Sat4j, un moteur libre de raisonnement en logique propositionnelle

Habilitation à diriger des recherches

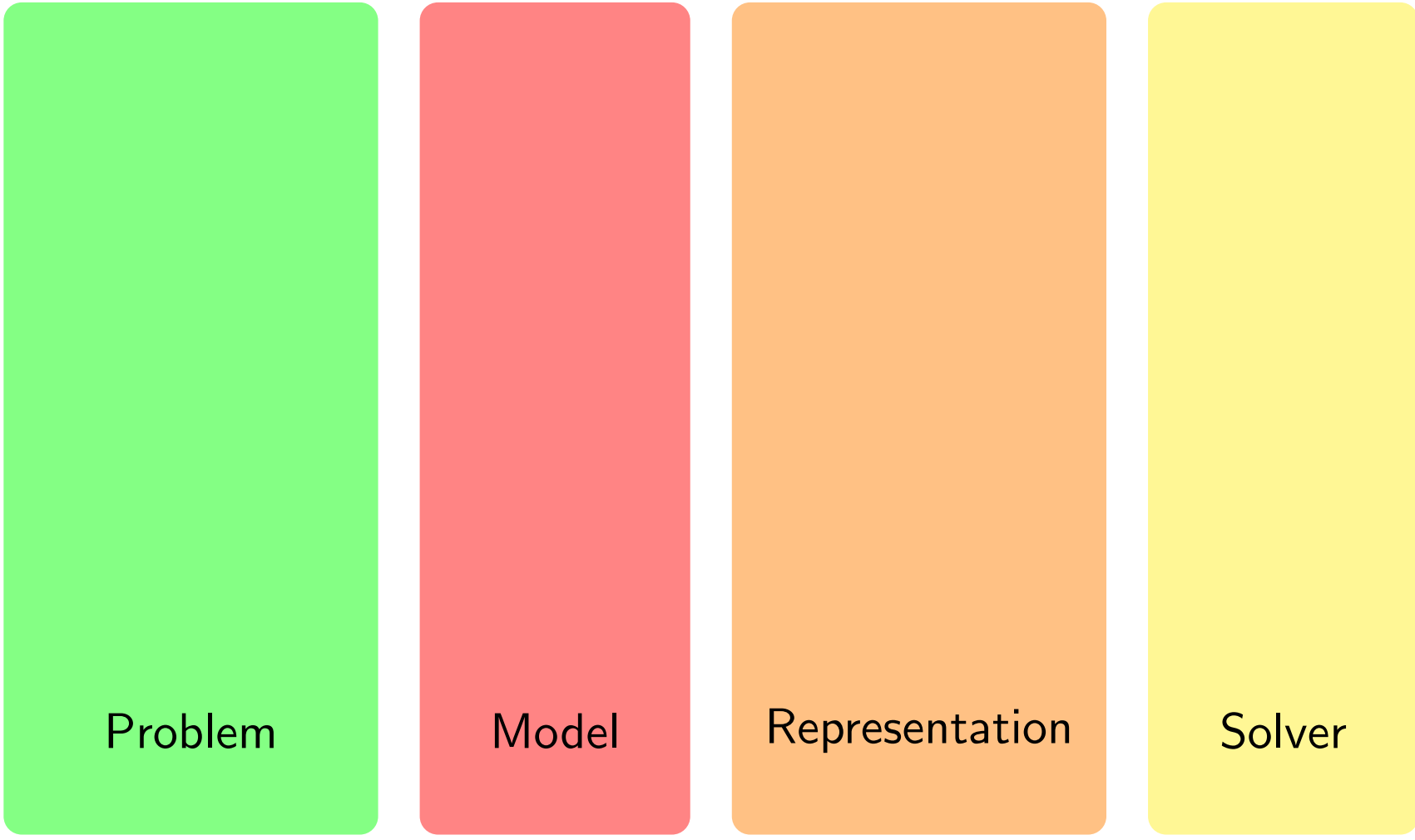
Daniel Le Berre

CRIL-CNRS UMR 8188 - Université d'Artois

Vendredi 3 décembre 2010, Lens

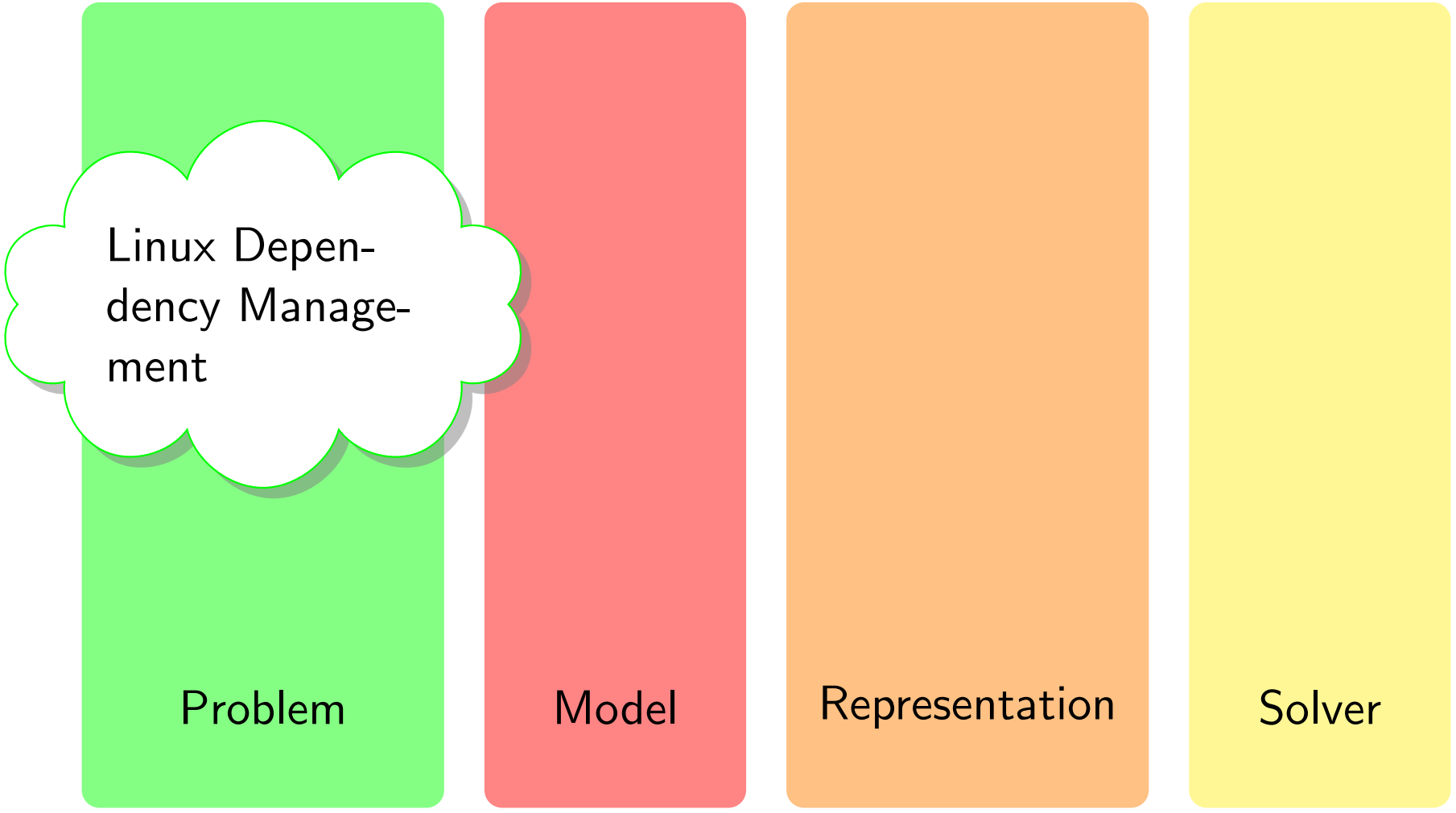
Outline

Solving real problems with a constraint solver

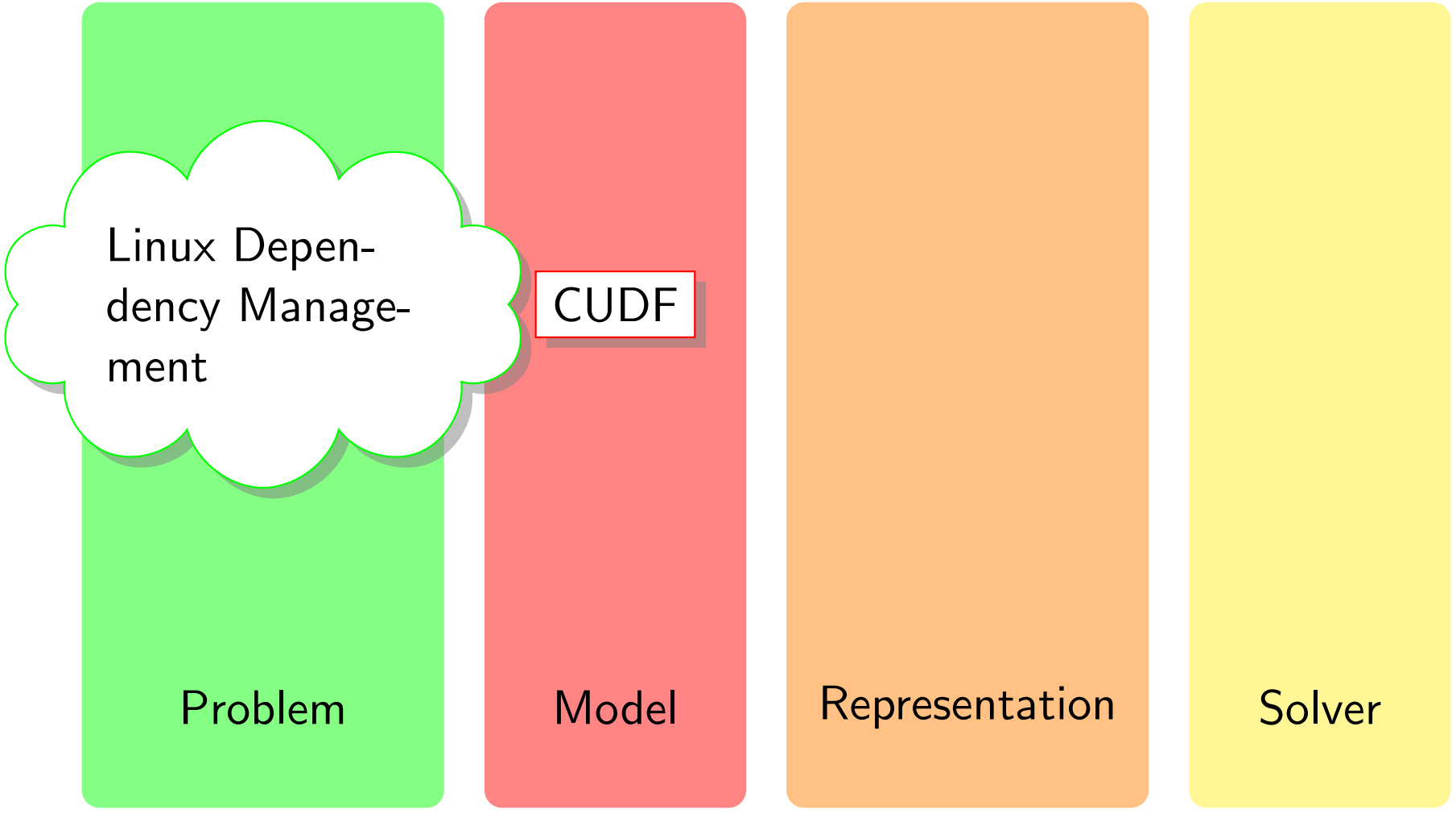


UNIVERSITÉ D'ARTOIS

Solving real problems with a constraint solver

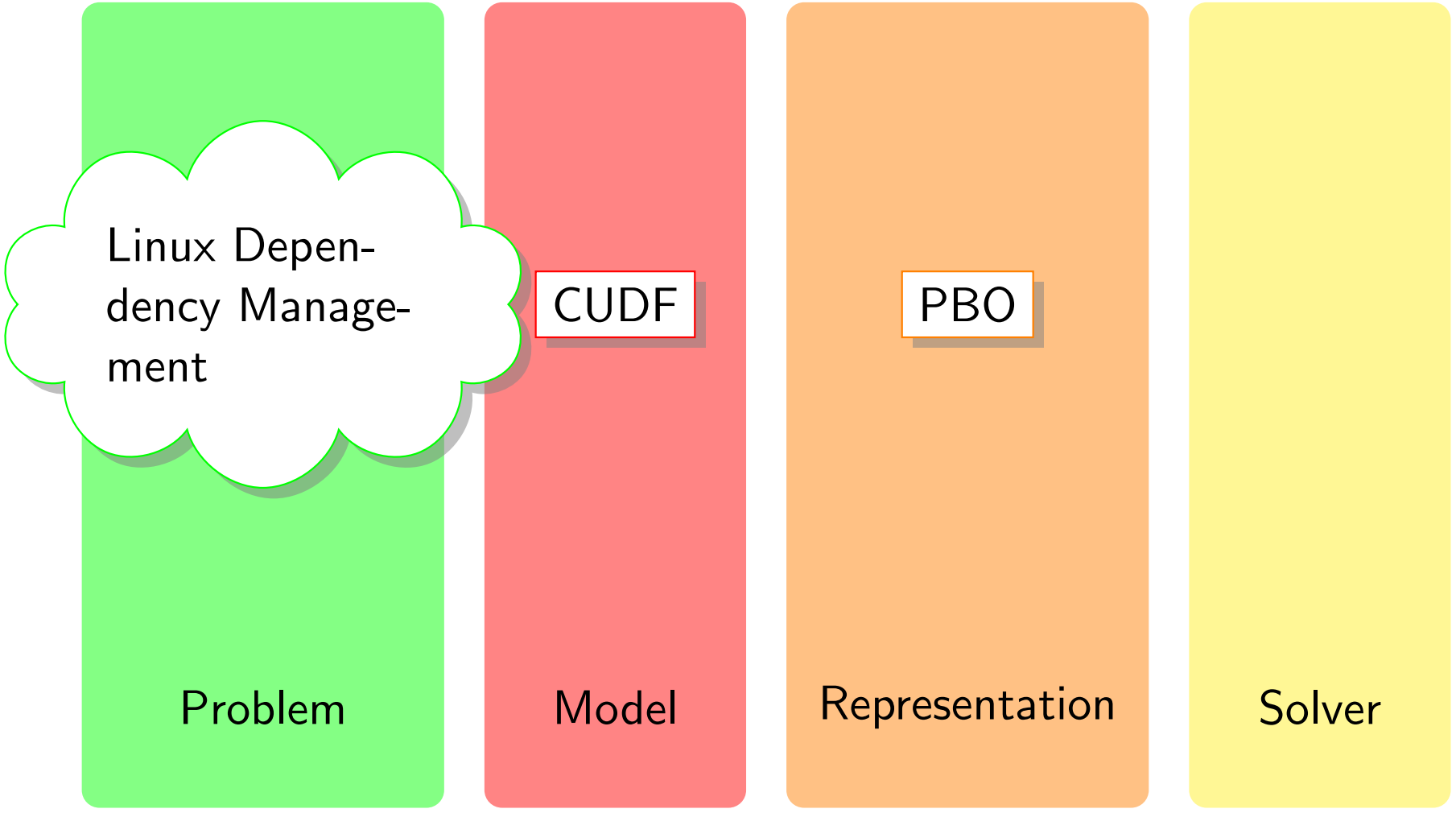


Solving real problems with a constraint solver



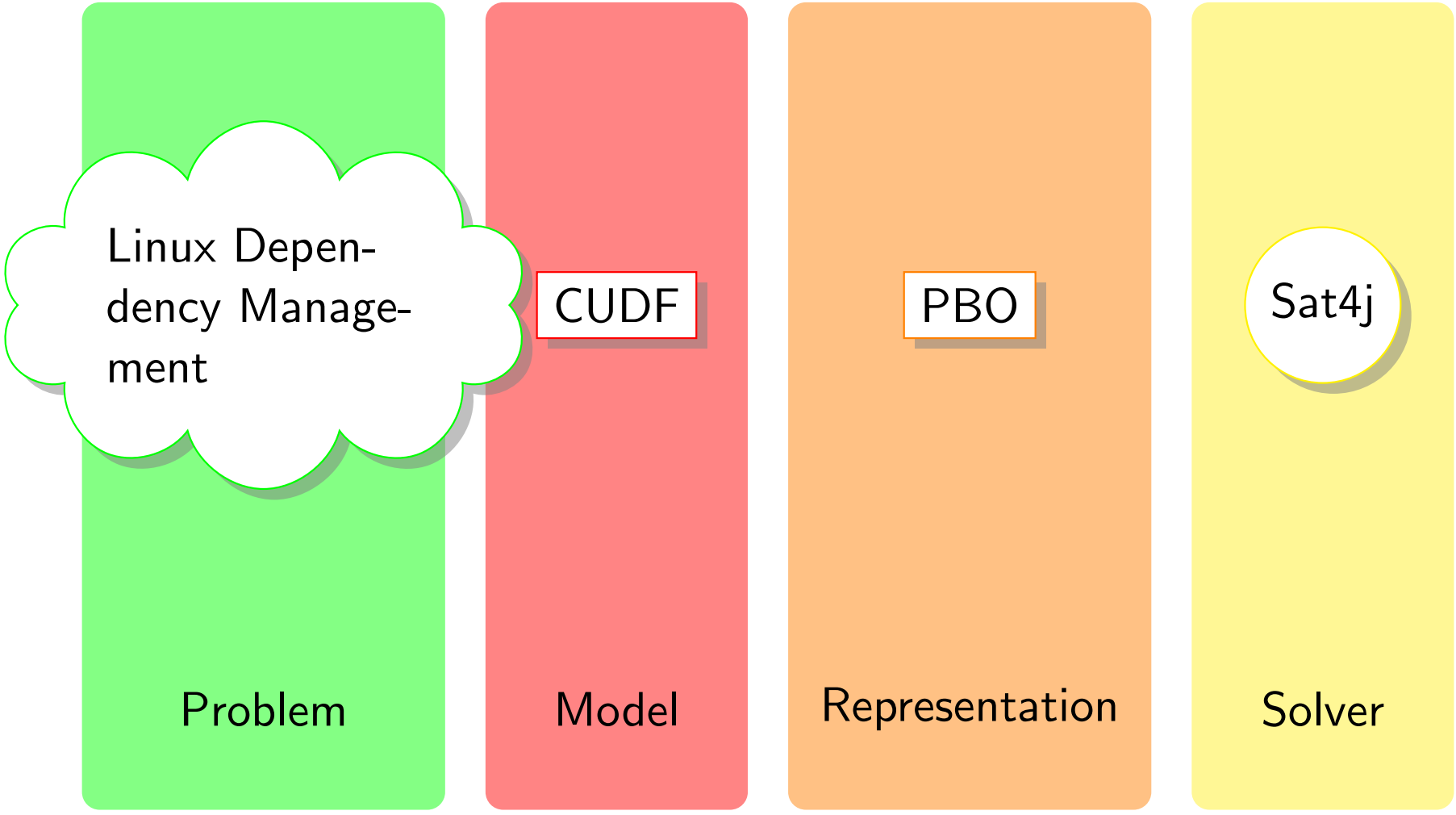
UNIVERSITÉ D'ARTOIS

Solving real problems with a constraint solver



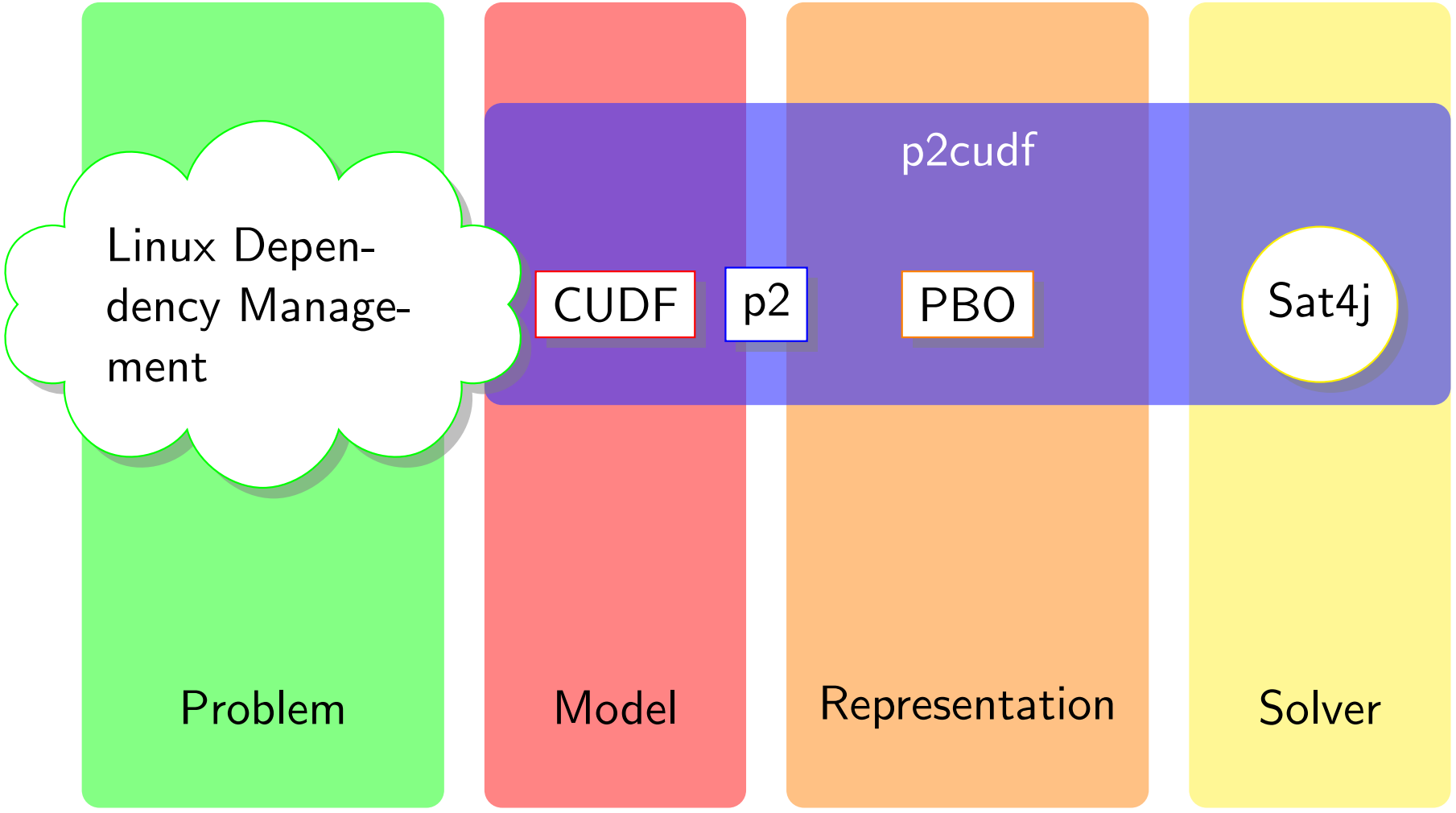
UNIVERSITÉ D'ARTOIS

Solving real problems with a constraint solver

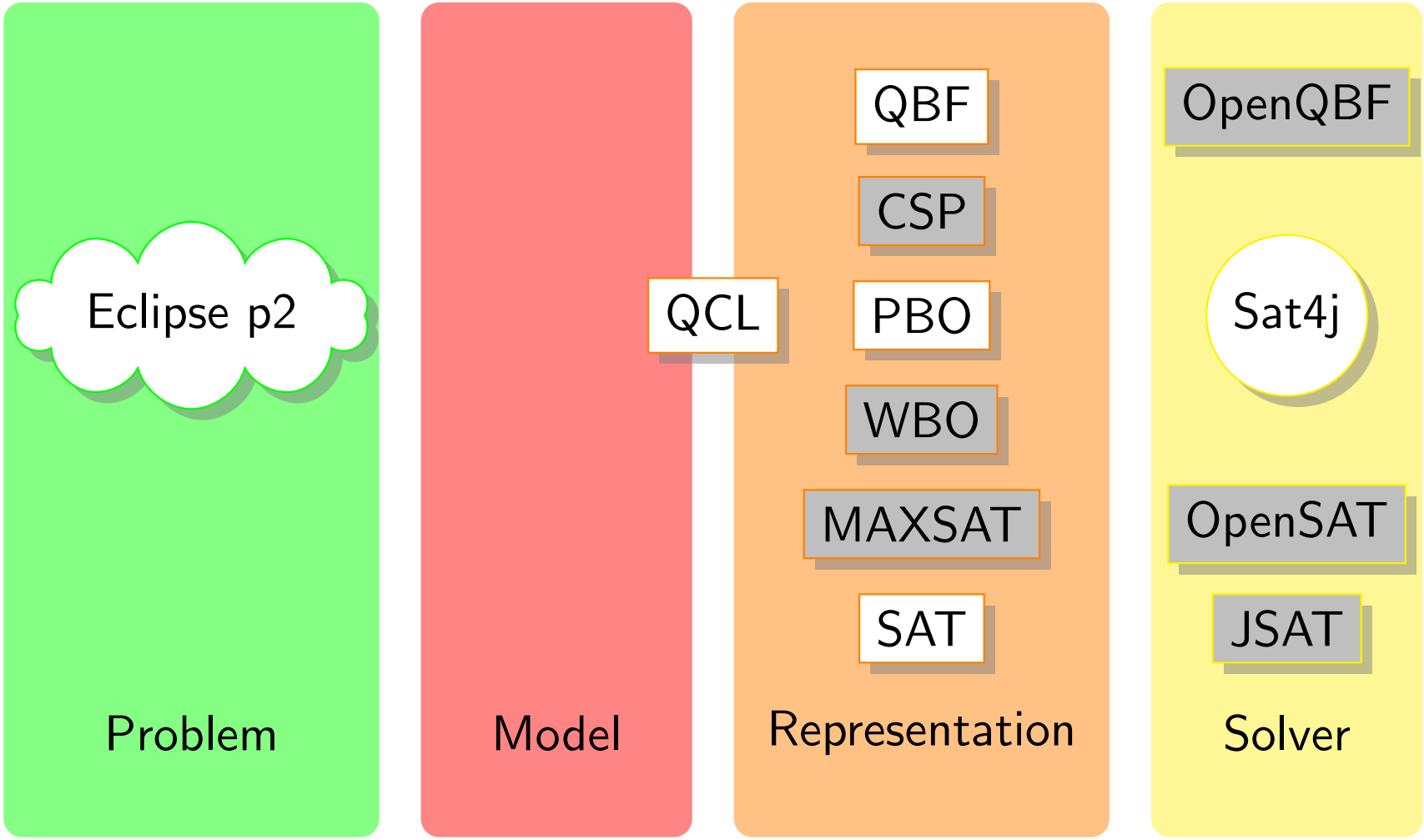


UNIVERSITÉ D'ARTOIS

Solving real problems with a constraint solver



Solving real problems with a constraint solver



UNIVERSITÉ D'ARTOIS

Outline

UNIVERSITÉ D'ARTOIS

Current softwares are composite !

- Linux distributions : made of packages
- Eclipse application : made of plugins
- Any complex software : made of libraries
- There are requirements between the diverse components

Dependency Management Problem

P a set of packages

depends requirement constraints

$$depends : P \rightarrow 2^{2^P}$$

conflicts impossible configurations

$$conflicts : P \rightarrow 2^P$$

Definition (consistency of a set of packages)

$Q \subseteq P$ is consistent with $(P, depends, conflicts)$ iff
$\forall q \in Q, (\forall dep \in depends(q), dep \cap Q \neq \emptyset) \wedge (conflicts(q) \cap Q = \emptyset)$.

Dependency Management Problem

P a set of packages

depends requirement constraints

$$depends : P \rightarrow 2^{2^P}$$

conflicts impossible configurations

$$conflicts : P \rightarrow 2^P$$

Definition (consistency of a set of packages)

$Q \subseteq P$ is consistent with $(P, depends, conflicts)$ iff
$\forall q \in Q, (\forall dep \in depends(q), dep \cap Q \neq \emptyset) \wedge (conflicts(q) \cap Q = \emptyset)$.

