A Verified SAT Solver with Watched Literals Using Imperative HOL

Peter Lammich

Mathias Fleury

Jasmin C. Blanchette

How reliable are SAT solvers?

Two ways to ensure correctness:

- certify the certificate
  - certificates are huge
- verification of the code
  - code will not be competitive
  - allows to study metatheory

TUM VU VRIJE UNIVERSITEIT AMSTERDAM max planck institut informatik

How reliable is the theory?

Conference version

Branch and Bound for Boolean Optimization and the Generation of Optimality Certificates
Javier Larrosa, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell (SAT 2009)

A literal l is *true* in I if $l \in I$, *false* in I if $\neg l \in I$, and *undefined* in I otherwise.

A clause set S is true in I if all its clauses are true in I. Then I is called a *model* of S, and we write $I \models S$ (and similarly if a literal or clause is true in I).

How reliable is the theory?

Conference version

Branch and Bound for Boolean Optimization and the Generation of Optimality Certificates
Javier Larrosa, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell (SAT 2009)

A literal l is *true* in I if $l \in I$, *false* in I if $\neg l \in I$, and *undefined* in I otherwise.

A clause set S is true in I if all its clauses are true in I. Then I is called a *model* of S, and we write $I \models S$ (and similarly if a literal or clause is true in I).

Journal version

A Framework for Certified Boolean Branch-and-Bound Optimization
Javier Larrosa, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell (JAR 2011)

literals of a clause C are false in I. A clause set S is true in I if all its clauses are true in I; if I is also total, then I is called a *total model* of S, and we write $I \models S$.

I certify your proof

IsaFoL project

Isabelle Formalisation of Logic

IsaFoL

- FO resolution
  by Schlichtkrull (ITP 2016)
- CDCL with learn, forget, restart, and incrementality
  by Blanchette, Fleury, Weidenbach (IJCAR 2016)
- GRAT certificate checker
  by Lammich (CADE-26, 2017)
- A verified SAT solver with watched literals
  by Fleury, Blanchette, Lammich (CPP 2018, now)

VRIJE UNIVERSITEIT AMSTERDAM

max planck institut informatik

IsaFoL

- FO resolution
  by Schlichtkrull (ITP 2016)

- CDCL with learn, forget, restart, and incrementality
  by Blanchette, Fleury, Weidenbach (IJCAR 2016)

- GRAT certificate checker
  by Lammich (CADE-26, 2017)

- A verified SAT solver with watched literals
  by Fleury, Blanchette, Lammich (CPP 2018, now)

VRIJE UNIVERSITEIT AMSTERDAM

max planck institut informatik

Abstract CDCL
Previous work

↑ refines

Watched Literals Calculus
Transition system

↑ refines

Watched Literals Algorithm
Non-deterministic program

↑ refines

Refined SAT solver
Towards efficient data structures

↑ refines

Executable SAT solver
Standard ML

Abstract CDCL
Previous work

DPLL

Candidate model

Clause

1.	¬B	∨	C	∨	A
2.	¬C	∨	¬B	∨	¬A
3.	¬A	∨	¬B	∨	C
4.	¬A	∨	B		

1. Guess
2. or propagate information
3. or take the opposite of the last guess if there is a conflict

DPLL

Candidate model

$A^?$

Clause

1.	¬B	∨	C	∨	A
2.	¬C	∨	¬B	∨	¬A
3.	¬A	∨	¬B	∨	C
4.	¬A	∨	B		

1. Guess
2. or propagate information
3. or take the opposite of the last guess if there is a conflict

DPLL

Candidate model

$A^?$ B

Clause

1. $\neg B \vee C \vee A$
2. $\neg C \vee \neg B \vee \neg A$
3. $\neg A \vee \neg B \vee C$
4. $\neg A \vee B$

1. Guess
2. or propagate information
3. or take the opposite of the last guess if there is a conflict

DPLL

Candidate model	Clause
$A^?$ B $\neg C$	1. $\neg B \vee C \vee A$
	2. $\neg C \vee \neg B \vee \neg A$
	3. $\neg A \vee \neg B \vee C$
	4. $\neg A \vee B$

1. Guess
2. or propagate information
3. or take the opposite of the last guess
   if there is a conflict

DPLL

Candidate model

¬A

Clause

1.	¬B	∨	C	∨	A
2.	¬C	∨	¬B	∨	¬A
3.	¬A	∨	¬B	∨	C
4.	¬A	∨	B		

1. Guess
2. or propagate information
3. or take the opposite of the last guess if there is a conflict

DPLL

Candidate model
¬A ¬C?

Clause					
1.	¬B	∨	C	∨	A
2.	¬C	∨	¬B	∨	¬A
3.	¬A	∨	¬B	∨	C
4.	¬A	∨	B		

1. Guess
2. or propagate information
3. or take the opposite of the last guess if there is a conflict

TUM VU VRIJE UNIVERSITEIT AMSTERDAM max planck institut informatik

DPLL

Candidate model

¬A ¬C? ¬B

Clause

1. ¬B ∨ C ∨ A
2. ¬C ∨ ¬B ∨ ¬A
3. ¬A ∨ ¬B ∨ C
4. ¬A ∨ B

1. Guess
2. or propagate information
3. or take the opposite of the last guess if there is a conflict

DPLL

Candidate model	Clause
¬A	1. ¬B ∨ C ∨ A
	2. ¬C ∨ ¬B ∨ ¬A
	3. ¬A ∨ ¬B ∨ C
	4. ¬A ∨ B
	5. ¬A

CDCL = DPLL +
non-chronological backtracking +
learning

in Isabelle

Propagate rule

```
C ∨ L ∈ N ⟹ M ⊨as ¬C ⟹ undefined_lit M L ⟹
    (M, N) ⇒CDCL (L # M, N)
```

TUM VU VRIJE UNIVERSITEIT AMSTERDAM max planck institut informatik

in Isabelle

Propagate rule

```
C ∨ L ∈ N ⟹ M ⊨as ¬C ⟹ undefined_lit M L ⟹
    (M, N) ⇒CDCL (L # M, N)
```

Problem:
Iterating over the clauses is inefficient

Abstract CDCL

Previous work

refines

Watched Literals Calculus

Transition system

refines

Watched Literals Algorithm

Non-Deterministic program

refines

Refined SAT solver

Towards efficient data structures

refines

Executable SAT solver

Standard ML

max planck institut
informatik

Watched Literals Calculus

Transition system

DPLL with Watched Literals

Candidate model

Clause

1. $\boxed{\neg B} \vee \boxed{C} \vee A$
2. $\boxed{\neg C} \vee \boxed{\neg B} \vee \neg A$
3. $\boxed{\neg A} \vee \boxed{\neg B} \vee C$
4. $\boxed{\neg A} \vee \boxed{B}$

1. Watch one true literal
2. or watch two unset literals
3. or watch a false literal
   if all other literals are false

To update:

DPLL with Watched Literals

Candidate model

A?

Clause

1. $\boxed{\neg B} \vee \boxed{C} \vee A$
2. $\boxed{\neg C} \vee \boxed{\neg B} \vee \neg A$
3. $\boxed{\neg A} \vee \boxed{\neg B} \vee C$
4. $\boxed{\neg A} \vee \boxed{B}$

1. Watch one true literal
2. or watch two unset literals
3. or watch a false literal
   if all other literals are false

To update:

DPLL with Watched Literals

Candidate model

$A^?$

Clause

1. $\boxed{\neg B} \vee \boxed{C} \vee A$
2. $\boxed{\neg C} \vee \boxed{\neg B} \vee \neg A$
3. $\boxed{\neg A} \vee \boxed{\neg B} \vee C$
4. $\boxed{\neg A} \vee \boxed{B}$

1. Watch one true literal
2. or watch two unset literals
3. or watch a false literal
   if all other literals are false

To update: **3. 4.**

DPLL with Watched Literals

Candidate model	Clause
$A^?$	1. $\neg B \vee C \vee A$
	2. $\neg C \vee \neg B \vee \neg A$
	3. $\neg A \vee \neg B \vee C$
	4. $\neg A \vee B$

1. Watch one true literal
2. or watch two unset literals
3. or watch a false literal
   if all other literals are false

To update: 4.

DPLL with Watched Literals

Candidate model

A?

Clause

1. $\boxed{\neg B} \vee \boxed{C} \vee A$
2. $\boxed{\neg C} \vee \boxed{\neg B} \vee \neg A$
3. $\neg A \vee \boxed{\neg B} \vee \boxed{C}$
4. $\boxed{\neg A} \vee \boxed{B}$

1. Watch one true literal
2. or watch two unset literals
3. or watch a false literal
   if all other literals are false

To update:

DPLL with Watched Literals

Candidate model	Clause
$A^?$ B	1. ¬B ∨ C ∨ A
	2. ¬C ∨ ¬B ∨ ¬A
	3. ¬A ∨ ¬B ∨ C
	4. ¬A ∨ B

1. Watch one true literal
2. or watch two unset literals
3. or watch a false literal
   if all other literals are false

To update:

B

DPLL with Watched Literals

Candidate model	Clause
$A^?$ B	1. ¬B ∨ C ∨ A
	2. ¬C ∨ ¬B ∨ ¬A
	3. ¬A ∨ ¬B ∨ C
	4. ¬A ∨ B

1. Watch one true literal
2. or watch two unset literals
3. or watch a false literal
   if all other literals are false

To update: **1. 2. 3.**

DPLL with Watched Literals

Candidate model

$A^?$ B

1. Watch one true literal
2. or watch two unset literals
3. or watch a false literal
   if all other literals are false

Clause

1. ¬B ∨ $\boxed{C}$ ∨ $\boxed{A}$
2. $\boxed{¬C}$ ∨ $\boxed{¬B}$ ∨ ¬A
3. ¬A ∨ $\boxed{¬B}$ ∨ $\boxed{C}$
4. $\boxed{¬A}$ ∨ $\boxed{B}$

To update: **2. 3.**

DPLL with Watched Literals

Candidate model

A? B

Clause

1. ¬B ∨ C ∨ A
2. ¬C ∨ ¬B ∨ ¬A
3. ¬A ∨ ¬B ∨ C
4. ¬A ∨ B

1. Watch one true literal
2. or watch two unset literals
3. or watch a false literal
   if all other literals are false

To update: **3.**

DPLL with Watched Literals

Candidate model

$A^?$ B ¬C

Clause

1. ¬B ∨ $\boxed{C}$ ∨ $\boxed{A}$
2. $\boxed{\neg C}$ ∨ $\boxed{\neg B}$ ∨ ¬A
3. ¬A ∨ $\boxed{\neg B}$ ∨ $\boxed{C}$
4. $\boxed{\neg A}$ ∨ $\boxed{B}$

1. Watch one true literal
2. or watch two unset literals
3. or watch a false literal
   if all other literals are false

To update: 3.

