Prioritised Unit Propagation by Partitioning the Watch Lists

Benjamin Kaiser
Robert Clausecker
Michael Mavroskoufis

Pragmatics of SAT
Workshop 2023
04.07.2023

1) Introduction
2) Priority Propagation
3) Heuristics for Prioritisation
4) Experimental Results
5) Conclusion

Introduction

Boolean Satisfiability Problem (SAT)

- **Boolean Formula** given in **Conjunctive Normal Form**:

$$F = \bigwedge_{i=1}^{m} \left(\bigvee_{j=1}^{k_i} l_{ij} \right) = ((l_{11} \vee \ldots \vee l_{1k_1}) \wedge \ldots \wedge (l_{m1} \vee \ldots \vee l_{mk_m}))$$

- Disjunctions $\bigvee_{j=1}^{k_i} l_{ij}$ are called **clauses**

- Clauses consist of **literals** $l_{ij} \in \{x_1, \neg x_1, x_2, \neg x_2, \ldots, x_n, \neg x_n\}$

  for variables $x_1, x_2, \ldots, x_n$

- Variables can be assigned $\{\textbf{true}, \textbf{false}\}$

SAT Solving

- to solve a satisfiable problem **(SAT)**:
  - Assign variables with each clause having a true literal
  - **NP-hard**

- to solve an unsatisfiable problem **(UNSAT)**:
  - proof that no such assignment exists

State-of-the-Art SAT Solving

- Modern SAT Solvers use

  Conflict Driven Clause Learning (CDCL)

- Basic Idea:
  - Choose a partial assignment of the variables
  - For all clauses check for conflicts and implications
  - Analyse conflicts and learn conflict clauses

Search for Conflicts & Implications

- Accomplished by **(Unit) Propagation**

- basic Idea:
  - all literals in a clause assigned **false** ==> Conflict
  - all but one literal assigned **false** and remaining literal unassigned ==> (**Unit) Implication**

Unit Propagation

- **traditional approaches** for fast unit propagation:
  - **deleting clauses** (inevitable)
  - **avoiding clause look-ups**
    - special data structures
      - e.g. **Two Watched Literals (TWL) scheme**

Priority Propagation (PriPro)

- Our contribution:

  adding prioritisation to unit propagation

Priority Propagation (PriPro)

- Our contribution:

  adding prioritisation to unit propagation

- Our Solver CaDiCaL_PriPro:

  Special Innovation Award at SAT Comp. 2021

Priority Propagation (PriPro)

- Our contribution:

  adding prioritisation to unit propagation

- **basic idea**:
  - select clauses to prioritise
  - only propagate prioritised clauses
  - if no conflict occurred, propagate other clauses, too

Priority Propagation (PriPro)

- Our contribution:

  adding prioritisation to unit propagation

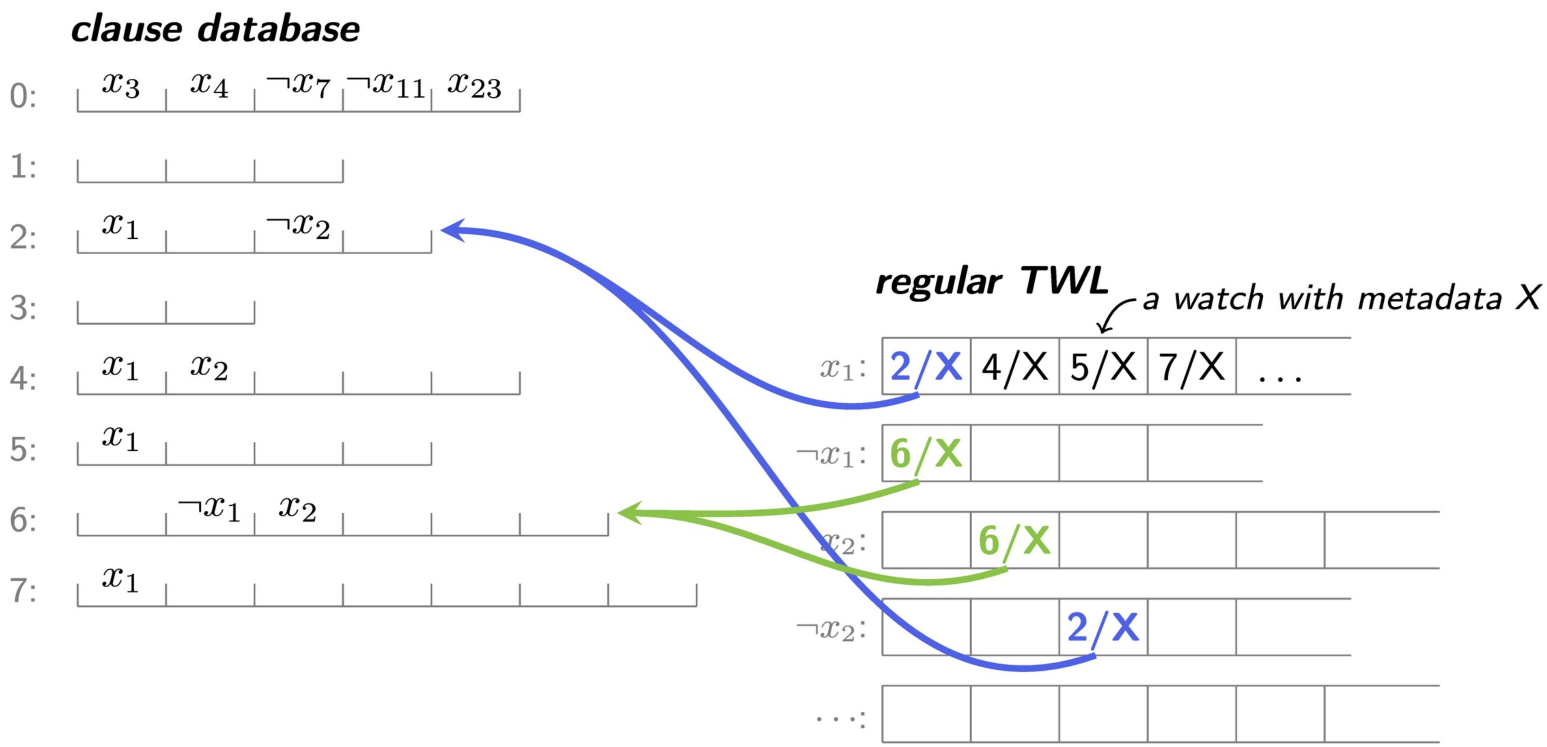
- **basic idea**:
  - select clauses to prioritise
  - only propagate prioritised clauses
  - if no conflict occurred, propagate other clauses, too

- implementation:
  - **additional TWL scheme** for prioritised clauses
  - **adapt unit propagation** to focus on prioritised clauses

Priority Propagation (PriPro)

