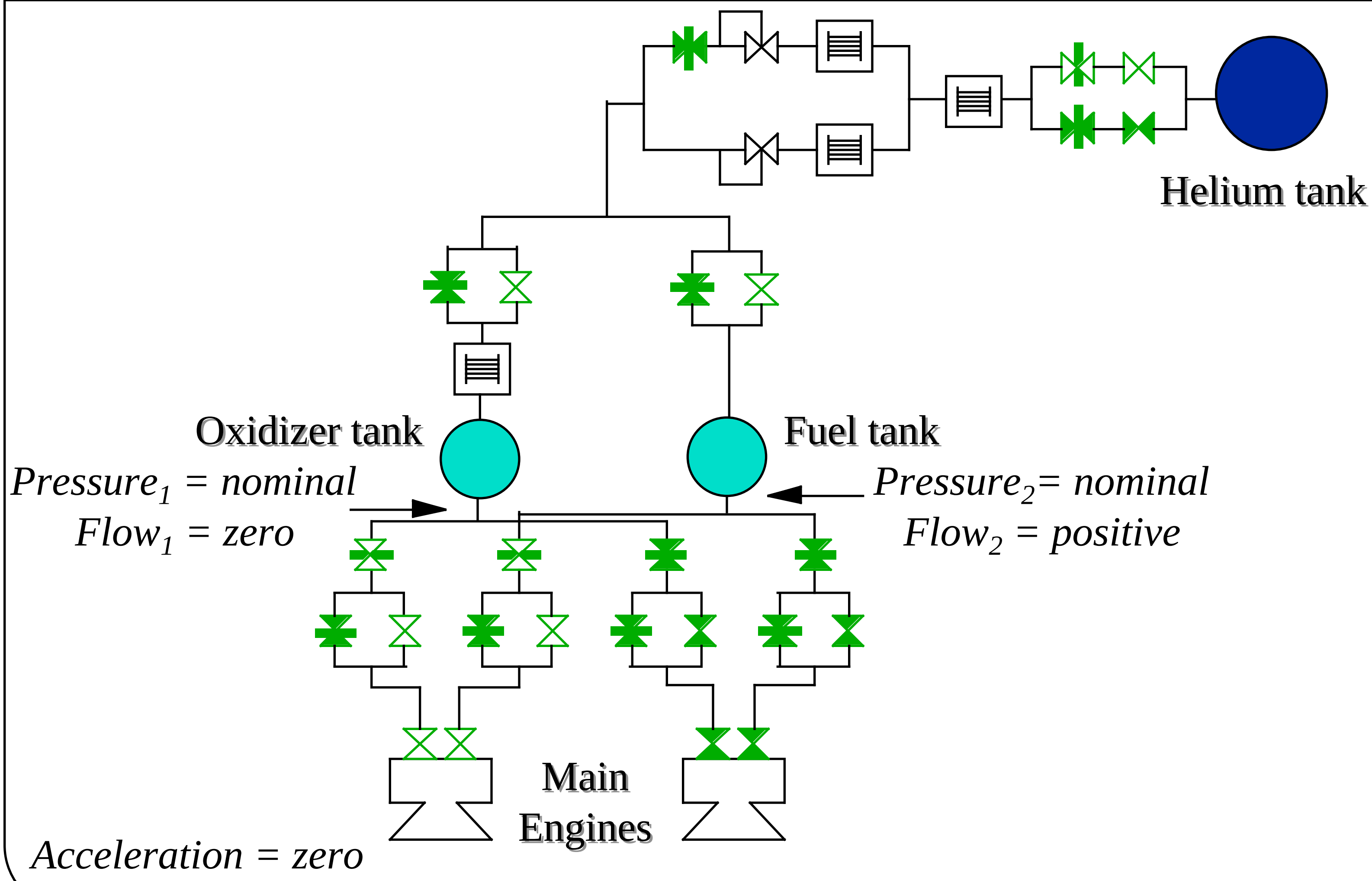
Example: Cassini propulsion system



Model-based diagnosis system

A diagnosis *system* is a triple (*SD*, *COMPS*, *OBS*)

- *SD* is a *system description*
  - expressed as a set of constraints, propositional clauses, first-order formulae …
- *COMPS* is finite set of *components*
  - a component *c* can be failed, $AB(c)$, or normal, $\neg AB(c)$
  - *SD* specifies the consequences of a component being normal or failed
- *OBS* is a set of *observations*
  - expressed as variable values, propositional clauses, first-order formulae…

Consistency-based diagnosis

- Given a set of components Δ subset of *COMPS*, a *candidate* is

$$Cand(\Delta) = \bigwedge_{c \in \Delta} AB(c) \wedge \bigwedge_{c \quad COMPS \setminus \Delta} \neg AB(c)$$