C

DPLL with Watched Literals

Candidate model

A? B ¬C

Clause

1. $\neg B \lor \boxed{C} \lor \boxed{A}$
2. $\boxed{\neg C} \lor \boxed{\neg B} \lor \neg A$
3. $\neg A \lor \boxed{\neg B} \lor \boxed{C}$
4. $\boxed{\neg A} \lor \boxed{B}$

1. Watch one true literal
2. or watch two unset literals
3. or watch a false literal
   if all other literals are false

To update:

DPLL with Watched Literals

Candidate model	Clause
¬A	1. ¬B ∨ $\boxed{C}$ ∨ $\boxed{A}$
	2. $\boxed{¬C}$ ∨ $\boxed{¬B}$ ∨ ¬A
	3. ¬A ∨ $\boxed{¬B}$ ∨ $\boxed{C}$
	4. $\boxed{¬A}$ ∨ $\boxed{B}$

1. Watch one true literal
2. or watch two unset literals
3. or watch a false literal
   if all other literals are false

To update:

¬A

DPLL with Watched Literals

Candidate model	Clause
¬A	1. ¬B ∨ C ∨ A
	2. ¬C ∨ ¬B ∨ ¬A
	3. ¬A ∨ ¬B ∨ C
	4. ¬A ∨ B
	5. ¬A

1. Watch one true literal
2. or watch two unset literals
3. or watch a false literal
   if all other literals are false

To update:

¬A

Watched literals invariant

1. Watch one true literal
2. or watch two unset literals
3. or watch a false literal
   if all other literals are false

Watched literals invariant

1. Watch one true literal
2. or watch two unset literals
3. or watch a false literal
   if all other literals are false

unless a conflict has been found

Watched literals invariant

1. Watch one true literal
2. or watch two unset literals
3. or watch a false literal
   if all other literals are false

unless a conflict has been found

or an update is pending

Watched literals invariant (less wrong)

this literal has been set earlier

1. Watch one true literal
2. or watch two unset literals
3. or watch a false literal
   if all other literals are false

unless a conflict has been found

or an update is pending

Watched literals invariant

1. Watch one true literal
2. or watch two unset literals
3. or watch a false literal
   if all other literals are false

1. Watch any literal
   if there is a true literal
2. or watch two unset literals
3. or watch a false literal
   if all other literals are false

Watched literals invariant

with blocking literals

1. Watch one true literal
2. or watch two unset literals
3. or watch a false literal
   if all other literals are false

1. Watch any literal
   if there is a true literal
2. or watch two unset literals
3. or watch a false literal
   if all other literals are false

Watched literals invariant

with blocking literals

$\neg B \lor C \lor A$

1. Watch one true literal
2. or watch two unset literals
3. or watch a false literal
   if all other literals are false

1. Watch any literal
   if there is a true literal
2. or watch two unset literals
3. or watch a false literal
   if all other literals are false

Watched literals invariant

with blocking literals

$\boxed{\neg B} \vee \boxed{C} \vee A$

1. Watch one true literal
2. or watch two unset literals
3. or watch a false literal
   if all other literals are false

1. Watch any literal
   if there is a true literal
2. or watch two unset literals
3. or watch a false literal
   if all other literals are false

(not yet refined to code)

Blocking Literals (historical perspective)

- Barcelogic Tool (?): save other watched literal in watch list

- Chu et al. (08): save some literals in watch list

- Minisat 2.1 (08): save one literal from the clause in watch list

- Ryan (04): binary/ternary clause handling

Finding invariants (11 new ones)

No high-level description

sledgehammer

Finding invariants (11 new ones)

No high-level description

sledgehammer

Correctness theorem

in Isabelle

If S is well-formed and $S \Rightarrow_{\mathtt{TWL}}^{!} T$ then

$$S \Rightarrow_{\mathtt{CDCL}}^{!} T$$

Abstract CDCL
Previous work

↑ refines

Watched Literals Calculus
Transition system

↑ refines

Watched Literals Algorithm
Non-deterministic Program

↑ refines

Refined SAT solver
Towards efficient data structures

↑ refines

Executable SAT solver
Standard ML

VU VRIJE UNIVERSITEIT AMSTERDAM

max planck institut informatik

Watched Literals Calculus

Transition system

Watched Literals Algorithm

Non-deterministic Program

Picking Next Clause

```
propagate_conflict_literal L S :=
  WHILE_T
    (λT. clauses_to_update T ≠ {})

    (λT. do {
       ASSERT(clauses_to_update T ≠ {})
       C ← SPEC (λC. C ∈ clauses_to_update T);
       U ← remove_from_clauses_to_update C T;
       update_clause (L, C) U
     }
    )

    S
```

Refinement Framework: non-deterministic exception monad

```
propagate_conflict_literal L S :=
  WHILE_T
    (λT. clauses_to_update T ≠ {})

    (λT. do {
       ASSERT(clauses_to_update T ≠ {})
       C ← SPEC (λC. C ∈ clauses_to_update T);
       U ← remove_from_clauses_to_update C T;
       update_clause (L, C) U
     }
    )

    S
```

Refinement Framework:
non-deterministic exception monad

Assertions

```
propagate_conflict_literal L S :=
 WHILE_T
   (λT. clauses_to_update T ≠ {})

   (λT. do {
     ASSERT(clauses_to_update T ≠ {})
     C ← SPEC (λC. C ∈ clauses_to_update T);
     U ← remove_from_clauses_to_update C T;
     update_clause (L, C) U
    }
   )

   S
```

Refinement Framework:
non-deterministic exception monad

```
propagate_conflict_literal L S :=
  WHILE_T
    (λT. clauses_to_update T ≠ {})

    (λT. do {
       ASSERT(clauses_to_update T ≠ {})
       C ← SPEC (λC. C ∈ clauses_to_update T);
       U ← remove_from_clauses_to_update C T;
       update_clause (L, C) U
     }
    )

    S
```

Non-deterministic getting of a clause

Refinement Framework: non-deterministic exception monad

```
propagate_conflict_literal L S :=
  WHILE_T
    (λT. clauses_to_update T ≠ {})

    (λT. do {
       ASSERT(clauses_to_update T ≠ {})
       C ← SPEC (λC. C ∈ clauses_to_update T);
       U ← remove_from_clauses_to_update C T;
       update_clause (L, C) U
     }
    )

    S
```

- More deterministic (order of the rules)
- But still non deterministic (decisions)
- Goals of the form

- More deterministic (order of the rules)
- But still non deterministic (decisions)
- Goals of the form

```
propagate_conflict_literal L S ≤ SPEC(λT. S ⇒TWL* T)
```

in Isabelle

✗ VCG's goals hard to read

✗ Very tempting to write fragile proofs

✓ sledgehammer

TUM VU VRIJE UNIVERSITEIT AMSTERDAM max planck institut informatik

Abstract CDCL

Previous work

↑ refines

Watched Literals Calculus

Transition system

↑ refines

Watched Literals Algorithm

Non-deterministic Program

↑ refines

Refined SAT Solver

Towards efficient data structures

↑ refines

Executable SAT solver

Standard ML

VRIJE UNIVERSITEIT AMSTERDAM

max planck institut informatik

Watched Literals Algorithm
Non-deterministic Program

↑

Refined SAT Solver
Towards efficient data structures

VRIJE UNIVERSITEIT AMSTERDAM

max planck institut informatik

DPLL with Watched Literals

Clauses (multisets)
1. $\boxed{\neg B} \vee \boxed{C} \vee A$
2. $\boxed{\neg C} \vee \boxed{\neg B} \vee \neg A$
3. $\neg A \vee \boxed{\neg B} \vee \boxed{C}$
4. $\boxed{\neg A} \vee \boxed{B}$

Clauses after refinement (lists)			
1.	$\boxed{\neg B}$	$\boxed{C}$	A
2.	$\boxed{\neg C}$	$\boxed{\neg B}$	¬A
3.	$\boxed{C}$	$\boxed{\neg B}$	¬A
4.	$\boxed{\neg A}$	$\boxed{B}$	

To update:

A: ¬A: 4 C: 1,3 ¬C: 2

B: 4 ¬B: 1,2,3

```
propagate_conflict_literal L S :=
  WHILE_T
    (λT. clauses_to_update T ≠ {})

    (λT. do {
       ASSERT(clauses_to_update T ≠ {})
       C ← SPEC (λC. C ∈ clauses_to_update T);
       U ← remove_from_clauses_to_update C T;
       update_clause L C U
     }
    )

    S
```

```
propagate_conflict_literal_list L S :=
  WHILE_T
    (λ(w, T). w < length (watched_by T L))

    (λ(w, T). do {
       C ← (watched_by T L) ! w;
       update_clause_list L C T
     }
    )

    (S, 0)
```

```
propagate_conflict_literal L S :=
  WHILE_T
    (λT. clauses_to_update T ≠ {})

    (λT. do {
       ASSERT(clauses_to_update T ≠ {})
       C ← SPEC (λC. C ∈ clauses_to_update T);
       U ← remove_from_clauses_to_update C T;
       update_clause L C U
     }
    )

    S
```

```
propagate_conflict_literal_list L S :=
  WHILE_T
    (λ(w, T). w < length (watched_by T L))

    (λ(w, T). do {
       C ← (watched_by T L) ! w;
       update_clause_list L C T
     }
    )

    (S, 0)
```

```
propagate_conflict_literal_list L S ≤ ⇓ conversion_between_states
    (propagate_conflict_literal L T)
```

in Isabelle

- More new invariants
- Aligning goals is hard...
- Fast code uses many invariants
- Forgotten and new invariants
- sledgehammer

TUM VU VRIJE UNIVERSITEIT AMSTERDAM max planck institut informatik

- Choice on the heuristics
- Choice on the data structures
- Prepare code synthesis

Decision heuristic

- Variable-move-to-front heuristic
- No correctness w.r.t. a standard implementation
- Behaves correctly:
  - returns an unset literal if there is one
  - no exception (out-of-bound array accesses)

VRIJE UNIVERSITEIT AMSTERDAM

max planck institut informatik

Abstract CDCL

Previous work

↑ refines

Watched Literals Calculus

Transition system

↑ refines

Watched Literals Algorithm

Non-deterministic program

↑ refines

Refined SAT Solver

Towards efficient data structures

↑ refines

Executable SAT Solver

Standard ML

VRIJE UNIVERSITEIT AMSTERDAM

max planck institut informatik

Refined SAT Solver

Towards efficient data structures

↑

Executable SAT Solver

Standard ML

```
sepref_definition executable_version
  is ‹propagate_conflict_literal_heuristics›
  :: ‹unat_lit_assn^k *_a state_assn^d →_a state_assn›
  by sepref
```

Synthesise imperative code and a refinement relation

```
sepref_definition executable_version
  is ‹propagate_conflict_literal_heuristics›
  :: ‹unat_lit_assn^k *_a state_assn^d →_a state_assn›
  by sepref
```

Synthesise imperative code and a refinement relation

```
main_loop S :=
   heap_WHILET
       (λ(finished, _). return (¬ finished))
       (λ(_, state).
           propagate state ⤜
           analyse_or_decide)
       (False, state) ⤜
    (λ(_, final_state). return final_state)
```

```
sepref_definition executable_version
  is ‹propagate_conflict_literal_heuristics›
  :: ‹unat_lit_assnᵏ *ₐ state_assnᵈ →ₐ state_assn›
  by sepref
```

Synthesise imperative code and a refinement relation

```
fun main_loop state =
 fn () =>
  let
   val (_, final_state) =
    heap_WHILET
      (fn (done, _) => (fn () => not done))
      (fn (_, state) =>
        (analyse_or_decide (propagate state ()) ()))
      (false, xi)
      ();
  in final_state end;
```

```
sepref_definition executable_version
  is ‹propagate_conflict_literal_heuristics›
  :: ‹unat_lit_assn^k *_a state_assn^d →_a state_assn›
  by sepref
```

Synthesise imperative code and a refinement relation

```
fun cdcl_twl_stgy_prog_wl_D_code x =
  (fn xi => fn () =>
    let
      val a =
        heap_WHILET (fn (a1, _) => (fn () => (not a1)))
          (fn (_, a2) =>
            (fn f_ => fn () => f_ ((unit_propagation_outer_loop_wl_D a2) ()) ())
              cdcl_twl_o_prog_wl_D_code)
          (false, xi) ();
    in
      let
        val (_, aa) = a;
      in
        (fn () => aa)
      end
        ()
```

