# Bridging Continuous and Discrete Optimization in Julia

Thibaut Cuvelier
Google Research France (Operations Research)
https://developers.google.com/optimization

# OR @ Google

**Routing**: logistics, Google Street View

- Open-source solver
- B2B API: GMPRO

**High-level solvers**:

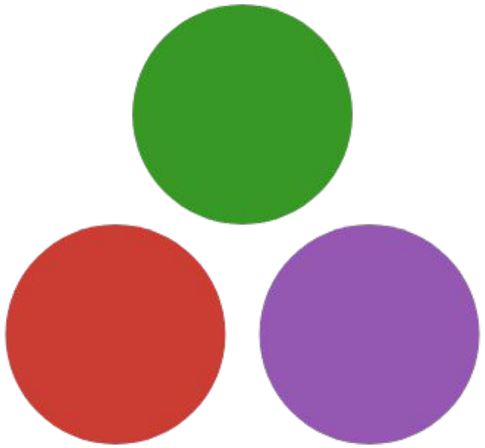
- Workforce scheduling (API)
- Shipping network design (API)

**Low-level solvers**:

- Glop: LP solver (simplex)
- CP-SAT: CP solver (SAT, LP), 10+ gold medals in MiniZinc competition
- PDLP: LP solver (first order)
- MathOpt: modelling layer

**Open-source product: OR-Tools**

# Julia for optimisation



01 **What is Julia?**

02 **Julia for optimisation**

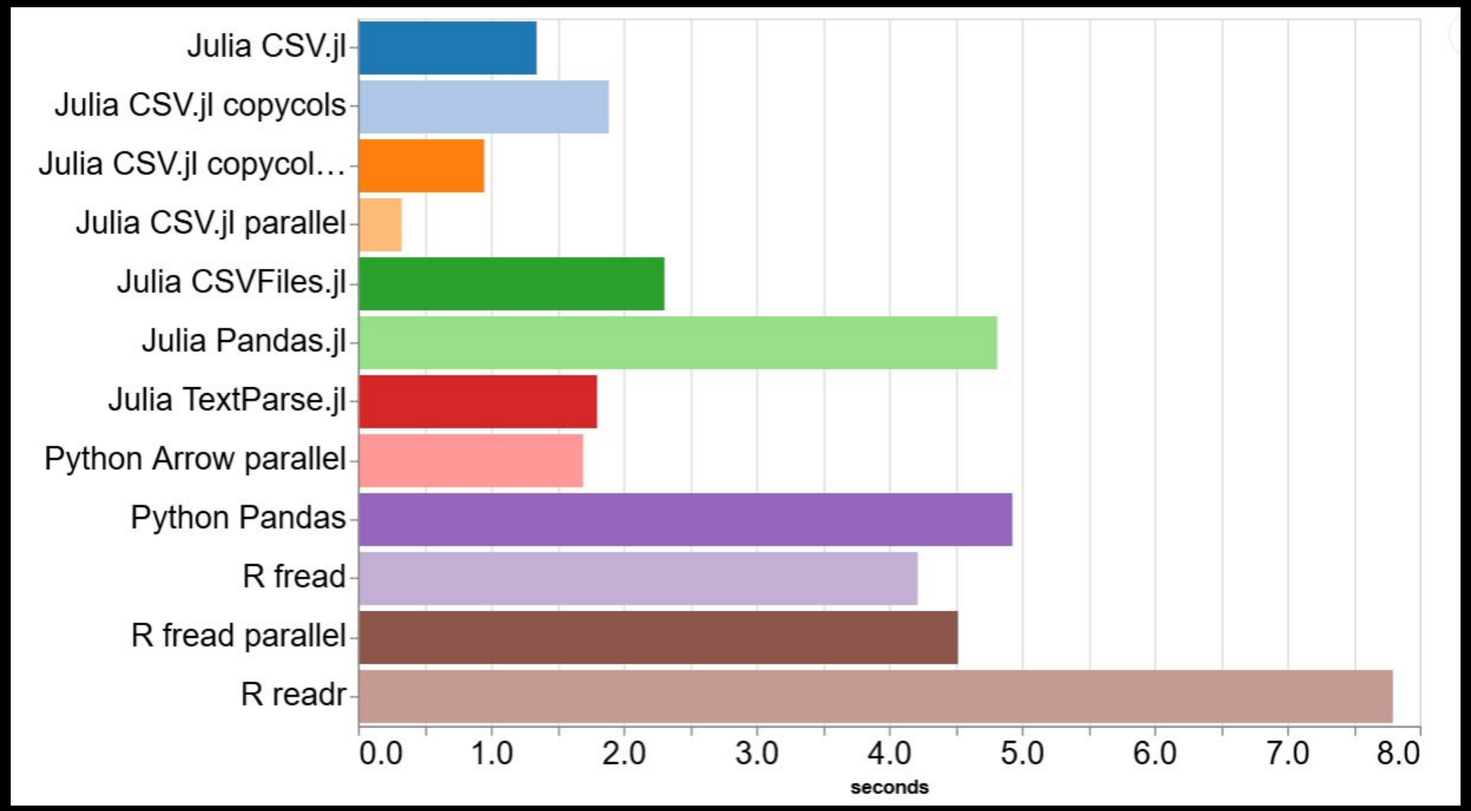
Ease of modelling
Ease of writing new solvers