Two Watched Literals

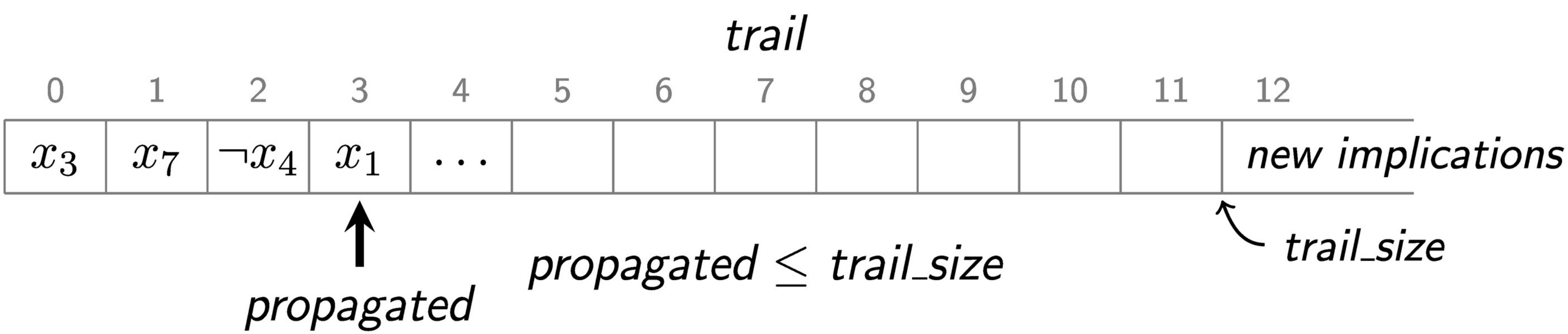
Two Watched Literals



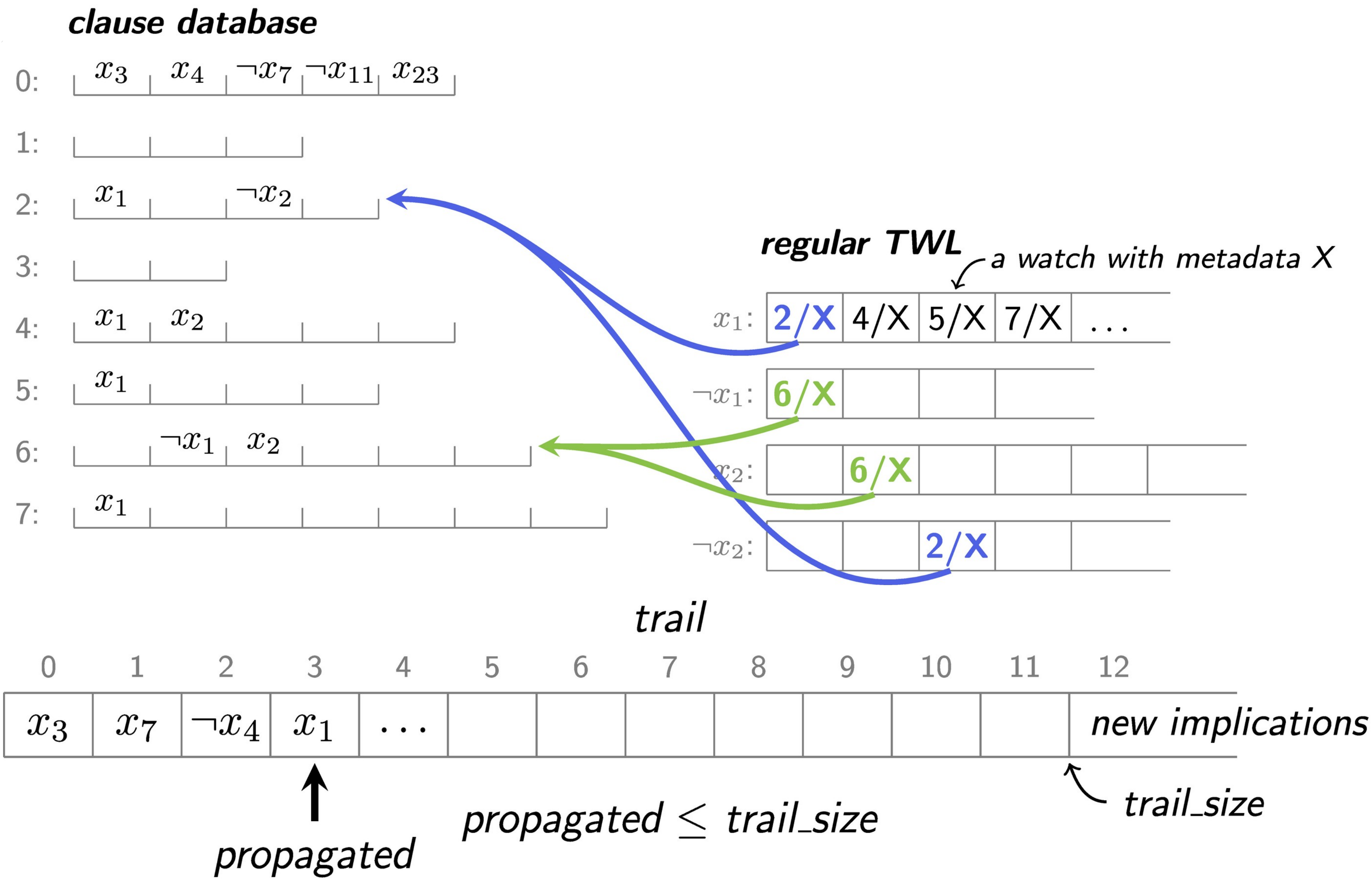
- basic idea:
  - only look clauses up if one of their watched literals is **false**
  - only those clauses can lead to conflict or implication

Trail

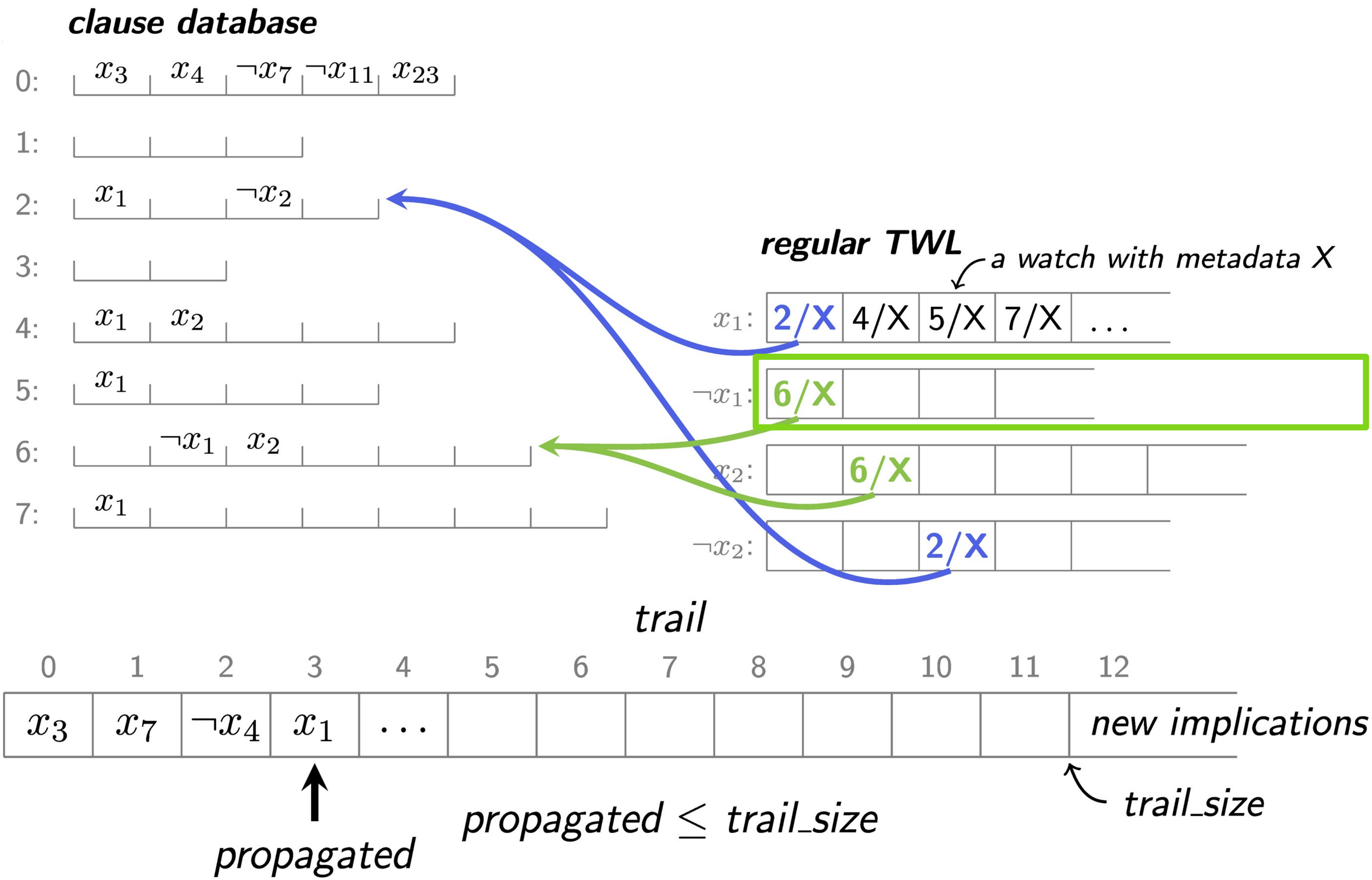
- **trail** indicates which literals have been assigned **true**
- contains literals **in order of assignment**
- (negations of) literals on the trail need to be **‘propagated’**
- **iterate through trail** to propagate literals



