8 Representation, Mutation and Recombination Part 3: Permutations and Trees

Permutation Representations

- ordering and sequencing problems form a special type
- the task needs to be solved by arranging some objects in a certain order
- example: production scheduling:
  - important thing is which tasks are scheduled before others (order)
- example: Travelling Salesman Problem (TSP):
  - important thing is which elements occur next to each other (adjacency)
- since we only want each task to happen once, or each city to be visited once, we express these problems as a permutation
- if there are n variables then the representation is a list of n integers, each of which occurs exactly once

Permutation Representation: Example: Back to the TSP

problem:

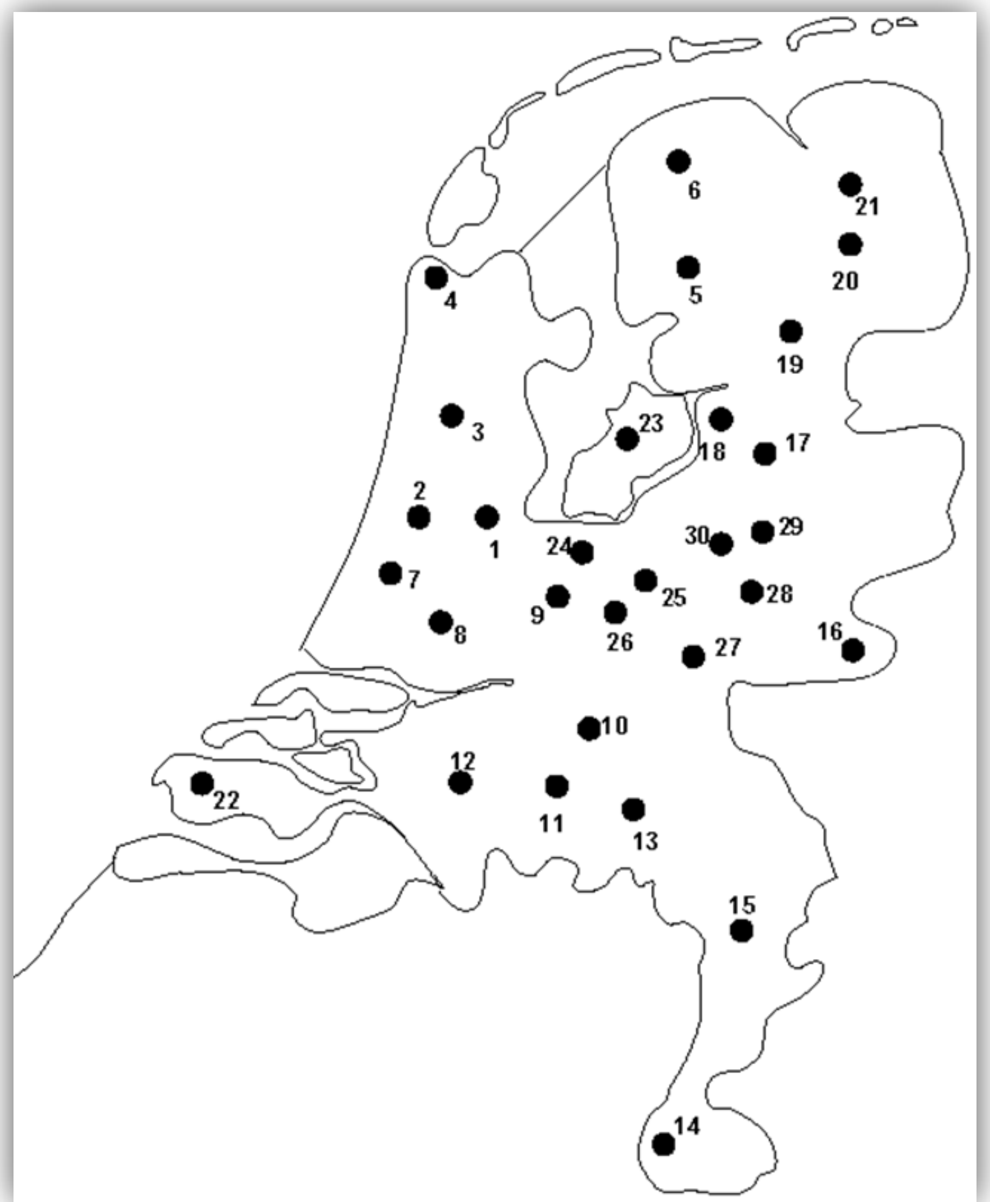
- given n cities, find a complete tour with minimal length

encoding:

- label the cities `1, 2, … , n`
- one complete tour is one permutation
- examples: `[1,2,3,4]`, `[3,4,2,1]`

HUGE search space:

- for 30 cities there are $30! \approx 10^{32}$ possible tours

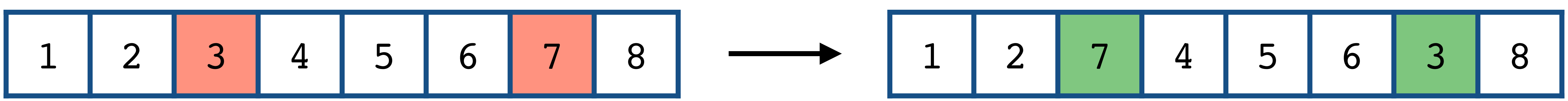


Permutation Representations: Mutation

- the mutation operators often used for other number representations lead to inadmissible solutions
- why?
- therefore we must change a set of gene values at the same time
- so the mutation parameter now reflects the probability that some operator is applied once to the whole genotype, rather than individually in each position

