Constraint Satisfaction Problems

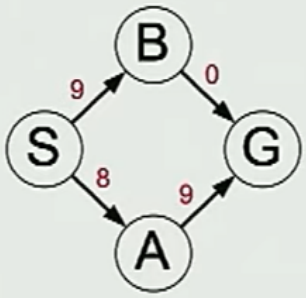
Overview

State based model review

State-based models

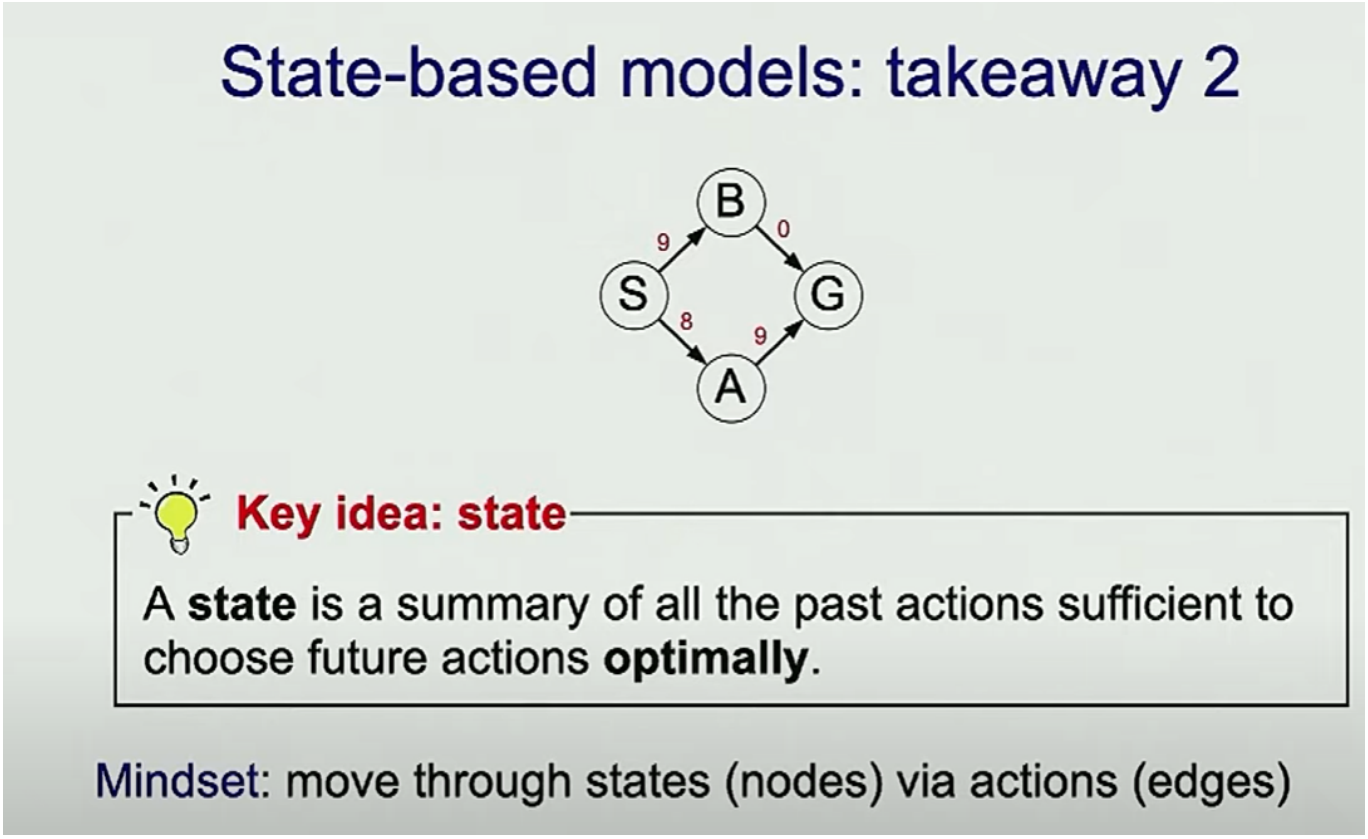
[Modeling]		
Framework	search problems	MDPs/games
Objective	minimum cost paths	maximum value policies
[Inference]		
Tree-based	backtracking	minimax/expectimax
Graph-based	DP, UCS, A*	value/policy iteration
[Learning]		
Methods	structured Perceptron	Q-learning, TD learning

State-based models: takeaway 1



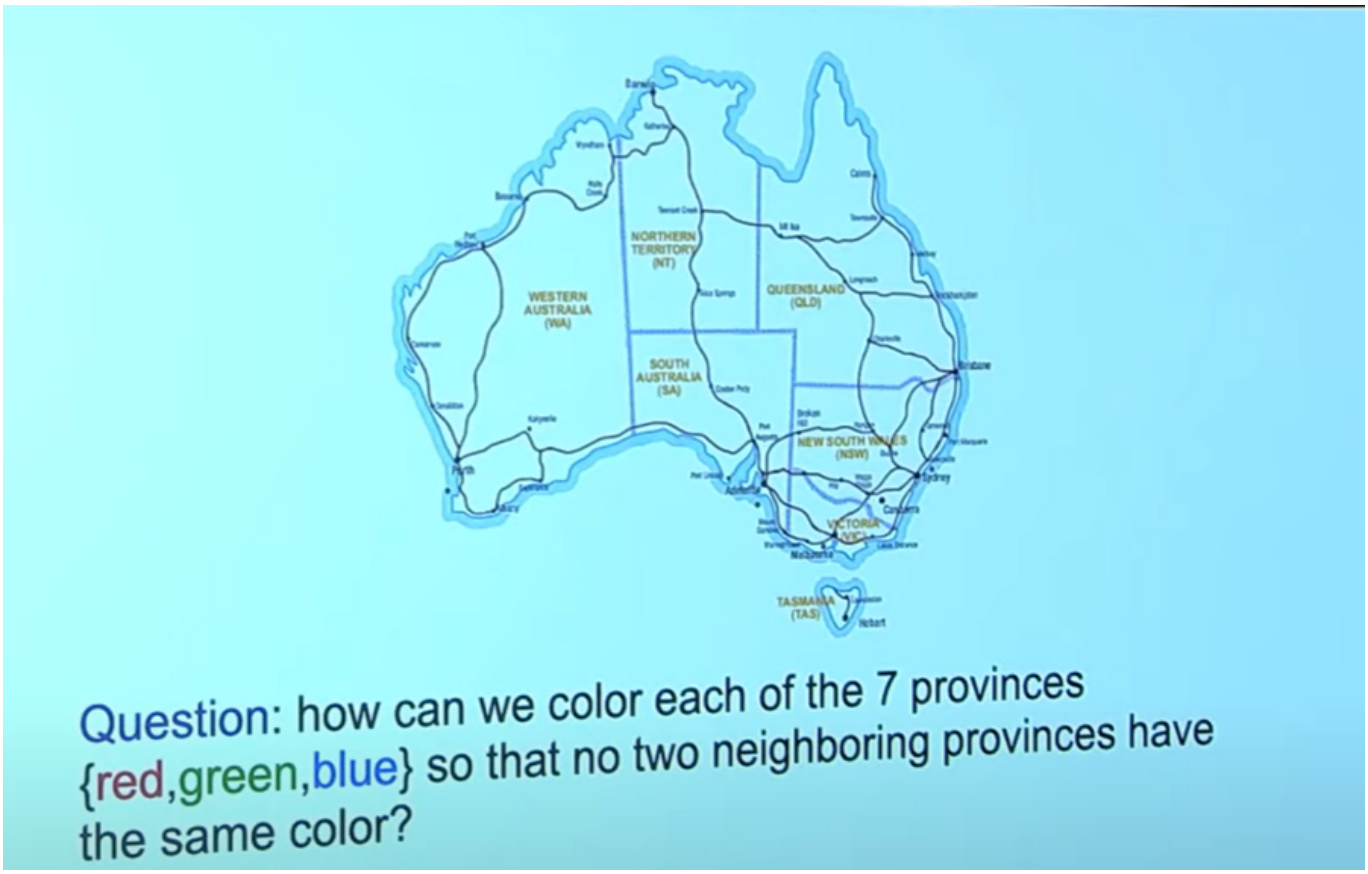
Key idea: specify locally, optimize globally

Modeling: specifies local interactions
Inference: find globally optimal solutions



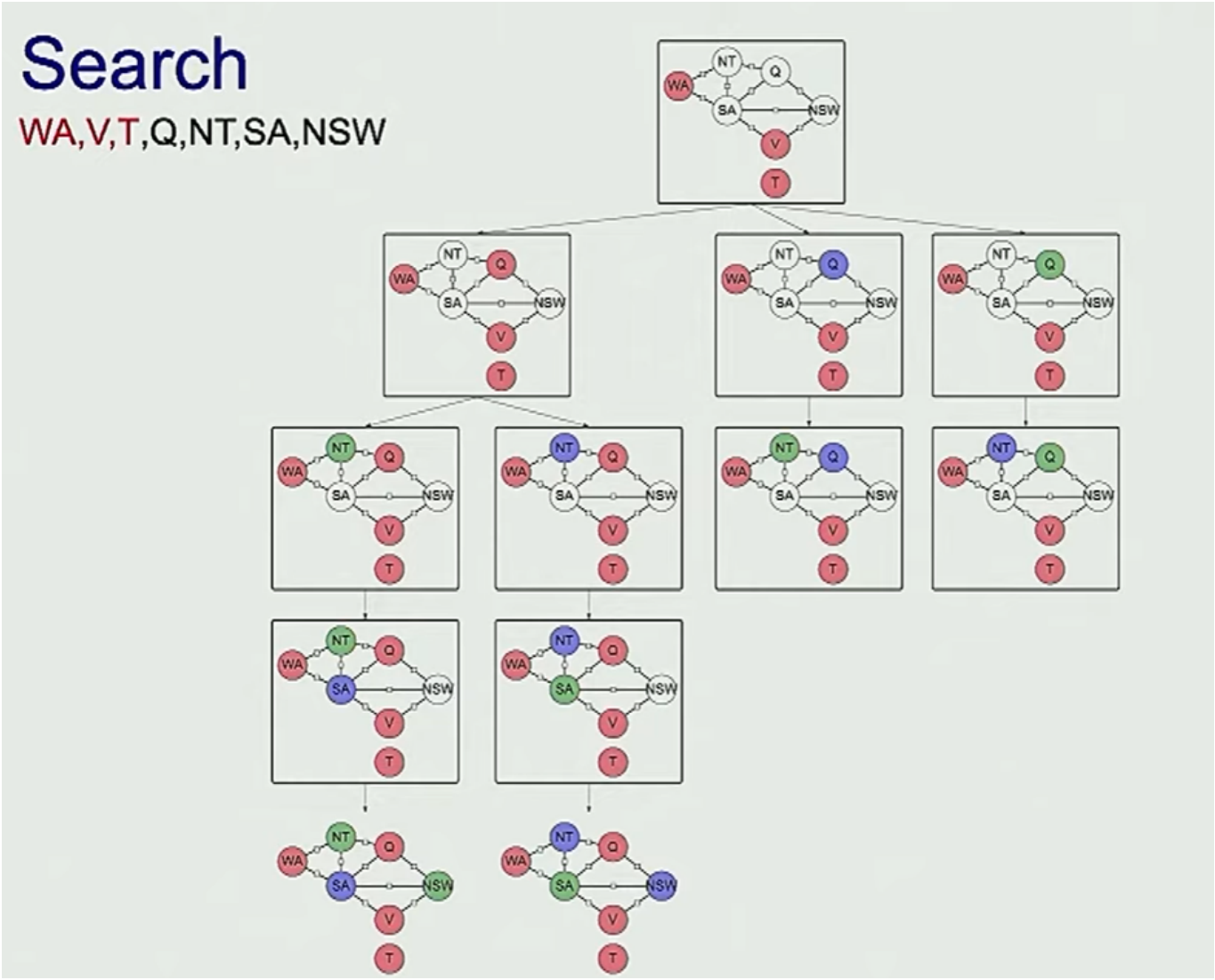
For the example we had, there are some constraints and you don't have to worry about the order(it doesn't matter)

