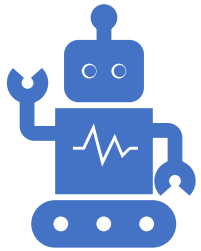


UiO : University of Oslo

institutt for informatikk

IN3050/IN4050, Lecture 3
Evolutionary algorithms 1



1: Introduction to evolution

2: Evolutionary algorithms

3: Components of an evolutionary algorithm

4: Binary, integer and real-valued representations

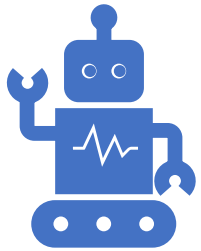
5: Permutation and tree-based representations

6: Example of a simple evolutionary algorithm



UiO : University of Oslo

institutt for informatikk

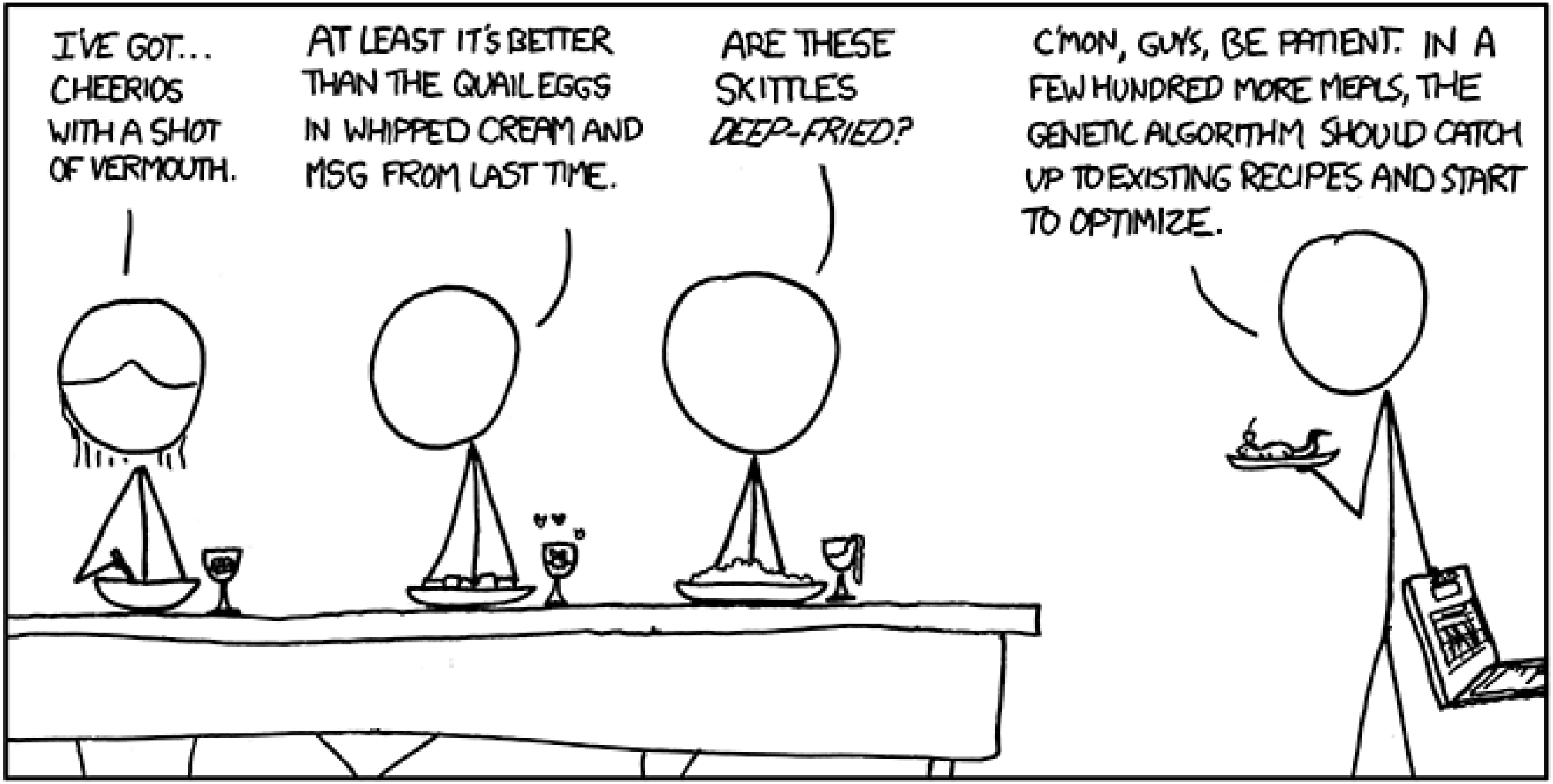


IN3050/IN4050, Lecture 3
Evolutionary algorithms 1

1: Introduction to evolution

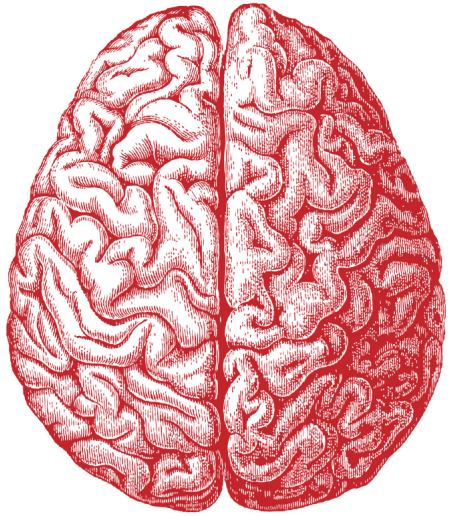
Kai Olav Ellefsen

Next video: Evolutionary algorithms



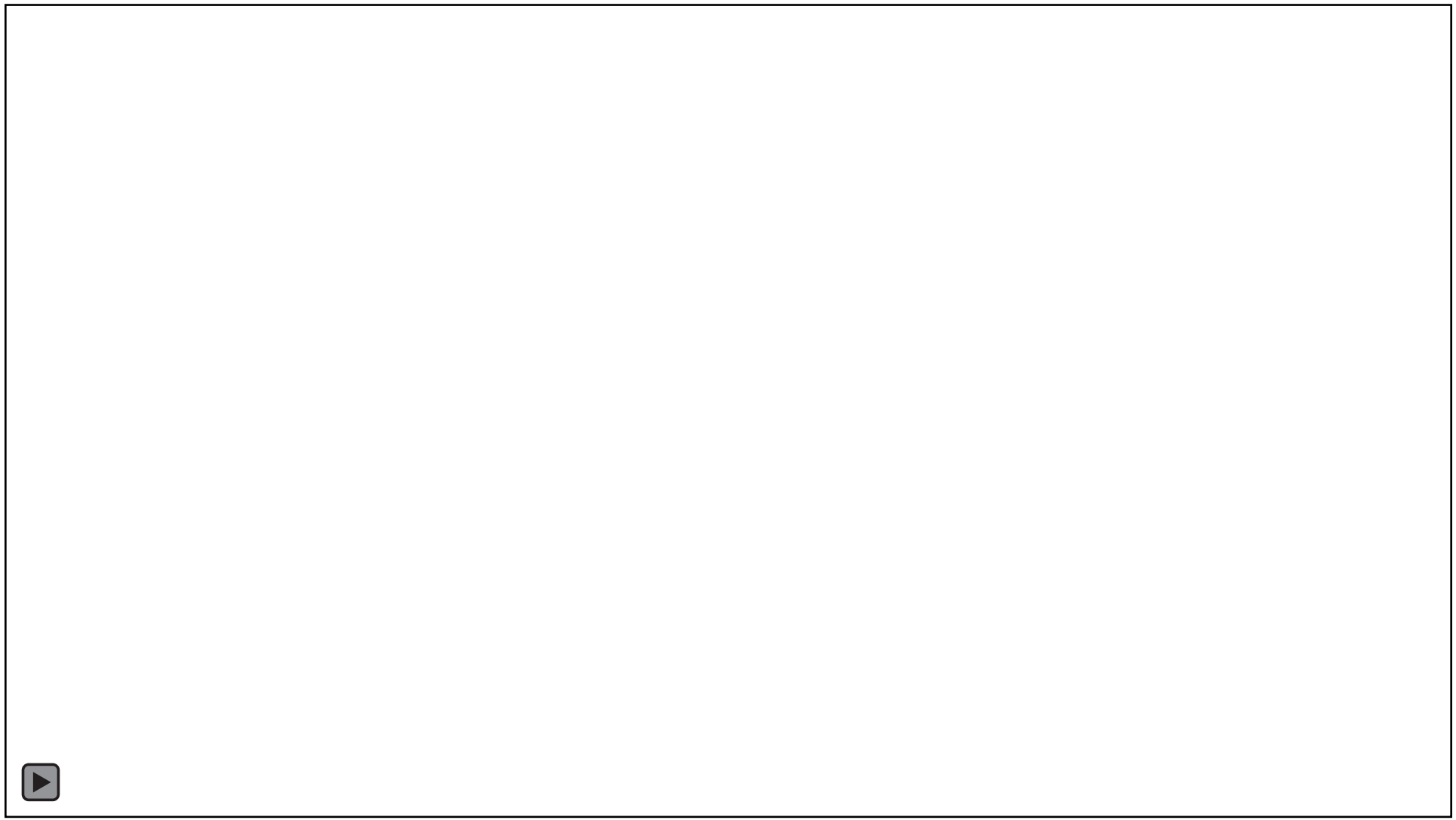
WE'VE DECIDED TO DROP THE CS DEPARTMENT FROM OUR WEEKLY DINNER PARTY HOSTING ROTATION.

Why Draw Inspiration from Evolution?

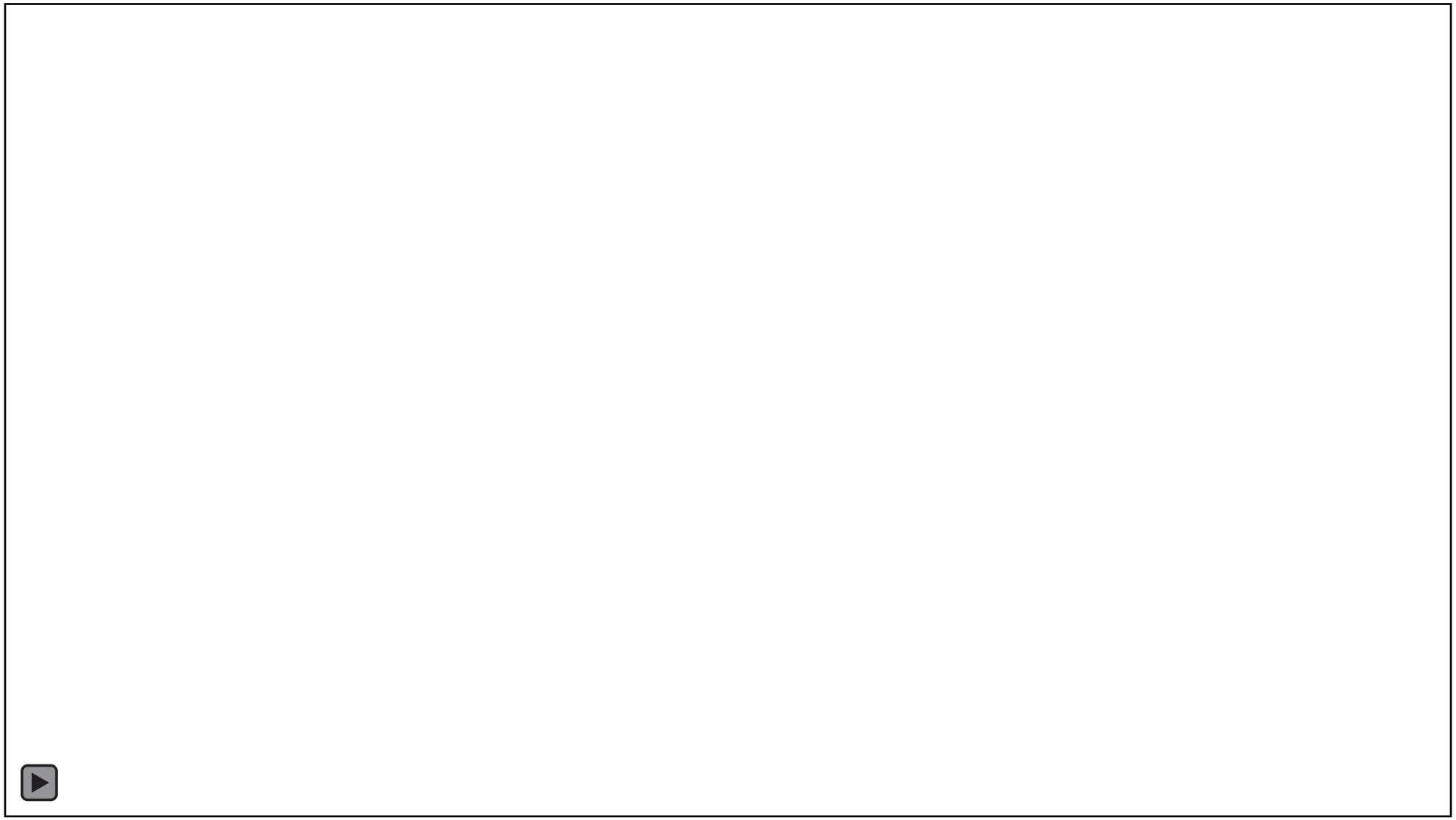








Video from https://www.youtube.com/watch?v=g0TaYhjpOfo



Video from https://www.youtube.com/watch?v=T-c17RKh3uE

Evolution

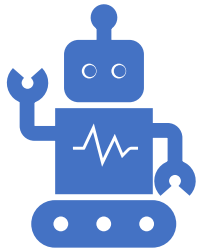


- **Biological evolution:**
  - Lifeforms adapt to a particular environment over successive generations.
  - Combinations of traits that are better adapted tend to increase representation in population.
  - Mechanisms: Variation (Crossover, Mutation) and Selection (Survival of the fittest).
- **Evolutionary Computing (EC):**
  - Mimic the biological evolution to optimize solutions to a wide variety of complex problems.
  - In every new generation, a new set of solutions is created using bits and pieces of the fittest of the old.



