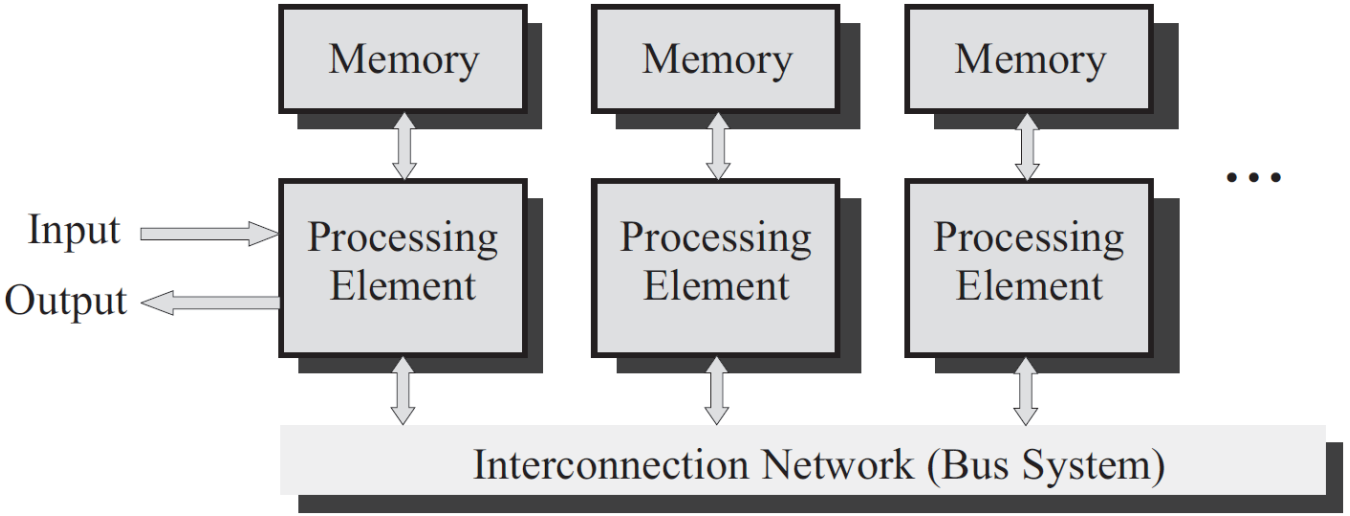
Subject to:

❶ memory constraint

❷ delay constraints

} (**performance model**)

Network Processor Architecture

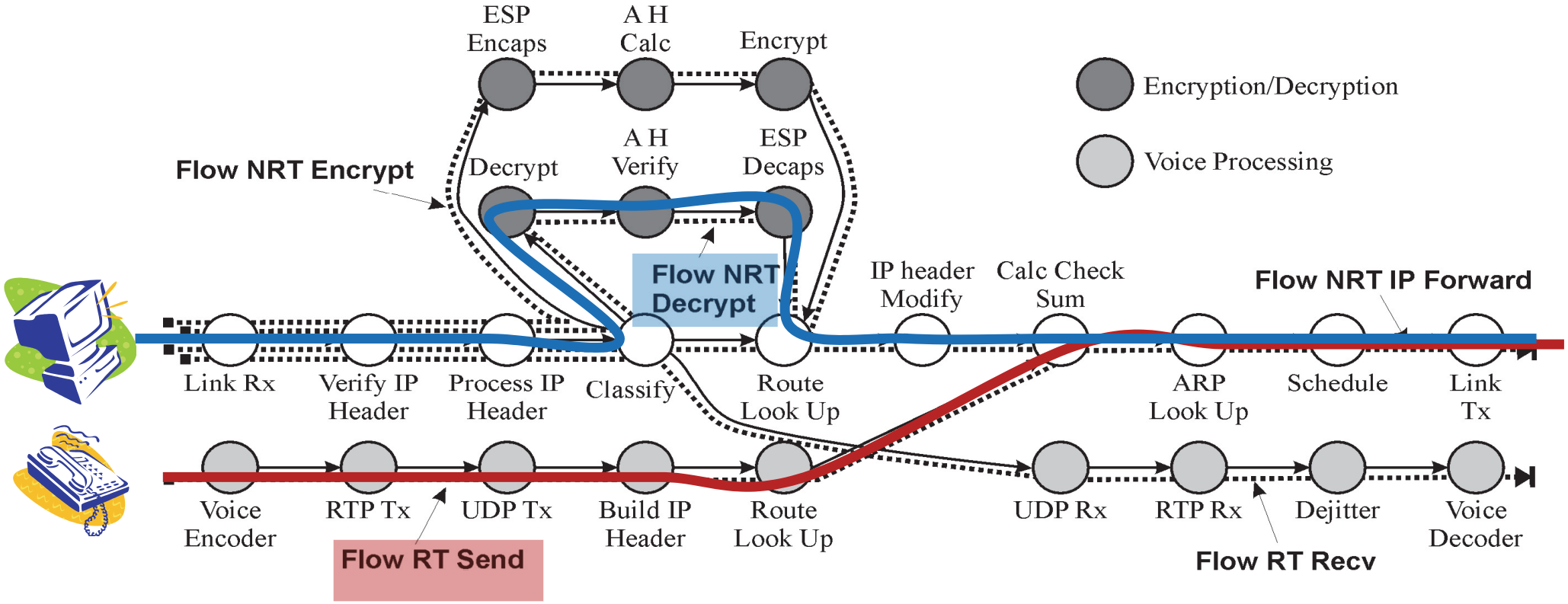


Network processor heterogeneous hardware/software architecture:

- available processing units
  - … are described in *resource set* `R = {ARM9, PowerPC, DSP, MEngine, Classifier, Cipher, LookUp, CheckSum}`
  - … have a *relative implementation cost* `cost(r)≥0, r∈R`
  - *...* and are *selected for a specific architecture* during the allocation step
    - with `alloc(r)=1` if a resource is selected and `0` otherwise

(*) ***Note:*** example from Simon Künzli: *Efficient Design Space Exploration for Embedded Systems*, Shaker Verlag, ISBN 3-8322-5246-0, 2006.

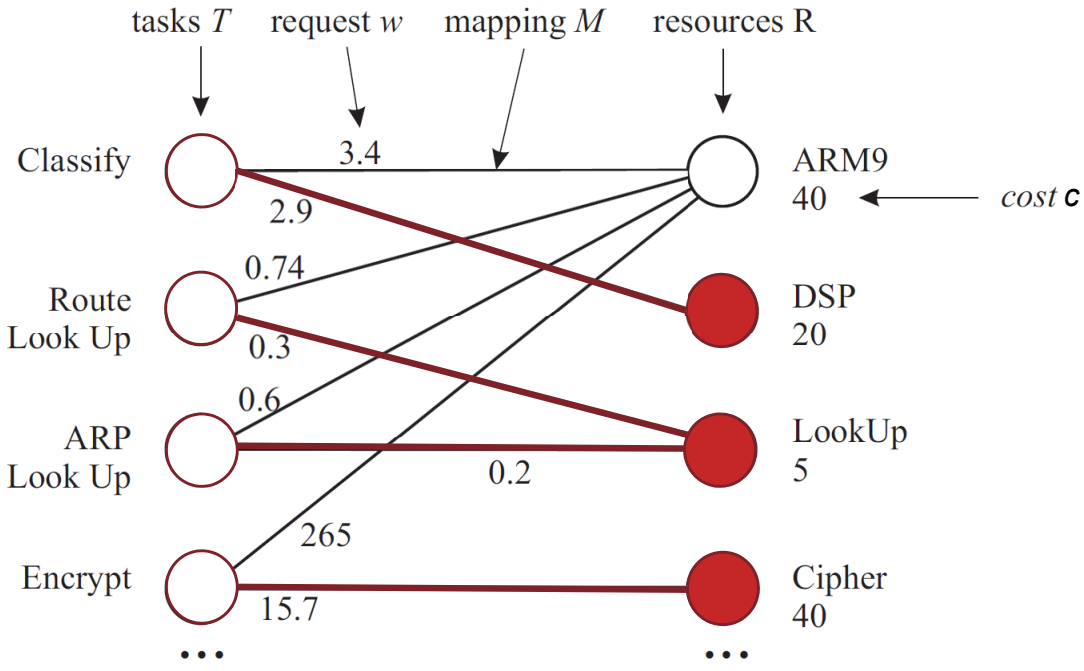
Network Processor Task Model



- *application structure:* set of streams $s \in S$ and set of tasks $t \in T$
  - each stream includes an ordered sequence of tasks $V(s) = [t_0, \ldots, t_n]$
- *example:*
  $S = \{RTSend, NRTDecrypt, NRTEncrypt, RTRecv, NRTForward\}$

Problem: Optimal Design of Network Processor

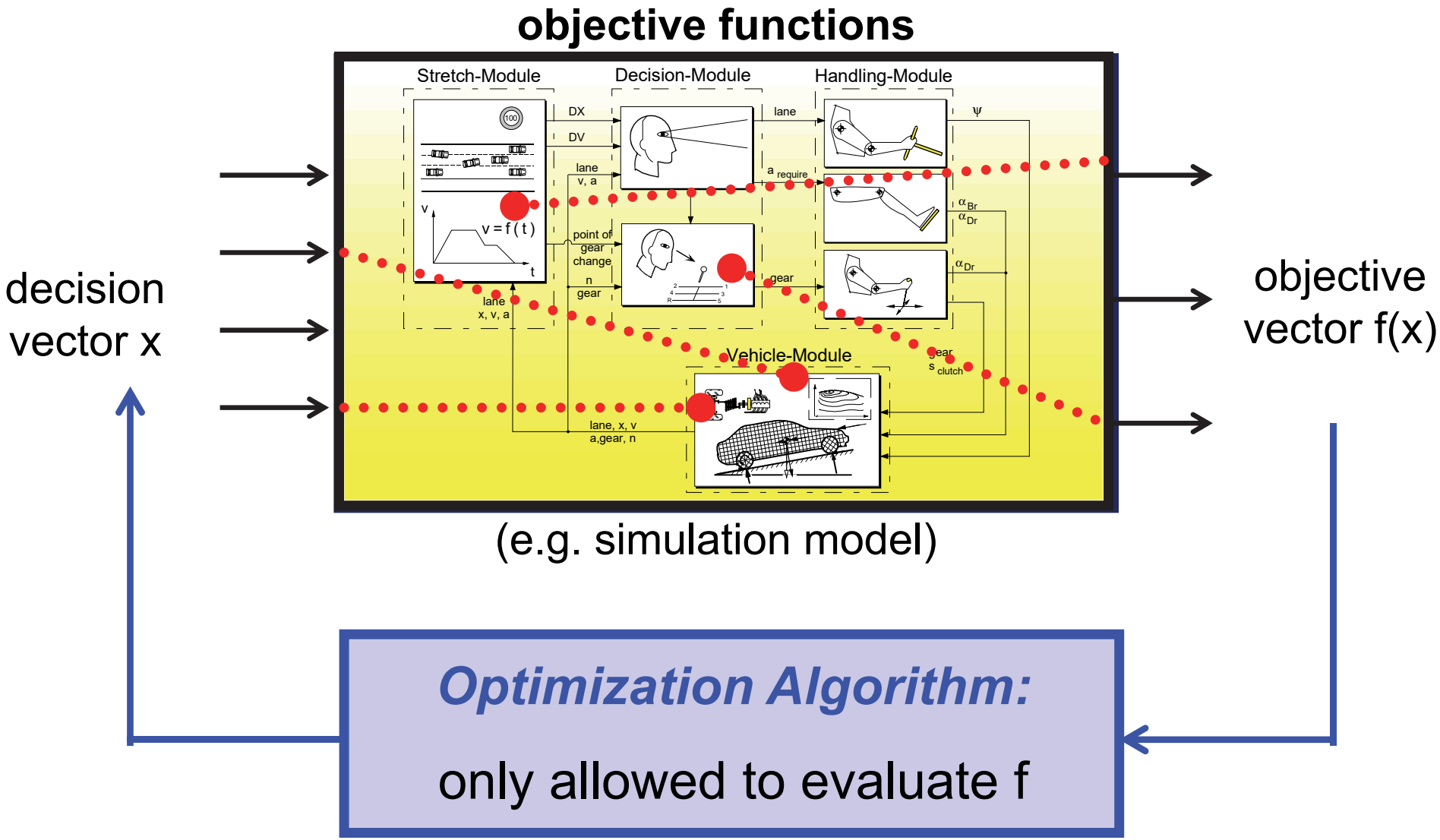
- mappings $M \subseteq T \times R$: all possible bindings of tasks
  - i.e., if $(t,r) \in M$, then task t could be executed on resource r
- request $w(r,t) \geq 0$
  - i.e., execution of one packet in t would use w computing units of r
- resource allocation cost $c(r) \geq 0$



➔ binding Z of tasks to resources $Z \subseteq M$ *(leading to actual implementation)*

- subset of mappings M s.t. every task $t \in T$ is bound to *exactly one allocated resource* $r \in R$ and $alloc(r) = 1$ and $r = bind(t)$

Method: Black-Box Optimization



Lecture Synopsis

- Introduction
- ***Multiobjective Optimization***
- Multiobjective Evolutionary Algorithms
- Implementation Aspects

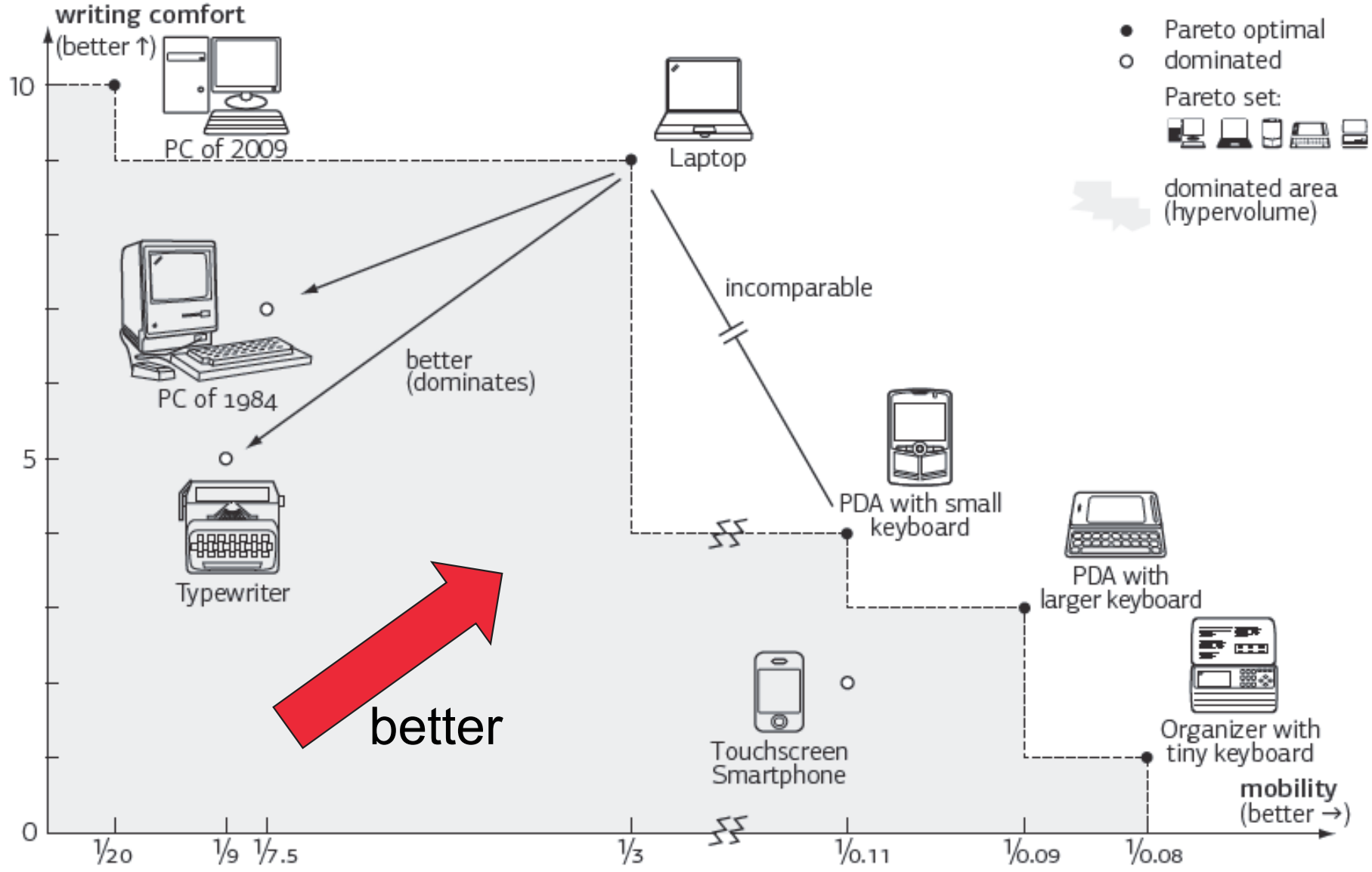
Multiobjective Optimization

example from Johannes Bader

- Let us suppose, we would like to select a typewriting device. Criteria are
  - mobility (related to weight)
  - comfort (related to keyboard size and performance)

