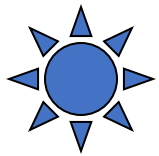


CSP

Zahoor Tanoli (PhD)

COMSATS Attock

Outline

- Constraint Satisfaction Problems (CSP)
- Backtracking search for CSPs
- Local search for CSPs

Constraint satisfaction problems (CSPs)

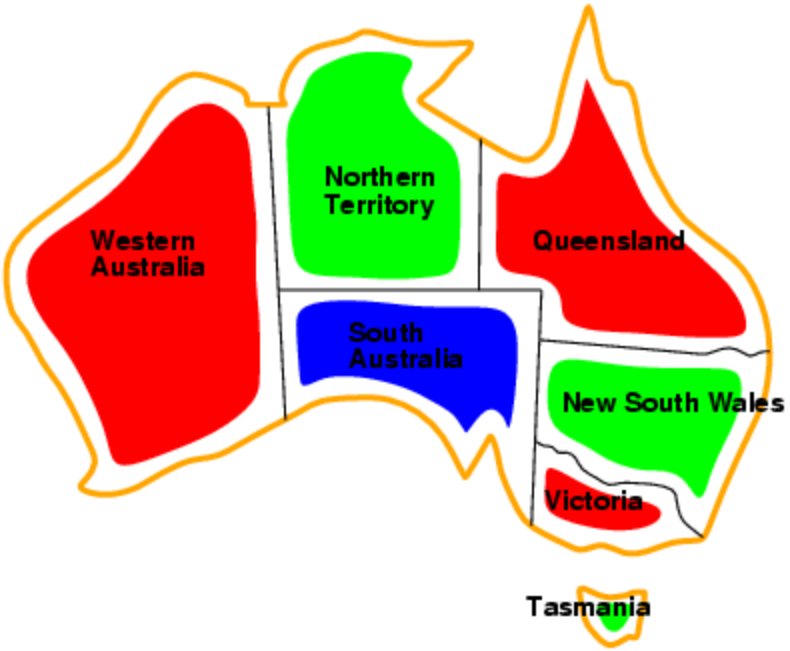
- Standard search problem:
  - state is a "black box“ – any data structure that supports successor function, heuristic function, and goal test
- CSP:
  - state is defined by variables X_i with values from domain D_i
  - goal test is a set of constraints specifying allowable combinations of values for subsets of variables
- Simple example of a formal representation language
- Allows useful general-purpose algorithms with more power than standard search algorithms

Example: Map-Coloring

- Variables *WA, NT, Q, NSW, V, SA, T*
- Domains D_i = {red,green,blue}
- Constraints: adjacent regions must have different colors

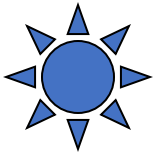
- e.g., WA ≠ NT, or (WA,NT) in {(red,green),(red,blue),(green,red), (green,blue),(blue,red),(blue,green)}

Example: Map-Coloring

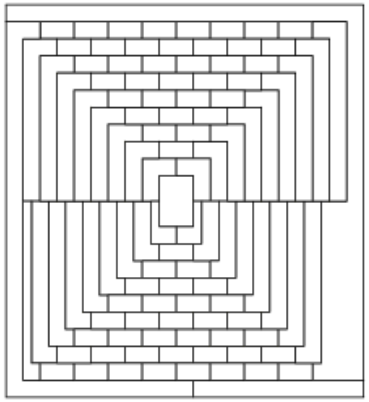


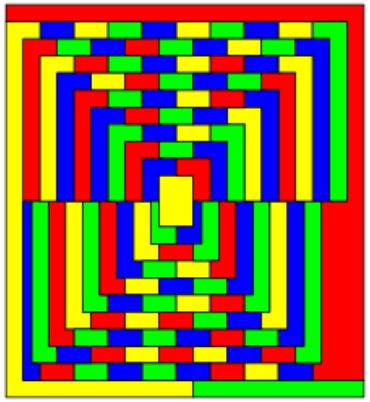
- Solutions are complete and consistent assignments, e.g., WA = red, NT = green,Q = red,NSW = green,V = red,SA = blue,T = green

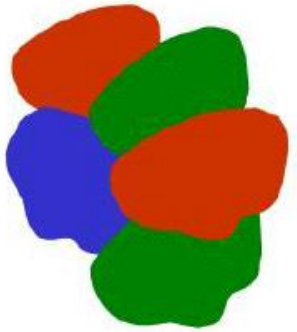
Graph Coloring as CSP



Pick colors for map regions,
avoiding coloring adjacent regions
with the same color





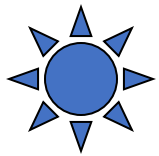


Variables regions

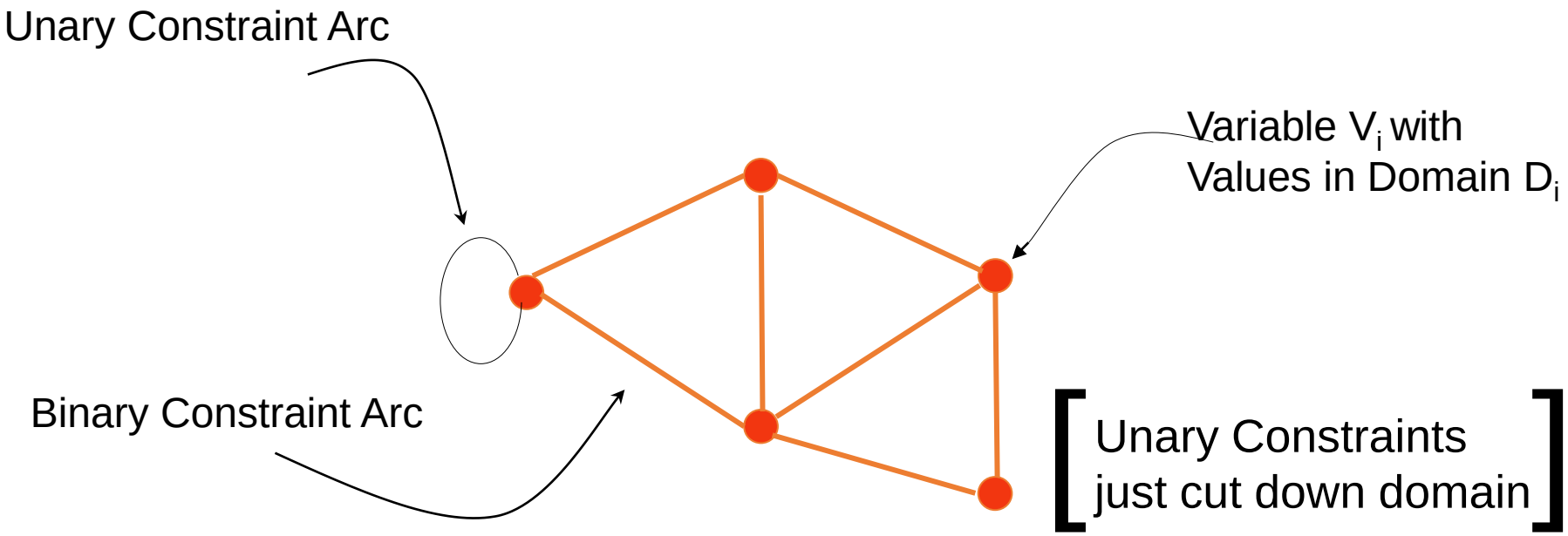
Domains colors allowed

Constraints adjacent regions must have different color

Constraint Satisfaction Problems

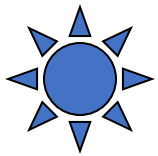


General Class of Problems: Binary CSP

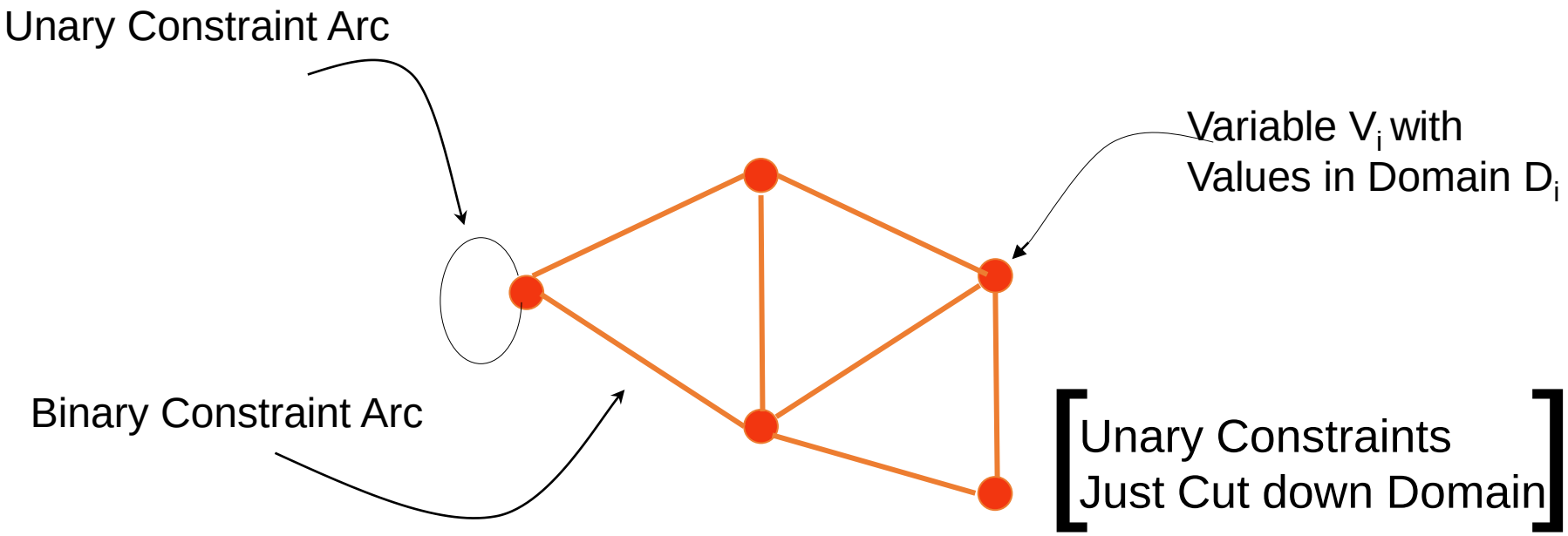


This Diagram is called a Constraint Graph

Constraint Satisfaction Problems



General Class of Problems: **Binary CSP**



This Diagram is called a Constraint Graph

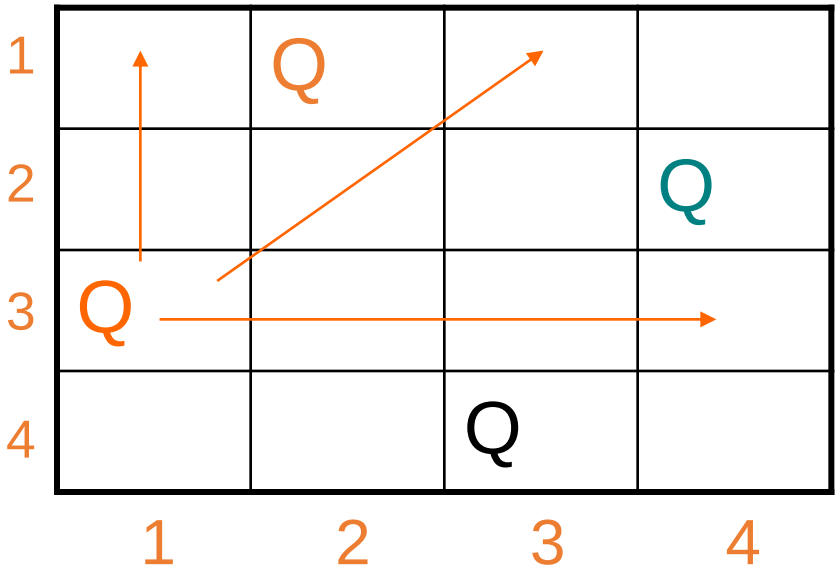
Basic Problem:

Find a $d_j \in D_i$ for each V_i, so that all constraints satisfied.
(Finding consistent labeling for variables)

N-Queens as CSP

(Classic Benchmark Problem)

Place N Queens on an NxN chessboard So that none can attack the other

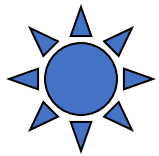


Variables — are board positions in NxN chessboard

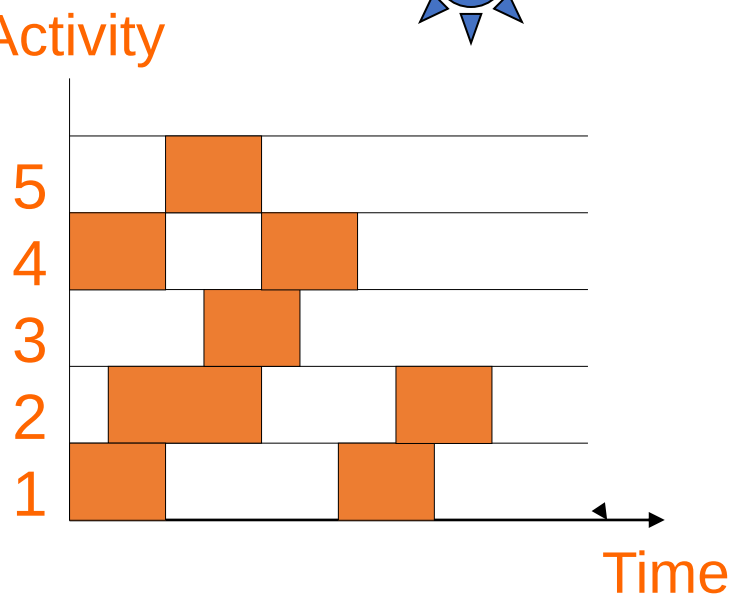
Domains — Queen or Blank

Constraints — Two positions on a line (Vertical, Horizontal, Diagonal) cannot both be Q

Scheduling as CSP



Choose time for activities e.g.
Terms to take required classes



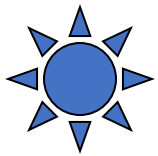
Variables are activities

Domains Sets of start times (or “chunks” of time)

Constraints

1. Activities that use same resource cannot overlap in time
2. Preconditions satisfied

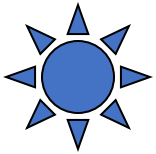
CSP Example



Given 40 courses (8.01, 8.02,…….8.40) and 10 Terms

(Fall 1, Spring 1, …….Spring 5). Find a legal schedule.

CSP Example



Given 40 courses (8.01, 8.02,…….8.40) and 10 Terms

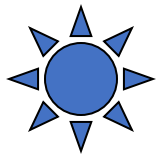
(Fall 1, Spring 1, …….Spring 5). Find a legal schedule.