The Color State Example

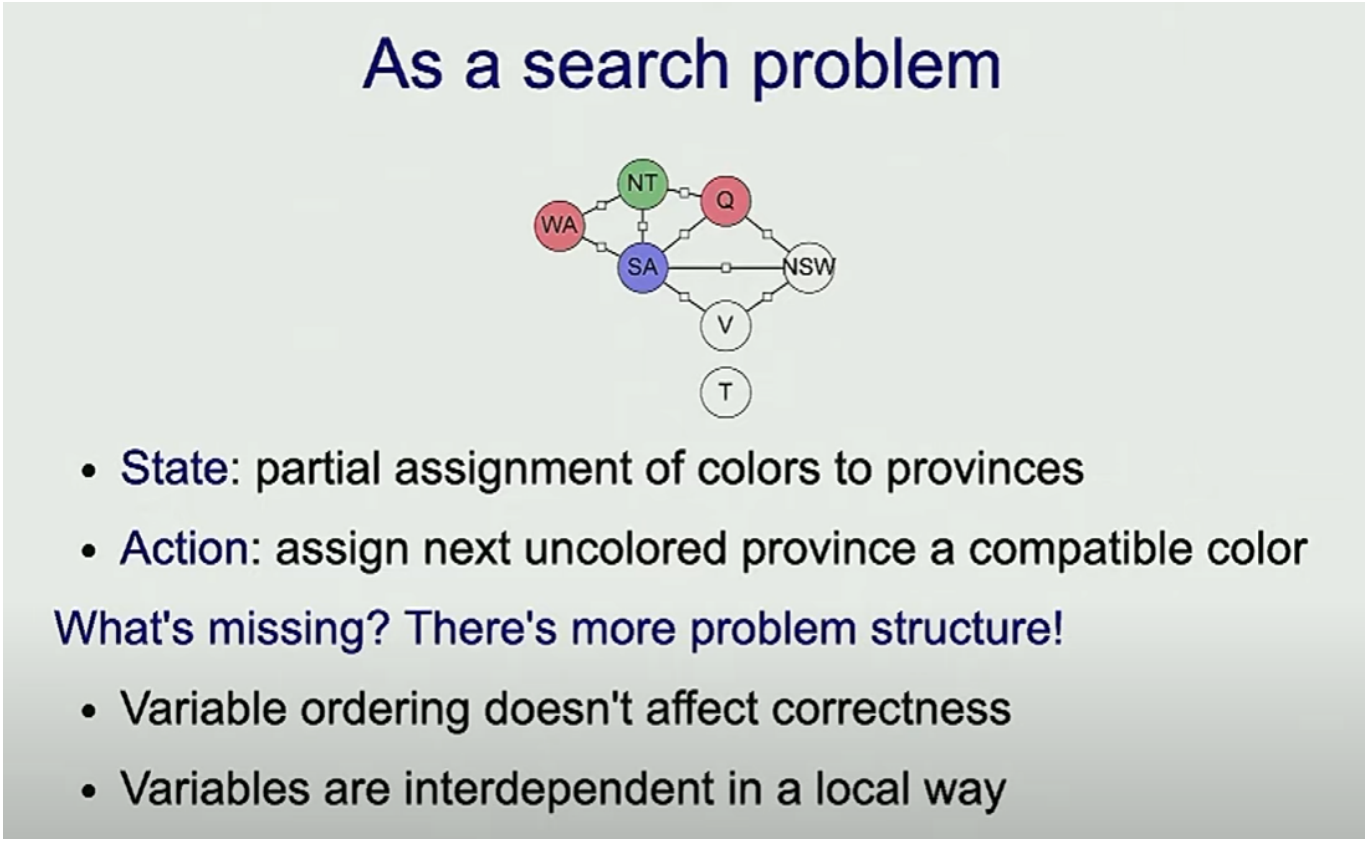


The order here doesn't matter

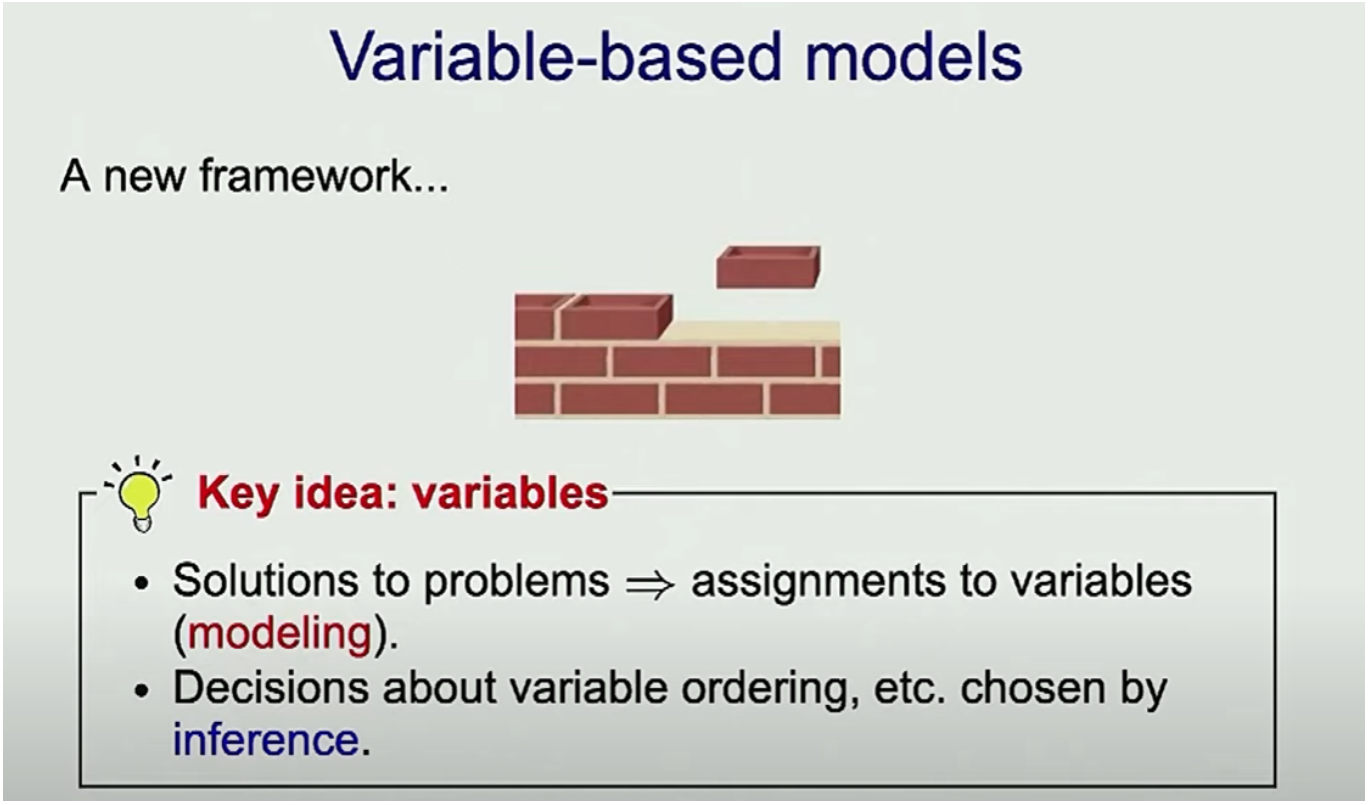

Map coloring
Western Australia
Northern Territory
Queensland
South Australia
New South Wales
Victoria
Tasmania
(one possible solution)



We can definitily consider this problem as a search problem, with different states And we give it an initial state Let's say somehow we start with WA, V, T as red, and try to search for solutions for the whole problem



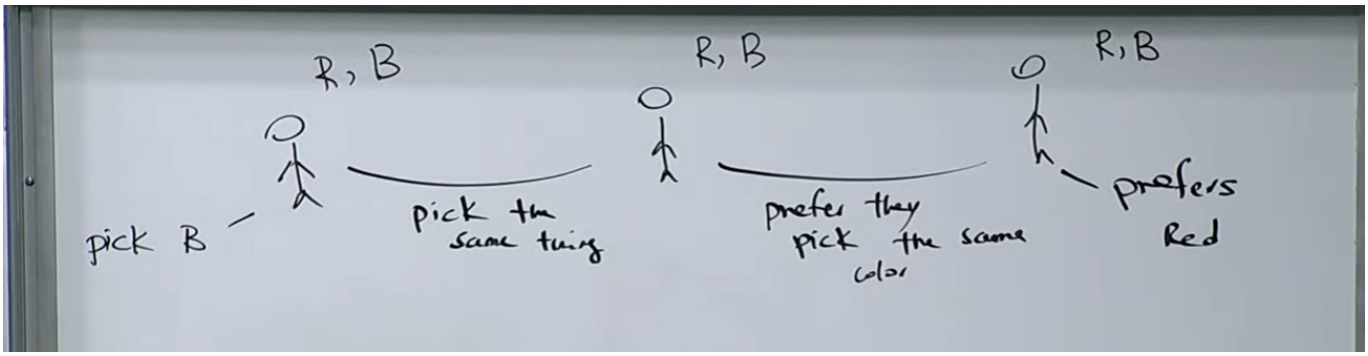
So state based model seems too complicated for this, we can try to make things easier



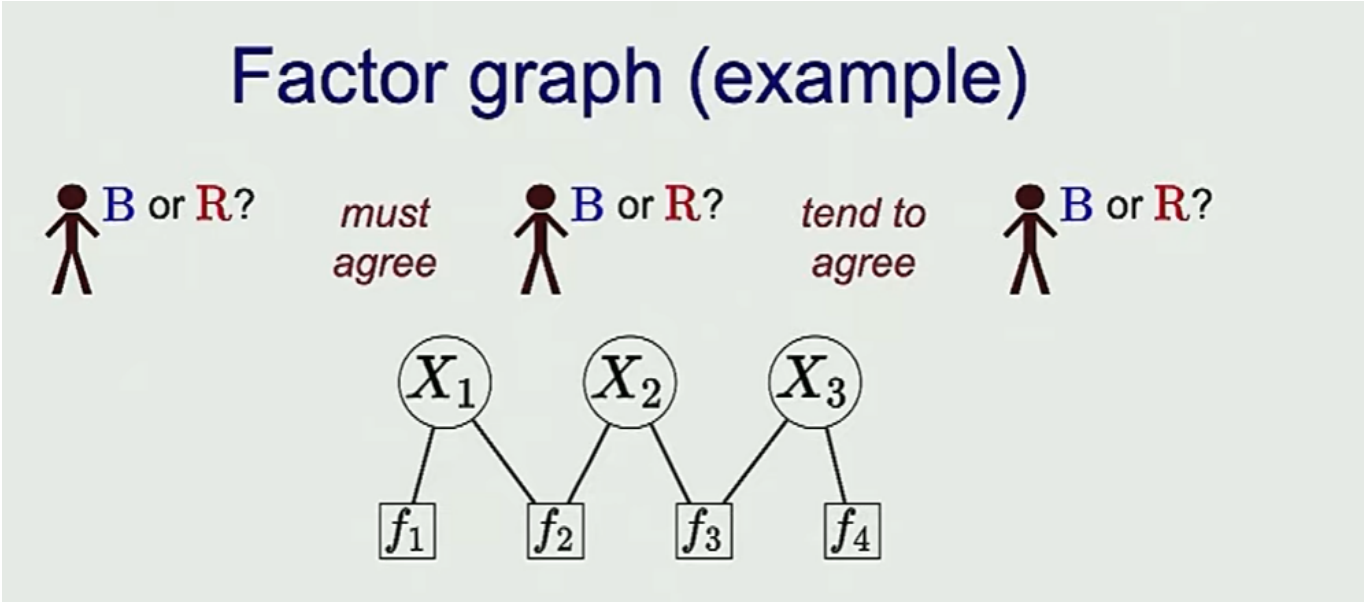
A higher level of abstraction

Factor graphs

So I have three people, each of them are gonna choose a color either red or blue, and we have a bunch of constraints



Factor graphs are gonna have some number of variables

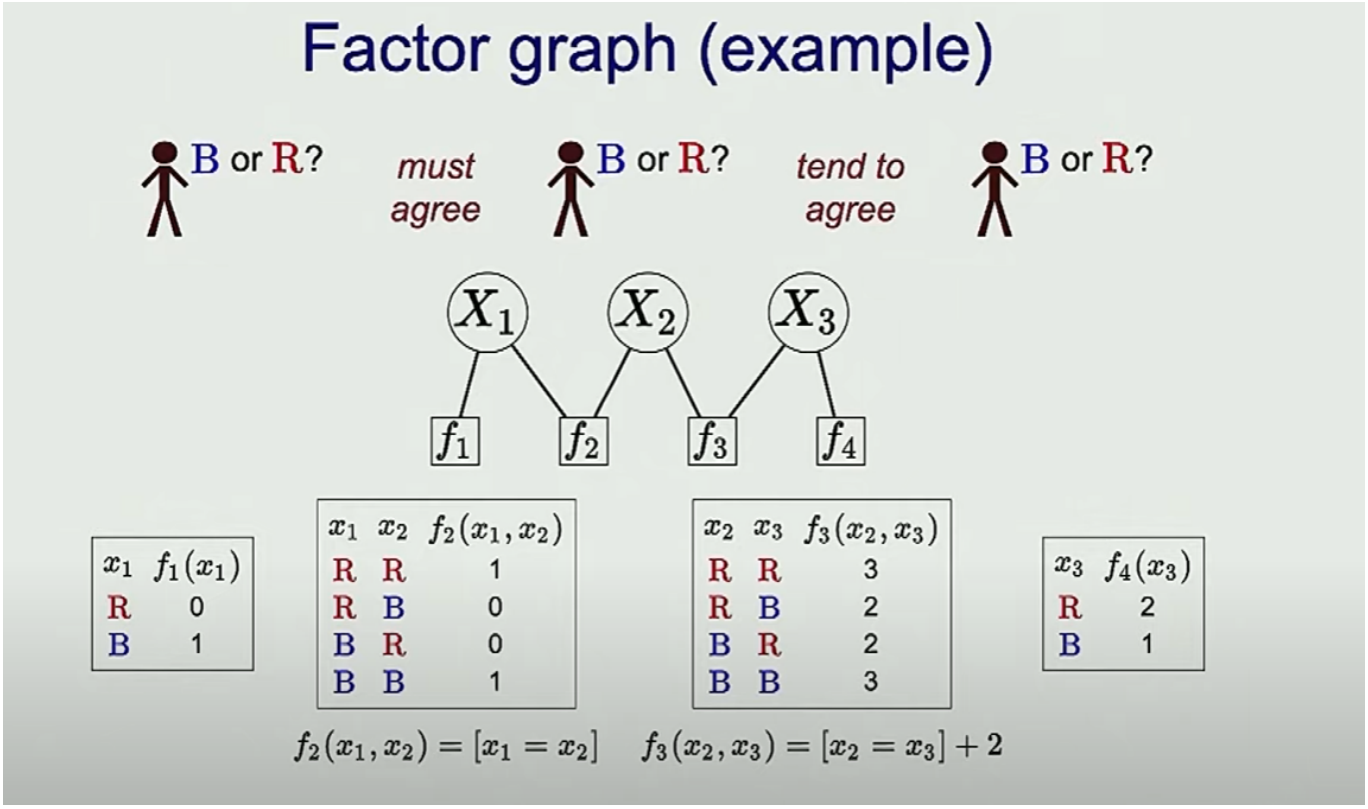


Factor of X1 is a function of X1, tells me how happy I would be if X1 takes value red or blue

Model the problem

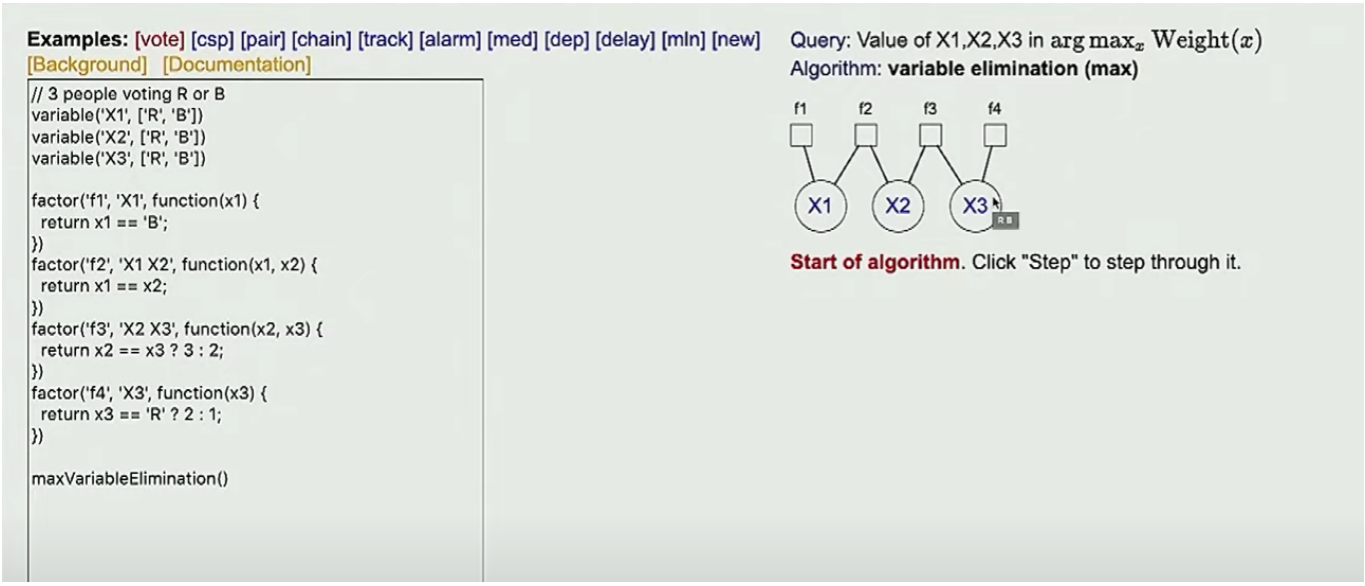
Factor graphs

- Variables: X1, X2, X3
  - For each variable, it's gonna belong to some domin {R, B}
  - Assignment: Assign values to variables
- Factors: reflacts the constraints that we need to satisfy
  - Scope of factor: set of variables a factor depend on
  - Arity of a factor: number of variables in the scope

