

Dynamic Constraint Solving

Handling Change and Uncertainty in Constraint Satisfaction

G rard Verfaillie, LAAS-CNRS, Toulouse, France
Narendra Jussien, EMN, Nantes, France

www.emn.fr/jussien/CP03tutorial



The contributors

J.Amilhastre P.Barahona R.Barták S.Beale C.Beck A.Bellicha T.Benoist P.Berlandier C.Bessière
J.Bidot P.Boizumault A.Borning E.Bourreau J.Branke J.Bresina K.Brown B.Carlson Y.Caseau A.Cesta
S.Chien C.Chiu C.Chou P.Codognet J.Conley C.Conway R.Cucchiara B.Daun A.Davenport P.David
E.Davis M.Deale R.Debruyne A.Dechter R.Dechter M.Drummond A.Elkhyari H.El Sakkout F.Fages
B.Falkenhainer B.Faltings H.Fargier G.Ferrand D.Fowler J.Fowler B.Freeman-Benson E.Freuder
M.Fromherz M.Gavanelli C.Gefflot Y.Georget C.Gervet U.Geske M.Ginsberg T.Gruenhagen C.Guéret
E.Hebrard B.Hnich P.Jarvis P.Jégou H.Jung N.Jussien M.Johnston R.Knight S.Kulkarni P.Laird
E.Lamma J.Lang M.Lauvergne H.Leung Y.Leung M.Littman J.Lee T.Le Provost D.Lesaint W.Lesaint
O.Lhomme S.Macho-Gonzalez S.Majercik J.Maloney S.Manandhar P.Marquis K.Marriott D.Mattfeld
F.Menezes I.Miguel M.Milano S.Mittal P.Mello S.Minton P.Modi U.Montanari P.Morris P.Moulder
C.Muller T.Müller N.Muscettola B.Neveu A.Oddi S.Ouis A.Parkes A.Philips M.Piccardi T.Pitassi
N.Policella P.Prosser L.Purvis Y.Qu G.Rabideau Y.Ran G.Ringwelski N.Roos F.Rossi B.Rottembourg
A.Roy H.Rudová M.Sannella T.Schiex H.Schlenker Q.Shen W.Shen R.Sherwood S.Smith T.Sola
A.Stechert P.Stuckey K.Swanson M.Tambe A.Tarim A.Tessier G.Trombettoni I.Tsamardinos J.Van Den
Herik P.Van Hentenryck B.Venable G.Verfaillie T.Vidal M.Wallace R.Wallace T.Walsh A.Wolf
N.Yorke-Smith M.Zweden.

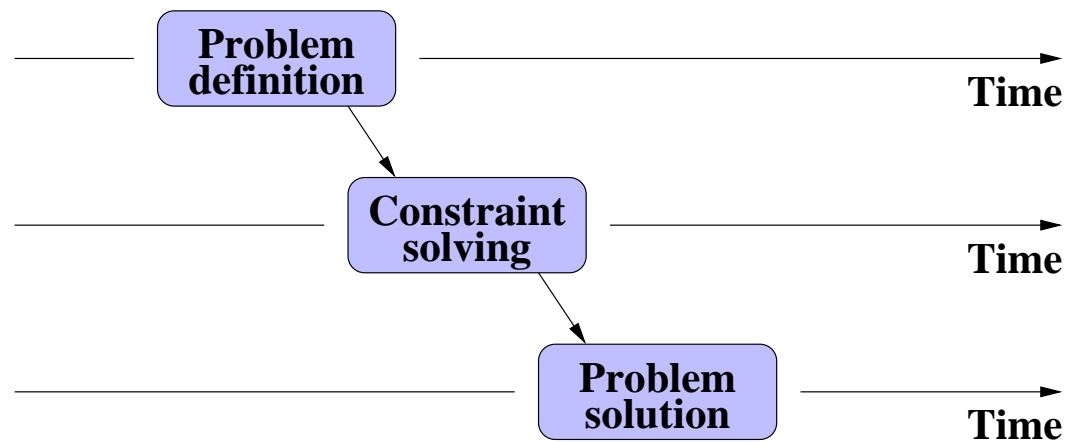
Tutorial outline

1. **Setting** and **requirements**;
2. **Reactive** methods:
 - (a) **solution** reuse;
 - (b) **reasoning** reuse;
3. **Proactive** methods:
 - (a) **robust** solutions;
 - (b) **flexible** solutions;
4. **Research directions**.

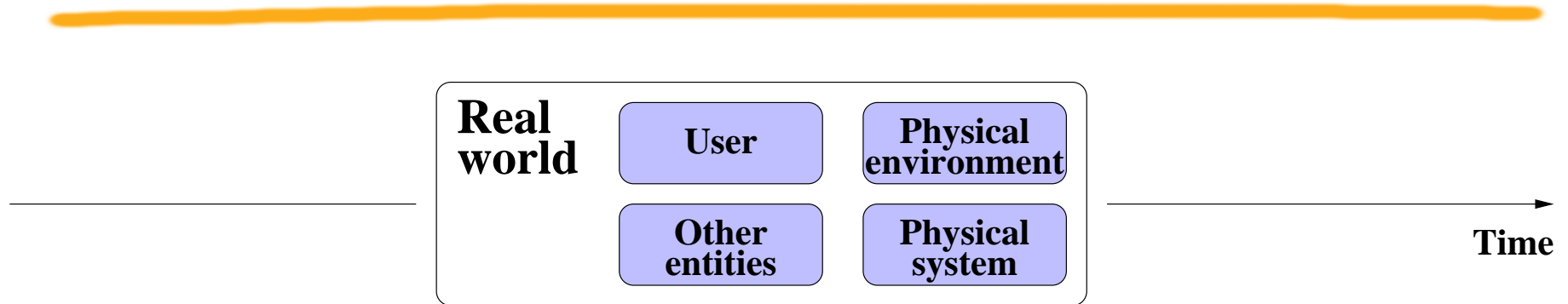
Tutorial outline

1. **Setting** and **requirements** ⇐
2. **Reactive** methods:
 - (a) **solution** reuse;
 - (b) **reasoning** reuse;
3. **Proactive** methods:
 - (a) **robust** solutions;
 - (b) **flexible** solutions;
4. **Research directions.**