Constraints:

Pre-requisites
Courses offered on limited terms
Limited number of courses per term
Avoid time conflicts

Note, CSPs are not for expressing (soft) preferences e.g. Minimize difficulty, balance subject areas etc.

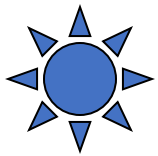
Choice of variables & values



Variables	Domains
A. Terms	Legal combinations of e.g. 4 courses (but this is huge set of values).

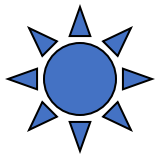
What are the Variables and what are the Values?

Choice of variables & values



Variables	Domains
A. Terms	Legal combinations of e.g. 4 courses (but this is huge set of values).
B. Term Slots	
Subdivide terms into Slots e.g. 4 of them (Fall 1,1)(Fall 1,2) (Fall 1,3)(Fall 1,4)	Courses offered during that term

Choice of variables & values

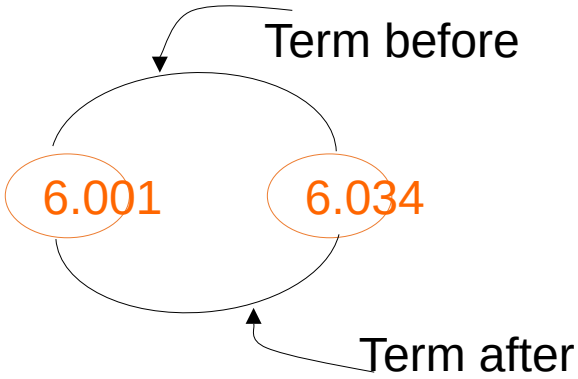


Variables	Domains
A. Terms	Legal combinations of e.g. 4 courses (but this is huge set of values).
B. Term Slots	
Subdivide terms into Slots e.g. 4 of them (Fall 1,1)(Fall 1,2) (Fall 1,3)(Fall 1,4)	Courses offered during that term
C. Courses?	Terms or term slots (term slots allow expressing constraint on limited number of courses/slots)

Constraints

Use courses as variables and term slots as values.

Pre-requisite →

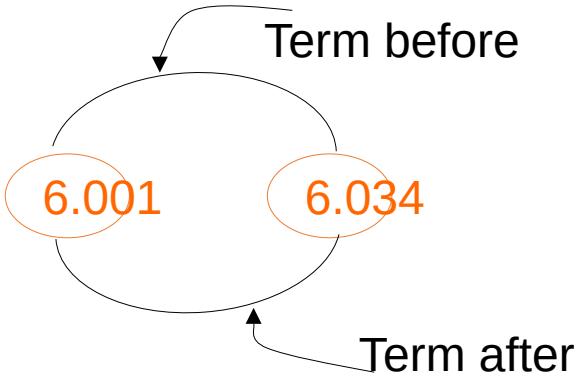


For pairs of courses that must be ordered

Constraints

Use courses as variables and term slots as values.

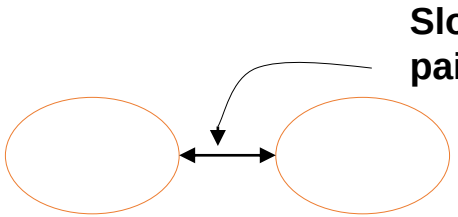
Pre-requisite →



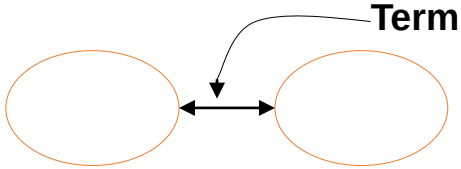
For pairs of courses that must be ordered

Courses offered only in some terms → **Filter Domain**

Limited number of Courses →

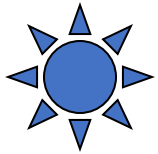


Avoid time conflicts →



For pairs offered at same or overlapping times/periods

Solving CSPs



Solving CSPs involves some combination of:

1- Constraint propagation, to eliminate values that could not be part of any solution

2- Search, to explore valid assignments

Constraint Propagation (aka Arc Consistency)

Arc consistency eliminates values from domain of variable that can never be part of a consistent solution.

$$V_i \longrightarrow V_j$$

Directed arc (V_i, V_j) is arc consistent if
$\forall$ **x$\in$D$_i$** $\exists$ **y$\in$D$_j$ such that (x,y)** is allowed by the constraint on the arc.

For every value in the domain of Vi, there exist some value in the domain of Vj that will satisfy the constraint on the arc

Constraint Propagation (aka Arc Consistency)

Arc consistency eliminates values from domain of variable that can never be part of a consistent solution.

$$V_i \longrightarrow V_j$$

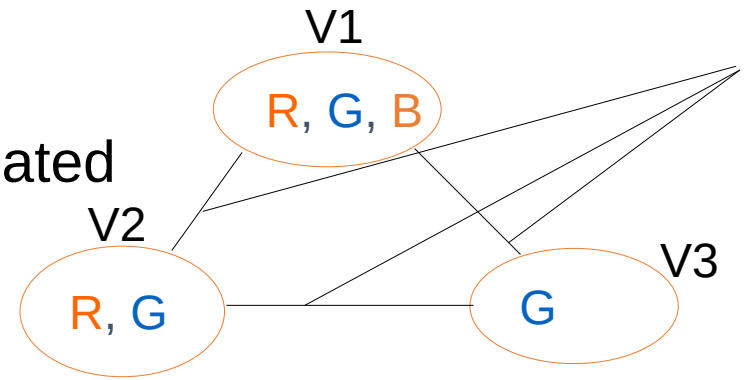
Directed arc (V_i, V_j) is arc consistent if
$\forall$ **$x \in D_i$** $\exists$ **$y \in D_j$ such that (x,y)** is allowed by the constraint on the arc.

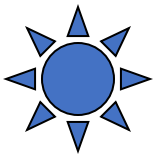
We can achieve consistency on arc by deleting values from D_i (domain of variable at tail of consistent arc) that fail this condition.

Constraint Propagation Example

Graph Coloring

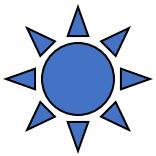
Initial domains are indicated





Different-color constraint

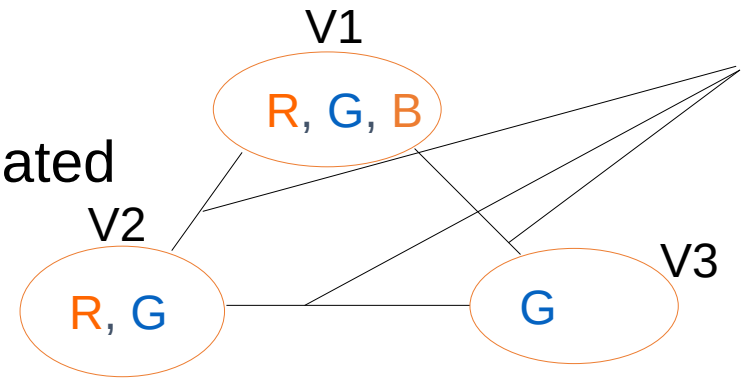
Constraint Propagation Example



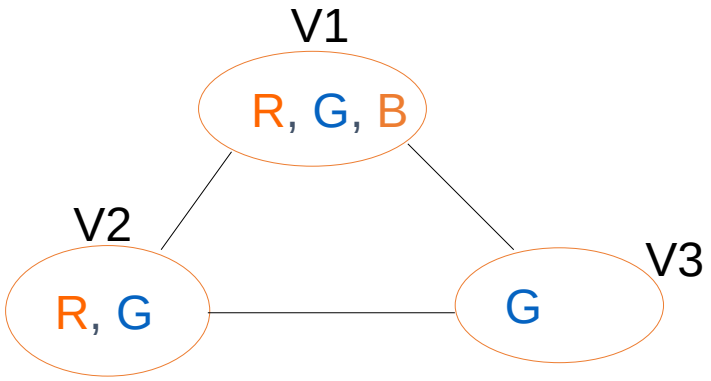
Different-color constraint

Graph Coloring

Initial domains are indicated

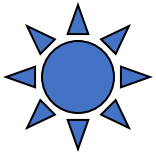


Arc Examined	Value Deleted



Each undirected constraint arc is really two directed arc constraint arcs, the effects shown above are from examining both arcs.

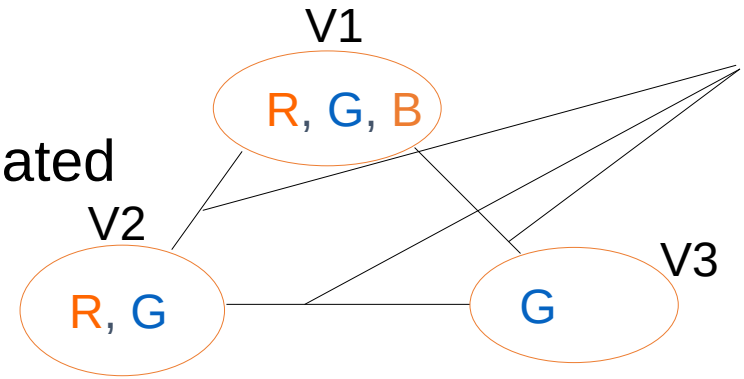
Constraint Propagation Example



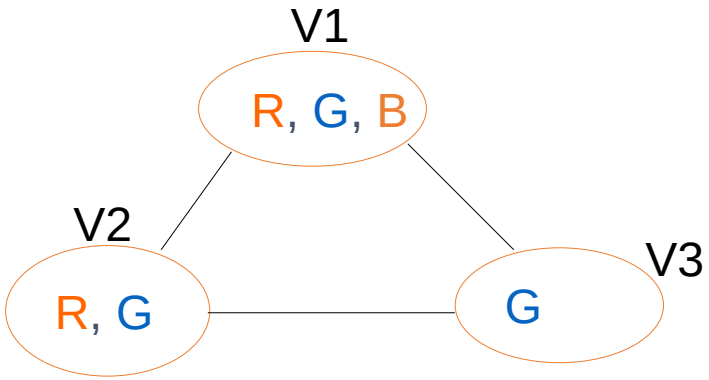
Different-color constraint

Graph Coloring

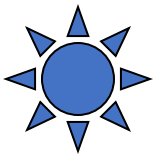
Initial domains are indicated



Arc Examined	Value Deleted
V1-V2	none



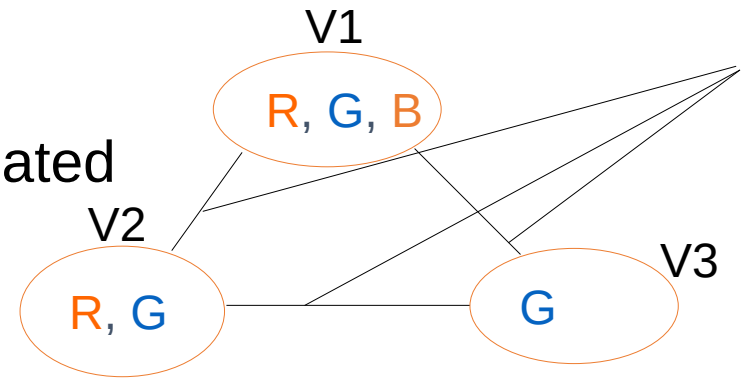
Constraint Propagation Example



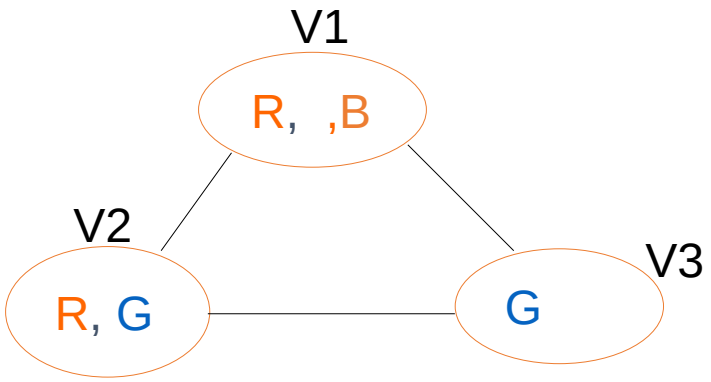
Different-color constraint

Graph Coloring

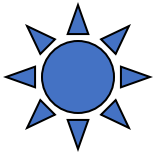
Initial domains are indicated



Arc Examined	Value Deleted
V1-V2	none
V1-V3	V1 (G)



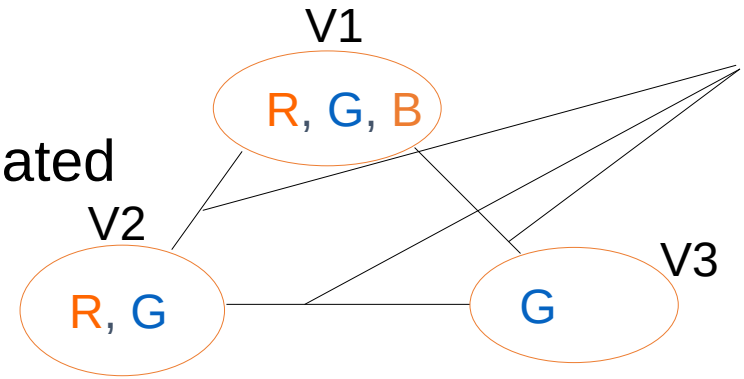
Constraint Propagation Example



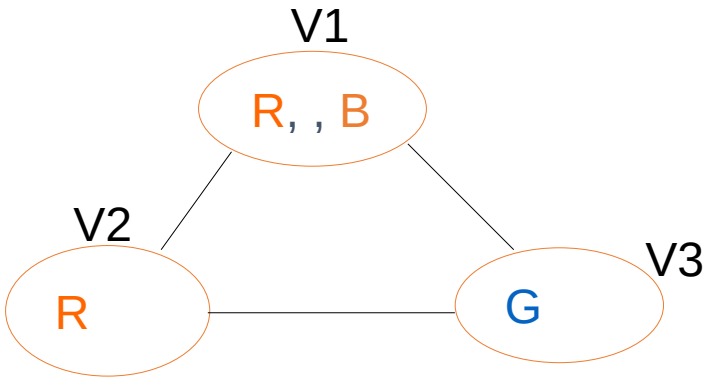
Different-color constraint

Graph Coloring

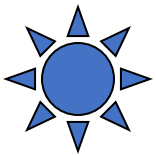
Initial domains are indicated



Arc Examined	Value Deleted
V1-V2	none
V1-V3	V1 (G)
V2-V3	V2 (G)