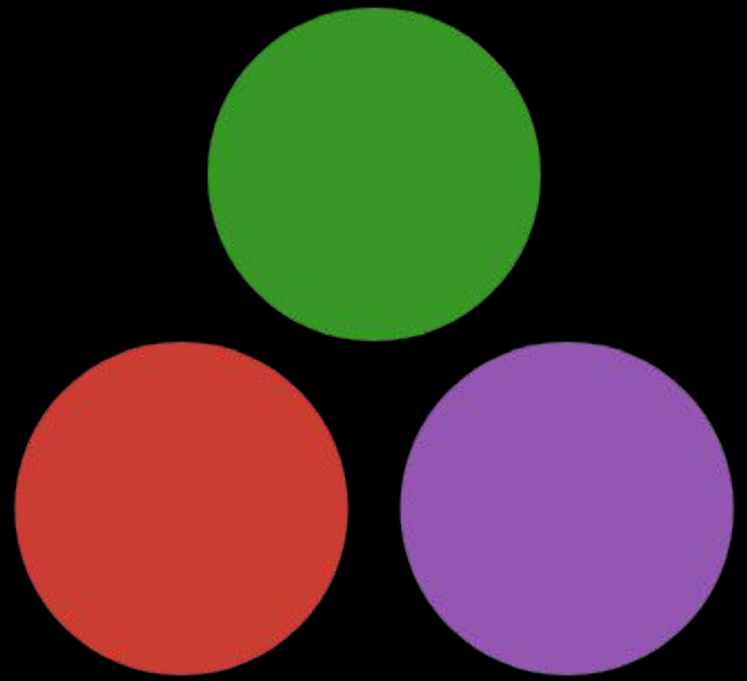
03 **Bridging the gap**

The example of CP-SAT

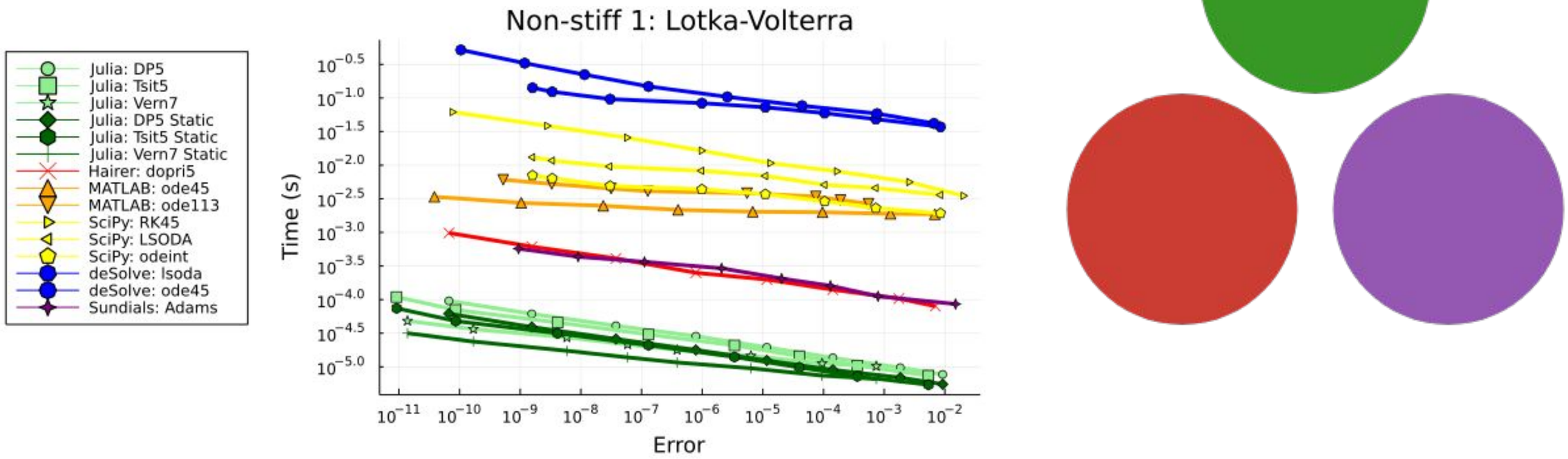
01

# Why Julia?


Julia CSV.jl
Julia CSV.jl copycols
Julia CSV.jl copycol…
Julia CSV.jl parallel
Julia CSVFiles.jl
Julia Pandas.jl
Julia TextParse.jl
Python Arrow parallel
Python Pandas
R fread
R fread parallel
R readr
0.0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
seconds



# Why Julia?


Julia: DP5
Julia: Tsit5
Julia: Vern7
Julia: DP5 Static
Julia: Tsit5 Static
Julia: Vern7 Static