Icon	Device	weight (kg)	comfort rating
	PC of 2009	20.00	10
	PC of 1984	7.50	7
	Laptop	3.00	9
	Typewriter	9.00	5
	Touchscreen Smartphone	0.11	2
	PDA with large keyboard	0.09	3
	PDA with small keyboard	0.11	4
	Organizer with tiny keyboard	0.08	1

Multiobjective Optimization



Basic Definitions

- We intend to minimize a vector-valued ***objective function***

$$f = (f_1; \ldots ; f_n) : X \rightarrow \mathbf{R}^n$$

- X denotes the ***decision space***, i.e., the feasible set of alternatives for the optimization problem
- The image of the decision space X using the objective function f is denoted as the ***objective space*** $Z \subset \mathbf{R}^n$ with

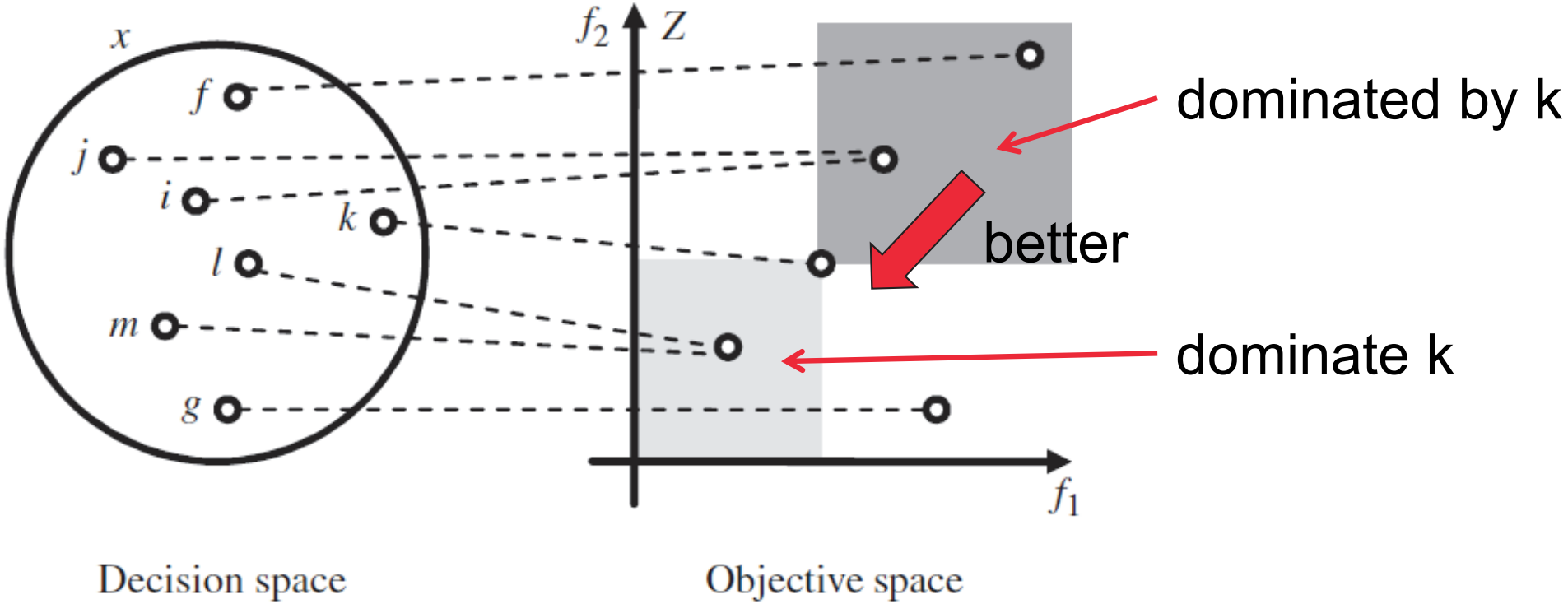
$$Z = \{ f(x) \mid x \in X \}$$

- A single alternative $x \in X$ is sometimes named 'solution' and the corresponding objective value $z = f(x) \in Z$ is named 'objective vector'.

Basic question: How do we define the minimum of a vector-valued function?

Pareto-Dominance

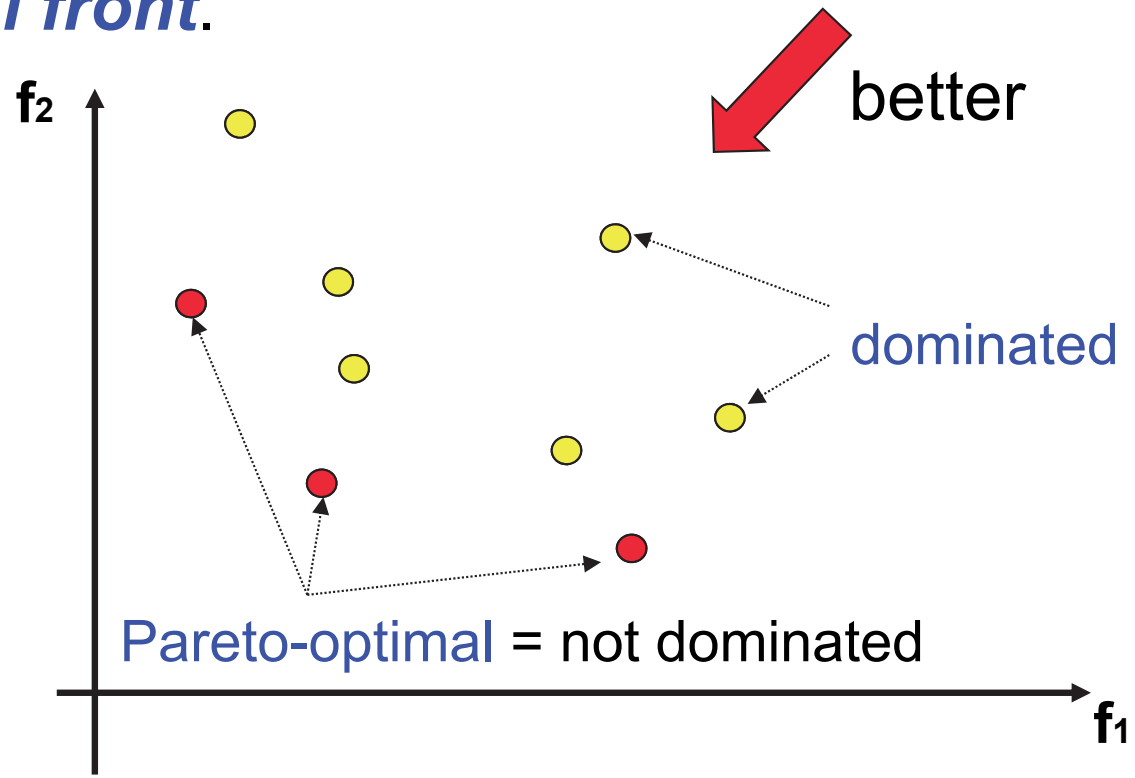
Definition : *A solution* $a \in \mathsf{X}$ *weakly Pareto-dominates a solution* $b \in \mathsf{X}$*, denoted as* $a \preceq b$*, if it is as least as good in all objectives, i.e.,* $f_i(a) \leq f_i(b)$ *for all* $1 \leq i \leq n$*. Solution* a *is better then* b*, denoted as* $a \prec b$*, iff* $(a \preceq b) \wedge (b \npreceq a)$*.*



Pareto-optimal Set

- A solution is named ***Pareto-optimal***, if it is not Pareto-dominated by any other solution in X.
- The set of all Pareto-optimal solutions is denoted as the Pareto-optimal set and its image in objective space as the ***Pareto-optimal front***.

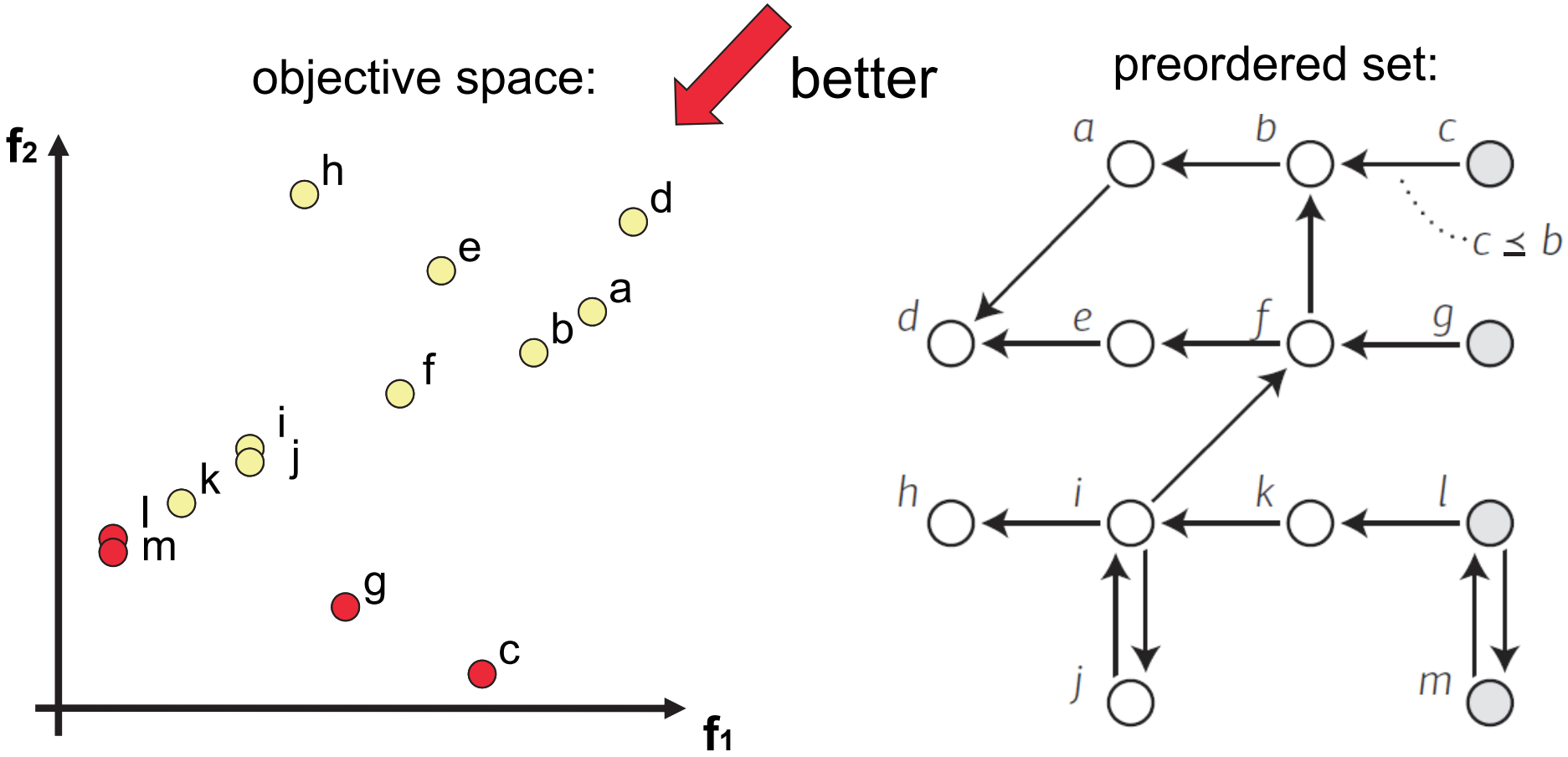
objective space Z:



Preorder

reflexivity: $a \preceq a$
transitivity: if $a \preceq b$ and $b \preceq c$, then $a \preceq c$

- The domination relation imposes a ***preorder*** on all design points: We are faced with a set of optimal solutions.



Optimization Alternatives

- Use of ***classical single objective optimization*** methods
  - simulated annealing, tabu search
  - integer linear program
  - other constructive or iterative heuristic methods
- Decision making is done before the optimization.

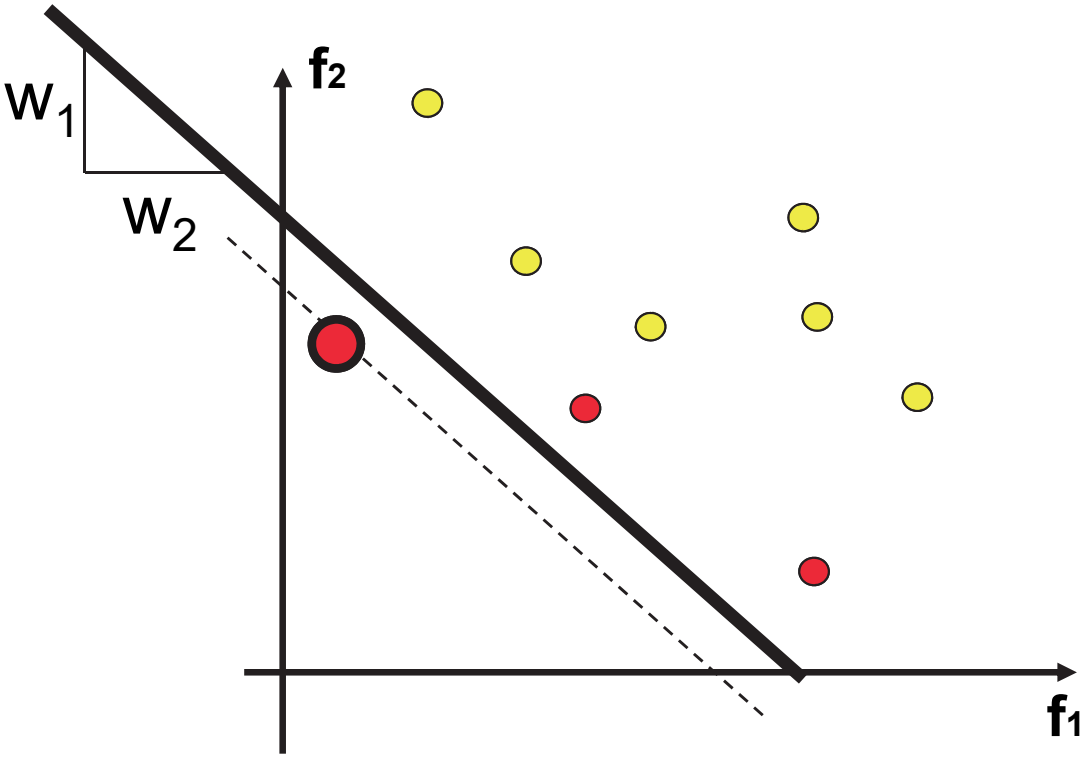
or

- ***Population based optimization methods***
  - evolutionary algorithms / genetic algorithms
- Decision making is done after the optimization.

Single Objective Problem

- There are many possibilities to reduce a multi-objective problem to a single-objective problem.
- ***Example 1***: Aggregation (weighted sum)

$$\text{minimize } f = w_1 f_1 + w_2 f_2 + \ldots. w_n f_n$$



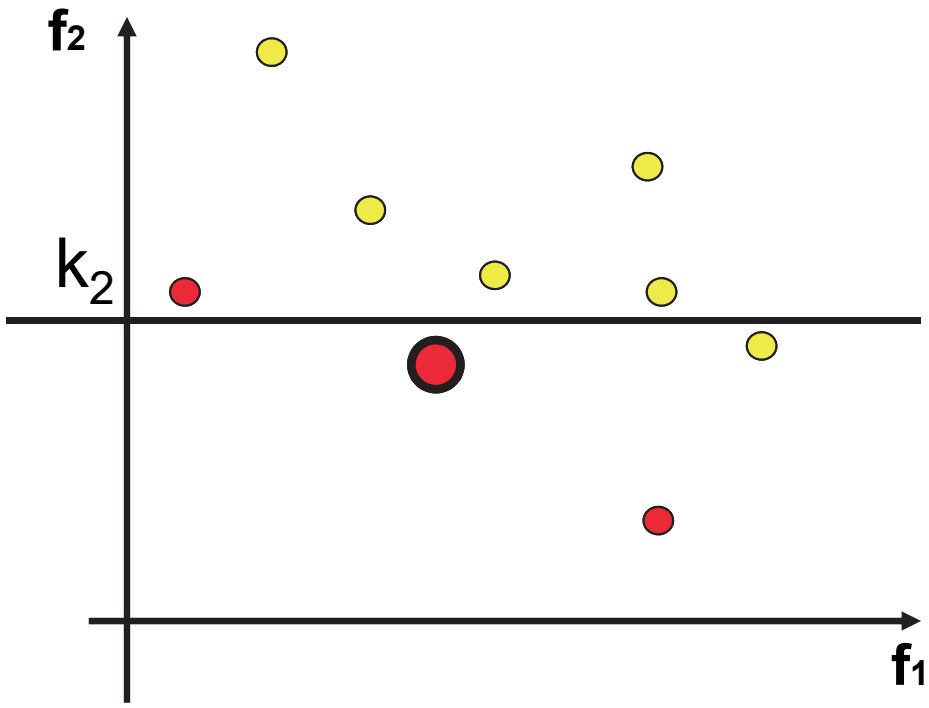
weights need to be fixed before optimization

not all Pareto points can be found

Single Objective Problem

- ***Example 2***: Single objective with constraints

$$\text{minimize } f = f_1 \text{ with } f_2 < k_2 \ \ldots \ f_n < k_n$$



bounds need to be fixed before optimization

all Pareto points can be found by 'scanning' through all bounds

Population-based Optimization

- In population-based methods, a whole ***set of solutions*** is investigated simultaneously.
- ***Evolutionary algorithms*** are black-box optimization methods which are randomized and population-based.