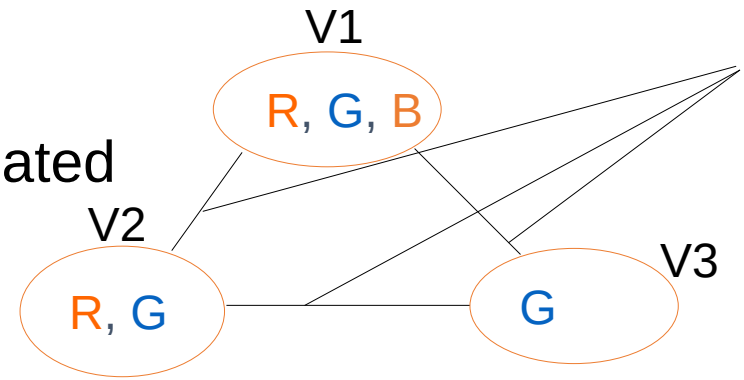
Constraint Propagation Example



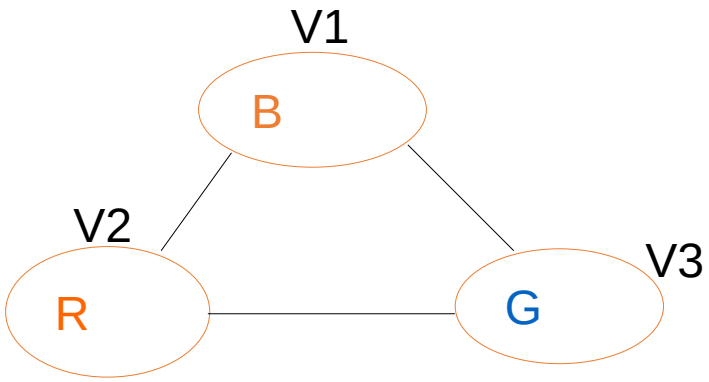
Different-color constraint

Graph Coloring

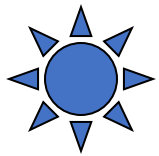
Initial domains are indicated



Arc Examined	Value Deleted
V1-V2	none
V1-V3	V1 (G)
V2-V3	V2 (G)
V1-V2	V1 (R)



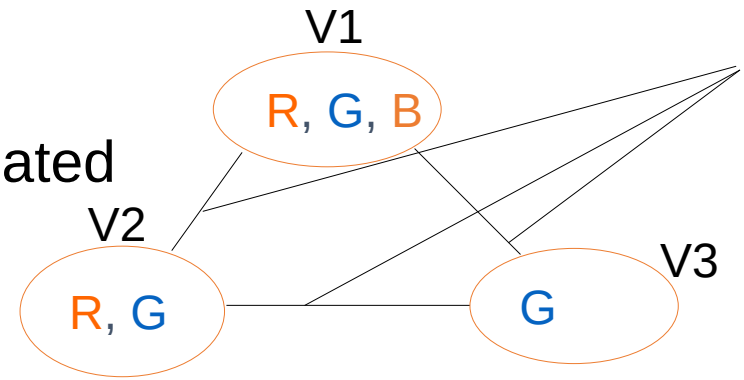
Constraint Propagation Example



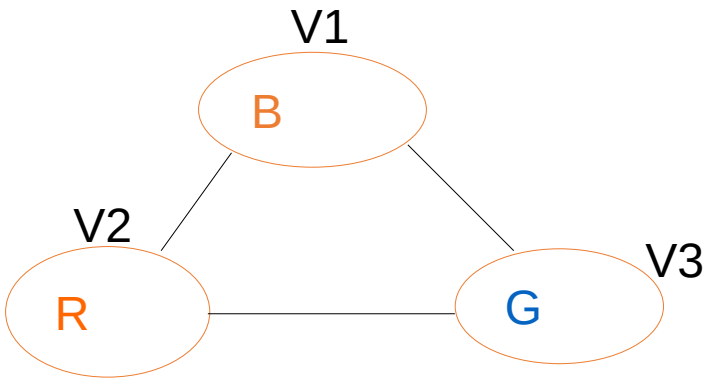
Different-color constraint

Graph Coloring

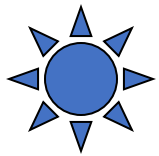
Initial domains are indicated



Arc Examined	Value Deleted
V1-V2	none
V1-V3	V1 (G)
V2-V3	V2 (G)
V1-V2	V1 (R)
V1-V3	none



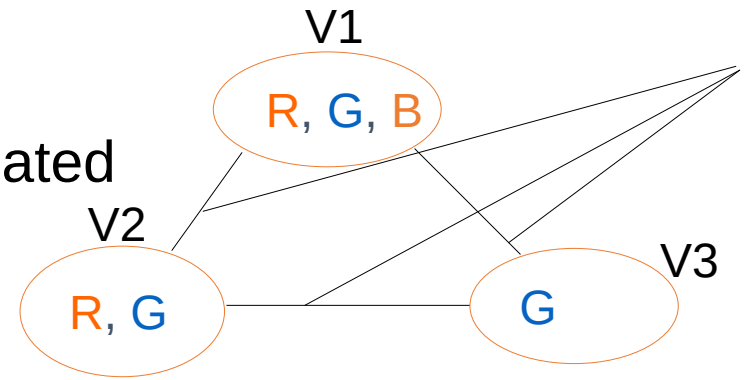
Constraint Propagation Example



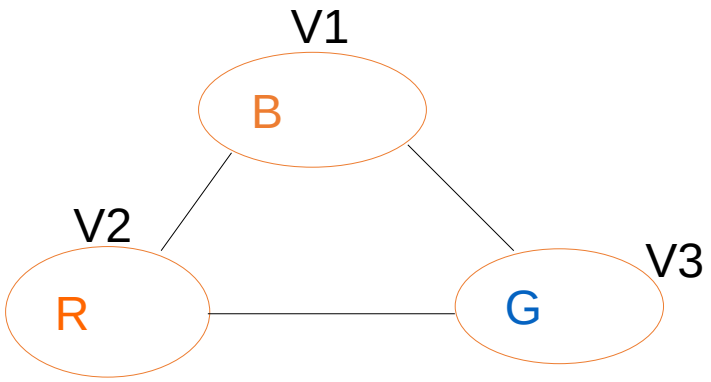
Different-color constraint

Graph Coloring

Initial domains are indicated

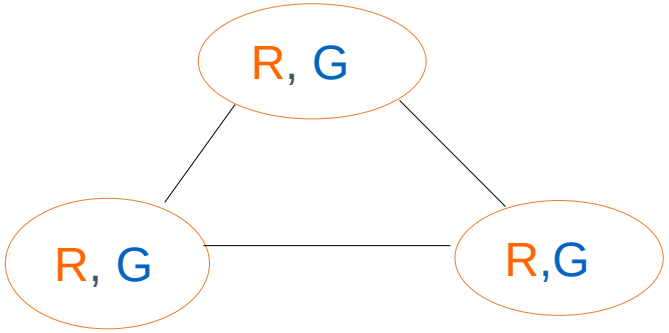


Arc Examined	Value Deleted
V1-V2	none
V1-V3	V1 (G)
V2-V3	V2 (G)
V1-V2	V1 (R)
V1-V3	none
V2-V3	none

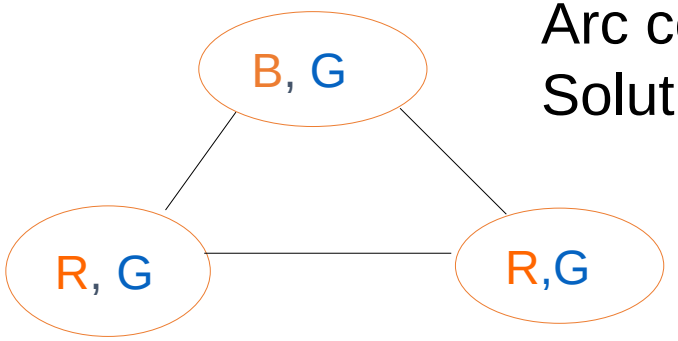


But arc consistency is not enough in general

Graph Coloring



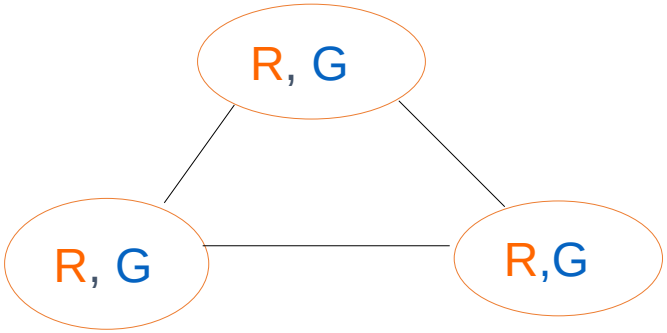
Arc consistent but no solution



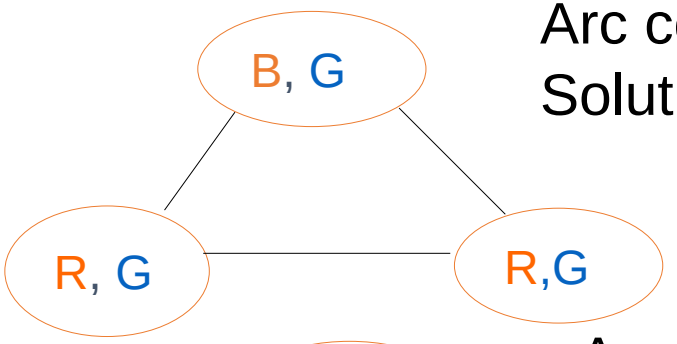
Arc consistent but 2 Solutions B,R,G; B,G,R

But arc consistency is not enough in general

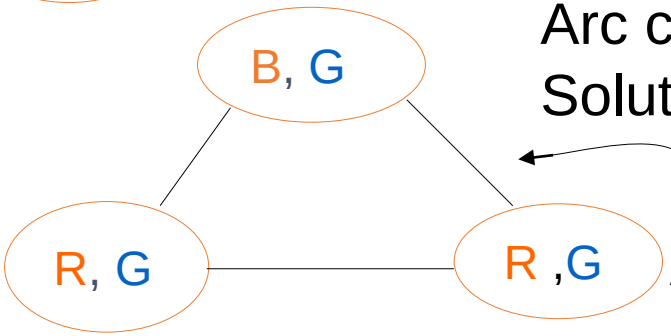
Graph Coloring



Arc consistent but no solution



Arc consistent but 2
Solutions B,R,G; B,G,R

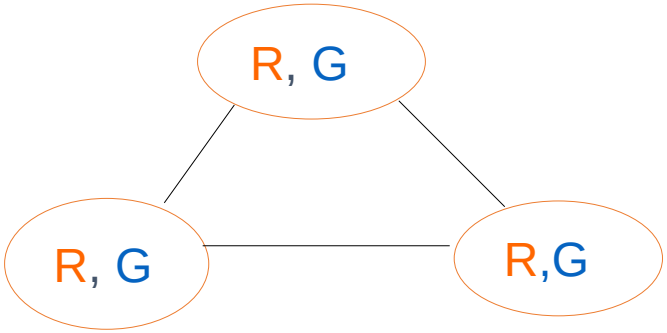


Arc consistent but 1
Solution B,R,G

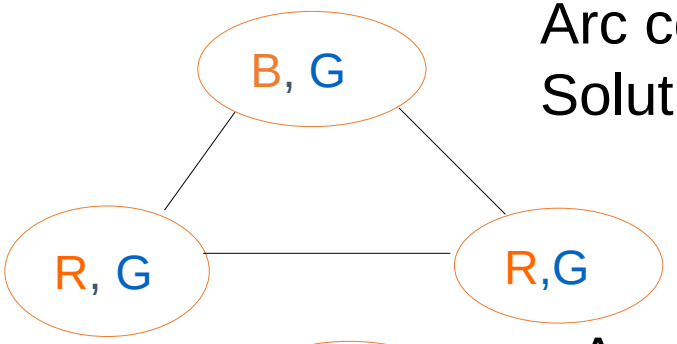
Assume B, R not allowed

But arc consistency is not enough in general

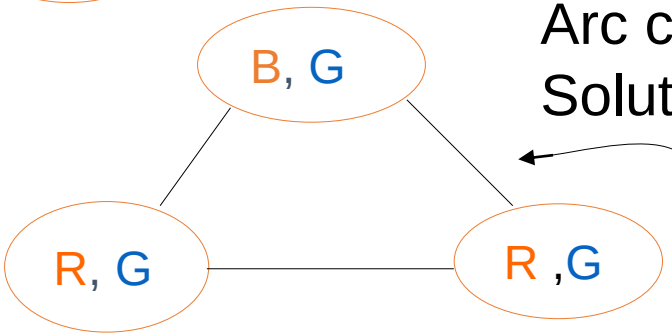
Graph Coloring



Arc consistent but no solution



Arc consistent but 2
Solutions B,R,G; B,G,R



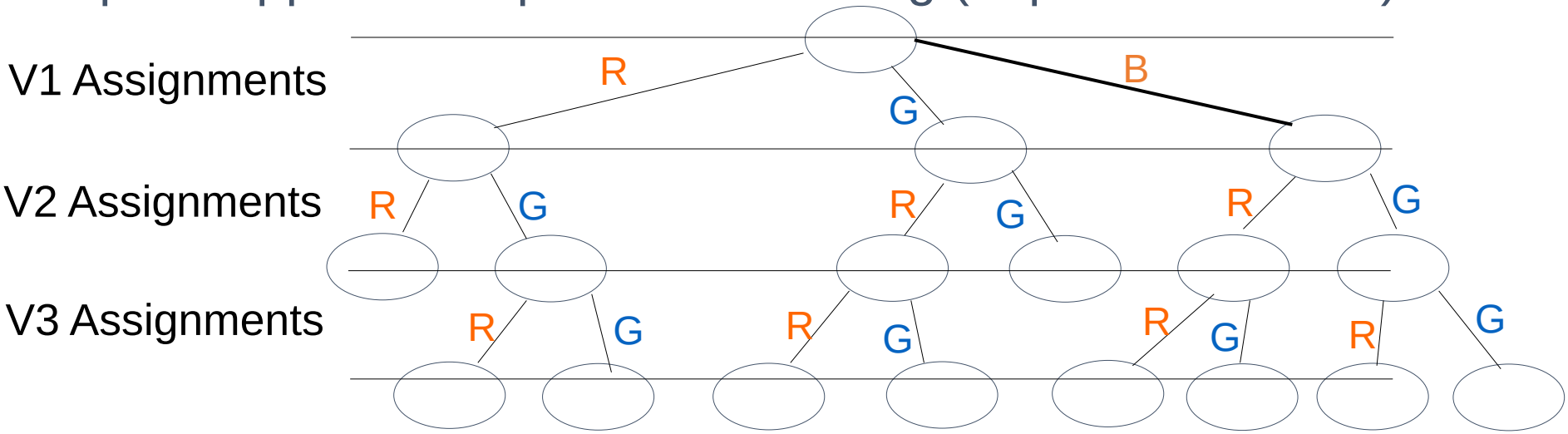
Arc consistent but 1
Solution B,R,G; B,G,R

