Implementing Constraint Handlers in SCIP

Gregor Hendel
Ambros Gleixner, Felipe Serrano

September 30

Research Center Matheon
Mathematics for Key Technologies
Berlin Mathematical School
MODAL
Mathematical Optimization and Data Analysis Laboratories

Outline

Outline

Why use constraints? Motivation

Callback: Default Constraint Handlers of SCIP

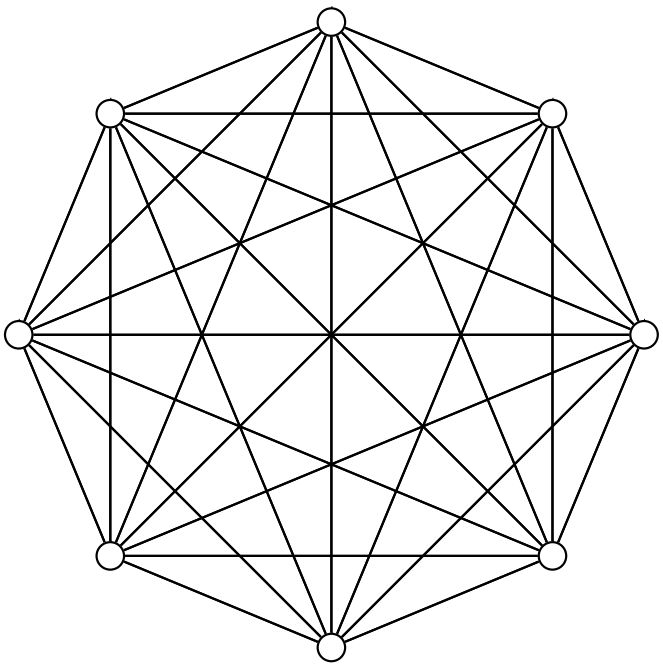
Implementation hints for the Nosubtour Constraint Handler

Traveling Salesman Problem (TSP)

Definition

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

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



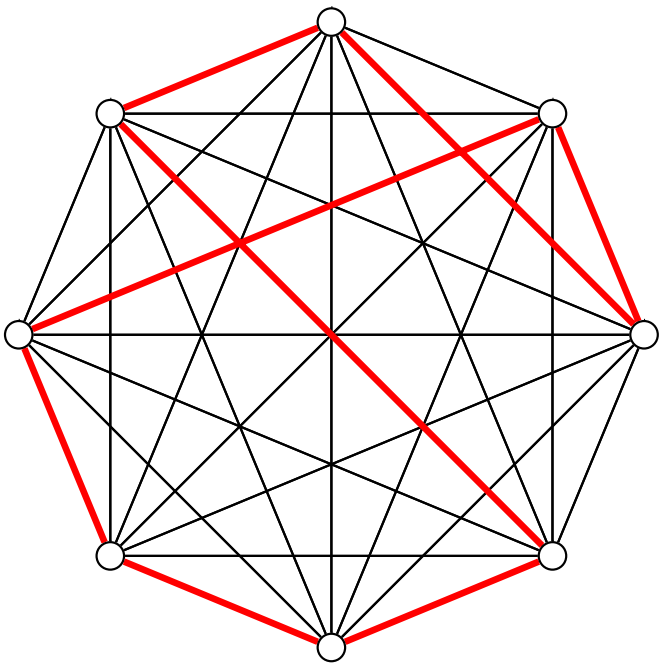
K_8

Traveling Salesman Problem (TSP)

Definition

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

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



K_8

Motivation For General Constraints:

MIP Formulation

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

Motivation For General Constraints:

MIP Formulation

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

CIP Formulation

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

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

Separation Problem

Given 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 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

Separation Problem

Given 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 ≥ 2, all subtour elimination constraints satisfied

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

Separation Problem

Given 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 ≥ 2, all subtour elimination constraints satisfied

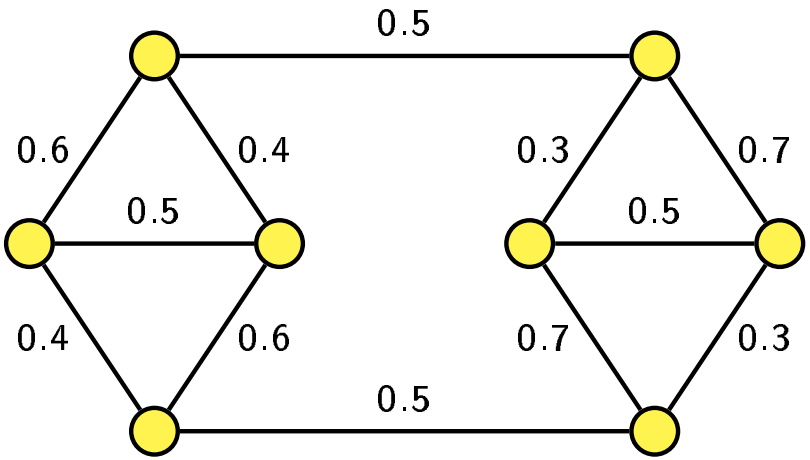
$\rightarrow$ $|V| - 1$ times MaxFlow-MinCut algo within Gomory-Hu-Algorithm

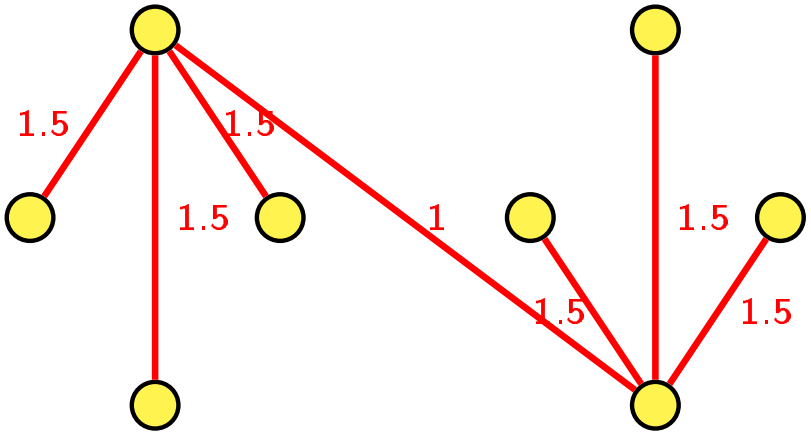
Result of Gomory-Hu-Algorithm

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





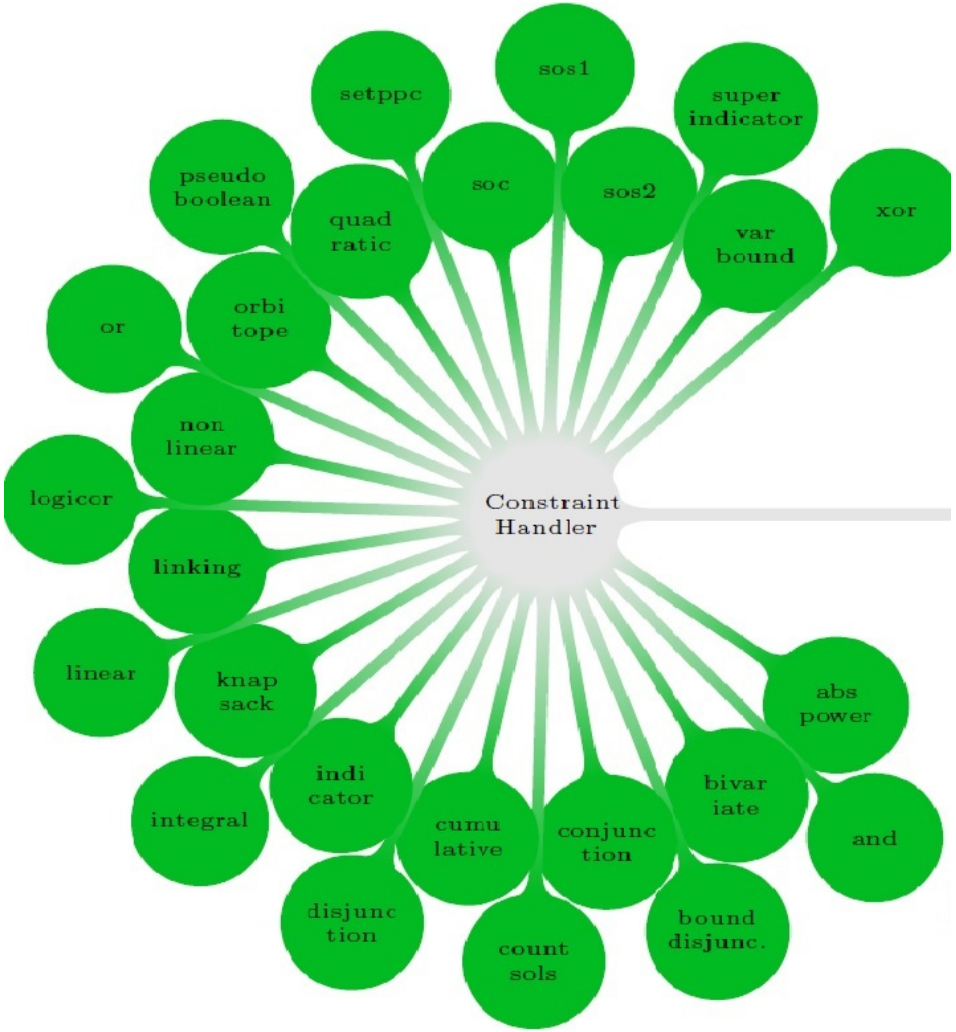
Outline

Why use constraints? Motivation

Callback: Default Constraint Handlers of SCIP

Implementation hints for the Nosubtour Constraint Handler

Default Constraint Handlers



Special Linear Constraints

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

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

Special Linear Constraints

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

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

Special Linear Constraints

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

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

Special Linear Constraints

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

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

Special Linear Constraints

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

```
presolved problem has 7 variables (4 bin, 1 int, 0 impl, 2 cont) and 5 constraints
      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:

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

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

The Integral Constraint Handler

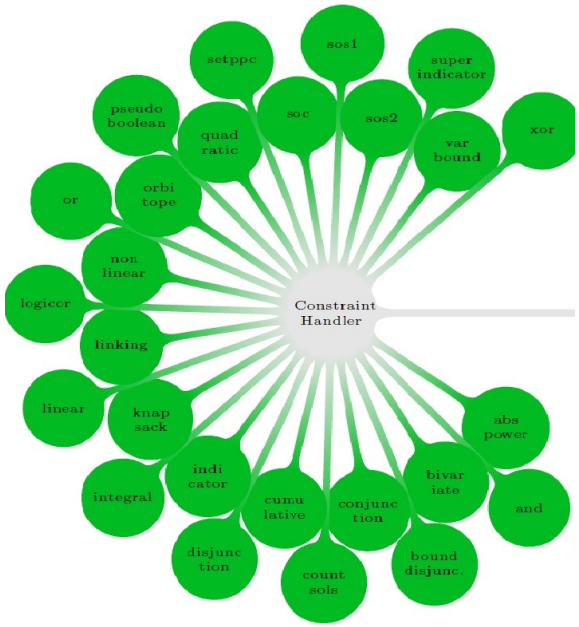
The integral constraint handler

- ensures the integrality of discrete problem variables in a solution
- does not need constraints because integrality restrictions are encoded in the type of a variable
- is the shortest example of a working constraint handler I know

Ingredients of a Constraint Handler

Callback Methods

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



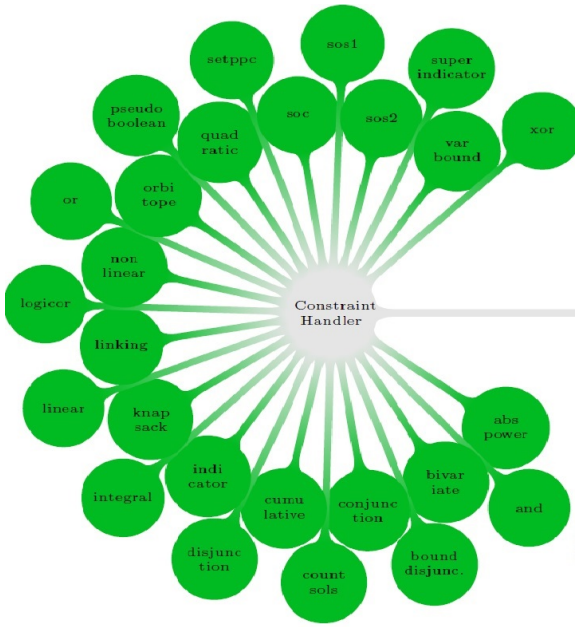
Further Ingredients

- Private data
- Interface methods
- Properties/Parameters

Ingredients of a Constraint Handler

Callback Methods

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



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 SCIPcreateConsBasicKnapsack(
   SCIP*                    scip,       /**< SCIP data structure */
   SCIP_CONS**              cons,       /**< pointer to hold the created constraint
   const char*              name,       /**< name of constraint */
   int                      nvars,      /**< number of items in the knapsack */
   SCIP_VAR**               vars,       /**< array with item variables */
   SCIP_Longint*            weights,    /**< array with item weights */
   SCIP_Longint             capacity    /**< capacity of knapsack */
   )
{
    SCIP_CONSDATA* consdata;
    SCIP_CALL( consdataCreate(scip, &consdata, nvars, vars,
                              weights, capacity) );
    SCIP_CALL( SCIPcreateConsBasic(scip, cons, name, conshdlr,
                              consdata) );
    return SCIP_OKAY;
}
```

Interface Methods

Including the knapsack constraint handler.