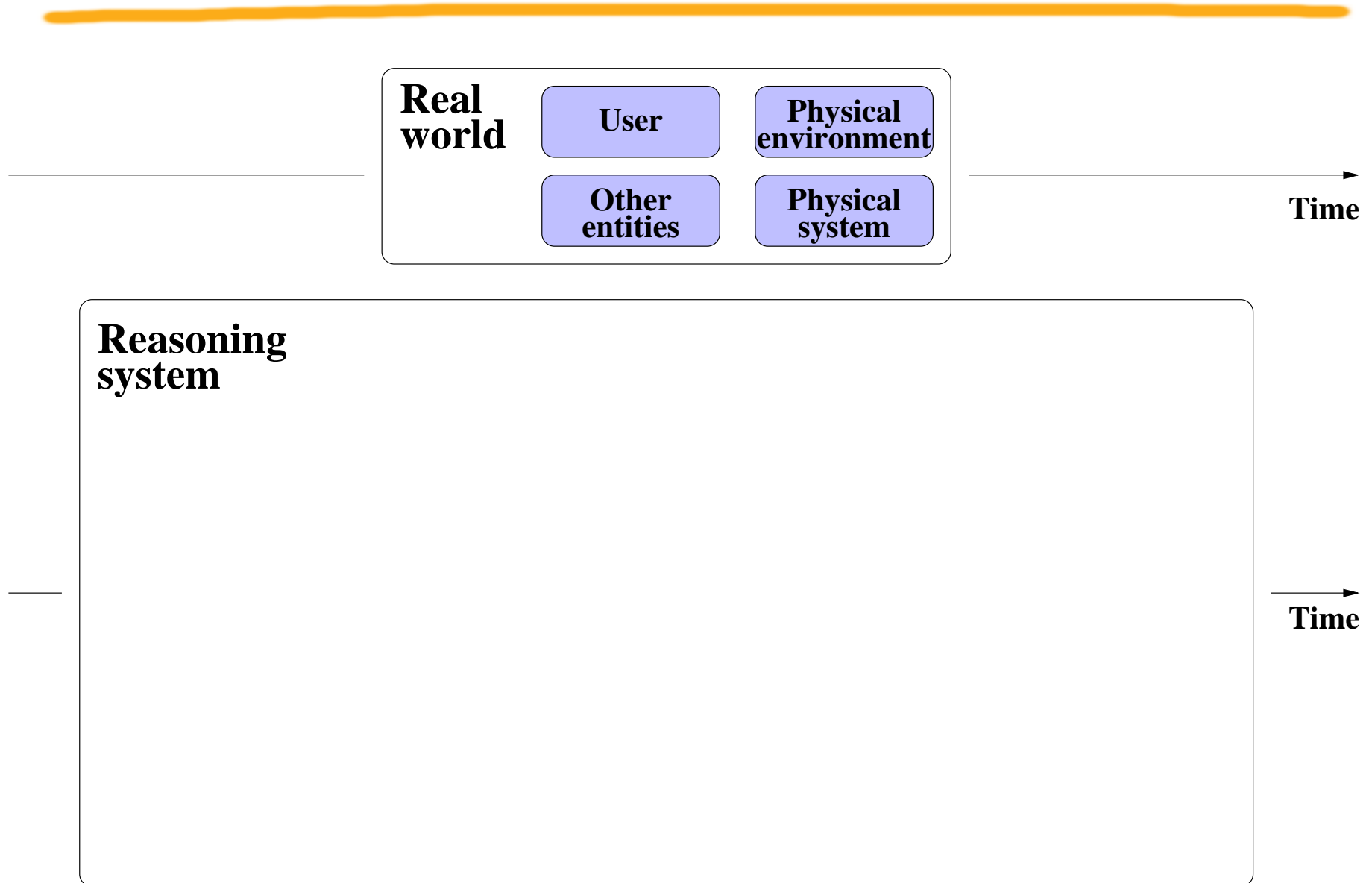
Constraint solving in a dynamic setting



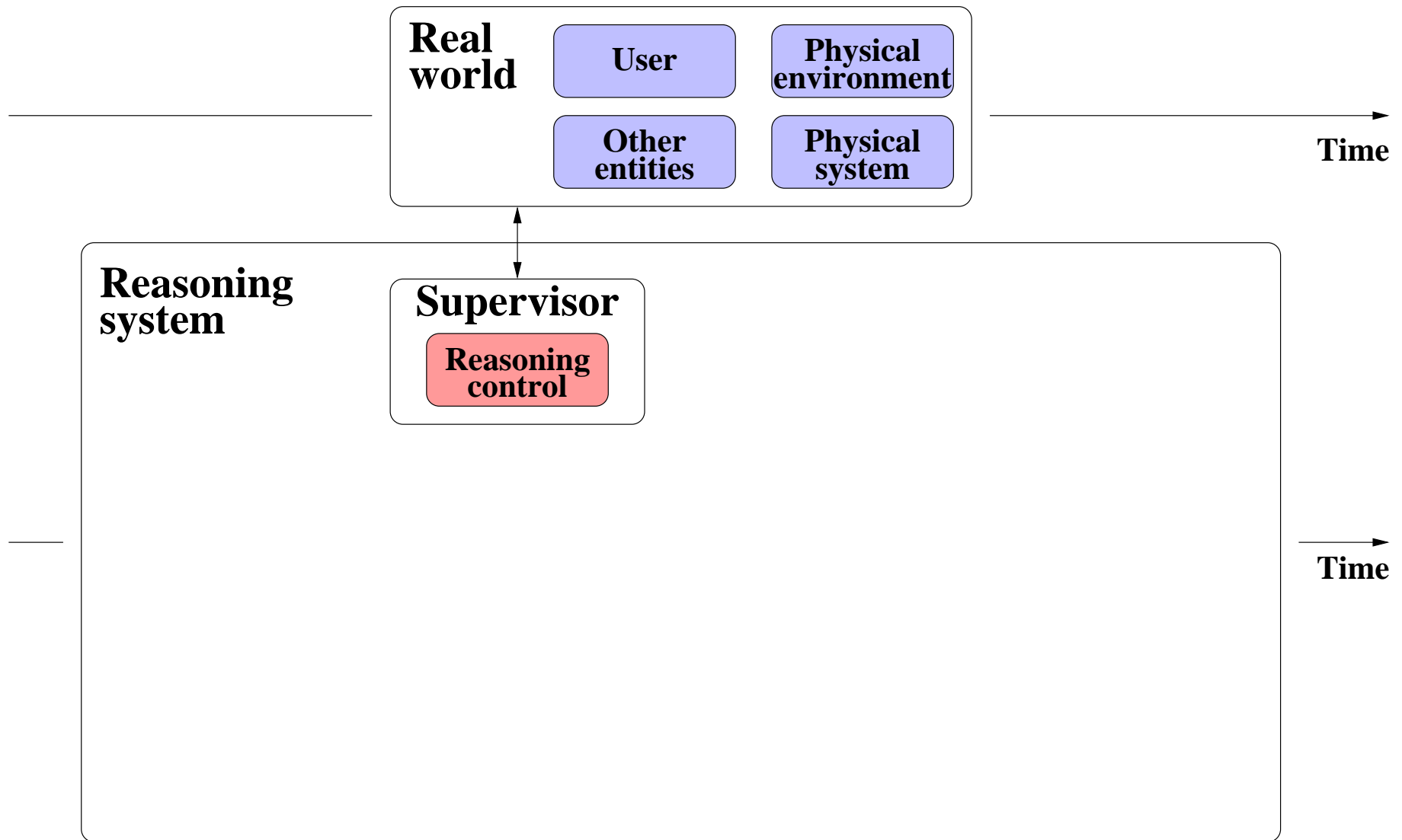
A dynamic world



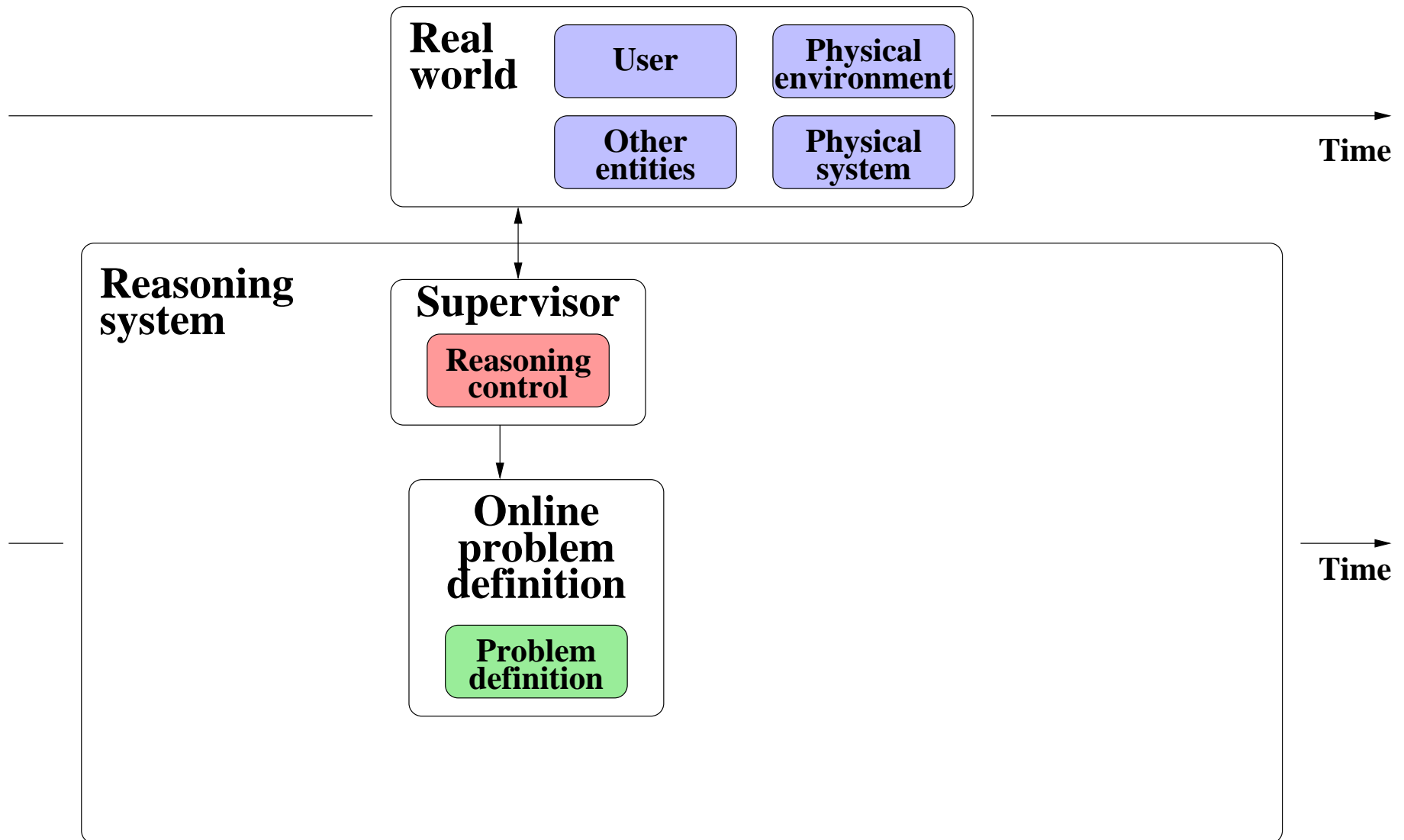
A dynamic reasoning system



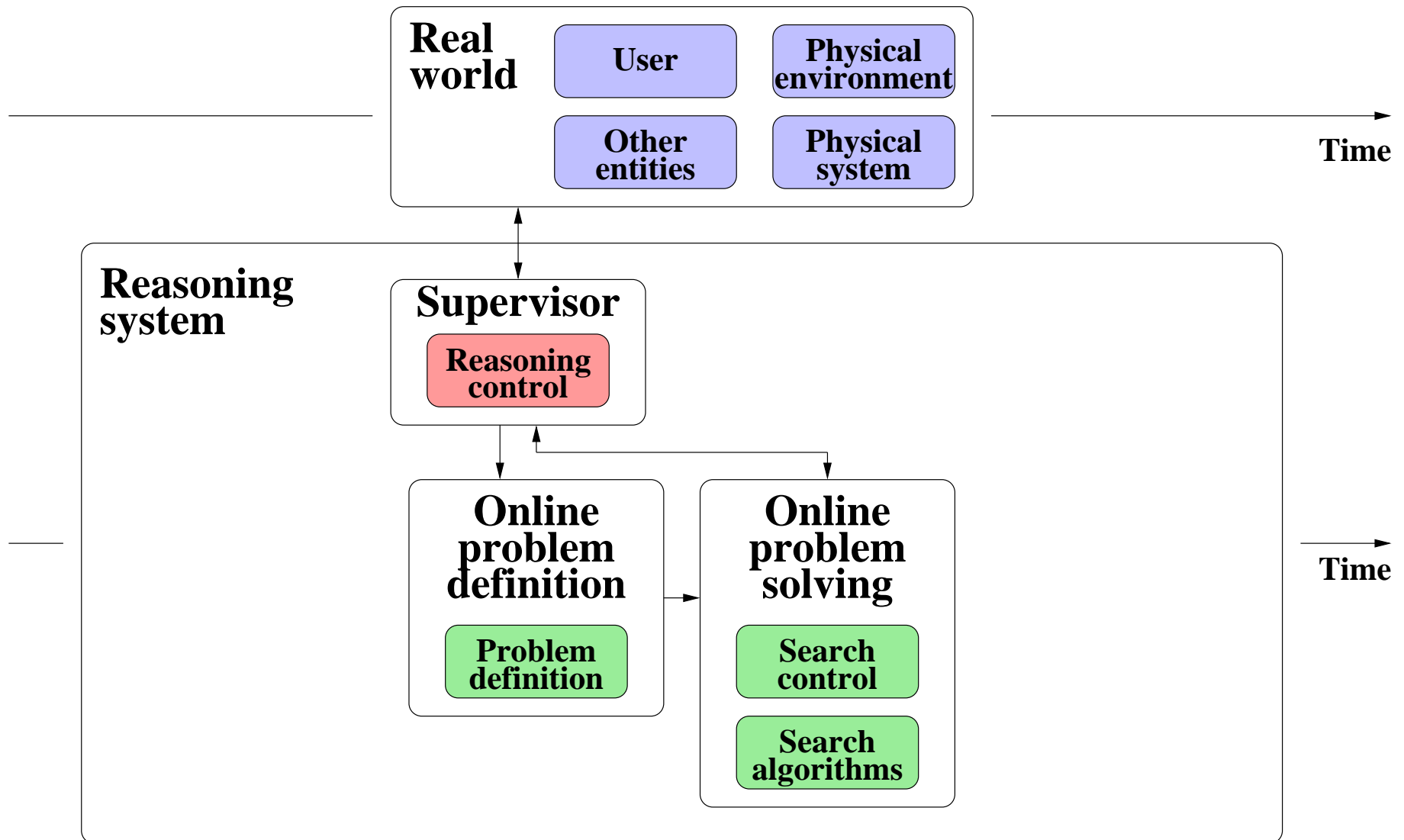
A reasoning system supervisor



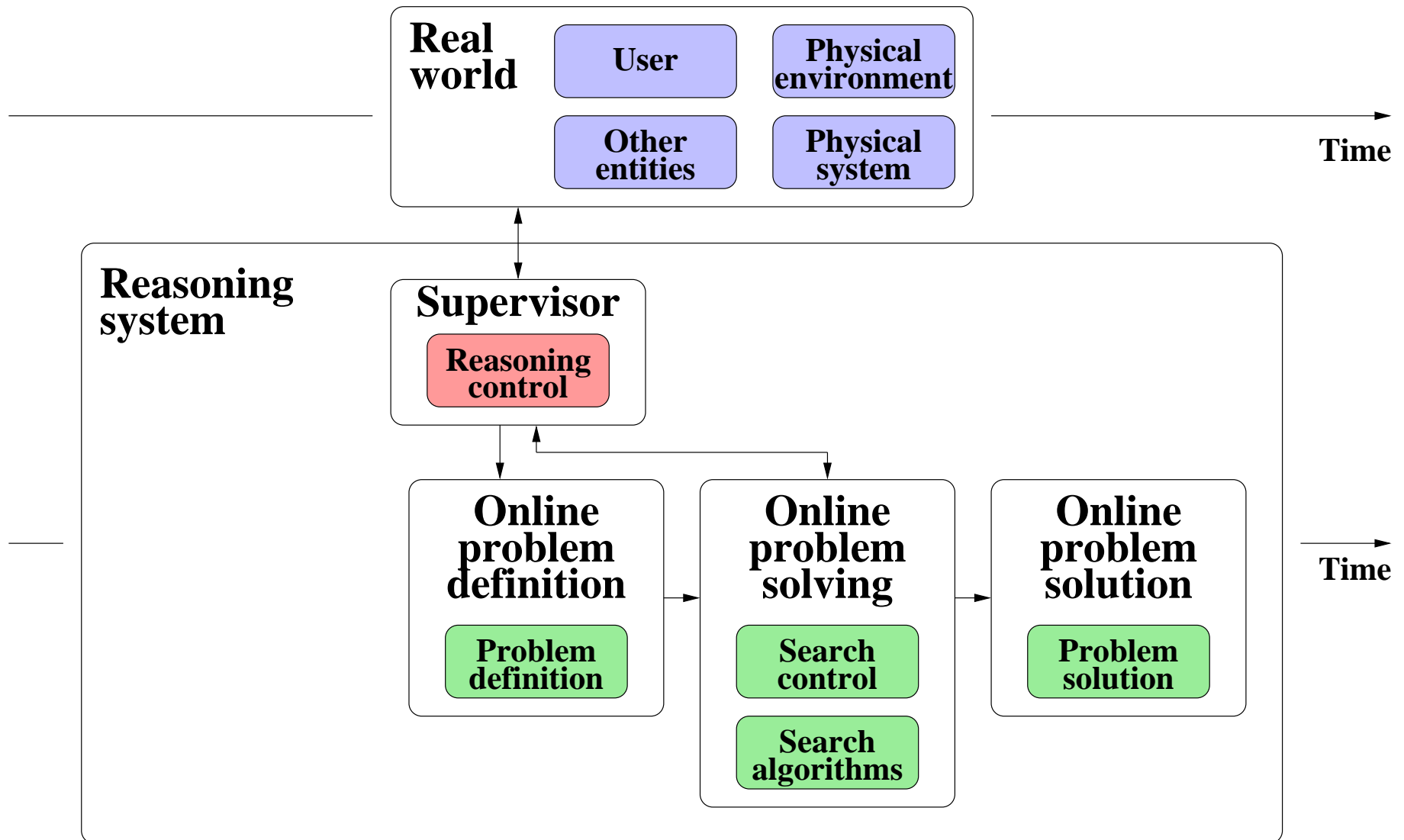
A dynamic problem definition



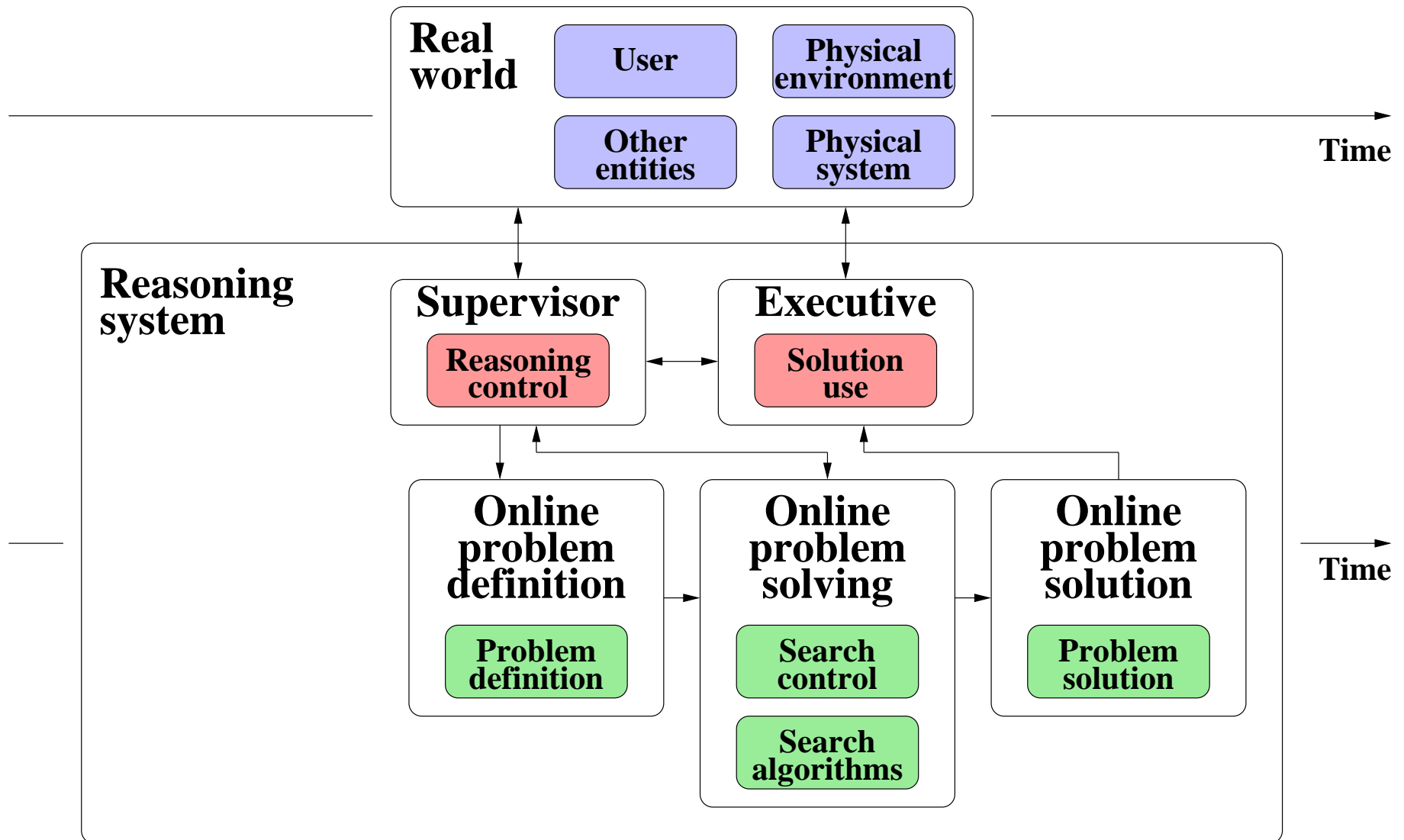
A dynamic problem solving



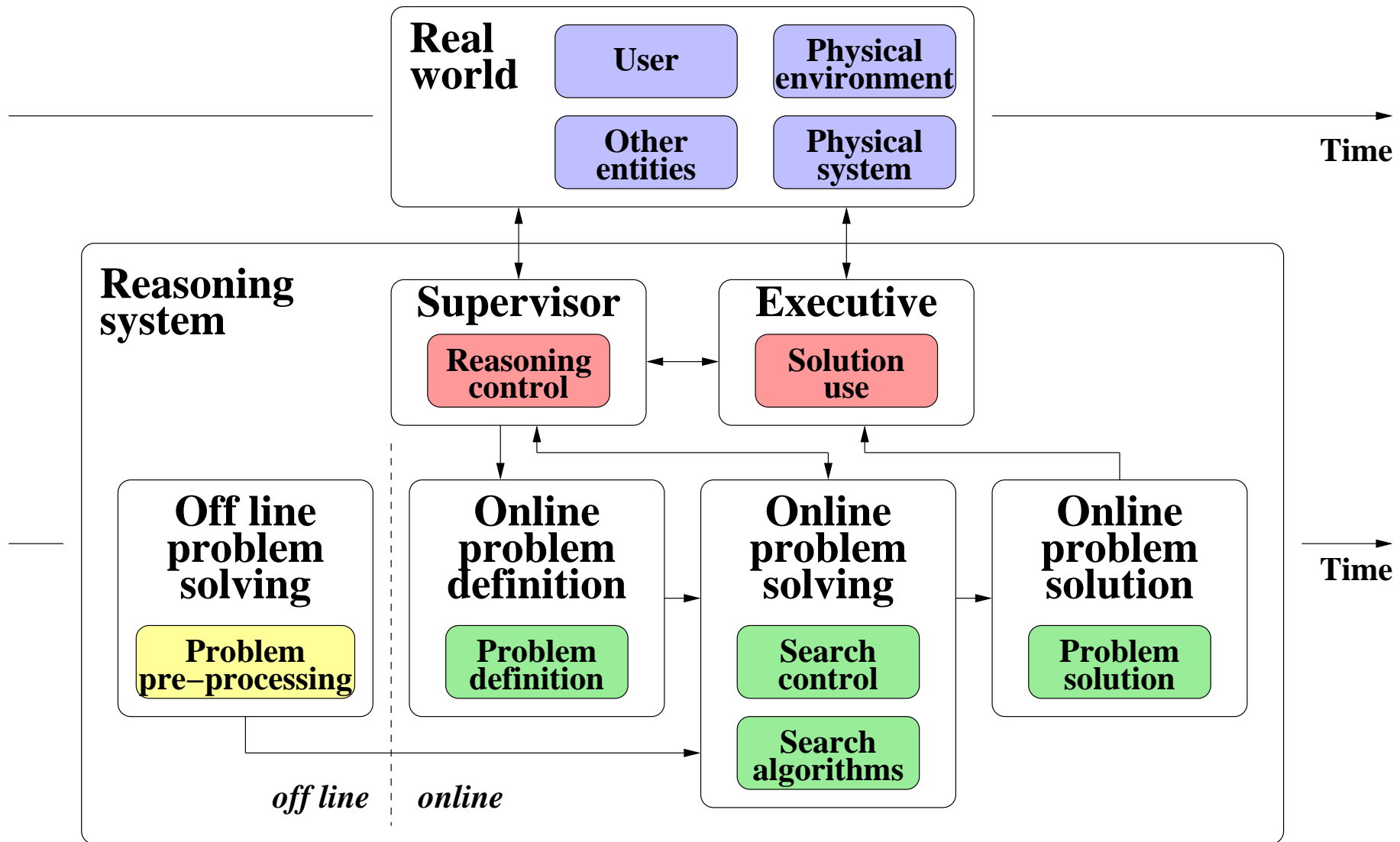
A dynamic problem solution



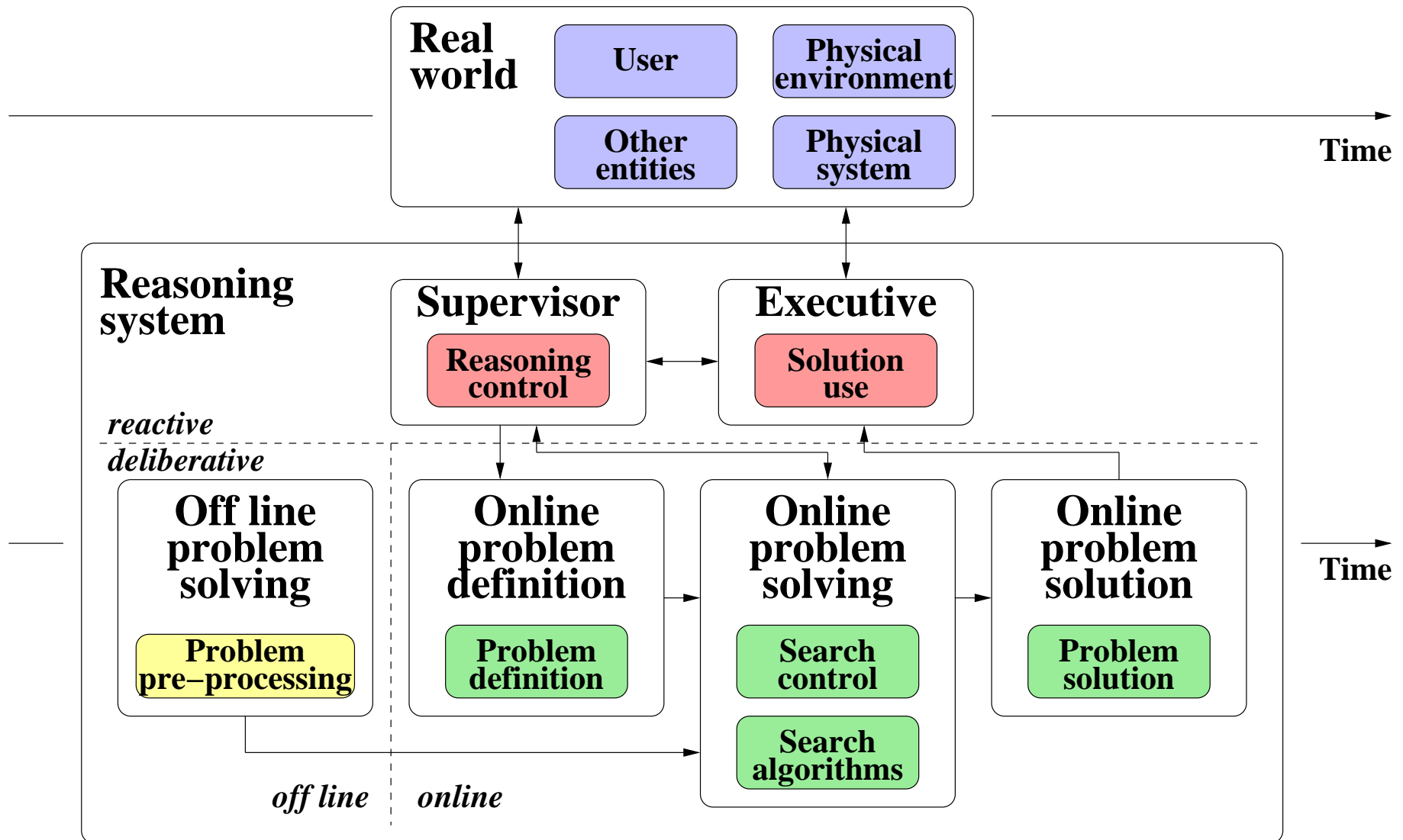
An executive



A static problem processing



Reactive and deliberative components



Main applications with change origins

1. online **planning** or **scheduling**
→ environment and physical system **states**, action **effects**, user **objectives**;