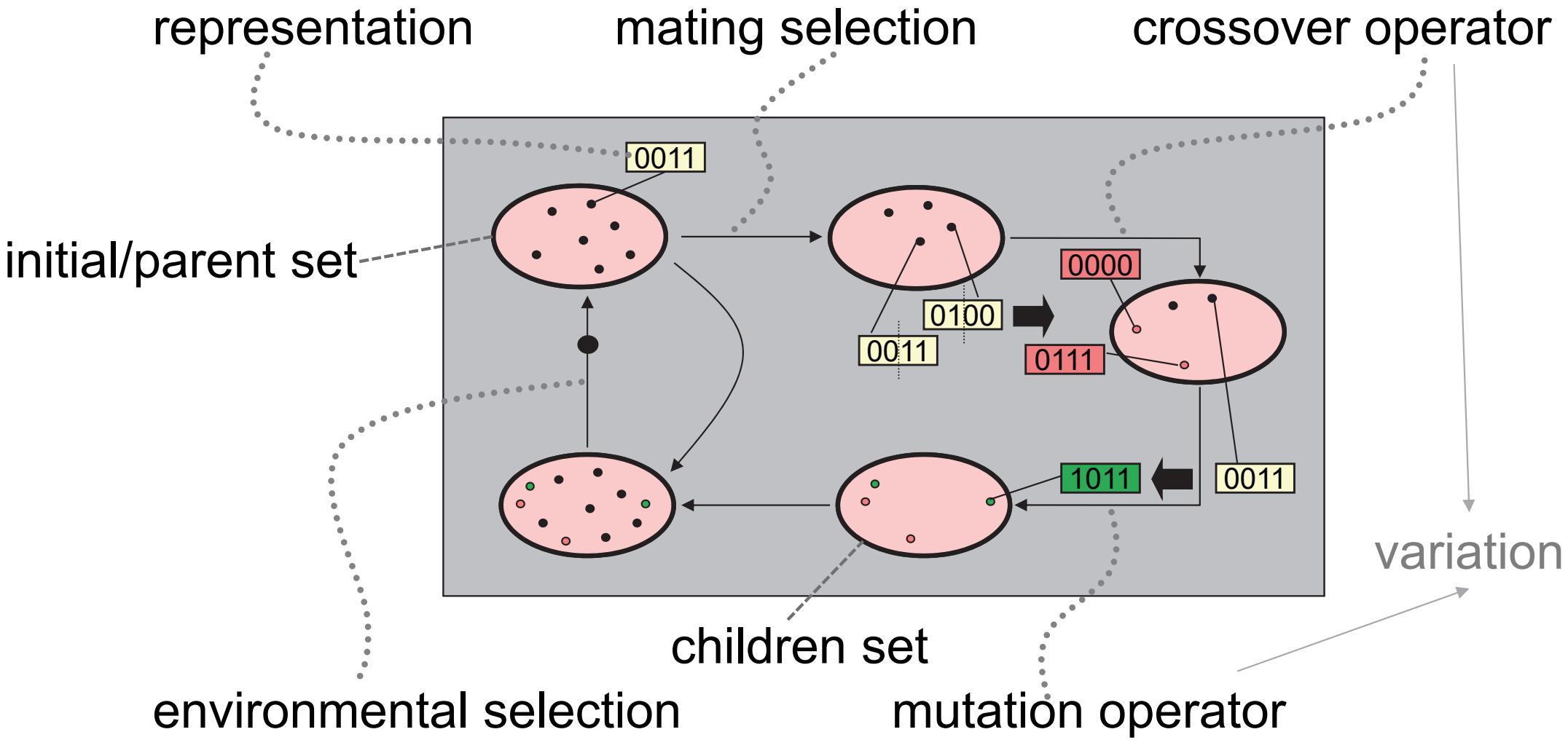
- Like other related methods, e.g. simulated annealing, they are based on the assumption that better solutions are preferably found in the ***neighborhood*** of good solutions.
- ***Local minima*** are avoided using
  - randomization, for example in the neighborhood operator or in the selection of better solutions
  - local and global neighborhood operators

Evolutionary Algorithm Cycle

Many variants of the following scheme exist.

1. A set of ***initial solutions*** (initial population) is chosen, usually at random. This set is named ‘parent’ set.
2. Solutions from the ‘parent’ set are selected (***mating selection***) as a basis for step 3.
3. ***Variation:*** Solutions from step 2 are changed using neighborhood operators, e.g. ***cross-over*** operators or ***mutation*** operators. The resulting set is named ‘children’ set.
4. Determine union of the ‘parent’ set and the ‘children’ set.
5. Solutions of the set from 4. are selected based on their merit to construct the new ‘parent’ set (***environmental selection***).

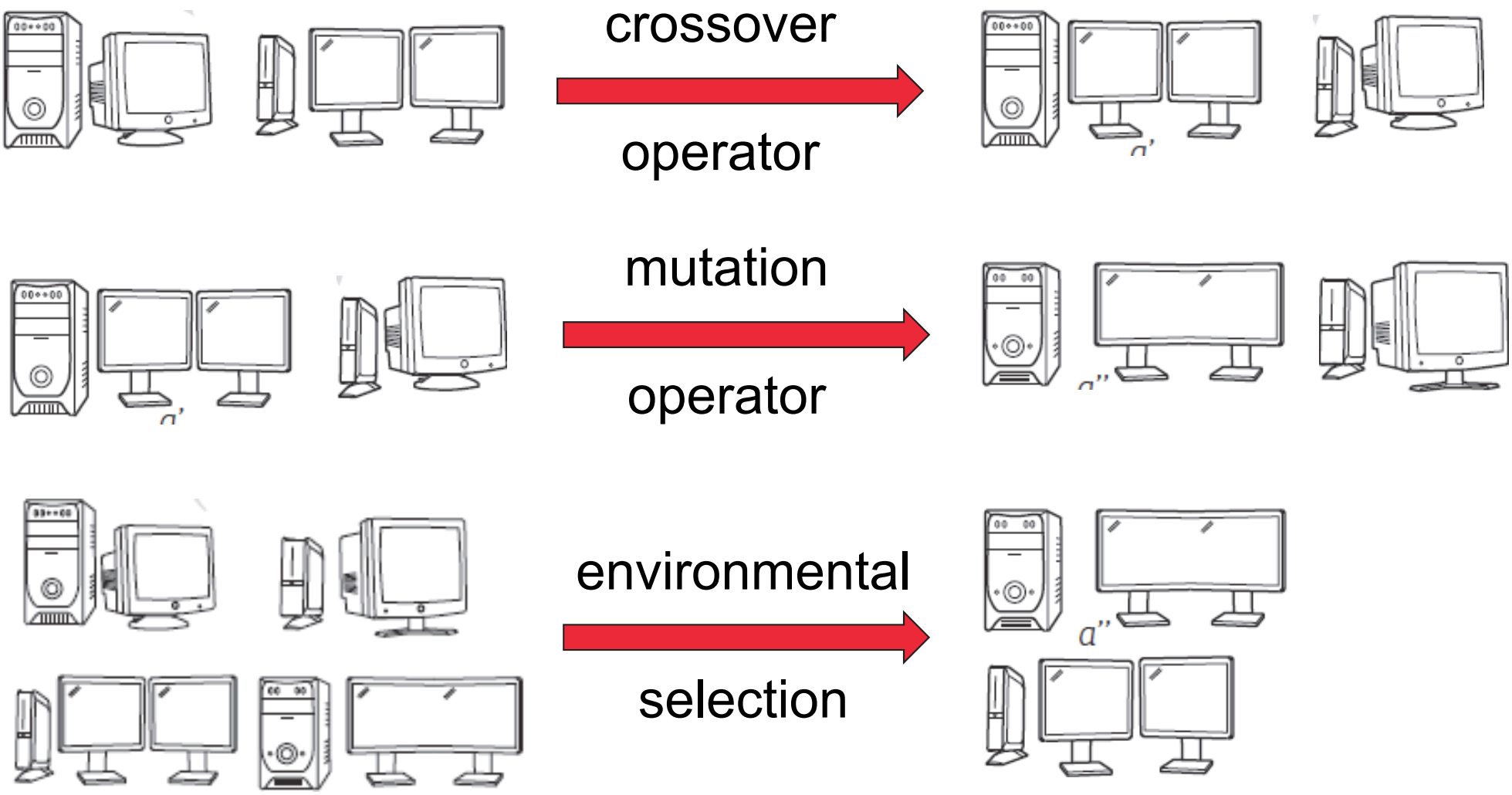
Evolutionary Algorithm Cycle



Evolutionary Algorithm Cycle

example from Johannes Bader

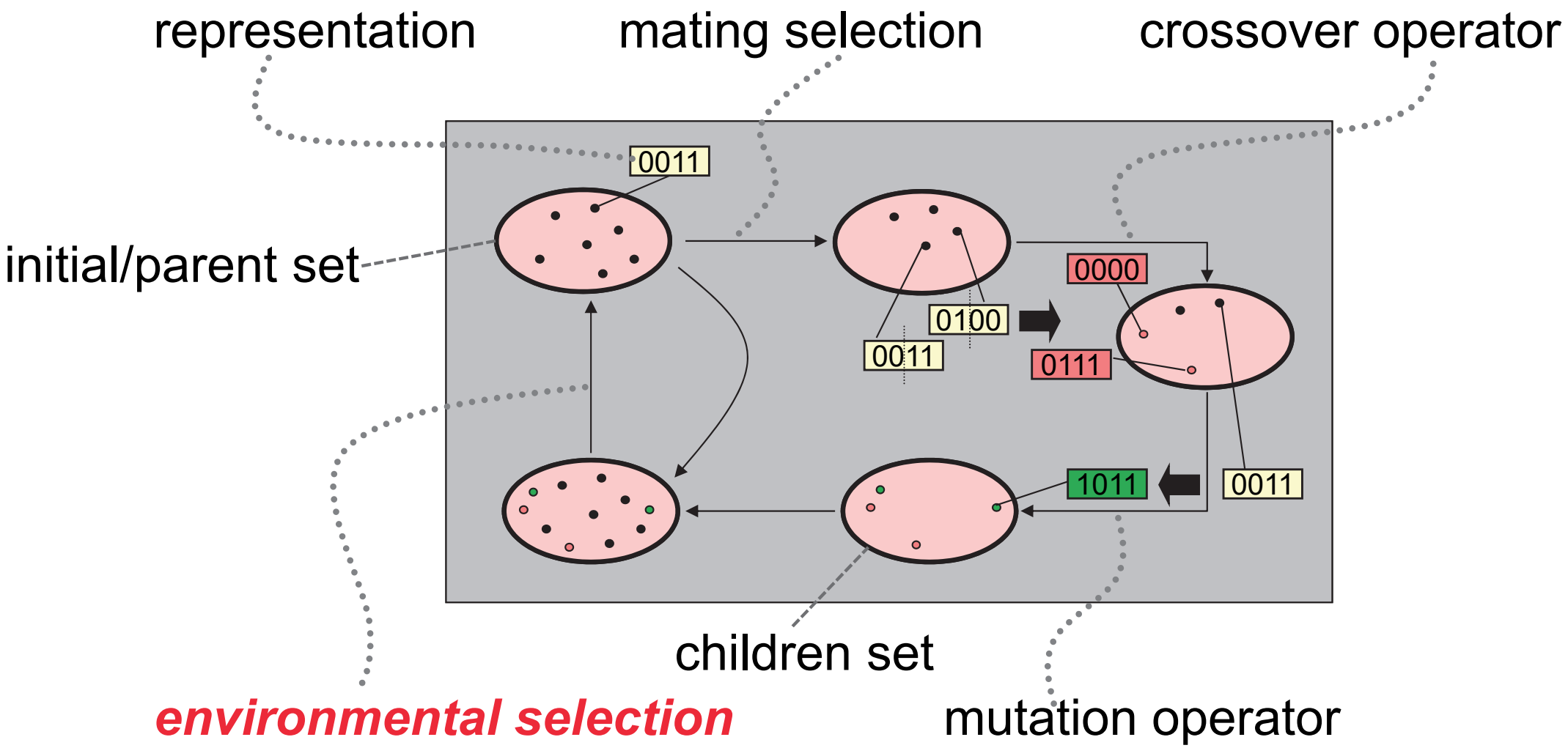
Some funny examples:



Lecture Synopsis

- Introduction
- Multiobjective Optimization
- ***Multiobjective Evolutionary Algorithm***
  - ***Environmental Selection***
  - Neighborhood Operator
- Implementation Aspects

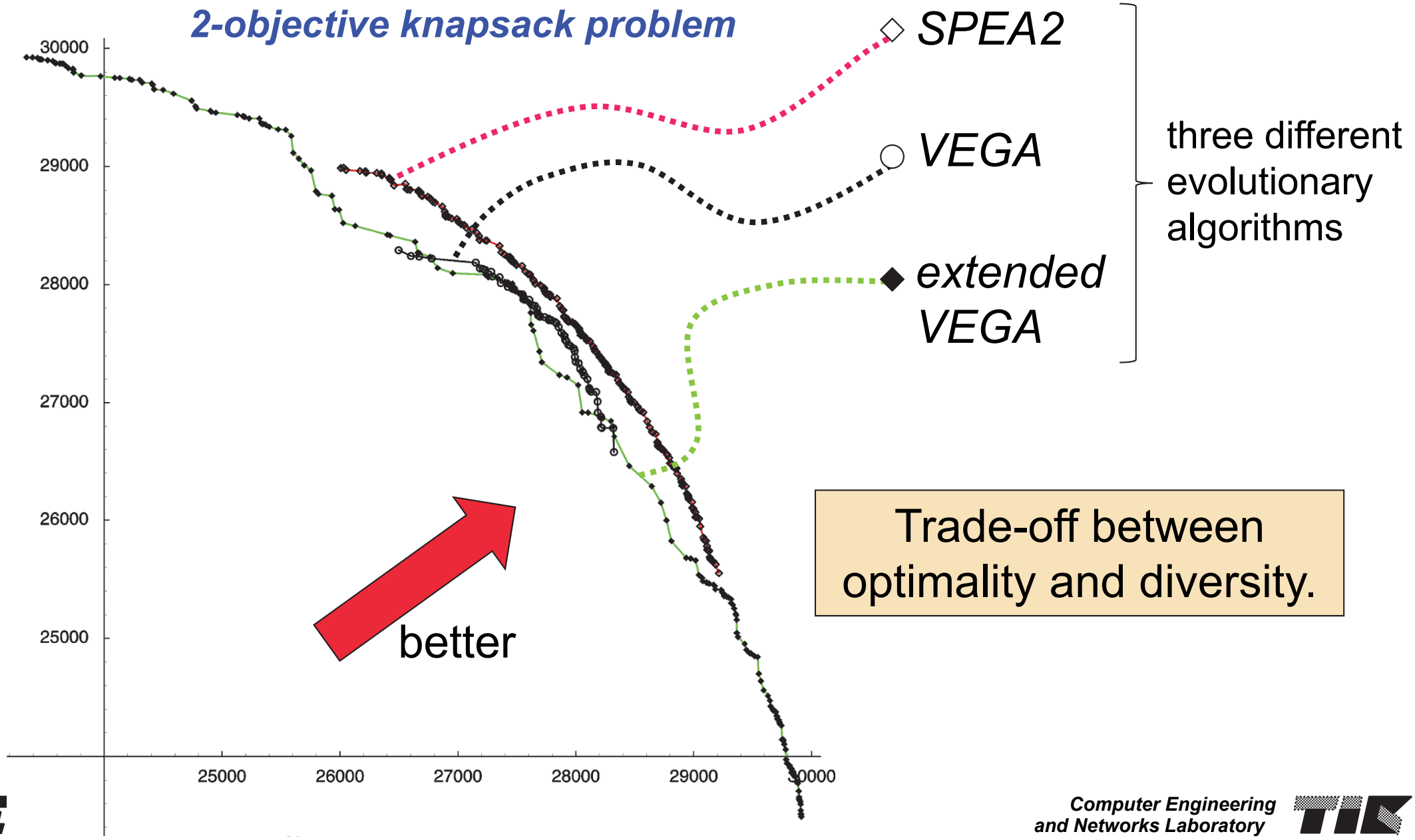
Evolutionary Algorithm Cycle



Environmental Selection

- How do we choose solutions that should be removed from the population?
- ***Informal criteria:***
  - solutions should be ‘close’ to the (unknown) Pareto-optimal front (optimality)
  - solutions should cover large parts of the objective space (diversity)
- ***Principle idea:***
  - We are optimizing sets. Therefore, we need to define an indicator that characterizes the optimality of the whole set.
  - We chose the optimal subset of solutions with respect to this set-indicator.

