BIENVENUE À OUIDAH !

NOUS SOMMES RAVIS DE VOUS ACCUEILLIR À **L'ACP SUMMER SCHOOL 2025**. CINQ JOURS D'ÉCHANGES PASSIONNANTS ET DE DÉCOUVERTES VOUS ATTENDENT DANS CE CADRE UNIQUE.

ENJOY YOUR STAY !

25-29 août 2025 / IRSP, Ouidah, Bénin

ACP SUMMER SCHOOL 2025

Fundamentals of Optimisation and Introduction to Constraint Programming

Hélène Verhaeghe

ACP

FRIARE

Irsia DIGITAL FOR DEVELOPEMENT

IFRI

Mens Molem Agitat UNIVERSITÉ D'ABOMEY-CALAVI

Artificial Intelligence

www.elsevier.com/locate/artint

A FIELD OF ARTIFICIAL INTELLIGENCE

Artificial Intelligence

Natural Language Processing

Machine Learning

Intelligent Robots

Constraint Satisfaction and Optimisation

Game Theory

Data Mining

Multiagent Systems

Knowledge Representation

Computer Vision

Artificial Intelligence

Natural Language Processing

Machine Learning

Intelligent Robots

Constraint Satisfaction and Optimisation

Game Theory

Data Mining

Multiagent Systems

Knowledge Representation

Computer Vision

CONSTRAINT SATISFACTION PROBLEMS (CSP)

A constraint satisfaction problem is defined as a triple $\langle X,D,C \rangle$, where

CONSTRAINT SATISFACTION PROBLEMS (CSP)

A constraint satisfaction problem is defined as a triple $\langle X,D,C \rangle$, where

- $X = \{X_1,\ldots,X_n\}$ is a set of variables

CONSTRAINT SATISFACTION PROBLEMS (CSP)

A constraint satisfaction problem is defined as a triple $\langle X,D,C \rangle$, where

- $X = \{X_1,\ldots,X_n\}$ is a set of variables
- $D = \{D_1,\ldots,D_n\}$ is a set of their respective domains of values

CONSTRAINT SATISFACTION PROBLEMS (CSP)

A constraint satisfaction problem is defined as a triple $\langle X,D,C \rangle$, where

- $X = \{X_1,\ldots,X_n\}$ is a set of variables
- $D = \{D_1,\ldots,D_n\}$ is a set of their respective domains of values
- $C = \{C_1,\ldots,C_n\}$ is a set of constraints

CONSTRAINT SATISFACTION PROBLEMS (CSP)

A constraint satisfaction problem is defined as a triple $\langle X,D,C \rangle$, where

- $X = \{X_1,\ldots,X_n\}$ is a set of variables
- $D = \{D_1,\ldots,D_n\}$ is a set of their respective domains of values
- $C = \{C_1,\ldots,C_n\}$ is a set of constraints

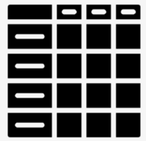
Exemples:

CONSTRAINT SATISFACTION PROBLEMS (CSP)

A constraint satisfaction problem is defined as a triple $\langle X,D,C \rangle$, where

- $X = \{X_1,\ldots,X_n\}$ is a set of variables
- $D = \{D_1,\ldots,D_n\}$ is a set of their respective domains of values
- $C = \{C_1,\ldots,C_n\}$ is a set of constraints

Exemples:



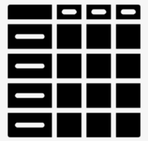
- Tournament scheduling problem: N sports team participate in a tournament. They all need to play against each other once. Can you schedule the match in N-1 days?

CONSTRAINT SATISFACTION PROBLEMS (CSP)

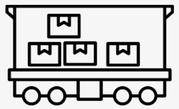
A constraint satisfaction problem is defined as a triple $\langle X,D,C \rangle$, where

- $X = \{X_1,\ldots,X_n\}$ is a set of variables
- $D = \{D_1,\ldots,D_n\}$ is a set of their respective domains of values
- $C = \{C_1,\ldots,C_n\}$ is a set of constraints

Exemples:



- Tournament scheduling problem: N sports team participate in a tournament. They all need to play against each other once. Can you schedule the match in N-1 days?



- Box packing: Given a rectangular container, and rectangular boxes. Can you fit the boxes in the container? If yes, how?

CONSTRAINT OPTIMISATION PROBLEMS (COP)

CONSTRAINT OPTIMISATION PROBLEMS (COP)

A constraint optimisation problem is a constraint satisfaction problem $\langle X,D,C \rangle$, with

CONSTRAINT OPTIMISATION PROBLEMS (COP)

A constraint optimisation problem is a constraint satisfaction problem $\langle X,D,C \rangle$, with

- An objectif function f: D-> $\mathbb{R}$

CONSTRAINT OPTIMISATION PROBLEMS (COP)

A constraint optimisation problem is a constraint satisfaction problem $\langle X,D,C \rangle$, with

- An objectif function f: D-> $\mathbb{R}$
- An optimal solution is a solution an objective of z^* such that, for every other solutions with objective value of z', z' >= z^* if we wish to minimise the objective, or z' <= z^* if we wish to maximise the objective

CONSTRAINT OPTIMISATION PROBLEMS (COP)

A constraint optimisation problem is a constraint satisfaction problem $\langle X,D,C \rangle$, with

- An objectif function f: D-> $\mathbb{R}$
- An optimal solution is a solution an objective of z^* such that, for every other solutions with objective value of z', z' >= z^* if we wish to minimise the objective, or z' <= z^* if we wish to maximise the objective

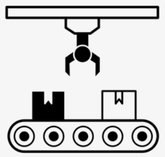
Exemple:

CONSTRAINT OPTIMISATION PROBLEMS (COP)

A constraint optimisation problem is a constraint satisfaction problem $\langle X,D,C \rangle$, with

- An objectif function f: D-> $\mathbb{R}$
- An optimal solution is a solution an objective of z^* such that, for every other solutions with objective value of z', z' >= z^* if we wish to minimise the objective, or z' <= z^* if we wish to maximise the objective

Exemple:



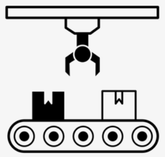
- Job-shop scheduling problem: Given N jobs to be executed on M machines, each job has a duration and a machine to be executed on, given a set of precedences between some pairs of jobs, can you find the minimum horizon schedule?

CONSTRAINT OPTIMISATION PROBLEMS (COP)

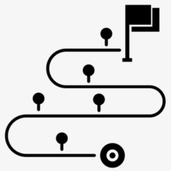
A constraint optimisation problem is a constraint satisfaction problem $\langle X,D,C \rangle$, with

- An objectif function f: D-> $\mathbb{R}$
- An optimal solution is a solution an objective of z^* such that, for every other solutions with objective value of z', z' >= z^* if we wish to minimise the objective, or z' <= z^* if we wish to maximise the objective

Exemple:



- Job-shop scheduling problem: Given N jobs to be executed on M machines, each job has a duration and a machine to be executed on, given a set of precedences between some pairs of jobs, can you find the minimum horizon schedule?



- Traveling Salesman Problem: Given N cities with given distance between pairs of cities, what is the smallest path to visit each cities and then come back to the starting point?

Artificial Intelligence

Constraint Satisfaction and Optimisation

Constraint Programming

Linear Programming

Meta-Heuristics

SMT

SAT Solving

Local Search

Artificial Intelligence

Constraint Satisfaction and Optimisation

Constraint Programming

Linear Programming

Meta-Heuristics

SMT

SAT Solving

Local Search

VARIOUS PARADIGMS FOR VARIOUS GOALS

	Complete exploration	Incomplete exploration
Methods	•Branch and bounds •**Constraint programming** •Integer programming •SAT solving •…	•Local search •Large Neighbourhood Search (LNS) •Genetic Algorithms •Meta-heuristics •…
Pros	Optimality proof	Aim to quickly find good solutions
Cons	Takes time to proof optimality	No Optimality proof

DISCRETE CSP AND COP

In CP, we target discrete CSP and COP, i.e. combinatorial (optimisation) problems

DISCRETE CSP AND COP

In CP, we target discrete CSP and COP, i.e. combinatorial (optimisation) problems

- Often difficult problems (NP-hard)

DISCRETE CSP AND COP

In CP, we target discrete CSP and COP, i.e. combinatorial (optimisation) problems

- Often difficult problems (NP-hard)
- Finite set of possible solutions (cardinal product of finite domains)

DISCRETE CSP AND COP

In CP, we target discrete CSP and COP, i.e. combinatorial (optimisation) problems

- Often difficult problems (NP-hard)
- Finite set of possible solutions (cardinal product of finite domains)
- Checking all the possible solutions one by one is not tractable

DISCRETE CSP AND COP

In CP, we target discrete CSP and COP, i.e. combinatorial (optimisation) problems

- Often difficult problems (NP-hard)
- Finite set of possible solutions (cardinal product of finite domains)
- Checking all the possible solutions one by one is not tractable
- Require intelligent exploration of the solution space to find good solutions

HOLY GRAIL OF PROGRAMMING

"Constraint programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it" - E. Freuder (1996)

DECLARATIVE PROGRAMMING

Declarative programming:

- The user describes what the program must accomplish, rather than describing how to accomplish it

Imperative programming:

- The user describes how a program operates step by step

DECLARATIVE PROGRAMMING

Declarative programming:

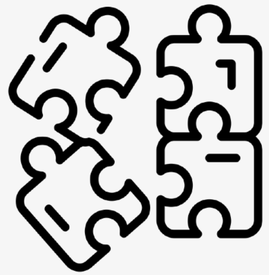
- The user describes what the program must accomplish, rather than describing how to accomplish it

Imperative programming:

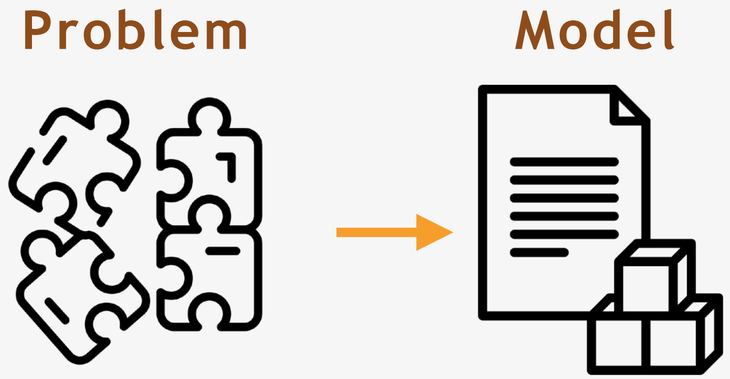
- The user describes how a program operates step by step

A CP Model Is a Declarative Description of the Solution

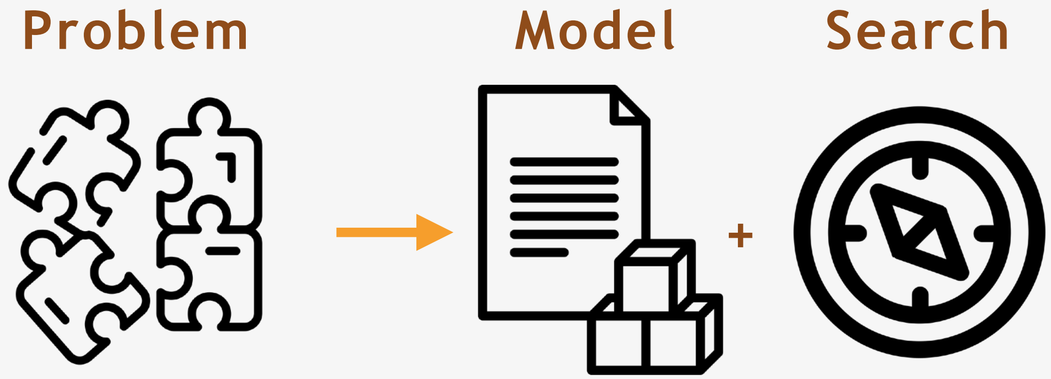
Problem



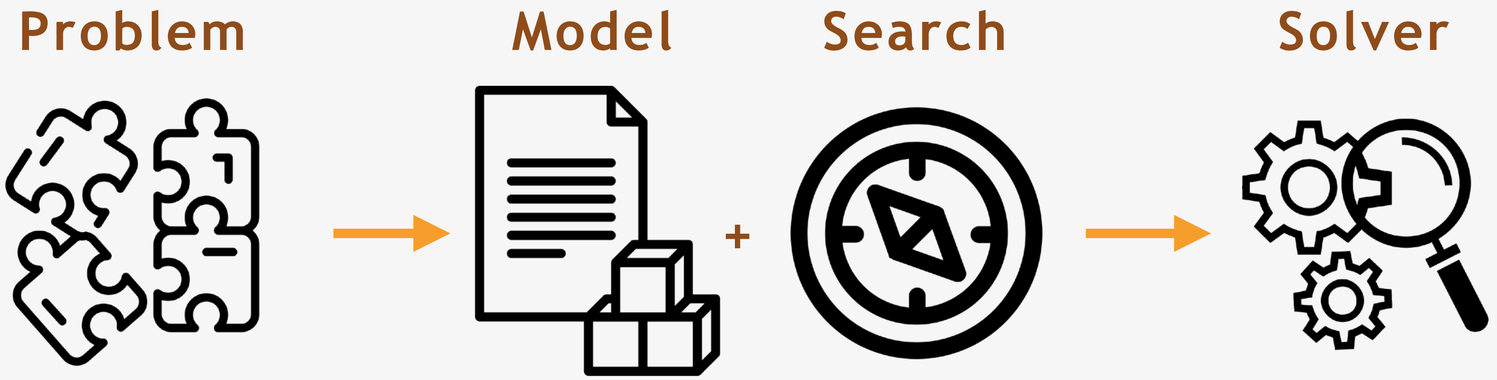
THE CONSTRAINT PROGRAMMING PARADIGM



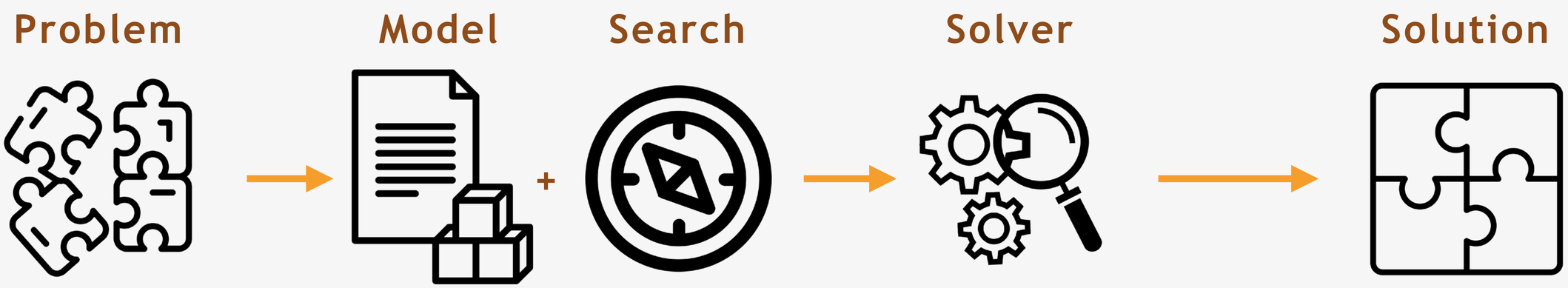
THE CONSTRAINT PROGRAMMING PARADIGM

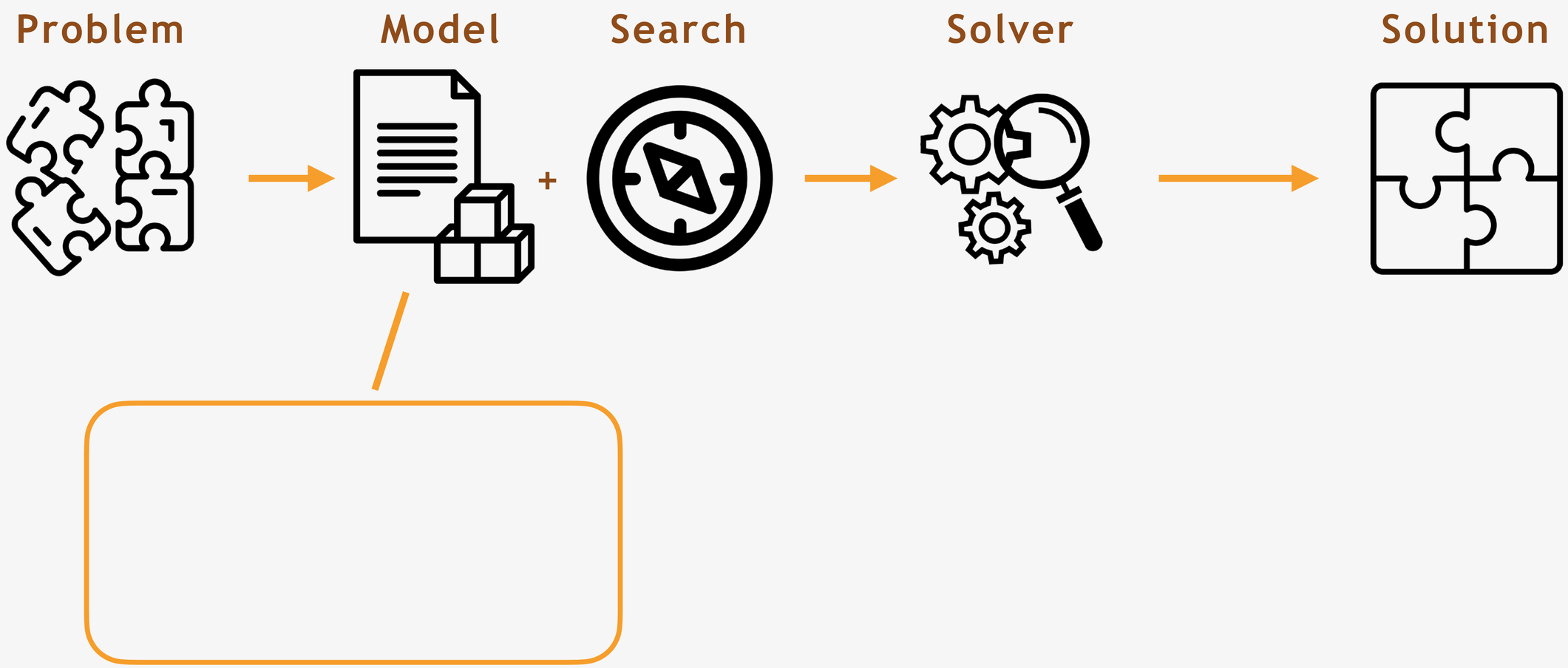


THE CONSTRAINT PROGRAMMING PARADIGM

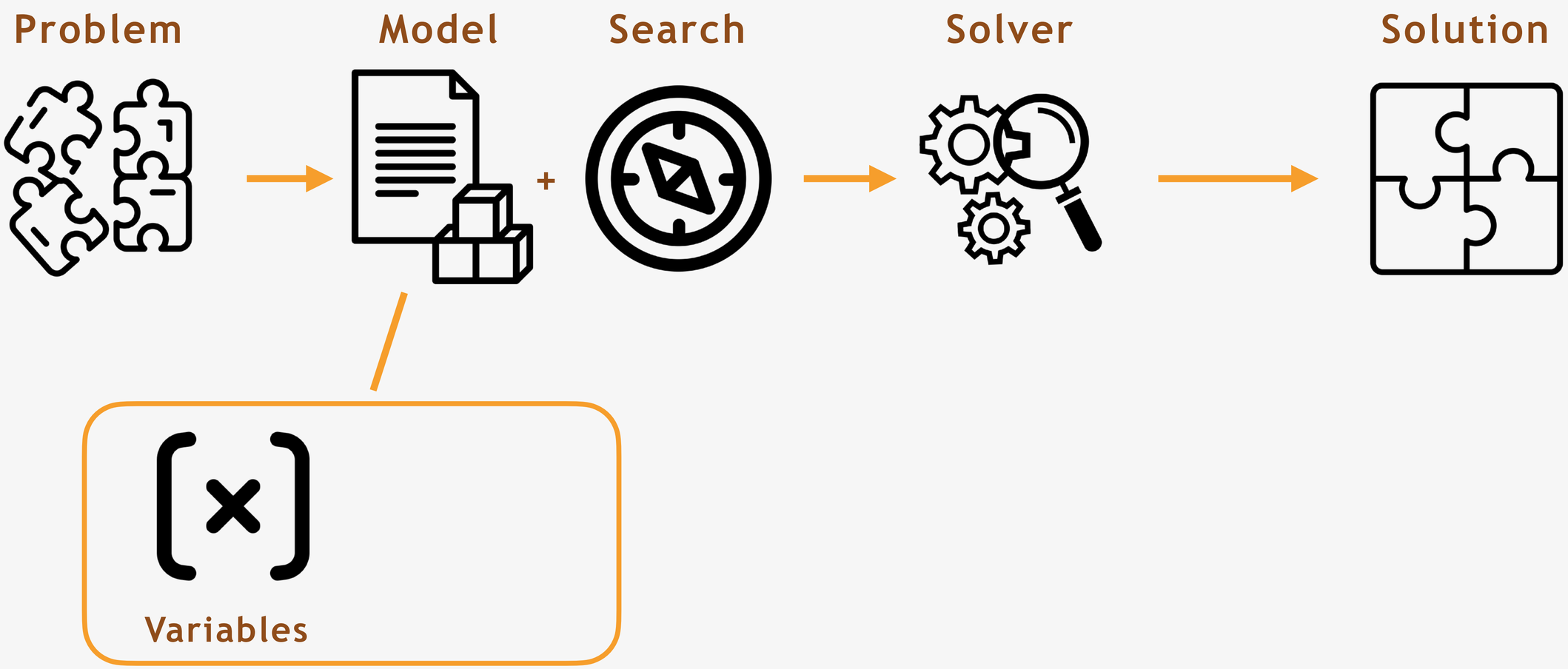


THE CONSTRAINT PROGRAMMING PARADIGM

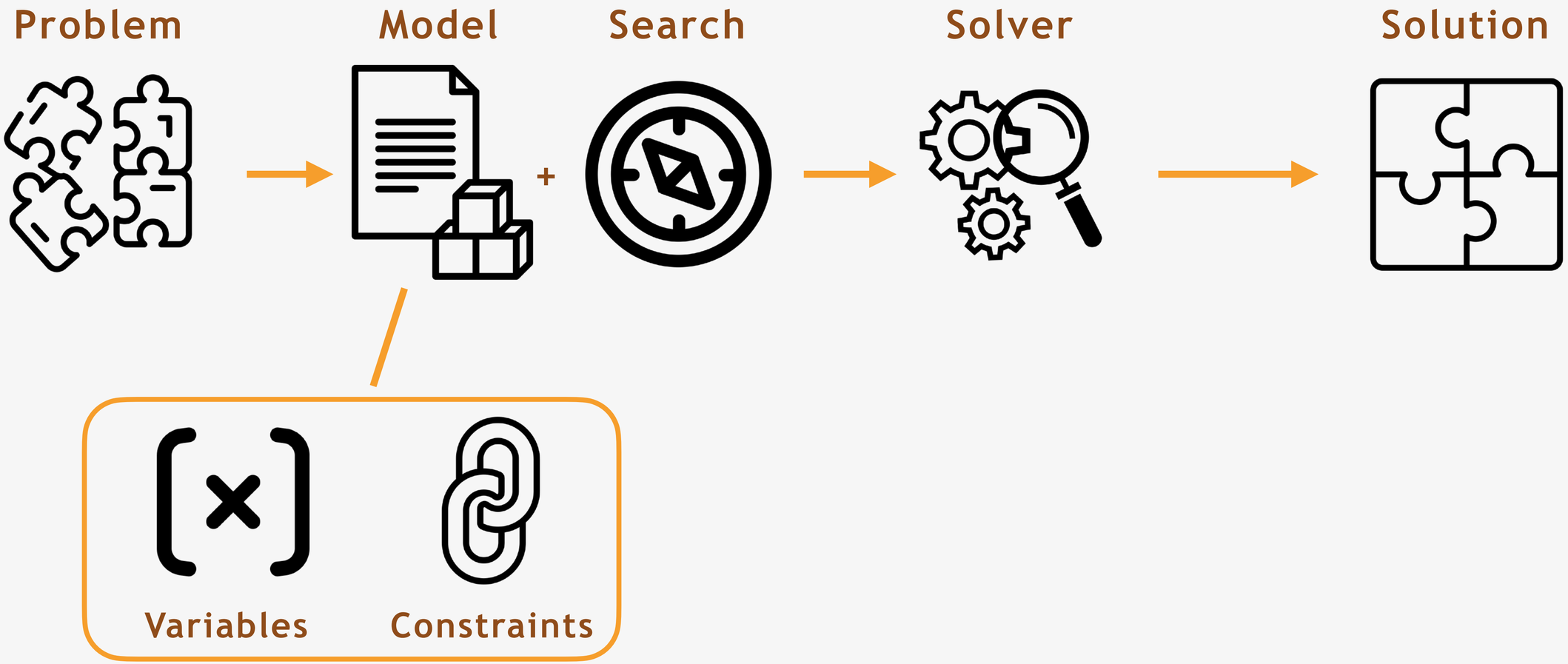



Problem
Model
Search
Solver
Solution
+

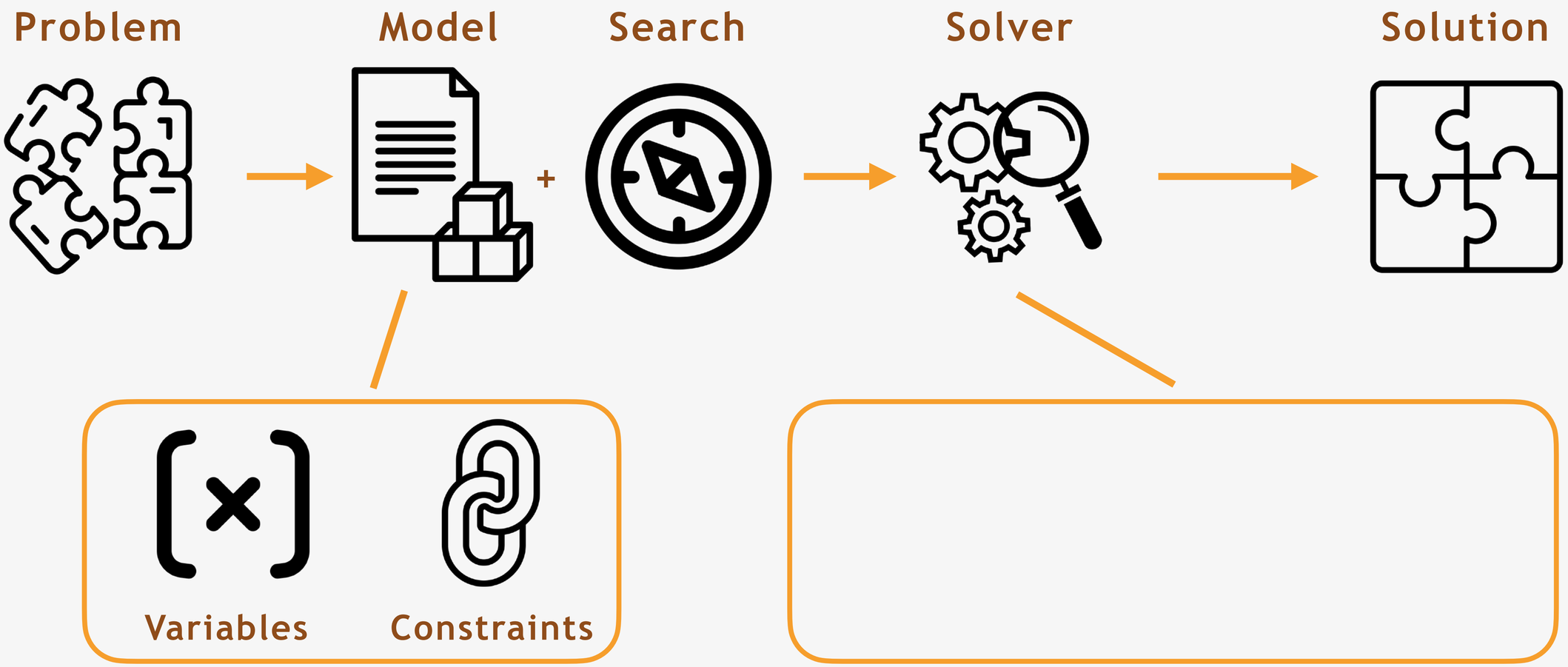
THE CONSTRAINT PROGRAMMING PARADIGM



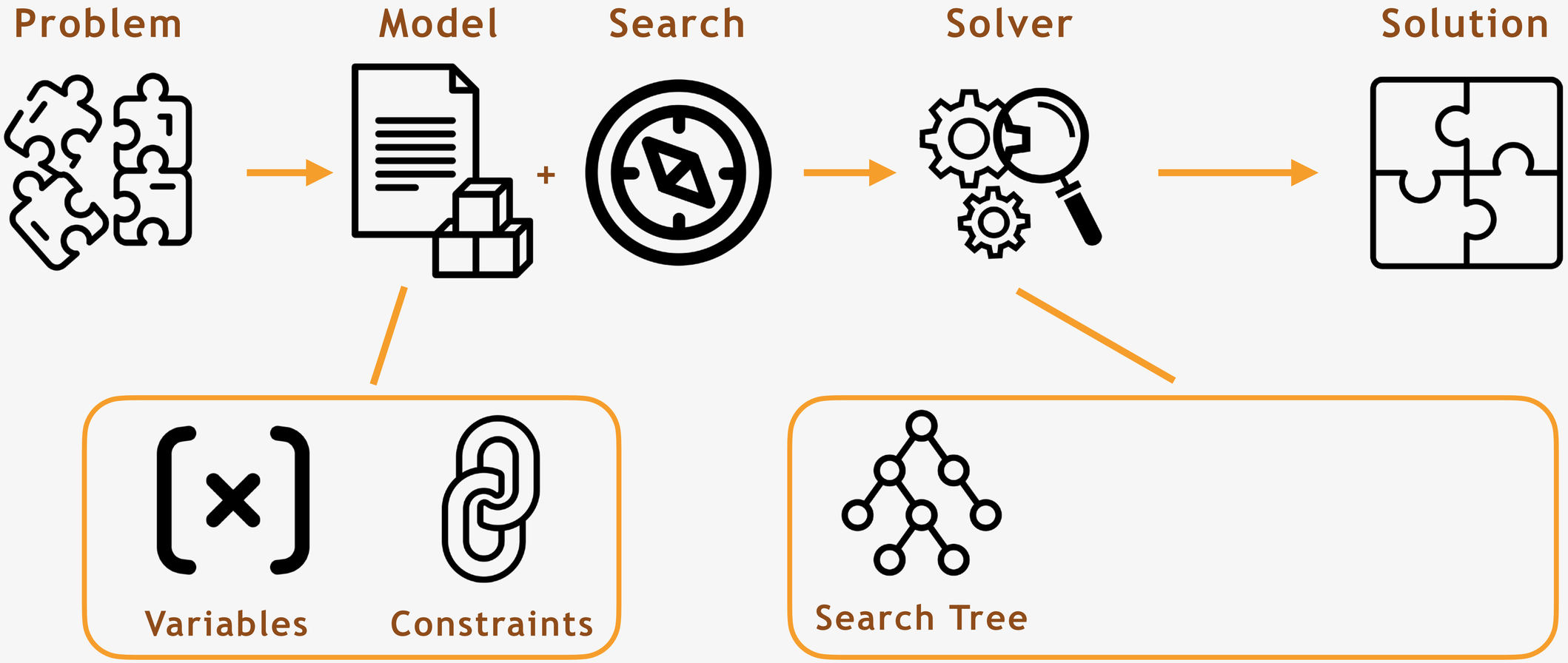
THE CONSTRAINT PROGRAMMING PARADIGM



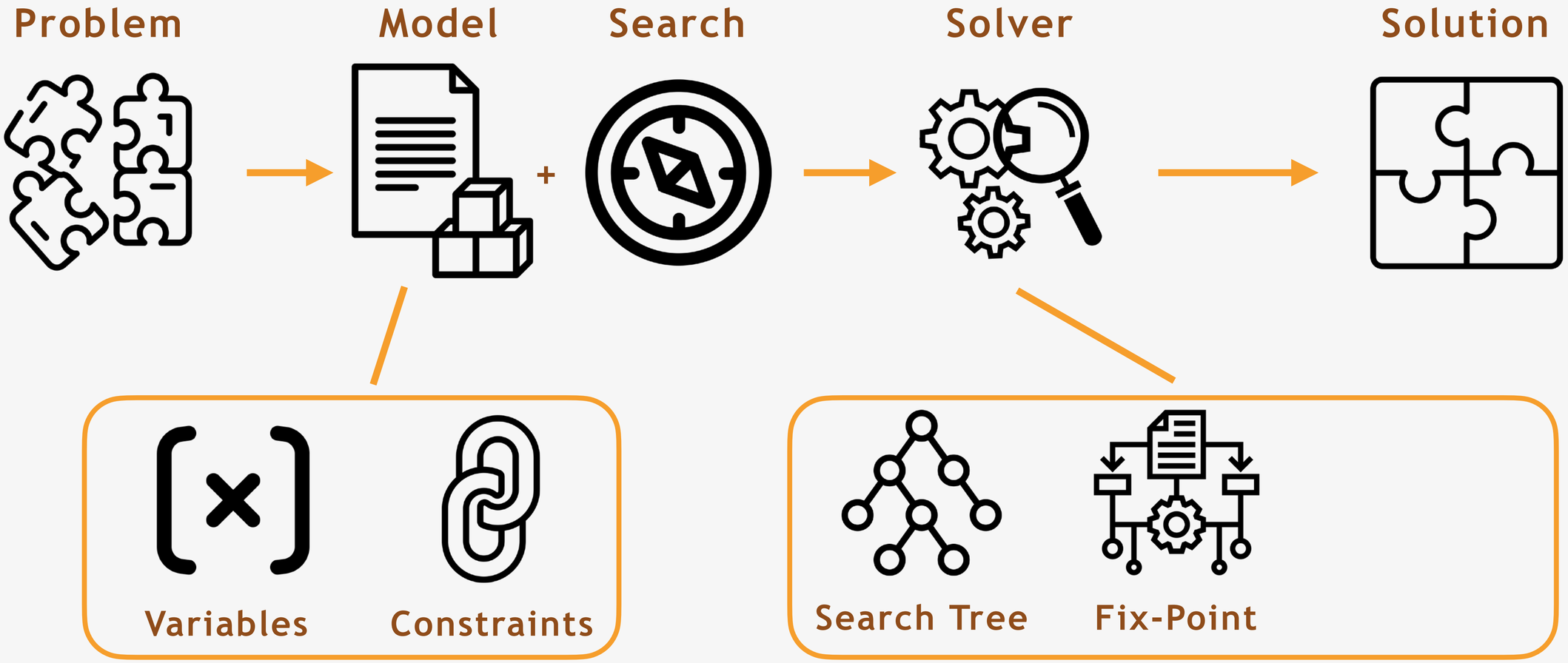
THE CONSTRAINT PROGRAMMING PARADIGM



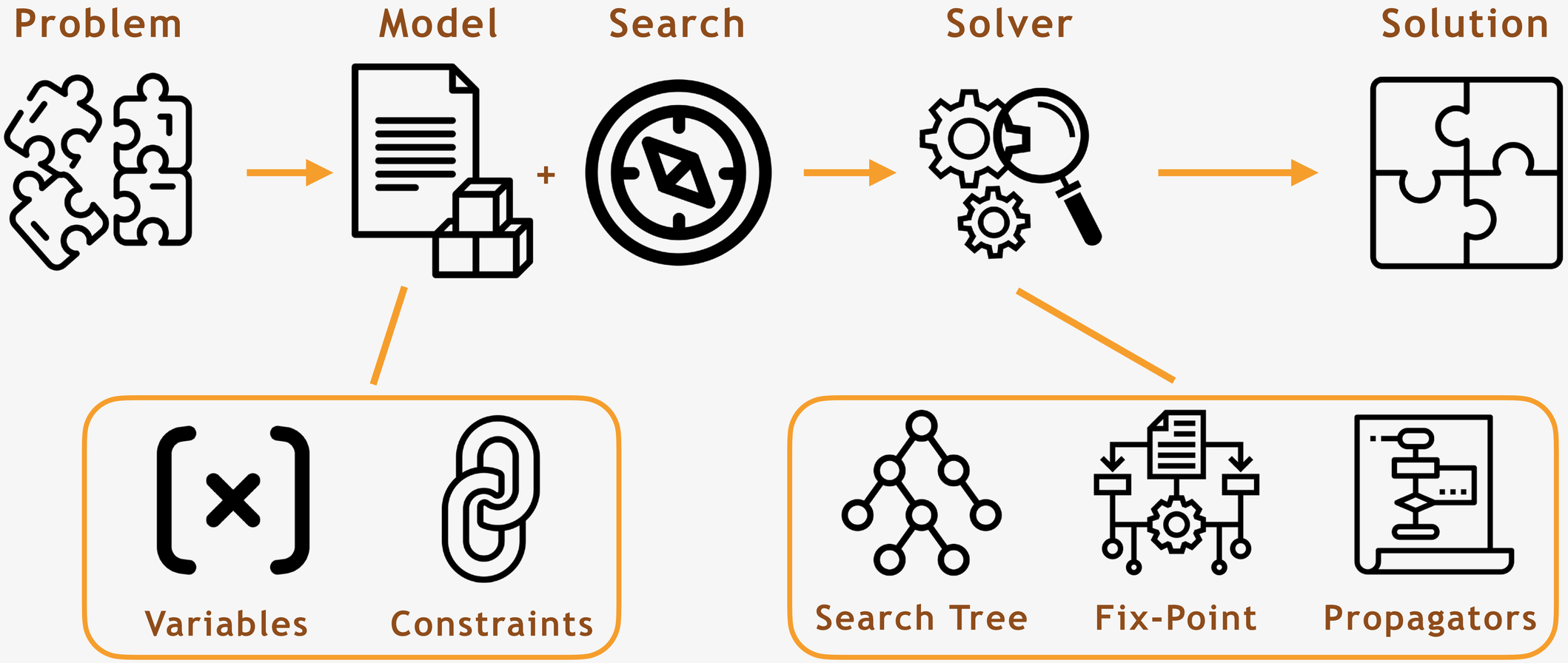
THE CONSTRAINT PROGRAMMING PARADIGM



THE CONSTRAINT PROGRAMMING PARADIGM



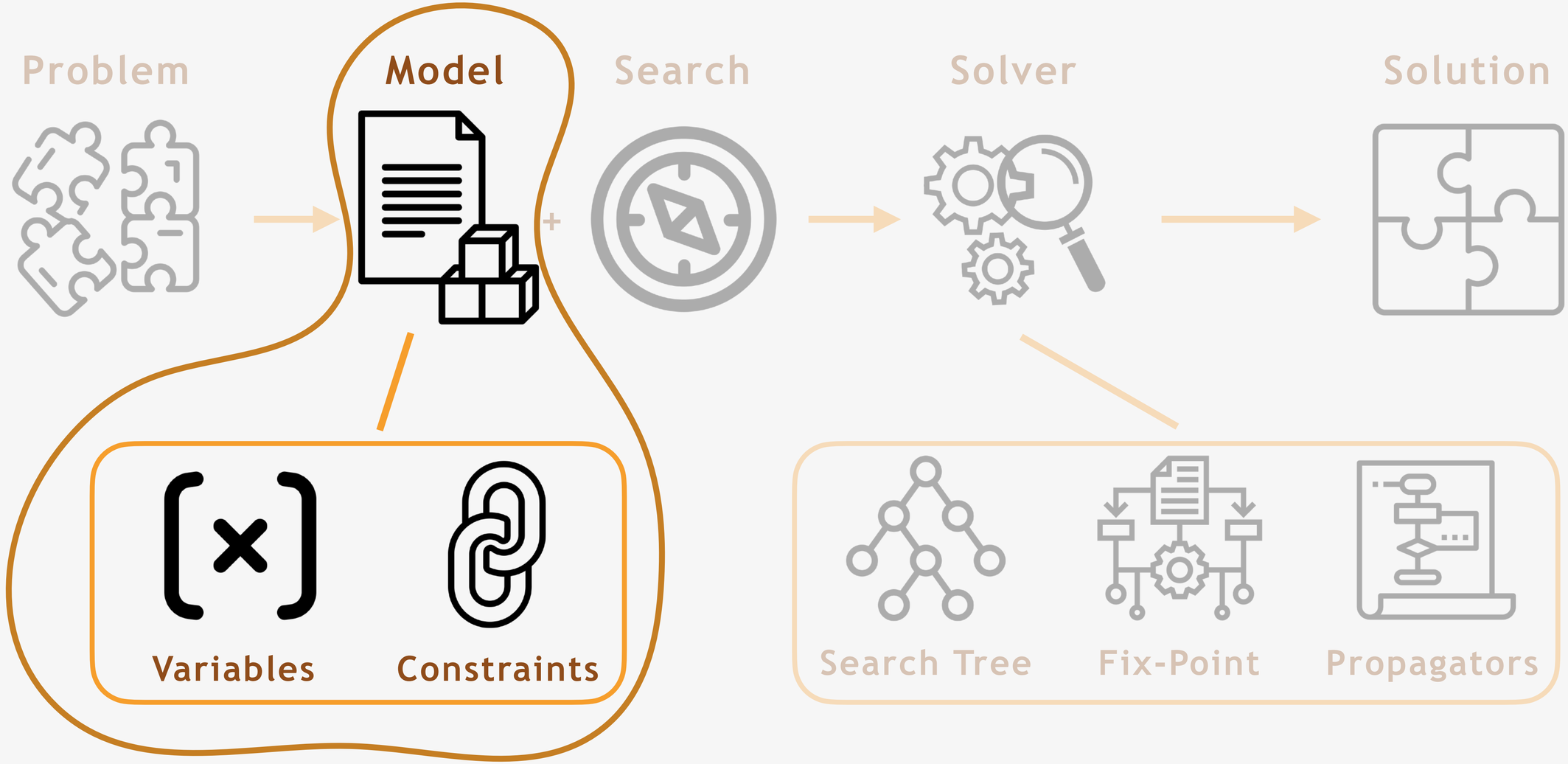
THE CONSTRAINT PROGRAMMING PARADIGM



CP = Model + Search

Model

THE CONSTRAINT PROGRAMMING PARADIGM



A FIRST EXAMPLE: THE SUDOKU

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

A FIRST EXAMPLE: THE SUDOKU

Let us do the exercise and describe what a solution should be!