Main applications with change origins

1. online **planning** or **scheduling**
→ environment and physical system **states**, action **effects**, user **objectives**;
2. online **failure diagnosis**, **computer vision**, or **situation tracking**
→ environment and physical system **dynamics**;

Main applications with change origins

1. online **planning** or **scheduling**
→ environment and physical system **states**, action **effects**, user **objectives**;
2. online **failure diagnosis**, **computer vision**, or **situation tracking**
→ environment and physical system **dynamics**;
3. computer-aided **system design** or **configuration**
→ user **requirements** and design **choices**;

Main applications with change origins

1. online **planning** or **scheduling**
→ environment and physical system **states**, action **effects**, user **objectives**;
2. online **failure diagnosis**, **computer vision**, or **situation tracking**
→ environment and physical system **dynamics**;
3. computer-aided **system design** or **configuration**
→ user **requirements** and design **choices**;
4. **user interface** management
→ user **actions**;

Main applications with change origins

1. online **planning** or **scheduling**
→ environment and physical system **states**, action **effects**, user **objectives**;
2. online **failure diagnosis**, **computer vision**, or **situation tracking**
→ environment and physical system **dynamics**;
3. computer-aided **system design** or **configuration**
→ user **requirements** and design **choices**;
4. **user interface** management
→ user **actions**;
5. **interactive** or **distributed** problem solving
→ user **choices**, other entities **choices**;

Main applications with change origins

1. online **planning** or **scheduling**
→ environment and physical system **states**, action **effects**, user **objectives**;
2. online **failure diagnosis**, **computer vision**, or **situation tracking**
→ environment and physical system **dynamics**;
3. computer-aided **system design** or **configuration**
→ user **requirements** and design **choices**;
4. **user interface** management
→ user **actions**;
5. **interactive** or **distributed** problem solving
→ user **choices**, other entities **choices**;
6. **interactive** problem **specification**; constraint program **debugging**
→ user **requirements**, problem **statement**;

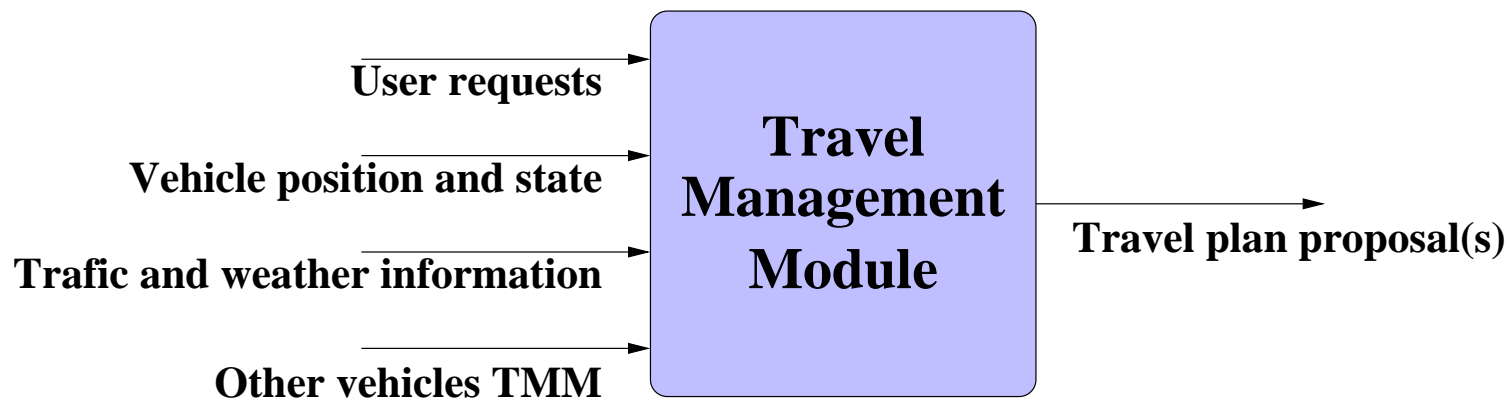
Main applications with change origins

1. online **planning** or **scheduling**
→ environment and physical system **states**, action **effects**, user **objectives**;
2. online **failure diagnosis**, **computer vision**, or **situation tracking**
→ environment and physical system **dynamics**;
3. computer-aided **system design** or **configuration**
→ user **requirements** and design **choices**;
4. **user interface** management
→ user **actions**;
5. **interactive** or **distributed** problem solving
→ user **choices**, other entities **choices**;
6. **interactive** problem **specification**; constraint program **debugging**
→ user **requirements**, problem **statement**;
7. **unordered search**
→ variable **assignments** or **unassignments**.

A reference application

A tool for **travel management**, embedded in a car, dedicated to the management of:

- the **route**;
- the tank **refueling** and car **maintenance**;
- the restaurant, hotel, and visit **reservations**;
- the **rendez-vous** with other people.



The main requirements in a dynamic setting

1. to minimize the **number of successive problem solvings**
→ to produce **robust** solutions i.e., solutions which **resist changes** as much as possible;

The main requirements in a dynamic setting

1. to minimize the **number of successive problem solvings**
→ to produce **robust** solutions i.e., solutions which **resist changes** as much as possible;
2. when a new problem solving is necessary,
to minimize as much as possible the **changes in the current solution**
→ to produce **flexible** and **stable** solutions;

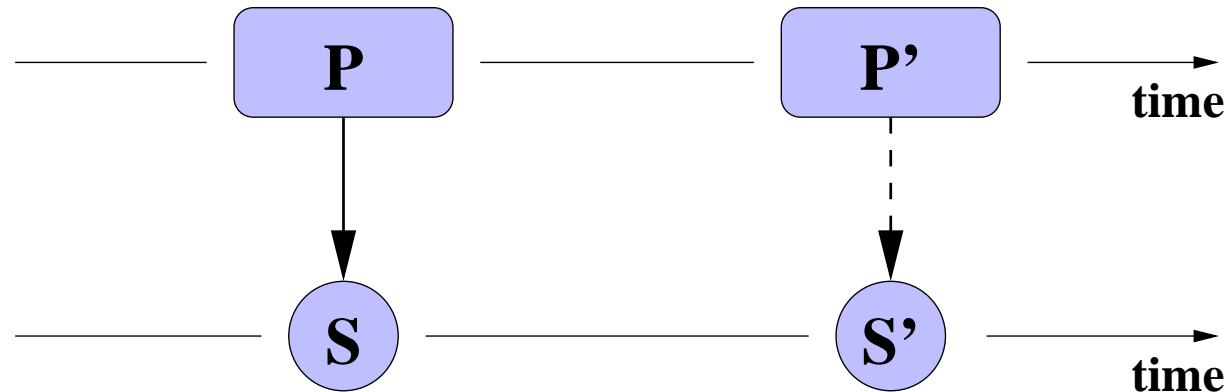
The main requirements in a dynamic setting

1. to minimize the **number of successive problem solvings**
→ to produce **robust** solutions i.e., solutions which **resist changes** as much as possible;
2. when a new problem solving is necessary,
to minimize as much as possible the **changes in the current solution**
→ to produce **flexible** and **stable** solutions;
3. when a new problem solving is necessary,
to perform it as **efficiently** as possible
→ to meet hard or soft **deadlines**;

The main requirements in a dynamic setting

1. to minimize the **number of successive problem solvings**
→ to produce **robust** solutions i.e., solutions which **resist changes** as much as possible;
2. when a new problem solving is necessary,
to minimize as much as possible the **changes in the current solution**
→ to produce **flexible** and **stable** solutions;
3. when a new problem solving is necessary,
to perform it as **efficiently** as possible
→ to meet hard or soft **deadlines**;
4. to keep producing **consistent** and **optimal** solutions.

A more formal definition



Nature of the changes: any component of the problem definition:

- set of **variables**: additions or removals;
- **domain** definitions: extension or restriction;
- set of **constraints**: additions or removals;
- **constraint** variables: additions or removals;
- **constraint** definitions: extension or restriction.

Basic components:
Constraint addition
or **removal**

Do not confuse with

- **Conditional** or **composite** CSPs, whose solutions do not have all the same structure: this structure depends on choices for some variables
→ changes from a structure to another one are included **in the problem definition**;
- **Open** or **interactive** CSPs, whose variable values can be acquired during problem solving
→ variable domains are defined **at solver's request**;
- CSPs that allow **dynamic systems** to be modelled, for example when representing planning/scheduling or situation tracking problems
→ system dynamics is included **in a static problem definition**.

With the problems we consider, changes come **from outside the problem definition**.

Two main classes of methods

- **Reactive** methods:
 - to use **no model** of the future changes;
 - when a change occurs, to **avoid** as much as possible **reasoning** again **from scratch**;

- **Proactive** methods:
 - to use **a model** of the future changes;
 - to be as **robust** as possible **facing** the possible **changes**.

Tutorial outline

1. **Setting** and **requirements**;
2. **Reactive** methods \Leftarrow
 - (a) **solution** reuse;
 - (b) **reasoning** reuse;
3. **Proactive** methods:
 - (a) **robust** solutions;
 - (b) **flexible** solutions;
4. **Research directions.**

Reactive methods

Main idea: to **reuse** as much as possible **previous reasonings**.

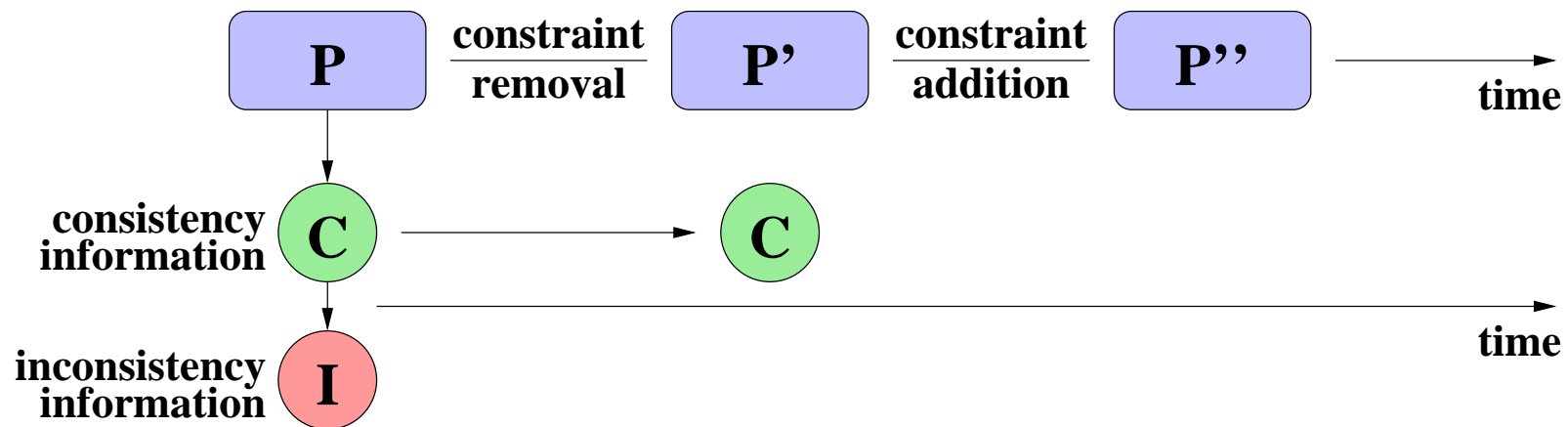
What is produced when reasoning on a problem, either from **constraint propagation** or from **search**? Information about:

1. **problem** consistency or inconsistency;
2. consistency or inconsistency of **complete assignments**;
3. **global** consistency or inconsistency of **partial assignments**;
4. **local** consistency or inconsistency of **partial assignments**.

To keep in mind

- **consistency** information is preserved by constraint **removals**:
what was consistent remains consistent;
- **inconsistency** information is preserved by constraint **additions**:
what was inconsistent remains inconsistent.

If a constraint removal is followed by a constraint addition, or if both occur in parallel, everything is **lost**, if nothing is done to **preserve information**.



Two classes of reactive methods

- **solution reuse**: to use **previous solutions** (consistency information), either as **heuristic**, or as **search starting point**
 - implicit assumption: a solution for the current problem exists **not too far** from the solutions which have been found for the previous problems, because problems are **close** to each other
 - using previous solutions tends to favour **efficiency** and solution **stability**;
- **reasoning reuse**: to reuse, among the **previously computed nogoods** (globally inconsistent partial assignments, inconsistency information), those that with certainty remain **valid**
 - to **save** as much as possible the results of the previous work: constraint propagation, search
 - can be seen as a form of **learning**.

Solution reuse

1. **solution perturbation**

→ synthesis of a **sequence of methods** which allows consistency to be straightforwardly recovered from a previous solution; limited to **functional** constraints;

2. **tree search**: depth first search, limited discrepancy search ...

→ use of the previous solution as a **heuristic**;

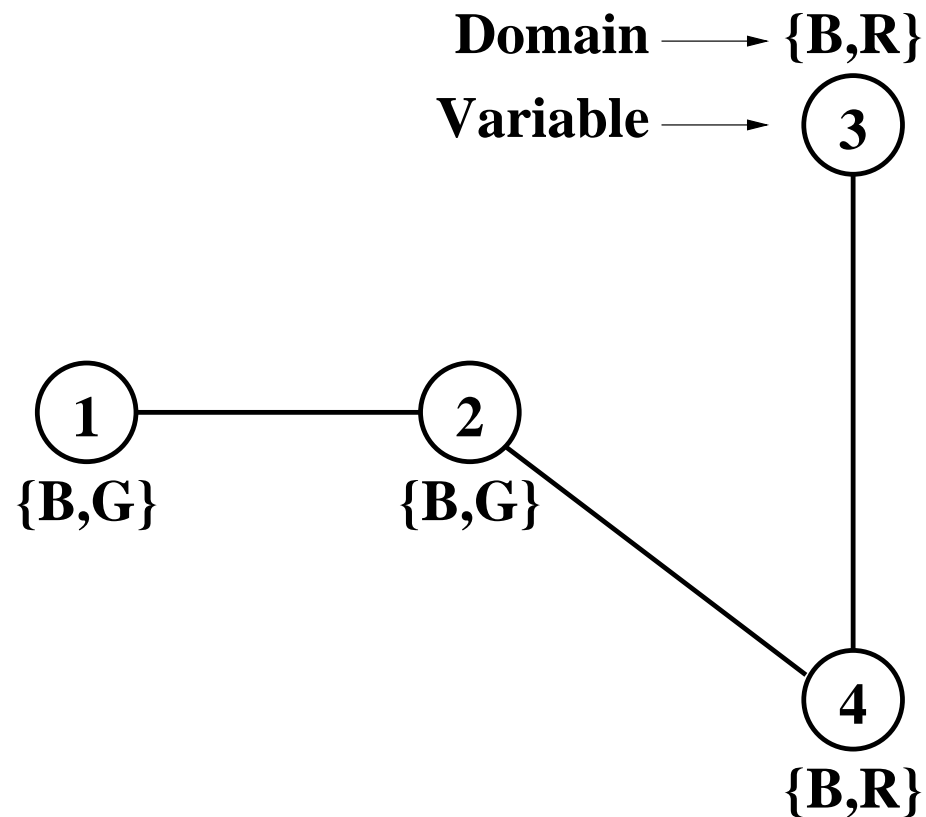
3. **local search**: heuristic repair, tabu search ...

