Lazy clause exchange policy for parallel SAT solvers

Gilles Audemard
CRIL, Lens, France

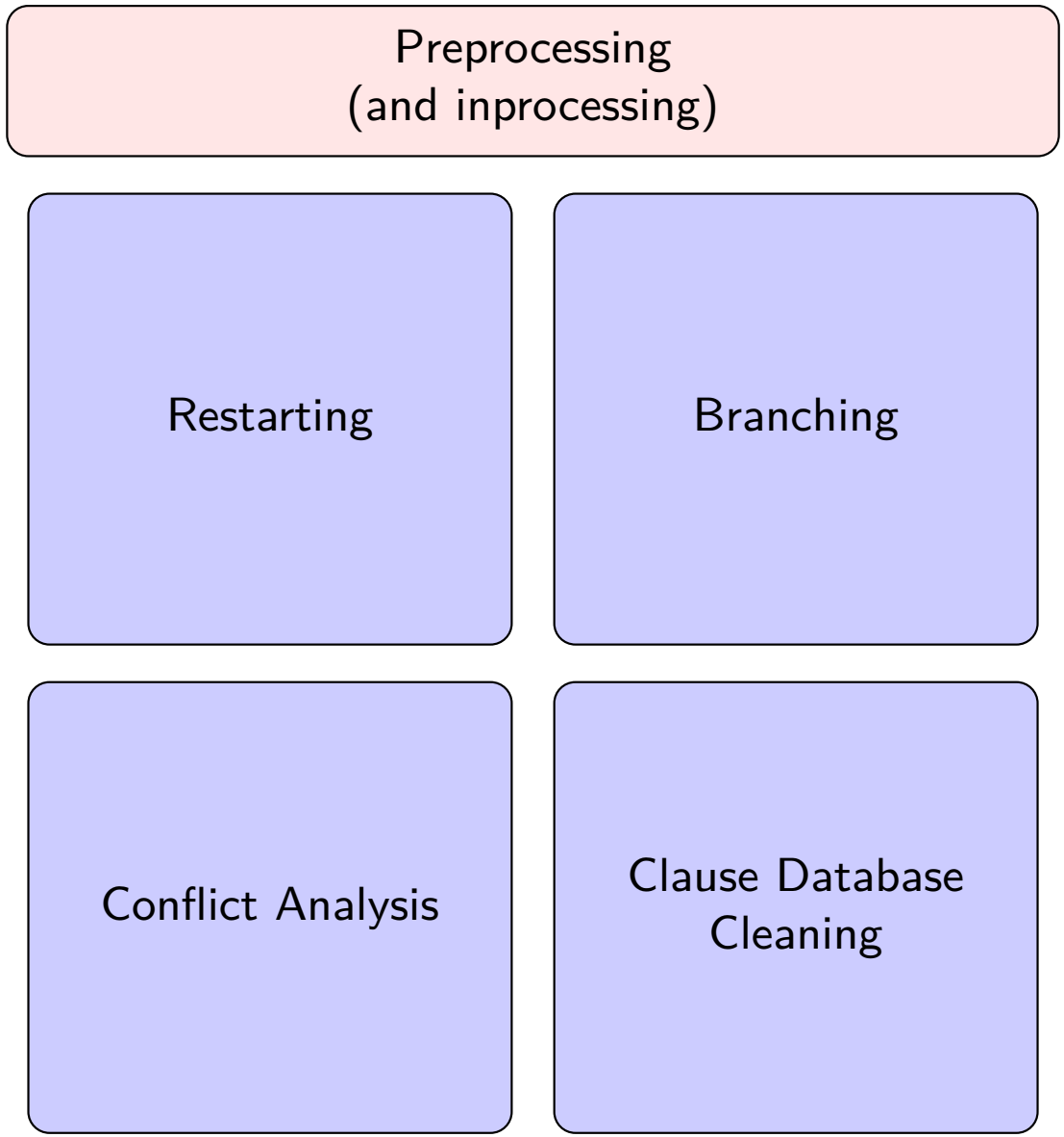
Laurent Simon
Labri, Bordeaux, France

ANR "Investments for the future"
CPU (ANR-10-IDEX-03-02)