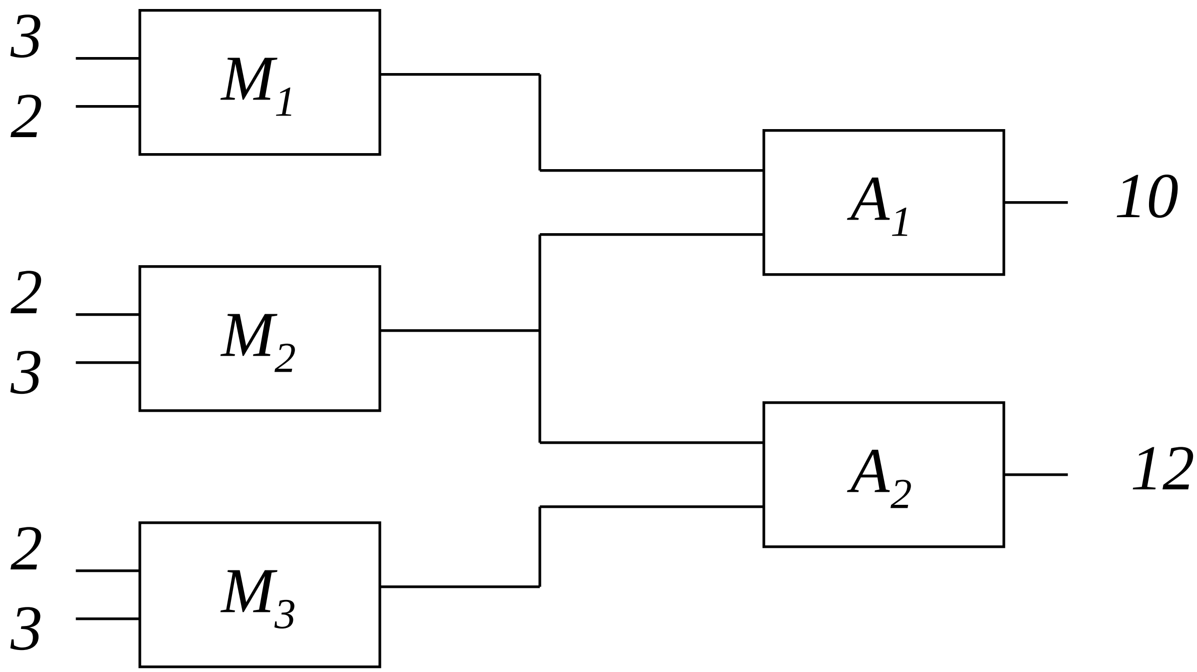
- A candidate $Cand(\Delta)$ is a *diagnosis* if and only if

