Introduction to SCIP

ZIB: Fast Algorithms – Fast Computers

Konrad-Zuse-Zentrum für Informationstechnik Berlin

- non-university research institute of the state of Berlin
- Research Units:
  - numerical analysis and modeling
  - visualization and data analysis
  - optimization
  - scientific information systems
  - distributed algorithms and supercomputing
- more information: `http://www.zib.de`

Today's Schedule

09:00 – 10:30	Introduction and Overview
10:30 – 11:00	Coffee Break
11:00 – 12:30	Installation and Testing Environment
12:30 – 14-00	Lunch Break
14:00 – 15:00	Parameter Tuning
15:00 – 15:30	Coffee Break
15:30 – 17:30	Programming Exercise

WiFi:

- eduroam
- "Gast im ZIB" (no password)

SCIP – Solving Constraint Integer Programs

http://scip.zib.de

SCIP – Solving Constraint Integer Programs

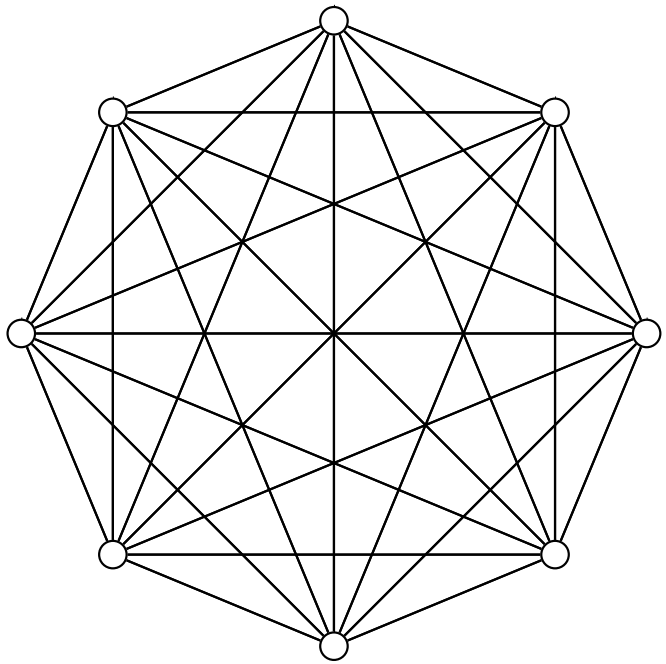
http://scip.zib.de

An example: the Traveling Salesman Problem

Definition (TSP)

Given a complete graph $G = (V, E)$ and distances d_e for all $e \in E$:

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



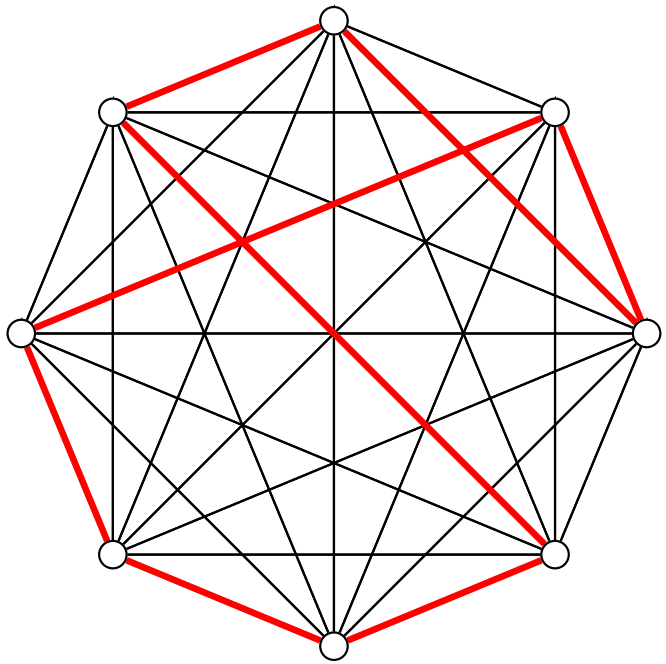
K_8

An example: the Traveling Salesman Problem

Definition (TSP)

Given a complete graph $G = (V, E)$ and distances d_e for all $e \in E$:

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



K_8

An example: the Traveling Salesman Problem

Definition (TSP)

Given a complete graph $G = (V, E)$ and distances d_e for all $e \in E$:

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

What is a Constraint Integer Program?

Mixed Integer Program

Objective function:

- ▷ linear function

Feasible set:

- ▷ described by linear constraints

Variable domains:

- ▷ real or integer values

$$\begin{aligned} \min \quad & c^T x \\ s.t. \quad & Ax \leq b \\ & (x_I, x_C) \in \mathbb{Z}^I \times \mathbb{R}^C \end{aligned}$$

Constraint Program

Objective function:

- ▷ arbitrary function

Feasible set:

- ▷ given by arbitrary constraints

Variable domains:

- ▷ arbitrary (usually finite)

$$\begin{aligned} \min \quad & c(x) \\ s.t. \quad & x \in F \\ & (x_I, x_N) \in \mathbb{Z}^I \times X \end{aligned}$$

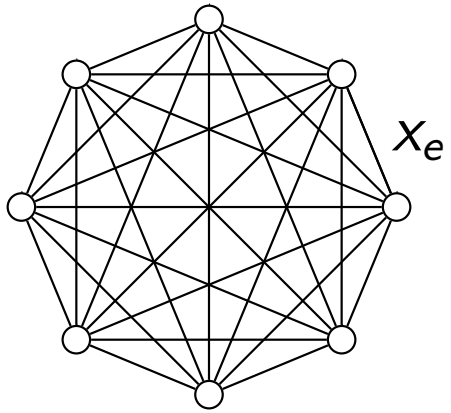
TSP – Integer Programming Formulation

Given

- complete graph $G = (V, E)$
- distances $d_e > 0$ for all $e \in E$

Binary variables

- $x_e = 1$ if edge e is used

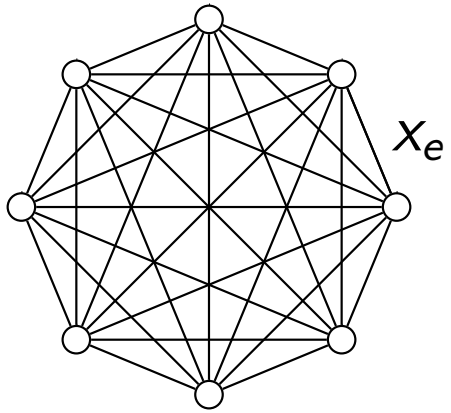


K_8

TSP – Integer Programming Formulation

Given

- complete graph $G = (V, E)$
- distances $d_e > 0$ for all $e \in E$



K_8

Binary variables

- $x_e = 1$ if edge e is used

$$\min \quad \sum_{e \in E} d_e \, x_e$$

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

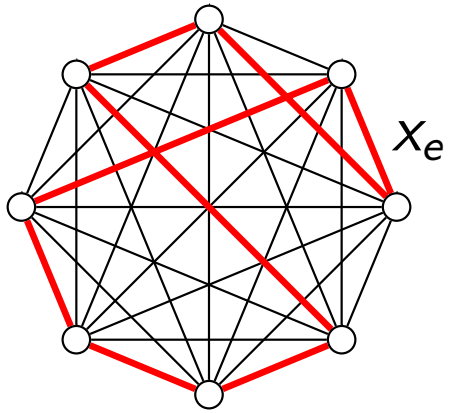
$$\sum_{e \in \delta(S)} x_e \geq 2 \qquad \forall S \subset V, S \neq \varnothing$$

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

TSP – Integer Programming Formulation

Given

- complete graph $G = (V, E)$
- distances $d_e > 0$ for all $e \in E$



K_8

Binary variables

- $x_e = 1$ if edge e is used

$$\min \quad \sum_{e \in E} d_e \, x_e$$

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

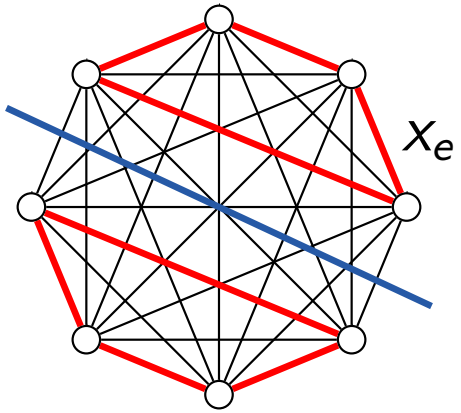
$$\sum_{e \in \delta(S)} x_e \geq 2 \qquad \forall S \subset V, S \neq \varnothing$$

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

TSP – Integer Programming Formulation

Given

- complete graph $G = (V, E)$
- distances $d_e > 0$ for all $e \in E$



K_8

Binary variables

- $x_e = 1$ if edge e is used

$$\min \quad \sum_{e \in E} d_e \, x_e$$

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

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

subtour elimination

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

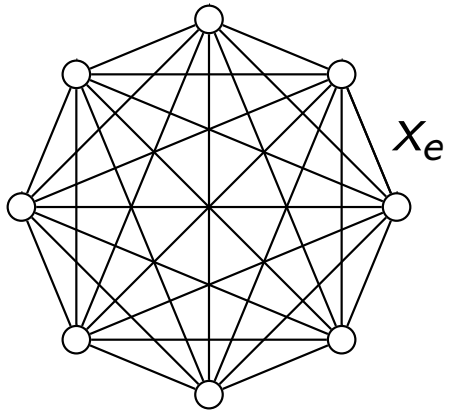
TSP – Integer Programming Formulation

Given

- complete graph $G = (V, E)$
- distances $d_e > 0$ for all $e \in E$

Binary variables

- $x_e = 1$ if edge e is used



K_8

$$\min \quad \sum_{e \in E} d_e \, x_e \qquad \text{distance}$$

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

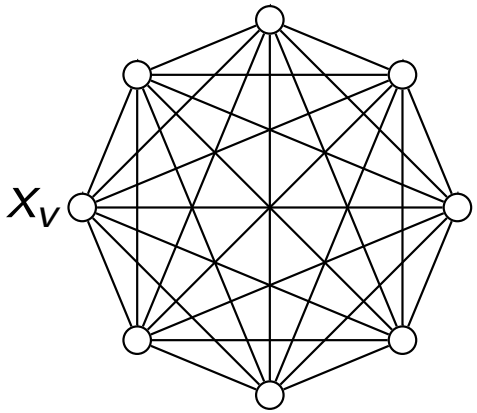
TSP – Constraint Programming Formulation

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

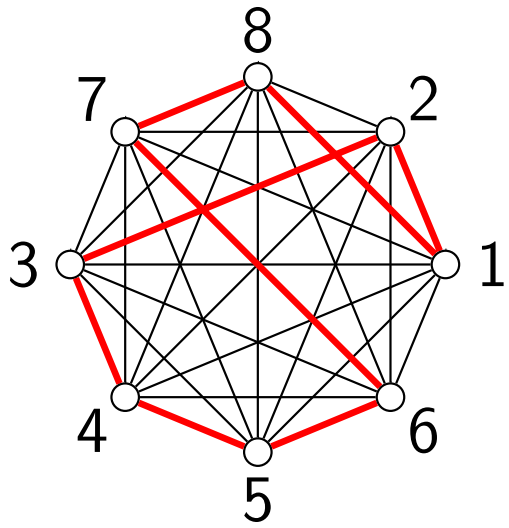
TSP – Constraint Programming Formulation

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



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

What is a Constraint Integer Program?

Constraint Integer Program

Objective function:

- ▷ linear function

Feasible set:

- ▷ described by arbitrary constraints

Variable domains:

- ▷ real or integer values

When no more branching possible:

- ▷ CIP becomes an LP/NLP

$$\begin{aligned} \min \quad & c^T x \\ s.t. \quad & x \in F \\ & (x_I, x_C) \in \mathbb{Z}^I \times \mathbb{R}^C \end{aligned}$$

Remark:

- ▷ arbitrary objective or variables modeled by constraints

What is a Constraint Integer Program?

Constraint Integer Program

Objective function:

- ▷ linear function

Feasible set:

- ▷ described by arbitrary constraints

Variable domains:

- ▷ real or integer values

When no more branching possible:

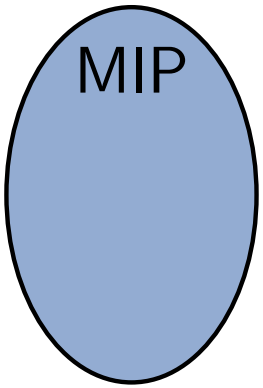
- ▷ CIP becomes an LP/NLP

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

(CIP formulation of TSP)

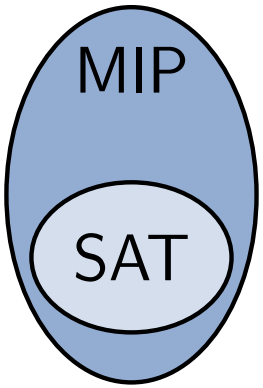
Single nosubtour constraint rules out subtours (e.g. by domain propagation). It may also separate subtour elimination inequalities.

- Mixed Integer Programs



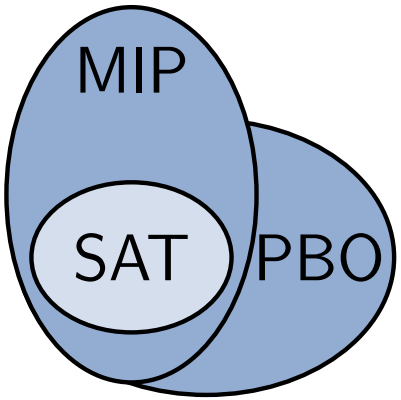
Constraint Integer Programming

- Mixed Integer Programs
- SATisfiability problems



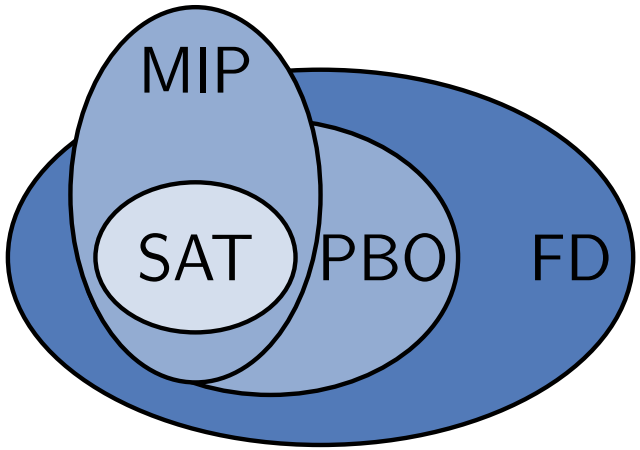
Constraint Integer Programming

- Mixed Integer Programs
- SATisfiability problems
- Pseudo-Boolean Optimization



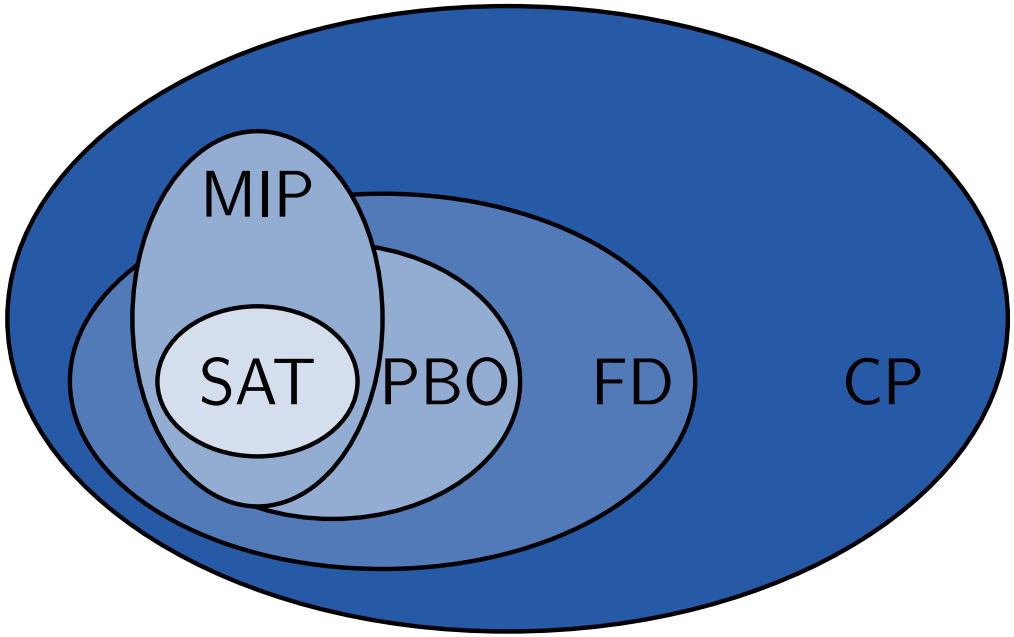
Constraint Integer Programming

- **M**ixed **I**nteger **P**rograms
- **SAT**isfiability problems
- **P**seudo-**B**oolean **O**ptimization
- **F**inite **D**omain



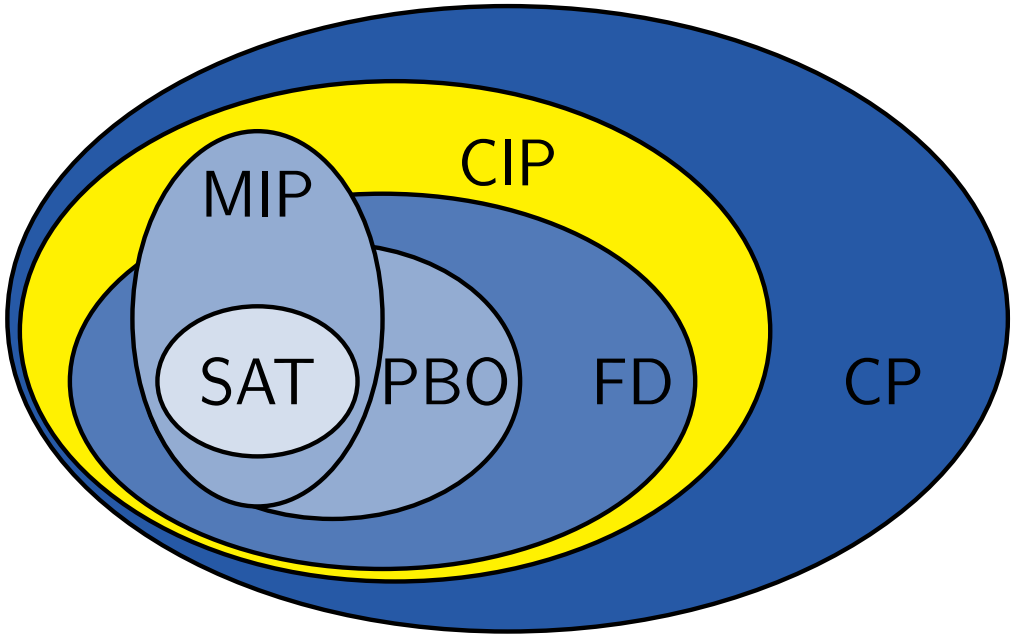
Constraint Integer Programming

- Mixed Integer Programs
- SATisfiability problems
- Pseudo-Boolean Optimization
- Finite Domain
- Constraint Programming



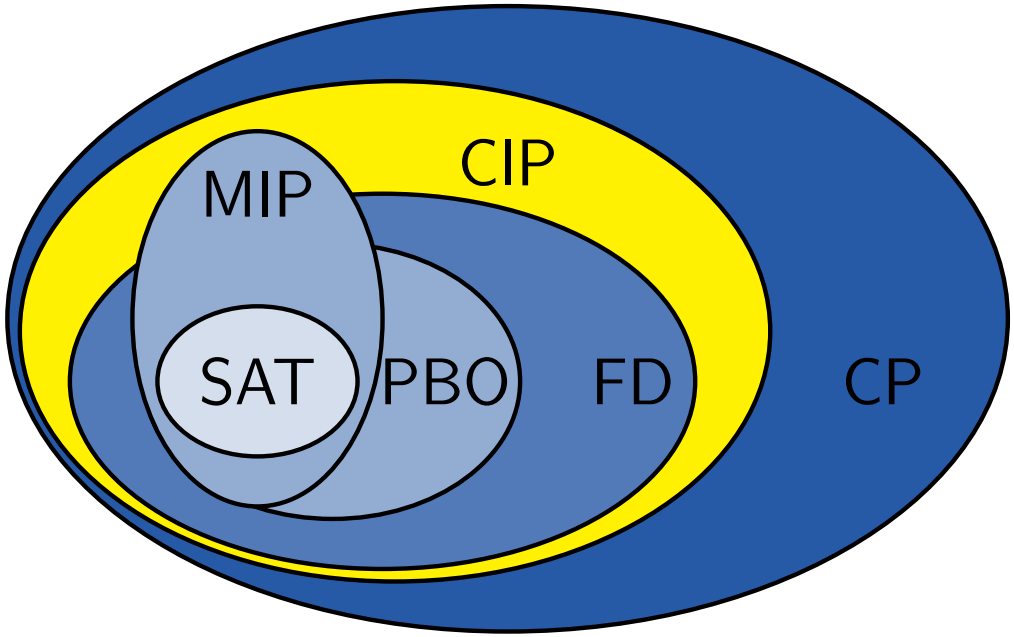
Constraint Integer Programming

- Mixed Integer Programs
- SATisfiability problems
- Pseudo-Boolean Optimization
- Finite Domain
- Constraint Programming
- Constraint Integer Programming



Constraint Integer Programming

- Mixed Integer Programs
- SATisfiability problems
- Pseudo-Boolean Optimization
- Finite Domain
- Constraint Programming
- Constraint Integer Programming



Relation to CP and MIP

- Every MIP is a CIP. *"MIP $\subsetneq$ CIP"*
- Every CP over a finite domain space is a CIP. *"FD $\subsetneq$ CIP"*

SCIP – Solving Constraint Integer Programs

http://scip.zib.de

How do we solve CIPs?