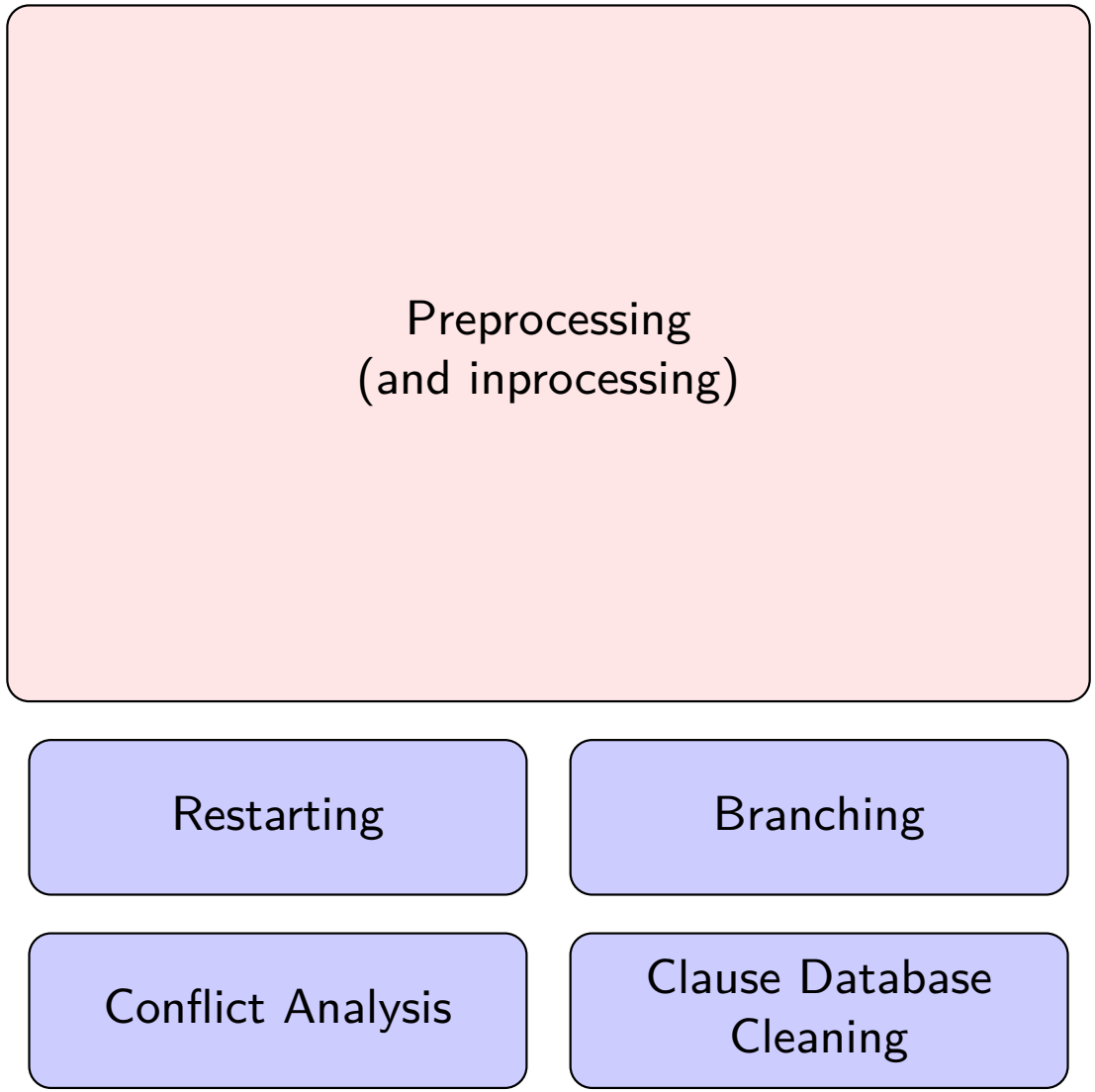
Today's Itinerary

Today's Itinerary

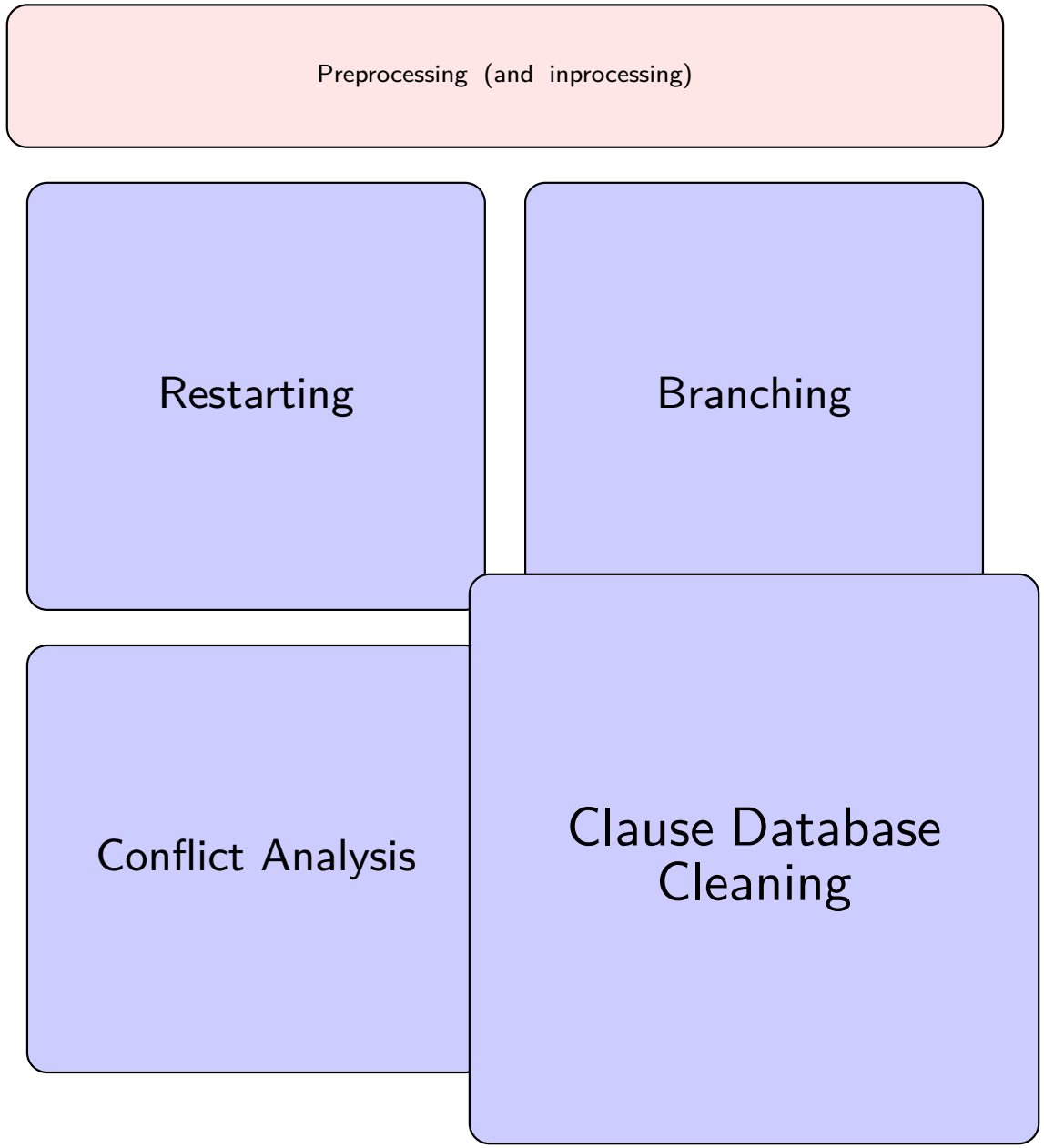
Ingredients of an efficient SAT solver



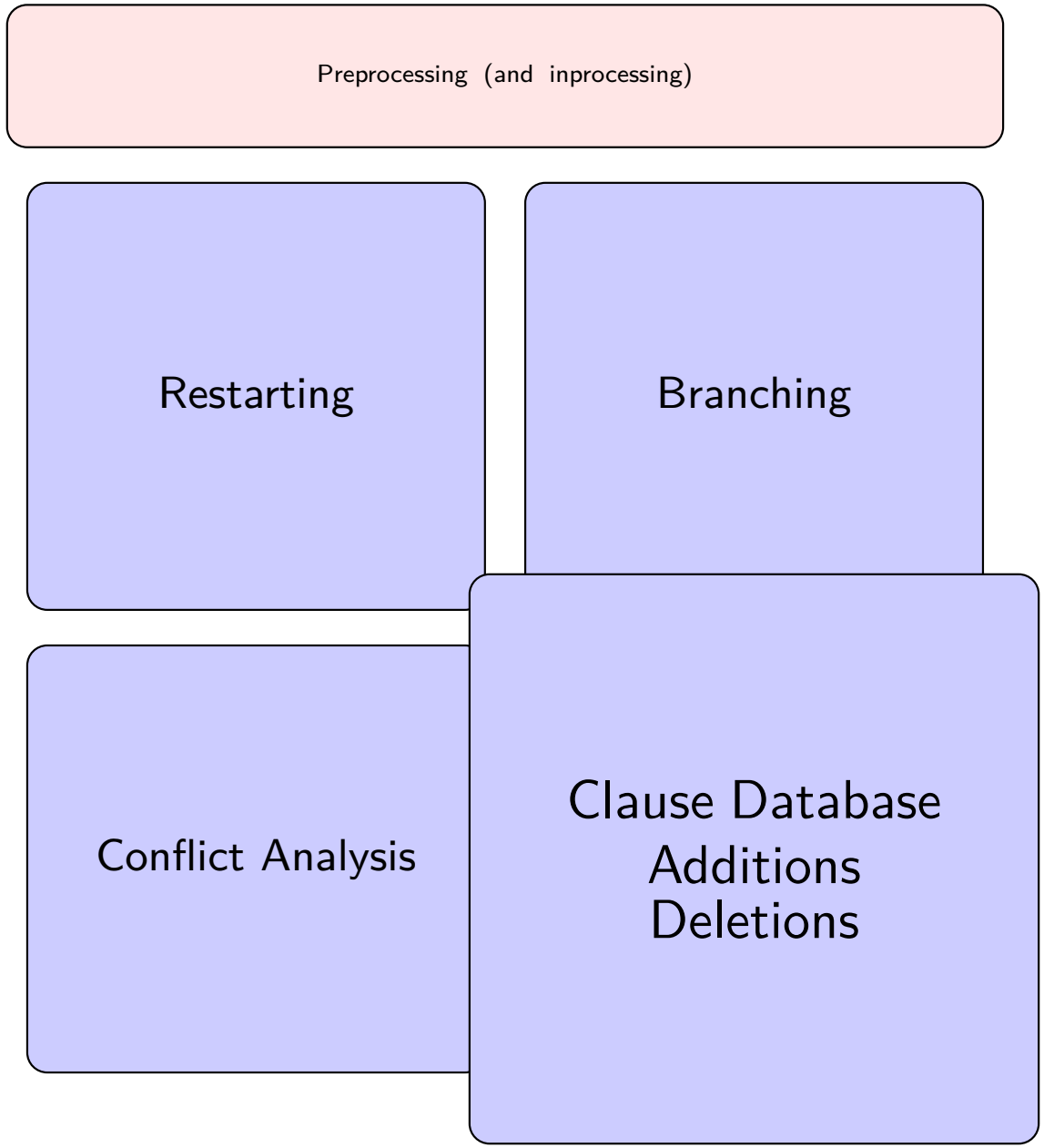
Ingredients of an efficient SAT solver



Ingredients of an efficient SAT solver



Ingredients of an efficient SAT solver



Today's Itinerary

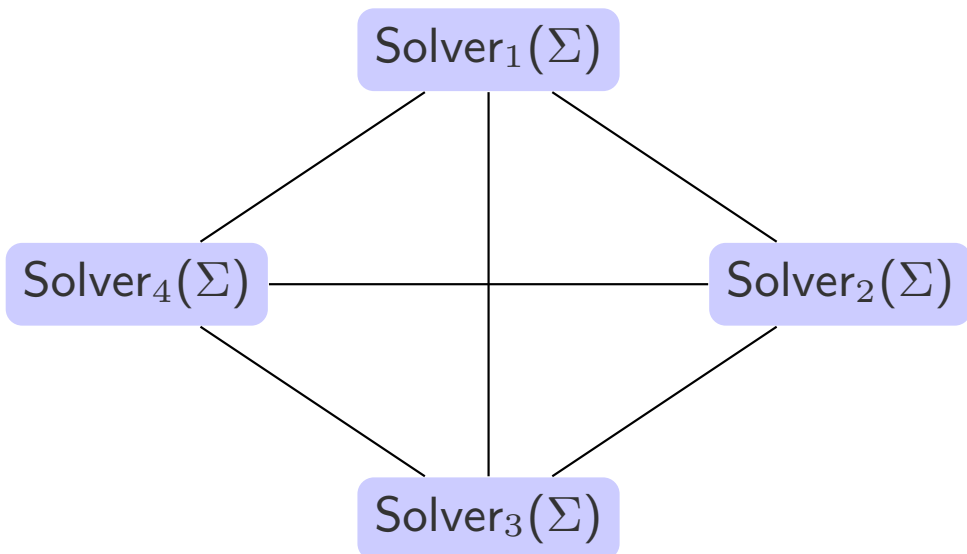
Many Cores