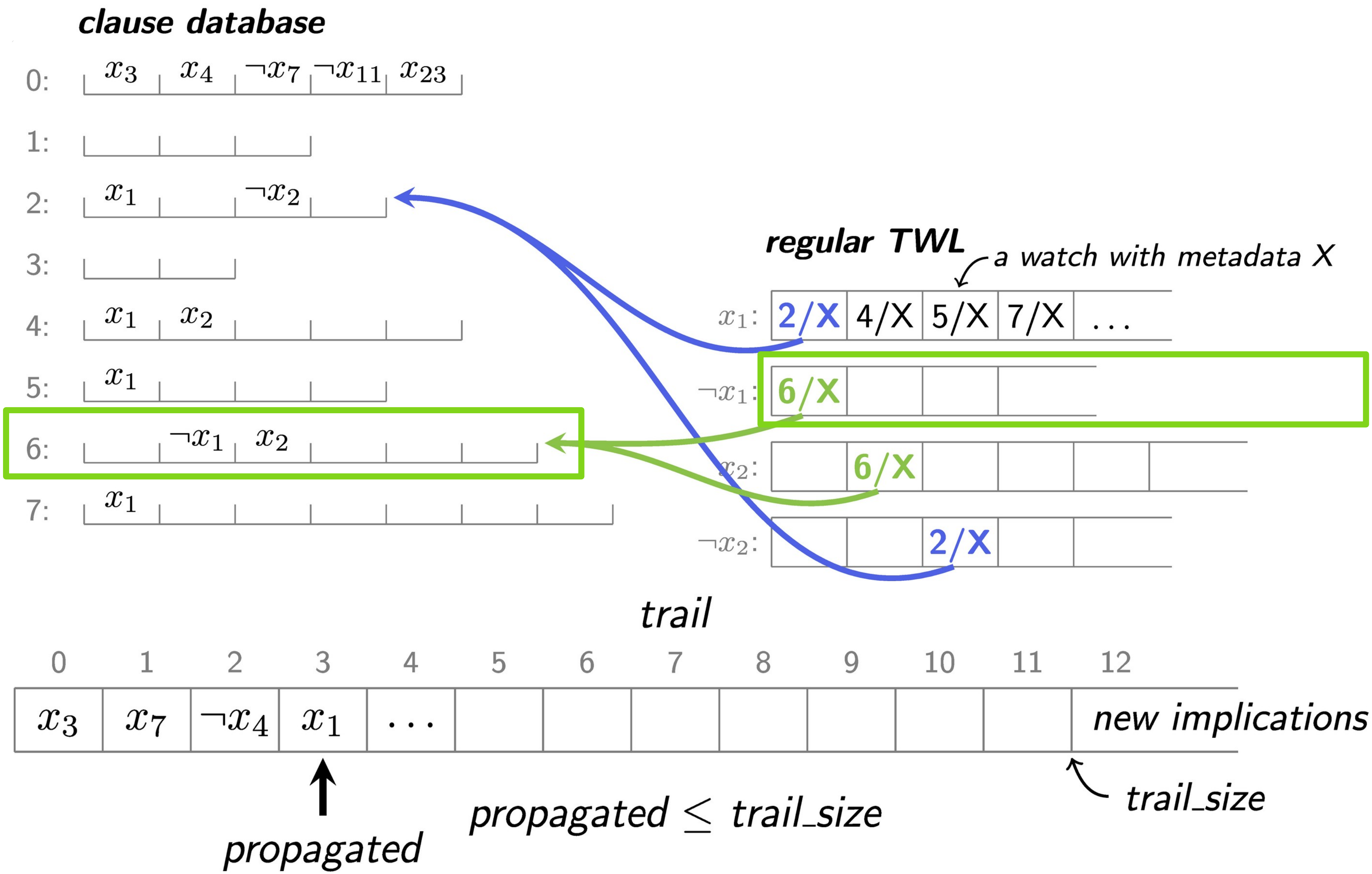
Propagation of Literals



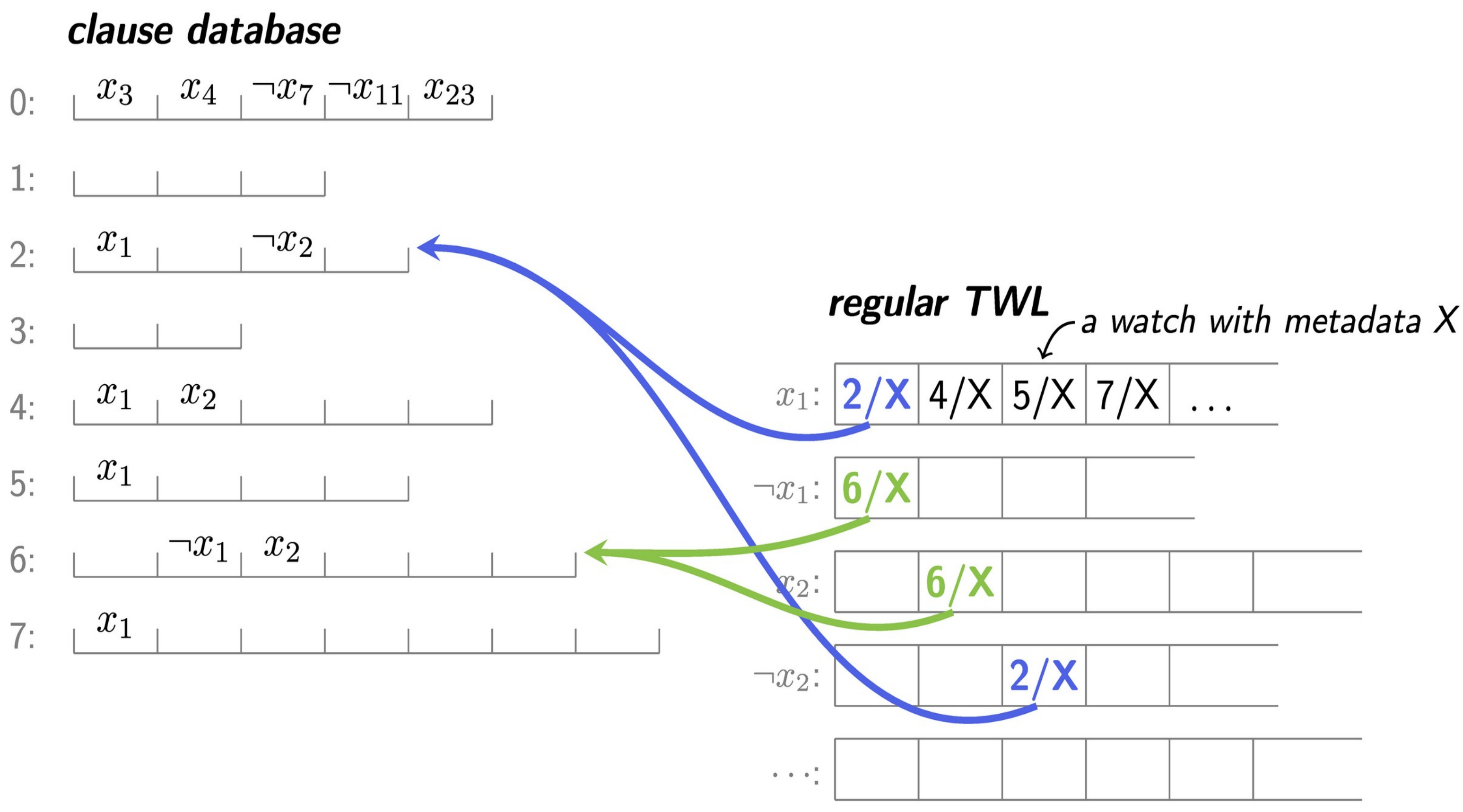
Propagation of Literals



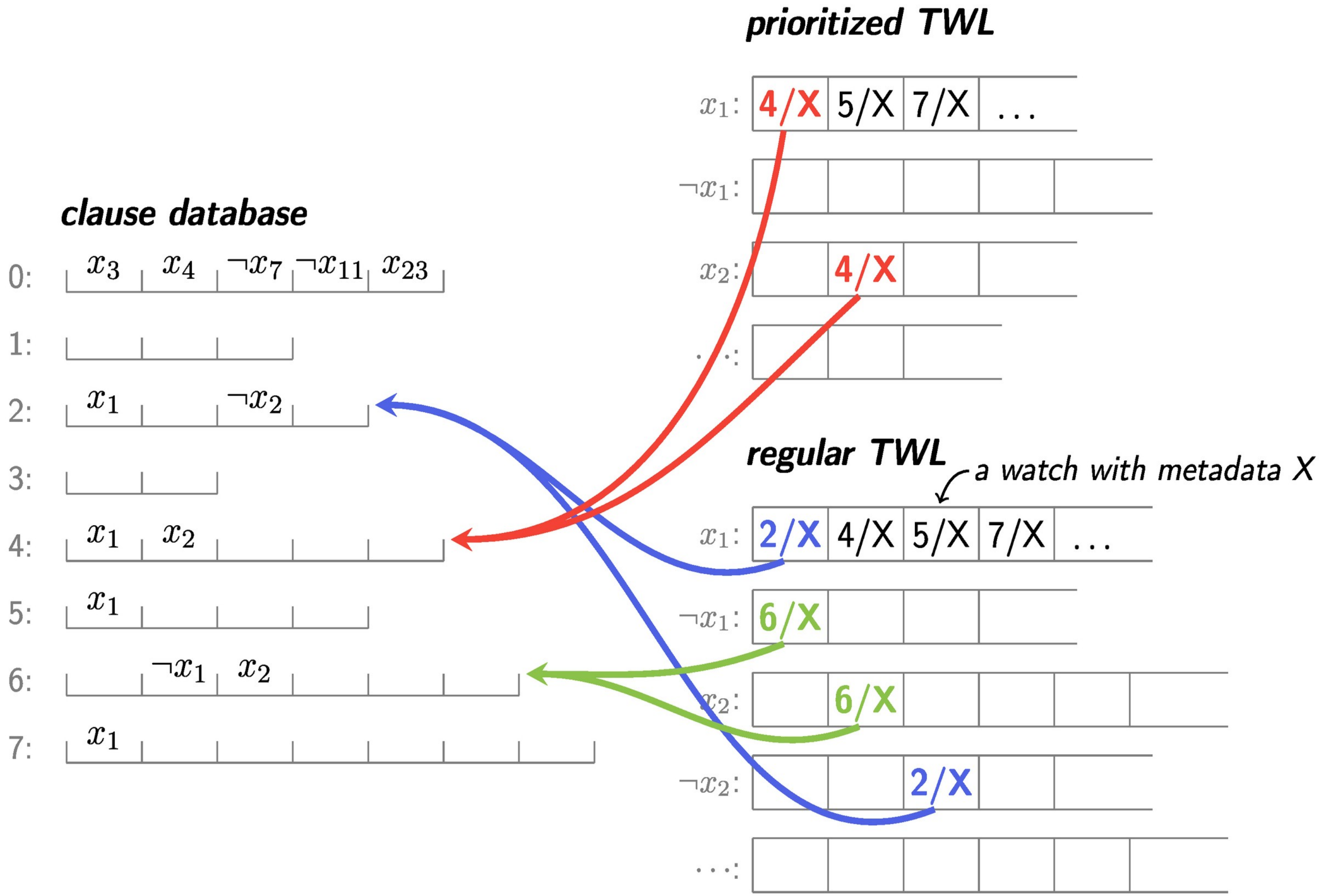
Propagation of Literals



Prioritised TWL scheme



Prioritised TWL scheme



Unit Propagation with PriPro

Unit Propagation with PriPro

Algorithm Unit Propagation

while $\neg conflict \wedge propagated \neq trail\_size$ **do**

  $lit \leftarrow trail[propagated]$

  $propagated \leftarrow propagated + 1$

  $conflict \leftarrow propagate\_watches\_of\ (\neg lit)$

end while

return $conflict$

Unit Propagation with PriPro

Algorithm Unit Propagation

while $\neg conflict \land propagated \neq trail\_size$ **do**

  $lit \leftarrow trail[propagated]$

  $propagated \leftarrow propagated + 1$

  $\boxed{conflict \leftarrow propagate\_watches\_of\ (\neg lit)}$

end while

return $conflict$

Unit Propagation with PriPro

Algorithm Unit Propagation with PriPro

while $\neg conflict \wedge propagated \neq trail\_size$ **do**
  $lit \leftarrow trail[propagated]$
  $propagated \leftarrow propagated + 1$
  $conflict \leftarrow propagate\_prioritised\_watches()$
  if $\neg conflict$ **then**
    $conflict \leftarrow propagate\_regular\_watches\_of(\neg lit)$
  end if
end while
return $conflict$