Permutation Representations: Swap Mutation

- pick two alleles at random and swap their positions



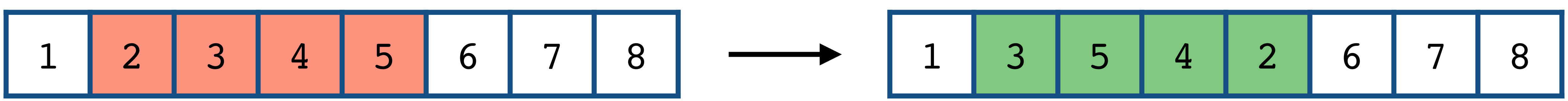
Permutation Representations: Insert Mutation

- pick two different genes at random
- move the second chosen to follow the first
- shift the rest along to accommodate
- this preserves most of the order and adjacency information

1	2	3	4	5	6	7	8	→	1	2	5	3	4	6	7	8
1	2	3	4	5	6	7	8	→	2	3	4	5	6	1	7	8

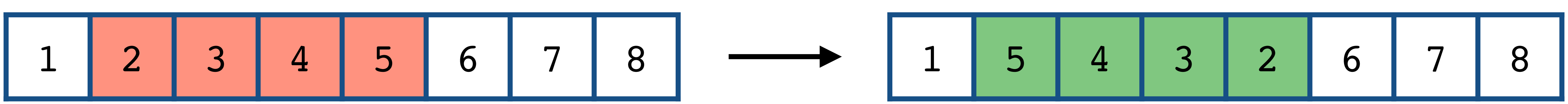
Permutation Representations: Scramble Mutation

- pick a random subset of genes
- randomly rearrange the alleles in those positions

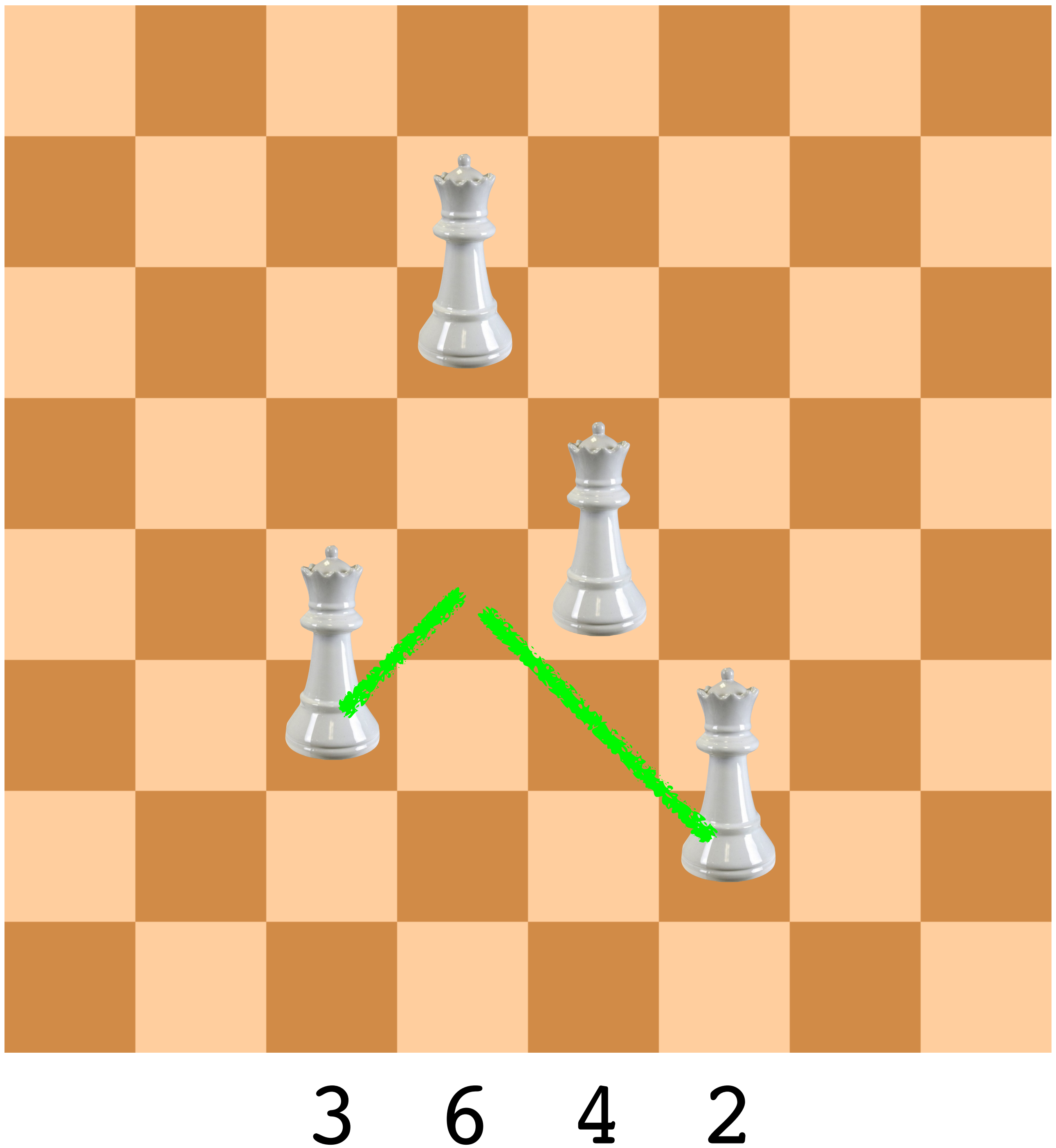


Permutation Representations: Inversion Mutation

- pick two alleles at random and invert the substring between them
- preserves most adjacency information
    - so good for TSP
- but disrupts order information
    - so bad for production scheduling