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

A FIRST EXAMPLE: THE SUDOKU

Let us do the exercise and describe what a solution should be!

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

A FIRST EXAMPLE: THE SUDOKU

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Let us do the exercise and describe what a solution should be!

- The solution is organised as a grid of 9x9 cells containing a sub division of 3x3 3x3 sub-grid

A FIRST EXAMPLE: THE SUDOKU

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Let us do the exercise and describe what a solution should be!

- The solution is organised as a grid of 9x9 cells containing a sub division of 3x3 3x3 sub-grid
- Each of these cell is filled with an integer from 1 to 9

A FIRST EXAMPLE: THE SUDOKU

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Let us do the exercise and describe what a solution should be!

- The solution is organised as a grid of 9x9 cells containing a sub division of 3x3 3x3 sub-grid
- Each of these cell is filled with an integer from 1 to 9
- Cells that had a filled number in the problem statement are filled with that exact number

A FIRST EXAMPLE: THE SUDOKU

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Let us do the exercise and describe what a solution should be!

- The solution is organised as a grid of 9x9 cells containing a sub division of 3x3 3x3 sub-grid
- Each of these cell is filled with an integer from 1 to 9
- Cells that had a filled number in the problem statement are filled with that exact number
- On each line of the grid, each of the numbers from 1 to 9 are present only once

A FIRST EXAMPLE: THE SUDOKU

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Let us do the exercise and describe what a solution should be!

- The solution is organised as a grid of 9x9 cells containing a sub division of 3x3 3x3 sub-grid
- Each of these cell is filled with an integer from 1 to 9
- Cells that had a filled number in the problem statement are filled with that exact number
- On each line of the grid, each of the numbers from 1 to 9 are present only once
- On each column of the grid, each of the numbers from 1 to 9 are present only once

A FIRST EXAMPLE: THE SUDOKU

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Let us do the exercise and describe what a solution should be!

- The solution is organised as a grid of 9x9 cells containing a sub division of 3x3 3x3 sub-grid
- Each of these cell is filled with an integer from 1 to 9
- Cells that had a filled number in the problem statement are filled with that exact number
- On each line of the grid, each of the numbers from 1 to 9 are present only once
- On each column of the grid, each of the numbers from 1 to 9 are present only once
- On each sub-grid, each of the numbers from 1 to 9 are present only once

A FIRST EXAMPLE: THE SUDOKU

Let us do the exercise and describe what a solution should be!

- The solution is organised as a grid of 9x9 cells containing a sub division of 3x3 3x3 sub-grid
- Each of these cell is filled with an integer from 1 to 9
- Cells that had a filled number in the problem statement are filled with that exact number
- On each line of the grid, each of the numbers from 1 to 9 are present only once
- On each column of the grid, each of the numbers from 1 to 9 are present only once
- On each sub-grid, each of the numbers from 1 to 9 are present only once

A FIRST EXAMPLE: THE SUDOKU

Let us do the exercise and describe what a solution should be!

- The solution is organised as a grid of 9x9 cells containing a sub division of 3x3 3x3 sub-grid
- Each of these cell is filled with an integer from 1 to 9
- Cells that had a filled number in the problem statement are filled with that exact number
- On each line of the grid, each of the numbers from 1 to 9 are present only once
- On each column of the grid, each of the numbers from 1 to 9 are present only once
- On each sub-grid, each of the numbers from 1 to 9 are present only once

Translation to more formal modelling

A FIRST EXAMPLE: THE SUDOKU

Let us do the exercise and describe what a solution should be!

- The solution is organised as a grid of 9x9 cells containing a sub division of 3x3 3x3 sub-grid
- Each of these cell is filled with an integer from 1 to 9
- Cells that had a filled number in the problem statement are filled with that exact number
- On each line of the grid, each of the numbers from 1 to 9 are present only once
- On each column of the grid, each of the numbers from 1 to 9 are present only once
- On each sub-grid, each of the numbers from 1 to 9 are present only once

Translation to more formal modelling

A FIRST EXAMPLE: THE SUDOKU

Let us do the exercise and describe what a solution should be!

- The solution is organised as a grid of 9x9 cells containing a sub division of 3x3 3x3 sub-grid
- Each of these cell is filled with an integer from 1 to 9
- Cells that had a filled number in the problem statement are filled with that exact number
- On each line of the grid, each of the numbers from 1 to 9 are present only once
- On each column of the grid, each of the numbers from 1 to 9 are present only once
- On each sub-grid, each of the numbers from 1 to 9 are present only once

Translation to more formal modelling

- Variables grid[i][j] for i,j $\in$ {0..8}

A FIRST EXAMPLE: THE SUDOKU

Let us do the exercise and describe what a solution should be!

- The solution is organised as a grid of 9x9 cells containing a sub division of 3x3 3x3 sub-grid
- Each of these cell is filled with an integer from 1 to 9
- Cells that had a filled number in the problem statement are filled with that exact number
- On each line of the grid, each of the numbers from 1 to 9 are present only once
- On each column of the grid, each of the numbers from 1 to 9 are present only once
- On each sub-grid, each of the numbers from 1 to 9 are present only once

Translation to more formal modelling

- Variables grid[i][j] for i,j ∈ {0..8}
- subgrid[i][j] = { grid[i*3+m][j*3+n] ı m,n ∈ {0..2} } for i,j ∈ {0..2}

A FIRST EXAMPLE: THE SUDOKU

Let us do the exercise and describe what a solution should be!

- The solution is organised as a grid of 9x9 cells containing a sub division of 3x3 3x3 sub-grid
- Each of these cell is filled with an integer from 1 to 9
- Cells that had a filled number in the problem statement are filled with that exact number
- On each line of the grid, each of the numbers from 1 to 9 are present only once
- On each column of the grid, each of the numbers from 1 to 9 are present only once
- On each sub-grid, each of the numbers from 1 to 9 are present only once

Translation to more formal modelling

- Variables grid[i][j] for i,j $\in$ {0..8}
- subgrid[i][j] = { grid[i*3+m][j*3+n] ı m,n $\in$ {0..2} } for i,j $\in$ {0..2}
- dom(grid[i][j]) = {1..9} for i,j $\in$ {0..8}

A FIRST EXAMPLE: THE SUDOKU

Let us do the exercise and describe what a solution should be!

- The solution is organised as a grid of 9x9 cells containing a sub division of 3x3 3x3 sub-grid
- Each of these cell is filled with an integer from 1 to 9
- Cells that had a filled number in the problem statement are filled with that exact number
- On each line of the grid, each of the numbers from 1 to 9 are present only once
- On each column of the grid, each of the numbers from 1 to 9 are present only once
- On each sub-grid, each of the numbers from 1 to 9 are present only once

Translation to more formal modelling

- Variables grid[i][j] for i,j ∈ {0..8}
- subgrid[i][j] = { grid[i*3+m][j*3+n] ı m,n ∈ {0..2} } for i,j ∈ {0..2}
- dom(grid[i][j]) = {1..9} for i,j ∈ {0..8}
- grid[i][j]=v_{ij} for each filled square

A FIRST EXAMPLE: THE SUDOKU

Let us do the exercise and describe what a solution should be!

- The solution is organised as a grid of 9x9 cells containing a sub division of 3x3 3x3 sub-grid
- Each of these cell is filled with an integer from 1 to 9
- Cells that had a filled number in the problem statement are filled with that exact number
- On each line of the grid, each of the numbers from 1 to 9 are present only once
- On each column of the grid, each of the numbers from 1 to 9 are present only once
- On each sub-grid, each of the numbers from 1 to 9 are present only once

Translation to more formal modelling

- Variables grid[i][j] for i,j ∈ {0..8}
- subgrid[i][j] = { grid[i*3+m][j*3+n] ı m,n ∈ {0..2} } for i,j ∈ {0..2}
- dom(grid[i][j]) = {1..9} for i,j ∈ {0..8}
- grid[i][j]=v_{ij} for each filled square
- allDifferent({grid[k][j] ı k ∈ {0..8}}) for j ∈ {0..8}

A FIRST EXAMPLE: THE SUDOKU

Let us do the exercise and describe what a solution should be!

- The solution is organised as a grid of 9x9 cells containing a sub division of 3x3 3x3 sub-grid
- Each of these cell is filled with an integer from 1 to 9
- Cells that had a filled number in the problem statement are filled with that exact number
- On each line of the grid, each of the numbers from 1 to 9 are present only once
- On each column of the grid, each of the numbers from 1 to 9 are present only once
- On each sub-grid, each of the numbers from 1 to 9 are present only once

Translation to more formal modelling

- Variables grid[i][j] for i,j ∈ {0..8}
- subgrid[i][j] = { grid[i*3+m][j*3+n] ı m,n ∈ {0..2} } for i,j ∈ {0..2}
- dom(grid[i][j]) = {1..9} for i,j ∈ {0..8}
- grid[i][j]=v_{ij} for each filled square
- allDifferent({grid[k][j] ı k ∈ {0..8}}) for j ∈ {0..8}
- allDifferent({grid[i][k] ı k ∈ {0..8}}) for i ∈ {0..8}

A FIRST EXAMPLE: THE SUDOKU

Let us do the exercise and describe what a solution should be!

- The solution is organised as a grid of 9x9 cells containing a sub division of 3x3 3x3 sub-grid
- Each of these cell is filled with an integer from 1 to 9
- Cells that had a filled number in the problem statement are filled with that exact number
- On each line of the grid, each of the numbers from 1 to 9 are present only once
- On each column of the grid, each of the numbers from 1 to 9 are present only once
- On each sub-grid, each of the numbers from 1 to 9 are present only once

Translation to more formal modelling

- Variables grid[i][j] for i,j ∈ {0..8}
- subgrid[i][j] = { grid[i*3+m][j*3+n] ı m,n ∈ {0..2} } for i,j ∈ {0..2}
- dom(grid[i][j]) = {1..9} for i,j ∈ {0..8}
- grid[i][j]=v_{ij} for each filled square
- allDifferent({grid[k][j] ı k ∈ {0..8}}) for j ∈ {0..8}
- allDifferent({grid[i][k] ı k ∈ {0..8}}) for i ∈ {0..8}
- allDifferent(subgrid[i][j]) for i,j ∈ {0..2}

A FIRST EXAMPLE: THE SUDOKU

Let us do the exercise and describe what a solution should be!

- The solution is organised as a grid of 9x9 cells containing a sub division of 3x3 3x3 sub-grid
- Each of these cell is filled with an integer from 1 to 9
- Cells that had a filled number in the problem statement are filled with that exact number
- On each line of the grid, each of the numbers from 1 to 9 are present only once
- On each column of the grid, each of the numbers from 1 to 9 are present only once
- On each sub-grid, each of the numbers from 1 to 9 are present only once

Translation to more formal modelling

Variables

- Variables grid[i][j] for i,j ∈ {0..8}
- subgrid[i][j] = { grid[i*3+m][j*3+n] ı m,n ∈ {0..2} } for i,j ∈ {0..2}
- dom(grid[i][j]) = {1..9} for i,j ∈ {0..8}
- grid[i][j]=v_{ij} for each filled square
- allDifferent({grid[k][j] ı k ∈ {0..8}}) for j ∈ {0..8}
- allDifferent({grid[i][k] ı k ∈ {0..8}}) for i ∈ {0..8}
- allDifferent(subgrid[i][j]) for i,j ∈ {0..2}

A FIRST EXAMPLE: THE SUDOKU

Let us do the exercise and describe what a solution should be!

- The solution is organised as a grid of 9x9 cells containing a sub division of 3x3 3x3 sub-grid
- Each of these cell is filled with an integer from 1 to 9
- Cells that had a filled number in the problem statement are filled with that exact number
- On each line of the grid, each of the numbers from 1 to 9 are present only once
- On each column of the grid, each of the numbers from 1 to 9 are present only once
- On each sub-grid, each of the numbers from 1 to 9 are present only once

Translation to more formal modelling

Variables

- Variables grid[i][j] for i,j ∈ {0..8}
- subgrid[i][j] = { grid[i*3+m][j*3+n] ı m,n ∈ {0..2} } for i,j ∈ {0..2}
- dom(grid[i][j]) = {1..9} for i,j ∈ {0..8}

Domain of the Variable

- grid[i][j]=v_{ij} for each filled square
- allDifferent({grid[k][j] ı k ∈ {0..8}}) for j ∈ {0..8}
- allDifferent({grid[i][k] ı k ∈ {0..8}}) for i ∈ {0..8}
- allDifferent(subgrid[i][j]) for i,j ∈ {0..2}

A FIRST EXAMPLE: THE SUDOKU

Let us do the exercise and describe what a solution should be!

- The solution is organised as a grid of 9x9 cells containing a sub division of 3x3 3x3 sub-grid
- Each of these cell is filled with an integer from 1 to 9
- Cells that had a filled number in the problem statement are filled with that exact number
- On each line of the grid, each of the numbers from 1 to 9 are present only once
- On each column of the grid, each of the numbers from 1 to 9 are present only once
- On each sub-grid, each of the numbers from 1 to 9 are present only once

Translation to more formal modelling

Variables

- Variables grid[i][j] for i,j ∈ {0..8}
- subgrid[i][j] = { grid[i*3+m][j*3+n] ı m,n ∈ {0..2} } for i,j ∈ {0..2}

Domain of the Variable

- dom(grid[i][j]) = {1..9} for i,j ∈ {0..8}

Constraint

- grid[i][j]=v_{ij} for each filled square
- allDifferent({grid[k][j] ı k ∈ {0..8}}) for j ∈ {0..8}
- allDifferent({grid[i][k] ı k ∈ {0..8}}) for i ∈ {0..8}
- allDifferent(subgrid[i][j]) for i,j ∈ {0..2}

A FIRST EXAMPLE: THE SUDOKU

Let us do the exercise and describe what a solution should be!

- The solution is organised as a grid of 9x9 cells containing a sub division of 3x3 3x3 sub-grid
- Each of these cell is filled with an integer from 1 to 9
- Cells that had a filled number in the problem statement are filled with that exact number
- On each line of the grid, each of the numbers from 1 to 9 are present only once
- On each column of the grid, each of the numbers from 1 to 9 are present only once
- On each sub-grid, each of the numbers from 1 to 9 are present only once

Translation to more formal modelling

Variables

- Variables grid[i][j] for i,j ∈ {0..8}
- subgrid[i][j] = { grid[i*3+m][j*3+n] ı m,n ∈ {0..2} } for i,j ∈ {0..2}

Domain of the Variable

- dom(grid[i][j]) = {1..9} for i,j ∈ {0..8}

Constraint

- grid[i][j]=v_{ij} for each filled square

Global Constraint

- allDifferent({grid[k][j] ı k ∈ {0..8}}) for j ∈ {0..8}
- allDifferent({grid[i][k] ı k ∈ {0..8}}) for i ∈ {0..8}
- allDifferent(subgrid[i][j]) for i,j ∈ {0..2}

VARIABLES AND THEIR DOMAINS

VARIABLES AND THEIR DOMAINS

What are variables in CP?

VARIABLES AND THEIR DOMAINS

What are variables in CP?

- Variables are the unknown of the problem

VARIABLES AND THEIR DOMAINS

What are variables in CP?

- Variables are the unknown of the problem
- They have a selection of allowed values they can be assigned to (Domain of the variable)

VARIABLES AND THEIR DOMAINS

What are variables in CP?

- Variables are the unknown of the problem
- They have a selection of allowed values they can be assigned to (Domain of the variable)
- A solution is an assignment of each variable to a value from its domain, which is also valid w.r.t. the constraints

VARIABLES AND THEIR DOMAINS

What are variables in CP?

- Variables are the unknown of the problem
- They have a selection of allowed values they can be assigned to (Domain of the variable)
- A solution is an assignment of each variable to a value from its domain, which is also valid w.r.t. the constraints

Exemple of variables:

- X, with dom(X) = {0,1} is a boolean variable
- Y, with dom(Y) = {2,4,6} is an integer variable

CONSTRAINTS

CONSTRAINTS

In Linear Programming, (Mixed) Integer Programming and Pseudo-Boolean, you have linear (in)equalities:

CONSTRAINTS

In Linear Programming, (Mixed) Integer Programming and Pseudo-Boolean, you have linear (in)equalities:

- 2*X+3*Y-4*Z <= 42

CONSTRAINTS

In Linear Programming, (Mixed) Integer Programming and Pseudo-Boolean, you have linear (in)equalities:

- 2*X+3*Y-4*Z <= 42

In SAT, you have CNF clauses:

CONSTRAINTS

In Linear Programming, (Mixed) Integer Programming and Pseudo-Boolean, you have linear (in)equalities:

- 2*X+3*Y-4*Z <= 42

In SAT, you have CNF clauses:

- $(X \vee Y) \wedge (\neg Z)$

CONSTRAINTS

In Linear Programming, (Mixed) Integer Programming and Pseudo-Boolean, you have linear (in)equalities:

- 2*X+3*Y-4*Z <= 42

In SAT, you have CNF clauses:

- $(X \vee Y) \wedge (\neg Z)$

CONSTRAINTS

In Linear Programming, (Mixed) Integer Programming and Pseudo-Boolean, you have linear (in)equalities:

- 2*X+3*Y-4*Z <= 42

In SAT, you have CNF clauses:

- $(X \vee Y) \wedge (\neg Z)$

In Constraint Programming, you have all the above, and more! Such as all the global constraints:

CONSTRAINTS

In Linear Programming, (Mixed) Integer Programming and Pseudo-Boolean, you have linear (in)equalities:

- 2*X+3*Y-4*Z <= 42

In SAT, you have CNF clauses:

- $(X \vee Y) \wedge (\neg Z)$

In Constraint Programming, you have all the above, and more! Such as all the global constraints:

- AllDifferent(X,Y,Z)

CONSTRAINTS

In Linear Programming, (Mixed) Integer Programming and Pseudo-Boolean, you have linear (in)equalities:

- 2*X+3*Y-4*Z <= 42

In SAT, you have CNF clauses:

- $(X \vee Y) \wedge (\neg Z)$

In Constraint Programming, you have all the above, and more! Such as all the global constraints:

- AllDifferent(X,Y,Z)
- Circuit(X,Y,Z)

CONSTRAINTS

In Linear Programming, (Mixed) Integer Programming and Pseudo-Boolean, you have linear (in)equalities:

- 2*X+3*Y-4*Z <= 42

In SAT, you have CNF clauses:

- $(X \vee Y) \wedge (\neg Z)$

In Constraint Programming, you have all the above, and more! Such as all the global constraints:

- AllDifferent(X,Y,Z)
- Circuit(X,Y,Z)
- Cumulative(starts, durations, ressources)

MODELLING IN MORE DETAILS

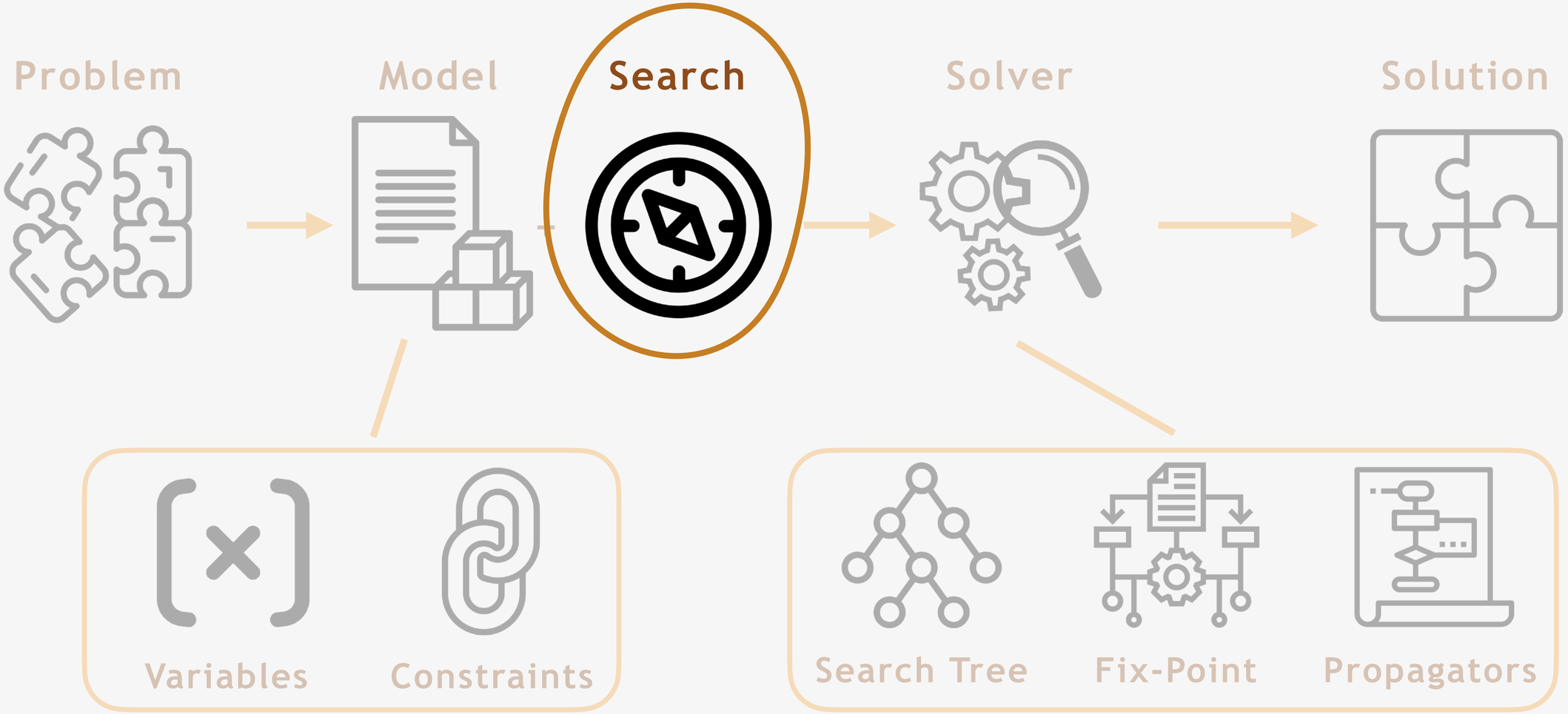
-Monday 11:00-12:30: Lecture

-Monday 14:00-15:30: Lab

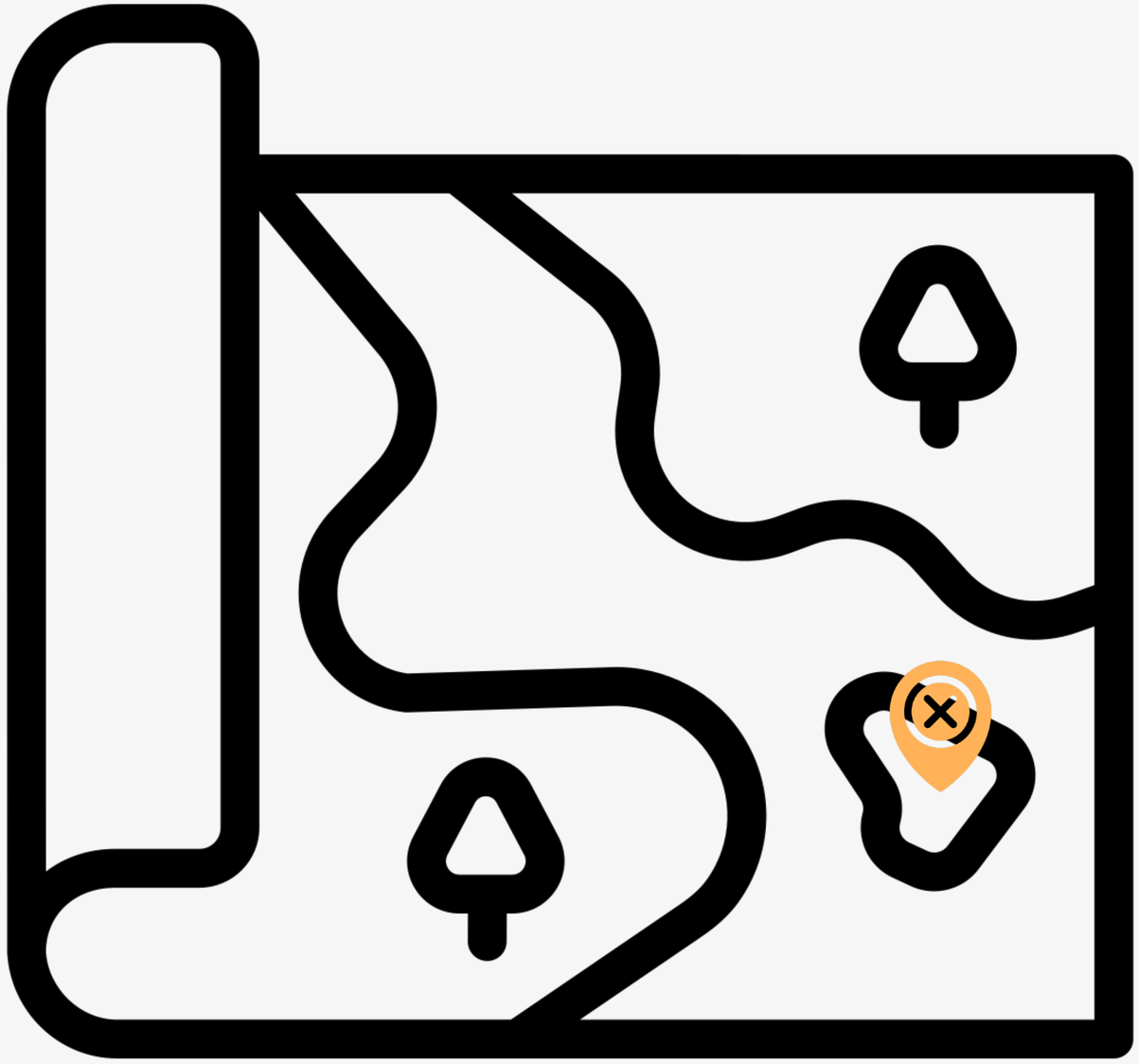
-Monday 16:00-17:30: Lab

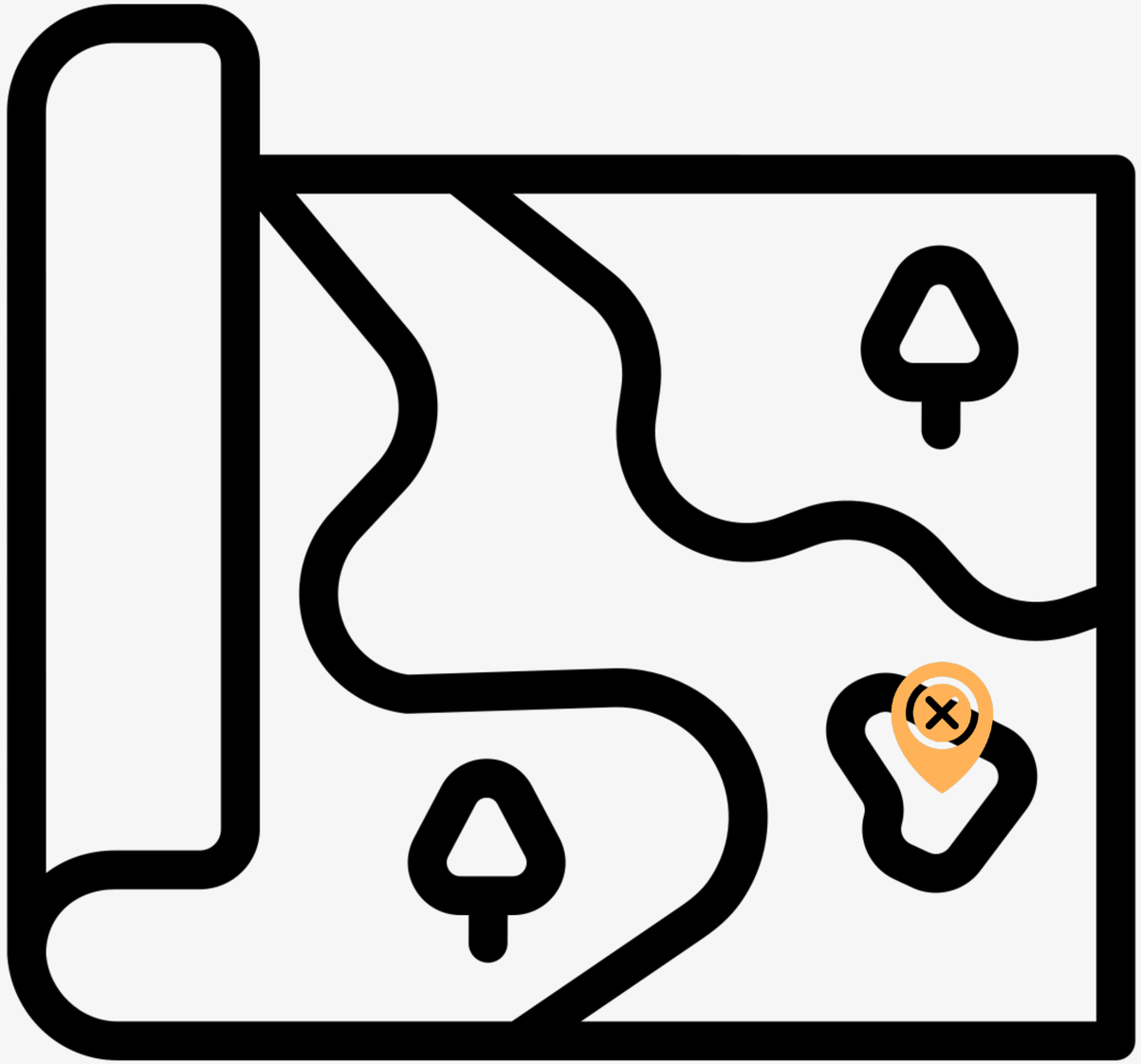
Search

THE CONSTRAINT PROGRAMMING PARADIGM

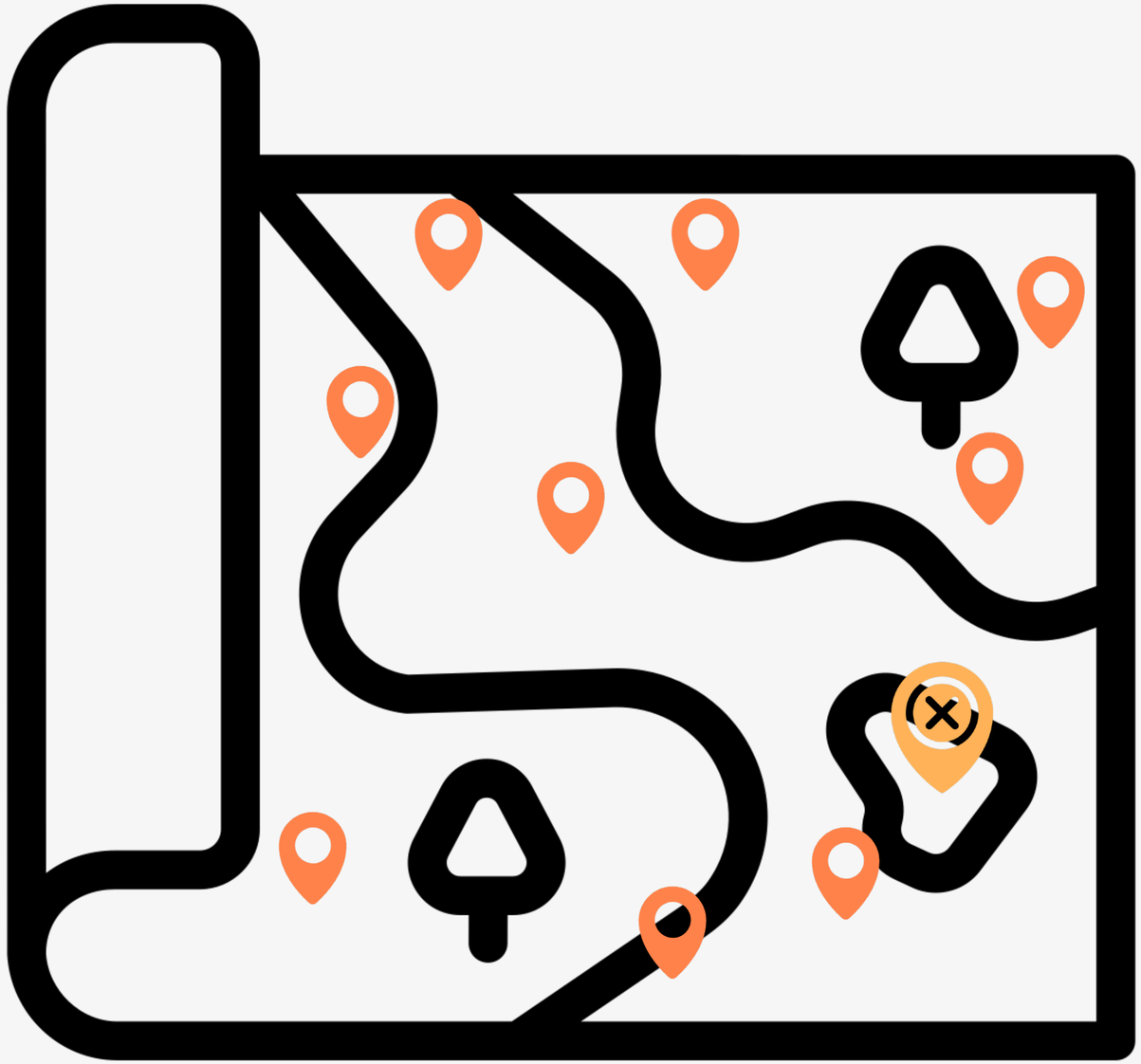


SEARCH



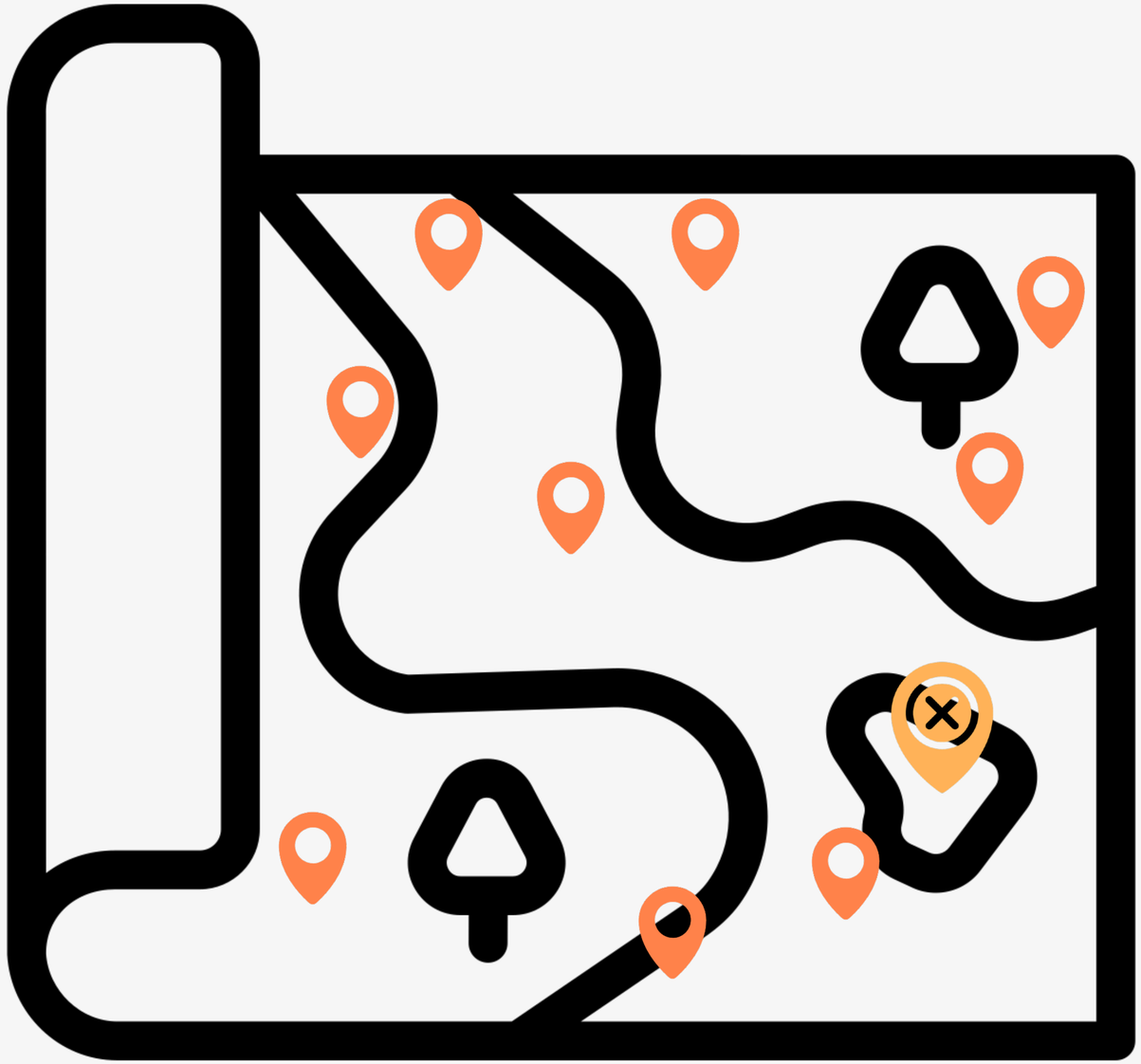


If you know the type of terrain (i.e., the type of problem), then you can have an idea on how to explore it!



If you know the type of terrain (i.e., the type of problem), then you can have an idea on how to explore it!

SEARCH



If you know the type of terrain (i.e., the type of problem), then you can have an idea on how to explore it!