→ use of the previous solution as a **starting point**;

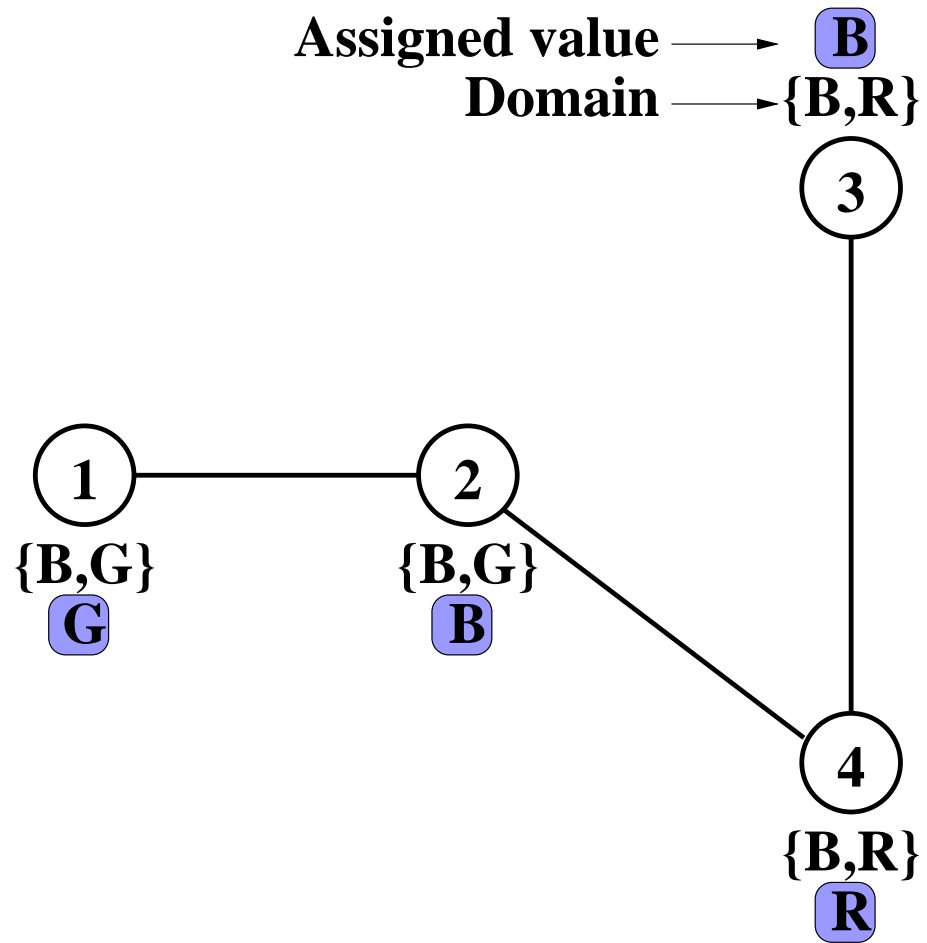
4. **variable unassignments** and **reassignments**: local changes

→ search for a **sequence of variable unassignments** and **reassignments** which allows consistency to be recovered from a previous solution.

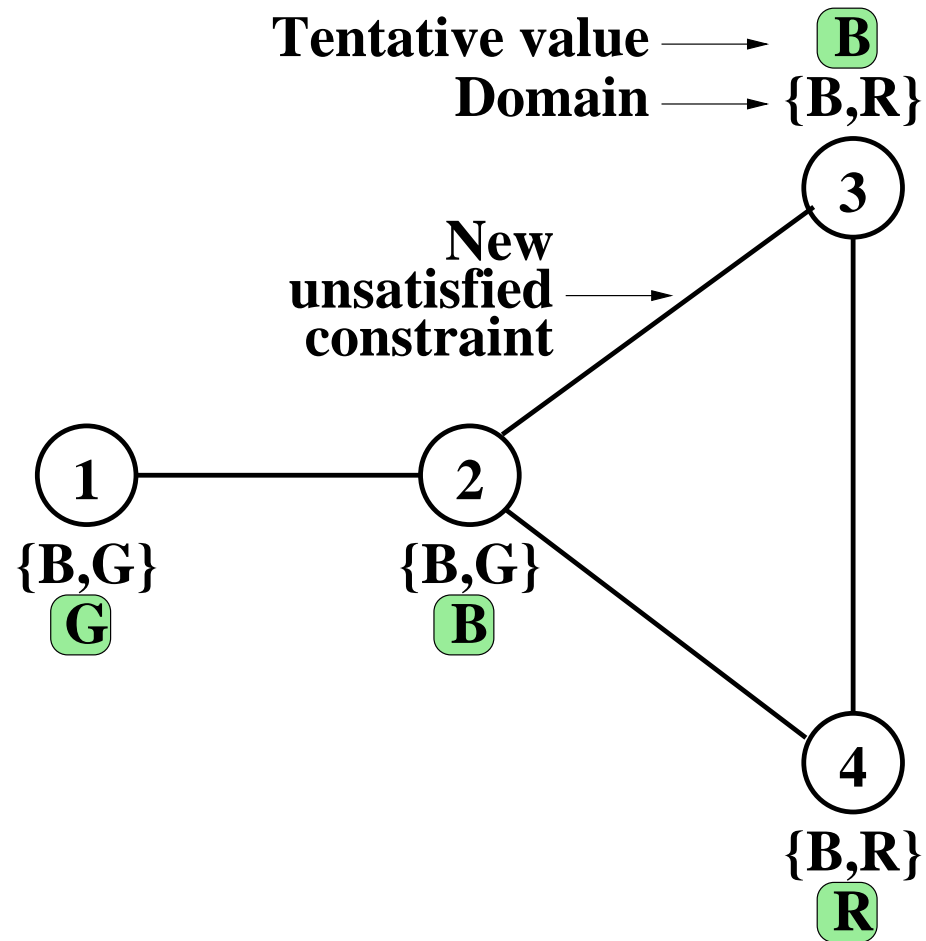
An example with a graph colouring problem and the LC algorithm



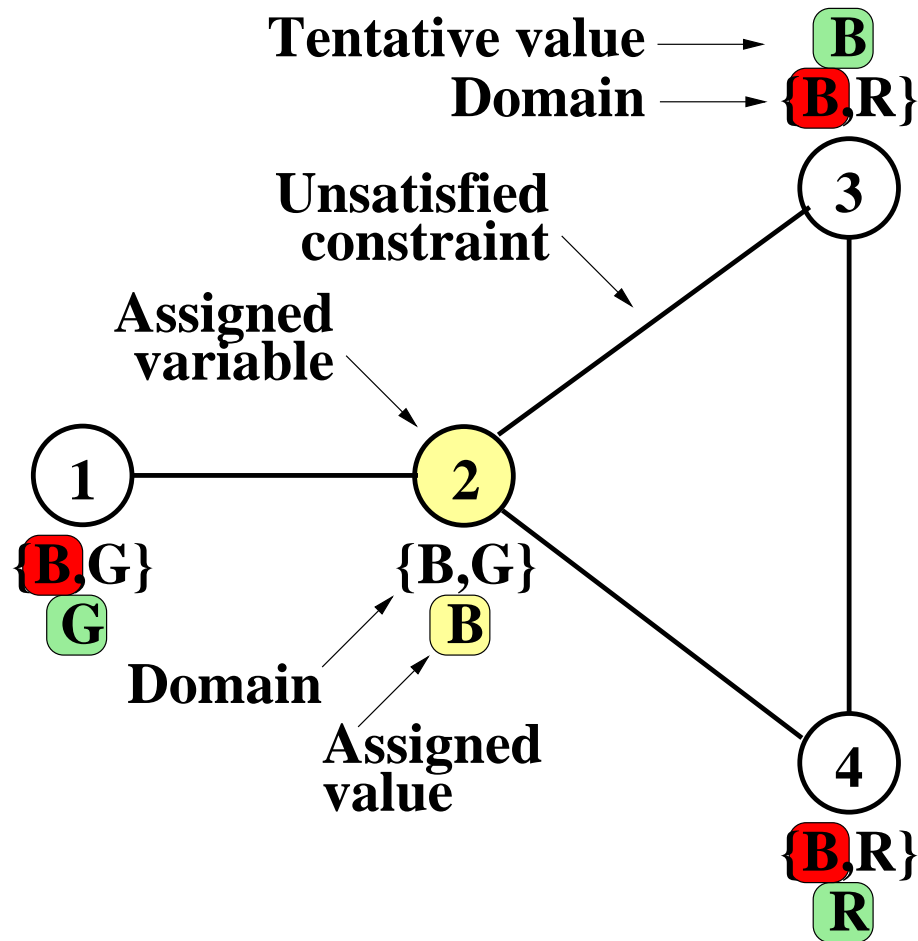
A solution



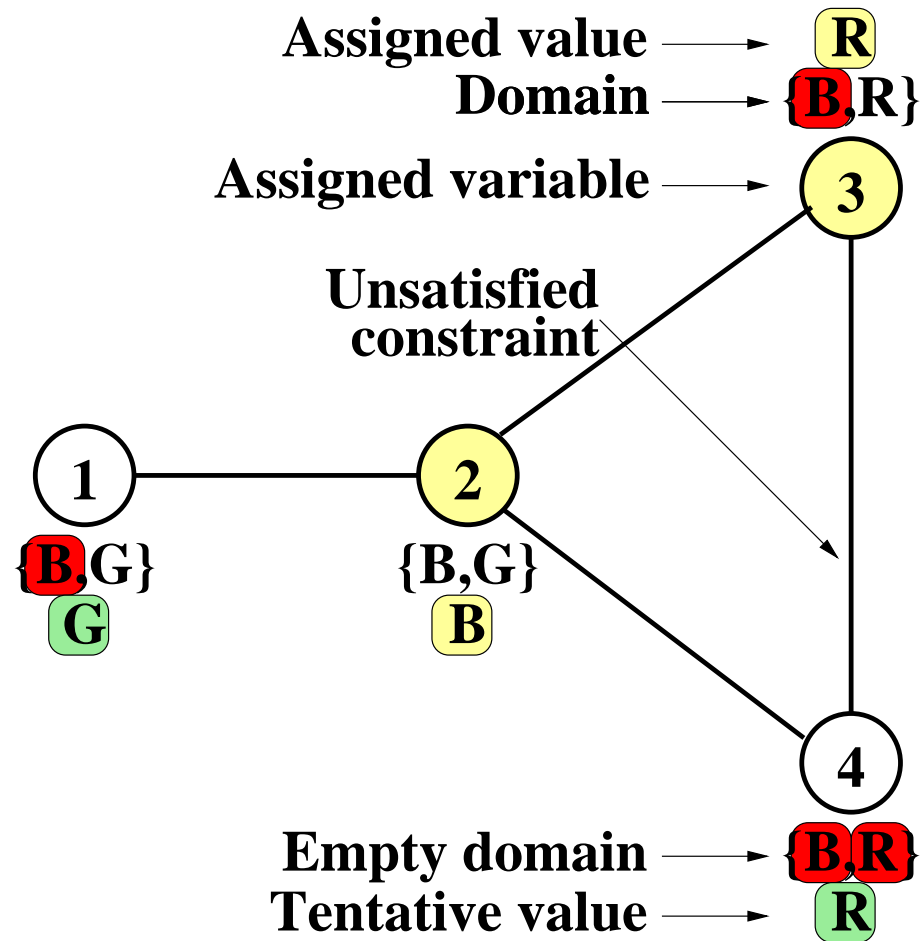
A new constraint between variables 2 and 3



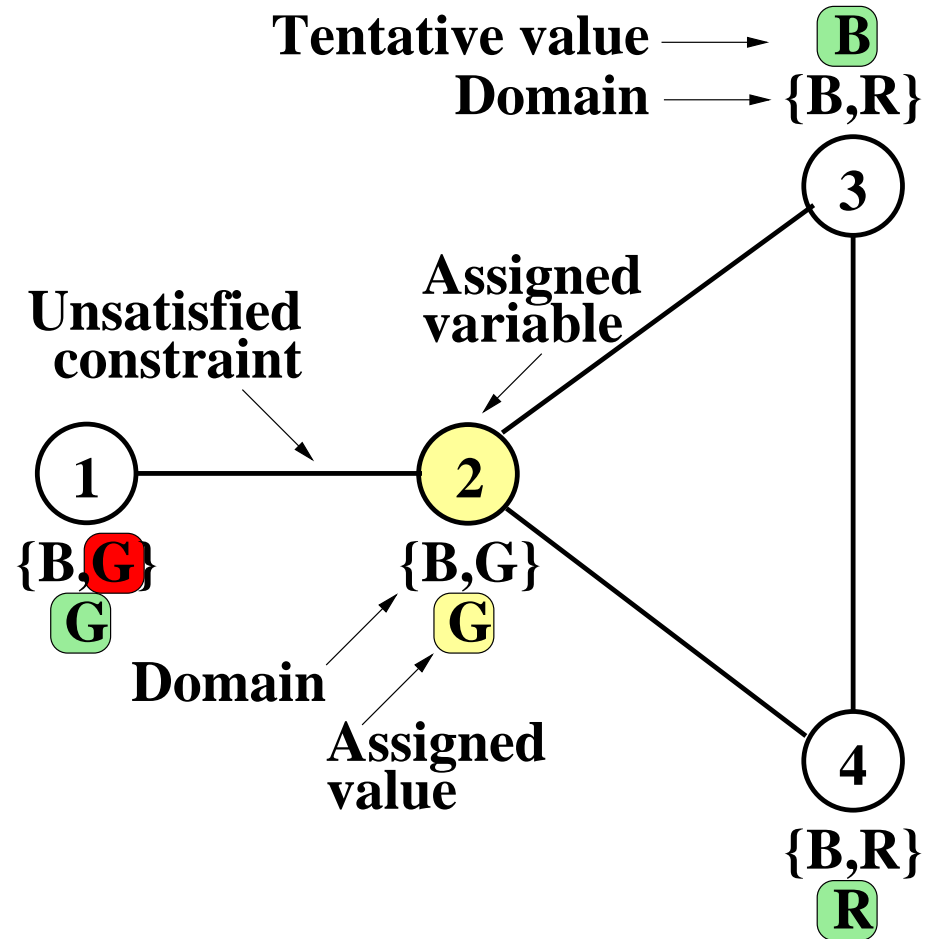
Let us choose variable 2 and assign it value B



