Inria
INVENTORS FOR THE DIGITAL WORLD

max planck institut
informatik

SAARLAND UNIVERSITY

SAARBRÜCKEN
GRADUATE SCHOOL OF
COMPUTER SCIENCE

Saarland
Informatics Campus

# A Verified SAT Solver with Two Watched Literals

**Jasmin Blanchette** Mathias Fleury Peter Lammich Christoph Weidenbach

VU VRIJE UNIVERSITEIT AMSTERDAM

Technische Universität München

# SAT solving

Given a CNF formula

$$\varphi = \bigwedge_i \bigvee_j L_{i,j}$$

is there a satisfying assignment?

Most used algorithm: CDCL, an improvement over DPLL

# 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

|  | Correctness | Applicability |
|---|---|---|
| Run of a SAT solver | Certificate: proof of (un)satisfiability | *a given* input |
| Theory behind SAT solvers | Proof | *every* input |

# IsaFoL project

Isabelle Formalization of Logic

TUM

VU VRIJE UNIVERSITEIT AMSTERDAM

max planck institut informatik

# Selected IsaFoL entries

- FO resolution
  by Schlichtkrull (ITP 2016)

- CDCL with learn, forget, restart, incrementality, 2WL
  by Blanchette, Fleury, Lammich, Weidenbach (IJCAR 2016, now)

- GRAT certificate checker
  by Lammich (CADE 2017)

- FO ordered resolution with selection
  by Schlichtkrull, Blanchette, Traytel, Waldmann (IJCAR 2018?)

# Selected IsaFoL entries

- FO resolution
  by Schlichtkrull (ITP 2016)

- CDCL with learn, forget, restart, incrementality, 2WL
  by Blanchette, Fleury, Lammich, Weidenbach (IJCAR 2016, now)

- GRAT certificate checker
  by Lammich (CADE 2017)

- FO ordered resolution with selection
  by Schlichtkrull, Blanchette, Traytel, Waldmann (IJCAR 2018?)

# Why?

- Eat our own dog food

  case study for proof assistants and automatic provers

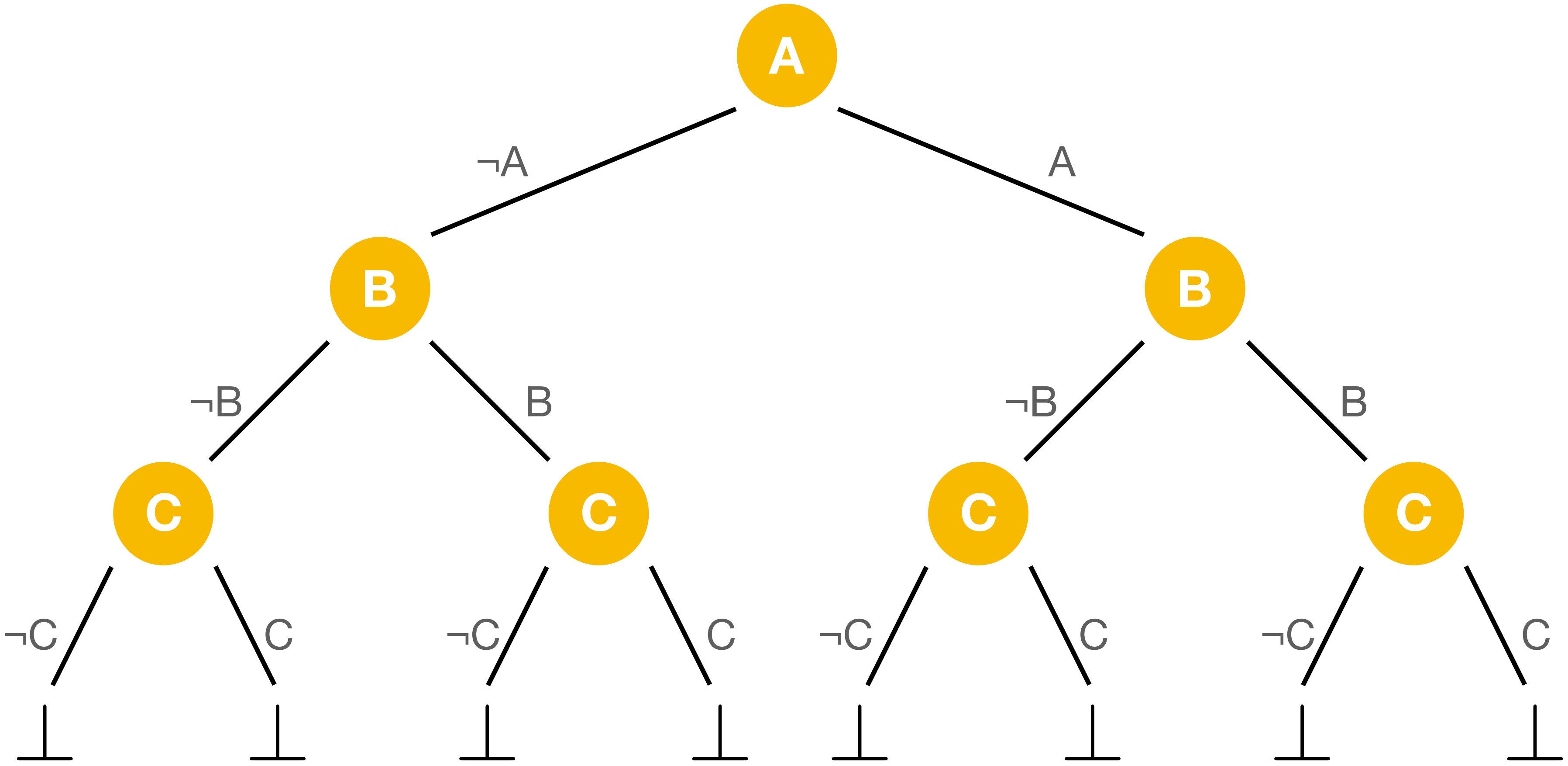
- Build libraries for state-of-the-art research

  *Automated Reasoning:*
  *The Art of Generic Problem Solving*
  (forthcoming textbook by Weidenbach)

# Truth table

**N =** $A \lor B \lor C \quad \neg A \lor B \lor C \quad \neg B \lor C \quad B \lor \neg C$
$\neg A \lor B \quad A \lor \neg B \lor \neg C \quad A \lor \neg C$

**Decide**



# DPLL

**N =** $A \lor B \lor C \quad \neg A \lor B \lor C \quad \neg B \lor C \quad B \lor \neg C$

$\neg A \lor B \quad A \lor \neg B \lor \neg C \quad A \lor \neg C$

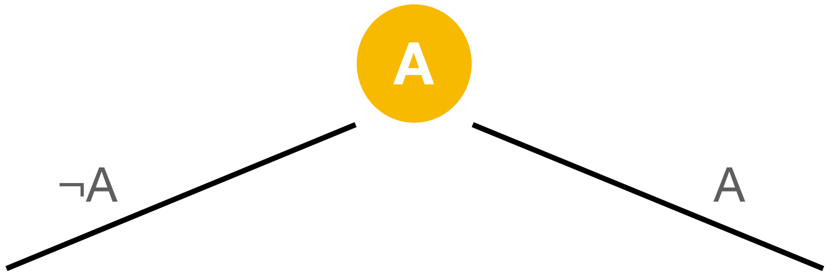
**Decide**

**Propagate**

# DPLL

**N =** $A \lor B \lor C \quad \neg A \lor B \lor C \quad \neg B \lor C \quad B \lor \neg C$

$\neg A \lor B \quad A \lor \neg B \lor \neg C \quad A \lor \neg C$



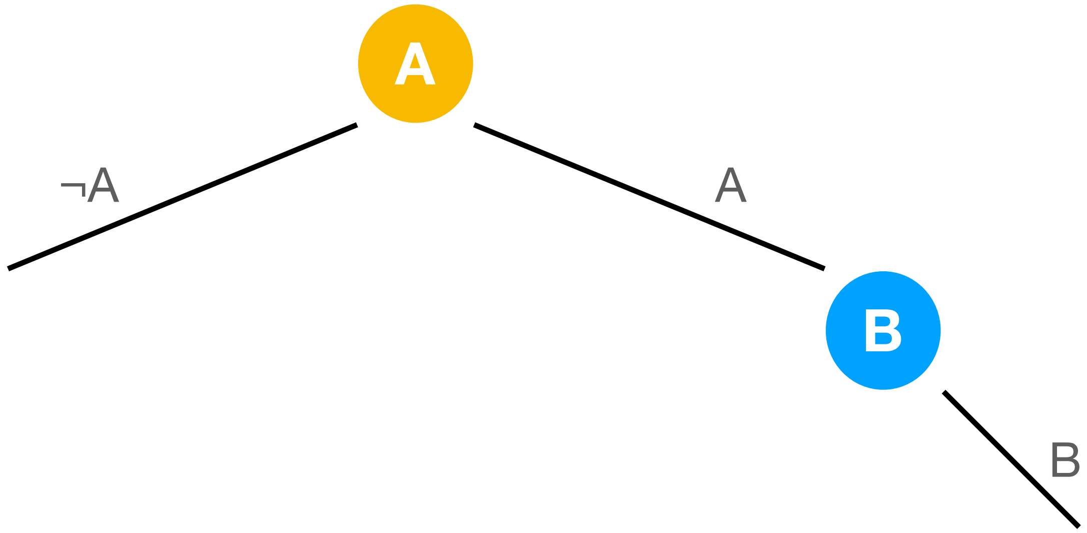
Decide

Propagate

# DPLL

**N =** $A \vee B \vee C \quad \neg A \vee B \vee C \quad \neg B \vee C \quad B \vee \neg C$

$\neg A \vee B \quad A \vee \neg B \vee \neg C \quad A \vee \neg C$



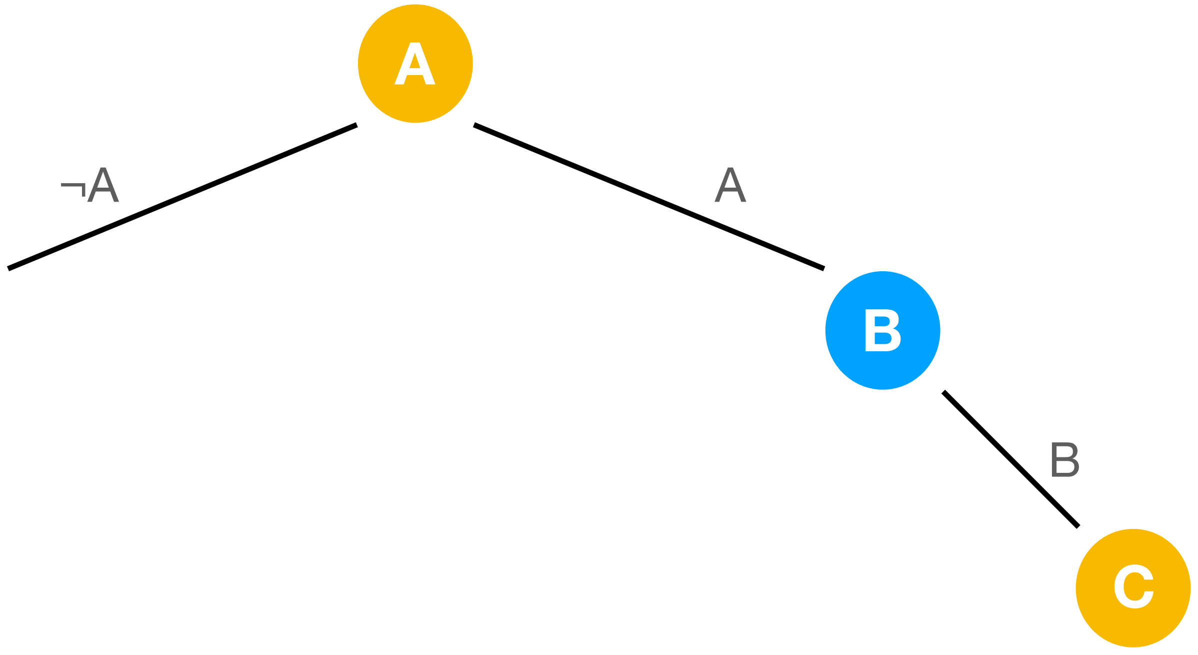
Decide

Propagate

# DPLL

**N =** $A \vee B \vee C \quad \neg A \vee B \vee C \quad \neg B \vee C \quad B \vee \neg C$

$\neg A \vee B \quad A \vee \neg B \vee \neg C \quad A \vee \neg C$



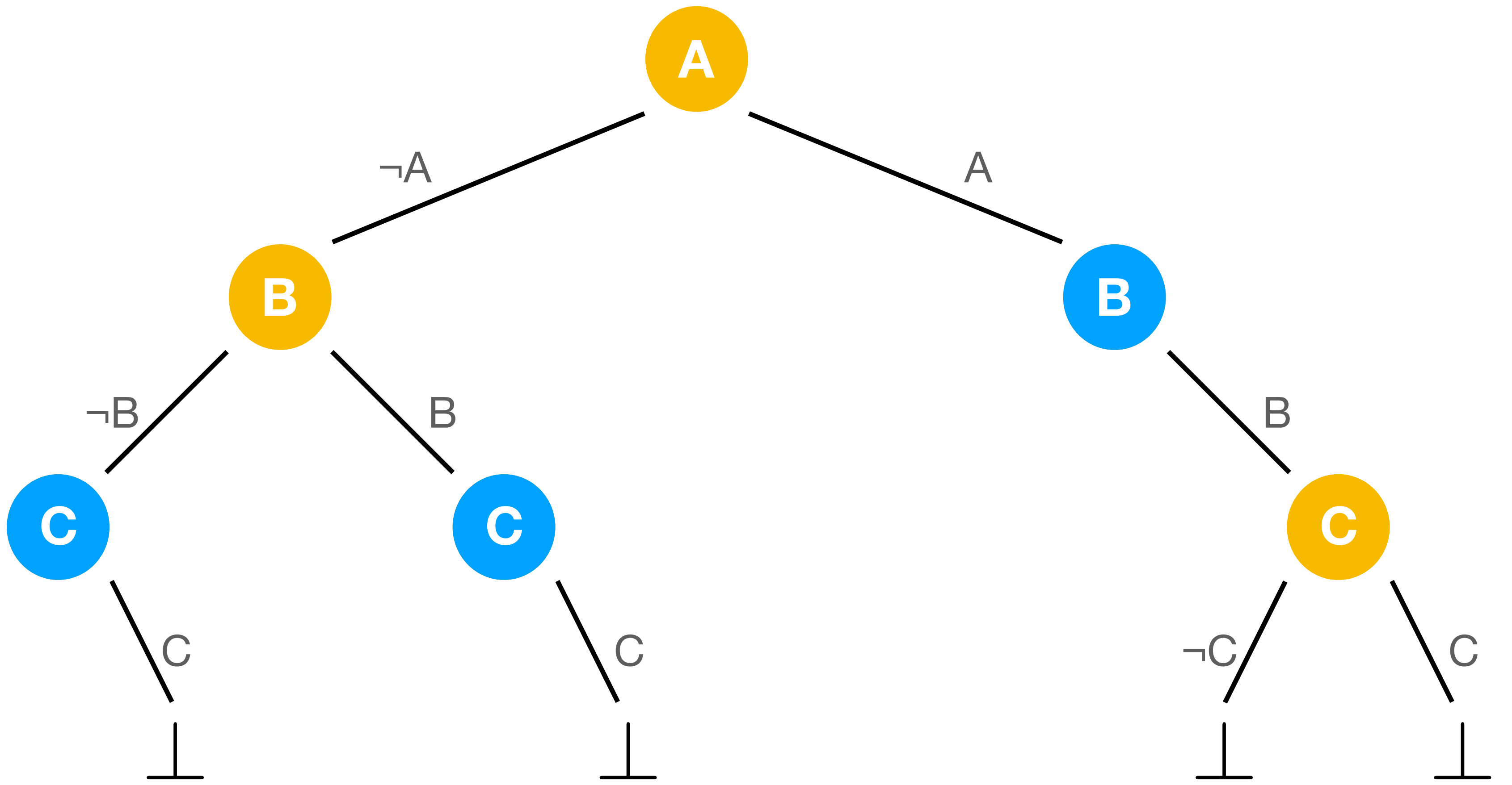
Decide

Propagate

# DPLL

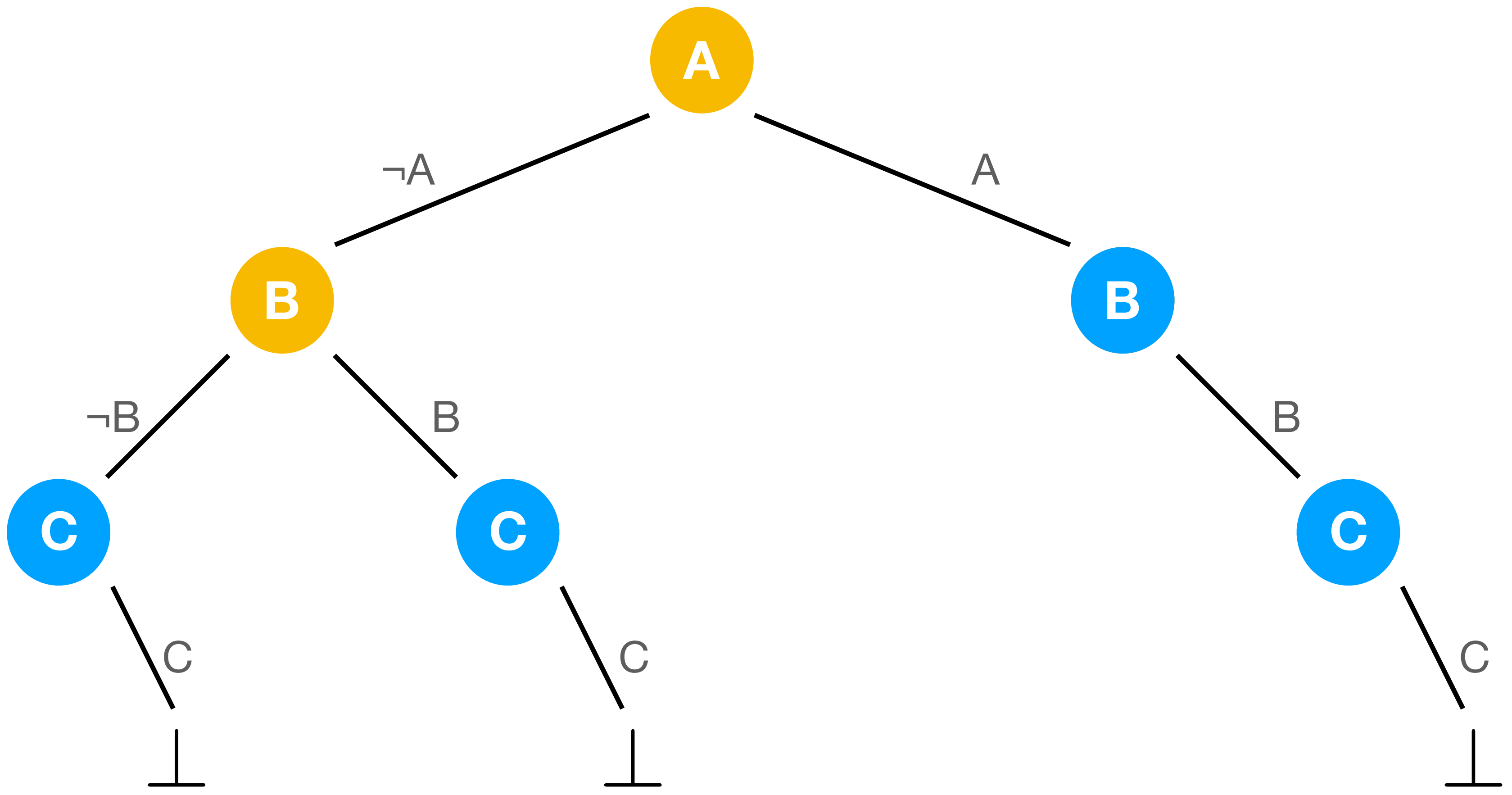
**N =** $A \vee B \vee C \quad \neg A \vee B \vee C \quad \neg B \vee C \quad B \vee \neg C$