MIP

- LP relaxation
- cutting planes

CP

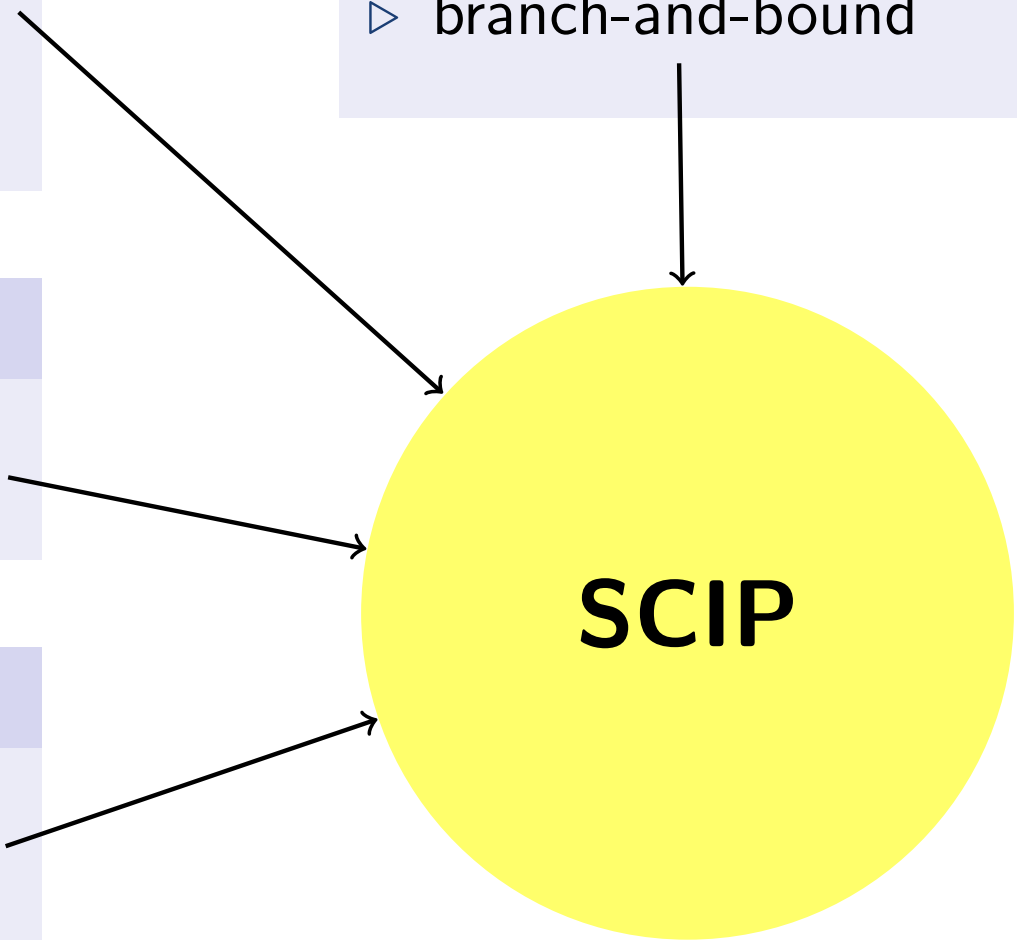
- domain propagation

SAT

- conflict analysis
- periodic restarts

MIP, CP, and SAT

- branch-and-bound



SCIP (Solving Constraint Integer Programs) ...

- provides a full-scale MIP and MINLP solver,
- is constraint based,
- incorporates
  - CP features (domain propagation),
  - MIP features (cutting planes, LP relaxation), and
  - SAT-solving features (conflict analysis, restarts),
- is a branch-cut-and-price framework,
- has a modular structure via plugins,
- is free for academic purposes,
- and is available in source-code under http://scip.zib.de !

Some more facts about SCIP

- interface and usability
  - user-friendly interactive shell
  - interfaces to AMPL, GAMS, ZIMPL, MATLAB, Python and Java
  - C++ wrapper classes
  - LP solvers: CLP, CPLEX, Gurobi, MOSEK, QSopt, SoPlex, Xpress
  - over 1 600 parameters and 15 emphasis settings
- documentation and guidelines
  - more than 450 000 lines of C code, 20% documentation
    - 30 000 assertions, 4 000 debug messages
  - HowTos: plugins types, debugging, automatic testing
  - 11 examples illustrating the use of SCIP
  - active mailing list (280 members)
- about 5000 downloads per year

SCIP Optimization Suite

- Toolbox for generating and solving constraint integer programs
- free for academic use, available in source code

SCIP Optimization Suite

- Toolbox for generating and solving constraint integer programs
- free for academic use, available in source code

ZIMPL

- model and generate LPs, MIPs, and MINLPs

SCIP

- MIP, MINLP and CIP solver, branch-cut-and-price framework

SoPlex

- revised primal and dual simplex algorithm

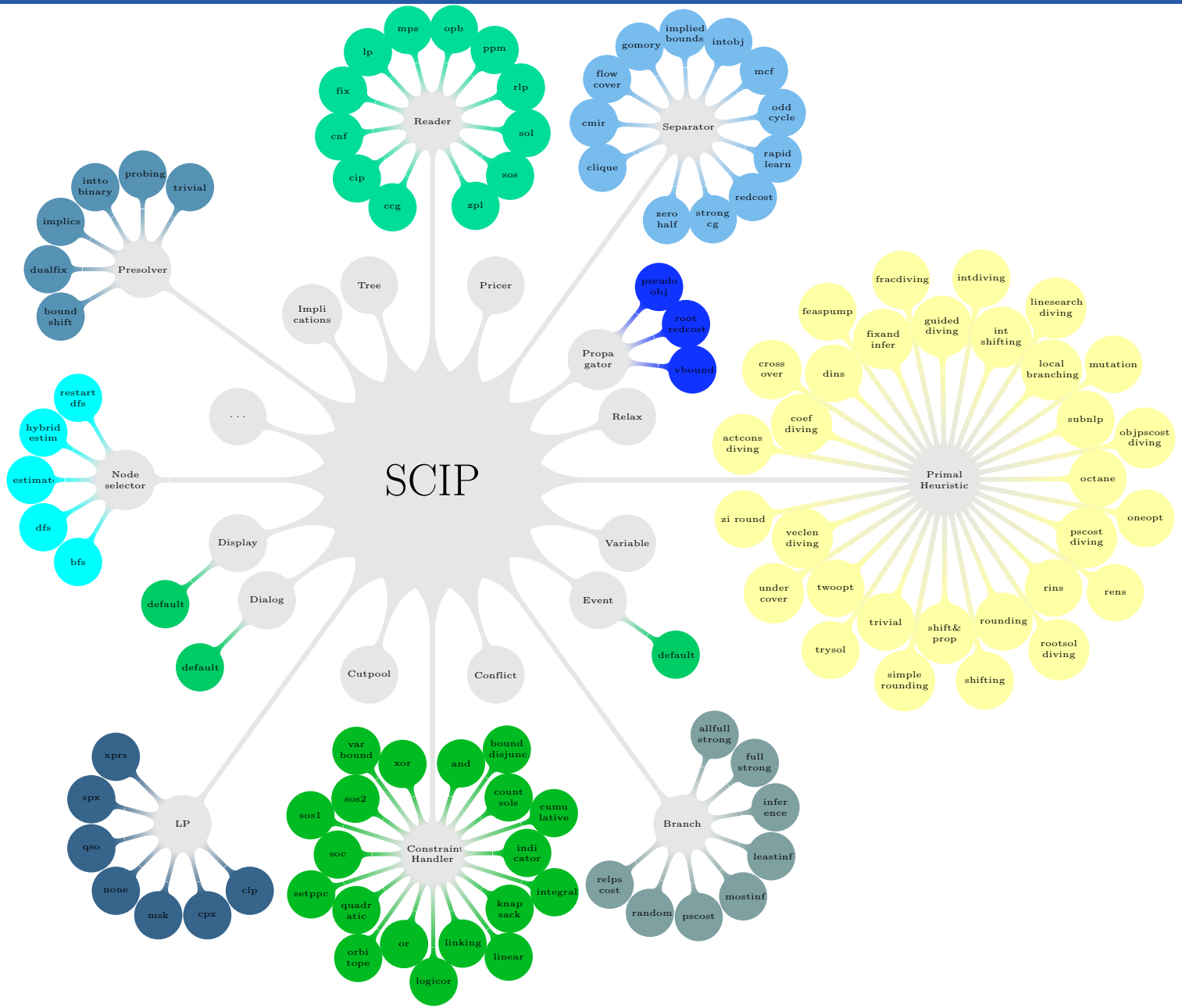
GCG → see talk tomorrow 9:40

- generic branch-cut-and-price solver

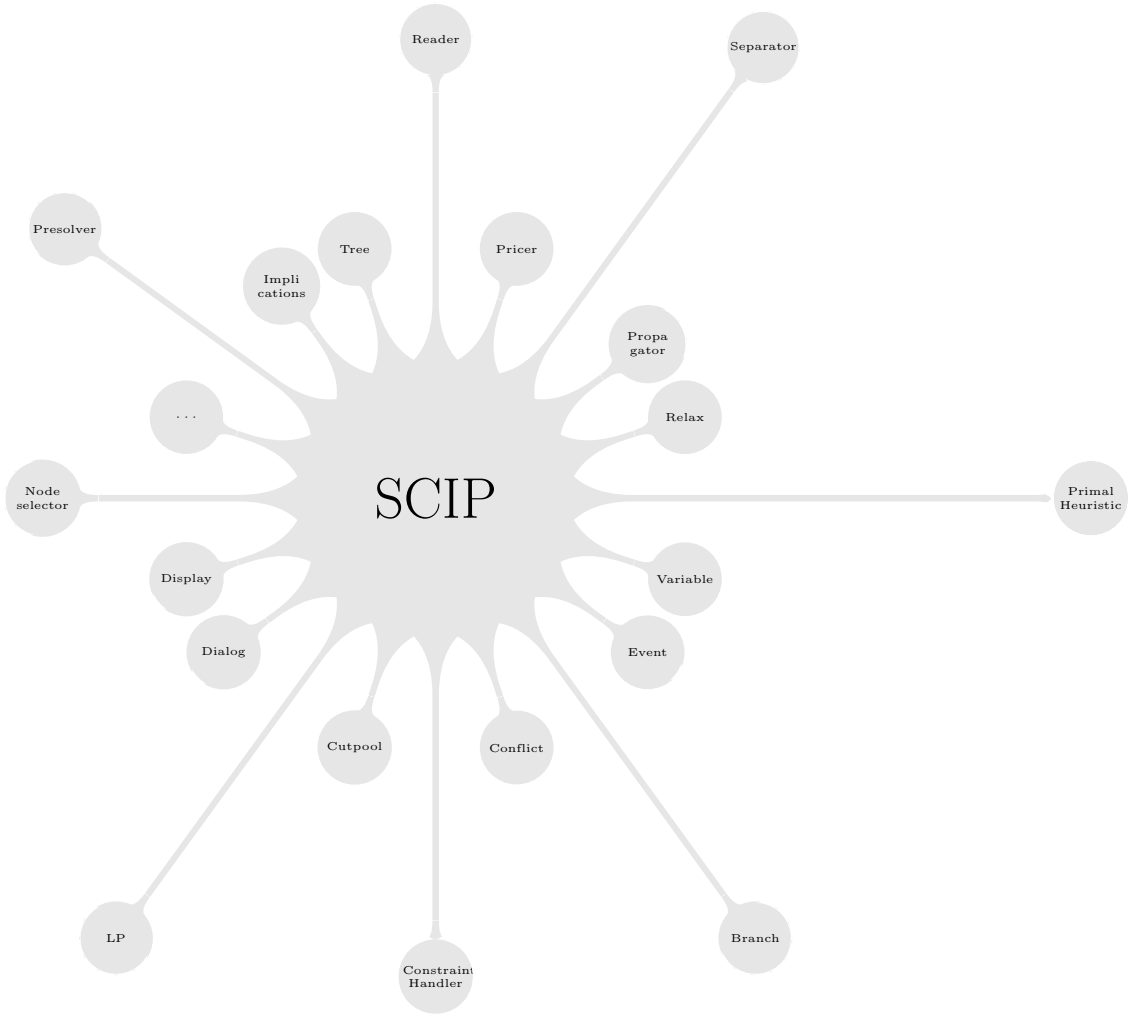
UG → see talk tomorrow 10:20

- framework for parallelization of MIP and MINLP solvers

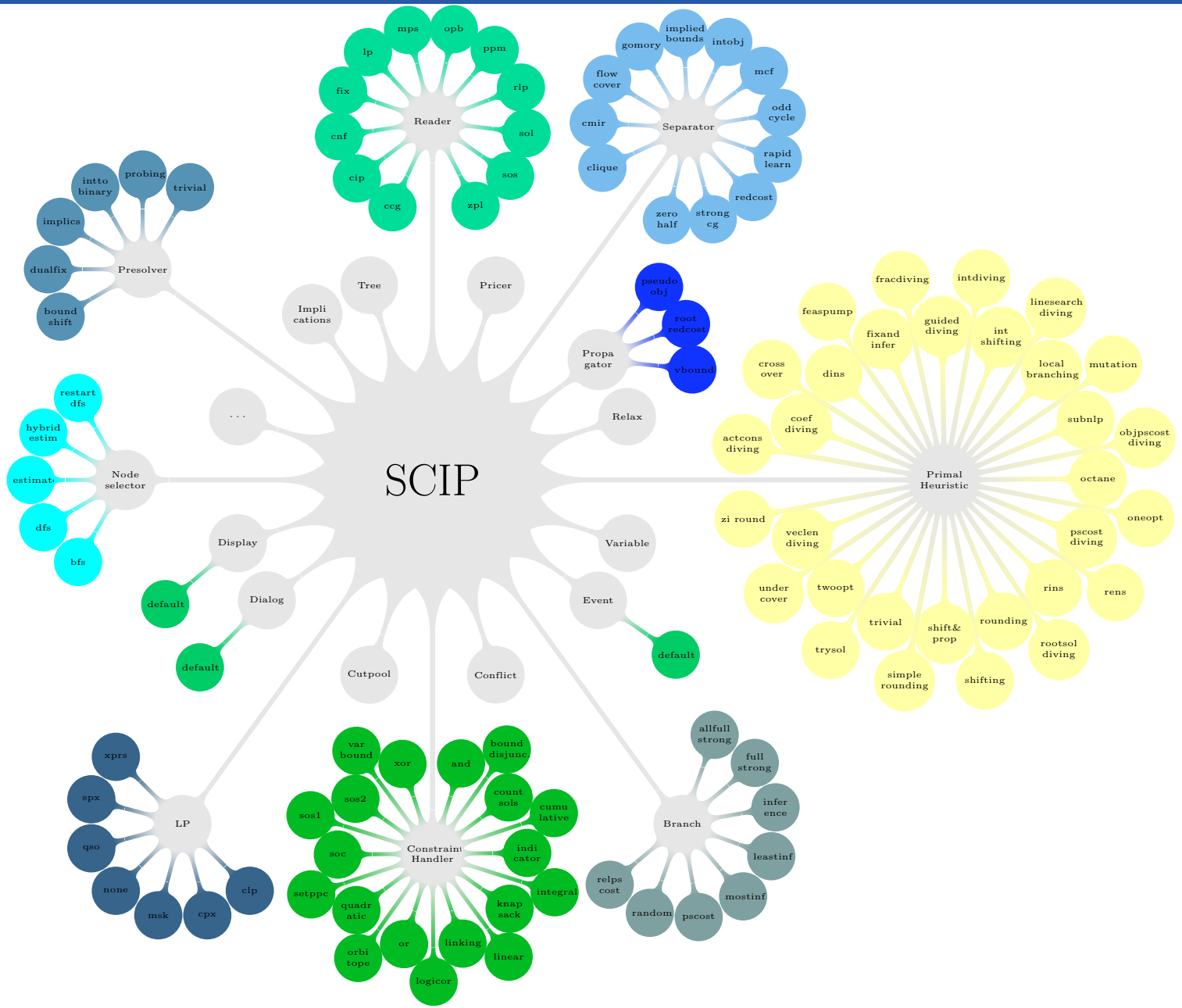
Structure of SCIP



Structure of SCIP



Structure of SCIP



Plugin based design

SCIP core

- branching tree
- variables
- conflict analysis
- solution pool
- cut pool
- statistics
- clique table
- implication graph
- ...

Plugin based design

SCIP core

- branching tree
- variables
- conflict analysis
- solution pool
- cut pool
- statistics
- clique table
- implication graph
- ...

Plugins

- external callback objects
- interact with the framework through a very detailed interface

Plugin based design

SCIP core

- branching tree
- variables
- conflict analysis
- solution pool
- cut pool
- statistics
- clique table
- implication graph
- ...

Plugins

- external callback objects
- interact with the framework through a very detailed interface
- SCIP knows for each plugin type:
  - the number of available plugins
  - priority defining the calling order (usually)
- SCIP does not know any structure behind a plugin

$\Rightarrow$ plugins are black boxes for the SCIP core

Plugin based design

SCIP core

- branching tree
- variables
- conflict analysis
- solution pool
- cut pool
- statistics
- clique table
- implication graph
- ...

Plugins

- external callback objects
- interact with the framework through a very detailed interface
- SCIP knows for each plugin type:
  - the number of available plugins
  - priority defining the calling order (usually)
- SCIP does not know any structure behind a plugin

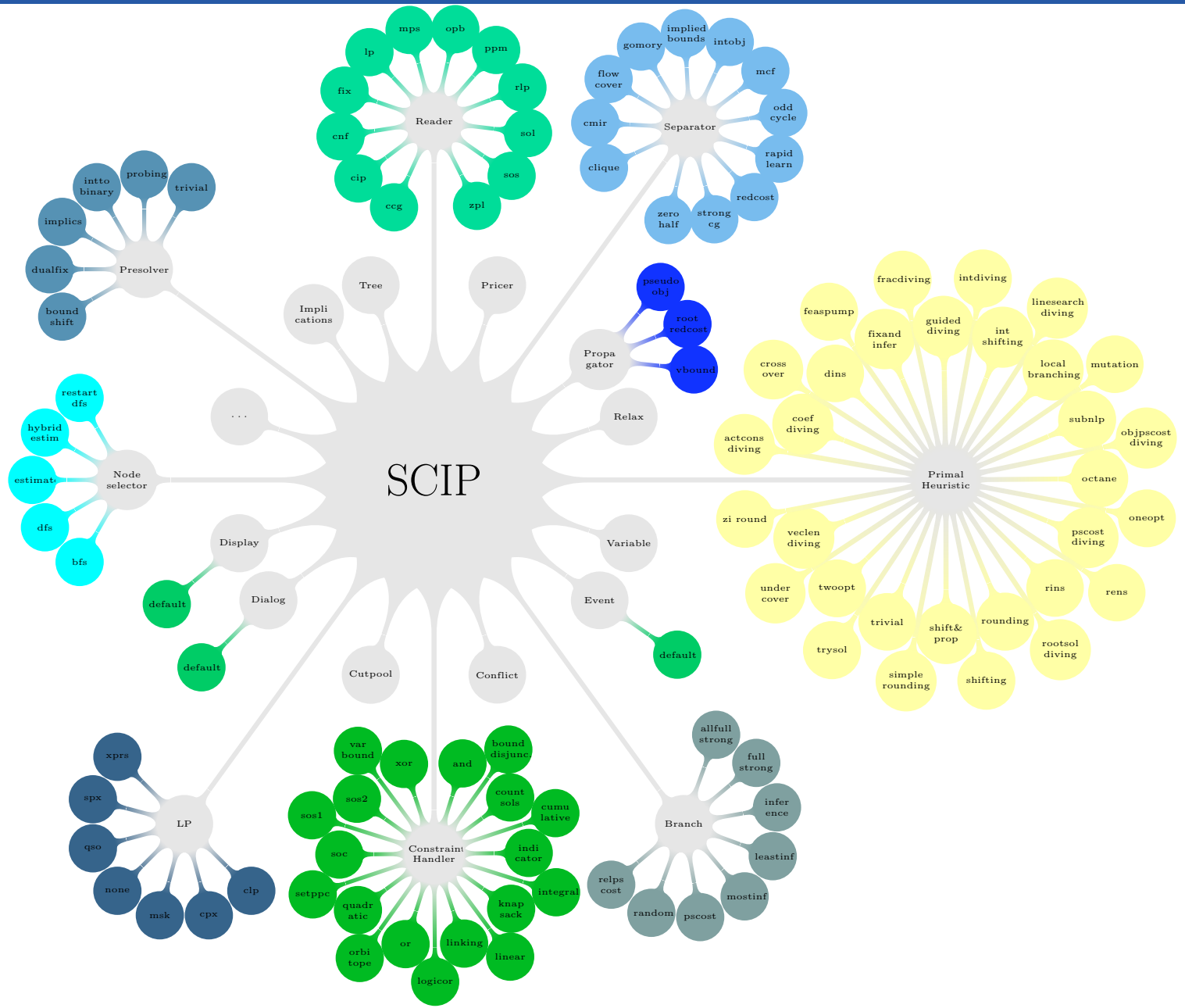
$\Rightarrow$ plugins are black boxes for the SCIP core

$\Rightarrow$ Very flexible branch-and-bound based search algorithm

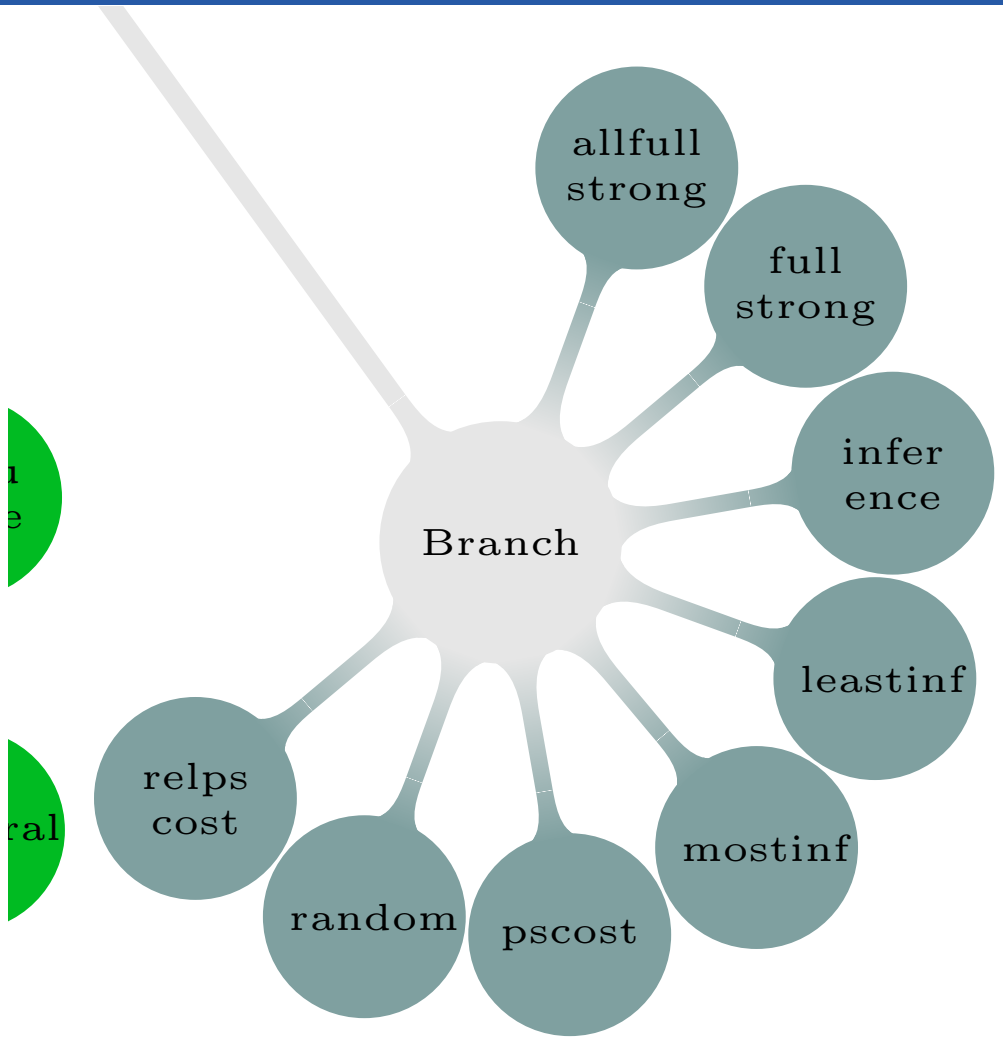
Types of Plugins

- 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
- ...

A closer look: branching rules



A closer look: branching rules



What does SCIP know about branching rules?