Choice on the data structures

Clauses: resizable arrays of (fixed sized) arrays

However, no aliasing

- Indices instead of pointers
- N[C] makes a copy, so only use N[C][i]

✓ Generates imperative code

✗ No error messages

✗ Transformations before generating code

Clauses of length 0 and 1

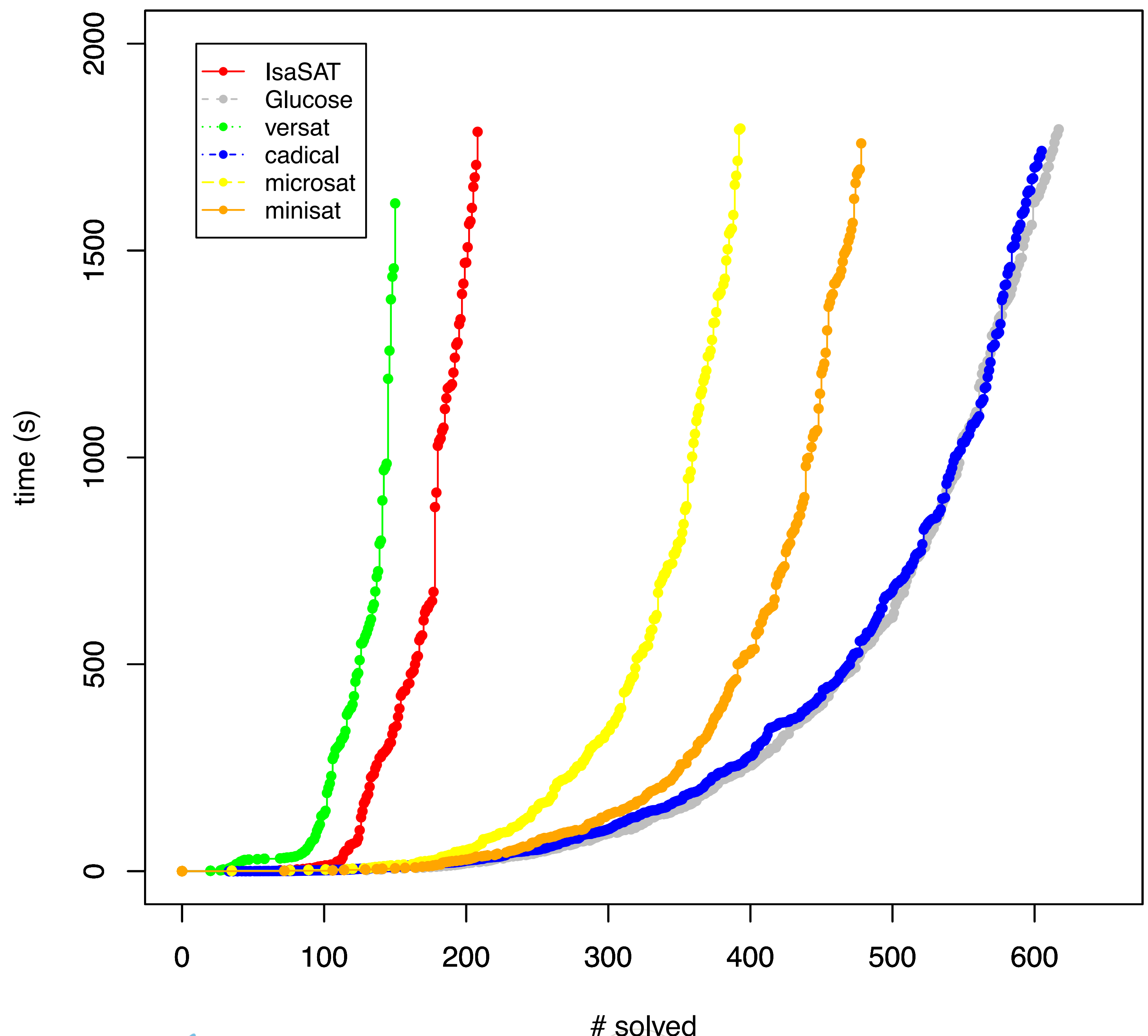
Once combined with an initialisation:

in Isabelle

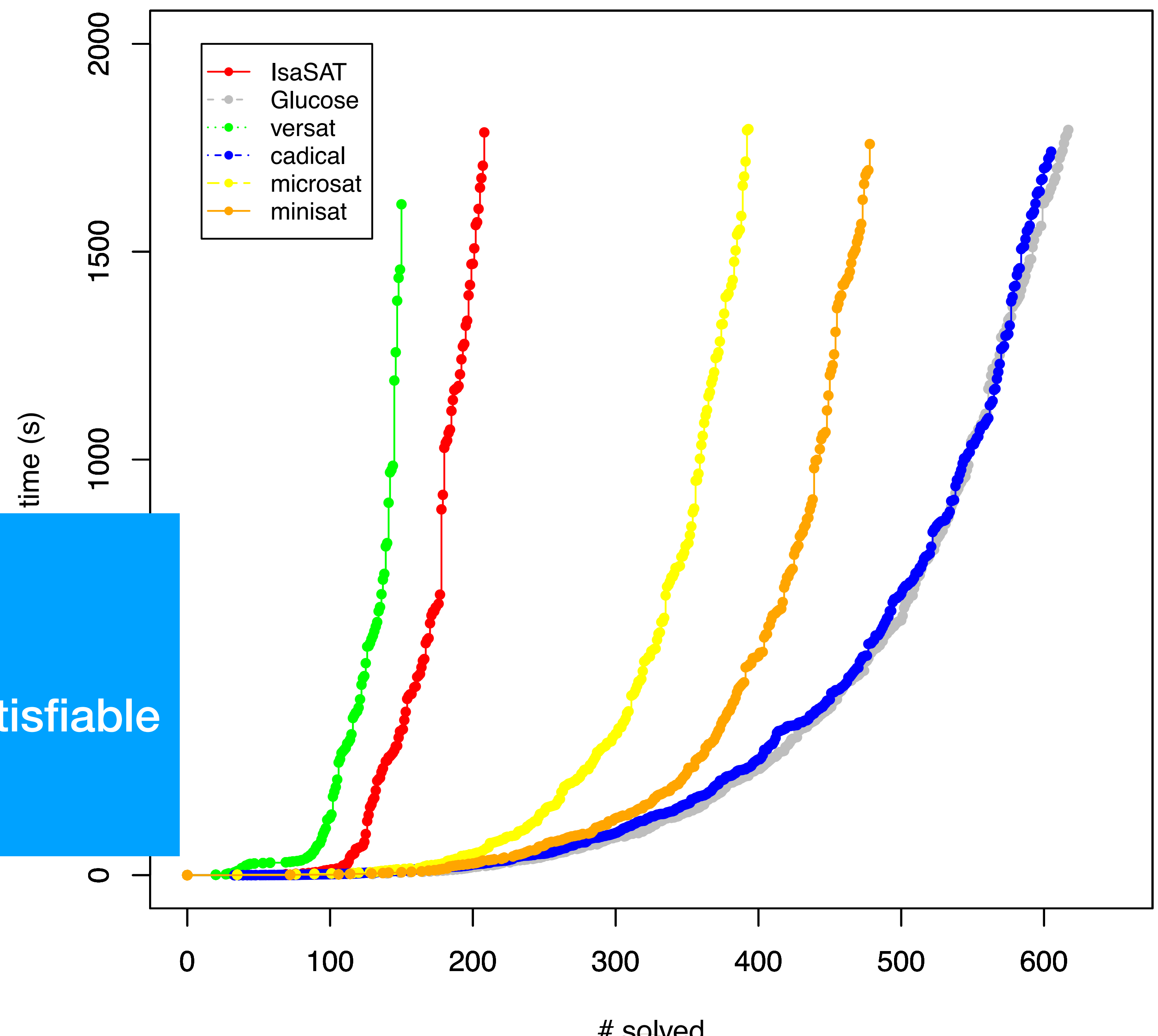
```
‹(IsaSAT_code, model_if_satisfiable)
    ∈ [λN. each_clause_is_distinct N ∧
        literals_fit_in_32_bit_integer N]_a
      clauses_as_lists^k → model›
```

Exported code tested with an unchecked parser
(easy and medium problems from the SAT competition 2009)

SAT-Comp ’09, ’15 (main track), and ’14 (all submitted problems), already preprocessed, duplicates removed



SAT-Comp ’09, ’15 (main track), and ’14 (all submitted problems), already preprocessed, duplicates removed



Correct up to:

- run-time checks
- checking the model is satisfiable

SAT-Comp '09, '15 (main track), and '14 (all submitted problems), already preprocessed

1192 problems,
30 minutes timeout

	#solved	Average time (s)	Crash or errors
versat	159	233	4 (?)
IsaSAT	200	147	3 (OOM)
microsat	483	297	
MiniSAT	582	280	
cadical	759	330	
glucose	784	337	

Abstract CDCL
Previous work

↑ refines

Watched Literals Calculus
Transition system

↑ refines

Watched Literals Algorithm
Non-deterministic program

↑ refines

Refined SAT Solver
Towards efficient data structures

↑ refines

Executable SAT solver
Standard ML

Abstract CDCL
Previous work

↑ refines

Watched Literals Calculus
Transition system

↑ refines

Watched Literals Algorithm
Non-deterministic program

↑ refines

Refined SAT Solver
Towards efficient data structures

- better implementation (trail, conflict)
- dynamic decision heuristic

↑ refines

Executable SAT solver
Standard ML

VU VRIJE UNIVERSITEIT AMSTERDAM

max planck institut informatik

Abstract CDCL
Previous work

refines ↑

Watched Literals Calculus
Transition system

refines ↑

Watched Literals Algorithm
Non-deterministic program

refines ↑

Refined SAT Solver
Towards efficient data structures

refines ↑

Executable SAT solver
Standard ML

- allow learned clause minimisation
- no reuse of restarts

- better implementation (trail, conflict)
- dynamic decision heuristic

- learned clause minimisation

Abstract CDCL
Previous work

- allow learned clause minimisation
- no reuse of restarts

↑ refines

Watched Literals Calculus
Transition system

- more invariants

↑ refines

Watched Literals Algorithm
Non-deterministic program

↑ refines

Refined SAT Solver
Towards efficient data structures

- better implementation (trail, conflict)
- dynamic decision heuristic
- learned clause minimisation

↑ refines

Executable SAT solver
Standard ML

VRIJE UNIVERSITEIT AMSTERDAM

max planck institut informatik

How hard is it?

	Paper	Proof assistant
Very abstract	13 pages	50 pages
Abstract CDCL	9 pages (½ month)	90 pages (5 months)
Watched Literals	1 page (C++ code of MiniSat)	600 pages (15 months)

Conclusion

Concrete outcome

- Watched literals optimisation
- Verified executable SAT solver

Methodology

- Refinement using the Refinement Framework
- No proof of heuristics (w.r.t. standard)

Future work

- Restarts and blocking literals (ongoing)
- Use SAT solver in IsaFoR

VRIJE UNIVERSITEIT AMSTERDAM

max planck institut informatik

Annex

What is in IsaSAT?