$$SD \quad OBS \quad Cand(\Delta)$$

  is satisfiable

- Handles single and multiple fault cases
- Does not require fault models

Example: Adder-multiplier



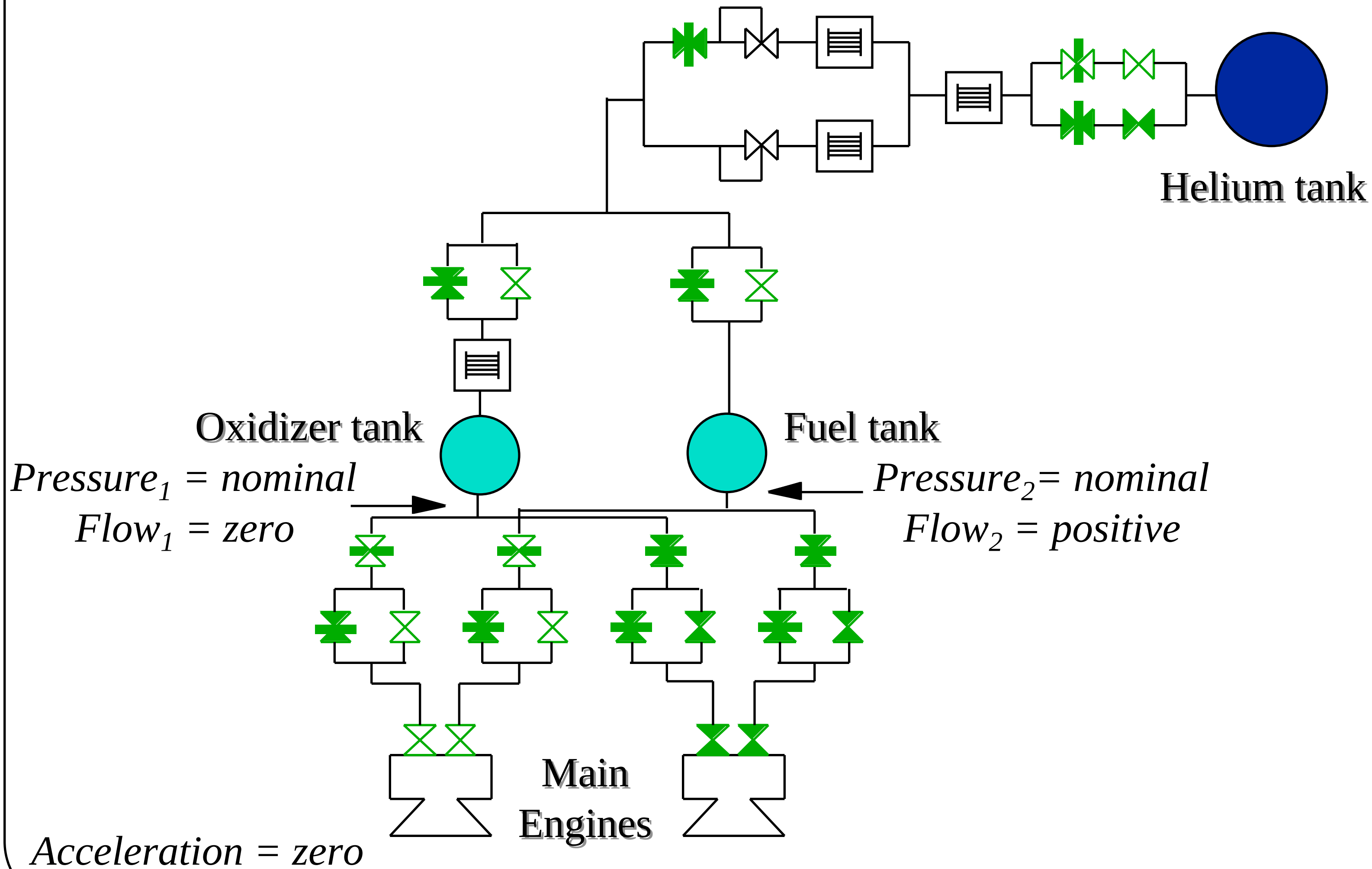
Conflicts

- A *conflict* is a disjunction of $AB(\ldots)$ literals entailed by $SD \quad OBS$
  - for conflict $conf$: $SD \quad OBS \quad \neg\, conf$ is inconsistent
- **Theorem:** Let Π be the set of conflicts of a system. A candidate $Cand(\Delta)$ is a diagnosis if and only if

$$\Pi \quad Cand(\Delta)$$

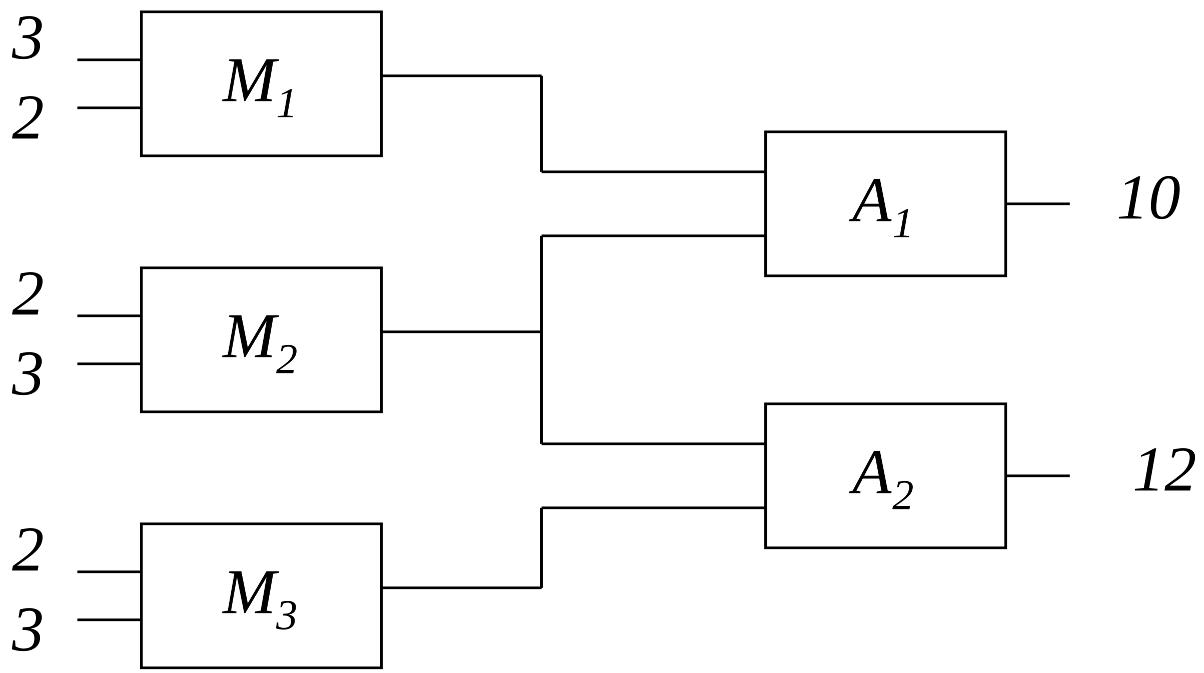
is satisfiable

Examples of conflicts

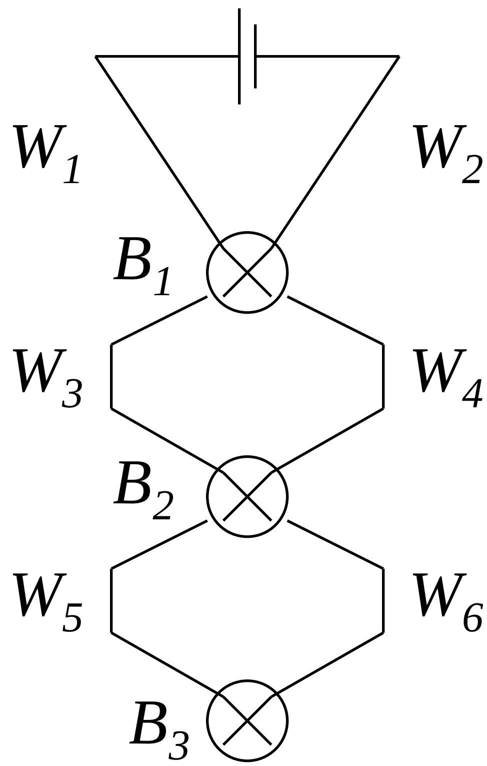


Example: Using conflicts

- Conflicts can be generated during consistency checking
- Can focus consistency checking



