# Artificial Intelligence

## Constraint Satisfaction Problems (CSPs)

CS 444 – Spring 2019
Dr. Kevin Molloy
Department of Computer Science
James Madison University

# Outline

- Examples of CSP
- Arc Consistency
- Example Problem

# Constraint Satisfaction Problems (CSPs)

Standard Search Problem:

State is a "black box" – any old data structure that supports a goal test, eval, successors, etc.

**CSPs:**

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

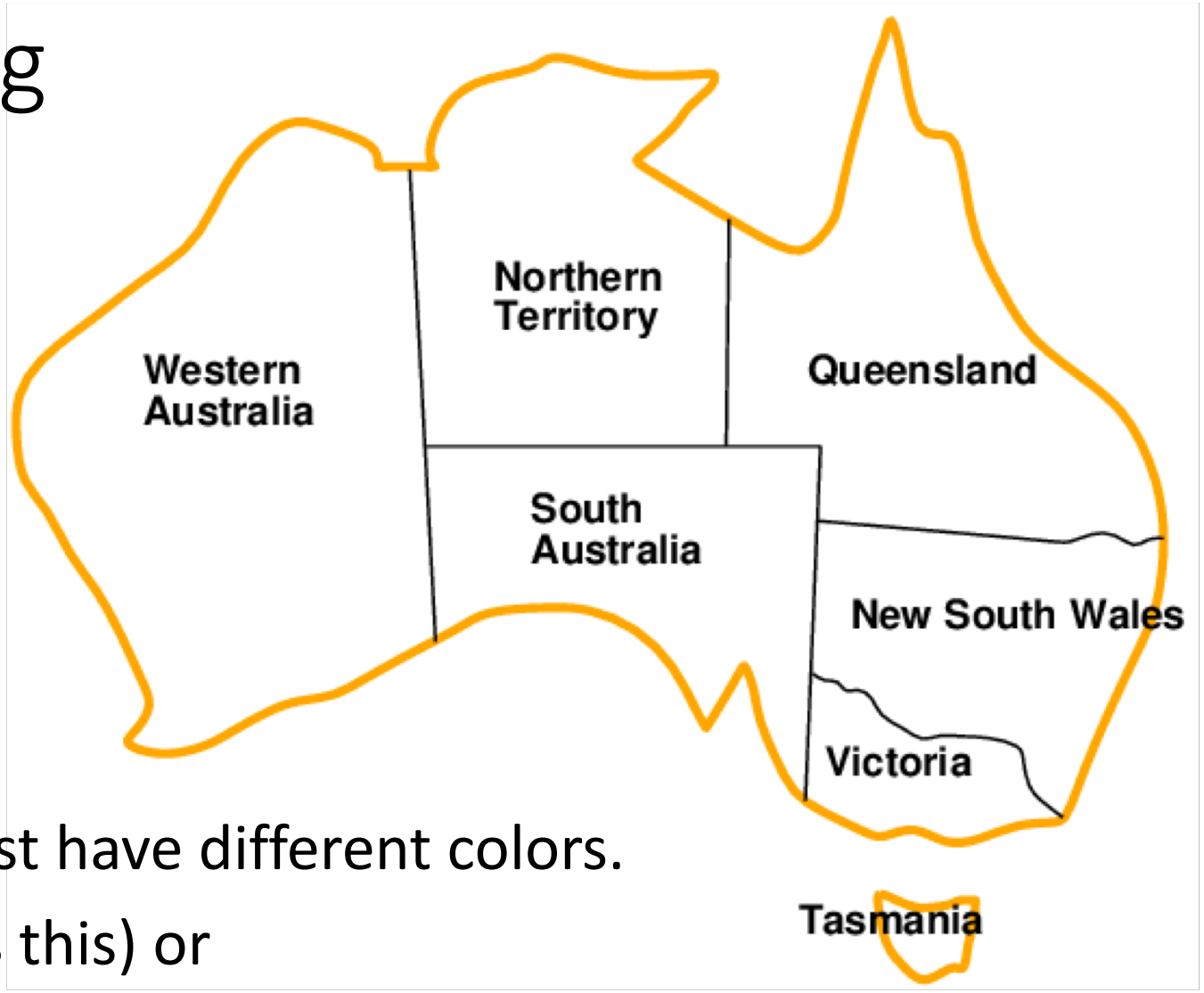
Allows useful **general-purpose algorithms** with more power than standard search algorithms (avoids coming up with problem specific heuristics).