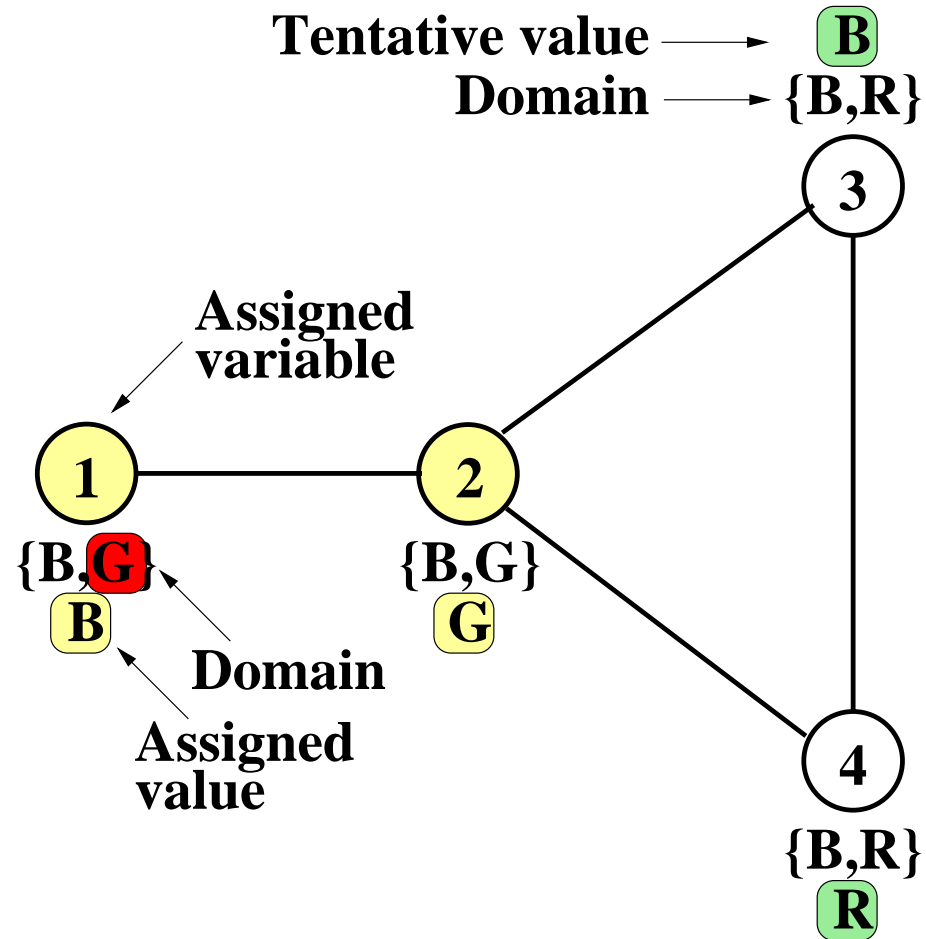
Let us assign variable 3 value R



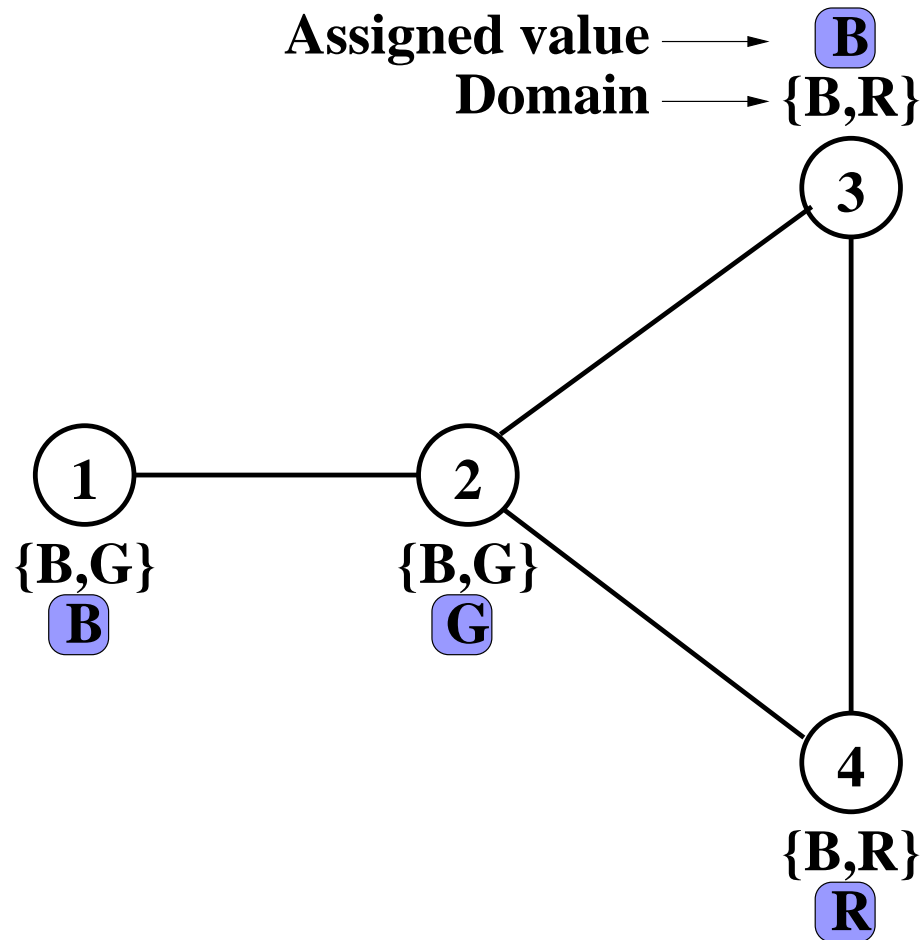
Let us come back to variable 2 and assign it value G



Let us assign variable 1 value B



A new solution



Reasoning reuse

Difficulty: constraint **removal**.

A **two-phase** process, in case of **constraint removal**:

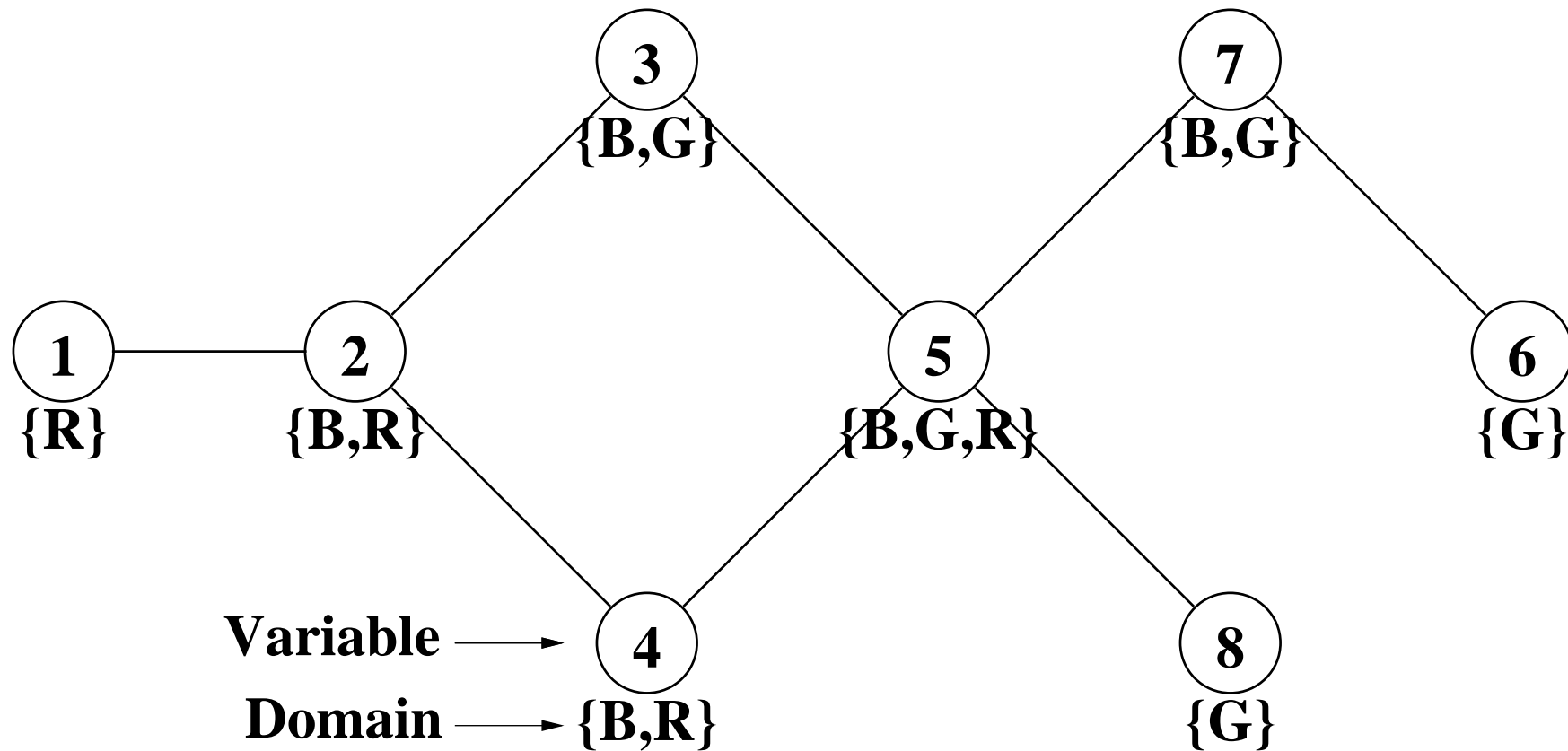
1. to **remove** only nogoods which became questionable;
2. to **propagate** constraints again from the nogoods which remain after phase 1.

Challenge: to limit as much as possible nogood removals in phase 1.

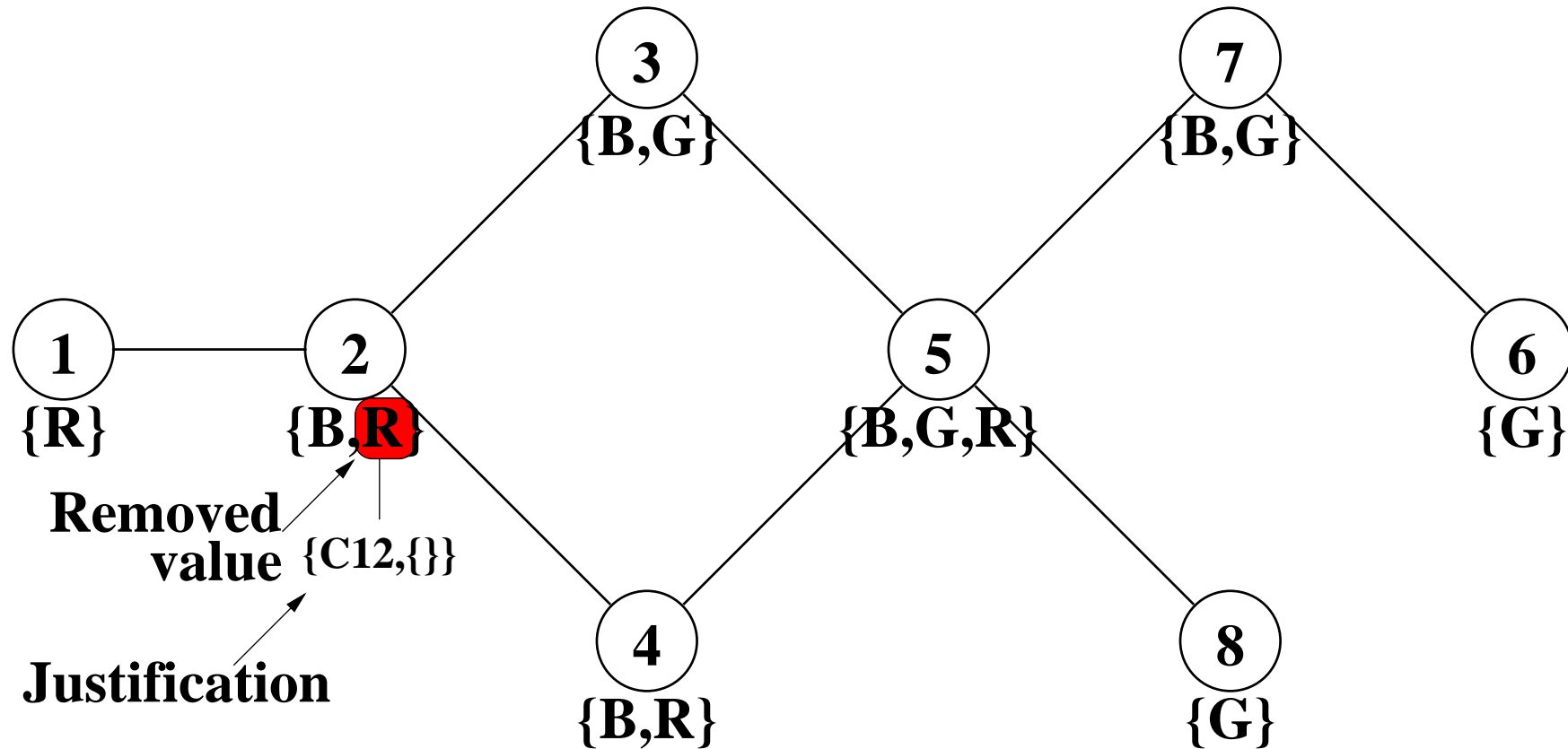
Two classes of reasoning reuse techniques

- **graph-based** techniques:
 - to use only **constraint graph** information to decide about the nogoods which became questionable;
- **justification-based** techniques:
 - to use **justifications** which have been computed and recorded during the propagation phase to decide about the nogoods which became questionable:
 - **minimal** justifications: for each nogood, recording of the constraint or set of constraints whose checking produced it;
 - **complete** justifications (**explanations**): for each nogood, recording of a set of constraints which implies it.

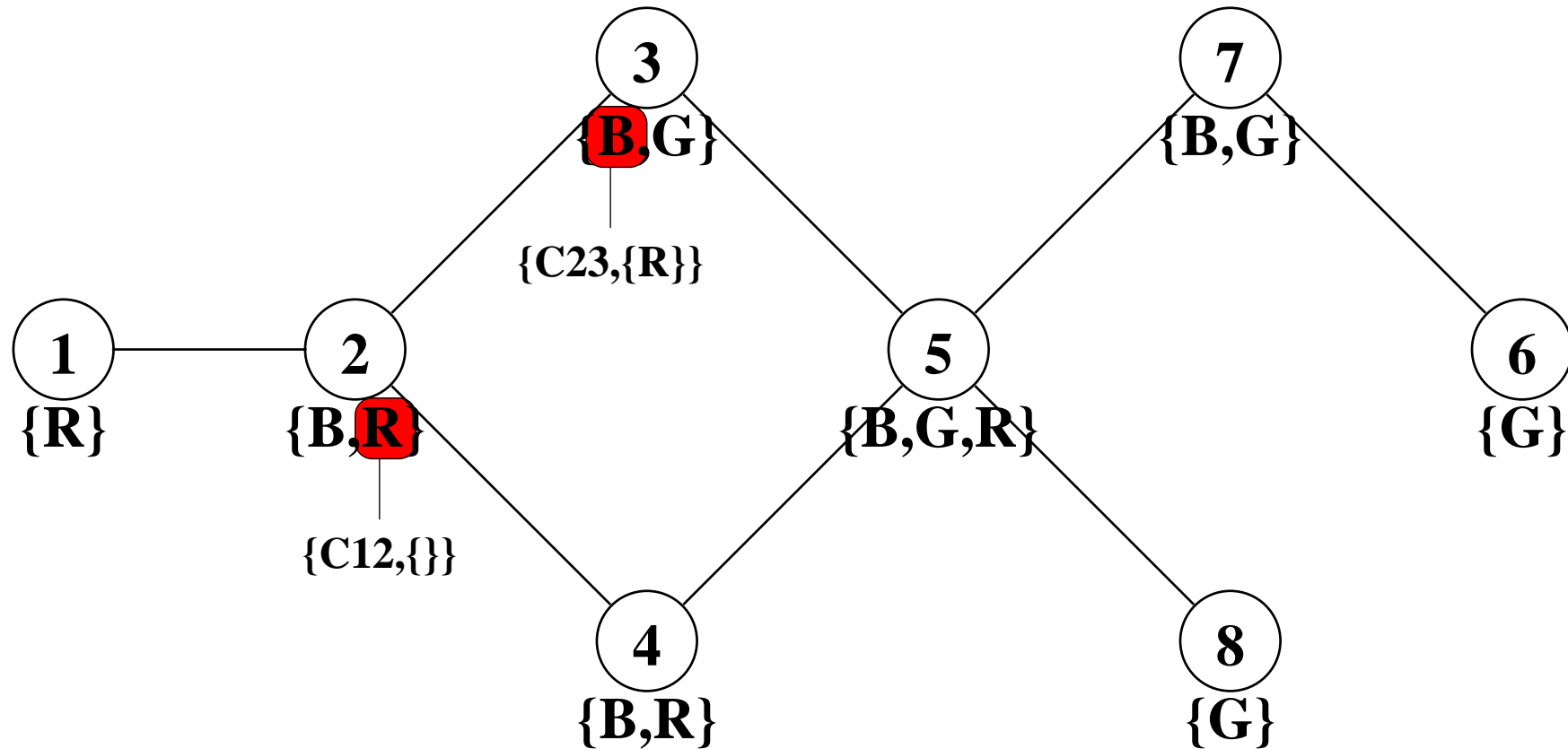
An example with a graph colouring problem and with complete justifications