What is the complexity of finding if a Q containing a specific package exists ?

Just as hard as SAT : NP-complete !

See how to decide satisfiability of $(\neg a \vee b \vee c) \wedge (\neg a \vee \neg b \vee c) \wedge a \wedge \neg c$

```
package: a
version: 1
conflicts: a = 2

package: a
version: 2
conflicts: a = 1

package: b
version: 1
conflicts: b = 2

package: b
version: 2
conflicts: b = 1

package: c
version: 1
conflicts: c = 2

package: c
version: 2
conflicts: c = 1
```

```
package: clause
version: 1
depends: a = 2 | b = 1 | c = 1

package: clause
version: 2
depends: a = 2 | b = 2 | c = 1

package: clause
version: 3
depends: a = 1

package: clause
version: 4
depends: c = 2

package: formula
version: 1
depends: clause = 1, clause = 2,
         clause = 3, clause = 4

request: satisfiability
install: formula
```

Dependencies expressed by clauses

- Dependencies can easily be translated into clauses :

```
package: a
version: 1
depends: b = 2 | b = 1, c = 1
```

$$a_1 \rightarrow (b_2 \vee b_1) \wedge c_1$$

$$\neg a_1 \vee b_2 \vee b_1, \neg a_1 \vee c_1$$

- Conflict can easily be translated into binary clauses :

```
package: a
version: 1
conflicts: b = 2, d = 1
```

$$\neg a_1 \vee \neg b_2, \neg a_1 \vee \neg d_1$$

From decision to optimization

- NP-complete, so we can use a SAT solver to solve it
- Finding a solution is usually not sufficient !
  - Minimizing the number of installed packages
  - Minimizing the size of installed packages
  - Keeping up to date versions of packages
  - Preferring most recent packages to older ones
  - ...
- In practice an aggregation of various criteria
- Need a more expressive representation language than plain CNF !

Outline

Representing with ~~MaxSat~~ MinUnsat

- Associate to each constraint (clause) a weight (penalty) w_i taken into account if the constraint is violated : Soft constraints ϕ (optional, recommended packages).
- Special weight (∞) for constraints that cannot be violated : hard constraints α (dependencies, conflicts)
- Find a model I of α that minimizes $weight(I, \phi)$ such that :
  - $weight(I, (c_i, w_i)) = 0$ if I satisfies c_i, else w_i.
  - $weight(I, \phi) = \sum_{wc \in \phi} weight(I, wc)$

Weight	∞	denomination
∞	yes	Sat
k	no	MaxSat
k	yes	Partial MaxSat
$\mathbb{N}$	no	Weighted MaxSat
$\mathbb{N}$	yes	Weighted Partial MaxSat

Representing optimization criteria with MaxSat ?

$\alpha \equiv \bigwedge_{p_v \in P}(p_v \rightarrow (\bigwedge_{dep \in depends(p_v)} dep), \infty) \wedge$
$\bigwedge_{conf \in conflicts(p_v)}(p_v \rightarrow \neg conf, \infty,) \wedge (q, \infty)$
denote the formula to satisfy for installing q.

Minimizing the number of installed packages (Partial MaxSat) :

$$\phi \equiv (\bigwedge_{p_v \in P, p_v \neq q} (\neg p_v, k)) \quad (1)$$

Minimizing the size of installed packages (Weighted Partial MaxSat) :

$$\phi \equiv (\bigwedge_{p_v \in P, p_v \neq q} (\neg p_v, size(p_v))) \quad (2)$$

Those problems are really Binate Covering Problems (CNF + objective function).

Linear Pseudo-Boolean decision and optimization problems

Linear Pseudo-Boolean constraint

$$-3x_1 + 4x_2 - 7x_3 + x_4 \leq -5$$

- variables x_i take their value in $\{0, 1\}$
- $\overline{x_1} = 1 - x_1$
- coefficients and degree are integral constants

Pseudo-Boolean decision problem : NP-complete

$$\begin{cases} (a_1) & 5x_1 + 3x_2 + 2x_3 + 2x_4 + x_5 \geq 8 \\ (a_2) & 5\overline{x_1} + 3\overline{x_2} + 2\overline{x_3} + 2\overline{x_4} + \overline{x_5} \geq 5 \\ (b) & x_1 + x_3 + x_4 \geq 2 \\ (c) & x_1 + \overline{x_2} + x_5 \geq 1 \end{cases}$$

Plus an objective function : Optimization problem, NP-hard

$$min : 4x_2 + 2x_3 + x_5$$

Representing optimization criteria using pseudo-boolean optimization

- We can now rewrite the previous optimization criteria in a simpler manner :
  - Minimizing the number of installed packages :

$$min : \sum_{p_v \in P, p_v \neq q} p_v$$

  - Minimizing the size of installed packages :

$$min : \sum_{p_v \in P, p_v \neq q} size(p_v) \times p_v$$

- We can express easily that only one version of package *libnss* can be installed :
  $libnss_1 + libnss_2 + libnss_3 + libnss_4 + libnss_5 \leq 1$

Using QCL to express preferences between versions

- QCL adds a new connective $\times$ to propositional logic to order alternatives : $firefox_{36} \times firefox_{25}$
- QCL is non monotonic

$$firefox_{36} \times firefox_{25} \models firefox_{36}$$
$$(firefox_{36} \times firefox_{25}) \wedge \neg firefox_{36} \models firefox_{25}$$
$$(firefox_{36} \times firefox_{25}) \wedge \neg firefox_{36} \wedge \neg firefox_{25} \models \bot$$

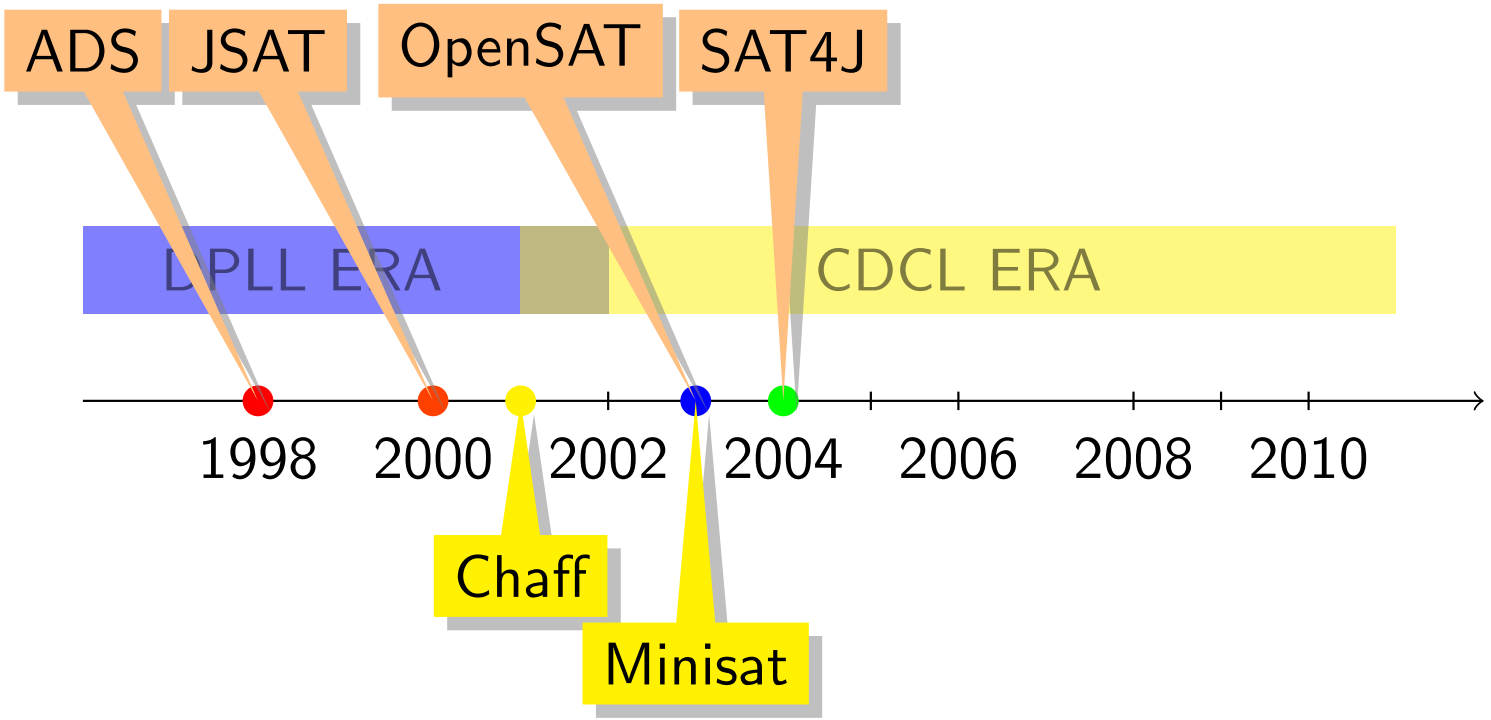
- QCL allows preferences to be embedded in any propositional formula :
  $gnome_{230} \rightarrow (evolution_{230} \times thunderbird_{30}) \wedge$
  $\neg gnome_{230} \rightarrow (thunderbird_{30} \times (kmail_3 \vee kmail_4) \times evolution_{230})$

How to put everything together ?

- Use PBO as main representation language.
- Translate MaxSat into an equivalent PBO problem (see later)
- Integration of QCL formula
  - Put in normal form (as one single basic choice formula)
  - Translate into a PBO problem (through WPMS)
- Main issue : we enter multi-criteria optimization !