Conflict Analysis

- conflict as lookup table (Minisat)
- and as explicit array (Minisat's "outl", to simplify proofs)

Decisions

- Variable move to front (Splatz, cadical)

Propagations

- Mostly following MiniSAT (without BLIT)

VRIJE UNIVERSITEIT AMSTERDAM

max planck institut informatik

```
for (i = j = 1; i < out_learnt.size(); i++)
  if (reason(var(out_learnt[i])) == CRef_Undef ||
     !litRedundant(out_learnt[i]))
     out_learnt[j++] = out_learnt[i];
```

```
fun minimize_and_extract_highest_lookup_conflict_code x =
 (fn ai => fn bid => fn bic => fn bib => fn bia => fn bi => fn () =>
  let
   val a =
    heap_WHILET
     (fn (_, (a1a, (_, a2b))) =>
      (fn f_ => fn () => f_ ((length_arl_u_code heap_uint32 a2b) ()) ())
       (fn x_a => (fn () => (Word32.< (a1a, x_a)))))
     (fn (a1, (a1a, (a1b, a2b))) =>
      (fn f_ => fn () => f_
       (((fn () => Array.sub (fst a2b, Word32.toInt a1a))) ()) ())
       (fn x_a =>
        (fn f_ => fn () => f_
         ((literal_redundant_wl_lookup_code ai bid a1 a1b x_a bia) ())
         ())
         (fn (a1c, (_, a2d)) =>
          (if not a2d
           then (fn () =>
                (a1, (Word32.+ (a1a, (Word32.fromInt 1)),
                     (a1c, a2b))))
           else (fn f_ => fn () => f_
                ((delete_from_lookup_conflict_code x_a a1) ()) ())
                (fn x_e =>
                  (fn f_ => fn () => f_ ((arl_last heap_uint32 a2b)
                   ()) ())
                   (fn xa =>
                    (fn f_ => fn () => f_
                     ((arl_set_u heap_uint32 a2b a1a xa) ()) ())
                     (fn xb =>
                      (fn f_ => fn () => f_
 ((arl_butlast heap_uint32 xb) ()) ())
 (fn xc => (fn () => (x_e, (a1a, (a1c, xc)))))))))))))
     (bic, ((Word32.fromInt 1), (bib, bi))) ();
  in
   let
    val (a1, (_, (a1b, a2b))) = a;
```

How much is missing?

Features (I)

10

Thank you, Norbert & Mate!

- arena based memory allocation for clauses and watchers
- blocking literals (BLIT)
- special handling of binary clause watches
- literal-move-to-front watch replacement (LMTF)
- learned clause minimization with poison
- on-the-fly hyper-binary resolution (HBR)
- learning additional units and binary clauses (multiple UIPs)
- on-the-fly self-subsuming resolution (OTFS)
- decision only clauses (DECO)
- failed literal probing on binary implication graph roots
- eager recent learned clause subsumption

Slides by Armin Biere

Features (II)

11

- stamping based VMTF instead of VSIDS
- subsumption for both irredundant and learned clauses
- inprocessing blocked clause decomposition (BCD) enabling . . .
- . . . inprocessing SAT sweeping for backbones and equivalences
- equivalent literal substitution (ELS)
- bounded variable elimination (BVE)
- blocked clause elimination (BCE)
- **dynamic sticky clause reduction**
- exponential moving average based restart scheduling
- delaying restarts
- trail reuse

How much is missing?

Features (I) 10

- arena based memory allocation for clauses and watchers
- blocking literals (BLIT)
- special handling of binary clause watches
- literal-move-to-front watch replacement (LMTF)
- learned clause minimization with poison
- on-the-fly hyper-binary resolution (HBR)
- learning additional units and binary clauses (multiple UIPs)
- on-the-fly self-subsuming resolution (OTFS)
- decision only clauses (DECO)
- failed literal probing on binary implication graph roots
- eager recent learned clause subsumption

Thank you, Norbert & Mate!

Splatz @ POS'15

Code only

Slides by Armin Biere

Features (II) 11

- stamping based VMTF instead of VSIDS
- subsumption for both irredundant and learned clauses
- inprocessing blocked clause decomposition (BCD) enabling ...
- ... inprocessing SAT sweeping for backbones and equivalences
- equivalent literal substitution (ELS)
- bounded variable elimination (BVE)
- blocked clause elimination (BCE)
- **dynamic sticky clause reduction**
- exponential moving average based restart scheduling
- delaying restarts
- trail reuse

Splatz @ POS'15

How much is missing?

Features (I)

10

- arena based memory allocation for clauses and watchers
- blocking literals (BLIT)
- special handling of binary clause watches
- literal-move-to-front watch replacement (LMTF)
- learned clause minimization with poison
- on-the-fly hyper-binary resolution (HBR)
- learning additional units and binary clauses (multiple UIPs)
- on-the-fly self-subsuming resolution (OTFS)
- decision only clauses (DECO)
- failed literal probing on binary implication graph roots
- eager recent learned clause subsumption

Splatz @ POS'15

Thank you, Norbert & Mate!

Slides by Armin Biere

Features (II)

11

- stamping based VMTF instead of VSIDS
- subsumption for both irredundant and learned clauses
- inprocessing blocked clause decomposition (BCD) enabling ...
- ... inprocessing SAT sweeping for backbones and equivalences
- equivalent literal substitution (ELS)
- bounded variable elimination (BVE)
- blocked clause elimination (BCE)
- **dynamic sticky clause reduction**
- exponential moving average based restart scheduling
- delaying restarts
- trail reuse

Splatz @ POS'15

Code only

Strengthening

How much is missing?

Features (I) 10

- arena based memory allocation for clauses and watchers
- blocking literals (BLIT)
- special handling of binary clause watches
- literal-move-to-front watch replacement (LMTF)
- learned clause minimization with poison
- on-the-fly hyper-binary resolution (HBR)
- learning additional units and binary clauses (multiple UIPs)
- on-the-fly self-subsuming resolution (OTFS)
- decision only clauses (DECO)
- failed literal probing on binary implication graph roots
- eager recent learned clause subsumption

Splatz @ POS'15

Thank you, Norbert & Mate!

Slides by Armin Biere

Features (II) 11

- stamping based VMTF instead of VSIDS
- subsumption for both irredundant and learned clauses
- inprocessing blocked clause decomposition (BCD) enabling ...
- ... inprocessing SAT sweeping for backbones and equivalences
- equivalent literal substitution (ELS)
- bounded variable elimination (BVE)
- blocked clause elimination (BCE)
- **dynamic sticky clause reduction**
- exponential moving average based restart scheduling
- delaying restarts
- trail reuse

Splatz @ POS'15

Code only

Strengthening

Change CDCL

VRIJE UNIVERSITEIT AMSTERDAM

max planck institut informatik

How much is missing?

Features (I)

10

- arena based memory allocation for clauses and watchers
- blocking literals (BLIT)
- special handling of binary clause watches
- literal-move-to-front watch replacement (LMTF)
- learned clause minimization with poison
- on-the-fly hyper-binary resolution (HBR)
- learning additional units and binary clauses (multiple UIPs)
- on-the-fly self-subsuming resolution (OTFS)
- decision only clauses (DECO)
- failed literal probing on binary implication graph roots
- eager recent learned clause subsumption

Splatz @ POS'15

Thank you, Norbert & Mate!

Slides by Armin Biere

Features (II)

11

- stamping based VMTF instead of VSIDS
- subsumption for both irredundant and learned clauses
- inprocessing blocked clause decomposition (BCD) enabling ...
- ... inprocessing SAT sweeping for backbones and equivalences
- equivalent literal substitution (ELS)
- bounded variable elimination (BVE)
- blocked clause elimination (BCE)
- **dynamic sticky clause reduction**
- exponential moving average based restart scheduling
- delaying restarts
- trail reuse

Splatz @ POS'15

Code only

Restarts (future)

Strengthening

Change CDCL

How much is missing?

Features (I) 10

- arena based memory allocation for clauses and watchers
- blocking literals (BLIT)
- special handling of binary clause watches
- literal-move-to-front watch replacement (LMTF)
- learned clause minimization with poison
- on-the-fly hyper-binary resolution (HBR)
- learning additional units and binary clauses (multiple UIPs)
- on-the-fly self-subsuming resolution (OTFS)
- decision only clauses (DECO)
- failed literal probing on binary implication graph roots
- eager recent learned clause subsumption

Splatz @ POS'15

Thank you, Norbert & Mate!

Slides by Armin Biere

Features (II) 11

- stamping based VMTF instead of VSIDS
- subsumption for both irredundant and learned clauses
- inprocessing blocked clause decomposition (BCD) enabling ...
- ... inprocessing SAT sweeping for backbones and equivalences
- equivalent literal substitution (ELS)
- bounded variable elimination (BVE)
- blocked clause elimination (BCE)
- **dynamic sticky clause reduction**
- exponential moving average based restart scheduling
- delaying restarts
- trail reuse

Splatz @ POS'15

Code only

Restarts (future)

Strengthening

Change WL

Change CDCL

How much is missing?

Features (I) 10

- arena based memory allocation for clauses and watchers
- blocking literals (BLIT)
- special handling of binary clause watches
- literal-move-to-front watch replacement (LMTF)
- learned clause minimization with poison
- on-the-fly hyper-binary resolution (HBR)
- learning additional units and binary clauses (multiple UIPs)
- on-the-fly self-subsuming resolution (OTFS)
- decision only clauses (DECO)
- failed literal probing on binary implication graph roots
- eager recent learned clause subsumption

Splatz @ POS'15

Thank you, Norbert & Mate!

Slides by Armin Biere

Features (II) 11

- stamping based VMTF instead of VSIDS
- subsumption for both irredundant and learned clauses
- inprocessing blocked clause decomposition (BCD) enabling . . .
- . . . inprocessing SAT sweeping for backbones and equivalences
- equivalent literal substitution (ELS)
- bounded variable elimination (BVE)
- blocked clause elimination (BCE)
- **dynamic sticky clause reduction**
- exponential moving average based restart scheduling
- delaying restarts
- trail reuse

Splatz @ POS'15

Code only

Restarts (future)

Strengthening

Change WL

Change CDCL

TUM

VU VRIJE UNIVERSITEIT AMSTERDAM

max planck institut informatik

How much is missing?

Features (I) 10

- arena based memory allocation for clauses and watchers
- blocking literals (BLIT)
- special handling of binary clause watches
- literal-move-to-front watch replacement (LMTF)
- learned clause minimization with poison
- on-the-fly hyper-binary resolution (HBR)
- learning additional units and binary clauses (multiple UIPs)
- on-the-fly self-subsuming resolution (OTFS)
- decision only clauses (DECO)
- failed literal probing on binary implication graph roots
- eager recent learned clause subsumption

Splatz @ POS'15

Thank you, Norbert & Mate!