UiO : University of Oslo

institutt for informatikk



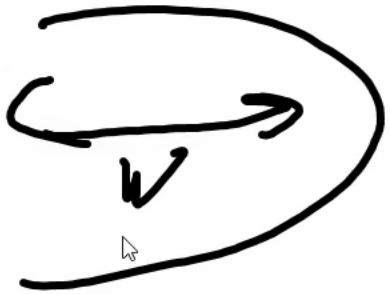
IN3050/IN4050, Lecture 3
Evolutionary algorithms 1

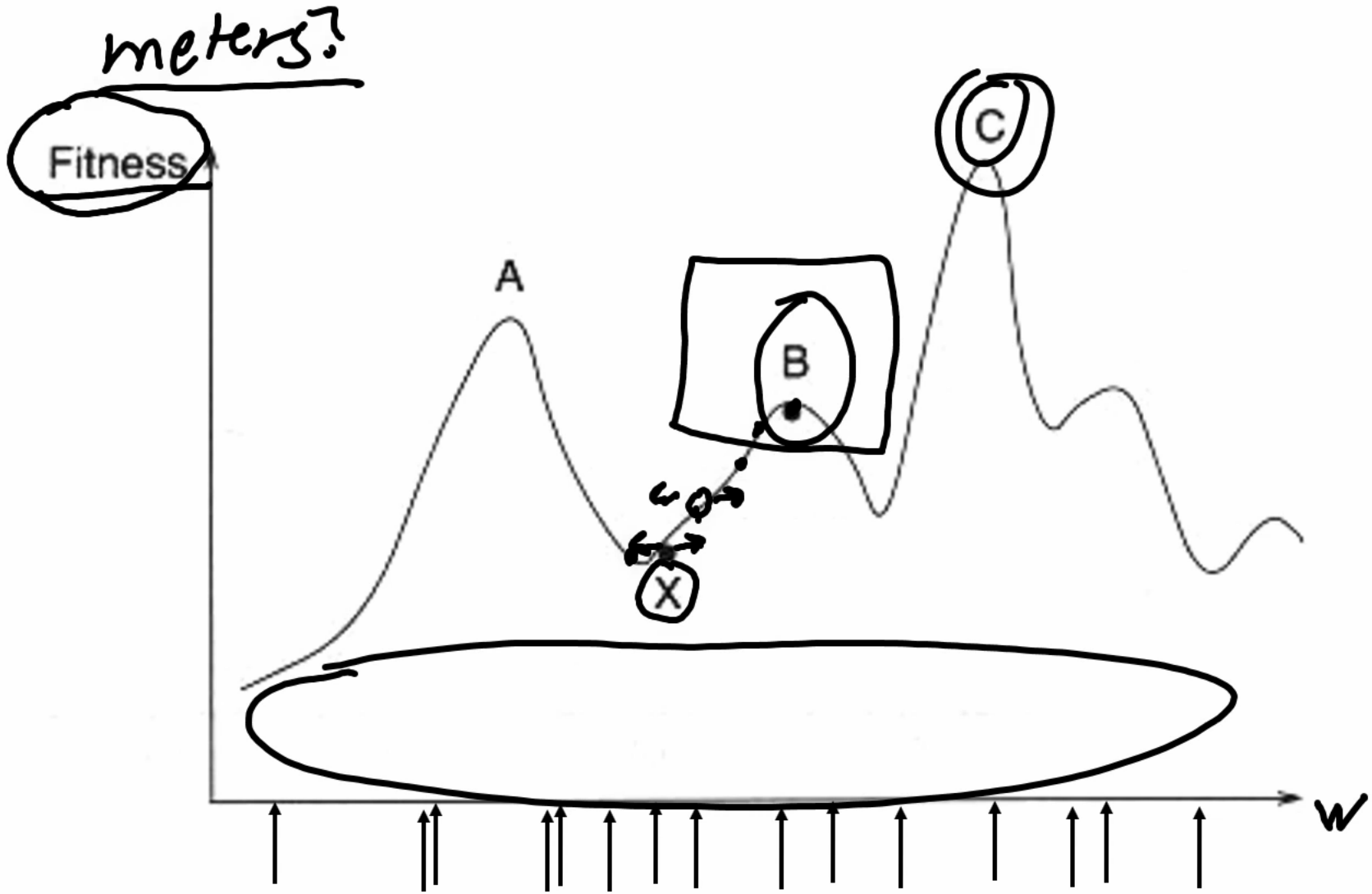
2: Evolutionary algorithms

Kai Olav Ellefsen

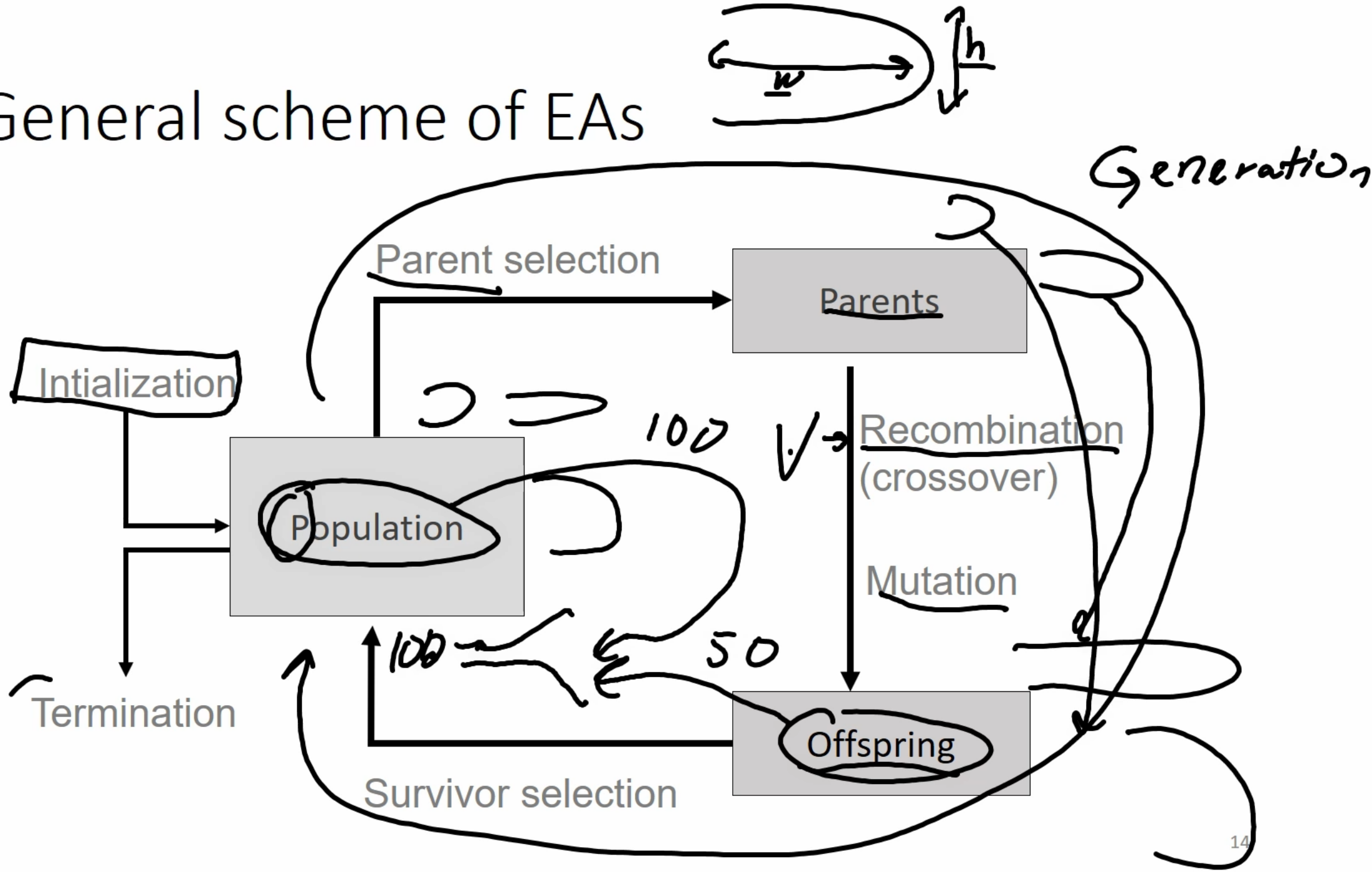
Next video: Components of an evolutionary algorithm

The Problem with Hillclimbing





General scheme of EAs

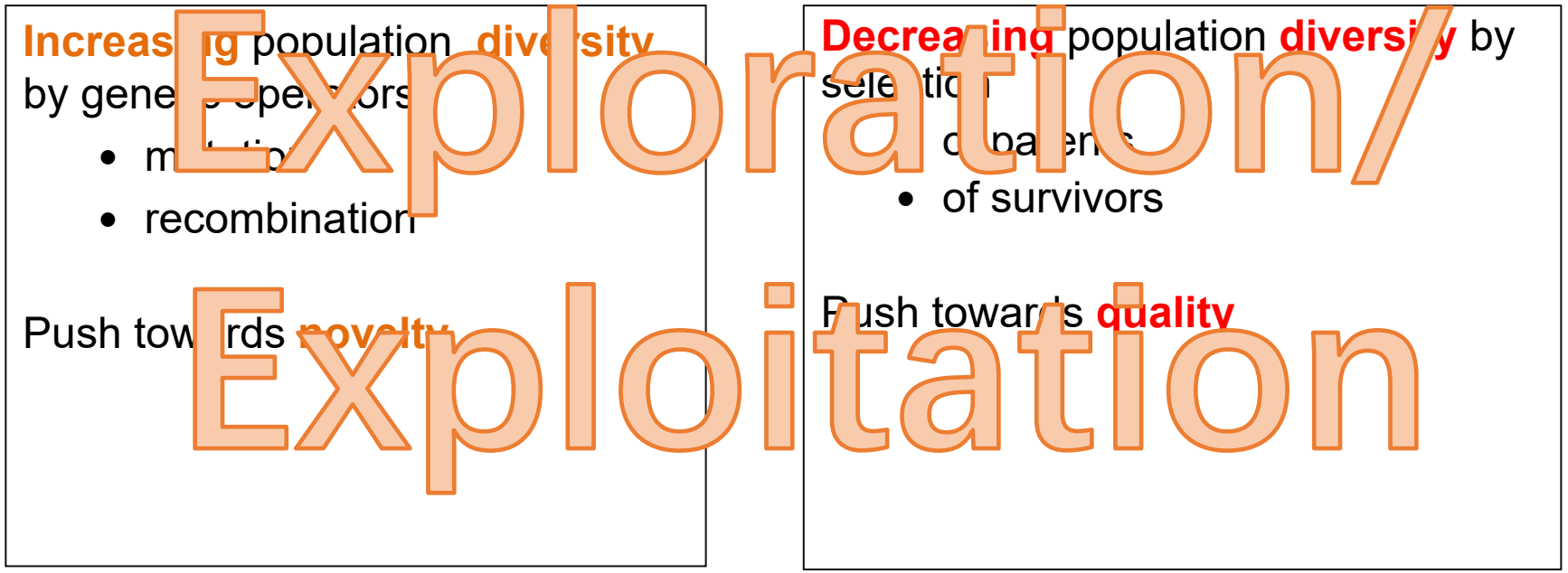


EA scheme in pseudo-code

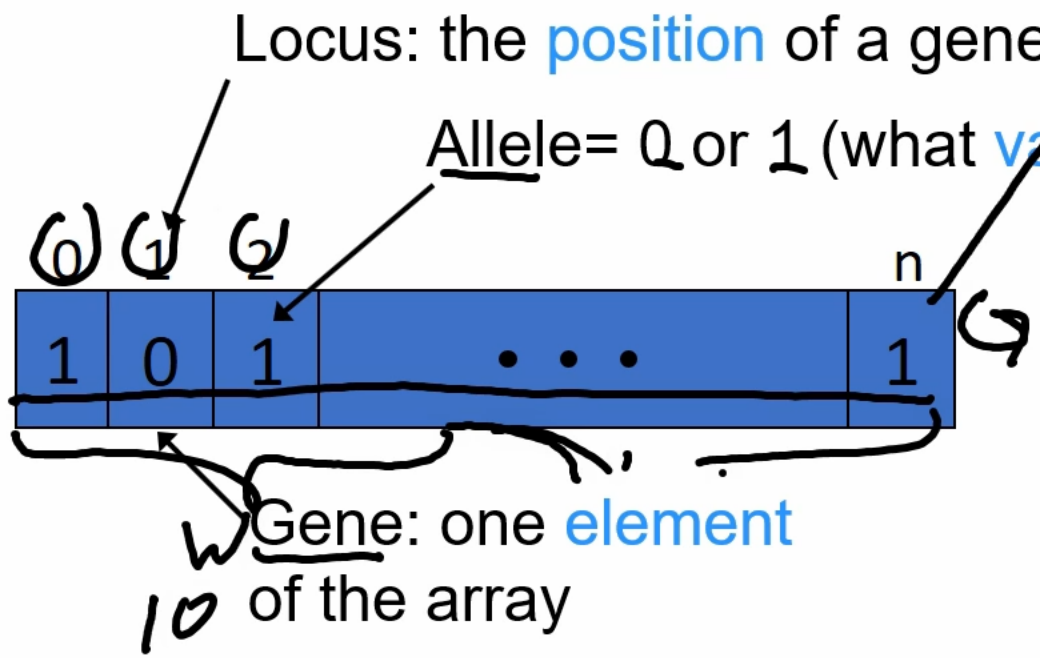
```
BEGIN
  INITIALISE population with random candidate solutions;
  EVALUATE each candidate;
  REPEAT UNTIL ( TERMINATION CONDITION is satisfied ) DO
    1 SELECT parents;
    2 RECOMBINE pairs of parents;
    3 MUTATE the resulting offspring;
    4 EVALUATE new candidates;
    5 SELECT individuals for the next generation;
  OD
END
```

Scheme of an EA:
Two pillars of evolution

There are two competing forces



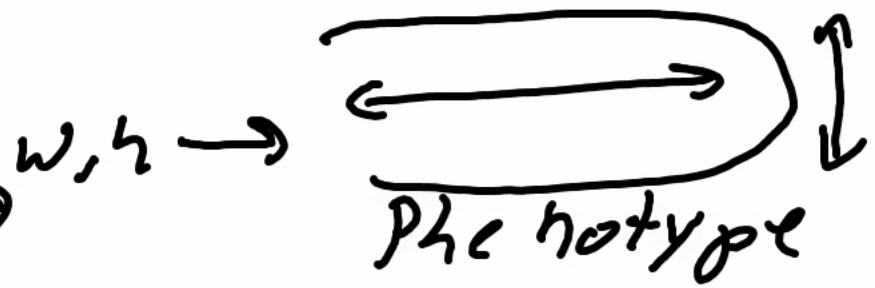
Representation: EA terms



Locus: the position of a gene

Allele= 0 or 1 (what values a gene can have)

Gene: one element of the array



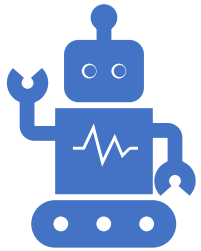
Genotype: a set of gene values

Phenotype: what could be built/developed based on the genotype



UiO : University of Oslo

institutt for informatikk



IN3050/IN4050, Lecture 3
Evolutionary algorithms 1

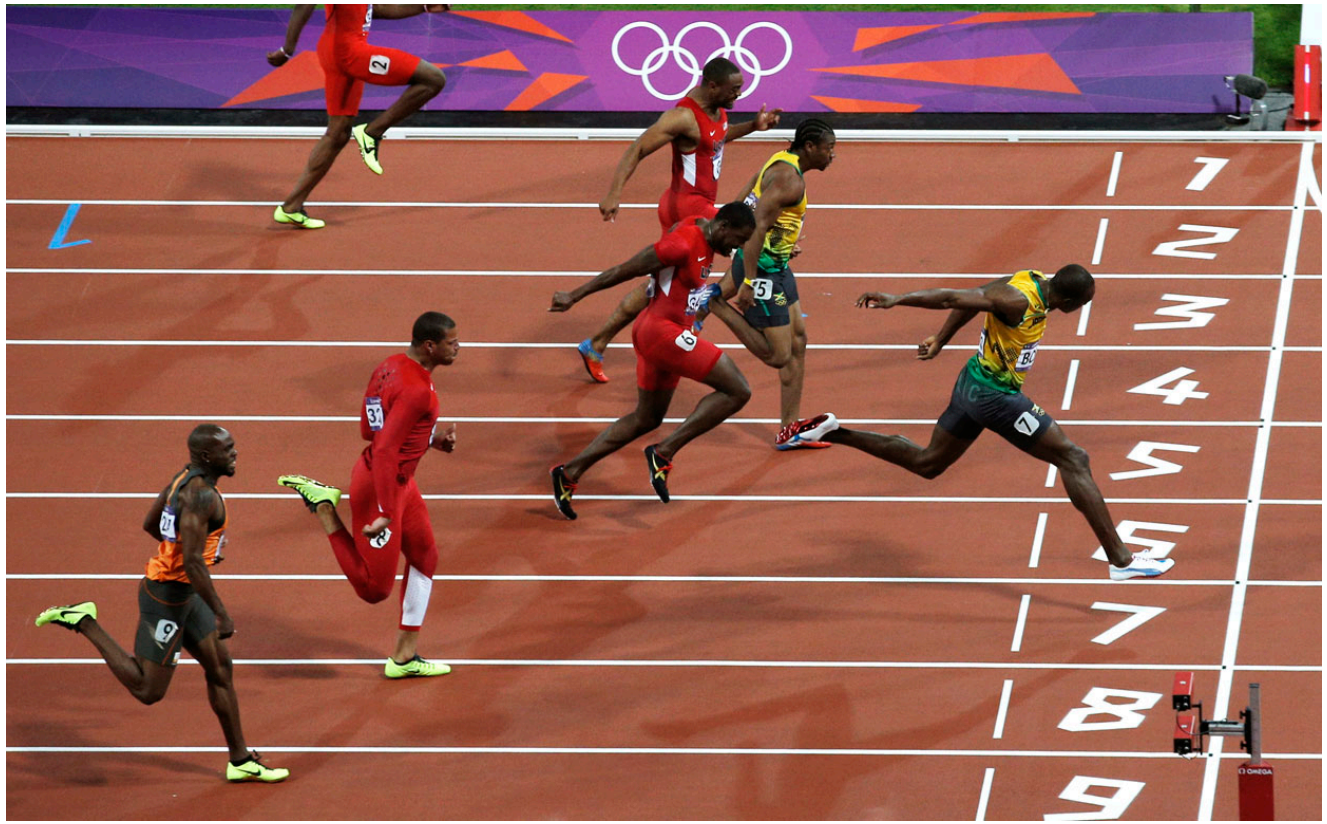
3: Components of an evolutionary algorithm