Hairer: dopri5
MATLAB: ode45
MATLAB: ode113
SciPy: RK45
SciPy: LSODA
SciPy: odeint
deSolve: lsoda
deSolve: ode45
Sundials: Adams
Non-stiff 1: Lotka-Volterra
Time (s)
Error

# Why Julia?

1

## Extremely fast

You don't need a second language for performance!

2

## Extremely readable for mathematics

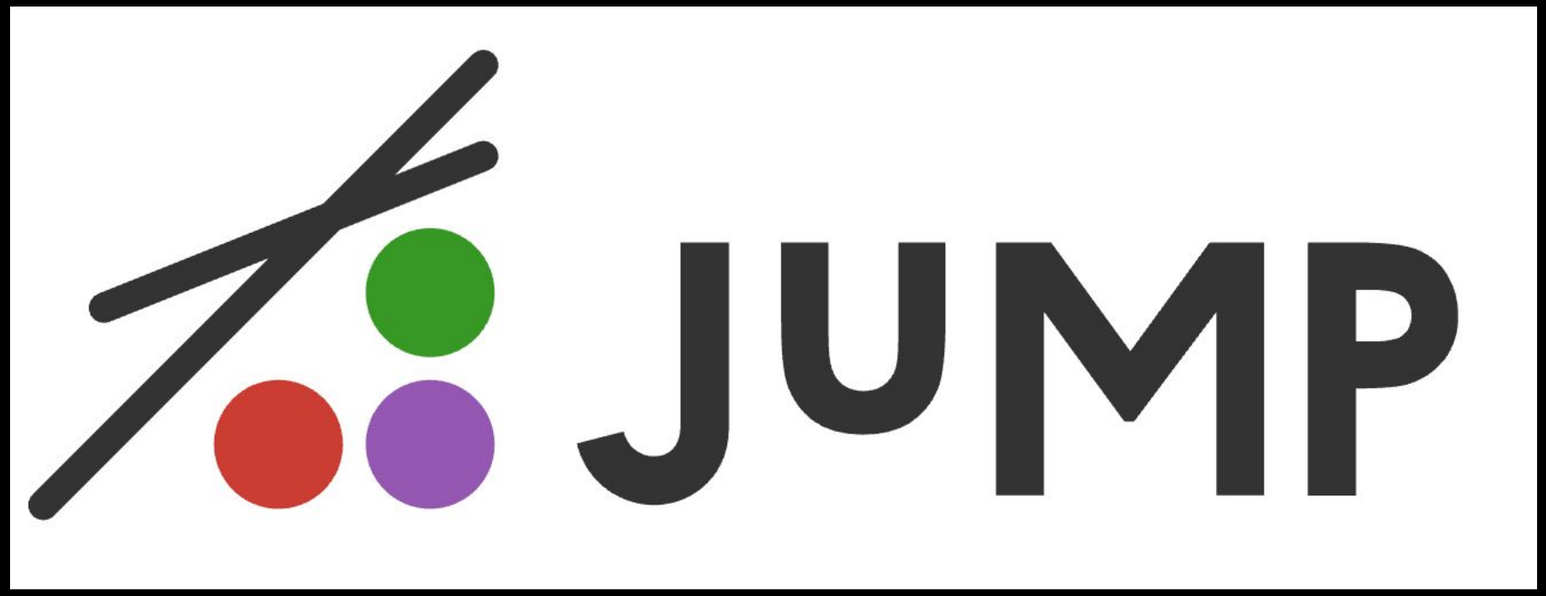
Your code reads like a paper!

```
@parameters σ ρ β
@variables x(t) y(t) z(t)

eqs = [D(D(x)) ~ σ * (y - x),
       D(y) ~ x * (ρ - z) - y,
       D(z) ~ x * y - β * z]
```