Unit Propagation with PriPro

Algorithm Unit Propagation with PriPro

while $\neg \mathit{conflict} \wedge \mathit{propagated} \neq \mathit{trail\_size}$ **do**

  $\mathit{lit} \leftarrow \mathit{trail}[\mathit{propagated}]$

  $\mathit{propagated} \leftarrow \mathit{propagated} + 1$

  $\mathit{conflict} \leftarrow \mathit{propagate\_prioritised\_watches}()$

  if $\neg \mathit{conflict}$ **then**

    $\mathit{conflict} \leftarrow \mathit{propagate\_regular\_watches\_of}(\neg \mathit{lit})$

  end if

end while

return $\mathit{conflict}$

of **all** remaining literals

Propagation of Prioritised Watches

- resembles original unit propagation

Propagation of Prioritised Watches

- resembles original unit propagation

- distinguish between
  - current literal w.r.t. regular watches
    - indicated by **propagated**
  - current literal w.r.t. prioritised watches
    - indicated by **pripro_propagated**

Propagation of Prioritised Watches

- resembles original unit propagation

Algorithm The *propagate_prioritised_watches* function

while $\neg conflict \land pripro\_propagated \neq trail\_size$ **do**

$\quad lit \leftarrow trail[pripro\_propagated]$

$\quad pripro\_propagated \leftarrow pripro\_propagated + 1$

$\quad conflict \leftarrow propagate\_prioritised\_watches\_of\ (\neg lit)$

end while

return $conflict$

Relation between Current Literals

Algorithm The *propagate_prioritised_watches* function

while $\neg conflict \land pripro\_propagated \neq trail\_size$ **do**

  $lit \leftarrow trail[pripro\_propagated]$

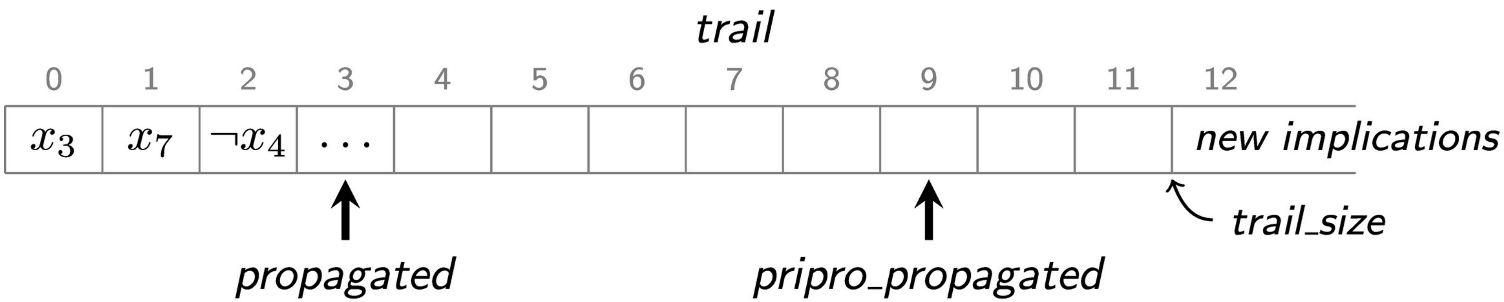
  $pripro\_propagated \leftarrow pripro\_propagated + 1$

  $conflict \leftarrow propagate\_prioritised\_watches\_of\ (\neg lit)$

end while

return *conflict*

Relation between Current Literals



$$\textit{propagated} \leq \textit{pripro\_propagated} \leq \textit{trail\_size}$$

Algorithm The *propagate_prioritised_watches* function

---

while $\neg \textit{conflict} \wedge \textit{pripro\_propagated} \neq \textit{trail\_size}$ **do**

  $\textit{lit} \leftarrow \textit{trail}[\textit{pripro\_propagated}]$

  $\textit{pripro\_propagated} \leftarrow \textit{pripro\_propagated} + 1$

  $\textit{conflict} \leftarrow \textit{propagate\_prioritised\_watches\_of}\ (\neg \textit{lit})$

end while

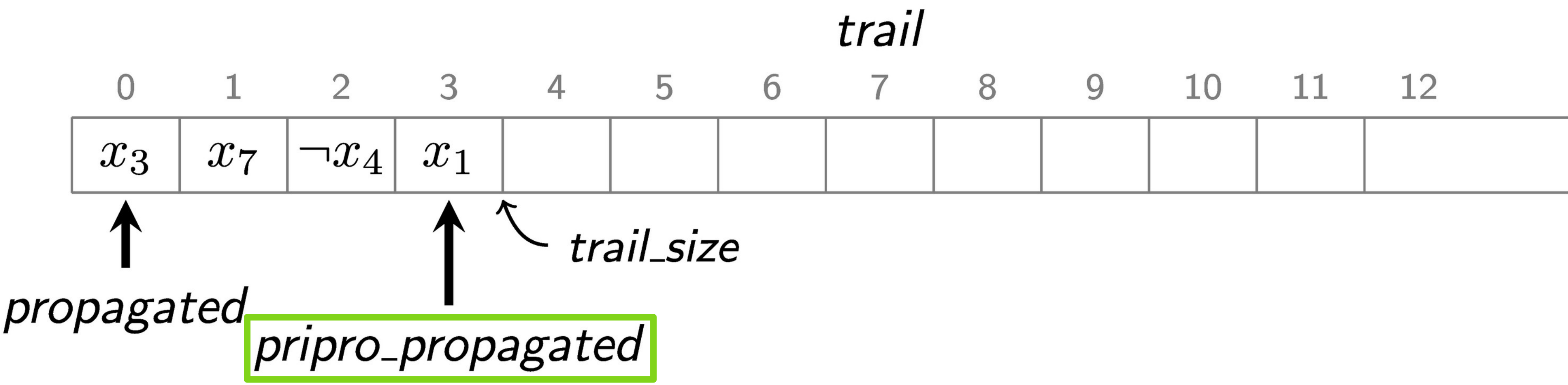
return $\textit{conflict}$

Relation between Current Literals

Relation between Current Literals

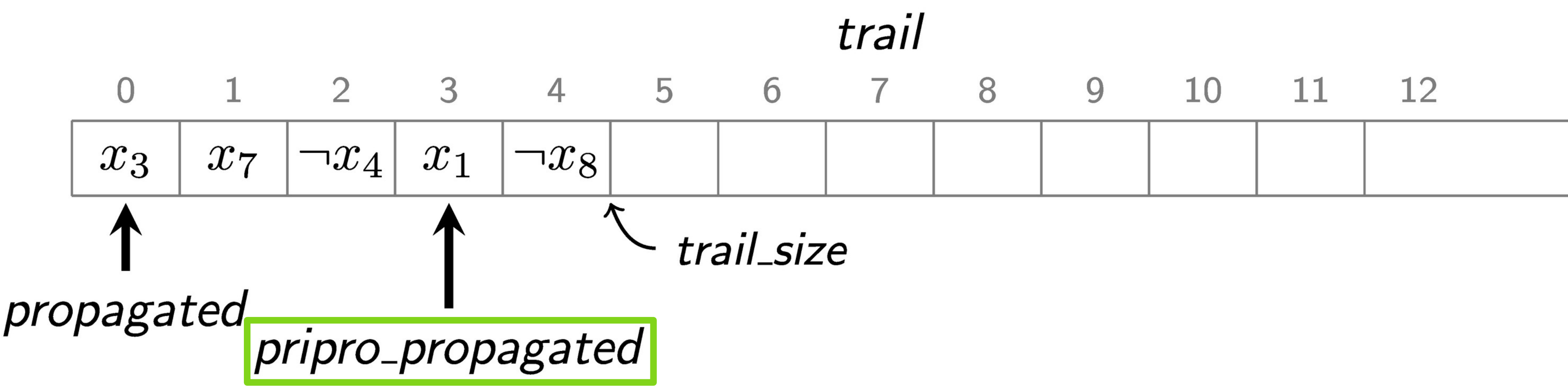
- current literal ≈ next to be propagated

Relation between Current Literals



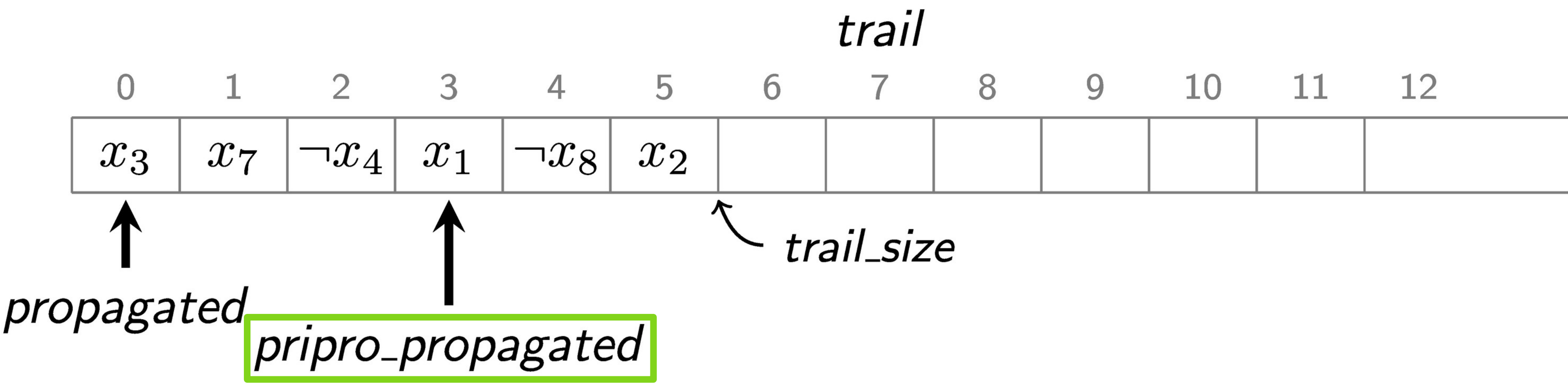
$$\textit{propagated} \leq \textit{pripro\_propagated} \leq \textit{trail\_size}$$