Kai Olav Ellefsen

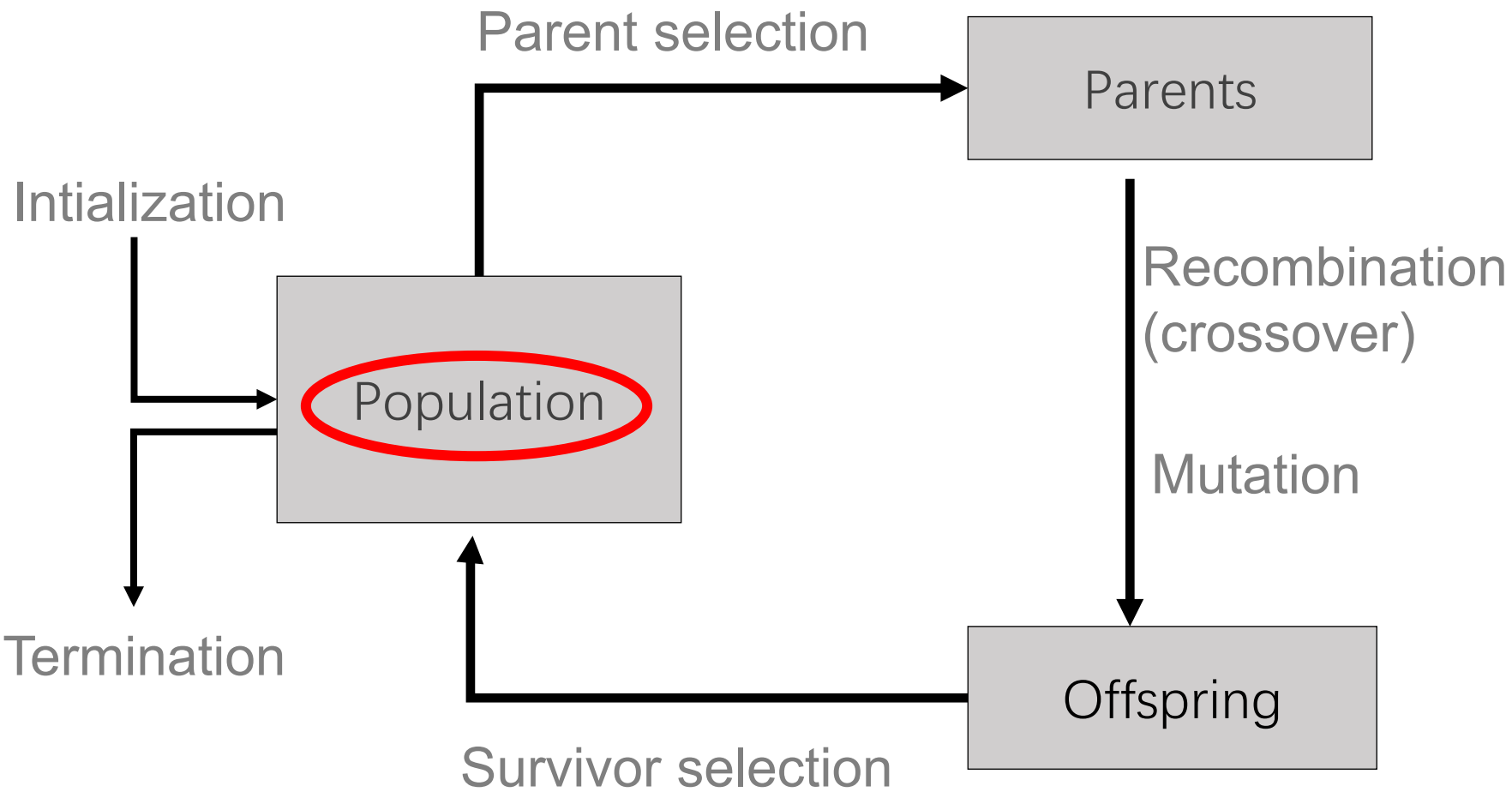
Next video: Binary, integer and real-valued representations

Main EA components: Evaluation (fitness) function

- Represents the task to solve
- Enables selection (provides basis for comparison)
- Assigns a single real-valued fitness to each phenotype



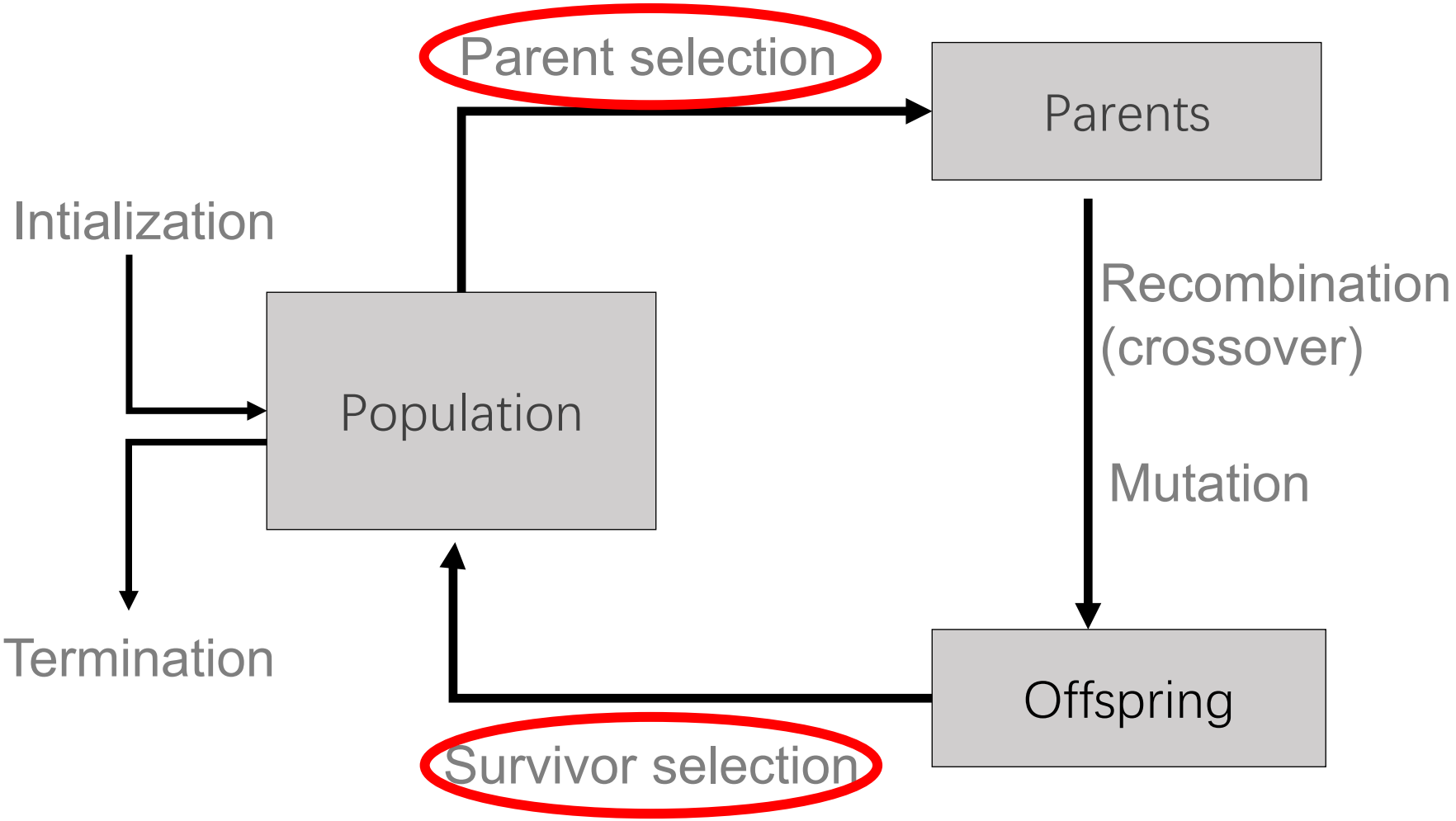
General scheme of EAs



Main EA components: Population

- The **candidate solutions (individuals)** of the problem
- Population is the basic unit of evolution, i.e., the **population is evolving**, not the individuals
- **Selection** operators act on **population level**
- **Variation** operators act on **individual level**

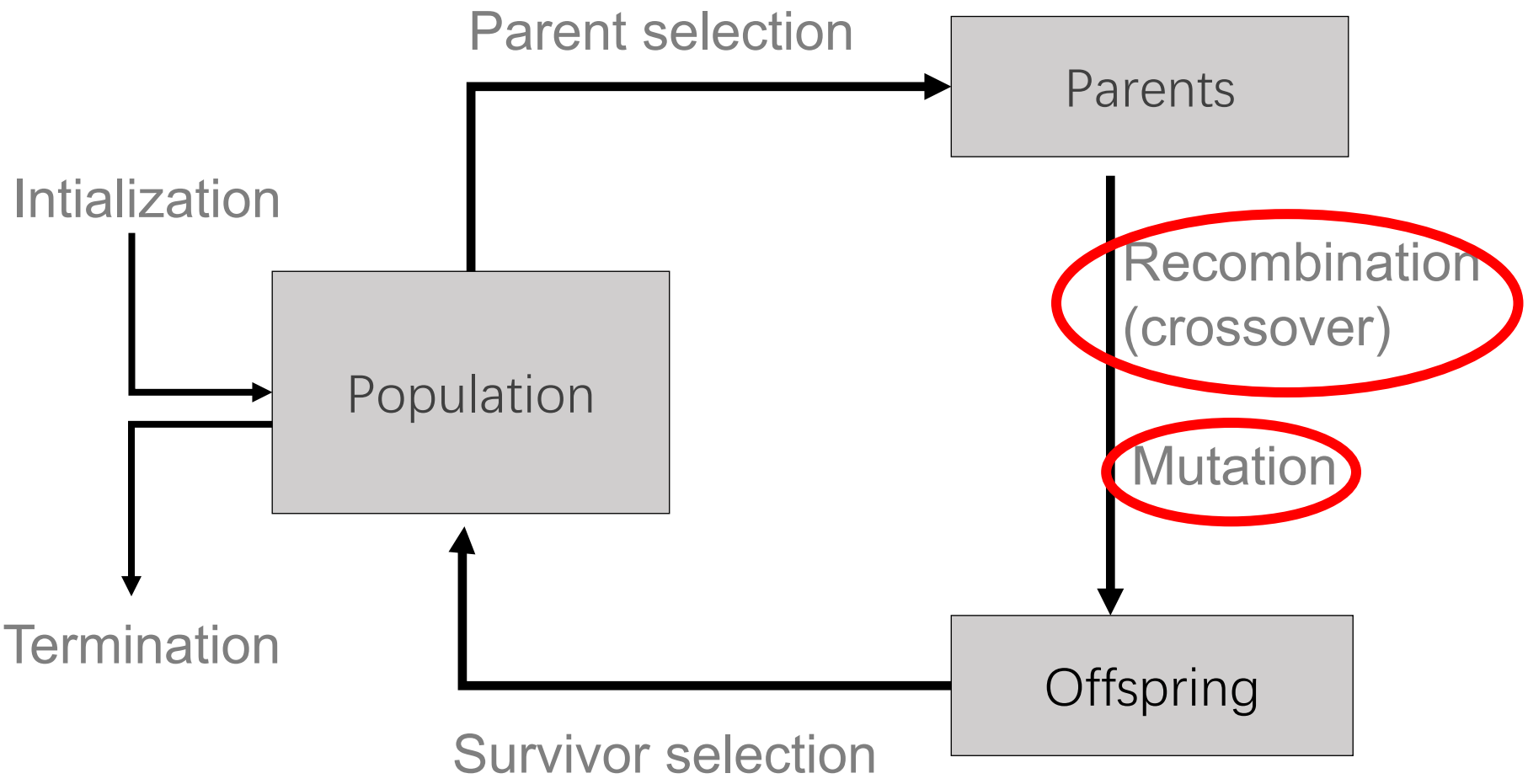
General scheme of EAs



Main EA components: Selection mechanisms

- Identify individuals
  - to become parents
  - to survive
- Pushes population towards higher fitness
- Parent selection is usually probabilistic
  - high quality solutions more likely to be selected than low quality, but not guaranteed
  - This *stochastic* nature can aid escape from local optima
- More on selection next week!