- SCIP knows the number of available branching rules
- each branching rule has a priority
- SCIP calls the branching rule in decreasing order of priority
- the interface defines the possible results of a call:
  - branched
  - reduced domains
  - added constraints
  - detected cutoff
  - did not run

How does SCIP call a branching rule?

```
/* start timing */
SCIPclockStart(branchrule->branchclock, set);

/* call external method */
SCIP_CALL( branchrule->branchexeclp(set->scip, branchrule,
   allowaddcons, result) );

/* stop timing */
SCIPclockStop(branchrule->branchclock, set);

/* evaluate result */
if( *result != SCIP_CUTOFF
   && *result != SCIP_CONSADDED
   && *result != SCIP_REDUCEDDOM
   && *result != SCIP_SEPARATED
   && *result != SCIP_BRANCHED
   && *result != SCIP_DIDNOTRUN )
{
   SCIPerrorMessage(
      "branching␣rule␣<%s>␣returned␣invalid␣result␣code␣<%d>␣from␣LP␣\
␣␣␣␣␣␣␣␣solution␣branching\n",
      branchrule->name, *result);
   return SCIP_INVALIDRESULT;
}
```

What can a plugin access?

Plugins are allowed to access all global (core) information

- branching tree
- variables
- conflict analysis
- solution pool
- cut pool
- statistics
- clique table
- implication graph
- ...

What can a plugin access?

Plugins are allowed to access all global (core) information

- branching tree
- variables
- conflict analysis
- solution pool
- cut pool
- statistics
- clique table
- implication graph
- ...

Ideally, plugins should not access data of other plugins!!!

What can a plugin access?

Plugins are allowed to access all global (core) information

- branching tree
- variables
- conflict analysis
- solution pool
- cut pool
- statistics
- clique table
- implication graph
- ...

Ideally, plugins should not access data of other plugins!!!

Branching Rules

- LP solution
- variables
- statistics

Constraint Handlers

Constraint handlers

- most powerful plugins in SCIP
- define the feasible region
- a single constraint may represent a whole set of inequalities

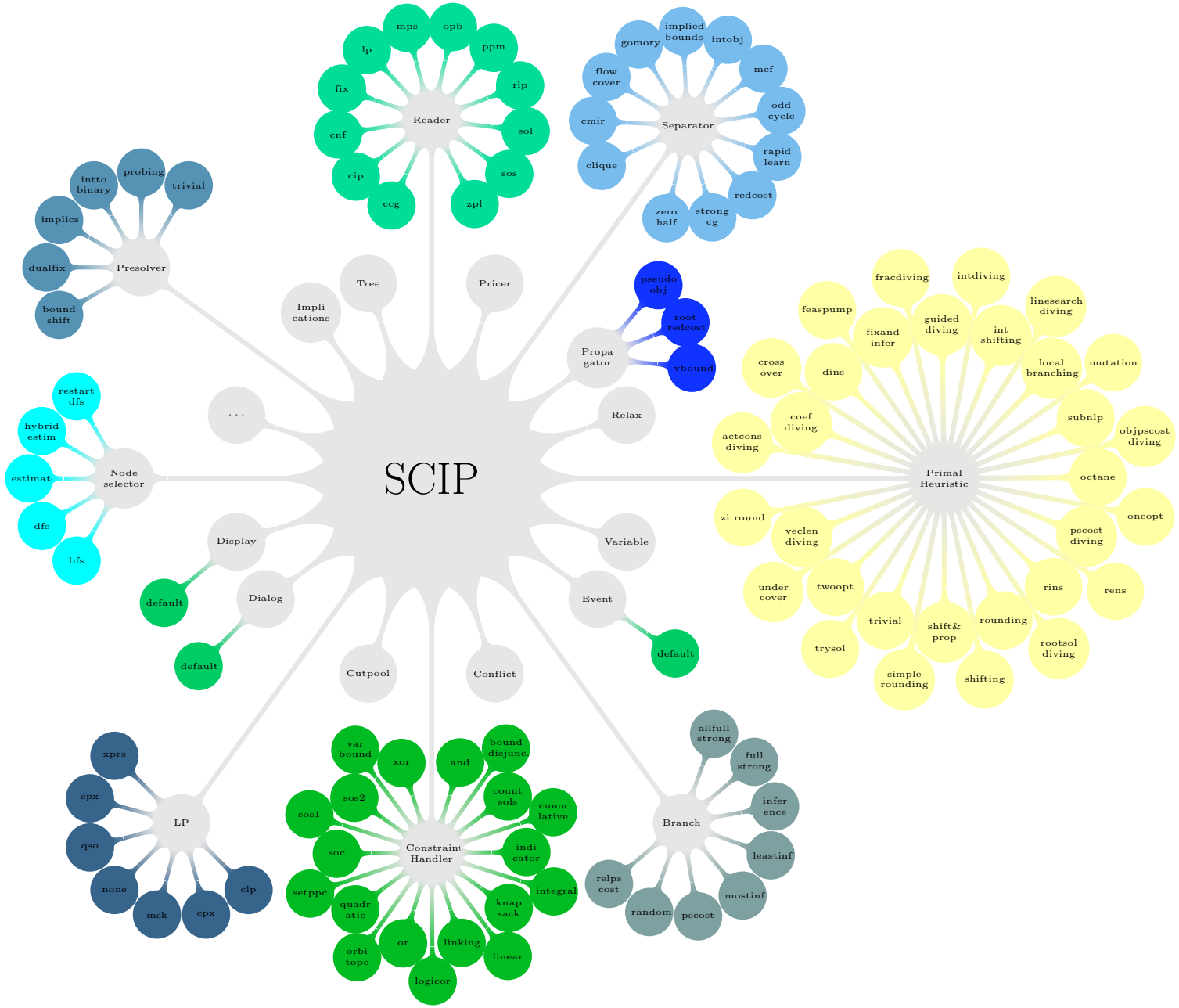
Functions

- check and enforce feasibility of solutions
- can add linear representation to LP relaxation
- constraint-specific presolving, domain propagation, separation

Result

- SCIP is constraint based
  - Advantage: flexibility
  - Disadvantage: limited global view

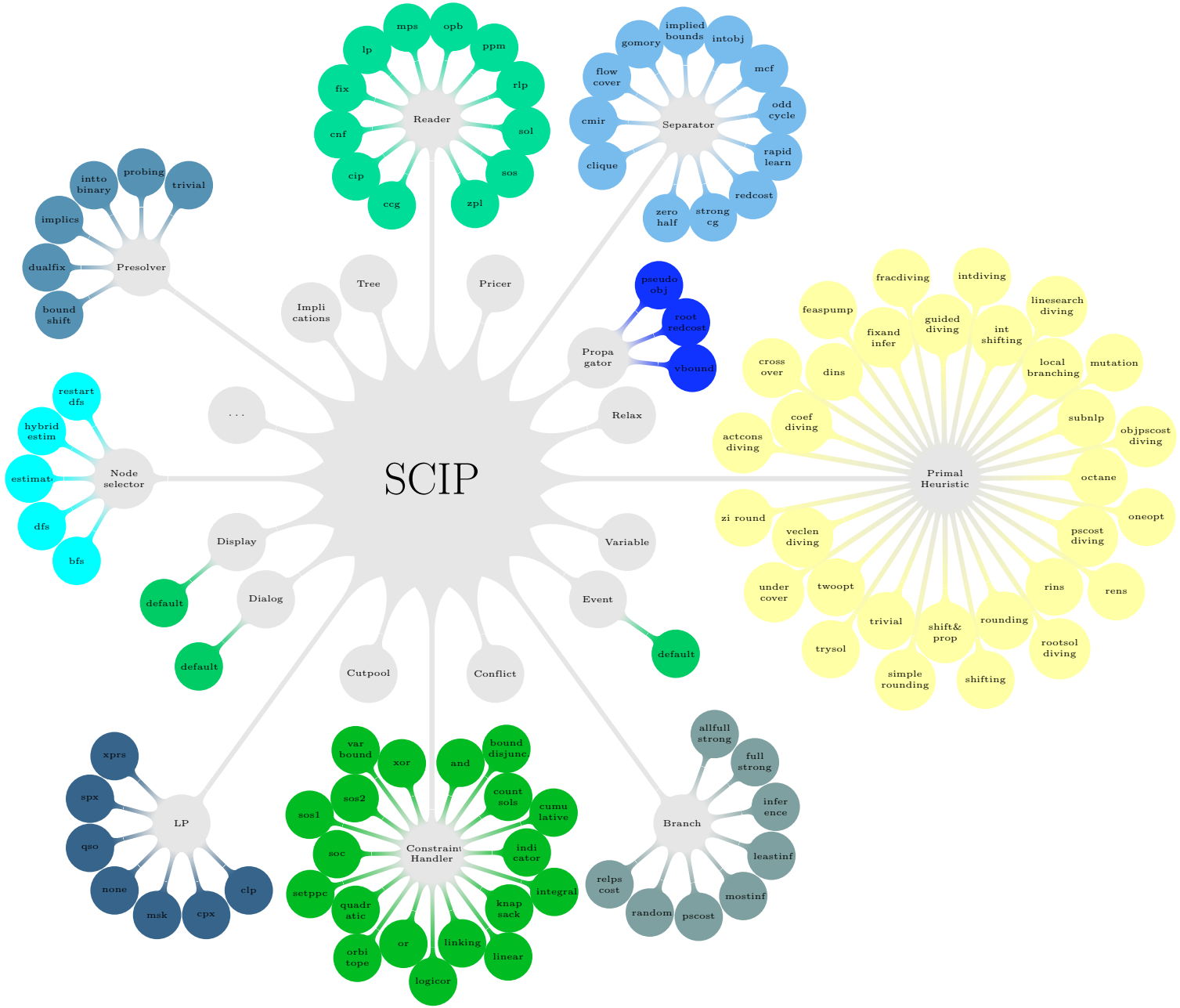
Default Plugins



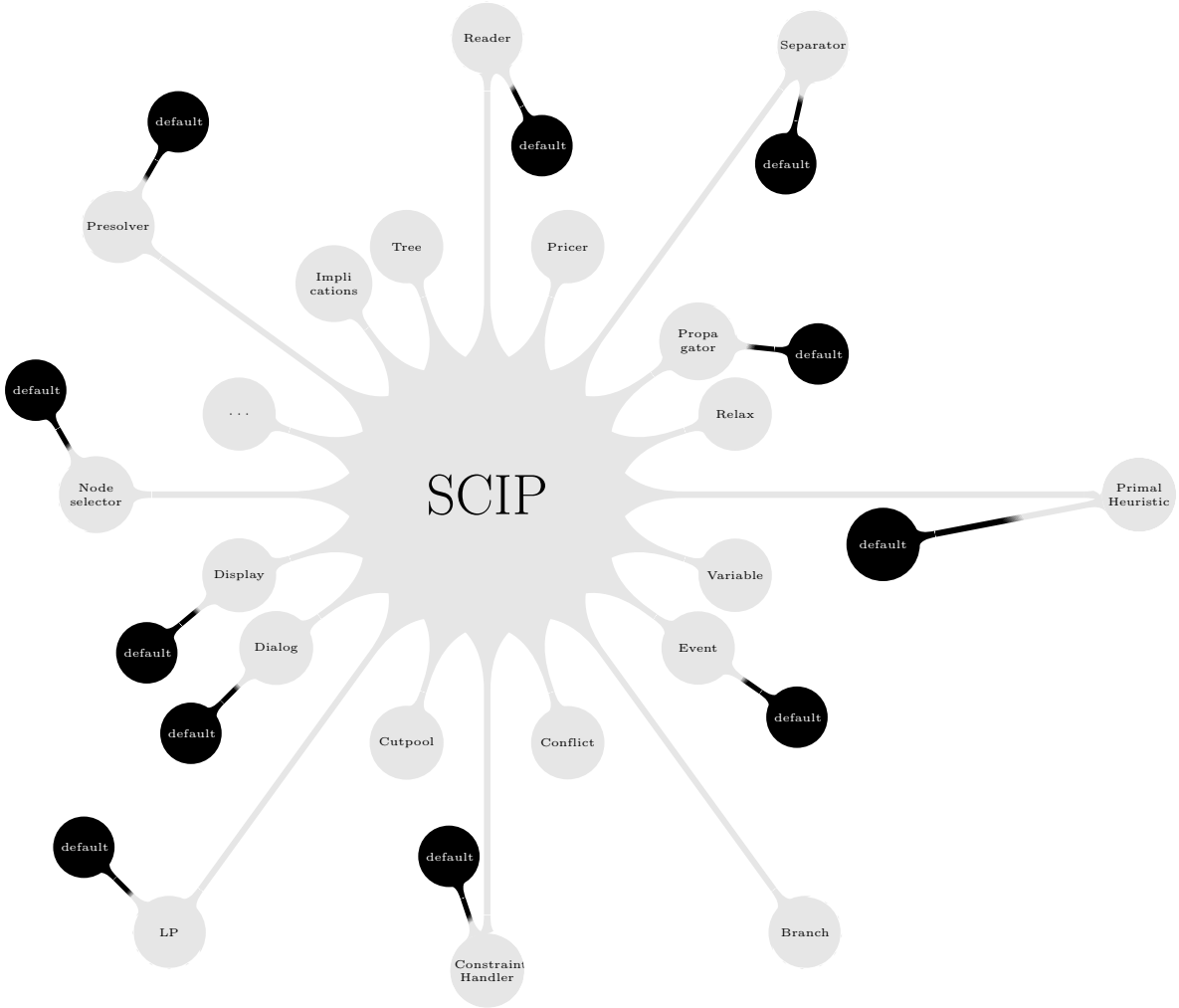
Number of default plugins

	SCIP Version									
Plugin	0.7	0.8	0.9	1.0	1.1	1.2	2.0	2.1	3.0	3.1
Branching Rules	6	7	8	8	8	8	8	8	8	9
Constraint Handlers	10	10	11	11	14	16	23	25	26	26
Node Selectors	3	7	3	5	5	5	5	5	5	7
Presolvers	2	7	5	5	6	6	6	5	9	9
Primal Heuristics	9	14	21	23	24	27	31	33	35	38
Propagators	0	1	2	2	2	2	3	5	7	8
Readers	2	4	6	6	11	13	15	16	17	18
Separators	3	6	7	8	10	10	12	13	13	13