The search given to the solver is guide you select. The choice is often made based on the knowledge you have of the problem

SEARCH IN MORE DETAILS

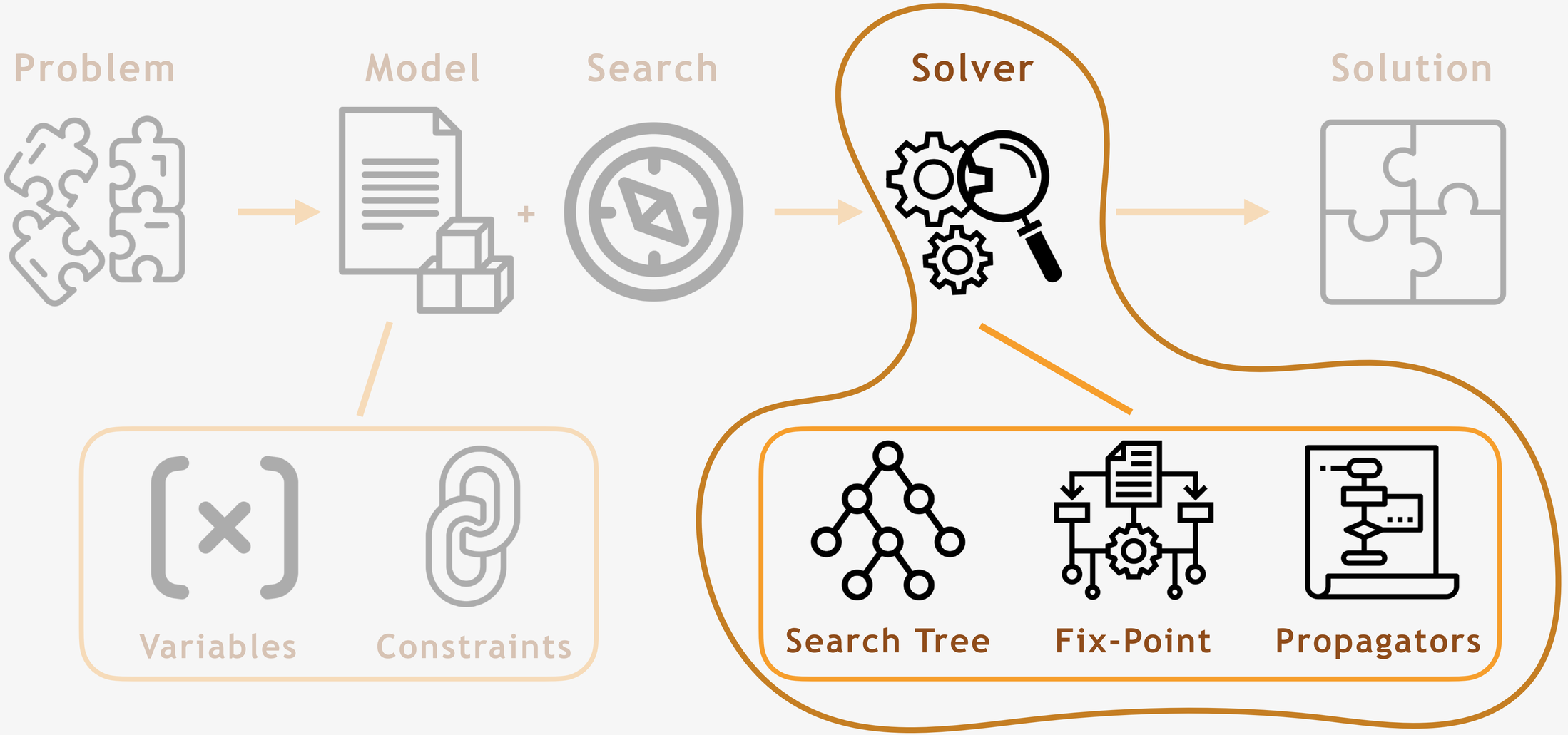
-Wednesday 9:00-10:30: Lecture

-Wednesday 11:00-12:30: Lecture

-Thursday 14:00-15:30: Lab

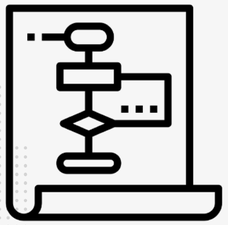
The Solver

THE CONSTRAINT PROGRAMMING PARADIGM

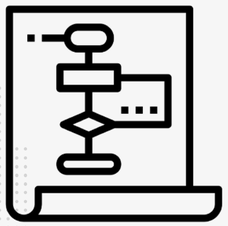


Propagators

GOAL OF A PROPAGATOR

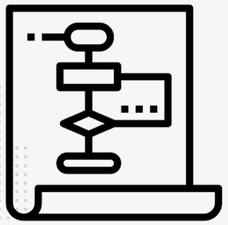


GOAL OF A PROPAGATOR



Goal: specialised algorithms which take the current state (i.e., variables and current domains) and filter invalid values using some reasoning corresponding to the constraint

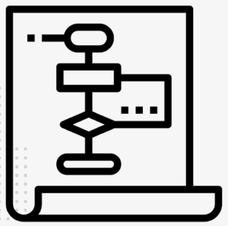
GOAL OF A PROPAGATOR



Goal: specialised algorithms which take the current state (i.e., variables and current domains) and filter invalid values using some reasoning corresponding to the constraint

Example:

GOAL OF A PROPAGATOR

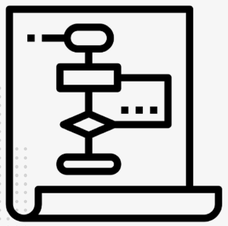


Goal: specialised algorithms which take the current state (i.e., variables and current domains) and filter invalid values using some reasoning corresponding to the constraint

Example:

- Variables X, Y, Z, with dom(X) = dom(Y) = {0,1} and dom(Z) = {0,1,2}

GOAL OF A PROPAGATOR

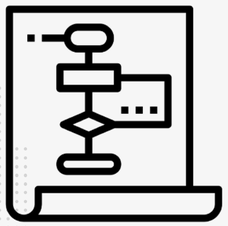


Goal: specialised algorithms which take the current state (i.e., variables and current domains) and filter invalid values using some reasoning corresponding to the constraint

Example:

- Variables X, Y, Z, with dom(X) = dom(Y) = {0,1} and dom(Z) = {0,1,2}
- Constraint AllDifferent(X,Y,Z)

GOAL OF A PROPAGATOR



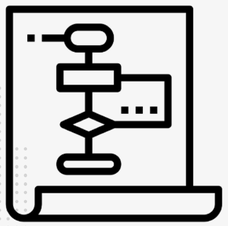
Goal: specialised algorithms which take the current state (i.e., variables and current domains) and filter invalid values using some reasoning corresponding to the constraint

Example:

- Variables X, Y, Z, with dom(X) = dom(Y) = {0,1} and dom(Z) = {0,1,2}
- Constraint AllDifferent(X,Y,Z)

- State before propagation: dom(X) = dom(Y) = {0,1} and dom(Z) = {0,1,2}

GOAL OF A PROPAGATOR



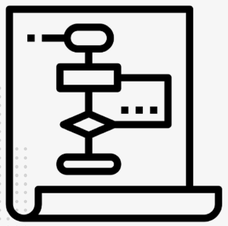
Goal: specialised algorithms which take the current state (i.e., variables and current domains) and filter invalid values using some reasoning corresponding to the constraint

Example:

- Variables X, Y, Z, with dom(X) = dom(Y) = {0,1} and dom(Z) = {0,1,2}
- Constraint AllDifferent(X,Y,Z)

- State before propagation: dom(X) = dom(Y) = {0,1} and dom(Z) = {0,1,2}
- By allDifferent reasoning, Z cannot be 0 or 1

GOAL OF A PROPAGATOR



Goal: specialised algorithms which take the current state (i.e., variables and current domains) and filter invalid values using some reasoning corresponding to the constraint

Example:

- Variables X, Y, Z, with dom(X) = dom(Y) = {0,1} and dom(Z) = {0,1,2}
- Constraint AllDifferent(X,Y,Z)

- State before propagation: dom(X) = dom(Y) = {0,1} and dom(Z) = {0,1,2}
- By allDifferent reasoning, Z cannot be 0 or 1
- State after propagation: dom(X) = dom(Y) = {0,1} and dom(Z) = {2}

PROPAGATORS IN MORE DETAILS

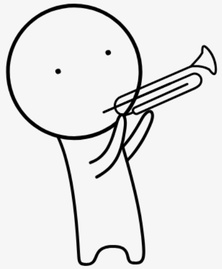
-Tuesday 9:00-10:30: Lecture

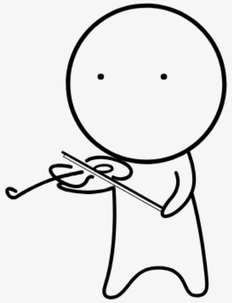
-Tuesday 11:00-12:30: Lecture

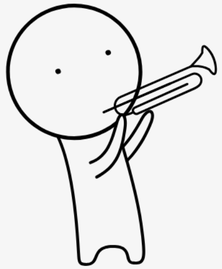
-Tuesday 14:00-15:30: Lab

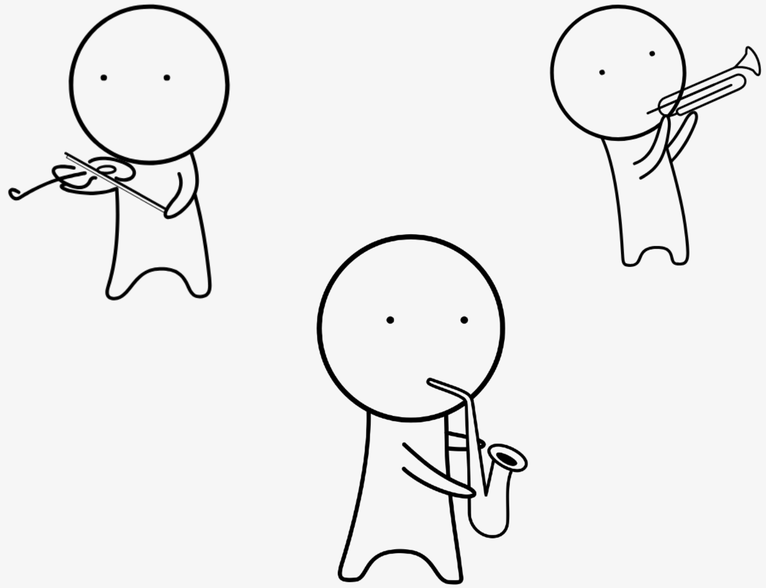
-Tuesday 16:00-17:30: Lab

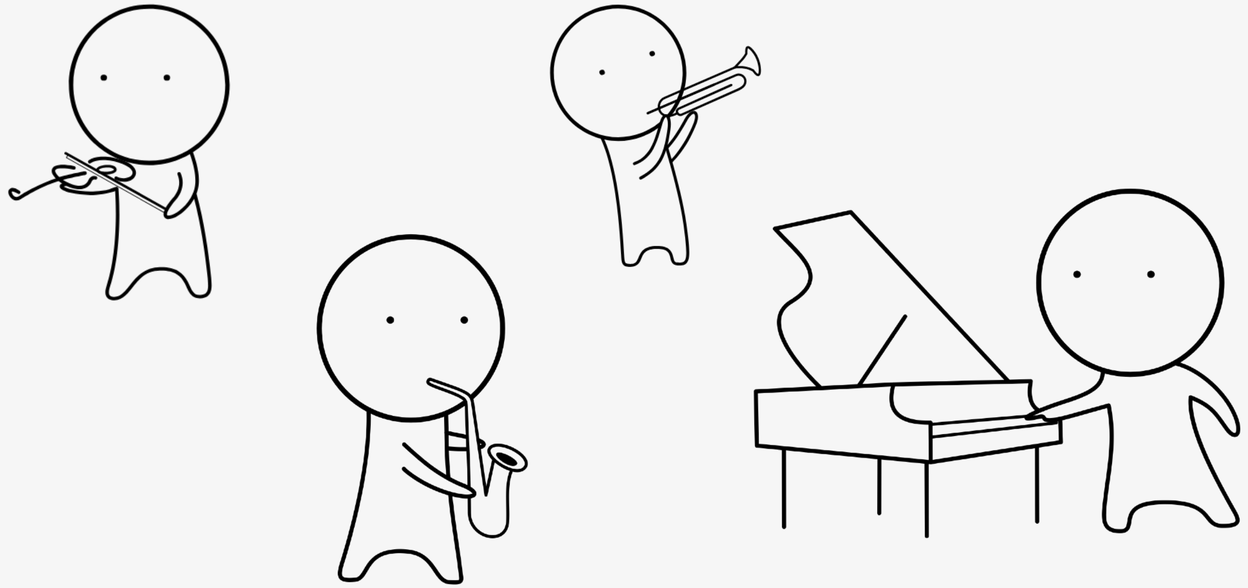
Fixpoint Algorithm

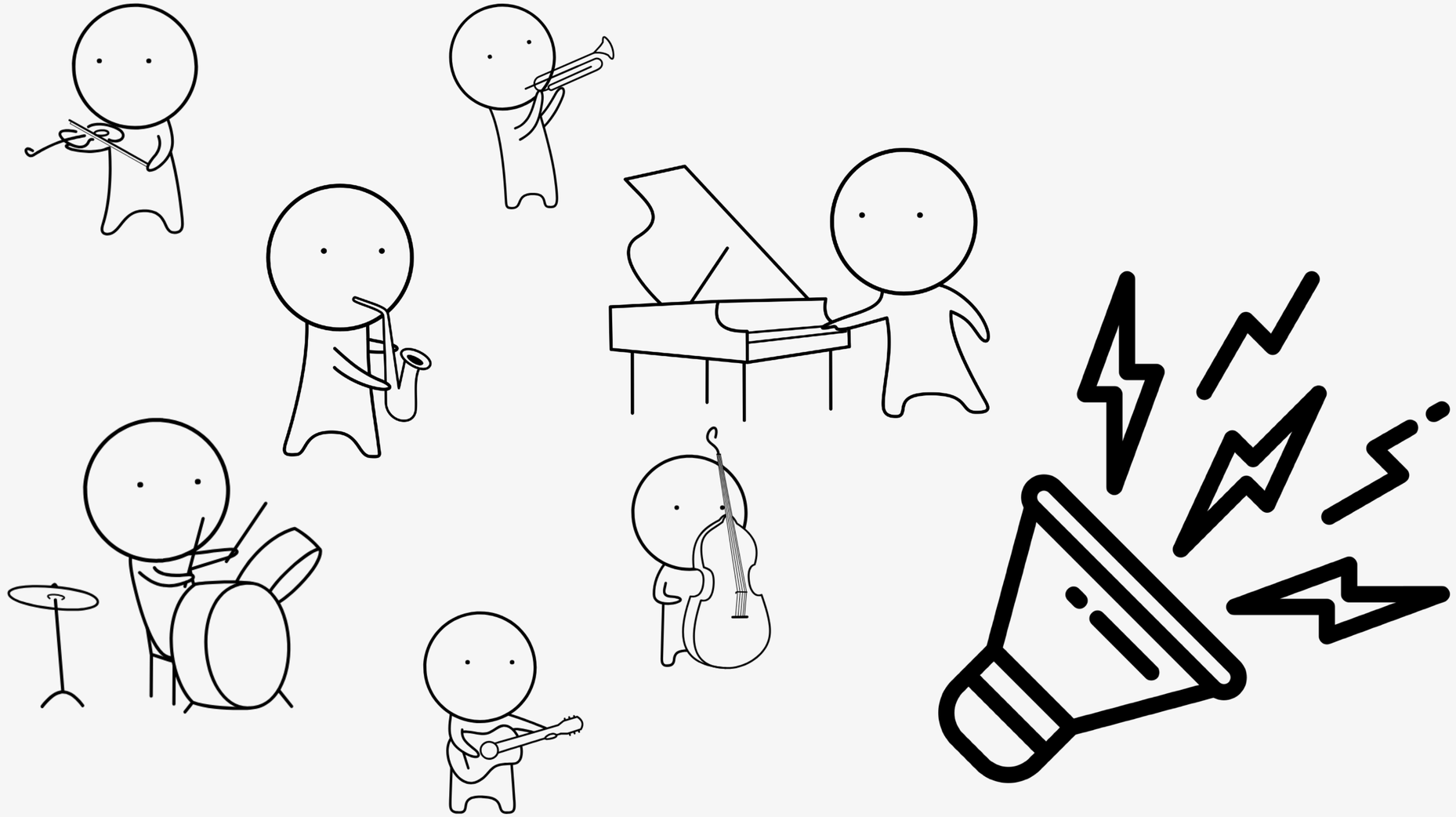


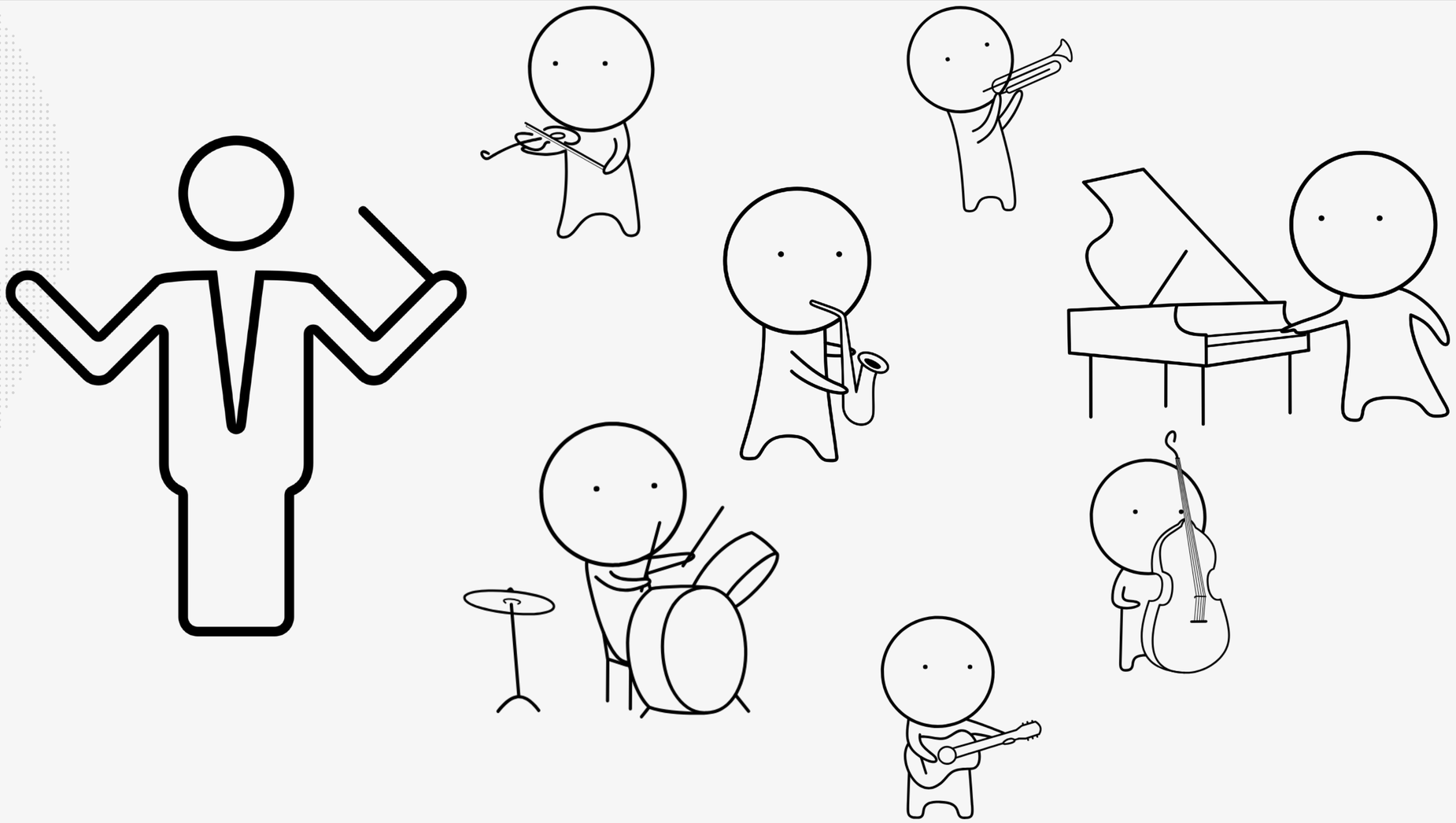


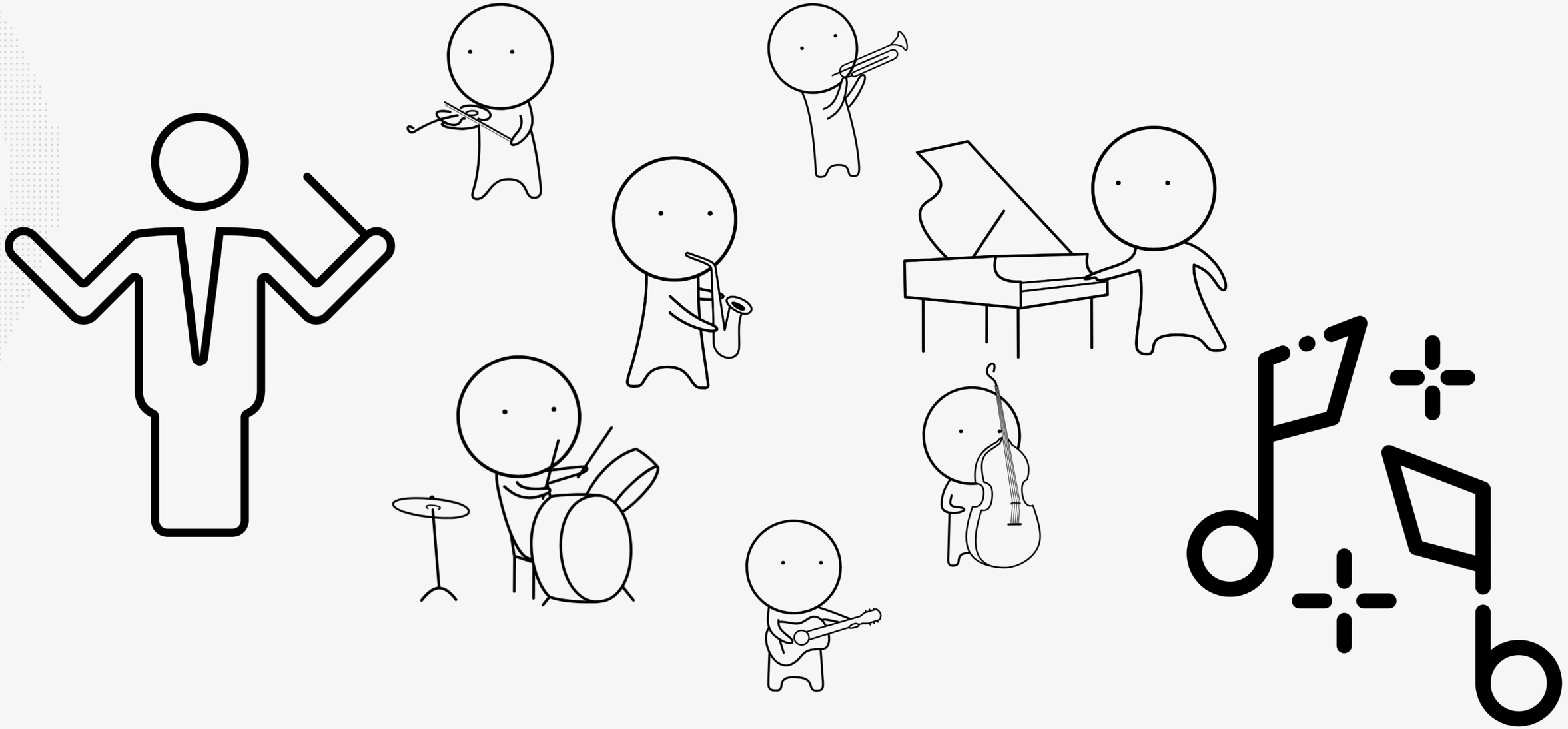




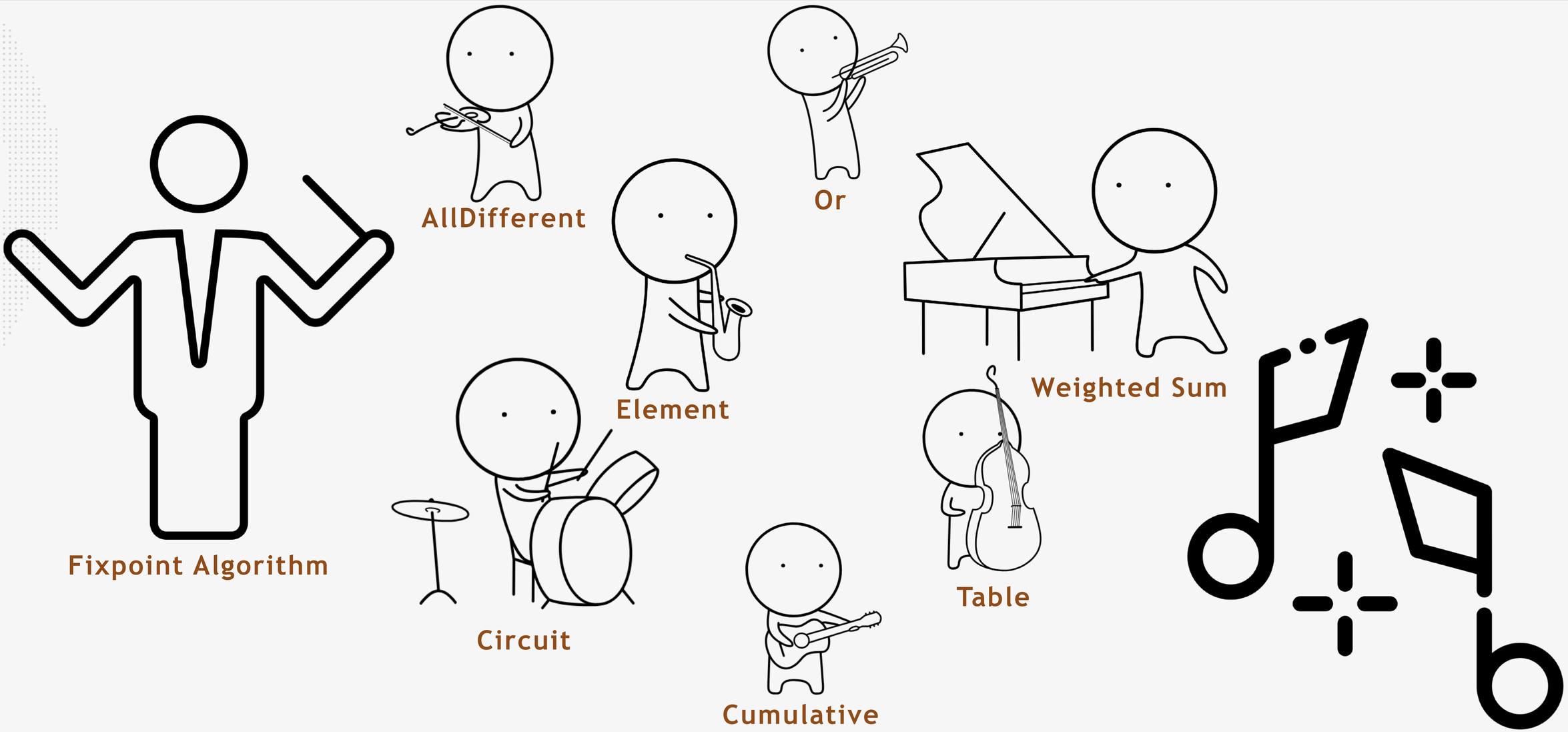






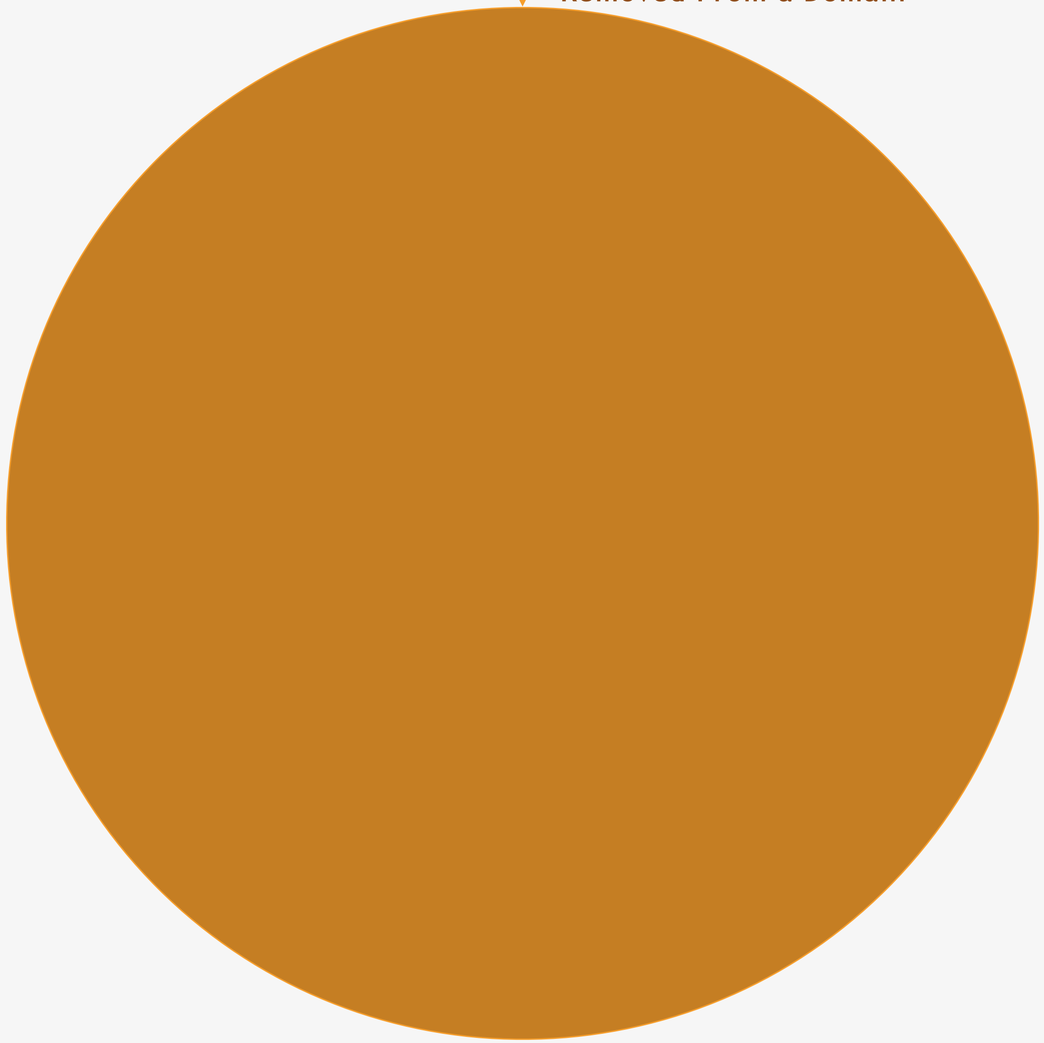


MAESTRO OF THE CONSTRAINTS PROPAGATORS

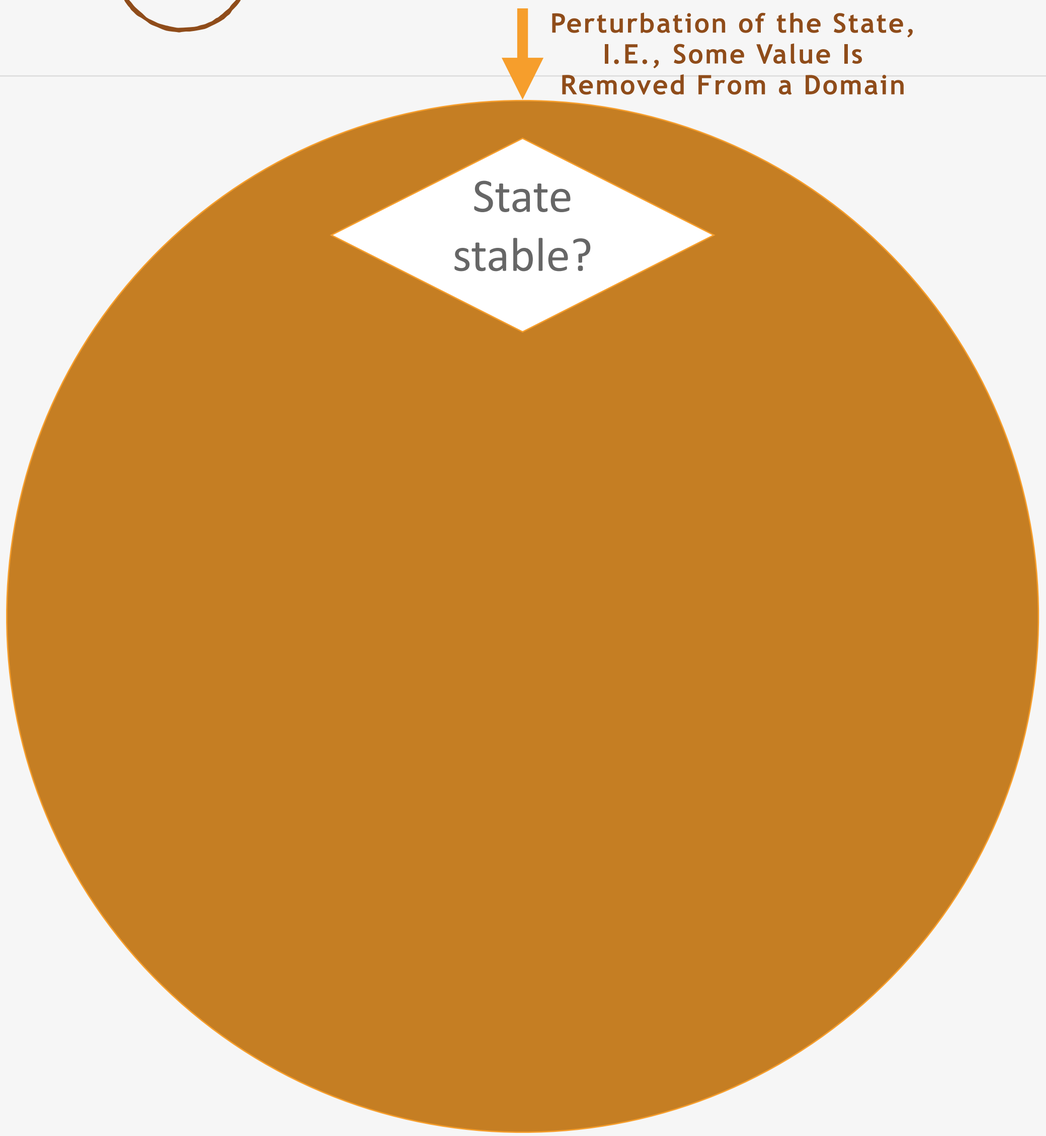


REACHING A STABLE STATE AGAIN

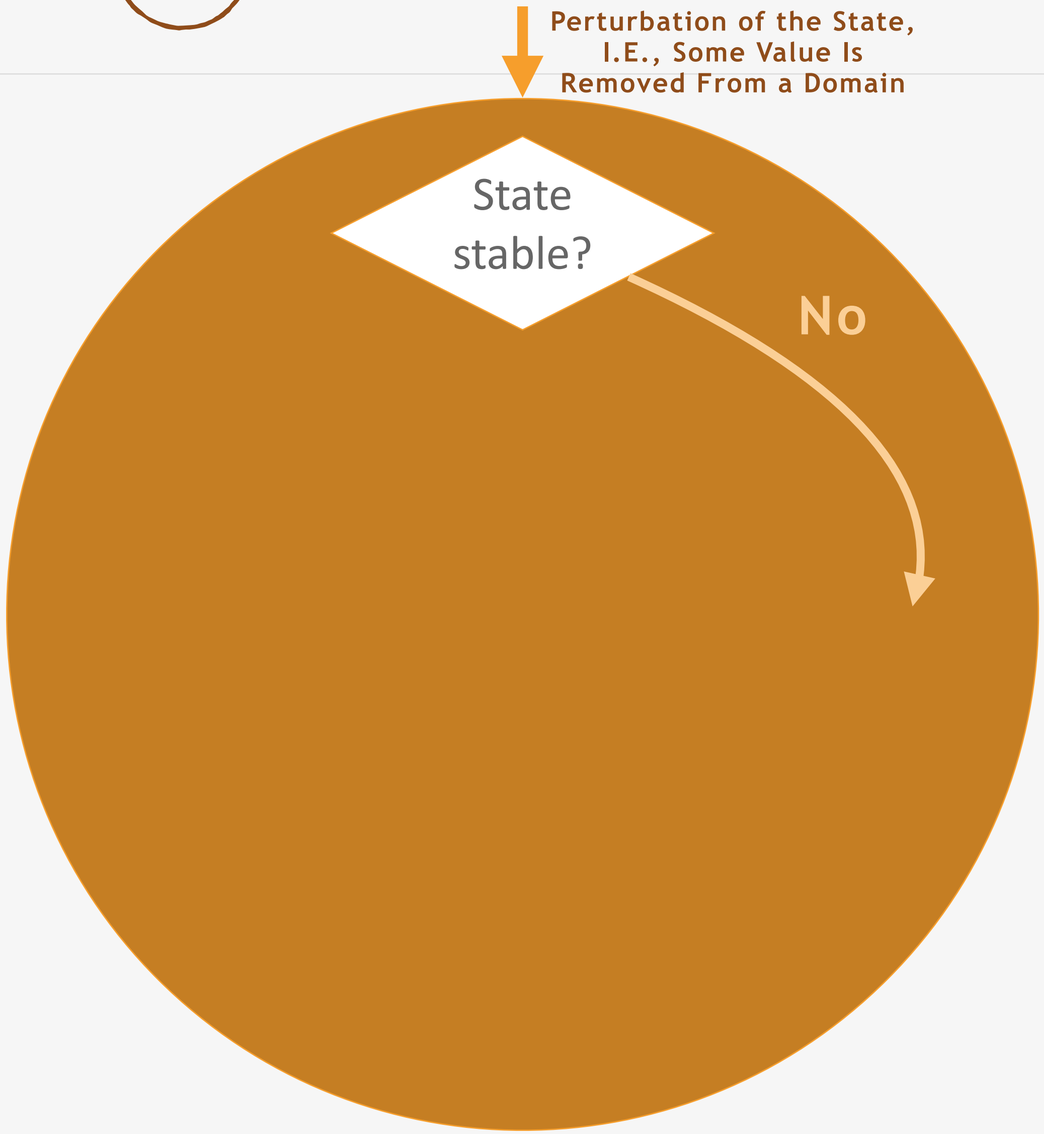
Perturbation of the State,
I.E., Some Value Is
Removed From a Domain