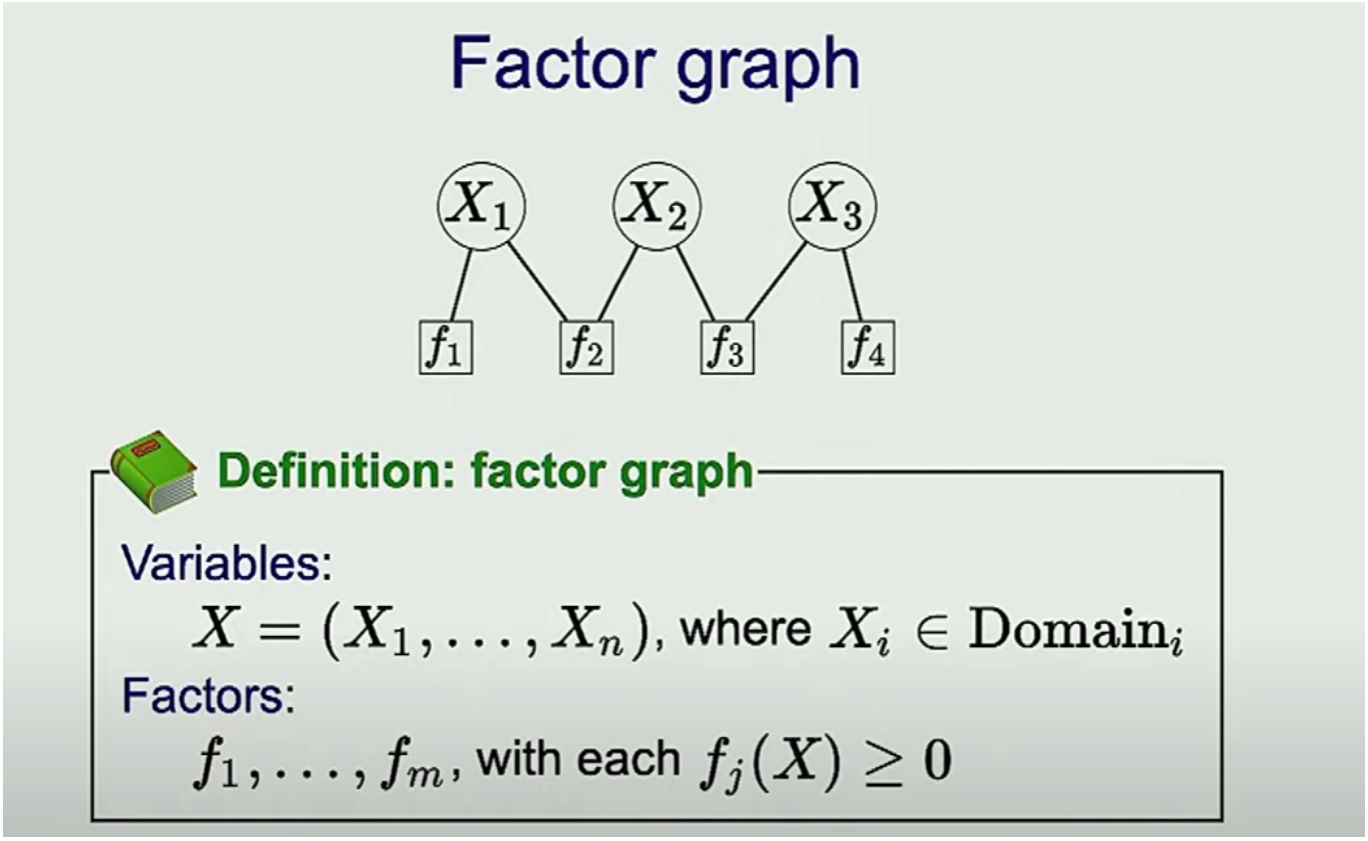


`[X1 = X2]`: An indicator function returns 1 if the statement is true, and 0 otherwise In the factor graph, 0 means I absolutly do not wanna choose this For `prefer`, you can add some values after the indicator function

Q: does the factor value matter? A: Yes, but in this class we will not foucus on this

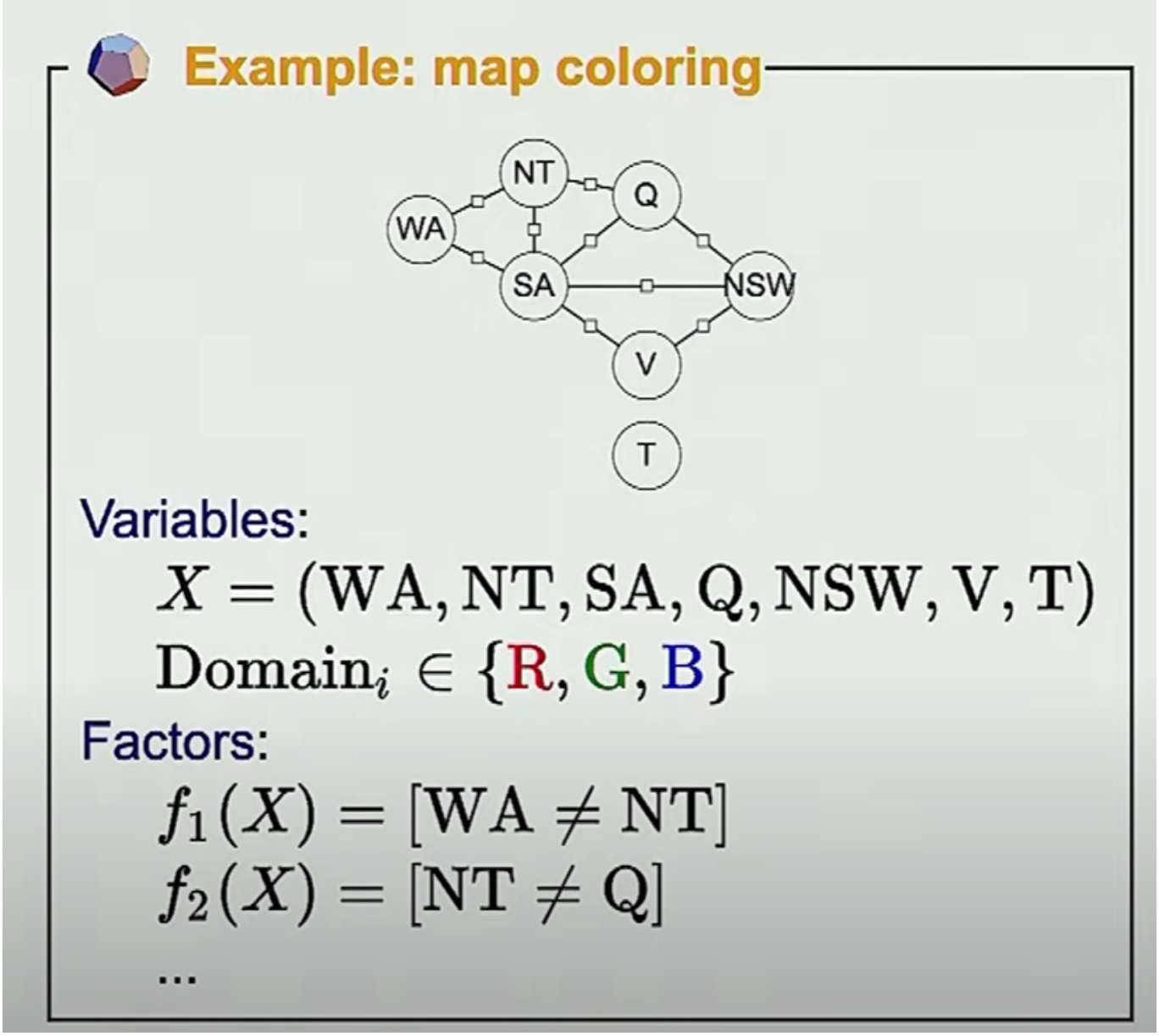


Formal Definition

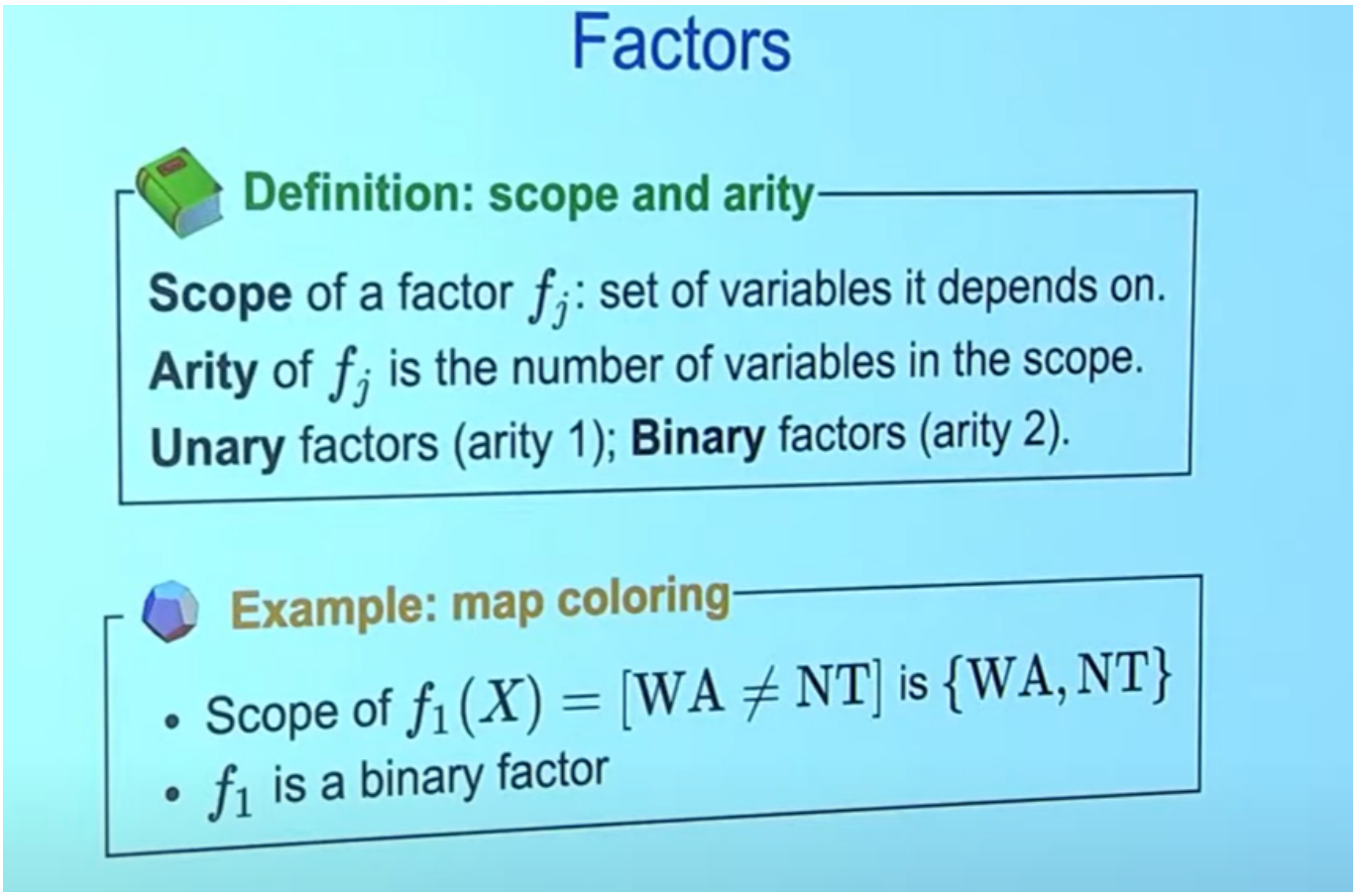


So more formally, A factor graph is

- A set of variables `X1` to `Xn`, and each of these variables lies on some domain (in this case `R` or `B`)
- And factors from `f1` to `fm`, which are functions over X that is going to be >= 0 It tells us what are the things that we really want



So the factors here are just telling don't give same color to two neighbours



Assignment weights

So factor tells us how happe we are for the single assinment The weight tells us how good the whole assignment is

the weight can just be the product of all the factors

Assignment weights (example)

x_1	$f_1(x_1)$
R	0
B	1

x_1	x_2	$f_2(x_1, x_2)$
R	R	1
R	B	0
B	R	0
B	B	1

x_2	x_3	$f_3(x_2, x_3)$
R	R	3
R	B	2
B	R	2
B	B	3

x_3	$f_4(x_3)$
R	2
B	1

x_1	x_2	x_3	Weight
R	R	R	$0 \cdot 1 \cdot 3 \cdot 2 = 0$
R	R	B	$0 \cdot 1 \cdot 2 \cdot 1 = 0$
R	B	R	$0 \cdot 0 \cdot 2 \cdot 2 = 0$
R	B	B	$0 \cdot 0 \cdot 3 \cdot 1 = 0$
B	R	R	$1 \cdot 0 \cdot 3 \cdot 2 = 0$
B	R	B	$1 \cdot 0 \cdot 2 \cdot 1 = 0$
B	B	R	$1 \cdot 1 \cdot 2 \cdot 2 = 4$
B	B	B	$1 \cdot 1 \cdot 3 \cdot 1 = 3$