Relation between Current Literals



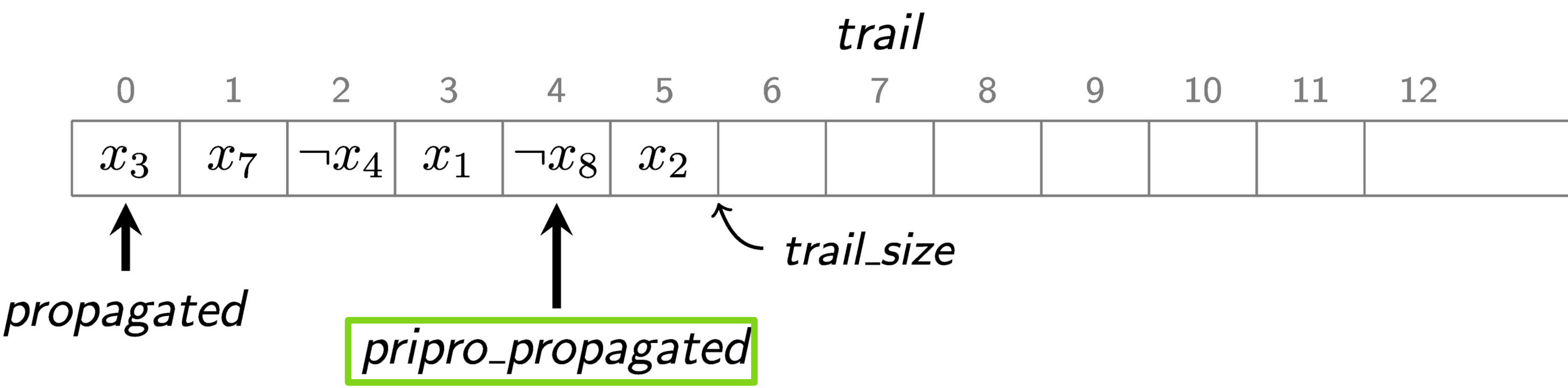
$$\textit{propagated} \le \textit{pripro\_propagated} \le \textit{trail\_size}$$

Relation between Current Literals



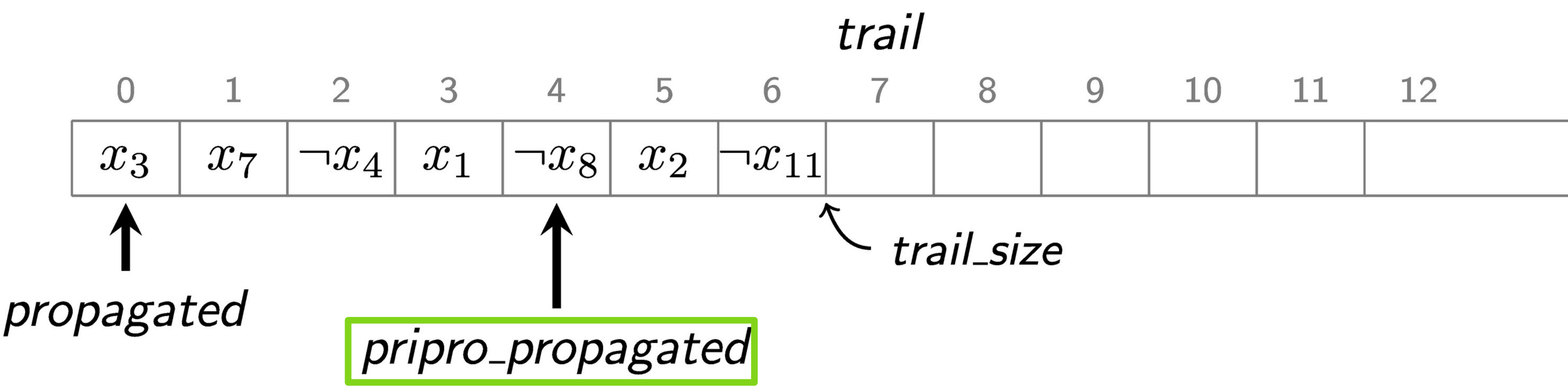
propagated $\leq$ *pripro_propagated* $\leq$ *trail_size*

Relation between Current Literals



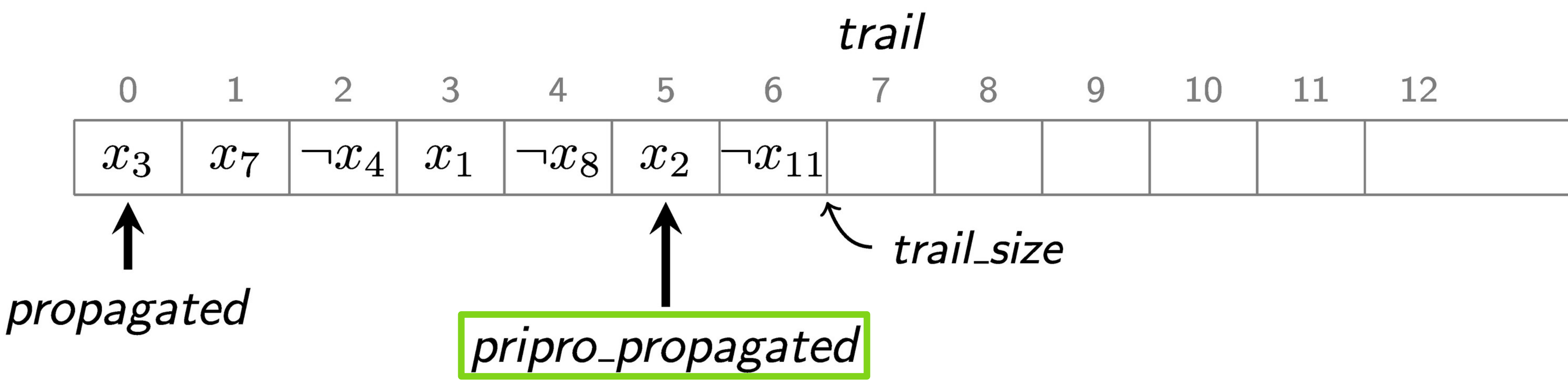
propagated $\leq$ *pripro_propagated* $\leq$ *trail_size*

Relation between Current Literals



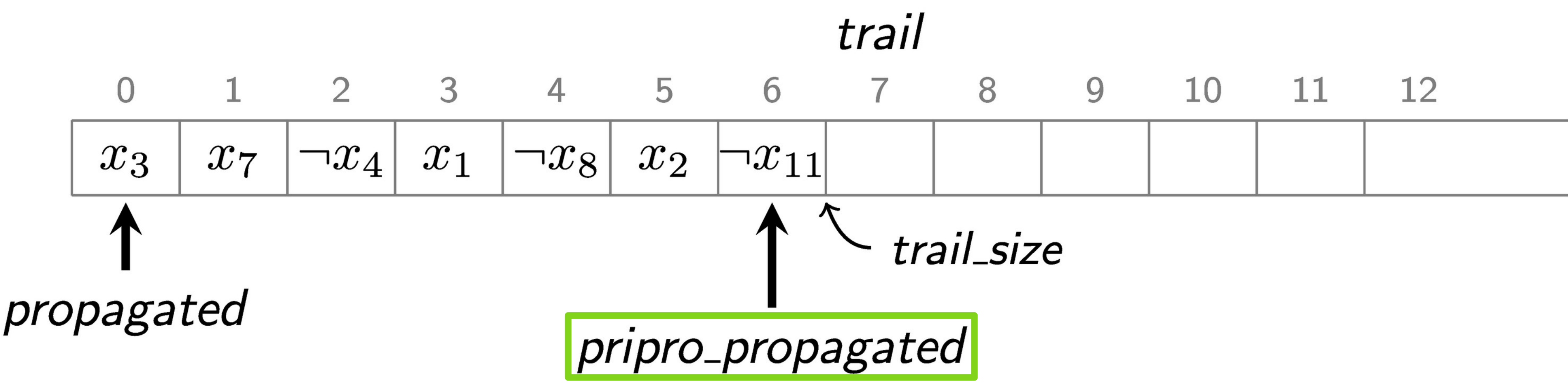
$$\textit{propagated} \leq \textit{pripro\_propagated} \leq \textit{trail\_size}$$

Relation between Current Literals



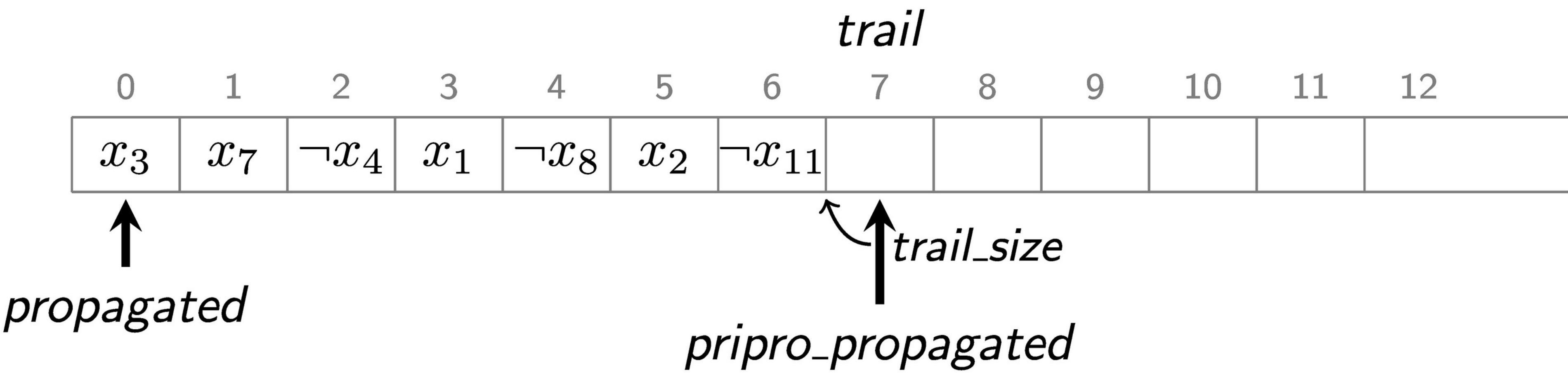
$$\textit{propagated} \leq \textit{pripro\_propagated} \leq \textit{trail\_size}$$