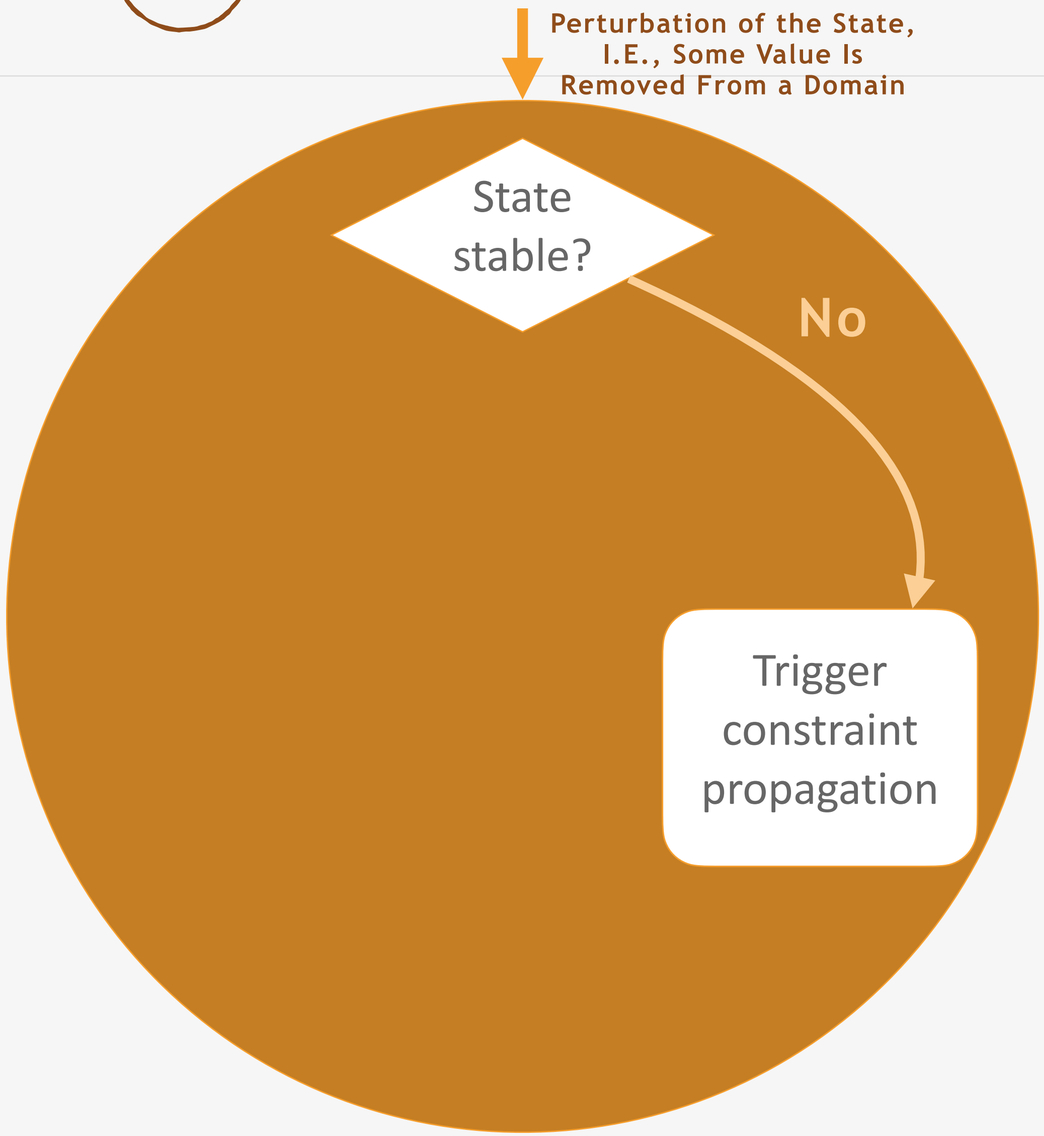
REACHING A STABLE STATE AGAIN



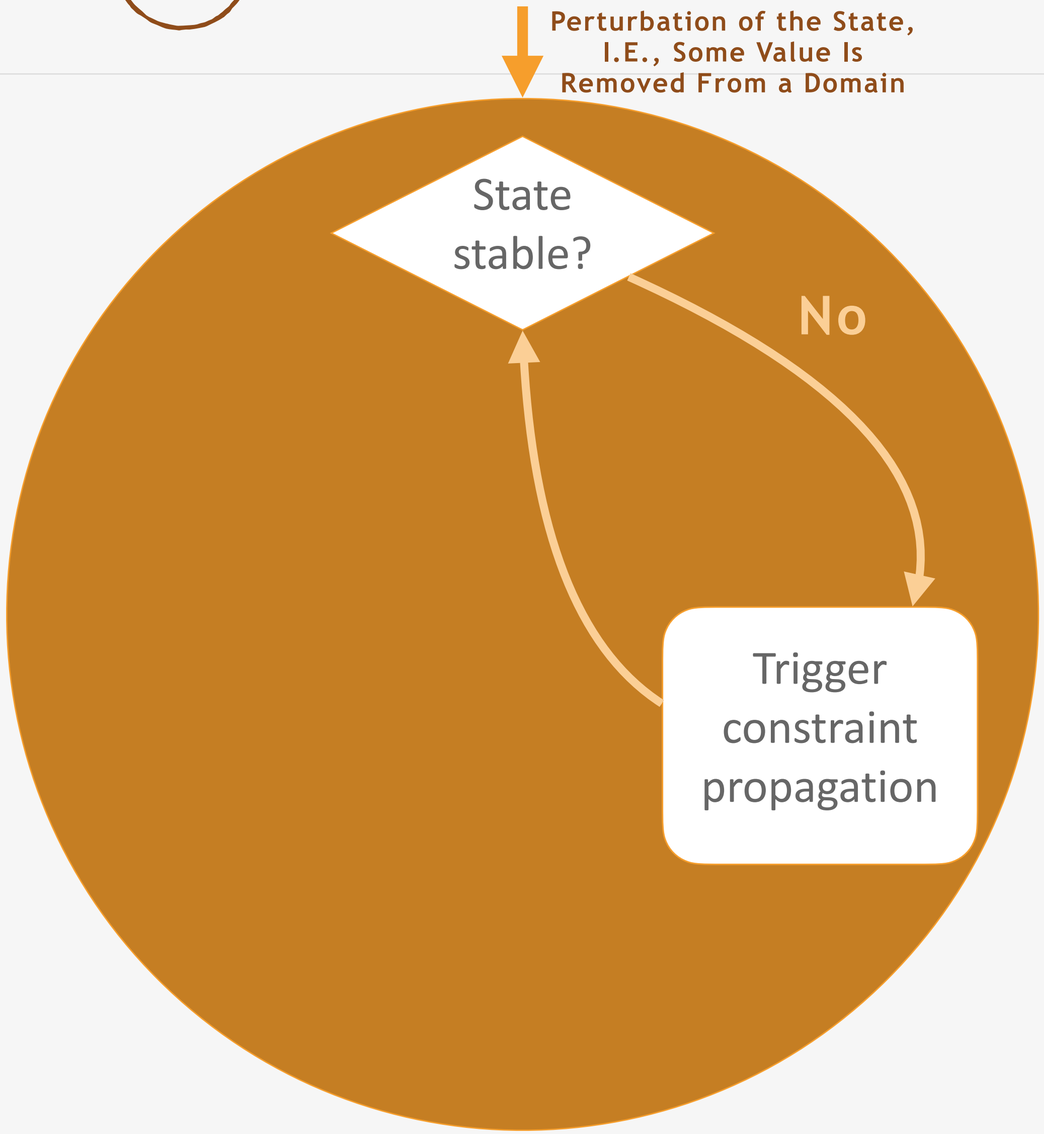
REACHING A STABLE STATE AGAIN



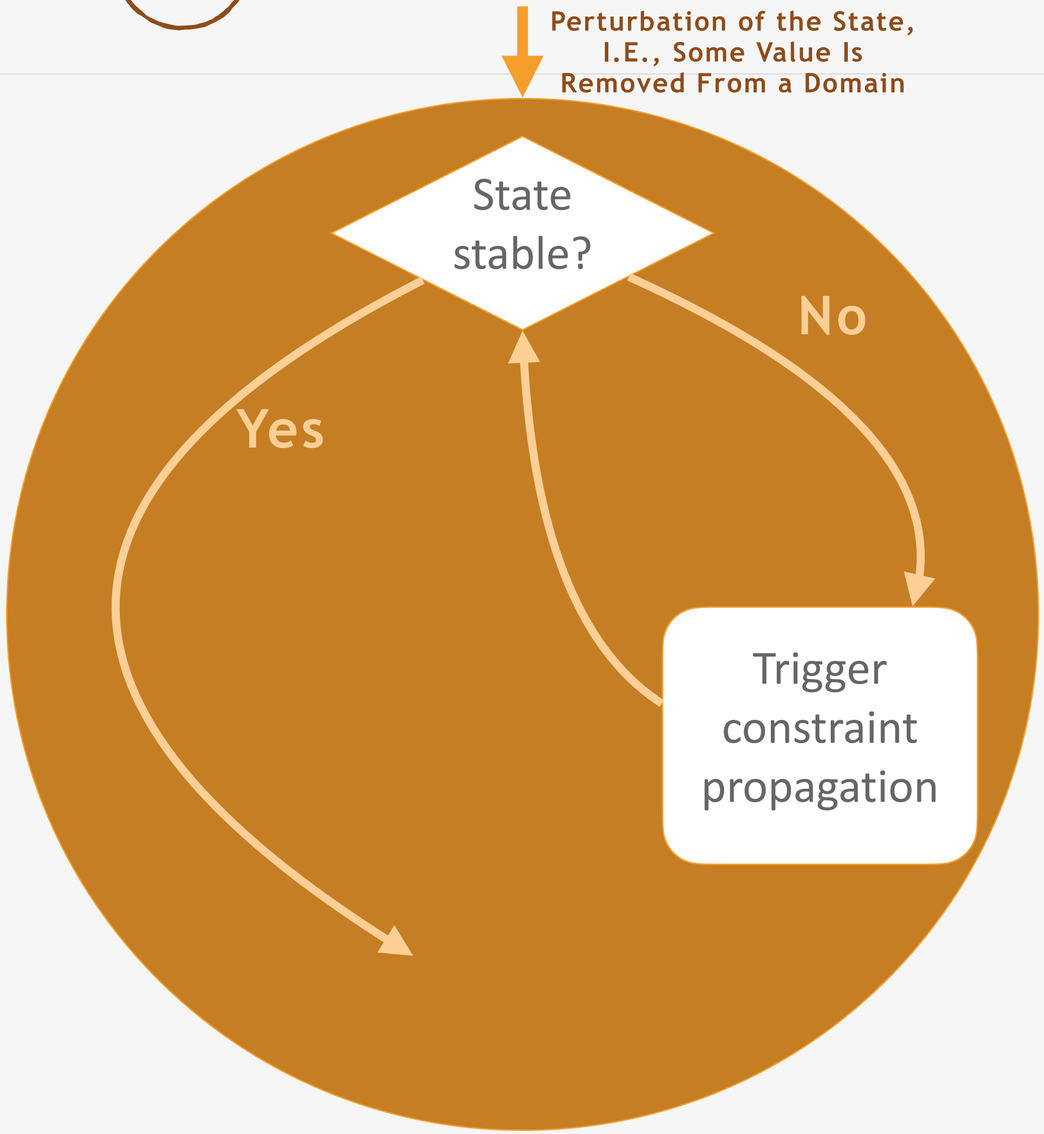
REACHING A STABLE STATE AGAIN



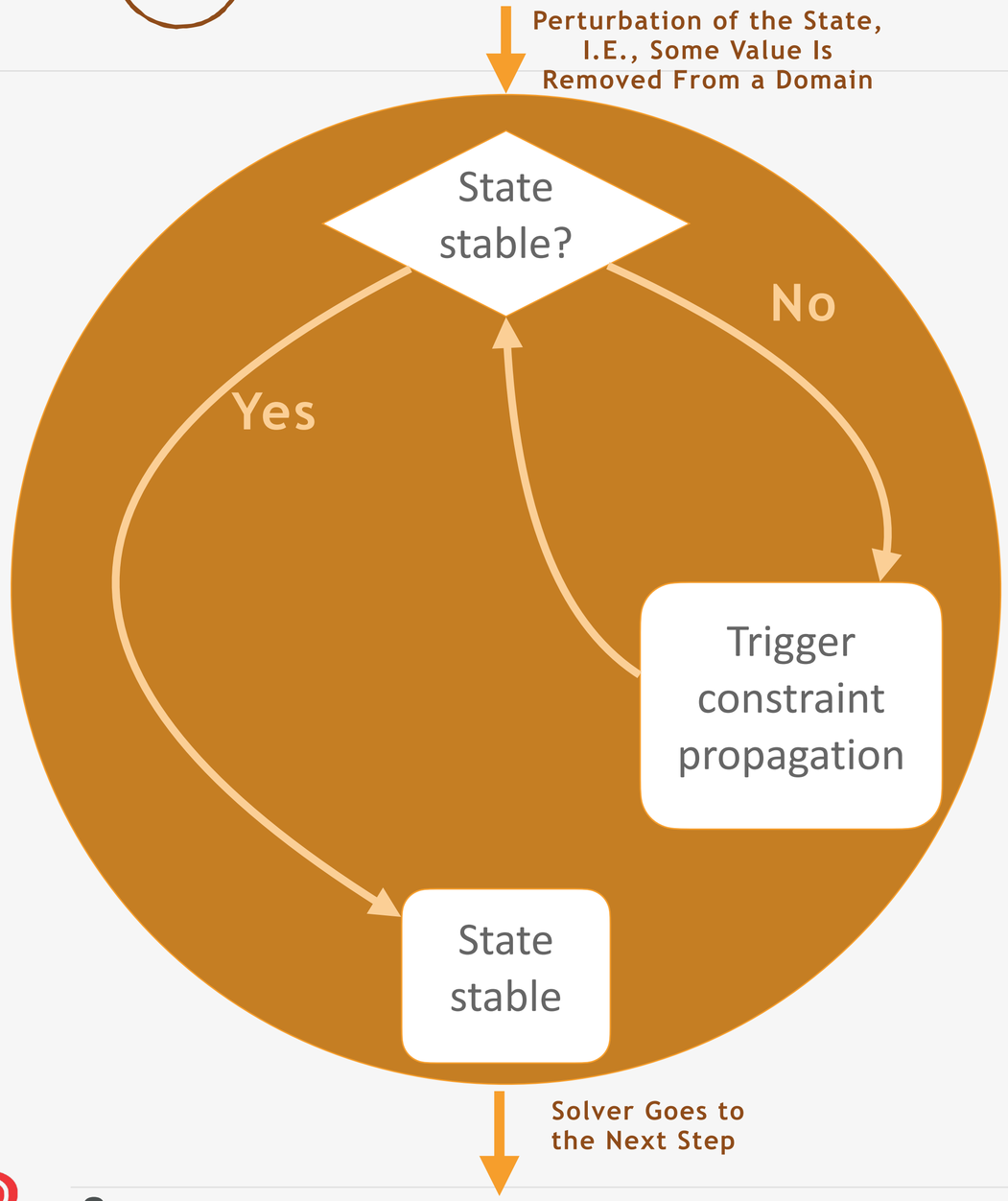
REACHING A STABLE STATE AGAIN



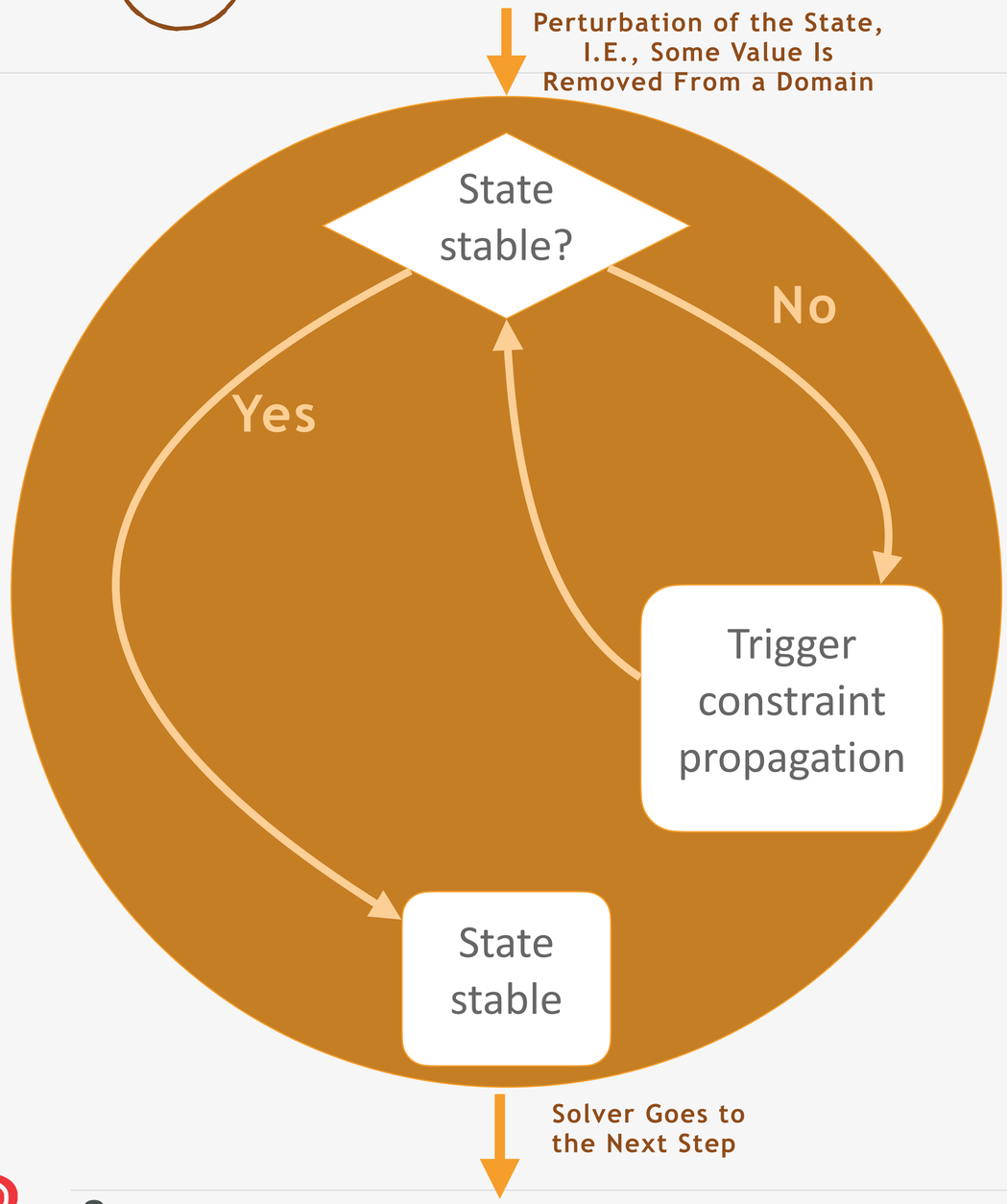
REACHING A STABLE STATE AGAIN



REACHING A STABLE STATE AGAIN

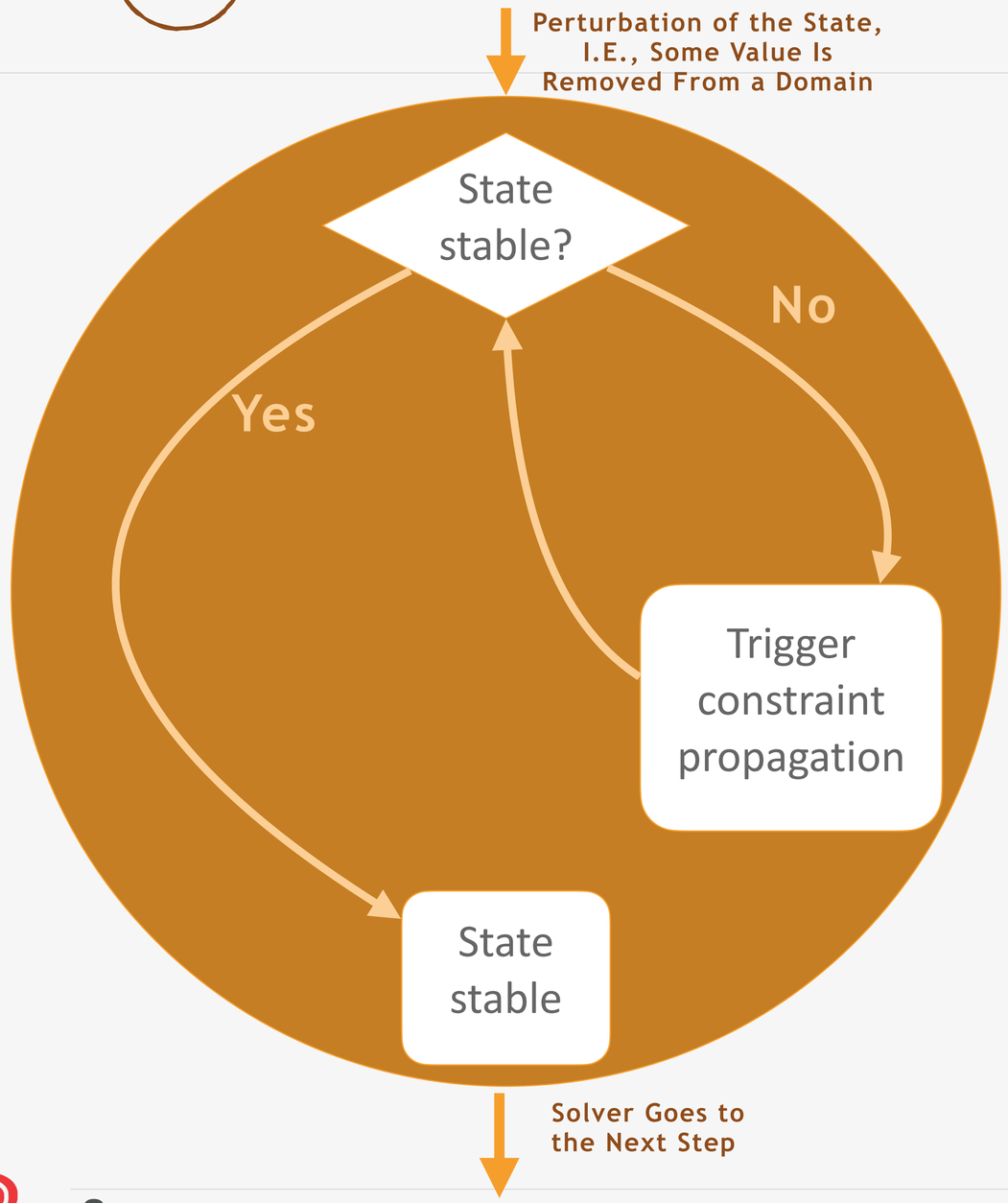


REACHING A STABLE STATE AGAIN



What Is a Stable State?

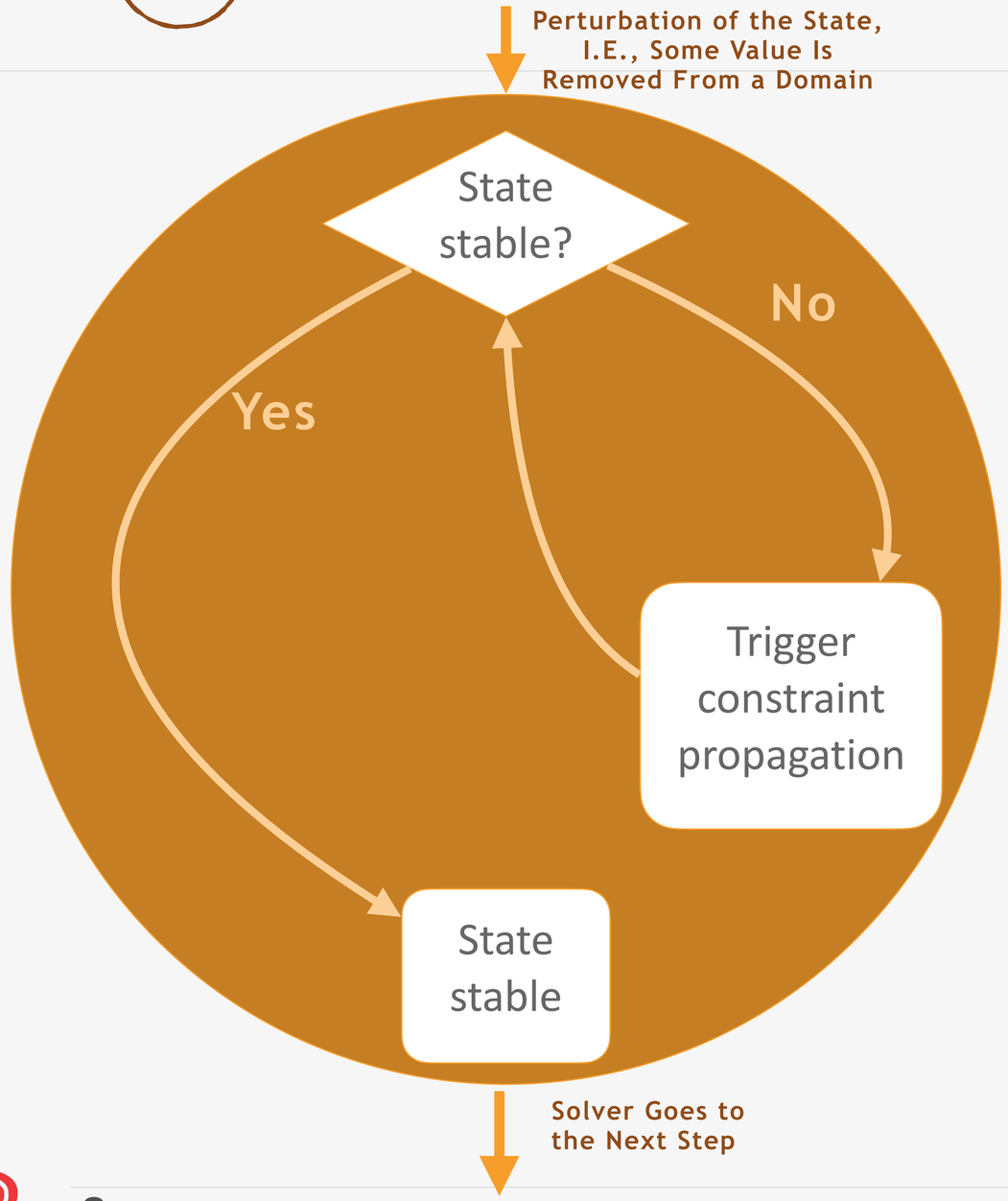
REACHING A STABLE STATE AGAIN



What Is a Stable State?

When no More Constraints Have Been Flagged “To Be Triggered”

REACHING A STABLE STATE AGAIN

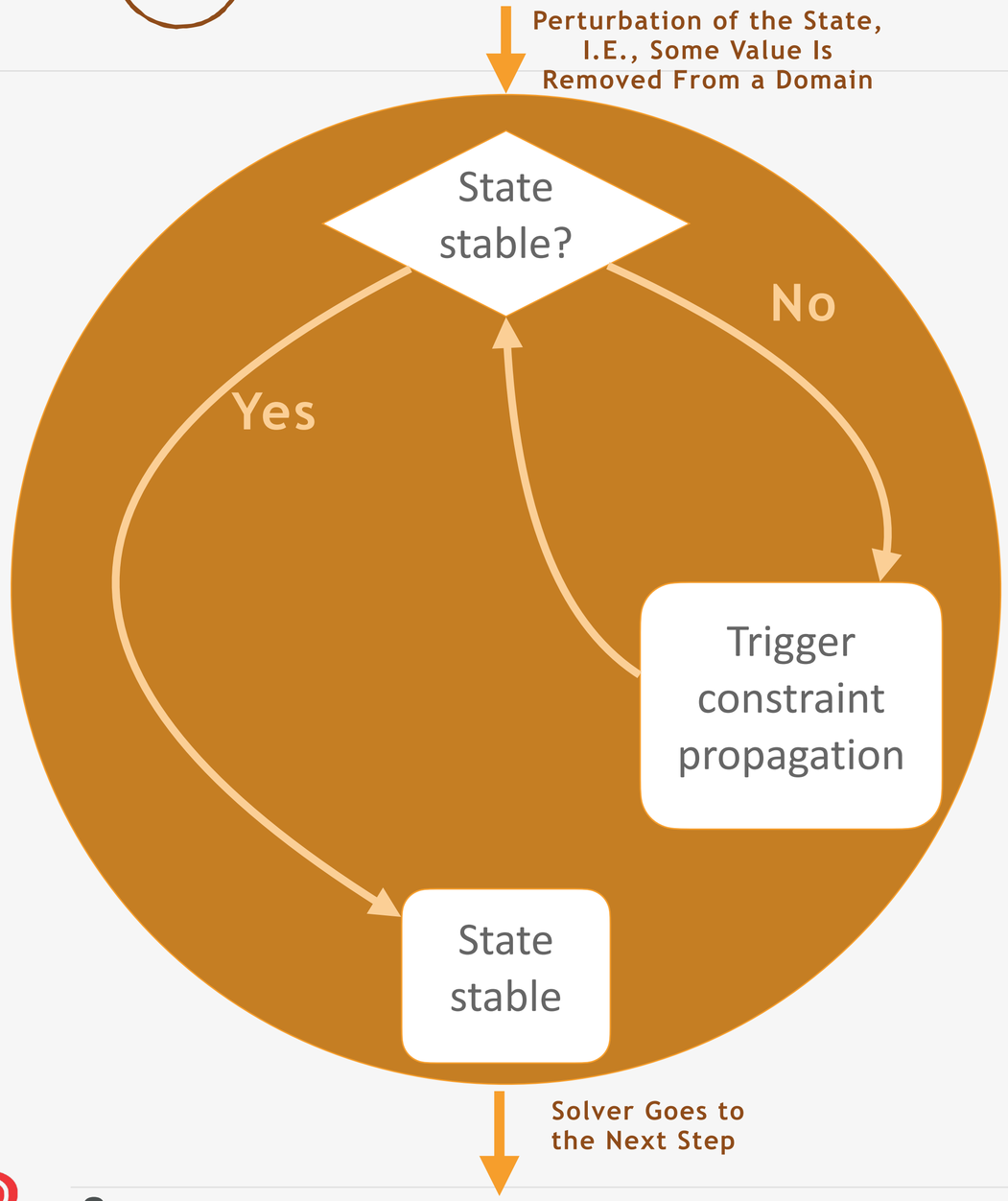


What Is a Stable State?

When no More Constraints Have Been Flagged “To Be Triggered”

Which Constraint To Trigger Next?

REACHING A STABLE STATE AGAIN



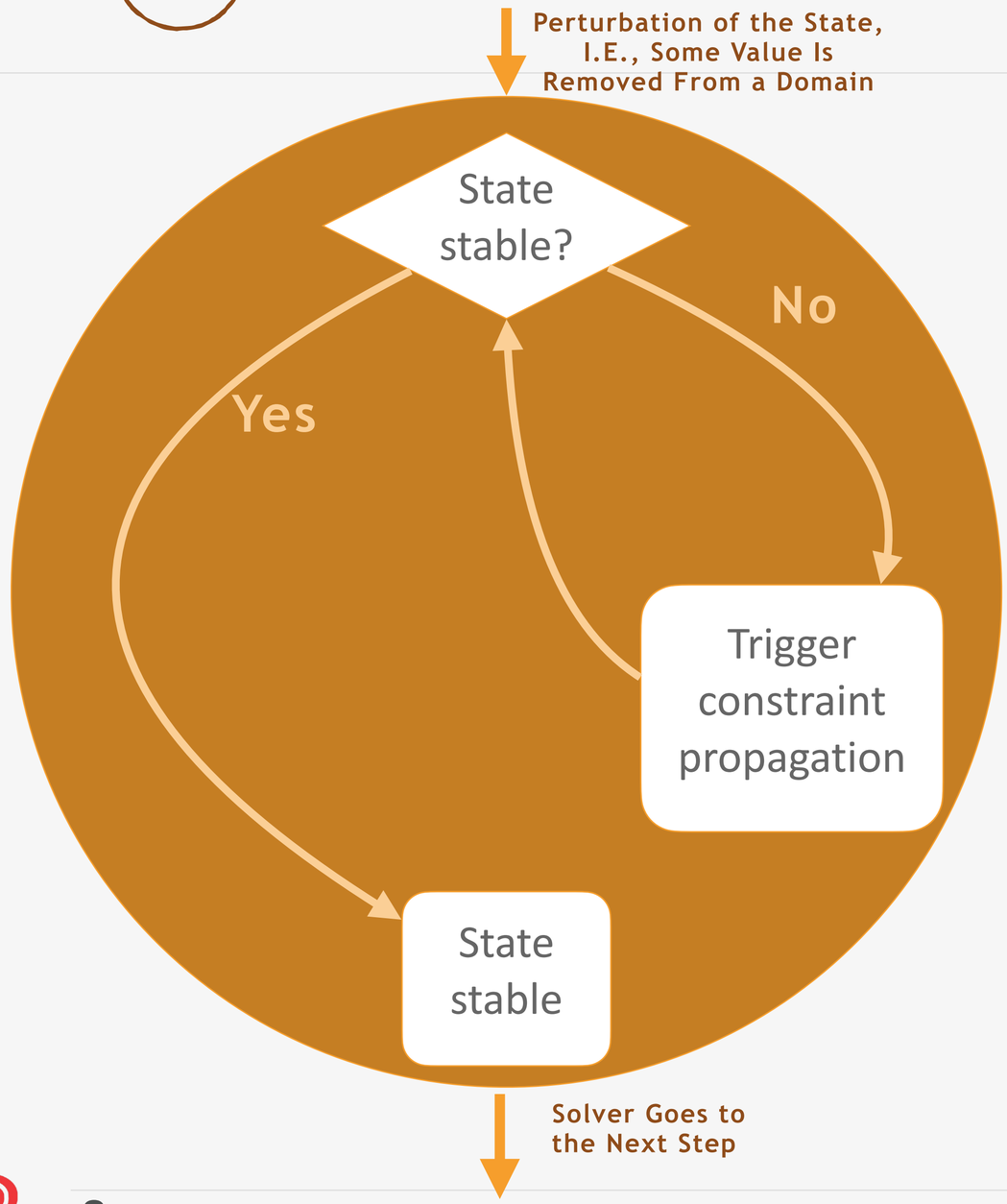
What Is a Stable State?

When no More Constraints Have Been Flagged “To Be Triggered”

Which Constraint To Trigger Next?

Depend on the Priorities of the Constraints (Priority Queue)

REACHING A STABLE STATE AGAIN



What Is a Stable State?

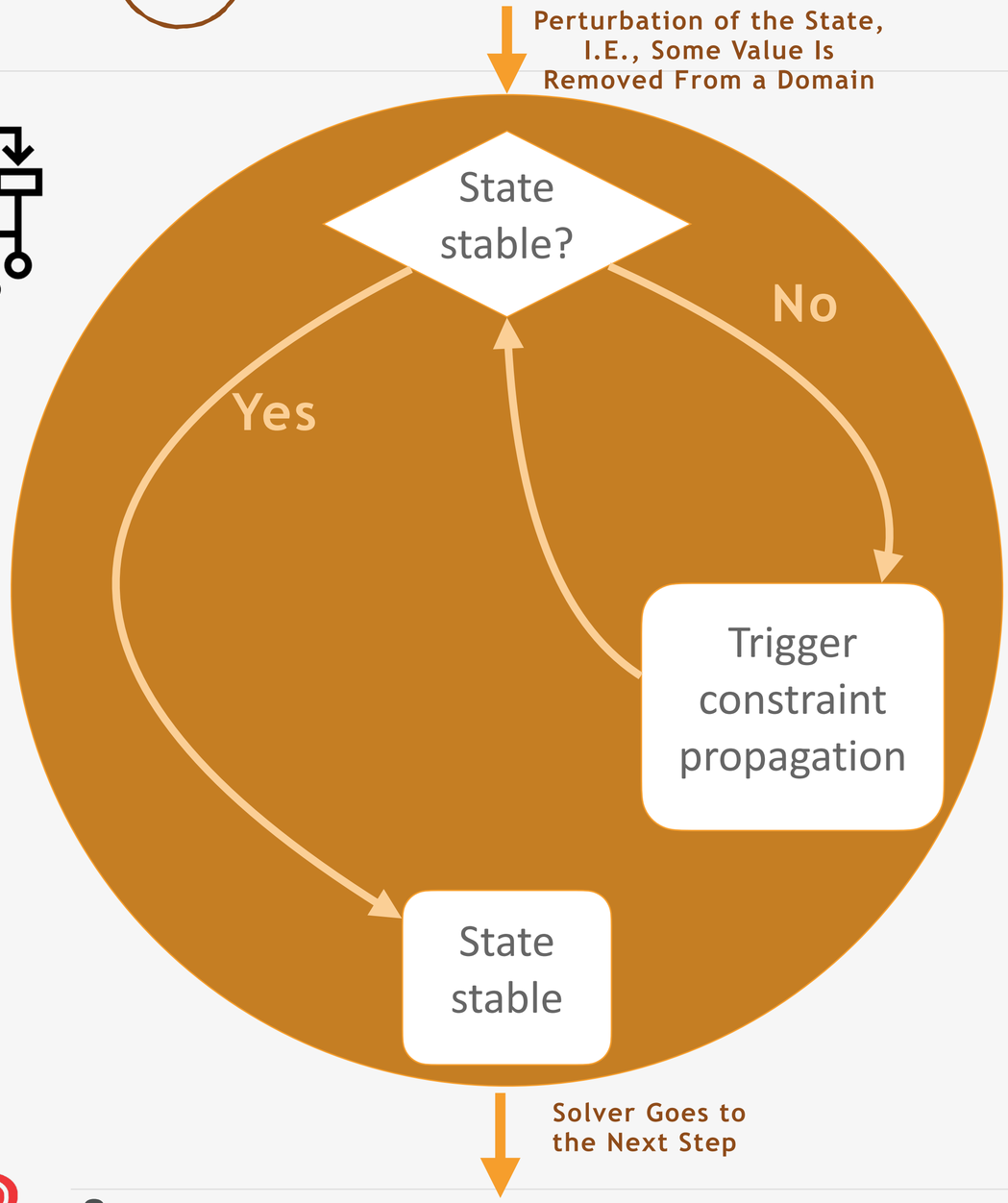
When no More Constraints Have Been Flagged “To Be Triggered”

Which Constraint To Trigger Next?

Depend on the Priorities of the Constraints (Priority Queue)

Propagation Will Remove Values From Domains, Which Then Will Flagged Constraints as “To Be Triggered”

REACHING A STABLE STATE AGAIN



What Is a Stable State?

When no More Constraints Have Been Flagged “To Be Triggered”

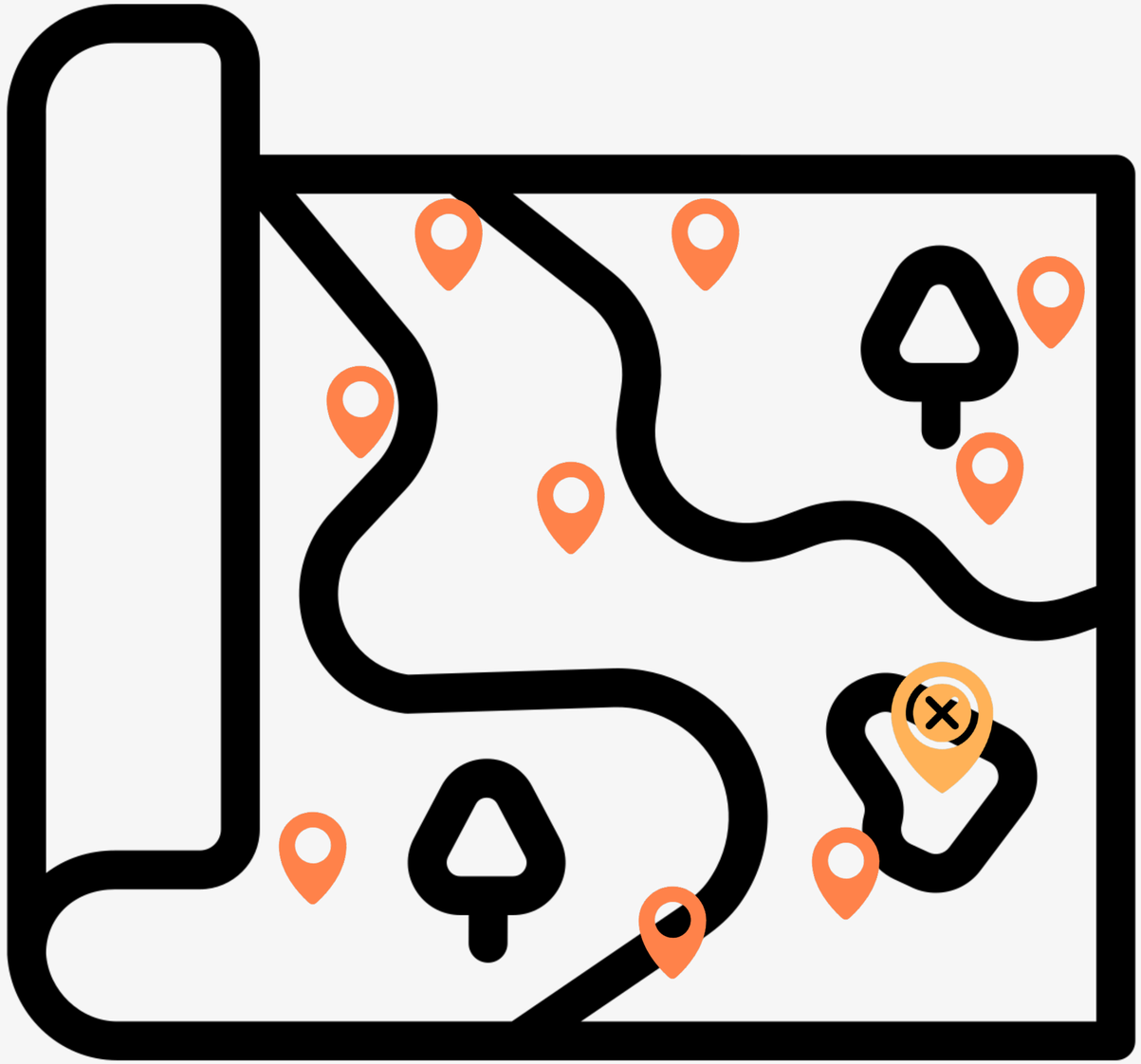
Which Constraint To Trigger Next?