Assignment weights

Definition: assignment weight

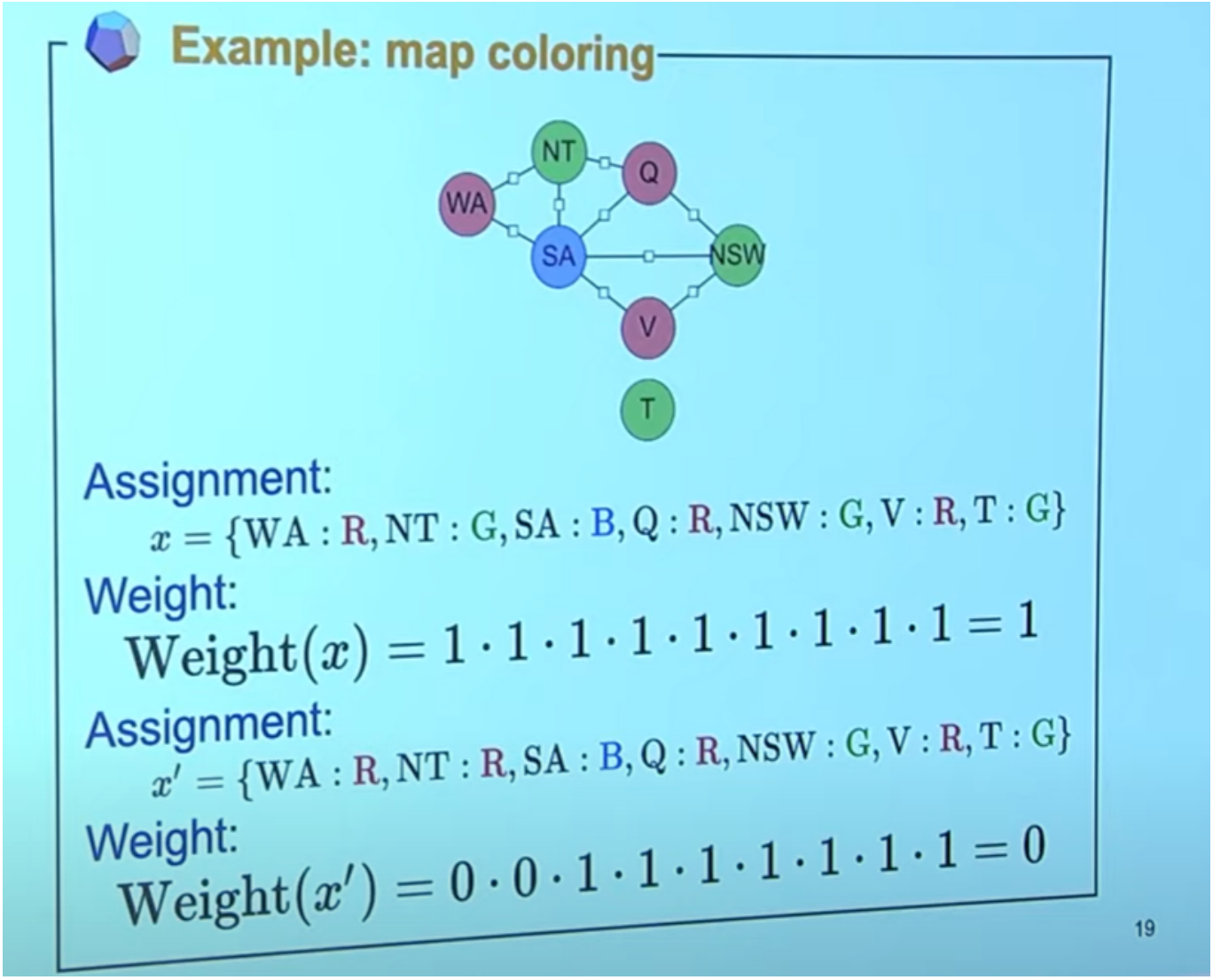
Each **assignment** $x = (x_1, \ldots, x_n)$ has a **weight**:

$$\text{Weight}(x) = \prod_{j=1}^{m} f_j(x)$$

Objective: find the maximum weight assignment

$$\arg\max_x \text{Weight}(x)$$

Objective: Find the x that maximize the weight



In this example, the indicator function is just 0 and 1 If we find the solution then the weight is gonna be 1 If not then it's gonna be 0

Constarint satisfaction problems

CSP is just factor graph where all factor function value is either 0 or 1 When we find an assignment where weight = 1, this means we find an assignment which satisfies all the constraints `iff`: If and only if

Constraint satisfaction problems

Definition: constraint satisfaction problem (CSP)

A CSP is a factor graph where all factors are **constraints**:

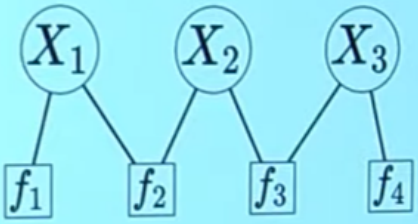
$$f_j(x) \in \{0, 1\} \text{ for all } j = 1, \ldots, m$$

The constraint is satisfied iff $f_j(x) = 1$.

Definition: consistent assignments

An assignment x is **consistent** iff $\text{Weight}(x) = 1$ (i.e., **all** constraints are satisfied).

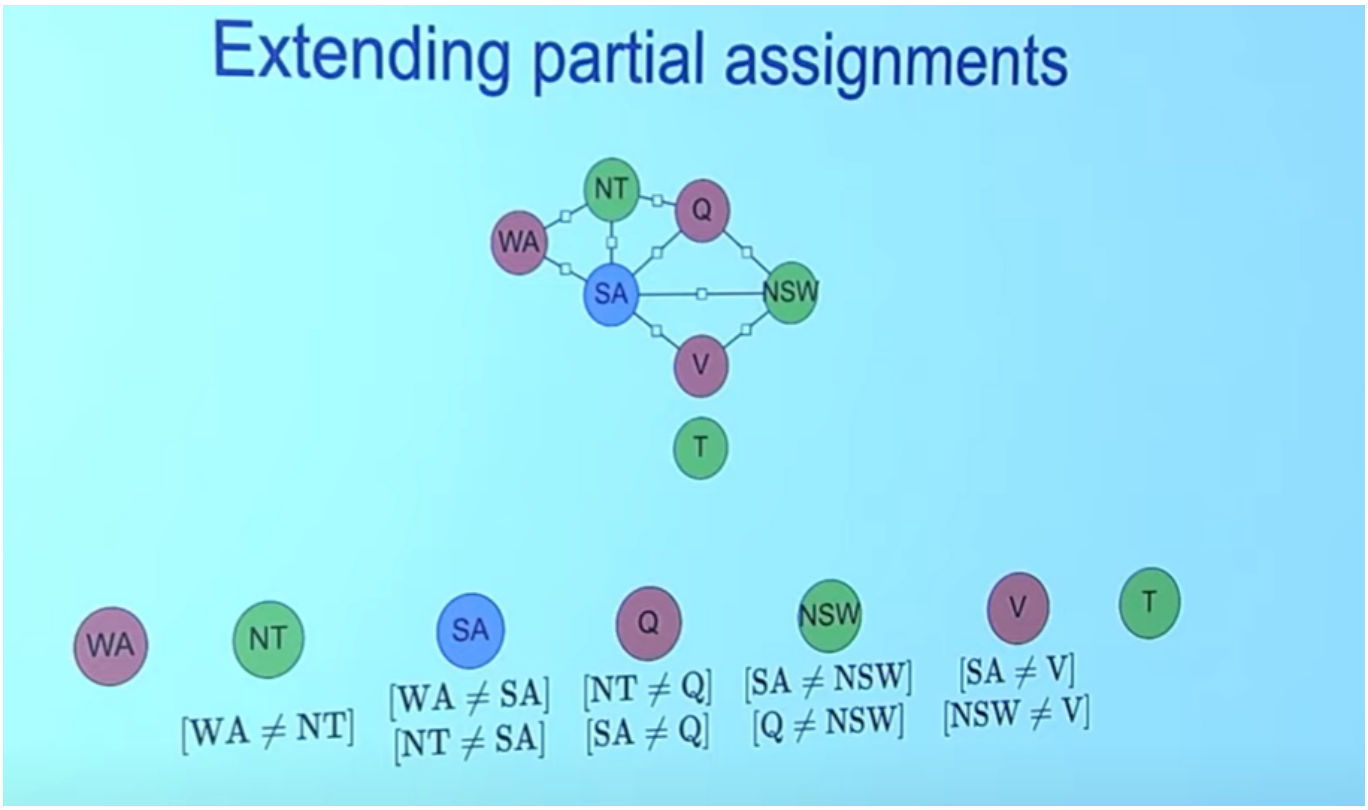
Summary so far



Factor graph (general)	**CSP** (all or nothing)
variables	variables
factors	constraints
assignment weight	consistent or inconsistent

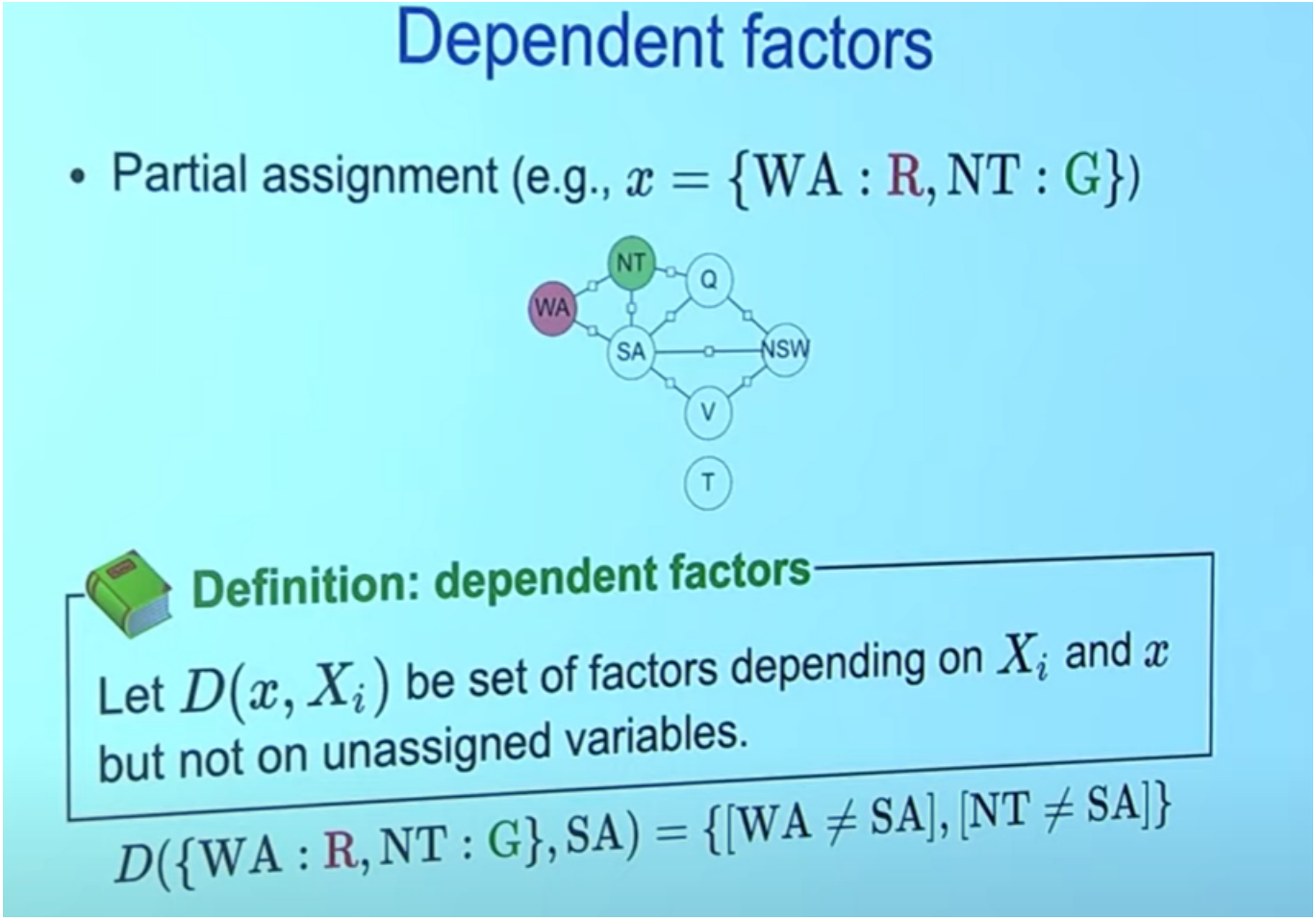
Dynamic ordering

Let go and talk about how we gonna solve this How to find an assignment to achieve consistency



What we might solve this in a human way: We pick some initial value and go with some order, try to find assignments satisfy the constraints

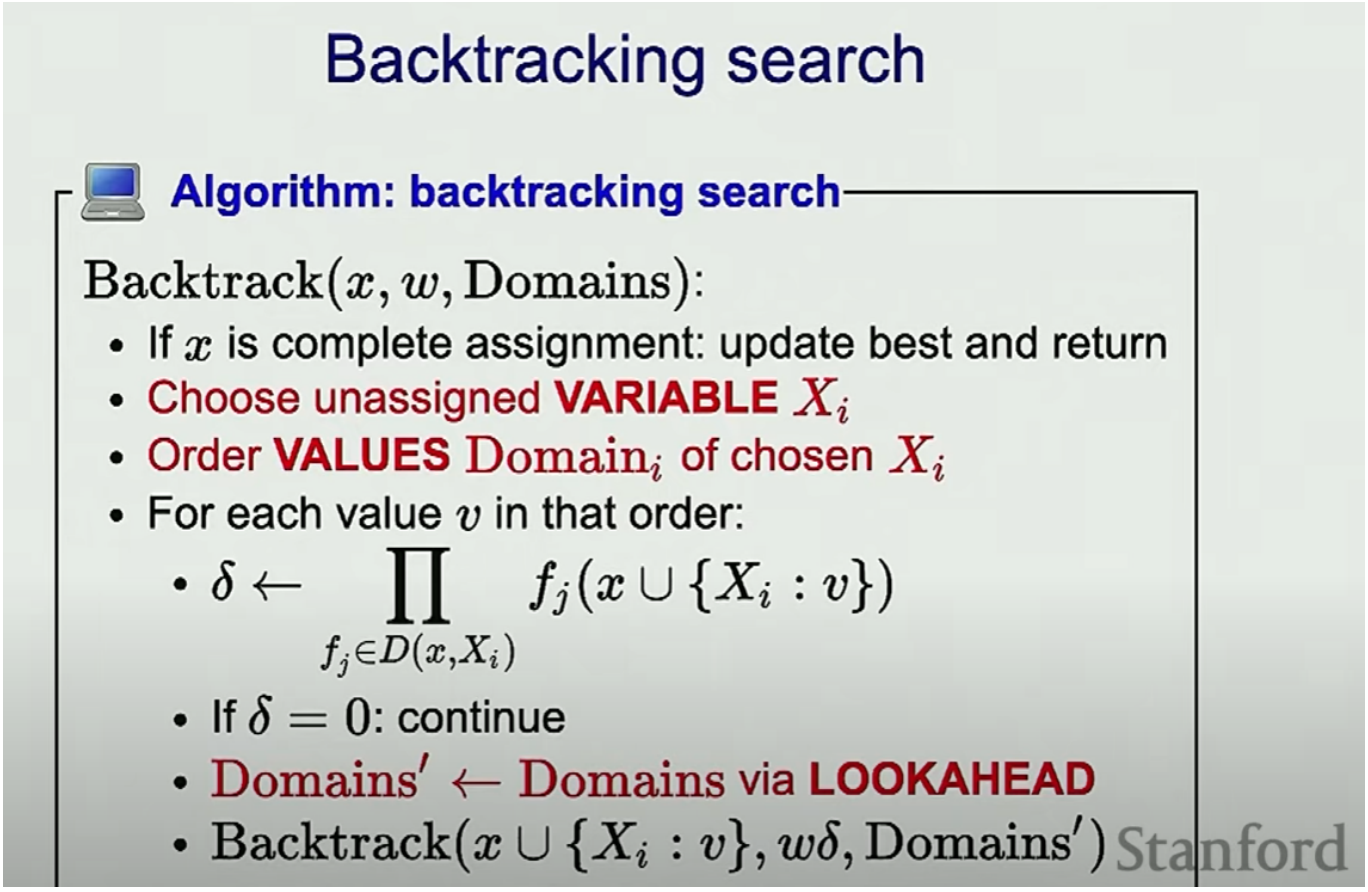
Dependent Factors



D(x, xi) `x` is the assignment `xi` is the new variable I am picking And it's gonna return a set of factors that depends on `x` and `xi`