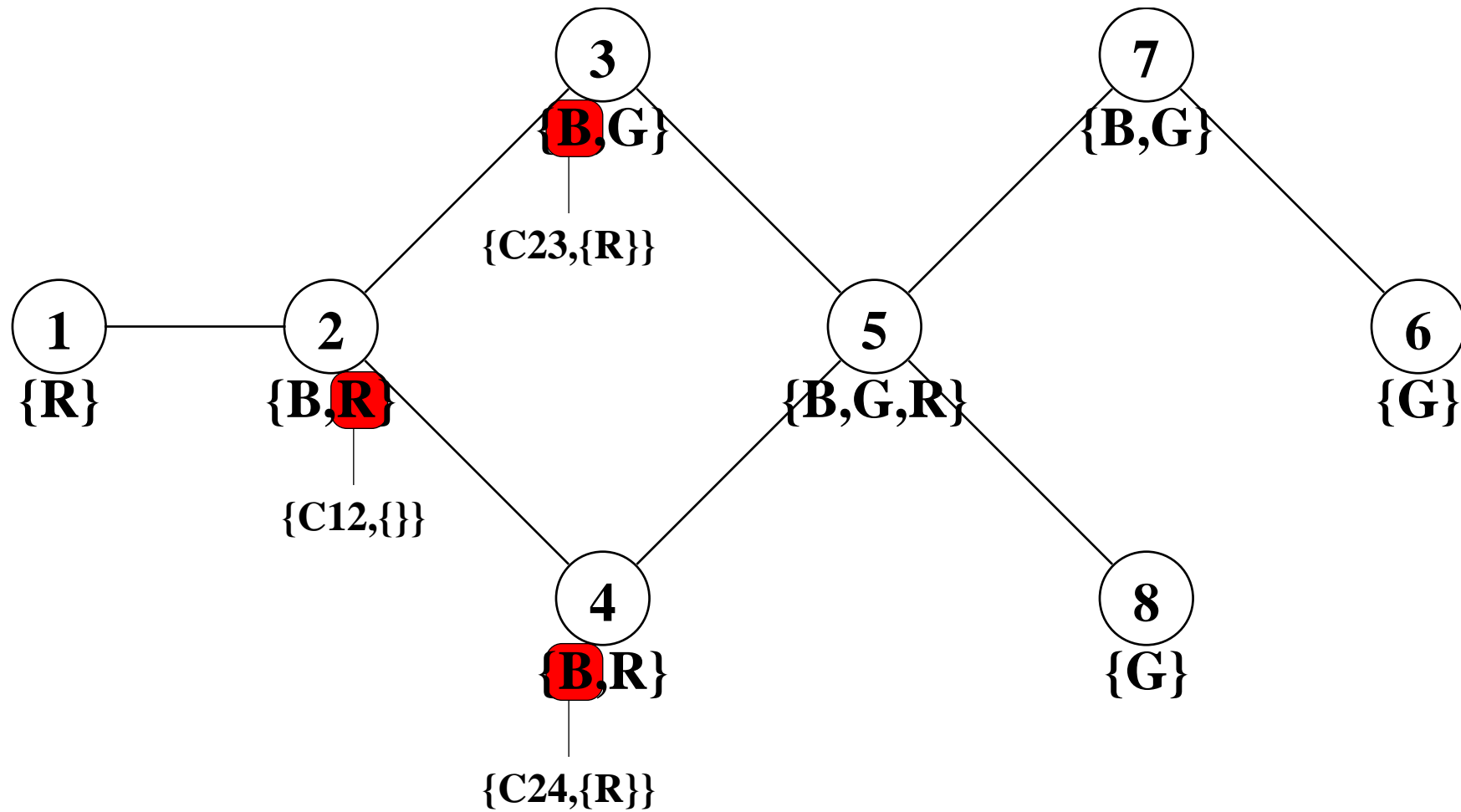
Arc-consistency enforcing on variable 2



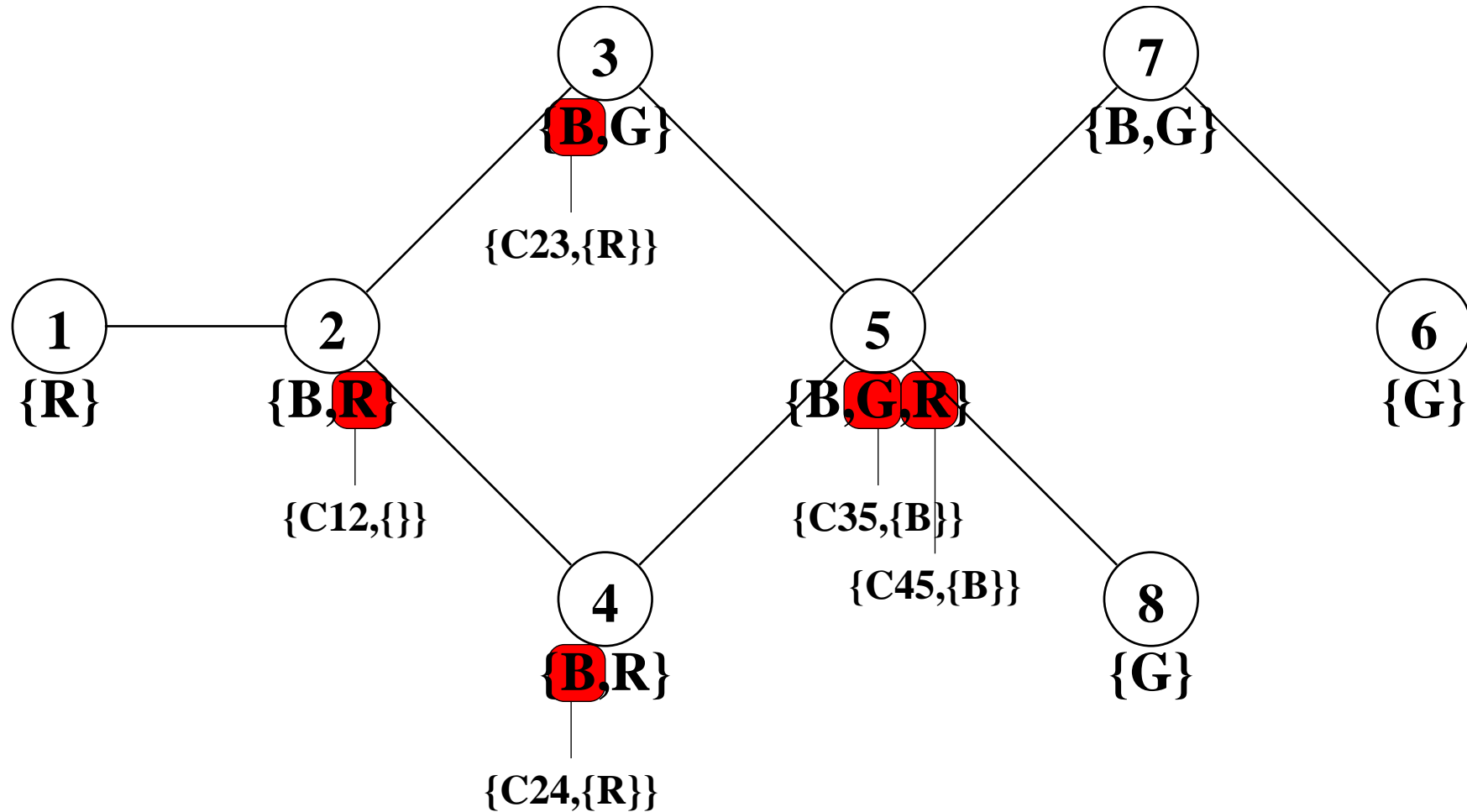
Arc-consistency enforcing on variable 3



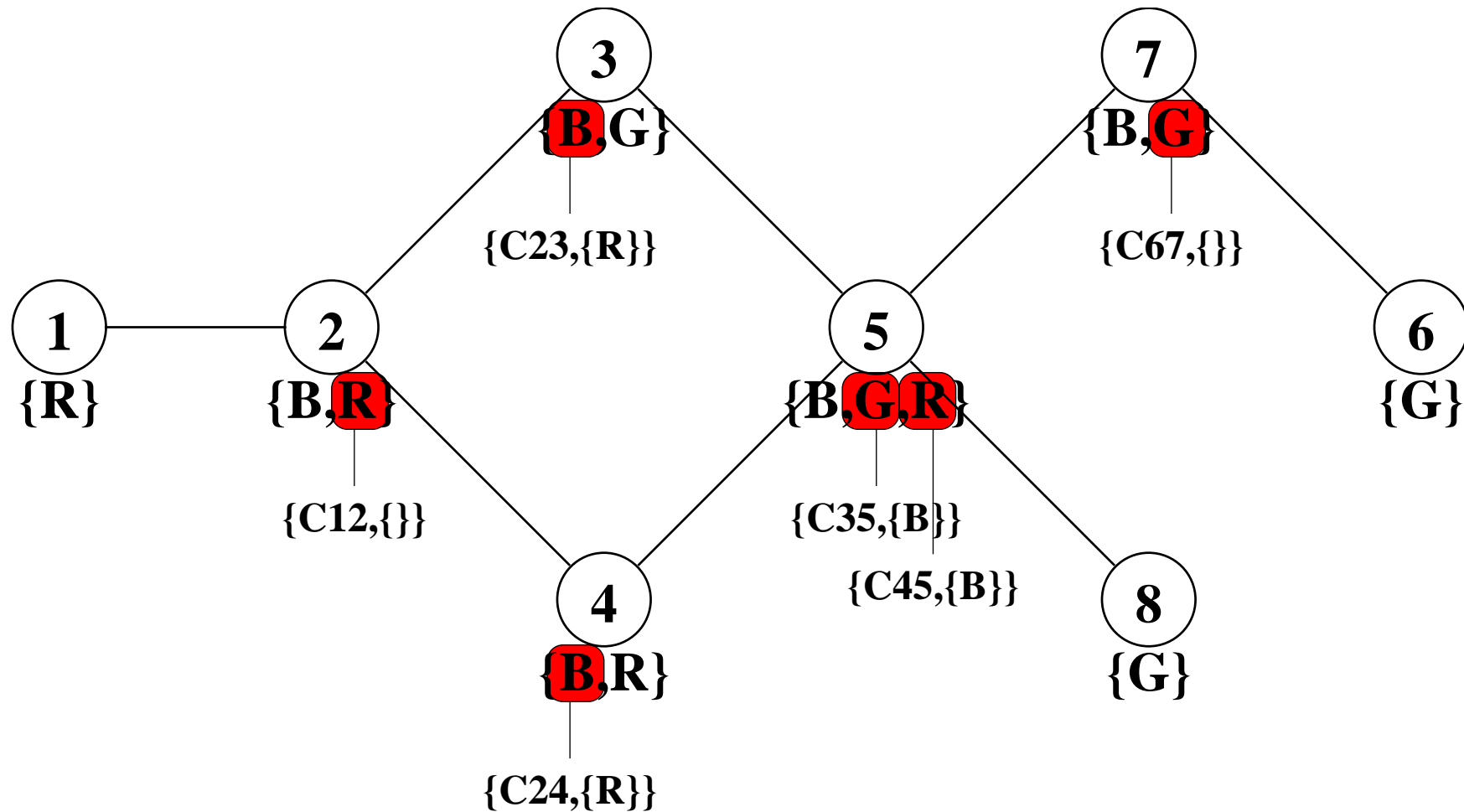
Arc-consistency enforcing on variable 4



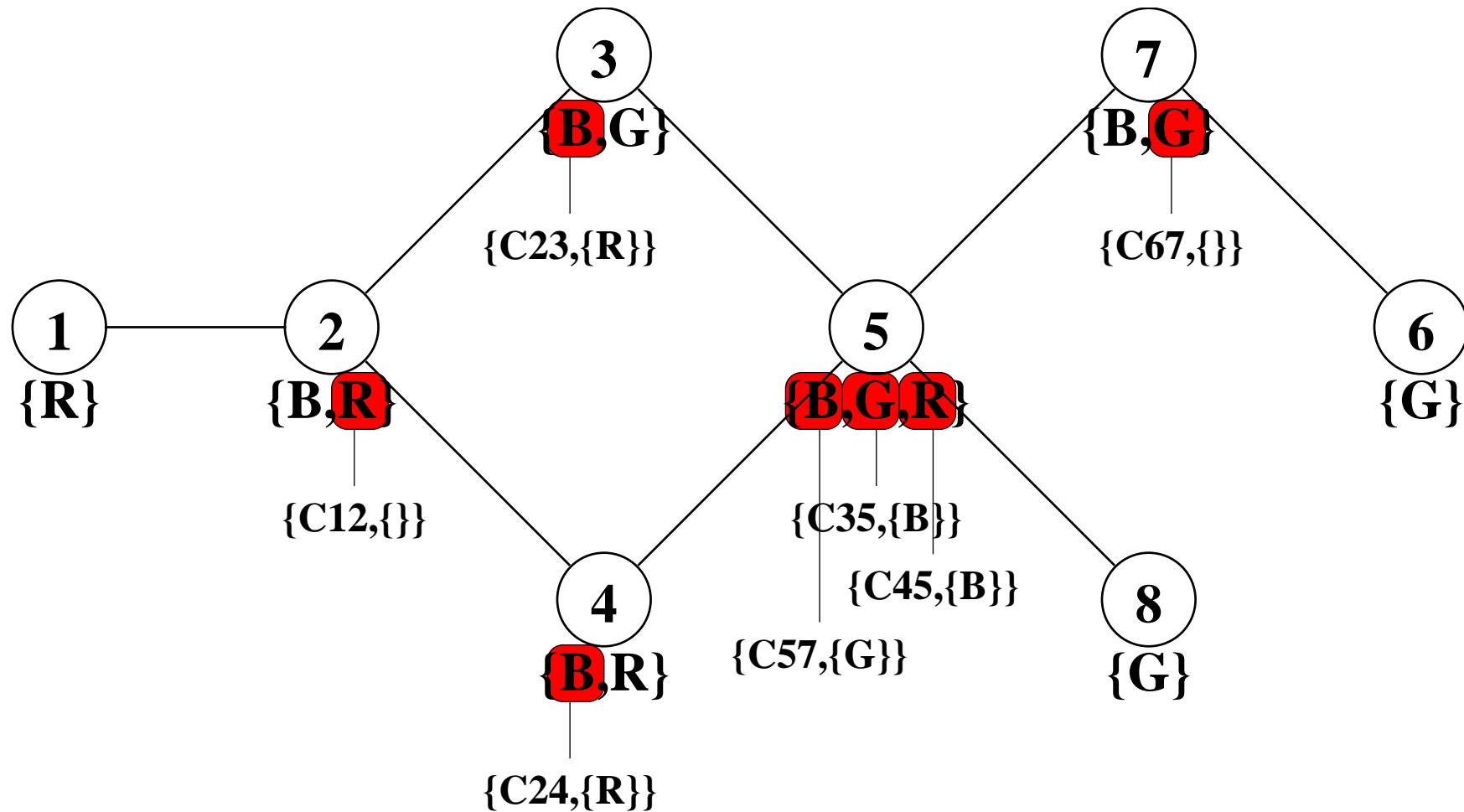
Arc-consistency enforcing on variable 5



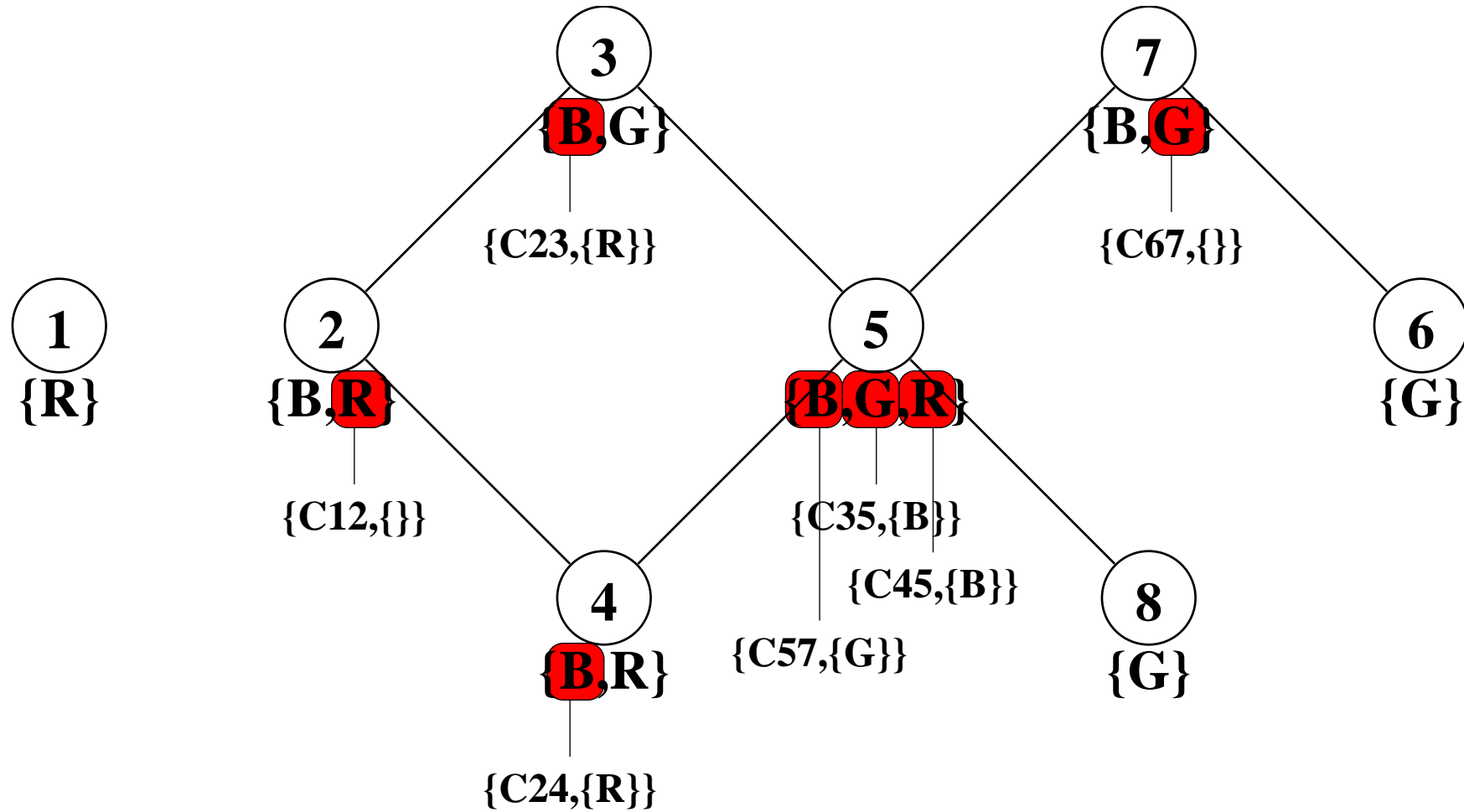
Arc-consistency enforcing on variable 7



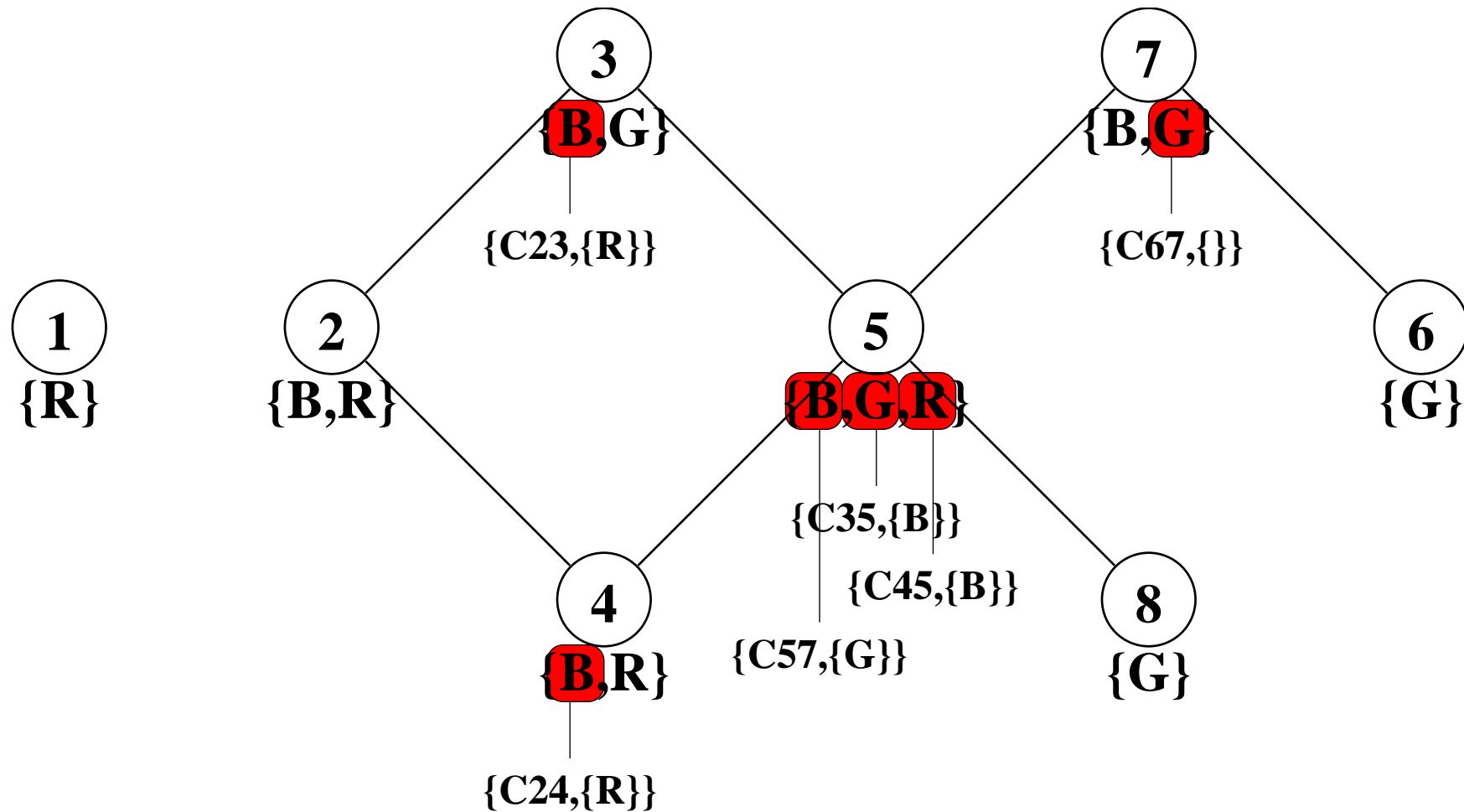
Arc-consistency enforcing on variable 5



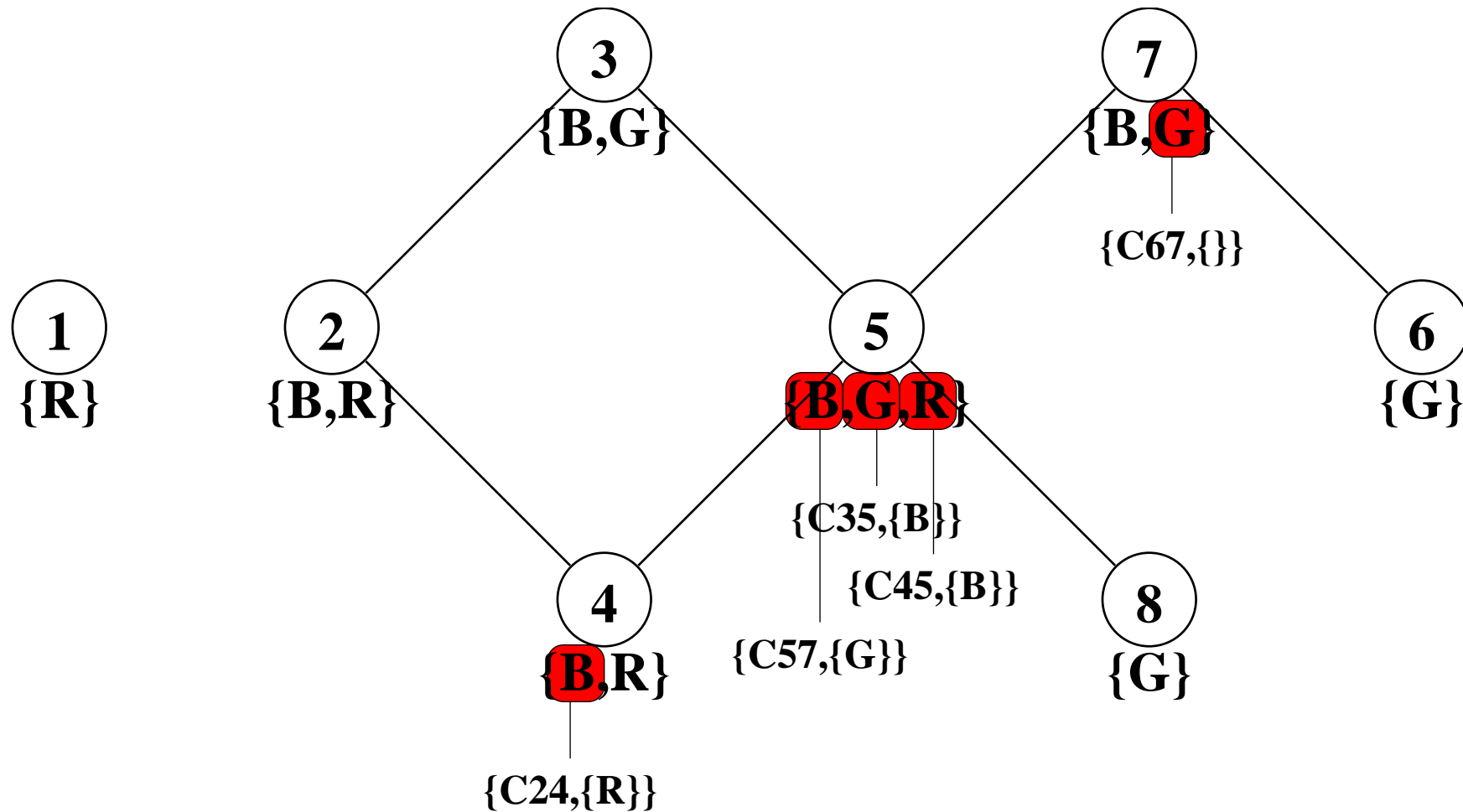
Removal of constraint 12



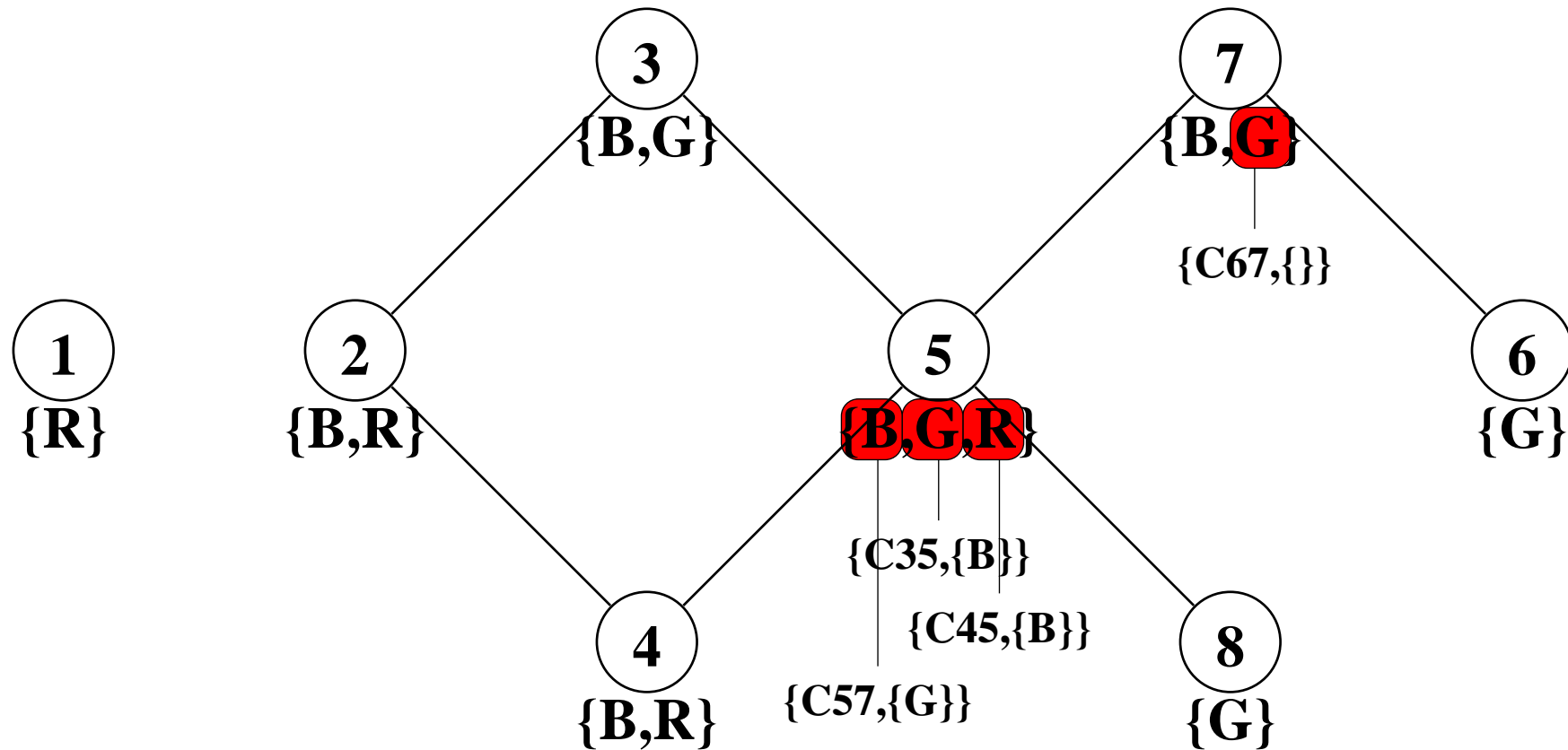
Phase 1: updating of variable 2



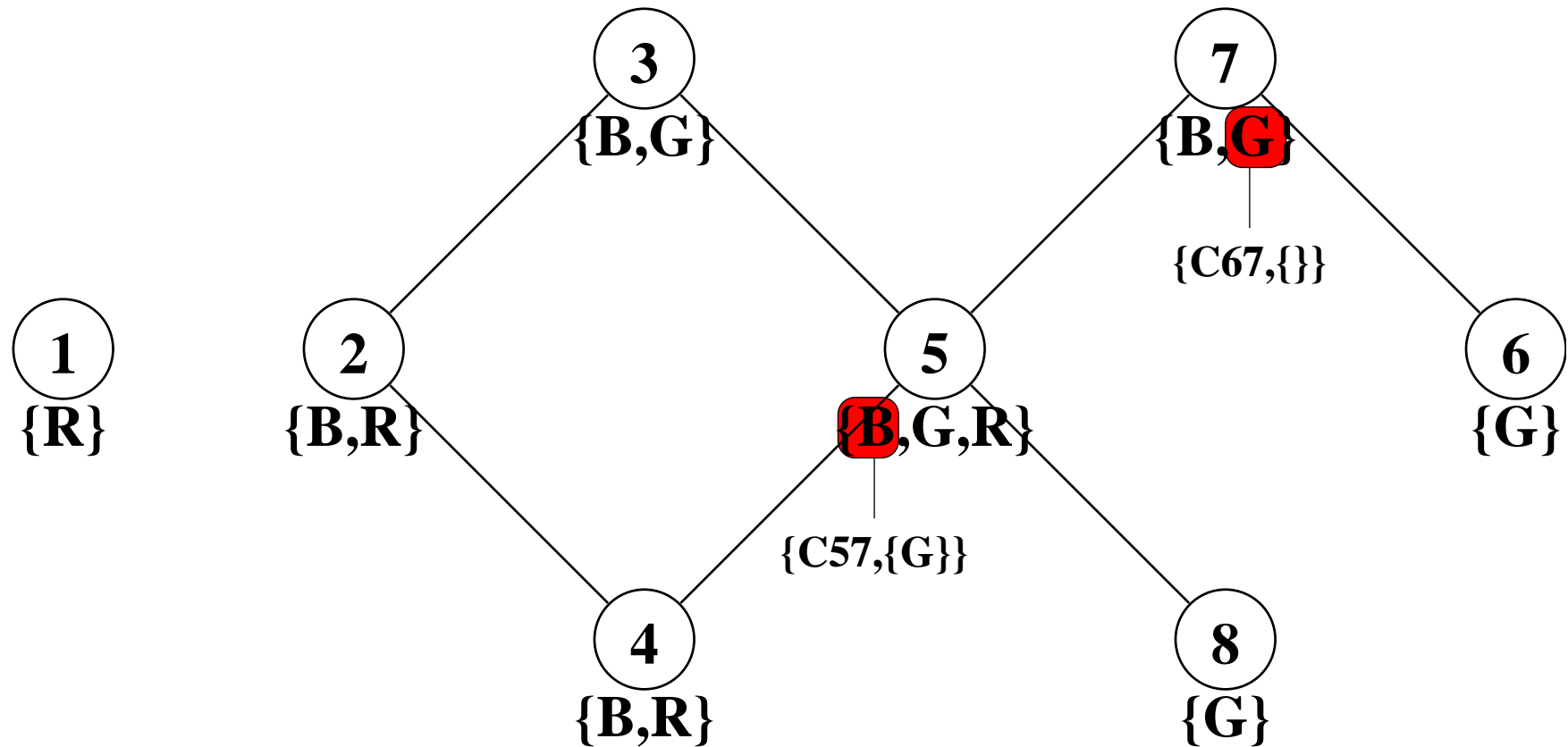
Phase 1: updating of variable 3



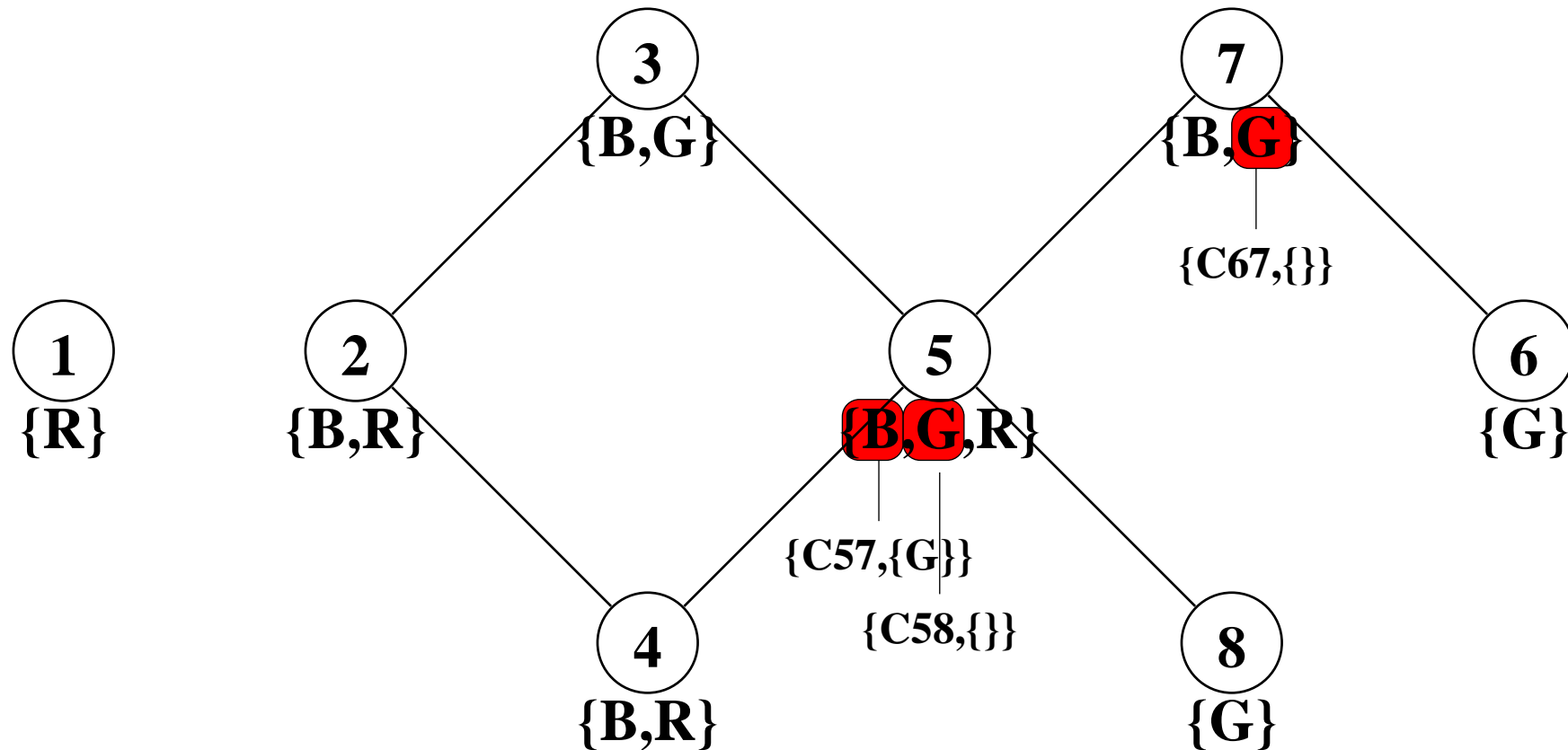
Phase 1: updating of variable 4



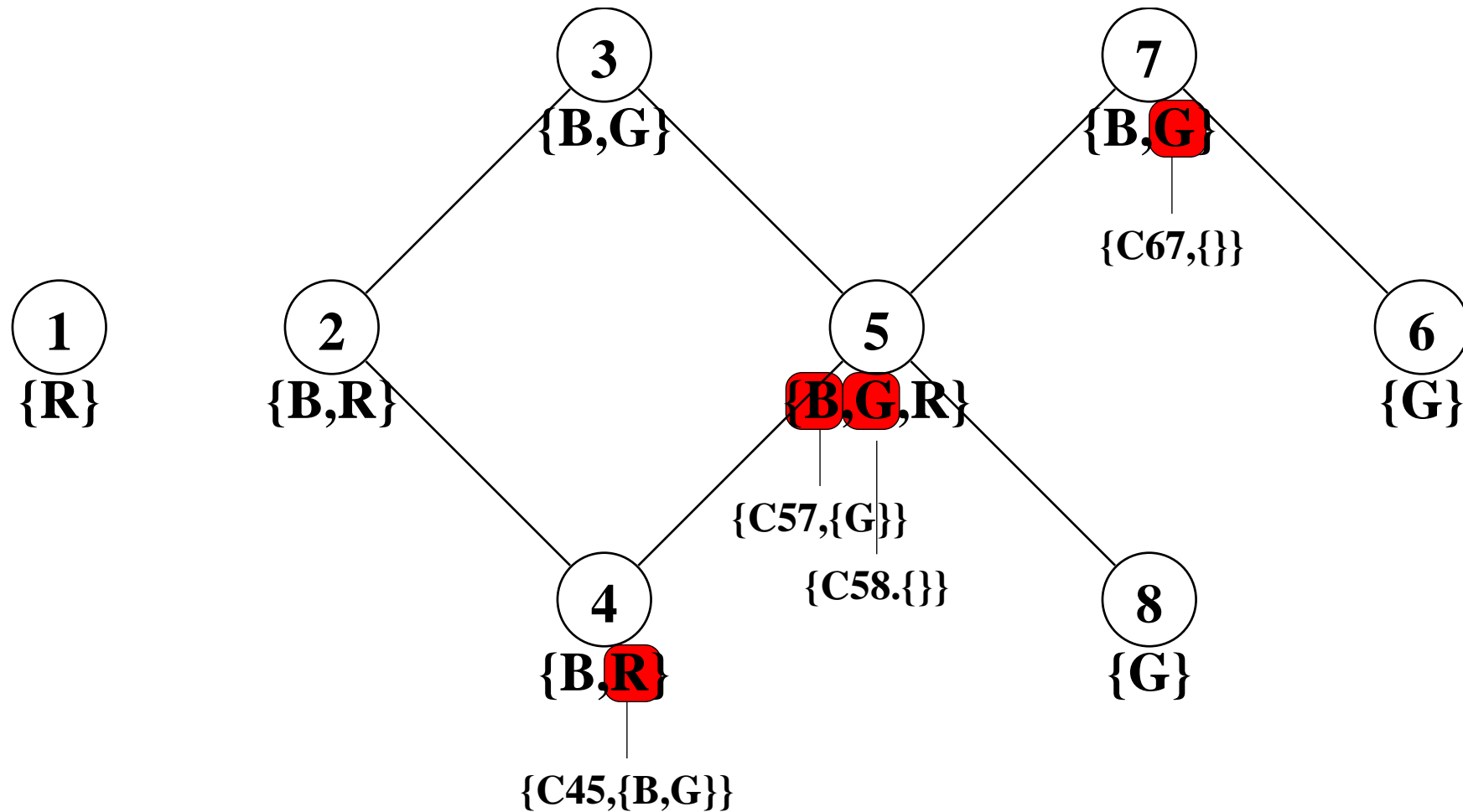
Phase 1: updating of variable 5



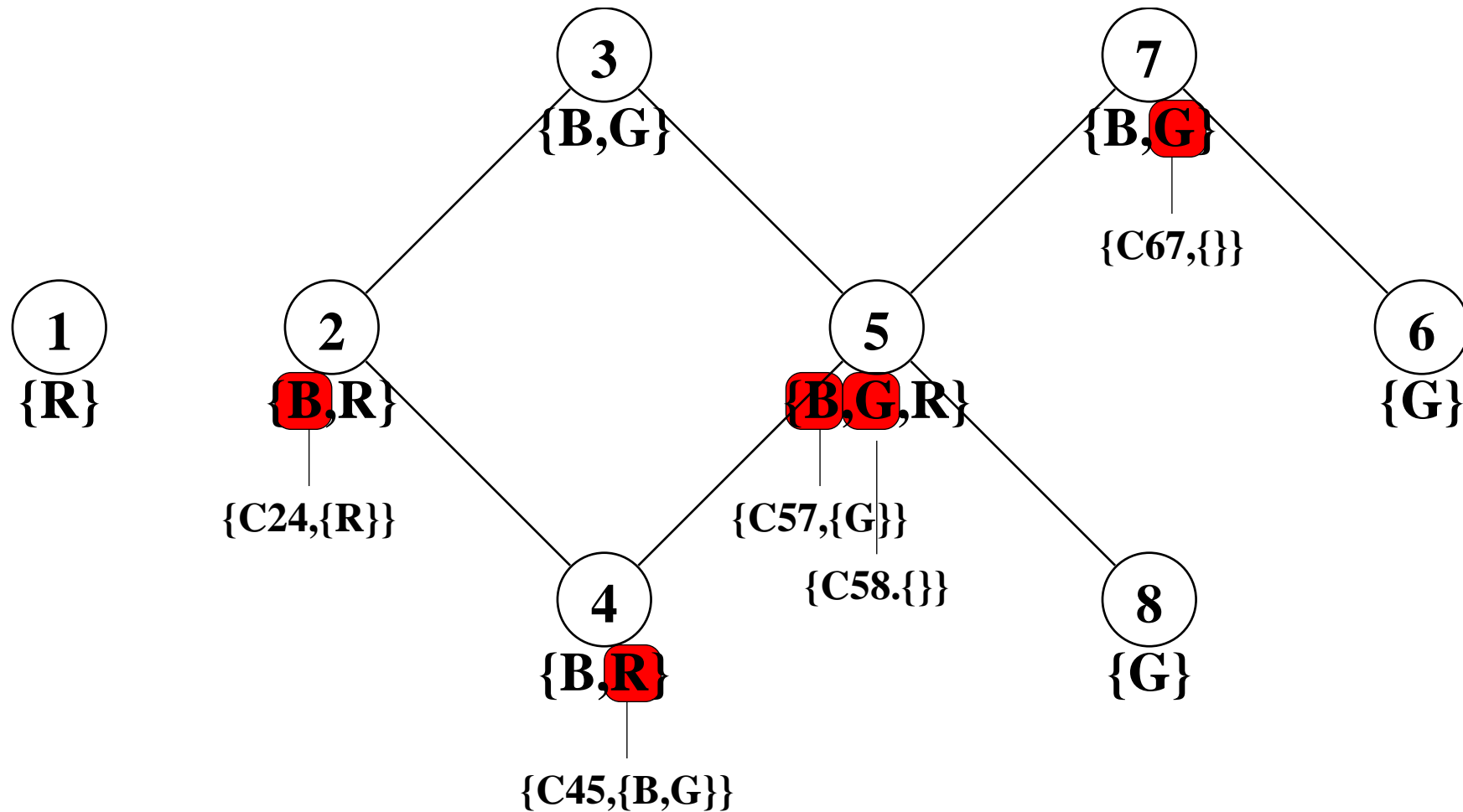
Phase 2: Arc-consistency enforcing on variable 5



Phase 2: Arc-consistency enforcing on variable 4



Phase 2: Arc-consistency enforcing on variable 2



To keep in mind

Reactive methods that reuse previous solutions or nogoods have the **same worst case complexity** as their static counterparts.

On some instances, reusing previous solutions or nogoods **may be less efficient** than solving the new problem from scratch.

Tutorial outline

1. **Setting** and **requirements**;
2. **Reactive** methods:
 - (a) **solution** reuse;
 - (b) **reasoning** reuse;
3. **Proactive** methods \Leftarrow
 - (a) **robust** solutions;
 - (b) **flexible** solutions;
4. **Research directions**.

Two classes of proactive methods

To produce:

- **robust** solutions, which have every chance to remain solutions in spite of the changes that may occur;