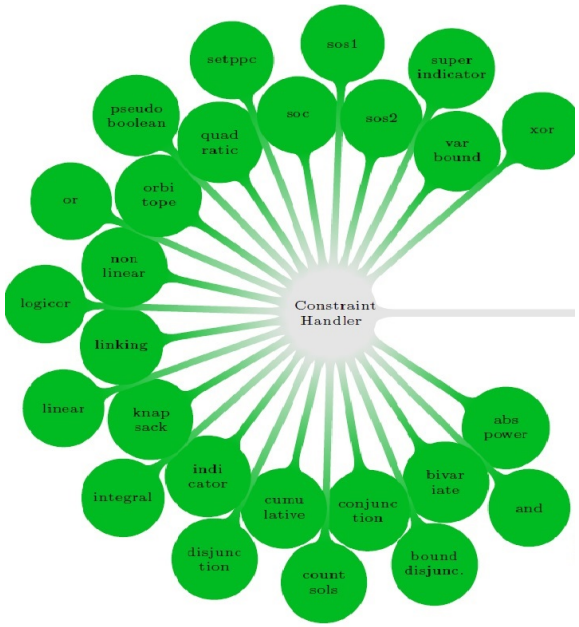
```
SCIP_RETCODE SCIPincludeConshdlrKnapsack(
    SCIP*         scip       // SCIP data structure
    )
{
SCIP_CONSHDLR* conshdlr;
SCIP_CONSHDLRDATA* conshdlrdata;

SCIP_CALL( conshdlrdataCreate(scip, &conshdlrdata) );


/* include constraint handler into SCIP */
SCIP_CALL( SCIPincludeConshdlrBasic(scip, &conshdlr,
    CONSHDLR_NAME, CONSHDLR_DESC,
    [...]
    consEnfolpKnapsack, consEnfopsKnapsack,
    consCheckKnapsack, consLockKnapsack,
    conshdlrdata) );

/* set non-fundamental callbacks via specific setter functions */
...
}
```

Ingredients of a Constraint Handler

Callback Methods

- Fundamental:
  - CONSCHECK
  - CONSENFOLP, CONSENFOPS
  - CONSLOCK

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



Further Ingredients

- Private data
- Interface methods
- Properties/Parameters

Callback Methods

What is a callback method?

Callback Methods

What is a callback method?

Nice quote from someone who cited an older Wikipedia version:

> *In computer programming, a callback is a reference to executable code, or a piece of executable code, that is passed as an argument to other code. This allows a lower-level software layer to call a subroutine (or function) defined in a higher-level layer.*

source:

`http://stackoverflow.com/questions/824234/what-is-a-callback-function`

Callback Methods

What is a callback method?

Nice quote from someone who cited an older Wikipedia version:

> *In computer programming, a callback is a reference to executable code, or a piece of executable code, that is passed as an argument to other code. This allows a lower-level software layer to call a subroutine (or function) defined in a higher-level layer.*

source:

`http://stackoverflow.com/questions/824234/what-is-a-callback-function`

All the functionality of constraint handlers (as well as all other plugin types of SCIP) is defined through callback functions.

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:

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

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

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

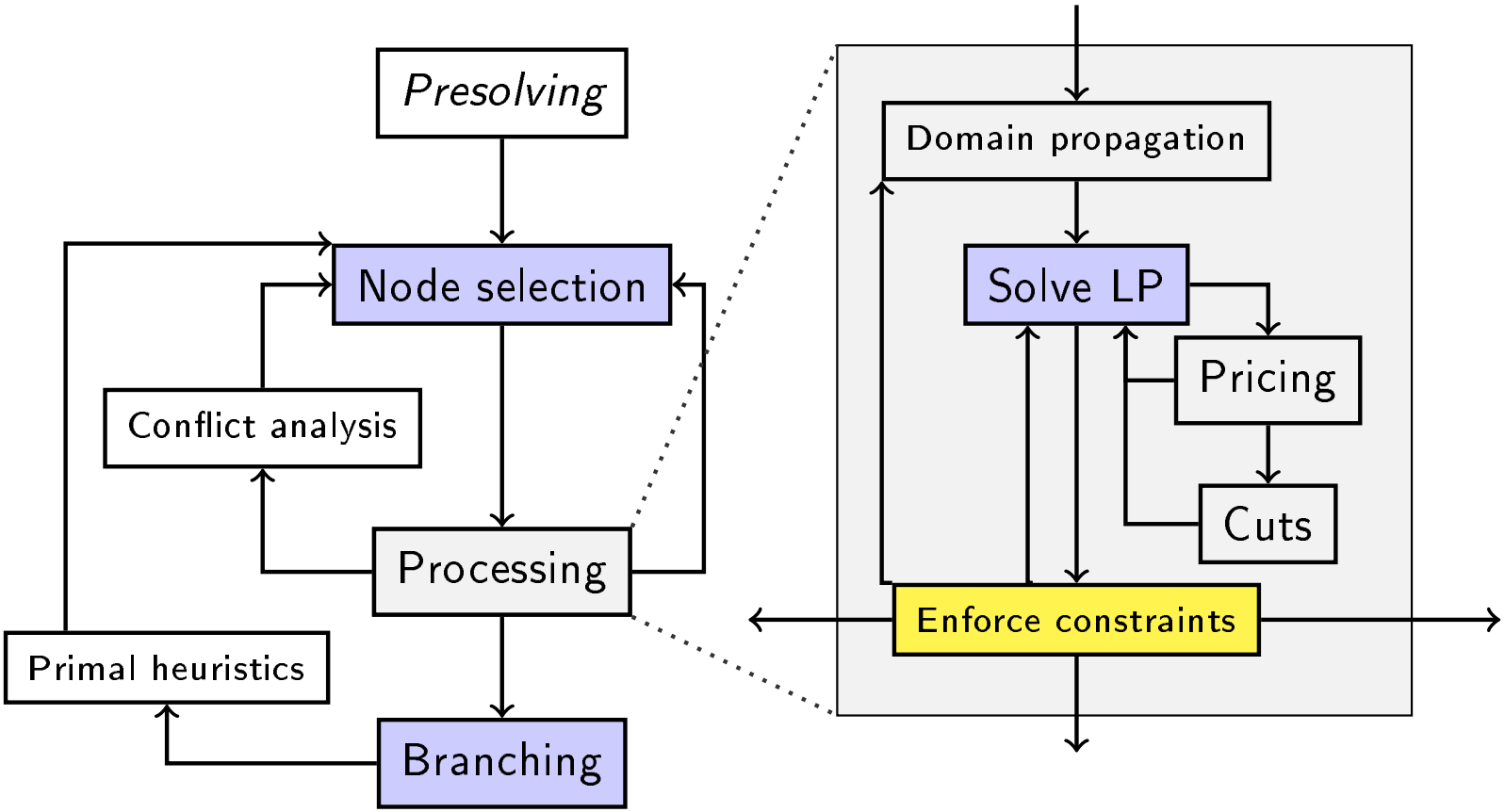
LP solution may violate a constraint not contained in the relaxation.

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".

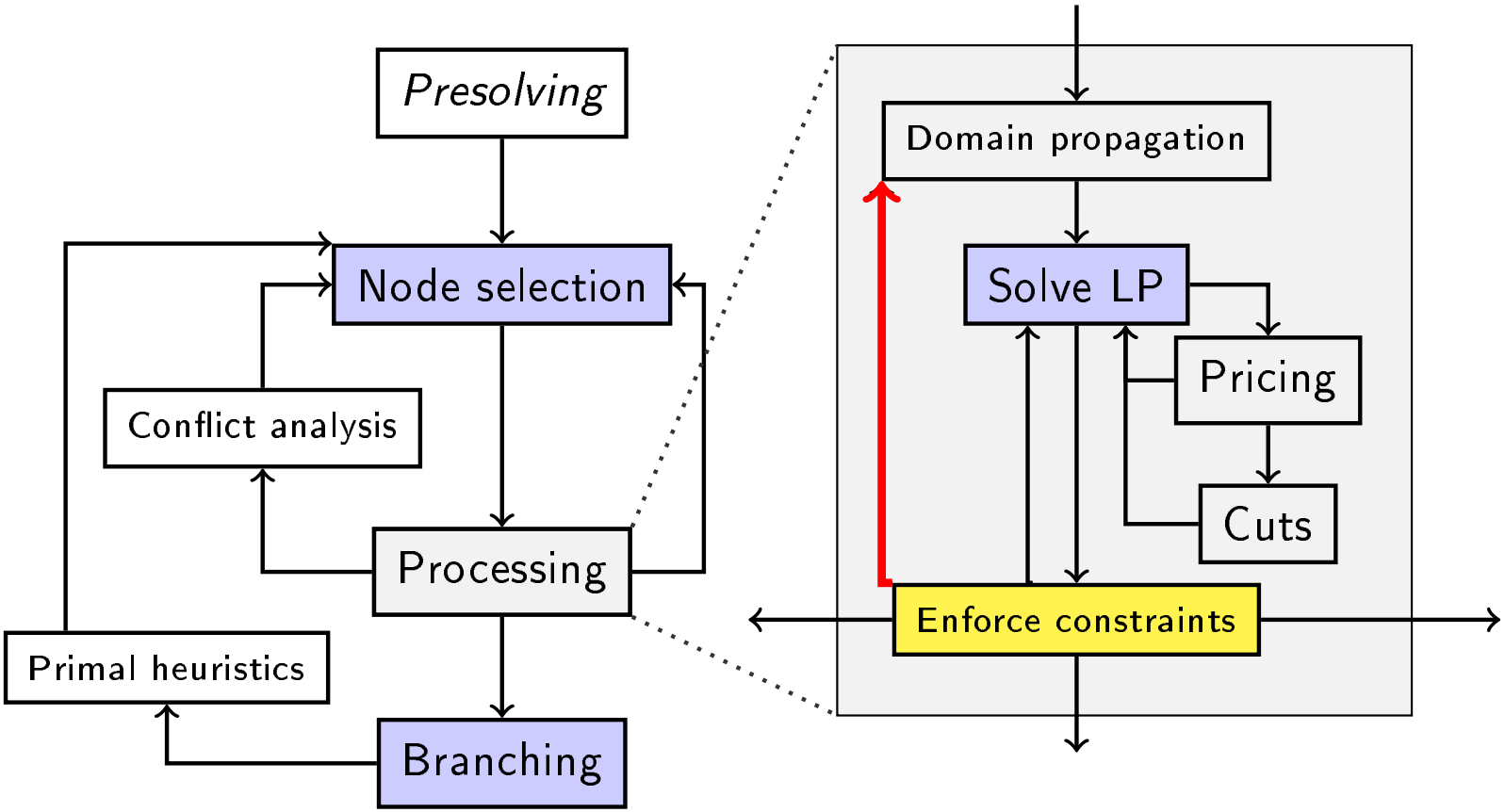
CONSENFOLP and CONSENFOPS



Enforcement result of constraint handler:

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

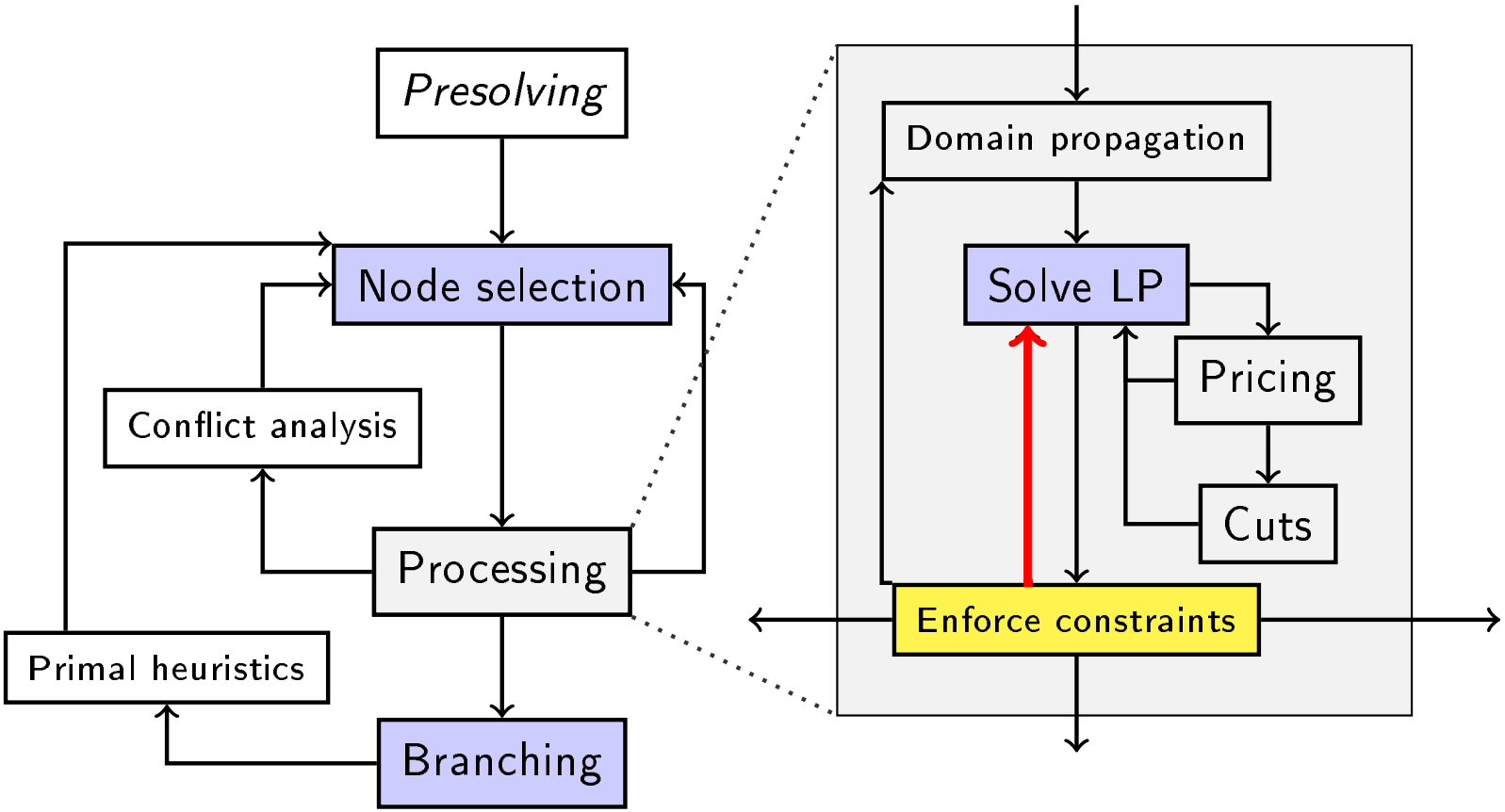
CONSENFOLP and CONSENFOPS



Enforcement result of constraint handler:

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

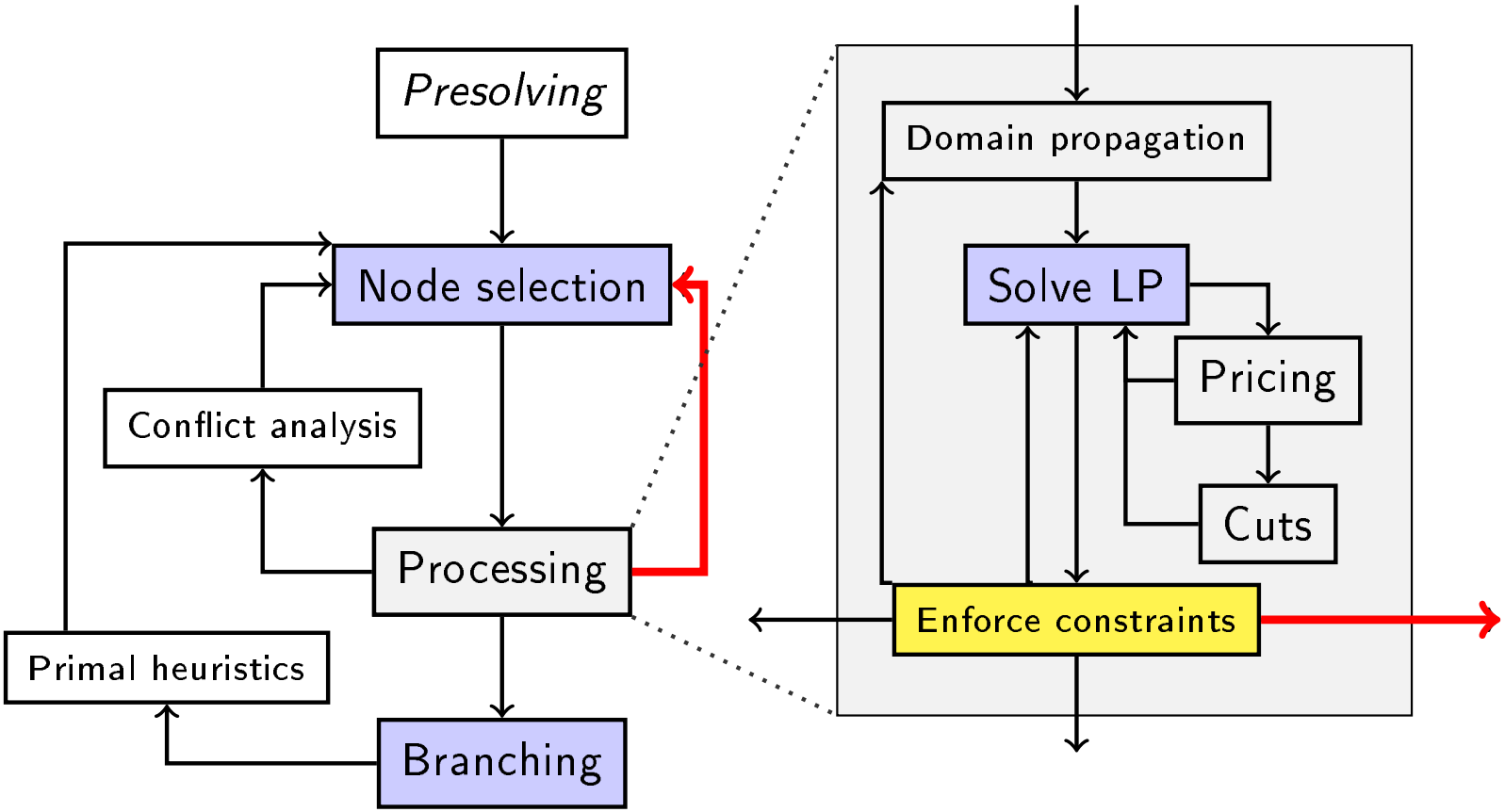
CONSENFOLP and CONSENFOPS



Enforcement result of constraint handler:

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

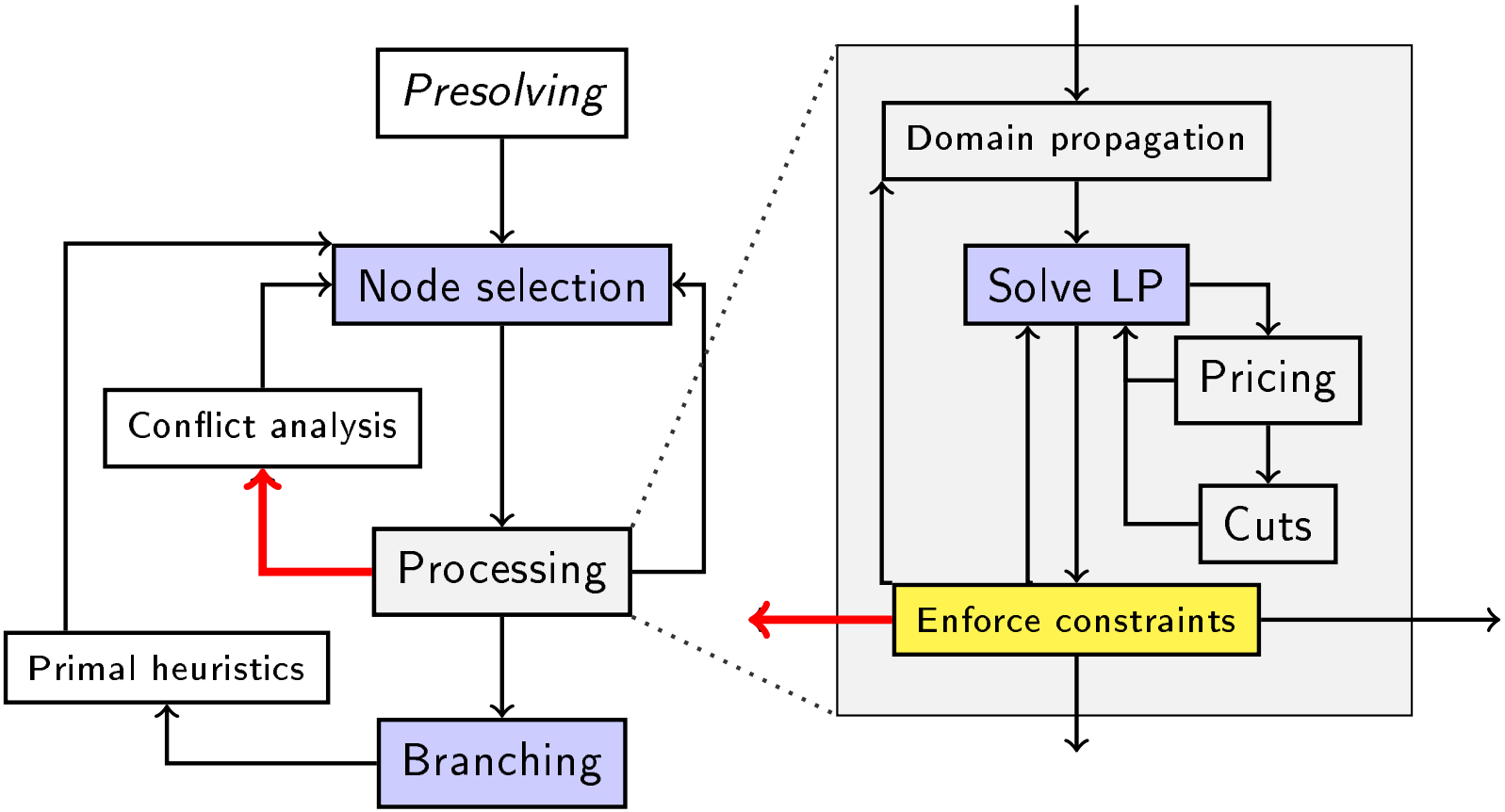
CONSENFOLP and CONSENFOPS



Enforcement result of constraint handler:

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

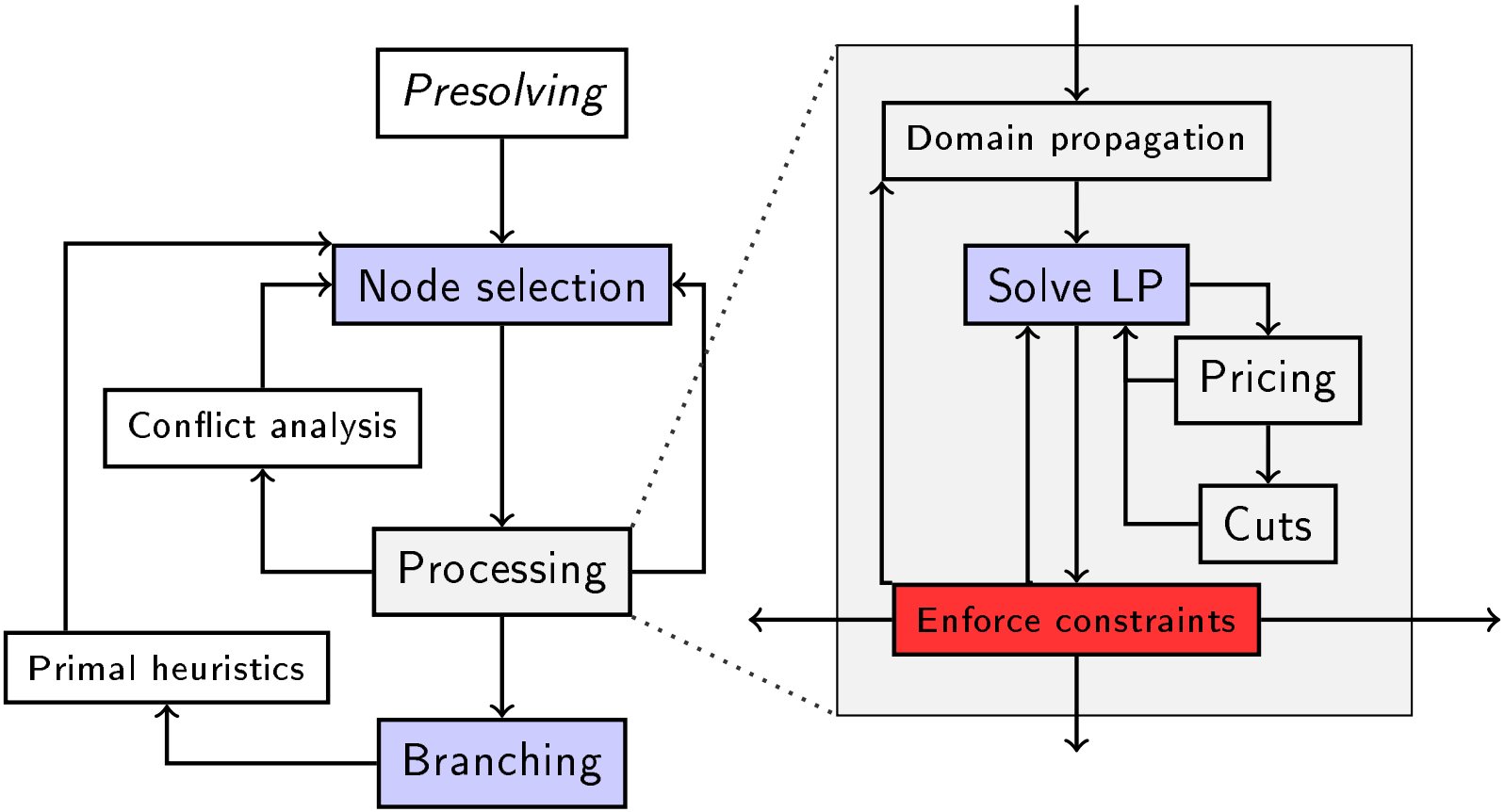
CONSENFOLP and CONSENFOPS



Enforcement result of constraint handler:

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

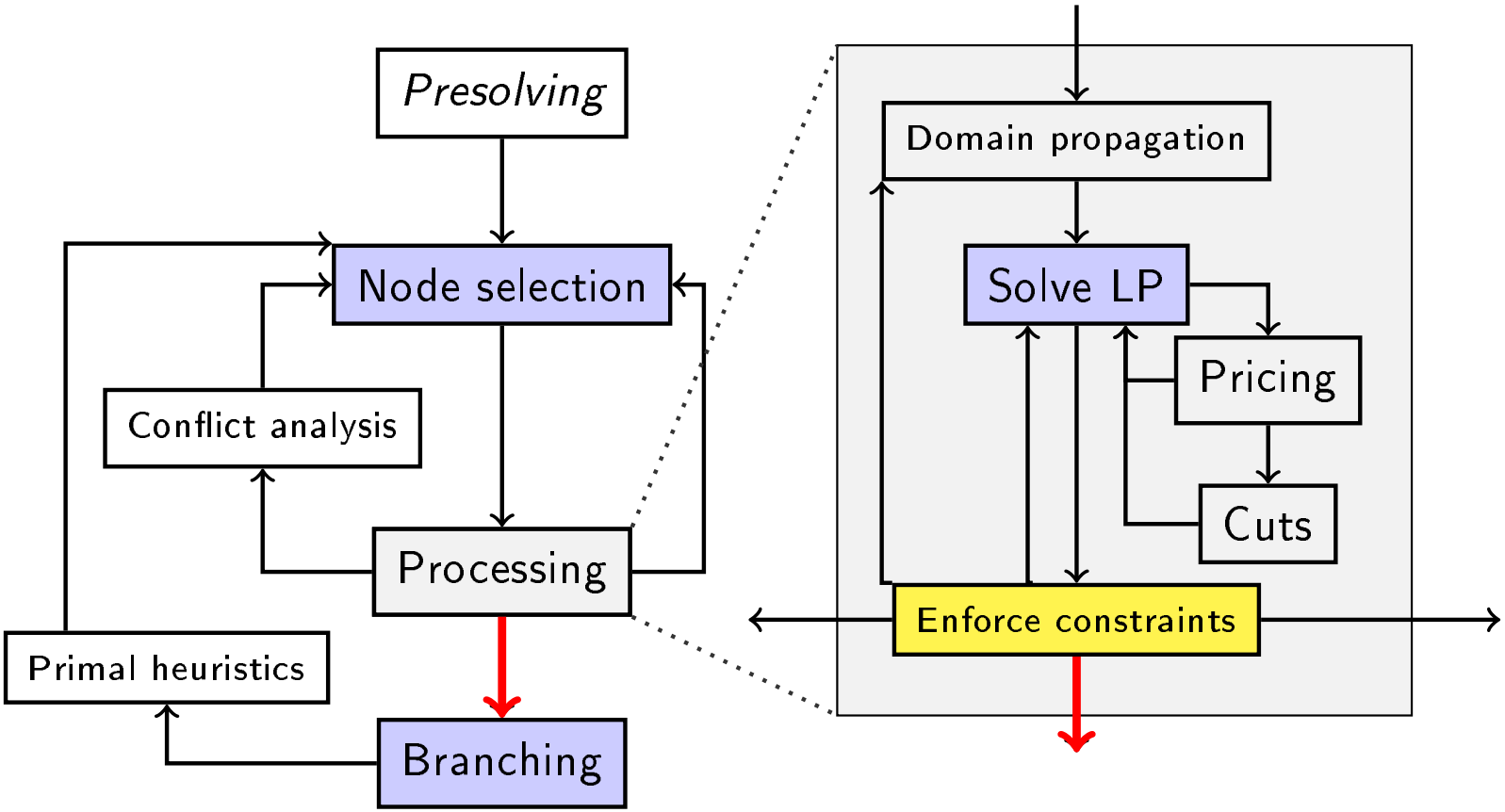
CONSENFOLP and CONSENFOPS



Enforcement result of constraint handler:

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

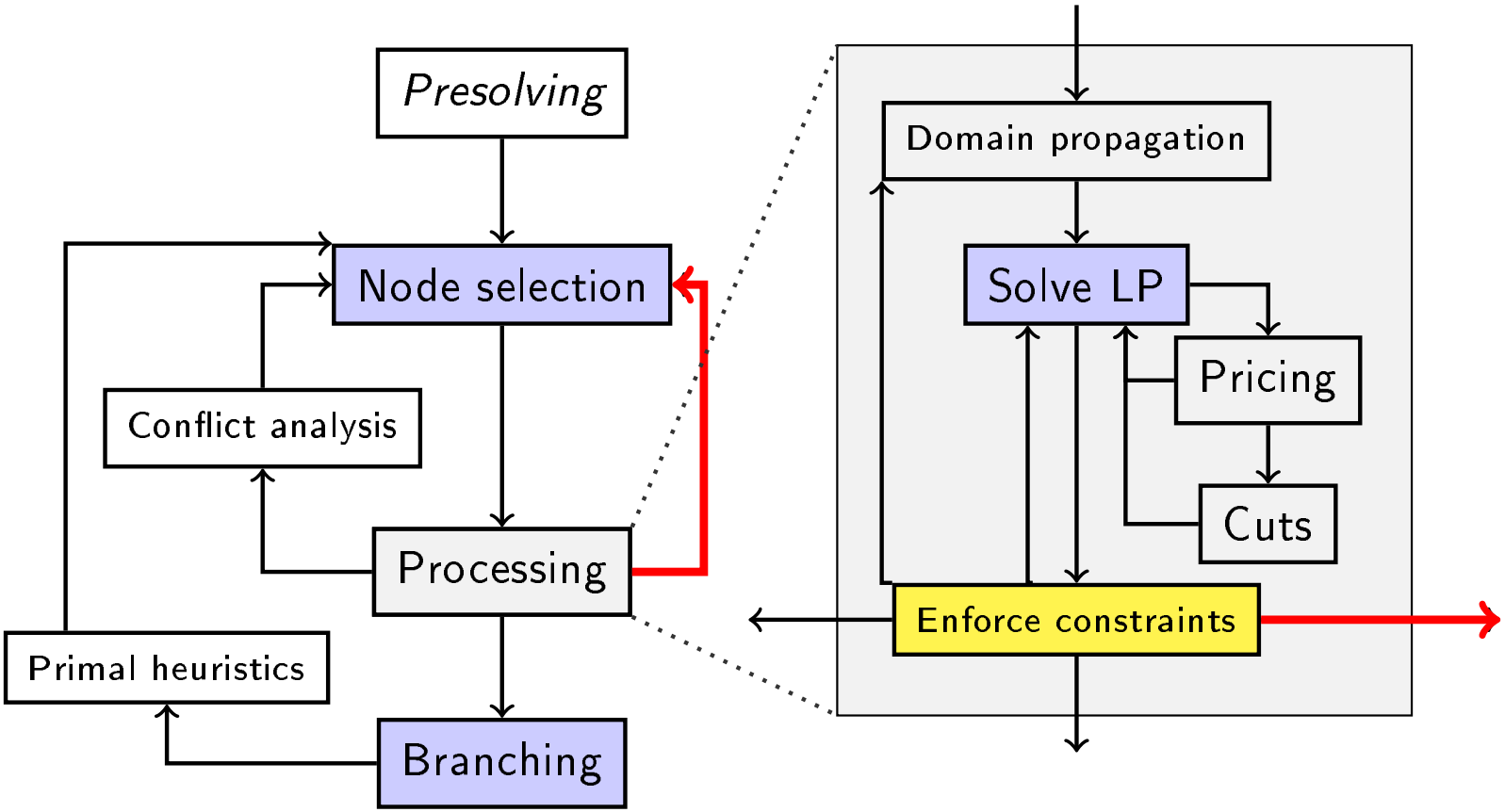
CONSENFOLP and CONSENFOPS



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

The Integral Constraint Handler

The integral constraint handler

- ensures the integrality of discrete problem variables in a solution
- does not need constraints because integrality restrictions are encoded in the type of a variable
- is the shortest example of a working constraint handler I know

The Integral Constraint Handler

The integral constraint handler

- ensures the integrality of discrete problem variables in a solution
- does not need constraints because integrality restrictions are encoded in the type of a variable
- is the shortest example of a working constraint handler I know

Question

How does the integral constraint handler enforce the LP solution?

The Integral Constraint Handler

The integral constraint handler

- ensures the integrality of discrete problem variables in a solution
- does not need constraints because integrality restrictions are encoded in the type of a variable
- is the shortest example of a working constraint handler I know

Question

How does the integral constraint handler enforce the LP solution?

Answer

It calls the branching rules to perform variable branching.