The Dependent Factor allow us to think about the next thing we should be worried about

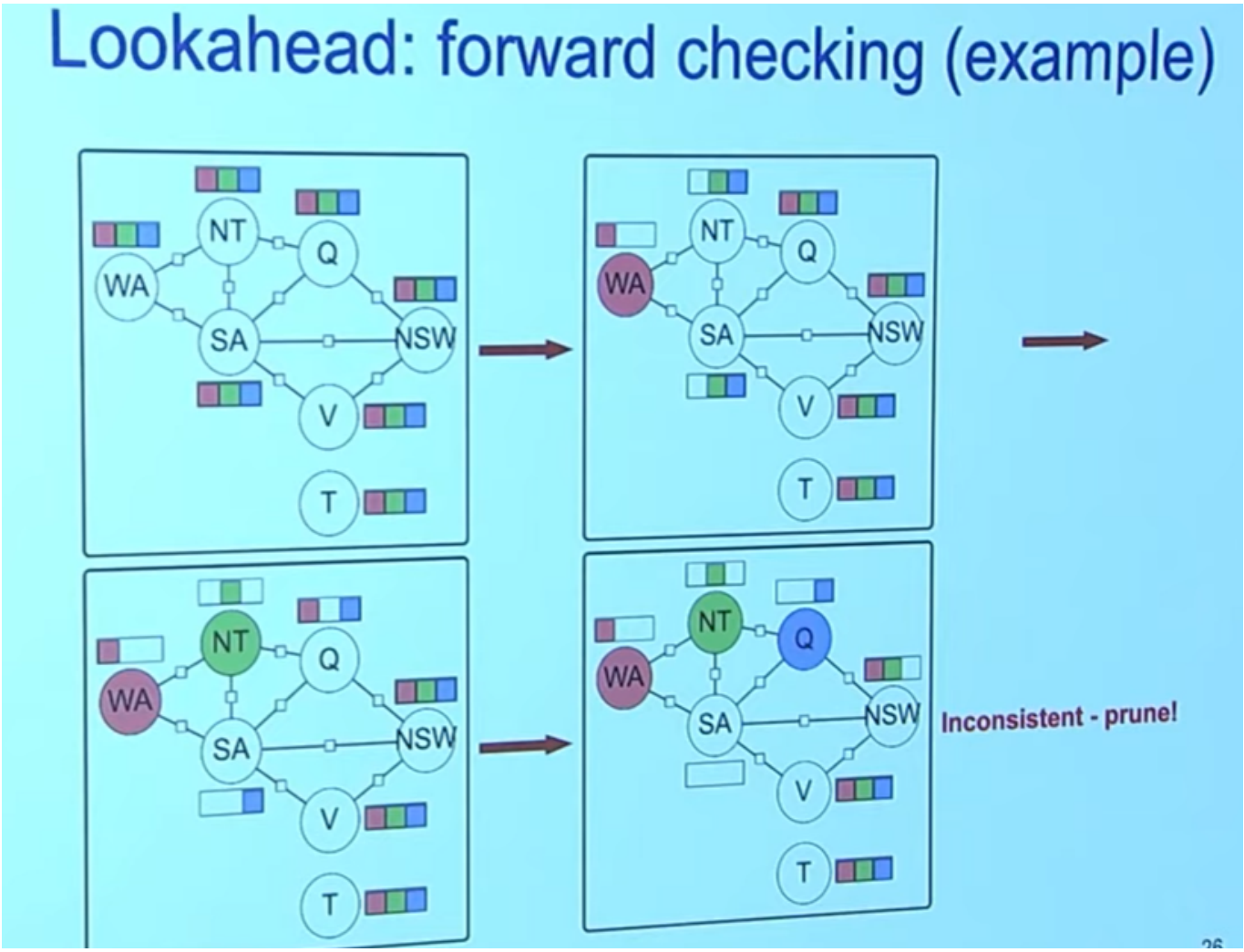
Backtracking Search



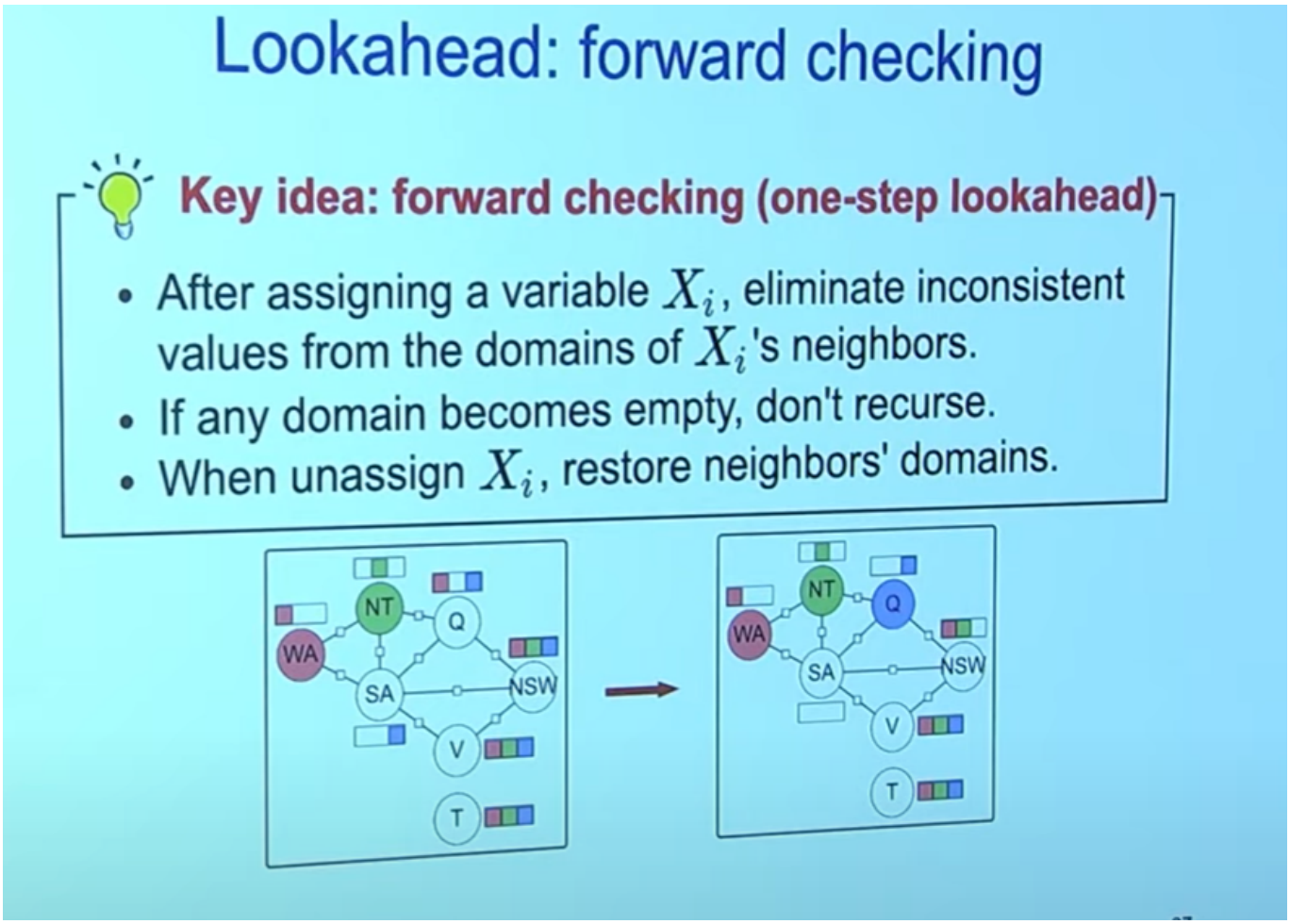
Backtrack(x, w, domains) x: Partial assignment we have w: Weight we have so far domains: domain of the variables that I have so far

- If `X` is complete then we are happy
  - If X is already a complete assignment, then we'll update the best thing we have and return (base case)
- Choose unassigned variable `Xi`
  - Figure put what is the next variable I am gonna pick
- Order values in Domain i of chosen `Xi`
  - We gonna pick some value for the variable, from the domain
  - Maybe the only two color availible for variable `Xi` is R and B, and we gonna order R and B with some heuristic
- And for each value in this order we decided, we gonna update the weight `w`
  - `delta - add on weighe` is gonna be the product of all factors of
    - partial assignments whatever you have decided so far (`X`)
    - Union whatever value ur looking at for this new variable `Xi` to pick (color for example)
  - `Fj` are the dependent factors of (partial assignment, new variable), meaning we are lookint current partial assignment and xi
  - If `delta` == 0, continue, meaning this assignment does not work(it made everything 0), and we should try other things
  - If `delta` != 0, meaning his assignment works and we can update our domain, that's the thing that can save us time
- Backtrack()
  - X: we extended it by value v
  - w: whatever weight we started * delta
  - domain: updated domain

Update domain - forward checking approach

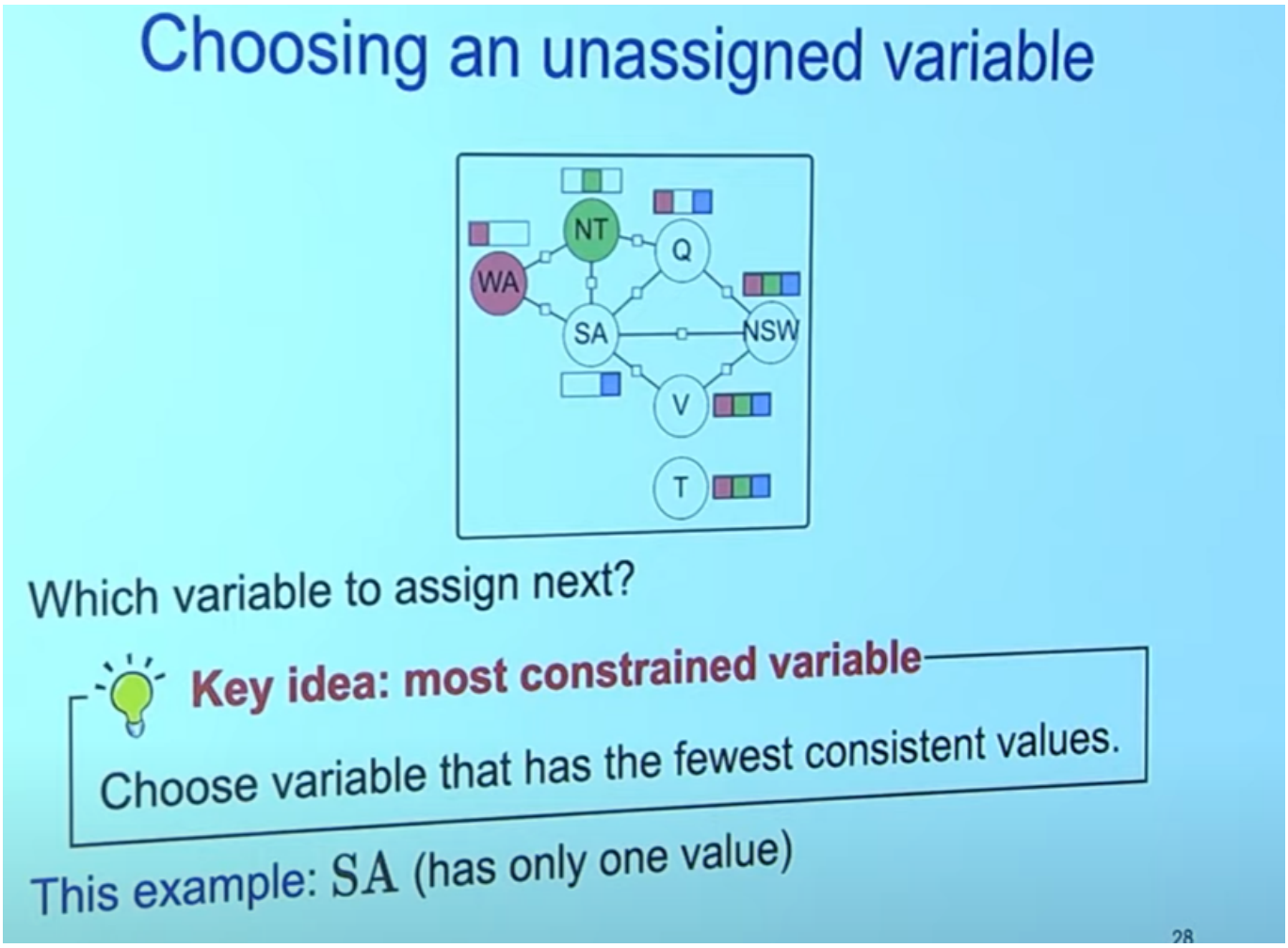


One example is: let's say we decided WA to be red, we can just look at its neibours and see if we can update domain for them(and we found they cannot use red)



Heuristic

Choose which unassigned variable `Xi`

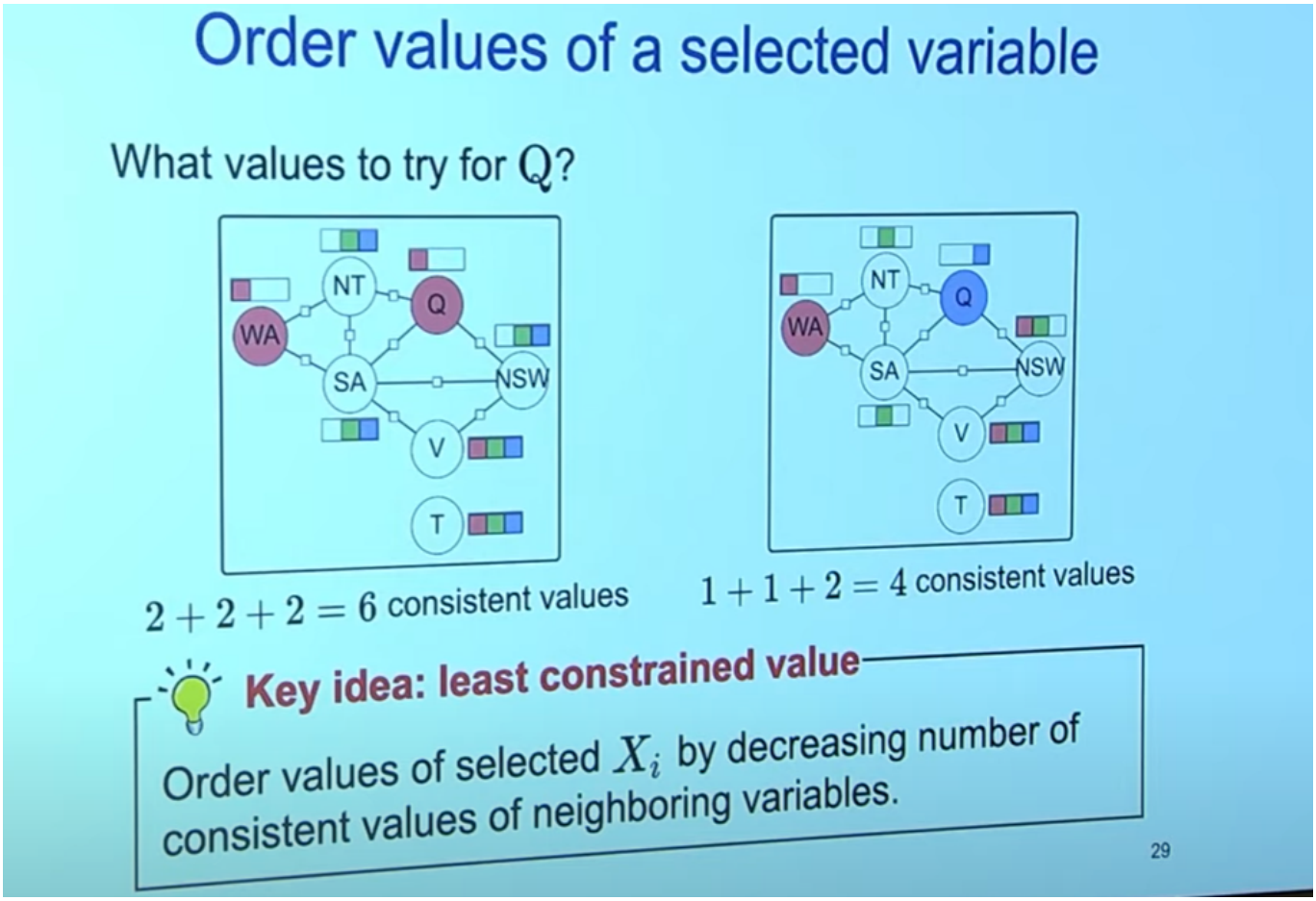


Pick the most constraint variable Why?

