# Implementation with SCIP

Ambros M. Gleixner

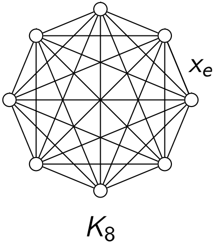
slides by Timo Berthold, Stefan Heinz, and Kati Wolter

1 December 2011

Berlin Mathematical School

DFG Research Center MATHEON
Mathematics for key technologies

# TSP – Integer Programming Formulation




Given

- $\triangleright$ Complete graph $G = (V, E)$
- $\triangleright$ for each $e \in E$ a distance $d_e > 0$

Binary variables

- $\triangleright$ $x_e = 1$ if edge $e$ is used

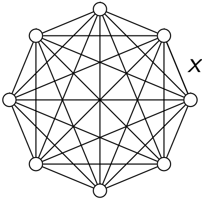
# TSP – Integer Programming Formulation

Given

- ▷ Complete graph $G = (V, E)$
- ▷ for each $e \in E$ a distance $d_e > 0$

Binary variables

- ▷ $x_e = 1$ if edge $e$ is used




$$
\begin{aligned}
\min \quad & \sum_{e \in E} d_e \, x_e \\
\text{subject to} \quad & \sum_{e \in \delta(v)} x_e = 2 && \forall v \in V \\
& \sum_{e \in \delta(S)} x_e \geq 2 && \forall S \subset V, S \neq \emptyset \\
& x_e \in \{0, 1\} && \forall e \in E
\end{aligned}
$$

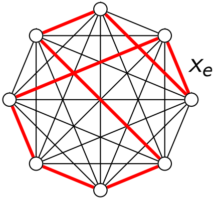
# TSP – Integer Programming Formulation

**Given**

- Complete graph $G = (V, E)$
- for each $e \in E$ a distance $d_e > 0$

**Binary variables**

- $x_e = 1$ if edge $e$ is used




$$
\begin{array}{llll}
\min & \sum_{e \in E} d_e \, x_e & & \\
\text{subject to} & \sum_{e \in \delta(v)} x_e = 2 & \forall v \in V & \text{Degrees} \\
& \sum_{e \in \delta(S)} x_e \geq 2 & \forall S \subset V, S \neq \emptyset & \\
& x_e \in \{0, 1\} & \forall e \in E &
\end{array}
$$

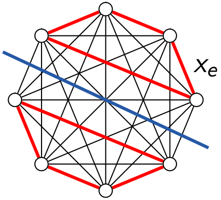
# TSP – Integer Programming Formulation

Given

- Complete graph $G = (V, E)$
- for each $e \in E$ a distance $d_e > 0$

Binary variables

- $x_e = 1$ if edge $e$ is used




$K_8$

$$\begin{aligned}
\min \quad & \sum_{e \in E} d_e \, x_e \\
\text{subject to} \quad & \sum_{e \in \delta(v)} x_e = 2 && \forall v \in V \\
& \sum_{e \in \delta(S)} x_e \geq 2 && \forall S \subset V, S \neq \varnothing \quad \text{NoSubtours} \\
& x_e \in \{0, 1\} && \forall e \in E
\end{aligned}$$

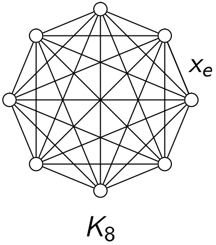
# TSP – Integer Programming Formulation

**Given**

- ▷ Complete graph $G = (V, E)$
- ▷ for each $e \in E$ a distance $d_e > 0$

**Binary variables**

- ▷ $x_e = 1$ if edge $e$ is used




$$\min \sum_{e \in E} d_e x_e \qquad \text{Distance}$$

$$\text{subject to} \quad \sum_{e \in \delta(v)} x_e = 2 \qquad \forall v \in V$$

$$\sum_{e \in \delta(S)} x_e \geq 2 \qquad \forall S \subset V, S \neq \varnothing$$

$$x_e \in \{0, 1\} \qquad \forall e \in E$$

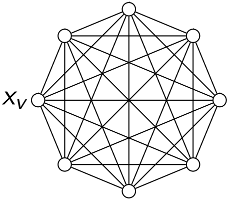
# TSP – Constraint Programming Formulation

Given

- Complete graph $G = (V, E)$
- for each $e \in E$ a distance $d_e > 0$

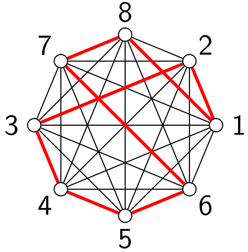
Integer variables

- $x_v$ position of $v \in V$ in tour




$K_8$

# TSP – Constraint Programming Formulation




**Given**

- ▷ Complete graph $G = (V, E)$
- ▷ for each $e \in E$ a distance $d_e > 0$

**Integer variables**

- ▷ $x_v$ position of $v \in V$ in tour

$$\begin{aligned} \min \quad & \text{length}(x_1, \ldots, x_n) & \\ \text{subject to} \quad & \text{alldifferent}(x_1, \ldots, x_n) & \\ & x_v \in \{1, \ldots, n\} & \forall v \in V \end{aligned}$$

# The Heart of the CIP Concept

## Mixed Integer Program

- Linear objective function
- Linear constraints
- Real and integer variables

## Constraint Program

- Arbitrary objective function
- Arbitrary constraints
- Arbitrary (discrete) variables

## Constraint Integer Program

- Linear objective function
- Arbitrary constraints
- Real and integer variables
- After fixing all integer variables: CIP becomes an LP

Remark:

- Arbitrary objective or variables modeled by constraints

→ For each type of constraint, one constraint handler is responsible.

# Constraint Integer Program (CIP)

## Characteristics

Objective function:

- linear function

Feasible region:

- described by arbitrary constraints
- after fixing all integer vars: CIP becomes an linear program (LP)

Variable domains:

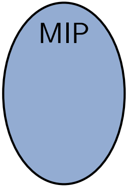
- real or integer values

$$
\begin{array}{llll}
\min & \sum_{e \in E} d_e\, x_e & & \\
s.t. & \sum_{e \in \delta(v)} x_e & = 2 & \forall\, v \in V \\
 & \text{nosubtour}(x) & & \\
 & x_e & \in \{0,1\} & \forall\, e \in E
\end{array}
$$

(CIP formulation of TSP)

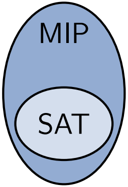
Single nosubtour constraint rules out subtours (e.g. by domain propagation). It may also separate subtour elimination inequalities.

- Mixed Integer Programs



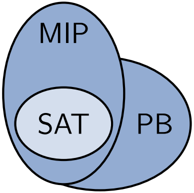
# Constraint Integer Programming

- Mixed Integer Programs
- Satisfiability



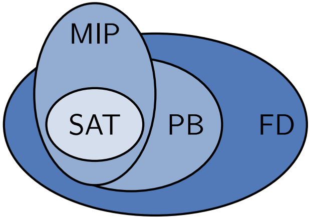
# Constraint Integer Programming

- Mixed Integer Programs
- Satisfiability
- Pseudo-Boolean



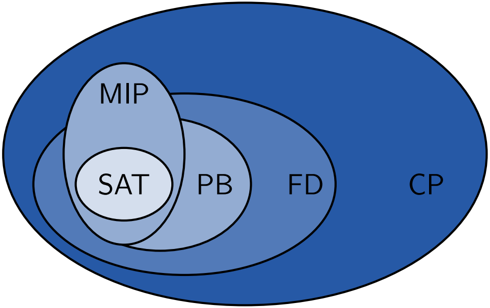
# Constraint Integer Programming

- Mixed Integer Programs
- Satisfiability
- Pseudo-Boolean
- Finite Domain



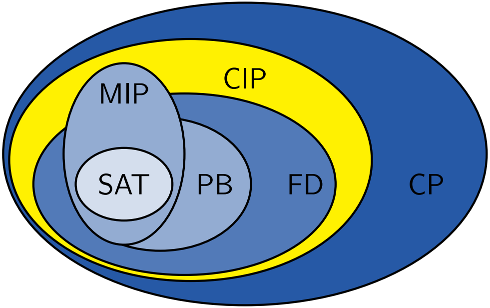
# Constraint Integer Programming

- ▷ Mixed Integer Programs
- ▷ Satisfiability
- ▷ Pseudo-Boolean
- ▷ Finite Domain
- ▷ Constraint Programming



# Constraint Integer Programming

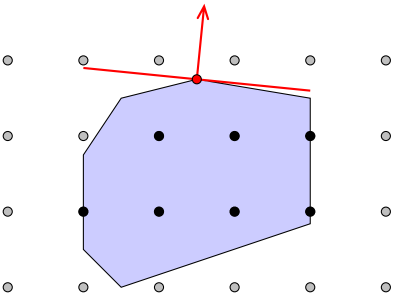
- Mixed Integer Programs
- Satisfiability
- Pseudo-Boolean
- Finite Domain
- Constraint Programming
- Constraint Integer Programming



# MIP Solving Techniques

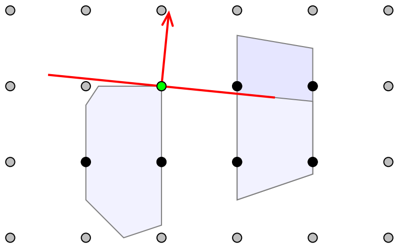
## Combination of ...

- LP relaxation
- branch-and-bound
- cutting planes



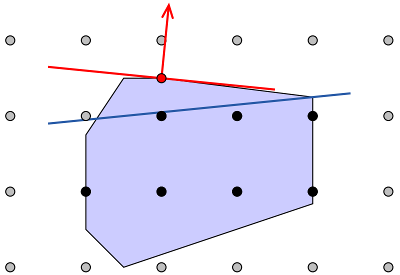
### Combination of ...

- LP relaxation
- branch-and-bound
- cutting planes



## Combination of ...

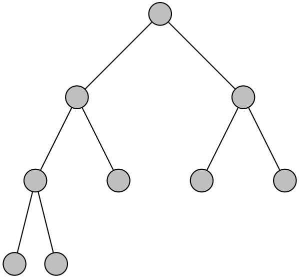
- LP relaxation
- branch-and-bound
- cutting planes



# CP Solving Techniques

## Combination of ...

- branch-and-bound
- domain propagation

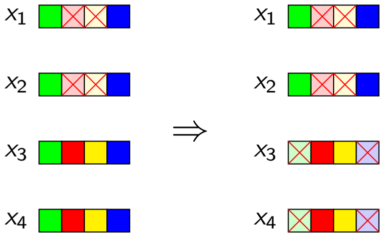


### Combination of ...

- branch-and-bound
- domain propagation

### Alldifferent constraint:

$\{x_1, x_2, x_3, x_4\} \in \{1, 2, 3, 4\}$
pairwise different.

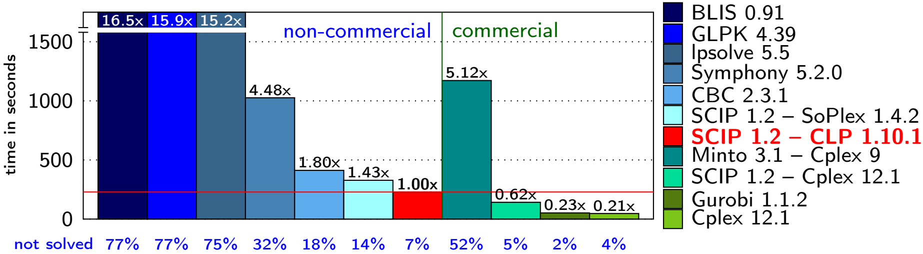


# SCIP Facts

- Approx. 275.640 lines of C code
  $\rightarrow$ 18% documentation
  $\rightarrow$ 20% assertions
- 7 examples illustrating the use of SCIP
- HowTos: each plugin type, debugging, automatic testing, ...
- C++ wrapper classes
- 7 interfaces to external linear programming solvers
  $\rightarrow$ CLP, CPLEX, Gurobi, Mosek, QSopt, SoPlex, XPRESS
- 6 different input formats
  $\rightarrow$ cnf, flatzinc, lp, mps, opb (pseudo-boolean), zimpl
- 876 parameters to play with
- Part of the ZIB Optimization Suite `http://zibopt.zib.de`

# Performance of SCIP

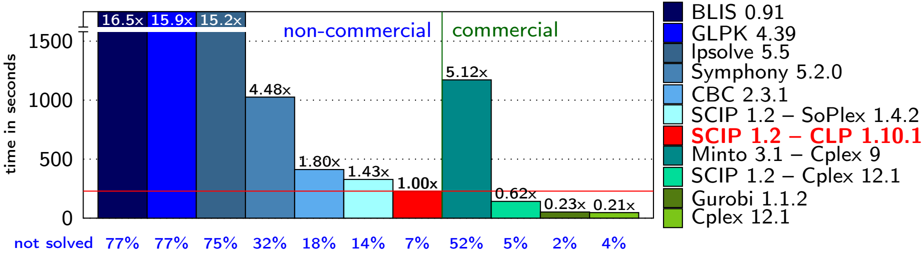
▷ One of the fastest noncommercial MIP solver




results taken from H. Mittelmann (09/17/2009)