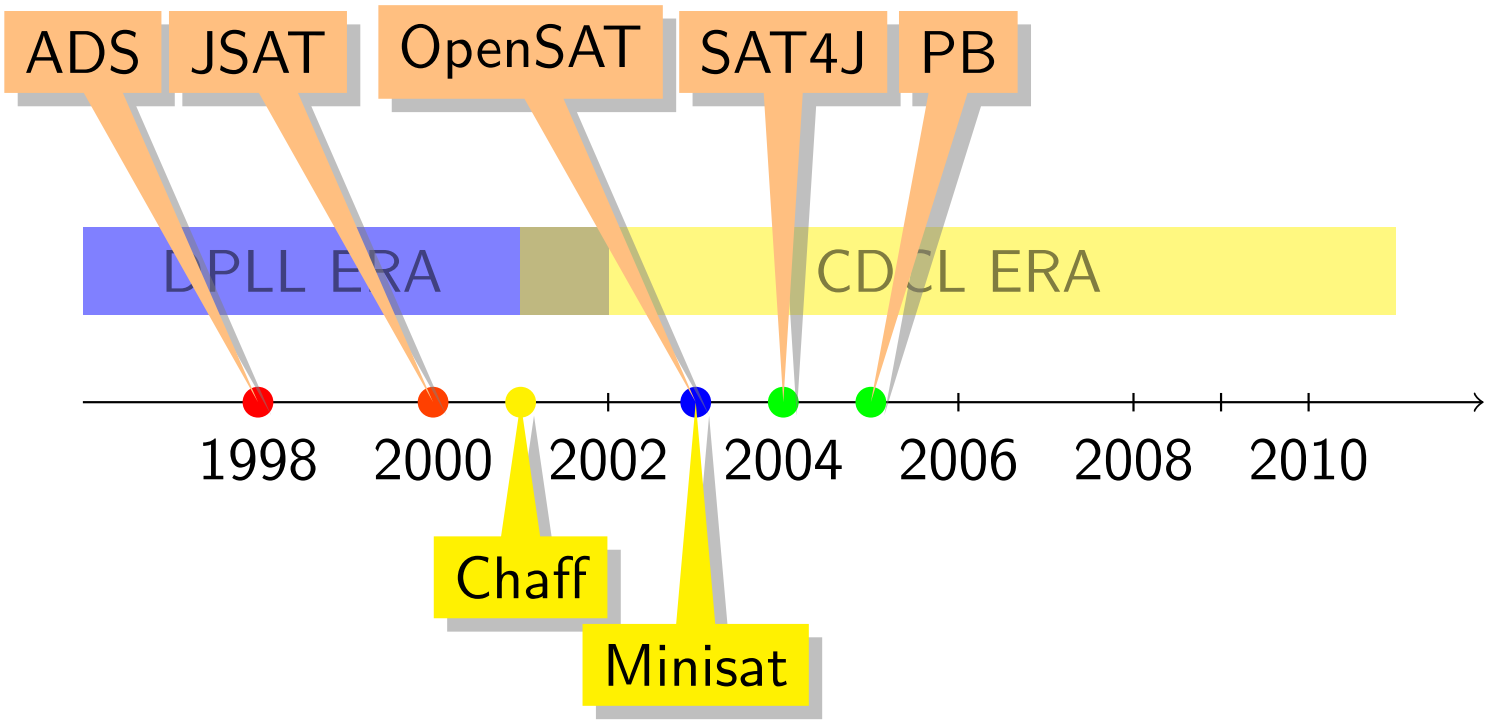
Outline

Sat4j, from ADS to p2cudf



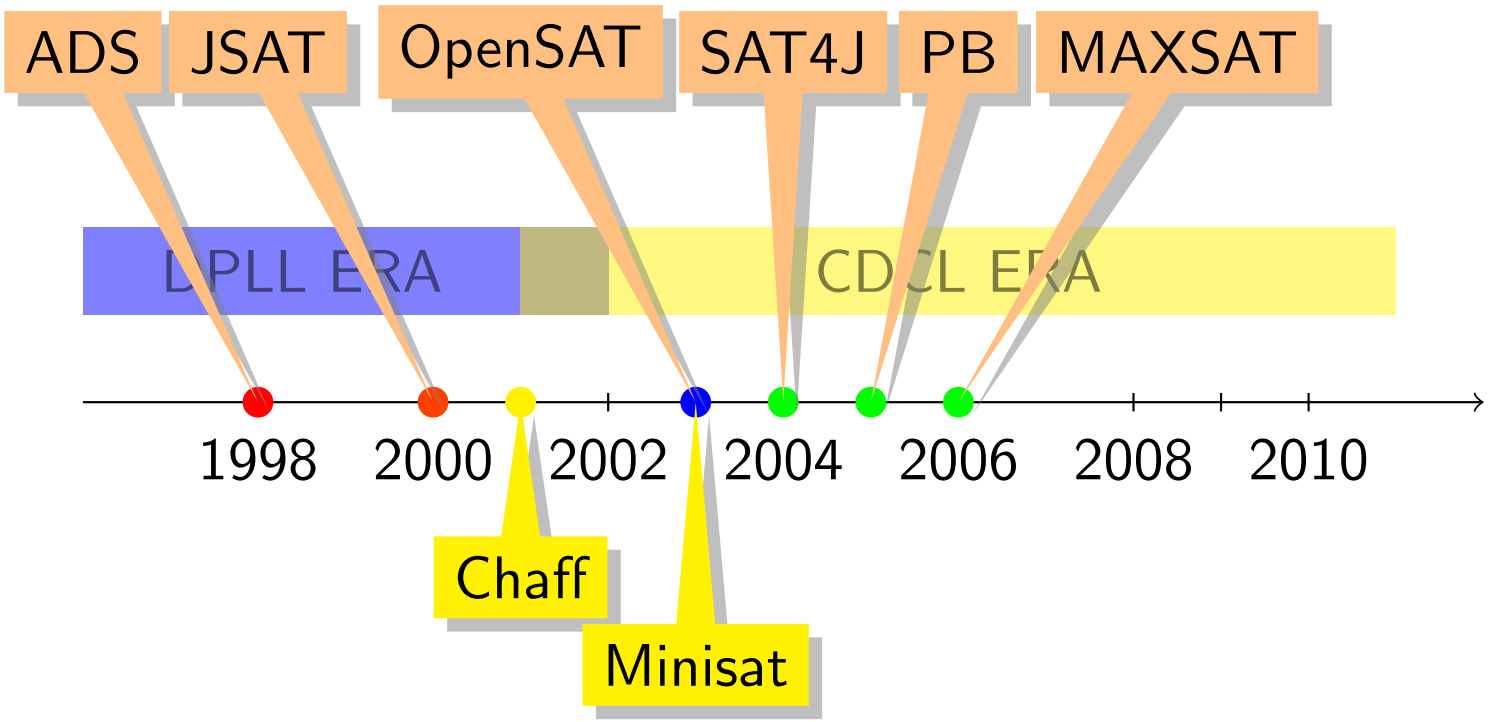
- Birth of SAT4J, Java implementation of Minisat.
- Open Source (licensed under GNU LGPL).

Sat4j, from ADS to p2cudf



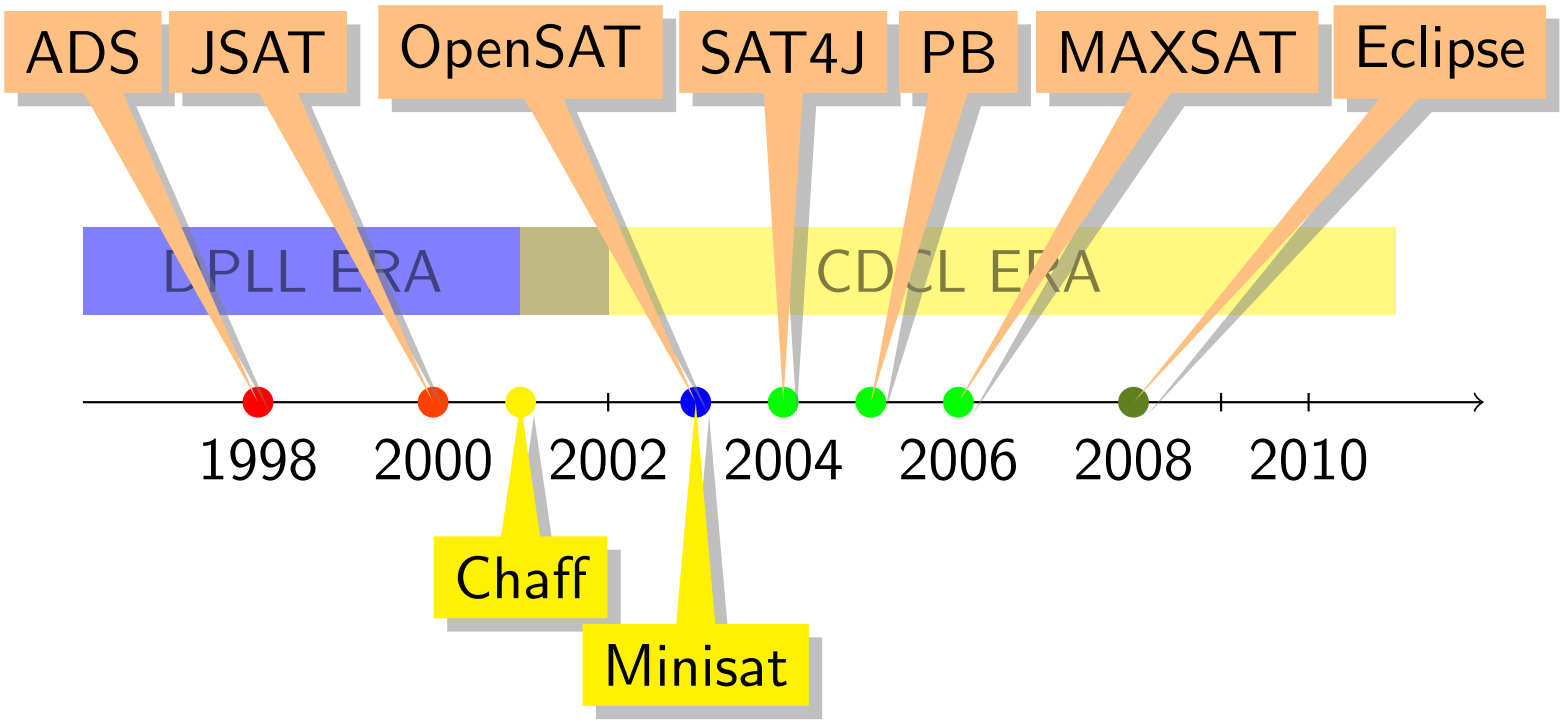
- Joint project with INESC → PB evaluation + Sat4j PB
- First CSP competition → release of Sat4j CSP

Sat4j, from ADS to p2cudf



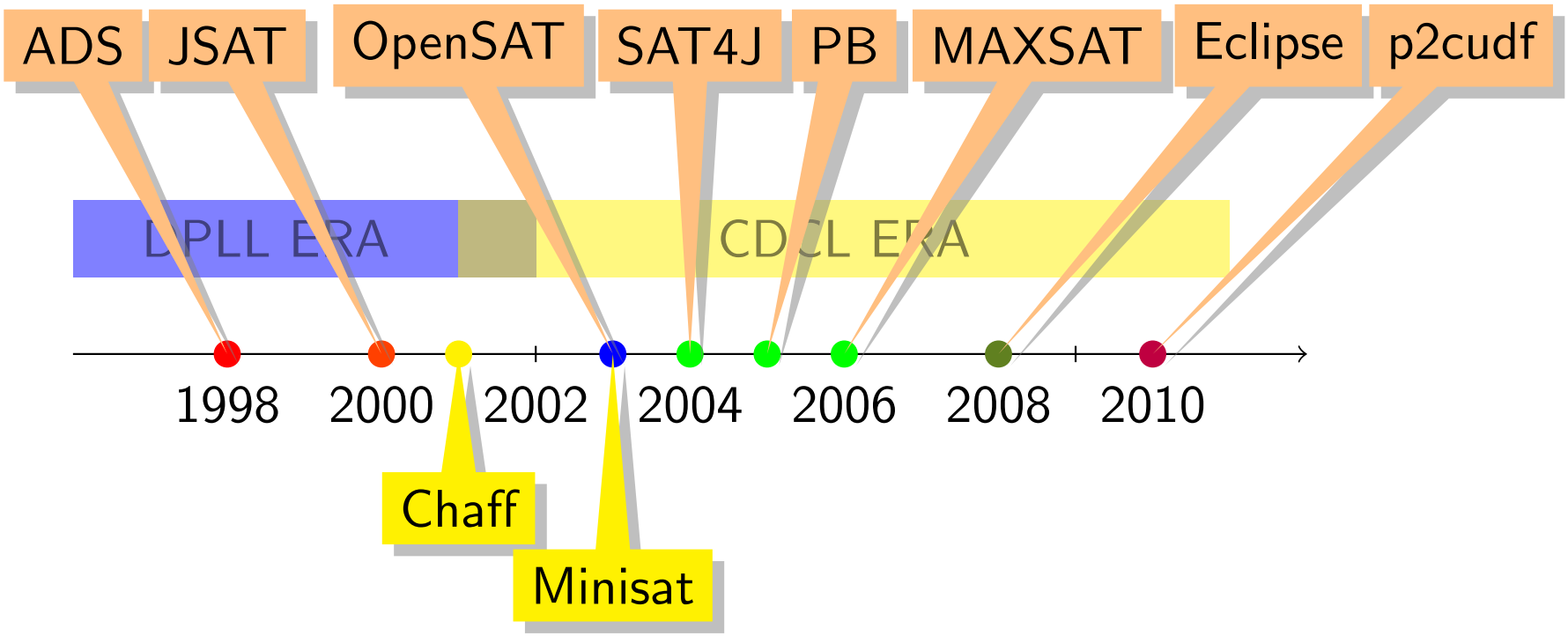
- First MaxSAT evaluation
- Birth of Sat4j MaxSAT

Sat4j, from ADS to p2cudf



- Integration within Eclipse [IWOCE09]
- Sat4j relicensed under both EPL and GNU LGPL

Sat4j, from ADS to p2cudf

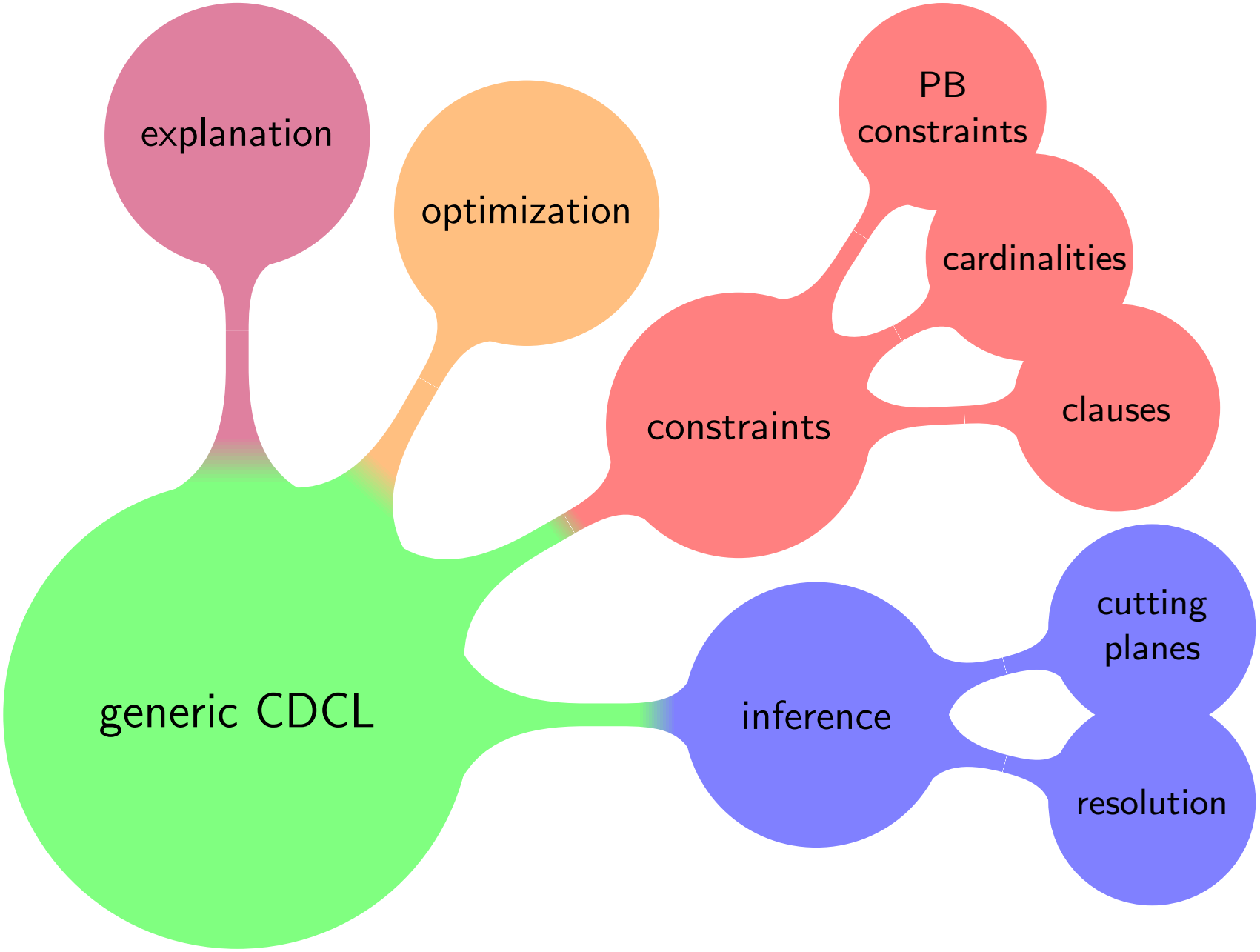


- Application to Linux dependencies : p2cudf [LoCoCo 2010]
- Licensed under EPL

Initial aim of the project

- Providing 100% Java SAT technology
- With an Open Source software
- Flexible enough to experiment our ideas
- Efficient enough to solve real problems
- Designed to be used in academia or production software

A flexible framework for solving propositional problems



A generic and flexible CDCL solver

Basis Minisat 1.10 specification + conflict minimization from Minisat 1.13

Static Restarts strategies Minisat, Biere, Luby

Generic Conflict minimization None, Simple, Expensive
works with all constraints and data structures

Learning LimitedLearning, LearnAllClauses, NoLearning, ...
learning is not coupled with conflict analysis

Learned clauses deletion Memory based, Glucose

Phase selection Random, Positive, Negative, AppearInLastLearnedClauses, RSAT phase saving

Lazy Data structures Watched Literals, Head/Tail

Default configuration

Pseudo Boolean solvers in SAT4J

SAT4J PB RES learn clauses. takes advantage of the full existing SAT machinery.