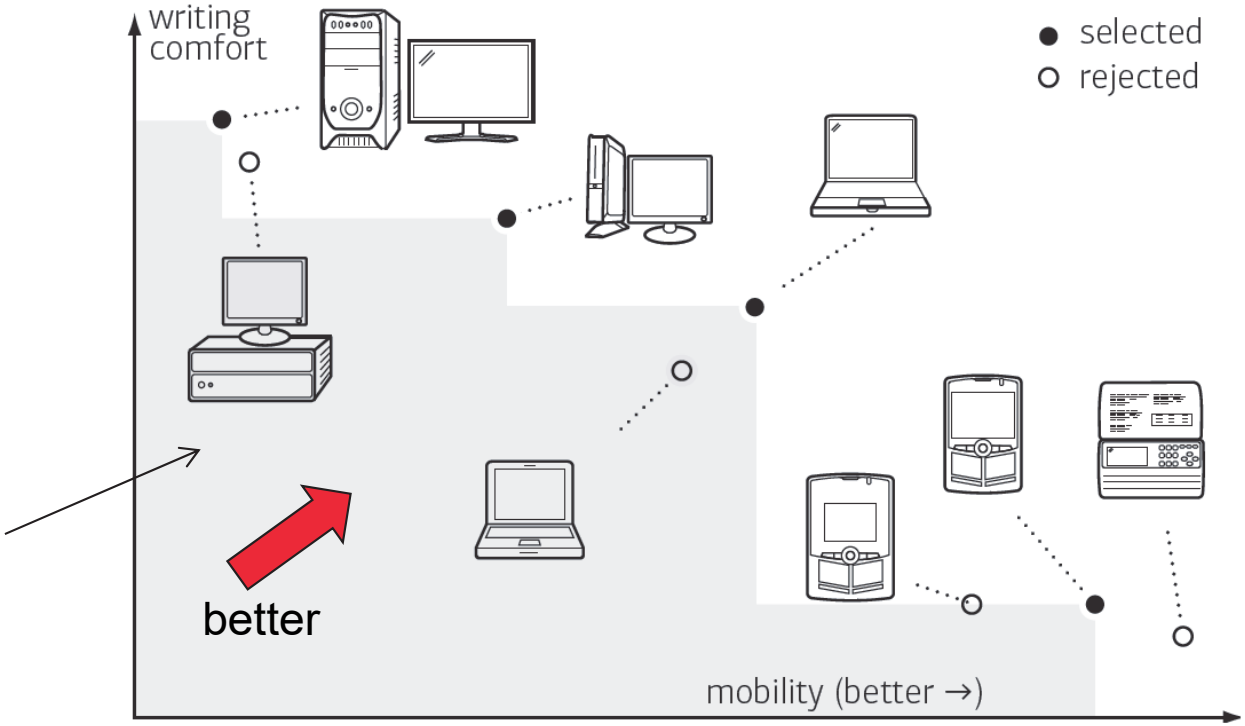
Optimality and Diversity



The Hypervolume Indicator

- ***Environmental selection:*** Select subset of solutions that ***maximizes hypervolume indicator***.
- Example: select optimal subset of 4 solutions from the 8 solutions in the set.

hypervolume indicator:
hypervolume of
the dominated subspace



The Hypervolume Indicator

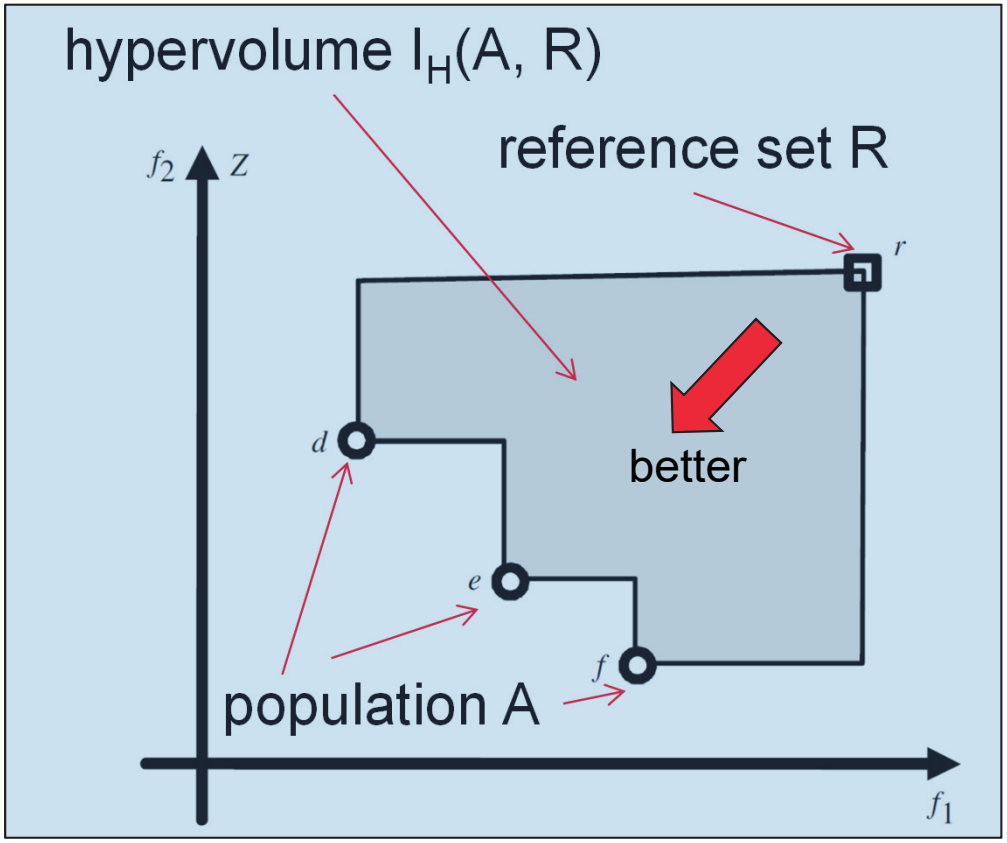
(For the following, we again minimize f.)

Given a set of solutions $A \subseteq X$ and a set of reference points $R \subset \mathbf{R}^n$. Then the hypervolume indicator $I_H(A, R)$ of A with respect to R is defined as

$$I_H(A, R) = \int_{z \in H(A,R)} dz$$

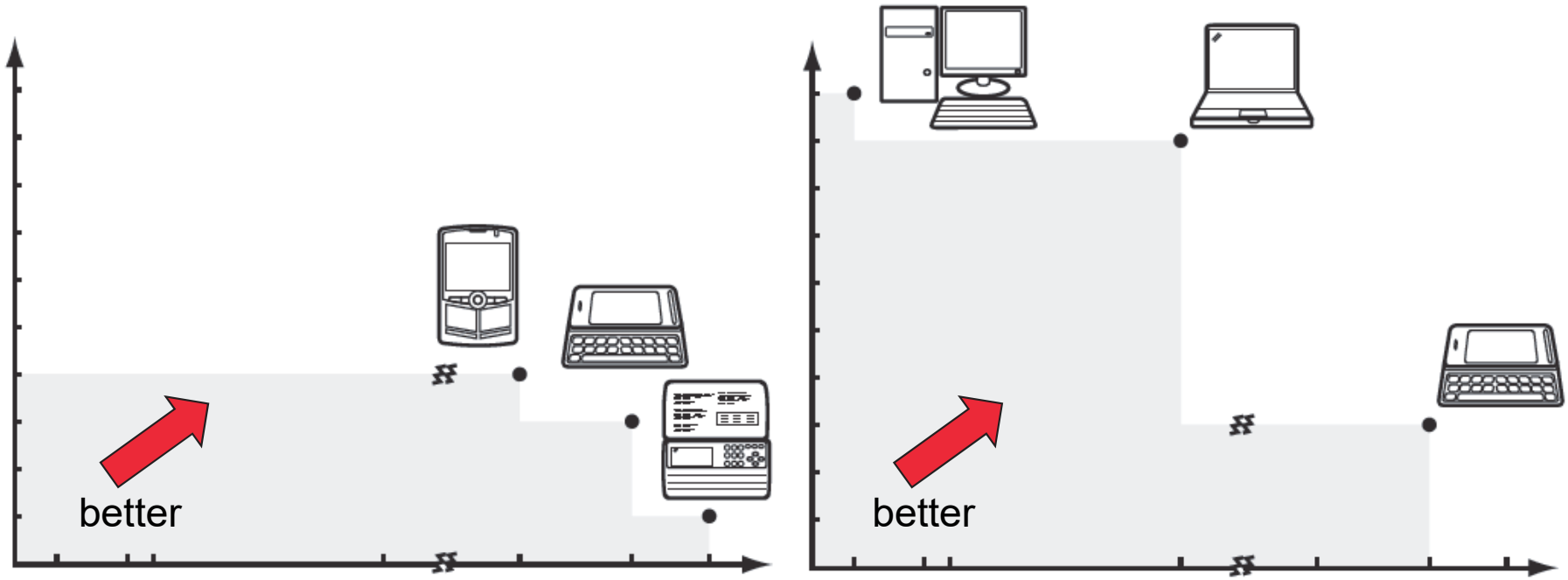
where $H(A, R)$ is the dominated space of A regarding R:

$$H(A, R) = \{z \in \mathbf{R}^n \mid \exists a \in A : \exists r \in R : (f(a) \leq z \leq r)\}$$



Hypervolume Indicator

- ***Why does the hypervolume indicator lead to diversity and optimality?***
- ***Diversity:*** It appears that the indicator well covers the intuitive notion of diversity in objective space.



Hypervolume Indicator

- ***Optimality:*** One can show the following relation between the hypervolume indicator and Pareto-dominance:
  - Suppose that a set of solutions ***A is better than a set B*** $(A \prec B)$, i.e. every solution of B is weakly Pareto-dominated by at least one solution *in* A $(A \preccurlyeq B)$, *but we do not have* $(B \preccurlyeq A)$.
  - Then, the ***hypervolume of A is larger than that of B***.

In other words, if I have a set of solutions with the largest possible hypervolume, then there is no other set that dominates it completely.

Therefore, it makes sense to ***determine a set of solutions that maximizes the hypervolume indicator***.

Template of a Practical Algorithm

Algorithm 1 Main Loop

1: generate initial set of solutions P of size m, i.e., randomly choose m solutions
2: **while** termination criterion not fulfilled **do**
3: $\quad P' \leftarrow heuristicSetMutation(P)$
4: $\quad$ **if** $I(P') \geq I(P)$ **then**
5: $\quad\quad P \leftarrow P'$
6: **return** P

hypervolume indicator

combines mating selection, crossover, mutation and environmental selection

The algorithm can be seen as a simple Greedy strategy: if the new population is not worse than the old one, use it in the next iteration

Template of a Practical Algorithm

Algorithm 2 Heuristic Set Mutation

```
1: procedure heuristicSetMutation(P)
2:     generate k solutions r_1, ..., r_k ∈ X based on P
3:     P' ← P ∪ {r_1, ..., r_k}
4:     while |P'| > m do
5:         for all a ∈ P' do
6:             δ_a ← I(P') − I(P' \ {a})
7:         choose p ∈ P' with δ_p = min_{a∈P'} δ_a
8:         P' ← P' \ {p}
9:     return P'
```

combines mating selection, crossover and mutation

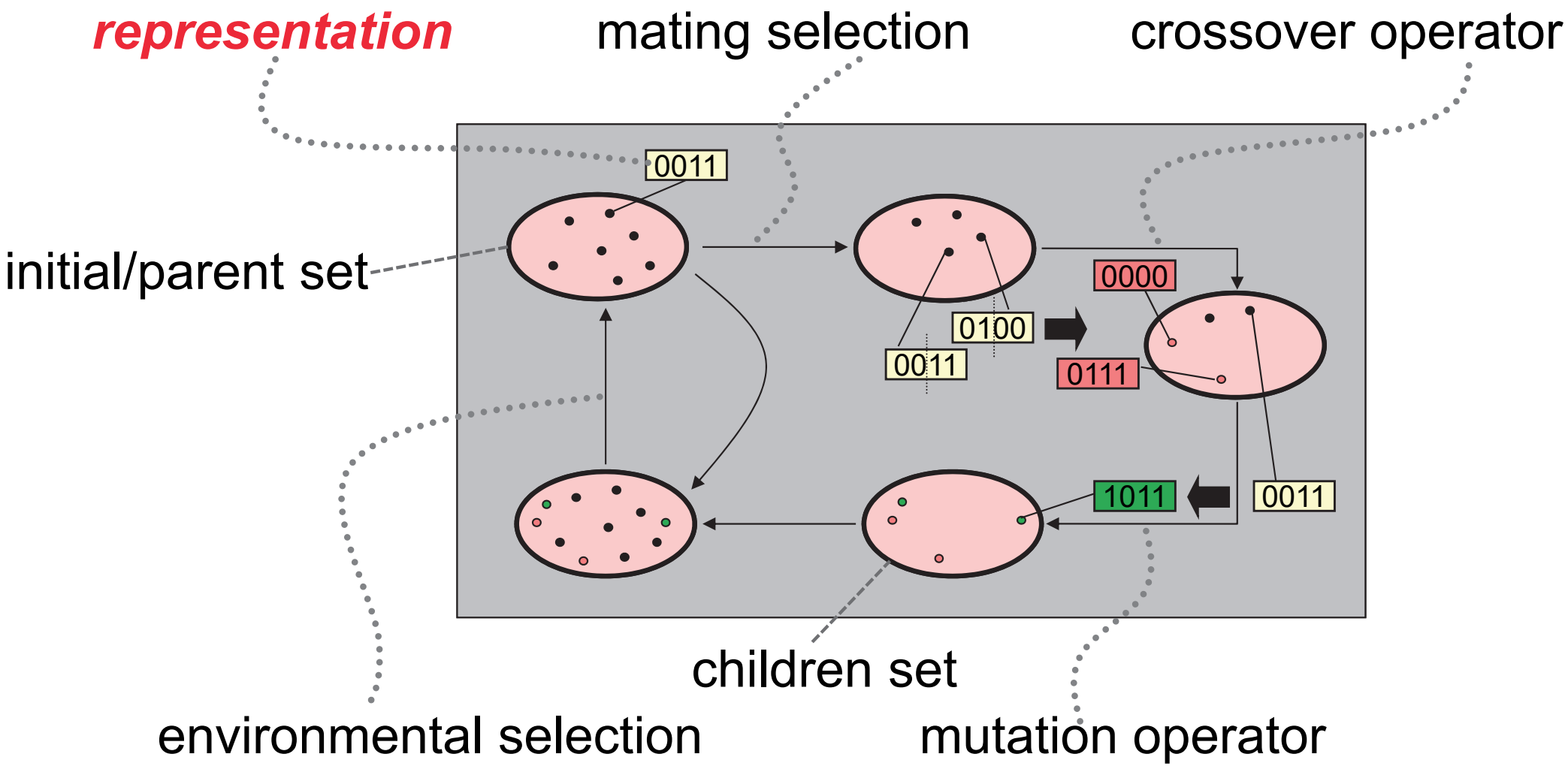
union of parent and children population

This is a ***heuristic to select the best m solutions*** in P': Solutions are removed from P' one-by-one; in each iteration, the solution with the smallest loss in hypervolume is removed.

Lecture Synopsis

- Introduction
- Multiobjective Optimization
- ***Multiobjective Evolutionary Algorithm***
  - Environmental Selection
  - ***Neighborhood Operator***
- Implementation Aspects

Evolutionary Algorithm Cycle



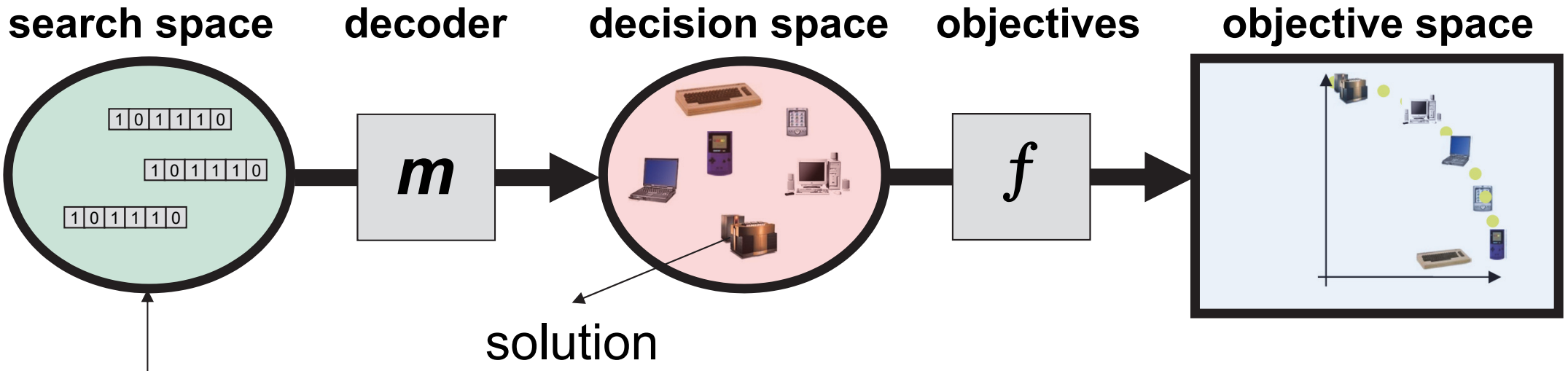
Representation and Neighborhood

- Usually, neighborhood operators such as crossover and mutation are based on a ***representation of solutions***.
- A representation corresponds to an ***abstract data structure*** that encodes a solution.
- ***Neighborhood operators*** work on representations.