- Multi-core architecture, cloud $\Rightarrow$ design of parallel solvers
- Different strategies
  - Divide and conquer

    ➔ Explicitly splits search w.r.t. assignments
  - Portfolio algorithms
    - Each thread: its own solver and the whole formula

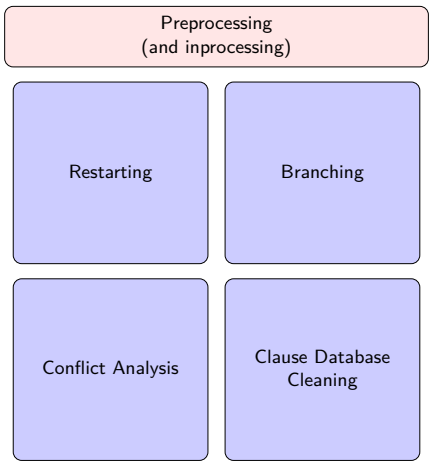
      ➔ Rely on orthogonal searches
  - Parallel solver
    - communication: learnt clauses

      ➔ All threads working on the same proof



Many Cores, Many more problems

Sharing clauses in parallel has many drawbacks



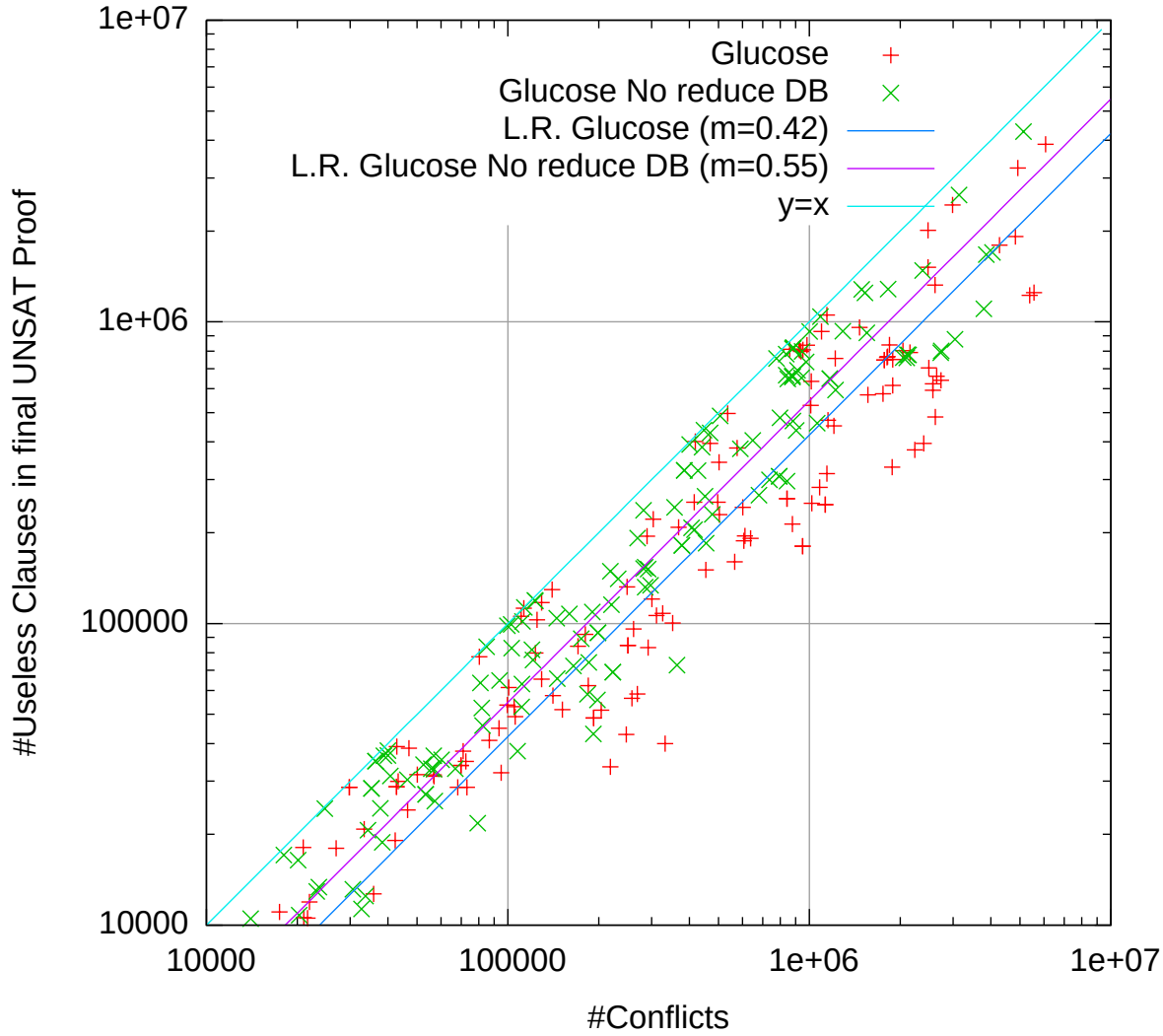
- Imported clauses can be bad (noise, wrong way, . . .)
- Imported clauses can be subsumed / useless
- Imported clauses can dominate learnt clauses
- Each thread has to manage many more clauses
- Many side effects on all core components