Swap versus Inversion Example: N-Queens



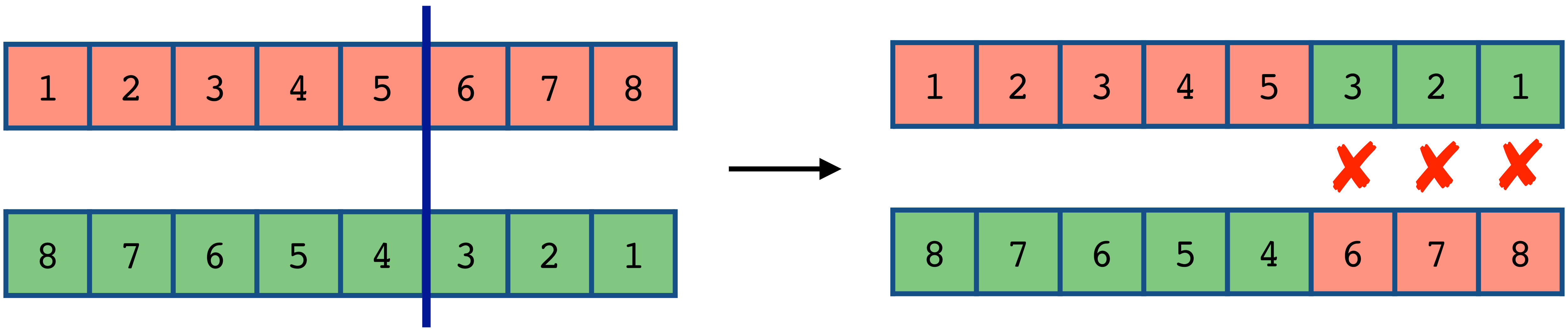
with an adjacency problem…

swap can damage a good sub-solution…

…while inversion can preserve it

Permutation Representations: Crossover Operators

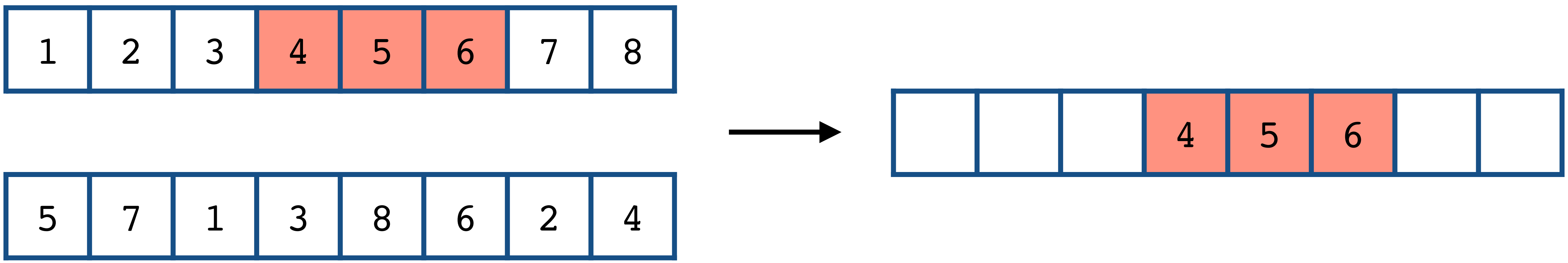
- normal crossover operators will often lead to inadmissible solutions:



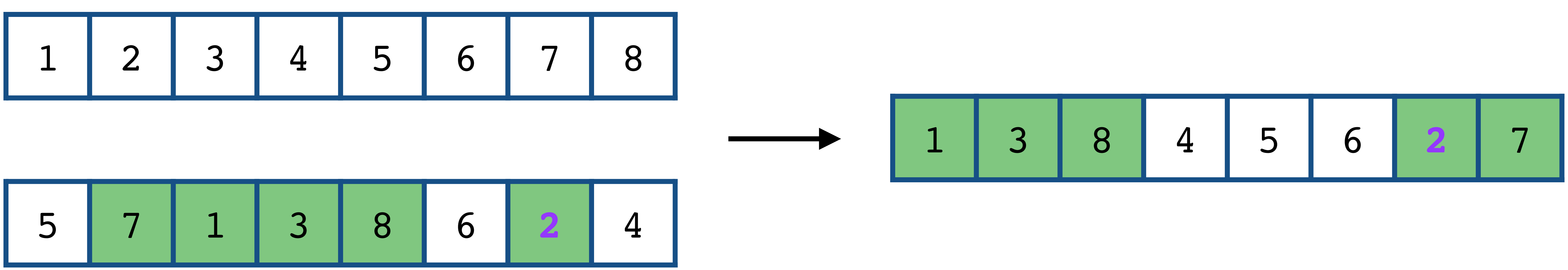
- so many specialised operators have been devised which focus on combining order or adjacency information from the two parents

Permutation Representations: Order 1 Crossover

- copy arbitrary part from first parent

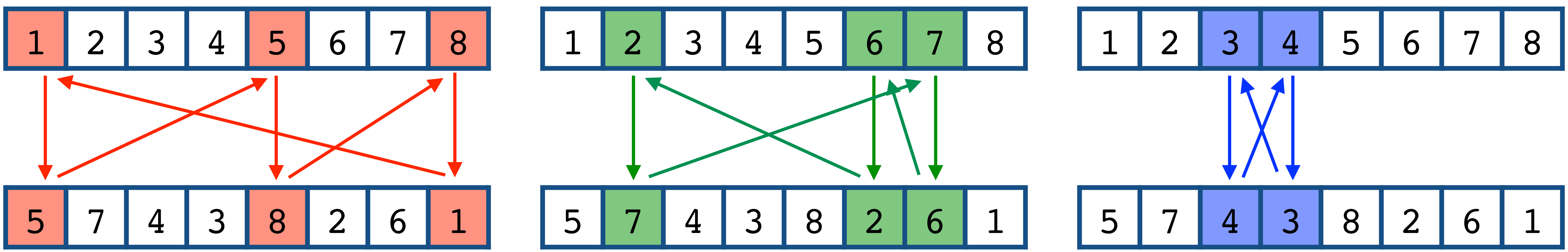


- copy rest of values from second parent in order, beginning at the ‘2’