- ***Simple example***:
  - Decision space: All partitionings of n objects into m blocks.
  - Representation of single solution using a vector of length n with elements in [1, m]. Example for n=6 and m=3:

1	2	3	4	5	6
2	1	1	1	3	2

Representation



solutions encoded by vectors, matrices, trees, lists, ...

Issues:

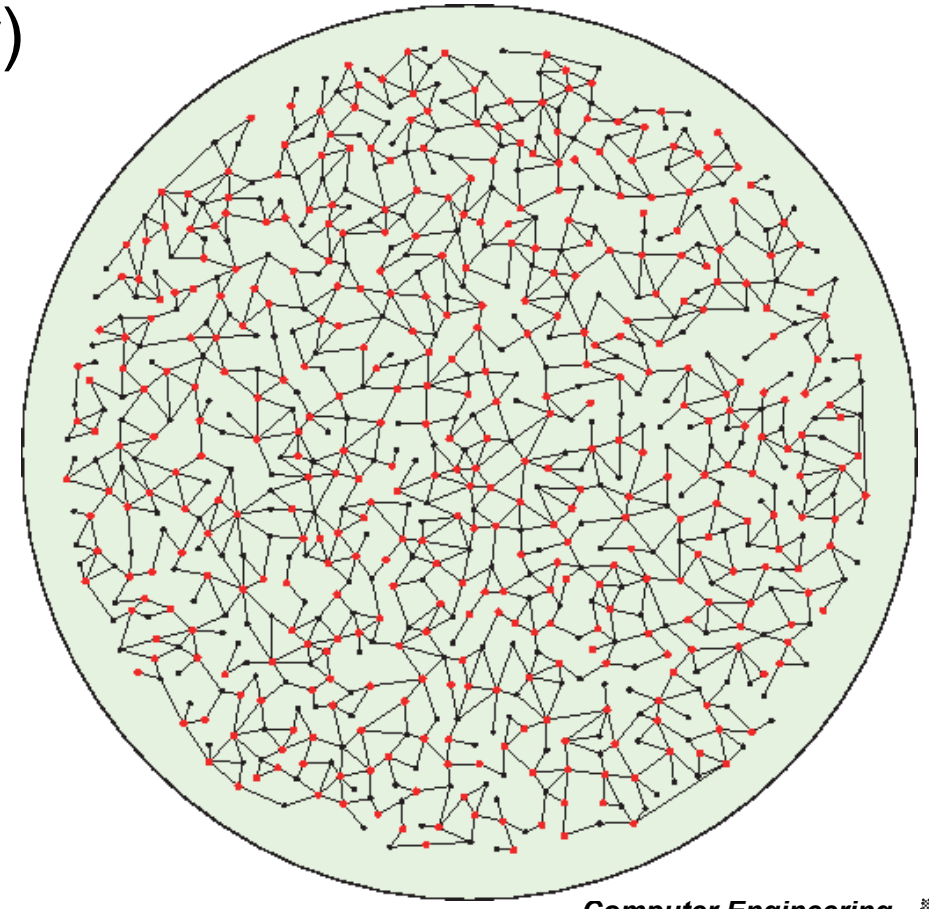
- completeness (each solution has an encoding)
- uniformity (all solutions are represented equally often)
- feasibility (each encoding maps to a feasible solution)

Example: Binary Vector Encoding

Given: graph

Goal: find minimum subset of nodes such that each edge is connected to at least one node of this subset (minimum vertex cover)

nodes	A	B	C	D	E	F	…
selected?	1	0	1	1	1	0	…

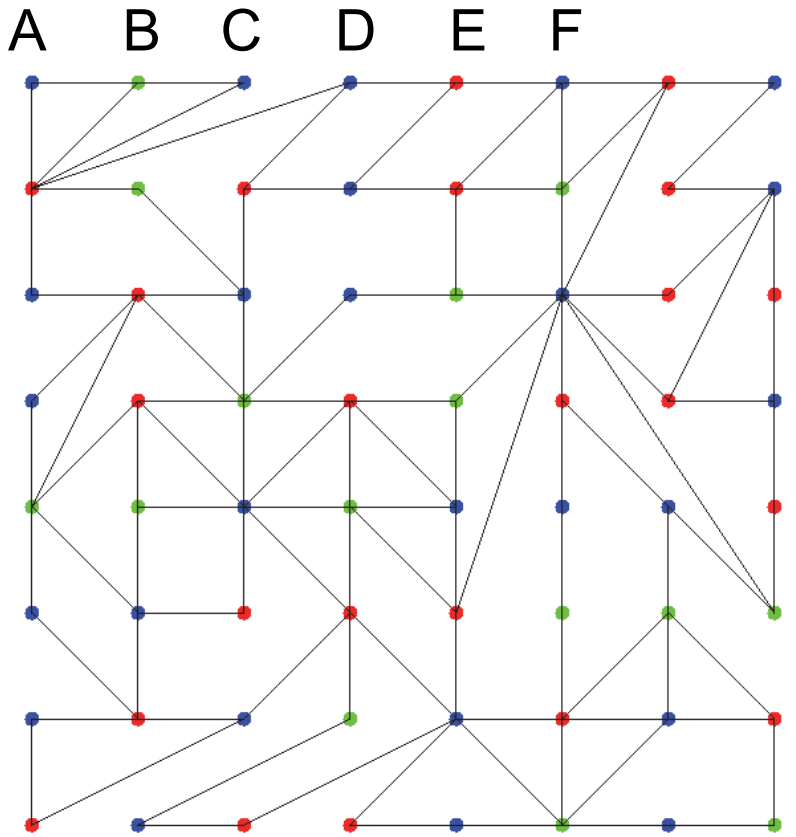


Example: Integer Vector Encoding

Given: graph, k colors

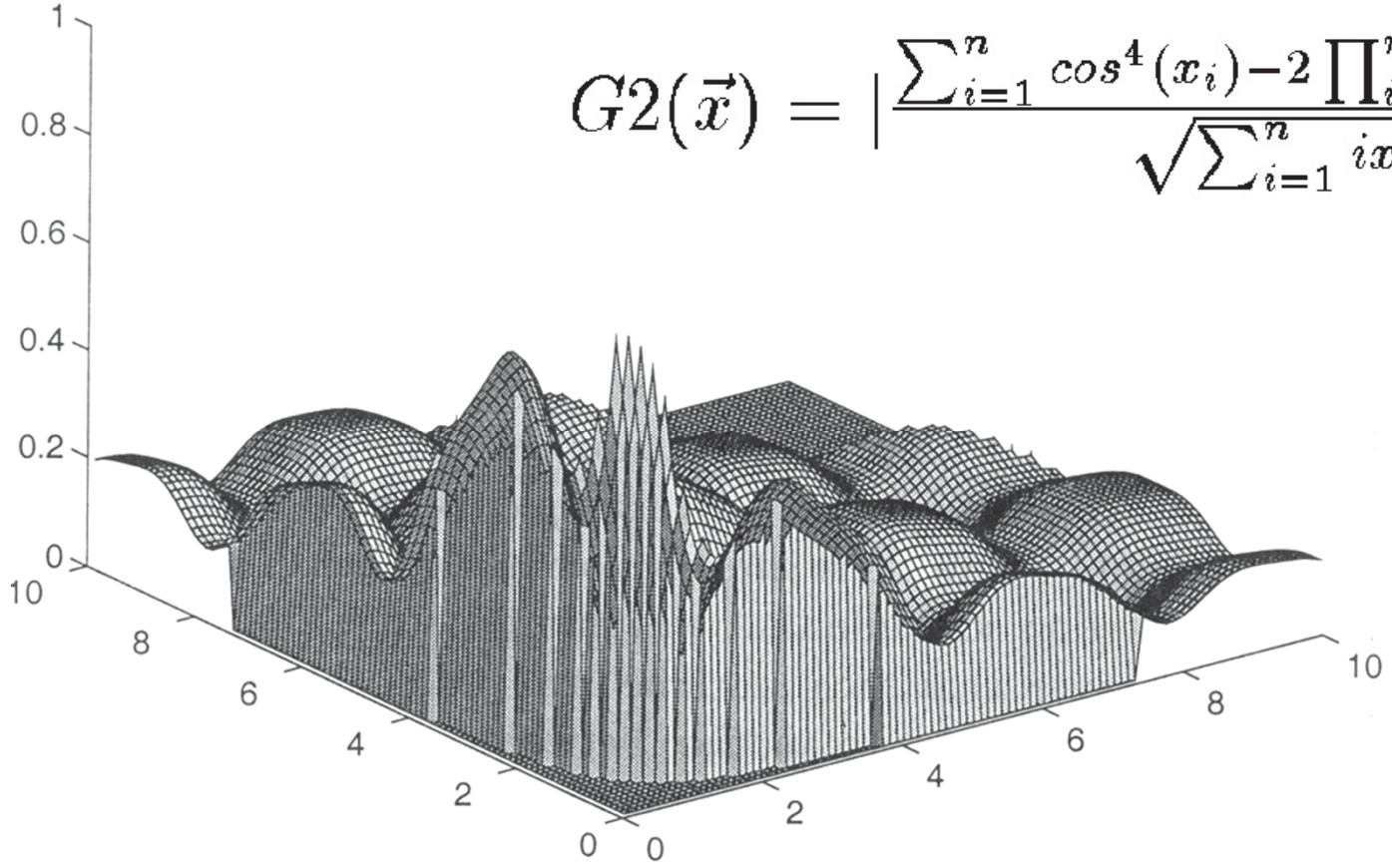
Goal: assign each node one of the k colors such that the number of connected nodes with the same color is minimized (graph coloring problem)

nodes	A	B	C	D	E	F	…
colors	1	2	1	1	3	1	…



Example: Real Vector Encoding

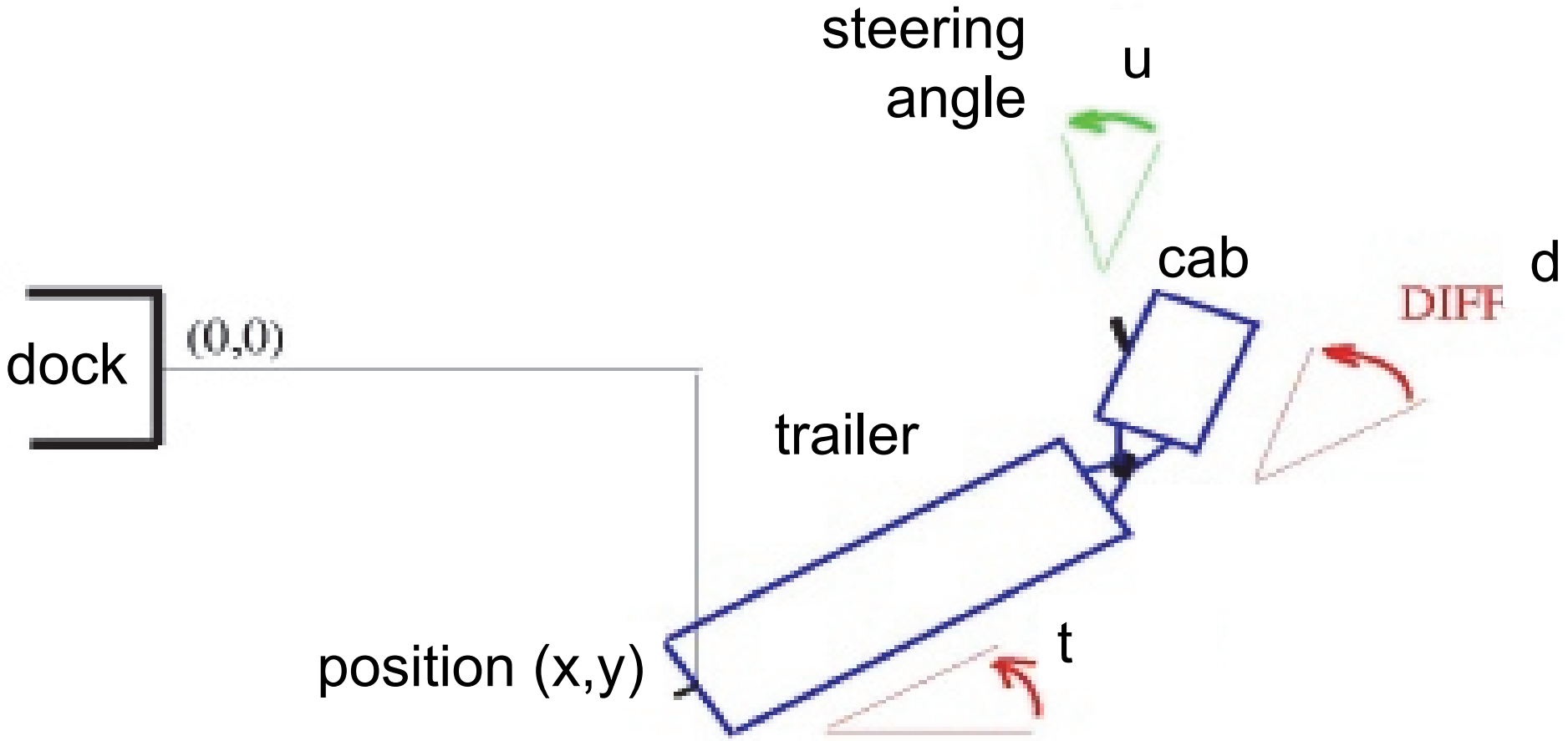
$$G2(\vec{x}) = \left| \frac{\sum_{i=1}^{n} cos^4(x_i) - 2\prod_{i=1}^{n} cos^2(x_i)}{\sqrt{\sum_{i=1}^{n} i x_i^2}} \right|$$



parameters	x_1	x_2	x_3	x_4	…	x_n
values	0.33	0.53	1.03	3.25	…	9.83

[Michalewicz, Fogel: How to Solve it. Springer 2000]

Tree Example: Parking a Truck



Goal:

find function c with

u = c(x, y, d, t)

Search Space for the Truck Problem

Operators:

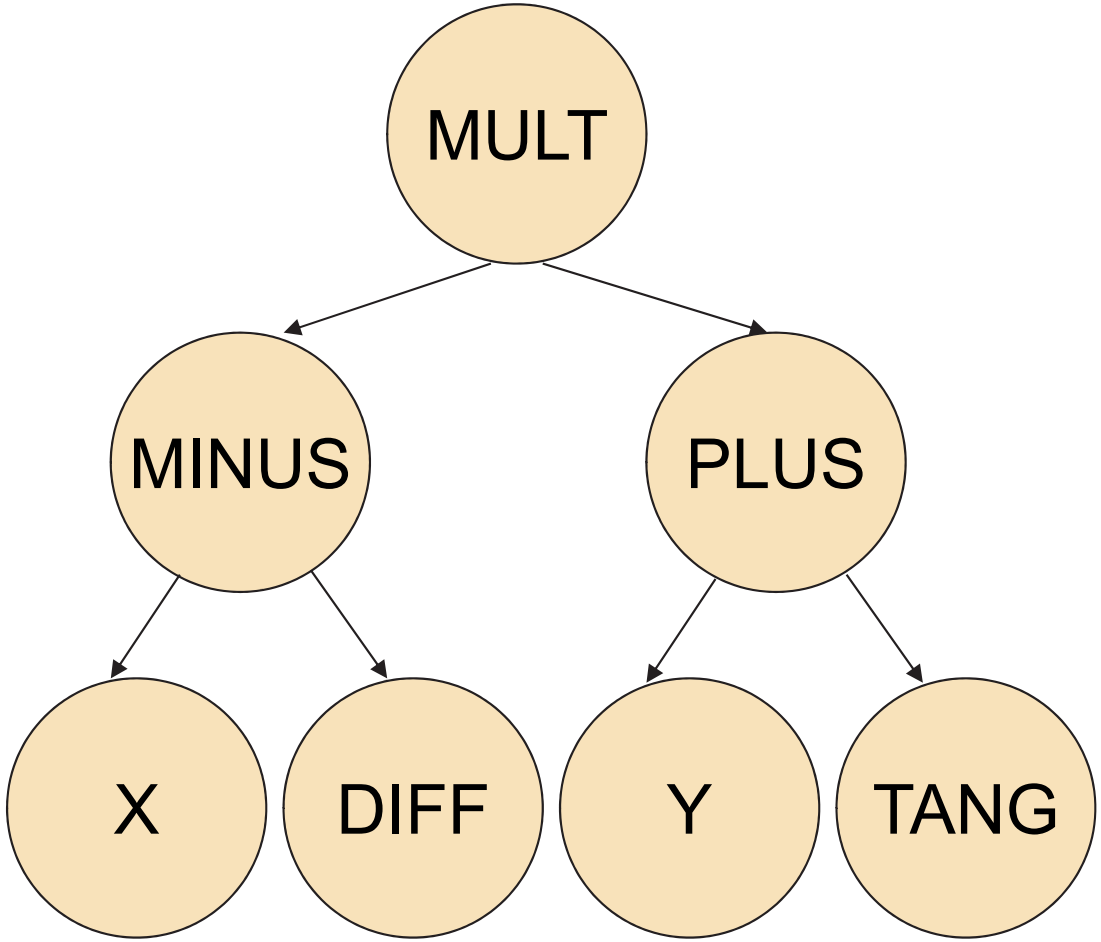
PLUS(a,b)	returns a+b
MINUS(a,b)	returns a-b
MUL(a,b)	returns a*b
DIV(a,b)	return a/b, if b <> 0, else 1
ATG(a,b)	returns atan2(a,b), if a<> 0, else 0
IFLTZ(a,b,c)	returns b, if a<0, else returns c