$\neg A \vee B \quad A \vee \neg B \vee \neg C \quad A \vee \neg C$



Decide

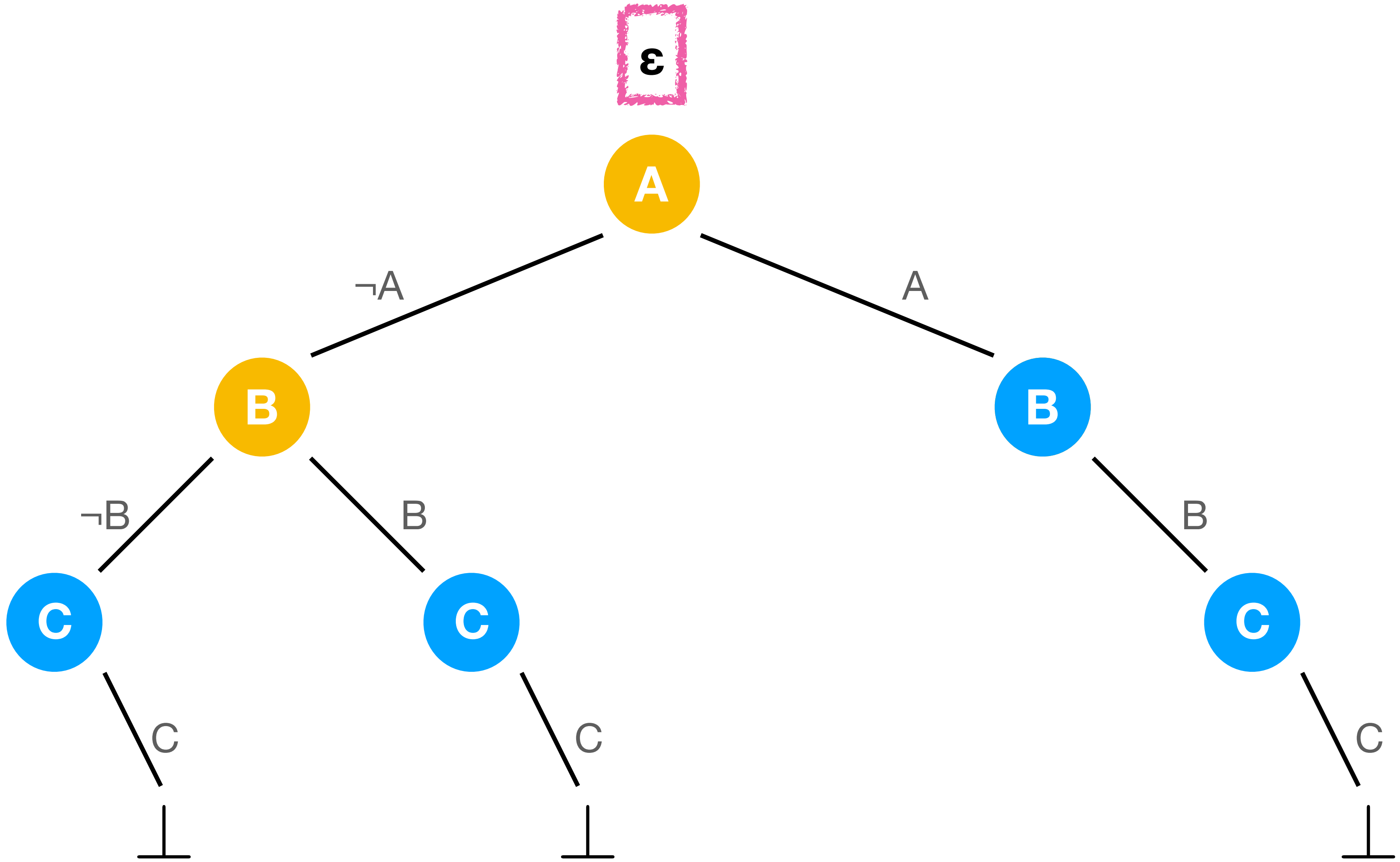
Propagate



State (trail)

Decide

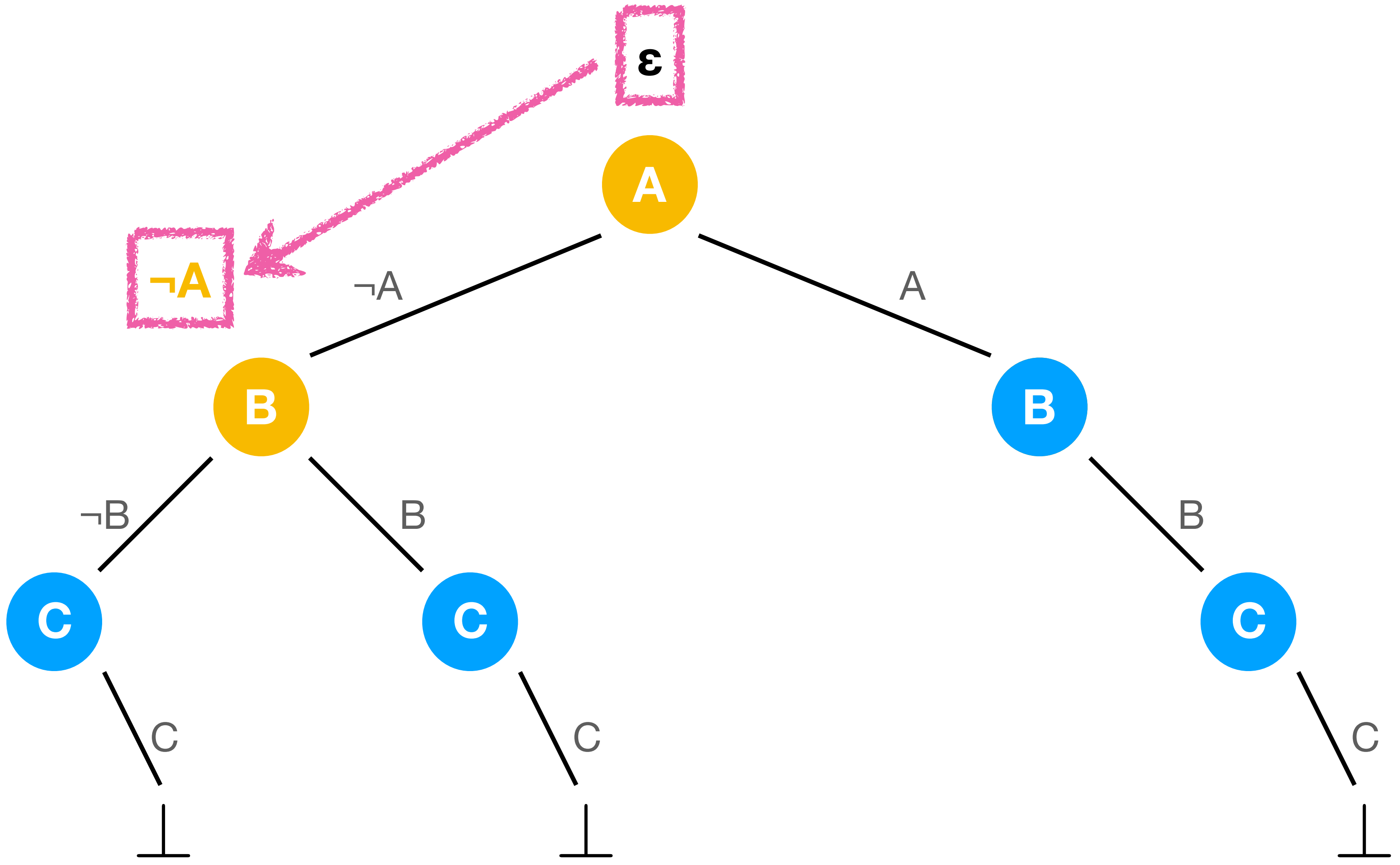
Propagate



State (trail)

Decide

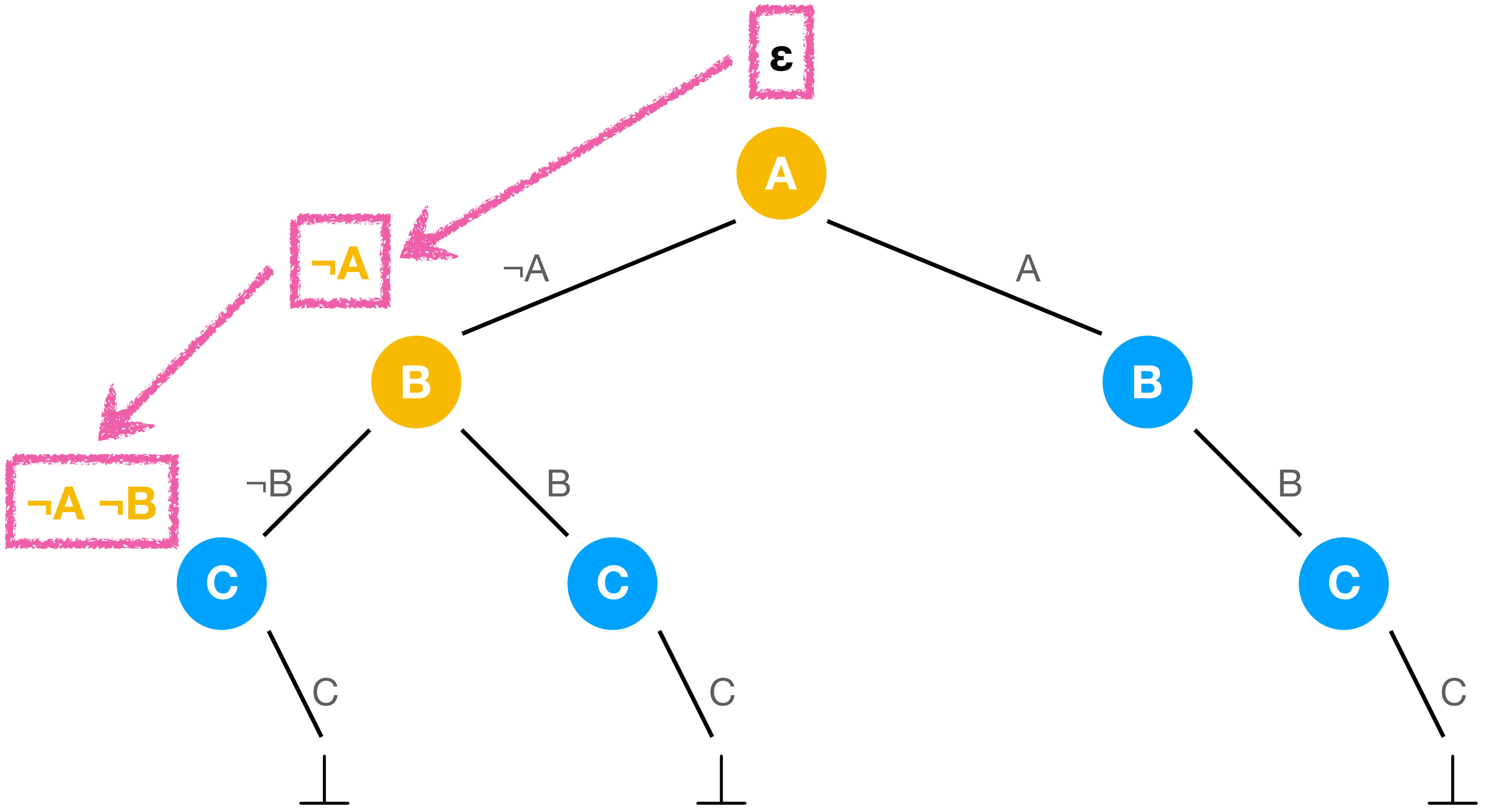
Propagate



State (trail)

Decide

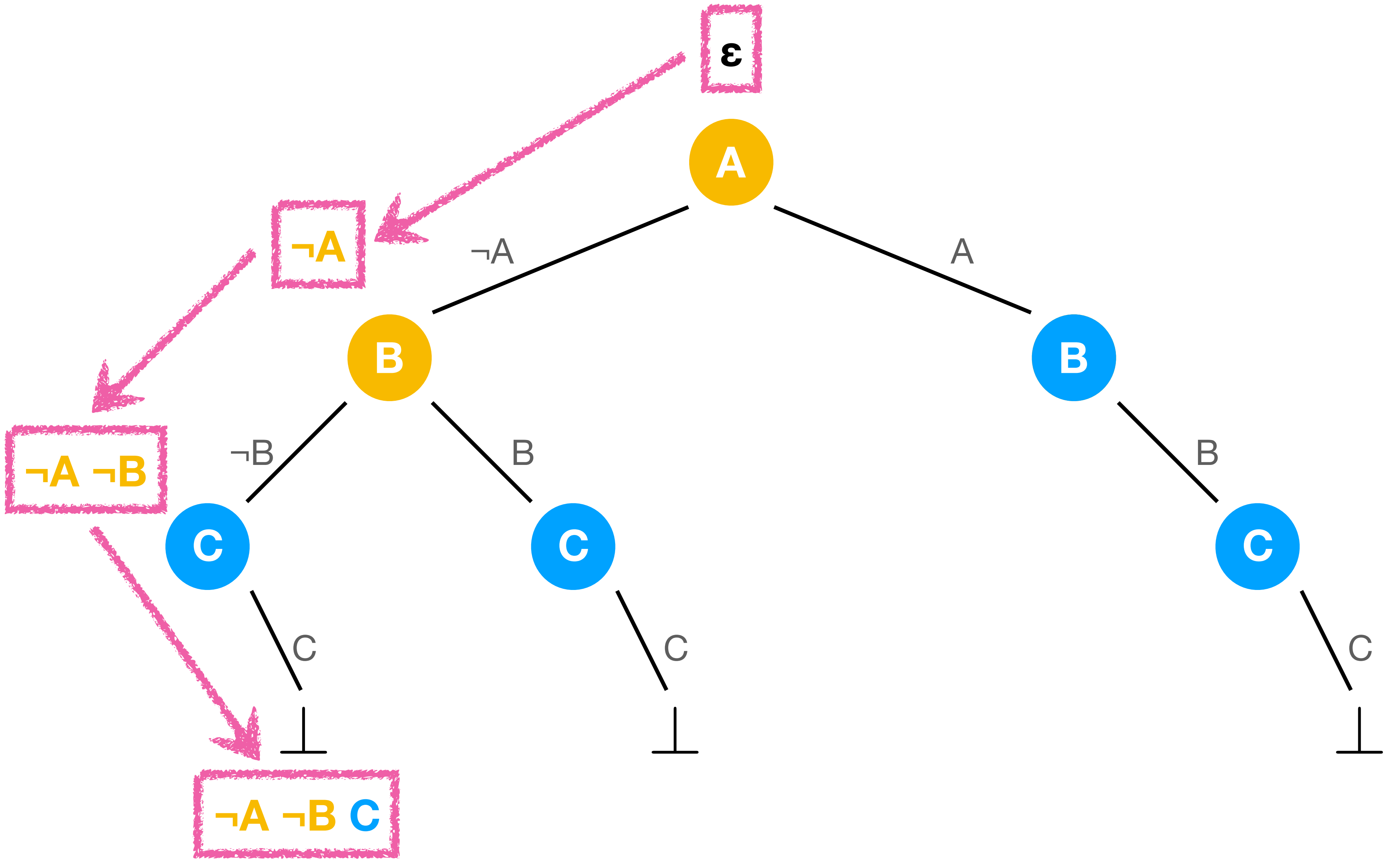
Propagate



State (trail)

Decide

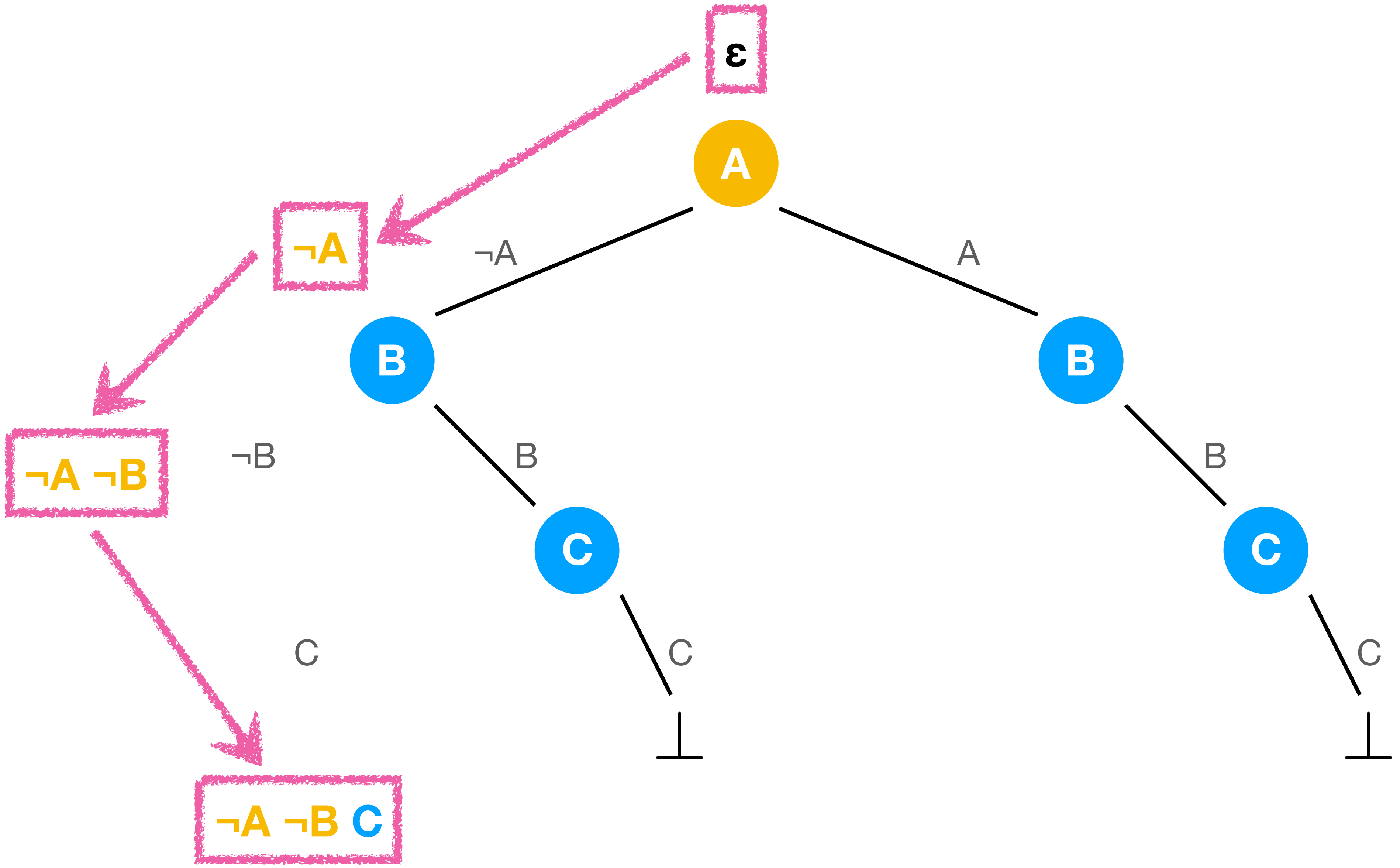
Propagate



State (trail)