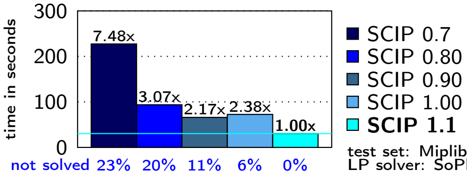
# Performance of SCIP

▷ One of the fastest noncommercial MIP solver




results taken from H. Mittelmann (09/17/2009)

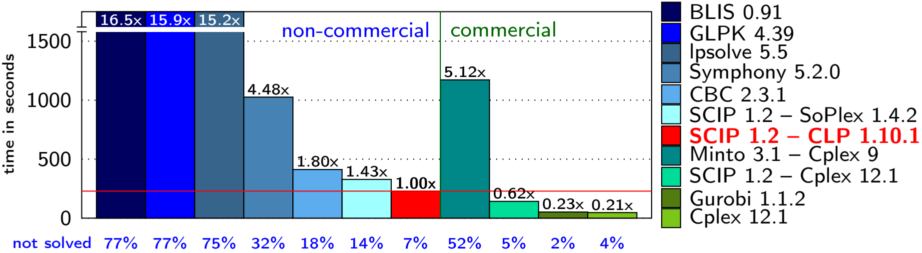
▷ Performance development




test set: Miplib2003
LP solver: SoPlex

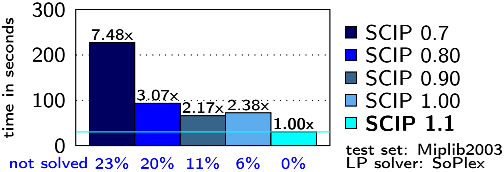
# Performance of SCIP

- One of the fastest noncommercial MIP solver




results taken from H. Mittelmann (09/17/2009)

- Performance development




- Winner of the "Pseudo-Boolean Evaluation 2009"

  `http://www.cril.univ-artois.fr/PB09/`

# ZIP Optimization Suite = SCIP + SoPlex + ZIMPL

Tool for generating and solving constraint integer programs

## ZIMPL

- A mixed integer programming modeling language
- Easily generating linear programs and mixed integer programs

## SCIP

- A mixed integer programming solver and constraint programming
- ZIMPL models can directly be loaded into SCIP and solved

## SoPlex

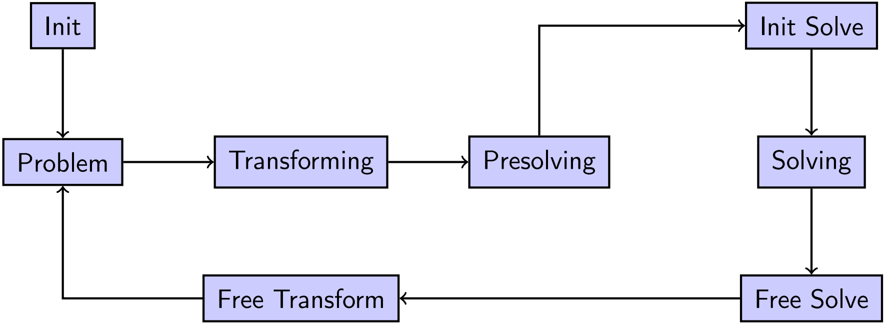
- A linear programming (LP) solver
- Solution process SCIP may use SoPlex as underlying LP solver

All three tools are available in source code and free for academic use

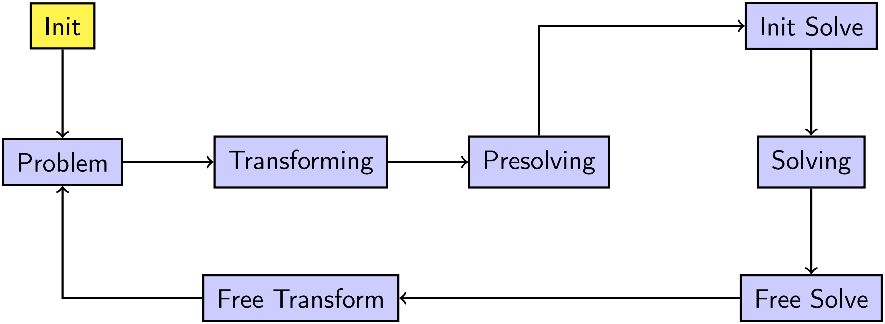
# Important Aspects

- SCIP is constraint based
  - Advantage: flexibility
  - Disadvantage: limited global view
- A constraint knows its variables, but a variable does not know the constraints it appears in.
- A single constraint may represent a whole set of inequalities, not only a single one.
- From the constraint programming perspective:
  - LP-relaxation is only an add-on!
  - Many constraints do not separate inequalities at all.

# Operational Stages



# Operational Stages




- Basic data structures are allocated and initialized.
- User includes required plugins (or just takes default plugins).

# SCIP Interface

Main SCIP interface: plugins.

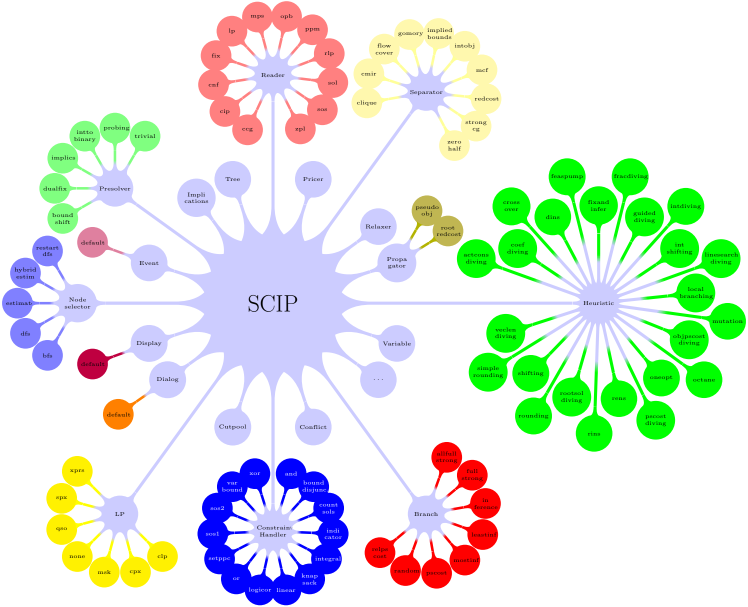
- Perform all problem specific actions.
- Each step calls user defined plugins.
- SCIP knows of plugins through "include" functions.
- Plugins may have private data.
- User defined callback functions (virtual functions in C++-interface).
- Yields modular structure.
- The MIP solver is realized through plugins.

Everything SCIP knows, it knows through plugins.

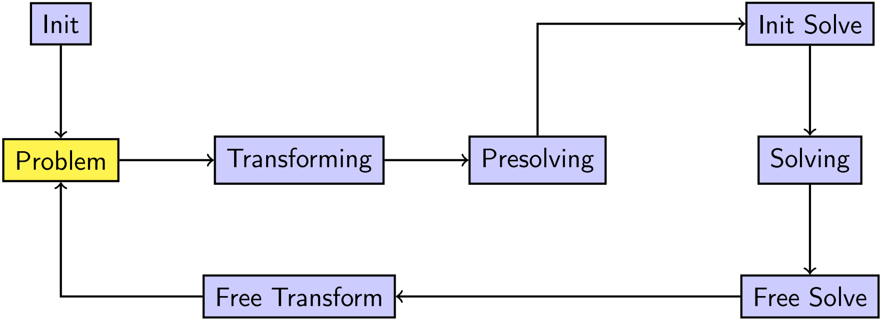
# Types of Plugins

- Constraint handler: assures feasibility, strengthens formulation
- Separator: adds cuts, improves dual bound
- Pricer: allows dynamic generation of variables
- Heuristic: searches solutions, improves primal bound
- Branching rule: how to divide the problem?
- Node selection: which subproblem should be regarded next?
- Presolver: simplifies the problem in advance, strengthens structure
- Propagator: simplifies problem, improves dual bound locally
- Reader: reads problems from different formats
- Event handler: catches events (e.g., bound changes, new solutions)
- Display: allows modification of output
- ...

# SCIP Structures

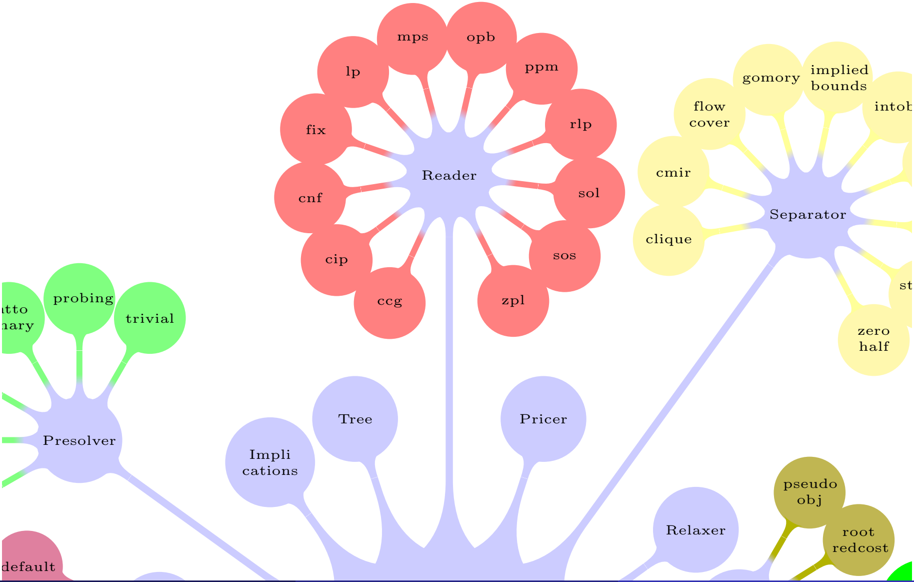


# Problem Specification

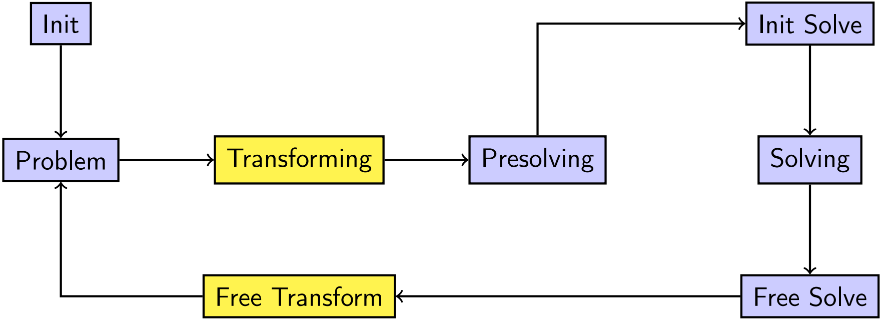



- User creates and modifies the original problem instance.
- Problem creation is usually done in file readers (`SCIPreadProb()`).

# SCIP Structures

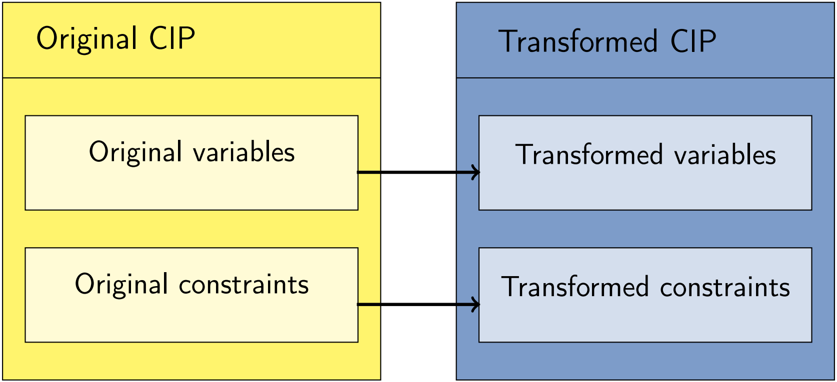


# Transformation



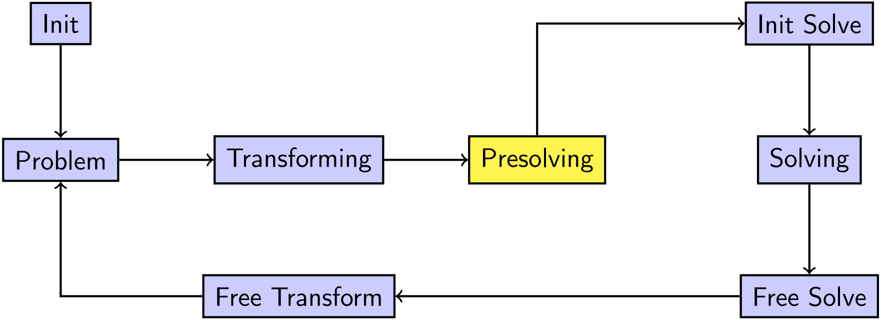

- ▷ Creates a working copy of the original problem.