Currently: Clauses are sent as soon as they are learnt, plus:

- MANYSAT 1.0: size ≤ 8
- MANYSAT 1.1: dynamically adjust the threshold
- PLINGELING: size ≤ 30 and LBD ≤ 8
- PENELOPE: LBD ≤ 8 (PSM allows more clauses exchanges)

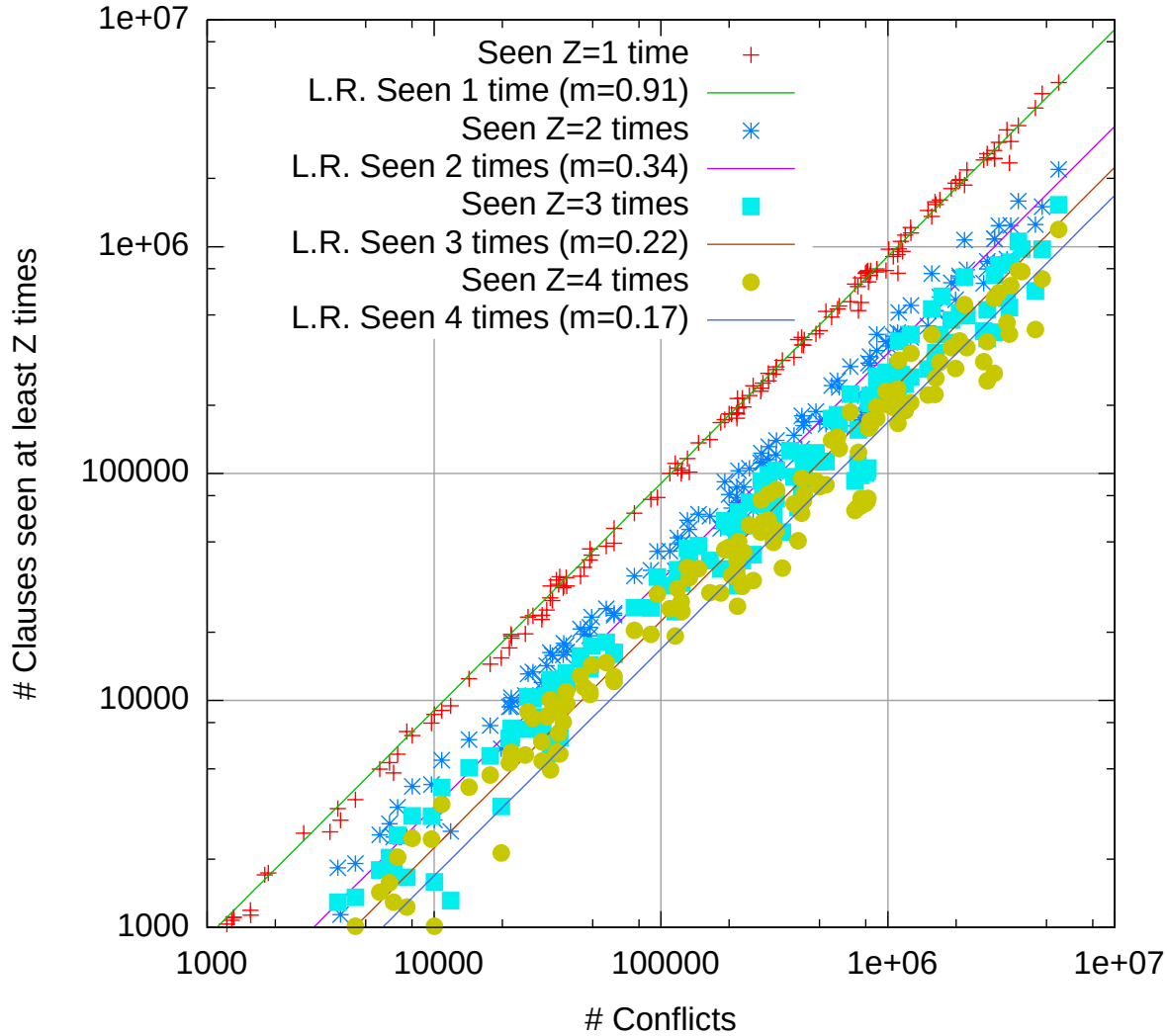
Today's Itinerary

Many *useless clauses* even in single engine solvers



- x-axis : Number of conflicts
- y-axis : Useless clauses in final proof (UNSAT formulas)

How many times clauses are seen in conflicts



- x-axis: Number of conflicts
- y-axis: Number of clauses seen at least Z times

Conclusions from the experiments

A lot of useless clauses even in a single engine!

➔ In parallel, this situation will be even worse!

Why sending a clause that is even not locally interesting?