Decide

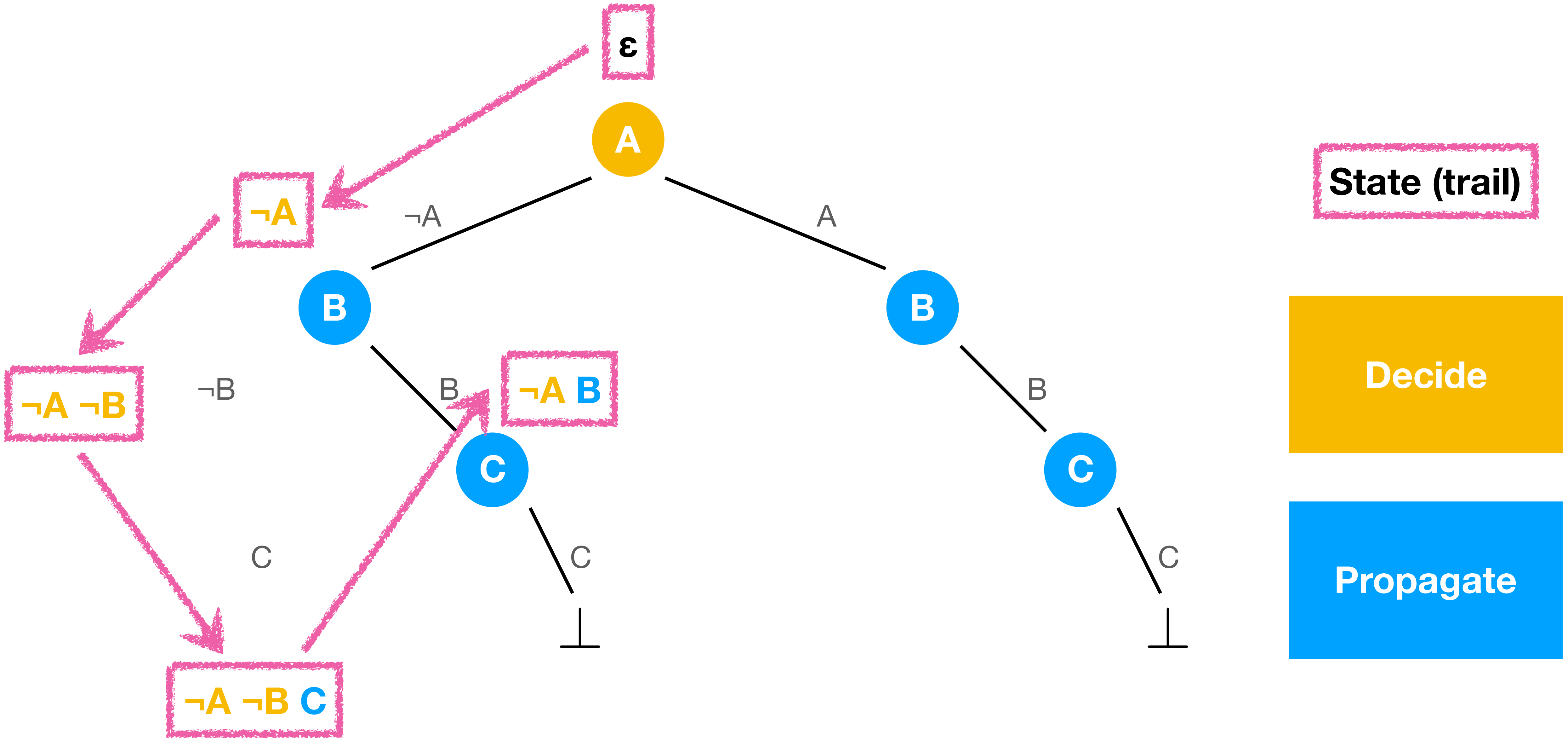
Propagate

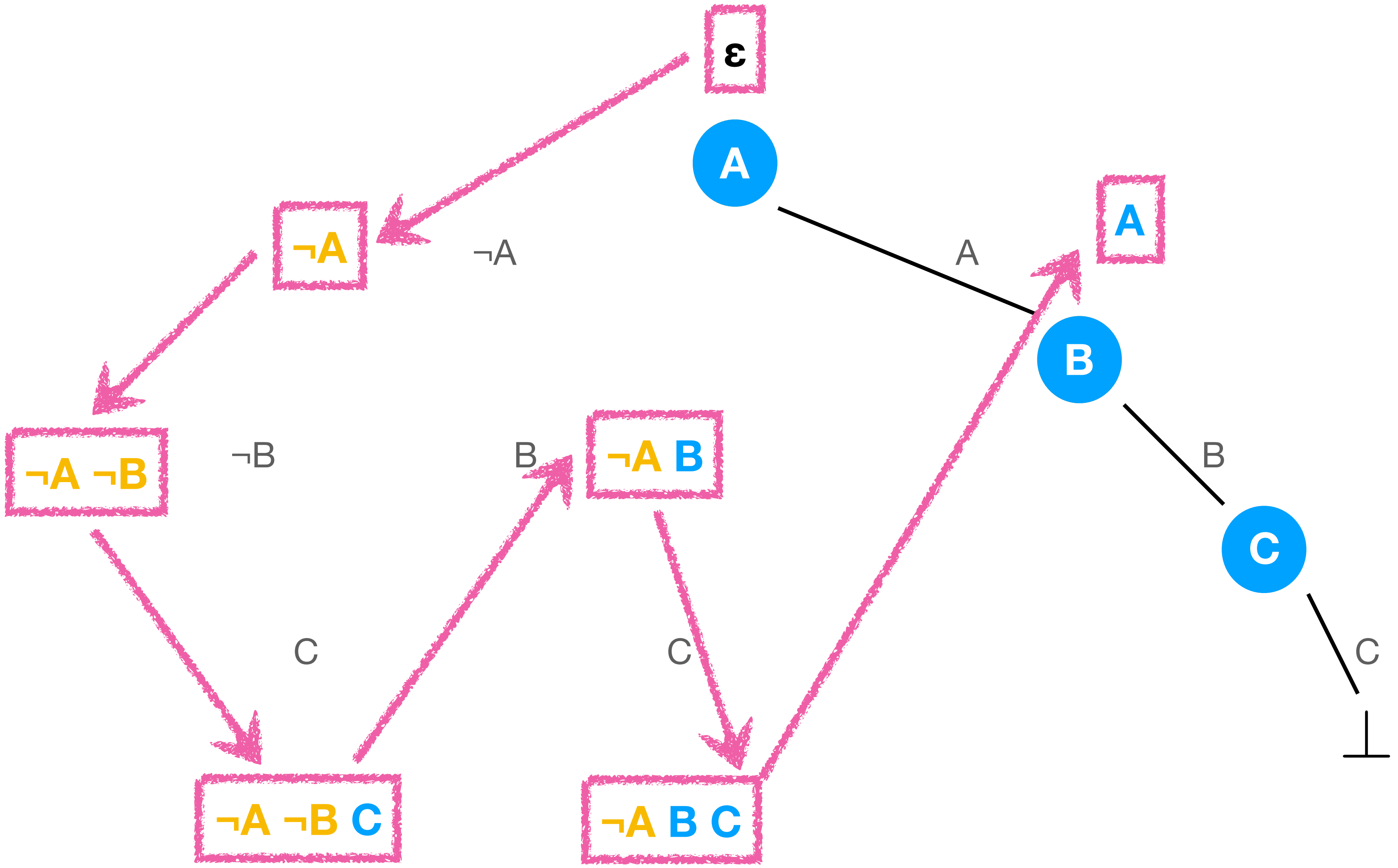


State (trail)

Decide

Propagate


ε
A
¬A
A
¬A
B
B
¬A ¬B
¬B
B
¬A B
B
C
C
C
C
C
¬A ¬B C
⊥
⊥
State (trail)
Decide
Propagate



ε

A

¬A

¬A

A

A

B

¬A ¬B

¬B

B

¬A B

B

C

C

C

C

¬A ¬B C

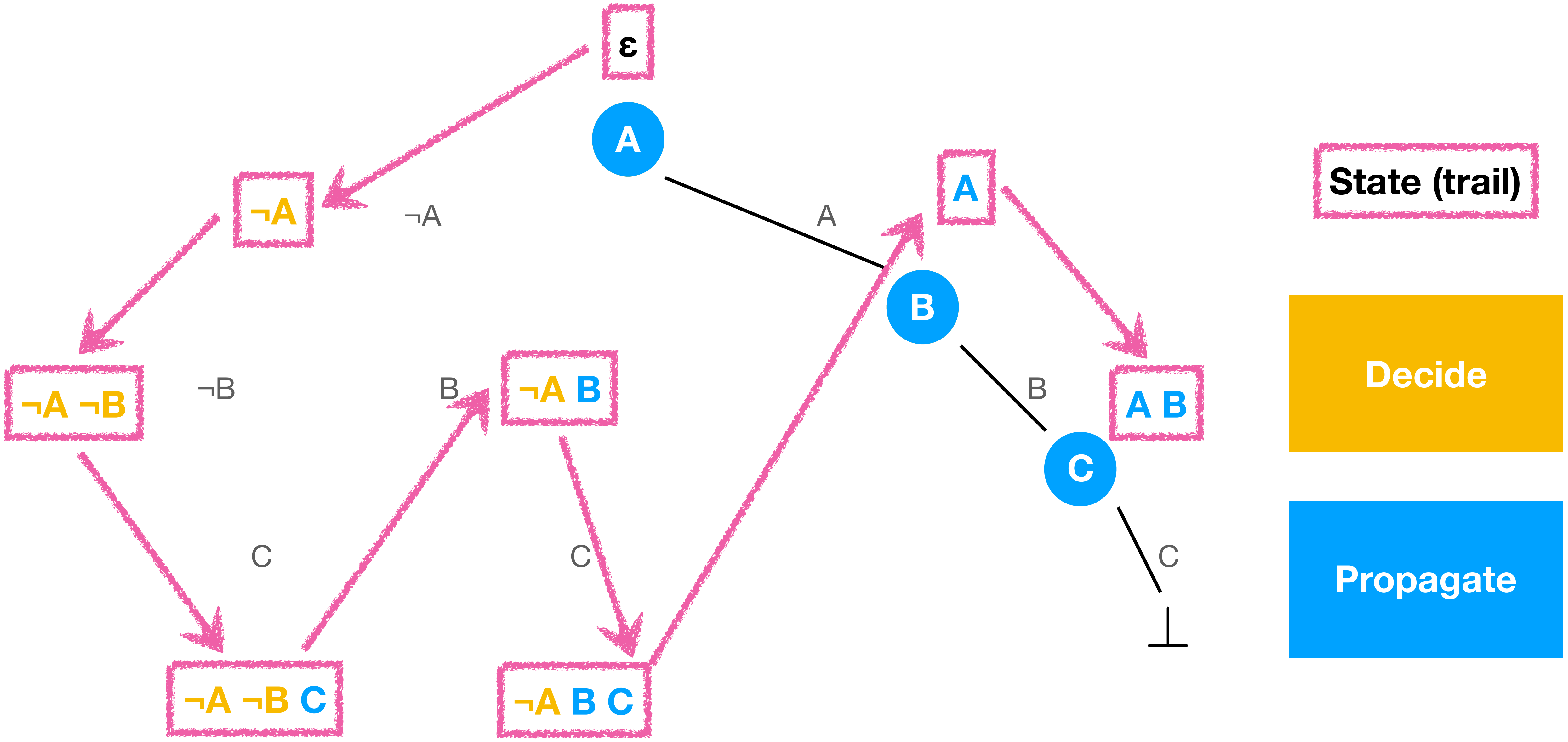
¬A B C

⊥

State (trail)

Decide

Propagate



ε

A

¬A

A

¬A ¬B

¬B

B

¬A B

B

B

A

A B

C

C

C

C

¬A ¬B C

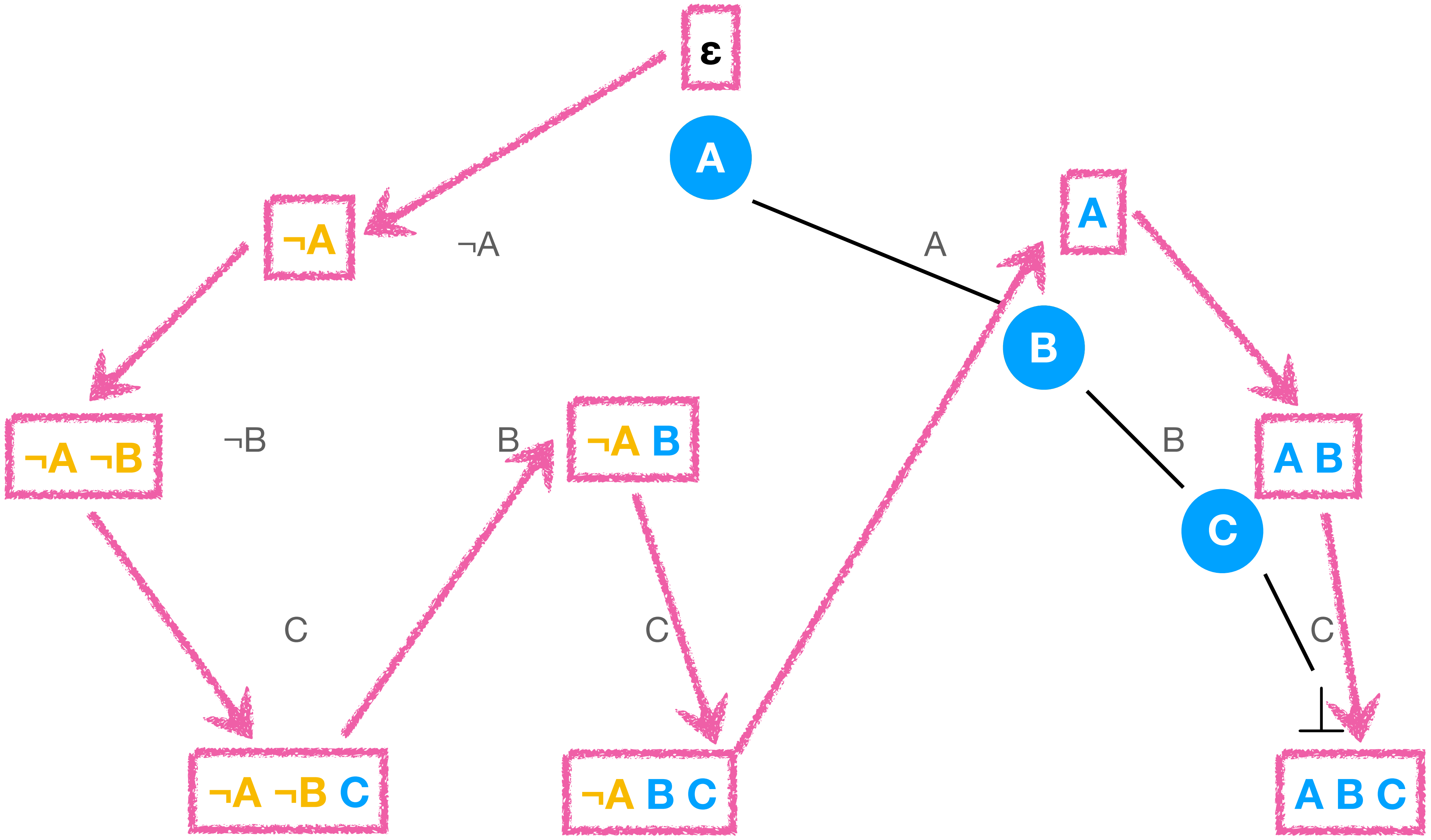
¬A B C

⊥

State (trail)

Decide

Propagate



State (trail)