# Original and Transformed Problem

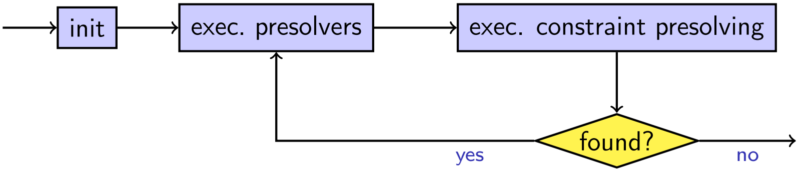



- data are copied into separate memory area
- presolving and solving operate on transformed problem
- original data can only be modified in problem modification stage

# Presolving

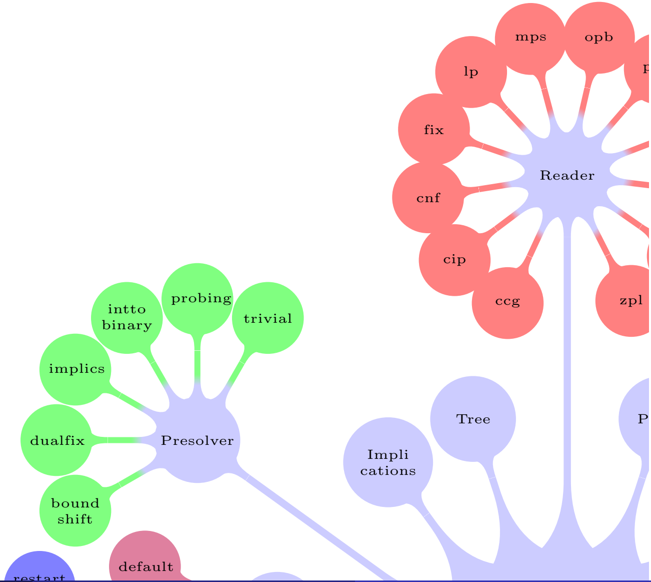

Init
Problem
Transforming
Presolving
Init Solve
Solving
Free Solve
Free Transform

# Presolving

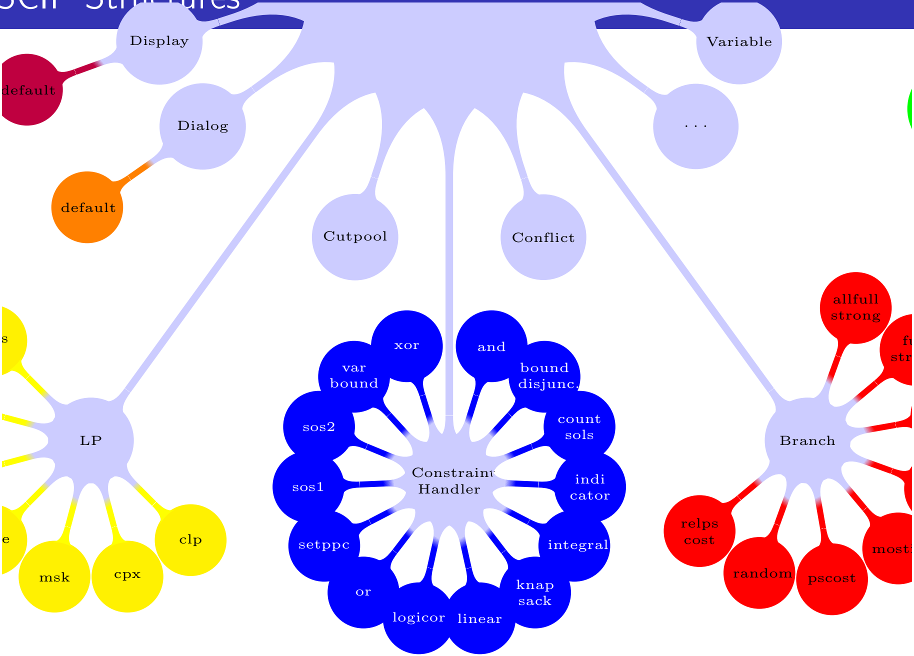



- ▷ Presolvers provide global presolving (e.g. dual fixing)
- ▷ Constraint Handlers provide constraint specific presolving, e.g.:
  - Domain tightening,
  - Coefficient modification,
  - Deletion of redundant constraints,
  - Constraint upgrading.
  - ...

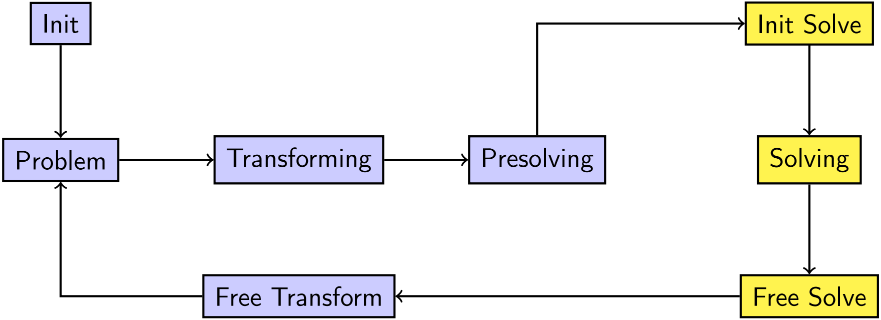
# SCIP Structures



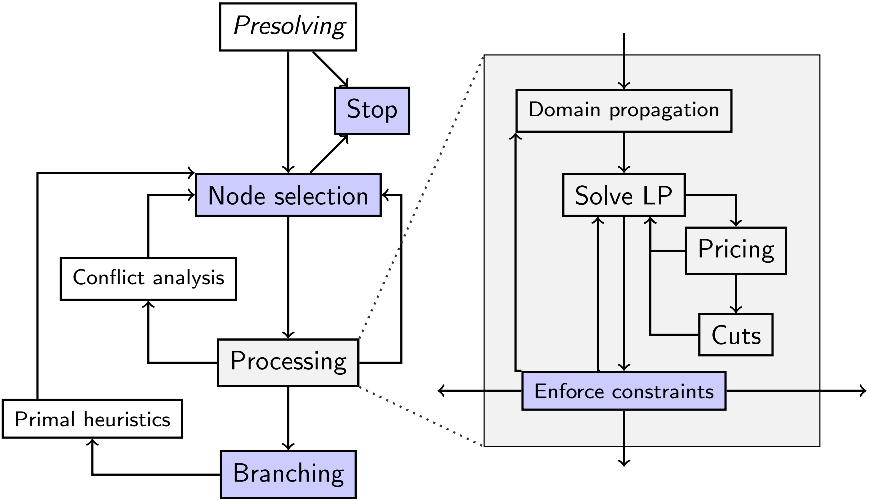
# SCIP Structures



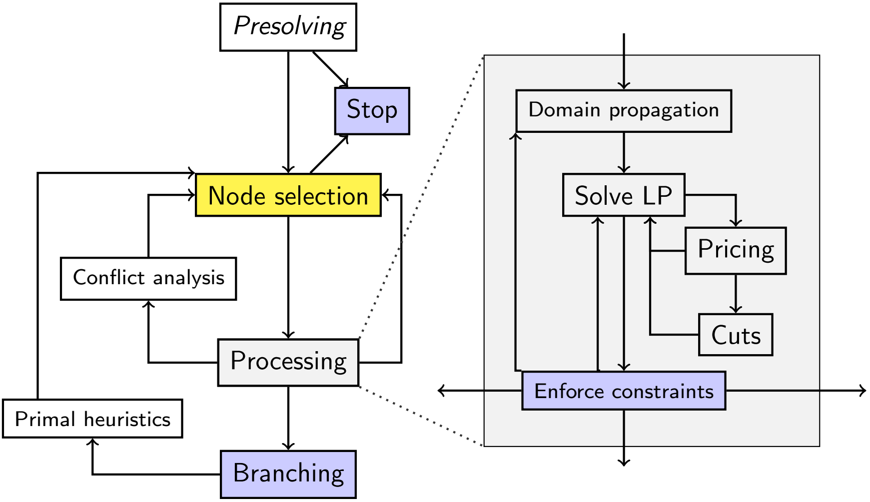
# Solving


Init
Problem
Transforming
Presolving
Init Solve
Solving
Free Solve
Free Transform

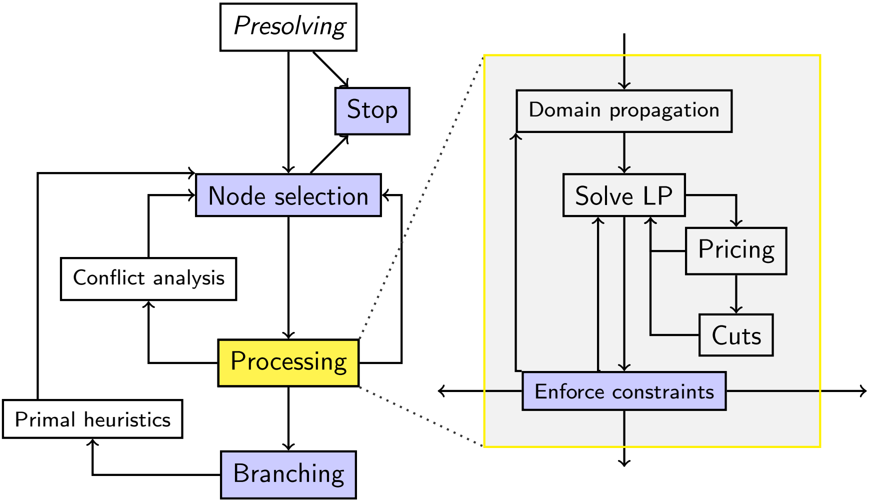
# Flow Chart SCIP



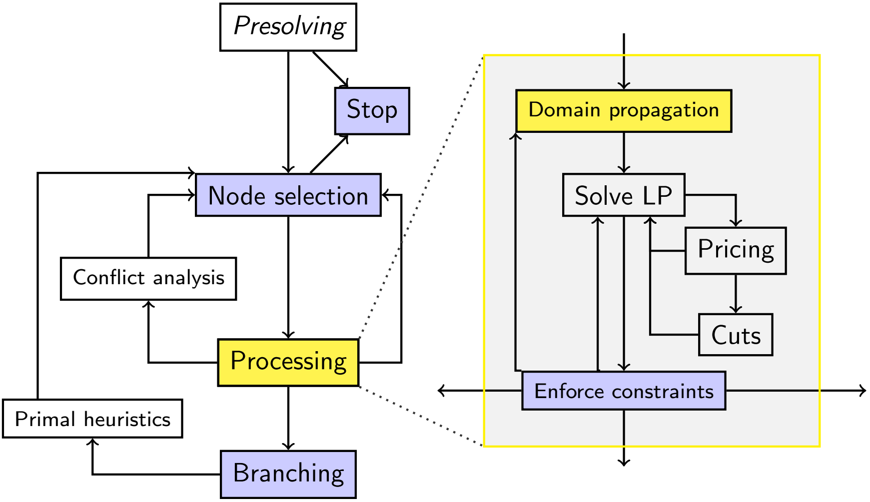
# Flow Chart SCIP



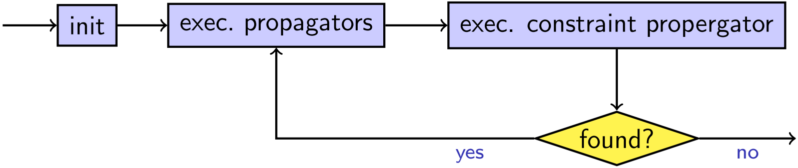
# Flow Chart SCIP



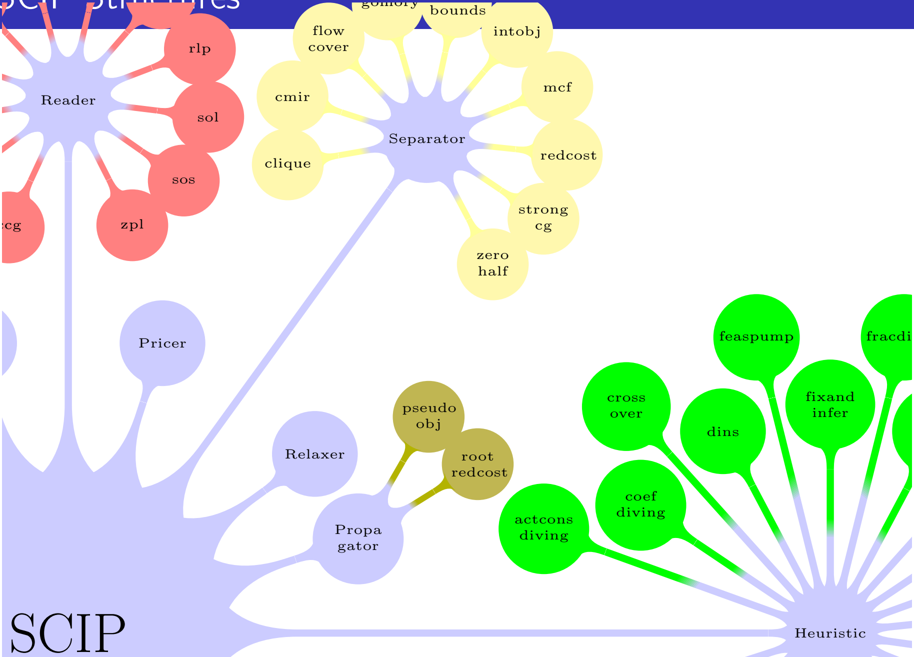
# Flow Chart SCIP



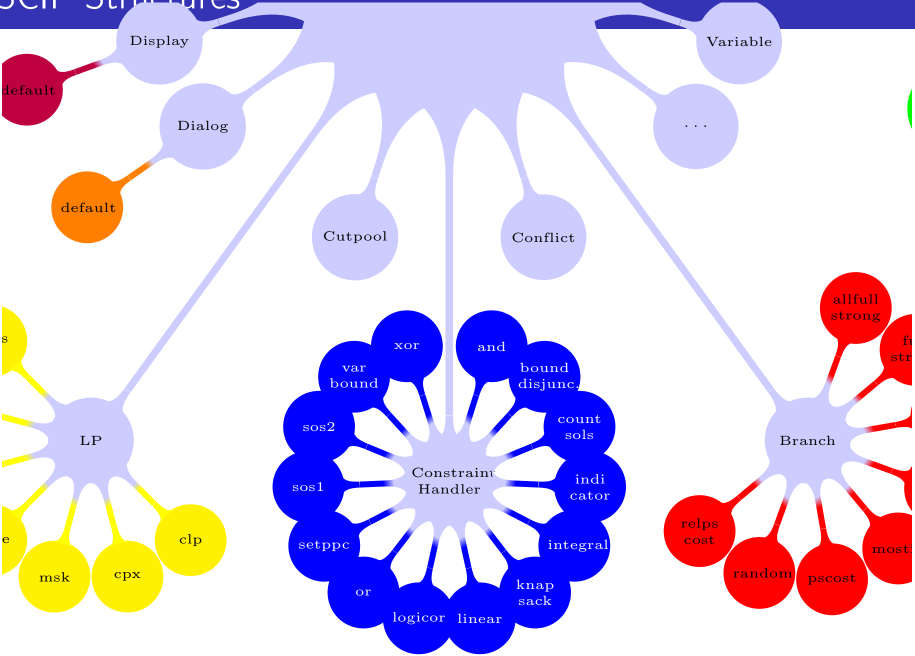
# Domain Propagation (Node Presolving)