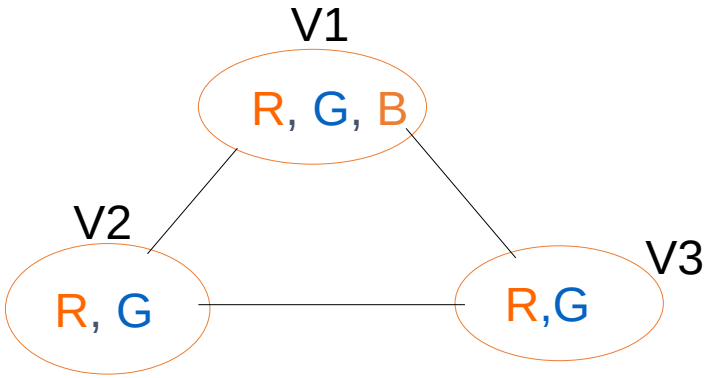
B, R not allowed

Need to do search to find solutions (if any)

Searching for solutions-backtracking (BT)

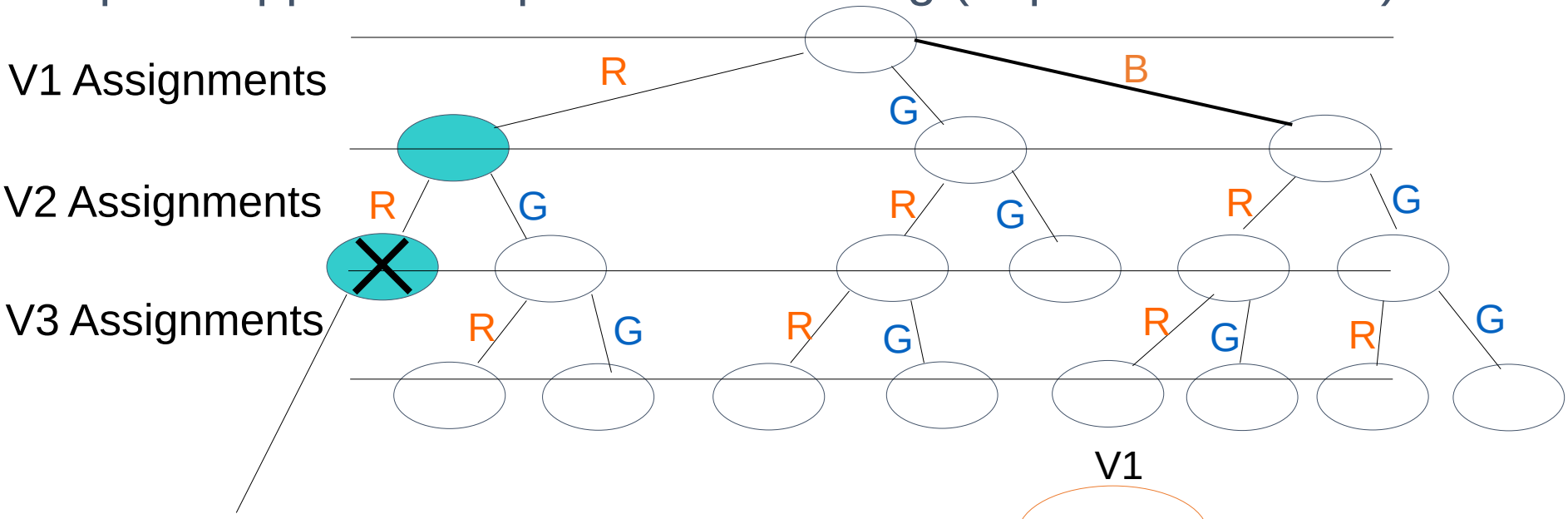
When we have too many values in domain (and/or constraints are weak), arc consistency doesn't do much. So, we need to search. Simplest approach is pure backtracking (depth-first search).





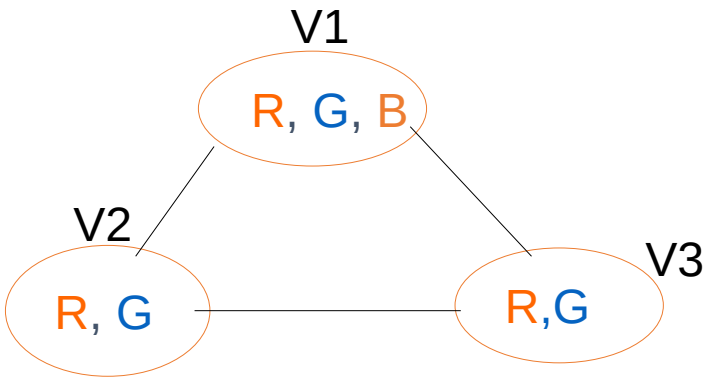
Searching for solutions-backtracking (BT)

When we have too many values in domain (and/or constraints are weak), arc consistency doesn't do much. So, we need to search. Simplest approach is pure backtracking (depth-first search).



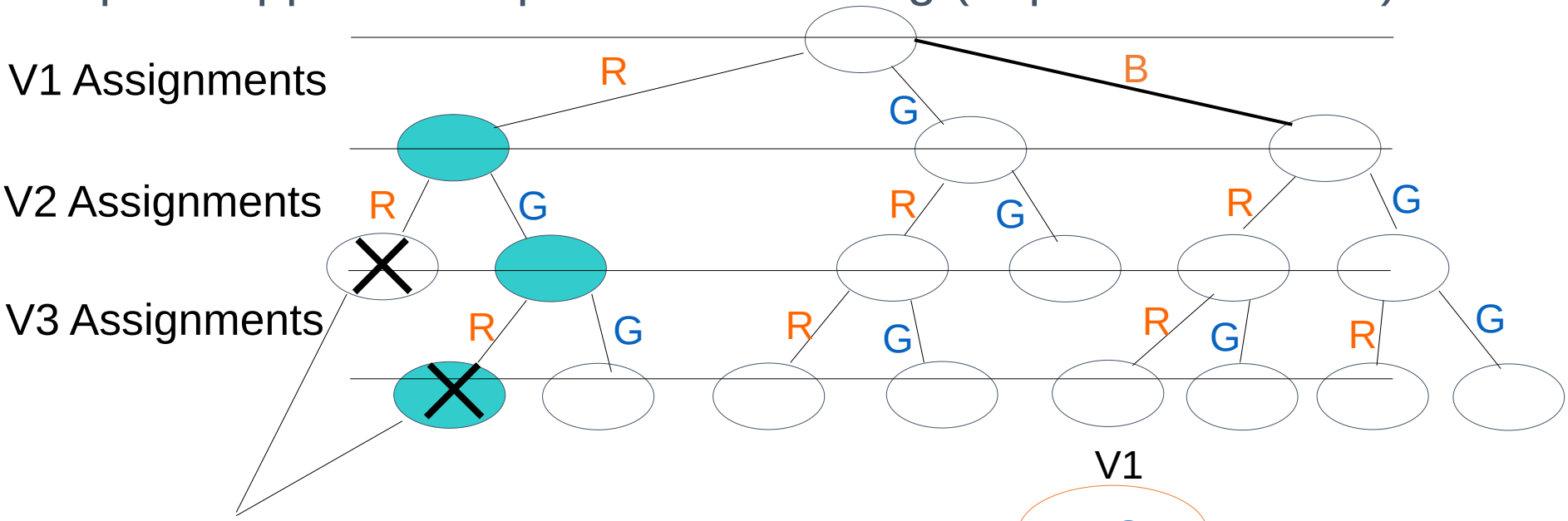
Inconsistent with V1 = R

Backup at inconsistent assignment



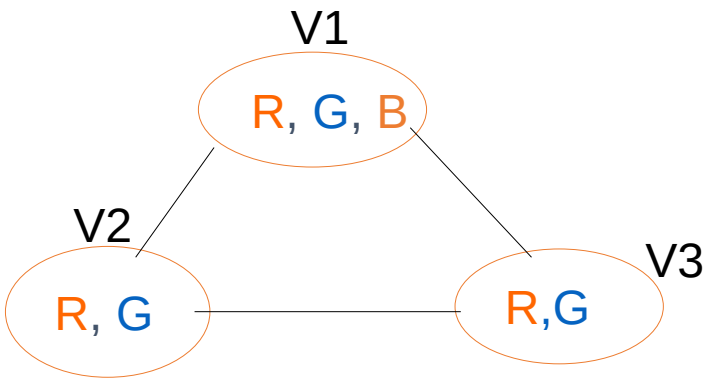
Searching for solutions-backtracking (BT)

When we have too many values in domain (and/or constraints are weak), arc consistency doesn't do much. So, we need to search. Simplest approach is pure backtracking (depth-first search).



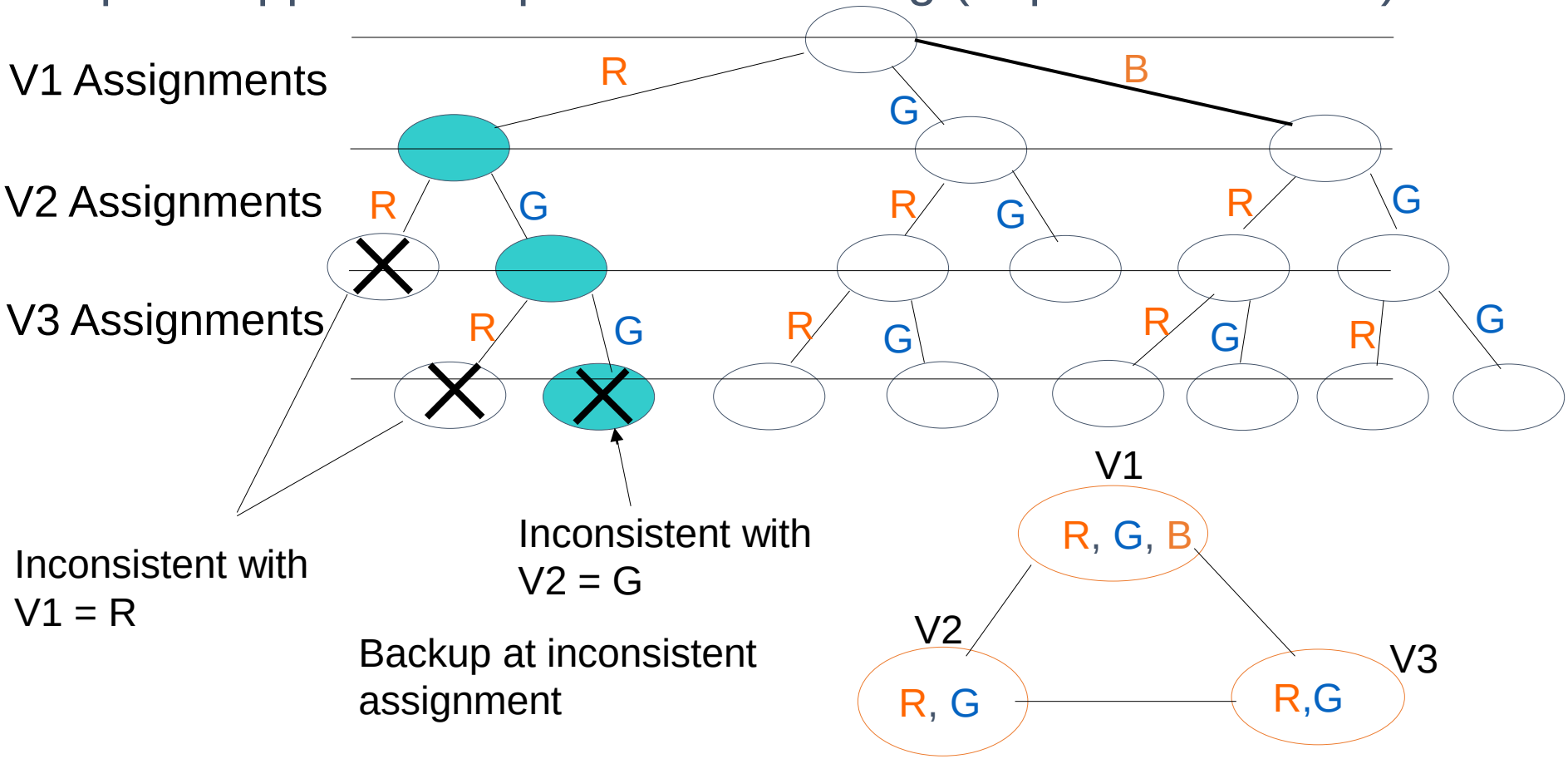
Inconsistent with V1 = R

Backup at inconsistent assignment



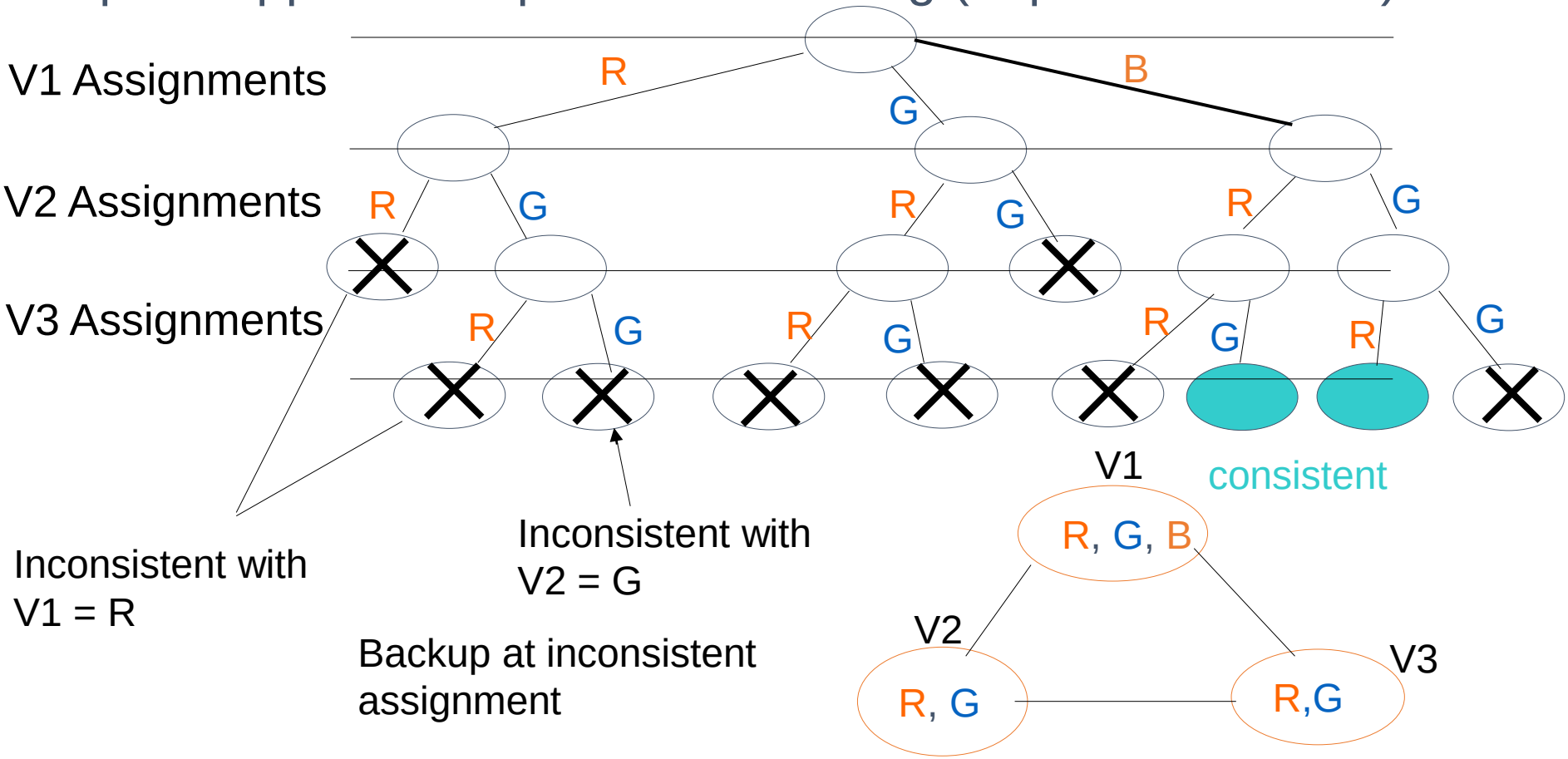
Searching for solutions-backtracking (BT)

When we have too many values in domain (and/or constraints are weak), arc consistency doesn't do much. So, we need to search. Simplest approach is pure backtracking (depth-first search).