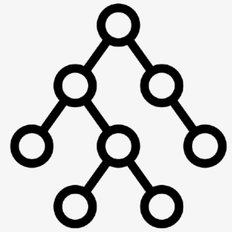
Depend on the Priorities of the Constraints (Priority Queue)

Propagation Will Remove Values From Domains, Which Then Will Flagged Constraints as “To Be Triggered”

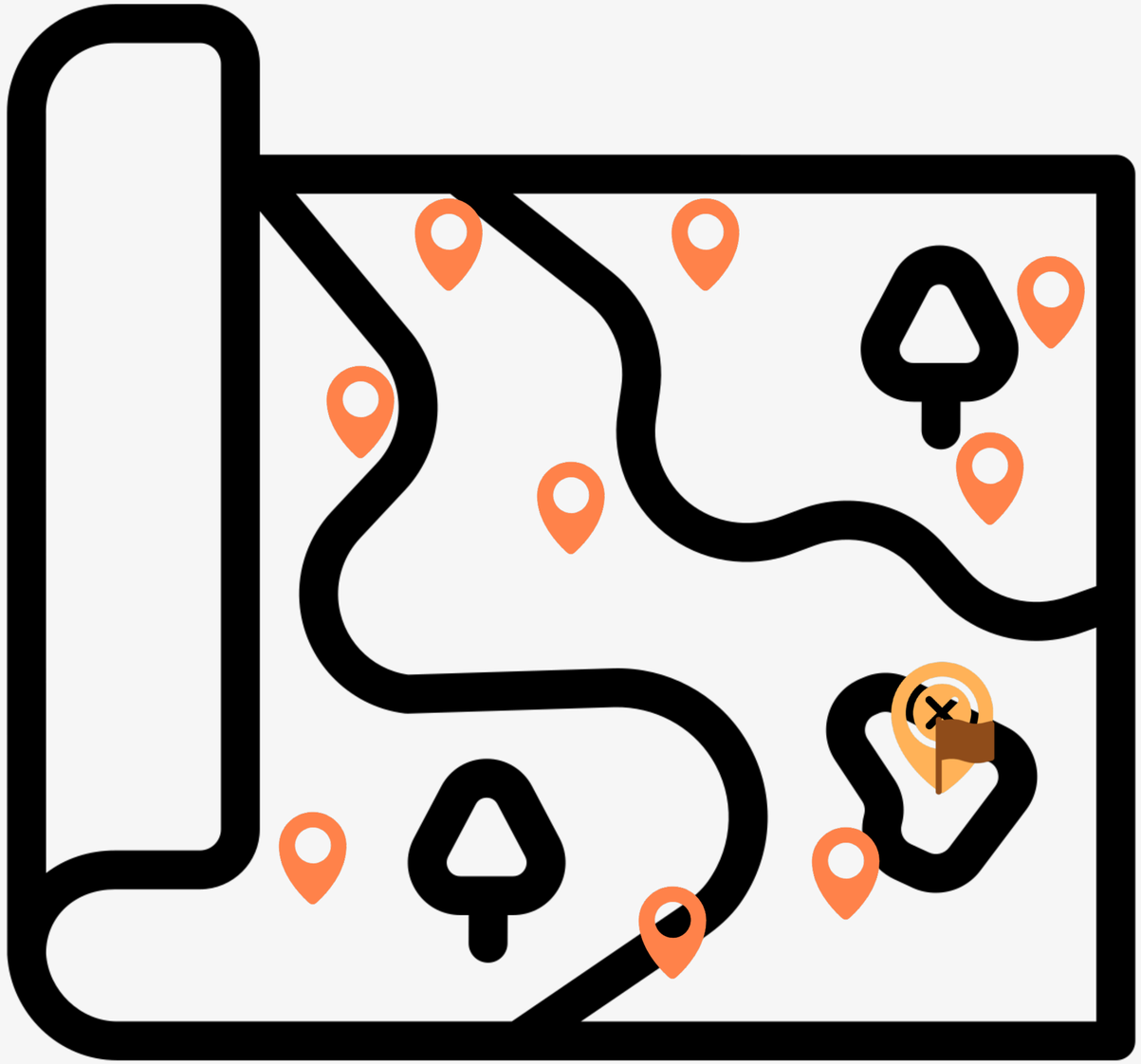
Search Tree

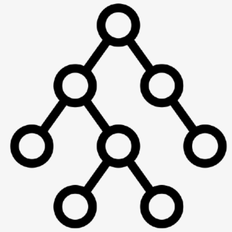
SEARCH TREE



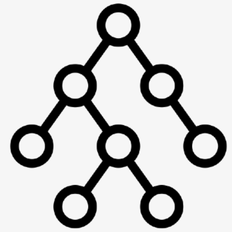


SEARCH TREE

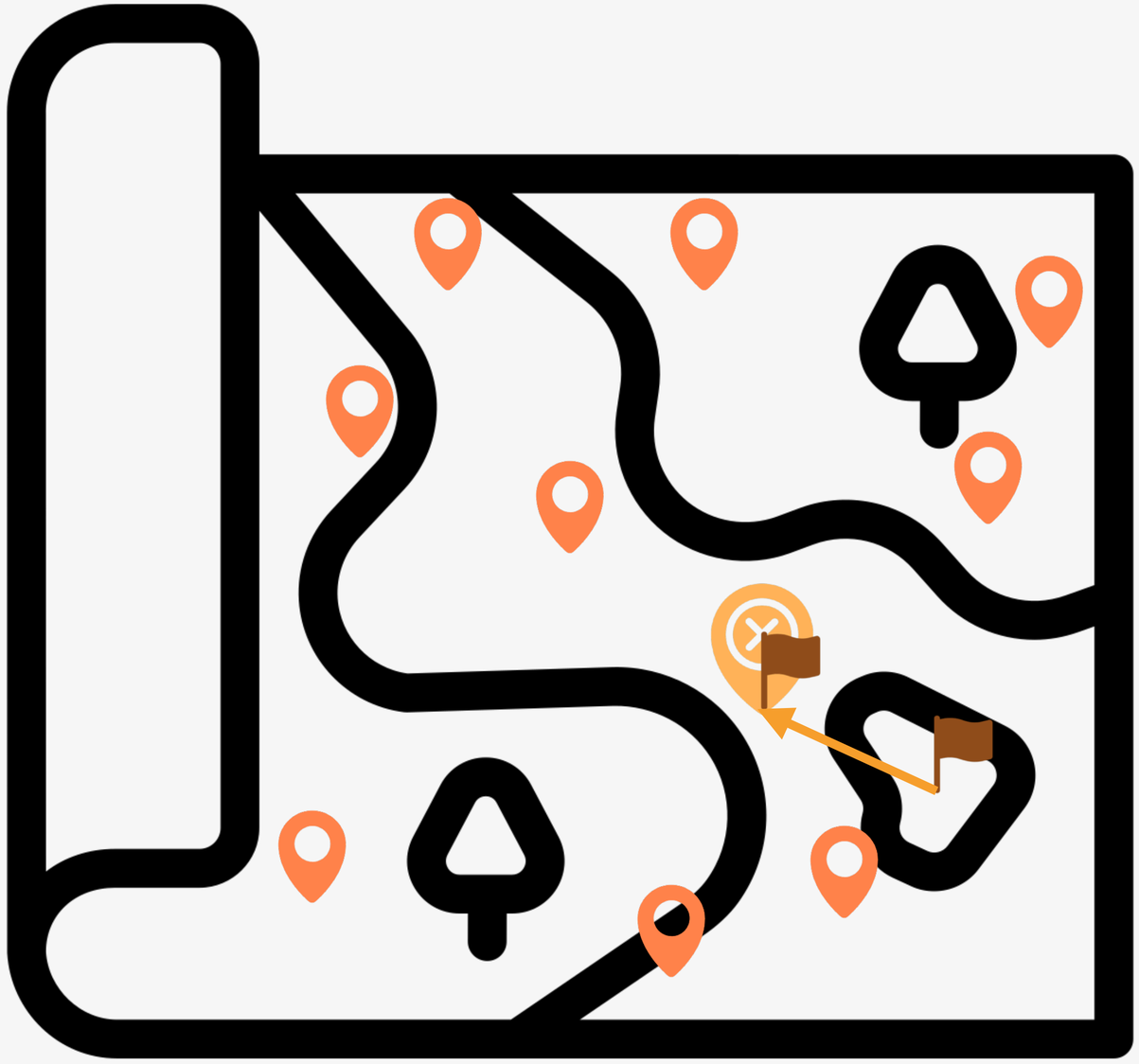


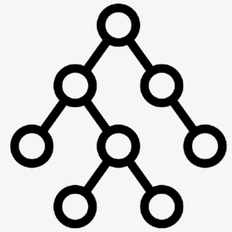


SEARCH TREE

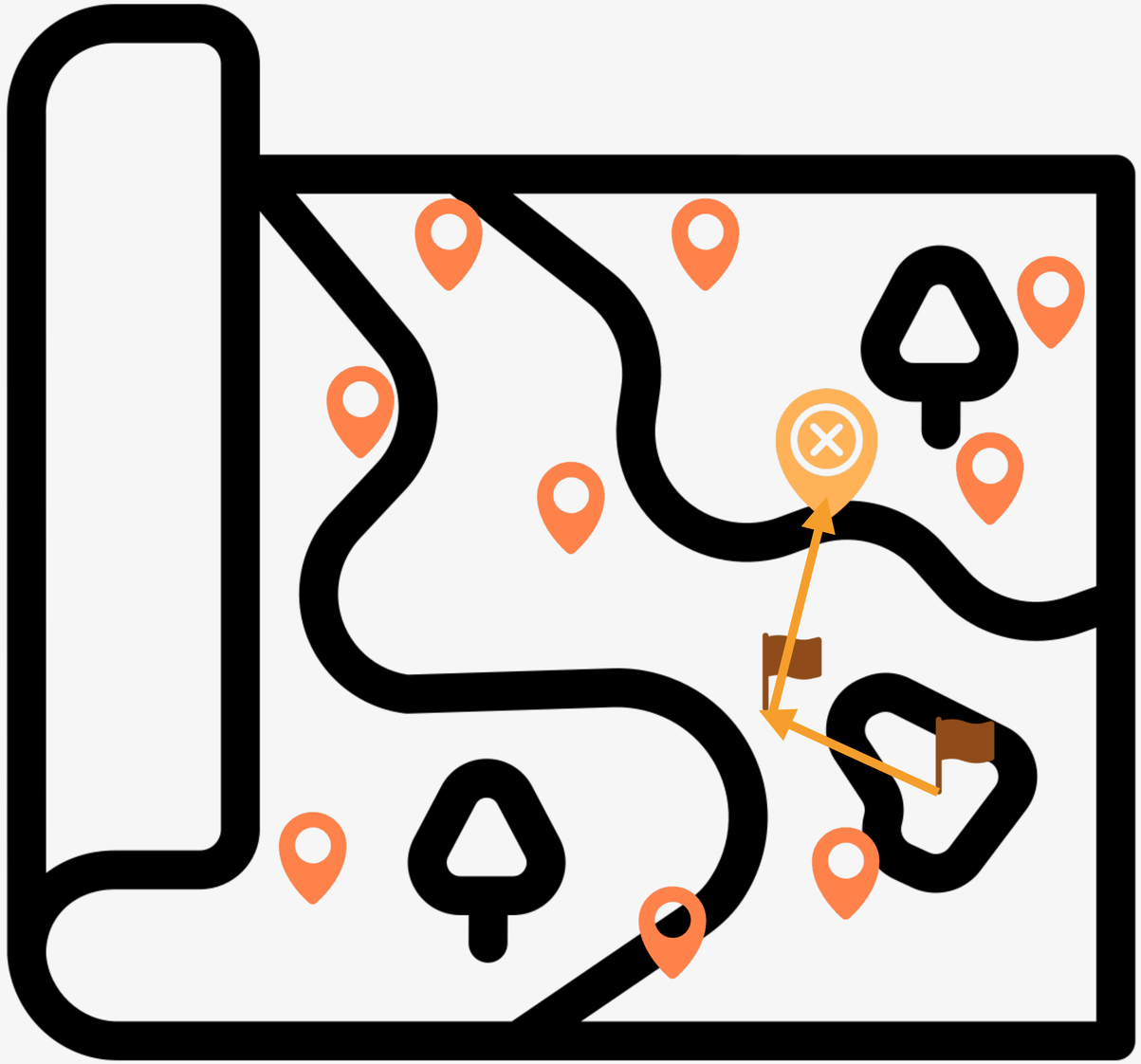


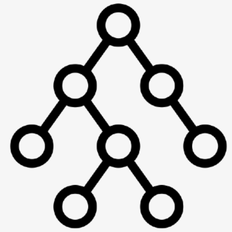
SEARCH TREE



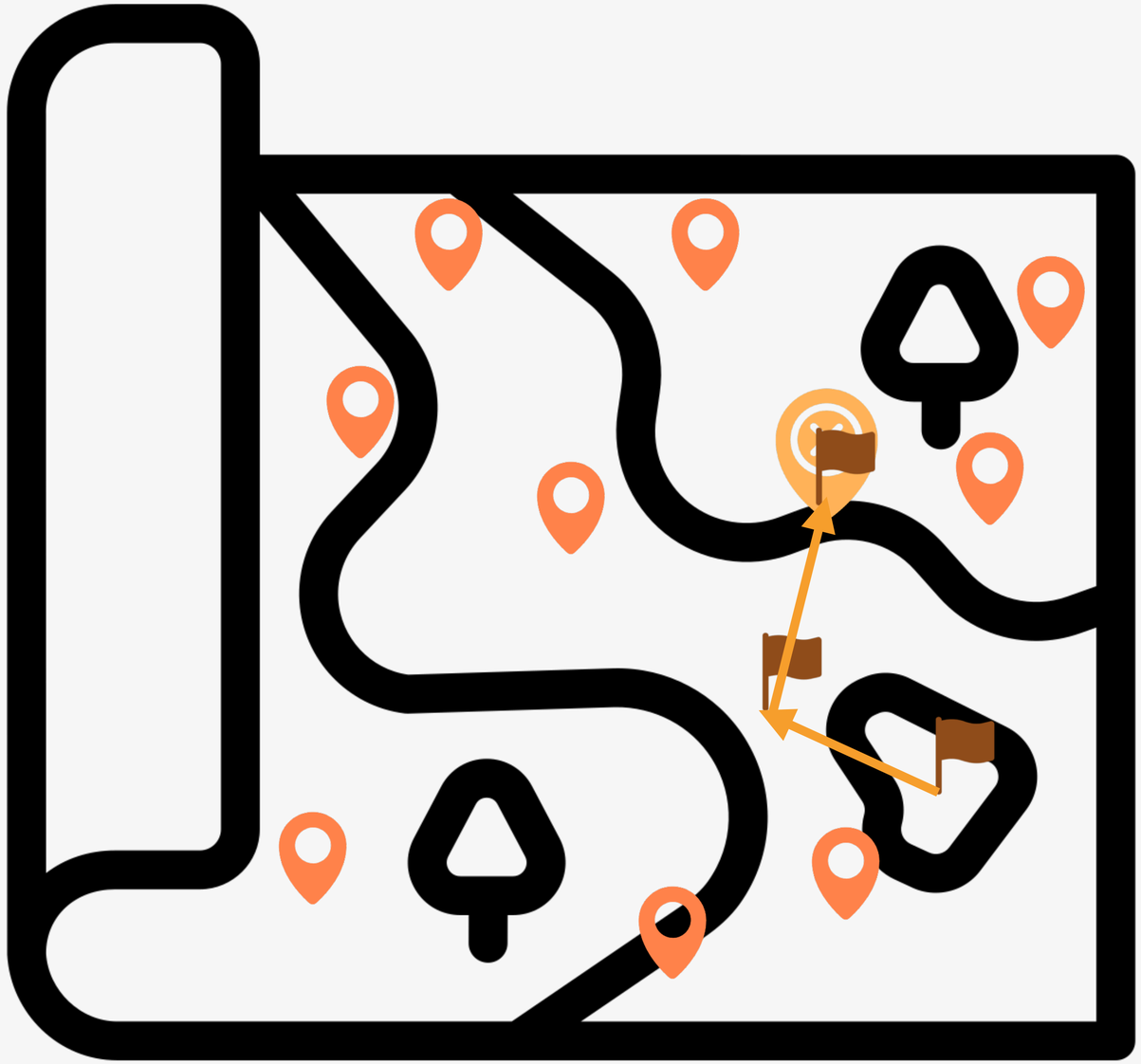


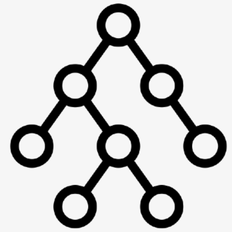
SEARCH TREE



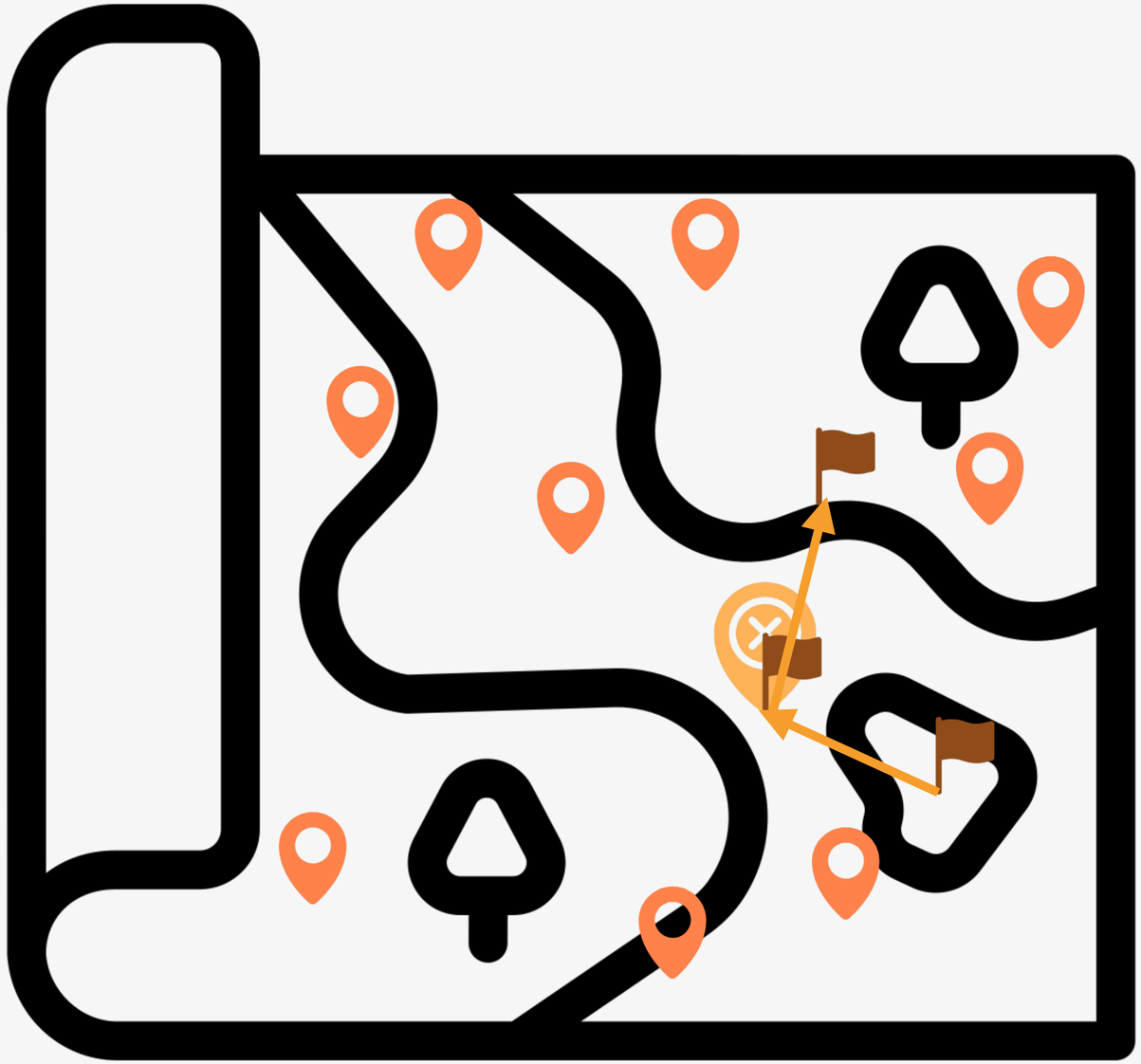


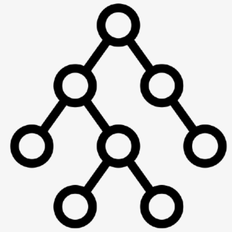
SEARCH TREE



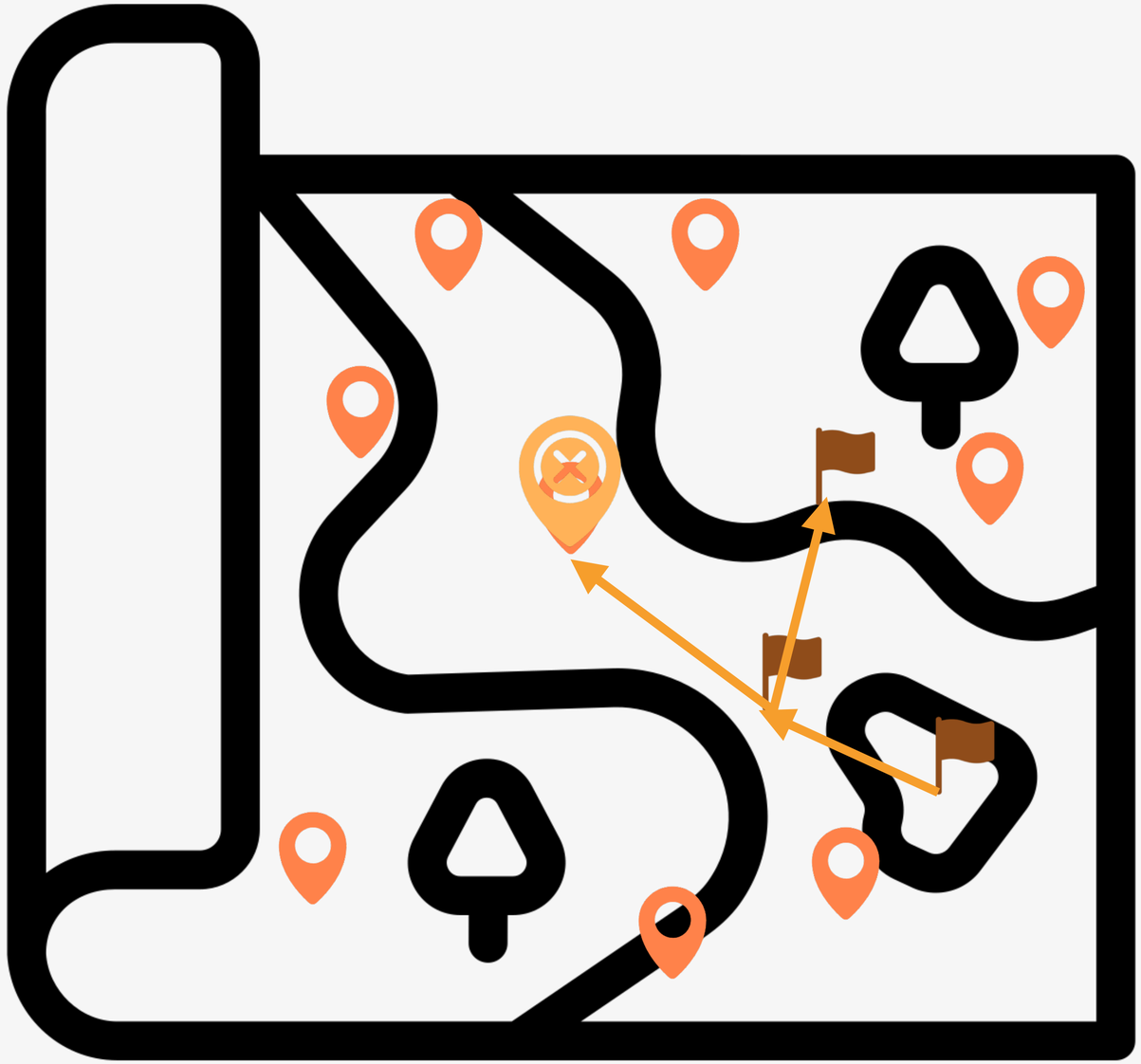


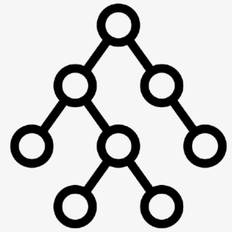
SEARCH TREE



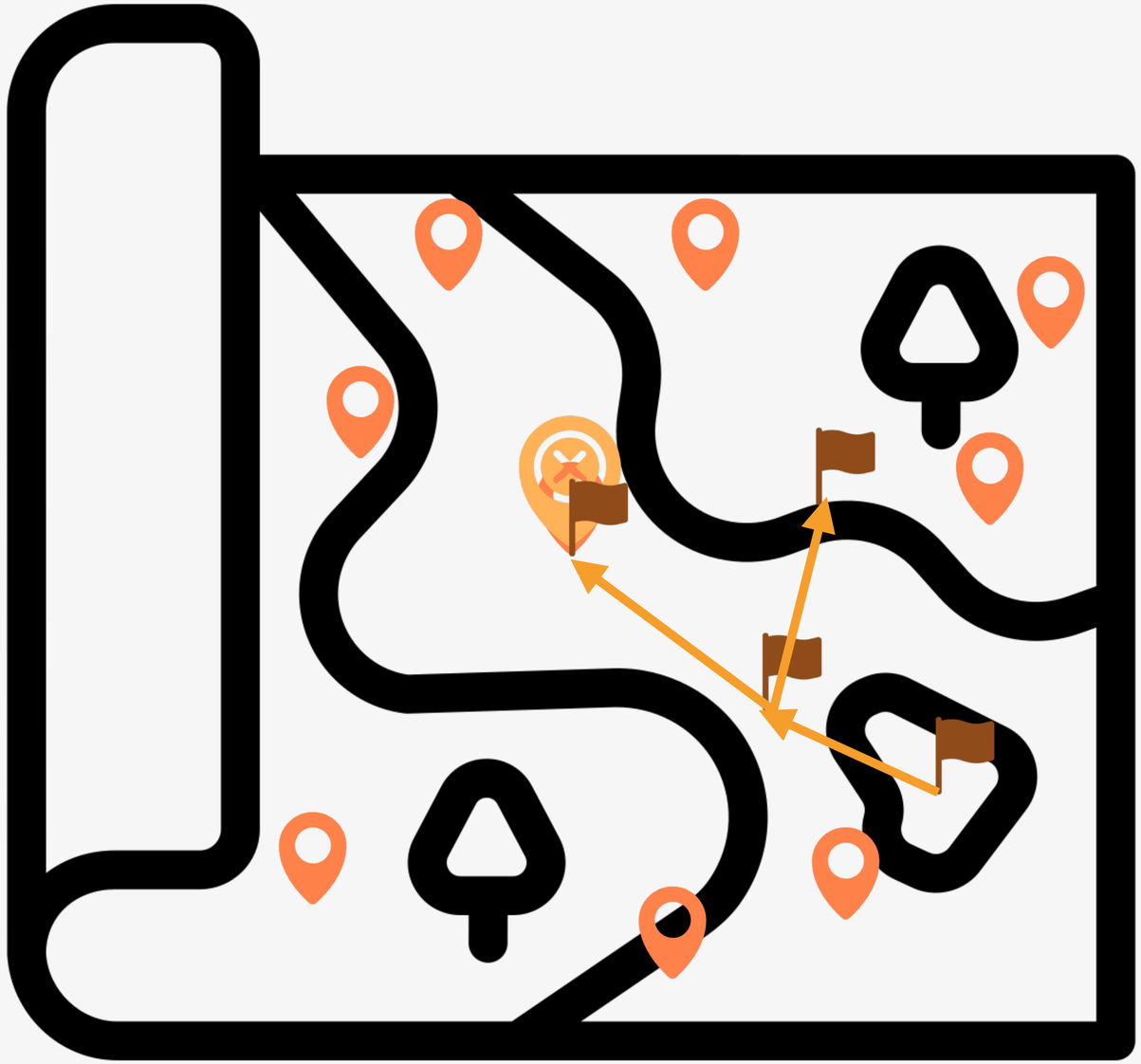


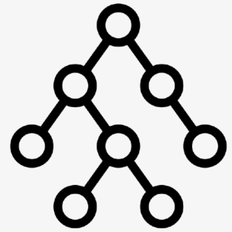
SEARCH TREE



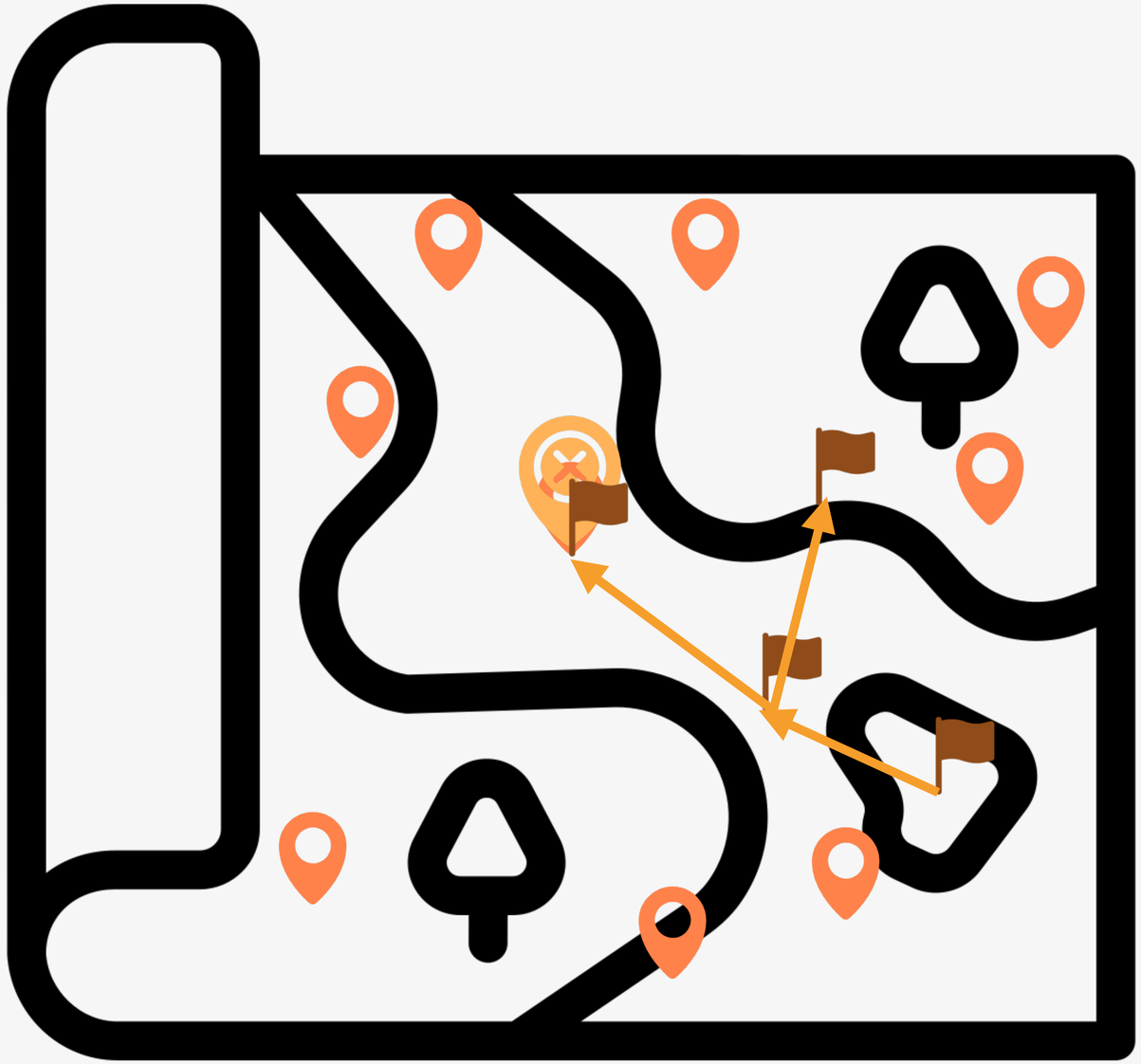


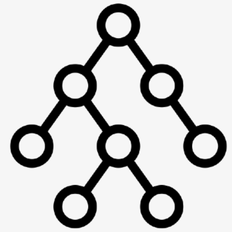
SEARCH TREE





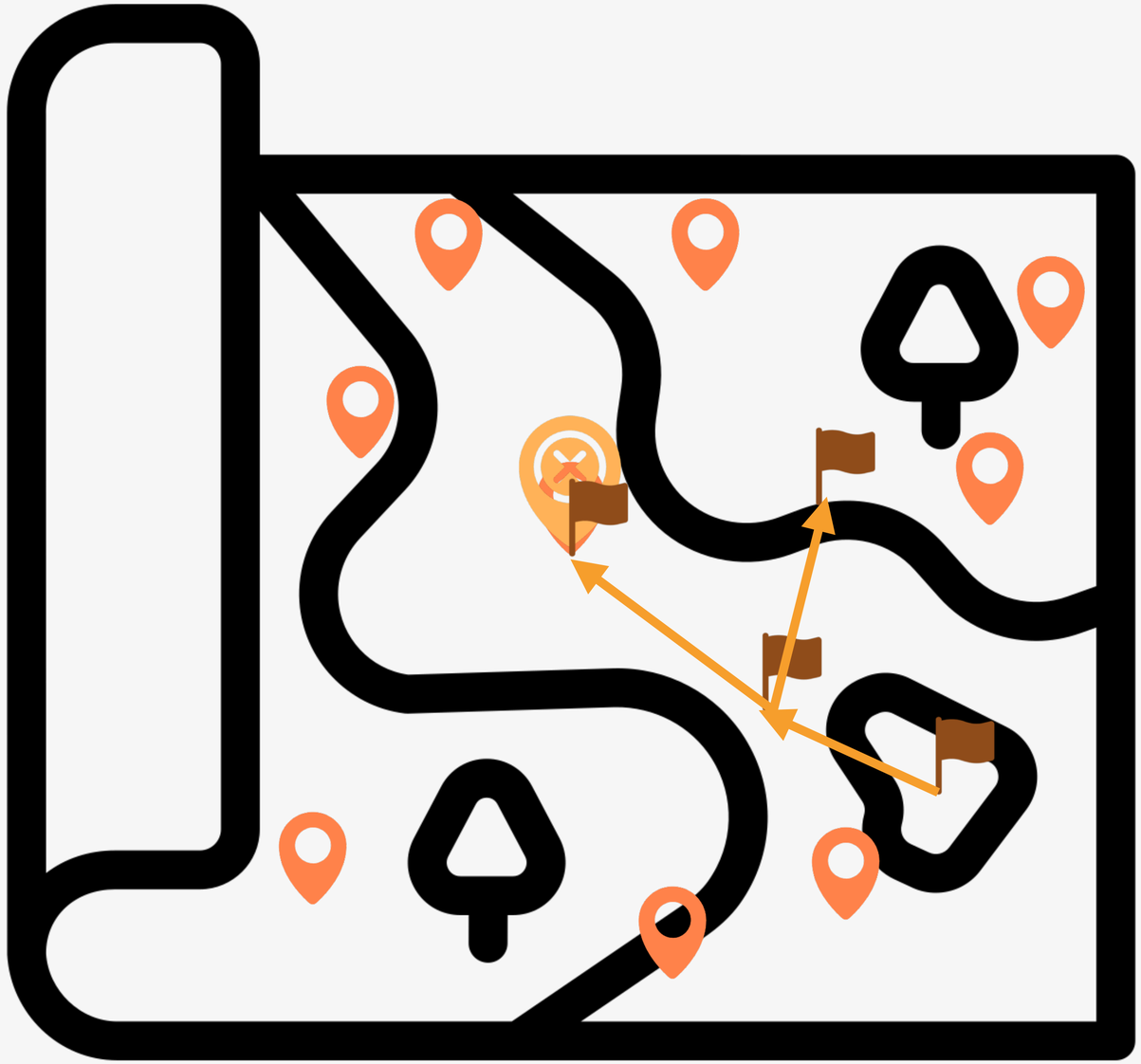
SEARCH TREE

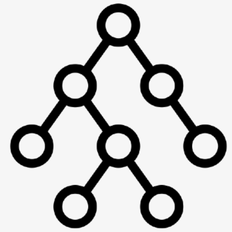




Goal of the search tree:

SEARCH TREE

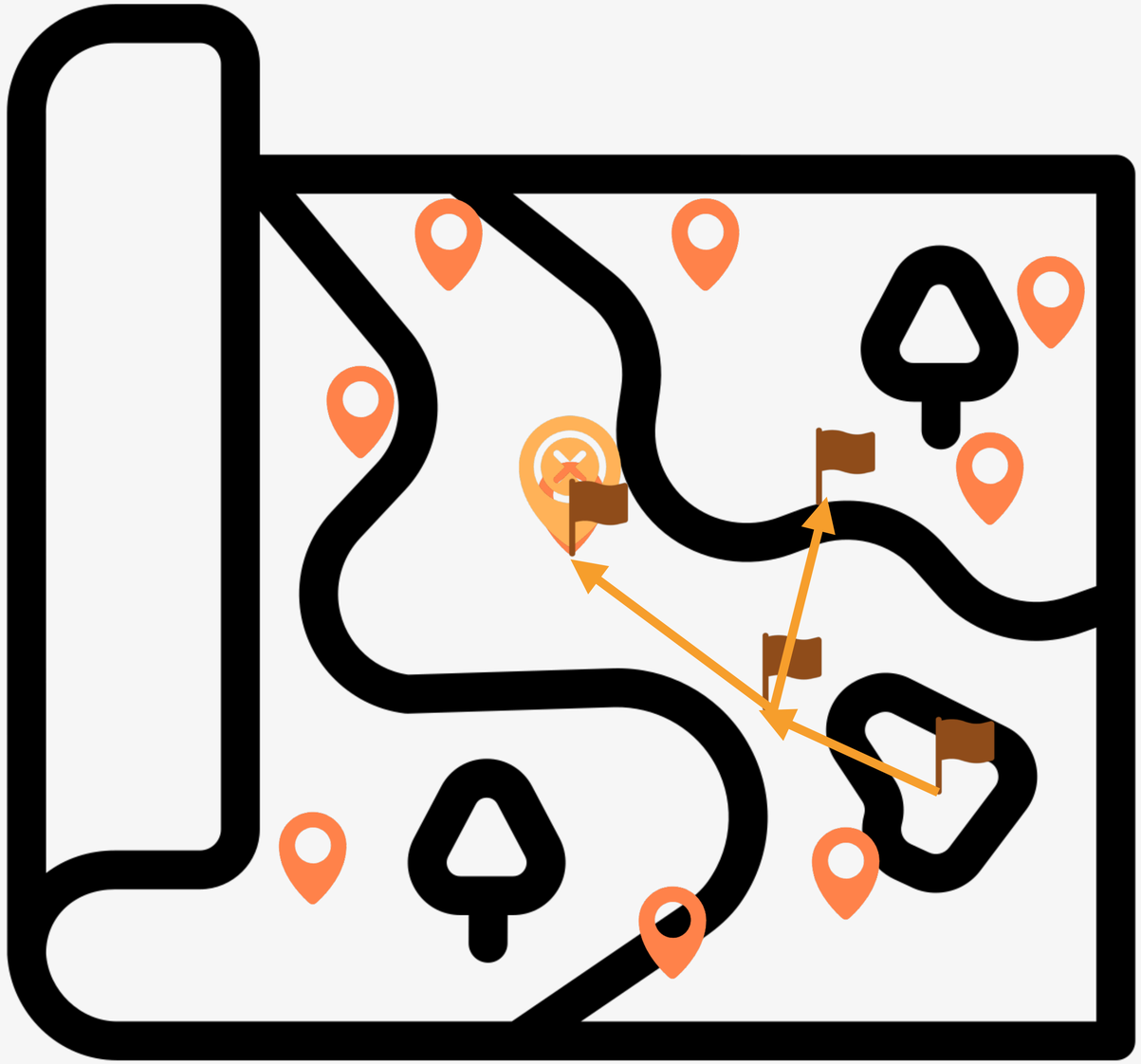


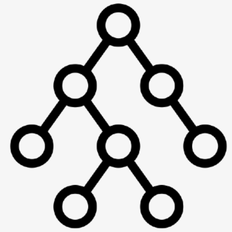


Goal of the search tree:

- **Follow the search at each step**

SEARCH TREE

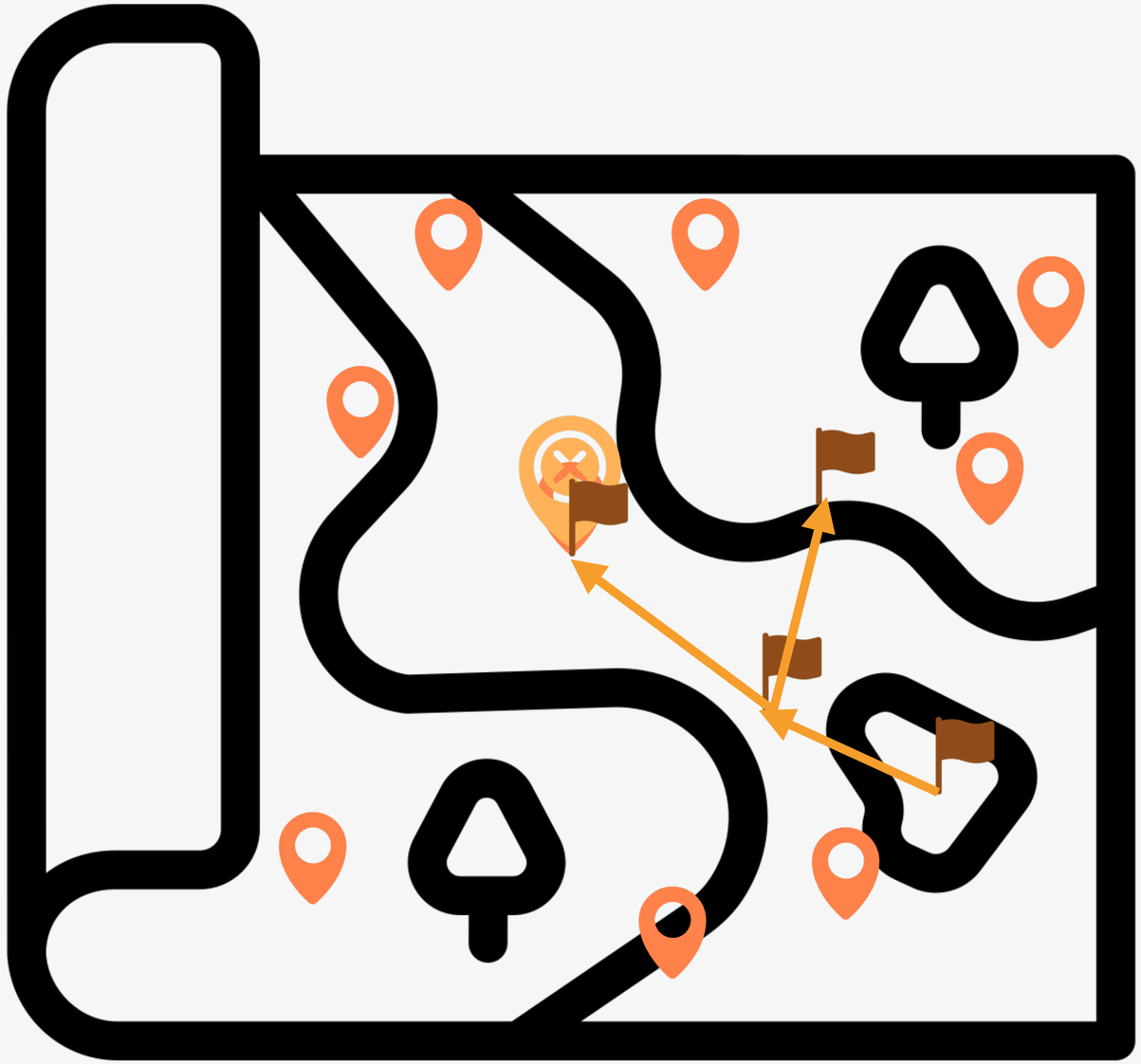


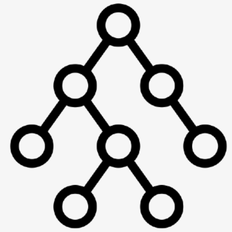


Goal of the search tree:

- **Follow the search at each step**
- **Call the fix-point after each step**

SEARCH TREE

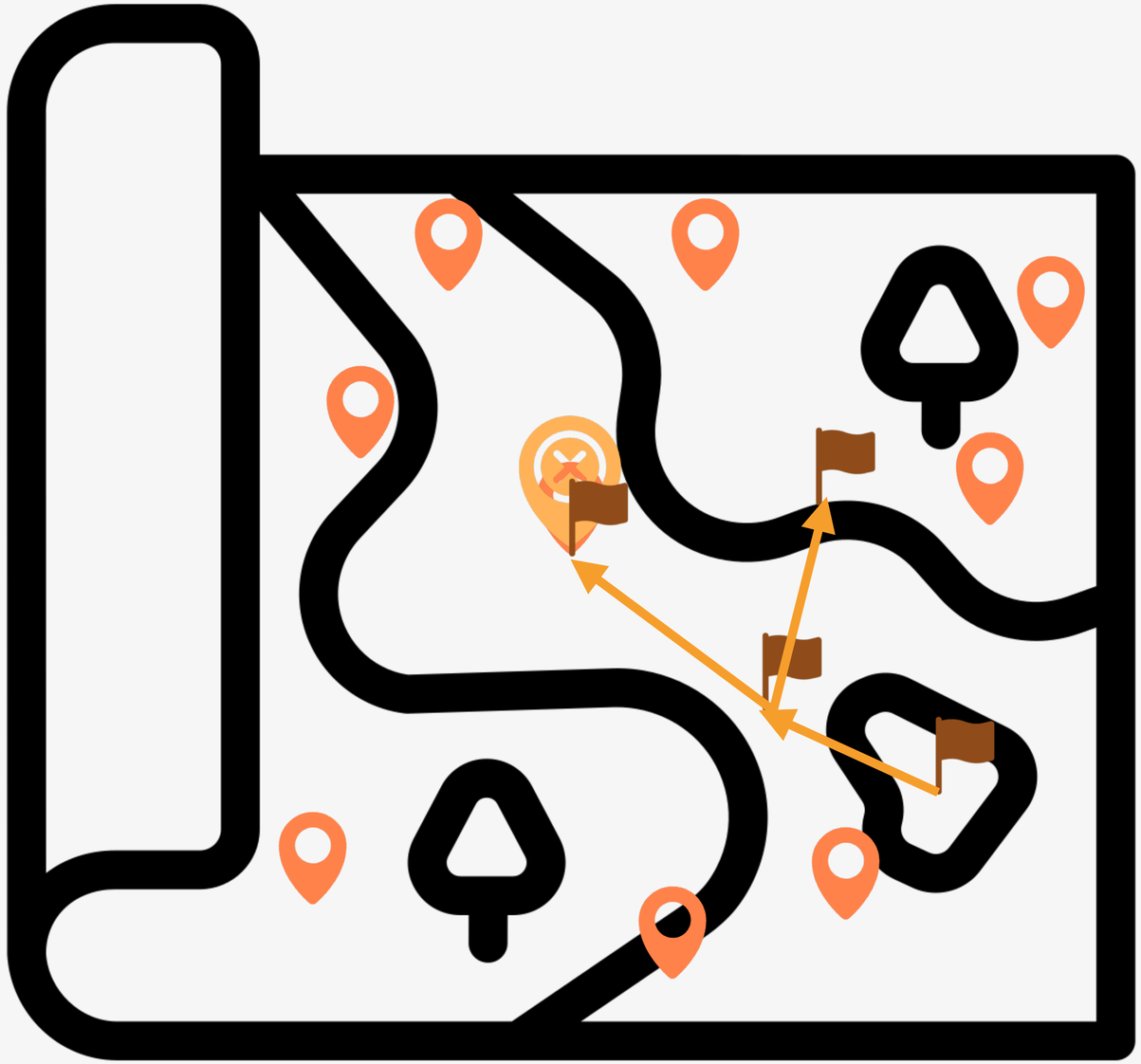


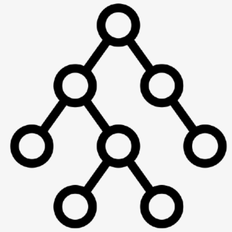


Goal of the search tree:

- Follow the search at each step
- Call the fix-point after each step
- Put the save point in order to go back

SEARCH TREE

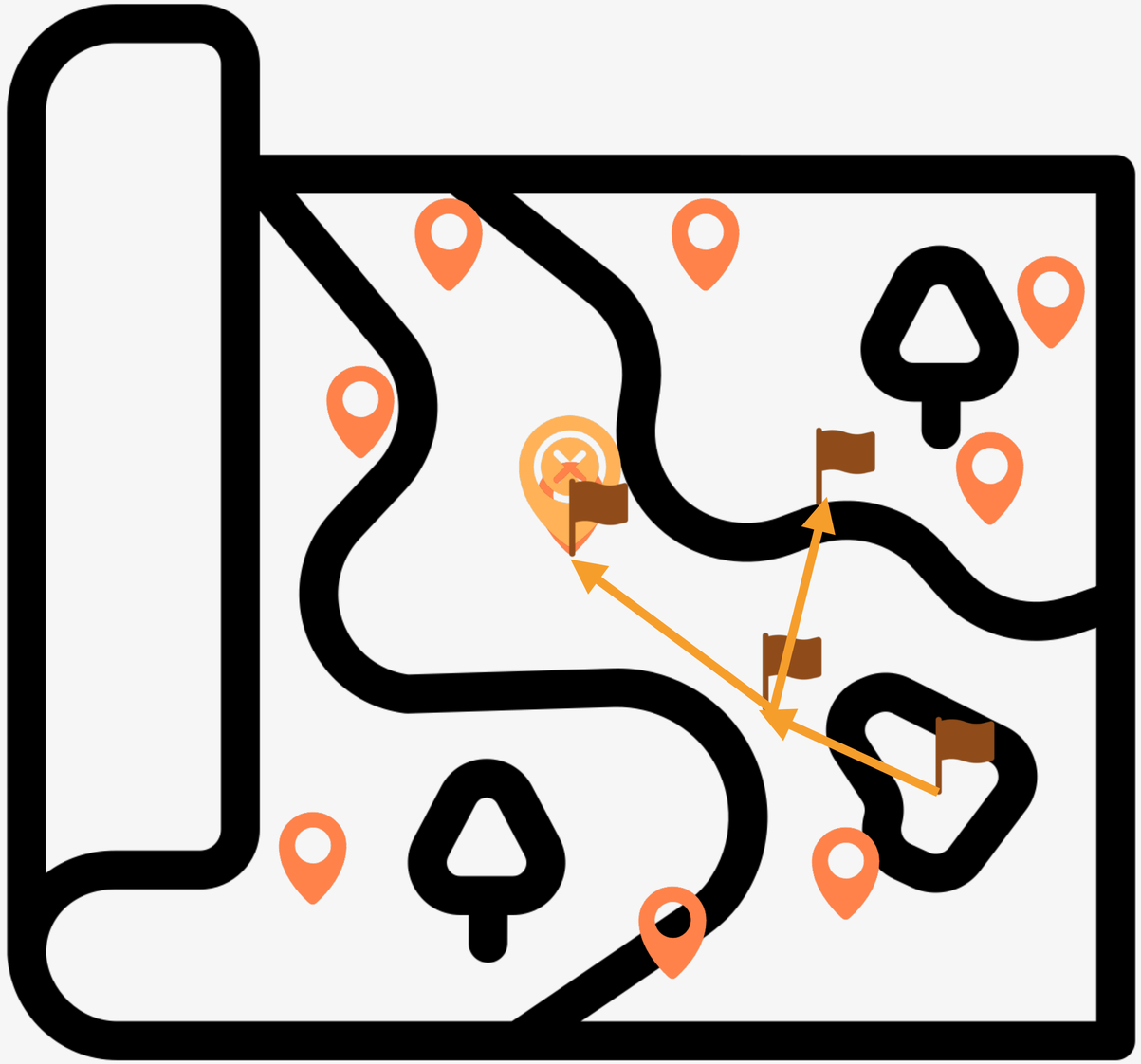


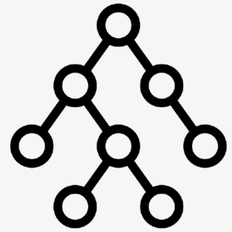


Goal of the search tree:

- Follow the search at each step
- Call the fix-point after each step
- Put the save point in order to go back
- Not explore twice the same places

SEARCH TREE





Goal of the search tree:

- Follow the search at each step
- Call the fix-point after each step
- Put the save point in order to go back
- Not explore twice the same places
- Explore until goal is reached (i.e., a solution, all solutions, optimal solution)

Exemple: Solving of a Minimisation ILP With Branch and Bound

Exemple: Solving of a Minimisation ILP With Branch and Bound

UB: $+\infty$

Exemple: Solving of a Minimisation ILP With Branch and Bound

UB:+∞

LB:3.5

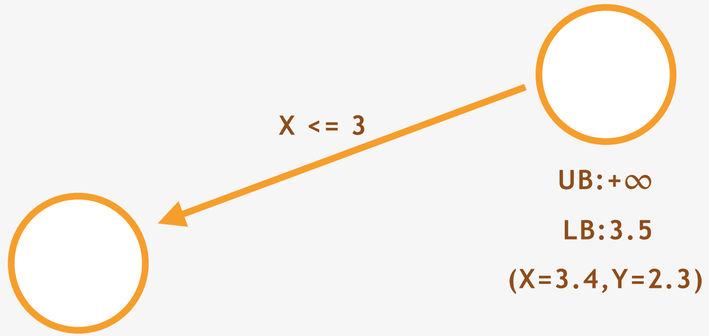
Exemple: Solving of a Minimisation ILP With Branch and Bound

UB:+∞

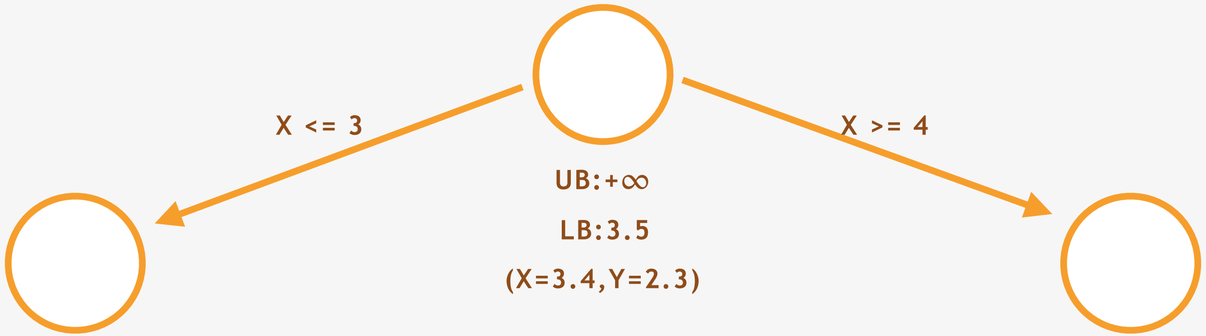
LB:3.5

(X=3.4,Y=2.3)

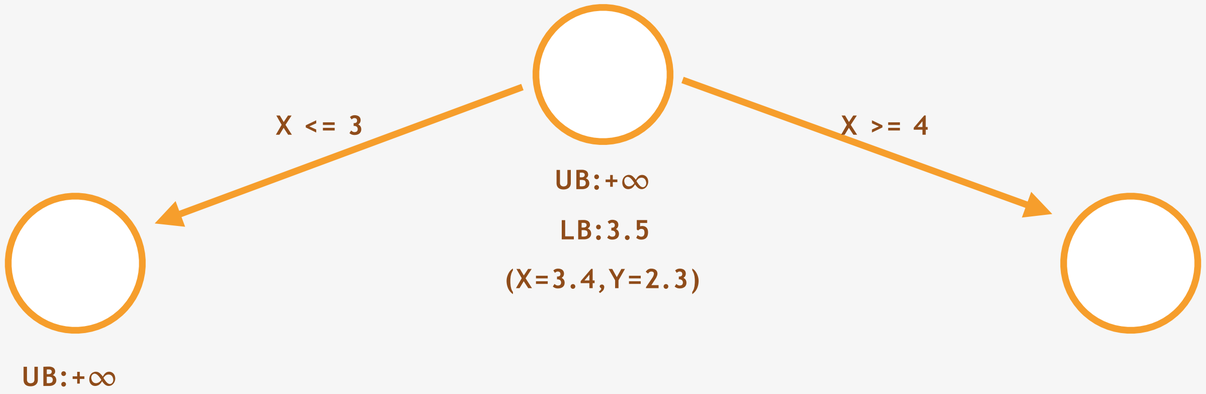
Exemple: Solving of a Minimisation ILP With Branch and Bound



Exemple: Solving of a Minimisation ILP With Branch and Bound

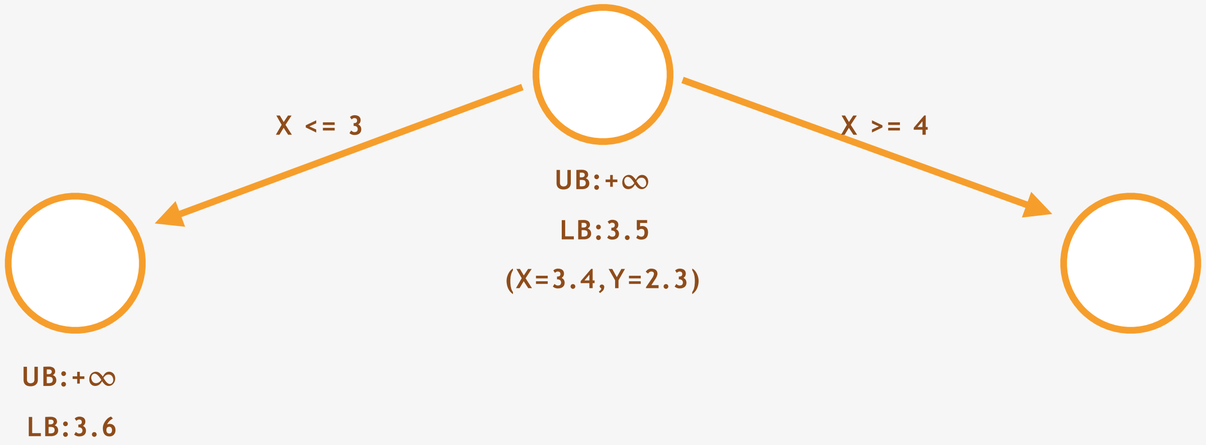


Exemple: Solving of a Minimisation ILP With Branch and Bound

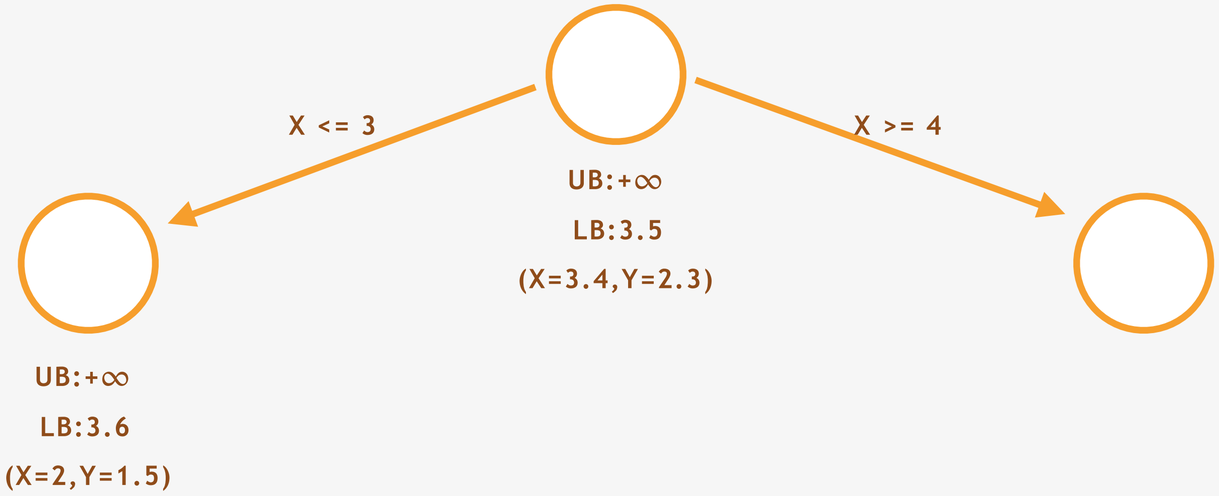


COUSIN OF THE BRANCH-AND-BOUND

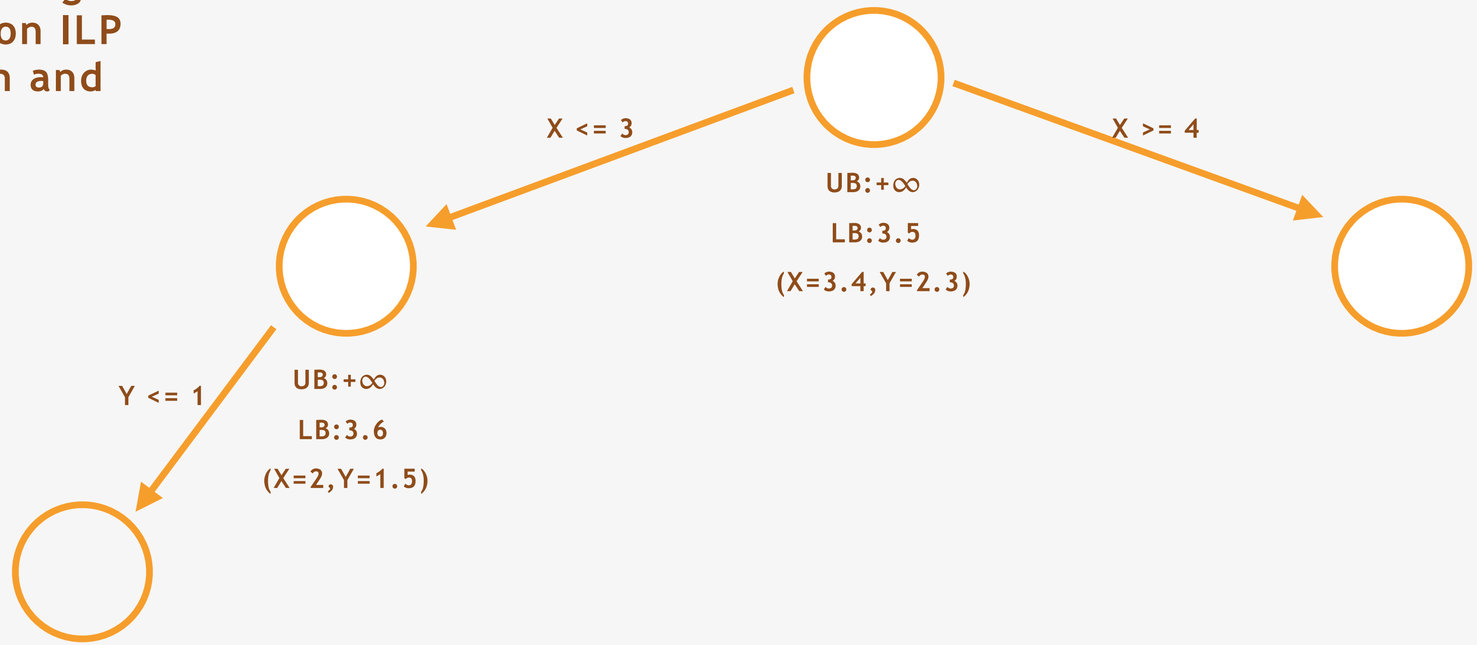
Exemple: Solving of a Minimisation ILP With Branch and Bound



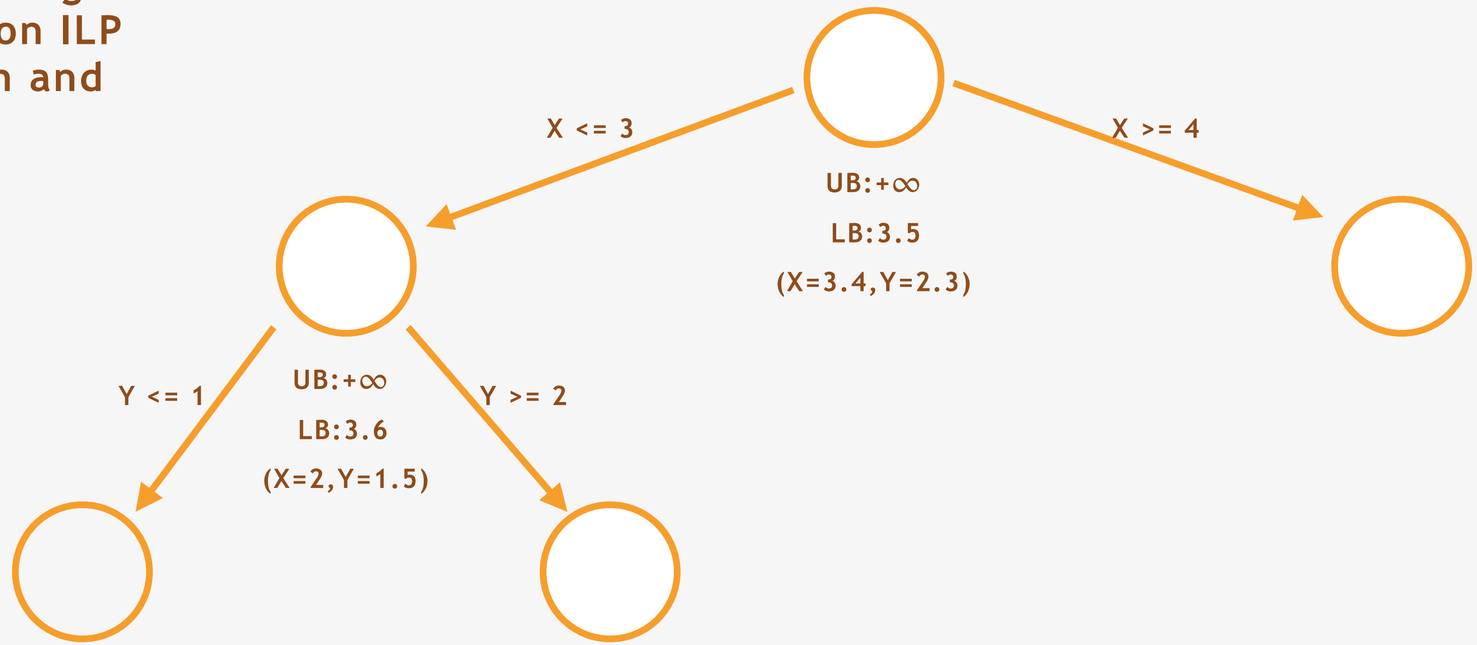
Exemple: Solving of a Minimisation ILP With Branch and Bound



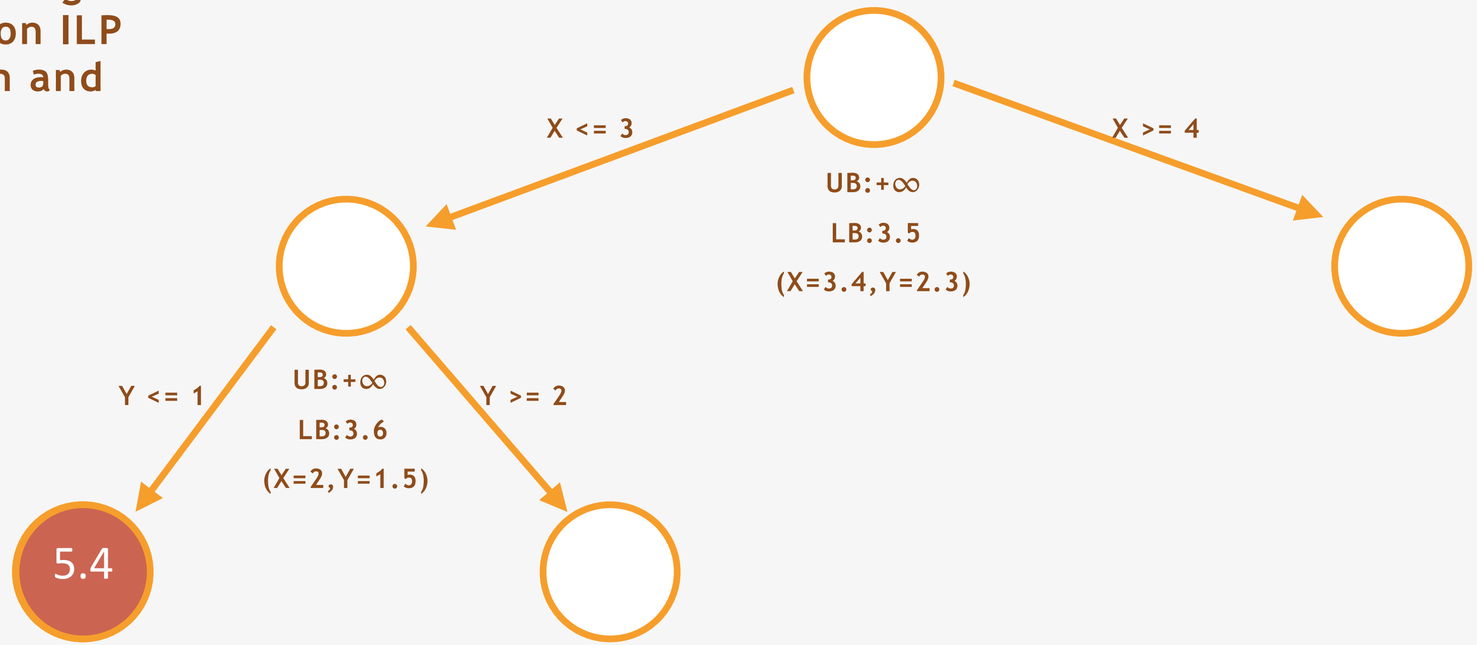
Exemple: Solving of a Minimisation ILP With Branch and Bound



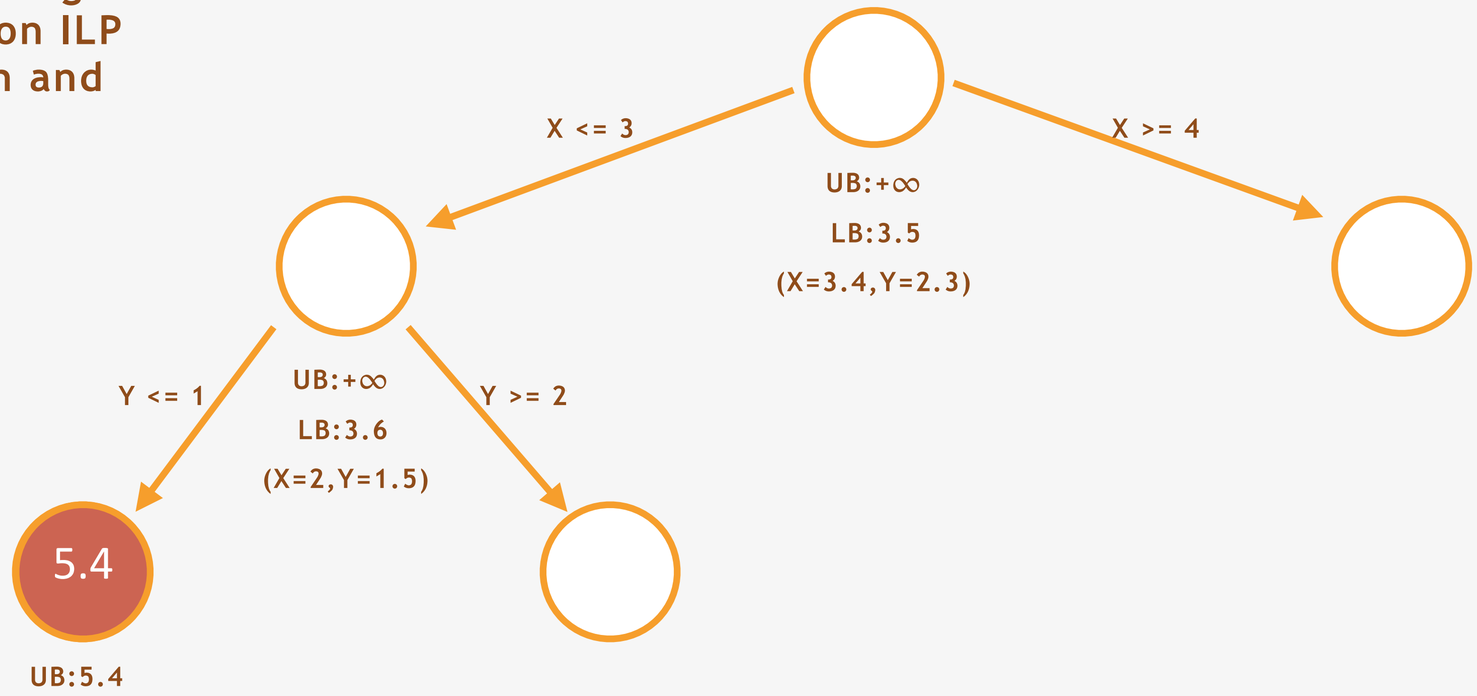
Exemple: Solving of a Minimisation ILP With Branch and Bound



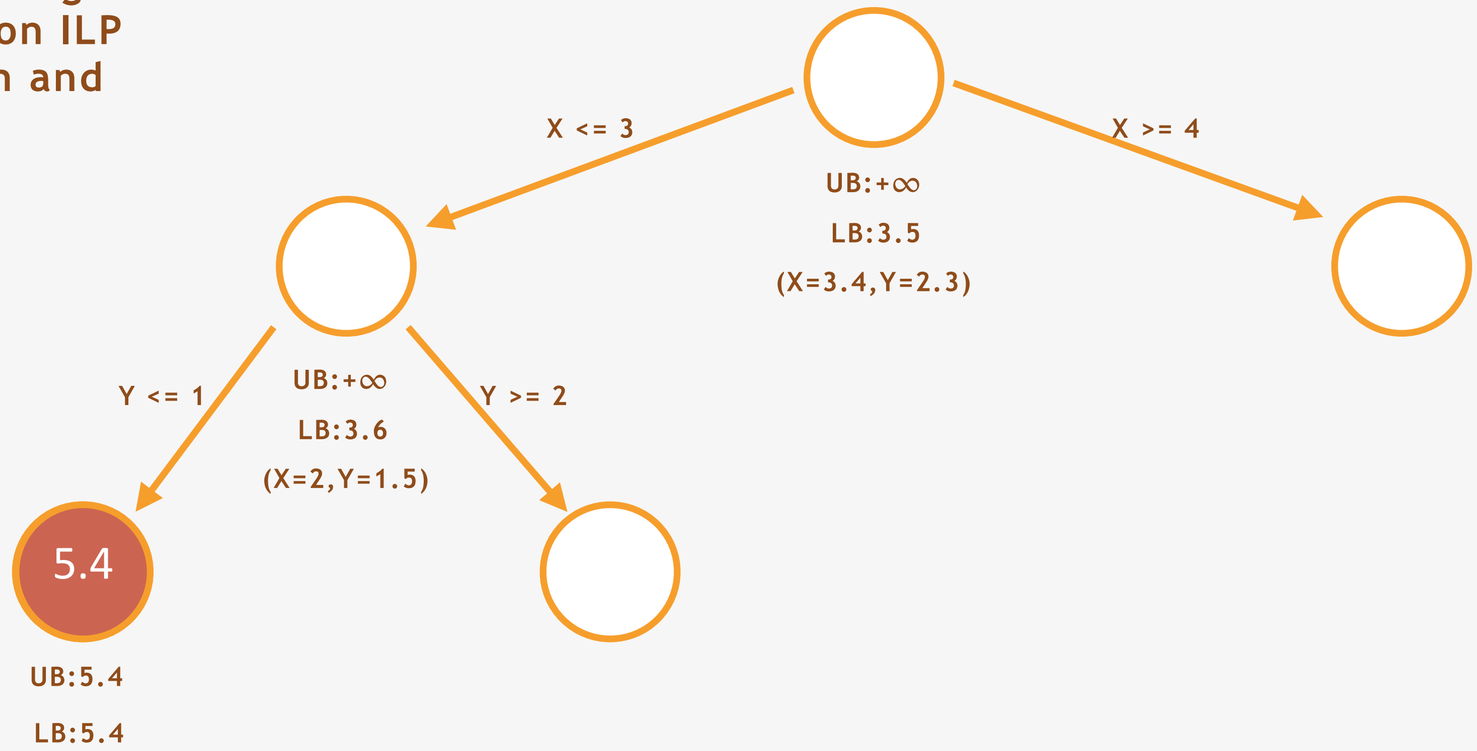
Exemple: Solving of a Minimisation ILP With Branch and Bound



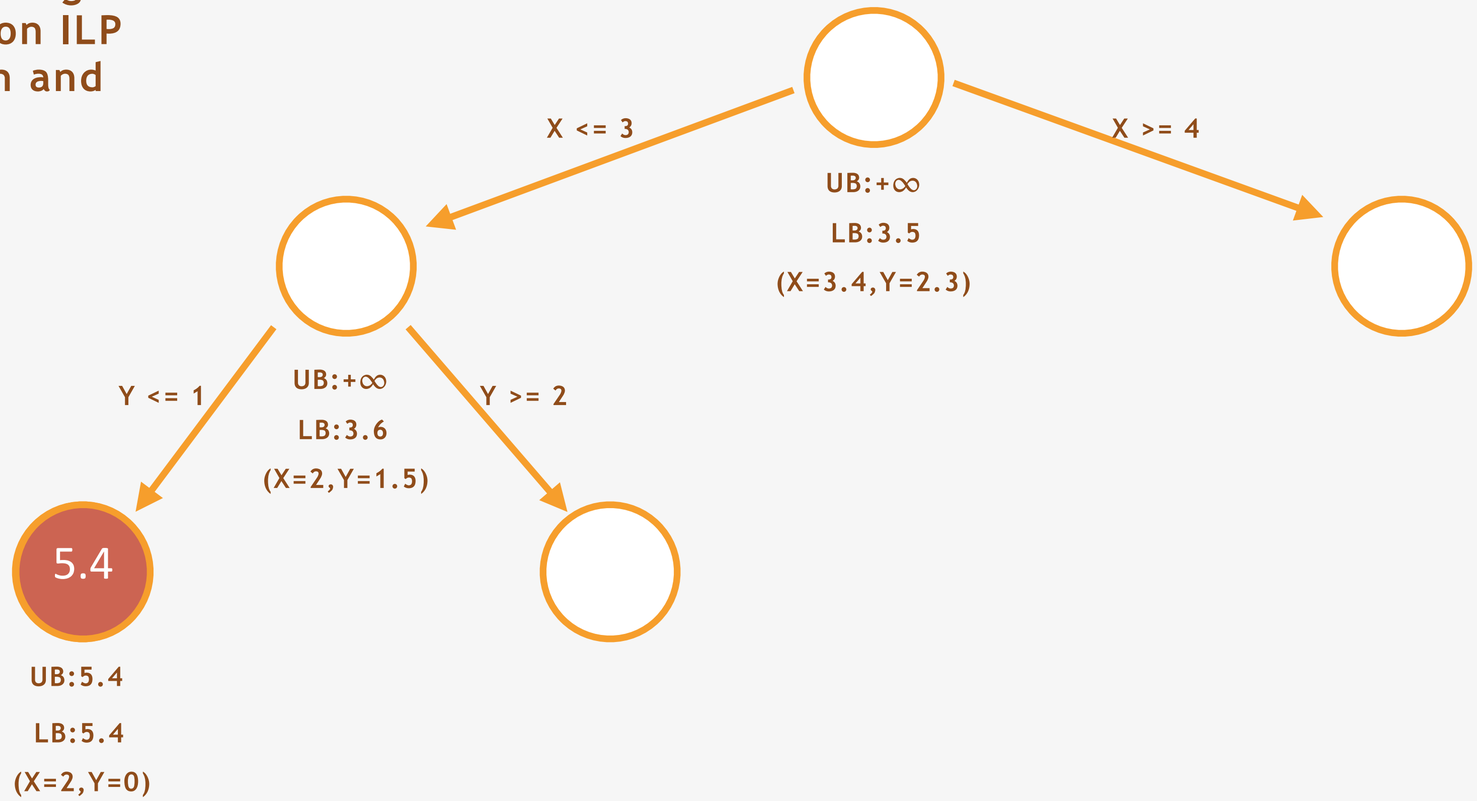
Exemple: Solving of a Minimisation ILP With Branch and Bound



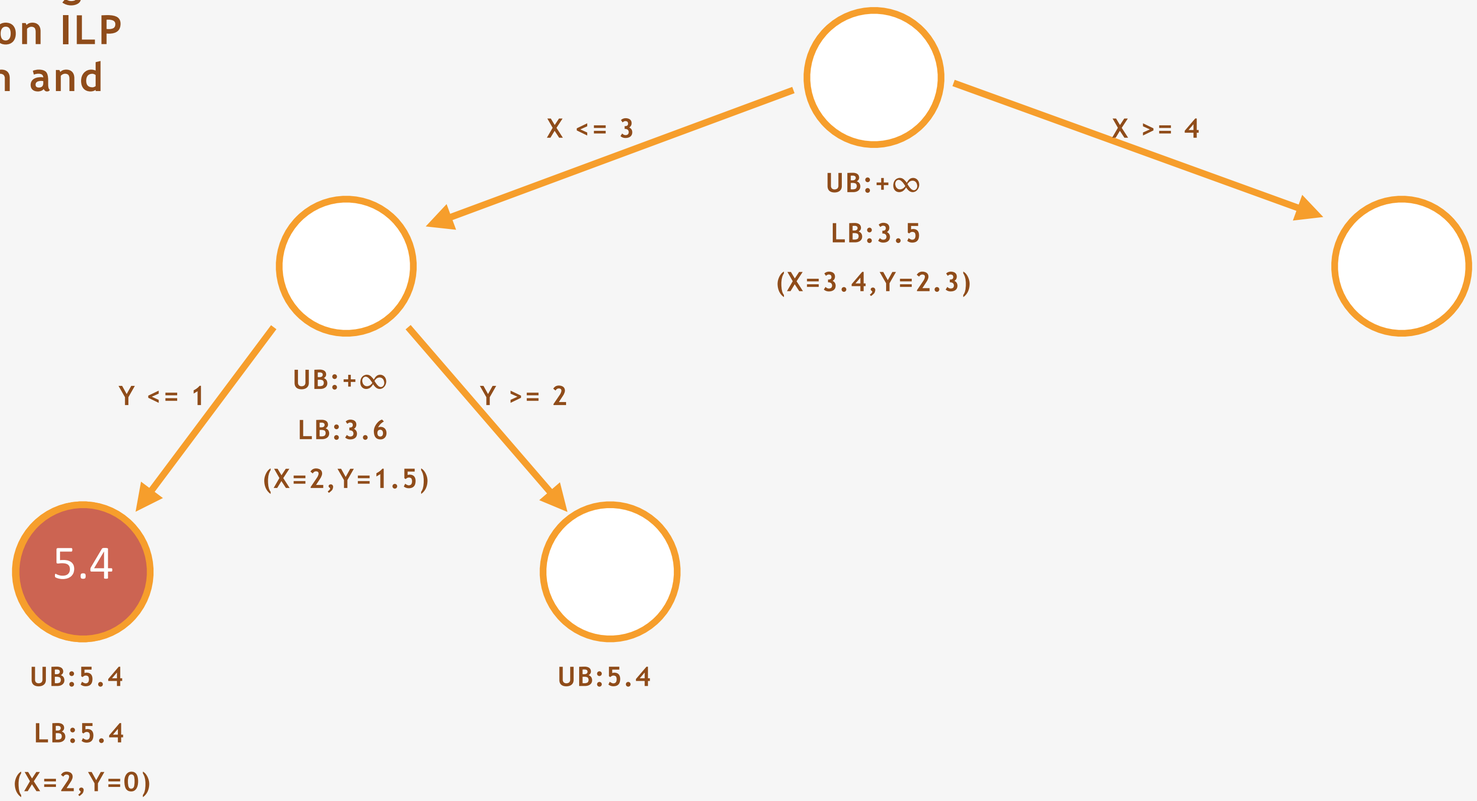
Exemple: Solving of a Minimisation ILP With Branch and Bound



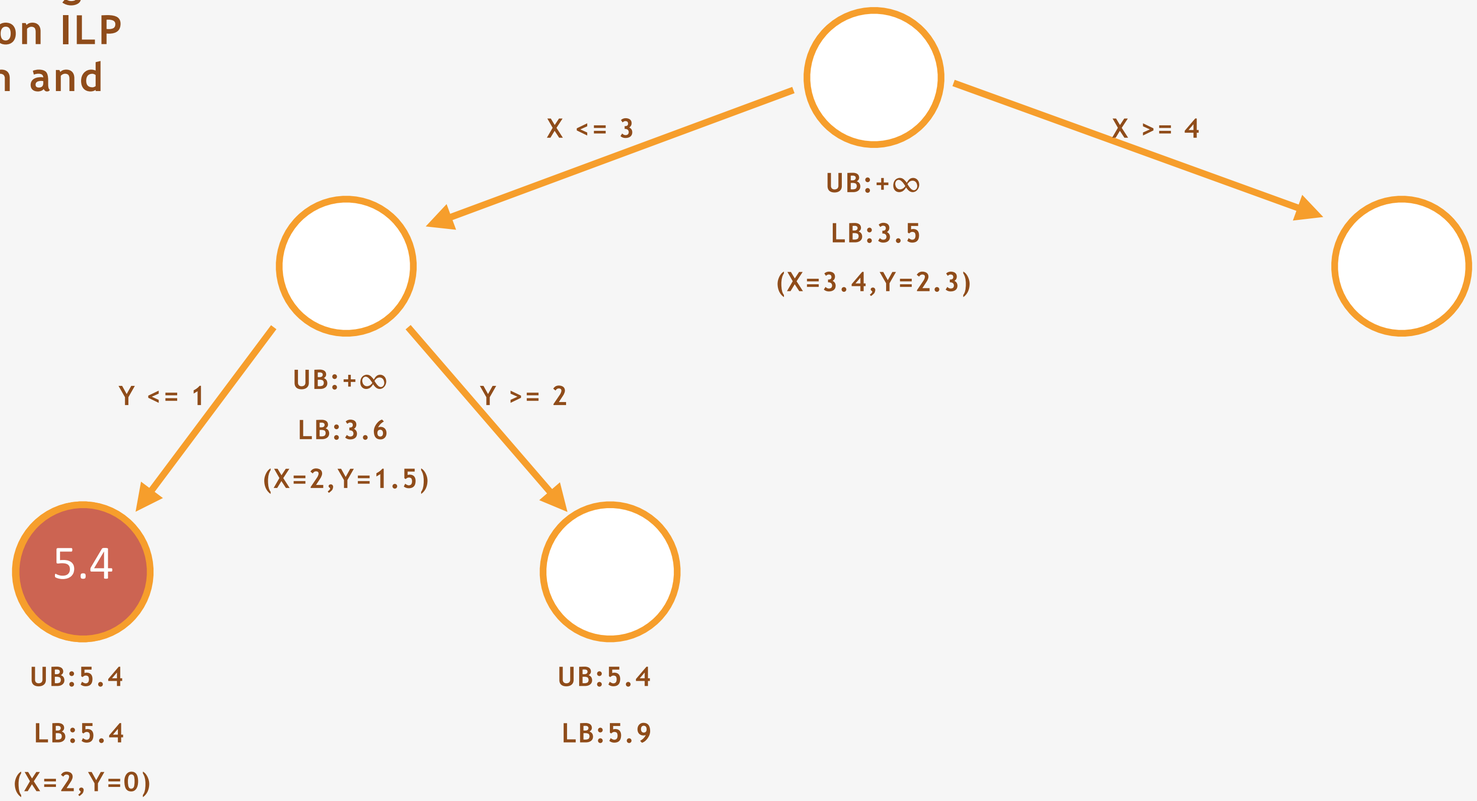
Exemple: Solving of a Minimisation ILP With Branch and Bound