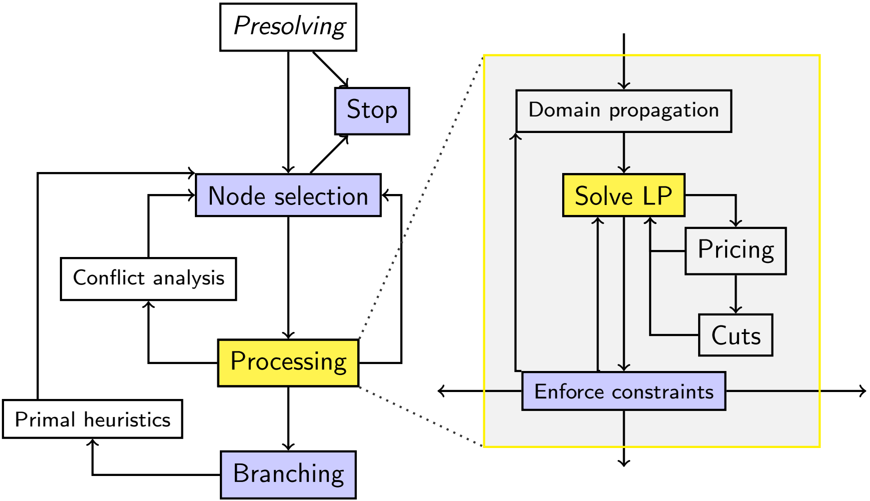

- ▷ Propagatore solvers provide local presolving
- ▷ Constraint Handlers provide constraint specific propagation, e.g.:
  - Domain tightening,
  - Coefficient modification,
  - Deletion of redundant constraints,
  - Constraint upgrading.
  - ...


Reader
rlp
sol
sos
zpl
Separator
bounds
flow cover
intobj
mcf
cmir
redcost
clique
strong cg
zero half
Pricer
Relaxer
Propa gator
pseudo obj
root redcost
Heuristic
feaspump
cross over
dins
fixand infer
coef diving
actcons diving
SCIP

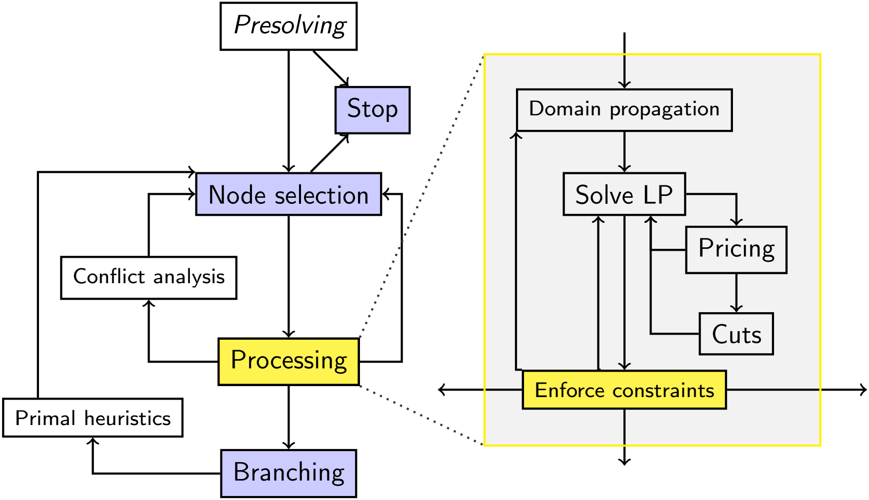
# SCIP Structures



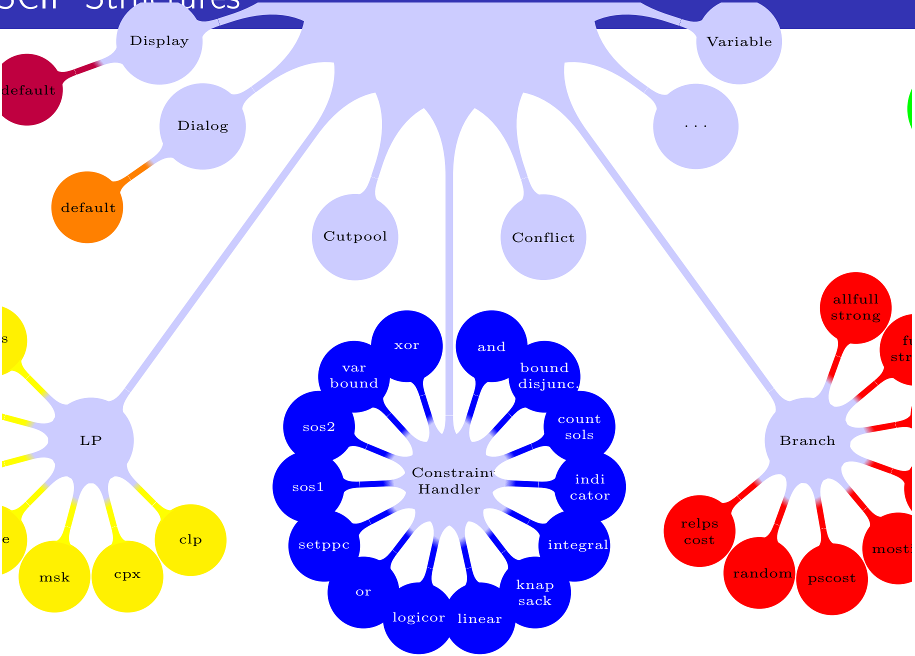
# Flow Chart SCIP



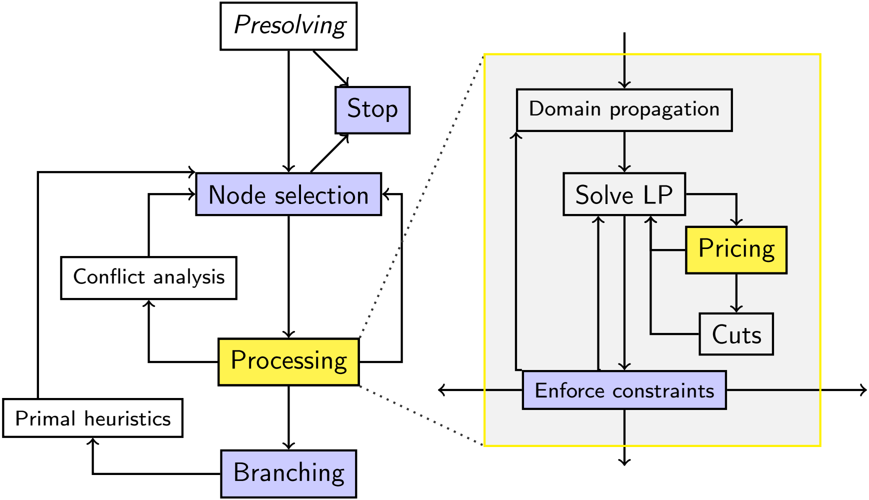
# Flow Chart SCIP



# SCIP Structures

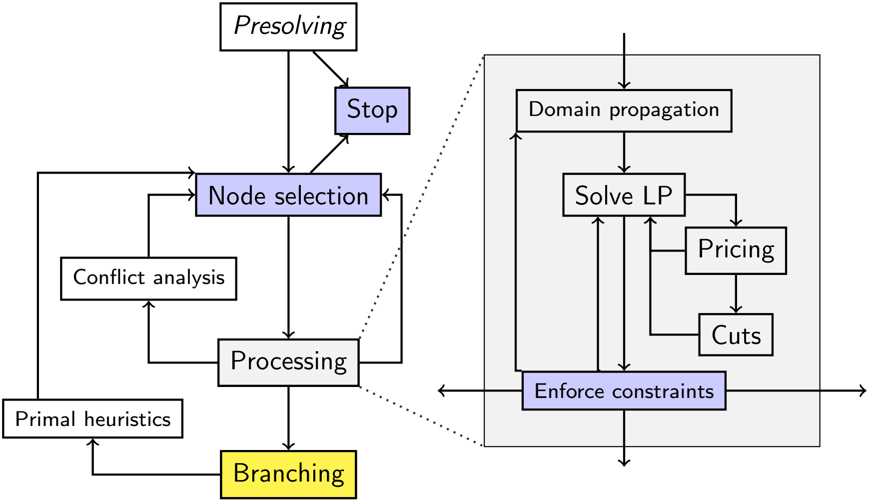


# Flow Chart SCIP



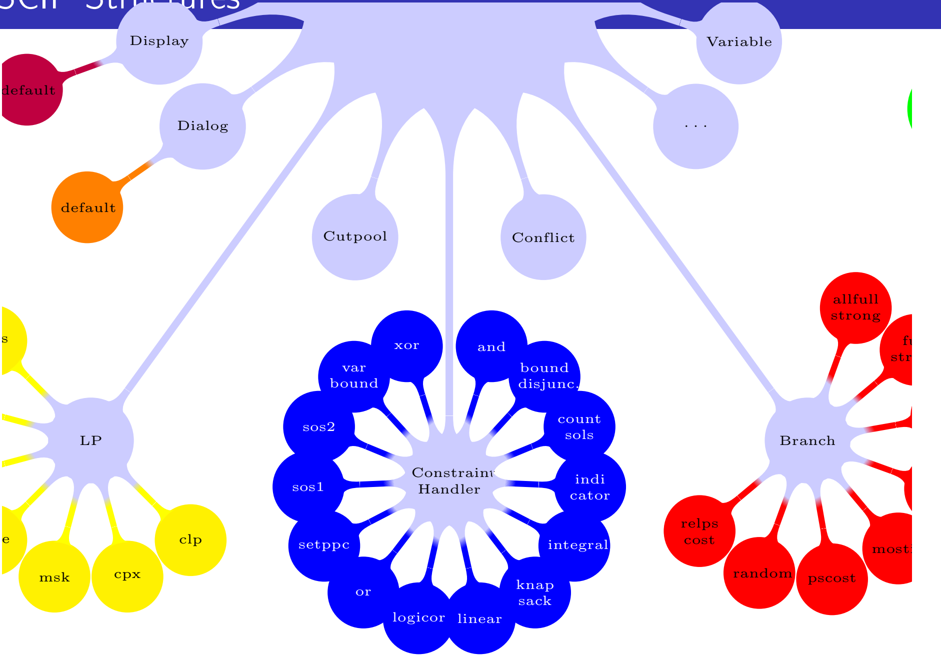

- Pricing: allows dynamic generation of variables

# Flow Chart SCIP

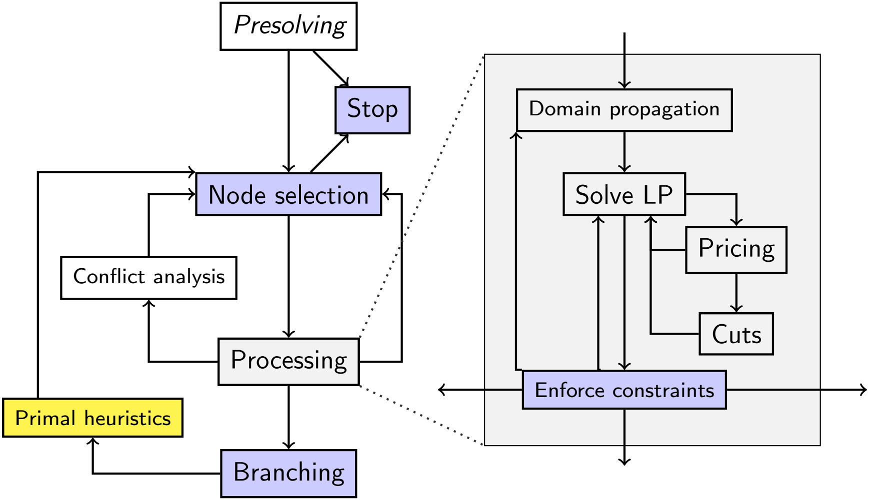



- Creates a branching with the infeasible solution no longer being feasible in the relaxations of the child nodes.