Decide

Propagate

**No more transitions and conflict: UNSAT**

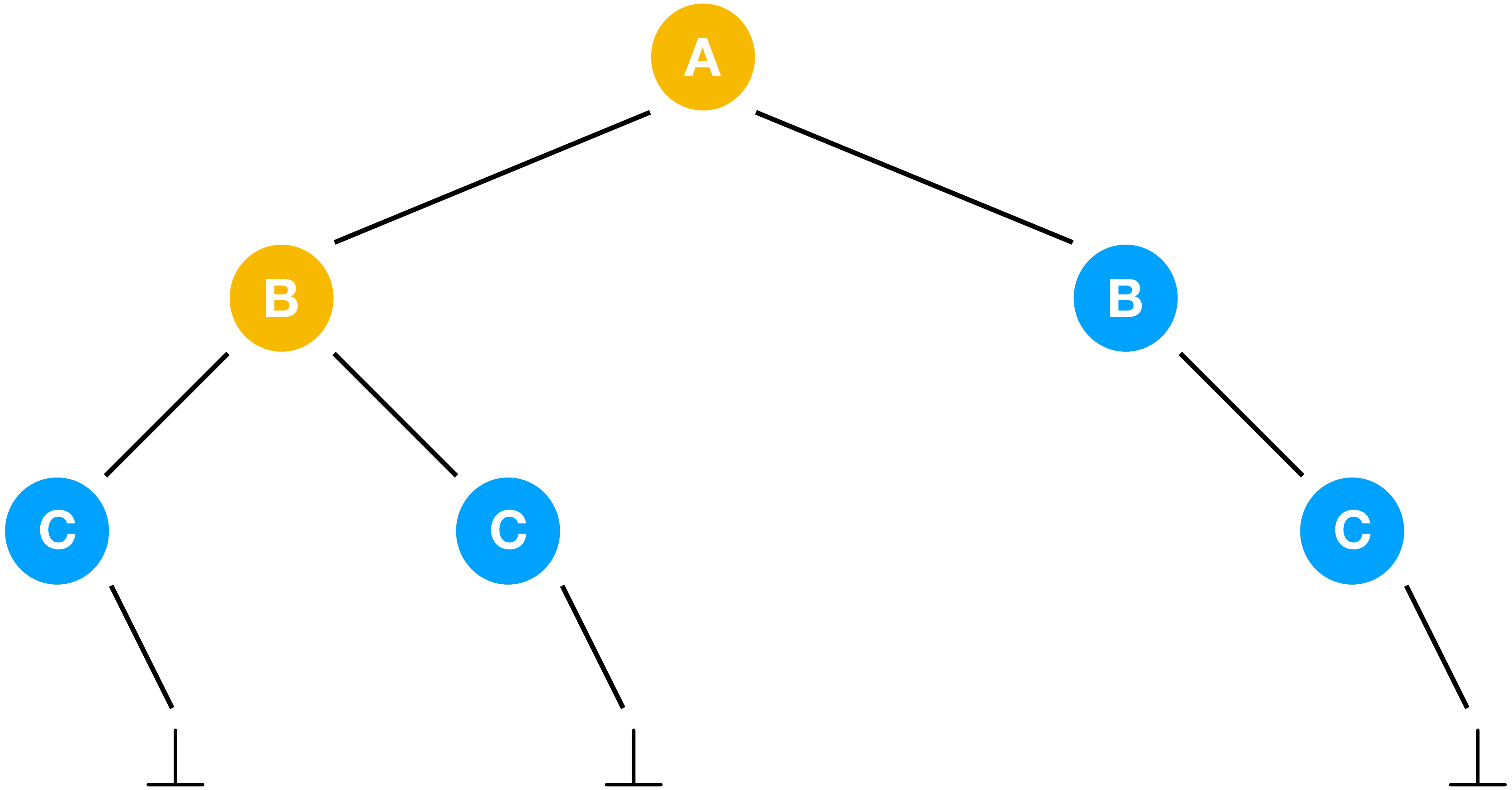
## In Isabelle

### State in Isabelle

Pair path-clauses: $(M, N)$

### Decide in Isabelle

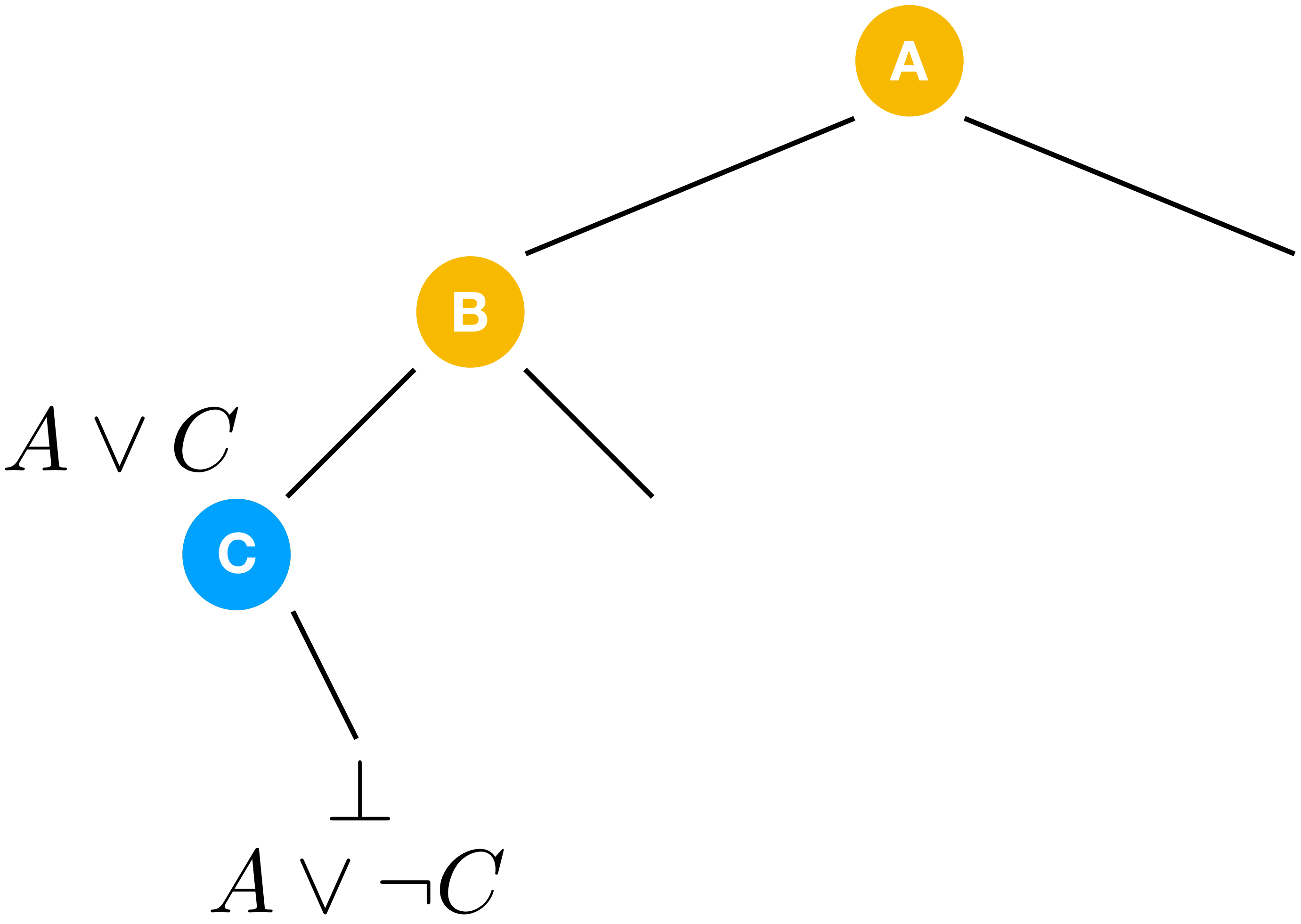
$$\text{undefined\_lit}\, M\; L \implies L \in N \implies (M, N) \Rightarrow_{\text{CDCL}} (M\, L, N)$$



Decide

Propagate

# DPLL+BJ

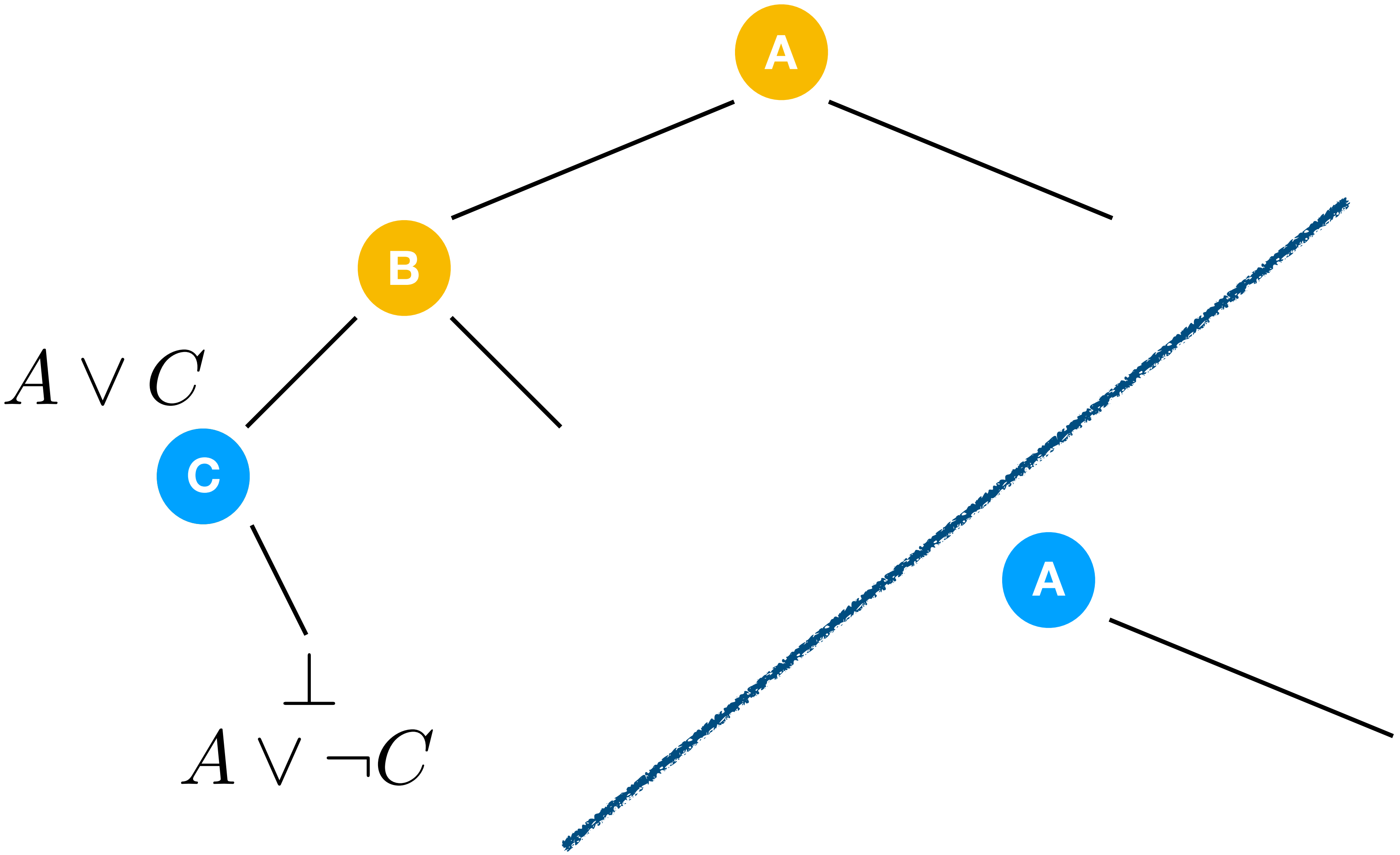


Decide

Propagate

Analyse + Backjump

# DPLL+BJ

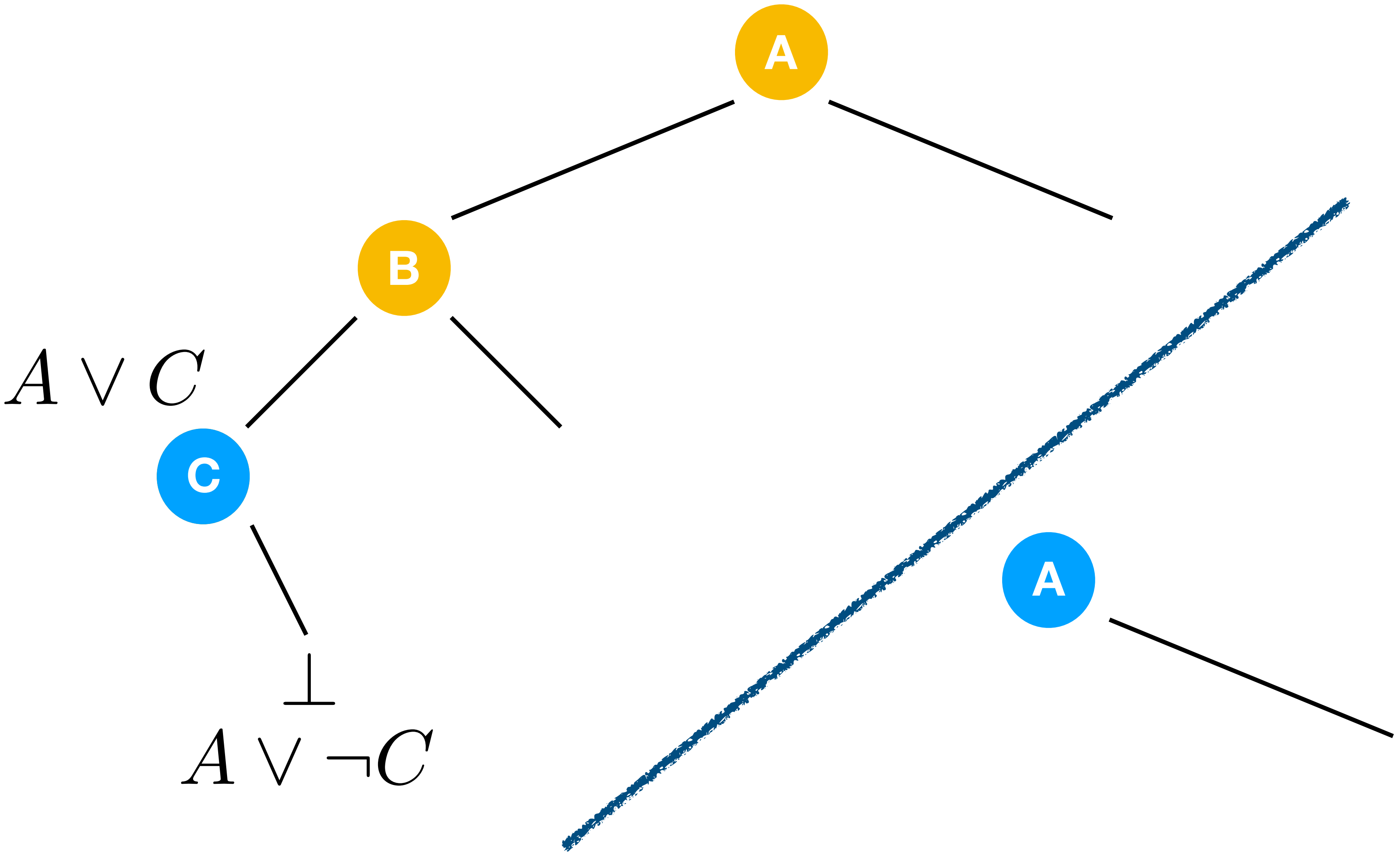


Decide

Propagate

Analyse + Backjump

# CDCL



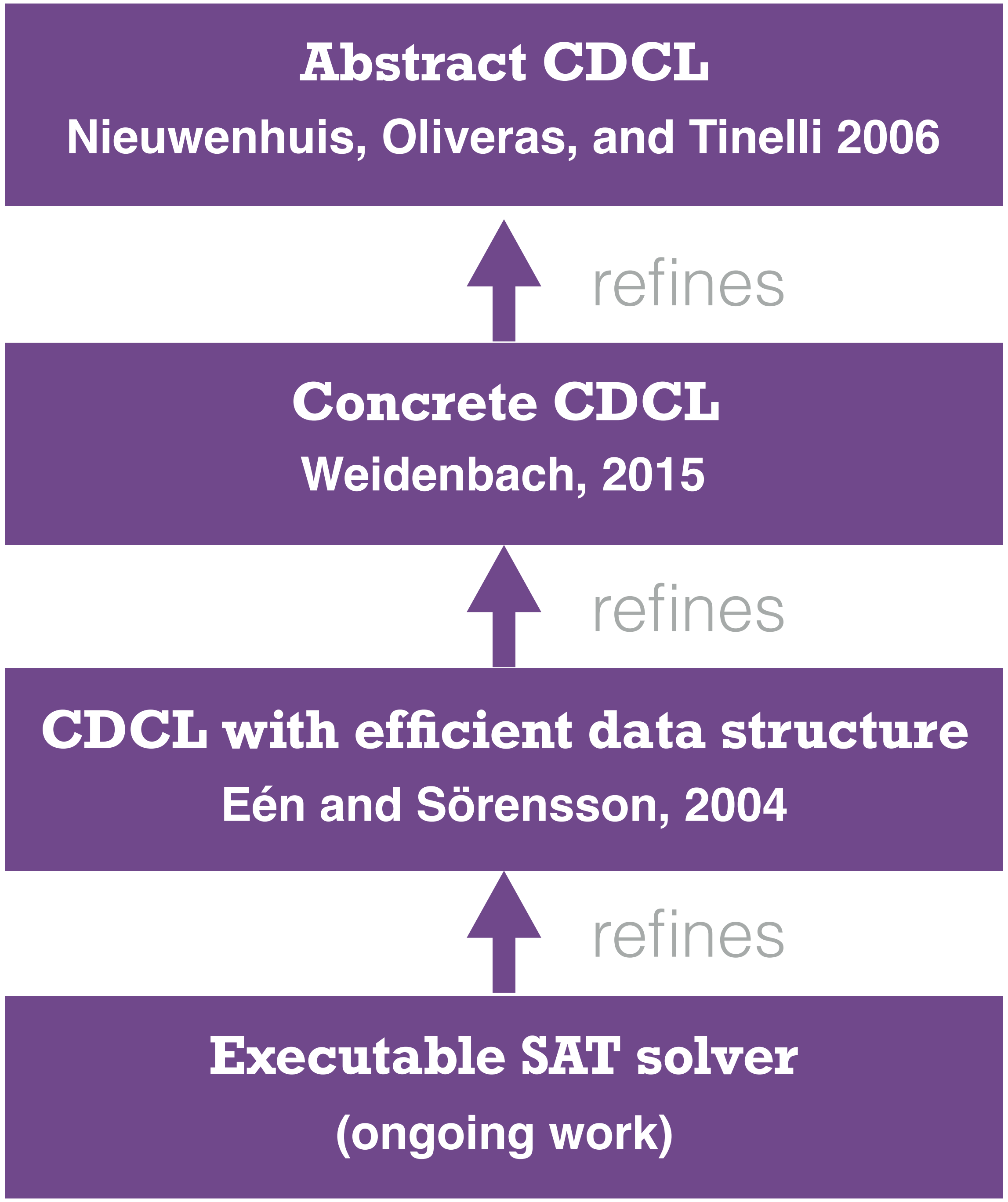
Decide

Propagate

Analyse + Backjump

Learn + forget clause

New learned clause: $A$



**Abstract CDCL**

Nieuwenhuis, Oliveras, and Tinelli 2006

refines

**Concrete CDCL**

Weidenbach, 2015

refines

**CDCL with efficient data structure**

Eén and Sörensson, 2004

refines

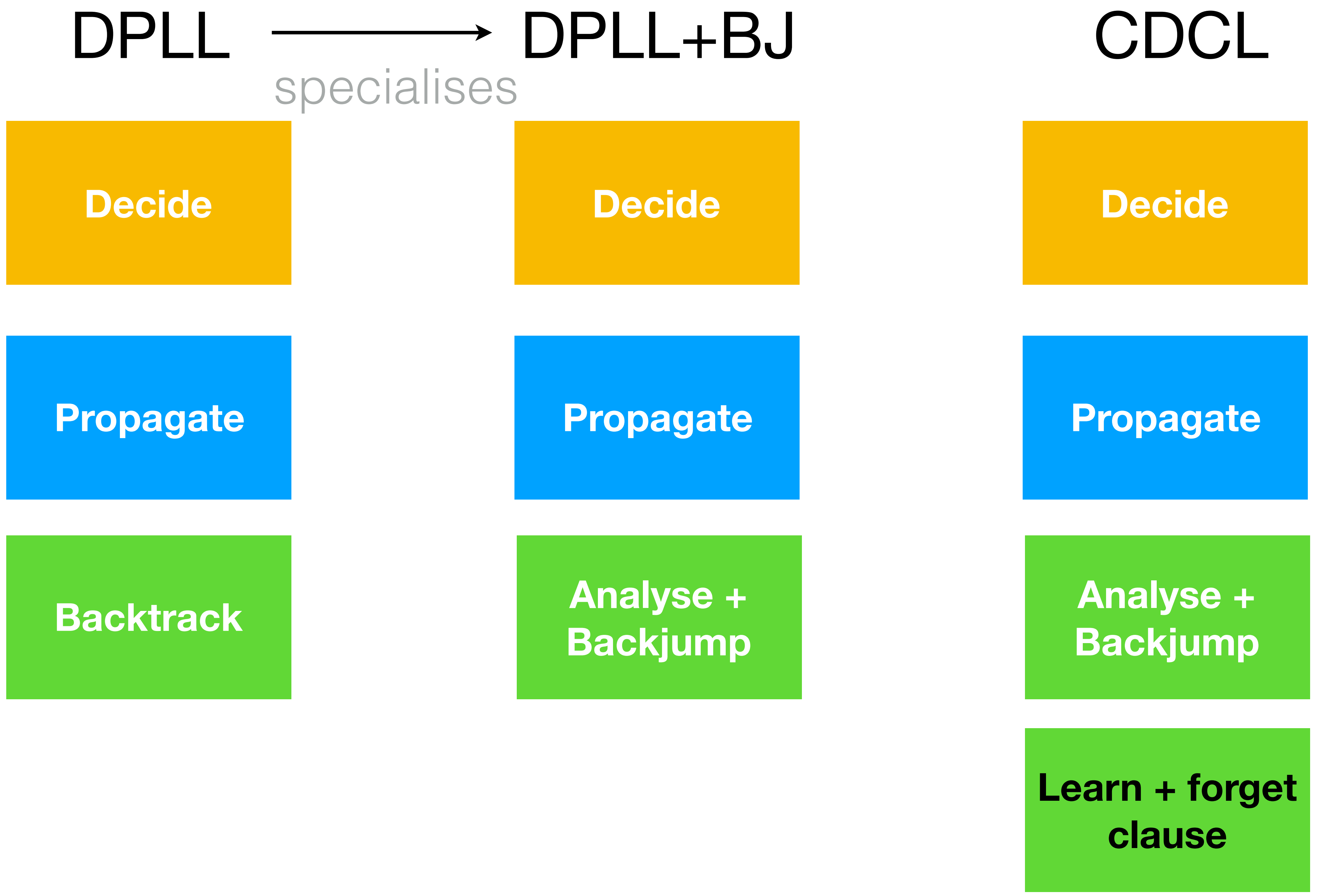
**Executable SAT solver**

(ongoing work)