General scheme of EAs

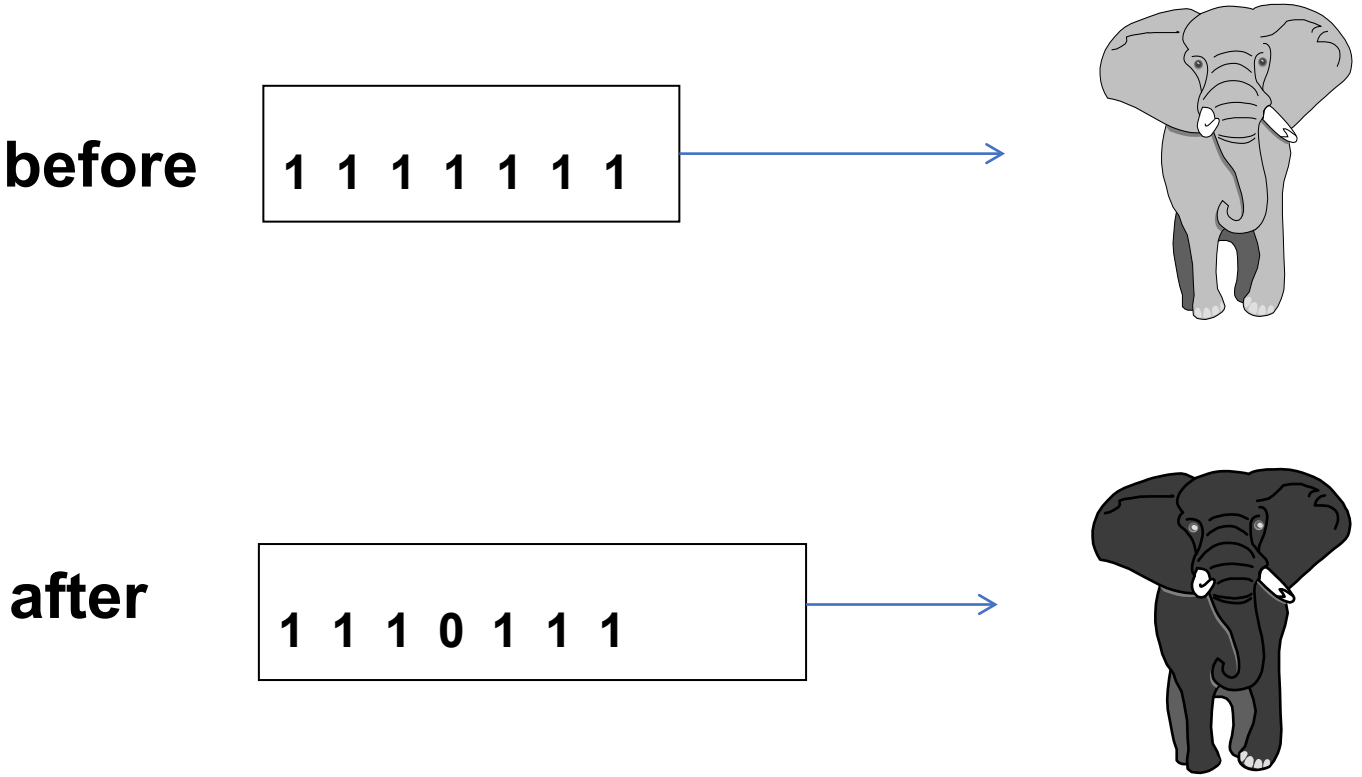


Main EA components: Variation operators

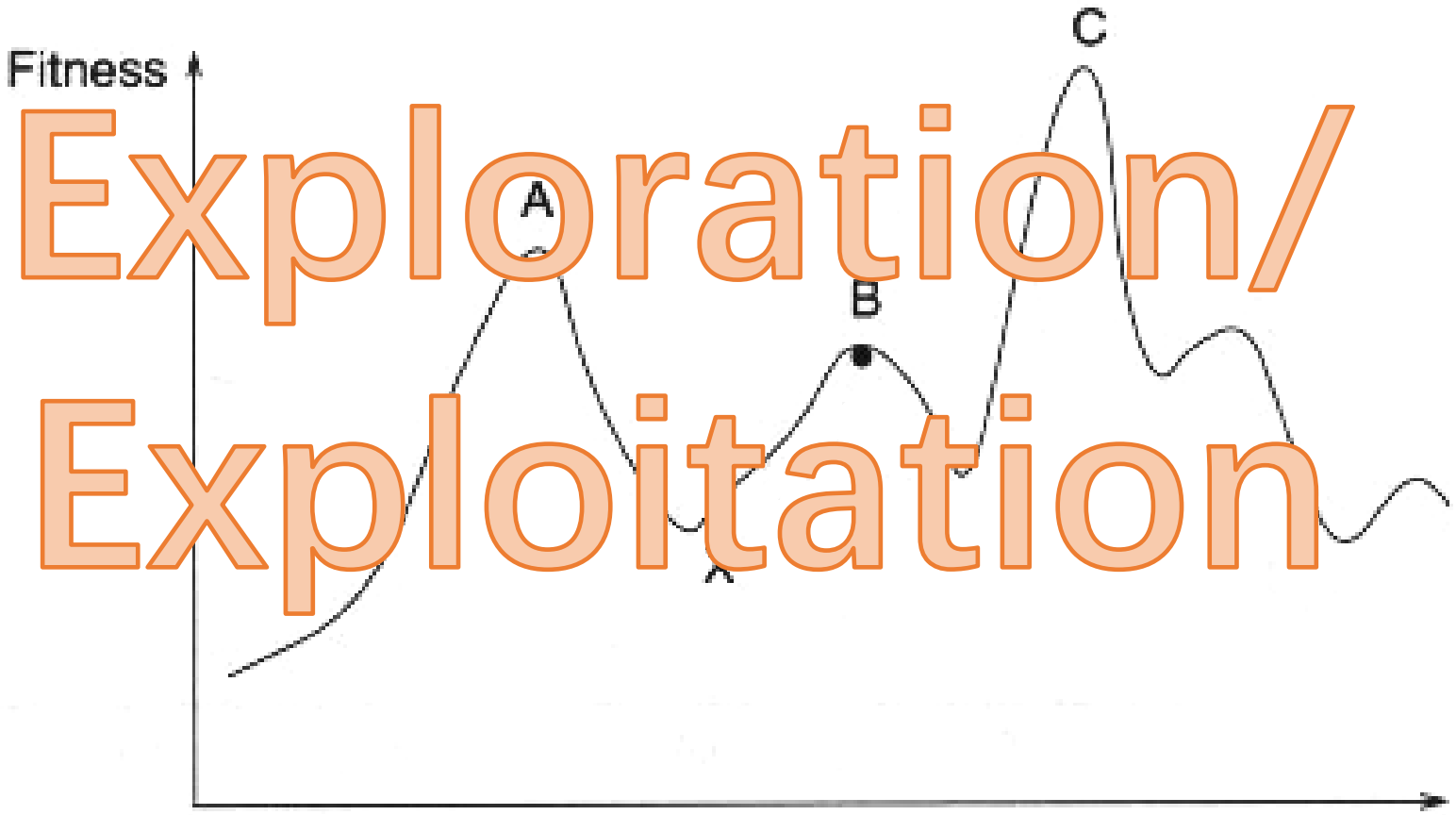
- Role: to generate new candidate solutions
- Usually divided into two types according to their **arity** (number of inputs to the variation operator):
  - Arity 1 : **mutation** operators
  - Arity >1 : **recombination** operators
  - Arity = 2 typically called **crossover**
  - Arity > 2 is formally possible, seldom used in EC

Main EA components: Mutation

- Role: cause small, random variance to a genotype
- Element of **randomness is essential** and differentiates it from other unary heuristic operators

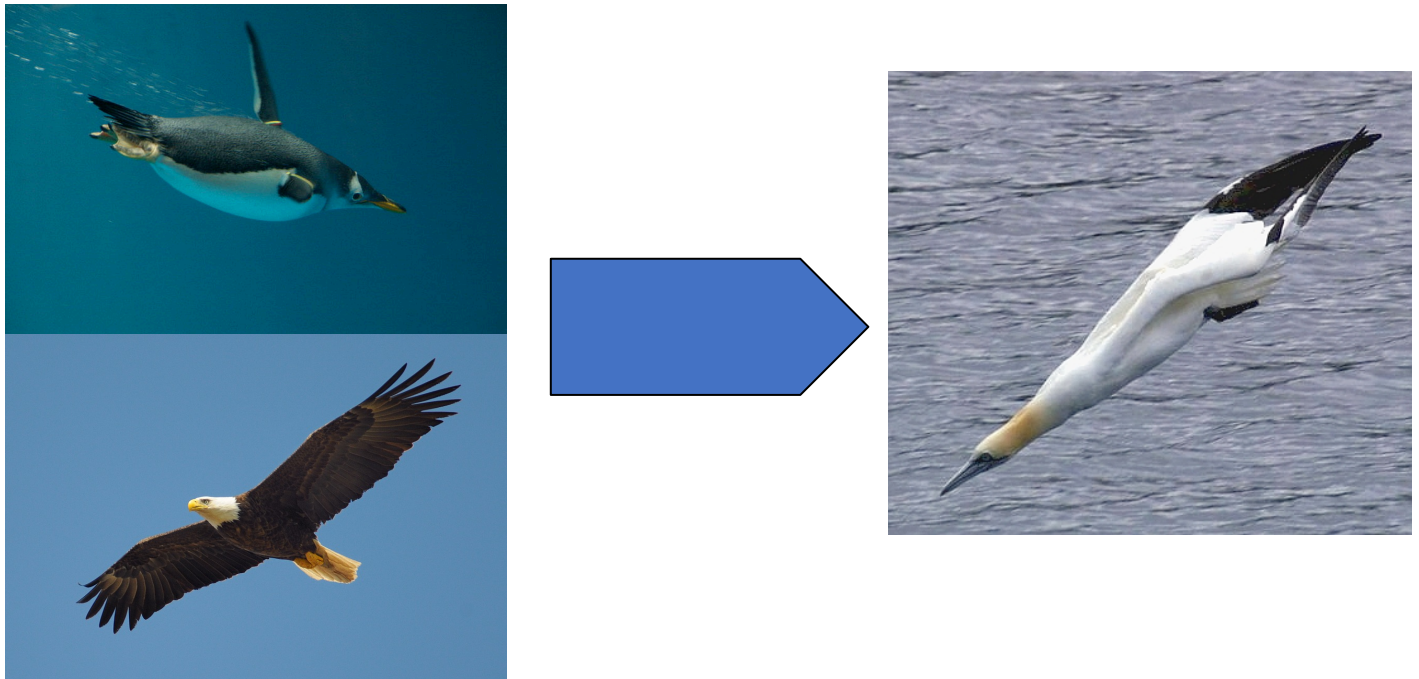


Why do we do Random Mutation?

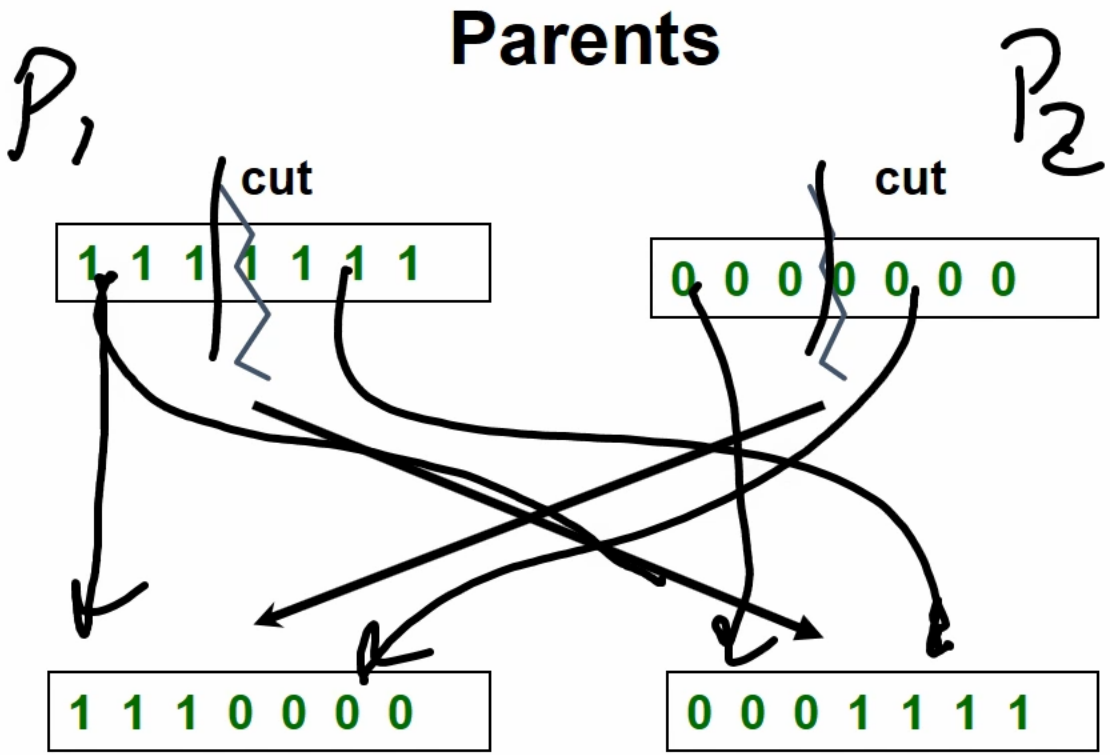


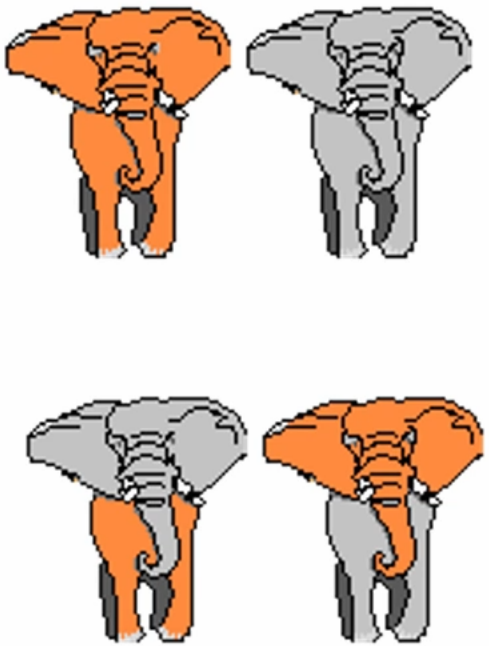
Main EA components: Recombination (1/2)

- Role: merges information from parents into offspring
- Choice of what information to merge is stochastic
- Hope is that some offspring are better by combining elements of genotypes that lead to good traits



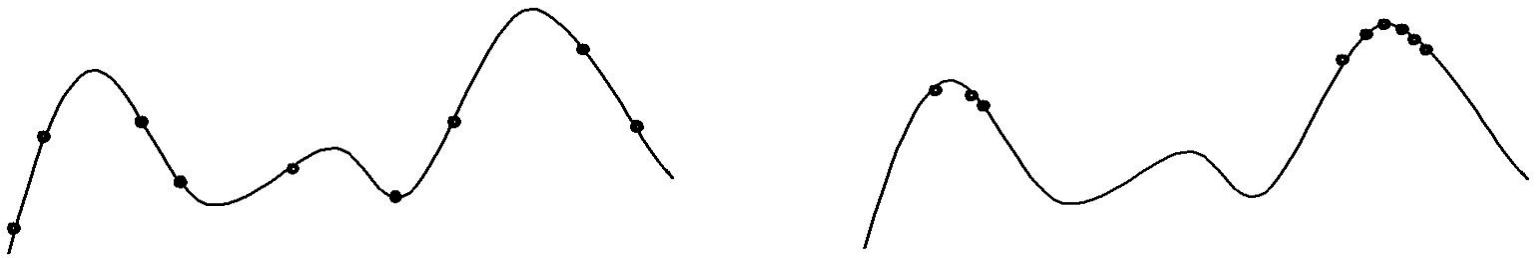
Main EA components: Recombination (2/2)



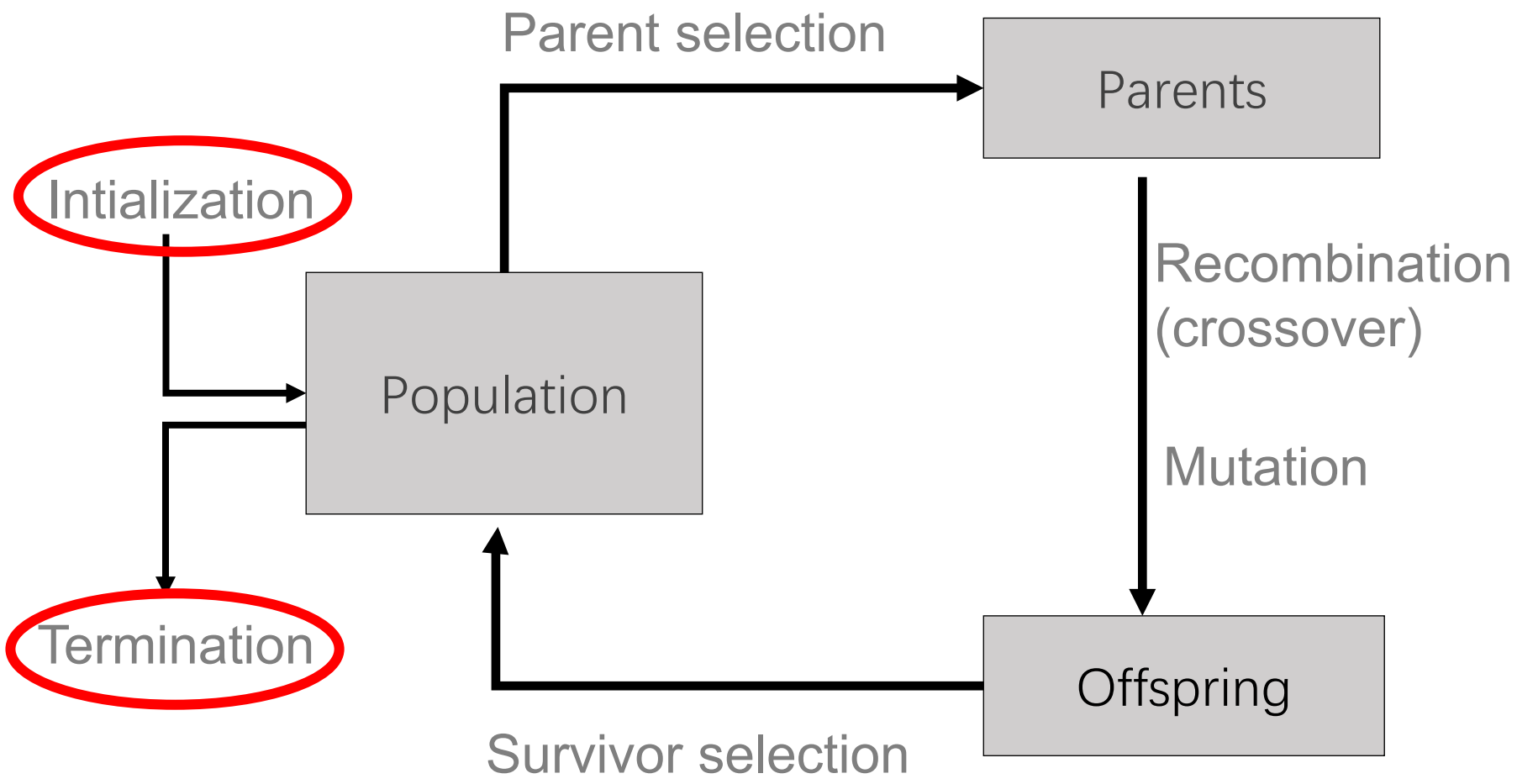


Crossover and/or mutation?