# SCIP Structures

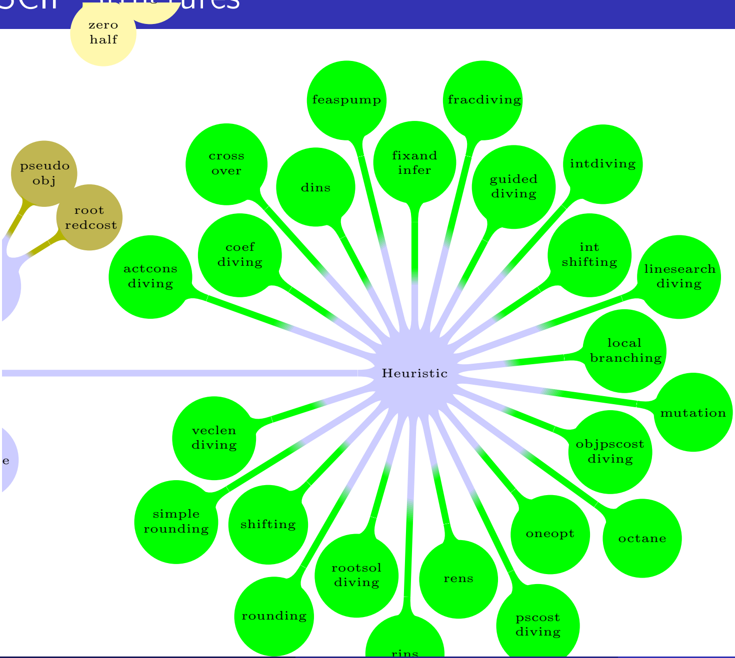


# Flow Chart SCIP

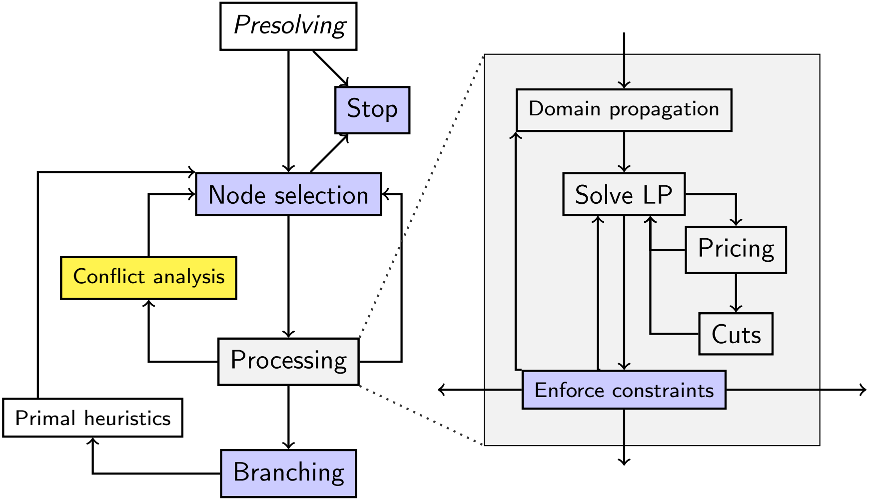



- Primal heuristics try to find feasible solutions (in addition to feasible LP solutions).

# SCIP Structures

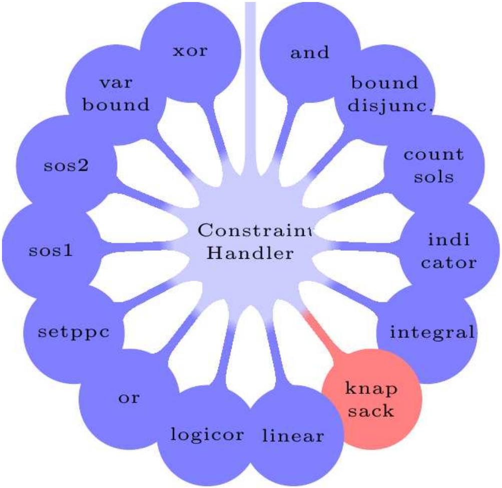


# Flow Chart SCIP




- Conflict analysis learns from infeasible subproblems

# Default Plugins



# Special Linear Constraints

$$
\begin{aligned}
\min \quad & x_1 + x_2 + x_3 + x_4 + x_5 + y_1 + y_2 \\
\text{s.t.} \quad & y_1 \le 3 + 4\,x_1 \\
& y_2 \le 4 + 5\,x_2 \\
& 3\,x_1 + 2\,x_2 + 4\,x_3 \le 8 \\
& x_2 + x_3 + x_4 = 1 \\
& 3\,x_1 + 4.5\,x_2 + 3.2\,x_5 + 0.8\,y_1 + 1.2\,y_2 \ge 4 \\
& x_1,\, x_2,\, x_3,\, x_4 \in \{0,1\} \\
& x_5 \in \mathbb{Z}_+ \\
& y_1,\, y_2 \in \mathbb{R}_+
\end{aligned}
$$

```
presolved problem has 7 variables (4 bin, 1 int, 0 impl, 2 cont) and 5
      2 constraints of type <varbound>
      1 constraints of type <knapsack>
      1 constraints of type <setppc>
      1 constraints of type <linear>
```

# Special Linear Constraints

$$
\begin{aligned}
\min \quad & x_1 + x_2 + x_3 + x_4 + x_5 + y_1 + y_2 \\
\text{s.t.} \quad & y_1 \le 3 + 4\,x_1 \\
& y_2 \le 4 + 5\,x_2 \\
& 3\,x_1 + 2\,x_2 + 4\,x_3 \le 8 \\
& x_2 + x_3 + x_4 = 1 \\
& 3\,x_1 + 4.5\,x_2 + 3.2\,x_5 + 0.8\,y_1 + 1.2\,y_2 \ge 4 \\
& x_1,\, x_2,\, x_3,\, x_4 \in \{0,1\} \\
& x_5 \in \mathbb{Z}_+ \\
& y_1,\, y_2 \in \mathbb{R}_+
\end{aligned}
$$

```
presolved problem has 7 variables (4 bin, 1 int, 0 impl, 2 cont) and 5
      2 constraints of type <varbound>
      1 constraints of type <knapsack>
      1 constraints of type <setppc>
      1 constraints of type <linear>
```

# Special Linear Constraints

$$
\begin{aligned}
\min \quad & x_1 + x_2 + x_3 + x_4 + x_5 + y_1 + y_2 \\
\text{s.t.} \quad & y_1 \le 3 + 4\,x_1 \\
& y_2 \le 4 + 5\,x_2 \\
& \color{red}{3\,x_1 + 2\,x_2 + 4\,x_3 \le 8} \\
& x_2 + x_3 + x_4 = 1 \\
& 3\,x_1 + 4.5\,x_2 + 3.2\,x_5 + 0.8\,y_1 + 1.2\,y_2 \ge 4 \\
& \color{red}{x_1,\, x_2,\, x_3,\, x_4 \in \{0,1\}} \\
& x_5 \in \mathbb{Z}_+ \\
& y_1,\, y_2 \in \mathbb{R}_+
\end{aligned}
$$

```
presolved problem has 7 variables (4 bin, 1 int, 0 impl, 2 cont) and 5
      2 constraints of type <varbound>
      1 constraints of type <knapsack>
      1 constraints of type <setppc>
      1 constraints of type <linear>
```

# Special Linear Constraints

$$
\begin{aligned}
\min \quad & x_1 + x_2 + x_3 + x_4 + x_5 + y_1 + y_2 \\
\text{s.t.} \quad & y_1 \le 3 + 4\,x_1 \\
& y_2 \le 4 + 5\,x_2 \\
& 3\,x_1 + 2\,x_2 + 4\,x_3 \le 8 \\
& x_2 + x_3 + x_4 = 1 \\
& 3\,x_1 + 4.5\,x_2 + 3.2\,x_5 + 0.8\,y_1 + 1.2\,y_2 \ge 4 \\
& x_1,\, x_2,\, x_3,\, x_4 \in \{0,1\} \\
& x_5 \in \mathbb{Z}_+ \\
& y_1,\, y_2 \in \mathbb{R}_+
\end{aligned}
$$

```
presolved problem has 7 variables (4 bin, 1 int, 0 impl, 2 cont) and 5
      2 constraints of type <varbound>
      1 constraints of type <knapsack>
      1 constraints of type <setppc>
      1 constraints of type <linear>
```

# Special Linear Constraints

$$
\begin{aligned}
\min \quad & x_1 + x_2 + x_3 + x_4 + x_5 + y_1 + y_2 \\
\text{s.t.} \quad & y_1 \le 3 + 4\,x_1 \\
& y_2 \le 4 + 5\,x_2 \\
& 3\,x_1 + 2\,x_2 + 4\,x_3 \le 8 \\
& x_2 + x_3 + x_4 = 1 \\
& 3\,x_1 + 4.5\,x_2 + 3.2\,x_5 + 0.8\,y_1 + 1.2\,y_2 \ge 4 \\
& x_1,\, x_2,\, x_3,\, x_4 \in \{0, 1\} \\
& x_5 \in \mathbb{Z}_+ \\
& y_1,\, y_2 \in \mathbb{R}_+
\end{aligned}
$$

```
presolved problem has 7 variables (4 bin, 1 int, 0 impl, 2 cont) and 5
      2 constraints of type <varbound>
      1 constraints of type <knapsack>
      1 constraints of type <setppc>
      1 constraints of type <linear>
```

# Knapsack Constraints

Feasible region of 0-1 knapsack problem:

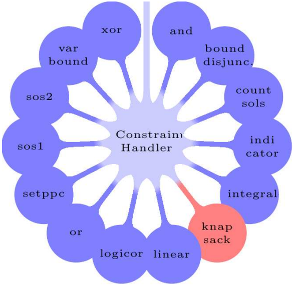
$$\{\ x \in \{0,1\}^{|N|}\ :\ \sum_{j \in N} a_j x_j \le b\ \}$$

- weights of the variables: $a_j \in \mathbb{Z}_+$ for all $j \in N$
- capacity of the knapsack: $b \in \mathbb{Z}_+$

# Ingredients of a Constraint Handler

## Callback Methods

- ▷ Fundamental:
  - CONSLOCK
  - CONSCHECK
  - CONSENFOLP, CONSENFOPS
- ▷ Additional:
  - CONSINIT..., CONSEXIT...
  - CONSSEPALP, CONSSEPASOL
  - CONSPROP, CONSRESPROP, CONSPRESOL
  - CONSACTIVE, CONSDEACTIVE, CONSENABLE, CONSDISABLE, CONSTRANS, CONSDELETE, CONSFREE
  - CONSPRINT, CONSCOPY, CONSPARSE




## Further Ingredients

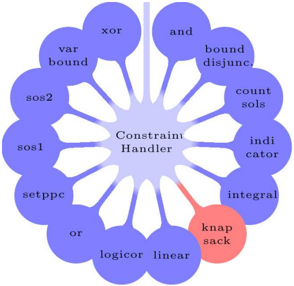
- ▷ Private data
- ▷ Interface methods
- ▷ Properties/Parameters

# Ingredients of a Constraint Handler

## Callback Methods

- ▷ Fundamental:
  - CONSLOCK
  - CONSCHECK
  - CONSENFOLP, CONSENFOPS

- ▷ Additional:
  - CONSINIT..., CONSEXIT...
  - CONSSEPALP, CONSSEPASOL
  - CONSPROP, CONSRESPROP, CONSPRESOL
  - CONSACTIVE, CONSDEACTIVE, CONSENABLE, CONSDISABLE, CONSTRANS, CONSDELETE, CONSFREE
  - CONSPRINT, CONSCOPY, CONSPARSE




## Further Ingredients

- ▷ Private data
- ▷ Interface methods
- ▷ Properties/Parameters

# Private Data

Constraint data: information needed to define single constraint

```
struct SCIP_ConsData
{
  SCIP_VAR**      vars;      // variables in knapsack
  int             nvars;     // number of variables
  SCIP_Longint*   weights;   // weights of variables
  SCIP_Longint    capacity;  // capacity of knapsack
};
```

Constraint handler data: information belonging to constraint handler itself

```
struct SCIP_ConshdlrData
{
  int   maxrounds;     // max nr. of sepa rounds per node
  int   maxsepacuts;   // max nr. of cuts per sepa round
};
```

# Interface Methods

## Creating a single knapsack constraint.

```
SCIP_RETCODE SCIPcreateConsKnapsack(
  SCIP*          scip,       // SCIP data structure
  SCIP_CONS**    cons,       // pointer to hold created cons
  const char*    name,       // name of constraint
  SCIP_VAR**     vars,       // array with variables
  int            nvars,      // number of variables in knapsack
  SCIP_Longint*  weights,    // array with weights
  SCIP_Longint   capacity,   // capacity of knapsack
  SCIP_Bool      separate,   // should constraint be separated?
  ...
)
{
  SCIP_CONSDATA* consdata;
  SCIP_CALL( consdataCreate(scip, &consdata, nvars, vars,
                            weights, capacity) );
  SCIP_CALL( SCIPcreateCons(scip, cons, name, conshdlr,
                            consdata, separate, ...) );
  ...
}
```