02

# Julia for optimisation



# This is a least-squares model

```
m, n = size(A)
model = Model(Ipopt.Optimizer)
@variable(model, x[1:n])
@variable(model, residuals[1:m])
@constraint(model, residuals == A * x - b)
@constraint(model, sum(x) == 1)
@objective(model, Min, sum(residuals.^2))
optimize!(model)
```

# What's behind?

1

## JuMP

The modelling layer

Like AMPL

2

## MathOptInterface

The generic interface for solvers

Like AMPL's ASL or MP

Great performance!

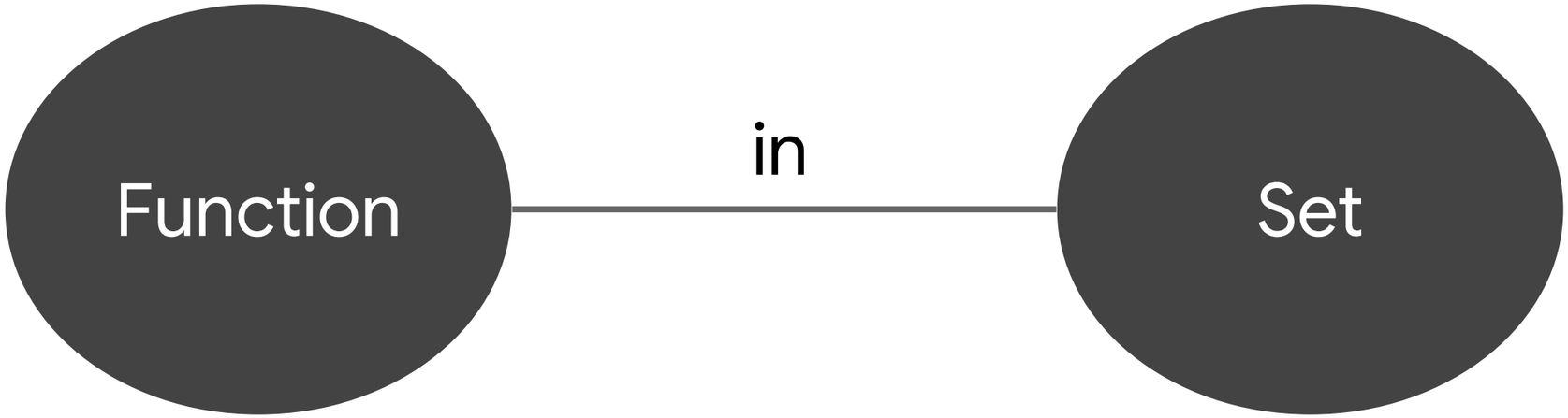
3

## A solver

Well, a solver, you know

JuMP and MathOptInterface were made for mathematical optimisation:
LPs, QPs, MIPs, conic programs, etc.

# MathOptInterface's formalism




Single variable
Affine expression
Quadratic expression
Nonlinear expression
Vector of expressions
...

Less than
Greater than
Equal to
Zero or one
SOCP cone
SDP cone
Exponential cone
...