# 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** (if language allows this) or

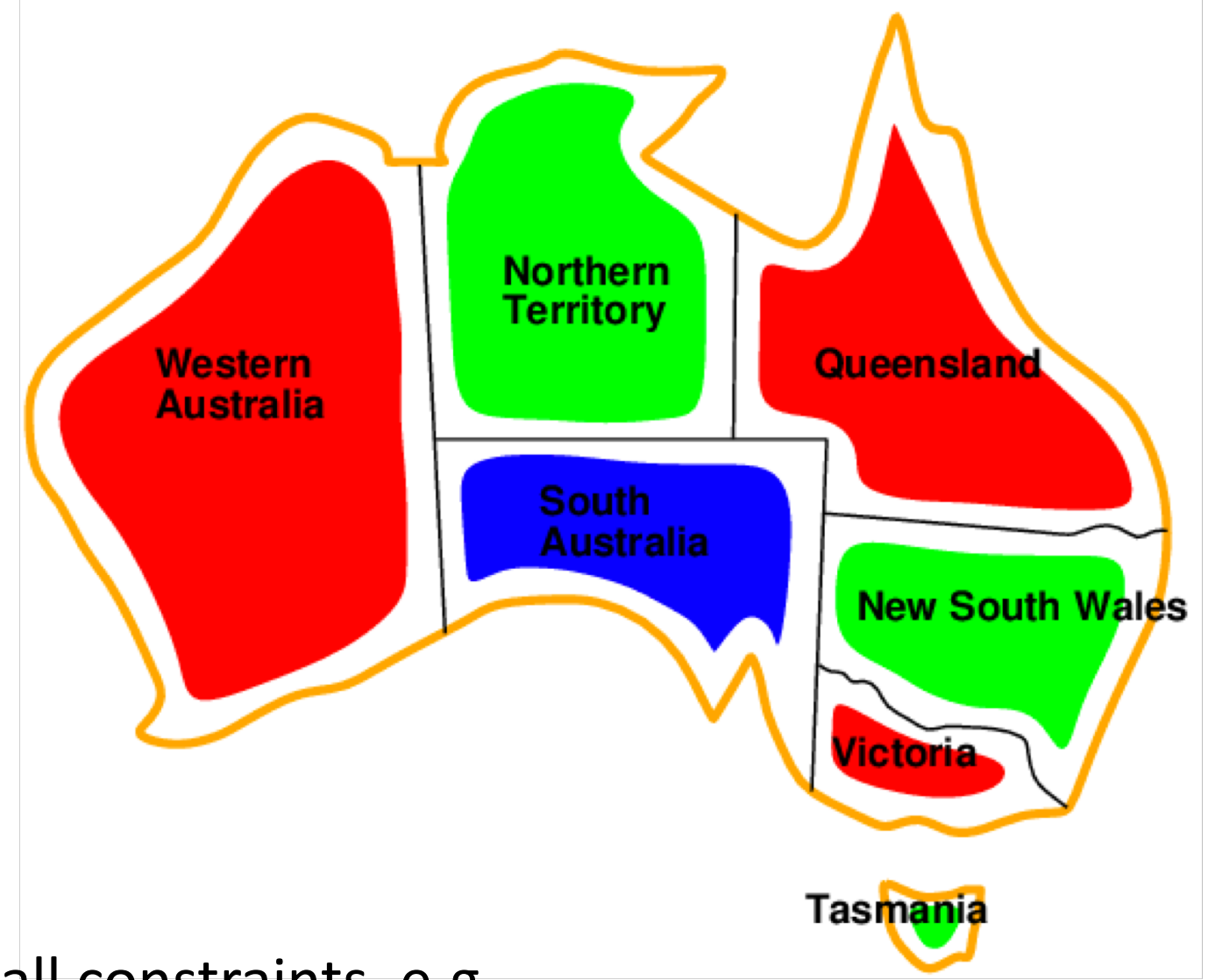
(WA, NT) ∈ { (red, green), (red, blue), (green, red), …}

# Example: Map Coloring

Variables:

WA, NT, Q, NSW, V, SA, T

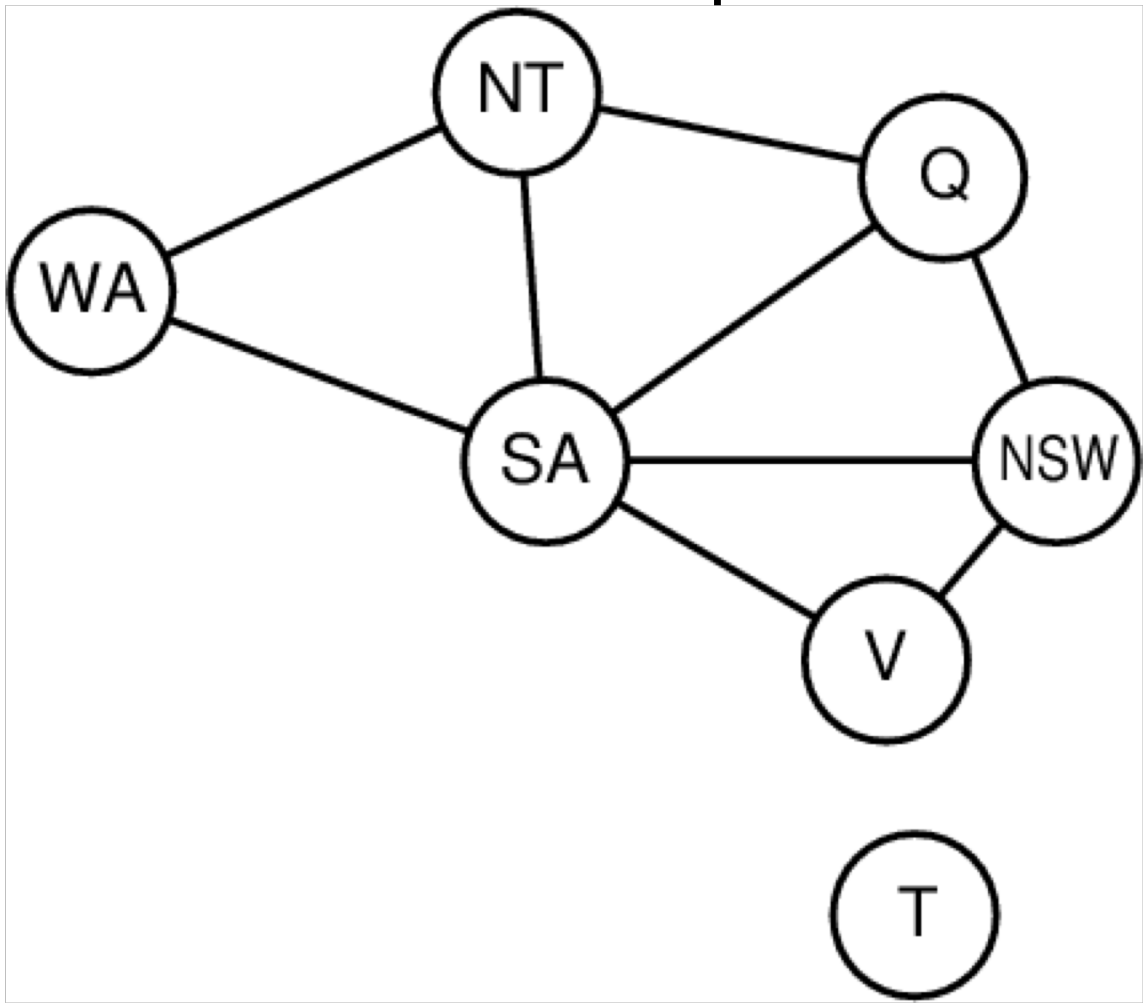
Domains $D_i$ = {red, green, blue}




Solutions are assignments satisfying all constraints, e.g.,

{WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green)

# Constraint Graph



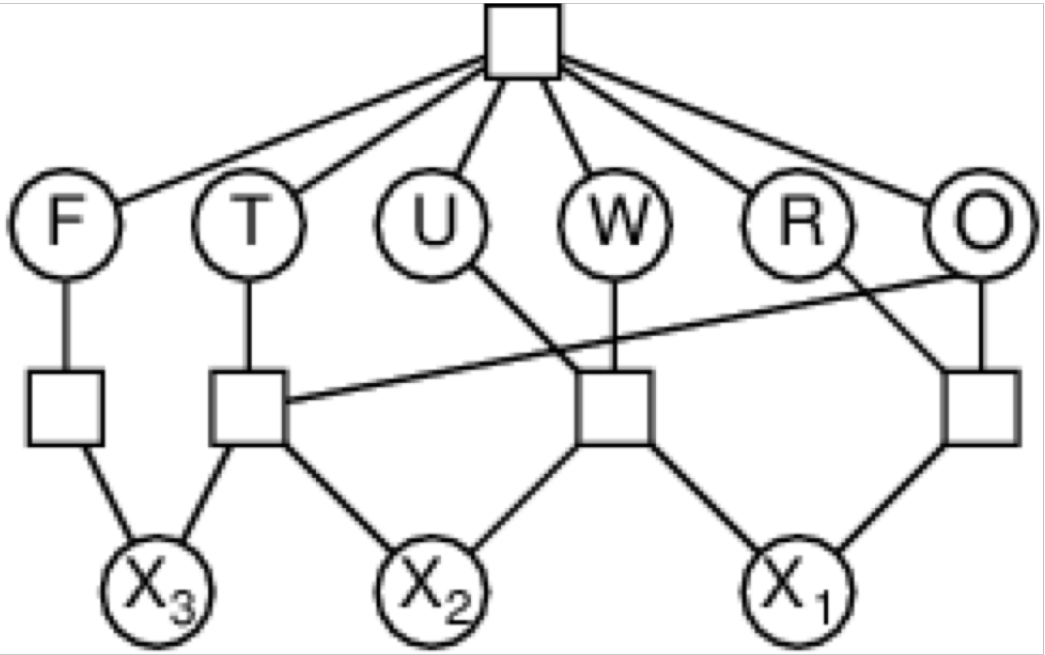

**Binary CSP**: each constraint relates at most two variables

**Constraint graph**: nodes are variables, arc show constraints

**General-purpose CSP algorithms use the graph structure** to speed up search (e.g., Tasmania is an independent subproblem)