Relation between Current Literals



propagated $\leq$ *pripro_propagated* $\leq$ *trail_size*

Relation between Current Literals



$$\textit{propagated} \leq \textit{pripro\_propagated} \leq \textit{trail\_size}$$

Relation between Current Literals

trail

0	1	2	3	4	5	6	7	8	9	10	11	12
x_3	x_7	$\neg x_4$	x_1	$\neg x_8$	x_2	$\neg x_{11}$						

$\uparrow$ *propagated*

$\uparrow$ *trail_size*

pripro_propagated

$$\textit{propagated} \leq \textit{pripro\_propagated} \leq \textit{trail\_size}$$

Relation between Current Literals

trail

0	1	2	3	4	5	6	7	8	9	10	11	12
x_3	x_7	$\neg x_4$	x_1	$\neg x_8$	x_2	$\neg x_{11}$	$\neg x_{10}$					

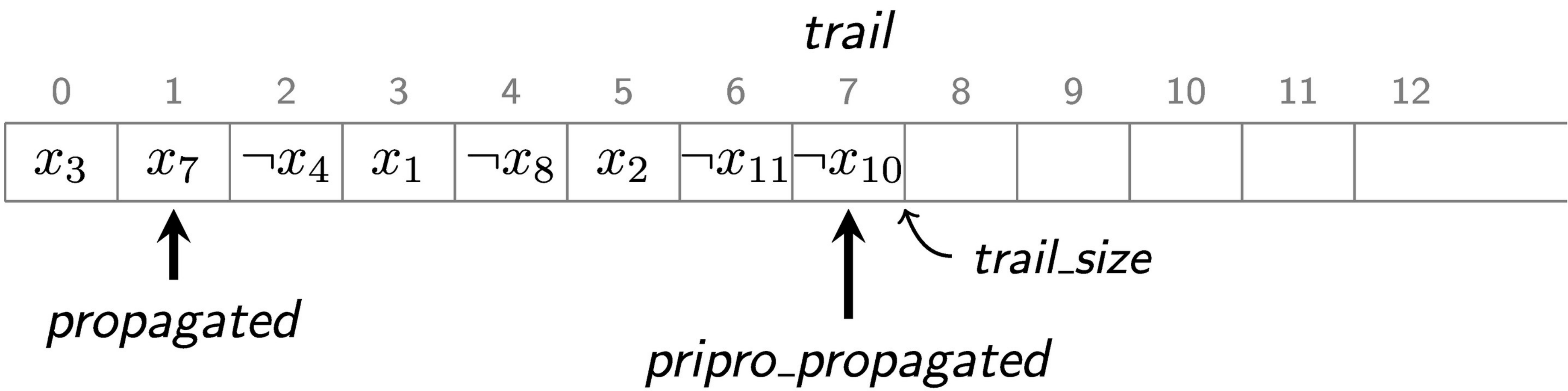
propagated

trail_size

pripro_propagated

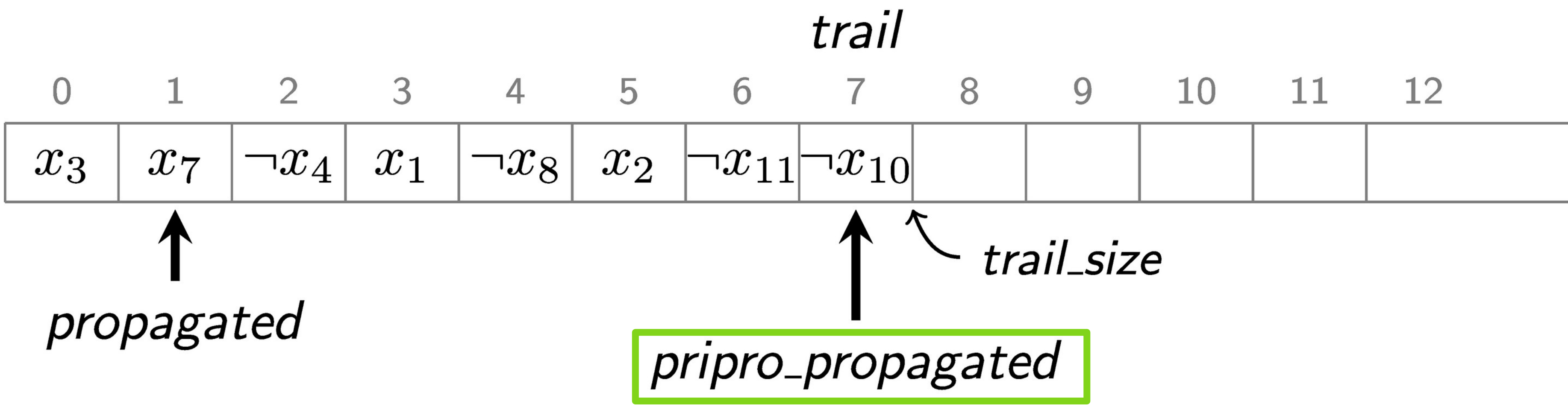
$$\textit{propagated} \leq \textit{pripro\_propagated} \leq \textit{trail\_size}$$

Relation between Current Literals



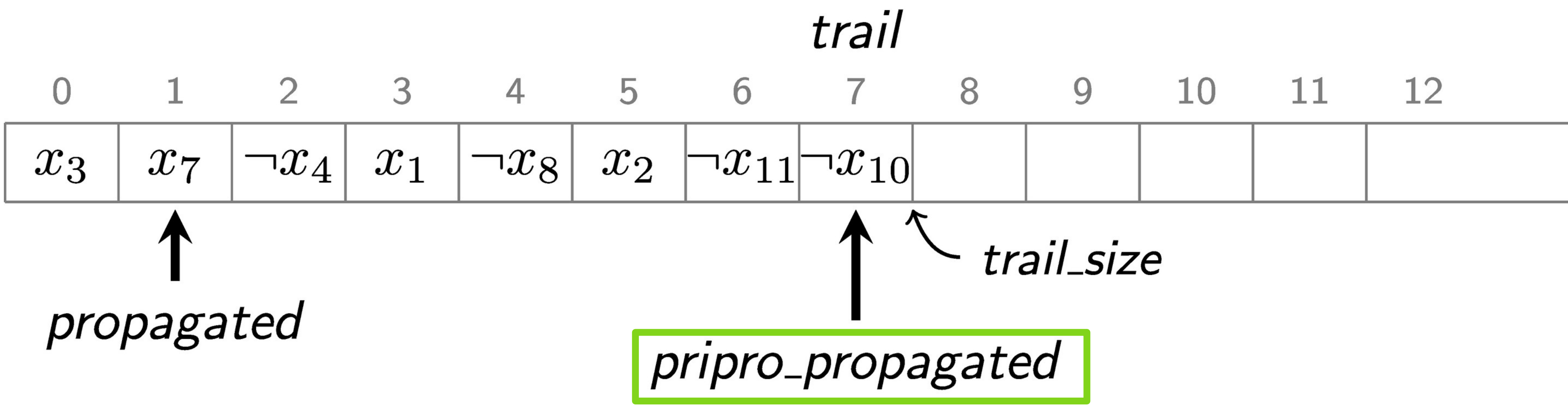
propagated $\leq$ *pripro_propagated* $\leq$ *trail_size*

Relation between Current Literals



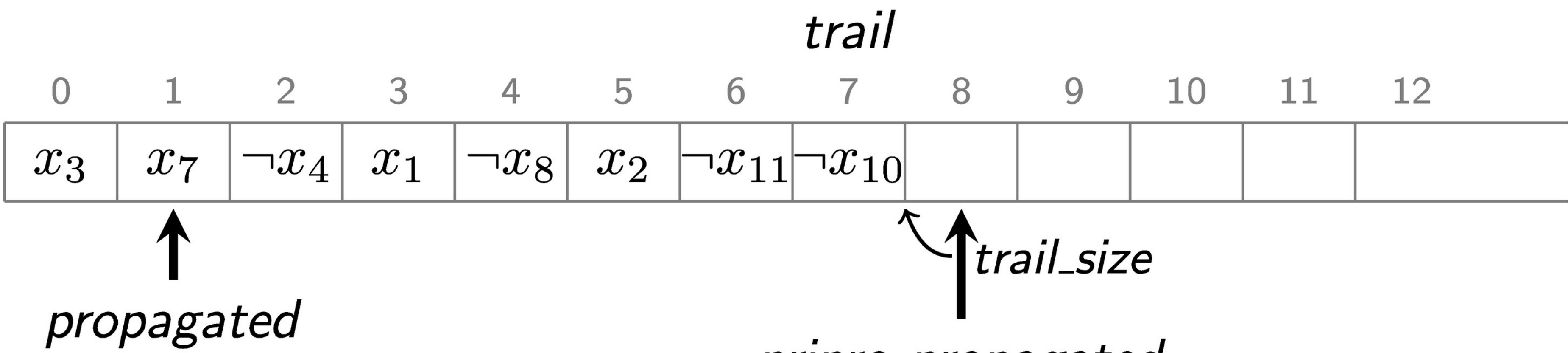
propagated $\leq$ *pripro_propagated* $\leq$ *trail_size*