- **Crossover** is **explorative**, it makes a *big* jump to an area somewhere “in between” two (parent) areas
- **Mutation** is **exploitative**, it creates random *small* diversions, thereby staying near (in the area of) the parent



General scheme of EAs



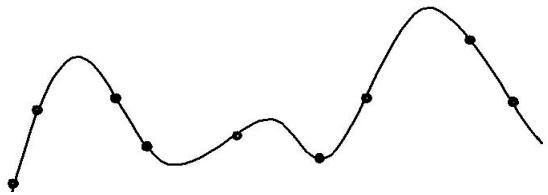
Main EA components: Initialisation / Termination

- Initialisation usually done at random,
  - Need to ensure even spread and mixture of possible allele values
  - Can include existing solutions, or use problem-specific heuristics, to “seed” the population

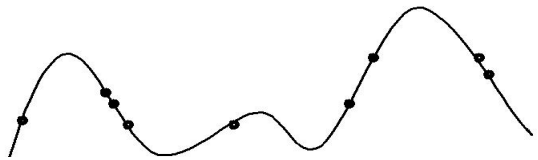
- Termination condition checked every generation
  - Reaching some (known/hoped for) fitness
  - Reaching some maximum allowed number of generations
  - Reaching some minimum level of diversity
  - Reaching some specified number of generations without fitness improvement

Typical EA behaviour: Stages

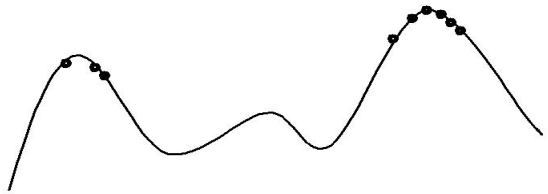
Stages in optimising on a 1-dimensional fitness landscape



Early stage:
quasi-random population distribution



Mid-stage:
population arranged around/on hills

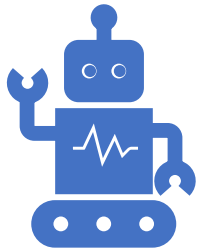


Late stage:
population concentrated on high hills



UiO : University of Oslo

institutt for informatikk



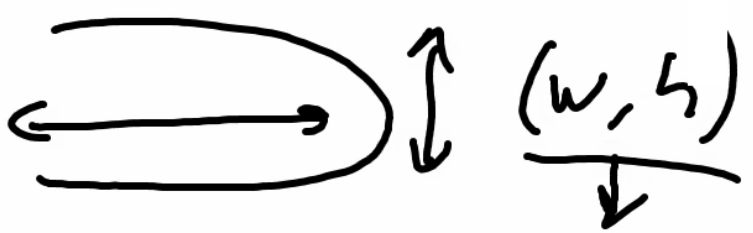
IN3050/IN4050, Lecture 3
Evolutionary algorithms 1

4: Binary, integer and real-valued representations

Kai Olav Ellefsen

Next video: Permutation and tree-based representations

Chapter 4: Representation, Mutation, and Recombination



- Role of **representation** and **variation operators**

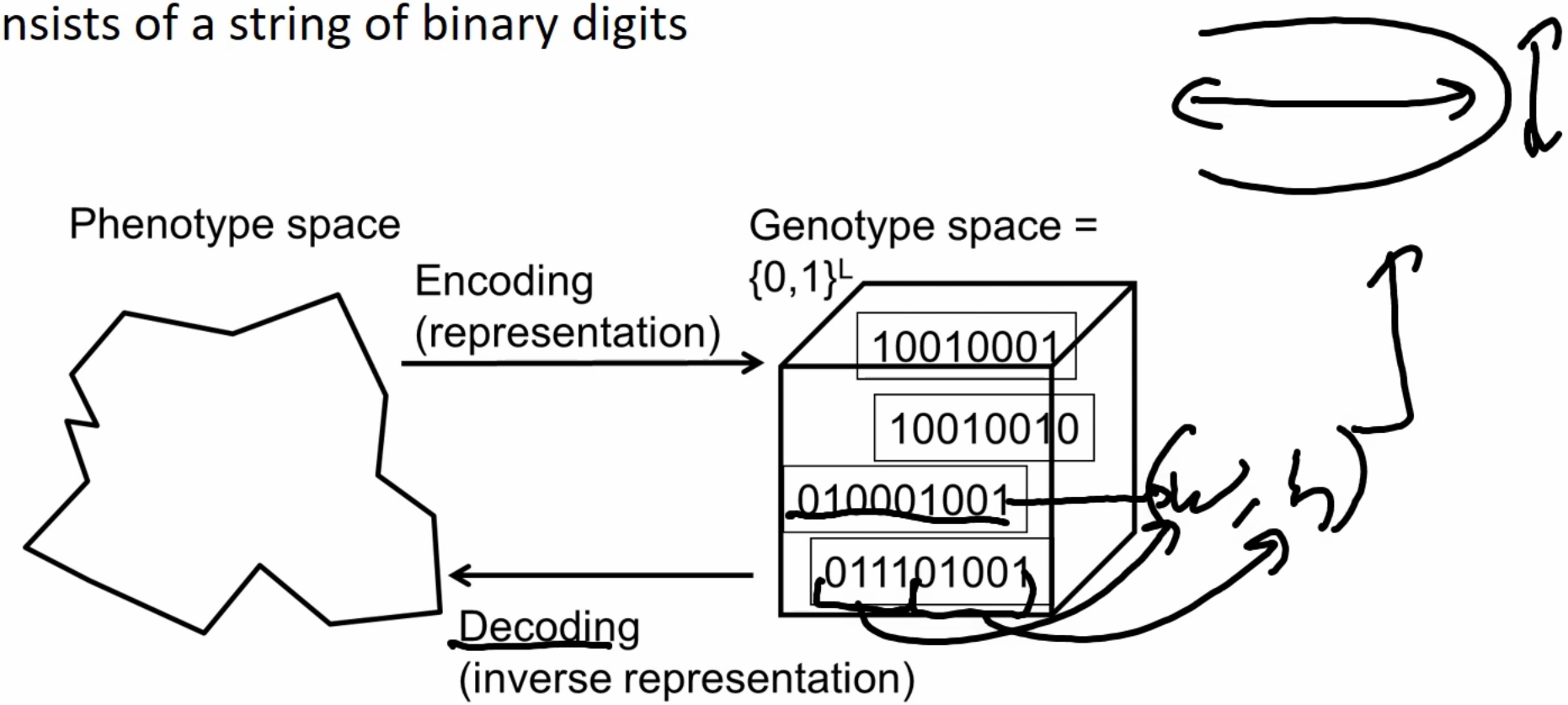
- Most common representation of genomes:
  - Binary
  - Integer
  - Real-Valued or Floating-Point
  - Permutation
  - Tree

Role of representation and variation operators

- First stage of building an EA and most difficult one: choose *right* representation for the problem
- Type of variation operators needed depends on chosen representation

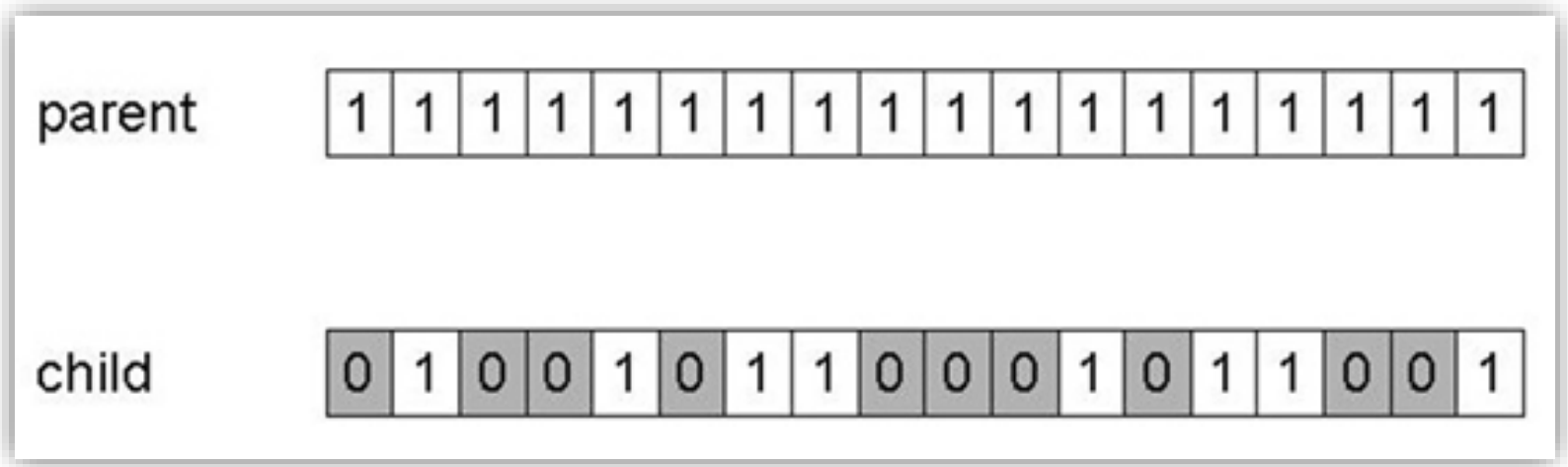
Binary Representation

- One of the earliest representations
- Genotype consists of a string of binary digits



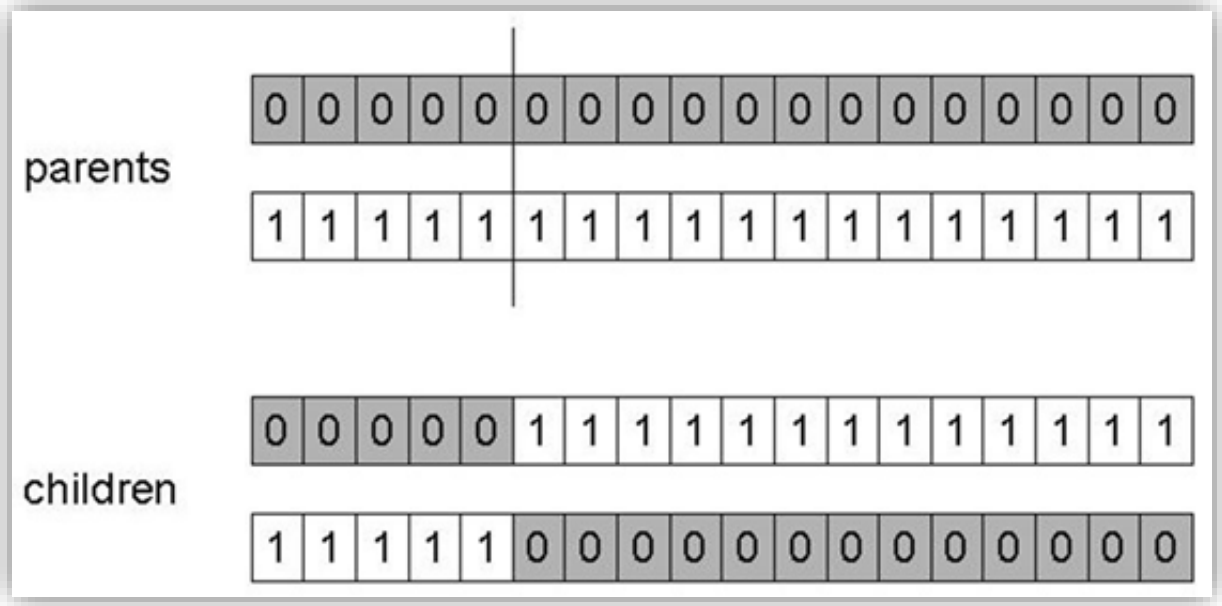
Binary Representation: Mutation

- Alter each gene independently with a probability p_m
- p_m is called the mutation rate



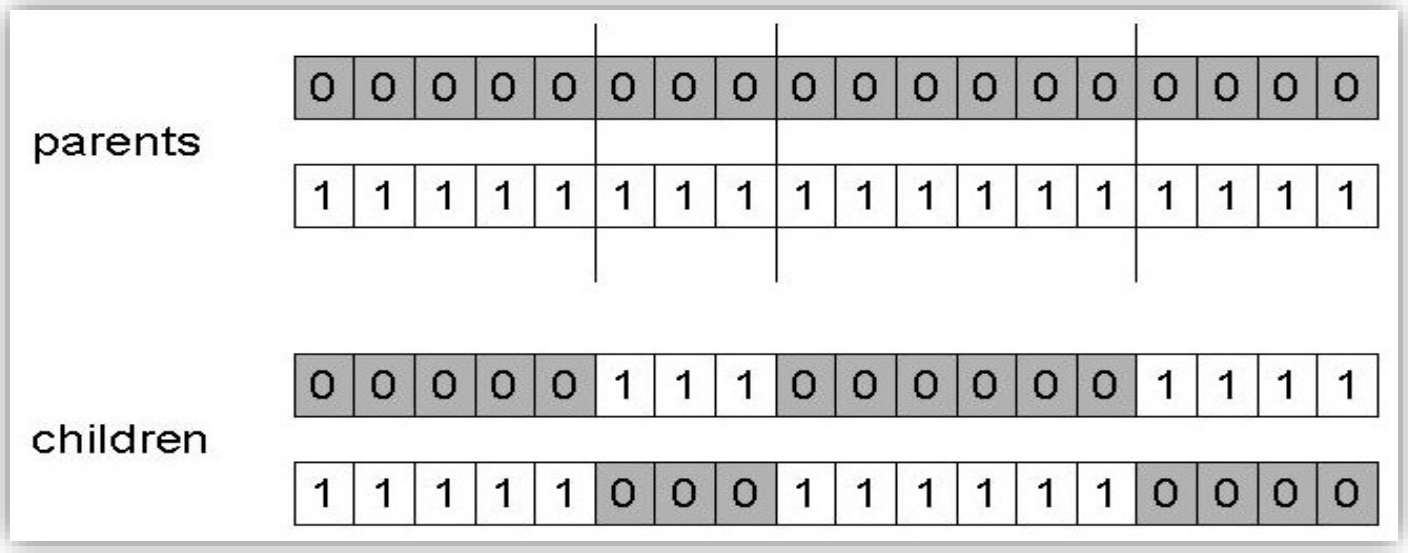
Binary Representation: 1-point crossover

- Choose a random point on the two parents
- Split parents at this crossover point
- Create children by exchanging tails



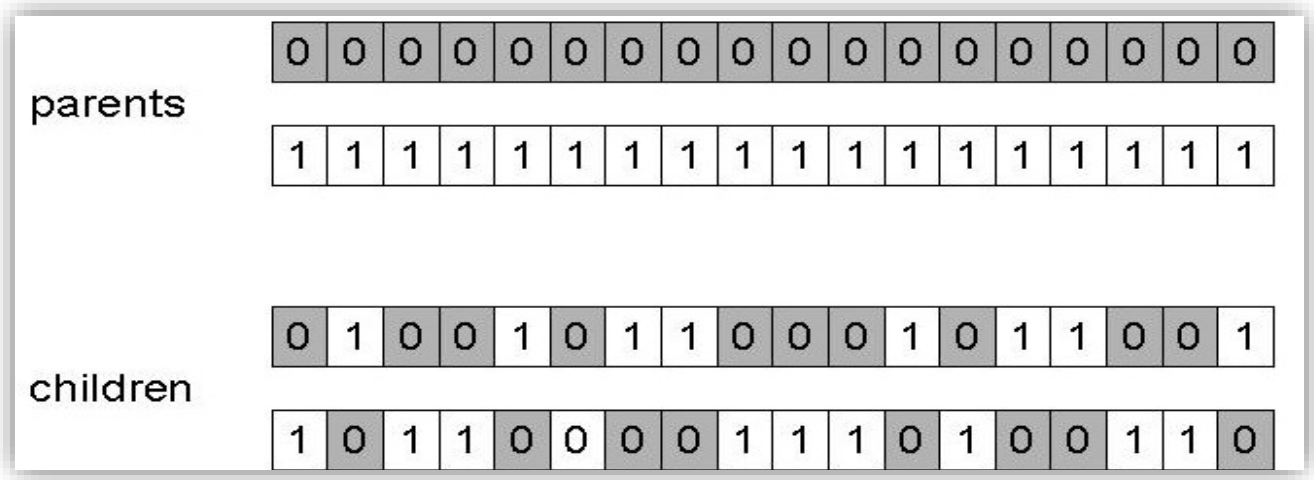
Binary Representation: n-point crossover