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

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

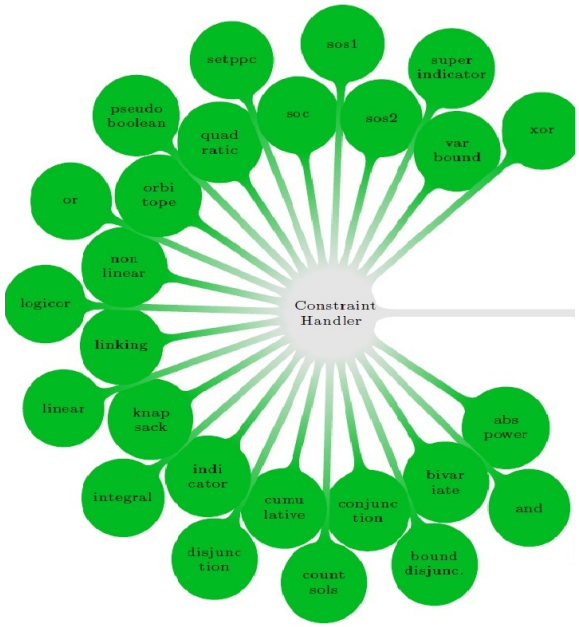
increasing: x_1 and x_3

decreasing: x_2

Ingredients of a Constraint Handler

Callback Methods

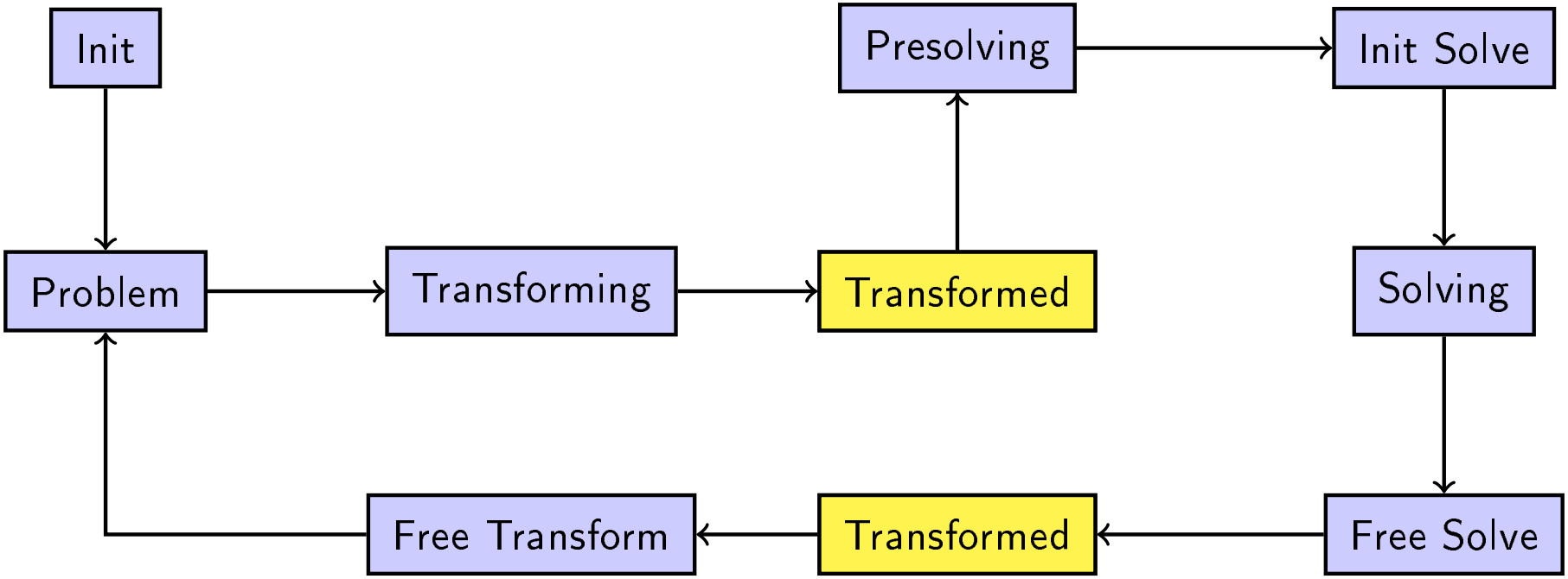
- Fundamental:
  - CONSCHECK
  - CONSENFOLP, CONSENFOPS
  - CONSLOCK
- Additional:
  - CONSINIT..., CONSEXIT...
  - CONSSEPALP, CONSSEPASOL
  - CONSPROP, CONSRESPROP, CONSPRESOL
  - CONSACTIVE, CONSDEACTIVE, CONSENABLE, CONSDISABLE
  - CONSTRANS, CONSDELETE, CONSFREE
  - CONSPRINT, CONSPARSE, CONSCOPY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



Further Ingredients

- Private data
- Interface methods
- Properties/Parameters

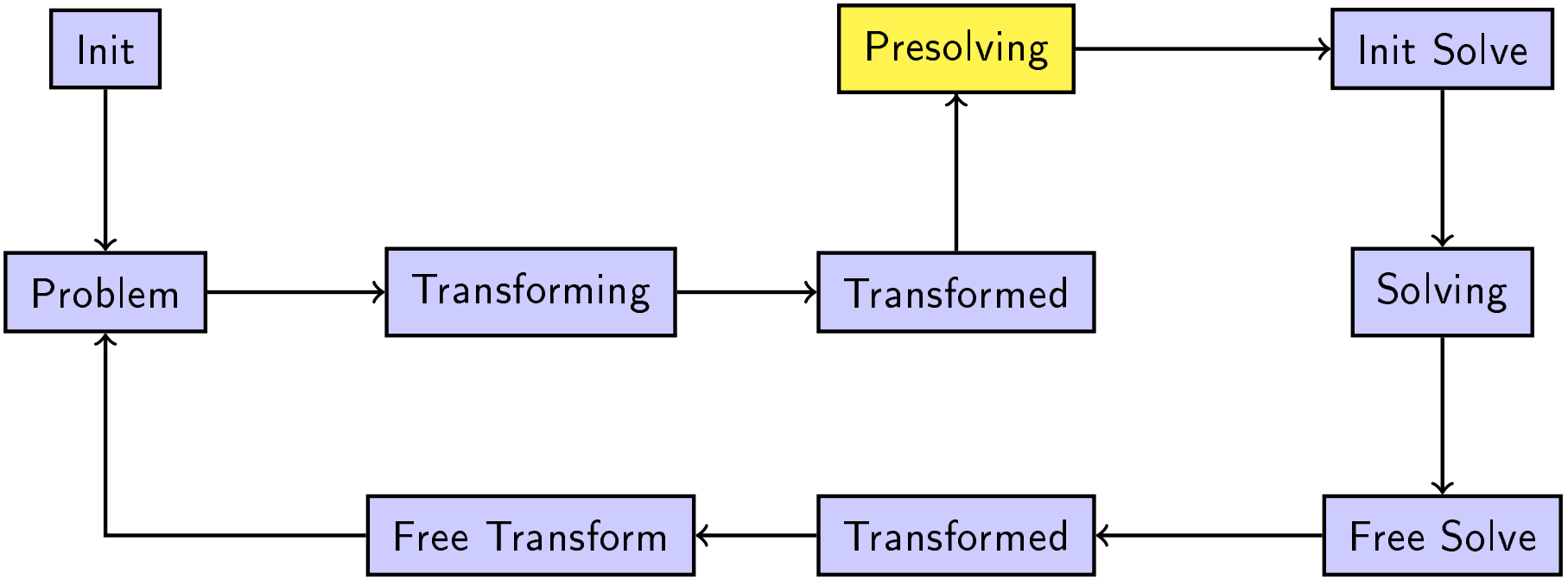
CONSINIT..., CONSEXIT...



CONSINIT and CONSEXIT:

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

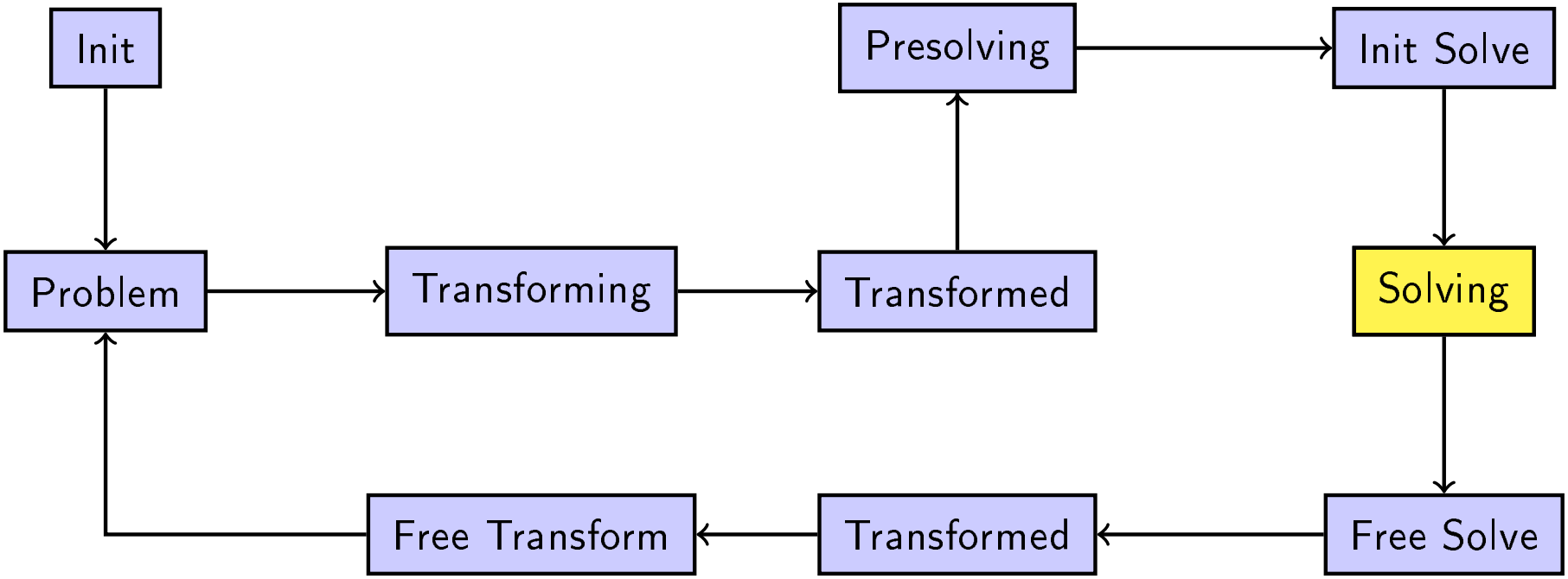
CONSINIT..., CONSEXIT...



CONSINITPRE and CONSEXITPRE:

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

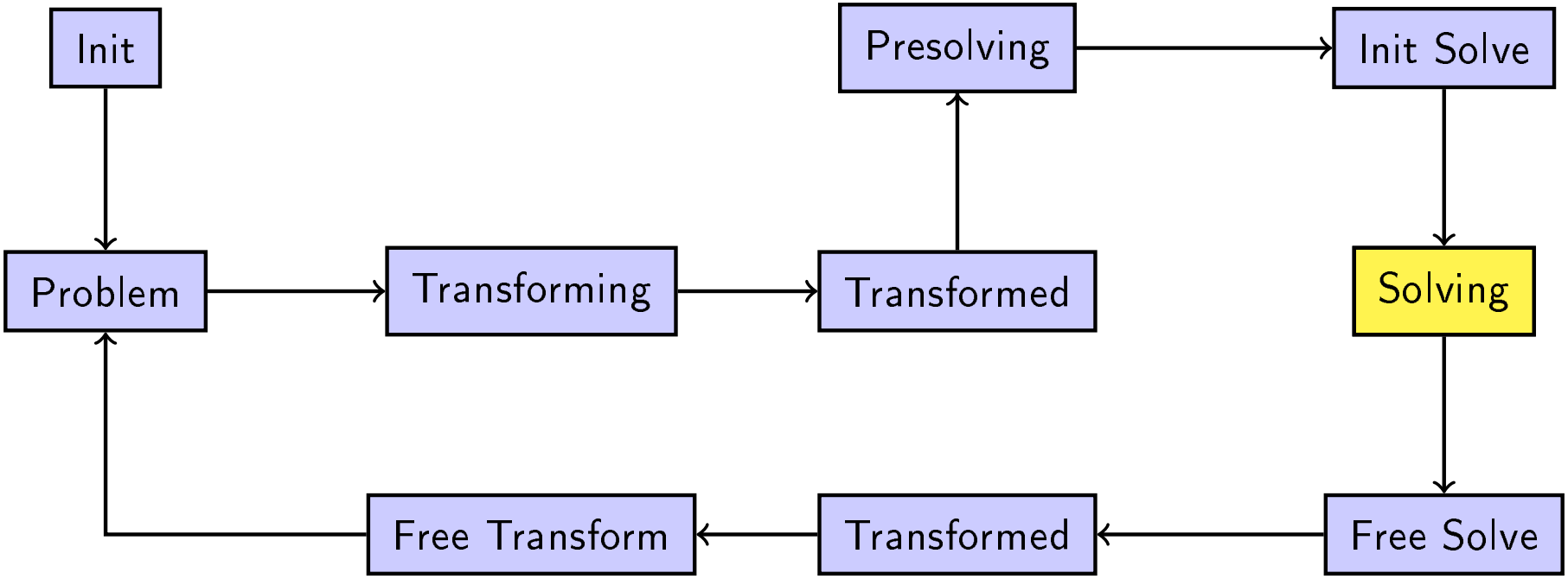
CONSINIT..., CONSEXIT...



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...