We will consider clauses seen 2-times (only 34% of learnt clauses)

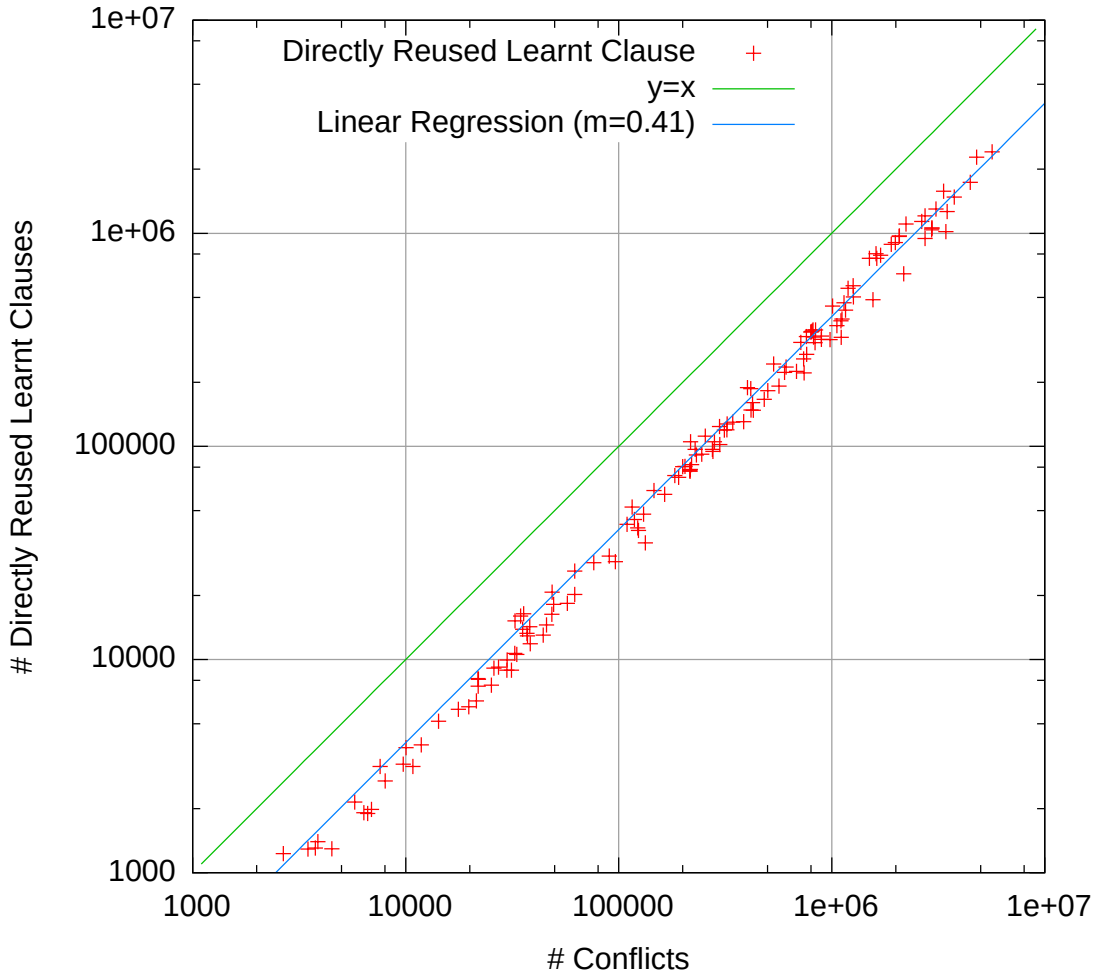
➔ Already filter out the majority of clauses

How to efficiently detect them?

Is there a window of recent clauses to check?

➔ Can we check only recent learnt clauses (and save time)?

Learnt and Directly Reused Clauses



- x-axis: Number of imported clauses (sum over 8 threads)
- y-axis: Number of promoted clauses (sum over 8 threads)

91% of clauses seen at least 1 time. Only 41% are immediately seen.

Lazy Exportation of Clauses

Clauses are sent during conflict analysis

We only export clauses when seen 2 times in conflict analysis and:

- Clauses with LBD $\leq$ median(LBD) and size $\leq$ average(SIZE)
  - Limits updated at each clause database cleaning

Unary clauses and very glue clauses are immediately sent

Lazy because we wait to have a good chance of local interest before considering sending it

Lazy Importation of Clauses

Problem of clauses importations:

- can destroy the current search effort
- clauses can be redundant
- many clauses to manage (performance impact)

How to be sure an imported clause is interesting before considering it?