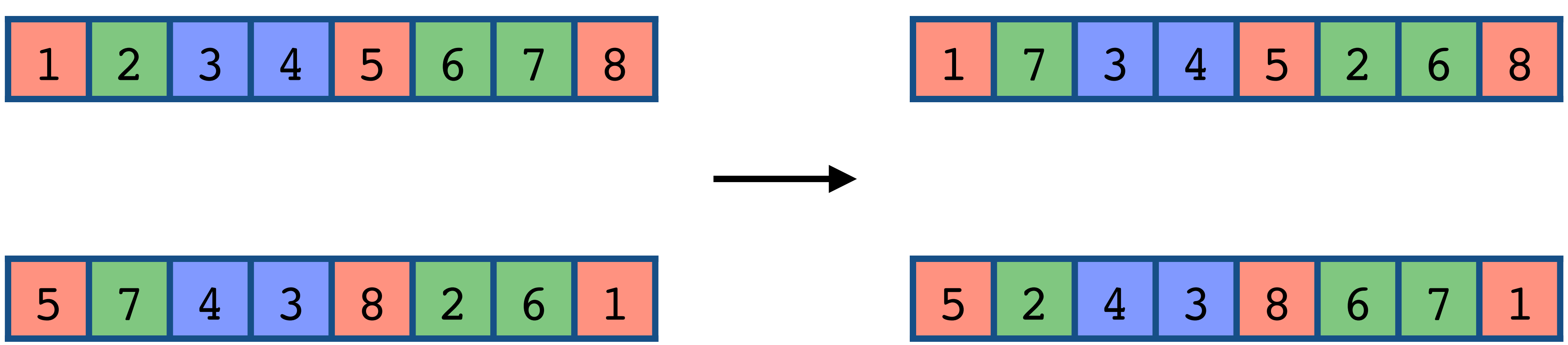


Permutation Representations: Cycle Crossover

- identify all cycles in parents:



- copy alternate cycles into offspring:



Permutation Representations: Edge Crossover

- intention: offspring should be created as far as possible using only the ‘edges’ present in one of the parents
- edges are when two values neighbour each other (wraparound)
- so intended for adjacency-based permutation problems
- creates only one child
- works by first constructing a table that lists which edges are present in the two parents
- if an edge is common to both parents it is marked with a ‘+’
- and common edges should be preserved in the child

Permutation Representations: Edge Crossover: Construct Edge Table

parent 1

1	2	3	4	5	6	7	8

parent 2

5	7	4	3	8	2	6	1

edge table

element	edges	element	edges
1	2,8,6,5	5	4,6,7,1
2	1,3,8,6	6	5,7,1,2
3	2,4+,8	7	6,8,5,4
4	3+,5,7	8	7,1,3,2

+ because 3 and 4 neighbour each other in both parents

Permutation Representations: Edge Crossover

- informal procedure:
    - construct the edge table (done ✔)
    - select an initial element, `entry`, at random and put it in the offspring
    - set the variable `current = entry`
    - remove all references to `current` from the lists of edges in the table
    - examine the list of edges for `current`:
        - if there is a common edge (marked with +), pick that to be the next current element
        - otherwise pick the entry in the list which itself has the shortest list of edges
        - ties are split at random
    - in the case of reaching an empty list:
        - choose a new current element at random

Permutation Representations: Edge Crossover: Create Child

element	edges	element	edges
1	2,8,6,5	**5**	**4,6,7,1**
2	1,3,8,6	6	5,7,1,2
3	2,4+,8	7	6,8,5,4
4	3+,5,7	8	7,1,3,2

choices	element selected	reason	partial result (child)
all	**5**	initial random choice	5
4,6,7,1	4	shortest list	5,4
3,5,7	3	common edge	5,4,3
2,8	8	random choice (tie)	5,4,3,8
7,1,2	7	shortest list	5,4,3,8,7
6	6	only choice	5,4,3,8,7,6
1,2	1	random choice (tie)	5,4,3,8,7,6,1
2	2	only choice	5,4,3,8,7,6,1,2