Slides by Armin Biere

Features (II) 11

- stamping based VMTF instead of VSIDS

- Unchecked array accesses (Isabelle takes care of it)
- No unbounded integers (in theory, not complete anymore)
- Restarts

- exponential moving average based restart scheduling
- delaying restarts
- trail reuse

Splatz @ POS'15

Code only

Restarts (future)

Strengthening

Change WL

Change CDCL

Update Strategy

A first idea

M

Clauses N

A

B

C

A better strategy

CBAM

Clauses N

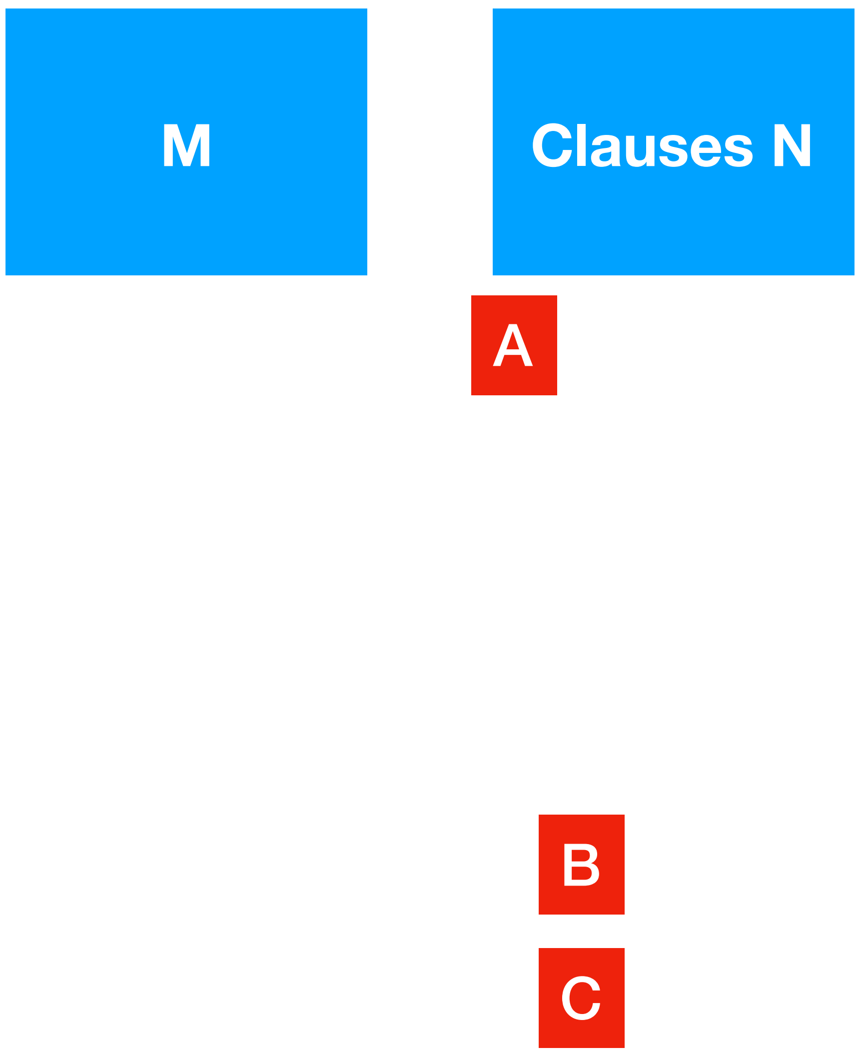
A

B

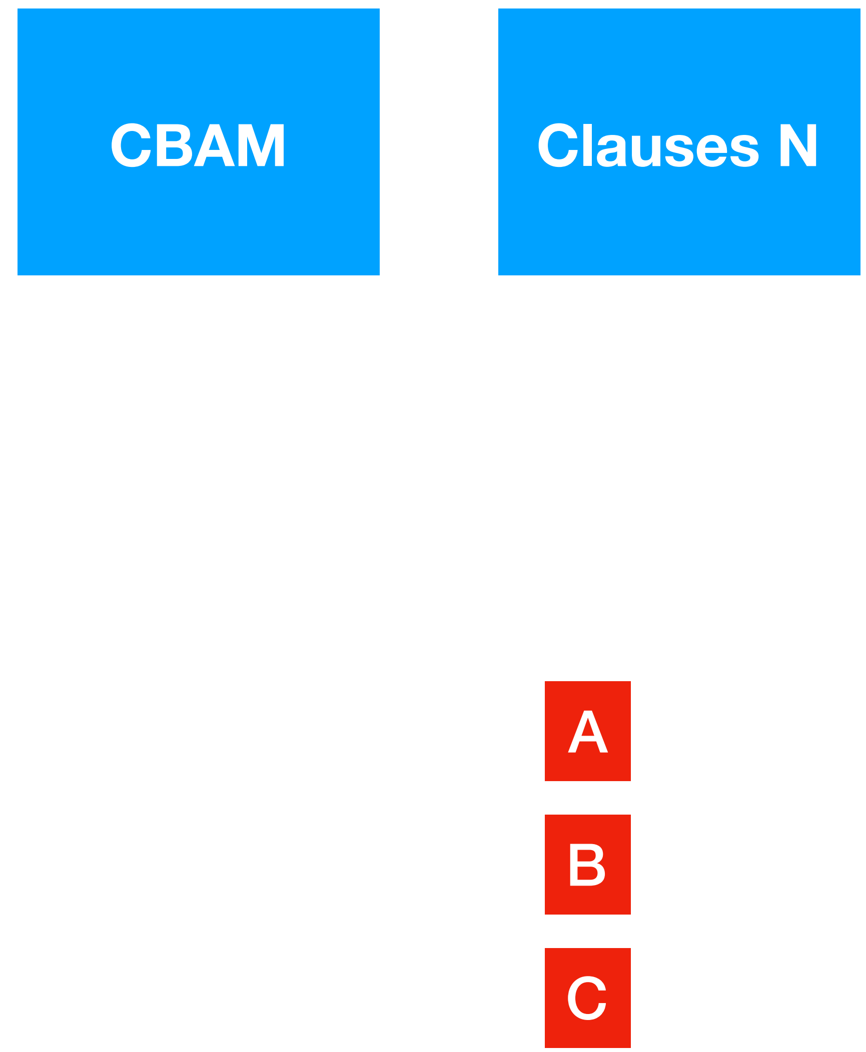
C

Update Strategy

A first idea

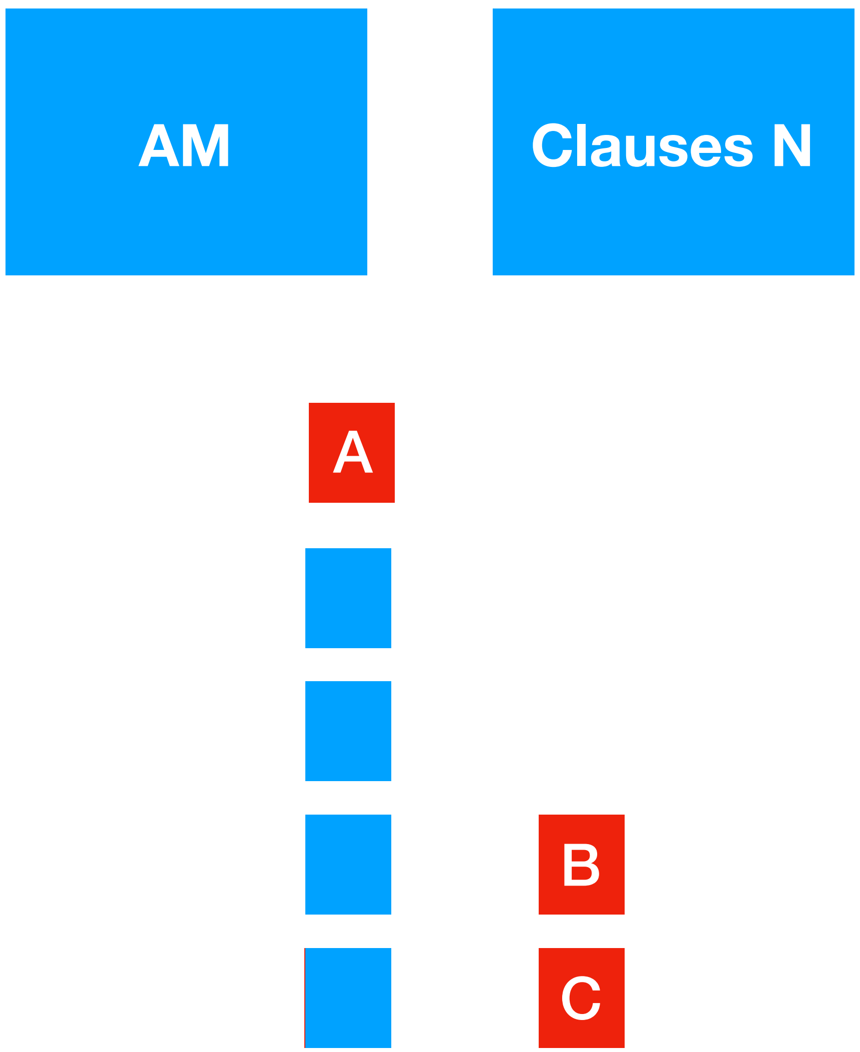


A better strategy

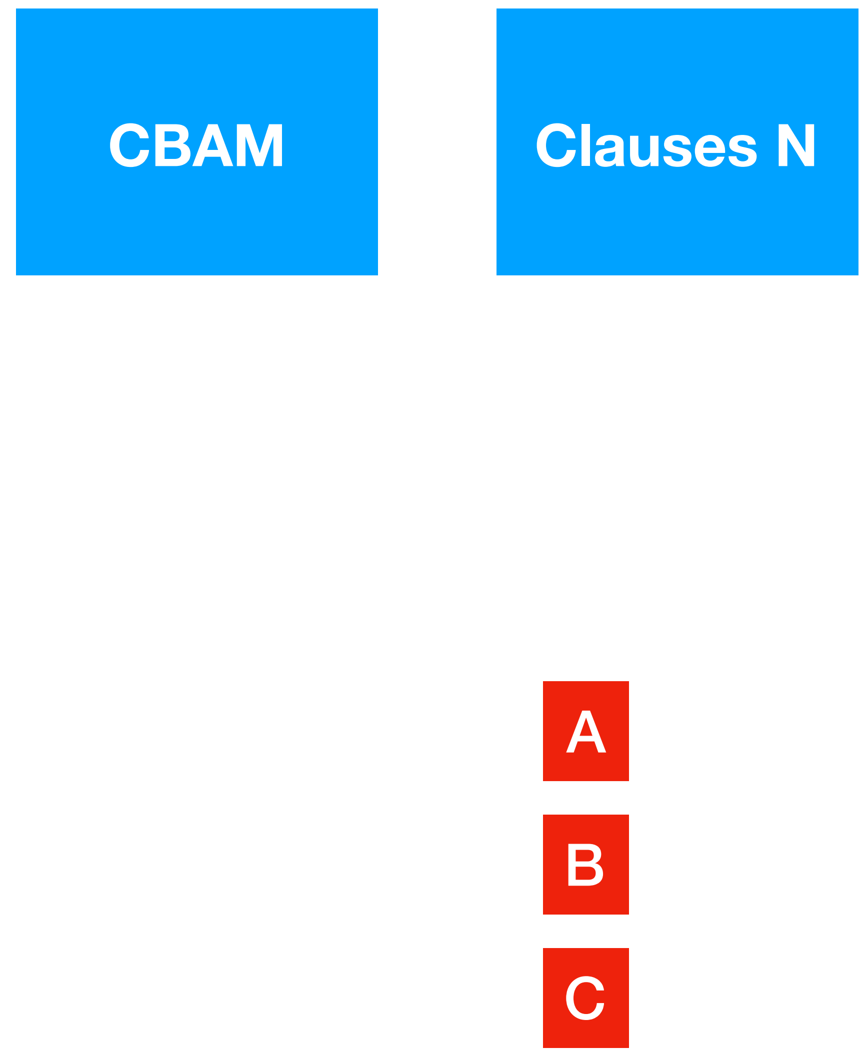


Update Strategy

A first idea

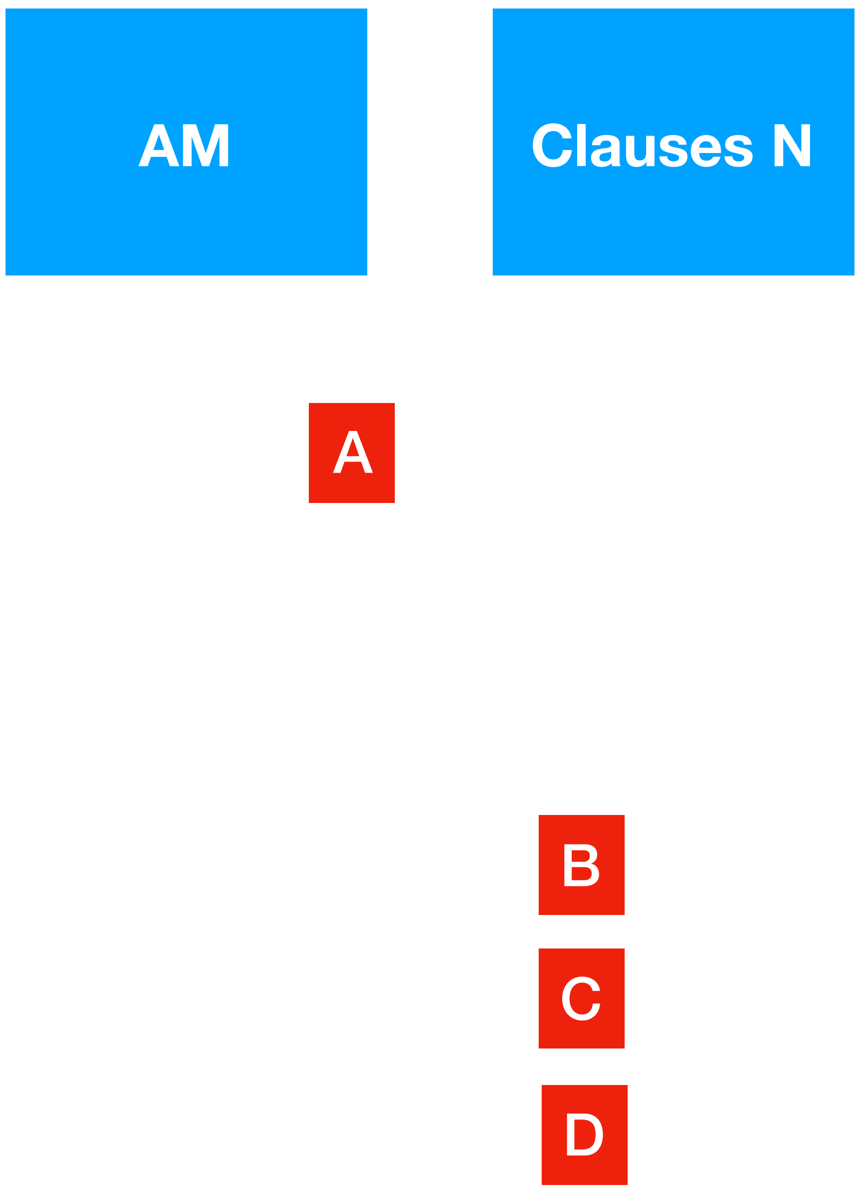