Relation between Current Literals



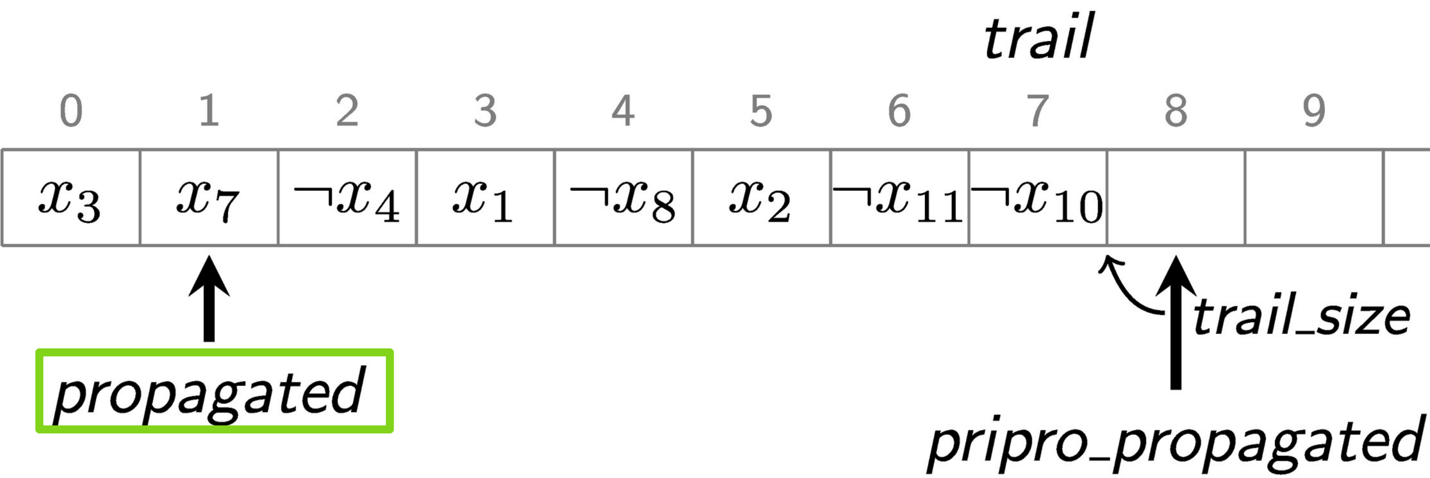
propagated $\leq$ *pripro_propagated* $\leq$ *trail_size*

Relation between Current Literals



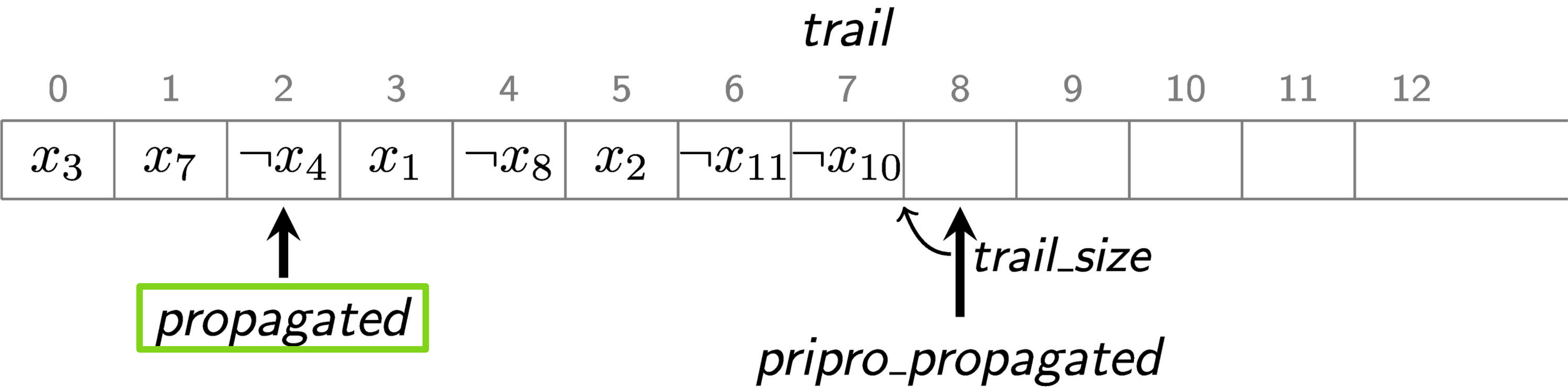
propagated $\leq$ *pripro_propagated* $\leq$ *trail_size*

Relation between Current Literals



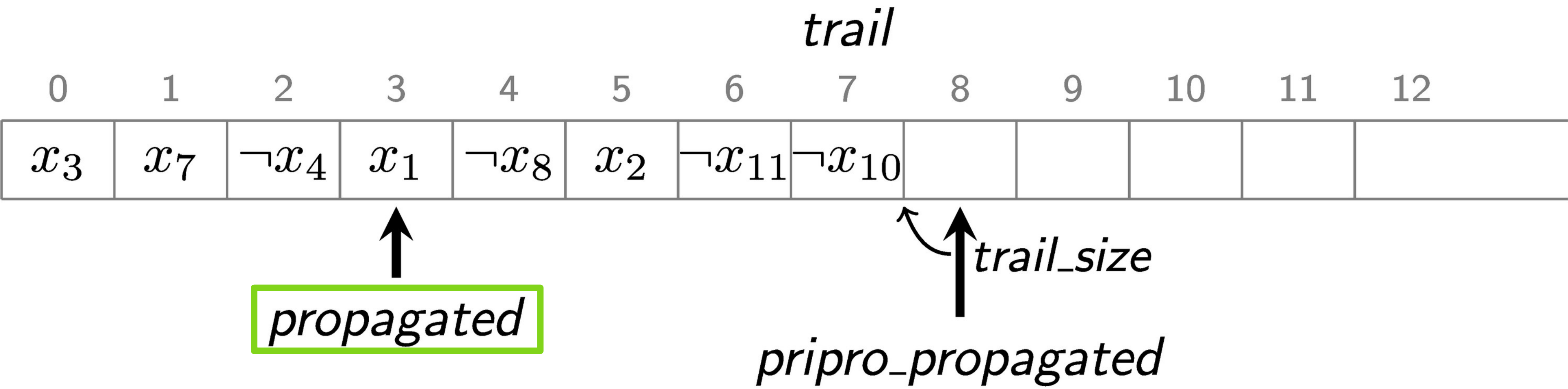
$$\textit{propagated} \leq \textit{pripro\_propagated} \leq \textit{trail\_size}$$

Relation between Current Literals



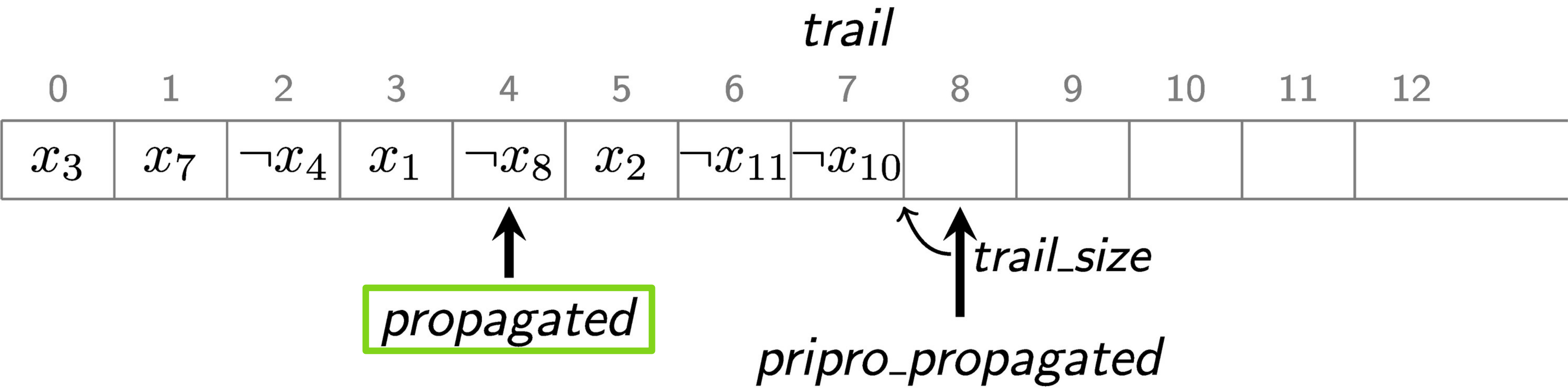
propagated $\leq$ *pripro_propagated* $\leq$ *trail_size*

Relation between Current Literals



$$\textit{propagated} \leq \textit{pripro\_propagated} \leq \textit{trail\_size}$$

Relation between Current Literals



propagated $\leq$ pripro_propagated $\leq$ trail_size

Relation between Current Literals

trail

0	1	2	3	4	5	6	7	8	9	10	11	12
x_3	x_7	$\neg x_4$	x_1	$\neg x_8$	x_2	$\neg x_{11}$	$\neg x_{10}$	x_5				

trail_size

propagated

pripro_propagated

propagated $\leq$ *pripro_propagated* $\leq$ *trail_size*

Summary: Priority Propagation

Summary: Priority Propagation

- considers the same clauses for propagation as before
- propagation order differs

Summary: Priority Propagation

- considers the same clauses for propagation as before
- propagation order differs
  - propagate all prioritised watches until end of trail
  - propagate regular watches of next literal on the trail
  - repeat until conflict or end of trail is reached

Heuristics for Prioritisation

Dynamic Prioritisation

- choose/adjust prioritised clauses at runtime

- to prioritise a clause (**upgrading)**:
  - move both watches:

    from regular to prioritised TWL scheme

- to de-prioritise a clause (**downgrading):**
  - move both watches:

    from prioritised to regular TWL scheme

Simple Heuristics

- parametrised upgrade heuristic
  - based on recent resolvents
  - upgrade on-the-fly

- parametrised downgrade heuristics
  - always downgrading all clauses at once
  - triggered by events + at constant interval

- other heuristics possible!