Fault models



Obs: B1 is off
B2 is off
B3 is on

Component modes

- Diagnosis can become indiscriminate without fault models
- Each component has a set of *modes* with associated models
  - normal modes
  - fault modes
- Each component has the *unknown* fault mode with the empty model
- Each mode has an associated *probability*

- *Diagnosis is the combinatorial optimization problem of finding the most likely component modes*

Combinatorial optimization problem

- A combinatorial optimization problem is a tuple (V, f, c)
- V is a set of *discrete* variables with *finite* domains
- An *assignment* maps each $v \in V$ to a value in v's domain
- f is a function that decides *feasibility* of assignments
  - $f(a)$ returns *true* if and only if assignment a is feasible
- c is a function that returns the *cost* of an assignment
  - $c(a)$ is the cost of assignment a
  - assignment a_1 is *preferred* over assignment a_2 if $c(a_1) < c(a_2)$
- Problem:

$$\min c(V) \text{ st } f(V)$$

Simple cost model

- Each variable has an associated cost of assigning it a value
  - $c(v_i = l_i)$ is the cost of assigning value l_i to variable v_i
- Cost of a complete assignment is the *sum* of the costs of the individual variable assignments
  - if assignment a is $v_1=l_1,\ldots,v_n=l_n$ then $c(a) = \Sigma_i\, c(v_i=l_i)$
- Costs of all variable values are non-negative
  - $c(v_i = l_i) \geq 0$
- Each variable has a minimum cost value with cost 0
  - generating a least cost assignment is straightforward
  - each variable is assigned a value with cost 0

Using the simple cost model

- Most probable diagnosis with *independent* component failures
  $p(v_1=l_1,\ldots,v_n=l_n) = p(v_1=l_1) \times \ldots \times p(v_n=l_n)$
  - let m_i be the most probable mode for component v_i
  - $c(v_i=l_i) = -\log(p(v_i=l_i)\,/\,p(v_i=m_i))$

    all costs are non-negative with $c(v_i=m_i) = 0$

    for any assignments a_1 and a_2, $c(a_1) \leq c(a_2)$ iff $p(a_1) \geq p(a_2)$

Best first search

```
function BFS(V, f, c)
    Initialize Agenda to a least cost assignment
    Initialize Solutions to the empty set
    while Agenda is non-empty do
        Let A be one of the least cost assignments in Agenda
        Remove A from Agenda
        if  f(A) is true then Add A to Solutions
        Add immediate successor assignments of A to Agenda
        if  enough solutions then return  Solutions
    endwhile
    return Solutions
end BFS
```

Required subroutines for *BFS*

- Generating a least cost assignment
- Generating the immediate successors of an assignment
  - *completeness*: every feasible assignment must be the (eventual) successor of the least cost assignment
  - *monotonicity*: if *b* is an immediate successor of *a*, then $c(a) \leq c(b)$
- Deciding that enough solutions have been generated
  - maximum number of solutions
  - minimum difference between cost of best feasible solution and the cost of the best assignment on the *Agenda*
  - minimum difference between costs of the last two assignments
- *Agenda* management as a priority queue

Representing assignments

- Each assignment is represented by the set of variable values that *differ* from the least cost assignment