A better strategy

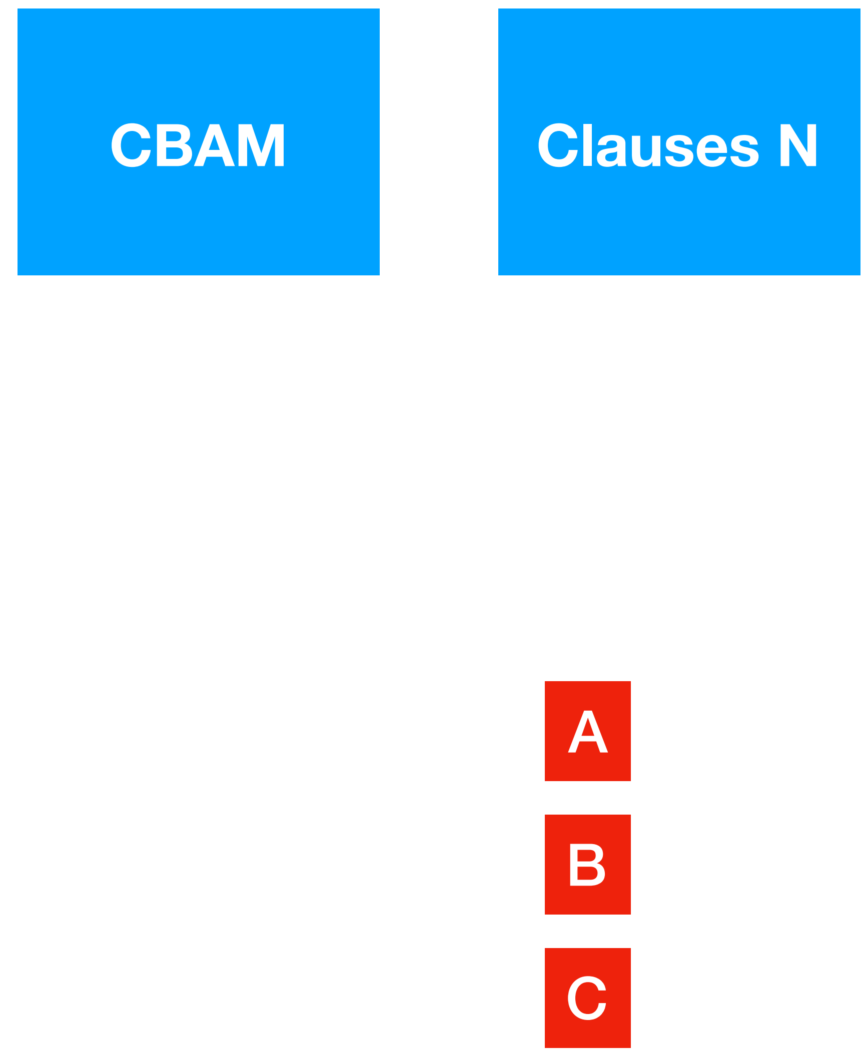


Update Strategy

A first idea



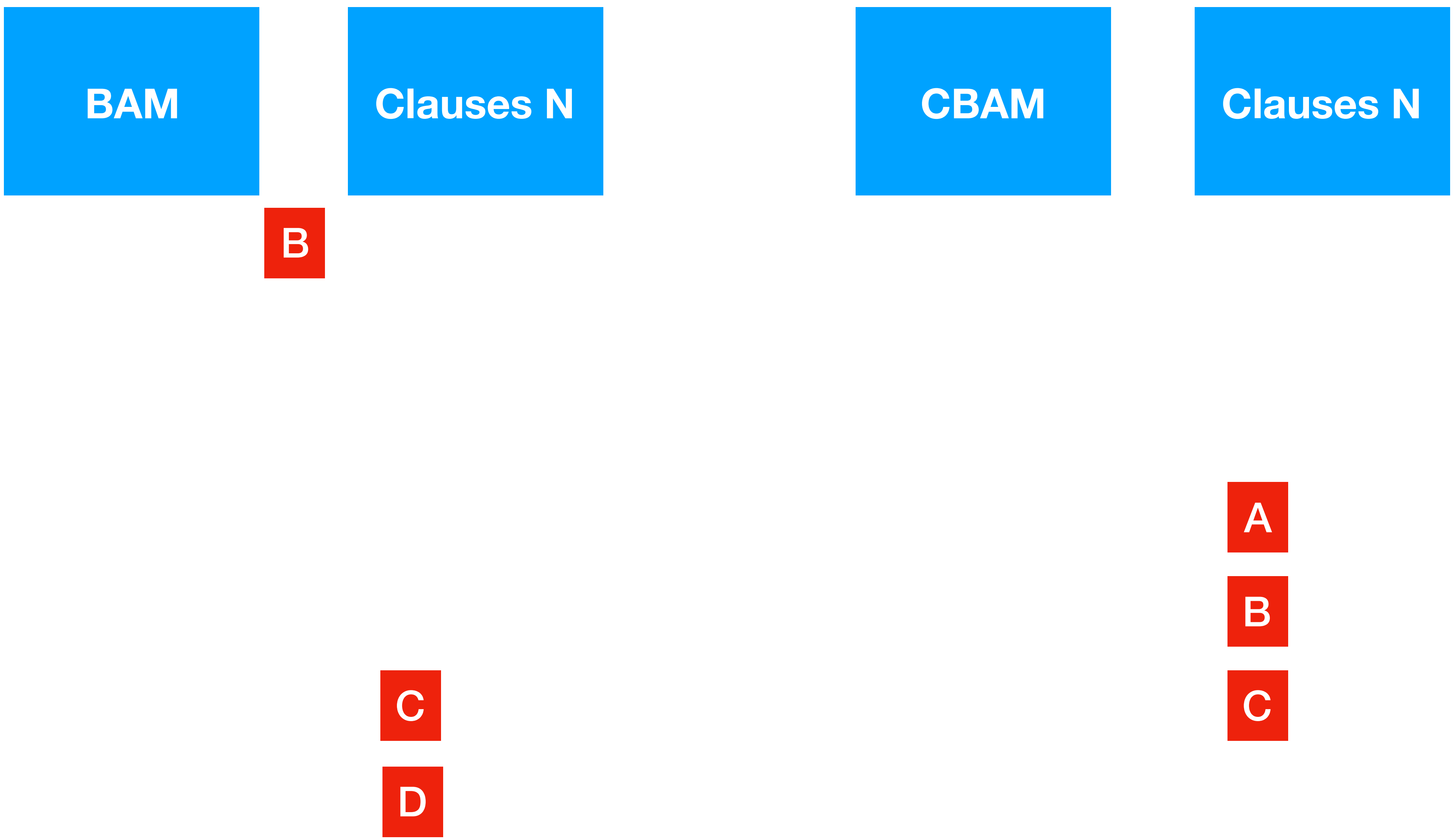
A better strategy



Update Strategy

A first idea

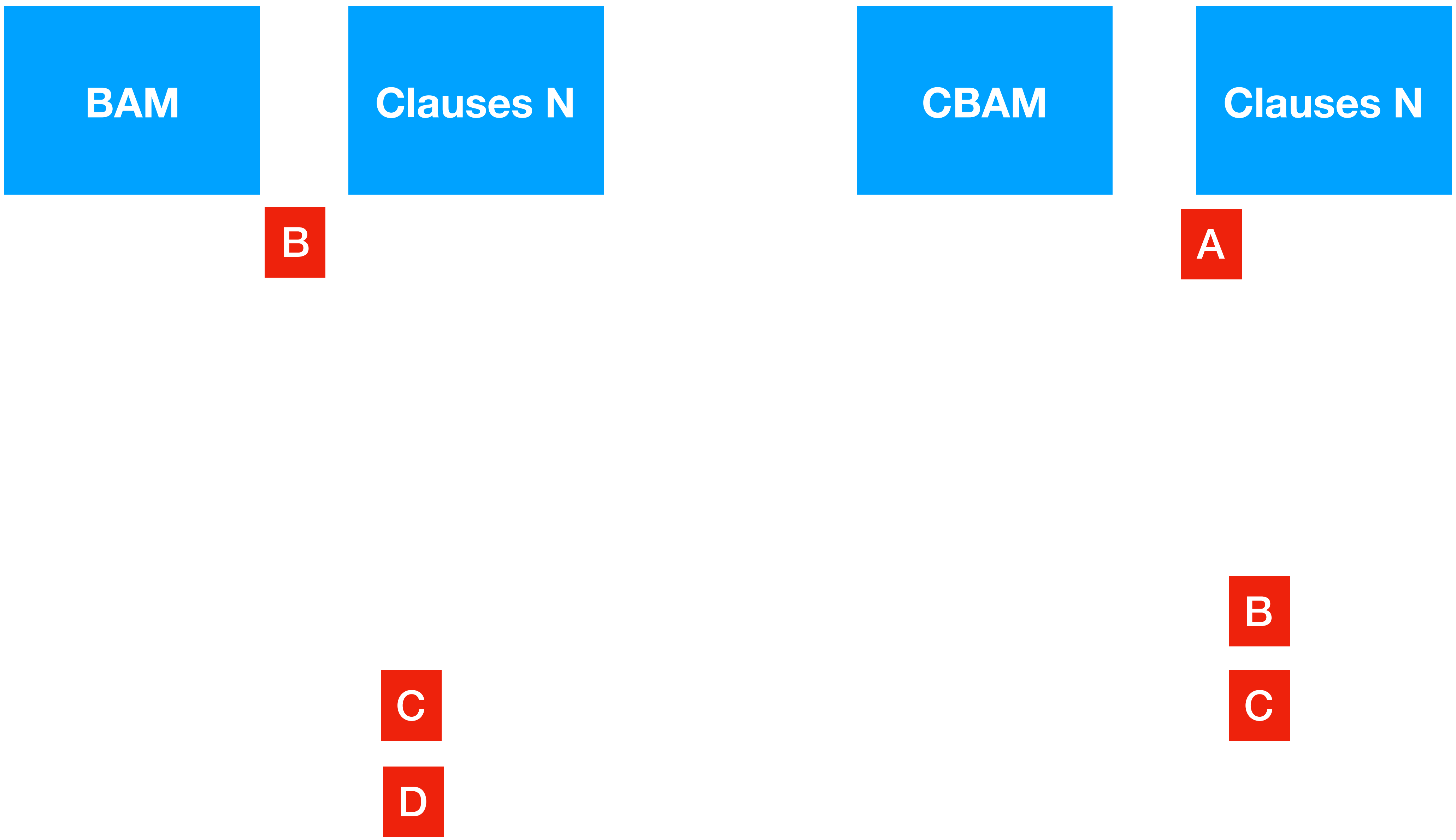
A better strategy



Update Strategy

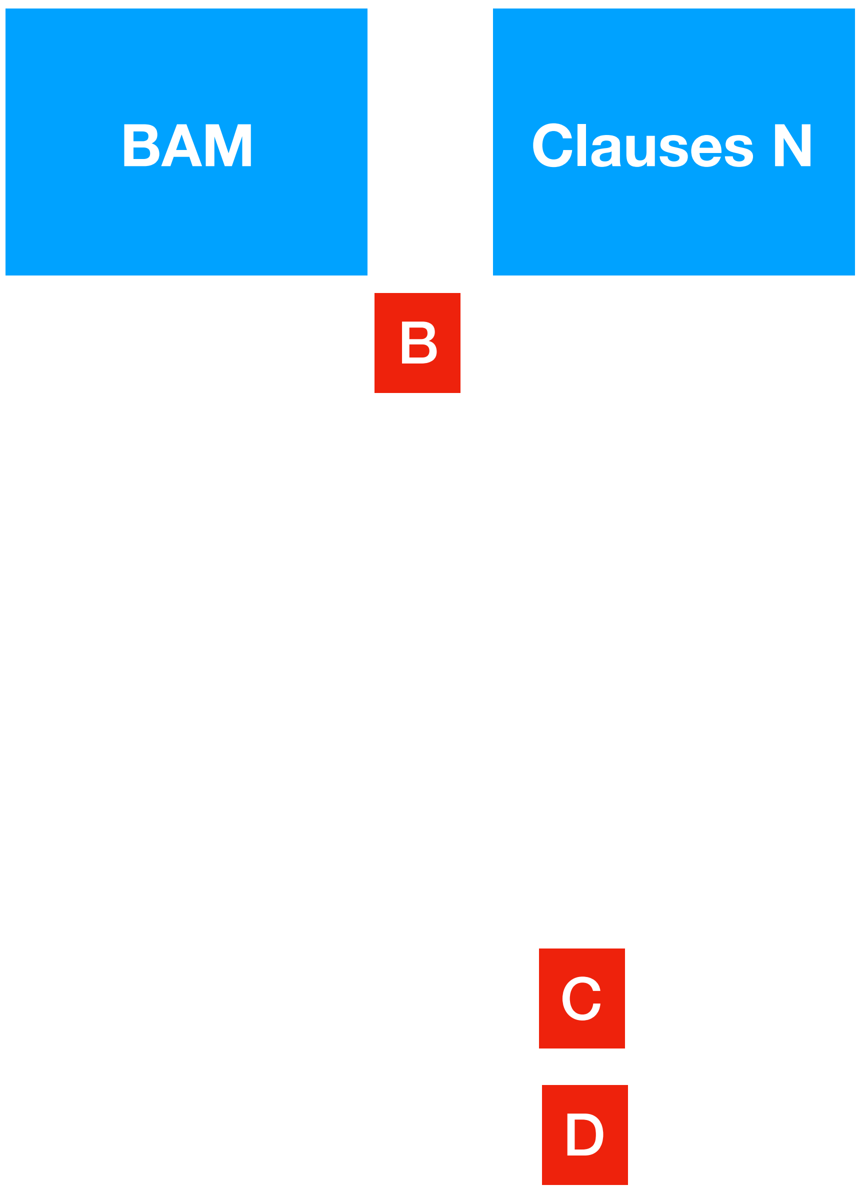
A first idea

A better strategy

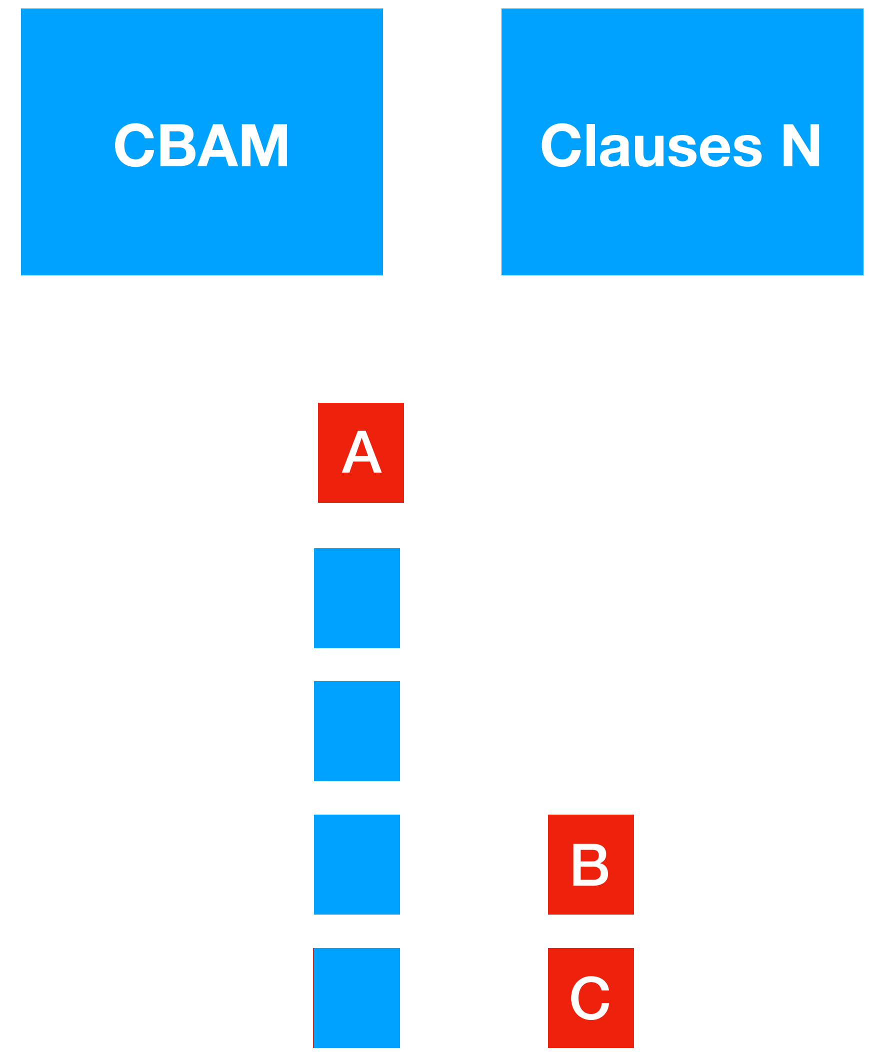