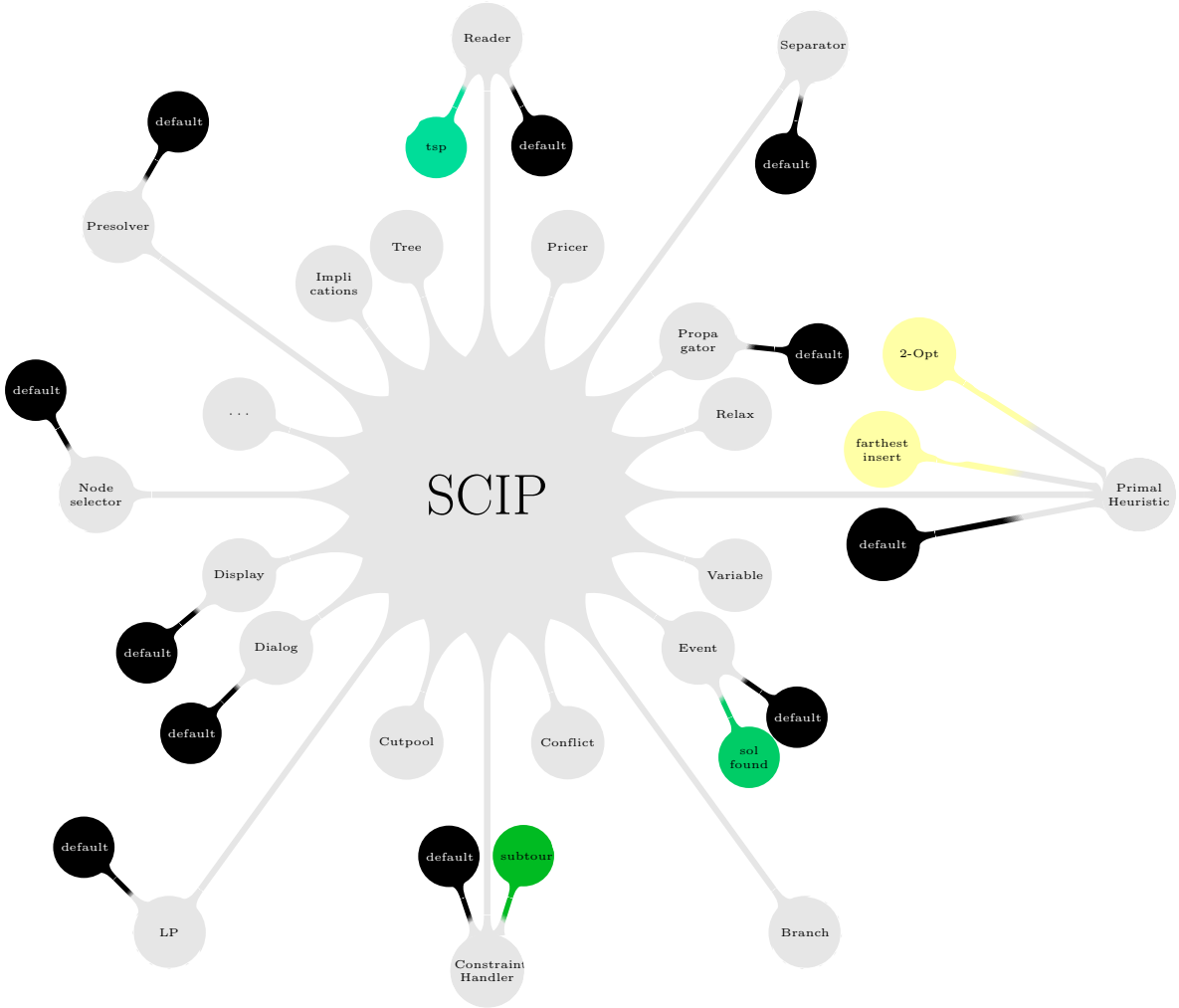
Extending SCIP



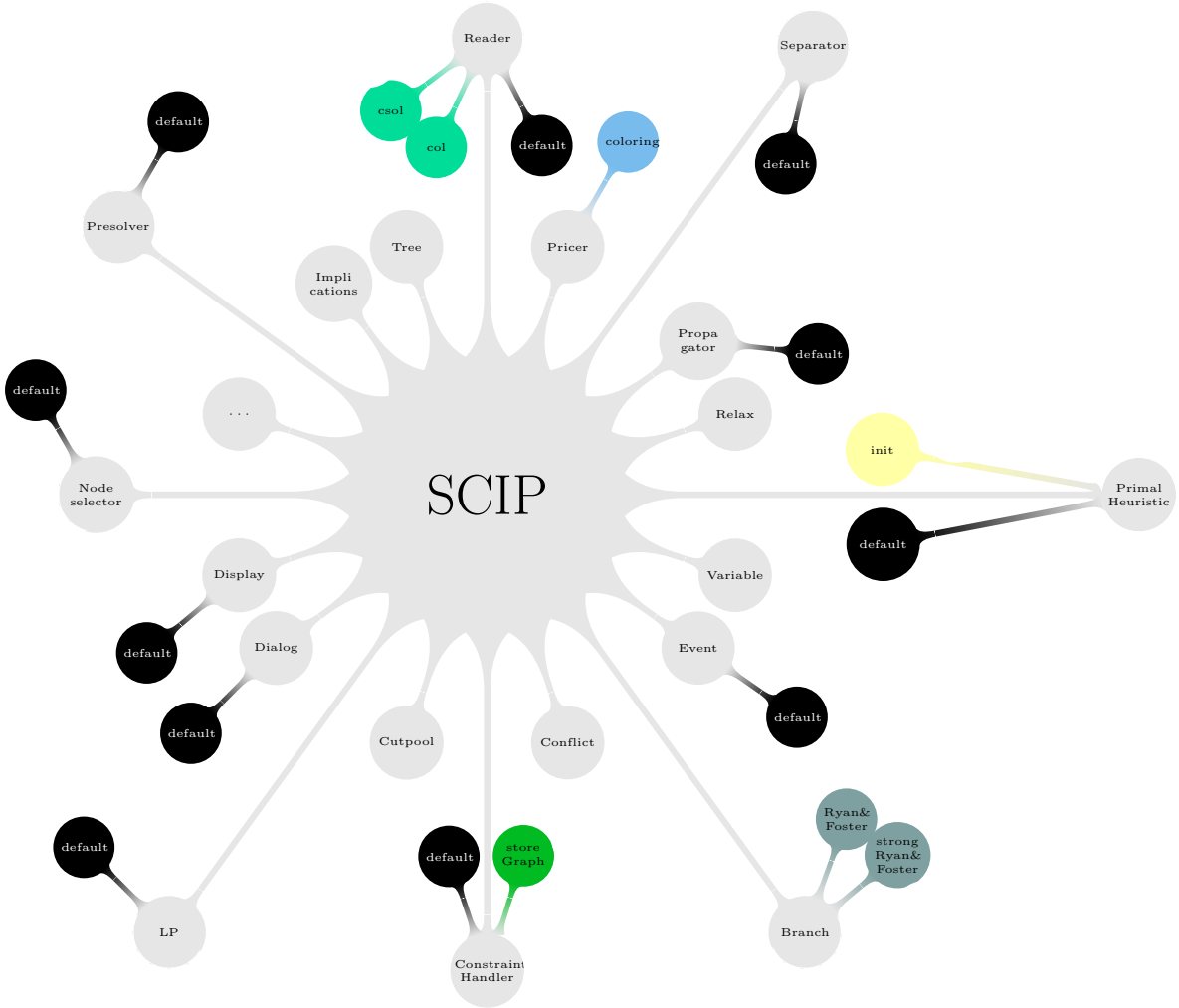
Extending SCIP



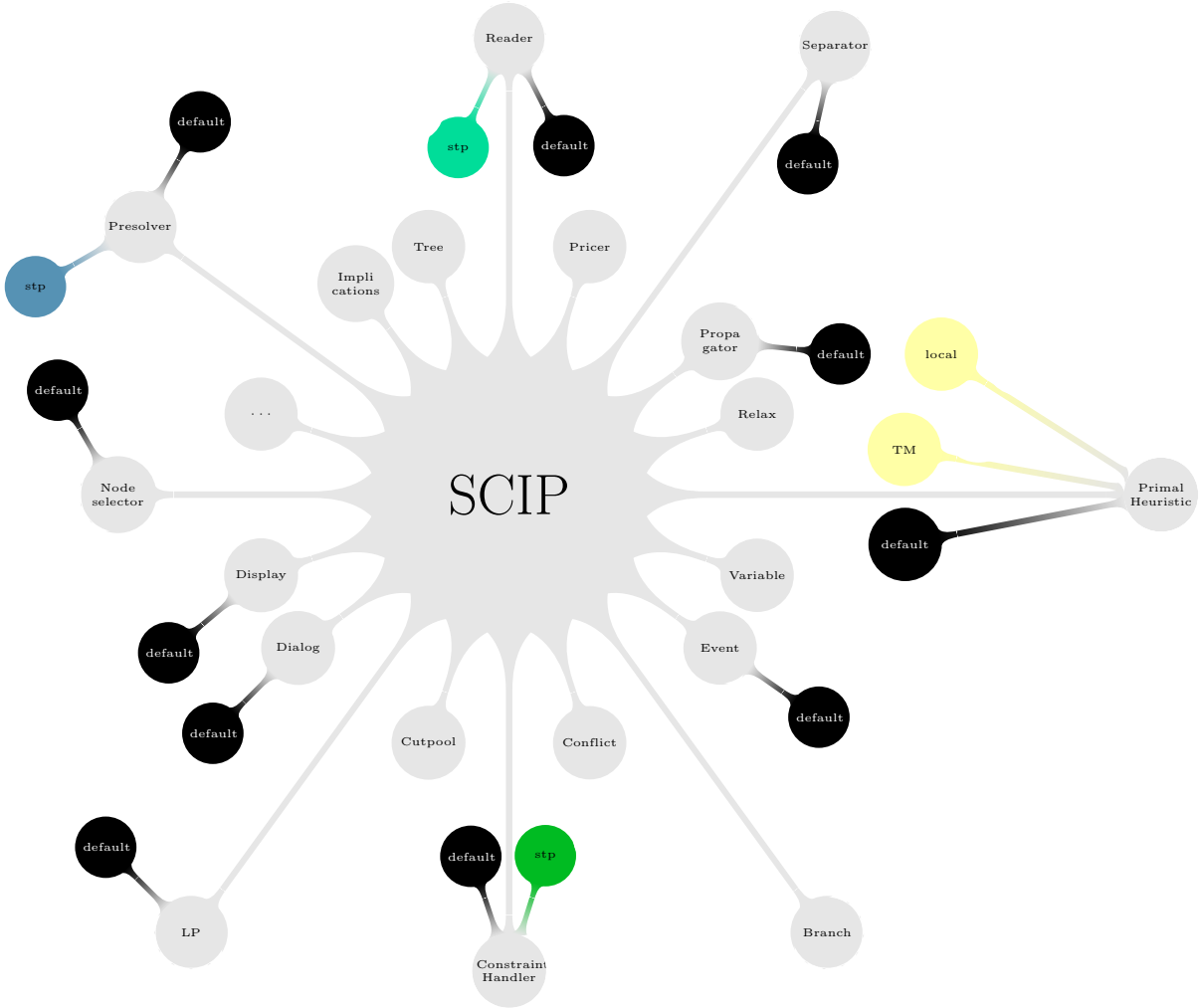
Extending SCIP: TSP



Extending SCIP: Coloring



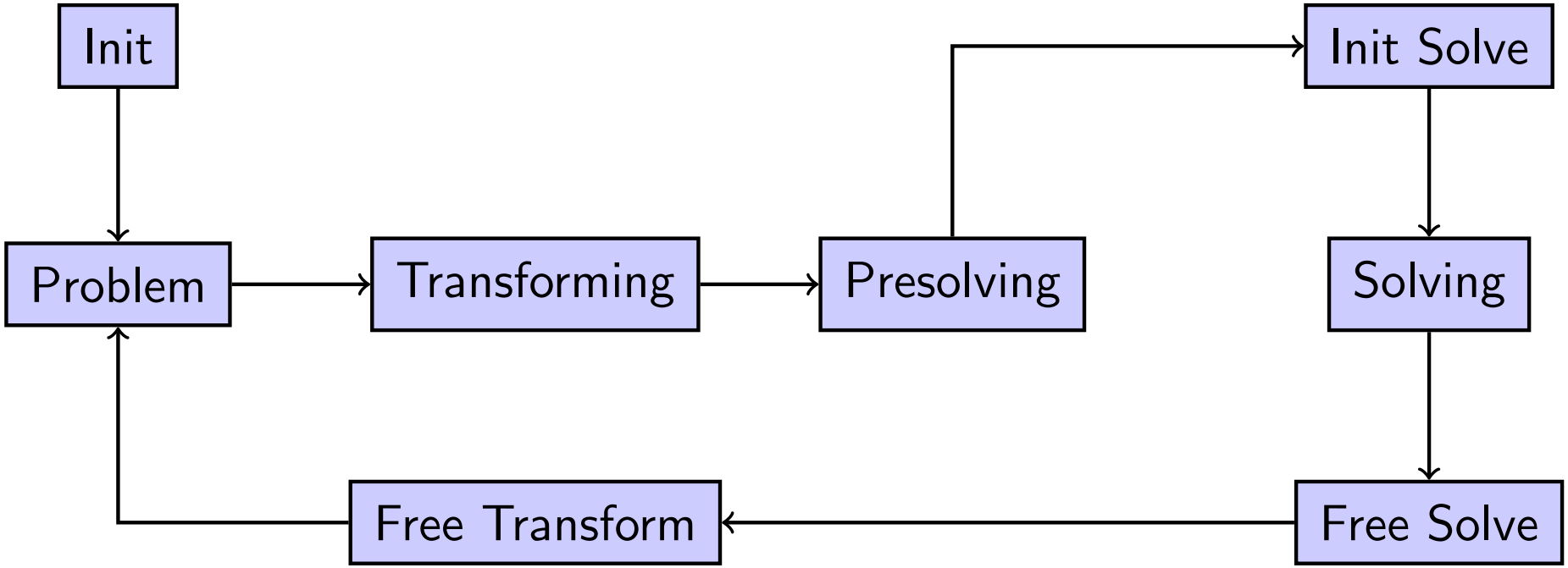
Extending SCIP: STP



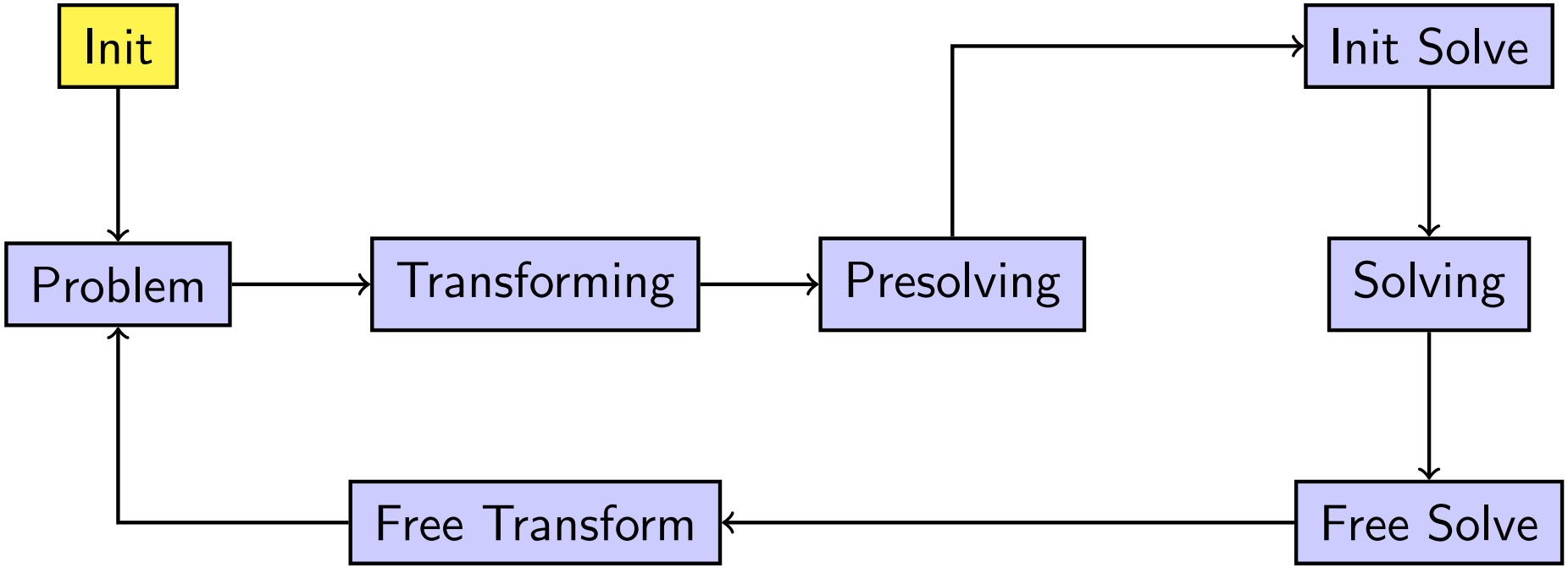
SCIP – Solving Constraint Integer Programs

http://scip.zib.de

Operational Stages

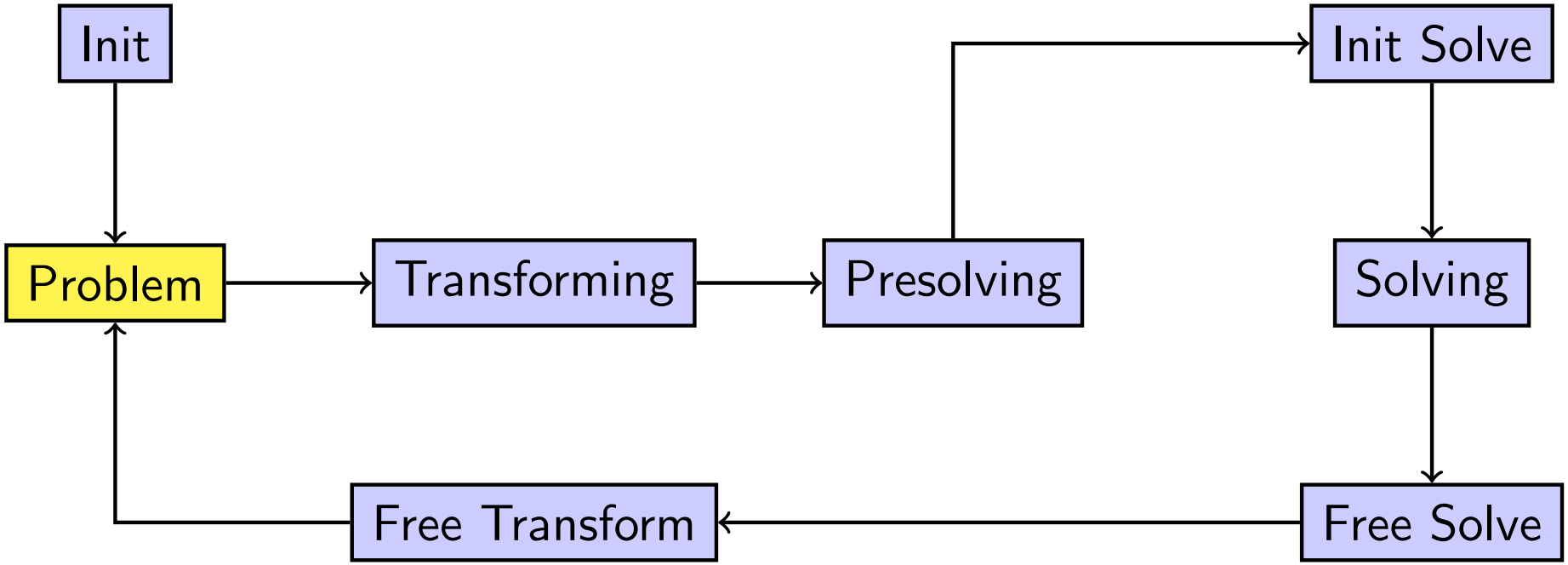


Operational Stages



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

Problem Specification



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

Define Variables (TSP Example)

```
SCIP_VAR* var;
SCIP_CALL(                          // return value macro
  SCIPcreateVar(
    scip,                           // SCIP pointer
    &var,                           // save in variable
    "varname",                      // pass variable name
    0.0,                            // lower bound
    1.0,                            // upper bound
    length,                         // obj. value
    SCIP_VARTYPE_BINARY,            // type
    TRUE,                           // initial
    FALSE,                          // removable
    NULL, NULL, NULL,               // no callback functions
    NULL                            // no variable data
  )
);
SCIP_CALL( SCIPaddVar(scip, var) );       // add var.
```

Define Variables (TSP Example)

```
SCIP_VAR* var;
SCIP_CALL(                          // return value macro
  SCIPcreateVarBasic(
    scip,                           // SCIP pointer
    &var,                           // save in variable
    "varname",                      // pass variable name
    0.0,                            // lower bound
    1.0,                            // upper bound
    length,                         // obj. value
    SCIP_VARTYPE_BINARY             // type
  )
);
SCIP_CALL( SCIPaddVar(scip, var) );        // add var.
```

TSP: Define Degree Constraints

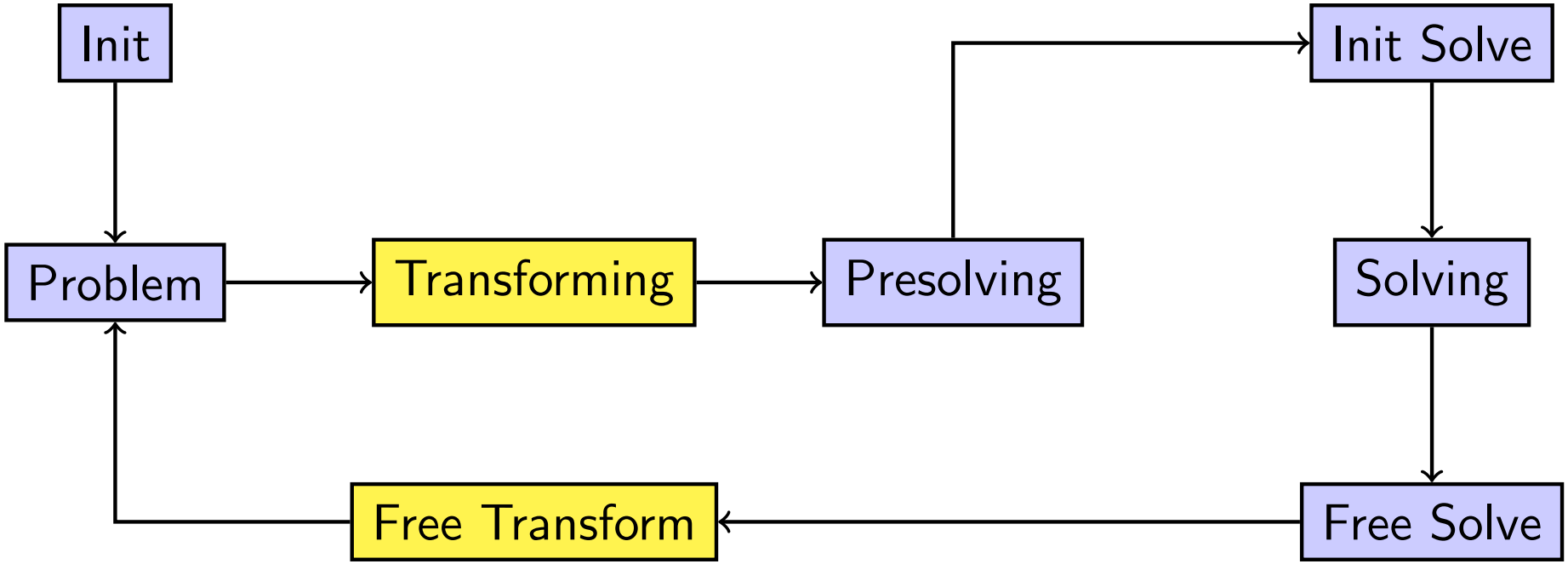
```
SCIP_CALL( SCIPcreateConsLinear(
    scip,           // SCIP pointer
    &cons,          // save in cons
    "consname",     // name
    nvar,           // number of variables
    vars,           // array of variables
    vals,           // array of values
    2.0,            // left hand side
    2.0,            // right hand side (equation)
    TRUE,           // initial?
    FALSE,          // separate?
    TRUE,           // enforce?
    TRUE,           // check?
    TRUE,           // propagate?
    FALSE,          // local?
    FALSE,          // modifable?
    FALSE,          // dynamic?
    FALSE,          // removable?
    FALSE           // stick at node?
  ) );
SCIP_CALL( SCIPaddCons(scip, cons) );       // add constraint
SCIP_CALL( SCIPreleaseCons(scip, &cons) ); // free cons. space
```

MIPs are specified using linear constraints only (may be “upgraded”).

TSP: Define Degree Constraints

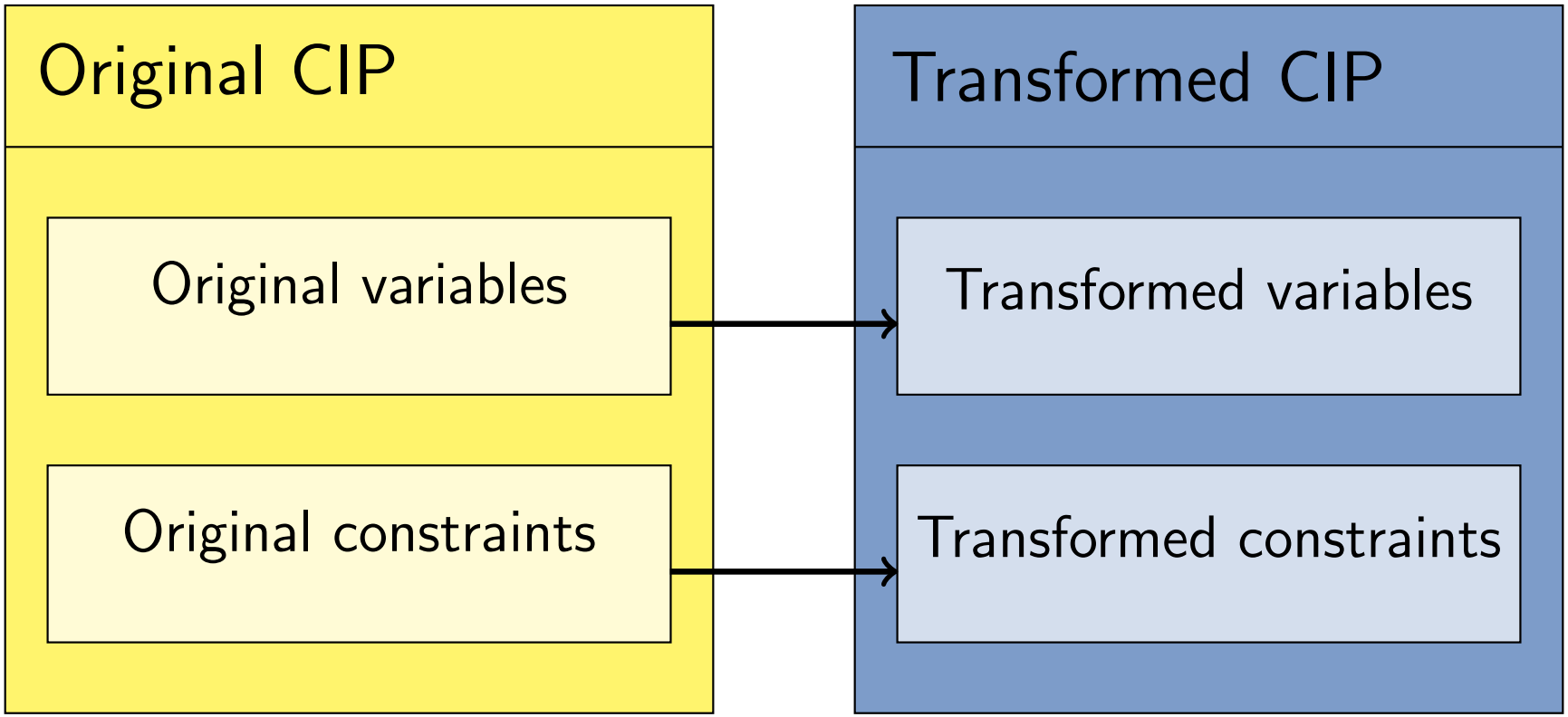
```
SCIP_CALL( SCIPcreateConsBasicLinear(
    scip,           // SCIP pointer
    &cons,          // save in cons
    "consname",     // name
    nvar,           // number of variables
    vars,           // array of variables
    vals,           // array of values
    2.0,            // left hand side
    2.0             // right hand side (equation)
  ) );
SCIP_CALL( SCIPaddCons(scip, cons) );       // add constraint
SCIP_CALL( SCIPreleaseCons(scip, &cons) ); // free cons. space
```

MIPs are specified using linear constraints only (may be “upgraded”).



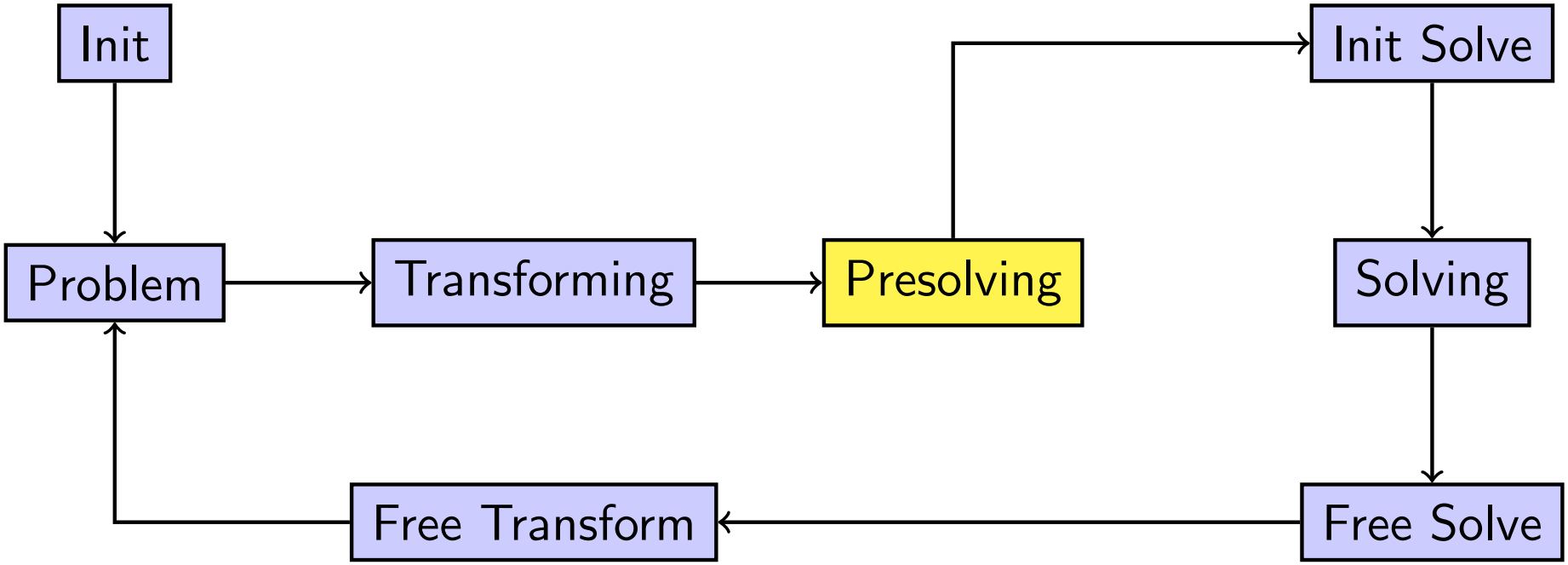
- Creates a working copy of the original problem.