Permutation Representations: Edge Crossover: Parents versus Child

parent 1

1	2	3	4	5	6	7	8

parent 2

5	7	4	3	8	2	6	1

child

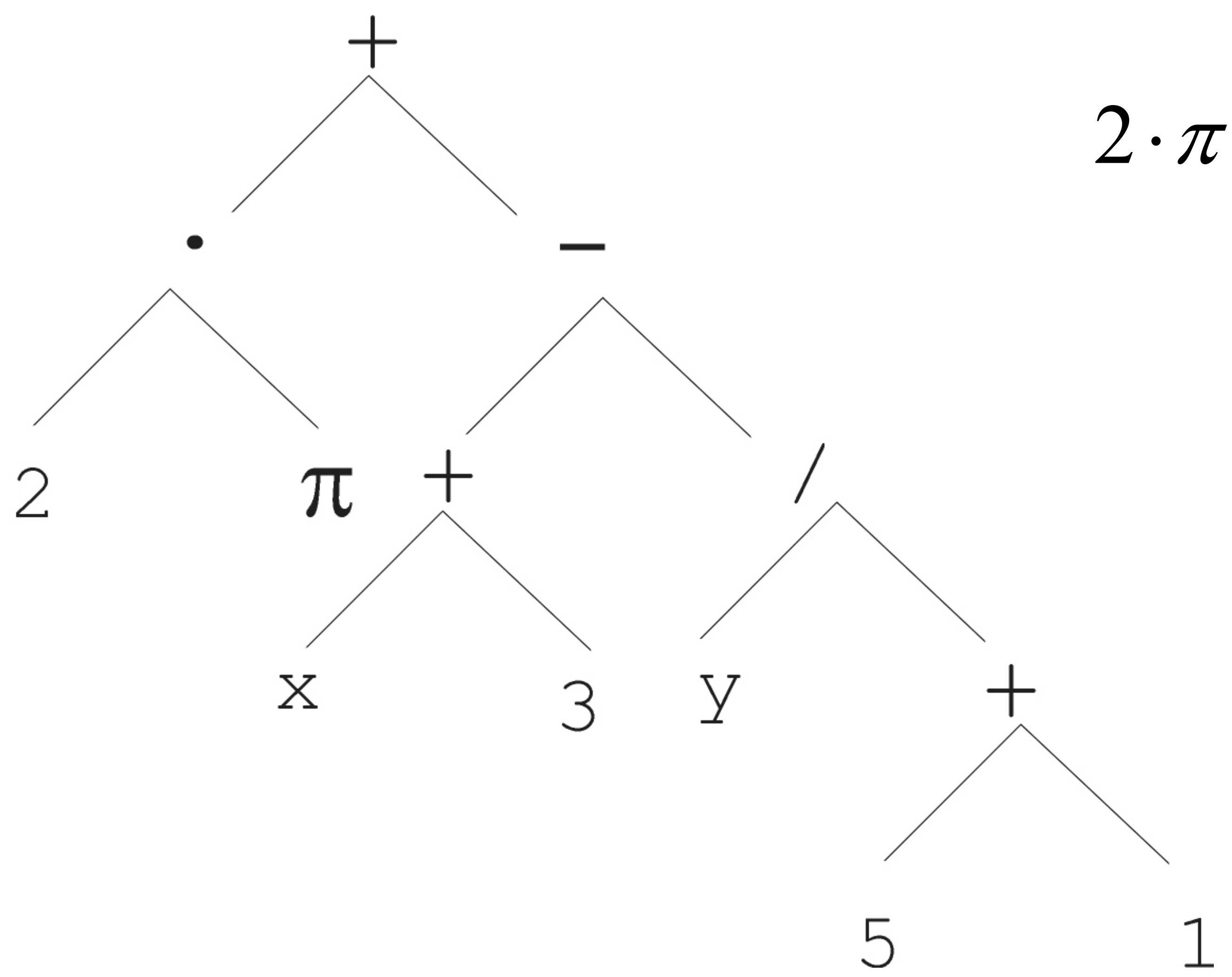
5	**4**	**3**	**8**	**7**	**6**	**1**	**2**

notice that many adjacencies between values have been preserved

Tree Representation

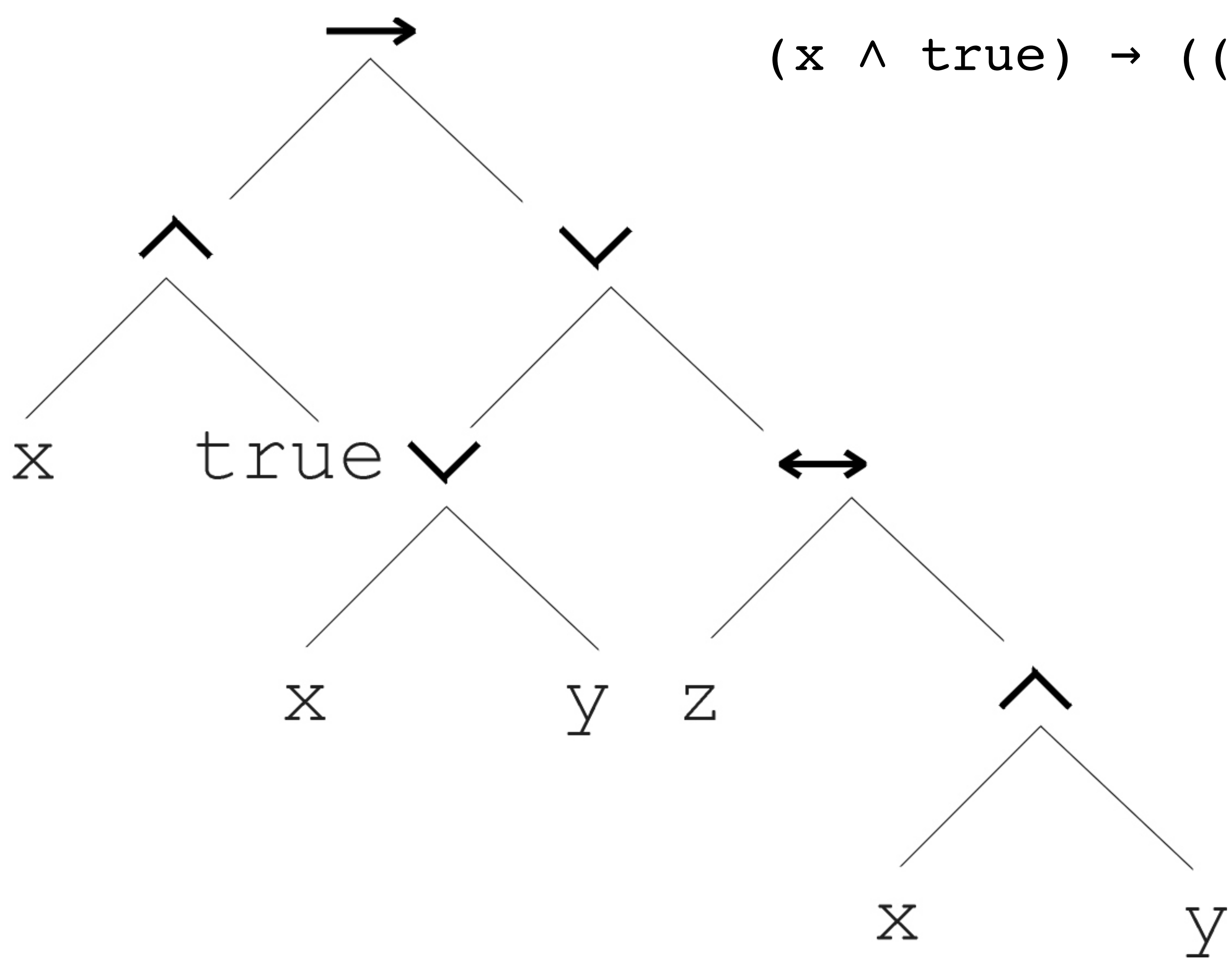
- trees are one of the most general structures for representing objects in computing
- they can be used to represent:
  - arithmetic formulae
  - logical formulae
  - parse trees
  - programs
  - and many other concepts
- they allow us to represent non-linear genotypes
- form the basis of genetic programming (GP)
  - where they can vary in depth and width