- Less options
- If it is gonna fail, let fail early

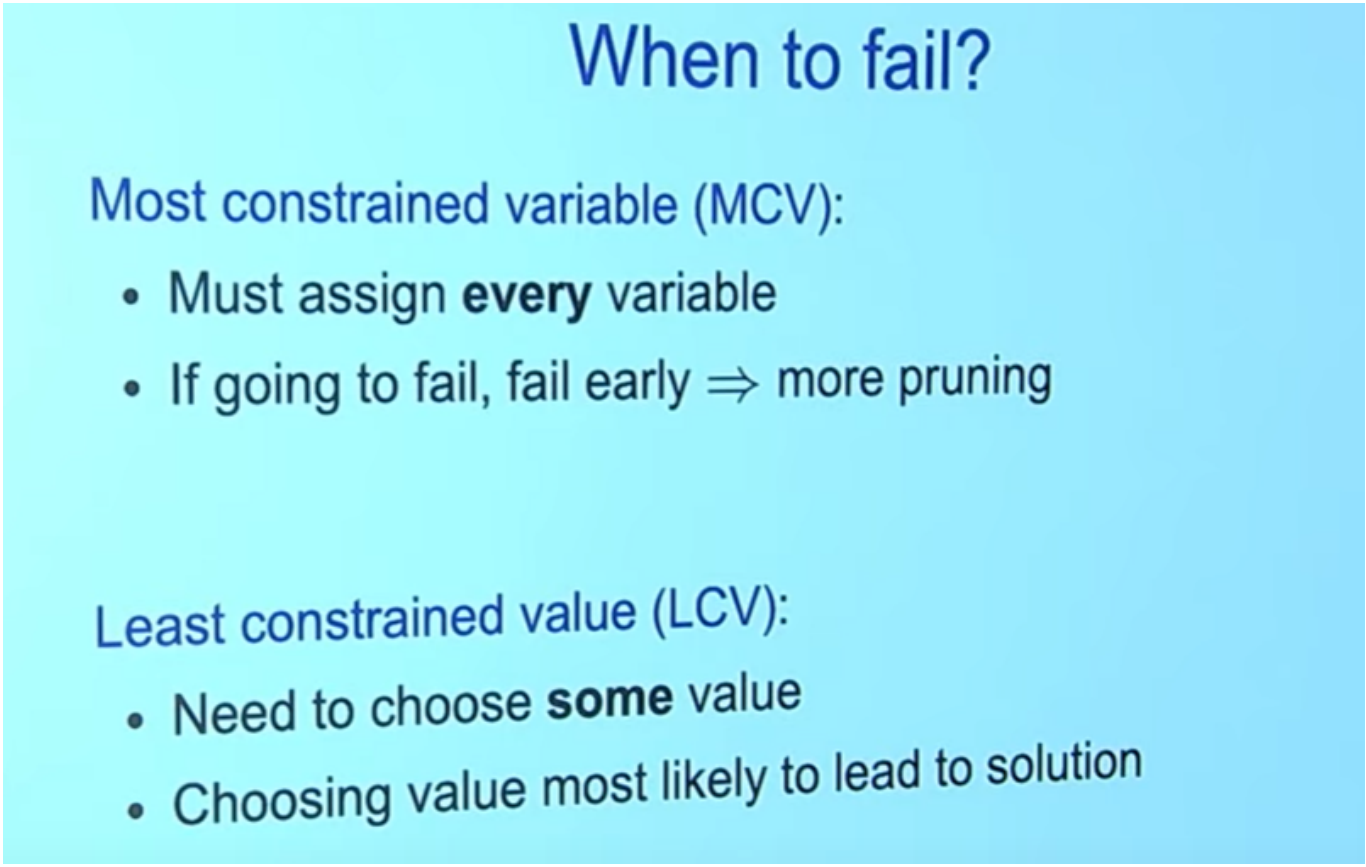
What Value to choose

Pick the least contraint value to leave options for other variables around you



We are gonna pick the one that leaves the most options possible So we gonna order the value of selected variable `Xi` by decreasing number of consistent values of neighboring variables

This will only work if we are doing forward checking



Arc Consistency

For forward checking, we are just checking neighbor node and update domain For Arc Consistency, it goes through the whole CSP and tries to update the furthur nodes ahead of us The main idea is to reduce the branching factors

Arc consistency

Idea: eliminate values from domains $\Rightarrow$ reduce branching

Example: numbers

Before enforcing arc consistency on X_i:

$X_i \in \text{Domain}_i = \{1, 2, 3, 4, 5\}$

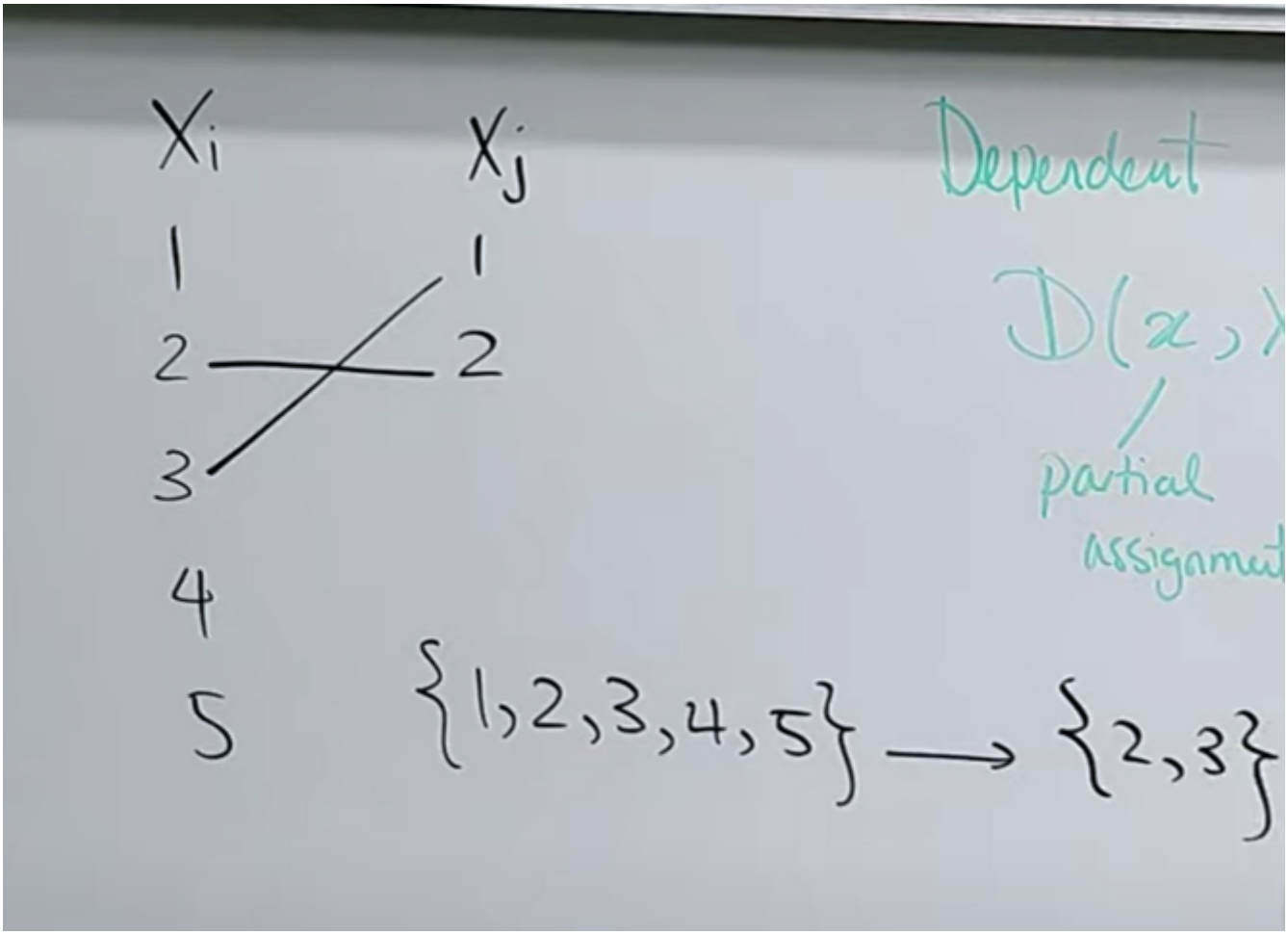
$X_j \in \text{Domain}_j = \{1, 2\}$

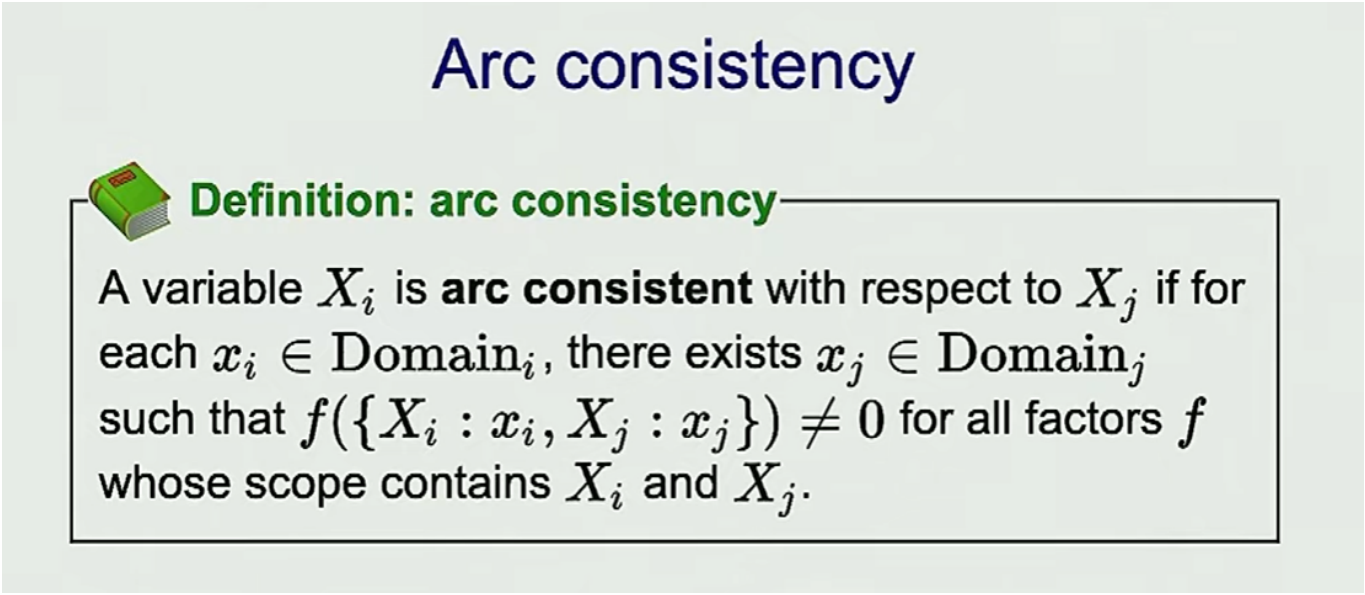
$f_1(X) = [X_i + X_j = 4]$

After enforcing arc consistency on X_i:

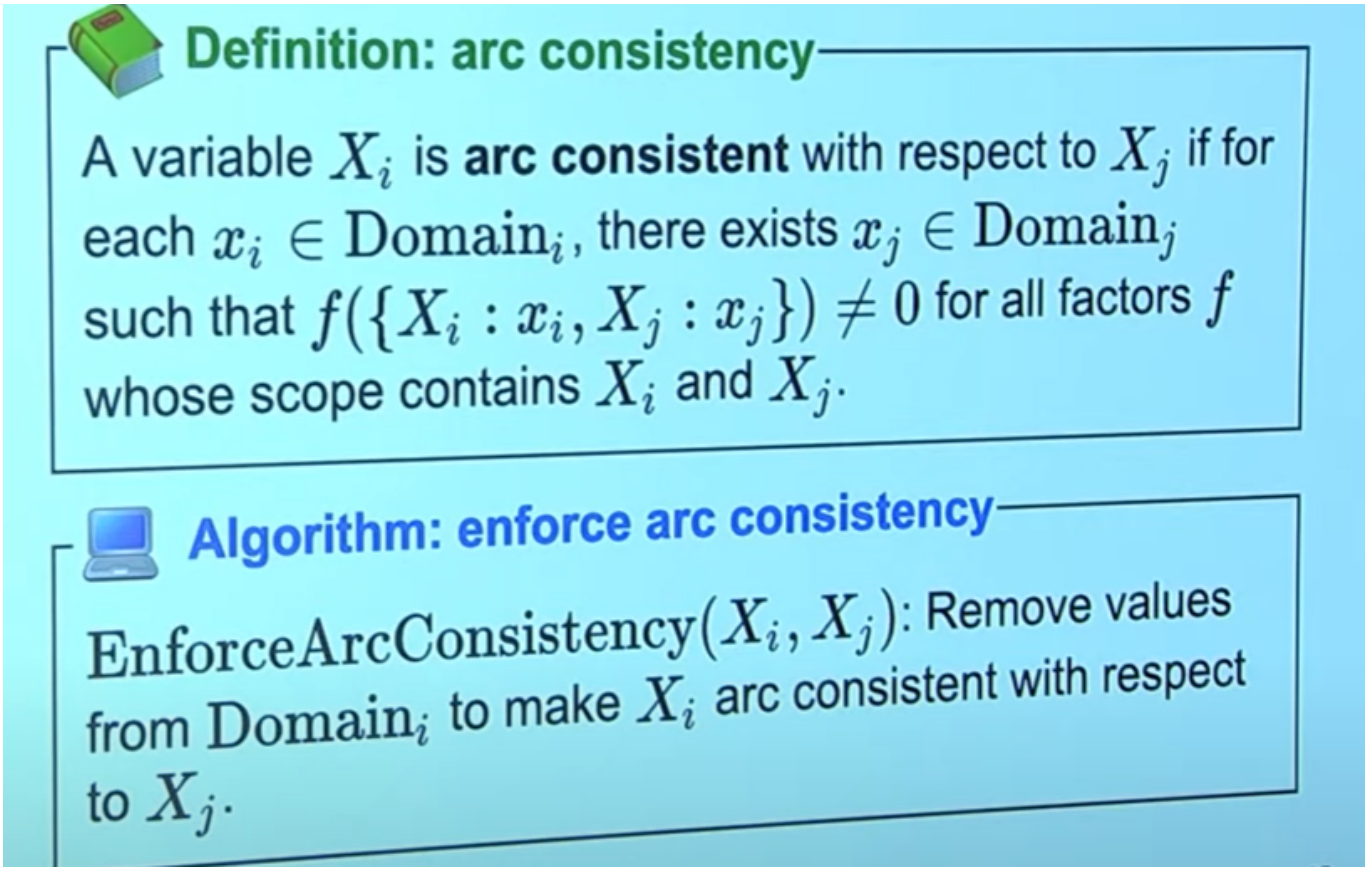
$X_i \in \text{Domain}_i = \{2, 3\}$

[whiteboard]