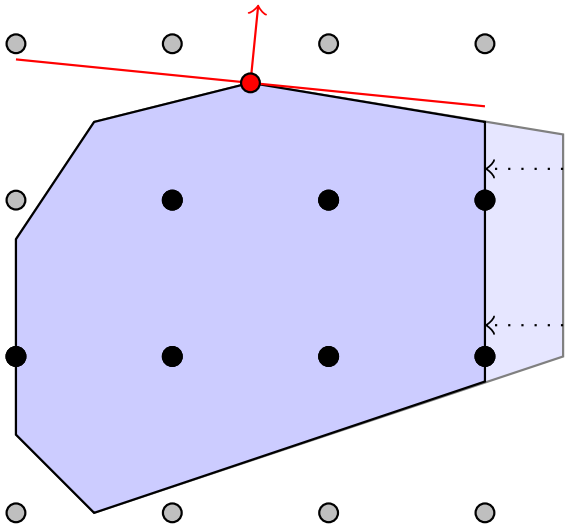
Original and Transformed Problem



- data is 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 Transform
Free Solve

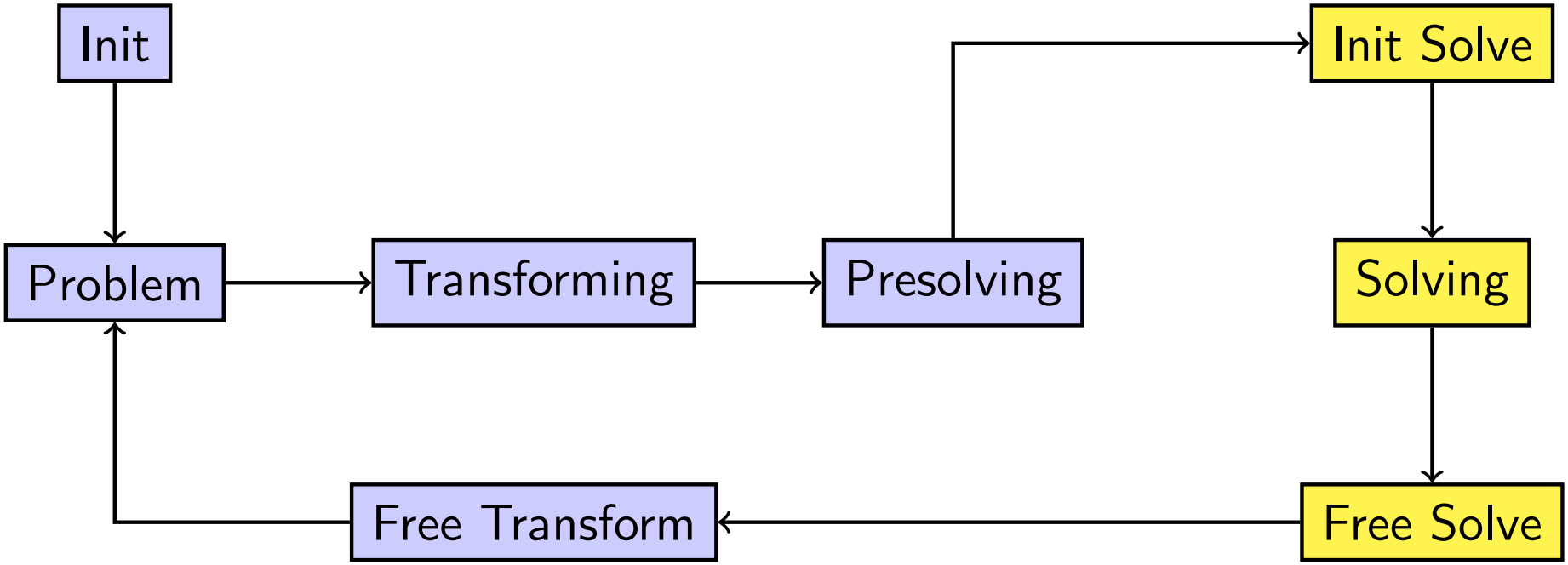


Task

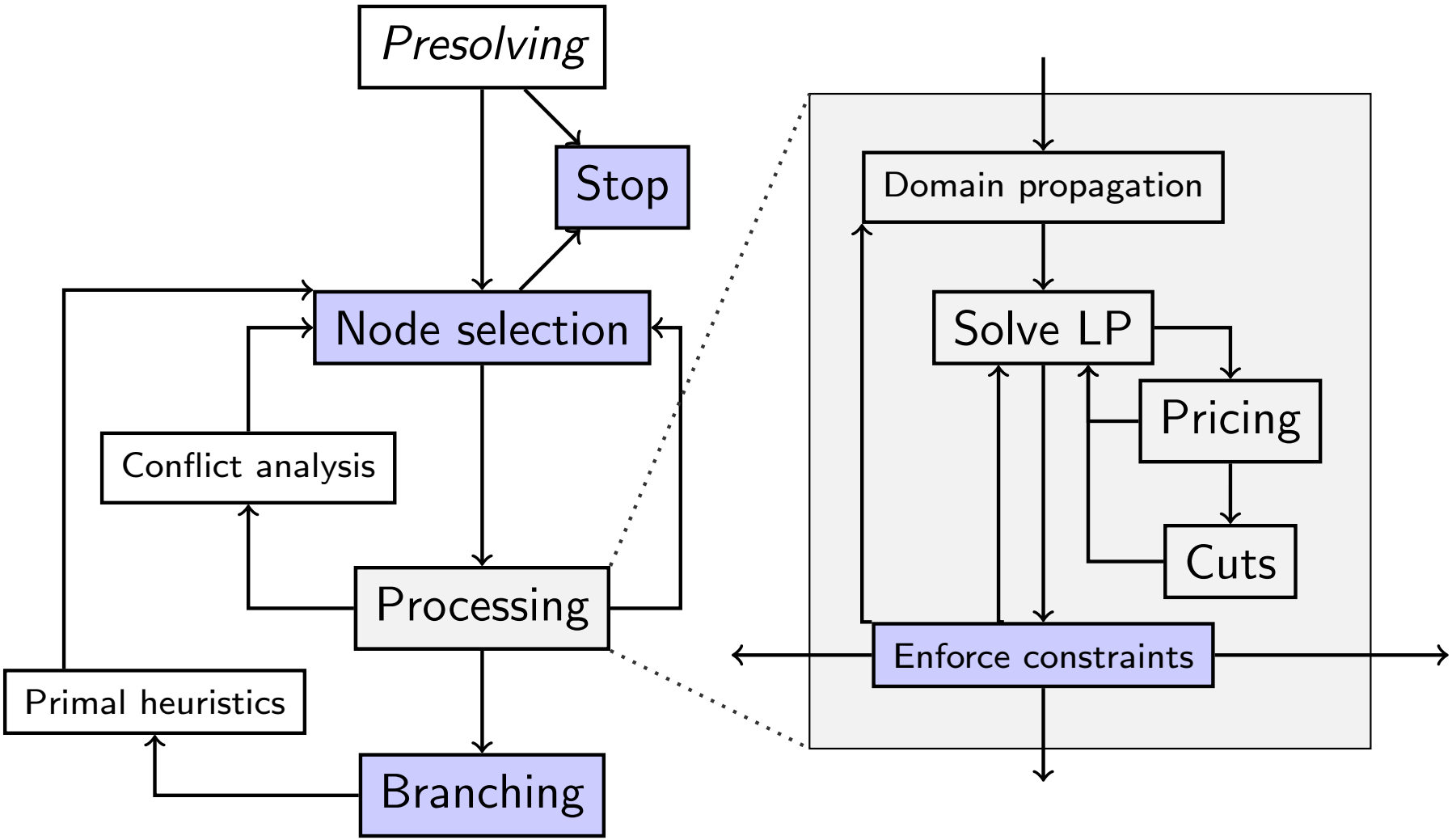
- ▷ reduce size of model by removing irrelevant information
- ▷ strengthen LP relaxation by exploiting integrality information
- ▷ make the LP relaxation numerically more stable
- ▷ extract useful information

Primal Reductions:
- ▷ based on feasibility reasoning
- ▷ no feasible solution is cut off

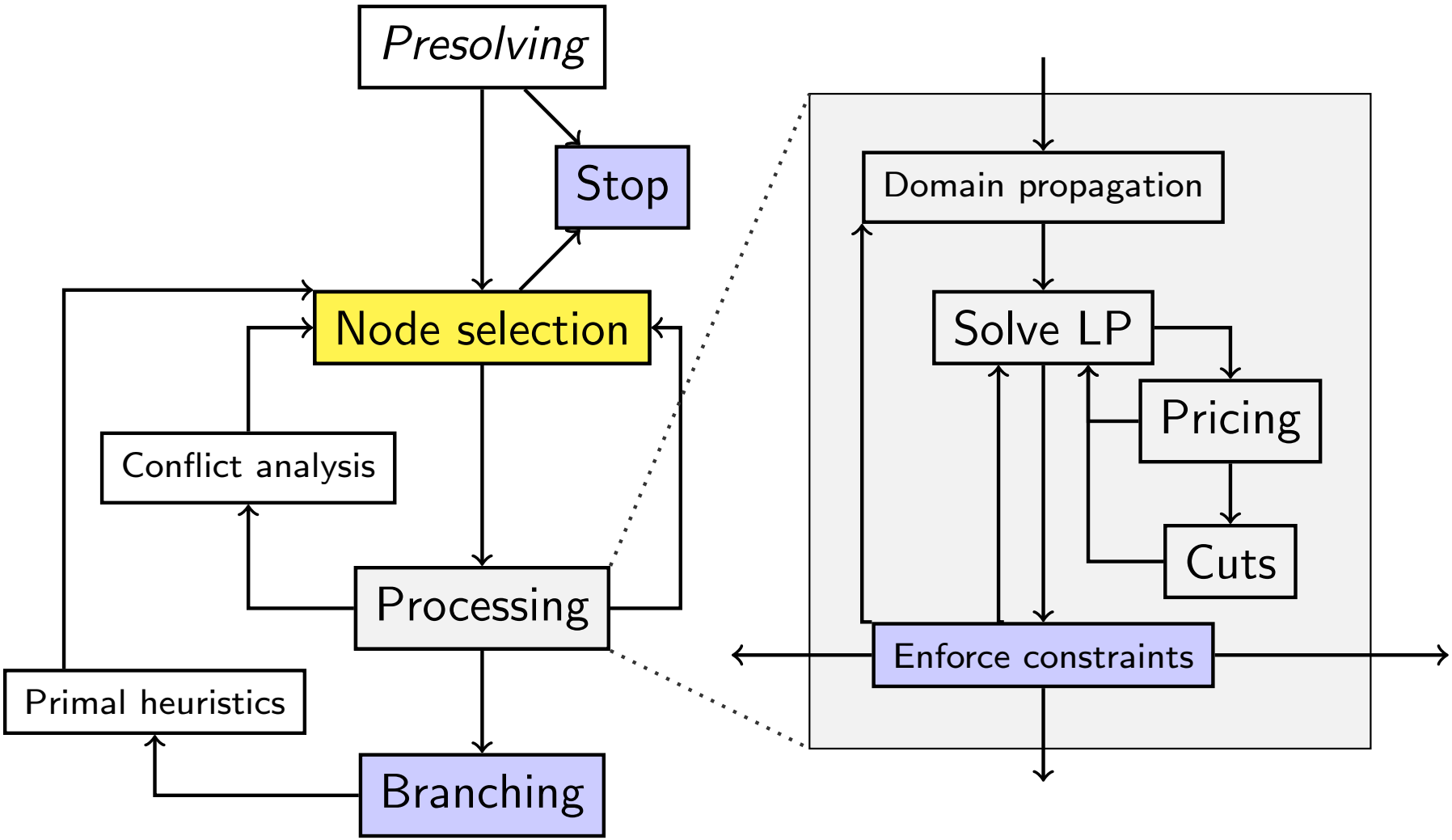
Dual Reductions:
- ▷ consider objective function
- ▷ at least one optimal solution remains

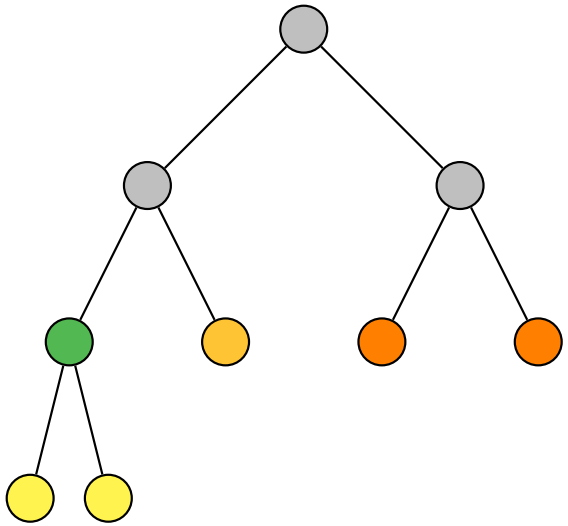

Init
Problem
Transforming
Presolving
Init Solve
Solving
Free Solve
Free Transform

Flow Chart SCIP



Flow Chart SCIP





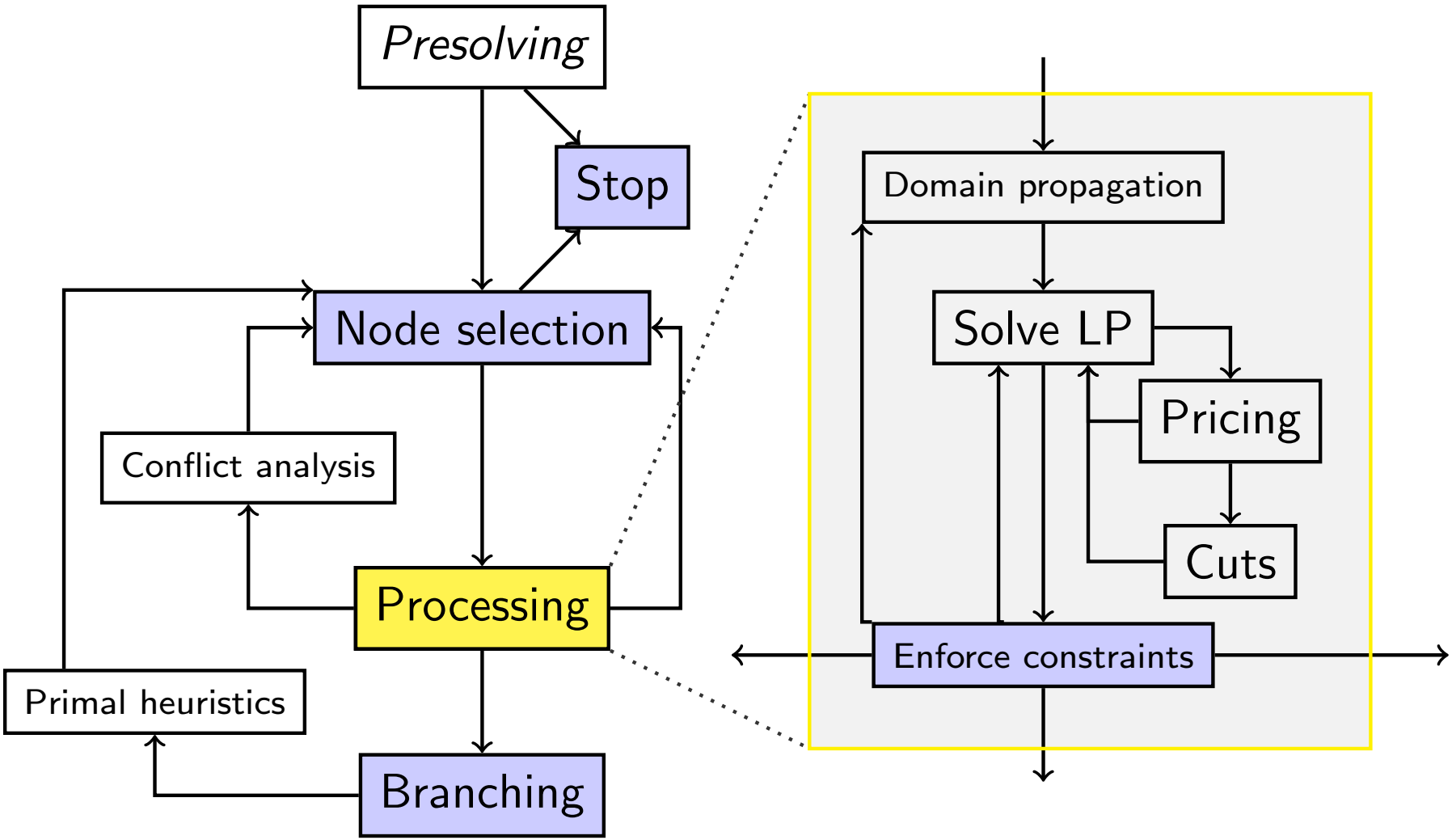
Task

- improve primal bound
- keep comp. effort small
- improve global dual bound

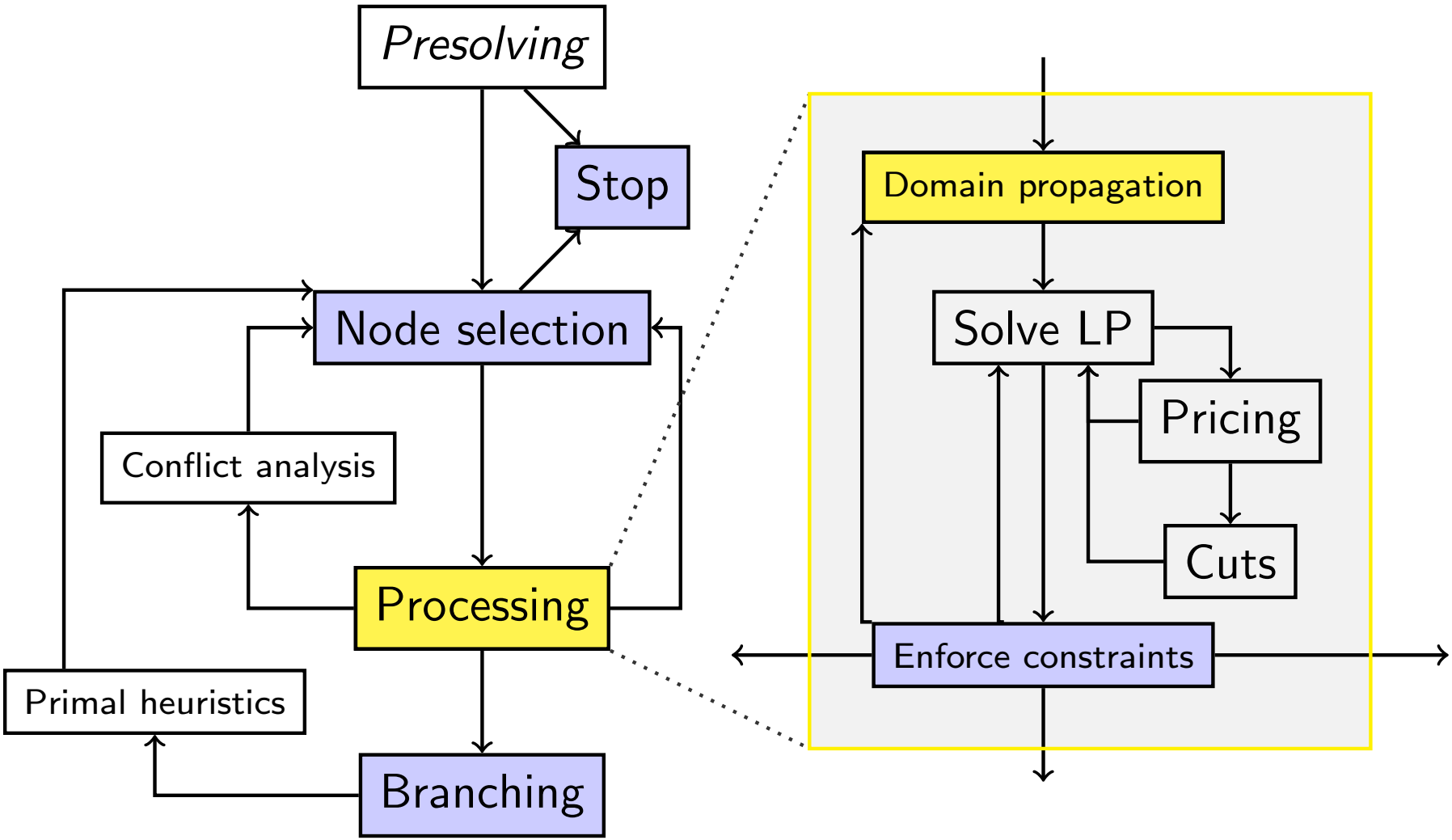
Techniques

- basic rules
  - depth first search (DFS)
    → early feasible solutions
  - best bound search (BBS)
    → improve dual bound
  - best estimate search (BES)
    → improve primal bound
- combinations
  - BBS or BES with plunging
  - hybrid BES/BBS
  - interleaved BES/BBS