# Example: Cryptarithmetic

```
   T W O
+  T W O
-------
F O U R
```



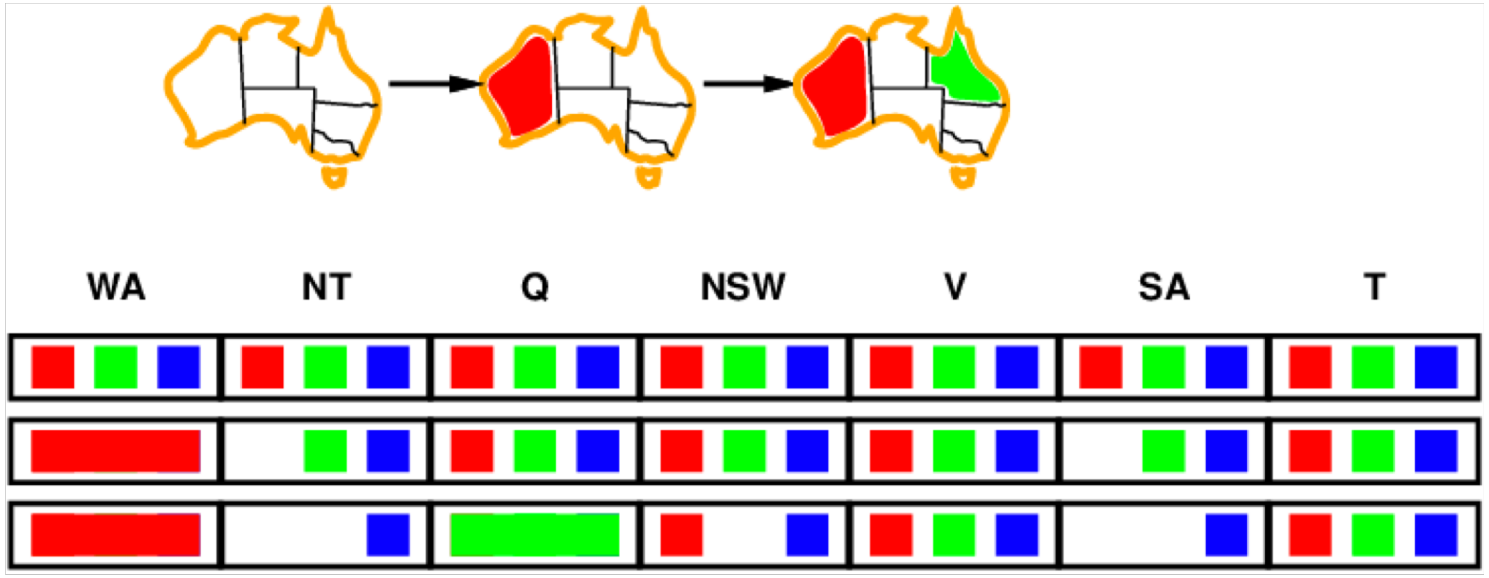

- Variables: F T U W R O $X_1$ $X_2$ $X_3$
- Domains: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
- alldiff(F, T, U, W, R, O)
- $O + O = R + 10 * X_1$
- $W + W + X_1 = U + 100 * X_2$
- Etc..

**Global constraints** involves an arbitrary number of variables (not necessarily all variables).

If m variables are involved, and I have n remaining values, if m > n, this constraint can not be satisfied.

# Constraint Propagation

Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection of ALL failures




BUT: NT and SA cannot both be blue!

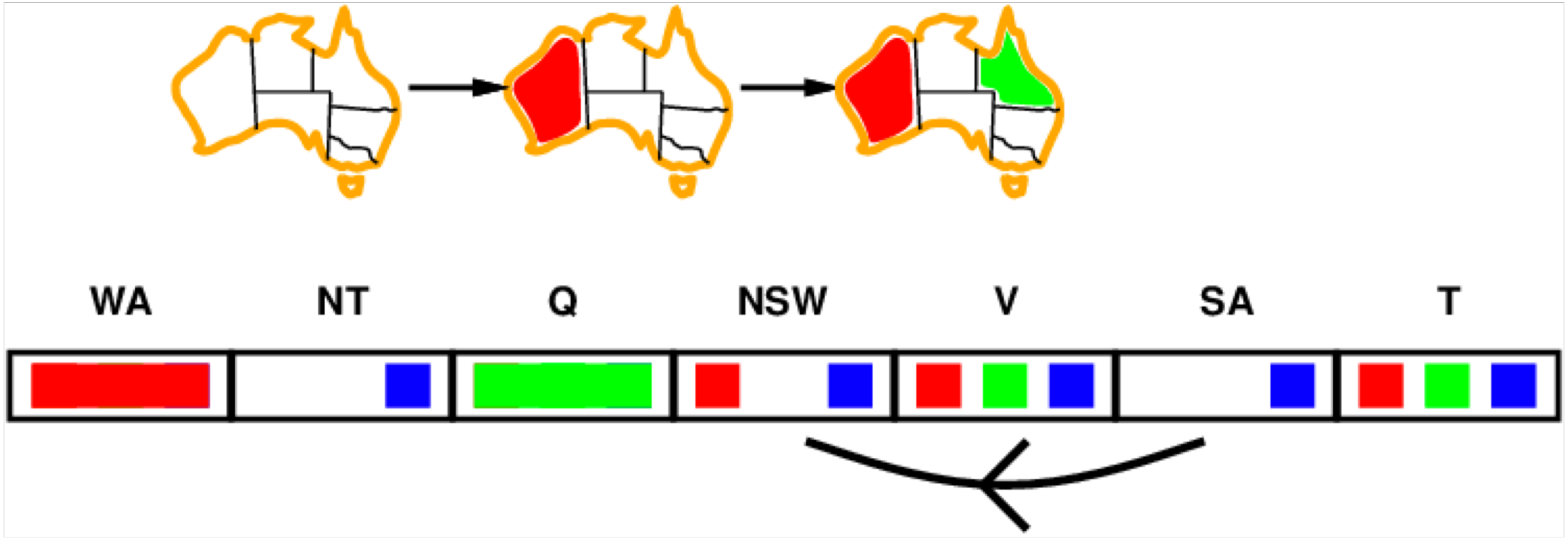
Constraint propagation enforces constraints locally at each step (over and over), and does not "chase" arc consistency