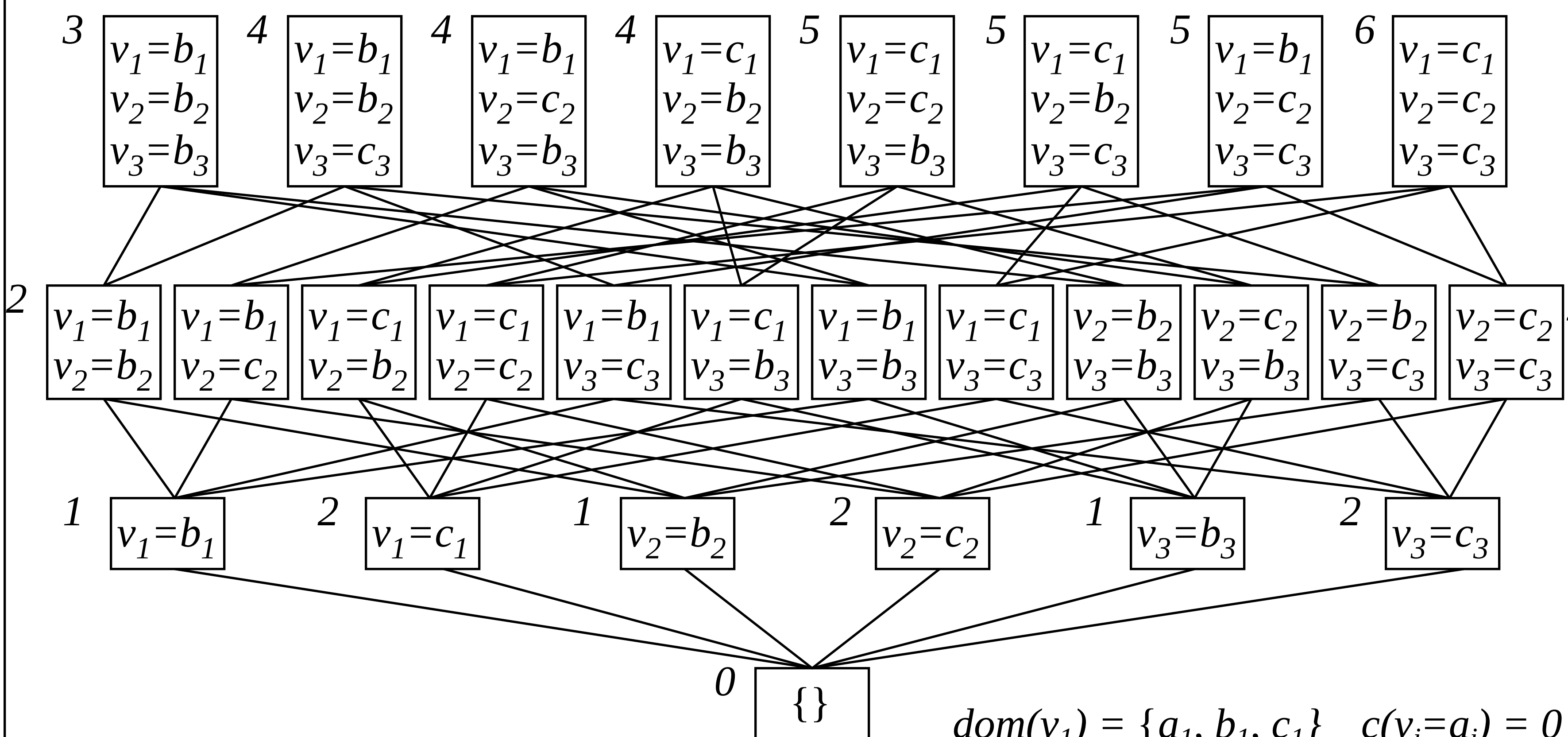
$$dom(v_1) = \{a_1, b_1, c_1\} \qquad c(v_i{=}a_i) = 0$$
$$dom(v_2) = \{a_2, b_2, c_2\} \qquad c(v_i{=}b_i) = 1$$
$$dom(v_3) = \{a_3, b_3, c_3\} \qquad c(v_i{=}c_i) = 2$$

- Least cost assignment $\{v_1{=}a_1, v_2{=}a_2, v_3{=}a_3\}$
- Assignment $\{v_1{=}a_1, v_2{=}a_2, v_3{=}b_3\}$ represented as just $\{v_3{=}b_3\}$

Basic successor function

- Assignment A_2 is an *immediate* successor of assignment A_1 if
  - the representation of A_1 is a *subset* of the representation of A_2;
  - the representations of A_1 and A_2 differ by exactly one variable value
  - *e.g.*, $\{v_3=b_3\}$ is an immediate successor of {}
  - *e.g.*, $\{v_3=b_3, v_2=b_2\}$ is an *eventual* successor, but *not* an immediate successor, of {}
- Definition of immediate successors is
  - *complete*: all assignments are eventual successors of the least cost assignment
  - *monotonic*: if A_2 is an immediate successor of A_1, then $c(A_1) \leq c(A_2)$

Successor lattice



$dom(v_1) = \{a_1, b_1, c_1\}$ $c(v_i{=}a_i) = 0$

$dom(v_2) = \{a_2, b_2, c_2\}$ $c(v_i{=}b_i) = 1$

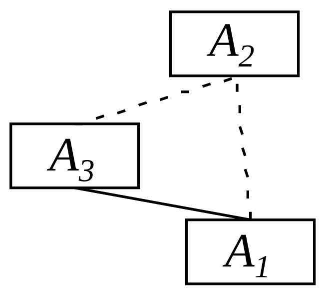
$dom(v_3) = \{a_3, b_3, c_3\}$ $c(v_i{=}c_i) = 2$

Using conflicts

- *Requirement*: whenever f determines that an assignment is infeasible, it returns a conflict
  - if assignment A is infeasible, then A itself is trivially a conflict
  - ideally, f should return a *minimal* infeasible subset of A as a conflict
  - conflicts can be generated using dependency tracking in a truth maintenance system

Focusing with conflicts

- *Lemma*: Let A_2 be an (eventual) successor of A_1 such that A_1 is subsumed by a conflict N, but A_2 is not. Then there exists an immediate successor A_3 of A_1 that is not subsumed by N such that A_2 is an (eventual) successor of A_3.