# Abstract CDCL

## Nieuwenhuis, Oliveras, and Tinelli 2006

| DPLL | DPLL+BJ | CDCL |
|---|---|---|
| Decide | Decide | Decide |
| Propagate | Propagate | Propagate |
| Backtrack | Analyse + Backjump | Analyse + Backjump |
|  |  | Learn + forget clause |



DPLL

Decide

Propagate

Backtrack

specialises

DPLL+BJ

Decide

Propagate

Analyse + Backjump

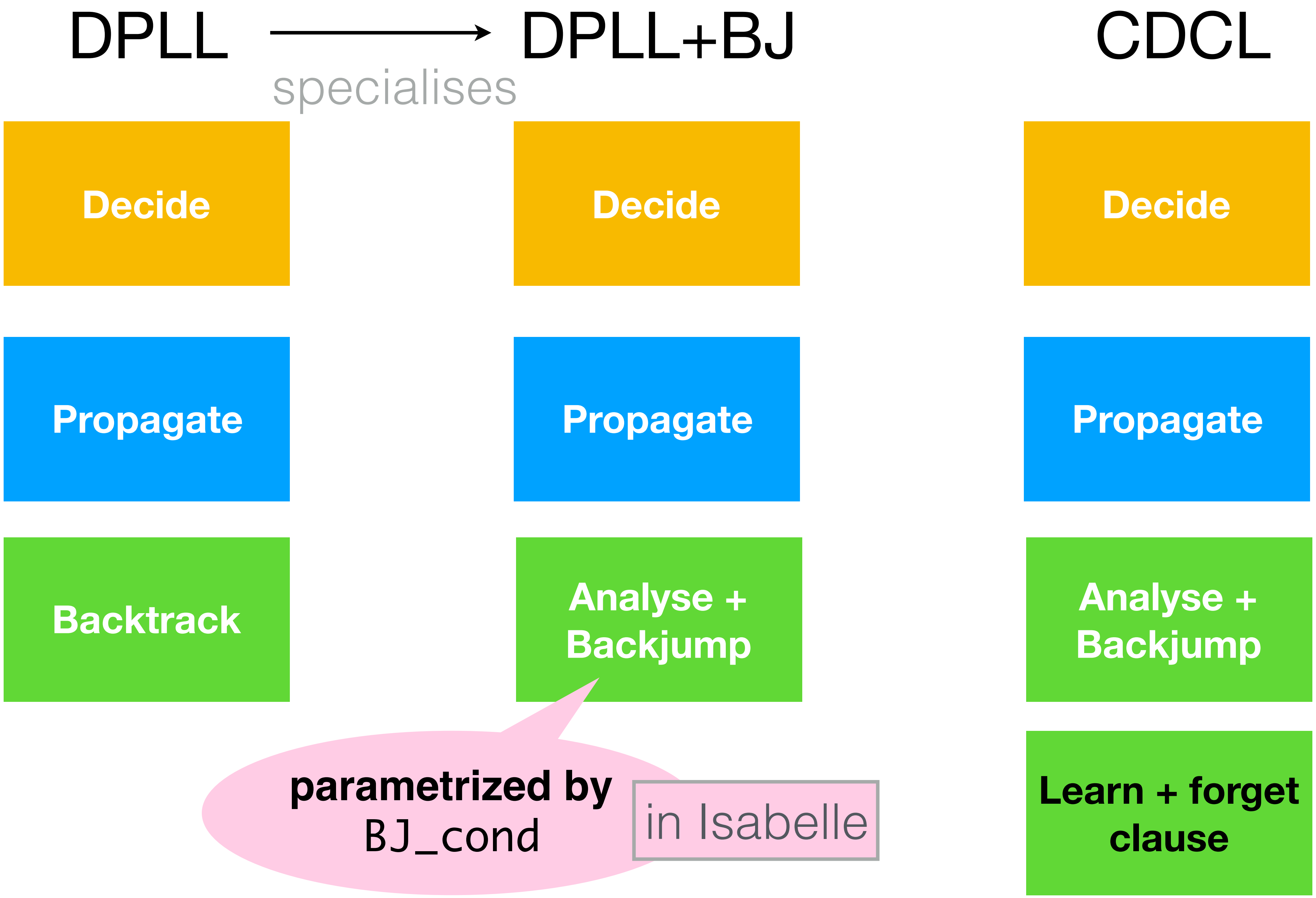
CDCL

Decide

Propagate

Analyse + Backjump

Learn + forget clause



DPLL

specialises

DPLL+BJ

CDCL

Decide

Propagate

Backtrack

Decide

Propagate

Analyse + Backjump

parametrized by `BJ_cond`

in Isabelle

Decide

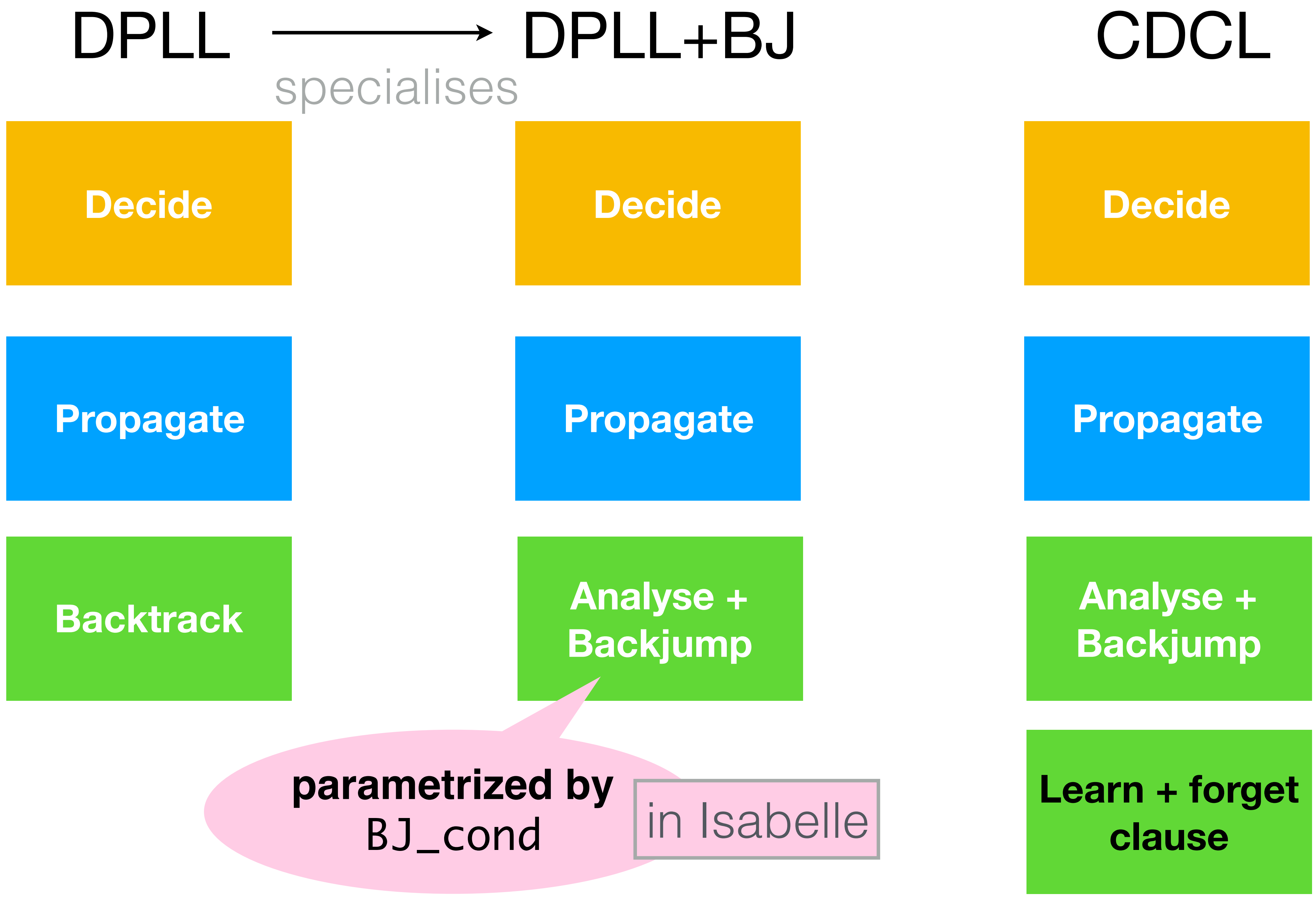
Propagate

Analyse + Backjump

Learn + forget clause

```
submodule DPLL ⊆ DPLL+BJ where
  BJ_cond = BT_cond
```

in Isabelle



DPLL → DPLL+BJ

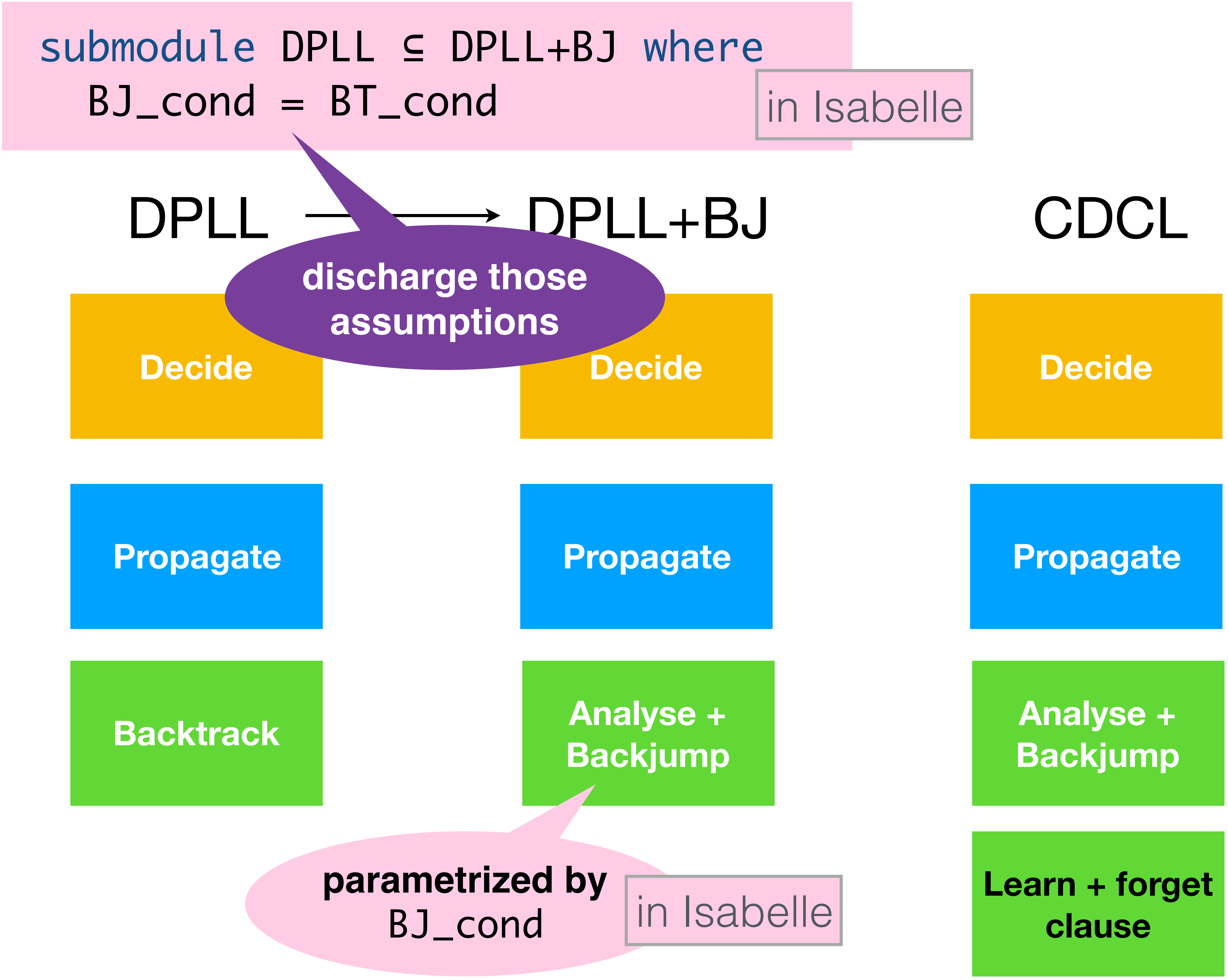
specialises

CDCL

| DPLL | DPLL+BJ | CDCL |
|---|---|---|
| Decide | Decide | Decide |
| Propagate | Propagate | Propagate |
| Backtrack | Analyse + Backjump | Analyse + Backjump |
| | | Learn + forget clause |

parametrized by `BJ_cond`

in Isabelle



```
submodule DPLL ⊆ DPLL+BJ where
  BJ_cond = BT_cond
```

in Isabelle

discharge those assumptions

DPLL → DPLL+BJ

CDCL

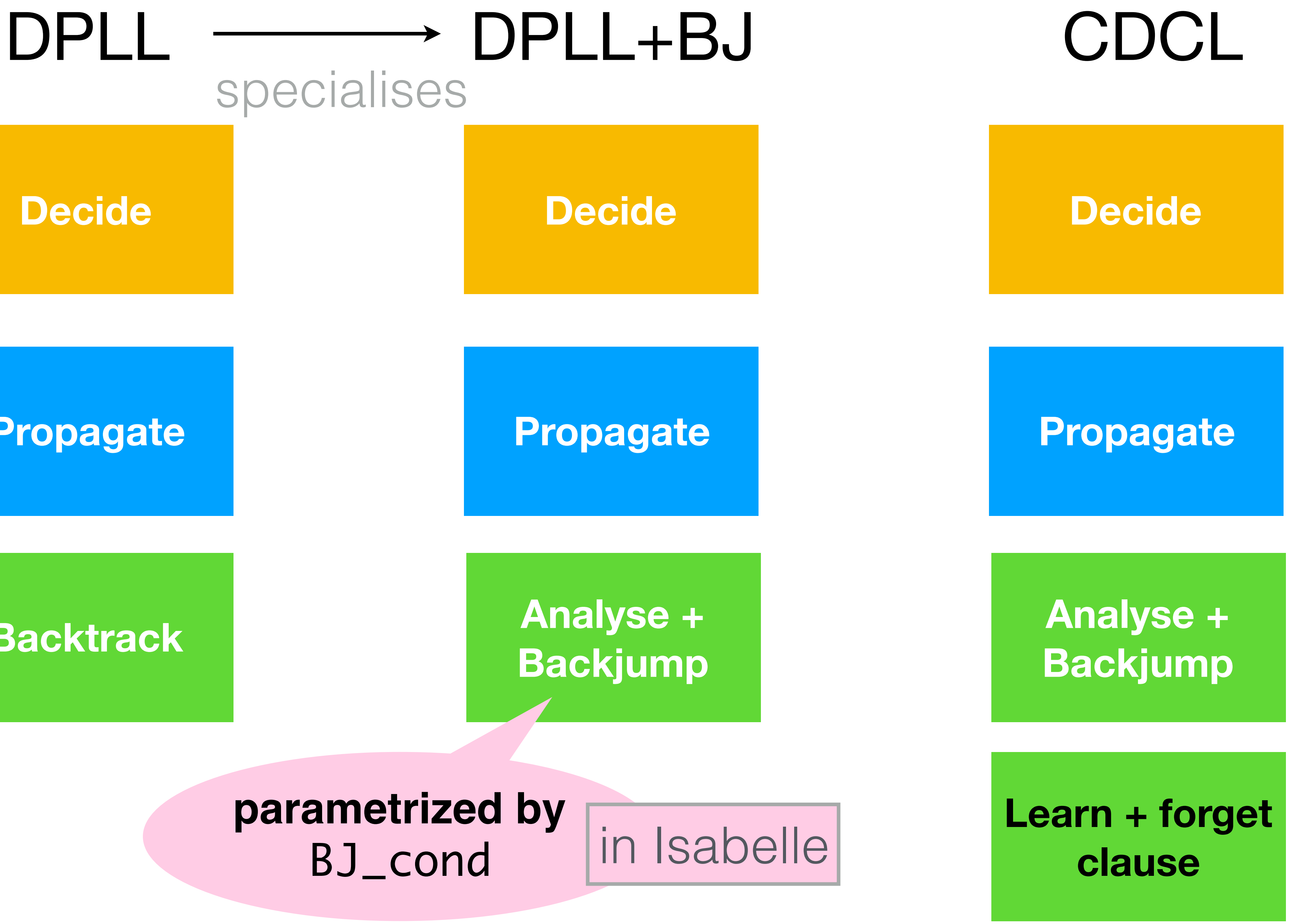
| DPLL | DPLL+BJ | CDCL |
|---|---|---|
| Decide | Decide | Decide |
| Propagate | Propagate | Propagate |
| Backtrack | Analyse + Backjump | Analyse + Backjump |
| | | Learn + forget clause |