When the domain of a neighbor Y of X is reduced, domains of neighbors of Y may also become inconsistent (e.g., NT and SA).

# Arc Consistency

Simplest form of constraint propagation makes each arc consistent

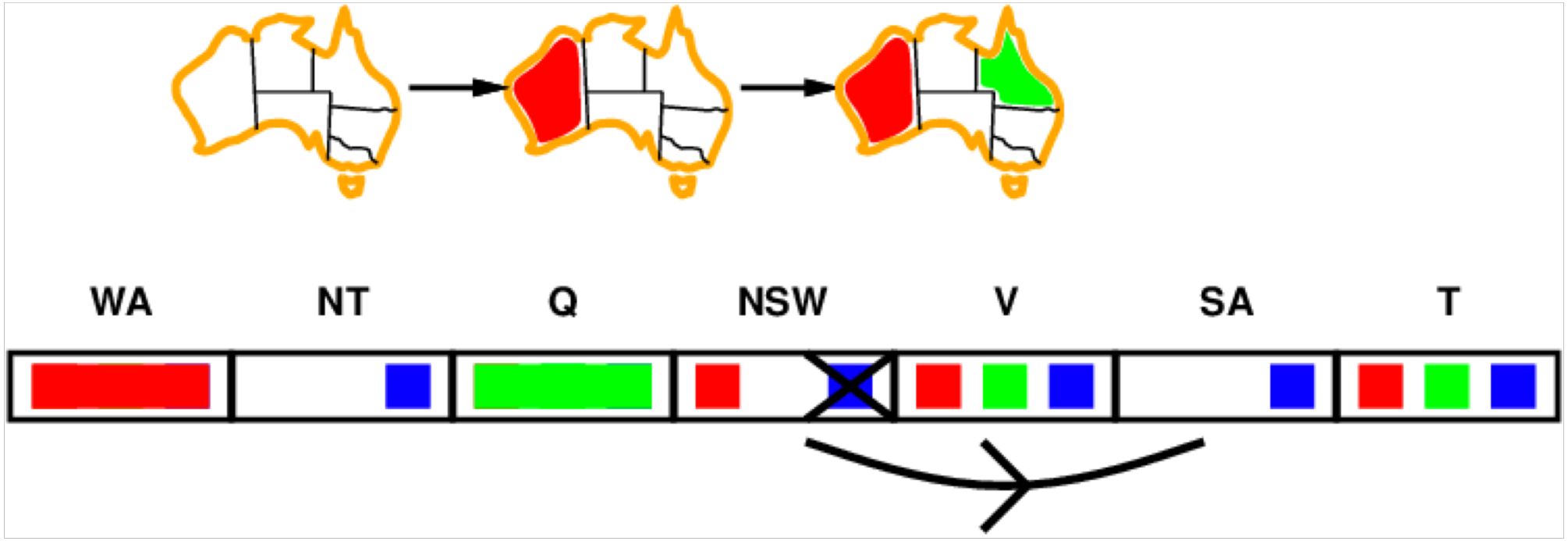
X → Y is consistent iff for every value of x of X there is some allowed value y of Y