- Choose n random crossover points
- Split along those points
- Glue parts, alternating between parents

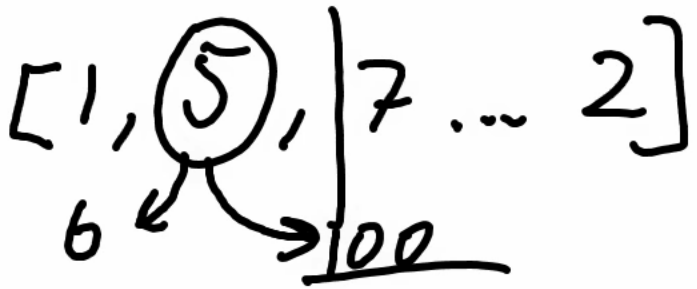


Binary Representation: Uniform crossover

- Assign 'heads' to one parent, 'tails' to the other
- Flip a coin for each gene of the first child
- Make an inverse copy of the gene for the second child
- Breaks more “links” in the genome



Integer Representation



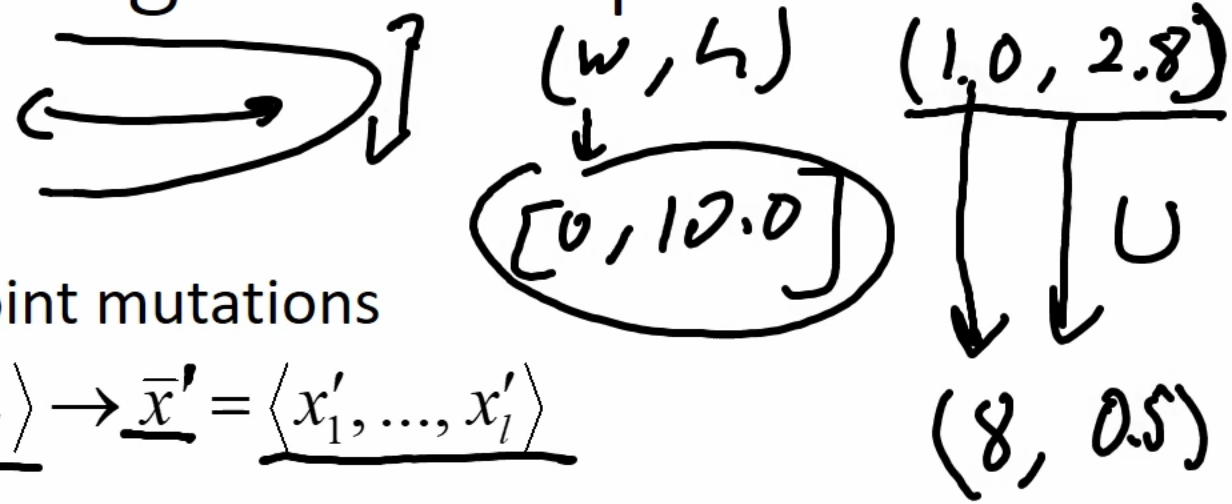
- Some problems naturally have integer variables,
  - e.g. image processing parameters

[blue, blue, blue, pink]
[1 1 1 4]

- Others take categorical values from a fixed set
  - e.g. {blue, green, yellow, pink}
    1 2 3 4
- N-point / uniform crossover operators work

- Extend bit-flipping mutation to make:
  - "creep" i.e. more likely to move to similar value
    - Adding a small (positive or negative) value to each gene with probability p
  - Random resetting (esp. categorical variables)
    - With probability p_m a new value is chosen at random

Real-Valued or Floating-Point Representation: Uniform Mutation



- General scheme of floating point mutations

$$\bar{x} = \langle x_1, \ldots, x_l \rangle \rightarrow \bar{x}' = \langle x'_1, \ldots, x'_l \rangle$$

$$x_i, x'_i \in [LB_i, UB_i]$$

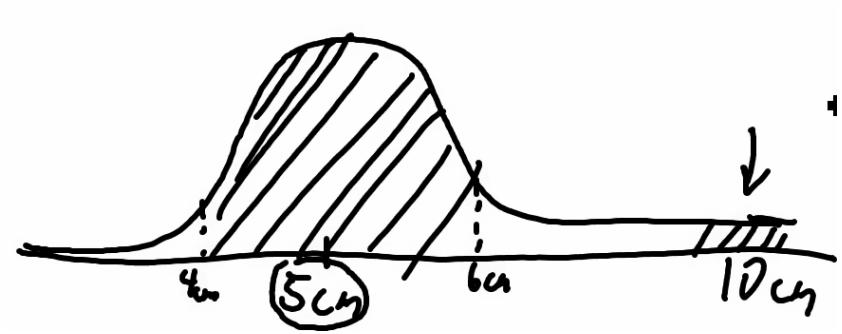
- **Uniform Mutation:** x'_i drawn randomly (uniform) from $[LB_i, UB_i]$

- Analogous to bit-flipping (binary) or random resetting (integers)

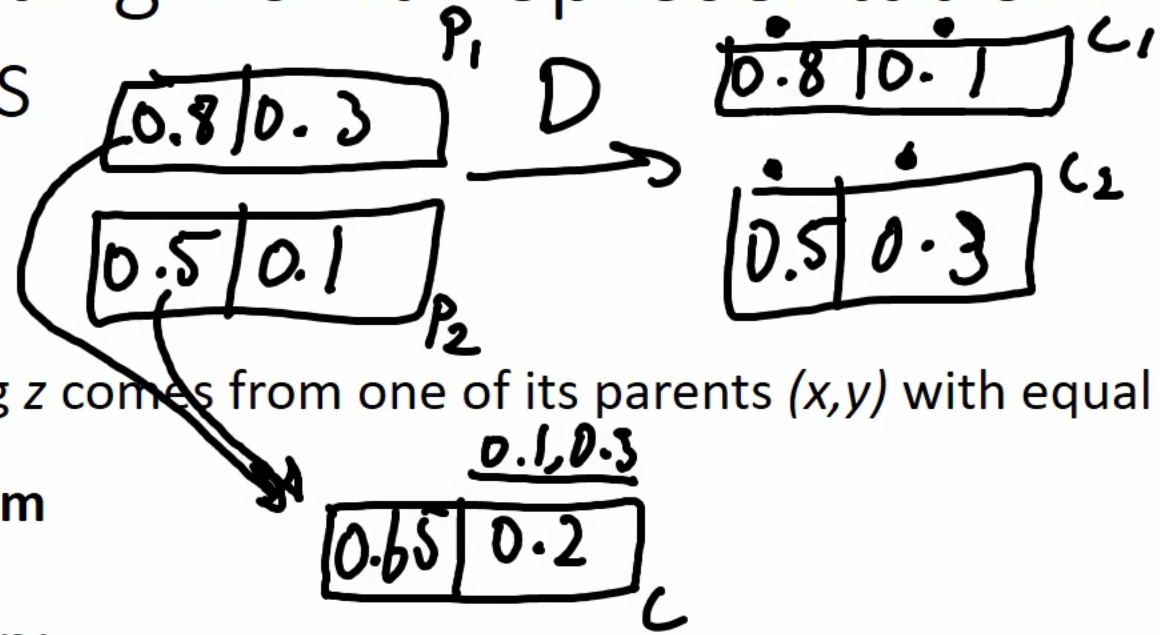
Real-Valued or Floating-Point Representation: Nonuniform Mutation

- Non-uniform mutations:
  - Most common method is to add random deviate to each variable separately, taken from N(0, σ) **Gaussian distribution** and then curtail to range

    $x'_i = x_i + N(0,\sigma)$
  - Standard deviation σ, **mutation step size**, controls amount of change (2/3 of drawings will lie in range (- σ to + σ))



Real-Valued or Floating-Point Representation: Crossover operators



- Discrete recombination:
  - each allele value in offspring *z* comes from one of its parents *(x,y)* with equal probability: $z_i = x_i$ or y_i
  - Could use **n-point** or **uniform**

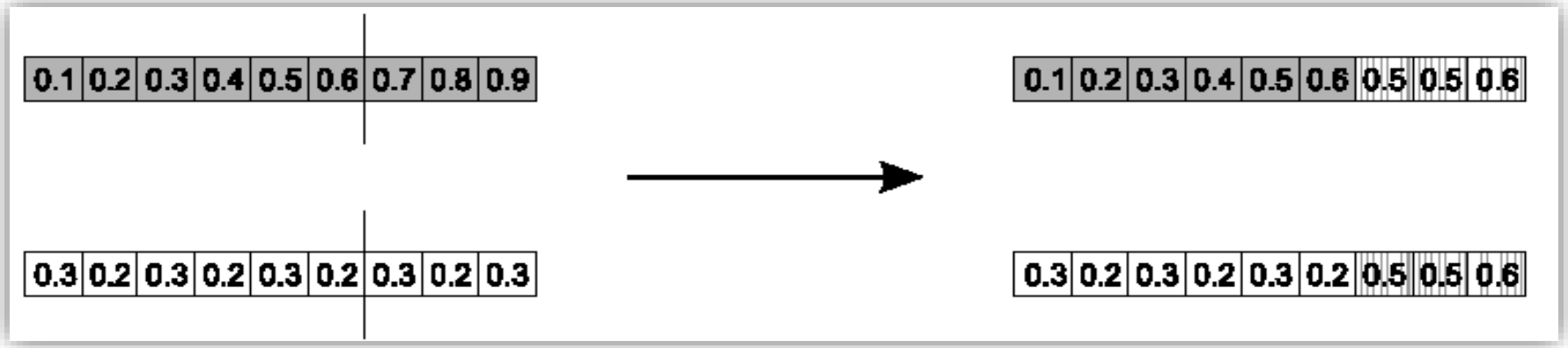
- Intermediate recombination:
  - exploits idea of creating children "between" parents (hence a.k.a. *arithmetic* recombination)
  - $z_i = \alpha x_i + (1 - \alpha) y_i$ where $\alpha : 0 \leq \alpha \leq 1$.
  - The parameter α can be:
    - constant: $\alpha = 0.5$ -> uniform arithmetical crossover
    - variable (e.g. depend on the age of the population)
    - picked at random every time

Real-Valued or Floating-Point Representation: Simple arithmetic crossover

- Parents: $\langle x_1,\cdots,x_n \rangle$ and $\langle y_1,\cdots,y_n \rangle$
- Pick a random gene (k) after this point mix values
- $child_1$ is:

$$\left\langle x_1,...,x_k,\alpha\cdot y_{k+1}+(1-\alpha)\cdot x_{k+1},...,\alpha\cdot \mathrm{y}_{\mathrm{n}}+(1-\alpha)\cdot x_n \right\rangle$$

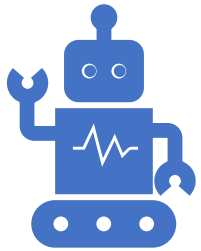
- reverse for other child. e.g. with $\alpha = 0.5$





UiO : University of Oslo

institutt for informatikk



IN3050/IN4050, Lecture 3
Evolutionary algorithms 1

5: Permutation and tree-based representations

Kai Olav Ellefsen

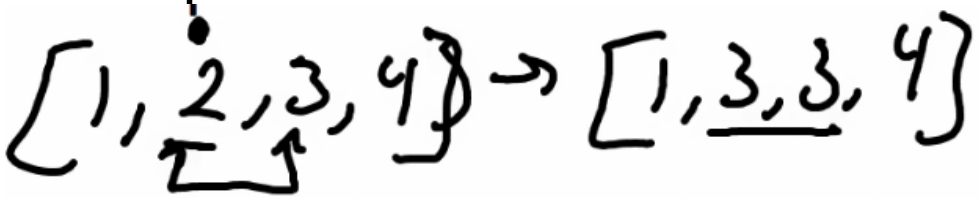
Next video: Example of a simple evolutionary algorithm

Permutation Representations

- Useful in ordering/sequencing problems
- Task is (or can be solved by) arranging some objects in a certain order. Examples:
  - production scheduling: important thing is which elements are scheduled before others (order)
  - Travelling Salesman Problem (TSP) : important thing is which elements occur next to each other (adjacency)
- if there are *n* variables then the representation is as a list of *n* integers, each of which occurs exactly once

[1, 2, 3, 4] [1, 3, 2, 4]

Permutation Representations: Mutation



- Normal mutation operators lead to inadmissible solutions
  - Mutating a single gene destroys the permutation
- Therefore must change at least two values
- Mutation probability now reflects the probability that some operator is applied once to the whole string, rather than individually in each position

Permutation Representations: Swap mutation

- Pick two alleles at random and swap their positions



Permutation Representations: Insert Mutation

- Pick two allele values at random
- Move the second to follow the first, shifting the rest along to accommodate
- Note that this preserves most of the order and the adjacency information



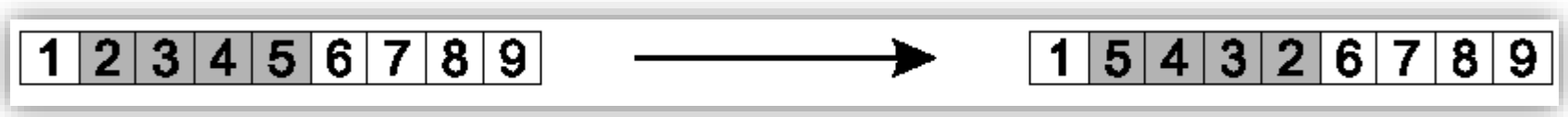
Permutation Representations: Scramble mutation

- Pick a subset of genes at random
- Randomly rearrange the alleles in those positions



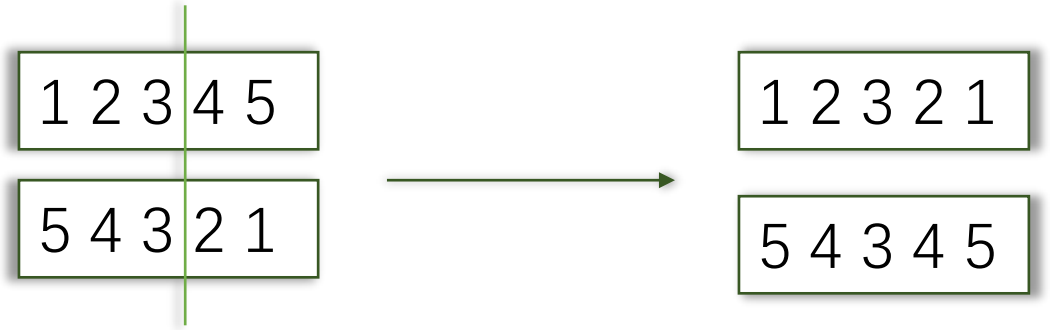
Permutation Representations: Inversion mutation

- Pick two alleles at random and then invert the substring between them.
- Preserves most adjacency information (only breaks two links) but disruptive of order information



Permutation Representations: Crossover operators

- "Normal" crossover operators will often lead to inadmissible solutions



- Many specialised operators have been devised which focus on combining order or adjacency information from the two parents

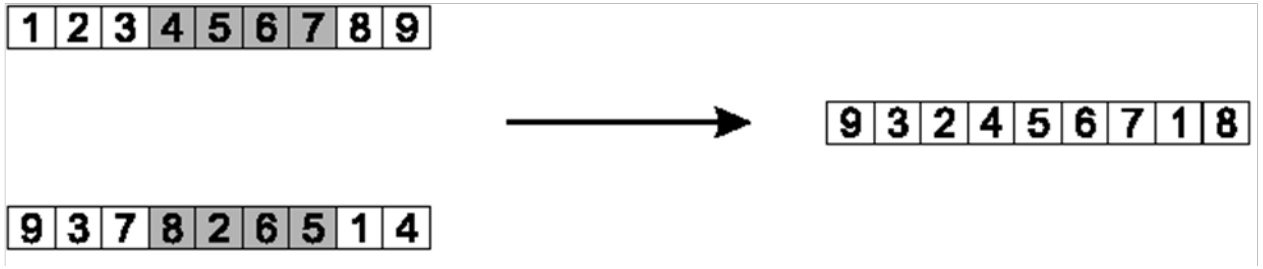
Permutation Representations: Conserving Adjacency

- Important for problems where adjacency between elements decides quality (e.g. TSP)