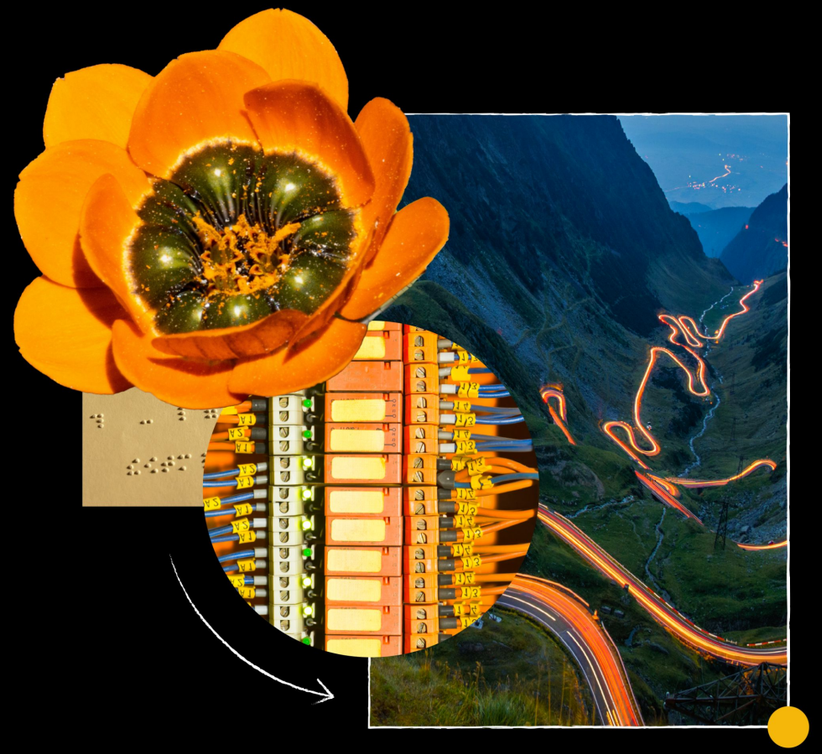
# Julia-native solvers

Mostly research codes:

- Hypatia: generic conic interior-point solver
- Pavito: MINLP
- Pajarito: mixed-integer convex solver
- Alpine: MINLP
- MathOptIIS: computing IIS in 600 lines of code
- FirstOrderLp: gradient descent for LP (now part of OR-Tools!)
- Manopt: optimisation on manifolds
- ...

03

# Bridging the gap



# MathOptInterface and constraint programming

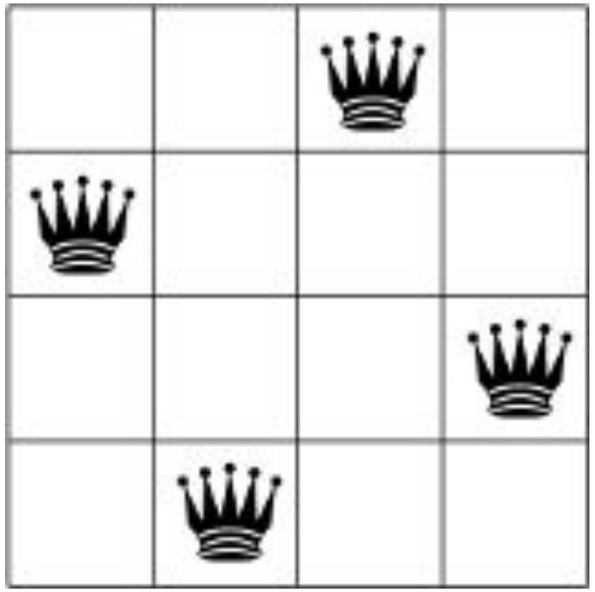
# Constraint programming

A constraint doesn't need to be algebraic!

"Global" constraints:

- `alldifferent`
- `notequal` ≠
- Reified constraints
- Interval constraints

In the end: potentially more efficient algorithms



- One queen per row: variables are column positions
- `alldifferent(positions)`: not all queens in the same column
- `alldifferent(positions[i] ± i)`: no two queens in the same diagonal

# Constraint programming and MathOptInterface

## A good fit?

- **Yes**, the function-in-set formalism is great!
  - Function-in-set:
    - Vector of affine expressions -in- `alldifferent`
    - Vector of affine expressions -in- `countdistinct` — one variable has a special role
    - Vector of affine expressions -in- `path(graph)`
    - ...
  - In total: ~ 80 sets in ConstraintProgrammingExtensions.jl
    - 11 backported to MathOptInterface

# Constraint programming and MathOptInterface

## A good fit?

- **Yes**, the solvers have the same way of thinking!
  - It's quite easy to write wrappers for existing solvers
  - Chuffed, CPLEX-CP, FlatZinc (many solvers like CP-SAT!)
- **No**, people want to build complex constraints like

$$|x| + |y| \bmod 5 == 2$$

That's still possible with MathOptInterface's nonlinear expressions

# MathOptInterface bridges

- Not all solvers support all constraints
  - Intervals: $0 \leq x \leq 10$
  - Semi-continuous variables: $x = 0$ *or* $x \geq 10$
  - SOC for a SDP solver
- MathOptInterface has the notion of bridge:
  - Map one constraint into another
  - Graph: nodes are constraint types, bridges are edges between them

And constraint programming? It maps more or less cleanly to MIP!