# Arc Consistency

Simplest form of constraint propagation makes each arc consistent

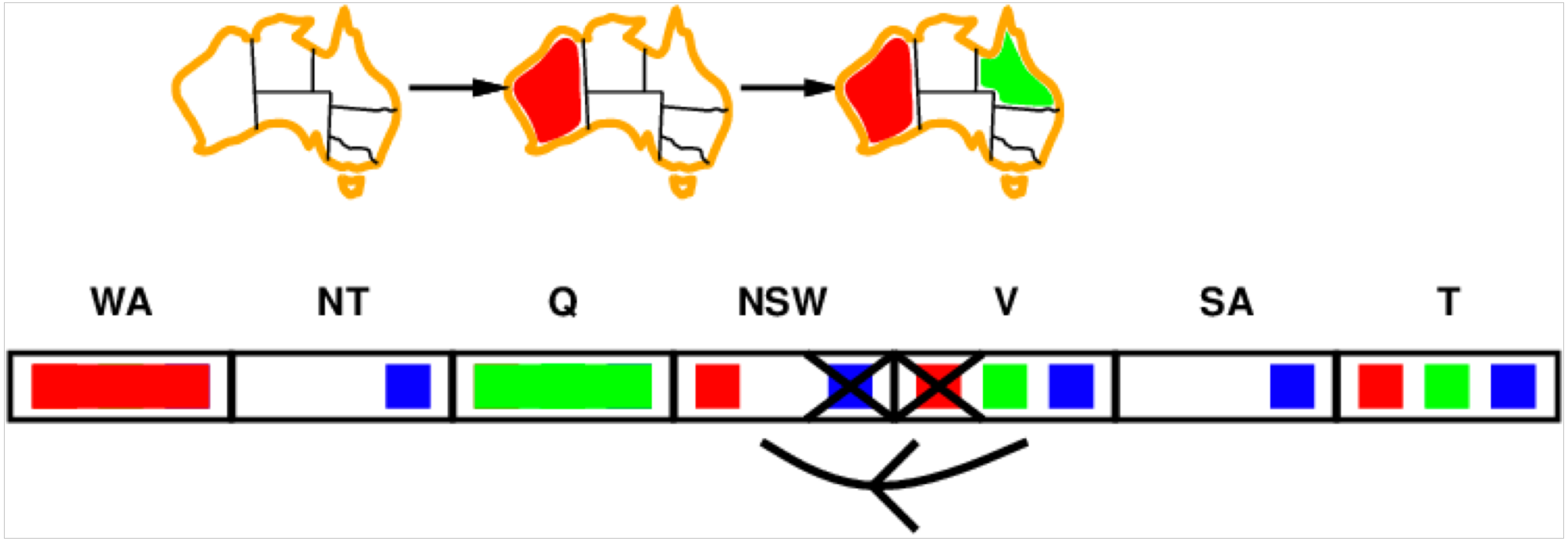
X → Y is consistent iff for every value of x of X there is some allowed value y of Y



# Arc Consistency

Simplest form of constraint propagation makes each arc consistent

X → Y is consistent iff for every value of x of X there is some allowed value y of Y




If a variable loses a value, its neighbors in the constraint graph need to be rechecked

# Maintaining Arc Consistency

If a variable loses a value, its neighbors in the constraint graph need to be rechecked.

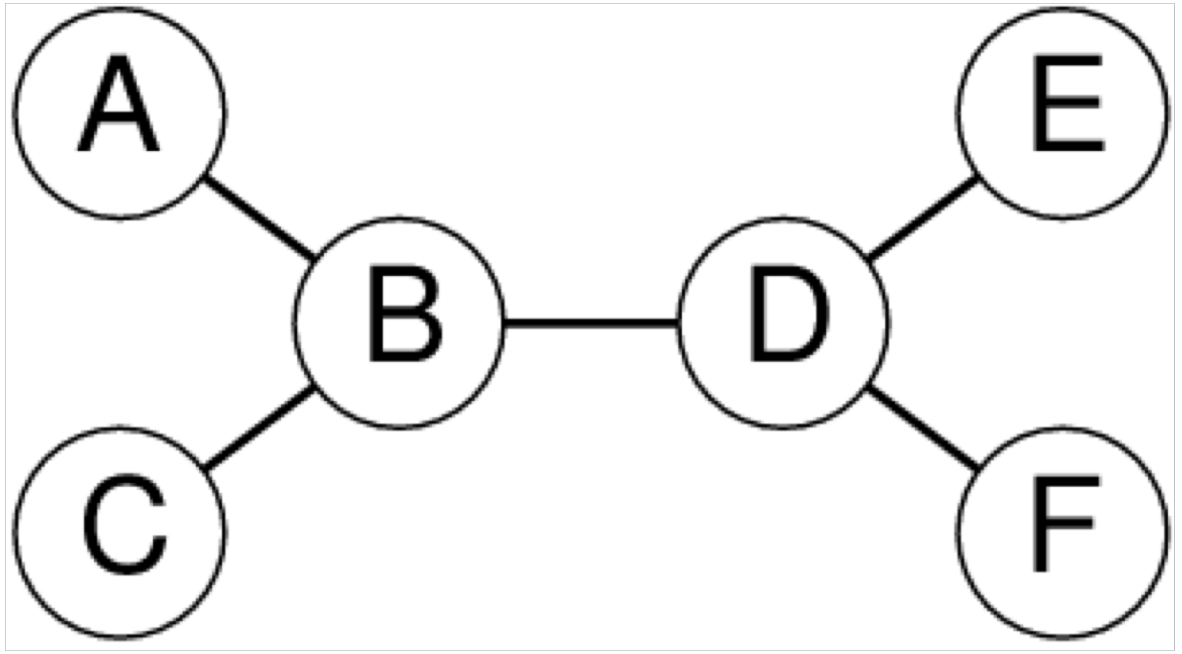
Recursively propagates constraints when changes are made to domains of variables