Idea: put it in probation

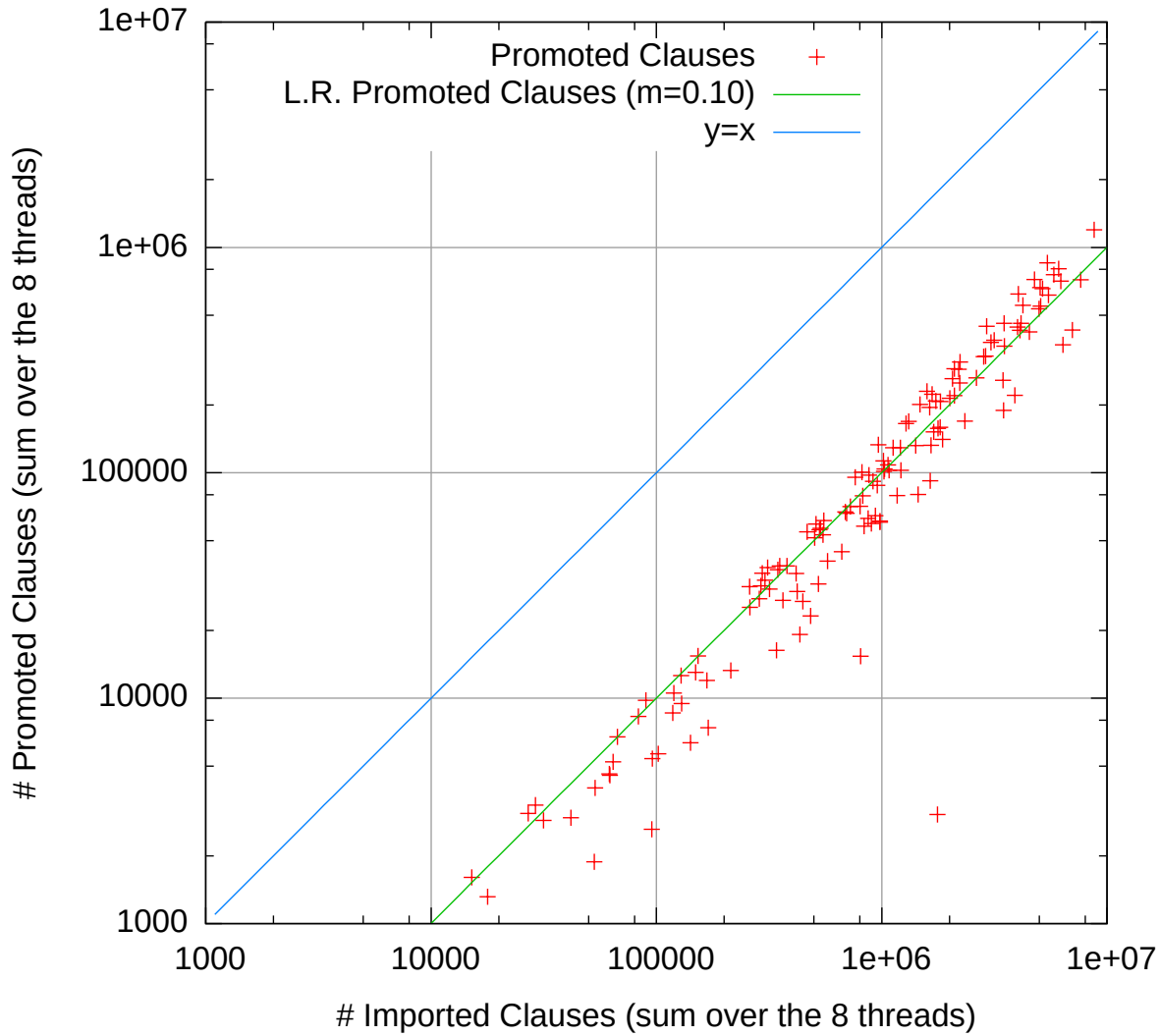
- Imported clauses are put in a 1-Watched scheme
- Will be promoted to a 2-Watched scheme only if found empty

Other Advantages

- Less efforts for propagations
- Can be seen as a dynamic Freezing/Reactivating strategy
- Clauses are imported with a local (and correct) LBD value

Today's Itinerary

How many promotions?



- x-axis: Number of imported clauses (sum over 8 threads)
- y-axis: Number of promoted clauses (sum over 8 threads)

Implementation details

Solvers have many restarts

➔ Import clauses at decision level 0 only

Imported clauses (1-Watched) database have its own deletion strategy

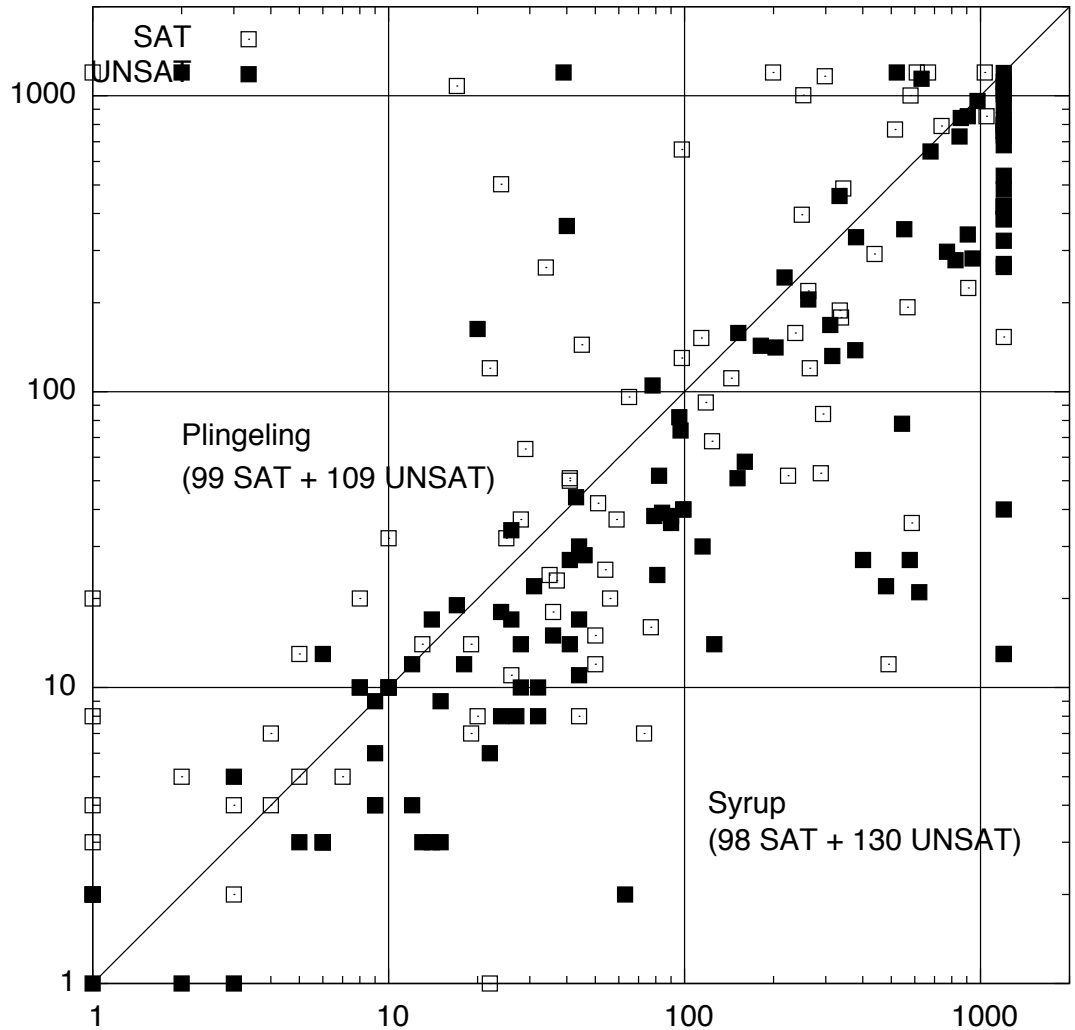
➔ Limits the impact of importation on the solver