Update Strategy

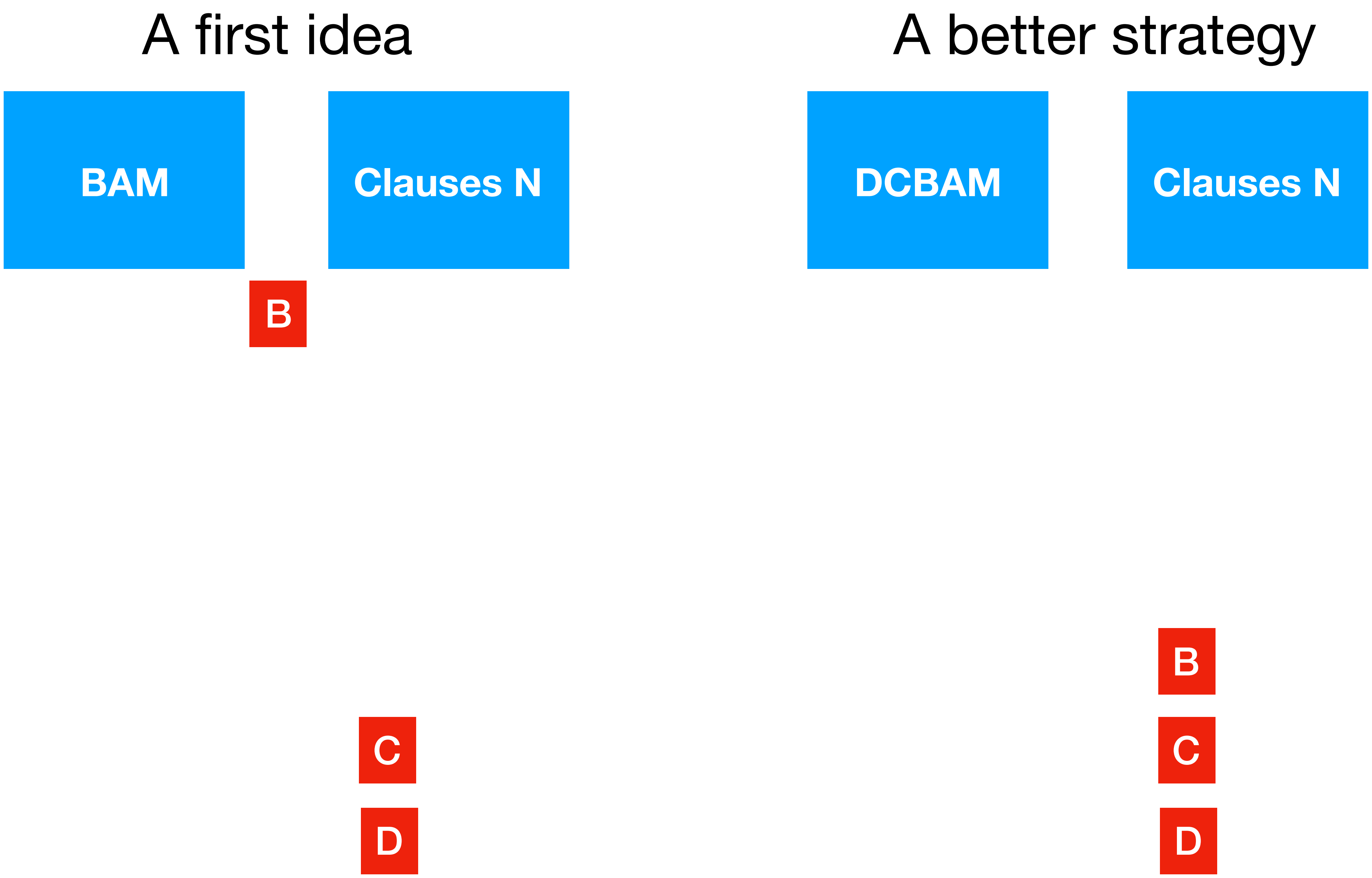
A first idea



A better strategy

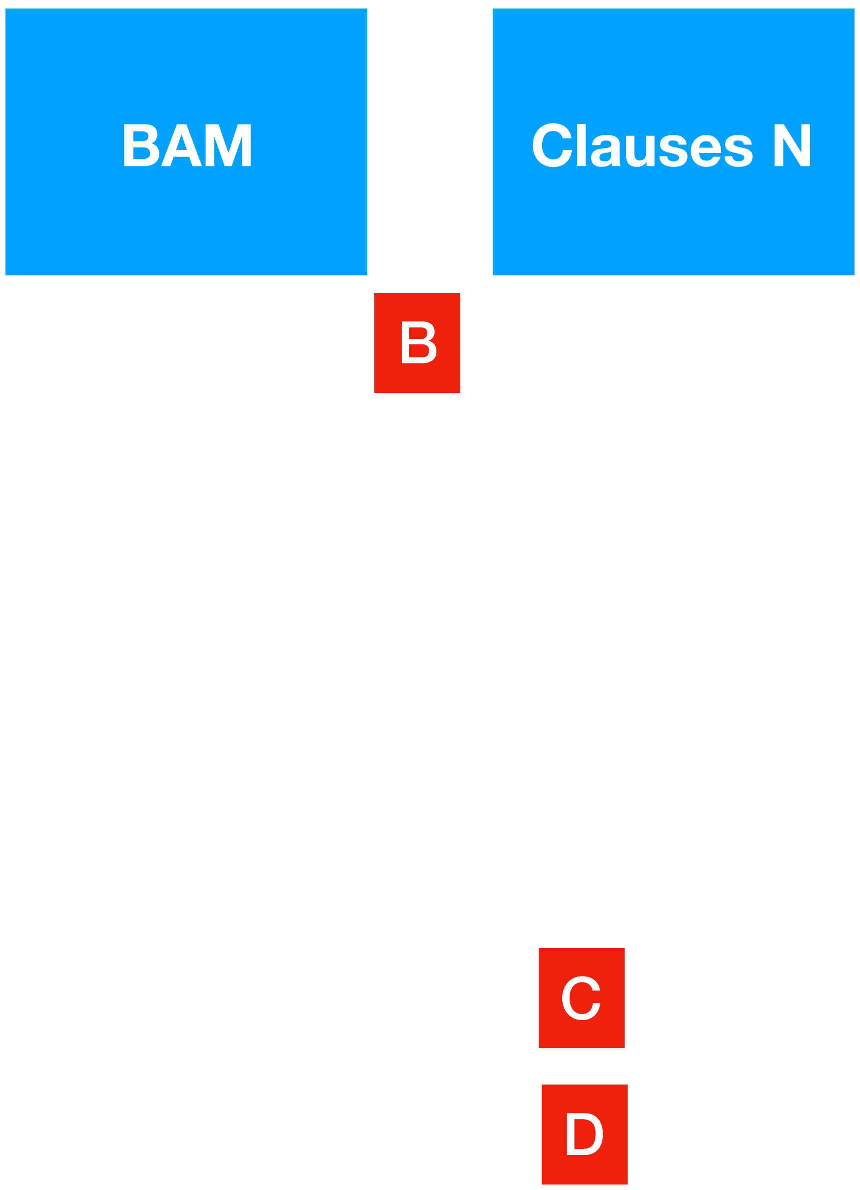


Update Strategy

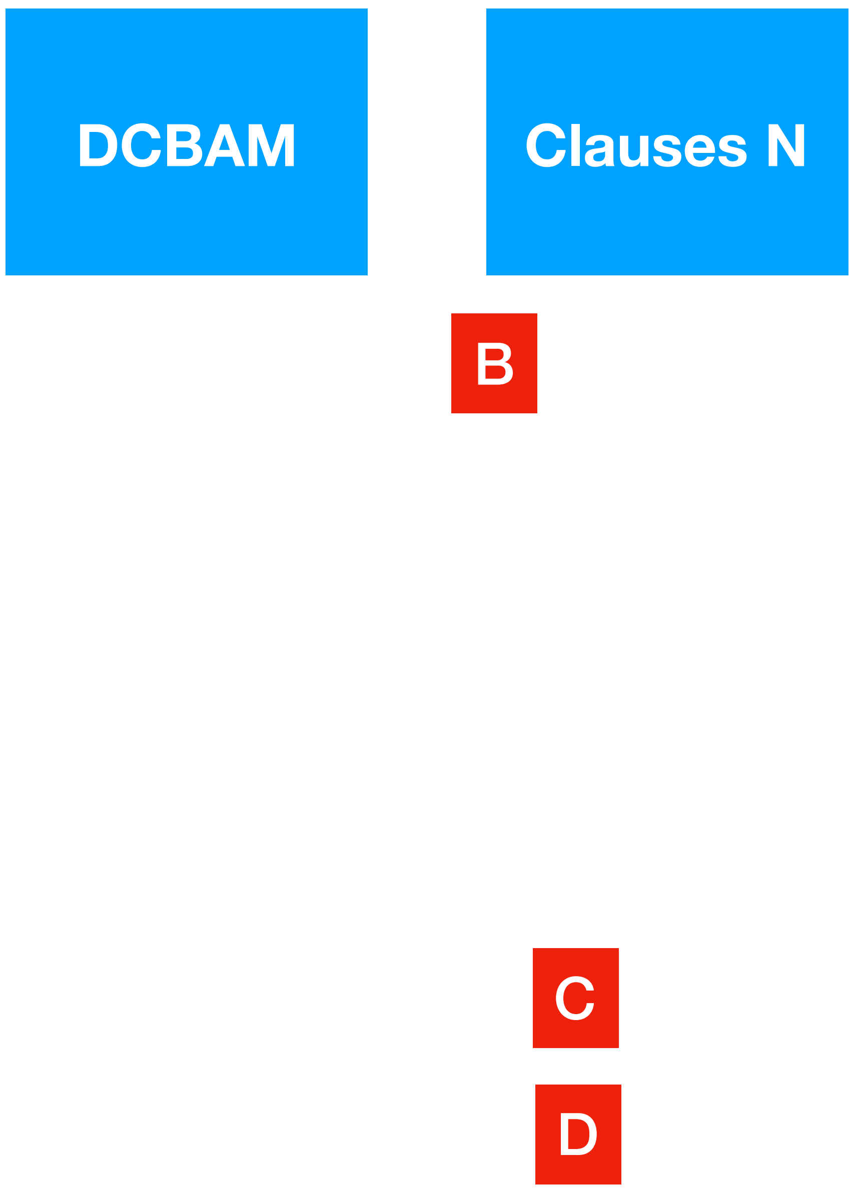


Update Strategy

A first idea



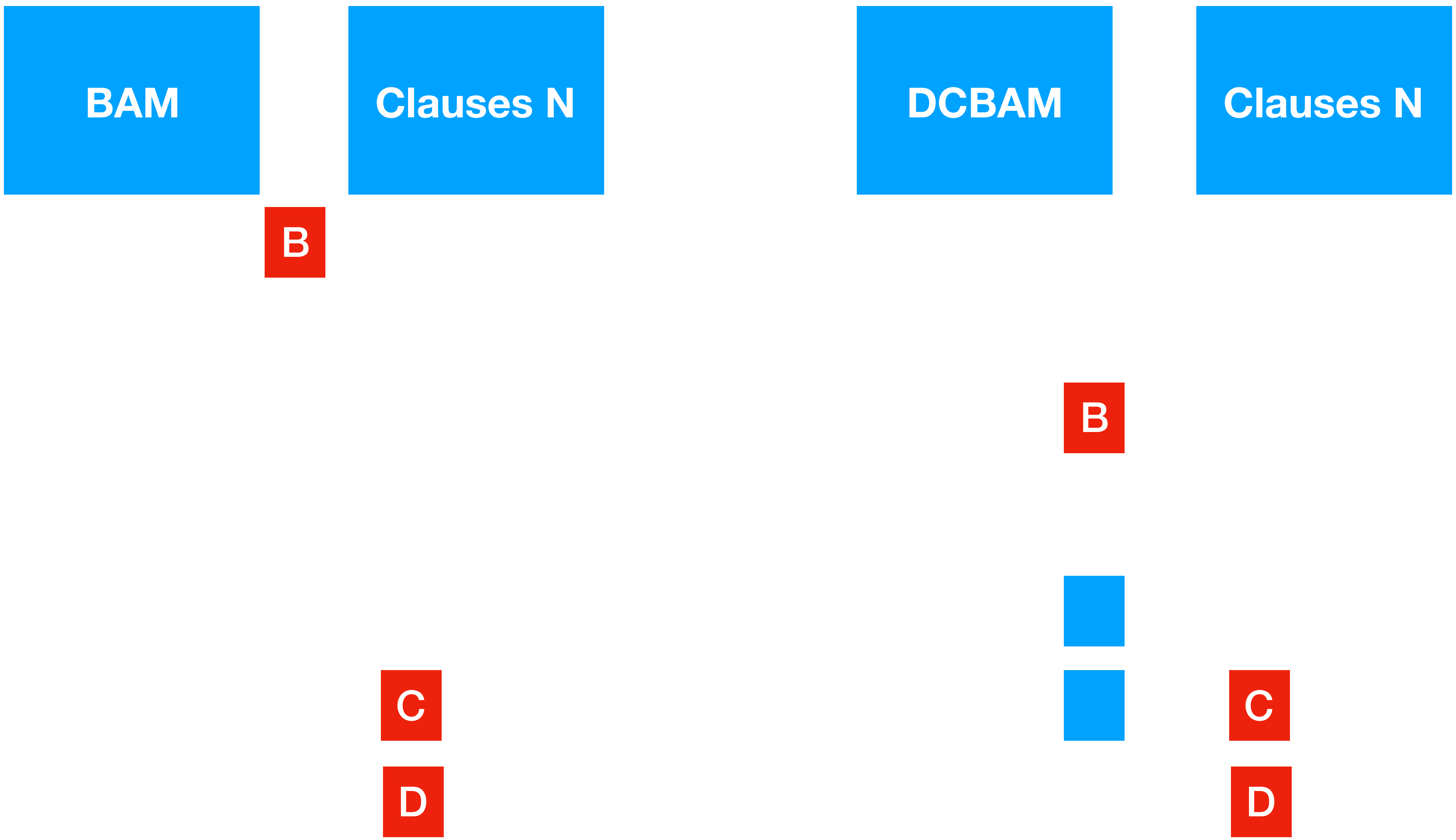
A better strategy



Update Strategy

A first idea

A better strategy



Update Strategy

A first idea

A better strategy