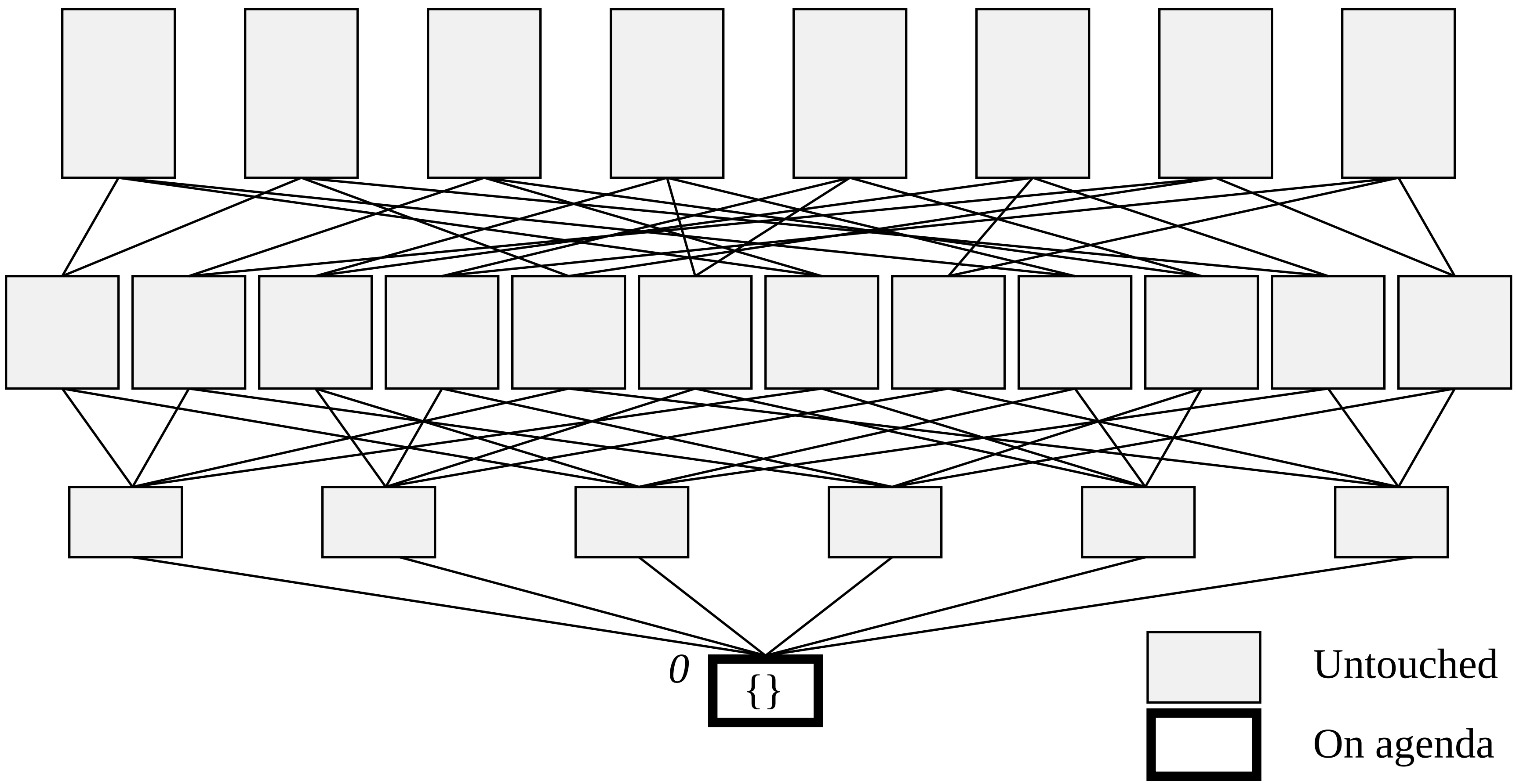


If an assignment A_1 is infeasible and is subsumed by a conflict N, then we need only generate those immediate successors of A_1 that are *not* subsumed by N

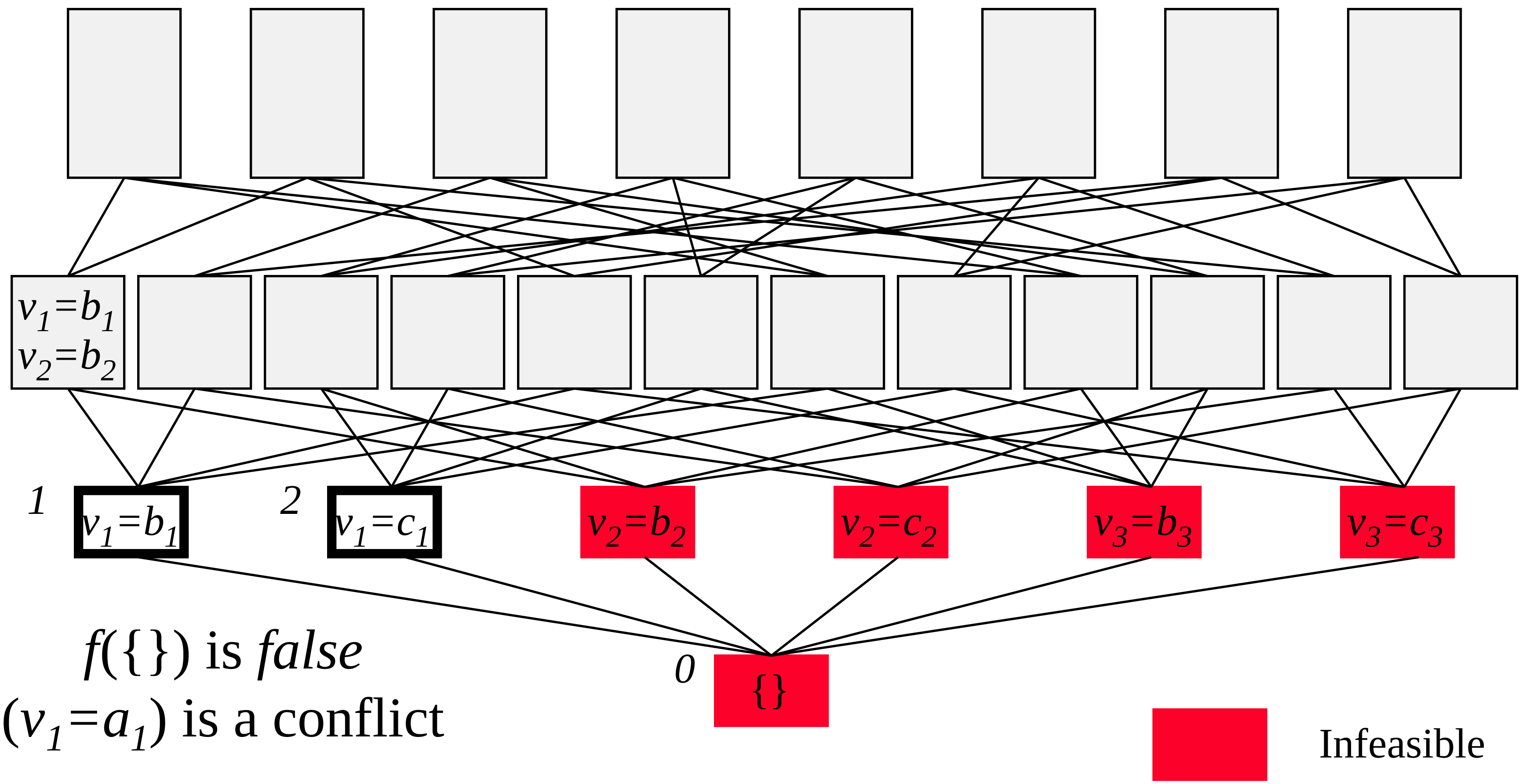
  – the lemma ensures that completeness is preserved