Tree Representation



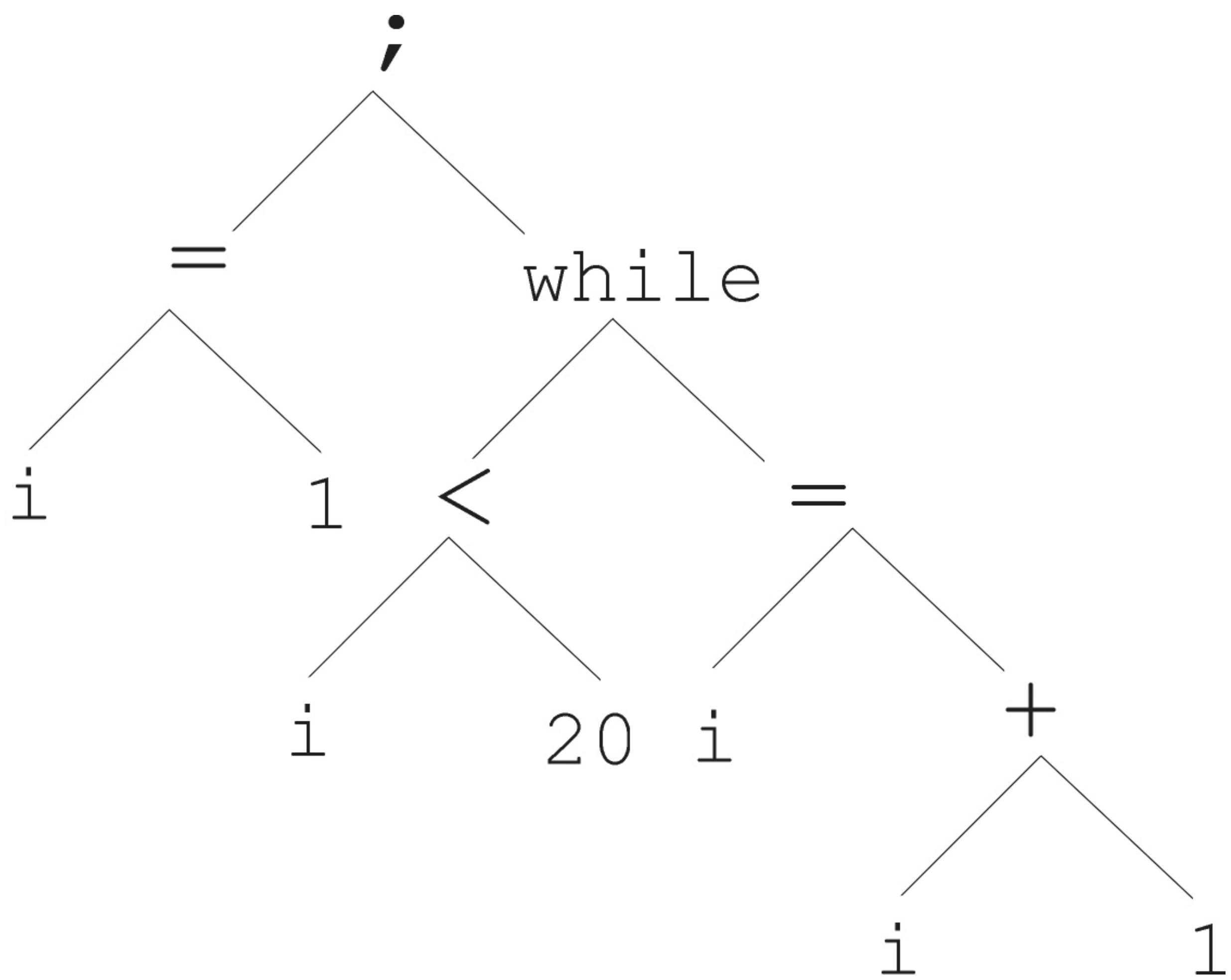
$$2 \cdot \pi + \left((x+3) - \frac{y}{5+1} \right)$$

Tree Representation



(x ∧ true) → ((x ∨ y) ∨ (z ↔ (x ∧ y)))