SAT4J PB CuttingPlanes learn PB constraints. No lazy data structure for constraints, need arbitrary precision arithmetic for correctness.

- The resolution based PB solver is usually faster than the CP based one.
- Some benchmarks can only be solved using CP solver (e.g. pigeon hole).
- The principles behind each solver are clear : no tweaks to solve a few more benchmarks during the PB evaluations !
- New in PB 2010 (Sat4j 2.2.1) : running both in parallel for improved wall clock running time.

Generalized use of selector variables

The minisat+ syndrom : is a SAT solver sufficient for all our needs ?

Selector variable principle : satisfying the selector variable should satisfy the selected constraint.

clause simply add a new variable

$$\bigvee l_i \quad \Rightarrow \quad s \vee \bigvee l_i$$

cardinality add a new weighted variable

$$\sum l_i \geq d \quad \Rightarrow \quad d \times s + \sum l_i \geq d$$

The new constraints is PB, no longer a cardinality !

pseudo add a new weighted variable

$$\sum w_i \times l_i \geq d \quad \Rightarrow \quad d \times s + \sum w_i \times l_i \geq d$$

if the weights are positive, else use

$$(d + \sum_{w_i<0} |w_i|) \times s + \sum w_i \times l_i \geq d$$

From Weighted Partial Max SAT to PBO

Once cardinality constraints, pseudo boolean constraints and objective functions are managed in a solver, one can easily build a weighted partial Max SAT solver

- Add a selector variable s_i per soft clause C_i : $s_i \vee C_i$
- Objective function : minimize $\sum s_i$
- Partial MAX SAT : no selector variables for hard clauses
- Weighted MAXSAT : use a weighted sum instead of a sum.
  Special case : do not add new variables for unit weighted clauses $w_k l_k$
  Ignore the constraint and add simply $w_k \times \overline{l_k}$ to the objective function.

Selector variables + assumptions = explanation

- From the beginning in Minisat 1.12
- Add a new selector variable per constraint
- Check for satisfiability assuming that the selector variables are falsified
- if UNSAT, analyze the final root conflict to keep only selector variables involved in the inconsistency
- Apply a minimization algorithm afterward to compute a minimal explanation (QuickXplain)
- Advantages :
  - no changes needed in the SAT solver internals
  - works for any kind of constraints !
- See in action during the unsat core track of the next SAT competition !

SAT4J today

- SAT4J MAXSAT considered state-of-the-art on Partial [Weighted] MaxSAT application benchmarks (2009).
- SAT4J PB (Res, CP) are not very efficient, but correct (arbitrary precision arithmetic).
- SAT4J SAT solvers can be found in various software from academia (Alloy 4, Forge, ...) to commercial applications (GNA.sim).
- SAT4J PB Res solves Eclipse plugin dependencies since June 2008 (Eclipse 3.4, Ganymede)
  - SAT4J ships with every product based on the Eclipse platform (more than 25 millions downloads from Eclipse.org since June 2008)
  - SAT4J helps to build Eclipse products daily (e.g. nightly builds on Eclipse.org, IBM, SAP, etc)
  - SAT4J helps to update Eclipse products worldwide daily

Outline

DP 60 and DLL62 : the very first SAT competition !

In the present paper, a uniform proof procedure for quantification theory is given which is feasible for use with some rather complicated formulas and which does not ordinarily lead to exponentiation. The superiority of the present procedure over those previously available is indicated in part by the fact that a formula on which Gilmore's routine for the IBM 704 causes the machine to compute for 21 minutes without obtaining a result was worked successfully by hand computation using the present method in 30 minutes [Davis and Putnam, 1960].

The well-formed formula (...) which was beyond the scope of Gilmore's program was proved in under two minutes with the present program [Davis et al., 1962]

A brief history of SAT competitive events

Since the beginning the SAT community has been keen to run competitive events

- 1992 : Paderborn
- 1993 : The second DIMACS challenge [standard input format] Johnson, D. S., & Trick, M. A. (Eds.). (1996). Cliques, Coloring and Satisfiability : Second DIMACS Implementation Challenge, Vol. 26 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science. AMS.
- 1996 : Beijing
- 1999 : SATLIB
- 2000 : SAT-Ex
- Since 2002 : yearly competition (or Race)

Why organizing a SAT competition ?

- Provide independent experimental results of existing SAT solvers :
  - Too many solvers and benchmarks available to maintain a SatEx like web site : use yearly snapshot.
  - Target the whole community : application, crafted and random benchmarks, complete/incomplete solvers.
- Foster the implementation of new SAT solvers.
- Gather new benchmarks.
- Promote SAT technology outside the community.
- Have fun :)

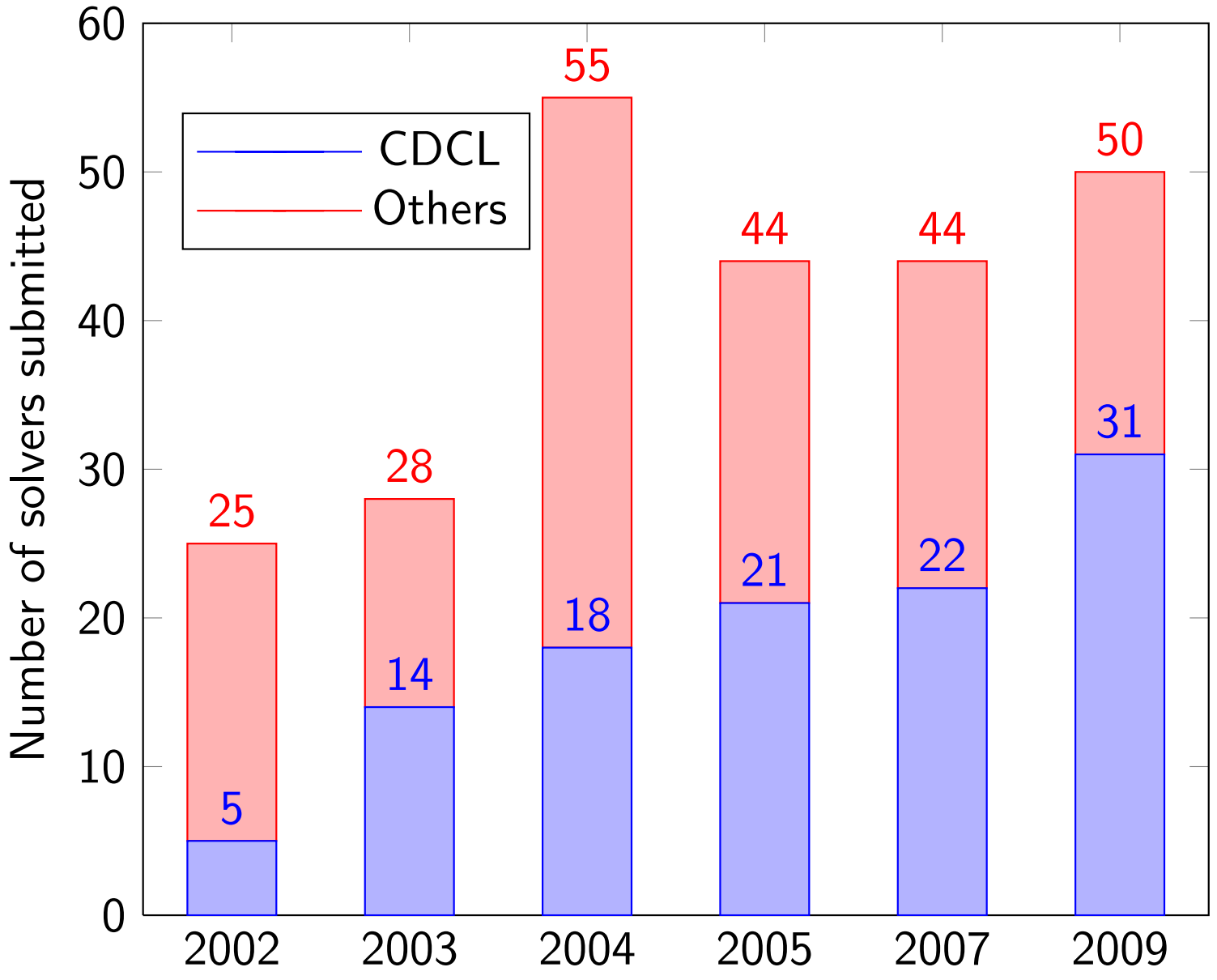
Limitations of the approach

Results depends on :

- available solvers and benchmarks.
- hardware (amount of RAM, size of L2 cache, etc).
- operating system (linux)
- competition rules (timeout, source code)
- the amount of computing resources available
- ...

We do not claim to have statistically meaningful results !

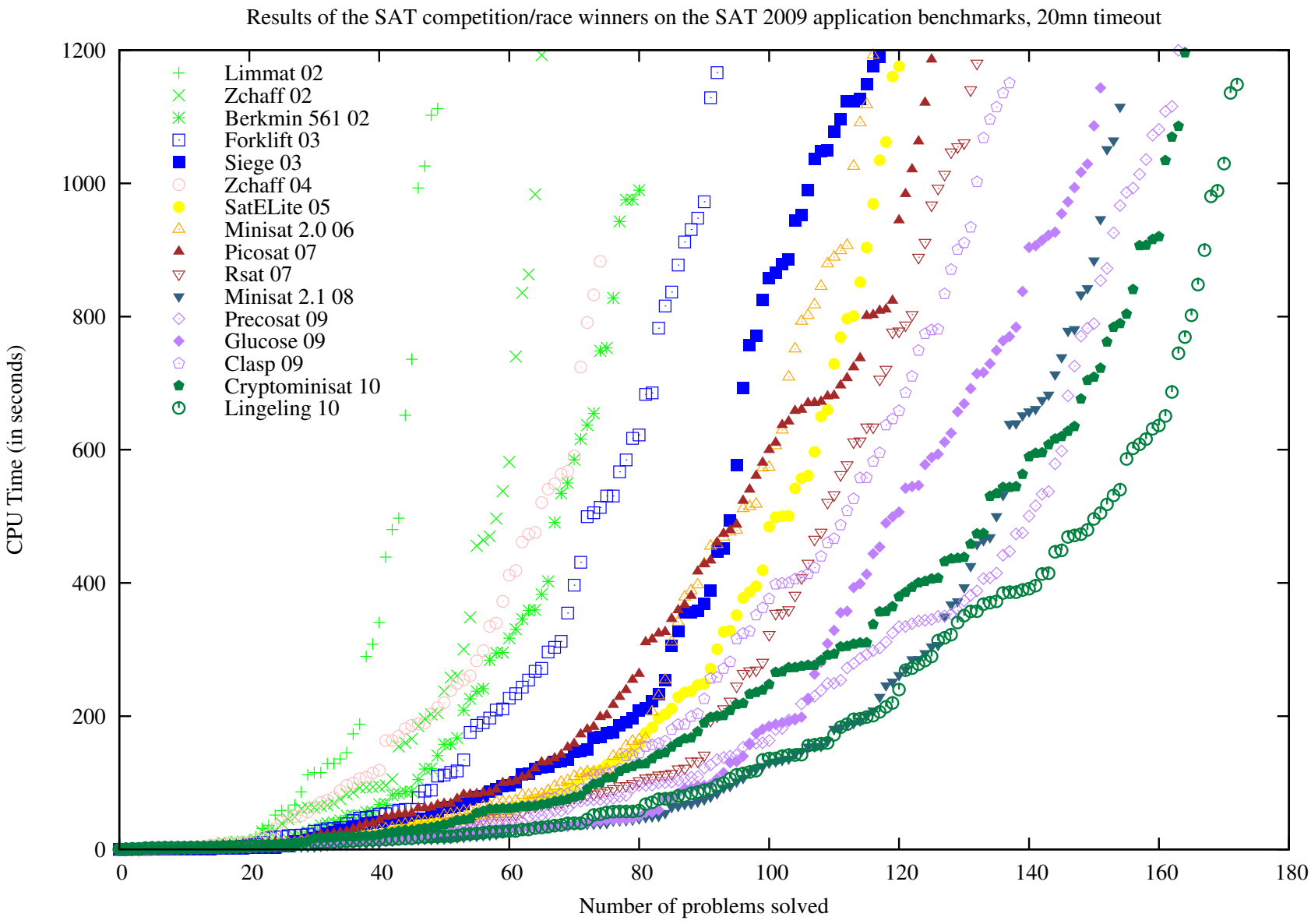
Huge success in the community !



Consequences of the competition

- Many efficient SAT solvers available for research purposes, several fully open source.
- SAT solvers are easier to use, more reliable.
- Online reference results.
- Many benchmarks gathered over the years.
- SAT technology is widely adopted in other area (Formal Verification, Software Engineering, Bioinformatics, etc).
- Some winners got a grant, a job, some money...
- Many offsprings : QBF, PB, MAXSAT, ASP, CSP, ...

What about raw performances ?



Lessons learned from the competition

- Allowing the author of a solver to validate its behavior improves the reliability of results ;
- Lower entry level when it becomes too hard (Minisat hack track) ;
- Make scoring scheme as simple and clear as possible ;
- Make the results easily browsable (web) ;
- Solvers and benchmarks should be made available for research purpose, preferably in source ;
- Do not be too tough with the competitors : enter fixed versions of solvers found buggy on the side of the competition for instance ;
- Be pragmatic (e.g. Siege, Eureka).

Requires a good hardware (cluster) and software (Sat-Ex from Laurent Simon, evaluation from Olivier Roussel) infrastructure.

Directions for improvements

- Taking into account multi-core architecture (wall-clock time vs CPU time, reproducibility) : see the next SAT competition.
- Better selection of benchmarks (TPTP/CASC, SMTLIB/SMT-COMP) ?
- Specific use case : interactive, batch, etc.
- Robustness assessment ?