- **flexible** solutions, which can be easily adapted when facing a change.

Robust solutions

Given some **model** of the possible changes, to produce solutions which will resist as best as possible these changes:

1. producing **stable solutions**, by favouring values that have every chance to be involved in solutions in the real world;
2. **probabilistic CSP** modelling: **uncertain constraints**; to search for complete assignments whose likelihood to be solution in the real world is the highest;
3. **mixed CSP** modelling: **controllable** and **uncontrollable variables**; to search for complete assignments of the controllable variables which will be solutions whatever the assignment of the uncontrollable variables is.
4. **stochastic constraint programming**: **controllable** and **uncontrollable variables**; to search for complete assignments of the controllable variables whose likelihood to be solution is the highest, taking into account the possible assignments of the uncontrollable variables.

Flexible solutions

To focus on **adaptability** in case of change:

1. producing and recording **sets of solutions**, by using **cross-product** representations;
2. producing and recording **conditional solutions**, by using more sophisticated representations such as **binary decision diagrams** (BDD) or **automata**;
3. producing **super solutions** i.e., solutions which, in case of a limited change (change of value for a limited number of variables), can recover consistency by performing a limited change too;
4. producing **partial solutions**, letting another process (for example, execution in planning and scheduling applications) making the remaining decisions according to the real situation: **least commitment** strategy.