This recursive constraint propagation approach detects failure earlier than forward checking

Can be preprocessing or run after each assignment (INFERENCE) in the backtracking search algorithm

Algorithm: **Maintaining arc consistency (MAC), also known as AC-3**

# Tree-structured CSPs




**Theorem**: if the constraint graph has no cycles (so, it's a tree), the CSP can be solved in $O(nd^2)$ time

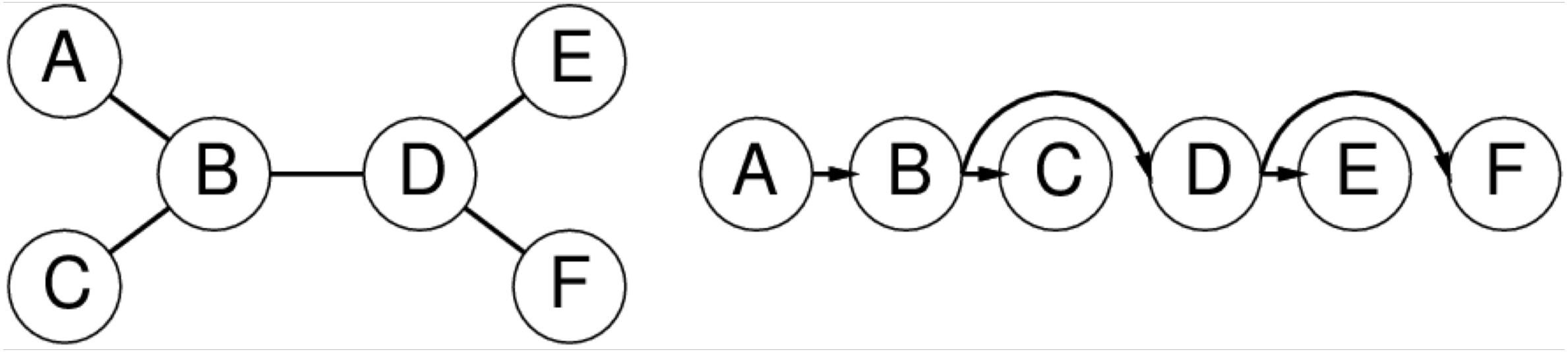
Compare to general CSPs, where worst-case is $O(d^n)$.

This property also applies to logical and probabilistic reasoning:

An important example of the relation between syntactic restrictions and the complexity of reasoning.

# Algorithm for Tree-structured CSPs

1. Choose a variable as root, order variables from root to leaves such that every node's parent precedes it in the ordering.