Permutation Representations: Conserving Adjacency

- Important for problems where adjacency between elements decides quality (e.g. TSP)
  - [1,2,3,4,5] is same plan as [5,4,3,2,1] -> order and position not important, but adjacency is.
- **Partially Mapped Crossover** and Edge Recombination are example operators



Permutation Representations: Conserving Order

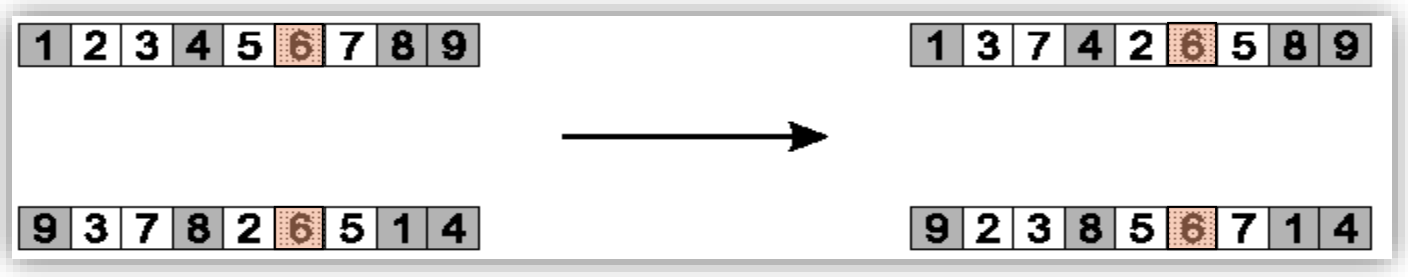
- Important for problems where **order** of elements decide performance (e.g. production scheduling)

Making breakfast:

1. **Start brewing coffee**
2. **Toast bread**
3. **Apply butter**
4. **Add jam**
5. **Pour hot coffee**

Permutation Representations: Conserving Order

- Important for problems order of elements decide performance (e.g. production scheduling)
  - Now, [1,2,3,4,5] is a very different plan than [5,4,3,2,1]
- Order Crossover and **Cycle Crossover** are example operators



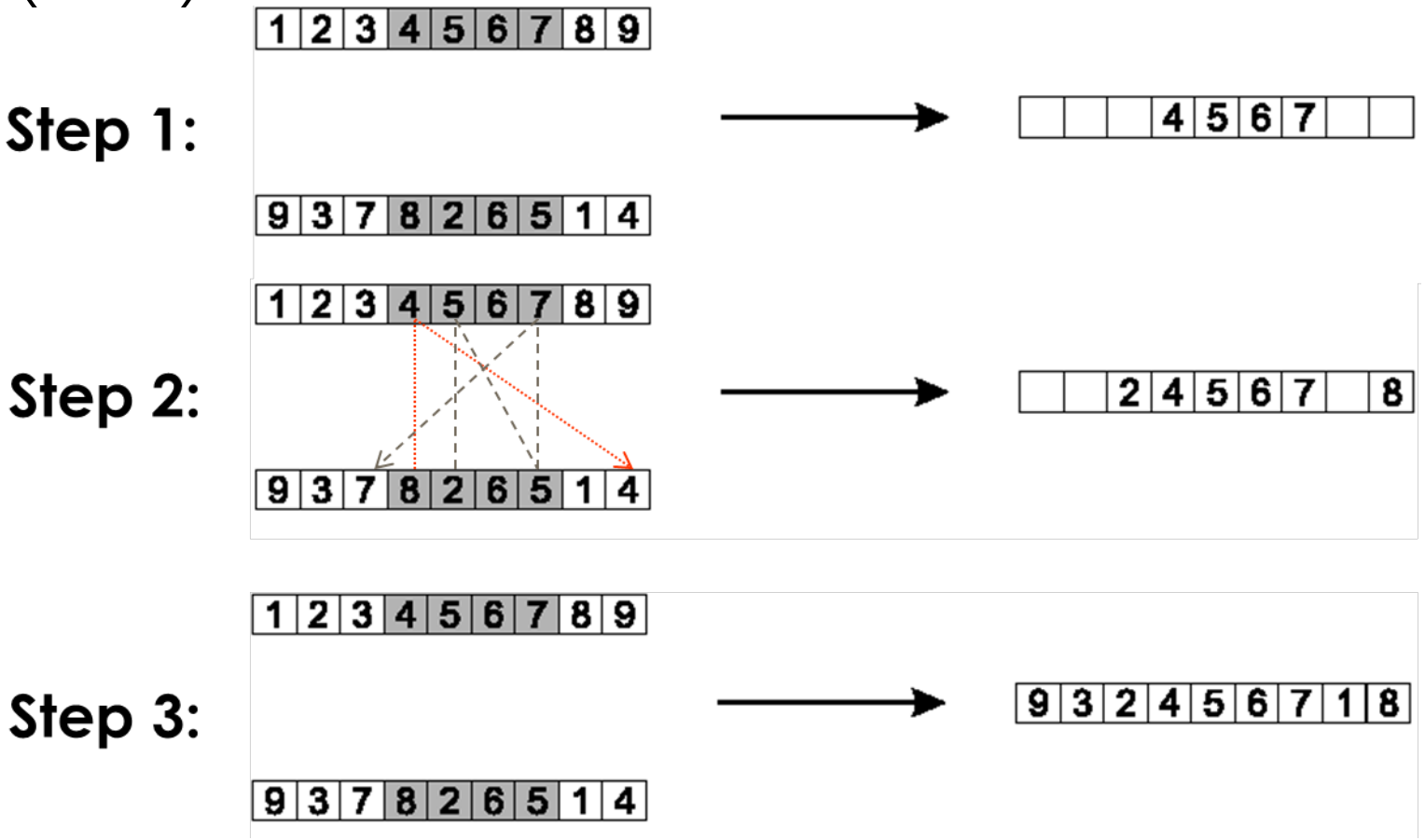
Permutation Representations: Partially Mapped Crossover (PMX) (1/2)

Informal procedure for parents P1 and P2:

1. Choose random segment and copy it from P1
2. Starting from the first crossover point look for elements in that segment of P2 that have not been copied
3. For each of these *i* look in the offspring to see what element *j* has been copied in its place from P1
4. Place *i* into the position occupied *j* in P2, since we know that we will not be putting *j* there (as is already in offspring)
5. If the place occupied by *j* in P2 has already been filled in the offspring *k*, put *i* in the position occupied by *k* in P2
6. Having dealt with the elements from the crossover segment, the rest of the offspring can be filled from P2.

Second child is created analogously

Permutation Representations: Partially Mapped Crossover (PMX) (2/2)



Permutation Representations: Edge Recombination (1/3)

- Works by constructing a table listing which edges are present in the two parents, if an edge is common to both, mark with a +
- e.g. [1 2 3 4 5 6 7 8 9] and [9 3 7 8 2 6 5 1 4]

Element	Edges	Element	Edges
1	2,5,4,9	6	2,5+,7
2	1,3,6,8	7	3,6,8+
3	2,4,7,9	8	2,7+, 9
4	1,3,5,9	9	1,3,4,8
5	1,4,6+		

Permutation Representations: Edge Recombination (2/3)

Informal procedure: once edge table is constructed

1. Pick an initial element, *entry*, at random and put it in the offspring
2. Set the variable *current element = entry*
3. Remove all references to *current element* from the table
4. Examine list for current element:
   - If there is a common edge, pick that to be next element
   - Otherwise pick the entry in the list which itself has the shortest list
   - Ties are split at random
5. In the case of reaching an empty list:
   - a new element is chosen at random

Permutation Representations: Edge Recombination (3/3)

Element	Edges	Element	Edges
1	2,5,4,9	6	2,5+,7
2	1,3,6,8	7	3,6,8+
3	2,4,7,9	8	2,7+, 9
4	1,3,5,9	9	1,3,4,8
5	1,4,6+		

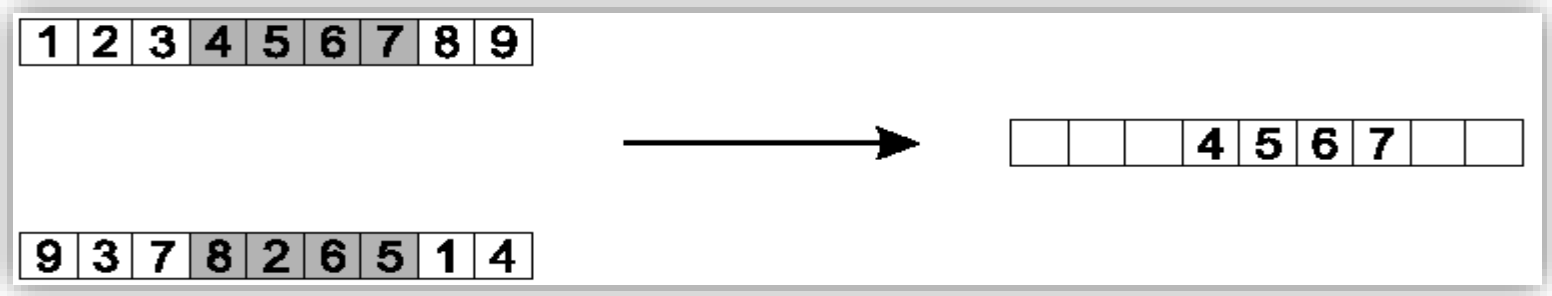
Choices	Element selected	Reason	Partial result
All	1	Random	[*1*]
2,5,4,9	5	Shortest list	[*1 5*]
4,6	6	Common edge	[*1 5 6*]
2,7	2	Random choice (both have two items in list)	[*1 5 6 2*]
3,8	8	Shortest list	[*1 5 6 2 8*]
7,9	7	Common edge	[*1 5 6 2 8 7*]
3	3	Only item in list	[*1 5 6 2 8 7 3*]
4,9	9	Random choice	[*1 5 6 2 8 7 3 9*]
4	4	Last element	[*1 5 6 2 8 7 3 9 4*]

Permutation Representations: Order crossover (1/2)

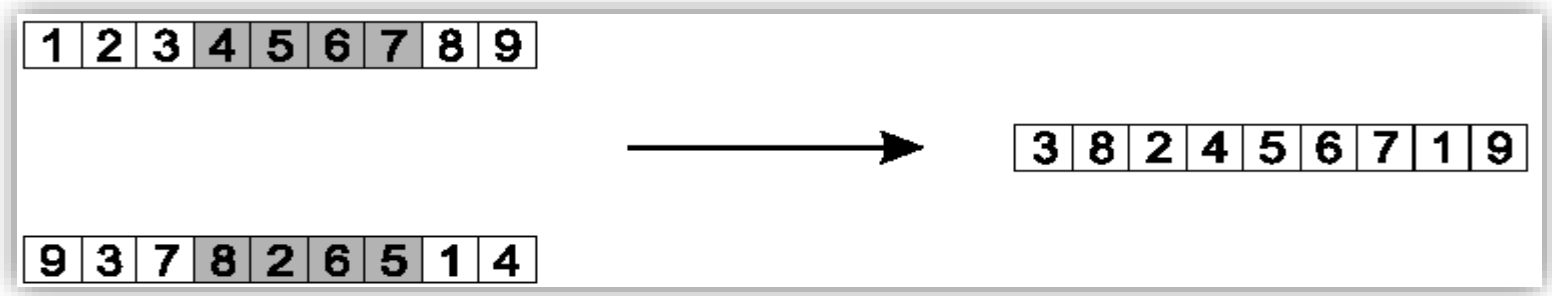
- Idea is to preserve relative order that elements occur
- Informal procedure:
  - 1. Choose an arbitrary part from the first parent
  - 2. Copy this part to the first child
  - 3. Copy the numbers that are not in the first part, to the first child:
    - starting right from cut point of the copied part,
    - using the **order** of the second parent
    - and wrapping around at the end
  - 4. Analogous for the second child, with parent roles reversed

Permutation Representations: Order crossover (2/2)

- Copy randomly selected set from first parent



- Copy rest from second parent in order 1,9,3,8,2



Permutation Representations: Cycle crossover (1/2)

Basic idea:

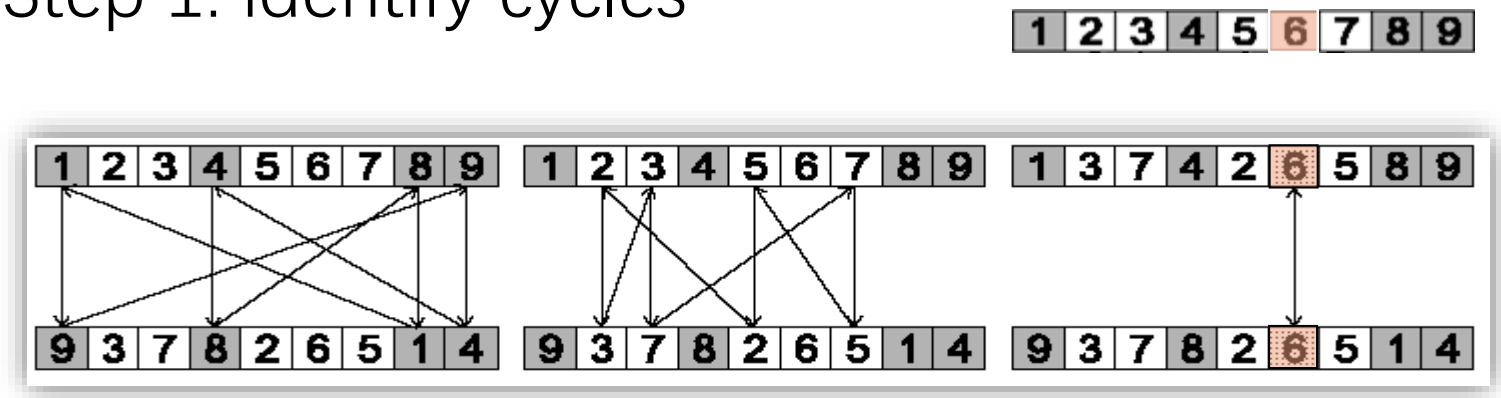
Each allele comes from one parent *together with its position*.