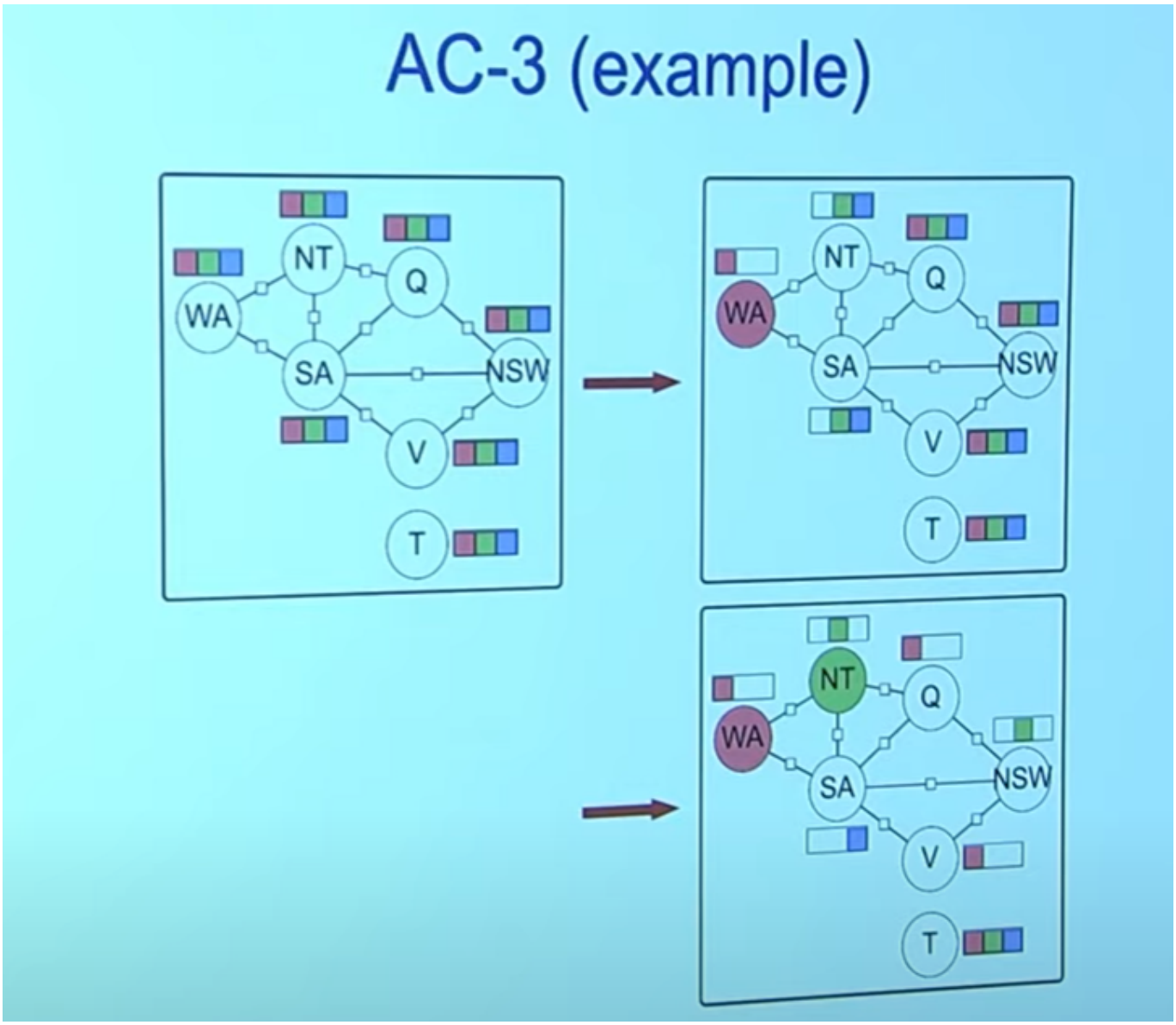


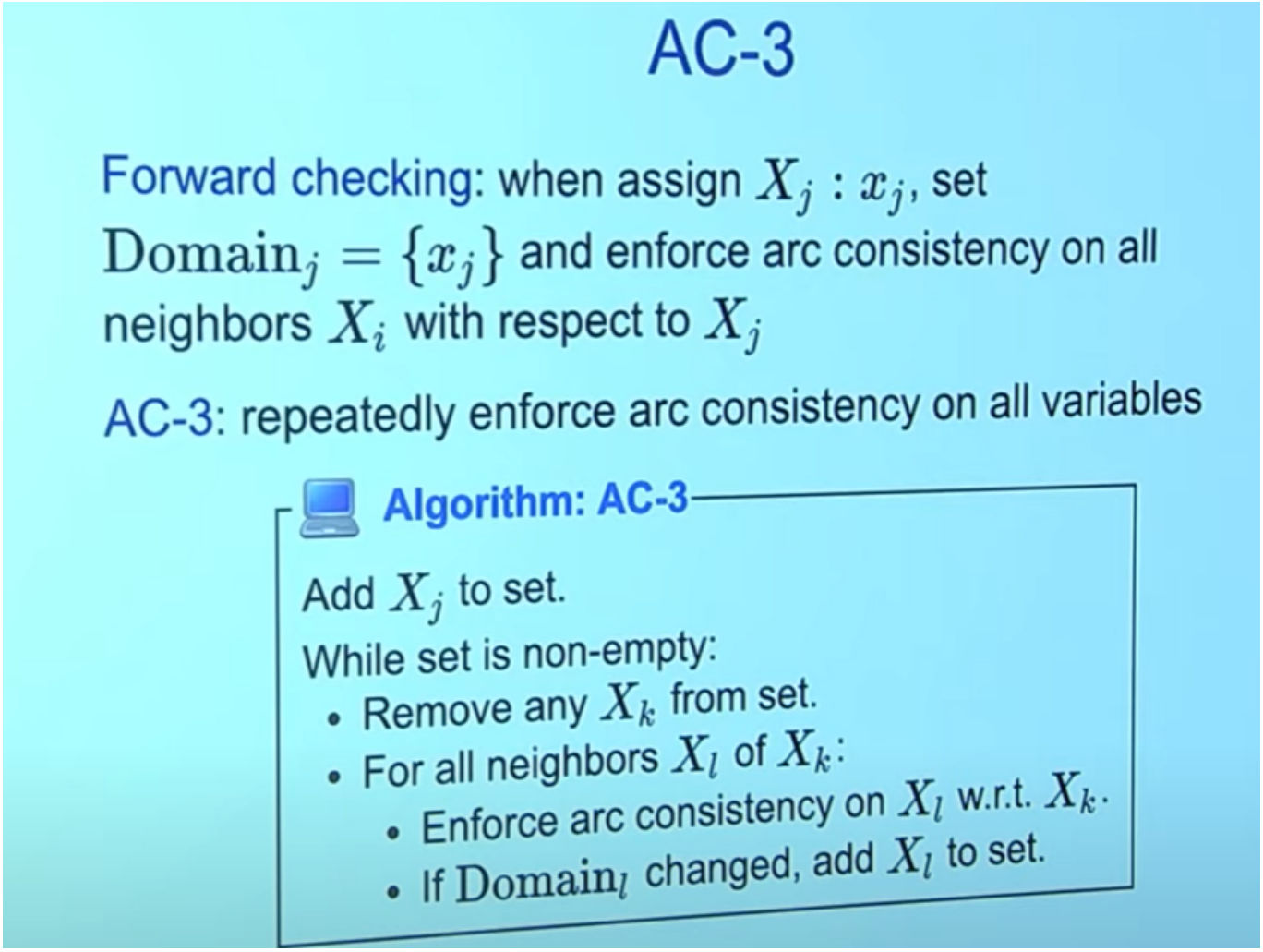


Basically it ensures everything should be consistant between Xi and Xj If it is inconsistent, remove it, and tries to make all factors are not euqal to 0



The only thing we are touching is Xi, to make it consistent with some other Xj


AC-3 (example)
NT
Q
WA
SA
NSW
V
T
NT
Q
WA
SA
NSW
V
T
NT
Q
WA
SA
NSW
V
T



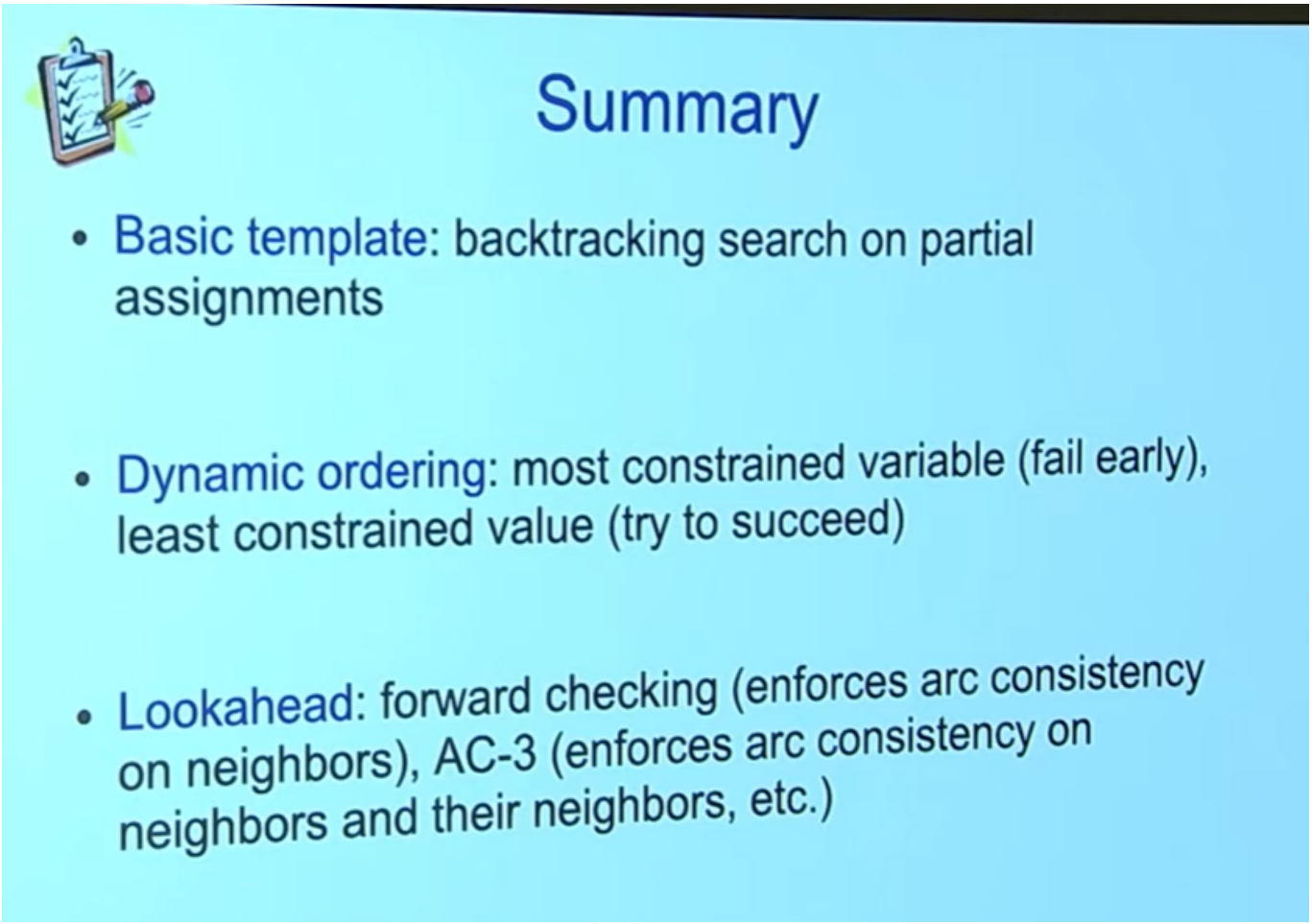
If the domain is changed, add the node back in Time complexity worst case scenorial O(ed^3)