Arguments:

X	position x
Y	position y
DIFF	cab angle d
TANG	trailer angle t

Search space: set of symbolic expression using the above operators and arguments

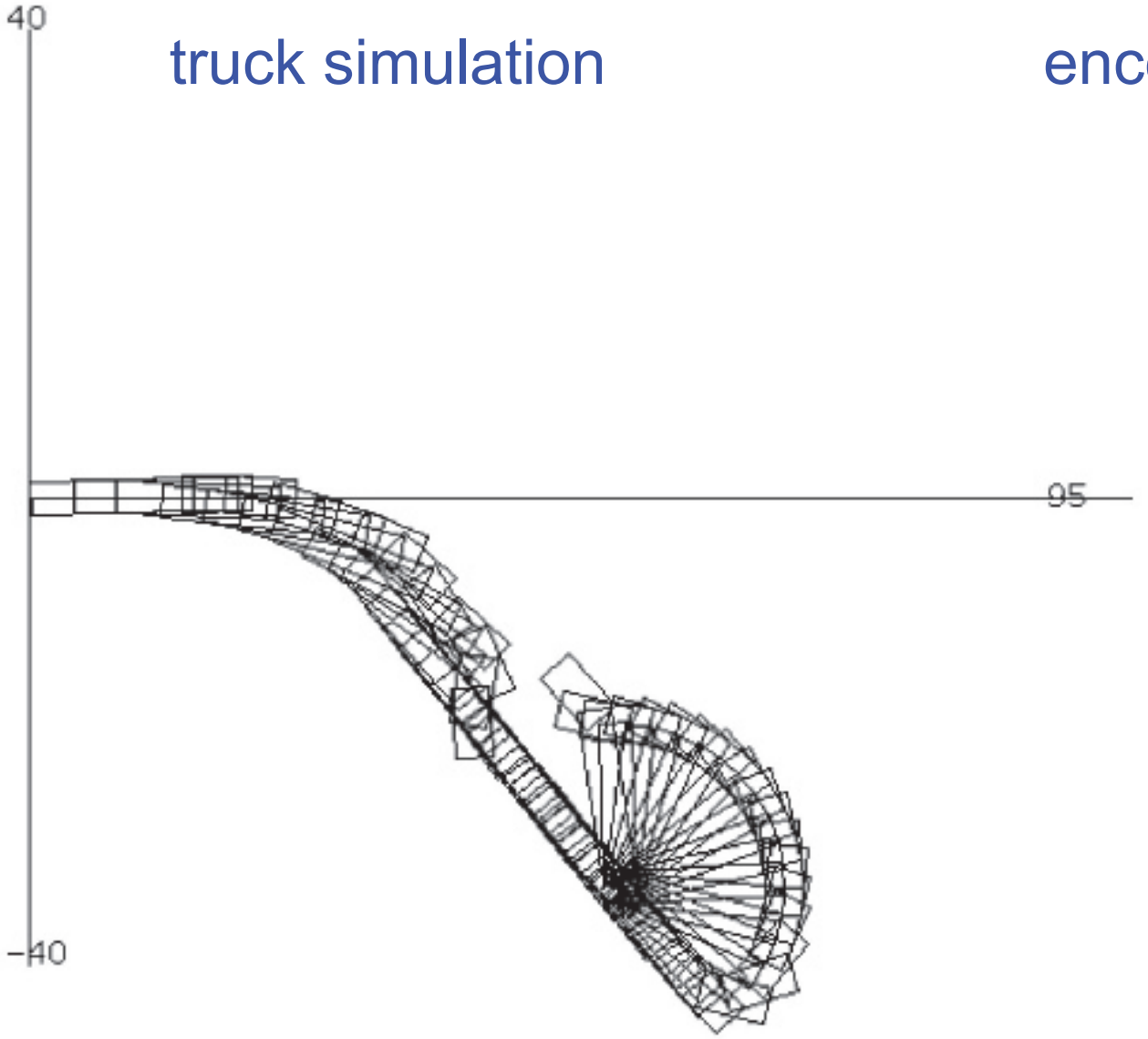
Example Solution: Tree Representation



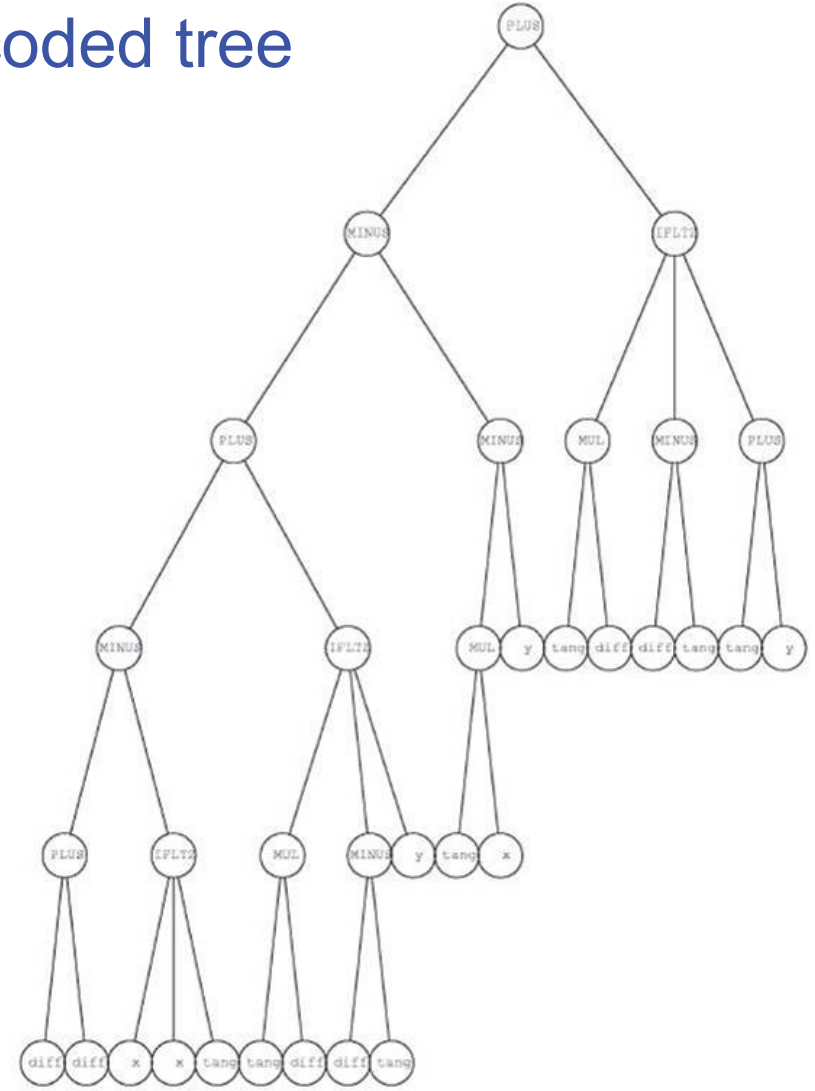
encodes the function (symbolic expression): u = (x – d) * (y + t)

A Solution Found by an Evolutionary Algorithm

truck simulation



encoded tree

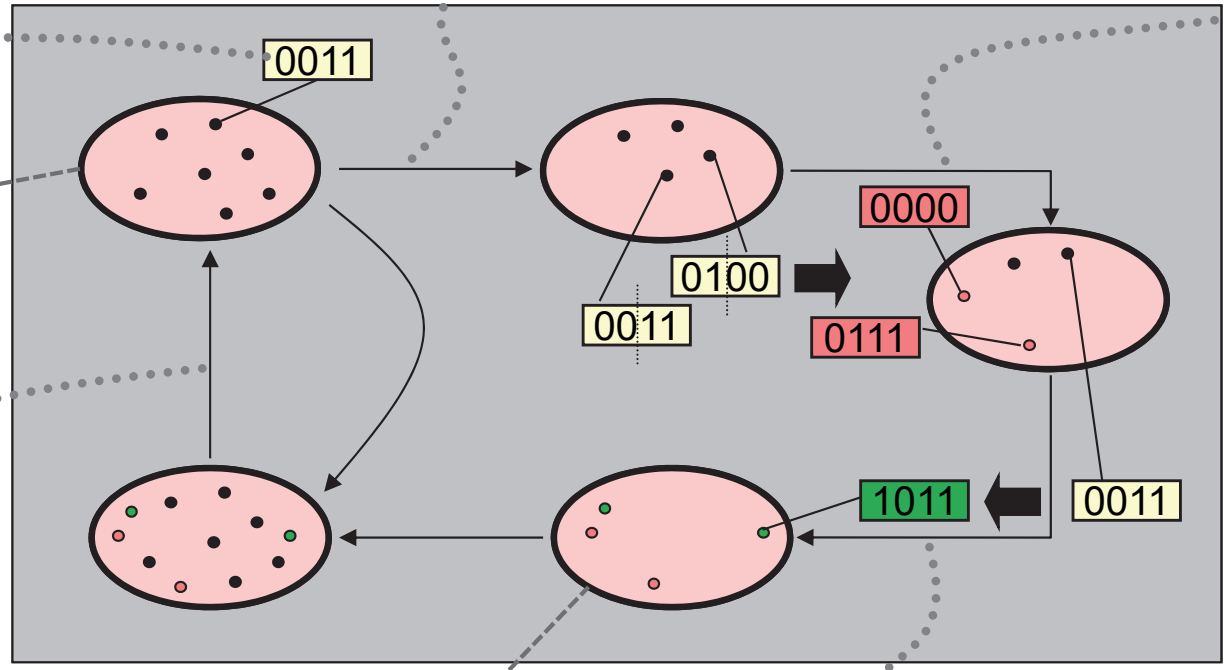


Evolutionary Algorithm Cycle

representation

mating selection

crossover operator



initial/parent set

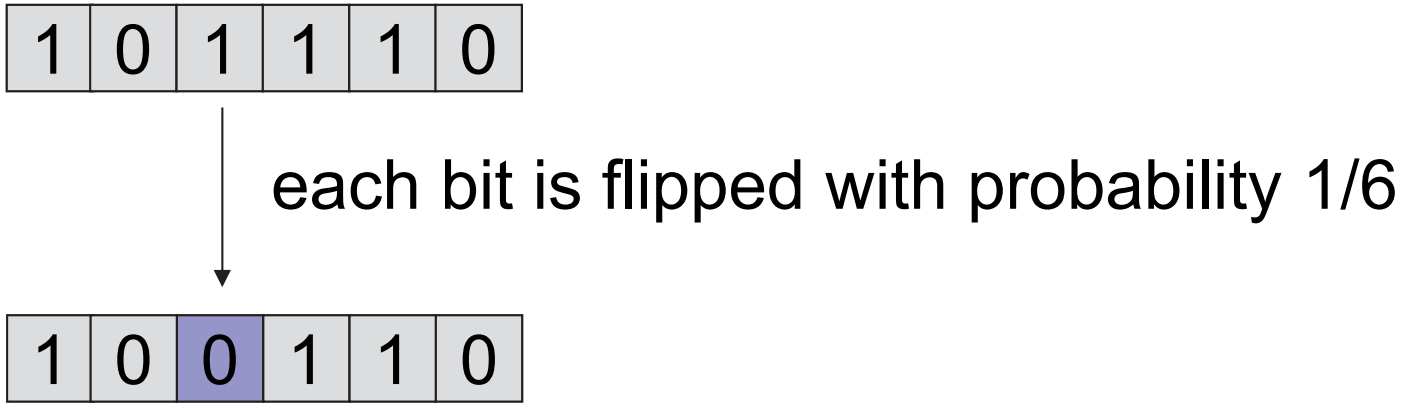
children set

environmental selection

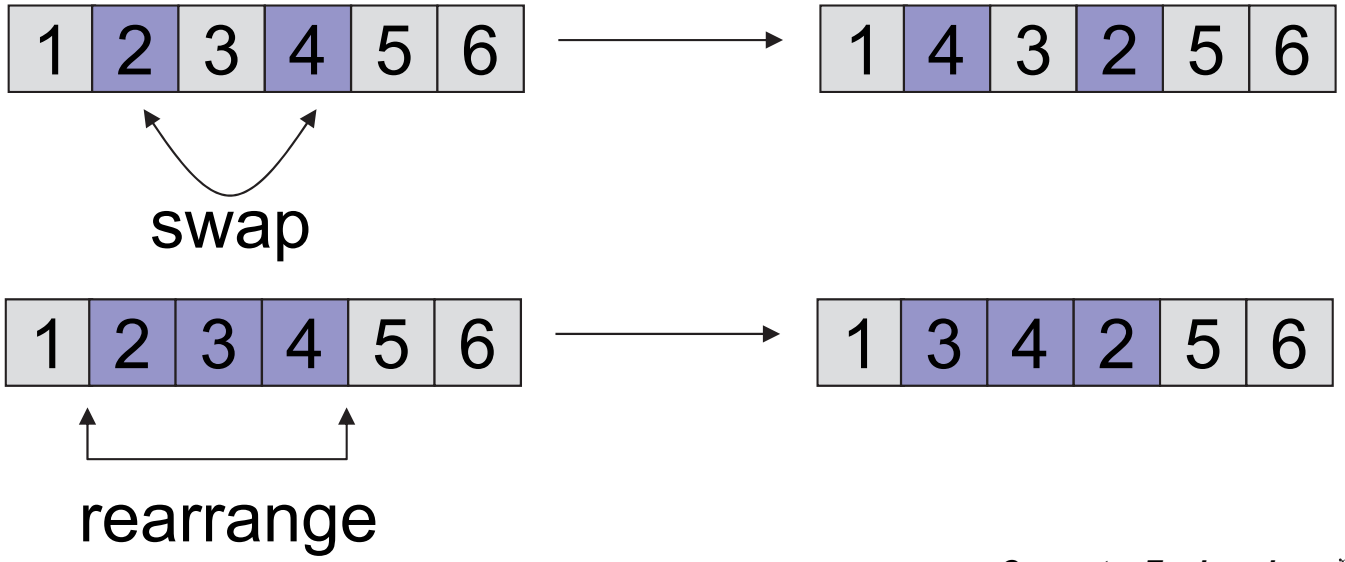
mutation operator

Vector Mutation: Examples

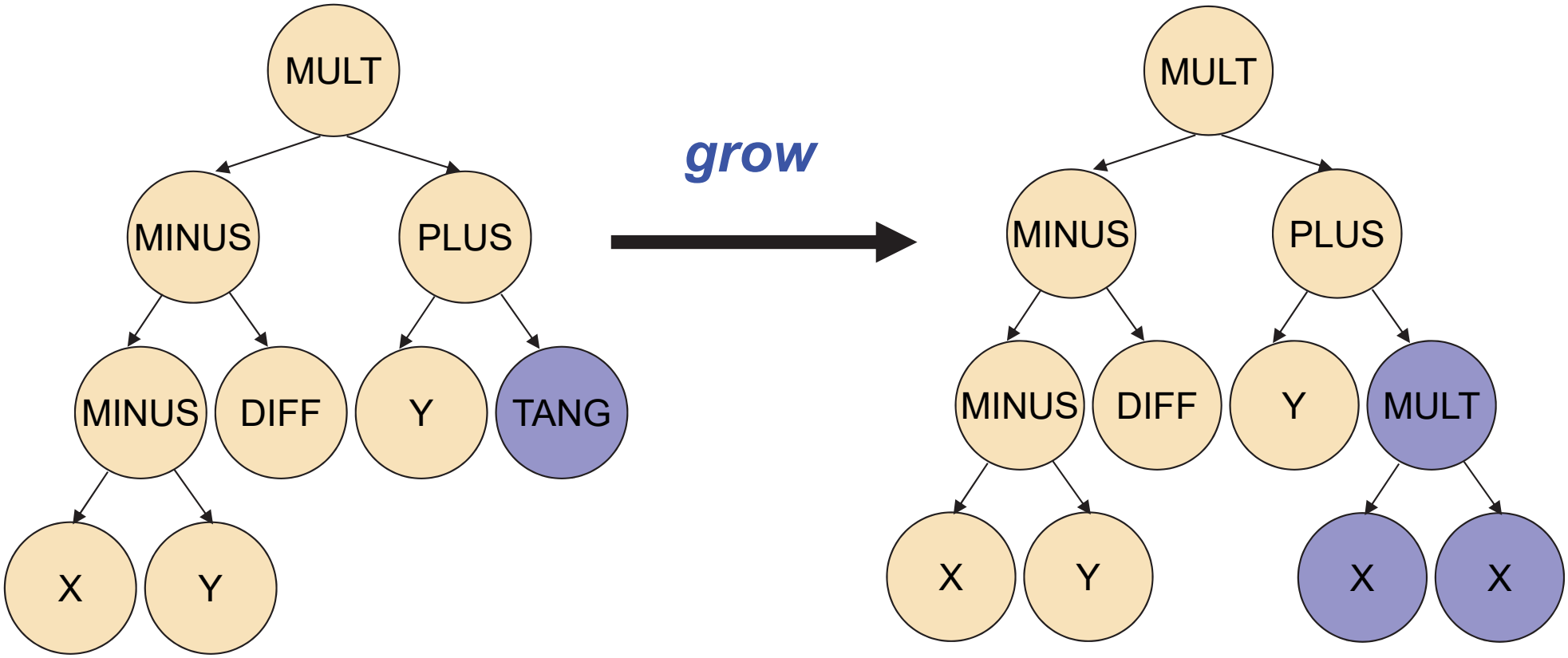
Bit vectors:



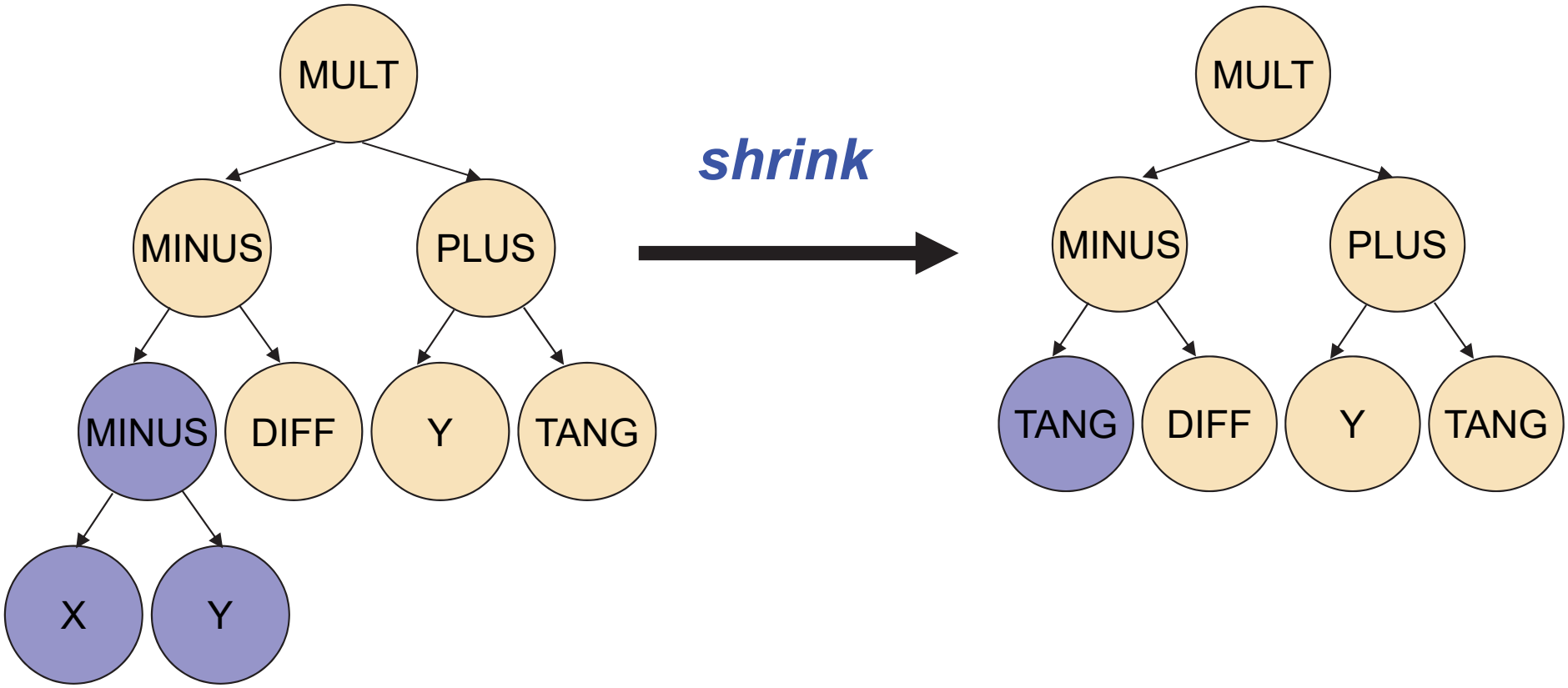
Permutations:



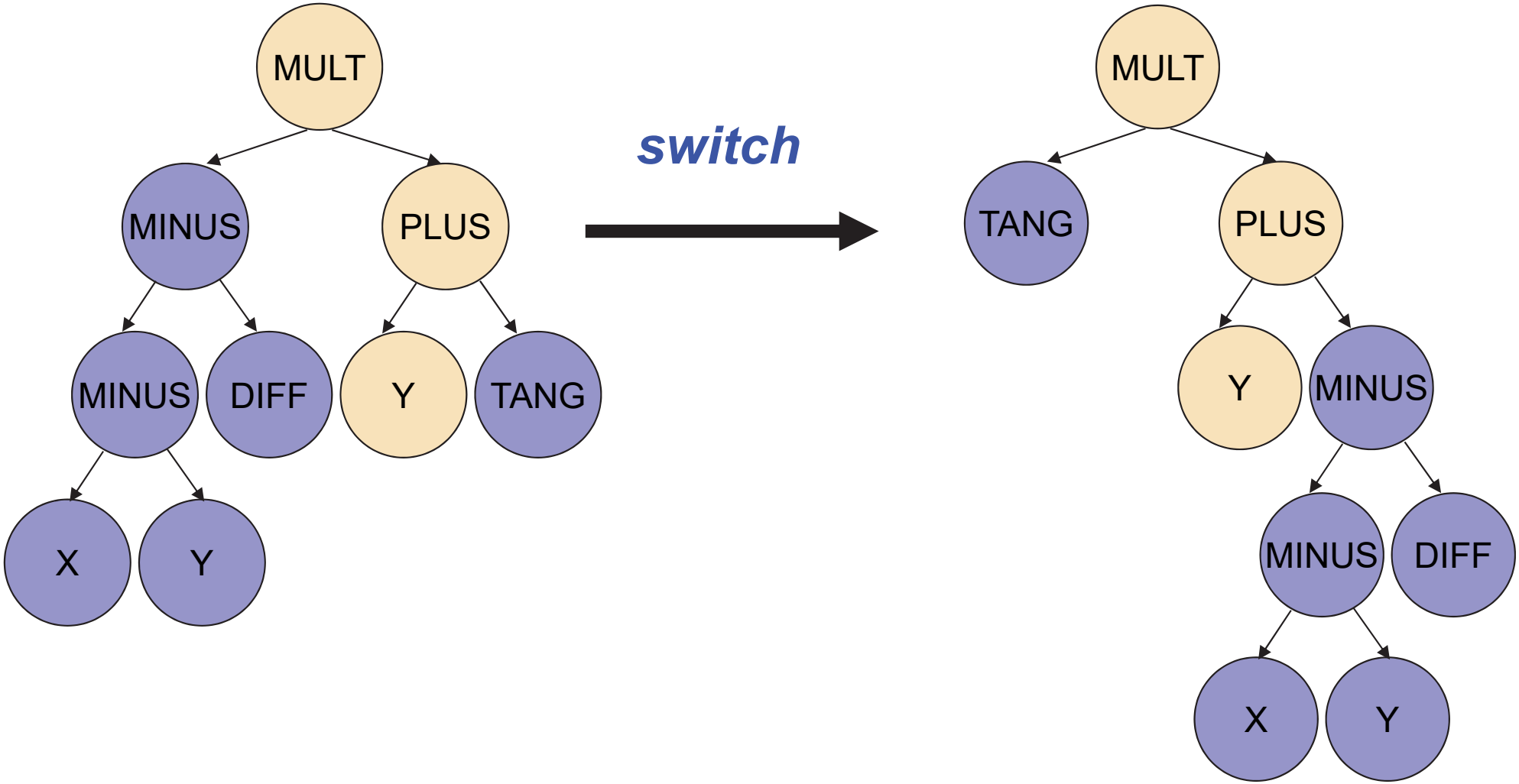
Mutation Operators on Trees: Grow



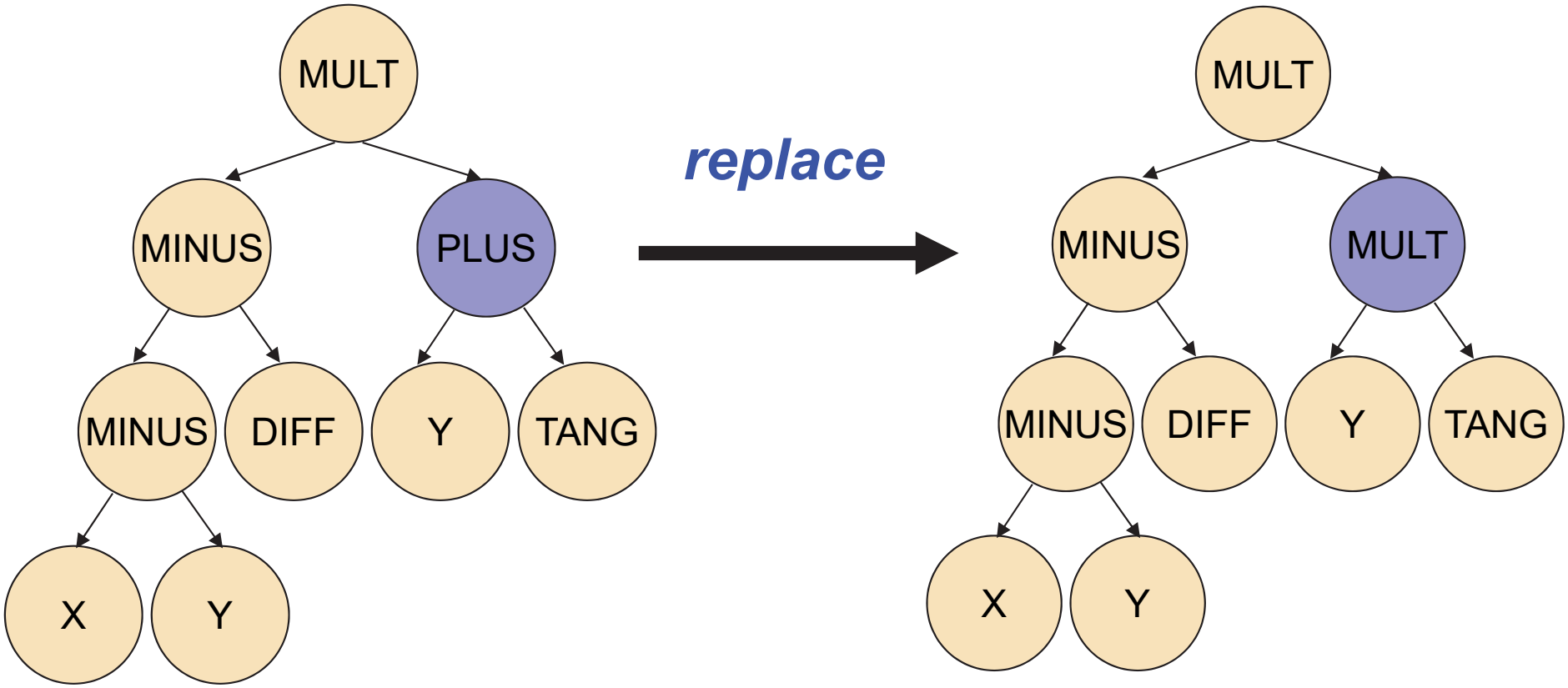
Mutation Operators on Trees: Shrink



Mutation Operators on Trees: Switch

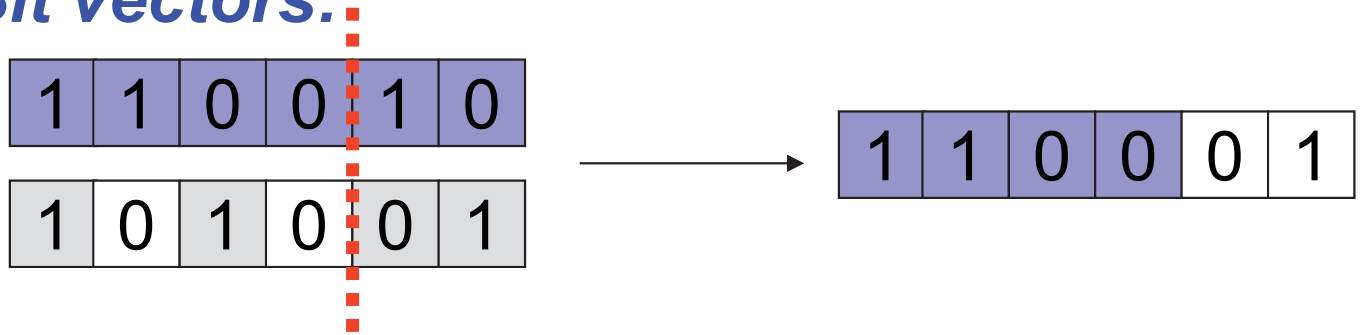


Mutation Operators on Trees: Replace

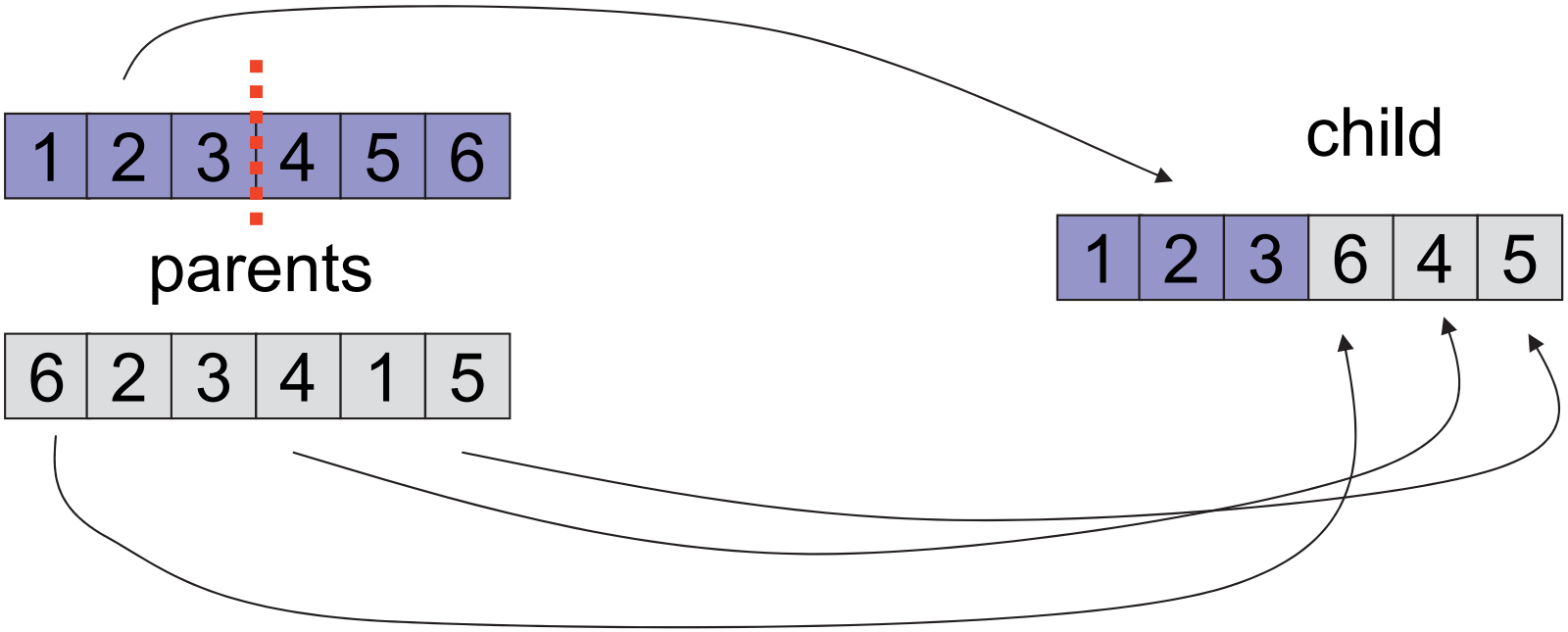


Vector Crossover: Examples

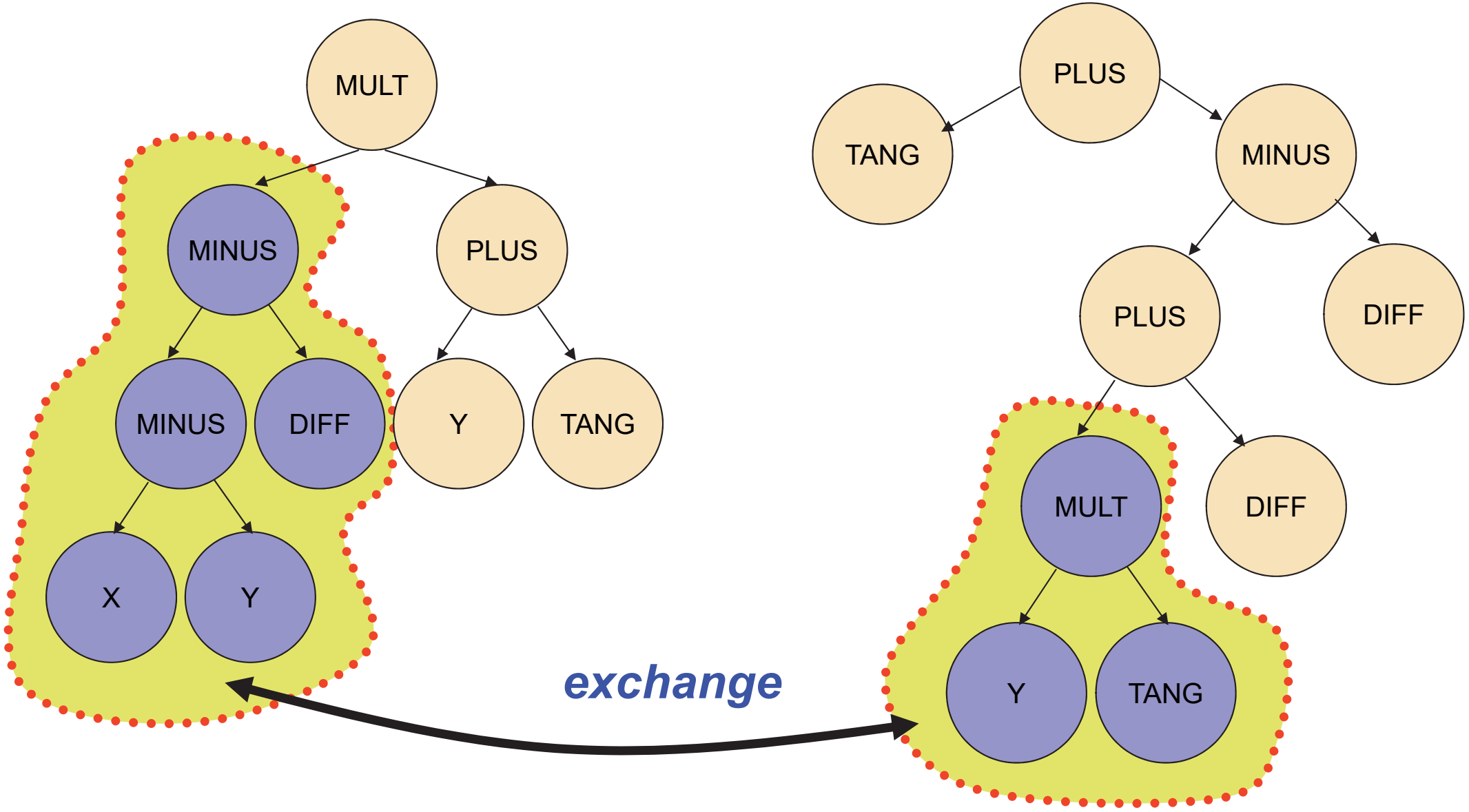
Bit vectors:



Permutations:



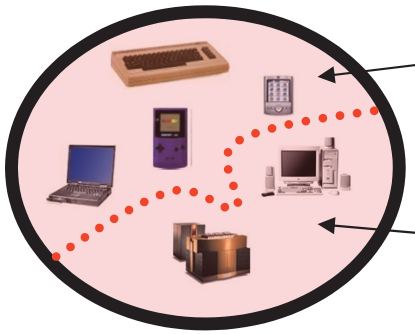
Recombination of Trees



Constraint Handling

Constraint: $g(x) \geq 0$

solution in decision space



feasible

$g \geq 0$

infeasible

$g < 0$

Approaches:

- representation is chosen such that decoding always yields a feasible solution
- construct initialization and neighborhood operators such that infeasible solutions are not generated
- add to children population only feasible solutions
- in environmental selection, preferably select feasible solutions
- calculate constraint violation g(x) and incorporate it into objective function using a penalty function: penalty(x) > 0 if g(x) < 0, penalty(x) = 0 if g(x) ≥ 0. For example, add penalty function to every objective.
- include the constraints as new objectives

Lecture Synopsis

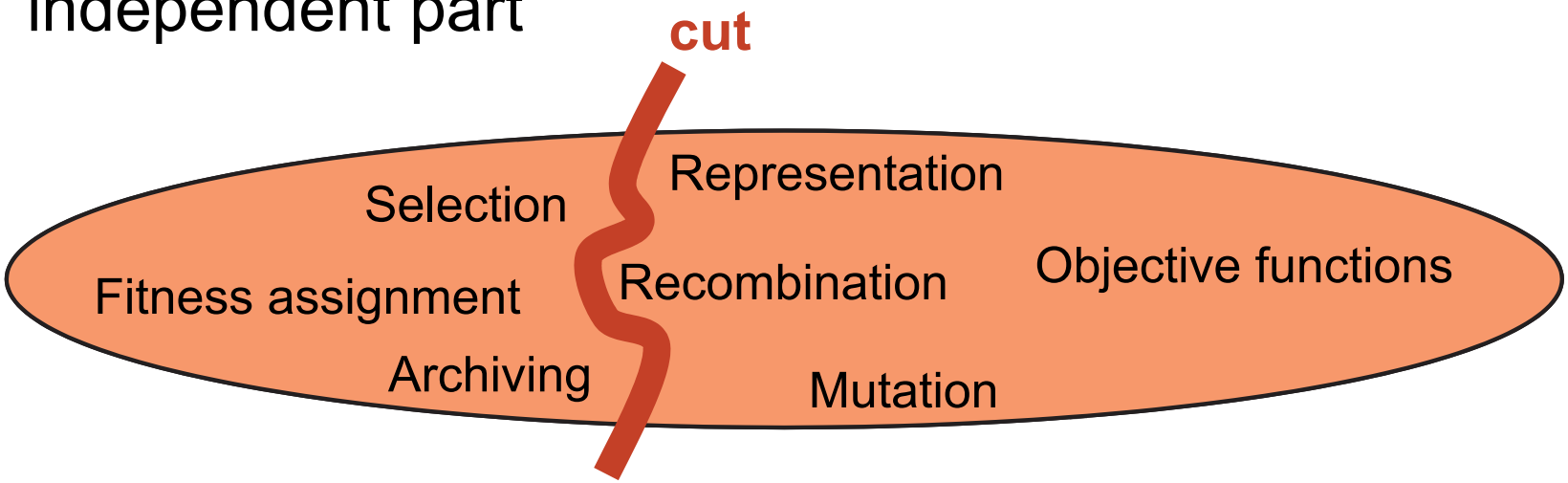
- Introduction
- Multiobjective Optimization
- Multiobjective Evolutionary Algorithms
- ***Implementation Aspects***

Implementation Framwork

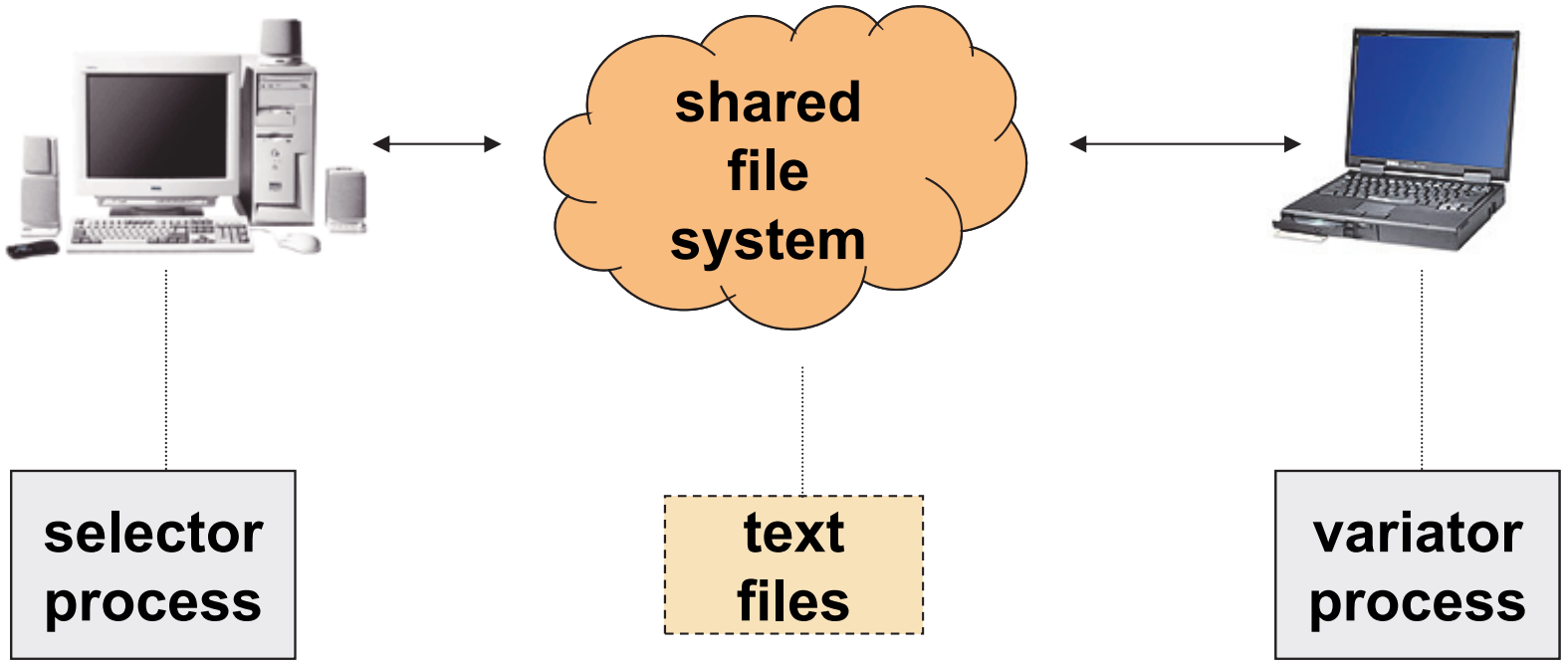
We want a framework that

- provides ready-to-use modules (algorithms / applications)
- is simple to use
- is independent of programming language and OS
- comes with minimum overhead

Idea: separate problem-dependent from problem-independent part



PISA: Implementation



application independent:

- environmental selection
- individuals are described by IDs and objective vectors

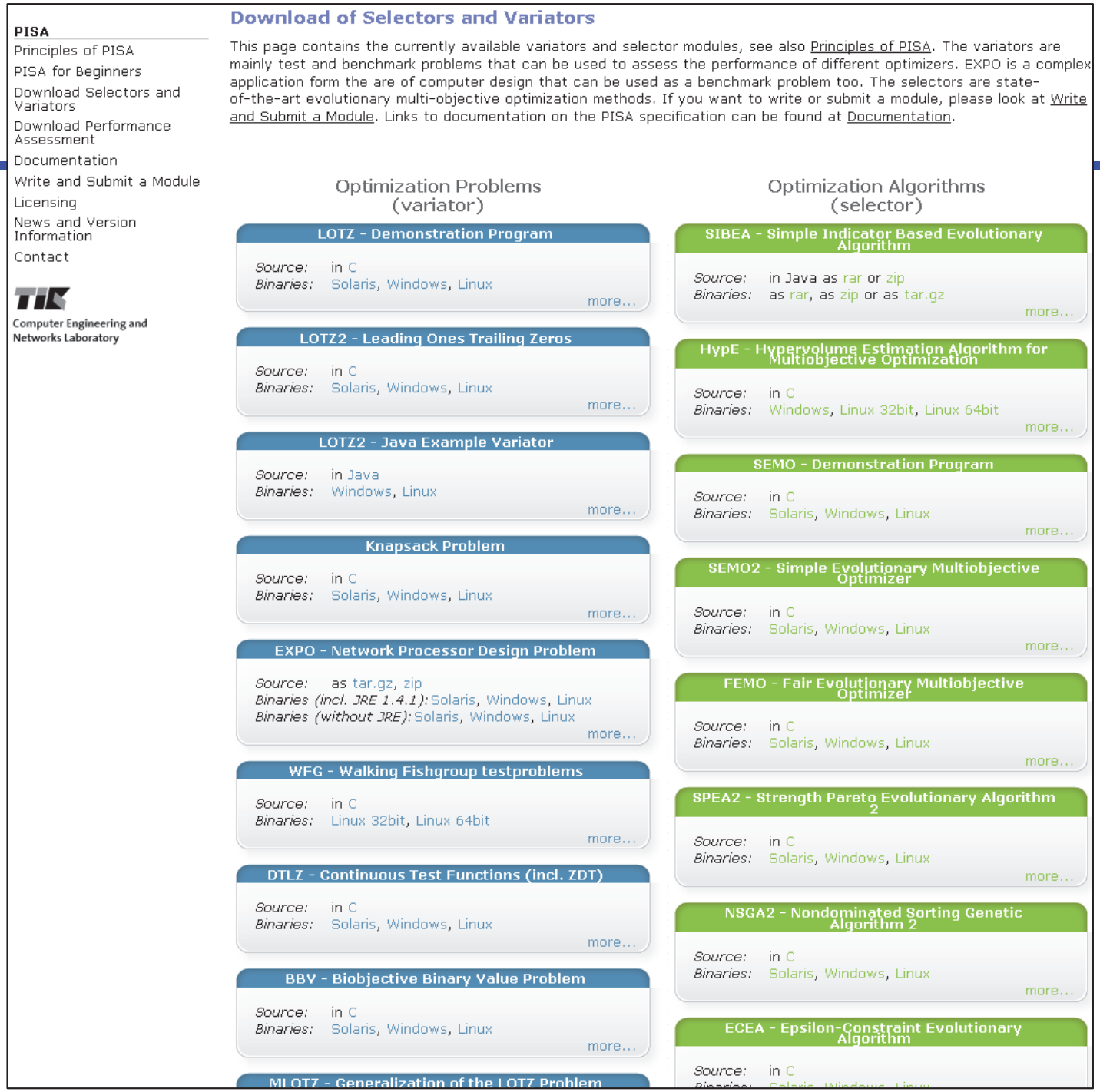
handshake protocol:

- state / action
- individual IDs
- objective vectors
- parameters

application dependent:

- neighborhood (cross-over and mutation) operators
- stores and manages individuals

PISA Installation



PISA
Principles of PISA
PISA for Beginners
Download Selectors and Variators
Download Performance Assessment
Documentation
Write and Submit a Module
Licensing
News and Version Information
Contact

Computer Engineering and Networks Laboratory

Download of Selectors and Variators

This page contains the currently available variators and selector modules, see also Principles of PISA. The variators are mainly test and benchmark problems that can be used to assess the performance of different optimizers. EXPO is a complex application form the are of computer design that can be used as a benchmark problem too. The selectors are state-of-the-art evolutionary multi-objective optimization methods. If you want to write or submit a module, please look at Write and Submit a Module. Links to documentation on the PISA specification can be found at Documentation.

Optimization Problems (variator)

LOTZ - Demonstration Program
Source: in C
Binaries: Solaris, Windows, Linux
more...

LOTZ2 - Leading Ones Trailing Zeros
Source: in C
Binaries: Solaris, Windows, Linux
more...

LOTZ2 - Java Example Variator
Source: in Java
Binaries: Windows, Linux
more...

Knapsack Problem
Source: in C
Binaries: Solaris, Windows, Linux
more...

EXPO - Network Processor Design Problem
Source: as tar.gz, zip
Binaries (incl. JRE 1.4.1): Solaris, Windows, Linux
Binaries (without JRE): Solaris, Windows, Linux
more...

WFG - Walking Fishgroup testproblems
Source: in C
Binaries: Linux 32bit, Linux 64bit
more...

DTLZ - Continuous Test Functions (incl. ZDT)
Source: in C
Binaries: Solaris, Windows, Linux
more...

BBV - Biobjective Binary Value Problem
Source: in C
Binaries: Solaris, Windows, Linux
more...

MLOTZ - Generalization of the LOTZ Problem

Optimization Algorithms (selector)

SIBEA - Simple Indicator Based Evolutionary Algorithm
Source: in Java as rar or zip
Binaries: as rar, as zip or as tar.gz
more...

HypE - Hypervolume Estimation Algorithm for Multiobjective Optimization
Source: in C
Binaries: Windows, Linux 32bit, Linux 64bit
more...

SEMO - Demonstration Program
Source: in C
Binaries: Solaris, Windows, Linux
more...

SEMO2 - Simple Evolutionary Multiobjective Optimizer
Source: in C
Binaries: Solaris, Windows, Linux
more...

FEMO - Fair Evolutionary Multiobjective Optimizer
Source: in C
Binaries: Solaris, Windows, Linux
more...

SPEA2 - Strength Pareto Evolutionary Algorithm 2
Source: in C
Binaries: Solaris, Windows, Linux
more...

NSGA2 - Nondominated Sorting Genetic Algorithm 2
Source: in C
Binaries: Solaris, Windows, Linux
more...

ECEA - Epsilon-Constraint Evolutionary Algorithm
Source: in C