Searching for solutions-backtracking (BT)

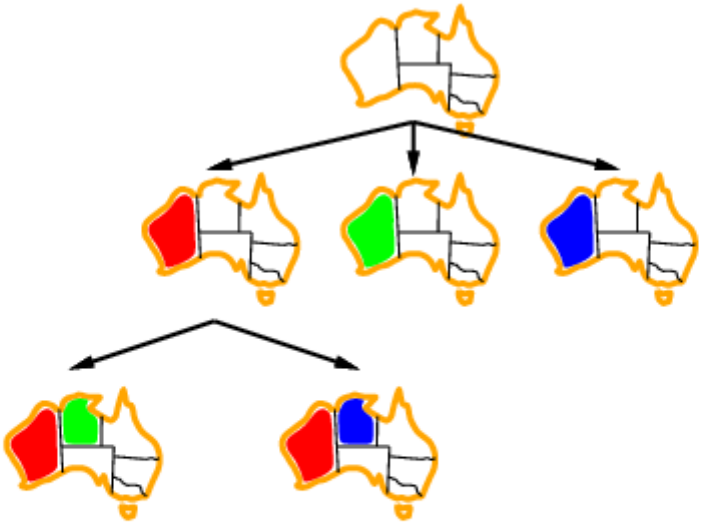
When we have too many values in domain (and/or constraints are weak), arc consistency doesn't do much. So, we need to search. Simplest approach is pure backtracking (depth-first search).



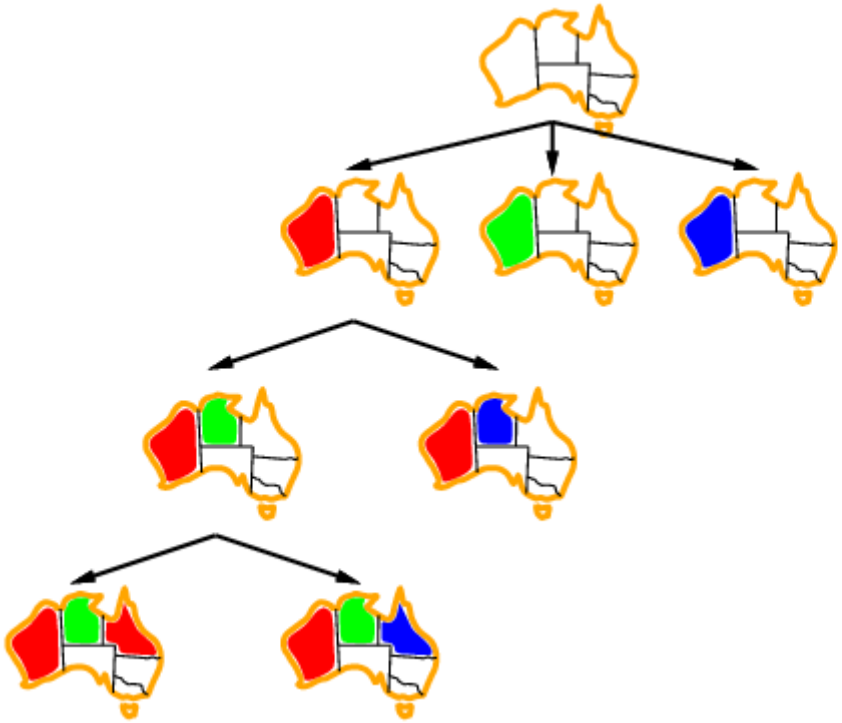
Backtracking example

Backtracking example

Backtracking example



Backtracking example



Improving backtracking efficiency

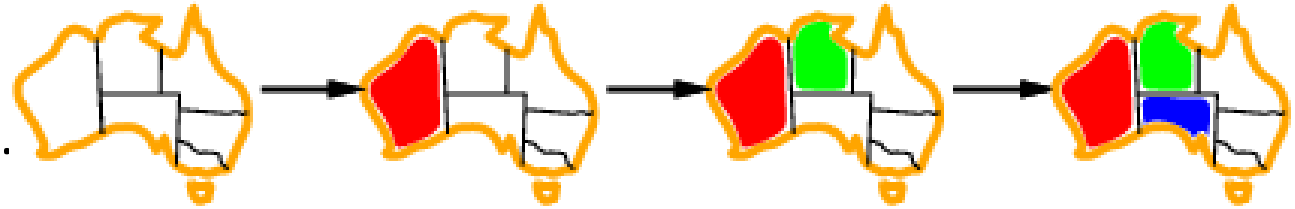
- General-purpose methods can give huge gains in speed:
  - Which variable should be assigned next?
  - In what order should its values be tried?
  - Can we detect inevitable failure early?

Most constrained variable

- Most constrained variable:

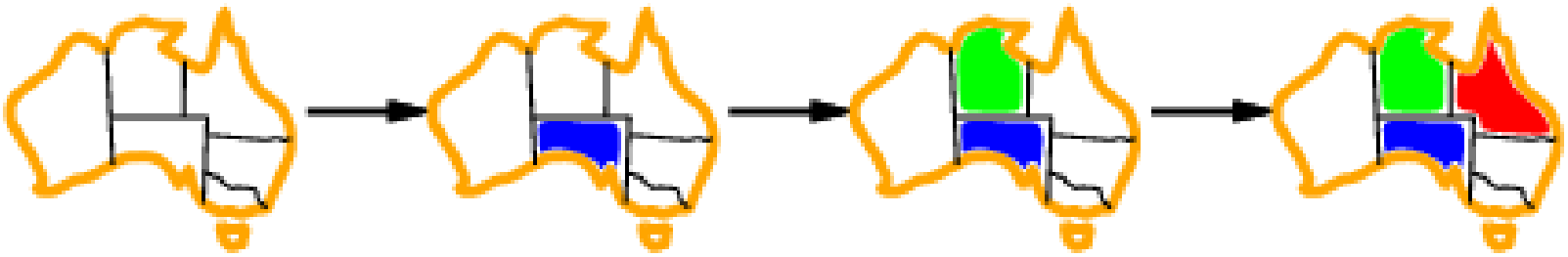
  choose the variable with the fewest legal values

- a.k.a.



Most constraining variable

- Tie-breaker among most constrained variables
- Most constraining variable:
  - choose the variable with the most constraints on remaining variables



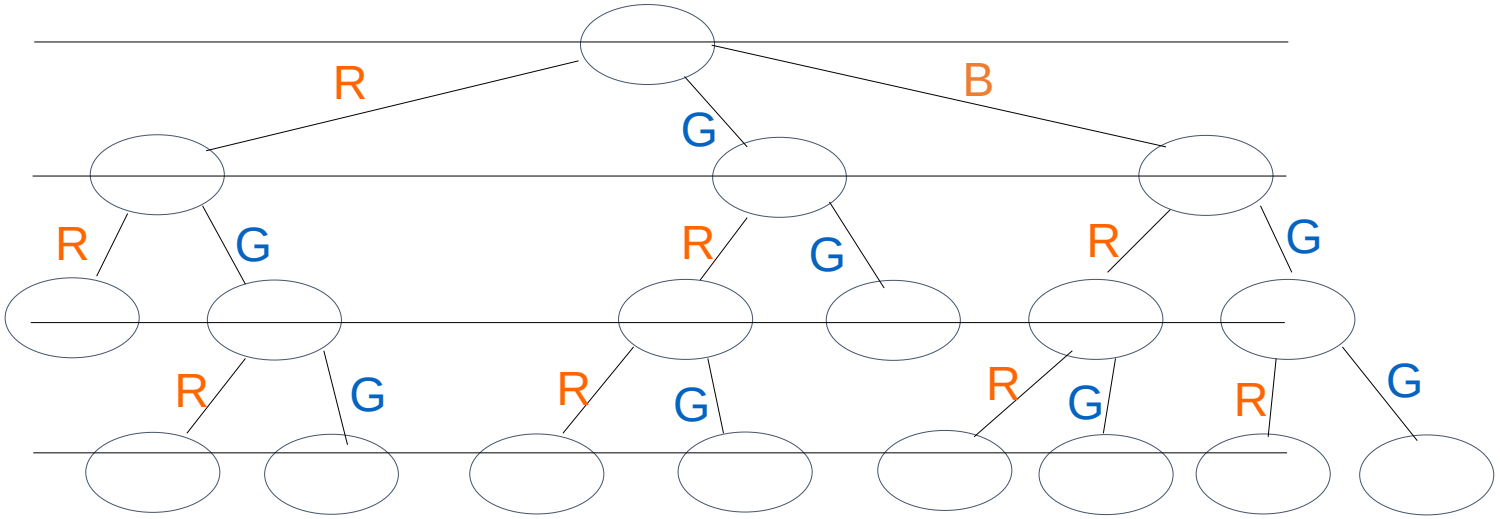
Least constraining value

- Given a variable, choose the least constraining value:
- the one that rules out the fewest values in the remaining variables

- Combining these heuristics makes 1000 queens feasible

Combine backtracking & constraint Propagation

A node in BT tree is <u>partial</u> assignment in which the domain of each variable has been set (tentatively) to singleton set.



Combine backtracking & constraint Propagation

A node in BT tree is partial assignment in which the domain of each variable has been set (tentatively) to singleton set.

Use constraint propagation (arc-consistency) to propagate the effect of this tentative assignment, i.e. eliminate values inconsistent with current values.

Combine backtracking & constraint Propagation

A node in BT tree is partial assignment in which the domain of each variable has been set (tentatively) to singleton set.

Use constraint propagation (arc-consistency) to propagate the effect of this tentative assignment, i.e. eliminate values inconsistent with current values.

Question: How much propagation to do?

Combine backtracking & constraint Propagation

A node in BT tree is partial assignment in which the domain of each variable has been set (tentatively) to singleton set.

Use constraint propagation (arc-consistency) to propagate the effect of this tentative assignment, i.e. eliminate values inconsistent with current values.

Question: How much propagation to do?

Answer: Not much, just local propagation from domains with unique assignments, which is called Forward Checking (FC). This conclusion is not necessarily obvious, but it generally holds in practice.

Constraint propagation

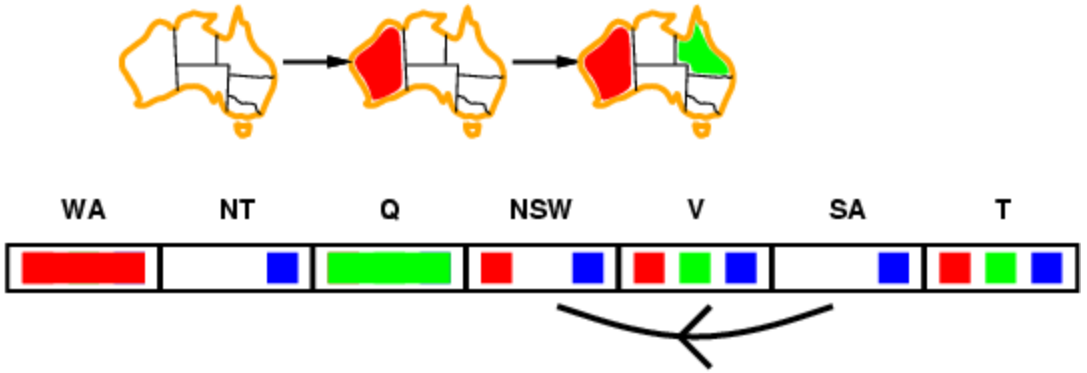
- Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:
- 

- NT and SA cannot both be blue!
- Constraint propagation repeatedly enforces constraints locally

Arc consistency

- Simplest form of propagation makes each arc consistent
- $X \rightarrow Y$ is consistent iff
-

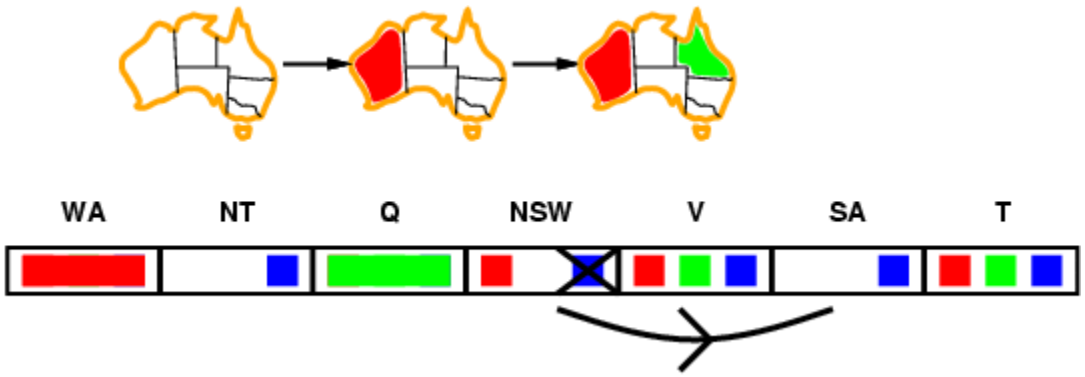
  for every value x of X there is some allowed y



Arc consistency

- Simplest form of propagation makes each arc consistent
- $X \rightarrow Y$ is consistent iff
- ———————————————