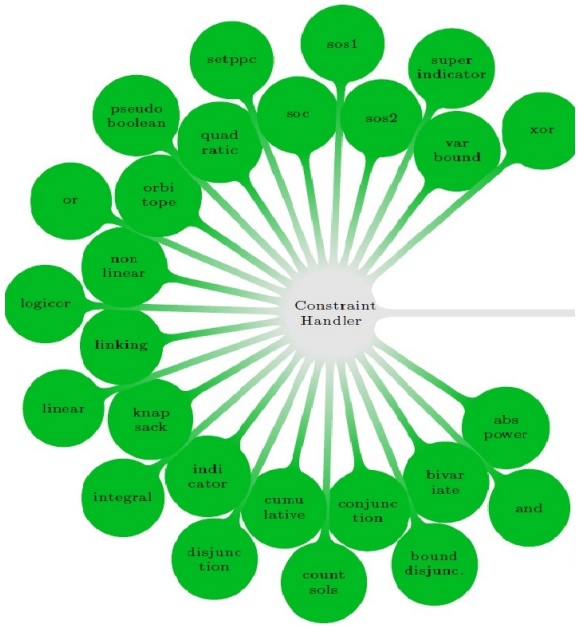
CONSINITLP:

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

Ingredients of a Constraint Handler

Callback Methods

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



Further Ingredients

- Private data
- Interface methods
- Properties/Parameters

CONSSEPALP, CONSSEPASOL

Feasible region of 0-1 knapsack problem:

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

Minimal Cover: $C \subseteq N$

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

Minimal Cover Inequality

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

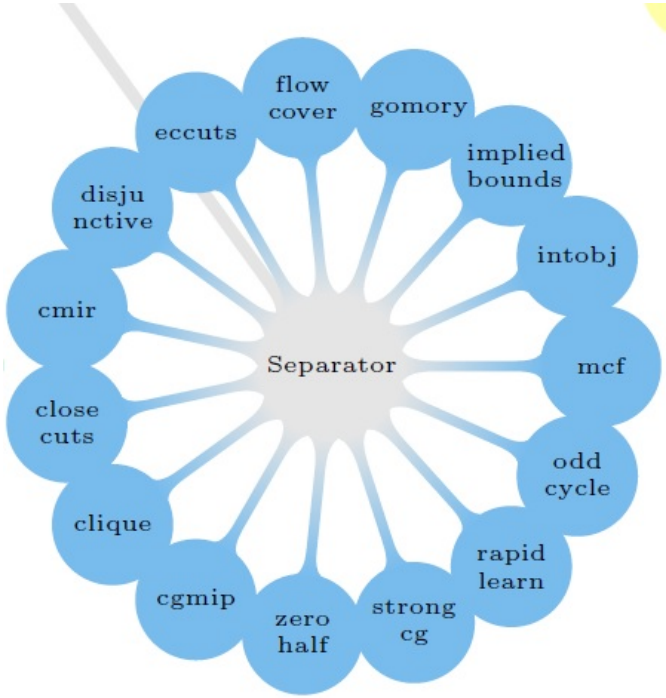
$5x_1 + 6x_2 + 2x_3 + 2x_4 \leq 8$

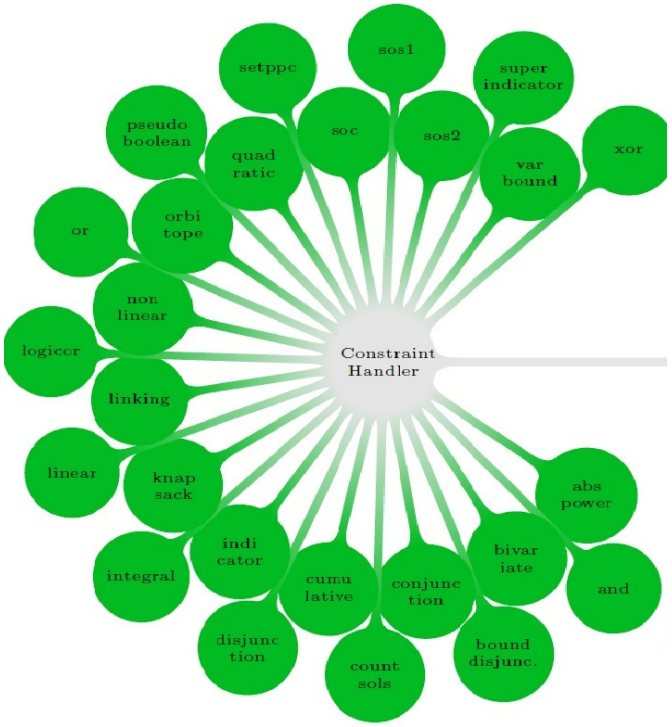
Minimal cover: $C = \{2, 3, 4\}$

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

separated in
knapsack constraint handler

CONSSEPALP, CONSSEPASOL





Separation is implemented in separators and constraint handlers.

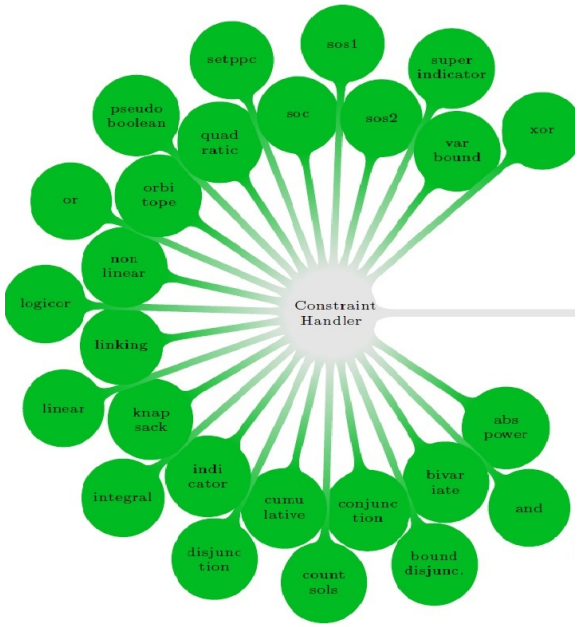
1. Separators with SEPA_PRIORITY ≥ 0 (decreasing order)
2. Constraint handlers (decreasing order of CONSHDLR_SEPAPRIORITY)
3. Separators with SEPA_PRIORITY < 0 (decreasing order)

Ingredients of a Constraint Handler

Callback Methods

- Fundamental:
  - CONSCHECK
  - CONSENFOLP, CONSENFOPS
  - CONSLOCK

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

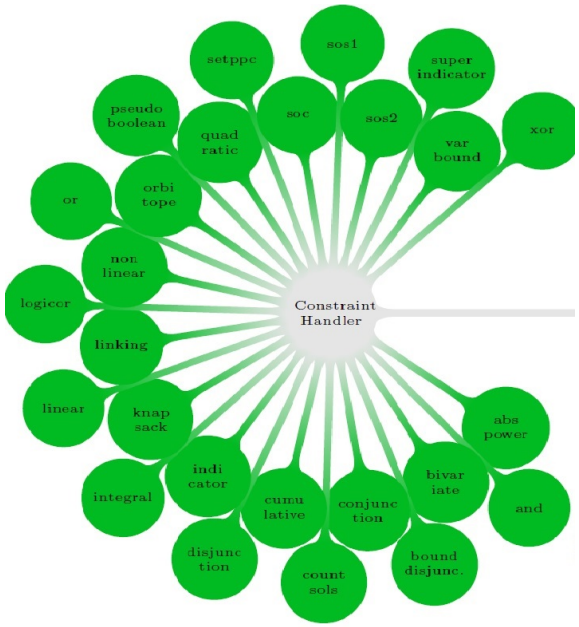


- Domain propagation
  (during subproblem processing)
- Reason for domain reductions
  (for conflict analysis)
- Presolving
  (before processing root node)

Ingredients of a Constraint Handler

Callback Methods

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



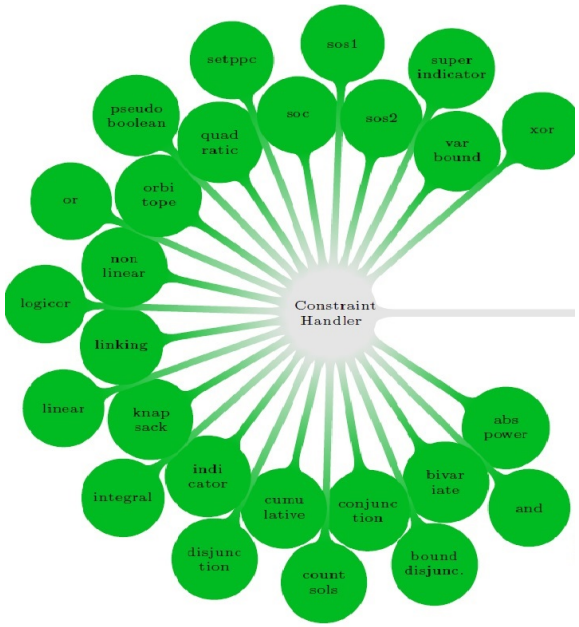
Called whenever ...

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

Ingredients of a Constraint Handler

Callback Methods

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



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

Outline

Why use constraints? Motivation

Callback: Default Constraint Handlers of SCIP

Implementation hints for the Nosubtour Constraint Handler

Exercise: TSP

MIP Formulation

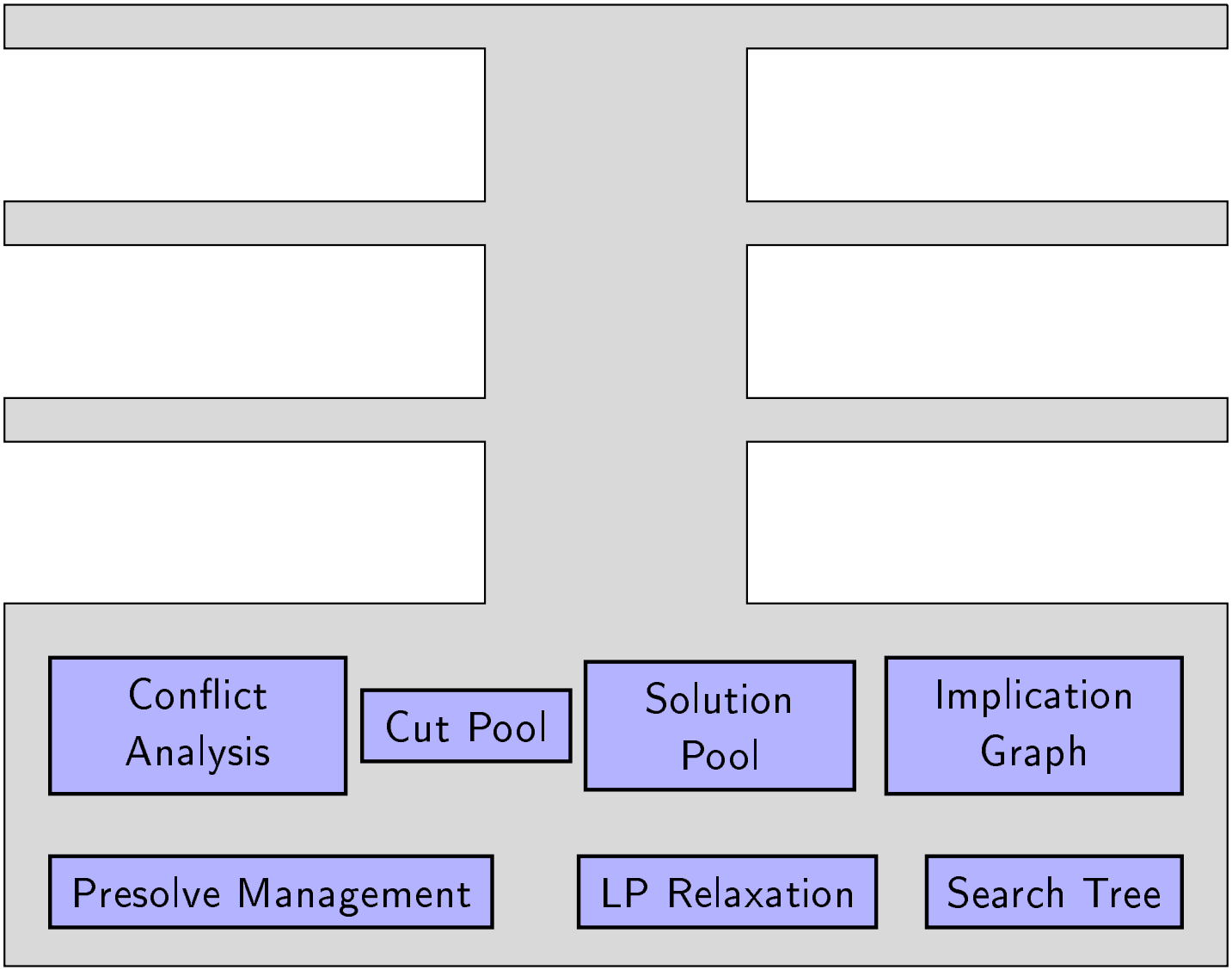
$$
\begin{aligned}
\min \quad & \sum_{e \in E} c_e \, x_e \\
s.t. \quad & \sum_{e \in \delta(v)} x_e = 2 \quad \forall \, v \in V \\
& \sum_{e \in \delta(S)} x_e \geq 2 \quad \forall \, S \subset V, S \neq \varnothing \\
& x_e \in \{0,1\} \quad \forall \, e \in E.
\end{aligned}
$$