Informal procedure:

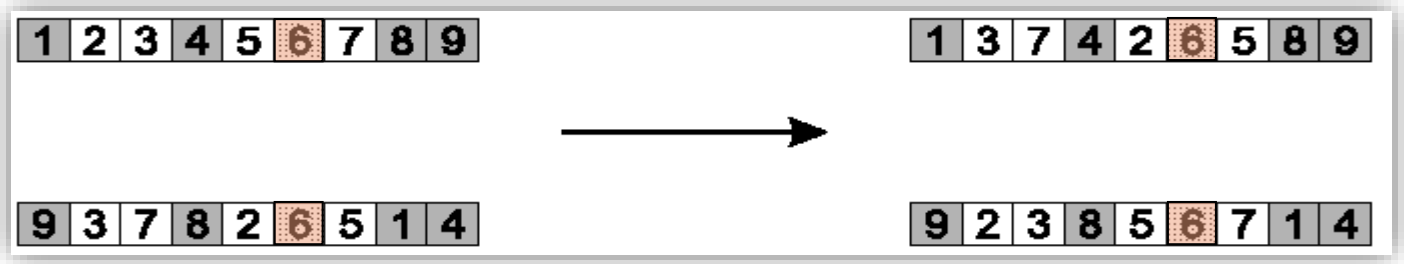
1. Make a cycle of alleles from P1 in the following way.
   (a) Start with the first allele of P1.
   (b) Look at the allele at the *same position* in P2.
   (c) Go to the position with the *same allele* in P1.
   (d) Add this allele to the cycle.
   (e) Repeat step b through d until you arrive at the first allele of P1.
2. Put the alleles of the cycle in the first child on the positions they have in the first parent.
3. Take next cycle from second parent

Permutation Representations: Cycle crossover (2/2)

- Step 1: identify cycles



- Step 2: copy **alternate** cycles into offspring



Genetic Programming: Tree Representation

- Trees are a universal form, e.g. consider

- Arithmetic formula:

$$2 \cdot \pi + \left((x+3) - \frac{y}{5+1} \right)$$

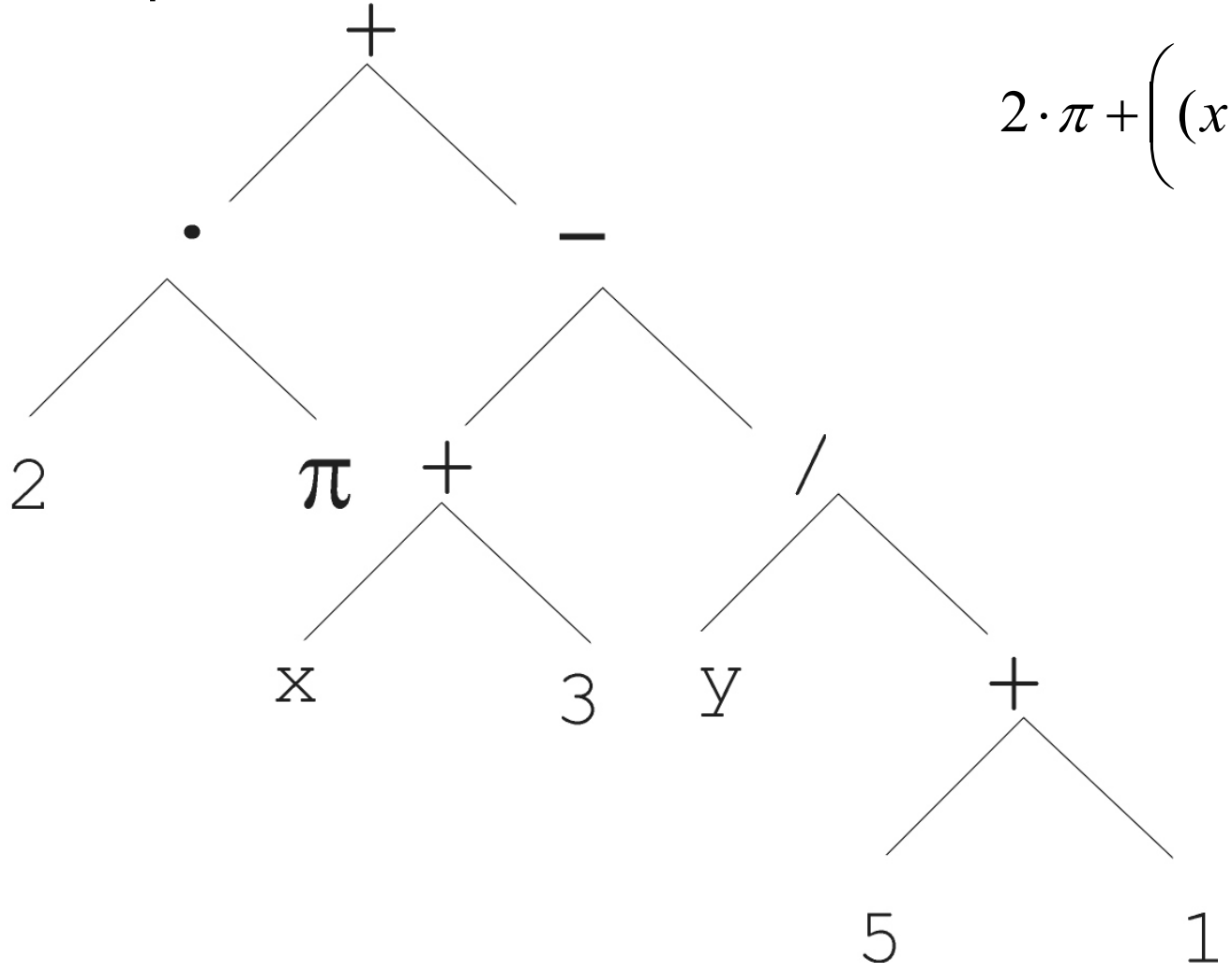
- Logical formula:

$(x \wedge \text{true}) \rightarrow ((x \vee y) \vee (z \leftrightarrow (x \wedge y)))$

- Program:

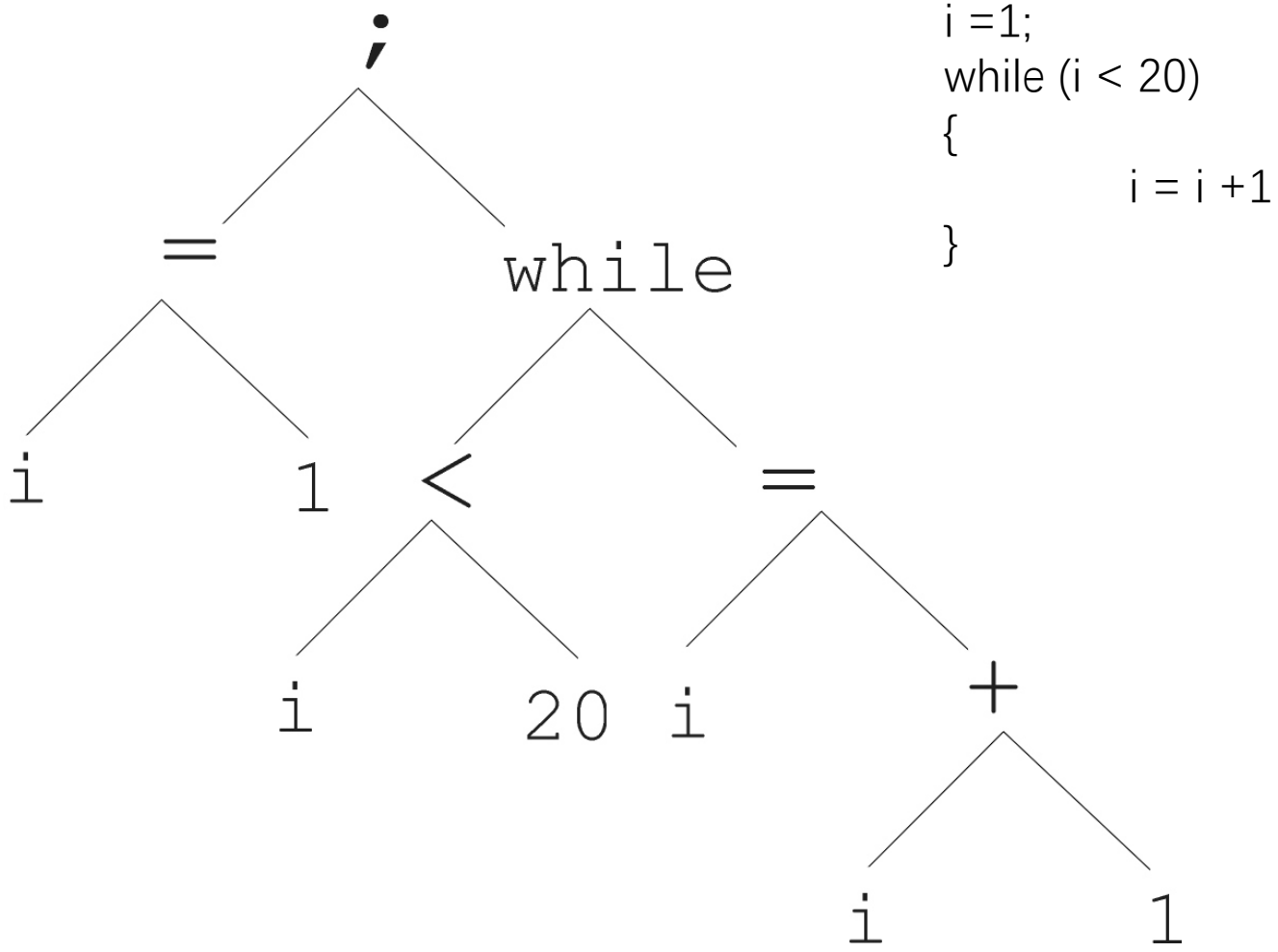
```
i =1;
while (i < 20)
{
        i = i +1
}
```

Genetic Programming: Tree Representation



$$2 \cdot \pi + \left((x+3) - \frac{y}{5+1} \right)$$

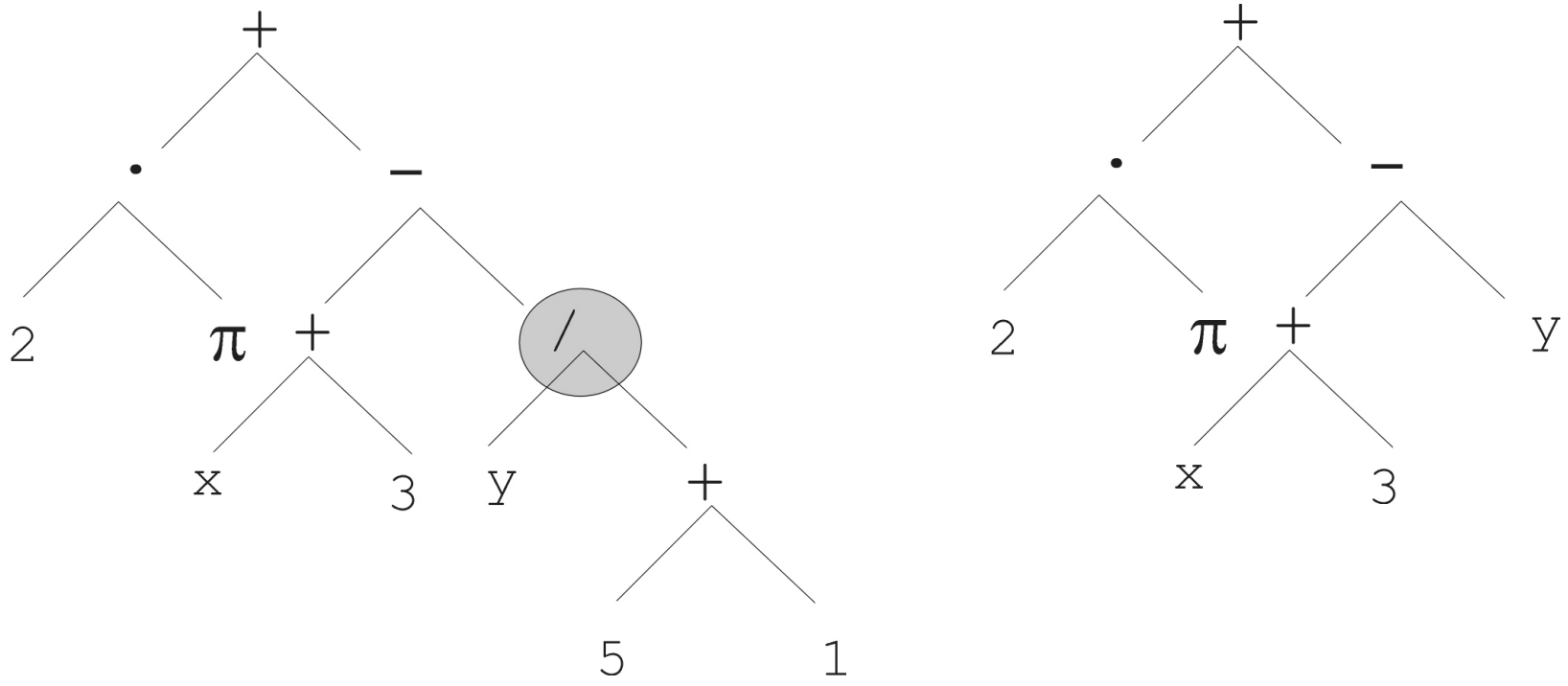
Genetic Programming: Tree Representation



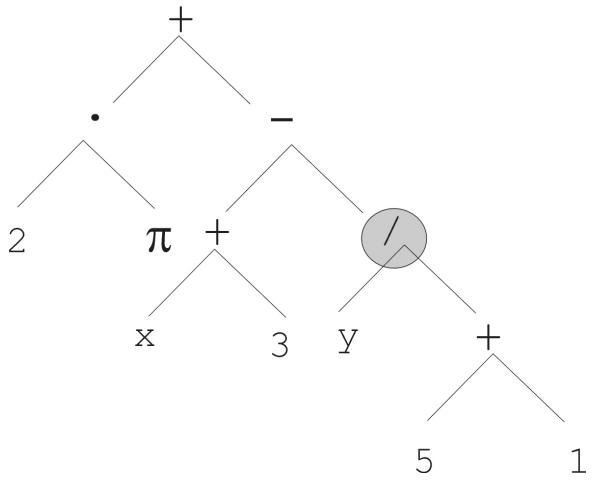

```
i =1;
while (i < 20)
{
        i = i +1
}
```

Genetic Programming: Mutation

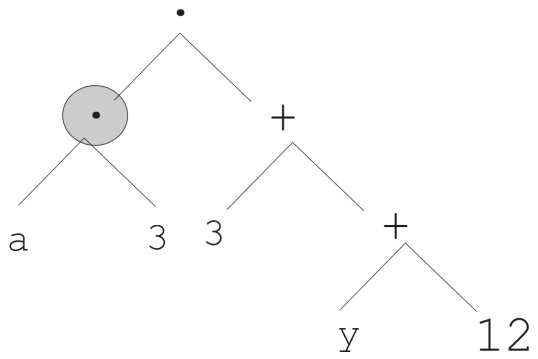
- Most common mutation: replace randomly chosen subtree by randomly generated tree



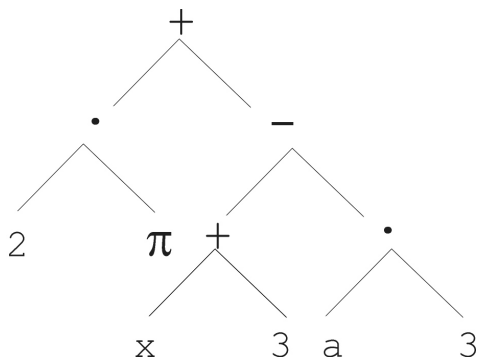
Genetic Programming: Recombination



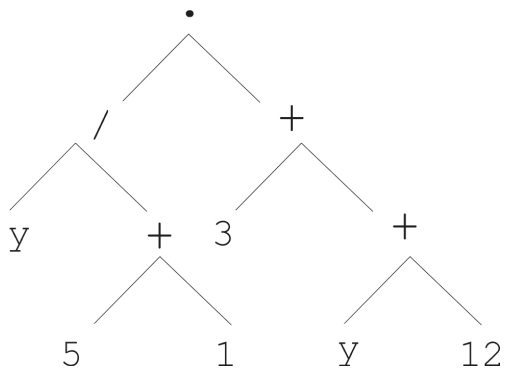
Parent 1



Parent 2



Child 1

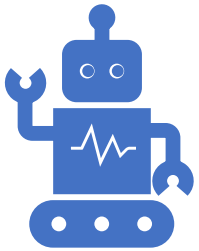


Child 2



UiO : University of Oslo

institutt for informatikk



IN3050/IN4050, Lecture 3
Evolutionary algorithms 1

6: Example of a simple evolutionary algorithm

Kai Olav Ellefsen

Next video: Example of a real-world evolutionary project

Genetic Algorithms: An example after Goldberg ’89

- Simple problem: max x^2 over {0,1,⋯,31}
- GA approach:
  - Representation: binary code, e.g., 01101 ↔ 13
  - Population size: 4
  - 1-point x-over, bitwise mutation
  - Roulette wheel selection
  - Random initialisation
- We show one generational cycle done by hand

X^2 example: Parent Selection

String no.	Initial population	x Value	Fitness $f(x) = x^2$	$Prob_i$	Expected count	Actual count
1	0 1 1 0 1	13	169	0.14	0.58	1
2	1 1 0 0 0	24	576	0.49	1.97	2
3	0 1 0 0 0	8	64	0.06	0.22	0
4	1 0 0 1 1	19	361	0.31	1.23	1
Sum			1170	1.00	4.00	4
Average			293	0.25	1.00	1
Max			576	0.49	1.97	2

X^2 example: Crossover

String no.	Mating pool	Crossover point	Offspring after xover	x Value	Fitness $f(x) = x^2$
1	0 1 1 0 | 1	4	0 1 1 0 0	12	144
2	1 1 0 0 | 0	4	1 1 0 0 1	25	625
2	1 1 | 0 0 0	2	1 1 0 1 1	27	729
4	1 0 | 0 1 1	2	1 0 0 0 0	16	256
Sum					1754
Average					439
Max					729

X^2 example: Mutation

String no.	Offspring after xover	Offspring after mutation	x Value	Fitness $f(x) = x^2$
1	0 1 1 0 0	1 1 1 0 0	26	676
2	1 1 0 0 1	1 1 0 0 1	25	625
2	1 1 0 1 1	1 1 0 1 1	27	729
4	1 0 0 0 0	1 0 1 0 0	18	324
Sum				2354
Average				588.5
Max				729