Tree Representation

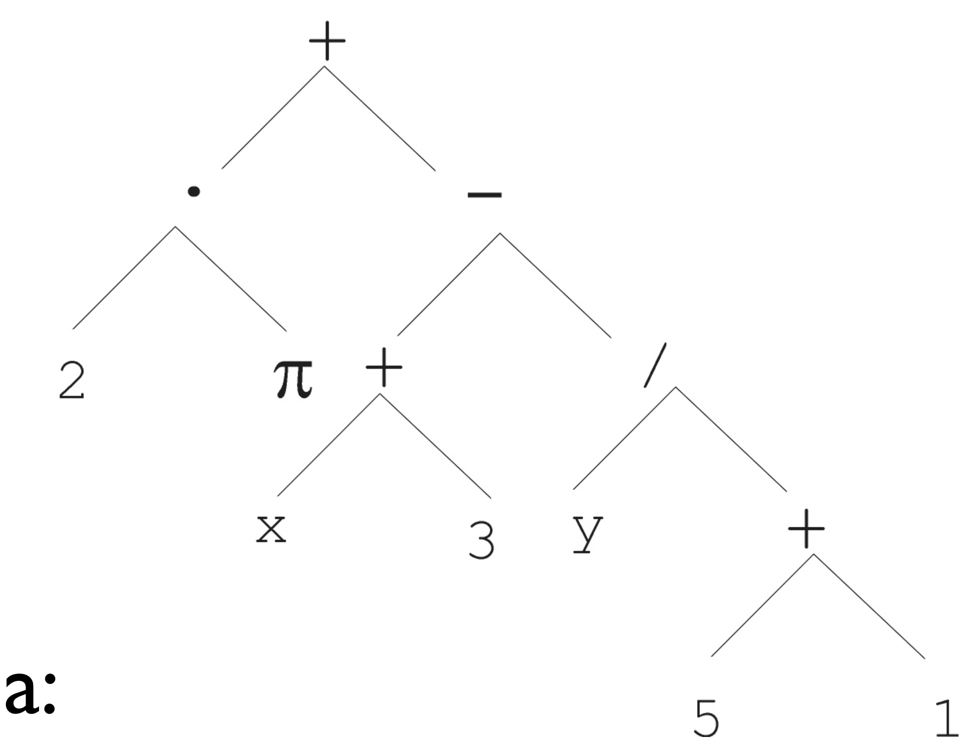



```
i =1;
while (i < 20)
{
   i = i + 1
}
```

Tree Representation

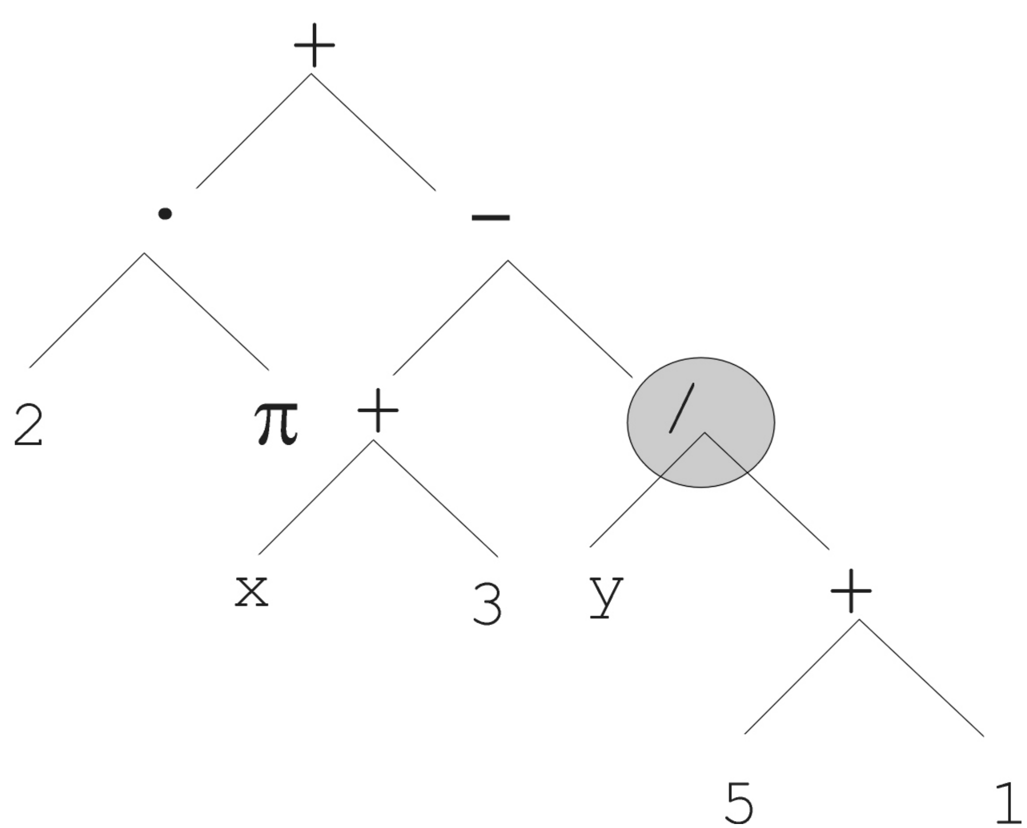
- symbolic expressions can be defined by
  - terminal set `T`
    - form the leaves
  - function set `F`
    - form the internal nodes
- for example, for an arithmetic formula:

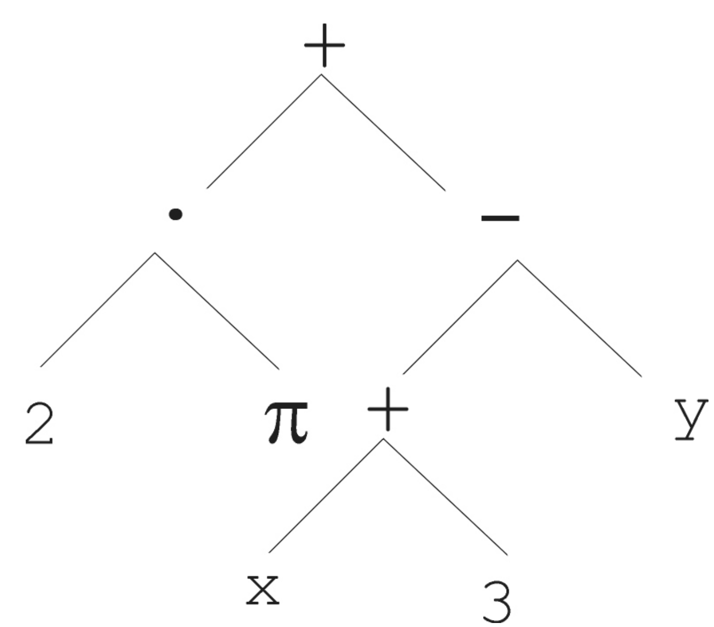
function set	{+, -, ., /}
terminal set	ℝ ∪ {x, y}



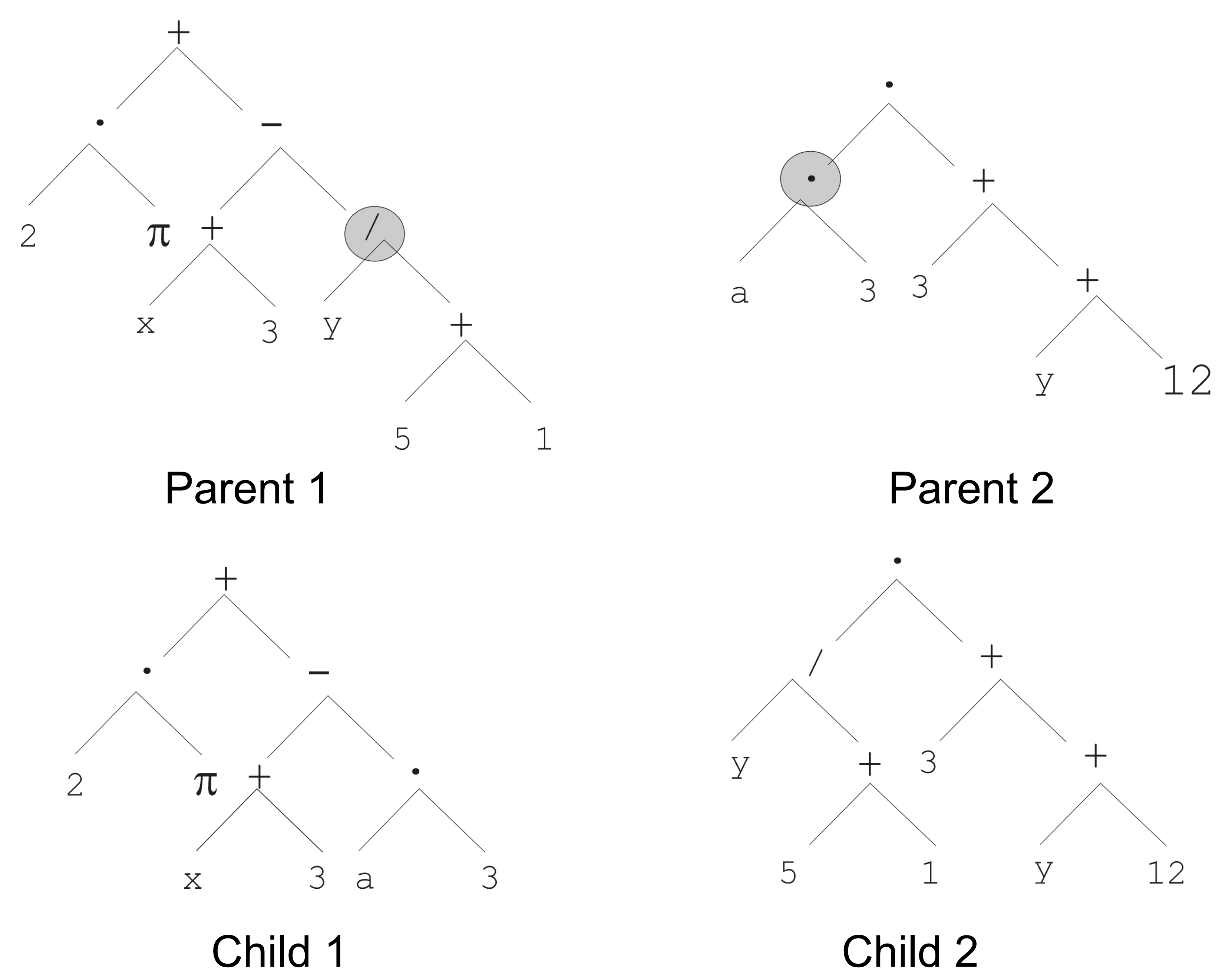
Tree Representation: Mutation

- *most common mutation operator:* replace a randomly chosen subtree with a randomly generated tree
- in this case, it's simply the variable 'y' from the terminal set
- but it could be larger





Tree Representation: Recombination



Reading & References

- slides based on and adapted from, Chapter 4 (and slides) of Eiben & Smith's *Introduction to Evolutionary Computing*
- W.M. Spears: Evolutionary Algorithms: The Role of Mutation and Recombination, Springer 2000
- K. Deb: Representations. Part 4 of T. Bäck, D. Fogel and Z. Michalewicz (editors) Evolutionary Computation 1: Basic Algorithms and Operators, Institute of Physics Press
  - *note that above link leads directly to a .pdf download*