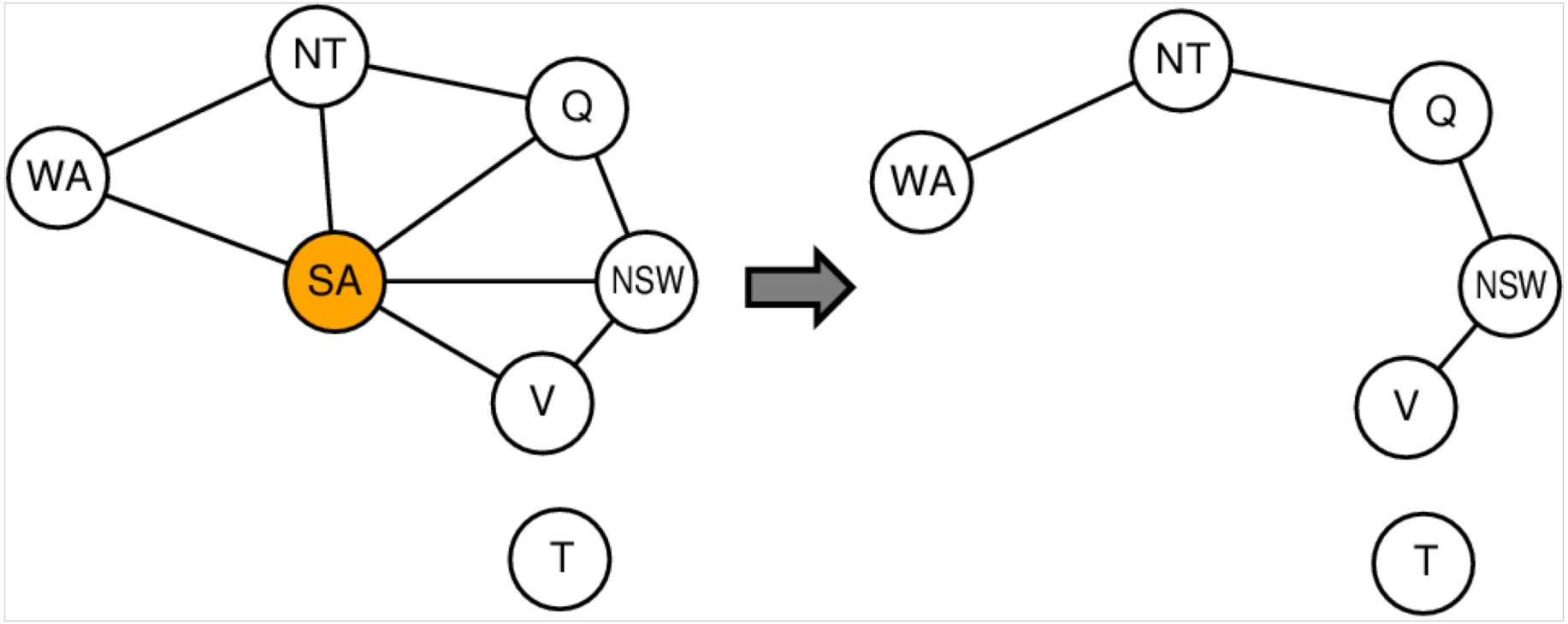



2. For j from n down to 2, apply Remove-Inconsistent(Parent($X_j$), $X_j$)

3. For j from 1 to n, assign $X_j$ consistently with Parent ($X_j$)

# Nearly Tree-structured CSPs

Conditioning: instantiate a variable, prune its neighbor's domains




**Cutset conditioning**: instantiate (in all ways) a set of variables such that the remaining constraint graph is a tree

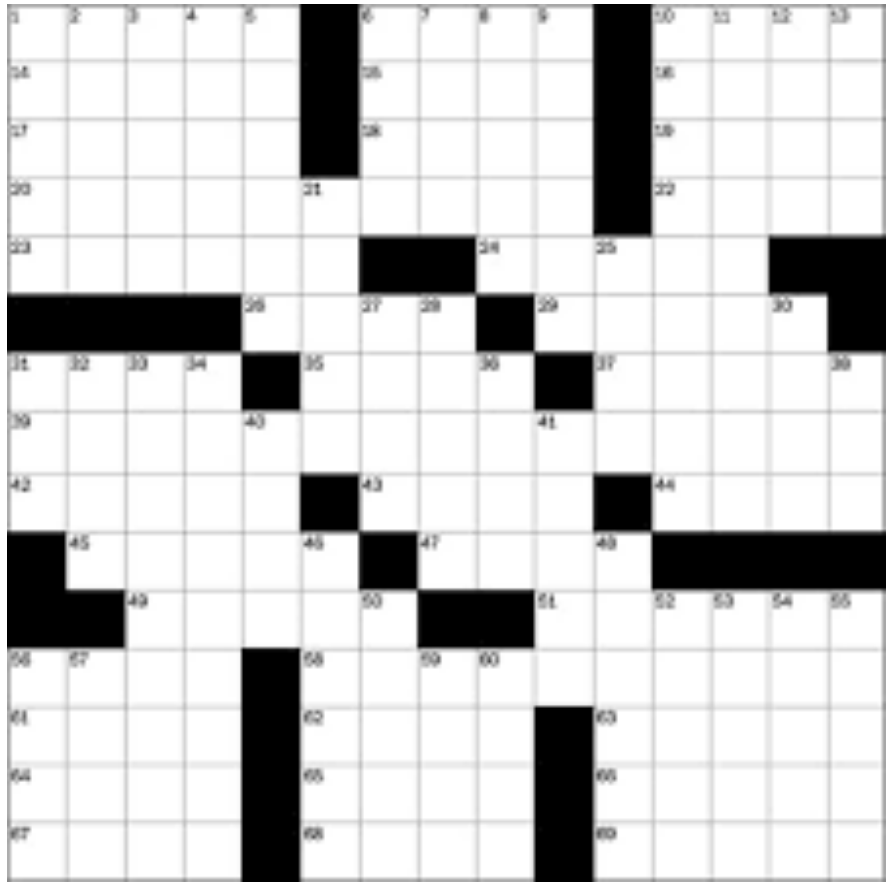
Cutset size c $\Longrightarrow$ runtime $O(d^c \cdot (n - c)d^2)$, very fast for small c

# Building crossword puzzles

Consider the problem of **constructing** (not solving) a crossword puzzle. You are given a grid (like the one to the right), and a dictionary of allowable words.

How would you approach this problem as a search problem? (like what we did w/uninformed search)

As a CSP, what would you constraint. The letters or the words, and why.



# In class work

6.2 Consider the problem of placing *k* knights on a *n* x *n* chessboard such that no two nights are attacking each other, and that $k \leq n^2$.

a) Choose a CSP formulation. What are the variables?

b) What are the possible values of each variable?

c) What sets of variables are constrained and how?

# In class work

6.6. Show how a single ternary constraint such as: **A + B = C** can be turned into 3 binary constraints using an auxiliary variable. You may assume a finite domain for all variables. Hint: Consider a new variable that takes on values that are pairs of other values, and consider constraints such as “X is the first element of the pair Y”.