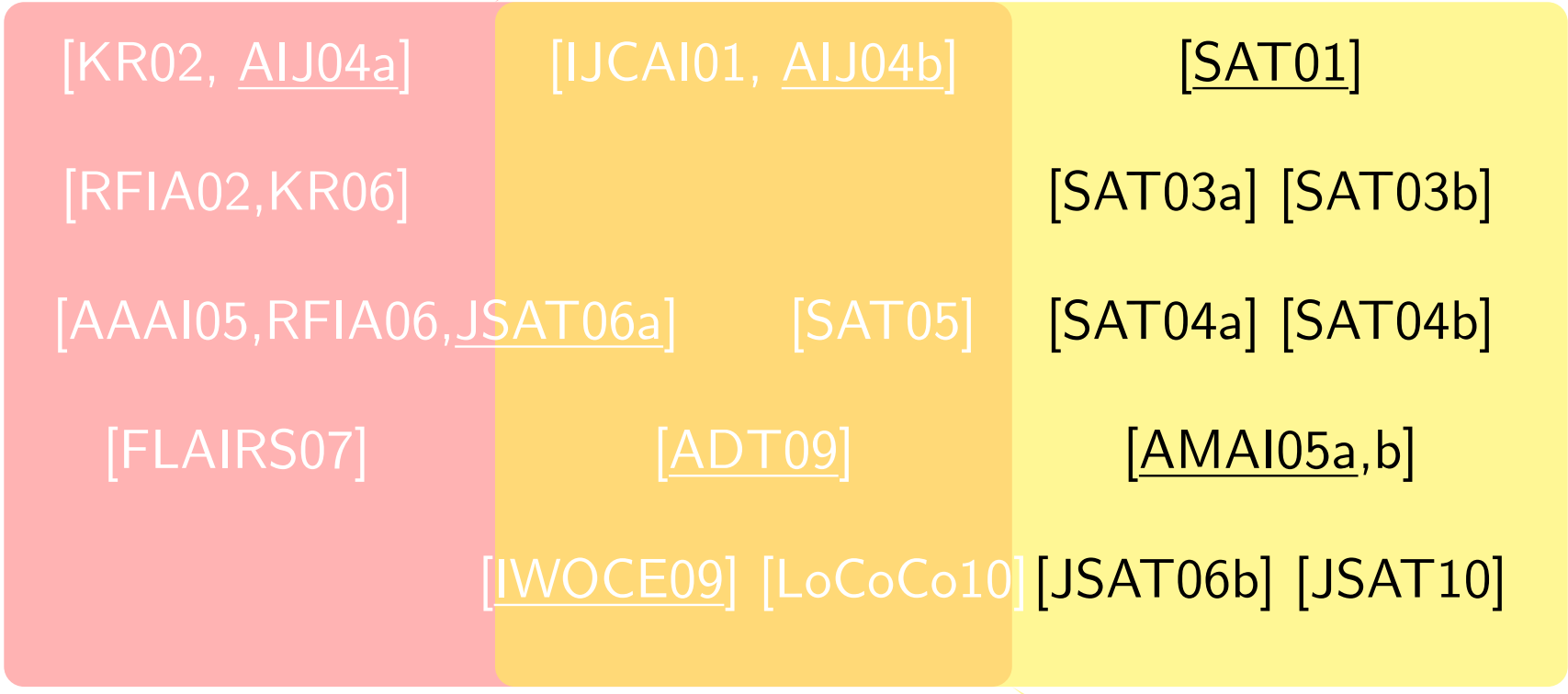
A note about FOSS in academia

- Without the availability of the source code of GRASP, SATO and RELSAT, no Chaff !
- Minisat is widely used because it is open source (MIT) and fast
- Without Sat4j being fully open source, and business friendly (EPL), no integration in Eclipse
- My opinion : drastic improvements in SAT solvers rely heavily on the exchange of knowledge through source code !
  (see the improvements in areas when the code of the solvers is not available ...)
- Tip to make "No commercial use" OSS friendly : GNU GPL + specific licenses

Outline

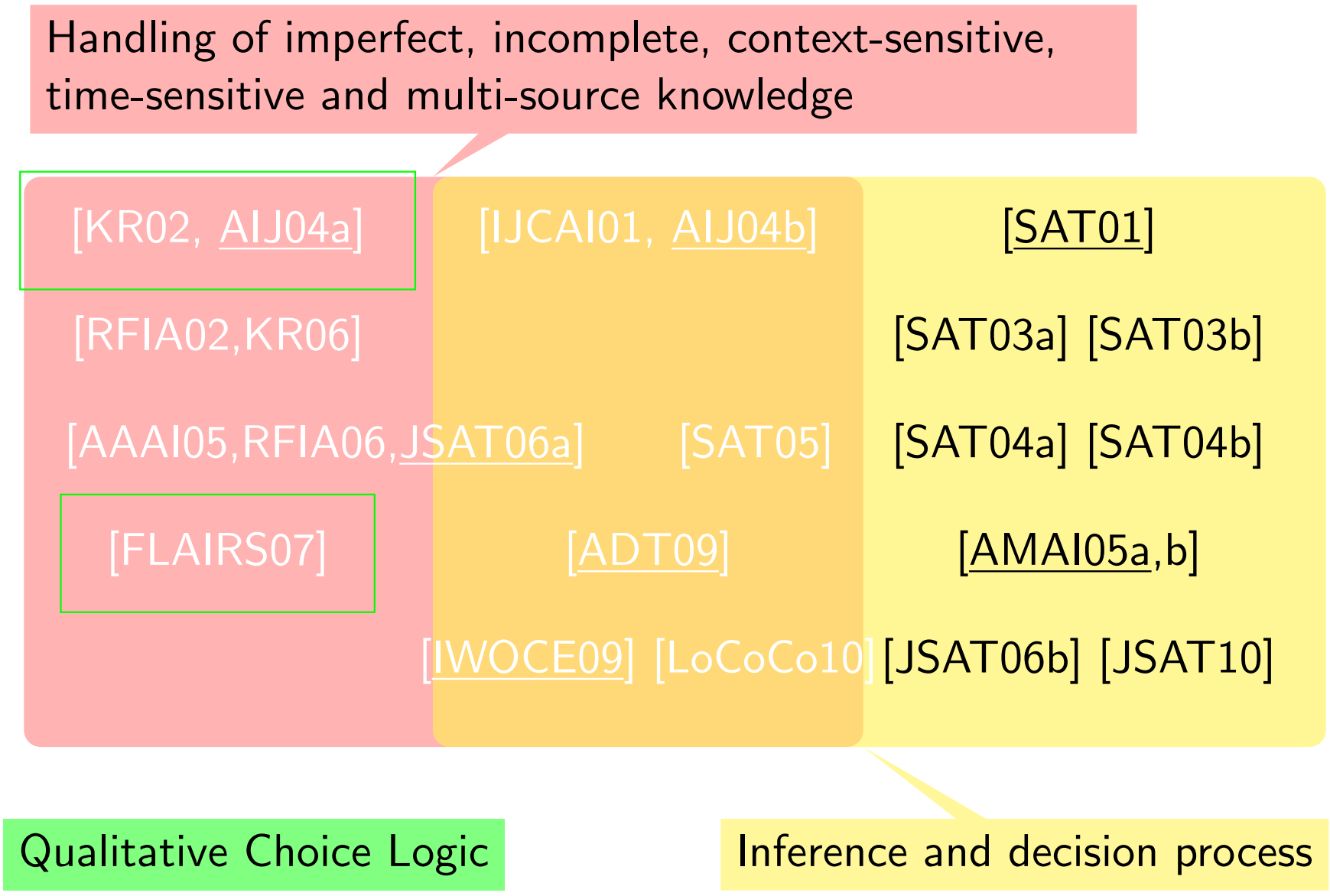
Summary of publications according to CRIL structure

Handling of imperfect, incomplete, context-sensitive, time-sensitive and multi-source knowledge

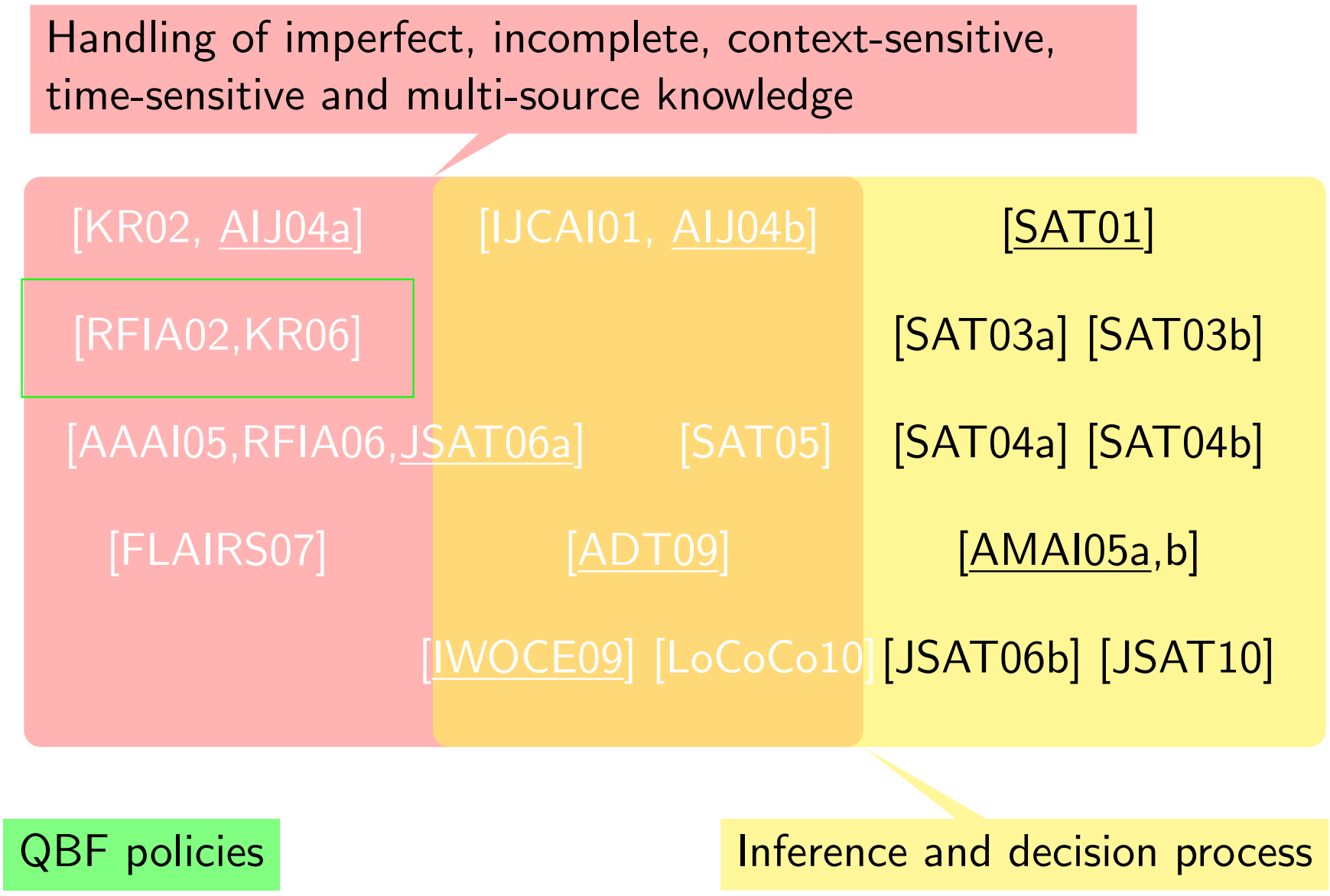


Inference and decision process

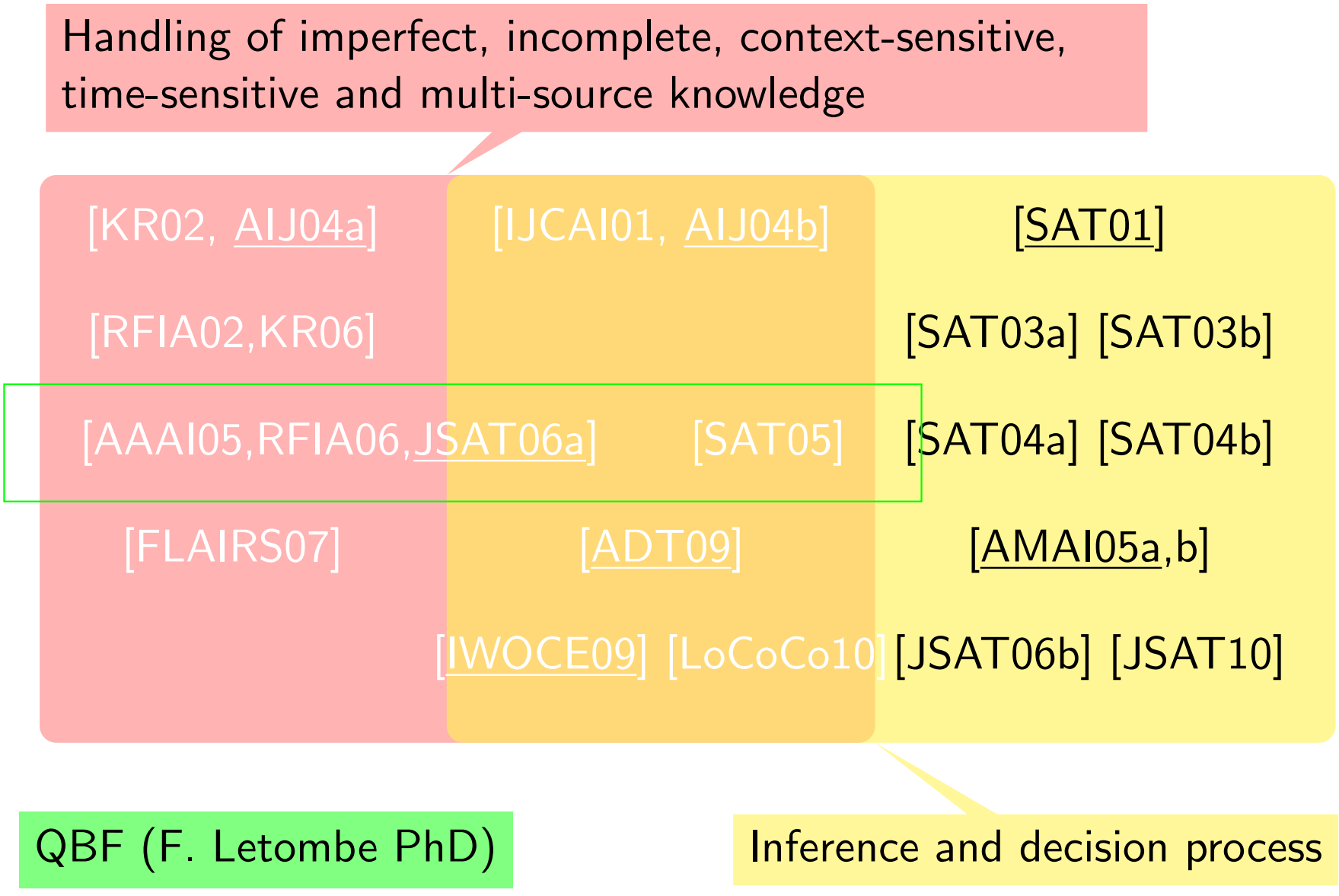
Summary of publications according to CRIL structure