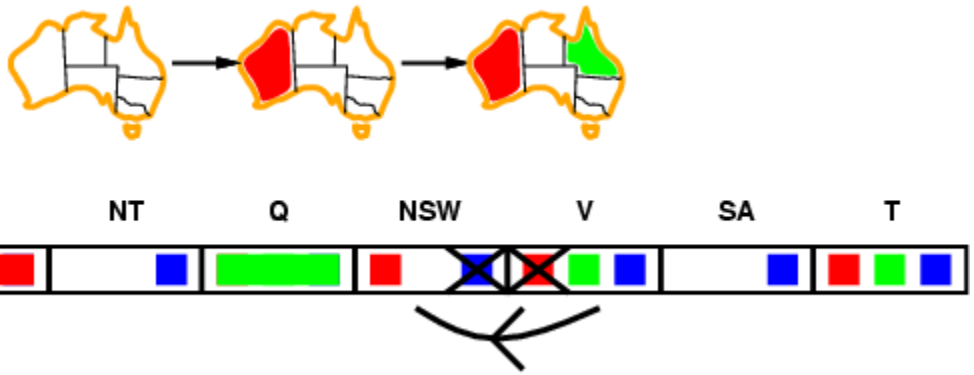
  for every value x of X there is some allowed y



Arc consistency

- Simplest form of propagation makes each arc consistent
- *X* → *Y* is consistent iff

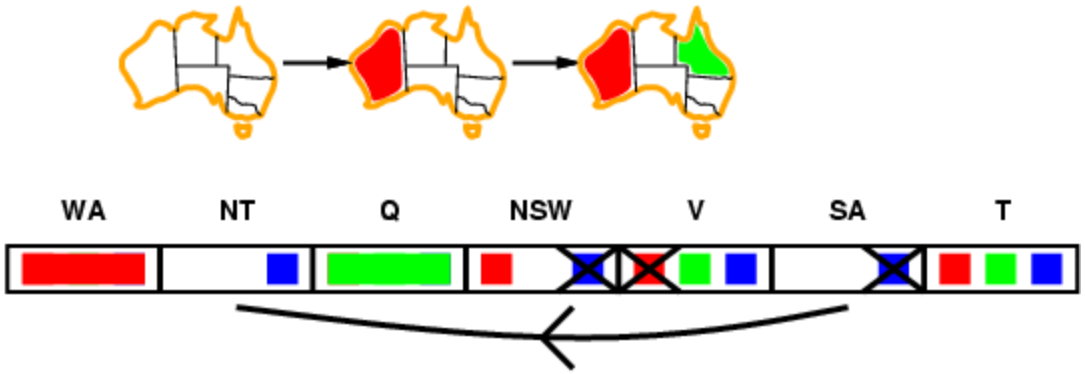
  for every value *x* of *X* there is some allowed *y*



- If *X* loses a value, neighbors of *X* need to be rechecked

Arc consistency

- Simplest form of propagation makes each arc consistent
- *X* → *Y* is consistent iff
- for every value *x* of *X* there is some allowed *y*

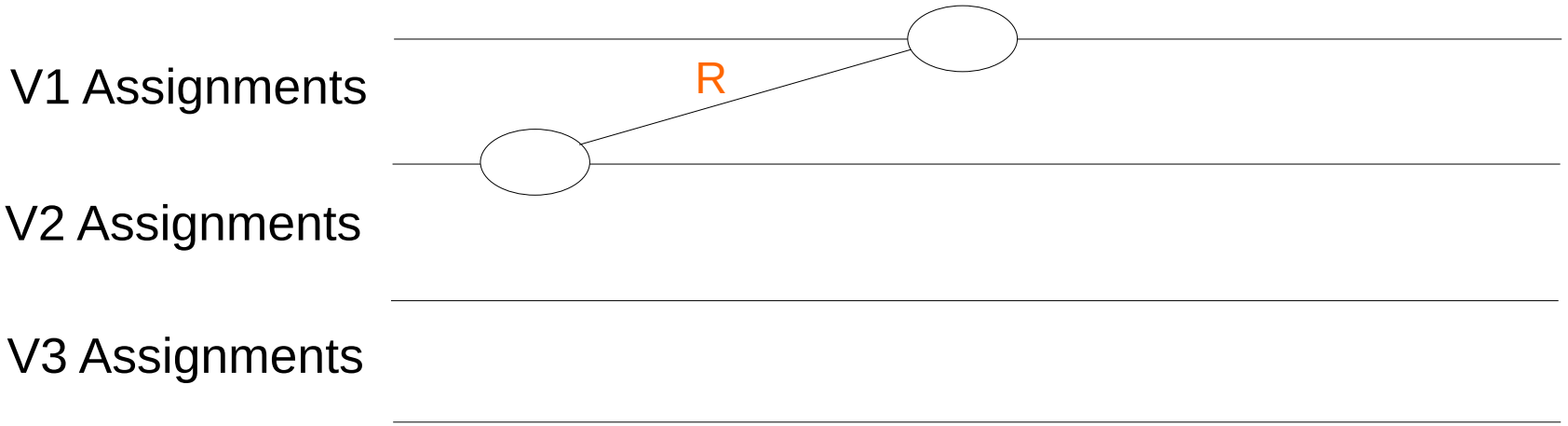


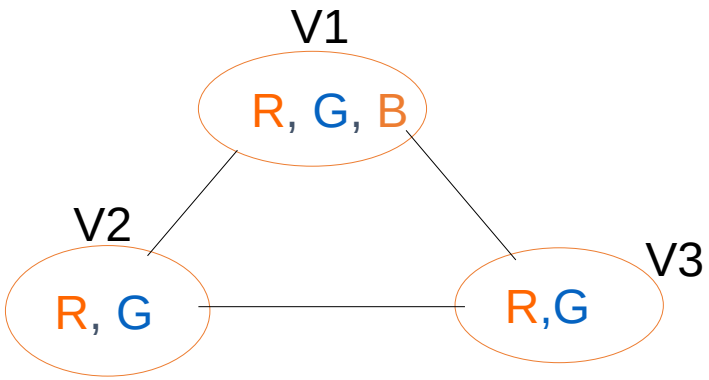
If *X* loses a value, neighbors of *X* need to be rechecked

- Arc consistency detects failure earlier than forward checking
- Can be run as a preprocessor or after each assignment

Backtracking with Forward Checking (BT-FC)

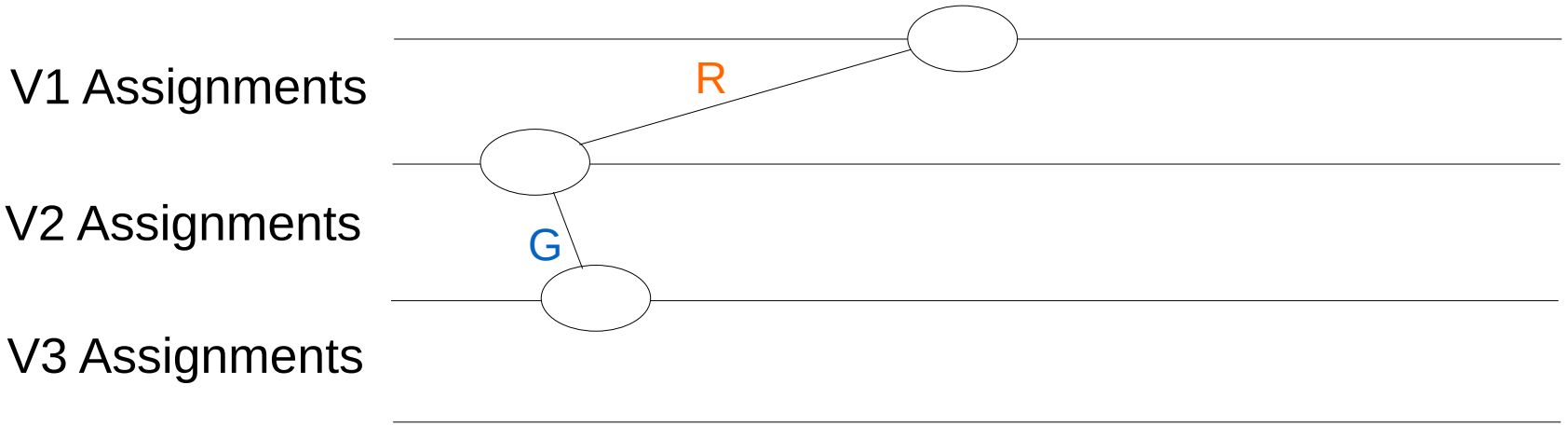
When examining assignment $v_i=d_k$, remove any values inconsistent with that assignment from neighboring domains in constraint graph.

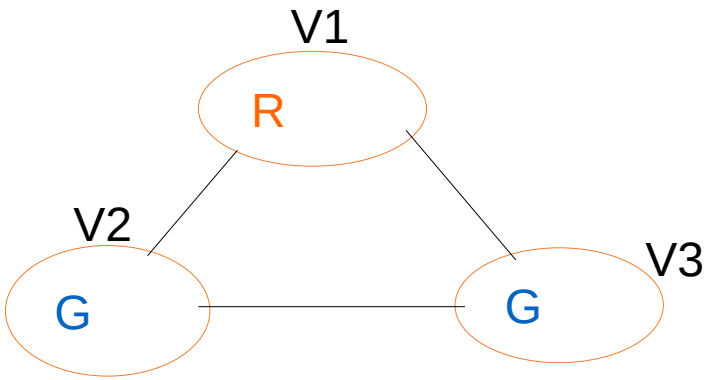




Backtracking with Forward Checking (BT-FC)

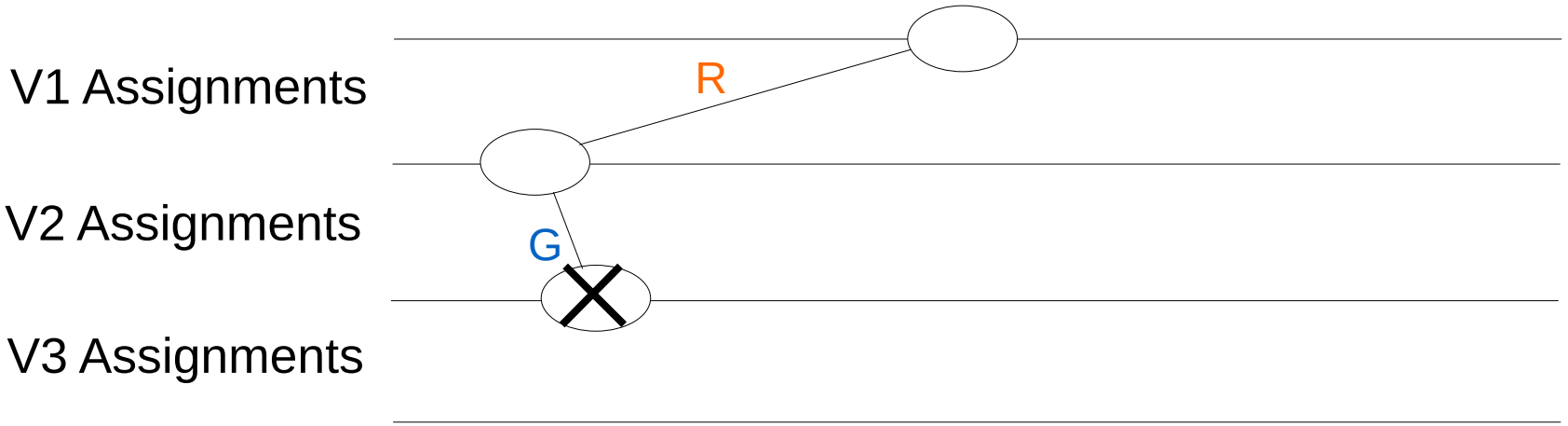
When examining assignment $v_i=d_k$, remove any values inconsistent with that assignment from neighboring domains in constraint graph.



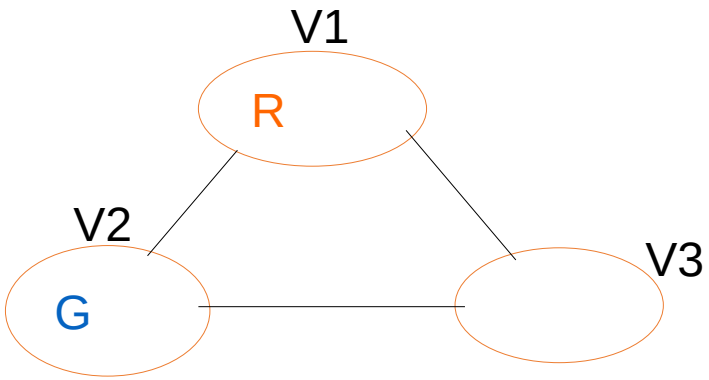


Backtracking with Forward Checking (BT-FC)

When examining assignment $v_i=d_k$, remove any values inconsistent with that assignment from neighboring domains in constraint graph.



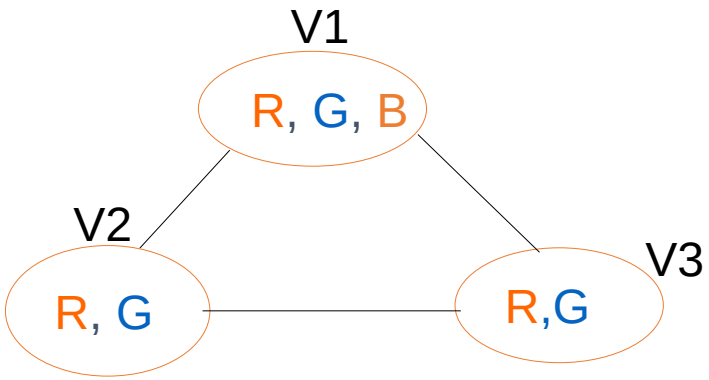
We have a conflict whenever a domain becomes empty



Backtracking with Forward Checking (BT-FC)

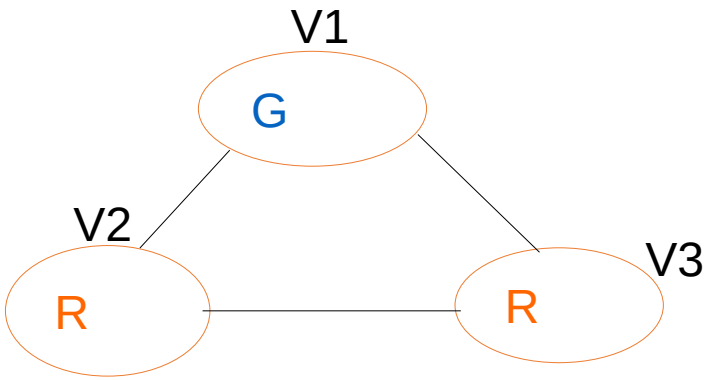
When examining assignment $v_i=d_k$, remove any values inconsistent with that assignment from neighboring domains in constraint graph.

When backing up, need to restore domain values, since deletions were done to reach consistency with tentative assignments considered during search.