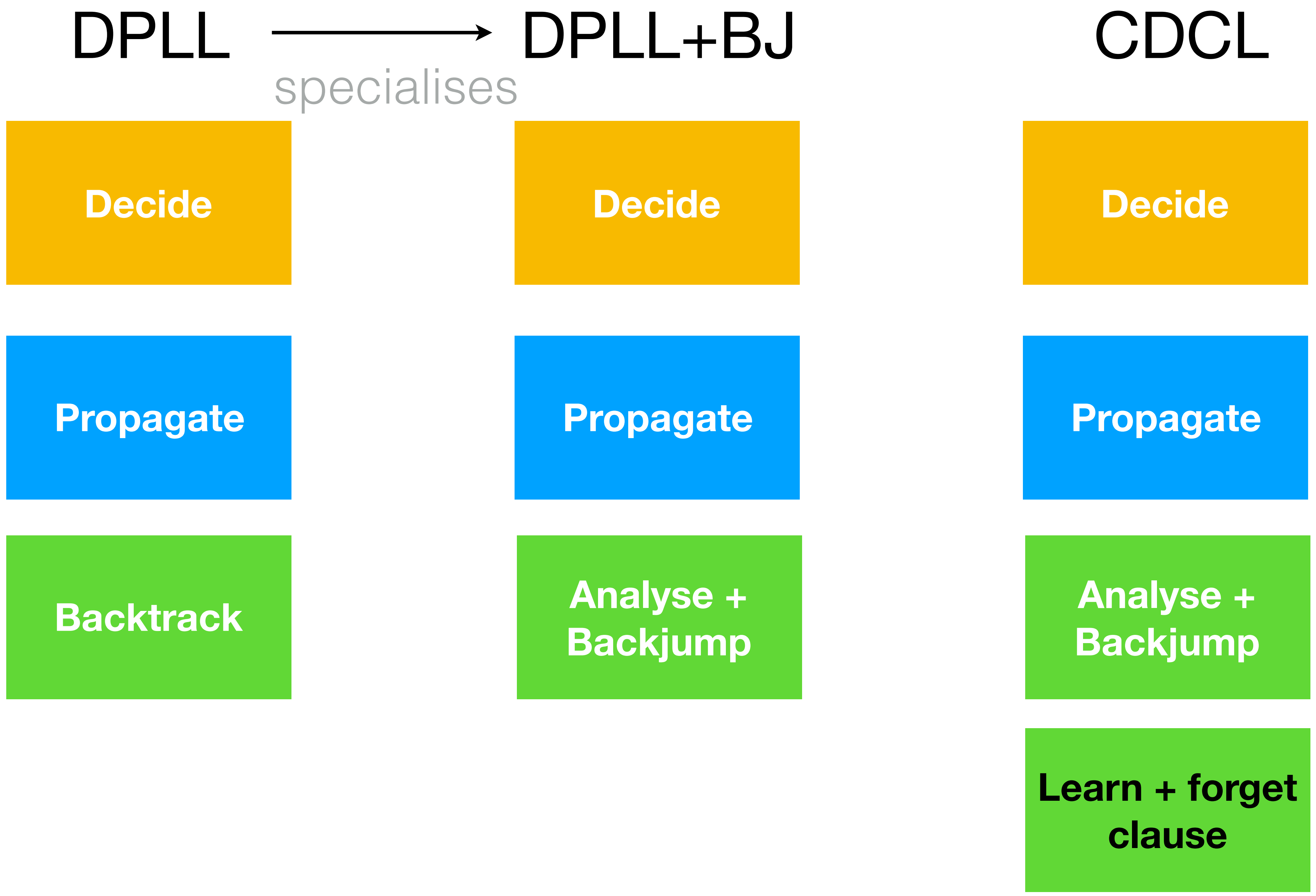
parametrized by `BJ_cond`

in Isabelle

```
submodule DPLL ⊆ DPLL+BJ where
  BJ_cond = BT_cond
```

in Isabelle





DPLL

specialises

DPLL+BJ

CDCL

Decide

Propagate

Backtrack

Decide

Propagate

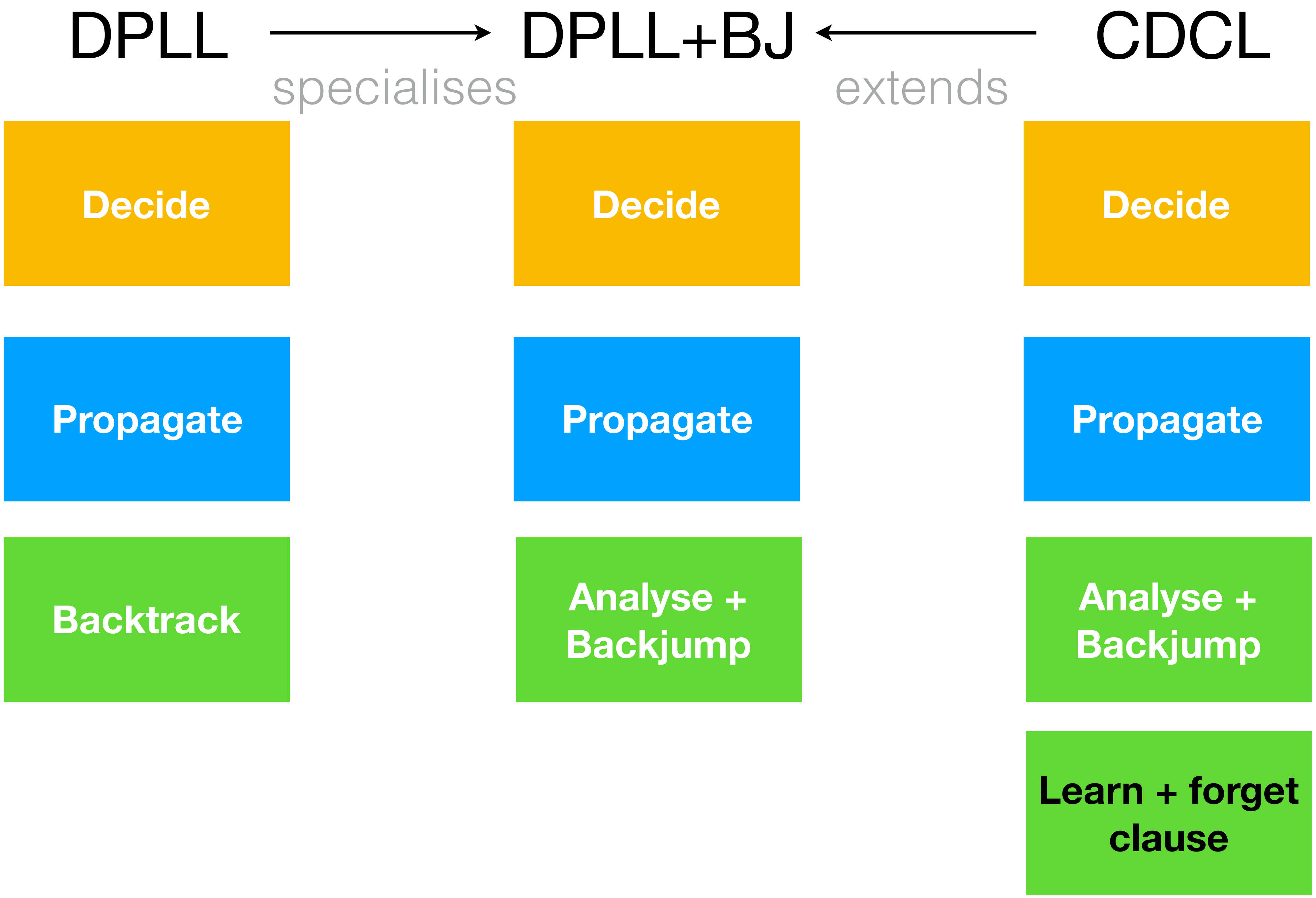
Analyse + Backjump

Decide

Propagate

Analyse + Backjump

Learn + forget clause



DPLL → DPLL+BJ ← CDCL

specialises

extends

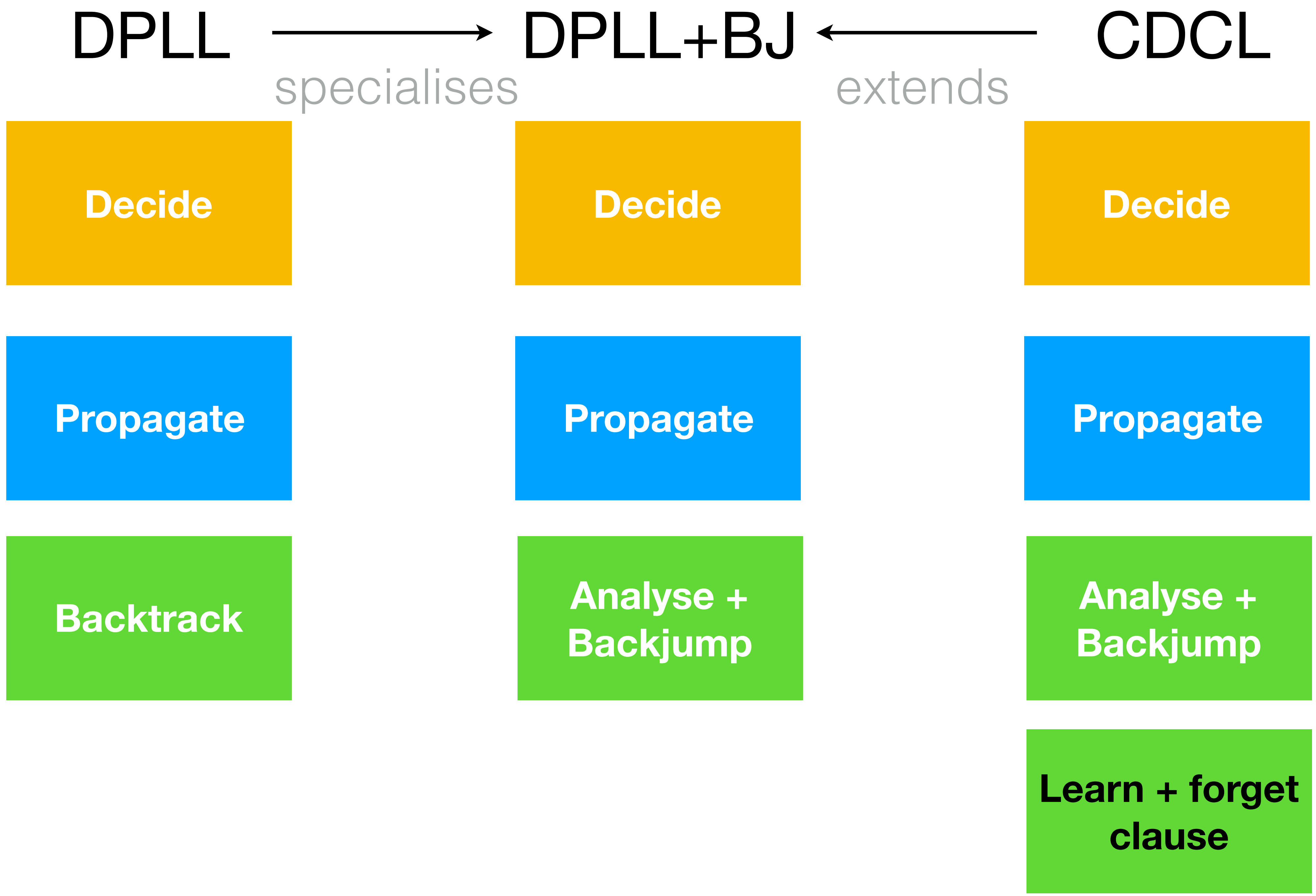
DPLL:
- Decide
- Propagate
- Backtrack

DPLL+BJ:
- Decide
- Propagate
- Analyse + Backjump

CDCL:
- Decide
- Propagate
- Analyse + Backjump
- Learn + forget clause

```
CDCL = DPLL+BJ + Learn
       + Forget
```

in Isabelle

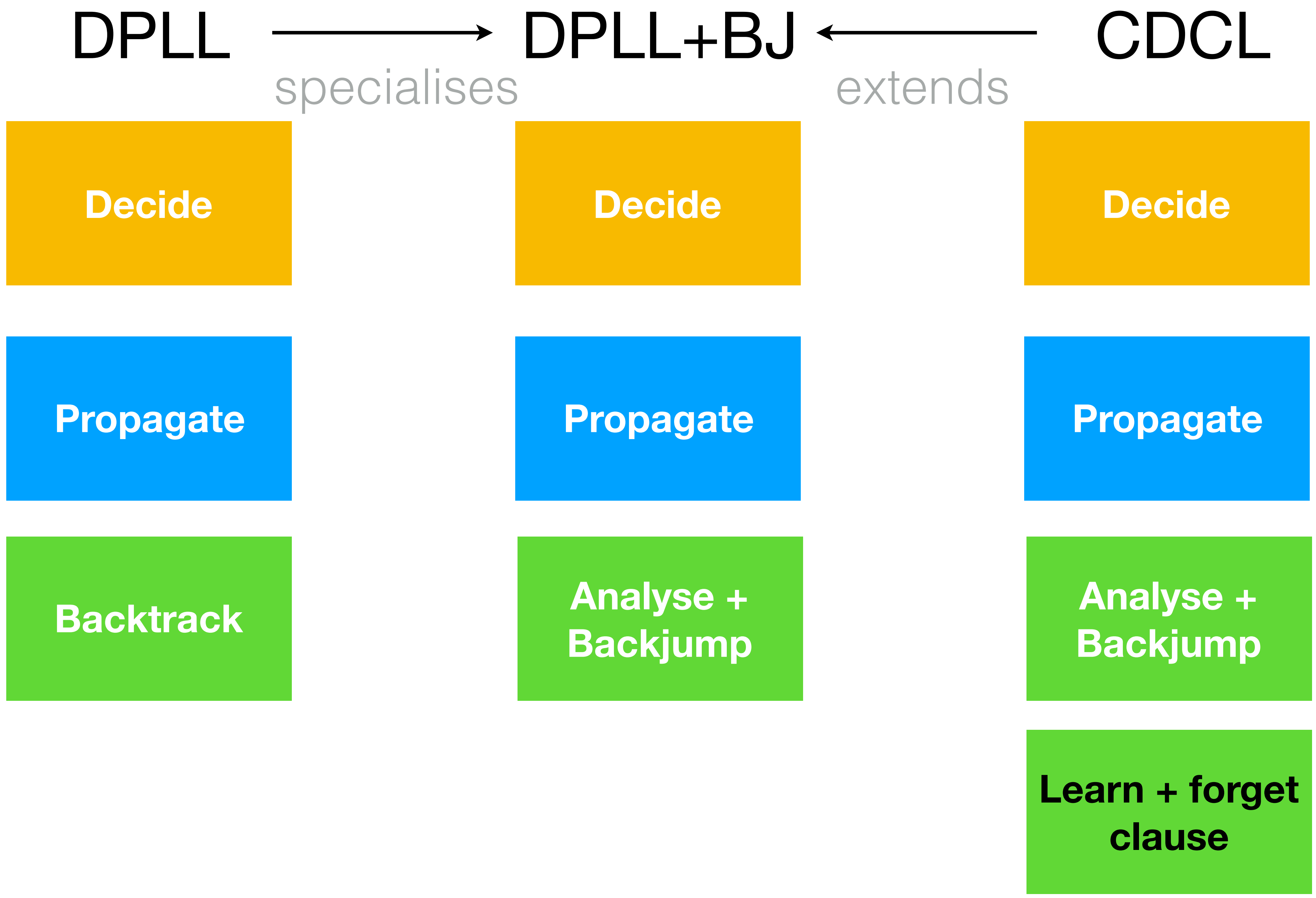


DPLL → DPLL+BJ ← CDCL

specialises

extends

| DPLL | DPLL+BJ | CDCL |
|---|---|---|
| Decide | Decide | Decide |
| Propagate | Propagate | Propagate |
| Backtrack | Analyse + Backjump | Analyse + Backjump |
| | | Learn + forget clause |



DPLL → DPLL+BJ ← CDCL

specialises

extends

DPLL:

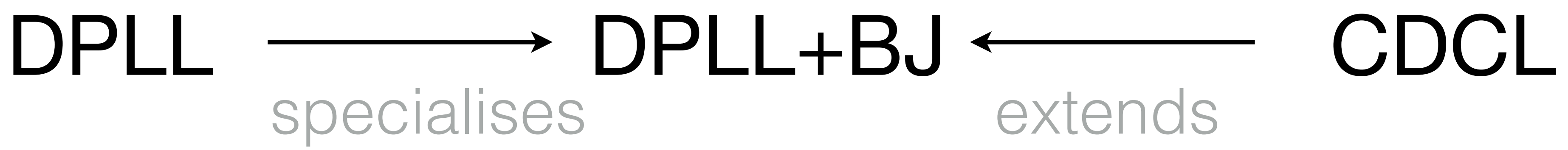
- Decide
- Propagate
- Backtrack

DPLL+BJ:

- Decide
- Propagate
- Analyse + Backjump

CDCL:

- Decide
- Propagate
- Analyse + Backjump
- Learn + forget clause



DPLL

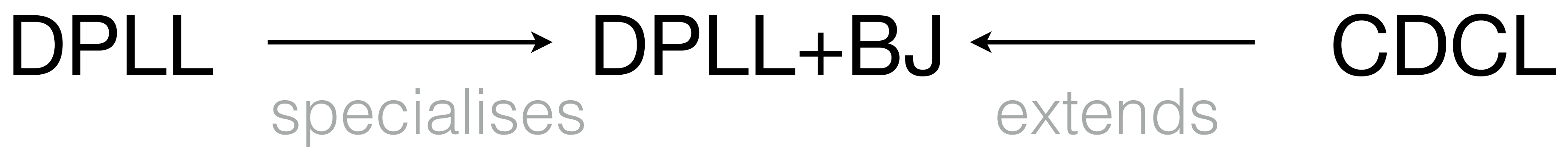
specialises

DPLL+BJ

extends

CDCL

**termination** **termination** **non-termination**



DPLL

specialises

DPLL+BJ

extends

CDCL

**termination**

**termination**

**non-termination**

**Learn + forget clause**

infinite chain of learn and forget

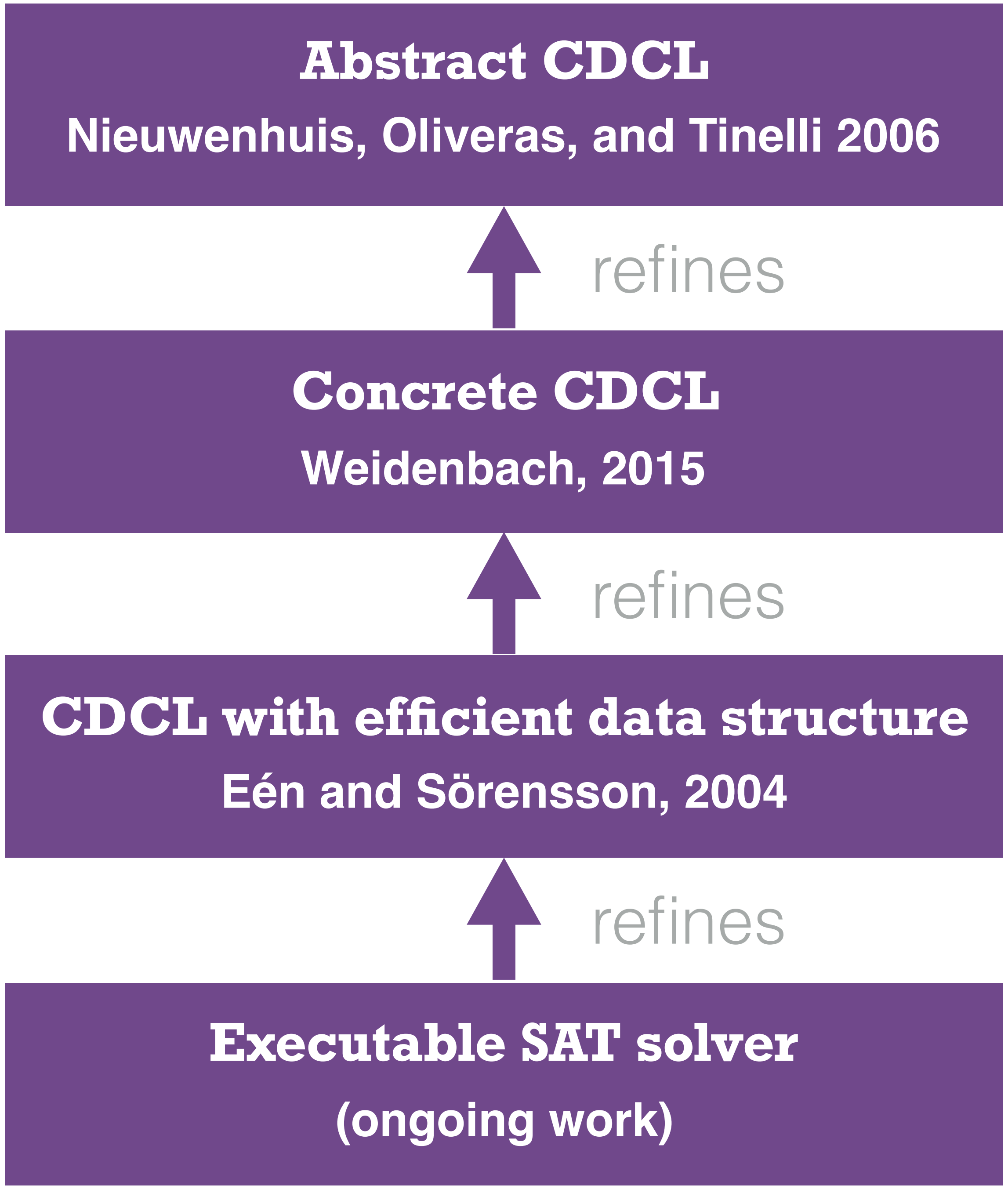
DPLL → DPLL+BJ ← CDCL

specialises extends

**termination** **termination** **non-termination**

**Analyse + Backjump** **Learn + forget clause**

~~infinite chain of learn and forget~~



**Abstract CDCL**
Nieuwenhuis, Oliveras, and Tinelli 2006

refines

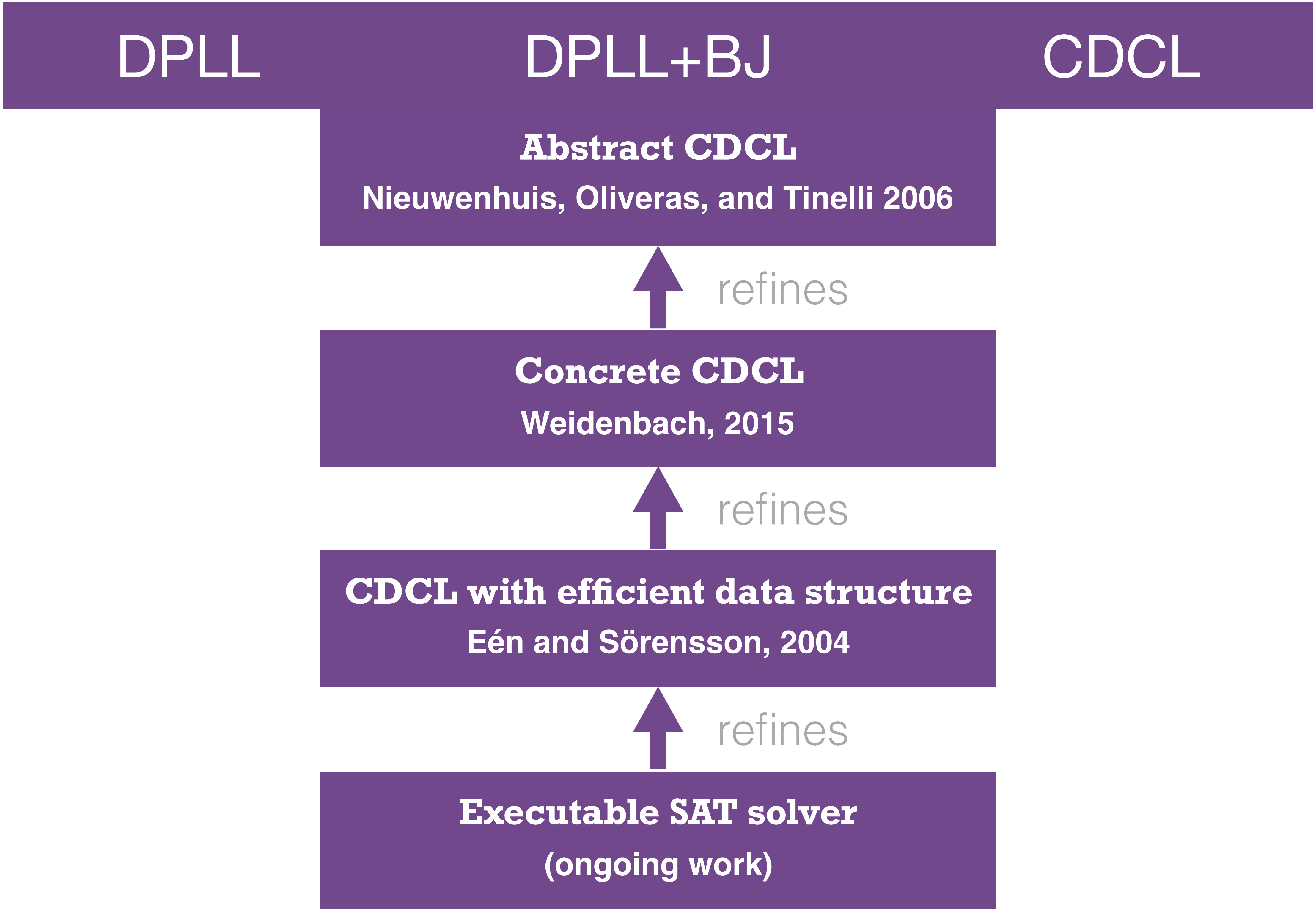
**Concrete CDCL**
Weidenbach, 2015

refines

**CDCL with efficient data structure**
Eén and Sörensson, 2004

refines

**Executable SAT solver**
(ongoing work)



DPLL

DPLL+BJ

CDCL

**Abstract CDCL**

Nieuwenhuis, Oliveras, and Tinelli 2006

refines

**Concrete CDCL**

Weidenbach, 2015

refines

**CDCL with efficient data structure**

Eén and Sörensson, 2004

refines

**Executable SAT solver**

(ongoing work)

# Concrete CDCL

Weidenbach, 2015

## Backjump

on paper

if $\mathrm{C} \in \mathrm{N}$ and $\mathrm{M} \models \neg \mathrm{C}$
and there is $\mathrm{C}'$ such that ...
$(\mathrm{M}, \mathrm{N}) \Rightarrow (\mathrm{L}\,\mathrm{M}', \mathrm{N})$

How do we get a suitable $C'$?

## Backjump

on paper

$\mathbf{if\ C \in N\ and\ M \models \neg C}$

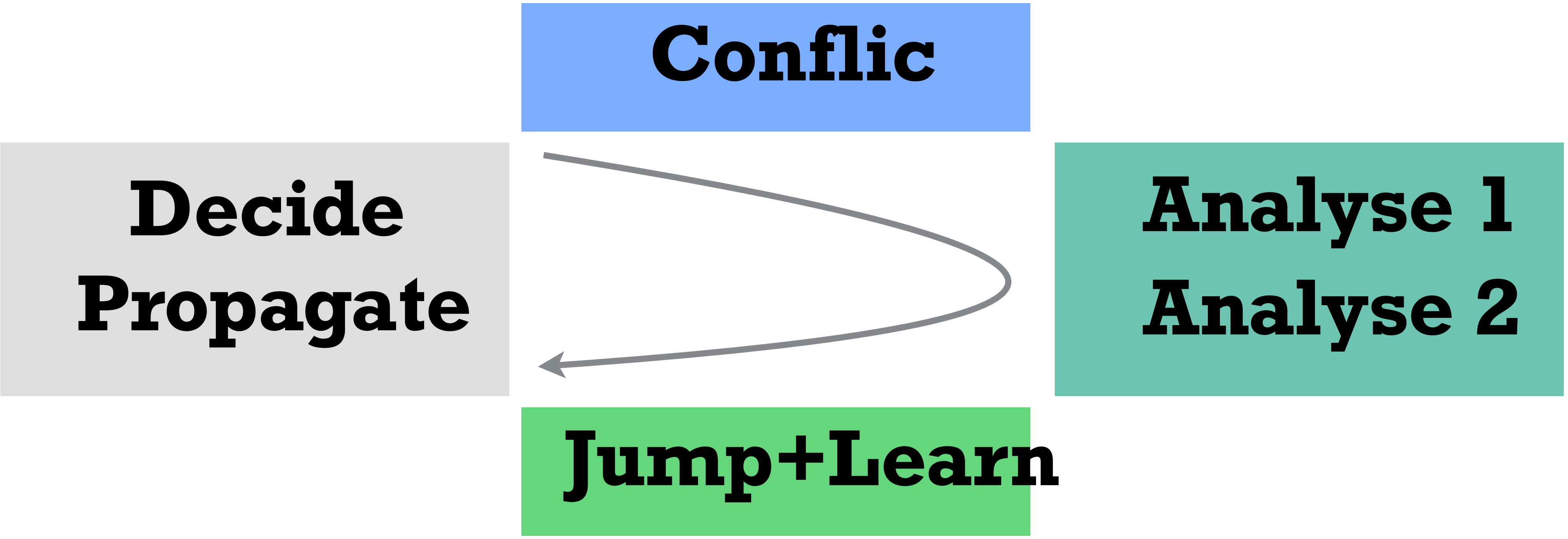
$\mathbf{and\ there\ is\ C'\ such\ that\ \ldots}$

$\mathbf{(M, N) \Rightarrow (L\, M', N)}$

How do we get a suitable $C'$?

- First unique implication point

# CDCL_conc



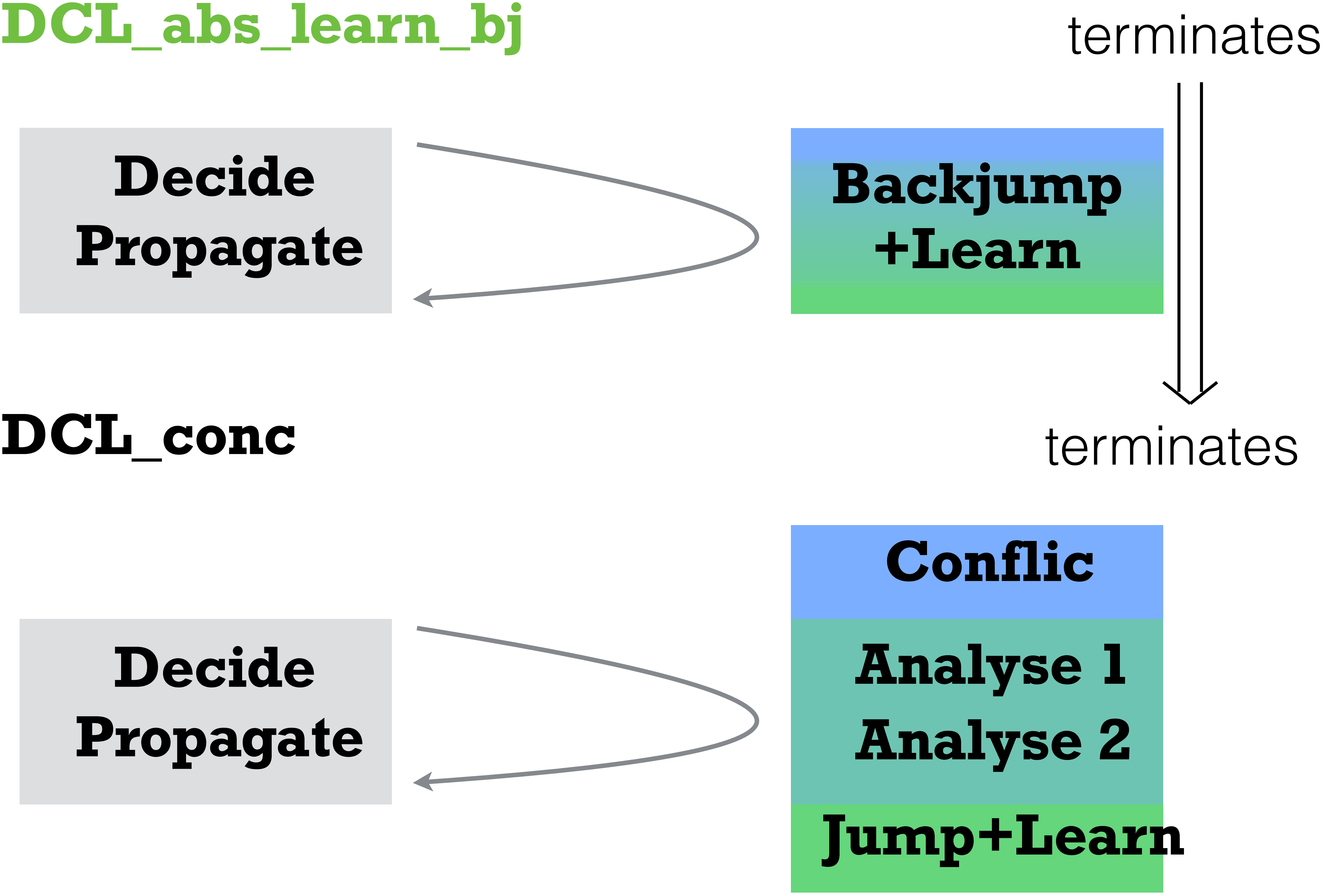
Conflic

Decide
Propagate

Analyse 1
Analyse 2

Jump+Learn

## CDCL_abs_learn_bj



terminates

Decide
Propagate

Backjump
+Learn

## CDCL_conc

terminates

Decide
Propagate

Conflic

Analyse 1
Analyse 2

Jump+Learn

**Theorem (no relearning):**
No clause can be learned twice.

# Theorem (no relearning):

No clause can be learned twice.

Proof. By contradiction. Assume CDCL learns the same clause twice, i.e., it reaches a state $(M;N;U;k;D \vee L)$ where Backtracking is applicable and $D \vee L \in (N \cup U)$.

More precisely, the state has the form $(M_1K^{i+1}M_2K_1^kK_2 \ldots K_n;N;U;k;D{\vee}L)$ where the $K_i$, $i > 1$ are propagated literals that do not occur complemented in D, as for otherwise D cannot be of level i. Furthermore, one of the $K_i$ is the complement of L.

But now, because $D \vee L$ is false in $M_1K^{i+1}M_2K_1^kK_2 \ldots K_n$ and $D \vee L \in (N \cup U)$

instead of deciding $K_1^k$ the literal L should be propagated by a reasonable strategy. A contradiction. Note that none of the $K_i$ can be annotated with $D \vee L$.

‹700 lines of proof›

in Isabelle

## **Theorem (no relearning):**

No clause can be learned twice.

Proof. By contradiction. Assume CDCL learns the same clause twice, i.e., it reaches a state $(M;N;U;k;D \vee L)$ where Backtracking is applicable and $D \vee L \in (N \cup U)$.

More precisely, the state has the form $(M_1K^{i+1}M_2K_1{}^kK_2 \ldots K_n;N;U;k;D\vee L)$ where the $K_i$, $i > 1$ are propagated literals that do not occur complemented in D, as for otherwise D cannot be of level i. Furthermore, one of the $K_i$ is the complement of L.

But now, because $D \vee L$ is false in $M_1K^{i+1}M_2K_1{}^kK_2 \ldots K_n$ and $D \vee L \in (N \cup U)$

instead of deciding $K_1{}^k$ the literal L should be propagated by a reasonable strategy. A contradiction. Note that none of the $K_i$ can be annotated with $D \vee L$.

## **Theorem (no relearning):**

No clause can be learned twice.

Proof. By contradiction. Assume CDCL learns the same clause twice, i.e., it reaches a state $(M;N;U;k;D \vee L)$ where Backtracking is applicable and $D \vee L \in (N \cup U)$.

More precisely, the state has the form $(M_1K^{i+1}M_2K_1^kK_2 \ldots K_n;N;U;k;D \vee L)$ where the $K_i$, $i > 1$ are propagated literals that do not occur complemented in D, as for otherwise D cannot be of level i. Furthermore, one of the $K_i$ is the complement of L.

But now, because $D \vee L$ is false in $M_1K^{i+1}M_2K_1^kK_2 \ldots K_n$ and $D \vee L \in (N \cup U)$

instead of deciding $K_1^k$ the literal L should be propagated by a reasonable strategy. A contradiction. Note that none of the $K_i$ can be annotated with $D \vee L$.

# Theorem (no relearning):

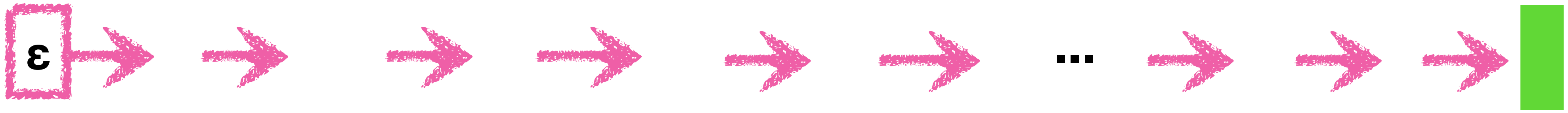
No clause can be learned twice.

Proof. By contradiction. Assume CDCL learns the same clause twice, i.e., it reaches a state $(M;N;U;k;D \lor L)$ where Backtracking is applicable and $D \lor L \in (N \cup U)$.

More precisely, the state has the form $(M_1K^{i+1}M_2K_1{}^kK_2 \ldots K_n;N;U;k;D\lor L)$ where the $K_i$, $i > 1$ are propagated literals that do not occur complemented in D, as for otherwise D cannot be of level i. Furthermore, one of the $K_i$ is the complement of L.

But now, because $D \lor L$ is false in $M_1K^{i+1}M_2K_1{}^kK_2 \ldots K_n$ and $D \lor L \in (N \cup U)$

instead of deciding $K_1{}^k$ the literal L should be propagated by a reasonable strategy. A contradiction. Note that none of the $K_i$ can be annotated with $D \lor L$.



# **Theorem (no relearning):**

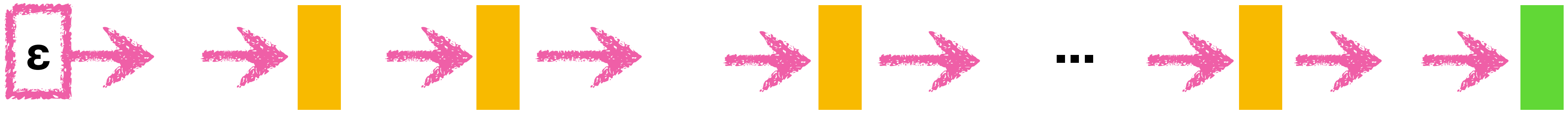
No clause can be learned twice.

Proof. By contradiction. Assume CDCL learns the same clause twice, i.e., it reaches a state $(M;N;U;k;D \lor L)$ where Backtracking is applicable and $D \lor L \in (N \cup U)$.

More precisely, the state has the form $(M_1K^{i+1}M_2K_1^kK_2 \ldots K_n;N;U;k;D\lor L)$ where the $K_i$, $i > 1$ are propagated literals that do not occur complemented in D, as for otherwise D cannot be of level i. Furthermore, one of the $K_i$ is the complement of L.

But now, because $D \lor L$ is false in $M_1K^{i+1}M_2K_1^kK_2 \ldots K_n$ and $D \lor L \in (N \cup U)$

instead of deciding $K_1^k$ the literal L should be propagated by a reasonable strategy. A contradiction. Note that none of the $K_i$ can be annotated with $D \lor L$.



# Theorem (no relearning):

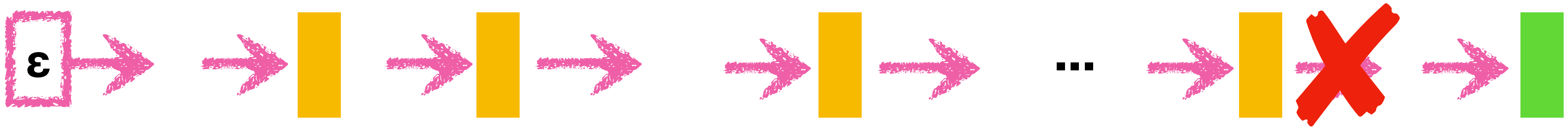
No clause can be learned twice.