CIP Formulation

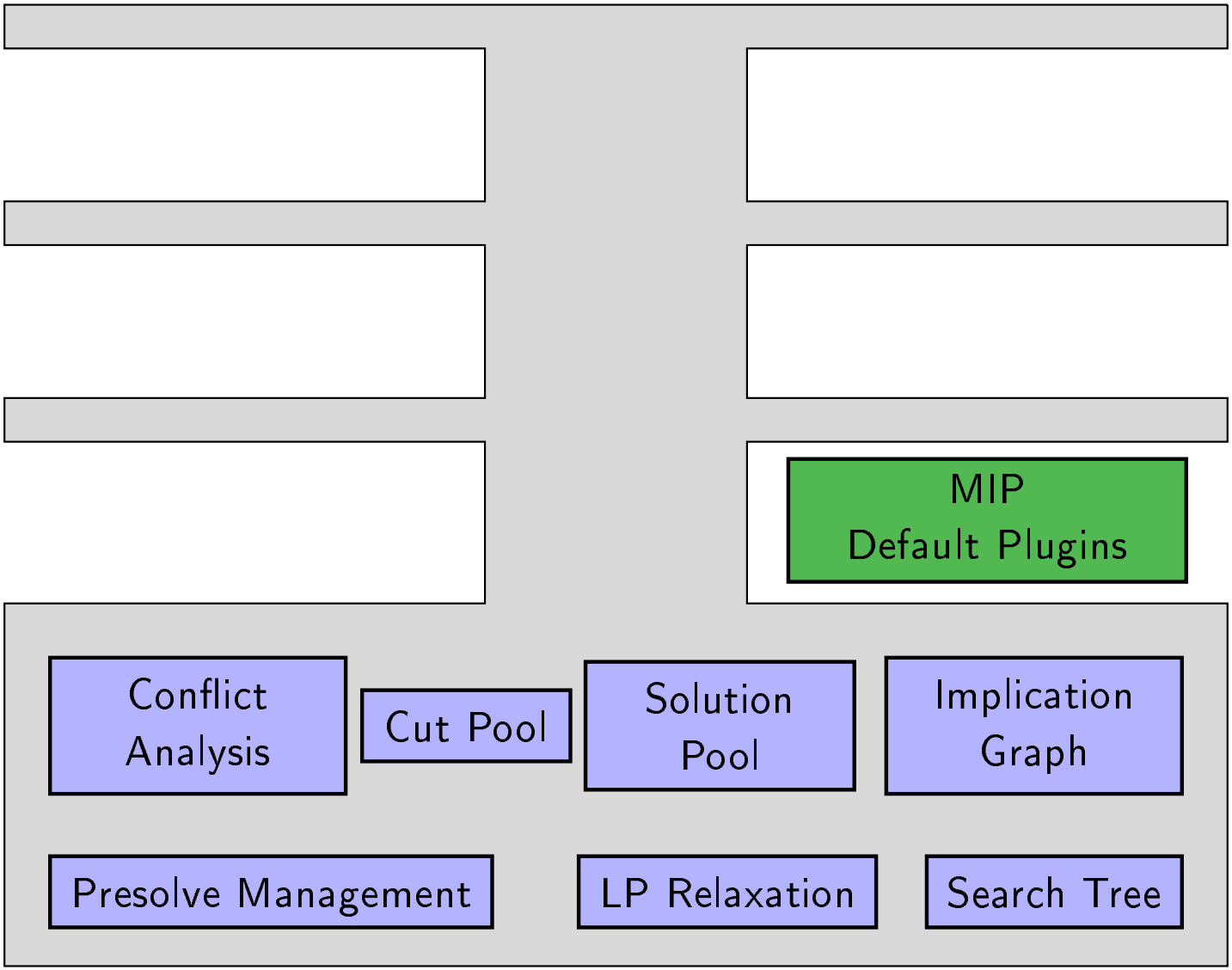
$$
\begin{aligned}
\min \quad & \sum_{e \in E} c_e \, x_e \\
s.t. \quad & \sum_{e \in \delta(v)} x_e = 2 \quad \forall \, v \in V \\
& \mathrm{nosubtour}(G, x) \\
& x_e \in \{0,1\} \quad \forall \, e \in E.
\end{aligned}
$$