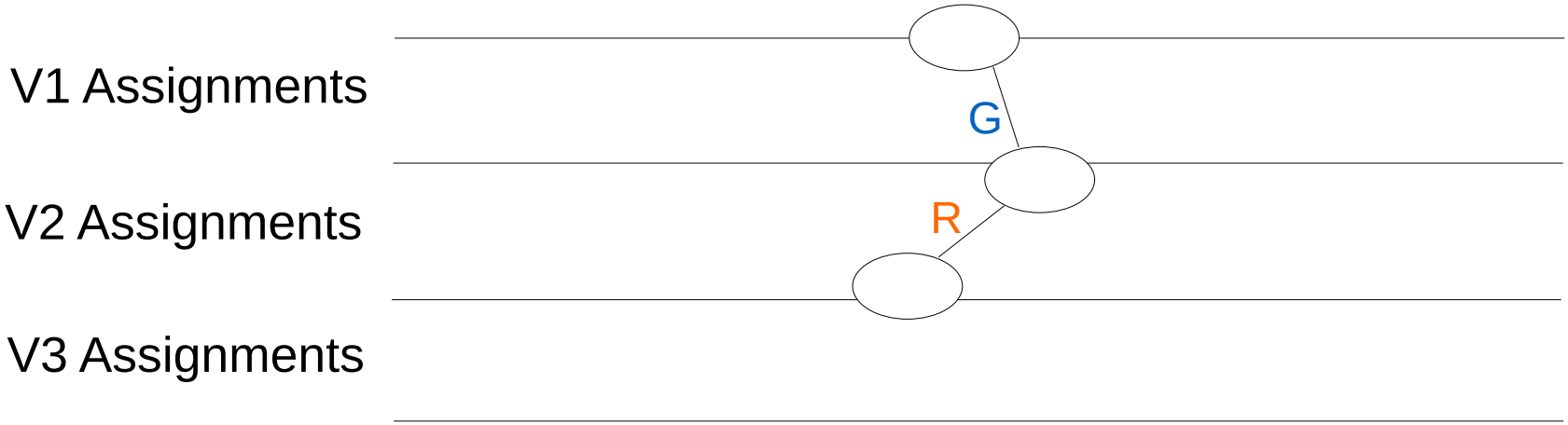
Backtracking with Forward Checking (BT-FC)

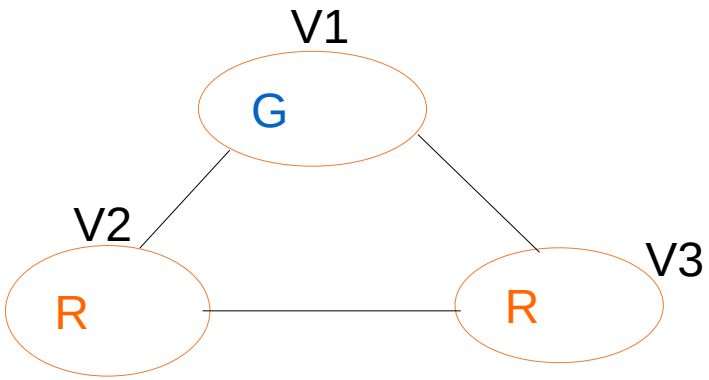
When examining assignment $v_i=d_k$, remove any values inconsistent with that assignment from neighboring domains in constraint graph.



Backtracking with Forward Checking (BT-FC)

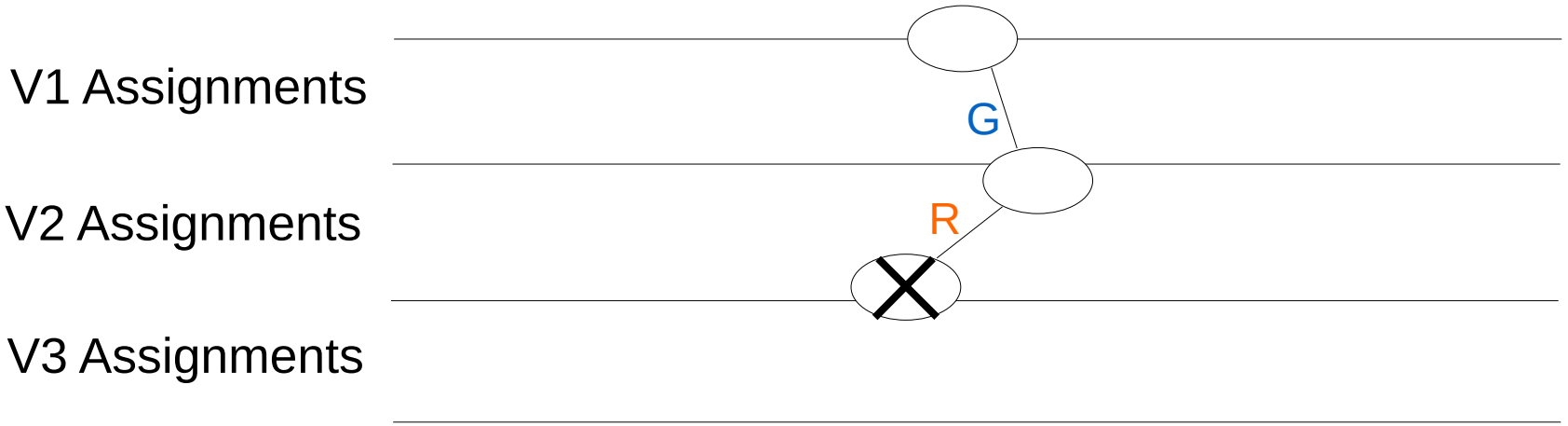
When examining assignment $v_i=d_k$, remove any values inconsistent with that assignment from neighboring domains in constraint graph.

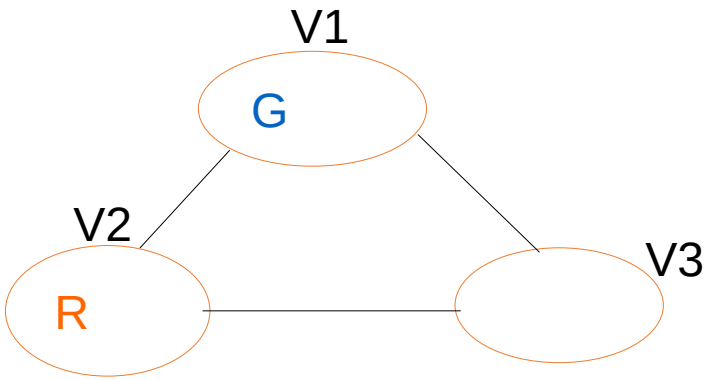




Backtracking with Forward Checking (BT-FC)

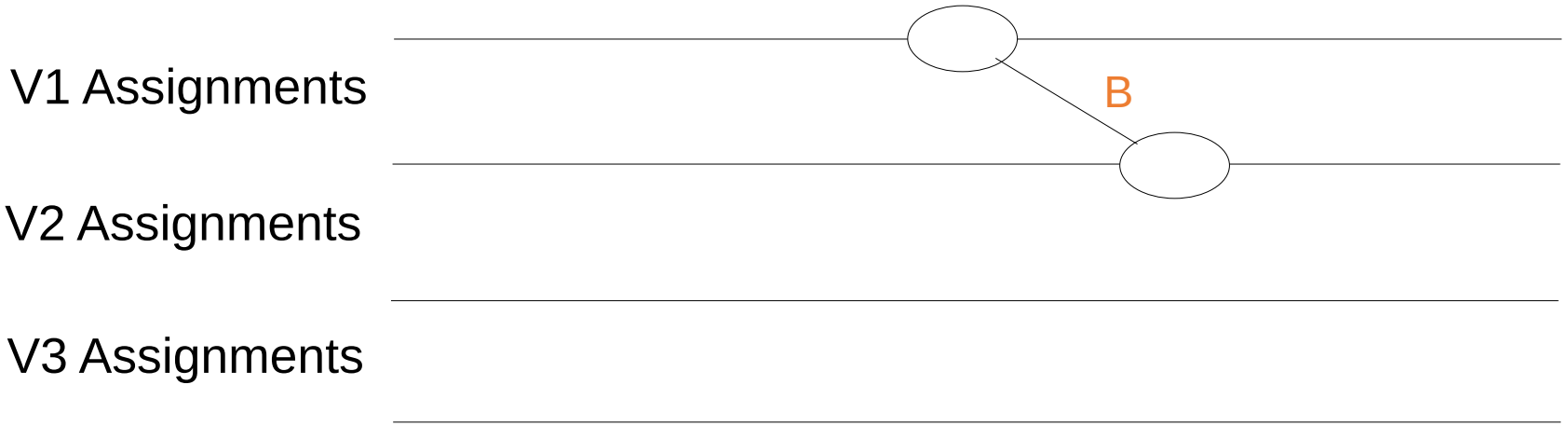
When examining assignment $v_i=d_k$, remove any values inconsistent with that assignment from neighboring domains in constraint graph.

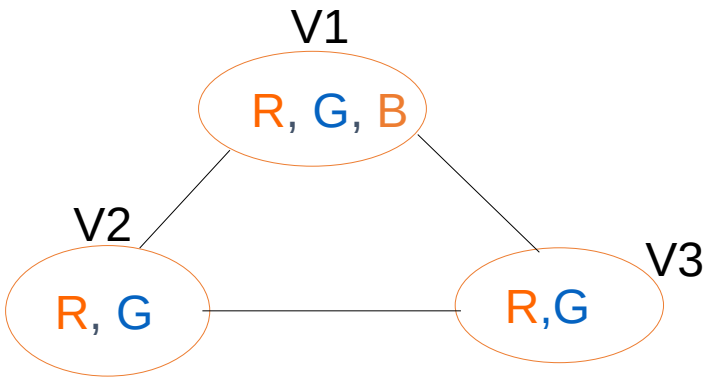




Backtracking with Forward Checking (BT-FC)

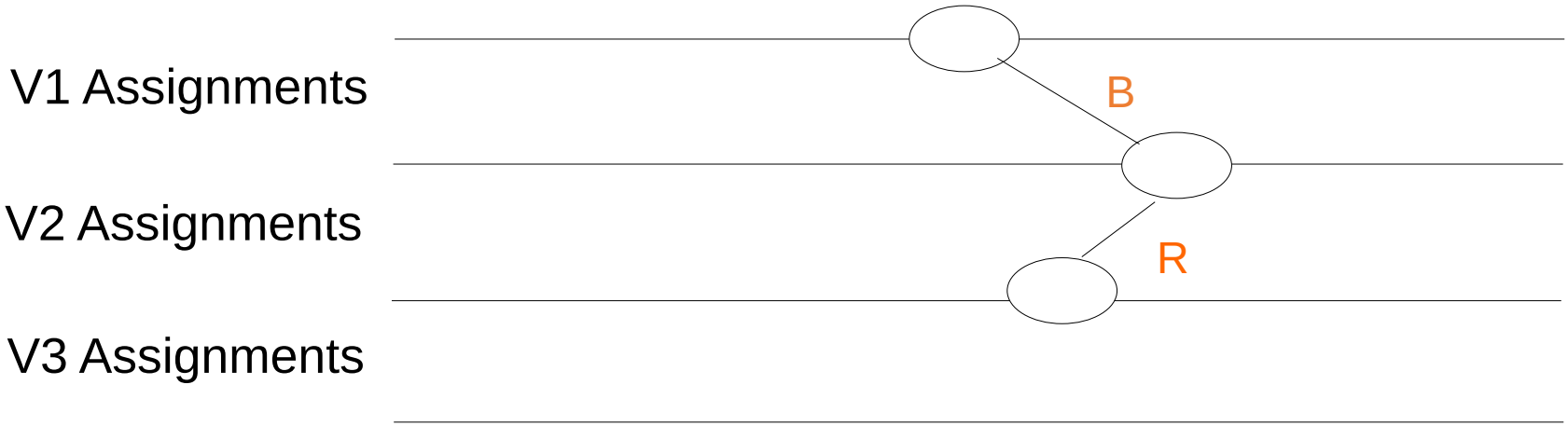
When examining assignment $v_i=d_k$, remove any values inconsistent with that assignment from neighboring domains in constraint graph.

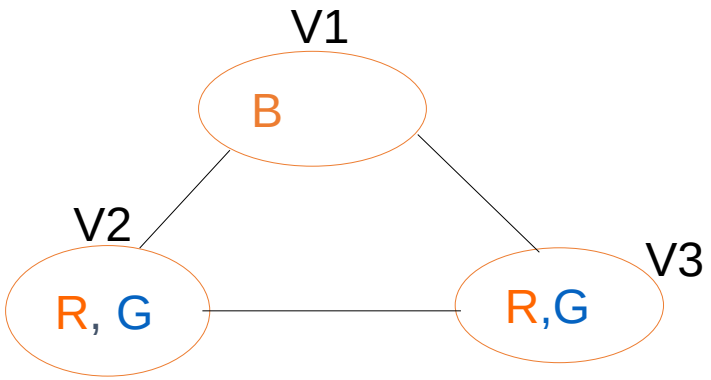




Backtracking with Forward Checking (BT-FC)

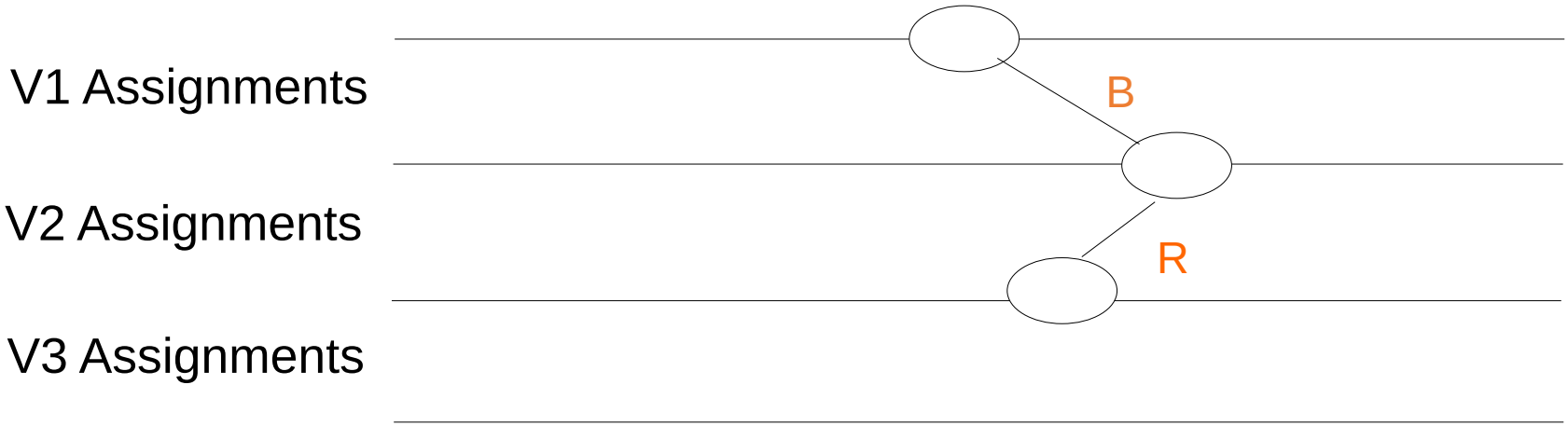
When examining assignment $v_i=d_k$, remove any values inconsistent with that assignment from neighboring domains in constraint graph.