Nearly 70 CP-to-MIP bridges in ConstraintProgrammingExtensions.jl

# MathOptInterface bridges

- Constraint programming maps more or less cleanly to MIP
- **But**: bridges work per constraint
  - You could have a much better formulation with several constraints

- Still, very useful to model higher-level constraints, like circuits
  - A MIP model, but easier to read
  - A richer representation can be useful for decompositions

# CP-SAT

# The CP-SAT sauce

1

## CP

High-level constraints

More efficient handling

Tree search, like a CP solver

2

## SAT

Internally, reformulate as a satisfaction problem

Clause generation: **why** is a branch of the search tree infeasible?

3

## LP relaxation

Find bounds (objective, variables) in the search tree

# The CP-SAT sauce

4

## Heuristics

SAT heuristics, RINS

Large neighbourhood search

5

## Scheduling

Many scheduling components:
interval variables,
scheduling constraints

6

## Portfolio

Many threads in parallel:
LNS, various heuristics,
all of them communicating

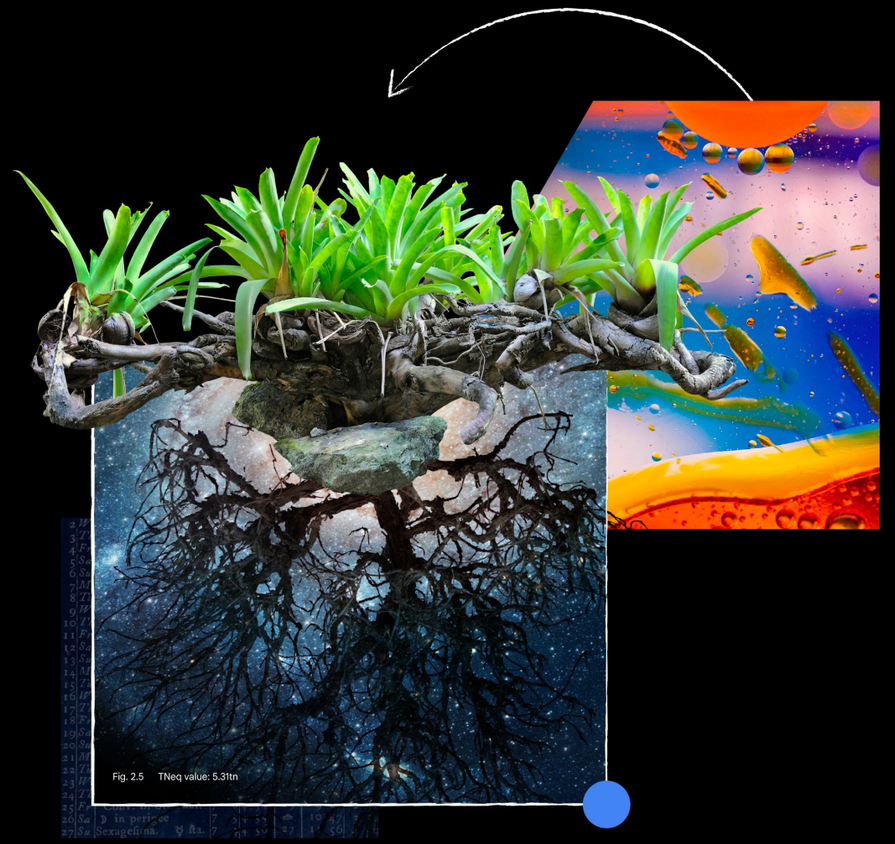
More on CP-SAT: masterclass at CPAIOR 2020

# CP-SAT's LP relaxation

- Relaxation: an LP encoding of a CP model
  - Only a few constraints have a relaxation, most are expanded
- Variable encoding for integer variables:
  - Dynamic: create a new variable, such as "x ≤ 5" during conflict detection
  - Static: when you will likely need many variables, do it up front
- LP relaxation means... cuts!
  - Goal: have an LP relaxation closer to the integer program
  - Typical MIP cuts for scheduling, for instance

04

# Final words



# By the way...

- All Google solvers for mathematical programming are now available in Julia!