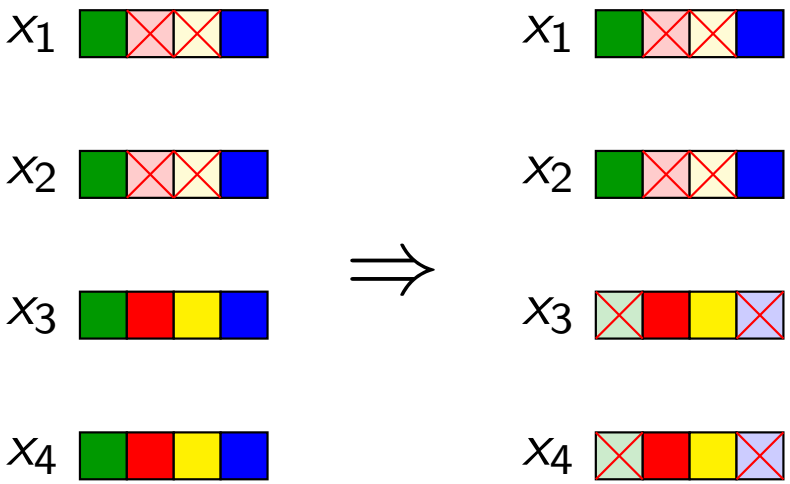
Flow Chart SCIP



Flow Chart SCIP



Domain Propagation



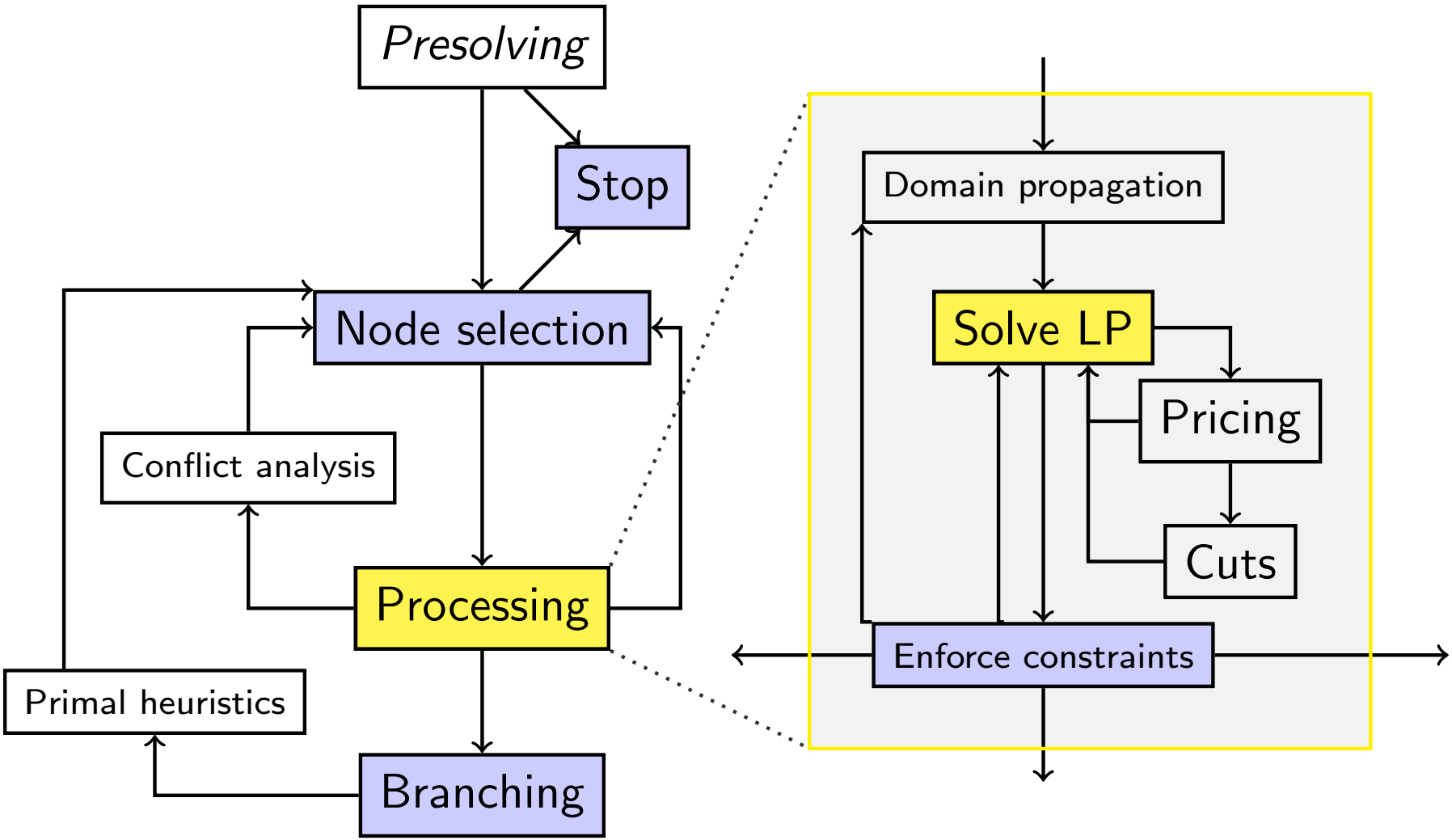
Task

- simplify model locally
- improve local dual bound
- detect infeasibility

Techniques

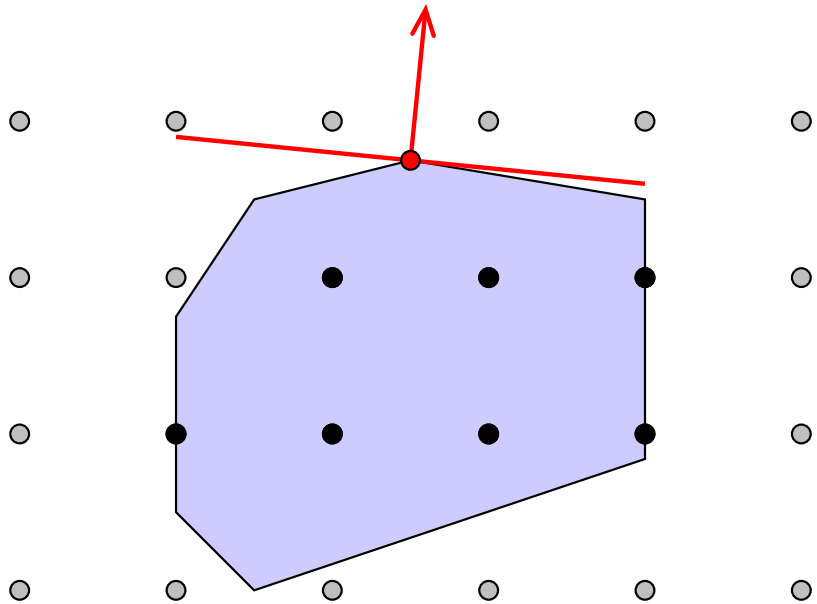
- constraint specific
  - each cons handler may provide a propagation routine
  - reduced presolving (usually)
- dual propagation
  - root reduced cost strengthening
  - objective function
- special structures
  - variable bounds

Flow Chart SCIP



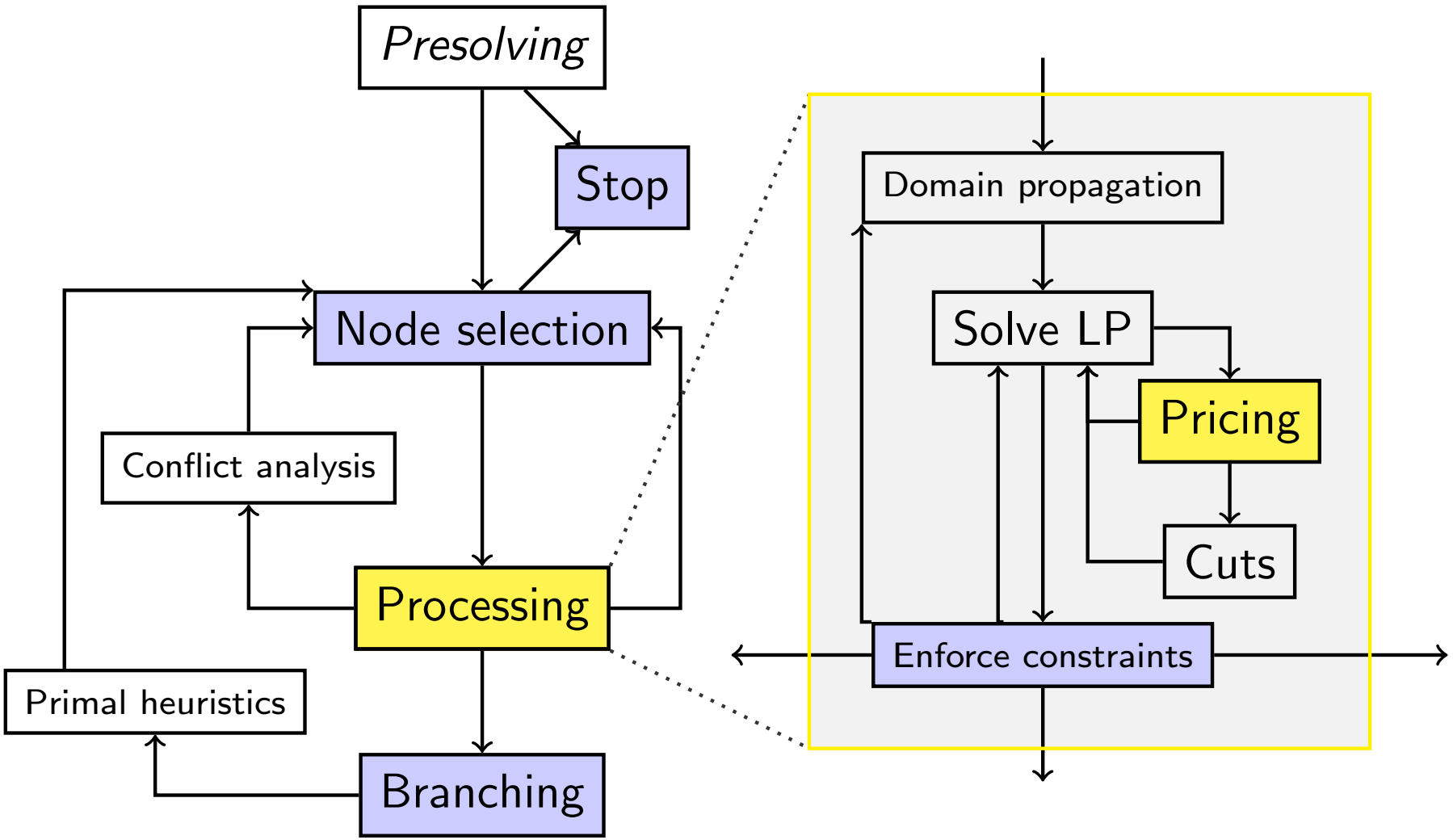
LP Solving

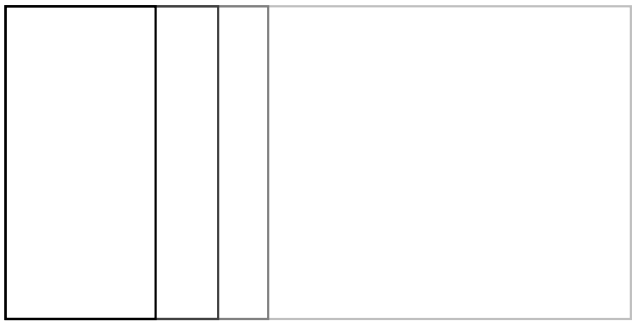
- LP solver is a black box
- interface to different LP solvers:
  SoPlex, CPLEX, XPress,
  Gurobi, CLP, ...
- primal/dual simplex
- barrier with/without crossover



- feasibility double-checked by SCIP
- condition number check
- resolution by changing parameters:
  scaling, tolerances, solving from scratch, other simplex

Flow Chart SCIP





Branch-and-Price

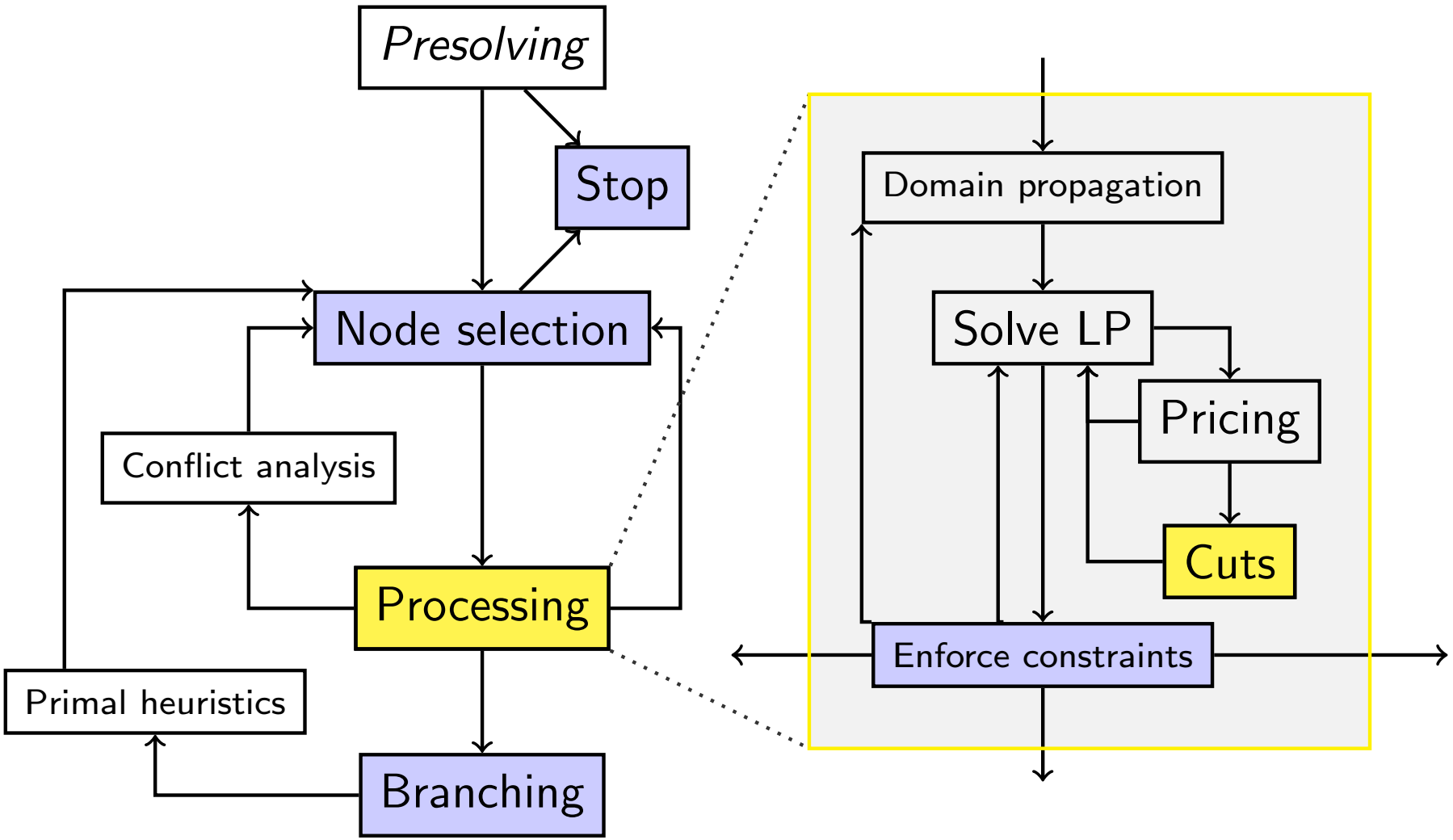
- huge number of variables
- start with subset
- add others, when needed

Pricing

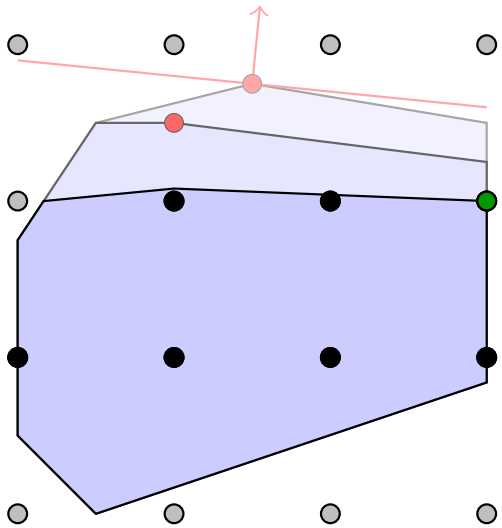
- find variable with negative reduced costs
- or prove that there exists none
- typically problem specific

- dynamic aging of variables
- problem variable pricer to add them again
- early branching possible
- lazy variable bounds

Flow Chart SCIP



Cutting Plane Separation



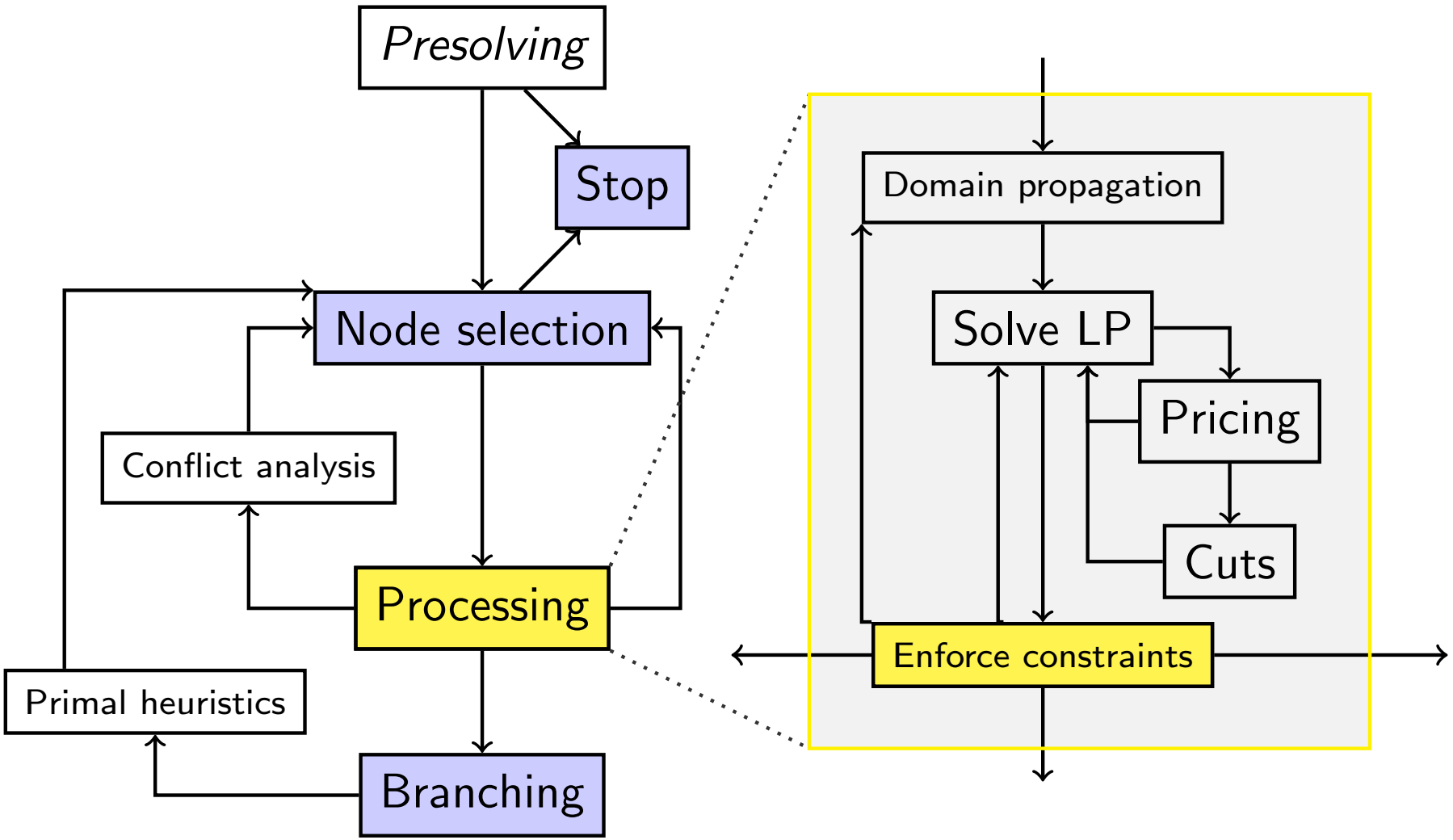
Task

- strengthen relaxation
- add valid constraints
- generate on demand

Techniques

- general cuts
  - complemented MIR cuts
  - Gomory mixed integer cuts
  - strong Chvátal-Gomory cuts
  - implied bound cuts
  - reduced cost strengthening
- problem specific cuts
  - 0-1 knapsack problem
  - stable set problem
  - 0-1 single node flow problem

Flow Chart SCIP



Constraint Enforcement

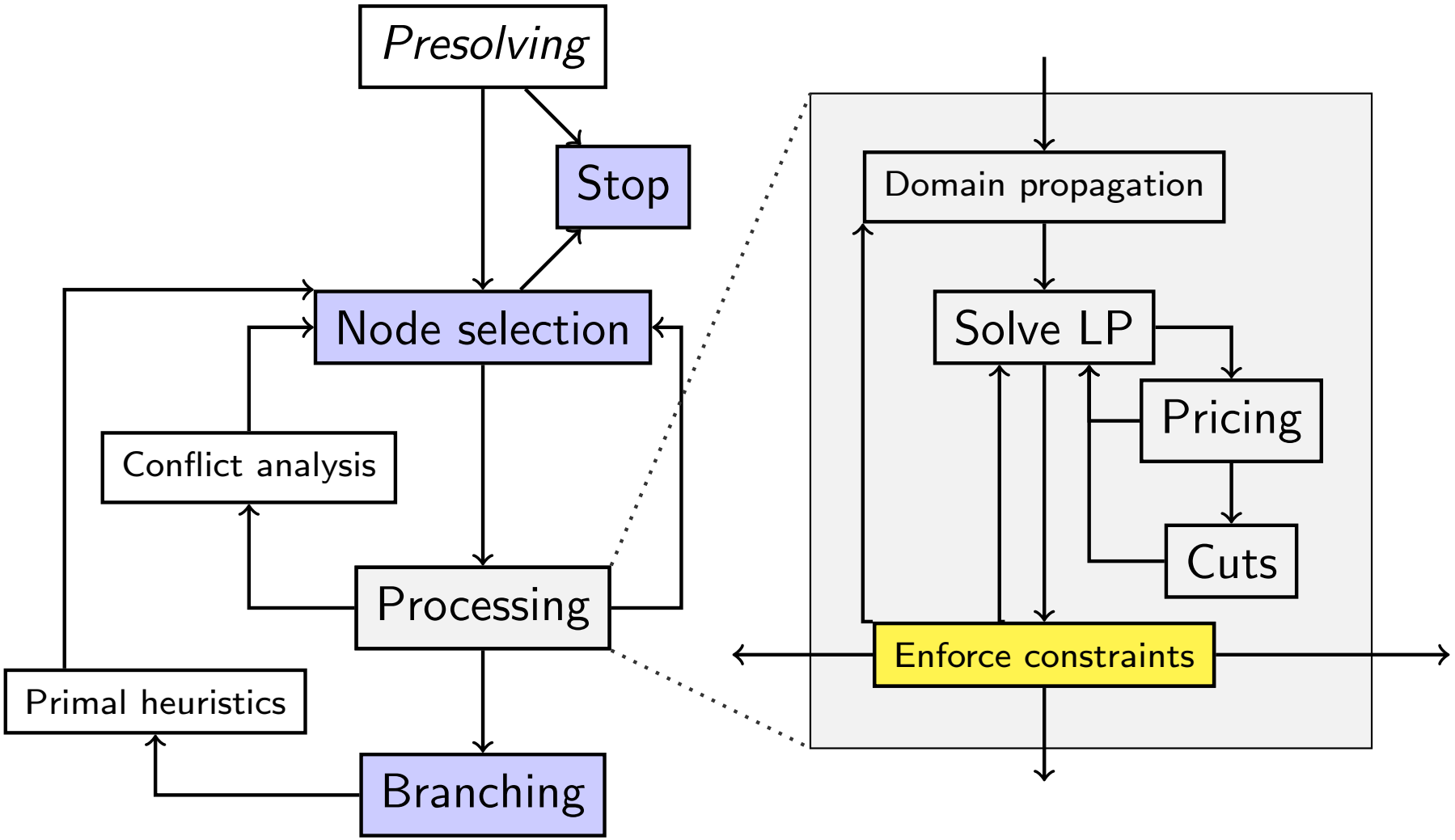
LP solution may violate a constraint not contained in the relaxation.

Enforcing is necessary for a correct implementation!

Constraint handler resolves the infeasibility by ...