Upgrade Heuristic

- **upgrade** during conflict analysis**:**
  - clauses appearing in conflict analysis (**resolvents)**
    - only if size/LBD smaller than some threshold
    - **default:** LBD ≤ 6
    - (**conflicting clause** is never upgraded)
  - newly learned clauses (**conflict clauses**):
    - regardless of size/LBD

Downgrade Heuristic

- **sporadically downgrade all clauses at once**
  - **forced downgrades** triggered by
    - any inprocessing techniques (easy combination)
    - clause database reductions (i.e. clause deletions)
    - rephasing
    - optionally at restarting
  - **scheduled downgrades**:
    - at constant interval (default: 10.000 conflicts)
    - interval is measured in conflicts
    - regardless of whether forced downgrades occurred

Experimental Results

Experiments

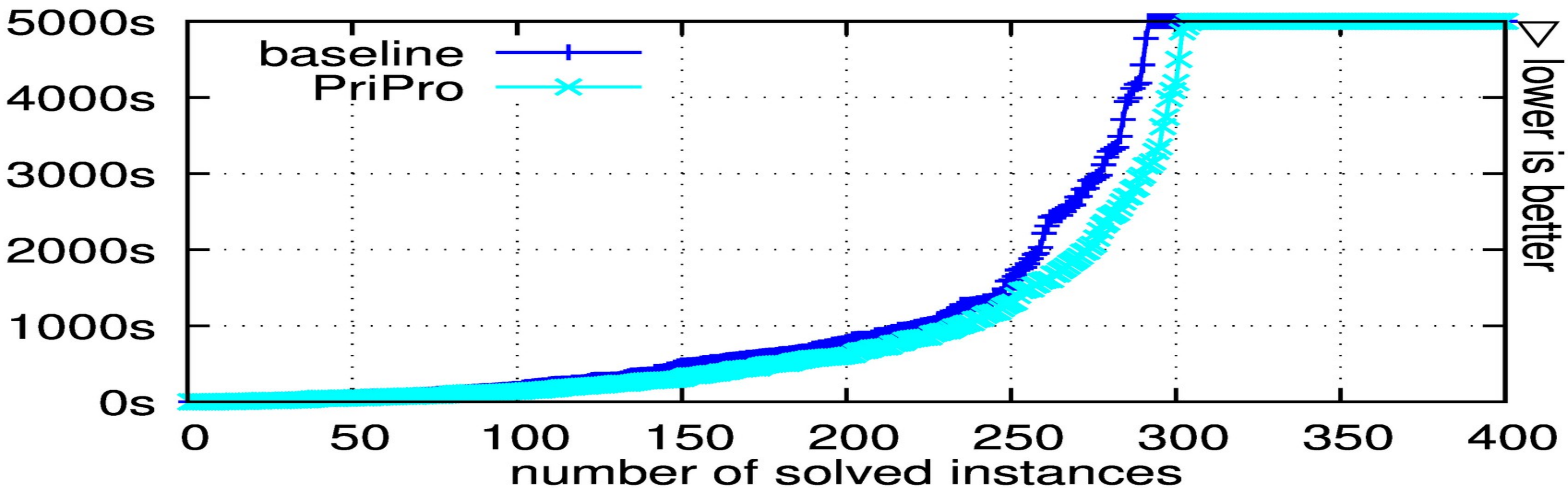
- tested on SAT Competition 2021 instances (400 problems)
- baseline solver: CaDiCaL v 1.4.0

- varying upgrade heuristics with
  - LBD limit: 0, 1, …, 8, no limit
  - size limit: 2, 4, …, 12, no limit

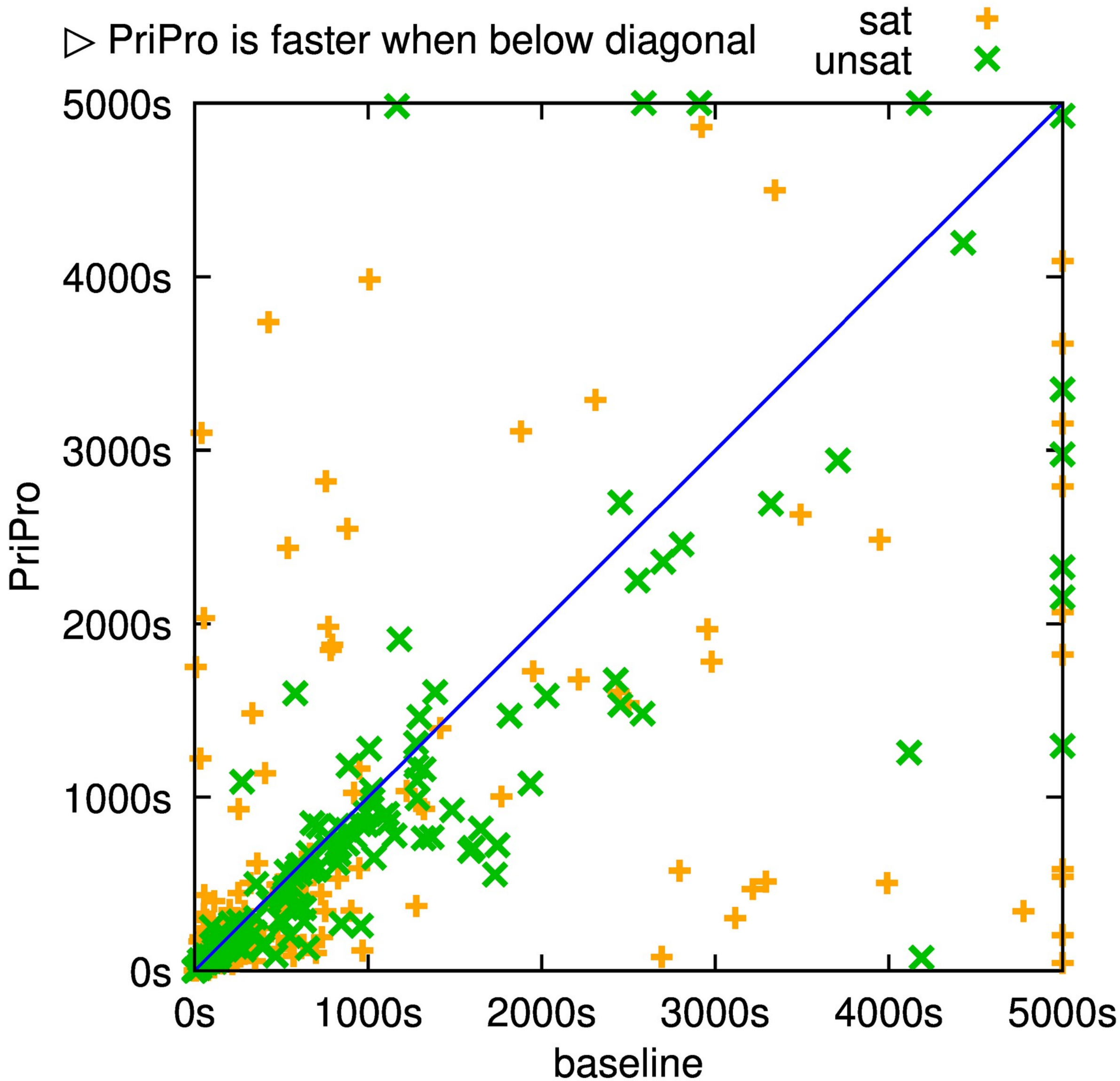
- downgrade heuristic
  - scheduled downgrade interval:
    2, 5, 10, 25, …, 1.000.000, only forced downgrades
  - optionally: downgrades at restarts

Good Results!

- **Improves performance** on both, SAT and UNSAT
  - mostly independent of parameters chosen
  - overall speed-up of about 10% (nPar2-score)
  - solved 5 to 15 instances more



Speed-up on UNSAT



Shorter Conflict Clauses Learned

- reduction of average size of learned clauses
- before learned clause minimisation:
  - about 7 % on SAT
  - about 11 % on UNSAT
- after learned clause minimisation:
  - about 6 % on SAT
  - about 21 % on UNSAT
- mostly independent of parameters chosen
- confirms hypothesis by Jingchao Chen [1]

[1] Jingchao Chen, Core First Unit Propagation, arXiv preprint: 1907.01192 (2019)

Conclusion

Summary: Priority Propagation

- **reordering clause look-ups** during Propagation
- **improves performance** on both, SAT and UNSAT
- reliable **speed-up** on UNSAT
- **smaller clauses learned** from conflict analysis
- already **simple heuristics** lead to good results
  - mostly **independent of parameters** chosen
- **easy to implement** in modern SAT solvers

Summary: Priority Propagation

- **reorder clauses** for propagation
- **improves performance** on both, SAT and UNSAT
- reliable **speed-up** on UNSAT
- **smaller clauses learned** from conflict analysis
- already **simple heuristics** lead to good results
  - mostly **independent of parameters** chosen
- **easy to implement** in modern SAT solvers

Thank you for your attention.