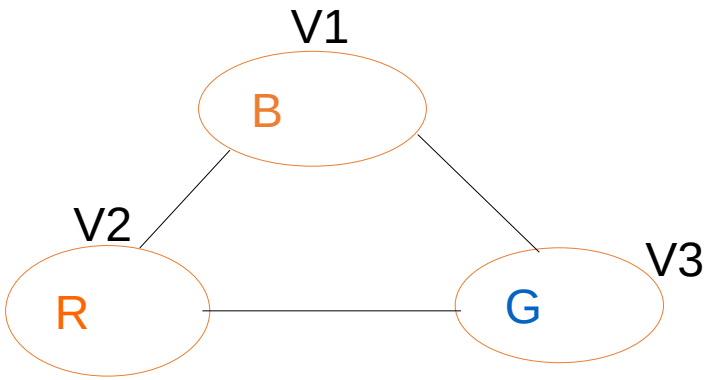




Backtracking with Forward Checking (BT-FC)

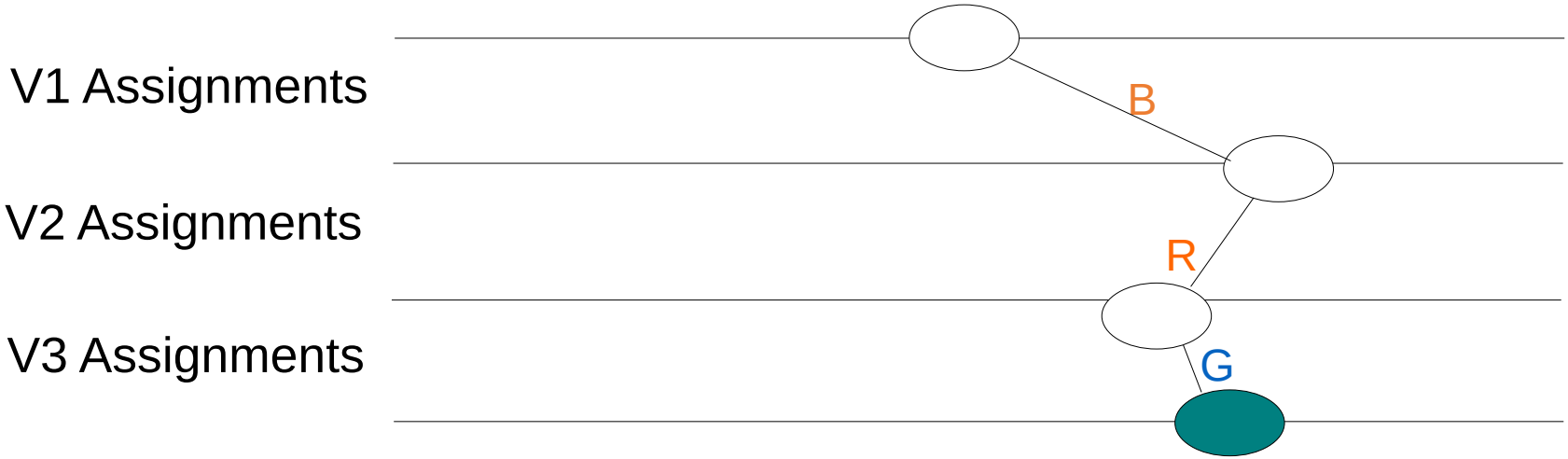
When examining assignment $v_i=d_k$, remove any values inconsistent with that assignment from neighboring domains in constraint graph.

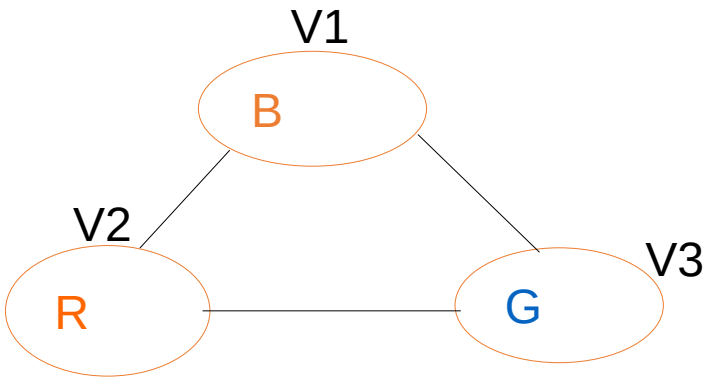




Backtracking with Forward Checking (BT-FC)

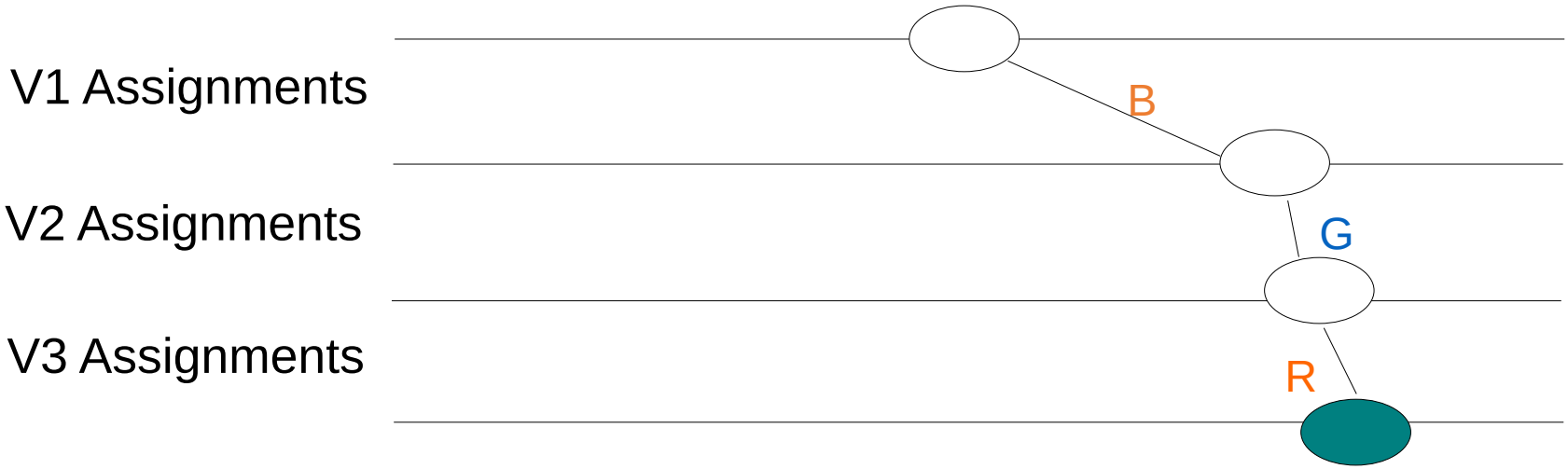
When examining assignment $v_i=d_k$, remove any values inconsistent with that assignment from neighboring domains in constraint graph.

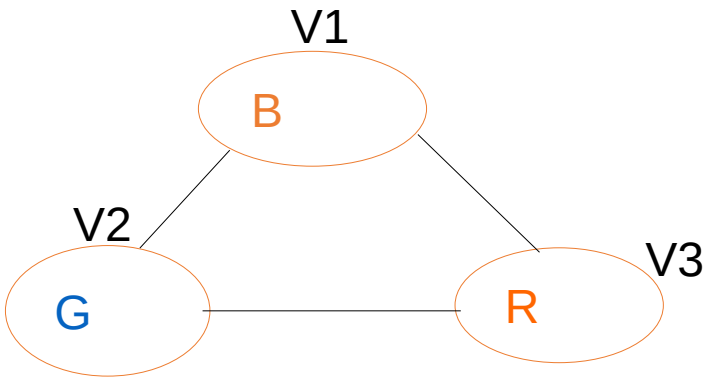




Backtracking with Forward Checking (BT-FC)

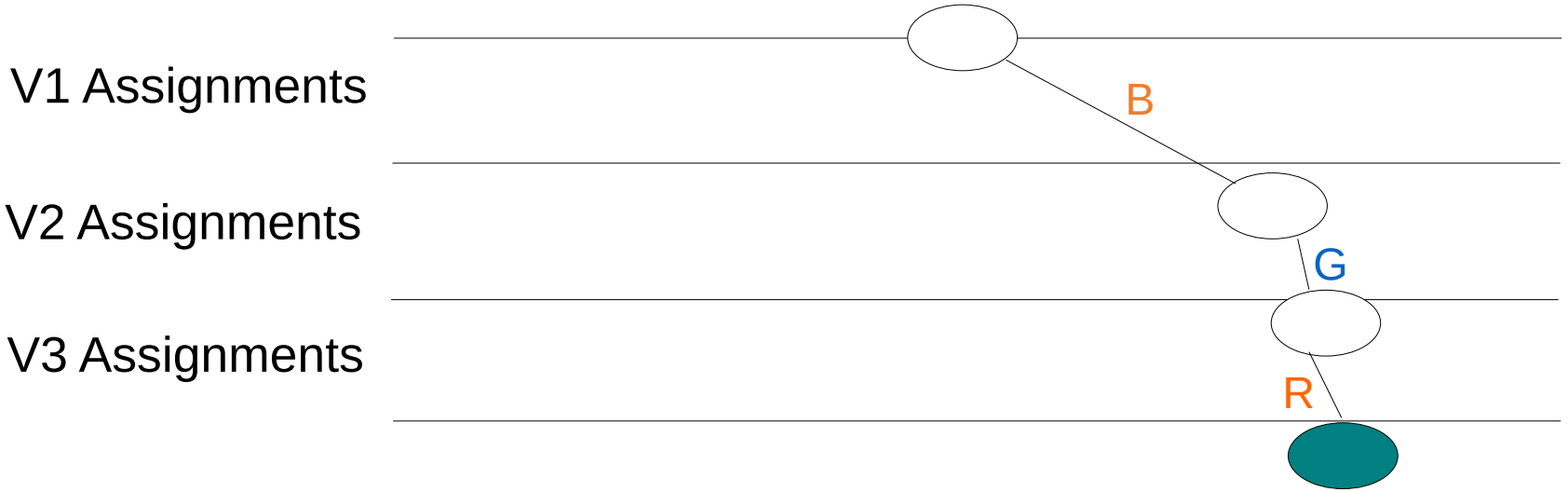
When examining assignment $v_i=d_k$, remove any values inconsistent with that assignment from neighboring domains in constraint graph.



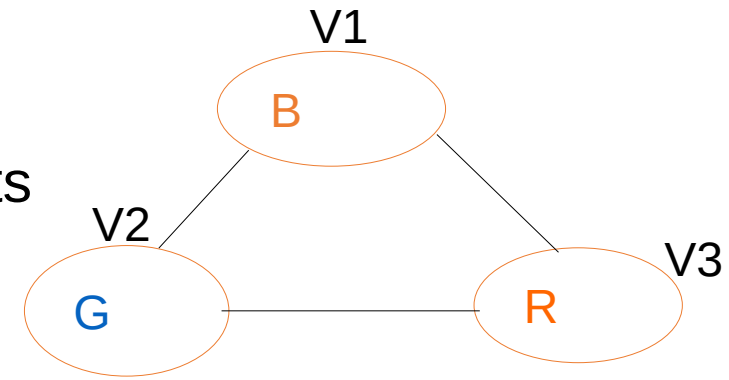


Backtracking with Forward Checking (BT-FC)

When examining assignment $v_i=d_k$, remove any values inconsistent with that assignment from neighboring domains in constraint graph.



No need to check previous assignments



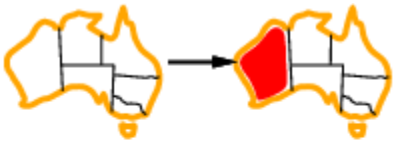
Generally preferable to pure BT

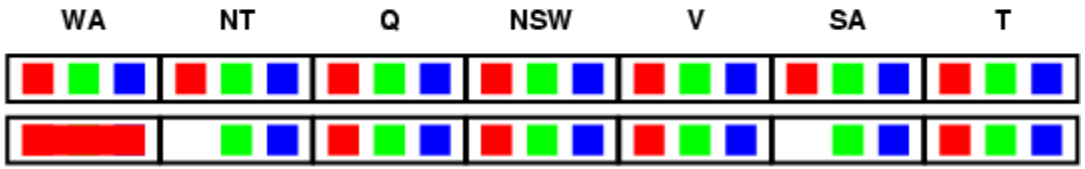
Forward checking

- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values
  -

Forward checking

- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values
  -



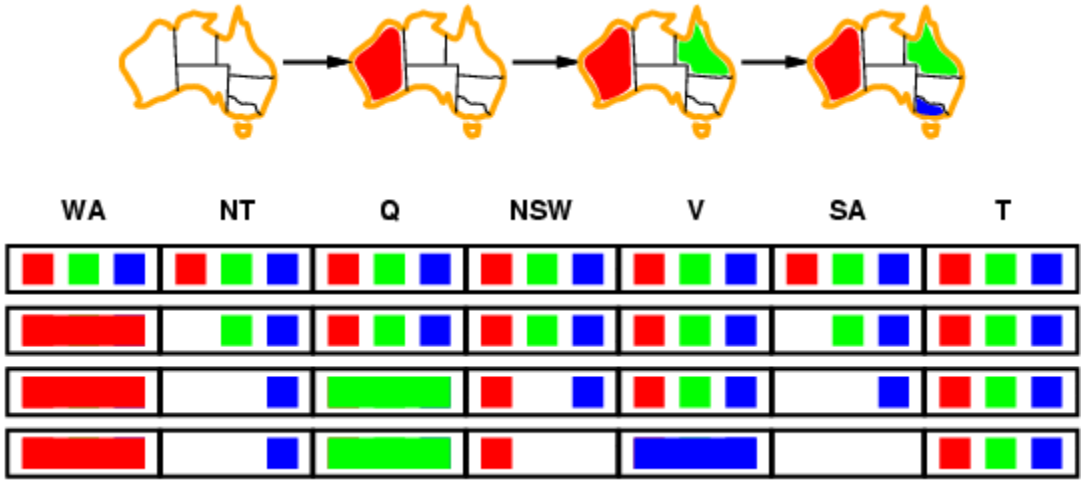


Forward checking

- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values
  -

Forward checking

- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values
  -



Backtracking with Forward Checking (BT-FC)

Backtracking (BT) with Forward checking (FC) is better to Plain BT as it eliminates the consideration of inconsistent assignments.

Makes the tree very simple for searching.

With BT if V3 is inconsistent with V1 – this would be discovered independently for every value of V2

But FC would delete it from the domain right away

Questions