# Interface Methods

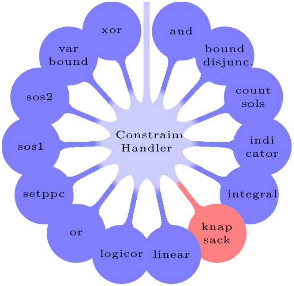
Including the knapsack constraint handler.

```
SCIP_RETCODE SCIPincludeConshdlrKnapsack(
  SCIP*           scip        // SCIP data structure
)
{
  SCIP_CONSHDLRDATA* conshdlrdata;

  SCIP_CALL( conshdlrdataCreate(scip, &conshdlrdata) );

  SCIP_CALL( SCIPincludeConshdlr(scip, CONSHDLR_CHECKPRIORITY,
             consCheckKnapsack, ..., conshdlrdata) );
  ...
}
```

# Ingredients of a Constraint Handler

## Callback Methods

- ▷ Fundamental:
  - CONSLOCK
  - CONSCHECK
  - CONSENFOLP, CONSENFOPS
- ▷ Additional:
  - CONSINIT..., CONSEXIT...
  - CONSSEPALP, CONSSEPASOL
  - CONSPROP, CONSRESPROP, CONSPRESOL
  - CONSACTIVE, CONSDEACTIVE, CONSENABLE, CONSDISABLE, CONSTRANS, CONSDELETE, CONSFREE
  - CONSPRINT, CONSCOPY, CONSPARSE




## Further Ingredients

- ▷ Private data
- ▷ Interface methods
- ▷ Properties/Parameters

# CONSLOCK

- Provides dual information for single constraints (useful for presolving, primal heuristics, ...)
- For each variable of a constraint, returns whether ...
  - increasing its value, and/or
  - decreasing its value

  may lead to a violation of the constraint

$$3\,x_1 - 5\,x_2 + 2\,x_3 \qquad \leq 7$$

increasing: $x_1$ and $x_3$

decreasing: $x_2$

# CONSCHECK

### Most important callback ...

- usually called by primal heuristics
- checks given solution for feasibility wrt all constraints of its type
- possible result values
  - SCIP_FEASIBLE
  - SCIP_INFEASIBLE

Given solution:

$$(x_1, x_2, x_3) = (1, 0, 1)$$

Knapsack constraints:

$$3x_1 + 6x_2 + 4x_3 \leq 8$$

$$2x_1 + \qquad + 2x_3 \leq 3$$

Result: SCIP_INFEASIBLE

# CONSENFOLP and CONSENFOPS

CONSENFOLP: checks LP solution for feasibility
CONSENFOPS: checks Pseudo solution for feasibility

**LP solution**

- solution of LP relaxation

$$\begin{aligned} \min \quad & x_1 - x_2 + x_3 \\ \text{s.t.} \quad & 3\,x_1 + 8\,x_2 + 4\,x_3 \leq 4 \\ & x_1,\, x_2,\, x_3 \in \{0,1\} \end{aligned}$$

LP solution: $(0, \frac{1}{2}, 0)$

**Pseudo Solution**

- solution of LP relaxation with only bound constraints
- used if LP solving disabled, or
- numerical difficulties occurred

$$\begin{aligned} \min \quad & x_1 - x_2 + x_3 \\ \text{s.t.} \quad & x_1,\, x_2,\, x_3 \in \{0,1\} \end{aligned}$$

Pseudo Solution: $(0, 1, 0)$

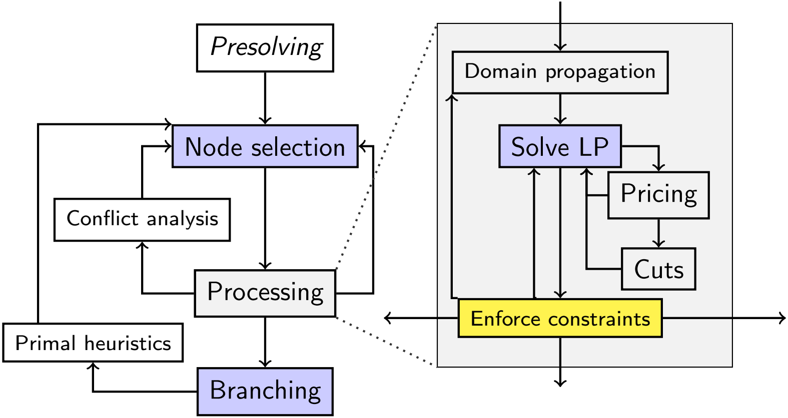
# CONSENFOLP and CONSENFOPS

LP solution may violate a constraint not contained in the relaxation.

> Enforcement callbacks are necessary for a correct implementation!

In addition, they can resolve an infeasibility by . . .

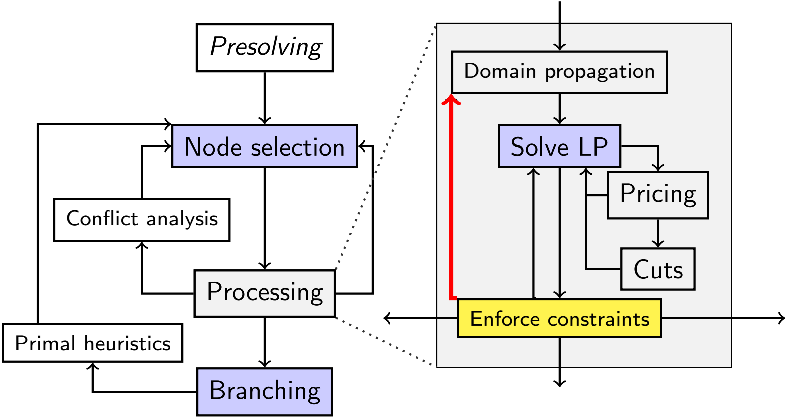
- reducing a variable's domain,
- separating a cutting plane (may use integrality),
- adding a (local) constraint,
- creating a branching,
- concluding that the subproblem is infeasible and can be cut off, or
- just saying "solution infeasible".




Enforcement result of constraint handler:

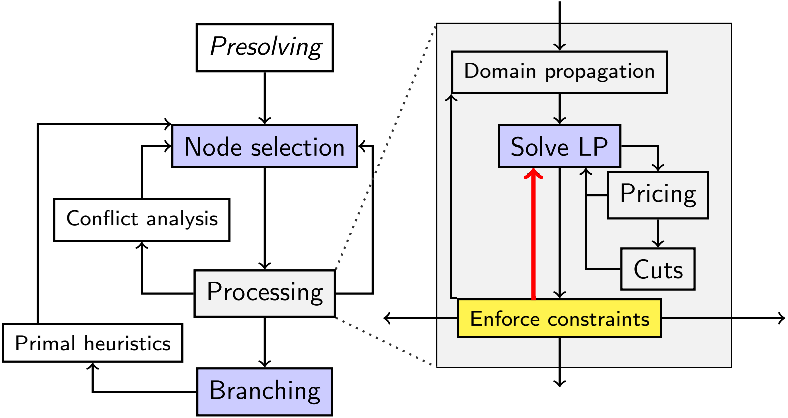
- reduced domain
- added constraint
- added cut
- branched
- cutoff
- infeasible
- feasible

# CONSENFOLP and CONSENFOPS



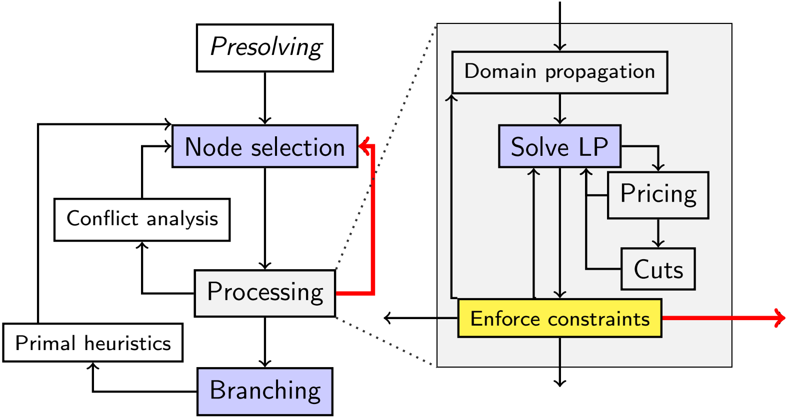

Enforcement result of constraint handler:

- reduced domain
- added constraint
- added cut
- branched
- cutoff
- infeasible
- feasible



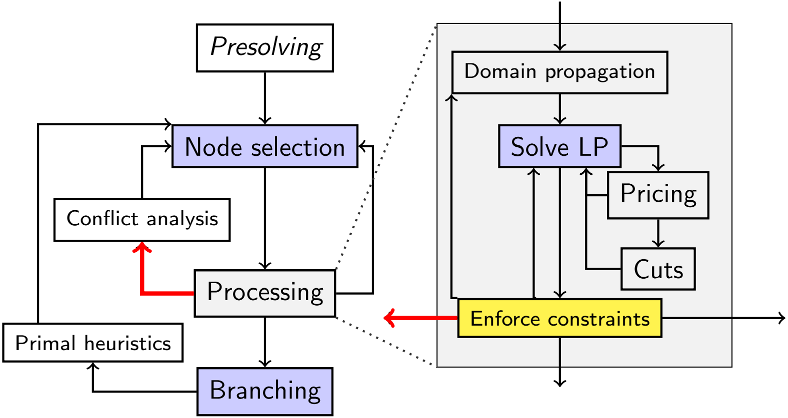

Enforcement result of constraint handler:

- reduced domain
- added constraint
- added cut
- branched
- cutoff
- infeasible
- feasible




Enforcement result of constraint handler:

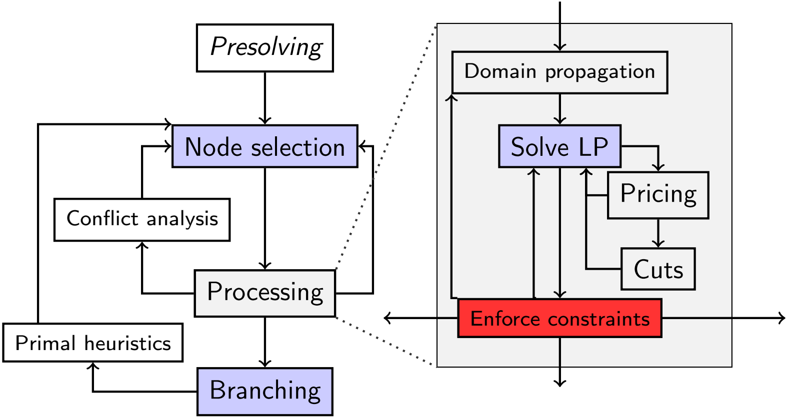
- reduced domain
- added constraint
- added cut
- branched
- cutoff
- infeasible
- feasible




Enforcement result of constraint handler:

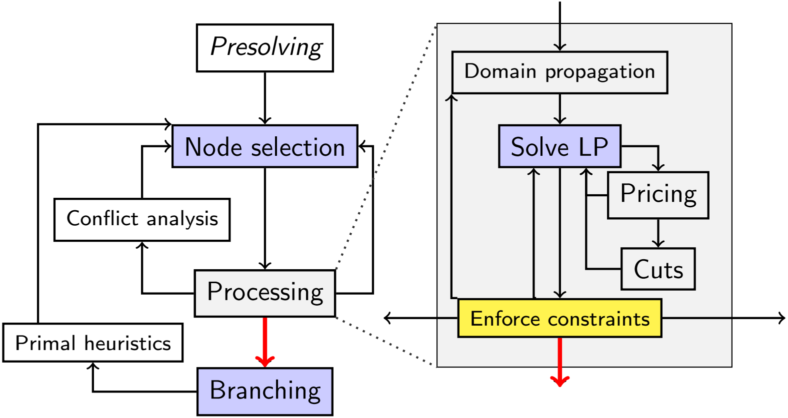
- reduced domain
- added constraint
- added cut
- branched
- cutoff
- infeasible
- feasible

# CONSENFOLP and CONSENFOPS



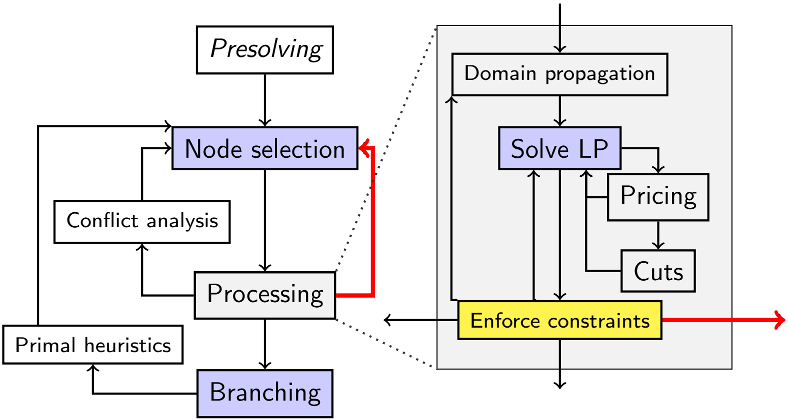

Enforcement result of constraint handler:

- reduced domain
- added constraint
- added cut
- branched
- cutoff
- infeasible
- feasible




Enforcement result of constraint handler:

- reduced domain
- added constraint
- added cut
- branched
- cutoff
- infeasible
- feasible



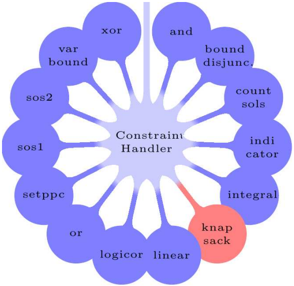

Enforcement result of constraint handler:

- reduced domain
- added constraint
- added cut
- branched
- cutoff
- infeasible
- feasible

# Ingredients of a Constraint Handler

## Callback Methods

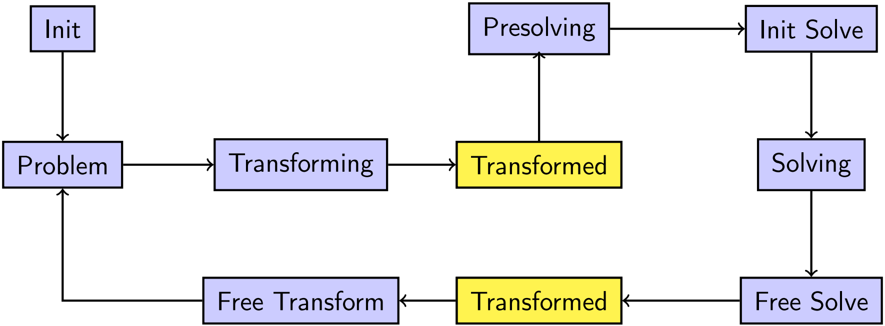
- **Fundamental:**
  - CONSLOCK
  - CONSCHECK
  - CONSENFOLP, CONSENFOPS
- **Additional:**
  - CONSINIT..., CONSEXIT...
  - CONSSEPALP, CONSSEPASOL
  - CONSPROP, CONSRESPROP, CONSPRESOL
  - CONSACTIVE, CONSDEACTIVE, CONSENABLE, CONSDISABLE, CONSTRANS, CONSDELETE, CONSFREE
  - CONSPRINT, CONSCOPY, CONSPARSE




## Further Ingredients

- Private data
- Interface methods
- Properties/Parameters

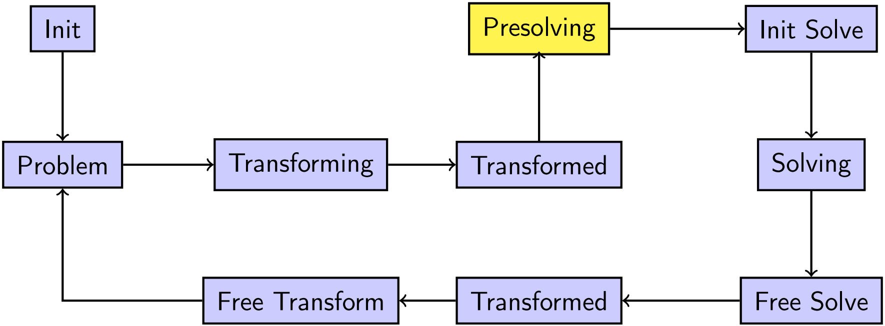
# CONSINIT..., CONSEXIT...




CONSINIT and CONSEXIT:

- called after problem was transformed /
  before transformed problem is freed
- initialize and free statistics in SCIP_ConshdlrData

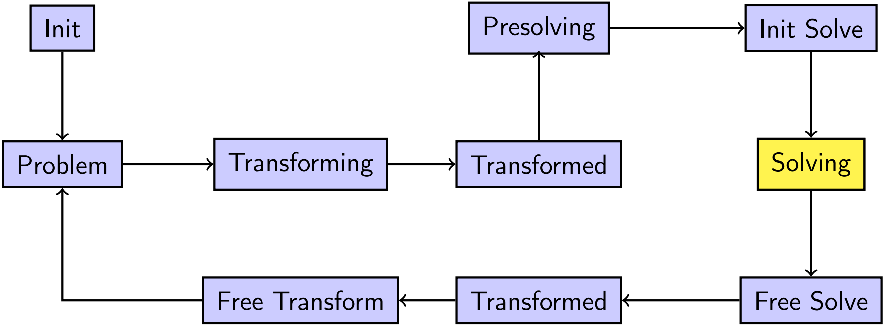
# CONSINIT..., CONSEXIT...




CONSINITPRE and CONSEXITPRE:

- called before presolving starts / after presolving is finished
- initialize and free presolving data

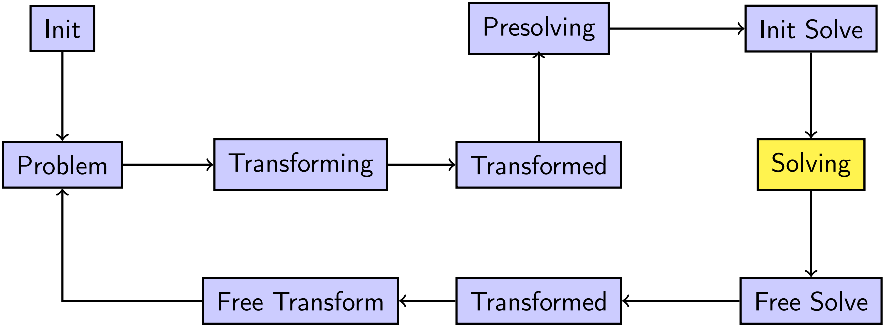
# CONSINIT..., CONSEXIT...




CONSINITSOL and CONSEXITSOL:

- called before branch-and-bound process starts /
  before branch-and-bound process is freed
- initialize and release branch-and-bound specific data

# CONSINIT..., CONSEXIT...




CONSINITLP:

- called before first LP relaxation is solved
- add linear relaxation of all "initial" constraints to the LP relaxation

# CONSSEPALP, CONSSEPASOL

Feasible region of 0-1 knapsack problem:

$$\{\ x \in \{0,1\}^{|N|}\ :\ \sum_{j \in N} a_j x_j \le b\ \}$$

## Minimal Cover: $C \subseteq N$

- $\sum_{j \in C} a_j > b$
- $\sum_{j \in C \setminus \{i\}} a_j \le b \quad \forall\, i \in C$

## Minimal Cover Inequality

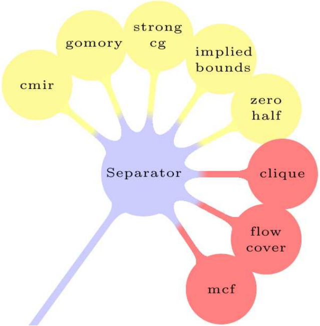
$$\sum_{j \in C} x_j \le |C| - 1$$

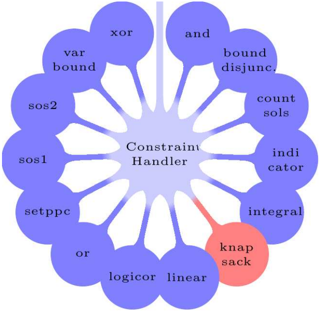
$$5x_1 + 6x_2 + 2x_3 + 2x_4 \le 8$$

Minimal cover: $C = \{2, 3, 4\}$

Minimal cover inequality:
$x_2 + x_3 + x_4 \le 2$

separated in
knapsack constraint handler



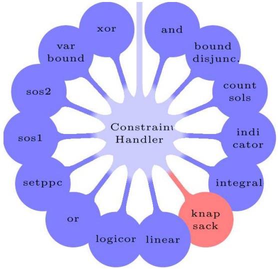

Separation is implemented in separators and constraint handlers.

1. Separators with `SEPA_PRIORITY` $\geq 0$ (decreasing order)
2. Constraint handlers (decreasing order of `CONSHDLR_SEPAPRIORITY`)
3. Separators with `SEPA_PRIORITY` $< 0$ (decreasing order)

# Ingredients of a Constraint Handler

## Callback Methods

- Fundamental:
  - CONSLOCK
  - CONSCHECK
  - CONSENFOLP, CONSENFOPS
- Additional:
  - CONSINIT..., CONSEXIT...
  - CONSSEPALP, CONSSEPASOL
  - CONSPROP, CONSRESPROP, CONSPRESOL
  - CONSACTIVE, CONSDEACTIVE, CONSENABLE, CONSDISABLE
  - CONSTRANS, CONSDELETE, CONSFREE
  - CONSPRINT, CONSPARSE, CONSCOPY

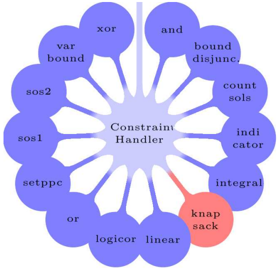



- Domain propagation (during subproblem processing)
- Reason for domain reductions (for conflict analysis)
- Presolving (before processing root node)

# Ingredients of a Constraint Handler

## Callback Methods

- ▷ **Fundamental:**
  - CONSLOCK
  - CONSCHECK
  - CONSENFOLP, CONSENFOPS
- ▷ **Additional:**
  - CONSINIT..., CONSEXIT...
  - CONSSEPALP, CONSSEPASOL
  - CONSPROP, CONSRESPROP, CONSPRESOL
  - CONSACTIVE, CONSDEACTIVE, CONSENABLE, CONSDISABLE
  - CONSTRANS, CONSDELETE, CONSFREE
  - CONSPRINT, CONSPARSE, CONSCOPY



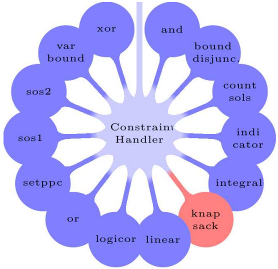

Called whenever ...

- ▷ SCIP enters/leaves subtree where local constraint exists
- ▷ constraint is enabled/disabled (no propagation, no separation)

# Ingredients of a Constraint Handler

## Callback Methods

- ▷ Fundamental:
  - CONSLOCK
  - CONSCHECK
  - CONSENFOLP, CONSENFOPS
- ▷ Additional:
  - CONSINIT..., CONSEXIT...
  - CONSSEPALP, CONSSEPASOL
  - CONSPROP, CONSRESPROP, CONSPRESOL
  - CONSACTIVE, CONSDEACTIVE, CONSENABLE, CONSDISABLE
  - CONSTRANS, CONSDELETE, CONSFREE
  - CONSPRINT, CONSPARSE, CONSCOPY

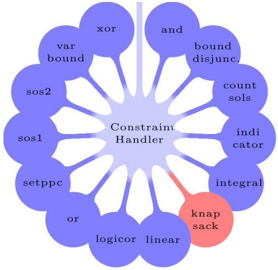



- ▷ Displaying, parsing problems
- ▷ Copying problems between different SCIP environments

# Ingredients of a Constraint Handler

## Callback Methods

- ▷ Fundamental:
  - CONSLOCK
  - CONSCHECK
  - CONSENFOLP, CONSENFOPS
- ▷ Additional:
  - CONSINIT..., CONSEXIT...
  - CONSSEPALP, CONSSEPASOL
  - CONSPROP, CONSRESPROP, CONSPRESOL
  - CONSACTIVE, CONSDEACTIVE, CONSENABLE, CONSDISABLE
  - CONSTRANS, CONSDELETE, CONSFREE
  - CONSPRINT, CONSPARSE, CONSCOPY




## Further Ingredients

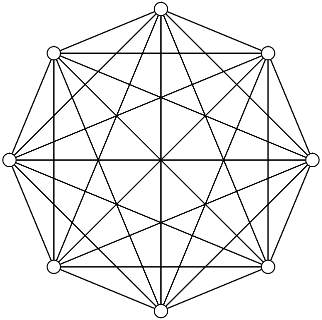
- ▷ Private data
- ▷ Interface methods
- ▷ Properties/Parameters

# Exercise: Traveling Salesman Problem (TSP)

**Definition**

Given a complete graph $G = (V, E)$ with edge lengths $c_e$.

Find a Hamiltonian cycle (cycle containing all nodes, tour) of minimum length.



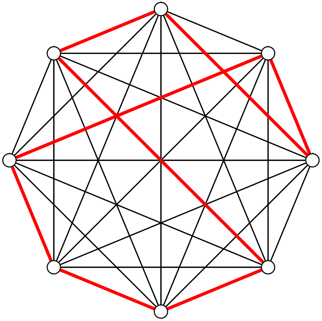
$K_8$

# Exercise: Traveling Salesman Problem (TSP)

**Definition**

Given a complete graph $G = (V, E)$ with edge lengths $c_e$.

Find a Hamiltonian cycle (cycle containing all nodes, tour) of minimum length.