(Gory) Details of Glucose-Syrup in the contest:

- One solver is created, then cloned
- Number of solvers adjusted to fit into 40% of allowed memory
- When approaching the max memory allowed, switch to *Panic Mode*: Almost Constant Memory consumption mode
- No tuning of cores strategies. Only the "French's Flair".

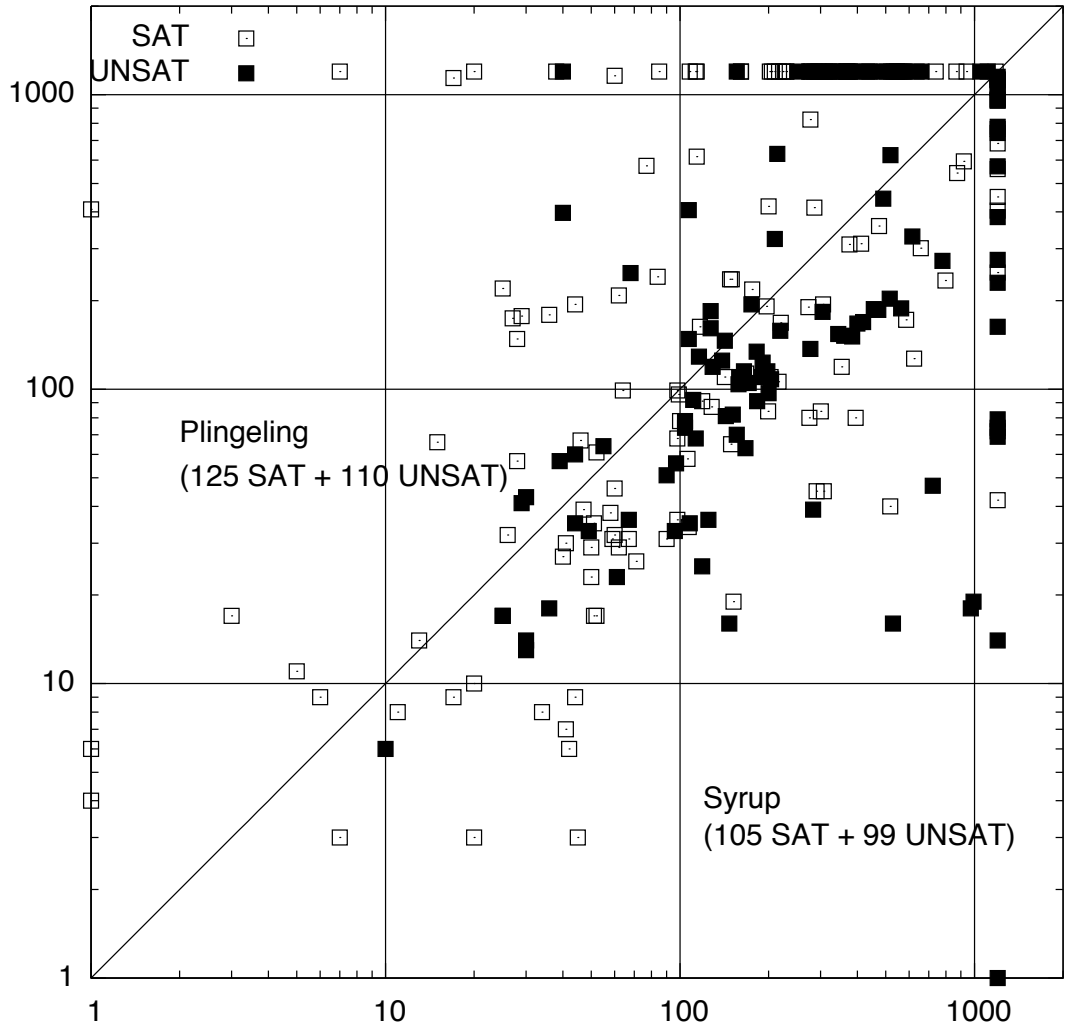
➔ Idea: as few parameters changes as possible

Syrup versus Plingeling on SAT11 benchmarks



- Syrup: 228 instances solved - 98 SAT / 130 UNSAT !!
- Plingeling: 208 instances solved - 99 SAT / 109 UNSAT

Syrup versus Plingeling on SAT13 benchmarks



- Syrup : 204 instances solved - 105 SAT / 99 UNSAT
- Plingeling : 235 instances solved - 125 SAT / 110 UNSAT !!

Today's Itinerary

Conclusion

A simple, efficient, more semantic way to exchange clauses between cores

Parallel Glucose seems very efficient on non crypto problems (like Glucose)

Special thanks to:

- Marijn Heule
  - For helping us debugging the unstable competition version
- Daniel le Berre
  - For fruitful discussions at the Banff Workshop

Please wait to download a stable-cleaned-improved version of glucose syrup