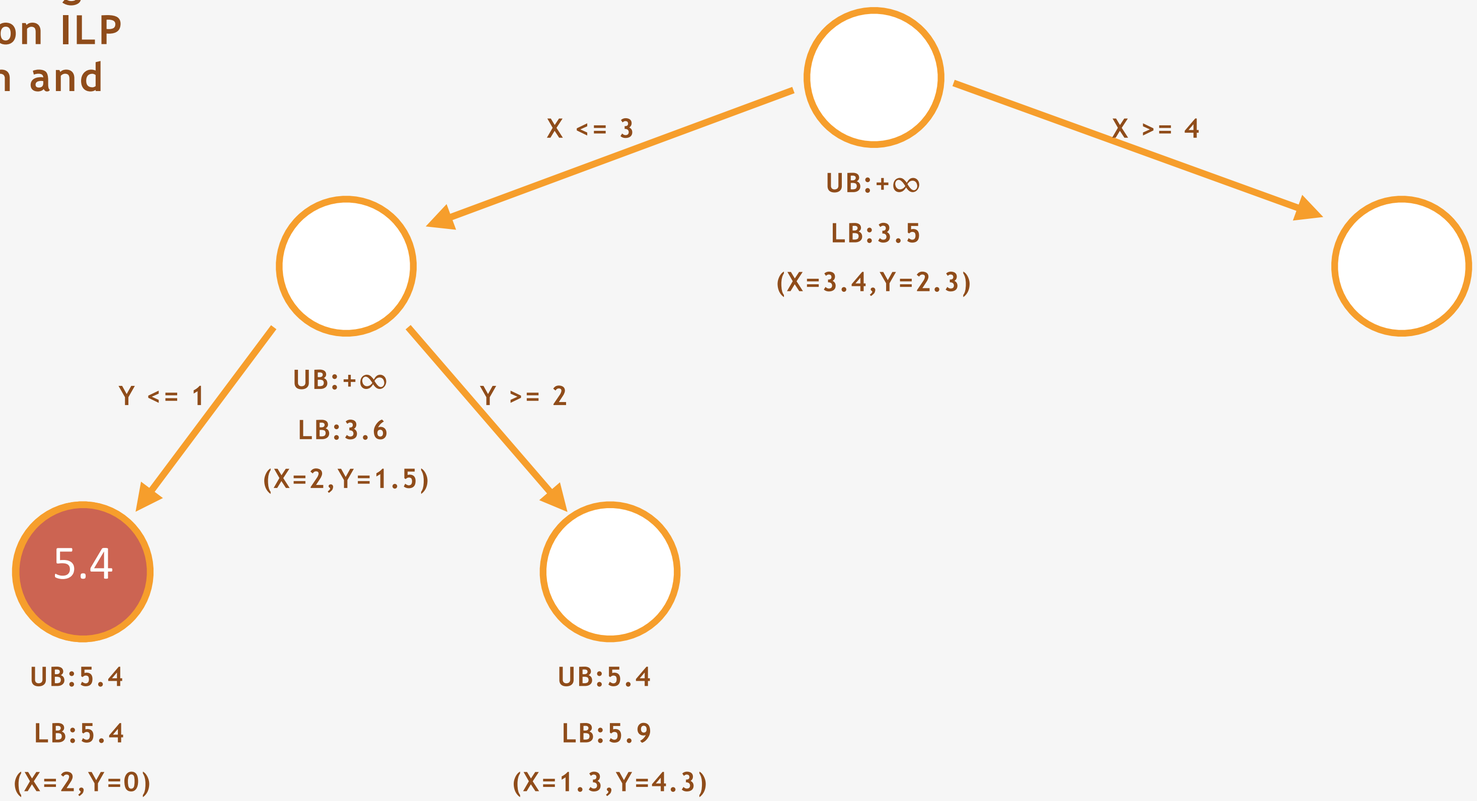
Exemple: Solving of a Minimisation ILP With Branch and Bound



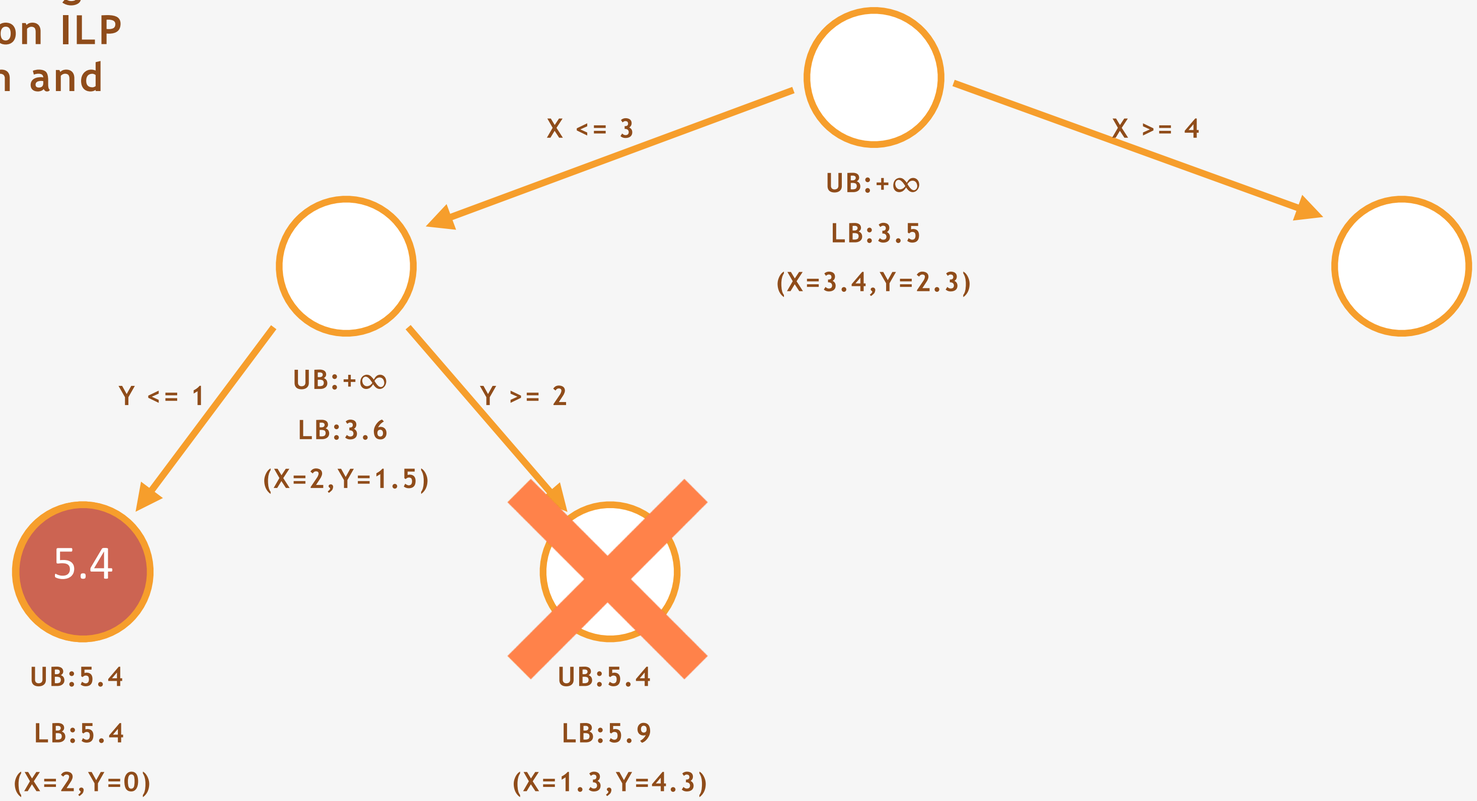
Exemple: Solving of a Minimisation ILP With Branch and Bound



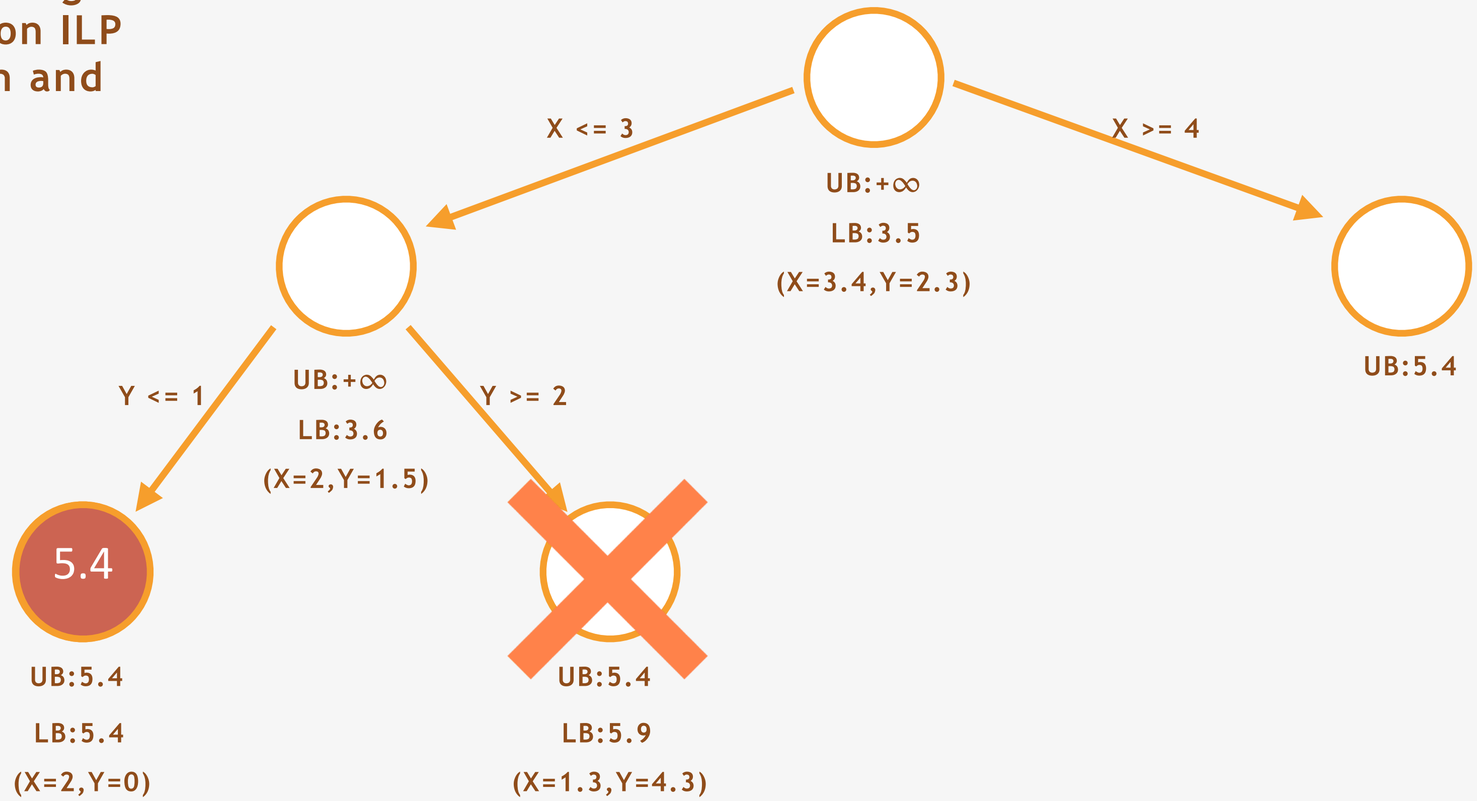
Exemple: Solving of a Minimisation ILP With Branch and Bound



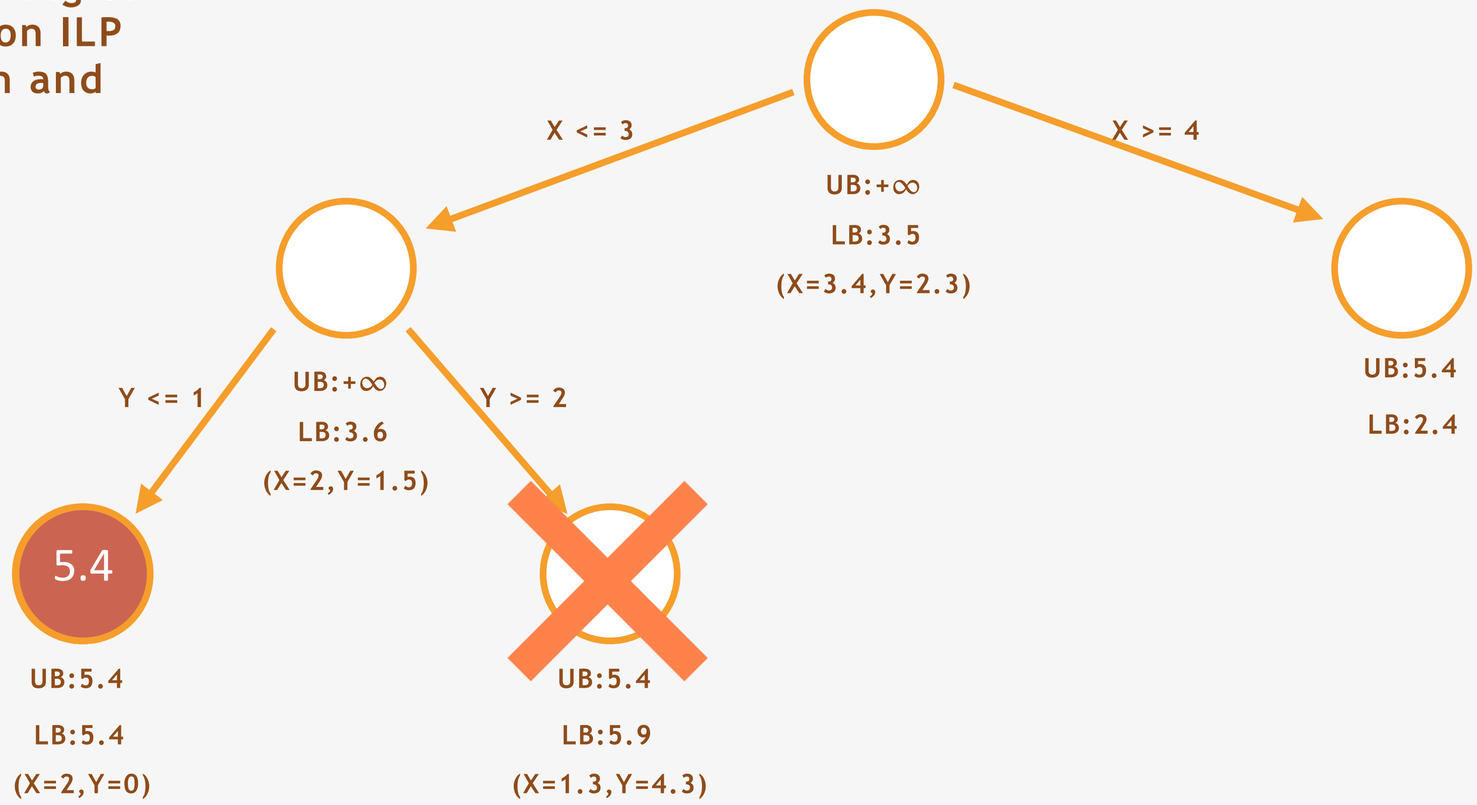
Exemple: Solving of a Minimisation ILP With Branch and Bound



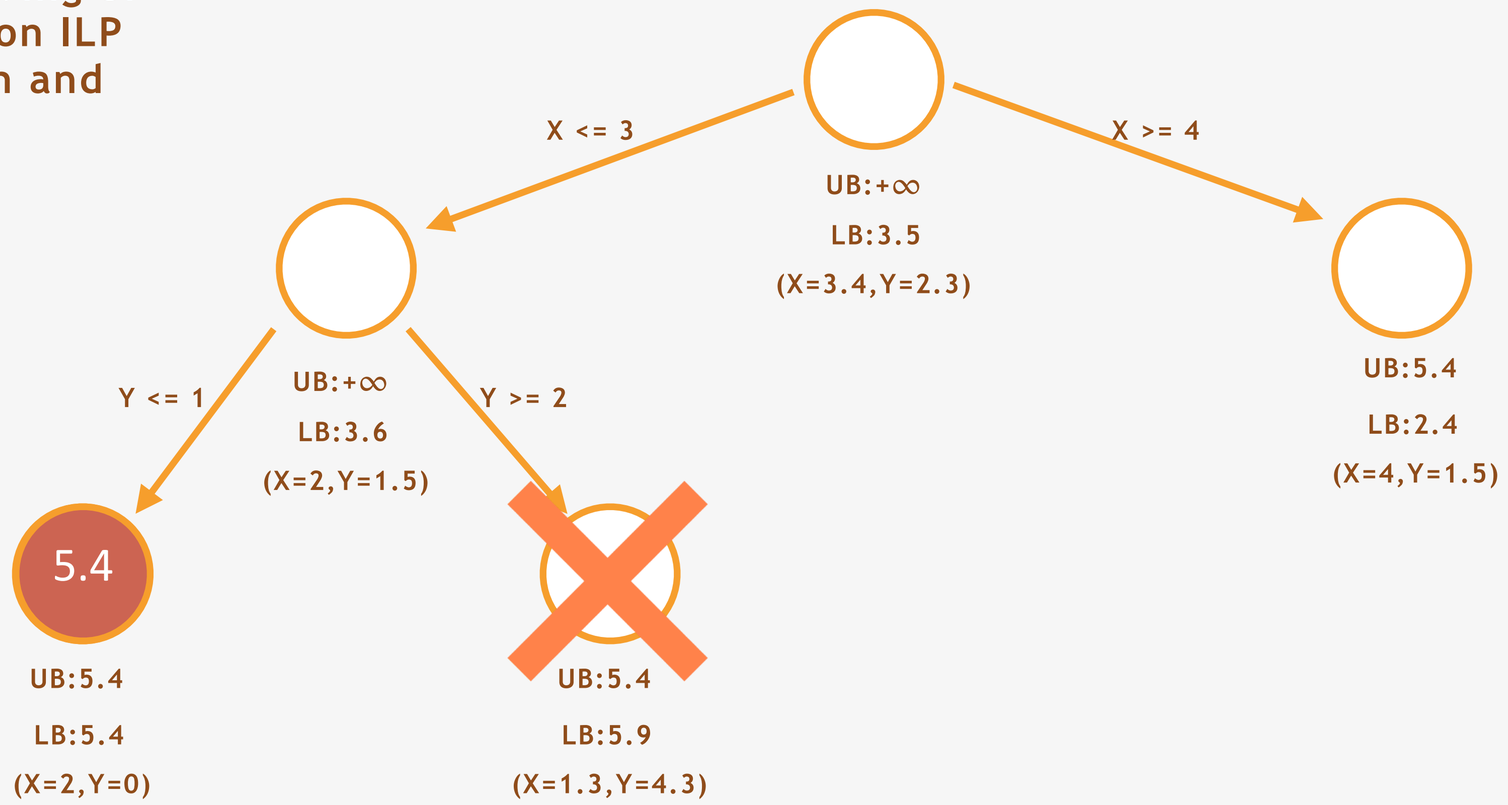
Exemple: Solving of a Minimisation ILP With Branch and Bound



Exemple: Solving of a Minimisation ILP With Branch and Bound

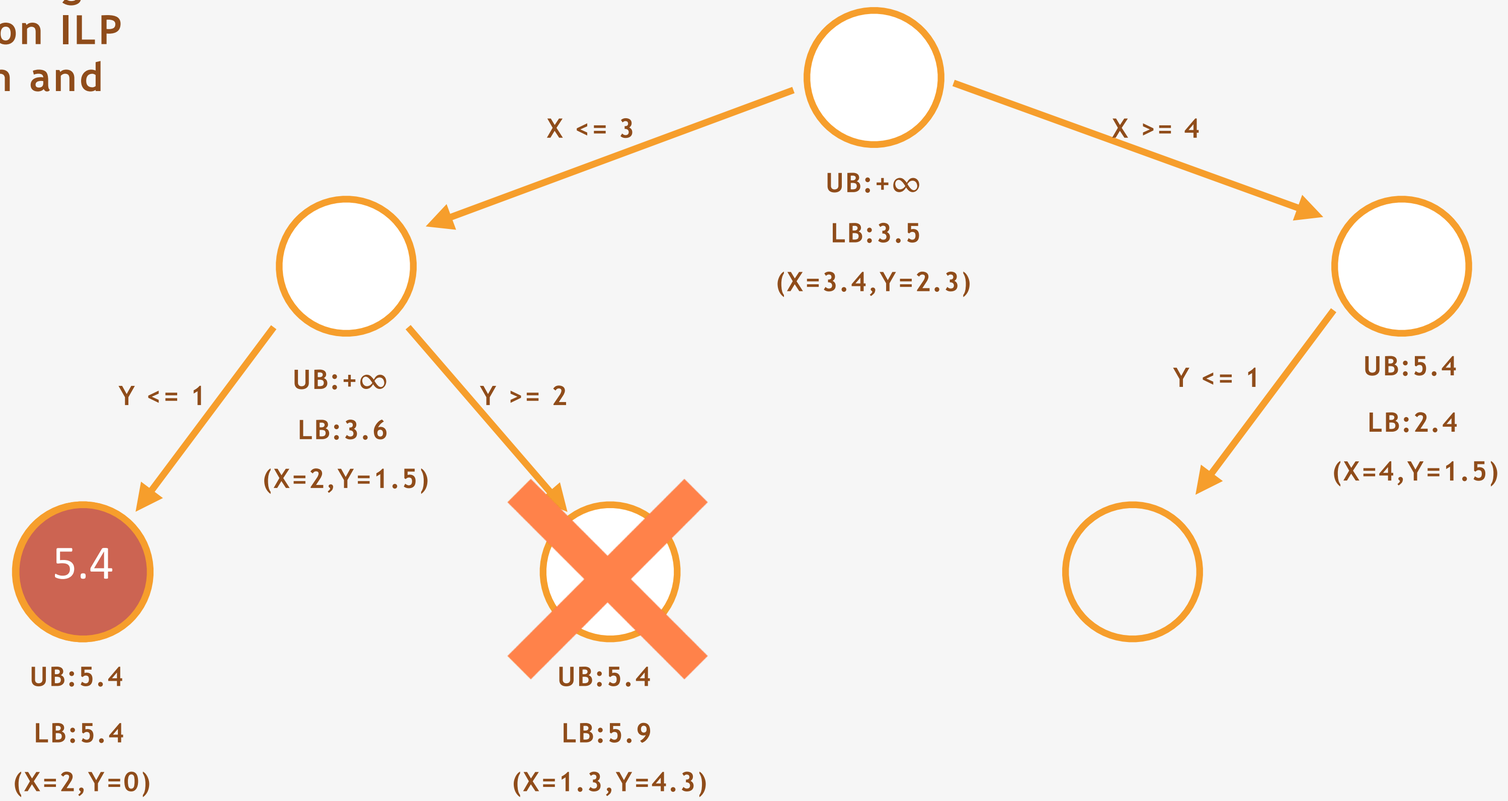


Exemple: Solving of a Minimisation ILP With Branch and Bound



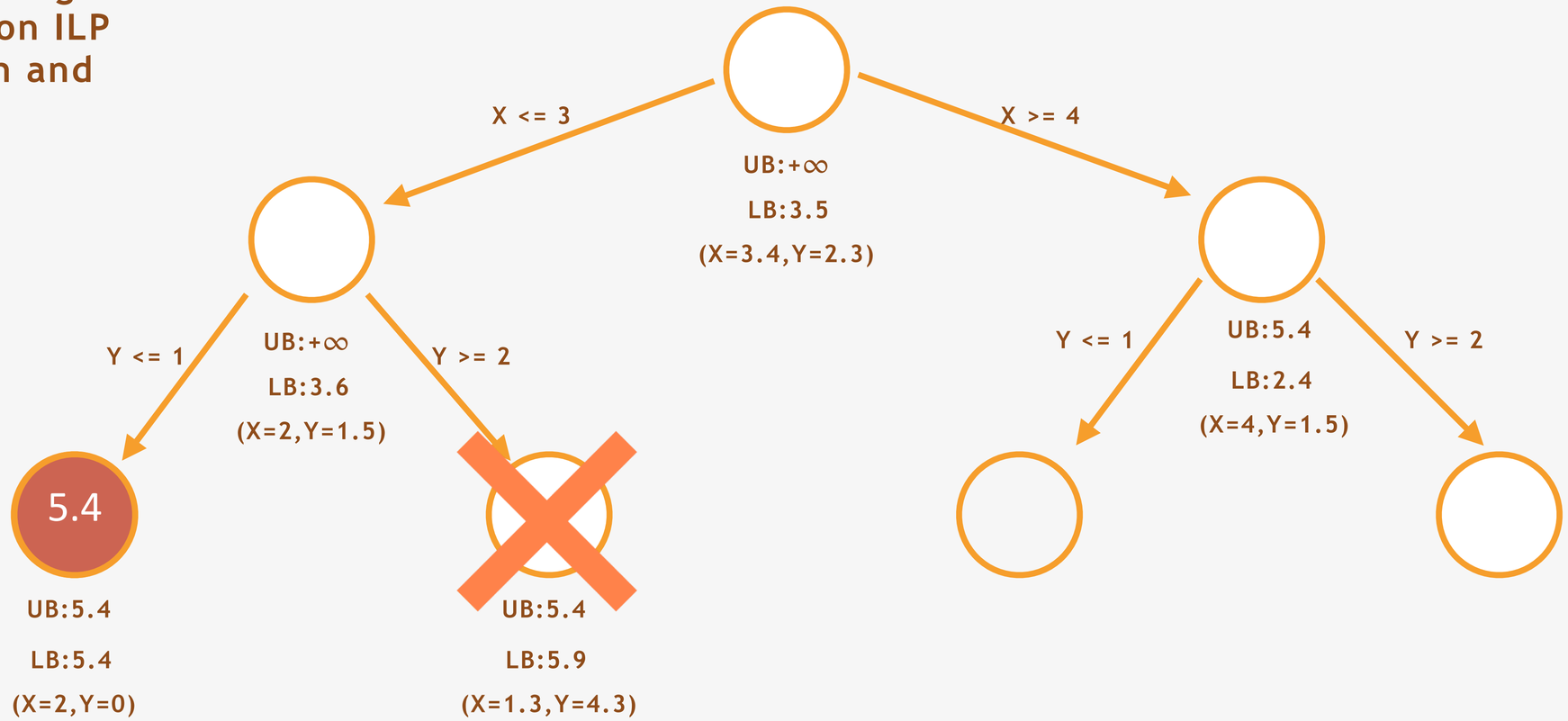
COUSIN OF THE BRANCH-AND-BOUND

Exemple: Solving of a Minimisation ILP With Branch and Bound

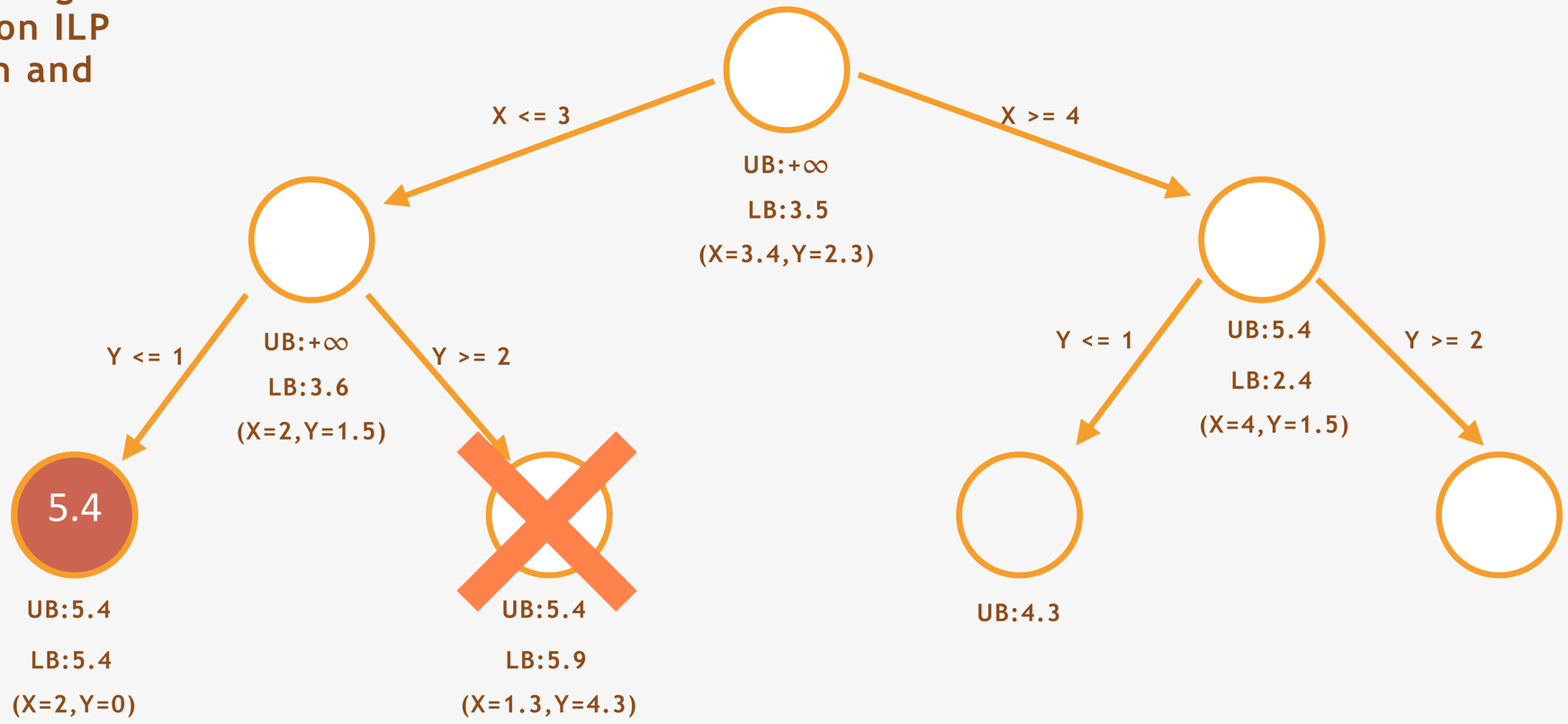


COUSIN OF THE BRANCH-AND-BOUND

Exemple: Solving of a Minimisation ILP With Branch and Bound

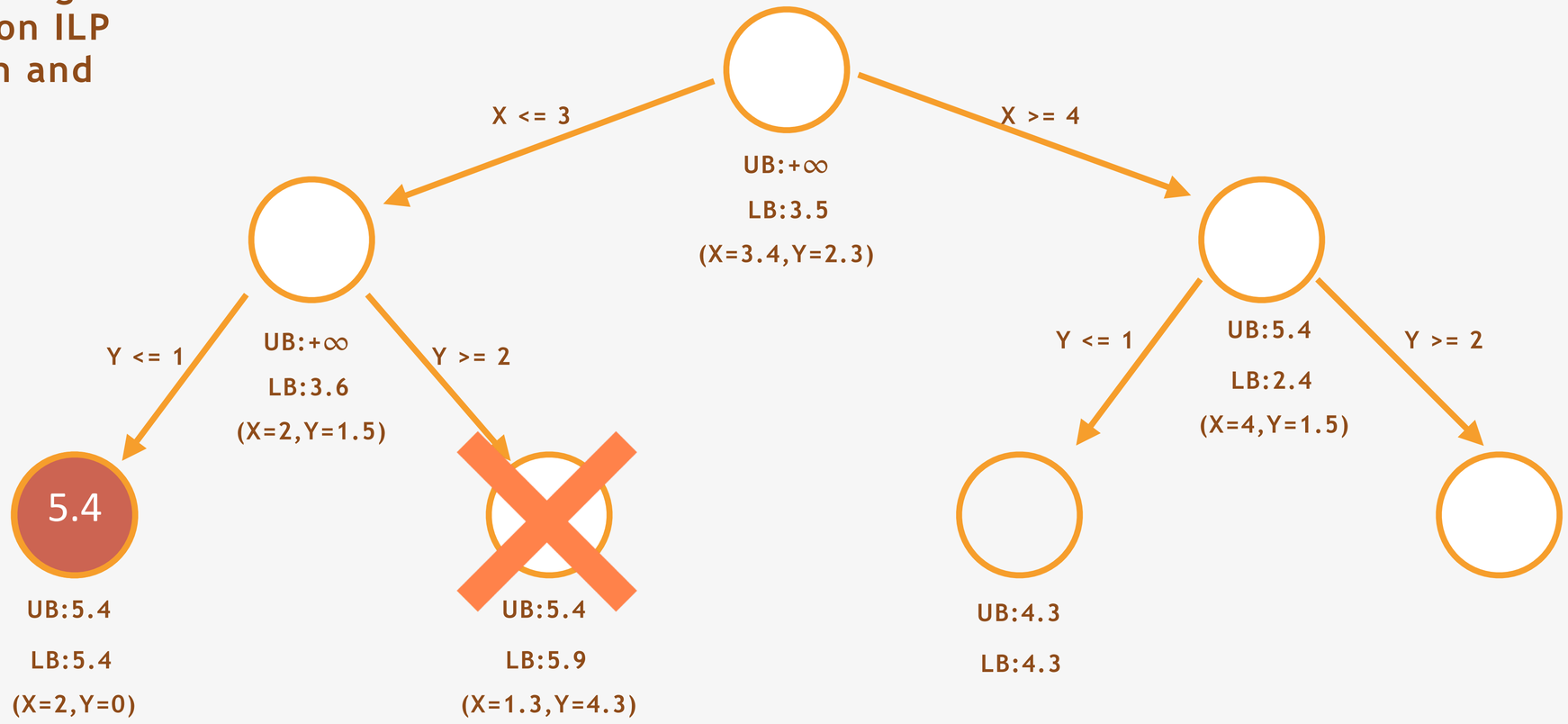


Exemple: Solving of a Minimisation ILP With Branch and Bound

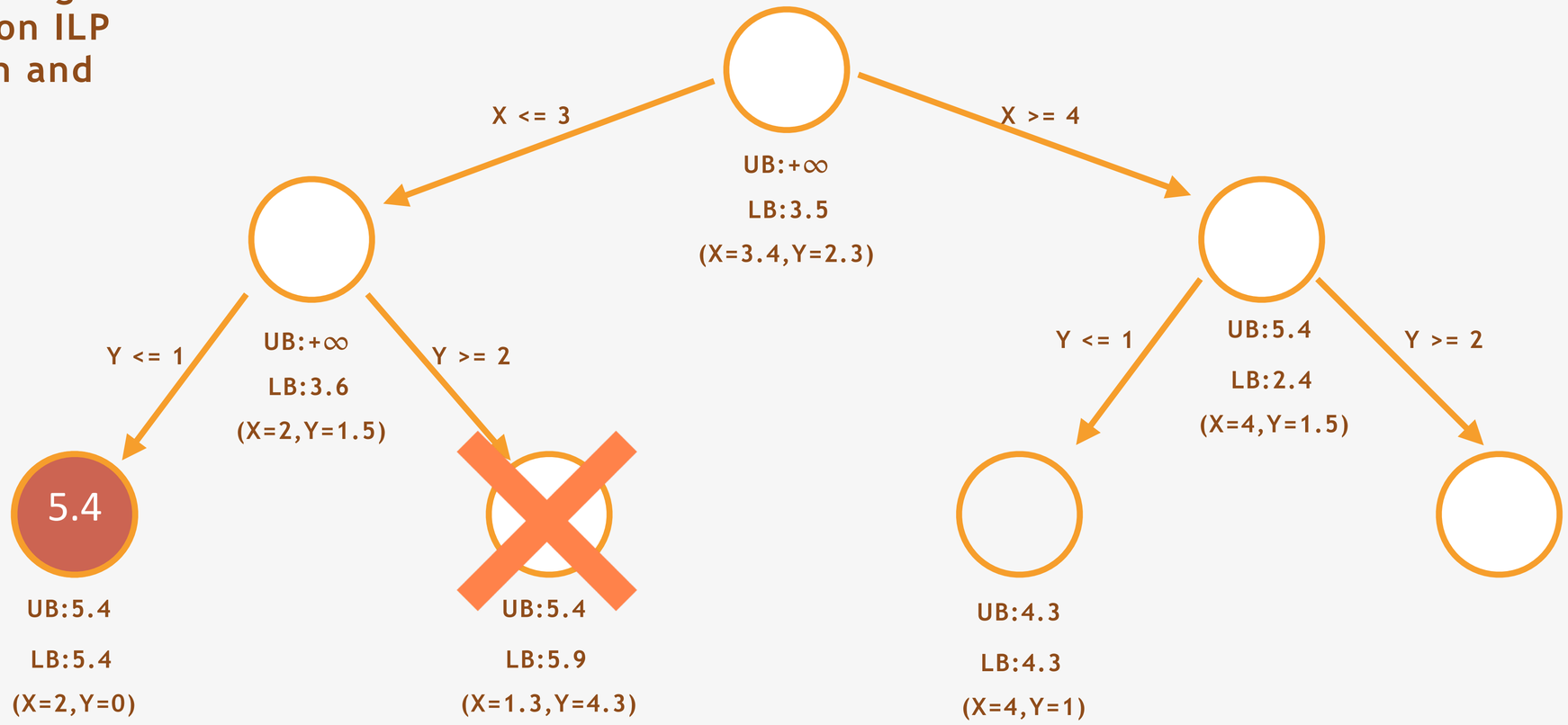


COUSIN OF THE BRANCH-AND-BOUND

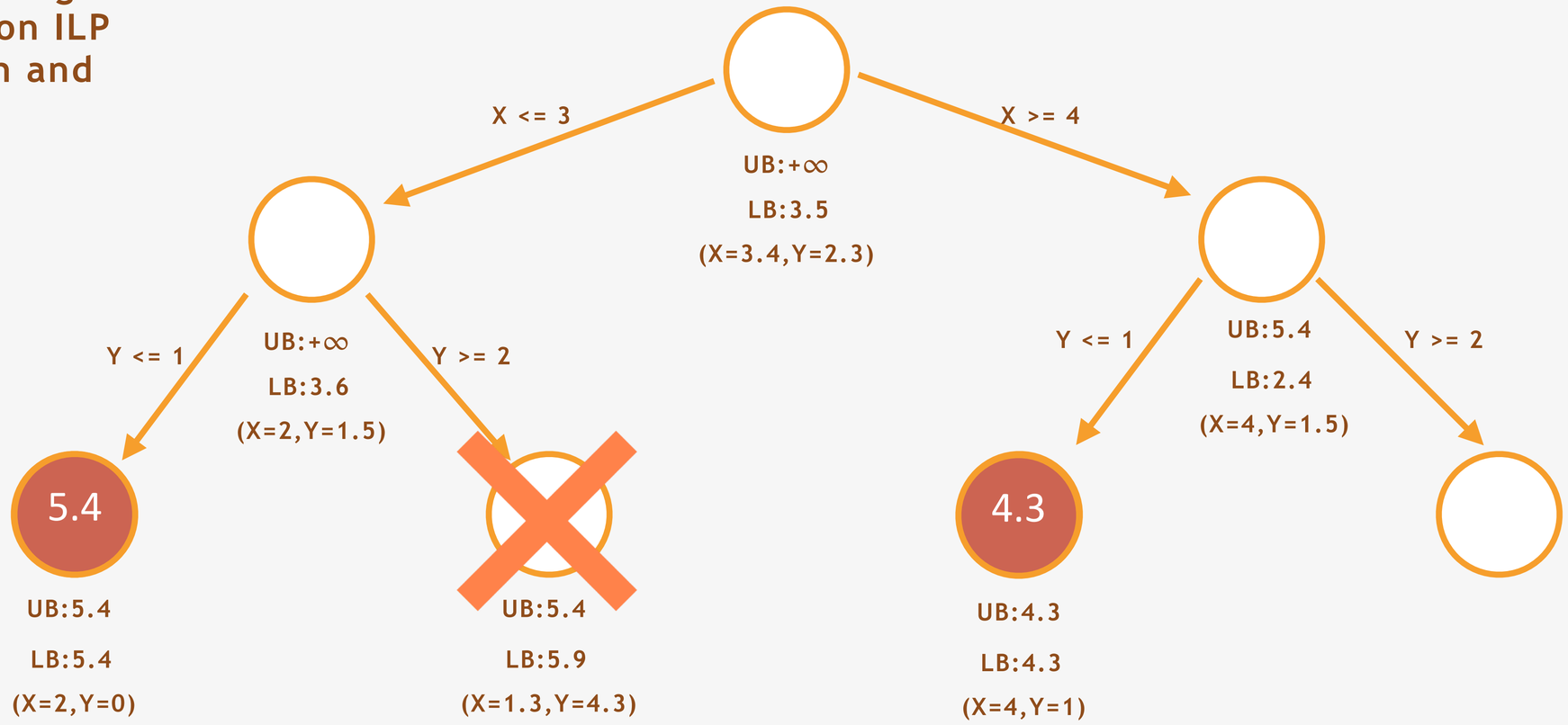
Exemple: Solving of a Minimisation ILP With Branch and Bound