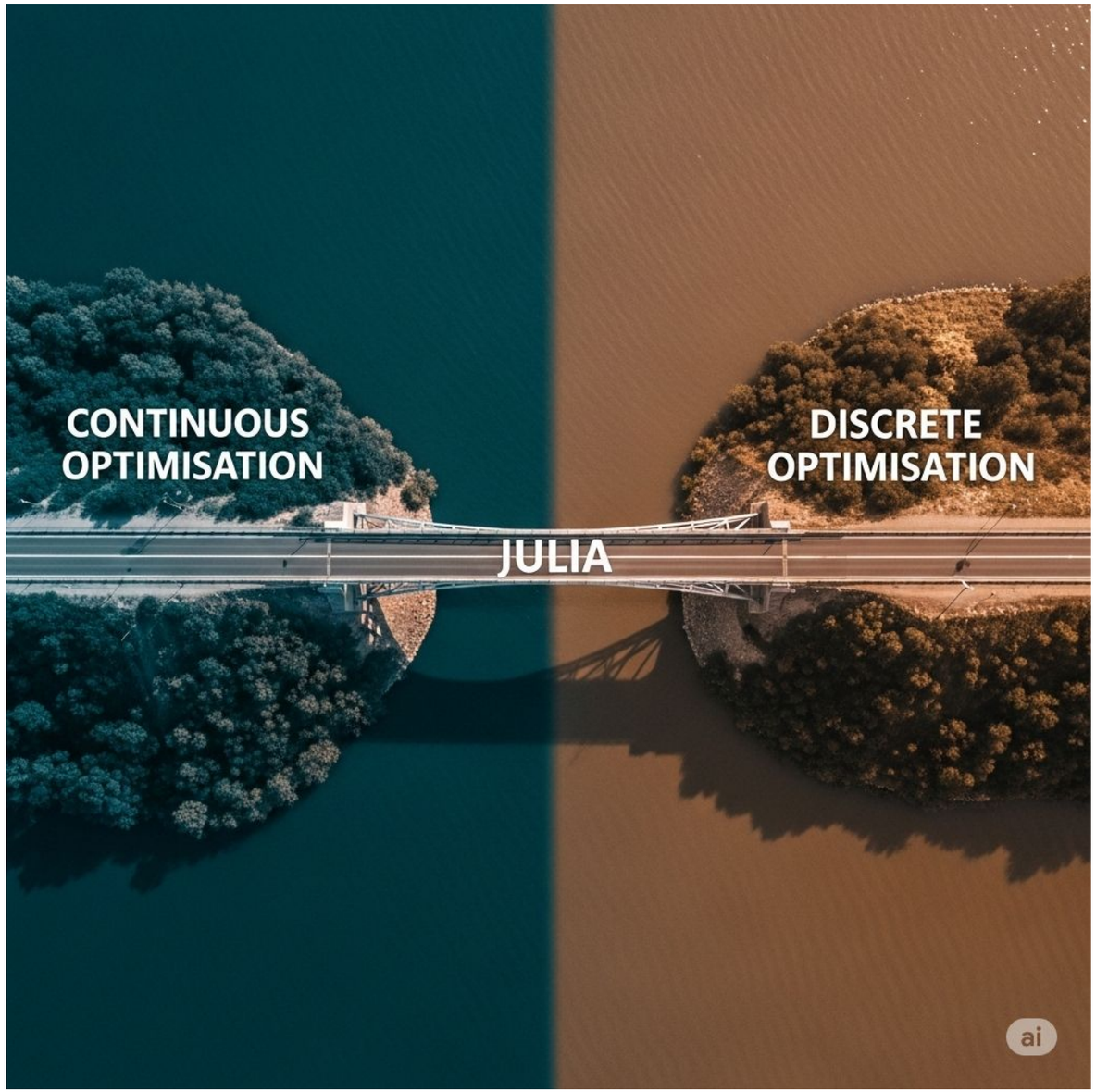
  ORTools.jl

  Details: JuliaCon 2025, July 24th

# Key takeaways

- **Julia is great!**
  - Great for scientific and numerical use cases — performance and syntax
  - Same syntax for continuous and discrete optimisation

- **Several bridges between continuous and discrete optimisation**
  - CP models are very similar to MIPs/LPs
  - State-of-the-art CP solvers use linear relaxations in novel ways



# Thank You