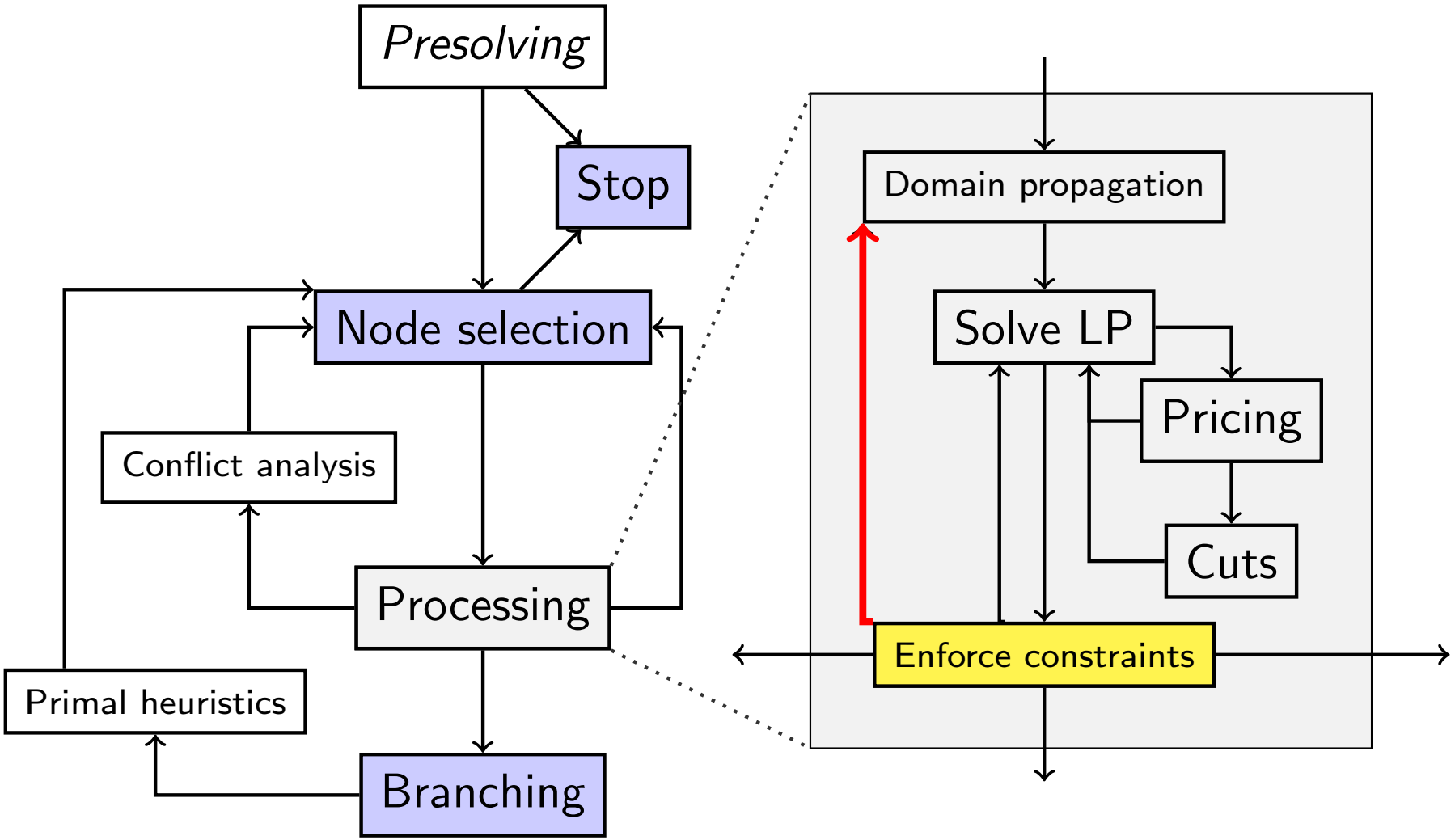
- Reducing a variable's domain,
- Separating a cutting plane (may use integrality),
- Adding a (local) constraint,
- Creating a branching,
- Concluding that the subproblem is infeasible and can be cut off, or
- Just saying "solution infeasible".

Constraint Enforcement



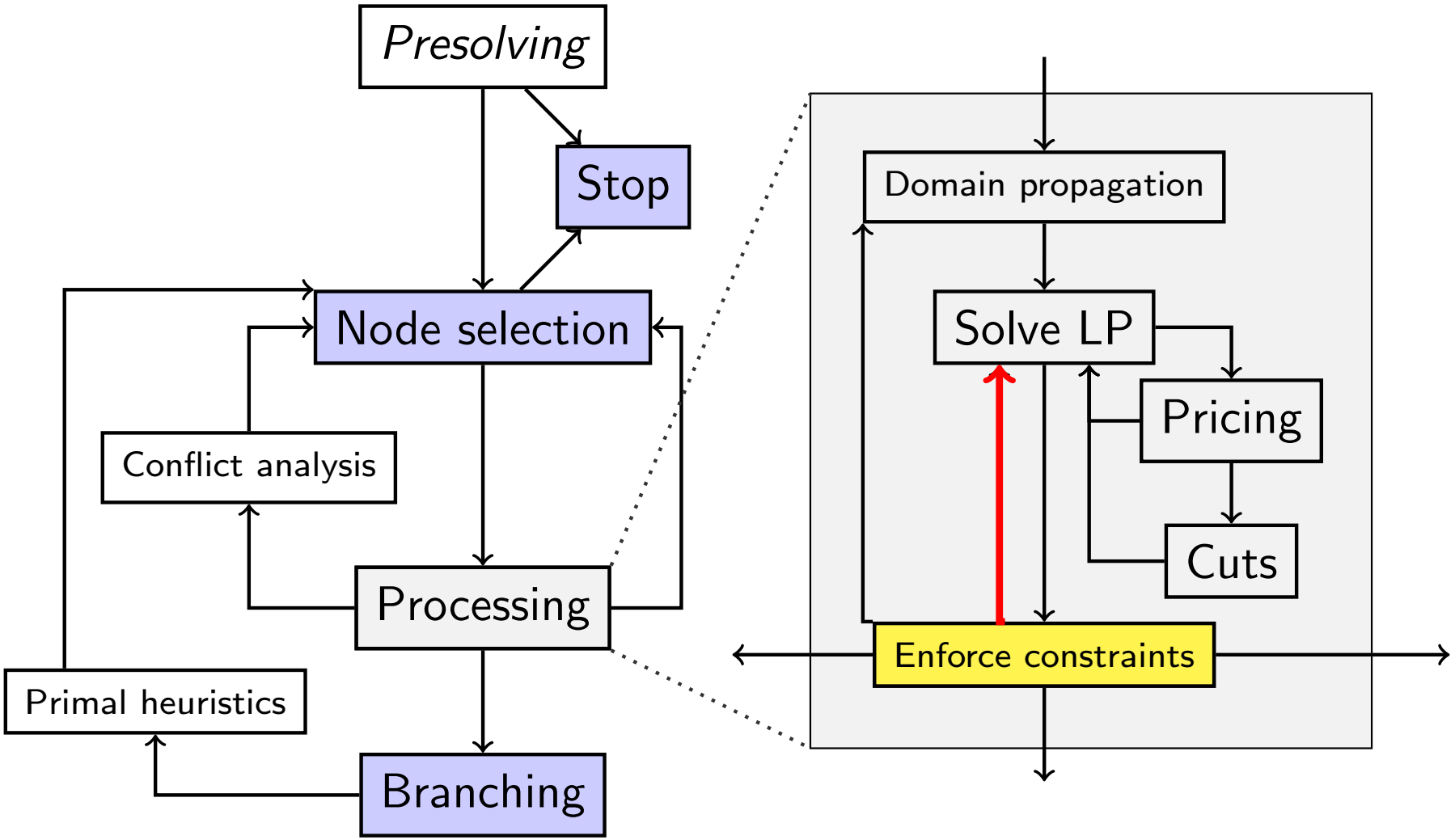
- Reduced domain
- Added constraint
- Added cut
- Branched
- Cutoff
- Infeasible
- Feasible

Constraint Enforcement



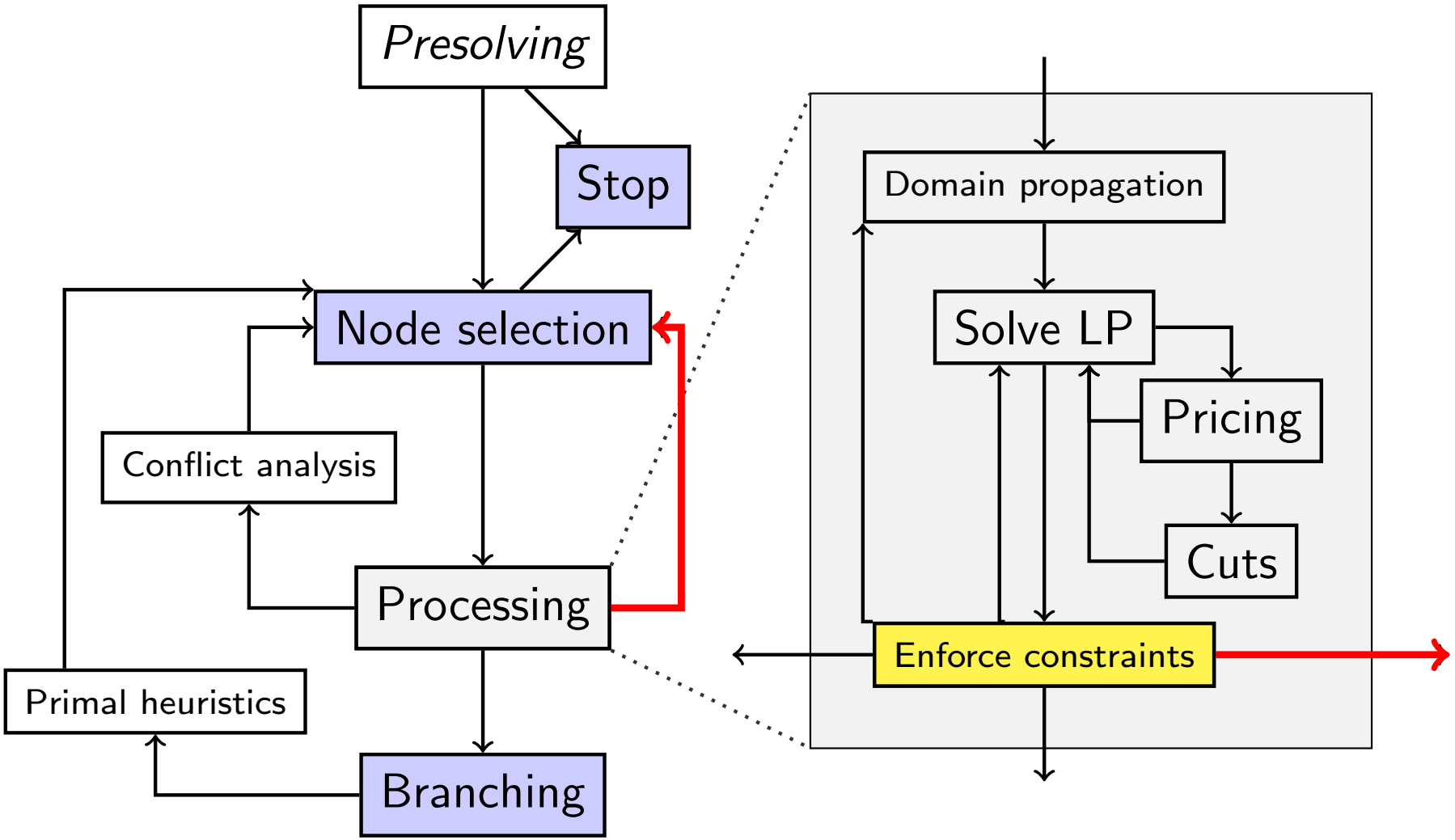
- Reduced domain
- Added constraint
- Added cut
- Branched
- Cutoff
- Infeasible
- Feasible

Constraint Enforcement



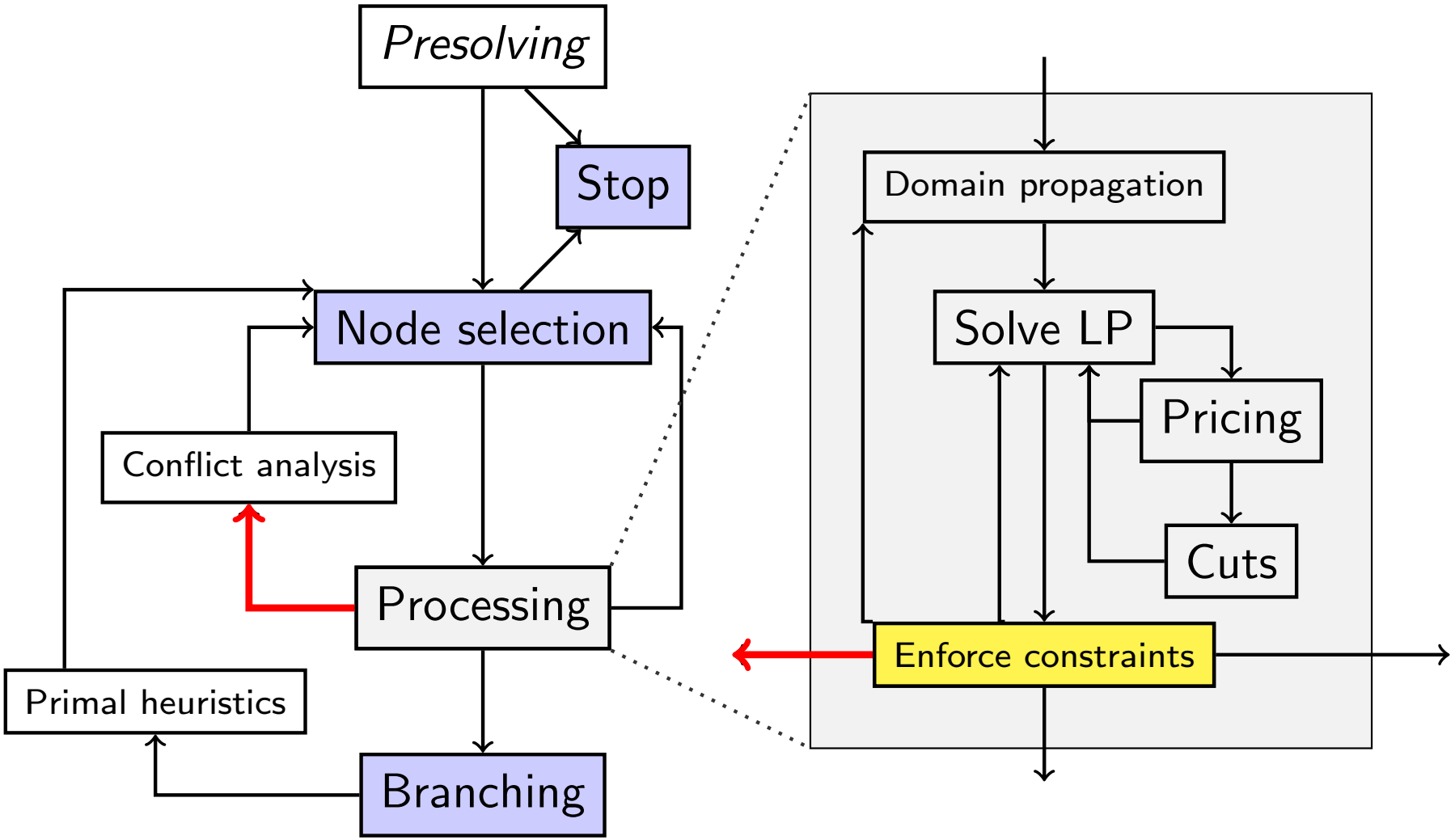
- Reduced domain
- Added constraint
- Added cut
- Branched
- Cutoff
- Infeasible
- Feasible

Constraint Enforcement



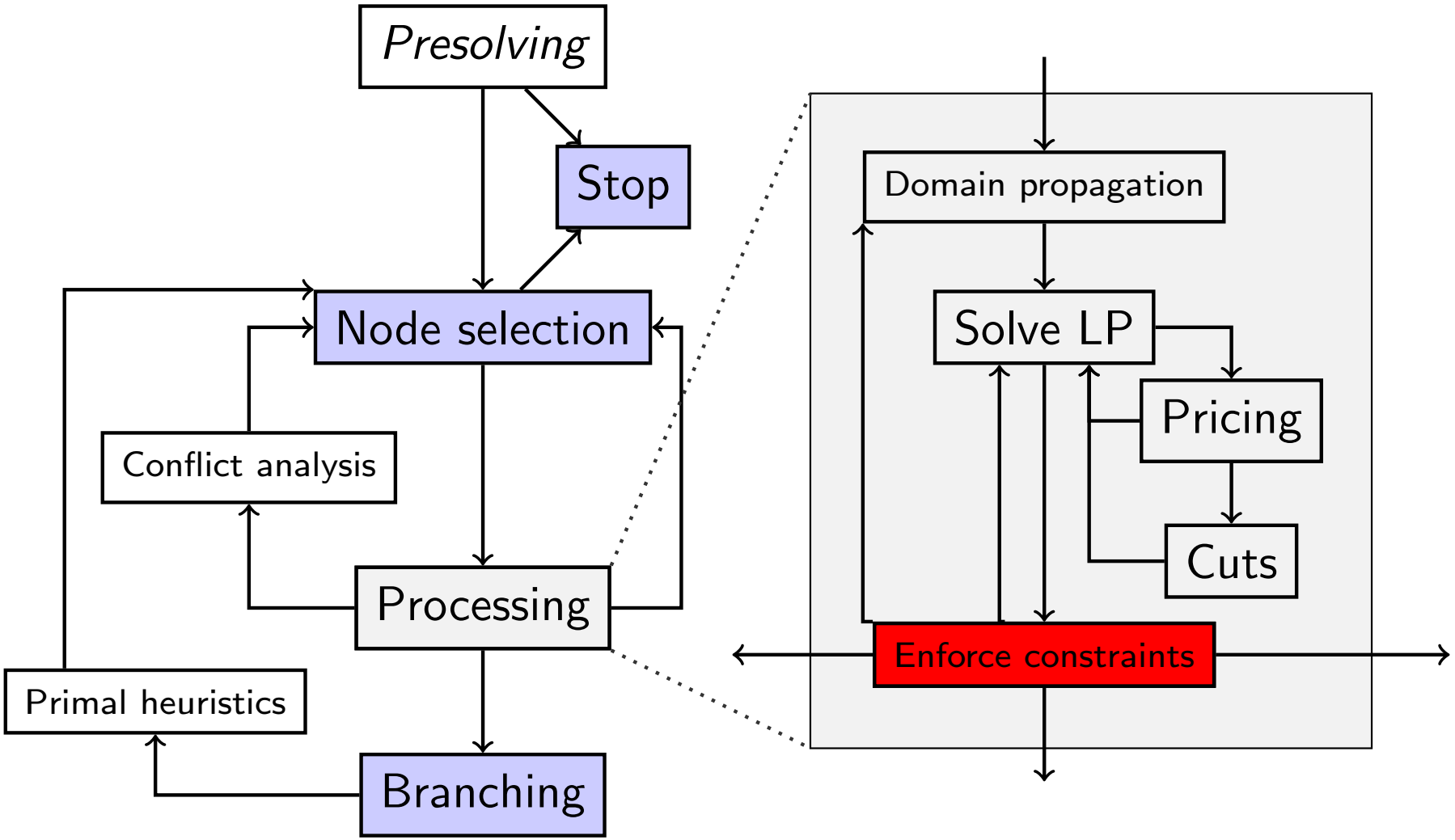
- Reduced domain
- Added constraint
- Added cut
- Branched
- Cutoff
- Infeasible
- Feasible

Constraint Enforcement



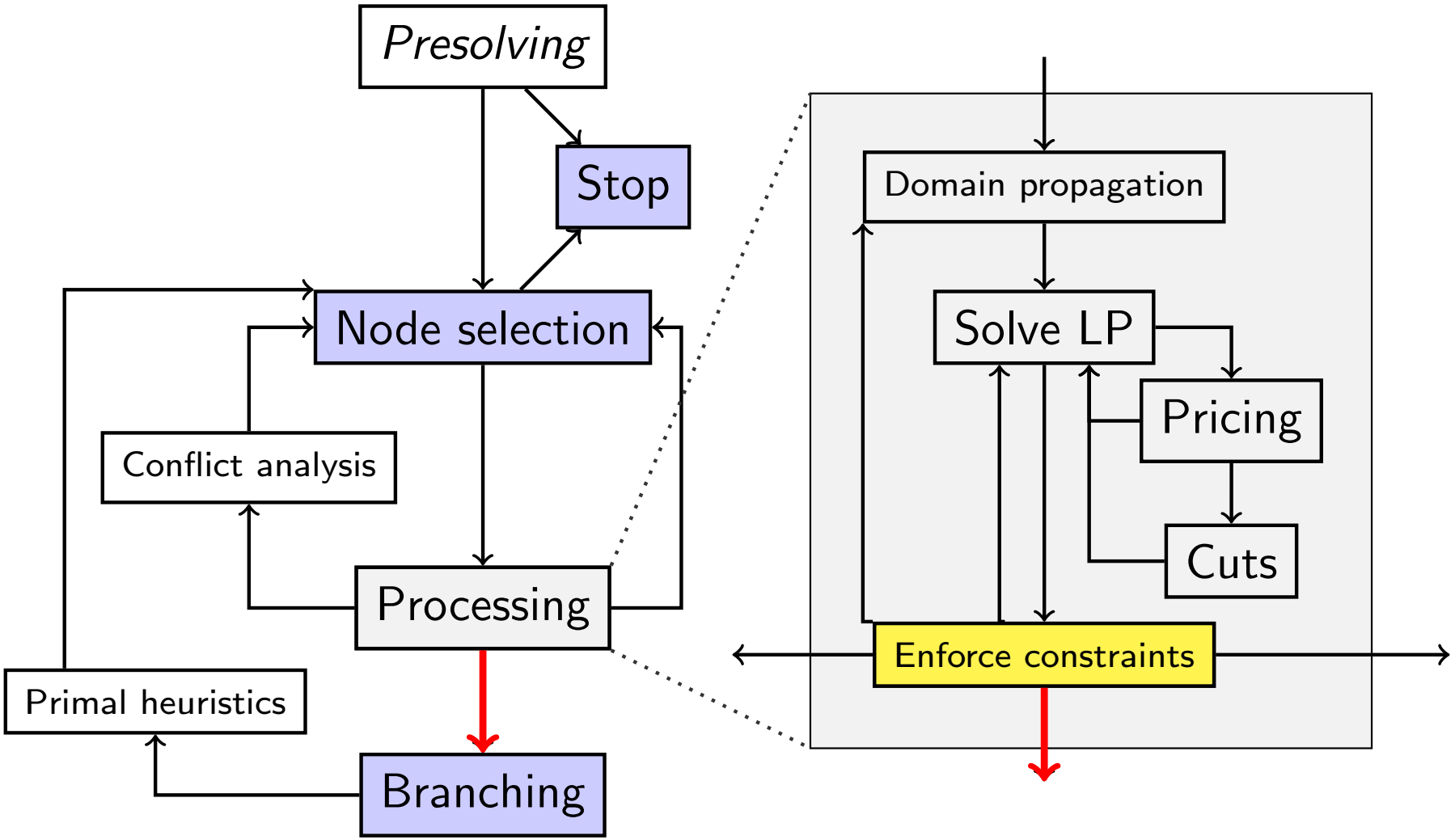
- ▷ Reduced domain
- ▷ Added constraint
- ▷ Added cut
- ▷ Branched
- ▷ Cutoff
- ▷ Infeasible
- ▷ Feasible