Exemple: Solving of a Minimisation ILP With Branch and Bound

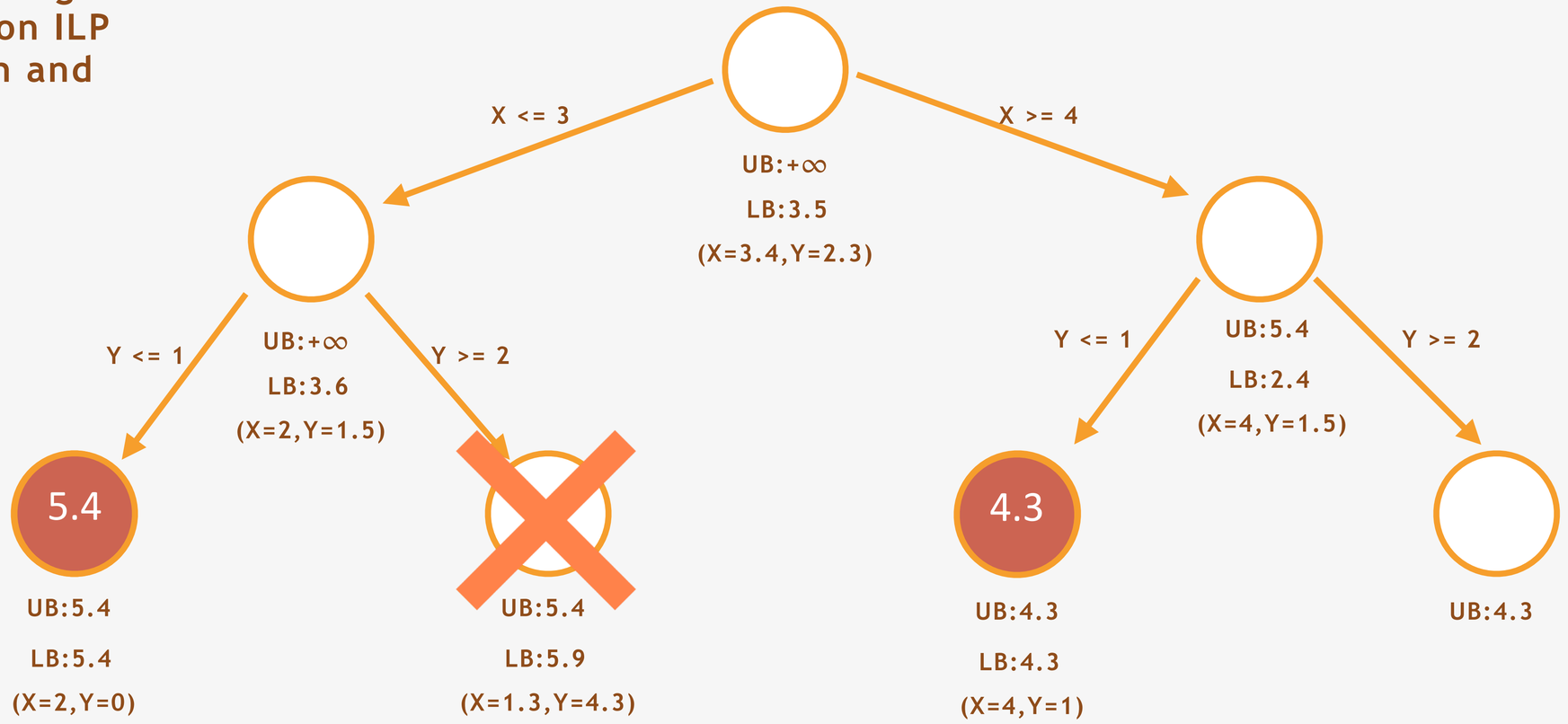


Exemple: Solving of a Minimisation ILP With Branch and Bound



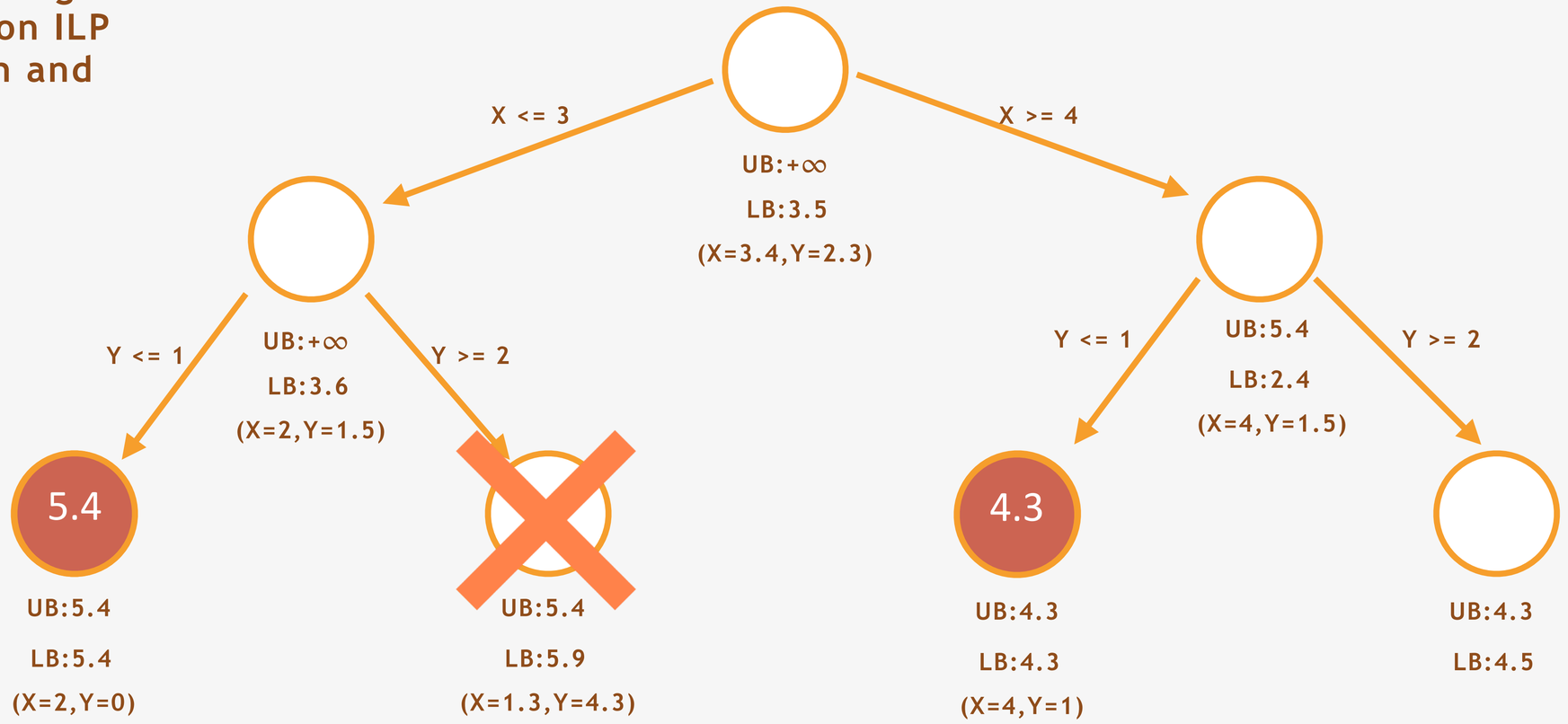
COUSIN OF THE BRANCH-AND-BOUND

Exemple: Solving of a Minimisation ILP With Branch and Bound

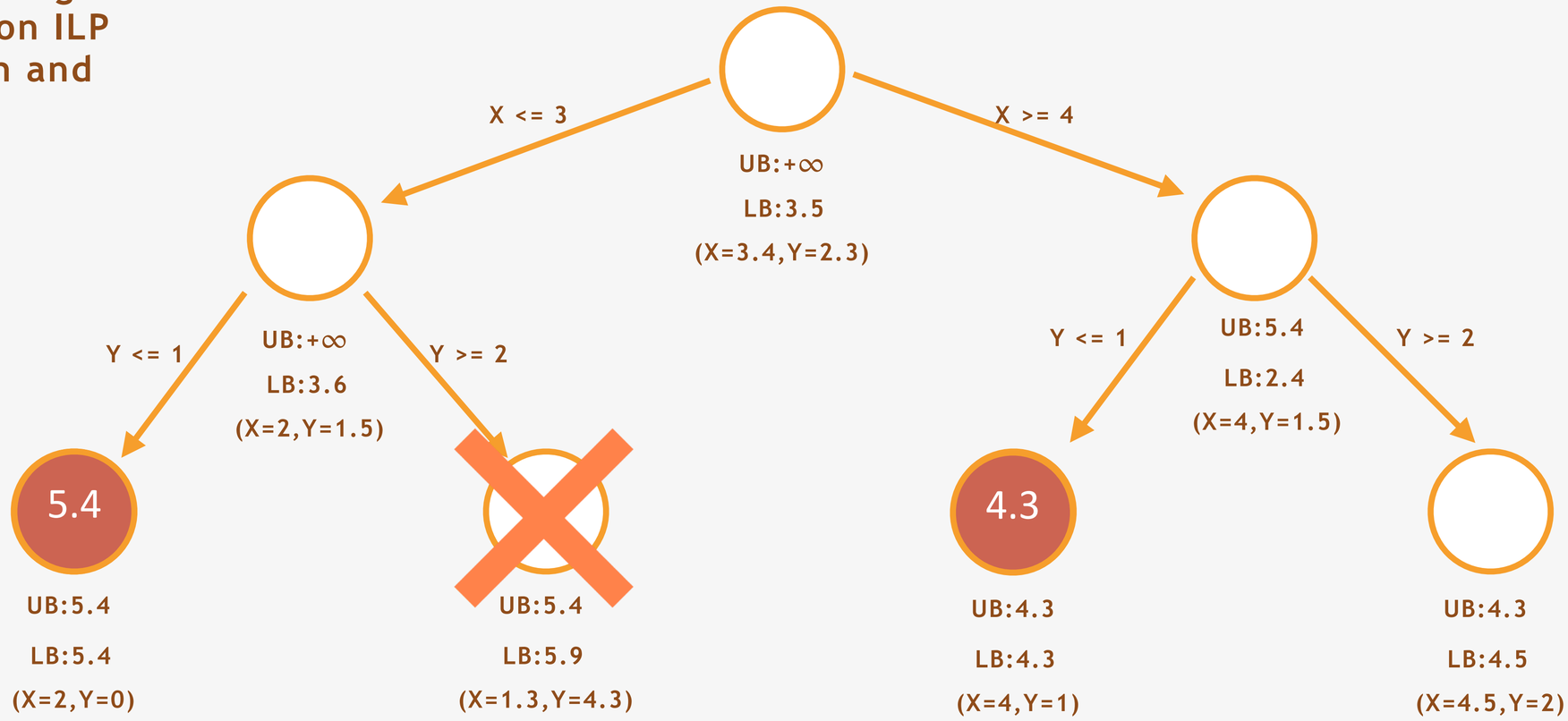


COUSIN OF THE BRANCH-AND-BOUND

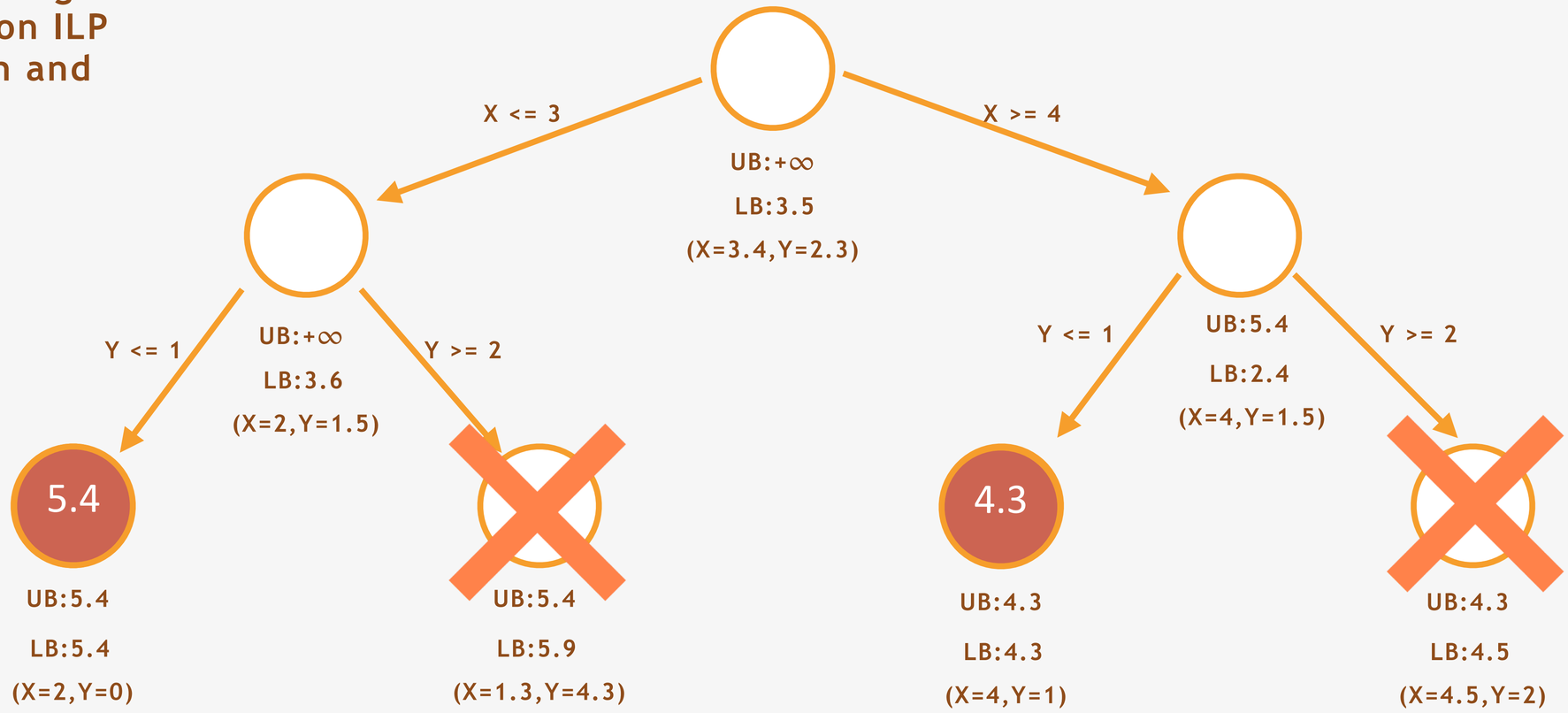
Exemple: Solving of a Minimisation ILP With Branch and Bound



Exemple: Solving of a Minimisation ILP With Branch and Bound

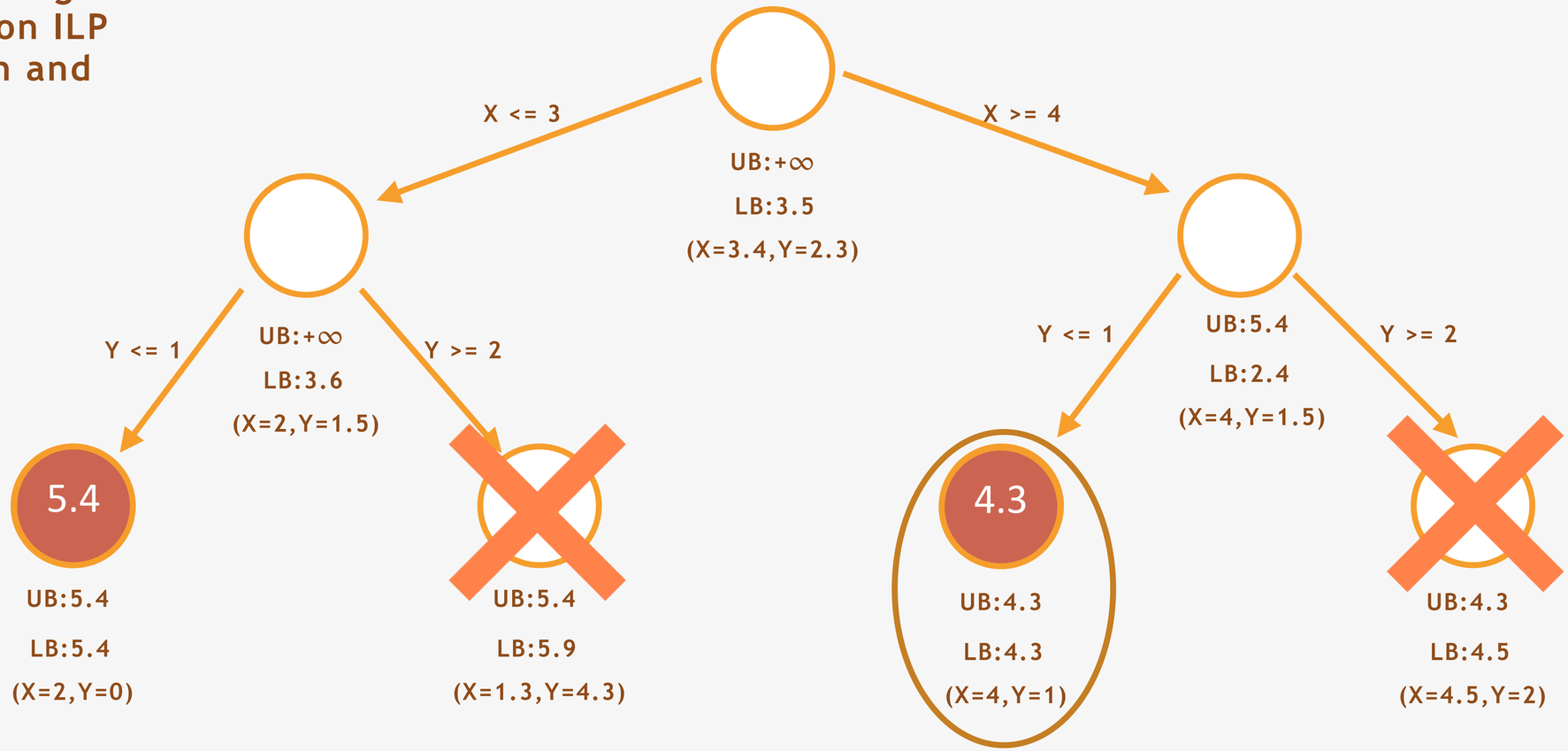


Exemple: Solving of a Minimisation ILP With Branch and Bound



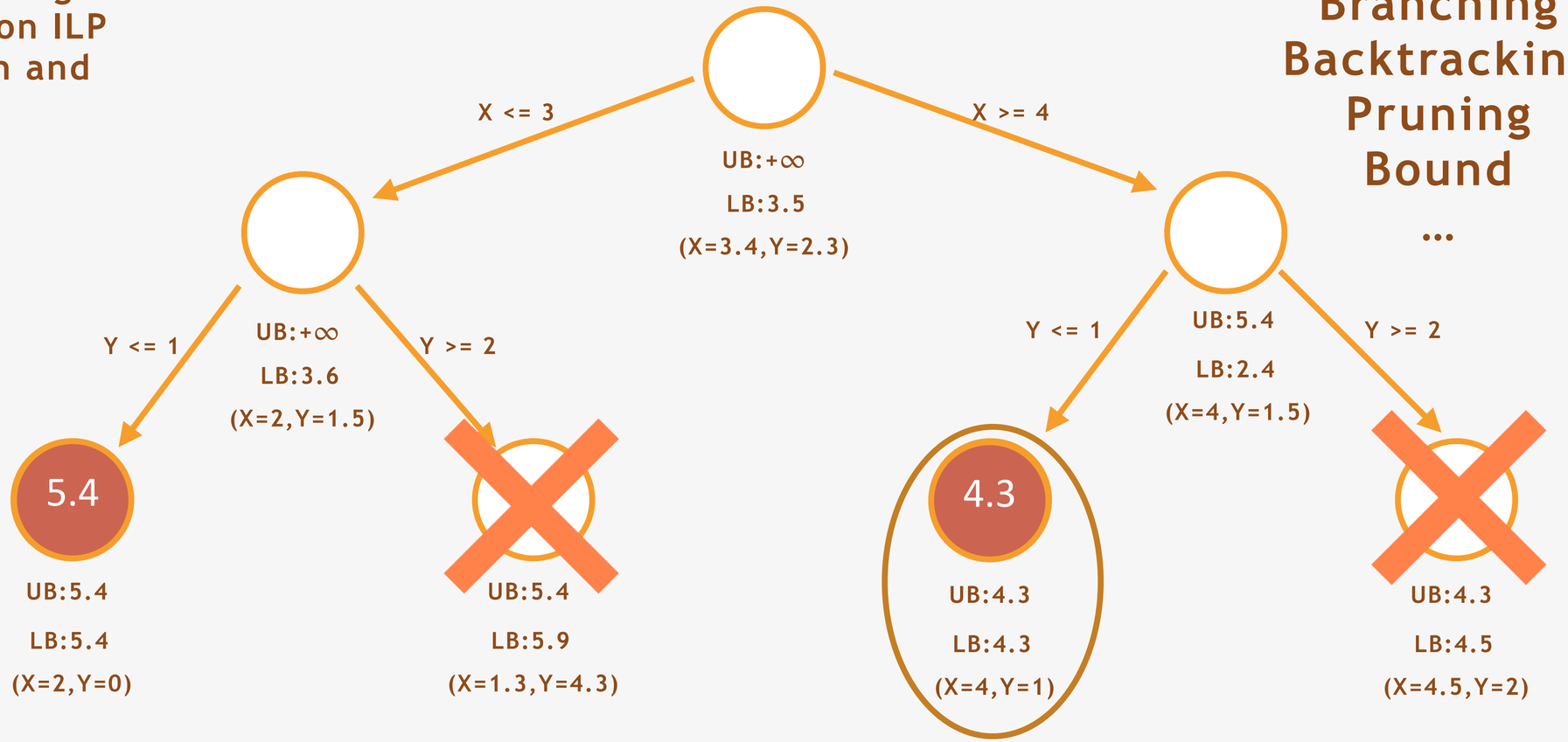
COUSIN OF THE BRANCH-AND-BOUND

Exemple: Solving of a Minimisation ILP With Branch and Bound



COUSIN OF THE BRANCH-AND-BOUND

Exemple: Solving of a Minimisation ILP With Branch and Bound



Elements Also Found in CP:

Branching
Backtracking
Pruning
Bound
...

SEARCH TREE IN MORE DETAILS

-Wednesday 9:00-10:30: Lecture

-Wednesday 11:00-12:30: Lecture

-Thursday 14:00-15:30: Lab

Advanced Solver Techniques

Hybrid solvers:

- Combination of the strengths of the various paradigms

SOME ADVANCED SOLVER TECHNIQUES

Hybrid solvers:

- Combination of the strengths of the various paradigms
- For example, OR-Tools CP-SAT-LP solver

Lazy Clause Generation and No-Goods:

SOME ADVANCED SOLVER TECHNIQUES

Hybrid solvers:

- Combination of the strengths of the various paradigms
- For example, OR-Tools CP-SAT-LP solver

Lazy Clause Generation and No-Goods:

- Learning of new clauses, new rules during the exploration to reduce the search

Proof-login Solvers:

SOME ADVANCED SOLVER TECHNIQUES

Hybrid solvers:

- Combination of the strengths of the various paradigms
- For example, OR-Tools CP-SAT-LP solver

Lazy Clause Generation and No-Goods:

- Learning of new clauses, new rules during the exploration to reduce the search

Proof-login Solvers:

- Output a formal verifiable logical proof that an optimal is optimal or that there exists no solution

Portfolio Solvers:

SOME ADVANCED SOLVER TECHNIQUES

Hybrid solvers:

- Combination of the strengths of the various paradigms
- For example, OR-Tools CP-SAT-LP solver

Lazy Clause Generation and No-Goods:

- Learning of new clauses, new rules during the exploration to reduce the search

Proof-login Solvers:

- Output a formal verifiable logical proof that an optimal is optimal or that there exists no solution

Portfolio Solvers:

- Combination of multiple solvers, often in parallel, which share information on the solving process

SOME ADVANCED SOLVER TECHNIQUES

Hybrid solvers:

- Combination of the strengths of the various paradigms
- For example, OR-Tools CP-SAT-LP solver

Lazy Clause Generation and No-Goods:

- Learning of new clauses, new rules during the exploration to reduce the search

Proof-login Solvers:

- Output a formal verifiable logical proof that an optimal is optimal or that there exists no solution

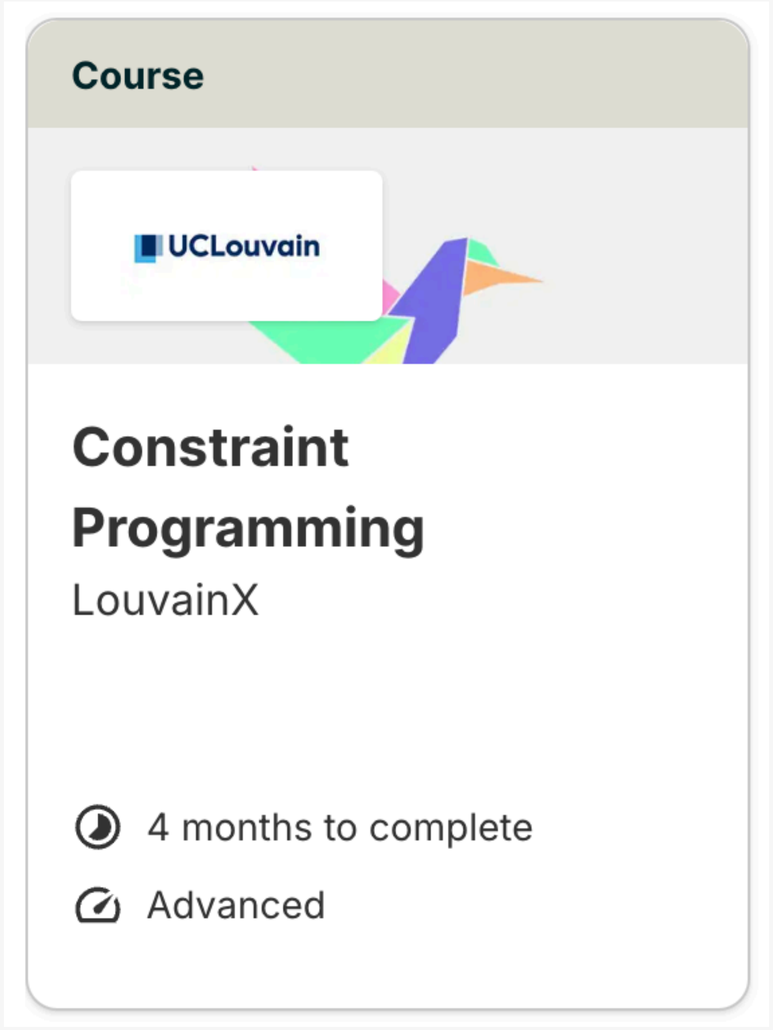
Portfolio Solvers:

- Combination of multiple solvers, often in parallel, which share information on the solving process

To Go Further…

ONLINE MOOC ON EDX

www.edx.org



Modelling problems, implementing global constraints, design searches,…

ONLINE MOOC ON COURSERA

Ouidah, Benin
25-29th August 2025

www.coursera.org

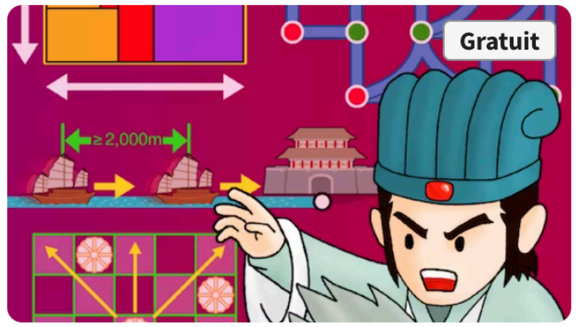
The University of Melbourne

Basic Modeling for Discrete Optimization

Compétences que vous acquerrez:
Programmation informatique

★ **4.8** (432 avis)

Intermédiaire · Cours · 1 à 4 semaines

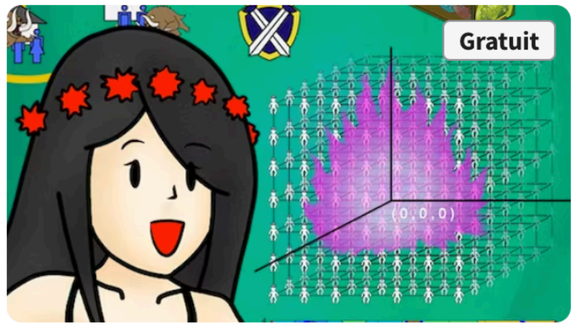


The University of Melbourne

Advanced Modeling for Discrete Optimization

★ **4.9** (131 avis)

Intermédiaire · Cours · 1 à 3 mois



The University of Melbourne

Solving Algorithms for Discrete Optimization

★ **4.8** (40 avis)

Intermédiaire · Cours · 1 à 4 semaines

Modelling of Combinatorial Optimisation Problems With MiniZinc

Solving Technologies for Combinatorial Optimisation

ACP YOUTUBE CHANNEL

Ouidah, Benin
25-29th August 2025

https://www.youtube.com/@associationforconstraintpr9021

Association for Constraint Programming

@associationforconstraintpr9021 · 475 abonnés · 53 vidéos

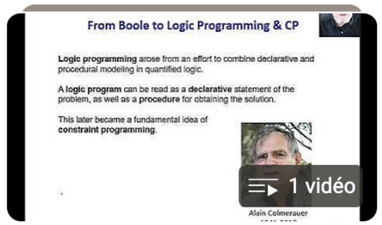
This channel features videos from the Association for Constraint Programming.

a4cp.org

S'abonner

Accueil Vidéos Playlists Communauté Chaînes

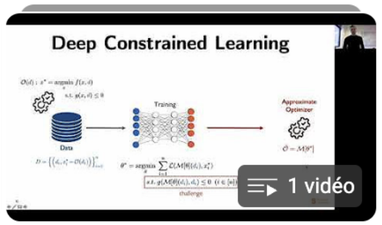
ACP Awards



Research Excellence Awards

Association for Constraint Programmin

Afficher la playlist complète



Early Career Researcher Awards

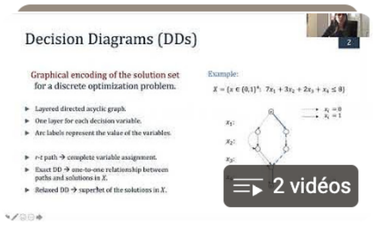
Association for Constraint Programmin

Afficher la playlist complète

Distinguished Service Awards

Association for Constraint Programmin

Afficher la playlist complète



Doctoral Research Awards

Association for Constraint Programmin

Afficher la playlist complète

Many Open Source Solvers, Do Not Hesitate To Dive in the Documentation and the Code To Know More!

Gecode

Many Open Source Solvers, Do Not Hesitate To Dive in the Documentation and the Code To Know More!

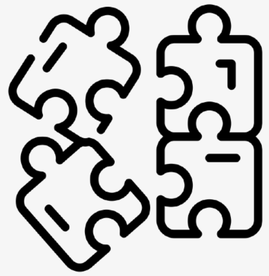
Specially Designed and Coded for Learning

Gecode

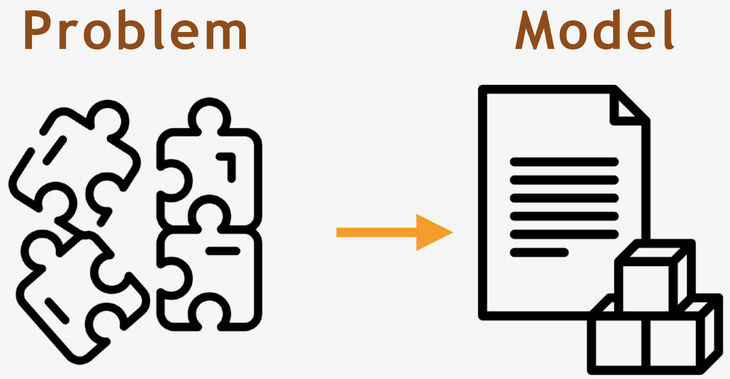
Google OR-Tools

To Conclude

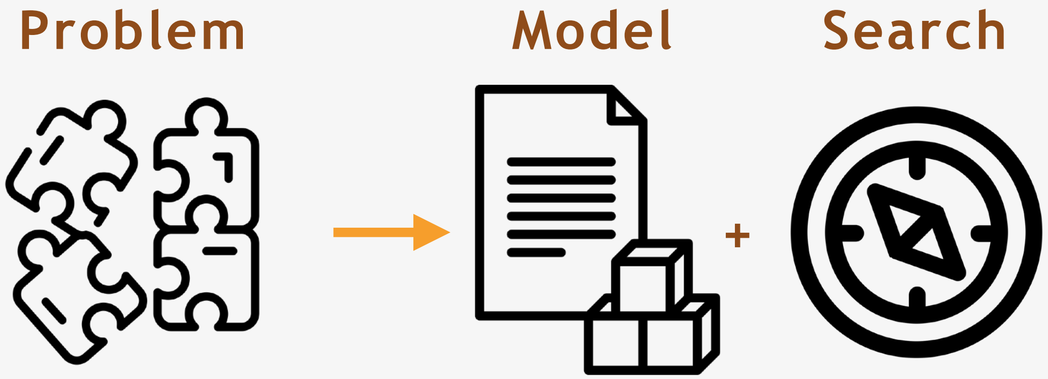
Problem



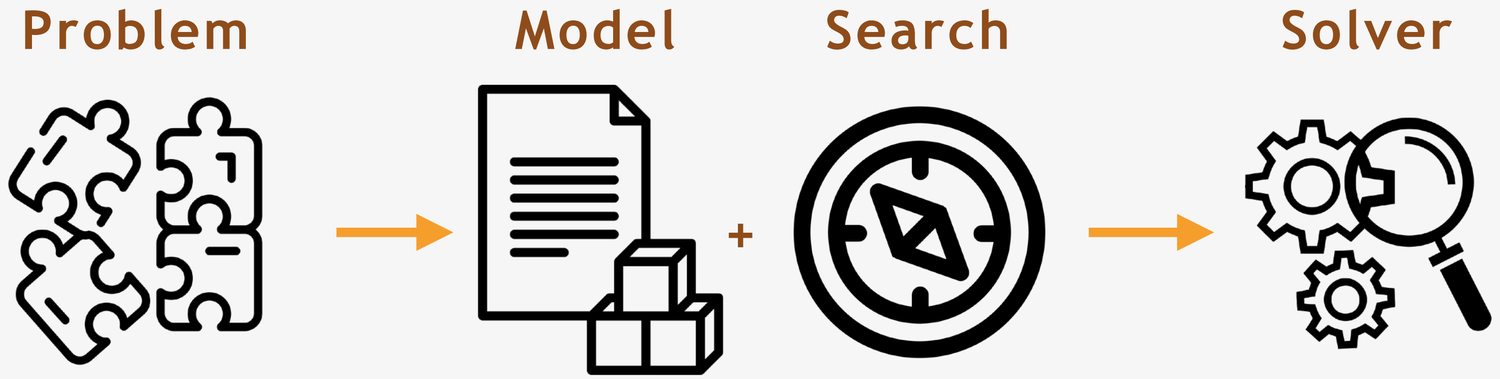
TO SOLVE A PROBLEM WITH CP YOU WILL NEED...



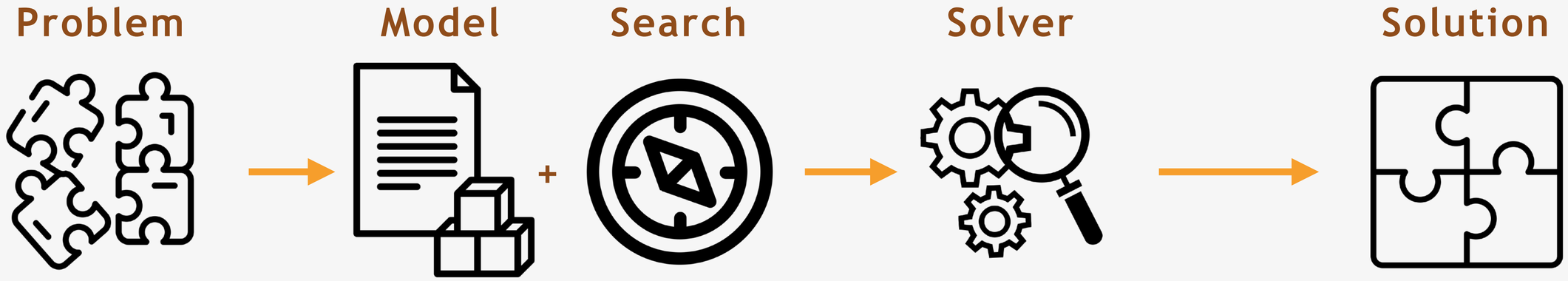
TO SOLVE A PROBLEM WITH CP YOU WILL NEED...



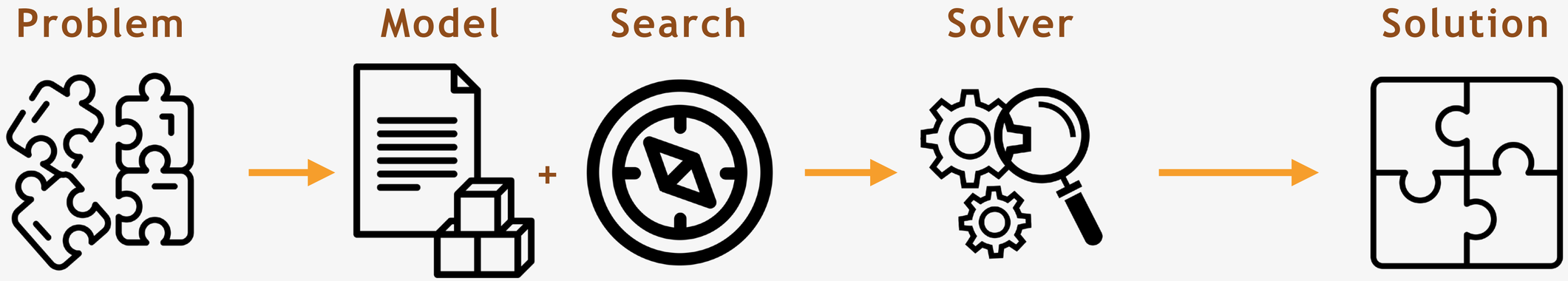
TO SOLVE A PROBLEM WITH CP YOU WILL NEED...



TO SOLVE A PROBLEM WITH CP YOU WILL NEED...

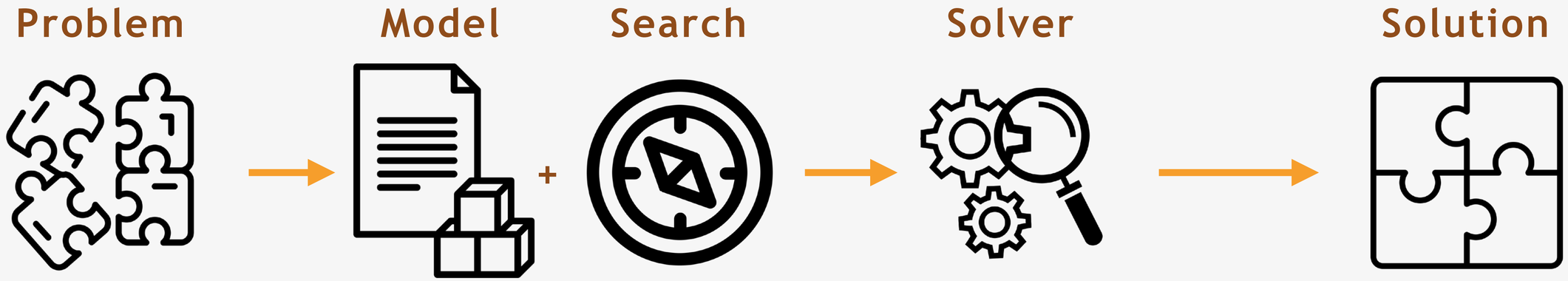


TO SOLVE A PROBLEM WITH CP YOU WILL NEED...



Model = declarative description of a solution to the problem, using variables and (global) constraints

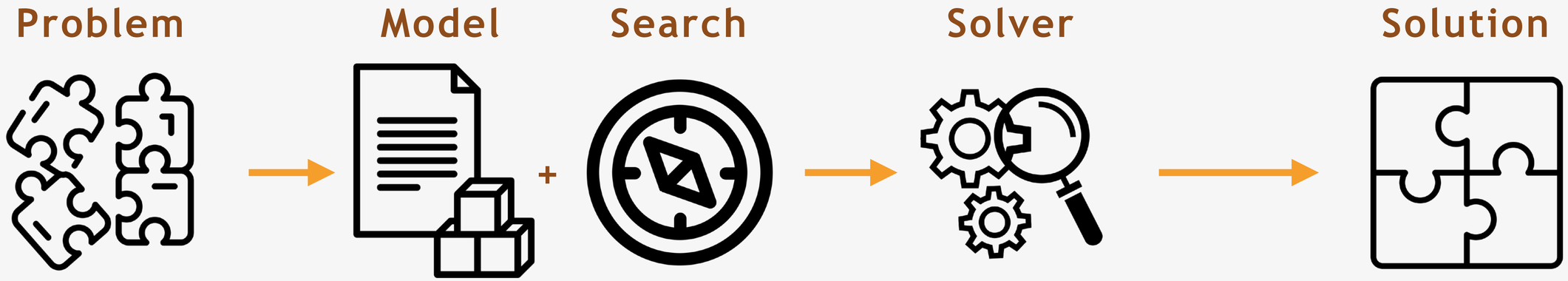
TO SOLVE A PROBLEM WITH CP YOU WILL NEED...



Model = declarative description of a solution to the problem, using variables and (global) constraints

Search = heuristic to guide the search tree

TO SOLVE A PROBLEM WITH CP YOU WILL NEED...



Model = declarative description of a solution to the problem, using variables and (global) constraints

Search = heuristic to guide the search tree

Solver = a search tree algorithm, exploring the solution space guided by the search heuristic, which regularly call the fix-point algorithm, which organises the calls to the concerned propagators to cut the values in the state that are not valid anymore