$K_8$

# Exercise: TSP

## MIP Formulation

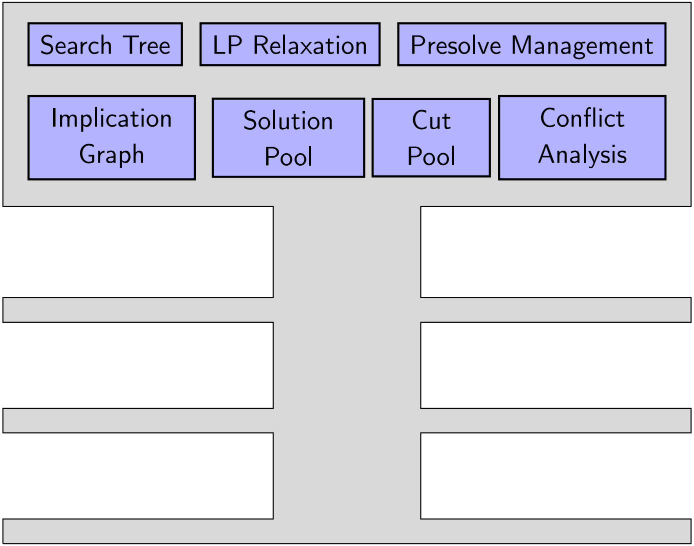
$$
\begin{aligned}
\min \quad & \sum_{e \in E} c_e\, x_e \\
s.t. \quad & \sum_{e \in \delta(v)} x_e = 2 && \forall\, v \in V \\
& \sum_{e \in \delta(S)} x_e \geq 2 && \forall\, S \subset V, S \neq \varnothing \\
& x_e \in \{0,1\} && \forall\, e \in E.
\end{aligned}
$$

## CIP Formulation

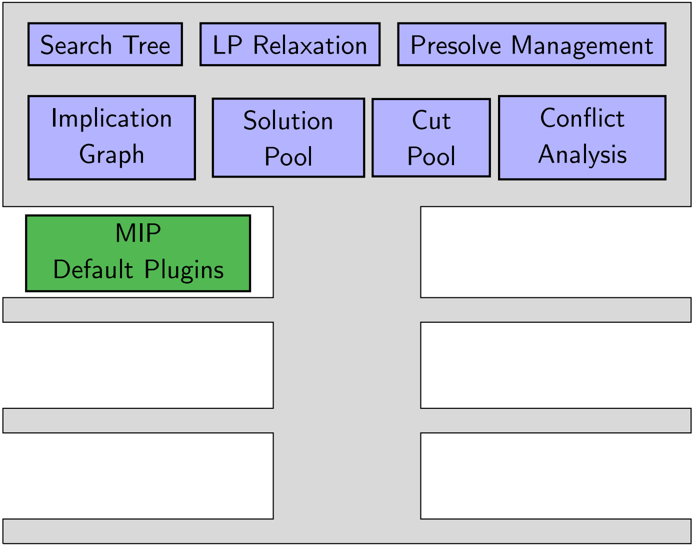
$$
\begin{aligned}
\min \quad & \sum_{e \in E} c_e\, x_e \\
s.t. \quad & \sum_{e \in \delta(v)} x_e = 2 && \forall\, v \in V \\
& \mathrm{nosubtour}(G, x) \\
& x_e \in \{0,1\} && \forall\, e \in E.
\end{aligned}
$$

$$\mathrm{nosubtour}(G, x) \;\Leftrightarrow\; \nexists\, C \subseteq \{e \in E \mid x_e = 1\} : C \text{ is a cycle of length } |C| < |V|$$

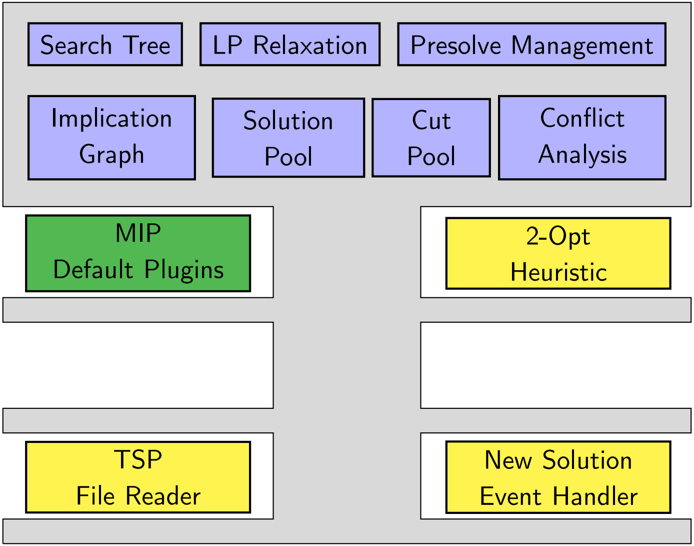
# Exercise: TSP



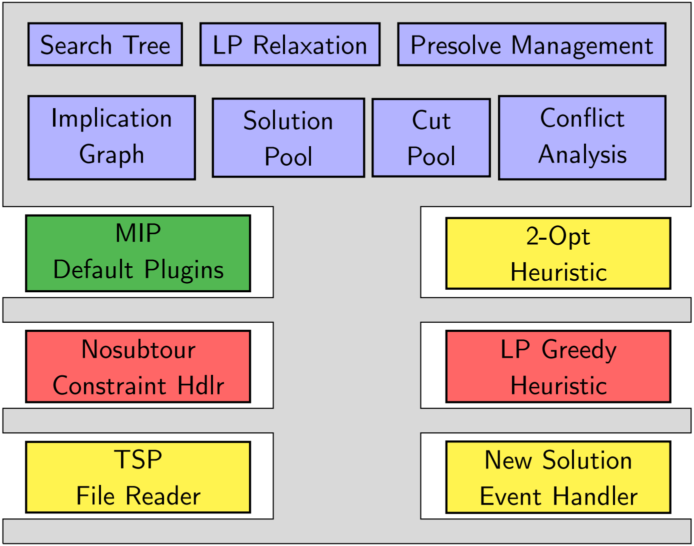
# Exercise: TSP



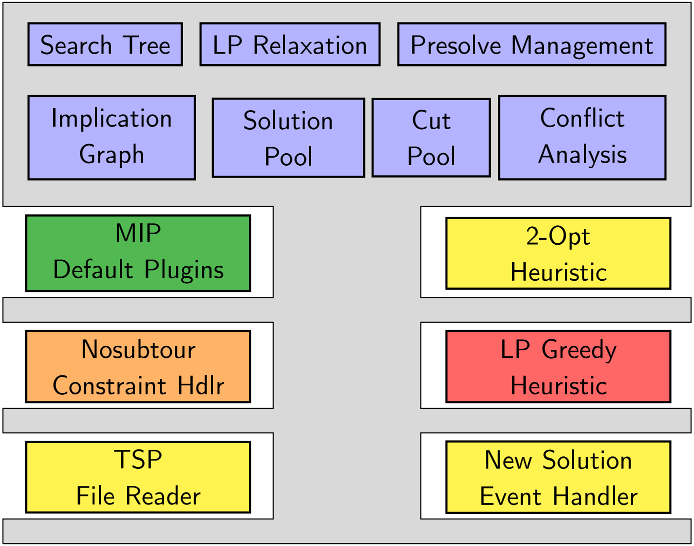
# Exercise: TSP



# Exercise: TSP



# Exercise: TSP

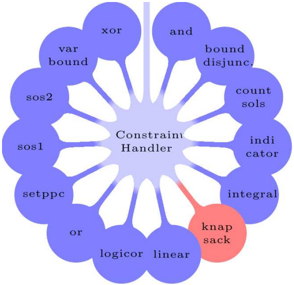


# Exercise: TSP

## Callback Methods

- ▷ Fundamental:
  - CONSLOCK
  - CONSCHECK
  - CONSENFOLP, CONSENFOPS

- ▷ Additional:
  - CONSINIT..., CONSEXIT...
  - CONSSEPALP, CONSSEPASOL
  - CONSPROP, CONSRESPROP, CONSPRESOL
  - CONSACTIVE, CONSDEACTIVE, CONSENABLE, CONSDISABLE
  - CONSTRANS, CONSDELETE, CONSFREE
  - CONSPRINT, CONSPARSE, CONSCOPY




## Further Ingredients

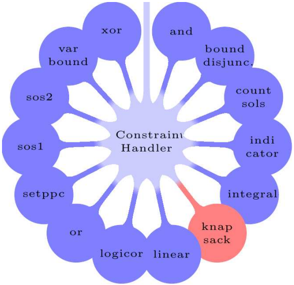
- ▷ Private data
- ▷ Interface methods
- ▷ Properties

# Exercise: TSP

## Callback Methods

- ▷ Fundamental:
  - CONSLOCK
  - CONSCHECK
  - CONSENFOLP, CONSENFOPS

- ▷ Additional:
  - CONSSEPALP, CONSSEPASOL
  - CONSTRANS, CONSDELETE



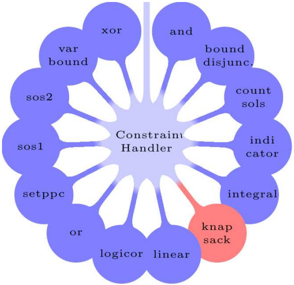

## Further Ingredients

- ▷ Private data
- ▷ Interface methods
- ▷ Properties

# Exercise: TSP

## Callback Methods

- Fundamental:
  - CONSLOCK
  - CONSCHECK
  - CONSENFOLP, CONSENFOPS
- Additional:
  - CONSSEPALP, CONSSEPASOL
  - CONSTRANS, CONSDELETE




## Further Ingredients

- Private data
- Interface methods
- Properties

# CONSSEPALP, CONSSEPASOL

## Separation Problem

Given complete graph $G = (V, E)$ and $x^* \in [0, 1]^{|E|}$.

- ▷ Decide whether $x^*$ satisfies all subtour elimination constraints.
- ▷ If not, find violated subtour elimination constraint.

## Separation Problem

Given complete graph $G = (V, E)$ and $x^* \in [0, 1]^{|E|}$.

- Decide whether $x^*$ satisfies all subtour elimination constraints.
- If not, find violated subtour elimination constraint.

Trivial observation:

- Consider $G = (V, E)$ with edge capacities $x^*_e$.
- $x^*$ violates at least one subtour elimination constraint
  $\Leftrightarrow \exists$ cut $\delta(S)$ with capacity $x^*(\delta(S)) < 2$.

## Separation Problem

Given complete graph $G = (V, E)$ and $x^* \in [0, 1]^{|E|}$.

- Decide whether $x^*$ satisfies all subtour elimination constraints.
- If not, find violated subtour elimination constraint.

Idea of separation algorithm:

- $\forall\, s, t \in V$: Find $(s, t)$-cut $\delta(S)$ of minimum capacity
- If all cut capacities $\geq 2$, all subtour elimination constraints satisfied

$\rightarrow$ $\binom{|V|}{2}$ times MaxFlow-MinCut algo

## Separation Problem

Given complete graph $G = (V, E)$ and $x^* \in [0, 1]^{|E|}$.

- Decide whether $x^*$ satisfies all subtour elimination constraints.
- If not, find violated subtour elimination constraint.

Idea of separation algorithm:

- $\forall\, s, t \in V$: Find $(s, t)$-cut $\delta(S)$ of minimum capacity
- If all cut capacities $\geq 2$, all subtour elimination constraints satisfied

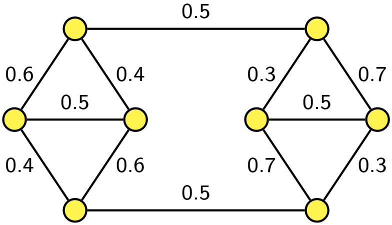
$\rightarrow$ $|V| - 1$ times MaxFlow-MinCut algo within Gomory-Hu-Algorithm

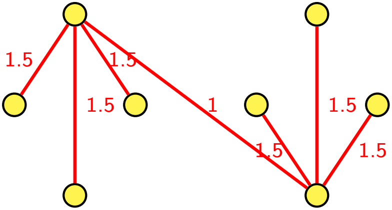
# Result of Gomory-Hu-Algorithm

## Gomory-Hu-Tree $T$

For all $s, t \in V$:

- capacity of minimum $(s,t)$-cut in $G$:
  minimum label $f_e$ of all edges $e$ in unique $(s,t)$-path in $T$
- minimum $(s,t)$-cut in $G$:
  bipartition of $V$ obtained be deleting this edge from $T$



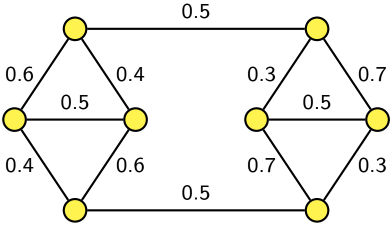



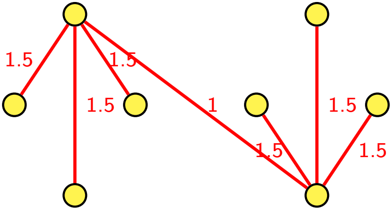
# Result of Gomory-Hu-Algorithm

## Gomory-Hu-Tree $T$

For all $s, t \in V$:

- capacity of minimum $(s, t)$-cut in $G$:
  minimum label $f_e$ of all edges $e$ in unique $(s, t)$-path in $T$
- minimum $(s, t)$-cut in $G$:
  bipartition of $V$ obtained be deleting this edge from $T$




`ghc_tree()` returns all minimum $(s, t)$-cuts with capacity $\leq 2 - \texttt{minviol}$

# Documentation

- Constraint Integer Programming
  Tobias Achterberg, Ph.D. dissertation, TU Berlin, 2007
- Primal Heuristics for Mixed Integer Programs
  Timo Berthold, Diploma thesis, TU Berlin, 2006
- Implementation of Cutting Plane Separators for Mixed Integer Programs
  Kati Wolter, Diploma thesis, TU Berlin, 2006

- `http://scip.zib.de`
  Doxygen documentation, HowTos, FAQ
- source code
  `scip.h`, `pub_*.h`, `type_*.h`