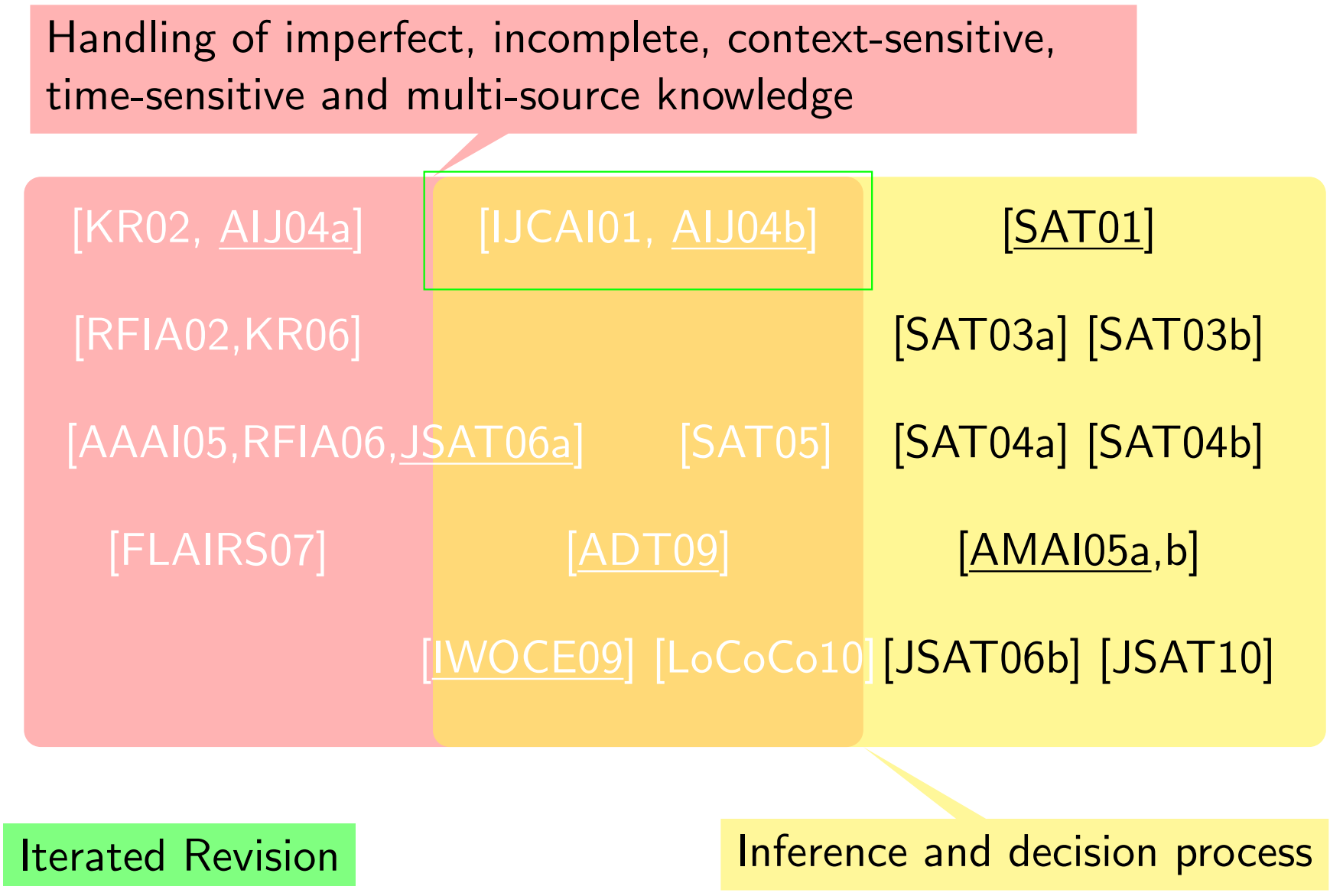
Summary of publications according to CRIL structure



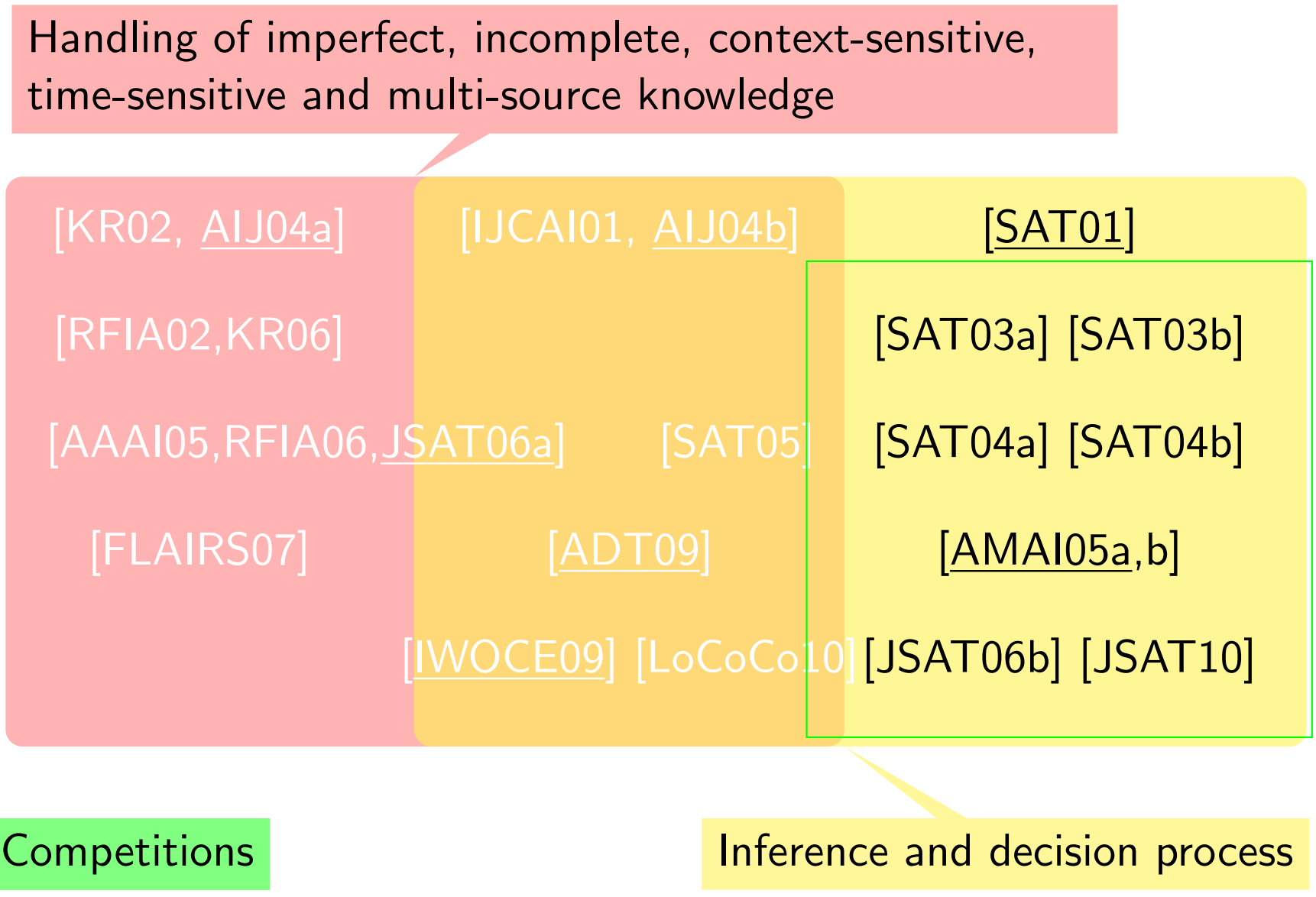
Summary of publications according to CRIL structure



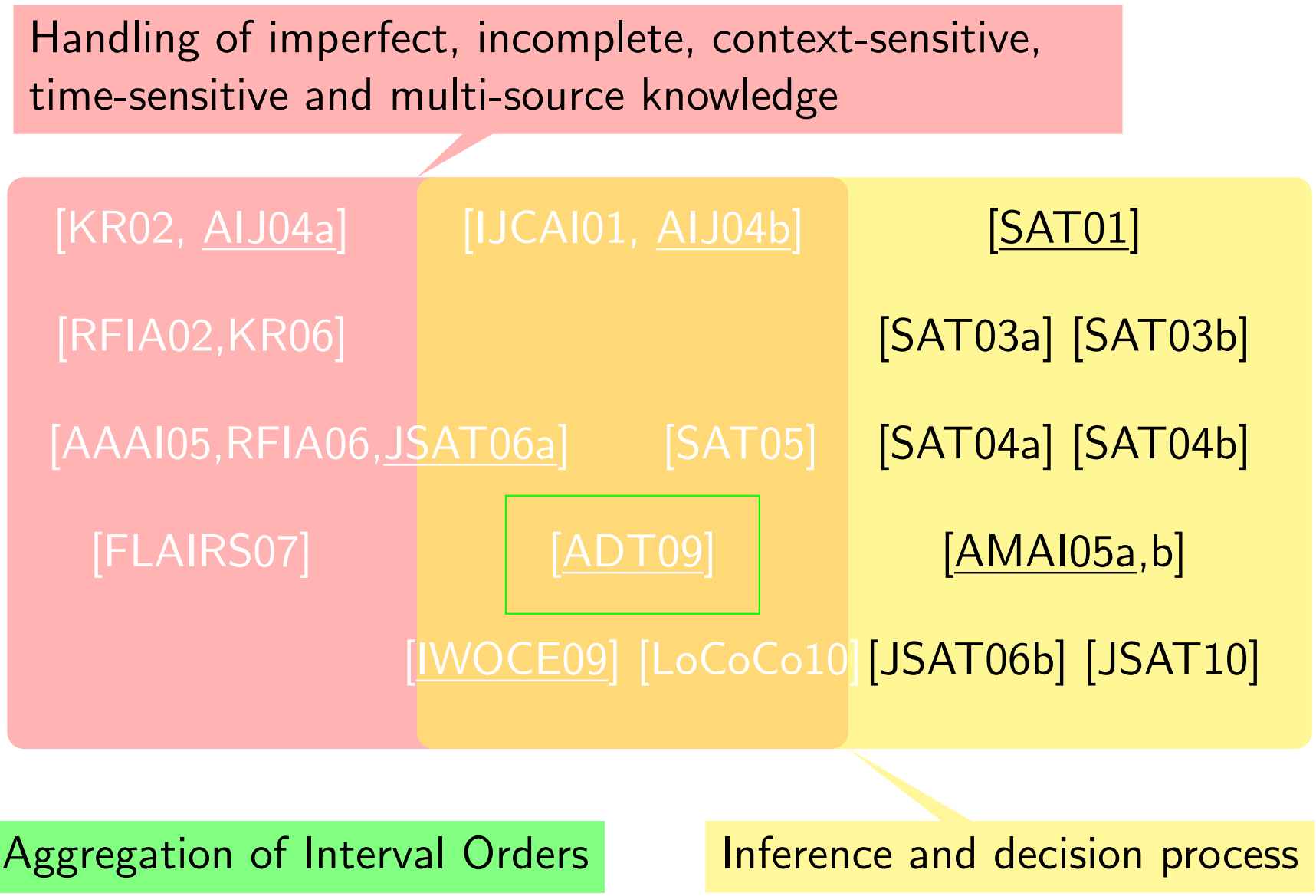
Summary of publications according to CRIL structure