Tutorial outline

1. **Setting** and **requirements**;
2. **Reactive** methods:
 - (a) **solution** reuse;
 - (b) **reasoning** reuse;
3. **Proactive** methods:
 - (a) **robust** solutions;
 - (b) **flexible** solutions;
4. **Research directions** ⇐

Potential research directions (1)

The study of the dynamic constraint solving setting **as a whole**, with **all its requirements** (robustness, flexibility, solution stability, optimality, efficiency . . .), remains certainly to be done.

Potential research directions (2)

Need for **revisiting** constraint reasoning methods and constraint programming tools in order to make them able to deal with such dynamic settings:

→ able to handle efficiently **two basic operations**:

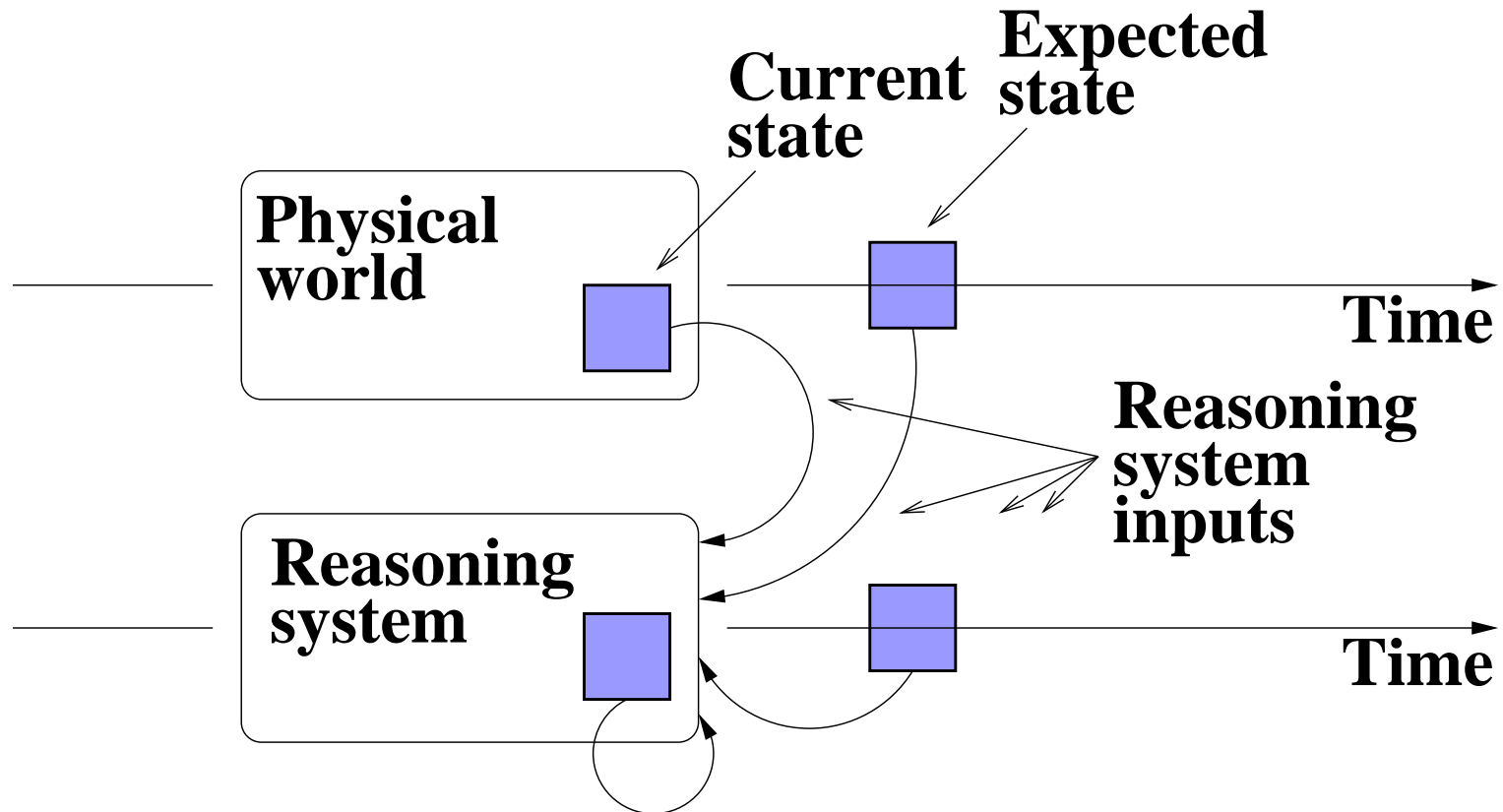
- constraint **addition**;
- constraint **removal**;

→ able to handle **three types of variables**:

- **decision** variables: controllable;
- **random** variables: uncontrollable;
- **state** variables: functions of the values taken by the variables of the previous two types.

→ able to reason at once about the **dynamics of the world** and about its **own dynamics**.

Auto-reasoning capabilities



Potential research directions (3)

To make the connection with other frameworks and approaches, such as:

- **stochastic satisfiability**;
- **bayesian networks**;
- **influence diagrams**;
- **Markov decision processes**;
- **game theory**;
- **temporal networks** with **uncertainty**;
- **anytime reasoning**;
- **time** and **resource-bounded computation**.

More information

See www.emn.fr/jussien/CP03tutorial

- this presentation;
- a commented bibliography.