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

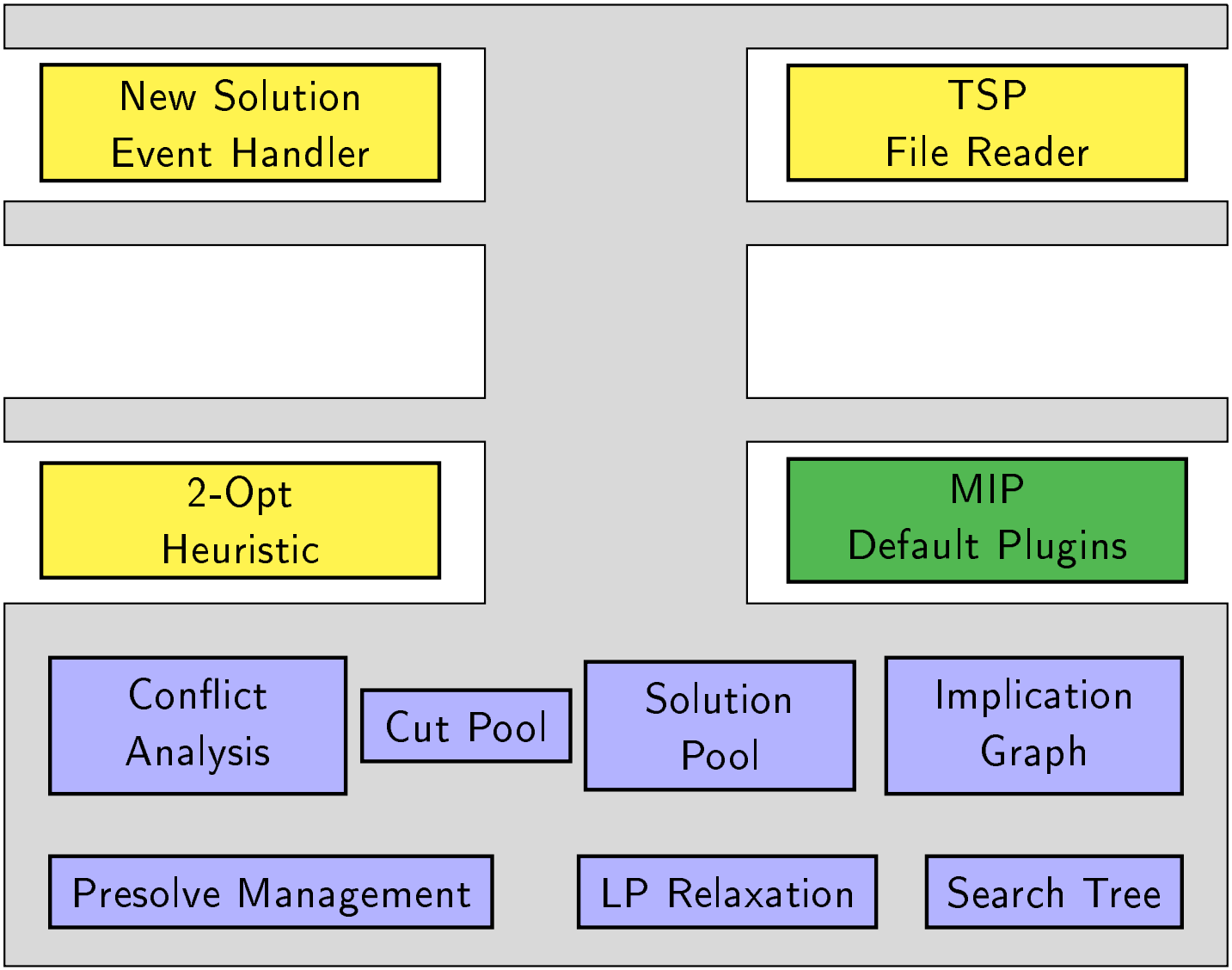
Exercise: TSP



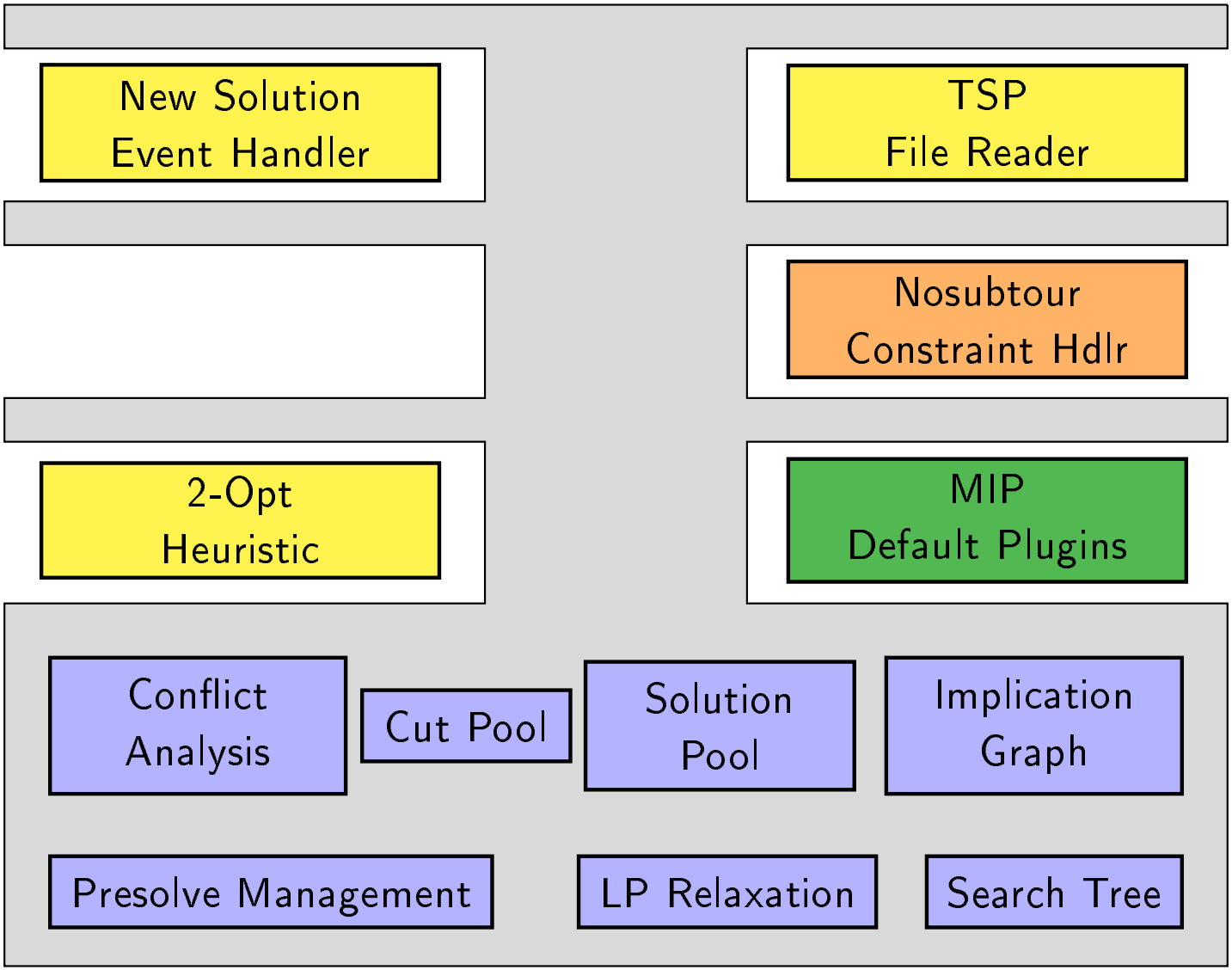
Exercise: TSP



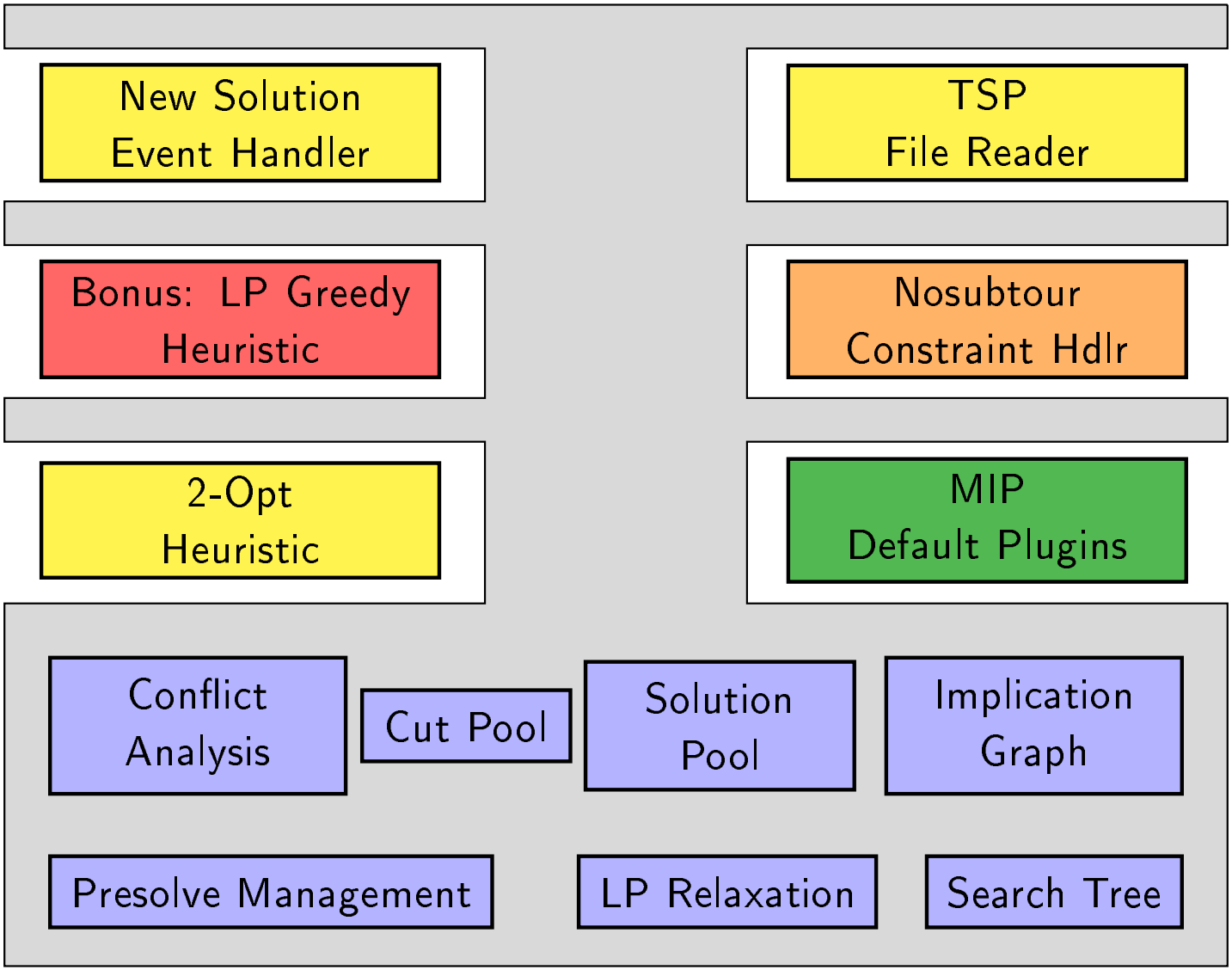
Exercise: TSP



Exercise: TSP



Exercise: TSP

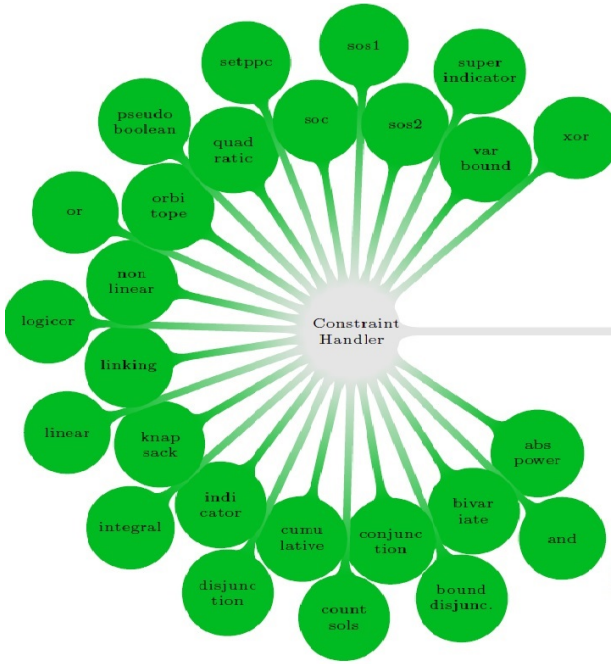


Exercise: TSP

Callback Methods

- Fundamental:
  - CONSCHECK
  - CONSENFOLP, CONSENFOPS
  - CONSLOCK

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



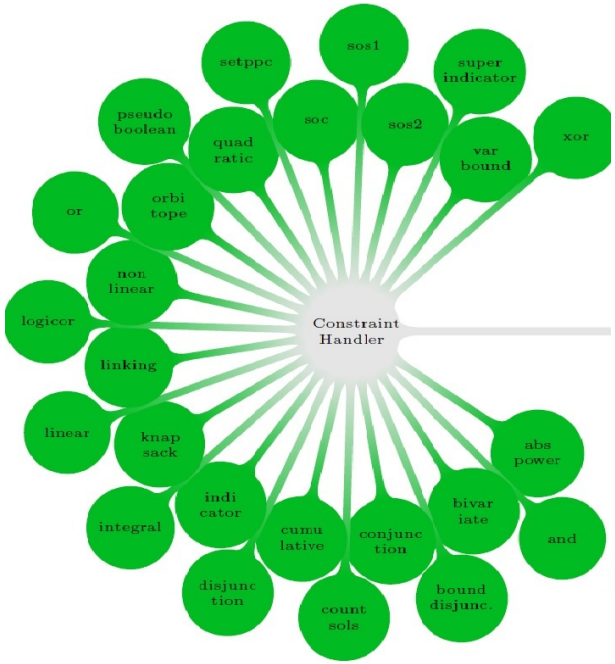
Further Ingredients

- Private data
- Interface methods
- Properties

Exercise: TSP

Callback Methods

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



Further Ingredients

- Private data
- Interface methods
- Properties

CONSCHECK

Feasibility Check

Given a point $y \in \mathbb{Q}^{|E|}$, decide whether y satisfies all subtour elimination constraints.

CONSCHECK

Feasibility Check

Given a point $y \in \mathbb{Q}^{|E|}$, decide whether y satisfies all subtour elimination constraints.

Check Priorities make this task easier:

- Constraint handlers check y in decreasing order of their check priority (CONSHDLR_CHECKPRIORITY).
- Check priority of $\mathrm{nosubtour}(G, x)$ smaller than linear and integral check priorities
- The solution y already satisfies integrality and (linear) degree constraints.

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

CONSCHECK

Feasibility Check

Given a point $y \in \mathbb{Q}^{|E|}$, decide whether y satisfies all subtour elimination constraints.

Check Priorities make this task easier:

- Constraint handlers check y in decreasing order of their check priority (CONSHDLR_CHECKPRIORITY).
- Check priority of $\mathrm{nosubtour}(G, x)$ smaller than linear and integral check priorities
- The solution y already satisfies integrality and (linear) degree constraints.

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

CONSCHECK

Feasibility Check

Given a point $y \in \mathbb{Q}^{|E|}$, decide whether y satisfies all subtour elimination constraints.

Check Priorities make this task easier:

- Constraint handlers check y in decreasing order of their check priority (CONSHDLR_CHECKPRIORITY).
- Check priority of $\mathrm{nosubtour}(G, x)$ smaller than linear and integral check priorities
- The solution y already satisfies integrality and (linear) degree constraints.

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

$\Rightarrow$ y is the characteristic vector of a collection of cycles when passed to the $\mathrm{nosubtour}$ constraint handler.

Understanding SCIP Callbacks

SCIP callback declarations are realized as macros:

What you see in `cons_nosubtour.c`:

```
static
SCIP_DECL_CONSCHECK(consCheckNosubtour)
{
    // Your Sophisticated Code
}
```

What the compiler sees (after preprocessor and some formatting):

```
static
SCIP_RETCODE consCheckNosubtour(SCIP* scip, SCIP_CONSHDLR* conshdlr,
   SCIP_CONS** conss, int nconss, SCIP_SOL* sol,
   SCIP_Bool checkintegrality, SCIP_Bool checklprows,
   SCIP_Bool printreason, SCIP_RESULT* result)
{
    // Your Sophisticated Code
}
```

`SCIP_DECL_CONSCHECK` is a macro defined in `type_cons.h`. It is responsible of expanding the callback function with the needed arguments and return value.

Understanding SCIP Callbacks even more

A compilable check callback

```
static
SCIP_DECL_CONSCHECK(consCheckNosubtour)
{
    /* TODO: add some more code */
    *result = SCIP_INFEASIBLE;

    return SCIP_OKAY;
}
```

- all callbacks must return a `SCIP_RETCODE`! Use `SCIP_OKAY` if everything worked
- often, callbacks require to correctly assign a `result`-pointer, examples:
  - `*result = SCIP_[IN]FEASIBLE`
  - `*result = SCIP_SEPARATED`
  - `*result = SCIP_DIDNOTFIND`

Find the full callback definition and possible return types in `type_<pluginname>.h`, eg, `type_cons.h` for constraint handlers or `type_heur.h` for primal heuristics.

The Graph implementation

The graph data structure for this exercise is defined in `GomoryHuTree.h`:

```
typedef struct Graph
{
   int       nuses;
   int       nnodes; /**< number of nodes of the graph */
   int       nedges; /**< number of edges */
   int       nedgesnonzero;

   GRAPHNODE *nodes; /**< array containing the nodes of the graph */

   GRAPHEDGE *edges; /**< array containing all halfedges ... */

} GRAPH;
```

The Graph implementation

Every graph edge is a half-edge in `GomoryHuTree.h`:

```
typedef struct GraphEdge
{
   double           cap;    /**< capacity used in maxflow */
   double           rcap;   /**< residual capacity used in maxflow */
   double           length; /**< length of the edge measured by some fixed metri

   struct GraphEdge *next;  /**< in incidence list of node from which edge is em
   struct GraphEdge *back;  /**< pointer to reverse edge */

   GRAPHNODE        *adjac; /**< pointer to adjacent node */

   SCIP_VAR*        var;

} GRAPHEDGE;
```

Every edge has a variable pointer `edge->var`.

The Graph implementation

Nodes of the graph only have two relevant members (for us):

```
1  typedef struct GraphNode
2  {
3     int              id;         /**< number of the node*/
4     ...
5     struct GraphEdge *first_edge;/**< in list of incident edges */
6     ...
7  } GRAPHNODE;
```

The Graph implementation

Question

How to traverse all incident edges of a node?

Answer:

```
GRAPHEDGE* edge = node->first_edge;
while( edge != NULL )
{
    /* do something with this edge */

    edge = edge->next;
}
```

The Gomory-Hu Black Box

Only one method from `GomoryHuTree.h` is relevant for us: `ghc_tree()`.
It has the following arguments:

- `GRAPH* gr`: the graph data structure
- `SCIP_Bool** cuts`: 2-dimensional array to return cuts, needs to be allocated by the user
- `int* ncuts`: pointer to store the number of violated cuts
- `double minviol` feasibility tolerance, use `SCIPfeastol(scip)`

The method returns TRUE if it found at least one violated cut, and `false` otherwise.

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

Documentation

- Constraint Integer Programming
  Tobias Achterberg, Ph.D. dissertation, TU Berlin, 2007

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

Please, . . .

- build groups of 3 people
  - laptop owner
  - C expert
  - IP expert
- raise your hand, if you get stuck.

Implementing Constraint Handlers in SCIP

Gregor Hendel
Ambros Gleixner, Felipe Serrano

September 30

Research Center Matheon
Mathematics for Key Technologies
Berlin Mathematical School
MODAL
Mathematical Optimization and Data Analysis Laboratories