  – the smaller the conflict, the fewer the immediate successors

Initializing the agenda



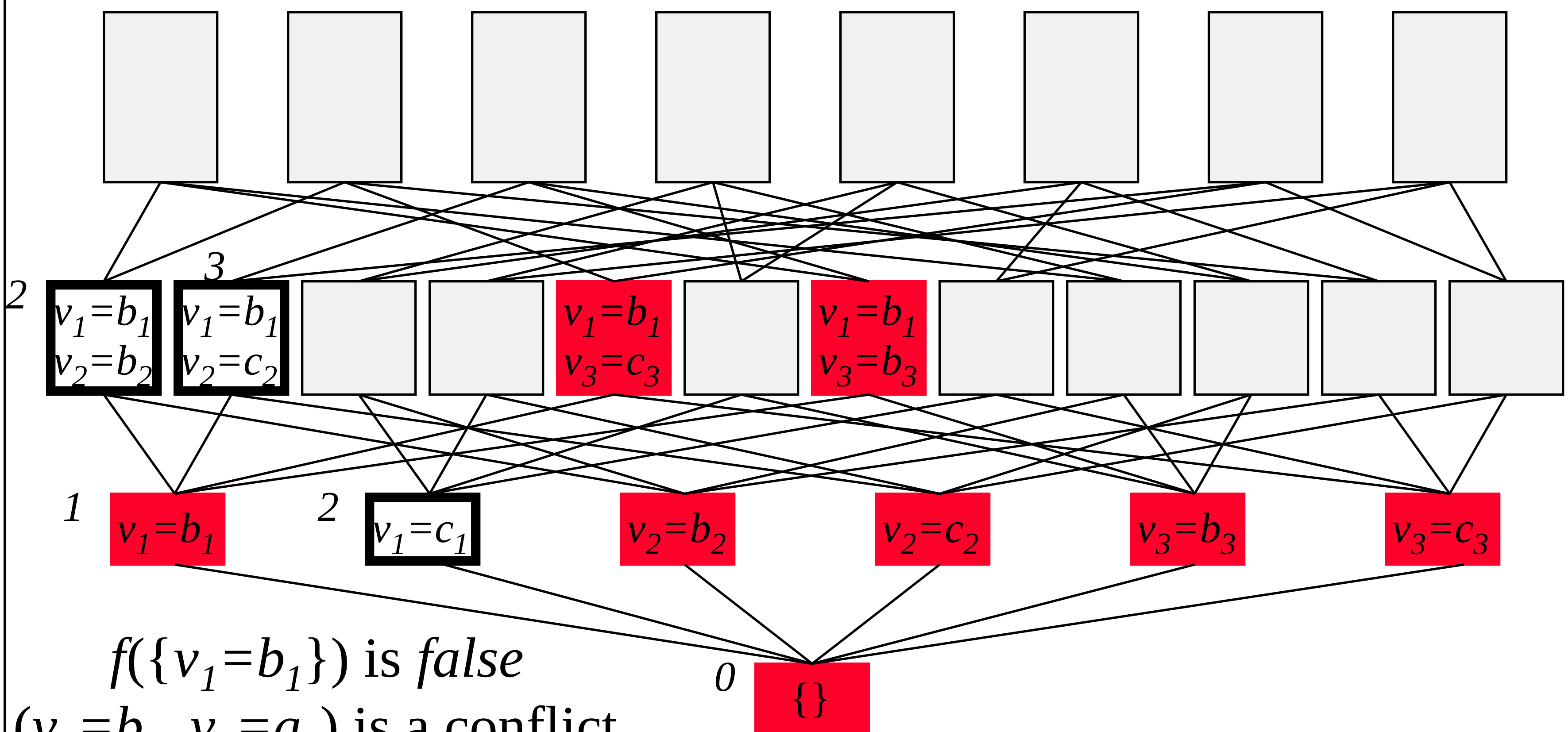
Assignment {} is infeasible



$f(\{\})$ is *false*

$(v_1=a_1)$ is a conflict

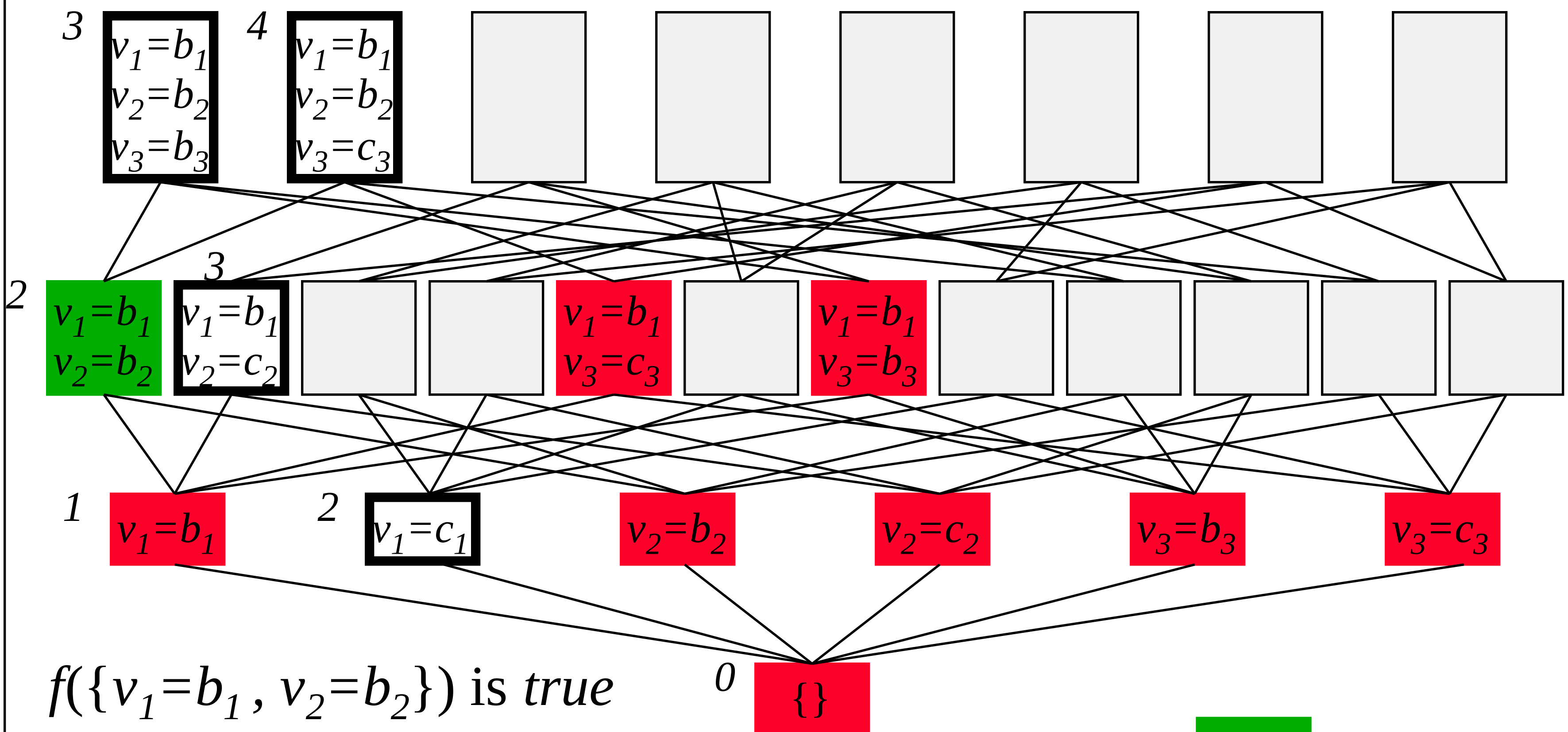
Assignment $\{v_1=b_1\}$ is infeasible



$f(\{v_1=b_1\})$ is *false*

$(v_1=b_1,\ v_2=a_2)$ is a conflict

Least cost feasible assignment found



$f(\{v_1=b_1, v_2=b_2\})$ is *true*

Decreasing agenda size

- Agenda size can be problematic in a best first search
  - for a branching factor *b*, agenda grows to size *O(bk)* after *k* checks
  - inserting *b* elements into the agenda after *k* checks is *O(b logb+b logk)*
- Immediate successors of an assignment are totally ordered
  - non-least cost successors only checked *after* least cost successor

  Insert only least cost successor onto agenda
  Sort remaining successors
  Each assignment has exactly two successors
  - least cost immediate successor
  - next more expensive sibling
- Size of the agenda is *bounded by* the number of checks
  - inserting *b* successors after *k* checks is *O(b logb + 2logk)*