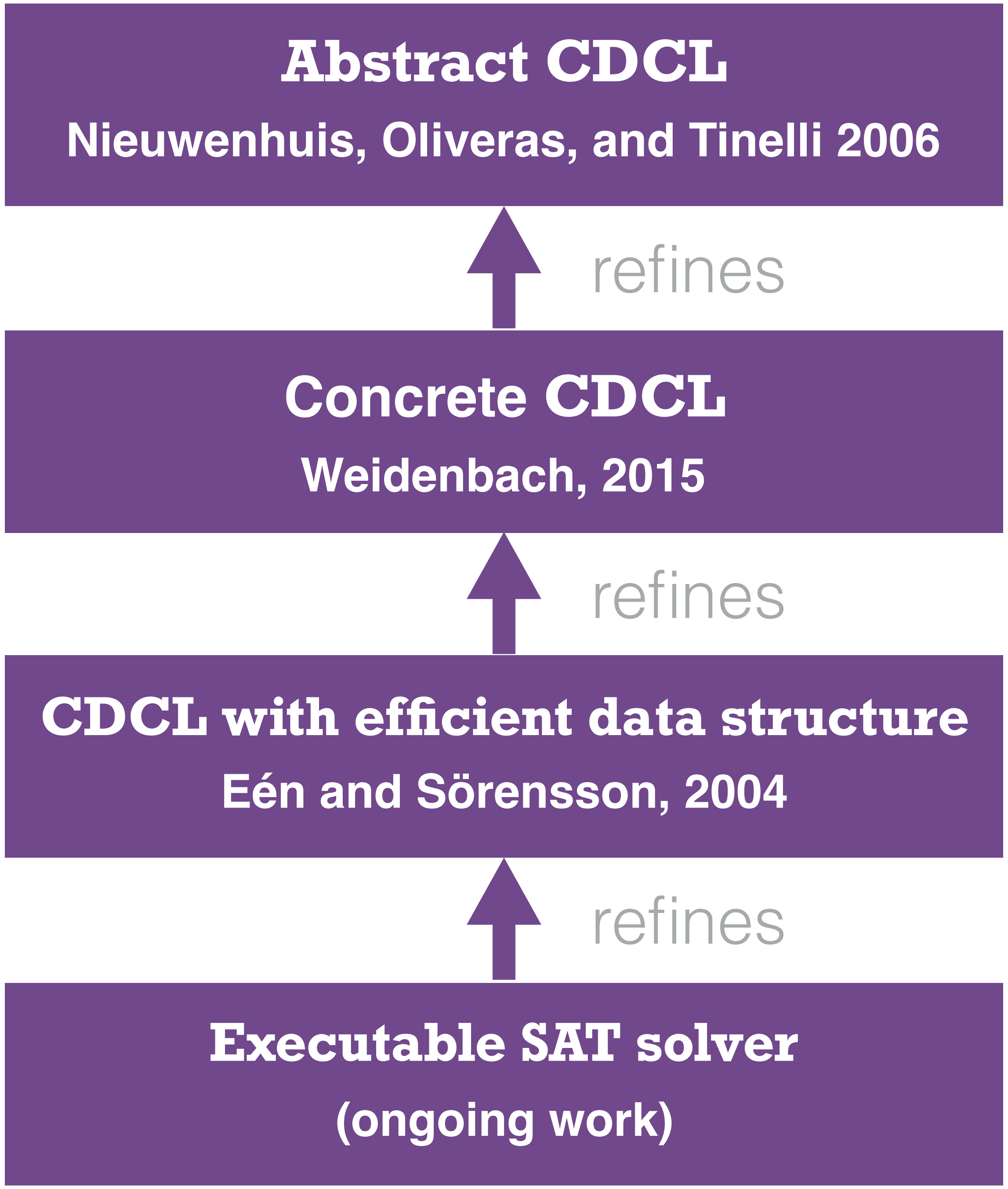
Proof. By contradiction. Assume CDCL learns the same clause twice, i.e., it reaches a state $(M;N;U;k;D \vee L)$ where Backtracking is applicable and $D \vee L \in (N \cup U)$.

More precisely, the state has the form $(M_1K^{i+1}M_2K_1{}^kK_2 \ldots K_n;N;U;k;D \vee L)$ where the $K_i$, $i > 1$ are propagated literals that do not occur complemented in D, as for otherwise D cannot be of level i. Furthermore, one of the $K_i$ is the complement of L.

But now, because $D \vee L$ is false in $M_1K^{i+1}M_2K_1{}^kK_2 \ldots K_n$ and $D \vee L \in (N \cup U)$

instead of deciding $K_1{}^k$ the literal L should be propagated by a reasonable strategy. A contradiction. Note that none of the $K_i$ can be annotated with $D \vee L$.





**Abstract CDCL**

Nieuwenhuis, Oliveras, and Tinelli 2006

refines

**Concrete CDCL**

Weidenbach, 2015

refines

**CDCL with efficient data structure**

Eén and Sörensson, 2004

refines

**Executable SAT solver**

(ongoing work)

| Rules | Theory | Practice | How is it done? |
|---|---|---|---|
| Propagate | Critical | Critical | Data structure |
| Decide | Don’t care | Critical | Heuristics |

# CDCL with efficient data structure

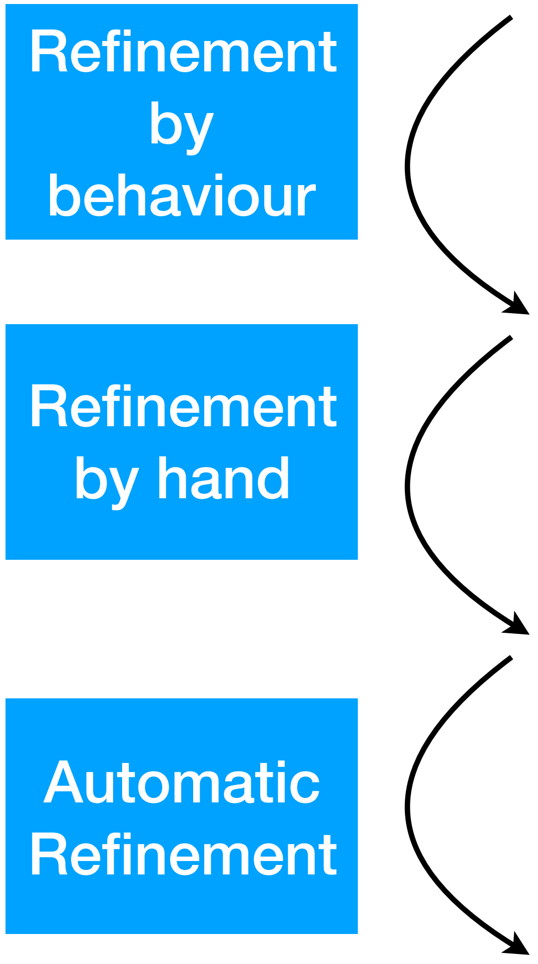
Eén and Sörensson, 2004

- Key data structure: two watched literals
- Nice to have formally

# Two Watched Literals

For each clause:

- Keep two literals unset or true
- If you can't:
  - propagate or
  - mark conflict or
  - ignore if one literal is true

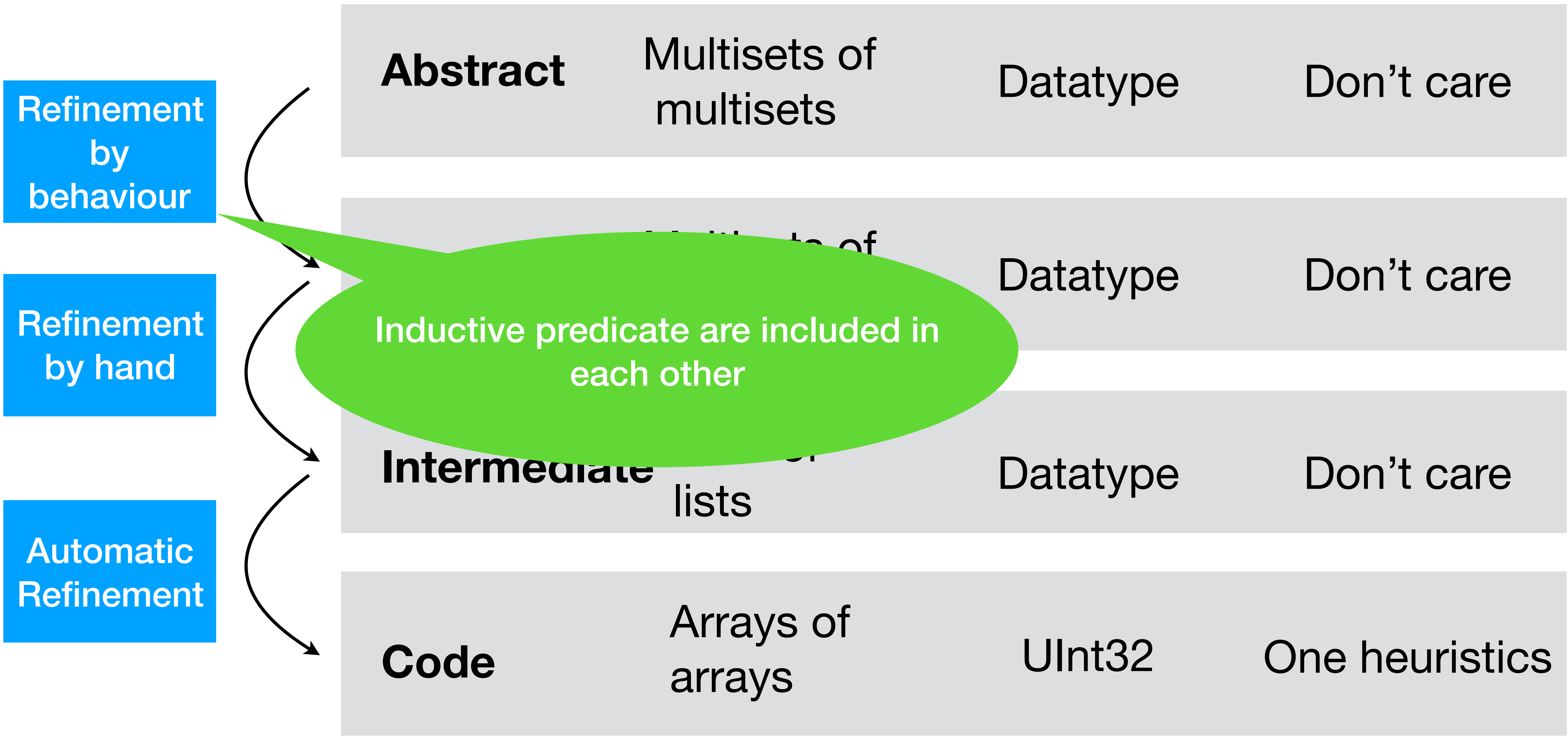


Refinement by behaviour

Refinement by hand

Automatic Refinement

| CDCL | Clauses | Literals | Decision |
|---|---|---|---|
| **Abstract** | Multisets of multisets | Datatype | Don’t care |
| **Concrete** | Multisets of multisets | Datatype | Don’t care |
| **Intermediate** | Lists of lists | Datatype | Don’t care |
| **Code** | Arrays of arrays | UInt32 | One heuristics |



Refinement by behaviour

Refinement by hand

Automatic Refinement

Inductive predicate are included in each other

| CDCL | Clauses | Literals | Decision |
|---|---|---|---|
| **Abstract** | Multisets of multisets | Datatype | Don't care |
|  | Multisets of | Datatype | Don't care |
| **Intermediate** | lists | Datatype | Don't care |
| **Code** | Arrays of arrays | UInt32 | One heuristics |

| CDCL | Clauses | Literals | Decision |
|---|---|---|---|
| **Abstract** | Multisets of multisets | Datatype | Don’t care |
| **Concrete** | Multisets of multisets | Datatype | Don’t care |
|  | Lists of | Datatype | Don’t care |
| **Code** | arrays | UInt32 | One heuristics |

Refinement by behaviour

Refinement by hand

Automatic Refinement

Refinement Framework generates aligns programs and generates conditions to prove

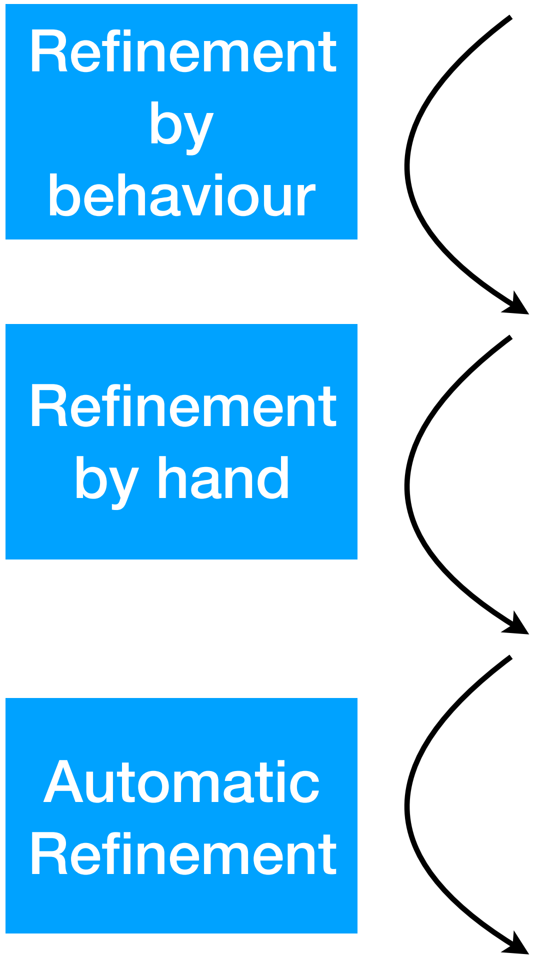
| CDCL | Clauses | Literals | Decision |
|---|---|---|---|
| **Abstract** | Multisets of multisets | Datatype | Don’t care |
| **Concrete** | Multisets of multisets | Datatype | Don’t care |
| **Intermediate** | Lists of lists | Datatype | Don’t care |
| **Code** | Arrays of arrays | UInt32 | One heuristics |

Refinement by behaviour

Refinement by hand

Automatic Refinement

Mapping of concrete and code operations, synthesis and precondition discharging done automatically



Refinement by behaviour

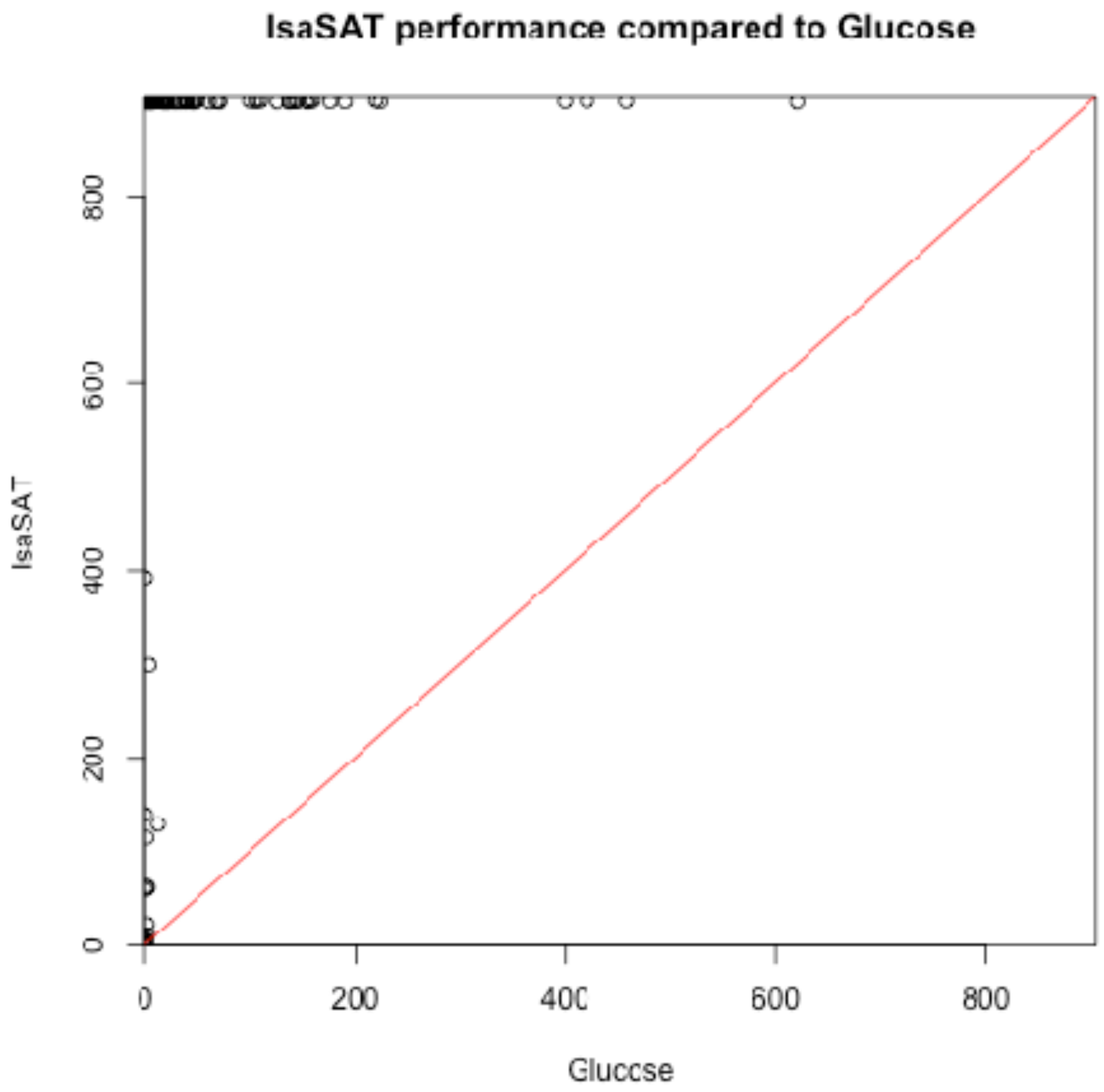
Refinement by hand

Automatic Refinement

| CDCL | Clauses | Literals | Decision |
|---|---|---|---|
| **Abstract** | Multisets of multisets | Datatype | Don’t care |
| **Concrete** | Multisets of multisets | Datatype | Don’t care |
| **Intermediate** | Lists of lists | Datatype | Don’t care |
| **Code** | Arrays of arrays | UInt32 | One heuristics |

Can also be changed

# How efficient is it compared to state-of-the-art Glucose?



# Some features of Glucose

| | Calculus | Code |
|---|---|---|
| **Presimplification of the problem** | Not relevant | |
| **Learned clause minimization** | Already generalized | Partial & TODO |
| **Conflict Representation** | Orthogonal | on-going |

# Some features of Glucose

| | Calculus | Code |
|---|---|---|
| **Forget + Restarts** | Included | TODO |
| **Trail reuse in Restarts** | Orthogonal | TODO (partially)? |
| **Hyper binary Resolution** | Not Expressible | |

# How hard is it?

| | Paper | Proof assistant |
|---|---|---|
| **Abstract CDCL** | 13 pages | 50 pages |
| **Concrete CDCL** | 9 pages<br>(½ month) | 90 pages<br>(5 months) |
| **Two-Watched** | 1 page<br>(C++ code of MiniSat) | 265 pages<br>(9 months) |

# Conclusion

Concrete outcome

- verified SAT solver framework
- verified executable SAT solver
- improve book draft

Methodology

- Refinement

Future work

- SAT Modulo Theories (e.g., CVC4, veriT, Yices, Z3)