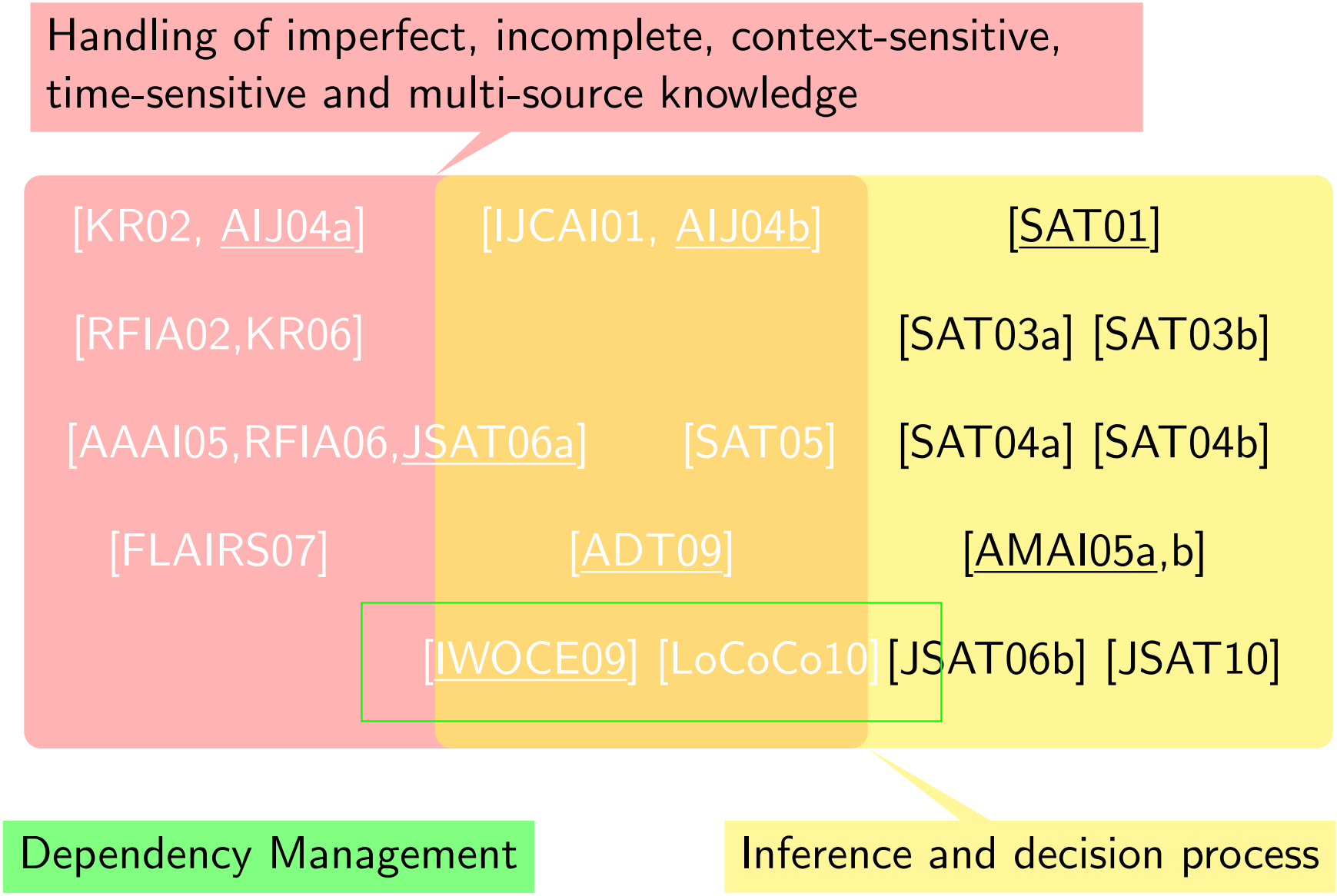
Summary of publications according to CRIL structure



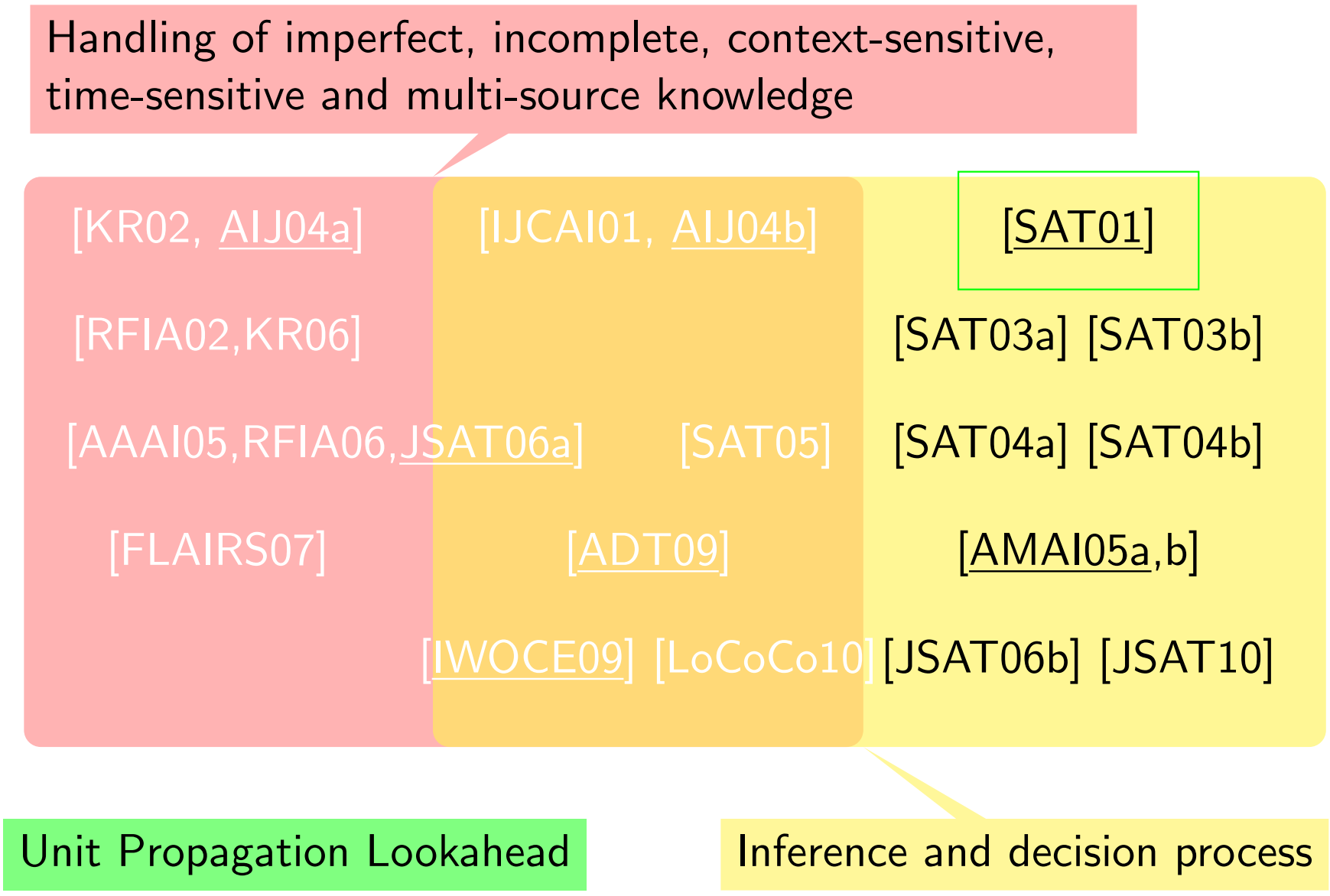
Summary of publications according to CRIL structure



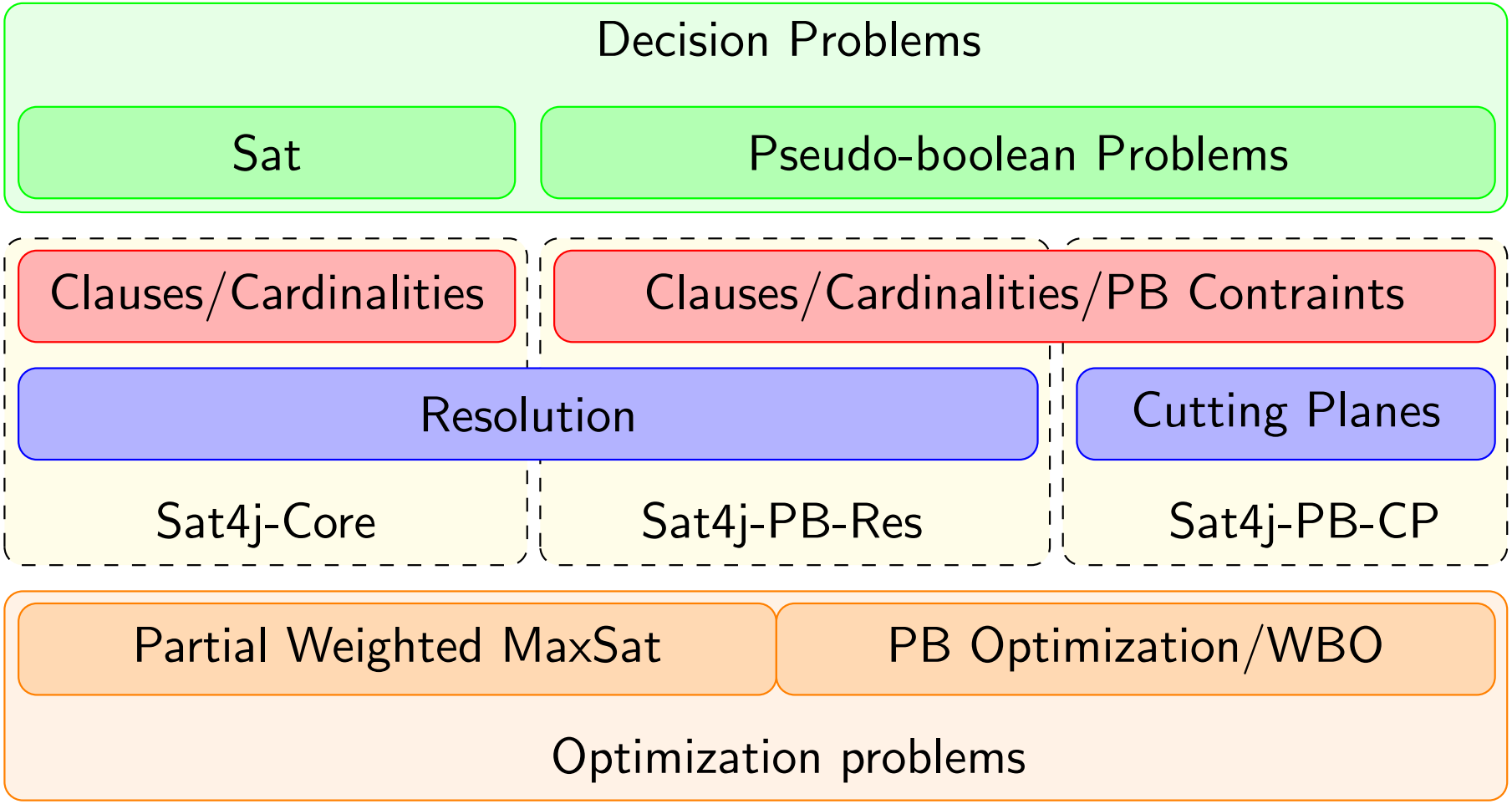
Summary of publications according to CRIL structure



Summary of publications according to CRIL structure



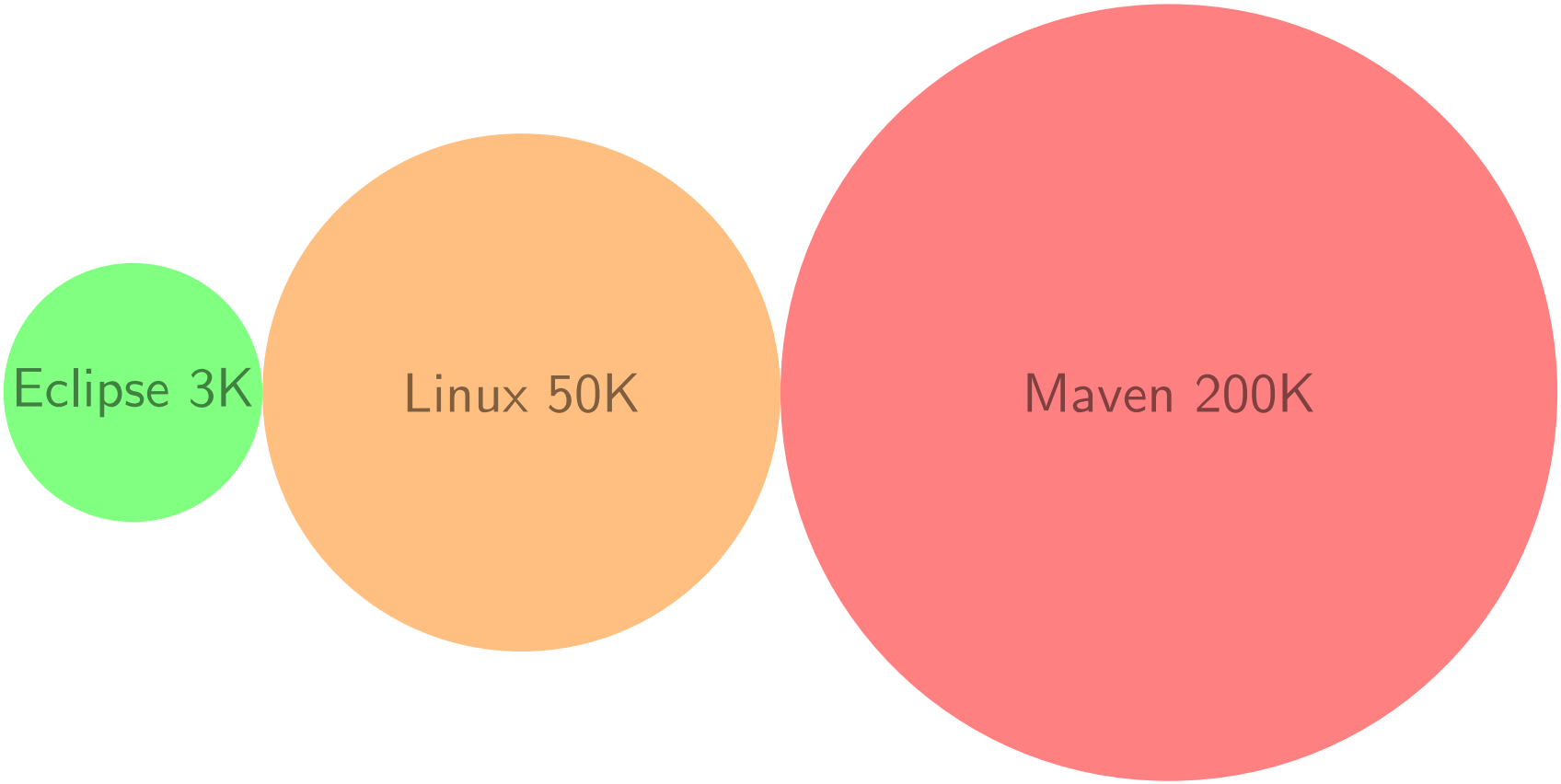
Summary of SAT4J functionalities



Future research directions

- Multi-criteria optimization problems
- Unified modeling language for extended propositional logic
- Efficient implementation of QCL inference
- Cooperation of solvers in multi-core settings
- Continue Sat4j development for PBO (SMT ?)
- Tech-A-Way prototype for configuring visits at Le Louvre Lens Museum

Scaling the dependency problem in an interactive setting



Thanks for your attention

Thanks to my co-authors during those past 10 years : Anne, Armando, Edward, Florian, Gerhard, Gilles, Hélène, Inês, Jérôme, João, Josep, Karima, Laurent, Mary-Anne, Massimo, Meltem, Olivier, Pascal, Paul, Pierre, Salem, Souhila, Sylvie.

Questions ?