Constraint Enforcement



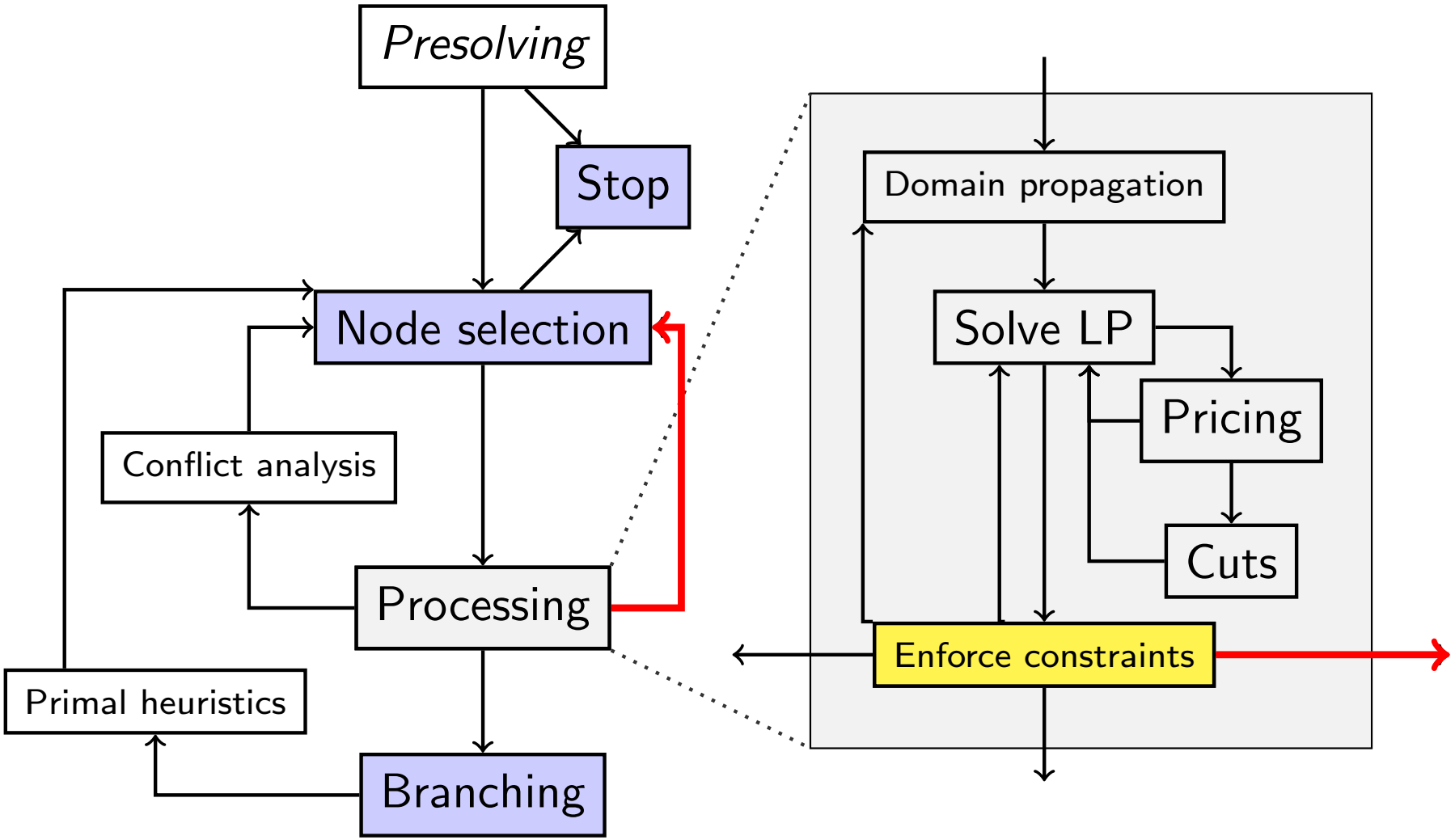
- Reduced domain
- Added constraint
- Added cut
- Branched
- Cutoff
- Infeasible
- Feasible

Constraint Enforcement



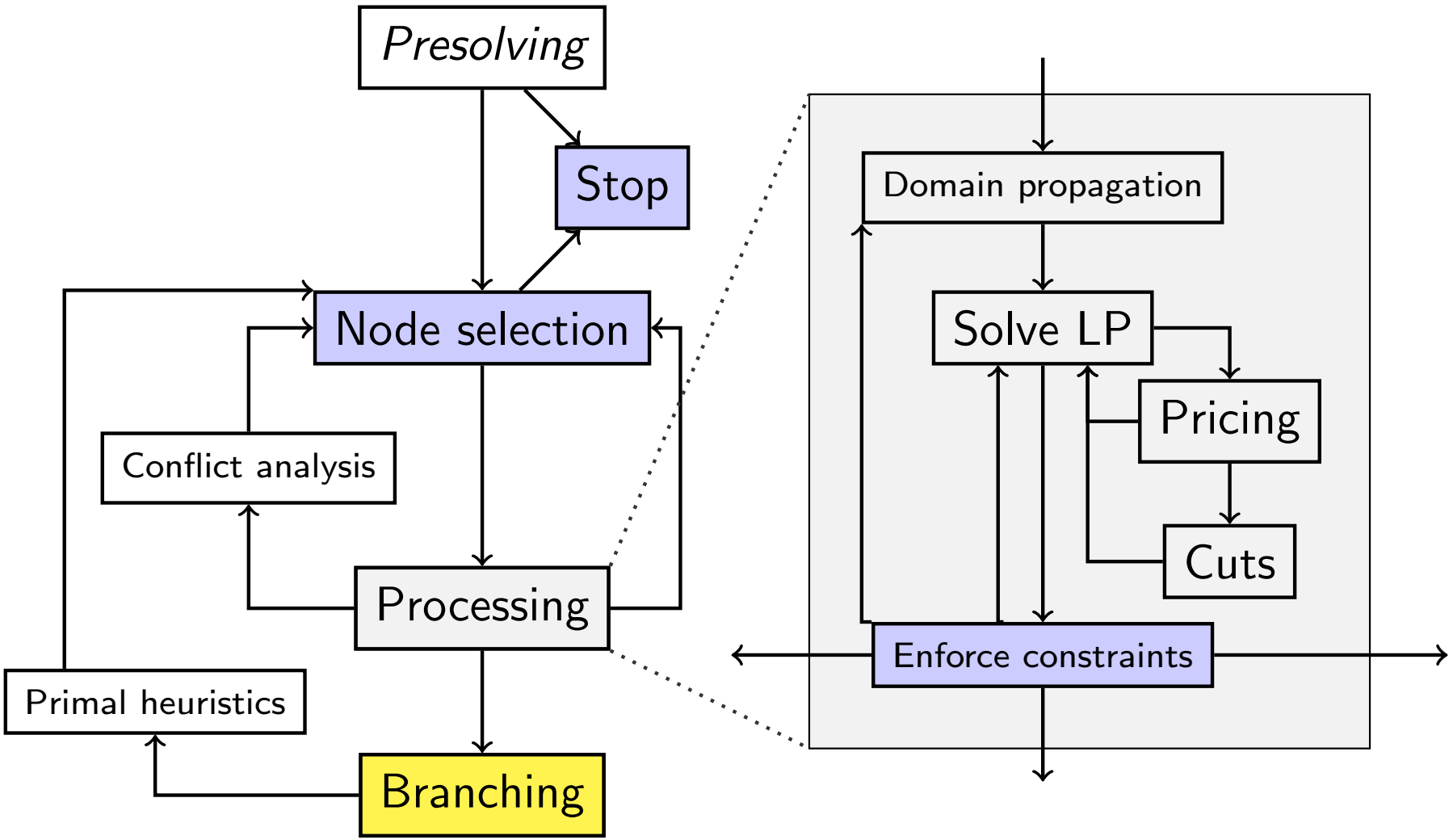
- ▷ Reduced domain
- ▷ Added constraint
- ▷ Added cut
- ▷ Branched
- ▷ Cutoff
- ▷ Infeasible
- ▷ Feasible

Constraint Enforcement

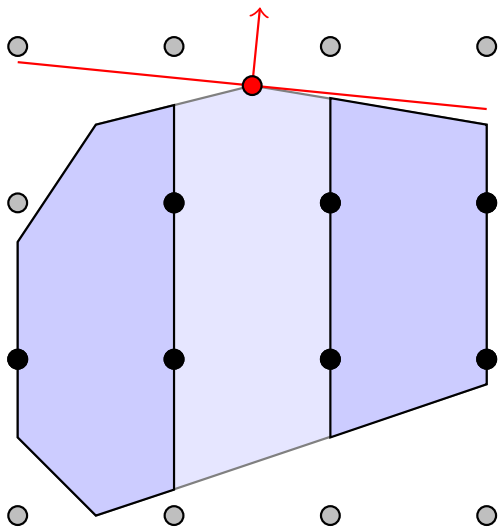


- Reduced domain
- Added cut
- Cutoff
- Infeasible
- Added constraint
- Branched
- Feasible

Flow Chart SCIP



Branching Rules



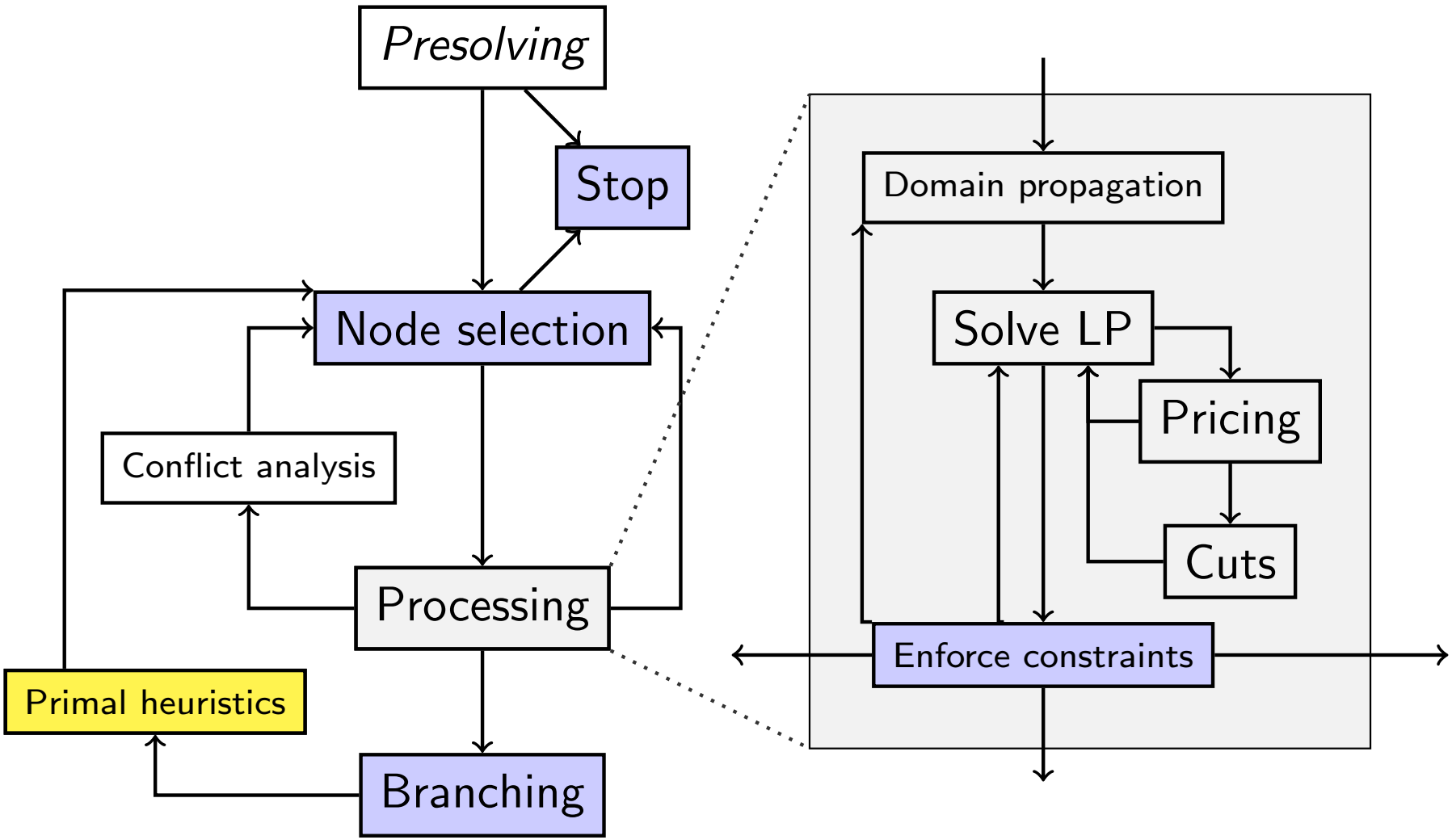
Task

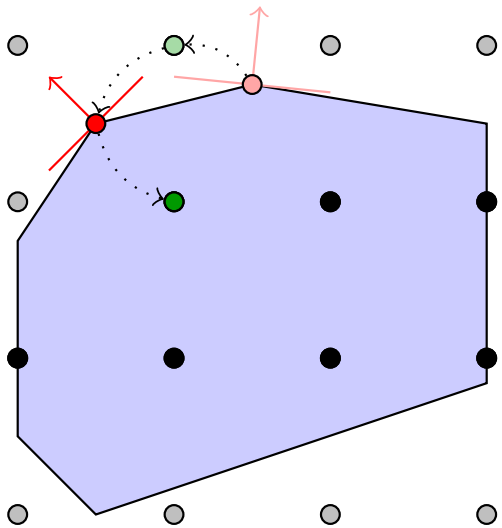
- divide into (disjoint) subproblems
- improve local bounds

Techniques

- branching on variables
  - most infeasible
  - least infeasible
  - random branching
  - strong branching
  - pseudocost
  - reliability
  - VSIDS
  - hybrid reliability/inference
- branching on constraints
  - SOS1
  - SOS2

Flow Chart SCIP





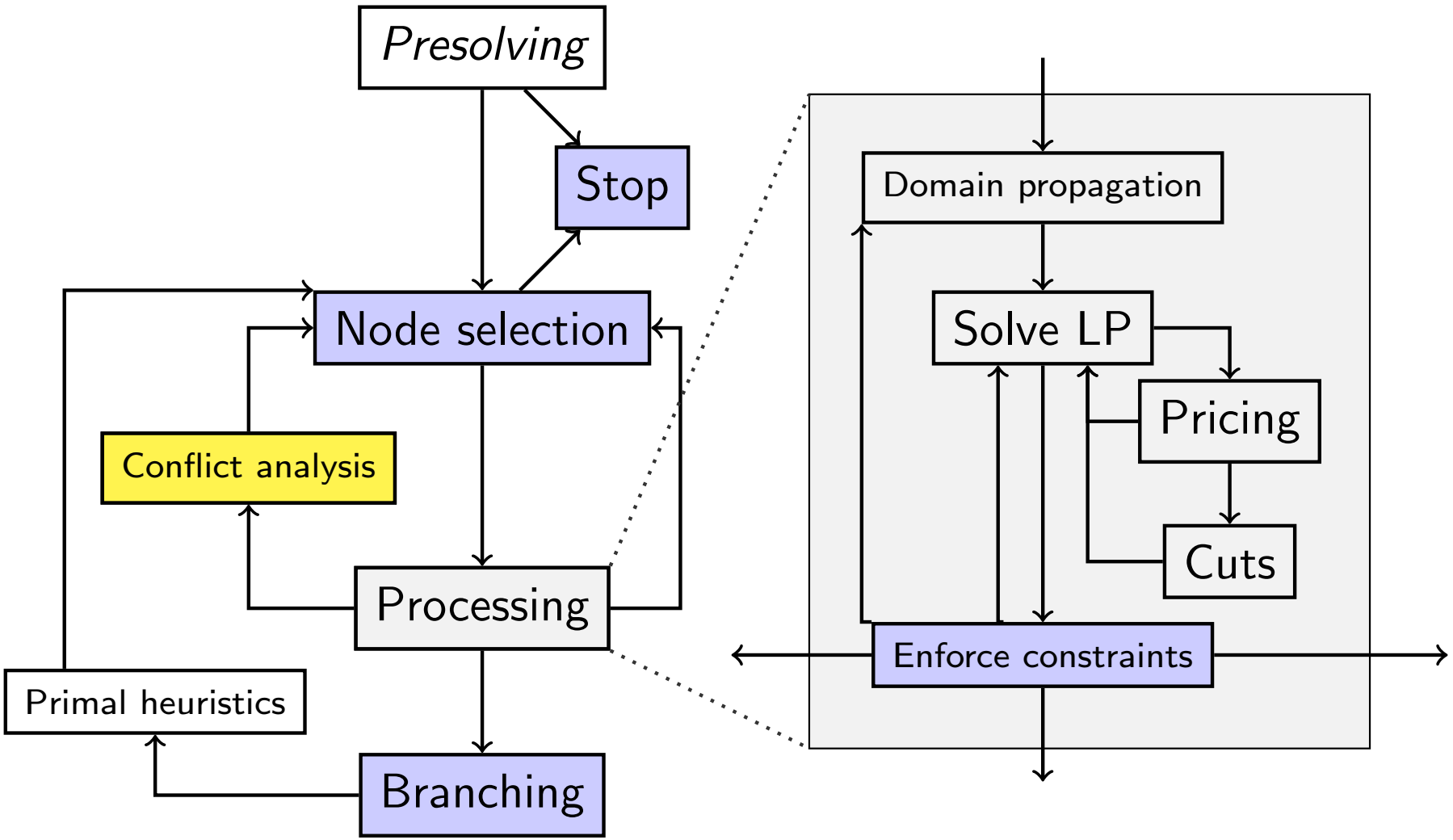
Task

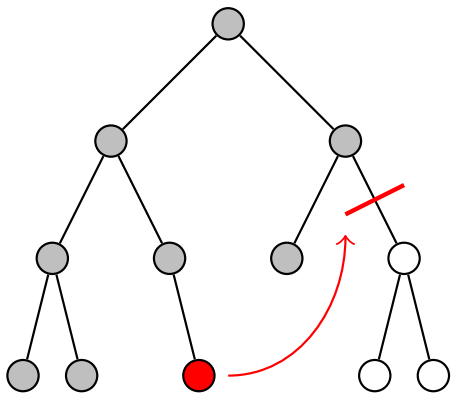
- improve primal bound
- effective on average
- guide remaining search

Techniques

- rounding
  - possibly solve final LP
- diving
  - least infeasible
  - guided
- objective diving
  - objective feasibility pump
- Large Neighborhood Search
  - RINS, local branching
  - RENS
- combinatorial

Flow Chart SCIP





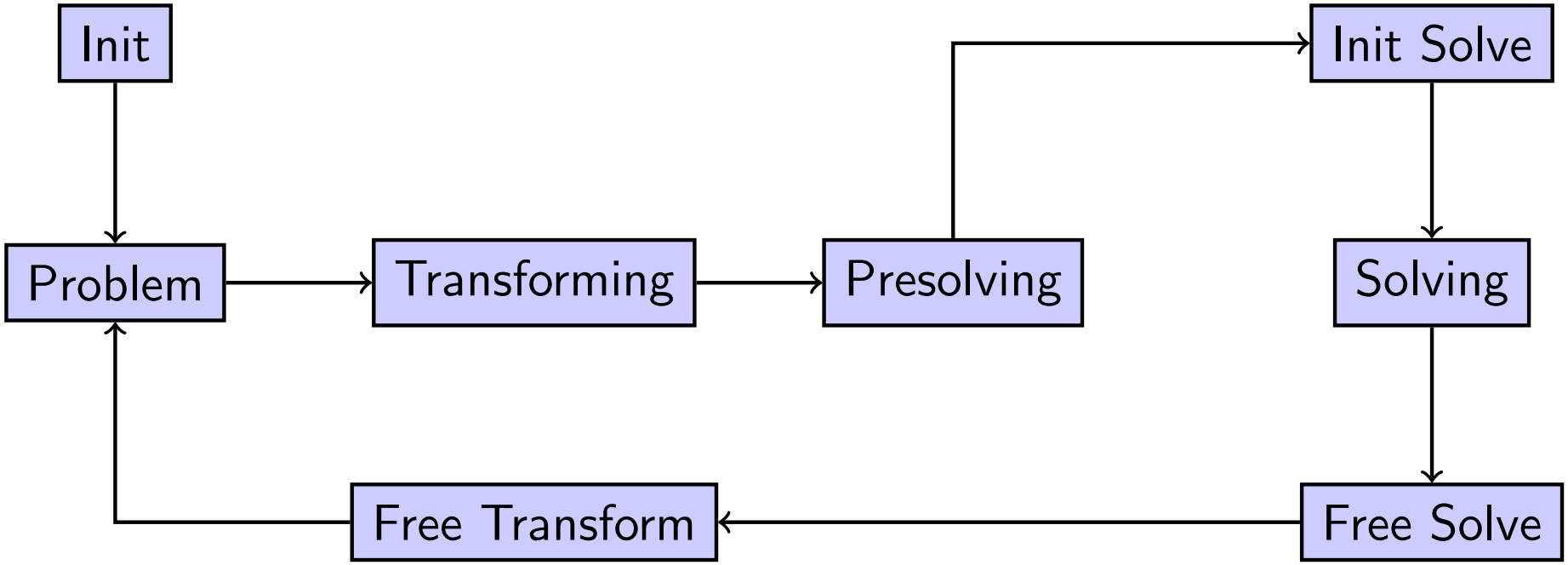
Task

- Analyze infeasibility
- Derive valid constraints
- Help to prune other nodes

Techniques

- Analyze:
  - Propagation conflicts
  - Infeasible LPs
  - Bound-exceeding LPs
  - Strong branching conflicts
- Detection:
  - Cut in conflict graph
  - LP: Dual ray heuristic
- Use conflicts:
  - Only for propagation
  - As cutting planes

Operational Stages



Implementation Hints

- SCIP has own memory handling.
  - Standard memory: `SCIPallocMemoryArray(scip,&p,10)`, `SCIPfreeMemoryArray(scip,&p)`
  - Block memory: `SCIPallocBlockMemoryArray(scip,&p,10)`, `SCIPfreeBlockMemoryArray(scip,&p,10)`
  - Fast buffer: `SCIPallocBufferArray(scip,&p,10)`, `SCIPfreeBufferArray(scip,&p)`
- Functions calling `SCIP_RETCODE <function>()` should use the `SCIP_CALL()` macro and should return `SCIP_RETCODE`.
- There are template files, e.g. `cons_xyz.{c,h}`.
- Only include `scip/scip.h`, `pub_*.h`, or `scip/scipdefplugins.h`.

Documentation

- Constraint Integer Programming
  Ph.D. dissertation of Tobias Achterberg, 2007.
- Solving mixed integer linear and nonlinear problems using the SCIP Optimization Suite
  ZIB Report, 2012.

- `http://scip.zib.de`
  Doxygen documentation, HowTos, FAQ
- See: `scip.h`, `pub_*.h`, `<plugin>.h` (e.g. `cons_linear.h`)
- If nothing helps: SCIP mailing list (scip@zib.de)

Wrap-up

SCIP is...

- a solver for MIP and MINLP
- a framework for branch-and-bound based algorithms (branch-and-cut, branch-and-price, ...)
- part of the SCIP Optimization Suite

Advantages of SCIP:

- broad scope ($\rightarrow$ next two days)
- available in source code ($\rightarrow$ coming next)
- actively developed ($\rightarrow$ tomorrow)
- hundreds of parameters to play with ($\rightarrow$ afternoon)
- easily extendable ($\rightarrow$ afternoon)

Today's Schedule

09:00 – 10:30	Introduction and Overview
10:30 – 11:00	Coffee Break
11:00 – 12:30	Installation and Testing Environment
12:30 – 14-00	Lunch Break
14:00 – 15:00	Parameter Tuning
15:00 – 15:30	Coffee Break
15:30 – 17:30	Programming Exercise

WiFi:

- eduroam
- "Gast im ZIB" (no password)