Limitations of AC-3

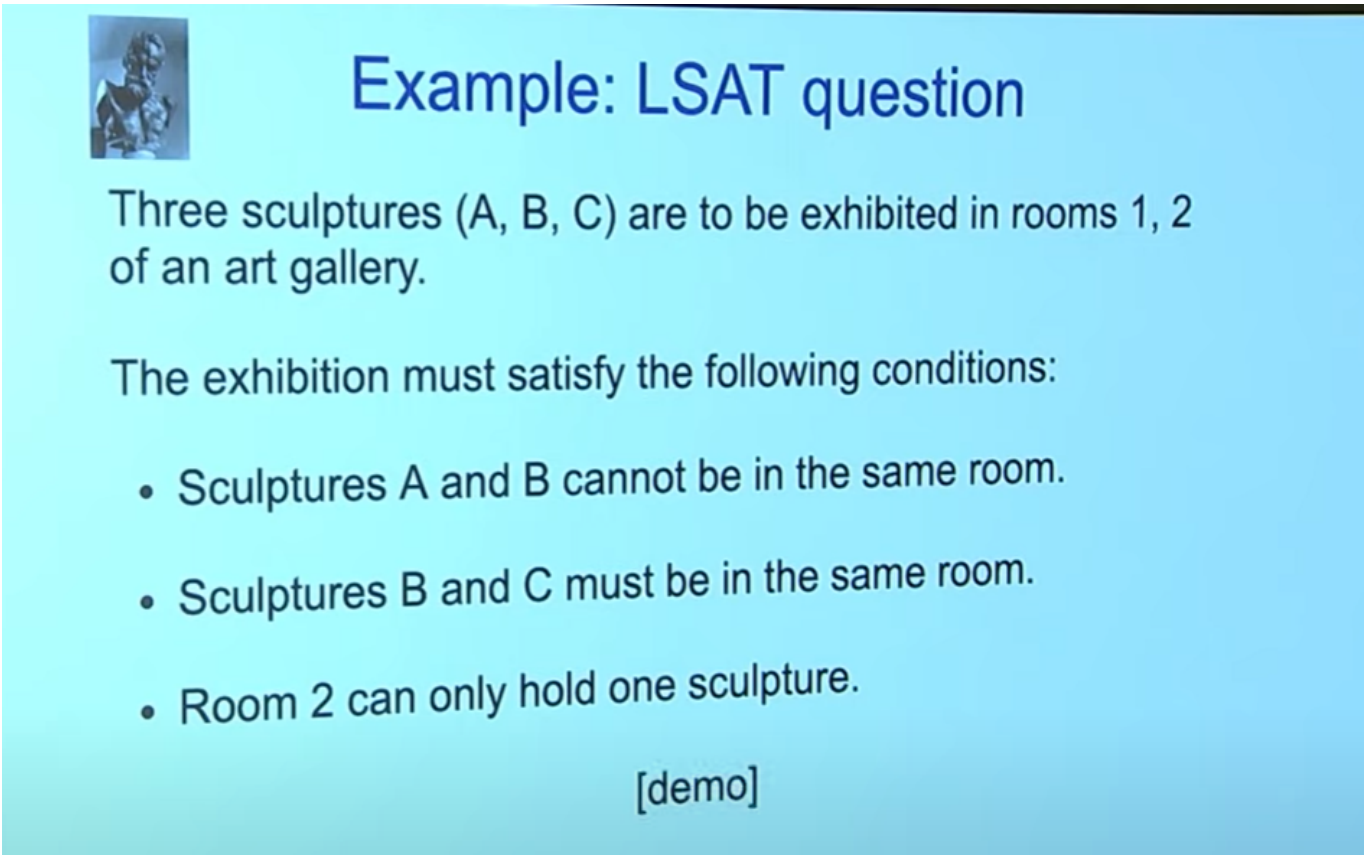
- Ideally, if no solutions, AC-3 would remove all values from a domain
- AC-3 isn't always effective:

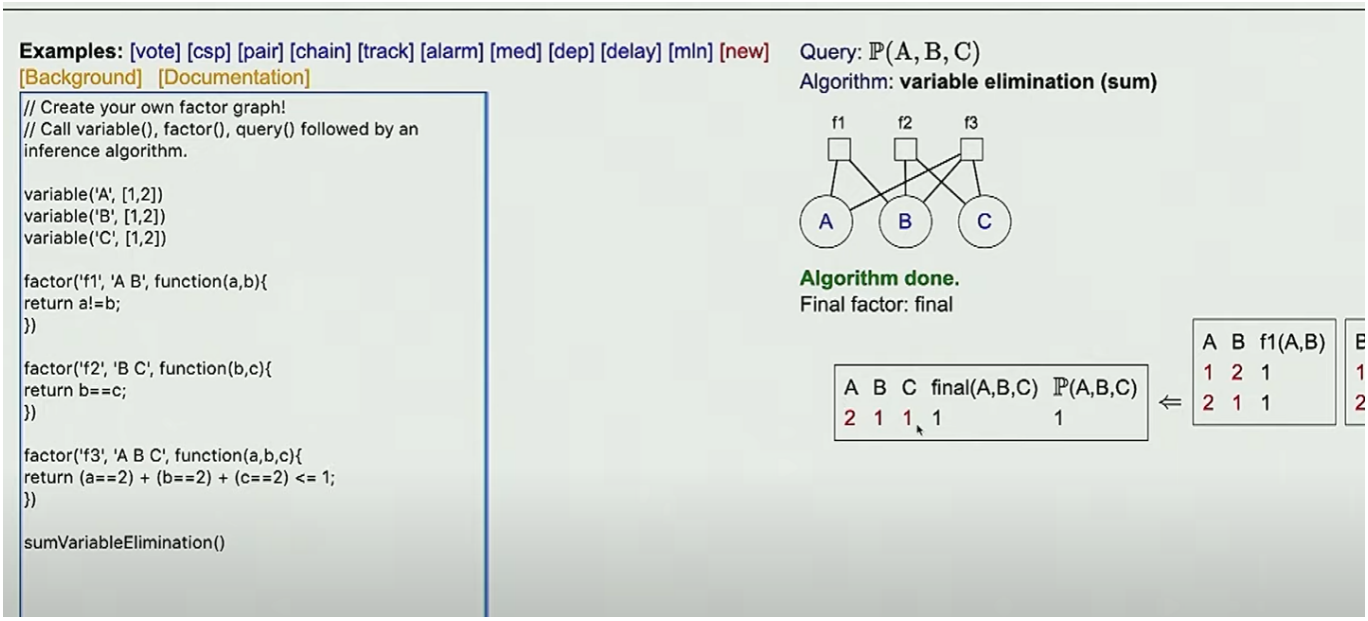
- No consistent assignments, but AC-3 doesn't detect a problem!
- Intuition: if we look locally at the graph, nothing blatantly wrong...

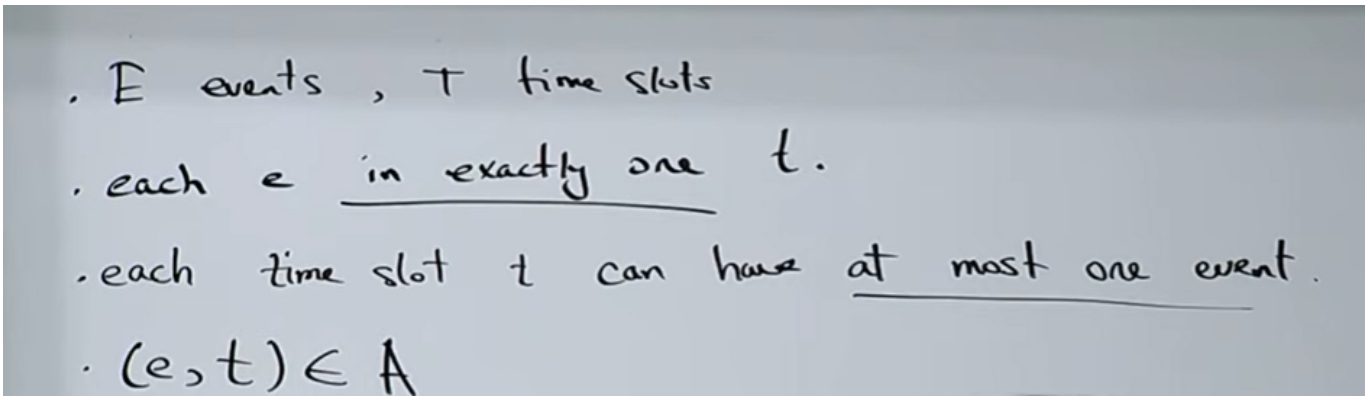
38


Summary
Basic template: backtracking search on partial assignments
Dynamic ordering: most constrained variable (fail early), least constrained value (try to succeed)
Lookahead: forward checking (enforces arc consistency on neighbors), AC-3 (enforces arc consistency on neighbors and their neighbors, etc.)

Modeling

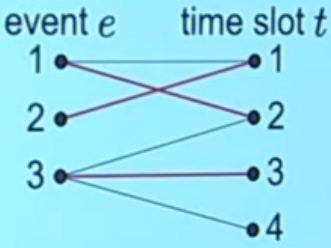

Example: LSAT question
Three sculptures (A, B, C) are to be exhibited in rooms 1, 2 of an art gallery.
The exhibition must satisfy the following conditions:
Sculptures A and B cannot be in the same room.
Sculptures B and C must be in the same room.
Room 2 can only hold one sculpture.
[demo]





Event and Timeslot should both belong some given set A

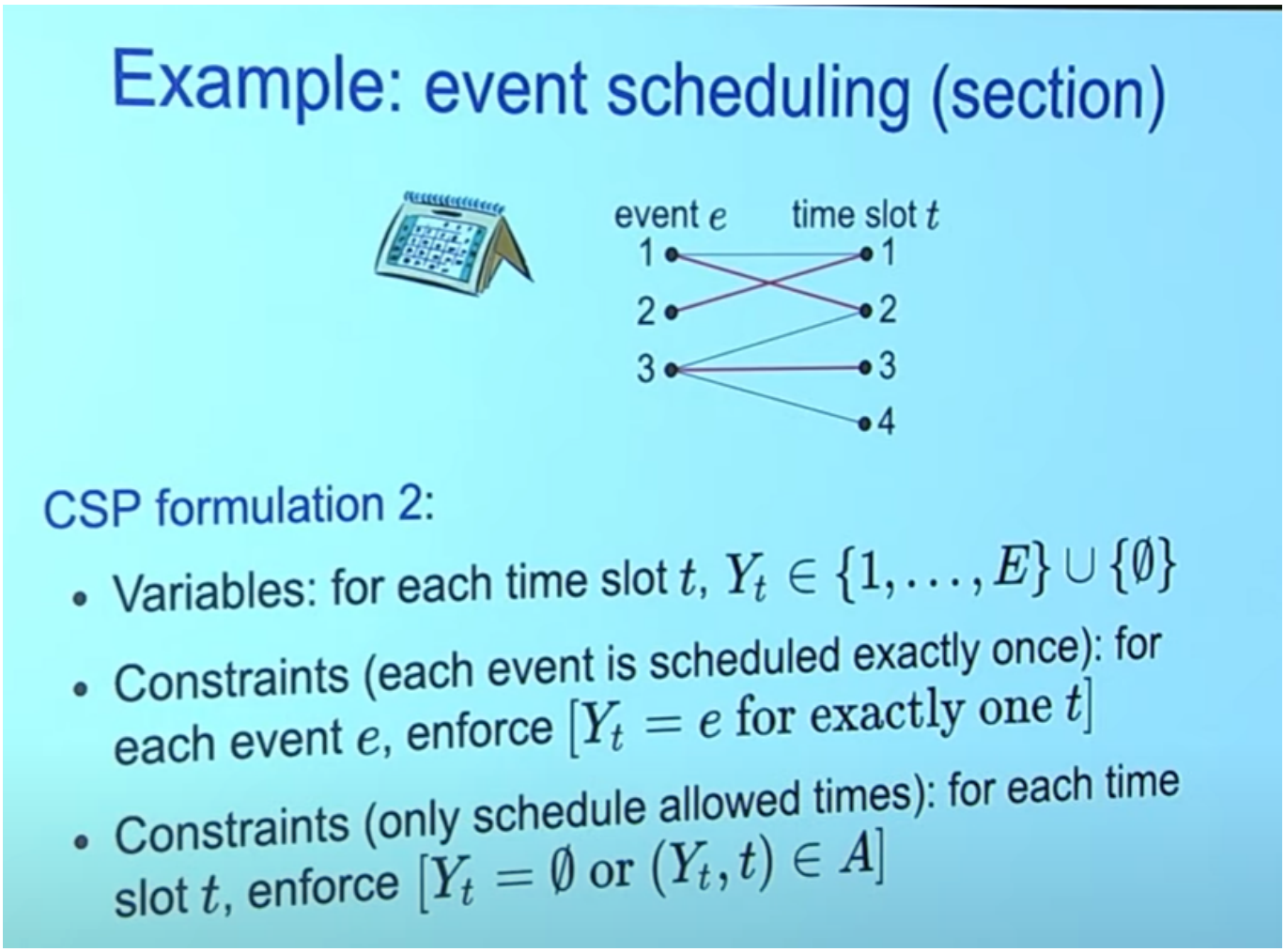
Example: event scheduling (section)



CSP formulation 1:

- Variables: for each event e, $X_e \in \{1, \ldots, T\}$
- Constraints (only one event per time slot): for each pair of events $e \neq e'$, enforce $[X_e \neq X_{e'}]$
- Constraints (only scheduled allowed times): for each event e, enforce $[(e, X_e) \in A]$

E as variable (we get constraint 1 for free) E number of variables of domain of size T E^2 binary factors (constraint 2) + E number of unary factors (constraint 3)



In this formulation: constraint 1: Each timeslot have an event or nothing constraint 2: each event have one unique t => all t have no duplicated e (think about sculpture example). This gonna be a t-ary factor

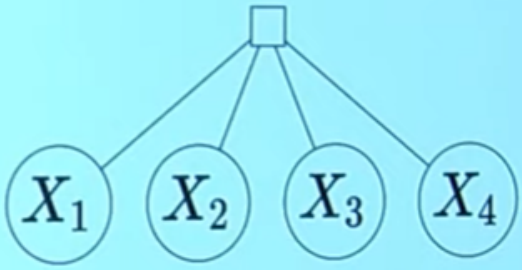
- So I have less factors but it is t-ary

Which one to choose? It really depends on the number of t or e, which one is greater.

If we have less event but more timeslots, formulation 1 is better (e^2 binary constraints). If we have less timeslot but more events, then formulation 2 is better ()

If we have n-ary constraint, we can transform it into binary constarint that are equivelent. The reason is usually our algothrim requires to have binary or unary constraints

N-ary constraints (section)



Variables: $X_1, X_2, X_3, X_4 \in \{0, 1\}$

Factor: $[X_1 \vee X_2 \vee X_3 \vee X_4]$

Examples:

$$\text{Weight}(\{X_1 : 0, X_2 : 0, X_3 : 0, X_4 : 0\}) = 0$$
$$\text{Weight}(\{X_1 : 0, X_2 : 1, X_3 : 0, X_4 : 0\}) = 1$$
$$\text{Weight}(\{X_1 : 0, X_2 : 1, X_3 : 0, X_4 : 1\}) = 1$$

N-ary constraints: attempt 1 (section)

Key idea: auxiliary variable

Auxiliary variables hold intermediate computation.

Factors:

Initialization: $[A_0 = 0]$

Processing: $[A_i = A_{i-1} \vee X_i]$

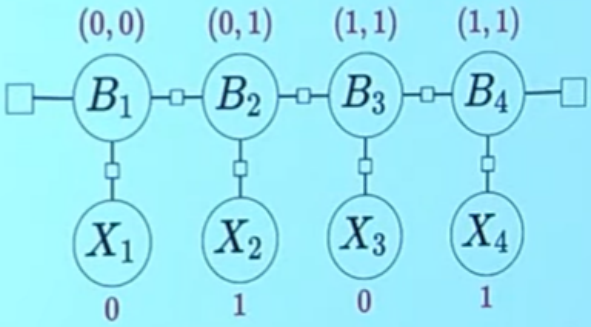
Final output: $[A_4 = 1]$

i	0	1	2	3	4
X_i:		0	1	0	1
A_i:	0	0	1	1	1

The idea is to define a new variable

N-ary constraints: attempt 2 (section)

Key idea: pack A_{i-1} and A_i into one variable B_i



Variables: B_i is (pre, post) pair from processing X_i

Factors:

- Initialization: $[B_1[1] = 0]$
- Processing: $[B_i[2] = B_i[1] \vee X_i]$
- Final output: $[B_4[2] = 1]$
- Consistency: $[B_{i-1}[2] = B_i[1]]$

47